Software and Computing in the Era of Artificial Intelligence

Lindsey Gray

2nd COFI Advanced Instrumentation and Analysis Techniques School

11 December 2023

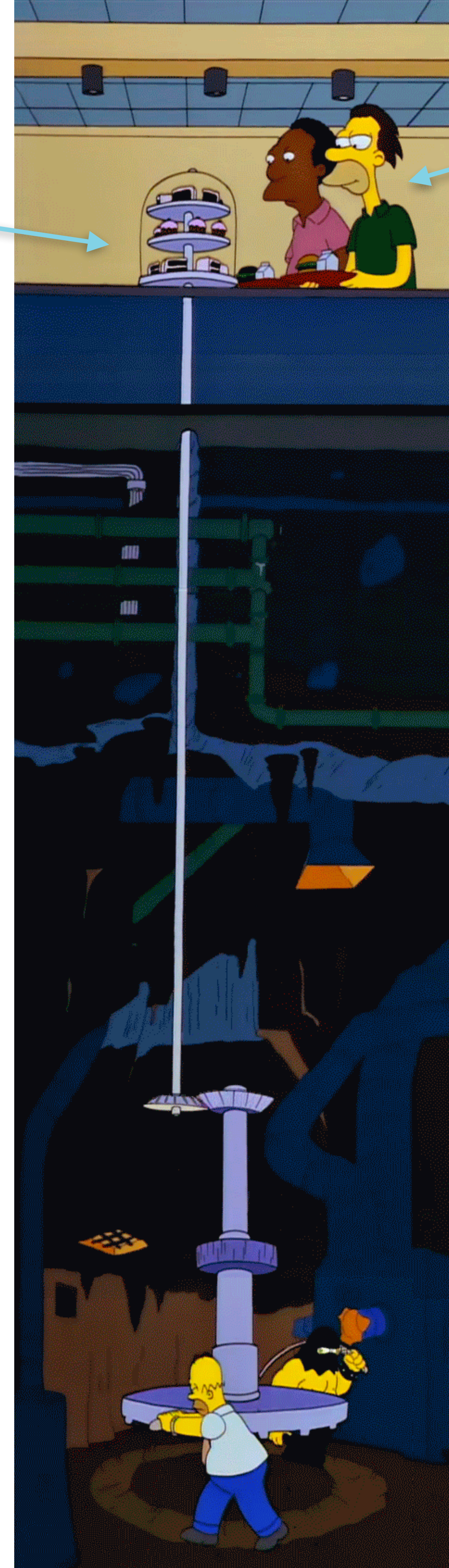
About Me!

- Fermilab - Staff Scientist, U.S. CMS L2 for Software and Computing R&D
- Physics
  - Multi-Vector Boson physics, usually involving a photon, occasionally jet final states
  - EFT measurements past and present, from modified vertex functions to the more refined modern approaches
  - Occasional forays into final states with boosted jets (didn’t really stick)
- Software
  - CMS E/gamma Reconstruction and Particle Flow
  - End-to-end HGCAL reconstruction with graph neural networks
  - Coffea - numpy-like analysis for HEP - more recently coffea 2023 dask migration
  - Infrastructure in/around analysis facilities:
    - Nvidia triton, dask/distributed, dask-awkward, dask-histogram, caches, network scheduling
- Hardware(-ish)
  - Smartpixels (neural networks in asics for on-pixel-sensor reconstruction)
    - Finding efficient neural networks that can produce understood error predictions
  - Precision timing detectors (CMS MIP Timing Detector)
    - Developed 4D vertexing algorithm, shaped physics case, initial detector design considerations, technical proposal, TDR, beam tests
- Future colliders: C3 hardware/software/analysis

Outline

- Needs of ML software and how that informs computing
  - Ingredients of ML computation in: basic neural networks, convolutions, graphs, etc.
  - Software ecosystem to build models and workflows
  - Accelerating ML workflows
  - Training and Inference
  - Modern data and model sizes (in and outside of science)
  - Scaling ML workflows
  - Impact of ML's dominance on modern computing hardware
- ML in High Energy Physics (HEP)
  - Interfacing HEP data to ML software systems
  - Typical use cases of ML in HEP
  - Case studies: non-typical / forward-looking uses of ML in HEP
    - Machine learned Particle Flow reconstruction
    - Single-hit track reconstruction in next-generation pixel detectors
    - Calorimeter simulation with diffusion models
    - Systematics-aware differentiable analysis
  - A bit of a vision statement…
- Summary of Promising Themes and Concluding Remarks

Machine Learning



Machine Learning Practitioners

Computing

First, A Note: Getting from Theory to Practice

- What does the following python expression evaluate to?
  - `sum([1/x**2 for x in range(1,10001)]) - sum(reversed([1/x**2 for x in range(1,10001)]))`

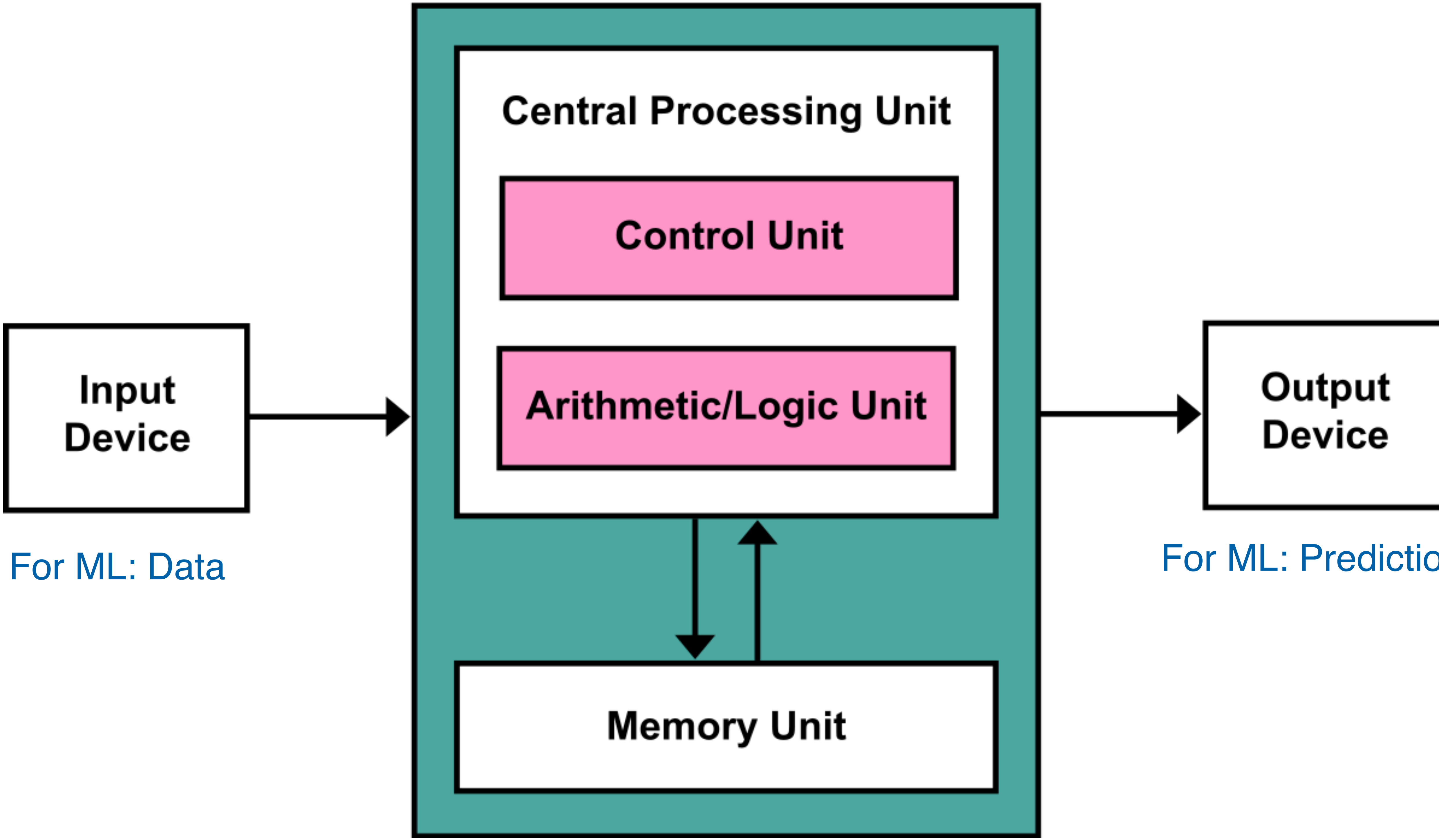
First, A Note: Getting from Theory to Practice

- What does the following python expression evaluate to?
  - sum([1/x**2 for x in range(1,10001)]) - sum(reversed([1/x**2 for x in range(1,10001)]))
  - ~5e-15 - floating point arithmetic is neither commutative nor associative!

- ML pedagogy often done in 'pristine' mathematical settings
  - Infinitely differentiable activation functions, compact sets for domain and range
  - Real numbers commute under addition, multiplication…
  - IEEE 754 (floating point arithmetic specification) is not the real numbers!
  - ReLU isn't a smooth function!
  - But… you've seen Nick giving demos for two days that function!?

- Determining effective, robust, and scalable computational techniques is vital for ML to work in practice
  - A lot of this is well hidden in numerical libraries, ML frameworks, and it should be
  - While this talk will cover higher level computational techniques, there's quite some depth to how we make machine learning software that actually works in practice

The Von Neumann Computer (1945)

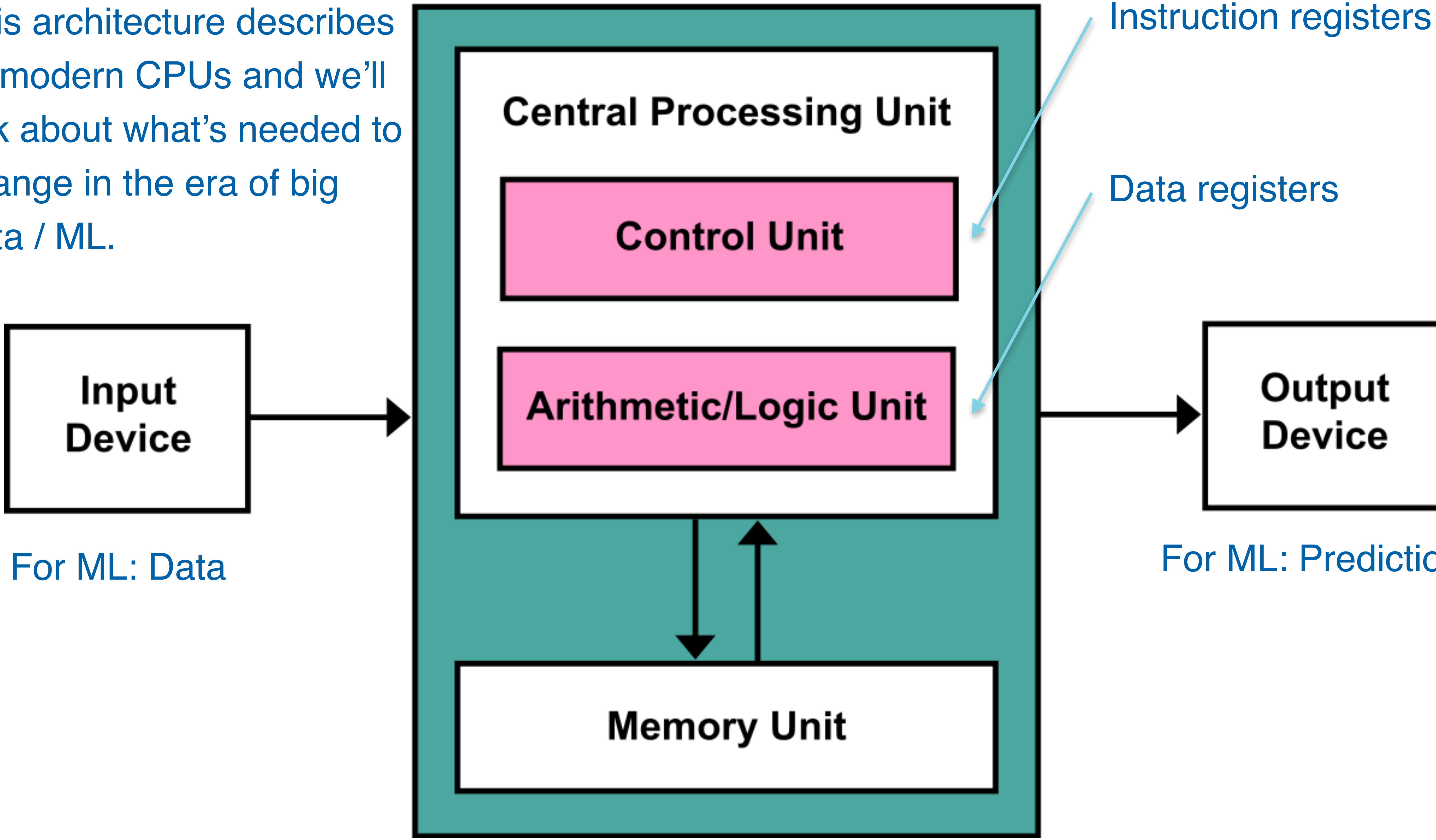


For ML: Data

For ML: Predictions

The Von Neumann Computer (1945)

This architecture describes all modern CPUs and we'll talk about what's needed to change in the era of big data / ML.



Instruction registers

Data registers

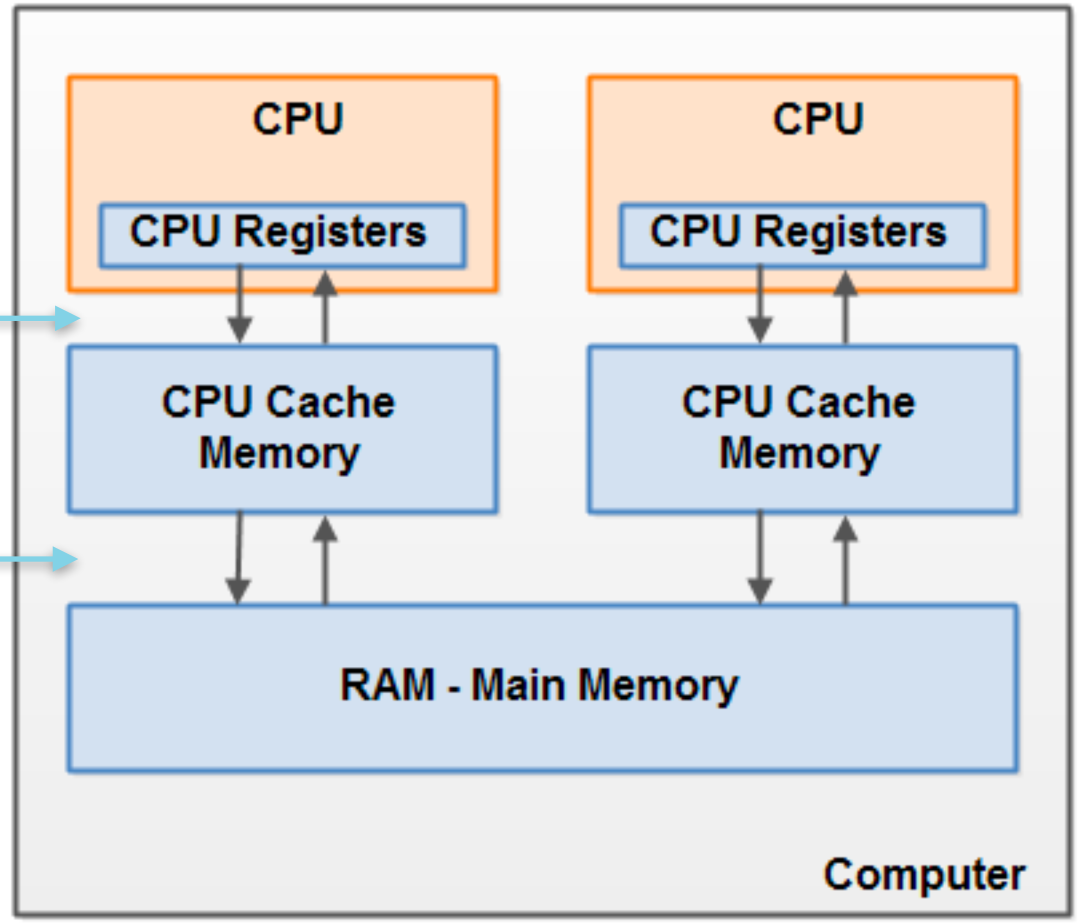
For ML: Data

For ML: Predictions

A note on that "memory unit" in modern computers

Cheap and fast!

Expensive and slow!



- Modern CPUs (and other processors) depend vitally on tiered memory structures to achieve efficient processing
  - This memory structure + the piecewise register-based execution defines modern computing
- Only in the past few years has this paradigm started to crack, due to ML

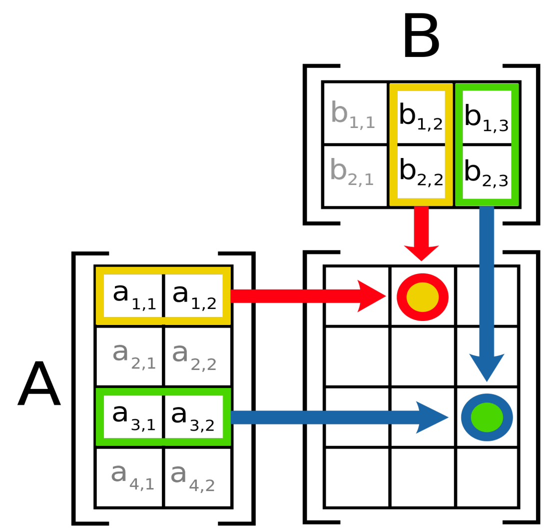
Ingredients of Fast ML Computation - General

- Pointwise, “vectorized”, function application

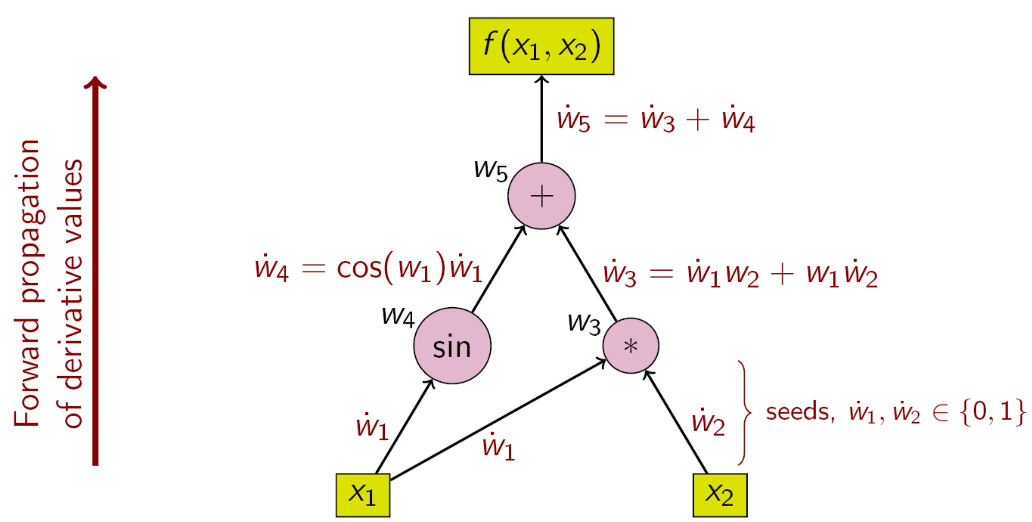
```
np.sin([0, π/2, π])
```

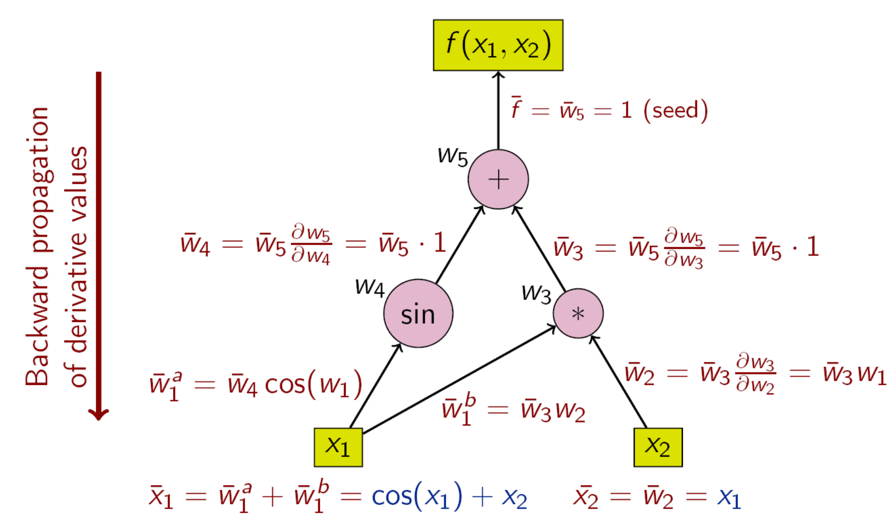
```
[sin(0), sin(π/2), sin(π)]
```

- Matrix Multiplication



- Automatic Differentiation



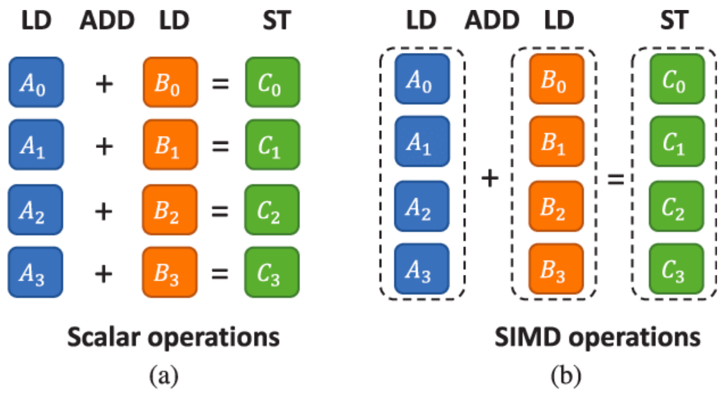


Ingredients of Fast ML Computation - General

- Pointwise, “vectorized”, function application

SISD - single instruction single data `np.sin([0, π/2, π])`

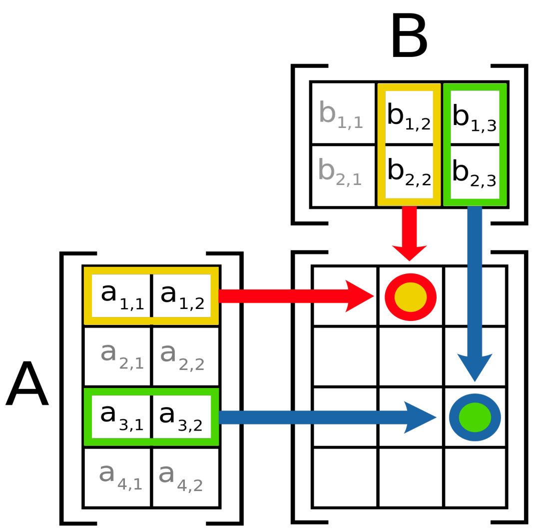
SIMD - single instruction multiple data `[sin(0), sin(π/2), sin(π)]`



