Computing in the time of DUNE; HPC computing solutions for LArSoft

G. Cerati (FNAL)

LArSoft Workshop

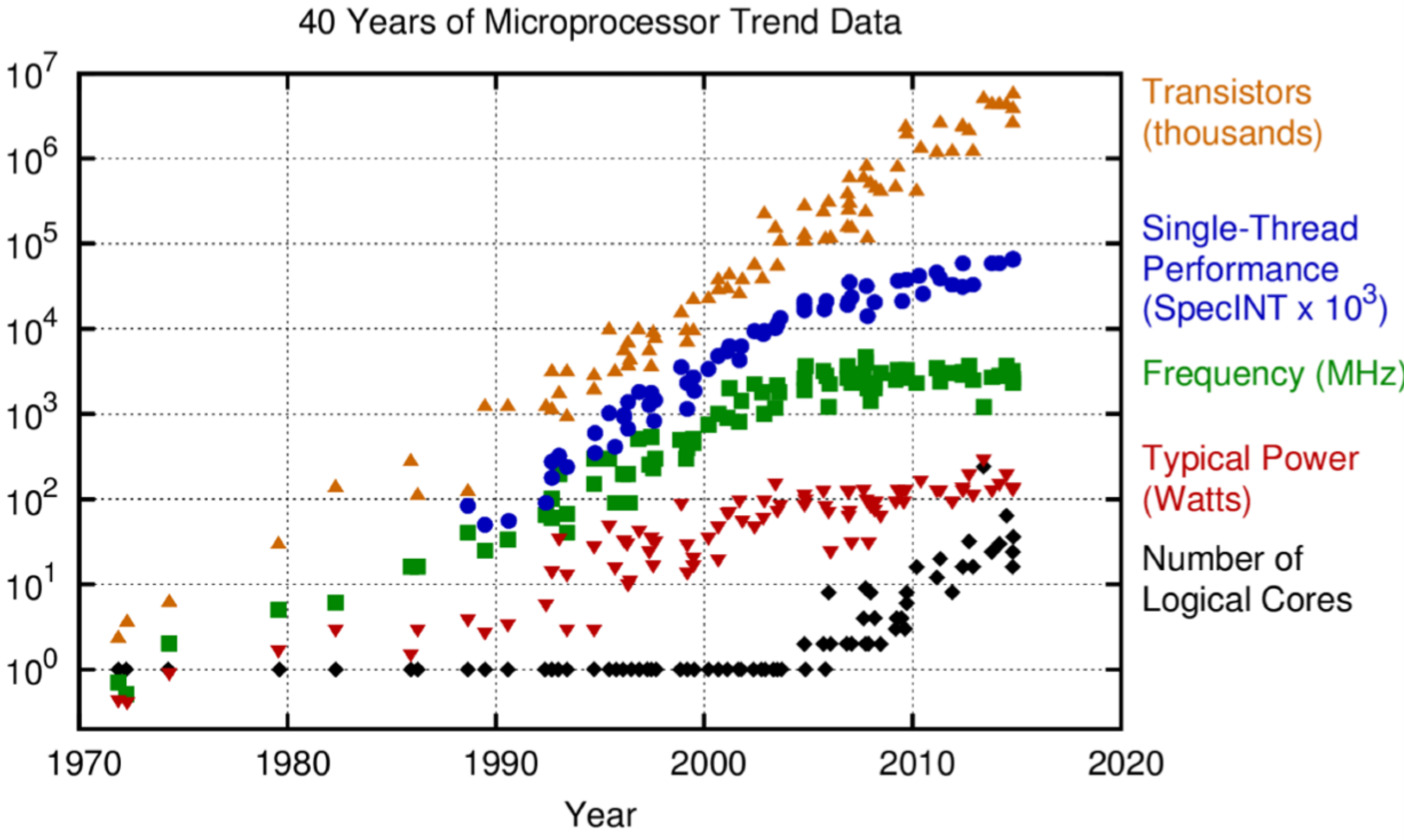
June 25, 2019

- Mostly ideas to work towards solutions!
- Technology is in rapid evolution…

Moore’s law

- We can no longer rely on **frequency** (CPU clock speed) to keep growing exponentially
  - nothing for free anymore
  - hit the **power wall**
- But **transistors** still keeping up to scaling
- Since 2005, most of the gains in single-thread performance come from **vector operations**
- But, number of **logical cores** is rapidly growing

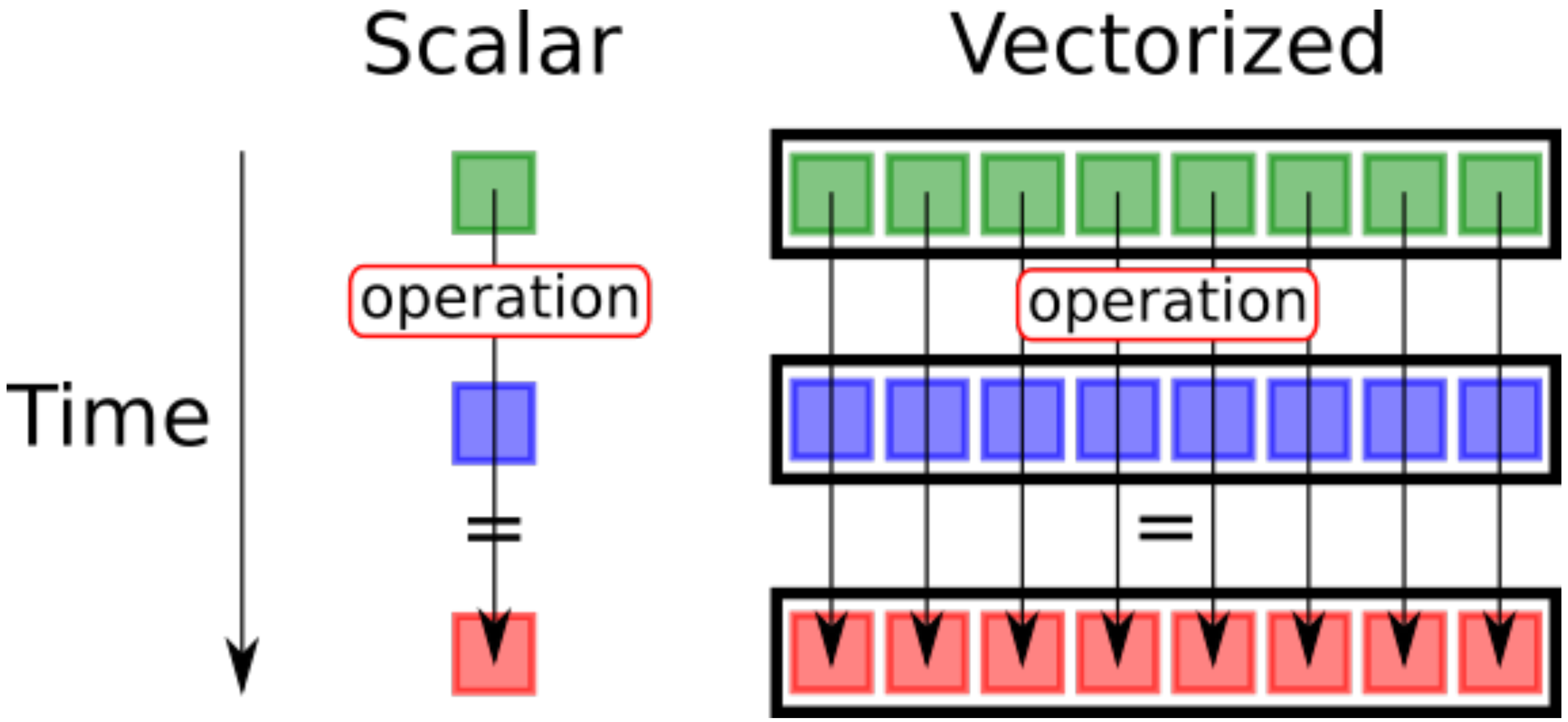
- Must exploit parallelization to avoid sacrificing on physics performance!



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Parallelization paradigms: data parallelism

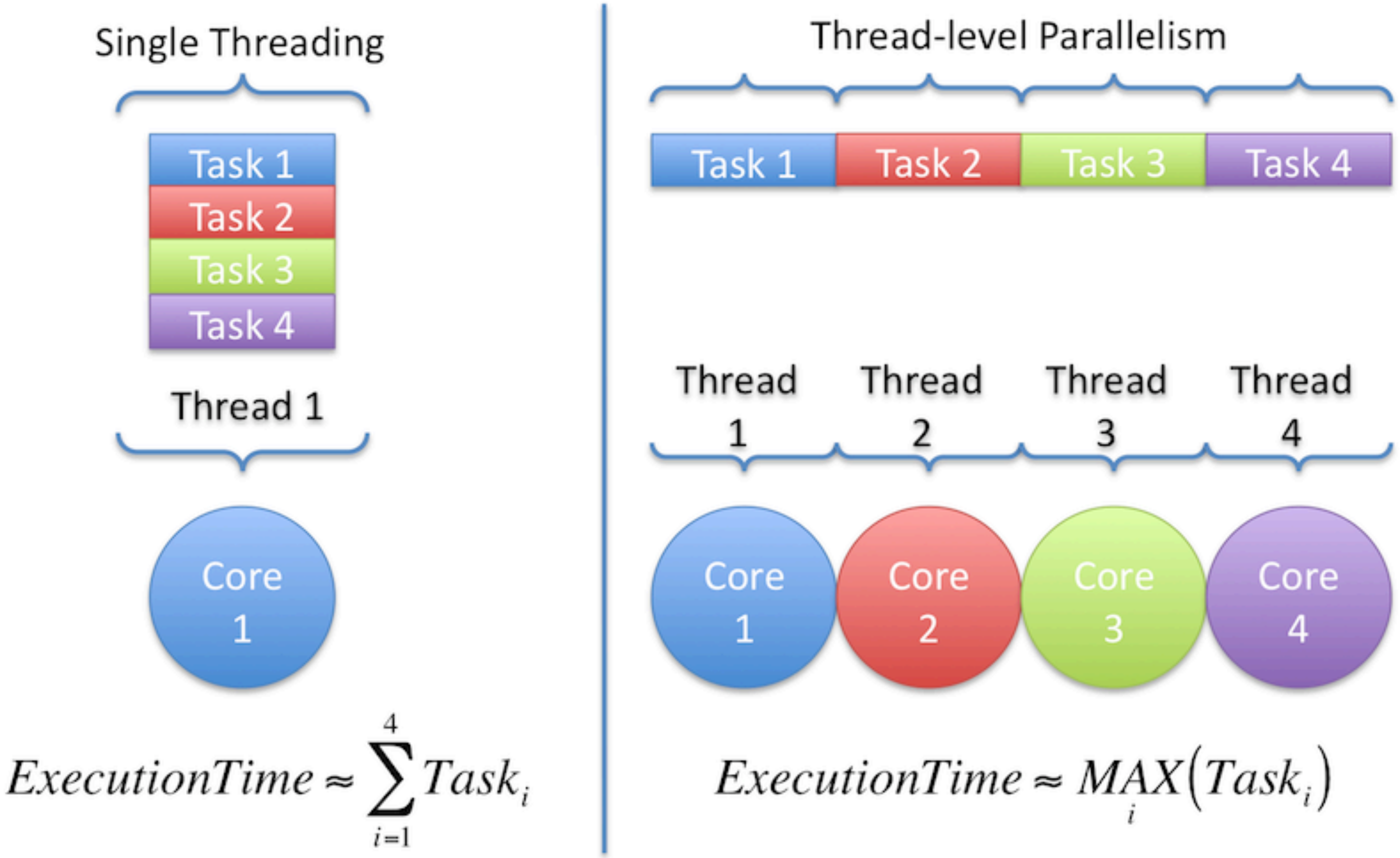
- Same Instruction Multiple Data model:
  - perform same operation in lock-step mode on an array of elements
- CPU vector units, GPU warps
  - AVX512 = 16 floats or 8 doubles
  - Warp = 32 threads
- Pros: speedup "for free"
  - except in case of turbo boost
- Cons: very difficult to achieve in large portions of the code
  - think how often you write 'if () {} else {}'



Parallelization paradigms: task parallelism

- Distribute independent tasks across different threads, threads across cores

- Pros:
  - typically easier to achieve than vectorization
  - also helps with reducing memory usage
- Cons:
  - cores may be busy with other processes
  - need to have enough work to keep all cores constantly busy and reduce overhead impact
  - need to cope with work imbalance
  - need to minimize sync and communication between threads

