BLUE WATERS

SUSTAINED PETASCALE COMPUTING

Analytical Performance Modeling and Simulation for Blue Waters

Torsten Hoefler

With contributions from: Greg Bauer, Steven Gottlieb, William Gropp, William Kramer, Marc Snir, IBM, and the Blue Waters team

Keynote at the Workshop on Productivity and Performance (PROPER 2010)
August 30th Ischia Italy


Imagine …

- … you're planning to construct a multi-million Dollar Supercomputer …
- … that consumes as much energy as a small [european] town …
- … to solve computational problems at an international scale and advance science to the next level …
- … with "hero-runs" of [insert verb here] scientific applications that cost $10k and more per run …

… and all you have (now) is …

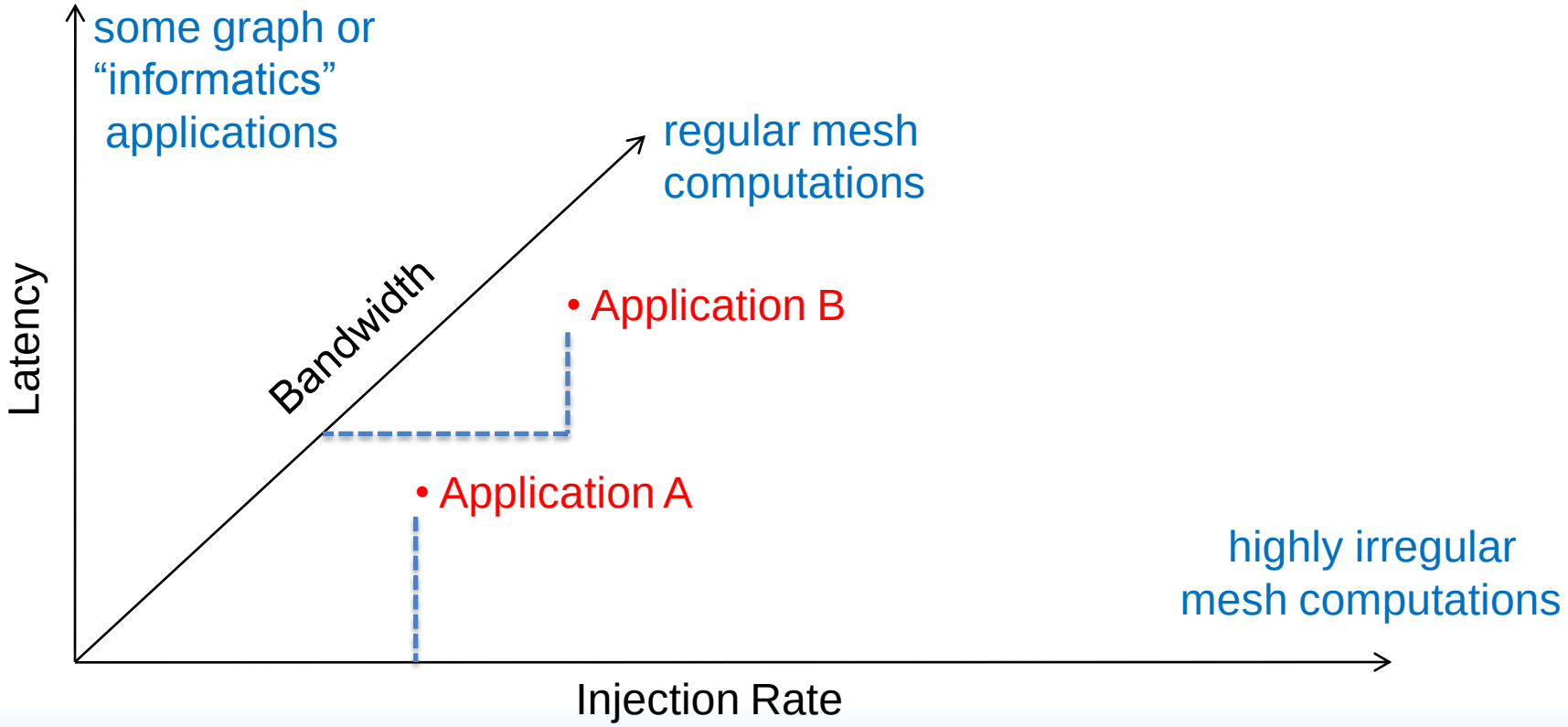
- … then you better plan ahead! (same for Exascale)

HPC is all about "Performance" – What is this?

- FLOP/s?
  - Merriam Webster "flop: to fail completely"
- HPCC: MB/s? GUPS? FFT-rate?
  - It's a multi-dimensional vector space
  - Identify independent vectors
    - network: bandwidth, latency, injection rate, …
    - memory and I/O: bandwidth, latency, random access rate, …
    - CPU: latency (pipeline depth), # execution units, clock speed, …
  - Categorize each vector for a particular machine
    - open a "feasible region" in the n-dimensional space

Example: Memory Subsystem

- each application has particular coordinates



Multi-dimensional Performance Metrics

- Very helpful for HW/SW co-design
  - Good for comparison of applications
  - Good for comparison of machines
  - Design application-optimized (specific) HW
- Not good for absolute estimates
  - App. A on machine A vs. app. B on machine B
  - "Absolute performance" is very hard to define
- Most useful (universal) metric: "**time** to solve problem X"
  - Also most important as time is fundamentally limited

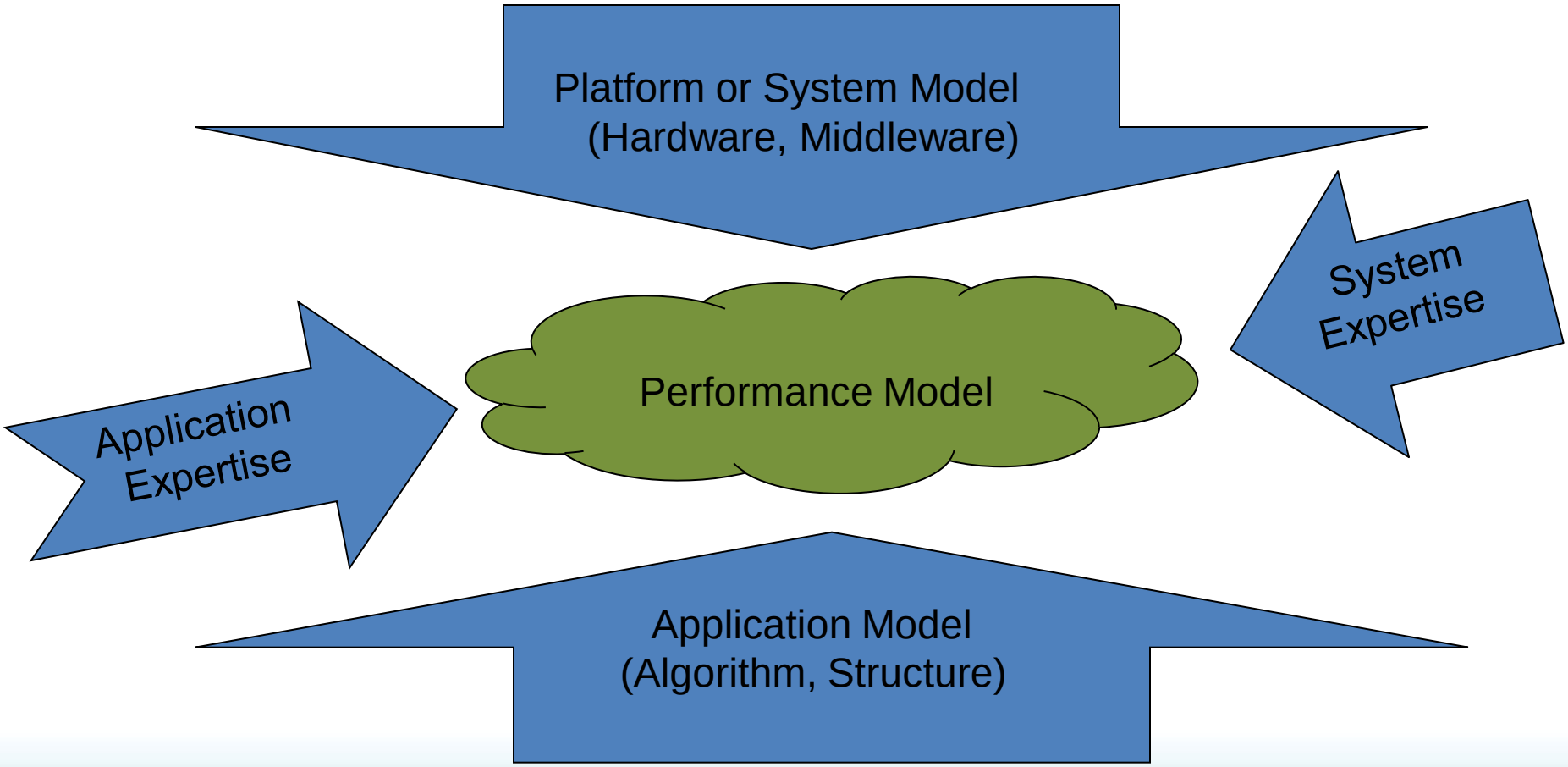
What is Performance Modeling?

- Understand the resource usage of an application on a particular architecture
  - We focus mostly on time as a resource
  - Generate analytic expressions to estimate runtime
- Closely related to “Performance Engineering”
  - Often builds on empirical techniques
- Also cutting into complexity theory
  - More pragmatic (asymptotes often insufficient)
  - Complex (low-order terms cannot be dropped)

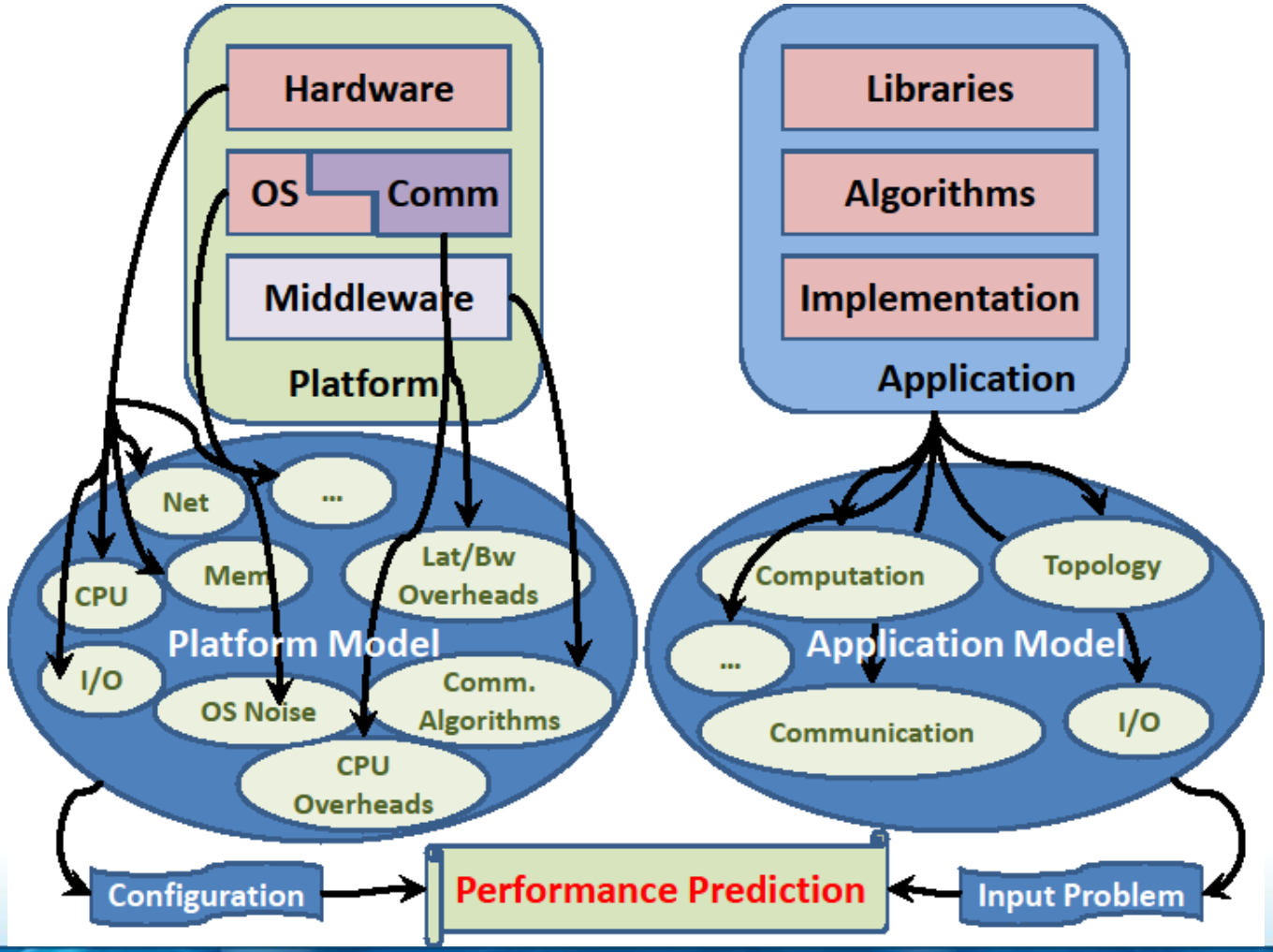
Performance Modeling and Productivity?

- Predict resources and costs to solve a particular problem instance
- Evaluate effectiveness of a computing platform to solve a particular problem
- Understand bottlenecks, i.e., HW/SW Co-Design
- Find performance bugs and assess upper and lower bounds of code optimizations
- Makes programmers **think** about the **structured performance profile** of an application or platform

