The Exascale Challenge:

How Technology Disruptions Fundamentally Change Programming Systems

John Shalf

Department Head: Computer Science and Data Sciences (CSDS)

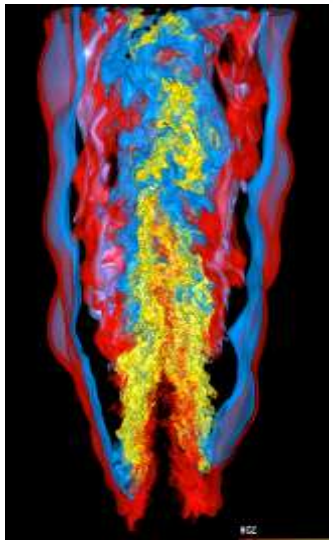
CTO: National Energy Research Scientific Computing Center (NERSC)

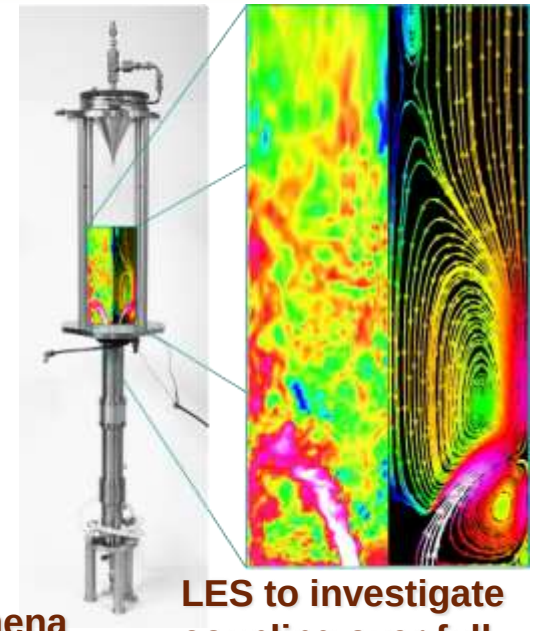
E3S Workshop, October 28, 2013

High End Modeling and Data Assimilation For Advanced Combustion Research

Approach: Combine unique codes and resources to maximize benefits of high performance computing for turbulent combustion research

Advanced "capability-class" solvers





DNS to investigate combustion phenomena at smallest scales
- *no modeling*
- *limited applicability*

LES to investigate coupling over full range of scales in experiments
- *minimal modeling*
- *full geometries*

Access to leading edge computational resources

CRF Computational Combustion and Chemistry Laboratory

Combustion Research and Computational Visualization Facility

Visualization Cluster:
34 Opteron™ processors with high-end graphics cards, Gigabyte Ethernet, 50 terabyte parallel file system.

EERE System:
256 Opteron™ processors, InfiniBand, 10 terabytes NFS disk storage.

BES System:
264 Opteron™ processors, InfiniBand, 15 terabytes NFS disk storage.

