Exascale Programming Challenges: Adjusting to the "New Normal" for Computer Architecture

John Shalf
Department Head: Computer Science and Data Sciences (CSDS)
CTO: National Energy Research Scientific Computing Center (NERSC)

HPC 2014
July 8, Cetraro, Italy

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.

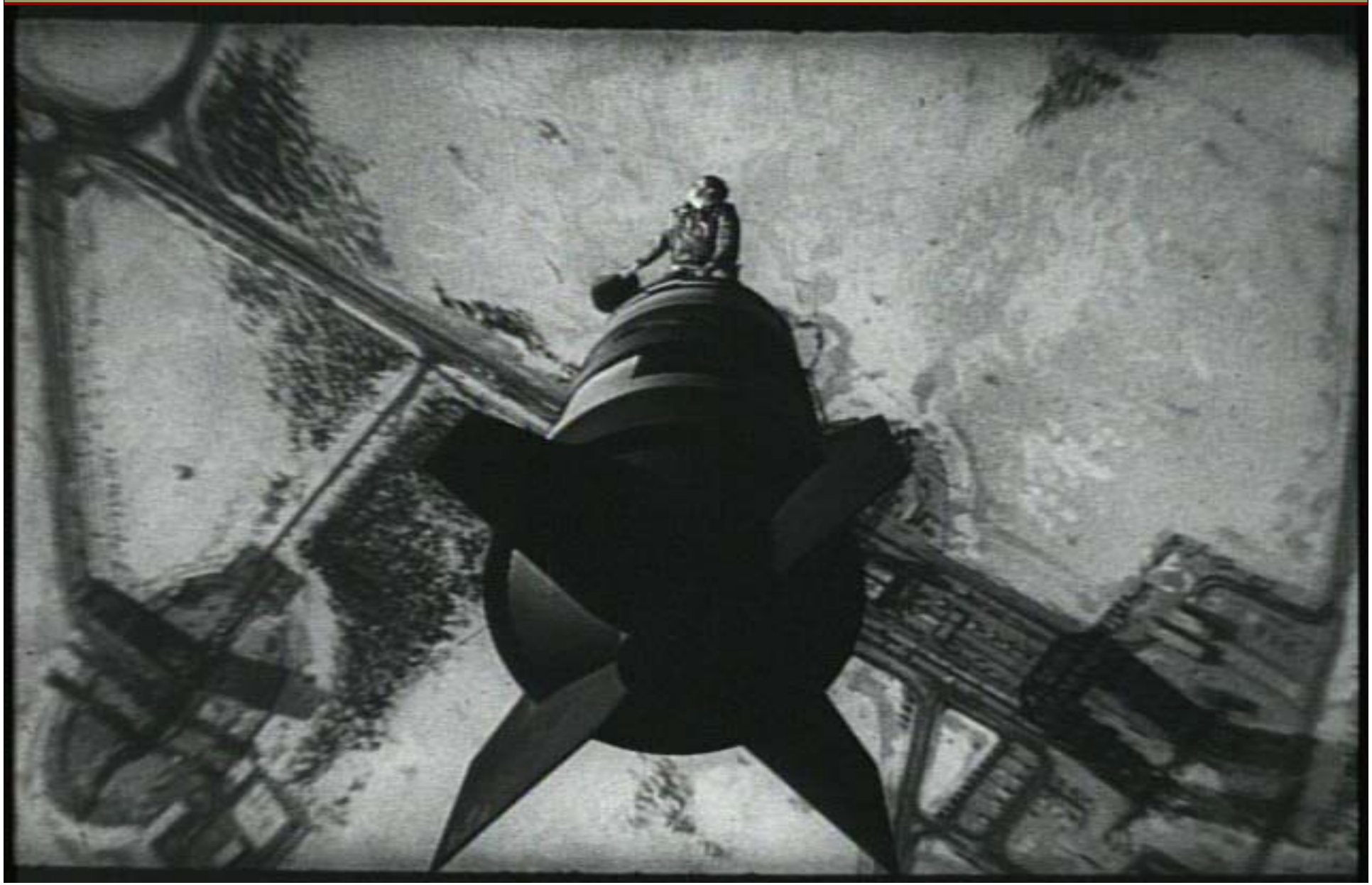
In 2004, however, a confluence of events changed forever the architectural landscape that underpinned MPI. Figure 1 summarizes the effects of these changes in terms of the year-over-year compound annual growth rate (CAGR) of several key system characteristics. This data, taken from an average of the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

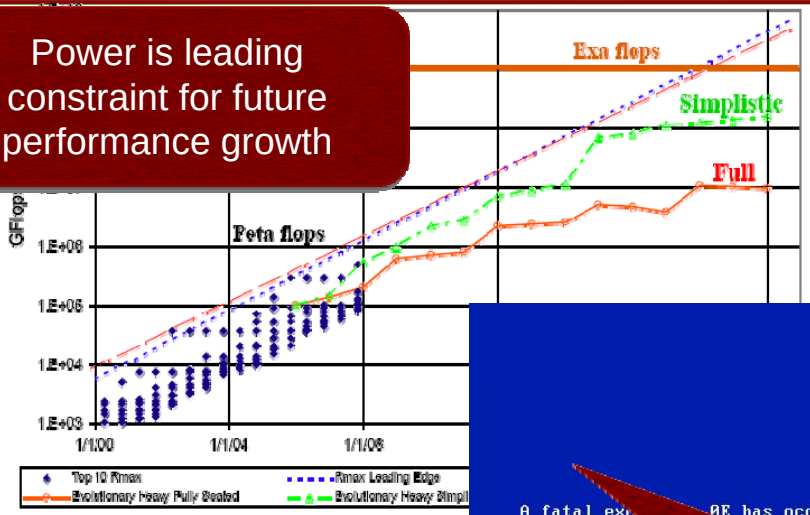
Moore's law has driven microprocessor architectures and high-performance computing (HPC) for decades. While variously interpreted as saying that microprocessor performance and memory chip density increase exponentially over time, the real statement is that a transistor's key linear dimensions (its *feature*

Original Title:
How I learned to Stop Worrying and Love Exascale

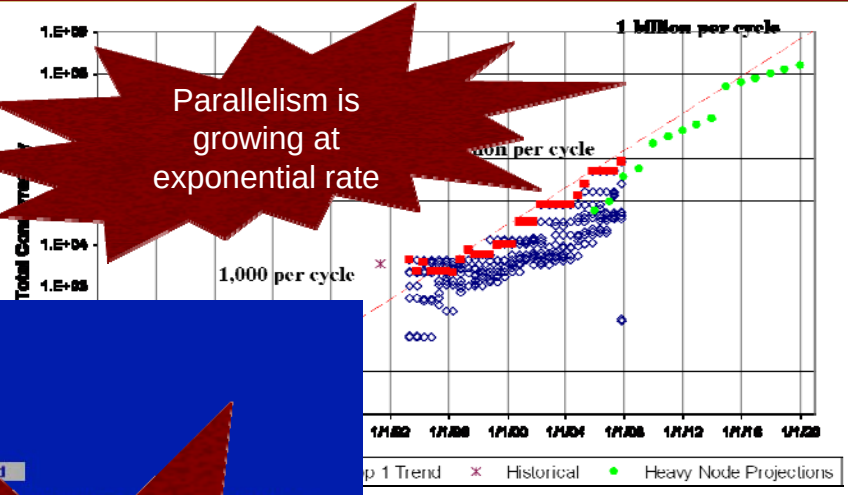


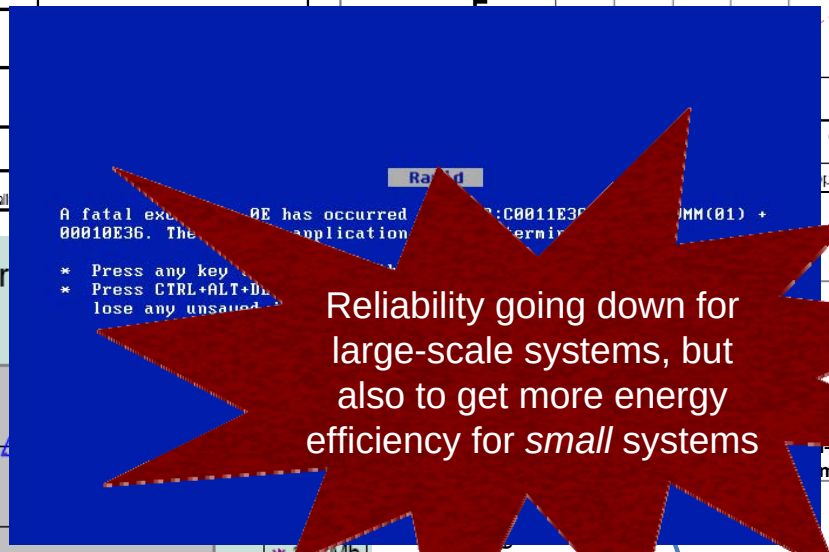
Technology Challenges for the Next Decade

Power is leading constraint for future performance growth

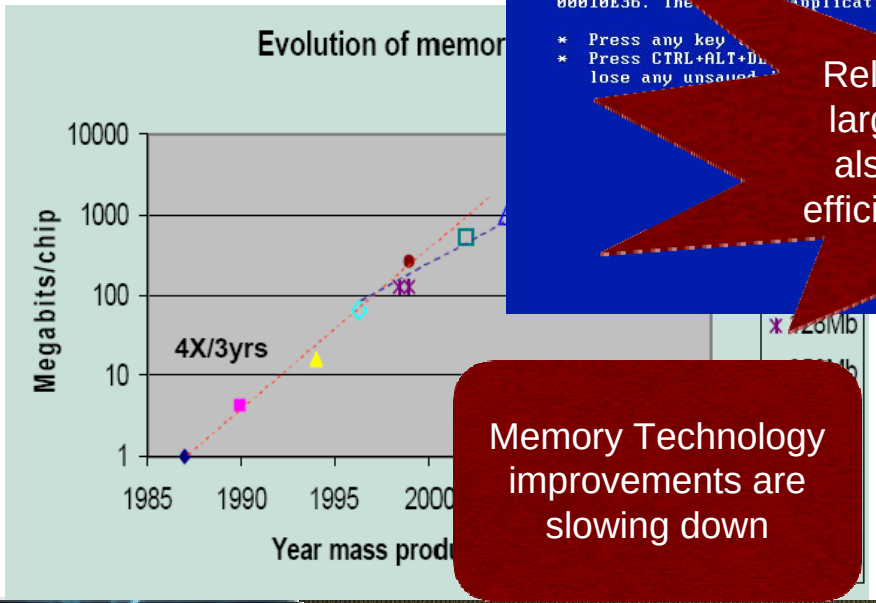


Parallelism is growing at exponential rate

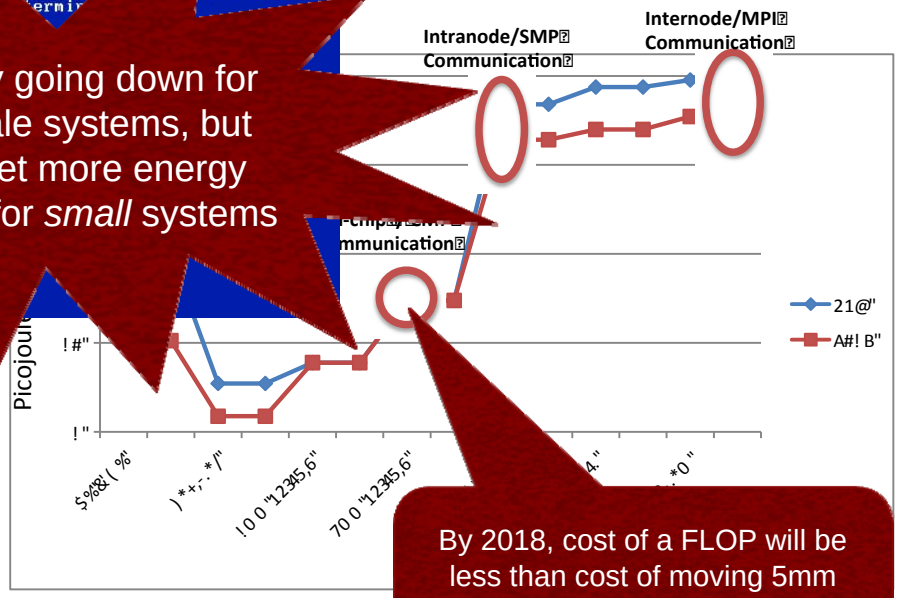




Reliability going down for large-scale systems, but also to get more energy efficiency for *small* systems



Memory Technology improvements are slowing down

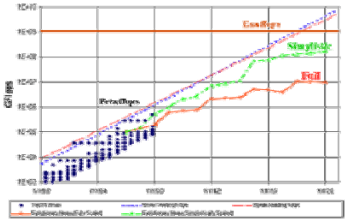


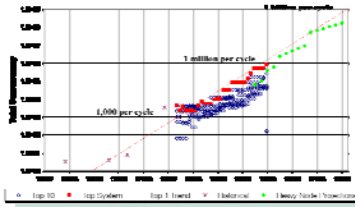
By 2018, cost of a FLOP will be less than cost of moving 5mm across the chip's surface (locality will *really* matter)

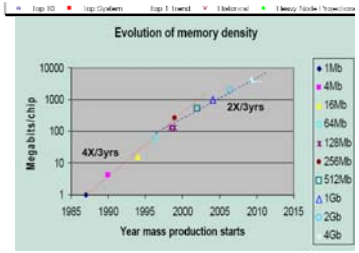
Whats wrong with current HPC Systes?

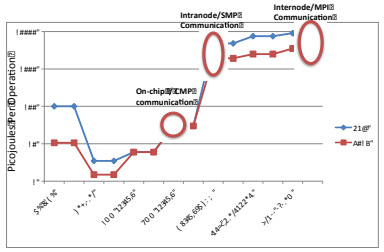
Designed for Constraints from 30 years ago! ***(wrong target!!)***

Old Constraints	New Constraints
• ***Peak clock frequency*** *as primary limiter for performance improvement*	• ***Power*** *is primary design constraint for future HPC system design*
• **Concurrency**: Modest growth of parallelism by adding nodes	• **Concurrency:** *Exponential growth of parallelism within chips*
• **Cost:** *FLOPs* are biggest cost for system: *optimize for compute*	• **Cost:** *Data movement dominates: optimize to minimize data movement*
• **Memory scaling***: maintain byte per flop capacity and bandwidth*	• **Memory Scaling:** *Compute growing 2x faster than capacity or bandwidth*
• **Locality***: MPI+X model (uniform costs within node & between nodes)*	• **Locality***: must reason about data locality and possibly topology*
• ***Uniformity****: Assume uniform system performance*	• ***Heterogeneity****: Architectural and performance non-uniformity increase*
• **Reliability***: It's the hardware's problem*	• **Reliability***: Cannot count on hardware protection alone*









Fundamentally breaks our current programming paradigm and computing ecosystem

The Programming Systems Challenge

Programming Models and Abstractions are a Reflection of the Underlying Machine Architecture

- *Express what is important for performance*
- *Hide complexity that is not consequential to performance*

Current Programming Abstractions are Increasingly Mismatched with Underlying Hardware Architecture

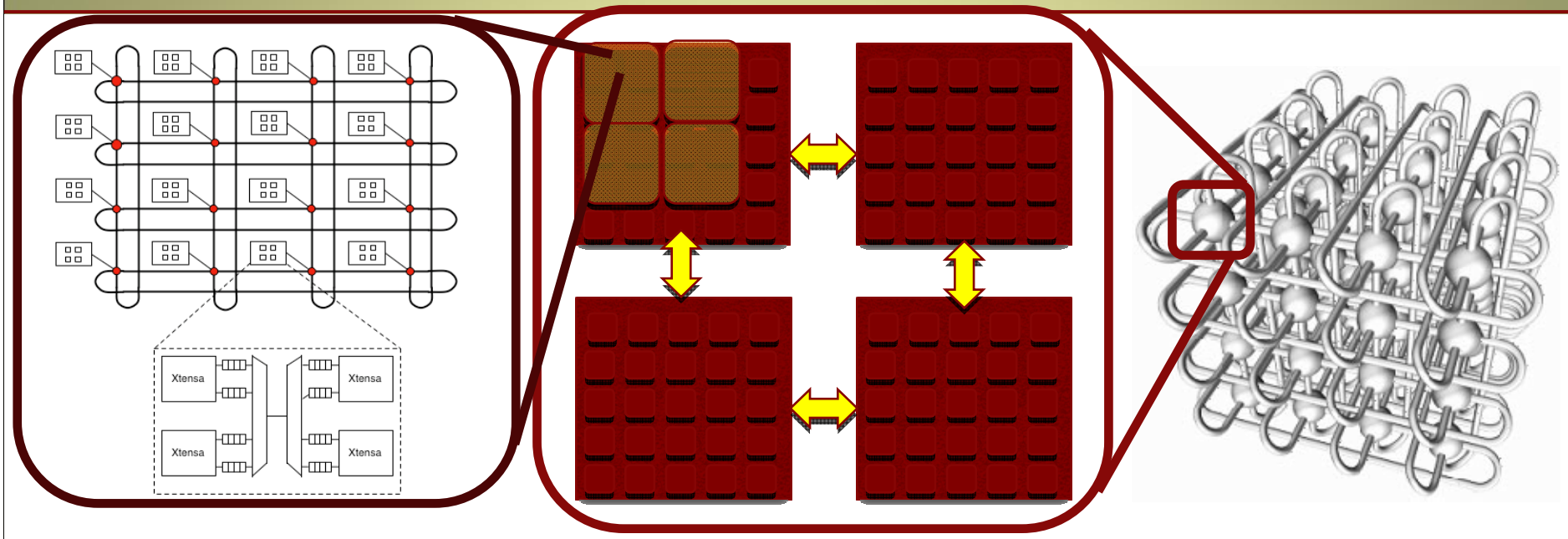
- *Changes in computer architecture trends/costs*
- *Performance and programmability consequences*

Technology changes have deep and pervasive effect on ALL of our software systems *(and how we program them)*

- *Change in costs for underlying system affect what we expose*
- *What to **virtualize***
- *What to make more **expressive/visible***
- *What to **ignore***

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)

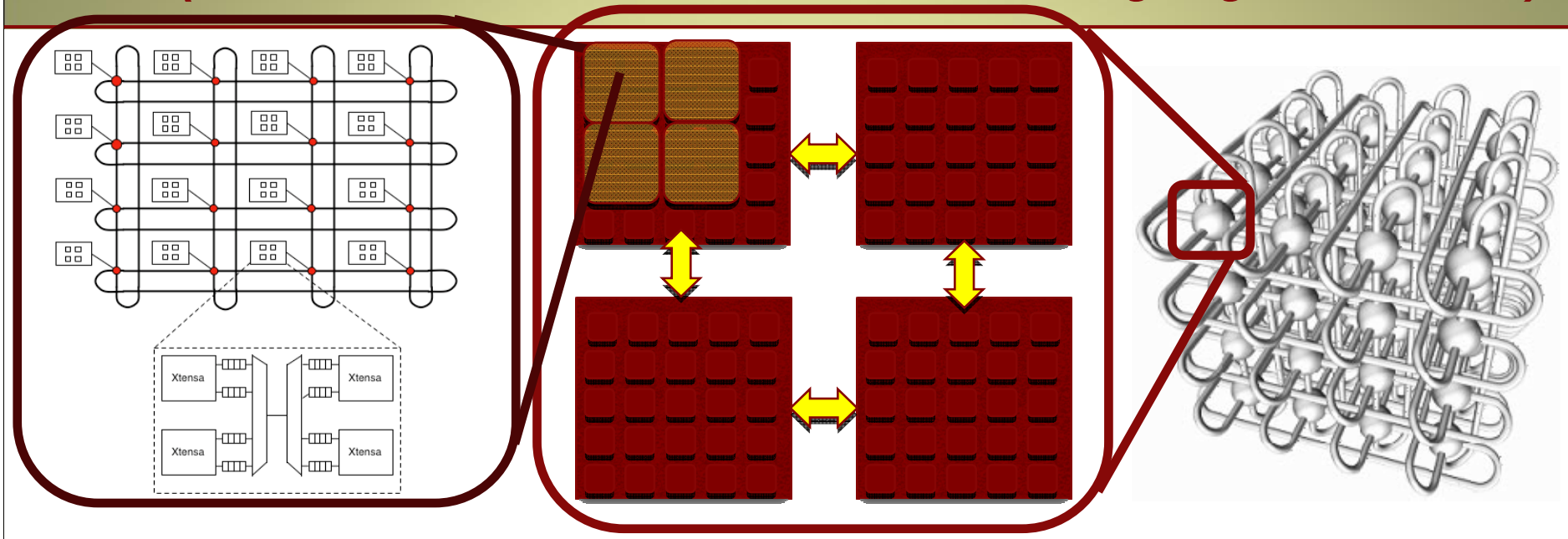
Fault Model for Node
- FIT rates, Kinds of faults
- Granularity of faults/recovery

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

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

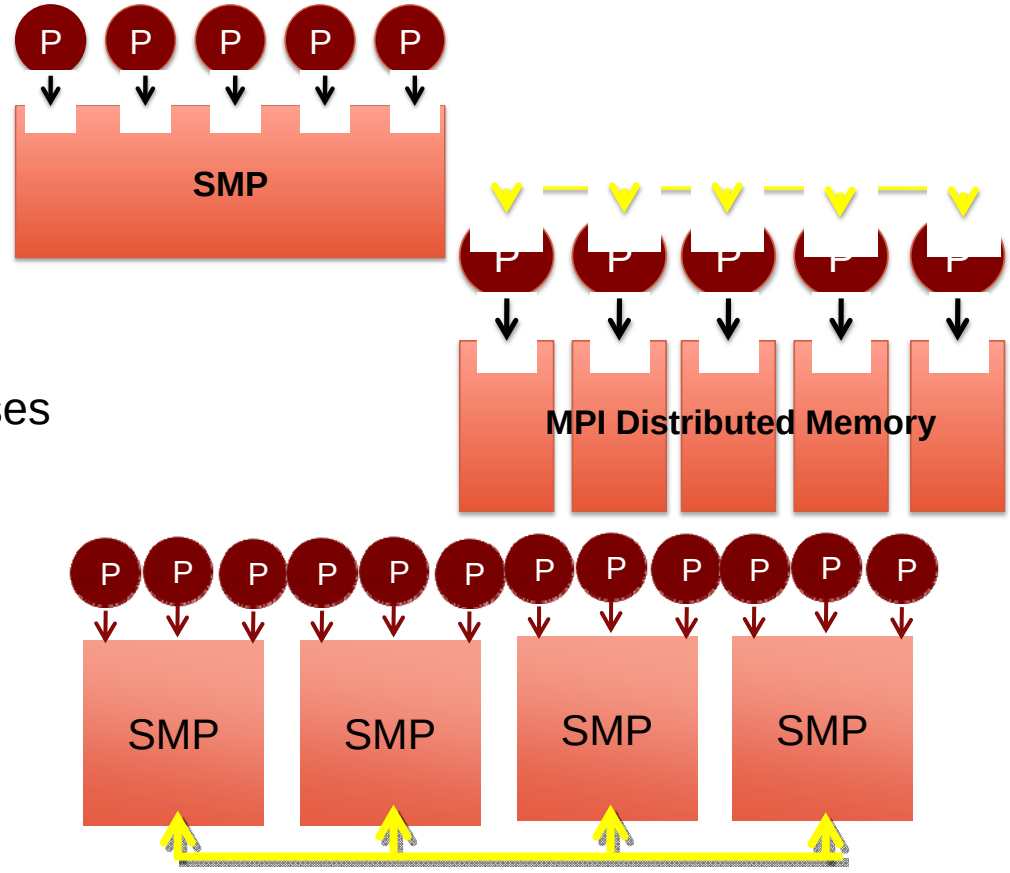
- No notion of non-local access
- int [nx][ny][nz];

Cluster: Distributed memory model

- CSP: Communicating Sequential Processes
- No unified memory
- int [localNX][localNY][localNZ];

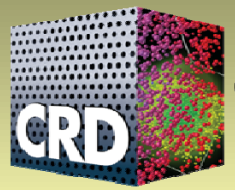
2-level Locality Model (core, node)

- Candidate Type Architecture (CTA)
- MPI+X model (for all practical purposes)



Whats Next?