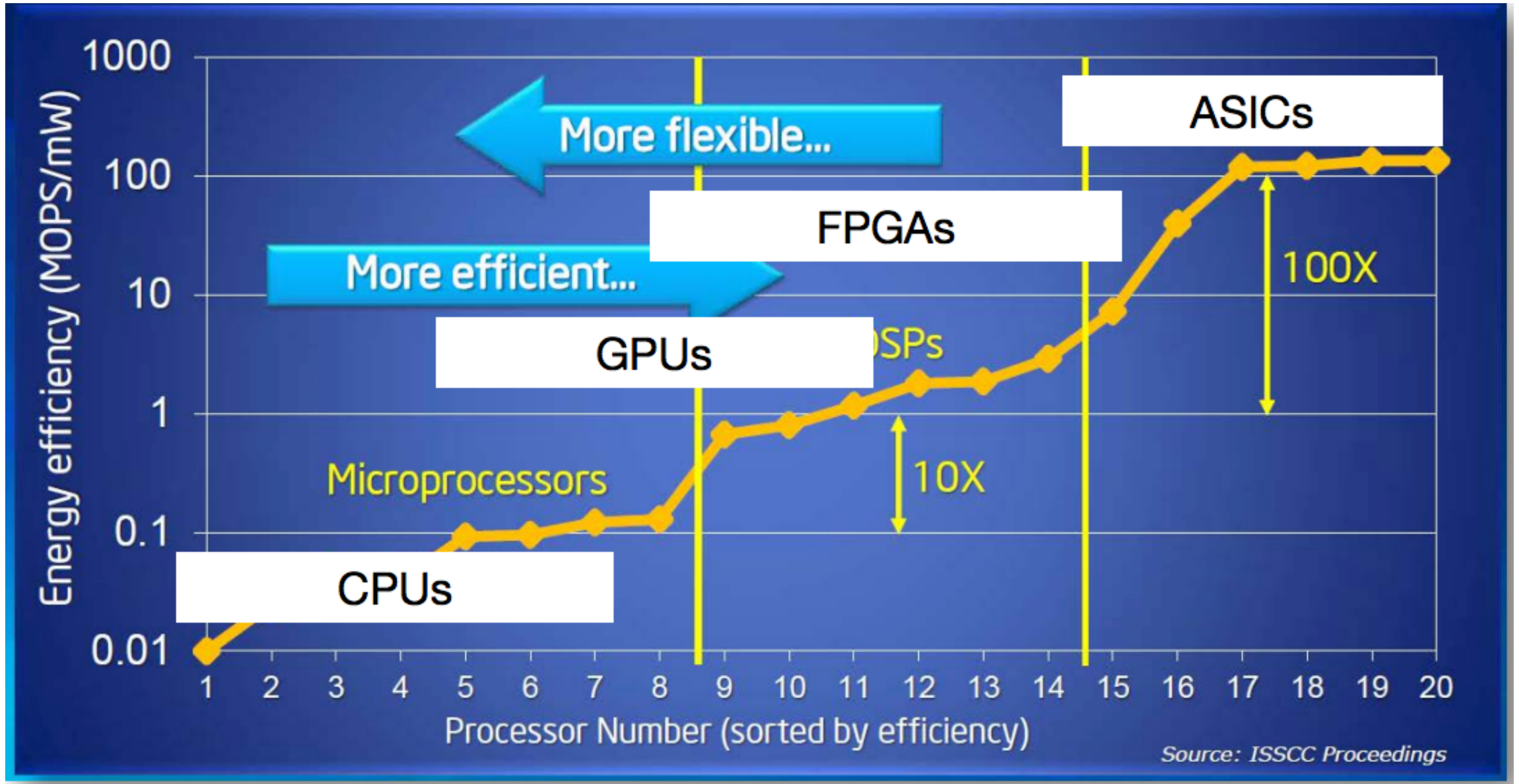


$$ExecutionTime \approx \sum_{i=1}^{4} Task_i$$

$$ExecutionTime \approx \underset{i}{MAX}(Task_i)$$

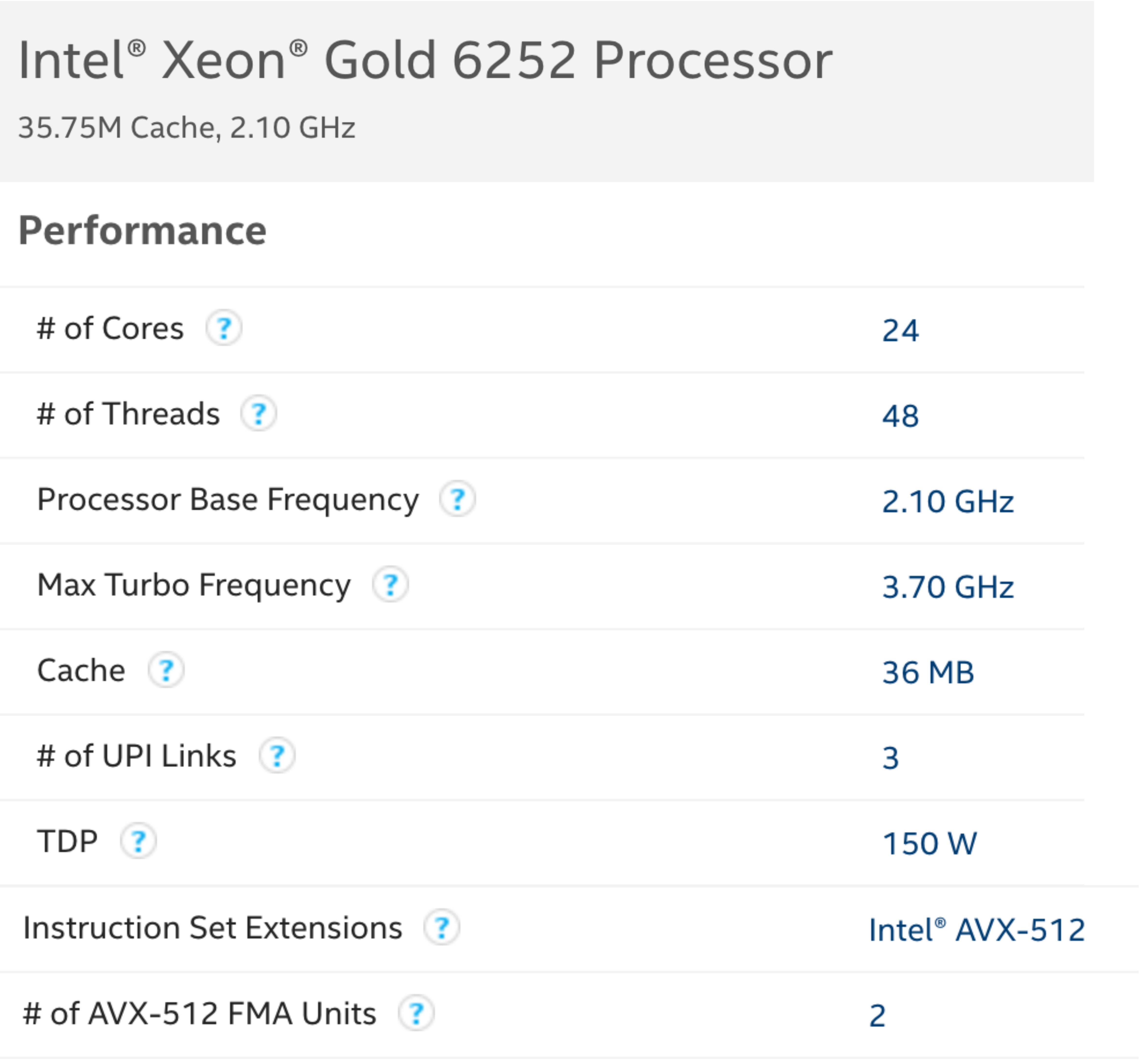
Emerging architectures

- It's all about power efficiency
- Heterogeneous systems
- Technology driven by Machine Learning applications



Source: Bob Broderson, Berkeley Wireless group

Intel Scalable Processors

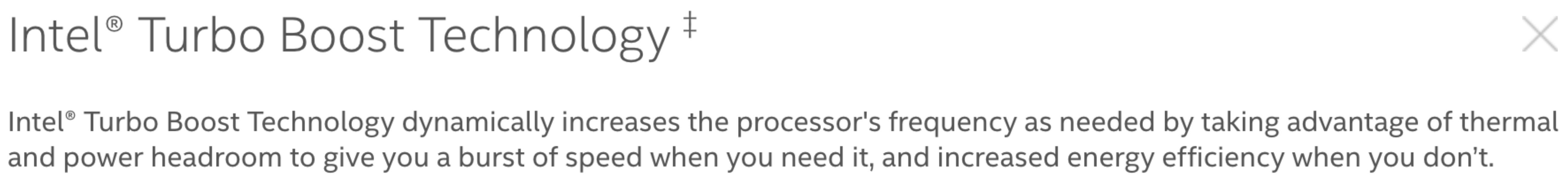


Intel® Xeon® Gold 6252 Processor

35.75M Cache, 2.10 GHz

Performance

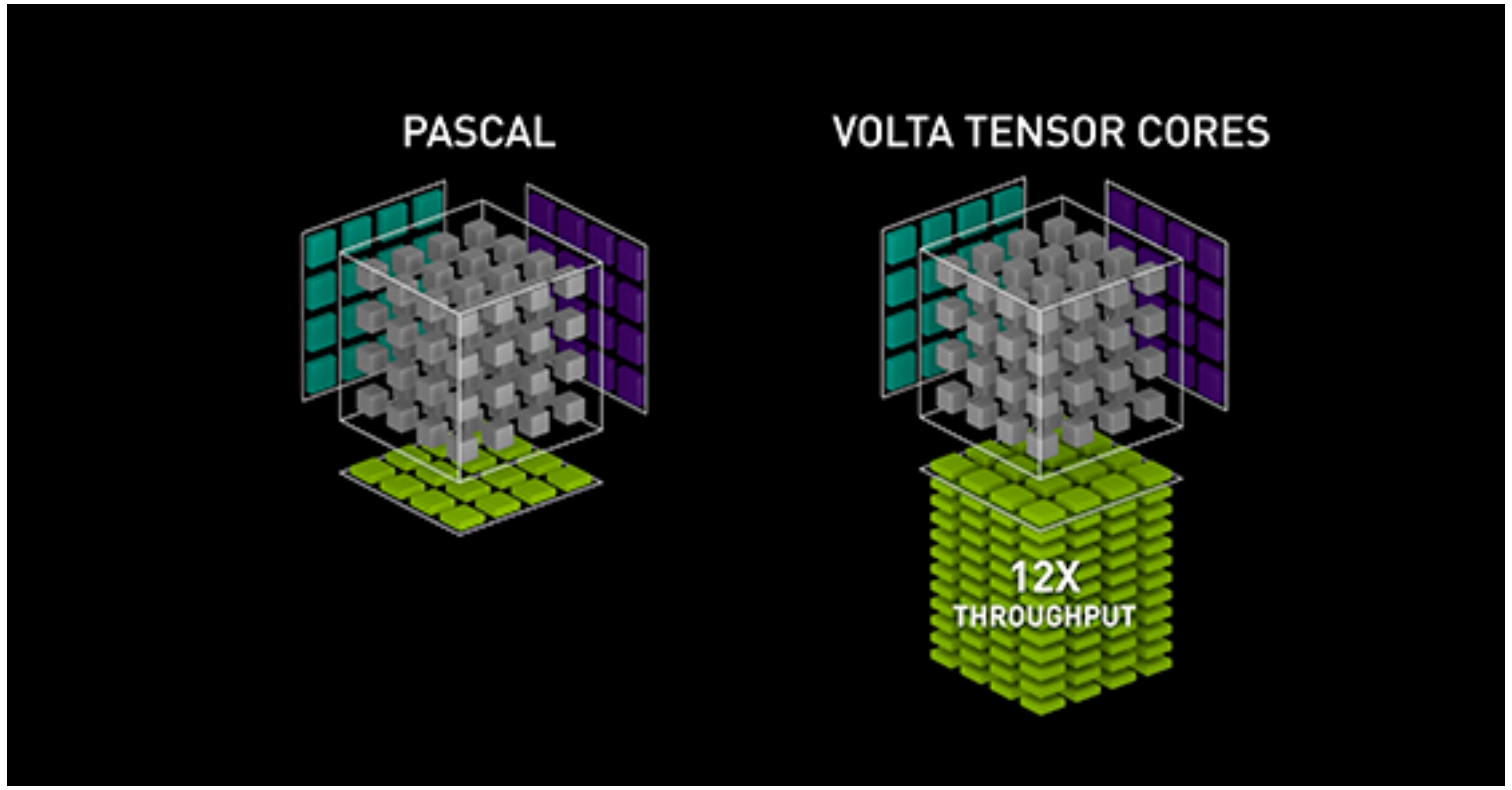
# of Cores	24
# of Threads	48
Processor Base Frequency	2.10 GHz
Max Turbo Frequency	3.70 GHz
Cache	36 MB
# of UPI Links	3
TDP	150 W
Instruction Set Extensions	Intel® AVX-512
# of AVX-512 FMA Units	2



Intel® Turbo Boost Technology ‡

Intel® Turbo Boost Technology dynamically increases the processor's frequency as needed by taking advantage of thermal and power headroom to give you a burst of speed when you need it, and increased energy efficiency when you don't.

NVIDIA Volta



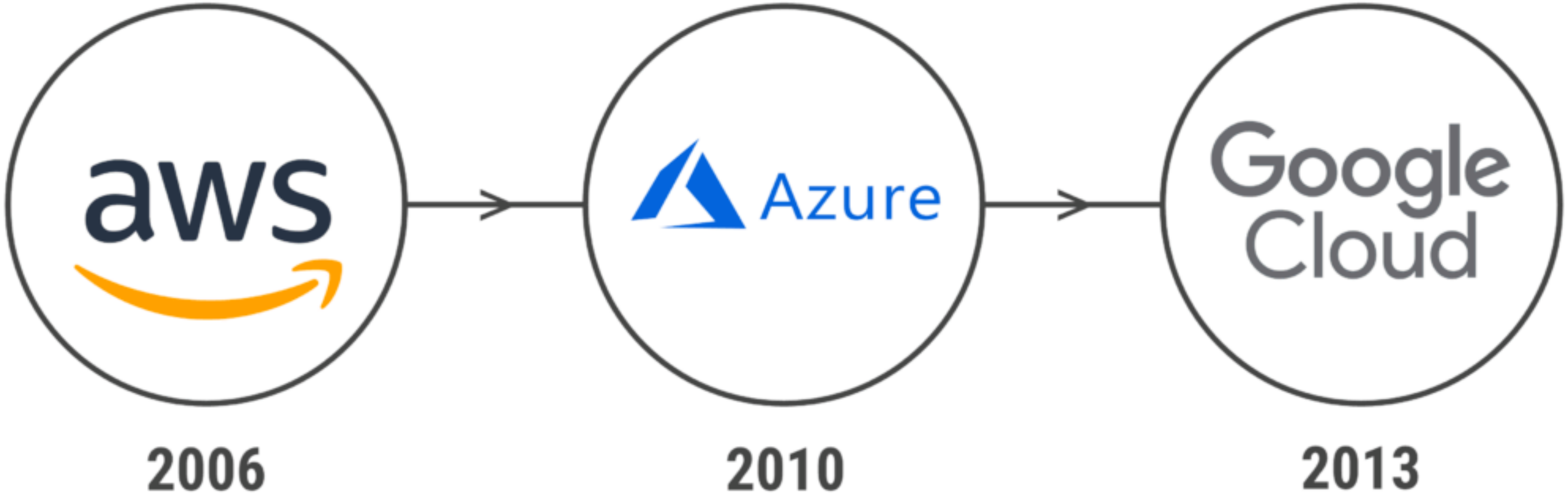
	Tesla V100 PCIe	Tesla V100 SXM2
GPU Architecture	**NVIDIA Volta**	
NVIDIA Tensor Cores	**640**	
NVIDIA CUDA® Cores	**5,120**	
Double-Precision Performance	**7 TFLOPS**	**7.8 TFLOPS**
Single-Precision Performance	**14 TFLOPS**	**15.7 TFLOPS**
Tensor Performance	**112 TFLOPS**	**125 TFLOPS**
GPU Memory	**32GB /16GB HBM2**	
Memory Bandwidth	**900GB/sec**	
ECC	**Yes**	
Interconnect Bandwidth	**32GB/sec**	**300GB/sec**
System Interface	**PCIe Gen3**	**NVIDIA NVLink**
Form Factor	**PCIe Full Height/Length**	**SXM2**
Max Power Comsumption	**250 W**	**300 W**
Thermal Solution	**Passive**	
Compute APIs	**CUDA, DirectCompute, OpenCL™, OpenACC**	