Joint OS-EERE Funding

DOE Office of Science Laboratories
- LBNL NERSC
- ORNL OLCF
- ANL ALCF

INCITE Program

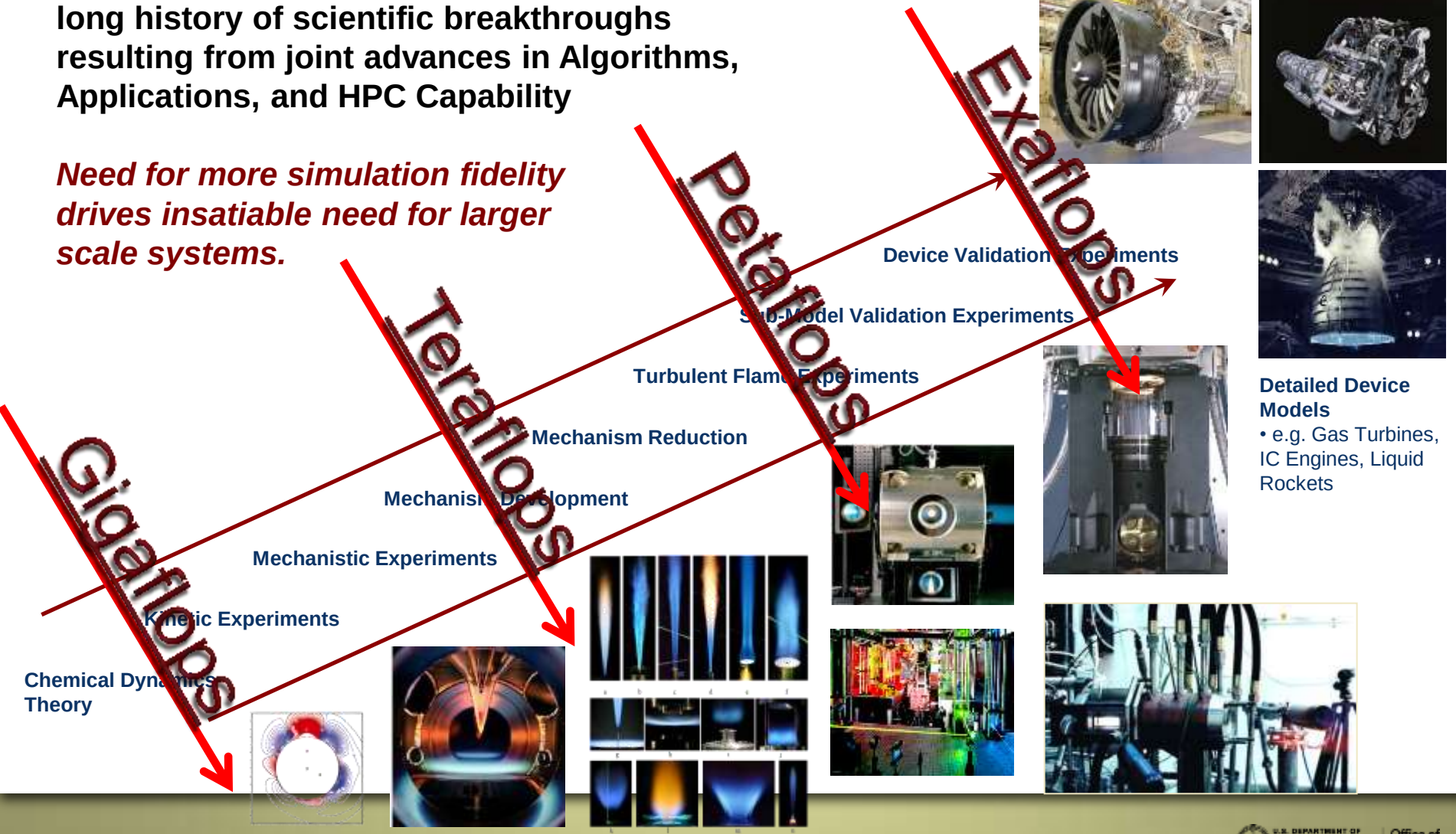
Image co

Ofelein, Chen: Sandia 2009

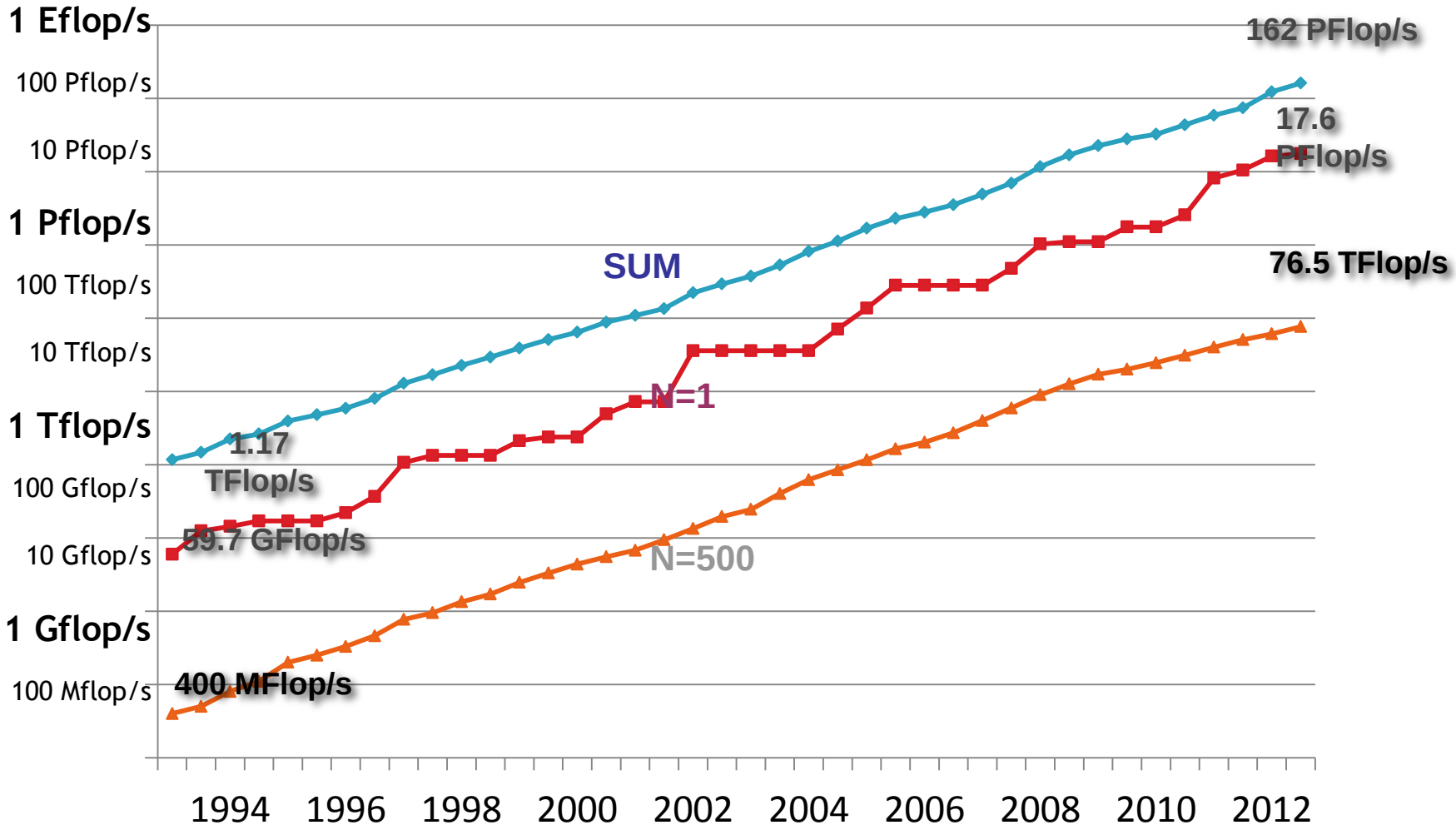
Scientific Breakthroughs Enabled by Algorithms, Applications, and HPC Capability

Combustion Research has demonstrated a long history of scientific breakthroughs resulting from joint advances in Algorithms, Applications, and HPC Capability

Need for more simulation fidelity drives insatiable need for larger scale systems.

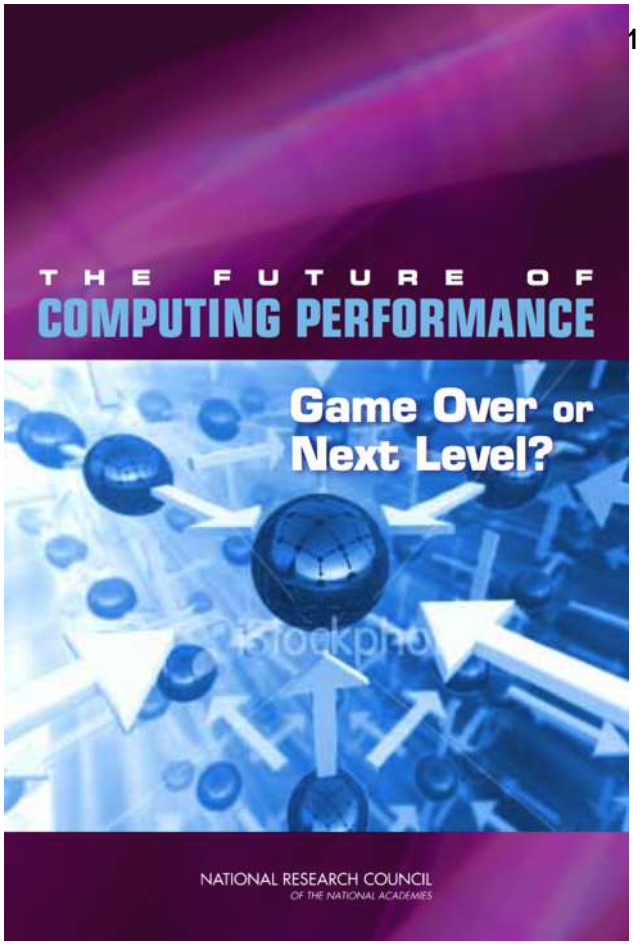


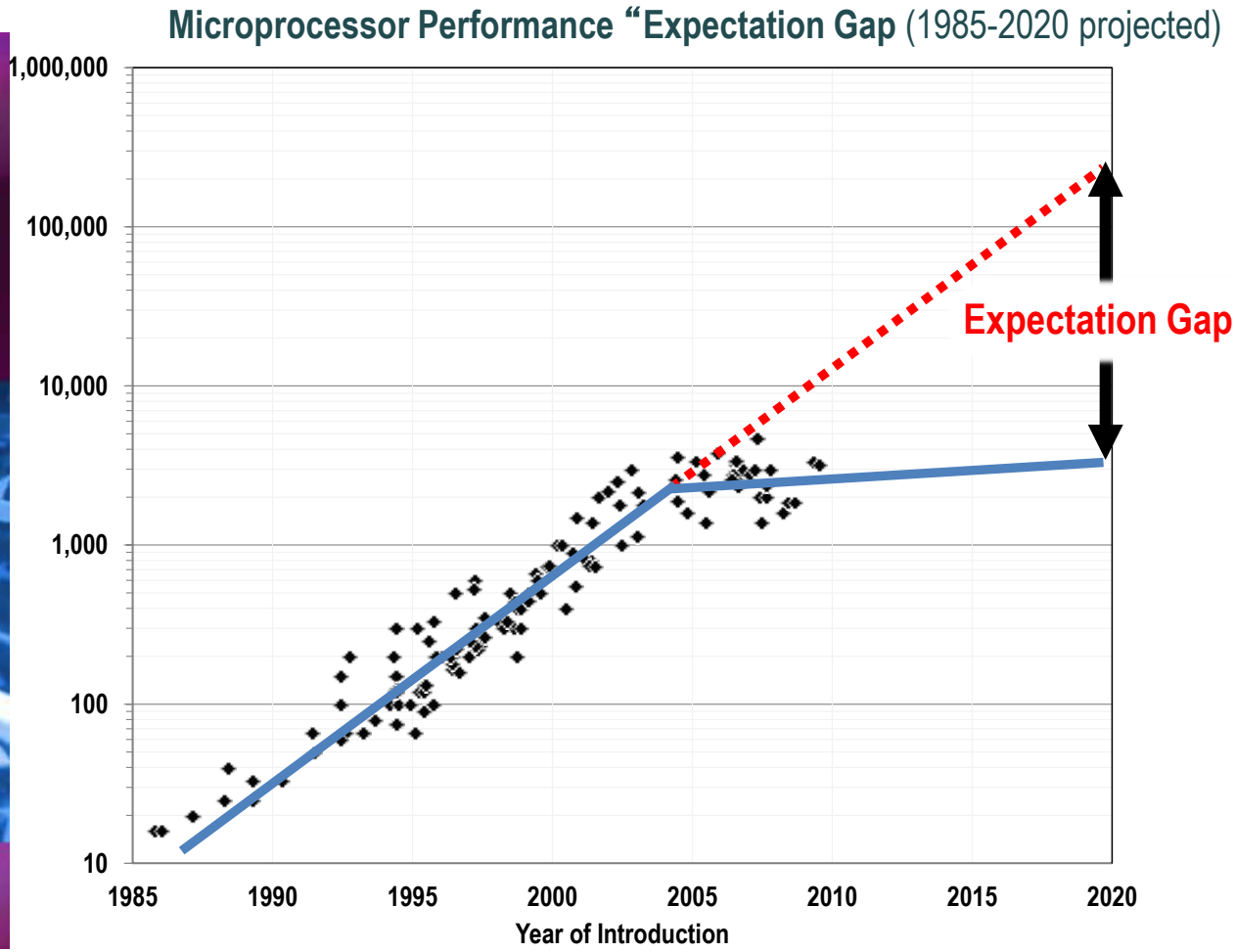
Two Decades of Exponential Performance Improvements



