Hardware Evaluation: Abstract Machine Models

SAND2018-1240PE

Jeanine Cook, Sandia National Laboratory

David Donofrio, Lawrence Berkeley National Laboratory

Ray Bair, Argonne National Laboratory

Jeff Kuehn, Los Alamos National Laboratory

Shirley Moore, Oak Ridge National Laboratory

Guest Speaker: Anshu Dubey, Argonne National Laboratory

Knoxville, TN
February 2018

EXASCALE COMPUTING PROJECT

Abstract Machine Model (AMM) Team

Team:

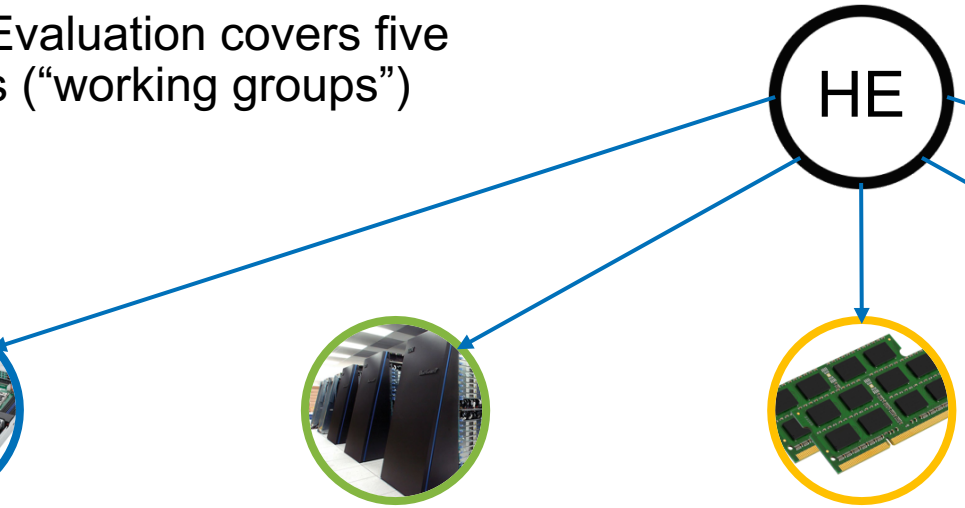
- Dave Donofrio, Phil Colella (LBNL)
- Jeanine Cook (SNL)
- Shirley Moore, Jeff Vetter (ORNL)
- Andrew Chien, Ray Bair (ANL)
- Jeff Kuehn (LANL)

Contributors:

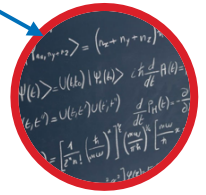
- Scott Pakin, Gary Grider (LANL)
- John Shalf, Taylor Groves, George Michelogiannakis (LBNL)

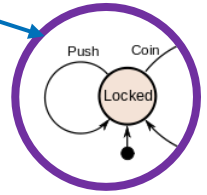
Hardware Evaluation - Overview

Hardware Evaluation covers five key areas (“working groups”)



Outreach/cross-project responsibilities here





Node-Level Simulation

Processor, Pipelines, Threading, Caches, Coherency, Network-on-Chip, Network Interfaces

Interconnect Modeling

Network Topologies, Congestion, Quality-of-Service, Silicon Photonics

Memory Technologies

Memory Media, Parallelism, Controllers, Non-Volatile, Coherency, Scratchpads

Analytic Modeling

High-Level Architecture Balance, “Ops-to-Bytes”, Instruction Mix, Branching, Vectorization

Abstract Machines and High-Level System Models

Outreach, Cross-ECP Communication, Non-NDA Models, High-Level Descriptions

Hardware Evaluation - Operation

- Each working group (except Abstract Machine Models) is responsible for developing analyses of “questions” that are important to the following:
  - DOE Leadership Computing Facilities fielding Exascale-systems
  - DOE Procurement Activities/Teams
  - ECP PathForward Vendor or DOE Technical Representatives

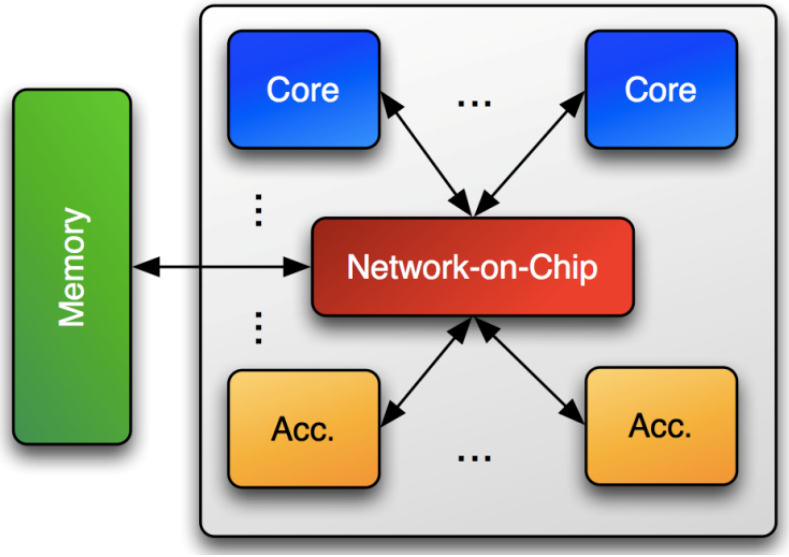
- **Example:** “what would the performance impact be on the A21 machine if only HBM-2 is available instead of HBM-3?”
  - Node Simulation, Analytical Modeling and Memory Technologies working groups would prepare models of ECP kernels/benchmarks and evaluate the trade off

Hardware Evaluation - Outreach

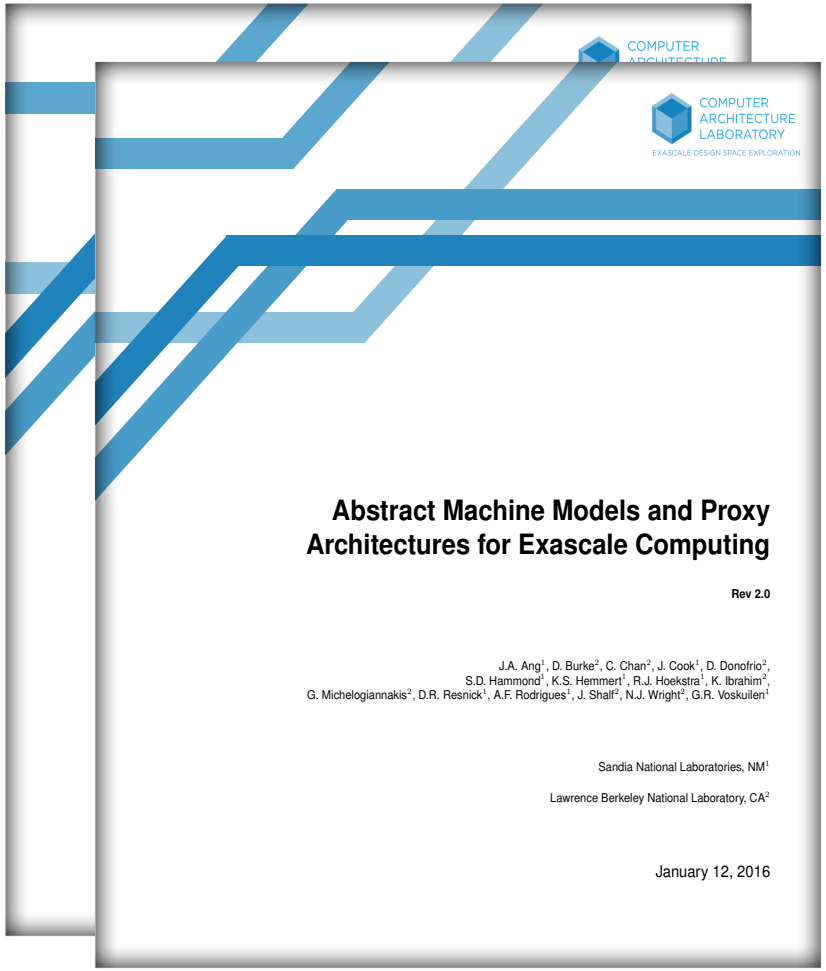
- Abstract Machine and High-Level System Models are the HE's primary outreach vehicles
  - ECP Application and Software Technology Teams
  - Broader DOE engagement
  - Possibly non-NDA models of some architectures
- Support high-level description of Exascale-class systems to enable better communication across the project and DOE
- Act as a conduit to H&I/HE within ECP
- Collaboration point for ECP Proxy Apps and Application Assessment projects

What is an Abstract Machine Model?

- Simplified abstractions that describe the organization and design of node and system architectures
- **Abstract Machine Models (AMMs)** represent minimum aspects of hardware architectures that are important / relevant for Application and Software Development

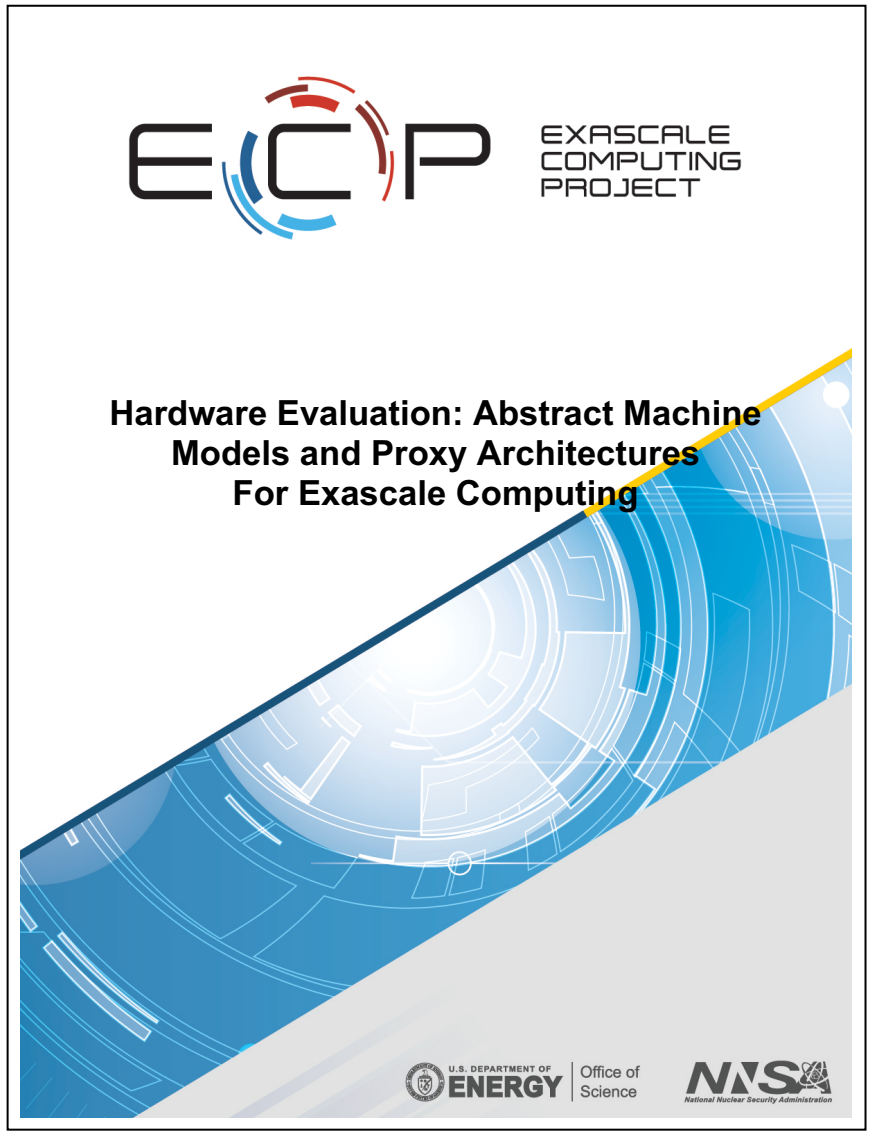


Previous AMM Documents



- Computer Architecture Lab (CAL) released two revisions
  - Rev 1: May 2014
  - Rev 2: Jan 2016
  - ASCR Funded collaboration between LBNL and SNL
- A view of exascale prior to the start of ECP
  - Some trends have shifted since publication, so we have created a new document…

Current AMM Document



- Reflects latest in known hardware / system trends for exascale
  - Targeting 2021/2023
- Created by HE Team in collaboration with external stakeholders
- Downloadable from the AMM WG homepage
  - https://confluence.exascaleproject.org/display/HTDSEW03

Why do we Need AMMs?

- HPC Systems are complex
- AMMs act as a communication vehicle between hardware architects / vendors and developers
  - Common language
  - Communicate system trends
- Allow focus on aspects of a machine that are important for
  - Performance
  - Portability
- Intention is to have a NDA-free models

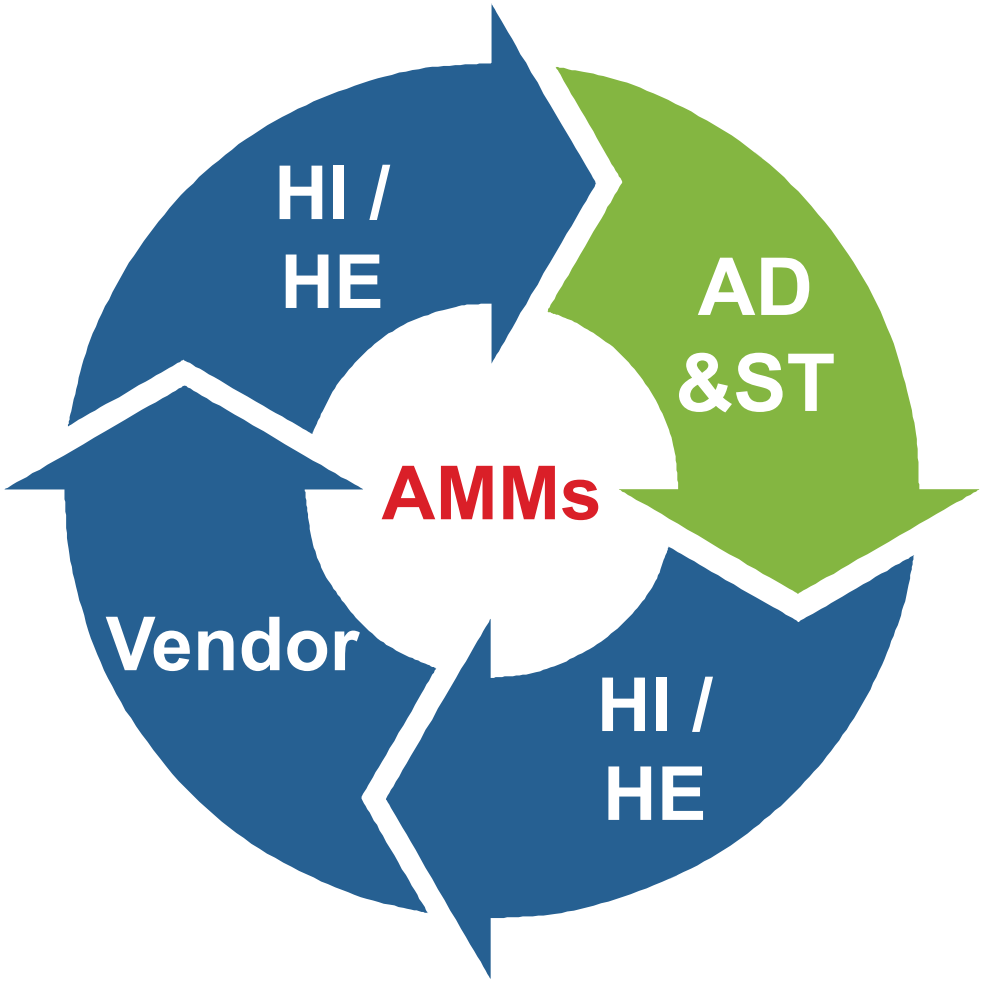
What is a Proxy Architecture?

When you want to get specific

- Abstract models are notional
  - Tells you about how resources may be organized, but not how many there are of each
- Proxy architecture provide concrete parametrizations
  - Capacities, bandwidths, latencies, etc
- Useful for guiding:
  - Design exploration
  - Determining impact on your code!

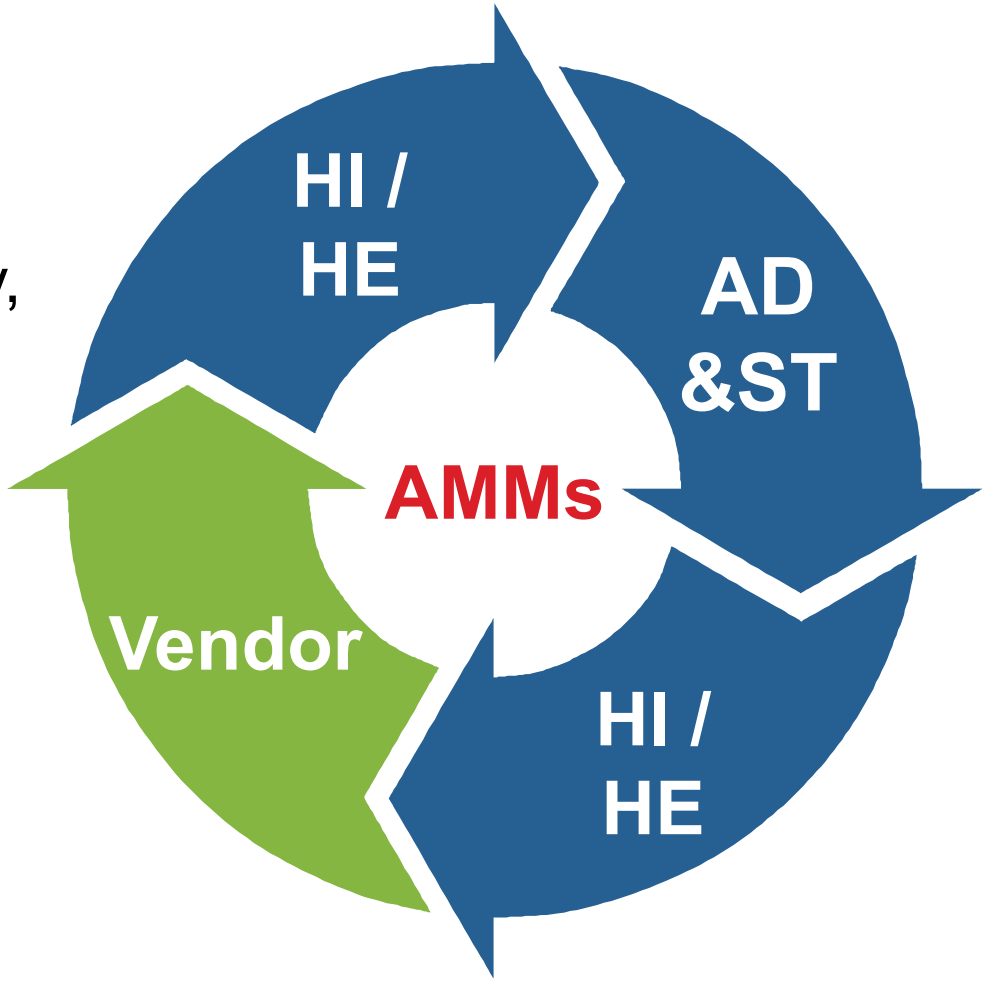
What do AMMs Hope to Clarify/Achieve? (1)

- Application Code Structure and Performance:
  - Expose the high-level structure of a proposed machine to permit reasoning about:
    - Abstract design of algorithms to leverage new hardware features for enhanced performance and/or power efficiency
    - Structure (or restructure) of DOE simulations that would be required for the proposed architecture,
    - Performance impact of various hardware options on a particular application
    - Impact of the above changes on the generality and portability of a particular application



What do AMMs Hope to Clarify/Achieve? (2)

- Impact of Proposed Hardware Changes:
  - Support feedback from DOE to the vendors to suggest changes or modifications to the proposed hardware to enhance programmability, usability, and performance
- Changes of interest include:
  - Reduction of programming burden and improvement in usability
  - Improvement in application portability
  - Enhancements to system software's ability to abstract features on behalf of the application
  - Enhancements to performance or power efficiency of applications

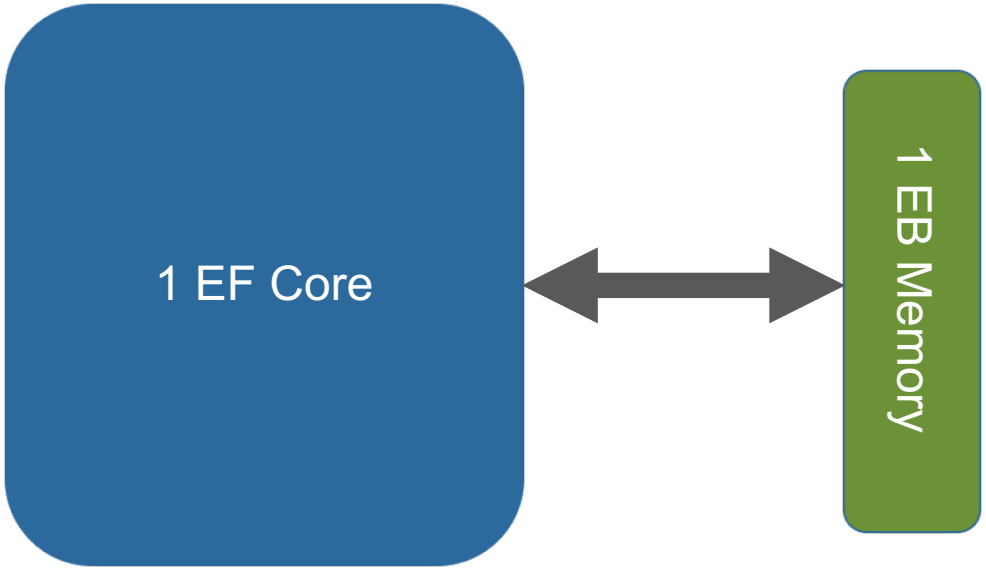


AMMs are useful but you must profile your application to make the most use of them

The real question: *”What are the Exascale systems going to look like?”*

“… and what do I need to do to prepare?”