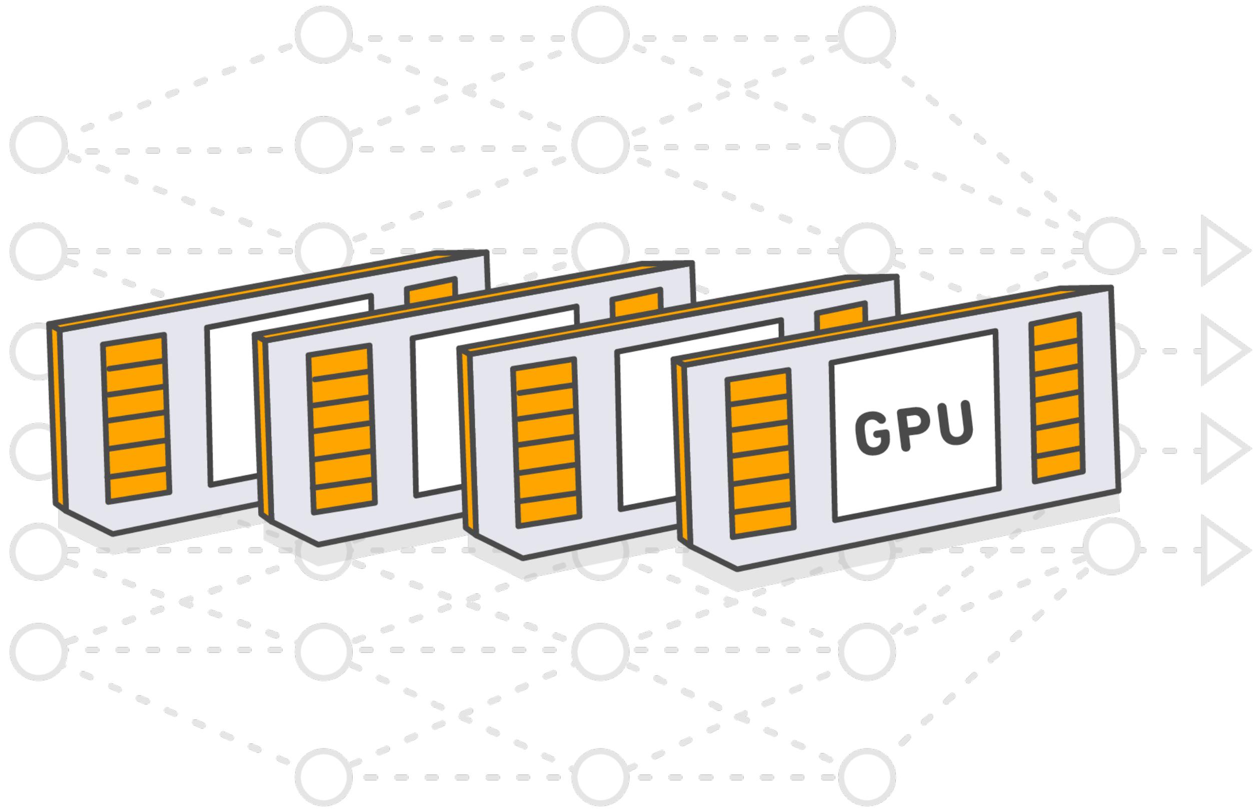
Next Generation DOE Supercomputers

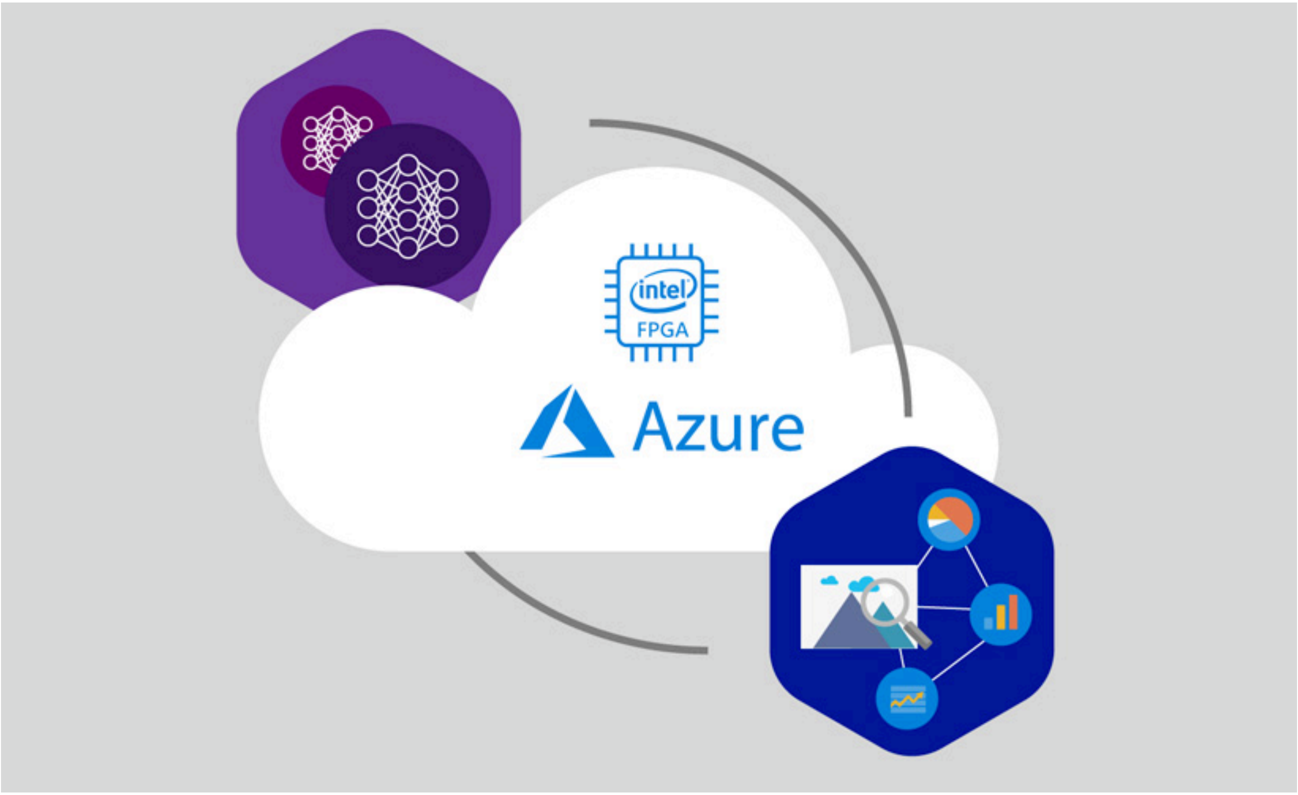
- Today - Summit@ORNL:
  - 200-Petaflops, Power9 + NVIDIA Tesla V100
- 2020 - Perlmutter@NERSC:
  - AMD EPYC CPUs + NVIDIA Tensor Core GPUs
  - *“LBNL and NVIDIA to work on PGI compilers to enable OpenMP applications to run on GPUs”*
  - Edison moved out already!
- 2021: Aurora@ANL
  - Intel Xeon SP CPUs + X^e GPUs
  - Exascale!
- 2021: Frontier@ORNL
  - AMD EPYC CPUs + AMD Radeon Instinct GPUs

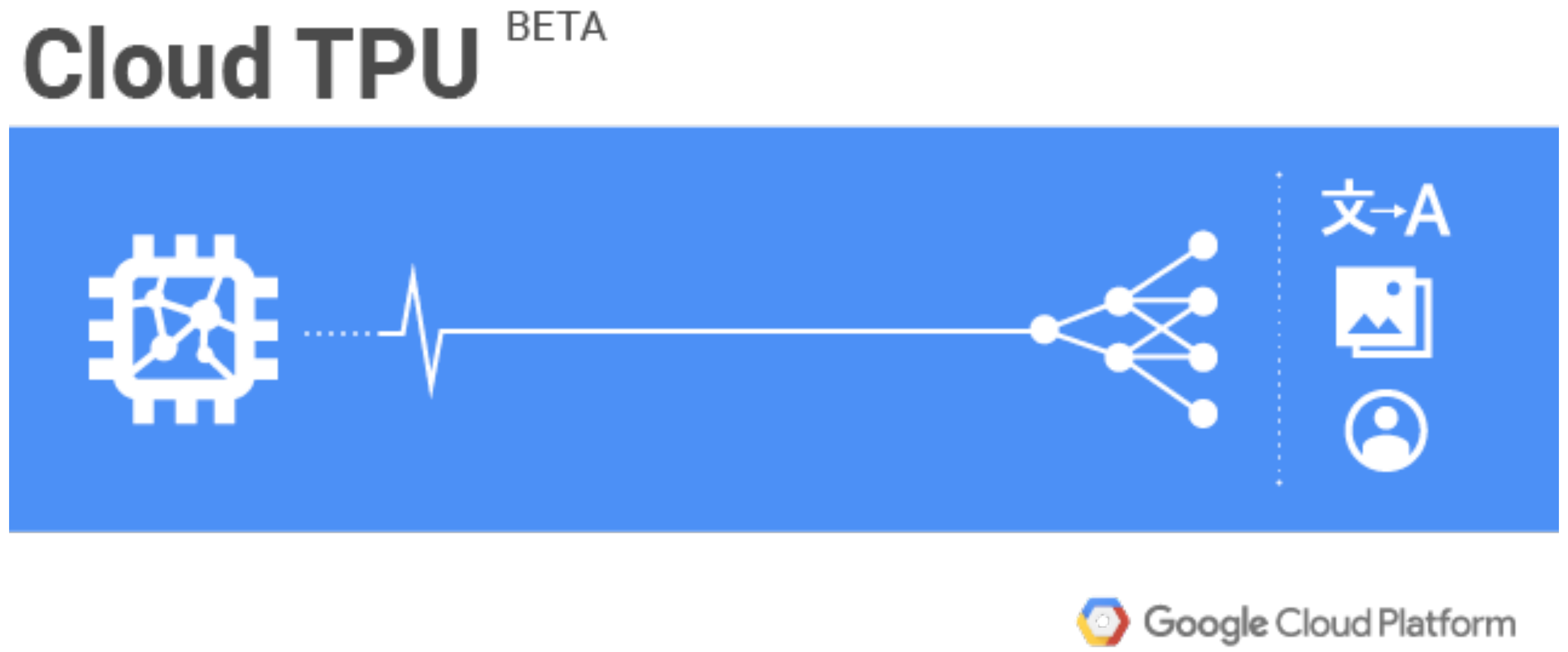
Commercial Clouds

- New architectures are also boosting the performance of commercial clouds







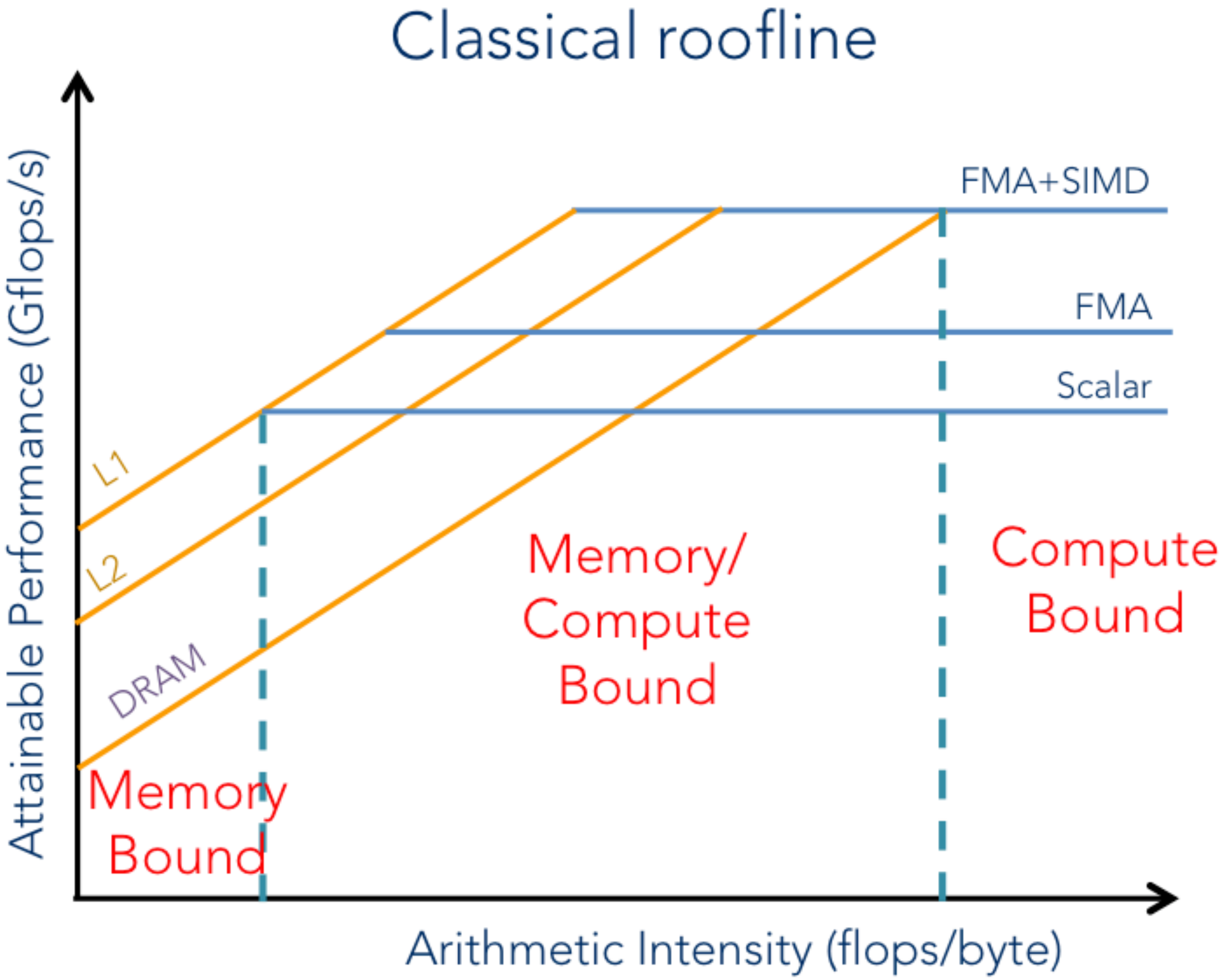


“Yay, let’s just run on those machines and get speedups”

“Yay, let’s just run on those machines and get speedups”

- The naïve approach is likely to lead to big disappointment: the code will hardly be faster than a good old CPU
- The reason is that in order to be efficient on those architectures the code needs to be able to exploit their features and overcome their limitations
- Features: SIMD-units, many cores, FMA
- Limitations: memory, offload, imbalance

- These can be visualized on the roofline plot
  - the typical HEP code is low arithmetic intensity…