Source: TOP500 November 2012

Computing Crisis is Not Just about Exascale

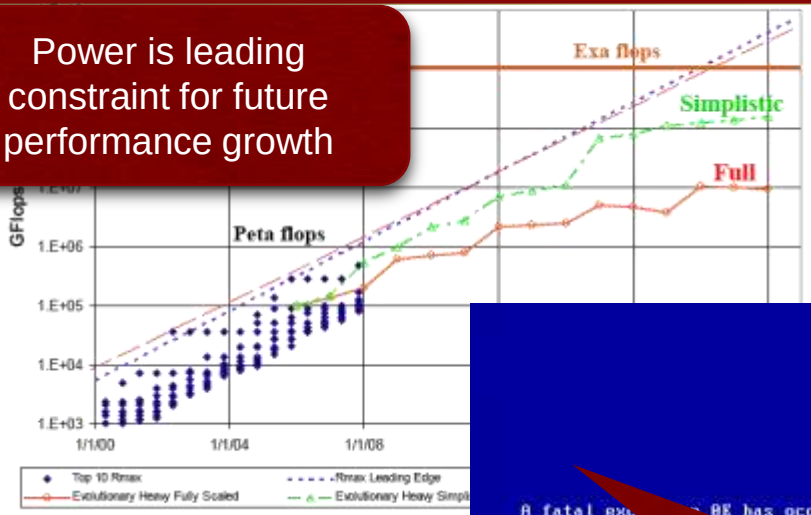




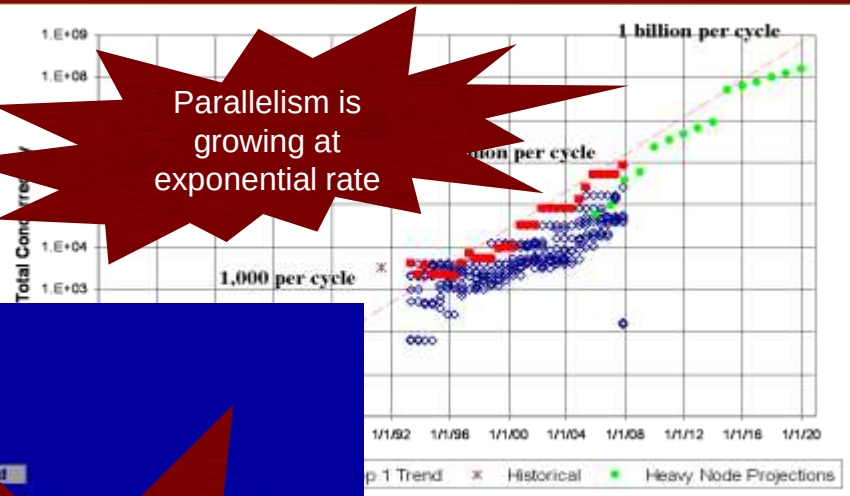
Industry motivated, path forward is unclear

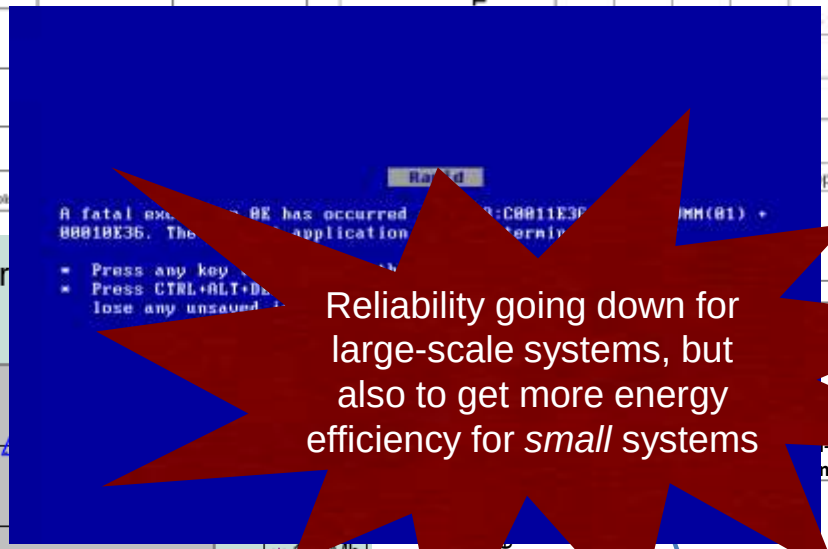
Technology Challenges for the Next Decade

Power is leading constraint for future performance growth

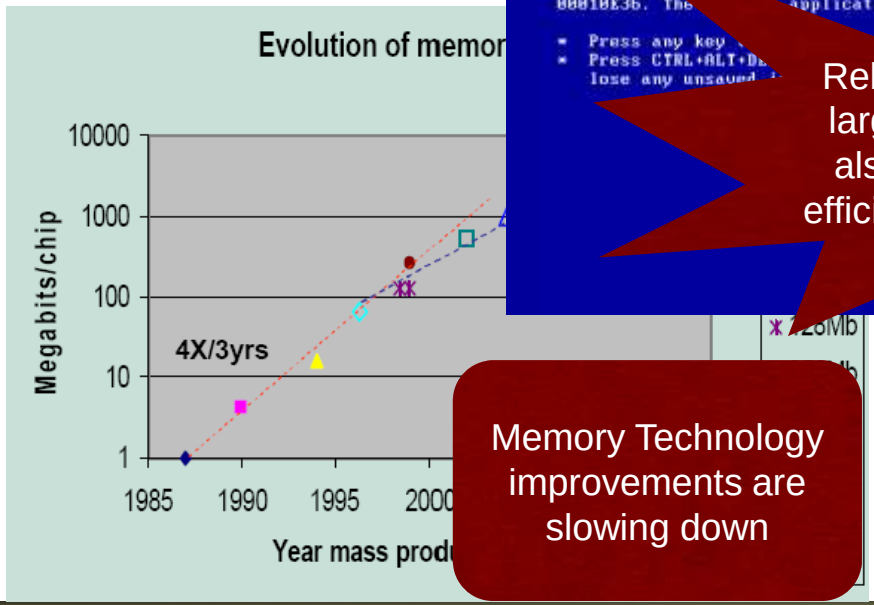


Parallelism is growing at exponential rate

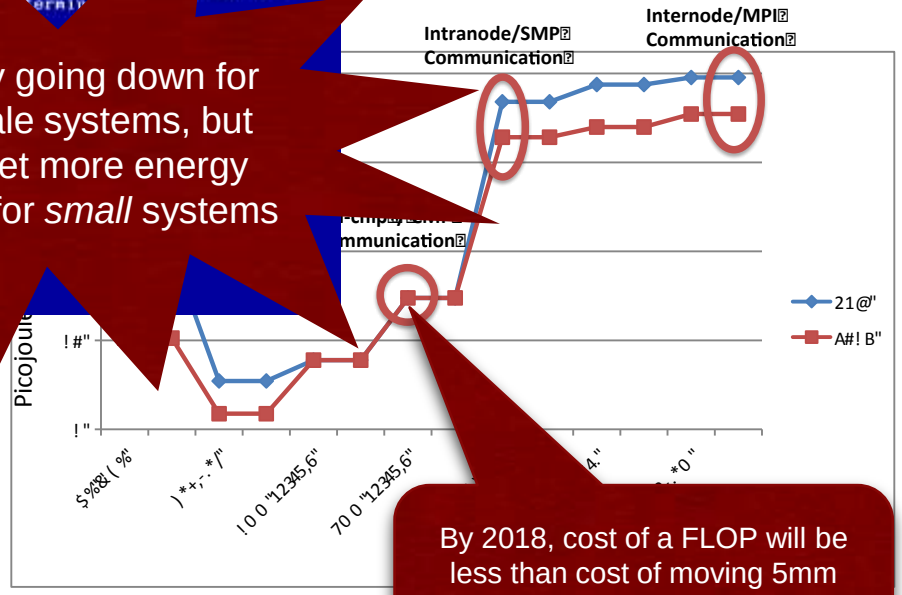




Reliability going down for large-scale systems, but also to get more energy efficiency for *small* systems



