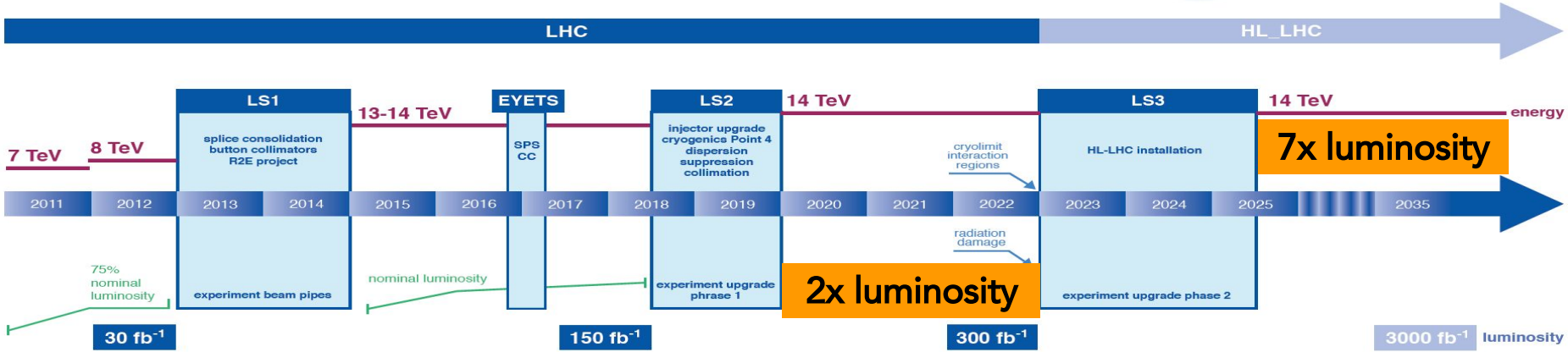
RDataFrame

easy parallel ROOT analysis at 100 threads

Enrico Guiraud for the ROOT team
CHEP 2018, Sofia, Bulgaria

ROOT: a foundation library

- The amount of data produced by HEP experiments is going to increase drastically
  - e.g. at CERN: HL-LHC, FCC, ...
- ROOT's mission does not change:
  bring physicists from collision to publication as effectively as possible



source: http://acceleratingnews.web.cern.ch/content/recent-progress-hilumi-project-0

A recipe for efficient HEP analyses

- ➔ strive for a **simple programming model**
- ➔ expose modern, elegant interfaces that are **easy to use correctly** and hard to use incorrectly
- ➔ allow to **transparently benefit from parallelism**

A recipe for efficient HEP analyses

- ➔ strive for a **simple programming model**
- ➔ expose modern, elegant interfaces that are

  easy to use correctly and hard to use incorrectly
- ➔ allow to **transparently benefit from parallelism**

HEP is not alone in these challenges:
we can **learn from the data science industry**
and bring back what physicists need, in the form they need it

---

A recipe for efficient HEP analyses

➔ strive for a **simple programming model**

➔ expose modern, elegant interfaces that are

easy to use correctly and hard to use incorrectly

➔ allow to **transparently benefit from parallelism**

HEP is not alone in these challenges:
we can **learn from the data science industry**
and bring back what physicists need, in the form they need it

---

RDataFrame, officially part of ROOT since v6.14, tries to incarnate these ideas in the context of HEP analyses and HEP data manipulation

RDF user guide, cheat-sheet

DOI 10.5281/zenodo.260230

An ergonomic, fast C++ dataframe

```
ROOT::EnableImplicitMT();
```

Run a parallel analysis

An ergonomic, fast C++ dataframe

```
ROOT::EnableImplicitMT();
```
Run a parallel analysis

```
ROOT::RDataFrame df(dataset);
```
on this (ROOT, CSV, ...) dataset

An ergonomic, fast C++ dataframe

```
ROOT::EnableImplicitMT();
```
Run a parallel analysis

```
ROOT::RDataFrame df(dataset);
```
on this (ROOT, CSV, ...) dataset

```
auto df2 = df.Filter("x > 0")
```
only accept events for which x > 0

An ergonomic, fast C++ dataframe

```
ROOT::EnableImplicitMT();
```
Run a parallel analysis

```
ROOT::RDataFrame df(dataset);
```
on this (ROOT, CSV, ...) dataset

```
auto df2 = df.Filter("x > 0")
```
only accept events for which x > 0

```
             .Define("r2", "x*x + y*y");
```
define r2 = $x^2 + y^2$

An ergonomic, fast C++ dataframe

```
ROOT::EnableImplicitMT();
```
Run a parallel analysis

```
ROOT::RDataFrame df(dataset);
```
on this (ROOT, CSV, ...) dataset

```
auto df2 = df.Filter("x > 0")
```
only accept events for which x > 0

```
             .Define("r2", "x*x + y*y");
```
define r2 = $x^2 + y^2$

```
auto rHist = df2.Histo1D("r2");
```
plot r2 for events that pass the cut