Ideal (for some) Exascale System



This is definitely ***NOT*** what the Exascale System will look like!

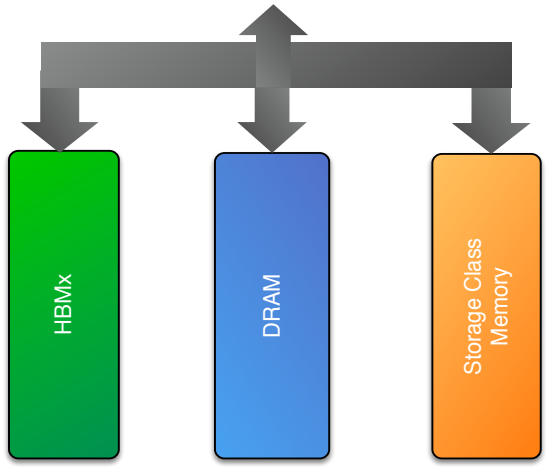
System Trends for Exascale

Back to reality…. We'll visit each of these in more detail

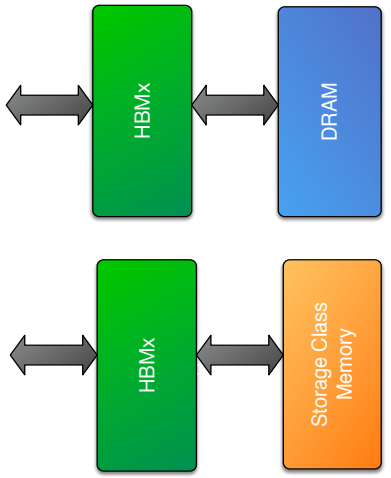
- *Processor / Compute Nodes*: Heterogeneity and acceleration are the key to extracting performance
- *Memory*: Hierarchy will likely look familiar, but technologies may not, DDR may get squeezed out by other memory technologies
- *Interconnect*: Node count and organization (including memory capacity) may amplify the importance of the interconnect
- *Storage*: Non-volatile technologies augmenting the hierarchy
- ***There is some good news too!***

Memory AMMs

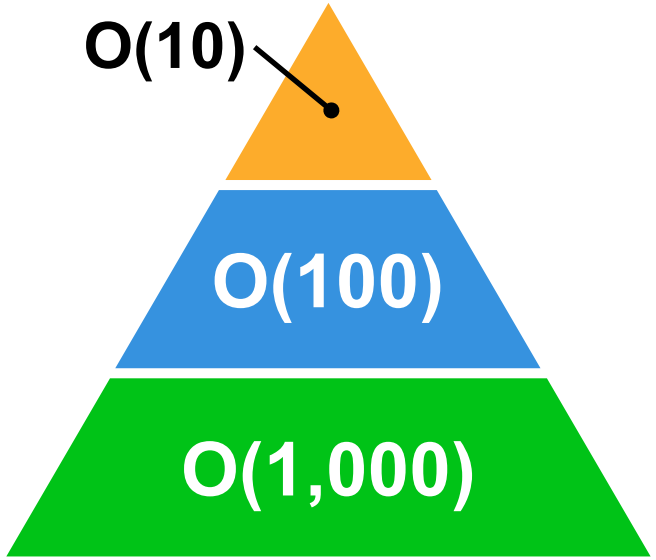
- HBM2/3 means more BW
  - Capacity challenges…
- Fundamental tradeoff of capacity, vs. performance
  - Drives selection of memory technology for a particular level in the hierarchy
- Performance differences in memory levels may require explicit data object placement
- KNL / GPU current examples of this tradeoff



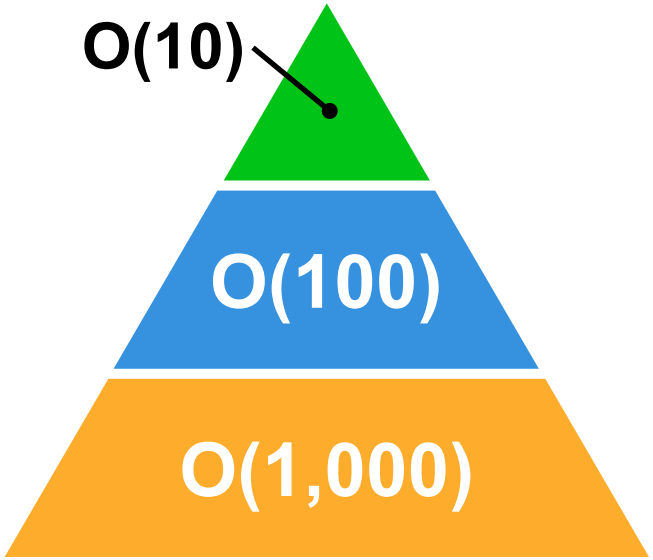
(a) One-Level (non-Cached) Memory System; One Physical Address Region



(b) Two-Level Cached Memory System



Bandwidth



Capacity

Processor AMMs

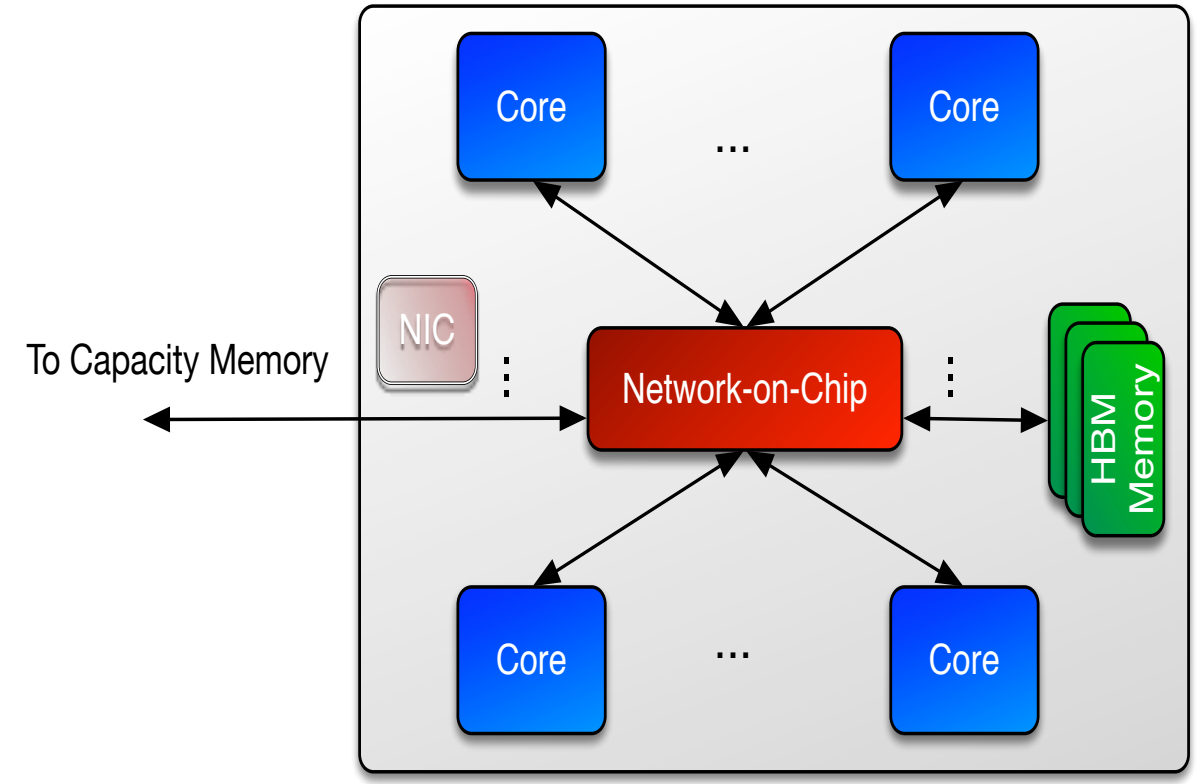
What can we expect in the exascale timeframe?

- Comparable GP core counts / node to contemporary systems O(48-64) cores
- Accelerators will have tighter integration on the node
  - Good news: Shorter latency, higher BW
  - Actionable news: Continued focus on data placement as NUMA domain count likely to increase
- Accelerators may, or may not, have an ISA independent of the GP core
  - Similar to GPU / CPU split today
- Accelerators could be a mix of function/application specific and general purpose
- Vector / SIMD widths growing
  - … but ISAs getting more capable!
- As exascale gets closer options for AMMs will narrow
- *In general, most of the computing elements found in a node will appear familiar, however, the organization may be different. But accelerators are **key** for performance.*

Processor AMMs – Homogenous

What can we expect in the exascale timeframe?

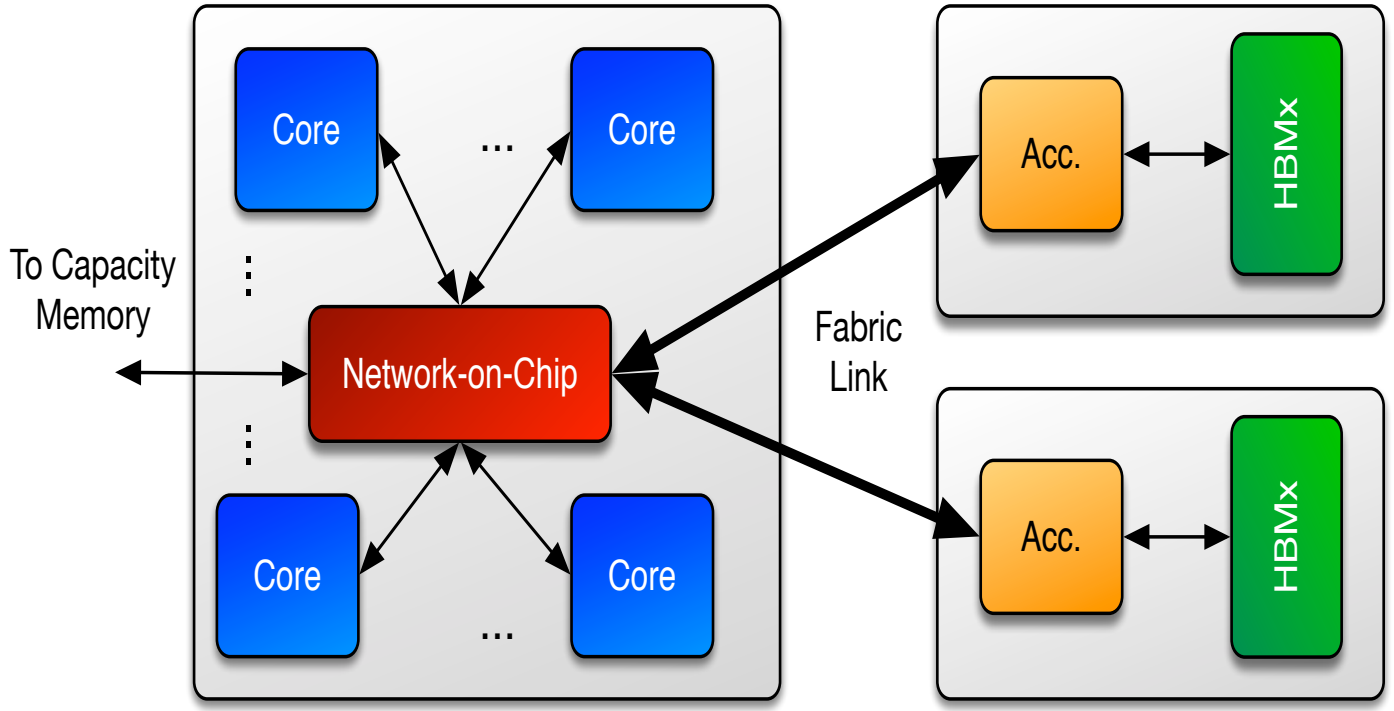
- Defining Characteristics:
  - Memory access is uniform from core's perspective
    - Mem access latency could vary depending on NoC dimensions
  - All cores are uniform
  - Familiar programming models able to target this AMM
- Examples include
  - Intel KNL, Skylake
  - ARM (Thunder X or similar)
  - AMD EPYC
  - IBM Power9



Processor AMMs – Heterogeneous Option 1

What can we expect in the exascale timeframe?

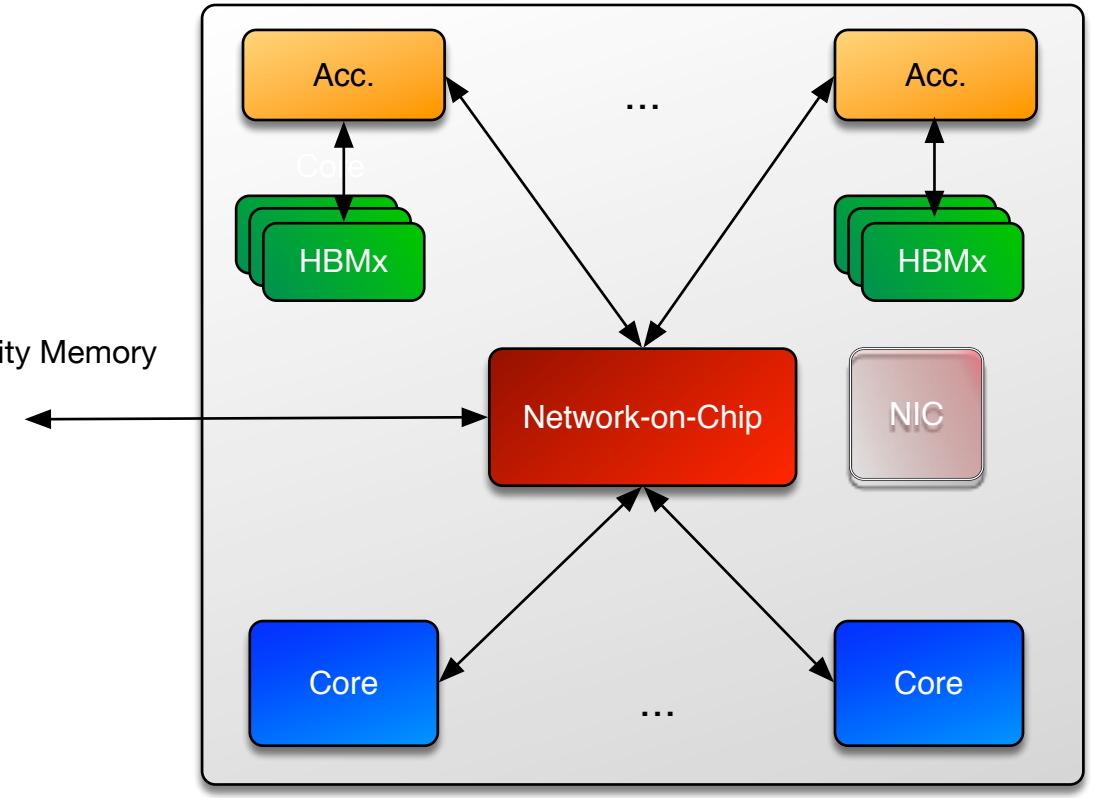
- Defining Characteristics:
  - Accelerator has different ISA from cores
  - Accelerator in separate memory space
    - Likely not coherent with cores
  - Accelerator accesses network through host CPU
  - Relatively high cost to move data from host CPU to accelerator
  - Familiar “offload” programming models able to target this AMM
  - Host CPU described by Homogenous AMM
- Summit and Sierra are examples



Processor AMMs - Heterogeneous Option 2

What can we expect in the exascale timeframe?

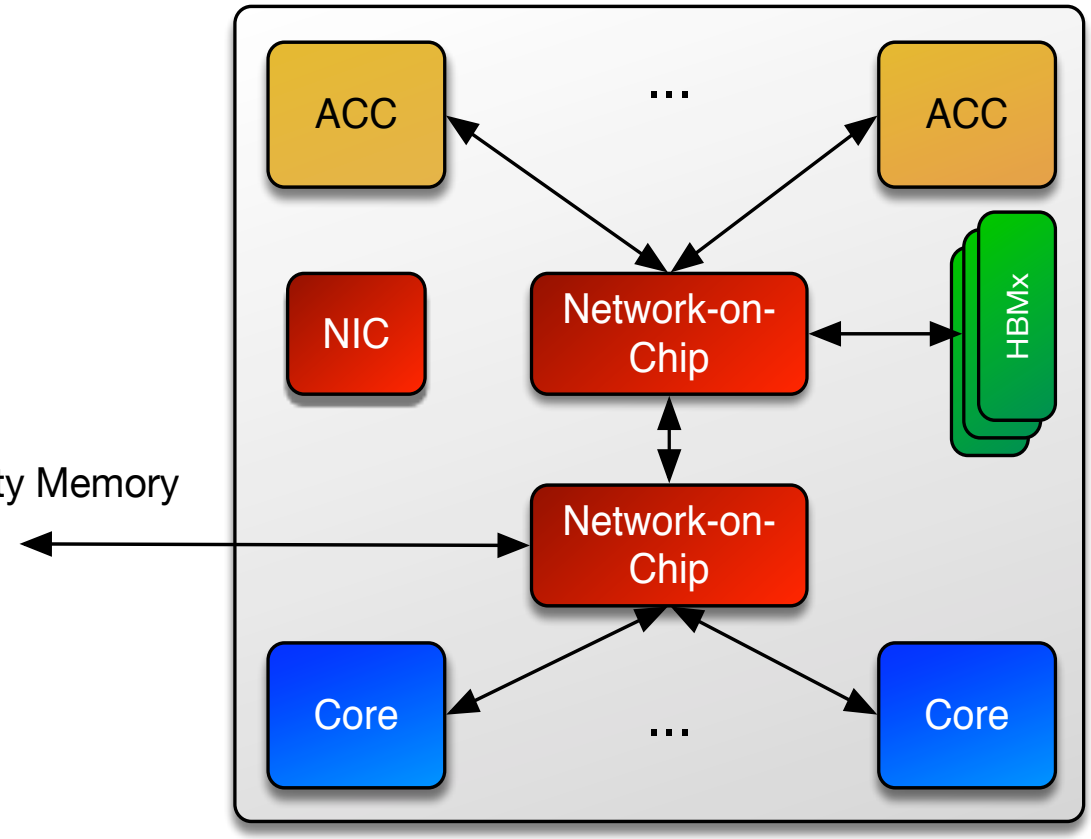
- Defining Characteristics:
  - Similar to Option 1, but with better integratio so less penalty for offload
  - Accelerators are homogenous but have different ISA from cores
  - Accelerator access to system interconnect is architecture dependent
  - Greater emphasis on using accelerators to obtain performance
  - Cores access HBM through the accelerators
    - Accelerators have direct access
  - Familiar “offload” programming models able target this AMM
- AMD’s APU and Roadrunner are examples



Processor AMMs - Heterogeneous Long Term

What can we expect in the exascale timeframe?