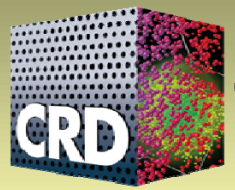
2-Level MPI+X is dominant, but insufficient!



COMPUTATIONAL RESEARCH DIVISION

What does an exascale node look like?

… at least as far as we know from current processor/system roadmaps

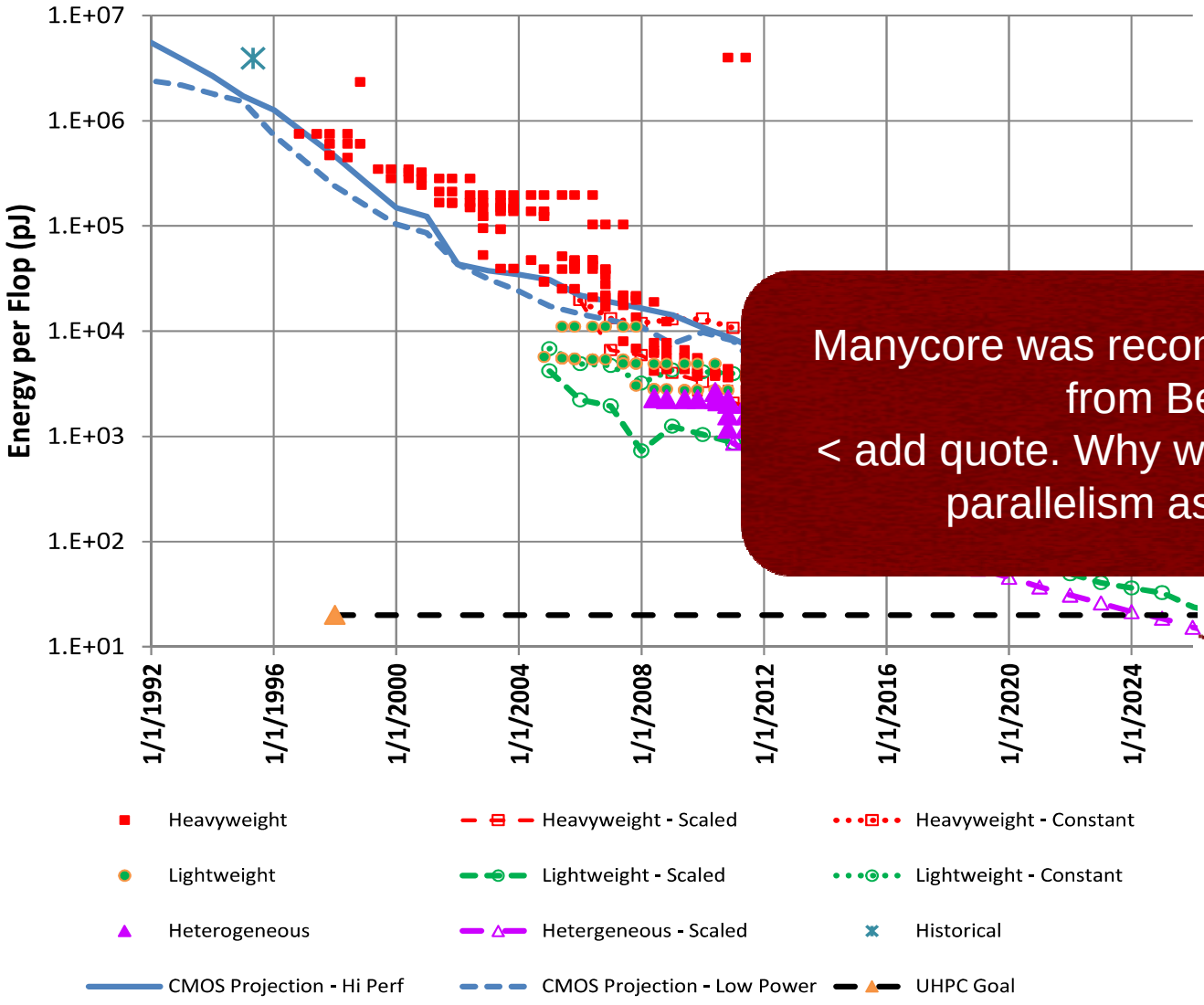


COMPUTATIONAL RESEARCH DIVISION

Physics brings the world together because we are all subjected to the same laws

Richard P. Feynman

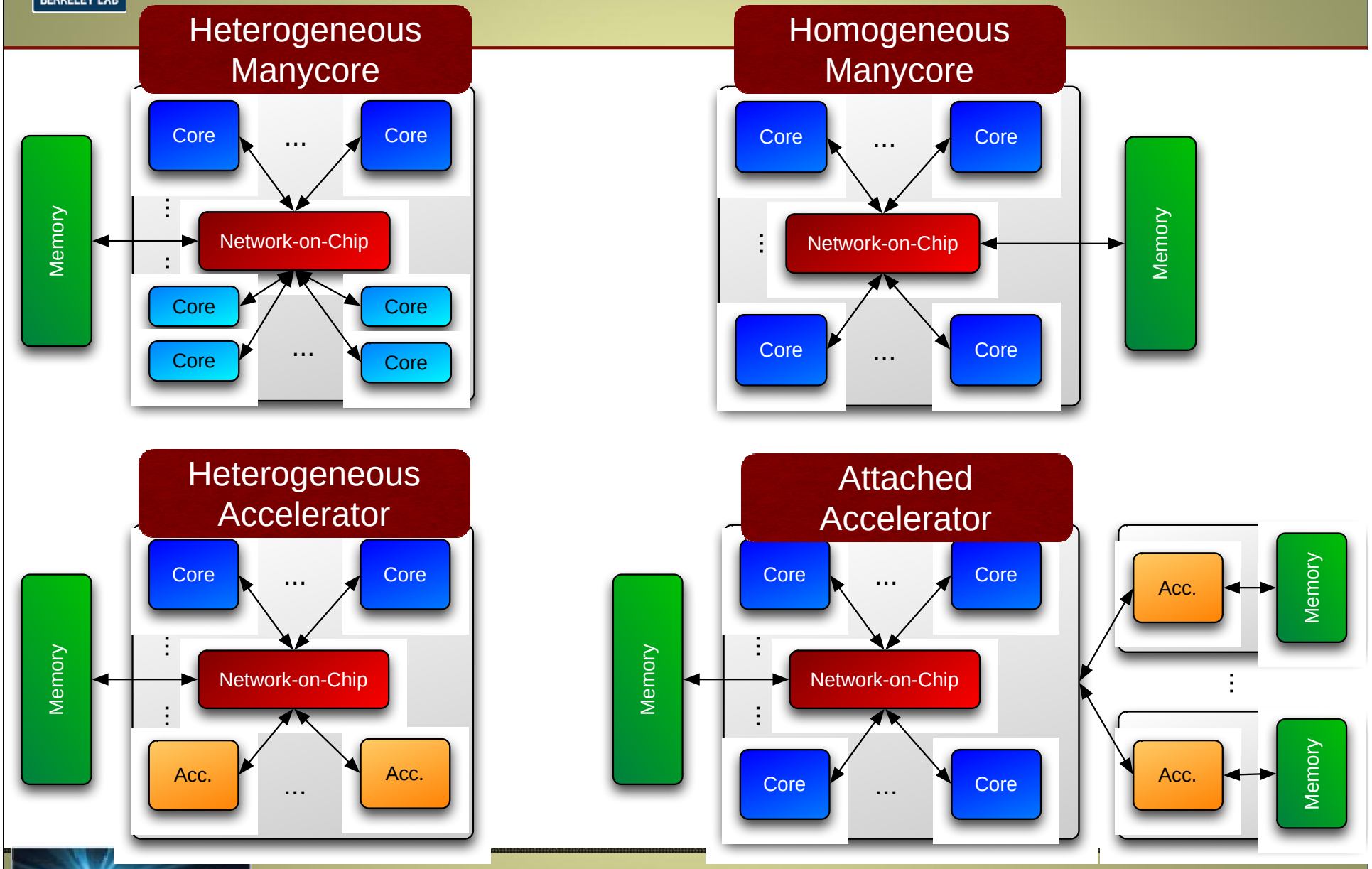
Hybrid Architectures:
Moving from side-show to necessity



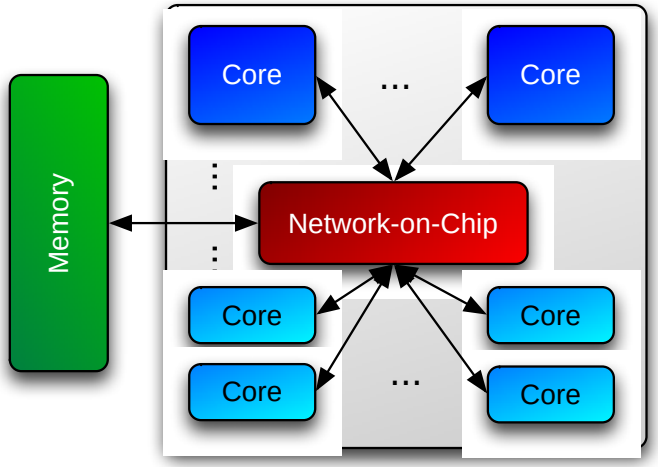
Manycore was recommendation of "View from Berkeley"
< add quote. Why would you exacerbate parallelism as a problem?>

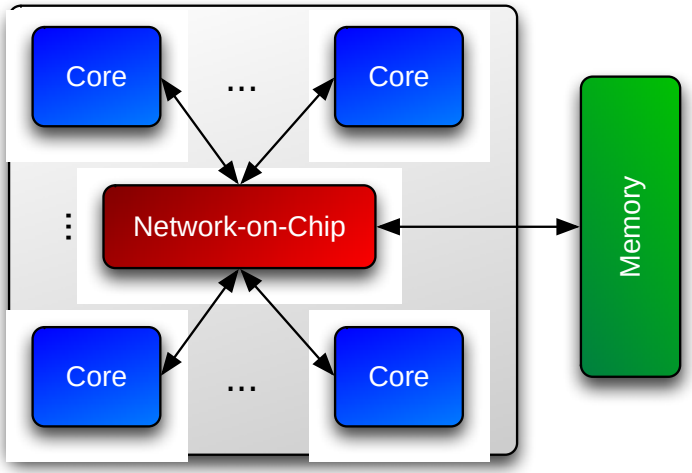
Hybrid or manycore is the only approach that crosses the exascale finish line

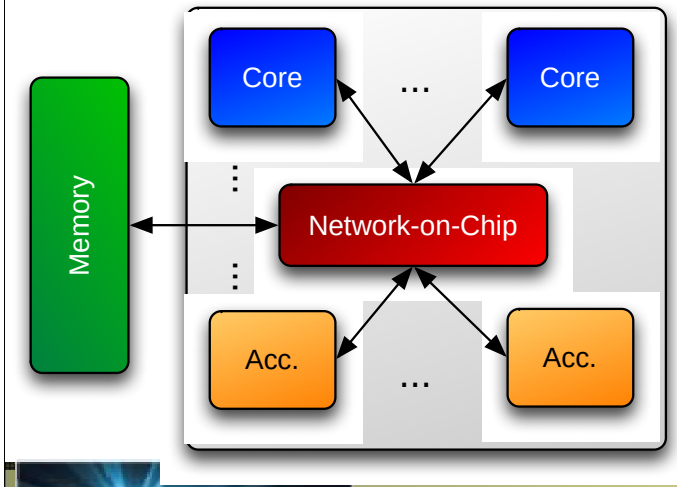
Current Architectural Families

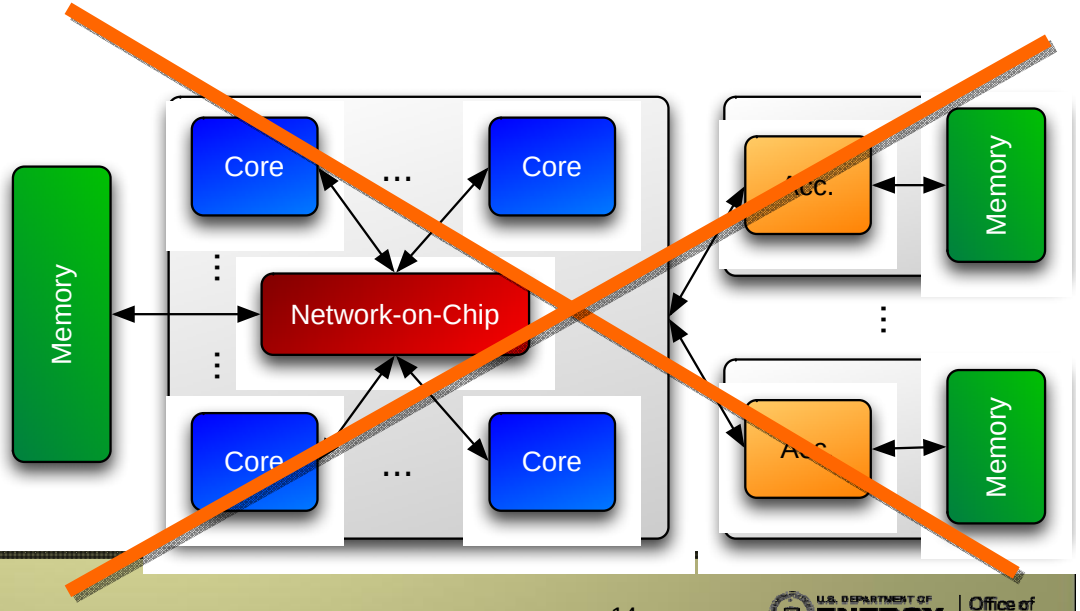


Current Architectural Families

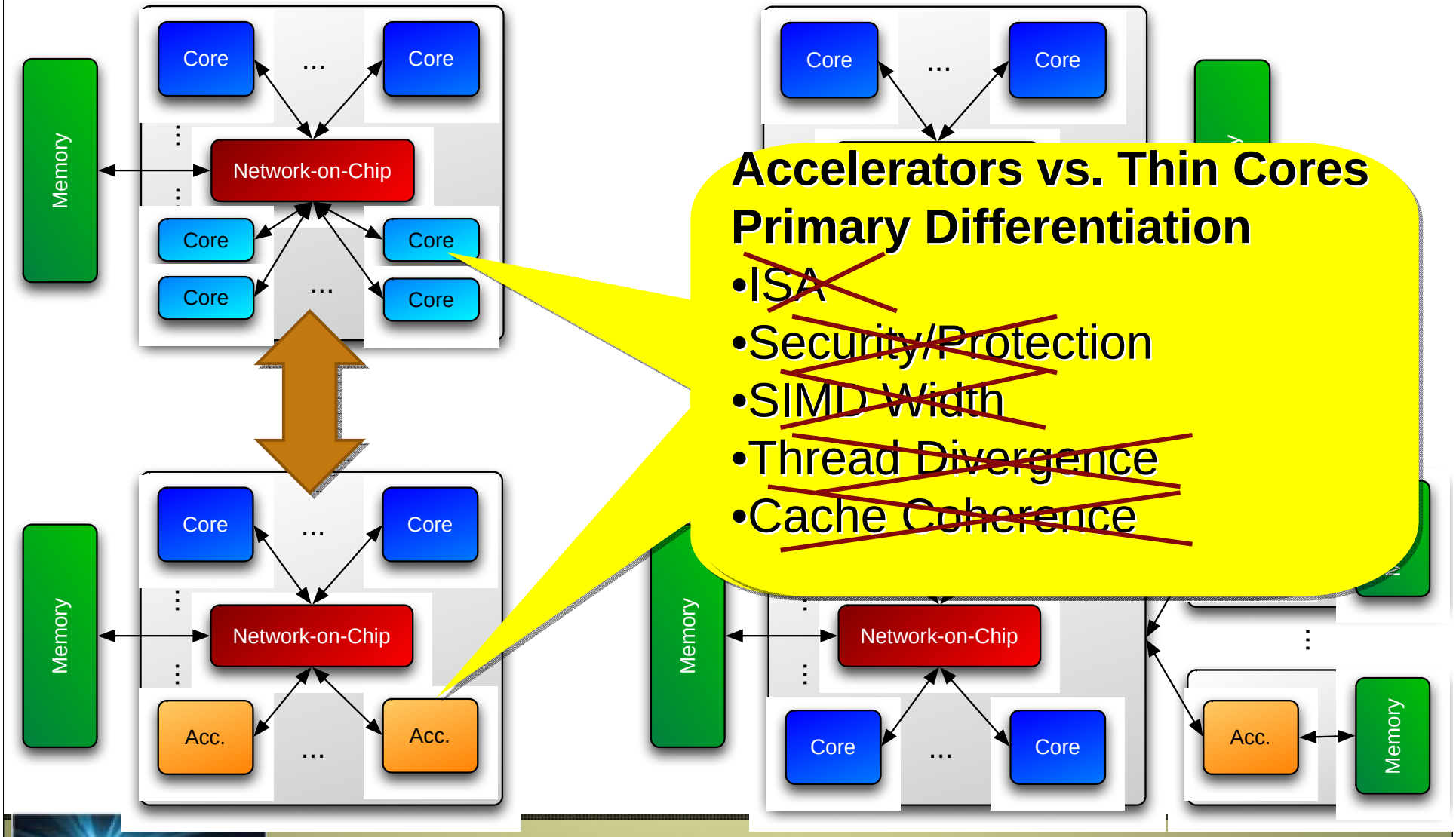






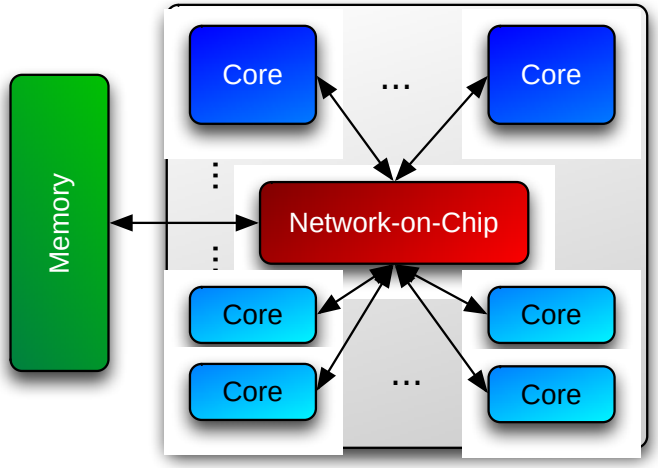


Architectural Convergence

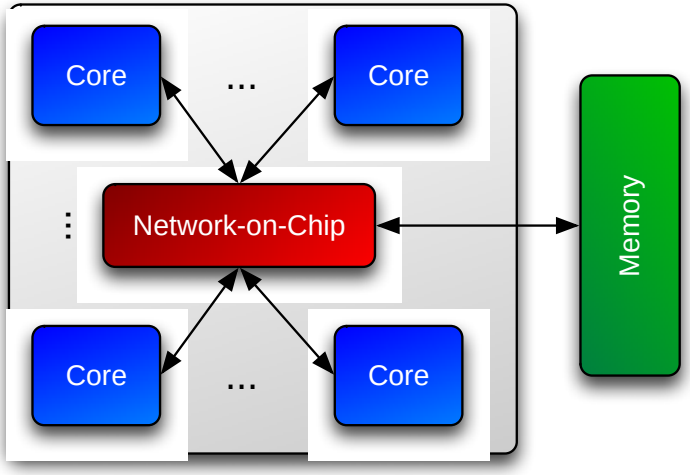


Reducing Space of Choices

Heterogeneous Manycore



Homogeneous Manycore

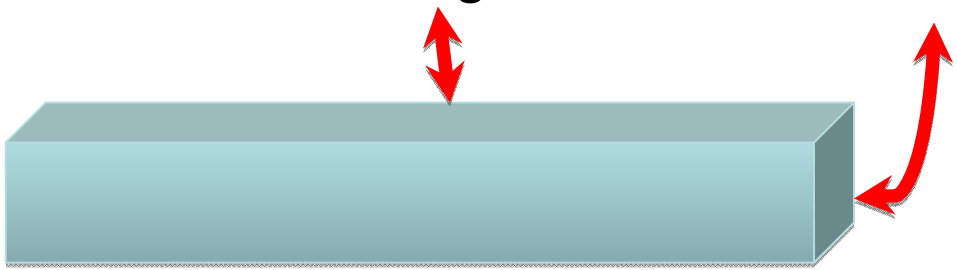


IBM/NVIDIA
AMD APU
MontBlanc

Intel Xeon Phi
~~**BlueGene/Q**~~

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

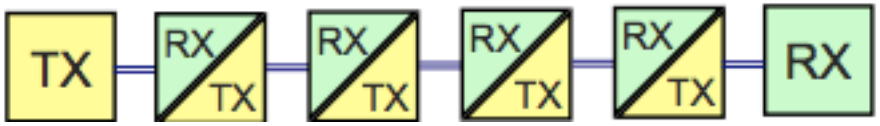
- **Cost to move a bit on copper wire:**
  - **Power = Bitrate * Length / cross-section area**



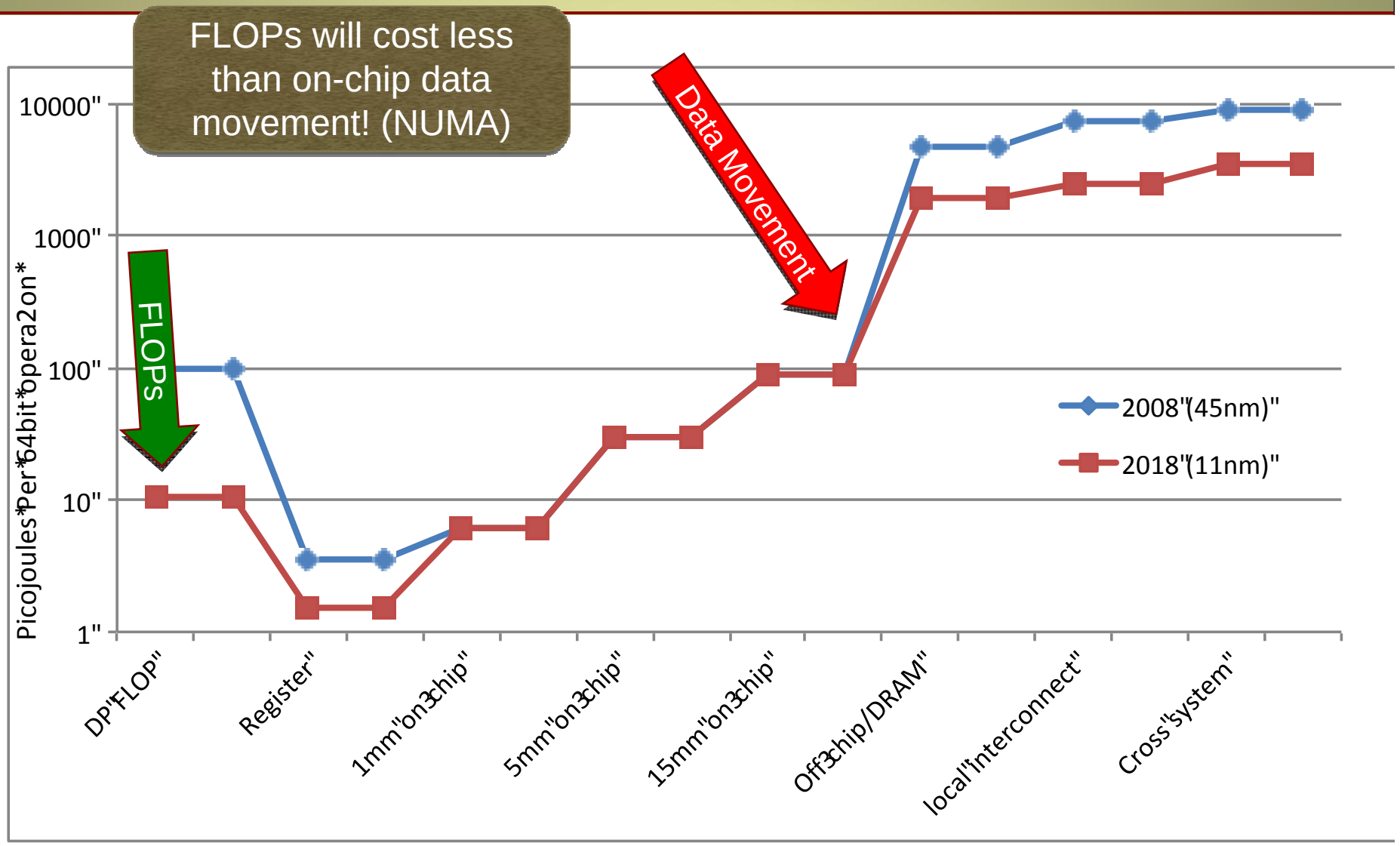
- **Wire data capacity constant as feature size shrinks**
- ***Cost to move bit proportional to distance***
- ***~1TByte/sec max feasible off-chip BW (10GHz/pin)***
- ***Photonics reduces distance-dependence of bandwidth***