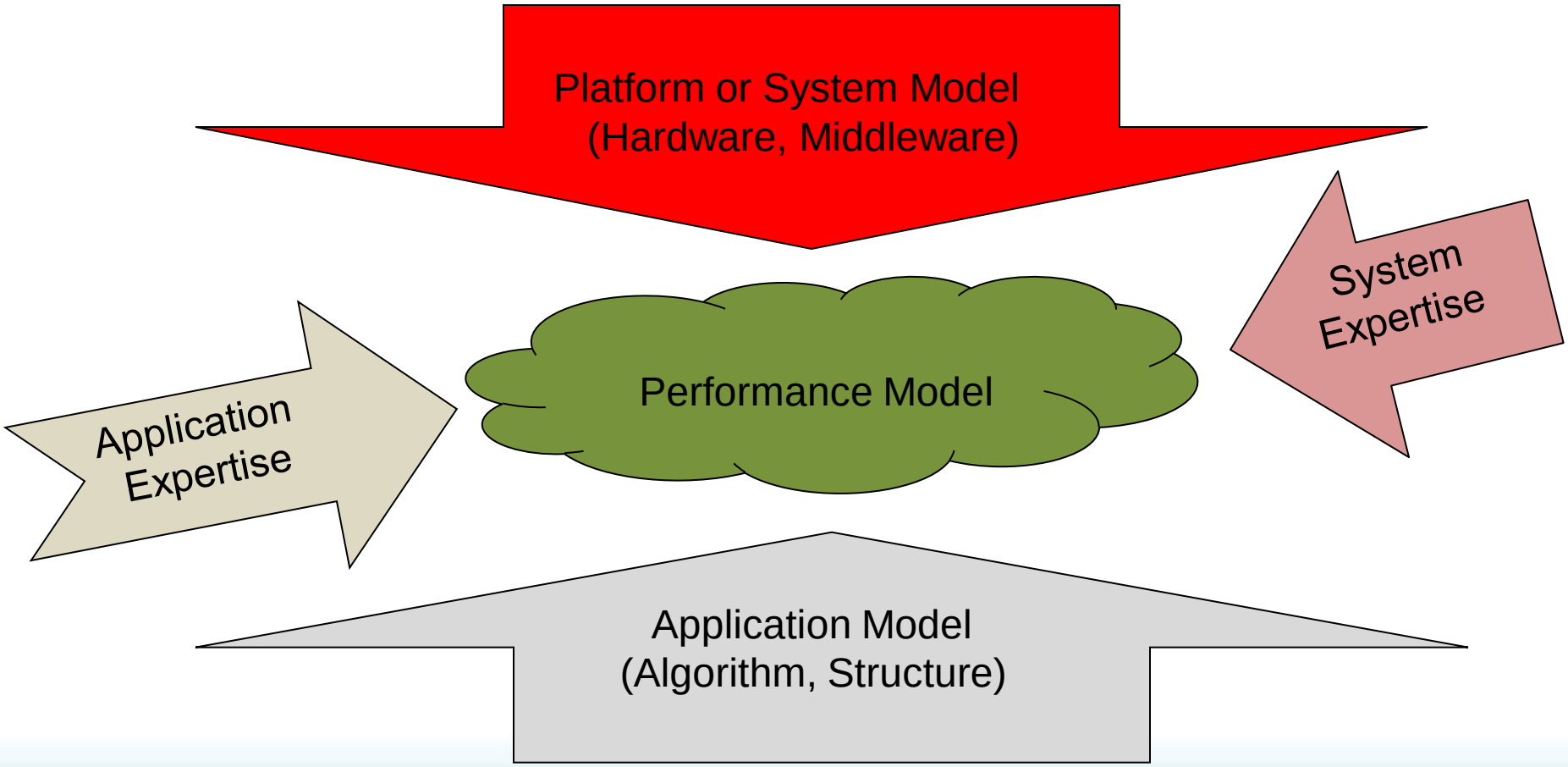
Performance Modeling from 10.000 Feet



Of course it's more complex than that



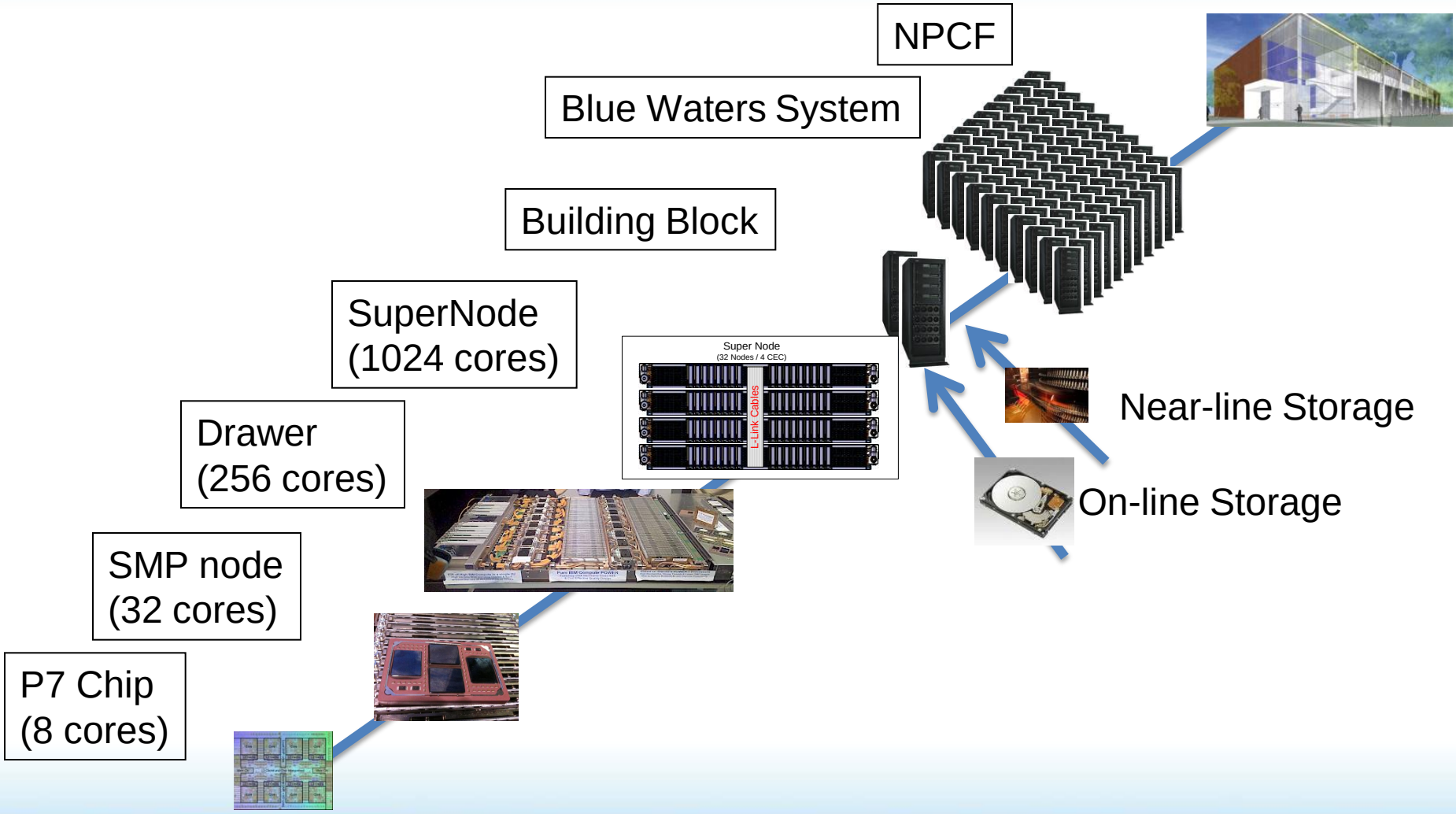
How to Derive a Platform Model



Example: Blue Waters - Overview

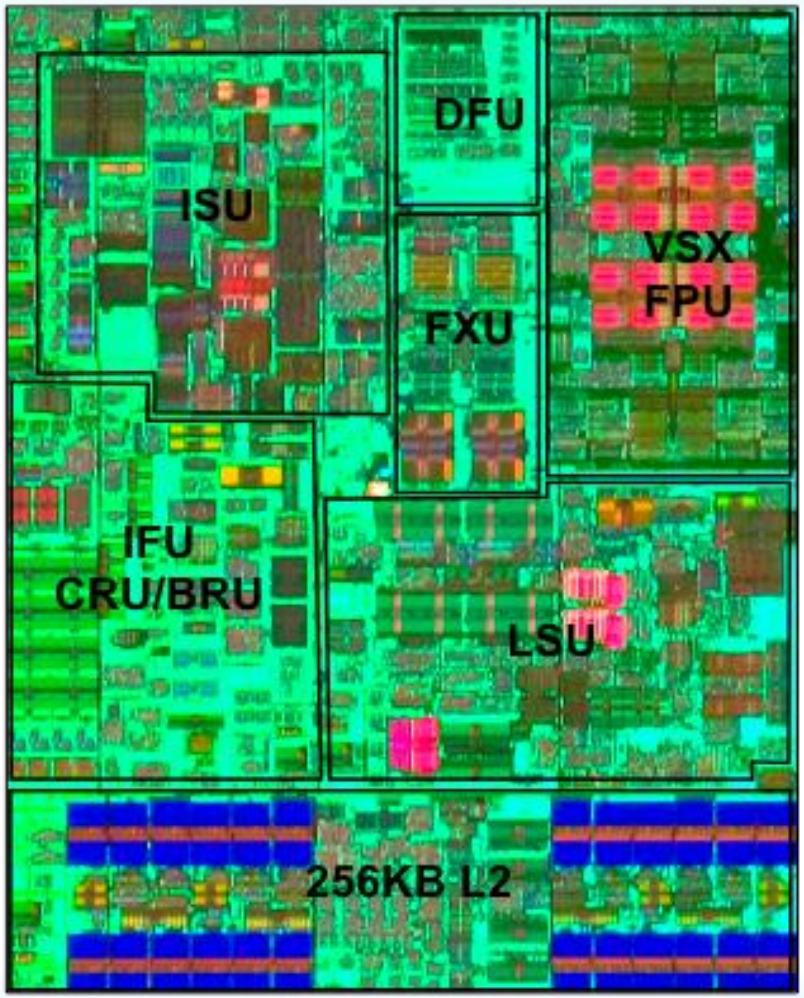
- >300.000 compute cores
  - Based on POWER7
- 10 PF/s peak (not important, more later)
  - 1 PF/s sustained
- Direct network with 1.1 TB/s per hub
  - Total approx. 10 PB/s
- >1.2 PB main memory
- >18 PB on-line disk storage
- >500 PB near-line storage

From Chip to Entire Integrated System



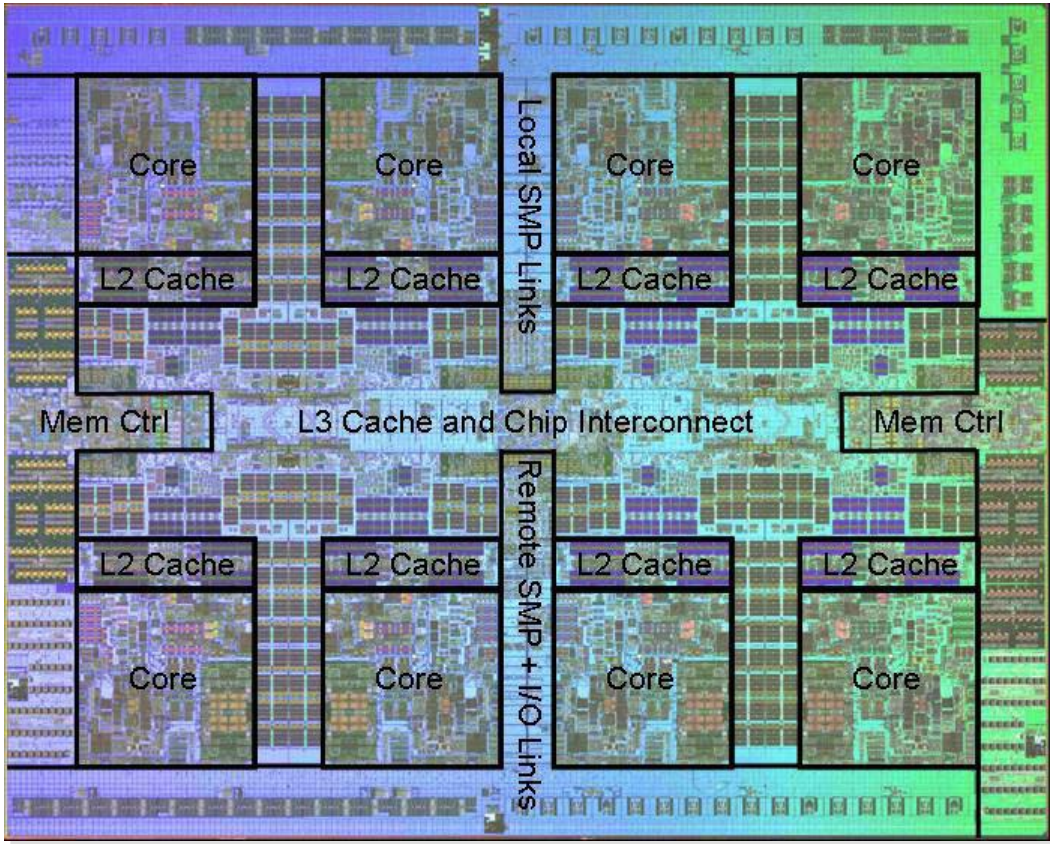
POWER7: Core

- Execution Units
  - 2 Fixed point units
  - 2 Load store units
  - 4 Double precision floating point
  - 1 Branch
  - 1 Condition register
  - 1 Vector unit
  - 1 Decimal floating point unit
  - 6 wide dispatch
- Recovery Function Distributed
- 1,2,4 Way SMT Support
- Out of Order Execution
- 32KB I-Cache
- 32KB D-Cache
- 256KB L2
  - Tightly coupled to core



POWER7 Chip (8 cores)

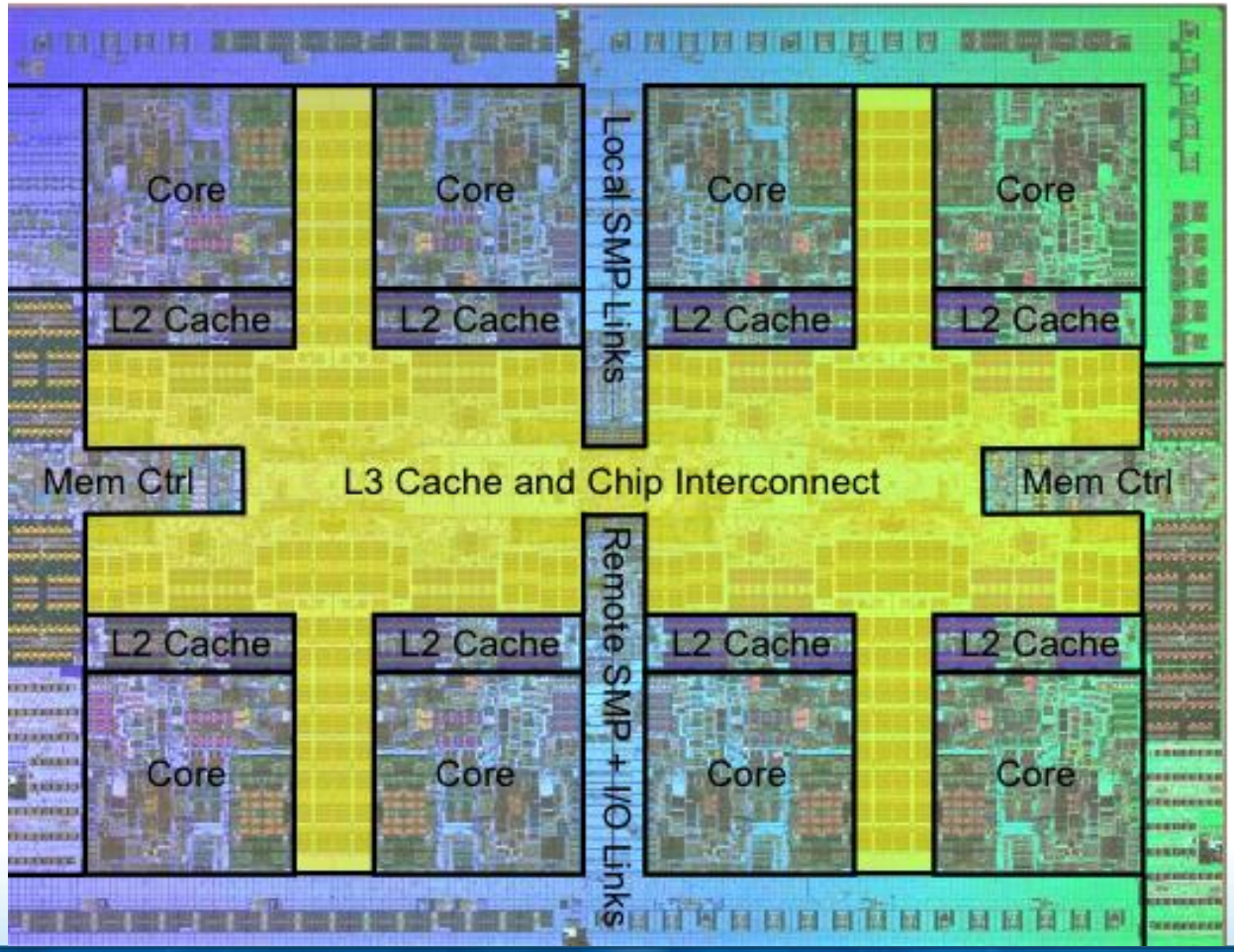
- Base Technology
  - 45 nm, 576 mm^2
  - 1.2 B transistors
- Chip
  - 8 cores
  - 4 FMAs/cycle/core
  - 32 MB L3 (private/shared)
  - Dual DDR3 memory
    - 128 GiB/s peak bandwidth
    - (1/2 byte/flop)
  - Clock range of 3.5 – 4 GHz



Quad-chip MCM

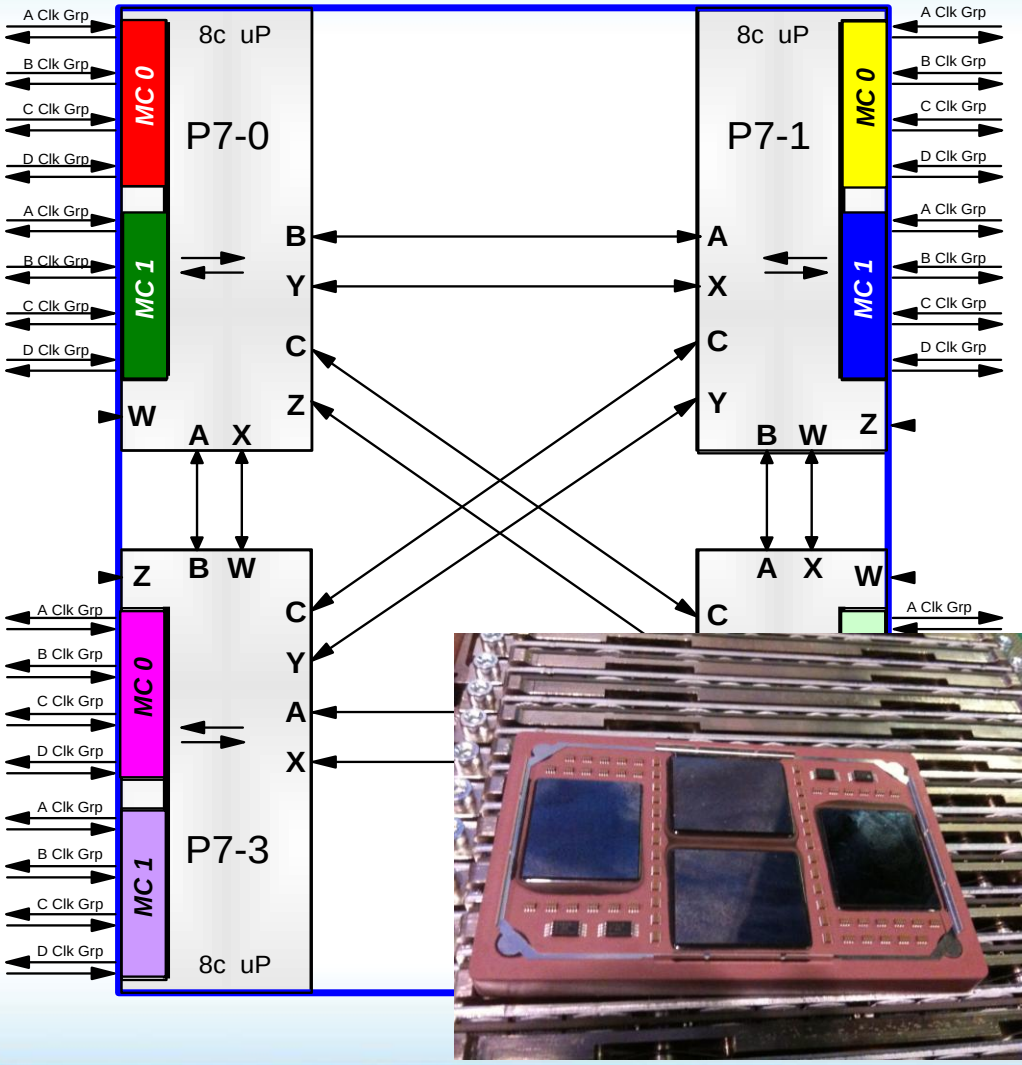
L3 Cache/On-Chip Communication

- L1 32KB Instruction / core
- L1 32KB Data / core
- L2 = 256KB / core
- L3 = 4MB eDRAM / core
- Fast private and shared region