An ergonomic, fast C++ dataframe

```
ROOT::EnableImplicitMT();
```
Run a parallel analysis

```
ROOT::RDataFrame df(dataset);
```
on this (ROOT, CSV, ...) dataset

```
auto df2 = df.Filter("x > 0")
```
only accept events for which x > 0

```
             .Define("r2", "x*x + y*y");
```
define r2 = $x^2 + y^2$

```
auto rHist = df2.Histo1D("r2");
```
plot r2 for events that pass the cut

```
df2.Snapshot("newtree", "out.root");
```
write the skimmed data and r2 to a new ROOT file

An ergonomic, fast C++ dataframe

```
ROOT::EnableImplicitMT();
ROOT::RDataFrame df(dataset);
auto df2 = df.Filter("x > 0")
            .Define("r2", "x*x + y*y");
auto rHist = df2.Histo1D("r2");
df2.Snapshot("newtree", "out.root");
```

Run a parallel analysis

on this (ROOT, CSV, ...) dataset

only accept events for which x > 0

define r2 = $x^2 + y^2$

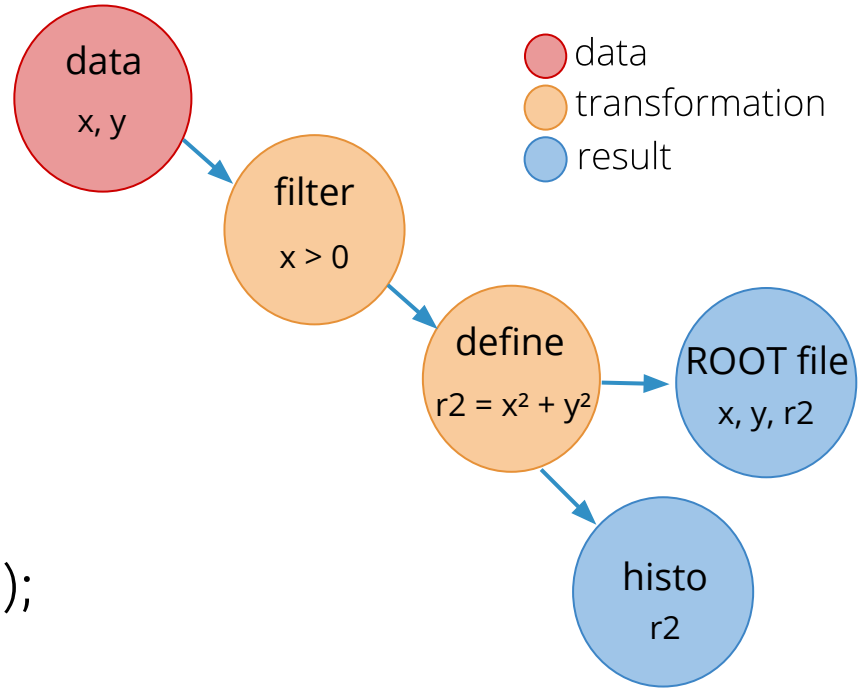
plot r2 for events that pass the cut

write the skimmed data and r2 to a new ROOT file

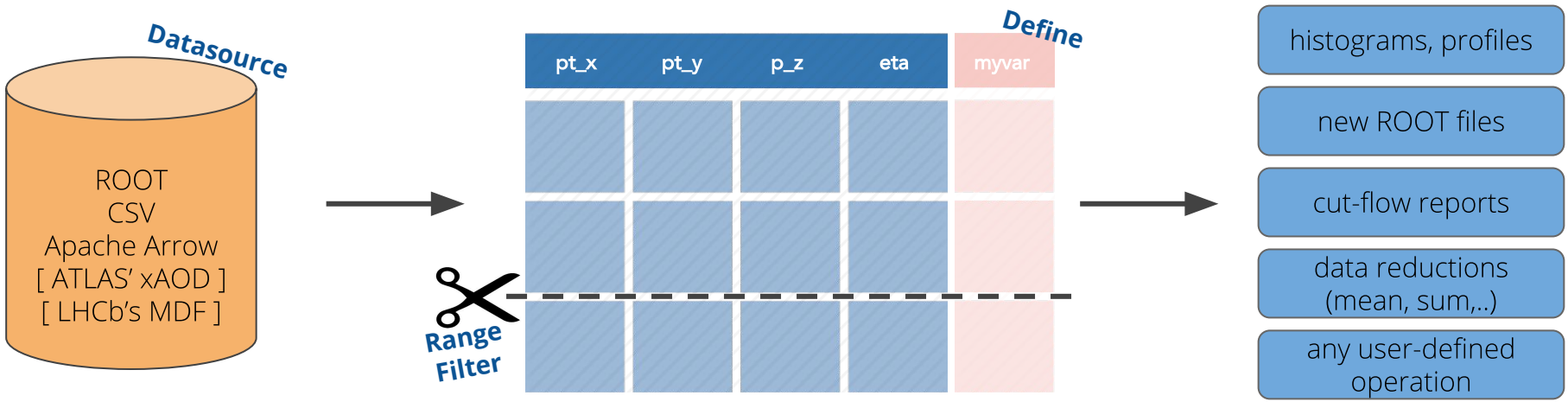
Lazy execution guarantees that all operations are performed in **one event loop**

Analyses as computation graphs

```
ROOT::EnableImplicitMT();
ROOT::RDataFrame df(dataset);
auto df2 = df.Filter("x > 0")
             .Define("r2", "x*x + y*y");
auto rHist = df2.Histo1D("r2");
df2.Snapshot("newtree", "newfile.root");
```



RDataFrame design *goals*



- ➔ being the fastest (most performant, easiest to work with) way to manipulate HEP data
- ➔ being the go-to ROOT analysis interface from 1 to 100 cores, laptop to cluster, with little to no change in user code
- ➔ full support for and consistent interfaces in both python and C++

Design principles

Elements of **declarative programming** (“user says what, ROOT chooses how”):

high level interfaces provide less typing, increased readability, abstraction of complex operations

. . . . . .

