Lecture 20: Distributed Memory Parallelism

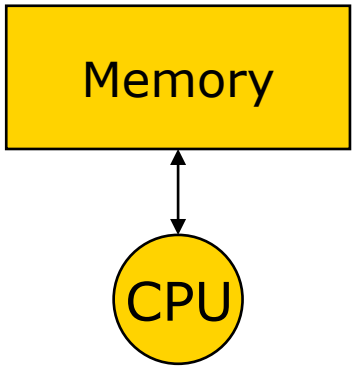
William Gropp
www.cs.illinois.edu/~wgropp

A Very Short, Very Introductory Introduction

- We start with a short introduction to parallel computing "from scratch" in order to get our terminology straight.
- You may already know this, but it will help get us synchronized (a term from parallel computing).

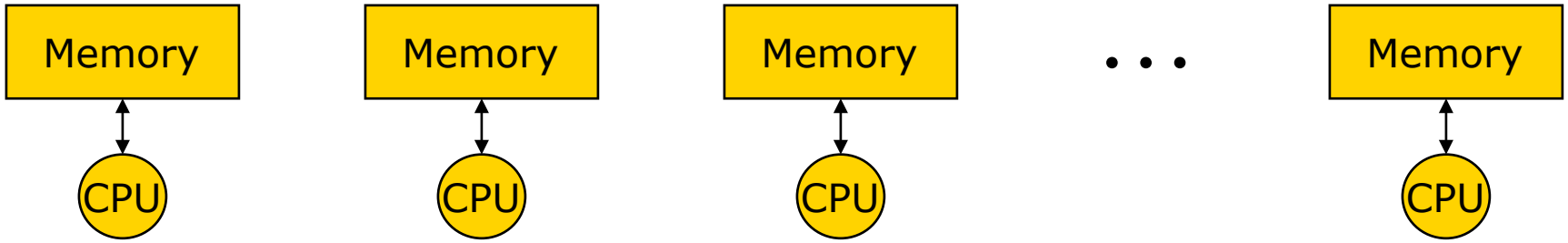
The Starting Point: A Single Computer

- A single ("normal") computer, circa 2000



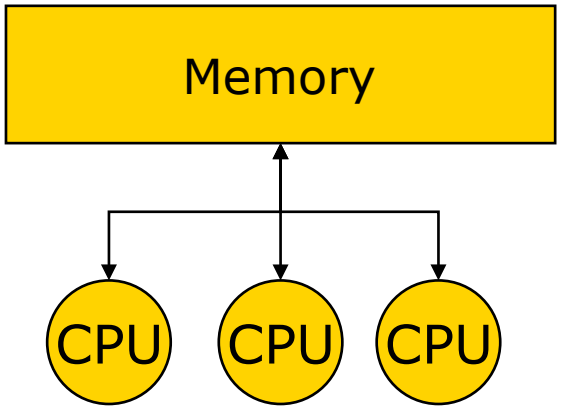
- Program: single sequence of instructions
  - "Usual" programming languages compiled into instructions understood by CPU
  - Most programming models, including parallel ones, assumed this as "the computer"
- Process: program + address space + program counter + stack

A Bunch of Serial Computers



- Program: Many instances of same sequential program operating on different data in difference address spaces
- Multiple independent processes with same program
- Examples
  - High throughput computing
  - seti@home and folding@home