- Matrix Multiplication

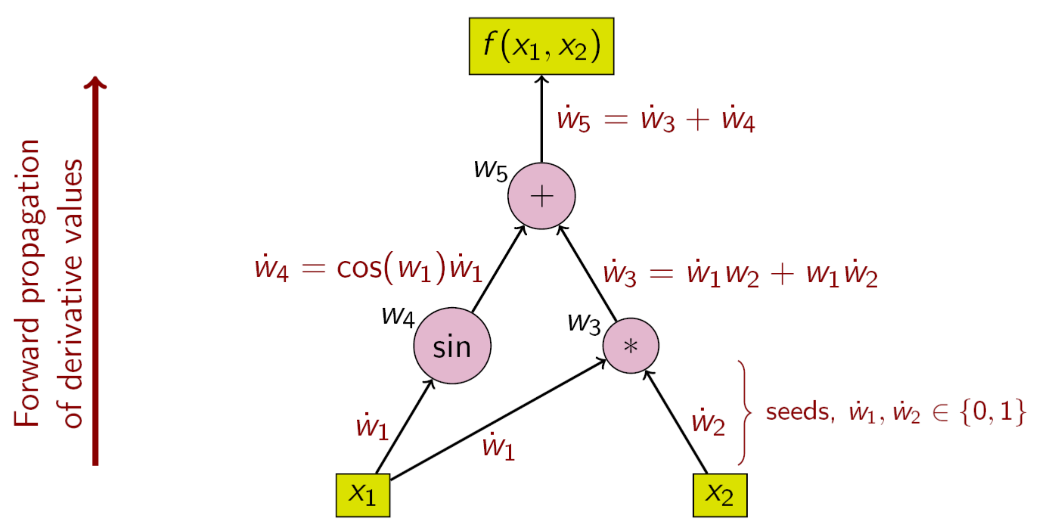
  Can think of as vectorized application of dot products.

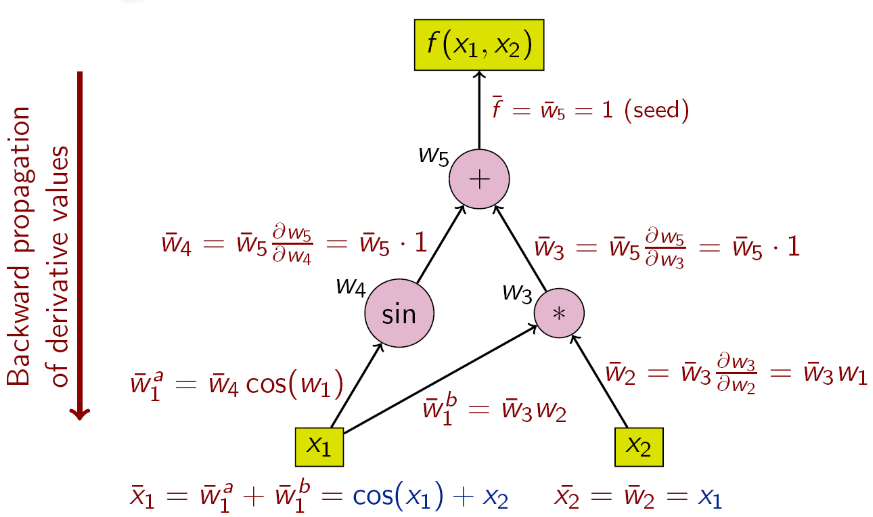
  Details of MatMul fill a lecture.



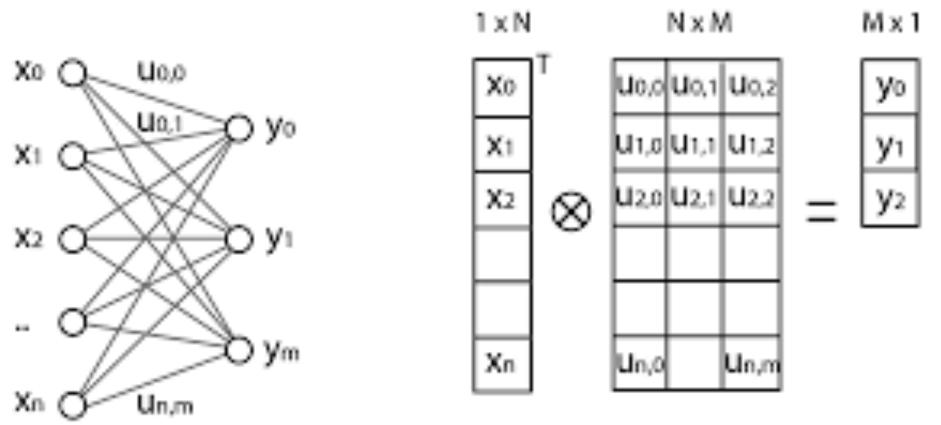
- Automatic Differentiation

Once functions known, calculation on data also vectorized.



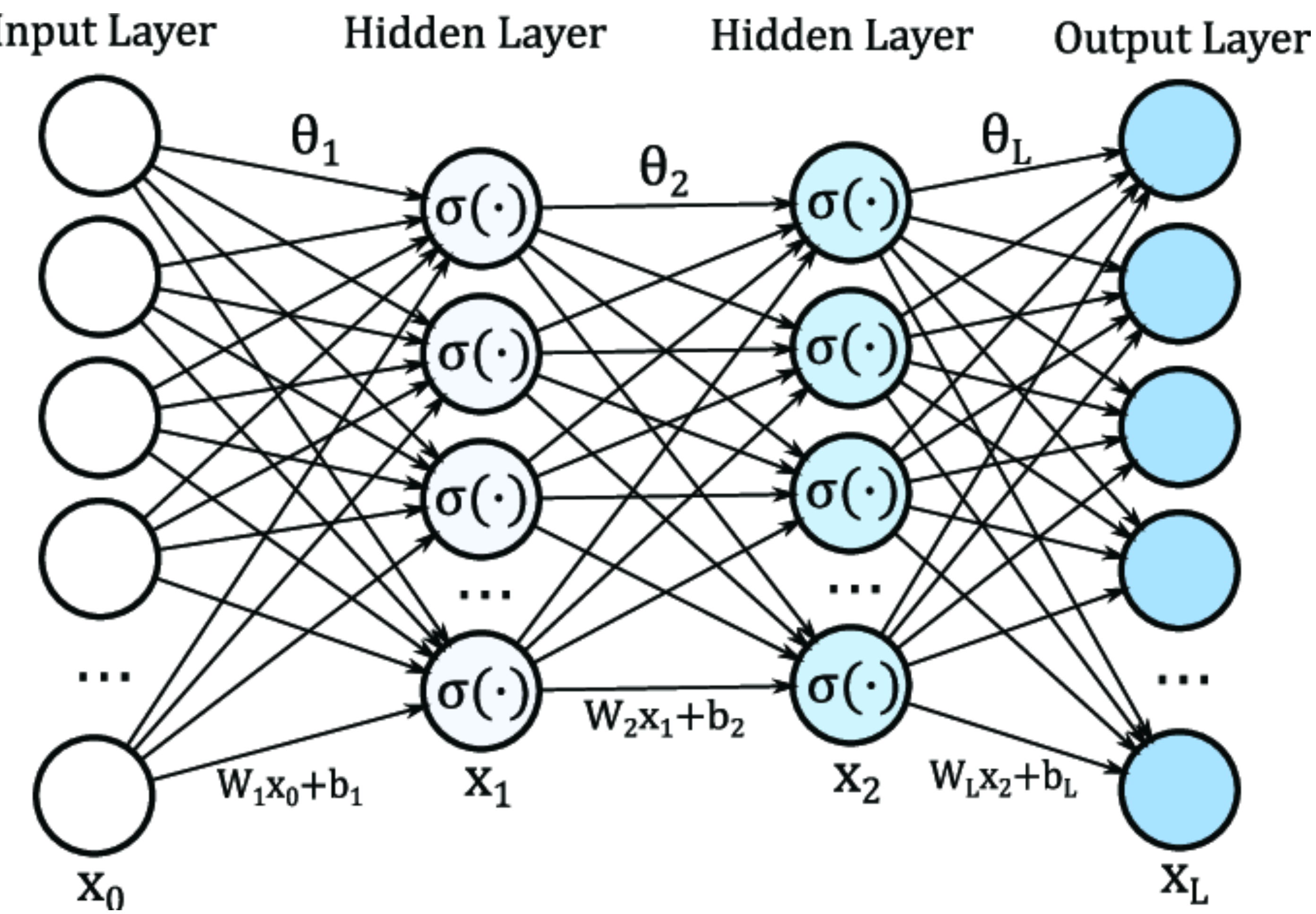


Ingredients of ML Computation: Fully Connected Networks



a) Fully connected layer

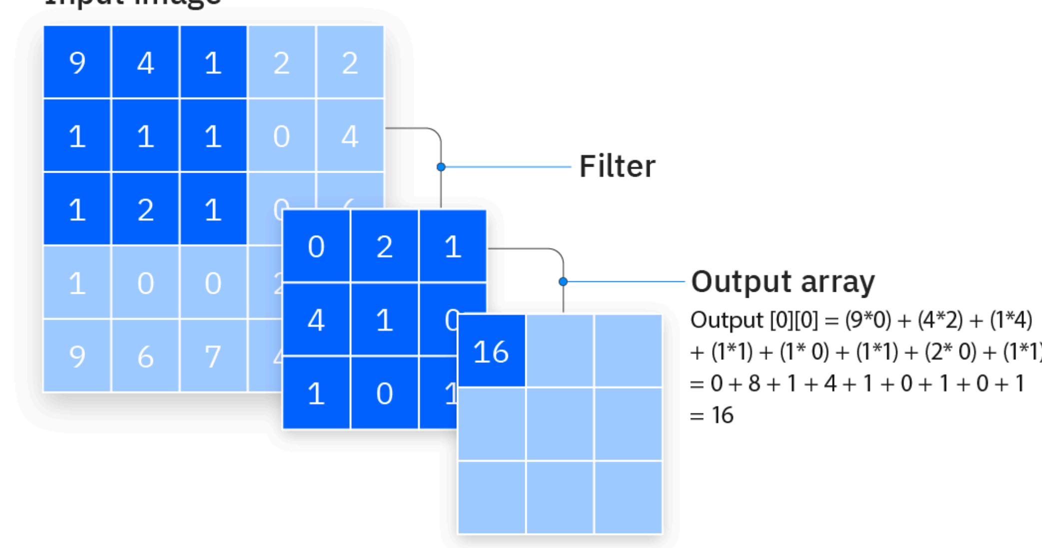
b) Expressed as a tensor product

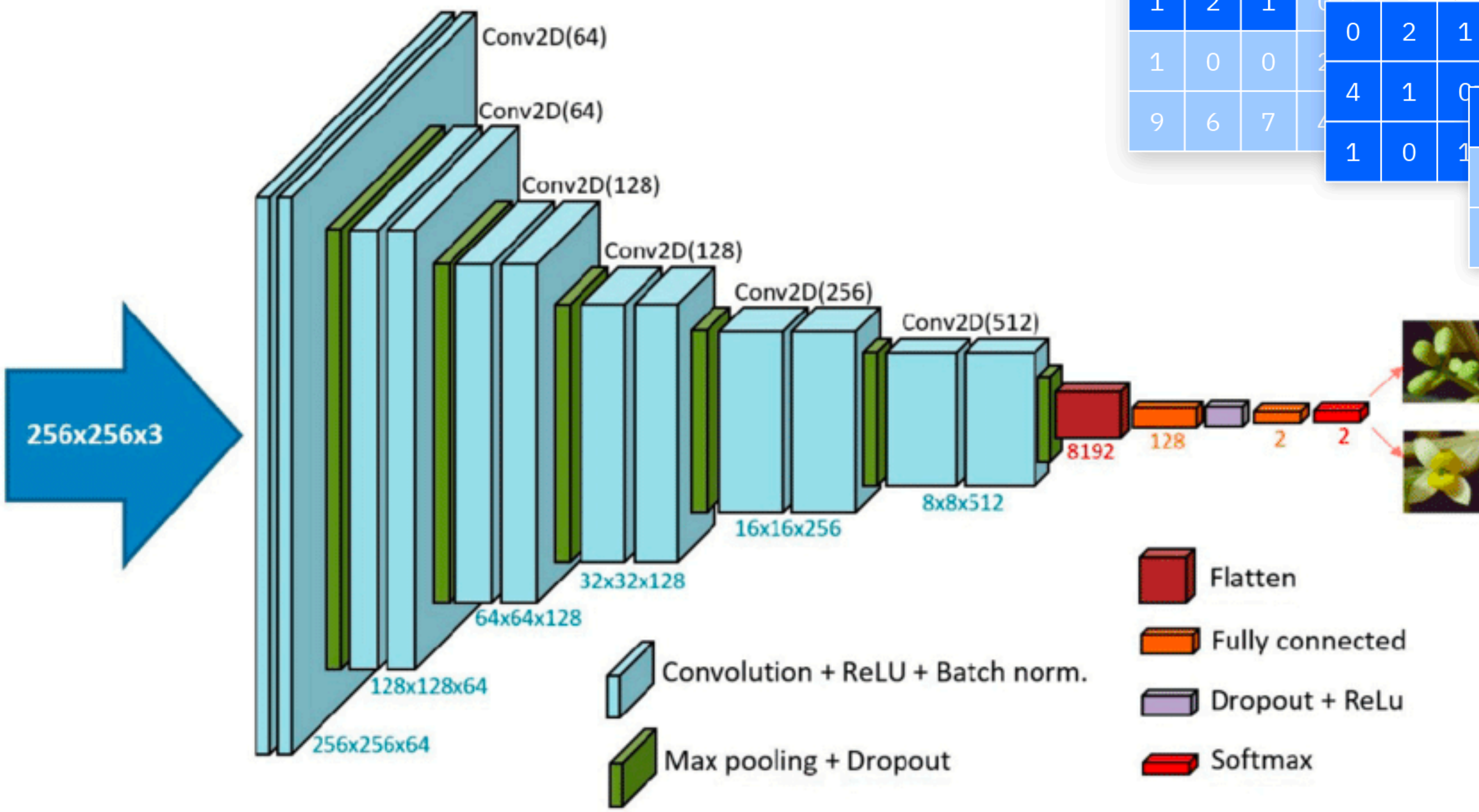


- Each layer is a matrix multiplication passed through a vectorized evaluation of the activation, defined recursively from inputs to outputs
- Manifestly SIMD computational structure!

Ingredients of ML Computation: Convolutional Networks

More detail: C. Adam's talk tomorrow





- Pointwise multiplication of convolutional filter by patch of image
  - Iterate over each point of the image (adjusting for padding)

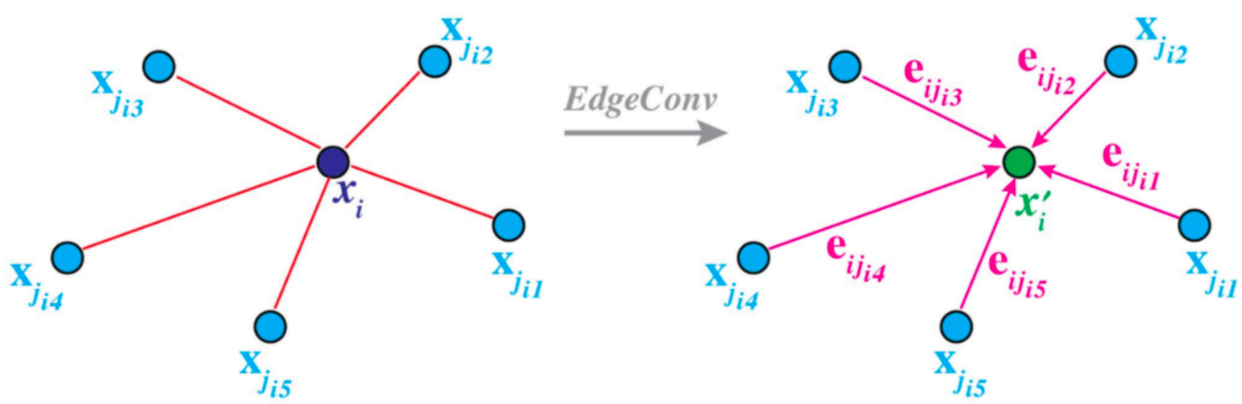
- This can be organized into efficient staging of data from memory to processor

explainer for convolutions: https://www.youtube.com/watch?v=KuXjwB4LzSA

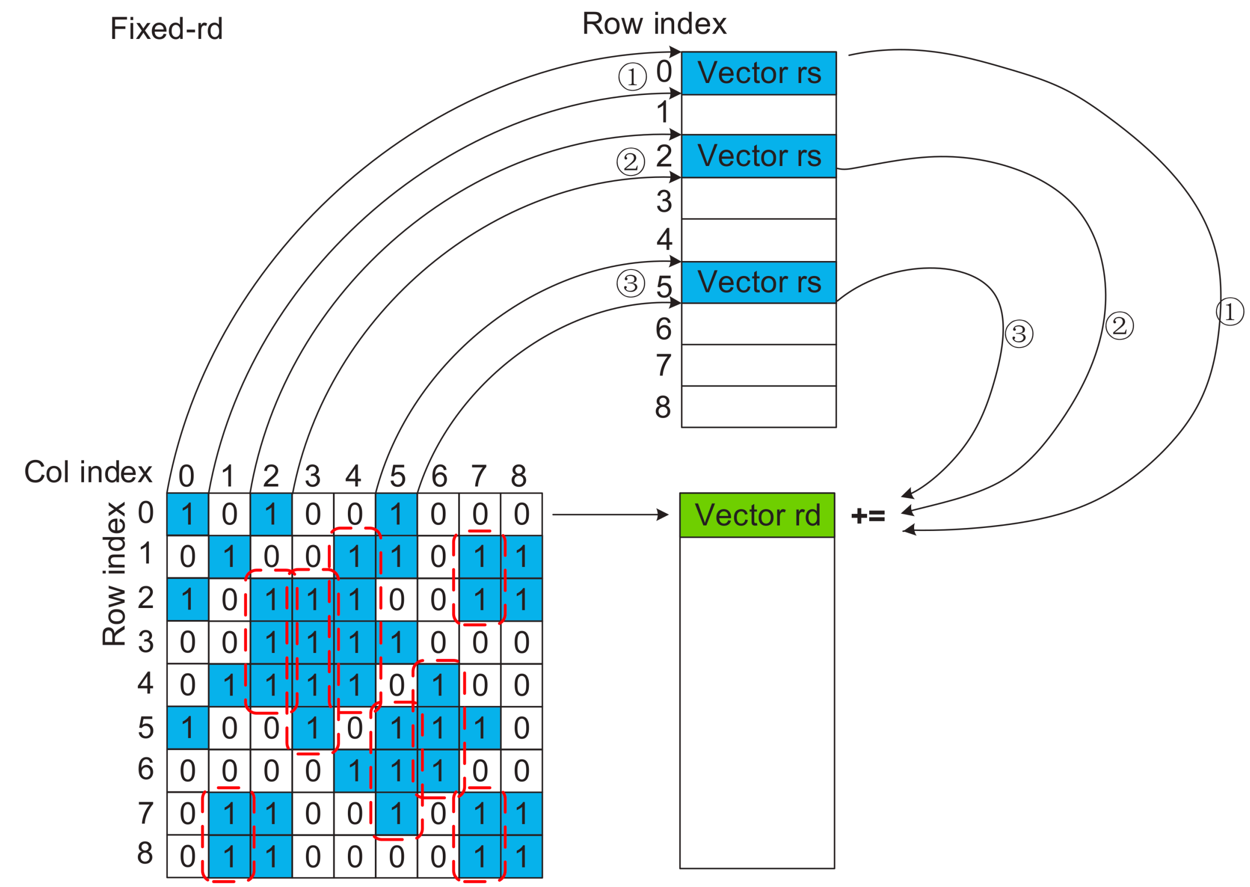
Ingredients of ML Computation: Message Passing Graph Networks

More detail: C. Adam’s talk tomorrow

Edges in a graph correspond to indices into a vector of input data



Most graph neural networks implemented in “message passing” formalism



Sparse matrix multiplication is split into vector additions

- Selecting random subsets of input data into messages along the graph is fundamentally inefficient on CPUs!
  - Does not apply to non-message passing GNNs but these are very new!

