Implications of Hardware Trends on Programming Models

John Shalf

& the Computer Architecture Lab team

ExaComm: First International Workshop on Communication Architectures at Extreme Scale

June 26, 2015

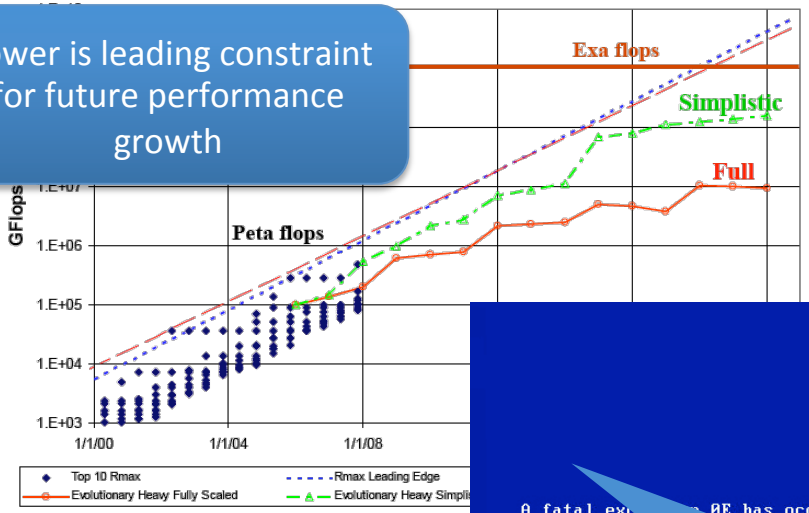
BERKELEY LAB

U.S. DEPARTMENT OF ENERGY | Office of Science

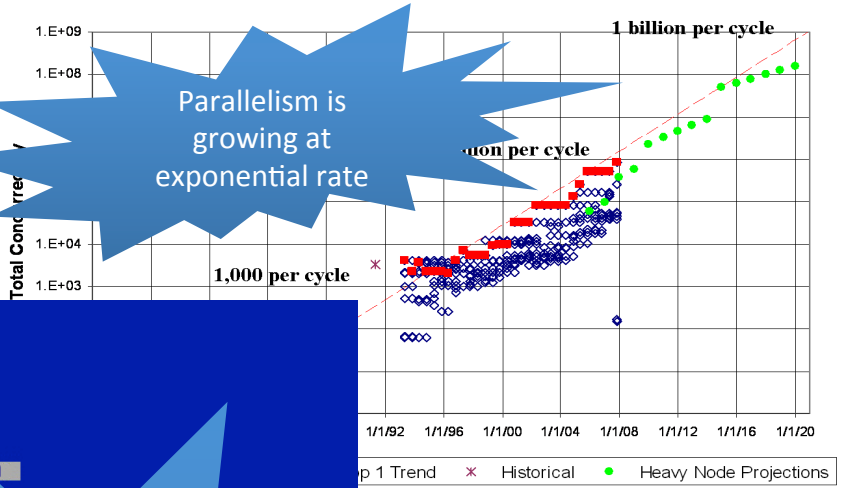
Sandia National Laboratories

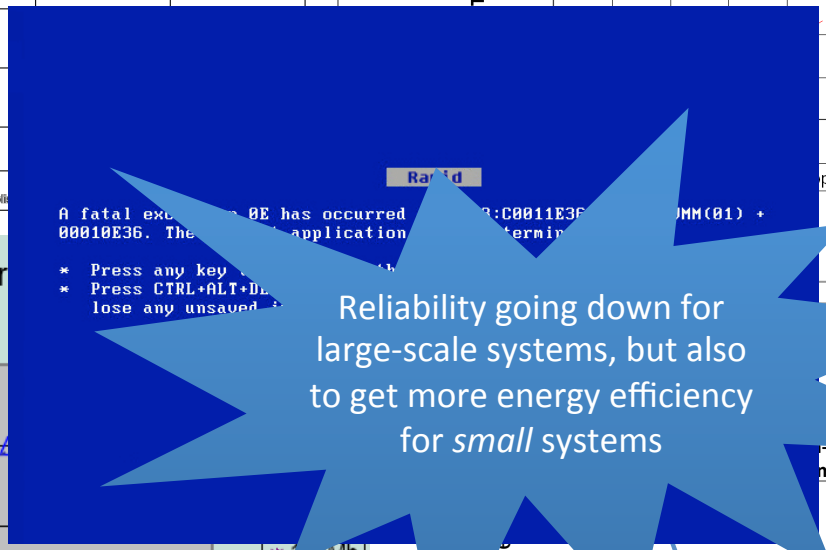
Technology Challenges for the Next Decade

Power is leading constraint for future performance growth

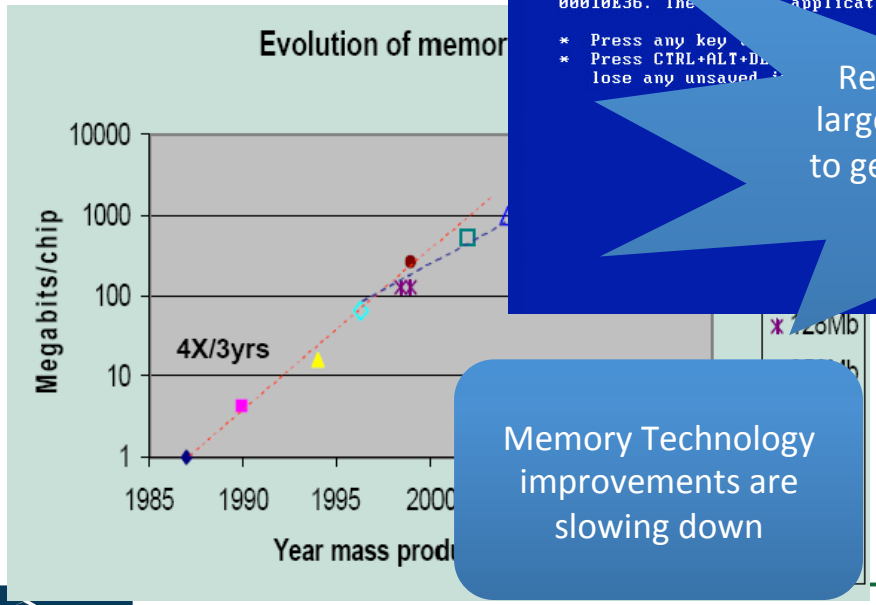


Parallelism is growing at exponential rate

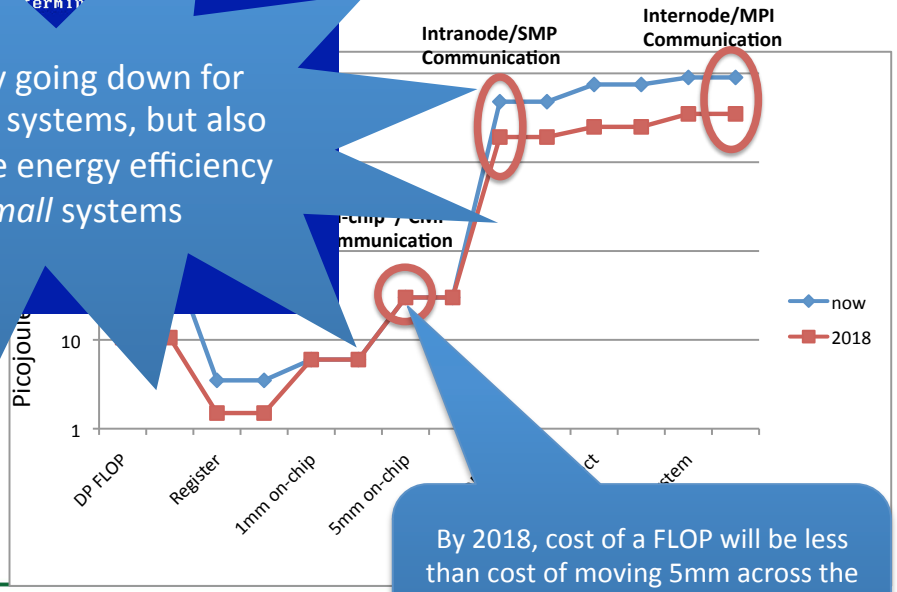




Reliability going down for large-scale systems, but also to get more energy efficiency for *small* systems



Memory Technology improvements are slowing down



By 2018, cost of a FLOP will be less than cost of moving 5mm across the chip's surface (locality will *really* matter)

Exascale Computing Trends: Adjusting to the "New Normal" for Computer Architecture

With two decades of data in hand about supercomputer performance, now is the time to take stock and look forward in terms of scaling models and their implications for future systems.

1521-9615/13/$31.00 © 2013 IEEE
COPUBLISHED BY THE IEEE CS AND THE AIP

PETER KOGGE
University of Notre Dame
JOHN SHALF
Lawrence Berkeley National Laboratory

We now have 20 years of data under our belt as to the performance of supercomputers against at least a single floating-point benchmark from dense linear algebra. Until approximately 2004, a single model of parallel programming—bulk synchronous using the message passing interface (MPI) model—was usually sufficient for translating complex applications into reasonable parallel programs.

In 2004, however, a confluence of events changed forever the architectural landscape that underpinned MPI. Figure 1 summarizes the effects of these changes in terms of the year-over-year compound annual growth rate (CAGR) of several key system characteristics. This data, taken from an average of the top 10 rankings reported by the TOP500 (www.top500.org), shows that sustained performance, in flops (floating point operations) per second, has grown consistently at about 1.9× per year. Before 2004, this growth came from a modest increase in the number of cores, coupled with substantial (50 percent or better per year) in core clock rate, and substantial gains in memory per core. After 2004, the growth in cores per year skyrocketed, while the average core clock growth disappeared, and memory per core even declined.

The first half of this article delves into the underlying reasons for these changes and what they mean to system architectures. The second half addresses the ramifications of these changes on our assumptions about technology scaling as well as their profound implications for programming and algorithm design in future systems.

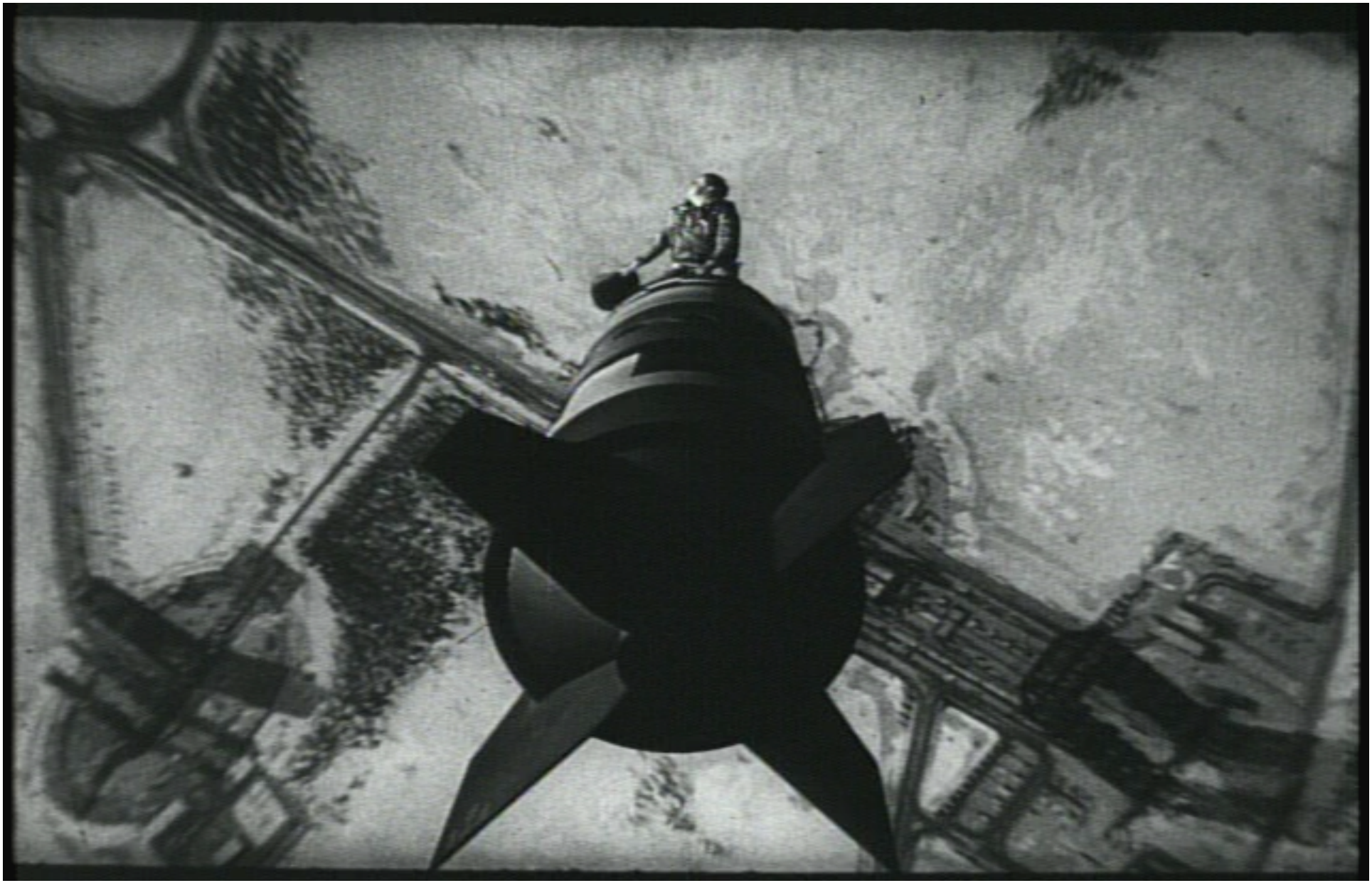
The Perfect Technological Storm

Moore's law has driven microprocessor architectures and high-performance computing (HPC) for decades. While variously interpreted as saying that microprocessor performance and memory chip density increase exponentially over time, the real statement is that a transistor's key linear dimensions (its *feature*

COMPUTER ARCHITECTURE LABORATORY

EXASCALE DESIGN SPACE EXPLORATION

How I learned to Stop Worrying and Love Exascale



Computer Architecture vs. Physics
Important not conflate one with the other

- **Physics (technological constraints)**
  - Cost of data movement
  - Capacity of DRAM cells
  - Clock frequencies (constrained by end of Dennard scaling)
  - Speed of Light
  - Melting point of silicon
- **Computer Architecture (design of the machine)**
  - Power management
  - ISA / Multithreading
  - SIMD widths

"Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints." – *Fred Brooks (IBM, 1962)*

Have converted many former "power" problems into "cost" problems

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

http://www.cal-design.org/publications

Abstract Machine Models and Proxy Architectures for Exascale Computing

Rev 1.1

J.A. Ang[1], R.F. Barrett[1], R.E. Benner[1], D. Burke[2], C. Chan[2], D. Donofrio[2], S.D. Hammond[1], K.S. Hemmert[1], S.M. Kelly[1], H. Le[1], V.J. Leung[1], D.R. Resnick[1], A.F. Rodrigues[1], J. Shalf[2], D. Stark[1], D. Unat[2], N.J. Wright[2]

Sandia National Laboratories, NM[1]
Lawrence Berkeley National Laboratory, CA[2]

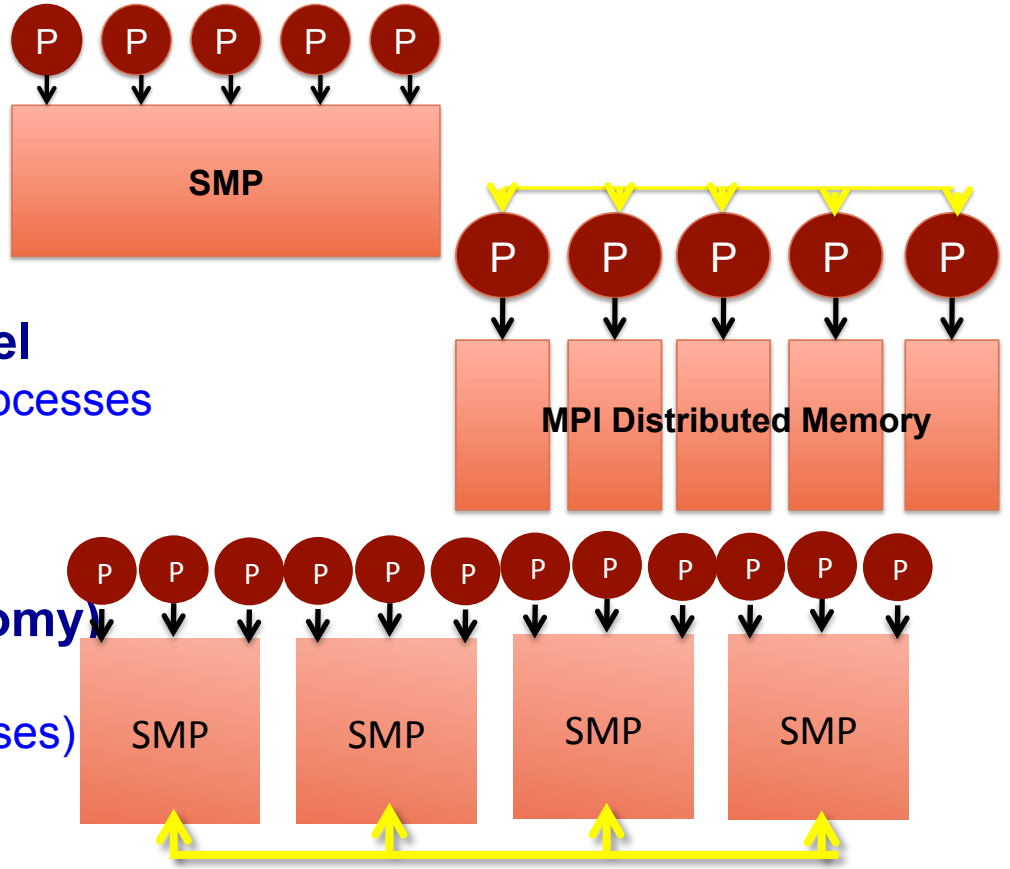
May, 16 2014

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

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
  - No unified memory
  - int [localNX][localNY][localNZ];
- **2-level (CTA in Martha Kim Taxonomy)**
  - Candidate Type Architecture (CTA)
  - MPI+X model (for all practical purposes)



- **Whats Next?** ?

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

BERKELEY LAB

Sandia National Laboratories

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**

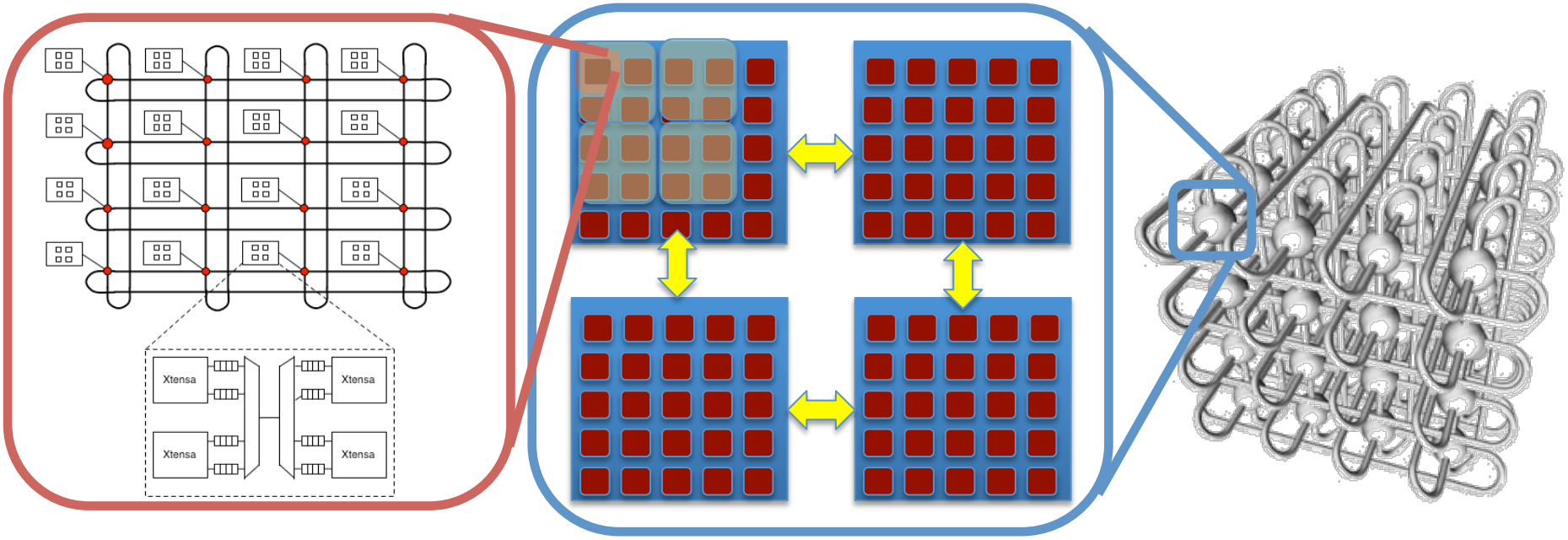
- **Codesign is quantitative trade-offs analysis because in a cost and power constrained environment, you need to know what you are wiling to give up to get what you want.**
  - Easy to ask for more features or BW to be added to the machine
  - It is much harder to evaluate what you are willing to give 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.***

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

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

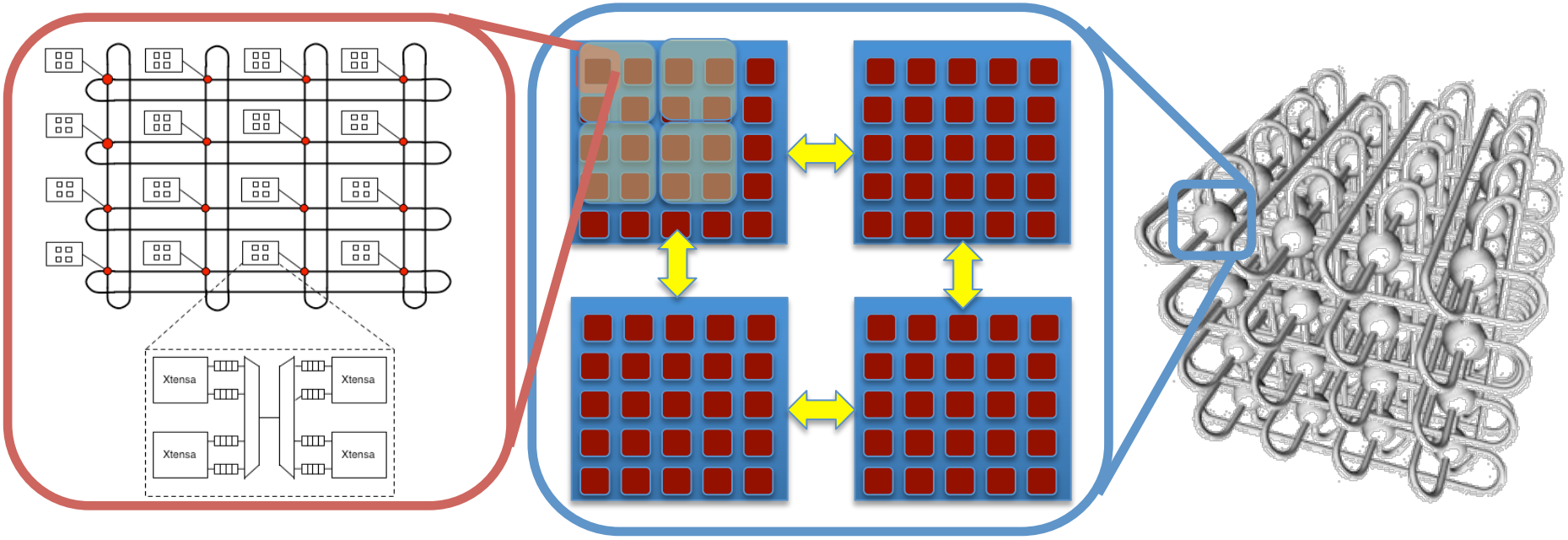
BERKELEY LAB

Sandia National Laboratories

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

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 it is well enough understood to support an automated mechanism to optimize layout or schedule***
  - 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