...and allow **transparent optimisations**, e.g. multi-thread parallelisation, lazy evaluation and caching

Design principles

Elements of **declarative programming** ("user says what, ROOT chooses how"):

high level interfaces provide less typing, increased readability, abstraction of complex operations

...and allow **transparent optimisations**, e.g. multi-thread parallelisation, lazy evaluation and caching

---

Elements of **functional programming** (pure functions, higher level functions):

push users towards coding in terms of **small reusable components**

less side-effects and less shared state increase **thread-safety and code correctness**

No templates: C++ → JIT → python

C++

```
d.Filter([](double t) { return t > 0.; }, {"th"})
.Snapshot<vector<float>>("t","f.root",{"pt_x"});
```

No templates: C++ → JIT → python

C++

```
d.Filter([](double t) { return t > 0.; }, {"th"})
.Snapshot<vector<float>>("t","f.root",{"pt_x"});
```

C++ with cling's just-in-time compilation

```
d.Filter("th > 0").Snapshot("t","f.root","pt_x");
```

No templates: C++ → JIT → python

C++

```
d.Filter([](double t) { return t > 0.; }, {"th"})
.Snapshot<vector<float>>("t","f.root",{"pt_x"});
```

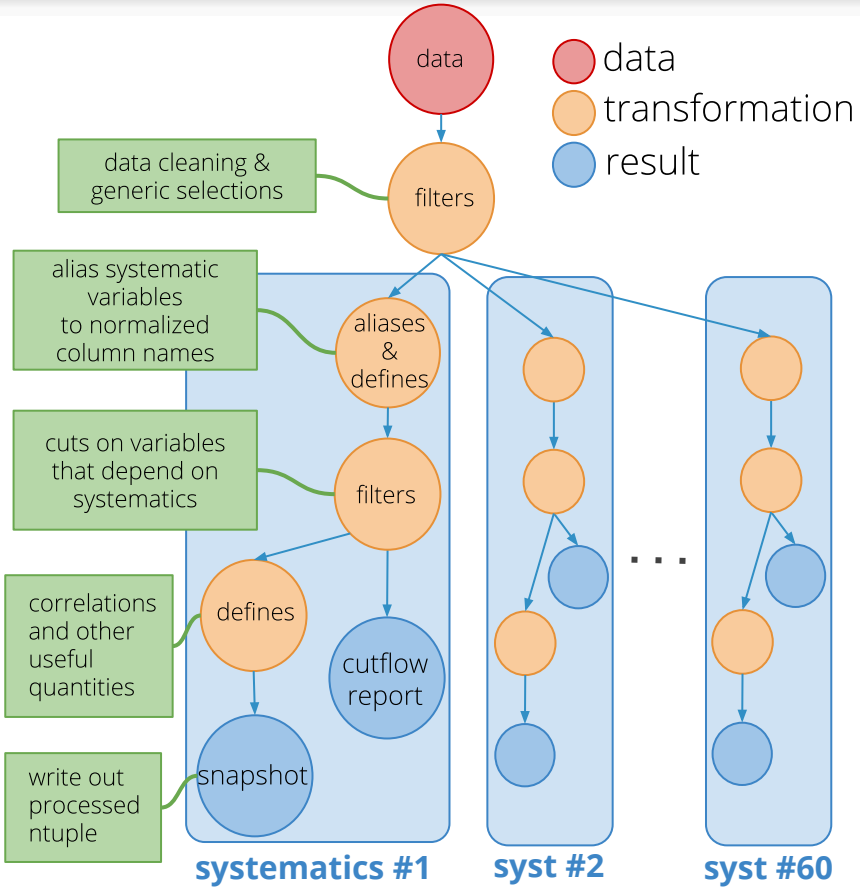
C++ with cling's just-in-time compilation

```
d.Filter("th > 0").Snapshot("t","f.root","pt_x");
```

PyROOT, automatically generated python bindings

```
d.Filter("th > 0").Snapshot("t","f.root","pt_x")
```