- Long story short on a CPU this has a high rate of pulling from slow memory!

ML Software Ecosystem for Building and Training ML Models

Lower Level

TensorFlow

PyTorch

scikit learn

Tensorflow GNN

torchvision

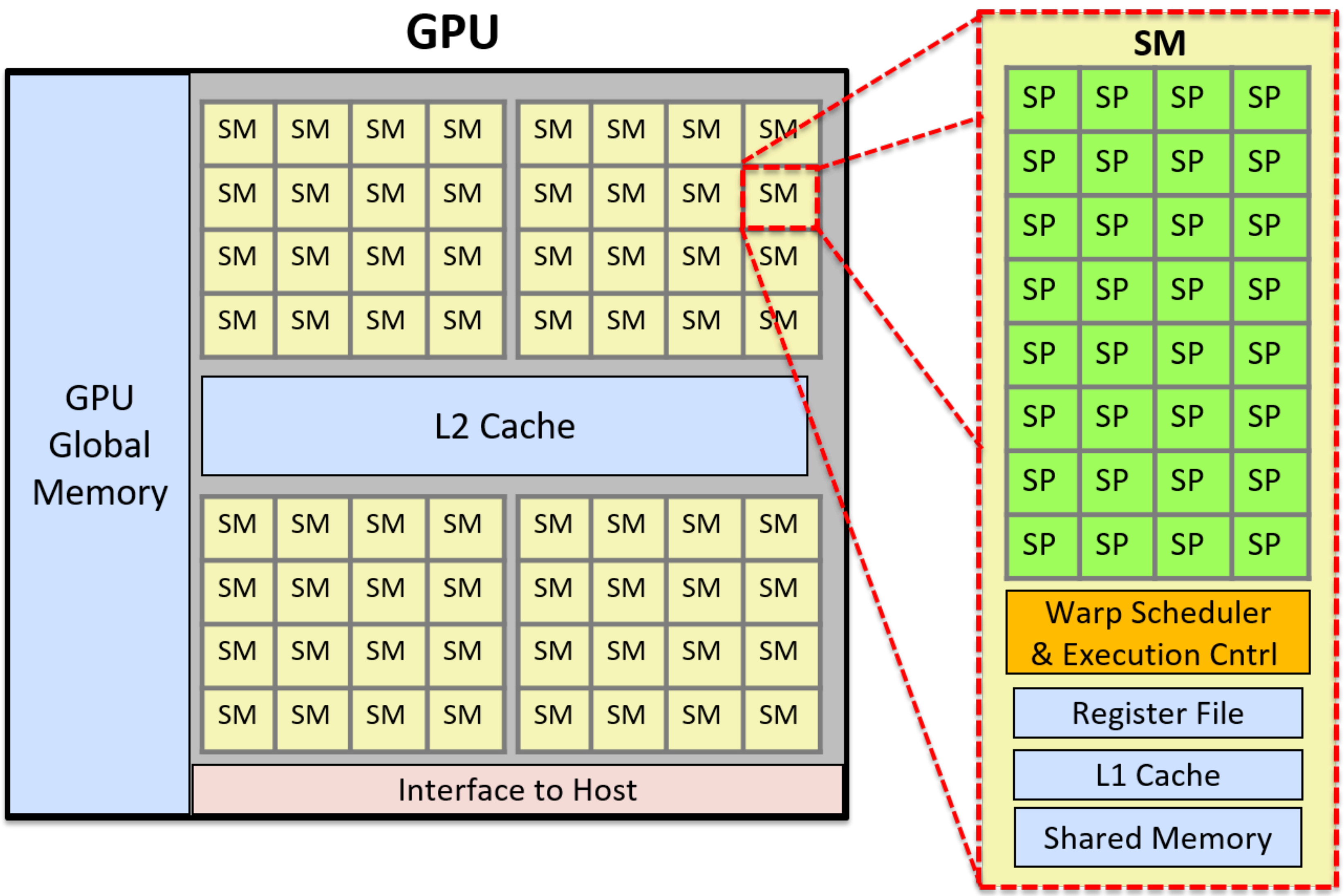
PyTorch geometric

PyTorch Lightning

Keras

Higher Level

Basics of ML Acceleration Hardware



SISD - single instruction single data

SIMD - … multiple data

SIMT - … multiple threads

- Slight departure from Von Neumann computer
  - Individual processing cores are significantly simpler than a CPU core
  - Cores heavily share memory, memory access heavily optimized for specific patterns
- GPU architecture is much more well adapted to matrix multiplication / ML!
  - be aware: GPUs tend to do computations in more narrow floating point types

The Mechanics of ML Training

For all available data * Epochs

Load model

Load training data into system memory from disk (get more if exhausted)

Load validation data into system memory from disk (“…”)

For all mini batches in loaded data

Load a mini-batch of data for training

Evaluate model outputs and forward gradients

Accumulate backwards gradient

Update model parameters

Load a mini-batch of data for validation

Evaluate model outputs, accuracy metrics

- Rules of thumb:
  - Access or move the data the least amount of times during training (memory access slow)
  - Memory size of model + gradients << amount of data held in memory (“…”)

- This tends to be encapsulated in tools like PyTorch / Tensorflow
  - And higher-level tools like PyTorch Lightning, but worth it to check your problem fits!

The Mechanics of ML Inference

Load model

While you are receiving data

Load a batch of data

Evaluate model outputs

Dispatch for use

- Significantly simpler workflow than inference
  - Can optimize away the evaluation of gradients
  - Manifestly pleasantly parallel if model fits in a single GPU

- Inference-only workflows are practically the last thing done in ML workflows
  - In HEP inference ends up being the predominant way in which a model is used
  - e.g. we train heavy-flavor tagging on millions of events, and evaluate on billions

- Main consideration here is whether latency or throughput matters!
  - This is incredibly use-case specific and has some interesting logical extrema…

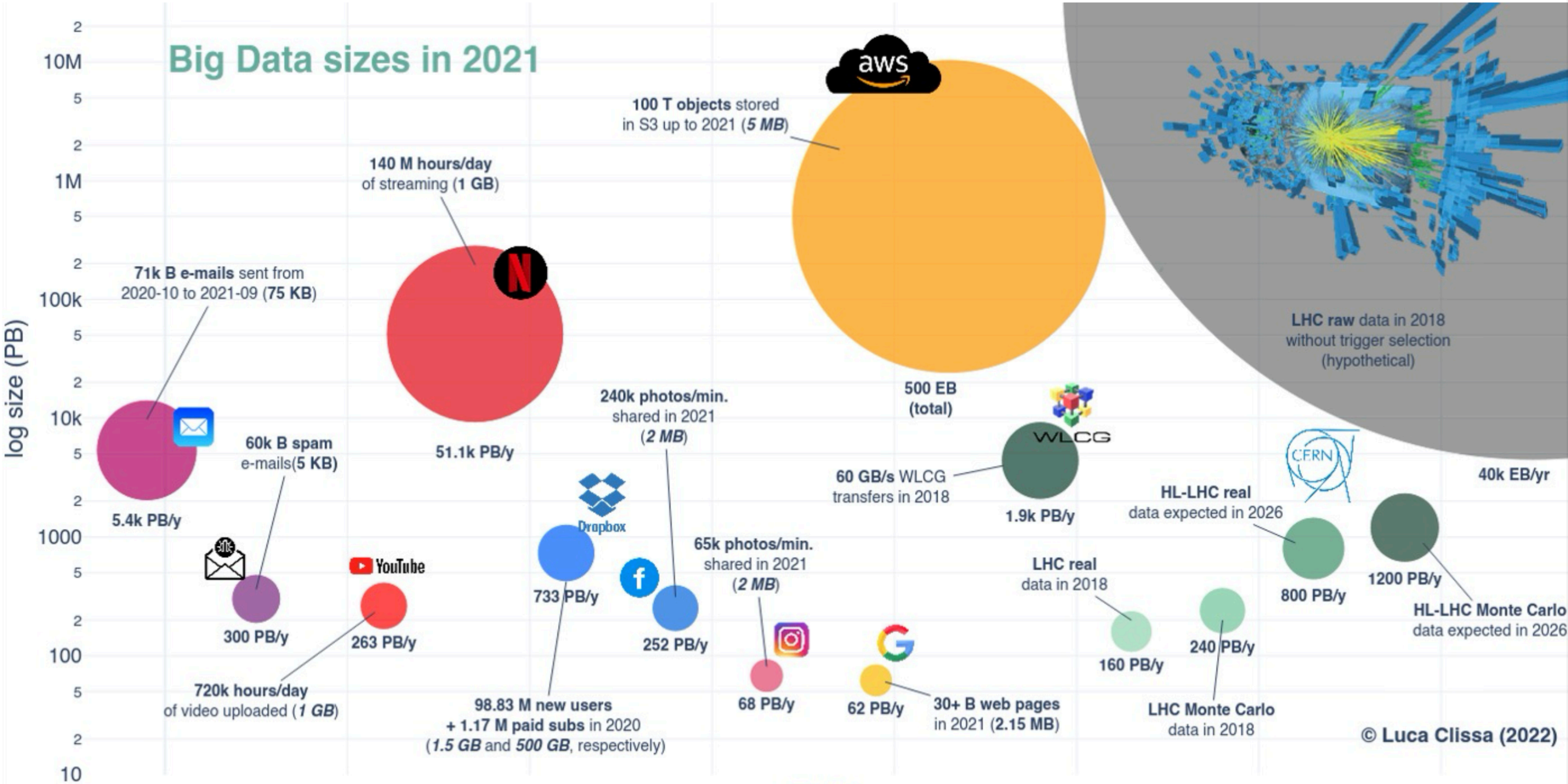
So where are all the bottlenecks?

- Memory bandwidth to processor first and foremost
  - Modern DDR4-5 system memory 50-100 GB/s RAM to CPU, depending on config
  - Modern DDR6 GPU memory > 700 GB/s, depending on memory bus bit width
  - From a memory transfer perspective alone the GPU is superior

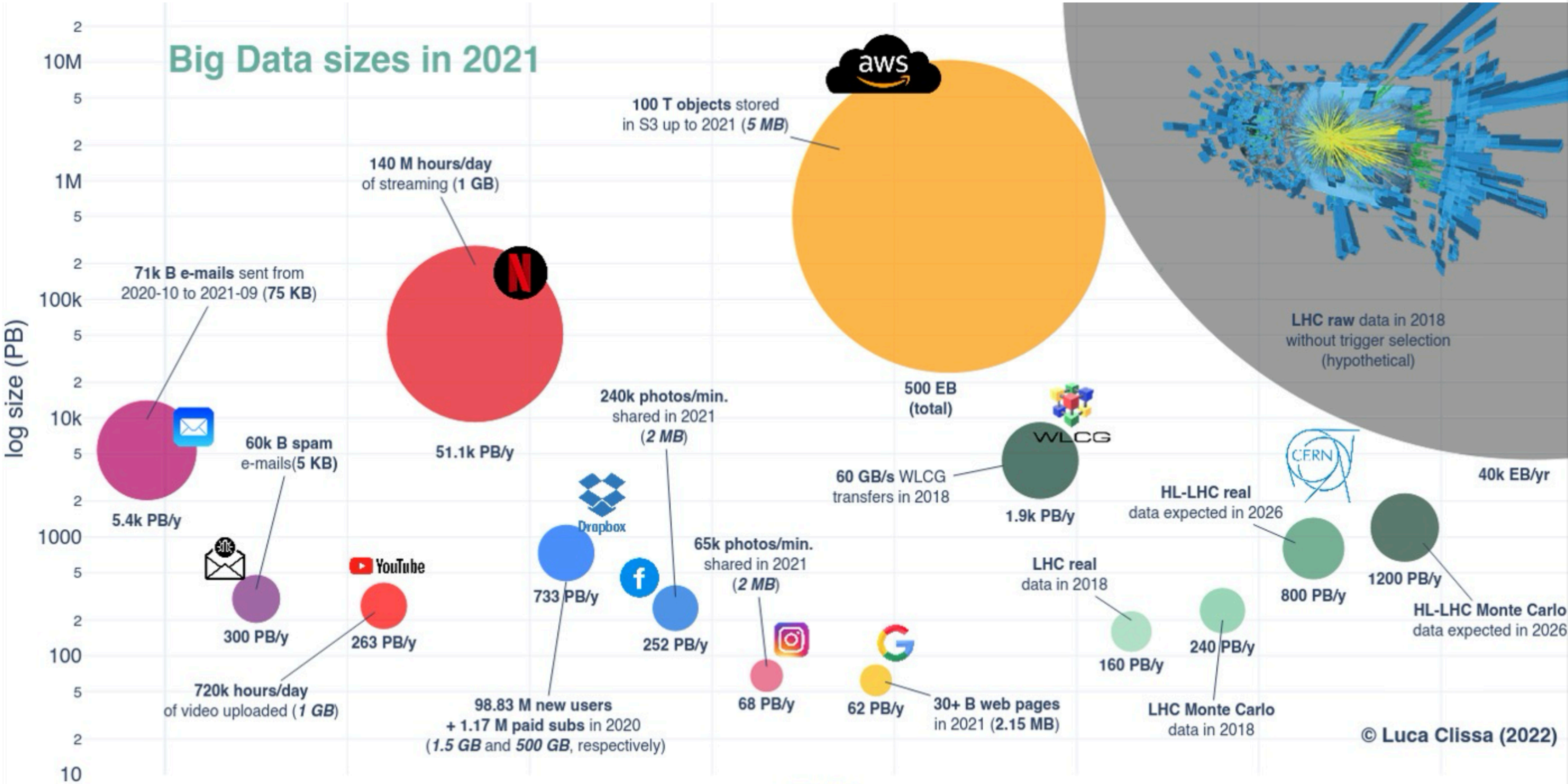
- Logic to do arithmetic
  - Modern CPUs have up to 32 SIMD 4-way units per core and O(10) cores
  - Modern GPUs have 1 floating point unit per core and > 6500 cores
  - GPUs not only have the memory bandwidth but the cores to utilize it

- GPUs a clear win on average - however - keep in mind
  - If you are randomly selecting large amounts of data (as in a graph network)
  - You are training on enormous amounts of data (large examples, many examples)
  - Models that are larger than a single GPU’s memory
  - If you need your model to be evaluated exceptionally quickly
    - i.e. ultra-low latency - GPUs get their speed from throughput

(Really) Big Data / Why we need to remove the bottlenecks

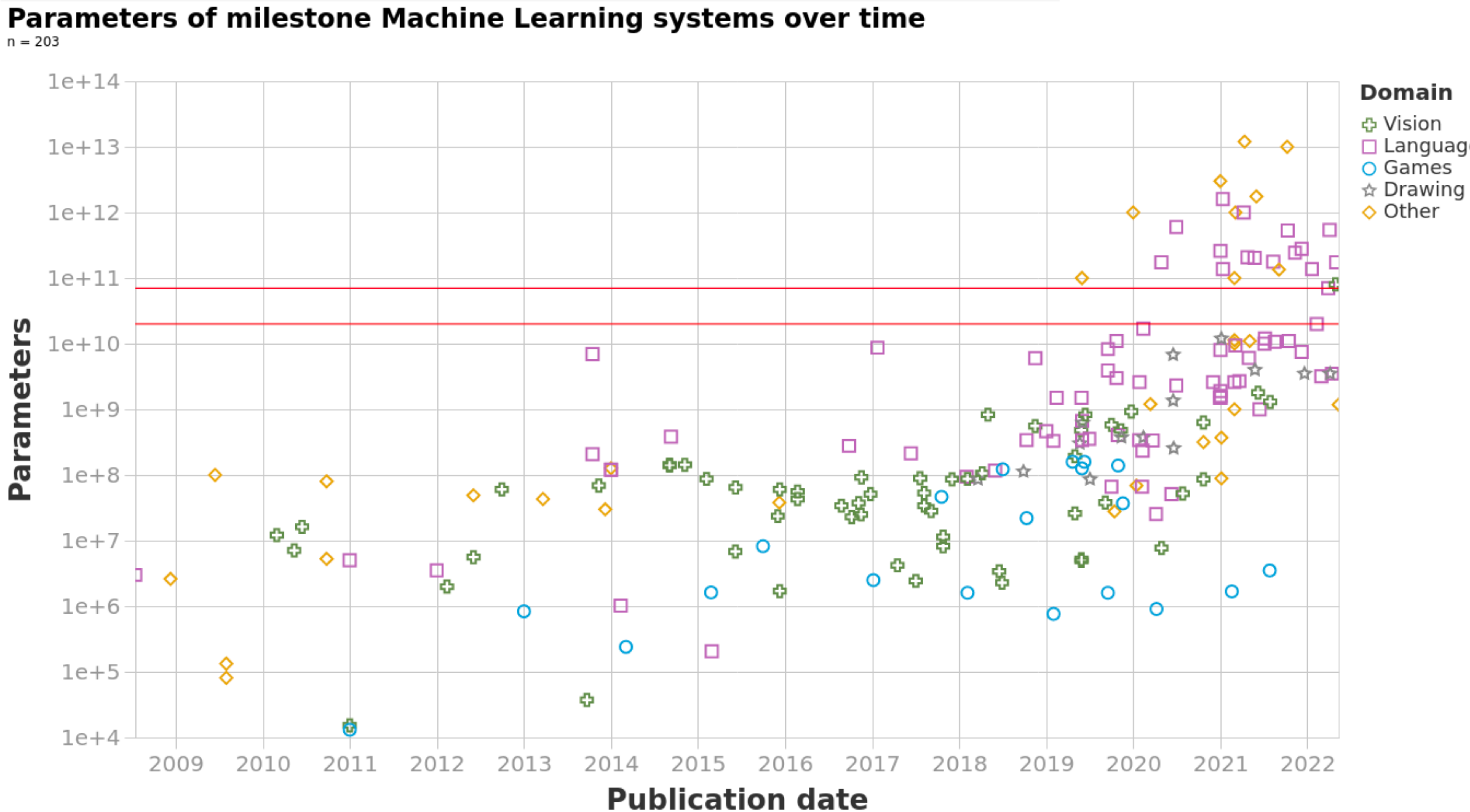


(Really) Big Data / Why we need to remove the bottlenecks

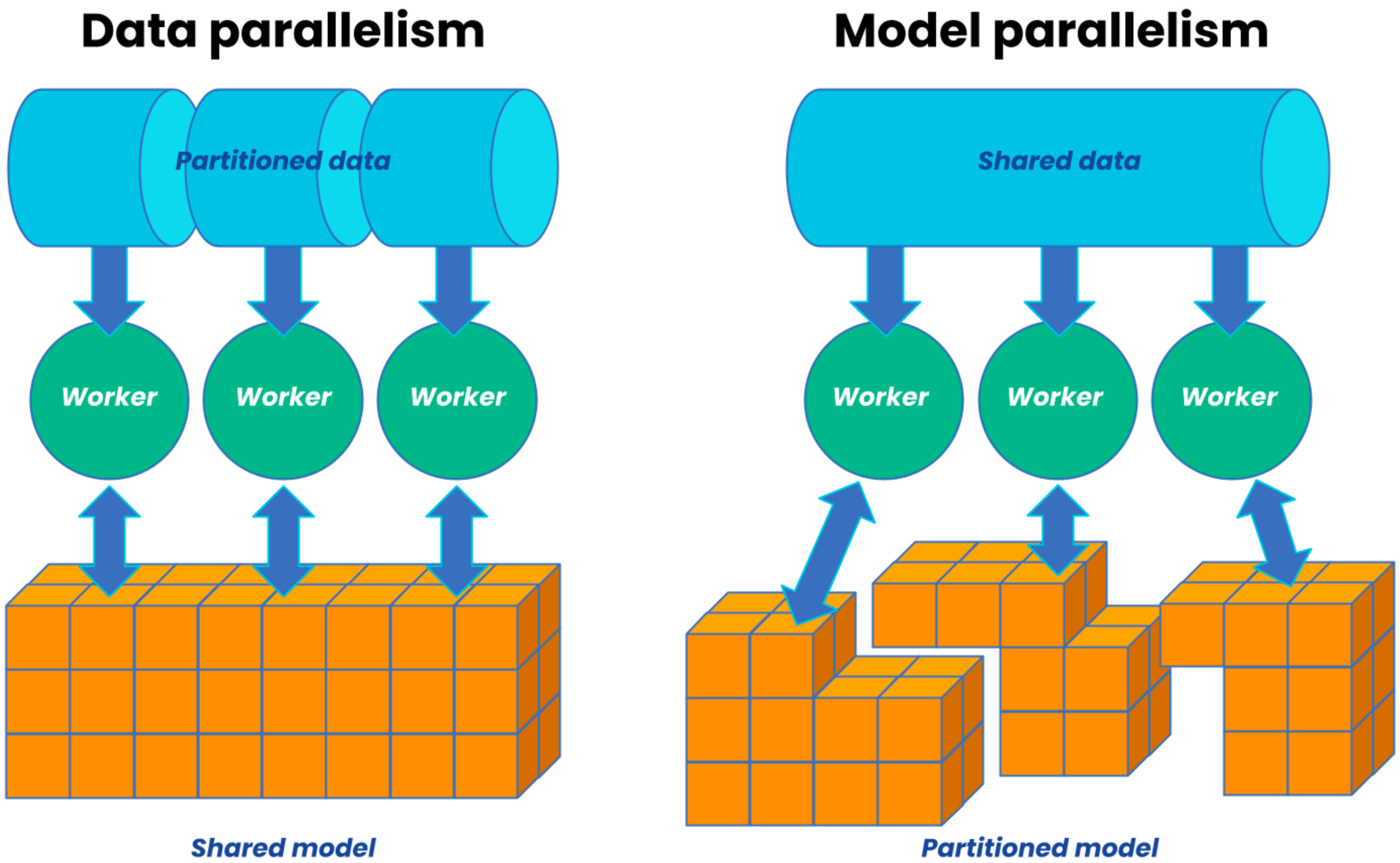


For any of these datasets you cannot fit it all on a single GPU!