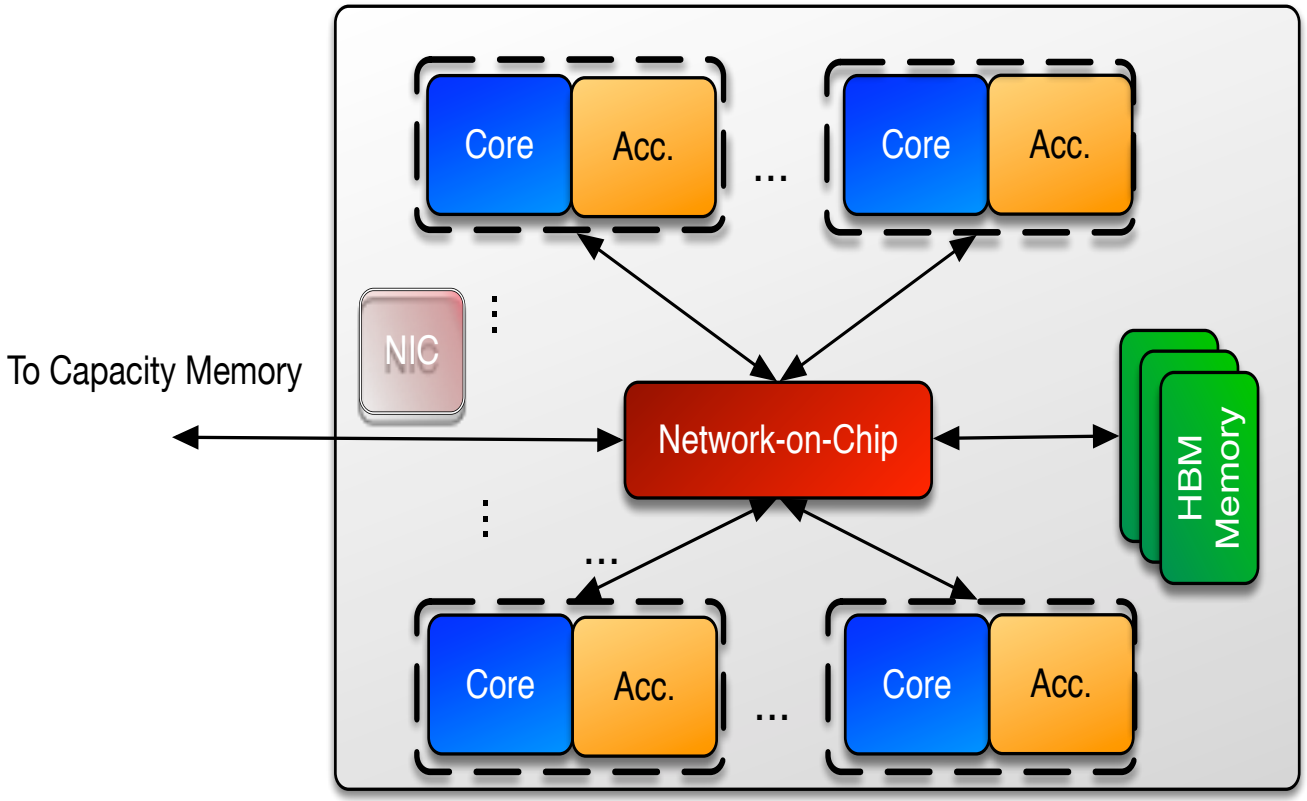
- Defining Characteristics:
  - Similar to Option 1 and 2,
  - Cores AND accelerators have direct access to HBM through the accelerators
  - Further out in time than Option 2, possible direction



Processor AMMs – Tightly Integrated

What can we expect in the exascale timeframe?

- Defining Characteristics:
  - Accelerator and core now tightly integrated in both space and memory access
    - Shared cache hierarchy
  - Offload to accelerator very low cost
  - Accelerator and cores may share a sin ISA
  - Familiar programming models likely wc well
- Very wide vector machines (ARM SVE, AVX512) are examples



Programming Implications

Advances in programmability

- Increasing importance and prevalence of accelerators
  - Can be a challenge for programmability and portability
- Ease of accelerator programming increasing through:
  - Compiler advances
  - Programming model support
  - OS / Runtime support

Interconnect Exascale trends

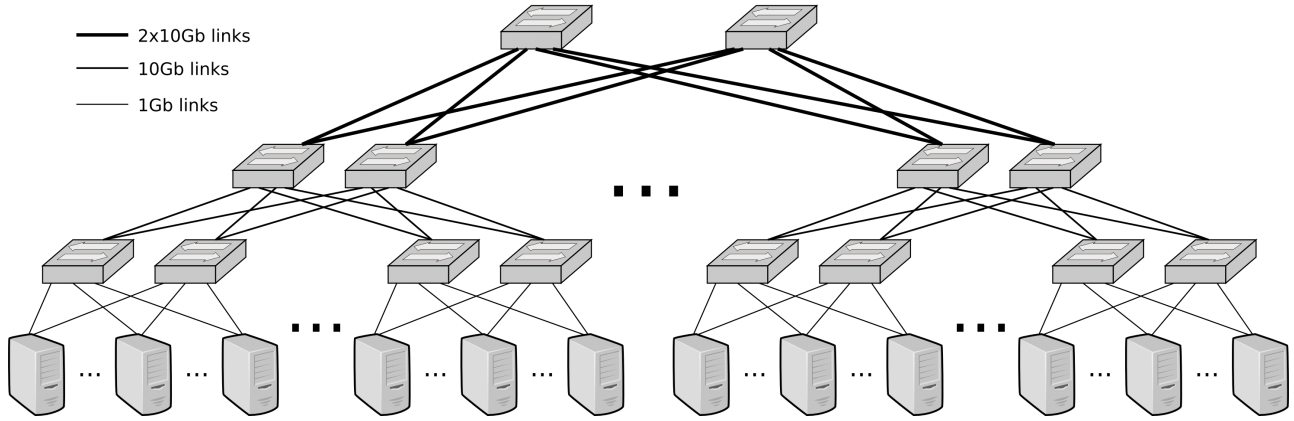
- Exascale (and other HPC) systems have many interconnects, each with its own topology and performance characteristics
  - **On Chip** – among cores and other functional units, often NUMA
  - **Intra-Node** – connecting sockets, accelerators, and NIC(s)
  - **Intra-Group** – connecting “close” nodes, e.g., nodes sharing a rack
  - **Global Networks** – connecting groups across the system, possibly with bandwidth tapering

Implications of Heterogeneous Networks

- Each level of network interfaces with the next, with differing levels of coherency and intelligence, e.g., atomics or tag matching
- Global network design weighs performance and capability against the number of optical cables (cost). Many are Dragonfly variants. Bandwidth is often hierarchical.
- Adaptive routing yields reduced network contention (overall), at the cost of variability in network performance.
- Placement of the nodes in your job can also impact variability, e.g., the pathways to I/O, burst-buffer, or service nodes can vary. Your paths may be shared with other jobs.

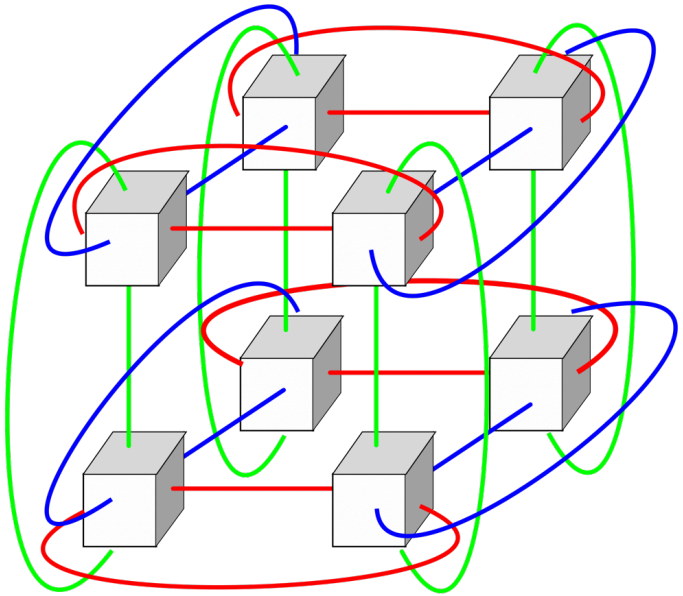
Global Network Topologies

- 3D Torus
  - Many links. Expensive to implement with optical cables
- Fat Tree
  - Tapered bandwidth (less links) as you go up the hierarchy
  - End point counts in the 1,000s
- Dragonfly
  - Number of links and routers increase linearly with system size
  - End point counts in the 10,000s

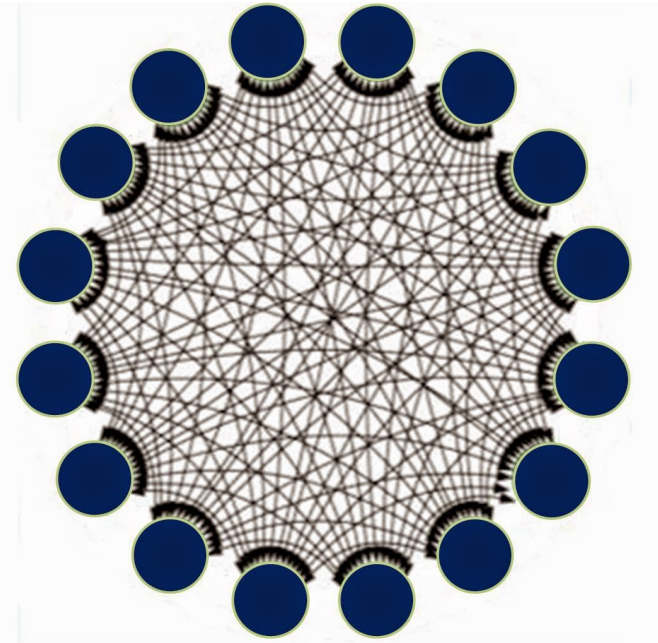


Fat Tree

3D Torus



Dragonfly



Strategies (1)

- On Chip and On Node – shared memory space
  - Manage locality to adapt to non-uniform communications
- Intra-Group – for rack-level tasks
  - Same techniques as global networks, with better performance characteristics intra-group

Strategies (2)

- Global Network
  - Most apps will design for the global network, because computations span many groups
  - Design for communications variability, it is inescapable at exascale
  - Avoid creating extra fabric contention when possible, e.g., separating I/O phases from communication phases
  - Understand the software overhead of your communication functions, and select lower overhead functions or programming models to reduce penalty from lightweight cores

Storage Technology Drivers

- Ubiquity of Flash
  - Bandwidth and IOPS is rapidly moving toward more use
  - Variety of device types not durable to reasonably durable
  - Will likely see entire scratch file systems on Flash by 2020-2025
- Emergence of higher tier solid state devices
  - 3DCrosspoint and other technology starting to push Flash at the high end, pushing Flash vendors to consider more and more markets even talking about pure capacity plays
- NVME and NVME over Fabric
  - Native on our fabrics global addressable block devices
  - Latency to remote storage in the microseconds
  - Making moves toward other non block interfaces like Key Value Stores

Tiers and Need for Shallower Stacks

- Tiers and their Implications
  - Economic drives will likely always require a tiered approach
  - Over time, the number of economic tiers may change between 2 and 5
  - API's are in turmoil for dealing with tiers (Lang's lema: more tiers=more tears)
    - Some claim a single byte addressable api is desirable but history shows us that byte addressable api's make users/apps expect a service level that may or may not be possible
- Need for Shallower Stacks
  - The advent of nearly ubiquitous globally addressable low latency storage in the microsecond level is problematic when much of the storage stack was built when latencies to disk devices was in milliseconds at best.
  - This will likely drive more of the solutions towards user space, which has security implications but does provide new dimensions of performance scalability, and "right sizing" of the storage resources

Non-file-block use and More Metadata

- Use of other than file and block
  - The block and file paradigms can not effectively exploit emerging devices that allow extremely low latency access to very small data sizes and devices that might be able to read in multiple dimensions, etc.
  - KVS access looks very promising for multi-dimensional data access methods
  - Producer/Consumer models are likely better serviced with non block/file access methods
- More Metadata Please
  - The growth of data will require more metadata to enable the needed provenance and manageability required for science.
  - The growth in metadata will very likely not be proportional to the data growth but instead will likely grow at a much faster rate than the data growth due to desire to enable more science and collection methods may end up being enabled by run time systems that will have far more metadata available

Example use case of AMMs

Using Abstract Machine Model for Software Architecture Design

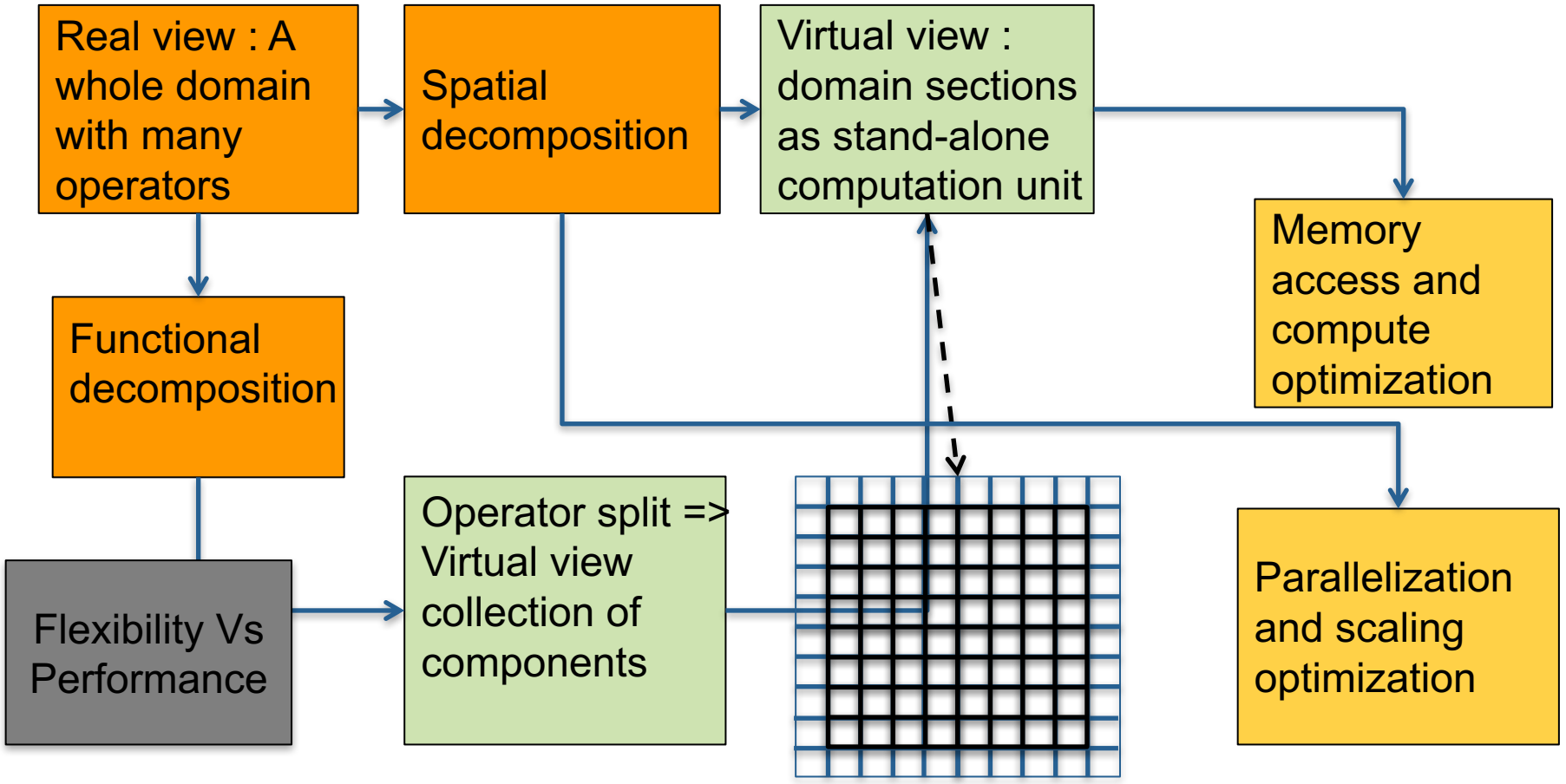
Anshu Dubey

Knoxville, TN
February 2018

Driving Principles

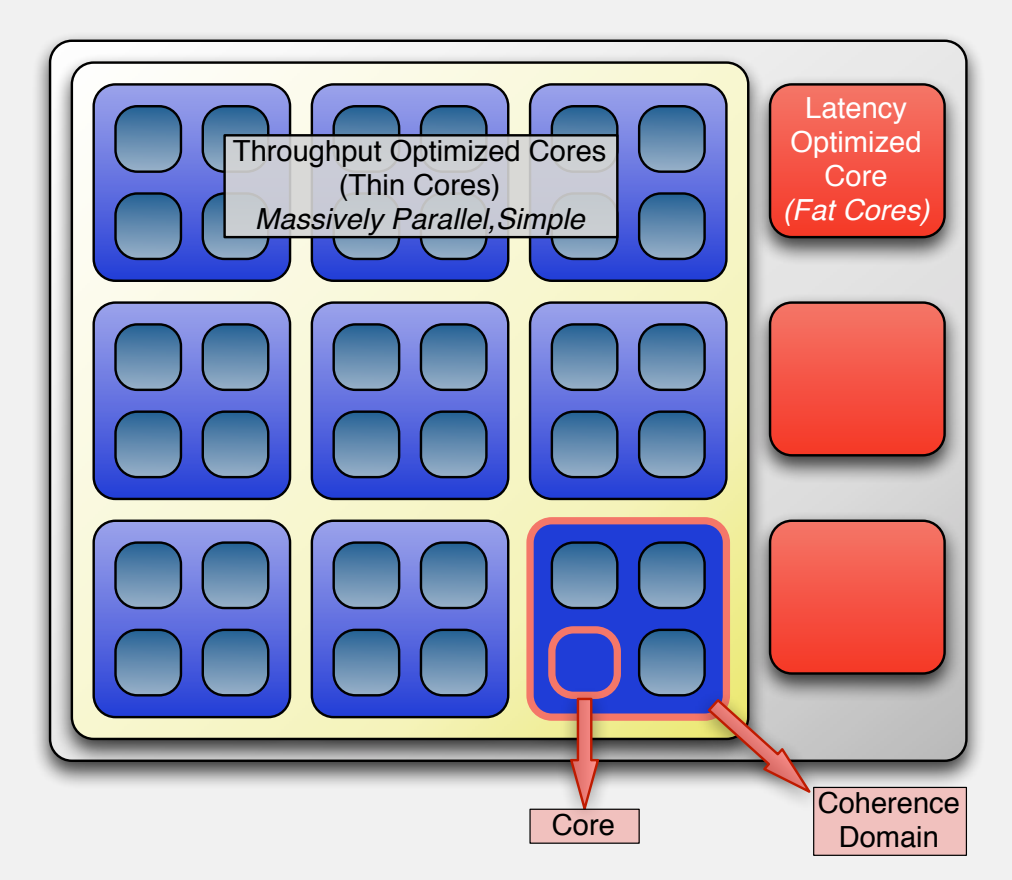
- Separation of concerns
  - Encapsulation of functionalities where possible
  - Abstractions for encapsulations
    - Offload complexity where possible
- Hard-nosed trade-offs
  - Flexibility and composability Vs raw performance
  - Extensibility and developer productivity