Exascale Strawman Arch

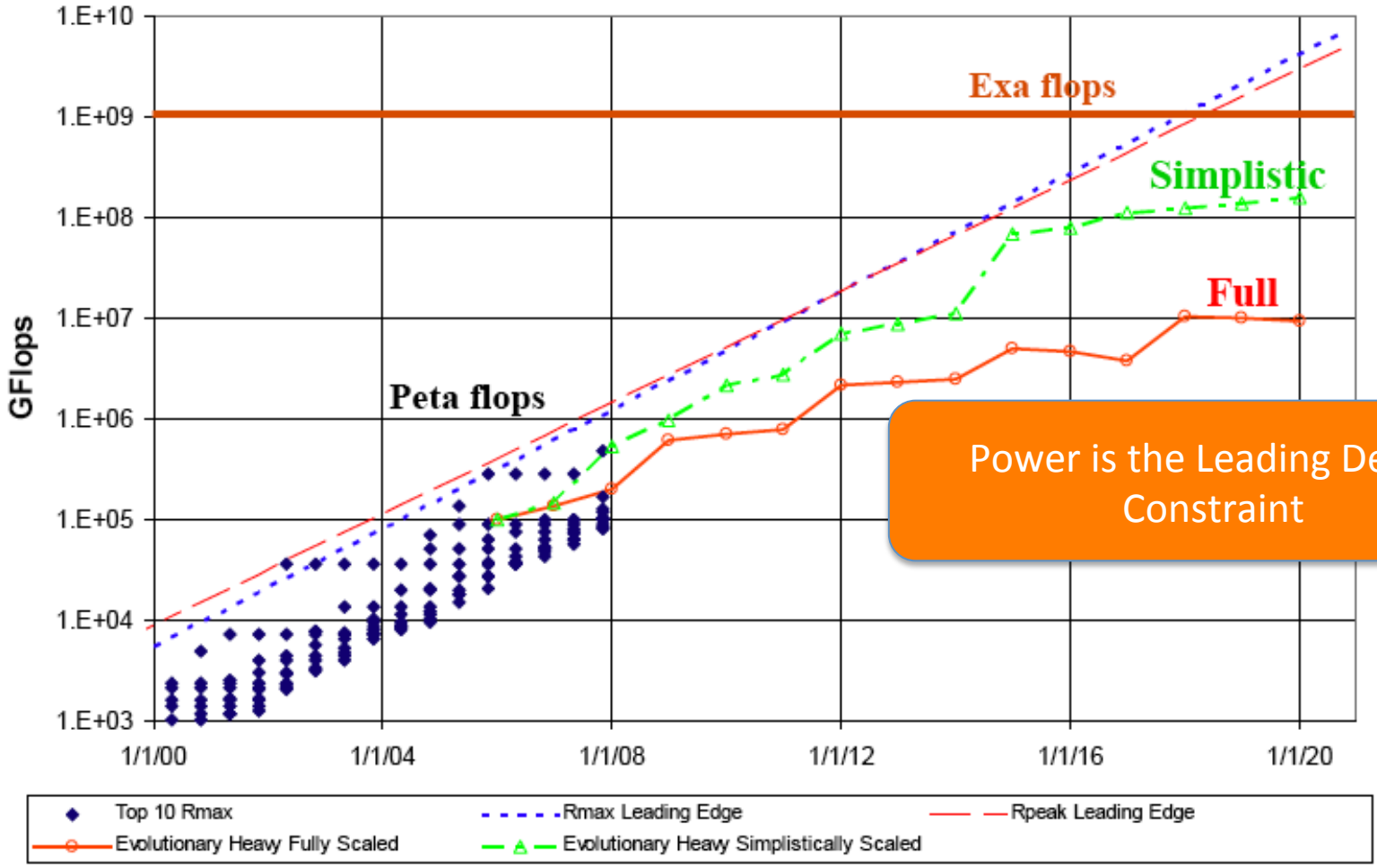
Based on input from DOE Fast Forward and Design Forward Projects

- ***Lets review where things are going in exascale concept designs***

Enduring Architecture Trends

- *Constraint*: **Can't scale the clock frequency**
  - ***Arch Response***: Only get performance from explicit parallelism
  - Have to include lightweight cores for max efficiency
- **Can't get both capacity and bandwidth in one memory technology**
  - Split of memory into fast-low-capacity and slow-high-capacity
  - NVRAM in this context is only there because low cost/bit
- **Communication overheads hurt strong scaling**
  - Integrate NIC on board the processor chip
  - Support light(er)weight messaging protocols (fewer steps)
- **A challenge to scale up parallel POSIX disk-based filesystem**
  - Burst-buffer hardware is here to stay (software for it unclear)
  - Whether it is on node or in I/O nodes (its slow enough that it looks the same)
- **Performance heterogeneity**
  - *Architecture response? (research question)*

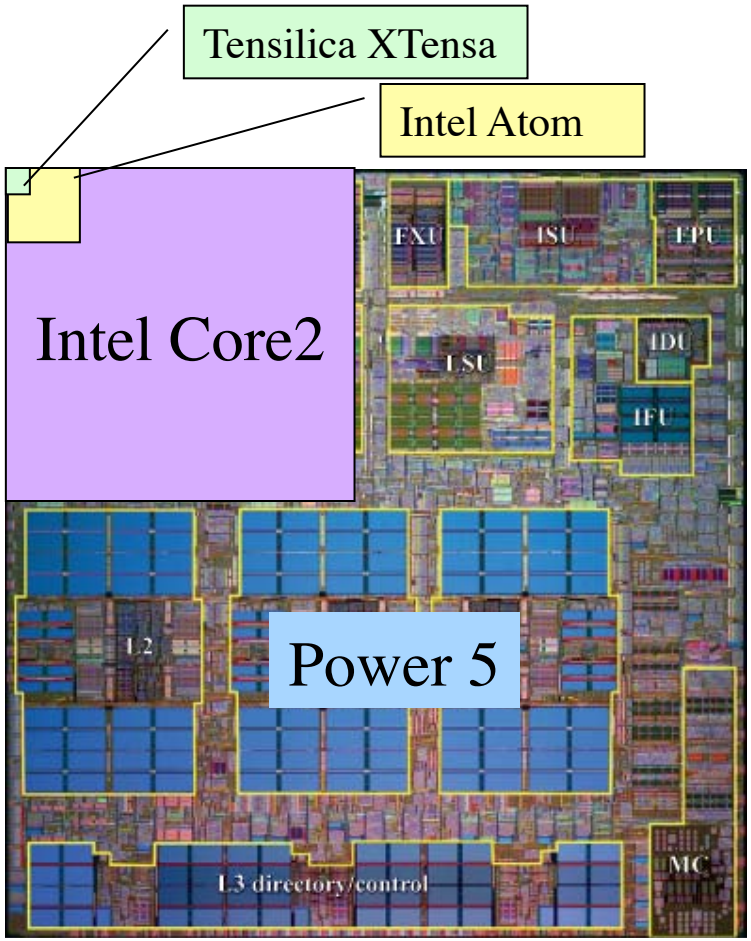
Current Technology Roadmaps will Depart from Historical Gains



Power is the Leading Design Constraint

From Peter Kogge, DARPA Exascale Study

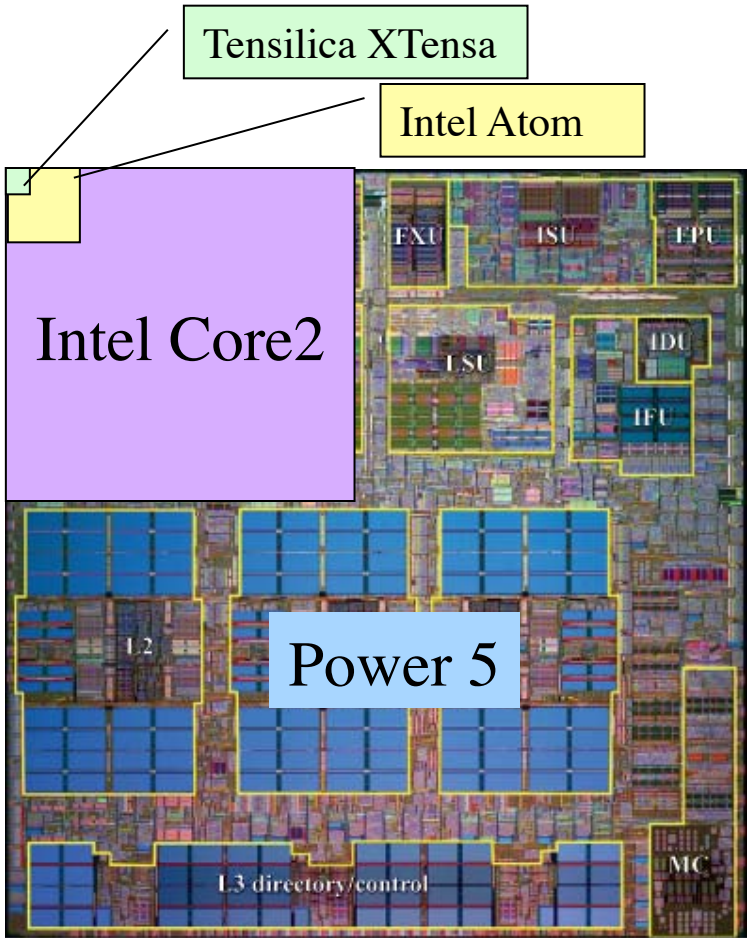
Low-Power Design Principles



- Cubic power improvement with lower clock rate due to V^2F
- Slower clock rates enable use of simpler cores
- Simpler cores use less area (lower leakage) and reduce cost
- Tailor design to application to REDUCE WASTE

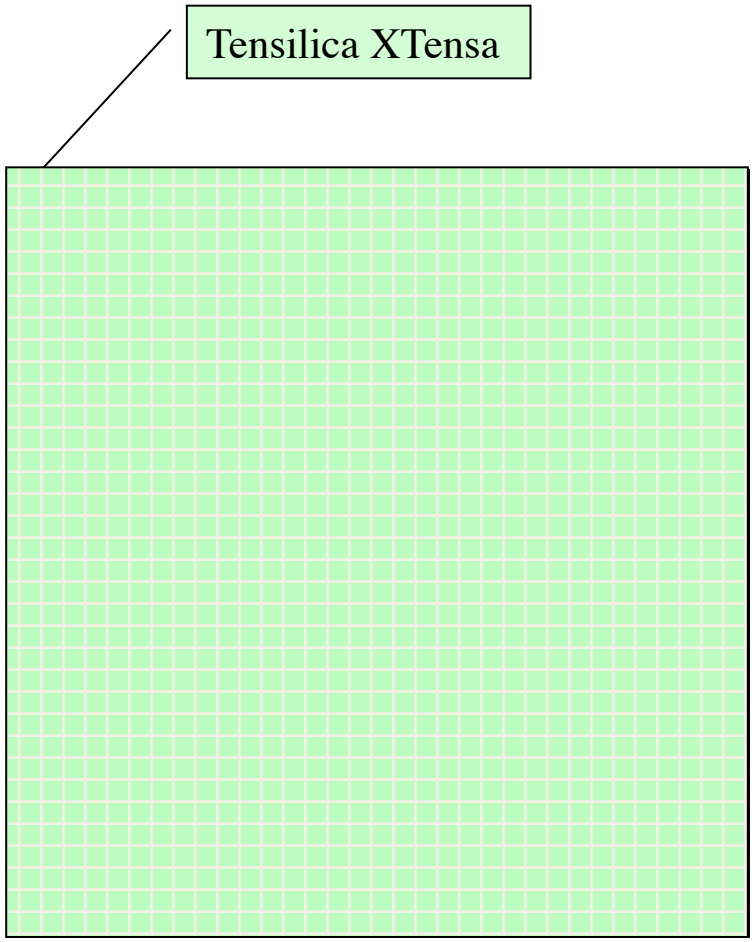
This is how iPhones and MP3 players are designed to maximize battery life and minimize cost

Low-Power Design Principles (2005)



- **Power5 (server)**
  - 120W@1900MHz
  - Baseline
- **Intel Core2 sc (laptop) :**
  - 15W@1000MHz
  - *4x more FLOPs/watt than baseline*
- **Intel Atom (handhelds)**
  - 0.625W@800MHz
  - 80x more
- **GPU Core or XTensa/Embedded**
  - 0.09W@600MHz
  - 400x more *(80x-120x sustained)*

Low Power Design Principles (2005)



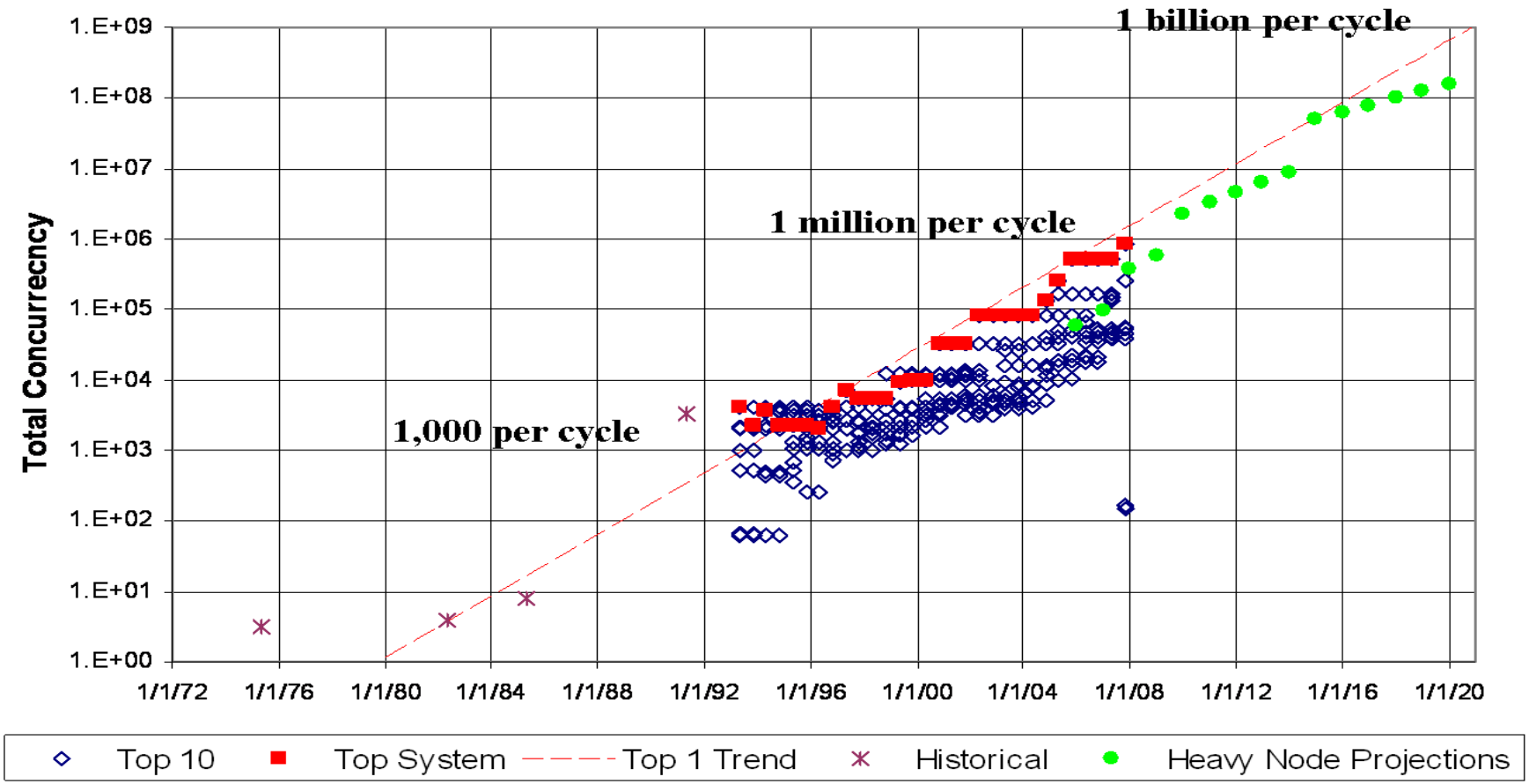
- **Power5 (server)**
  - 120W@1900MHz
  - Baseline
- **Intel Core2 sc (laptop) :**
  - 15W@1000MHz
  - *4x more FLOPs/watt than baseline*
- **Intel Atom (handhelds)**
  - 0.625W@800MHz
  - 80x more
- **GPU Core or XTensa/Embedded:**
  - 0.09W@600MHz
  - 400x more (80x-100x sustained)

Even if each simple core is 1/4th as computationally efficient as complex core, you can fit hundreds of them on a single chip and still be 100x more power efficient.

"The shift toward increasing parallelism is not a triumphant stride forward based on breakthroughs . . . parallel architecture; it is actually a retreat from even greater challenges that thwart efficient silicon implementation of traditional uniprocessor architectures." **Kurt Keutzer**

**"Landscape of Parallel Computing Research,
The View From Berkeley"
July 2006**

Projected Parallelism for Exascale *(2008 projection)*



How much parallelism must be handled by the program?

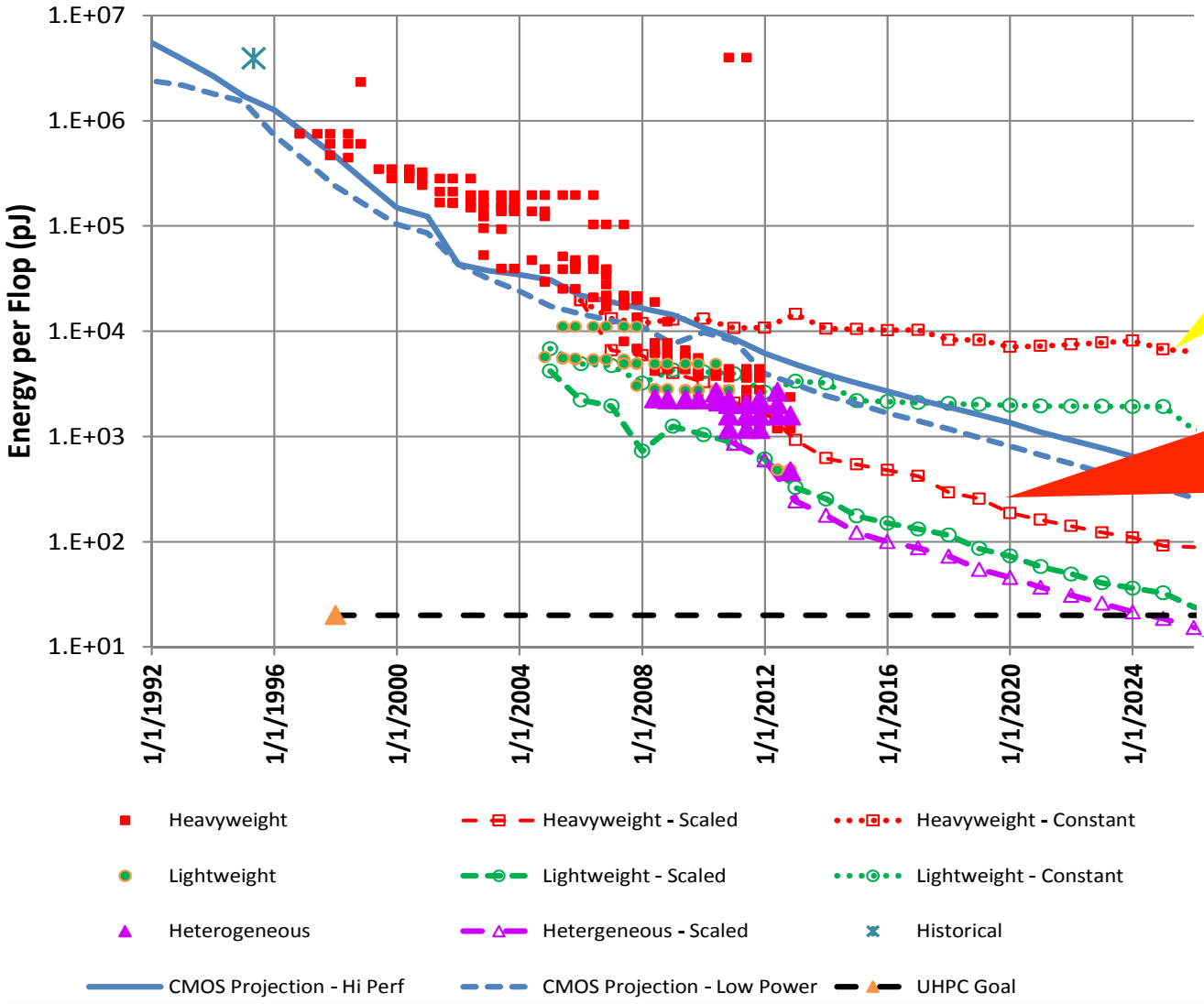
From Peter Kogge (on behalf of Exascale Working Group), "Architectural *Challenges* at the Exascale Frontier", June 20, 2008

Need 1Million-way parallelism to reach an Exaflop . . .

. . . . And possibly another 1000x just to hide latency

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Hybrid Architectures:
Moving from side-show to necessity



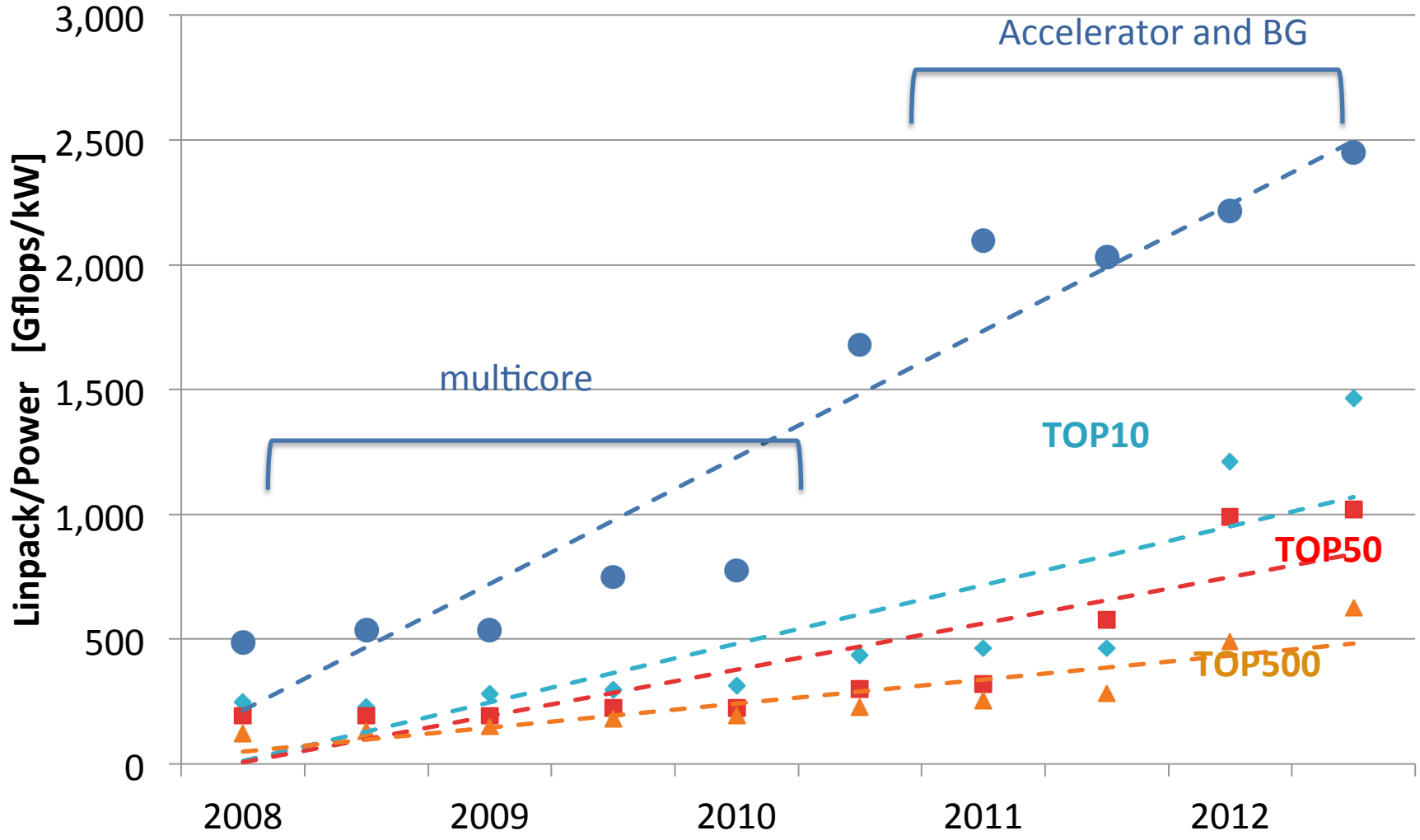
Can continue with conventional x86 architectures if you want.