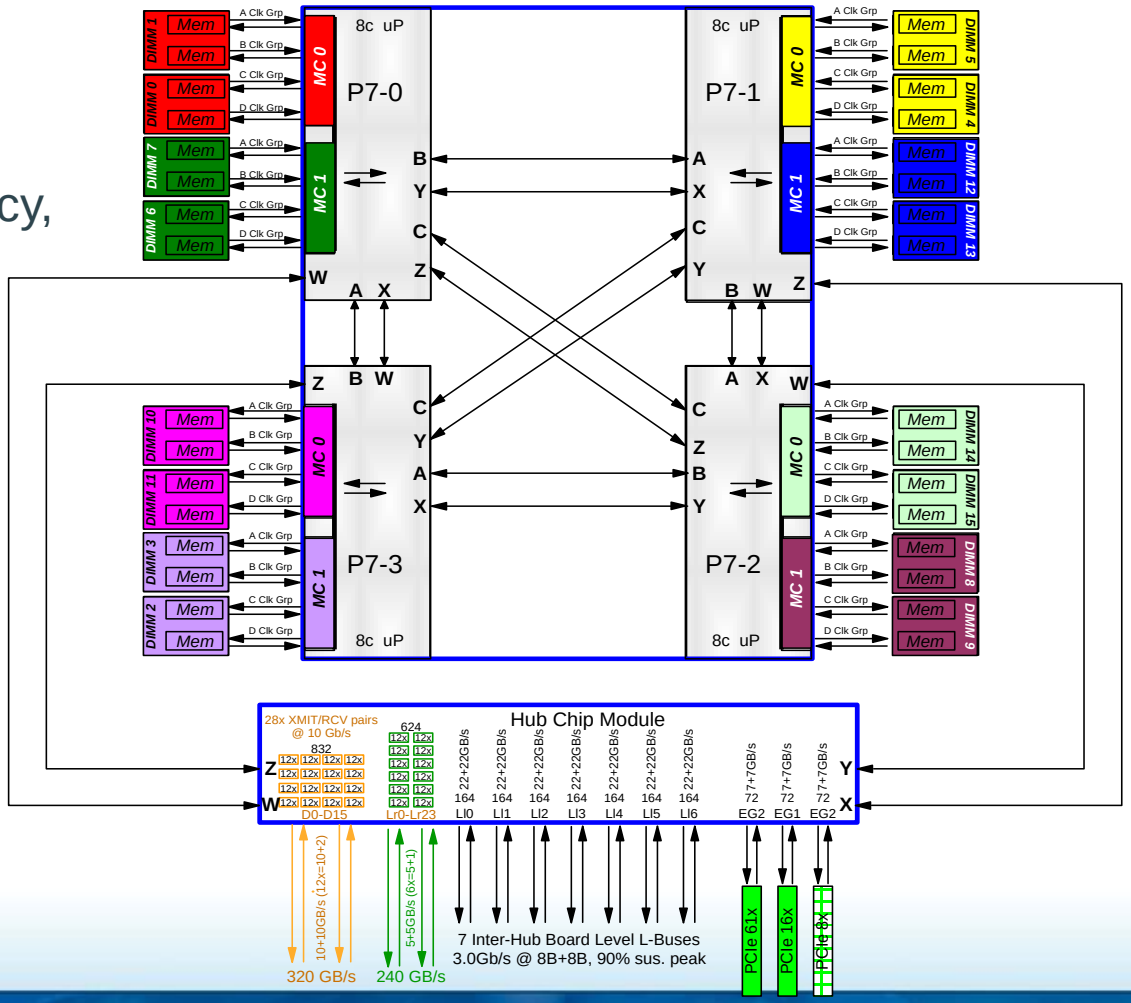
Quad Chip Module (4 chips)

- 32 cores
  - 32 cores*8 F/core*4 GHz = 1 TF
- 4 threads per core (max)
  - 128 threads per package
- 4x32 MiB L3 cache
  - 512 GB/s RAM BW (0.5 B/F)
- 800 W (0.8 W/F)



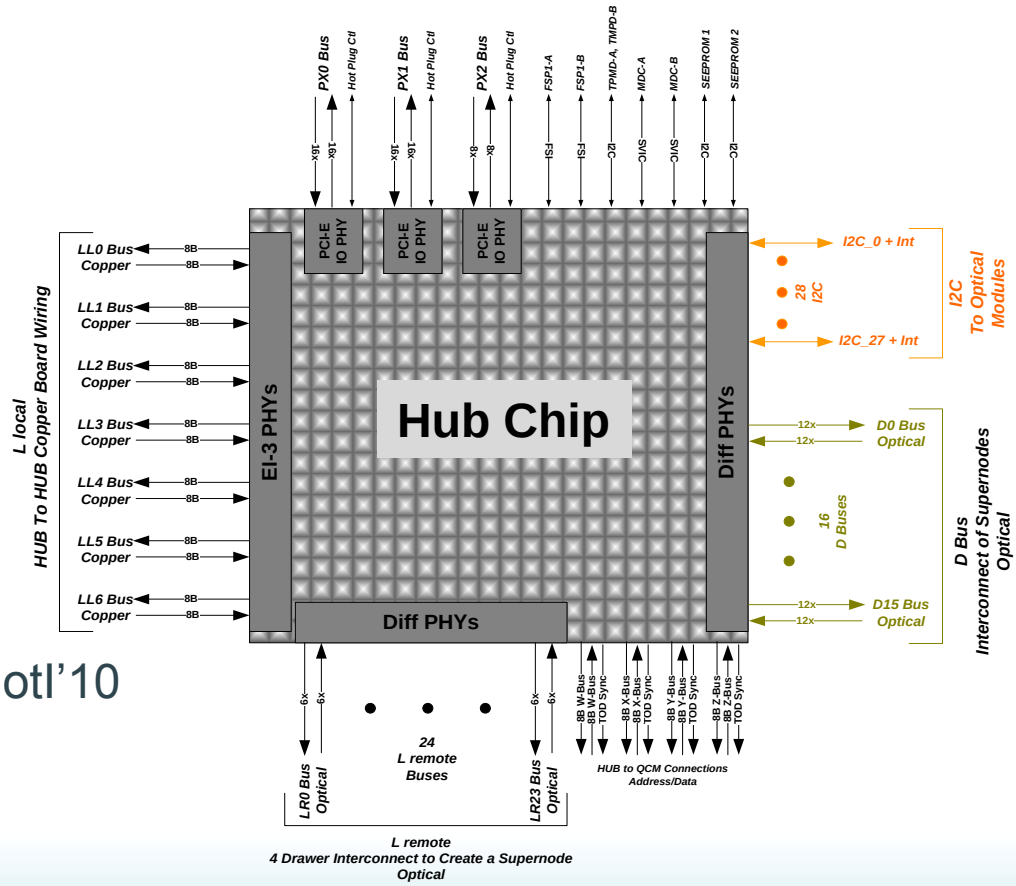
Adding a Network Interface (Hub)

- Connects QCM to PCI-e
  - Two 16x and one 8x PCI-e slot
- Connects 8 QCM's via low latency, high bandwidth, copper fabric.
  - Provides a message passing mechanism with very high bandwidth
  - Provides the lowest possible latency between 8 QCM's



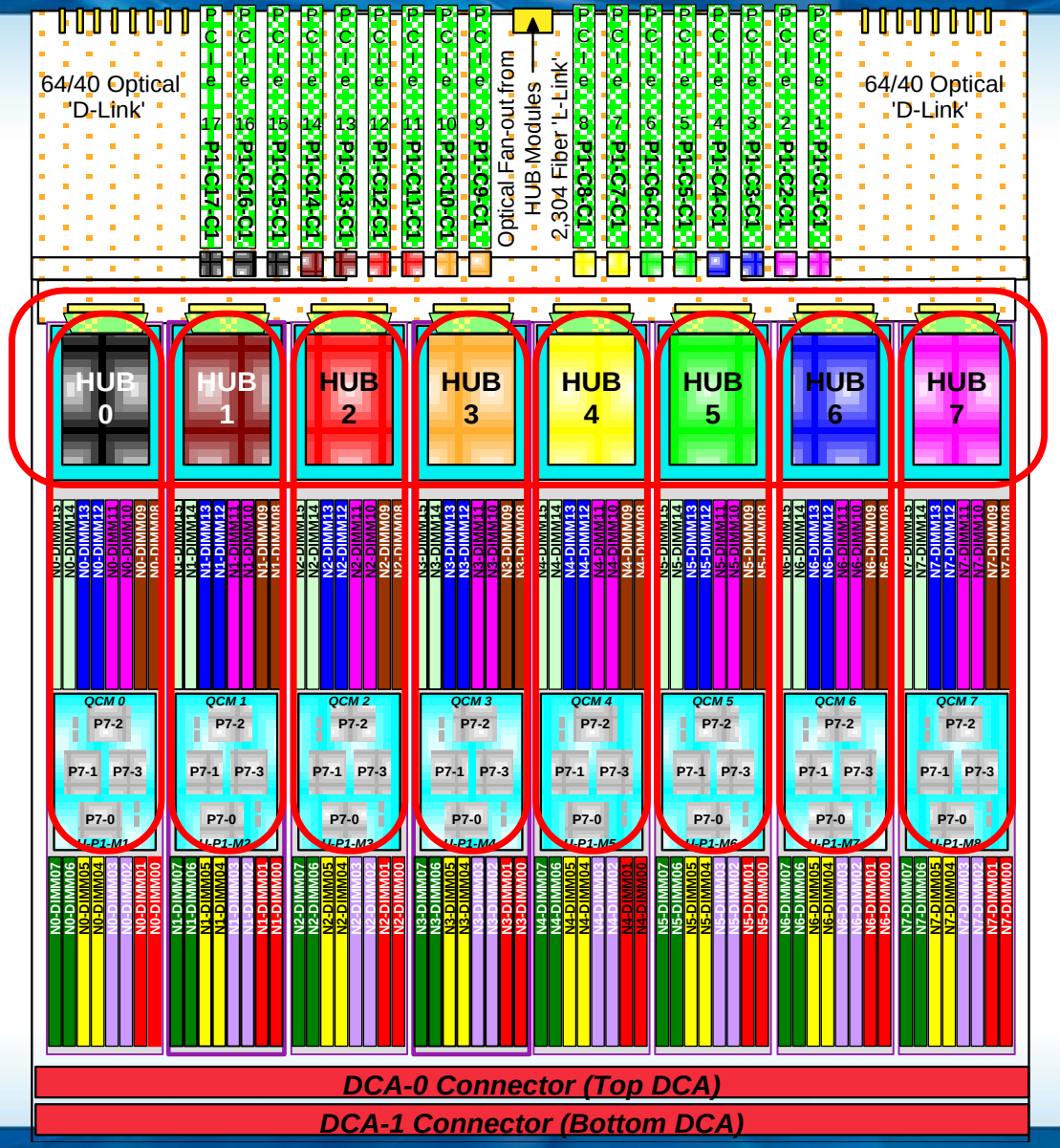
1.1 TB/s HUB

- 192 GB/s Host Connection
- 336 GB/s to 7 other local nodes
- 240 GB/s to local-remote nodes
- 320 GB/s to remote nodes
- 40 GB/s to general purpose I/O
- cf. “The PERCS interconnect” @HotI’10