Fitting all that data probably requires many parameters…

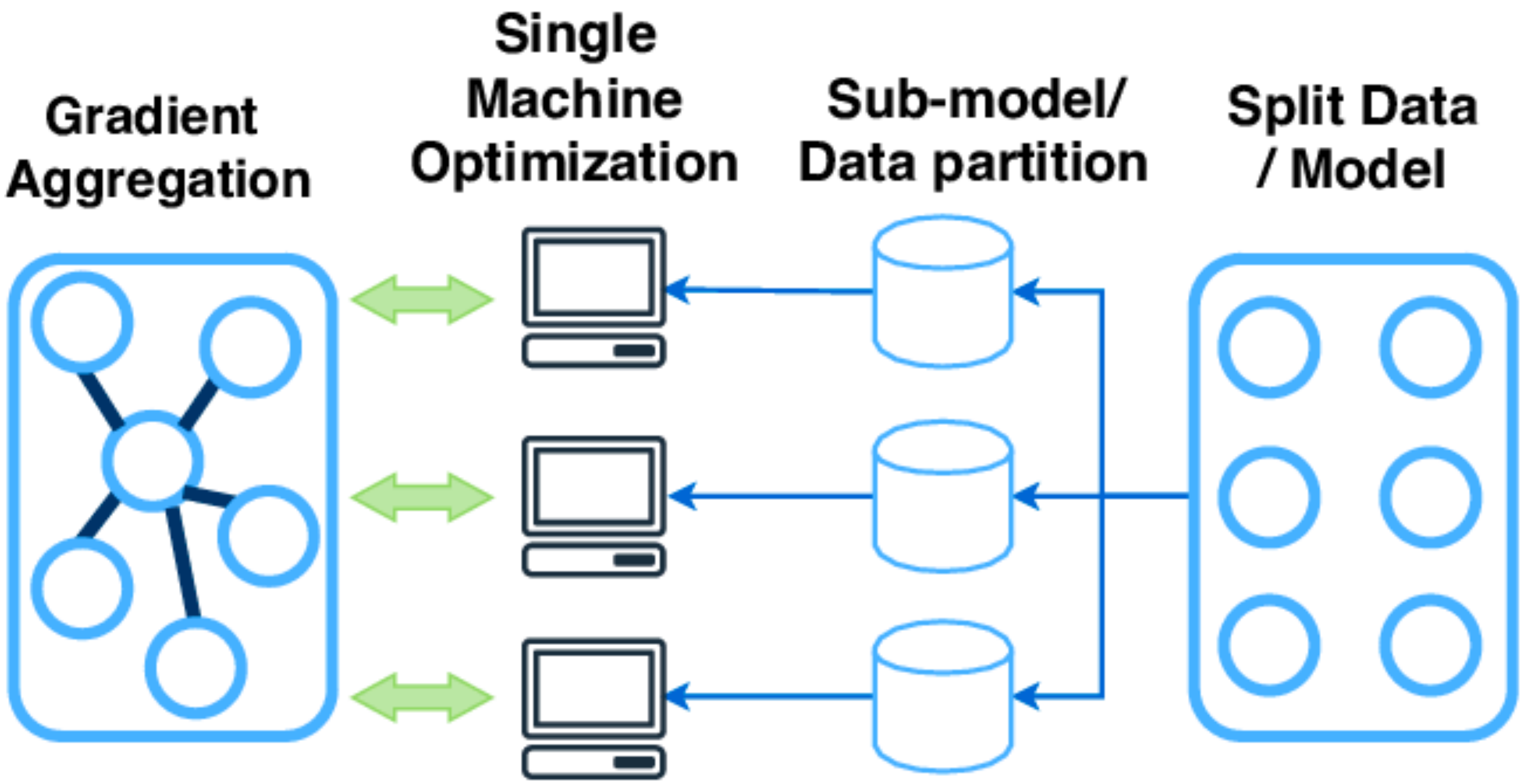


Scaling ML Workflows to Large Resources: Training



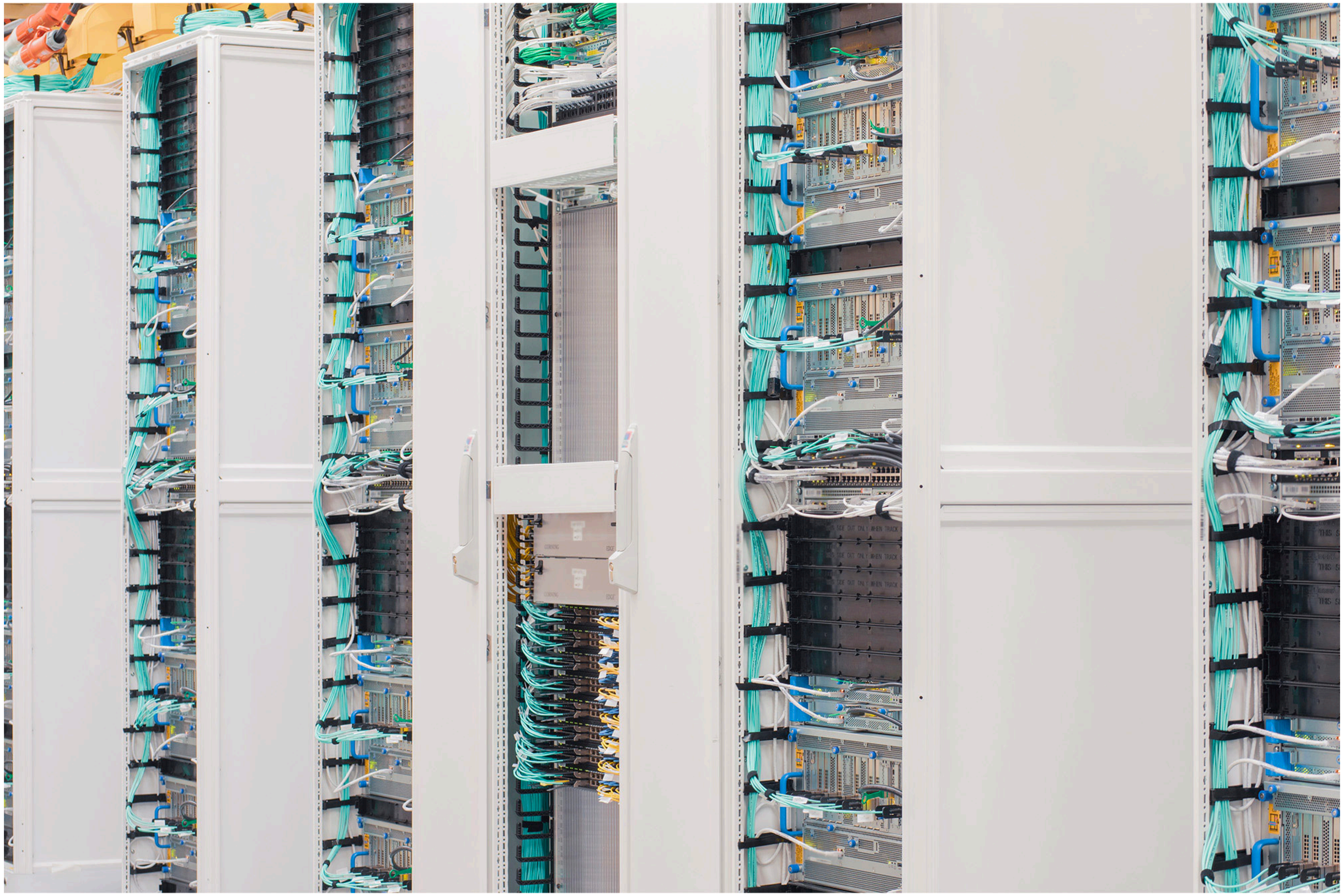
- We can surmount the challenge of these large data and models by distributing either across multiple accelerators and computers
  - With an extremely large performance penalty for any data that must cross that boundary
  - However, this looks structurally like large matrix inversion problems so reasonably optimized supercomputing infrastructure exists!

Scaling ML Workflows to Large Resources: Practicalities



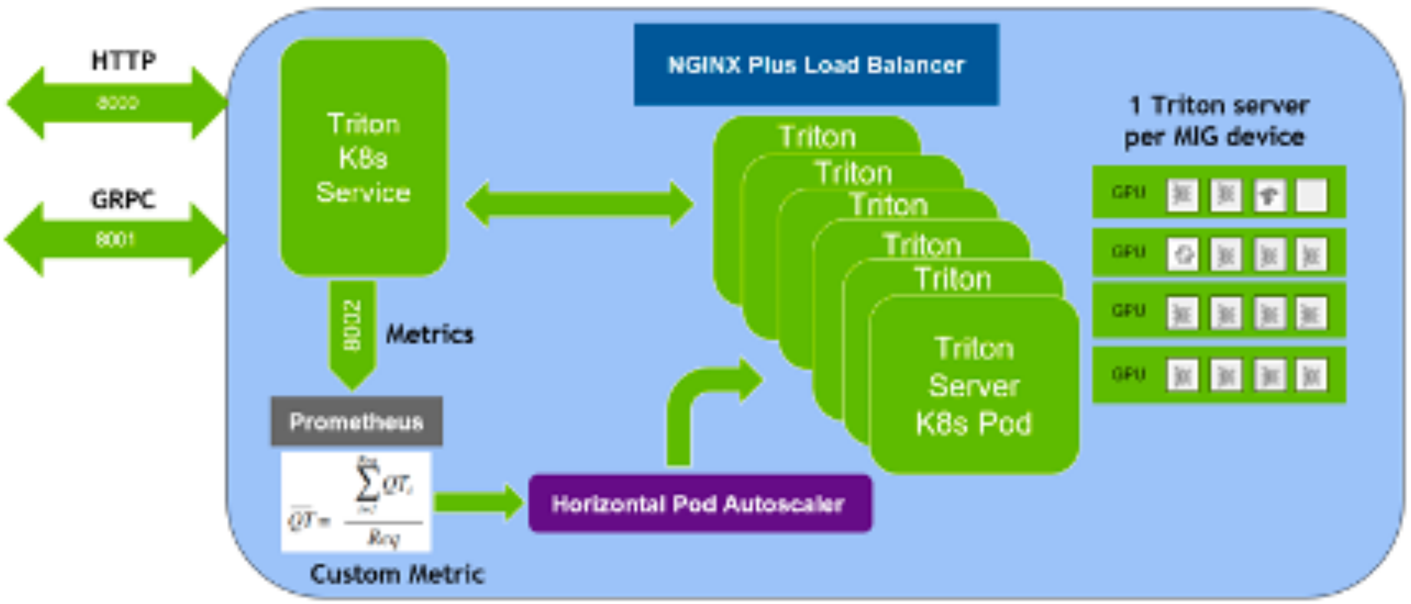
- Generally, instead of feeding all data and the model through one GPU
  - Replicate or split the model amongst multiple GPUs
  - Process the data and each step of the model *storing the gradients for network transport!*
  - Pull gradients together, apply chain rule if needed
  - *Distribute parameter updates back across network*

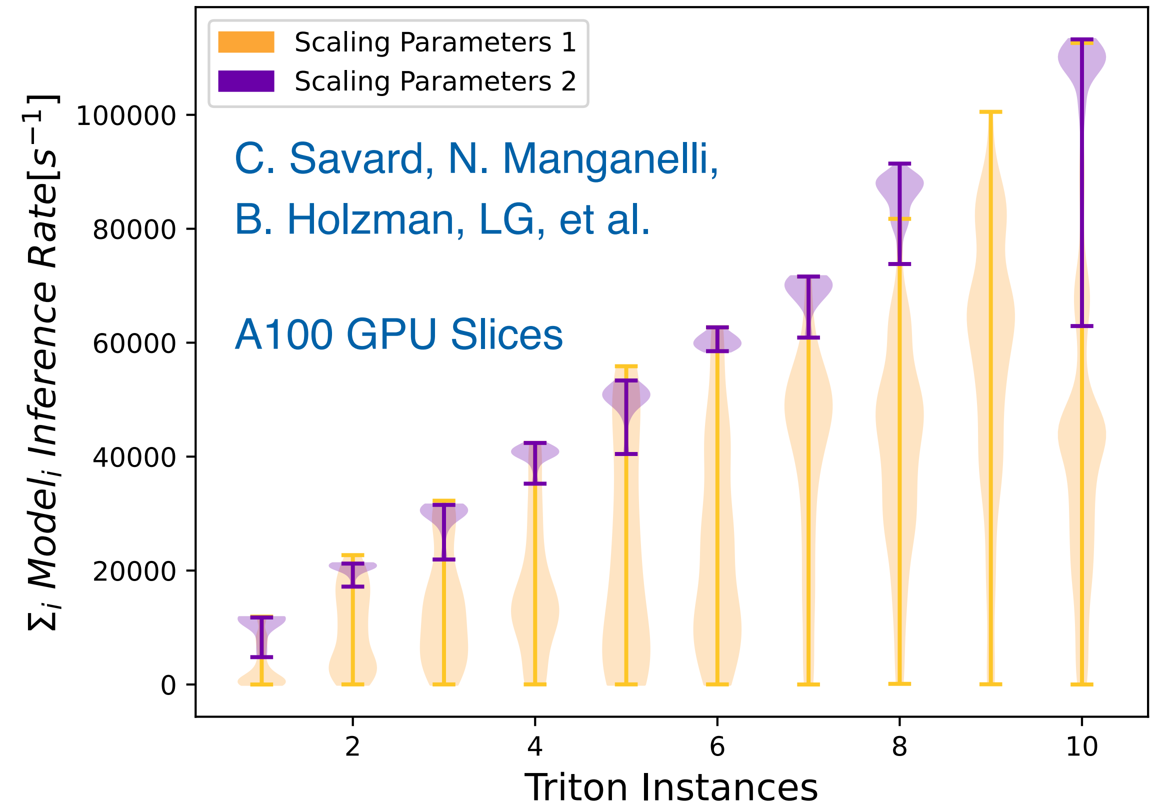
Example: ChatGPT 3/3.5



- 285000 CPU cores
- 10000 V100 GPUs
- 400 Gbit/s interconnect

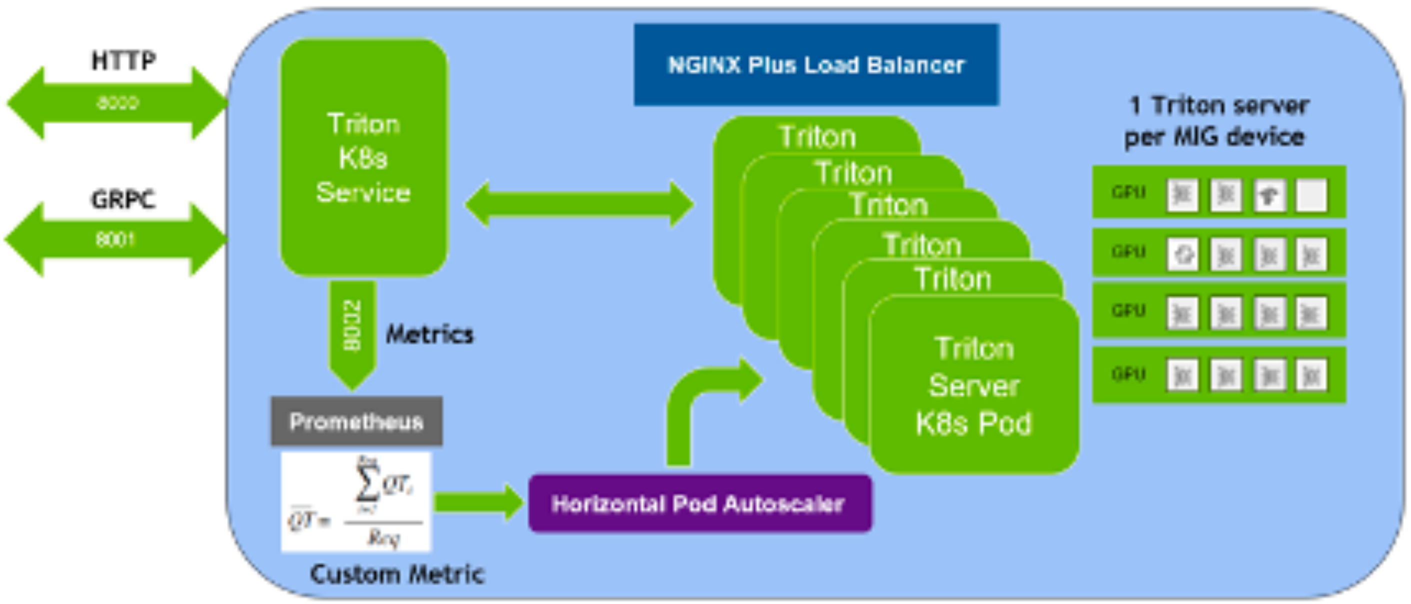
Scaling Inference Workflows

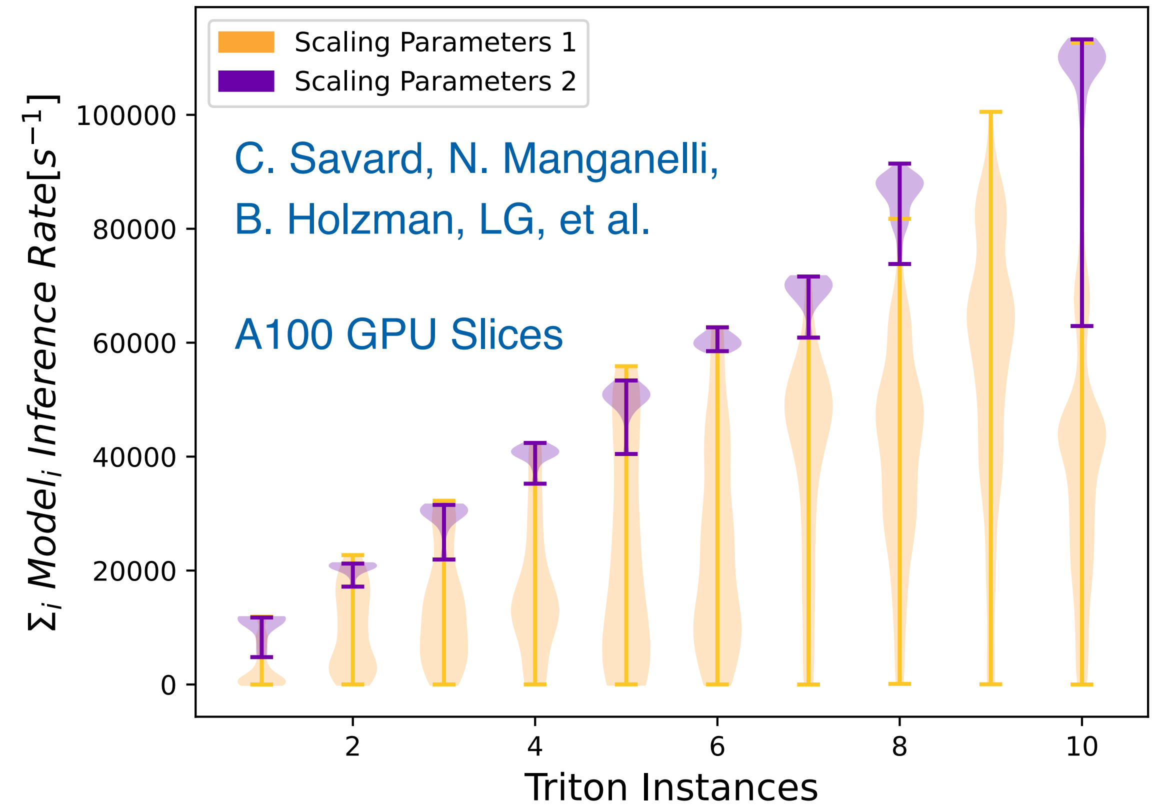




- Scaling inference operates along similar principles but is less expensive computationally and in terms of bandwidth
  - No gradients, no model updates
  - More options for hardware (to discuss in a few slides)
  - Primary considerations become model popularity and resource acquisition
- One popular option that supports many ML software stacks is nVidia Triton
  - Fairly well worked out in terms of features
  - Easily managed and scaled to need
  - Good throughput for complex models (a GNN above), makes real-time usage possible

