

# Exascale Computing Challenges: Application Perspectives



Erik Draeger, Lawrence Livermore National Laboratory (LLNL)  
*Deputy Director of Application Development for the DOE Exascale Computing Project (ECP)*

2023 PSAAP IV Preproposal Meeting

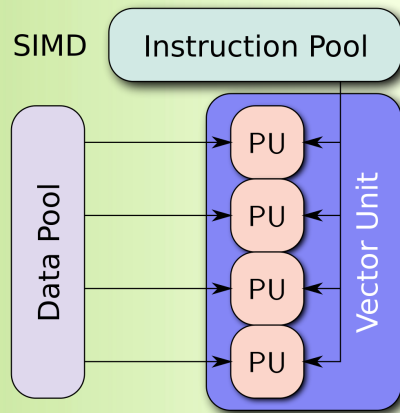
August 8, 2023

# We are at a transition point in HPC

## Vector Era

MFLOP/s - GFLOP/s

- Parallelism through vector processors.
- Codes often written at very low level to make optimal use of hardware.



1980s

## Distributed Memory Era

GFLOP/s - TFLOP/s – PFLOP/s

- Parallelism through MPI.
- Using an optimal parallel algorithm was critical to avoid duplication of work or unnecessary communication.
- Once distributed, code could be treated serially.

- For the most part, an MPI code ran anywhere. For best performance, key kernels could be tuned.
- As CPU frequencies stopped increasing, parallelism became more extreme and specialized hardware more common.

10s to 100s of cores

1000s of cores

10<sup>4</sup> to 10<sup>6</sup> cores

1990s

2000s

2010s

# We are at a transition point in HPC

## Heterogeneous Era

PFLOP/s - EFLOP/s

- CPUs + accelerators with separate memory spaces to start, unclear what else will join the fray.
- Massive fine-grained parallelism required.
- Programming model has to match the architecture.
- Architectural landscape is changing rapidly, with an unclear future.

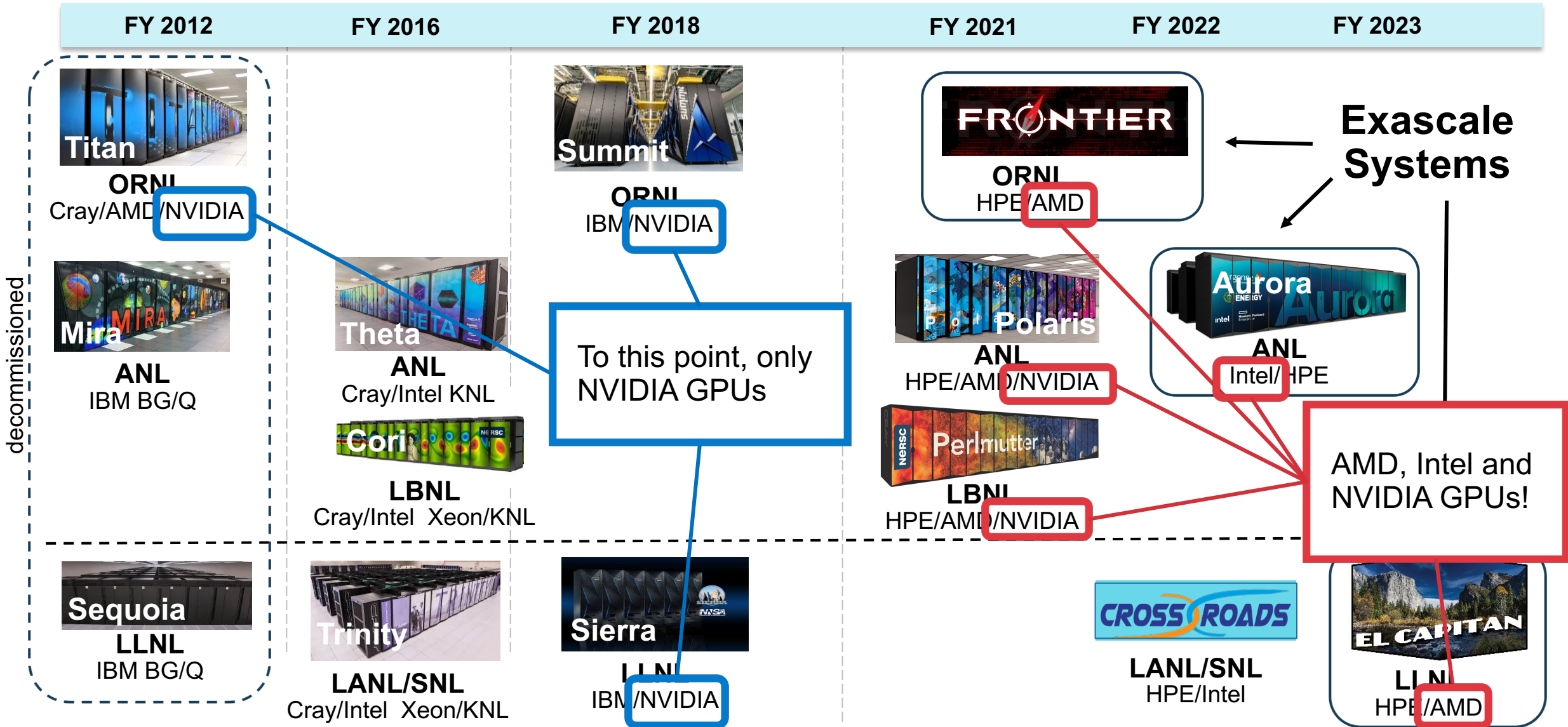
## Heterogeneity is the new reality

- Computational horsepower has significantly outpaced memory capacity and speed.
- Separate memory spaces add complexity, and can cause performance issues (e.g. NUMA) or errors if not handled correctly.
- Performance or portability?
- Refactoring an existing code is a lot of work! You really don't want to have to do it again in ten years.