We have gotten to 150PF by accepting lightweight cores (Not by arch breakthrough!!!) What trades will are you willing to make to get next 10x

Lightweight cores **OR** Hybrid is the only approach that crosses the exascale finish line

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Power Efficiency over Time

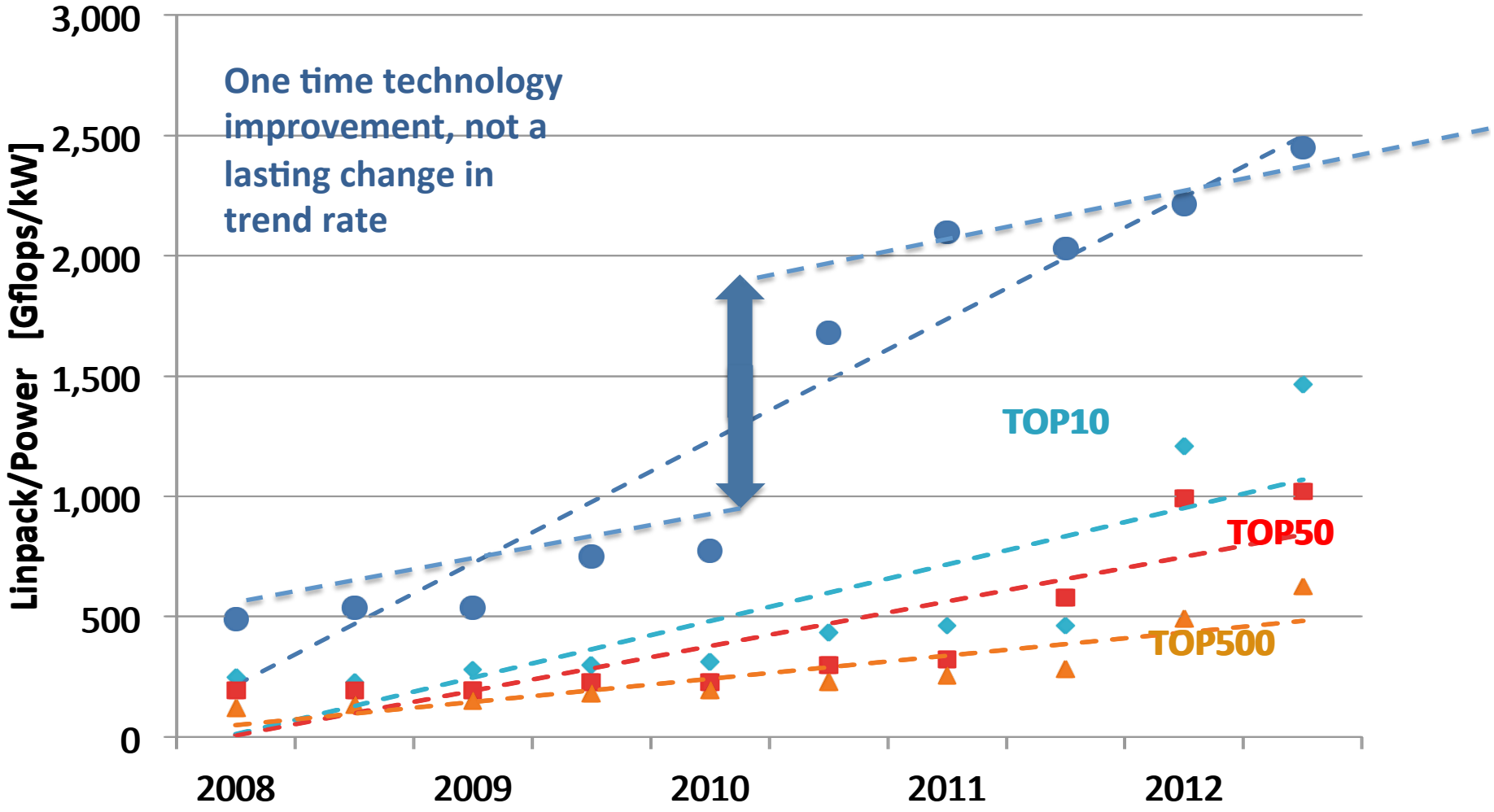


Data from: TOP500 November 2012

BERKELEY LAB

Sandia National Laboratories

Power Efficiency over Time



Data from: TOP500 November 2012

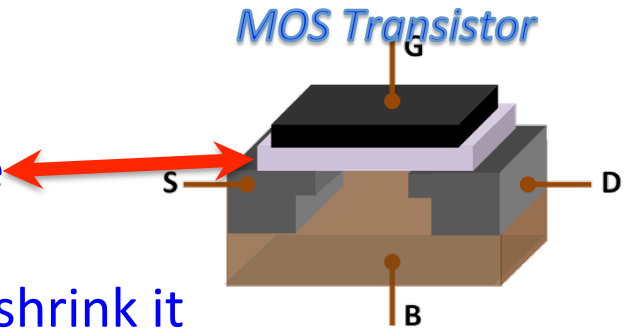
The problem with Wires:

Energy to move data proportional to distance

- **Energy Efficiency of copper wire:**
  - **Power = frequency * Length / cross-section-area**

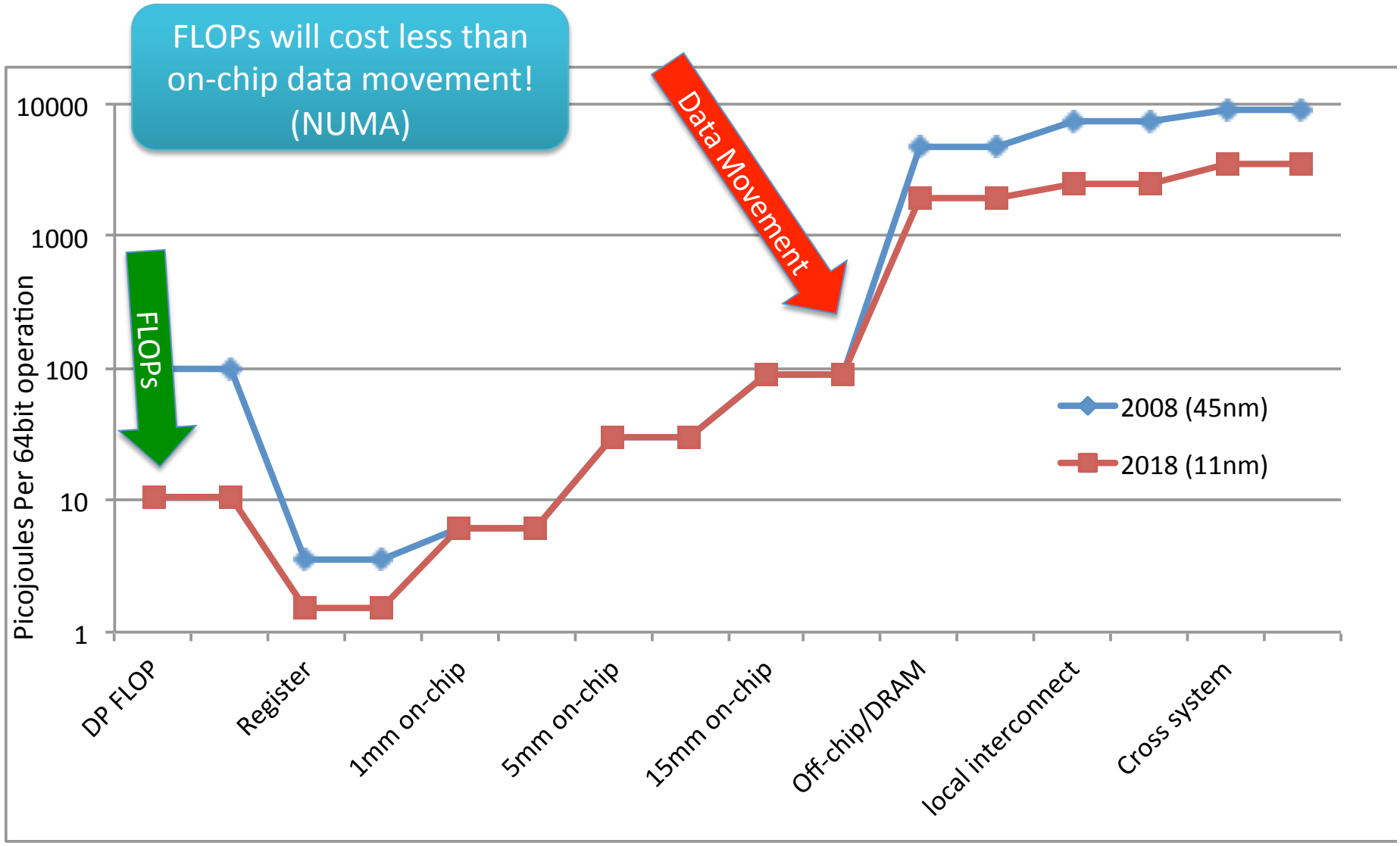
  - Wire efficiency *does not improve* as feature size shrinks

- **Energy Efficiency of a Transistor:**
  - Power = V^2 * frequency * Capacitance
  - Capacitance ~= Area of Transistor
  - Transistor efficiency improves as you shrink it



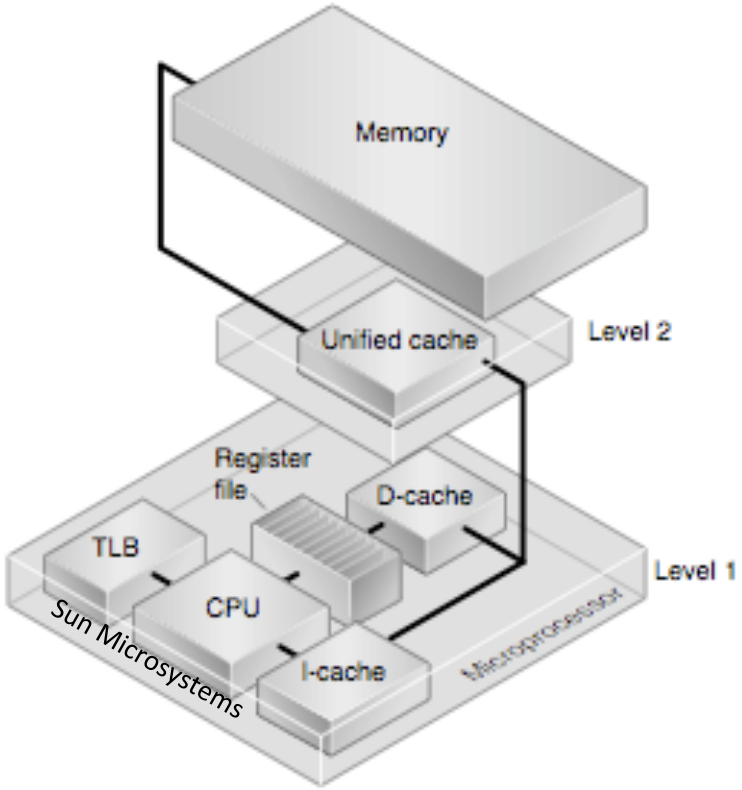
- ***Net result is that moving data on wires is starting to cost more energy than computing on said data***

Cost of Data Movement Increasing Relative to Ops

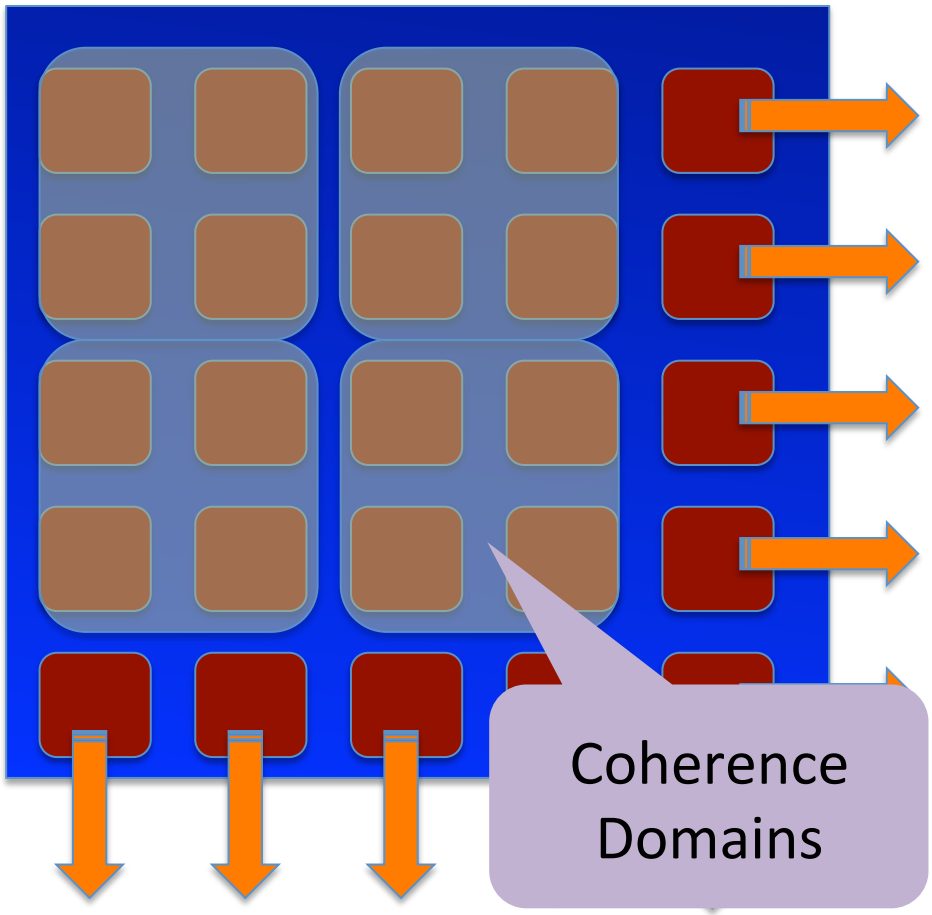


Data Locality Management

Vertical Locality Management
(spatio-temporal optimization)



Horizontal Locality Management
(topology optimization)

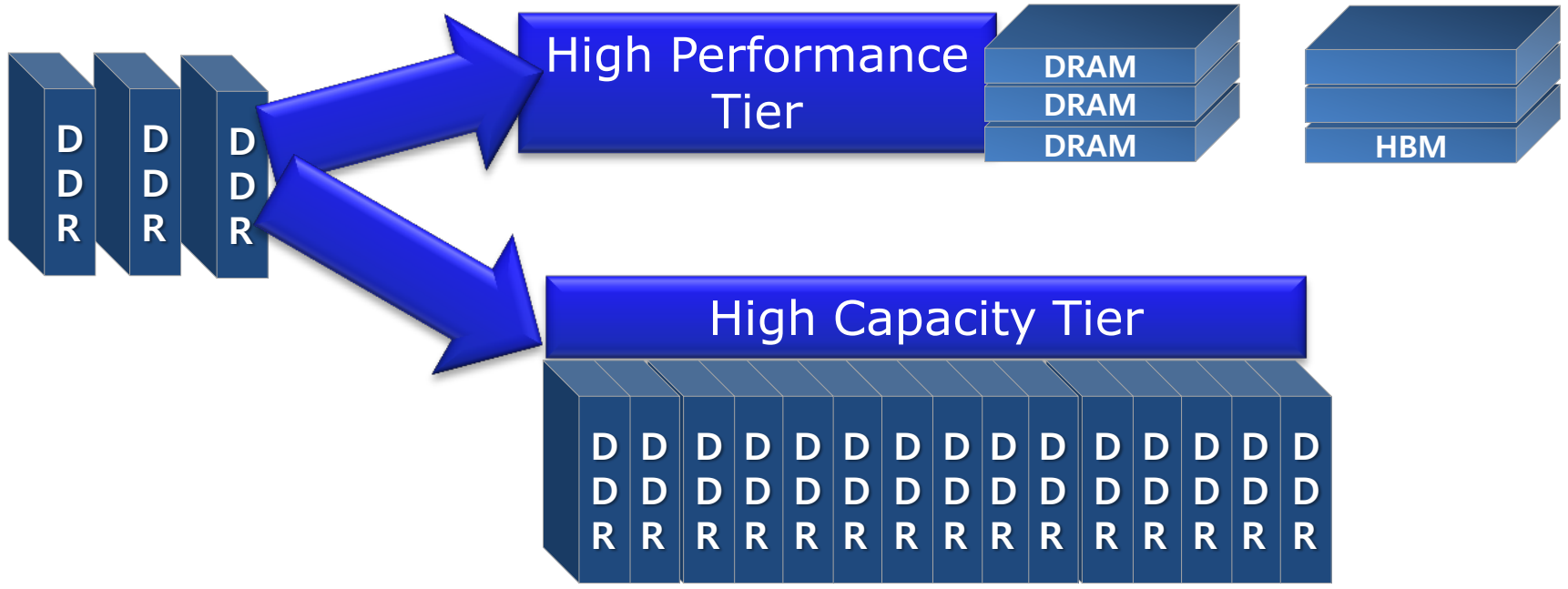


Can Get Capacity *OR* Bandwidth But Cannot Get Both in the Same Technology

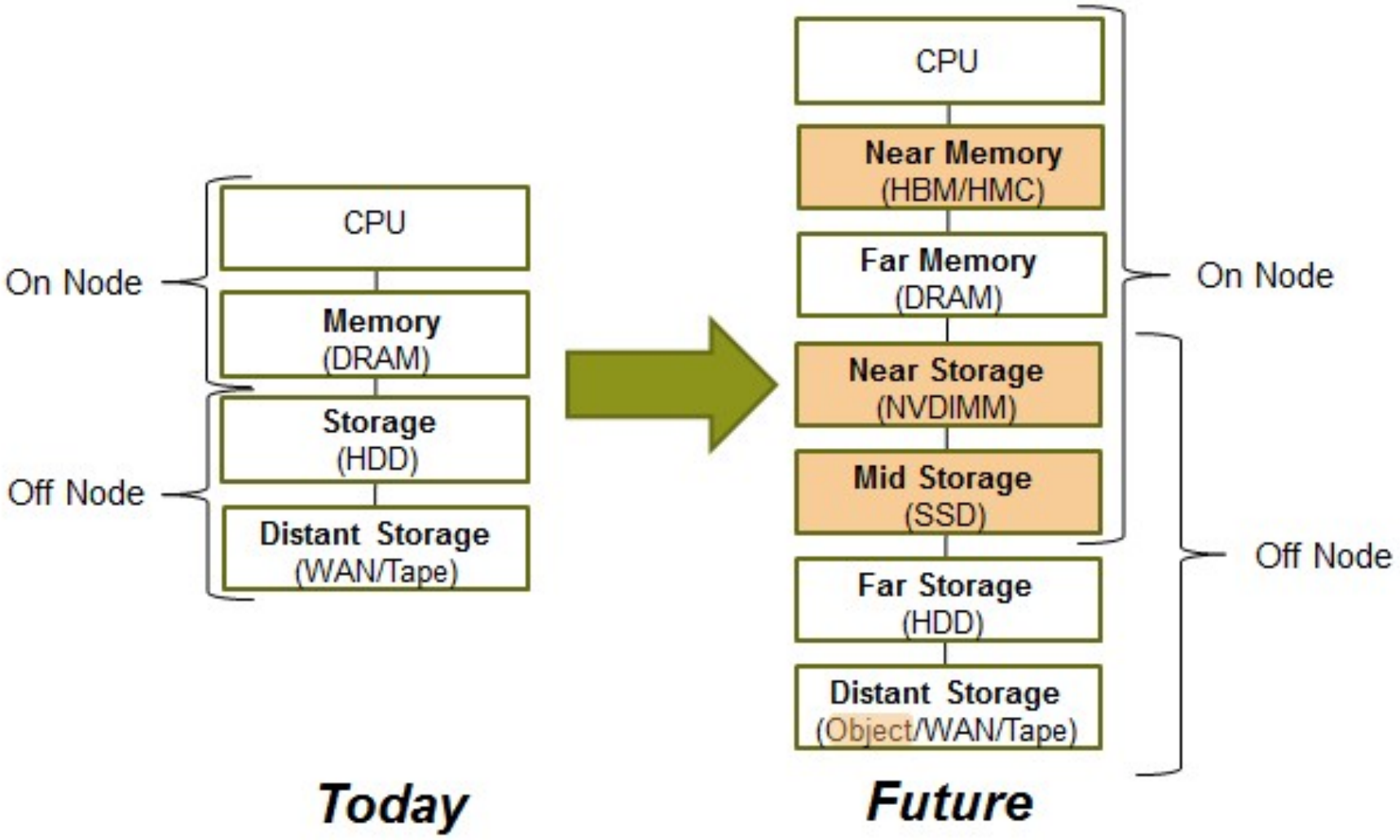
Cost (increases for higher capacity and cost/bit increases with bandwidth)

Power

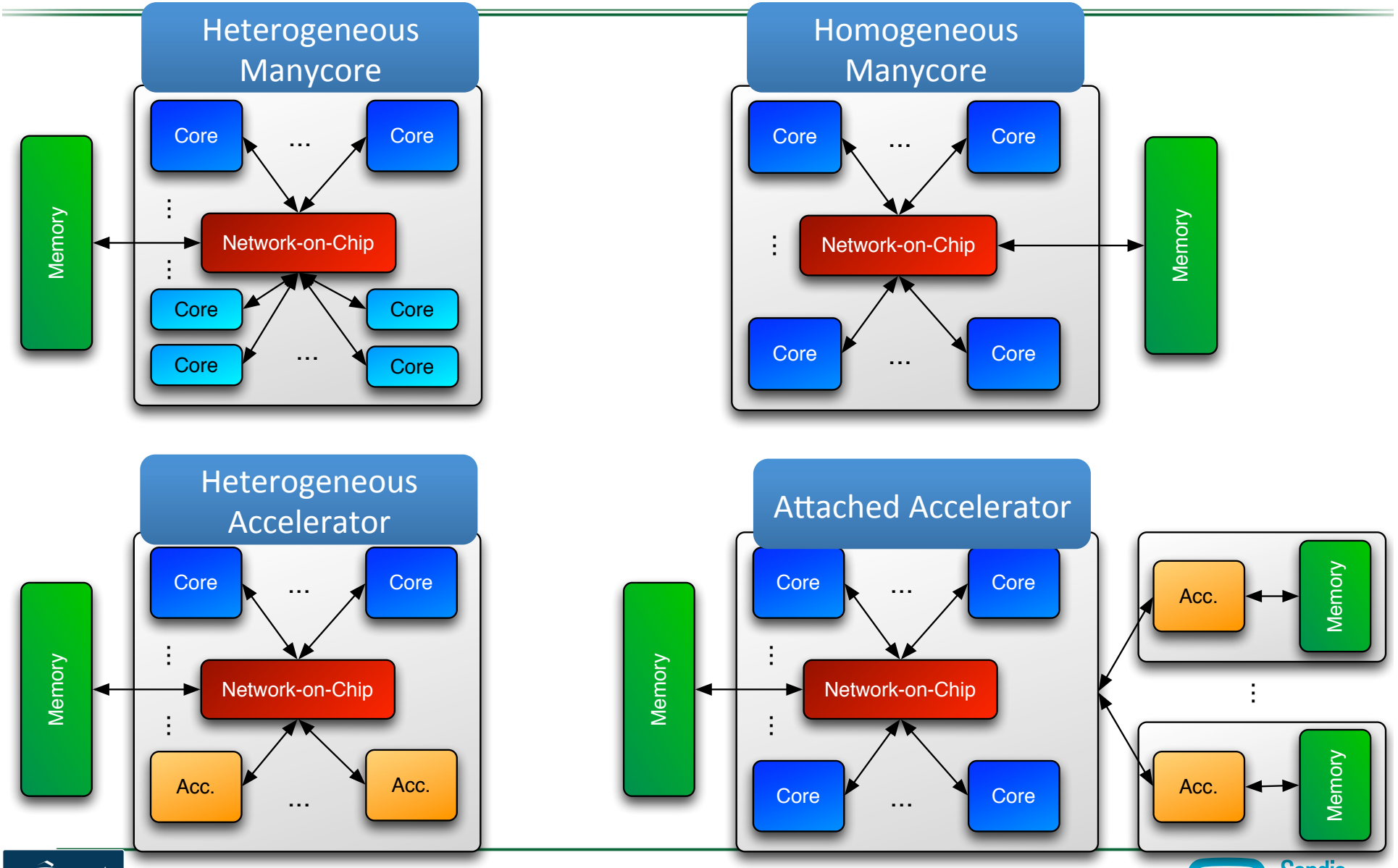
Bandwidth\Capacity	16 GB	32 GB	64 GB	128 GB	256 GB	512 GB	1 TB
4 TB/s							
2 TB/s	Stack/PNM						
1 TB/s			Interposer				
512 GB/s				HMC organic			
256 GB/s					DIMM		
128 GB/s							NVRAM



