HSF
HEP Software Foundation

University of Glasgow | School of Physics & Astronomy

Parallel Processing in HEP

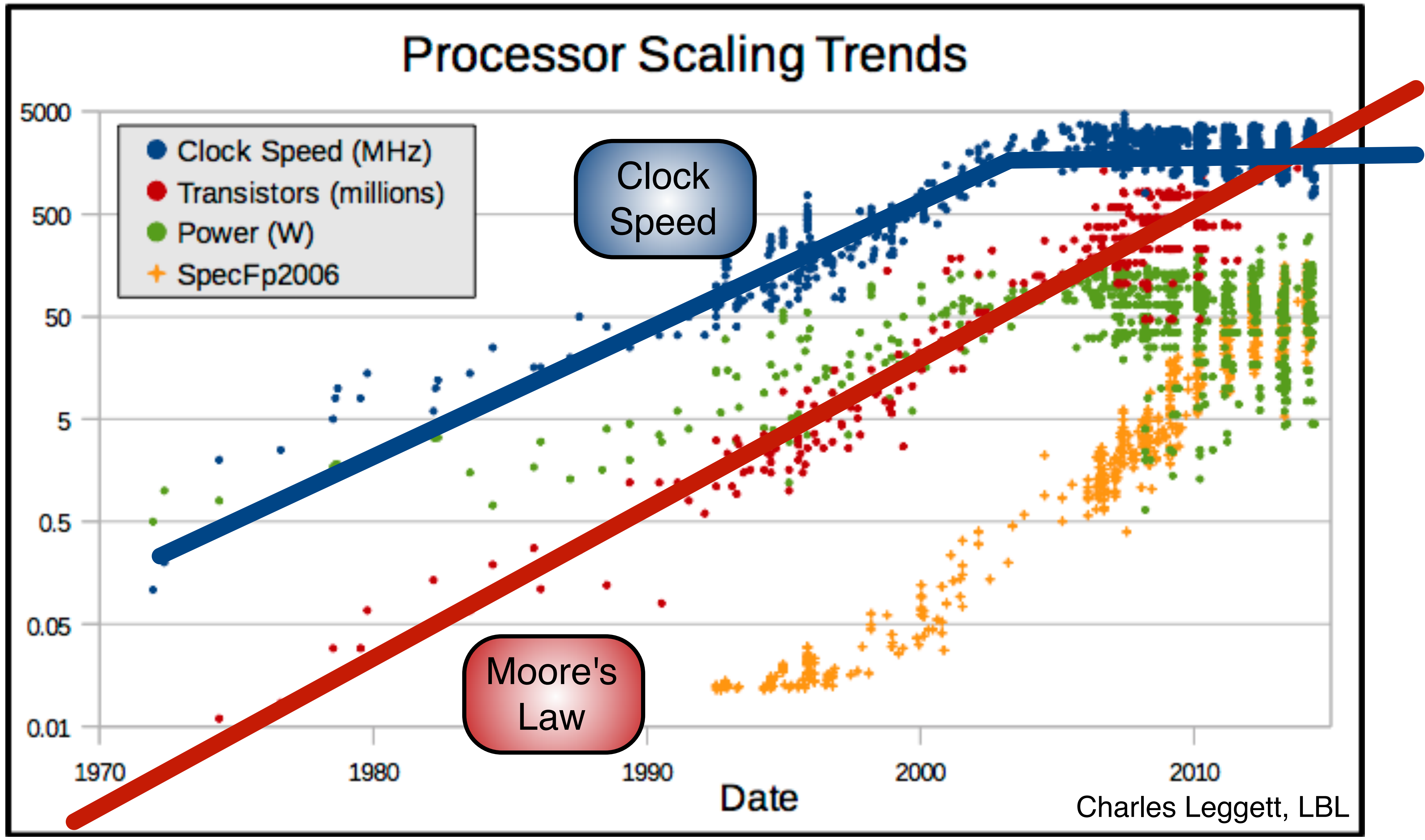
Graeme Stewart

HEP Software

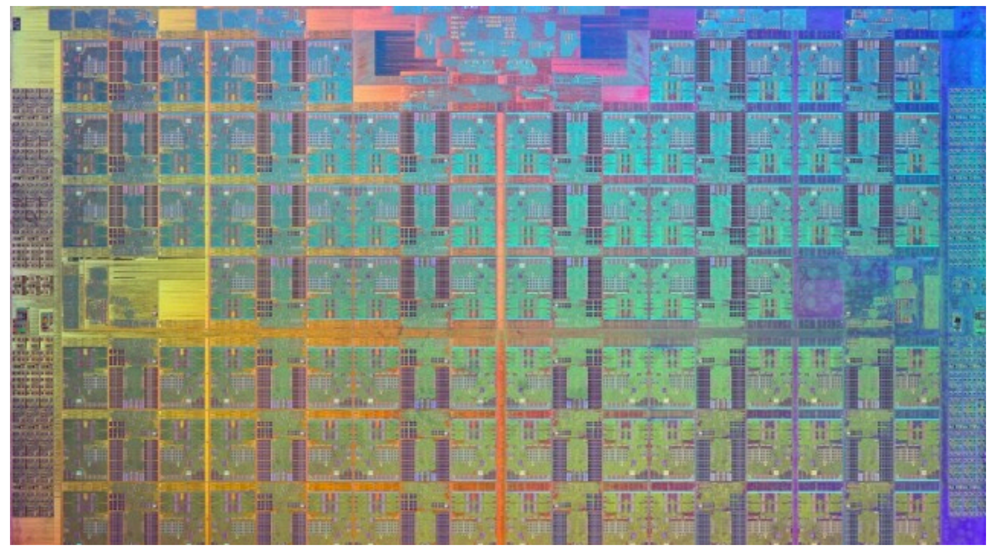
- High Energy Physics has a vast investment in software
  - Estimated to be around 50M lines of C++
  - Which would cost several N x 100M$ to develop commercially (N ~ 5)
- It's a critical part of our physics production pipeline, from triggering all the way to analysis and final plots
- LHC experiments use about 600k CPU cores every hour of every day and have around 400PB of data stored on disk and 600PB on tape
- The costs are then really significant and ongoing with challenges in front of us
  - HL-LHC, Intensity Frontier, Concurrency, Language Modernisation, …
- Projection is that raw hardware improvements (even if we can fully utilise them) will fall short of HL-LHC needs by about x10
  - We need to substantially improve our software

CPU Processor Evolution

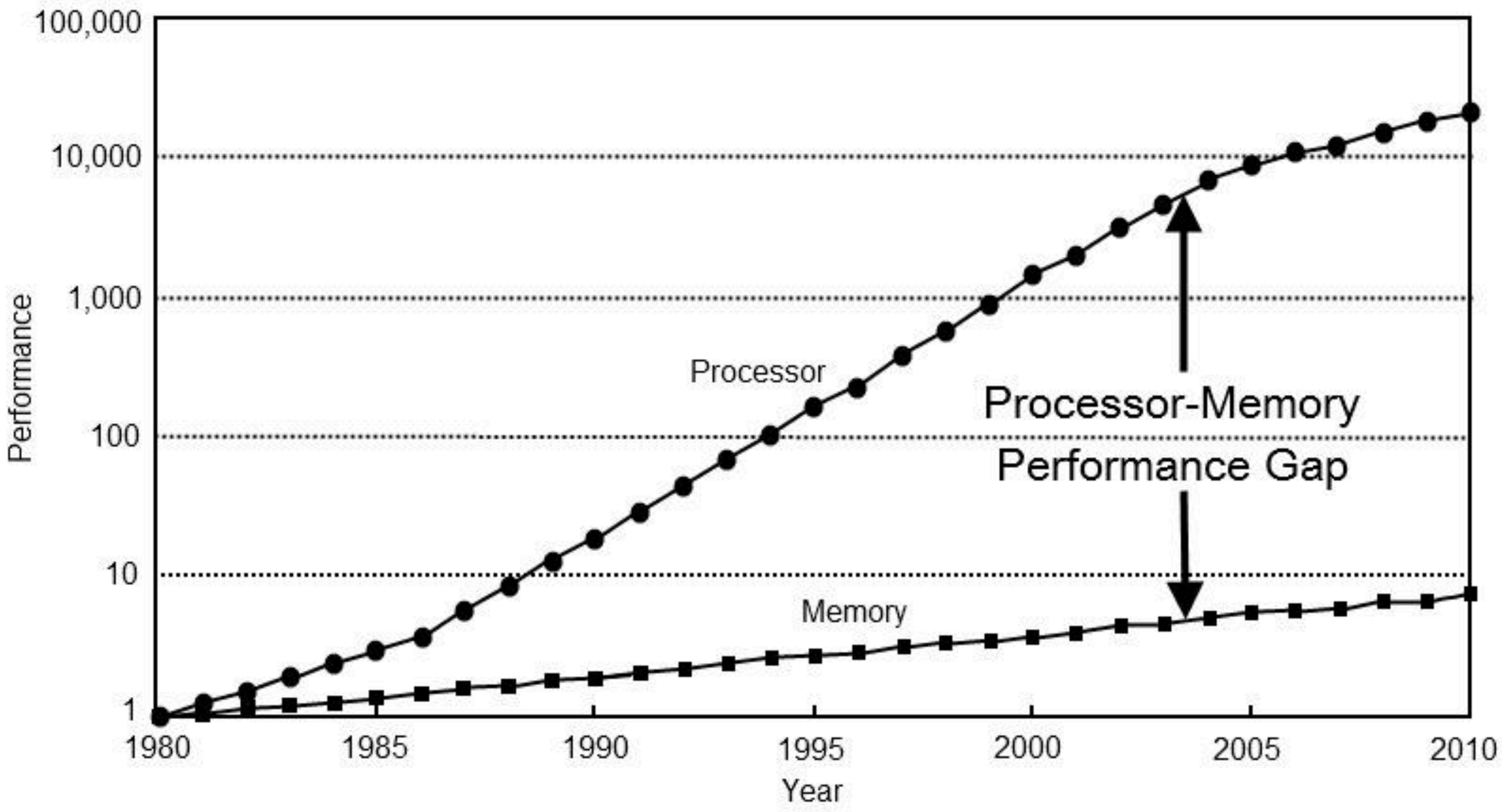
- Moore’s Law continues to deliver increases in transistor density
    - Doubling time is lengthening
    - IBM recently demonstrated 5nm wafer fabrication
- Clock speed scaling failed around 2006
    - No longer possible to ramp the clock speed as process size shrinks
        - Leak currents become important source of power consumption
    - So we are basically stuck at ~3GHz clocks from the underlying Wm^{-2} limit
    - This is the *Power Wall*



Current Silicon



- As performance increases cannot come from clock speed anymore they need to come from other avenues:
  - Processor Cores
    - Growing number of CPU cores in servers and in laptops
    - New low power many-core designs, e.g., Xeon Phi with 64 cores and 256 threads
  - Wide Registers
    - CPU register widths have grown from 64 to 512bits in x86_64 (our workhorse)
      - This is 8 doubles or 16 floats wide
  - Larger Caches and on Package Memory
    - Increased cache sizes help with memory latency, but real value scales sub-linearly
  - Improved micro-architectures



Increasing CPU FLOP performance opens a huge gap compared to memory performance

Drivers for the Future

- Low power devices
  - Driven by mobile technology and Internet of Things
- Data centre processing
  - Extremely large clusters running fairly specialist applications
- Machine learning
  - New silicon devices specialised for training machine learning algorithms, particularly low precision calculations
- Exascale computing
  - Not in itself general purpose, but poses many technical problems whose solutions can be general
- Energy efficiency is a driver for all of these developments
  - Specialist processors would be designed for very specific tasks
  - Chips would be unable to power all transistors at once: dark silicon is unlit when not used

There will be no longer a one-size-fits all hardware and our code will need to adapt to multiple architectures

Challenges for HEP

- In contrast to the days of the free lunch* HEP now has to do far more work to exploit modern hardware effectively at the same time as our computing needs are growing hugely
- Also, we do not start from scratch, but from a large legacy code base of millions of lines of C++
  - All written with a different processing model in mind, often decades ago
  - We will need architecture awareness, but also keep the code maintainable
- We need to equip the community with new skills: proper knowledge and effective training
- However, these are not challenges we face alone
  - Many other scientific disciplines face similar issues in data scales
  - Industry considers us to be rather interesting
    - We use a lot of computing and we have problems that are hard to solve
    - Matches their desire to develop tools to make modern hardware more useful for users
- Where are we active? How does that related to other fields? How can we interact for mutual benefit?