Photonics requires no redrive and passive switch little power

Copper requires to signal amplification even for on-chip connections



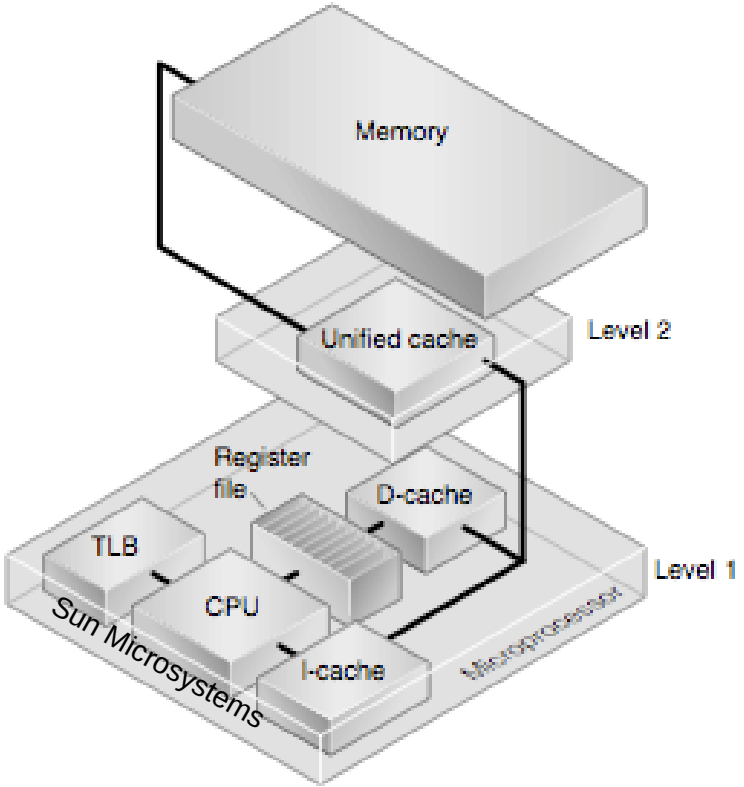
Cost of Data Movement Increasing Relative to Ops

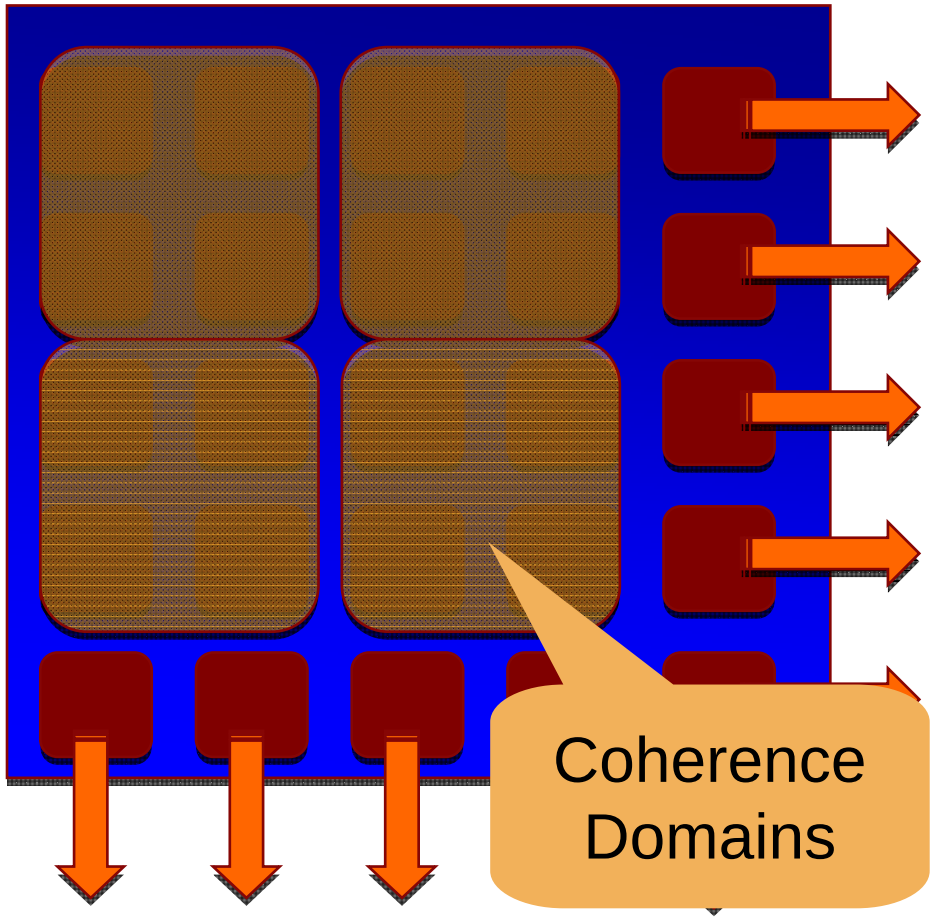


Data Locality Management *within a Node*

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

Power Constraint

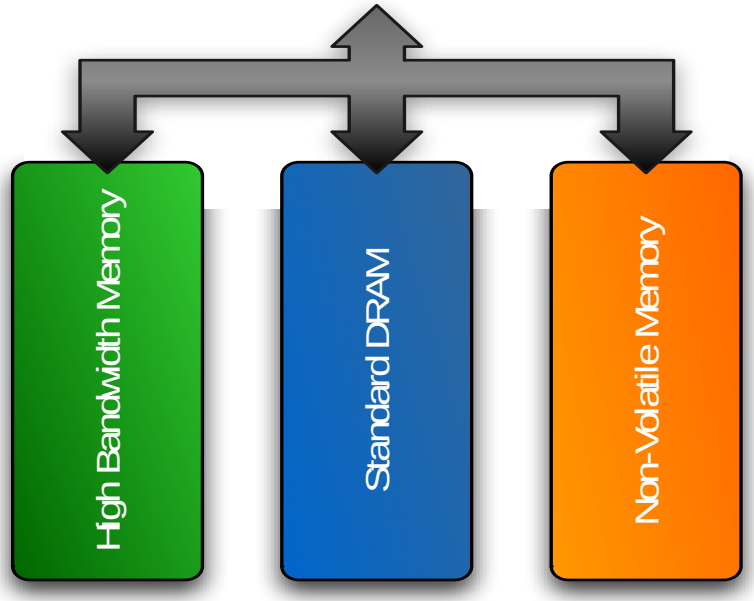
Bandwidth\Capacity	16 GB	32 GB	64 GB	128 GB	256 GB	512 GB	1 TB
4 TB/s							
2 TB/s	Stack/PNM						
1 TB/s			Interposer				
512 GB/s				HMC Organic			
256 GB/s							
128 GB/s						DIMM	
64 GB/s							NVRAM

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)

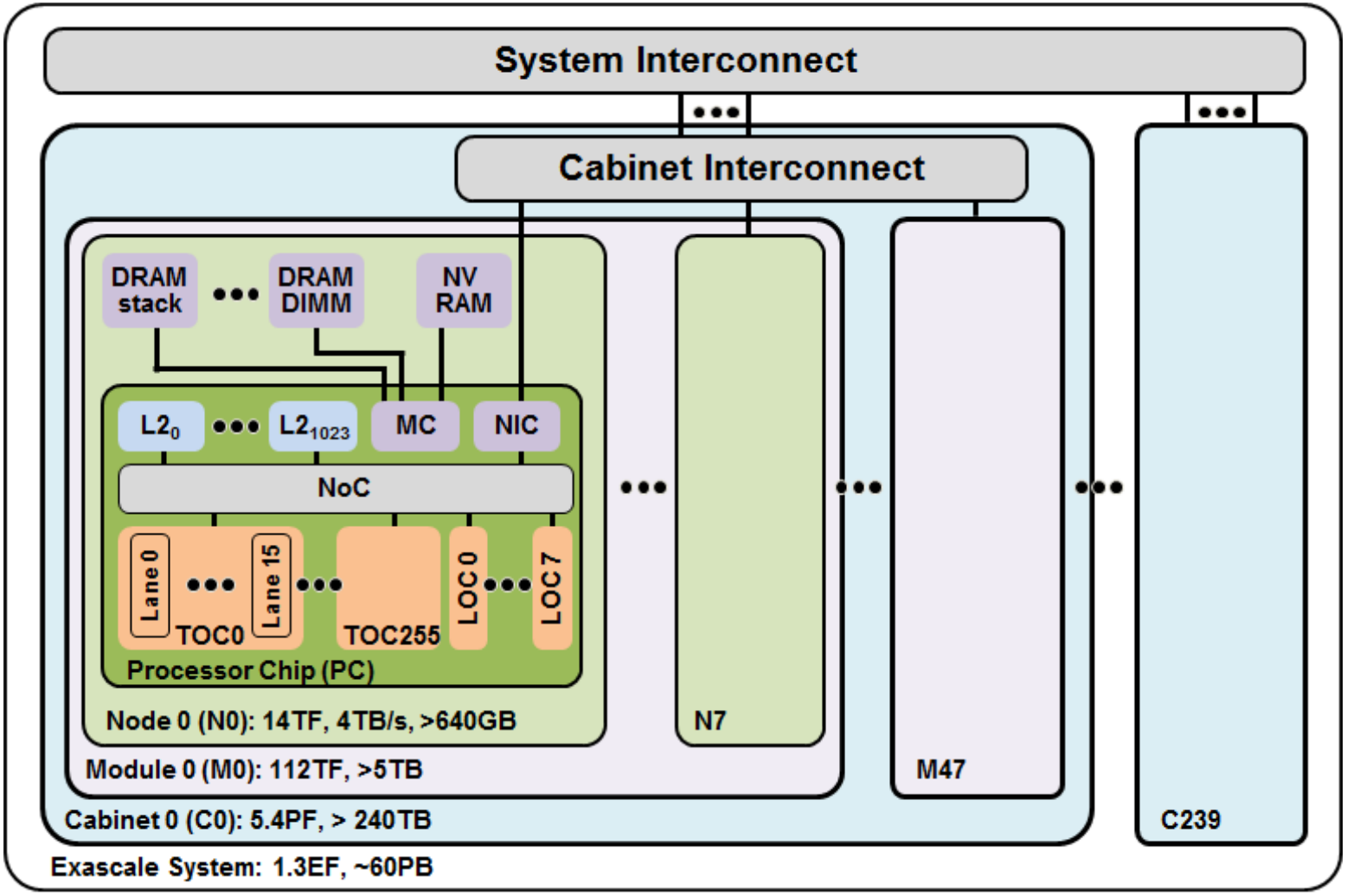
New Paradigm

- **DDR4**: ~1 byte per flop capacity w
  < 0.01 bytes/flop BW
- **Stacked Memory**: ~1 byte per flop capacity
  < 0.01 bytes/flop capacity
- **Non-Volatile Memory**
  Consumes more energy on write than read



NVIDIA's Gryphon Processor Concept

(Yes… this is the public version)

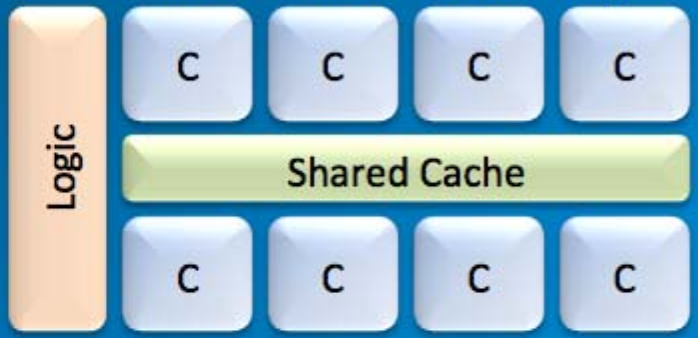


Shekhar Borkar: Intel

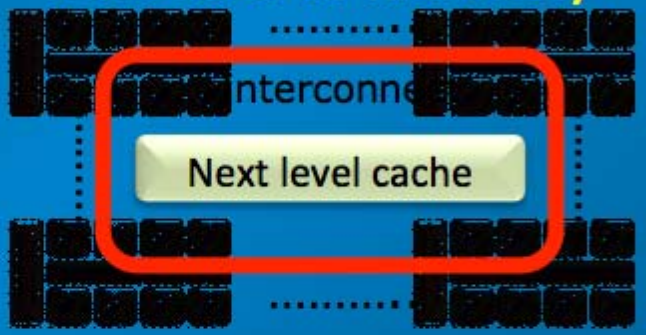
Straw-man Exascale Processor

Simplest Core

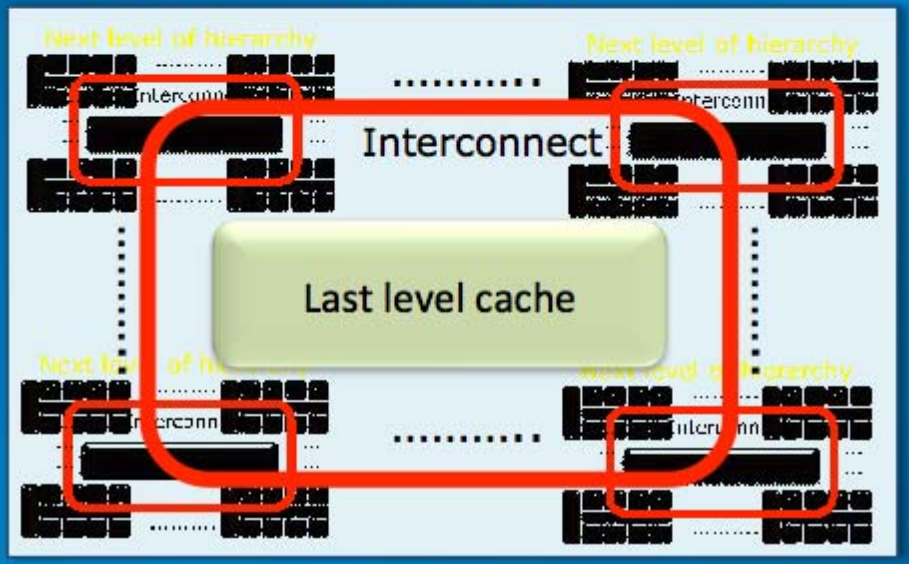
First level of hierarchy



Next level of hierarchy

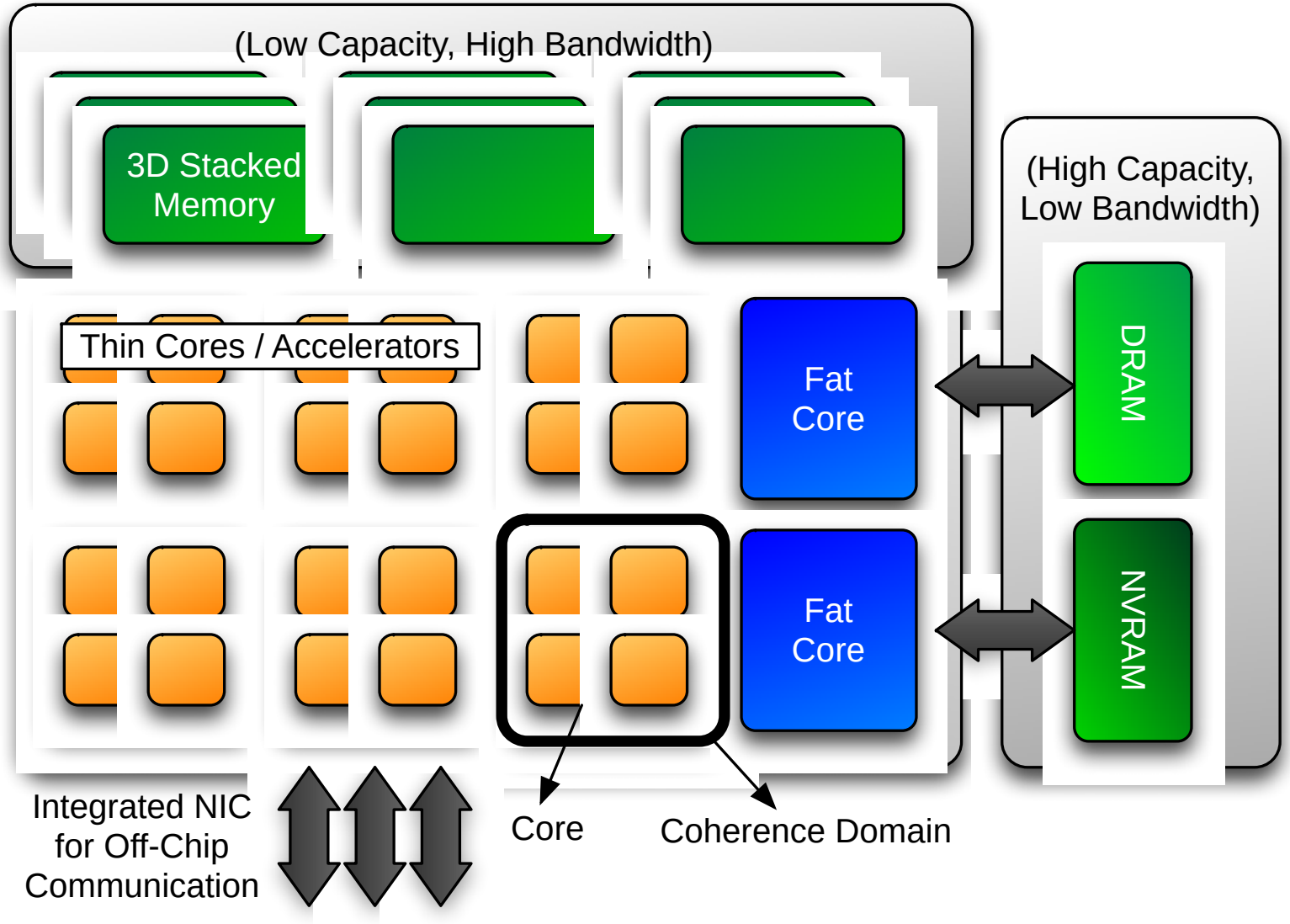


Processor



Technology	7nm, 2018
Die area	500 mm2
Cores	2048
Frequency	4.2 GHz
TFLOPs	17.2
Power	600 Watts
E Efficiency	34 pJ/Flop

Abstract Machine Model for Exascale



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

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?

What do the programmers/code-teams want?

2009 Exascale Roadmapping Workshop code teams *(want 10x)*

"Willing to change everying, but want to minimize the number of lines of code I need to change to get good performance when I move to a new machine"

Minimizing code changes to get performance is defined as "performance portability"

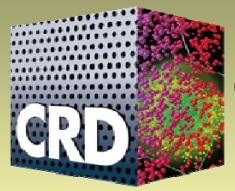
- Naturally high performance code if you program to the right abstractions
- "good performance by construction"

Is it a revolution… or is it really a revolution? *(discussions with Paul)*

- Vector to MPI we still preserved 90% of our F77 code
- Was that a revolution or expensive evolutionary transition?