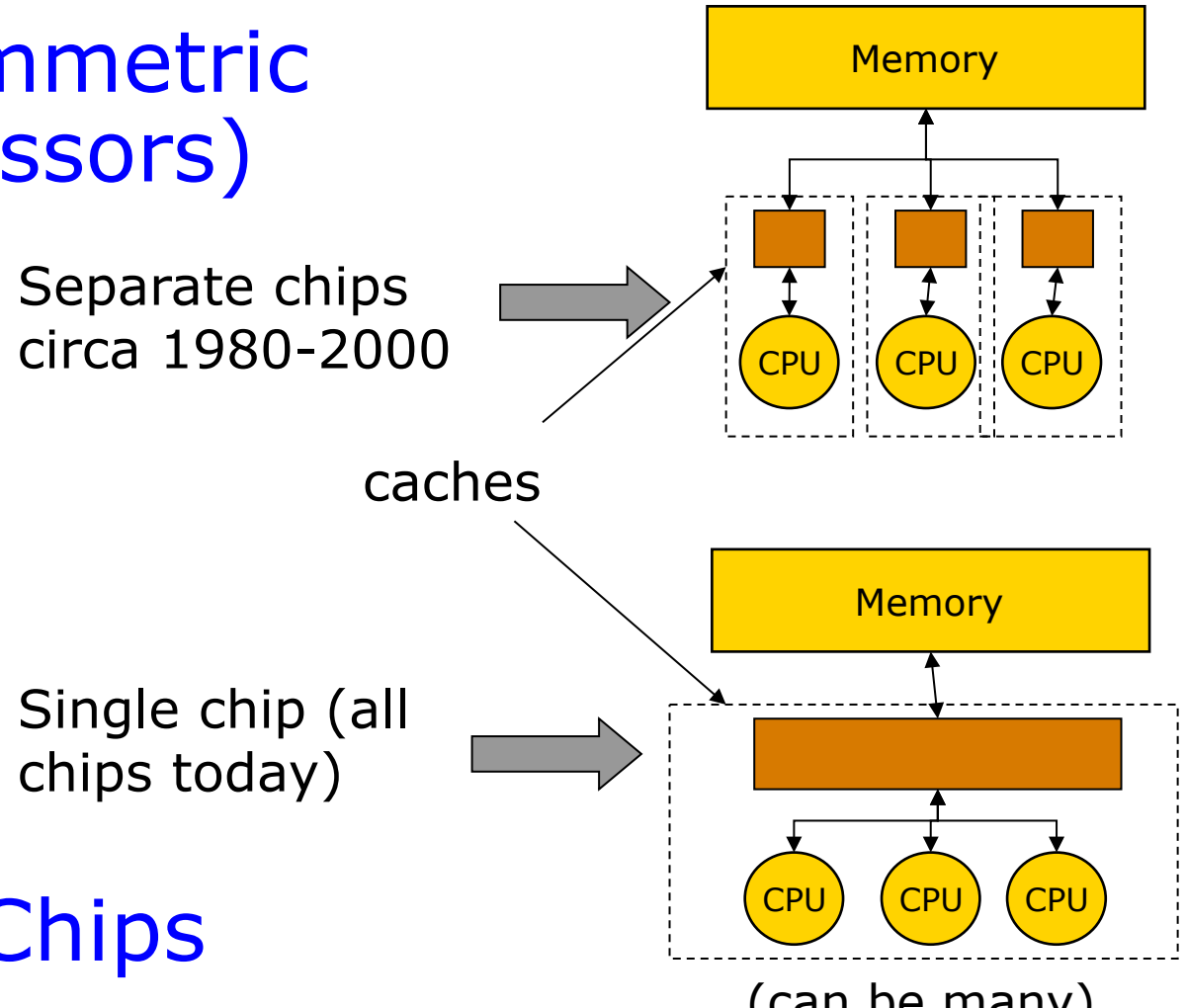
Multiple CPU's per Memory System



- May use multiple processes with own memory plus some way of sharing memory and coordinating access (operating-system-specific)
- One process may have multiple threads (pc + stack)'s operating on same address space, guided by same program
  - Sequential programming languages with smart compilers
  - Threads; OpenMP, OpenACC versions of C and Fortran