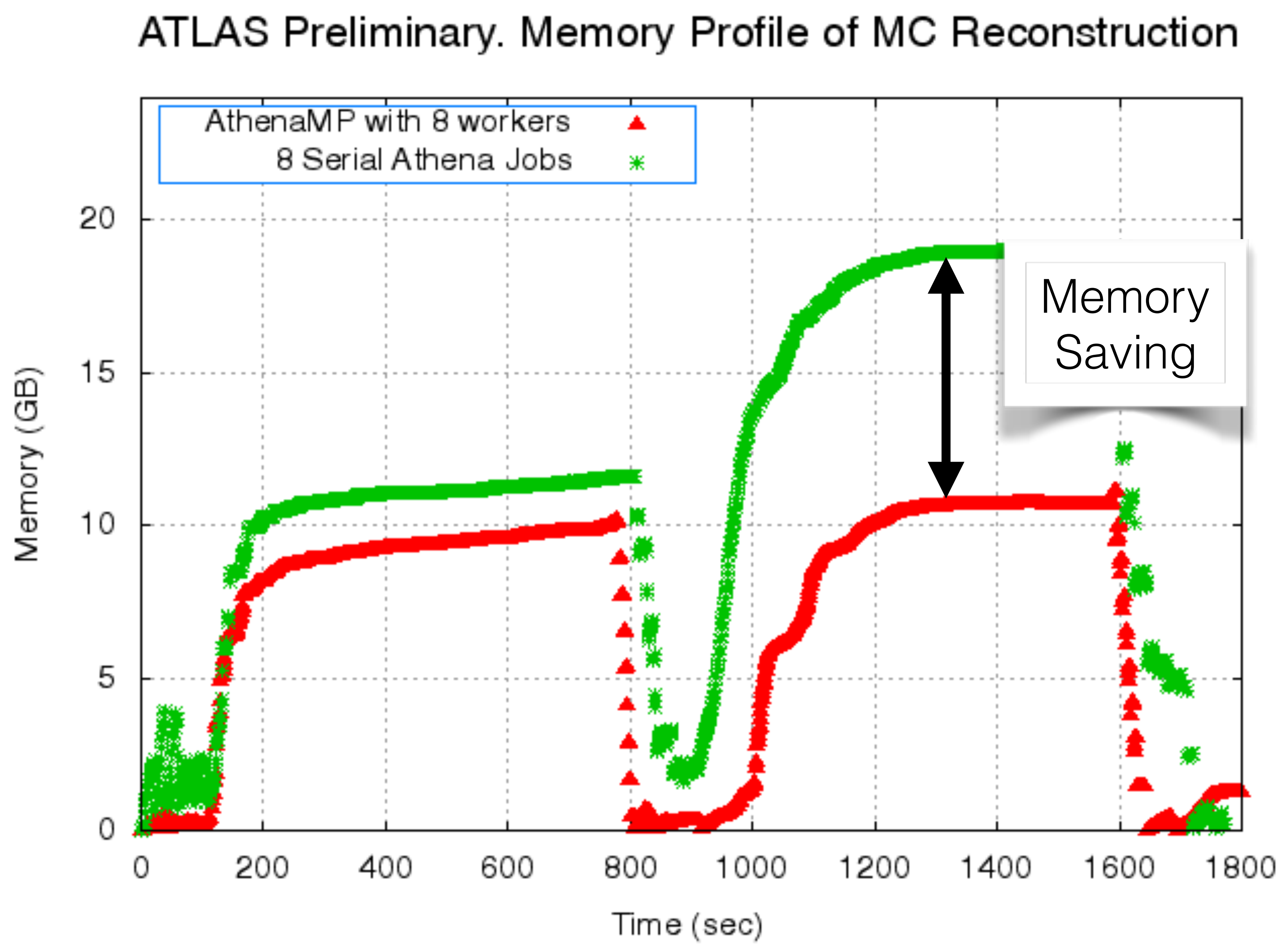
*Clock scaling

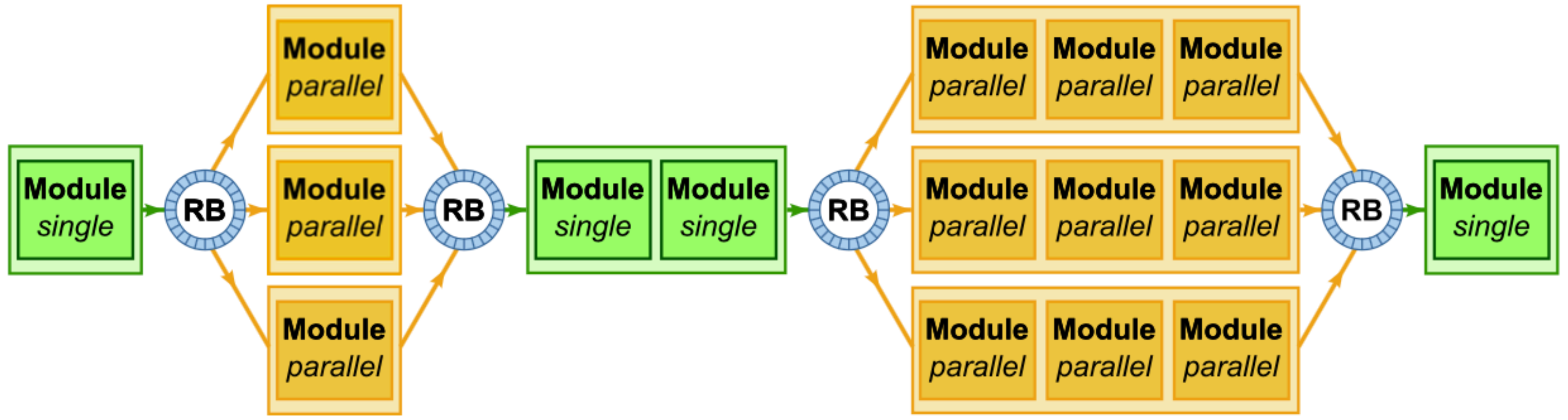
Climbing the Memory Wall

- For the big LHC experiments shear memory consumption is a major issue
    - Complicated detectors with 100M+ channels, large field maps, precision geometry and material maps
    - All this is needed to support a precision physics program
- In the early days, we adapted to multi-core architectures simply by running multiple independent jobs on server machines
    - However, memory was used poorly and it was rarely possible to exploit hyper-thread cores
        - So losing 15-20% of potential throughput with our current code
- Memory per core has to be reduced and effective use of memory hierarchy becomes vital
- The *Memory Wall* is both size and rate
    - N.B. energy costs of accessing DRAM are high

Multi-Processing

- As most jobs shared large static memory structures between them a multi-processing approach could save memory
- Initialise a job using a single core
- Then fork event processing workers as needed
    - Linux kernel will share the physical pages associated with immutable data (known as *Copy on Write*)
    - Significant resource savings
- Worker processes can manage the whole event loop (ATLAS) or only pieces of it (Belle II)
    - Simplifies the work needed to ensure that all event bookkeeping is managed properly





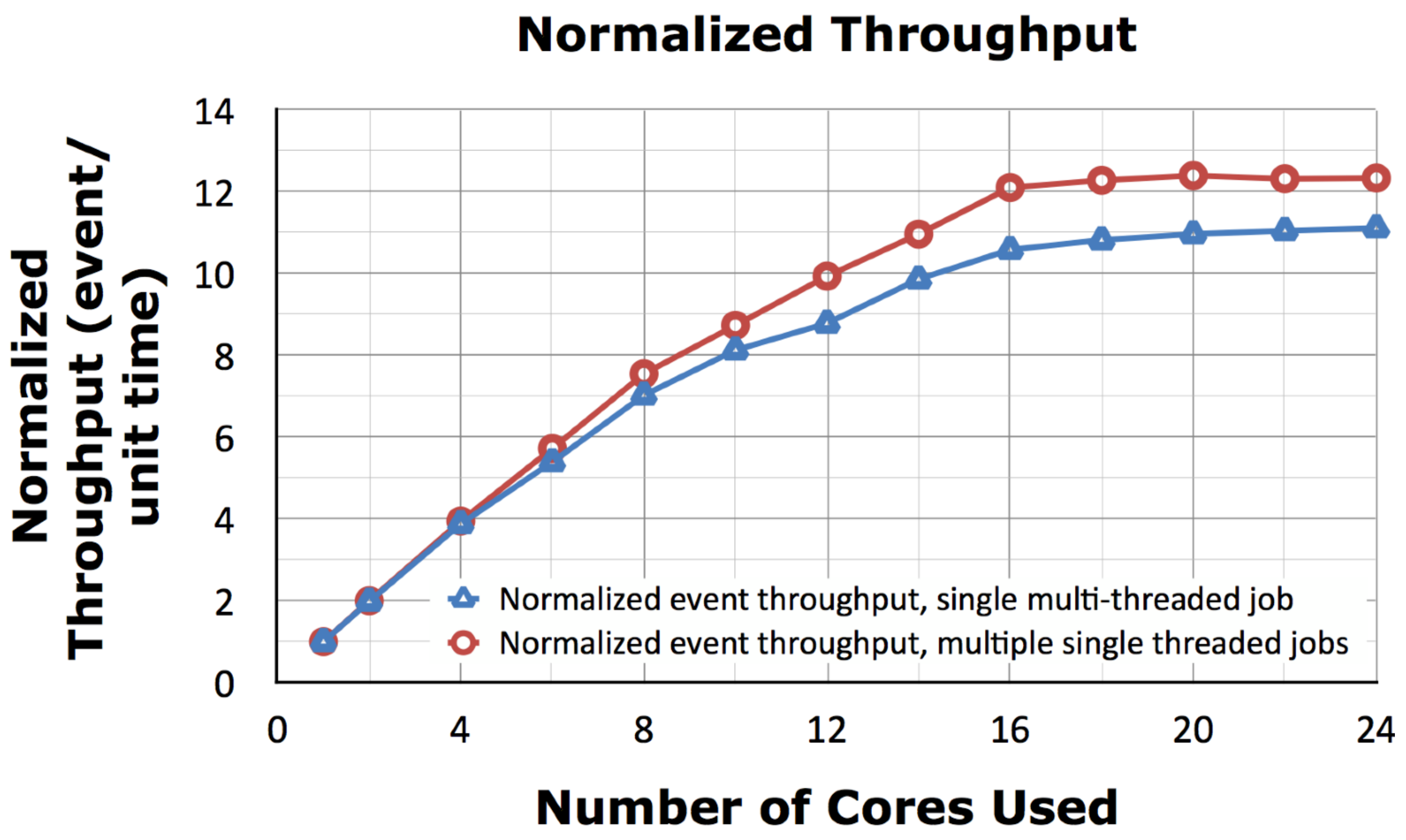
Belle II multi-processing parallelises only slower parts of the HLT saving even more resources

