

# 12<sup>th</sup> Inverted CERN School of Computing

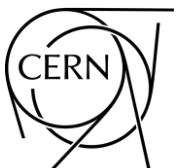
4 to 7 March 2019  
CERN, IT Amphitheatre (31/3-004)

*Live webcast, slides & recording at*  
<https://indico.cern.ch/e/iCSC-2019>

## Lecturers

Daniel Hugo CAMPORA PEREZ  
Patrick EMONTS  
Giorgio LOPEZ  
Evangelos MOTESNITSALIS  
Dmitry NEVEROV  
Riccardo POGGI  
Alexander RUEDE  
Mikhail SIZOV

CERN, and University of Seville, Spain  
Max Planck Institute of Quantum Optics, Germany  
CERN  
CERN  
Nagoya University, Japan  
University of Geneva, Switzerland  
CERN, and KIT-IPE, Germany  
Budker Institute of Nuclear Physics, Russia



Copyright © 2019 CERN and the CERN School of Computing.

iCSC 2019 booklet compiled and edited by Joelma Tolomeo and Nikos Kasioumis.

All content of the lecture materials contained herein are owned by the respective presentation authors.

Front page image: CERN computing facilities views from 2018 (IT-PHO-CCC-2018-001-23)  
© 2018 CERN, Photographer: Grossir Anthony - CERN



It is with great pleasure that I warmly welcome you all, lecturers and attendees alike, to this Inverted CERN School of Computing (iCSC) 2019. There are three reasons why I feel particularly happy and privileged to be writing these words.

First, this year we have a very rich program, based on proposals received from students of three past schools: CSC 2018 (Tel Aviv, Israel), Thematic CSC 2018 (Split, Croatia) and CSC 2017 (Madrid, Spain). It consists of a record 11 hours of lectures and 10 hours of hands-on exercises, covering a big range of topics, including Artificial Intelligence, Machine Learning, Pattern Recognition, Big Data, Container Orchestration, Tensor Networks, Computational Physics, Numerical Analysis, Track Finding, and more. During the exercises, you will have a chance to learn, hands-on, various technologies such as Keras and TensorFlow, FPGAs, Kubernetes, Hadoop, Apache Spark and others. I'm sure you will find the classes both relevant and very interesting!

Secondly, this is already the 12th edition of the Inverted CSC. The initial idea was to give the floor to former CSC students, so that they could share their knowledge and expertise with their colleagues. The fact that we've reached the 12th edition this year, and that we actually received more proposals for classes than we were able to accommodate, proves that this idea is still valid.

And finally, on a more personal note, the Inverted CSC remains close to my heart. I was one of the lecturers of its first edition back in 2005, giving a lecture about software security - even though I worked on something else at that time. I still remember how this experience had pushed me to develop further my knowledge and passion in security, and - consequently - heavily influenced my professional life. I hope this year's iCSC will have a similar positive impact on the careers of the lecturers!

I wish to thank and congratulate the lecturers for their significant work put in preparing the lectures; the mentors for their invaluable input and feedback; Joelma Tolomeo (the School's Administrative Manager) and Nikos Kasioumis (the Technical Manager) for the work behind the scenes; and finally, you - the attendees - for your interest and presence.

Please join me in enjoying this great learning and knowledge-sharing experience!



**Sebastian Łopieński**  
Director  
CERN School of Computing

Welcome	3
Contents	4
Core team	5
Biographies & lecture abstracts	6
Timetable	15
Slides	17
 <b>Daniel Hugo Campora Perez</b>	
A practical approach to Convolutional Neural Networks	18
 <b>Patrick Emonts</b>	
Tensor Networks: How physicists can tackle exponentially hard problems	36
 <b>Giorgio Lopez</b>	
Hardware Acceleration through FPGAs – an introduction	50
 <b>Evangelos Motesnitsalis</b>	
Big Data Technologies and Physics Analysis with Apache Spark	70
 <b>Dmitrii Neverov</b>	
Global track finding algorithms	89
 <b>Riccardo Poggi</b>	
How container orchestration can strengthen your micro-services: the approach of Kubernetes	106
 <b>Alexander Ruede</b>	
A Scientist's Guide to FPGAs	116
 <b>Mikhail Sizov</b>	
Efficient C++ implementation of custom FEM kernel with Eigen	122

---

**CORE TEAM**

---

**Director** **Sebastian ŁOPIEŃSKI**

**Administrative Manager** **Joelma TOLOMEO**

**Technical Manager** **Nikos KASIOUMIS**

**Mentors** **Thomas KECK**

**Sebastian ŁOPIEŃSKI**

**Lorenzo MONETA**

**Alberto PACE**

**Danilo PIPARO**

**Sebastien PONCE**

**Ivica PULJAK**

**Are STRANDLIE**

**Enric TEJEDOR**



The inverted School of Computing (iCSC) is part of the annual series of schools organized by the CERN School of Computing <http://cern.ch/csc>

---

## LECTURER BIOGRAPHIES

---

Every lecturer presenting here at the iCSC 2019 has been specially selected to advance their ideas, to further develop CSC-related themes, and to share and promote their knowledge. The Lecturers have been students of the past schools: CSC 2018 (Tel Aviv, Israel), Thematic CSC 2018 (Split, Croatia) and CSC 2017 (Madrid, Spain).

The lectures are broadcast via webcast and will be recorded. All slides, recordings, etc. can be found at <https://indico.cern.ch/e/iCSC-2019>

# Daniel Hugo Campora Perez

---



*CERN, and University of Seville, Spain*

Coming from Sunny Sevilla with a Computer Engineering degree and having spent some five years at CERN with various contracts, I'm currently a Doctoral Student, optimizing reconstruction algorithms in LHCb with an Artificial Intelligence twist.

I have many good questions, but I'm still searching for good answers!

Mentor(s)

**Thomas Keck, Lorenzo Moneta**

Lecture(s)

**A practical approach to Convolutional Neural Networks**

The field of Artificial Intelligence is blooming with many techniques and developments in the last years. Convolutional Neural Networks is a class of feed-forward neural networks that are typically used to analyze images and extract information.

We will explore the mathematical foundation of CNNs, join the hype and get some practical hands-on experience with the technology that will change our everyday lives.

# Patrick Emonts

---



*Max Planck Institute of Quantum Optics, Garching, Germany*

My name is Patrick Emonts and I am currently working on my PhD in theoretical physics at the Max Planck Institute of Quantum Optics in Munich.

My main focus is the development of tensor network algorithms to simulate lattice gauge theories.

In 2017, I graduated in physics at the RWTH Aachen about Monte Carlo simulation of complex solid state systems.

Working as a software engineer during my studies, I got some interesting insights into the field of optical character recognition.

In my free time, you will probably find me skiing or climbing.

Mentor(s)

**Danilo Piparo**

Lecture(s)

## **Tensor Networks: How physicists can tackle exponentially hard problems**

In recent years, tensor networks have become a viable alternative to Monte Carlo calculations and exact diagonalization for the simulation of many-body systems.

As they represent a formulation of quantum mechanical wavefunctions with polynomially many parameters, they make calculations of large systems feasible.

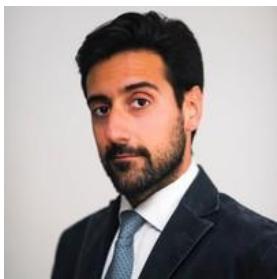
They have already found wide application in condensed matter physics and start to be an interesting tool for high energy physics as well.

In this lecture series, I will introduce the basic concepts of tensor networks.

We will start with an introduction of the necessary basics of quantum mechanics and linear algebra and focus on the algorithmic side of tensor networks in the second lecture.

# Giorgio Lopez

---



CERN

I was born in Napoli where I graduated in Computer Science Engineering and did my PhD in Electronics Engineering working on real time video processing on FPGAs.

Since 2015 I'm working at CERN in the Electric Power Converters group where my role is designing and developing digital logic for FPGAs in the controls of power converters.

Mentor(s)

**Ivica Puljak, Alberto Pace**

Lecture(s)

**Hardware Acceleration through FPGAs – an introduction**

FPGAs are a more and more ubiquitous technology. They offer the benefits of fast, application-tailored hardware, typically associated with ASICs, while enabling fast prototyping, upgradability and low costs. This makes them an ideal ally in HEP computing, specifically in areas where high performance is needed and/or specifications and needs may vary.

The lectures will focus on the intrinsic parallel processing characteristics of FPGAs, emphasizing how they can be exploited to implement data-intensive algorithms. Focus will also be put on concepts like hardware/software partitioning (very important to help the most performing parts of the systems in collaborating with the legacy CPU oriented codebase).

A simple hands on exercises session will be added to let the students get acquainted with the main tools and the VHDL language.

# Evangelos Motesnitsalis

---



CERN

I am a Technical Coordinator at the IT Department of CERN, working on utilization of public clouds for archiving purposes.

Before that, I worked as a Data Engineer at the Databases group where I supported the scientific communities in their quest to perform big data analytics over physics and accelerator data. I led the development of the “Hadoop-XRootD Connector” library, a project that provides direct access of data from XRootD-based storage systems directly into Hadoop and Spark.

I am also a former Escalation Engineer and Big Data Devops Support Engineer at Amazon Web Services in Dublin, Ireland. I obtained my MSc in Distributed Systems from Imperial College London in 2015 and prior to that, I studied at King’s College London and Aristotle University of Thessaloniki.

Mentor(s)

**Enric Tejedor, Sebastian Łopieński**

Lecture(s)

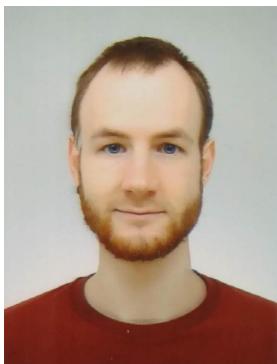
## **Big Data Technologies and Physics Analysis with Apache Spark**

The Large Hadron Collider is scheduled to shut down for a 2 years maintenance period since December 2018. However, the already collected data -which are stored in a dedicated custom storage service- between April 2015 and November 2018, exceed 150 PBs in total. To analyse these data, more and more teams at CERN decide to use Big Data Technologies to perform Physics Analysis and ”Data Reduction”, i.e. produce smaller reusable datasets for frequent access. These technologies show great potential in speeding up the existing procedures.

This lecture will provide an overview of the latest trending big data technologies in the Hadoop and Spark ecosystems with focus on their main architecture characteristics, and then will target a number of important questions: How can we perform Physics Analysis with Big Data Technologies? What are the problems faced? What are the challenges and the available data sources? What are the other domain in which Big Data Analytics are applied at CERN?

# Dmitry Neverov

---



*Nagoya University, Japan*

Born in the science town of snowy Novosibirsk of Siberia, Russia; After finishing school moved to Saint-Petersburg where I worked on phenomenology in pp collisions.

After graduation from master course got accepted to MEXT scholarship of Japan, and moved to Nagoya to work on Belle II experiment.

For my PhD I am developing analysis tools, primarily modified tracking, that are required for a search for low charge magnetic monopoles.

Mentor(s)

**Are Strandlie, Sebastien Ponce**

Lecture(s)

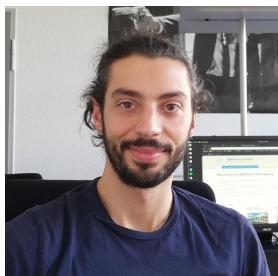
**Global track finding algorithms**

In high energy physics experiments the reconstruction of tracks of charged particles provides the core for the measurements of these particles' properties. Track finding algorithms can roughly be divided into two main categories: local and global. Local track finding algorithms try to link individual hits one by one while utilizing a variety of smart techniques to mitigate combinatorial complexity, whereas global track finding algorithms treat all hits simultaneously.

In this lecture we will look into track finding algorithms in wire chambers that are performed on all hits at once. The search is done by converting individual hit parameters to a curve in a dual space using Legendre or Hough transform, with the intersection of multiple curves corresponding to a track compatible with given hits. Then, the problem of finding a track is translated to the problem of finding most densely populated regions in the dual space which can be done effectively and quickly by a quadtree search.

# Riccardo Poggi

---



*University of Geneva, Switzerland*

Originally from Italy where I received my MSc in Physics, I am now pursuing a PhD with the University of Geneva. Being part of the ATLAS collaboration I have the opportunity to analyse LHC data and develop firmware for FPGAs with high bandwidth memory.

In the past I worked at CERN on the upgrade of the software infrastructure for the trigger and  
data acquisition system of the ATLAS experiment.

While outside CERN I worked for Accenture as a senior software architect specialised in Oracle Cloud.

I am looking forward to meeting you all once again!

Mentor(s)

**Enric Tejedor, Sebastian Łopieński**

Lecture(s)

**How container orchestration can strengthen your micro-services: the approach of Kubernetes**

With the rise of container technologies during the past few years there have been many paradigms shifts in terms of software development, deployment and maintenance, especially in conjunction with micro-service architectures.

The lecture covers these fundamental concepts and focuses on the challenges of container orchestration. Crucial aspects like horizontal and vertical scaling, availability, fault-tolerance and rolling updates are among the topics covered by the lecture which will then also be experienced during the hands-on exercises.

# Alexander Ruede

---



*CERN, and KIT-IPE, Germany*

I am an electronics engineering doctoral student at the Karlsruhe Institute of Technology and the CMS experiment, based at CERN.

My work focuses on the implementation of a variety of algorithms to measure luminosity in CMS for the high luminosity era of the LHC. Therefore I explore different approaches in software, FPGAs and in heterogeneous systems in order to meet the timing and precision requirements. I am a hardware and FPGA enthusiast!

Mentor(s)

**Ivica Puljak, Alberto Pace**

Lecture(s)

**A Scientist's Guide to FPGAs**

Field Programmable Gate Arrays (FPGAs) have become ubiquitous in a variety of technological and scientific fields. Their versatility make them an ideal match not only for computing intensive tasks but also for the differing requirements of custom electronics that often can be found in experimental setups.

This seminar leads the audience into the fully programmable and intrinsically parallel world of FPGAs. After an introduction to digital design and the anatomy of an FPGA, the design flow and required way of thinking will be presented. The seminar will be completed by a comparison of hardware and software-driven computation as well as an overview of the application of FPGAs in different fields and tasks.

# Mikhail Sizov



*Budker Institute of Nuclear Physics of the Siberian Branch of the RAS, Russia*

I obtained my Bachelor's degree with topic "Creating of the isolated environment for untrusted applications" and Master's topic was "Development of the automated data processing system for mobile muon densitometer".

Currently, I'm a PhD student at Budker Institute of Nuclear Physics (BINP) and my research is dedicated to practical use of domain specific languages in control systems.

I also teach programming at Geology and Physics Departments of Novosibirsk State University.

Mentor(s)

**Danilo Piparo, Sébastien Ponce**

Lecture(s)

**Efficient C++ implementation of custom FEM kernel with Eigen**

We will start from examples of problems solved by finite element method - equilibrium magnetic fields, structural deflection calculations. Then we will talk about foundation of FEM method key concepts such as stiffness matrix and impact of high matrix dimensions and sparse characteristic to ways data can be calculated more efficiently.

To implement kernel we will introduce Eigen, a C++ linear algebra library that eliminates intermediate temporary objects by utilizing expression templates technique and generates efficient high-level math code with most of complexity taken from you.

---

**TIMETABLE**

---

Monday, 4 March 2019		Tuesday, 5 March 2019		Wednesday, 6 March 2019		Thursday, 7 March 2019	
08:30	Welcome coffee	08:30	Welcome coffee	08:30	Welcome coffee	08:30	Welcome coffee
09:00	A word from the IT Department Head	09:00	Hardware Acceleration Through FPGAs - Basics of VHDL (lecture 2) - Giorgio Lopez	09:00	Tensor Networks - Singular Value Decomposition (exercise 1) - Patrick Emonts	09:00	Big Data Technologies and Physics Analysis with Apache Spark (exercise 1) - Vaggelis Motsatsis
09:10	Introduction to the Inverted CSC - Sebastian						
09:15	A practical approach to Convolutional Neural Networks (lecture)						
10:15	Coffee						
10:45	A Scientist's Guide to FPGAs - Alexander Ruede	11:00	Coffee	10:00	Tensor Networks - Application of the iTEBD Algorithm (exercise 2) - Patrick Emonts	10:00	Big Data Technologies and Physics Analysis with Apache Spark (exercise 2) - Vaggelis Motsatsis
11:45	Lunch break / WIT Diversity Talk	11:15	Tensor Networks - The iTEBD Algorithm (lecture 2) - Patrick Emonts	11:00	Coffee	11:15	How container orchestration can strengthen your micro-services: the approach of Kubernetes (lecture) - Riccardo Poggi
		12:15	Lunch break	12:15	Lunch break	13:00	Data Centre visit
13:30	Global track finding algorithms - Dmitri Neverov	13:30	Big Data Technologies and Physics Analysis with Apache Spark (lecture 1) - Vaggelis Motsatsis	14:30	Efficient C++ implementation of custom FEM kernel with Eigen - Mikhail Sizov	14:30	Efficient C++ implementation of custom FEM kernel with Eigen - Mikhail Sizov
14:30	Hardware Acceleration Through FPGAs - Basic Concepts (lecture 1) - Giorgio Lopez	14:30	Big Data Technologies and Physics Analysis with Apache Spark (lecture 2) - Vaggelis Motsatsis	15:30	Coffee	15:30	Coffee
15:30	Coffee			16:00	Hardware Acceleration Through FPGAs - First Experiments in VHDL (exercise 1) - Giorgio Lopez	16:00	How container orchestration can strengthen your micro-services: the approach of Kubernetes (exercise 1) - Riccardo Poggi
16:00	A practical approach to Convolutional Neural Networks (exercise 1) - Daniel Hugo Campora Perez			17:00	Hardware Acceleration Through FPGAs - Easy DSP Applications (exercise 2) - Giorgio Lopez	17:00	How container orchestration can strengthen your micro-services: the approach of Kubernetes (exercise 2) - Riccardo Poggi
17:00	A practical approach to Convolutional Neural Networks (exercise 2) - Daniel Hugo Campora Perez						

 School opening (room 31-3-004), Visit Data Centre (room 513-1-021)  
 Lectures in the IT Amphiitheatre (room 31-3-004)  
 Exercises in room 513-1-024  
 Coffee, Lunch breaks

---

**SLIDES**

---

## Outline

# A practical approach to Convolutional Neural Networks

Daniel Hugo Cámpora Pérez  
inverted CERN School of Computing, Mar 5th - 7th, 2019

Universidad de Sevilla  
CERN



1

Introduction

Some key ANN concepts

Convolutional neural networks

Overview of historically important networks

Cifar-10 example

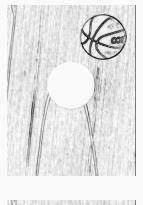
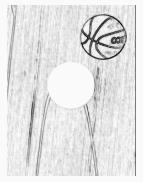
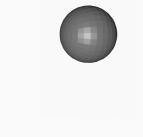
RICH reconstruction example

Bibliography

2

## Some concepts and history

Stages of Visual Representation, David Marr, 1970s

	Input image		Edge image		2½D model		3D model
Perceived intensities	Zero crossings, blobs, edges, bars, ends, virtual lines, groups, curves, boundaries	Local surface orientation and discontinuities in depth and in surface orientation	3D models hierarchically organized in terms of surface and volumetric primitives				

3

Semantic segmentation

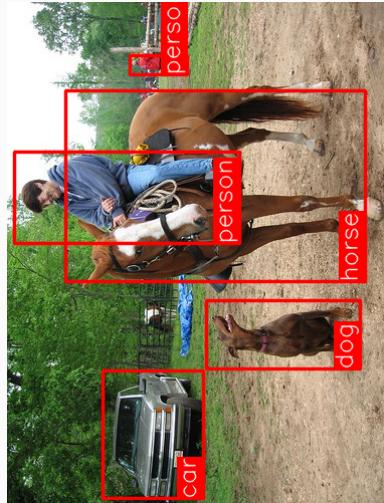
In semantic segmentation, our goal is to classify the different objects in the image, and identify their boundaries.



4

Object detection

In object detection, the objective is to find and classify the objects in an image.



5

Face detection

Face Detection, Viola & Jones, 2001



Object recognition

Object Recognition from Local Scale-Invariant Features, David Lowe,  
1999



6

३

Having 3D representations of objects, said objects are recognized in a variety of scenarios, including scenarios with occlusion.



7

## PASCAL Visual Object Challenge (VOC)



Object from 20 categories. Success is measured by how many were correctly classified.



8

## ImageNet challenge



The ImageNet challenge was created in 2009. Similarly to the VOC, images and categories are presented. However, 1.4M images with 1k object classes are presented. A classification is successful if the sought category is in the first 5 categories outputted by the algorithm.

For instance, the following classification would be correct:

- Scale
- T-shirt
- Steel drum
- Drumstick
- Mud turtle



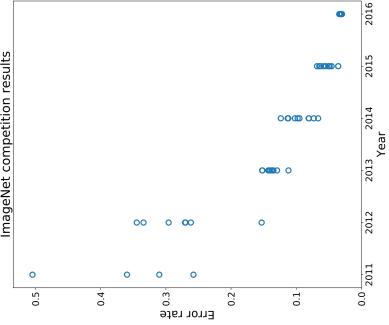
9

## ImageNet challenge (2)



In 2012, the Convolutional Neural Network named AlexNet got 10% less error rate than any of its competitors.

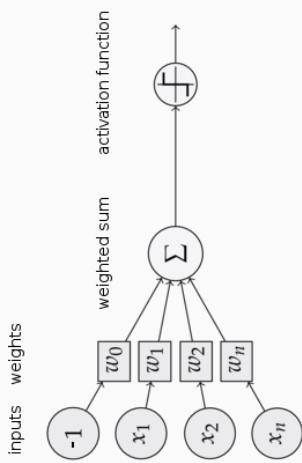
Recently (2015), the ImageNet challenge has been solved with an error lower to that of humans!



10

## The perception

You can think of a perceptron as a mathematical model, inspired in a single neuron.



11

## The perceptron (2)

It consists of:

- Inputs  $-1, x_1, x_2, \dots, x_n$
- Weights  $w_0, w_1, w_2, \dots, w_n$
- Activation function  $f(x)$

With these ingredients, the output of the perceptron is calculated as follows,

$$y = f\left(\sum_{j=0}^n w_j x_j\right) \quad (1)$$

12

## Activation functions

The activation function  $f(x)$  defines the relation between the states of the neighbouring neurons and the state of the neuron under study. It must be chosen according to the nature of the state of the neuron.

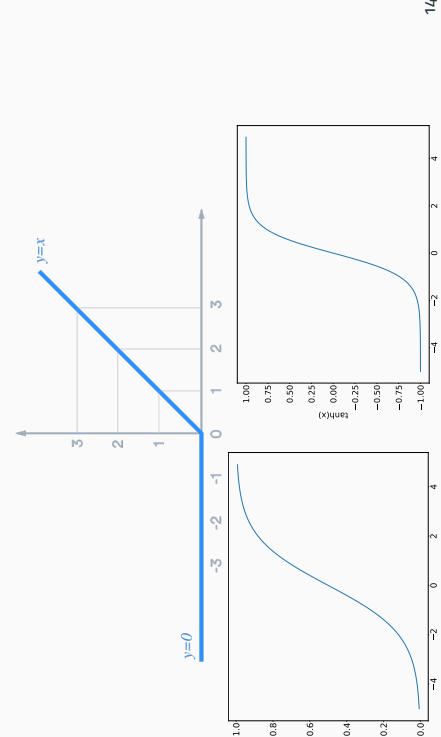
Here are some typical activation functions:

- $R(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$
- $\sigma(x) = \frac{1}{1 + e^{-x}}$
- $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

21

## Visually

The Rectified Linear Unit function is widely used, as it is very cheap to compute. Sigmoid ( $\sigma$ ) and hyperbolic tangent ( $tanh$ ) are also pretty common.



13

14

## Differentiability

It's useful to have an activation function that is differentiable to let our network learn with the backpropagation algorithm. ReLU ( $R$ ), the sigmoid ( $\sigma$ ) and hyperbolic tangent ( $\tanh$ ) are indeed differentiable,

$$R'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \quad (2)$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \quad (3)$$

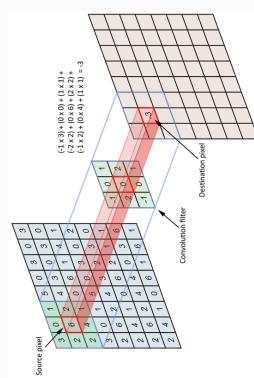
$$\tanh'(x) = 1 - \tanh^2(x) \quad (4)$$

15

## What is a convolution?

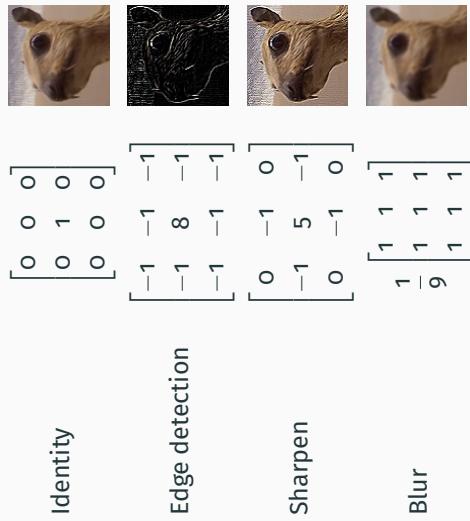
An image convolution is a transformation pixel by pixel, done by applying to an image some transformation defined by a set of weights, also known as a **filter**. Let  $s$  be a set of source pixels, and  $w$  a set of weights, a pixel  $y$  is transformed as follows:

$$y = \sum_{i=0}^n s_i w_i \quad (5)$$

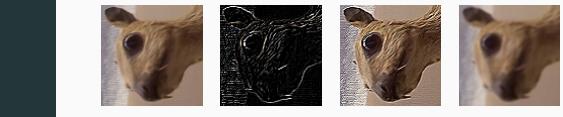


## iSCC

## Examples of filters



## iSCC



## iSCC

## iSCC

An image convolution is a transformation pixel by pixel, done by applying to an image some transformation defined by a set of weights, also known as a **filter**. Let  $s$  be a set of source pixels, and  $w$  a set of weights, a pixel  $y$  is transformed as follows:

$$y = \sum_{i=0}^n s_i w_i \quad (5)$$

16

## Architecture overview



Regular neural networks don't scale well to full images. To see this, let's have a look at a subset of the CIFAR-10 dataset:



Figure 1: A subset of the CIFAR-10 dataset.  
(<https://www.cs.toronto.edu/~kriz/cifar.html>)

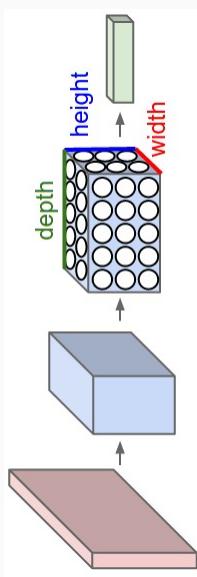
This dataset is composed of images of size  $32 \times 32 \times 3$  (width  $\times$  height  $\times$  color channels), categorized to what they contain. Note even though these are 2D images, due to the color channels, we are dealing with volumes.

18

## Architecture overview (2)



If we have a perceptron connect to all the inputs, this would imply  $32 \times 32 \times 3 = 3072$  weights. This number quickly runs out of hand when scaling to larger images.



Usually, Convolutional Neural Networks deal with this problem by using a feedforward network, and having **local connectivity** between the layers, that is, we will connect each neuron to only a local region of the input volume.

19

## Types of layers



We will look at three basic kinds of layers:

- Convolutional layers
- Pooling layers
- Fully-connected layers

To describe these layers we will refer to their connectivity and three dimensions: width, height and depth.

- A weight *activation map* determines the impact of the filter
- Weights can be shared across activation functions to reduce the number of learnable parameters
- Depth (or channels) conceptually will learn different *features*
- Translation equivariant: If the input changes, the output changes in the same way

20



## Convolutional layers

A convolutional layer consists of a set of learnable filters. Every filter is spatially small in width and height, but extends through the full depth of the input volume.

- Learnable filters are applied to **local regions**

21

## Convolutional layer examples



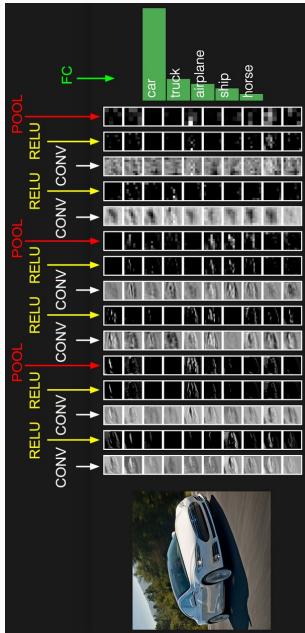
- Example 1 (<http://cs231n.github.io/assets/conv-demo/index.html>)
- Example 2 ([https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic))
- Example 3 (<https://thomaskeck.github.io/talks/DeepLearning.html#/2/6>)

22

## Learnt filters



- Intuitively, the network will learn filters that activate when they see some type of **visual feature**, like an edge or a blotch of some color on the first layer, or eventually honeycomb or wheel-like patterns on higher layers of the network.



23

## Convolutional layers spatial arrangement



Three parameters control the size of the output volume:

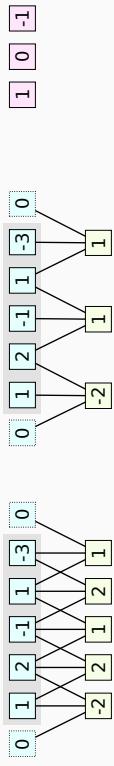
- **Depth** Number of filters we would like to use, each searching for a different characteristic.
- **Stride** Determines how the filter is slid across the input. It is uncommon to slide more than three pixels at a time.
- **Zero-padding** Determines the number of pixels filled with zeroes around the border. Sometimes, this is useful to allow certain strides.

We can compute the spatial size of the output volume as a function of the input volume size  $W$ , the filter size  $F$ , stride  $S$  and padding  $P$  as:

$$\frac{W - F + 2P}{S} + 1 \quad (6)$$

24

## Example: Convolutional layers spatial arrangement



$$\frac{5 - 3 + 2 \cdot 1}{1} + 1 = 5 \quad (7)$$

$$\frac{5 - 3 + 2 \cdot 1}{2} + 1 = 3, \quad (8)$$

25

## Convolutional layers: Parameters

For each convolutional layer in our network, there will be associated weights or parameters to each of the neurons in the layer. For a layer  $l - 1$  connected to layer  $l$ , with depths  $D_{l-1}$  and  $D_l$  respectively, and filter size  $F$ , the number of parameters required for a convolutional layer is:

$$\text{parameters} = \text{connection weights} + \text{bias weights}$$
$$= F \times F \times D_{l-1} \times D_l + D_l$$

26

## Convolutional layers parameter sharing

Following the assumption that if one feature is useful to compute at some spatial position  $(x_0, y_0)$ , then it should also be useful to compute at a different position  $(x_1, y_1)$ , it is possible to reduce dramatically the number of parameters (weights) in a CNN.

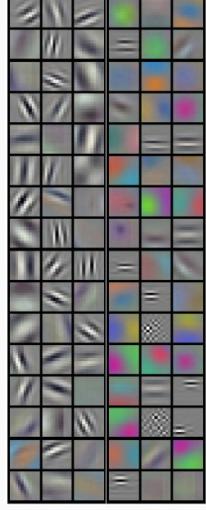


Figure 2: Example filters learnt by Krizhevsky et al. [3], who won the ImageNet challenge (<http://www.image-net.org/challenges/LSVRC>) in 2012. The first convolutional layer had a size of  $55 * 55 * 96 = 290\,400$  neurons with  $11 * 11 * 3 + 1 = 364$  inputs each. This makes for  $105\,705\,600$  weights, only feasible due to parameter sharing.

27

## Pooling layers

The function of a pooling layer, also known as a *subsampling* layer, is to progressively reduce the spatial size of the representation, to reduce the amount of parameters and computation in the network. The pooling is typically done using the *average* or *maximum* function, applied to the subset in consideration.

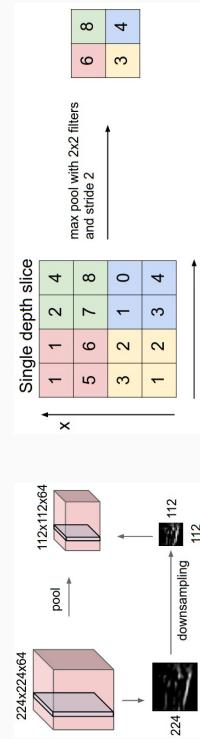


Figure 3: Pooling using the maximum function, with  $2 \times 2$  filters and stride 2.

28

## Other types of layers

- **Normalization layers** Many types of normalization layers have been proposed, for instance with the intention of implementing inhibition schemes observed in the biological brain. BatchNorm is a popular one.

- **Fully-Connected layers** Neurons in a fully-connected layer have full connections to all activations in the previous layer, as seen in regular neural networks. They are used in the output layers of CNNs.

- **Dropout layers** Dropout is a regularization technique to reduce overfitting. It consists in randomly connecting only a subset of neurons in backpropagation steps. Example (<https://thomaskeck.github.io/talks/DeepLearning.html#/2/10>)

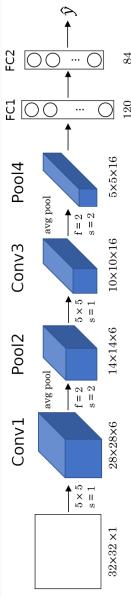
29

## Overview of historically important networks

The Modified National Institute of Standards and Technology (MNIST) dataset consists in a training set of 60000 images of handwritten digits, ranging from 0 to 9, and a separate test set of 10000 handwritten digits. It is therefore a classification problem, consisting in classifying the input  $28 \times 28 \times 1$  images into one of 10 classes.



30



Conv1:

- Input volume size  $W = 32$
- Filter size  $F = 5$
- Padding  $P = 0$
- Stride  $S = 1$
- $\frac{W - F + 2P}{S} + 1 = \frac{32 - 5 + 2 \cdot 0}{1} + 1 = 28$
- Depth (number of filters)  $D = 6$

Hence, the size of Conv1 is  $28 \times 28 \times 6$

31



Some observations:

- The network consists of two convolutions, two pooling layers, and two fully connected layers
- Both the sigmoid ( $\sigma$ ) and hyperbolic tangent ( $\tanh$ ) were used as activation functions
- Average pooling is used

32

32

## LeNet - 5, breakdown of layers (2)

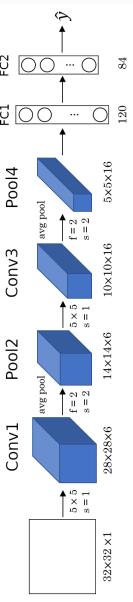


- Pool2:
- Input volume size 28
  - Filter size 2
  - Stride 2
  - For every  $2 \times 2$  pixels, we obtain a single one, effectively dividing the volume size by 2 on each axis

Hence, the size of Pool2 is  $14 \times 14 \times 6$

33

## LeNet - 5, breakdown of layers (3)



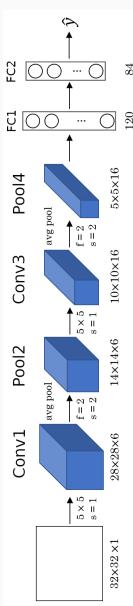
Conv3:

- Input volume size  $W = 14$
- Filter size  $F = 5$
- Padding  $P = 0$
- Stride  $S = 1$
- $\frac{W - F + 2P}{S} + 1 = \frac{14 - 5 + 2 \cdot 0}{1} + 1 = 10$
- Depth (number of filters)  $D = 16$

The size of Conv3 is  $10 \times 10 \times 16$

34

## LeNet - 5, breakdown of layers (4)



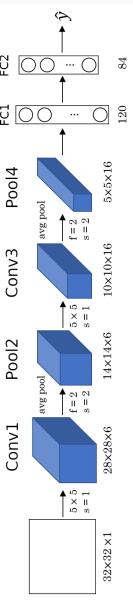
Pool4:

- Input volume size 10
- Filter size 2
- Stride 2
- For every  $2 \times 2$  pixels, we obtain a single one, effectively dividing the volume size by 2 on each axis

The size of Pool4 is  $5 \times 5 \times 6$

35

## LeNet - 5, breakdown of layers (5)



FC1:

- Input size is  $5 \times 5 \times 16 = 400$
- Output size is 120

Therefore, the fully connected layer connects all 400 inputs to 120 outputs.

36

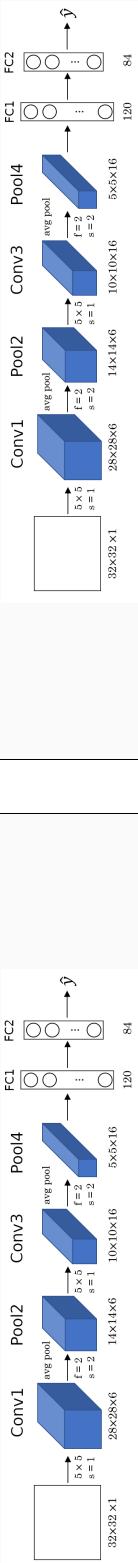
## LeNet - 5, breakdown of layers (6)



- FC2:
- Input size is 120
  - Output size is 84
- The fully connected layer connects all 120 inputs to 84 outputs.