Strategies to exploit modern architectures

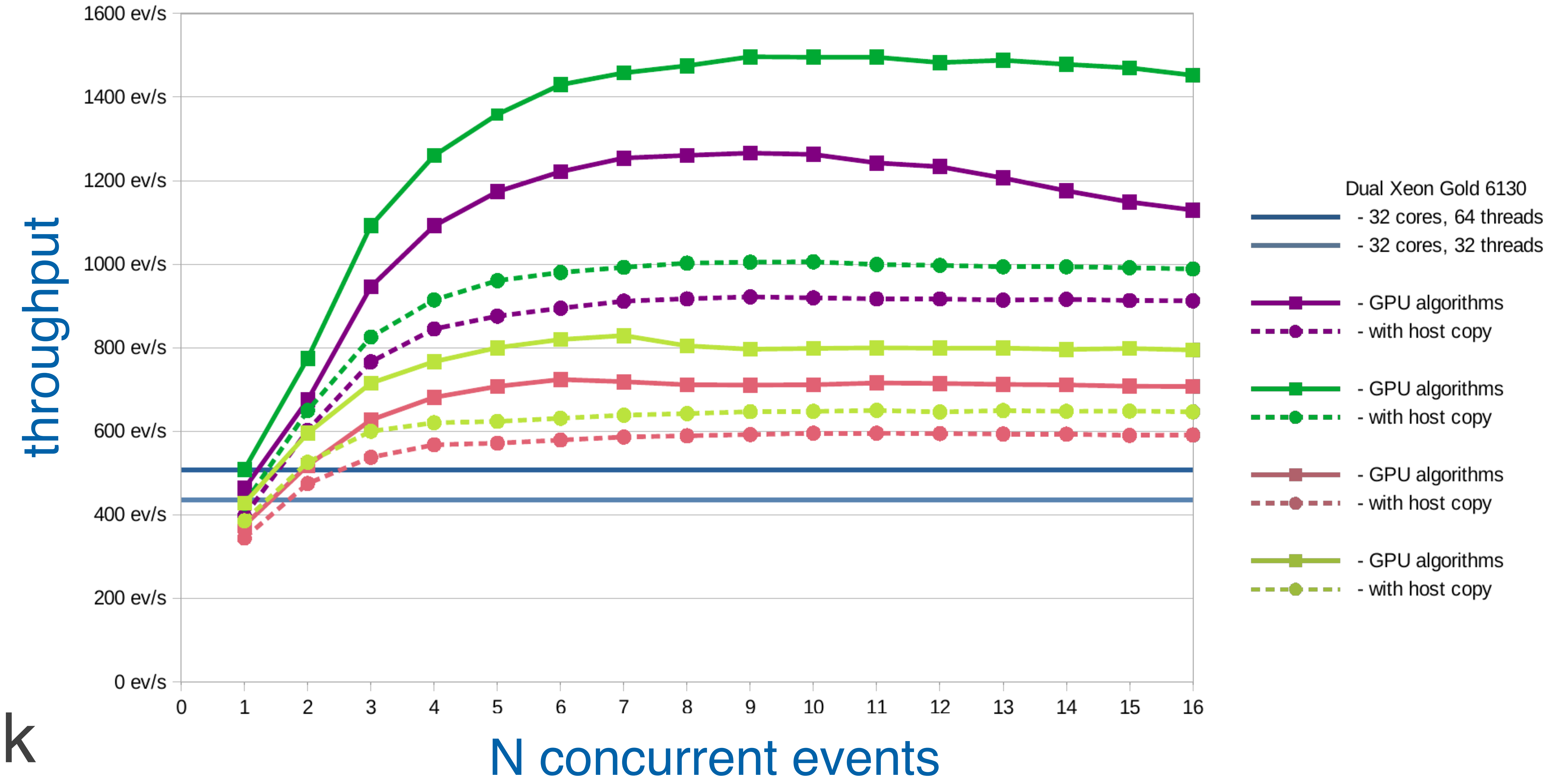
- Three models are being pursued:
  1. stick to good old algorithms, re-engineer them to run in parallel
  2. move to new, intrinsically parallel algorithms that can easily exploit architectures
  3. re-cast the problem in terms of ML, for which the new hardware is designed

- There's no right approach, each of them has its own pros and cons
  - my personal opinion!

- Let's look at some lessons learned and emerging technologies that can potentially help us with this effort

Some lessons learned from LHC friends

- Work started earlier on the LHC experiments to modernize their software
- Still in R&D phase, but we can profit of some of the lessons learned so far
- A few examples:
  - hard to optimize a large piece of code: better to start small then scale up
  - writing code for parallel architectures often leads to better code, usually more performant even when not run in parallel
    - better memory management
    - better data structures
    - optimized calculations
  - HEP data from a single event is not enough to fill resources
    - need to process multiple events concurrently, especially on GPUs
  - Data format conversions can be bottleneck

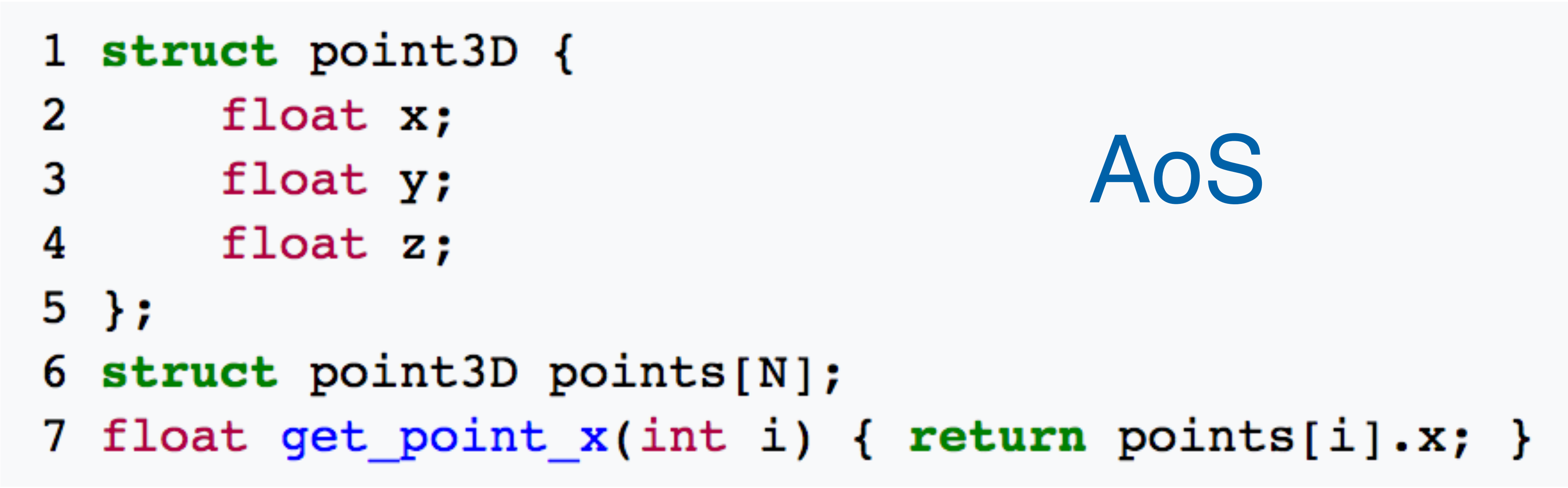


CMS Patatrack project
https://patatrack.web.cern.ch/patatrack/

Data structures: AoS, SoA, AoSoA?

- Efficient representation of the data is a key to exploit modern architectures

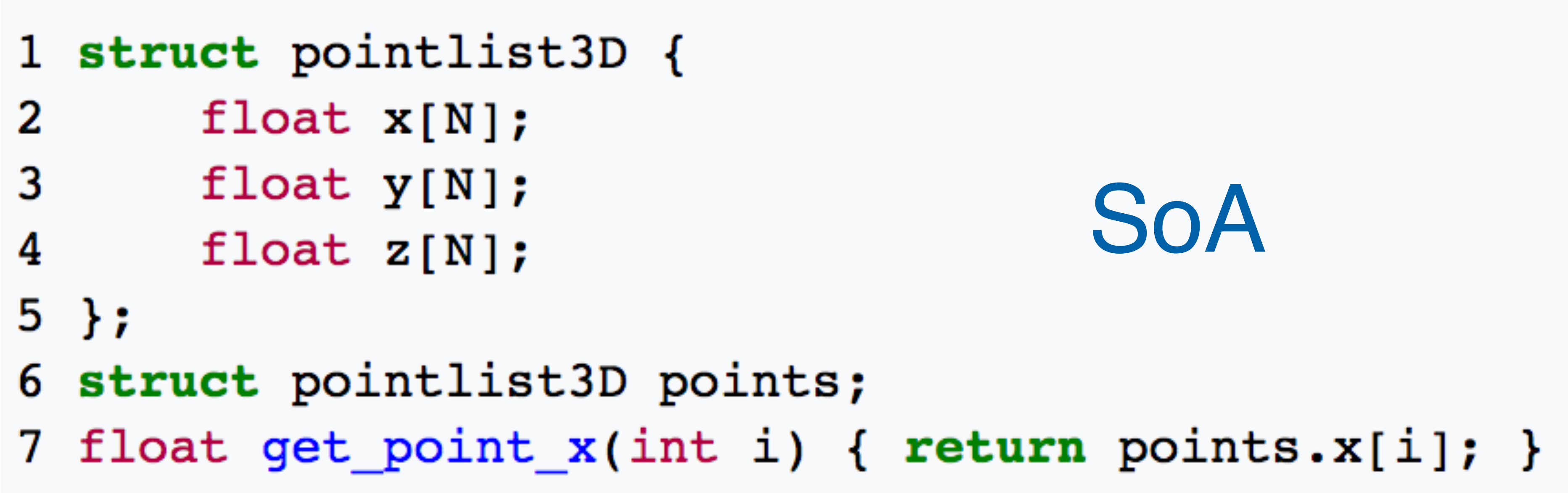
- Array of Structures:
  - this is how we typically store the data
  - and also how my serial brain thinks
- Structure of Arrays:
  - more efficient access for SIMD operations, load contiguous data into registers
- Array of Structures of Arrays
  - one extra step for efficient SIMD operations
  - e.g. Matriplex from CMS R&D project



```
struct point3D {
    float x;
    float y;
    float z;
};
struct point3D points[N];
float get_point_x(int i) { return points[i].x; }
```

AoS

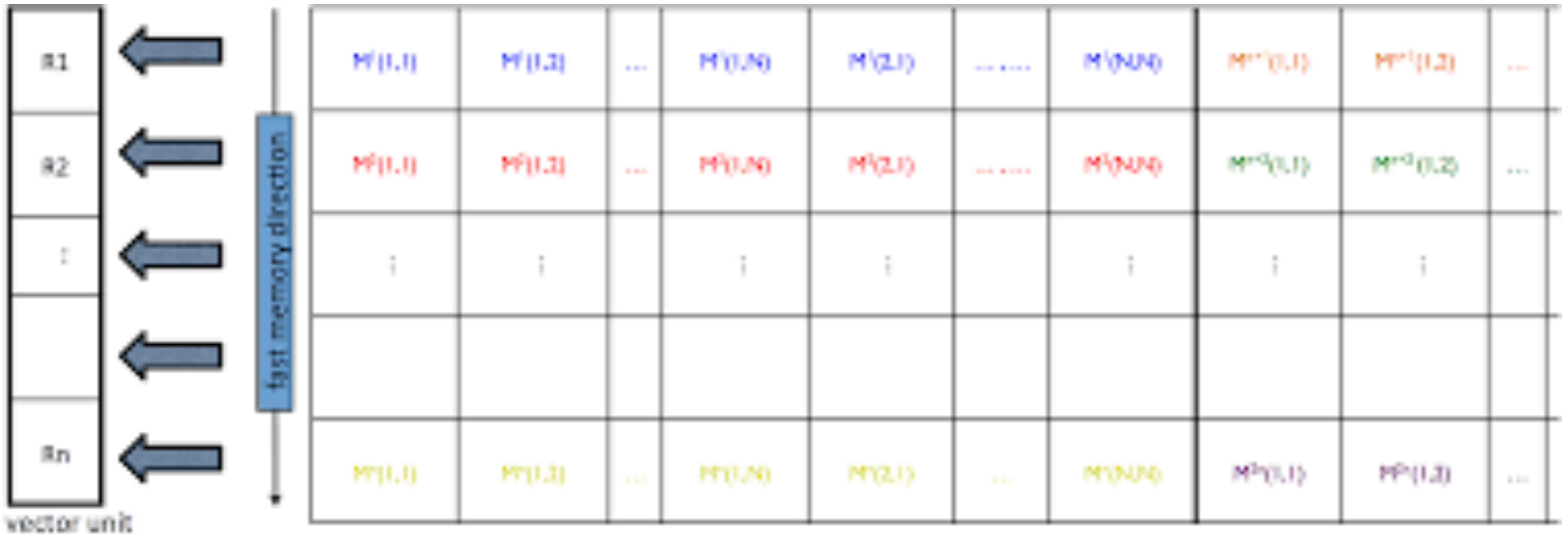
https://en.wikipedia.org/wiki/AOS_and_SOA



```
struct pointlist3D {
    float x[N];
    float y[N];
    float z[N];
};
struct pointlist3D points;
float get_point_x(int i) { return points.x[i]; }
```

SoA

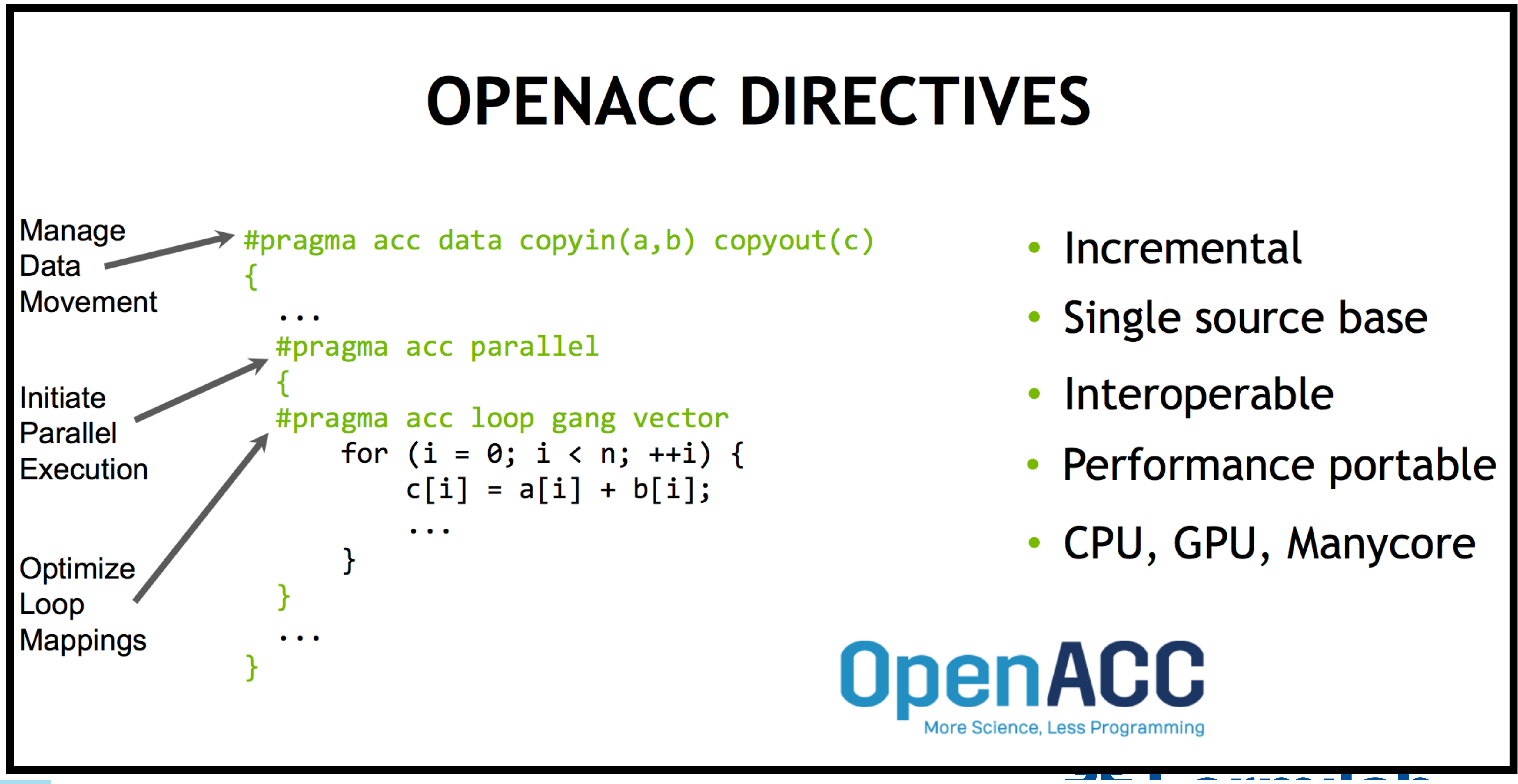
CMS Parallel Kalman Filter



http://trackreco.github.io/

Heterogeneous hardware… heterogeneous software?