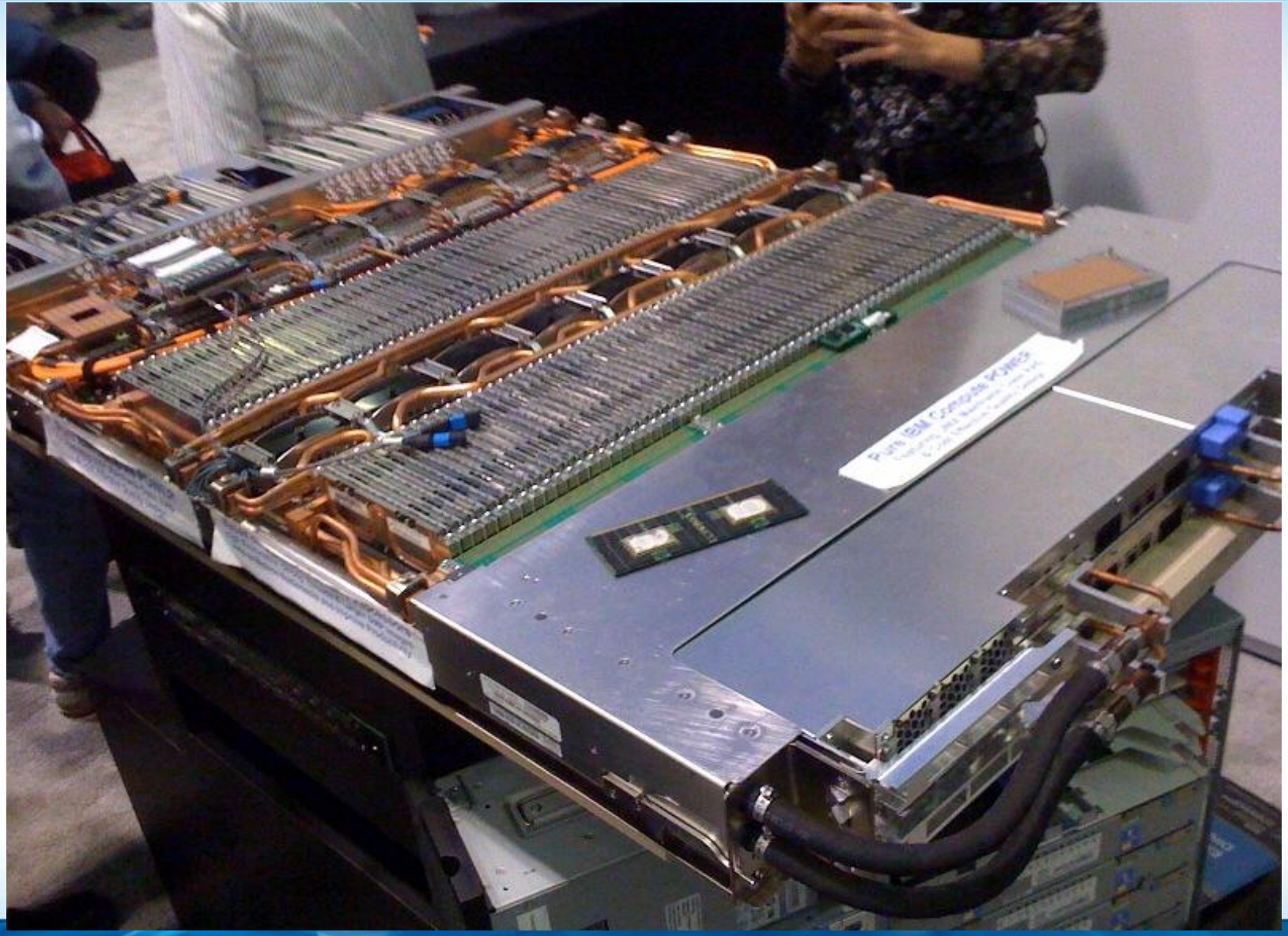
Drawer

- 8 nodes
  - 32 chips
  - 256 cores



First Level Interconnect

- L-Local
- HUB to HUB Copper Wiring
- 256 Cores

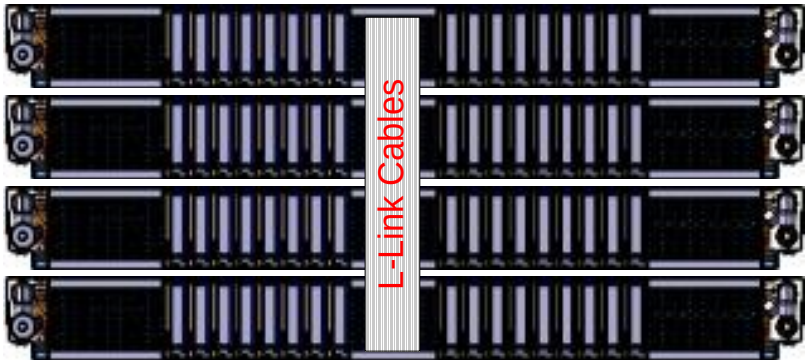


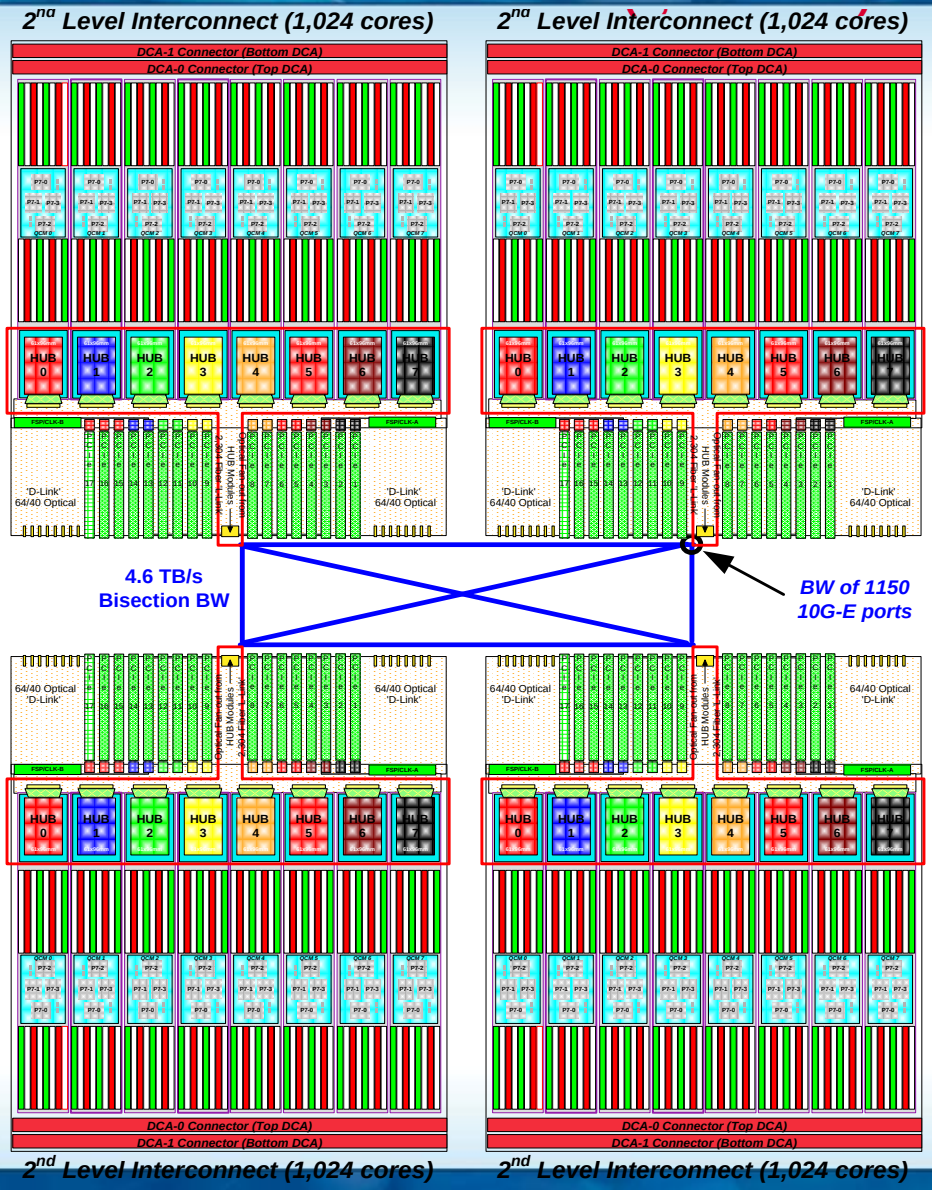
Supernode

Second Level Interconnect

- Optical 'L-Remote' Links from HUB
- 4 drawers
- 1,024 Cores

Super Node
(32 Nodes / 4 CEC)





Source: IBM

National Petascale Computing Facility

Photo: NCSA

Designing a Platform Model

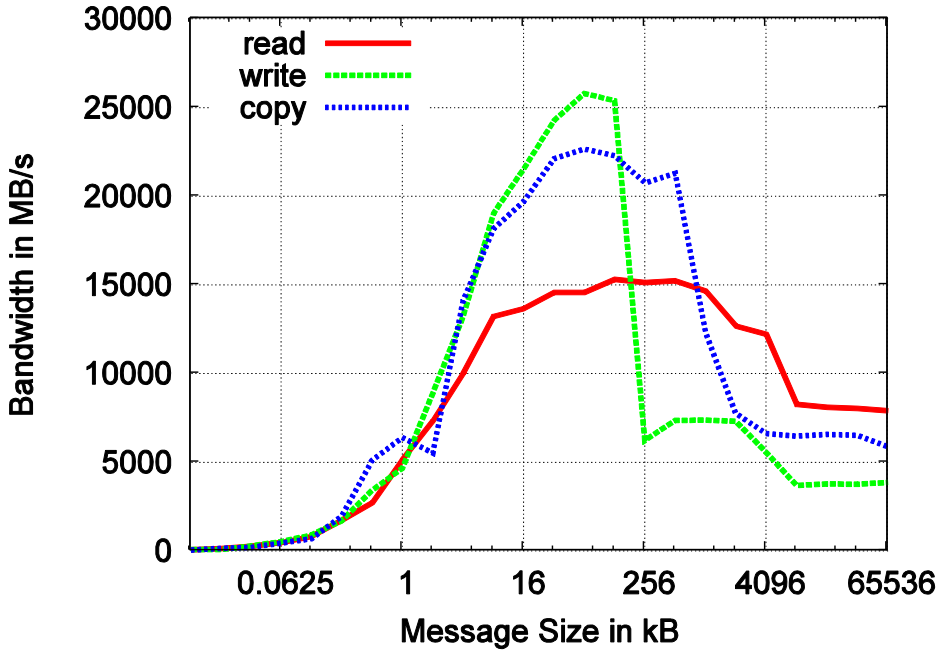
- Key Platform parameters:
  - Memory
  - CPU and Caches
  - Network
  - I/O subsystem (e.g., disk)
  - OS (e.g., noise)
  - Middleware Parameters (e.g., MPI)

Determine a Memory Model

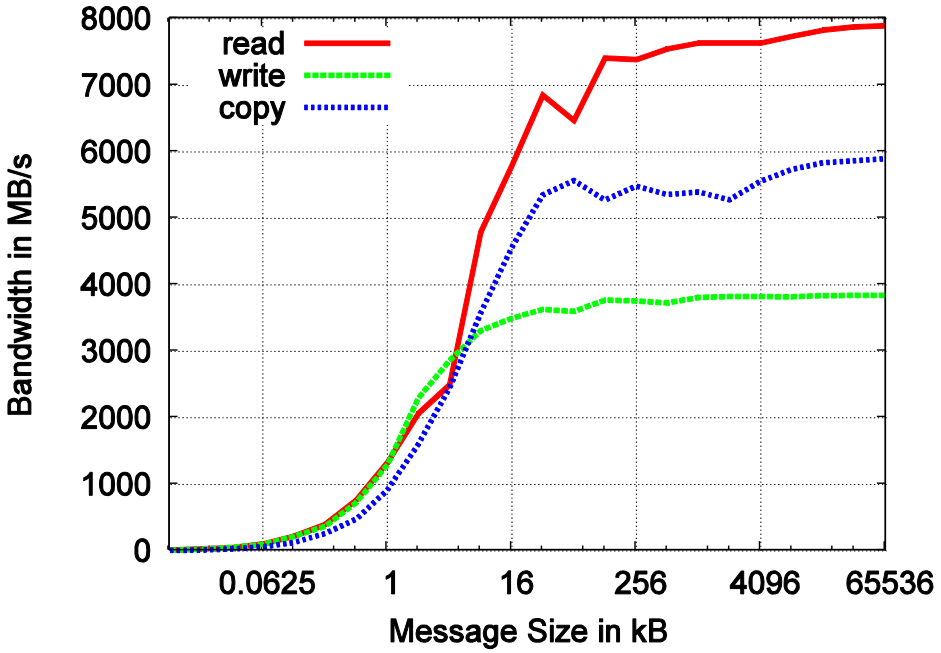
- Memory parameters
  - Latency – time to access a single element (cf. L)
  - Access Rate – per-element overhead (cf. g)
  - Bandwidth - linear access time (cf. G)
- Netgauge memory benchmark
  - Random pointer-chase to measure latency
  - Bulk random access to measure access rate (cf. GUPS)
  - Bulk linear access to measure bandwidth (cf. stream)

Single Core Memory on the IBM 780 (P7 MR)

512 MB (8 Byte) random read: 143 MB/s, write: 320 MB/s, copy: 121 MB/s
latency: 132 ns



cached



uncached

compiled with gcc 4.3.4

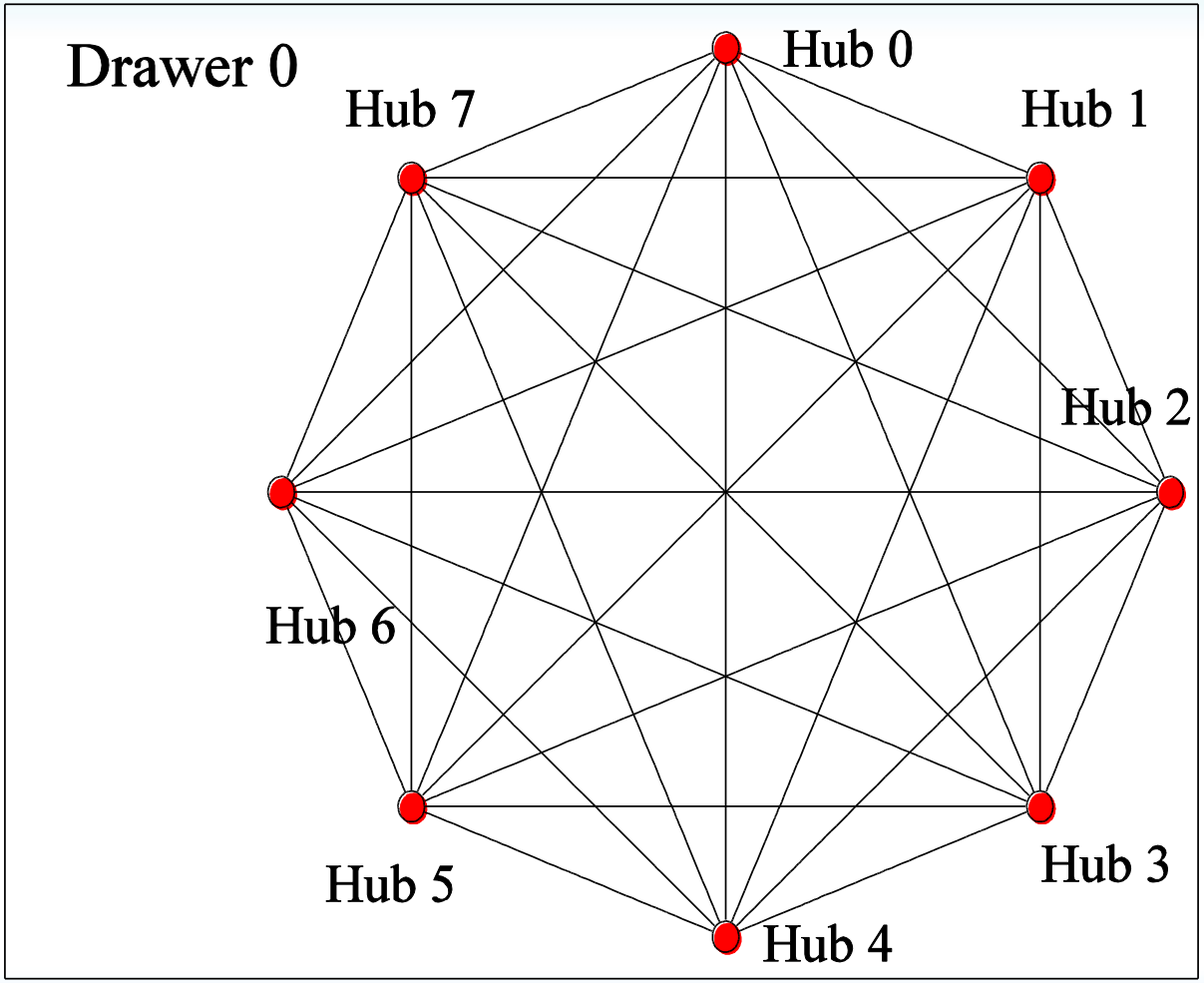
Single Core Modeling Strategy

- Modeling performance of modern CPUs is well understood but very complex
  - Upper bounds are (among others) FLOP-rate and memory performance
  - Details are not too important as we mostly care about scalability
  - Semi-empirical modeling: run application benchmarks to determine single-core models
  - Simulation: simulate CPU performance (Mambo)

Determining a Network Model

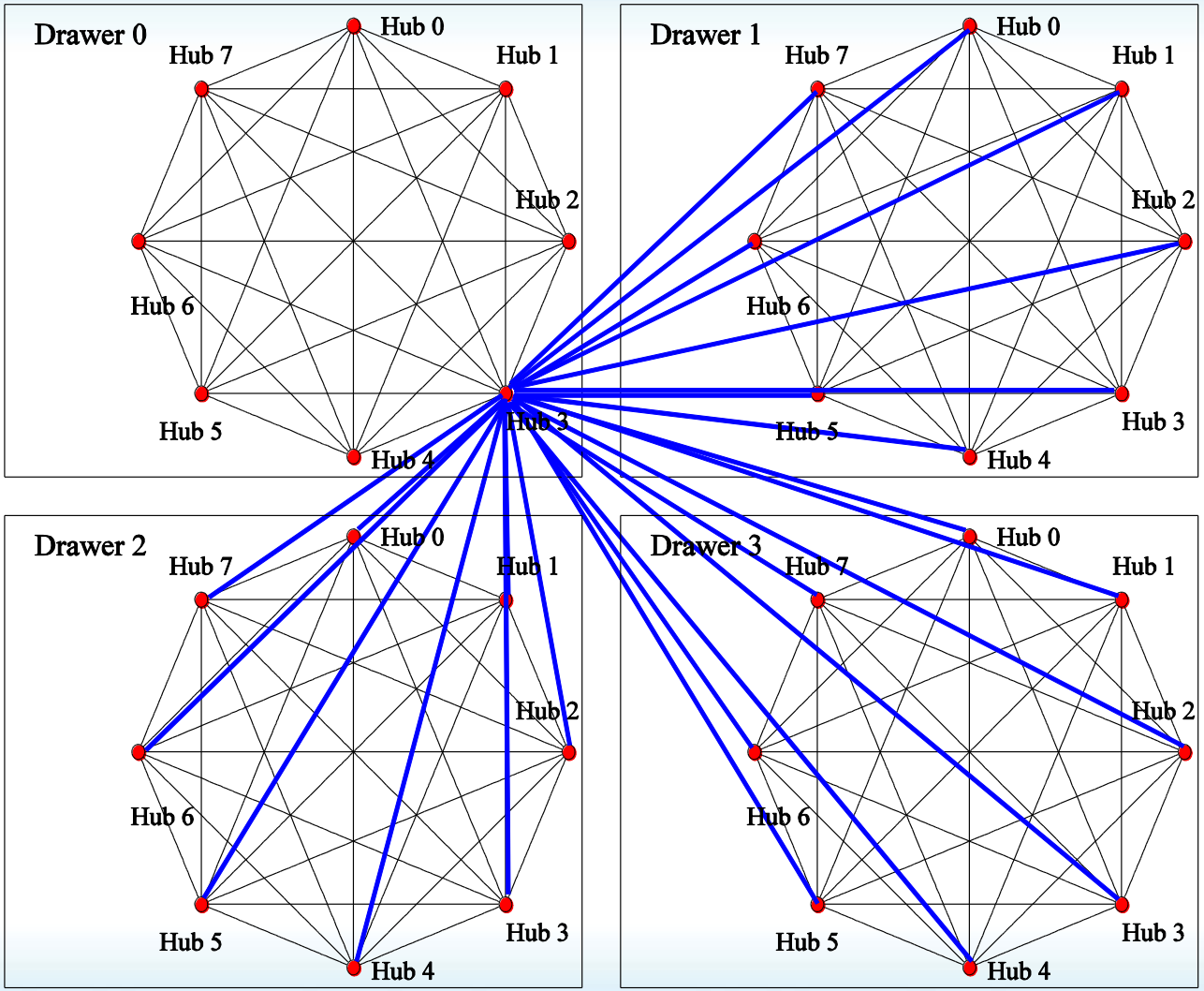
- The network is most important
  - Becomes dominant at scale
  - We cannot simply “run” benchmarks to determine a model as in the single-core case (too expensive)
- Parameters:
  - Topology (bisection bandwidth)
  - Collective parameters (e.g., Alltoall bandwidth)
  - Point-to-point parameters (e.g., LogGP)
    - latency, CPU overhead, injection rate/gap, bandwidth