First: what is the abstract model to represent the machine

Second: what are the correct programming abstractions productively map programs to that abstract model of the machine

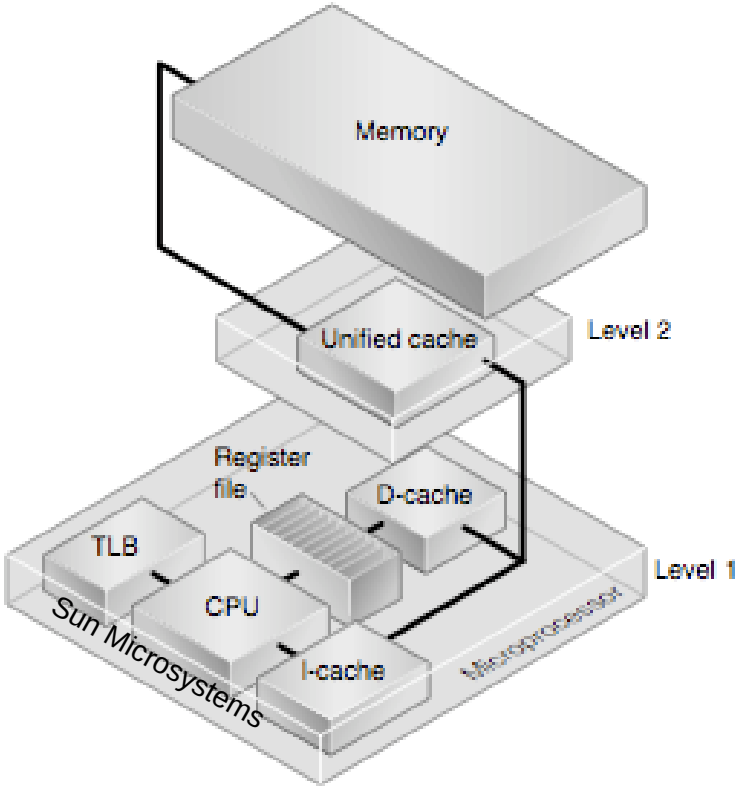


COMPUTATIONAL RESEARCH DIVISION

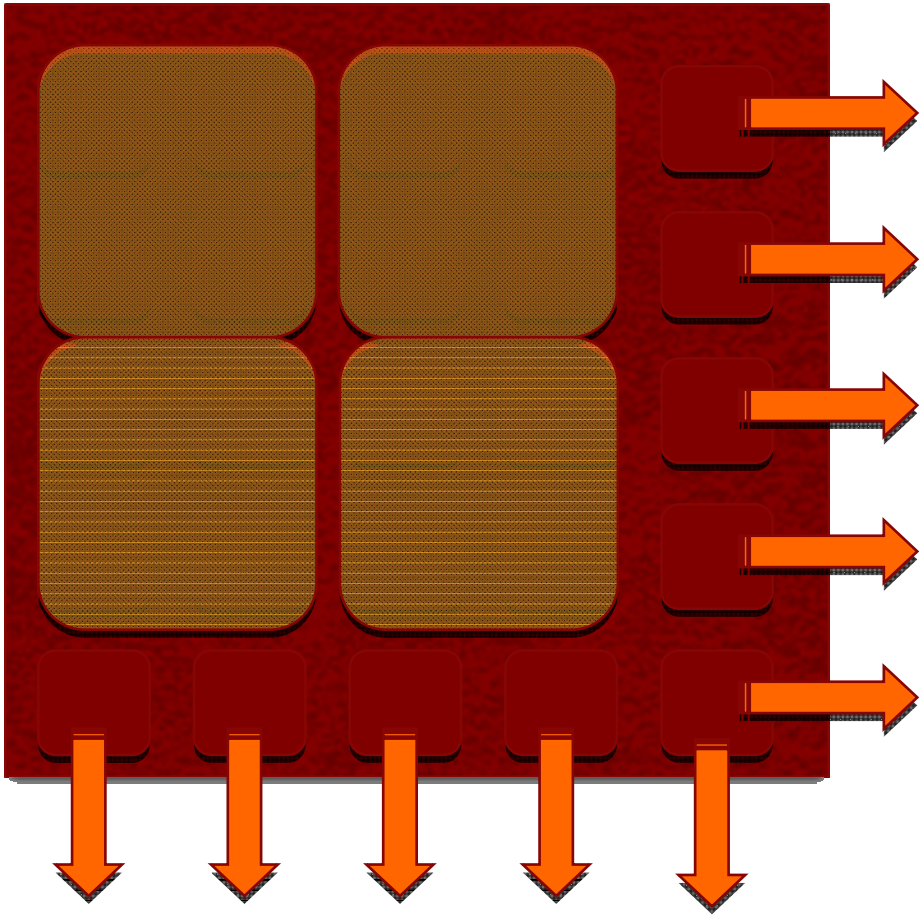
Response To Data Locality Challenge

Data Locality Management

Vertical Locality Management
(spatio-temporal optimization)



Horizontal Locality Management
(topology optimization)



Current Practices (2-level Parallelism)

NUMA Effects Ignored (with huge consequence)

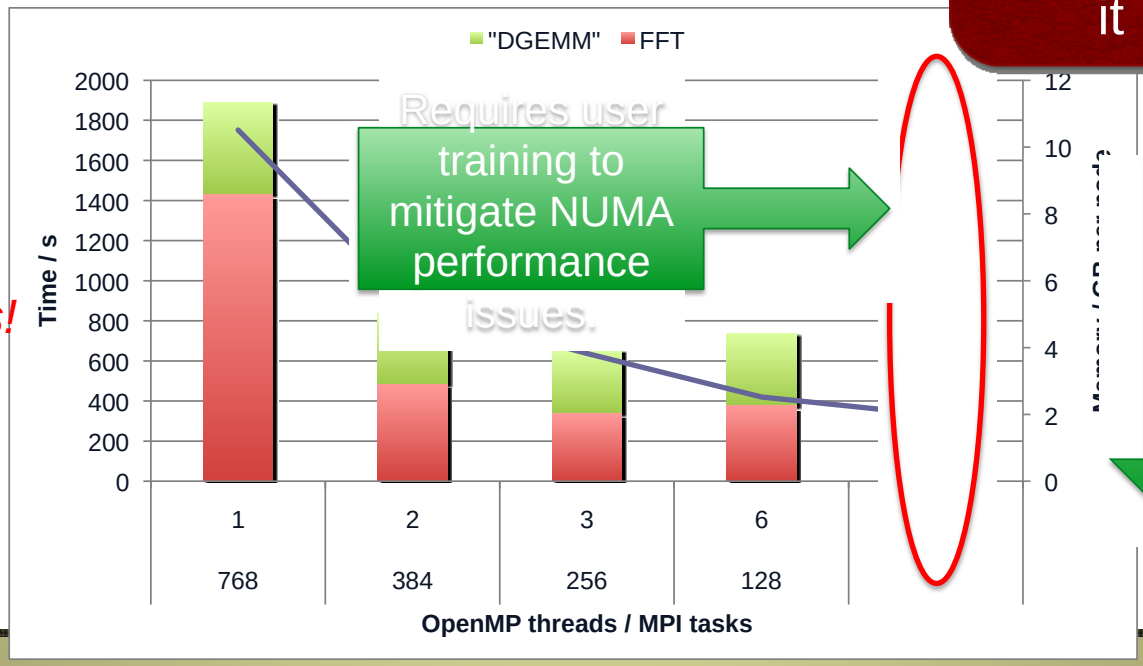
MPI+OMP Hybrid

- Reduces memory footprint
- Increases performance up to NUMA-node limit
- *Then programmer responsible for matching up compu layout!! (UGH!)*
- *Makes library writing difficult and Makes AMR nearly i*

It's the Revenge of the SGI Origin2000

We punted on the solution then, and we haven't solved it NOW!!

Bad News!



Expressing Hierarchical Layout

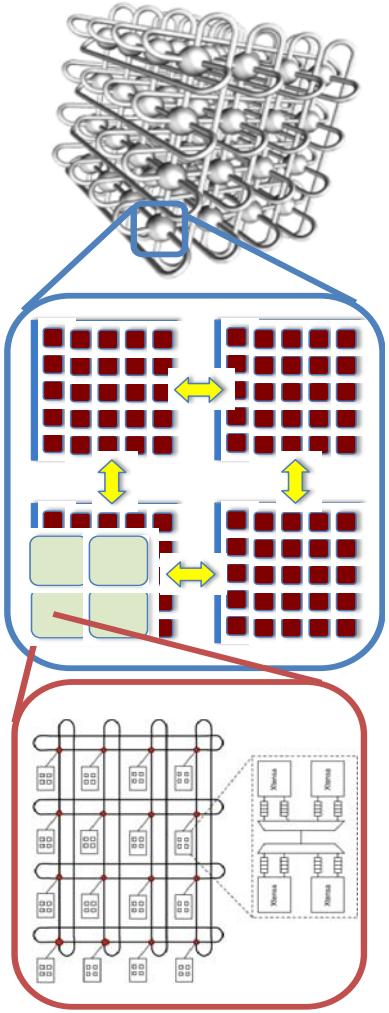
Old Model (OpenMP)

- Describe how to parallelize loop iterations
- Parallel “DO” divides loop iterations evenly among processors
- . . . but where is the data located?

New Model (Data-Centric)

- 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]);
```

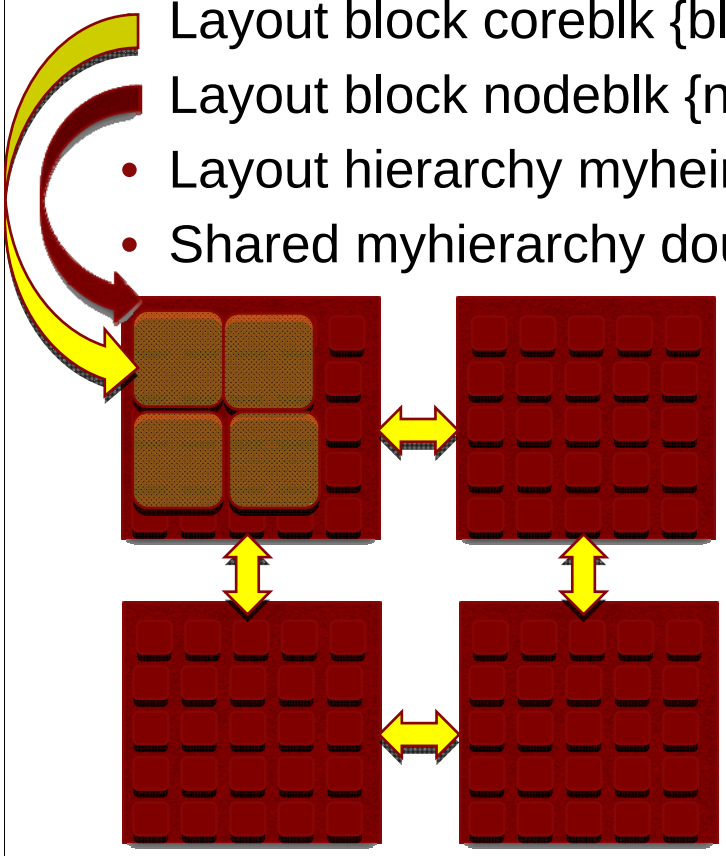


Data-Centric Programming Model

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

Building up a hierarchic

- Layout block coreblk {blockx
- Layout block nodeblk {nnx,n
- Layout hierarchy myheirarch
- Shared myhierarchy double



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

- **Then use data-localized parallel loop**
- **Foreach(TileCollection, Tile(a))**

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

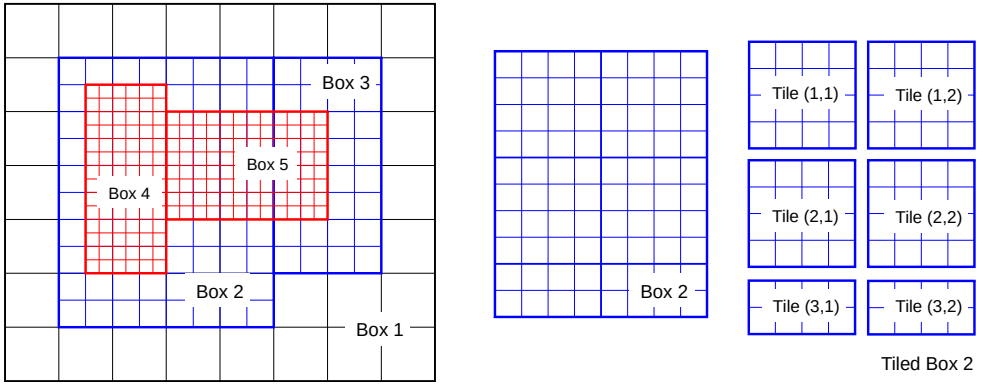
- ***And if layout changes, this loop remains the same***

Satisfies the request of the application developers (minimize the amount of code that changes)

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-grained parallelism: Asynchrony across tiles and across nodes
- 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!!!*

Abstraction for Memory Layout

Didem Unat
Dan Quinlan

Support different layouts for various cache coherence scenarios

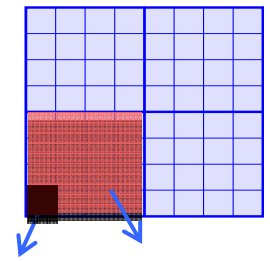
Require minimum code modification when the memory layout is changed

Memory layout options

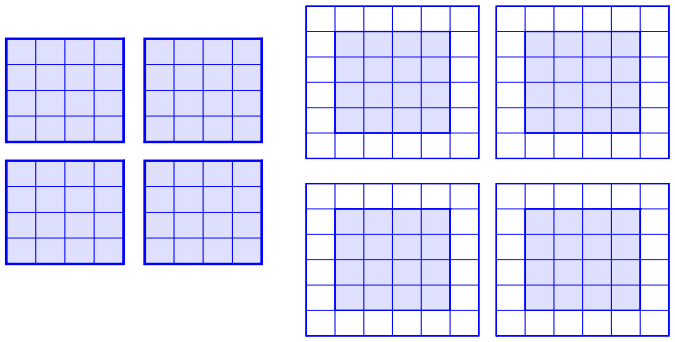
- Specified at the array construction thru a flag or
- export DATA_LAYOUT={LOG | SEP | REG}

The solvers remain unchanged !!!

a) Logical Tiles

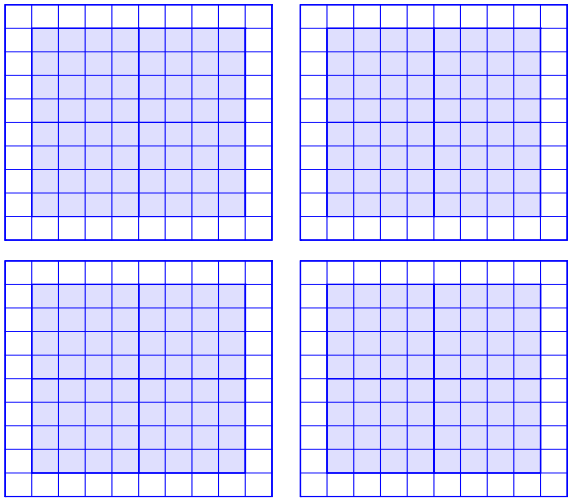


b) Separated Tiles



Separated tiles with halos

c) Regional Tiles



Lambdas for Loop Traversal
Decouple Loop Traversal from Loop Body

Why?

- Hides complicated loop traversal ordering behind the iterator interface
- Can change how the loop is parallelized
- Can add GPU acceleration under the hood
- Programmer does not need to implement them all

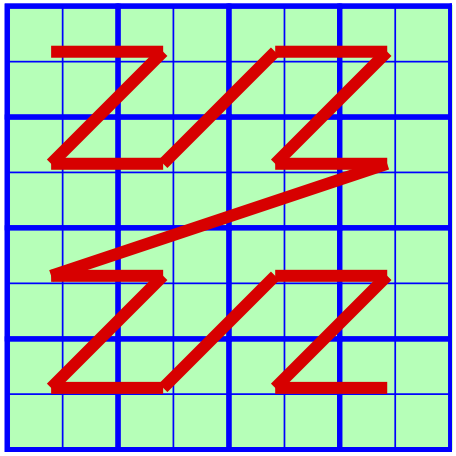
Introduce a language construct (such as doeach) to make it clean

Based off of CHAPEL iteration spaces but can use C++11 lambdas

```
doeach tl in tiledA

 ! Apply the following

end doeach
```



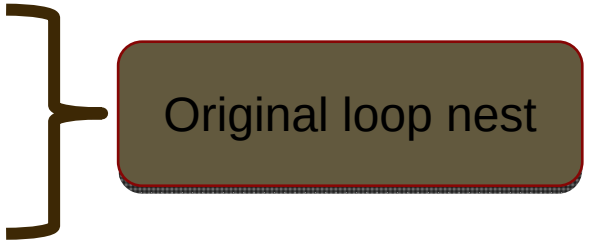
TiDA: Iterating over Tiles

Didem Unat (SC2013)

```
do j=lo(2), hi(2)
   do i=lo(1), hi(1)

      B(i,j)= A(i,j) ...

   end do
end do
```



Iterating over Tiles: Compiler Support

Tile traversal can be hidden behind an iterator or a loop construct

```
call TidaAlloc(tiledA,size,layout)
do tileno=1, ntiles (tiledA)
```

Looks the same on GPUs and on manycore CPUs *(OMP and OpenACC under the covers)*

```
  do j=lo(2), hi(2)
    do i=lo(1), hi(1)

      B(i,j)= A(i,j) ...

    end do
  end do
end do
```

Iterating over Tiles: Compiler Support

```
call TidaAlloc(tiledA,size,layout)
do tileno=1, ntiles (tiledA)

  tl = get_mtile(tiledA, tileno)
  lo = lwb(tl)
  hi = upb(tl)
  A => dataptr(tiledA, tileno)
  B => dataptr(tiledB, tileno)

  do j=lo(2), hi(2)
    do i=lo(1), hi(1)

      B(i,j)= A(i,j) ...