Example: PDE's, What we Did Before



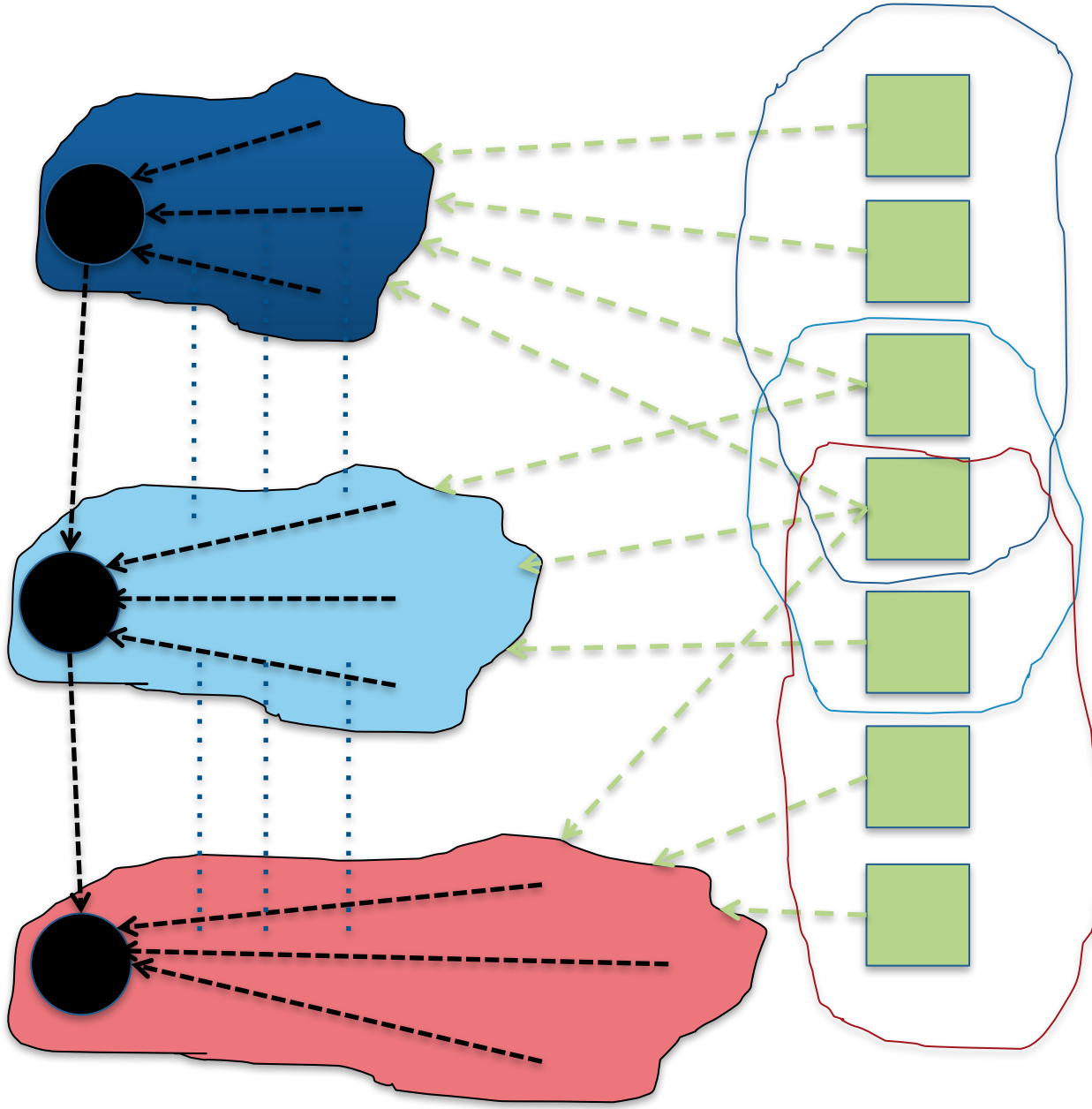
Customizations can be hidden under the virtual views as needed

Abstract Machine Model



Hierarchy of Meshes

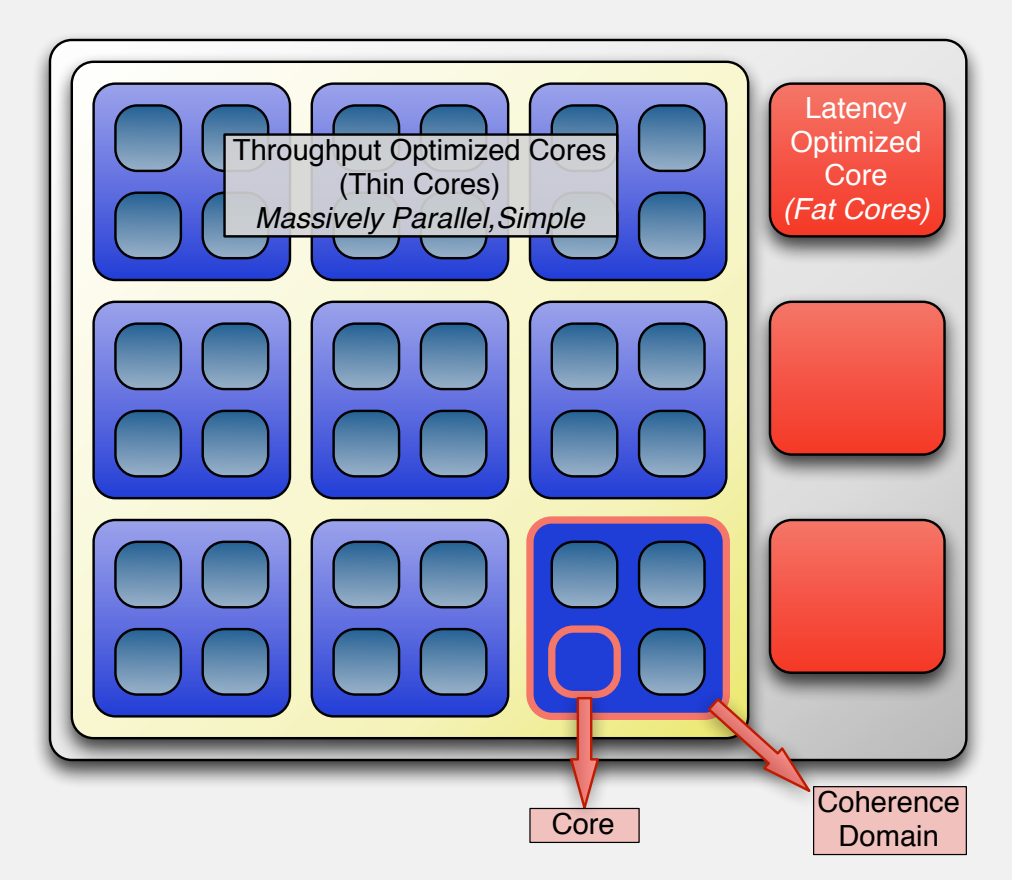
- Communications within a level
  - Exchange of data for filling ghost cells
  - Redistribution of work in re-gridding
- Across level communications
  - A level grabs data from the next coarsest level
    - Filling the ghost cells at fine-coarse boundaries
    - Refluxing at those same boundaries
    - Creating new patches when refining
- Each level needs to know only about itself and the next coarsest level



- Automatically converts to distributed meta-data
- Makes for better scalability
- Regridding and load distribution can become short range communications
- Many more domains of disjointed communications
- Less contention

Abstract Machine Model

(what are the critical elements for spatial optimizations?)



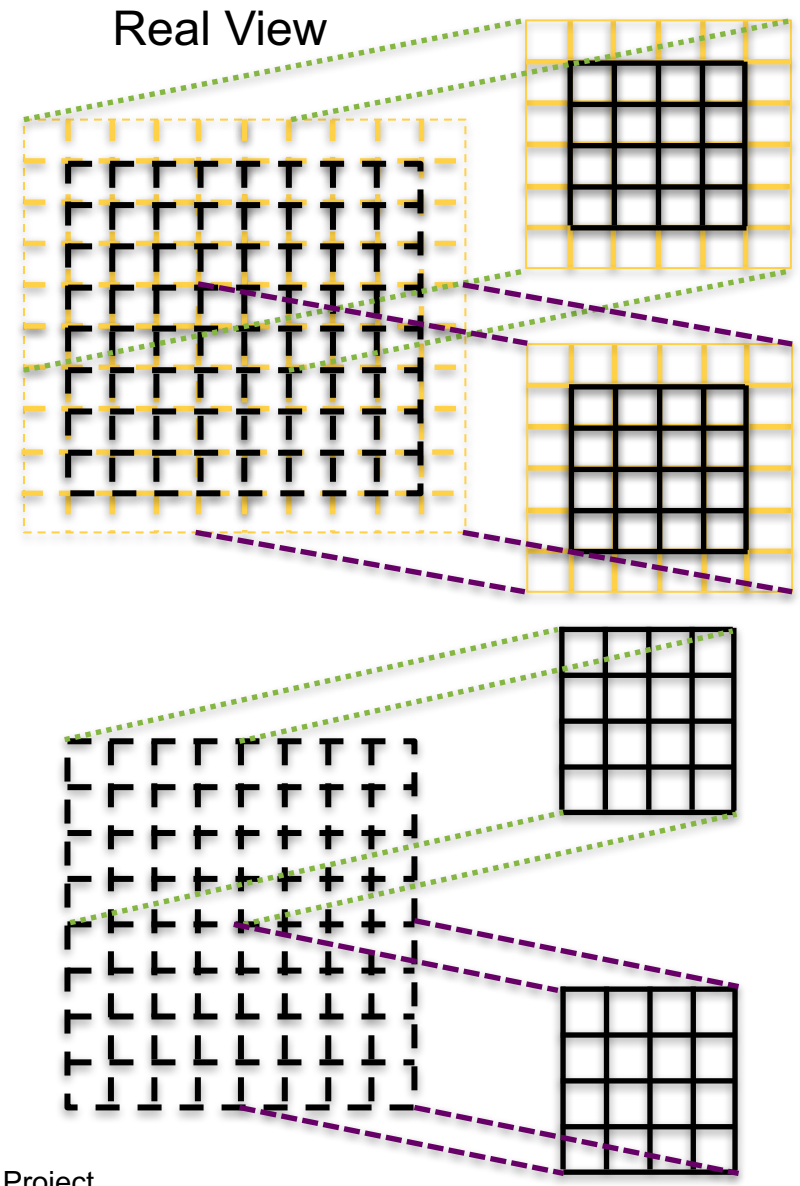
- Map levels to nearby coherence domains
  - Data locality managed within them
  - Fewer interactions outside, can perhaps be overlapped

Move away from compute-centric to data-centric programming

Exploiting the Hierarchy

- Think of each level as an independent domain of operation
  - Could be split further if needed
- One entity per domain that knows about the domain of the next coarse level
- The same entities knows all the dependences on the neighboring domain
- Assign resources to each domain depending upon some weighing criterion
  - The resource may be shared with another domain, but the operations within the domain don't know that

Hierarchical Decomposition



Tiling

- Hierarchy of tiling:
  - Larger non-overlapping tile maps to coherence domain
  - Smaller overlapping tile exposes more parallelism
  - Parameterize end-points, shape tiles as needed

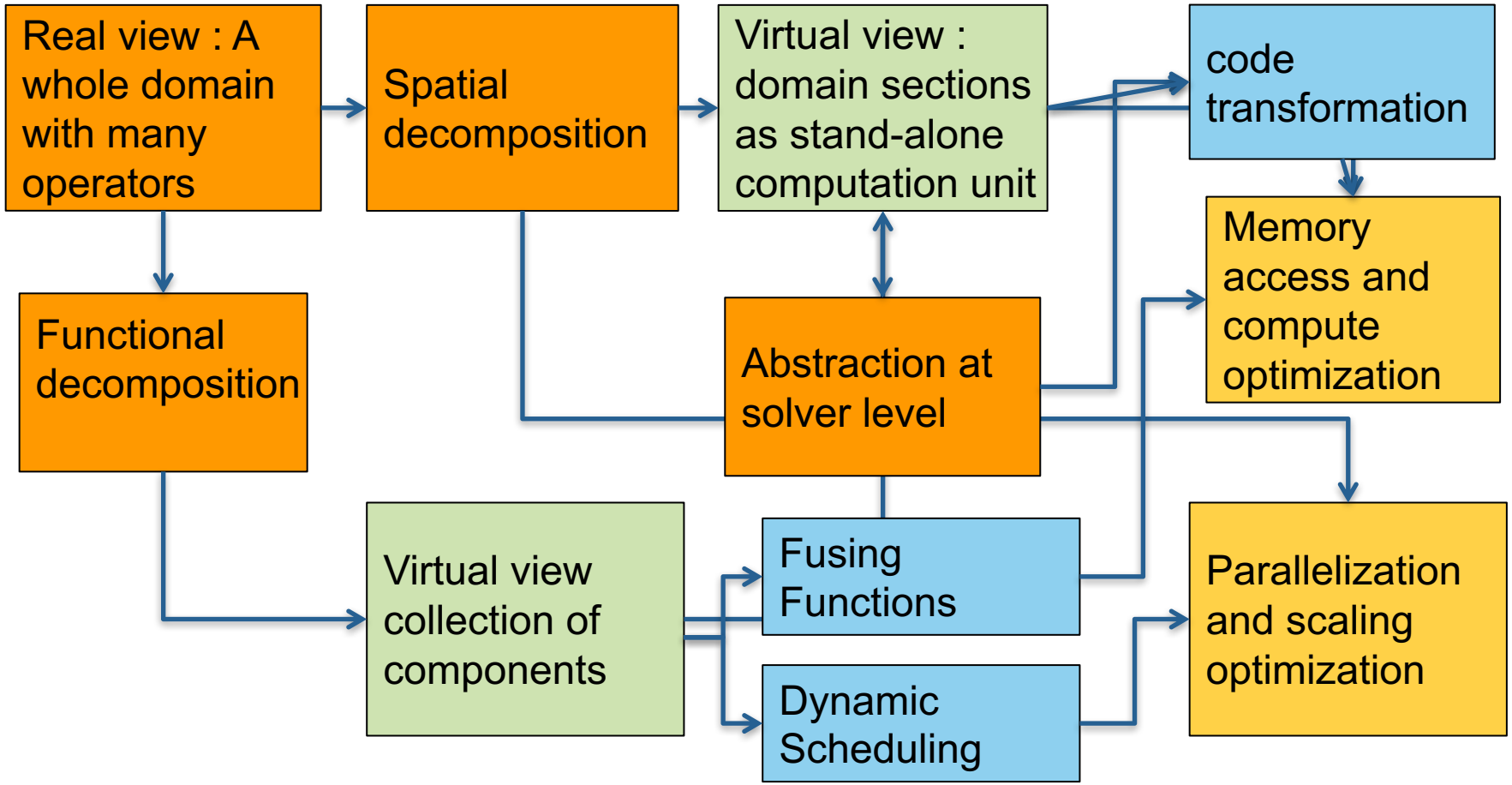
Asynchronous Execution

- With hierarchical spatial and functional decomposition rich collection of tasks
  - Articulate dependencies explicitly
  - Let the framework find the unit of computation that is ready and hand it to client code with all the necessary data
    - Under the hood, framework can be managing dependencies
    - If client code assumes not-in-place update each of the tiles is a task with neighborhood dependencies
- Can be made into build or run environment specifications through appropriate parameterization

Putting it all Together

- The construction of operators
  - Express computation in the form of stencil operators or other appropriate abstraction
  - Specify the part of the domain, and the conditions under which the operators apply
- Mix-mode parallelism
  - Parameters to control the degree of tiling or other forms of mix-mode parallelism
    - Could be handed to the compiler when technology arrives
  - Framework forms the data containers
- Dynamic tasking
  - Smarter iterators that are aware of mix-mode parallelism and dependencies
  - The iterating loops give up control and do while loops

Example: PDE's

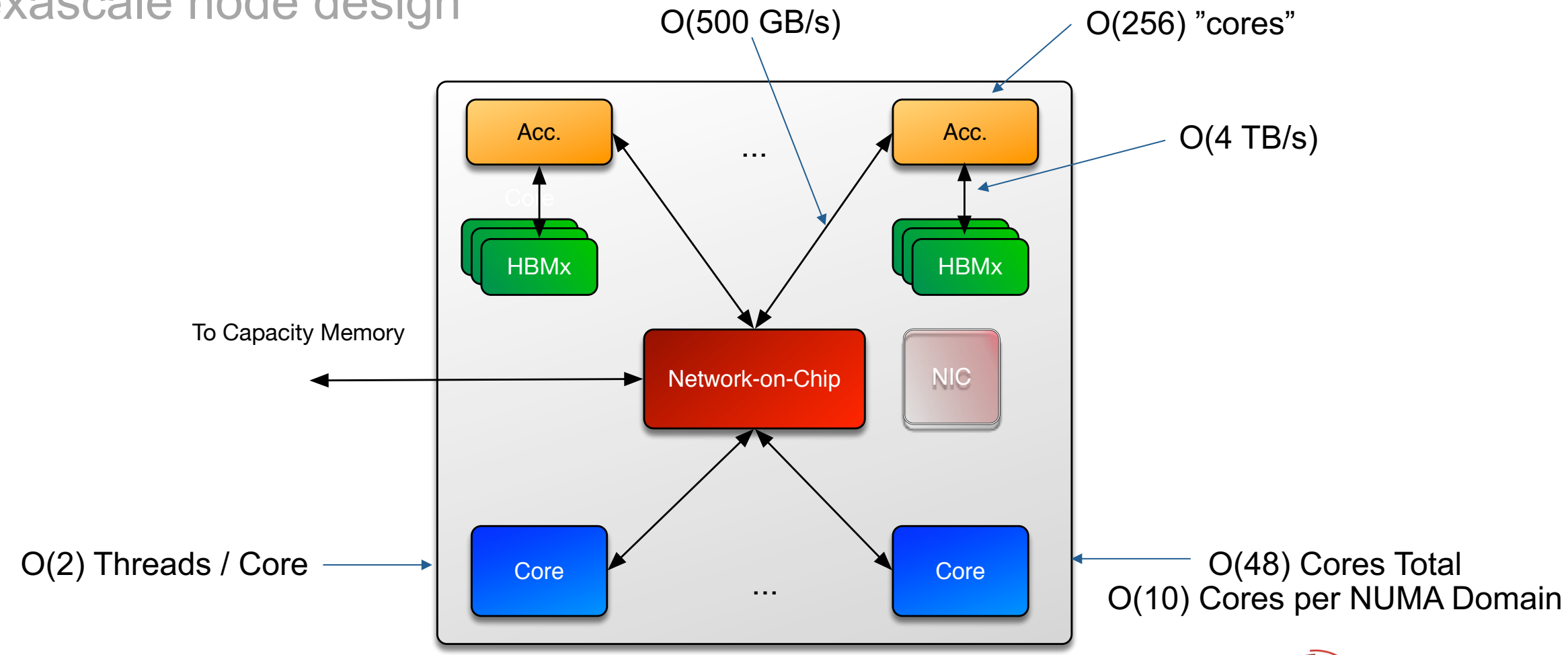


So what's a developer to do?

Some high-level suggestions to prepare

Putting it all together…. NEED TO SCRUB NUMBERS

Combining Abstract Models and Proxy Architectures for a potential exascale node design



Preparing for Exascale

What can you do today, on existing machines?

- Many aspects of the system design are in flux
  - Some vendor-specific optimizations will not be available until late in the project
  - In spite of this, there are concrete actions that can be taken *today*
- Optimize your application to reduce memory capacity
  - It is likely that available (fast) memory per core, or the FLOP to Byte-of-Capacity ratio will decrease
- Node performance is increasing at a faster rate than interconnect
  - Developers of communication-bound applications may want to put more efforts into algorithms that reduce the volume of communication.