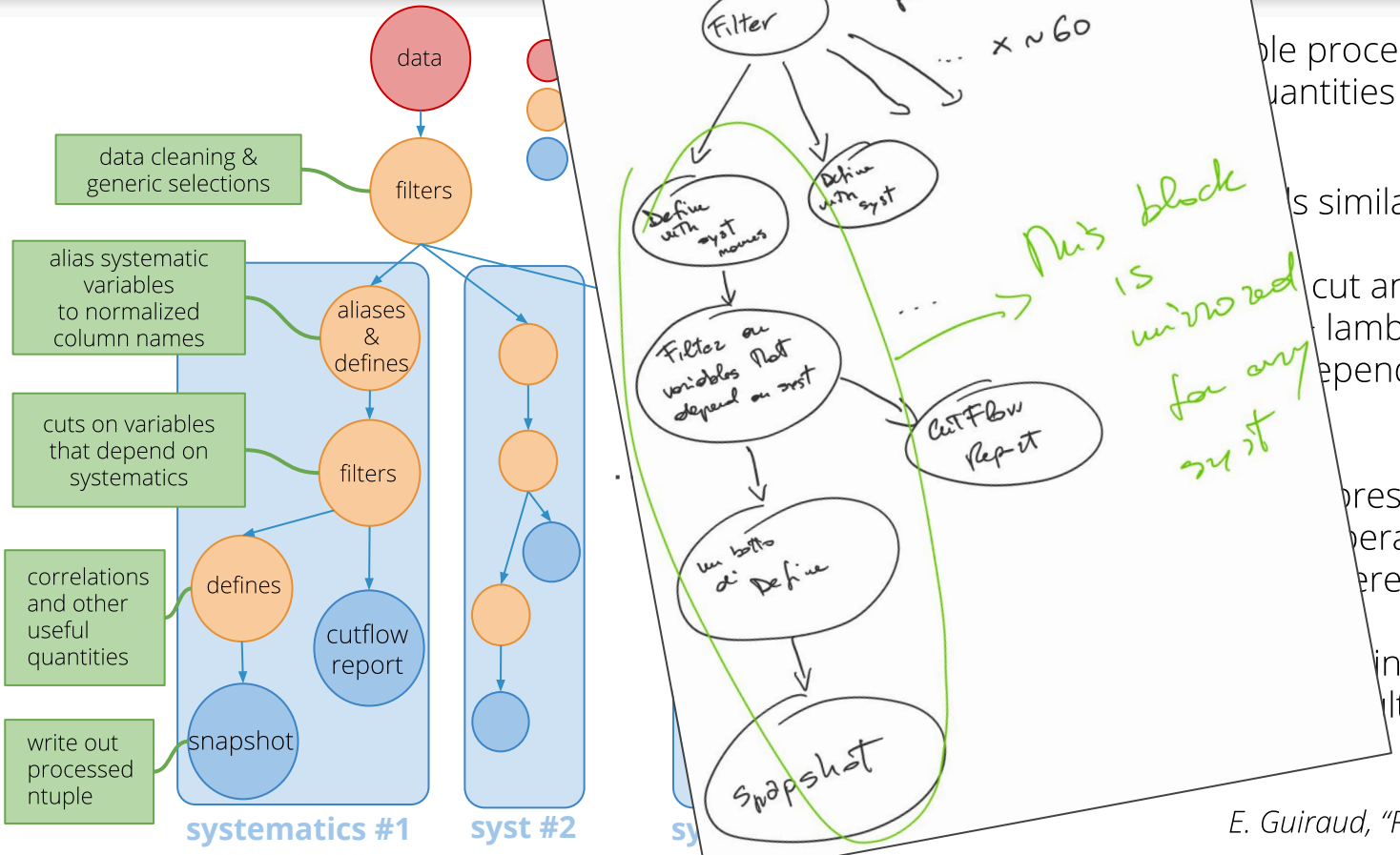
Case study: ATLAS SUSY ntuple → ntuple



Local ntuple → ntuple processing, MC data is processed to add quantities relevant for publication

- ➔ program's **main** reads similarly to this graph
- ➔ the definition of each cut and new quantity is encapsulated in a C++ lambda or a free function that can be tested independently from the rest of the code
- ➔ the large blue boxes represent one single function that applies the same operations to an RDF variable and is re-used for all different systematics
- ➔ cuts, calculations and writing of the 60 output trees all happen in the same multi-thread event loop

Case st tuple → ntuple



ble processing, MC data is
uantities relevant for publication

ls similarly to this graph

cut and new quantity is
lambda or a free function
ependently from the rest of

resent one single function
erations to an RDF variable
erent systematics

ing of the 60 output trees
lti-thread event loop

High-level customization points: RDataSource

➔ RDataFrame **can read non-ROOT data** through RDataSource objects

➔ **third parties** can implement and **seamlessly integrate** RDataSource implementations for their format of choice

High-level customization points: RDataSource

- ➔ RDataFrame **can read non-ROOT data** through RDataSource objects
- ➔ **third parties** can implement and **seamlessly integrate** RDataSource implementations for their format of choice
- ➔ CSV and Apache Arrow currently supported via RDataSource
- ➔ prototypes for LHCb's MDF binary data format and ATLAS' xAOD event model