    end do
  end do
end do
```

Tile traversal can be hidden behind an iterator or a loop construct

With a compiler support, metadata retrieval can be hidden from the programmer

Kokkos: A C++ Templated Implementation (Sandia): same ideas, different package

Developed at Sandia

- Main target is molecular dynamics simulations & sparse linear algebra
- Array of Struct and Struct of Array support for CPU/GPUs
  - Layout changes are invisible to the user code

Uses C++ template meta-programming and operator overloading

Multidimensional Array

- Layout, Allocation, and Access parameters

  View <double **, **Layout**, **Device, RandomRead**> a[“a”, N, M];
- Accesses:

  a(i,j)
- Layout: row-major or column-major, can be extended for tiling

  parallel_for

  which takes a functor and iteration space as arguments

Programming Abstractions for Data Locality

Lugano Workshop, April 2014

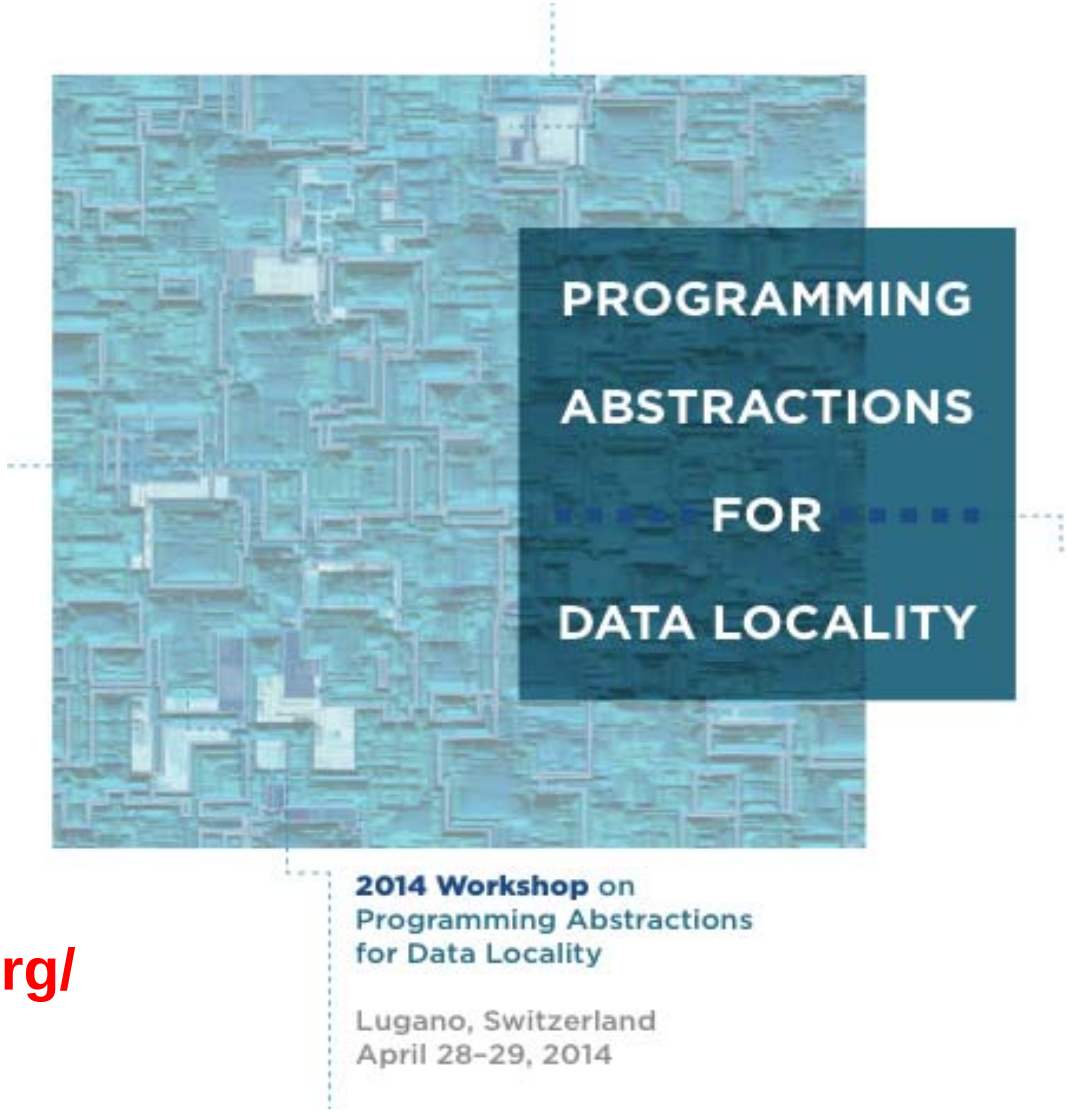
Organized a workshop at Lugano, Switzerland in April

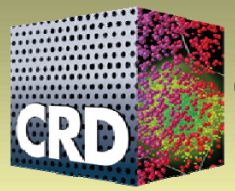
- To discuss emerging approaches
- Document common abstractions that appear in multiple implementations
- Opportunity to formalize this abstraction (reference standard)

PADAL report will be available in late August

•Co-released technical report with DOE, CSCS/ETH, INRIA, NSF, and others . . .

http://www.padalworkshop.org/





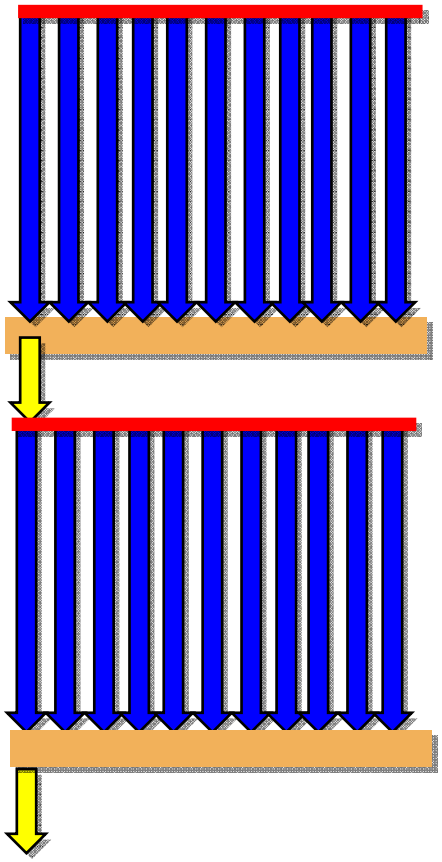
COMPUTATIONAL RESEARCH DIVISION

Heterogeneity / Inhomogeneity

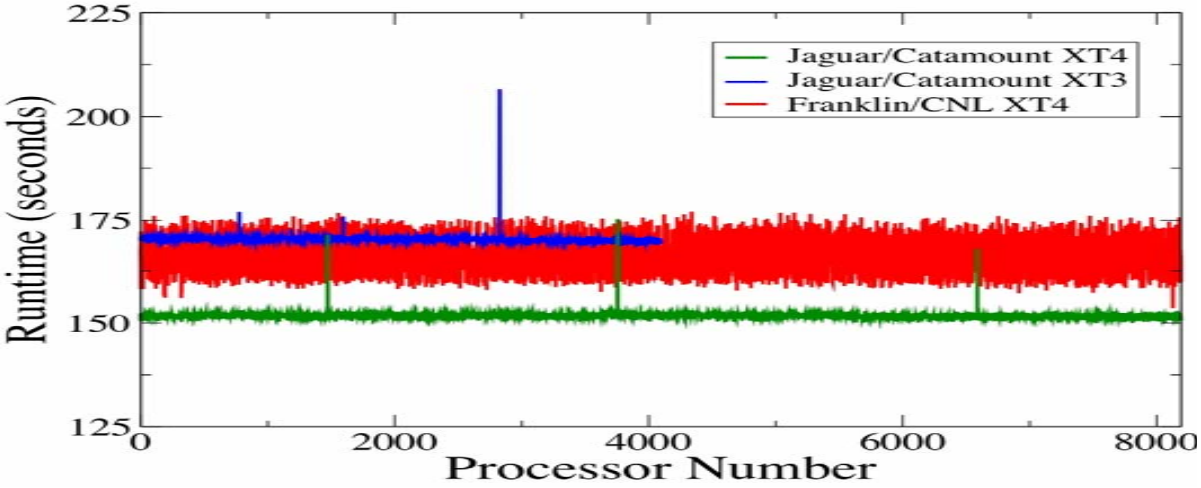
looking beyond BSP execution models

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

Bulk Synchronous Execution

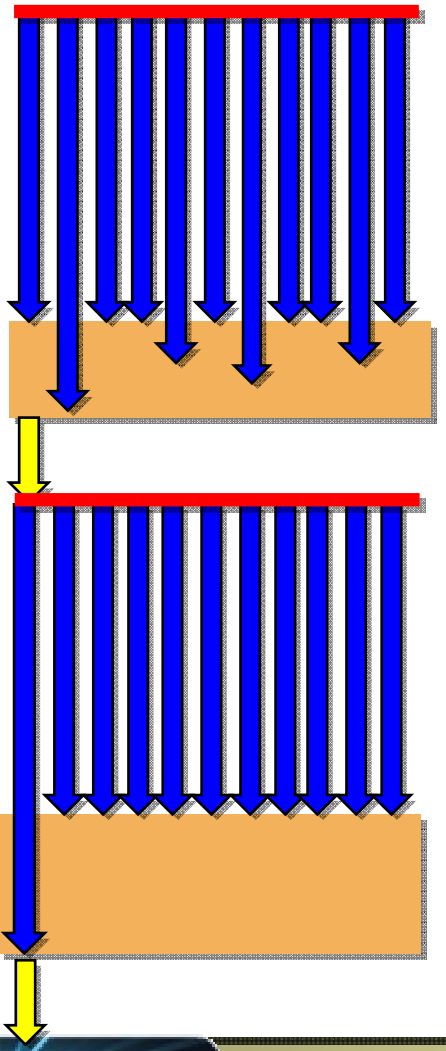


- **Heterogeneous compute engines (hybrid/GPU computing)**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  - *thermal throttling – no longer guarantee deterministic clock rate*
- **Nonuniformities in process technology creates non-uniform operating characteristics for cores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

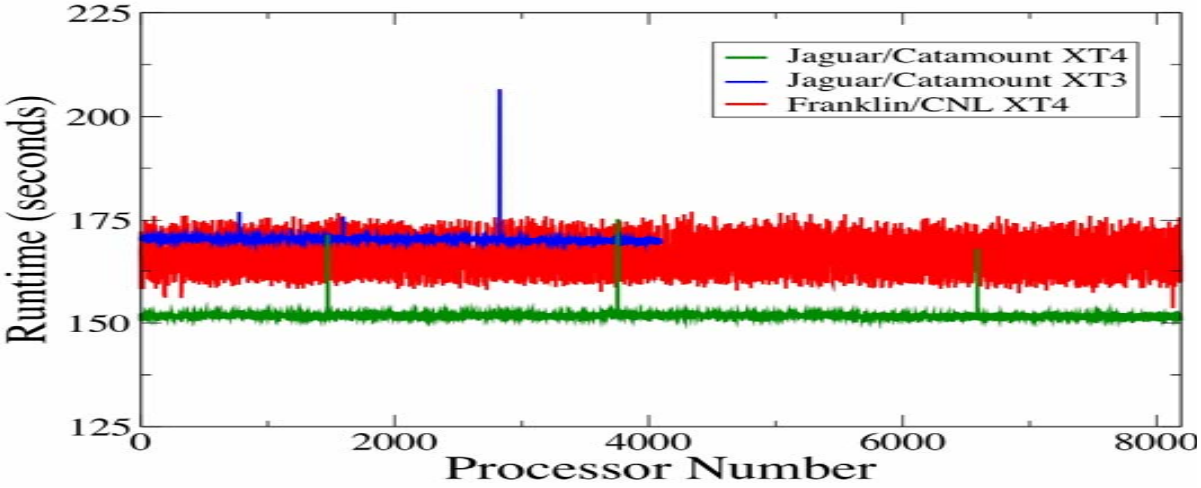


Assumptions of Uniformity is Breaking *(many new sources of heterogeneity)*

Bulk Synchronous Execution



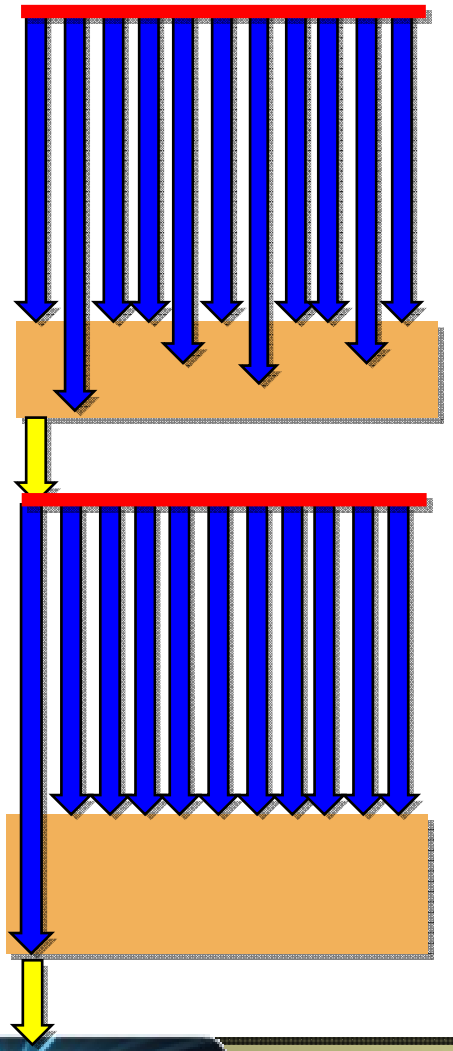
- **Heterogeneous compute engines (hybrid/GPU computing)**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  – *thermal throttling – no longer guarantee deterministic clock rate*
- **Nonuniformities in process technology creates non-uniform operating characteristics for cores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

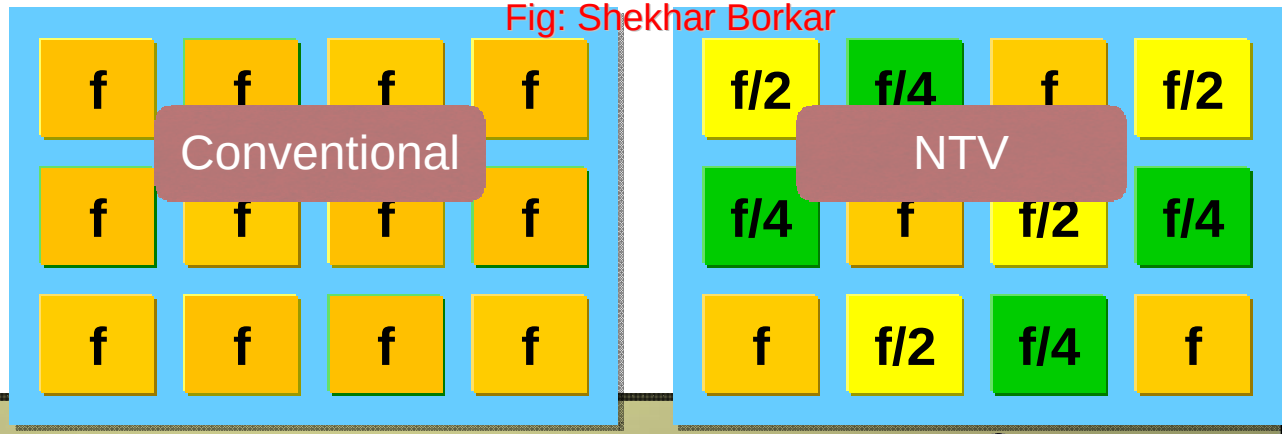
The really big opportunities for energy efficiency require codesign!

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*

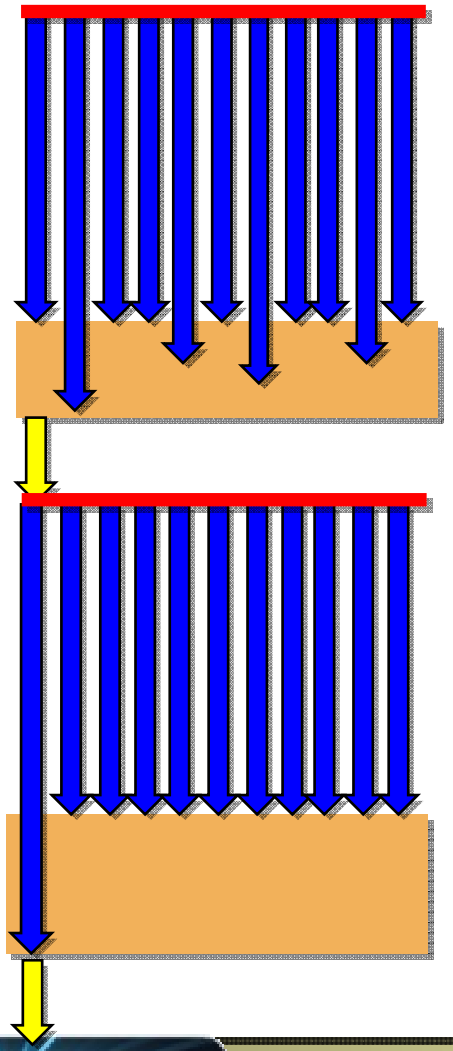
Fig: Shekhar Borkar



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

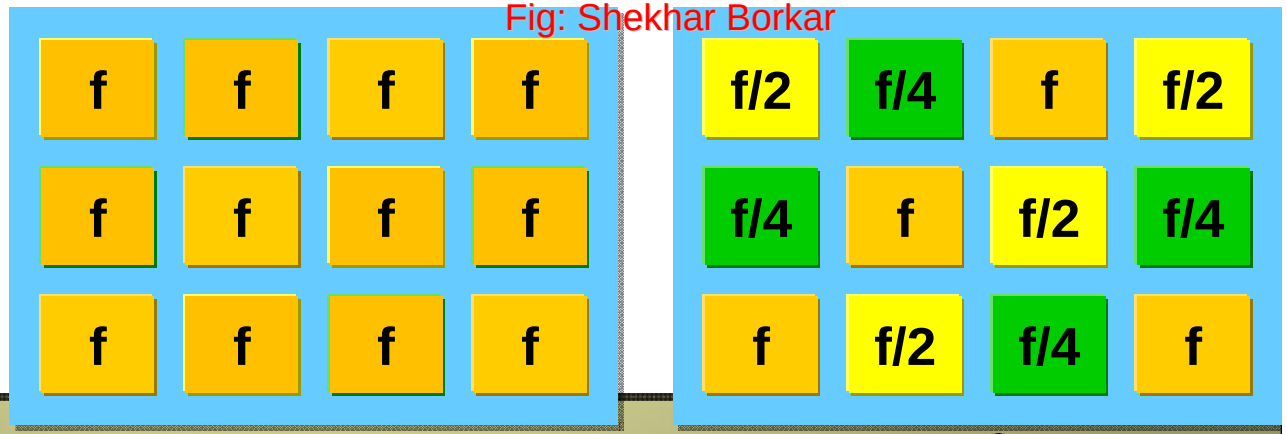
- Memory is faster, then odds are that the software will run faster
- if its better, that's good!

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)

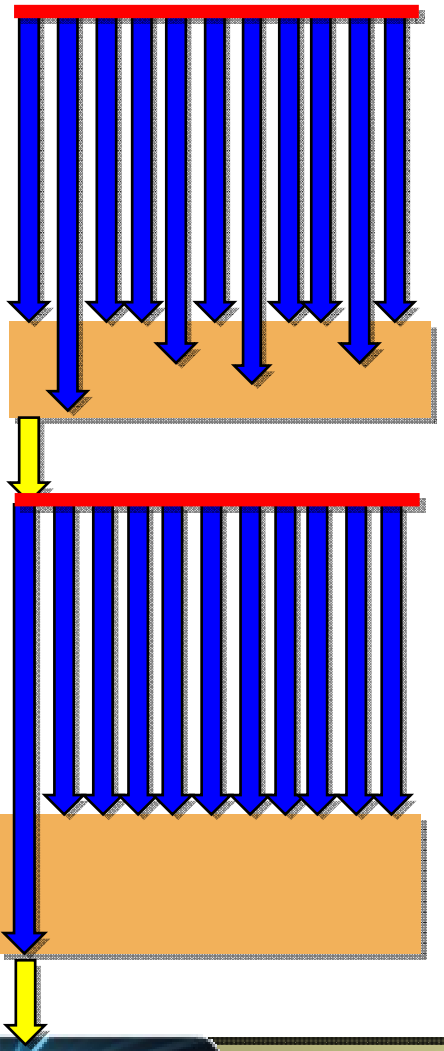
If software CANNOT exploit these radical hardware concepts (such as NTV), then it would be better to not have done anything at all!

Fig: Shekhar Borkar

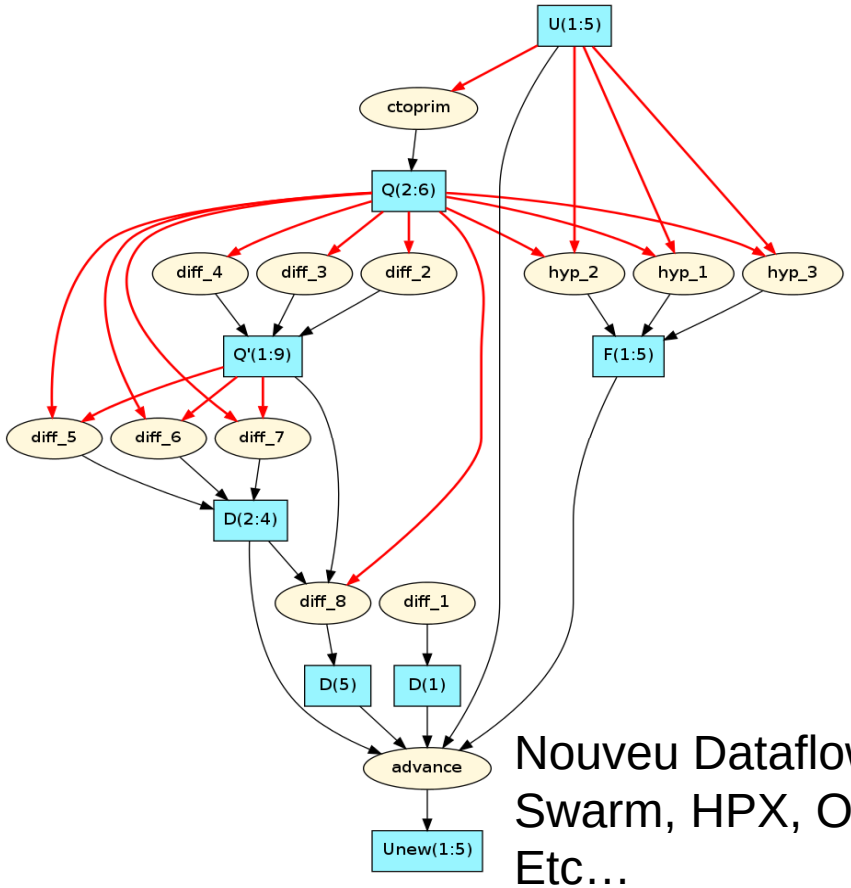


Assumptions of Uniformity is Breaking *(many new sources of heterogeneity)*

Bulk Synchronous Execution Model



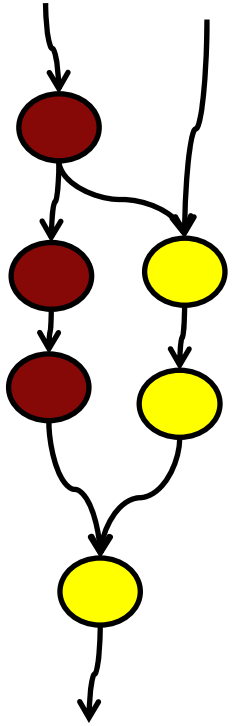
Asynchronous Execution Model



Nouveu Dataflow Swarm, HPX, OCR, Etc…

Resurgent Interest in Functional Semantics
(languages, coordination languages, or runtimes)

Its one of the only "safe" ways to program for this kind of execution model *(most other options lead to insanity)*



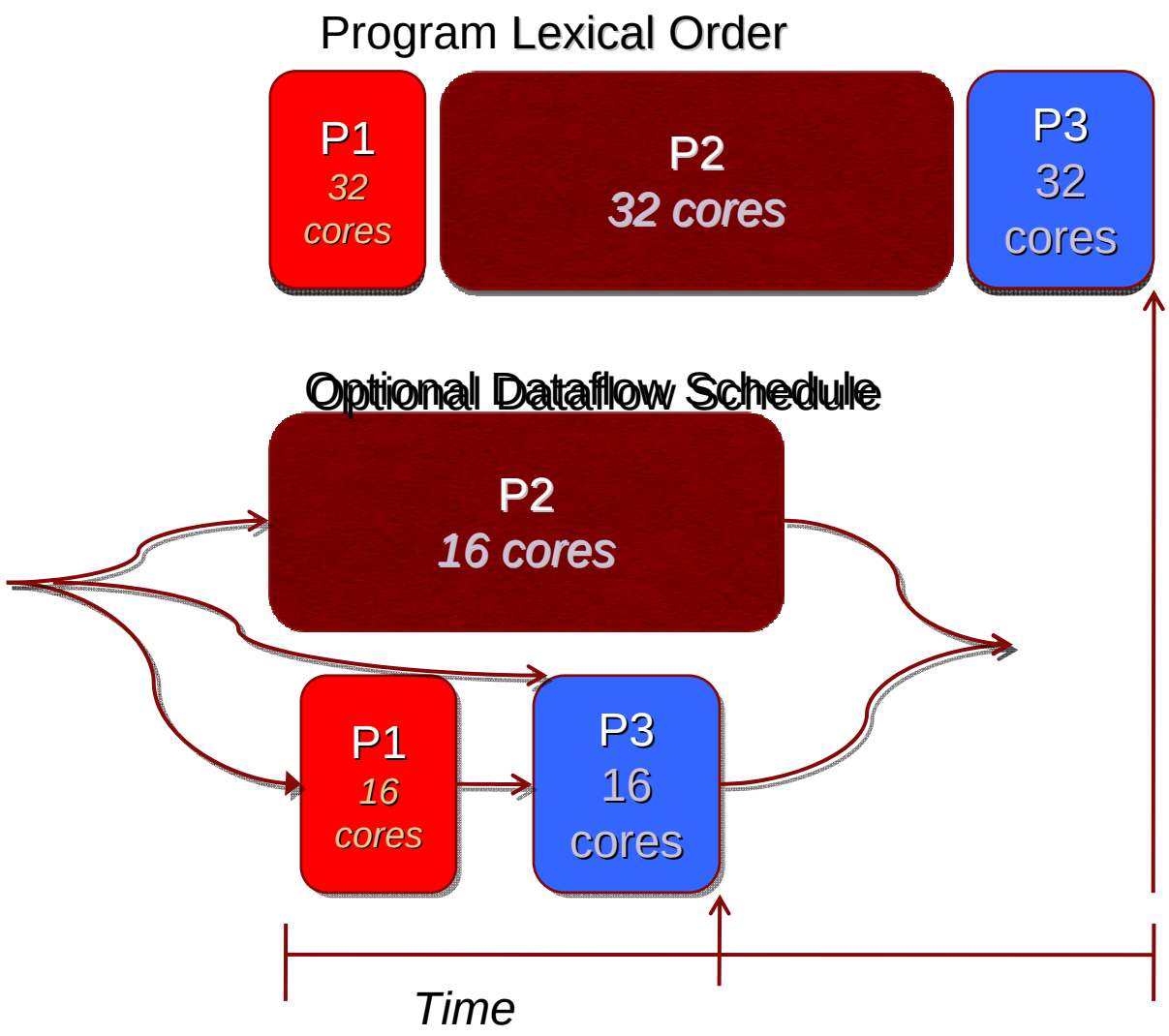
Requirements: *express computation declaratively*

- Stateless
- No side-effects
- Only operate on data you were handed

Benefits of "Isolation"

- Data dependence becomes statically analyzable
- Exposes implicit parallelism (DAG as constraint and runtime has a lot of freedom to control schedule)
- Trivial data migration or task migration (containment)
  - Local stores, accelerators and other disjoint memories are not a problem
- Know where data is needed OR when it is needed (but getting both is hard)

Functional Partitioning to Reduce Pressure on Domain Decomposition



Examples using TBB (functionally complete, but overheads high)

Hand-roll implementation Libraries to formalize

Schedule independent physics To Execute Concurrently

This is hard to do without functional semantics

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)

Conclusions

Emerging hardware constraints are increasingly mismatched with our current programming paradigm

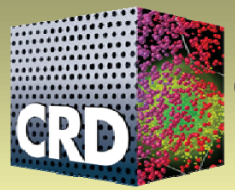
- Current emphasis is on preserving FLOPs
- The real costs now are not FLOPs… it is data movement
- Requires shift to a data-locality centric programming paradigm and hardware features to support it

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**



COMPUTATIONAL RESEARCH DIVISION

The End

For more information go to

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

Other Hardware Trends

Cloud/Datacenter NEEDS high performance internal fabric *(requirements are starting to align with HPC)*

Old Hardware Drivers for Clouds

- **COTS: Lowest cost off-the-shelf Ethernet gear** (HPC pushed towards high-performance fabrics for best TCO.)
- **External vs. Internal**: TCP/IP primarily to external network loads for web services (HPC primarily internally focused traffic patterns)
- **Throughput vs. Overhead**: Throughput valued more than low latency + overheads (HPC needed lower latency)
- **Overhead**s: Stacked VMs for elasticity and dynamic loads (but hurt HPC due to performance heterogeneity and overheads)
- **Contention**: Provision nodes for loosely coupled random traffic (tightly coupled jobs: provision contiguous topologies)

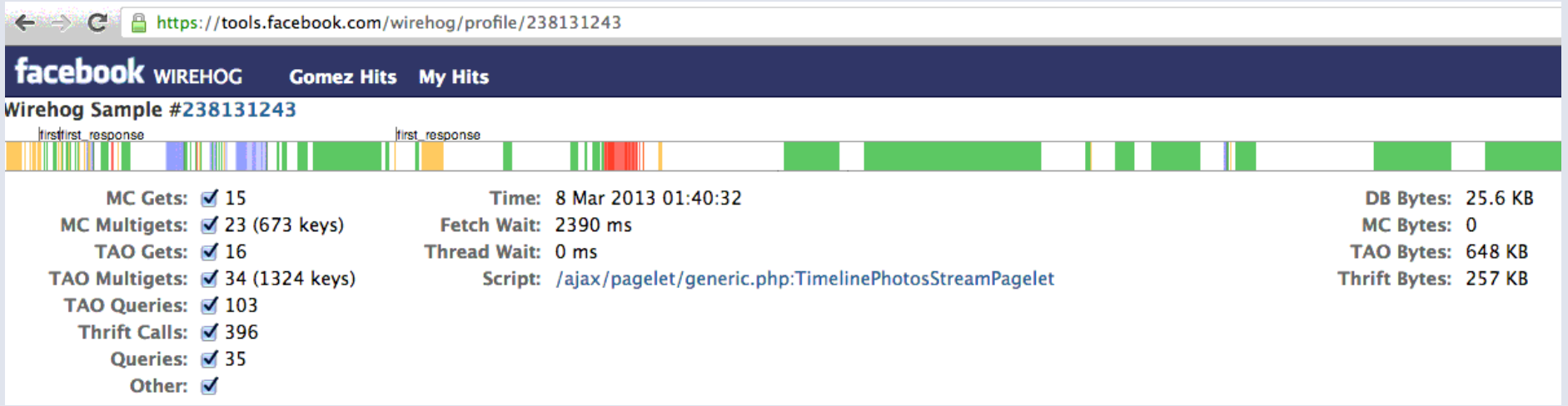
New Developments in Drivers for Cloud/Datacenter

- **Bikash Koley, Google Inc. (OI2012)**: **80%+** of Google traffic now internal facing *(used to be the other way around)*