Processor / Node Suggestions

Suggested Best Practices	
Exascale Trend	**Programming Practice**
Increasingly heterogeneous accelerators	Shared CPU-GPU address spaces will improve data transport and performance over pre-exascale systems. From the GPU perspective, GPU memory is used like in-package memory, and CPU memory is used like on-node DRAM.
On-package accelerators	These benefit from united memory space, making coding easier. However, one must still partition the work between GPU and CPU, often explicitly.
Increased data parallelism (SIMD)	Increasing generalization of SIMD units makes it more likely that code will vectorize, but the increasing SIMD unit width makes it less likely that all the units will be scheduled in most loops. Roofline models may assist analysis and optimization here.
Exacerbated node NUMA issues	The large number of processors in nodes dictates a hierarchical on-chip network, yielding NUMA memory access and non-uniform processor to processor communications. To date, this has been mainly a code tuning issue, but it also may have design implications
Potentially heterogeneous nodes/ performance portability	In some cases, one can design for performance portability across node types with success. In other cases inner modules must be coded for each node type. Tools like Kokkos and Raja can be aide in performance portability while reducing node-specific coding.

Memory Suggestions

Suggested Best Practices	
Exascale Trend	**Programming Practice**
Near, low capacity, high BW	Arithmetic performance is best when all operands come from registers and near memory. When possible, block work to fit. Unless all data fits in near memory, data structures should be explicitly partitioned between HBM and DRAM.
On-node memory (e.g., DRAM)	If possible, stage data from DRAM into HBM, asynchronously. This memory may also be useful for reference tables (read-mostly). This may involve explicitly adding data movement instructions to the code.
Far, high capacity, heterogeneous (non-uniform latency)	Has multiple use cases, depending on the code, for example (1) as a burst buffer, staging data to/from capacity storage (e.g., disk), (2) checkpoint storage or buffer, and (3) large, intermediate data store or database. It may be faster to compress data going to this tier.
Partitioned address spaces	API calls to explicitly allocate data in particular memory.

Interconnect Suggestions

Suggested Best Practices	
Exascale Trend	**Programming Practice**
Fabric topology and bandwidth	Local (in-rack) connectivity is typically better, but the performance of large applications depends more on contention across the fabric. The primary code adaptation is to keep I/O intensive phases out of the way of communication intensive phases.
Effect of processor core type in node to network interface	Trends are yielding nodes with many, relatively lightweight processors e.g., cores. Often it takes multiple cores to feed a network interface at full capacity, dictating communications parallelism, e.g., multiple MPI tasks per node. This also drives up node memory requirements.
Software overhead	Since cores tend to be lightweight, it becomes increasingly important to use low software overhead communications, e.g., switching applications to MPI3 or PGAS can improve performance.

Feedback (Slide WIP)

Spur discussion from the session

- What features are most useful?
- How would you use AMMs?
- What parameters are critical?
- Point out the things the developers can do now, on today's machines that will continue to be valuable
  - For instance, optimize for mem capacity
  - The increasing gap between node performance and interconnect capabilities would require developers of communication-bound applications to put more efforts into algorithms that reduce the volume of communication.

Backup

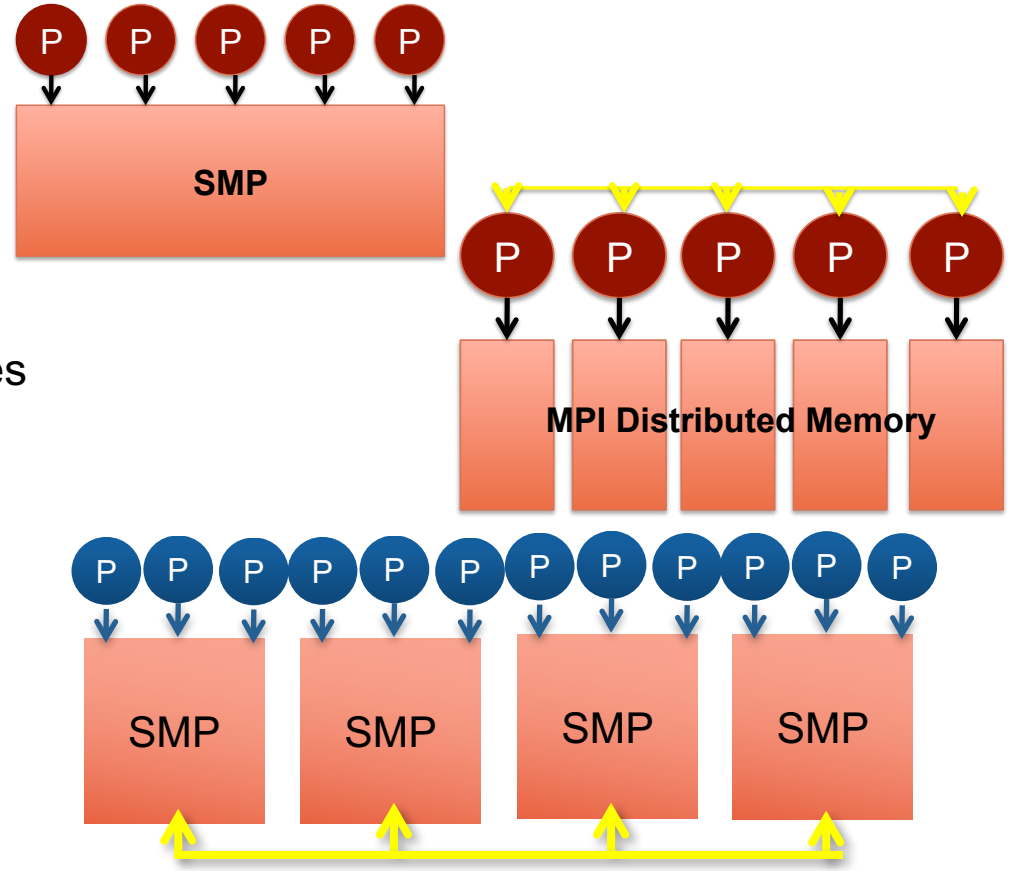
The Programming Systems Challenge

- **Programming Models and Abstractions are a Reflection of the Underlying Machine Architecture**
  - *Express what is important for performance*
  - *Hide complexity that is not consequential to performance*
- **Current Programming Abstractions are Increasingly Mismatched with Underlying Hardware Architecture**
  - *Changes in computer architecture trends/costs*
  - *Performance and programmability consequences*
- **Technology changes have deep and pervasive effect on ALL of our software systems *(and how we program them)***
  - *Change in costs for underlying system affect what we expose*
  - *What to **virtualize***
  - *What to make more **expressive/visible***
  - *What to **ignore***

The Programming Model is a Reflection of the Underlying *Abstract* *Machine Model*

Martha Kim, Columbia U. Tech Report "Abstract Machine Models and Scaling Theory"
http://www.cs.columbia.edu/~martha/courses/4130/au13/pdfs/scaling-theory.pdf

- **Equal cost SMP/PRAM model**
  - No notion of non-local access
  - int [nx][ny][nz];
- **Cluster: Distributed memory model**
  - CSP: Communicating Sequential Processes
  - Distributed memory message passing
  - int [localNX][localNY][localNZ];
- **2-level (CTA in Martha Kim Taxonomy)**
  - Candidate Type Architecture (CTA)
  - MPI+X model (for all practical purposes)

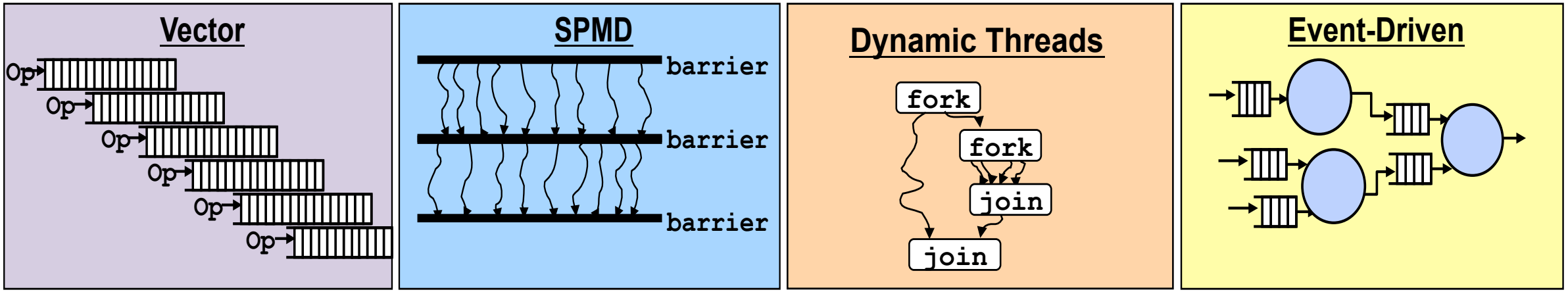


2-Level MPI+X is dominant, but insufficient!

- **Whats Next?**

Execution Models (*Another manifestation of an AMM)*

Examples of parallel execution models



- What is the parallelism model?
- How do we balance *productivity* and *implementation efficiency*
- Is the number of processors exposed in the model
- How much can be hidden by compilers, libraries, tools?

Durable Programming Abstractions
our current practices fail in this area

- **What we need for portable programming is better abstractions**

- **Identifying the abstract machine model is the first step along the path to identifying powerful/portable abstractions**

- **Focus on durable abstractions**
  - Features that are long-term trends for future processors
  - (not targeting ephemeral/temporary or proprietary features)

What is Performance Portability?

Definition from ASCR Programming Models Report (2009)

Minimize the number of lines of code that need to change to re-tune when moving between different vendor architectures of the same generation and to future generations of the same vendor architecture

The fact that we have to rewrite so much of our code for different machines says more about our programming environment than about the machines

(the programming environment targets the wrong abstract machine model)

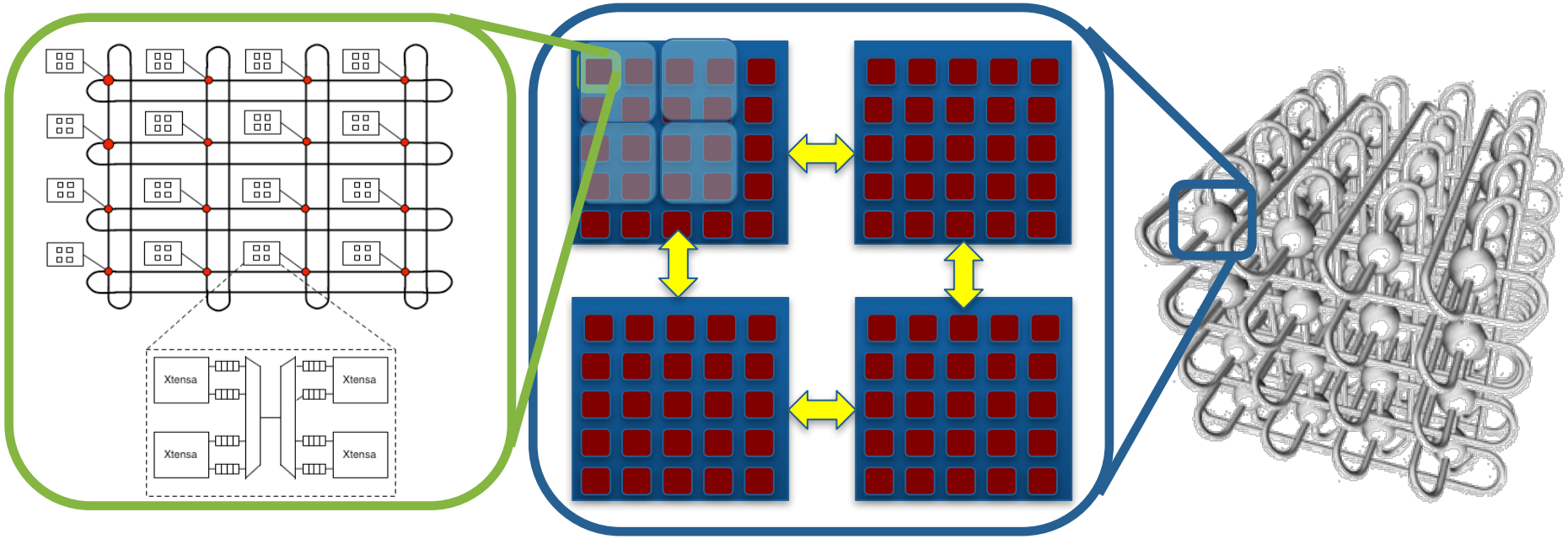
How Do We Use AMMs and Proxy Hardware to Reason about Exascale Challenges?

The Importance of Codesign *(navigating a complex design space)*

- **Changing Hardware is very expensive and takes a long lead time**
- **Changing Software (rewriting our codes) is very expensive and takes a long lead time**
- **Codesign is quantitative trade-offs analysis because in a cost and power constrained environment, you need to know what you are wiling to give up to get what you want.**
  - Easy to ask for more features or BW to be added to the machine
  - It is much harder to evaluate what you are willing to give up
  - particularly when the cost functions are highly non-linear and machines do not yet exist (need models)
- **CoDesign is *risk mitigation for expensive decisions.***

Parameterized Machine Model

(what do we need to reason about when designing a new code?)



Cores
- How Many
- Heterogeneous
- SIMD Width

Network on Chip (NoC)
- Are they equidistant or
- Constrained Topology (2D)

On-Chip Memory Hierarchy
- Automatic or Scratchpad?
- Memory coherency method?

Node Topology
- NUMA or Flat?
- Topology may be important
- Or perhaps just distance

Memory
- Nonvolatile / multi-tiered?
- Intelligence in memory (or not)

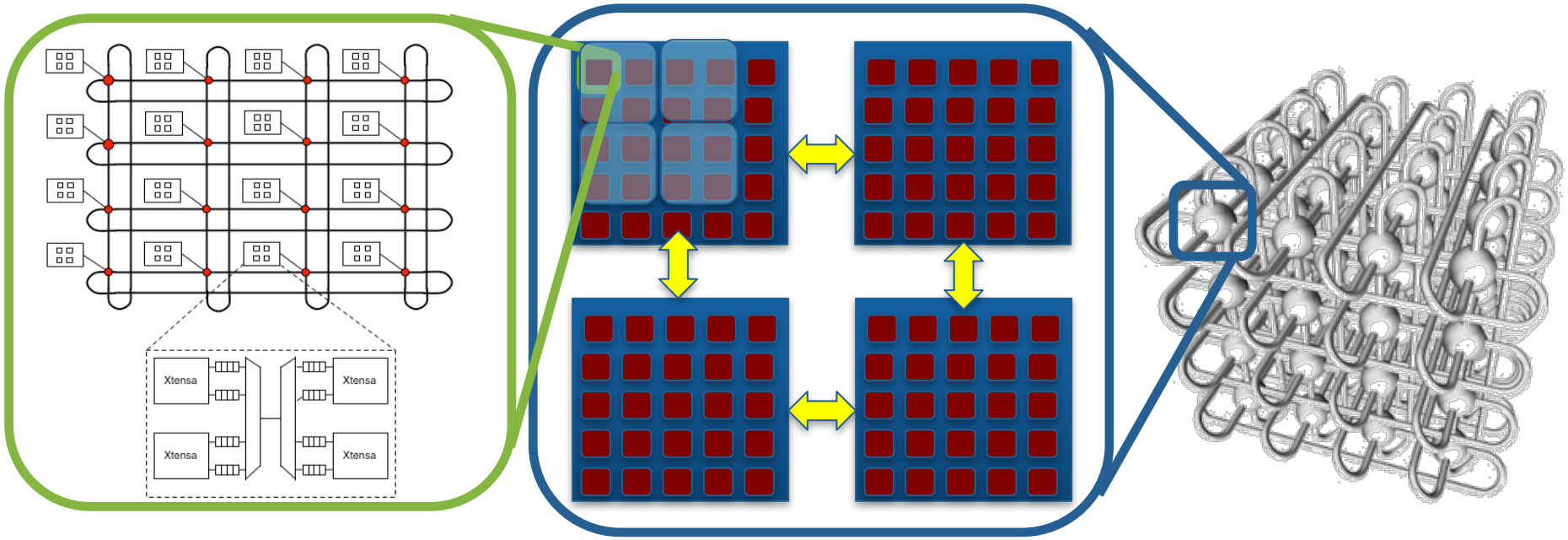
Fault Model for Node
- FIT rates, Kinds of faults
- Granularity of faults/recovery

Interconnect
- Bandwidth/Latency/Overhead
- Topology

Primitives for data movement/sync
- Global Address Space or messaging?
- Synchronization primitives/Fences

Abstract Machine Model

(what do we need to reason about when designing a new code?)



For each parameterized machine attribute, can

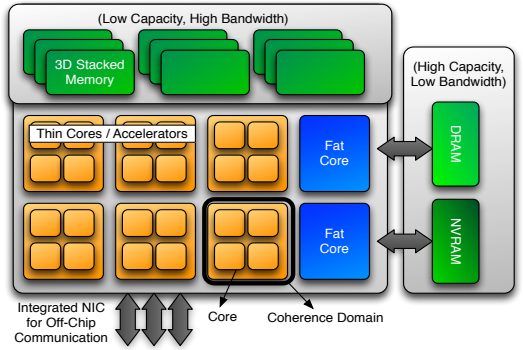
- **Ignore it:** *If ignoring it has no serious power/performance consequences*
- **Expose it (*unvirtualize*):** *If there is not a clear automated way of make decisions*
  - Must involve the human/programmer in the process ***(make pmodel more expressive)***
  - Directives to control data movement or layout (for example)
- **Abstract it (*virtualize*):** *If well enough understood to support automated mechanism to optimize layout or sched*
  - This makes programmers life easier (one less thing to worry about)

Want model to be as simple as possible, but not neglect any aspects of the machine that are important for performance

Programming Model Challenges (why is MPI+X not sufficient?)

- **Lightweight cores not fast enough to process complex protocol stacks at line rate**
  - Simplify MPI or add lighter thread match/dispatch extensions
  - Or use the memory address for endpoint matching (GAS/PGAS)
- **Can no longer ignore locality (especially inside of node)**
  - Its not just memory system NUMA issues anymore
  - On chip fabric is not infinitely fast (Topology as first class citizen)
  - Relaxed-relaxed consistency (or no guaranteed HW coherence)
- **New Memory Classes & memory management**
  - NVRAM, Fast/low-capacity, Slow/high-capacity
  - How to annotate & manage data for different classes of memory
- **Asynchrony/Heterogeneity**
  - New potential sources of performance heterogeneity
  - Is BSP up to the task?