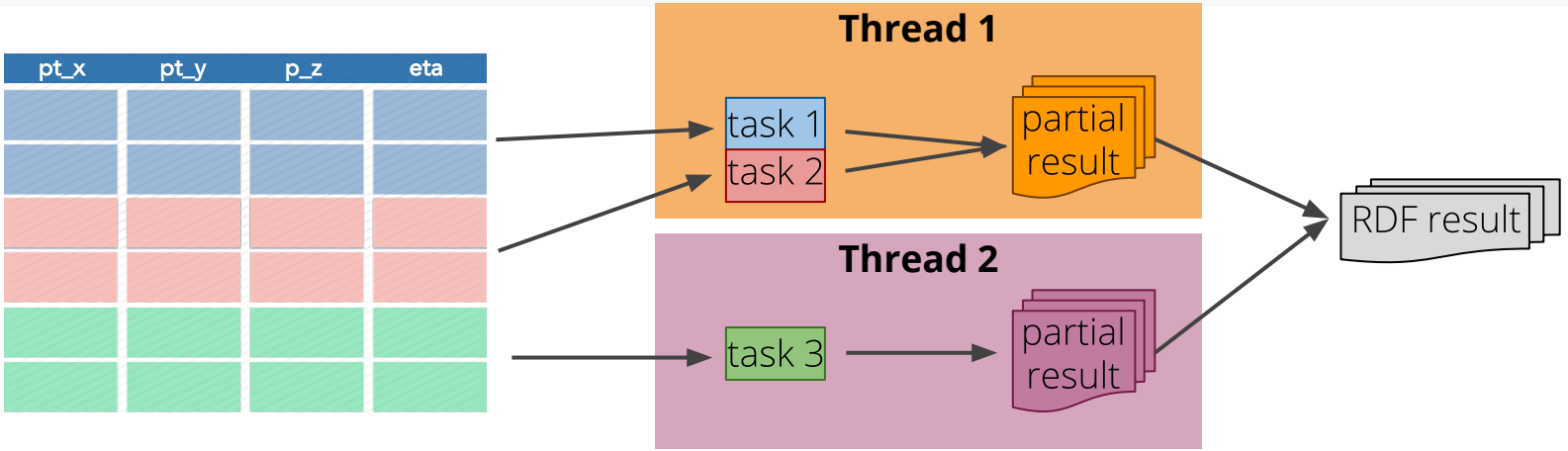
DOI 10.5281/zenodo.1303038

High-level customization points: RDataSource

- ➔ RDataFrame **can read non-ROOT data** through RDataSource objects
- ➔ **third parties** can implement and **seamlessly integrate** RDataSource implementations for their format of choice
- ➔ CSV and Apache Arrow currently supported via RDataSource
- ➔ prototypes for LHCb's MDF binary data format and ATLAS' xAOD event model

Users can write the **same code independently of the data format** analyzed

RDataFrame's parallelization scheme



Task-based parallelism

- ➔ each task processes a range of entries (thanks to inherent independence of HEP events)
- ➔ **cannot overcommit**, plays well with e.g. experiment frameworks
- ➔ range granularity is the same as TTree compression's to **avoid redundant decompressions**
- ➔ **Intel TBB** is currently ROOT's task scheduler and thread pool manager
- ➔ **RDF parallel *writing*** is also task-based, see G. Amadio, "Writing ROOT Data in Parallel", Track 5

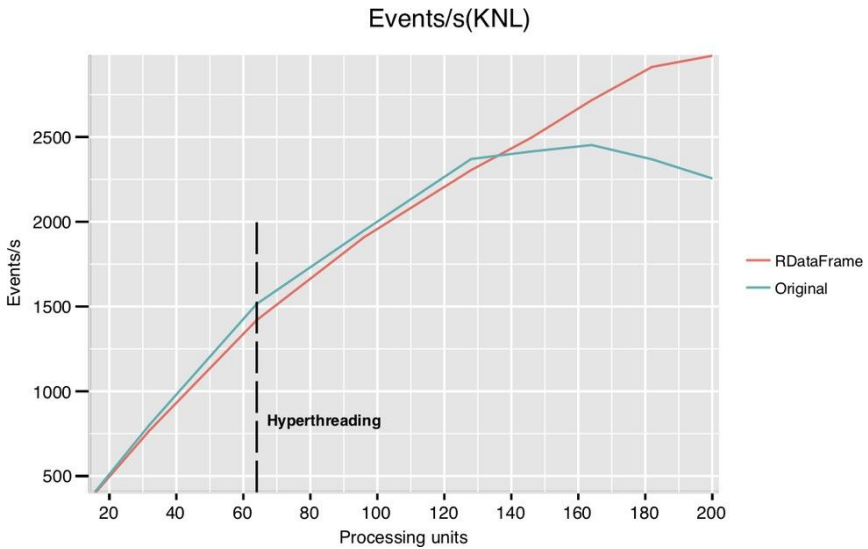
Does it scale? Is it fast?

