Abstract Machine Models for Exascale Computing

John Shalf

Lawrence Berkeley National Laboratory

XStack PI Meeting, May 28-29, 2014

BERKELEY LAB

U.S. DEPARTMENT OF ENERGY | Office of Science

Sandia National Laboratories

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

CISE CSE 2013 Article

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.

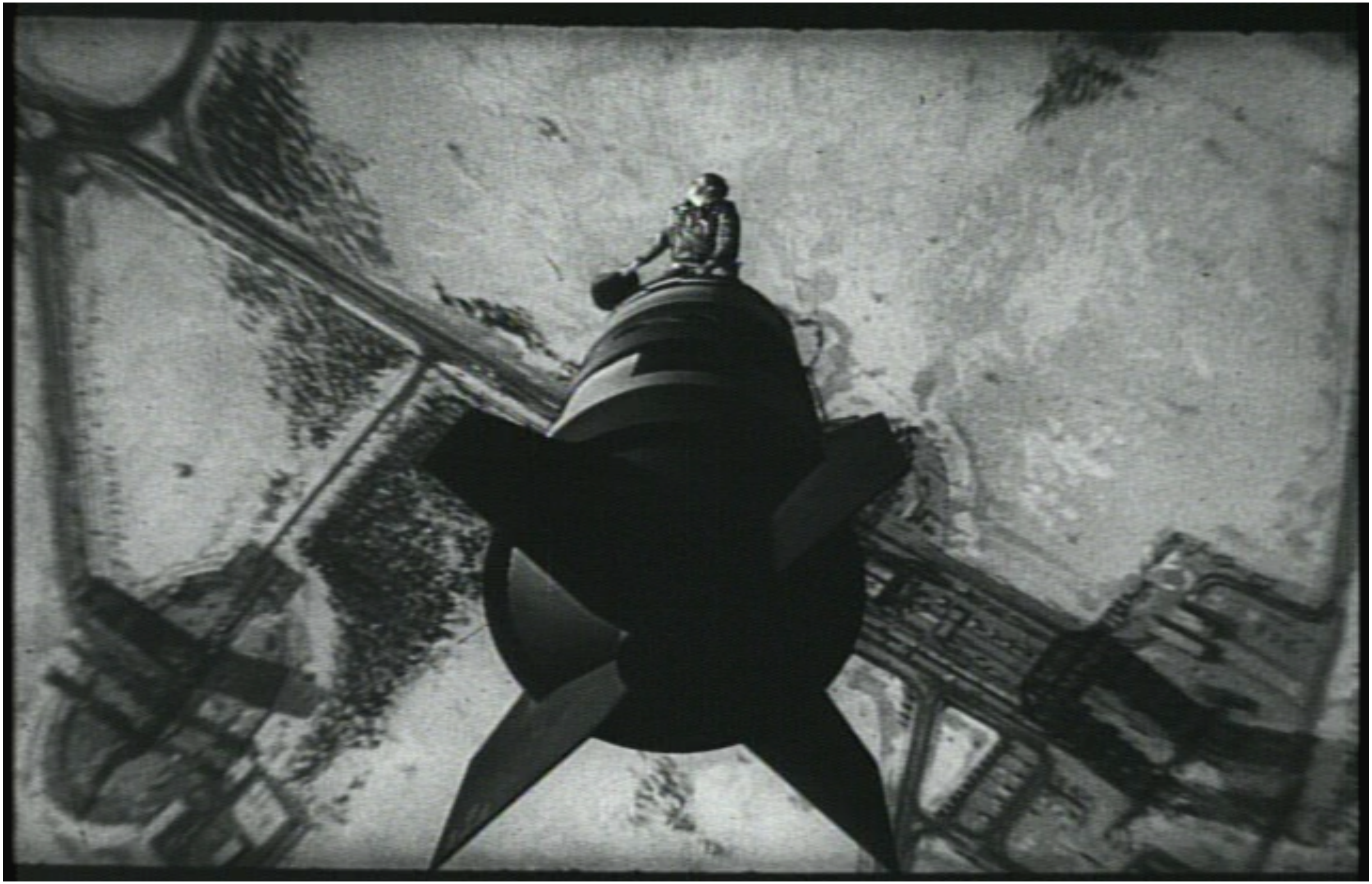
In 2004, however, a confluence of events changed forever the architectural landscape that underpinned MPI. Figure 1 summarizes the effects of these changes in terms of the year-over-year compound annual growth rate (CAGR) of several key system characteristics. This data, taken from an average of the 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.

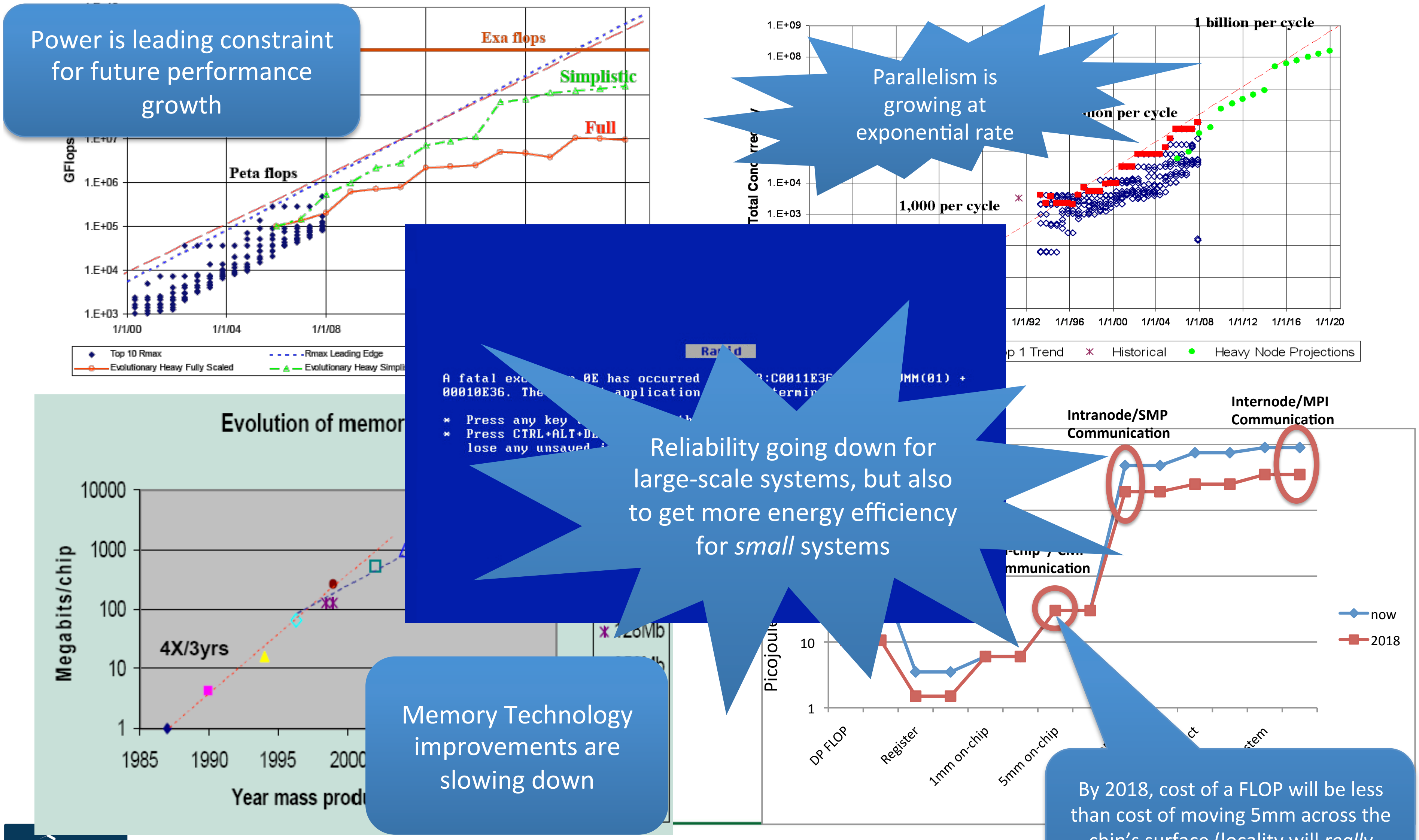
The Perfect Technological Storm

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*

Original Article Title;
How I learned to Stop Worrying and Love Exascale



Technology Challenges for the Next Decade



Power is leading constraint for future performance growth

Parallelism is growing at exponential rate

```
Rapid

A fatal exc      0E has occurred      :C0011E36        MM(01) +
00010E36. The        application         termin

*  Press any key
*  Press CTRL+ALT+DE
   lose any unsaved
```

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)

Whats wrong with Current Programming Environments?

*Designed for Constraints from 30 years ago! **(wrong target!!)***

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

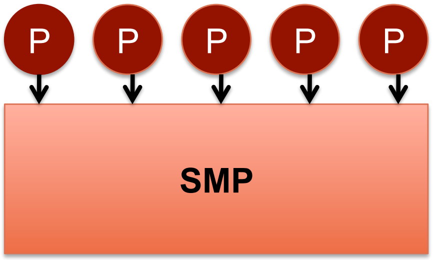
- ***Power** is primary design constraint for future HPC system design*
- **Cost**: *Data movement dominates: optimize to minimize data movement*
- **Concurrency**: *Exponential growth of parallelism within chips*
- **Memory Scaling:** *Compute growing 2x faster than capacity or bandwidth*
- **Locality**: *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

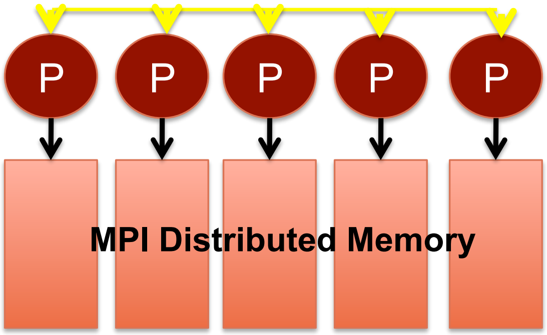
The Programming Model is a Reflection of the Underlying *Abstract Machine Model*

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

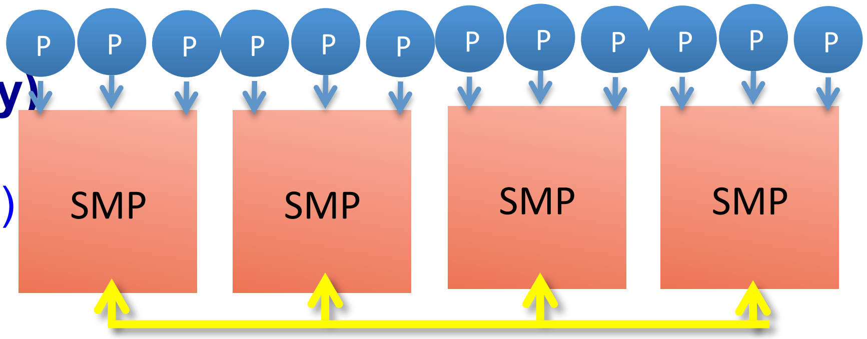
- **Equal cost SMP/PRAM model**
  - No notion of non-local access
  - int [nx][ny][nz];



- **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)