Memory Technology improvements are slowing down

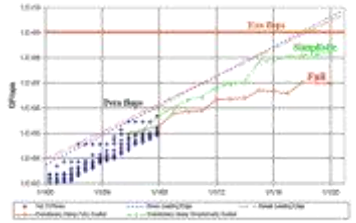


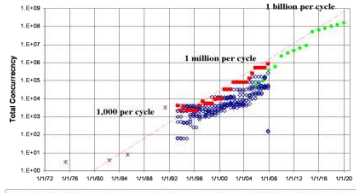
By 2018, cost of a FLOP will be less than cost of moving 5mm across the chip's surface (locality will *really* matter)

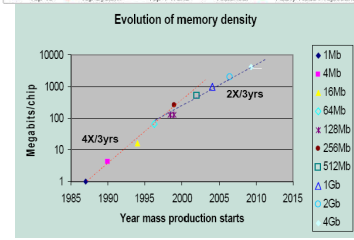
Whats wrong with current HPC Systes?

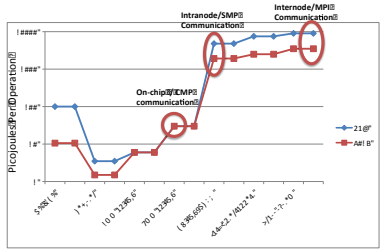
Designed for Constraints from 30 years ago! ***(wrong target!!)***

Old Constraints	New Constraints
• ***Peak clock frequency*** *as primary limiter for performance improvement*	• ***Power*** *is primary design constraint for future HPC system design*
• **Cost:** *FLOPs* are biggest cost for system: *optimize for compute*	• **Cost:** *Data movement dominates: optimize to minimize data movement*
• **Concurrency**: Modest growth of parallelism by adding nodes	• **Concurrency:** *Exponential growth of parallelism within chips*
• **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 are a Reflection of the Underlying Machine Architecture

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

Programming Models are Increasingly Mismatched with Underlying Hardware Architecture

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

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

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

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

Equal cost SMP/PRAM model

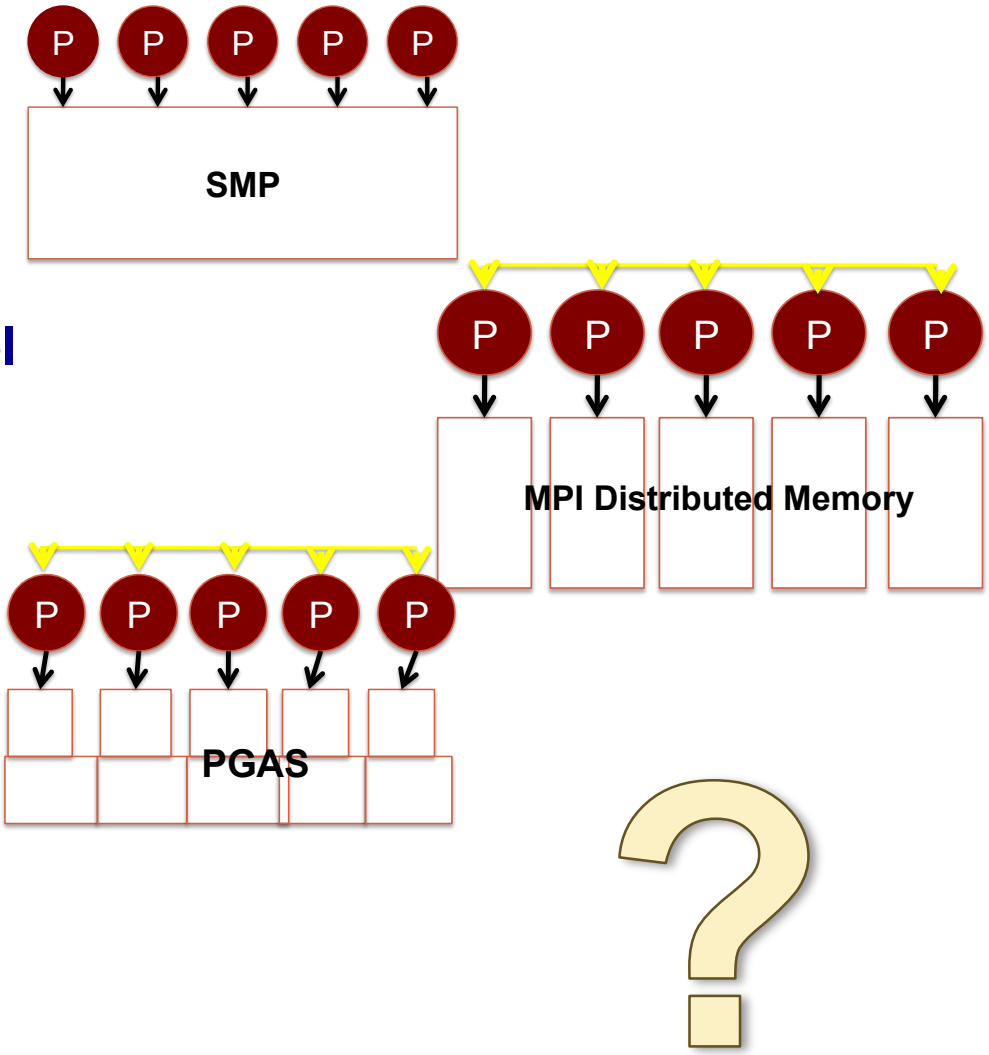
- No notion of non-local access
- int [nx][ny][nz];

Cluster: Distributed memory model

- No unified memory
- int [localNX][localNY][localNZ];

PGAS for horizontal locality

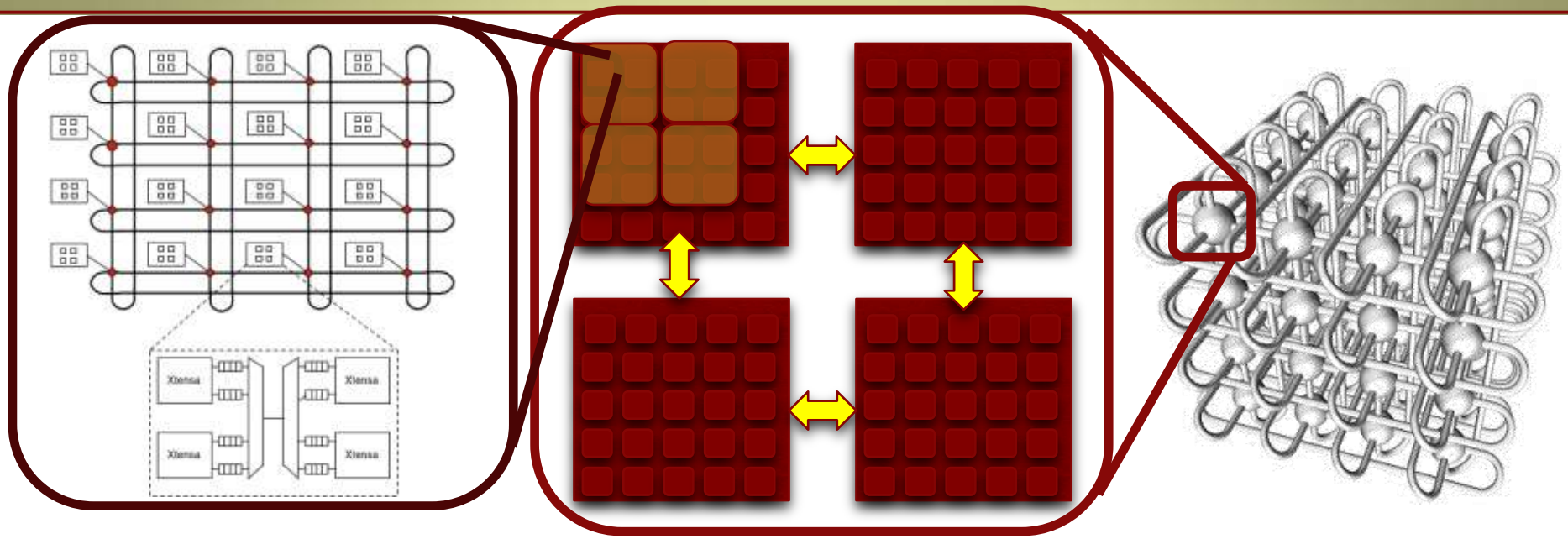
- Data is LOCAL or REMOTE
- shared [Horizontal] int [nx][ny][nz];



Whats Next?