- While many parallel programming concepts are valid across platforms, optimizing code for a specific architecture means making it worse for others
  - don't trust cross platform performance comparisons, they are never fair!
- Also, if you want to be able to run on different systems, you may need to have entirely different implementations of your algorithm (e.g. C++ vs CUDA)
  - even worse, we may not even know where the code will eventually be run…
- There is a clear need for portable code!
  - and portable so that performance are "good enough" across platforms
- Option 1: libraries
  - write high level code, rely on portable libraries
  - Kokkos, Raja, Sycl, Eigen…
- Option 2: portable compilers
  - decorate parallel code with pragmas
  - OpenMP, OpenACC, PGI compiler



OPENACC DIRECTIVES

Manage Data Movement

Initiate Parallel Execution

Optimize Loop Mappings

```
#pragma acc data copyin(a,b) copyout(c)
{
  ...
  #pragma acc parallel
  {
  #pragma acc loop gang vector
      for (i = 0; i < n; ++i) {
          c[i] = a[i] + b[i];
          ...
      }
  }
  ...
}
```

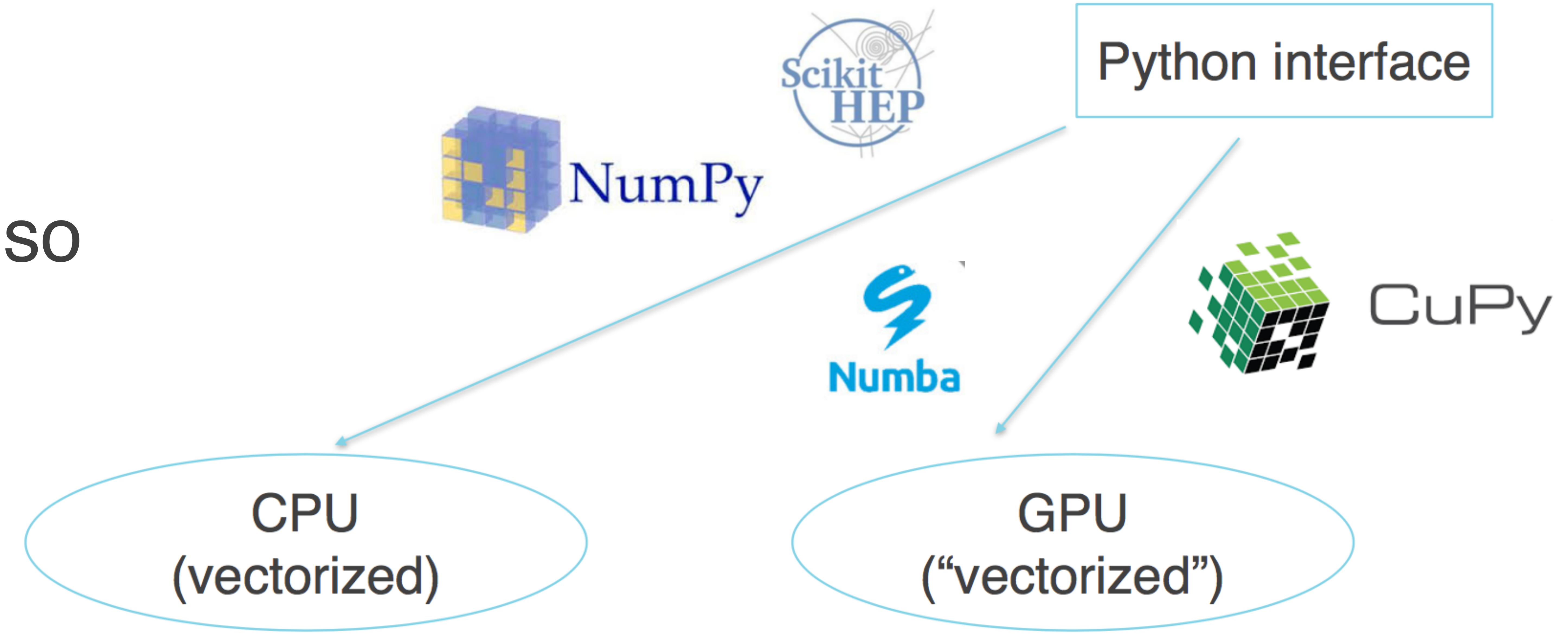
- Incremental
- Single source base
- Interoperable
- Performance portable
- CPU, GPU, Manycore

OpenACC
More Science, Less Programming

PGI Compilers for Heterogeneous Supercomputing, March 2018

Array-based programming

- New kids in town already know numpy… and we force them to learn C++
- Array-based programming is natively SIMD friendly
- Usage actually growing significantly in HEP for analysis
  - Scikit-HEP, uproot, awkward-array
- Portable array-based ecosystem
  - python: numpy, cupy
  - c++: xtensor
- Can it become a solution also for data reconstruction?



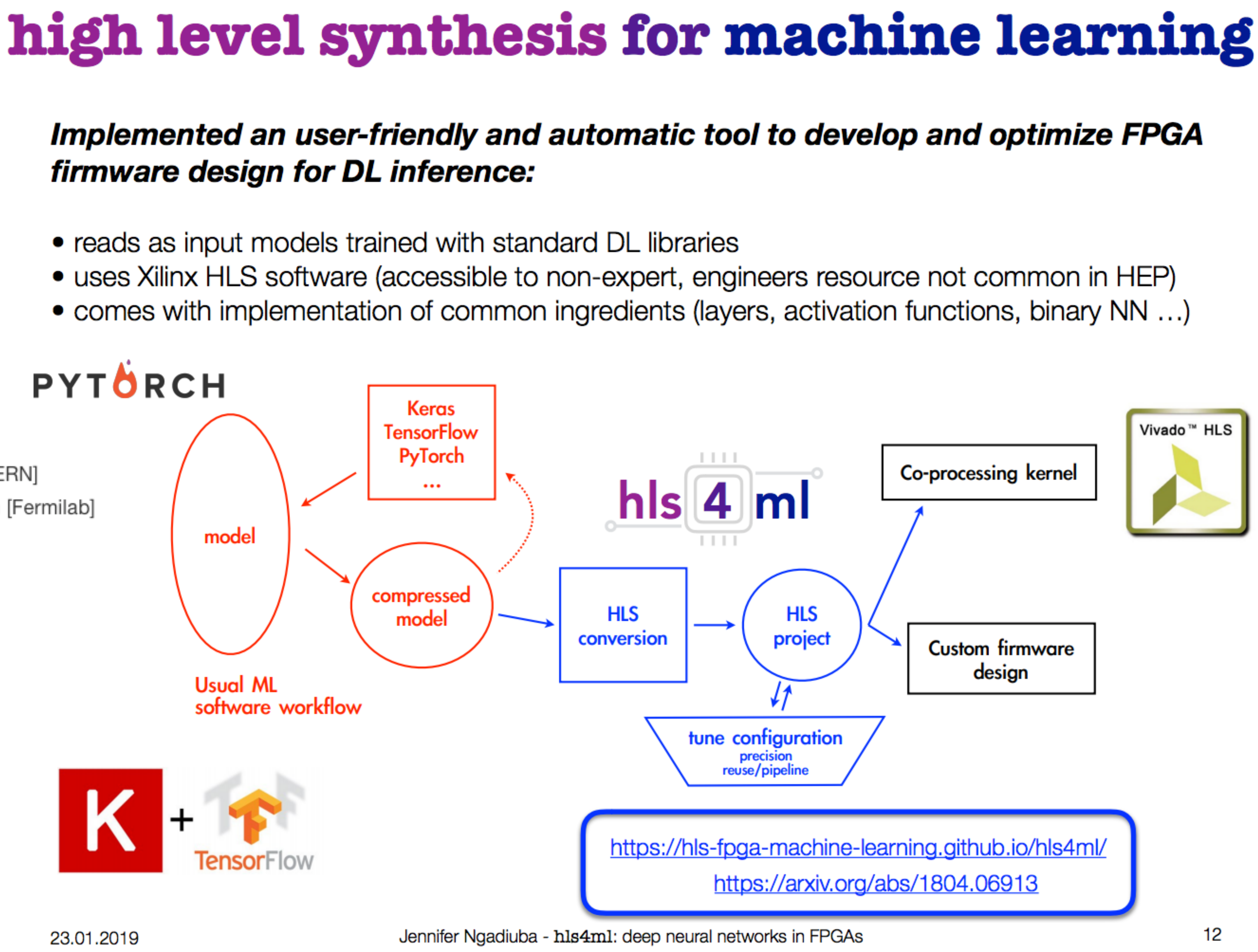
Python interface

CPU
(vectorized)

GPU
("vectorized")

HLS4ML

- Vladimir Loncar, Jennifer Ngadiuba, Maurizio Pierini, Sioni Summers [CERN]
- Javier Duarte, Sergo Jindariani, Benjamin Kreis, Ryan Rivera, Nhan Tran [Fermilab]
- Edward Kreinar [Hawkeye360]
- Song Han, Philip Harris, Dylan Rankin [MIT]
- Zhenbin Wu [University of Illinois at Chicago]
- Mark Neubauer [University of Illinois Urbana-Champaign]
- Shih-Chieh Hsu [University of Washington]
- Giuseppe Di Guglielmo [Columbia University]



high level synthesis for machine learning

Implemented an user-friendly and automatic tool to develop and optimize FPGA firmware design for DL inference:

- reads as input models trained with standard DL libraries
- uses Xilinx HLS software (accessible to non-expert, engineers resource not common in HEP)
- comes with implementation of common ingredients (layers, activation functions, binary NN …)

https://hls-fpga-machine-learning.github.io/hls4ml/
https://arxiv.org/abs/1804.06913

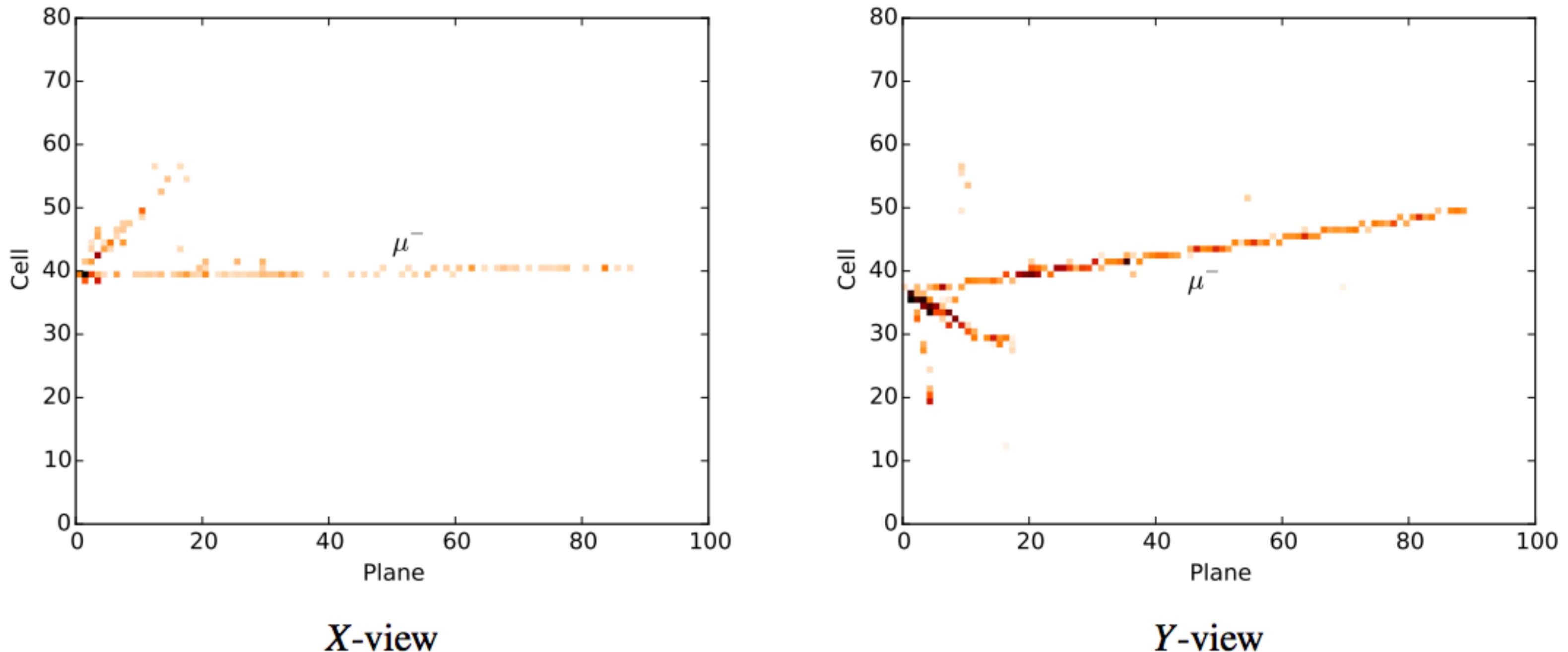
23.01.2019 Jennifer Ngadiuba - **hls4ml**: deep neural networks in FPGAs 12