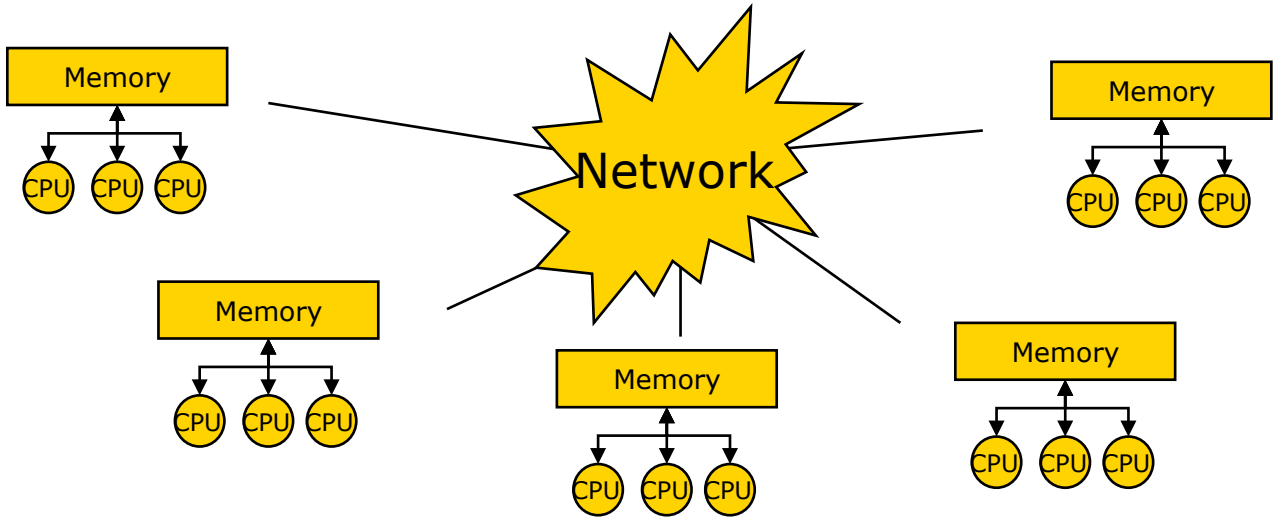
Hardware Variations on Multiple CPU's per Memory System

- SMPs (Symmetric Multiprocessors)