37

## LeNet - 5, breakdown of layers (7)



FC2:

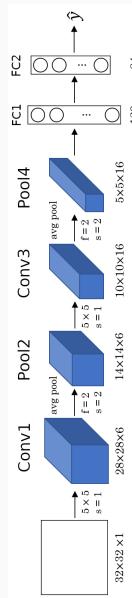
- Input size is 84
- Output size is one element per possible classification value, therefore 10

Finally, the output layer is effectively another FC layer.

28

38

## LeNet - 5, learnable weights



- How many weights do we require?

$$\begin{aligned}
 \text{parameters} &= \text{connection weights} + \text{bias weights} \\
 &= F \times F \times D_{l-1} \times D_l + D_l \\
 &= 5 \times 5 \times 1 \times 6 + 6 = 156
 \end{aligned}$$

Given we have  $28 \times 28$  output volumes, the number of learnable parameters of Conv1 is  $28 \times 28 \times 156 = 122304$

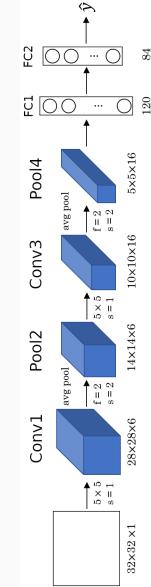
39

40

## LeNet - 5, learnable weights



## The ImageNet challenge



41

- In total, we have about 340 000 parameters in LeNet-5. Not a lot for today's standards, but definitely impressive for 1998.

As we introduced before, the ImageNet challenge presents about  $1 \cdot 10^6$  images of 1000 classes. An answer is the five most probable classes from the output of the algorithm, and if the correct one is contained within those five, the answer is correct.

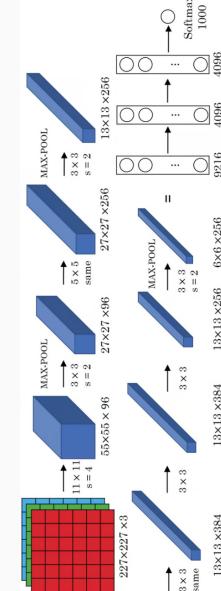


42

## AlexNet

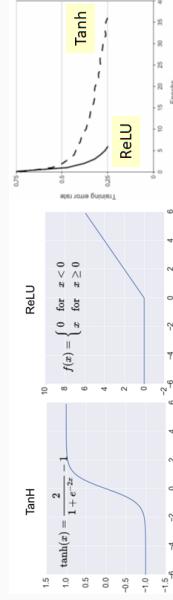


AlexNet is a CNN introduced in 2012 by Krizhevsky et al. [3] that won the ImageNet challenge by a margin of more than 10%. The network topology of Alexnet is as follows:



43

- The Rectified Linear Unit function is introduced. It is claimed to converge up to 6 $\times$  faster than the sigmoid or hyperbolic tangent:

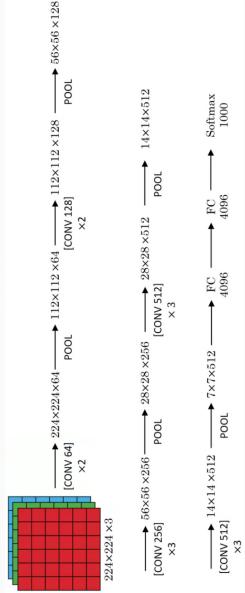


- Similar topology to LeNet, but bigger
- Max pooling is used
- It uses padding to keep same size
- About 60M parameters
- Multiple GPUs are used in learning, and so layers are separated in two at every step

44

VGG-16

Recently, many versions of deep neural networks have made an appearance in literature. One such example is VGG, from Simonyan and Zisserman [4]. Various versions of this neural network exist, the one below is VGG-16:



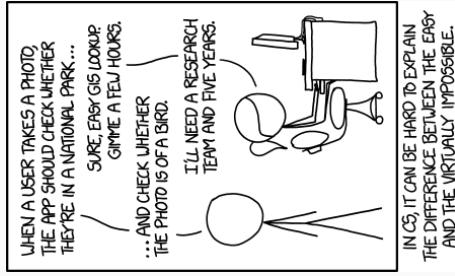
45

VGG-16 - Remarks

- Note how complex designs are referred to as modular blocks:
    - $[CONV\ 64] \times 2$  refers to two convolutional nets with 64 channels (depth 64)
      - POOL - Max pooling is assumed, if not specified otherwise
      - ReLU is assumed, if not specified otherwise
    - CONV is always a  $3 \times 3$  filter, with stride  $S = 1$ , and padding  $P = 1$  (*same*)
    - POOL is always a  $2 \times 2$  MAX-POOL, with stride 2
    - About 138M parameters
  - The design follows a pattern: Every successive CONV layer doubles in number of channels (*depth*), and halves in width and height

46

## Example: Classifying images



THE DIFFERENCE BETWEEN THE EASY  
AND THE VIRTUALLY IMPOSSIBLE.

Daniel Hugo Campdora Perez

Cifar-10 example

## Well, let's actually do it

Our input will be the CIFAR-10 data set, which contains a training set of 50 000 images of 10 different classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck). Additionally, it has 10 000 extra images, which can be used for the validation and test sets.



48

## What to expect in these images

There is no *standard of photography* in these images. Some may be taken up close, others relatively far from the object. The object we want to identify may be rotated, deformed or cropped. Light conditions may vary...

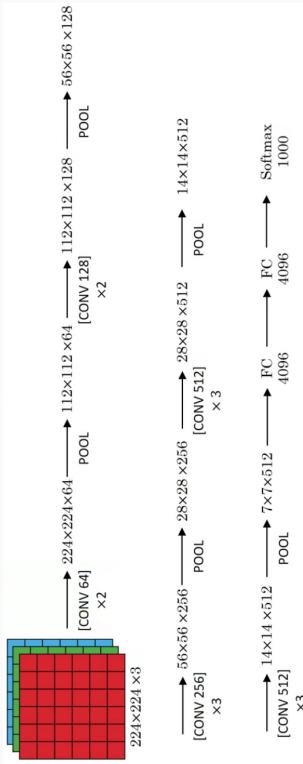
Given all the variations in conditions, instead of just taking the images we will use a *generator* to simulate them, taking as a basis the elements in our training set.

49

## Taking inspiration

We can draw inspiration in an existing topology configuration, like that one from VGG-16.

- No need to reinvent the wheel
- As we will see later, it is even possible to reuse learnt models



50

## Our convolutional neural network

Here is a proposed simple design,

- Input is 32x32x3
- CONV 32
- POOL
- CONV 64
- POOL
- FC 256
- Output

31

51

## Printing the topology of the network

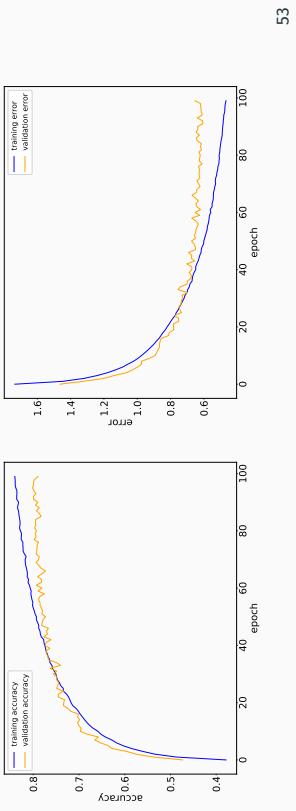
```

Layer (type)          Output Shape         Param #
=====
conv2d_1 (Conv2D)     (None, 32, 32, 32)      896
activation_1 (Activation) (None, 32, 32, 32)      0
max_pooling2d_1 (MaxPooling2D) (None, 16, 16, 32)    0
conv2d_2 (Conv2D)     (None, 16, 16, 64)      18496
activation_2 (Activation) (None, 16, 16, 64)      0
max_pooling2d_2 (MaxPooling2D) (None, 8, 8, 64)      0
flatten_1 (Flatten)   (None, 4096)            0
dense_1 (Dense)       (None, 256)             1048832
activation_3 (Activation) (None, 256)            0
dense_2 (Dense)       (None, 10)              2570
activation_4 (Activation) (None, 10)             0
=====
Total params: 1,070,794
Trainable params: 1,070,794
Non-trainable params: 0
  
```

52

## Checking the efficiency of our network

- Finally, let's check how we did. Depending on how we separated our data set into a training set and a validation set, results may vary.
- Epochs: 100
  - Test loss: 0.6520
  - Test accuracy: 0.7893

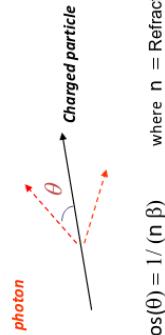


53

## iQSC

### Problem formulation

Cherenkov radiation principle



### RICH reconstruction example

$$\cos(\theta) = 1 / (n \beta) \quad \text{where } n = \text{Refractive Index} = c/c_M = n(E_{ph})$$

$$\beta = v/c = p/E = p/\sqrt{p^2 + m^2} = 1/\sqrt{1 + (m/p)^2}$$

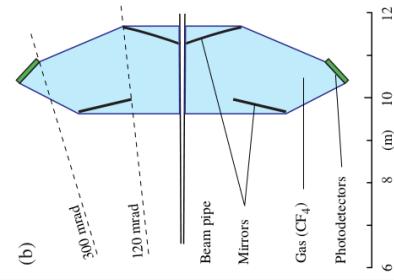
$\beta$ = velocity of the charged particle in units of speed of light (c)  
 $p, E, m$  = momentum, Energy, mass of the charged particle.  
 $c_M$  = Speed of light in the Medium (Phase velocity),  
 $E_{ph}$  = Photon Energy,  $\lambda$ =Photon Wavelength.

➤ Theory of Cherenkov Radiation: Classical Electrodynamics by J.D.Jackson ( Section 13.5 )

## RICH detectors in LHCb



In LHCb, we have two RICH detectors. Below is a schematic XZ view of RICH1:



55

## Analytical solution



The analytical solution consists in *creating photons*, ie. associations of detected pixels in the HPDs / MapMTs with their originating track segments through the Rich detector.

Once this association is found, a likelihood minimisation algorithm is run in order to find the most likely candidate for each particle.

- Photon creation, heavily involving ray tracing
- Likelihood minimisation

56

## Analytical solution - Ray tracing



The creation of the photons involves a ray tracing algorithm from each segment to the candidate pixels. In turn, this means solving the quartic equation:

$$4e^2 d_y^2 \sin^4 \beta - 4e^2 d_y R \sin^3 \beta + (d_y^2 R^2 + (e + d_x)^2 R^2 - 4e^2 d_y^2) \sin^2 \beta + 2e d_y (e - d_x) R \sin \beta + (e^2 - R^2) d_y^2 = 0. \quad (3)$$

This can be solved, using for example a routine in the CERN library [6], and gives four solutions for  $\sin \beta$ , two complex and two real. Of the real solutions one is the "backward" reflection (that would exist if the mirror were a complete sphere, shown as M' in Fig. 3); the other is the desired solution, and can be selected from knowledge of the RICH detector geometry. The value of  $\cos \beta$  can then be extracted using Eq. 2, and the coordinates of the reflection point M determined.

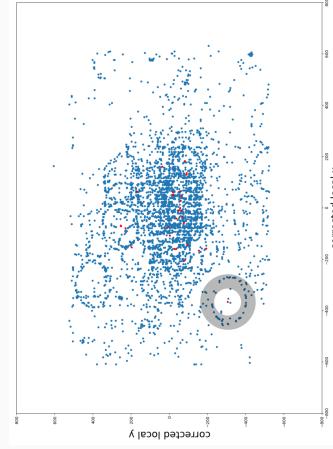
57

## Bringing ML into the fray



Alternatively, we are studying whether it is possible to transform the problem into a classification problem from an image into a particle ID (one of *pion, muon, electron, kaon, proton, deuteron*):

- Each track is extrapolated onto the detector plane (red dots)
- A corona shape is fed onto a Convolutional Neural Network to identify the particle



58

## Motivation

If we observe the surrounding area of a single track extrapolation and convert it to polar, we end up with the figures below:



Figure 4: Left: Pion. Right: Electron.

## Exploring possibilities

Some CNNs are more amenable to solving the problem at hand. In fact, the problem looks very similar to the classical MNIST problem. One such CNN is:

Layer (type)	Output Shape	Param #
conv2d_28 (Conv2D)	(None, 30, 30, 32)	320
conv2d_29 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_13 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_23 (Dropout)	(None, 14, 14, 64)	0
flatten_13 (Flatten)	(None, 12544)	0
dense_24 (Dense)	(None, 128)	1605760
dropout_24 (Dropout)	(None, 128)	0
dense_25 (Dense)	(None, 6)	774

Total params: 1,625,350

59

## Expected results

Particle type	# particles	Accuracy	Loss
electron	76	0.013158	5.97799
kaon	540	0.557407	1.147595
muon	12	0.000000	12.524119
pion	2058	0.986015	0.140477
proton	299	0.309448	2.457402

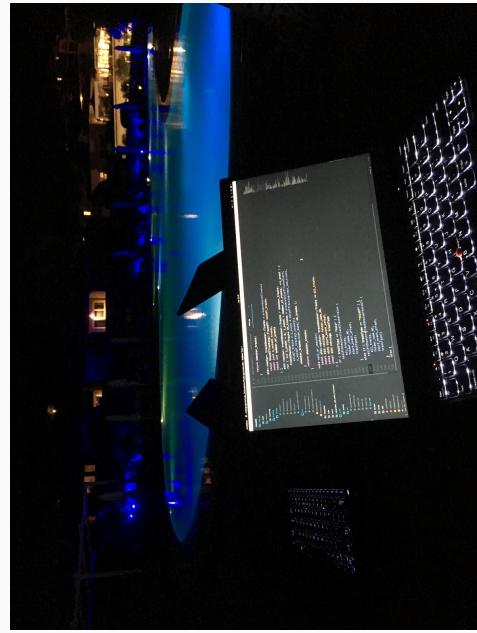
Predictions (%tot)	electron	kaon	muon	pion	protoa	protox	Purity (%)
electron	0.034	0.034	0.000	2.412	0.057	1.316	
kaon	0.000	10.084	0.000	5.829	2.178	55.741	
muon	0.000	0.000	0.000	0.369	0.034	0.000	
pion	0.034	1.374	0.000	66.533	1.006	96.501	
proton	0.000	2.647	0.000	4.322	3.049	30.435	

Efficiency (%)	1	50.000	71.327	0.000	83.727	48.143	1
ID eff (%)	1	K->pi,Pr,D	: 90.047	pi->e,m,pi	: 87.226		
W+ID eff (%)	1	K->e,m,pi	: 9.953	pi->K,Pr,D	: 12.774		

34

## See you in the exercise section!



## iSCC

## iSCC

## iSCC

Some CNNs are more amenable to solving the problem at hand. In fact, the problem looks very similar to the classical MNIST problem. One such CNN is:

Layer (type)	Output Shape	Param #
conv2d_28 (Conv2D)	(None, 30, 30, 32)	320
conv2d_29 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_13 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_23 (Dropout)	(None, 14, 14, 64)	0
flatten_13 (Flatten)	(None, 12544)	0
dense_24 (Dense)	(None, 128)	1605760
dropout_24 (Dropout)	(None, 128)	0
dense_25 (Dense)	(None, 6)	774

Total params: 1,625,350

60

## Resources i

- [1] CS231n: Convolutional Neural Networks for Visual Recognition.  
<http://cs231n.stanford.edu/>
- [2] LeCun et al., 1998. Gradient-based learning applied to document recognition.
- [3] Krizhevsky A., Sutskever I. and Hinton G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Part of: Advances in Neural Information Processing Systems 25 (NIPS).
- [4] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition.

63

## Bibliography

---

# Tensor Networks

## How physicists can tackle exponentially hard problems

March 4 – 7, 2019 | Patrick Emonts | Max Planck Institute of Quantum Optics



## Overview

---

- Introduction
- Matrix Product States
- iTEBD
- References

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

Slide 2

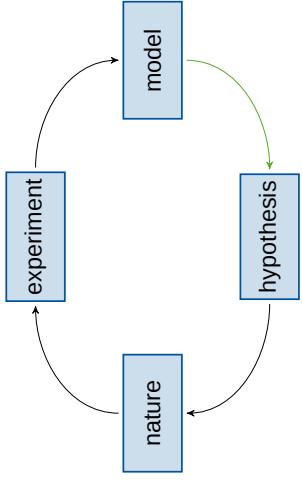


## Motivation

---

Section 1

### Introduction



```
graph TD; experiment[experiment] --> model[model]; model --> hypothesis[hypothesis]; hypothesis --> nature[nature]; nature --> experiment;
```

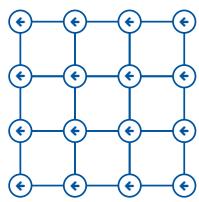
Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

Slide 4



## How complex is this problem?

We take a system that can take two states  $\uparrow$  and  $\downarrow$



$$\text{Number of possibilities}$$

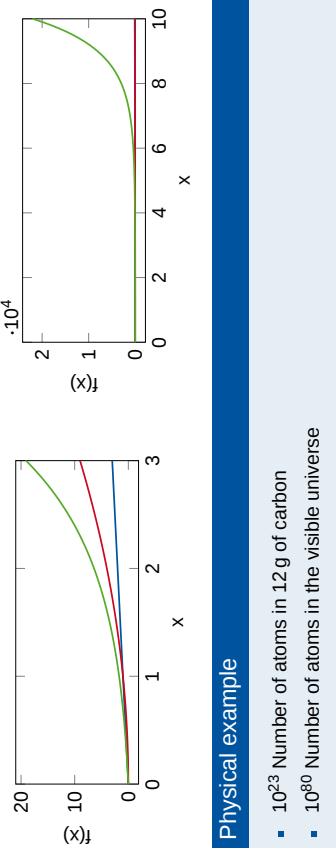
$$Z = 2^N$$

Slide 5

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts



## How bad is exponential scaling?



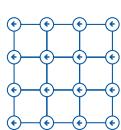
Slide 6

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts



## Quantum Mechanics in 2 slides – Slide 1

Hilbert space  $\mathcal{H}$  vector space of all possible configurations  
 state  $|\psi\rangle$  vector in  $\mathcal{H}$  that describes the state of the system  
 Hamilton operator  $H$  linear operator that describes the energy of the system



Note on Notation: Bra and Ket vectors

$ \psi\rangle$ is a column vector	$\langle\psi $ is a row vector
$\begin{array}{ c c c }\hline \overline{\Psi}_0 & \overline{\Psi}_1 & \overline{\Psi}_2 \\ \hline \end{array}$	$\begin{array}{c} \overline{\Psi}_0 \\ \overline{\Psi}_1 \\ \overline{\Psi}_2 \end{array}$

Slide 7

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts



## Quantum Mechanics in 2 slides – Slide 2

Schrödinger equation  $i\frac{d}{dt}H|\psi(t)\rangle = H|\psi(t)\rangle$   
 time-independent Schrödinger equation (time-ind. Hamiltonian)

$$H|\psi\rangle = E|\psi\rangle$$

Expectation values

$$\begin{aligned} \text{Probability theory} \\ \langle X \rangle = P(X)X \end{aligned}$$

$$\begin{aligned} \text{Quantum mechanics} \\ \langle E \rangle = \frac{\langle \psi | H | \psi \rangle}{\langle \psi | \psi \rangle} \end{aligned}$$

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts



## Expressing spins with matrices

### Definitions

$$\begin{aligned} |\uparrow\rangle &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ |\downarrow\rangle &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{aligned}$$

### Calculating with spins

$$\begin{aligned} S_z |\uparrow\rangle &= \frac{1}{2} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ &= \frac{1}{2} |\uparrow\rangle \end{aligned}$$

Slide 9

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts



## Combining multiple spins

Consider a system consisting of two spins that can have two values ( $\downarrow$  and  $\uparrow$ )

### Hilbert space $\mathcal{H}$

$$\mathcal{H} = \text{span} \{ |\downarrow_1\rangle |\downarrow_2\rangle, |\downarrow_1\rangle |\uparrow_2\rangle, |\uparrow_1\rangle |\downarrow_2\rangle, |\uparrow_1\rangle |\uparrow_2\rangle \}$$

Spins on different sites are combined by tensor products

$$\begin{aligned} |\downarrow_1\rangle |\downarrow_2\rangle &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} & |\downarrow_1\rangle |\uparrow_2\rangle &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \\ |\uparrow_1\rangle |\downarrow_2\rangle &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} & |\uparrow_1\rangle |\uparrow_2\rangle &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \end{aligned}$$

Slide 10

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts



## Letting spins interact

### Interaction of two spins

$$H = -J(S_1^z \otimes S_2^z)$$

### Matrix representation

$$\begin{aligned} H &= -J \left( \frac{1}{2} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \right) \otimes \left( \frac{1}{2} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \right) \\ &= -\frac{J}{4} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \end{aligned}$$

Slide 11

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts



## Calculating an expectation value

### Preparation of a state

$$\begin{aligned} |\psi\rangle &= \sqrt{0.5} |\uparrow_1\rangle |\downarrow_2\rangle + \sqrt{0.5} |\downarrow_1\rangle |\uparrow_2\rangle \\ &= \sqrt{0.5} |\uparrow_1\rangle |\downarrow_2\rangle + \sqrt{0.5} |\downarrow_1\rangle |\uparrow_2\rangle \\ &= \sqrt{0.5} |\uparrow\downarrow\rangle + \sqrt{0.5} |\downarrow\uparrow\rangle \end{aligned}$$

### Expectation value

$$\begin{aligned} \langle H \rangle &= \langle \psi | H | \psi \rangle / \langle \psi | \psi \rangle \\ &= \langle \psi | H | \psi \rangle \\ &= -\frac{J}{4} (0 \cdot \sqrt{0.5} \cdot \sqrt{0.5}) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ \sqrt{0.5} \\ \sqrt{0.5} \\ 0 \end{pmatrix} \\ &= \frac{J}{4} \end{aligned}$$

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

Slide 12

## Summary – Introduction

### Computation

- Computational complexity of many-body systems scales exponentially with the system size
- We cannot solve those systems exactly and have to use approximate methods

### Physics

- Quantum mechanical systems evolve according to the Schrödinger equation
- We are interested in the ground-state  $|\psi\rangle$  and expectation values  $\langle E \rangle = \frac{\langle \psi | H | \psi \rangle}{\langle \psi | \psi \rangle}$

Slide 13

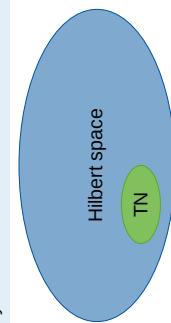
Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

HILBERT

## Tensor Networks

### Idea

Use an Ansatz with polynomially many parameters although the Hilbert space has exponentially many states



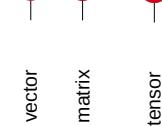
We explore only a small part of the Hilbert space

Slide 15

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

HILBERT

## Pictorial representation



- The number of legs determines the number of indices of the object
- A connection  $\Leftrightarrow$  Contraction of indices

Slide 16

HILBERT

### Section 2

#### Matrix Product States

### Pictorial representation

## Section 2

### Matrix Product States

HILBERT

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

Slide 14

### Physics

### Matrix Product States

Pictorial representation as Arrays

=

0	
1	
2	
3	
4	

=

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

=

(0,4)	(1,4)	(2,4)	(3,4)
(1,3)	(2,3)	(3,3)	(4,3)
(2,2)	(3,2)	(4,2)	(5,2)
(3,1)	(4,1)	(5,1)	(6,1)
(4,0)	(5,0)	(6,0)	(7,0)

17

Sensor Networks | March 4 - 7, 2018 | Patrick Emonts

1

## Calculations with pictures

$$\mathbf{v}_i = \sum_{j=1}^n A_{ij} \mathbf{b}_j$$

SM10 18

Tensor Networks | March 4 - 7, 2018 | Patrick Emonts

## Calculations with pictures – Quiz

$$i \quad B \\ A = C$$

100

$$c = \sum_{i,j} A_{ij} B = \text{Tr}[AB]$$

Sensor Networks | March 4 – 7, 2019 | Patrick Emonts

四

## Tensor manipulations – Grouping

Tensor to Matrix	Pictorial language
$A_{i,j,k} = A_{i,(jk)}$	$i \xrightarrow{k} j = i \xrightarrow{j} \text{red circle} = i \xrightarrow{j} \text{blue line}$

Tensor Networks | March 4 - 7, 2019 | Patrick Emonet

Slide 2C

## Grouping

Pictorial language

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)

四

## Tensor manipulations – Grouping

Tensor to Matrix	Pictorial language
$A_{i,j,k} = A_{i,(jk)}$	$i \xrightarrow{k} j = i \xrightarrow{j} \text{red circle} = i \xrightarrow{j} \text{blue circle}$

Tensor Networks | March 4 - 7, 2019 | Patrick Emonet

Slide 2C

## Tensor manipulations – Splitting

### Splitting of tensor

$$A = U \cdot S \cdot V^\dagger$$

Slide 21

Tensor Networks | March 4 - 7, 2019 | Patrick Emonts

MIT-QC

## Singular Value Decomposition

### Singular Value Decomposition

$$M = U \cdot S \cdot V^\dagger$$

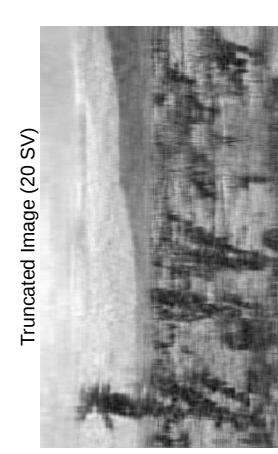
$$M = U \cdot S \cdot V^\dagger,$$

**M** arbitrary  $m \times n$  matrix**U** unitary  $m \times m$  matrix**S** diagonal  $m \times n$  matrix**V** unitary  $n \times n$  matrix

Slide 22

Tensor Networks | March 4 - 7, 2019 | Patrick Emonts

MIT-QC



Truncated Image (20 SV)



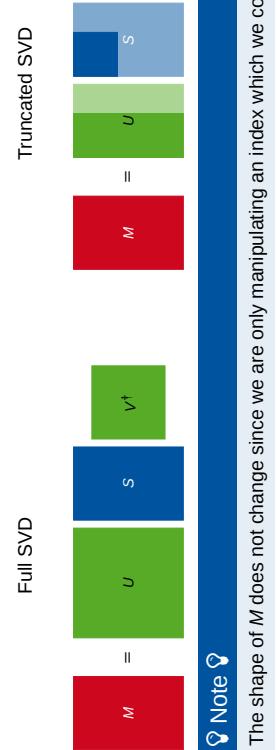
Original Image

MIT-QC

Tensor Networks | March 4 - 7, 2019 | Patrick Emonts

Slide 24

## SVD – Example



**?** Note **?**  
The shape of  $M$  does not change since we are only manipulating an index which we contract.

MIT-QC

Tensor Networks | March 4 - 7, 2019 | Patrick Emonts

Slide 23

## Back to formulas: What is a Tensor Network?

### A general quantum mechanical state

$$|\psi\rangle = \sum_{\sigma_1, \dots, \sigma_n} c_{\sigma_1, \sigma_2, \dots, \sigma_n} |\sigma_1 \sigma_2 \dots \sigma_n\rangle$$

Example:  $|\psi\rangle = \sqrt{0.5} |\uparrow\rangle |\downarrow\rangle + \sqrt{0.5} |\downarrow\rangle |\uparrow\rangle$

$$\begin{aligned} \mathcal{H} &= \text{span}\{ |\downarrow_1\rangle |\downarrow_2\rangle, \\ &\quad |\downarrow_1\rangle |\uparrow_2\rangle, \\ &\quad |\uparrow_1\rangle |\downarrow_2\rangle, \\ &\quad |\uparrow_1\rangle |\uparrow_2\rangle \} \\ c_{\downarrow_1, \downarrow_2} &= 0 \\ c_{\downarrow_1, \uparrow_2} &= \sqrt{0.5} \\ c_{\uparrow_1, \downarrow_2} &= \sqrt{0.5} \\ c_{\uparrow_1, \uparrow_2} &= 0 \end{aligned}$$

Slide 25

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts



## Back to formulas: What is a Tensor Network?

### A general quantum mechanical state

$$|\psi\rangle = \sum_{\sigma_1, \dots, \sigma_n} c_{\sigma_1, \sigma_2, \dots, \sigma_n} |\sigma_1 \sigma_2 \dots \sigma_n\rangle$$

### Problem

The coefficients depend on the configuration of all spins. Thus, there are exponentially many coefficients.

### A fancy way to write a quantum mechanical state

$$|\psi\rangle = \sum_{\sigma_1, \dots, \sigma_n} \underbrace{\sum_{a_1, \dots, a_{n-1}}}_{c_{\sigma_1, \sigma_2, \dots, \sigma_n}} A_{a_1}^{\sigma_1} A_{a_2}^{\sigma_2} \dots A_{a_{n-1}}^{\sigma_{n-1}} A_{a_n}^{\sigma_n} |\sigma_1 \sigma_2 \dots \sigma_n\rangle$$

Slide 26

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts



## Tensor Networks – Thinking about Indices

### A Tensor Network State

$$|\psi\rangle = \sum_{\sigma_1, \dots, \sigma_n} \sum_{a_1, \dots, a_{n-1}} A_{a_1}^{\sigma_1} A_{a_2}^{\sigma_2} \dots A_{a_{n-1}}^{\sigma_{n-1}} A_{a_n}^{\sigma_n} |\sigma_1 \sigma_2 \dots \sigma_n\rangle$$

### Dimensions of object A

$\sigma$ : physical index: ( $\uparrow$ ,  $\downarrow$ )

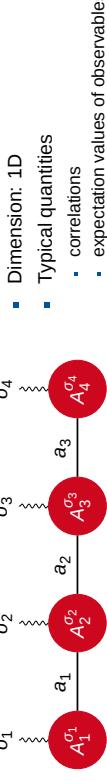
Dimension of physical index

$d$  ( $\sim 10$ )

Dimension of virtual index

$D$  ( $\sim 100$ )

$a$ : virtual index



Slide 27

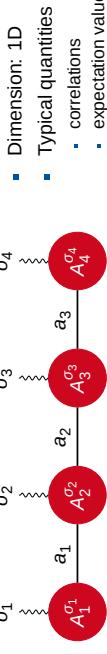
Tensor Networks | March 4 – 7, 2019 | Patrick Emonts



## Matrix Product States

### A Tensor Network State

$$|\psi\rangle = \sum_{\sigma_1, \dots, \sigma_n} \sum_{a_1, \dots, a_{n-1}} A_{a_1}^{\sigma_1} A_{a_2}^{\sigma_2} \dots A_{a_{n-1}}^{\sigma_{n-1}} A_{a_n}^{\sigma_n} |\sigma_1 \sigma_2 \dots \sigma_n\rangle$$



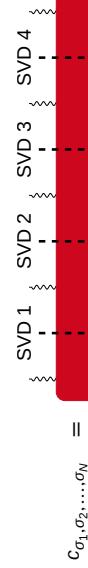
Tensor Networks | March 4 – 7, 2019 | Patrick Emonts



## Matrix Product States – How to get the Tensors?

A general quantum mechanical state

$$|\psi\rangle = \sum_{\sigma_1 \dots \sigma_n} c_{\sigma_1, \sigma_2, \dots, \sigma_n} |\sigma_1 \sigma_2 \dots \sigma_n\rangle$$



Matrix Product state

$$|\psi\rangle = \sum_{\sigma_1, \dots, \sigma_n} \sum_{a_1, \dots, a_{n-1}} A_{a_1}^{\sigma_1} A_{a_2}^{\sigma_2} \dots A_{a_{n-1}}^{\sigma_{n-1}} A_{a_n}^{\sigma_n} |\sigma_1 \sigma_2 \dots \sigma_n\rangle$$

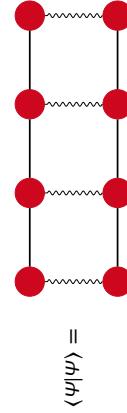
Slide 29

Tensor Networks | March 4 - 7, 2019 | Patrick Emonts



## Matrix Product States – Bra, Ket and Norms

$$|\psi\rangle = \dots$$



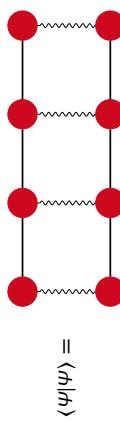
Tensor Networks | March 4 - 7, 2019 | Patrick Emonts

Slide 30



## Matrix Product States – Why contraction order matters!

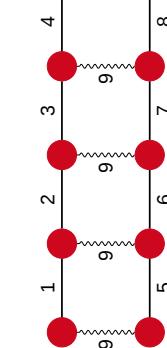
Different contraction orders yield different contraction complexities



## Matrix Product States – Why contraction order matters!

Number of operations

- |   |                        |
|---|------------------------|
| 1 | $\mathcal{O}(D^2 d^2)$ |
| 2 | $\mathcal{O}(D^2 d^3)$ |
| 3 | $\mathcal{O}(D^2 d^4)$ |
| 4 | $\mathcal{O}(D d^5)$   |
| 5 | $\mathcal{O}(D^2 d^2)$ |



! Don't try this at home !

The number of matrix elements needed scales exponentially with the number of sites  $N$ .



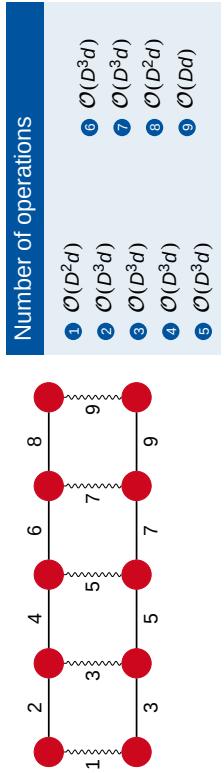
Tensor Networks | March 4 - 7, 2019 | Patrick Emonts

Slide 32



Tensor Networks | March 4 - 7, 2019 | Patrick Emonts

## Matrix Product States – Why contraction order matters!



### Complexity

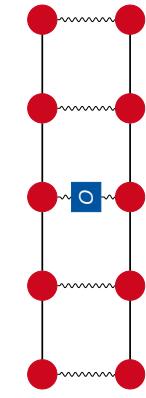
The number of matrix elements does not depend on the number of sites  $N$  at all and the procedure scales linear in time with  $N$ .

Slide 33

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

HPC

## Matrix Product States – Calculation of an expectation value



$$\langle \psi | O | \psi \rangle =$$

$$\langle O \rangle = \frac{\langle \psi | O | \psi \rangle}{\langle \psi | \psi \rangle}$$

$$\langle O_i \rangle = \frac{\langle \psi | O_i | \psi \rangle}{\langle \psi | \psi \rangle}$$

Slide 34

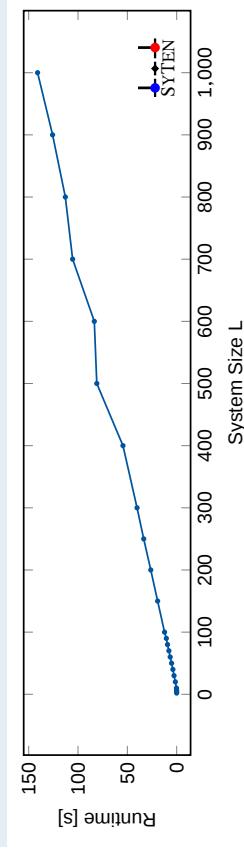
Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

HPC

## Example – DMRG calculation

### Spin 1 Heisenberg chain

$$H = \sum_i S_i^x S_{i+1}^x + S_i^y S_{i+1}^y + S_i^z S_{i+1}^z$$



Data provided by Claudio Hubig, MPQ

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

HPC

## Summary – MPS

- MPS is an Ansatz to describe many-body states with polynomially many parameters
- The pictorial description simplifies the formulation of calculations and algorithms
- We have to be careful about the order of contractions

Slide 36

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

HPC

### Section 3

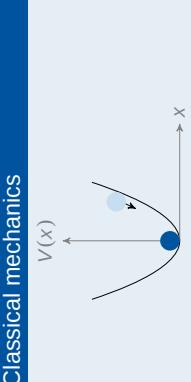
iTEBD

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

Slide 37

### Minimization of energy

**Goal**  
Find the groundstate of a Hamiltonian  $H$ , i.e. find the state with the smallest energy eigenvalue.

<b>Classical mechanics</b> 	<b>Quantum mechanics</b> Find $ \psi_{\min}\rangle$ such that $E_{\min} = \frac{\langle \psi_{\min}   H   \psi_{\min} \rangle}{\langle \psi_{\min}   \psi_{\min} \rangle}$ is minimal.
--	---

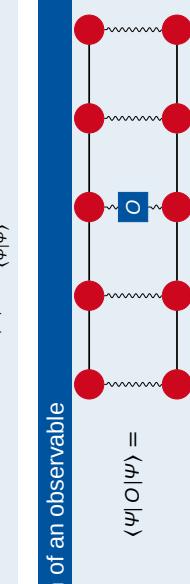
Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

Slide 38

### Reminder – Calculation of energies

**Expectation value**  
 $\langle E \rangle = \frac{\langle \psi | H | \psi \rangle}{\langle \psi | \psi \rangle}$

**Calculation of an observable**  
 $\langle \psi | O | \psi \rangle =$



Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

Slide 40

### Reminder – Tensor network notation

**A Tensor Network State**  
 $|\psi\rangle = \sum_{\sigma_1, \dots, \sigma_n} \sum_{a_1, \dots, a_{n-1}} A_{a_1}^{\sigma_1} A_{a_1, a_2}^{\sigma_2} \dots A_{a_{n-2}, a_{n-1}}^{\sigma_{n-1}} A_{a_{n-1}}^{\sigma_n} |\sigma_1 \sigma_2 \dots \sigma_n\rangle$



Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

Slide 39

## Energy minimization via imaginary time evolution

### Motivation

The ground state is the state with the smallest energy. All other states are suppressed more quickly by an exponential.