- **Dennis Abts, Google Inc. (2011 book):** High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities
- **Nathan Farrington, Facebook (OI2013)**: Every **1kb** of external traffic entering the datacenter generates **930kb** of internal traffic.

Facebook seeing huge internal traffic requirements *(Nathan Farrington, Facebook Inc., Presented at OIC2013)*

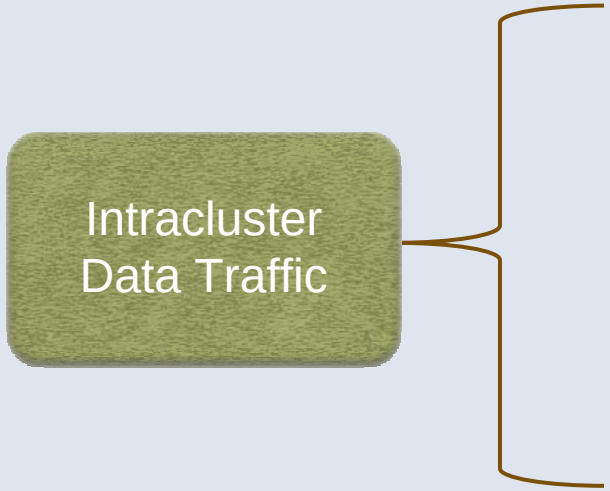
HTTP request amplification

This 1 KB HTTP request generated 930 KB of internal network traffic

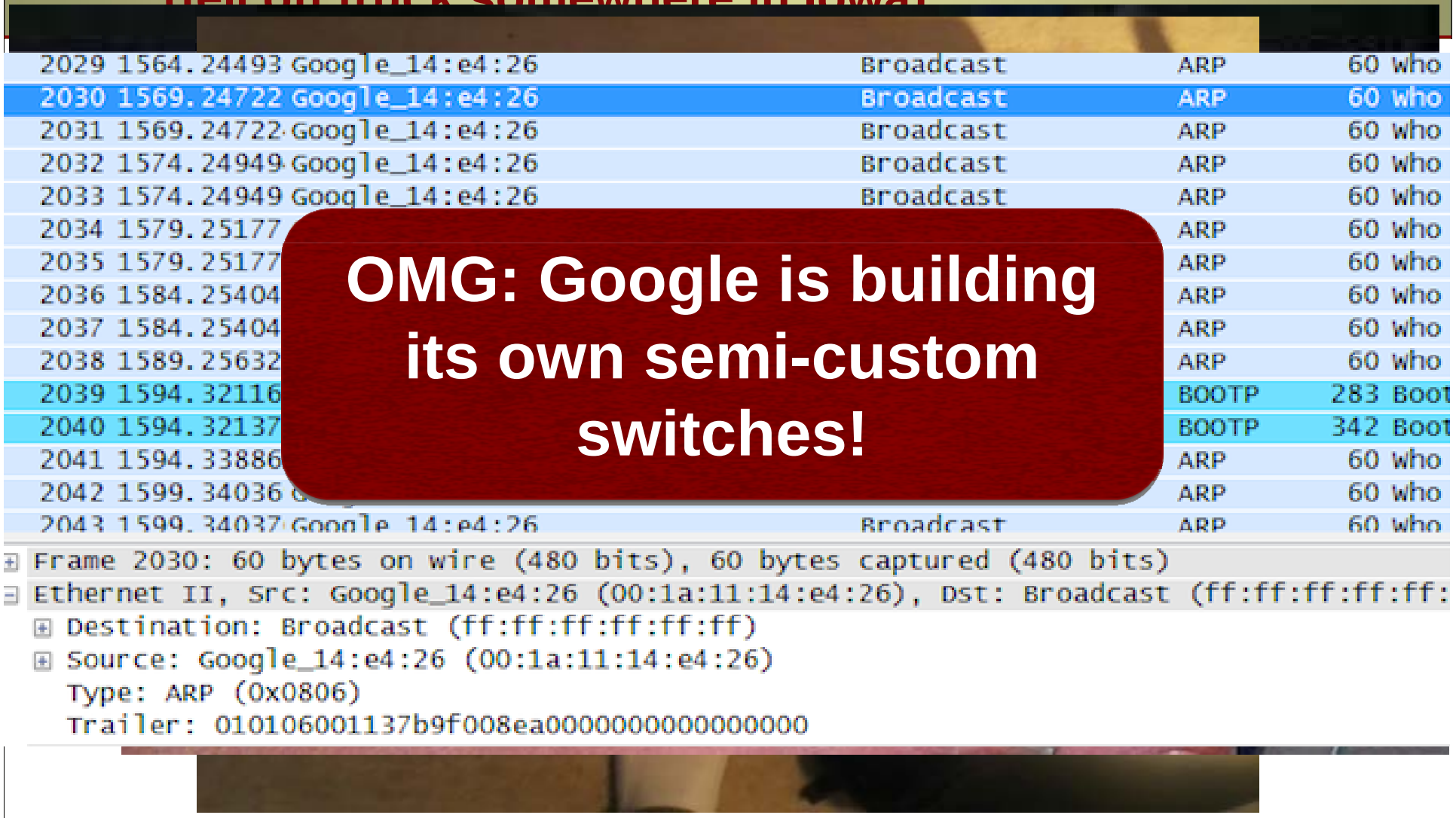


Majority of Facebook Traffic is Intra-Cluster *(Nathan Farrington, Facebook Inc., Presented at OIC2013)*

Egress traffic from one rack



Google "Pluto Switch"
(fell off truck somewhere in Iowa)



OMG: Google is building its own semi-custom switches!

Responses to New Cloud/Datacenter Requirements

Custom Intra-Center Fabrics *(Google not waiting !!)*

- Who the heck cares what you use for transport within the center?
- Meaner/Leaner communications software stack
- Modify switches to use more effective congestion control or avoidance
  - TCP/IP just uses inefficient lagging indicators of congestion or waits for packet drop + AIMD avoidance
- Optical Circuit Switches: why use packet switching for persistent flows? *(e.g. OpenFlow, but make it a hard circuit for QoS guarantees)*

System on Chip (SOC): Move NIC into CPU chip (silicon motherboard)

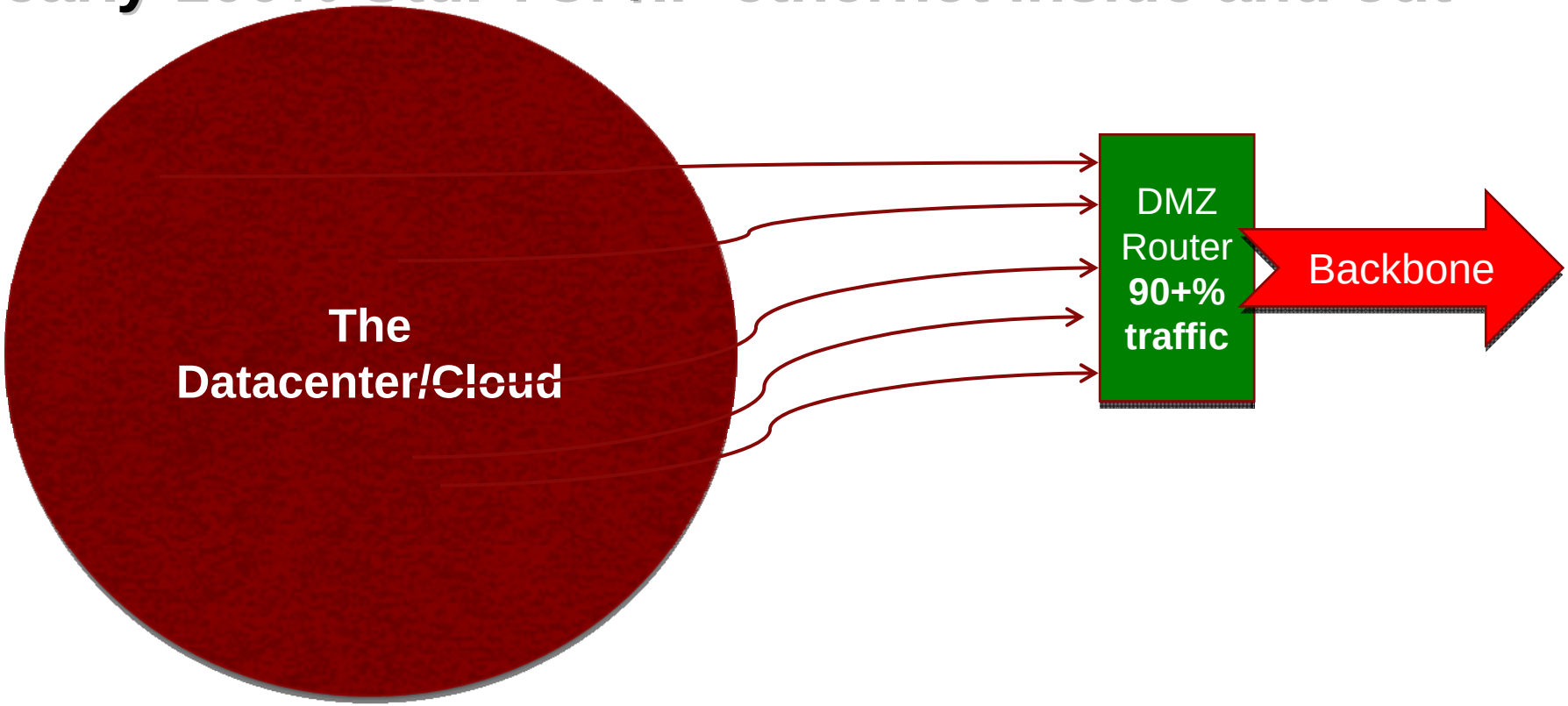
- **Use Moore's law** to put more *peripherals* onto chip instead of more *cores*
- **Reduces component count** *(reduces cost, size and complexity of motherboards)*
- **Reduces power** (*fewer off-chip connections)*
- **Reduces sources of failure** *(fewer solder joints and connectors… ask ORNL about that)*
- **Increases performance** *(factor of 20x reduction in software overheads)*

Old/New Conception of Cloud/Datacenters
(Simplified Conceptual Model)

Old Conception

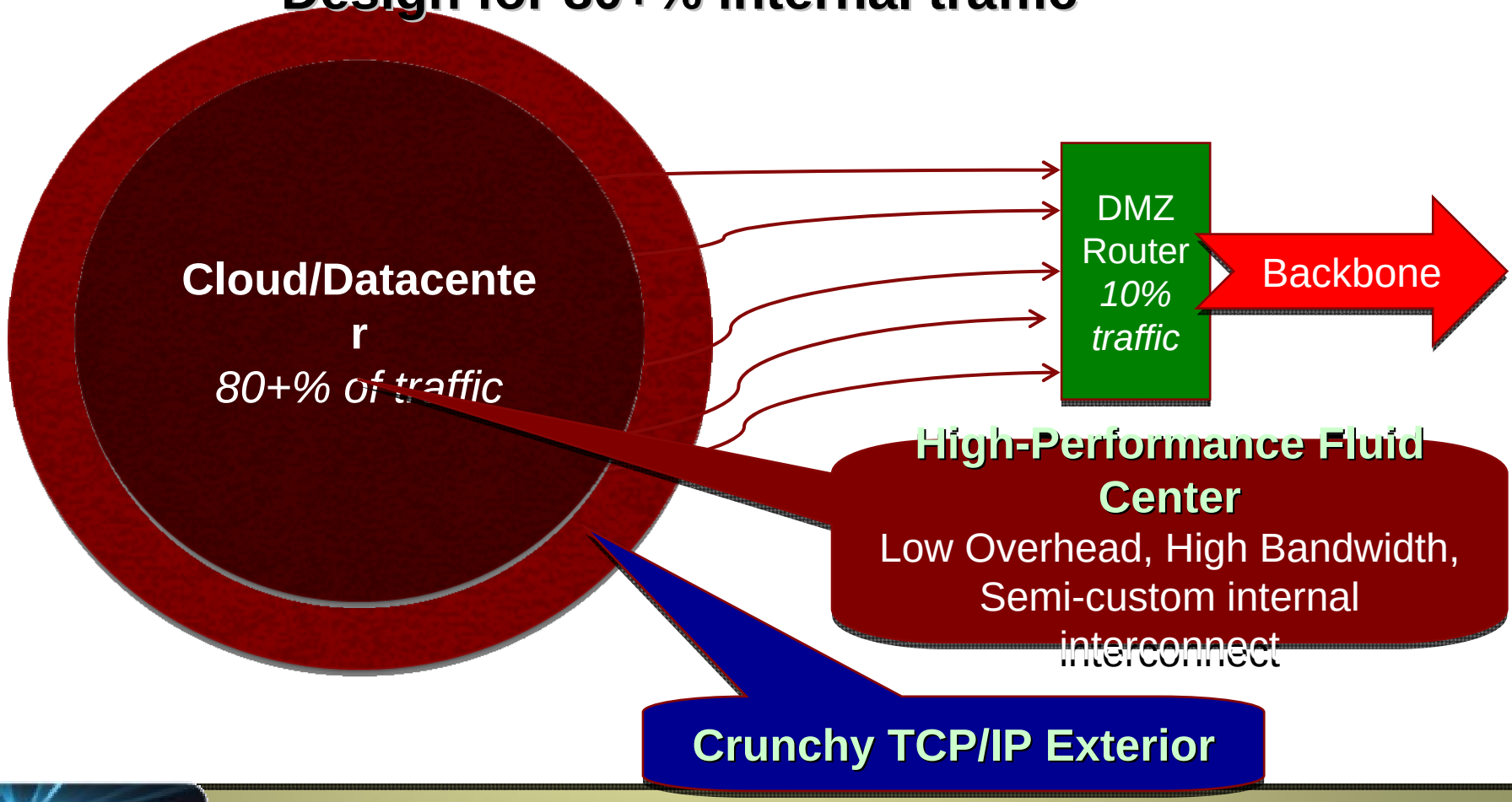
Designed for externally facing TCP/IP

Nearly 100% Std. TCP/IP ethernet inside and out

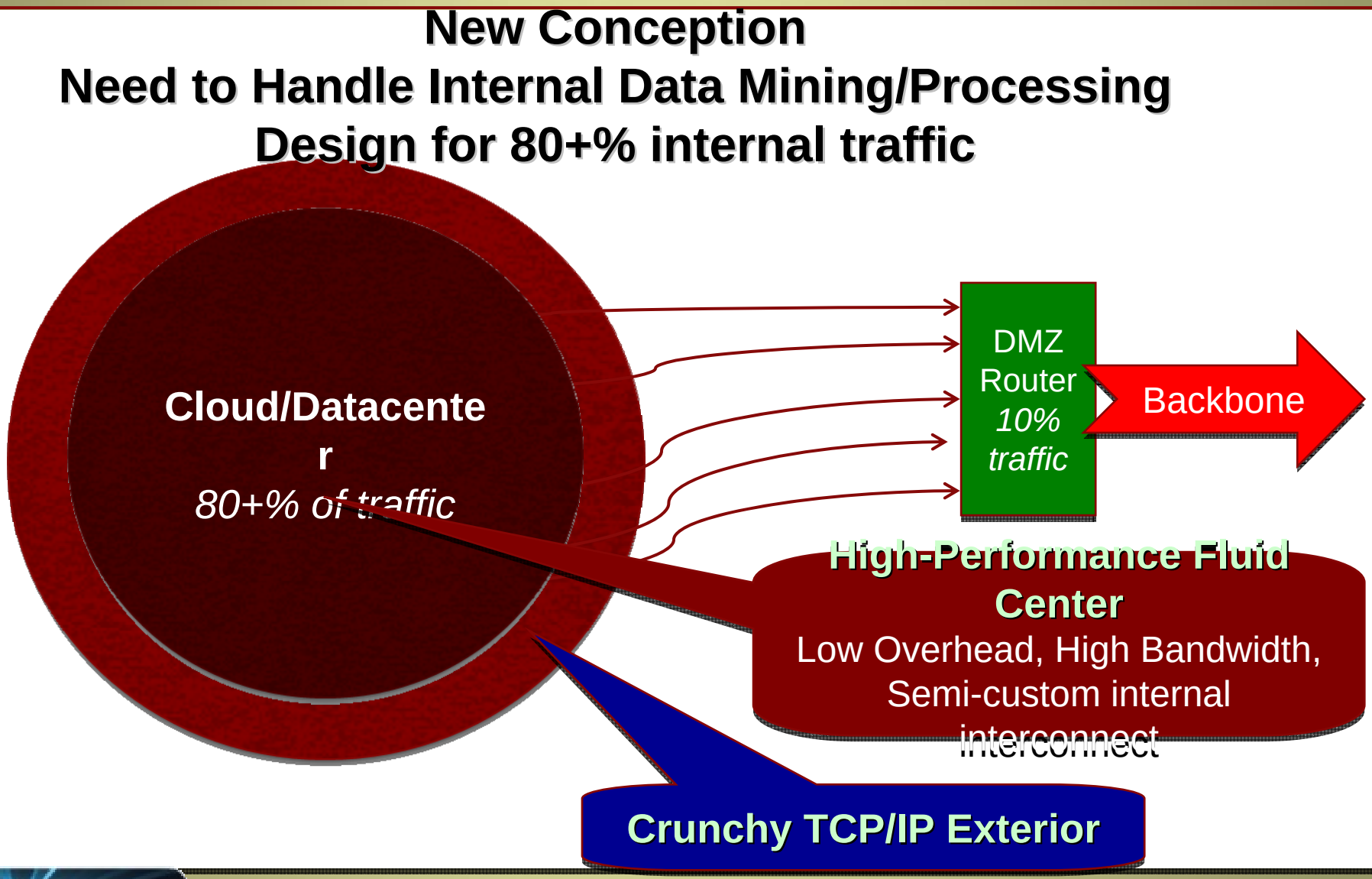


Old/New Conception of Cloud/Datacenters *(Simplified Conceptual Model)*

New Conception
Need to Handle Internal Data Mining/Processing
Design for 80+% internal traffic

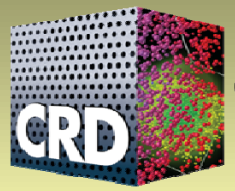


Looks Like Conceptual Diagram of a Typical HPC System



Convergence with HPC Requirements? (scorecard)

- **COTS: Lowest cost off-the-shelf Ethernet gear** (HPC pushed towards high-performance fabrics for best TCO.)
- **External vs. Internal**: TCP/IP primarily to external network loads for web services (HPC primarily internally focused traffic patterns)
- **Throughput vs. Overhead**: Throughput valued more than low latency + overheads (HPC needed lower latency)
- **Contention**: Provision nodes for loosely coupled random traffic (tightly coupled jobs: provision contiguous topologies)
- **Performance Variation**: Dynamic behavior for elasticity and cost (but hurt HPC due to performance heterogeneity and overheads)
- **Resilience**: Loosely coupled jobs, depend on software to tolerate failure (HPC tightly coupled parallelism depends on HW to avoid failures… software not very tolerant of faults)



COMPUTATIONAL RESEARCH DIVISION

System on Chip

A NEW strategy for making use of Commodity Technology

Technology Investment Trends

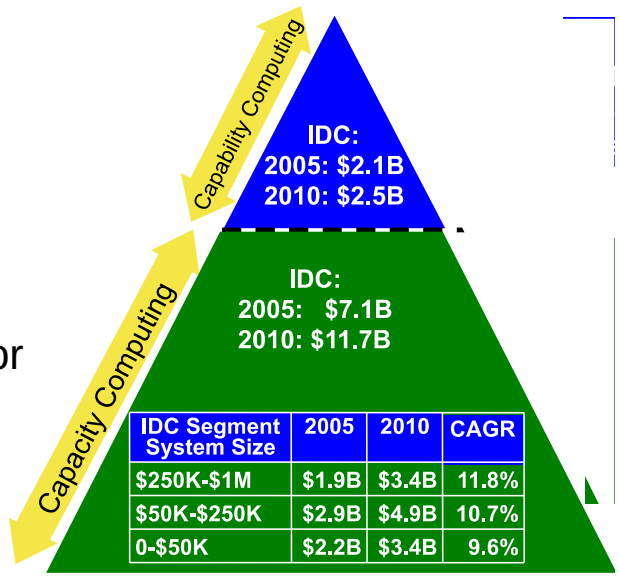
Image below From Tsugio Makimoto: ISC2006

1990s - R&D computing hardware dominated by desktop/COTS

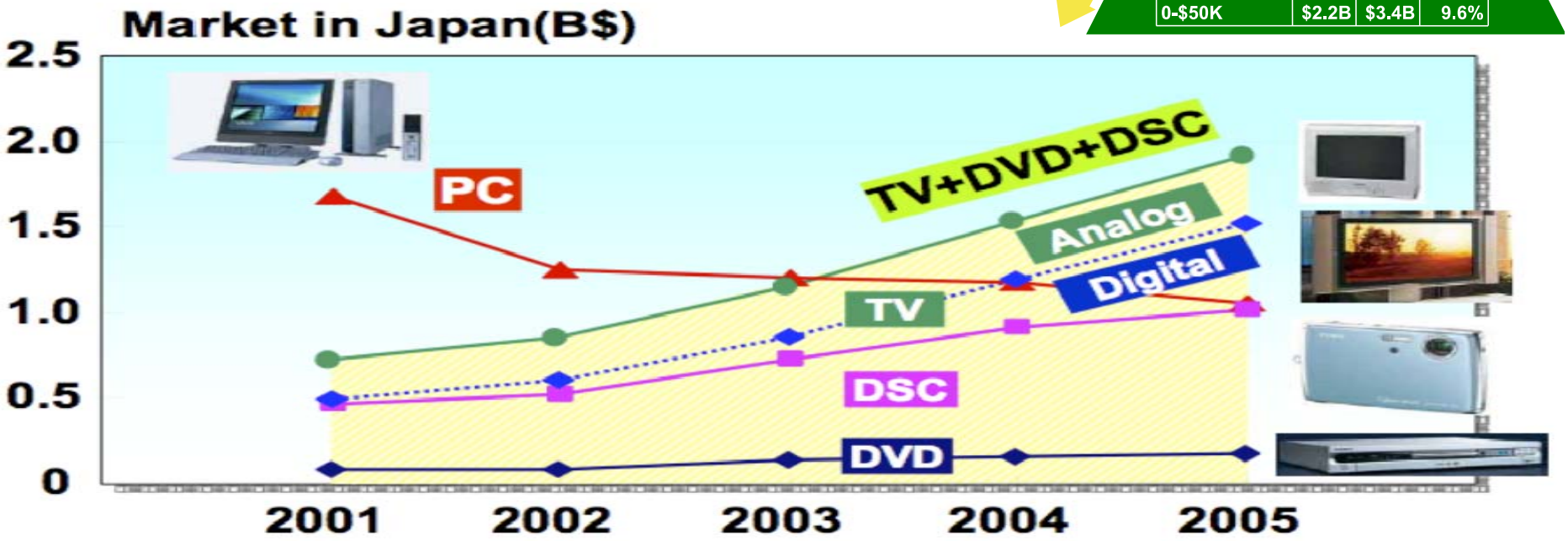
- Had to learn how to use COTS technology for HPC
- Thomas Sterling's "Beowulf Cluster"

2010 - R&D investments moving rapidly to consumer electronics/ embedded processing

- Must learn how to leverage embedded/consumer processor technology for future HPC systems
- Think "Beowulf chip"



IDC Segment System Size	2005	2010	CAGR
$250K-$1M	$1.9B	$3.4B	11.8%
$50K-$250K	$2.9B	$4.9B	10.7%
0-$50K	$2.2B	$3.4B	9.6%



Building an SoC from IP Logic Blocks

Its legos with a some extra integration and verification cost
*(Bill Dally's "shopping List") **(anonymized price quotes)***

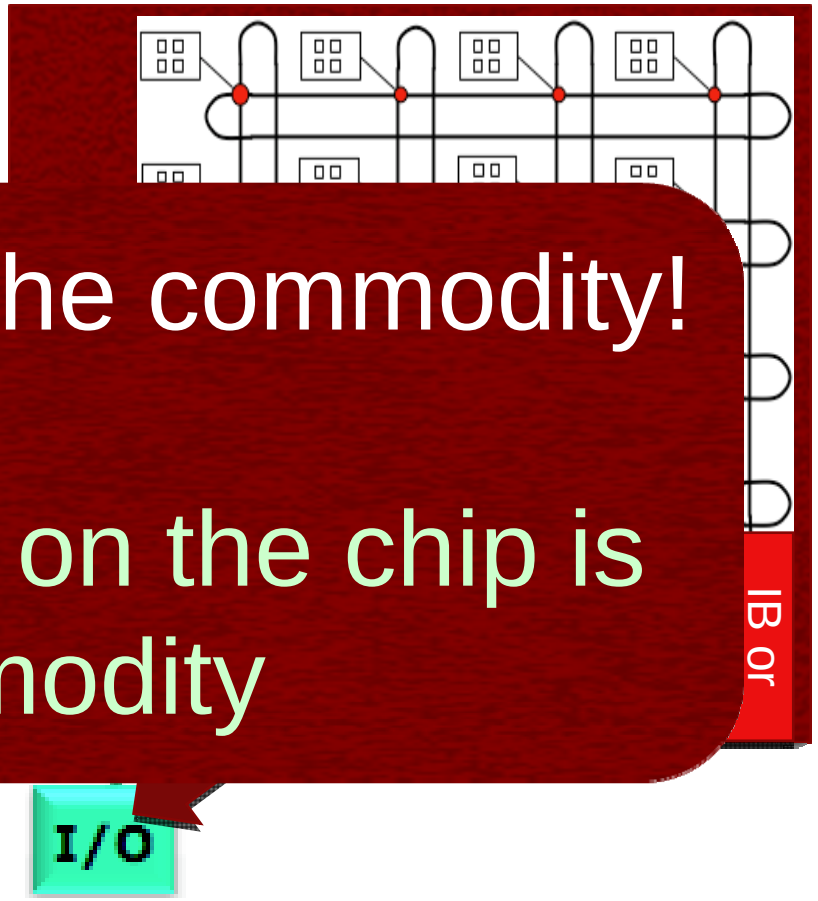
Processor Core (ARM, Tensilica, MIPS deriv)
With extra "options" like DP FPU, ECC
IP license cost $150k-$500k

NoC F
IP

DDR3
(Dena
+ Phy
IP

PCIe G
IP License: $250k

Integrated FLASH Controller
IP License: $150k



The Chip is NOT the commodity!

The stuff you put on the chip is the commodity

10GigE or IB DDR 4x Channel
IP License: $150k-$250k

Berkeley Sumpercomputer Predicts Your Doom

Written by whatsrequired

0 tweet Share 0 Digg digg

Photo:

Image from Blatch

The University of California at Berkeley is rolling out a new breed of supercomputer, specially designed to predict the challenges presented by climate change, ultimately leading humanity to our doom and the computers to their rightful place as the masters of our earthly domain.

The idea driving the claim that supercomputers can be revolutionized is the radical notion

Search!

with permission by photographer Lane Hartwell

conteurs Performing Carolina Dram

About

Spidey Senses is written by Ted Rheingold, a passionate thirty-something living in San Francisco. He's started and runs both the biggest dog info, care and community site and cat info and community site (aka Dogster and Catster =) and posts articles about online communities and business development at the Dogster, Inc. company blog.

Recent Mini-Updates

- Now that's a perfect fit! Big congrats @dshen http://ds.ly/IKNVnR Help more great things grow. about 2 hours ago
- Sleeping lamb, smiling monkey http://plixi.com/p/96517546 about 3 hours ago
- @sarahkunst are u on instagram on path? Been posting more there. Mabel was conceived just before or after (oops) the 4 of us had drinks ;) about 3 hours ago
- @Aloisius happy birthday! And invite me when u do! 1 day ago

Firefox Extension

- Who Is This Person? Research a person by searching their name against relevant websites.

Recent Site Readers