2010s

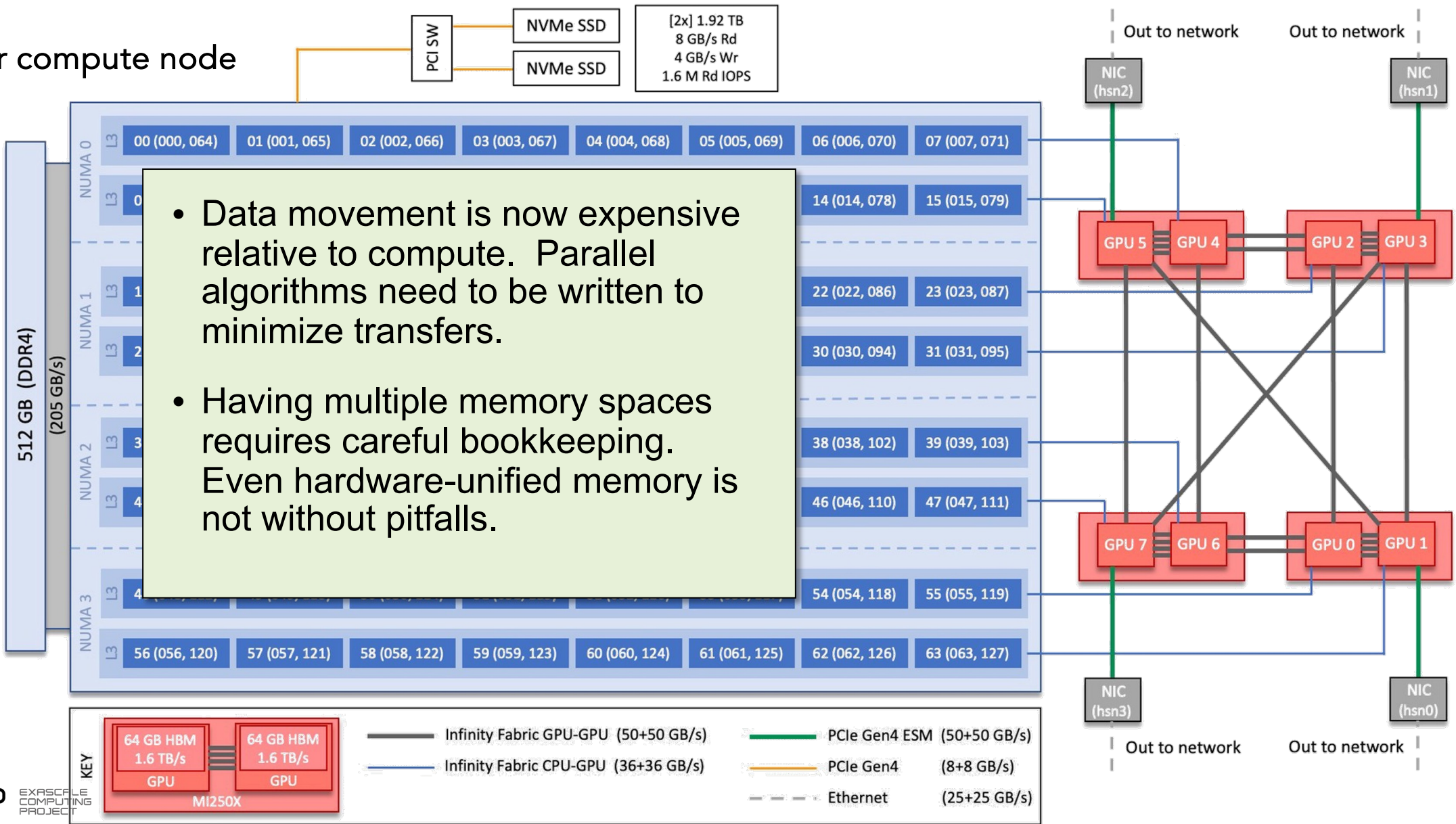
2020s

# DOE HPC Roadmap to Exascale Systems



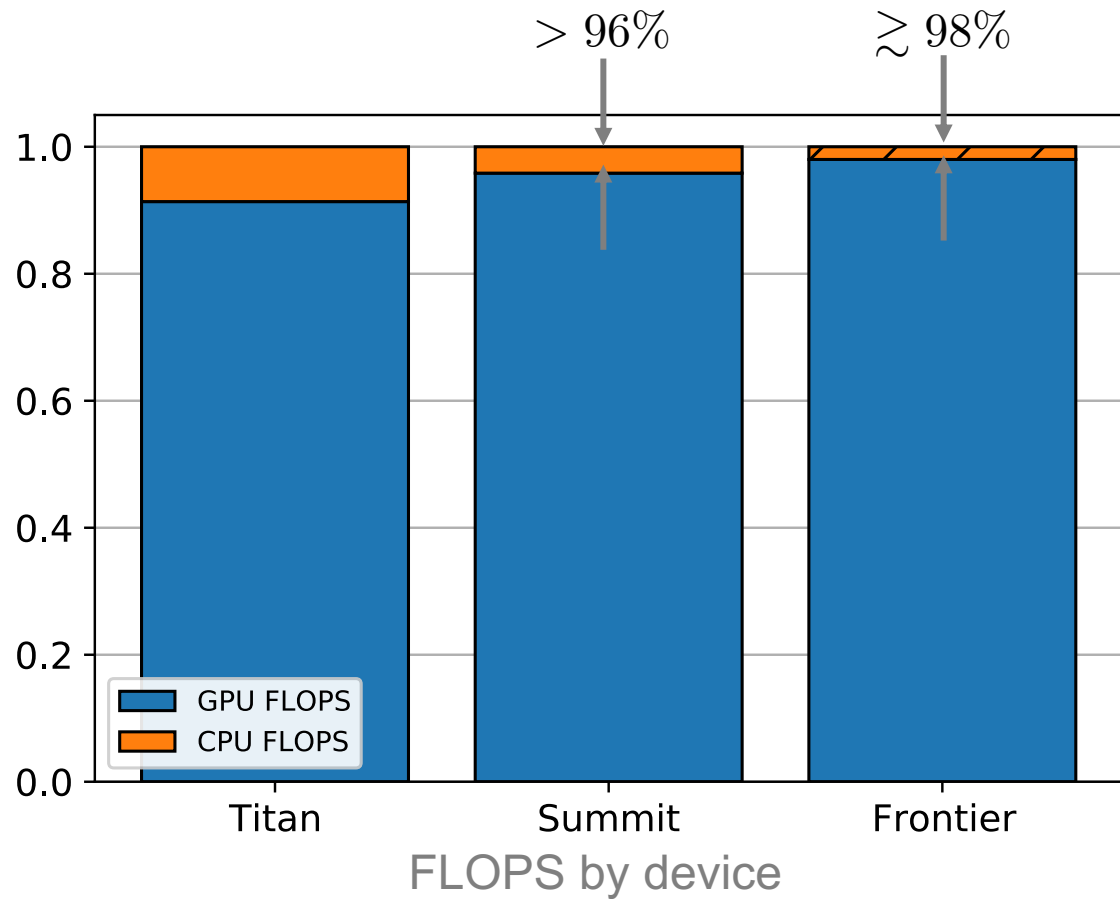
# Why heterogeneous computing is hard

## Frontier compute node



- Data movement is now expensive relative to compute. Parallel algorithms need to be written to minimize transfers.
- Having multiple memory spaces requires careful bookkeeping. Even hardware-unified memory is not without pitfalls.

# Performance on current and next-gen HPC architectures requires effective use of accelerators



Getting performance on-node is the real challenge

- We used to think of scaling as running  $O(100k) - O(1M)$  MPI ranks
- Starting in 2016 (Summit) the FLOPS per node has risen dramatically (48 TF)
- This focuses effort on “scaling in” instead of “scaling out”
- Bottom line: we need to do more work per node on fewer MPI ranks.
- Using the GPUs well is critical!

# Why using accelerators is hard: SIMD/SIMT

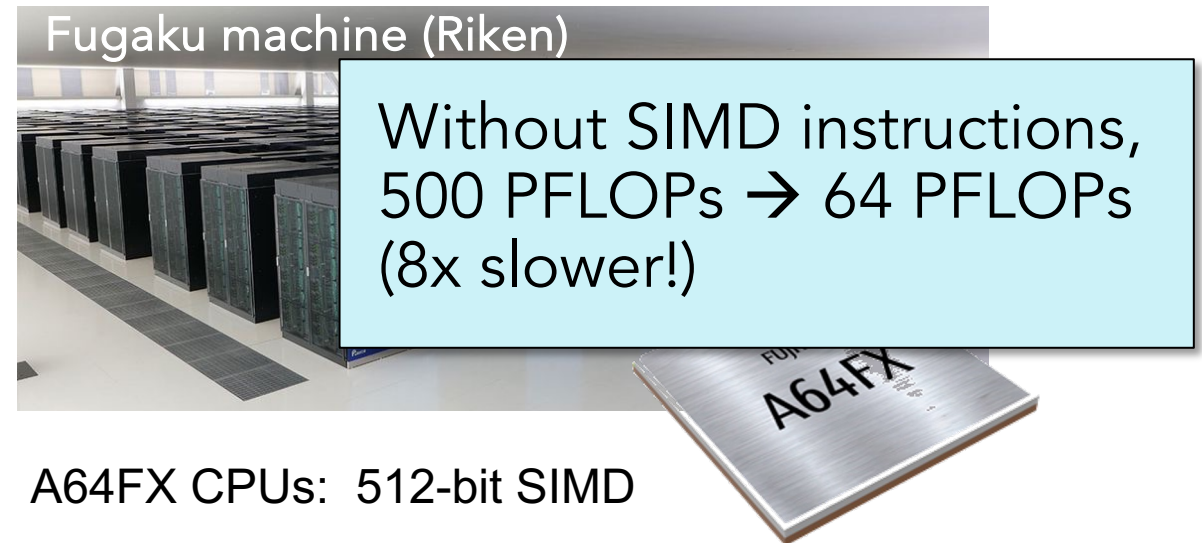
- GPUs use SIMT (Single Instruction, Multiple Threads). GPU architecture is designed around the assumption of highly concurrent workloads.
- Threads that follow different code paths are executed separately (sequentially).
- All CPUs now utilize vector instructions (SIMD) to achieve their advertised peak performance.
- SIMD = Single Instruction, Multiple Data. Requires chunks of data all traversing the same code path at the same time.
- If compiler can't find a full SIMD instruction, it reverts to sequential.

## Scalar Operation

$$\begin{array}{l} A_1 \times B_1 = C_1 \\ A_2 \times B_2 = C_2 \\ A_3 \times B_3 = C_3 \\ A_4 \times B_4 = C_4 \end{array}$$

## SIMD Operation

$$\begin{array}{l} A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} \times \begin{array}{l} B_1 \\ B_2 \\ B_3 \\ B_4 \end{array} = \begin{array}{l} C_1 \\ C_2 \\ C_3 \\ C_4 \end{array}$$



# GPUs have forced us to reevaluate everything

- The rapid change from distributed memory, CPU-only systems to heterogeneous, CPU-GPU has shaken up computational science.
- Some algorithms are fundamentally incompatible with SIMD/SIMT architectures. Others need to be carefully tuned for each type of GPU.
- Long-standing codes were designed around assumptions that no longer hold, it's unclear how to adapt them for an uncertain future.