### Time evolution in imaginary time

$$\begin{aligned} |\psi_0\rangle &= \lim_{\delta \rightarrow \infty} \frac{\exp(-t/\delta) |\psi\rangle}{\|\exp(-t/\delta) |\psi\rangle\|} \\ &= \lim_{\delta \rightarrow \infty} \frac{U(\delta) |\psi\rangle}{\|U(\delta) |\psi\rangle\|} \end{aligned}$$

Slide 41

Tensor Networks | March 4 - 7, 2019 | Patrick Emonts



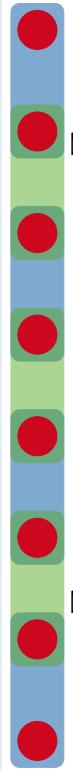
## Trotterization of an operator

### Evolution operator

$$U(\delta) = e^{-\delta H}$$

### I sing Model

$$H = \sum_i S_i^x S_{i+1}^z = \sum_i h_i h_{i+1}$$



$$H_{\text{even}} = \sum_{i \text{ even}} h_{i,i+1}$$

$$H = H_{\text{even}} + H_{\text{odd}}$$

Slide 42

Tensor Networks | March 4 - 7, 2019 | Patrick Emonts

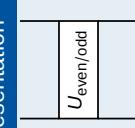


## Trotterization of an operator

### Evolution operator

$$U(\delta) = e^{-\delta H}$$

### Pictorial representation



### Trotterization of an operator

$$\begin{aligned} U(\delta) &= e^{-\delta H} \\ &= e^{-\delta H_{\text{even}}} e^{-\delta H_{\text{odd}}} e^{-\delta^2 [H_{\text{even}}, H_{\text{odd}}]} \\ &\approx e^{-\delta H_{\text{even}}} e^{-\delta H_{\text{odd}}} \end{aligned}$$

Slide 43

Tensor Networks | March 4 - 7, 2019 | Patrick Emonts

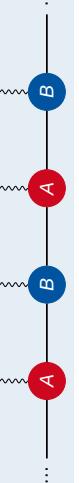


## Making life easy: infinite systems

### General MPS



### Translationally invariant MPS



Tensor Networks | March 4 - 7, 2019 | Patrick Emonts

Slide 44



## Back to the start: An MPS with diagonal matrices

We started with

$$c_{\sigma_1, \sigma_2, \dots, \sigma_N} = \begin{array}{c} \text{---} \\ \vdots \\ \vdots \\ \vdots \\ \text{---} \end{array}$$

and got

$$c_{\sigma_1, \sigma_2, \dots, \sigma_N} = \begin{array}{c} \text{---} \\ \vdots \\ \vdots \\ \vdots \\ \text{---} \end{array} \quad \begin{array}{c} \text{---} \\ \vdots \\ \vdots \\ \vdots \\ \text{---} \end{array}$$

Slide 45

Tensor Networks | March 4 - 7, 2019 | Patrick Emonts



## The iTEBD algorithm

We start with an infinite system that consists of two sites A and B



### Disclaimer

This algorithm is proven to be numerically unstable. You should NOT use it in research, it is shown here due to its simplicity.

Slide 46

Tensor Networks | March 4 - 7, 2019 | Patrick Emonts



## The iTEBD algorithm

We start with an infinite system that consists of two sites A and B



### Disclaimer

This algorithm is proven to be numerically unstable. You should NOT use it in research, it is shown here due to its simplicity.

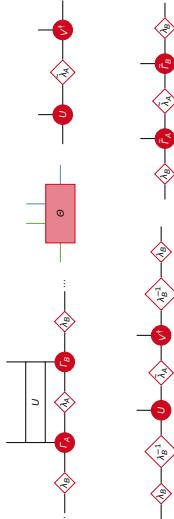
Slide 46

Tensor Networks | March 4 - 7, 2019 | Patrick Emonts



## The iTEBD algorithm

- 1 Apply the operator  $U$  to sites A and B
- 2 Contract all indices and group indices (blue and green)
- 3 Compute SVD of the tensor
- 4 Reintroduce  $\lambda_B$
- 5 Update  $r_A$  and  $r_B$
- 6 Repeat the procedure with the sites B and A



Tensor Networks | March 4 - 7, 2019 | Patrick Emonts

**Results for an Ising spin system**

**Ising Model**

$H = \sum_i S_i^z S_{i+1}^z = \sum_i h_i h_{i+1}$

Parameters

- D = 1
- δ = 0.001

Tensor Networks | March 4 – 7, 2019 | Patrick Emonds

Slide 49

**Tensor Networks**

How physicists can tackle exponentially hard problems

March 4 – 7, 2019 | Patrick Emonds | Max Planck Institute of Quantum Optics

MPQ

**Application on odd an even sites**

**Application to even sites**

**Application to odd sites**

Tensor Networks | March 4 – 7, 2019 | Patrick Emonds

Slide 48

**Outlook**

**PEPS**

**MERA**

Tensor Networks | March 4 – 7, 2019 | Patrick Emonds

Slide 50

## Section 4

### References

#### References I

- Basics** ■ Jacob C. Bridgeman and Christopher T. Chubb. "Hand-waving and Interpretive Dance: An Introductory Course on Tensor Networks". In: *Journal of Physics A: Mathematical and Theoretical* 50.22 (June 2, 2017), p. 223001
- Román Orús. "A practical introduction to tensor networks: Matrix product states and projected entangled pair states". In: *Annals of Physics* 349 (Oct. 2014), pp. 117–158
- Ulrich Schollwöck. "The density-matrix renormalization group in the age of matrix product states". In: *Annals of Physics* 326.1 (Jan. 2011), pp. 96–192
- ITEBD** ■ G. Vidal. "Classical Simulation of Infinite-Size Quantum Lattice Systems in One Spatial Dimension". In: *Physical Review Letters* 98.7 (Feb. 12, 2007)

Slide 52

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

EPFL

Tensor Networks | March 4 – 7, 2019 | Patrick Emonts

EPFL

# Hardware Acceleration Through FPGAs

1. Basic Concepts and Examples

Giorgio Lopez  
CERN TE/EP/CCE

## Summary

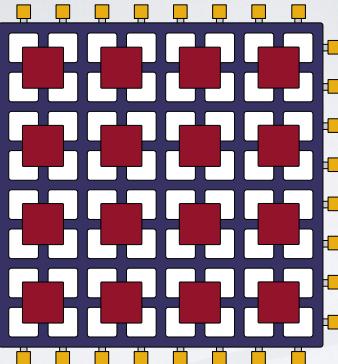
- Elements of an FPGA
- FPGA Timing
- Parallel Processing Paradigms and Challenges
- FPGA-based System Architectures
- Hardware / Software Partitioning
- FPGA Design Flow Basics

2

## Elements of an FPGA

- Basic components of an FPGA include:
- Array of Logic Blocks
  - Block RAMs
  - Clock Management Circuitry
  - I/O Blocks and Interconnect Elements
  - DSP Elements (Multipliers, Accumulators, etc.)
- Fancier models may include:
- Transceivers
  - Physical CPU cores
  - External Bus Interfaces (ex. PCIe)

3



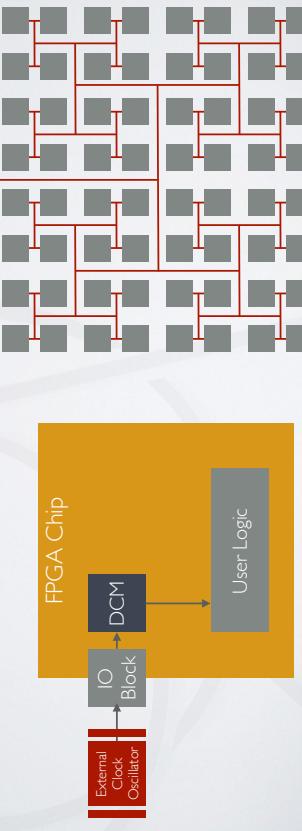
## Elements of Logic Blocks

- Look Up Tables (combinatorial): truth-table-like implementation of any N-inputs boolean function
- Flip Flops (sequential): atomic blocks of clocked memory storing a single bit

4

# FPGA Clocking

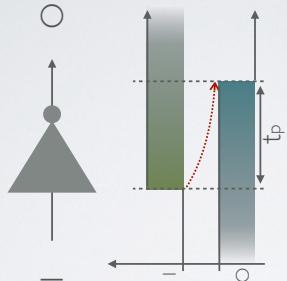
- Clock Management Units (DCMs, PLLs): align internally distributed clock with external oscillator / input. Can also alter clock frequency.



5

# Timing in FPGAs

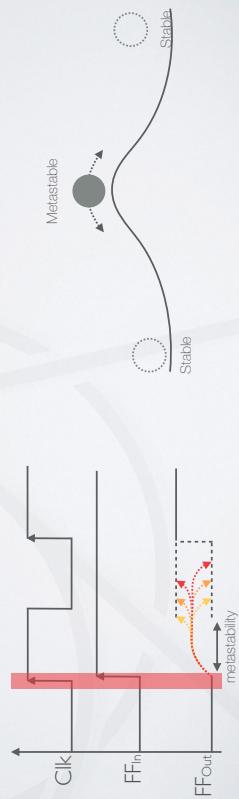
- Propagation of digital signals, most notably on long combinatorial paths, implies delays.
- This is due to the electrical characteristics of logic gates and data transfer lines and can be made worse by:
  1. Fan-In (# of inputs to the logic gate)
  2. Fan-Out (# of circuits fed by logic gate)
  3. Length of transfer line (good routing crucial)



6

# Metastability

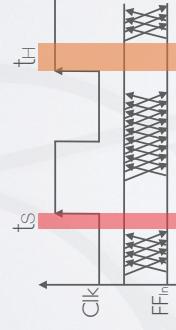
- Flip Flops sample their inputs on the active edge of clock
- Signal must be stable around this edge for correct sampling
- Failure in meeting this condition can lead to "metastability", a condition where the output of a Flip Flop can get to an intermediate, unstable voltage value, staying in this state for unpredictable time before finally settling in one of the two stable states.



7

# Timing Constraints

- A "safety window" around the clock edge must be established: this is given by the Hold Time  $t_H$  and Setup Time  $t_S$



- Too long/short combinatorial paths between Flip Flops can "challenge" the fulfillment of these constraints

8

# Metastability and Asynchronous Inputs

- Even if we correctly route signals internally, external signals will always be asynchronous

- This is also true for signals coming from different clock domains



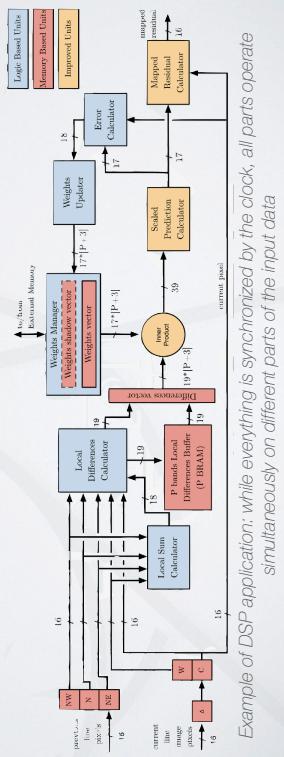
Solution:

- Synchronizer chains: cascades of 2 or 3 flip flops on “synchronization boundaries” which make metastability way less likely ( $P_{\text{event}} = \sim 10^{-21}$ )

9

# Data Intensive Applications: Why Use FPGAs?

- As opposed to the flow in a CPU which is primarily sequential, digital circuits are intrinsically parallel
- FPGAs are the most accessible and fast-time-to-market solution for digital circuit implementation of applications
- When the data processing flow is predominant on complex control logic an FPGA can greatly accelerate the performance of their application



# Challenges of Pipelining

- There's always an initial time to “fill the pipeline” where only a subset of steps will be actually operating (can be negligible)
- For pipelining to be effective all stages must take the same time to be executed (to avoid idle times)



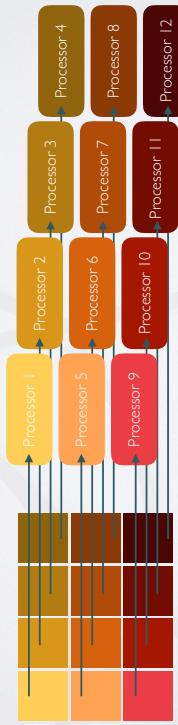
# Parallel Processing Paradigms: Pipelining

- When implementing a data processing algorithm which implies several phases, each step of the algorithm can be given to a specific unit, with “checkpoints” between stages. At any given time, each unit will be performing its transformation on a different set of data.



## Parallel Processing Paradigms: Matrix Partitioning

A matrix-like set of data (e.g. an image) can be decomposed in subparts or tiles, each to be independently handled and processed by a specific unit.



13

## Challenges of Matrix Partitioning

- Since all processing units are identical, the paradigm works well if the workload is also identical for each tile.
- If input data size can vary this can complicate the scheme

14

## Parallel Processing Paradigms: Dynamically Scheduled Partitioning

A more sophisticated example: when the workload can be different from tile to tile, a scheduler can allocate resources to different chunks of the input data (also to accomodate changes in the size of the input itself).



15

The scheduler could be implemented in a processor core (even on board of the FPGA)

## Typical System Architectures: Standalone FPGA Systems

- The FPGA operates without the aid of a CPU; an external (DRAM) memory may or may not be present.
- Most indicated when the algorithm is data-intensive and of relatively low control complexity.



16

# Typical System Architectures: Multiprocessor Arrangement

- The FPGA behaves as an additional processor, residing on the system bus and receiving input data by the CPU via DMA on a local memory, transferring back the results either through DMA or through mapped registers,
- Control flow managed by the CPU / computationally dense portions of the algorithm accelerated via hardware implementation.



# Typical System Architectures: System-on-Chip Architecture

- An evolution of the previous scheme, where the whole System is on the FPGA chip itself: the CPU can be either a physical CPU core embedded on the Silicon die or a "soft-CPU" IP block (or even more than one...)
- Can be an easy access solution for complex algorithms
- Typically some RAM (internal or external w.r.t. the FPGA) will be needed



# Hardware / Software Partitioning: Why?

- Typically, the engineering cost required to translate a whole algorithm into a digital circuit implementation is not reasonable.
- As a rule of thumb, 90% of the time is spent in execution of 10% of the code
- Complex applications are composed by "sequential" or "control intensive" tasks and "data intensive" tasks.
- While "data intensive" tasks would crucially benefit from parallel implementations on FPGA technology, the same doesn't hold for the control part of the algorithm, which is more aptly carried out by a processor.

19

# Hardware / Software Partitioning: How?

- A crucial step is understanding which parts of the SW algorithm are taking up most of the processing time
- Profiling tools are used for this purpose (they must be run on the target platform/architecture)
- In addition, it is generally more beneficial to keep complex control logic in the SW domain, whereas data-intensive tasks are more easily translated into a digital circuit

20

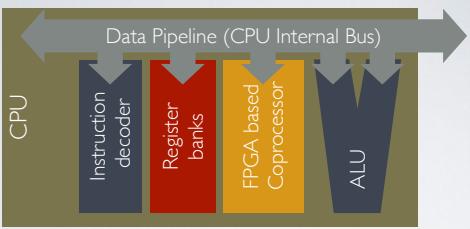
## Connections Scheme: Bus-Connected FPGA Coprocessor



- In HW/SW partitioning the FPGA acts as a coprocessor to the CPU (like a FPU would do)
- Several schemes are possible: for instance one where the FPGA exchanges data with RAM through a DMA and "wakes" the CPU with an IRQ line (e.g: for longer processing and/or larger in/out data transfers)

21

## Connections Scheme: Instruction Pipeline Connection



- A more tightly coupled scheme, useful for operation with smaller operands and faster execution times (otherwise the CPU would get stalled)
- Special instructions, not recognized by the CPU, are processed by the FPGA coprocessor directly, which takes inputs and returns outputs/statuses on the CPU data pipeline (the internal bus)

22

## Hardware / Software Partitioning: Advanced Strategies

- In general, HW/SW partitioning is not a trivial task and several approaches exist in literature to explore the space of solutions.
- Examples include algorithms based on simulated annealing, branch and bound and so on...

23

## Communication Bottlenecks

- FPGA processing can be fast, but when HW/SW partitioning is used we also have to keep in mind the bandwidth of communication between CPU/RAM and FPGA.
- Example: a circuit to obtain the mean of arrays of 1024 64-bits values
  - Input: 1024 values array from external RAM (size = 65Kb)
  - Output: average value to external RAM (size = 64 bit)
- Mean in  $\log_2(1024) = 10$  clock cycles @ (let's say) 100MHz = 100ns
  - RAM bandwidth (DDR2): 3200MB/s - transfer of 65Kb takes 2.56us
  - Transfer time of the array from RAM to FPGA is dominant!

24

# Basics of the Design Flow for FPGAs

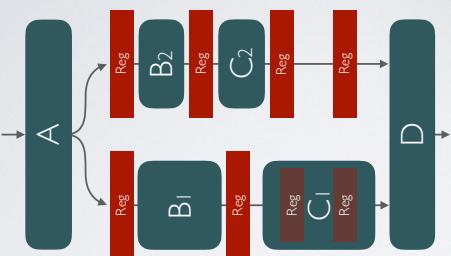
- Main Steps of the FPGA design flow:
- Hardware Description (HDL/Schematics/etc.)
- Synthesis
- Mapping
- Place and Route
- Bitstream Generation and Deployment

26

# Synchronization Challenges

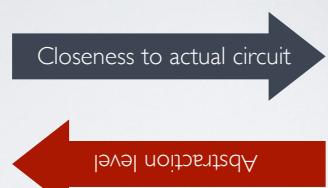
- Another problem that arises when dealing with complex digital circuits is synchronization: all parts which operate on the same piece of data must be timed coherently.
- For instance, in a pipeline, where stages which operate at different times on the same (or correlated) pieces of data.
- In addition, some stages may have internal registers to shorten combinatorial delays. This also needs to be considered.

25



# Hardware Description

- The first step in the design flow is the description of the user logic. This can be specified in a broad variety of abstraction levels, including:



- High level programming languages
- Behavioral descriptions
- Register Transfer Level description
- Schematic Capture

27

28

# Translate and Map

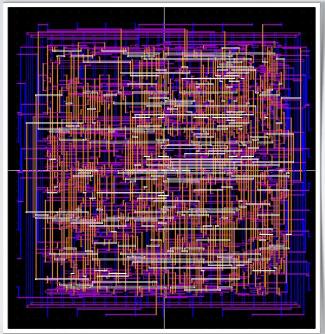
- The “netlist” generated by the synthesizer is translated on the physical logic cells that compose the programmable fabric of the FPGA.
- The mapping then fits the obtained design into the resources which are physically available on the specifically targeted FPGA chip. This includes logic cells, block RAM but also clocking resources, DSP primitives and so on...

29

# Place and Route

- The logic blocks are then assigned to physical locations on the device based on the given design constraints (which specify timing requirements and the position on the FPGA chip of the I/O ports needed by the user logic design).

30



# Bitstream Generation and Deployment

- The bitstream is a binary file generated from the routed design, ready to be downloaded to the physical FPGA device.
- Deployment on the FPGA is carried out through standard interfaces like JTAG ports and can be volatile or non volatile (e.g. through on board or external SPI Flash memories)

31



# Simulating the Design

- Simulation is a very important step in the design of digital circuits
- A gradual approach to simulation (e.g. bottom up) is recommended to help bugs emerge earlier
- Simulation can be either behavioral (doesn't take into account delays in the implementation) or time-accurate (either post synthesis or post place and route, requires more resources and can change if the design is re-synthesized)
- In the majority of cases, behavioral simulation already can make any flaw in the design evident.

32

# Take Home Messages

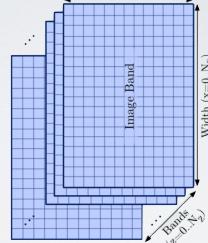
- FPGAs can be powerful allies in intensive data processing applications
- It is possible (and advisable) to focus on specific parts of the algorithm to be implemented in hardware, leaving the rest in software
- Several processing paradigms and system architectures are possible: it is important to choose wisely, according to the specific application/algorithm

33

## A Standard Compliant Compressor for Hyperspectral Images

Hyperspectral Images are three-dimensional arrays of pixels. Each layer represents a spectral band. Typical image size: 1 billion pixels and over

Host Systems such as satellites and aircrafts: low memory, low power, communication links towards base stations are limited in bandwidth



An efficient compression scheme is crucial

35

## Backup Slides

A couple of real life examples

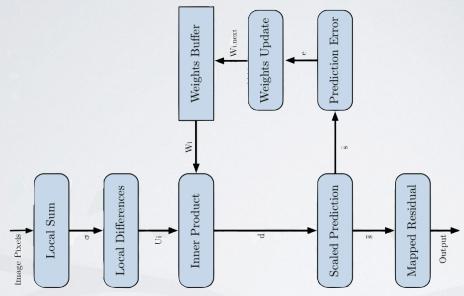
## CCSDS 123.0 Standard Algorithm

CCSDS-123.0 is a lossless, causal algorithm. It is linear and follows an adaptive approach: weights are updated at each pixel on the basis of the error in the previous pixel prediction.

Number of arithmetic operations per single pixel = 100 ca. (rough estimate)

...and this is NOT the number of instructions per pixel!

36



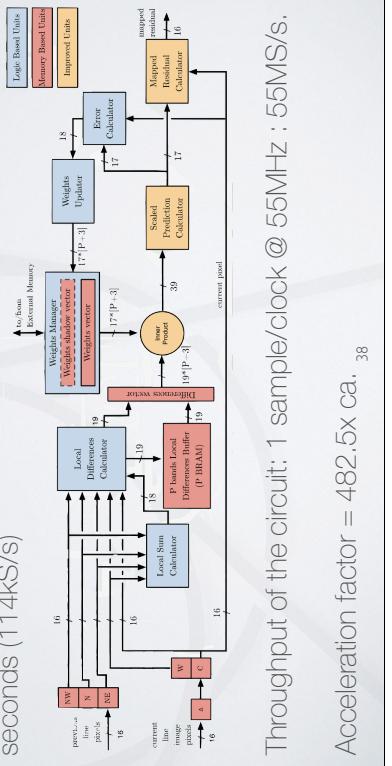
## Constraints and Strategies

- The algorithm needed to be implemented on a space graded platform. As a consequence no hard-CPU core adoption was possible
  - High performance was a crucial requirement (throughput over 5 Msamples/sec)
  - The full algorithm was translated into a digital circuit with a pipeline approach: different pixels enter the pipeline in different clock cycles and, at full speed, one sample per clock cycle is processed

37

## Implementation Results

- The time for execution of the standard software algorithm on an image of 1024x1024x6 pixels (which is quite small compared to the average dataset) on an intel i7 workstation with 32GB of RAM is about 55 seconds ( $114\text{ks/s}$ )



- Acceleration factor = 482,5x ca. <sup>38</sup>

- Throughput of the circuit: 1 sample/clock @ 55MHz : 55MS/s.

# Hardware Acceleration Through FPGAs

## 2. Basics of VHDL

Giorgio Lopez  
CERN TE/EPC/OCE

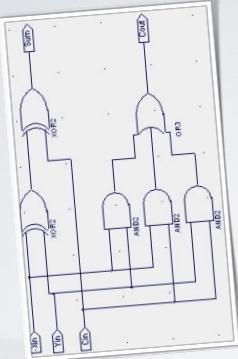
# Summary

- FPGA-based Design Techniques
- VHDL and Design Styles
- Anatomy of an HDL Project / SoC Project
- Importance of Simulation and Testbenches
- Special Signals, Clocks and Resets
- VHDL Signal Types
- Basic Constructs of VHDL

2

# FPGA-based Design Techniques: Schematic Entry

To describe user logic in an FPGA it is possible to manually draw the building blocks (multiplexers, logic gates, counters, etc.) and their connections by using a GUI.



Several disadvantages:

- lack of portability across platforms
- lack of maintainability
- hard to handle for large and complex projects

Not very used anymore, support is dropping.

3

# FPGA-based Design Techniques: HDL Languages

- Hardware Description Languages (HDL) enable a formal description of the behavior and/or structure of a digital circuit.
- More scalable, can be managed with versioning systems
- Most common examples: VHDL or Verilog

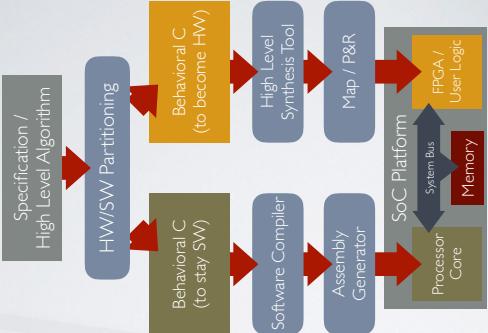
```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 entity signed_adder is
5   port (
6     a : in std_logic;
7     b : in std_logic;
8     clk : in std_logic;
9     q : out std_logic_vector(15 downto 0);
10    n : out std_logic_vector(15 downto 0);
11    b : out std_logic_vector(15 downto 0));
12 end signed_adder;
13
14 architecture signed_adder of signed_adder is
15 begin
16   architecture is
17   begin
18     process (a, b, clk)
19       begin
20         if (a >= b) then
21           q <= a;
22         else
23           q <= b;
24         end if;
25       end process;
26     end if;
27   end process;
28   begin
29     if (a >= b) then
30       if (a >= b) then
31         q <= a;
32       else
33         q <= b;
34       end if;
35     end if;
36   end process;
37 end signed_adder;
```

4

# FPGA-based Design Techniques:

## High Level Synthesis

- HLS tools enable automatic translation of blocks from a high level language (such as C/C++) into an HDL
- Typical use is in HW/SW partitioned architectures
- For the implementation to be effective, though, a deep knowledge of the tools and a proper constraining of the translation is needed



5

## VHDL Design Styles

- VHDL can be written according to several basic styles, depending on the used constructs and the way logic is described. Main styles are:

- Behavioral VHDL
- Dataflow VHDL
- Structural VHDL

6

## VHDL Design Styles:

### Structural VHDL

- Structural VHDL describes the structure (as in, the components that are visible in a structure). The visible components are instantiated in the declarative part of the architecture body.

```
architecture structural of mux4tol is
component and3
port(in1,in2,in3 : in std_logic;
out : out std_logic);
end component;
component or4
port(in1,in2,in3,in4 : in std_logic;
out : out std_logic);
end component;
begin
A0 : and3 port map( in1 => NOR s0,
in2 => NOR s1,
in3 => in0 );
A1 : and3 port map( in1 => s0,
in2 => NOR s1,
in3 => in1,
out => out0 );
A2 : and3 port map( in1 => NOR s0,
in2 => s1,
in3 => in2,
out => out1 );
A3 : and3 port map( in1 => s0,
in2 => s1,
in3 => in3,
out => out2 );
OUT : or4 port map( in1 => out0,
in2 => out1,
in3 => out2,
in4 => out3,
outc => muxout );
... architecture dataflow of mux4tol is
begin
muxout <= out0 OR out1 OR out2 OR out3;
out0 <= in0 AND NOT s0 AND NOT s1;
out1 <= in1 AND s0 AND NOT s1;
out2 <= in2 AND NOT s0 AND s1;
out3 <= in3 AND s0 AND s1;
```

7

## VHDL Design Styles:

### Dataflow VHDL

- In Dataflow VHDL the boolean or arithmetic transformation applied to data are explicitly described with signal assignments
- Keep in mind: all statements are concurrent!

```
... architecture dataflow of mux4tol is
begin
muxout <= out0 OR out1 OR out2 OR out3;
out0 <= in0 AND NOT s0 AND NOT s1;
out1 <= in1 AND s0 AND NOT s1;
out2 <= in2 AND NOT s0 AND s1;
out3 <= in3 AND s0 AND s1;
```

8

# VHDL Design Styles: Behavioral VHDL

- Behavioral VHDL describes the operation of the digital circuit with processes where concurrent statements are elaborated in a sequential way with the control flow constructs of traditional programming languages (if..else.., case.., etc)

6

# Anatomy of an HDL Project

- Elements that compose an HDL Project are:
    - Modules Hierarchy
    - Top Level Entity
    - Design Constraints

—

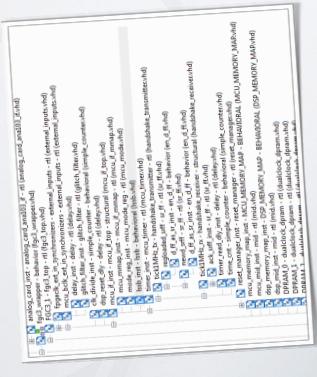
# VHDL Design Styles Recap

- Dataflow can be used for simple units and/or where a higher visibility of the logical connections between signals is desirable
  - Structural is mainly useful when only an interconnection of other building blocks is to be put in place (e.g: high hierarchical level modules)
  - Behavioral better describes more complex control flows (e.g: Finite State Machines)
  - Nonetheless, mixed approaches can be used!

10

# Modules Hierarchy

- The VHDL modules in a project compose a “tree” of nested entities which, all together, implement the user logic.

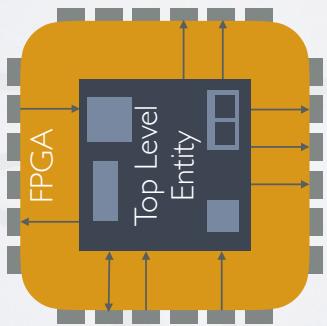


12

Giorgio Lopez

# Top Level Entity

- This is the module that holds all connection with the “external world”. It wraps all the logic and its inputs and outputs correspond to the physical IO blocks of the FPGA.



# Timing Constraints

- All complementary information which is needed for the design to be implemented on the device according to the timing requirements is put in the timing constraints. These may include, for example:

- External and derived clock period
- Timing relationships between externally fed signals
- Paths to be ignored in the timing analysis
- Multi - cycle signals
- And so on...

14

# I/O Constraints

- These constraints are mainly needed to bind signals to physical locations on the FPGA

- A typical example are I/O Location Constraints, which assign top level entity ports to physical IO blocks on the FPGA

- They can also specify other characteristics of the IO signals, e.g. drive strength, type of electric termination, digital voltage standard

NET "PORTLED_VS_BLUE"	LOC = P2	1057ANDARD = LVCMOS33	SLEN = SLOW	DRIVE = 8;
NET "PORTLED_VS_GREEN"	LOC = K5	1057ANDARD = LVCMOS33	SLEN = SLOW	DRIVE = 8;
NET "PORTLED_VS_RED"	LOC = P1	1057ANDARD = LVCMOS33	SLEN = SLOW	DRIVE = 8;
NET "C62_DIN<0>"	LOC = F19	1057ANDARD = LVCMOS33	SLEN = SLOW	DRIVE = 8;
NET "C62_DIN<1>"	LOC = F18	1057ANDARD = LVCMOS33	SLEN = SLOW	DRIVE = 8;
NET "C62_DIN<2>"	LOC = F17	1057ANDARD = LVCMOS33	SLEN = SLOW	DRIVE = 8;
NET "C62_DIN<3>"	LOC = F16	1057andard = LVCMOS33	SLEN = SLOW	DRIVE = 8;
NET "C62_ER_CKE"	LOC = E20	1057andard = LVCMOS33	PULLUP ;	
NET "DSP_ER_CKE"	LOC = A11	1057andard = LVCMOS33	PULLUP ;	
NET "DSP_ER_CS"	LOC = G7	1057andard = LVCMOS33	PULLUP ;	

# Creating a SoC Project

- “Soft” CPU core
- Custom block from RTL VHDL code
- IP block by tool vendor

15

Giorgio Lopez

Hardware Acceleration Through FPGAs

# Creating a SoC Project

- IP blocks can be typically customized in several aspects to match the needs of the designer (i.e.: CPU cores can be made more performing and feature rich or lower in resource footprint)
  - The tools allow easy generation of the memory map helping the development of drivers to access the peripherals from the CPU at the high abstraction level of the C/C++ code or the OS

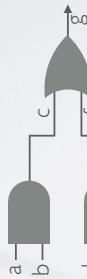


17

HDL is not programming!

A very important thing to remember (if not the most),

Typical gotcha:



yield the same result!

This is because HDL is describing logic circuits, not operations!

**HDI STATEMENTS ARE CONCURRENT! THINK HARDWARE!**

# Structure of a VHDL Module

A VHDL module is characterized by:

- Used libraries declarations
  - Entity declaration
  - Ports list (input output inout etc.)
  - Architecture head and body (entity implementation)
  - Other features (generics, multiple architectures, etc.)

```
entity topLevel is
  port (clk : in std_logic;
        rst : in std_logic;
        d   : in std_logic;
        q   : out std_logic);
end topLevel;

architecture rtl of topLevel is
begin
  signal intern : unsigned(2 downto 0) := 0;
begin
  process(clk, rst)
    begin
      if (rst = '1') then
        intern <= 0;
      elsif (clk'event and clk = '1') then
        intern <= intern + d;
      end if;
      q <= intern;
    end process;
  end;
```

8

Non-Synthesizable VHDL

- Some constructs in VHDL make code non synthesizable
    - A remarkable case: delays
    - Another one: loops. Loops in synthesizable VHDL are only to replicate logic. For other purposes, use sequential processes
    - Nevertheless, non-synthesizable VHDL is ok for testbenches
    - It's important to know what is synthesizable and what is not

Giorgio Lopez

# The importance of simulation

- One of the biggest problems in dealing with FPGAs is design validation.
- Once the design is deployed on the device access to signals is very limited
  - Only the top level ports
  - Test points? Maybe, but how many?
  - Internal Logic Analyzers (they occupy resources, modify the routed design and can break timing)
  - Finding bugs can be **VERY** frustrating and time consuming this way
  - Only solution is to use simulation as an integrated part of the design flow

21

# Testbench design

- Testbench design becomes a very important phase of the design flow (comparable to the development of the actual logic)
- A sophisticated testbench can (and should) include features such as:
  - Input randomization (or constrained randomization)
  - Assertions for automatic output validation
  - File I/O (e.g.: to read input vector)
  - Code Coverage verification
  - Bus transaction modeling
- There are languages and methodologies which are specifically developed with verification in mind (see SystemVerilog, UVM)...and they are more complex than VHDL itself!

22

# Special Signals Handling: Clock

Special care must be taken with clock for multiple reasons:

- Clock is fed to an enormous number of Flip Flops: high fanout / need for careful buffering
  - Skew must be kept low (balanced clock trees)
- All this is taken care by the tools almost transparently. But still:
- Remember to constraint accurately
  - Avoid "playing" with clock (e.g: don't use gating, there are other ways to obtain the same effect)
  - Typically, there can be more than a single clock in a design: watch out for domain boundaries!

23

# Special Signals Handling: Reset - 1

Reset can be either synchronous or asynchronous:

- Synchronous reset takes effect on next clock edge and is treated by synthesizer as any other synchronous signal (timing closure takes care of correctness of reset propagation)
- Asynchronous reset is instantaneous and takes effect regardless of the presence of clock edges

24

## Special Signals Handling: Reset - 2

Since asynchronous resets are not handled by timing closure, special care must be taken with their use for multiple reasons:

- Asynchronous resets should be kept active for a sufficient time to make sure they propagate correctly to all circuits
- De-assertion of an asynchronous reset should be simultaneous across device to avoid state to progress in some areas while some other are still being kept reset
- Typical choice is to use some additional logic to de-assert the asynchronous reset **synchronously** with clock edge

25

## Signal Types in VHDL - 1

- std\_logic : represents a single bit of information.  
It can be "0" or "1" but also hold other states:
  - the most common are "X" for unknown, "U" for unresolved and "Z" for high impedance (they are typically useful when simulating).

```
signal flag : std_logic;  
  
flag <= '0';  
  
26
```

## Signal Types in VHDL - 2

- std\_logic\_vector : represents an array of std\_logic and can be used for buses.

```
signal data_bus : std_logic_vector(7 downto 0);  
  
data_bus <= "01001001";  
  
data_bus <= x"FA";  
  
data_bus(1) <= '1';  
  
data_bus(3 downto 0) <= "0101";  
  
27
```

66

## Signal Types in VHDL - 3

When dealing with arithmetic operations, it's most convenient to use the Unsigned and Signed types (according to the type of data/operation)

- Part of ieee.numeric\_std package
- Sign extension is taken care of automatically
- Anyway VHDL will whine if you don't

## Signal Types in VHDL - 4

- Arrays are custom types that can typically be used to represent blocks of memory

```
type memory16x32 is array(0 to 15) of \  
    std_logic_vector(31 downto 0);  
  
signal memblock : memory16x32;  
  
memblock(12) <= x"5A5A";
```

29

## Signal Casting in VHDL

- VHDL is strongly typed; if assignments have different signal types on the two sides the synthesizer will issue an error.
- Casting from one type to another is necessary

```
signal data_bus : std_logic_vector(3 downto 0);  
signal operand : unsigned(3 downto 0);  
  
operand <= unsigned(data_bus);
```

- Recommendation: use std\_logic/std\_logic\_vector for entity ports

30

## Basic Constructs of VHDL

- Signals and Assignments
- Processes and Variables
- When .. else statement
- Case statement

31

## Signals and Assignments

- Assignations describe the physical connections between signals
- They can be simple wires or describe more complex structures like logic gates or multiplexers/LUTs

```
x_test <= test_in;  
z <= a OR (b AND c) OR d;  
muxout <= in0 when s = '0' else  
in1;  
lut_q <= "1000" when s = "100" else  
"0100" when s = "011" else  
"0010" when s = "010" else  
"0001" when s = "001" else  
"0000";
```

32

## Processes and Variables

- Processes are used to describe in a higher level of abstraction combinatorial or sequential logic. They are activated when a state change happens on any of the signals in their "sensitivity list".
  - Sequential processes: only clock signal (and asynchronous sets/resets, if present)
  - Combinatorial processes: all signals which appear on the right hand side of an assignment

```

process(S, A0, A1, A2, A3)
begin
  case S is
    when "00" => maxout <= A0;
    when "01" => maxout <= A1;
    when "10" => maxout <= A2;
    when others => maxout <= A3;
  end case;
end process;

```

33

## Case statement

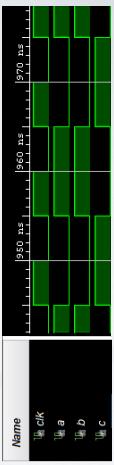
- Used in processes
  - Similar in structure and syntax as in programming languages
  - Typically used for Finite State Machines
  - It's very important to specify values of outputs for all cases (doing the opposite may result in latches).