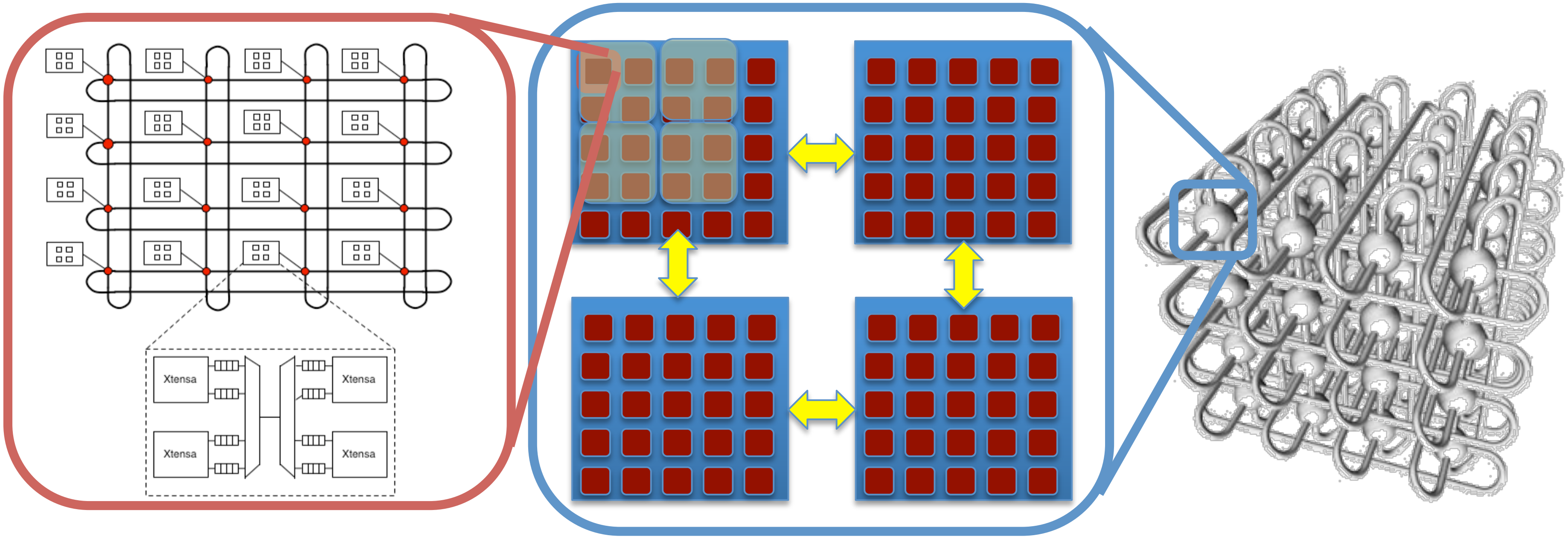


- **Whats Next?**

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

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

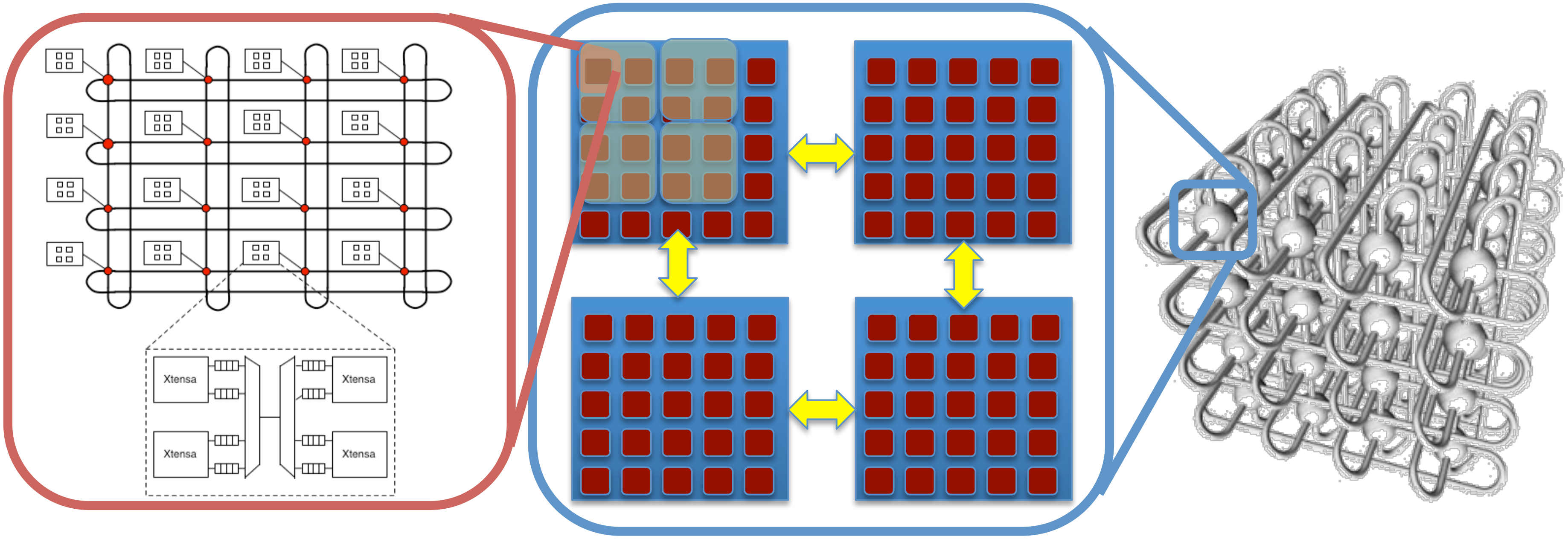
- Bandwidth/Latency/Overhead
- Topology

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 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

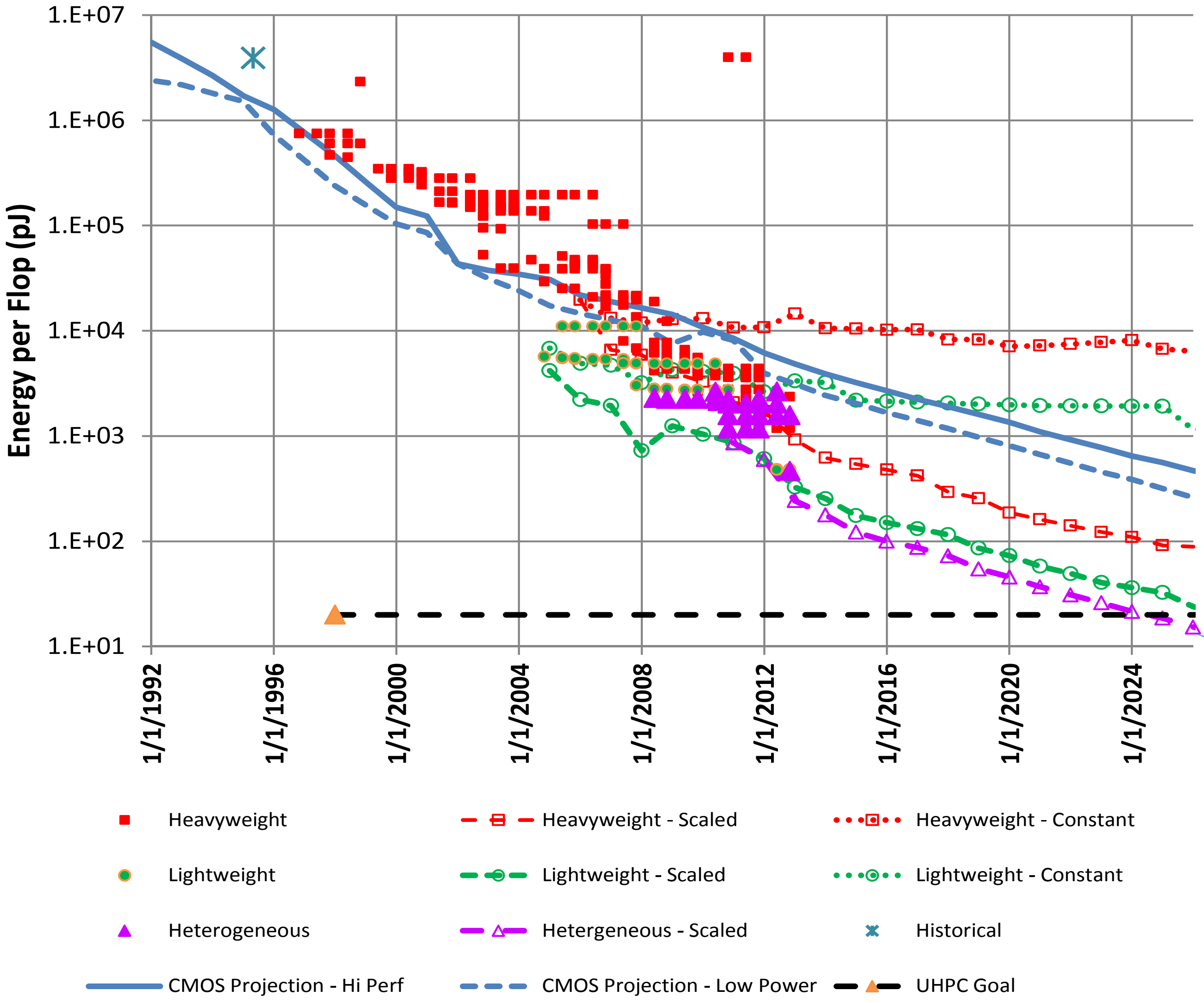
Exascale Strawman Arch

Based on input from DOE Fast Forward and Design Forward Projects

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

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Hybrid Architectures:
Moving from side-show to necessity



Hybrid is the only approach that crosses the exascale finish line

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

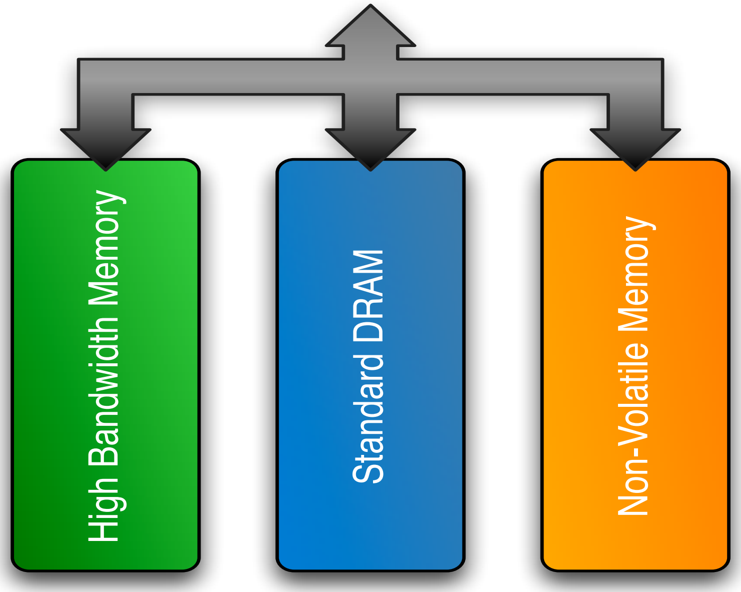
Bandwidth\Capacity	16 GB	32 GB	64 GB	128 GB	256 GB	512 GB	1 TB
4 TB/s							
2 TB/s	Stack/PNM						
1 TB/s			Interposer				
512 GB/s				HMC organic			
256 GB/s					DIMM		
128 GB/s							NVRAM