# The Exascale Computing Project

7  
Years  
\$1.8B

- A seven-year, \$1.8B R&D effort that launched in 2016

6  
Core DOE  
Labs

- Argonne
- Lawrence Berkeley
- Lawrence Livermore
- Oak Ridge
- Sandia
- Los Alamos

- Staff from most of the 17 DOE national laboratories take part in the project
- 6 HPC vendors participated in Path Forward supporting R&D

3  
Technical  
Focus  
Areas

- Hardware and Integration
- Software Technology
- **Application Development**

81  
R&D Teams  
1000  
Researchers

- 81 research teams, roughly 10 researchers per team

# Science and beyond: Applications and discovery in ECP

National security

Energy security

Economic security

Scientific discovery

Earth systems

Health care

**24 applications and 6 co-design projects**

- Including **62 separate codes**
- Representing over **10 million lines of code**
- Many supporting large user communities
- Covering broad range of mission critical S&E domains
- Mostly all MPI or MPI+OpenMP on CPUs at beginning of ECP
- Each project defines a domain-specific challenge problem for final benchmark
- Applications are evaluated in one of two categories
- Performance – achieve a **50x** performance increase
- Capability – utilize new architectures for expanded S&E

Cosmological probe of the standard model of particle physics

Validate fundamental laws of nature

Plasma wakefield accelerator design

Light source-enabled analysis of protein structure and design

Find, predict, and synthesize materials and properties

Control magnetically confined plasmas

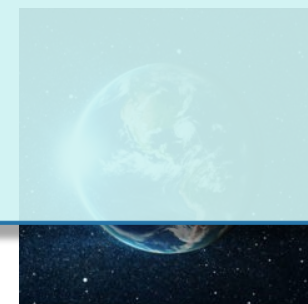
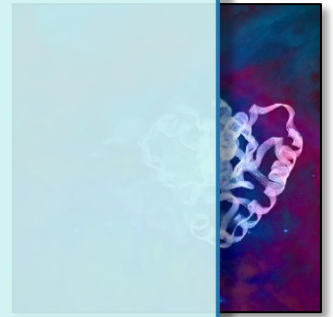
Demystify origin of chemical elements

Accurate regional impact assessments in **Earth system models**

Stress-resistant crop analysis and catalytic conversion of **biomass-derived alcohols**

Genomics for analysis of biogeochemical cycles, climate change, environmental remediation

Accelerate and translate **cancer research** (partnership with NIH)



**Biofuel catalyst design**

# Four key ingredients of an ECP Application Development Project



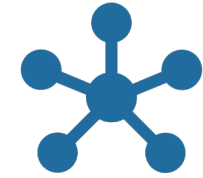
Science goal



Algorithmic  
innovation



Porting



Integration

# Four key ingredients of an ECP Application Development Project



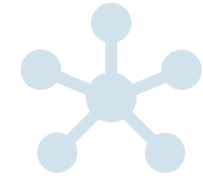
Science goal



Algorithmic  
innovation



Porting



Integration

# Algorithmic innovation goes beyond simply porting code

*"The downside of ... benchmarks is that innovation is chiefly limited to the architecture and compiler. Better data structures, algorithms, programming languages, ...cannot be used, since that would give a misleading result. The system could win because of, say, the algorithm, and not because of the hardware or the compiler. While these guidelines are understandable when the foundations of computing are relatively stable, as they were in the 1990s and the first half of this decade, they are undesirable during a programming revolution. For this revolution to succeed, we need to encourage innovation at all levels."*

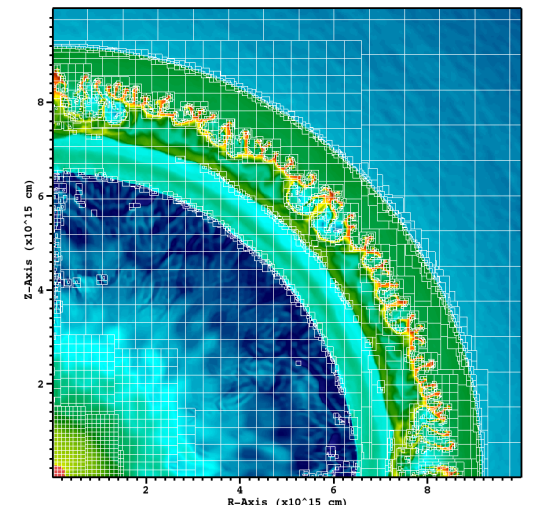
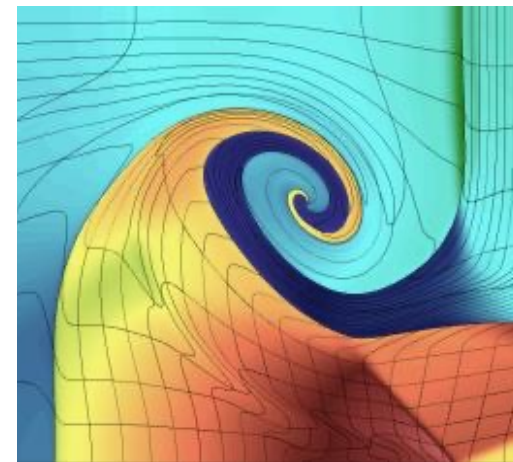
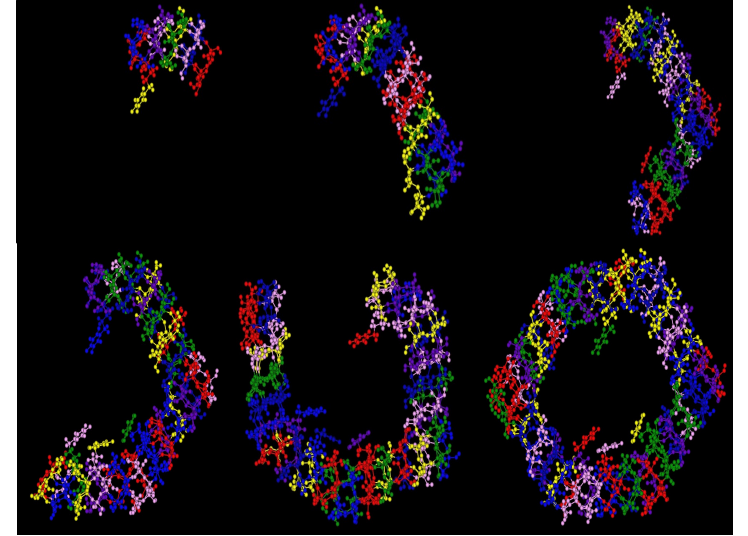
-Hennessy and Patterson, Computer Architecture, A Quantitative Approach

# GPUs do best for codes given ...

- ✓ massive fine-grained parallelism
- ✓ concentrated performance bottlenecks
- ✓ weak scaling problems
- ✓ high arithmetic intensity and/or low data movement
- ✓ minimal branching
- ✓ high FLOP to byte (of storage) ratio
- ✓ use of specialized instructions

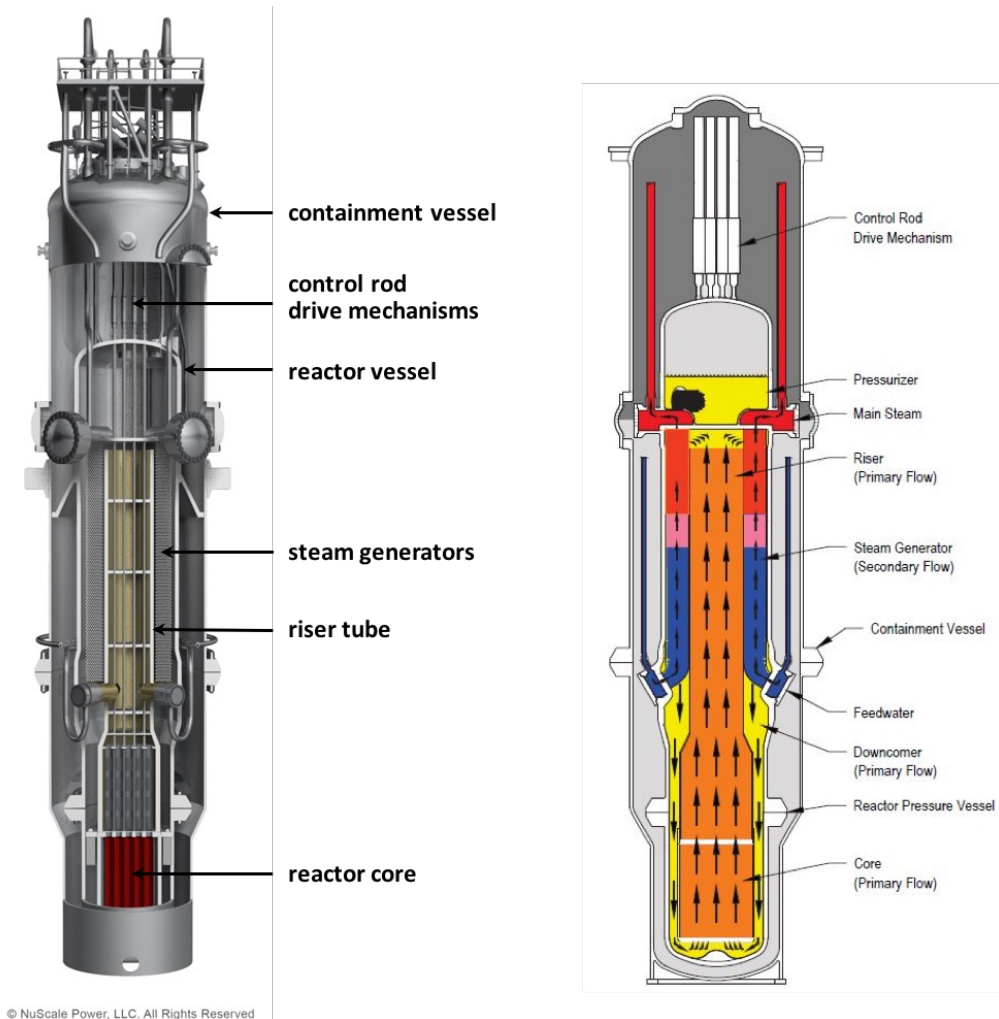
# Algorithmic innovation: domain-driven adaptations critical for making efficient use of exascale systems