- LL Topology
  - 24 GB/s
  - 7 links/Hub
  - Fully connected
  - 8 Hubs



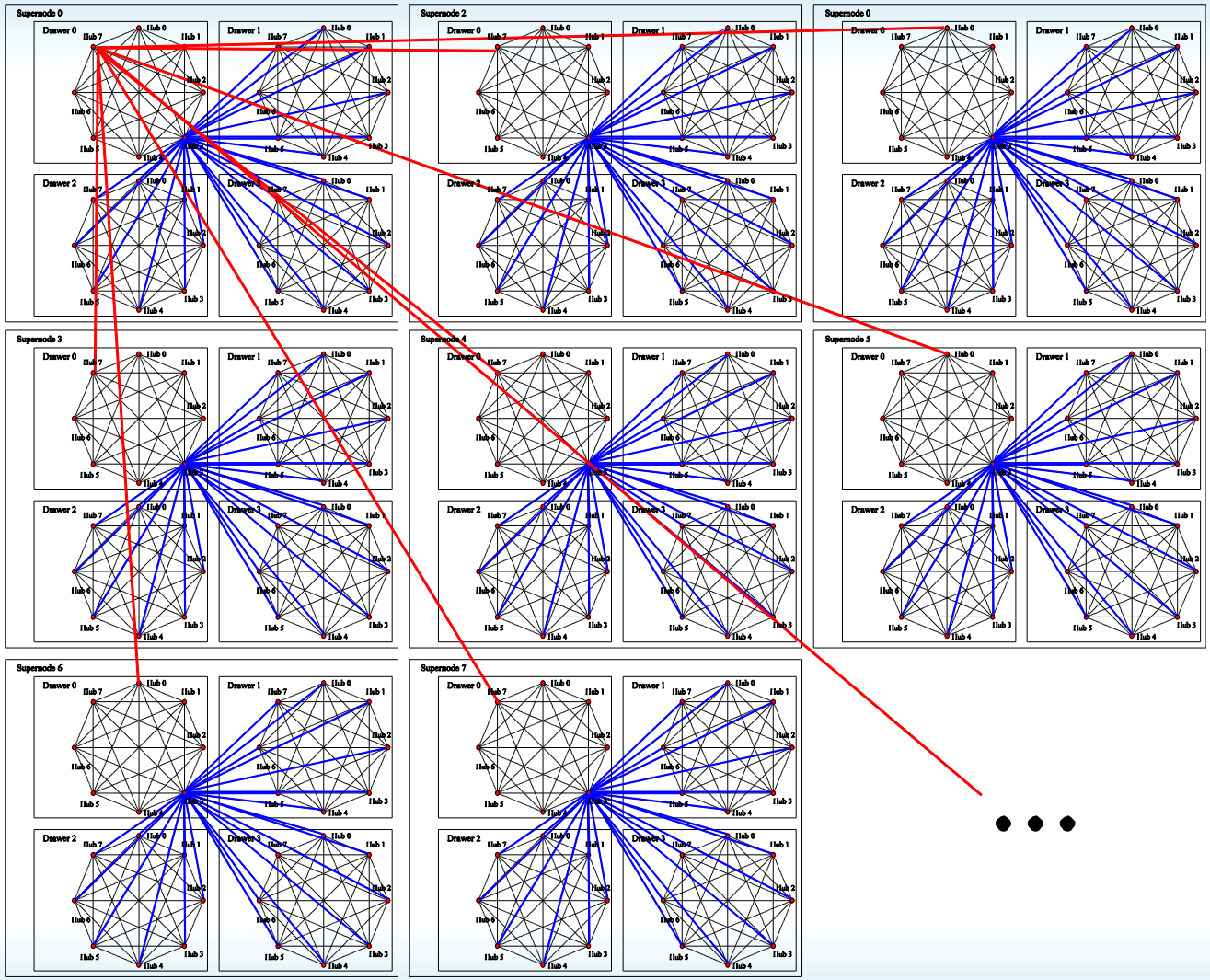
Source: B. Arimilli et al. *"The PERCS High-Performance Interconnect"*

- LR Topology
  - 5 GB/s
  - 24 links/Hub
  - Fully connected
  - 4 Drawers
  - 32 Hubs



Source: B. Arimilli et al. *"The PERCS High-Performance Interconnect"*

- D Topology
  - 10 GB/s
  - 16 links/Hub
  - Fully connected
  - 512 SNs
  - 2048 Drawers
  - 16384 Hubs

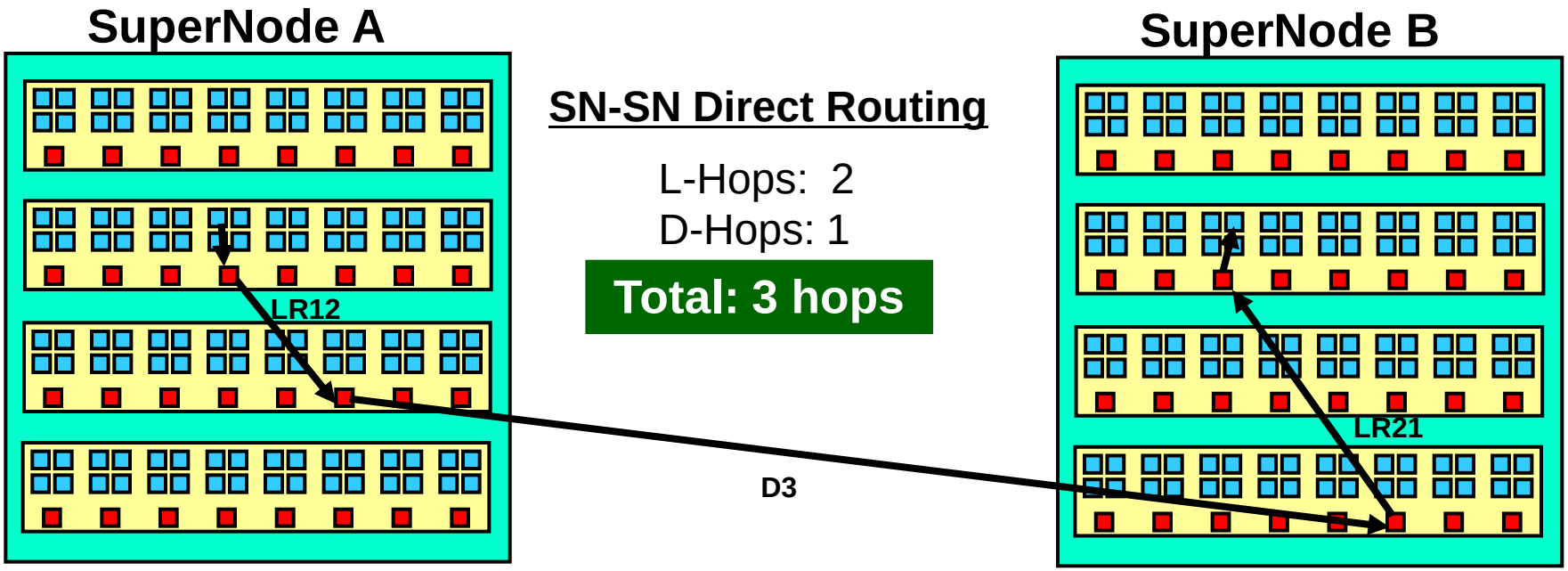


Source: B. Arimilli et al. *"The PERCS High-Performance Interconnect"*

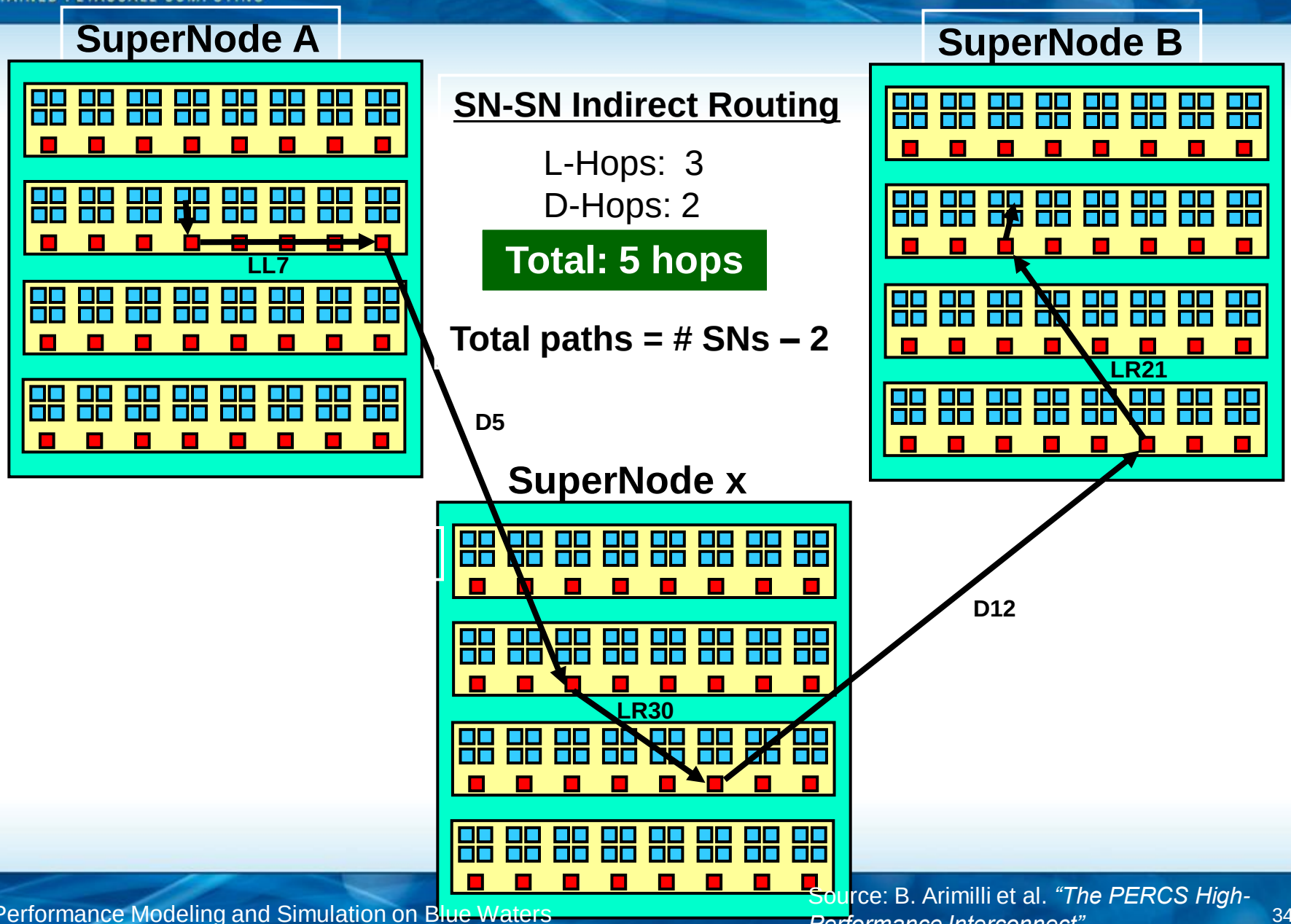
Routing

- Deterministic routing
  - Hits Borodin-Hopcroft bound ($c = \Theta(\sqrt{P})$)
  - Bad worst-case performance

- Indirect (random) routing
  - Increases latency
  - Effectively halves (global) bandwidth
  - Worst-case becomes very unlikely



Source: B. Arimilli et al. *"The PERCS High-Performance Interconnect"*



SN-SN Indirect Routing

L-Hops: 3
D-Hops: 2

Total: 5 hops

Total paths = # SNs – 2

Source: B. Arimilli et al. *"The PERCS High-Performance Interconnect"*

Example Model: BW Global Alltoall Bandwidth

- Assume all communications happen simultaneously
- Derive upper (expected) bound for direct routing
  - Simple counting argument
- Each CN can be reached through series of LL, LR, D
  - Not more than one D link or LL-LR, LR-LL, LL-LL, LR-LR
- Denote e(P) as number of nodes reachable through path P
  - e(LL) =7, e(LR)=24, e(D)=d (variable number of D-links)
- # nodes reachable in two hops:
  - e(LL-D)=e(D-LL)=7d
  - e(LR-D)=e(D-LR) = 24d

Source: B. Arimilli et al. *"The PERCS High-Performance Interconnect"*

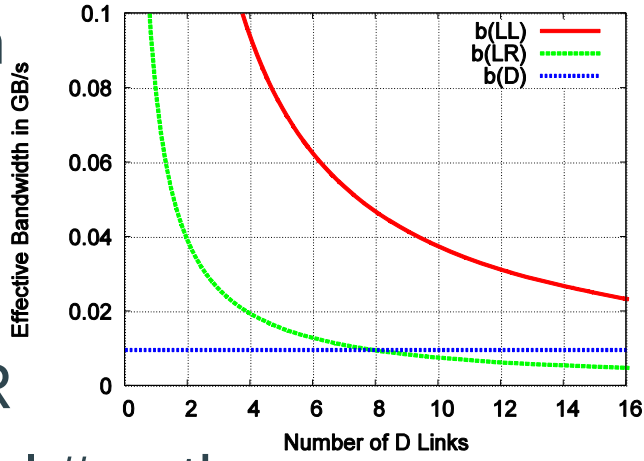
Example: Alltoall Bandwidth Continued …

- # nodes reachable in three hops:
  - e(LL-D-LL) = 49d
  - e(LL-D-LR)=e(LR-D-LL) = 168d
  - e(LR-D-LR)=576d
- Number of paths from each source: 31+1024d
- Now count the number of paths (i.e., congestion) through each LL, LR, D link c(L): (omitted uninteresting parts of summation)
  - c(LL) = 1+64d
  - c(LR) = 1+64d
  - c(D) = 1024

Source: B. Arimilli et al. *"The PERCS High-Performance Interconnect"*

Example: Alltoall Bandwidth Continued …

- Effective (expected) bandwidths for each link-type:
  - bandwidth = link bandwidth / congestion
  - b(LL) = 24 GB/s / (1+64d)
  - b(LR) = 5 GB/s / (1+64d)
  - b(D) = 10 GB/s / 1024
- If d<8, D is the bottleneck, otherwise LR
- Bandwidth per PE: 5 GB/s / 1+64d * total # paths
  - d=9 (294912 cores): 80.13 GB/s
  - d=10 (327680 cores): 80.11 GB/s



total ~ 0.8 PB/s ☺

- Connection to P7 chips: 4*24 GB/s = 96 GB/s ~ 83% ☺

Source: B. Arimilli et al. *"The PERCS High-Performance Interconnect"*

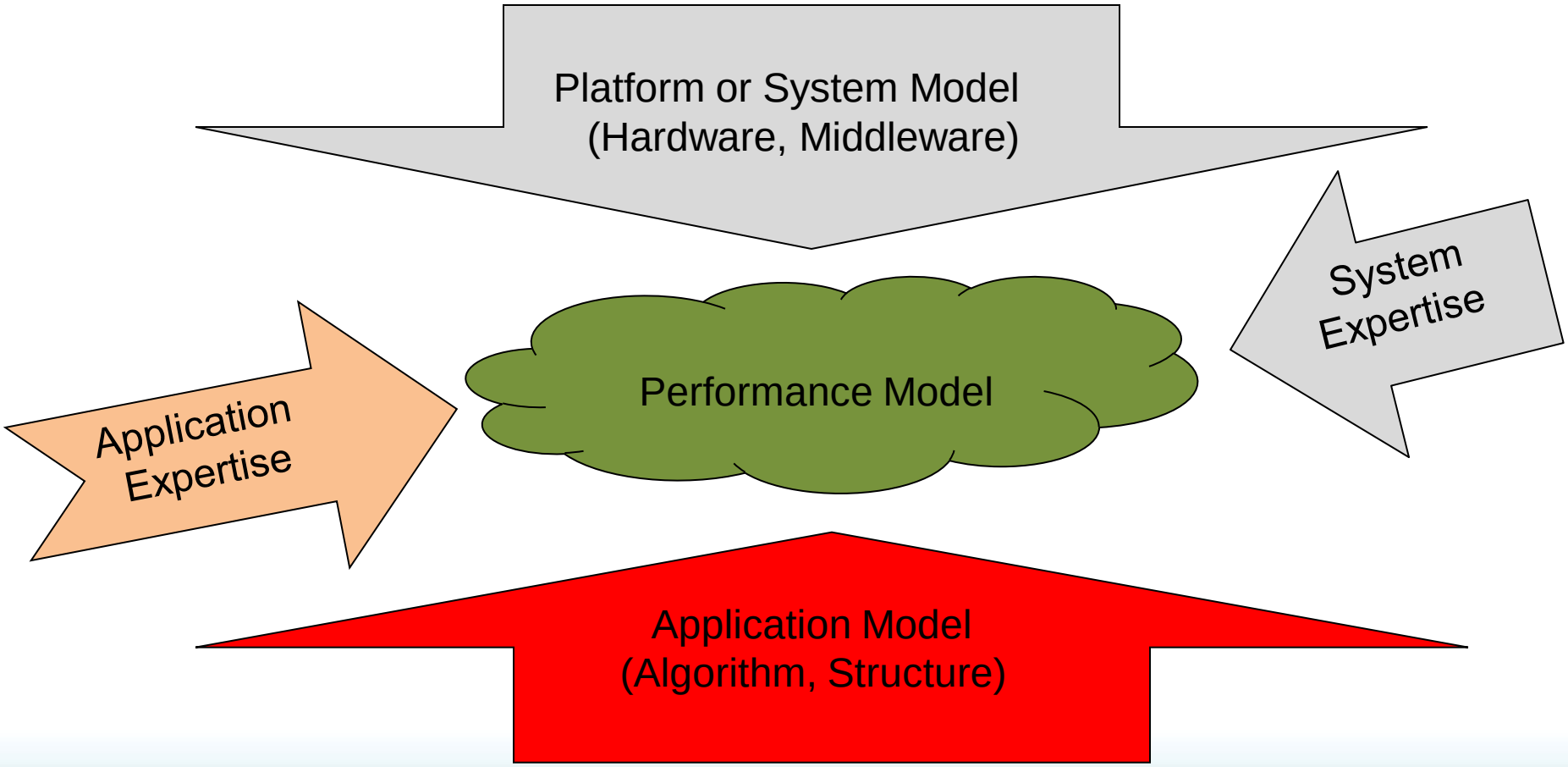
Influence of the Routing Strategy

- Indirect routing would effectively half the bandwidth
  - First route to intermediate supernodes
- Not desirable for global Alltoall case
  - Other patterns could benefit
  - Example: bad pattern for direct routing:
    - all nodes in SN A send to all nodes in SN B
    - congestion of 32, i.e., 312.5 MB/s
    - (optimal) indirect routing would improve to 5 GB/s
- Detailed routing strategy is still under discussion
  - May be influenced by outcome of application modeling

Complete Network Models

- LogGP parameters can be measured (Netgauge)
  - Or estimated based on documentation
- Traffic patterns need to be modeled or simulated
  - Performance depends on task-to-node mapping
  - Significant effort but only done once per machine
  - NCSA is working on a detailed system model of Blue Waters

How to Derive an Application Model



Application Models

- Are often much more complex than platform models
- Serial performance models:
  - CPU + Cache requirements
  - Memory requirements
  - I/O requirements
- Parallel performance models:
  - Amdahl's law (scaling)
  - Communication overhead
  - Communication/computation overlap