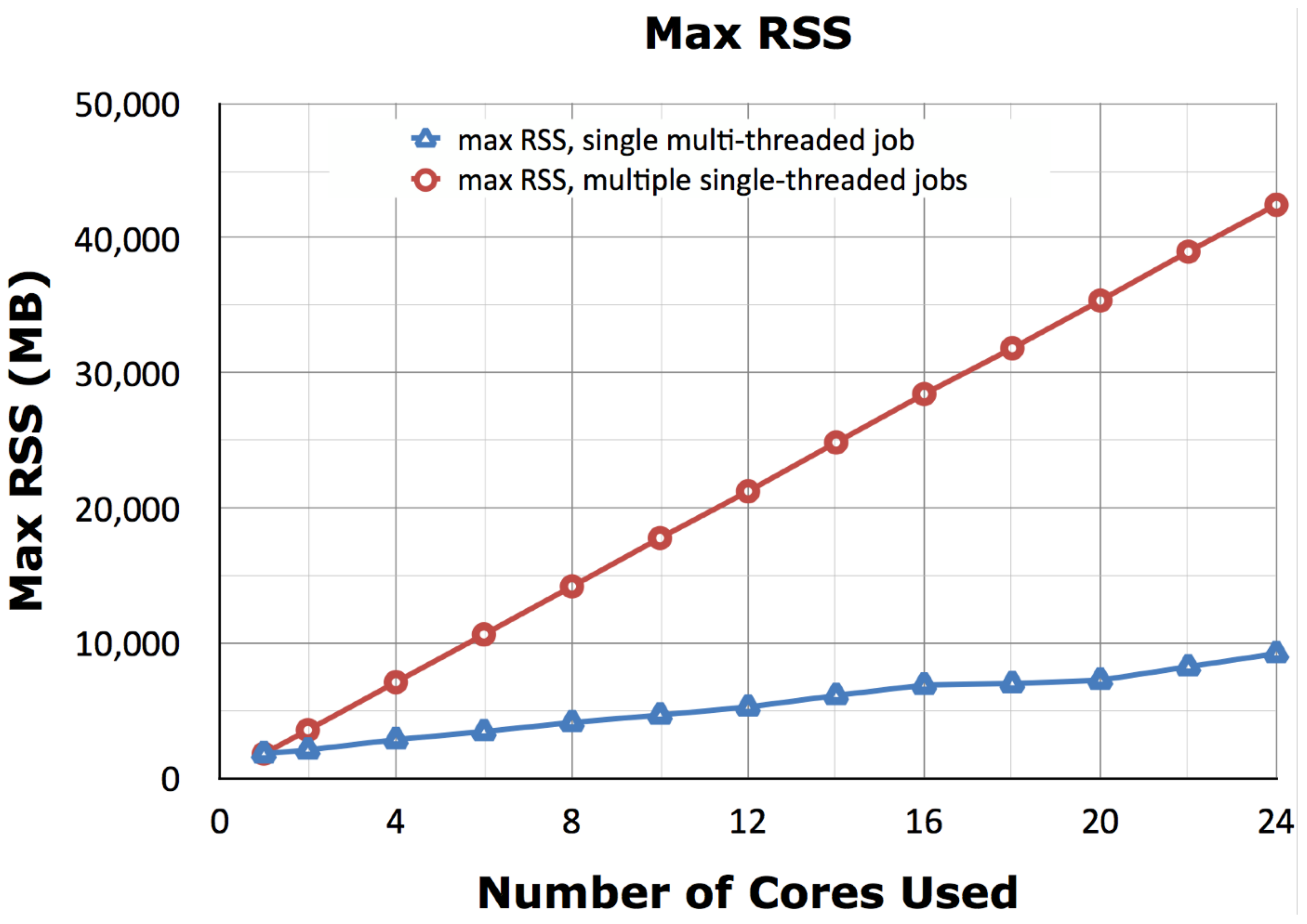
Multi-Threading Frameworks

- Memory sharing with forked multi-processes is pretty crude
- Multi-threaded programming shares memory much more effectively
  - Needed by CMS and ATLAS for sure
  - But is a much harder model to program against — easy to violate some assumptions about the state of shared data objects
- Experiment frameworks provide the scaffolding that the physics code uses
  - Big community effort to understand how to do this in the Concurrency Forum (e.g., identifying the best libraries to use)
  - Try to handle as much of the difficult parts of multi-threading in the framework itself
    - e.g., the scheduler should ensure that physics algorithms cannot run until their input data is ready and no consumer of their output will run until it has finished (there are O(100) algorithms typically)
      - Avoid many issues with data races
  - We need to train all developers not to do destructive things and train experts in the right key skills
- For the case of simulation we are helped a lot because Geant4 and GeantV have developed multi-threading capabilities

CMSSW

- CMS pioneered multi-threading among the LHC experiments
  - Implemented the design pattern identified in the community
  - Built a framework that could adapt to different levels of thread safety and factorised tacking problematic code
- Now multiple techniques employed to mitigate blocking and stalls from contention or state changes
  - Data prefetching, in-event parallelism
- Serial bottlenecks become worse as number of threads increases (a.k.a. Amdahl's Law)
  - However, good performance now seen even with many-core machines like the Xeon Phi, running up to 240 threads