Scaling Inference Workflows

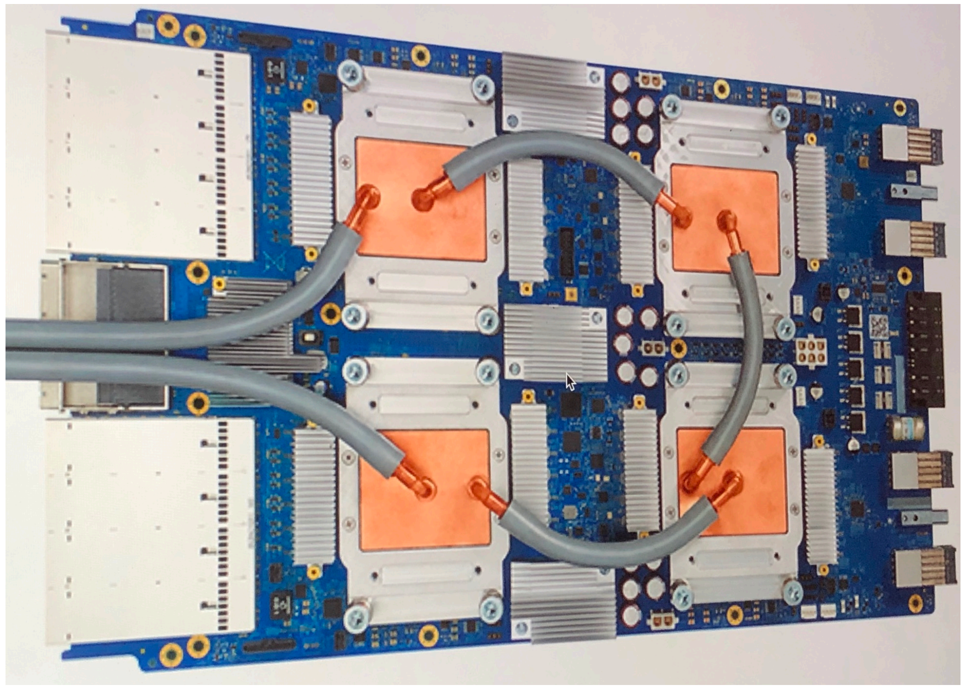




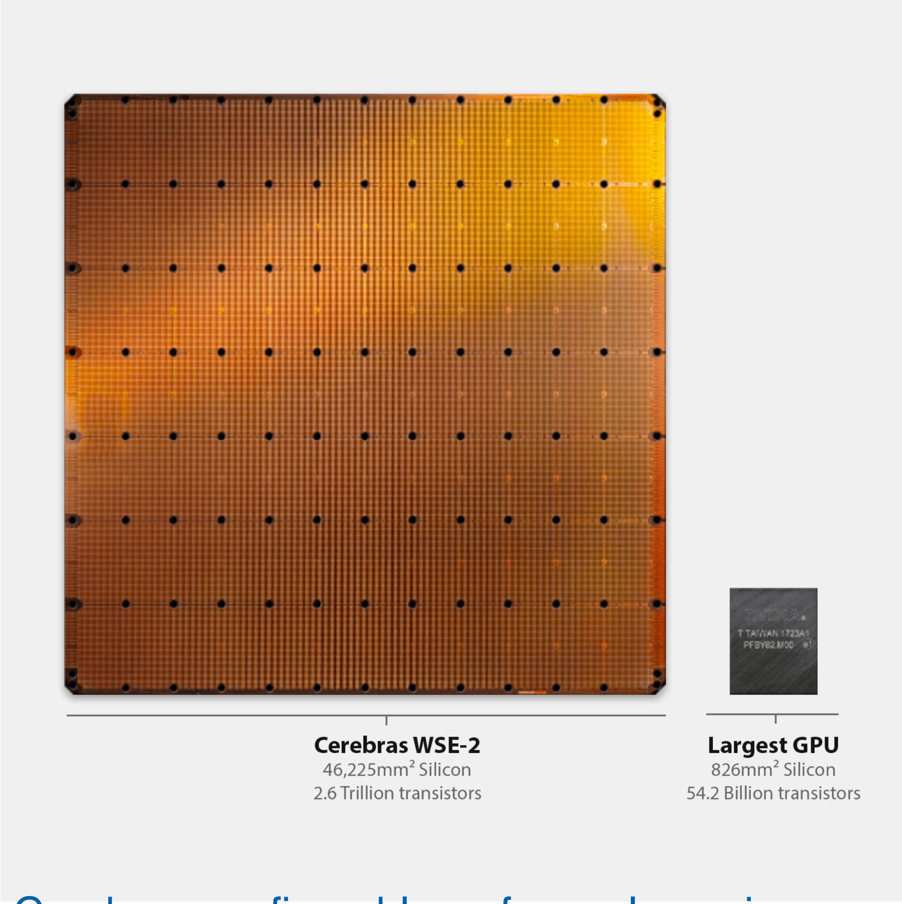
- Technologies like this provide different avenues for operational efficiency
  - Naïve scaling would use a locally attached GPU
  - This can easily leave an inference-only GPU data-starved and wasting energy
  - Intelligent and reactive scaling of ML hardware provides better ways to utilize the hardware you have and to serve *more* inference requests of different kinds using the same amount of hardware, and minimize technical overhead

The impact of ML on Computing Hardware

- GPUs are everywhere in present day data centers
- But we are still limited in scale! If memory <-> processing bandwidth could be increased further we could train more complex models faster.
  - Given the incredible demand for complex ML workflows extremely non Von Neumann architectures have appeared on the market
  - Huge focus on model compression / fixed point math



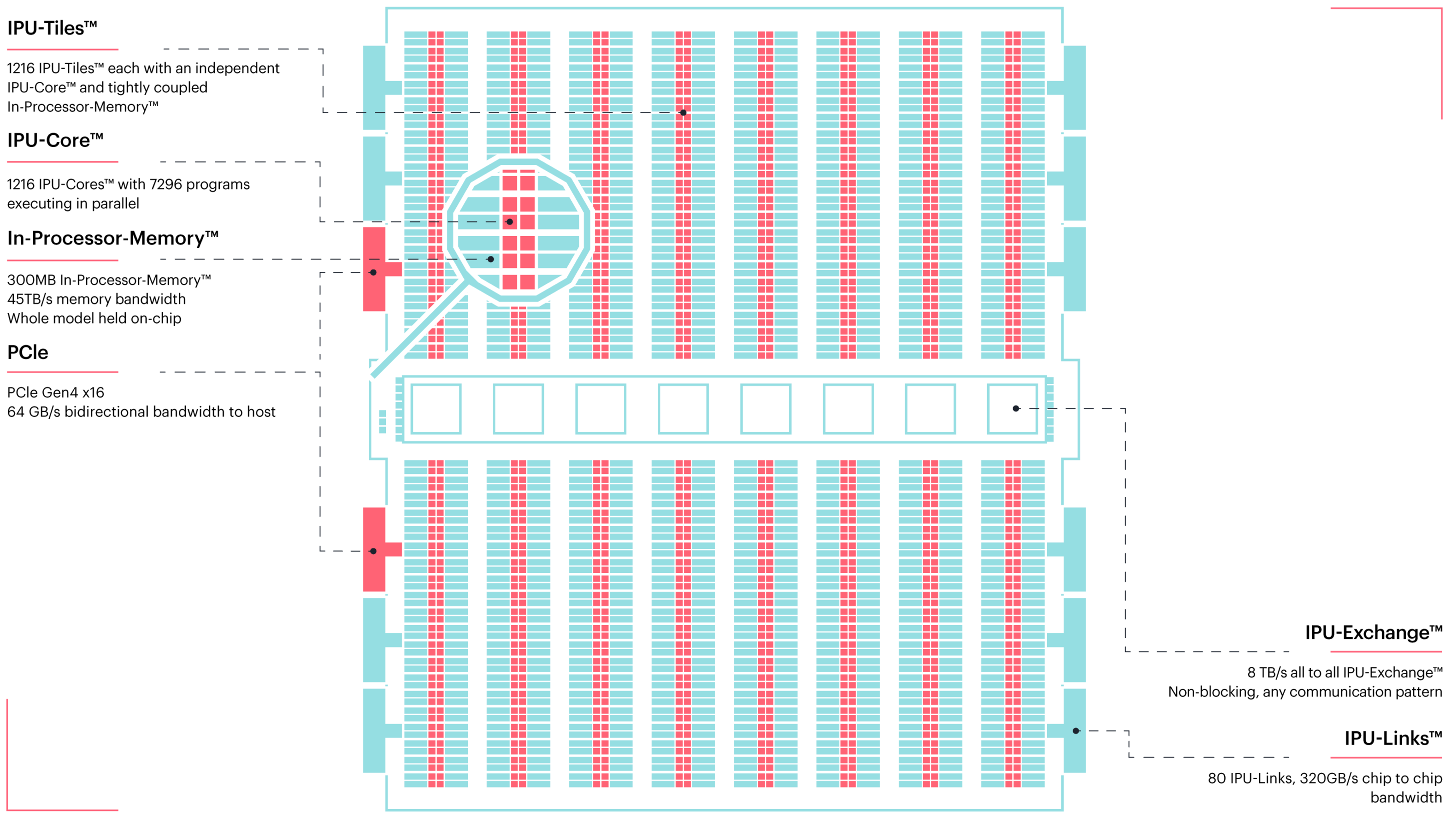
Google TPU v3
V4 with 1.2 GB/s bandwidth
> 200 TOps/s



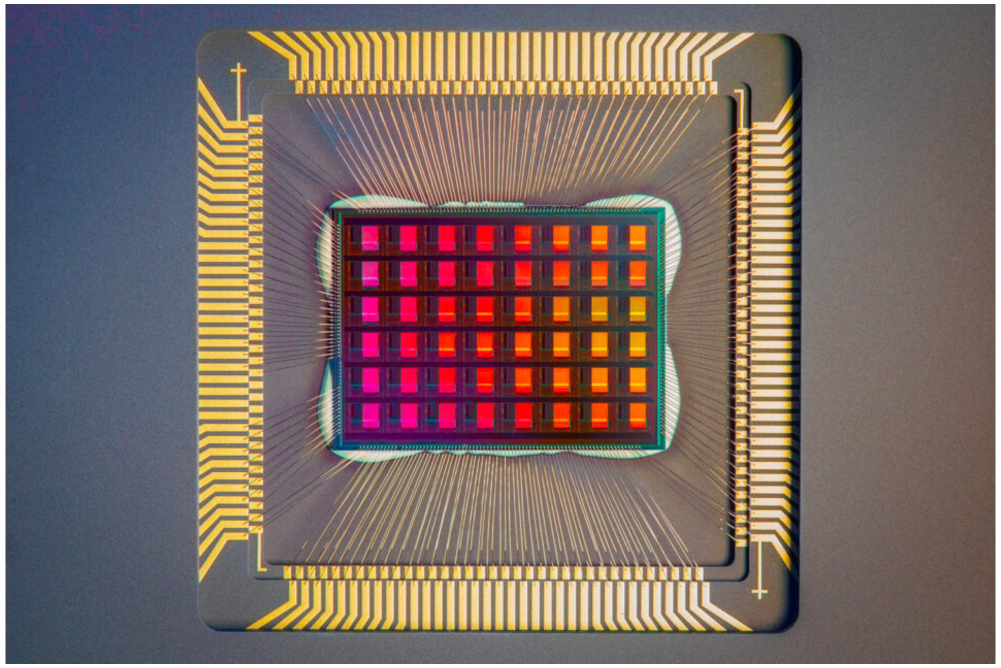
Cerebras configurable wafer scale engine
20 PB/s memory bandwidth

The impact of ML on Computing Hardware

- GPUs are everywhere in present day data centers
- But we are still limited in scale! If memory <-> processing bandwidth could be increased further we could train more complex models faster.
  - Given the incredible demand for complex ML workflows extremely non Von Neumann architectures have appeared on the market
  - Huge focus on model compression / fixed point math / energy efficiency



Graphcore IPU
4x improvement in GNN execution speed vs GPU

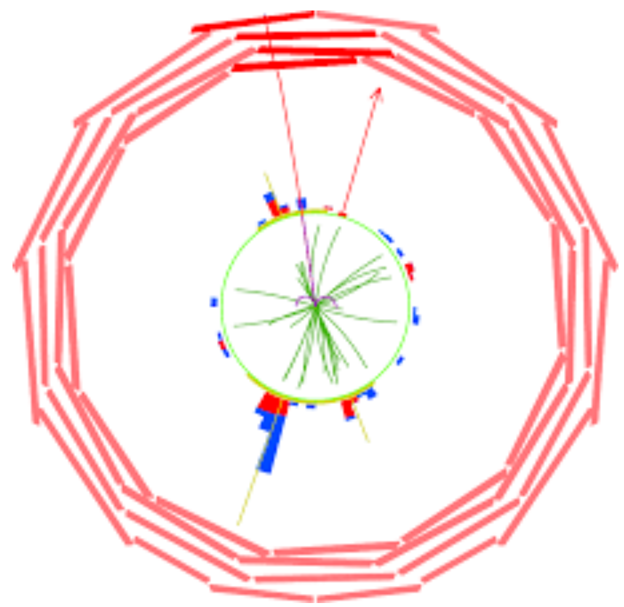


W. Wan, et al. NeuRRam analog compute-in-memory chip, 2x more energy efficient than digital designs

Some Practicalities and Common Examples

Matching Physics Software with Machine Learning Software

????

f()

- This goes more for collider HEP data than astrophysics
  - Event driven data models are often much more structured than image data
  - The majority of the history of ML frameworks, and most of the devices used to compute ML efficiently, expect regularly structured data
    - GNNs are a fairly recent addition that can deal with more varied structure in data
  - ROOT-based file i/o and tools have predominantly dealt with things in an event-wise treatment, does not match well with using GPUs efficiently
  - We also tend to not have that many GPUs available to do things with!

Python ecosystem tools

uproot

Awkward Array

RDataFrame

DASK

Parquet

- From Nick M.'s presentations you've seen the python ecosystem tools that deal with this impedance mismatch for training
  - ROOT also provides a way to do this with RDataFrame
- The state of data ingestion has improved dramatically in the last five years
  - However, the ML and data science ecosystem is older than these tools
  - It is worth it to explore other tools that have been engineered for python data science
  - Dask, parquet, etc. can all help to improve data prep and pipelines into ML

Typical Use Cases of ML in Collider Physics

CMS Physics TDR (2006)

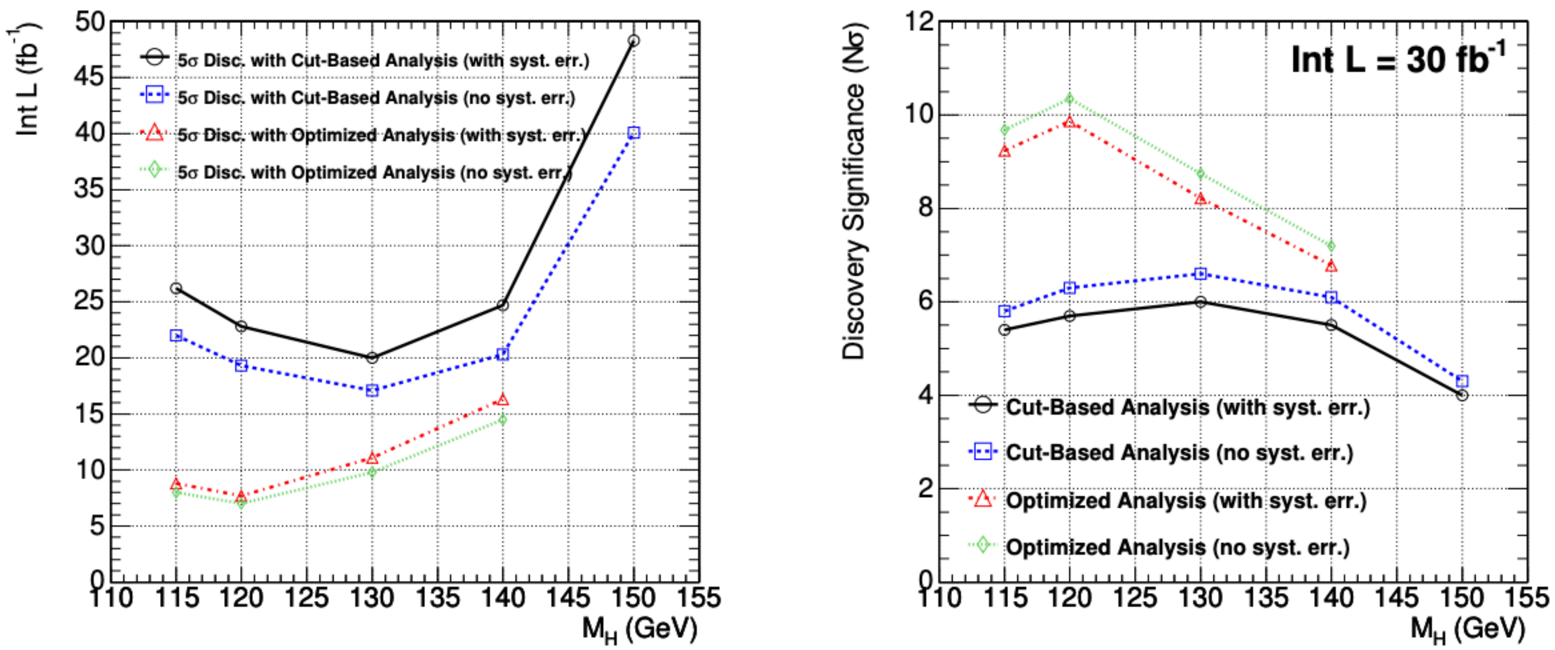
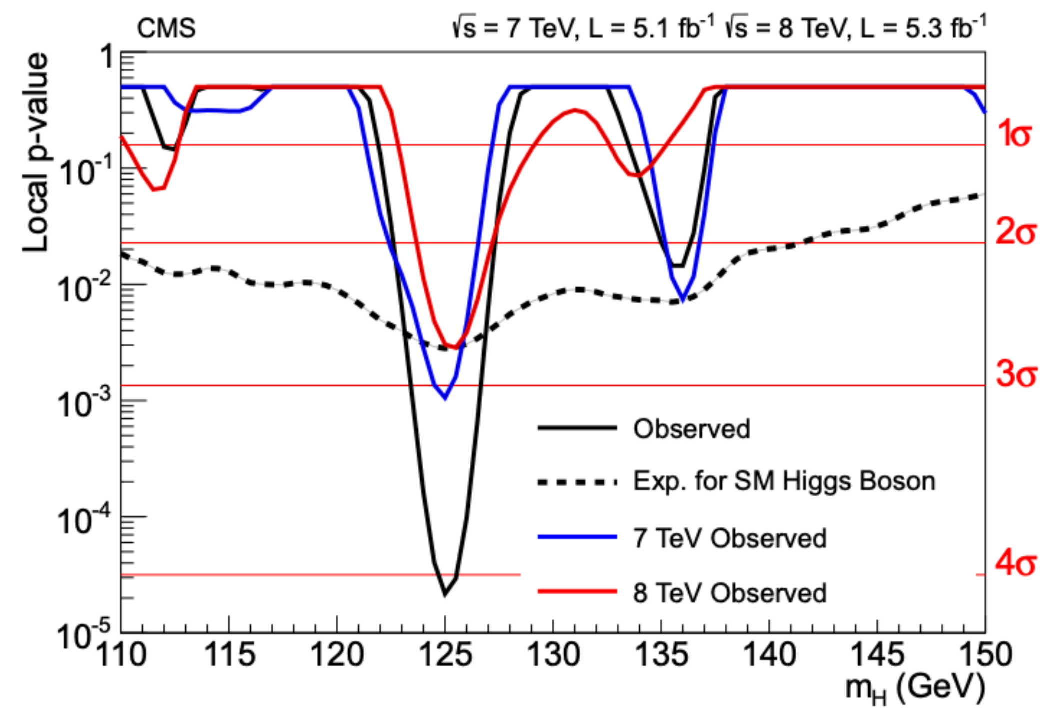
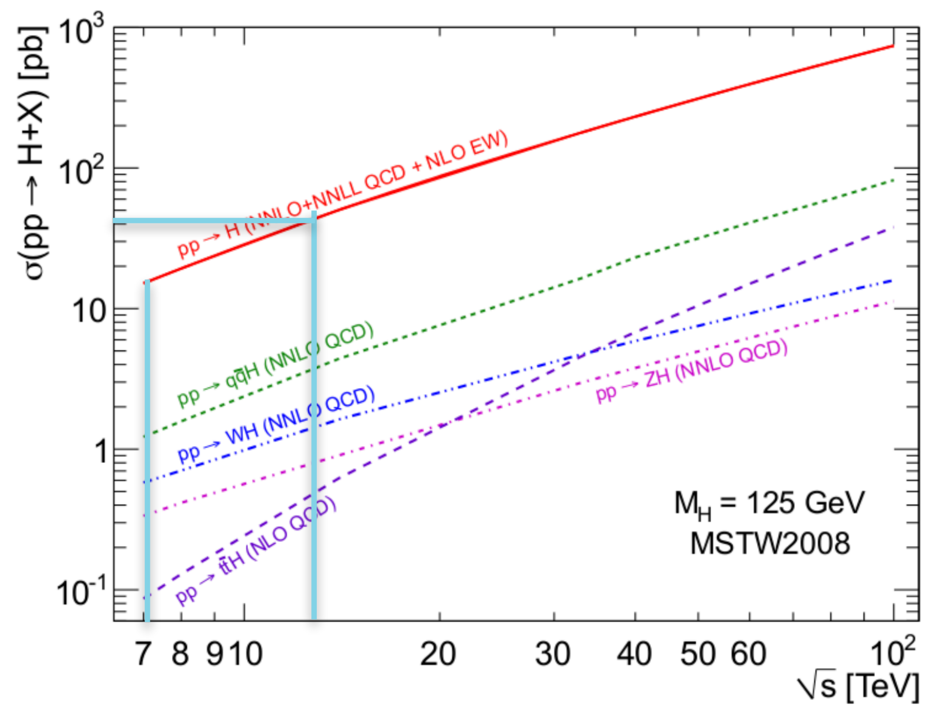


Figure 2.10: Integrated luminosity needed for a 5σ discovery (left) and discovery sensitivity with an integrated luminosity of 30 fb^{-1} (right) with the optimised analysis. The results from the cut-based analysis in 12 categories are also shown for comparison.