Rough Application Classification

Static →	Adaptive →	Dynamic
fixed control- and dataflow	fixed control-flow and variable dataflow	variable control- and dataflow
e.g., structured n-dimensional meshes/tori	e.g., AMR, N-body, unstructured mesh	e.g., high-impact AMR, graph computations
e.g., MILC, WRF, Abinit, Sweep3D	e.g., TDDFT/Octopus, Enzo, ACES III	e.g., PBGL, BFS

easy to model … → **… hard to model**

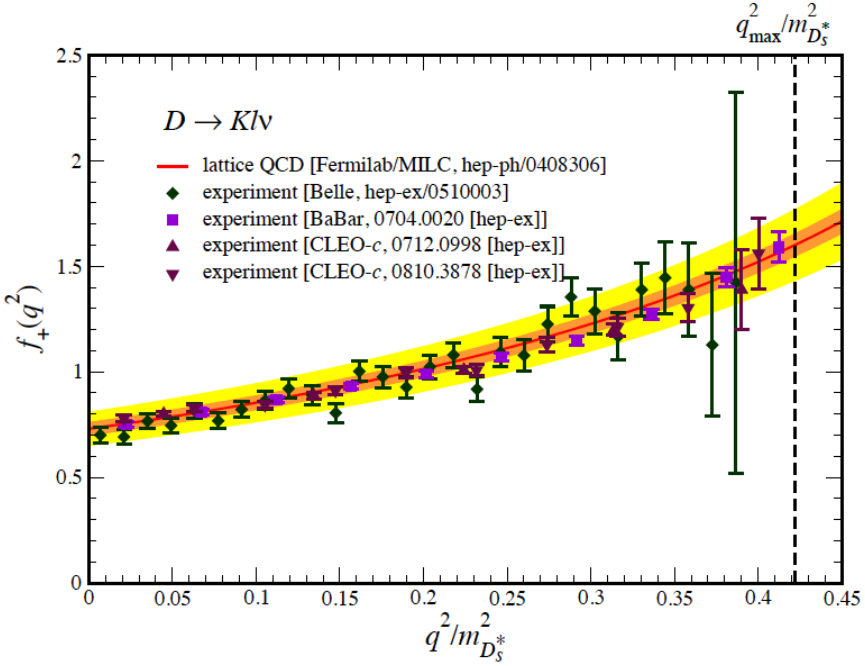
A Strategy for Static Applications

- Two of the three NSF Petascale benchmarks are static
  - Turbulence & Quantum Chromodynamics

- Application modeling in three simple steps:
  1. Identify all performance-critical parameters
  2. Identify code-blocks that share a performance signature
     - derive serial performance model for each
  3. Determine application communication pattern
     - including communication sizes

An Application Modeling Example: MILC

- MIMD Lattice Computation
  - Gains deeper insights in fundamental laws of physics
  - Determine the predictions of lattice field theories (QCD & Beyond Standard Model)
  - Major NSF application
- Challenge:
  - High accuracy (computationally intensive) required for comparison with results from experimental programs in high energy & nuclear physics



MILC - Performance-critical Parameters

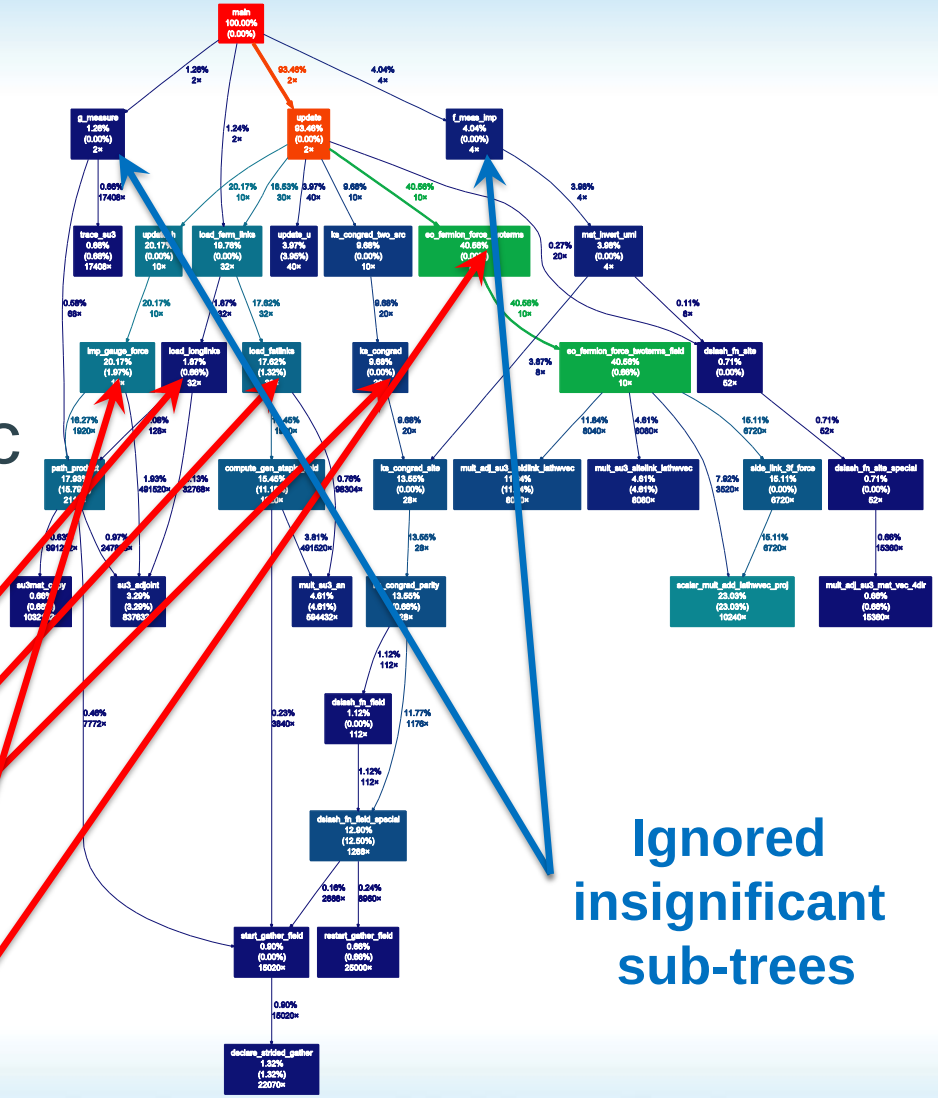
Name	simple	complex	comment
P	X		Number of processes
nx, ny, nz, nt	X		Lattice size in x,y,z,t
warms, trajecs	X		Warmup rounds and trajectories
traj_between_meas	X		Number of “steps” in each trajectory
beta, mass1, mass2, error_for_propagator		X	Physical parameters – influence convergence of conjugate gradient
max_cg_iterations		X	Limits CG iterations per step

- If parameters are more complex (e.g., input files) then the user has to distill them into single values (domain specific)

MILC – Critical Blocks

- Identify sub-trees in call-graph with same performance characteristic
- Five blocks in MILC

Name	Function
LL	load_longlinks
FL	load_fatlinks
CG	ks_congrad
GF	imp_gauge_force
FF	eo_fermion_force



Communication Pattern

- Four-dimensional p2p communication topology
  - Prime-factor decomposition of P (→ square)
- Total number of p2p messages

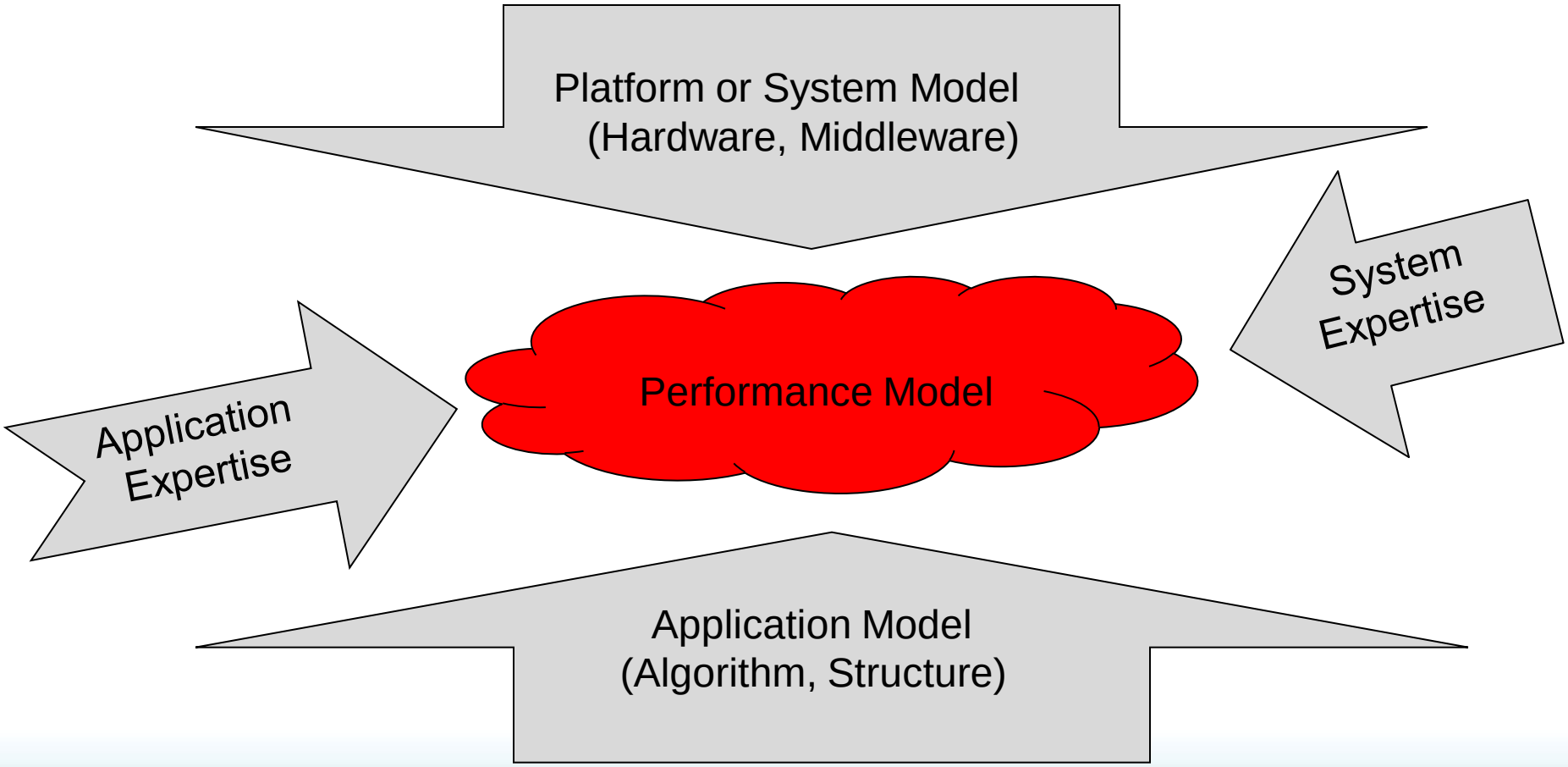
Type	Number of Messages
FF	(trajecs + warms) · steps · 1616
GF	… (for LL, FL, CG)

  - Counted manually (profiling tools and source)
- Collective Communication
  - Single MPI_Allreduce per CG iteration

Ideas for Automation – Tool support

- Model each function as a critical block
  - Automatic decomposition leads to more blocks
  - User needs to provide asymptotic scaling function
    - e.g., T ~ nx*ny*nz*nt,
  - Statistical runs could automatically fit the parameters (can model cache linearly)
  - Similar techniques can be used for message counting
- Should investigate tool support for this!

How to Derive an Application Model



Applying a Machine Model

1. Determine sequential baseline
   - Use single-core application runs as accurate CPU/memory model
2. Process-to-node mapping
   - On-node vs. off-node communication
   - Network congestion
3. Model communication overheads
   - Uses message counts and collective counts

Test system for Model Validation

- BW hardware details are still confidential
- Single Power7 MR is not useful for validation
  - No scaling runs possible

- We use a 1920-core Power5+ system
  - 120 nodes 16 cores per node
  - HPS switch
  - POE/IBM MPI Version 5.1

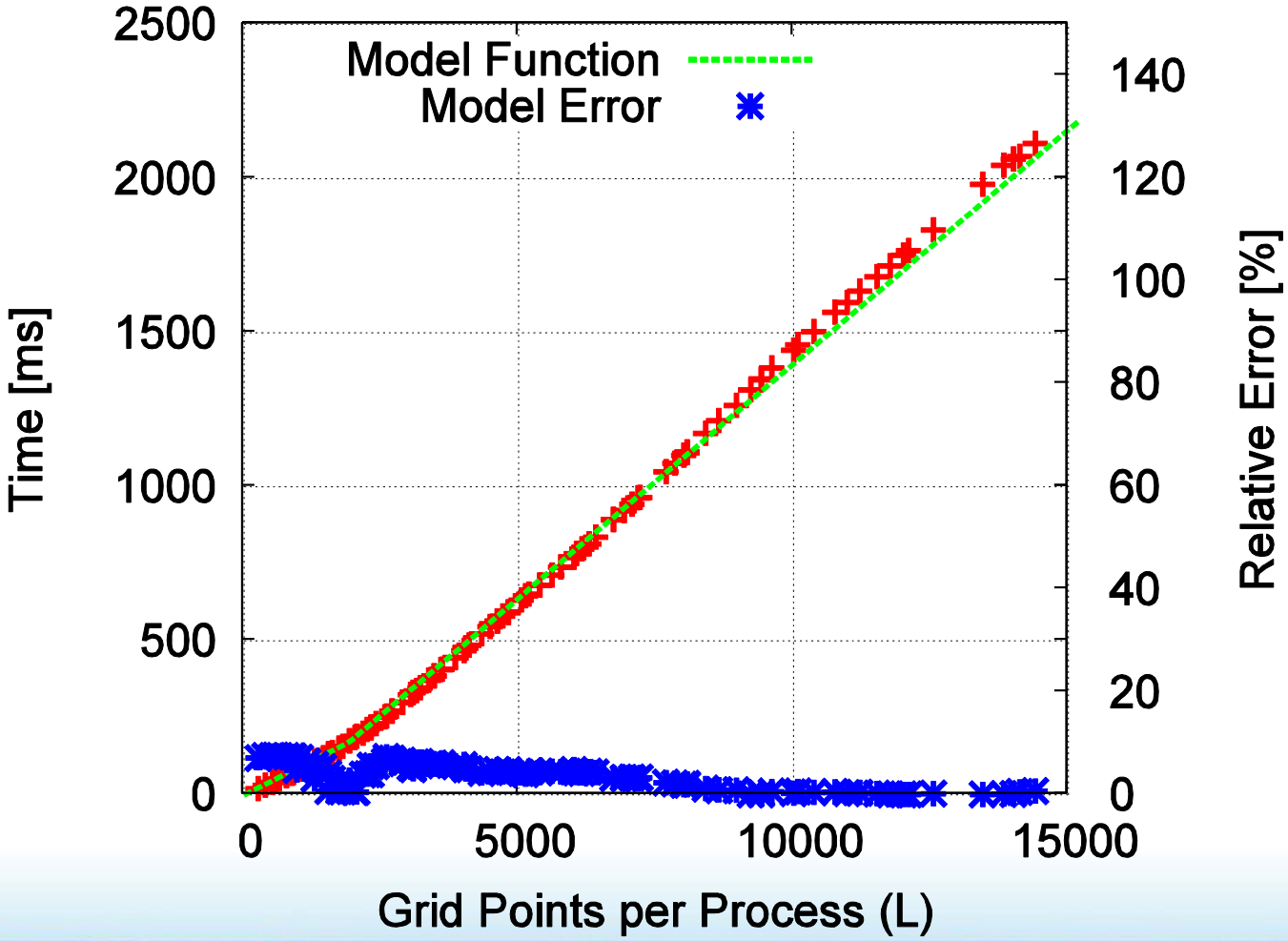
Sequential Baseline

- Stepwise linear function to represent cache influence
  - Chose two steps, different CPUs might need more
  - Volume V = nx*ny*nz*nt; Type B = {LL, FL, GF, CG, FF}
  - Cache holds s(B) data elements

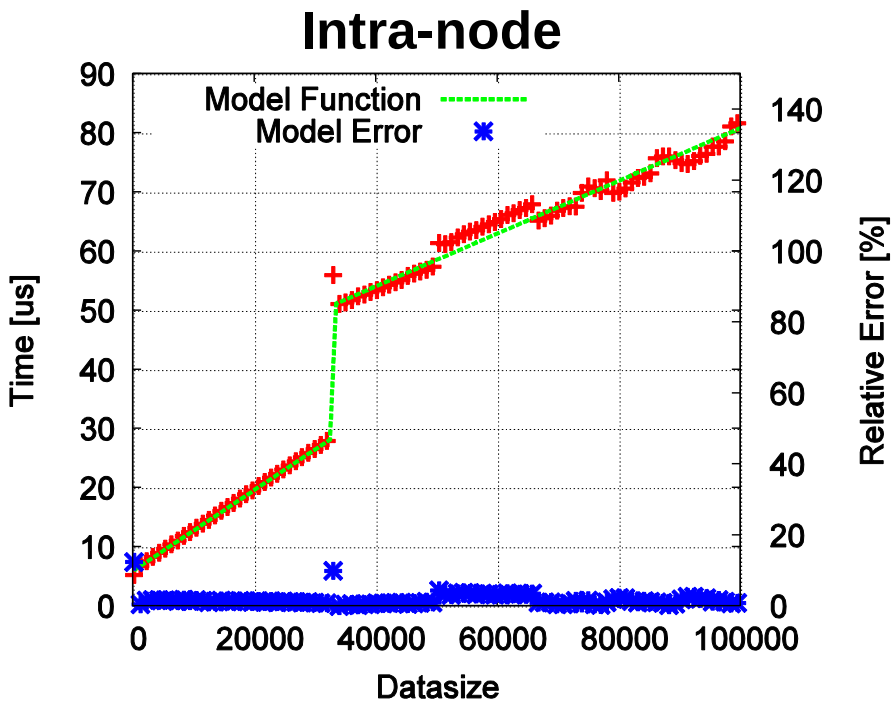
$$T(\mathcal{B}, V) = t_1(\mathcal{B}) \cdot min\{s(\mathcal{B}), V\} + t_2(\mathcal{B}) \cdot max\{0, V - s(\mathcal{B})\}$$