Old Paradigm for off-chip memory

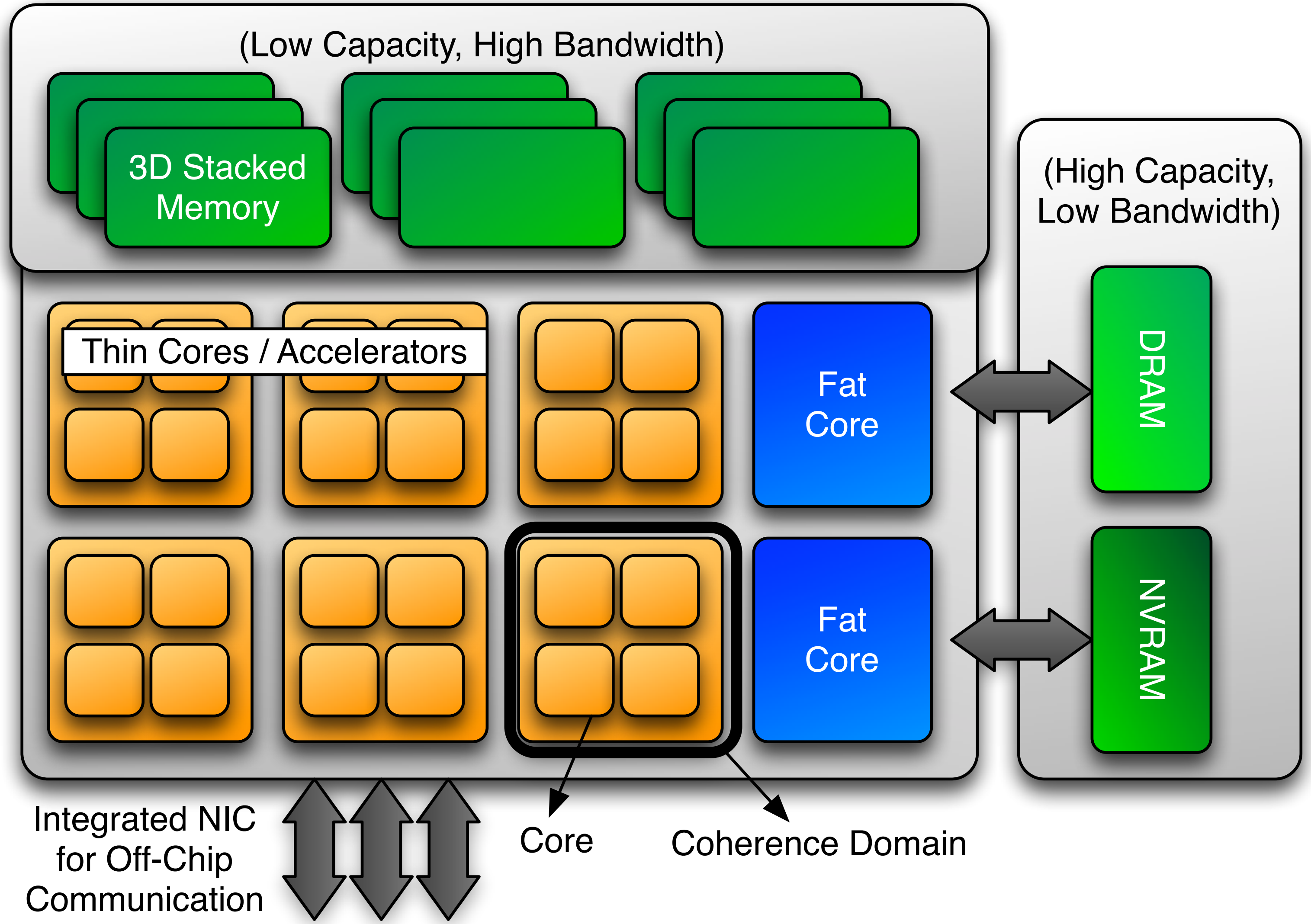
- One kind of memory (JEDEC/DDRx)
- ~1 byte per flop memory capacity
- ~1 byte per flop bandwidth (0.25 typical)

New Paradigm

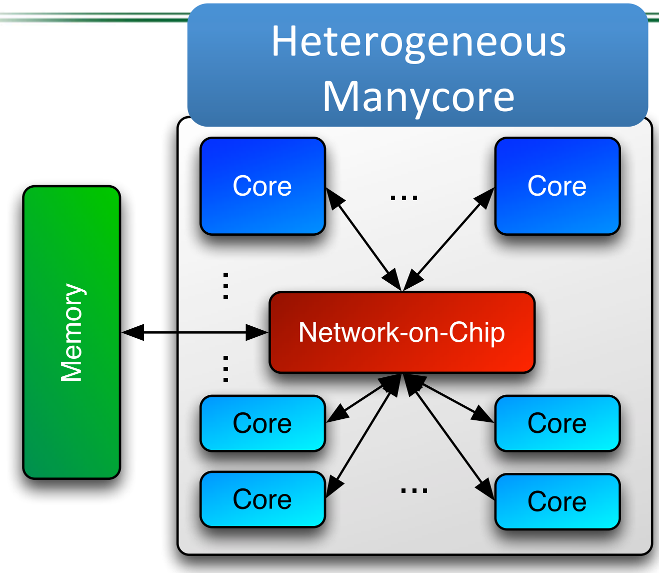
- **DDR4**: ~1 byte per flop capacity w
  < 0.01 bytes/flop BW
- **Stacked Memory**: ~1 byte per flop capacity
  < 0.01 bytes/flop capacity

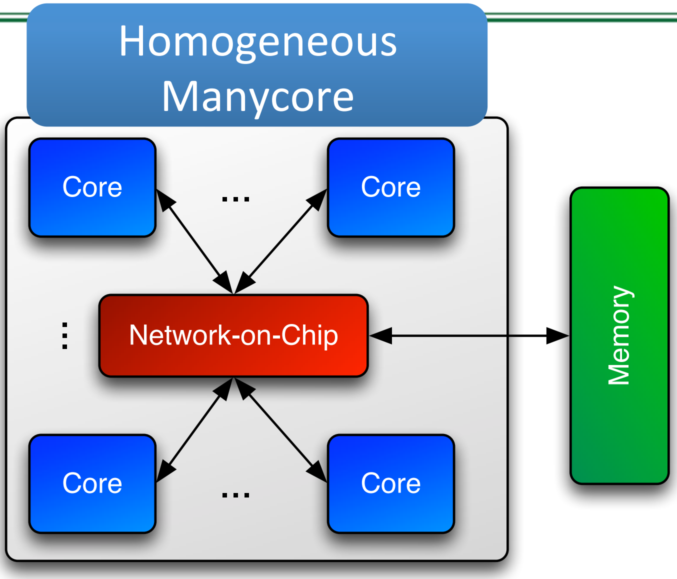


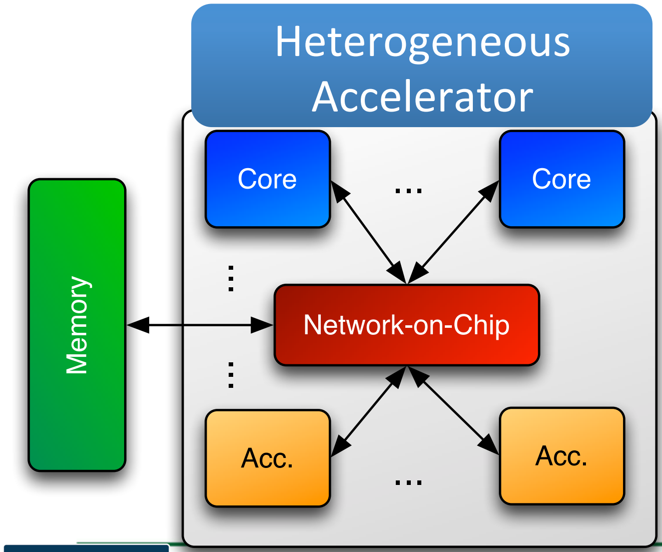
Updated CAL AMM Model

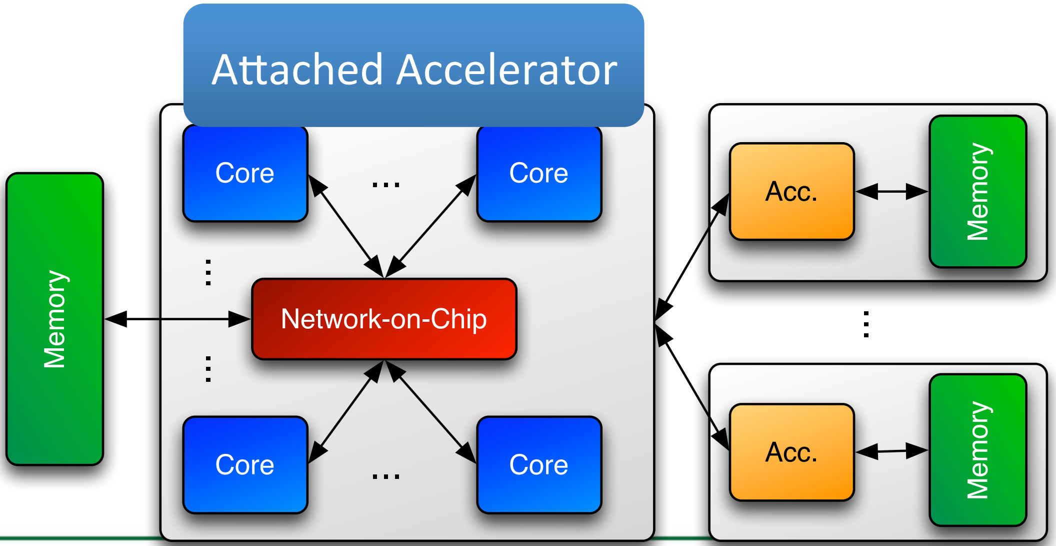

(Low Capacity, High Bandwidth)
3D Stacked Memory
(High Capacity, Low Bandwidth)
Thin Cores / Accelerators
Fat Core
Fat Core
DRAM
NVRAM
Integrated NIC for Off-Chip Communication
Core
Coherence Domain