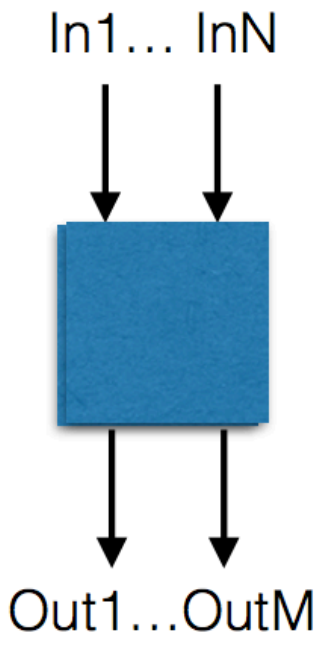




Christopher Jones, FNAL

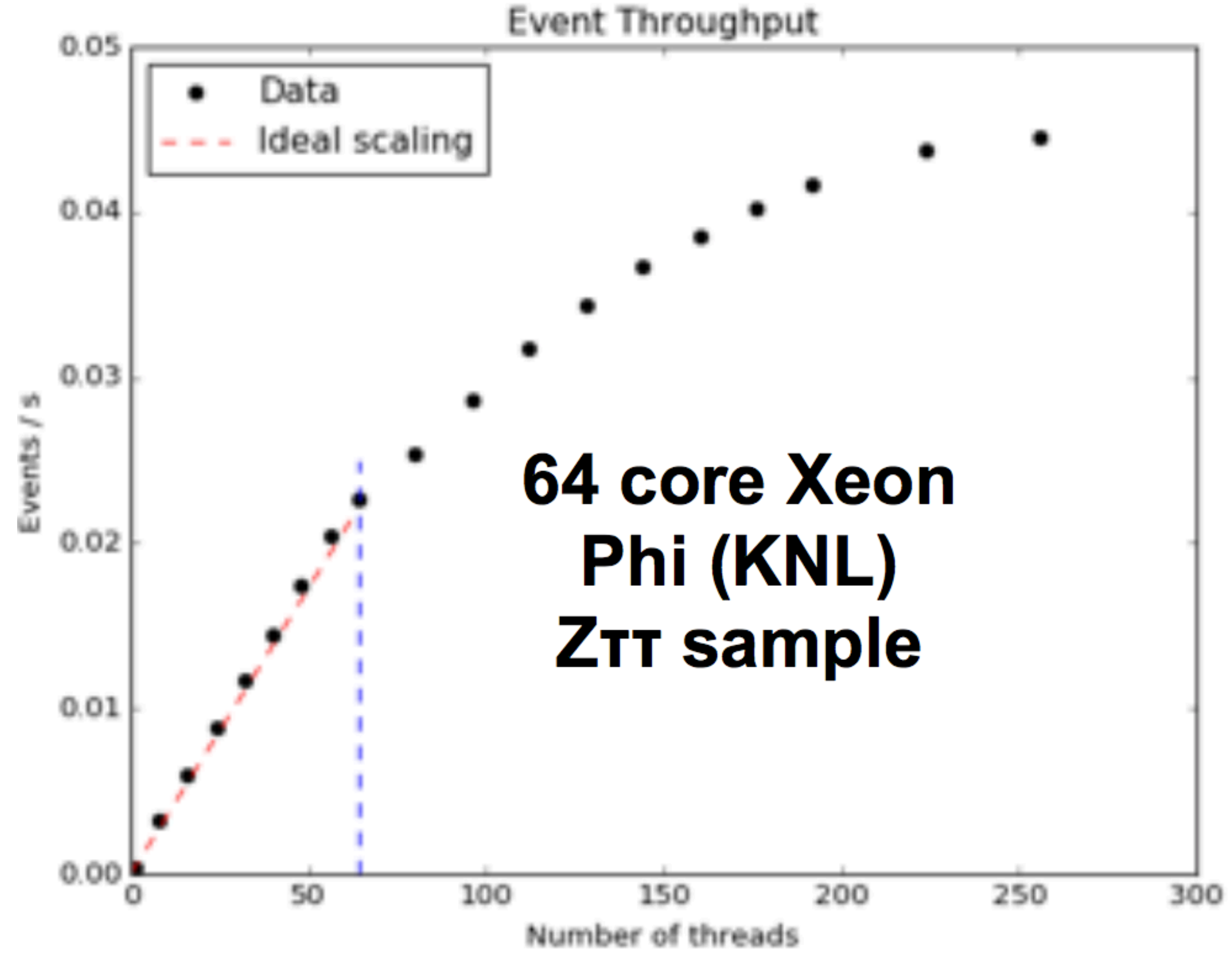
Gaudi

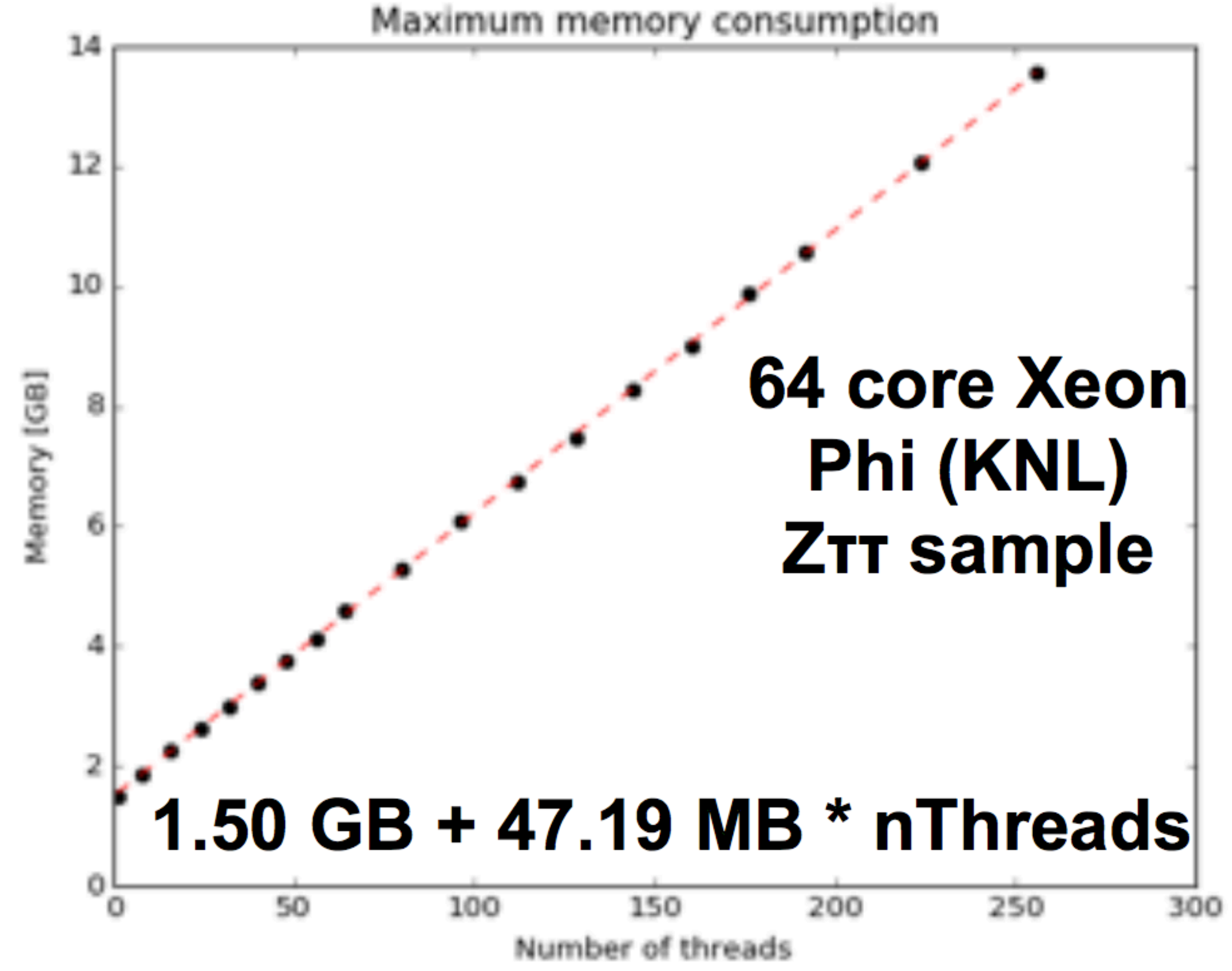
- Gaudi framework, used by ATLAS and LHCb, also evolving towards multi-threading
  - Data flow implements protection against basic data races
  - Control flow implements logic needed for early rejection in the trigger
    - Combined into an efficient single processing graph
- A more radical approach, pursued by LHCb, is to use functional programming patterns
  - Here an algorithm ‘declares’ its data dependencies as part of its method signature
  - This removes all state from its declaration
    - Even the event store might be optional




```
template <typename ... Out, typename... In, typename Traits_>
class MultiTransformer<std::tuple<Out...>(const In&...),Traits_>
{
   virtual std::tuple<Out...> operator()(const In&...) const = 0;
}
```

Example of a Gaudi Functional *MultiTransformer,*
Gerhard Raven, NIKEF

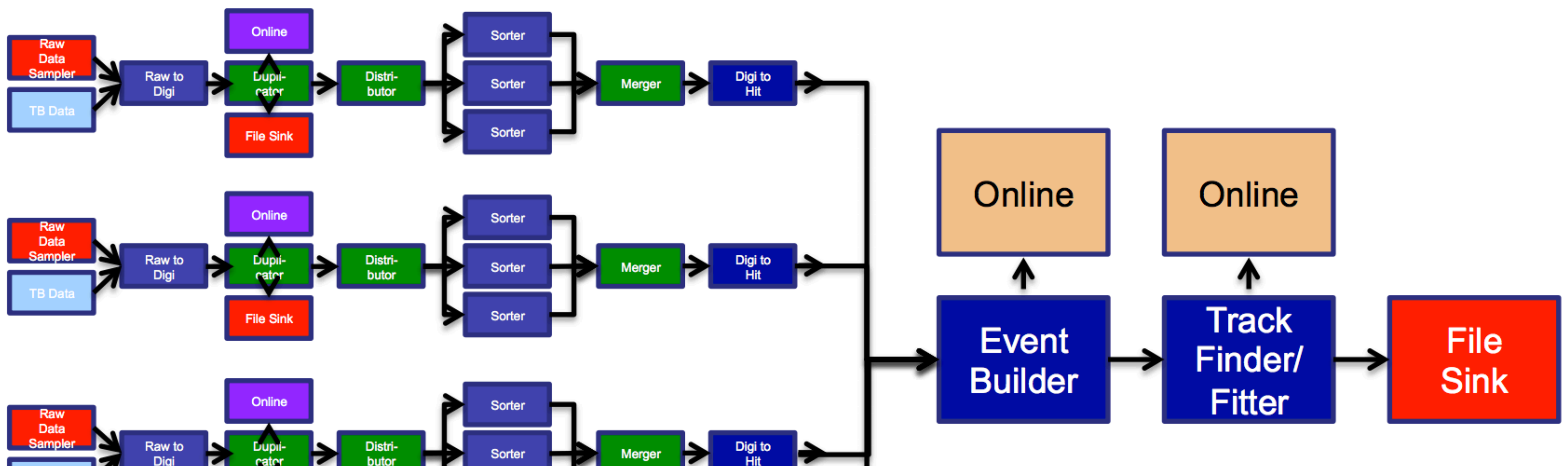




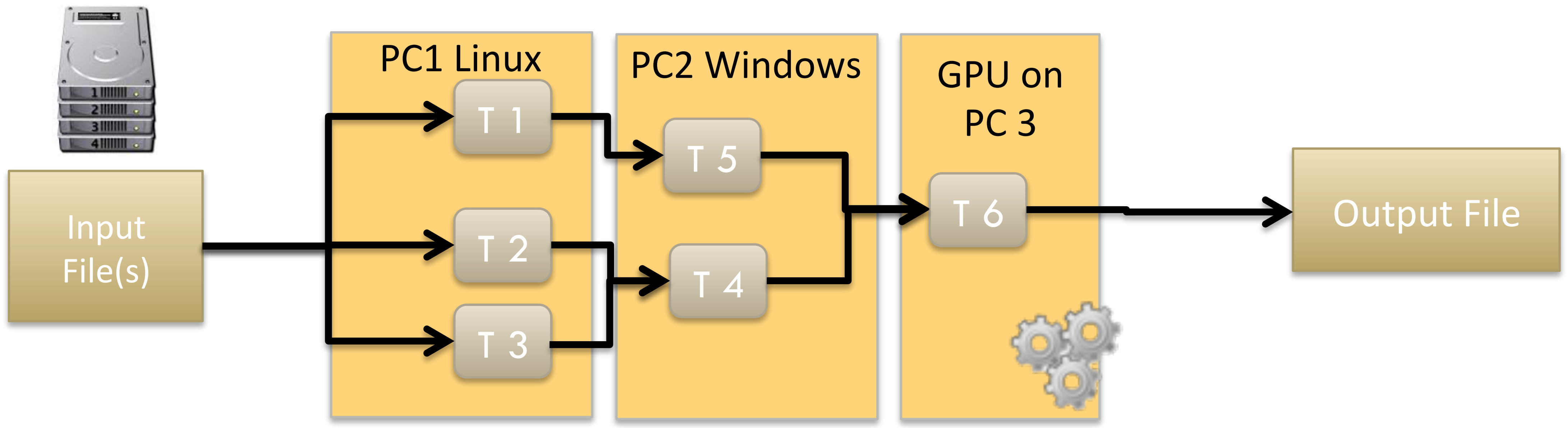
Performance of ATLAS Simulation using multi-threaded Gaudi framework
Steve Farrell, LBL

FairROOT and O2

- Multi-threading is not the only way to decompose processing
  - As we saw already multi-processing can be used
- FAIR experiments and ALICE developing this model to allow for heterogeneous continuous processing of their data
  - Topology of independent processing units, connected by message queues
  - Multiple message serialisation technologies supported
- Fits very well for a data reduction pipeline — no binary trigger decisions, instead keep most interesting pieces of events



Multiple input pipelines

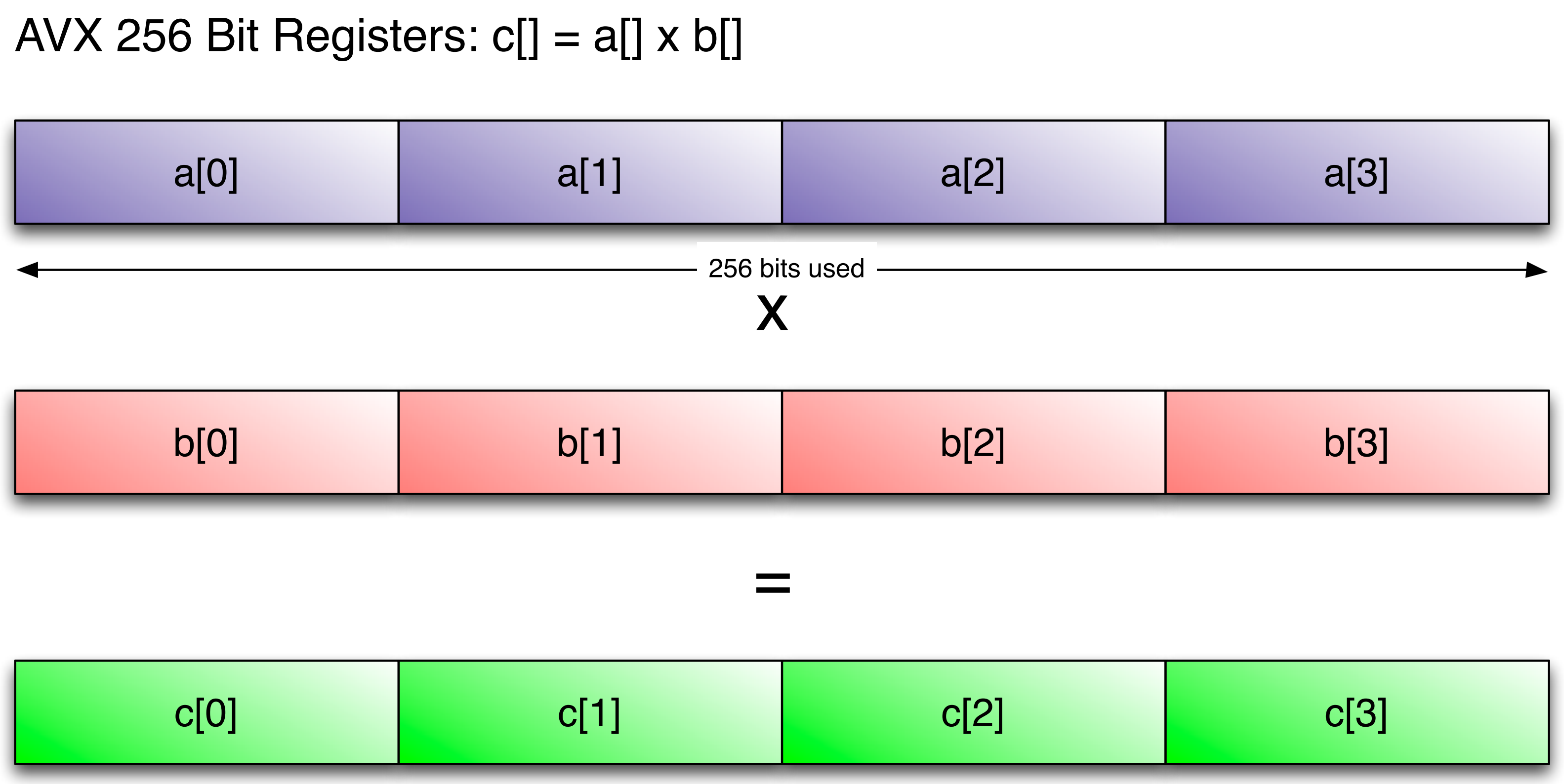


Easy incorporation of different technologies, such as GPGPUs
Events can go to different parts of a cluster
Mohammed Al-Turney, GSI