Observation of Higgs (2012)

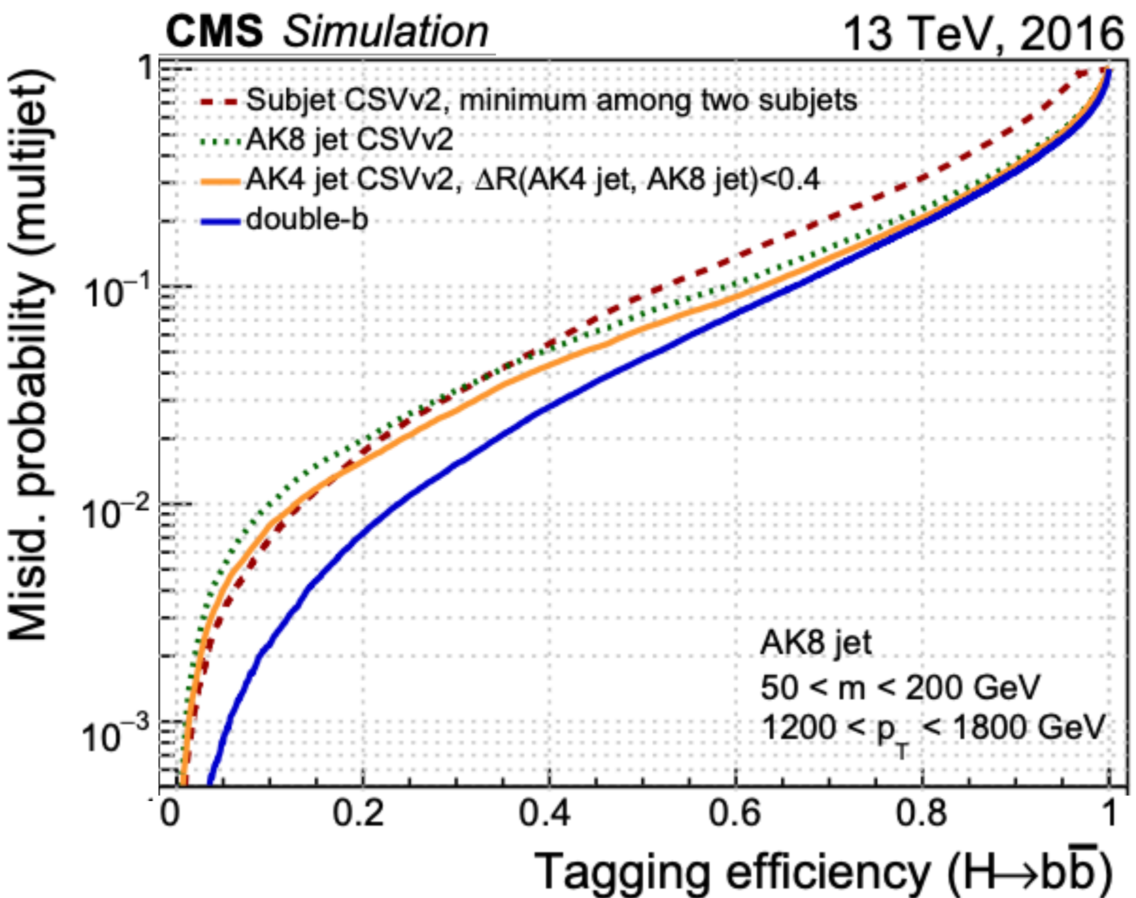


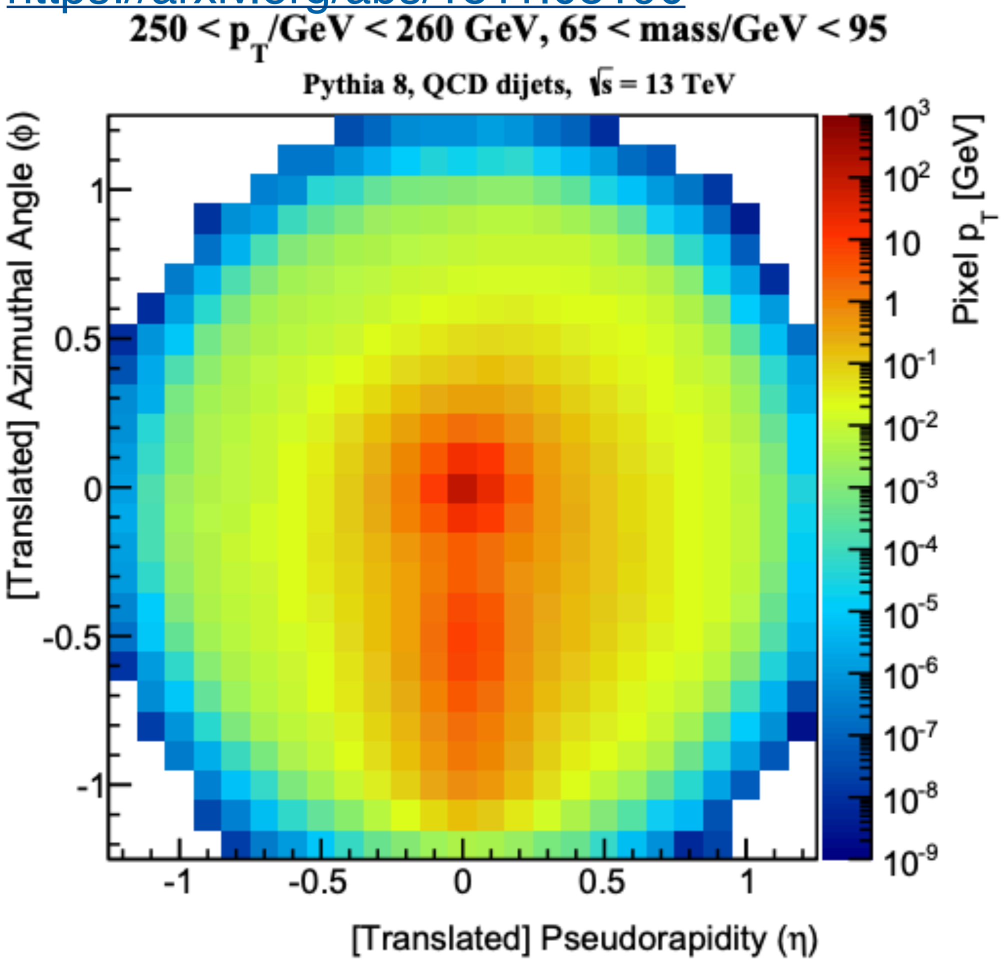


- ML optimized analyses pursued since early days of CMS experiment
- Still, advances in computing allow us to discover the Higgs with ~4x less cross section but the **same** amount of data.

Typical Use Cases of ML in Collider Physics

https://arxiv.org/abs/1511.05190

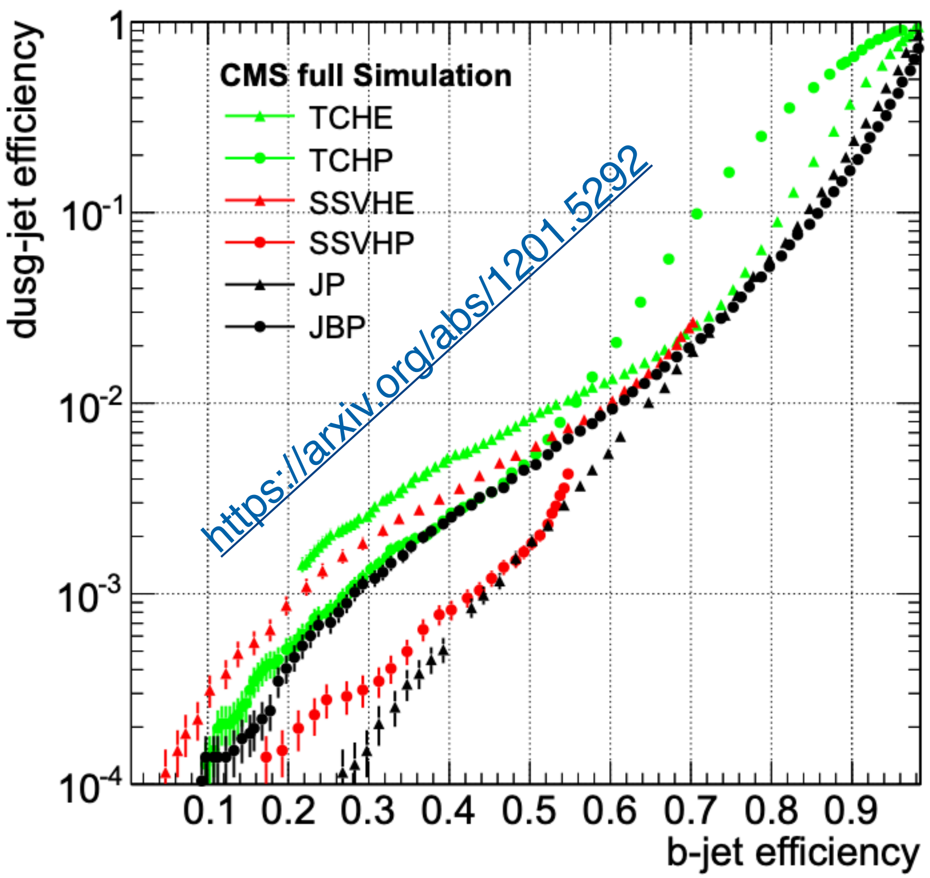


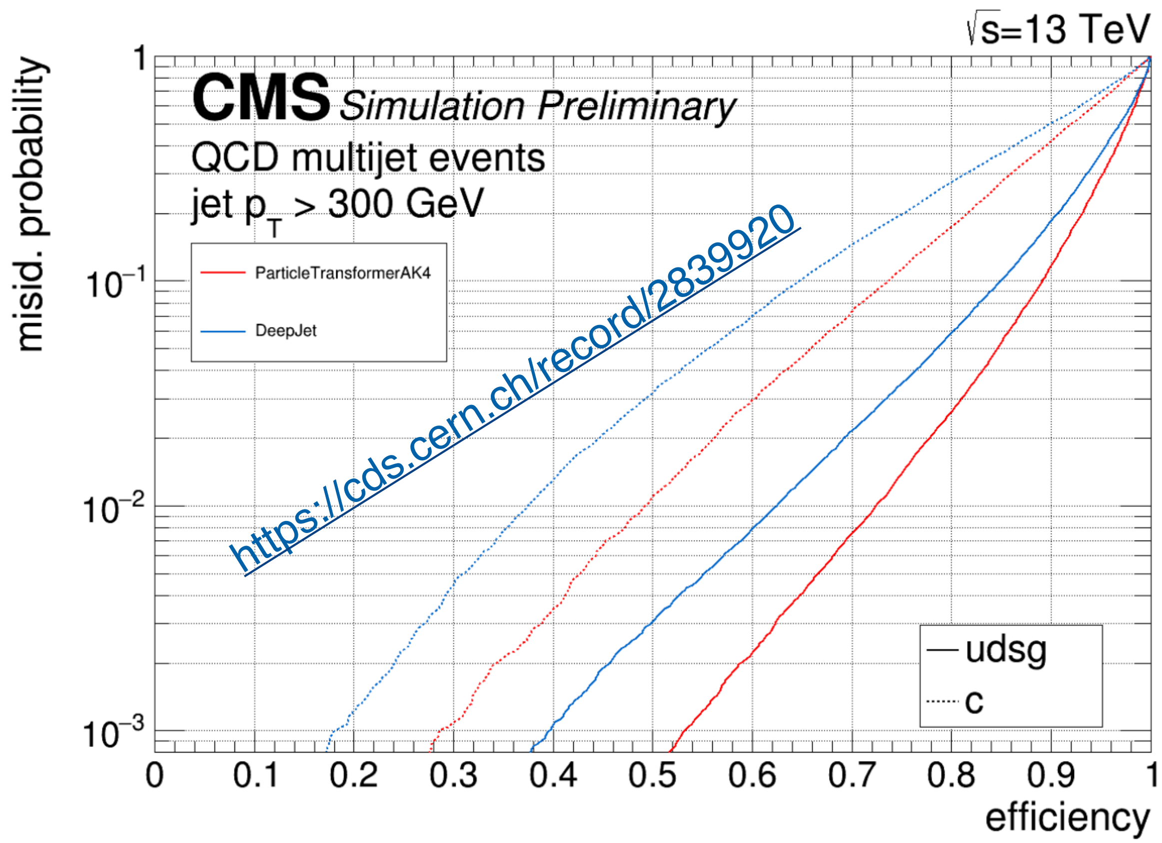


https://arxiv.org/abs/1712.07158

- The most clear area for ML application in high energy physics is jet tagging
  - Some history examples above demonstrating the progression of computing techniques explored to try to make ever-better jet taggers
- The complexity of jet tagging yields a wealth of complex information that is difficult for humans to utilize completely
  - The amount of low level information is large, and varying due to the structure of QCD

Typical Use Cases of ML in Collider Physics





	Parameters	Time (CPU) [ms]	Time (GPU) [ms]	$1/\varepsilon_b$ at $\varepsilon_s = 30\%$
ResNeXt-50	1.46M	7.4	0.22	1147 ± 58
P-CNN	348k	1.6	0.020	759 ± 24
PFN	82k	**0.8**	**0.018**	888 ± 17
ParticleNet-Lite	**26k**	2.4	0.084	1262 ± 49
ParticleNet	366k	23	0.92	$\mathbf{1615 \pm 93}$

https://arxiv.org/pdf/1902.08570.pdf

- Advancing computing techniques yielding better initial data representations improved background rejection multiple integer factors during life of LHC
  - This improvement is significantly faster than we accumulate data
- Even more recent models improve upon this on multiple axes

A Note on Using Cutting Edge Models Effectively

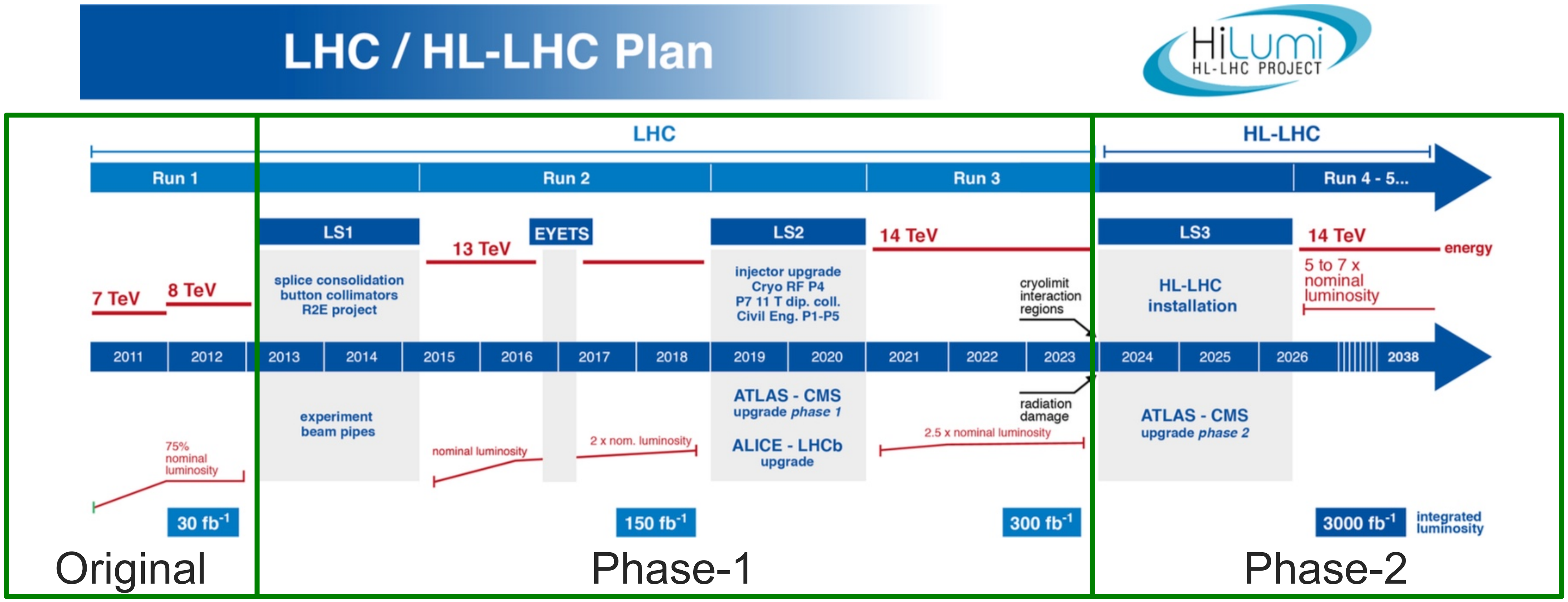
- ML Technology often outstrips the computing capabilities of physicists trying to do analysis
  - Probably the best example of this is CMS physicists evaluating cutting edge GNNs on CPU because it's the only way to (eventually) get the job done

- The combination of better computing hardware and software infrastructure is changing issues like this quickly for the better
  - Array-programming-based analysis techniques organize data better for ingestion into models
  - Technology stacks like Nvidia triton make it easy to provision powerful GPU resources
  - This is starting to make use of complex models significantly more convenient since evaluation is quick and data preparation and inference result unpacking are significantly easier compared to previous workflows

- Here, again, advances in ML computing techniques and infrastructure are making life easier for physicists
  - As well as clearly informing the shape of the computing hardware we want to focus on

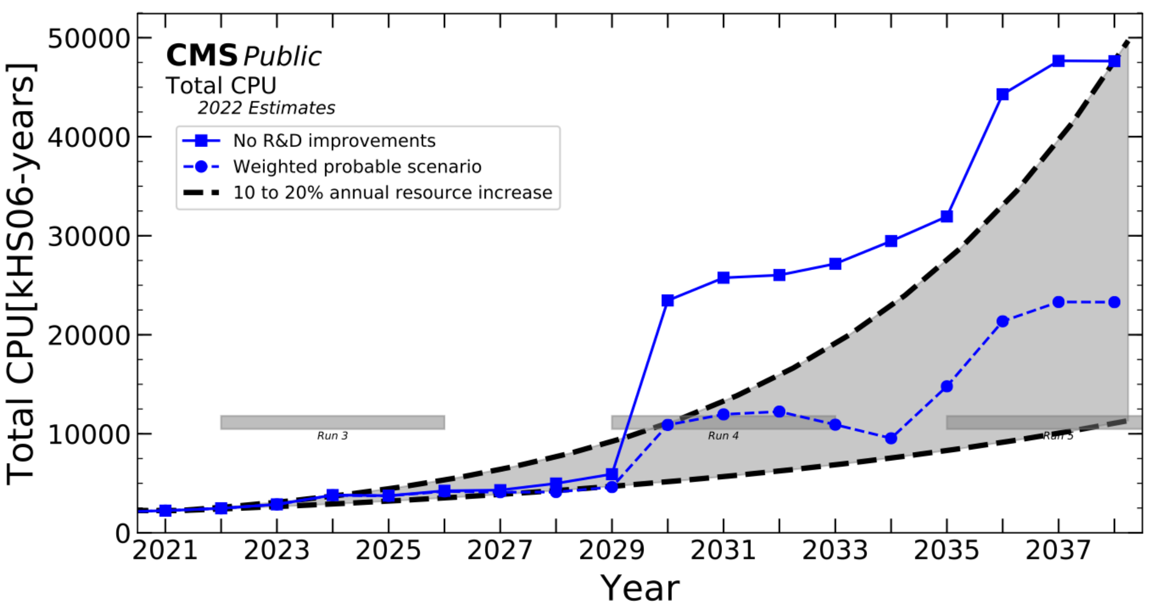
Case Studies from High Energy Particle Physics

Computing Needs for HL-LHC and ML R&D Possible Impact



CMS faces serious computing challenges for HL-LHC.

A major component of computing R&D right now is understanding what accelerator use and ML may bring in terms of improvements.