- Inherent strong scaling challenges on GPU-based systems →
  - Ensembles vs. time averaging
  - Fluid dynamics, seismology, molecular dynamics, time-stepping
- Increased dimensions of (fine-grained) parallelism to feed GPUs
  - Ray tracing, Markov Chain Monte Carlo, fragmentation methods
- Localized physics models to maximize "free flops"
  - MMF, electron subcycling, enhanced subgrid models, high-order discretizations
- Alternatives to sparse linear systems
  - Higher order methods, Monte Carlo
- Reduced branching
  - Event-based models



# Example: modeling and simulation of small modular reactors

- ExaSMR is a coupled multiphysics ECP application to perform “virtual experiment” simulations of small modular nuclear reactor designs.
- Small modular nuclear reactors present significant simulation challenges
  - Small size invalidates existing low-order models
  - Natural circulation flow requires high-fidelity fluid flow simulation
- Two primary methods:
  - Monte Carlo neutronics
  - CFD with turbulence models

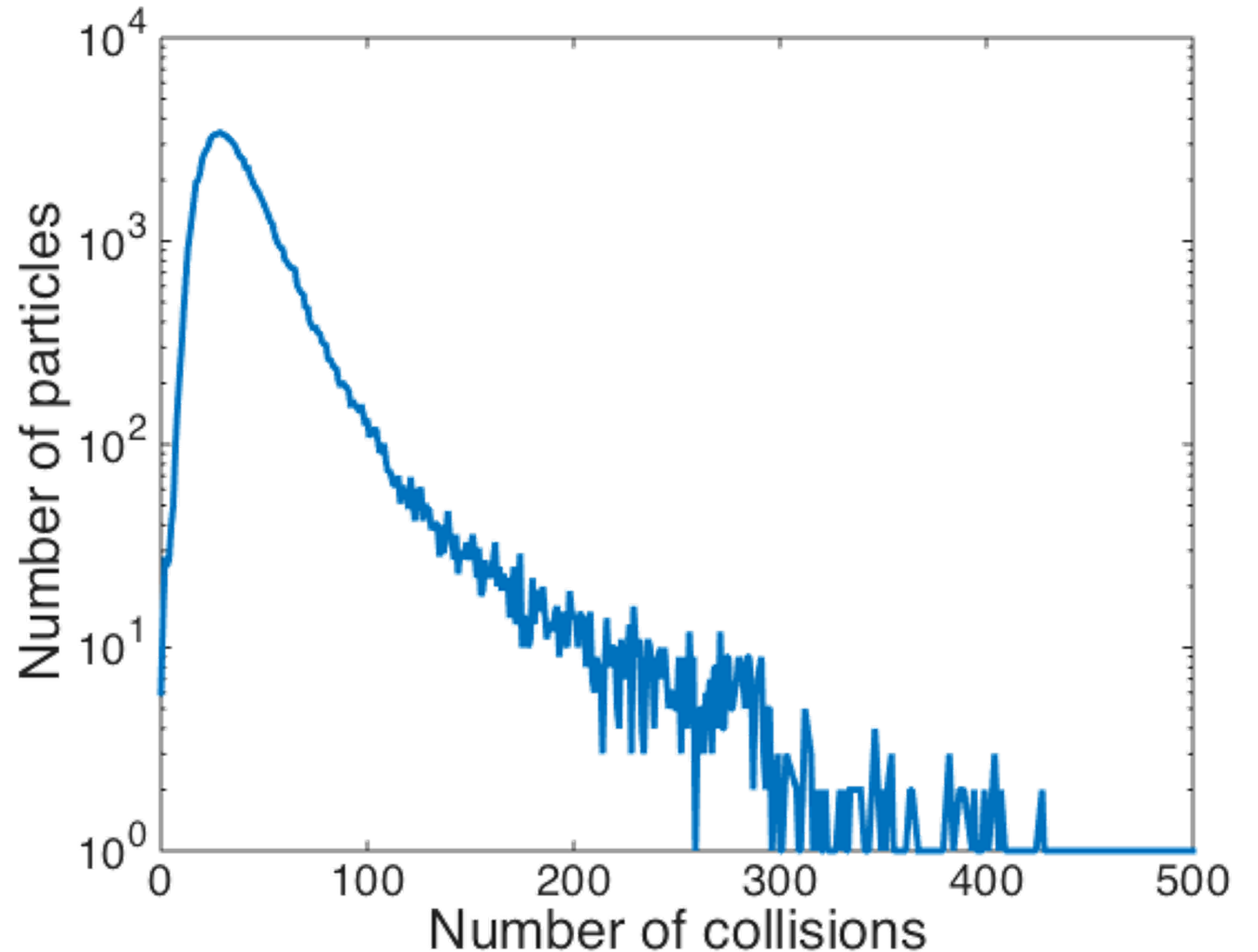
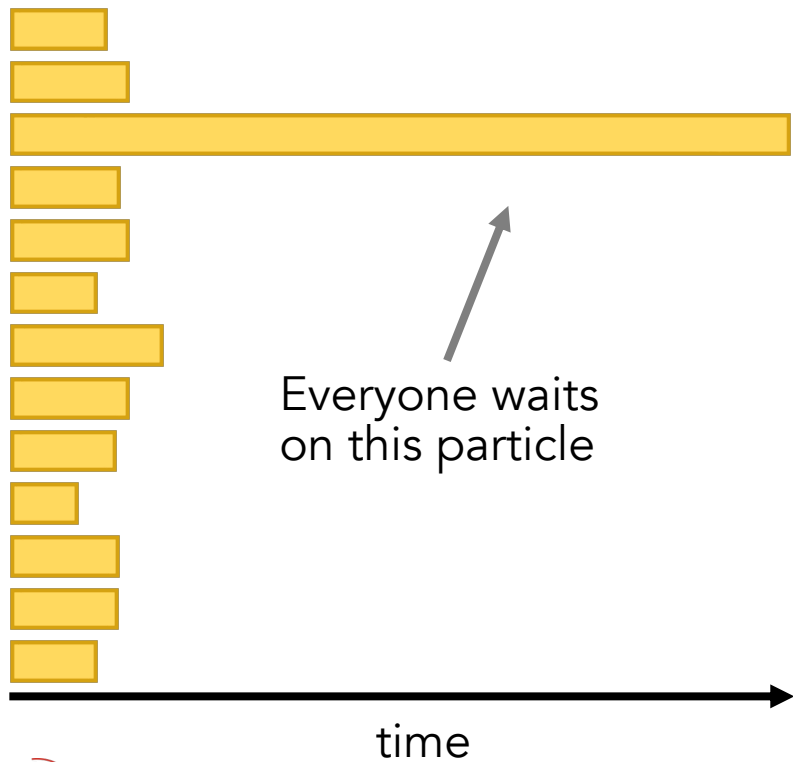


Reproduced with permission



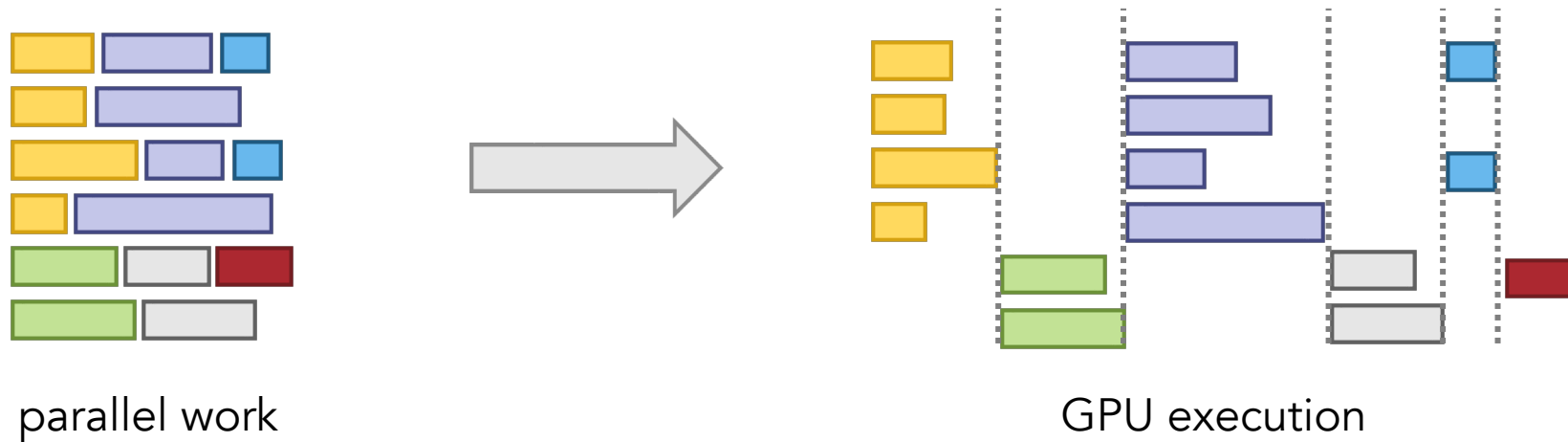
# Neutron transport: random particle statistics poorly suited to GPUs

- Stochastic history-based algorithm follows particles from birth to death.
- Most particles are short-lived, a few are not.