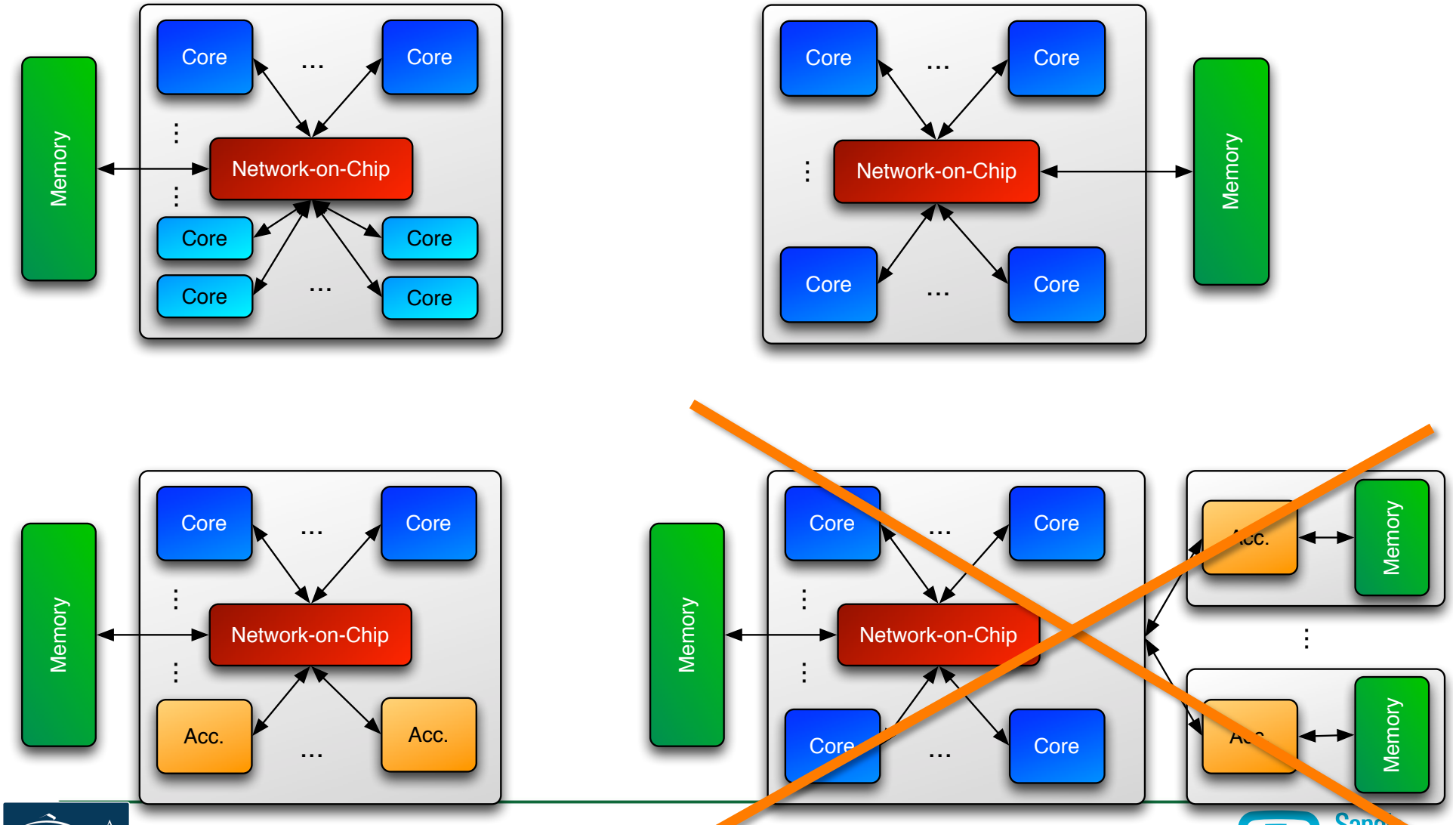
Trends in the Memory/Storage Subsystem



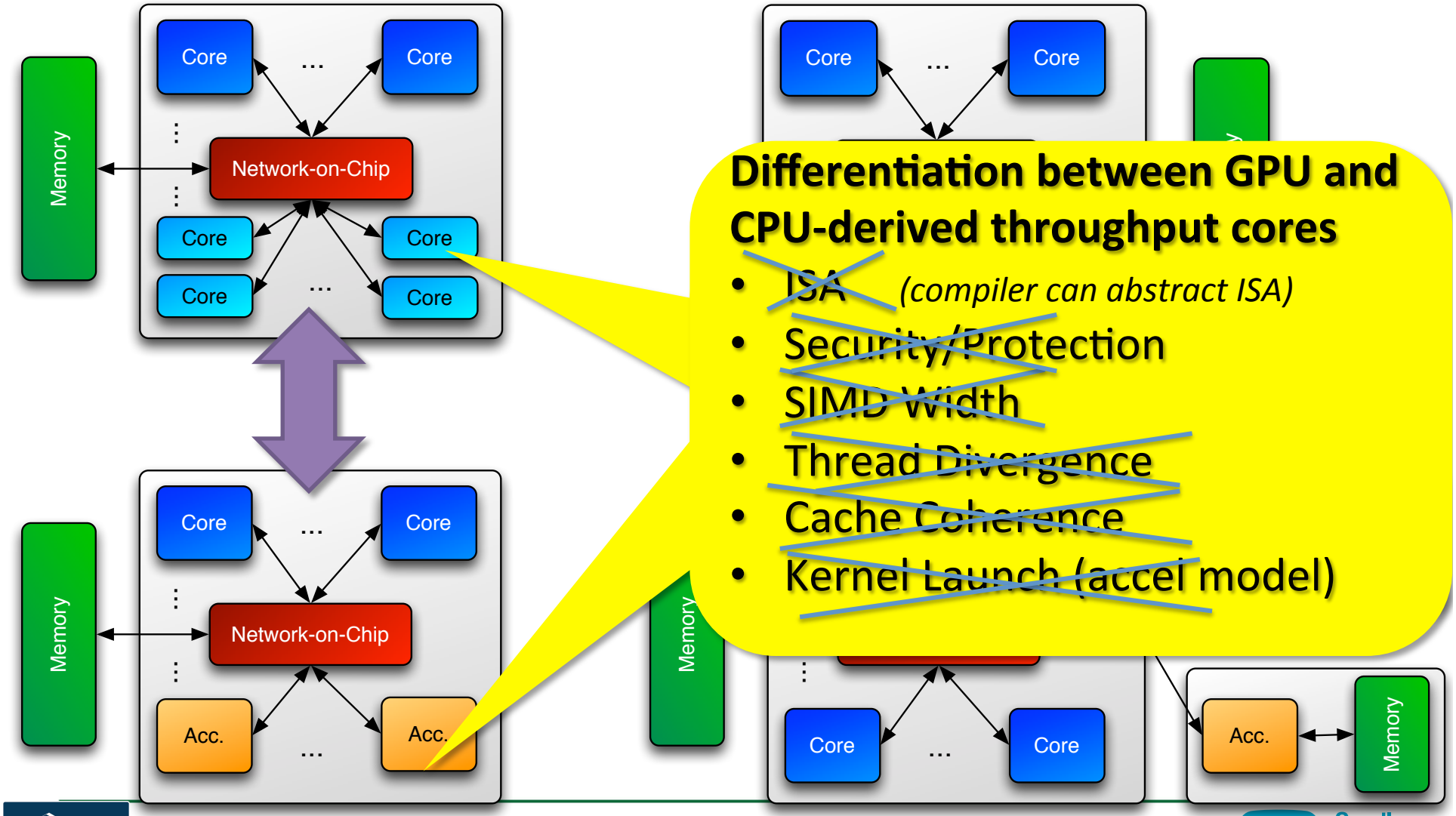
Families of AMMs



Families of AMMs



Families of AMMs

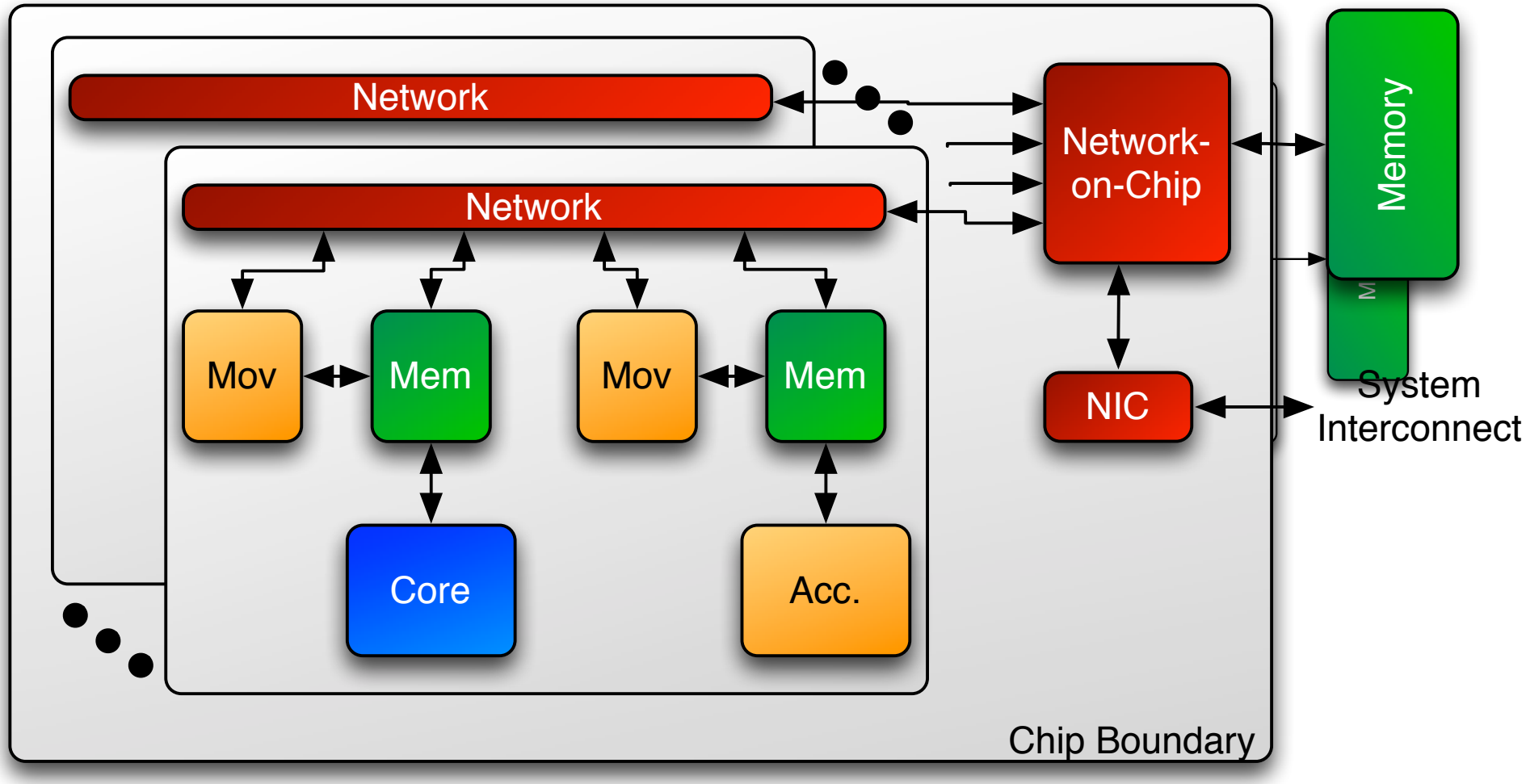


Differentiation between GPU and CPU-derived throughput cores

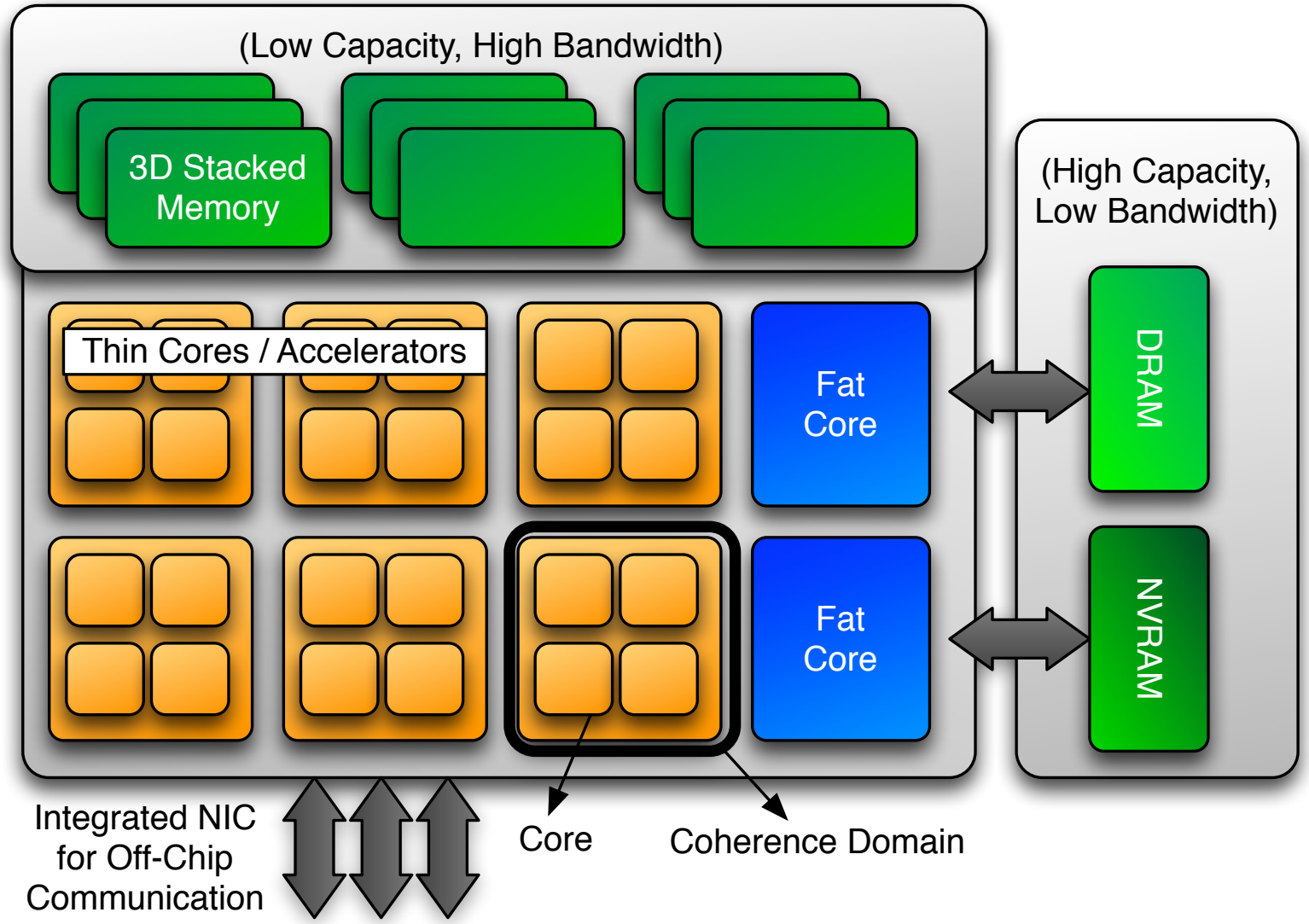
- ~~ISA~~ *(compiler can abstract ISA)*
- ~~Security/Protection~~
- ~~SIMD Width~~
- ~~Thread Divergence~~
- ~~Cache Coherence~~
- ~~Kernel Launch (accel model)~~

Are these the only possible AMMs?

***NO**: this is just a reflection of what is seen developing in industry. Specialization & other architectures possible. See Sandia XGC Project*

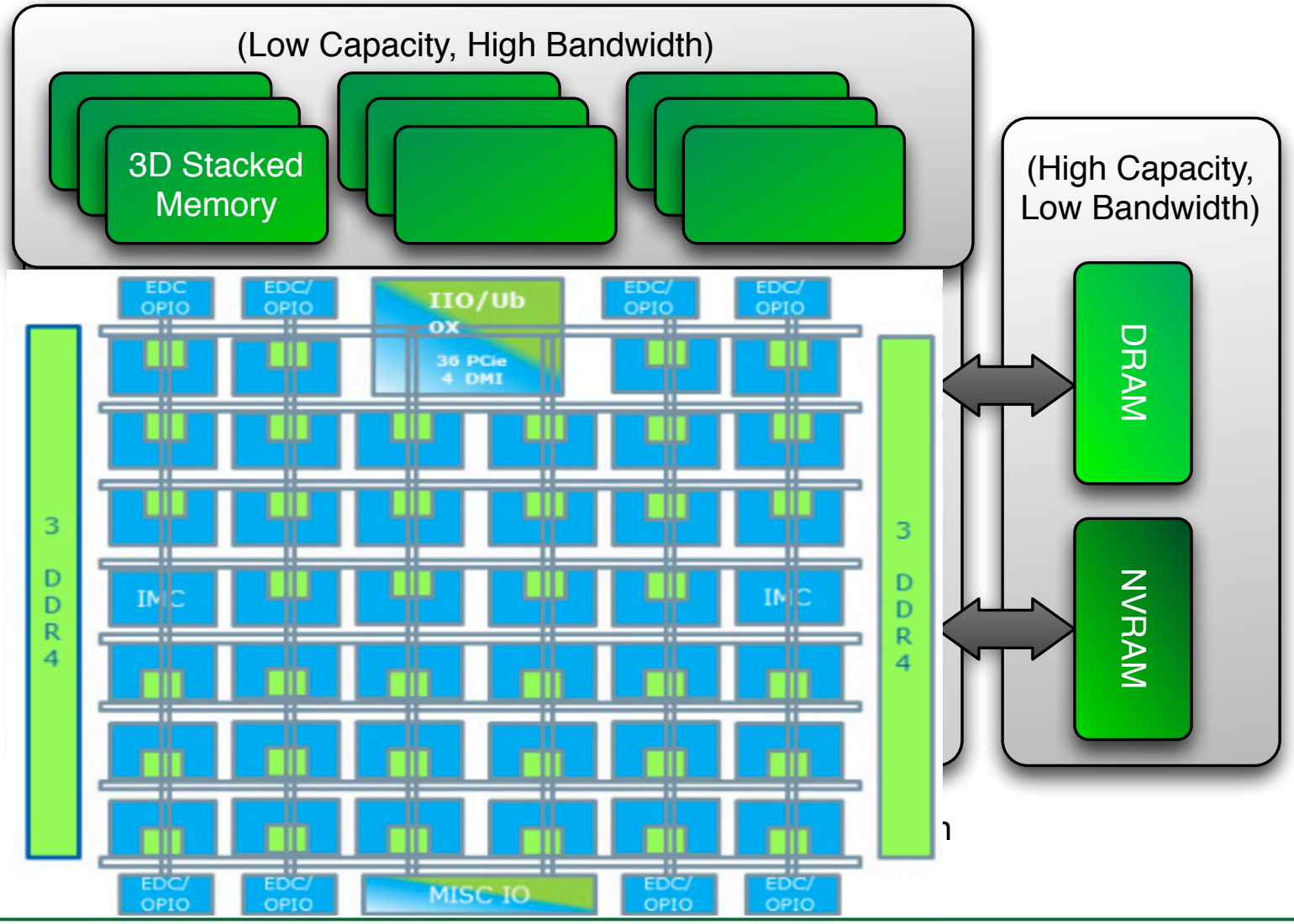


Abstract Machine Model

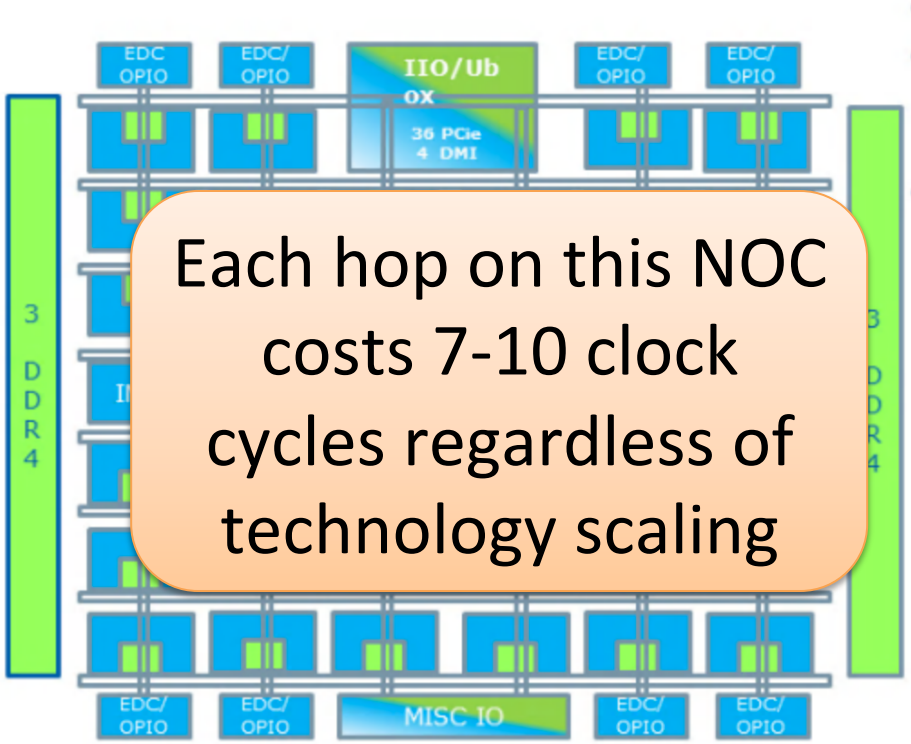


Exascale Node Schematic Model

(also for all pre-exascale systems)

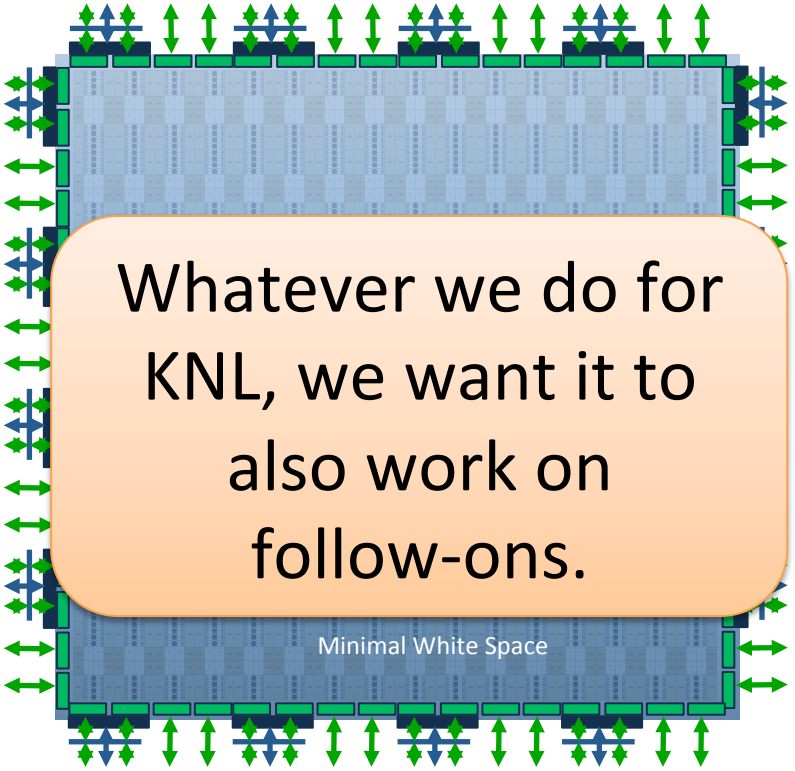


KNL Mesh
On-Chip Interconnect
36 islands (2015)
2 cores/island



Each hop on this NOC costs 7-10 clock cycles regardless of technology scaling

Future Mesh
On-Chip Interconnect
1024 islands (2022)
8 cores/island



Whatever we do for KNL, we want it to also work on follow-ons.