35

Again, the classic HDL trap!

- Classic programming statements can be used in processes, but this is NOT programming; the statements, in order, are all scheduled to happen at the end of the process

process (clk) is  
begin  
if rising\_edge(clk) then  
    a <= b;  
    b <= c;  
    c <= a;  
    a <= c;  
end if;  
end process;



**A trap to avoid: incomplete case (or if) statements**

- If we remove the default assignments to outputs, the synthesizer won't know what to do with outputs that are unspecified for given cases
  - They will keep their value, generating a register in a sequential process
  - In a combinatorial process, though, this creates unwanted "latches" (which are bad practice for FPGA design, in general).

```

class cur_state {
    when IDLE =>
        i� en_i == '1' then
            next.state <= CNV_H;
        end if;

    when CH_H =>
        i� cnv_H_dashed == '1' then
            max_en_o.state <= CNV_H;
        end if;

        adc_cnv_o <= [1:0];
        adc_cmb_tmr_en <= [1:1];
        chb_tmr_en <= [1:1];
        cnv_I_o >>
        when CNV_I == '1' then
            i� adc_cmb_tmr_dashed == '1' then
                next.state <= CLK_PULSE;
            end if;
            adc_cnv_o <= [1:1];
            when CLK_PULSE == '1' then
                i� adc_cmb_tmr_en == '1' then
                    adc_cmb_tmr_en <= [1:1];
                end if;
                adc_cmb_tmr_en <= [1:1];
                adc_clk_tick_timer_en <= '1';
                adc_clk_tick_timer_psc <= '0';
            when CLK_PULSE == '0' then
                i� en_i == '1' then
                    next.state <= CNV_H;
                end if;
                data_en_o <= '1';
            when CNV_H == '1' then
                max_en_o <= IDLE;
            end case;
}

```

36

# Take Home Messages

- HDLs are not programming languages. They are a tool to describe digital logic circuits.
- Watch Out for Non-Synthesizable Code and for the traps of a “programmer mindset”
- Simulation is **essential!** Writing good testbenches is as important as writing good logic.
- Handle adequately resets and clock signals

37

# Big Data Technologies and Physics Analysis with Apache Spark

Inverted CERN School of Computing 2019

4-6 March 2019

CERN OpenLab

Evangelos Motesnitsalis

## Contents

1. Introduction to Big Data
2. Big Data Systems Architecture:
  - Architecture Principles
  - Distributed Filesystem
  - Cluster Manager
  - Processing Framework
3. Popular Big Data Frameworks:
  - Apache Hadoop
    - Hadoop Distributed Filesystem (HDFS)
    - Apache YARN
    - Hadoop MapReduce
  - Apache Spark
4. Standard Physics Analysis Procedures
5. Big Data Tools and Approaches for HEP
6. Example Workloads
7. Projects beyond Physics Analysis
8. Conclusions

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

## Learning Goals

	Important big data concepts
	Main architecture characteristics of big data technologies
	Popular big data frameworks such as Apache Hadoop and Apache Spark
	Example of physics analysis with Apache Spark

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

CERN OpenLab

Evangelos Motesnitsalis

## I am...

-  Technical Coordinator for 2 months  
(still Data Engineer at heart)
-  Research Fellow in 2017 – today  
Technical Student in 2014 – 2015  
Summer Student in 2013
-  5+ years of Big Data experience

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

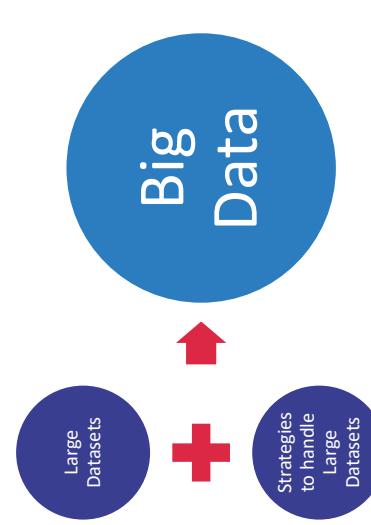
5

## Introduction to Big Data

6

## Introduction to Big Data

A bit of history...



Evangelos Motesnitsalis - inverted CERN School of Computing 2019

7

## Introduction to Big Data

8



Evangelos Motesnitsalis - inverted CERN School of Computing 2019

9

# Big Data Systems Architecture

72

## Architecture Overview

Top Level Abstractions

Distributed Processing Frameworks

Cluster Resource Manager

Distributed Filesystem



CERN OpenLab

9

9

## Architecture Principles

Resource Pooling: combining storage space, CPU and memory for different use cases and frameworks



Data Persistence: high replication factor and automatic restoration



High Availability: fault tolerance for source code execution and hardware components



Data Ingestion: ability to import raw data, RDBMS data, etc.



Parallelization: follow programming patterns that allow batch processing



Scalability: horizontal with new machines, vertical with bigger machines



Evangelos Motesnitsalis - inverted CERN School of Computing 2019

11

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

12

## Distributed File System

A software framework that allows users to access and process distributed data



Heterogeneity and Scalability



Usually centralized yet highly available metadata



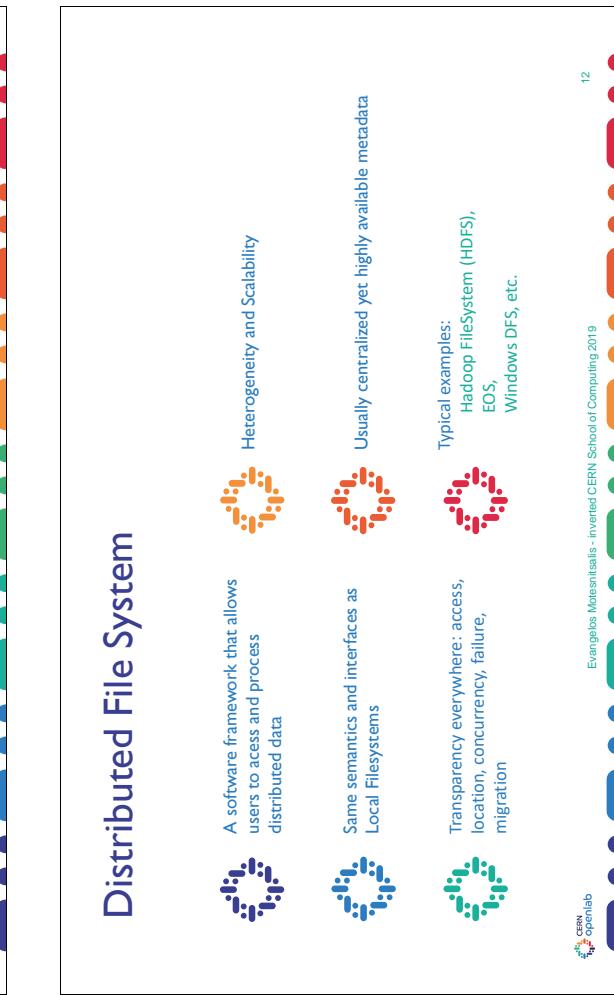
Typical examples:



Hadoop FileSystem (HDFS), EOS, Windows DFS, etc.

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

12



## Cluster Manager

	A software framework that runs distributively on cluster nodes
	Multiple software components, multiple execution locations
	Resource allocation and service configurations

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

13

## Distributed Processing Framework

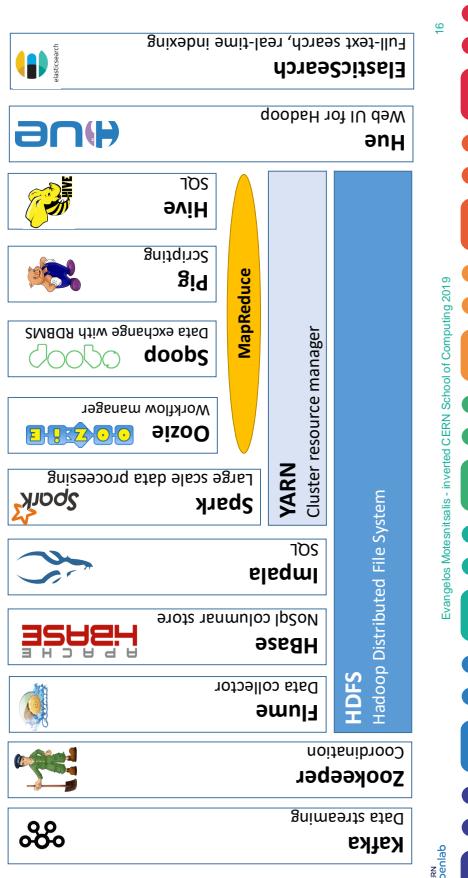
	A software framework that allows distributed computations
	Usually follows specific programming models (e.g. MapReduce)
	Code is automatically deployed in multiple locations

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

14

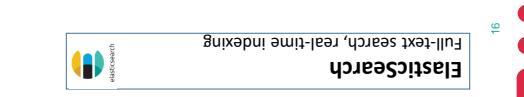
# Big Data Frameworks

## The Big Data Ecosystem



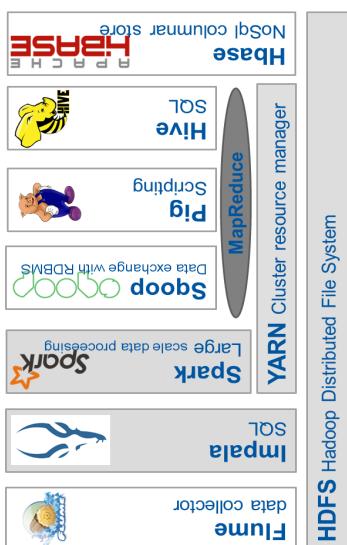
Evangelos Motesnitsalis - inverted CERN School of Computing 2019

15



16

## The Hadoop Ecosystem



17

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

## Apache Hadoop

## Apache Hadoop

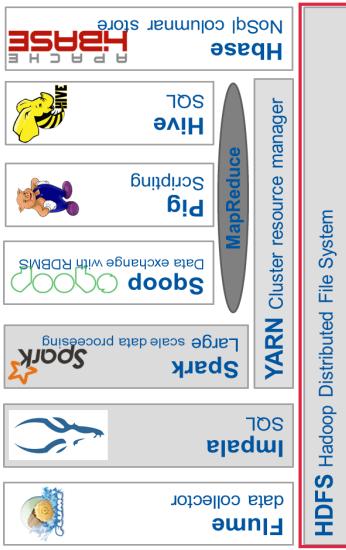
A Framework for Large-scale Distributed Data Processing



Evangelos Motesnitsalis - inverted CERN School of Computing 2019

19

## The Hadoop Ecosystem

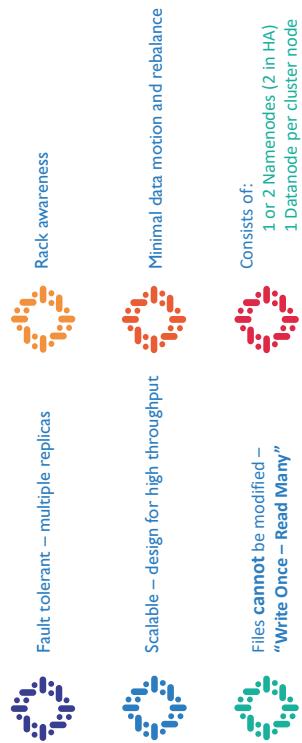


Evangelos Motesnitsalis - inverted CERN School of Computing 2019

20

## Hadoop Distributed File System

### The Filesystem of Hadoop

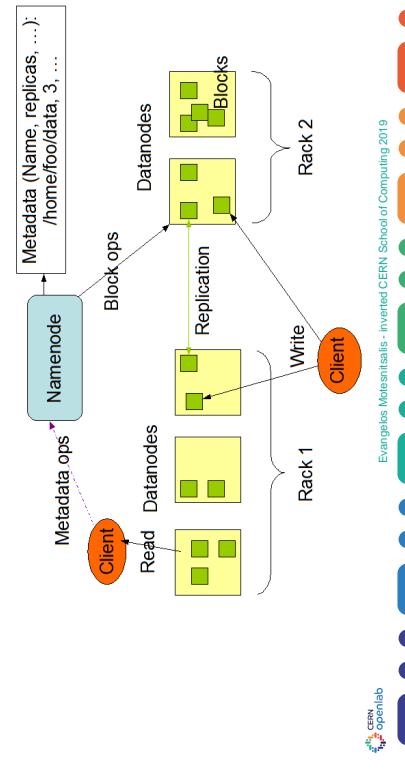


Evangelos Motesnitsalis - inverted CERN School of Computing 2019

21

## Hadoop Distributed File System

### The Filesystem of Hadoop



Evangelos Motesnitsalis - inverted CERN School of Computing 2019

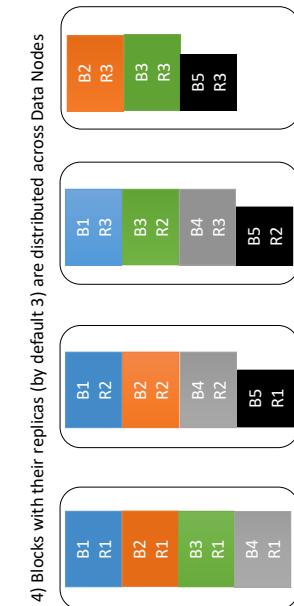
22

## Hadoop Distributed File System

### 1)One 1176 MB File to be stored on HDFS

B1 256MB	B2 256MB	B3 256MB	B4 256MB	B5 152MB
-------------	-------------	-------------	-------------	-------------

2) Splitting into 256MB blocks



DataNode1 DataNode2 DataNode3 DataNode4

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

23

## Hadoop Distributed File System

### Interacting with HDFS

```
hdfs dfs -ls          #listing home dir
hdfs dfs -ls /user   #listing user dir...
hdfs dfs -du -h /user #space used
hdfs dfs -mkdir newdir #creating dir
hdfs dfs -put myfile.csv #storing a file on HDFS
hdfs dfs -get myfile.csv #getting a file fr HDFS
```

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

24

## The Hadoop Ecosystem

**HDFS**

**YARN** Cluster resource manager

**MapReduce**

**Impala** SQL

**Flink** data collector

**Spark** large scale data processing

**Sqoop** Data exchange with RDBMS

**Pig** scripting

**Hive** NoSQL columnar store

**Hbase**

## Apache Hadoop YARN

Yet Another Resource Negotiator

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

## Hadoop Distributed File System

Data Flow in HDFS

1. Data ingestion

2a. Visualize

2b. Low latency store

3. Publish

Shell/Notebook

Graphical UI

NameNode

DataNode

DataNode

DataNode

DataNode

DATA SOURCE

## Apache Hadoop YARN

Yet Another Resource Negotiator

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

## Apache Hadoop YARN

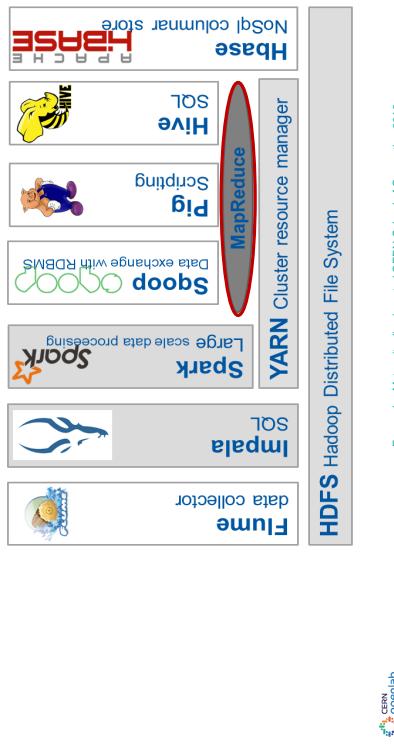
Interacting with YARN

```
yarn application -list          #listing apps submitted  
yarn application -status <id>    #details about app  
yarn application -kill <id>      #kill running app
```

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

29

## The Hadoop Ecosystem



Evangelos Motesnitsalis - inverted CERN School of Computing 2019

30

## Hadoop MapReduce

The First Batch Processing Framework



Optimized for local data access



Good for huge data sets and offline analysis  
but does not fit every use case



Time consuming and not interactive

CERN OpenLab

## Hadoop MapReduce

Hello World – aka « Wordcount »

```
//REDUCER method body  
reduce(String key, Iterator values):  
    // key: word  
    // values: a list of counts  
    for each v in values:  
        result += ParseInt(v);  
    Emit(result)
```

---

```
//MAP method body  
map(String key, String value)  
    // key: document name  
    // value: document contents  
    for each word w in value  
        EmitIntermediate(w, "1")
```

CERN OpenLab

## Big Data Technologies and Physics Analysis with Apache Spark

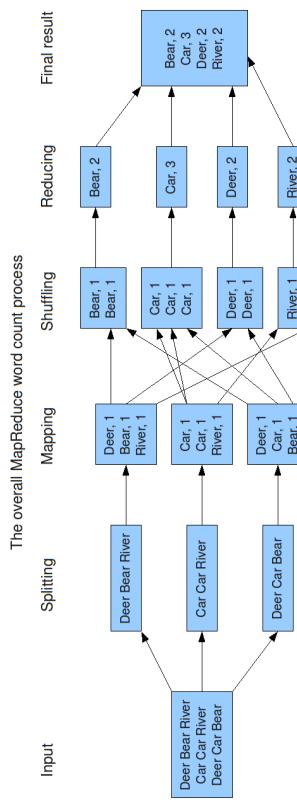
Evangelos Motesnitsalis - inverted CERN School of Computing 2019

31

32

## Hadoop MapReduce

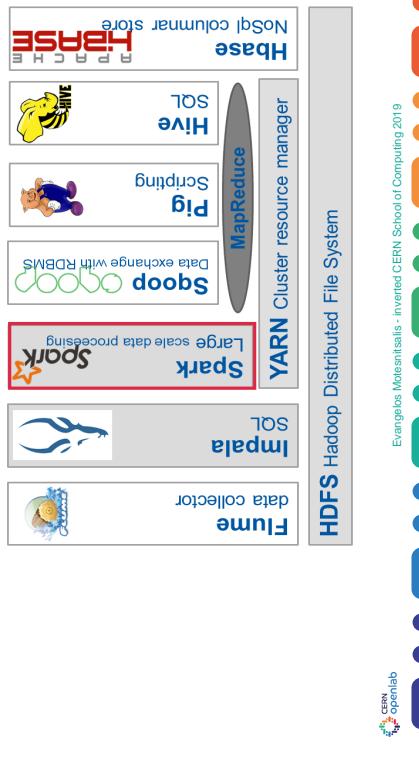
Hello World – aka « Wordcount »



Evangelos Motesnitsalis - inverted CERN School of Computing 2019

33

## The Hadoop Ecosystem



Evangelos Motesnitsalis - inverted CERN School of Computing 2019

34

## Apache Spark



### Overview

Compatible with multiple cluster managers:

- Apache YARN
- Apache Mesos
- Kubernetes
- Standalone



Apache Spark is an open source cluster computing framework

Brought fast, iterative, near real-time processing with no strict programming model – everything that MapReduce lacked.



Multiple File Formats and Filesystem Compatibility



Consists of multiple components:

- Spark SQL
- Spark MLlib
- Spark Graph
- Spark Structured Streaming



35

36

78

## Apache Spark

### Basic Concepts of Apache Spark

 RDDs: Resilient Distributed Dataset, the basic abstraction of Spark is collection of partitioned data with primitive values

 Spark supports complex processing patterns based on DAG

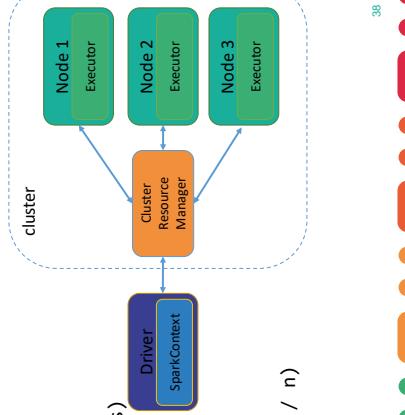
 Directed Acyclic Graph: A finite directed graph with no directed cycles

 Staged Data are kept **in-memory**

37 CERN OpenLab

## Apache Spark

### Driver and Executors



```

import scala.math.random
val slices = 3
val n = 100000 * slices
val rdd = sc.parallelize(1 to n, slices)
val sample = rdd.map { i =>
    val x = random
    val y = random
    if (x*x + y*y < 1) 1 else 0
}
val count = sample.reduce(_ + _)
println(s"Pi is roughly ${count / n}")
  
```

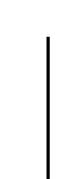
38 CERN OpenLab

## Apache Spark

### Hello World – aka « Wordcount »

 RDD input: [w1 w2 w3 w1 w3 w3]

 flatMap(lambda x: x.split(' ')) → RDD words: [w1, w2, w3, w1, w1, w3, w1, w3, w3]

 map(lambda x: (x, 1)) → RDD words with initial counts: [(w1,1), (w2,1), (w3,1), (w1,1), (w1,1), (w3,1), (w3,1)]

 reduceByKey(lambda x,y:x+y) → RDD words with final counts: [(w1,4), (w2,1), (w3,4)]

39 Evangelos Motesnitsalis - inverted CERN School of Computing 2019 CERN OpenLab

## Apache Spark

### Hello World – aka « Wordcount »

```

text_file = sc.textFile("/user/emotes/datasets/")
counts = text_file.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("/user/emotes/outputFolder/")
  
```

---

## Apache Spark

### Hello World – aka « Wordcount »

```

#defining dataframe with schema from parquet files
val df = spark.read.parquet("/user/emotes/datasets/")
#counting the number of pre-filtered rows with DF API
df.filter($"llusername".contains("emo-test")).count
#counting the number of pre-filtered rows with SQL
df.registerTempTable("my_table")
spark.sql("SELECT count(*) FROM my_table where llusername like '%emo-test%'").show
  
```

40 Evangelos Motesnitsalis - inverted CERN School of Computing 2019 CERN OpenLab

## Standard Physics Analysis Procedures

### WLCG Worldwide LHC Computing Grid



43

## HEP Data Processing

Physics Analysis is typically done with the ROOT Framework which uses physics data that are saved in ROOT format files.

At CERN these files are stored within the EOS Storage Service.

## Standard Physics Analysis Procedures

### WLCG Worldwide LHC Computing Grid



43

**EOS Service**  
A disk-based, low-latency storage service with a highly-scalable hierarchical namespace, which enables data access through the XRootD protocol.



CERN Openlab

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

42

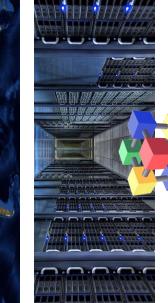
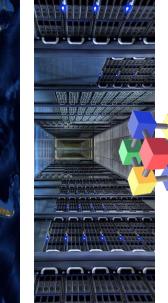


Data Analysis Framework

42

## WLCG

### Worldwide LHC Computing Grid



Evangelos Motesnitsalis - inverted CERN School of Computing 2019

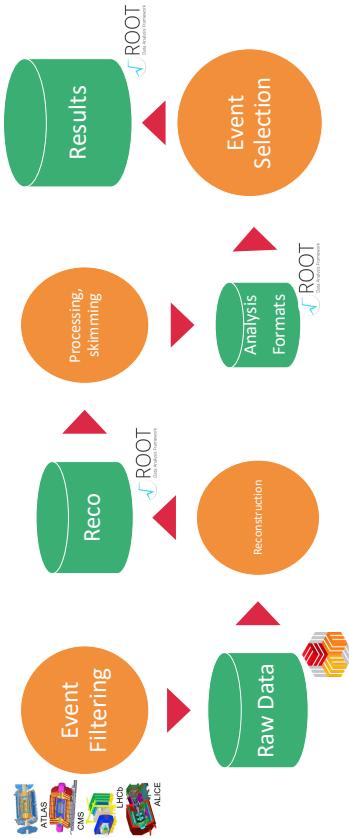
Evangelos Motesnitsalis - inverted CERN School of Computing 2019

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

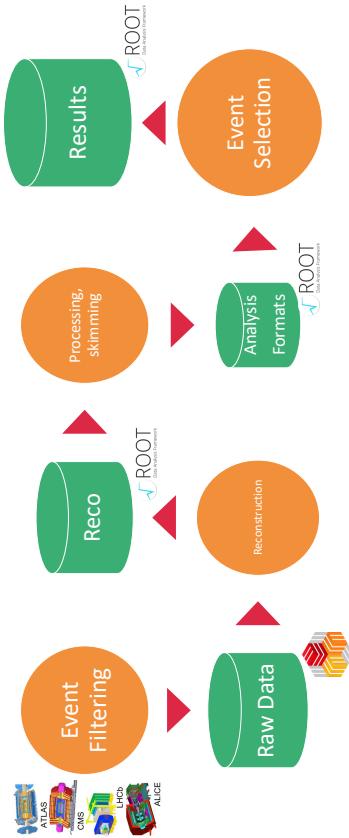
## LHC Data Flow at CERN



Evangelos Motesnitsalis - inverted CERN School of Computing 2019

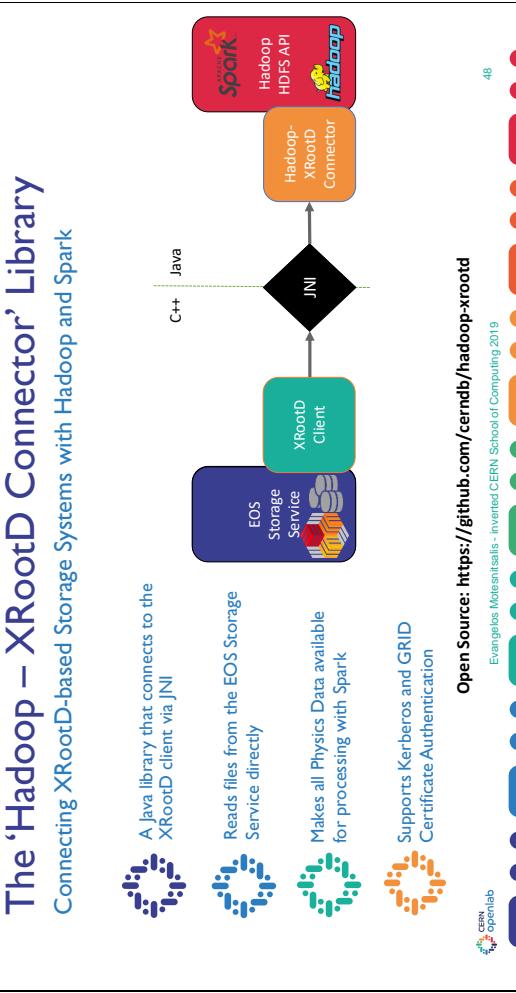
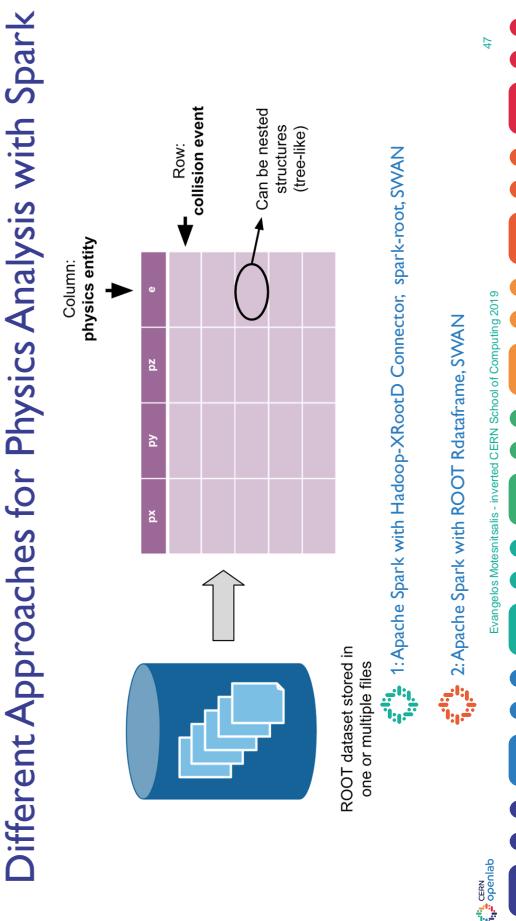
42

## LHC Data Flow at CERN



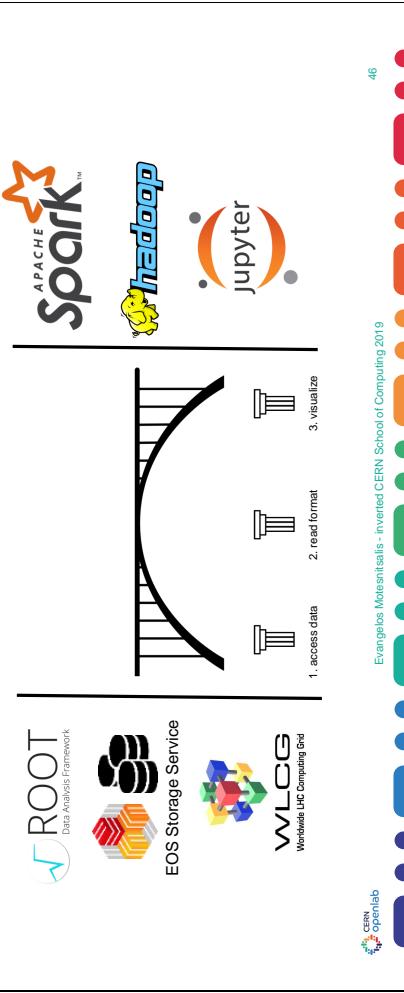
Evangelos Motesnitsalis - inverted CERN School of Computing 2019

42



## Bridging the Gap

Physics Analysis is typically done with the ROOT Framework which uses physics data that are saved in ROOT format files. At CERN these files are stored within the EOS Storage Service.



## Big Data Tools for High Energy Physics

## ROOT RDataFrame

The diagram illustrates the integration of ROOT and Apache Spark. It shows a central **Spark Backend** (yellow box) interacting with a **Driver** (blue box) and an **Executor** (blue box). The **Executor** contains a **Mapper** and a **Reducer**. The **Spark Cluster** (represented by a cloud icon) manages tasks. Arrows indicate data flow from **EOS** to the **Spark Backend**, and from the **Spark Backend** to the **Executor**. A legend at the bottom right defines icons for EOS, Spark Dataframes, CERN Openlab, and a presentation slide.

## Overview

The diagram shows the **SWAN** service (blue cloud icon) integrated with **Apache Spark** (orange star icon). **Apache Spark** runs on **MESOS** (blue hexagon icon), **kubernetes** (red square icon), or **openstack** (black cloud icon). **Apache Spark** interacts with **EOS Storage Service** (grey cylinder icon) via a circular arrow. **Apache Spark** also connects to **Jupyter** notebooks (orange icon) and **HDFS** (red cube icon). A legend at the bottom right defines icons for SWAN, Jupyter, HDFS, MESOS, Kubernetes, OpenStack, and a presentation slide.

## The 'Spark – Root' Library

The diagram shows the **Spark – Root** library (blue logo) integrated with **ROOT** (orange logo). It highlights that **Spark** can read **ROOT** TTrees and infer their schema. **Root files** are imported into **Spark** DataFrames/Datasets/RDDs. The **Open Source** link is <https://github.com/diana-hep/spark-root/>. A legend at the bottom right defines icons for Spark, ROOT, CERN Openlab, and a presentation slide.

## SWAN Service and Spark Integration

The diagram shows the **SWAN** service (blue cloud icon) integrated with **Apache Spark** (orange star icon). It highlights features like web-based interactive analysis using PySpark in the cloud, user-friendly environments for collaboration, direct access to EOS and HDFS, and full integration with IT Spark and Hadoop Clusters. The URL <https://swan.web.cern.ch/> is provided. A legend at the bottom right defines icons for SWAN, PySpark, HDFS, and a presentation slide.

# Physics Analysis with Apache Spark

83

## CMS Data Reduction and Analysis Facility

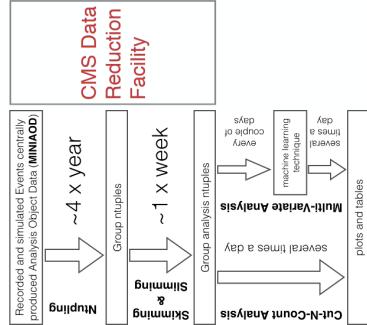
Performing Physics Analysis and Data Reduction with Apache Spark

-  Main goal was to be able to reduce 1 PB of data in 5 hours or less
-  It offers an alternative for 'ad-hoc' data reduction for each research group
-  Bridge the gap between High Energy Physics and Big Data communities
-  Investigate new ways to analyse physics data and improve resource utilization and time-to-physics
-  Data Reduction refers to event selection and feature preparation based on potentially complicated queries
-  We now have fully functioning Analysis and Reduction examples tested over CMS Open Data

# The Use Case of the CMS Data Reduction Facility

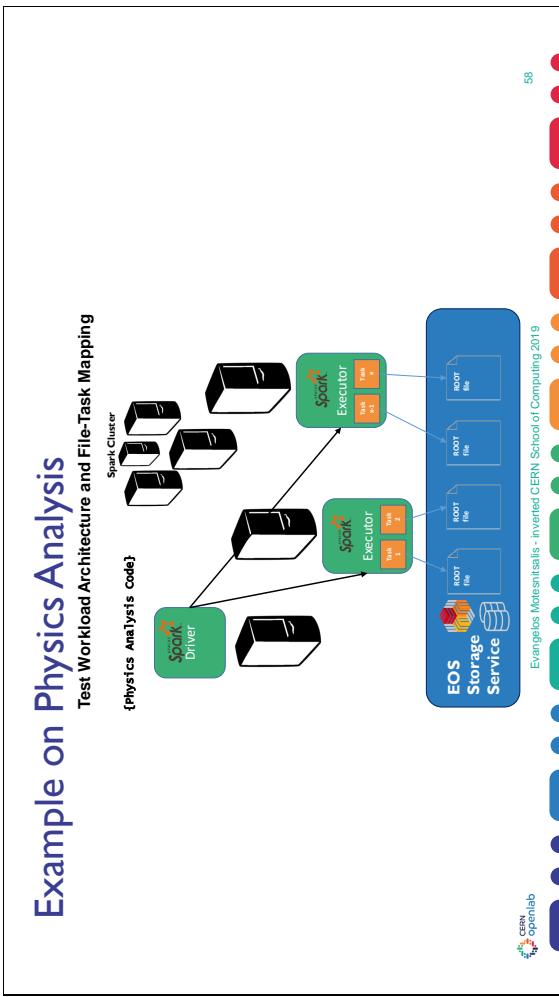
54

## CMS Data Reduction and Analysis Facility



# Examples

## Example on Physics Analysis



## Example on Physics Analysis with SWAN

Spark clusters connection

You are going to connect to Hadoop

Add a new option

A Apache Spark

Selected configuration

- spark.yarn.access.hadoopFileInputFormat
- spark.driver.memory
- TG
- spark.executor.instances

Simple example with Spark

The screenshot illustrates the use of Spark in SWAN.

The current build allows to connect directly connections on a local memory datasets.

Moreover, it's now easier to use Spark on memory datasets.

Import the necessary modules

The system module is available to perform the necessary imports.

In [1]:

```
from pyspark import SparkContext
```

Integration of SWAN with Spark clusters

This notebook demonstrates how to quickly make a basic connection to a cluster to effect modifications to an external Spark cluster. In Spark version we are going to work with 2.1. In what we are going to do is connect to the Spark cluster as previously described in the SWAN web form.