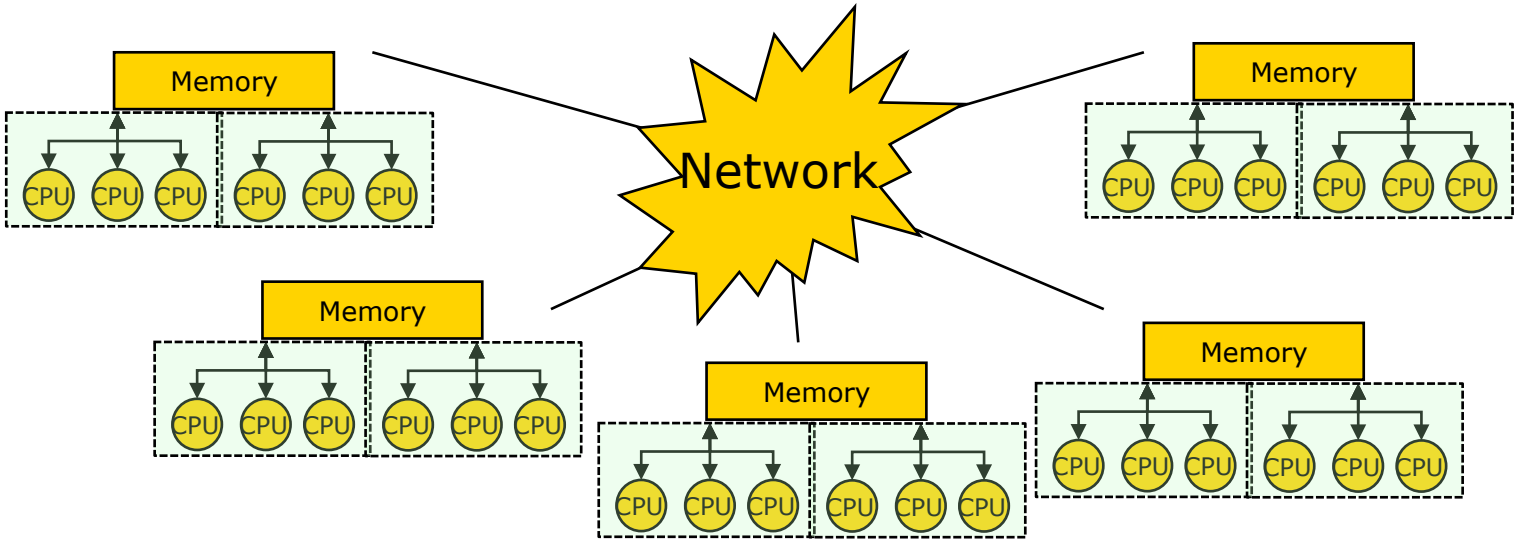
- Multicore Chips

Multiple Memory Systems



- Program: multiple processes, each of which may be multi-threaded, with separate address space
- Program needs some way of expressing communication among processes, so data can move between address spaces
- Many large-scale machines look like this, leading to more interest in programming models that combine the shared and distributed memory approaches to programming.

Multiple Memory Systems



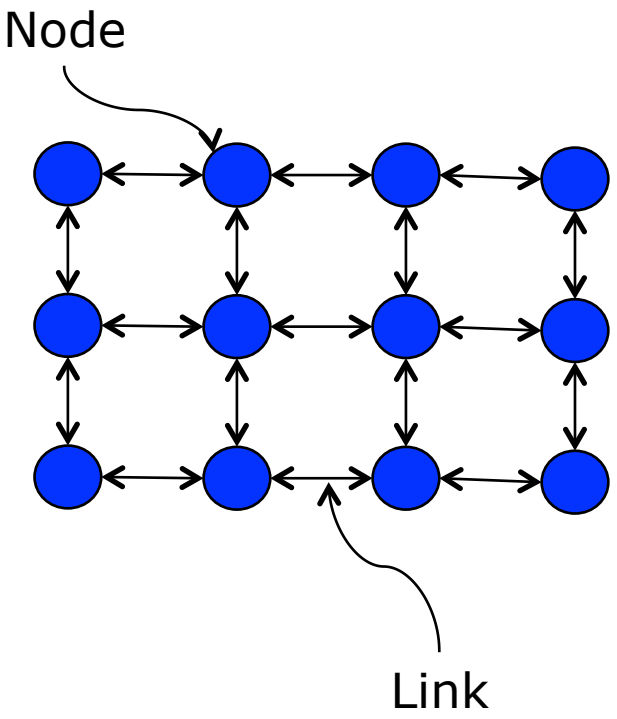
- Most systems have several chips sharing memory, physically placed on the same board (or part of a board)
- We'll call this a *node*
- Beware: Definitions change with time (what we call a core used to be a processor) and often carry assumptions
  - Nodes in the near future may not share memory or may not provide cache-coherent shared memory, even within a single chip

What about that "Network"?

- The network or interconnect is what distinguishes a distributed memory parallel computer
- Important characteristics of parallel computer interconnects include:
  - How are the nodes connected together
    - This is the *topology* of the network
  - How are the nodes attached to the network?
  - What is the performance of the network?
- One important performance measure is the *bisection bandwidth:*
  - If I cut the network in half, and add up the bandwidth across all of the cut links, what is the minimum value across all possible cuts?