Families of AMMs

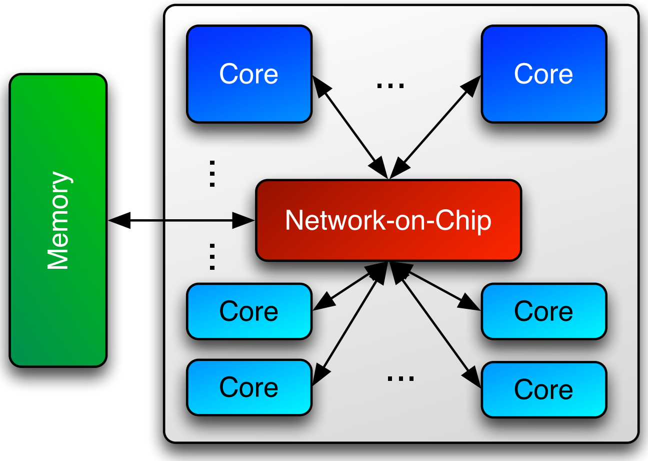

Heterogeneous Manycore

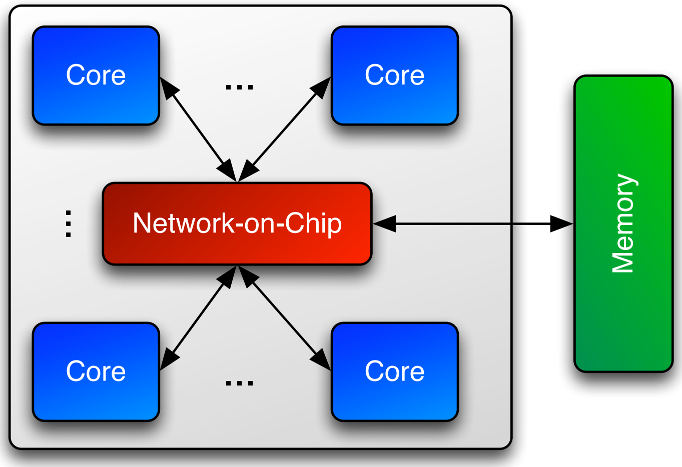

Homogeneous Manycore

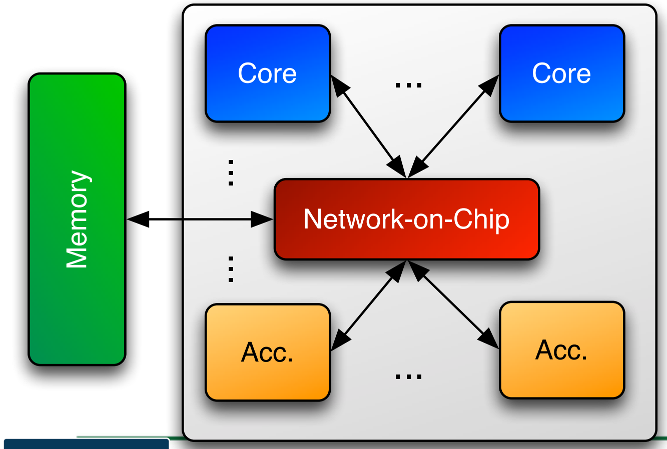

Heterogeneous Accelerator

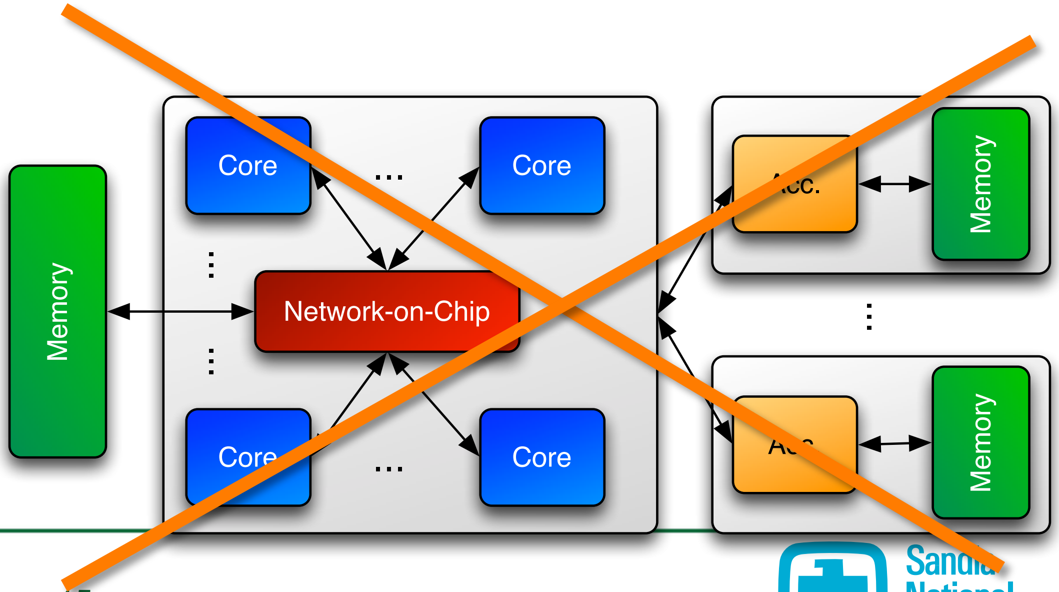

Attached Accelerator

Families of AMMs

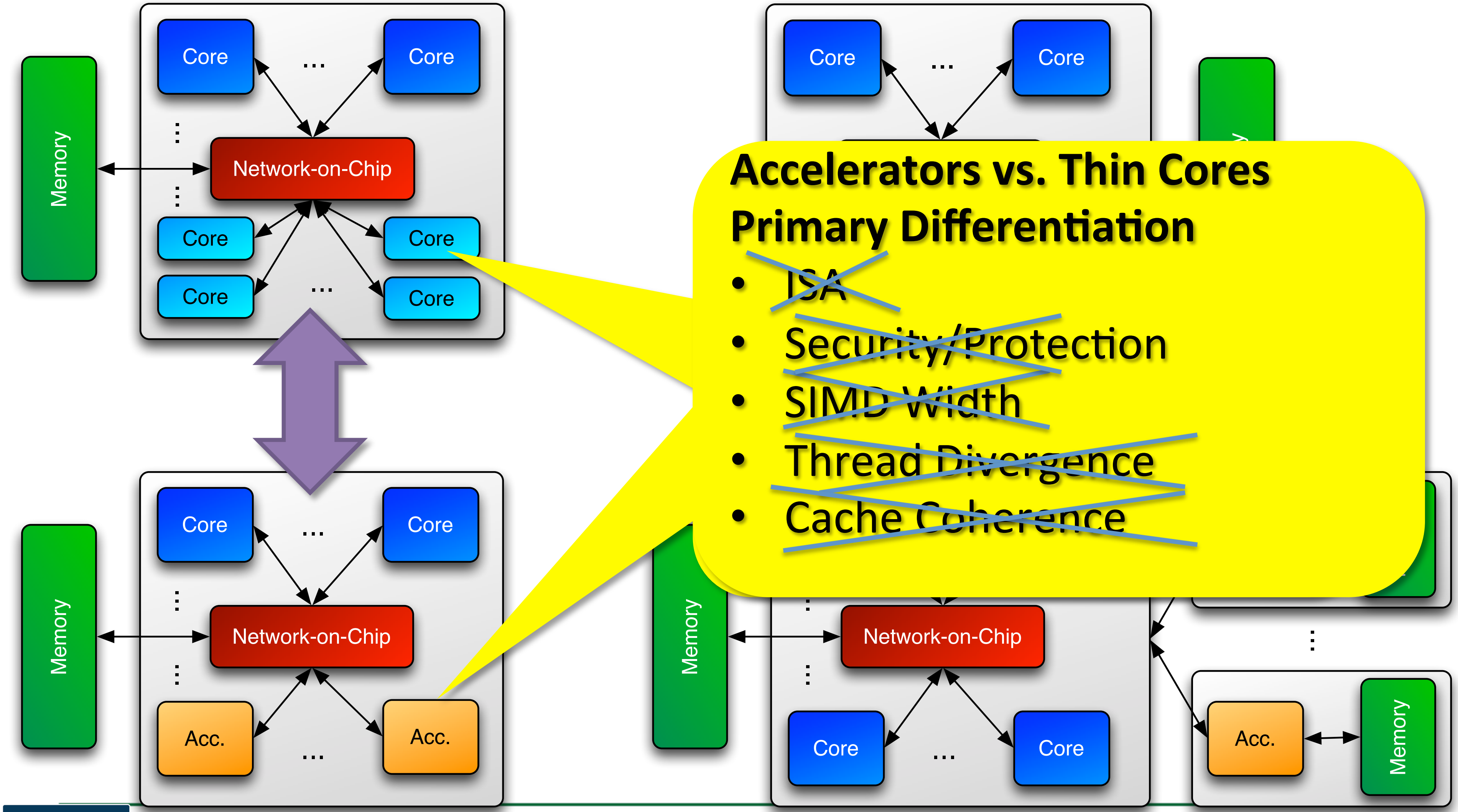








Families of AMMs

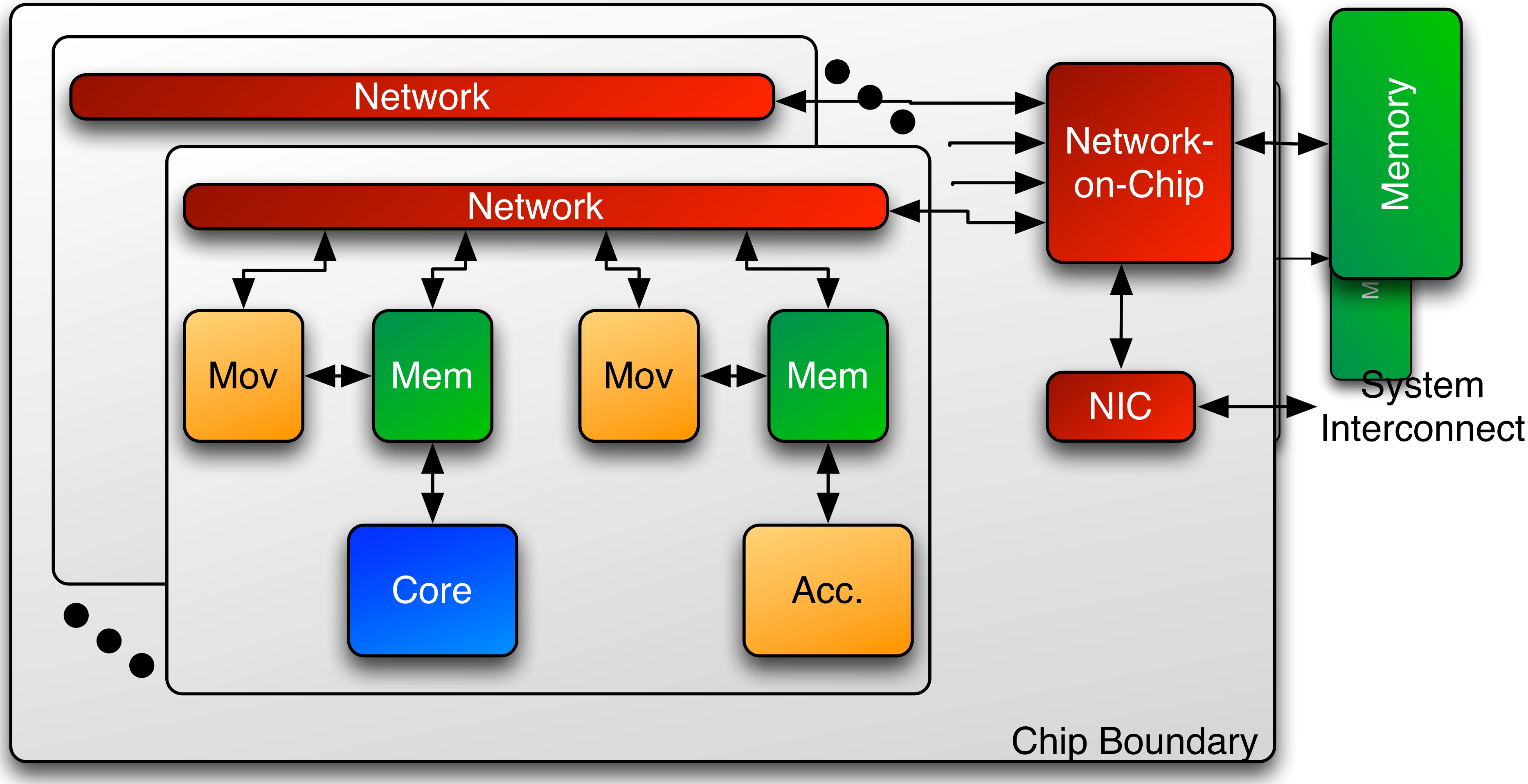


Accelerators vs. Thin Cores Primary Differentiation

- ~~ISA~~
- ~~Security/Protection~~
- ~~SIMD Width~~
- ~~Thread Divergence~~
- ~~Cache Coherence~~

Are these the only possible AMMs?

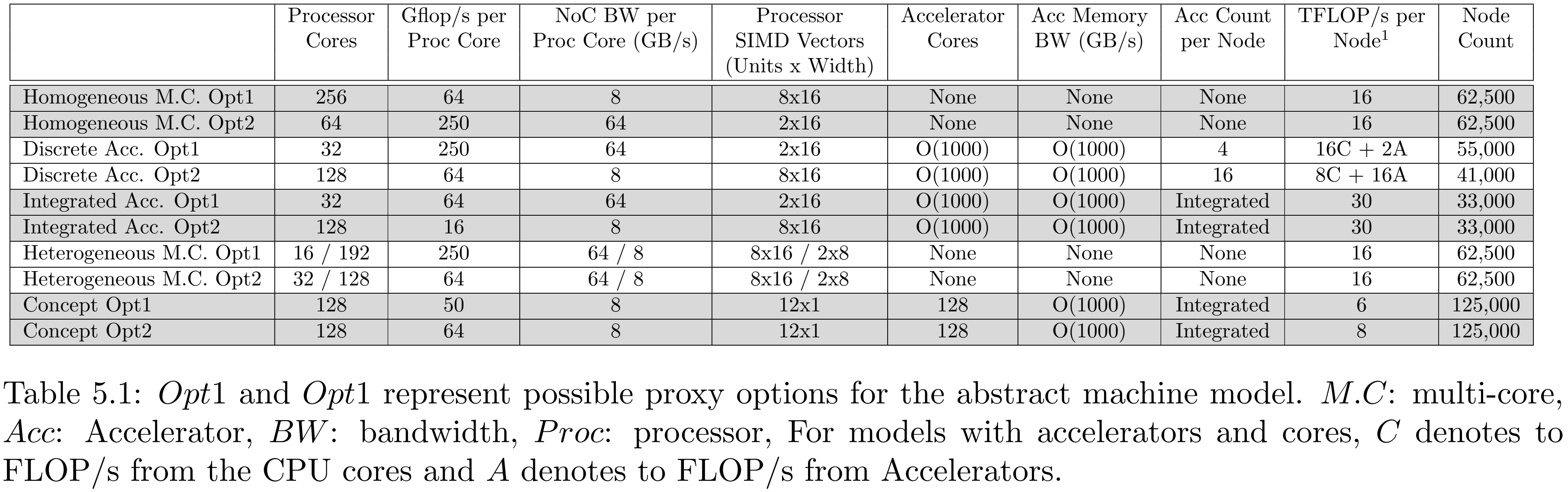
***NO**: this is just a reflection of what is seen developing in industry. Specialization & other architectures possible. See Sandia XGC Project*



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
- **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?