HPC Opportunities for LArTPC

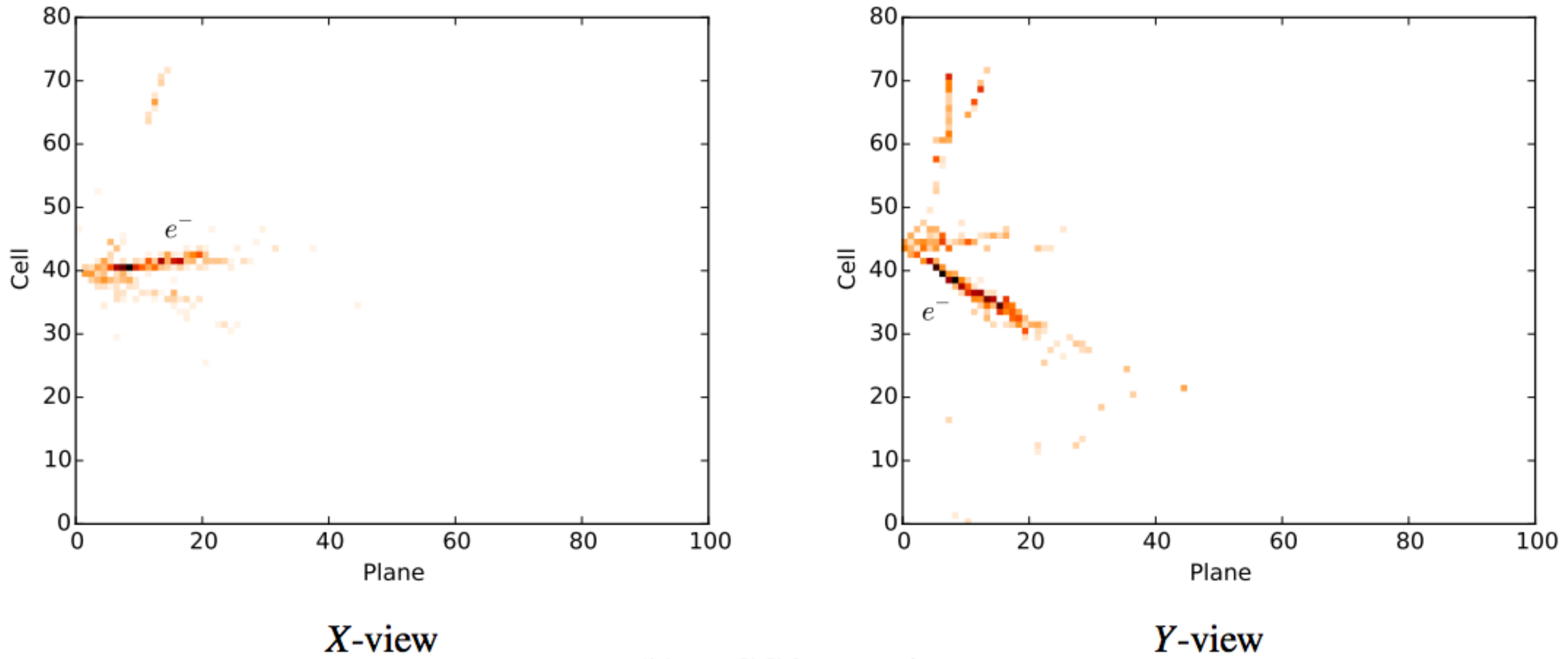
HPC Opportunities for LArTPC: ML

- LArTPC detectors produce gorgeous images:
  natural to apply convolutional neural network techniques
  - e.g. NOVA, uB, DUNE… event classification, energy regression, pixel classification
- LArTPCs can also take advantage of different types of network: Graph NN
- Key: our data is sparse, need to use sparse network models!

Aurisano et al, arXiv:1604.01444

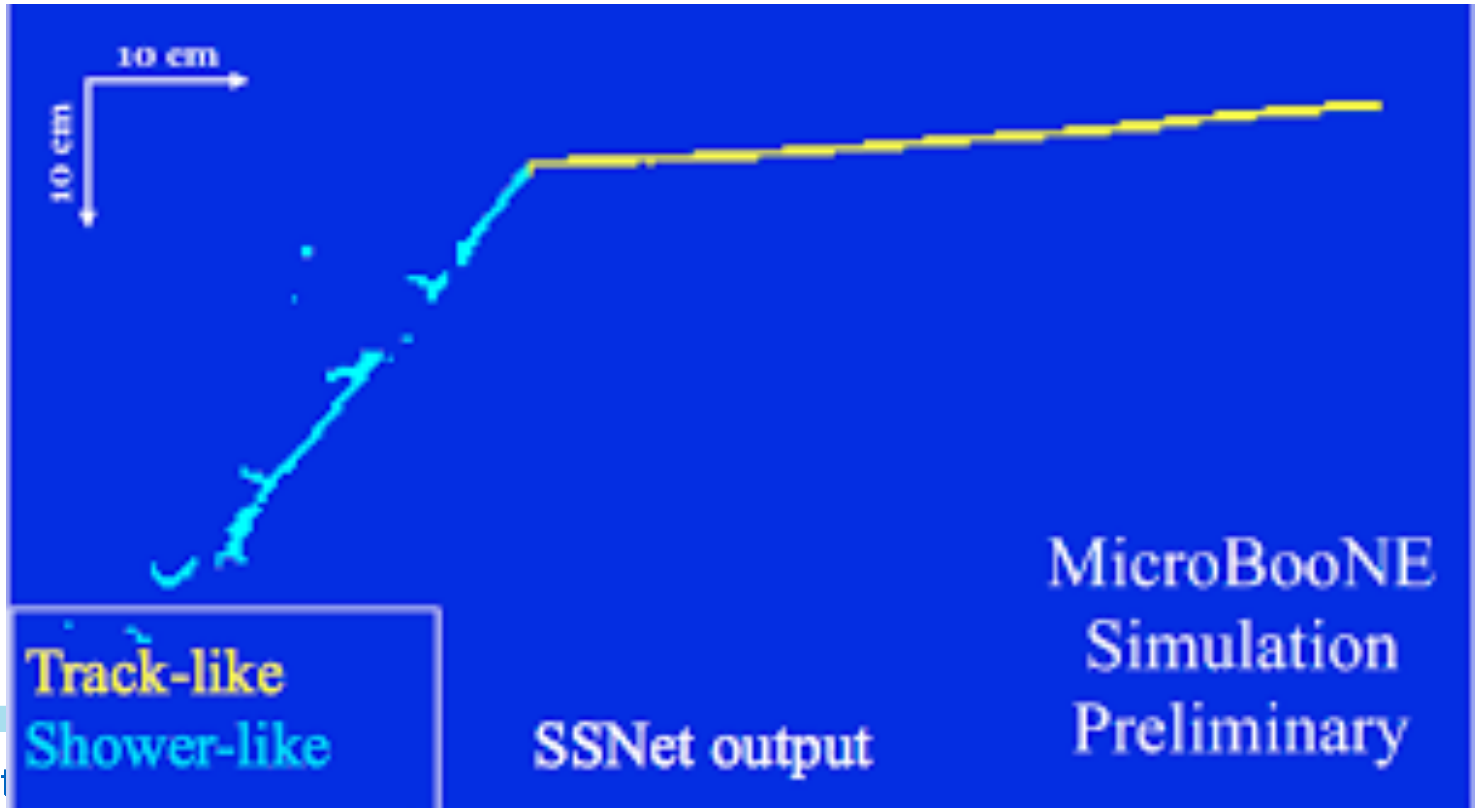


(a) ν_μ CC interaction.



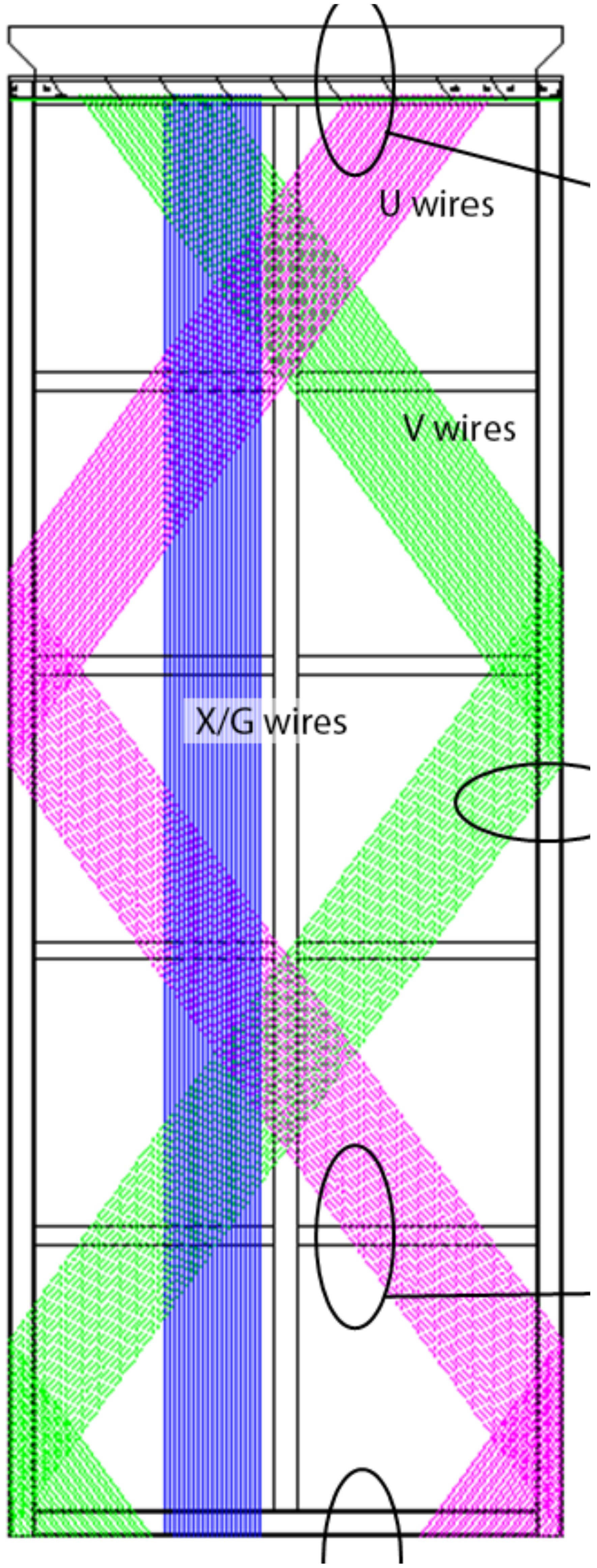
(b) ν_e CC interaction.