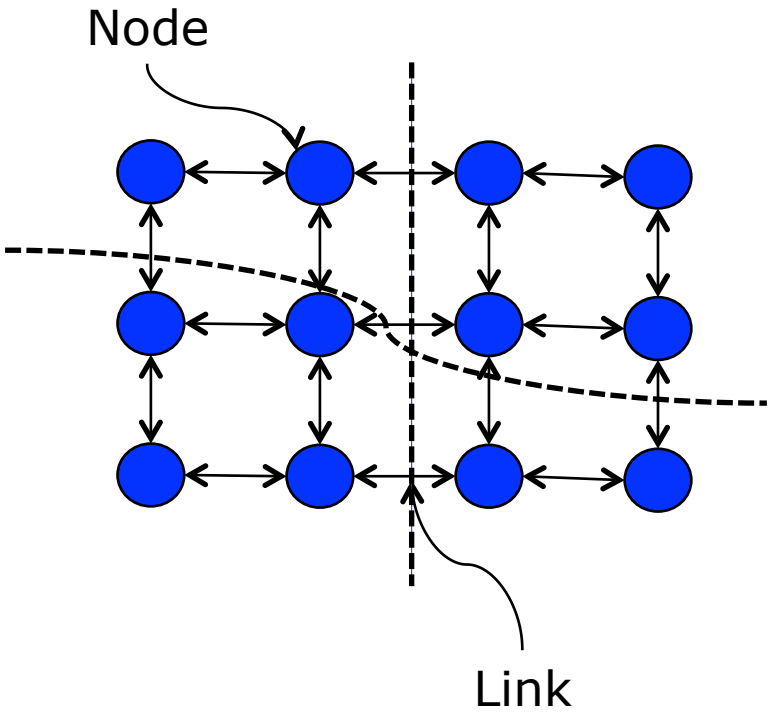
A Simple Interconnect

- Each node (vertex in the graph, shown as a circle) connected to nearest neighbors in x, y direction (but not diagonal)
- Data *routed* from one node to the other by traveling over one or more links (edges of the graph)



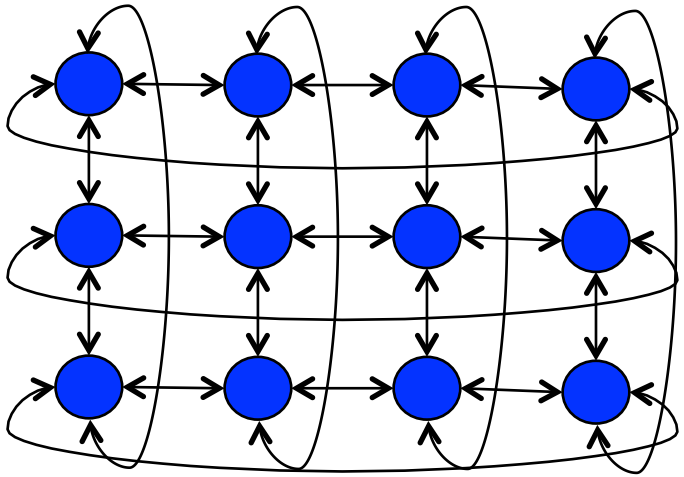
Bisection Bandwidth

- Try all cuts in half, take minimum
- In this case, bisection bandwidth is 3*individual link bandwidth



Mesh and Torus Interconnects

- Each compute node is a node on a mesh or torus
- May be 1, 2, 3 or more dimensions
- Torus simply adds links connecting ends together
  - Clever physical layout keeps these links the same length
- Note a rectangular subset of a mesh is a mesh, but a rectangular subset of a torus is not necessarily a torus (it's a mesh)