Static Check Tools

- Developed by ATLAS, CMS and CERN SFT for C++
  - Can help check for thread unsafe constructs
    - Use of mutable statics (even indirectly)
    - Violations of const — even some allowed by C++ standard (mutable, const_cast)
  - And can hint at lower level optimisations
    - Optimising divisions with reciprocal multiplications
- These checks are particularly important as multi-threaded coding mistakes can appear only in certain data race conditions, which are very hard to debug
- This is a good example of developments started in HEP that could be of much more general use to the open source community
  - There are others, such as much needed performance tools like igprof

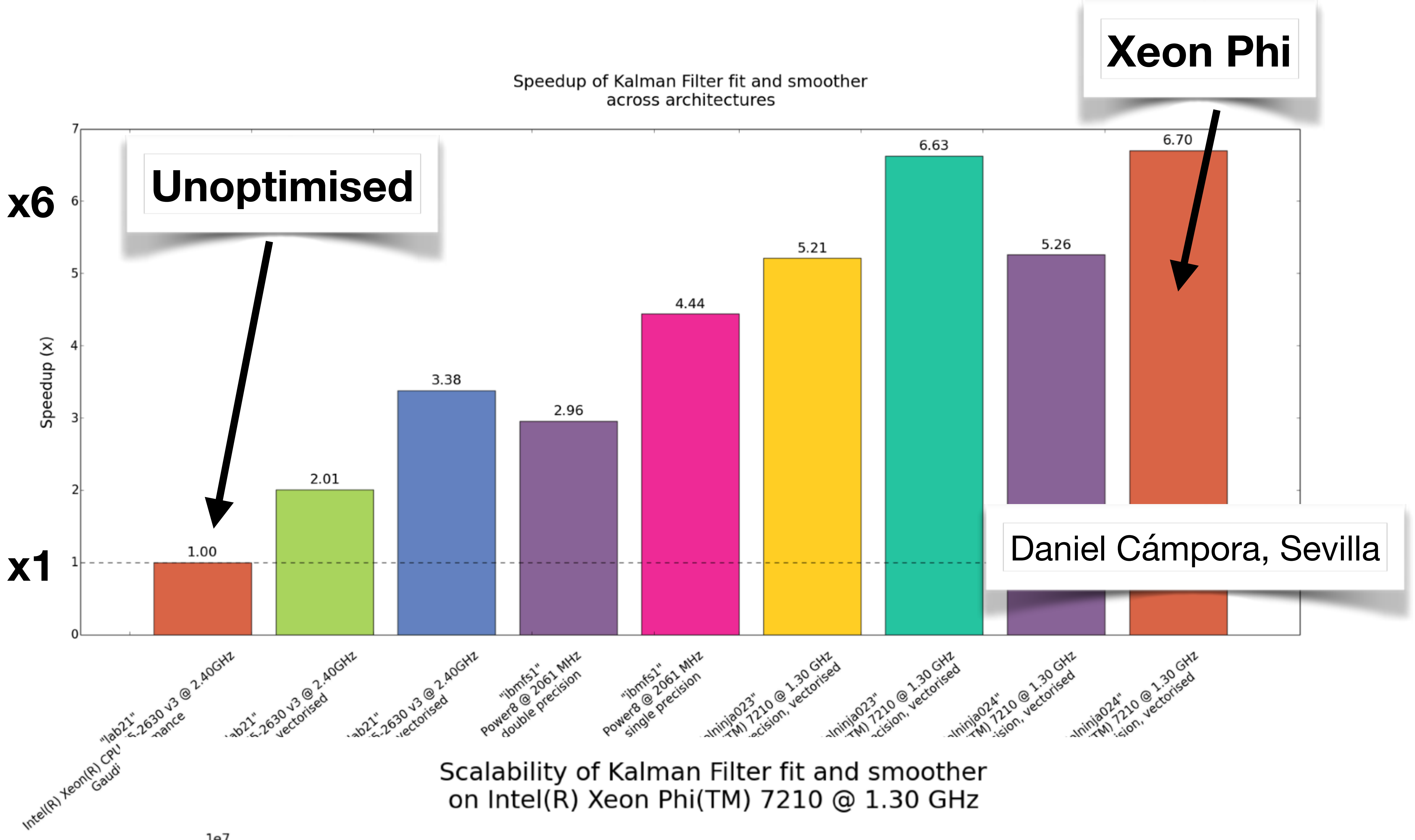
The Vector Challenge

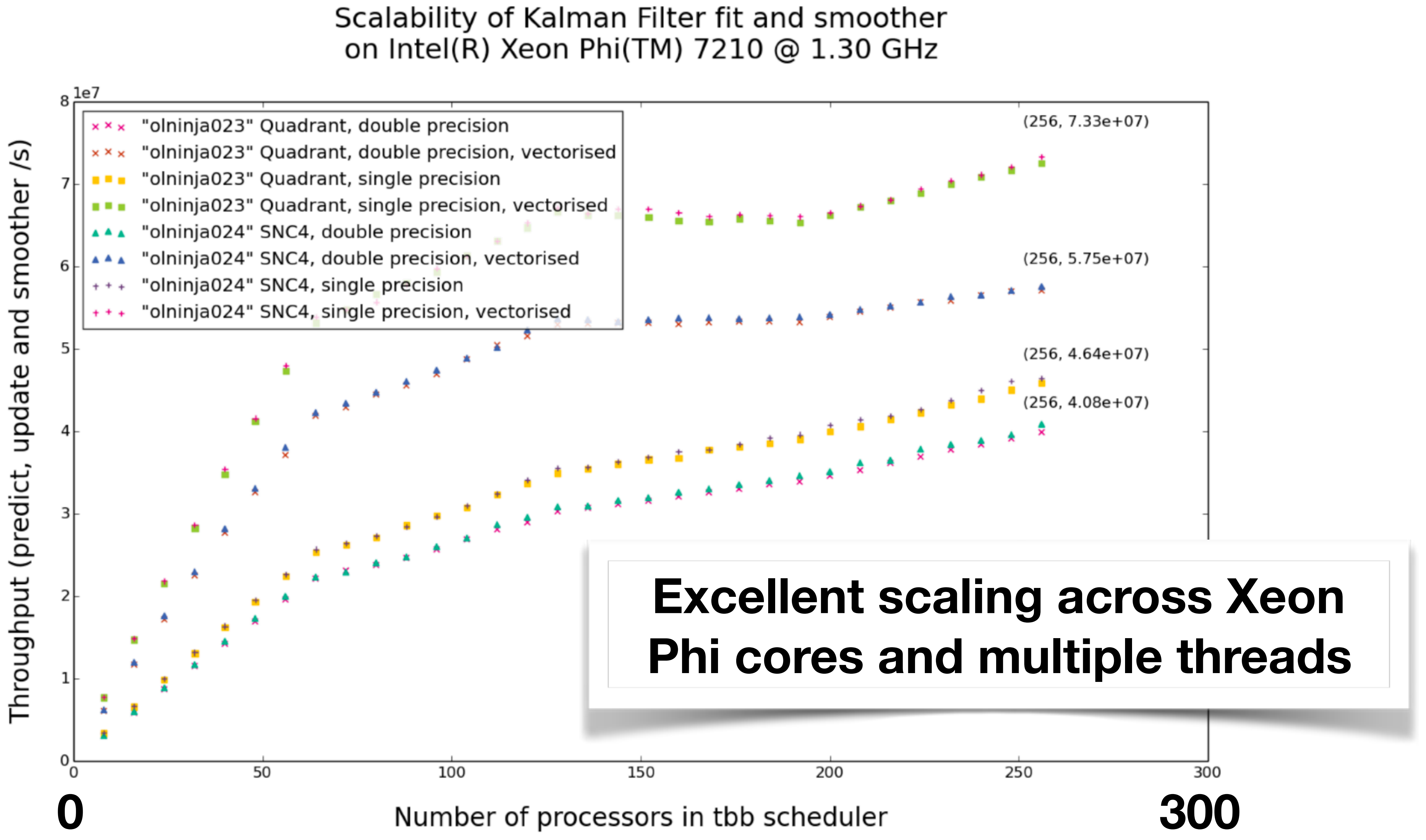


- CPU vector registers offer parallelised computing by performing the same operations on multiple pieces of data, so called Single Instruction Multiple Data (SIMD)
- GPUs offer similar capabilities, running kernels on many small cores
- Greatly enhanced FLOP rates, if they can be used effectively
    - Xeon Phi utilising 512bit Fused-Multiply-Add can process data at 3TFLOPS, ~400GB/s
        - If that rate could be achieved it would lead to a serious memory bandwidth problem
    - Which is why GPU calculations are often actually dominated by data transfer times
- In comparison to multi-core processing, where many independent tasks are run, vector processing is more associated with hot spots and computational kernel optimisation
    - This is a separate axis on which performance can be obtained

Kalman Filter

- Takes 70% of the processing time in LHCb Stage 1 HLT
    - Therefore an ideal candidate for gaining time, especially for LHC Run 3
- Fully vectorised implementation, precision configurable
    - All determined at compile time
- Optimised data layout (AOSOA) with alignment set by vector length
- x6.7 speed up is about as much as can be obtained due to memory bandwidth constraints