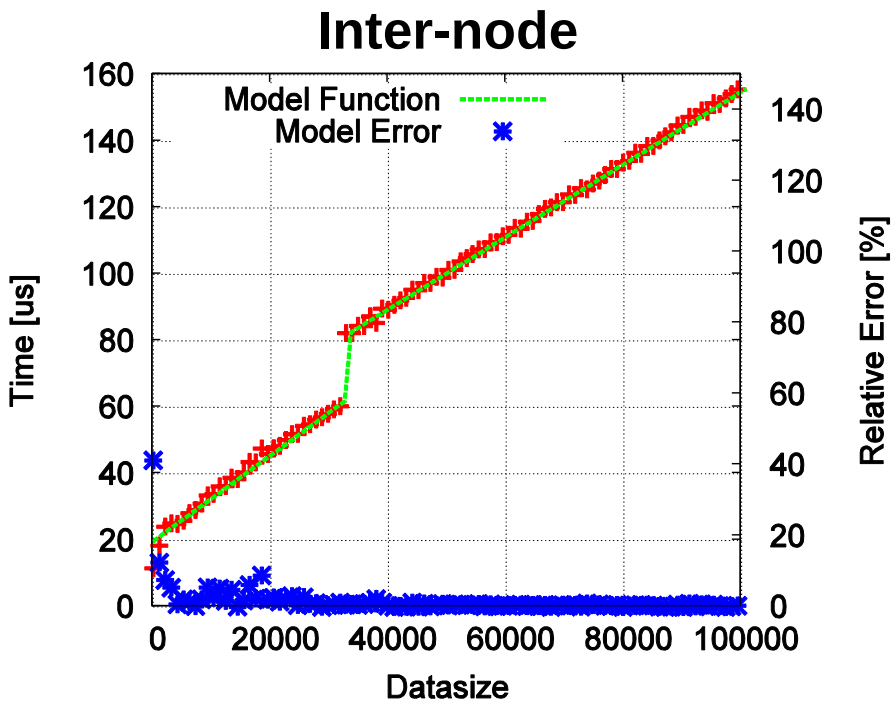
$\mathcal{B}$	$t_1(\mathcal{B})[\mu s]$	$t_2(\mathcal{B})[\mu s]$	$s(\mathcal{B})$
FF	255	326	2500
GF	88	157	1900
LL	1.3	2.2	2500
FL	30	56	2000
CG	0.425	0.483	1200

Example block: GF



System Model: Communication Parameters

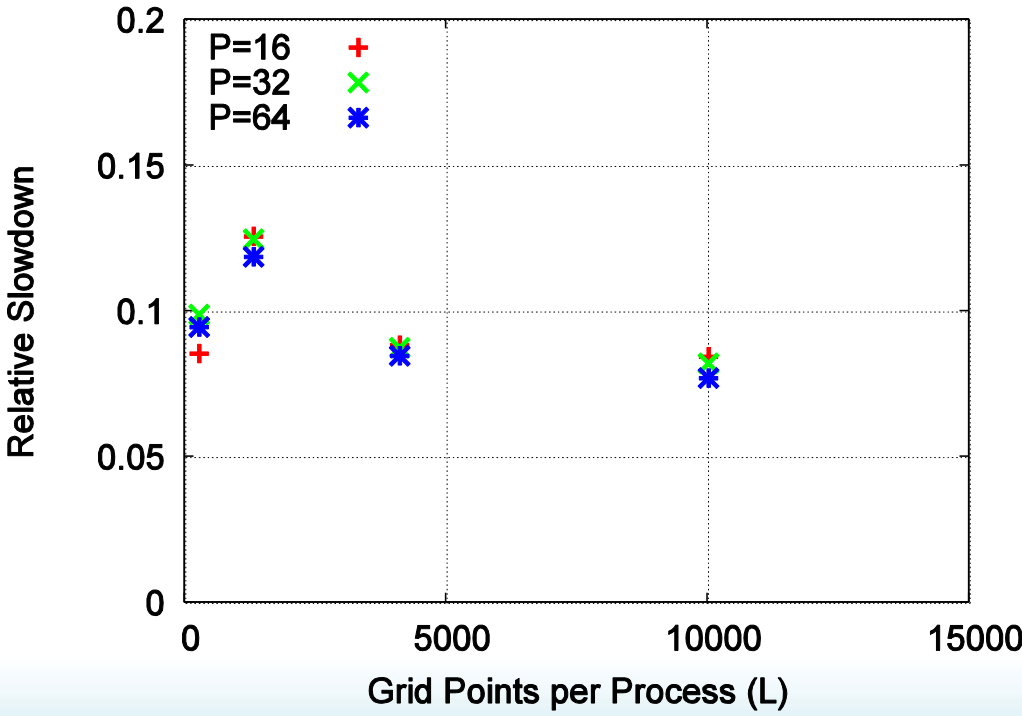




link	range	L	o	g	G
intra	$0 < S \leq 32768$	2.7	3.5	3.0	0.00068
intra	$S > 32768$	2.7	33.5	3.0	0.00045
...	...	...	...	...	...

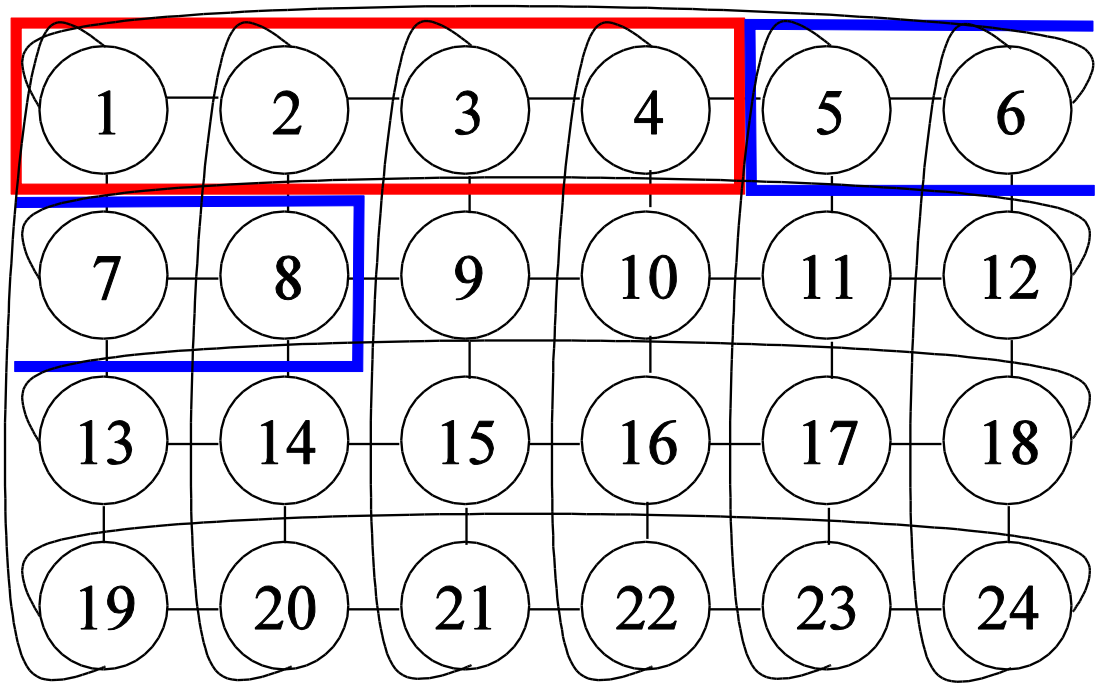
On-node Memory Contention

- Two cores share one memory controller
  - Congestion has non-trivial performance effects
  - Empirical analysis
  - Assume fixed 10% slowdown



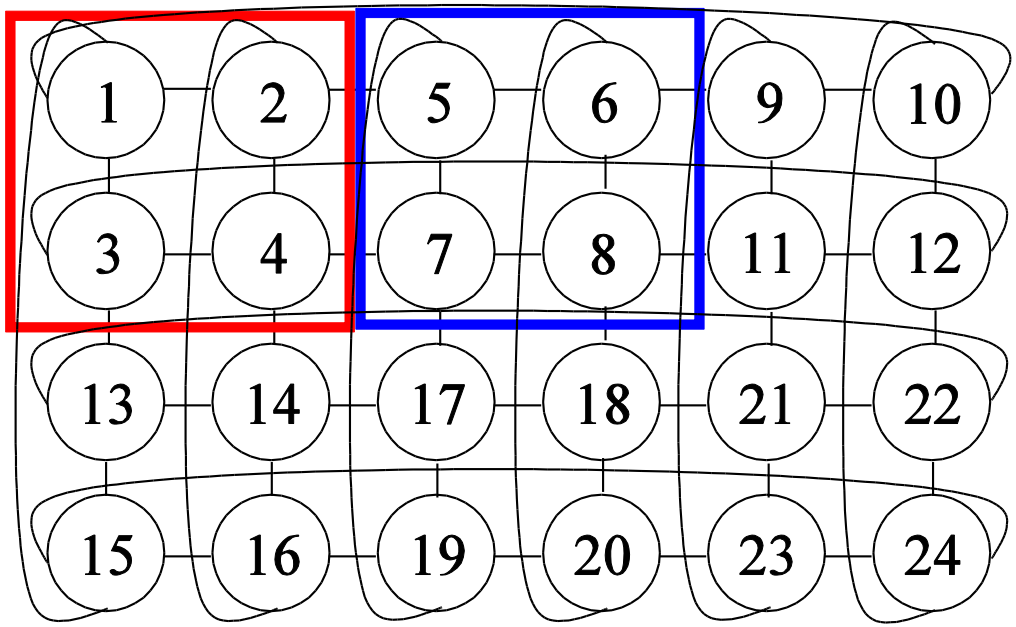
Process-to-Node Mapping

- Trivial linear default mapping
- With 4 processes per node:
  - 6 internal edges
  - 10 remote edges
- Wrap-around
  - Looses two internal edges
  - Unbalanced communication



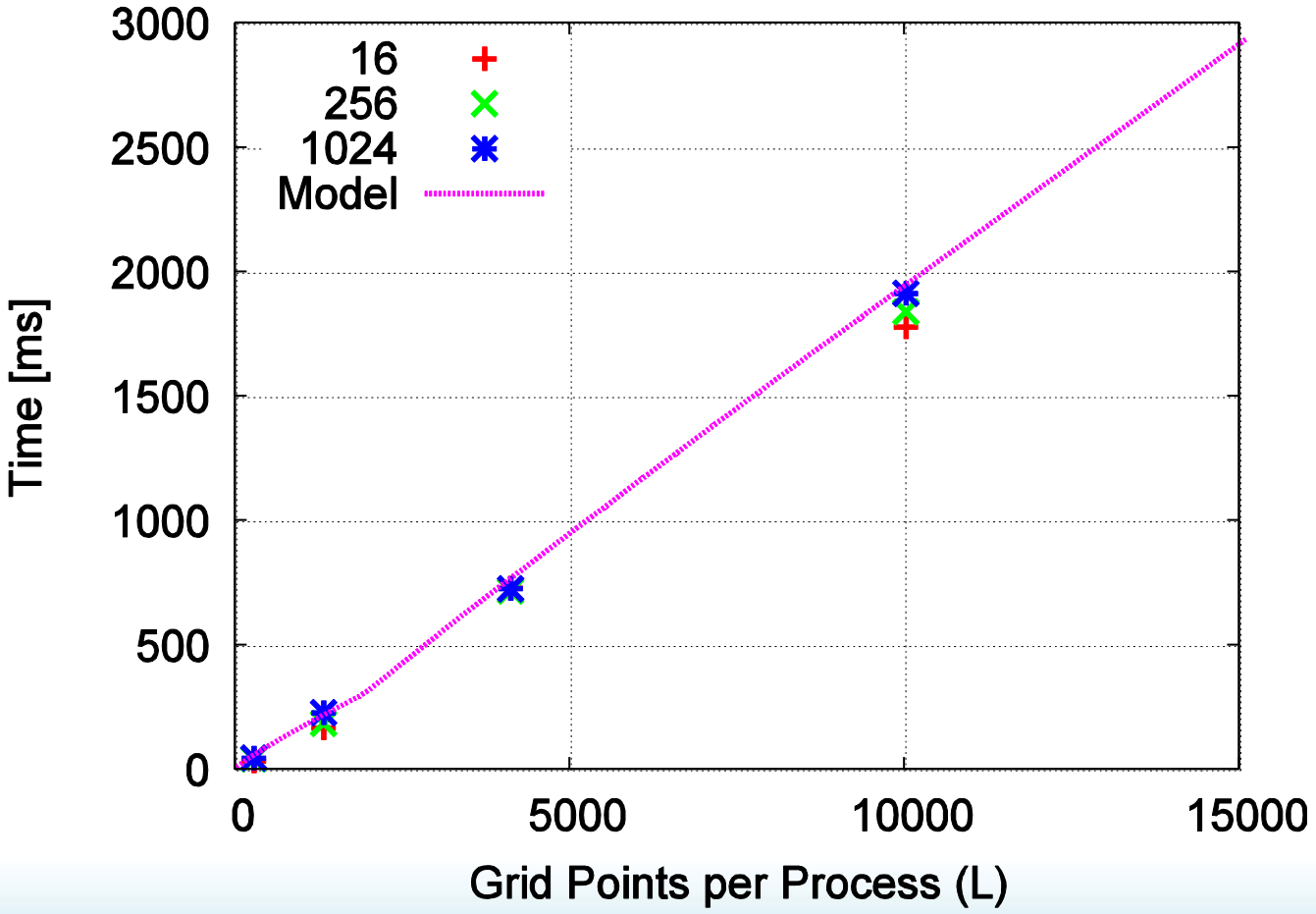
Optimized Process-to-Node Mapping

- Optimal mapping
  - cf. Lagrange multiplier
  - 8 internal edges
  - 8 remote edges



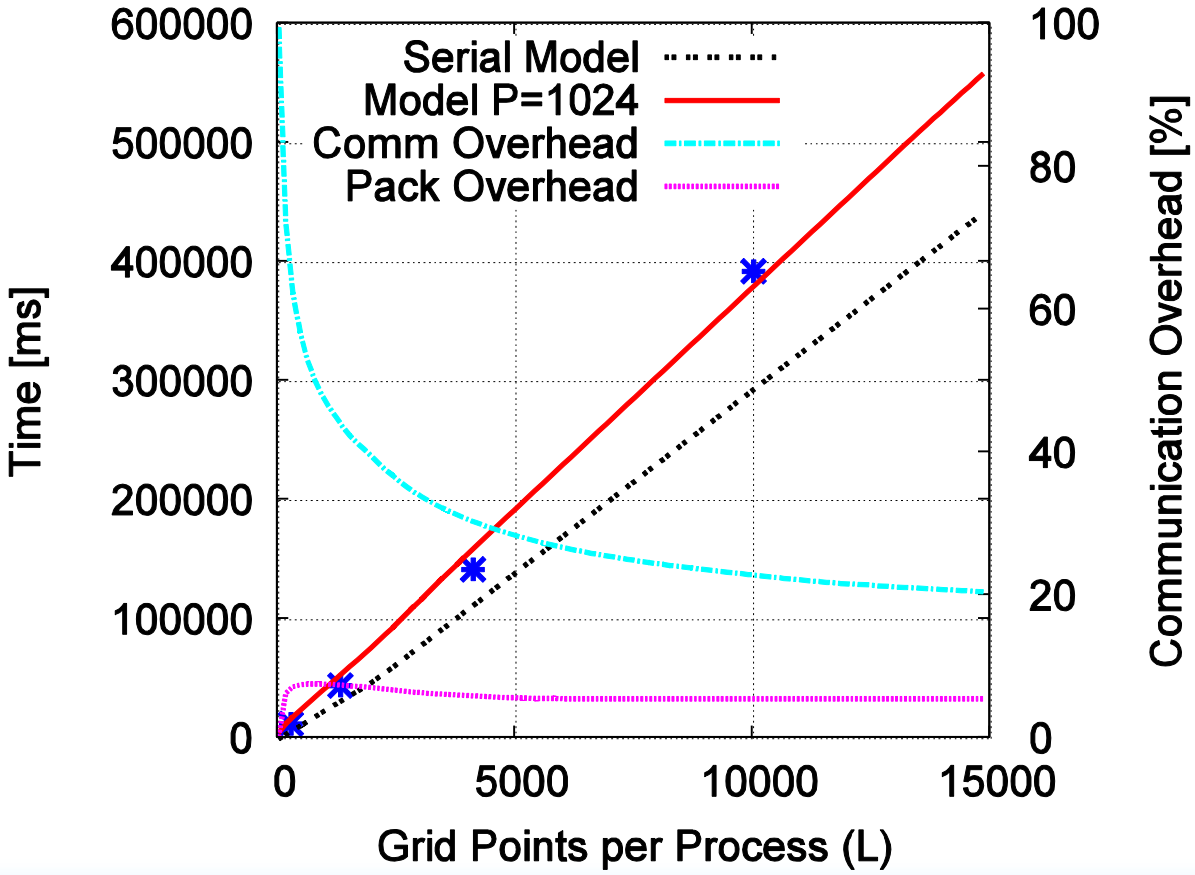
- Similar for 4d mapping
  - 16 cores, optimal sub-block: $\sqrt[4]{16} = 2 \cdot 2 \cdot 2 \cdot 2$
  - ½ remote edges