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Proxy Machine Model

		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 Latency	microseconds	1	0.4	0.02	0.02
Registers	KB/chip				

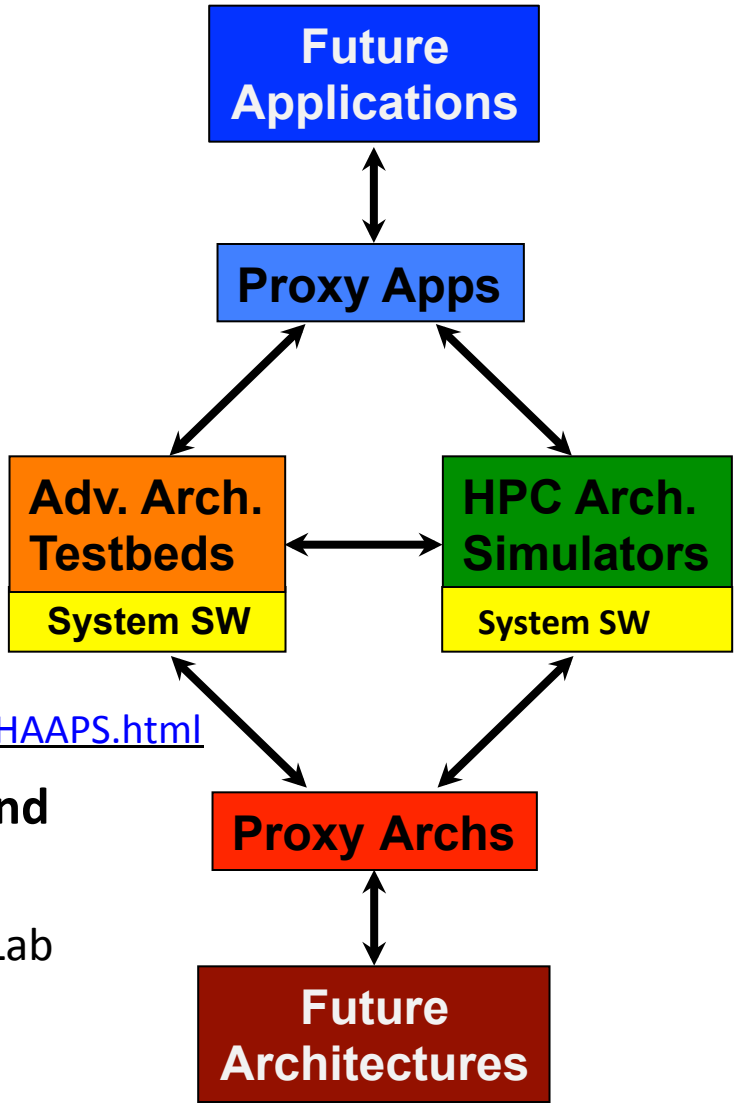
- exaNode1 and 2 are many core architectures
- exaNode1 uses commodity NIC and memory technology
- exaNode2 uses custom on-board NIC and faster memory technology
- exaPIM: Processing Near Memory, cache-less architecture

BERKELEY LAB

Sandia National Laboratories

Proxy Machines w/ Proxy Apps

- **Proxy Applications (Mantevo):**
  - Application source for architecture-centric optimization and analysis
  - http://mantevo.org
- **HPC Architectural Analysis Frameworks:**
  - http://www.cal-design.org/
  - http://www.opensocfabric.org/
  - http://SST-simulator.org
- **ASC Advanced Architecture Test Beds:**
  - Evolving examples of COTS "state-of-the-art"
  - http://www.sandia.gov/asc/computational_systems/HAAPS.html
- **Abstract Machine Model (AMM) Definitions and associated Proxy Architectures**
  - Supported by SC/ASCR Computer Architecture Lab
  - http://crd.lbl.gov/assets/pubs_presos/CALAbstractMachineModelsv1.1.pdf



AMMs vs. Proxy Machine Models

AMM is the topology and schematic for future machines

The Proxy Machine Model fills that in with speeds and feeds

	Processor Cores	Gflop/s per Proc Core	NoC BW per Proc Core (GB/s)	Processor SIMD Vectors (Units x Width)	Accelerator Cores	Acc Memory BW (GB/s)	Acc Count per Node	TFLOP/s per Node[1]	Node Count
Homogeneous M.C. Opt1	256	64	8	8x16	None	None	None	16	62,500
Homogeneous M.C. Opt2	64	250	64	2x16	None	None	None	16	62,500
Discrete Acc. Opt1	32	250	64	2x16	O(1000)	O(1000)	4	16C + 2A	55,000
Discrete Acc. Opt2	128	64	8	8x16	O(1000)	O(1000)	16	8C + 16A	41,000
Integrated Acc. Opt1	32	64	64	2x16	O(1000)	O(1000)	Integrated	30	33,000
Integrated Acc. Opt2	128	16	8	8x16	O(1000)	O(1000)	Integrated	30	33,000
Heterogeneous M.C. Opt1	16 / 192	250	64 / 8	8x16 / 2x8	None	None	None	16	62,500
Heterogeneous M.C. Opt2	32 / 128	64	64 / 8	8x16 / 2x8	None	None	None	16	62,500
Concept Opt1	128	50	8	12x1	128	O(1000)	Integrated	6	125,000
Concept Opt2	128	64	8	12x1	128	O(1000)	Integrated	8	125,000

Table 5.1: *Opt*1 and *Opt*1 represent possible proxy options for the abstract machine model. *M.C*: multi-core, *Acc*: Accelerator, *BW*: bandwidth, *Proc*: processor, For models with accelerators and cores, *C* denotes to FLOP/s from the CPU cores and *A* denotes to FLOP/s from Accelerators.

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 thread match/dispatch extensions
  - Or use the memory address for endpoint matching (GAS)
- **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?

Whats wrong with MPI3+OMP4

It will work,

But there are some things we could do better
(a LOT better)

Implications for Future Programming Models

What are the big challenges
for Future Programming Systems

BERKELEY LAB

U.S. DEPARTMENT OF ENERGY | Office of Science

Sandia National Laboratories

Emerging Technology Constraints
(we didn't design our codes with these in mind)

Old Constraints

- ***Peak clock frequency*** *as primary limiter for performance improvement*
- **Cost**: *FLOPs* are biggest cost for system: *optimize for compute*
- **Concurrency**: Modest growth of parallelism by adding nodes
- **Memory scaling***: maintain byte per flop capacity and bandwidth*
- **Locality***: MPI+X model (uniform costs within node & between nodes)*
- ***Uniformity****: Assume uniform system performance*
- **Reliability***: It's the hardware's problem*

New Constraints

- ***Power*** *is primary design constraint for future HPC system design*
- **Cost**: *Data movement dominates: optimize to minimize data movement*
- **Concurrency**: *Exponential growth of parallelism within chips*
- **Memory Scaling:** *Compute growing 2x faster than capacity or bandwidth*
- **Locality***: must reason about data locality and possibly topology*
- ***Heterogeneity****: Architectural and performance non-uniformity increase*
- **Reliability***: Cannot count on hardware protection alone*

Fundamentally breaks our current programming paradigm and computing ecosystem

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 completely rewrite our codes to fit future machine constraints says more about our programming environment than about the machines

(the programming env. targets the wrong abstractions)

How do you GET Performance Portability?

Better Abstractions for programming the underlying machine architecture

How do we build up these abstractions?

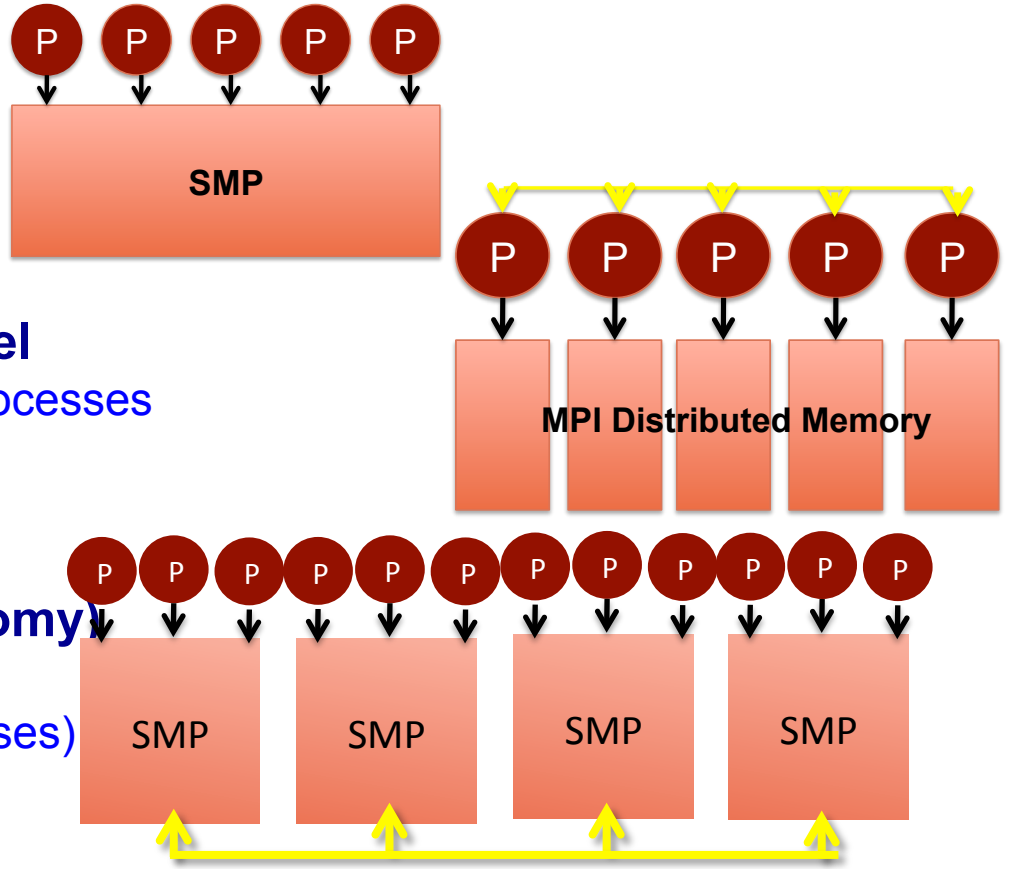
Start with an abstract machine model (AMM)

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

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
  - No unified memory
  - int [localNX][localNY][localNZ];
- **2-level (CTA in Martha Kim Taxonomy)**
  - Candidate Type Architecture (CTA)
  - MPI+X model (for all practical purposes)



- **Whats Next?** ?

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

BERKELEY LAB

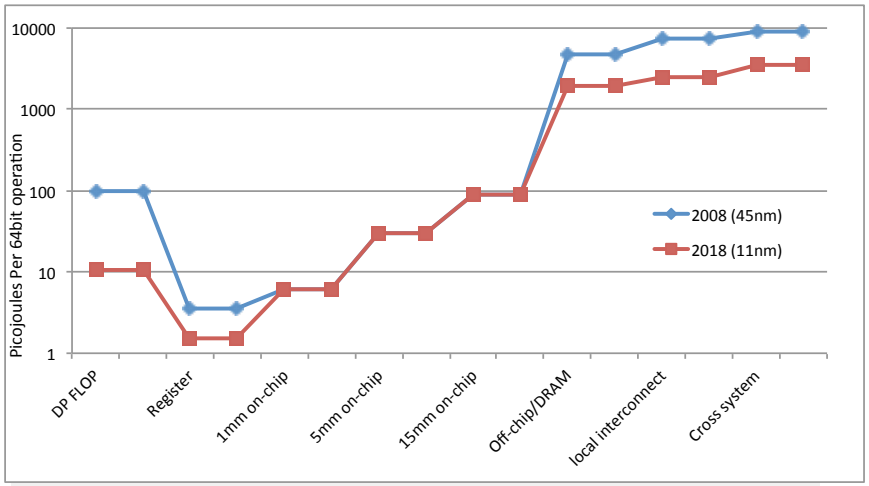
Sandia National Laboratories

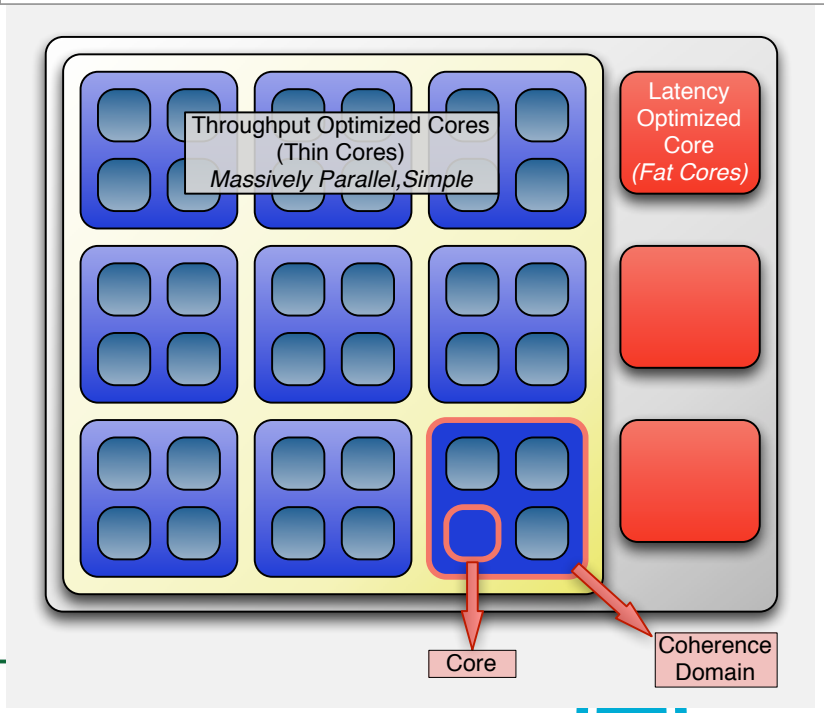
Durable Programming Abstractions
our current practices fail in this area

- **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)

Data Centric / Global Address Space

- **Motivation**
  - Data movement cost exceeds compute
  - Cost on-chip now distance dependent
  - Complexity of enumerating hundreds of cores (millions of MPI ranks)
- **Value Proposition**
  - Reduce cost of data movement (simpler compared to MPI 2-sided)
  - Data centric computation (compute on data where it is located… in-situ)
  - Make this all much simpler to describe
- **Implementations/Existence proofs**
  - UPC/UPC++:
  - Co-Array Fortran / CAF2:
  - RAJA/Kokkos: NNSA is putting majority of its investment behind this path.





Old Model (Parallel DO and life was good)

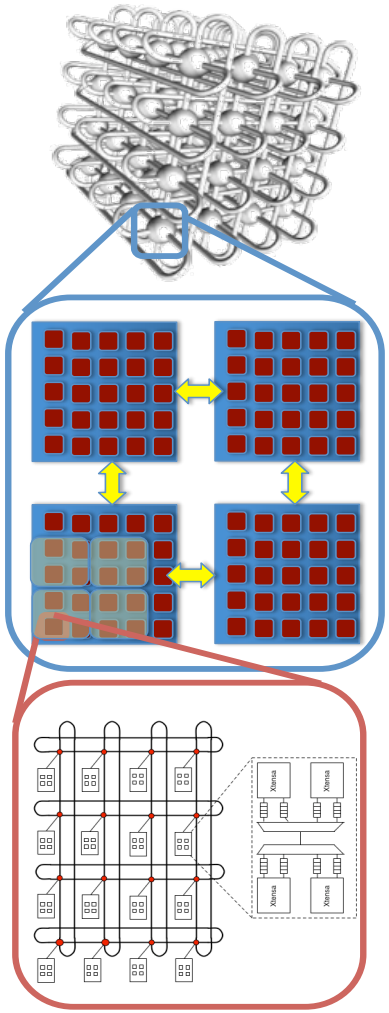
```
DO I=2,N
     B(I) = (A(I) + A(I-1)) / 2.0
ENDDO
```

Expressing Hierarchical Layout

- **Old Model (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
  - Loop statements operate on data where it is located
  - 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]);
```



COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

OpenACC Example (directives in red)
John Levesque presentation

CRAY THE SUPERCOMPUTER COMPANY

Keep data on the accelerator with acc_data region

```
!$acc data copyin(cix,ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10,ci11,&
!$acc& ci12,ci13,ci14,r,b,uxyz,cell,rho,grad,index_max,index,&
!$acc& ciy,ciz,wet,np,streaming_sbuf1, &
!$acc&    streaming_sbuf1,streaming_sbuf2,streaming_sbuf4,streaming_sbuf5,&
!$acc&    streaming_sbuf7s,streaming_sbuf8s,streaming_sbuf9n,streaming_sbuf10s,&
!$acc&    streaming_sbuf11n,streaming_sbuf12n,streaming_sbuf13s,streaming_sbuf14n,&
!$acc&    streaming_sbuf7e,streaming_sbuf8w,streaming_sbuf9e,streaming_sbuf10e,&
!$acc&    streaming_sbuf11w,streaming_sbuf12e,streaming_sbuf13w,streaming_sbuf14w, &
!$acc&    streaming_rbuf1,streaming_rbuf2,streaming_rbuf4,streaming_rbuf5,&
!$acc&    streaming_rbuf7n,streaming_rbuf8n,streaming_rbuf9s,streaming_rbuf10n,&
!$acc&    streaming_rbuf11s,streaming_rbuf12s,streaming_rbuf13n,streaming_rbuf14s,&
!$acc&    streaming_rbuf7w,streaming_rbuf8e,streaming_rbuf9w,streaming_rbuf10w,&
!$acc&    streaming_rbuf11e,streaming_rbuf12w,streaming_rbuf13e,streaming_rbuf14e, &
!$acc&    send_e,send_w,send_n,send_s,recv_e,recv_w,recv_n,recv_s)
  do ii=1,ntimes
        o o o
     call set_boundary_macro_press2
     call set_boundary_micro_press
     call collisiona
     call collisionb
     call recolor
```

OMP4 example (from OMP4 docs)

```
subroutine vec_mult(p, v1, v2, N)
   real    ::  p(N), v1(N), v2(N)
   integer ::  i
   call init(v1, v2, N)
   !$omp target data map(to: v1, v2) map(from: p)
   !$omp target
   !$omp parallel do
      do i=1,N
                     p(i) = v1(i) * v2(i)
      end do
   !$omp end target
   !$omp end target data
   call output(p, N)
end subroutine
```

And you have to do this for EVERY SINGLE LOOP!