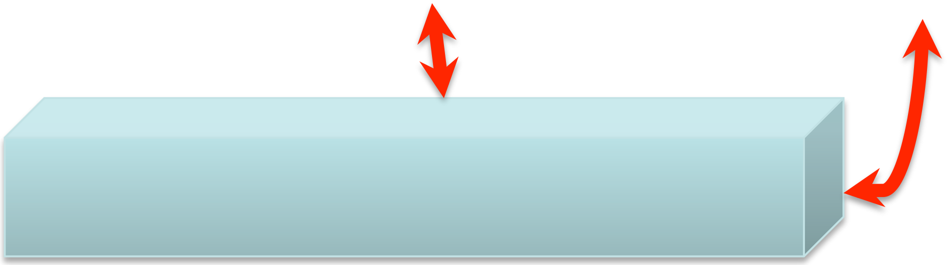
Implications for Future Programming Models

What are the big challenges

for Future Programming Systems

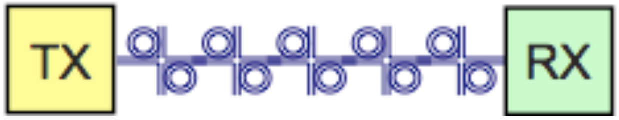
The Problem with Wires:
Energy to move data proportional to distance

- **Cost to move a bit on copper wire:**
  - **Power = Bitrate * Length / cross-section area**

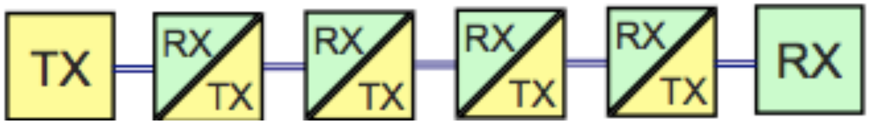


- **Wire data capacity constant as feature size shrinks**
- ***Cost to move bit proportional to distance***
- ***~1TByte/sec max feasible off-chip BW (10GHz/pin)***
- ***Photonics reduces distance-dependence of bandwidth***

Photonics requires no redrive and passive switch little power

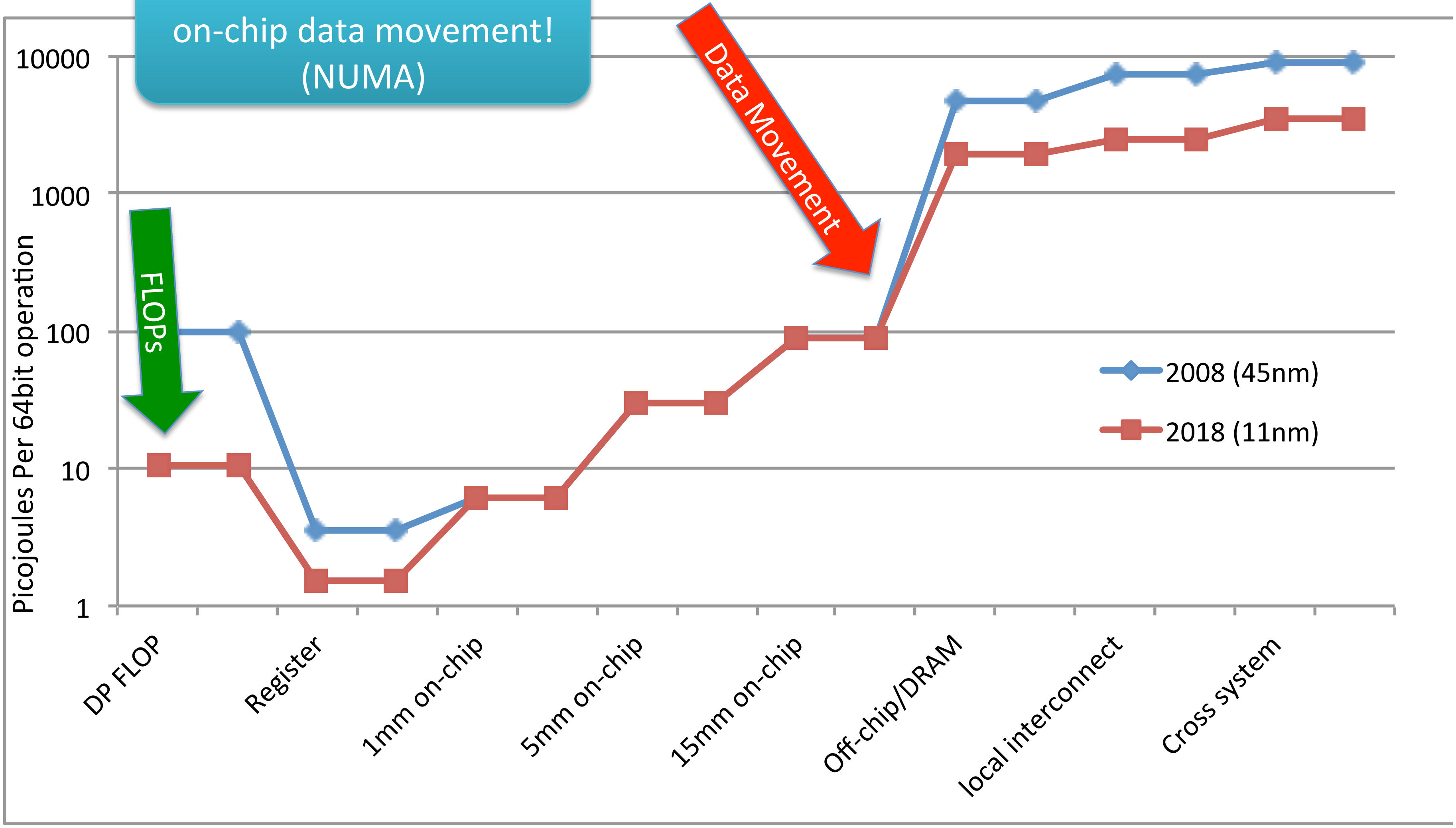


Copper requires to signal amplification even for on-chip connections



Cost of Data Movement Increasing Relative to Ops

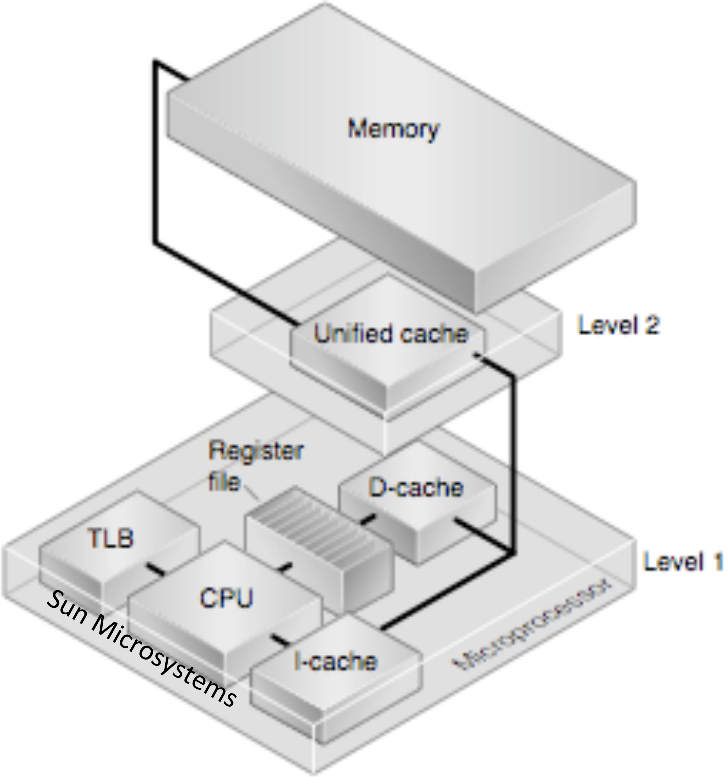
FLOPs will cost less than on-chip data movement! (NUMA)

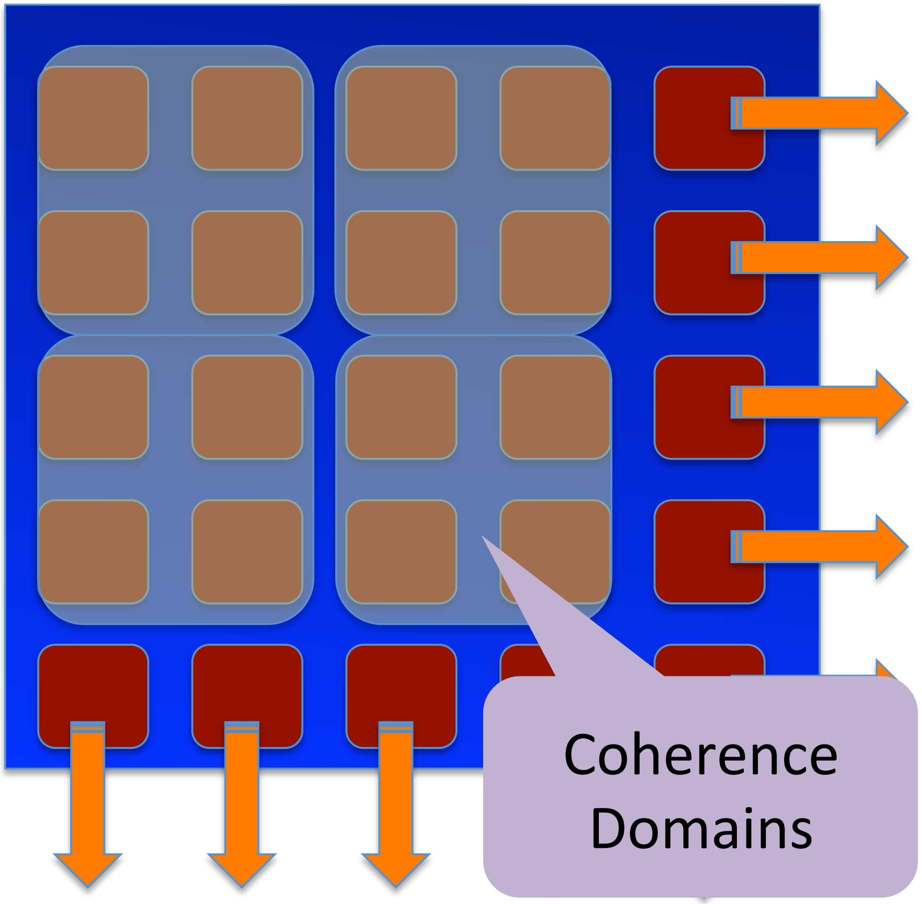


Data Locality Management

Vertical Locality Management
(spatio-temporal optimization)

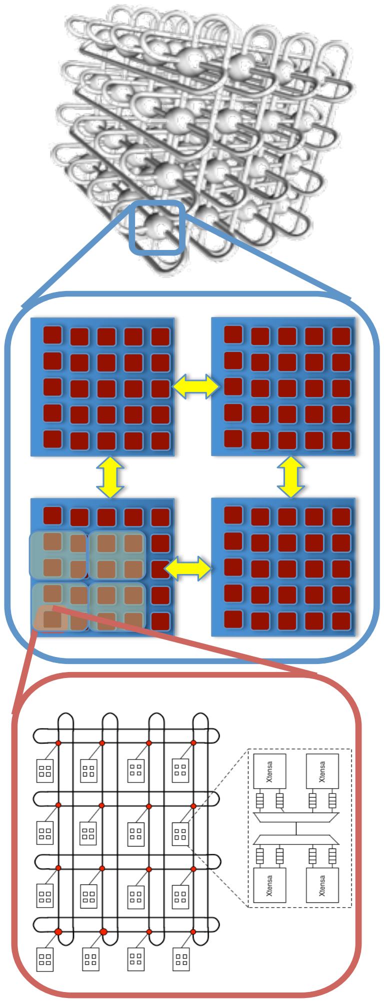
Horizontal Locality Management
(topology optimization)





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?