You!

e purpose

mputers

a research

signing a

f

OpenSoC: Abstract Fabric

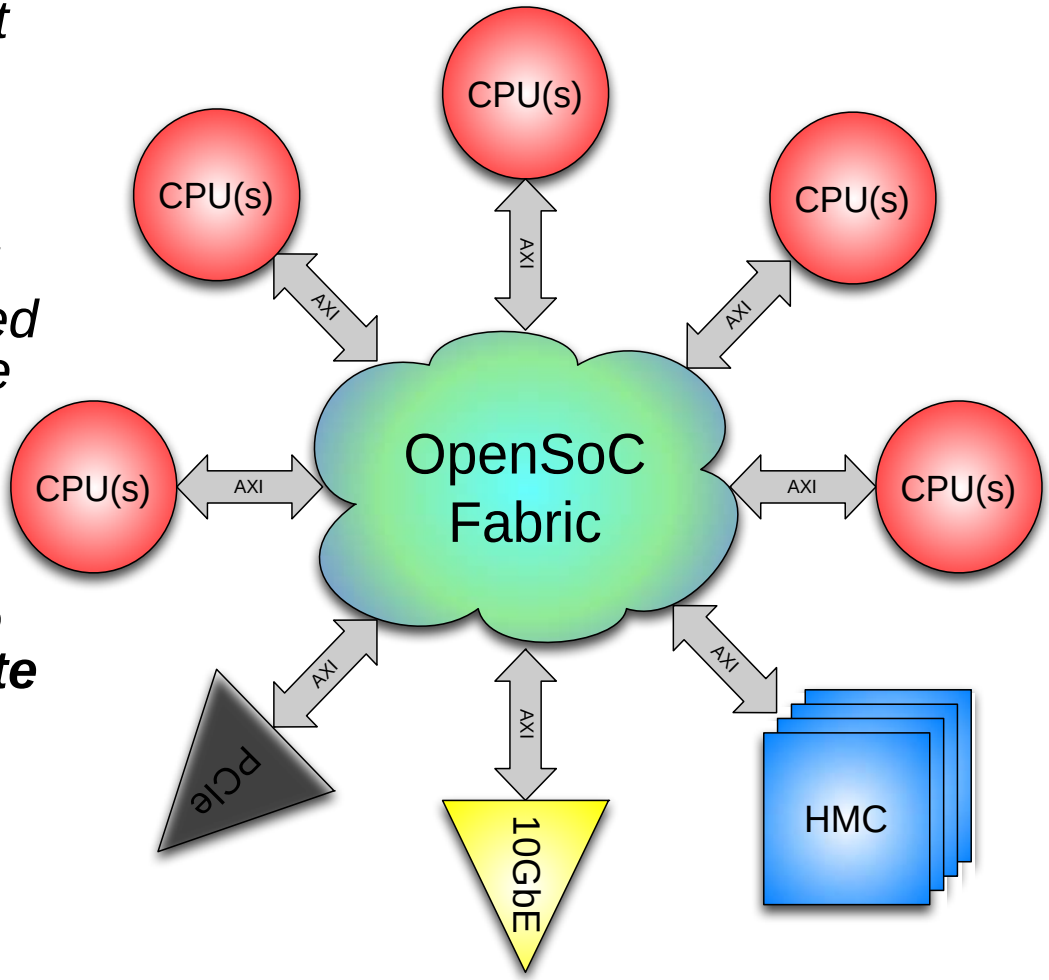
System-on-Chip (SoC) could revolutionize energy efficient computing

Seymour Cray 1977: *"Don't put anything in to a supercomputer that isn't necessary."*

Mark Horowitz 2007*: "Years of research in low-power embedded computing have shown only one design technique to reduce power: reduce waste."*

SoC Revolution enables us to achieve goal of reducing waste

- *Enable us to include ONLY what we need for HPC.*
- *Tighter component integration*
- *Fewer losses for inter-chip wiring for peripherals*



http://www.opensocfabric.org/

Conclusions

Emerging hardware constraints are increasingly mismatched with our current programming paradigm

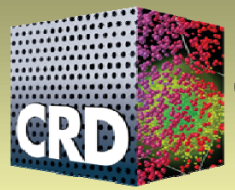
- Current emphasis is on preserving FLOPs
- The real costs now are not FLOPs… it is data movement
- Requires shift to a data-locality centric programming paradigm and hardware features to support it

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**



COMPUTATIONAL RESEARCH DIVISION

The End

For more information go to

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

Bonus:

Data layout (currently, make it more expressive)

- Need to support hierarchical data layout that closer matches architecture
- Automated method to select optimal layout is elusive, but type-system can support minimally invasive user selection of layout

Horizontal locality management (virtualize)

- Flexibly use message queues and global address space
- Give intelligent runtime tools to dynamically compute cost of data movement

Vertical data locality management (make more expressive)

- Need good abstraction for software managed memory
- Malleable memories (allow us to sit on fence while awaiting good abstraction)

Heterogeneity (virtualize)

- Its going to be there whether you want it or not
- Pushes us towards async model for computation (post-SPMD)

Parallelism (virtualize)

- Need abstraction to virtualize # processors (but must be cognizant of layout)
- For synchronous model (or sections of code) locality-aware iterators or loops enable implicit binding of work to local data.
- For async codes, need to go to functional model to get implicit parallelism
  - Helps with scheduling
  - Does not solve data layout problem

Summary

- There is progress in Exascale with many projects now focused and on their way, e.g. FastForward, Xstack, and Co-Design Centers in the U.S.
- HPC has moved to low power processing, and the processor growth curves in energy-efficiency could get us in the range of exascale feasibility
- Memory and data movement are still more open challenges
- Programming model needs to address heterogeneous, massive parallel environment, as well as data locality
- Exascale applications will be challenge just because their sheer size and the memory limitations

DOE Strategy for Exascale Computing

Designing the computing environment for the future

Objective: *Enable DOE scientists and engineers to use the most advanced computational hardware and software for discovery science.*

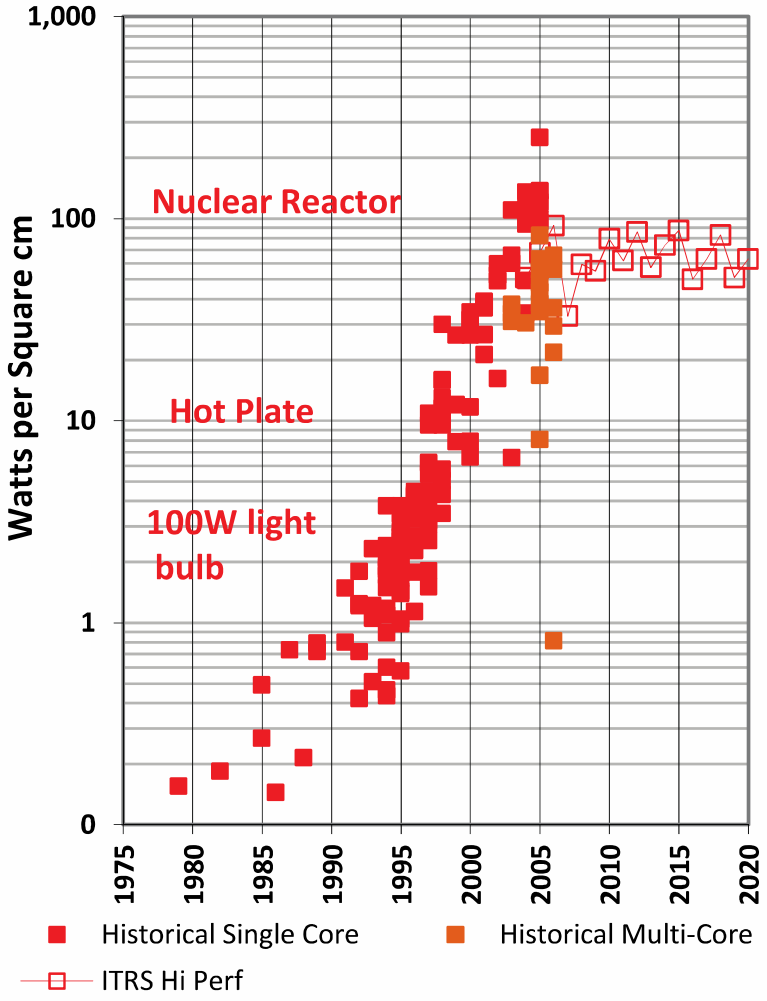
The Challenge of our Decade: *Performance growth in fixed power budget*

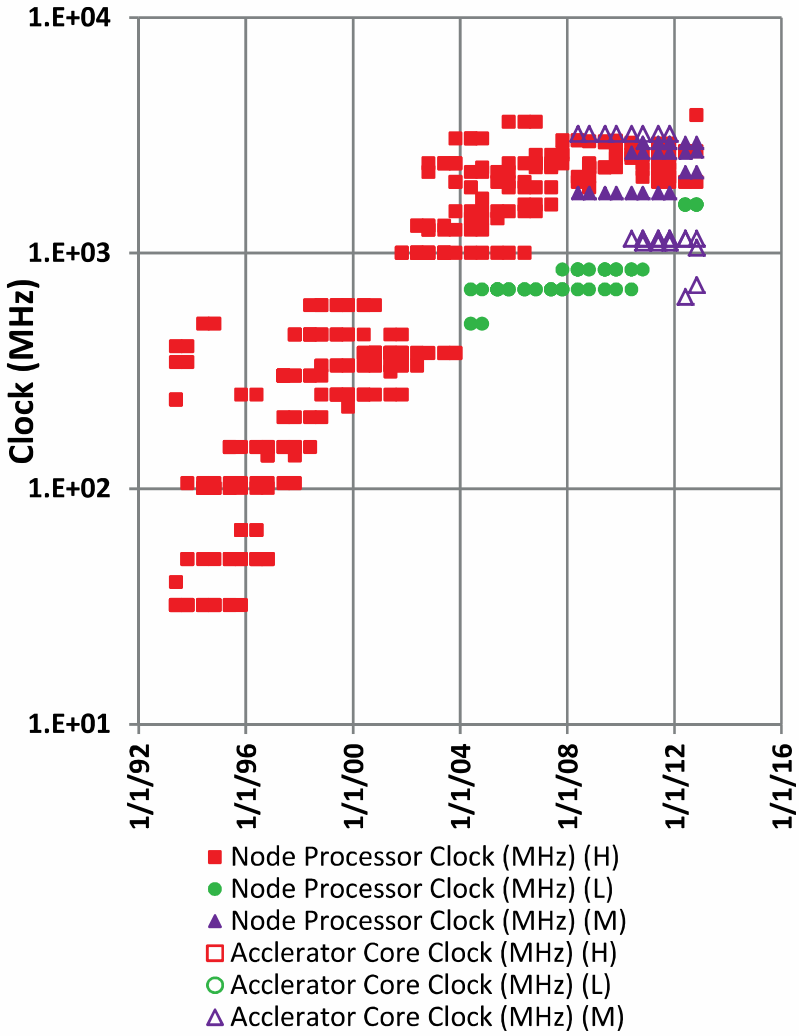
- The challenge is as dramatic as transition from vector to MPP
- This transition affects *all* computing for science from smallest to the largest scale
- Fundamentally breaks our software infrastructure (need to re-architect)

Approach: *Components of CoDesign Process*

- **XStack**: Translate emerging architectural trends into advanced software technology *(operating systems, communications libraries, programming systems)*
- **Fast Forward:** *$60M* public/private partnerships to accelerate development of computing technologies to deliver 100x more *usable* operations per watt in 10 yrs
- **CoDesign Centers**: Software Design Space Exploration, “proxy applications” and application prototyping to facilitate codesign
- **Hardware Design Space Exploration**: CAL hardware design space and “proxy hardware” using architectural simulation and modeling to facilitate codesign

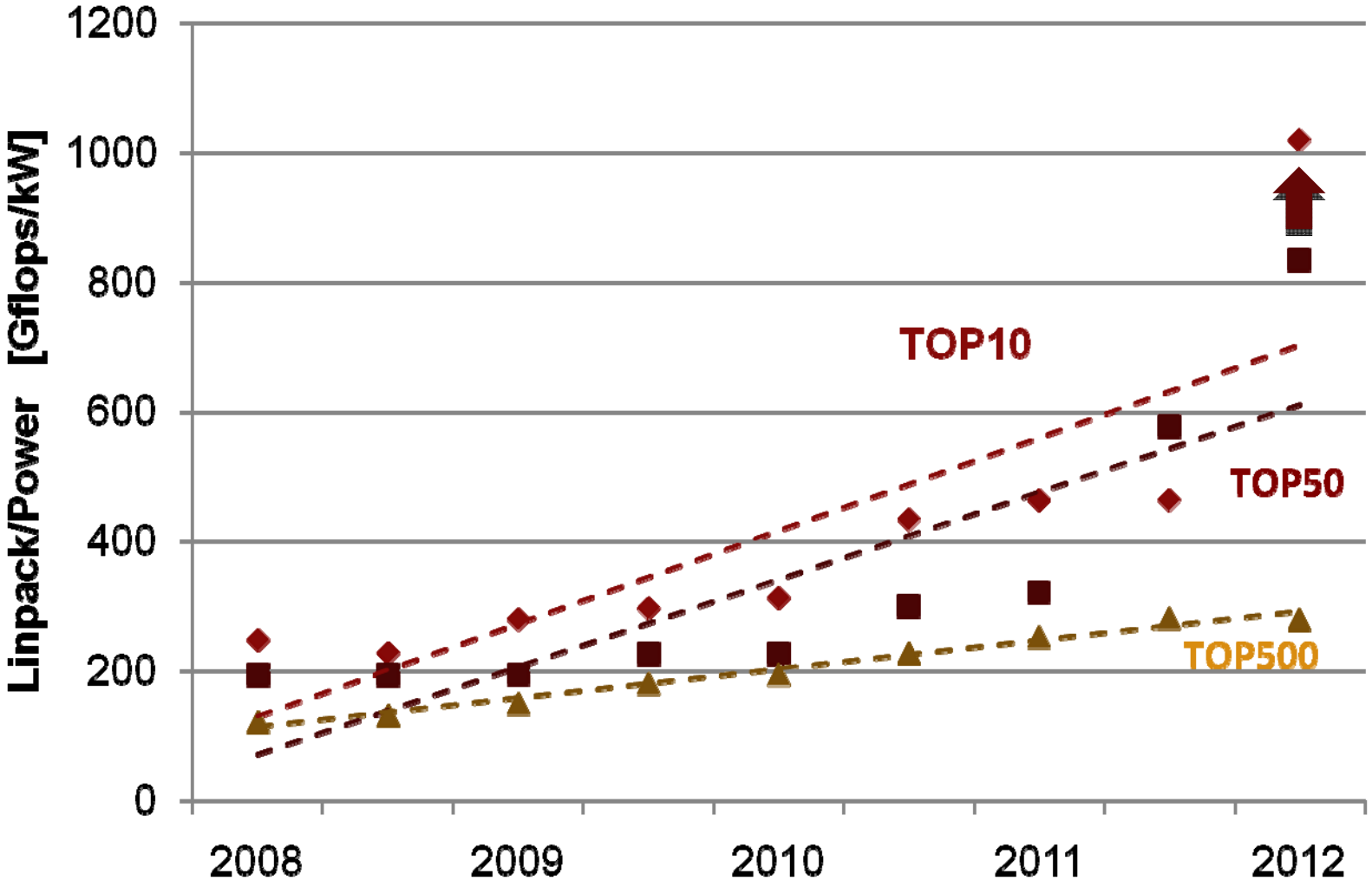
The Power and Clock Inflection Point in 2004





Source: Kogge and Shalf, IEEE CISE

Power Efficiency has gone up significantly in 2012



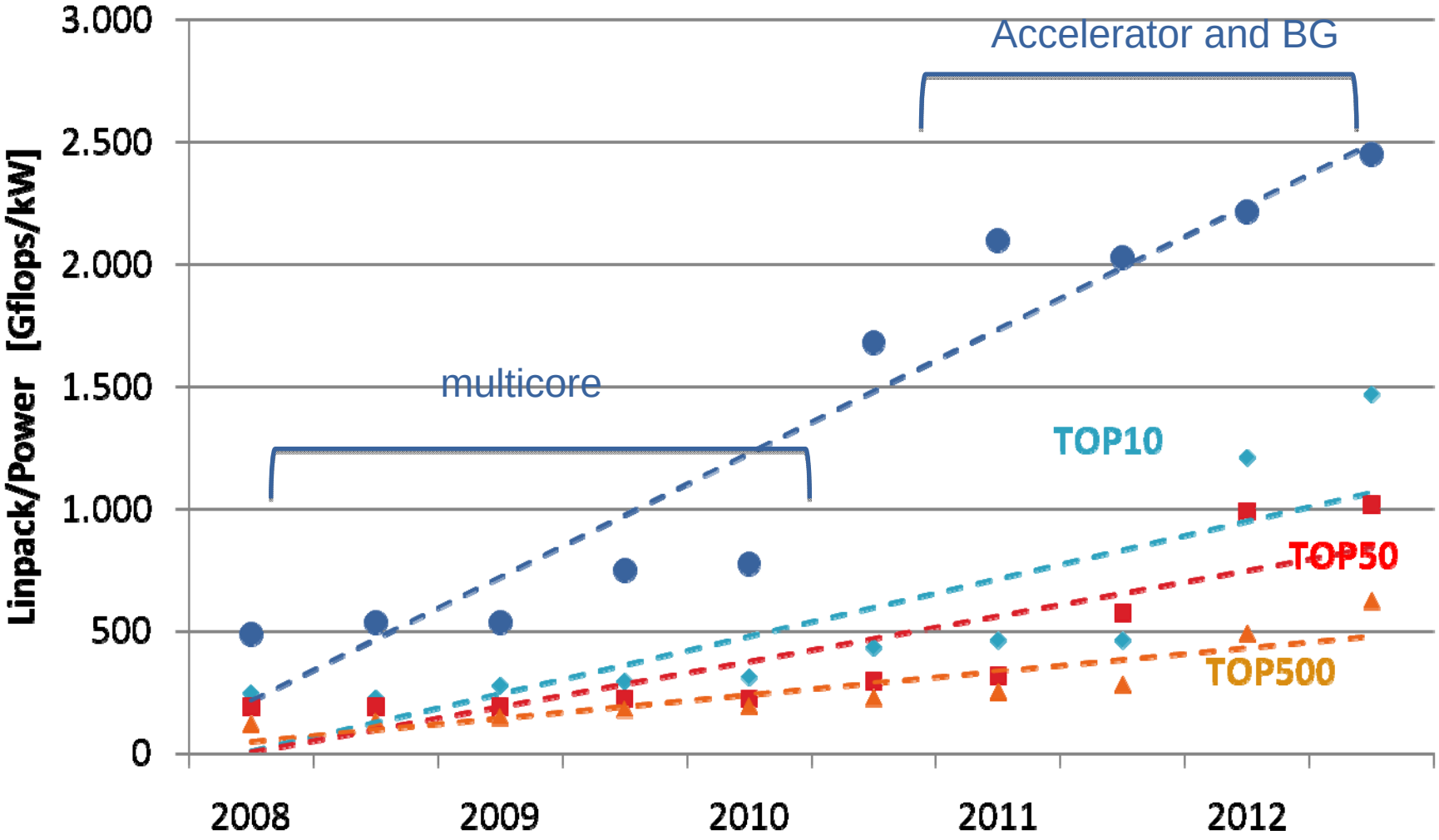
Data from: TOP500 November 2012

Most Power Efficient Architectures

Computer	Rmax/ Power
Appro GreenBlade, Xeon 8C 2.6GHz, Infiniband FDR, **Intel Xeon Phi**	**2,450**
Cray XK7, Opteron 16C 2.1GHz, Gemini, **NVIDIA Kepler**	**2,243**
BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	**2,102**
iDataPlex DX360M4, Xeon 8C 2.6GHz, Infiniband QDR, **Intel Xeon Phi**	**1,935**
RSC Tornado, Xeon 8C 2.9GHz, Infiniband FDR, **Intel Xeon Phi**	**1,687**
SGI Rackable, Xeon 8C 2.6GHz, Infiniband FDR, **Intel Xeon Phi**	**1,613**
Chundoong Cluster, Xeon 8C 2GHz, Infiniband QDR, **AMD Radeon HD**	**1,467**
Bullx B505, Xeon 6C 2.53GHz, Infiniband QDR, **NVIDIA 2090**	**1,266**
Intel Cluster, Xeon 8C 2.6GHz, Infiniband FDR, **Intel Xeon Phi**	**1,265**
Xtreme-X , Xeon 8C 2.6GHz, Infiniband QDR, **NVIDIA 2090**	**1,050**

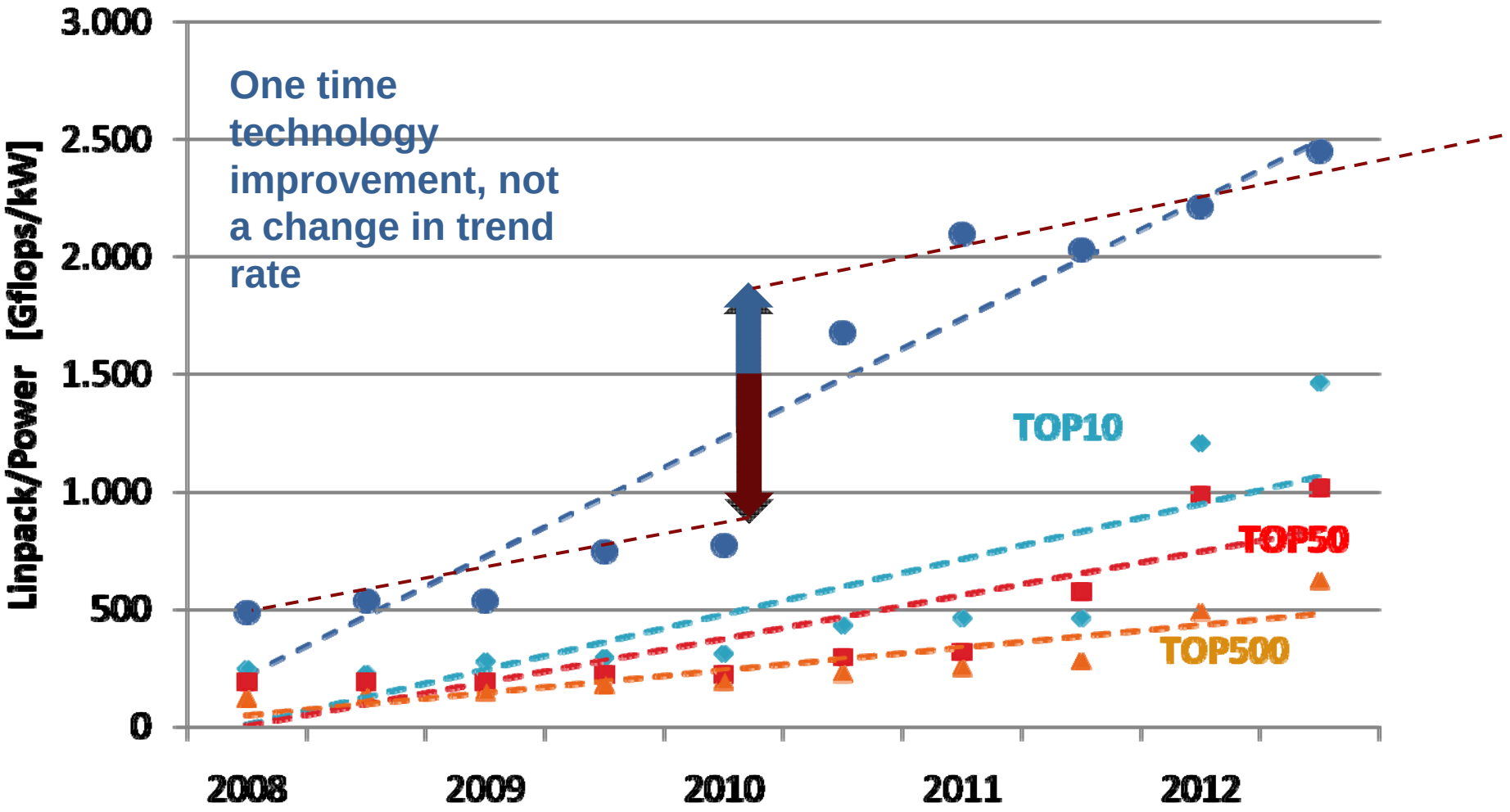
[Tflops/MW] = [Mflops/Watt]

Power Efficiency over Time



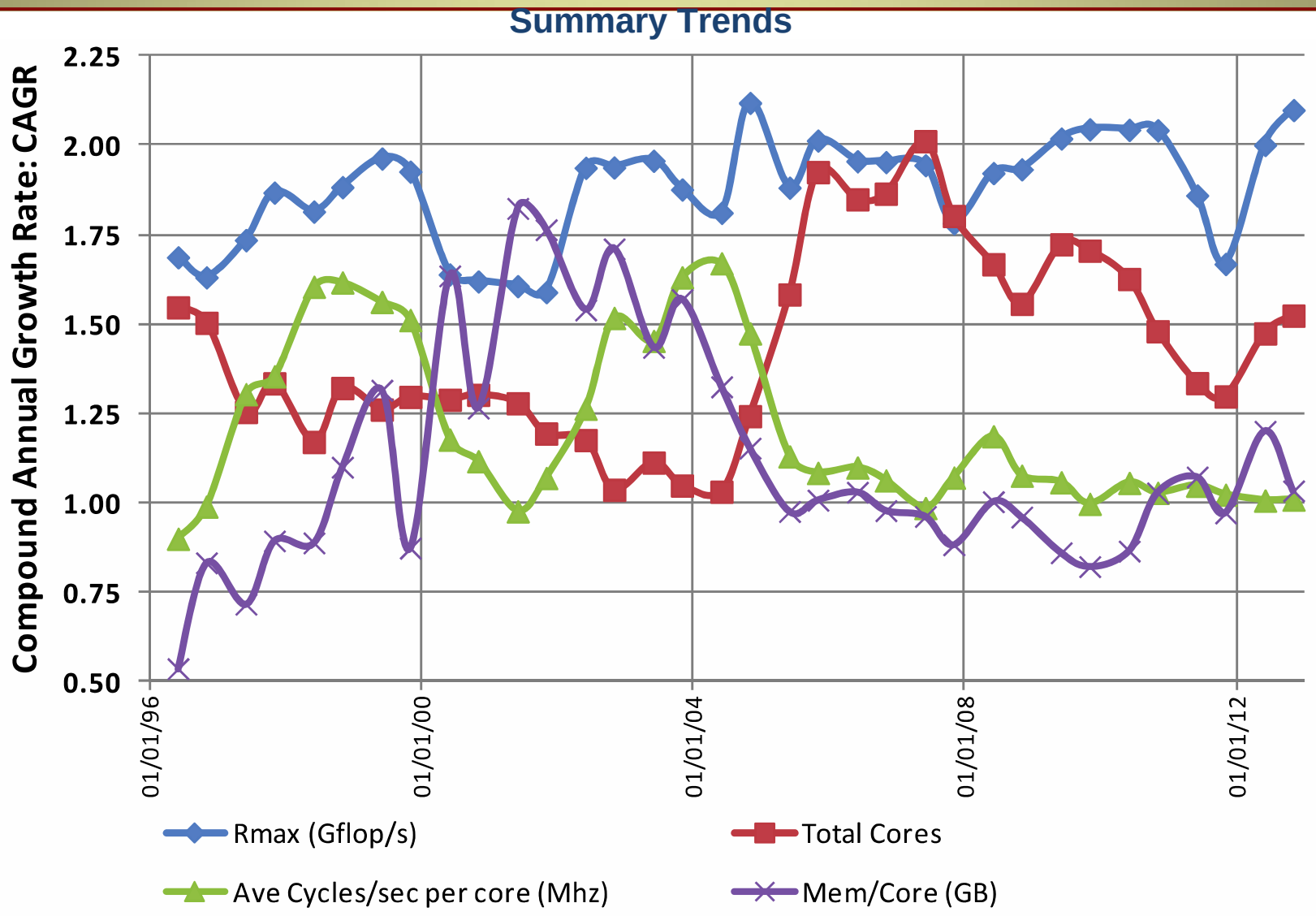
Data from: TOP500 November 2012

Power Efficiency over Time



Data from: TOP500 November 2012

It's the End of the World as We Know It!



Source: Kogge and Shalf, IEEE CISE 2013

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 *(AMM says what is bad, Proxy says just how bad!)*

	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

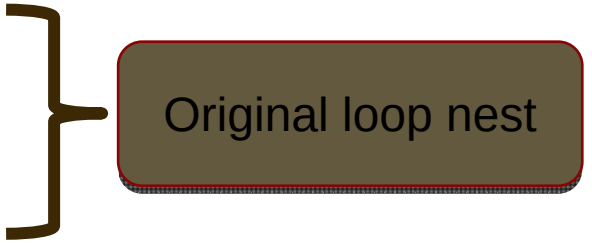
Table 5.1: *Opt*1 and *Opt*1 represent possible proxy options for the abstract machine model. *M.C*: multi-core, *Acc*: Accelerator, *BW*: bandwidth, *Proc*: processor, For models with accelerators and cores, *C* denotes to FLOP/s from the CPU cores and *A* denotes to FLOP/s from Accelerators.

Iterating over Tiles