# Branching code is highly undesirable on SIMT architectures (GPUs)

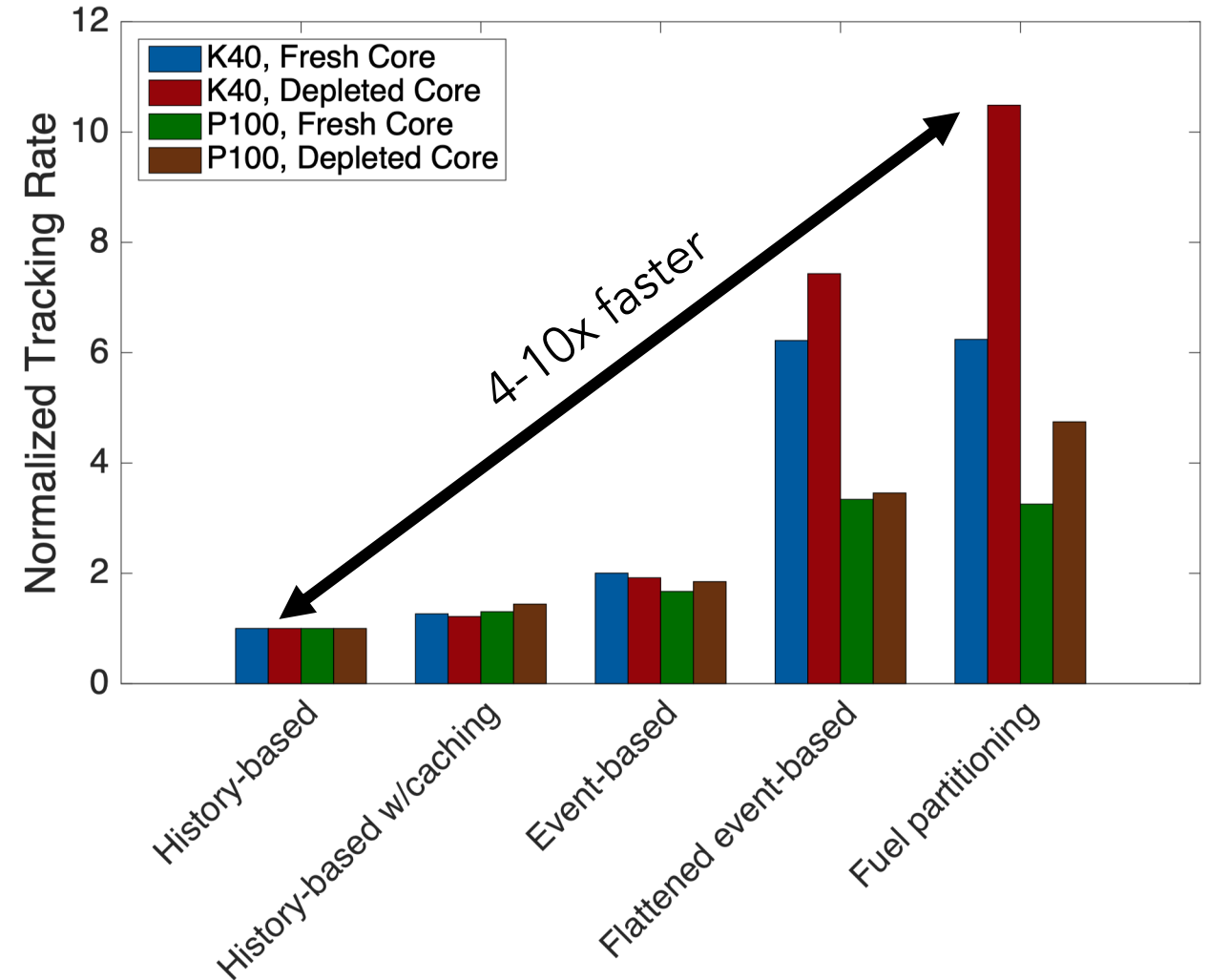
Even when each particle has roughly the same amount of work, **thread divergence** is a big problem when random sampling sends them down different code paths



Need to rethink code execution based on the target hardware. For example, parallelizing over **events** (i.e. common code paths) rather than particles.

# New event-based algorithm gave ExaSMR significant speedup

- Parallelizing over events is a much better match for a SIMT architecture than parallelizing over particles.
- Further improvements gained by identifying parts of the system that have significantly different behavior and separating them out.
- Smaller, focused kernels allow for better occupancy, i.e. more efficient use of the hardware



# Four key ingredients of an ECP Application Development Project



Science goal



Algorithmic  
innovation



Porting



Integration

# Porting must be done with hardware in mind

## Map algorithm to GPUs

- Rewrite, profile, and optimize
  - Generally preserve the exact answer
- Data Layout for memory coalescing
- Loop ordering
- Kernel flattening
- Increased locality
- Recomputing vs. storing
- Reduced branching
- Eliminating copies

## Map calculation to GPUs

- Reduced communication
- Reduced synchronization
- Increased parallelism
- Reduced precision
- Optimized linear algebra

## Identify opportunities for improvement

- Mathematical representation
- “On the fly” recomputing vs. lookup tables
- Prioritization of new physical models
- Alternate discretizations (high AI)
- Localized subgrid models
- Sparse → dense systems
- Defining weak scaling target
- Initial condition from ROM

Hardware has significant impact on all aspects of simulation strategy

# Choosing the right programming model is all about balancing trade-offs

## GPU-specific kernels

- Isolate the computationally-intensive parts of the code into CUDA/HIP/SYCL kernels.
- Refactoring the code to work well with the GPU is the majority of effort.

## Loop pragma models

- Offload loops to GPU with OpenMP or OpenACC.
- Most common portability strategy for Fortran codes.