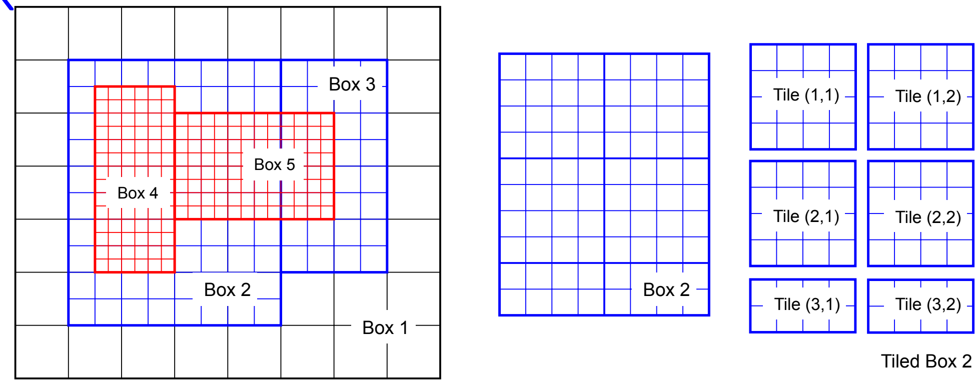


- **New Model (Data-Centric)** ***also in big data***
  - Describe how data is laid out in memory
  - Change applies to ALL Loop statements operate data where it is located (in-situ)
  - Similar to MapReduce, but need more sophisticated descriptions of data layout for scientific codes

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

Tiling Formulation: *abstracts data locality, topology, cache coherence, and parallelism*

- **Expose massive degrees of parallelism through domain decomposition**
  - Represent an atomic unit of work
  - Task scheduler works on tiles
- **Core concept for data locality**
  - **Vertical data movement**
    - *Hierarchical partitioning*
  - **Horizontal data movement**
    - *Co-locate tiles sharing the same data by respecting tile topology*
- **Multi-level parallelism**
  - Coarse-grain parallelism: across tiles
  - Fine-grain parallelism: vectorization, instruction ordering within tile
- ***Centralize and parameterize tiling information at the data structures***
  - Direct approach for memory affinity management for data locality
  - Expose massive degrees of parallelism through domain decomposition
  - *Overcomes challenges of relaxed coherency & coherence domains!!!*



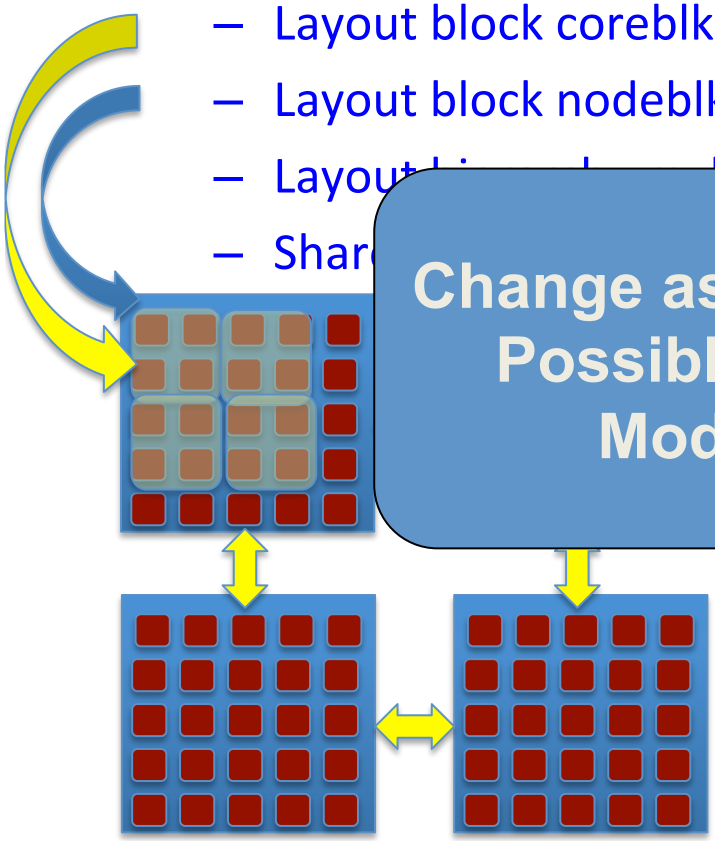
COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATIO

Data-Centric Programming Model

(current compute-centric models are mismatched with emerging hardware)

- **Building up a hierarchical layout**
  - Layout block coreblk {blockx,blocky};
  - Layout block nodeblk {nnx,nny,nnz};
  - Layout
  - Shar

SIAM PP2008



Change as Few Lines of Code as Possible for Each Machine Model or Generation

allel loop

```
do_local(j=0;j<ny;j++;a){
   do_local(k=0;k<nz;k++;a){
         a[i][j][k]=C*a[i+1]…>
```

- ***And if layout changes, this loop remains the same***

Satisfies the request of the application developers
(Change code in one place… affects apply globally to app.)

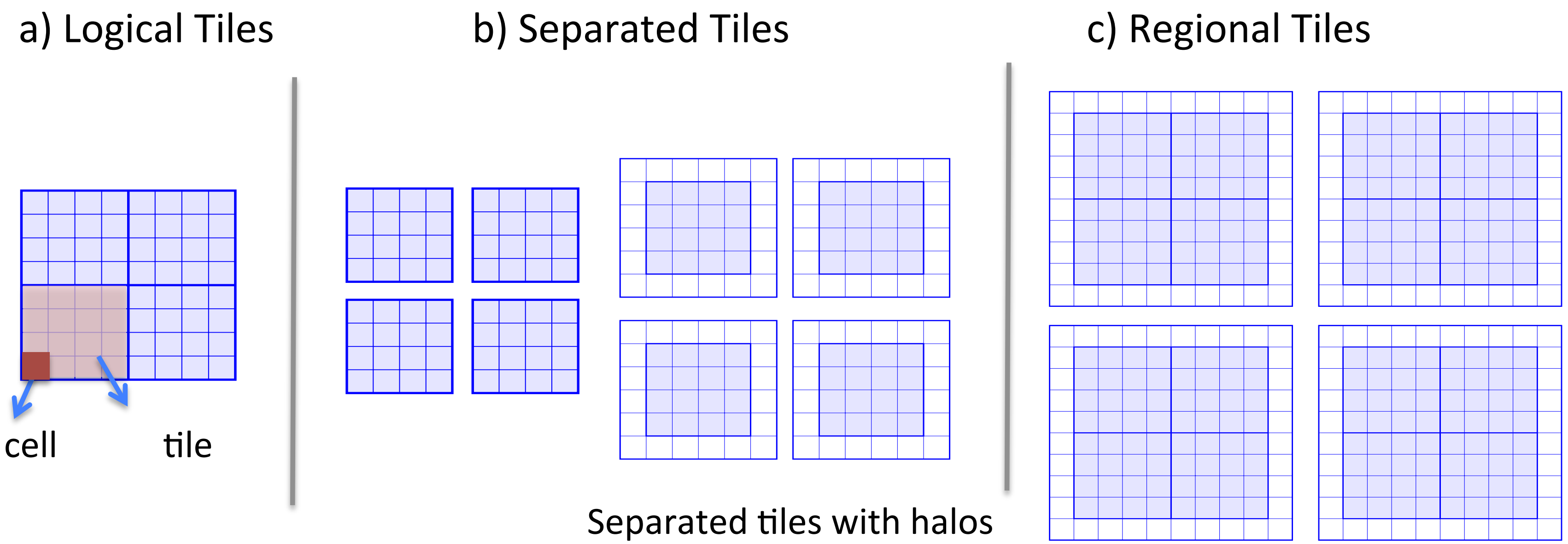
BERKELEY LAB

Sandia National Laboratories

Abstraction for Memory Layout

Didem Unat
Dan Quinlan

- **Support different layouts for various cache coherence scenarios**
- **Require minimum code modification when the memory layout is changed**
- **Memory layout options**
  - Specified at the array construction thru a flag or
  - `export DATA_LAYOUT={LOG | SEP | REG}`
- **The solvers remain unchanged !!!**



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**

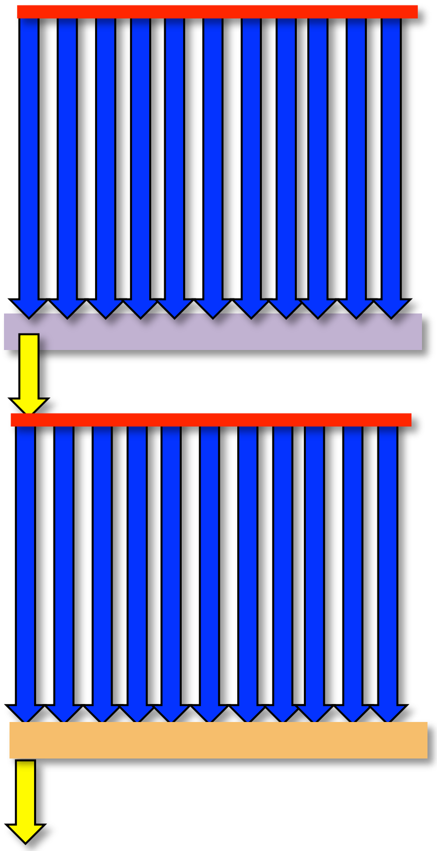
  (Programming Abstractions for Data Locality)

  http://www.padalworkshop.org/

Heterogeneity / Inhomogeneity Async Programming Models?