From Abstract Models to Proxy Architectures



- **AMM** tells you what key hardware features of the node are and how topologically they are connected together
- **Proxy Architecture** fills in those speeds and feeds so you can understand *how important* a particular resource is to app
- *Proxy Architecture enables* ***thought experiments about trade-offs***

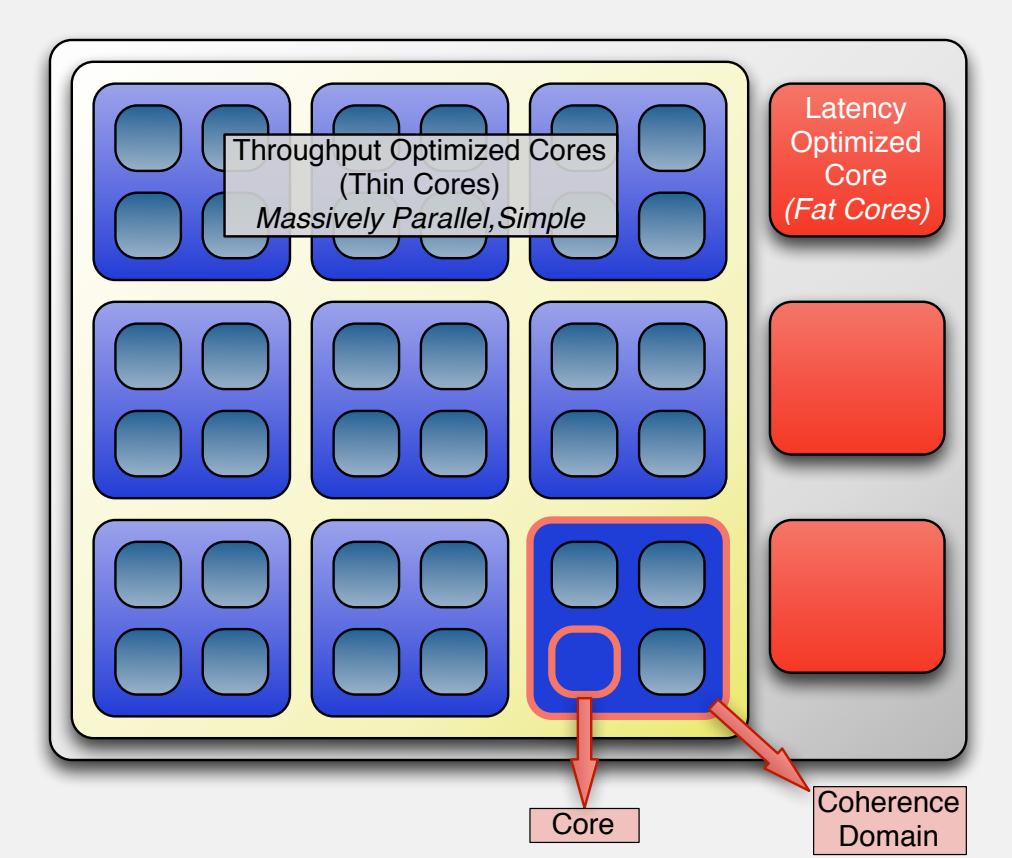
		Hopper	exaNode1	exaNode2	exaPIM
Memory BW	TB/s/node (chip)	0.05	1	4	1
Memory Size	GB/node (chip)	32	256	32	256 (16)
Flops	TF/node (chip)	0.03	10	10	0.7
# of Cores	Cores/chip	6	1024	1024	64
# of Chips	Chips/node	4	1	1	16
Cache (last L)	$/core (KB)	1024	32-256	32-256	0
Cache L1	$/core (KB)	64	32-64	32-64	0
NIC BW	GB/s	1	100	400	25
NIC Overhead	microseconds	1	0.4	0.02	0.02
Registers	KB/chip				

- exaNode1 and 2 are many core architectures
- exaNode1 uses commodity NIC and memory technology
- exaNode2 uses custom on-board NIC and faster memory technology
- exaPIM: Processing Near Memory, cache-less architecture

Data Locality

Abstract Machine Model

(what are the critical elements for spatial optimizations?)



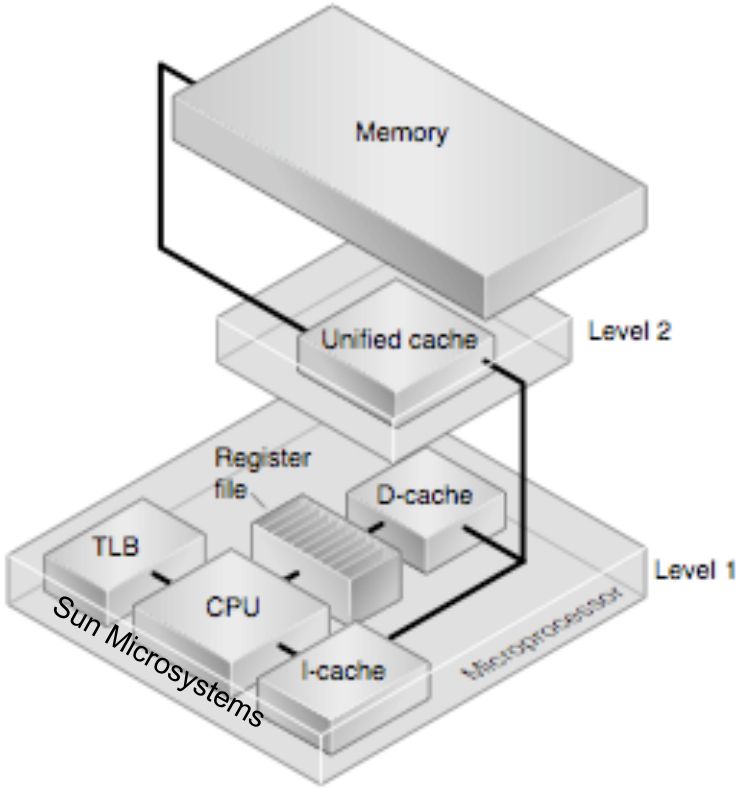
- The number of cores on a chip will be on the order of 1000s
  - *Expect 100x concurrency*
- Maintaining cache coherence is NOT scalable
  - *Expect coherence domains*
- Flat and infinitely fast on-chip interconnect is NO longer practical
  - *Expect complex NOCs*
- Processing elements within a node are NOT equidistant.
  - *Expect non-uniformity*

Move away from compute-centric to data-centric programming

Data Locality Management

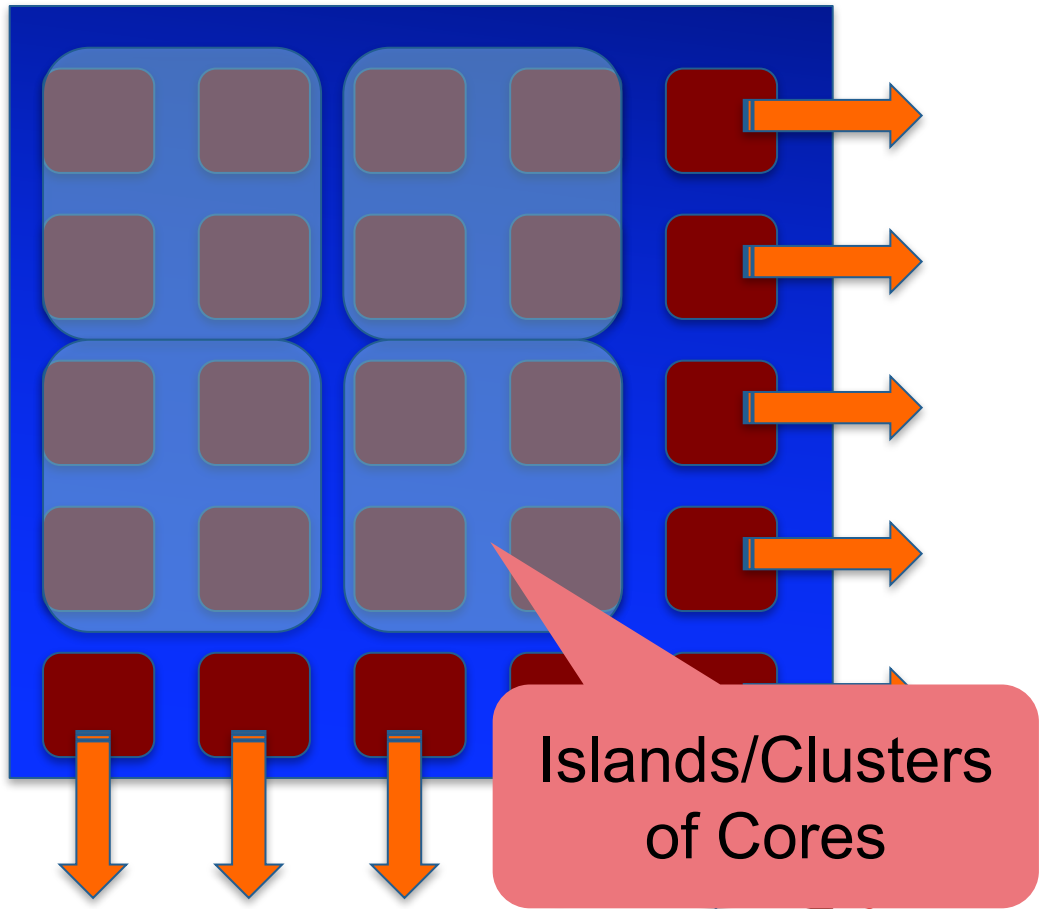
Vertical Locality Management

(spatio-temporal optimization)

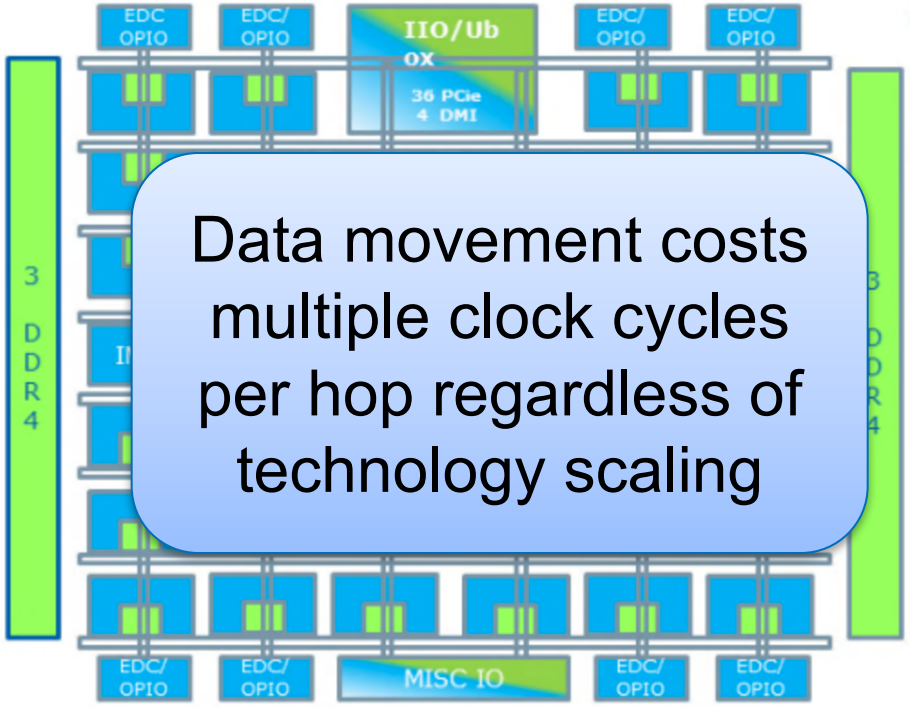


Horizontal Locality Management

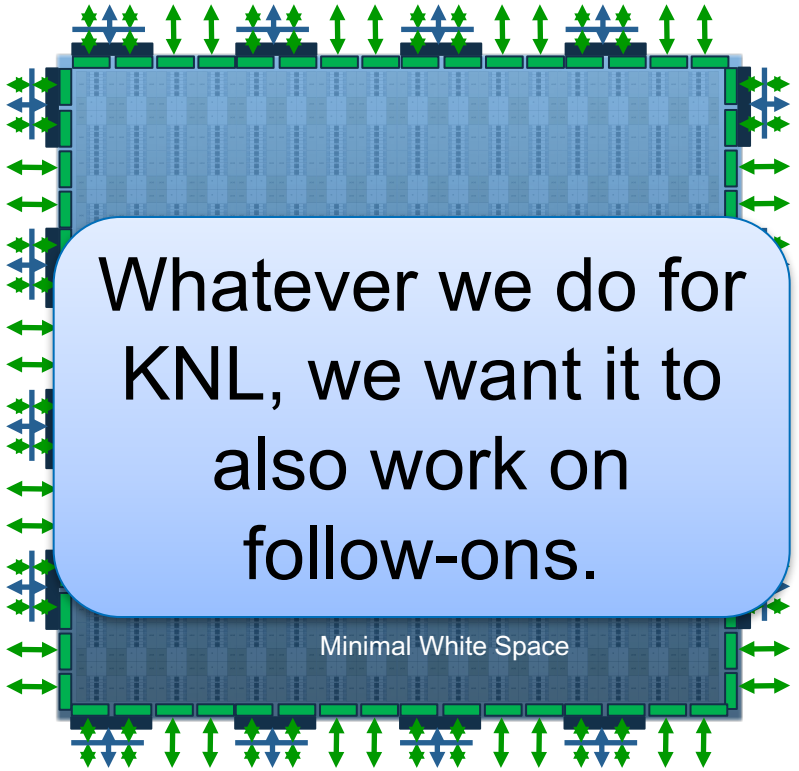
(topology optimization)



KNL Mesh On-Chip Interconnect 36 islands (2015)



Some Future Mesh On-Chip Interconnect Many more islands (202x) More non-uniformity

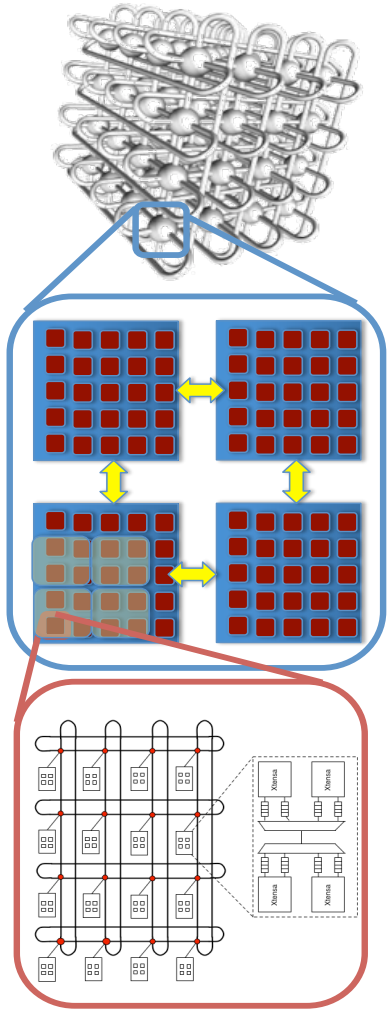


Towards a Data Centric Computing Model

- **Old Model (current practice with OpenMP)**
  - Describe how to parallelize loop iterations
  - Parallel “DO” divides loop iterations evenly among processors
  - . . . but where is the data located?

- **New Model (Data-Centric)** ***also in big data***
  - Describe how data is laid out in memory
  - Change applies to ALL Loop statements operate data where it is located (in-situ)
  - Similar to MapReduce, but need more sophisticated descriptions of data layout for scientific codes

```
forall_local_data(i=0;i<NX;i++;A)
  C[j]+=A[j]*B[i][j]);
```



Data Locality

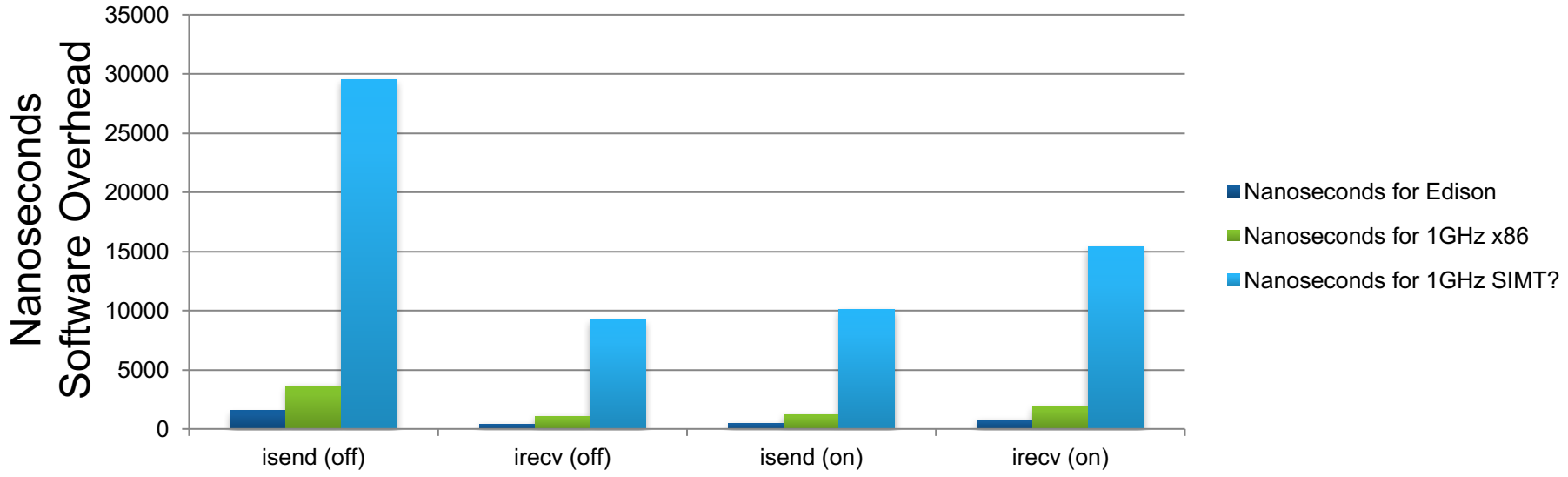
- **Trend: increasing localization of data movement**
  - More NUMA domains (now within the chip)
  - Higher cost of moving off-chip
  - On Cori, we can ignore with some modest cost
- **Observations**
  - This is physics (not architecture), so hard to change
  - NUMA memory locality different than NUMA on chip (both worse)
- **What Apps teams should think about**
  - Need to think about how you can express your algorithm and data layouts in a way that maps easily to a 2D array of processor elements within a chip.
- **What ST teams should think about**
  - How can we provide features in the programming environment that automate the tedious process of binding data & work to specific cores (pinning) in a manner that is constrained by topology metadata

Lighter Weight Communications

Overhead for Messaging

- Lightweight Cores see higher overhead of complex messaging schemes
- More pressure on Strong Scaling

Avg cycles per call *(to do nothing)* On Intel Ivybridge	**Off Node**	**On-Node**
iSend()	3,692 cycles	1,262 cycles
iRecv()	1,154 cycles	1,924 cycles



You must strong scale to weak scale! *(a simple PDE solver example)*

- **Weak Scaling**
  - Double problem resolution in any dimension
  - Cuts step size by 2x to maintain Courant stability condition
- **Past machine scaling matched Courant scaling**
  - 2x increase in memory size (for weak scaling)
  - Came with 2x clock frequency increase (1 byte/flop)
- **Today: no clock frequency increase**
  - You can still weak scale between nodes (there is more memory)
  - Must strong scale in the node to weak scale across the machine

Reducing messaging overhead

(both on chip and off chip)

Is essential for strong scaling!

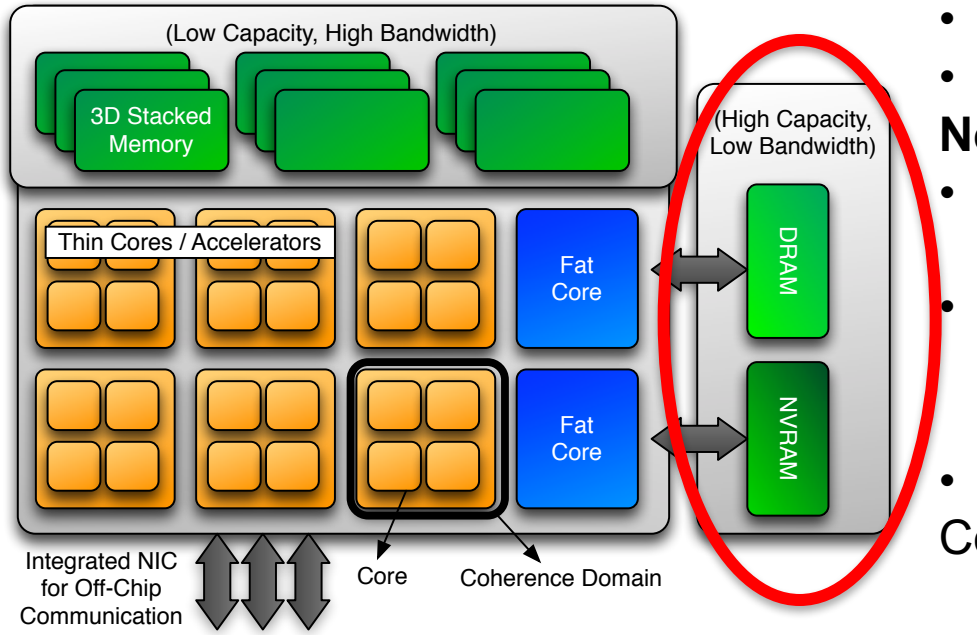
(must reduce or eliminate software critical path!)