```
do j=lo(2), hi(2)
   do i=lo(1), hi(1)

      B(i,j)= A(i,j) ...

   end do
end do
```



Iterating over Tiles (Lambda Functions)

```
do tileno=1, ntiles (tiledA)

  tl = get_mtile(tiledA, tileno)
  lo = lwb(tl)
  hi = upb(tl)
  A => dataptr(tiledA, tileno)
  B => dataptr(tiledB, tileno)

  do j=lo(2), hi(2)
    do i=lo(1), hi(1)

      B(i,j)= A(i,j) ...

    end do
  end do
end do
```

Tiling loop

Get tile and its range

Get data ptrs

Element Loops

Loop body remains unchanged

Changes to tile size or layout are invisible to the loops

Iterating over Tiles: Compiler Support

```
do tileno=1, ntiles (tiledA)

  tl = get_mtile(tiledA, tileno)
  lo = lwb(tl)
  hi = upb(tl)
  A => dataptr(tiledA, tileno)
  B => dataptr(tiledB, tileno)

  do j=lo(2), hi(2)
    do i=lo(1), hi(1)

      B(i,j)= A(i,j) ...

    end do
  end do
end do
```

Tile traversal can be hidden behind an iterator or a loop construct

With a compiler support, metadata retrieval can be hidden from the programmer

Iterating over Tiles: Compiler Support

```
call TidaAlloc(tiledA,size,layout)
do tileno=1, ntiles (tiledA)
```

```
  do j=lo(2), hi(2)
    do i=lo(1), hi(1)

      B(i,j)= A(i,j) ...

    end do
  end do
end do
```

Loop Traversal

~~Decouple the loop traversal from the loop body, why?~~

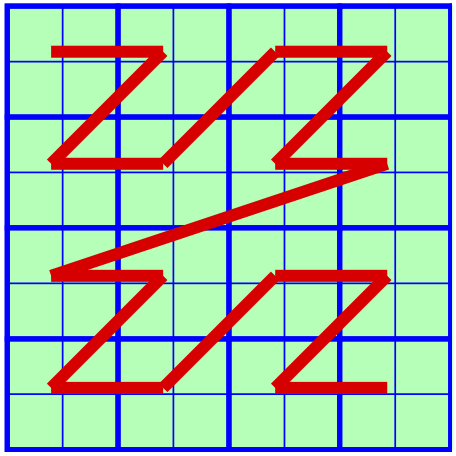
- Can hide any complicated loop traversal ordering behind the iterator interface
- Can change how the loop is parallelized
- Can add GPU acceleration under the hood
- Programmer does not need to implement them all

Introduce a language construct (such as doeach) to make it clean

```
doeach tl in tiledA

 ! Apply the following

end doeach
```



Motivating Examples

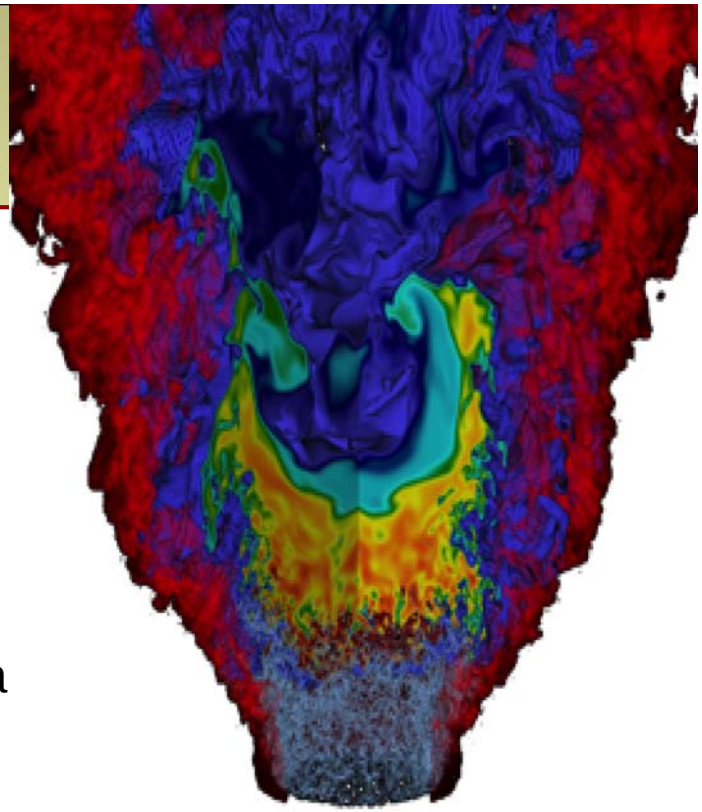
PDE solvers on block structured grids

SMC Proxy App

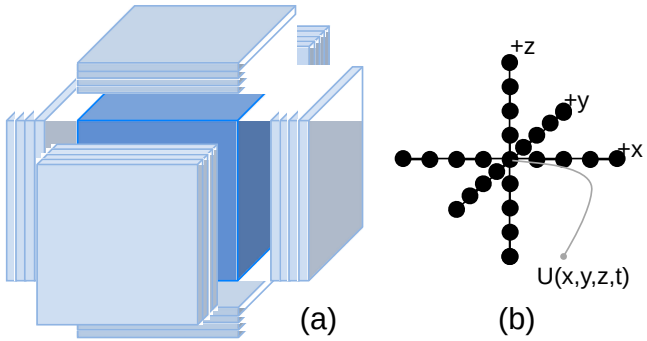
- Developed at the Combustion co-design center
- Compressible Navier Stokes solver
  - Uses the same discretization approach as the petascale application code S3D
  - Captures both the dynamical core and the chemica kinetics components of S3D
  - Uses eight-order finite difference approximation in space and a low-storage Runge-Kutta algorithm in time.

The exascale target for SMC is 50 or more chemical species

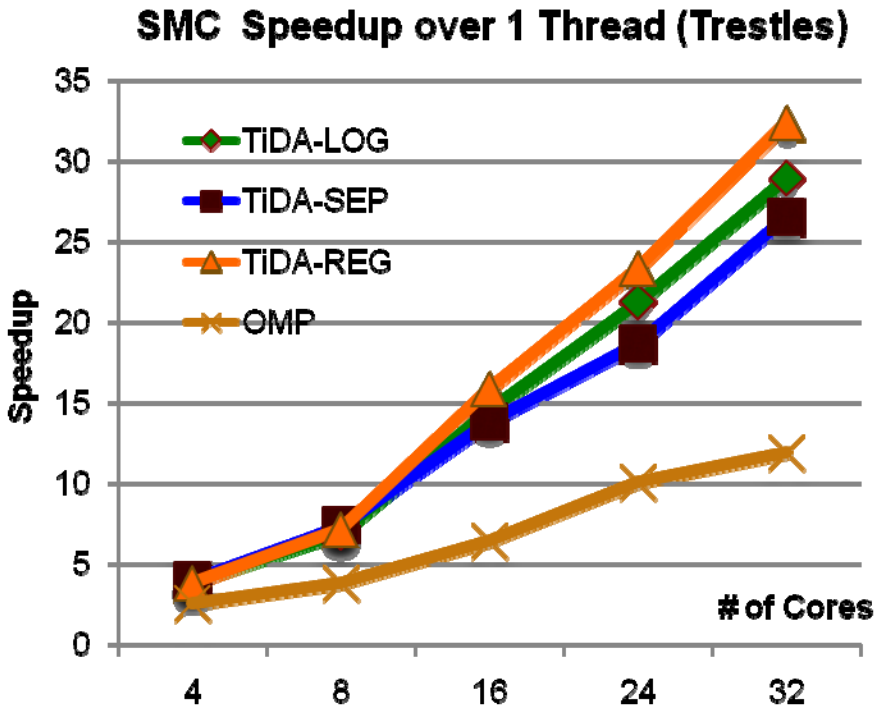
- Results are for 9 species

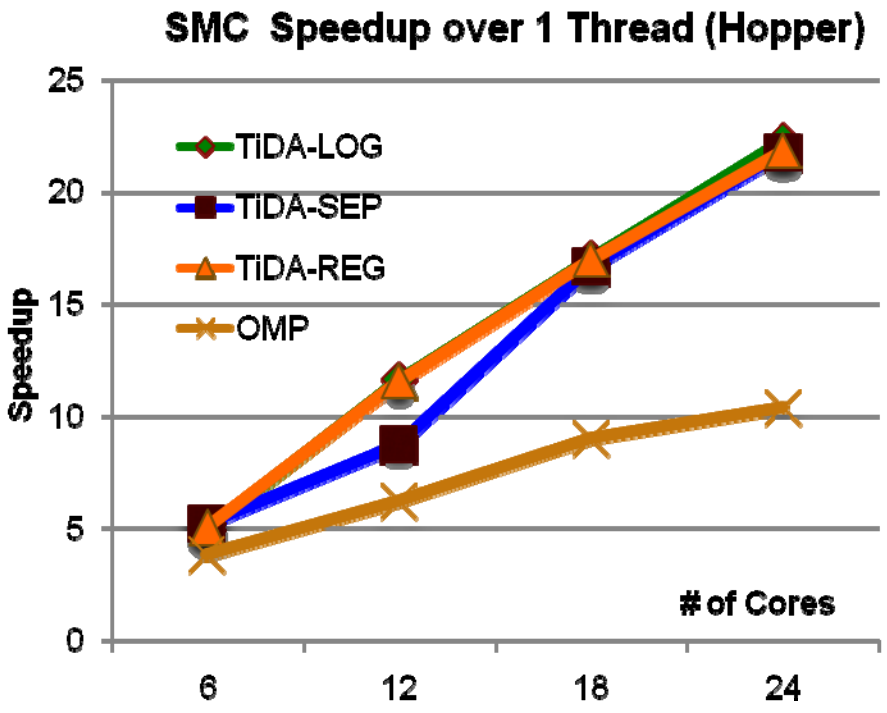


Source: John Bell (LBNL)



SMC Proxy App





TiDA achieves 32x and 22x speedups over single OMP thread on Trestles and Hopper, respectively

What are the NRE and MFR Costs?

sample "quote" from a design firm (probably a bit low, but gives you an idea of rough cost model) (again: anonymized)

Parameter	Value
Foundry	TSMC
Process Technology	40nm 1P10M (m1 7x2y) , RDL, Bump, VTCp, ESD
Die size estimate	16mm x 16mm
IP	See IP Summary Slide
Package	1156 FCBGA, 3:2:3-layer, 35x35 body size, 1 mm ball pitch
Tester Platform	Agilent-93K-640-300MHz
Test Time	Wafer Sort: 10s Final Test: 10s

Component	Amount
Manufacturing NRE Masks, 12 Prototype Char Wafers, 150 Prototypes, Process Eng, Product Eng, Project Management	$1,512,970
Package NRE Package Tooling and Package Engineering	$23,460
Test Development NRE Test Engineering and Tester Rental Time	$82,800
IP NRE IP Licensing Fees, Support and Maintenance	$918,000
Characterization NRE Char units, Tester rental, Test Engineering, Process Engineering, Char report	$52,000
Qualification NRE Q&R Engineering, HTOL, TMCL, HTSL, UHAST, ESD & LU	$225,000
Total NRE	$2,814,230

IP Description
PCIe Gen 2 PHY $200k (could be IB)
PCIe Gen 2 End point controller $80K
DDR3 PHY $338K
DDR3 Controller $100K

First 2.5 Ku	Next 2.5 Ku	Next 5 Ku	Next 10 Ku	Additional
$121.60	$119.76	$117.97	$87.78	$74.80

With Marty Deneroff

What are the NRE and MFR Costs?

sample "quote" from a design firm (probably a bit low, but gives you an idea of rough cost model) (again: anonymized)

Notice that the steady state MFR cost bottoms out after about 10k units

A typical large-scale HPC system requires more than 10k units (sockets)

Moreover, Spreading a $10M NRE over 100k units (small-run for limited # of large scale HPC systems) is about $100/unit.

Economically Practical Design Point

Test Time	Wafer Sort: 10s Final Test: 10s

IP Description
PCIe Gen 2 PHY $200k (could be IB)
PCIe Gen 2 End point controller $80K
DDR3 PHY $338K
DDR3 Controller $100K

Total NRE $2,814,230

First 2.5 Ku	Next 2.5 Ku	Next 5 Ku	Next 10 Ku	Additional
$121.60	$119.76	$117.97	$87.78	$74.80

Commoditization Strategies
(alternative approaches to amortize NRE)

Chip is the commodity (CPU and GPU w/1TF/chip in today's tech)

- NRE: $1B to design each generation
- Mfr. Costs: $100/chip for 240mm and pennies for 7mm
- Most costs are in verification of full custom circuit design IP is mostly proprietary
- Design and Verification NRE is shared across products using that chip
- GPUs and CPUs specialized to different market (some waste)

ASICs using commodity IP (for a 0.5TF chip but more control of design)

- NRE: $2M in IP, $5M in assembly and verification, $2M for Mask + fab
- Mfr. Costs: $200/chip for initial 10K, and $100/chip beyond 50k chips
- NRE spread across 200k chips for large system is $50/chip
- Still have SW costs (but same baseline as commodity chip)
- No extra baggage in design (only include what you need for broad HPC application mix. Concentrate design + verification costs on small subset of design that needs to change)

Potential System Architectures
(*original version from 2009 workshop)*

Systems	2009	2015	2018
System peak	**2 Peta**	**100-200 Peta**	**1 Exa**
Power	**6 MW**	**~10 MW**	**~20 MW**
System memory	0.3 PB	5 PB	10 PB
Node performance	125 GF	400 GF	1-10TF
Node memory BW	25 GB/s	200 GB/s	>400 GB/s
Node concurrency	12	O(100)	O(1000)
Interconnect BW	1.5 GB/s	25 GB/s	50 GB/s
System size (nodes)	18,700	250,000-500,000	O(million)
Total concurrency	225,000	O(million)	O(billion)
Storage	15 PB	150 PB	500 PB
IO	0.2 TB	10 TB/s	50 TB/s
MTTI	days	days	O(1 day)

Potential System Architectures
(updates for 2014… what did we get wrong)

Systems	2009	2015	2018 2024
System peak	**2 Peta**	**100-200 Peta**	**1 Exa**
Power	**6 MW**	**~10 MW *15MW***	**~20 MW**
System memory	0.3 PB	~5 PB ***yes!***	10 PB
Node performance	125 GF	400 GF ***3TF***	1-10TF 10-12TF
Node memory BW	25 GB/s	200 GB/s (2-level!!) ***100GB/s@100GB + 500GB/s@16GB***	>400 GB/s (2-level) 250GB/s@200GB + 4TB/s @ 32-64GB
Node concurrency	12	O(100) *yes*	O(1000) *yes*
Interconnect BW (node)	1.5 GB/s	25 GB/s ***10-15GB/s***	50 GB/s ***100+ GB/s***
System size (nodes)	18,700	250,000-500,000 30,000 – 60,000	O(million) ***yes***
Total concurrency	225,000	O(million)	O(billion)
Storage	15 PB	150 PB	500PB
IO	0.2 TB	10 TB/s ***+ burst buffer 100 TB***	50 TB/s ***+ burst buffer***