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

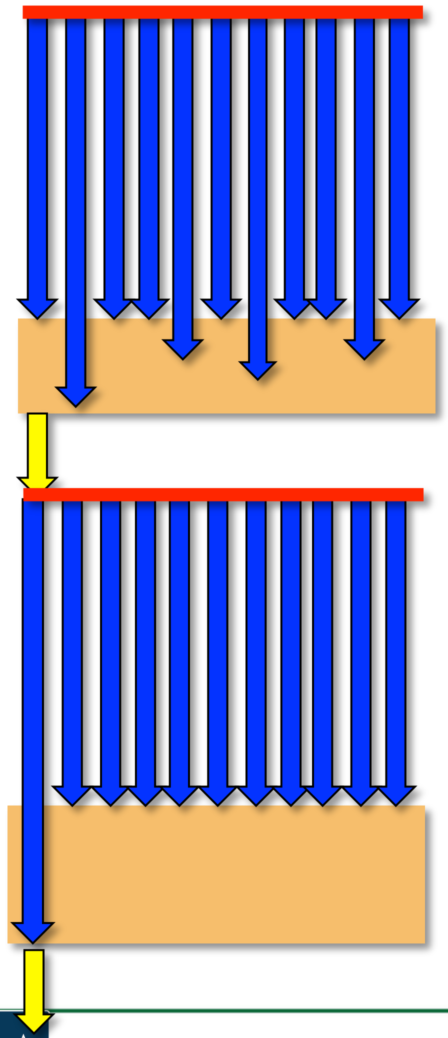
Bulk Synchronous Execution



- **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*

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

Bulk Synchronous Execution

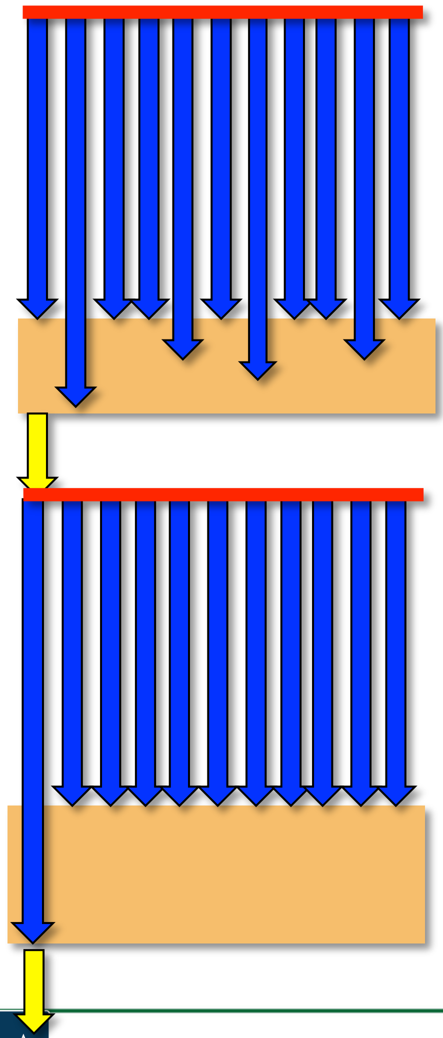


- **Heterogeneous compute engines (hybrid/GPU computing)**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  - *thermal throttling – no longer guarantee deterministic clock rate*
- **Nonuniformities in process technology creates non-uniform operating characteristics for cores 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 really 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*

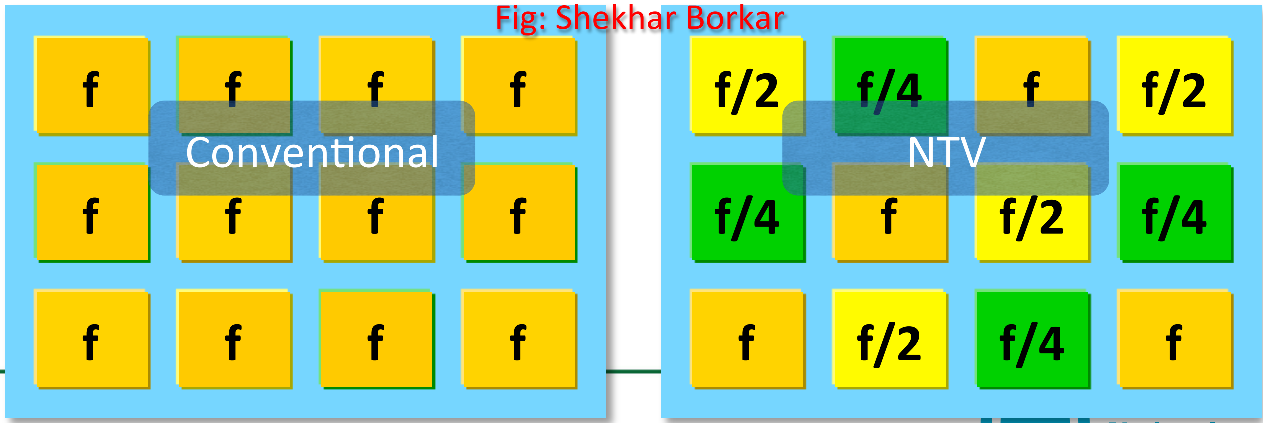


Fig: Shekhar Borkar

Conventional

f	f	f	f
f	f	f	f
f	f	f	f

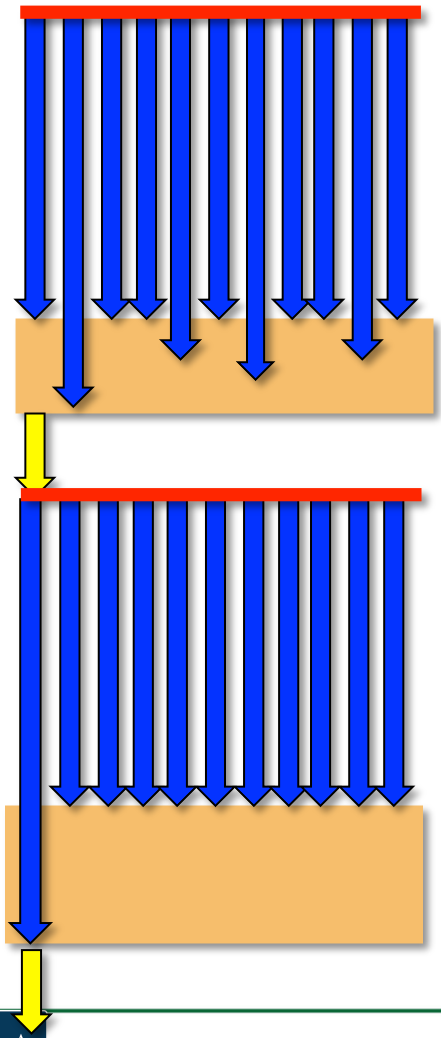
NTV

f/2	f/4	f	f/2
f/4	f	f/2	f/4
f	f/2	f/4	f

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

The really big opportunities for energy efficiency require codesign!

Bulk Synchronous Execution



- **Improving energy efficiency or performance of individual components doesn't really need co-design**
  - *Memory is faster, then odds are that the software will run faster*
  - *if its better, that's good!*
- **The really *big* opportunities to improve energy efficiency may require a shift in how we program systems**
  - *This requires codesign to evalute the hardware and new software together*
  - *HW/SW Interaction unknown (requires HW/SW codesign)*
- ***If software CANNOT exploit these radical hardware concepts (such as NTV), then it would be better to not have done anything at all!***