No disk reads, KNL, 64 physical cores

Monte Carlo QCD Low-Pt events generation+ analysis on the fly

Ad-hoc implementation (patched ROOT 5 + POSIX threads) vs RDF

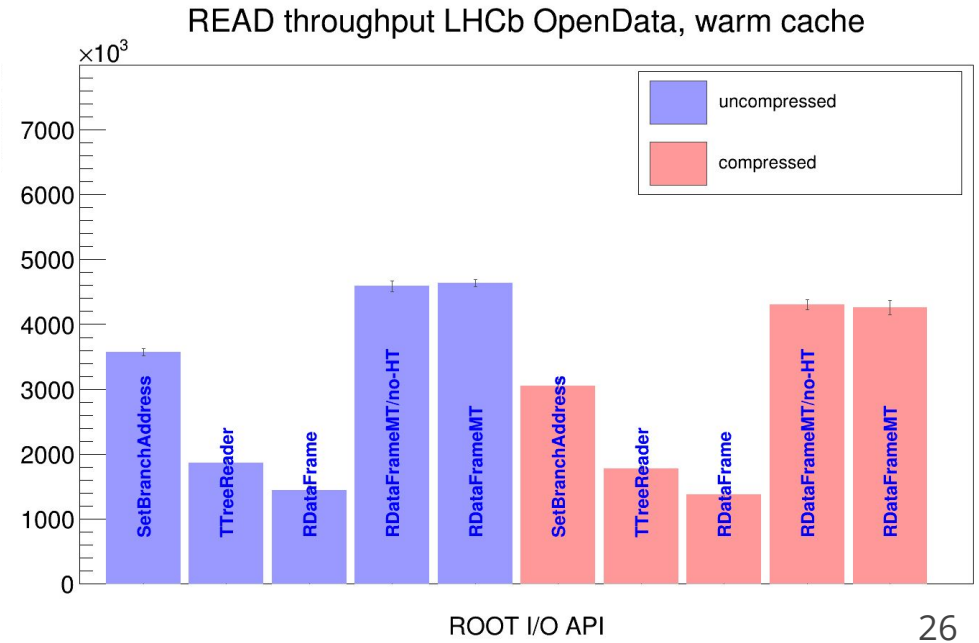
Performance analysis by X. Valls Pla



Read speed on SSD, 4 physical cores @ 3.6GHz

TTree+SetBranchAddress vs TTreeReader vs RDataFrame

Original results by J. Blomer



Does it scale? Is it fast?