```
#pragma omp target data if(N>THRESHOLD) map(from: p[0:N])
{
    #pragma omp target if (N>THRESHOLD) map(to: v1[:N], v2[:N])
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
    init_again(v1, v2, N);
    #pragma omp target if (N>THRESHOLD) map(to: v1[:N], v2[:N])
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = p[i] + (v1[i] * v2[i]);
}
```

So you think you can do better?

Yes.
Alternatives exist

… that have no users or commercial support …
We should work on that…

BERKELEY LAB

U.S. DEPARTMENT OF ENERGY | Office of Science

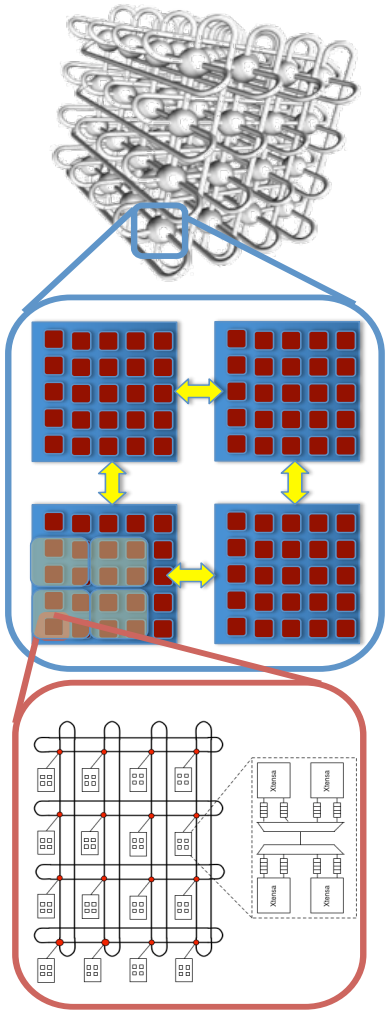
Sandia National Laboratories

Towards a Data Centric Computing Model

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

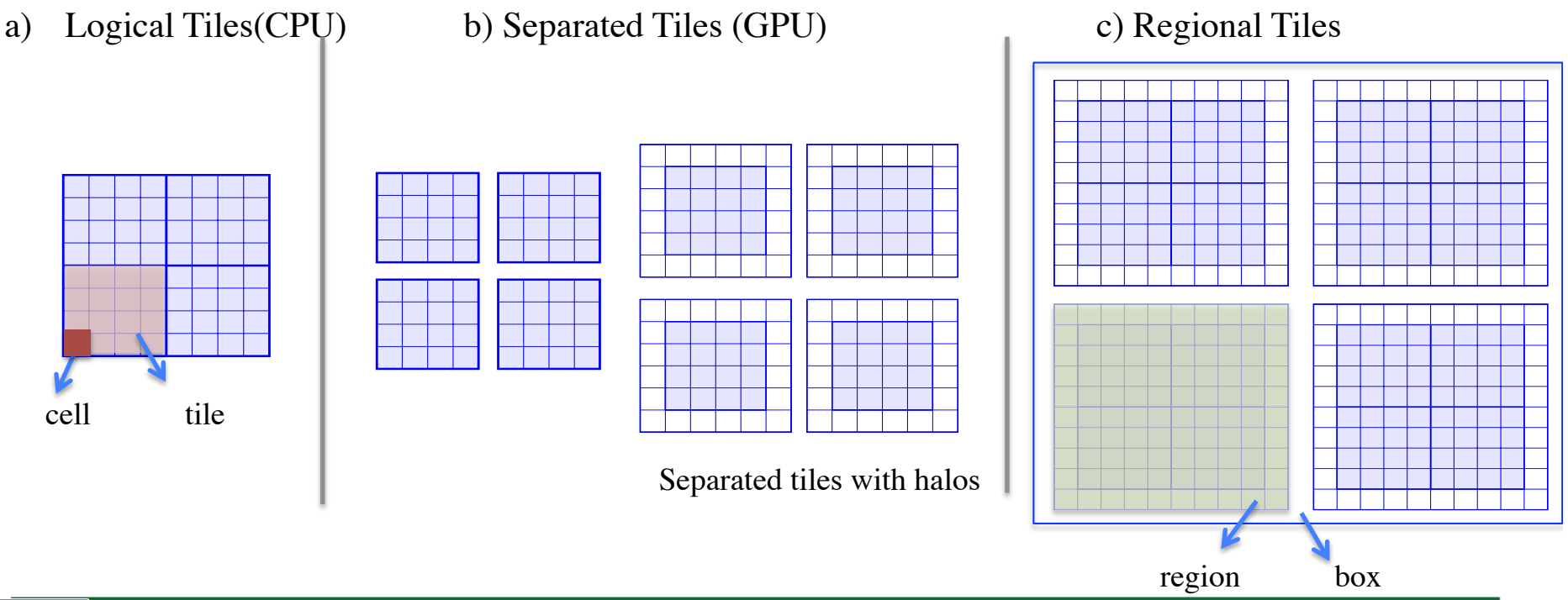
- **New Model (Data-Centric)** ***also in big data***
  - Describe how data is laid out in memory
  - Change applies to ALL Loop statements operate data where it is located (in-situ)
  - Similar to MapReduce, but need more sophisticated descriptions of data layout for scientific codes

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

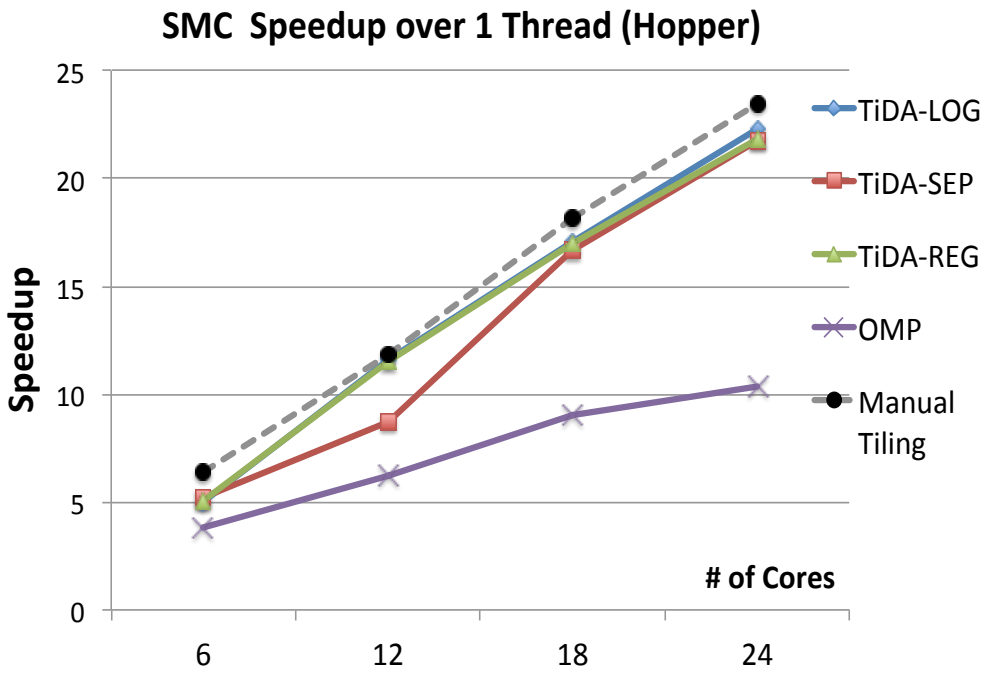


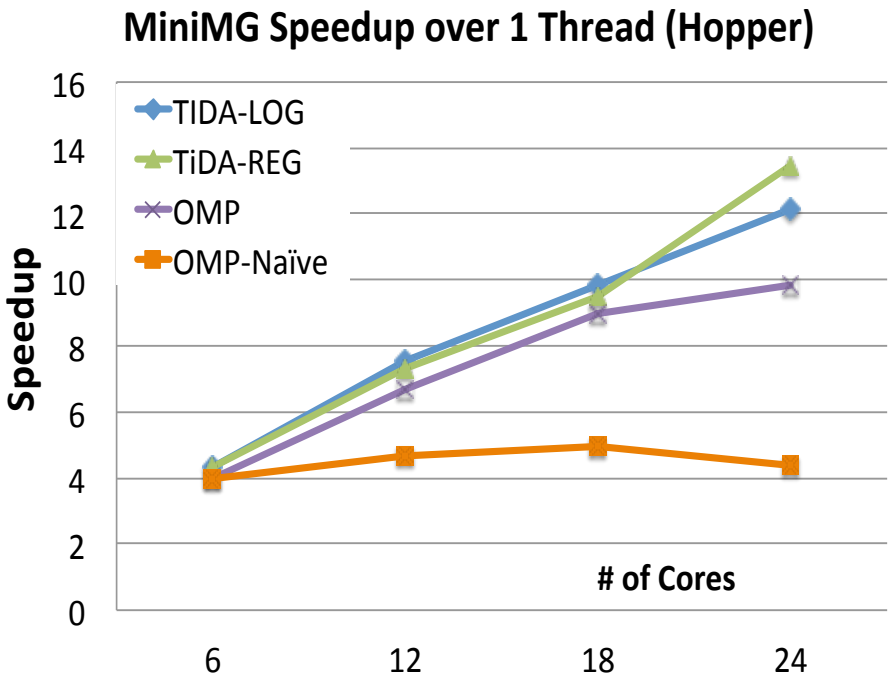
Tiling: Abstraction for Memory Layout CAF2, UPC++, Chapel, TiDA, Raja/Kokkos

- **Current languages over-specify data layout and its connection to the iteration space Need abstraction to separate the data layout from the iteration space (Compilation also destroys index/layout information)**
  - **Use metadata to abstract information about the data layout & index space**
  - **Use Lambda Functions to abstract the iteration space for computation**
- **Enables data layout or tiling to change, but solvers remain unchanged !!!**



Example TiDA Performance for SMC Proxy App and Geometric Multigrid (MiniGMG)





- **Integrated into BoxLib (production AMR library) for testing/demonstration**
- **Naïve code with TiDA outperforms native OpenMP code and matches (or even exceeds) performance of manually otpimized tiling**
- **TiDA uses HWLOC() hardware locality mapping library to →**
  - automate optimal placement of data tiles
  - Automate pinning of threadIDs to processors on multicore systems

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Embedded DSLs *(another approach)*

Math:

Operator to compute in place

$$\phi := \Delta^h(\phi^5)$$

Where ϕ is defined on a union of Boxes in a DIM-dimensional rectangular lattice with mesh spacing *h*, and Δ^h is the standard 2*DIM + 1 point discretization of the Laplacian.

Simple Code using DSL for Users to Write:

```
double f(double& x, int y){return pow(x,y);}
Stencil lapStencil;
  for (dir = 0; dir < DIM; dir++){
    Point pt = getUnitv(dir);
    lapStencil += Shift(pt) -2*Shift(pt*0) + Shift(-pt))/(h*h);
  };
void foo::operator(LevelData<RectGridArray<double>& a_phi){
  a_phi.exchange();
  Iterator it(a_phi);
  for (it.begin();it.ok();it++){
   RectGridArray<double>& phiPatch =a_phi[it()];
   phiPatch = f(.,5)@phiPatch;
   phiPatch = lapStencil(phiPatch) on
   a_phi.getBox(dit());
  }
};
```

10 Level Software Managed Cache Code generated from DSL

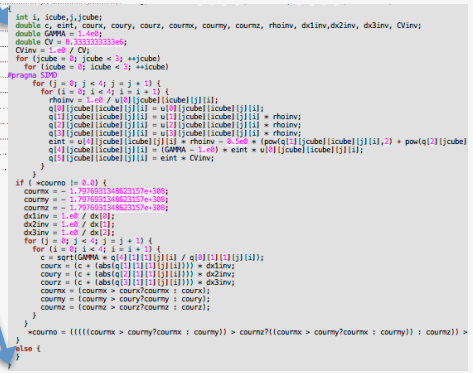
4 Level Hand Written Software Managed Cache Example

Generated code from DSL
Too Complex for Users to Write
(That is code in the background)

4th order stencil computation from CNS Co-Design Proxy-App

Automatically generated code:

- 500+ lines of optimized code
- 10 levels of memory hierarchy
- MPI used at highest level
- Optimized for Many-Core
- 9 levels of software managed cache memory levels
- Optimized for NUMA architecture
- lowest level vectorized
- Developed using SNL SST Micro Exascale Architecture Simulator



BERKELEY LAB

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 CS 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

Data Locality Abstractions
(is it time for standardization?)

- **Many Examples in library and DSL form**
  - **HTA:** Hierarchical Tiled Arrays
  - **TiDA:** Tiling as a Durable Abstraction
  - **RAJA & KOKKOS:** C++ Template Metaprogram Lib *(many other examples!!)*
- **All arrived at similar underlying concepts**
  - Lamba functions to relax loop nest order
  - Abstracts data physical layout from logical layout
- **When many different projects independently arrive at the same or very similar solutions**
  - *Perhaps they have found a reasonably optimal solution*
  - *Its time to talk about standardization (MPI forum)*
- **For Tiling Abstractions, see PADAL**

  (Programming Abstractions for Data Locality)

  http://www.padalworkshop.org/

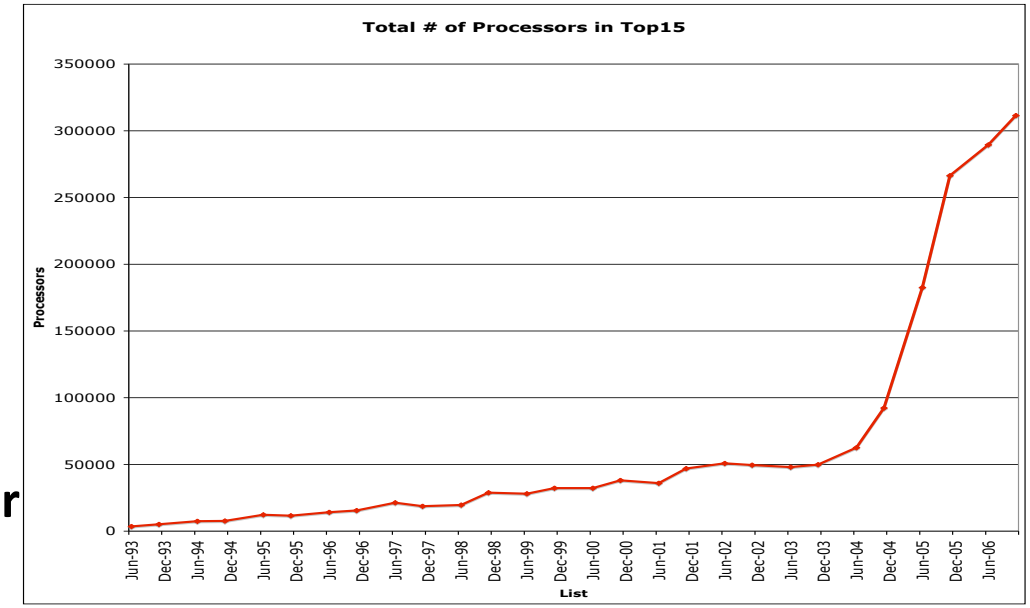
Interconnects

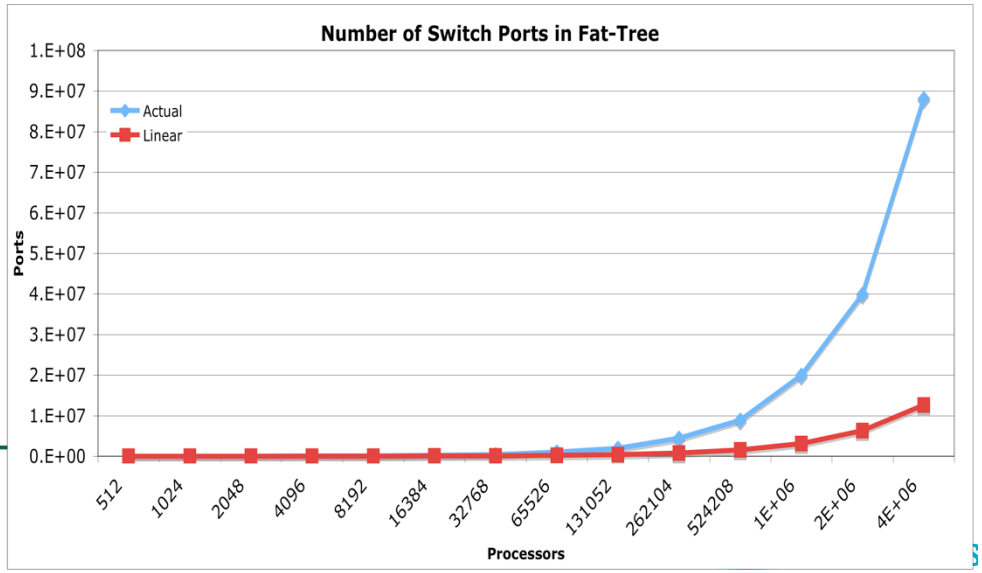
Technology Trends and Effects on Application Performance

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Scalable Interconnects

- Can't afford to continue with Fat-trees or other Fully-Connected Networks (FCNs)
- But will Ultrascale applications perform well on lower degree networks like meshes, hypercubes or torii. Or high-radix routers/tapered dragonfly?
- How can we wire up a custom interconnect topology for each application?



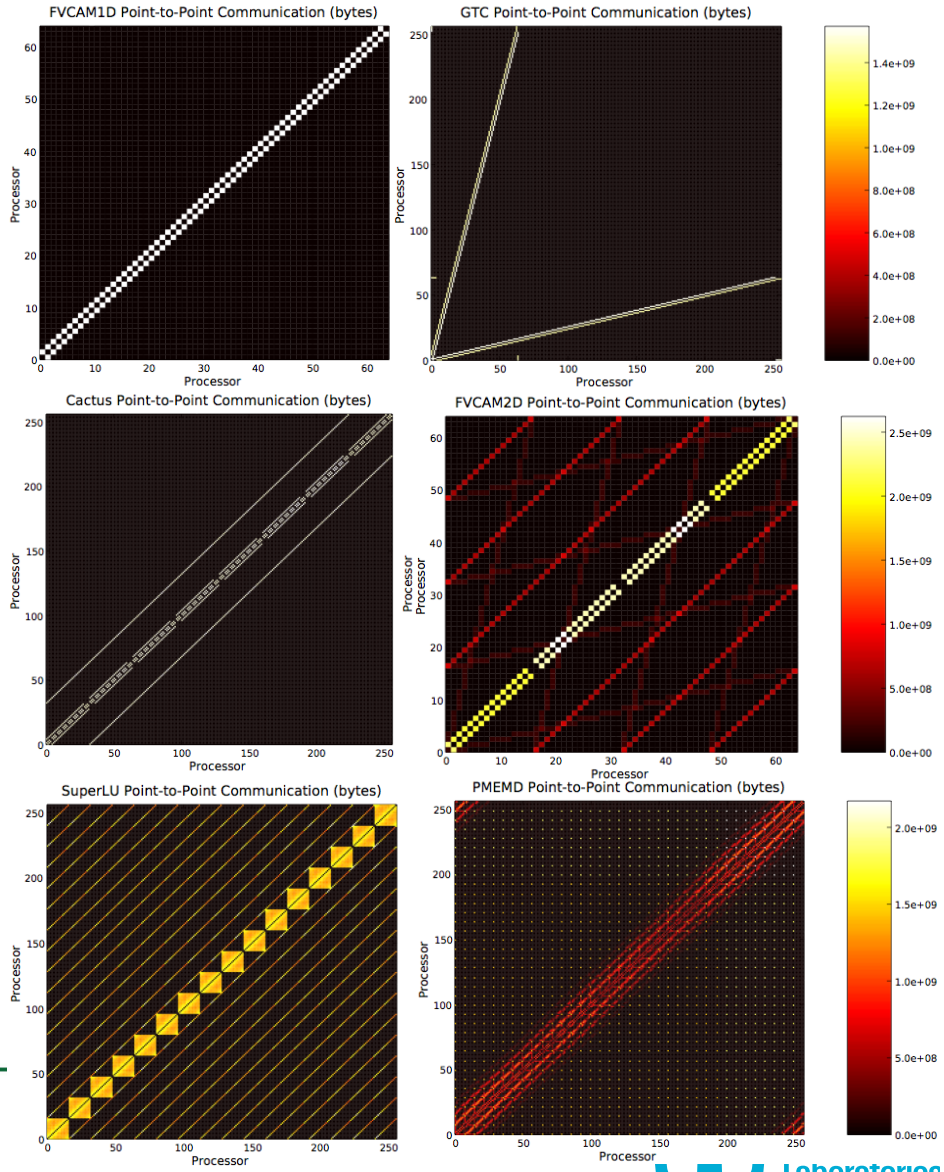


BERKELEY LAB

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Interconnect Design Considerations for Message Passing Applications

- **Application studies provide insight to requirements for Interconnects (both on-chip and off-chip)**
  - On-chip interconnect is 2D planar (crossbar won't scale!)
  - Sparse connectivity for most apps.; crossbar is overkill
  - No single best topology
  - Most point-to-point message exhibit sparse topology + often bandwidth bound
  - Collectives tiny and primarily latency bound
- **Ultimately, need to be aware of the on-chip interconnect topology in addition to the off-chip topology**
  - Adaptive topology interconnects (HFAST)
  - Intelligent task migration?

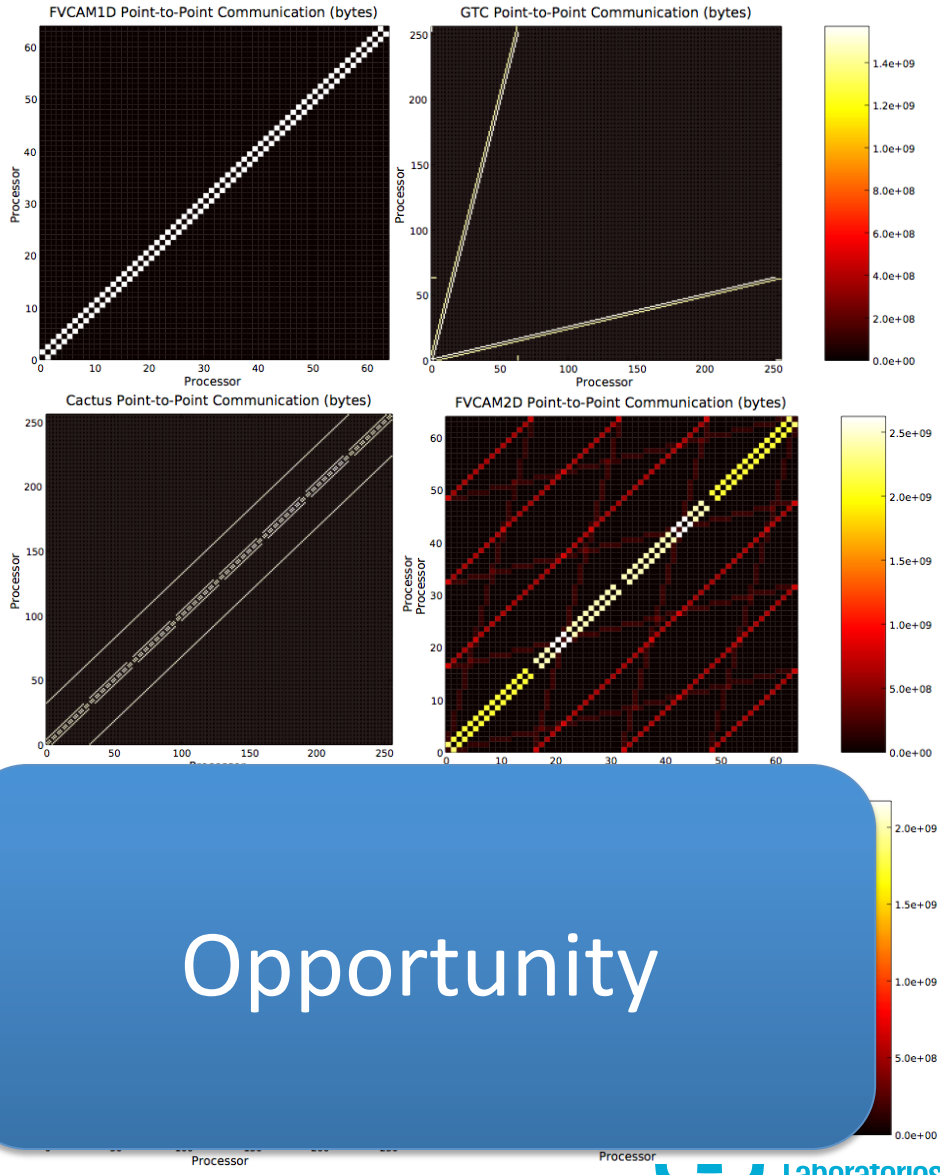


BERKELEY LAB
Laboratories

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Interconnect Design Considerations for Message Passing Applications

- **Application studies provide insight to requirements for Interconnects (both on-chip and off-chip)**
  - On-chip interconnect is 2D planar (crossbar won't scale!)
  - Sparse connectivity for most apps.; crossbar is overkill
  - No single best topology
  - Most point-to-point message exhibit sparse topology + often bandwidth bound
  - Collectives tiny and primarily latency bound
- **Ultimately, need to be aware of the on chip interconnect topology in addition to the off-chip topology**
  - Adaptive topology interconnects (HFAST)
  - Intelligent task migration?



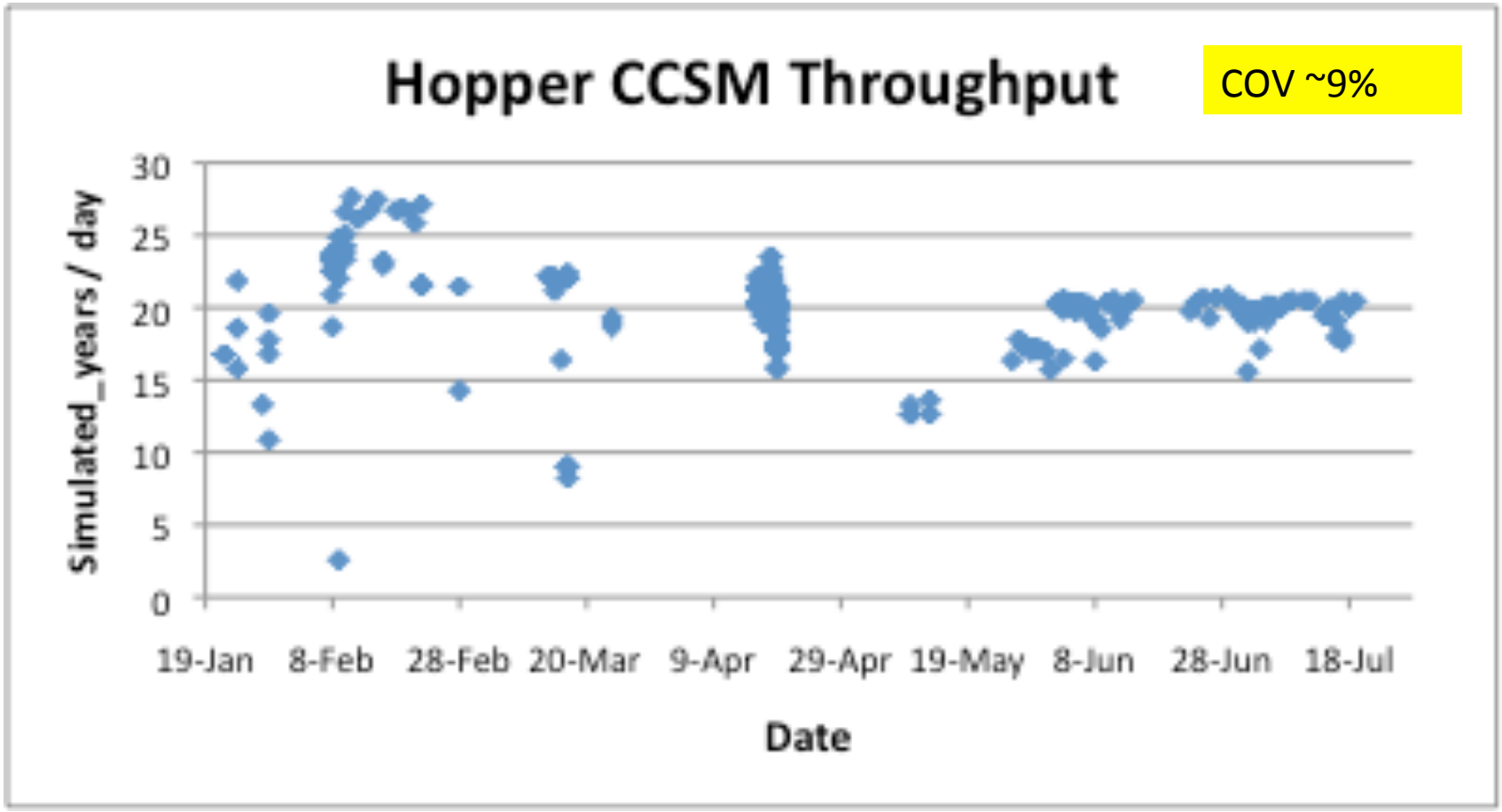
BERKELEY LAB

Laboratories

CCSM Performance Variability

(trials of embedding communication topologies)

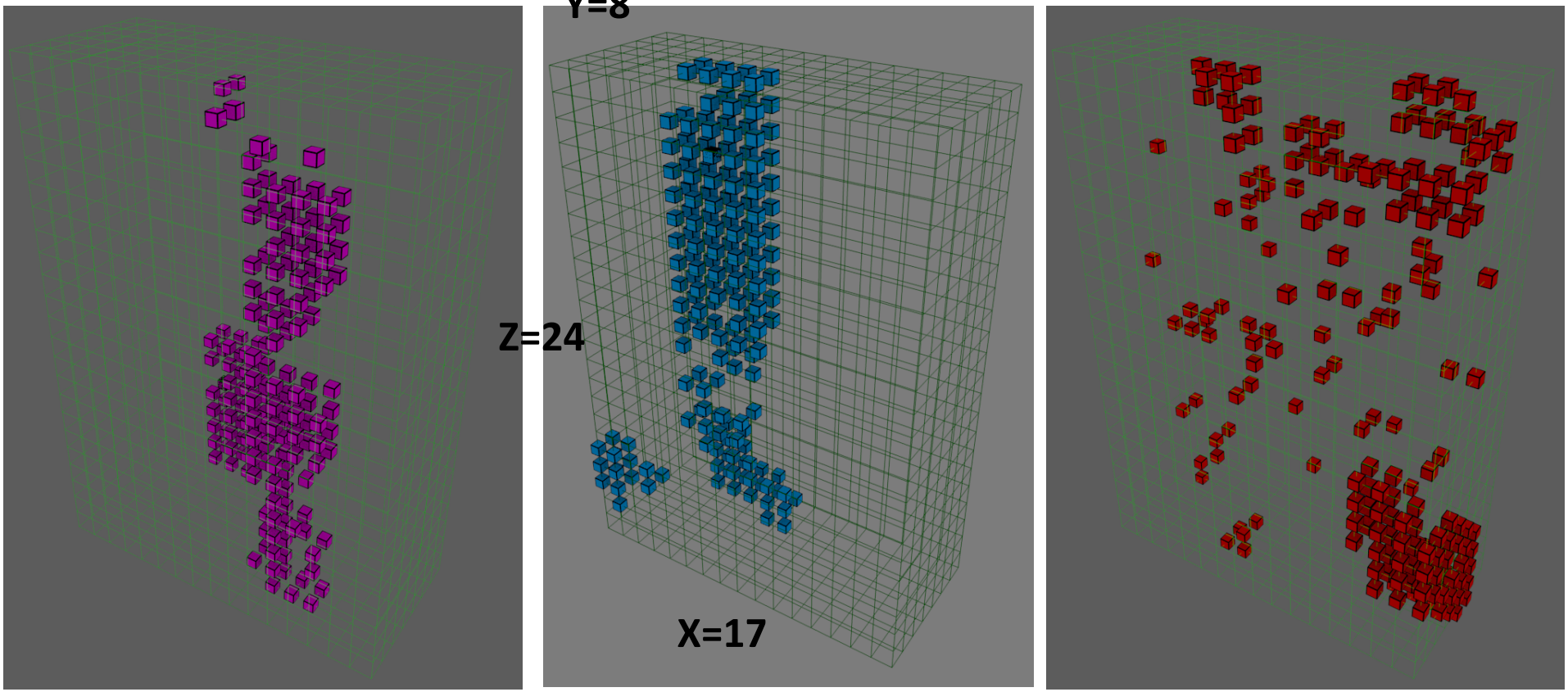
- **Result of 311 runs of the coupled climate model showing model throughput as a function of completion date.**



Data from Harvey Wasserman

Node placement of a fast, average and slow run

from Katie Antypas



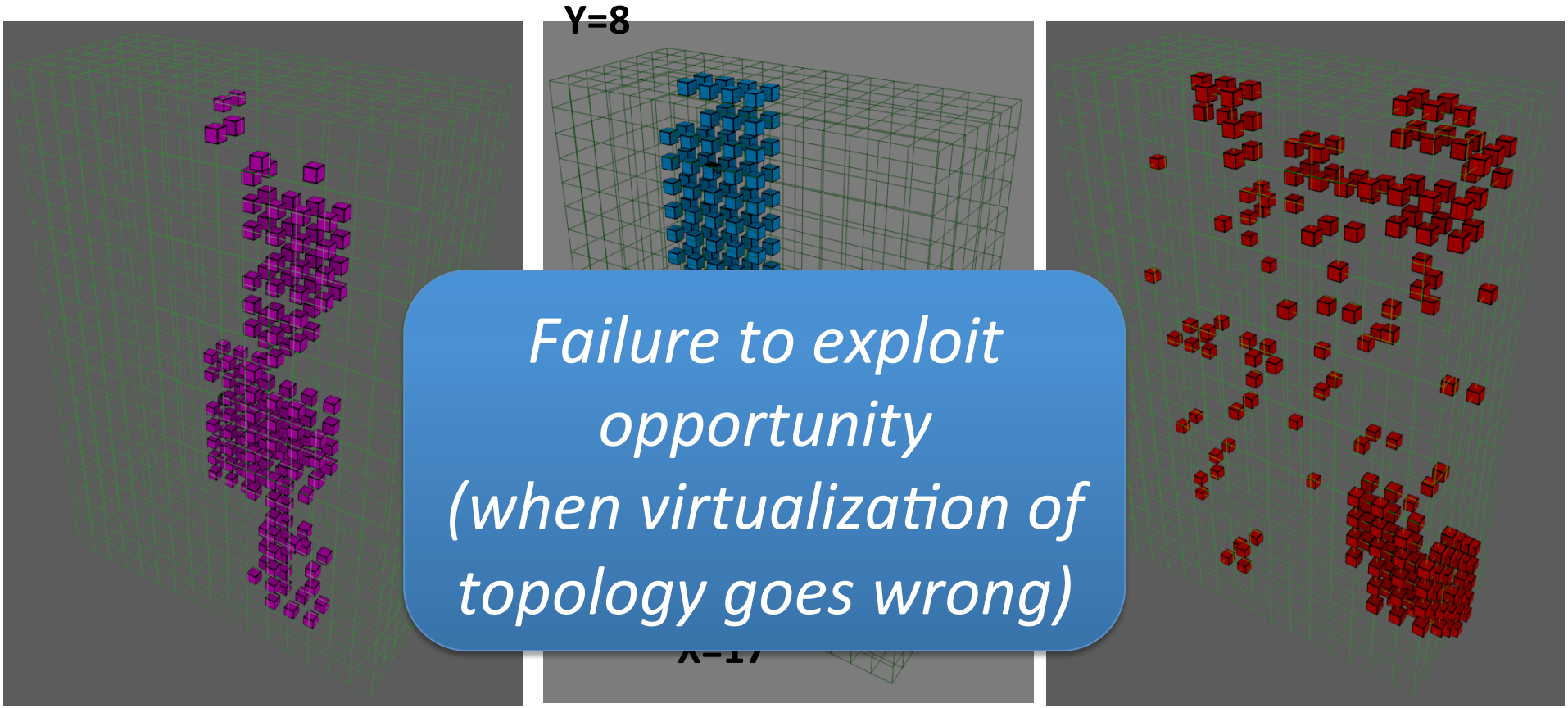
Fast run: 940 seconds

Average run: 1100 seconds

Slow run: 2462 seconds

Node placement of a fast, average and slow run

from Katie Antypas

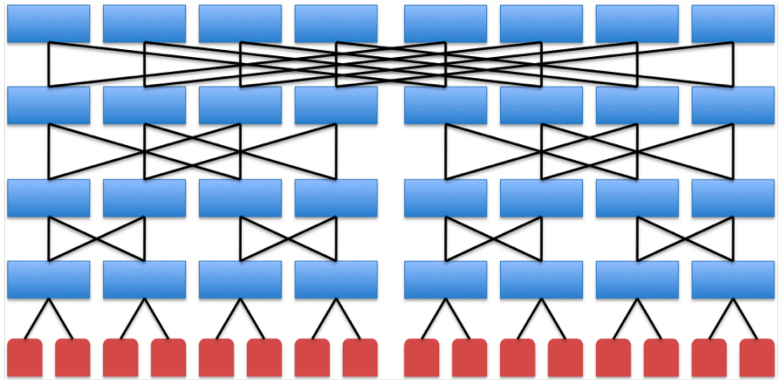


Fast run: 940 seconds

Average run: 1100 seconds

Slow run: 2462 seconds

Topology Optimization
(turning Fat-trees into Fit-trees)



A 2-ary 4-tree with 16 nodes.

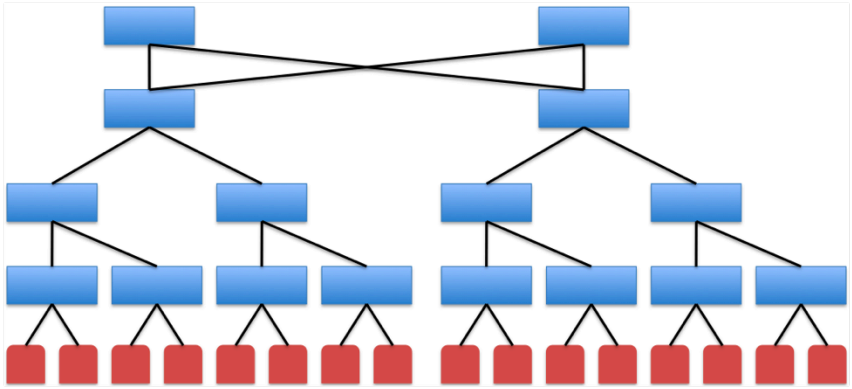
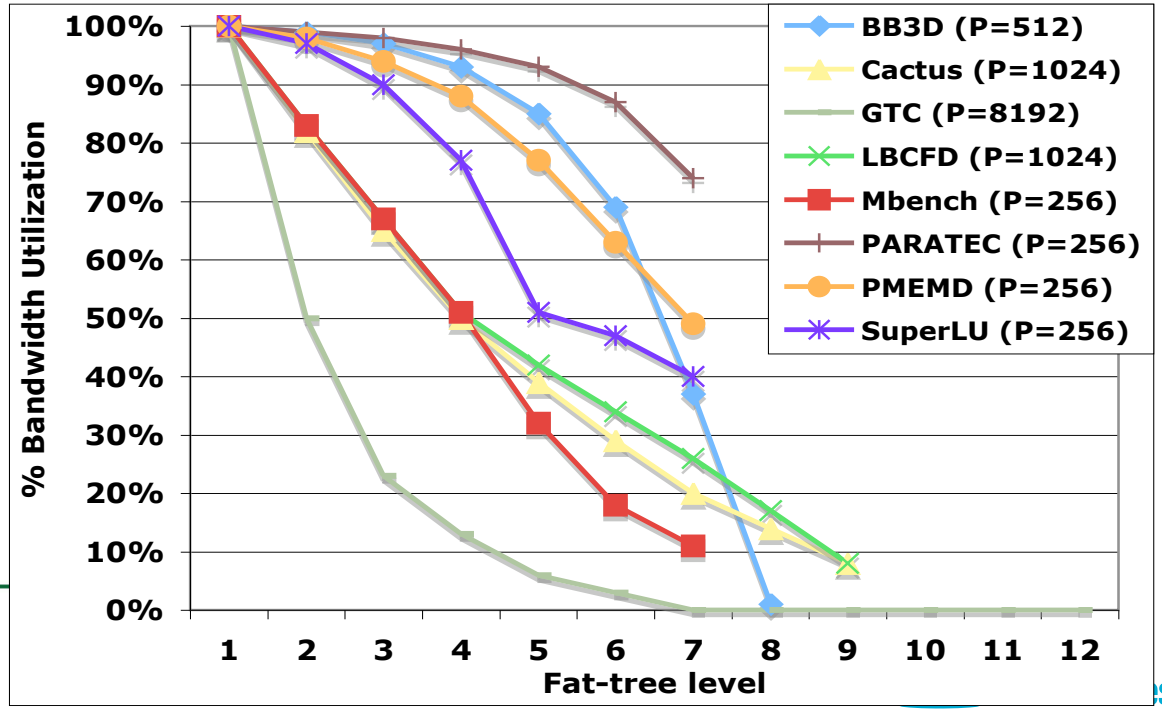


Figure 2: A $(2, 2, 4)$-TL fit-tree with 16 nodes.

- **A Fit-tree uses OCS to prune unused (or infrequently used) connections in a Fat-Tree**
- **Tailor the interconnect bandwidth taper to match application data flows**



Light(er)weight Messaging

Moving towards Global Address Space and MPI3 RMA

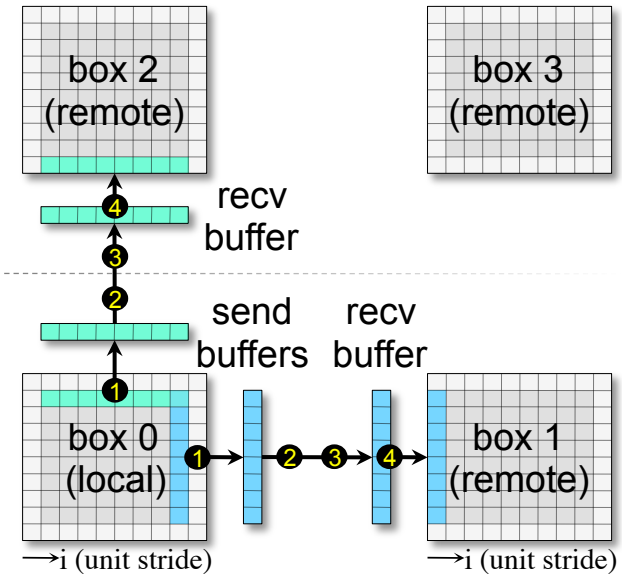
Communications

- **Trend:**
  - **Network/NIC bandwidth scaling, but per-core performance *NOT*.**
  - **Simpler cores take longer to run MPI protocol stack (NOC ingress issues)**
- **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 communication (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**
- **CS teams: already deeply involved in this area**

Yelick/Zheng: LBNL

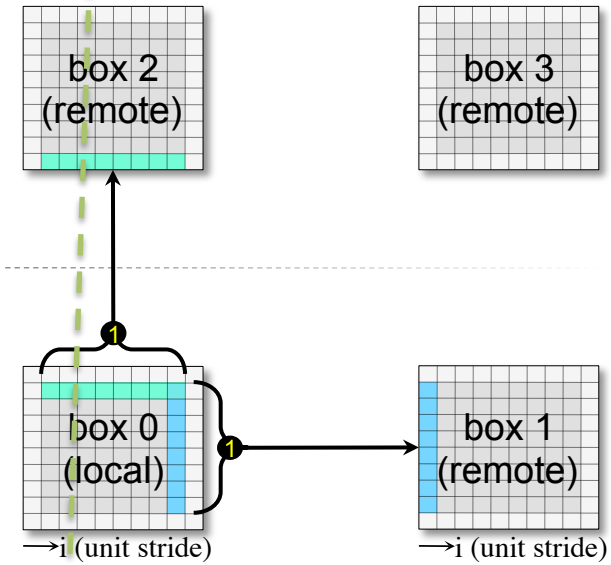
Communication (MPI+OMP vs. 1-sided RMA)

MPI+OpenMP



- **MPI 2-sided** requires *4 steps* to communicate ghost zones with neighbors for best performance*
- **OpenMP** has *no mechanism* to handle multi-level memory hierarchy

PGAS/GAS/MPI3rma



- **One-sided** enables direct boundary exchange in only *1 step* while delegating details to runtime
- **Global Addr**optimizes for memory hierarchy with *hierarchical teams, locales and recursive array tiling*

*An alternative approach using MPI data types with fewer steps is possible but its performance is much slower than the 4-step approach.

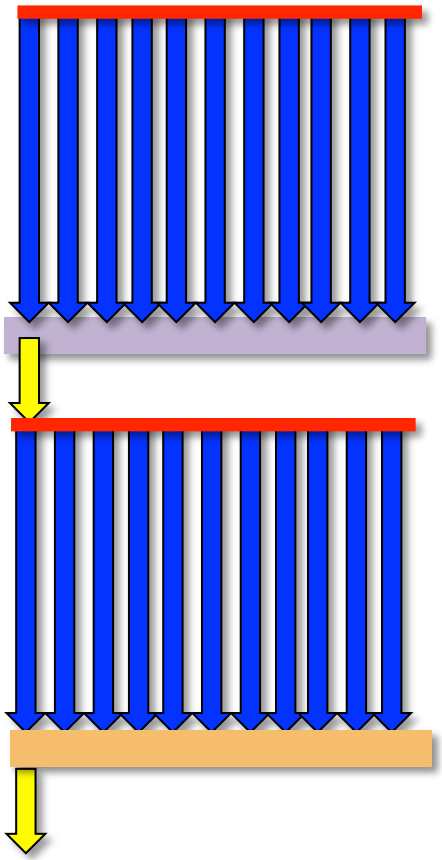
Beyond Bulk Synchronous

Looking Beyond Bulk Sync

- **Motivation**
  - Many sources of machine performance variation (e.g. power management, failure recovery, congestion….)
  - Many sources of algorithmic variation
  - Difficult to coordinate 1million processors to do the same thing simultaneously (bulk synchronous)
- **Value Proposition**
  - Describe task dependencies and have the computer handle the complex scheduling
  - Reduces workload on user to manage the scheduling (we use bulk sync because it requires less thought…)
- **Implementations**
  - OCR: Intel's open community runtime (serves multiple impls.)
  - HPX: Indiana/LSU (Sterling), slated to be part of C++17 standard.
  - Charm++: Pre-dates MPI < and many others >

Performance Heterogeneity

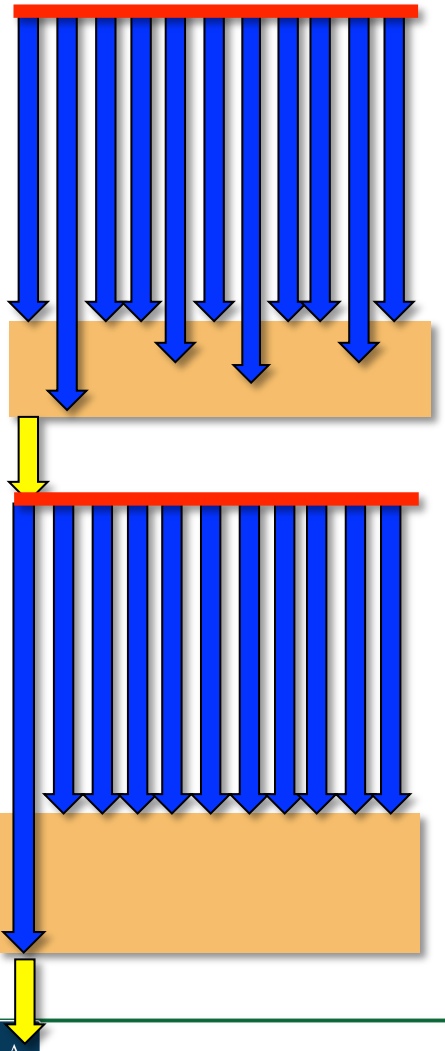
Bulk Synchronous Execution



- **Heterogeneous compute engines (hybrid/GPU computing)**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  - *thermal throttling – no longer guarantee deterministic clock rate*
- **Nonuniformities in process technology creates non-uniform operating characteristics for cores on a CMP**
  - *Near Threshold Voltage (NTV)*
- **Fault resilience introduces inhomogeneity in execution rates**
  - *error correction is not instantaneous*
  - *And this will get WAY worse if we move towards software-based resilience*

Performance Heterogeneity

Bulk Synchronous Execution

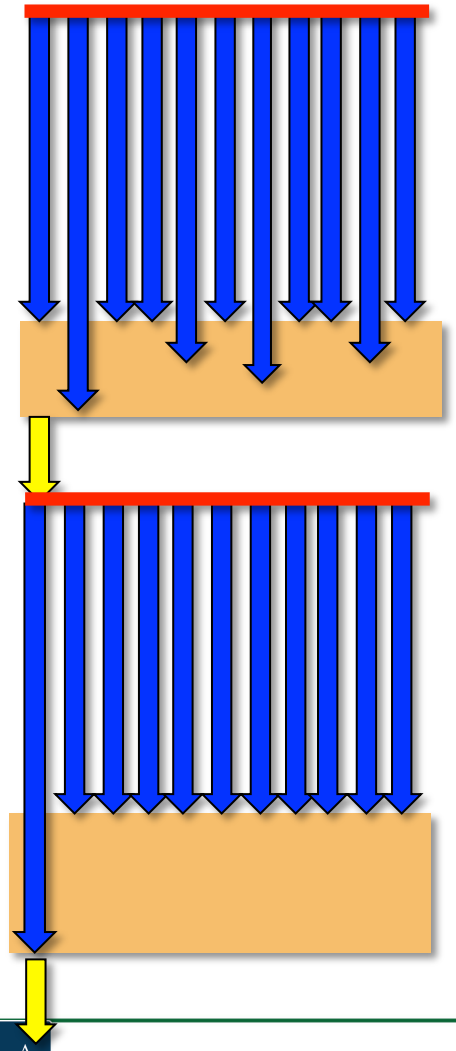


- **Heterogeneous compute engines (hybrid/GPU computing)**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  - *thermal throttling – no longer guarantee deterministic clock rate*
- **Nonuniformities in process technology creates non-uniform operating characteristics for cores on a CMP**
  - *Near Threshold Voltage (NTV)*
- **Fault resilience introduces inhomogeneity in execution rates**
  - *error correction is not instantaneous*
  - *And this will get WAY worse if we move towards software-based resilience*

Near Threshold Voltage (NTV): *Shekhar Borkar (Intel)*

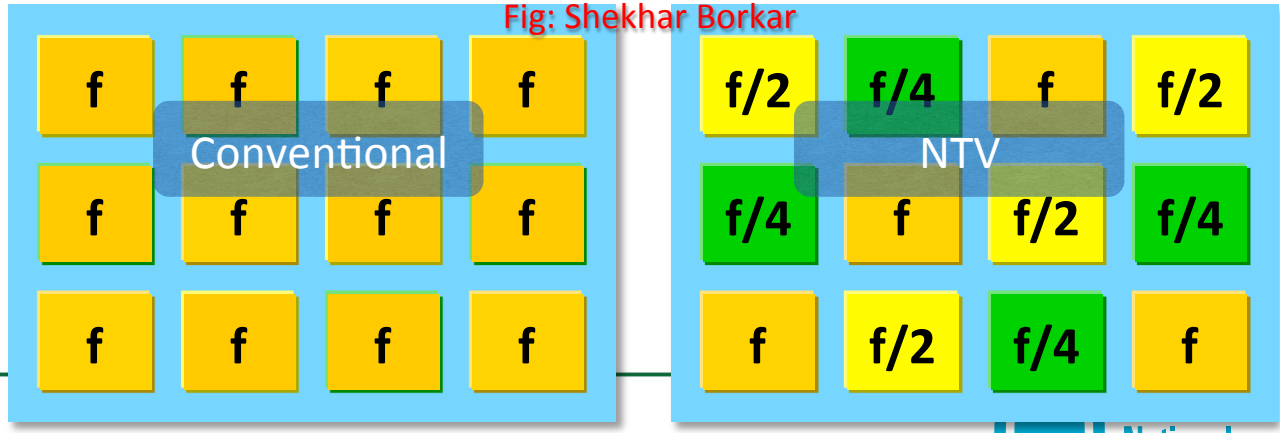
The big opportunities for energy efficiency require codesign!

Bulk Synchronous Execution



- **Heterogeneous compute engines (hybrid/ GPU computing)**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  - *thermal throttling – no longer guarantee deterministic clock rate*
- **Nonuniformities in process technology creates non-uniform operating characteristics for cores on a CMP**
  - ***Near Threshold Voltage (NTV)***
- **Fault resilience introduces inhomogeneity in execution rates**
  - *error correction is not instantaneous*
  - *And this will get WAY worse if we move towards software-based resilience*

Fig: Shekhar Borkar

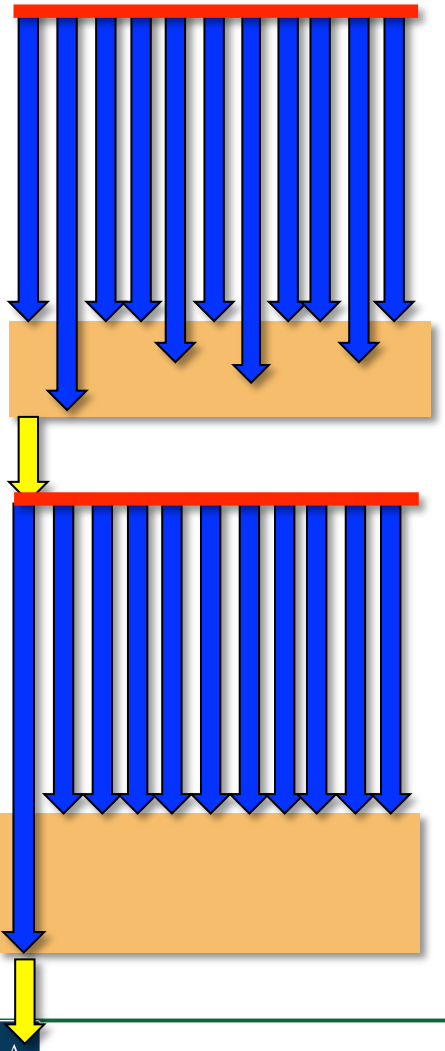


Performance Heterogenity

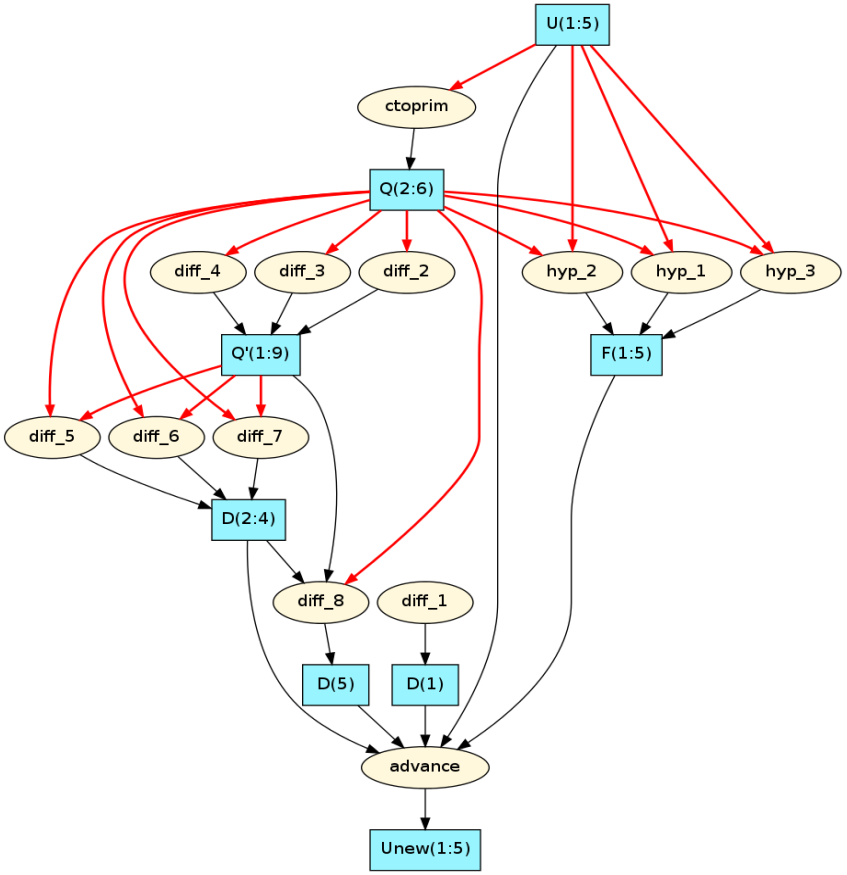
- **Trend**
  - Sources of performance nonuniformity are increasing
  - Unclear if overheads and complexity overwhelm benefits *(adding async to load-balanced code just adds overhead)*
- **Challenge**
  - This is architectural (you can turn it off potentially)
  - But if you DO turn it off, then you 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 new algorithms (this is not a straight port)
  - need this to conduct the experiment… (don't think about this as presupposing the solution)
- **What CS 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

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

Bulk Synchronous Execution

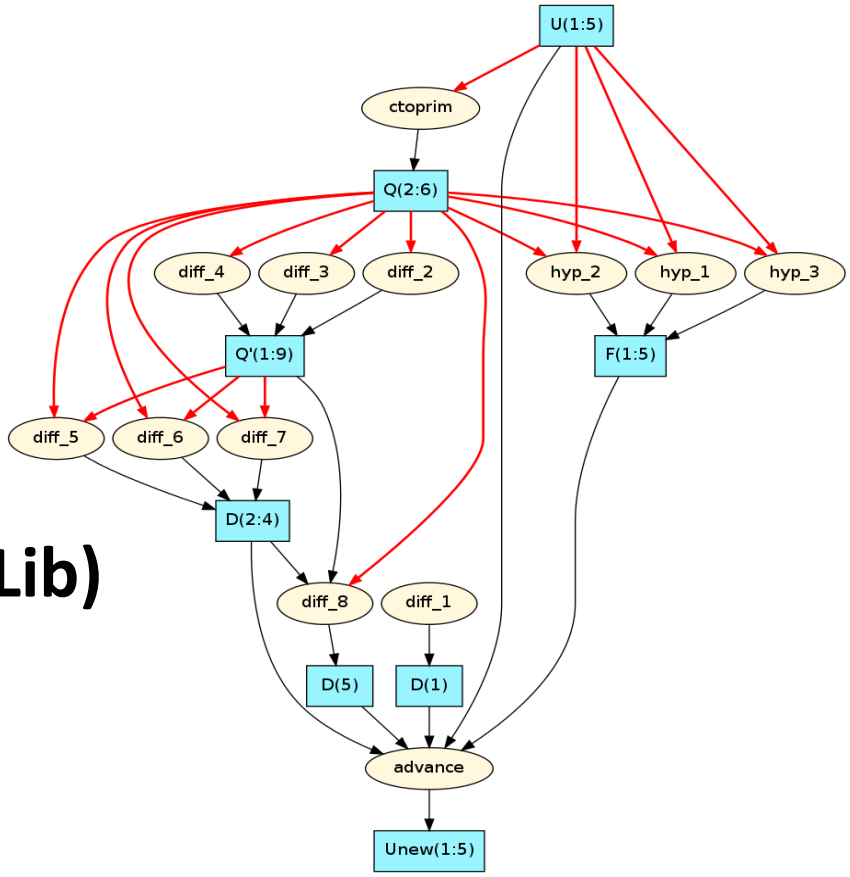


Asynchronous / DAG Execution Model



DAG Scheduling Doesn't Need to be Dynamic to be useful

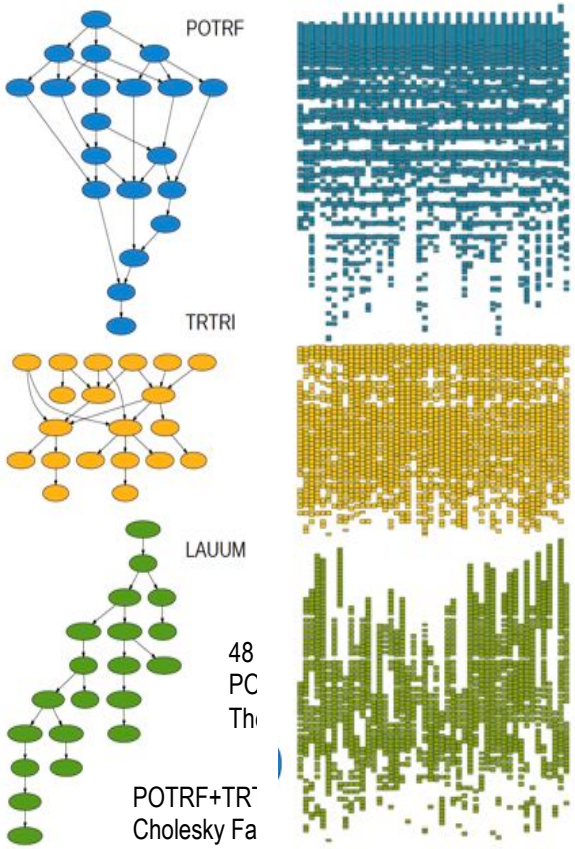
- **Bulk Synchronous**: Most of the existing HPC universe
- **Static Dataflow schedule**: PLASMA/MAGMA
- **Semi-static schedule**: Most AMR libraries (Chombo, BoxLib)
- **Full Dynamic Schedule**: OCR, HPX, Charm++



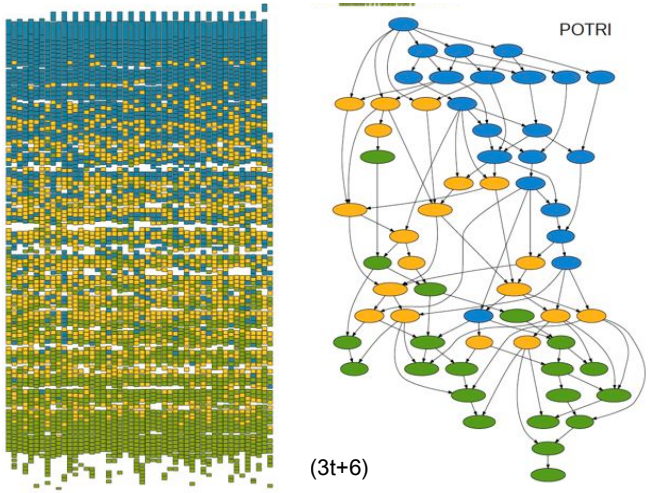
Opportunities for Asynchronous Execution

J.Dongarra

Bulk Synchronous
(current practice)



Asynchronous / DAG Model / static schedule
(production interface is still topic of research)

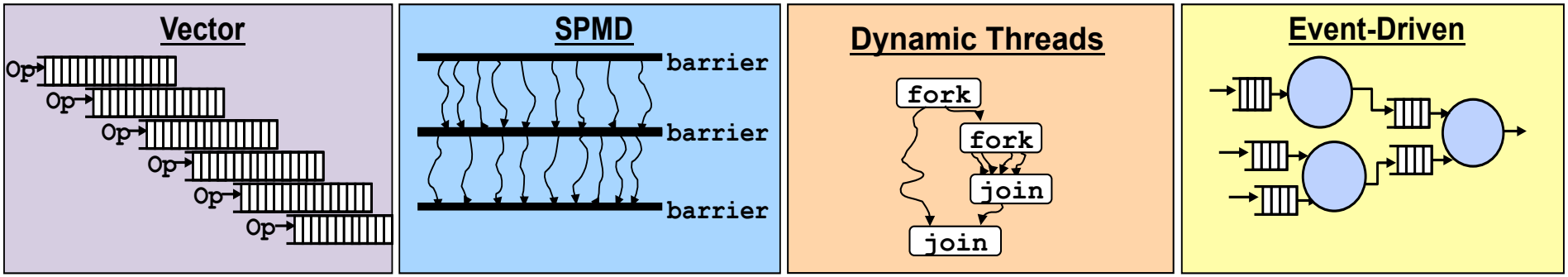


Finding General Purpose programming model to express these constructs requires research.

Clear that OMP4 tasking model is not a productive way to express DAGs (not for domain scientists at least, but could be the underlying model used by a library or pmodel)

Execution Models (*what the heck is it?*)

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?

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 CS teams 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?

Resilience

- **Trend:**
  - Hard to tell if there is a trend
  - Its obviously getting harder, but data is stochastic
  - Issue is that we think about it all wrong
- **Challenge**
  - You don't know where the failures will occur. Must plan for them to occur anywhere
  - Software bugs sometimes indistinguishable from HW failures
  - *Titan sysadmin example*
- **Applications teams**
  - Can you add features to catch errors (right now we don't look)
  - How do you currently respond to errors or isolate them
  - How can I find out the root cause for errors (debuggability).
- **CS teams**
  - What can we do to automate identification of errors and prevent propagation of tainted information (CD has a lot of blanks that need to be filled in)
  - Techniques for isolating errors are tedious (can we automate them?)

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 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 CS 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…)

Conclusions on Heterogeneity

- **Sources of performance heterogeneity increasing**
  - Heterogeneous architectures (accelerator)
  - Thermal throttling
  - Performance heterogeneity due to transient error recovery

- **Current Bulk Synchronous Model not up to task**
  - Current focus is on removing sources of performance variation (jitter), is increasingly impractical
  - Huge costs in power/complexity/performance to extend the life of a purely bulk synchronous model

Embrace performance heterogeneity: Study use of asynchronous computational models (e.g. SWARM, HPX, and other concepts from 1980s)

The Programming Systems Challenge

- **Programming Models are a Reflection of the Underlying Machine Architecture**
  - *Express what is important for performance*
  - *Hide complexity that is not consequential to performance*
- **Programming Models 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 Importance of Codesign

- **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 center modeling and evaluation approach is focused on providing quantitative information about those cost trade-offs to enable rational and thoughtful decision making for both code teams and for our industry partners who will be developing the machines that run those codes.** ***(risk mitigation for expensive decisions)***

Conclusions

- **Emerging hardware constraints are increasingly mismatched with our current programming paradigm**
  - Current emphasis is on preserving FLOPs
  - The real costs now are not FLOPs… it is data movement
  - Requires shift to a data-locality centric programming paradigm and hardware features to support it
- **Technology Changes Fundamentally Disrupt our Programming Environments**
  - The programming environment and associated "abstract machine model" is a reflection of the underlying machine architecture
  - Therefore, design decisions can have deep effect your entire programming paradigm
  - The BIGGEST opportunities in energy efficiency and performance improvements require HW and SW considered together (codesign)
- **Performance Portability Should be Top-Tier Metric for codesign**
  - Know what to **IGNORE**, what to **ABSTRACT**, and what to make more **EXPRESSIVE**

The End

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

Whats wrong with MPI3+OMP4

It will work,

But there are some things we could do better
(a LOT better)

COMPUTER ARCHITECTURE LABORATORY

Old Model (Parallel DO and life was good)