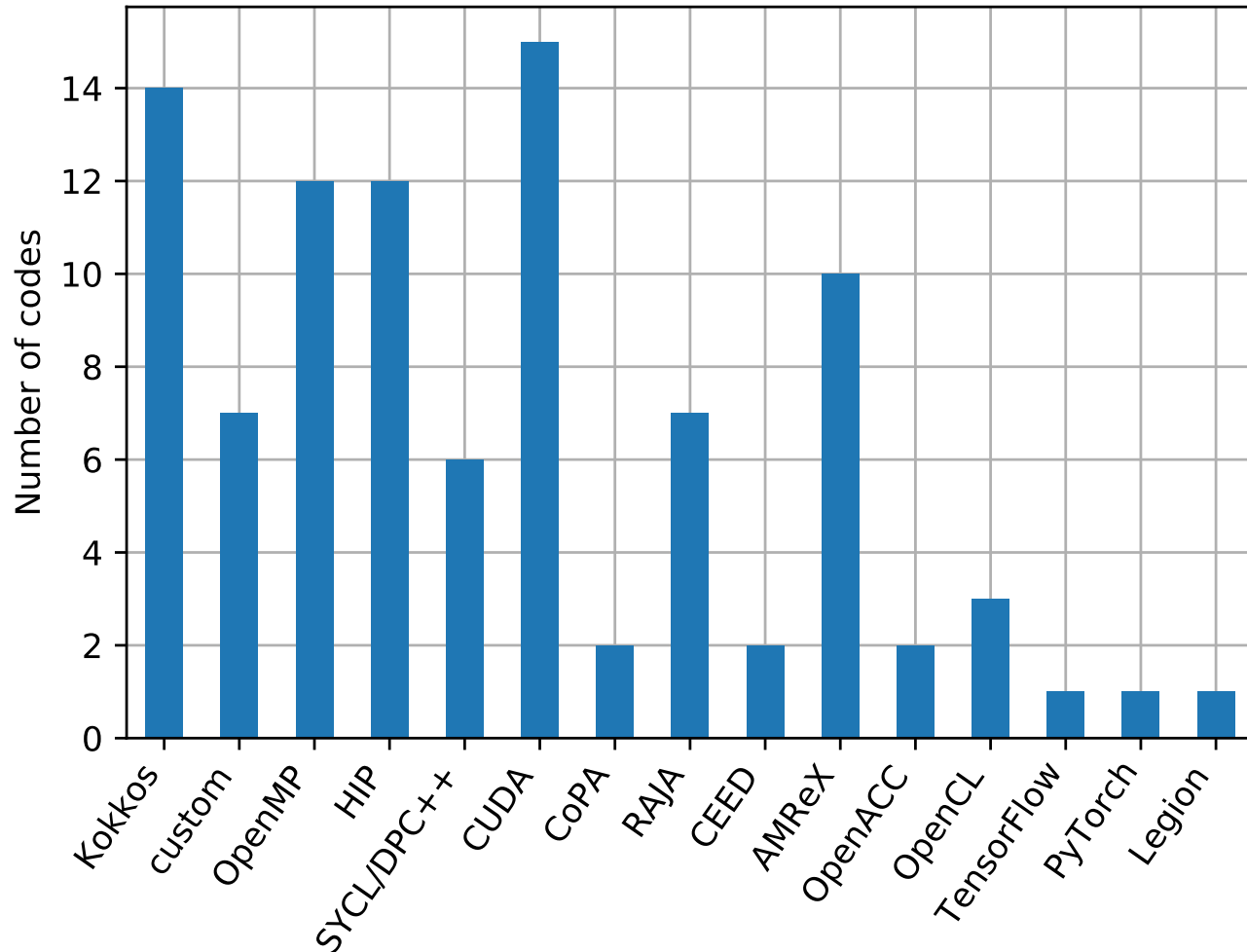
## C++ abstractions

- Fully abstract loop execution and data management using advanced C++ features.
- Kokkos and RAJA developed by NNSA in response to increasing hardware diversity.

## Co-design frameworks

- Design application with a specific motif to use common software components
- Depend on co-design code (e.g. CEED, AMReX) to implement key functions on GPU.

# Programming models used in ECP applications

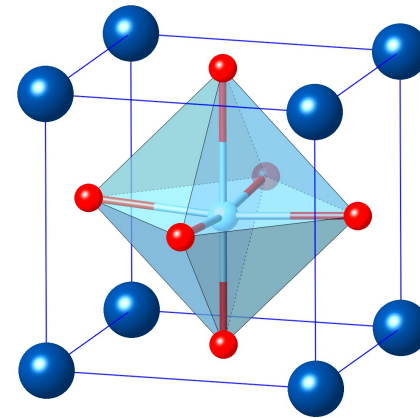
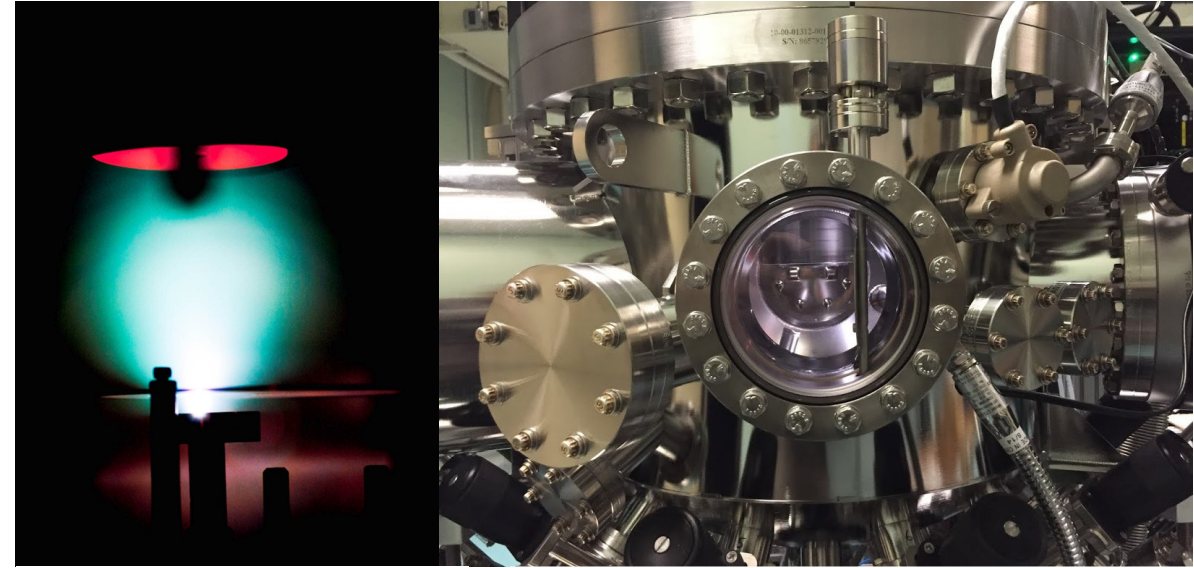


Platform portability provided by co-design projects (CoPA, CEED, AMReX)	33%
Native (CUDA/HIP/SYCL) or custom implementations	33%
ST programming models (Kokkos, RAJA, Legion)	18%
Directive-based programming models: (OpenMP, OpenACC)	16%

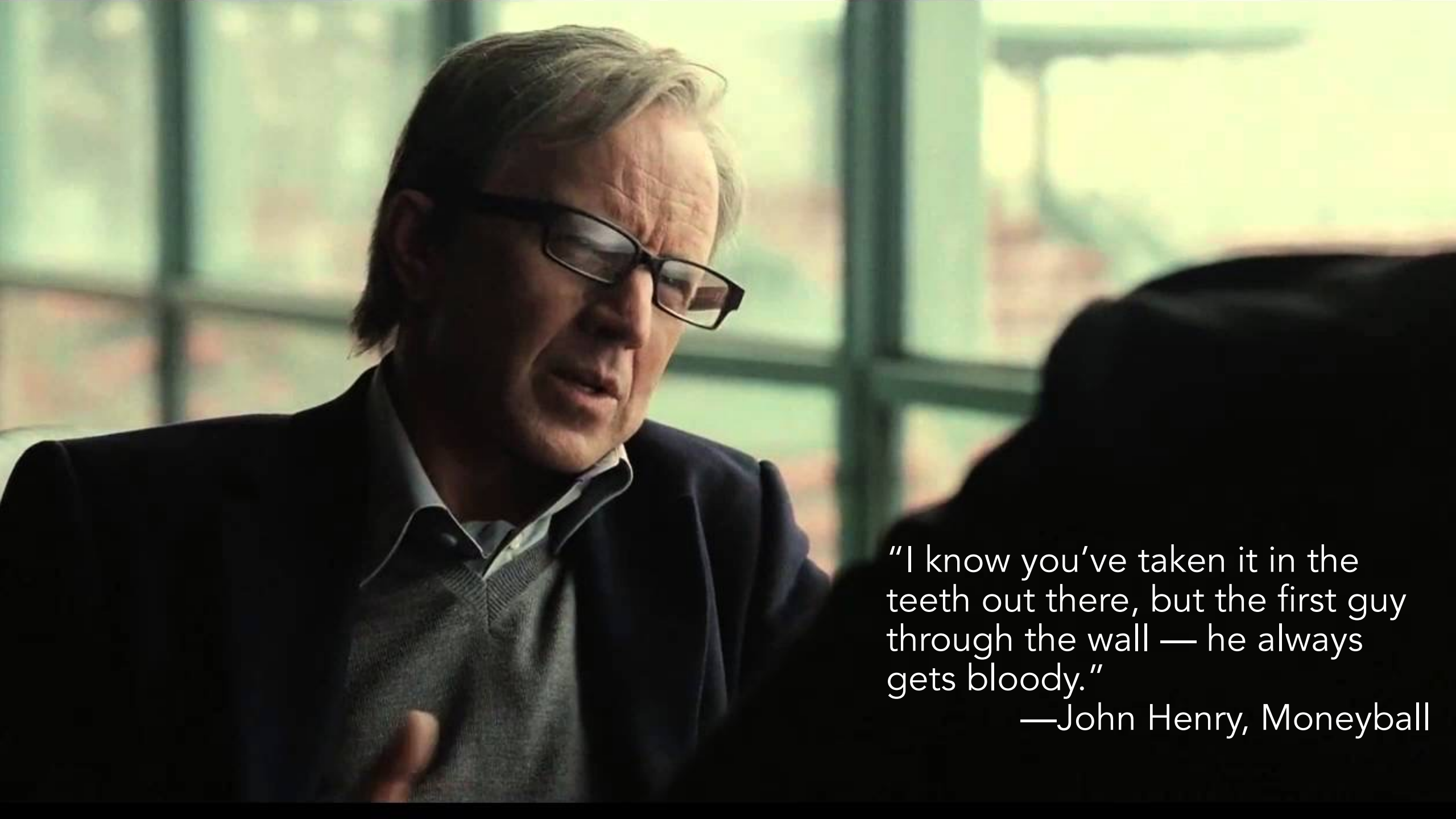
- Use of co-design/ST technologies provides significant benefit. Fine-scale architectural details provided by co-design technologies
- Large percent of custom implementations reflects difficulty of universal platform-portable programming models that span diverse apps

# Example: Quantum Monte Carlo for Materials

- To predict, understand, and design next generation materials requires reliable, non-empirical, atomistic quantum mechanics-based methods.
- ECP application QMCPACK implements multiple Quantum Monte Carlo (QMC) algorithms to achieve this. Primary focus for ECP is on the real-space diffusion Monte Carlo (DMC) and orbital space auxiliary field QMC (AFQMC) algorithms to enable cross-validation.
- OpenMP was selected as the GPU programming model to maximize future portability.