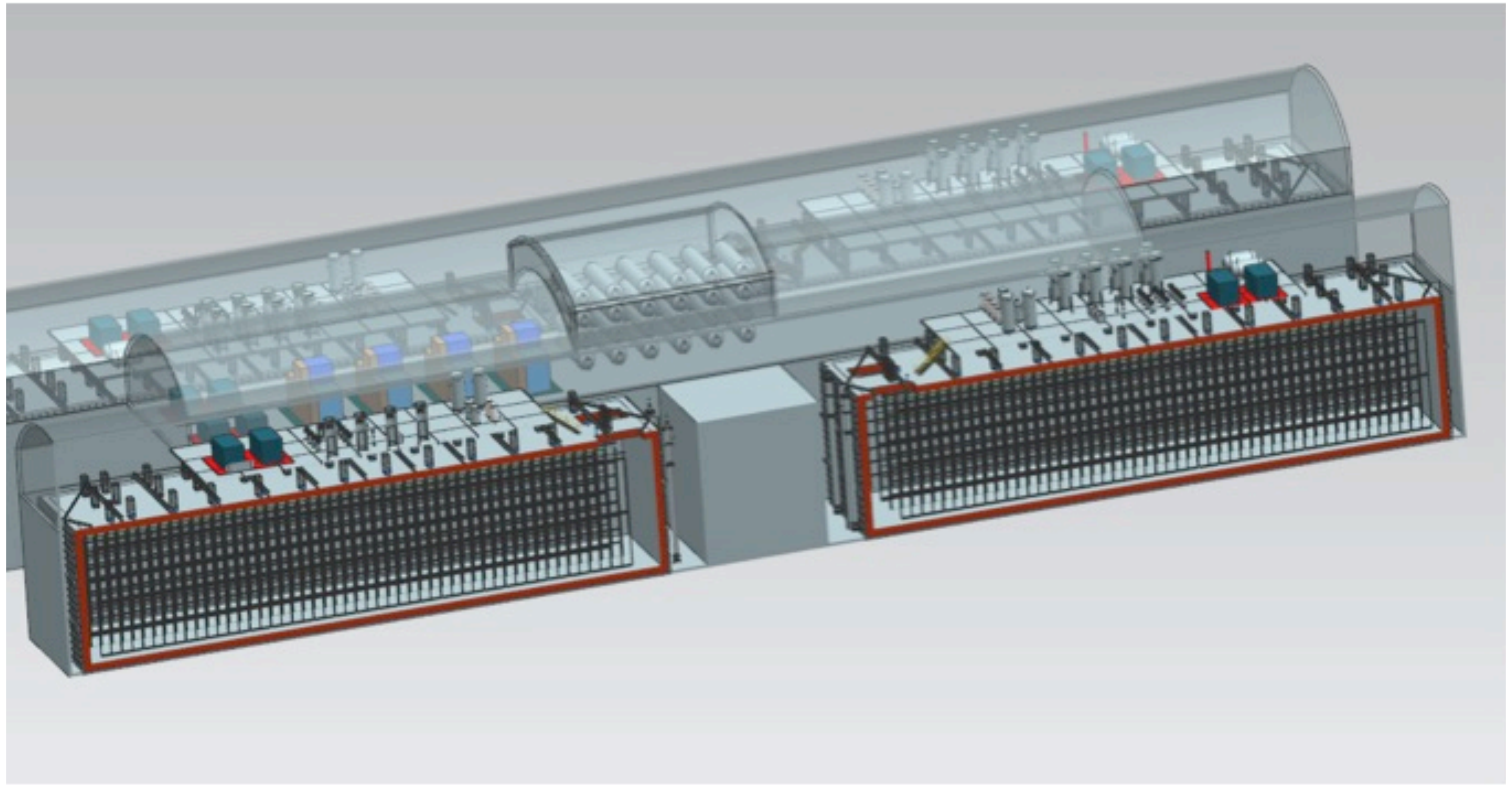
MicroBooNE, arXiv:1808.07269

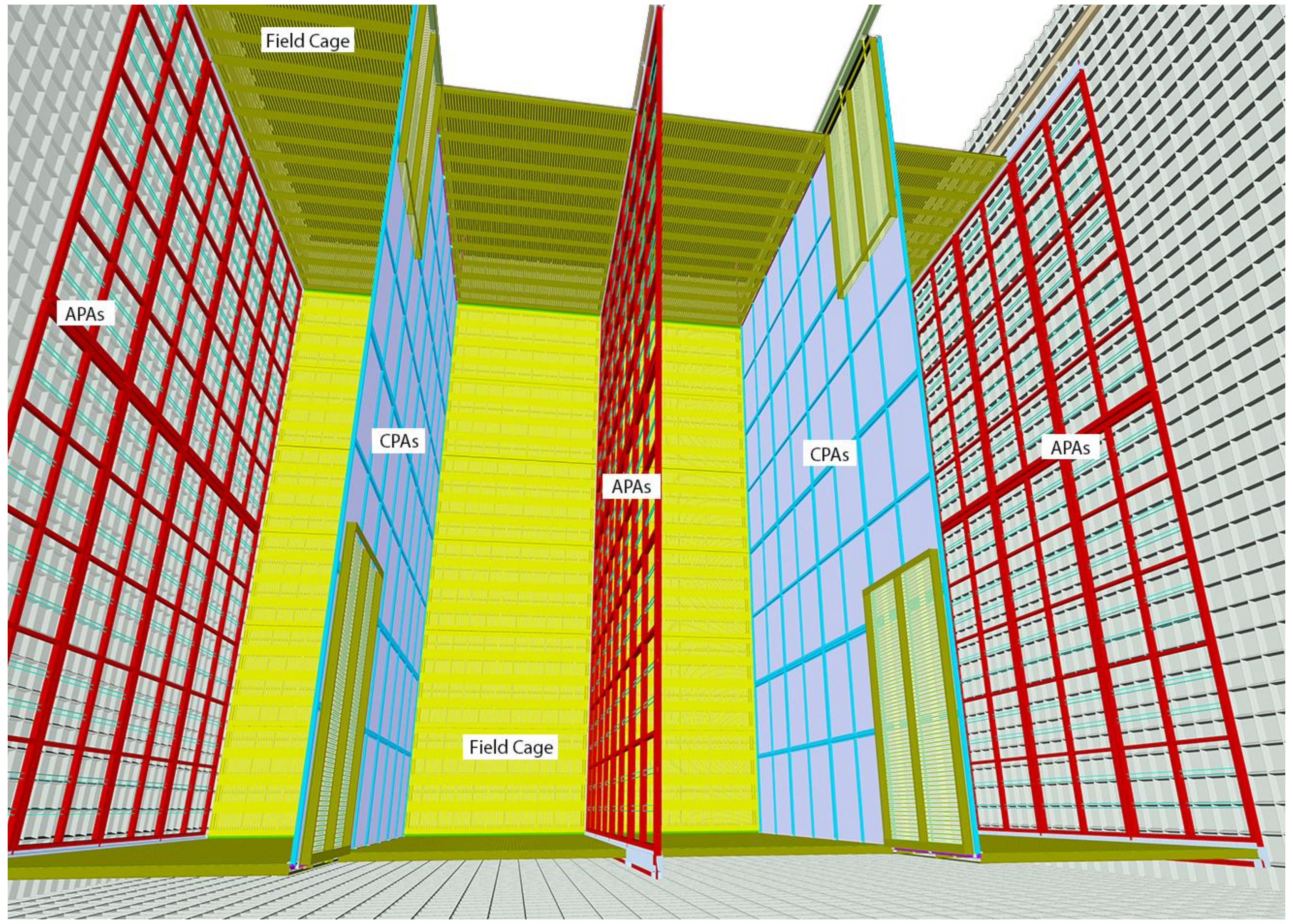


HPC Opportunities for LArTPC: parallelization

- LArTPC detectors are naturally divided in different elements
  - modules, cryostats, TPCs, APAs, boards, wires
- Great opportunity for both SIMD and thread-level parallelism
  - potential to achieve substantial speedups on parallel architectures
- Work has actually started…







First examples of parallelization for LArTPC

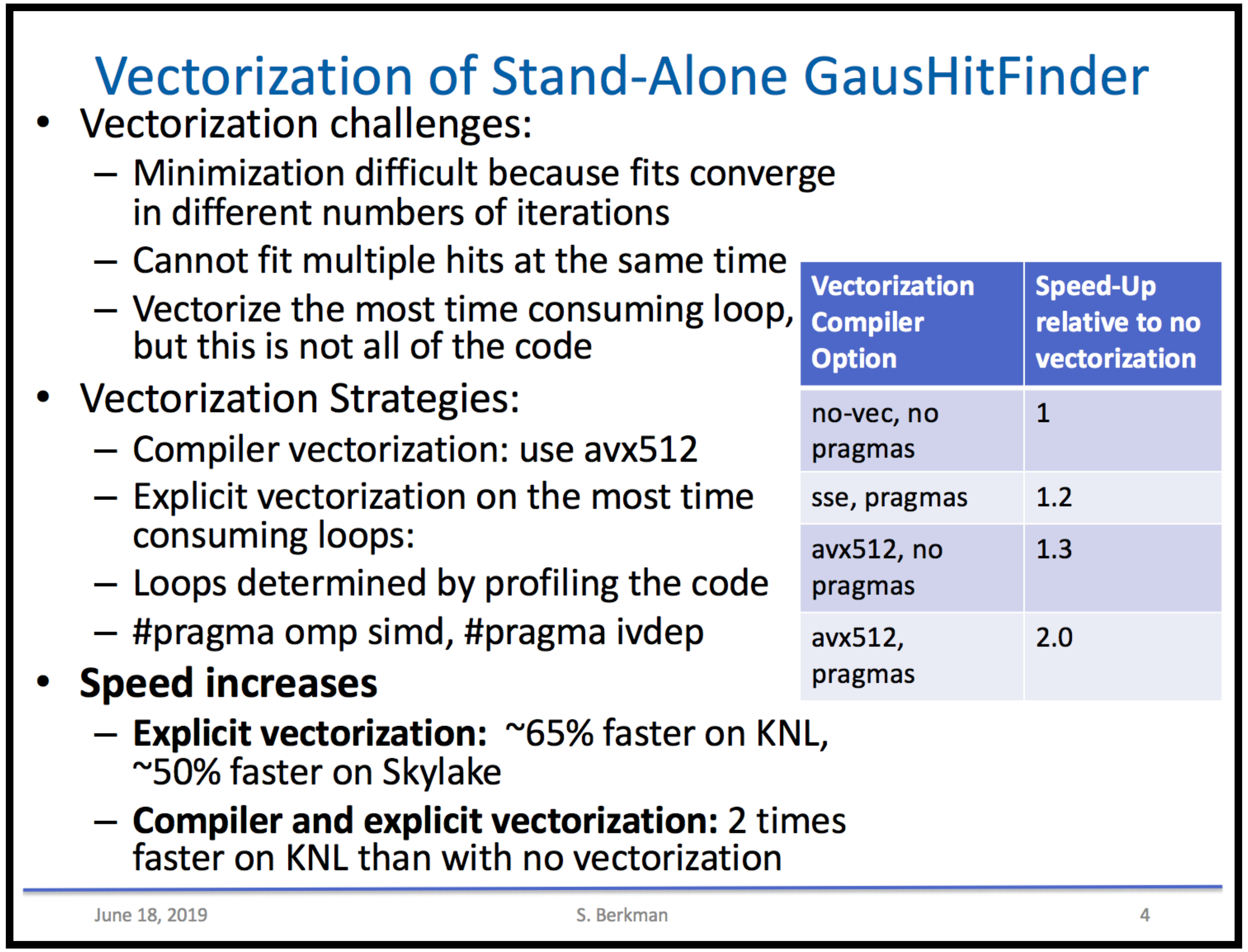
- Art multithreaded and LArSoft becoming thread safe (SciSoft team)

- Icarus testing reconstruction workflows split by TPC
  - Tracy Usher@LArSoft Coordination meeting, May 7, 2019

- DOE SciDAC-4 projects are actively exploring HPC-friendly solutions
  - more in the next slides…

Vectorizing and Parallelizing the Gaus-Hit Finder

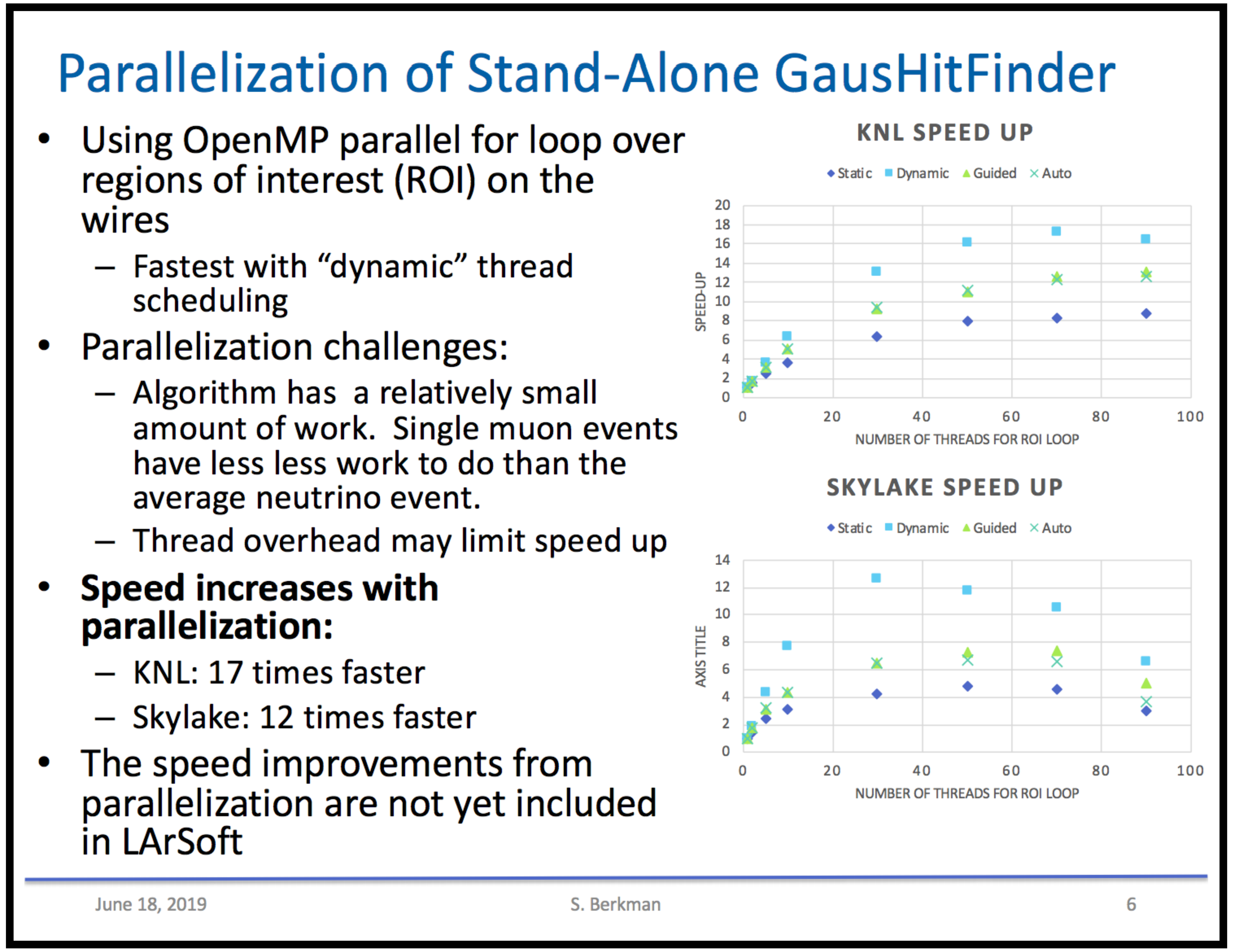
https://computing.fnal.gov/hepreco-scidac4/
(FNAL, UOregon)



Vectorization of Stand-Alone GausHitFinder

- Vectorization challenges:
  - Minimization difficult because fits converge in different numbers of iterations
  - Cannot fit multiple hits at the same time
  - Vectorize the most time consuming loop, but this is not all of the code
- Vectorization Strategies:
  - Compiler vectorization: use avx512
  - Explicit vectorization on the most time consuming loops:
  - Loops determined by profiling the code
  - #pragma omp simd, #pragma ivdep
- **Speed increases**
  - **Explicit vectorization:** ~65% faster on KNL, ~50% faster on Skylake
  - **Compiler and explicit vectorization:** 2 times faster on KNL than with no vectorization

Vectorization Compiler Option	Speed-Up relative to no vectorization
no-vec, no pragmas	1
sse, pragmas	1.2
avx512, no pragmas	1.3
avx512, pragmas	2.0



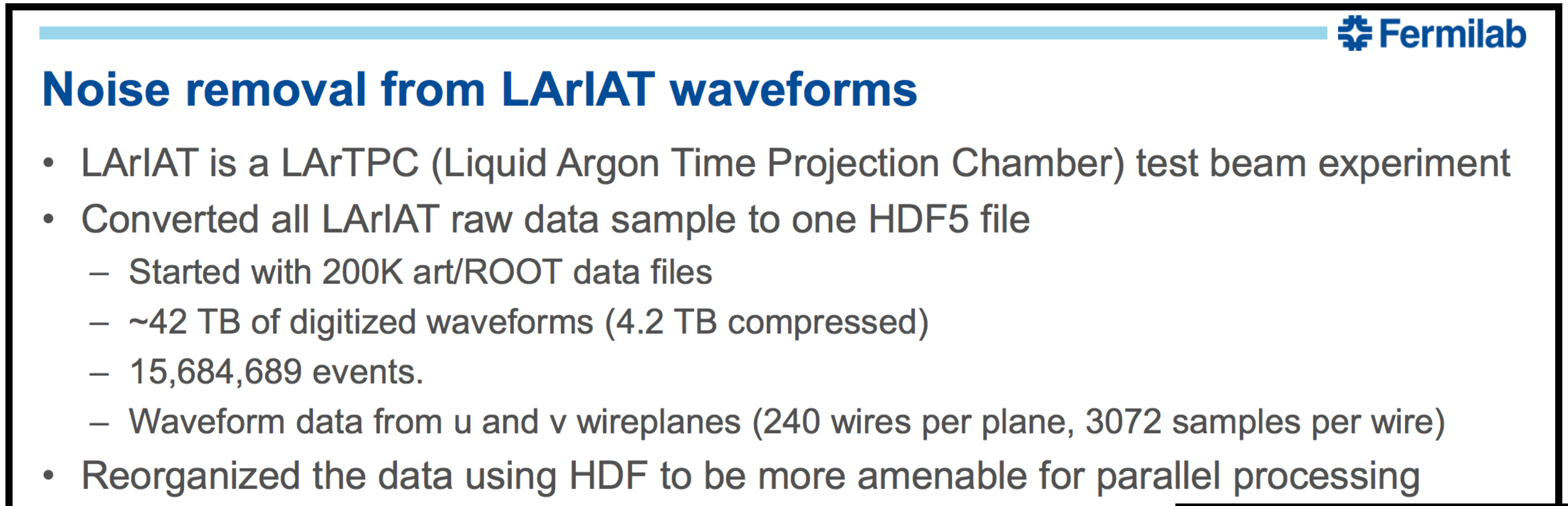
Parallelization of Stand-Alone GausHitFinder

- Using OpenMP parallel for loop over regions of interest (ROI) on the wires
  - Fastest with “dynamic” thread scheduling
- Parallelization challenges:
  - Algorithm has a relatively small amount of work. Single muon events have less less work to do than the average neutrino event.
  - Thread overhead may limit speed up
- **Speed increases with parallelization:**
  - KNL: 17 times faster
  - Skylake: 12 times faster
- The speed improvements from parallelization are not yet included in LArSoft

Sophie Berkman@LArSoft Coordination meeting, June 18, 2019

Integration in LArSoft is underway!