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)

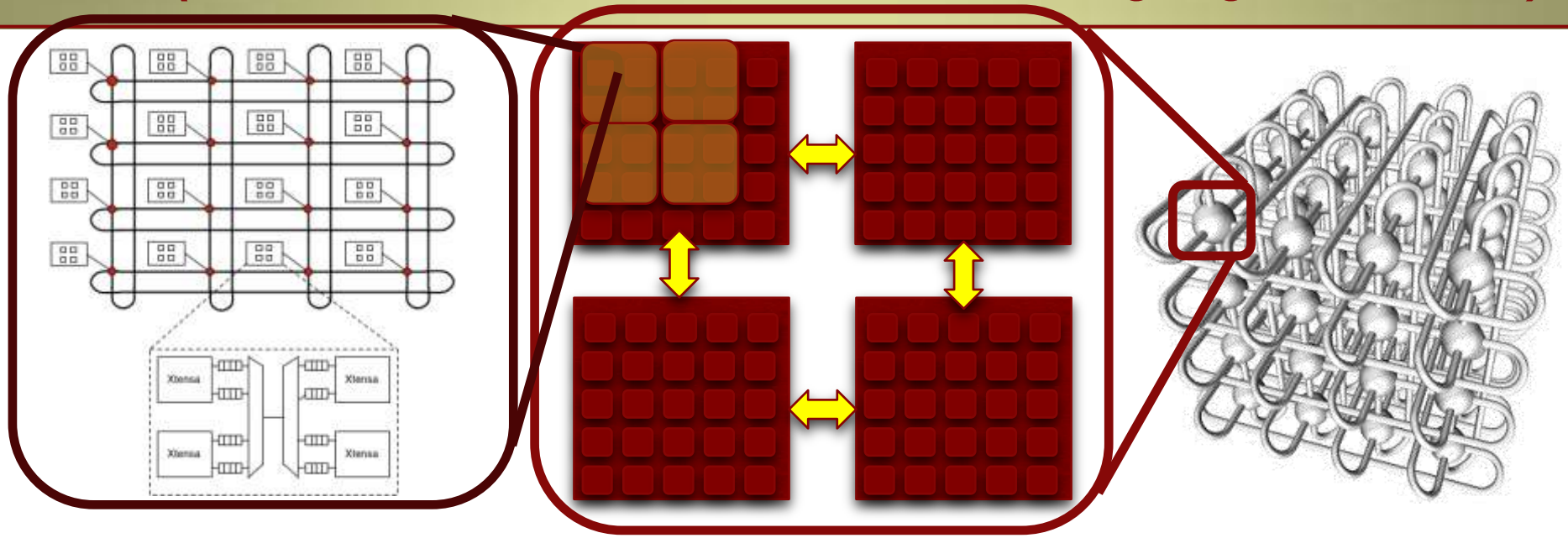
Fault Model for Node
- FIT rates, Kinds of faults
- Granularity of faults/recovery

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

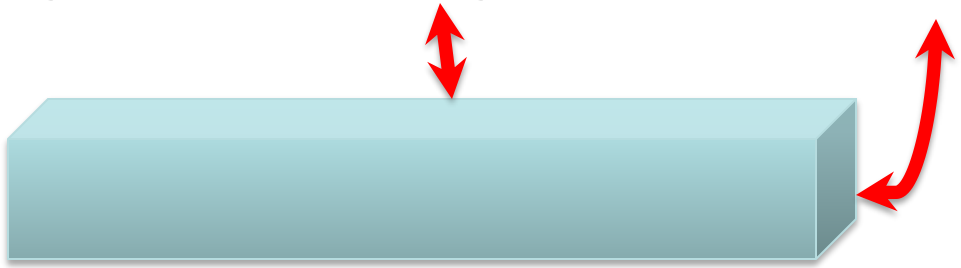
- **Ignore it:** *If ignoring it has no serious power/performance consequences*
- **Expose it (*unvirtualize*):** *If there is not a clear automated way of make decisions*
  - Must involve the human/programmer in the process ***(make pmodel more expressive)***
  - Directives to control data movement 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 Problem with Wires:

Energy to move data proportional to distance

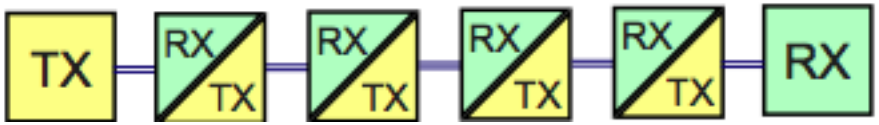
- **Cost to move a bit on copper wire:**
  - **energy = bitrate * Length^2 / cross-section area**



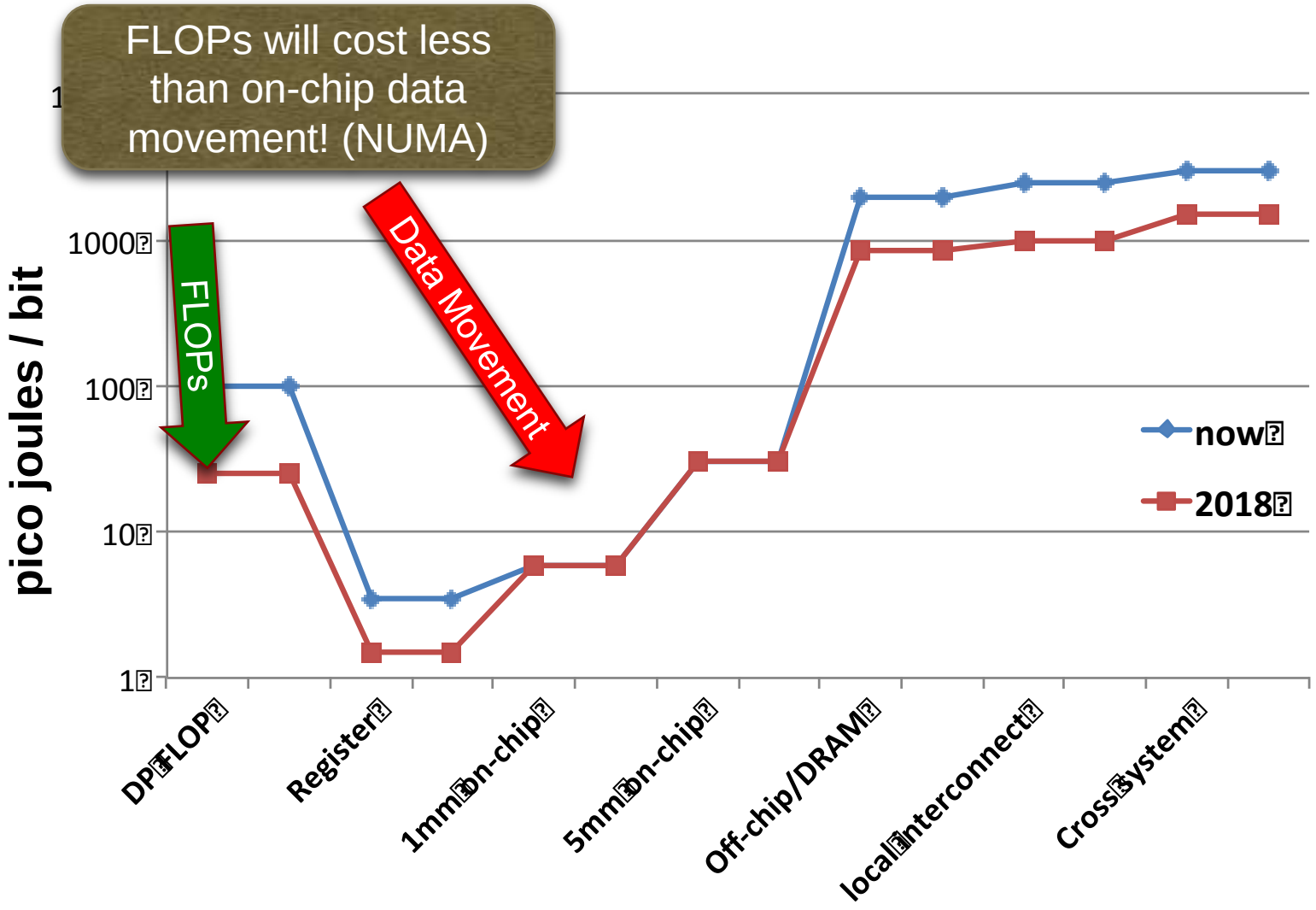
- **Wire data capacity constant as feature size shrinks**
- ***Cost to move bit proportional to distance***
- ***~1TByte/sec max feasible off-chip BW (10GHz/pin)***
- ***Photonics reduces distance-dependence of bandwidth***