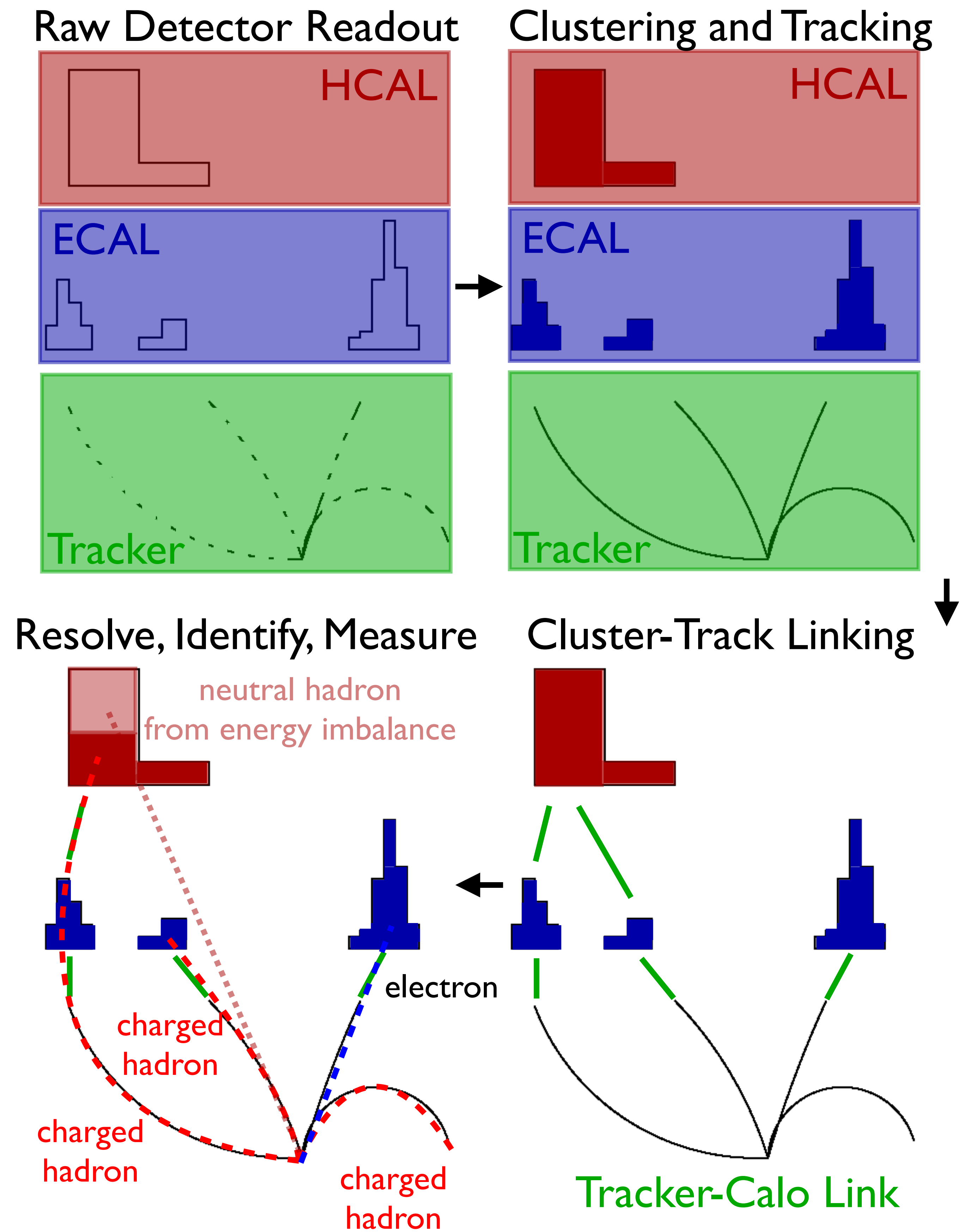


Fermilab

Machine Learned Particle Flow

https://arxiv.org/abs/2203.00330

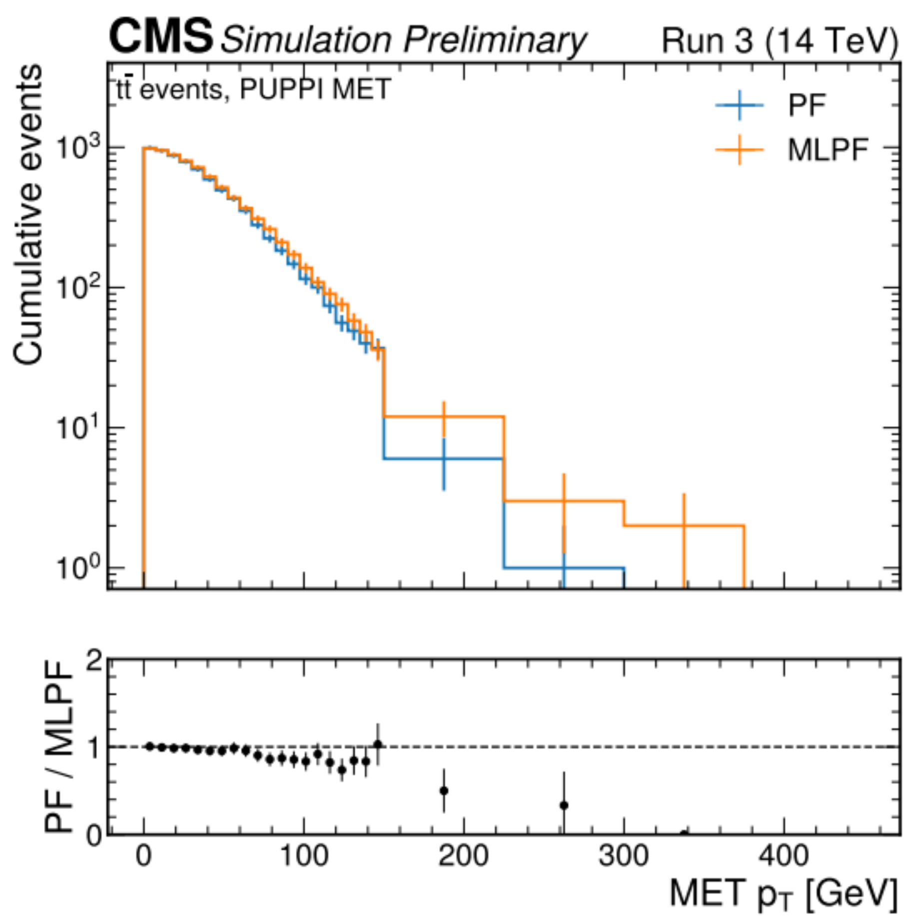
- "Particle Flow" reconstruction a core capability in CMS
  - Attempt to combine various detector reconstructions into a minimal and optimized measurement of all particles in the detector
  - In traditional programming this requires
    - recalibration
    - multiple iterative fits
    - exhaustive clustering with bespoke rules
  - The above is not very friendly to GPU programming and could be considered difficult to maintain
- MLPF project aims to
  - Learn the existing algorithm with ML
  - Eventually learn a simulation-truth based algorithm

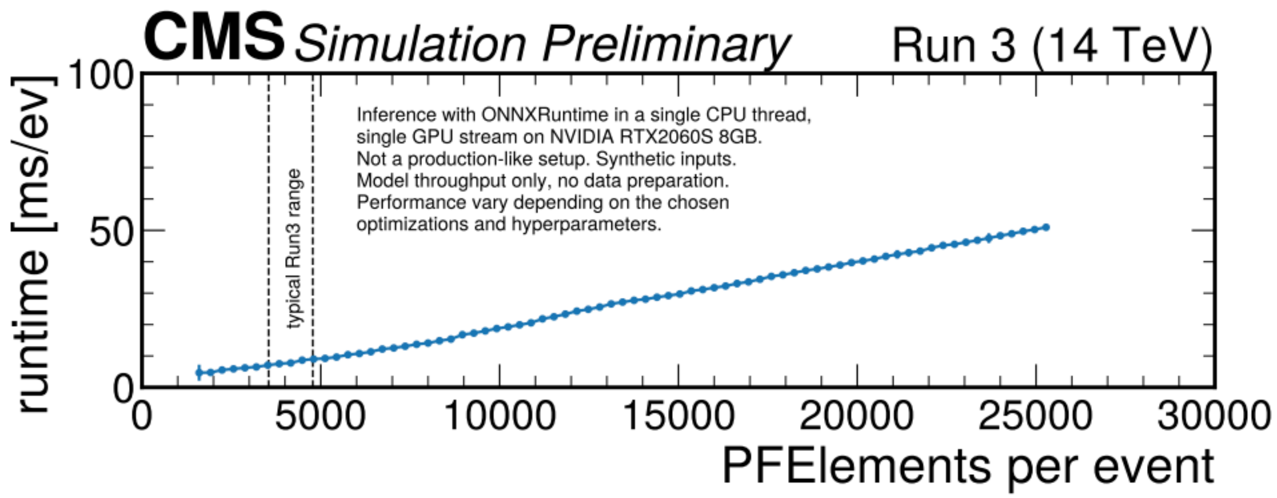


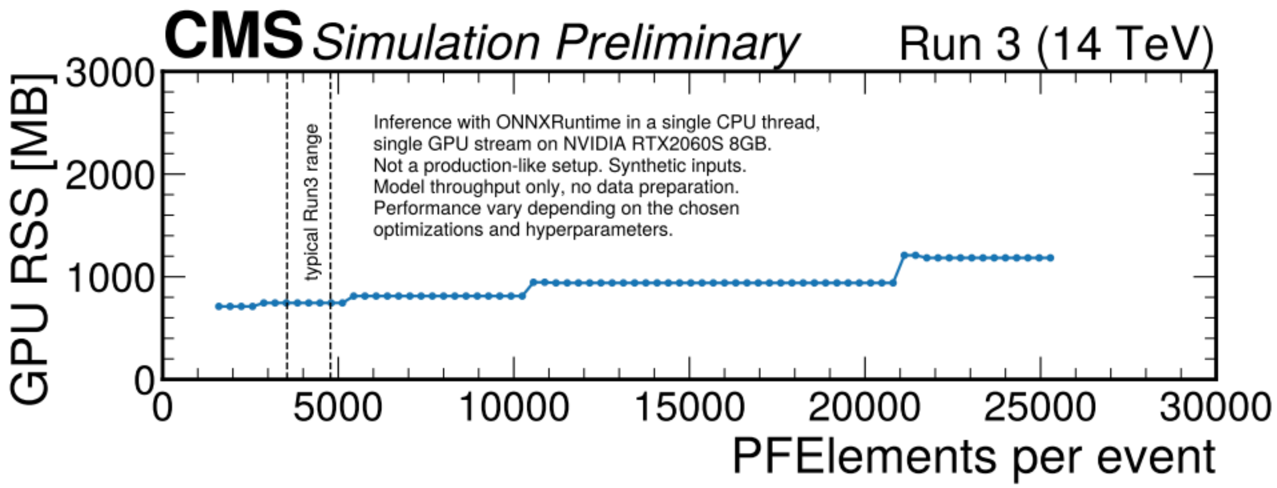
J. Pata et al.

Fermilab

Machine Learned Particle Flow





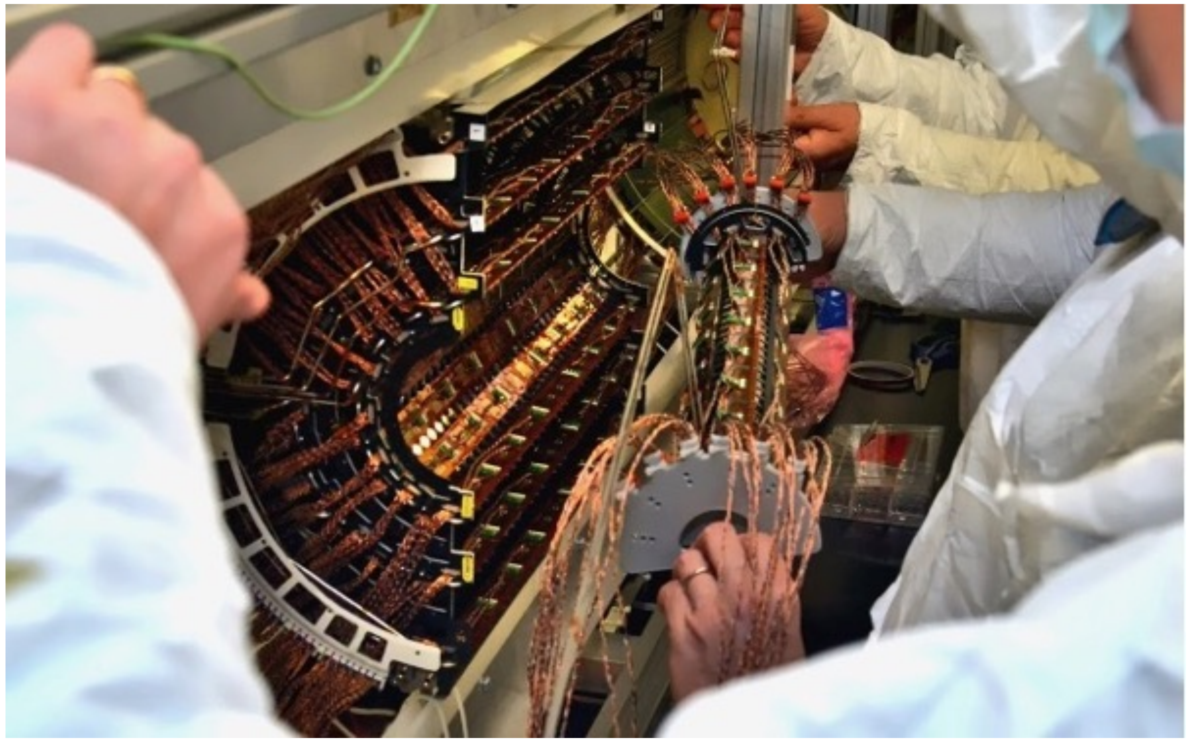


- From a computing perspective - extremely promising
  - Further model tuning necessary to completely capture physics performance
- Assuming physics performance is achieved:
  - Replace iterative, combinatorial algorithm with optimized knn + graph neural network
  - Observe linear scaling of computation time, minimal linear scaling in memory usage
- MLPF is interesting from the computing perspective since it

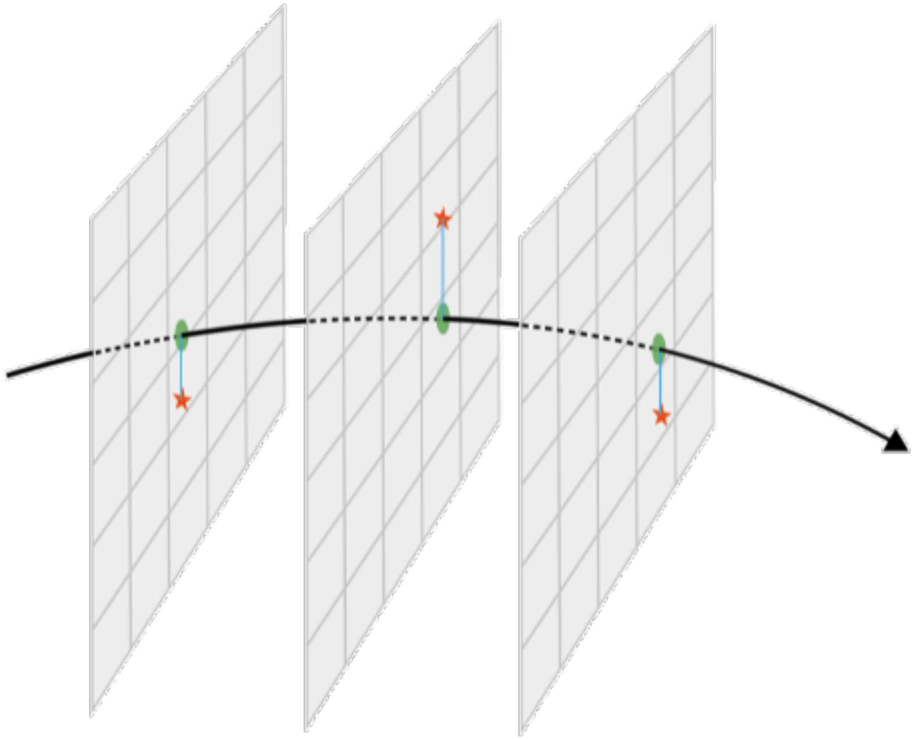
(Single-hit) Track Reconstruction

LG, J. Dickinson, et al.

- Track reconstruction a vital part of modern particle physics
  - Connect ionization deposits of charged particle together to estimate trajectory
- Real-time tracking pivotal capability in HL-LHC Triggers
  - But missed the pixel detector due to data rate



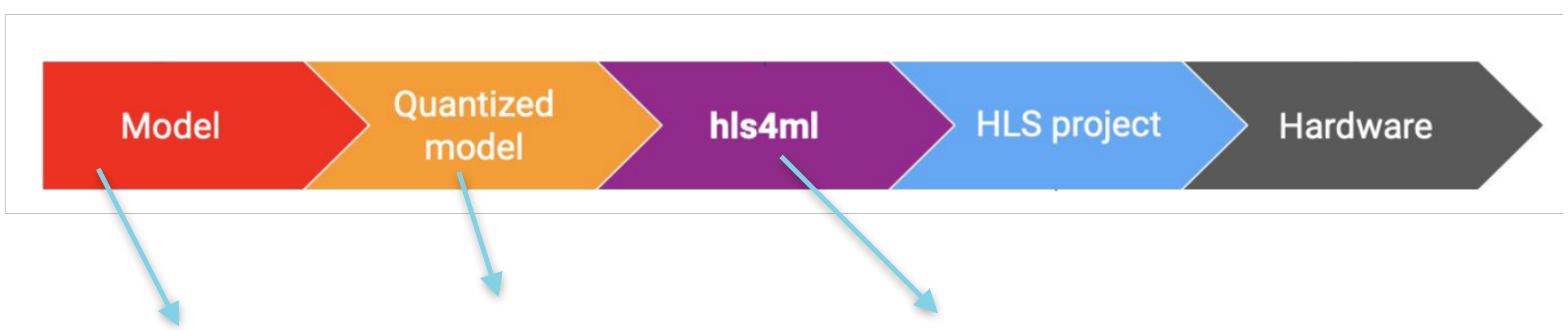
- “Smartpixels” project aims to develop on-sensor AI that provides pixel level triggering or trajectory estimates with one plane of silicon



https://ml4physicalsciences.github.io/2023/files/NeurIPS_ML4PS_2023_133.pdf

Single-hit Track Reconstruction

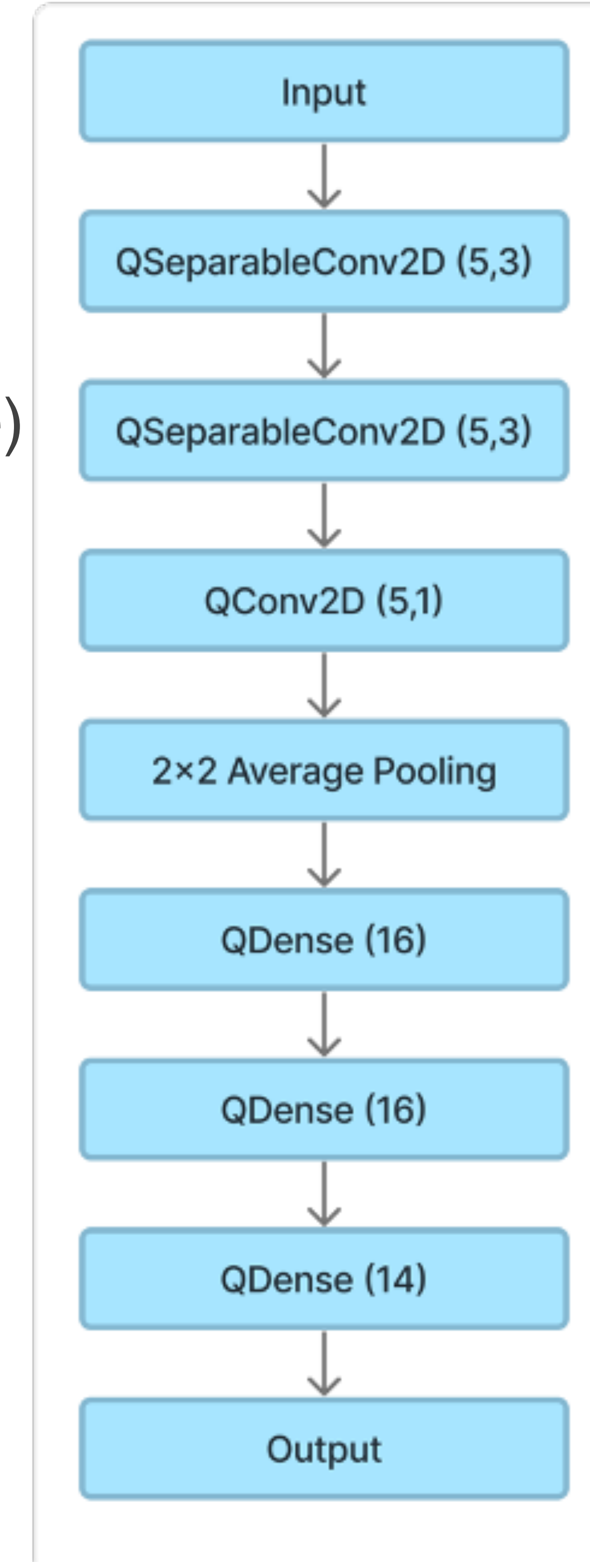
- Quantized aware training an interesting remix of tricks from variational training procedures
  - Full-precision weights kept for gradients
  - Outputs between layers quantized to N bits (adds quantization noise)
- Tracking model is a mixture density network
  - Predicts central values in addition to full covariance matrix
- Network implementable in digital logic achieved through hls4ml workflow
  - Targeting digital ASIC as the final hardware implementation



Trained in Keras

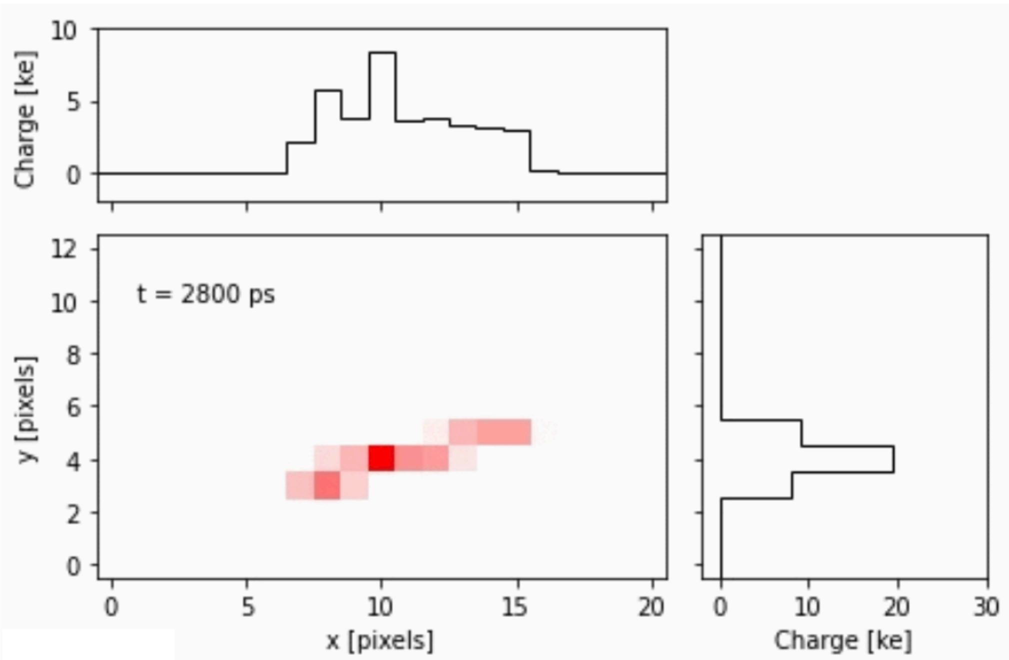
Quantized aware training with Qkeras

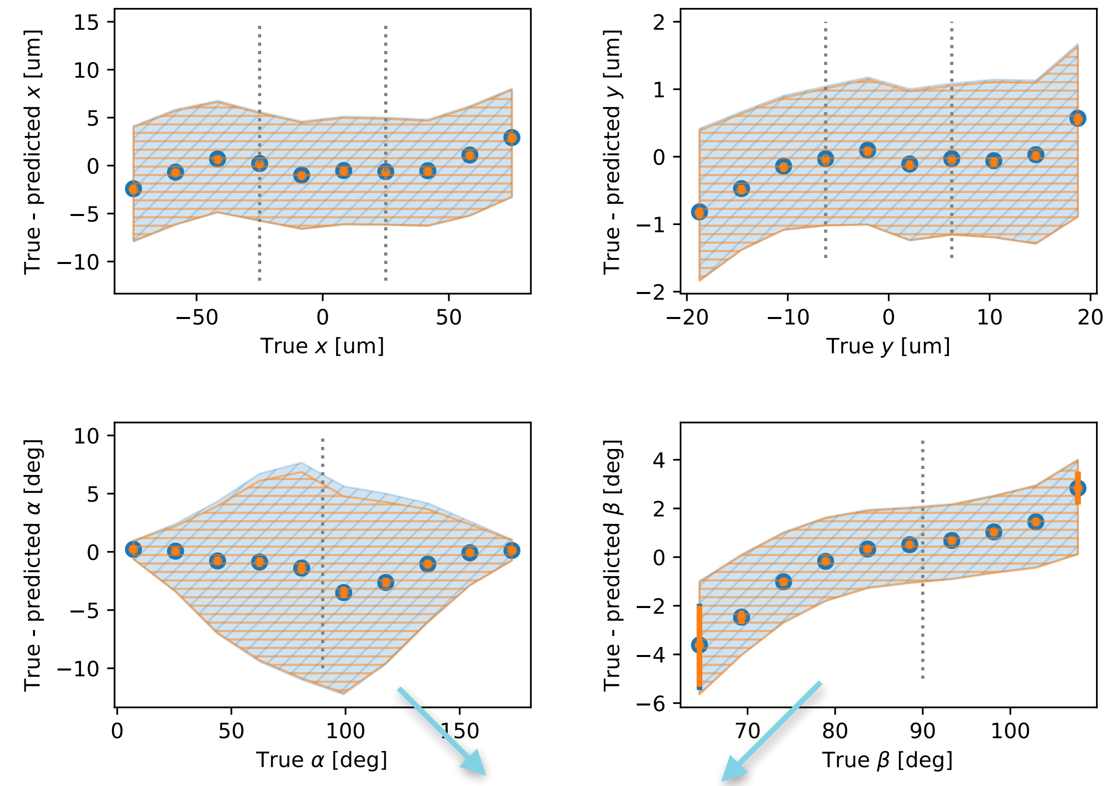
hls4ml converts trained python model into C++ code for high level synthesis



See more on HLS4ML in M. Liu's presentation later this week!