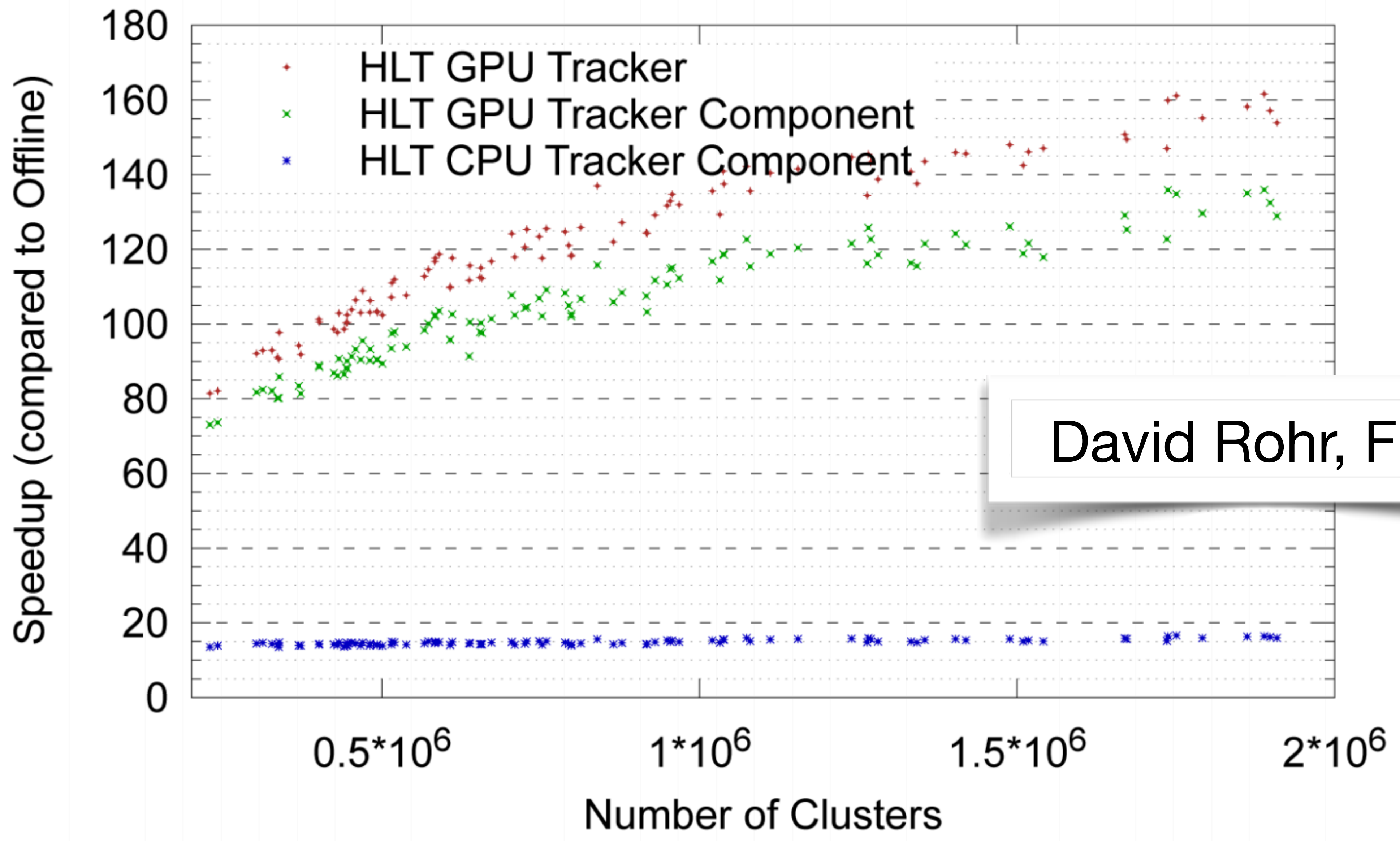




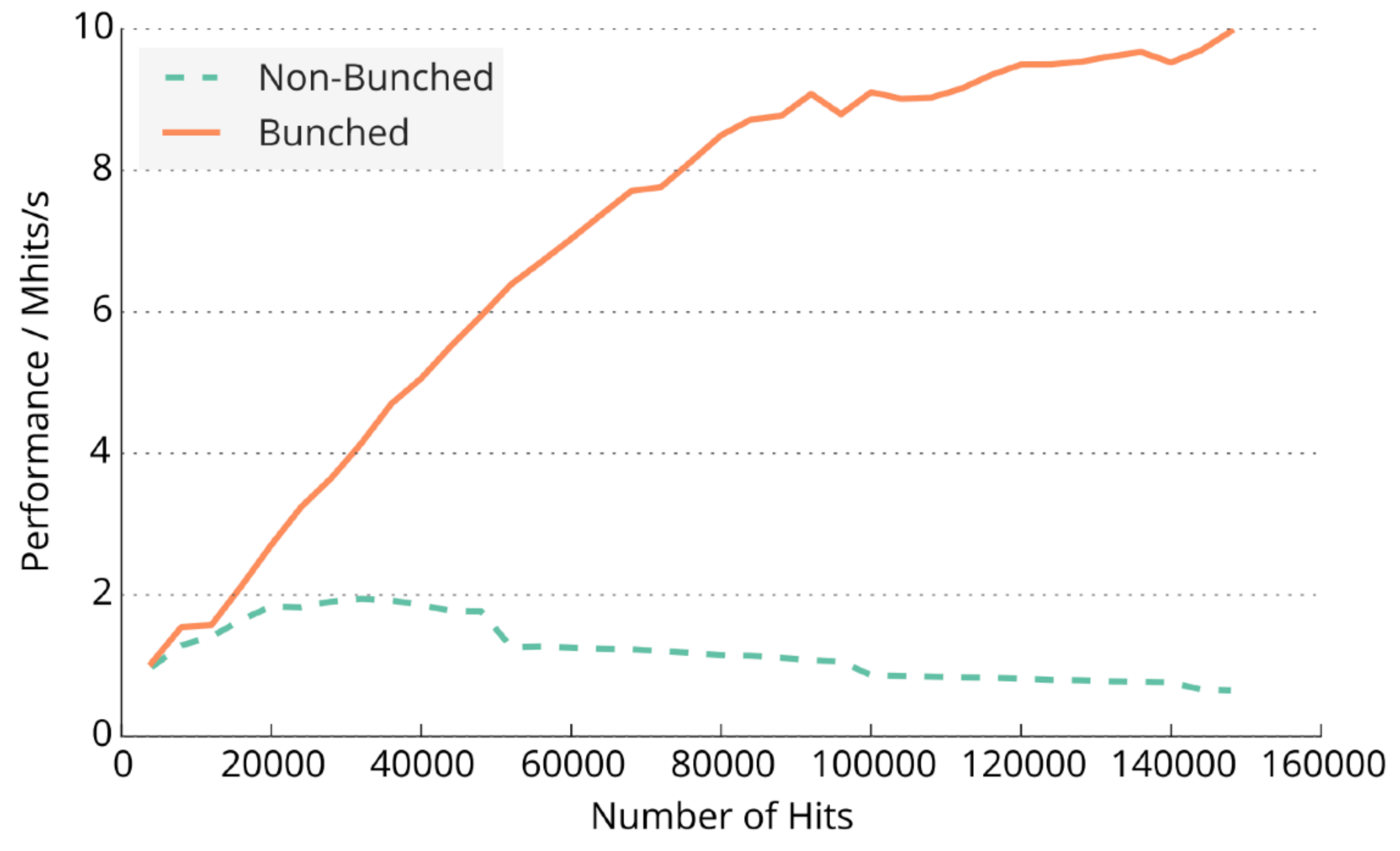
*Similar work here

GPUs

- Particularly in message passing frameworks it's easy to change devices
    - Allows for GPU tracking — offers huge throughput compared to similar code on CPUs
- Careful separation of algorithm (e.g., Hough Transform) from backend
    - Ensure physics logic is separated from device optimisation
- GPUs definitely suitable for some parts of of our processing chain
    - e.g., these online examples and training of deep neural networks
    - There are some other projects that deliver very high performance solutions to HEP problems using GPUs, like Hydra
- General (grid) deployment and use of GPUs remains unclear
    - Commodity GPUs seem to offer better price/performance for HEP problems



David Rohr, FIAS

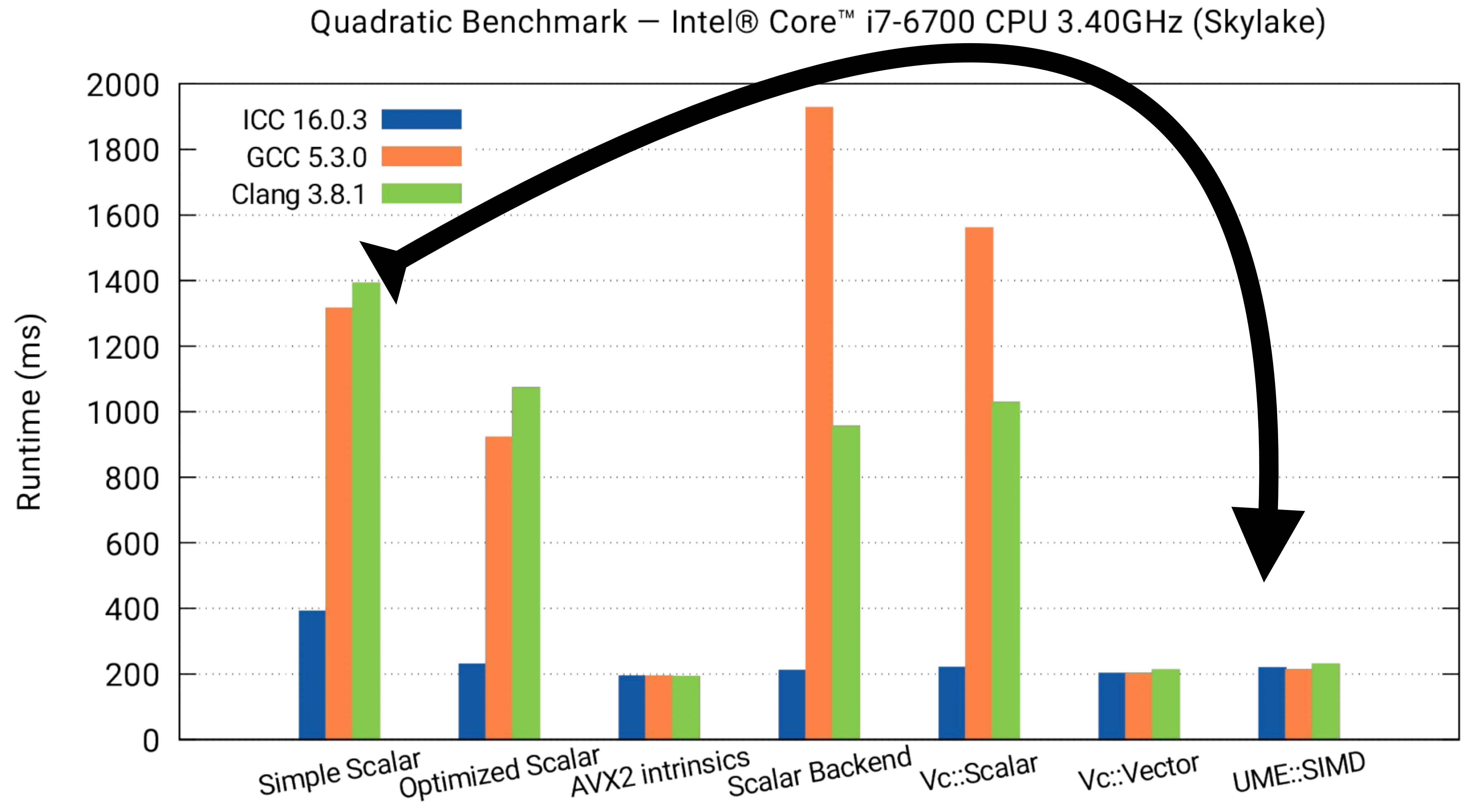


Benefits of collecting multiple bunch crossings for GPU throughput — amortise costs of data transfer
L Bianchi, Julich

VecCore

- C++ has no native type support for vector width grouping of data
  - Use of compiler intrinsics is verbose and not portable
  - Auto-vectorisation feature of compilers is patchy, fragile and often requires strong hinting
- Worthwhile investing here in lower level library code that can be used with optimised backends (Vc, UME::SMID, CUDA)
  - But still providing a useful level of abstraction to the developer
- VecCore provides uniform interface for vector types, arithmetic and logic, maths functions, scatter & gather
- Code becomes more portable and maintainable
- Adopted by ROOT and GeantV projects
  - But it's really independent of HEP