Communications

- **Trend:**
  - Network/NIC bandwidth scaling, but per-core performance *NOT*.
  - Simpler cores take longer to run MPI protocol stack
- **Challenge**
  - Wimpy cores present challenge to current threads/MPI relationship
    - One rank per chip creates amdahl bottleneck
    - Adding ranks per chip creates challenge for strong scaling
- **Solutions**
  - Lighter weight communication protocols (e.g. PGAS)
  - Side-cores (Thor or BG/Q MPI)
  - All processors as peers in communciaiton (e.g. MPI thread multiple)
  - Direct message queues between all processors (hardware support)
    - Intel direct msg queues, AMD XTQ, NVIDIA malleable memory (can you use these features?)
- **What applications teams should think about**
  - What is your high-level abstractions for communication? (e.g. halo exchange)
  - Can it be extended to work with these different comm options? (PGAS, side-core, direct message queues, all threads as peers)
  - Which path is most effective and most maintainable in the long term
- **ST teams: already deeply involved in this area**

Memory Management Challenge

Memory Spaces



Old Paradigm for off-chip memory

- One kind of memory (JEDEC/DDRx)
- ~1 byte per flop memory capacity
- ~1 byte per flop bandwidth (0.25 typical)

New Paradigm

- **DDR4**: ~1 byte per flop Capacity and
  < 0.01 bytes/flop Bandwidth
- **Stacked Memory**:
  < 0.01 bytes/flop Capacity
  ~1 byte per flop Bandwidth
- **Non-Volatile Memory**

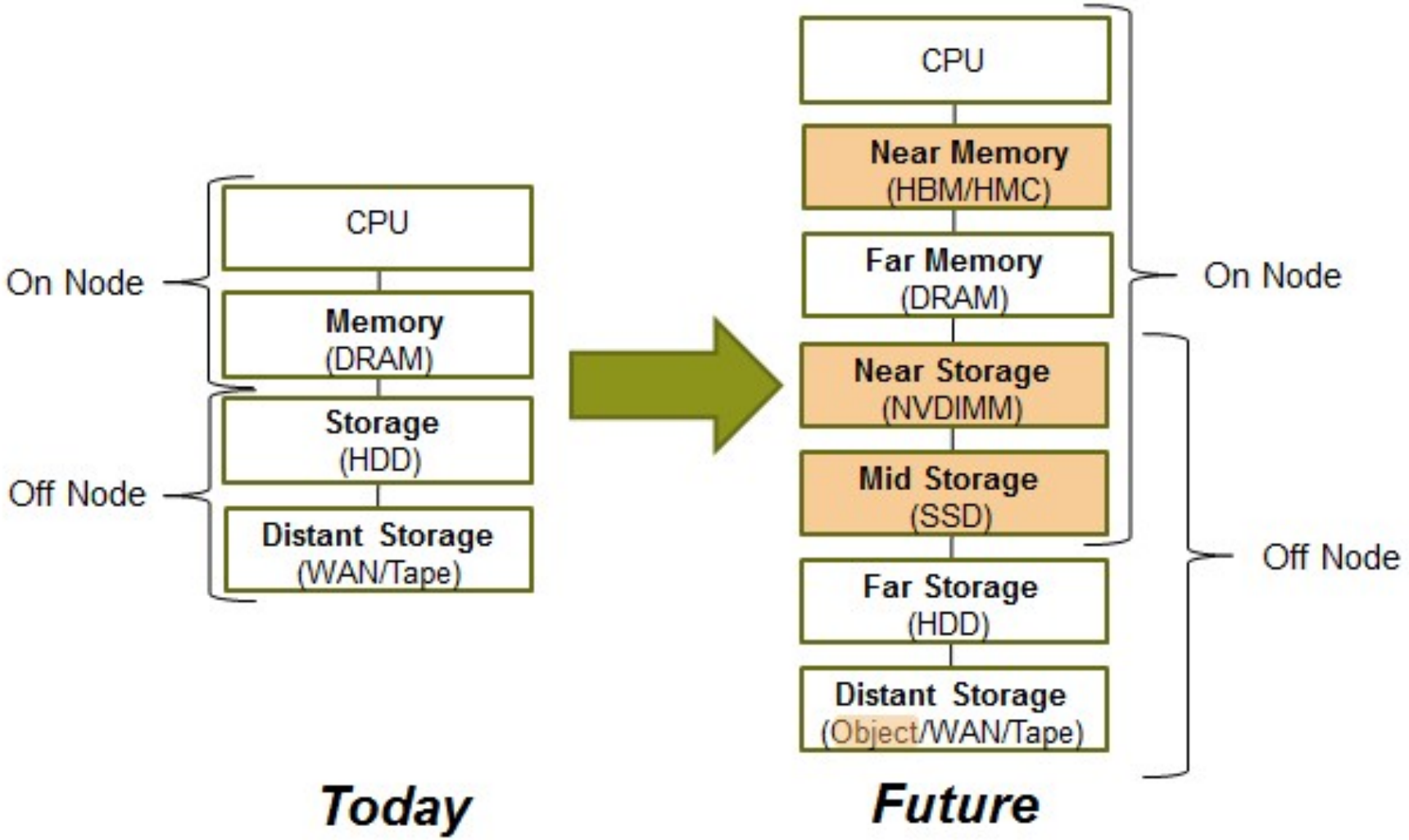
Consumes more energy and latency on write than read

Cost Constrained (cost/bit increases w/bandwidth and capacity)

Power Constraint

Bandwidth\Capacity	16 GB	32 GB	64 GB	128 GB	256 GB	512 GB	1 TB
4 TB/s							
2 TB/s	Stack/PNM						
1 TB/s			Interposer				
512 GB/s				HMC organic			
256 GB/s							
128 GB/s						DIMM	
64 GB/s							NVRAM

Trends in the Memory/Storage Subsystem



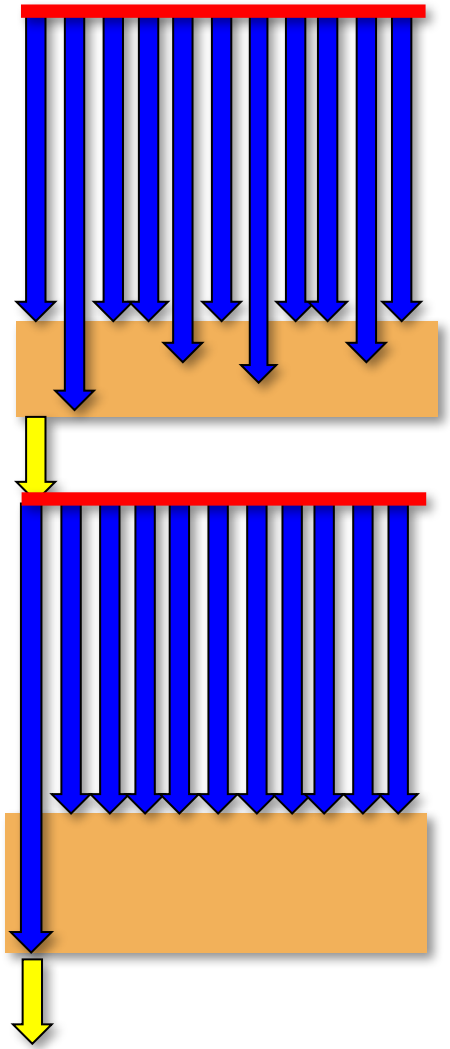
Memory Spaces

- **Trend:**
  - More kinds of memory with more diverse characteristics
  - Not like cache (NVRAM for example cannot easily be treated as just another level of the memory hierarchy)
  - Last level of memory just got a LOT slower
  - Fast DRAM is a one-time hit (0.01 bytes/flop), but might be stable
- **What applications (AD) teams should think about**
  - Can you use the high-capacity slow DRAM (eventually NVRAM) at all? (is 0.01 bytes/flop bandwidth just too slow?)
  - Can you live with 0.01 bytes/flop capacity? (can get you the bandwidth)
  - Or perhaps we should just IGNORE the slower memory (less complex)
- **What ST teams should think about**
  - Can we automate data motion & assignment for these memory spaces? (can we treat it like a cache?)
  - If not, what interfaces should we be providing to our apps programmers to make use of these spaces? (this is not rocket science…)

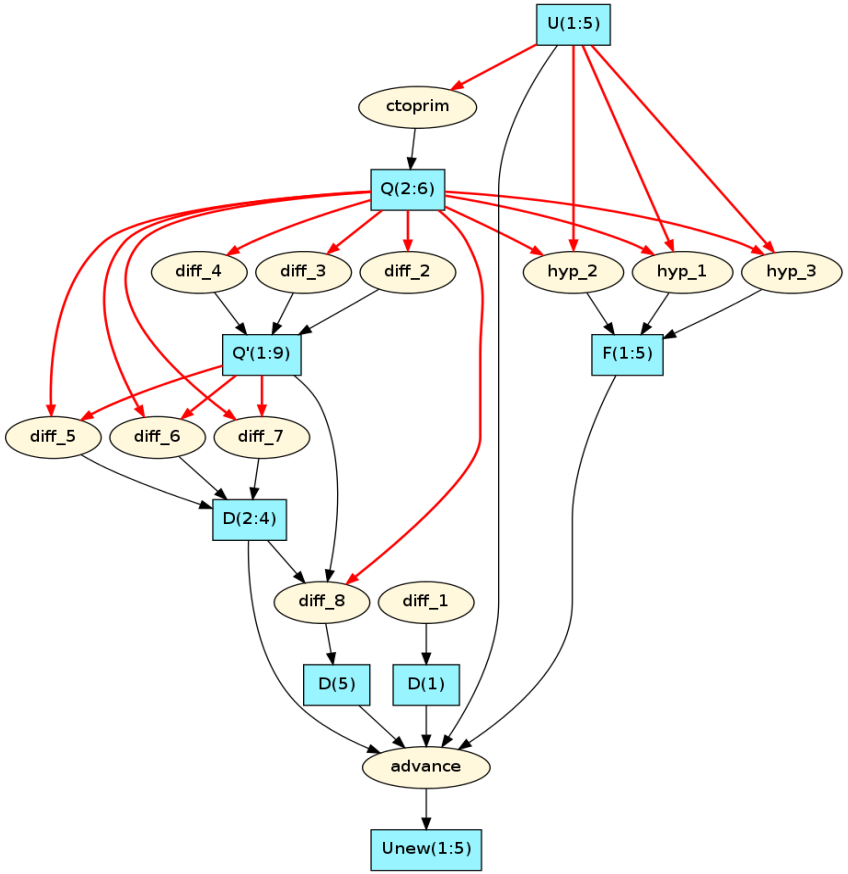
Heterogeneity / Inhomogeneity

Assumptions of Uniformity is Breaking
(many new sources of heterogeneity)

Bulk Synchronous Execution



Asynchronous Execution Model



Performance Heterogeneity

- **Trend**
  - Sources of performance nonuniformity are increasing
  - Unclear if overheads and complexity overwhelm benefits *(adding async to load-balanced code just adds overhead)*
- **Challenge**
  - This is architectural (you can turn it off potentially)
  - But if you DO turn it off, then you will pay a cost in both energy efficiency and performance (lowest common denominator)
- **What applications teams should think about**
  - Can your algorithm be reformulated to tolerate performance non-uniformity (that is predictable? That is unpredictable?)
  - need this to conduct the experiment… (don't think about this as presupposing the solution)
- **What ST teams should think about**
  - If you have some good examples of async algorithms can you develop a model that determines what the right trade-off is between runtime scheduling overheads (more in-depth discussion by Cy Chan)

SIMD

- **Where are we now**
  - KNL: 2 x 8 DP word SIMD
  - GPU: 16-32 DP word SIMT
- **Trend: slow to no growth**

- **Challenge**
  - Compilers do terrible job of SIMD and provide little feedback
  - Q: Why do I care? My code is bandwidth limited?
  - A: Because load-store also depends on SIMD (will greatly limit your L1 Load/store bandwidth if not SIMD'ized).
- **What applications teams should think about**
  - Good news is SIMD supporting more vector-like constructs (so constraints may start to look more like old vectors)
  - You know the drill (C$IVDEP) or ask Sam Williams
- **What should STteams think about?**
  - Do way have a play here? (is this primarily code generation?)
  - Can we create tools that provide more feedback than existing tools/compilers?

Wrap Up & Summary

- **AMM tells you what key hardware features of the node are and how topologically they are connected together**

- **Proxy Architecture fills in those speeds and feeds so you can understand *how important* a particular resource is to application/algorithm design**

- ***Proxy Architecture enables thought experiments about trade-offs***

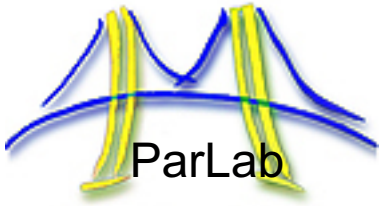
The End

For more information go to
http://www.cal-design.org/

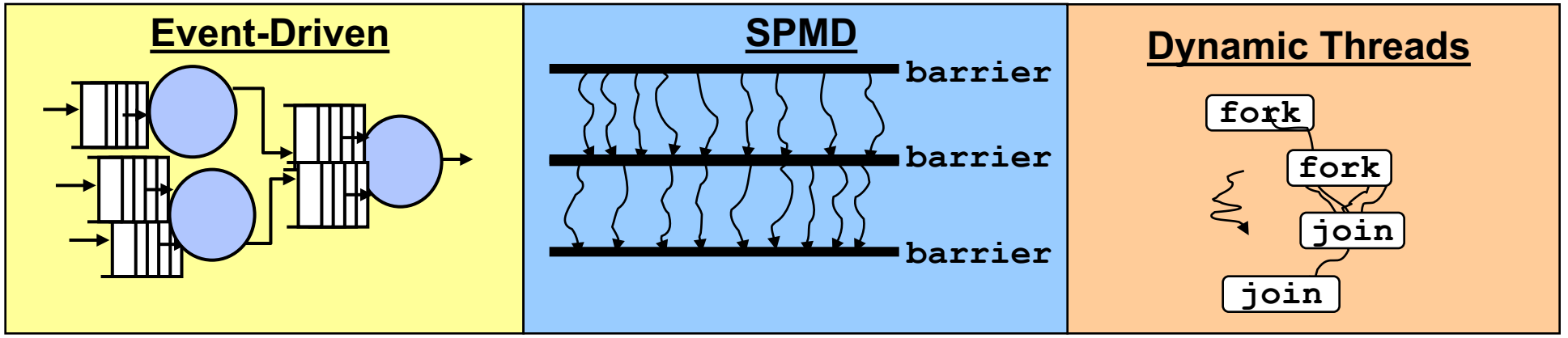
Bigger Picture with execution models

Design Space Exploration is a Multi-Level Mapping Problem

Abstract Machine Models



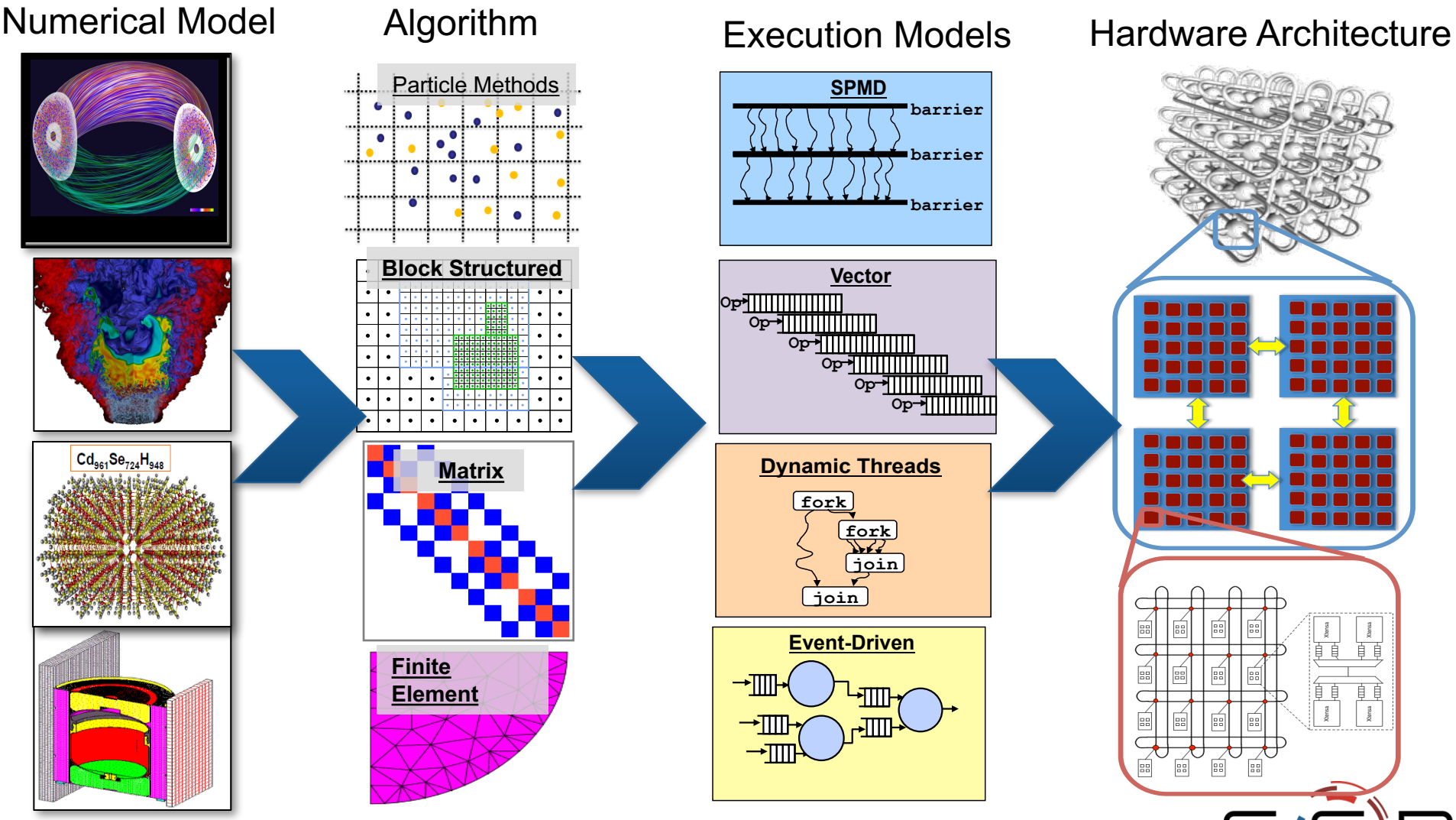
Major categories of successful parallel execution models



- What is the parallelism model for multicore *(exascale)*?
  - Must balance *productivity* and *implementation efficiency*
- Is the number of processors exposed in the model
  - HPCS Language thrust: can we virtualize the processors?
- How much can be hidden by compilers, libraries, tools?