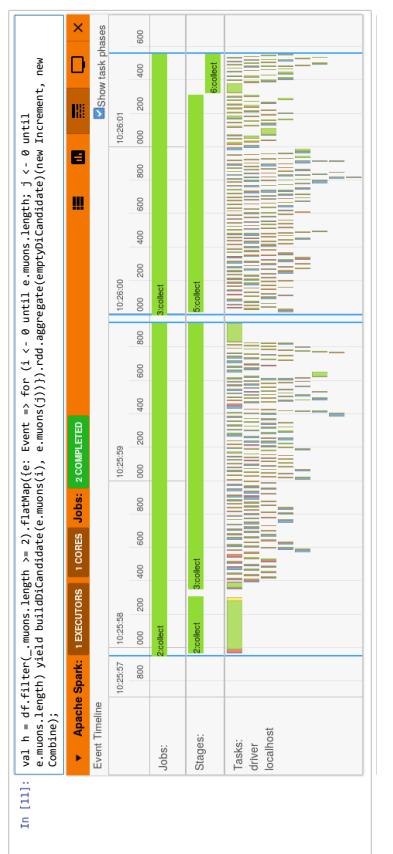
Step 1 - Access the necessary credentials to access the Spark cluster

```
import requests
from bs4 import BeautifulSoup
url = "http://spark.yarn.swan.cern.ch:8088/proxy/1/cluster"
r = requests.get(url)
print(BeautifulSoup(r.text, "lxml"))
if r.status_code != 200:
    raise ValueError("Error creating connection, return code: %s" % r.status_code)
```

CERN Openlab

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

## Example on Physics Analysis with SWAN



## Example on Physics Analysis with SWAN

Apache Spark		1 EXECUTORS		4 CORES		Jobs		2 COMPLETED		Tasks		Submission Time		Duration	
Job ID	Job Name	Status	Completed	Stages	Status	Completed	Tasks	Completed	Tasks	Completed	Tasks	Completed	Submission Time	Duration	
2	reduce	COMPLETED	2/2				48 / 48						5 minutes ago	3s	
5	reduce	COMPLETED					32 / 32						Submission Time	Duration	
4	coalesce	COMPLETED					15 / 15						5 minutes ago	0s	
3	foreach	COMPLETED	1/1 (1 skipped)				32 / 32						Submission Time	1m20s	
6	coalesce	FAILED					8 / 10						Unknown	-	
7	foreach	FAILED					31 / 32						5 minutes ago	1m20s	

FEDERAL BUREAU OF INVESTIGATION - 2010

8

Final Result

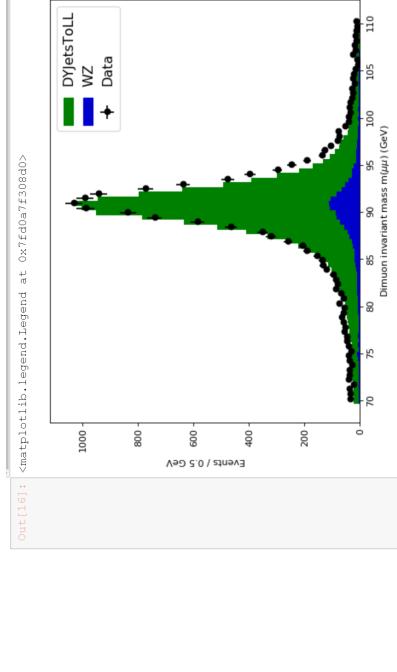
OK, OK, one with better graphics



Evangelos Motesnitsalis

Big Data Technologies and Physics Analysis with Apache Spark

## Final Result



That is what we will do in the exercises ☺

86

## Projects beyond Physics Analysis

65

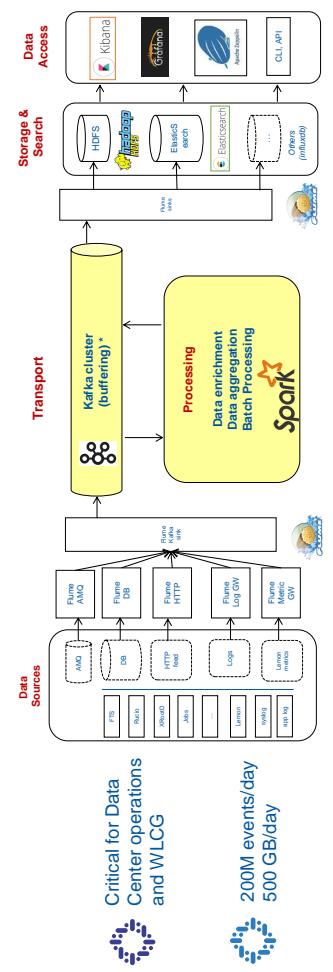
## Next Accelerator Logging Service (NXCALS)



Credits: BE-CO-DS  
Evangelos Motesnitsalis - inverted CERN School of Computing 2019

67

## Data Center and WLCG Monitoring Systems



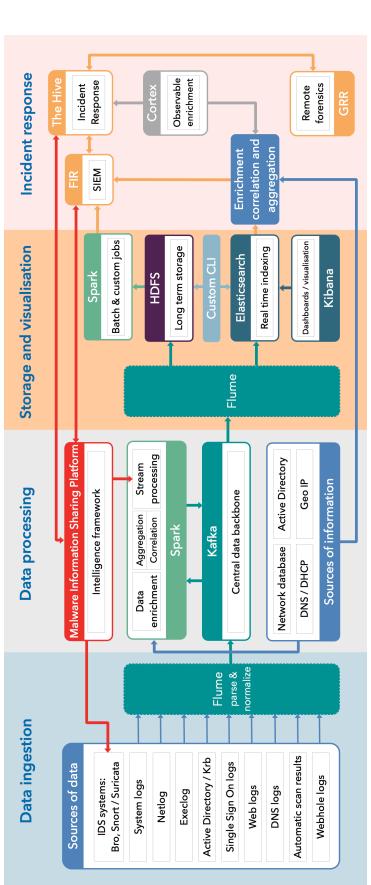
Credits: IT-CM-MMM  
Evangelos Motesnitsalis - inverted CERN School of Computing 2019

68

## Data Center and WLCG Monitoring Systems

66

## Computer Security Intrusion Detection



69

70

71

72

## Conclusions

### Conclusions

There is a broad ecosystem of Big Data Frameworks, most of which share the same architecture principles such as resource pooling, high availability, fault tolerance, etc.

Popular Big Data Frameworks such as Apache Spark show great potential in bridging the gap between the High Energy Physics community and the Big Data community.

There are now available tools and services to use these big data technologies in order to perform analytics on physics, infrastructure, and accelerator data.

Evangelos Motesnitsalis - inverted CERN School of Computing 2019

71

72

73

## Acknowledgements

My mentors, Sebastian Lopienksi and Enric Tejedor Saverda

Colleagues at the CERN Hadoop, Spark, and streaming services who kindly helped with material and feedback

CMS members of the Big Data Reduction Facility, DIANA/HEP Fermilab



The CSC Team, especially joelma and Nikos



CERN Openlab

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

87

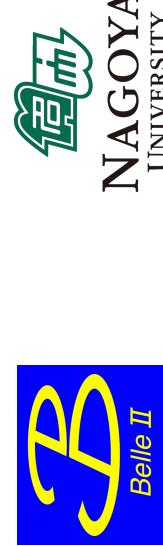
# Thank you

[emotes@cern.ch](mailto:emotes@cern.ch)



73

CERN  
Openlab



NAGOYA  
UNIVERSITY



Kobayashi-Maskawa Institute  
for the Origin of Particles and the Universe

## Global track finding algorithms

Dmitrii Neverov

Inverted CERN School of Computing  
4-7<sup>th</sup> March 2019

### Who am I

- ▲ Belle II experiment, Japan
  - e+e- collider
  - B physics
- ▲ Search for magnetic monopoles
  - Non-helical tracking
- ▲ “Physics” background
  -

Dmitrii Neverov - Global track finding

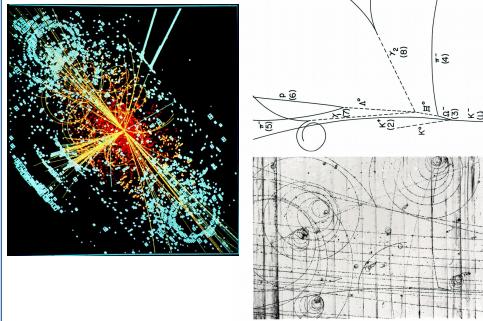
2

Introduction

4

### Physics

- ▲ Fundamental laws govern collisions
- ▲ Final state particles are all we get
- ▲ Charged particles produce tracks
  - Set of points (x,y,z) in space
- ▲ Disentangle all hits into subsets
- ▲ Fit tracks



Dmitrii Neverov - Global track finding

Dmitrii Neverov

Global track finding algorithms

## Track finding

### Challenges

1. Track multiplicity (pileup, low pt)
2. Fake hits
3. Measurement errors (resolution, alignment)
4. Material effects / inhomogeneous magnetic field

Dmitrii Neverov - Global track finding

## 5

## Track finding

### Challenges

1. Track multiplicity (pileup, low pt)
2. Fake hits
3. Measurement errors (resolution, alignment)
4. Material effects / inhomogeneous magnetic field

- Local track finding
  - Hit by hit
  - Segment by segment
- Global track finding
  - All hits at once

Dmitrii Neverov - Global track finding

## 6

## Track finding

### Challenges

1. Track multiplicity (pileup, low pt)
2. Fake hits
3. Measurement errors (resolution, alignment)
4. Material effects / inhomogeneous magnetic field

- Local track finding
  - Hit by hit
  - Segment by segment
- Global track finding
  - All hits at once

Dmitrii Neverov - Global track finding

## Track seeding and following

- ▶ Seed track window
- ▶ Find hits compatible with the track
- ▶ Update and advance to new layer
- ▶ Smart filters

▶ Roughly  $\sim n^{2-3}$

Dmitrii Neverov - Global track finding

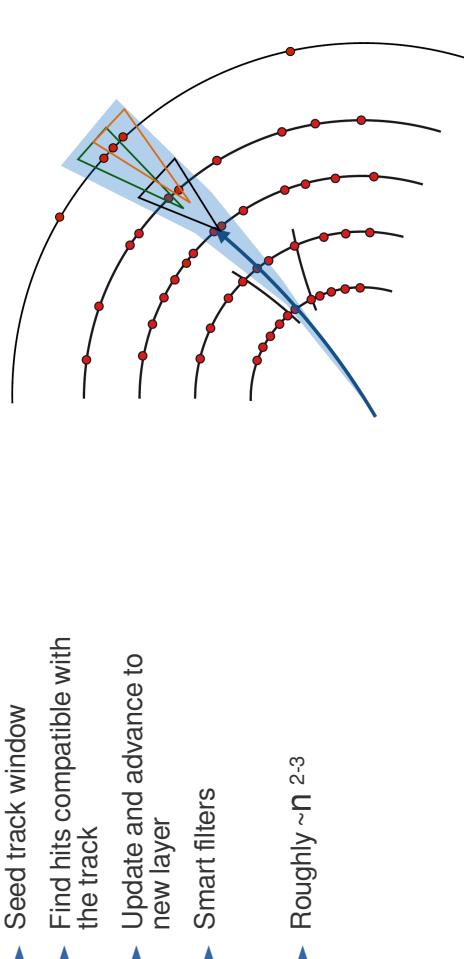
## Cellular automaton

- ▶ Build clusters
- ▶ Build segments
- ▶ Build tracks

▶ Roughly  $\sim n^2$

Dmitrii Neverov - Global track finding

9

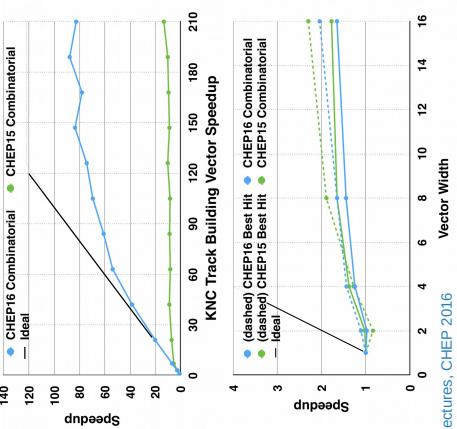


## Speedup

- ▶ Can be parallelised well

11

### KNC Track Building Parallel Speedup



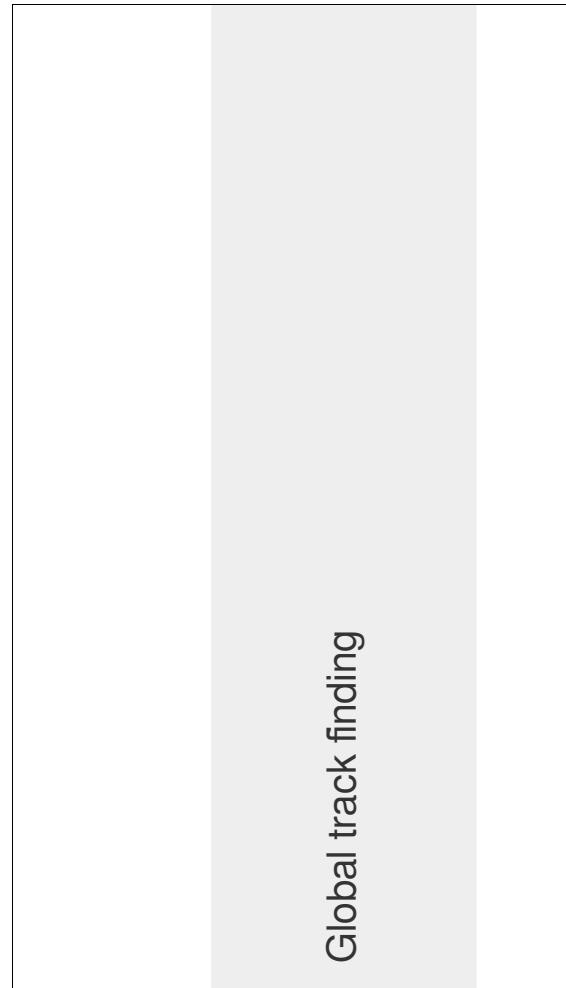
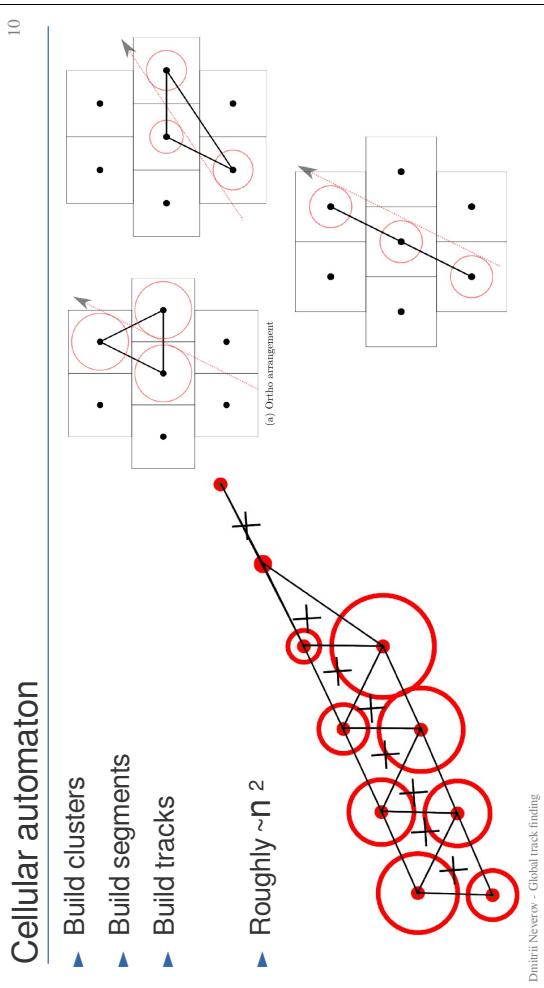
Dmitrii Neverov - Global track finding

11

CHeP2015 Reference Platforms		Preliminary Results	
Xeon E5-2620 Sandy Bridge (SNB)	Xeon Phi 7120P Knights Corner (KNC)	Xeon Phi 7230 Knights Landing (KNL)	Tesla K40
6x2x2	6x4	64x4	2580 CUDA cores
Logical Cores	124 GHz	1.3 GHz	875 MHz
Clock rate	2.5 GHz	1208	1430
GFLOPS	256 bits	512 bits	32 thread warp
SIMD width	-64-384 GB	16 GB	12 GB
Memory	42.6 GB/s	352 GB/s	475 & 90 GB/s
Bandwidth			288 GB/s

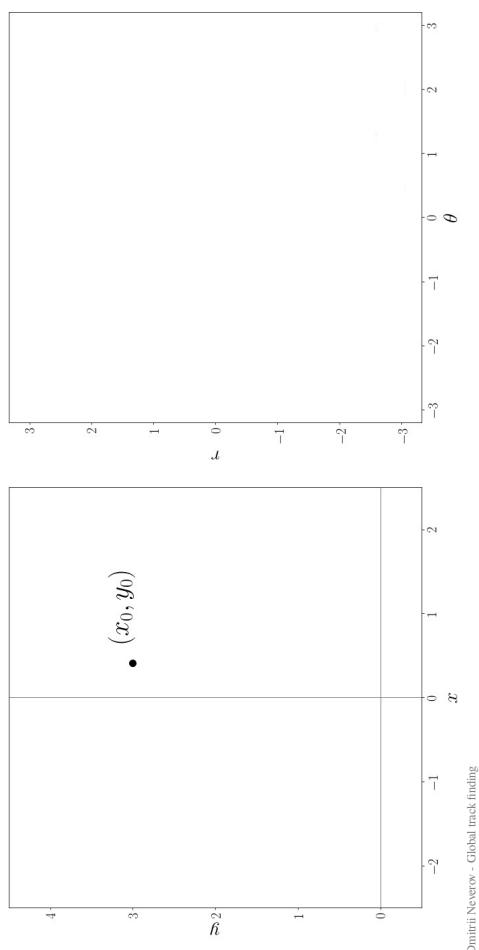
Kalman filter tracking on parallel architectures, CHeP 2016

## Global track finding

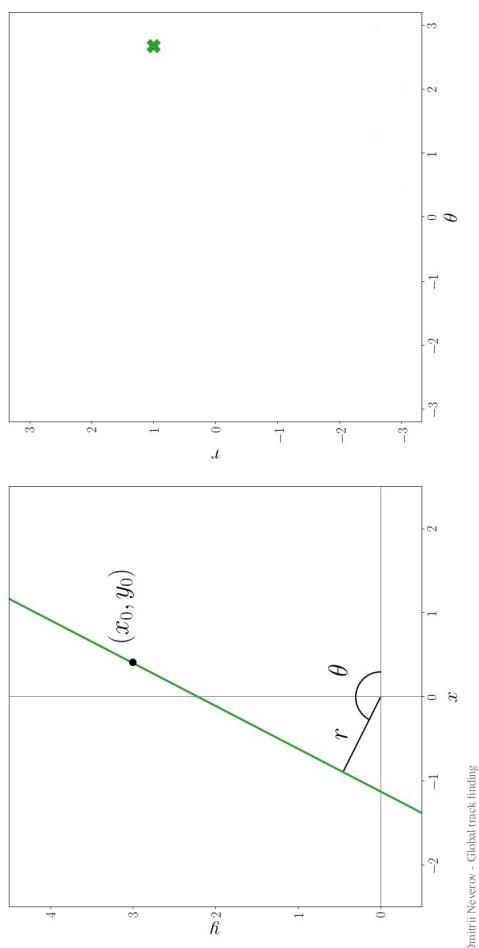


## Hough transform

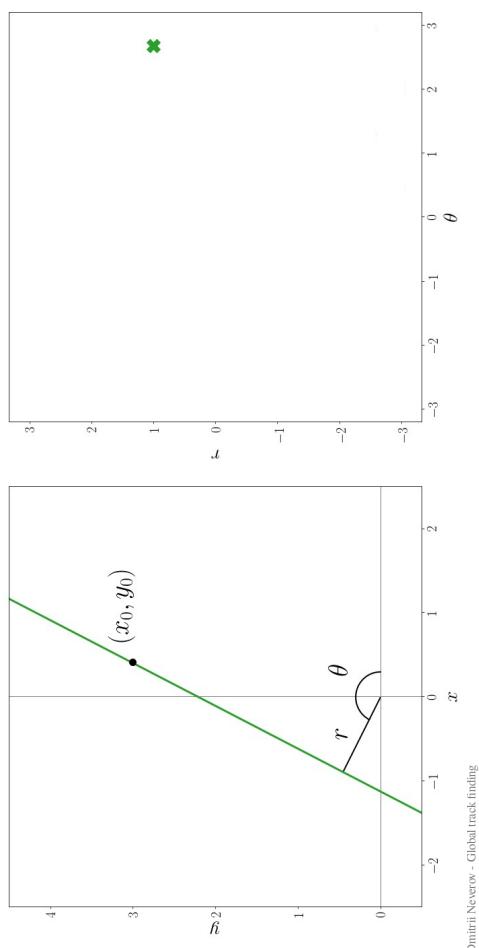
13 Dmitrii Neverov - Global track finding



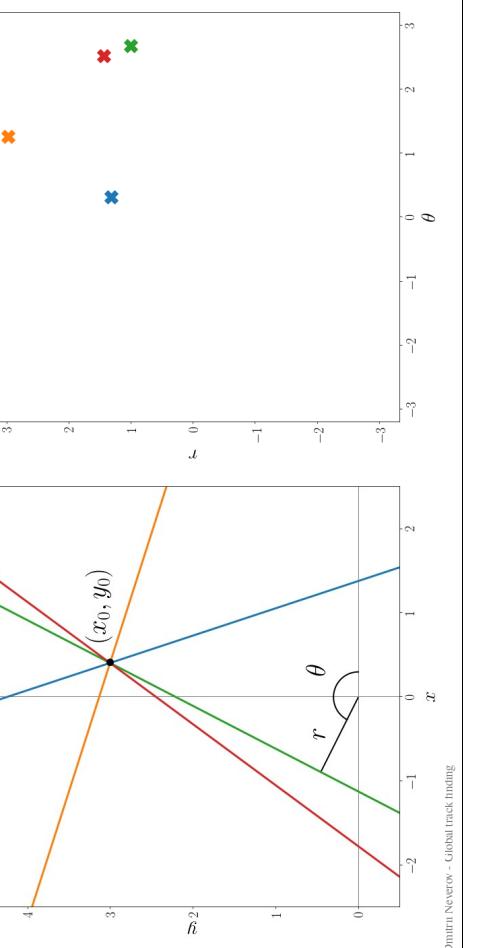
14 Dmitrii Neverov - Global track finding



15 Dmitrii Neverov - Global track finding

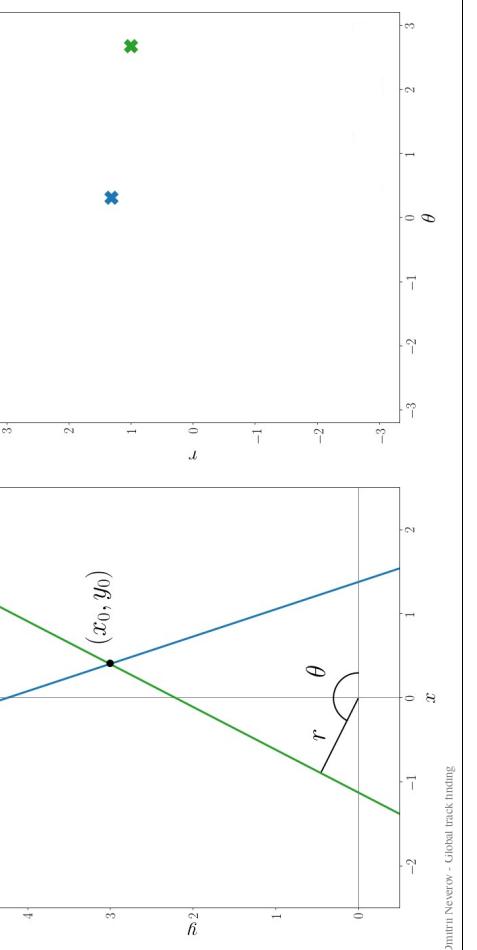


16 Dmitrii Neverov - Global track finding

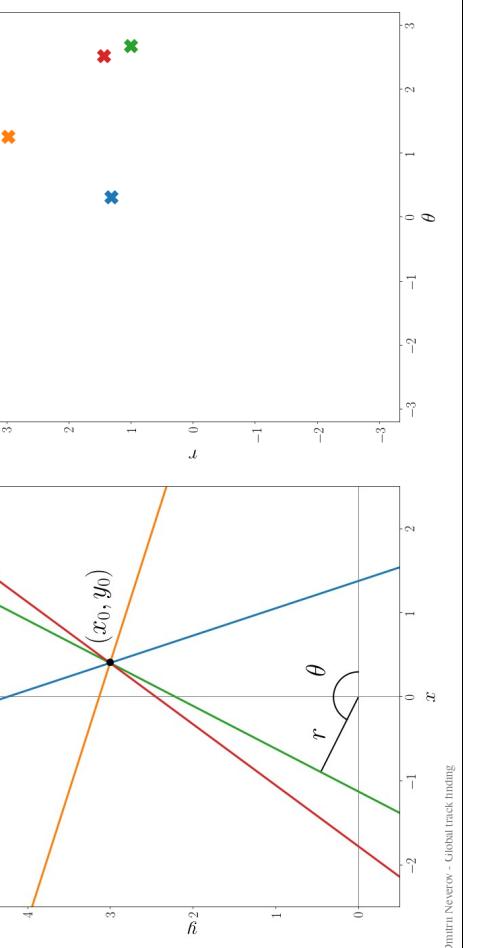


## Hough transform

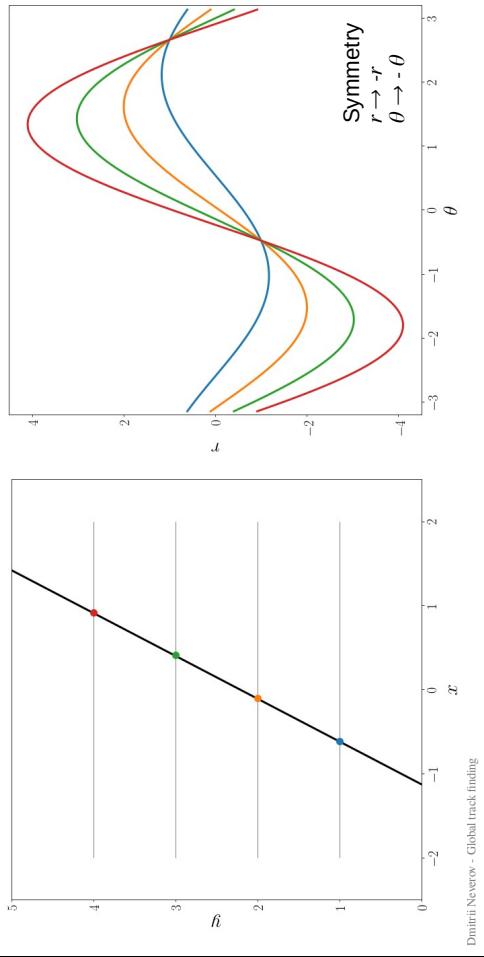
15 Dmitrii Neverov - Global track finding



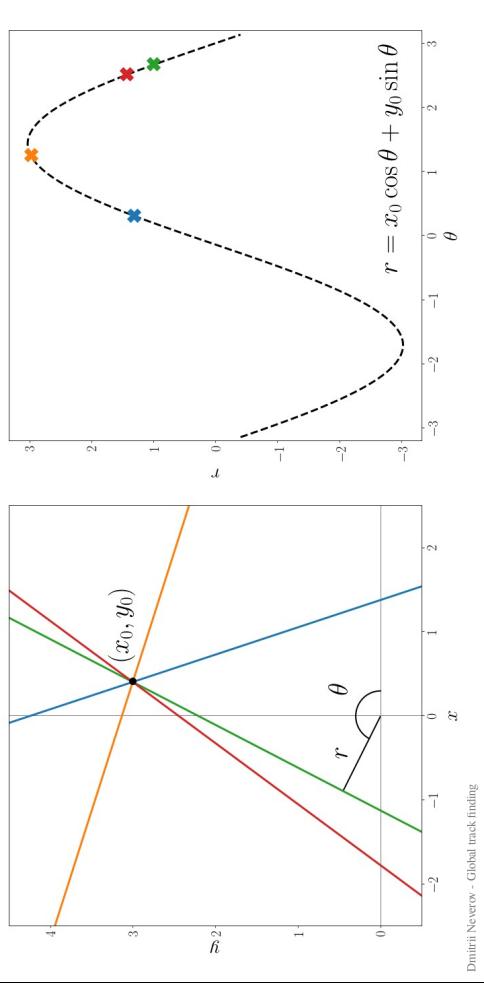
16 Dmitrii Neverov - Global track finding



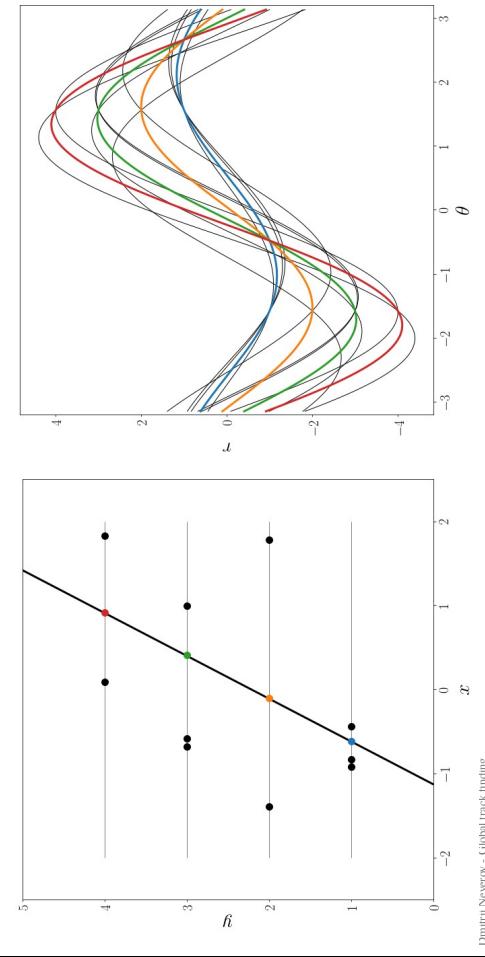
18

Hough transform: finding a line

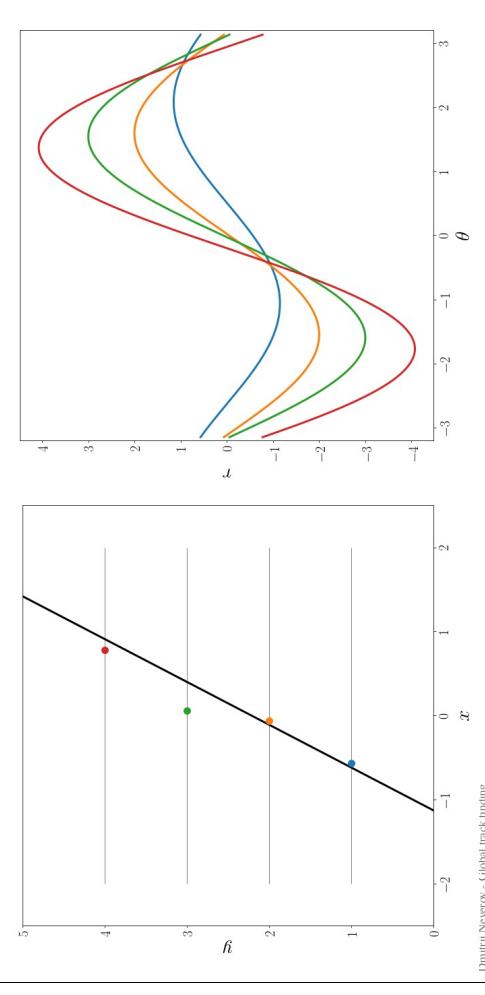
17

Hough transform

20

Hough transform: noise

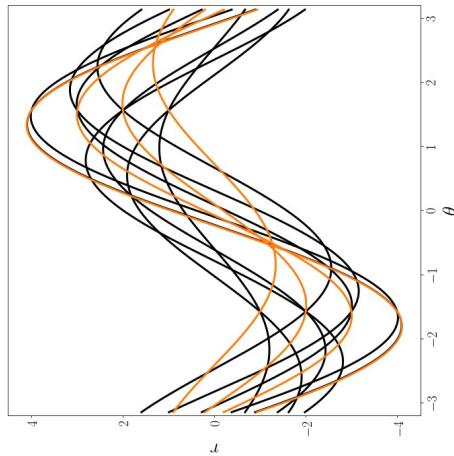
19

Hough transform: errors

21

## Hough transform summary

- ▲ Hits  $\leftrightarrow$  curves in hough space
- ▲ Noise  $\leftrightarrow$  extra curves
- ▲ Errors  $\leftrightarrow$  spread intersections

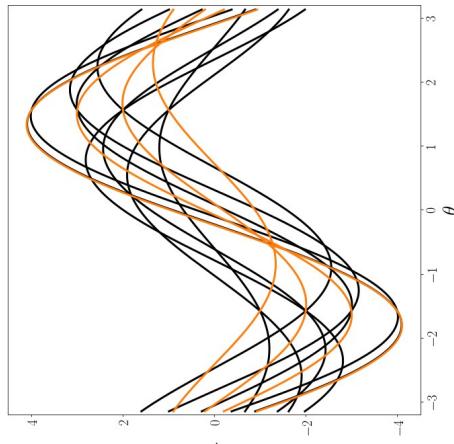


Dmitriii Neverov - Global track finding

22

## Hough transform summary

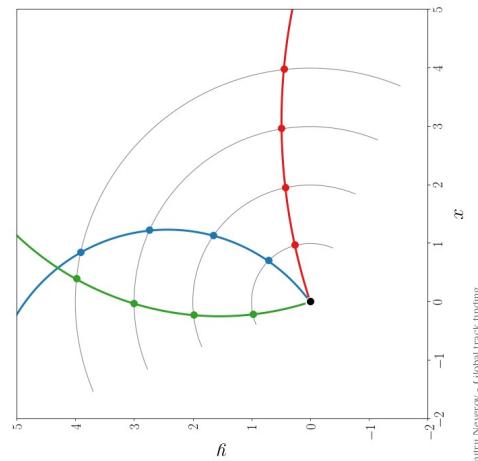
- ▲ Hits  $\leftrightarrow$  curves in hough space
- ▲ Noise  $\leftrightarrow$  extra curves
- ▲ Errors  $\leftrightarrow$  spread intersections
- ▲ Pattern finding translated to finding most dense regions in hough space
- ▲ Basic idea behind global track finders
- ▲ Can be extended