No dis

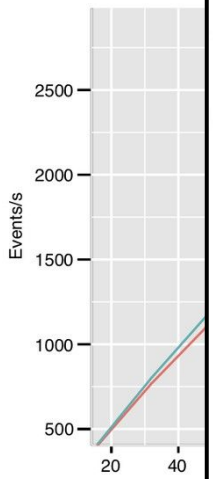
Monte Carlo C

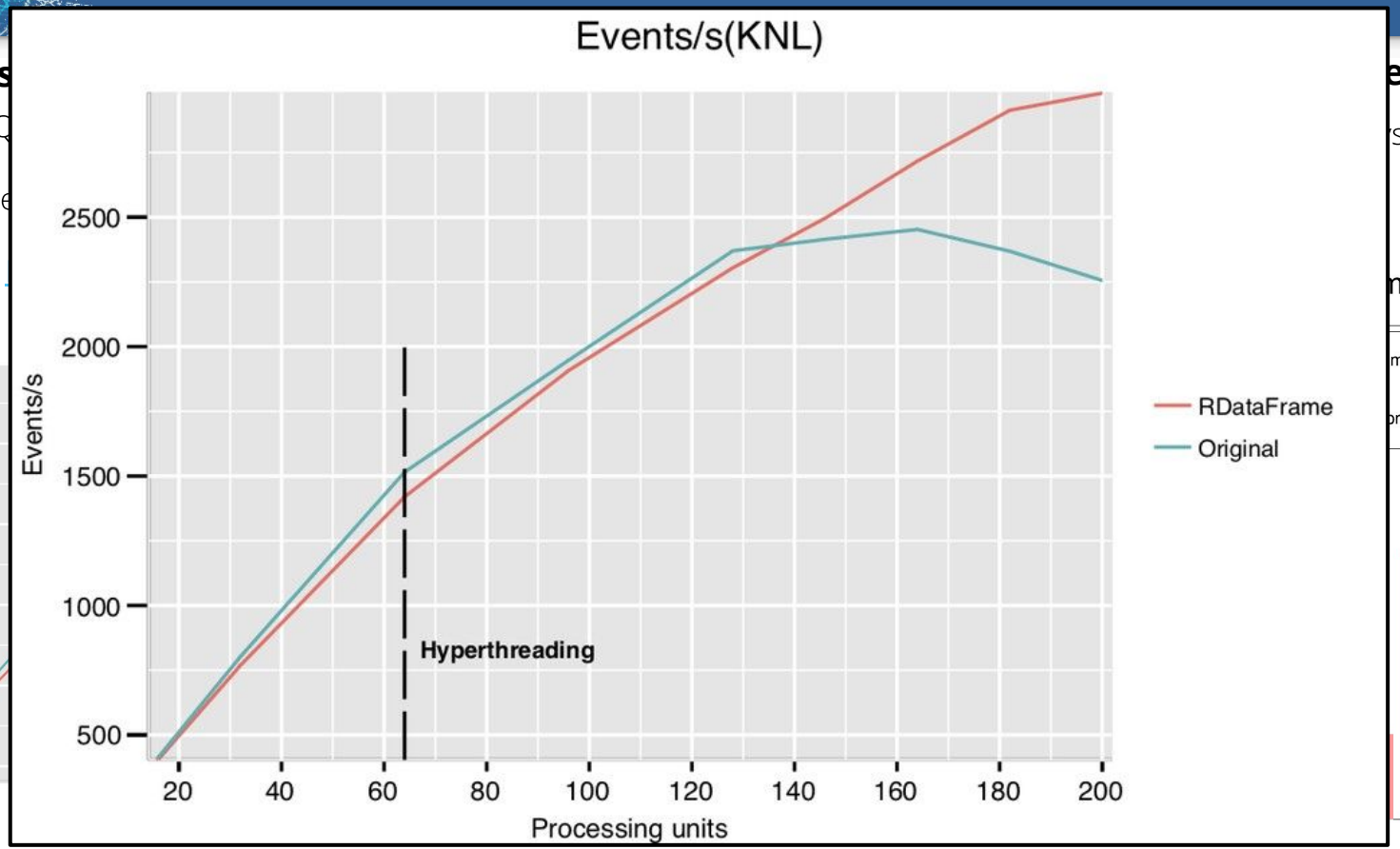
Ad-hoc impleme

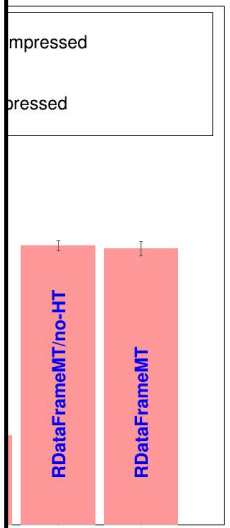
es @ 3.6GHz

's RDataFrame

n cache







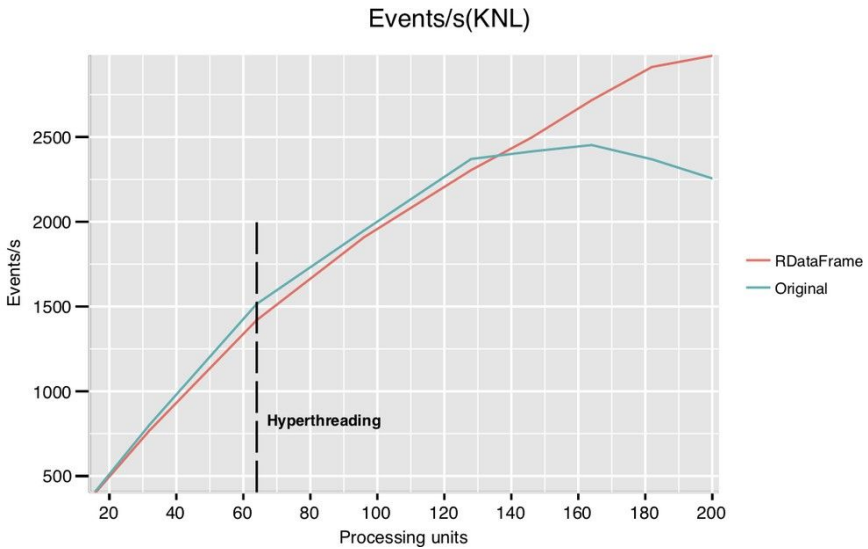
Does it scale? Is it fast?