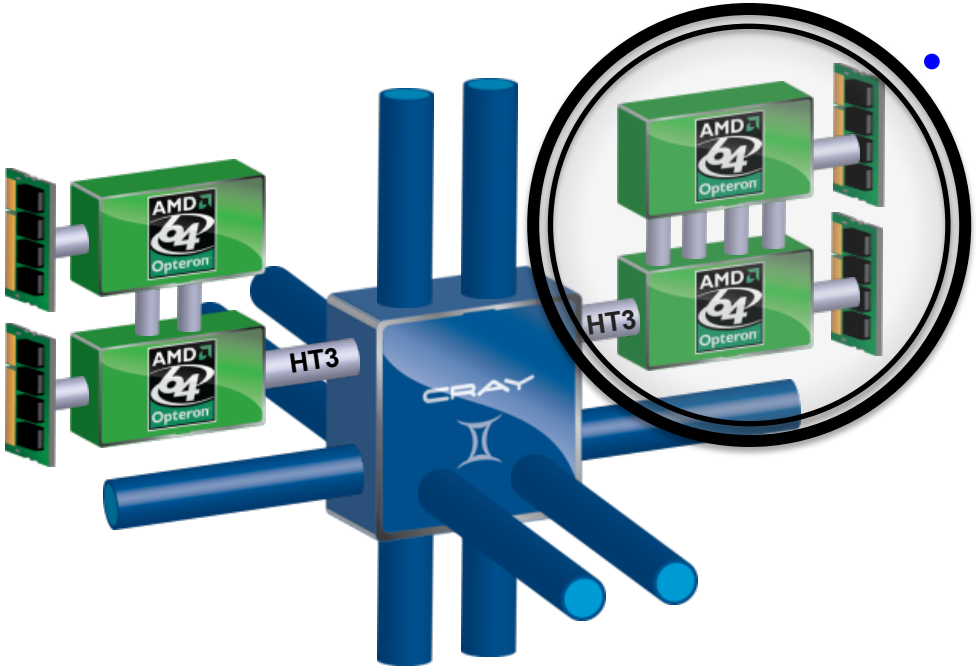
Mesh and Torus Features

- Advantages
  - Constant cost to scale to more nodes
  - Simple routing algorithms
  - Relatively easy to reason about performance
  - Matches certain problems well
- Disadvantages
  - Bisection bandwidth does not scale with size
    - For general communication patterns, network contention limits performance

Question

- How does the bisection bandwidth of a 2D mesh scale with the number of nodes? Consider an n x n mesh (so n^2 nodes) and compute the bisection bandwidth per node as a function of n and the link bandwidth L.

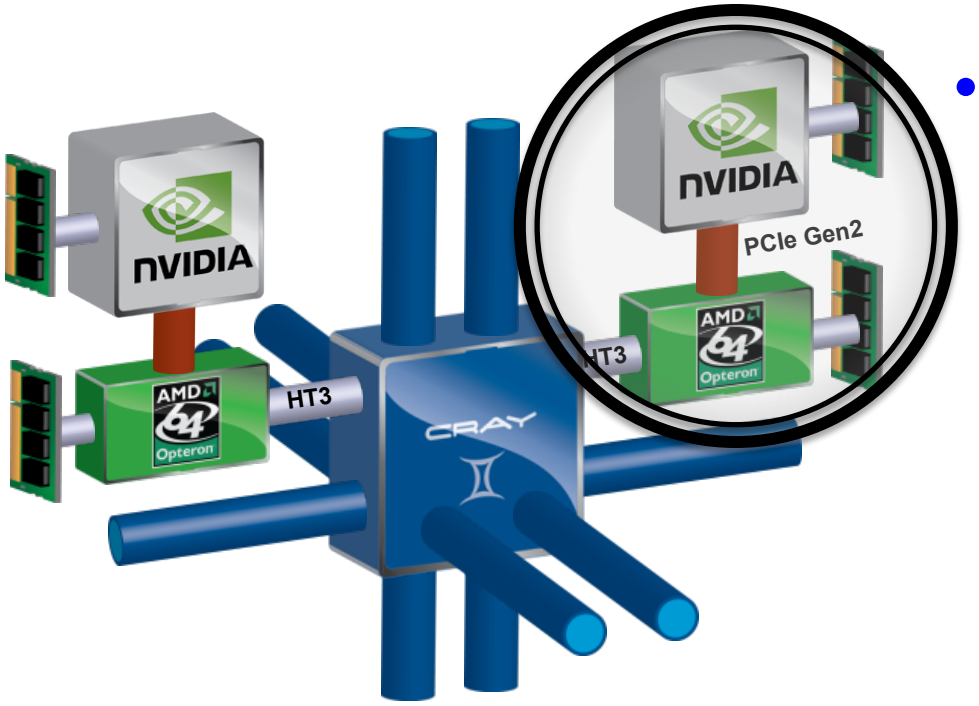
Cray XE6 Nodes



Blue Waters contains 22,640 Cray XE6 compute nodes.