Dmitriii Neverov - Global track finding

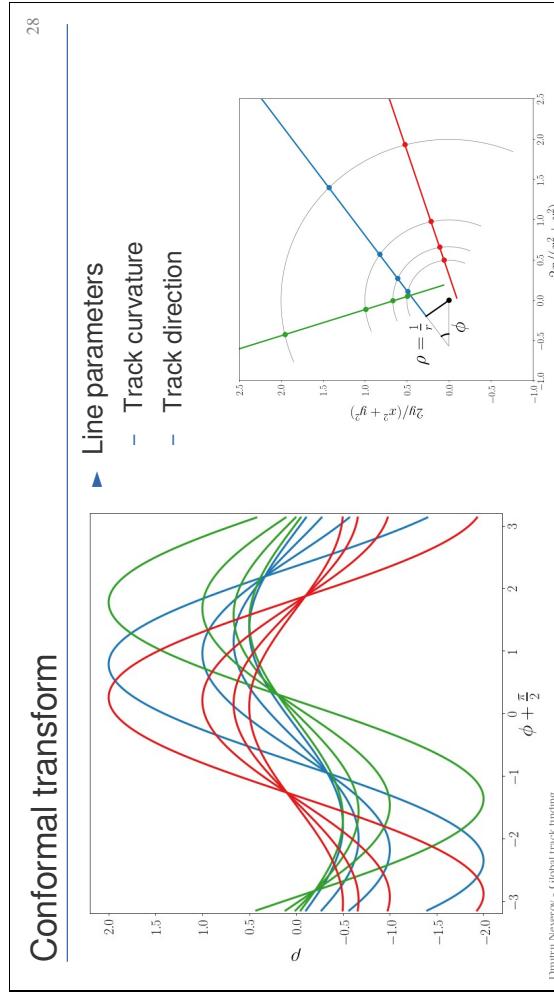
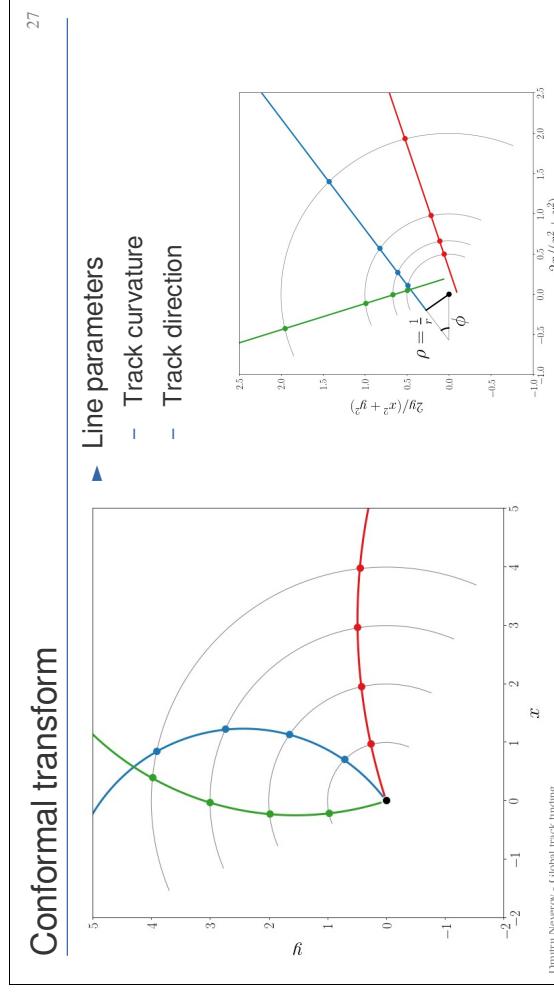
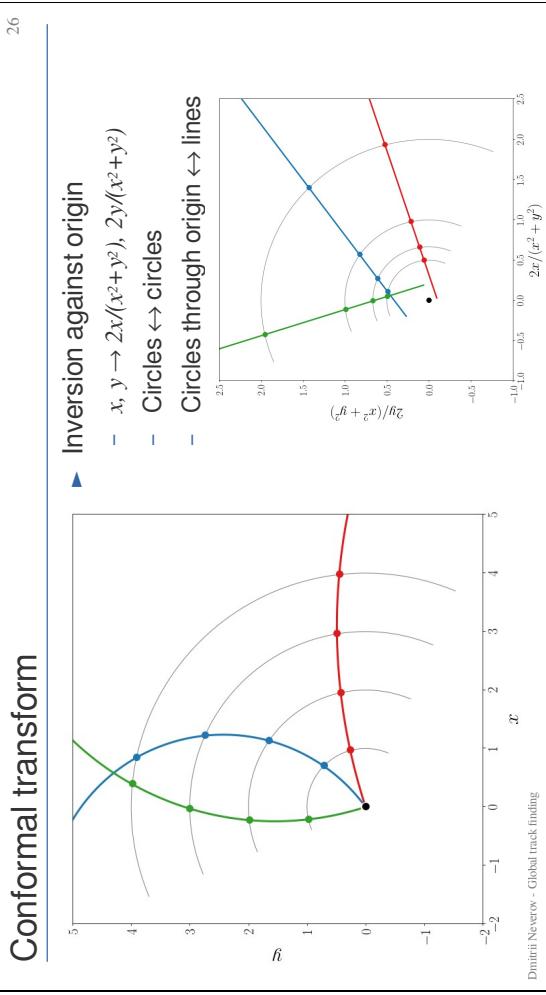
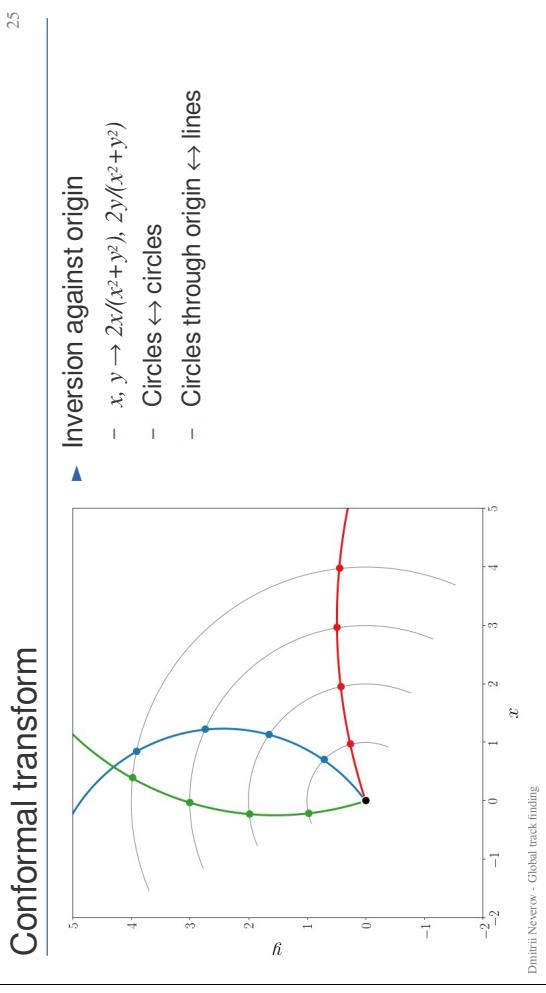
24

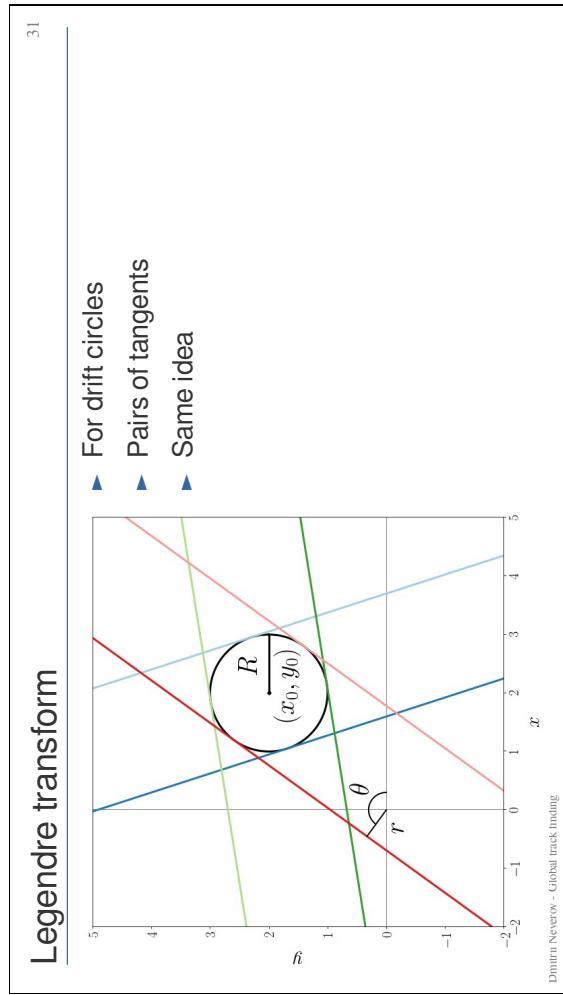
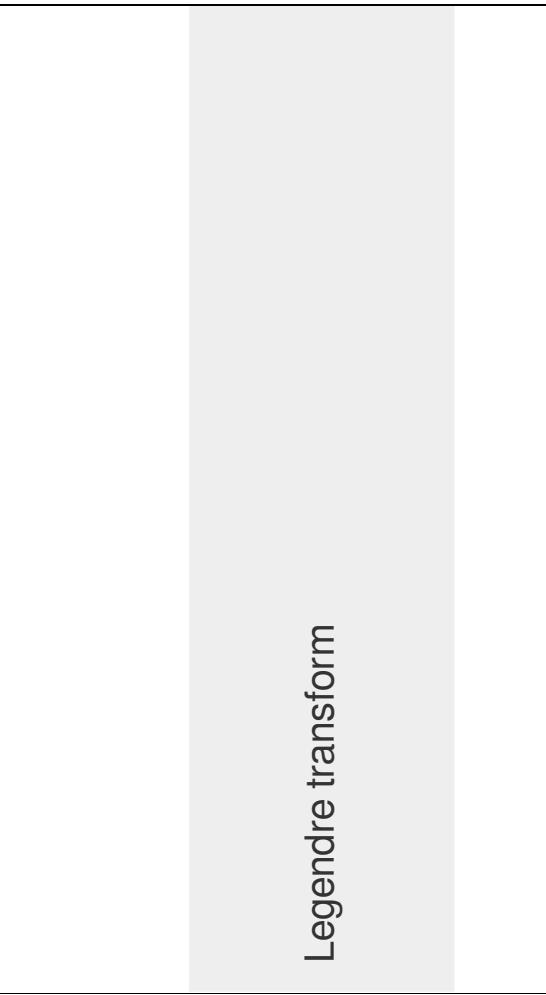
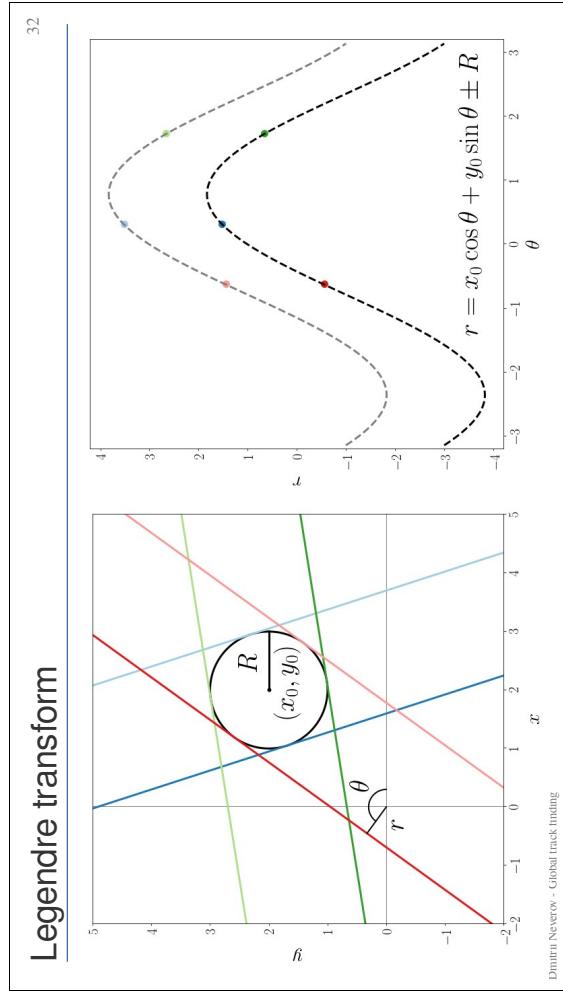
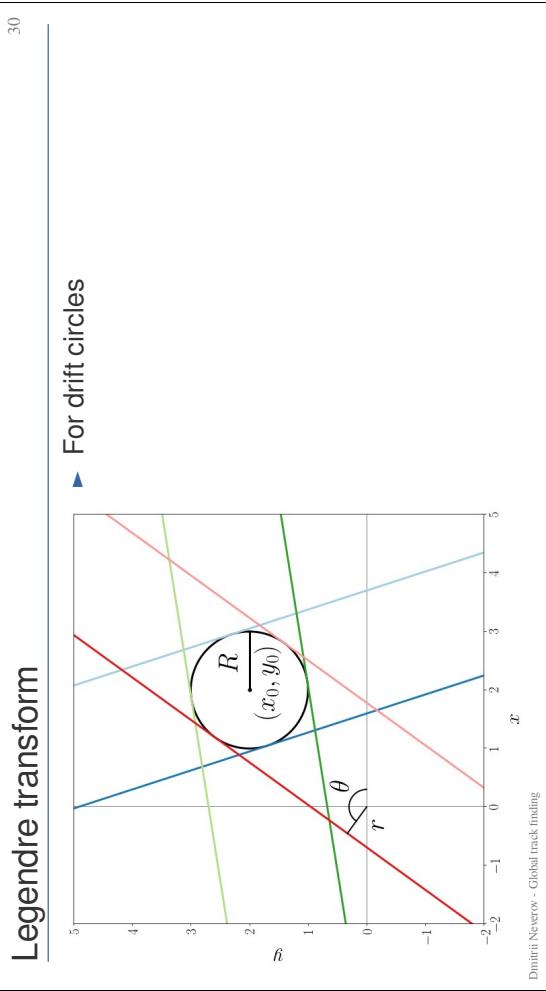
## Conformal transform



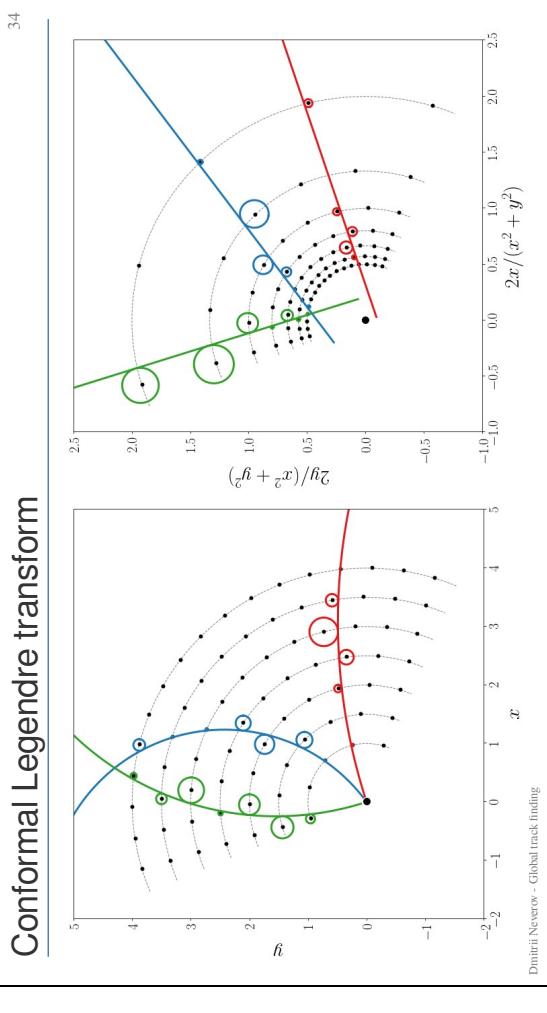
Dmitriii Neverov - Global track finding

## Conformal Hough transform

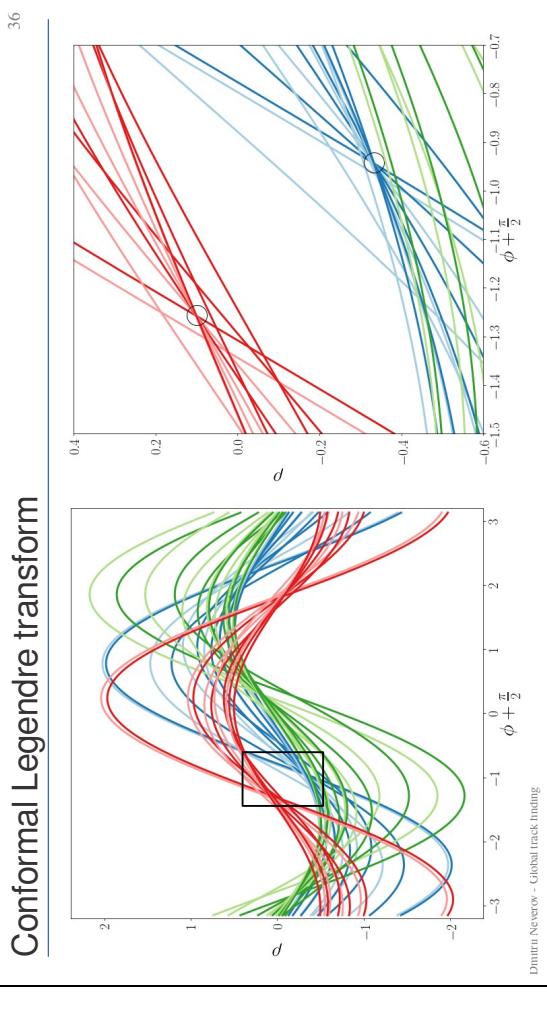




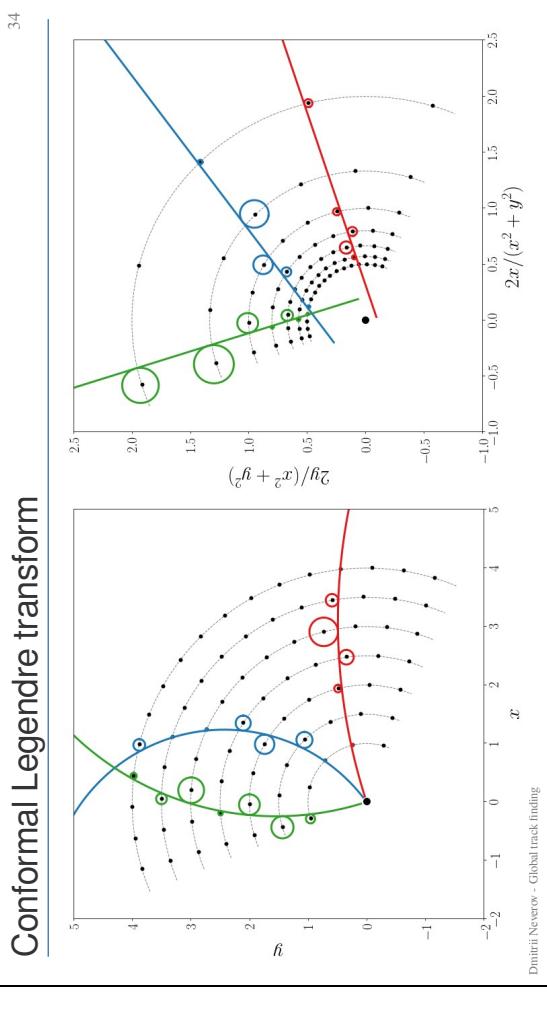
Conformal Legendre transform



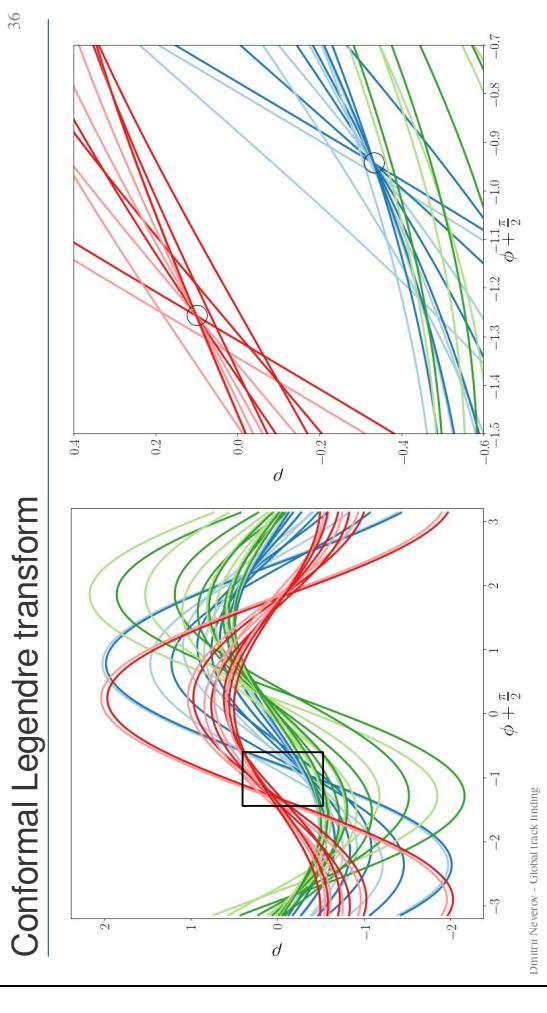
Conformal Legendre transform

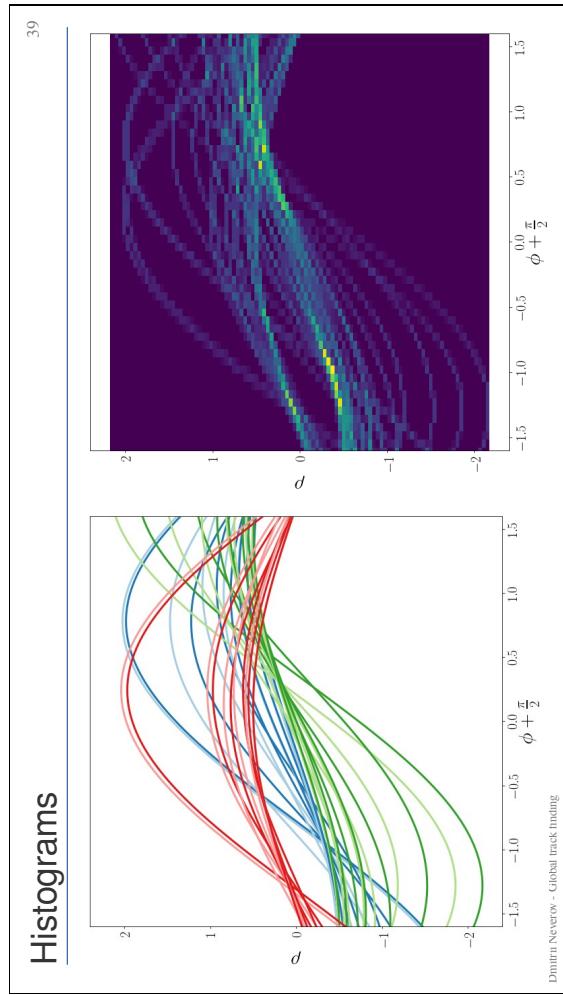
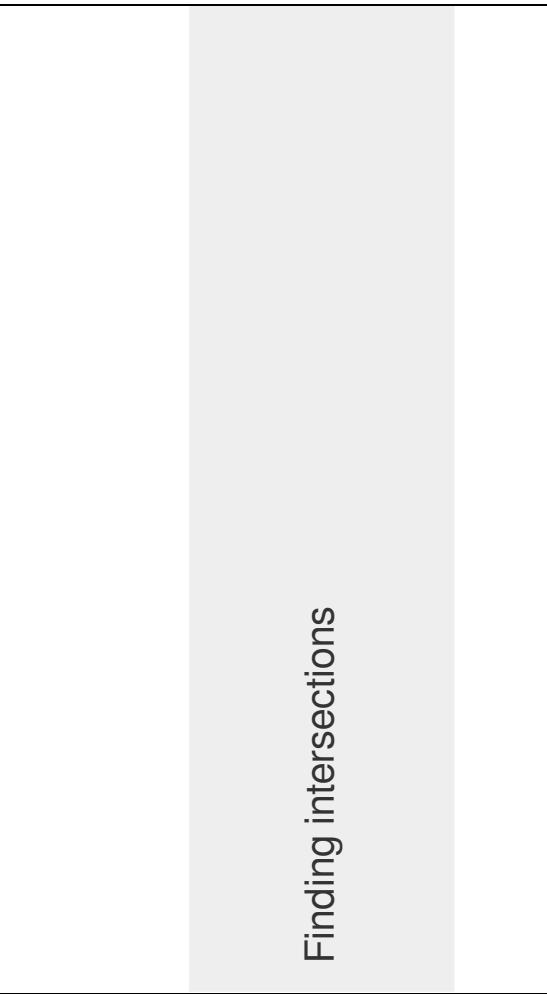
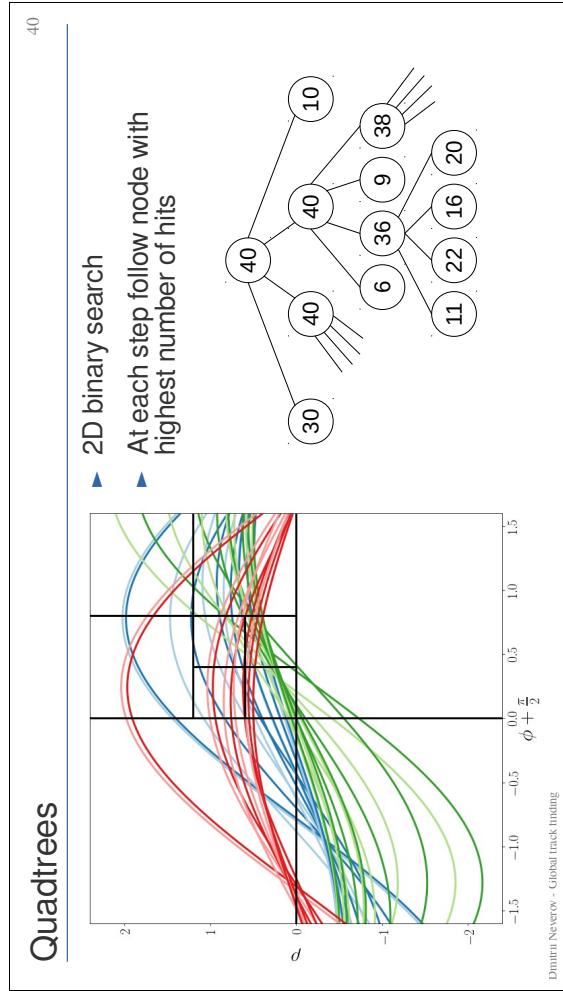
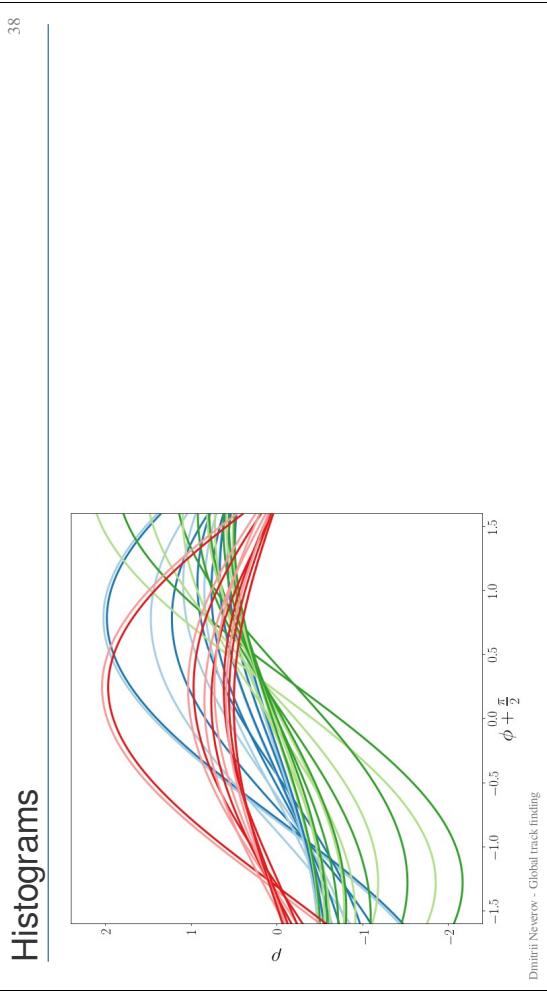


Conformal Legendre transform



Conformal Legendre transform





41

## Quadtrees

- ▶ Stop when desired precision or depth is reached
- ▶ Take found hits
- ▶ Repeat from the beginning

Discussion

43

## Conformal Legendre

- ▶ Complexity does not depend on curvature (pt)

Dmitrii Neverov - Global track finding

44

## Conformal Legendre

- ▶ Complexity does not depend on curvature (pt)

Dmitrii Neverov - Global track finding

41

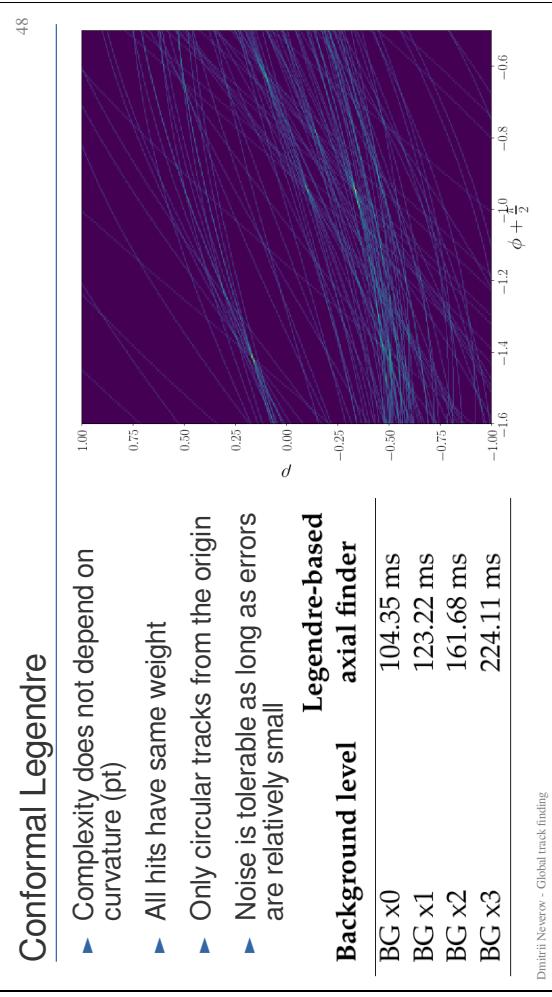
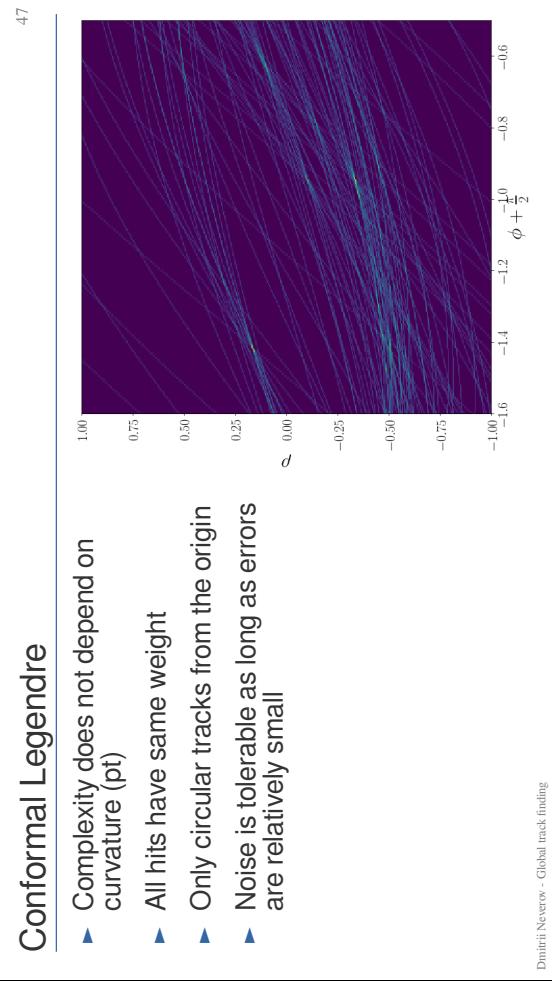
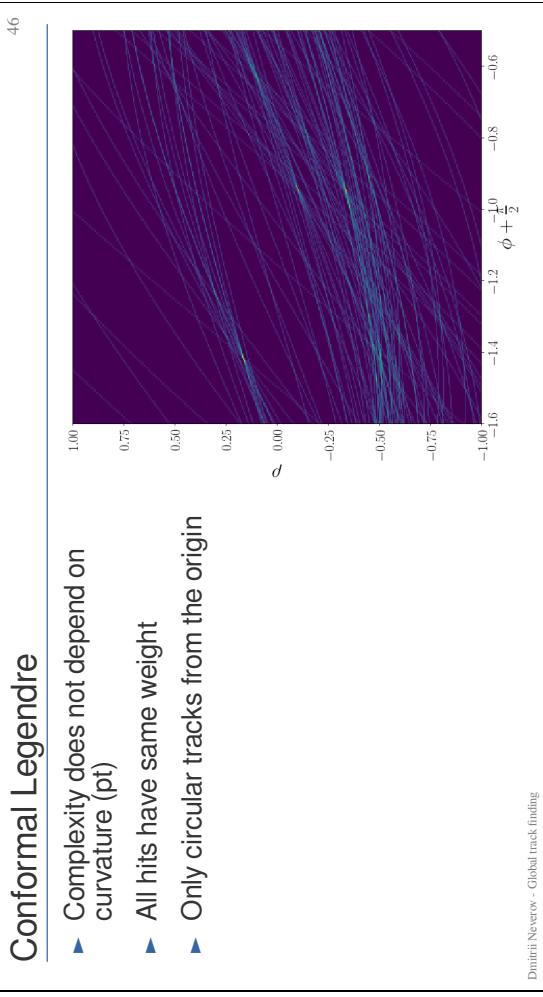
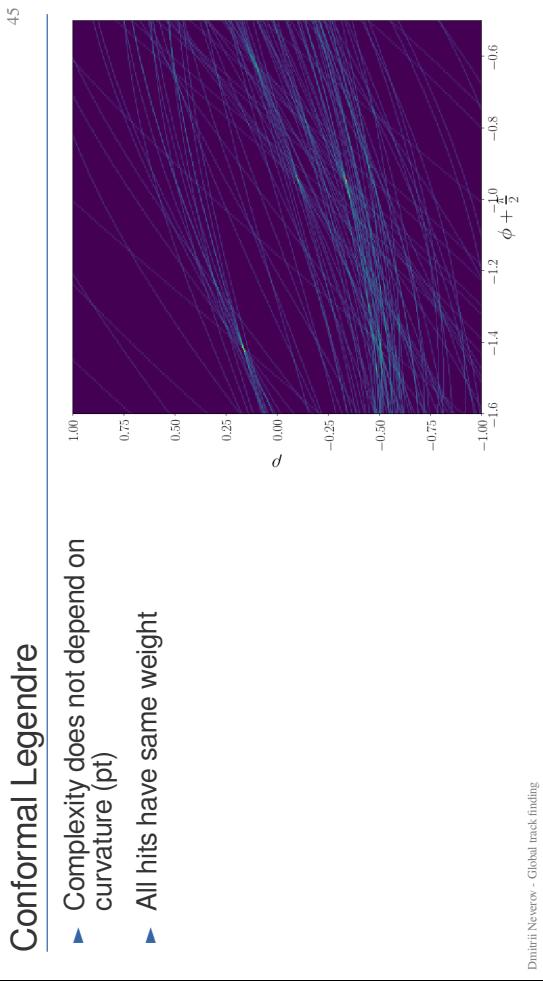
## Quadtrees

Dmitrii Neverov - Global track finding

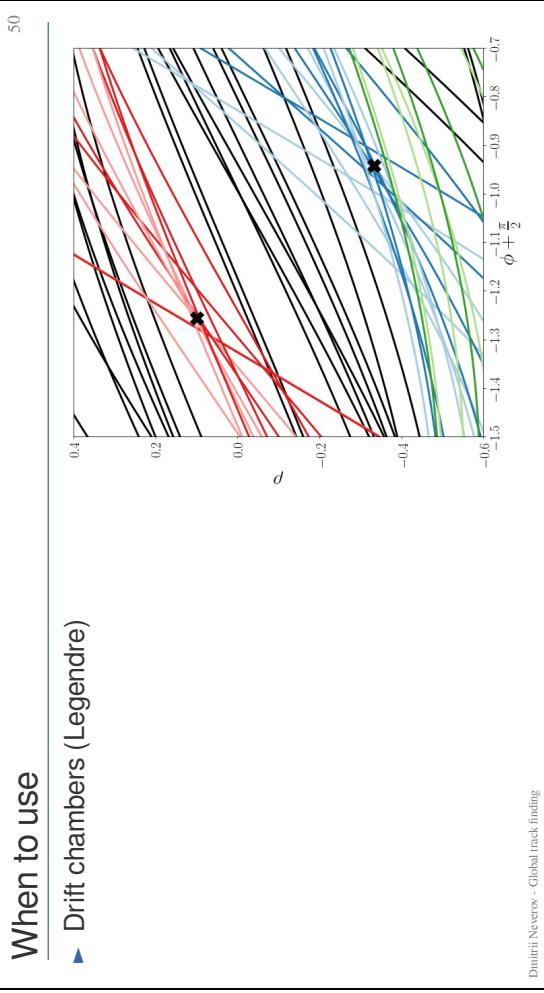
43

## Conformal Legendre

Dmitrii Neverov - Global track finding



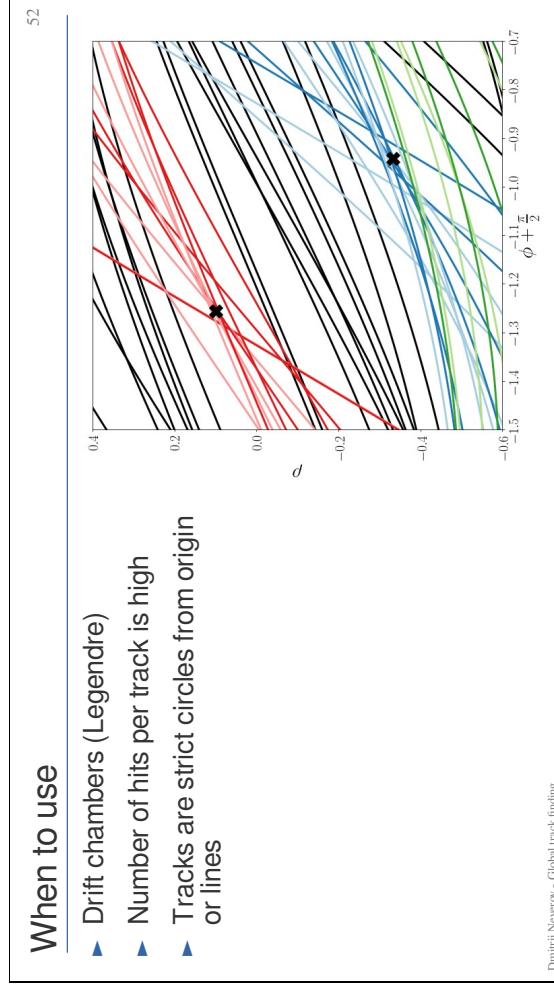
49

**When to use****When to use**

- Drift chambers (Legendre)

50

51

**When to use**

- Drift chambers (Legendre)

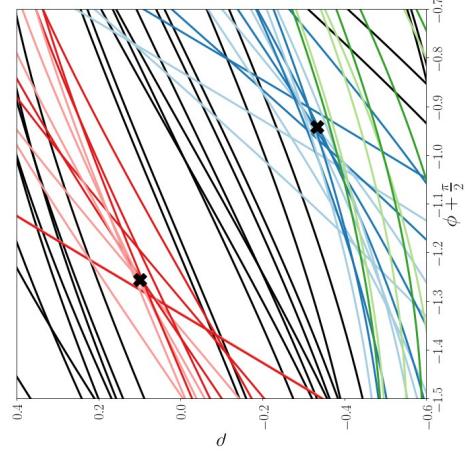
- Number of hits per track is high

## When to use

- ▶ Drift chambers (Legendre)
- ▶ Number of hits per track is high
- ▶ Tracks are strict circles from origin or lines
- ▶ High efficiency across all pt is desired
- ▶ Measurements are precise
- ▶ High noise
- ▶ High multiplicity