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



The Logical Conclusion

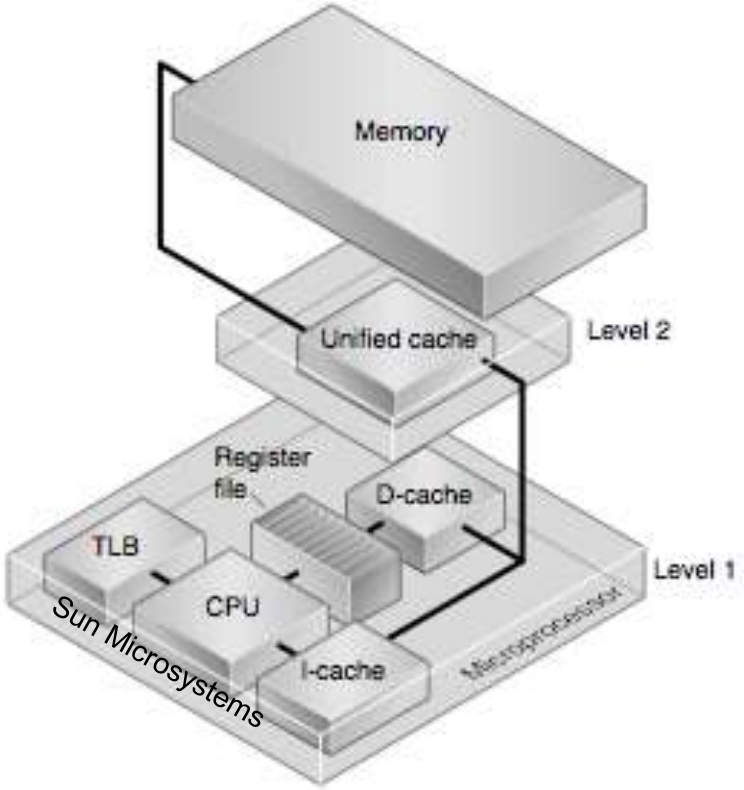
If FLOPS are free, then why do we need an "exaflops" initiative?

"Exa"-anything has become a bad brand

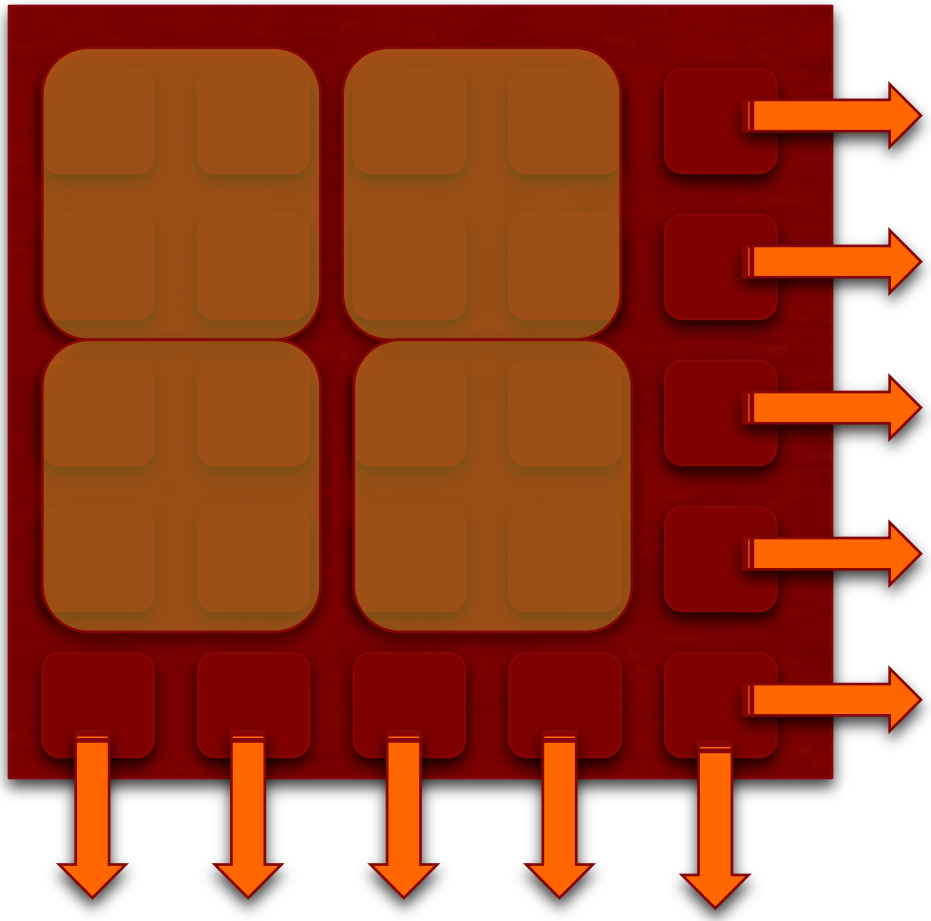
- **Associated with buying big machines for the labs**
- **Associated with "old" HPC**
- **Sets up the community for "failure", if "goal" can't be met**

Data Locality Management

Vertical Locality Management
(spatio-temporal optimization)



Horizontal Locality Management
(topology optimization)



Current Practices (2-level Parallelism)

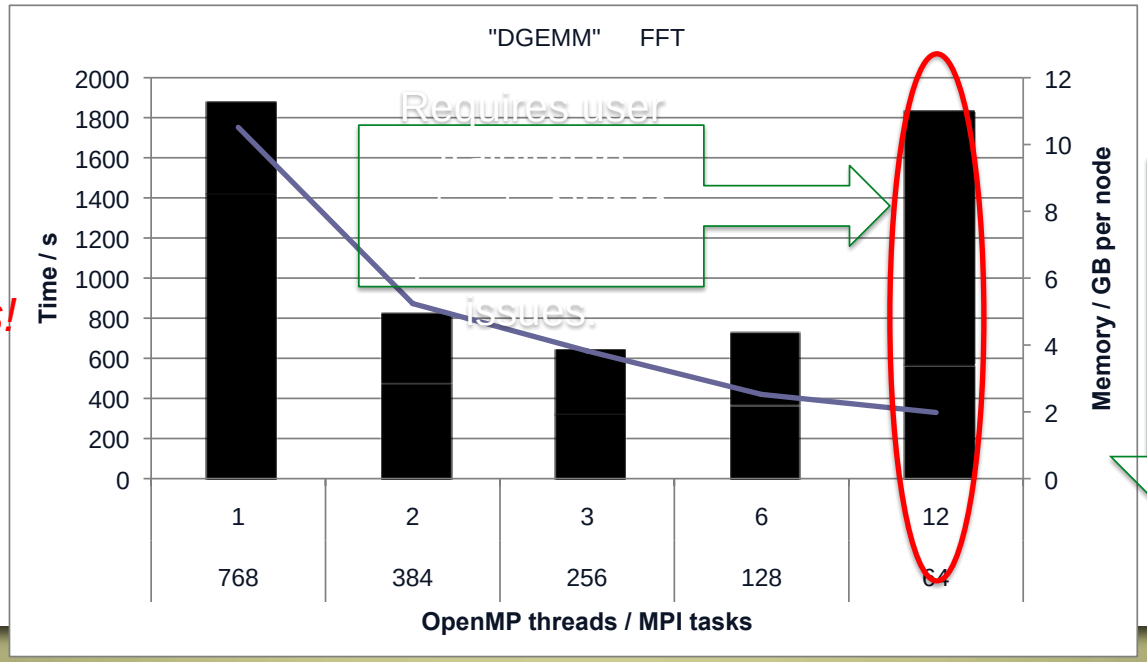
NUMA Effects Ignored (with huge consequence)

MPI+OMP Hybrid

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

It's the Revenge of the SGI Origin2000

Bad News!