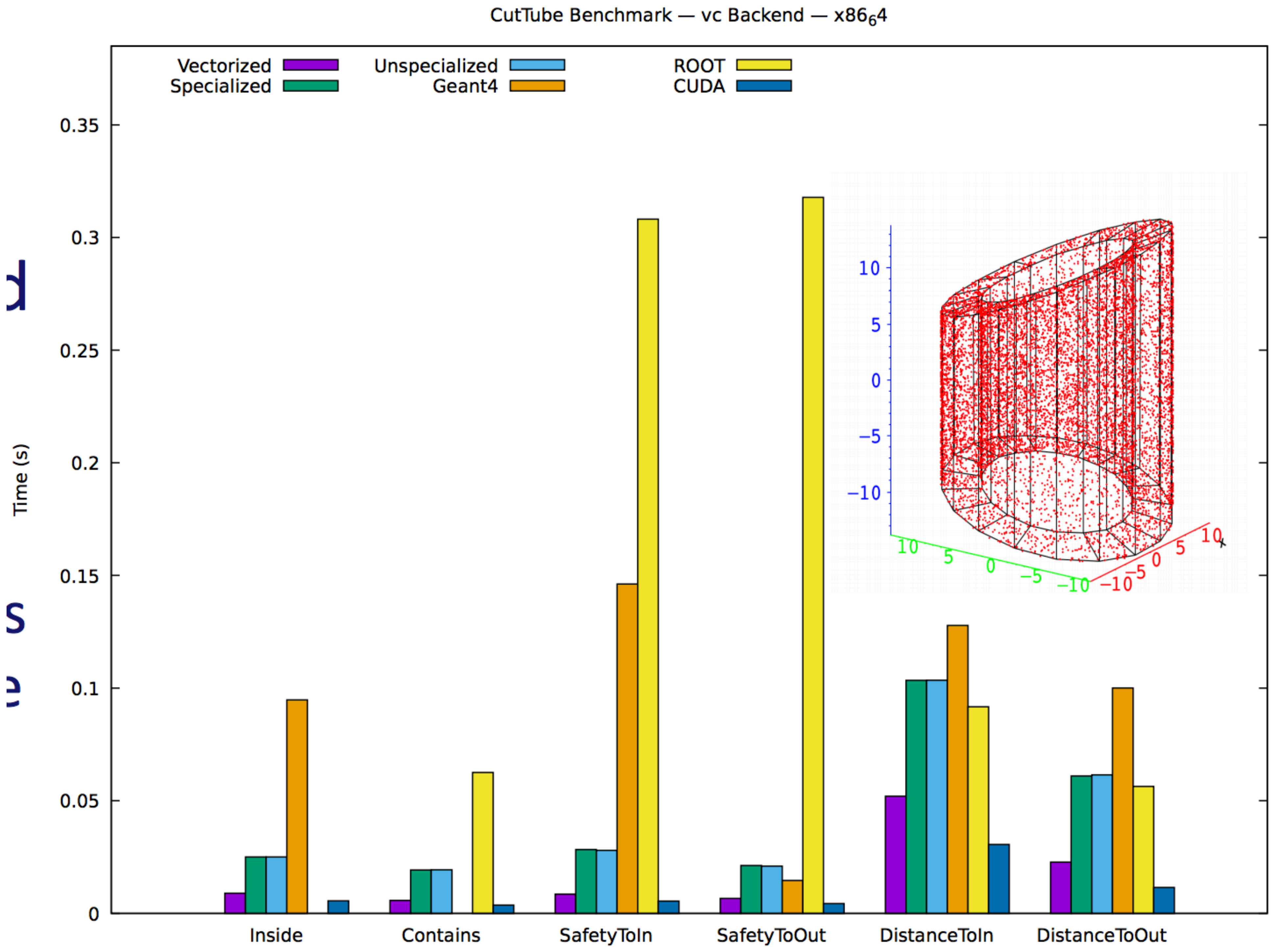
Significant speedup now uniform across compilers



Guilherme Amadio et al.

VecGeom

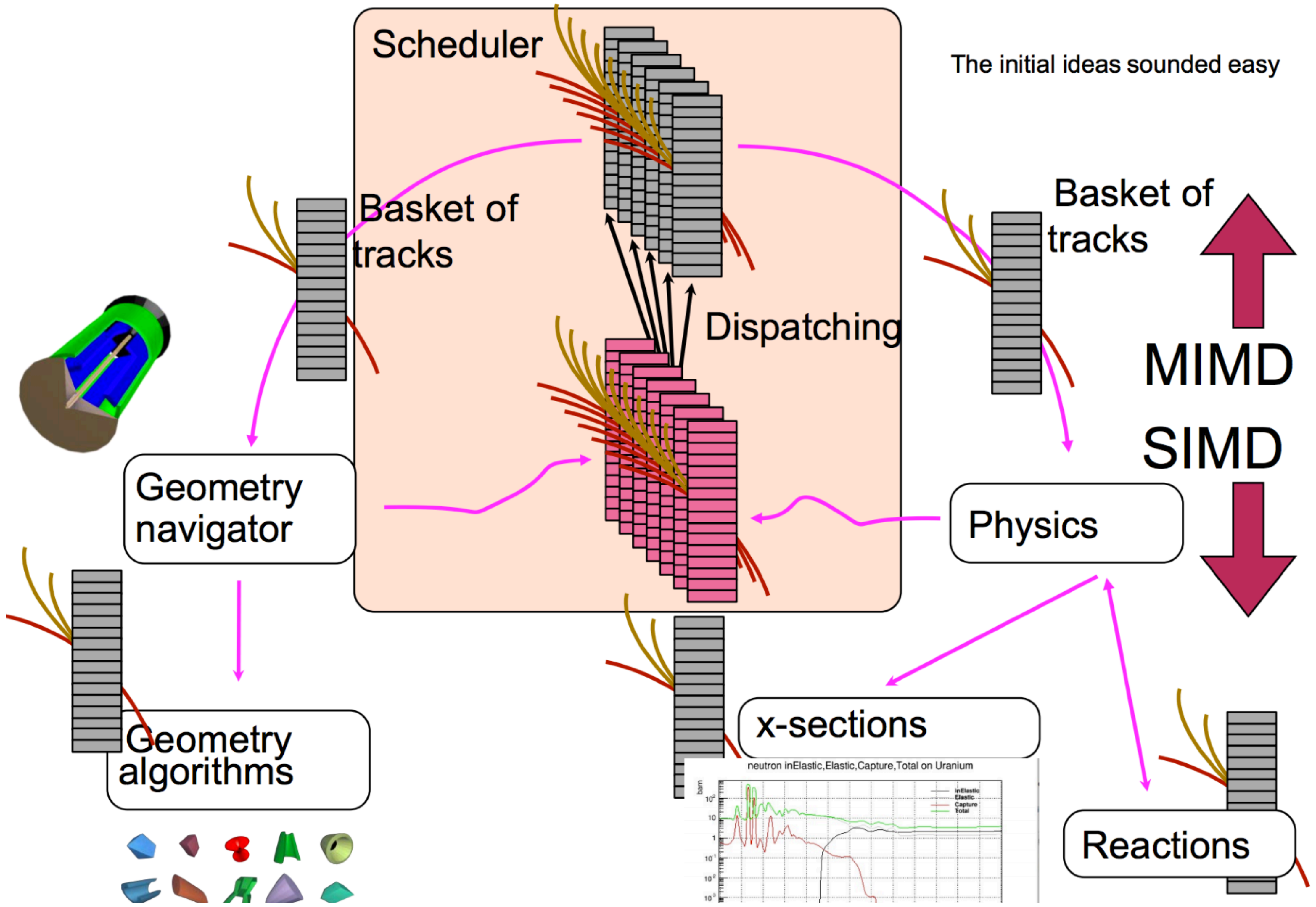
- Simulation takes up a huge part of of the LHC computing budget, mostly full simulation with Geant4
- Hard problem to execute effectively on modern hardware
  - Lots of particles in different places, different physics
- Developed vectorised geometry that would exploit SIMD for calculations
  - Has produced significant speed ups for core geometry calculations
- VecGeom code builds on top of VecCore for portable and maintainability geometry
  - ROOT, Geant4, GeantV