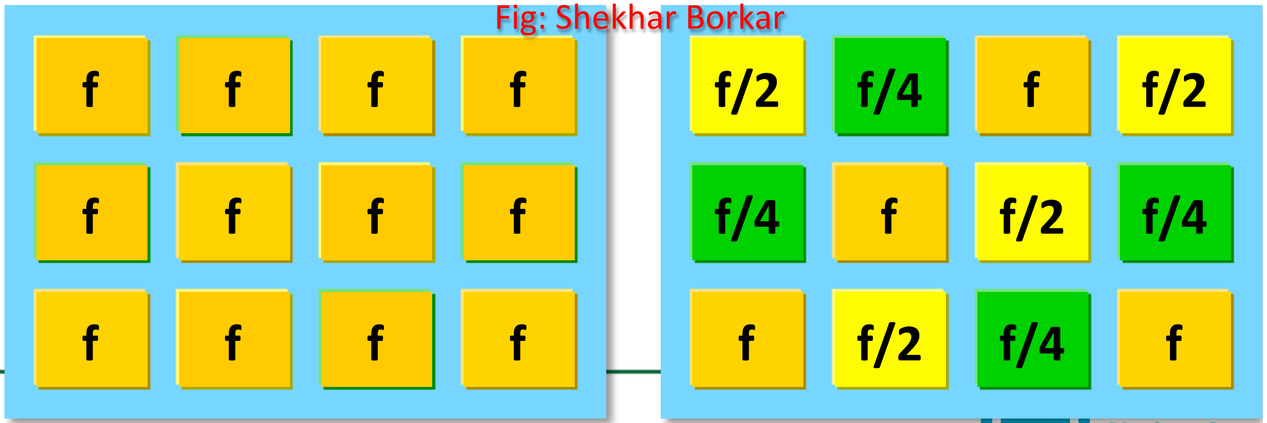


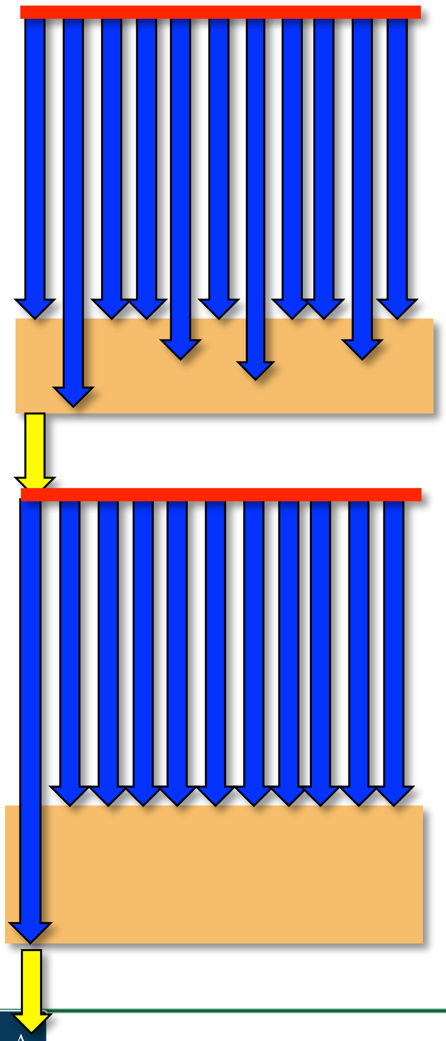
Fig: Shekhar Borkar

f	f	f	f
f	f	f	f
f	f	f	f

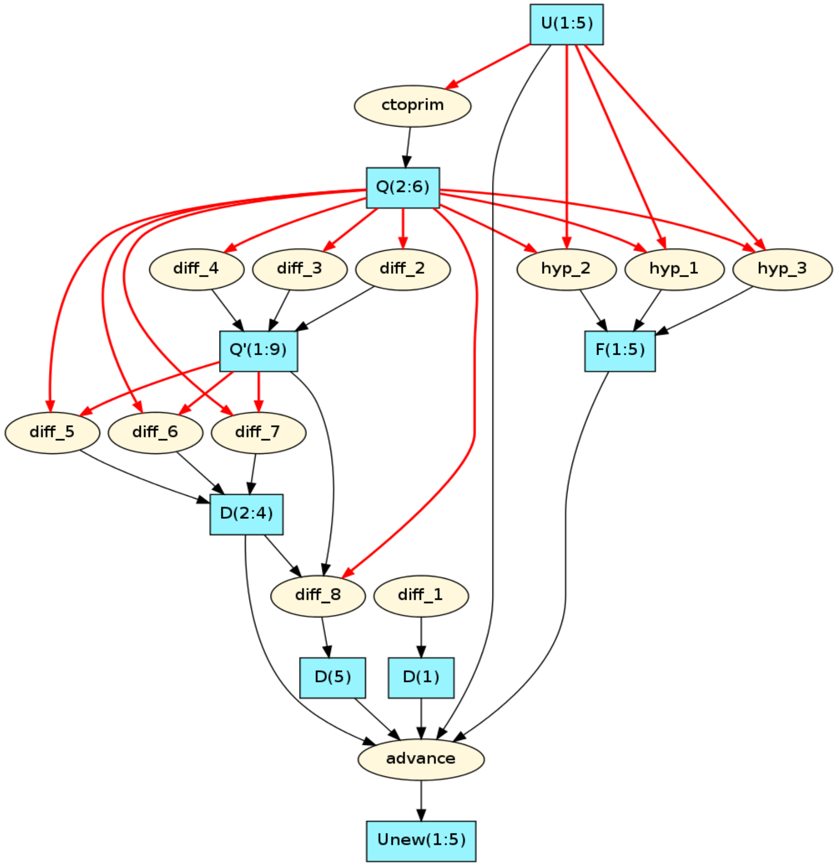
f/2	f/4	f	f/2
f/4	f	f/2	f/4
f	f/2	f/4	f

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

Bulk Synchronous Execution



Asynchronous Execution Model



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***

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**

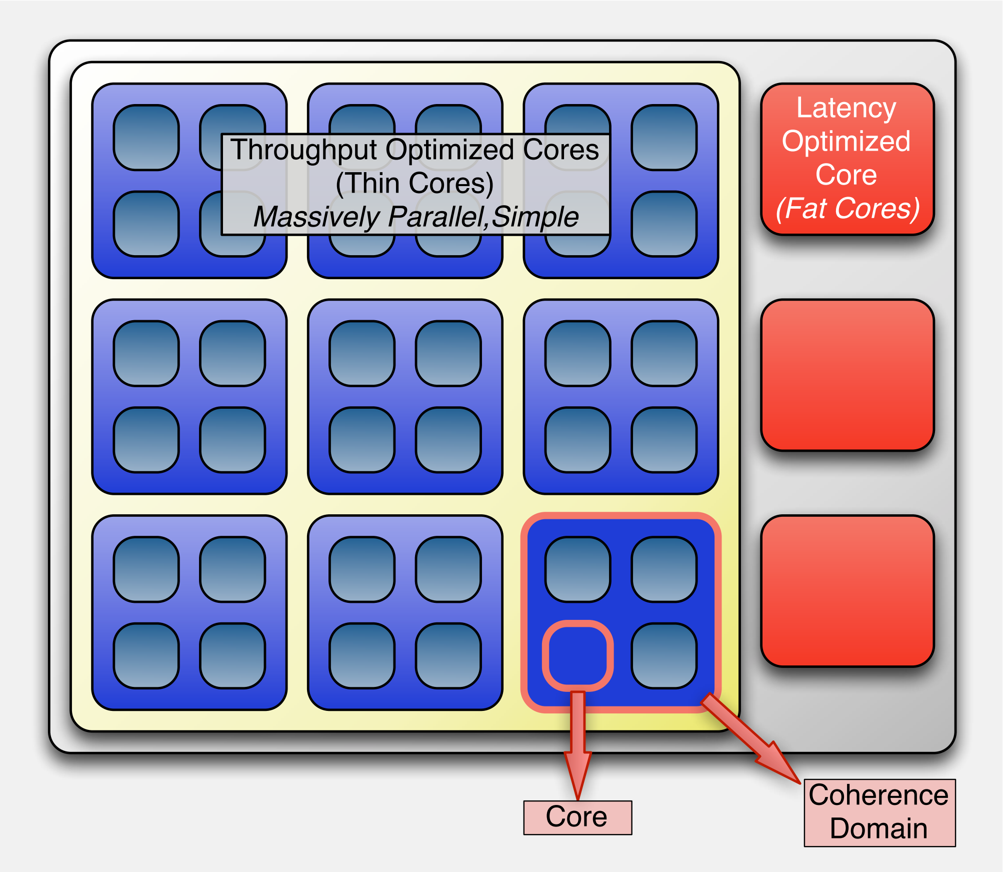
The End

For more information go to

http://www.cal-design.org/
http://www.nersc.gov/
http://crd.lbl.gov/

Abstract Machine Model

(what are the critical elements for spatial optimizaitons?)

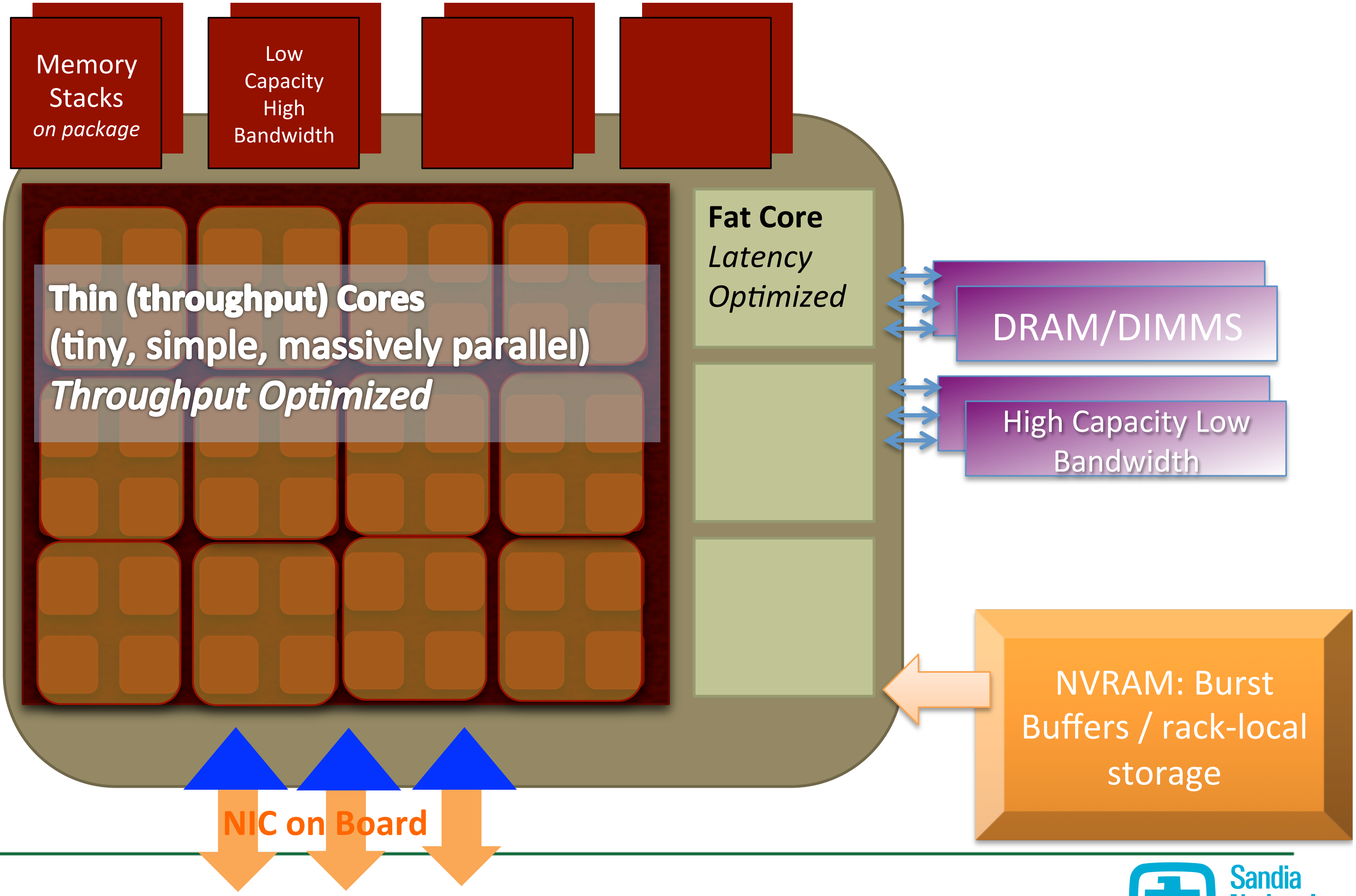


- **The number of cores on a chip will be on the order of 1000s**
  - *Expect 100x concurrency*
- **Maintaining cache coherence is NOT scalable**
  - *Expect coherence domains*
- **Flat and infinitely fast on-chip interconnect is NO longer practical**
  - *Expect complex NOCs*
- **Processing elements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

Move away from compute-centric to data-centric programming

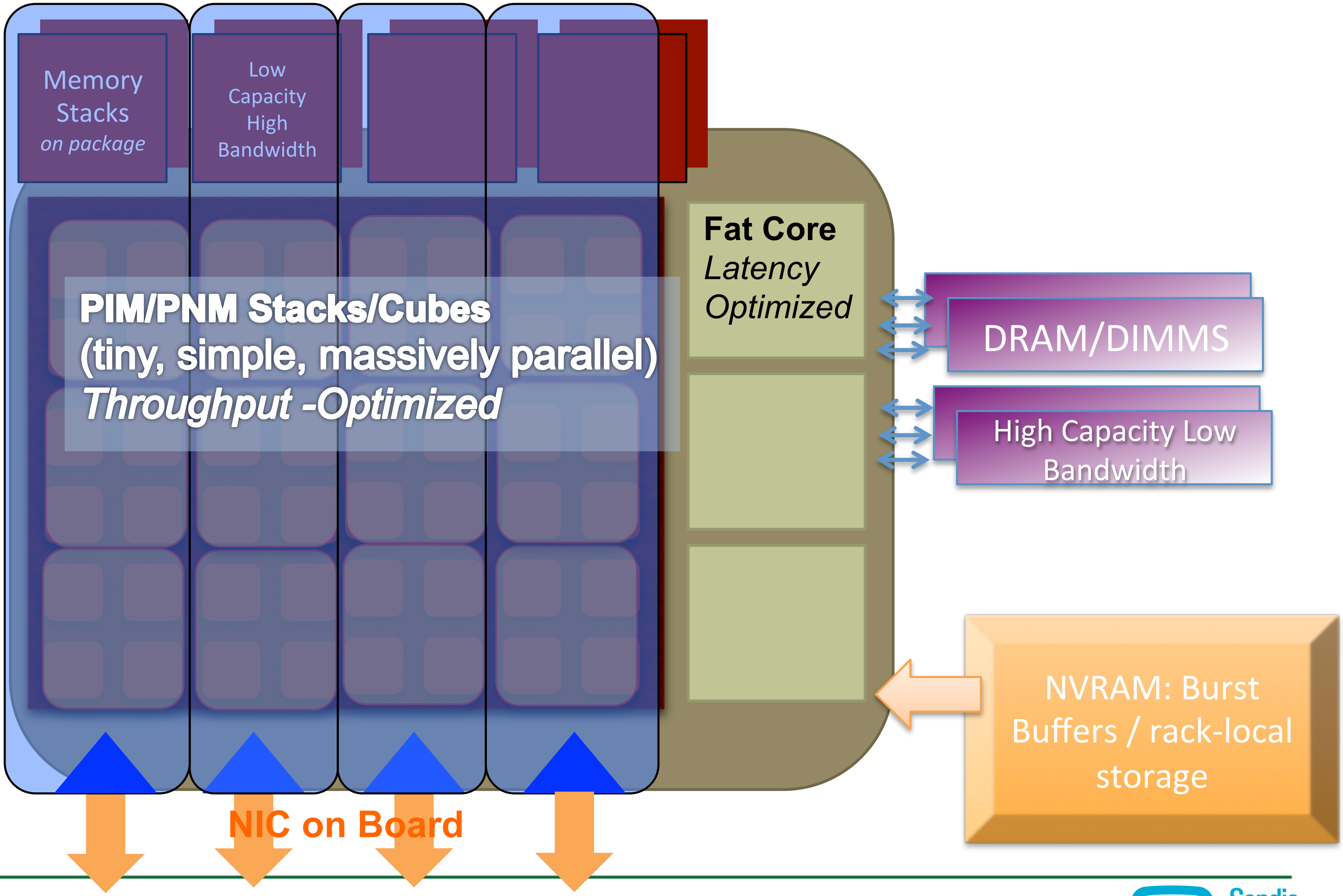
Emerging Fast Forward Exascale Node Architecture

Abstract Machine Model



Emerging Fast Forward Exascale Node Architecture

Abstract Machine Model


Memory Stacks
on package
Low Capacity High Bandwidth
Fat Core
Latency Optimized
PIM/PNM Stacks/Cubes
(tiny, simple, massively parallel)
Throughput -Optimized
DRAM/DIMMS
High Capacity Low Bandwidth
NVRAM: Burst Buffers / rack-local storage
NIC on Board