- Dual-socket Node
  - Two AMD Interlagos chips
    - 16 core modules, 64 threads
    - 313.6 GFs peak performance
    - 64 GBs memory
      - 102 GB/sec memory bandwidth
  - Gemini Interconnect
    - Router chip & network interface
    - Injection Bandwidth (peak)
      - 9.6 GB/sec per direction
    - 2 links in x, z, direction, only 1 link in y direction

Cray XK7 Nodes

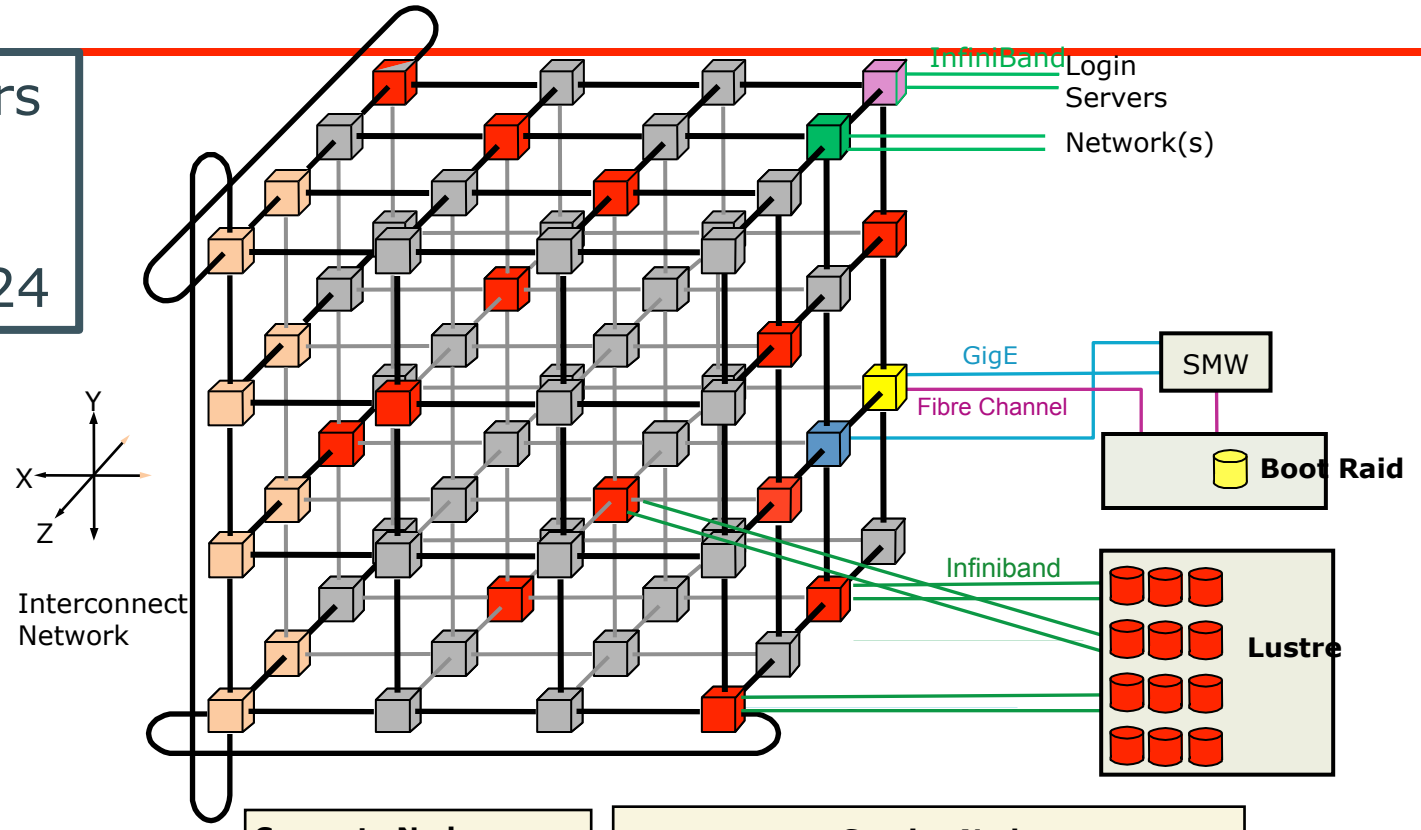


- Dual-socket Node
  - One AMD Interlagos chip
    - *Same as XE6 nodes*
  - One NVIDIA Kepler chip
    - >1 TF peak performance
    - 6 GBs GDDR5 memory