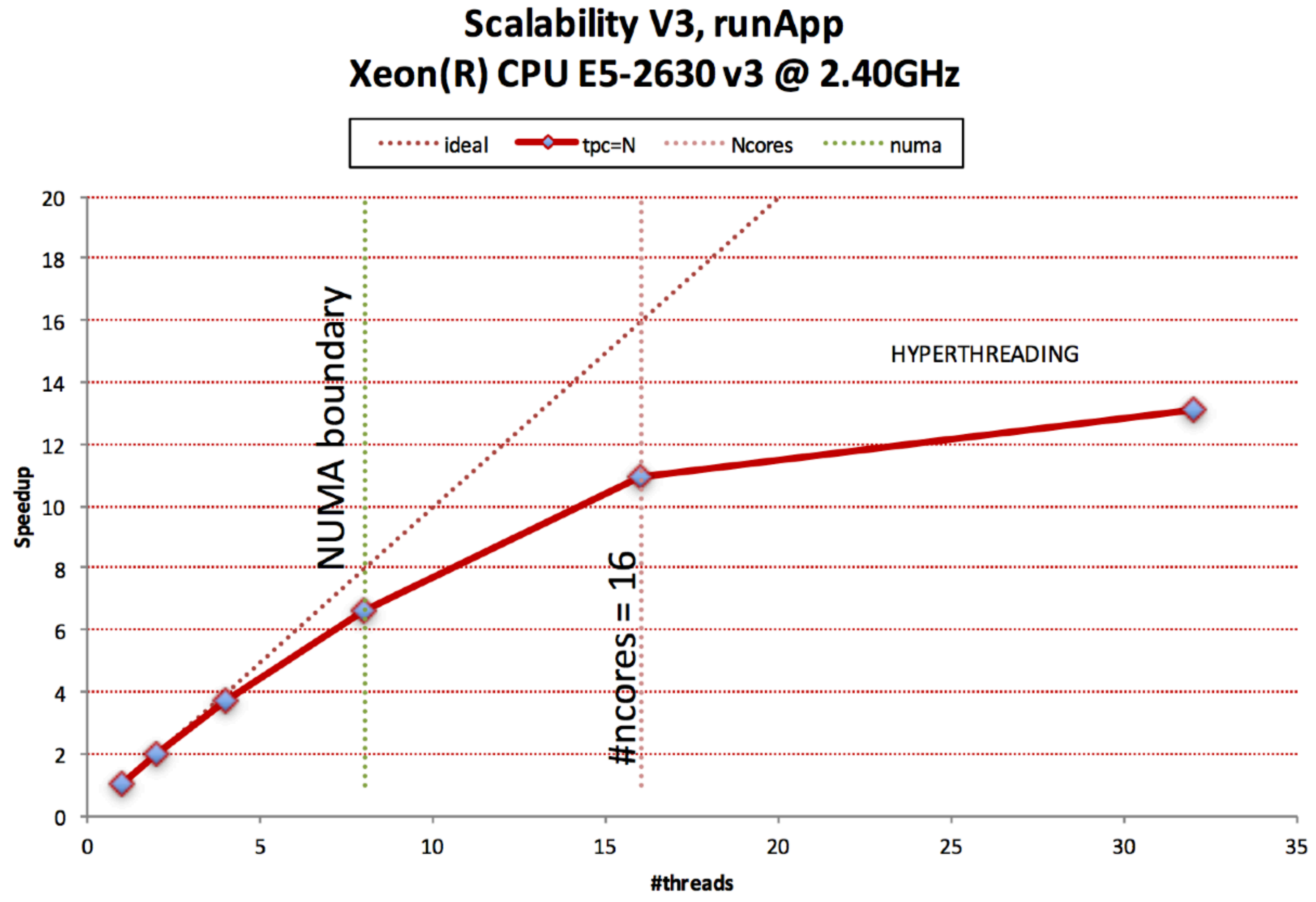


CutTube geometry speedup
Mihaela Gheata, CERN

GeantV

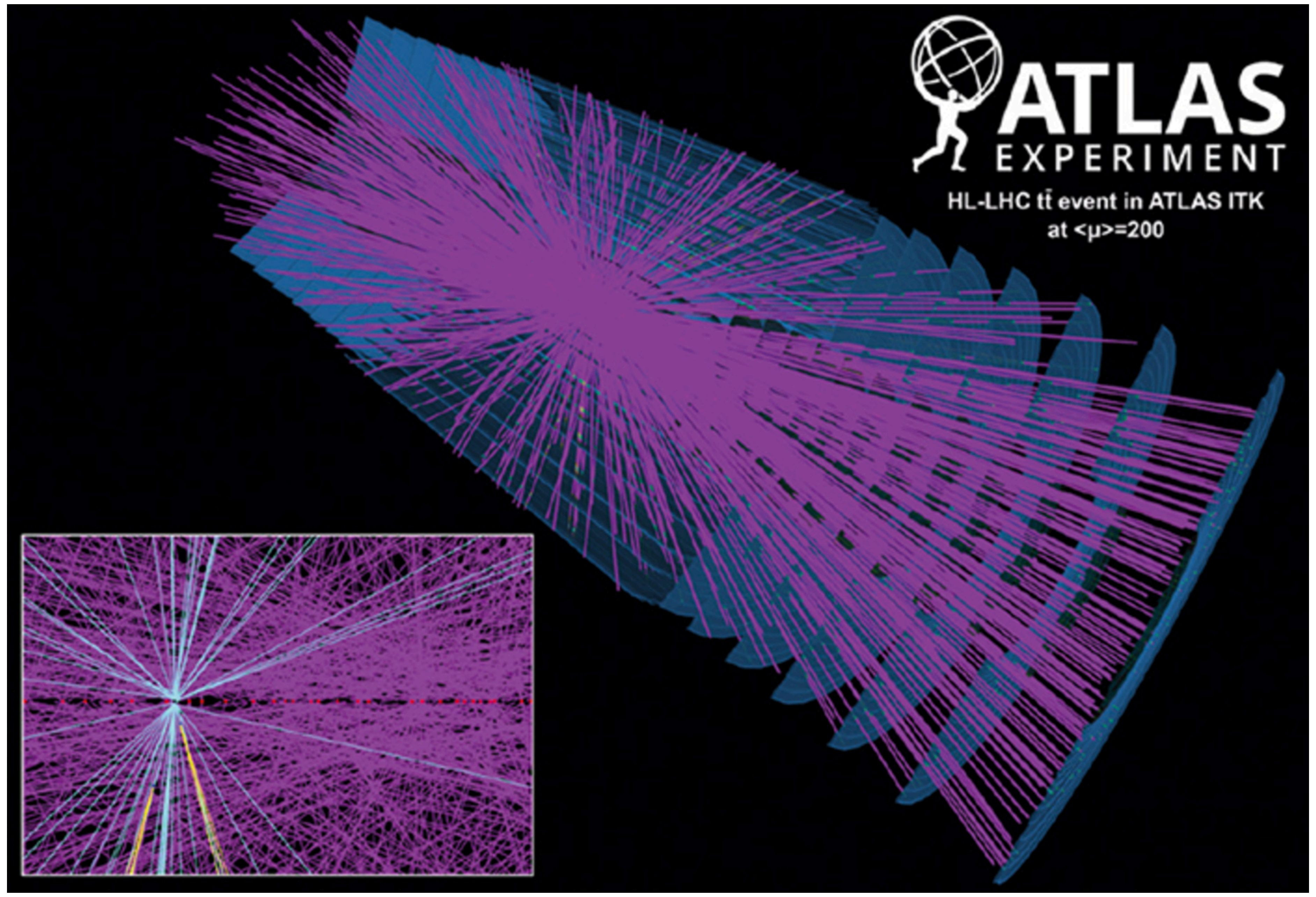
- GeantV project attempts to rewrite simulation targeting modern CPU and GPU architectures from the beginning
- Many particles of the same type and many particles within the same volumes
- Key idea is basketisation: sort particles by volume and type
  - Then transport these particles together
  - Aiming for x3-5 speedup over Geant4
- Early prototypes show a lot of promise
  - But it's not at all easy work and the sorting itself becomes a significant issue
  - Now being optimised to reduce these overheads, become aware of non-uniform memory access (NUMA)
- We have a long way to go and expert help is absolutely needed
- Significant collaboration via CERN OpenLab with Intel

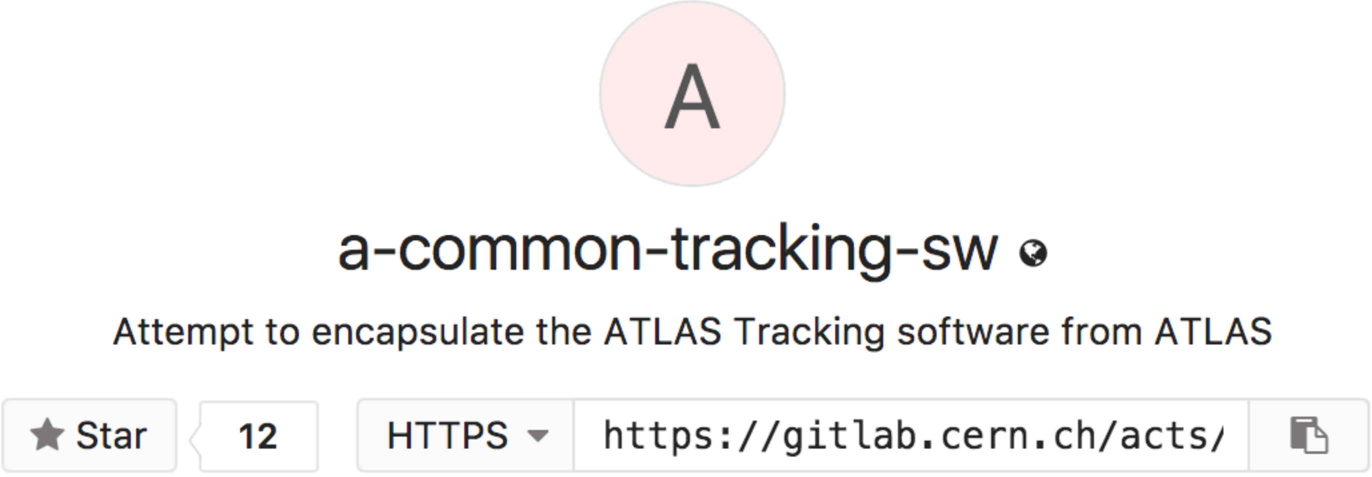




Reconstruction for LHC

- Architecture challenges aside, there is also a huge physics challenge for reconstruction at the LHC
    - Run 3: LHCb software trigger at 40MHz; ALICE Pb-Pb at 50kHz
    - Run 4: ATLAS and CMS pile up ~200 and 10kHz HLT output
- We need to modernise our reconstruction algorithms
    - Preserve physics performance
    - Run well on modern architectures
- There are many examples, e.g., ACTS (A Common Tracking Software) project
    - Factorise and improve ATLAS tracking code to be a generic toolkit
        - Other experiments can use it
        - Avoids mistakes we made in the past





Future Analysis

- Data analysis is at the other end of the pipe…
- Landscape becomes much more heterogeneous and ‘chaotic’
- However, it also becomes more amenable to generic tools
  - And our tools become more interesting to other communities
  - KPMG data analytics group have used ROOT in a couple of their projects (minimisation in particular is ‘best of breed’)
- Strengthening the python ecosystem is a key part of this activity
  - Python is extremely popular in data analytics
  - There is no other **dynamic binding system** between C++ and Python and other communities are very impressed that we have this
    - This is a unique contribution from our community, very interesting to other people
- Development of bridges (to other languages) and ferries (for data) is vital

Data Analytics

Spark SQL · cloudera IMPALA · Ibis · APACHE KUDU · HAWQ · APACHE DRILL · Dremel · Google BigQuery

- Data analytics is a big business now
- We are investigating different data formats and finding that ROOT does a great job for our data
    - Other formats have their merits, but HEP data is complex and highly structured
    - For long lived data there is also a lifetime problem — very fast changing landscape
- Radical analysis model would be to share data in a large cluster that can give access to slimmed columns of data of interest
    - Alternative to multiple rounds of skimming
    - It doesn't matter how much data there is in total, as long as you can get your data plotted in 'interactive' time (see this nice talk from the Astro domain at EPS)
    - Could we build such a store from current and future technology?
- Could use a declarative language to form an analysis

Muon object schema:

```
collection(record(
    pt = real(0, almost(inf)),
    eta = real,
    phi = real(-pi, pi)))
```

Example query:

```
muons.filter(mu => mu.pt > 5)
     .map(mu => mu.pt*sinh(mu.eta))
     .max
```

Return type:

```
union(null, real)
```

Femtocode prototype
Jim Pivarski, FNAL

ROOT and TDataFrame

- Parallel and Functional analysis coming to ROOT via TDataFrame
  - Inspired by pandas or spark, but working with ROOT data
- Way to express analysis as stating *what* to do, but to let the backend decide *how* to do it
  - Can parallelise much more easily and reorder calculations for maximum efficiency
  - Lazy evaluation helps optimise the workflow graph and will cache results
- Type safe using templates and JIT (just in time) compilation of expressions

```
TTreeReader data(tree);
TTreeReaderValue<A> x(data, "x");
TTreeReaderValue<B> y(data, "y");
TTreeReaderValue<C> z(data, "z");

while (reader.Next()) {
   if (IsGoodEvent(x, y, z))
      DoStuff(x, y, z);
}
```

```
ROOT::EnableImplicitMT();
TDataFrame data(tree, {"x","y","z"});


data.Filter(IsGoodEvent)
    .Foreach(DoStuff);
```

- Boiler plate removed
- No explicit event loop
- `DoStuff()` needs to be thread safe!

HEP Software Foundation

- HEP Software Foundation established in 2015 to address the software and computing challenges we face together
- Tackling the common problems in exploiting modern hardware and building the community across experiments
  - Providing concrete advice to developers (e.g., project templates)
  - Training and documentation (WikiToLearn, more to come next year)
  - Helping communication, dissemination and career advancement (new journal)
  - Help organise community meetings to key topics (GeantV, Analysis, Machine Learning, Tracking)
- Preparing a Community White Paper to be delivered shortly
  - 10 year roadmap of the developments we need to face the challenges of High Luminosity LHC, Intensity Frontier, …

Not all topics equally advanced — dependent on volunteer effort from busy people

HEP Software Foundation

HSF Community White Paper

- Community White Paper has active groups on many topics
  - Outcome should be a roadmap to HL-LHC with objectives for 1, 3 and 5 years:
    - 1 year prototypes and initial studies
    - 3 year studies to give input into LHC experiment TDRs
    - 5 year real projects to deliver software for high luminosity
  - Emphasise: catalyse common projects, promote commonality, attract new effort, set priorities
  - Links between the groups should be made for a coherent approach, e.g., training and machine learning are really cross cutting themes
- We have managed to involve non-HEP people in this process, but as well as inviting people in, we should go out
  - More involvement with the wider open source community, academia, industry
    - SciPy, StrangeLoop, Chaos Computer Club,

Reaching (further) Out

- Two workshops for HEP and Computer Science run as part of the US NSF Software Infrastructure for Sustained Innovation initiative
  - Trying to bridge the gap between HEP and Computer Science
    - HEP offers at-scale production-capable systems for a variety of CS research topics: scalable systems, data management, parallel processing
    - “industry scale, academic openness” — Developing collaborations requires HEP to recognize that CS research interests differ from HEP research interests.
    - Communication gaps exist: not only jargon, but also in simply formulating the problem to solve. Significant assumptions and/or incomplete problem descriptions. HEP people often describe the problem in terms of the solution we have at the moment.
    - We recognized that building common terminology and involving CS researchers earlier in the process will be beneficial.
- There is planning for more focused events in the future — this is the start of a process
  - Should meld with the CWP outcomes

Thanks to Pete Elmer for the summary points

Conclusions

- Software and computing are a critical part of High Energy Physics
- There is a huge challenge to be faced now in adapting to the recent and anticipated developments in commodity computing
  - In the context of large **increasing needs** for software and computing in the near future
- HEP is becoming more organised as a community that tries to solve these problems in common
  - HSF, Diana-HEP, AIDA2020 are good examples
  - That is at least the minimum that we need to prove to funding agencies that we can work efficaciously
- At the same time recognise that the scale of our problems is not unique, neither in academia nor in industry
  - We should strengthen our engagement with other communities and reach out, as much as draw in
  - We can contribute our experience and expertise if we find the right common language to frame problems and adapt our work to software ecosystems that already exist

Acknowledgements

Such a talk as this is impossible to prepare without the help of many colleagues in our field. Particular thanks are due to:

Nils Braun, Daniel Cámpora, Federico Carminati, Peter Elmer, Andrei Gheata, Benedikt Hegner, Chris Jones, Pere Mato, Marc Paterno, Gerhard Raven, Martin Ritter, Scott Snyder, Mohammad Al-Turany, Jiaheng Zou

And to all my colleagues in the HEP Software Foundation

HEP Software Foundation