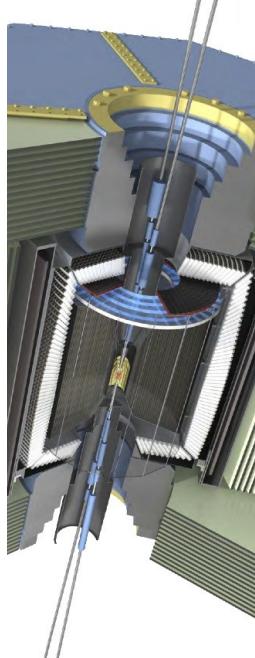
Dmitrii Neverov - Global track finding

53



## Example of Belle II

- ▶ Tracking detectors
  - Central Drift Chamber 56 layers
  - Inner Vertex Detector 6 layers



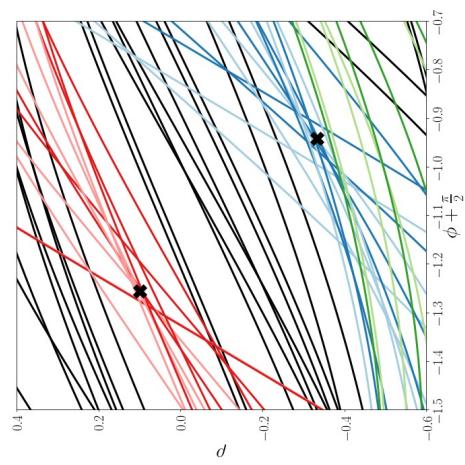
55

## When to use

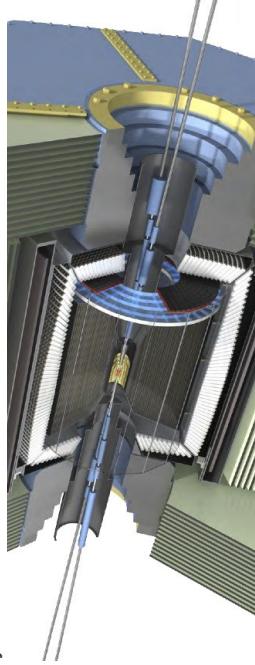
- ▶ Drift chambers (Legendre)
- ▶ Number of hits per track is high
- ▶ Tracks are strict circles from origin or lines
- ▶ High efficiency across all pt is desired
- ▶ Measurements are precise
- ▶ High noise
- ▶ High multiplicity

Dmitrii Neverov - Global track finding

54



56

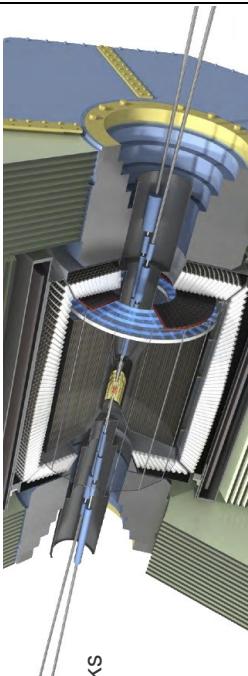


Dmitrii Neverov - Global track finding

## Example of Belle II

- ▶ Tracking detectors
  - Central Drift Chamber 56 layers
  - Inner Vertex Detector 6 layers
- ▶ Environment
  - ~11 tracks / event
  - High noise especially close to IP  
200 - 500%
- ▶ Challenges
  - Reconstruct **all** tracks
  - Low fake rate

Dmitrii Neverov - Global track finding

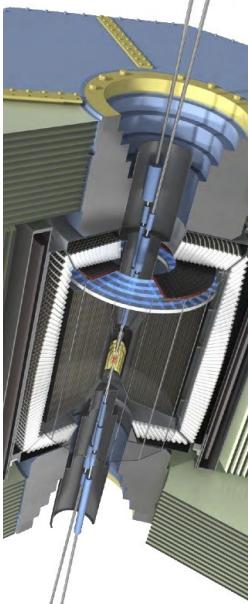


57

## Example of Belle II

- ▶ Tracking detectors
  - Central Drift Chamber 56 layers
  - Inner Vertex Detector 6 layers
- ▶ Environment
  - ~11 tracks / event
  - High noise especially close to IP  
200 - 500%
- ▶ Challenges
  - Reconstruct **all** tracks
  - Low fake rate

Dmitrii Neverov - Global track finding

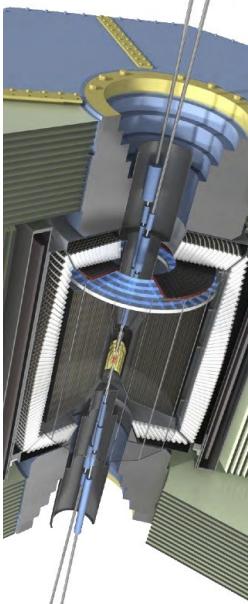


58

## Example of Belle II

- ▶ Tracking detectors
  - Central Drift Chamber 56 layers
  - Inner Vertex Detector 6 layers
- ▶ Environment
  - ~11 tracks / event
  - High noise especially close to IP  
200 - 500%
- ▶ Challenges
  - Reconstruct **all** tracks
  - Low fake rate

Dmitrii Neverov - Global track finding

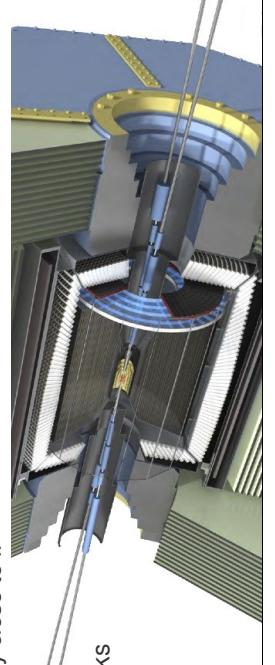


## Example of Belle II

- ▶ Tracking detectors
  - Central Drift Chamber 56 layers
  - Inner Vertex Detector 6 layers
- ▶ Environment
  - ~11 tracks / event
  - High noise especially close to IP  
200 - 500%
- ▶ Challenges
  - Reconstruct **all** tracks
  - Low fake rate

Dmitrii Neverov - Global track finding

Thank you



59

## Example of Belle II

- ▶ Tracking detectors
  - Central Drift Chamber 56 layers
  - Inner Vertex Detector 6 layers
- ▶ Environment
  - ~11 tracks / event
  - High noise especially close to IP  
200 - 500%
- ▶ Challenges
  - Reconstruct **all** tracks
  - Low fake rate

Dmitrii Neverov - Global track finding

## Legendre: Mathematical sense

- Switching to different variables
- By the simple fact of  $d(xy) = xdy + ydx$
- Thermodynamics
  - Internal energy  $\leftrightarrow$  Enthalpy etc.
- Theoretical mechanics
  - Lagrangian  $\leftrightarrow$  Hamiltonian

$$f(x, y) \quad u = \frac{\partial f}{\partial x} \quad w = \frac{\partial f}{\partial y}$$

$$df = udx + wdy$$

$$dg = udx - ydw$$

$$dh = -xdu + wdy$$

$$dk = -xdu - ydw$$

$$g(x, w) = f - wy$$

$$h(u, y) = f - ux$$

$$k(u, w) = f - ux - wy$$

## Tracks from a slightly displaced vertex

- Repeat search wrt point of closest approach of a candidate

Dmitrii Neverov - Global track finding

## Legendre: Geometrical sense

- For one-dimensional functions  $f(x) \leftrightarrow F(p)$
- Concave case
 
$$F(p) = \sup_x [px - f(x)] = -\inf_x [f(x) - px]$$
- Convex case
 
$$F(p) = \sup_x [f(x) - px] = -\inf_x [px + F]$$
- Supremum is found at a  $x_0$  where
 
$$\left. \frac{df}{dx} \right|_{x=x_0} = p$$

(p, F(p)) in Legendre space corresponds to a certain tangent line to the original function f(x)

Dmitrii Neverov - Global track finding

## Fig. 1. The Legendre transform corresponding (a) to a convex and (b) to a concave function.

Dmitrii Neverov - Global track finding

65

### (Drift) circles

- Derivative
 
$$p = -\frac{x - x_0}{\sqrt{R^2 - (x - x_0)^2}} \Rightarrow x = x_0 - \frac{|p|R}{\sqrt{p^2 + 1}}$$
- Concave and convex parts
 
$$F_1(p) = f_1(x) - px = y_0 - x_0 p + R\sqrt{p^2 + 1}$$

$$F_2(p) = x_0 p - y_0 + R\sqrt{p^2 + 1}$$
- Switch to polar coordinates
 
$$r = x \cos(\theta) + y \sin(\theta)$$
- All tangent lines to a circle in real space are represented as two sinusograms in legendre space

Dmitrii Neverov - Global track finding

**iSC** HOW CONTAINER ORCHESTRATION CAN STRENGTHEN YOUR MICRO-SERVICES

Riccardo Poggi CERN School of Computing March 2019

### THE APPROACH OF KUBERNETES

The diagram illustrates the evolution of microservices architecture through three numbered steps:

- 1 MONOLITH APPLICATION**: Shows a single large building labeled "MICRO-SERVICES ARCHITECTURE".
- 2 CONTAINERISED MICRO-SERVICES**: Shows multiple smaller buildings labeled "CONTAINERISED MICRO-SERVICES".
- 3 CONTAINER ORCHESTRATION**: Shows a city skyline with a central building labeled "HOUSTON EXPRESS HAMBURG" and "CONTAINER EX ORCHESTRATION".

**MONOLITH APPLICATION**

**CONTAINERISED MICRO-SERVICES**

**CONTAINER ORCHESTRATION**

**APPLICATION**

SHOPPING CART	ORDER STATUS
ACCOUNT ACTIVATION	INVENTORY
DATABASE	

CERN School of Computing Riccardo Poggi - iSC 2019

**iSC** HOW CONTAINER ORCHESTRATION CAN STRENGTHEN YOUR MICRO-SERVICES

Riccardo Poggi CERN School of Computing March 2019

### THE APPROACH OF KUBERNETES

The diagram illustrates the evolution of microservices architecture through three numbered steps:

- 1 MONOLITH APPLICATION**: Shows a single large building labeled "MICRO-SERVICES ARCHITECTURE".
- 2 CONTAINERISED MICRO-SERVICES**: Shows multiple smaller buildings labeled "CONTAINERISED MICRO-SERVICES".
- 3 CONTAINER ORCHESTRATION**: Shows a city skyline with a central building labeled "HOUSTON EXPRESS HAMBURG" and "CONTAINER EX ORCHESTRATION".

**MONOLITH APPLICATION**

**CONTAINERISED MICRO-SERVICES**

**CONTAINER ORCHESTRATION**

**APPLICATION**

SHOPPING CART	ORDER STATUS
ACCOUNT ACTIVATION	INVENTORY
DATABASE	

**APPLICATION**

**USER INTERFACE**

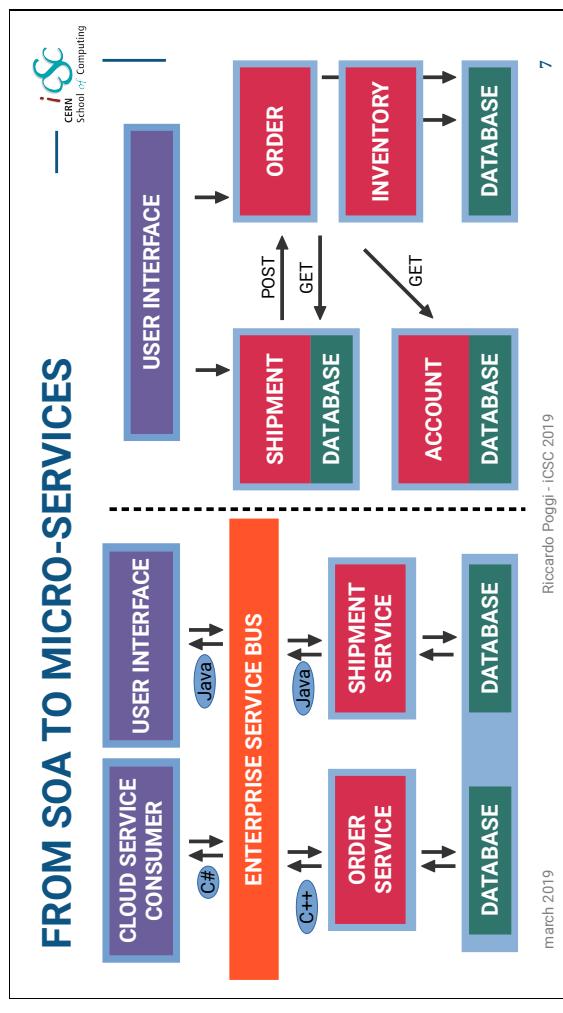
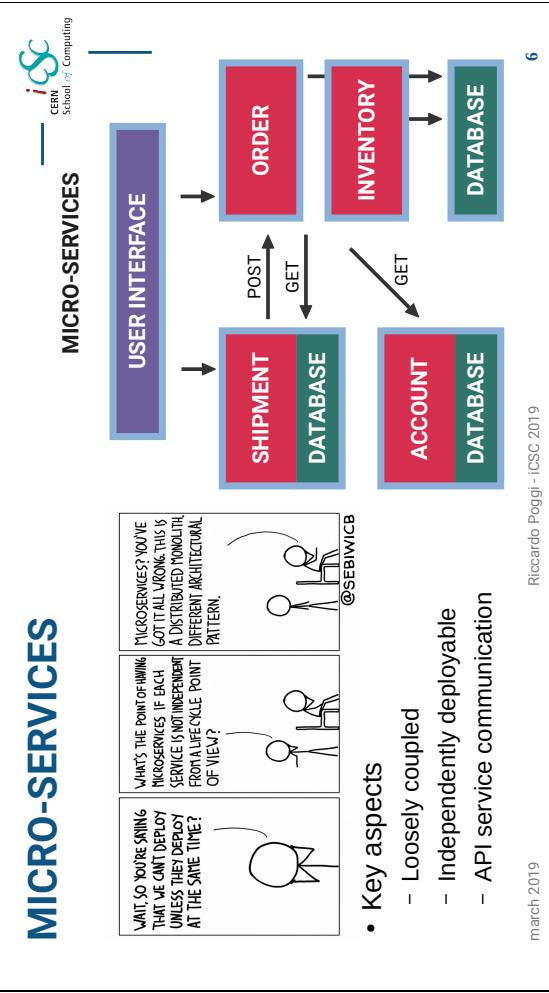
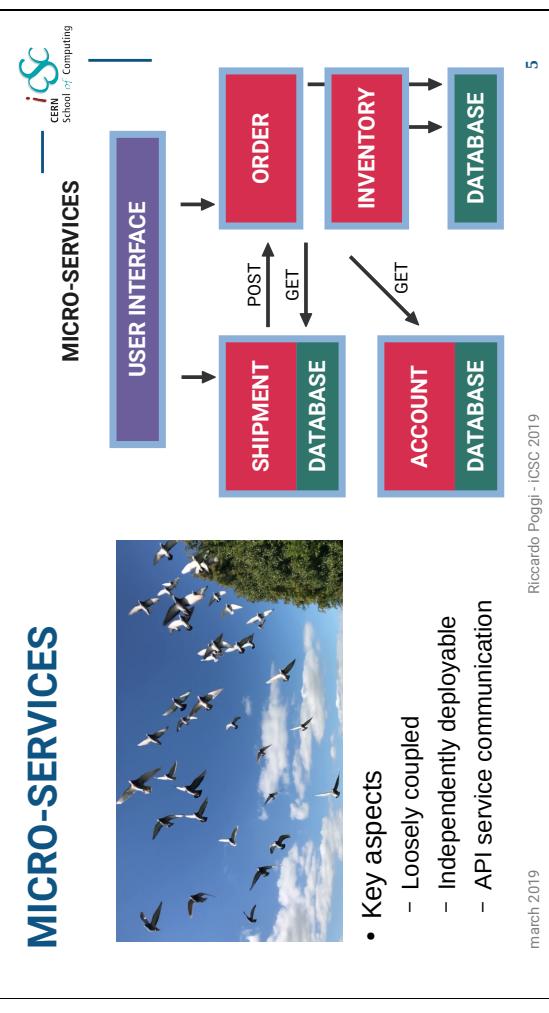
**SHOPPING CART**

**ACCOUNT ACTIVATION**

**INVENTORY**

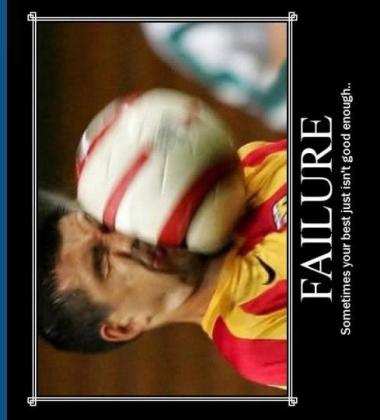
**DATABASE**

**CERN School of Computing Riccardo Poggi - iSC 2019**



## UNEXPECTED FAILURE

“ Dealing with unexpected failures is one of the hardest problems to solve especially in a distributed system ”



FAILURE  
Sometimes your nest just isn't good enough.

march 2019

Riccardo Poggi - iCSC 2019

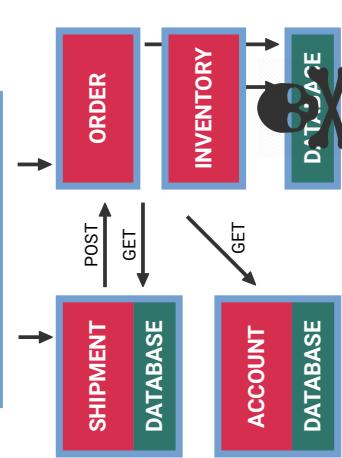
9

## FAULT-TOLERANCE

- Fault-tolerance
  - System able to continue proper operation in the event of failure of one or more of its components
- Resilience
  - Graceful degradation
  - The ability of maintaining functionality when portions of a system break down

10

Riccardo Poggi - iCSC 2019

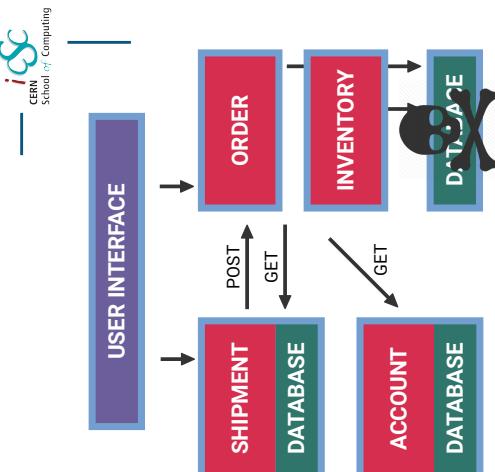


## HIGH-AVAILABILITY

- Redundancy
  - Eliminate single points of failure
  - Failure of a component does not mean failure of the entire system
- Reliable crossover
  - Not to have crossover be a single point of failure
- Monitoring
  - Detection of failures as they occur
  - A user may never see a failure, but the maintenance activity must

11

Riccardo Poggi - iCSC 2019

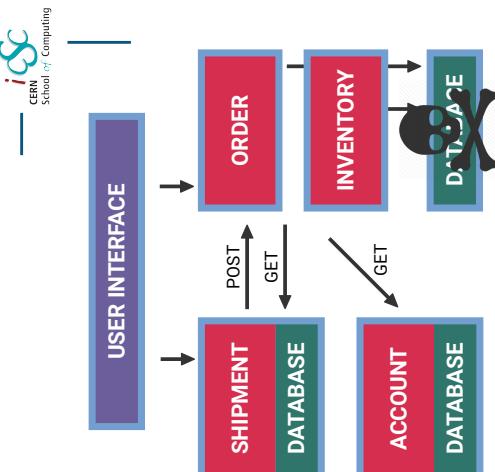


## HIGH-AVAILABILITY

- Redundancy
  - Eliminate single points of failure
  - Failure of a component does not mean failure of the entire system
- Reliable crossover
  - Not to have crossover be a single point of failure
- Monitoring
  - Detection of failures as they occur
  - A user may never see a failure, but the maintenance activity must

12

Riccardo Poggi - iCSC 2019



## FAIL-OVER POLICY

- Fail-over policy
  - Failure as an unrecoverable critical issue
  - Implementing the behaviour a service follows in case of its own failure
- Last action before failing
  - Does the service holds important data which needs to be saved?
  - Does the service has a configuration or status which needs to be saved?
- Termination
  - "Failure" can also be externally induced
  - Graceful kill (close)



### SIGKILL SIGTERM

```
def sigterm_handler(signum, frame):
    # save the state here or do whatever you want
    Print('booyan! bye bye')
    sys.exit(0)

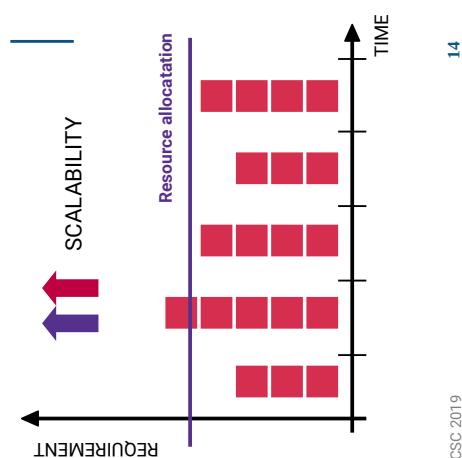
signal.signal(signal.SIGTERM, sigterm_handler)
```

Riccardo Poggi - iCSC 2019

13

## SCALABILITY & ELASTICITY

- Requirement as a function of time
  - Resource allocation and server instantiation
  - Scalability
    - Increasing the capacity
    - The available resources match the current and future usage plans
    - Scaling up: increasing the ability of an individual server
    - Scale out: adding multiple servers
  - Elasticity
    - Increasing or reducing the capacity based on the load
    - The available resources match the current demands as closely as possible

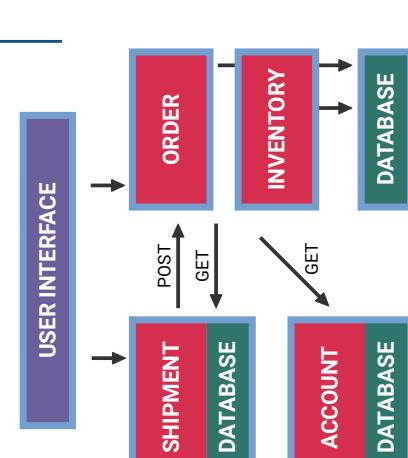


Riccardo Poggi - iCSC 2019

14

## SCALABILITY & ELASTICITY

- Requirement as a function of time
  - Resource allocation and server instantiation
  - Scalability
    - Increasing the capacity
    - The available resources match the current and future usage plans
    - Scaling up: increasing the ability of an individual server
    - Scale out: adding multiple servers
  - Elasticity
    - Increasing or reducing the capacity based on the load
    - The available resources match the current demands as closely as possible



Riccardo Poggi - iCSC 2019

15

## CONTINUOUS DELIVERY

- Independent deploy
- Without service interruption
  - No downtime!
- Rebuild and redeploy
  - only one or a small number of services



Riccardo Poggi - iCSC 2019

16

## STATEFUL VS. STATELESS

```

graph TD
    UI[USER INTERFACE] --> O[ORDER]
    O -- POST --> S[SHIPMENT]
    O -- GET --> I[INVENTORY]
    S -- GET --> DB1[DATABASE]
    I -- GET --> A[ACCOUNT]
    A -- GET --> DB2[DATABASE]
  
```

**Stateful**

- Possess saved data in a database that they read from and write to directly
- If it shares DB with other micro-services less decoupled
- When it terminates it has to save its state (fail-over policy)
- Handle request and return responses
- All necessary information supplied on the request and can be forgot after the response
- No permanent data
- Nothing to save when it terminates

**Stateless**

- Possess no data in a database that they read from and write to directly
- If it shares DB with other micro-services less decoupled
- When it terminates it has to save its state (fail-over policy)
- Handle request and return responses
- All necessary information supplied on the request and can be forgot after the response
- No permanent data
- Nothing to save when it terminates

March 2019 Riccardo Poggi - iCSC 2019

18

## CONTAINER

### VIRTUAL MACHINES

APP	APP	APP
GUEST OS	GUEST OS	GUEST OS
HYPERVISOR		

### HOST OPERATING SYSTEM

VMs have their own OS kernel, while containers share it with the host OS

March 2019 Riccardo Poggi - iCSC 2019

19

## CONTINUOUS DELIVERY

- Independent deploy
- Without service interruption
  - No downtime!
- Rebuild and redeploy
  - only one or a small number of services

```

graph TD
    UI[USER INTERFACE] --> O[ORDER]
    O -- POST --> S[SHIPMENT]
    O -- GET --> I[INVENTORY]
    S -- GET --> DB1[DATABASE]
    I -- GET --> A[ACCOUNT]
    A -- GET --> DB2[DATABASE]
  
```

New Version

March 2019 Riccardo Poggi - iCSC 2019

17

## STATEFUL VS. STATELESS

- Stateful
  - Possess saved data in a database that they read from and write to directly
  - If it shares DB with other micro-services less decoupled
  - When it terminates it has to save its state (fail-over policy)
  - Handle request and return responses
  - All necessary information supplied on the request and can be forgot after the response
  - No permanent data
  - Nothing to save when it terminates
- Stateless
  - Possess no data in a database that they read from and write to directly
  - If it shares DB with other micro-services less decoupled
  - When it terminates it has to save its state (fail-over policy)
  - Handle request and return responses
  - All necessary information supplied on the request and can be forgot after the response
  - No permanent data
  - Nothing to save when it terminates

March 2019 Riccardo Poggi - iCSC 2019

18

## CONTAINER RUNTIME

- Container Runtime
  - In a OCI/CNI compatible version is a daemon process
  - Creates and executes a container
- To fully create a container:
  1. Creates the rootfs filesystem.
  2. Creates the container
  - Set process namespaces and cgroups
  3. Connects the container to a network
  4. Starts the user process

March 2019

22

Riccardo Poggi - iCSC 2019

## CGROUPS & NAMESPACES

### CGROUP

Cpu, memory, I/O, ...

March 2019

21

Riccardo Poggi - iCSC 2019

### NAMESPACES

Cgroup, IPC, network, mount, PID, User, ...

March 2019

22

Riccardo Poggi - iCSC 2019

## DOCKER IMAGE

- Docker images are built from a base image
- Base images are built up using instructions
  - Run a command
  - Add a file or directory
  - Create an environment variable
  - What process to run when launching a container from this image

March 2019

23

Riccardo Poggi - iCSC 2019

## ENTERS DOCKER

- The most widely known container runtime is Docker
- But there are also others
  - rkt, containerd, lxd, singularity, etc..

March 2019

23

Riccardo Poggi - iCSC 2019

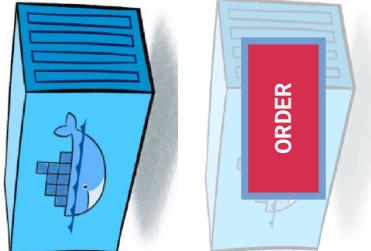
## DOCKERFILE

```
# Dockerfile
FROM ubuntu:latest
RUN apt-get update
RUN apt-get install -y python python-pip
COPY . /app
CMD python /app/order_service.py
```

```
$ docker build -t my-image .
$ docker run my-image
```

march 2019

25



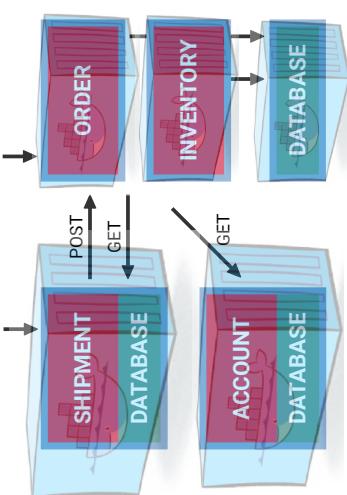
Riccardo Poggi - iCSC 2019

## CONTAINERISED MICRO-SERVICES

- Apply containers to micro-services architecture
  - One-to-one map for single independent services container
  - Decoupling inside/outside container
- Questions still to be solved
  - Tightly coupled processes inside one container?
  - Everything running on one single node
  - Redundancy and scalability

march 2019

26 Riccardo Poggi - iCSC 2019



## CONTAINERISED MICRO-SERVICES

- Apply containers to micro-services architecture
  - One-to-one map for single independent services container
  - Decoupling inside/outside container
- Questions still to be solved
  - Tightly coupled processes inside one container?
  - Everything running on one single node
  - Redundancy and scalability

march 2019

27 Riccardo Poggi - iCSC 2019

## CONTAINERISED MICRO-SERVICES

- Apply containers to micro-services architecture
  - One-to-one map for single independent services container
  - Decoupling inside/outside container
- Questions still to be solved
  - Tightly coupled processes inside one container?
  - Everything running on one single node
  - Redundancy and scalability

march 2019

28 Riccardo Poggi - iCSC 2019

## CONTAINERISED MICRO-SERVICES



Riccardo Poggi - iCSC 2019

25

26 Riccardo Poggi - iCSC 2019

27 Riccardo Poggi - iCSC 2019

**HOW CONTAINER ORCHESTRATION CAN STRENGTHEN YOUR MICRO-SERVICES**

Riccardo Poggi

March 2019

## ORCHESTRATION

CERN School of Computing

• Orchestration
 

- Automated arrangement
- Coordination
- Management

• Useful tool for
 

- Service Discovery
- Load Balancing
- Health checks
- Auto-scaling
- Zero-downtime deploys
- (And much more...)

march 2019 Riccardo Poggi - iCSC 2019 30

## KUBERNETES

CERN School of Computing

• Master
 

- The machine that controls Kubernetes nodes

• Node
 

- The machines that perform the requested and assigned tasks

• Pod
 

- A group of one or more containers deployed to a single node

• kubectl
 

- Command line configuration tool for Kubernetes

march 2019 Riccardo Poggi - iCSC 2019 32

## PLATFORM OVERVIEW

CERN School of Computing

User workloads	—	APPS	JOBS	SERVICES
Distributed container management	—	ORCHESTRATOR		
Local container management	—	CONTAINER RUNTIME		
Container agnostic infrastructure	—	INFRASTRUCTURE		

Riccardo Poggi - iCSC 2019 31

How container orchestration can strengthen your micro-services: the approach of Kubernetes

Riccardo Poggi

## POD

- A Pod is the basic building block of Kubernetes
  - The smallest and simplest unit
- “One-container-per-Pod”
  - Most common Kubernetes use case
  - Pod as a wrapper around a single container
- Encapsulate multiple co-located containers
  - Tightly coupled
  - Need to share resources

march 2019  
Riccardo Poggi - iCSC 2019  
33

```
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  containers:
    - image: ubuntu:18-demo
      name: ubuntu
```

```
$ kubectl create -f pod.yaml
pod demo created
$ kubectl get pod demo
NAME READY STATUS RESTARTS AGE
demo 1/1 Running 0 1m
$ kubectl delete pod demo
pod demo deleted
```

Riccardo Poggi - iCSC 2019

34

## REPLICASET

- Kubernetes Controller
  - Changes the system to move it from the current to the desired state
- ReplicaSet
  - Ensures that a specified number of pod replicas are running at any given time

```
$ kubectl create -f replicaset.yaml
replicaset "demo" created
$ kubectl get replicaset demo
NAME DESIRED CURRENT AGE
demo 2 2 40s
$ kubectl scale --replicas=4 replicaset/demo
replicaset "demo" scaled
$ kubectl getreplicaset demo
replicaset "demo" deleted
```

Riccardo Poggi - iCSC 2019

34

## DEPLOYMENT

- Deployment controller
  - Declarative update for Pods and ReplicaSet
- Rollout
  - Ensure max unavailable/surge
  - e.g. at least 75% are up (25% max unavailable)
- Roll back

march 2019  
Riccardo Poggi - iCSC 2019  
35

```
apiVersion: v1
kind: Replicaset
metadata:
  name: demo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: demo
  template:
    metadata:
      labels:
        app: demo
    spec:
      containers:
        - image: ubuntu:18-demo
          name: ubuntu
```

```
apiVersion: v1
kind: Deployment
metadata:
  name: demo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: demo
  template:
    metadata:
      labels:
        app: demo
    spec:
      containers:
        - image: ubuntu:18-demo
          name: ubuntu
          env:
            - name: VERSION
              value: "v1"
```

```
$ kubectl create -f deployment.yaml
deployment "demo" created
$ kubectl get deployment demo
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
demo 2 2 2 2 40s
$ # alternatives: "kubectl edit" or "kubectl apply -f"
$ kubectl patch deployment -p '{"spec": [{"...": "value": "v2"}]}'
"demo" patched
$ kubectl rollout undo deployment/demo
$ kubectl delete deployment demo
deployment "demo" deleted
```

Riccardo Poggi - iCSC 2019

35

## SERVICE

- Service
  - Abstraction to functionally group Pods
  - e.g. Front-end Pods, back-end Pods
- Consistent front for a set of Pods to offer a given service
- Possible to scale up and down Pods

```
apiVersion: v1
kind: Service
metadata:
  name: demo
spec:
  ports:
    - port: 80
      targetPort: 8080
  selector:
    run: demo
```

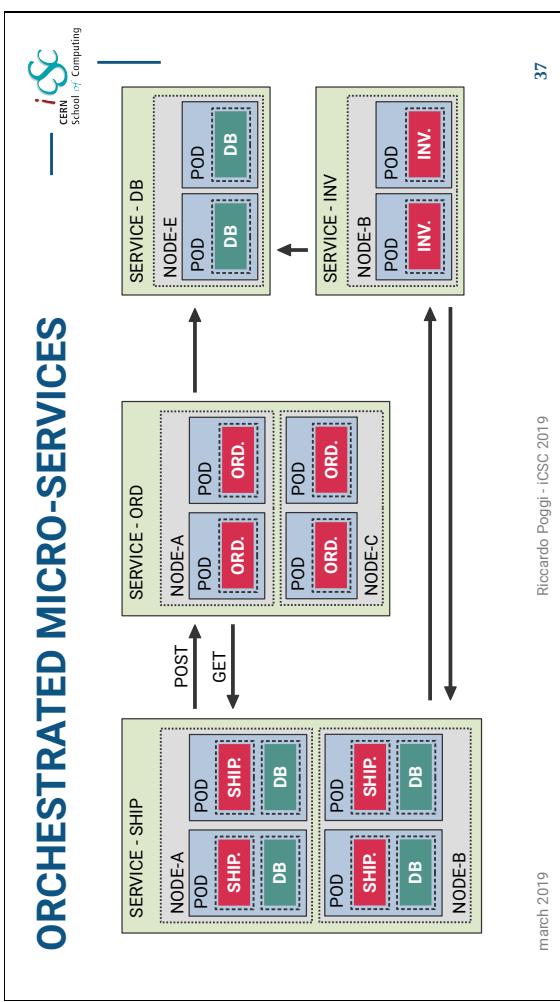
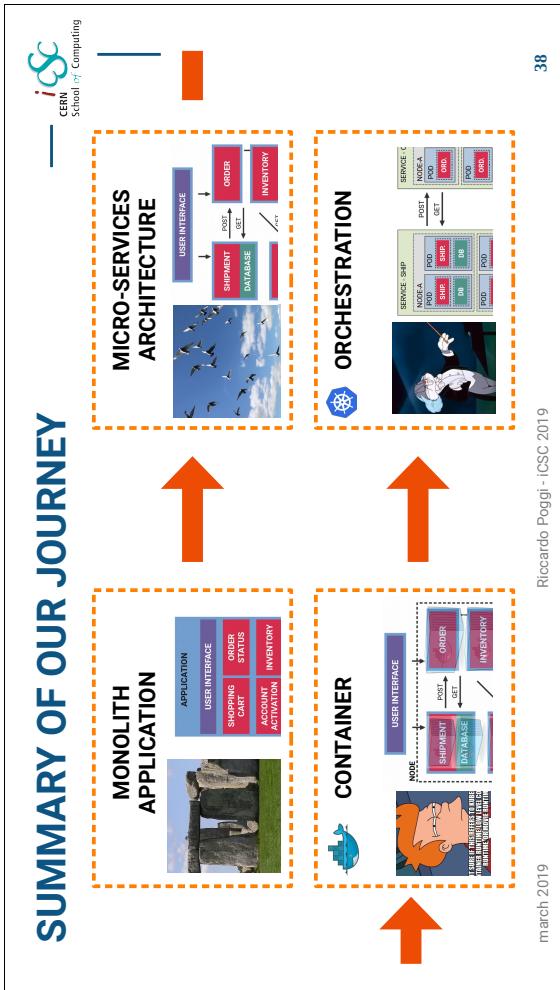
```
$ kubectl create -f deployment.yaml
$ kubectl create -f service.yaml
service "demo" created
NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE
demo 10.254.132.169 <none> 80/TCP 30s
$ kubectl scale deployment/demo --replicas=4
$ kubectl delete svc demo
$ kubectl delete deployment demo
```

Riccardo Poggi - iCSC 2019

36

# How container orchestration can strengthen your micro-services: the approach of Kubernetes

Riccardo Poggi



38

march 2019

37

Riccardo Poggi - iCSC 2019

115

# A Scientist's Guide to FPGAs

**CERN School of Computing**

ICSC 2019  
Alexander Ruede  
(CERN/KIT-IPE)

**KIT**  
Karlsruhe Institute of Technology

A. Scientist's Guide to FPGAs – Alexander Ruede – ICSC 2019

3

## Content

1. Introduction
2. The Emergence of FPGAs
3. Digital Design
4. Anatomy of FPGAs
5. Classical Design Flow
6. CPU / GPU vs. FPGA
7. Pros & Cons
8. Applications
9. Examples
10. Getting Started

A. Scientist's Guide to FPGAs – Alexander Ruede – ICSC 2019

2

## 2. The Emergence of FPGAs

The timeline illustrates the evolution of programmable logic technology:

- 1959: Invention of the MOSFET
- 1961: First Integrated Circuit (IC) Computer
- 1963: CMOS
- 1965: Moore's Law
- 1975: Programmable Logic Array (PLA)
- 1985: First FPGA (Xilinx XC2064)
- 1993: EEPROM & FLASH Memory
- 2015: Intel acquires Altera
- 2016: 16nm Virtex UltraScale+ (indicated by a large blue arrow)

**Note:** All dates are rather indicative than exact

A. Scientist's Guide to FPGAs – Alexander Ruede – ICSC 2019

4

## 1. Introduction

**FPGA:** Field-Programmable Gate Array

Growing  $\left\{ \begin{array}{l} \text{market} \\ \text{applications} \end{array} \right\}$  of FPGAs

The graph plots 'Total Cost' (y-axis) against 'Volume' (x-axis). A red curve represents ASICs, which decreases in cost as volume increases. A purple curve represents FPGAs, which increases in cost as volume increases. The two curves intersect at a 'crossover point'. After the crossover point, FPGAs become more cost-effective than ASICs.