Noise filtering on LArIAT data



Fermilab

Noise removal from LArIAT waveforms

- LArIAT is a LArTPC (Liquid Argon Time Projection Chamber) test beam experiment
- Converted all LArIAT raw data sample to one HDF5 file
  - Started with 200K art/ROOT data files
  - ~42 TB of digitized waveforms (4.2 TB compressed)
  - 15,684,689 events.
  - Waveform data from u and v wireplanes (240 wires per plane, 3072 samples per wire)
- Reorganized the data using HDF to be more amenable for parallel processing
- Processing the entire LArIAT raw data sample
  - First step of reconstruction is noise reduction using FFTs

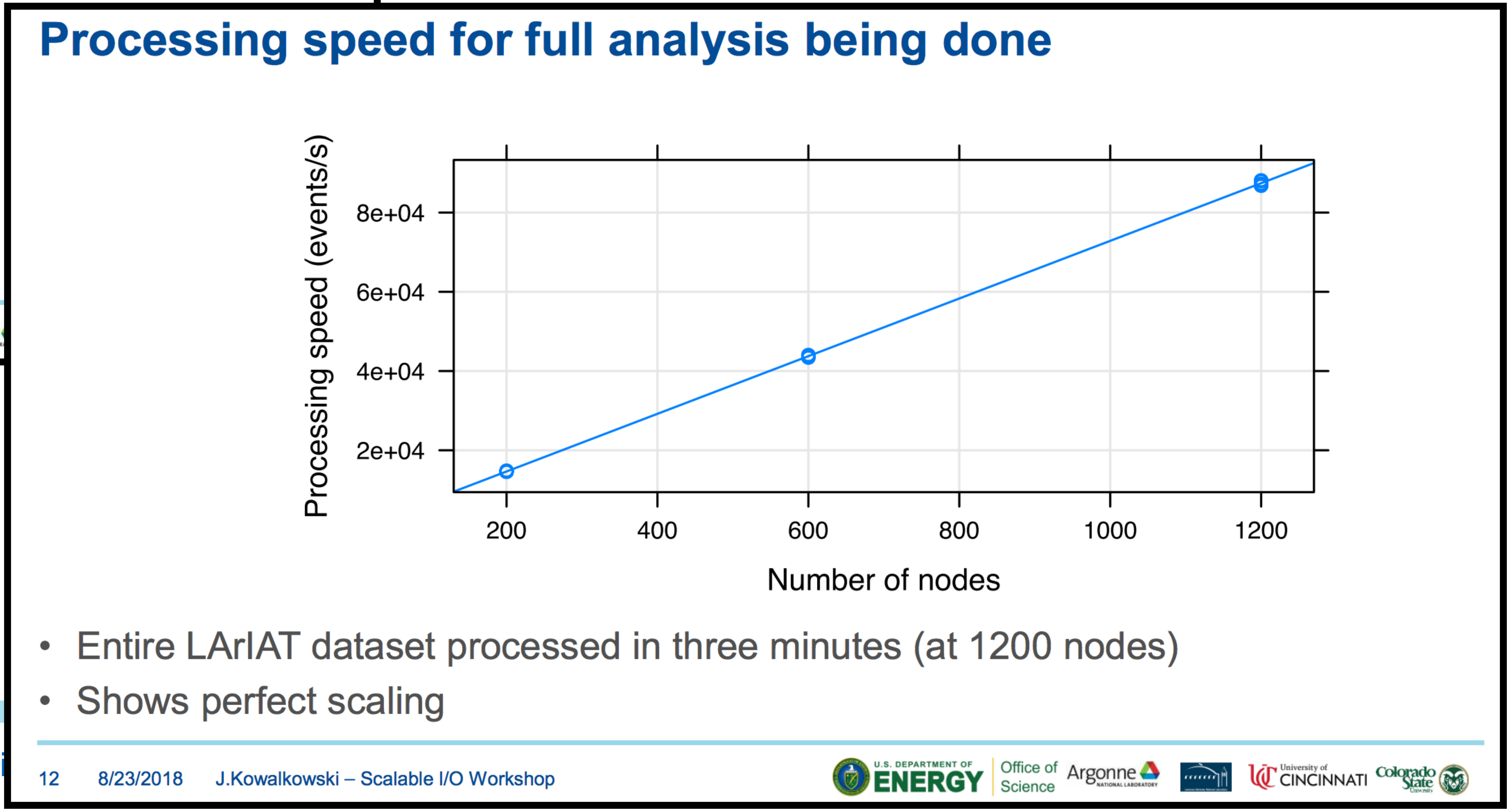
9 8/23/2018 J.Kowalkowski – Scalable I/O Workshop

FERMILAB-CONF-18-577-CD

https://computing.fnal.gov/hep-on-hpc/

(FNAL, Argonne, Berkeley,
UCincinnaty, Colorado State)

J.Kowalkowski@Scalable I/O Workshop 2018



Processing speed for full analysis being done

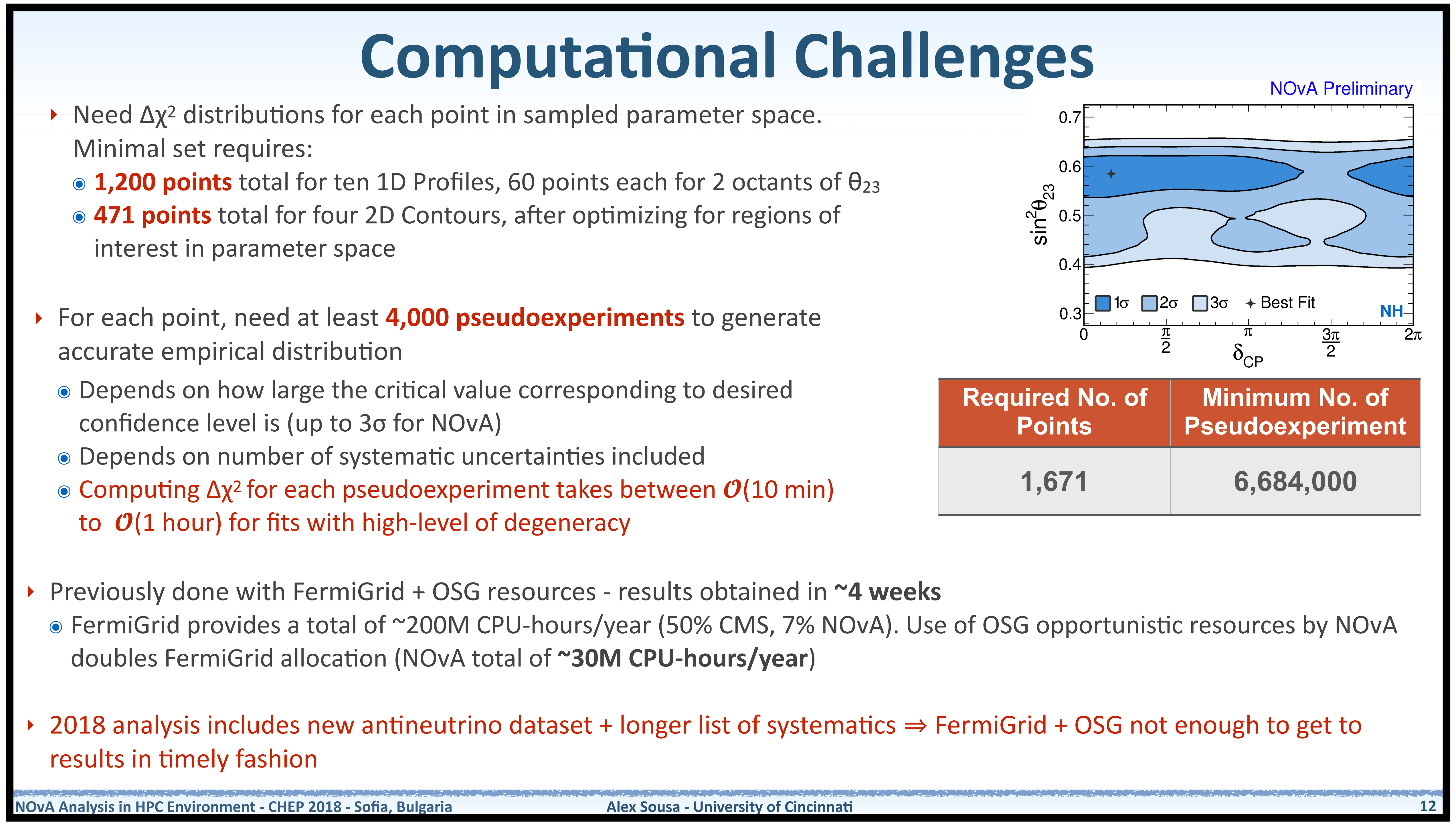
- Entire LArIAT dataset processed in three minutes (at 1200 nodes)
- Shows perfect scaling

12 8/23/2018 J.Kowalkowski – Scalable I/O Workshop

Oscillation parameter extraction with Feldman-Cousins fits

https://computing.fnal.gov/hep-on-hpc/

(FNAL, Argonne, Berkeley, UCincinnaty, Colorado State)

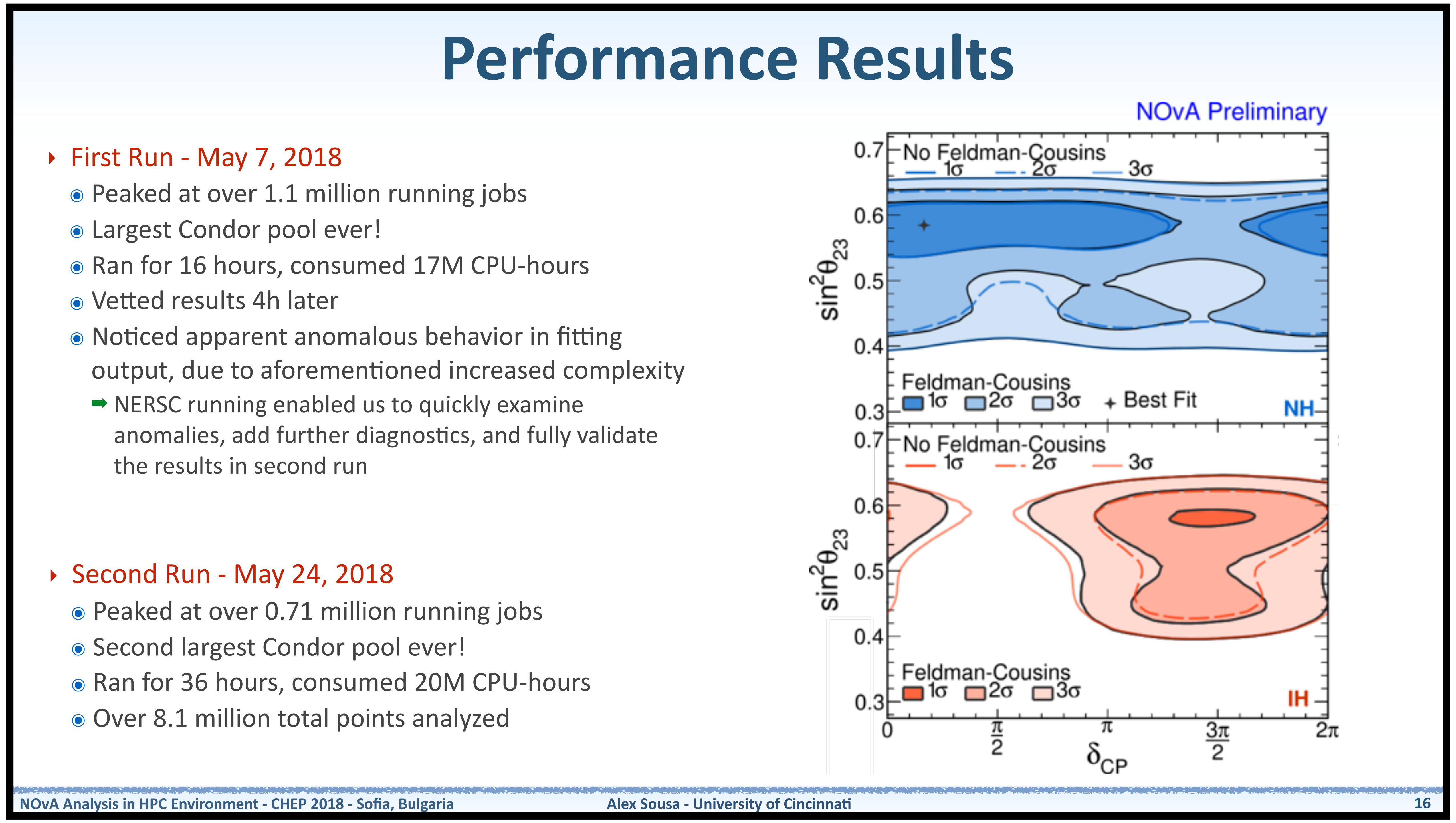


Computational Challenges

- Need Δχ² distributions for each point in sampled parameter space. Minimal set requires:
  - **1,200 points** total for ten 1D Profiles, 60 points each for 2 octants of θ_{23}
  - **471 points** total for four 2D Contours, after optimizing for regions of interest in parameter space
- For each point, need at least **4,000 pseudoexperiments** to generate accurate empirical distribution
  - Depends on how large the critical value corresponding to desired confidence level is (up to 3σ for NOvA)
  - Depends on number of systematic uncertainties included
  - Computing Δχ² for each pseudoexperiment takes between $\mathcal{O}$(10 min) to $\mathcal{O}$(1 hour) for fits with high-level of degeneracy
- Previously done with FermiGrid + OSG resources - results obtained in **~4 weeks**
  - FermiGrid provides a total of ~200M CPU-hours/year (50% CMS, 7% NOvA). Use of OSG opportunistic resources by NOvA doubles FermiGrid allocation (NOvA total of **~30M CPU-hours/year**)
- 2018 analysis includes new antineutrino dataset + longer list of systematics ⇒ FermiGrid + OSG not enough to get to results in timely fashion

Required No. of Points	Minimum No. of Pseudoexperiment
1,671	6,684,000

NOvA Analysis in HPC Environment - CHEP 2018 - Sofia, Bulgaria Alex Sousa - University of Cincinnati 12



Performance Results

- First Run - May 7, 2018
  - Peaked at over 1.1 million running jobs
  - Largest Condor pool ever!
  - Ran for 16 hours, consumed 17M CPU-hours
  - Vetted results 4h later
  - Noticed apparent anomalous behavior in fitting output, due to aforementioned increased complexity
    - ➡ NERSC running enabled us to quickly examine anomalies, add further diagnostics, and fully validate the results in second run
- Second Run - May 24, 2018
  - Peaked at over 0.71 million running jobs
  - Second largest Condor pool ever!
  - Ran for 36 hours, consumed 20M CPU-hours
  - Over 8.1 million total points analyzed

NOvA Analysis in HPC Environment - CHEP 2018 - Sofia, Bulgaria Alex Sousa - University of Cincinnati 16

A.Sousa@CHEP2018

50x speedup achieved thanks to supercomputer!

Exploit HPC for LArTPC workflows?

- Many workflows of LArTPC experiments could exploit HPC resources
  - simulation, reconstruction (signal processing), deep learning (training and inference), analysis

- Our experiments operate in terms of production campaigns
  - typically at a give time of the year, in advance of conferences
  - most time consuming stages are then frozen for longer periods of time, with faster second-pass processing repeated multiple times (this is happening now in uB)
  - HPC centers are a possible resource for the once/twice-per-year heavy workflows!
    - something like signal processing + DL inference?

- See next talk for more discussions on future workflows!