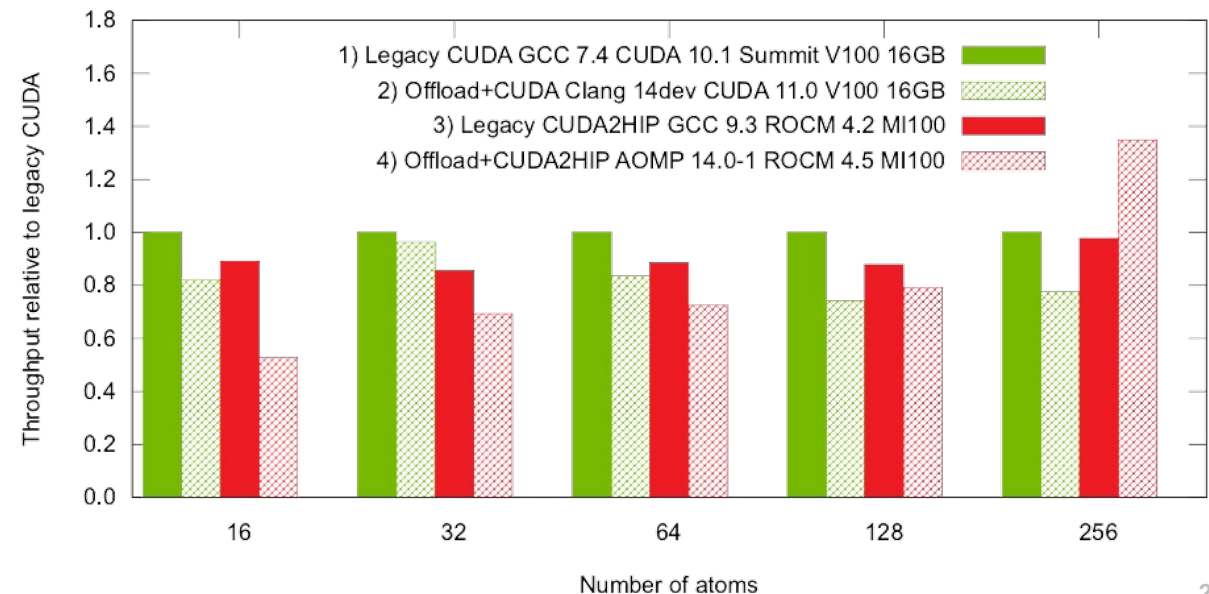
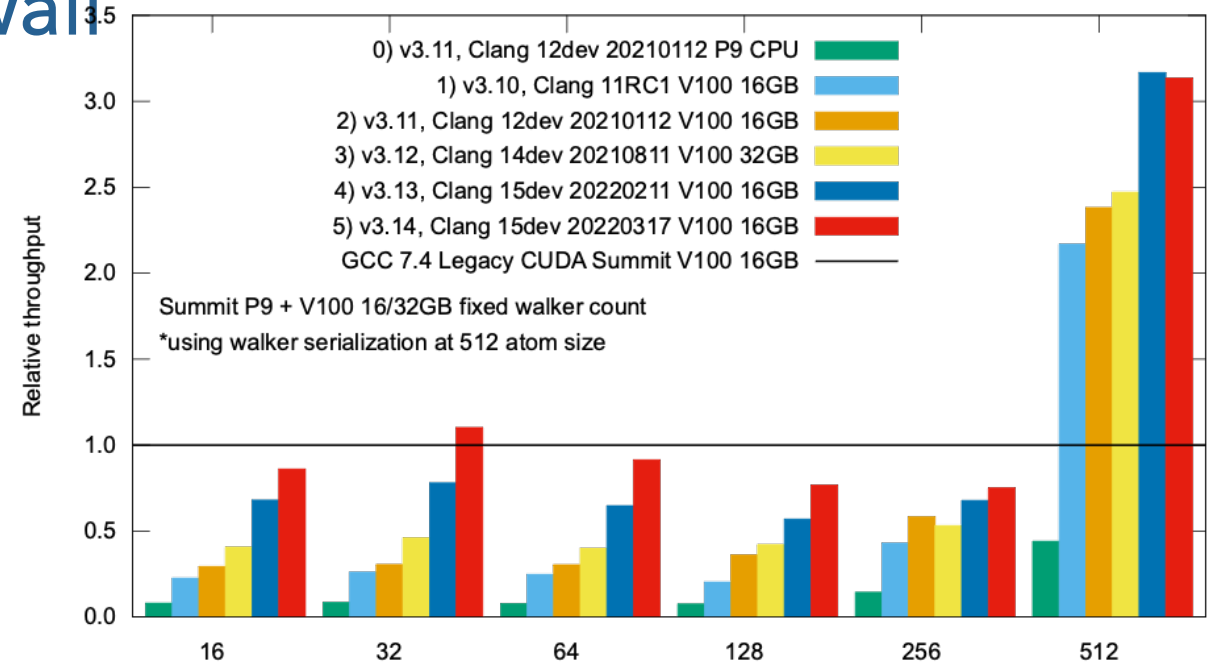


“I know you’ve taken it in the teeth out there, but the first guy through the wall — he always gets bloody.”