• Well established in HEP experiments

• Finding the way into data centers

• Can be substitution for:
 

- ASICs (traditionally)
- Processors (recently)

A. Scientist's Guide to FPGAs – Alexander Ruede – ICSC 2019

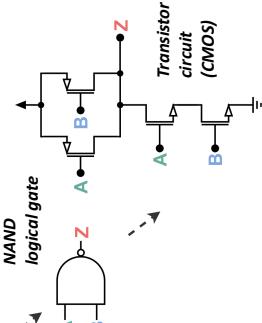
3

## 3. Digital Design

### Digital Logic

- Digital information is processed and stored in *binary* form
- Boolean algebra and truth tables are used to express combinatorial logic circuits
- Basic logical function (AND, OR, etc.) are abstracted in *logical gates*
- Gates can efficiently be implemented in transistor circuits (e.g. CMOS)
- Every logical function can be implemented by using gates

A	B	Z	Truth table of logical function
0	0	1	
0	1	1	
1	0	1	
1	1	0	



A Scientist's Guide to FPGAs – Alexander Ruede – ICS 2019

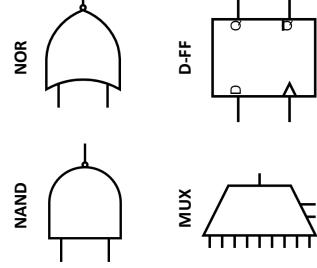
5

## 3. Digital Design

### Digital Building Blocks and Processes

There are two kinds of processes with different building blocks:

1. **Combinatorial**  
→ “Instant” state changes, e.g.:
  - Classical gates (especially NAND & NOR)
  - Multiplexer (MUX)
2. **Synchronous**  
→ “Clocked” state changes, e.g.:
  - Flip Flop (e.g. D-FF)
  - FIFO (First-In First-Out)



A Scientist's Guide to FPGAs – Alexander Ruede – ICS 2019

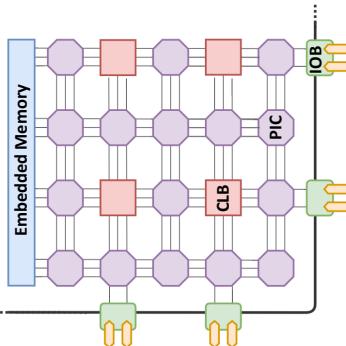
6

## 4. Anatomy of FPGAs

### Architecture

Configurable logic blocks, interconnected by a switch matrix and surrounded by I/Os.

- CLB: Configurable Logic Block
- IOB: Input-Output Block
- PIC: Programmable Interconnect
- Clock Management
- Memory
- Hardened Cores



A Scientist's Guide to FPGAs – Alexander Ruede – ICS 2019

7

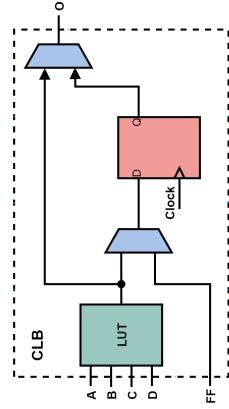
## 4. Anatomy of FPGAs

### Configurable Logic Block (CLB)

(Also “logic cell” or “logic element”)

- Logic functions are implemented in Look-Up-Tables (LUTs)
- LUTs can implement any arbitrarily defined n-input Boolean function
- D-type flip-flops (storage elements) can be triggered on either clock edge
- Flip-flops can take input from outside the CLB or from the LUT

Simplified example CLB with one 4-input LUT and one flip-flop



A Scientist's Guide to FPGAs – Alexander Ruede – ICS 2019

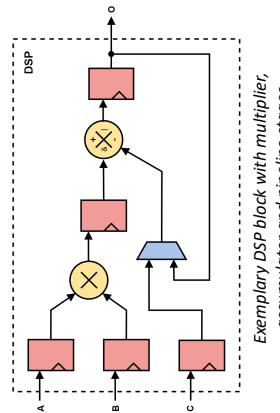
8

## 4. Anatomy of FPGAs

**Hardened Cores** (Also called "IP cores")  
Complicated tasks (e.g., multiplication) take up a lot of logic cells  
→ **Hardened cores** in silicon for more effective use of resources

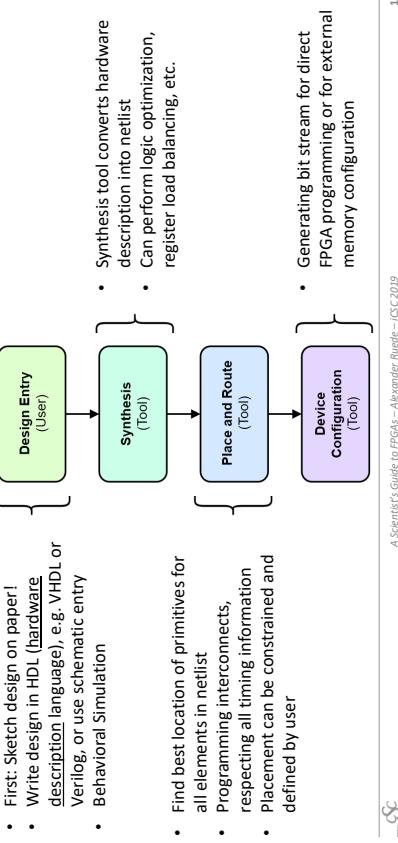
Typical cores found in modern FPGAs:

- Memory (Block RAM)
- FIFO
- Shift Register
- DSP blocks
- Clocking (Programmable PLL)
- Communication interfaces (e.g. PCIe)
- Serializer/Deserializer (SerDes)
- CPU



A Scientist's Guide to FPGAs – Alexander Ruedel – ICS-C 2019

## 5. Classical Design Flow



10

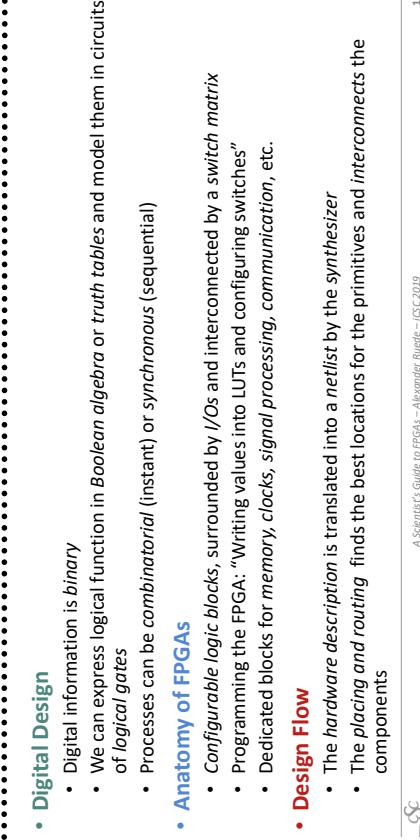
## 5. Classical Design Flow

### How it looks like...

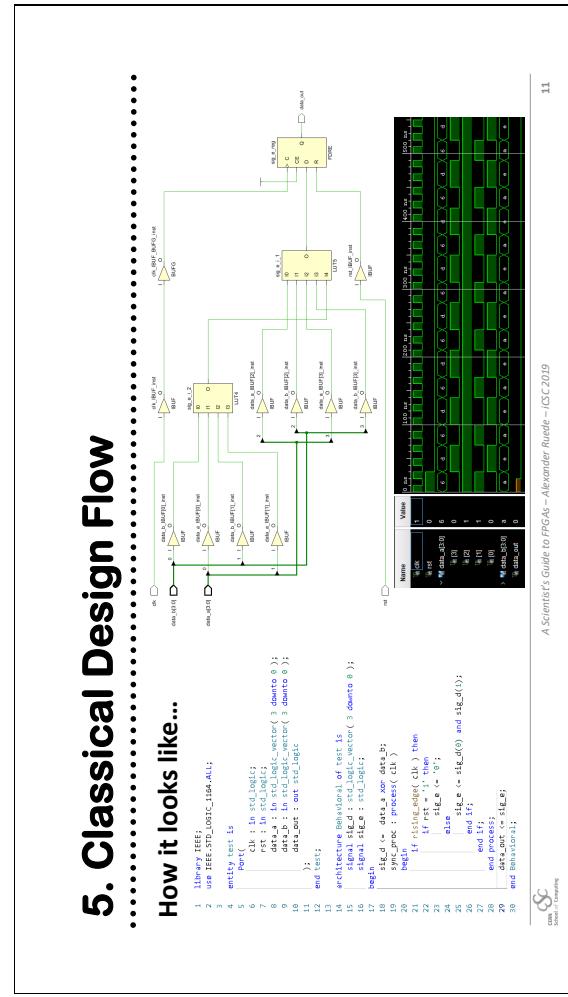
- Digital Design
  - Digital information is *binary*
  - We can express logical function in *Boolean algebra* or *truth tables* and model them in circuits of *logical gates*
  - Processes can be *combinatorial* (instant) or *synchronous* (sequential)
- Anatomy of FPGAs
  - Configurable logic blocks, surrounded by I/Os and interconnected by a switch matrix
  - Programming the FPGA: "Writing values into LUTs and configuring switches"
  - Dedicated blocks for *memory*, *clocks*, *signal processing*, *communication*, etc.
- Design Flow
  - The hardware description is translated into a *netlist* by the *synthesizer*
  - The *placing and routing* finds the best locations for the primitives and *interconnects* the components

A Scientist's Guide to FPGAs – Alexander Ruedel – ICS-C 2019

## Recapture

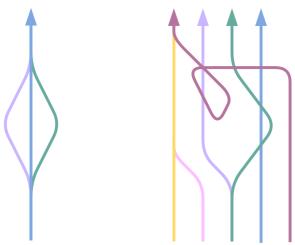


12



## 6.1 Software vs. HDL

- Software / CPU
  - Specifying a sequence of instructions
  - Implicit sequential processes
  - Explicit concurrency
  - Fixed memory hierarchy
- HDL / FPGA
  - Describing structure and behavior of digital components
  - (Implicit) Arbitrary concurrency
  - Synchronous and/or purely combinatorial processes
  - Options to explicitly express time (in simulation)
  - Flexible memory hierarchy



A Scientist's Guide to FPGAs – Alexander Ruede – ICS 2019

## 6.2 GPU vs. FPGA

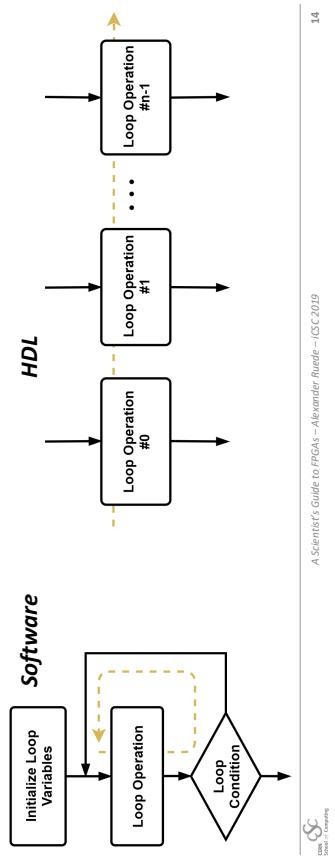
- GPU: Large array of ALU cores, including cache memory and thread interfaces
  - FPGA: Large array of logic blocks, surrounded by I/Os, connected by a switch matrix
- | GPU                         | FPGA   |
|-----------------------------|--|
| • Data path and data types: | Fixed <span style="color: green;">X</span> Fully customizable        |
| • Memory:                   | complex hierarchies <span style="color: blue;">X</span> Flexible     |
| • Floating Point:           | Native support <span style="color: purple;">X</span> Limited support |
| • Power Efficiency*:        | Low to Medium <span style="color: red;">X</span> Very High           |

\* "GPU vs FPGA Performance Comparison" - BERTEN DSP S.L., BWP001 v1.0, 2016

A Scientist's Guide to FPGAs – Alexander Ruede – ICS 2019

## 6.1 Software vs. HDL

### Example: For-Loop



A Scientist's Guide to FPGAs – Alexander Ruede – ICS 2019

## 7. Pros & Cons

### Pros

- High flexibility
- Customizable data types
- Arbitrary concurrency
- Connectivity
- Power efficiency
- Real time suitability
- Suitable for safety critical applications

### Cons

- High complexity
- Limited "math support"
- Effort for floating point implementation
- It's **hardware** design - software knowledge does not apply
- Cost (for large devices)

A Scientist's Guide to FPGAs – Alexander Ruede – ICS 2019

## 8. Application Areas

Some of *many* application areas:

- Networking/Telecommunication  
→ High throughput, I/O density
- Space  
→ Radiation hardness, reconfigurability
- Medical/Scientific Instrumentation  
→ Connectivity and customizability
- Image Processing  
→ High throughput, parallelism, interfacing
- Machine Learning  
→ Flexible data types, power efficiency



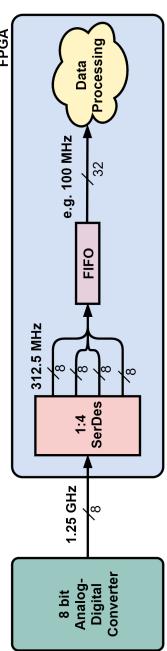
## 9.1 Example: High Speed Data Converter

### Typical challenge in experimental setup:

Interfacing to high speed analog-digital converter (ADC)

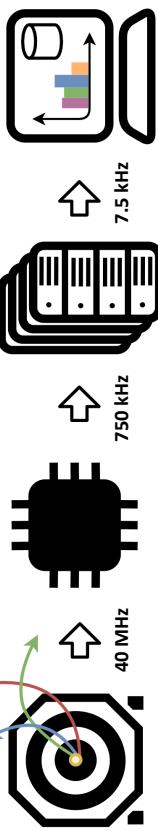
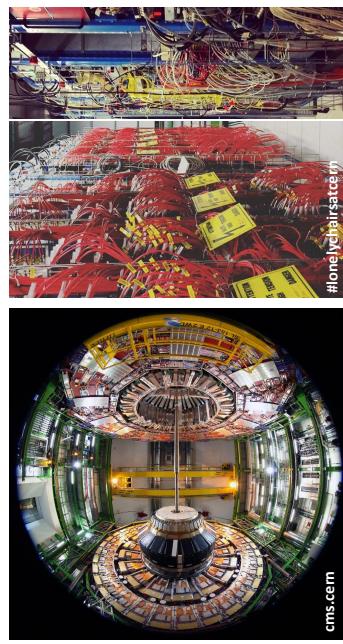
Example: 8 bit ADC with 1.25 GHz sampling rate

- Challenge: Interface between high frequency sampling rate and lower FPGA-internal processing frequency  
→ Deserialize data!



## 9.2 Example: CMS Phase II DAQ

- Phase II Trigger & DAQ

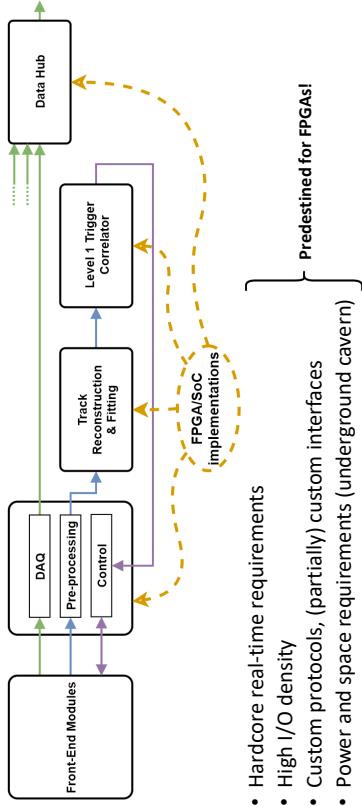


### Level-1 Trigger

### High-Level Trigger

- |     |  |                         |
|-----|--|-------------------------|
| CMS | • ~50 k High-speed links from detector | • FPGA implementation   |
|     | • ~50 Tb/s throughput                  | • 12.5 µs decision time |
|     |  | • 98.125% rejection     |
|     |  | • 99% rejection         |

## 9.2 Example: CMS Phase II Tracker



A Scientist's Guide to FPGAs – Alexander Ruede – ISCC 2019

21

## 10. Getting Started

- Visit Giorgios lecture!
- Low-cost development boards:
  - TinyFPGA (open source) – Lattice XO2/iCE40 FPGA
  - Digilent Basys3 – Xilinx Artix-7 FPGA
- Most Vendors tool chains are for free, open source chains exist for some FPGAs (Project IceStorm for Lattice iCE40 FPGAs)
- Try out different languages: VHDL, Verilog, Migen, MyHDL,...
- Start with blinking LED (the “Hello World” of hardware)
- Trial and error + online education
- Realize your project!

A Scientist's Guide to FPGAs – Alexander Ruede – ISCC 2019

22

## Backup: Languages

- Languages are often divided into two subsets:
  - **Synthesizable** → can be translated into a physical design
  - **Non-synthesizable** → only used for simulation
- **Hardware Description Languages (HDLs)**  
→ *Good for advanced design optimization*
  - **Describe and simulate hardware on behavioral/RTL level**
  - VHDL, Verilog (the “classical” HDLs)
  - SystemVerilog (enhanced Verilog)
  - Migen, Chisel, Clash, Spinal ... (unconventional/experimental languages)
- **High Level Synthesis (HLS)**  
→ *Good for fast development cycles*
  - Higher level abstraction, based on C/C++
  - HDL as output product
  - OpenCL: Cross-Platform parallel language (good for SoCs)

A Scientist's Guide to FPGAs – Alexander Ruede – ISCC 2019

23

core  
classroom

## Outline

# Efficient C++ implementation of custom FEM kernel with Eigen

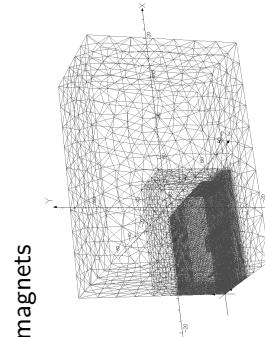
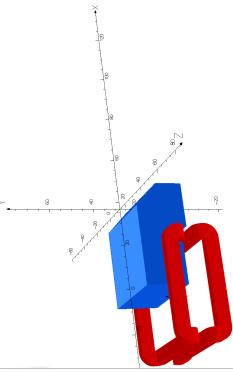
Mikhail Sizov

INP SB RAS

January 31, 2019

- ➊ Problem statement
- ➋ Basis of Finite element analysis
- ➌ Eigen
- ➍ Shape functions
- ➎ Principle of minimum energy
- ➏ Global coefficient matrix
- ➐ Obtaining solution

## Design of electromagnets



Lens quality is determined by features of magnetic field (e.g. uniformity)

- Qualified labor, but includes a lot of trial and error
- Simulating fields in 3D with existing tools takes **up to several days**
- Existing software can't reuse intermediate calculations

Goals:

- Reduce human labor
- Reduce total count of full 3D modeling

Solution: randomizing geometries with genetic algorithms

Takes thousands of iterations to get good results, need to be *fast*

## Problem statement

Lens quality is determined by features of magnetic field (e.g. uniformity)

- Qualified labor, but includes a lot of trial and error
- Simulating fields in 3D with existing tools takes **up to several days**
- Existing software can't reuse intermediate calculations

Goals:

- Reduce human labor
- Reduce total count of full 3D modeling

Solution: randomizing geometries with genetic algorithms

Takes thousands of iterations to get good results, need to be *fast*

## Problem statement

Use custom 2D finite element method kernel:

- Calculates field in 2D projection of magnet
- Single evaluation takes  $O(n^2)$  instead of  $O(n^3)$
- Caches calculations
- Uses uniformity of magnet field in target area as a quality measure
- Less precision
- Suitable for limited set of geometries (either long bodies or solids of revolution)

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 5 / 34



## Problem statement in 2D

Coil current  $J_z$  generates magnetic field ( $B$ )

Ferromagnetic core with permeability  $\mu$  forms magnetic field

- For 2D projection of long magnet,  $A_z$  (magnetic potential along  $Z$ ) fully determines magnetic field Known:

- Maxwell equations
- Object geometry and properties (e.g. permeability  $\mu$  or BH curve of material)
- Current  $J_z$
- Boundary conditions for  $A_z$  (e.g. potential is zero at  $\infty$ )

Unknown: Magnetic potential  $A_z$  in rest of domain area

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 6 / 34



## Finite element method

### Why FEM?

Solves partial differential problems in *finite* domain, e.g.

- Fluid flow
- Heat transfer
- Structural analysis

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 7 / 34



## Mikhail Sizov

## Efficient C++ implementation of custom FEM kernel with Eigen

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 8 / 34



## Finite element method

- Numerical computations (approximate)
  - Differential equations translated to **algebraic equation systems**
  - Domain geometry is approximated with finite elements (e.g. triangles)
  - Method works when differential equations on linear functions result in polynomial equations.
- 

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 9 / 34

## Why Eigen?

- Fixed and dynamic-sized matrices and vectors
- Sparse/dense matrices and vectors
- Multi-platform, multi-compiler
- Expression templates (remove temporaries, lazy evaluation)
- Optimized fixed-size matrices: (no dynamic memory allocation, unrolled loops)
- Standard numeric types (`std::complex`, `integers`, `float`)
- Interface for custom types

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 11 / 34

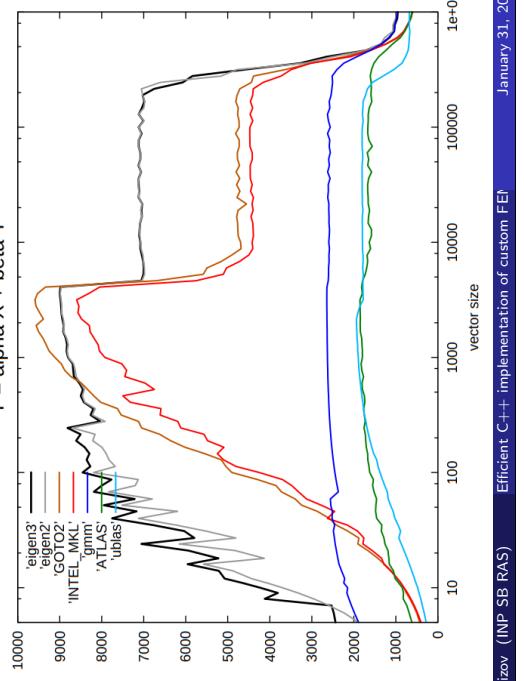
## Eigen

- Eigen is a C++ *header-only* template library for linear algebra.
- Operates with *vectors* and *matrices*
- Other popular linear algebra libraries are:
  - OpenBLAS (C++)
  - Armadillo (C++)
  - scipy (python)
  - cubLAS (CUDA)

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 10 / 34

## Why Eigen?

Benchmark provided on Eigen website (higher is better):



Efficient C++ implementation of custom FEM kernel with Eigen

## FEM first step : create a Mesh

Start with geometry approximation by simple shapes (in 2D: triangles, convex quadrilaterals)  
 Geometry approximation is called *mesh*



Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 13 / 34

## Mesh

```
Eigen::Vector2d p1 = {x1, y1}; // point coordinates
Eigen::Vector2d p2 = {x2, y2};
Eigen::Vector2d p3 = {x3, y3};
// Vector2d - static size of 2 (up to 4 defined)
// "d" stands for double (could also be int, float, double or
std::complex<T>)
```

```
// VectorXd - uses dynamic memory allocation
typedef Matrix<double, 2, 1> Eigen::Vector2d;
// Implemented as dense Matrix with size known at compile time
Eigen::Matrix<double, 2, 3> geometry;
geometry << x1, y1, x2, y2, x3, y3; // Dense matrix, other way
```

Element's geometry can also be stored in dense matrix:

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} = \begin{bmatrix} A_{z1} \\ A_{z2} \\ A_{z3} \end{bmatrix}$$

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 15 / 34

## FEM first step : create a Mesh

Each single triangle is a *finite element*, it has:

- 3 triangle vertices (nodes)
- Vertix coordinates  $(x_1, y_1), (x_2, y_2), (x_3, y_3)$
- Constant material properties within:  $J_z$  or  $\mu$
- Node values ( $A_{z1}^e, A_{z2}^e, A_{z3}^e$ )

**Target**  $A_z^e$  is **linearly interpolated between nodes**, e.g.

$$A_z^e(x, y) = a + bx + cy, \text{ more on that later}$$

Node values either known from boundary conditions or unknowns  
 Node values shared between elements ( $A_z$  is continuous function)

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 14 / 34

## Shape functions

Interpolation within element is linear. By definition it must return node values on node coordinates  $A_z^e(x_i, y_i) = A_{zi}^e$   
 Let's introduce set of 3 interpolating functions  $\Phi_i(x, y)$  (shape function), that is:

- Linear  $\Phi_i(x, y) = \alpha_i + \beta_i x + \gamma_i y$
- Equals 1 on it's own node  $\Phi_i(x_i, y_i) = 1$
- Equals 0 on other 2 nodes  $\Phi_i(x_j, y_j) = 0$ , given  $i \neq j$

Now, we can express  $A_z^e(x, y)$  as :

$$A_z^e(x, y) = [\Phi_1(x, y) \quad \Phi_2(x, y) \quad \Phi_3(x, y)] \begin{bmatrix} A_{z1} \\ A_{z2} \\ A_{z3} \end{bmatrix} = \sum_{i=1}^3 \Phi_i(x, y) A_{zi}^e$$

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 16 / 34

Efficient C++ implementation of custom FEM kernel with Eigen

Mikhail Sizov

## Shape functions(implementation)

### Shape functions(implementation)

Shape functions coefficient matrix  $\Phi_i(x, y) = \alpha_i + \beta_i x + \gamma_i y$  is known from geometry:

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \\ \gamma_1 & \gamma_2 & \gamma_3 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}^{-1}$$

Element's shape functions coefficients can be calculated from element coordinates:

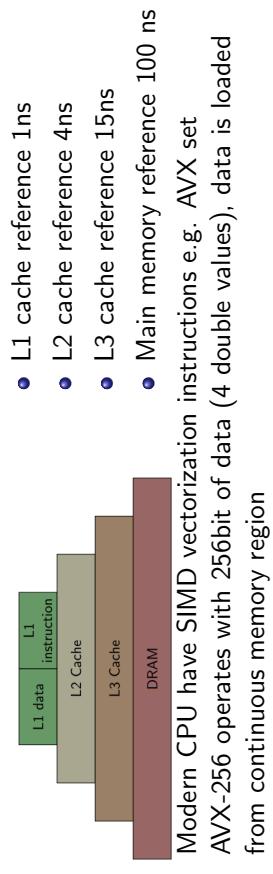
```
Eigen::Matrix<double> 3, 3, RowMajor> geometry;
// Dense 3x3 matrix to store geometry
geometry.block<3,1>(0, 0) = Vector3d::Ones(); // fill first column
geometry.block<1,2>(0, 1) = p1; // block assignments
geometry.block<1,2>(1, 1) = p2;
geometry.block<1,2>(2, 1) = p3;
auto shape = geometry.inverse();
// get actual shape coefficients
```

Specific memory layout can improve performance (ColMajor vs RowMajor):

$$mCol = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$



## Typical structure of CPU memory caches



Modern CPU have SIMD vectorization instructions e.g. AVX set  
AVX-256 operates with 256bit of data (4 double values), data is loaded  
from continuous memory region

$$C[5][6] = 41*6+42*16+43*26...$$

Cache line has 64B, data is prefetched



Mikhail Sizov (IMP SB RAS) Efficient C++ implementation of custom FEM

January 31, 2019 19 / 34

Mikhail Sizov (IMP SB RAS) Efficient C++ implementation of custom FEM

January 31, 2019 20 / 34

Mikhail Sizov

Efficient C++ implementation of custom FEM kernel with Eigen

## Shape functions(implementation)

Best option, no extra cache misses:

RowMajor	<table border="1"><tr><td>41</td><td>42</td><td>42</td><td>43</td><td>43</td><td>44</td><td>44</td><td>45</td></tr></table>	41	42	42	43	43	44	44	45
41	42	42	43	43	44	44	45		
ColMajor	<table border="1"><tr><td>6</td><td>16</td><td>26</td><td>36</td><td>46</td><td>56</td><td>66</td><td>76</td></tr></table>	6	16	26	36	46	56	66	76
6	16	26	36	46	56	66	76		

Common variant, matrix type is the same, second matrix has cache misses

RowMajor	<table border="1"><tr><td>41</td><td>42</td><td>42</td><td>43</td><td>43</td><td>44</td><td>44</td><td>45</td></tr></table>	41	42	42	43	43	44	44	45
41	42	42	43	43	44	44	45		
RowMajor	<table border="1"><tr><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td></tr></table>	6	7	8	9	10	11	12	13
6	7	8	9	10	11	12	13		

Worst scenario, every hit is cache miss

ColMajor	<table border="1"><tr><td>41</td><td>51</td><td>61</td><td>71</td><td>81</td><td>91</td><td>2</td><td>12</td></tr></table>	41	51	61	71	81	91	2	12
41	51	61	71	81	91	2	12		
RowMajor	<table border="1"><tr><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td></tr></table>	6	7	8	9	10	11	12	13
6	7	8	9	10	11	12	13		

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 21 / 34

## System total energy

Shape functions + potential values at nodes define  $A_z$  in domain (solves problem)  
*How to calculate unknown  $A_z$  at nodes?*

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 22 / 34

## System total energy

Let's write potential energy per unit length of magnet field for an element:

$$U(A_z^{el}(x, y)) = \frac{1}{4\pi\mu} \int \left| \nabla A_z^{el}(x, y) \right|^2 dS + \frac{1}{c} \int A_z^{el}(x, y) \cdot J_z dS$$

Substituting shape functions  $\Phi_i$  and vector of node values  $\vec{A}_z^{el}$  into  $A_z^{el}(x, y)$ , we would get:

$$U(A_z) = \frac{1}{4\pi\mu} \vec{A}_z^{el T} C \vec{A}_z^{el} + \frac{S}{3c} J_z \sum_{i=1}^3 A_{zi}^{el}$$

$\mu$  – element permeability

$S$  – element square

$C$  – element coupling matrix (from shape functions)

$$C = \begin{bmatrix} \beta_1 & \gamma_1 \\ \beta_2 & \gamma_2 \\ \beta_3 & \gamma_3 \end{bmatrix} \begin{bmatrix} \beta_1 & \beta_2 & \beta_3 \\ \gamma_1 & \gamma_2 & \gamma_3 \end{bmatrix}$$

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 23 / 34

Mikhail Sizov

Efficient C++ implementation of custom FEM kernel with Eigen

## Global coefficient matrix

Global coefficient matrix is sparse (real example for 9x9 regular mesh):

$$C = \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & -4 & 1 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & -2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -4 & 2 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 2 & -8 & 2 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 2 & -8 & 2 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & -8 \end{bmatrix}$$

- ➊ Most values are zeros
- ➋ Symmetric!
- ➌ Typical node count for 2D is ~250k, about 1k of non-zeros

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 / 25

## Global coefficient matrix

Dedicated types exist for efficiently storing sparse matrices:

```
typedef Eigen::Triplet<double> T;
std::vector<T> tripletList;
tripletList.reserve(estimation_of_entries);
for (...) {
    // ...
    tripletList.push_back(T(i, j, v_ij));
}
SparseMatrixType mat(rows, cols);
mat.setFromTriplets(tripletList.begin(), tripletList.end());
// mat is ready to go!
```

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 / 26

## Sparse matrix memory layout:

Values:	22	7	3	5	14	1	17	8
InnerIndices:	1	2	0	2	4	2	1	4
OuterStarts:	0	2	4	5	6	8		

- ➊ Values: stores the coefficient values of the non-zeros.
- ➋ InnerIndices: stores the row (in row major specialization) indices of the non-zeros.
- ➌ OuterStarts: stores for each column (resp. row) the index of the first non-zero in the previous two arrays.

0	3	0	0	0
22	0	0	17	
7	5	0	1	0
0	0	0	0	
0	0	14	0	8

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 / 27

## Mikhail Sizov

## Global coefficient matrix

Dedicated types exist for efficiently storing sparse matrices:

```
typedef Eigen::Triplet<double> T;
std::vector<T> tripletList;
tripletList.reserve(estimation_of_entries);
for (...) {
    // ...
    tripletList.push_back(T(i, j, v_ij));
}
SparseMatrixType mat(rows, cols);
mat.setFromTriplets(tripletList.begin(), tripletList.end());
// mat is ready to go!
```

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 / 26

## Obtaining solution

Let's use physics principle of minimal potential energy:  
 From all possible values of free  $A_{zi}^e$  parameters, we need to choose set that gives minimal energy, so derivative of energy by each of  $A_{zi}^e$  is zero:

$$\frac{\partial U}{\partial A_{zi}^e} = [...] = \frac{\partial U}{\partial A_{zi}} = 0$$

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 / 28

## Efficient C++ implementation of custom FEM kernel with Eigen

## Obtaining solution

$k$  - global node index [1.. $n$ ]

$$\frac{\partial U}{\partial A_{zk}} = M_k \sum_{j=1}^n A_{zj} C_{kj} + J_k = 0$$

Reorder node names so that unknown potential values have indexes [1.. $m$ ] and known have indexes [ $m+1..n$ ]

$$\sum_{j=1}^m A_{zj} C_{kj} = - \sum_{j=m+1}^n A_{zj} C_{kj} - \frac{J_k}{M_k}$$

Can be viewed as matrix equation:

$$Ax = B$$

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 29 / 34

## Obtaining solution

Eigen library has solver concept that either solves linear equation system or returns an error

Has several solvers that are tailored to different problems, e.g.

**SimplicialLDLT** – for medium sparse problems (2D Poisson)

**ConjugateGradient** – for large symmetric problems (3D Poisson)

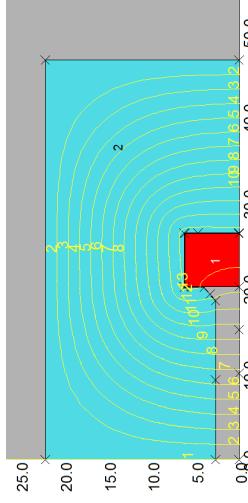
**SparseQR** – for least squares problems

If solution exists, such solver would return all node values for magnetic potential

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 30 / 34

## Obtaining solution

After solving linear equation system, we get all node values and can also calculate magnetic field



```
//define a solver
Eigen::SimplicialLDLT<Eigen::SparseMatrix<double>> solver;
solver.analyzePattern(A); // checks for non-zero values
solver.factorize(A);
if(solver.info()!=Success) {
    // decomposition failed
    return;
}
x1 = solver.solve(b1);
if(solver.info()!=Success) {
    // solving failed
    return;
}
x2 = solver.solve(b2); //can be used for different right parts
```

Mikhail Sizov (INP SB RAS) Efficient C++ implementation of custom FEM January 31, 2019 31 / 34

Mikhail Sizov

Efficient C++ implementation of custom FEM kernel with Eigen

## Summary

## Summary

Using 2D FEM calculations, we can speed up magnets design in early stage, and people are working in almost real time.

One single calculation with 500x500 nodes takes fractions of seconds, while full 3D simulation could take several days.

To calculate fields using finite element method:

- ➊ Generate triangle mesh for problem domain
- ➋ Construct linear interpolation functions for each triangle
- ➌ Express total energy of system in terms of interpolation functions and node values
- ➍ Calculate integrals or differentials of interpolation functions
- ➎ At this step energy is written as polynomial function of node values
- ➏ Apply principle of minimal energy (minimize energy with respect to each unknown node value)
- ➐ Compose system of linear equations (with unknown node values)
- ➑ Solve resulting linear equation system





For more information on the  
CERN School of Computing:  
<http://cern.ch/csc>