Expressing Hierarchical Layout

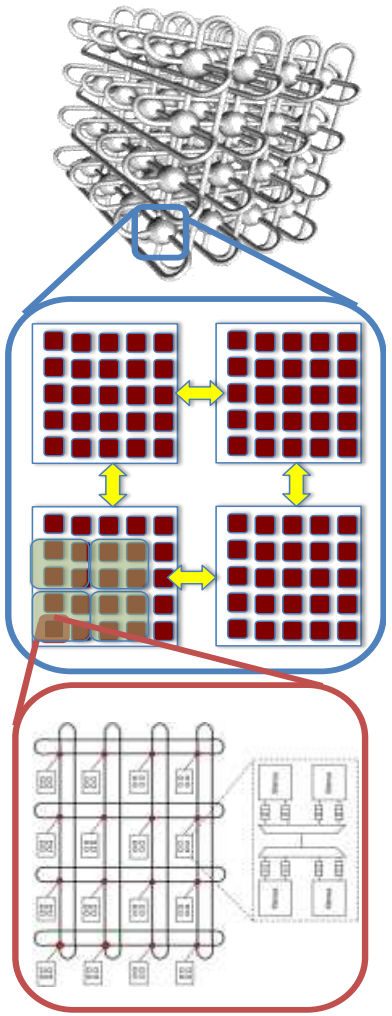
Old Model (OpenMP)

- 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

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



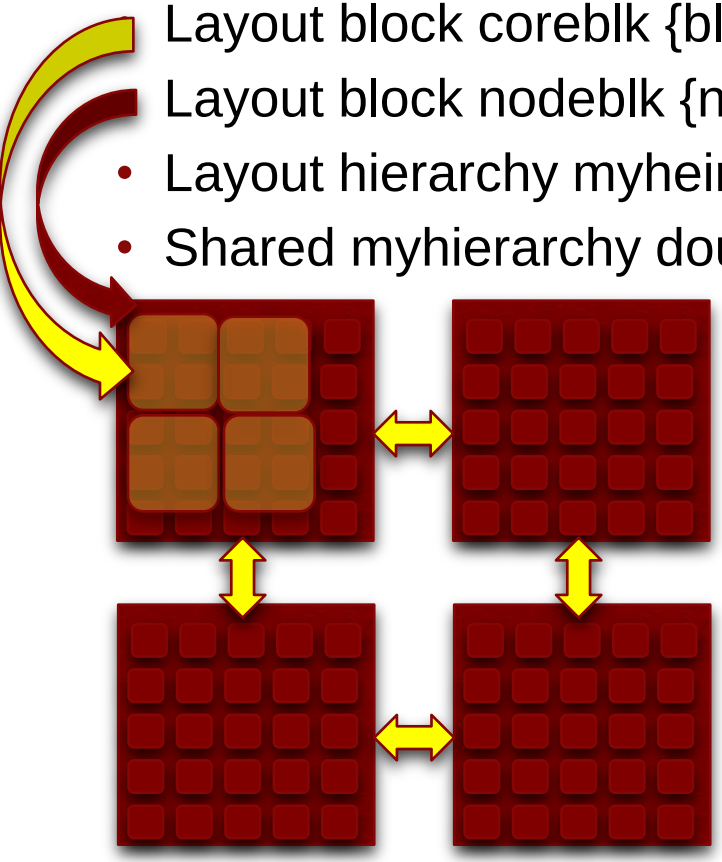
U.S. DEPARTMENT OF ENERGY | Office of Science

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 hierarchy myheirarchy {coreblk,nodeblk};
- Shared myhierarchy double a[nx][ny][nz];



- **Then use data-localized parallel loop**

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

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

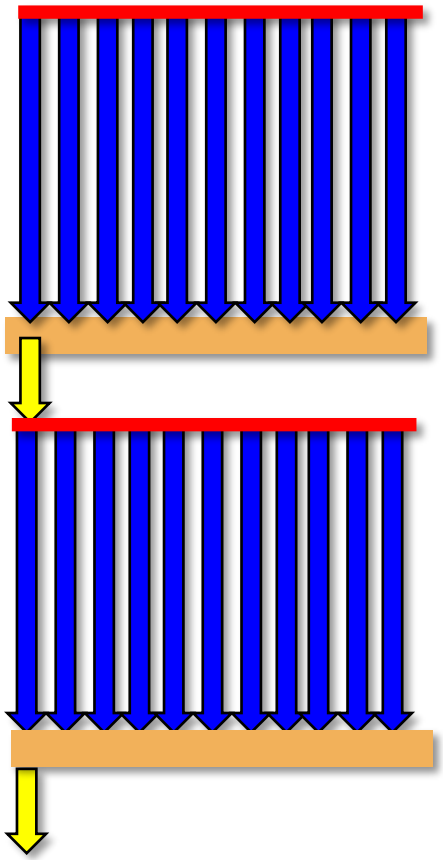
Satisfies the request of the application developers (minimize the amount of code that changes)

U.S. DEPARTMENT OF ENERGY | Office of Science

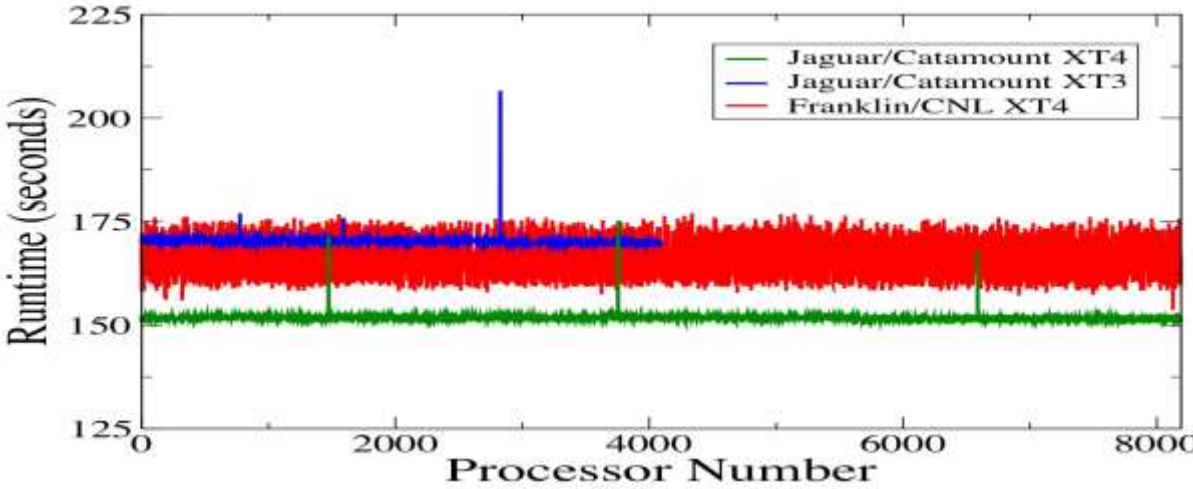
Heterogeneity / Inhomogeneity Async Programming Models?