The Mapping Problem

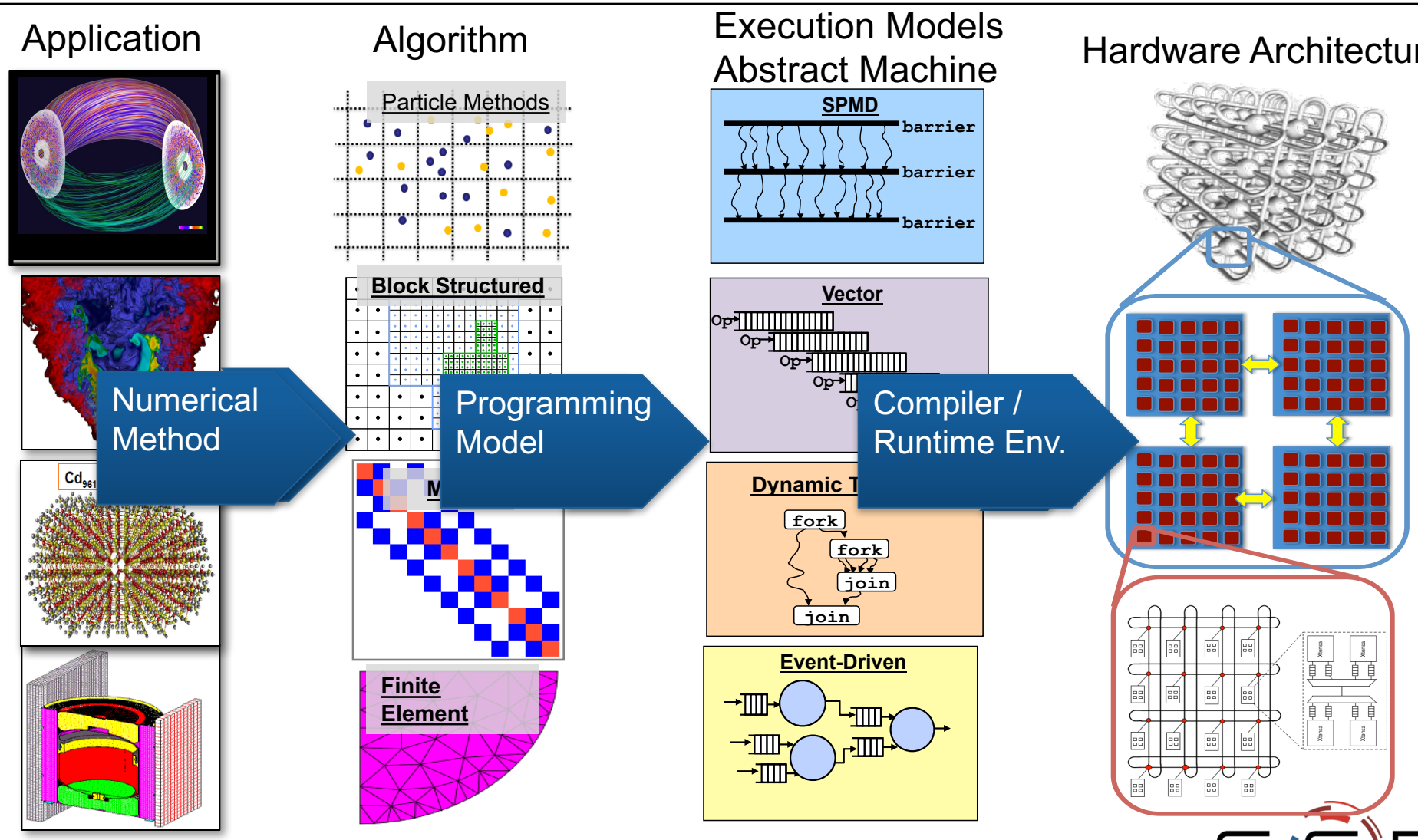
What is the best model to map computation onto underlying parallel hardware?



EXASCALE COMPUTING PROJECT

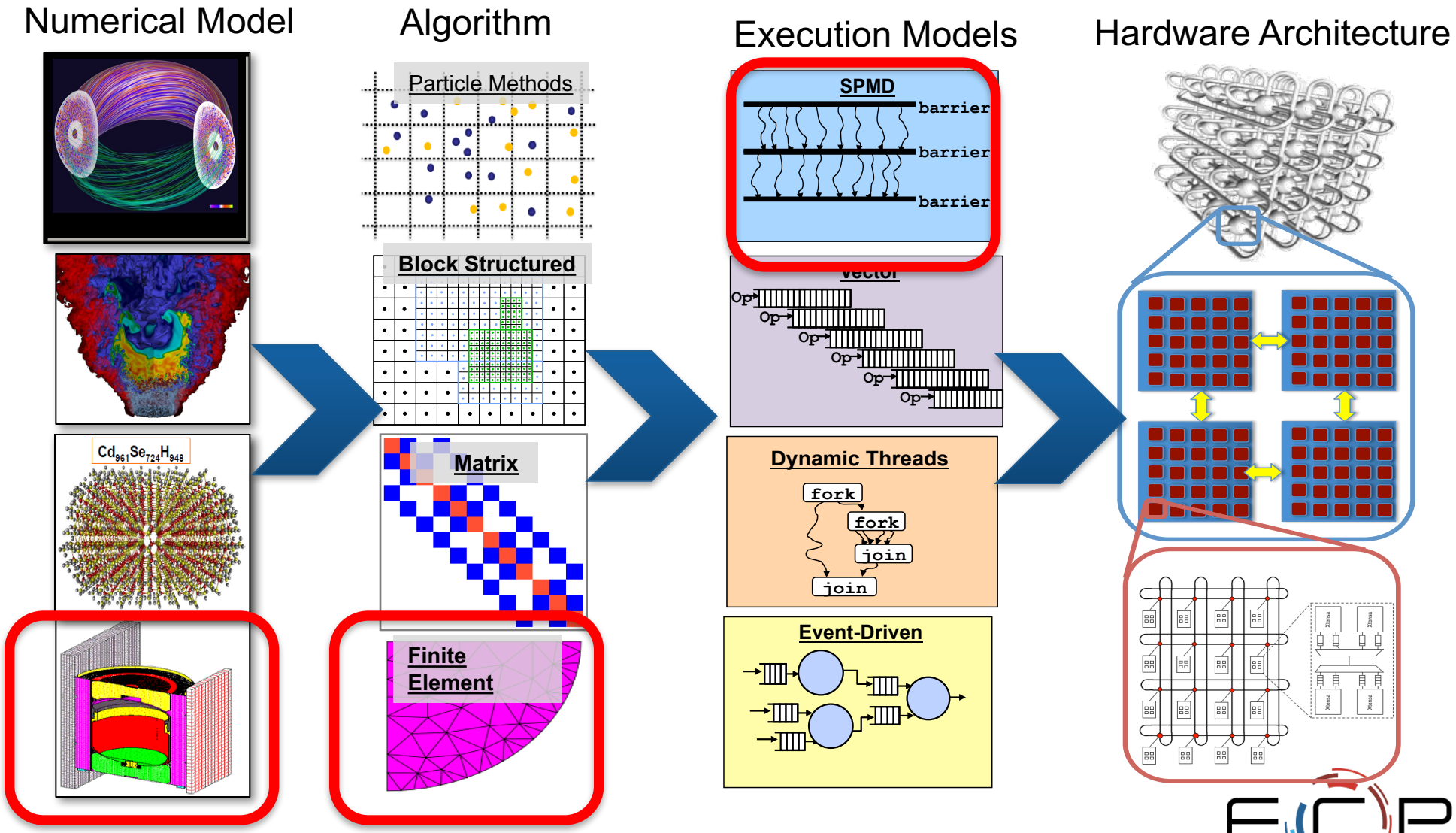
The Mapping Problem

What is the best model to map computation onto underlying parallel hardware?



You Could Formulate Direct Representation of FEM Solver *(choose your own story)*

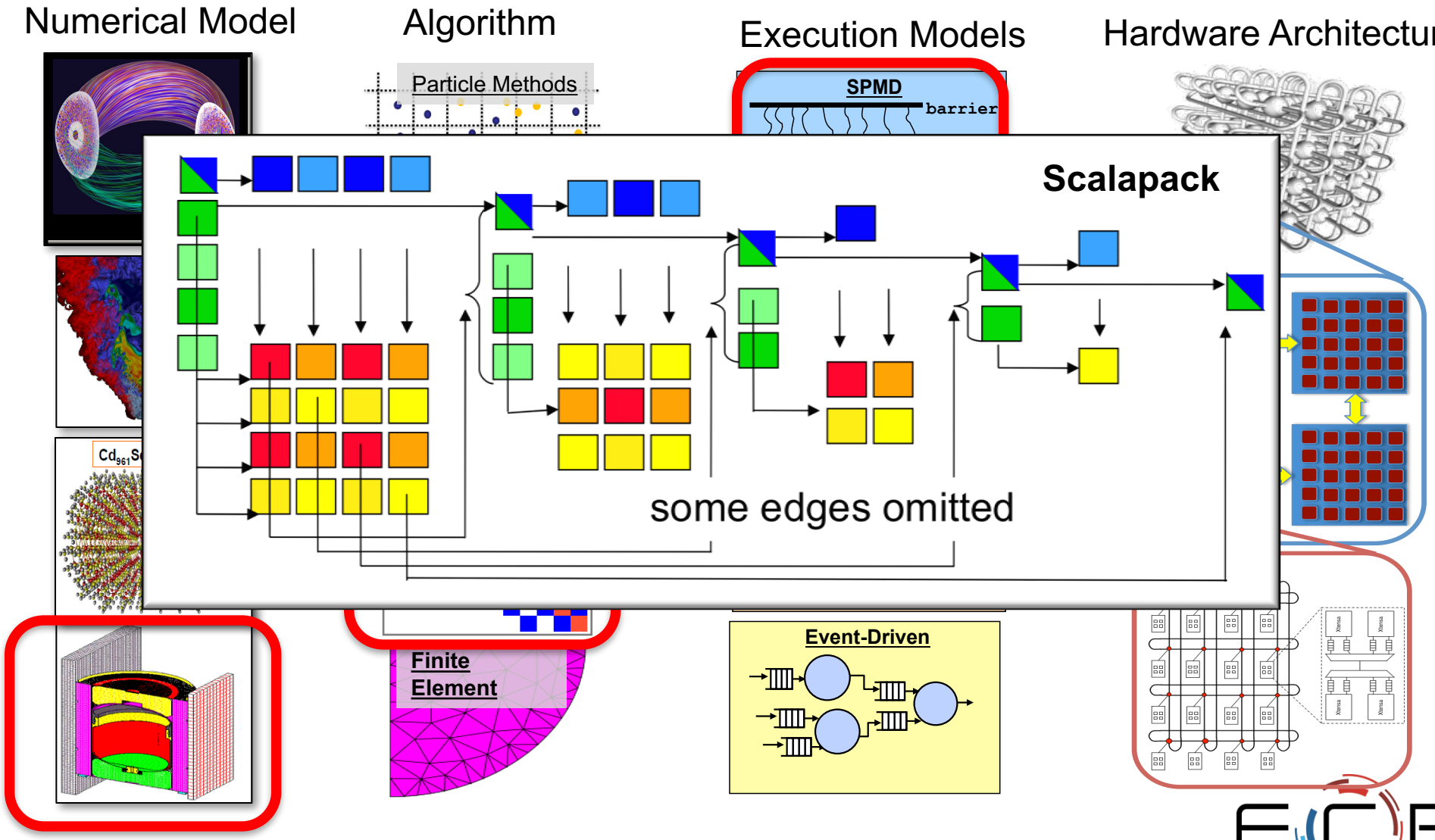
What is the best model to map computation onto underlying parallel hardware?



Or Formulate as Sparse Matrix Problem

(Use Scalapack to solve in Parallel using SPMD paradigm)

What is the best model to map computation onto underlying parallel hardware?



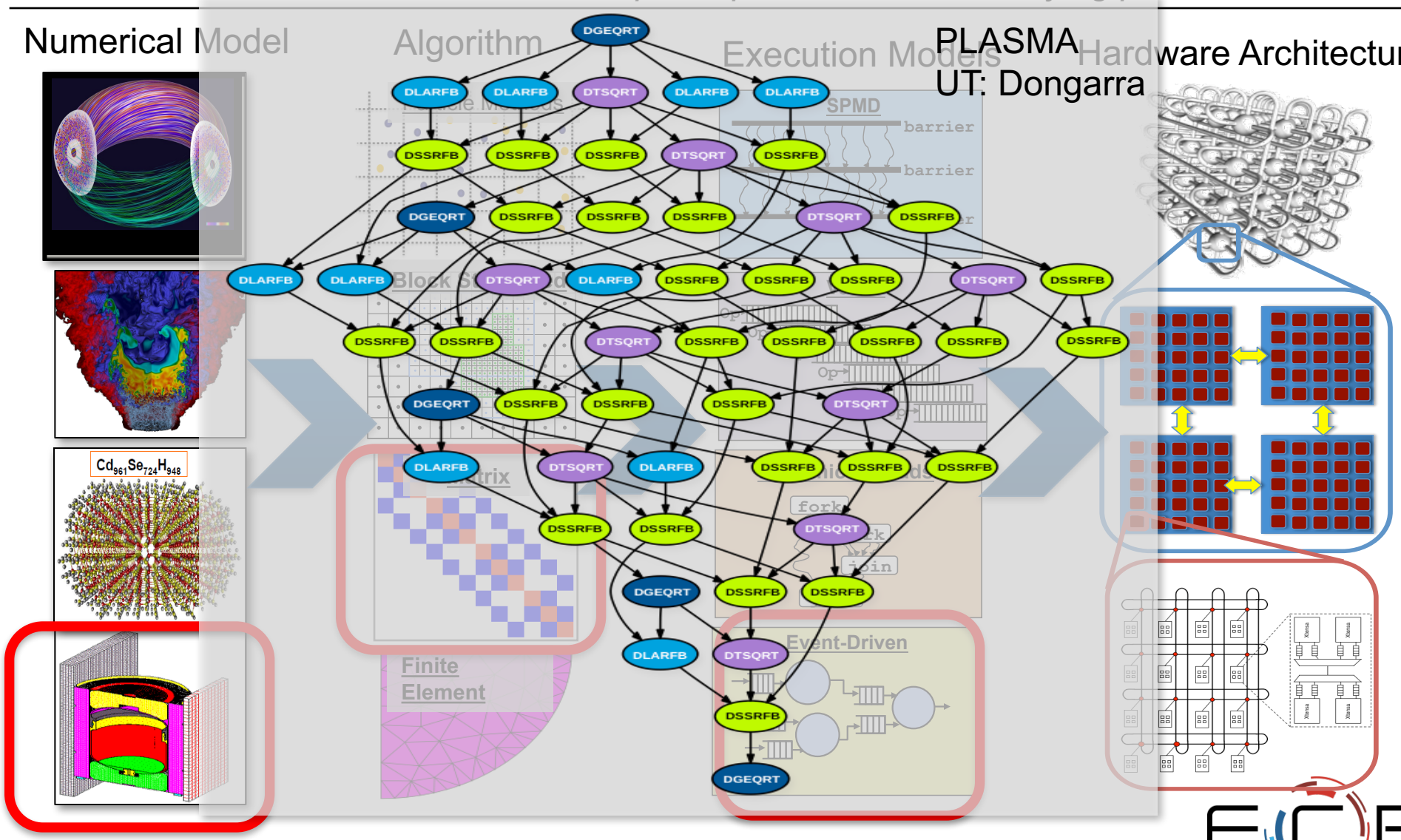
Or Formulate as Sparse Matrix Problem

(Use PLASMA Event-Driven Model for Solver)

What is the best model to map computation onto underlying parallel hardware?

Numerical Model | Algorithm | Execution Models | Hardware Architecture

PLASMA
UT: Dongarra



EXASCALE COMPUTING PROJECT

Or Formulate as Sparse Matrix Problem

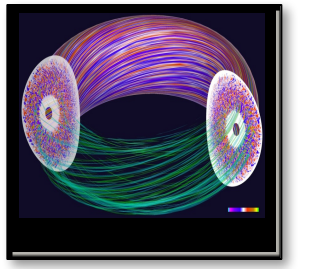
(Use Many-threading Model for Solver)

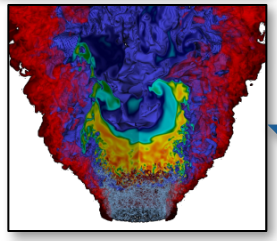
What is the best model to map computation onto underlying parallel hardware?

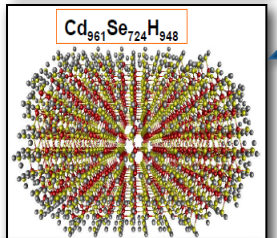
Numerical Model

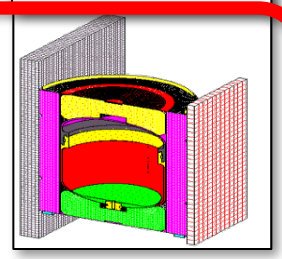
Algorithm

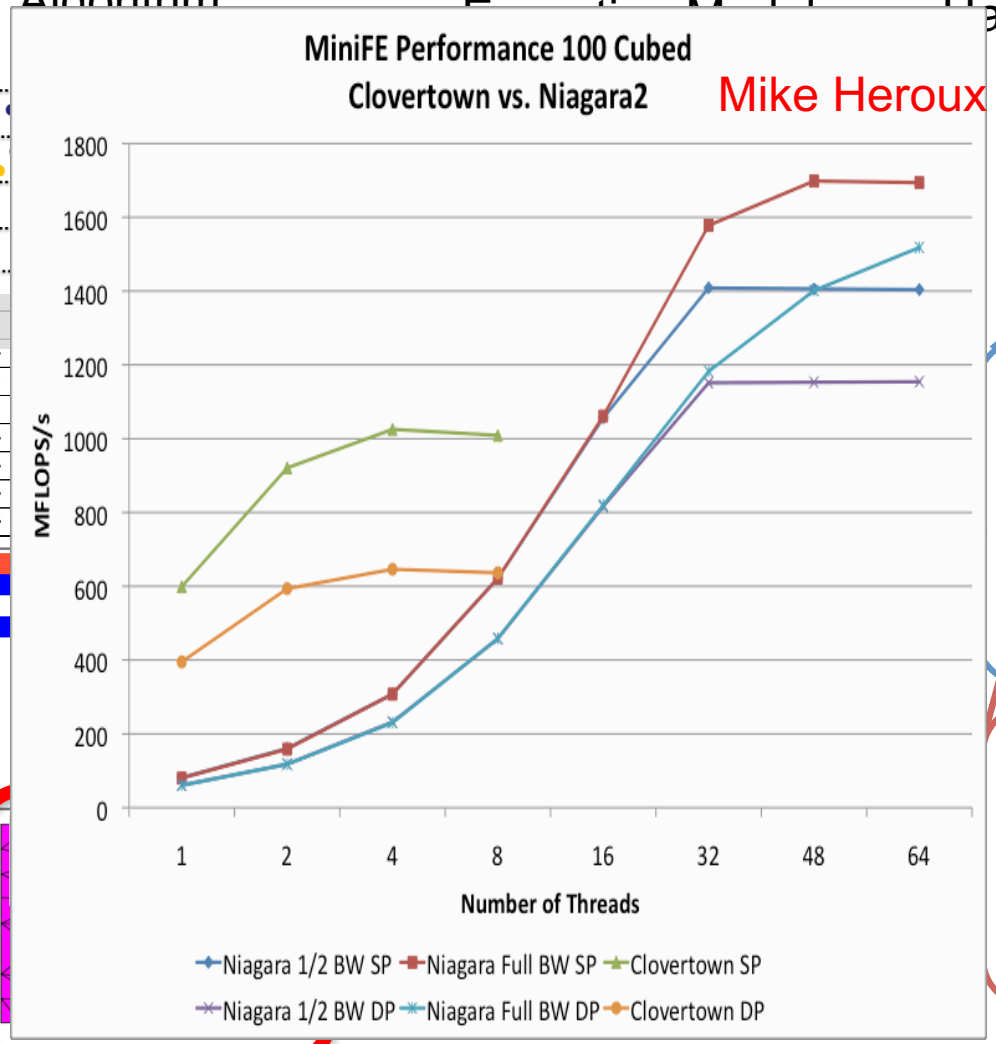
Hardware Architecture

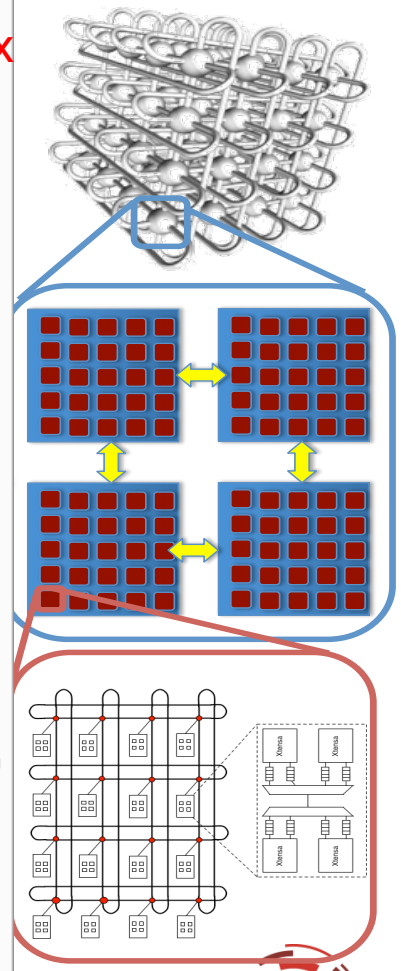












EXASCALE COMPUTING PROJECT