Single-hit Track Reconstruction



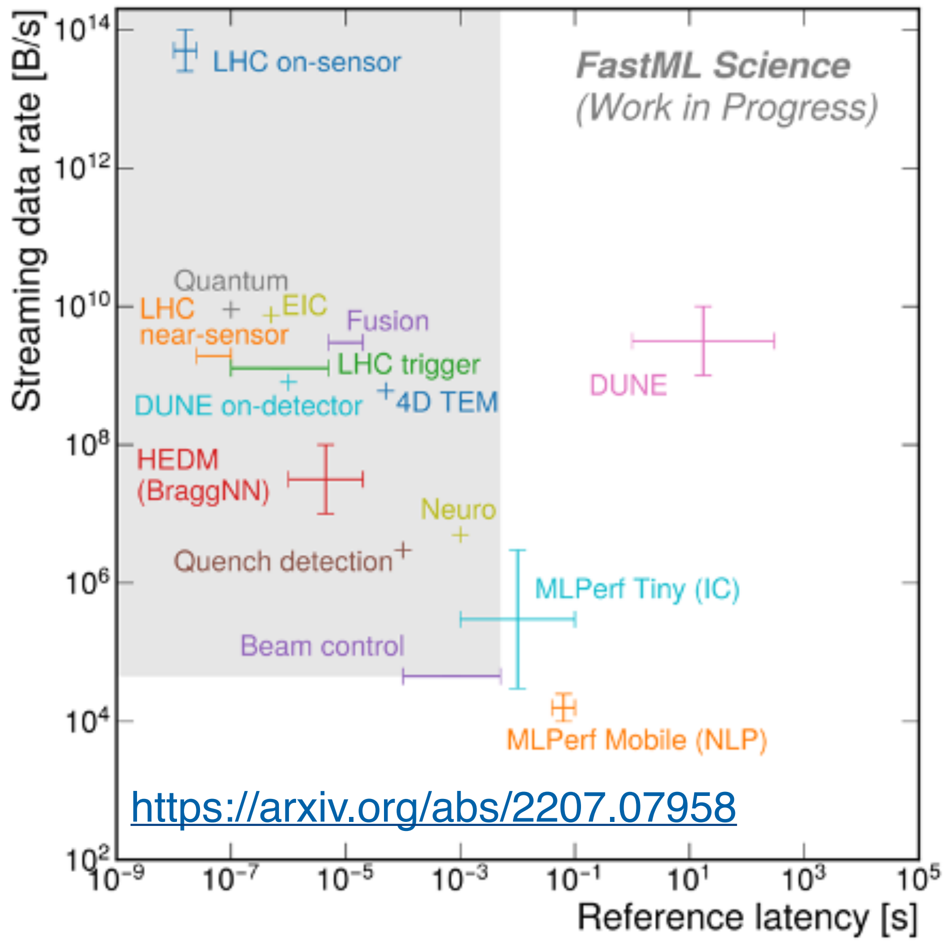


$<\sigma_\alpha> = 3.8°$

$<\sigma_\beta> = 1.7°$

- Model optimization driven by stringent compute needs
  - < 300 uWatt average power usage, implementation size O(100) um2
- Model capacity tested in interesting ways not often encountered in ML
  - Convolutional network uses signed four bit weights (+/- 8 values)
  - Fully connected layers uses signed 8 bit weights
  - Design choice driven by size of input data (13x21x4 bits!) vs. 14x8 bits values and covariance matrix

Single-hit Track Reconstruction



Alveo U250 Synthesis		45nm ASIC Synthesis		45nm ASIC Synthesis	
Clock Period	5 ns	Clock Period	5 ns	Clock Period	25 ns
Latency	1.46 μs	Latency	27 μs	Latency	135 μs
Interval	1.38 μs	Area estimate	1.4 mm^2	Area estimate	1.3 mm^2

- Target latency < 1us (not there yet!), new inference every 25ns

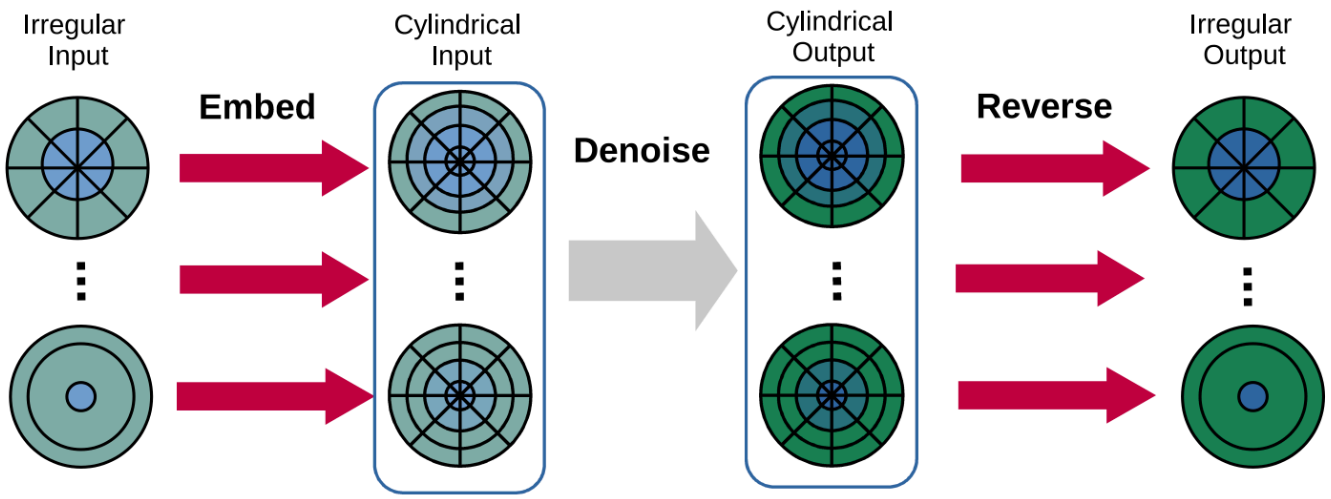
- Design optimization targets the extreme limits of latency and throughput
  - Only possible by designing a custom processor, no instructions just this network!

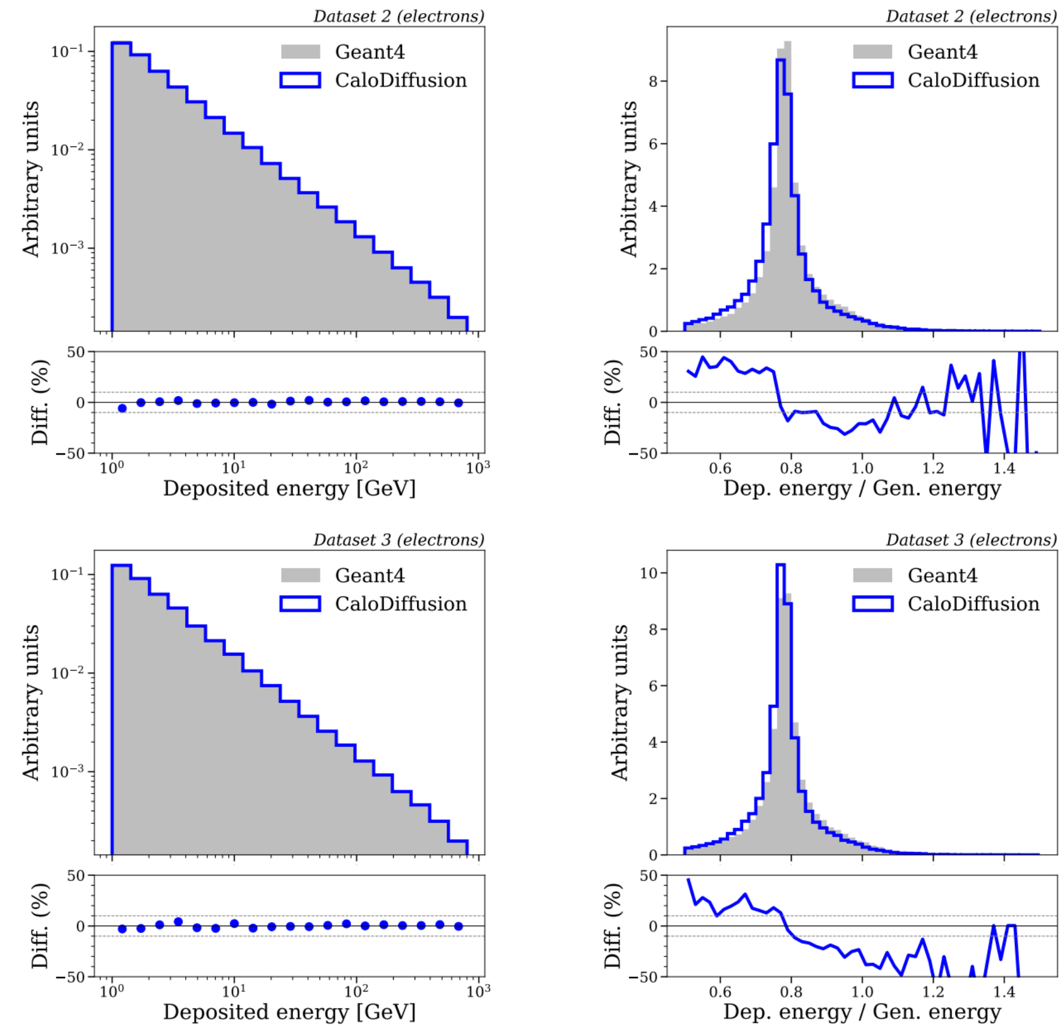
- This is one of the most stringent computing environments available in physics
  - Thinking about and doing computing is entirely different here :-)

Calorimeter simulation with diffusion models

https://arxiv.org/abs/2308.03876

O. Amram, K. Pedro





- This work aims to use diffusion models to construct fast and accurate particle physics simulations
  - Work like this will be critical to meeting the data challenge of the HL-LHC
  - Simulation tools like GEANT are too computationally intensive to use for all of our simulation needs, and other fast simulation tools sacrifice too much simulation fidelity!
- This method employs an interesting mapping technique to deal with irregular particle physics detector designs

Calorimeter simulation with diffusion models

Dataset	Batch Size	Time/Shower [s] CPU	Time/Shower [s] GPU
1 (photons)	1	9.4	6.3
(368 voxels)	10	2.0	0.6
	100	1.0	0.1
1 (pions)	1	9.8	6.4
(533 voxels)	10	2.0	0.6
	100	1.0	0.1
2 (electrons)	1	14.8	6.2
(6.5K voxels)	10	4.6	0.6
	100	4.0	0.2
3 (electrons)	1	52.7	7.1
(40.5K voxels)	10	44.1	2.6
	100	-	2.0

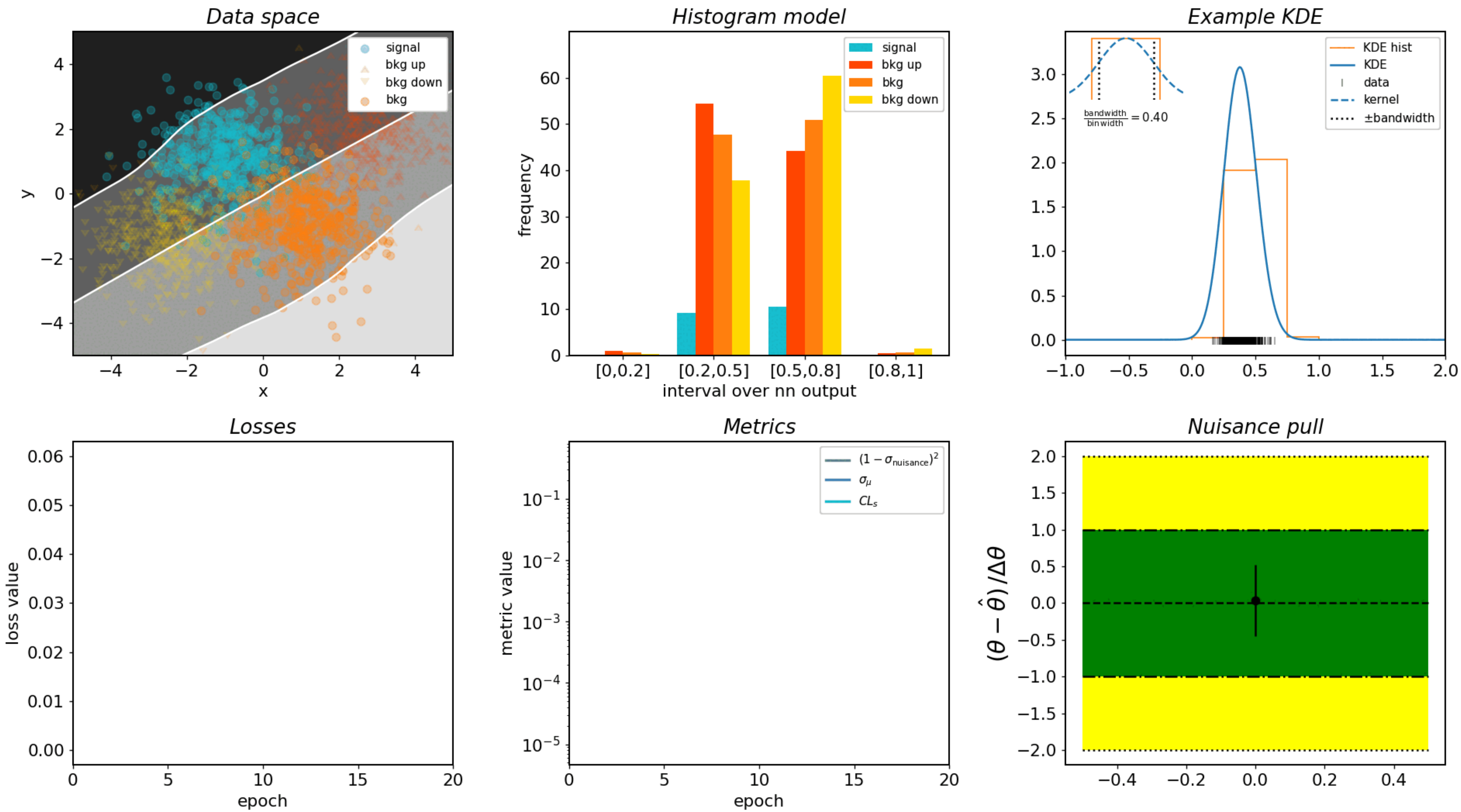
TABLE III. The shower generation time for `CaloDiffusion` on CPU and GPU for various batch sizes.

- Combined with the performance results on the previous page paints a promising picture for a viable solution to HL-LHC simulation needs
  - Here we see the throughput dependence of GPU performance!
  - In most complex examples significantly faster than GEANT already
- This is an excellent example of AI/ML improving basic HEP computing needs
  - Without technologies like this our computing is significantly more expensive

Systematics Aware Differentiable Analysis

https://github.com/gradhep/neos

arxiv.org/abs/2203.05570



Systematics Aware Differentiable Analysis

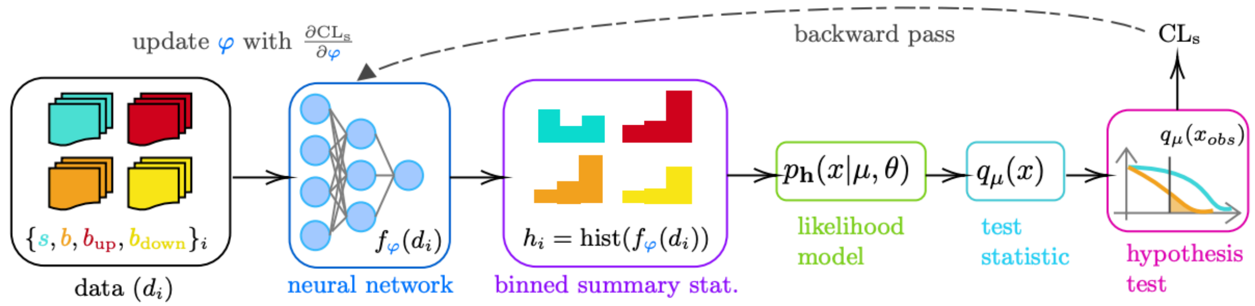


Figure 1. The pipeline for **neos**. The dashed line indicating the backward pass involves updating the weights φ of the neural network via gradient descent.

- What NEOS indeeds to do (similar to INFERNO technique):

(i) Construction of a learnable 1-D summary statistic from data (with parameters φ)

(ii) Binning of the summary statistic, e.g. through a histogram

(iii) Statistical model building, using the summary statistic as a template

(iv) Calculation of a test statistic, used to perform a frequentist hypothesis test of signal versus background

(v) A p-value (or $\mathrm{CL_s}$[1] value) resulting from that hypothesis test, used to characterise the sensitivity of the analysis

We can express this workflow as a direct function of the input dataset $\mathcal{D}$ and observable parameters φ:

$$\mathrm{CL_s} = f(\mathcal{D}, \varphi) = (f_{\mathrm{sensitivity}} \circ f_{\mathrm{test\,stat}} \circ f_{\mathrm{likelihood}} \circ f_{\mathrm{histogram}} \circ f_{\mathrm{observable}})(\mathcal{D}, \varphi). \qquad (1)$$

- Yields some interesting requirements on computing when considering a full physics analysis utilizing large-scale computing
  - Large number of bins, multiple channels, high numbers of systemics imply a large Jacobian and/or many gradients to transfer and keep track of
  - If such a technique viable for complete physics analysis, are there limitations in our compute infrastructure that would prevent wide-scale adoption?

A vision statement…

- I think there's a significant body of evidence that ML has radically altered what can be done with computing and what is considered as computing
  - These last four case studies demonstrate ML are only a few such examples
  - What's between them all is a unifying language containing powerful concepts that fundamentally relate efficient computing strategies to data

- Someone with a background in physics and ML can meaningfully enter into research that can change physics experiment design at multiple levels
  - From improving our basic computing capabilities, and how much data we can deal with, to enabling previous inaccessible detector hardware capabilities
  - This is purely from the abstraction that ML brings by separating detailed knowledge of algorithm design from being able to quickly and efficiently utilize data

- ML practitioners and researchers in HEP should feel empowered to work towards the sources of their data and bring these powerful tools to bear in ever-more exotic and constrained computing environments
  - The degree of creativity in physics afforded by ML is unmatched

Concluding Remarks

- AI / ML has reshaped computing needs and capabilities within HEP
  - Fueled by the end of Moore's law and ever larger amounts of data we collect
  - Non-von Neumann architectures change the way we need to build our reconstruction and analysis programs to yield speedups
  - Newer accelerators are coming out with even more radical designs that we must understand and incorporate to our computing infrastructure

- AI / ML is guiding the state of computing
  - It is such a powerful tool for distilling data that the demand for ever-more-powerful algorithms guides modern computing infrastructure development
  - Fundamental research is not the leader here and so we must follow
  - … but as one of the originators of data science our field physics should have little issue with following in this case

- AI / ML, as a powerful algorithmic abstraction, changes the kinds problems can be considered usefully from a computing perspective
  - … and this may be the most major change it brings to computing in the end