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

Bulk Synchronous Execution

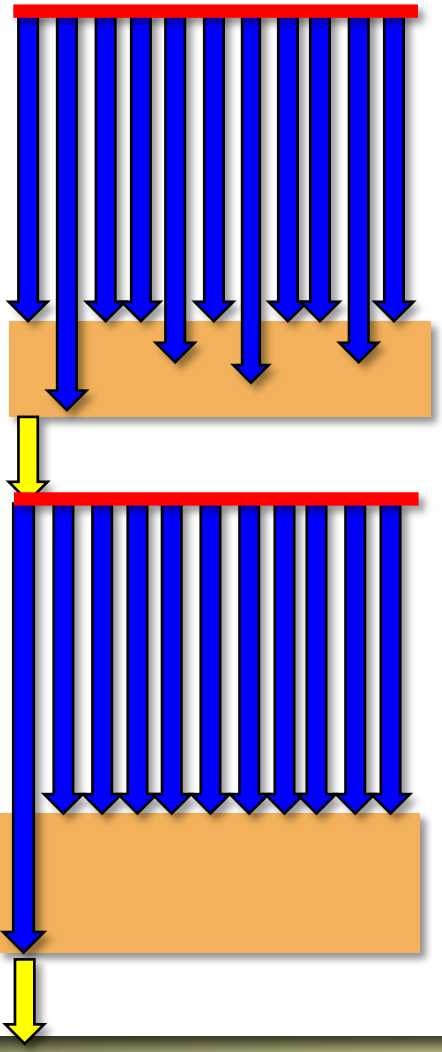


- **Heterogeneous compute engines (hybrid/GPU computing)**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  - *thermal throttling – no longer guarantee deterministic clock rate*
- **Nonuniformities in process technology creates non-uniform operating characteristics for cores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

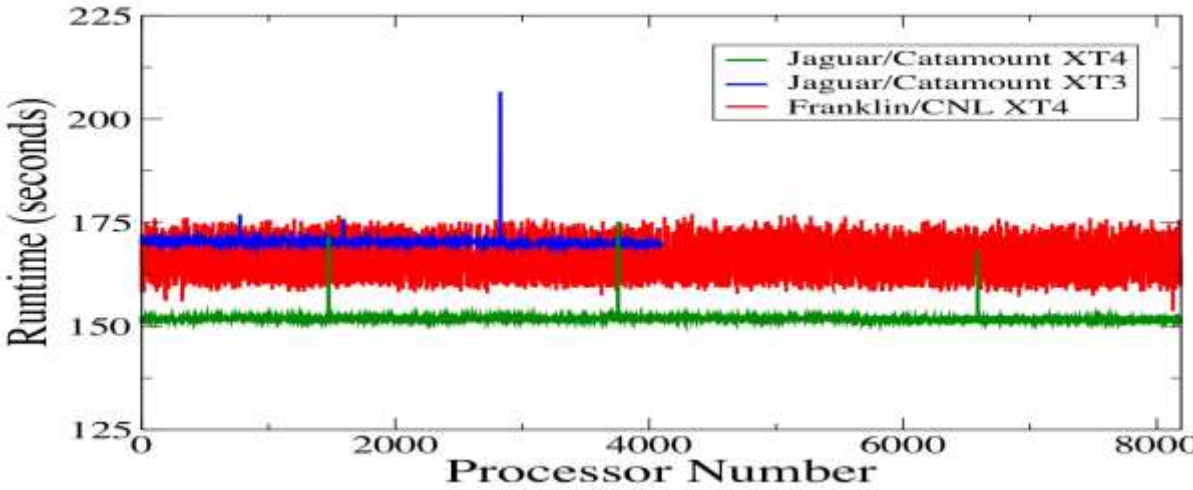


Assumptions of Uniformity is Breaking *(many new sources of heterogeneity)*

Bulk Synchronous Execution

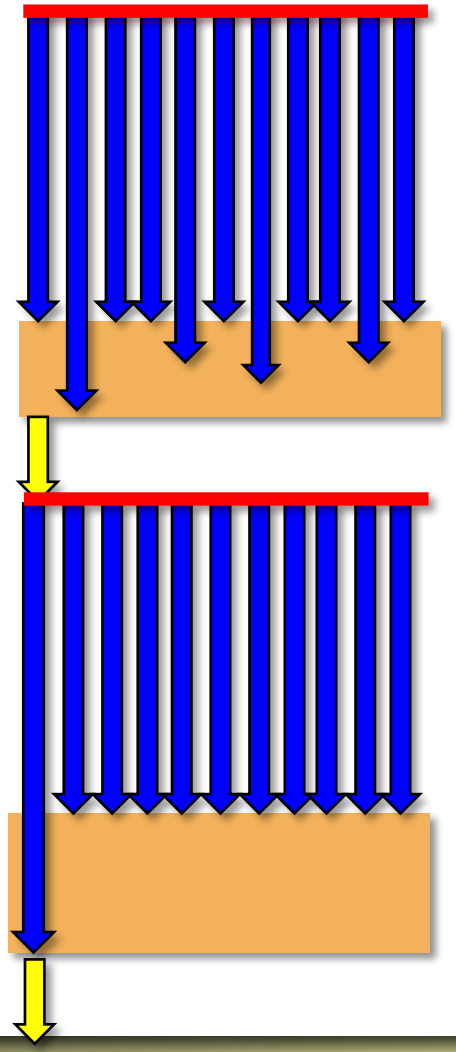


- **Heterogeneous compute engines (hybrid/GPU computing)**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  - *thermal throttling – no longer guarantee deterministic clock rate*
- **Nonuniformities in process technology creates non-uniform operating characteristics for cores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

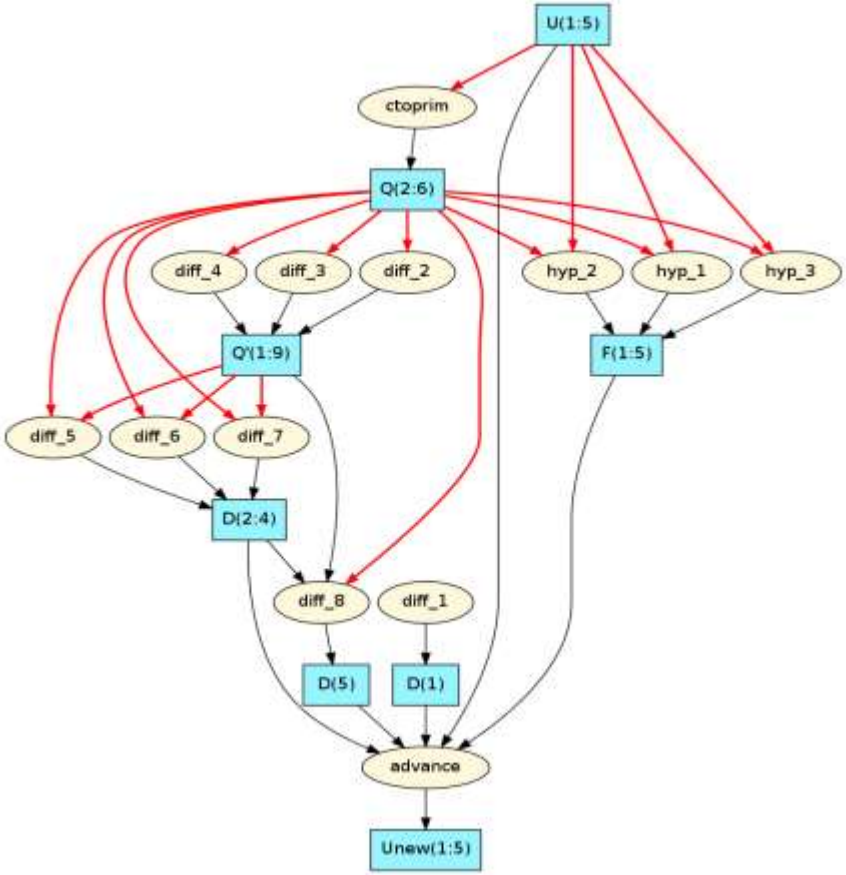


Assumptions of Uniformity is Breaking *(many new sources of heterogeneity)*

Bulk Synchronous Execution



Asynchronous Execution Model



U.S. DEPARTMENT OF ENERGY | Office of Science

Conclusions on Heterogeneity

Sources of performance heterogeneity increasing

- Heterogeneous architectures (accelerator)
- Thermal throttling
- Performance heterogeneity due to transient error recovery

Current Bulk Synchronous Model not up to task

- Current focus is on removing sources of performance variation (jitter), is increasingly impractical
- Huge costs in power/complexity/performance to extend the life of a purely bulk synchronous model

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

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 Environemnt

- 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
- Hardware/Software Codesign MUST consider ergonomic decisions about your programming environment together with performance

Performance Portability Should be Top-Tier Metric for CoDesign process

- Know what to **IGNORE**, what to **ABSTRACT**, and what to make more **EXPRESSIVE**