Parallel Performance Example: GF

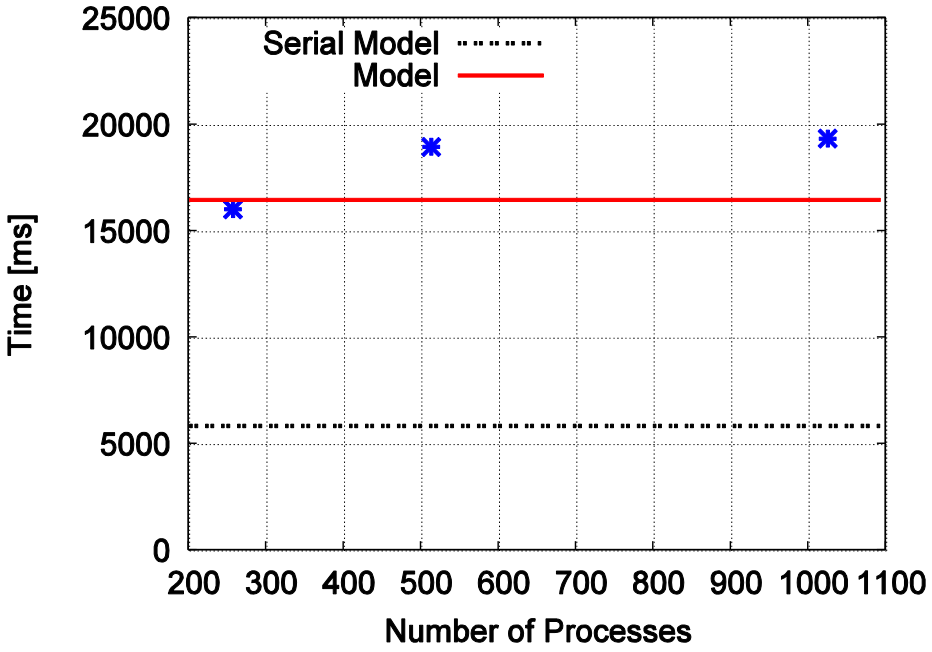


Parallel Performance Model

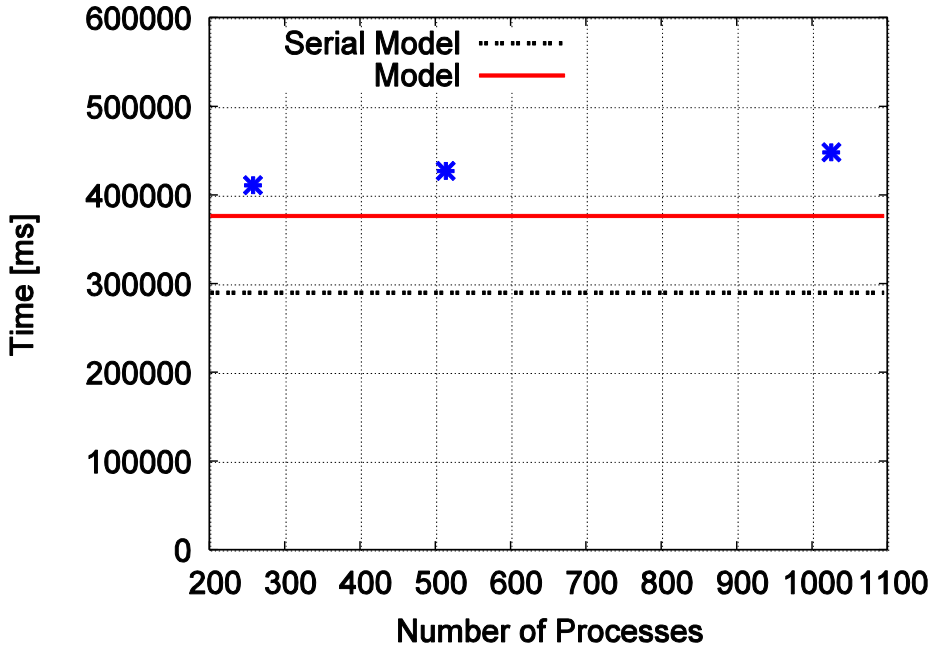
- Example:
  quantify maximum benefit of MPI datatypes!
  - cf. EuroMPI'10



Scaling with the Number of Processes



$V=6^4$



$V=10^4$

Generating a P7 Model and Prediction

- Back to this guy:

Sequential Baseline – Now very Simple!

$$T(\mathcal{B}, V) = t_1(\mathcal{B}) \cdot min\{s(\mathcal{B}), V\} + t_2(\mathcal{B}) \cdot max\{0, V - s(\mathcal{B})\}$$

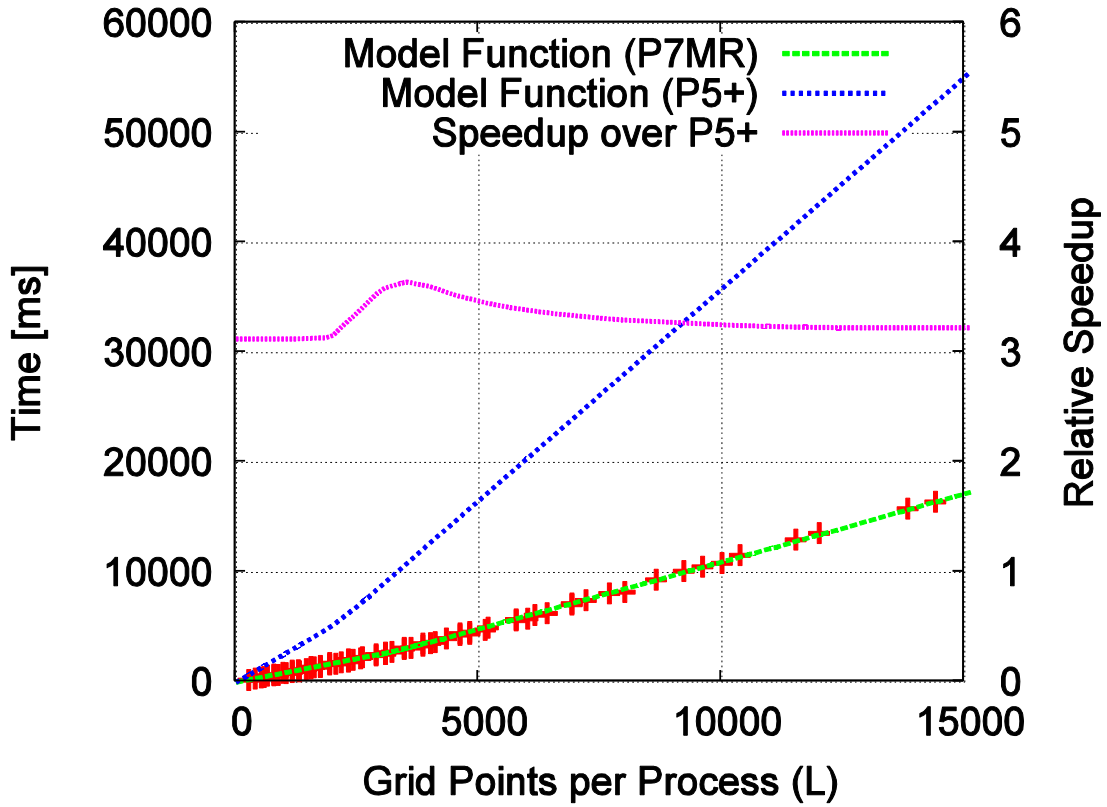
Power5+

$\mathcal{B}$	$t_1(\mathcal{B})[\mu s]$	$t_2(\mathcal{B})[\mu s]$	$s(\mathcal{B})$
FF	255	326	2500
GF	88	157	1900
LL	1.3	2.2	2500
FL	30	56	2000
CG	0.425	0.483	1200

Power7 MR

$\mathcal{B}$	$t_1(\mathcal{B})[\mu s]$	$t_2(\mathcal{B})[\mu s]$	$s(\mathcal{B})$
FF	62.4	92	3000
GF	27.8	48	4000
LL	0.425	0.68	4000
FL	11.4	20	3500
CG	0.239	-	∞

Power7 MR Sequential Model



- relative error is less than 10%

Communication Parameters (P7 MR)

- Intra-node parameters benchmarked (Netgauge)

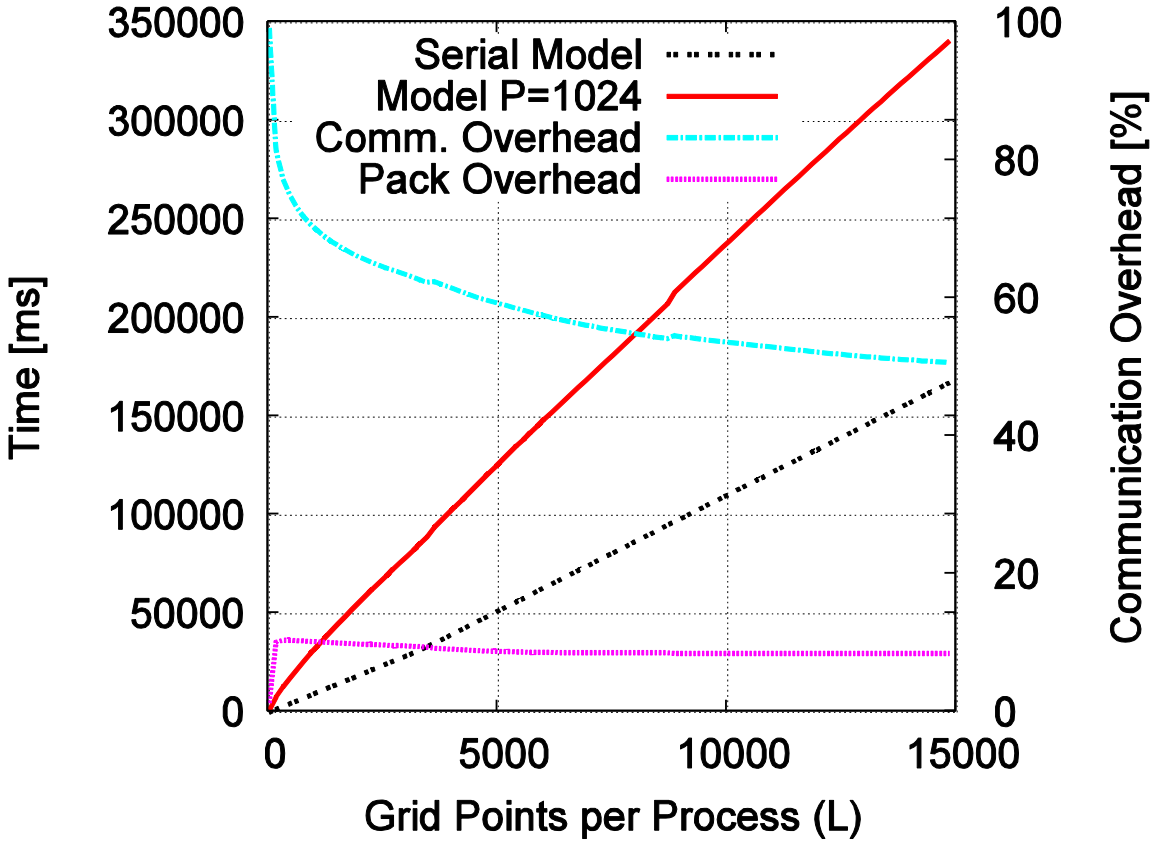
link	range	L	o	g	G
intra	$0 < S \leq 32768$	1.43	0.48	1.03	0.00018
intra	$S > 32768$	1.43	5.66	1.16	0.00022

- Network parameters from (public) documentation
  - Latency ~ $1\mu s$
  - Overhead ~ $0.5\,\mu s$
  - Gap ~ $0.5\,\mu s$
  - Gap per Byte (bandwidth) ~ 0.2 ns (5 GB/s)
- Allreduce : $1\mu s \cdot log_2(300000) = 18.19\mu s \approx 20\mu s$

Process-to-Node Mapping

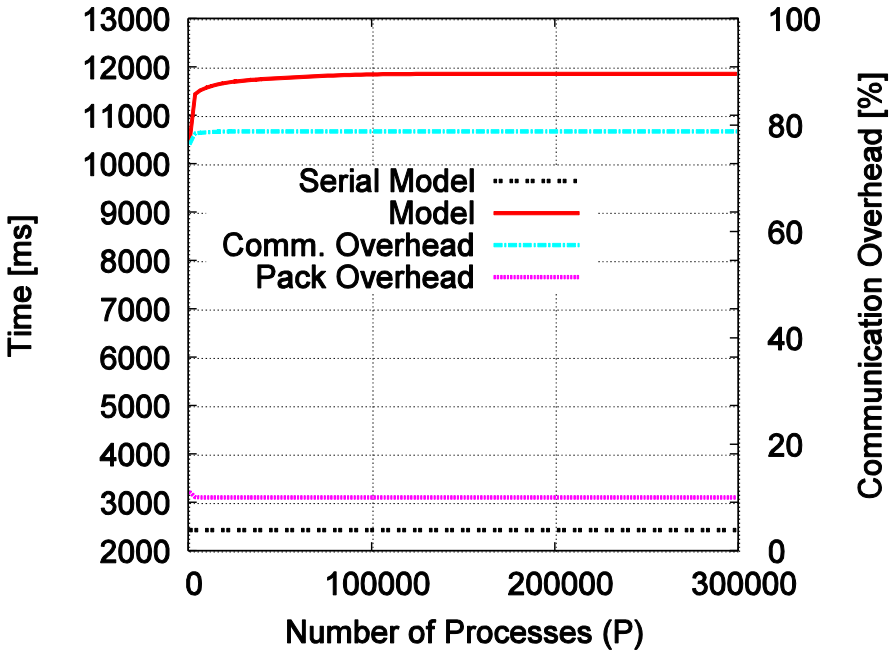
- assume two 2x2x2x2 sub-blocks per node
  - 8 edges per process
  - 16 * 4 local edges per sub-block
  - 2*2*2 edges between the two blocks
    - 72 local edges
    - 56 remote edges
    - local : remote ratio = 1 : 0.7778

Complete P7 MR Prediction

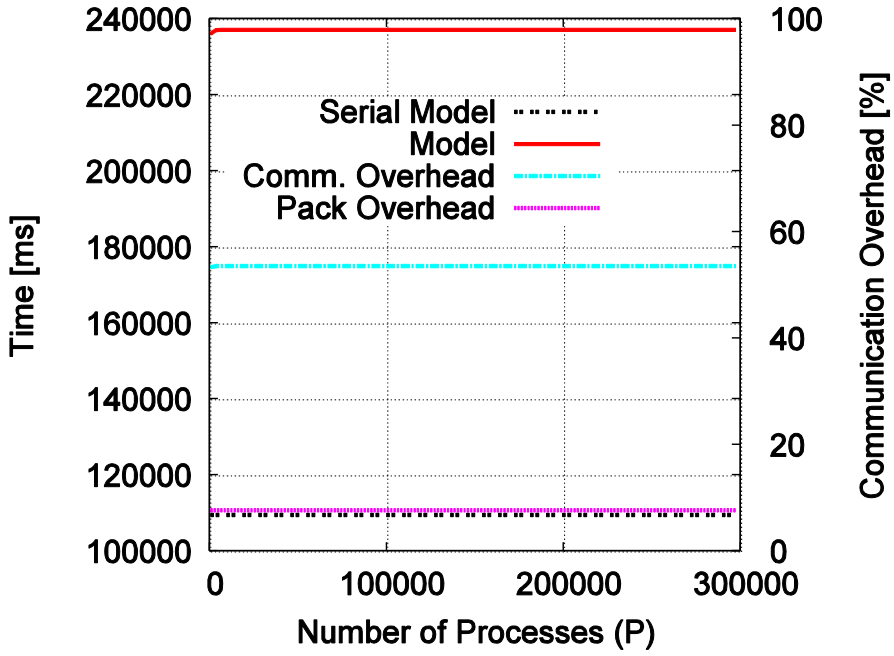


- Shared memory communication causes high overhead

Weak Scaling to 300000 Cores



$V=6^4$



$V=10^4$

Benefits of the Model

- Allows to estimate (with varying P and V)
  - Communication overheads
    - on-node vs. off-node communication
    - evaluate decomposition and mapping strategies offline
  - Time spent in message packing
    - upper bound for benefits of datatypes (cf. EuroMPI'10)
  - Scaling bottlenecks
    - Allreduce is relatively non-critical for scaling up
      - ignoring OS noise for now ☺
  - Predict runtime at scale
    - estimate overheads at large scale
  - …

Takeaways, Questions & Discussion

- Performance Modeling should become routine!
  - Modeling during application and system design
  - Maintain models through application lifetime
  - Improves productivity
  - Needs tool support
    - Existing, needs glue
  - It's relatively simple!
    - For static applications