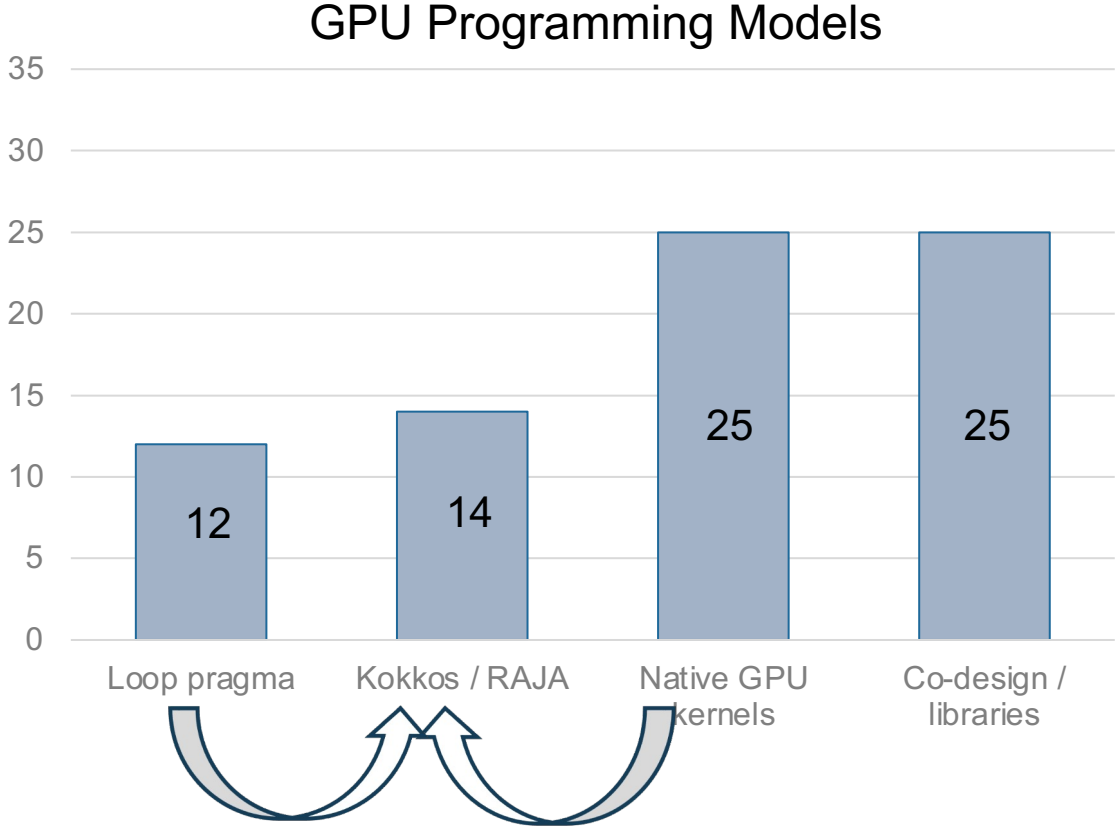
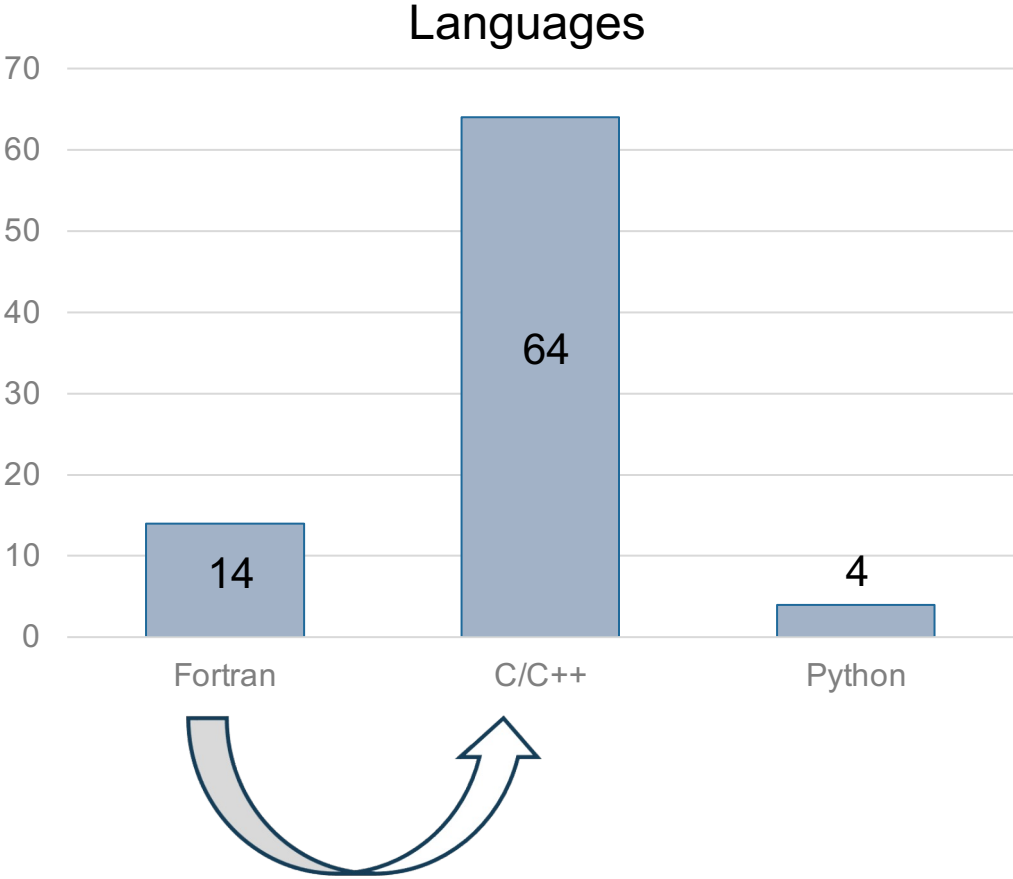
—John Henry, Moneyball

# QMCPACK was first through the wall

- QMCPACK had a working CUDA implementation of the code that proved invaluable in understanding where OpenMP performance was falling short.
- OpenMP offload runtimes are not yet consistently performant across vendors. Initial OpenMP results were significantly slower than CUDA.
- With careful performance analysis and by working closely with the vendors, the QMCPACK team was able to steadily improve performance of their OpenMP version until it is now on par with CUDA.



# Distribution of ECP programming models has changed over time



Programming language/model choices have evolved over course of ECP

# Four key ingredients of an ECP Application Development Project



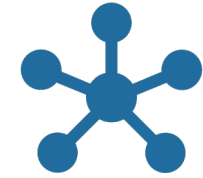
Science goal



Algorithmic innovation

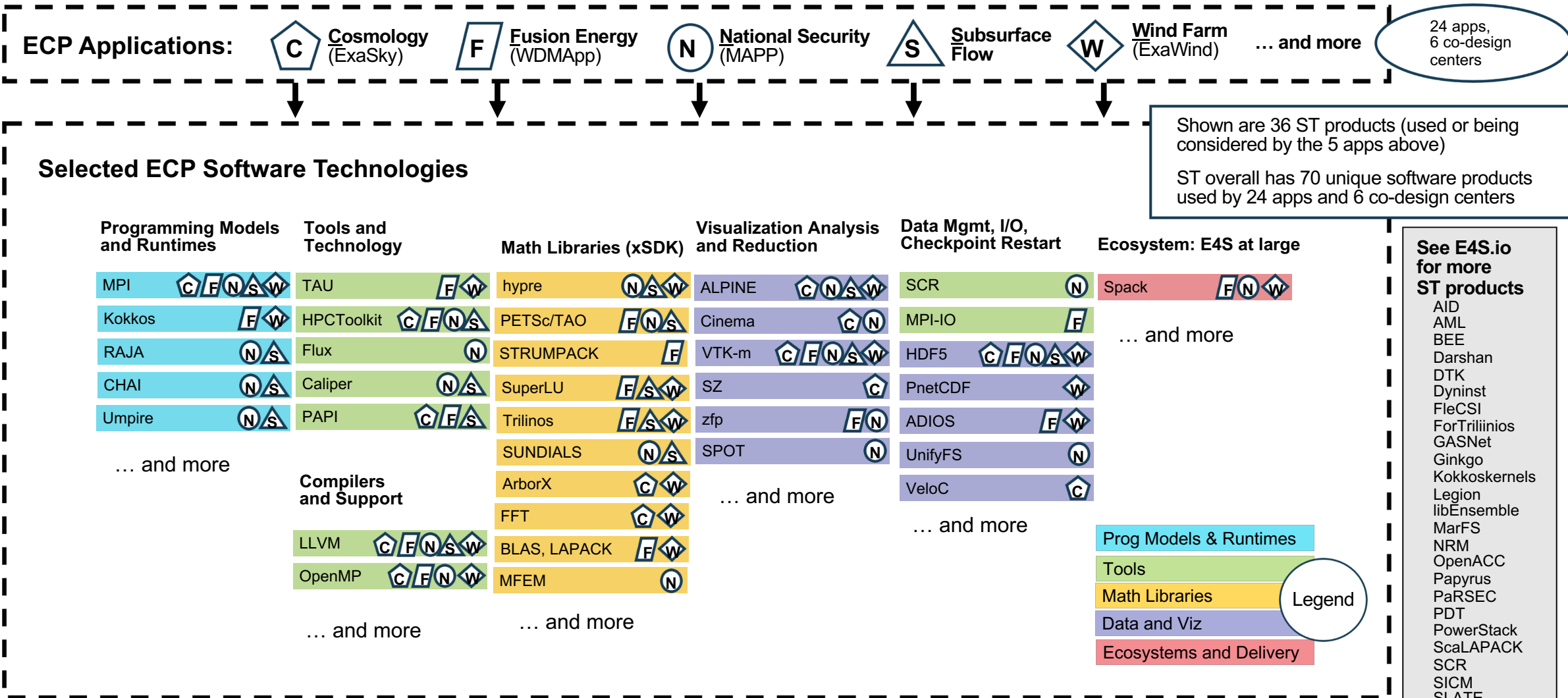


Porting




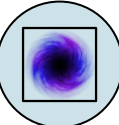







Integration

# Integration: ECP applications rely heavily on high quality software tools and libraries



# Extreme-scale Scientific Software Stack (E4S)

- E4S: HPC software ecosystem – a curated software portfolio
- A **Spack-based** distribution of software tested for interoperability and portability to multiple architectures
- Available from **source, containers, cloud, binary caches**
- Leverages and enhances SDK interoperability thrust
- Not a commercial product – an open resource for all
- Growing functionality: Nov 2021: E4S 21.11 – 91 full release products

 Community Policies Commitment to software quality	 DocPortal Single portal to all E4S product info	 Portfolio testing Especially leadership platforms
 Curated collection The end of dependency hell	 Quarterly releases Release 1.2 – November	 Build caches 10X build time improvement
 Turnkey stack A new user experience	 <a href="https://e4s.io">https://e4s.io</a>	 E4S Strategy Group US agencies, industry, international



<https://spack.io>

Spack lead: Todd Gamblin (LLNL)



<https://e4s.io>

E4S lead: Sameer Shende (U Oregon)



Also includes other products, e.g.,  
**AI:** PyTorch, TensorFlow, Horovod  
**Co-Design:** AMReX, Cabana, MFEM

# Final thoughts

- This is an exciting and terrifying time to be doing computational science.
- Those who take the time to understand the hardware they are running on and/or coding for will have a major advantage over those who try to use past practices blindly.
- For computational capabilities, don't reinvent the wheel! Build on the successes of others whenever possible.
- For applied math, re-examine and question everything! Many best practices are based on assumptions from the past that no longer apply. There are many opportunities for innovation.



LLNL-PRES-852697

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

**Disclaimer**

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.