```
DO I=2,N
    B(I) = (A(I) + A(I-1)) / 2.0
ENDDO
```

BERKELEY LAB

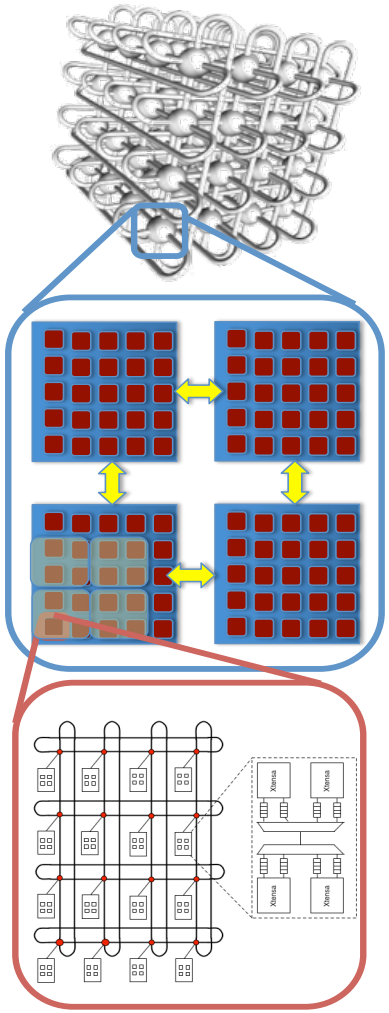
Sandia National Laboratories

Expressing Hierarchical Layout

- **Old Model (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
  - Loop statements operate on data where it is located
  - 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]);
```



COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

OpenACC Example (directives in red)
John Levesque presentation

CRAY THE SUPERCOMPUTER COMPANY

Keep data on the accelerator with acc_data region

```
!$acc data copyin(cix,ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10,ci11,&
!$acc& ci12,ci13,ci14,r,b,uxyz,cell,rho,grad,index_max,index,&
!$acc& ciy,ciz,wet,np,streaming_sbuf1, &
!$acc&    streaming_sbuf1,streaming_sbuf2,streaming_sbuf4,streaming_sbuf5,&
!$acc&    streaming_sbuf7s,streaming_sbuf8s,streaming_sbuf9n,streaming_sbuf10s,&
!$acc&    streaming_sbuf11n,streaming_sbuf12n,streaming_sbuf13s,streaming_sbuf14n,&
!$acc&    streaming_sbuf7e,streaming_sbuf8w,streaming_sbuf9e,streaming_sbuf10e,&
!$acc&    streaming_sbuf11w,streaming_sbuf12e,streaming_sbuf13w,streaming_sbuf14w, &
!$acc&    streaming_rbuf1,streaming_rbuf2,streaming_rbuf4,streaming_rbuf5,&
!$acc&    streaming_rbuf7n,streaming_rbuf8n,streaming_rbuf9s,streaming_rbuf10n,&
!$acc&    streaming_rbuf11s,streaming_rbuf12s,streaming_rbuf13n,streaming_rbuf14s,&
!$acc&    streaming_rbuf7w,streaming_rbuf8e,streaming_rbuf9w,streaming_rbuf10w,&
!$acc&    streaming_rbuf11e,streaming_rbuf12w,streaming_rbuf13e,streaming_rbuf14e, &
!$acc&    send_e,send_w,send_n,send_s,recv_e,recv_w,recv_n,recv_s)
  do ii=1,ntimes
        o o o
     call set_boundary_macro_press2
     call set_boundary_micro_press
     call collisiona
     call collisionb
     call recolor
```

OMP4 example (from OMP4 docs)

```
subroutine vec_mult(p, v1, v2, N)
   real    ::  p(N), v1(N), v2(N)
   integer ::  i
   call init(v1, v2, N)
   !$omp target data map(to: v1, v2) map(from: p)
   !$omp target
   !$omp parallel do
      do i=1,N
                     p(i) = v1(i) * v2(i)
      end do
   !$omp end target
   !$omp end target data
   call output(p, N)
end subroutine
```

And you have to do this for EVERY SINGLE LOOP!

```
#pragma omp target data if(N>THRESHOLD) map(from: p[0:N])
{
    #pragma omp target if (N>THRESHOLD) map(to: v1[:N], v2[:N])
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
    init_again(v1, v2, N);
    #pragma omp target if (N>THRESHOLD) map(to: v1[:N], v2[:N])
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = p[i] + (v1[i] * v2[i]);
}
```

So you think you can do better?

Yes.
Alternatives exist

… that have no users or commercial support …
We should work on that…

BERKELEY LAB

U.S. DEPARTMENT OF ENERGY | Office of Science

Sandia National Laboratories

Major XStack Areas

- **Domain Specific Languages**
  - represent code in language that is familiar to domain scientist
  - Hide differences between CORI and CORAL architectures
  - Automatically generate GPU and CPU code & automatically tune
  - *Examples: DTEC (LLNL), X-Tune (Utah)*

- **Data Locality Centric / Global Address Space Languages and Libraries**
  - Current languages describe parallelization of the computation. *Data must move to follow the computation.*
  - Future systems data movement more expensive than computation
  - Specify location of data. *Computation occurs where data is located.* Reduce cost of moving data (Global Address Space)
  - *Examples: Co-Array Fortran (Rice), UPC/UPC++ (LBL), TiDA (LBL), RAJA (LLNL)*

- **Alternatives to Bulk Sync Models (Async/DAG Execution Models)**
  - Bulk sync models have all processes execute in lock-step. Pervasive model and easiest to support with MPI+X.
  - Future machines have many sources of irregularity (power management) and so do algorithms (adaptive)
  - *Examples: OCR (Intel), HPX (IU/LSU), Charm++ (UIUC), Legion (Stanford/LANL)*

Other Tools

- **Resilience**
  - **GVR**: API to describe data to save to NVRAM and recover upon failure
  - **Containment Domains**: An API to organize your code into containers that can self-check and recover (prevents errors from bleeding to other parts of your code). <Titan Example>

- **Correctness**
  - **Parsimonius**: Compiler analysis to automatically determine how much floating point precision you need (is your algorithm stable)
  - **Corvette**: Compiler analysis tools to determine if your program is correct after you parallelize it.

Domain Specific Languages

- **Motivation:**
  - Computer languages are closer to hardware, and far from the way domain scientists reason about problems.
  - Machine architectures too divergent -- must write different machine-code for different machines (e.g. CUDA vs. OMP).
- **Value Proposition:**
  - Write in a simplified language that looks more like the domain scientists mathematics (familiar syntax).
  - Computer/DSL automatically converts to optimized GPU and CPU code.
- **Examples (Does it Work)**
  - Good existence proofs, but
  - general purpose implementation has been elusive
- **Risks:**
  - Too narrow (solves only one problem)
  - too general (leading to support issues)

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Automate the Generation of Complex Code for Exascale using Domain Specific Languages

From DTEC: Dan Quinlan

Math:

Operator to compute in place

$$\phi := \Delta^h(\phi^5)$$

Where ϕ is defined on a union of Boxes in a DIM-dimensional rectangular lattice with mesh spacing *h*, and Δ^h is the standard 2*DIM + 1 point discretization of the Laplacian.

Simple Code using DSL for Users to Write:

```
double f(double& x, int y){return pow(x,y);}
Stencil lapStencil;
  for (dir = 0; dir < DIM; dir++){
    Point pt = getUnitv(dir);
    lapStencil += Shift(pt) -2*Shift(pt*0) + Shift(-pt))/(h*h);
  };
void foo::operator(LevelData<RectGridArray<double>& a_phi){
  a_phi.exchange();
  Iterator it(a_phi);
  for (it.begin();it.ok();it++){
   RectGridArray<double>& phiPatch =a_phi[it()];
   phiPatch = f(.,5)@phiPatch;
   phiPatch = lapStencil(phiPatch) on
   a_phi.getBox(dit());
  }
};
```

10 Level Software Managed Cache Code generated from DSL

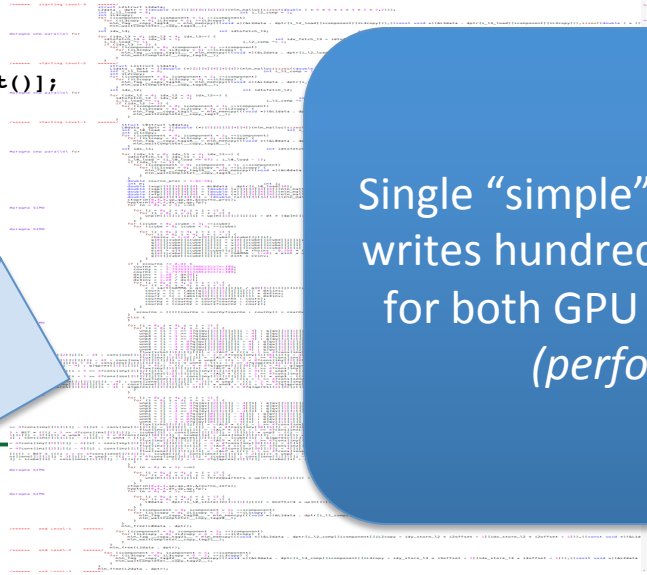
4 Level Hand Written Software Managed Cache Example

Generated code from DSL Too Complex for Users to Write (That is code in the background)

4th order stencil computation from CNS Co-Design Proxy-App

Automatically generated code:

- 500+ lines of optimized code
- 10 levels of memory hierarchy
- Optimized for Many-Core
- 9 levels of software managed cache memory levels
- Optimized for NUMA architecture
- lowest level vectorized



Single “simple” specification of code in DSL writes hundreds of lines of optimized code for both GPU and CPU implementations. *(performance portability)*

BERKELEY LAB

Why the heck should the facilities get involved?

Isn't this a research issue?

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

The Valley of Death