No disk reads, KNL, 64 physical cores

Monte Carlo QCD Low-Pt events generation+ analysis on the fly

Ad-hoc implementation (patched ROOT 5 + POSIX threads) vs RDF

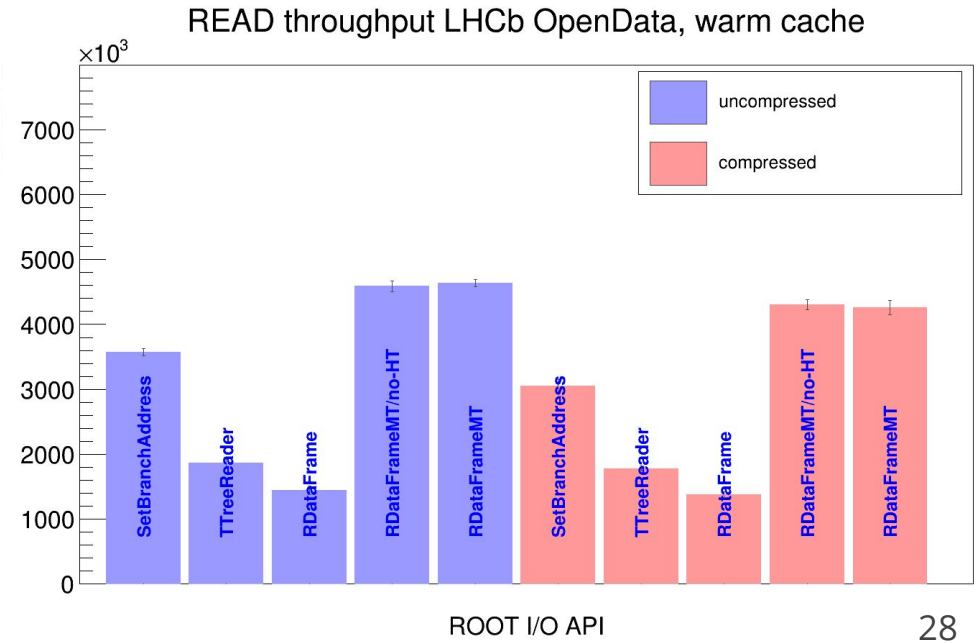
Performance analysis by X. Valls Pla



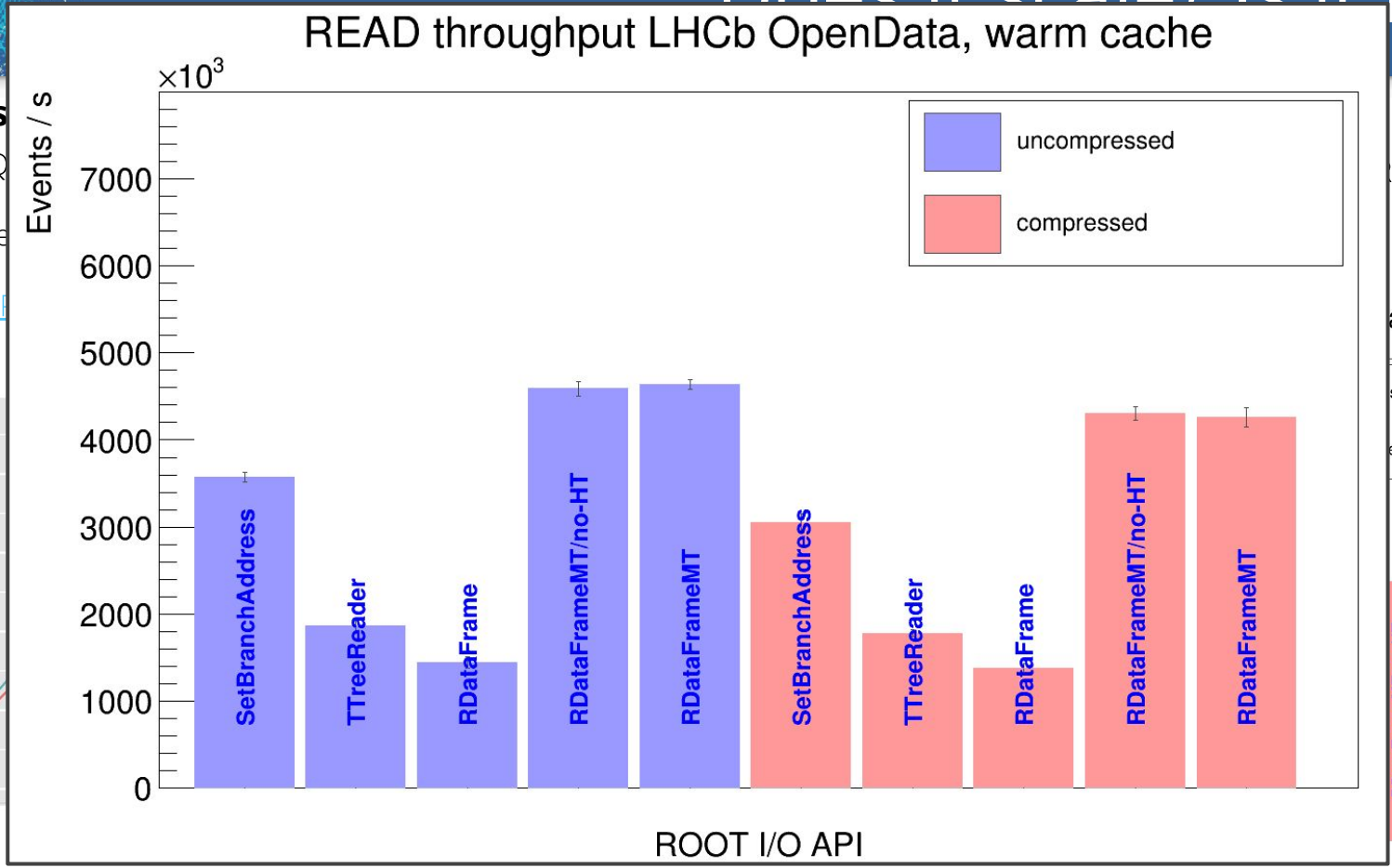
Read speed on SSD, 4 physical cores @ 3.6GHz

TTree+SetBranchAddress vs TTreeReader vs RDataFrame

Original results by J. Blomer



Does it scale? Is it fast?



ROOT I/O API

Summary, outlook

- ROOT provides a **modern, high-level, type-safe, parallel** interface for data analysis and manipulation

- **RDataFrame** is available since ROOT v6.14
  - performant, scales to many-core architectures,
  - has already been used successfully by physicists of major LHC experiments

Summary, outlook

- ROOT provides a **modern, high-level, type-safe, parallel** interface for data analysis and manipulation

- **RDataFrame** is available since ROOT v6.14
  - performant, scales to many-core architectures,
  - has already been used successfully by physicists of major LHC experiments

For the future

- more pythonic pyROOT bindings (conversion to/from numpy, python lambdas, …)
- **distributed execution of RDataFrame analyses**: working prototype for python+Spark
- integration with TMVA's inference layer
- low-level performance optimization