      - 180 GB/sec bandwidth
  - Gemini Interconnect
    - *Same as XE6 nodes*

Blue Waters contains 4,224 Cray XK7 compute nodes.

Gemini Interconnect Network

Blue Waters
3D Torus
Size
24 x 24 x 24

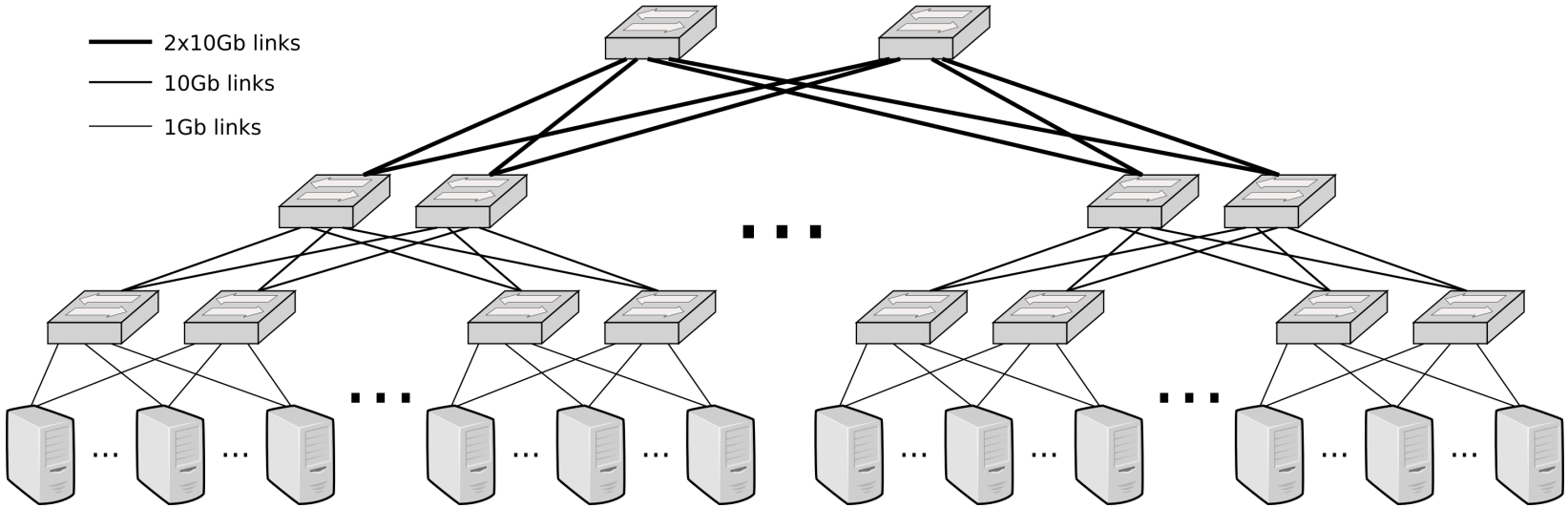


Service Nodes spread throughout the torus

Multilevel Networks

- Network made up of switches with many ports that are connected to
  - ♦ Compute nodes
  - ♦ Other switches
- Modern switches can handle many ports simultaneously (> 100)

Example Fat Tree Network



From
http://www-sop.inria.fr/members/Fabien.Hermenier/btrpcc/img/fat_tree.png

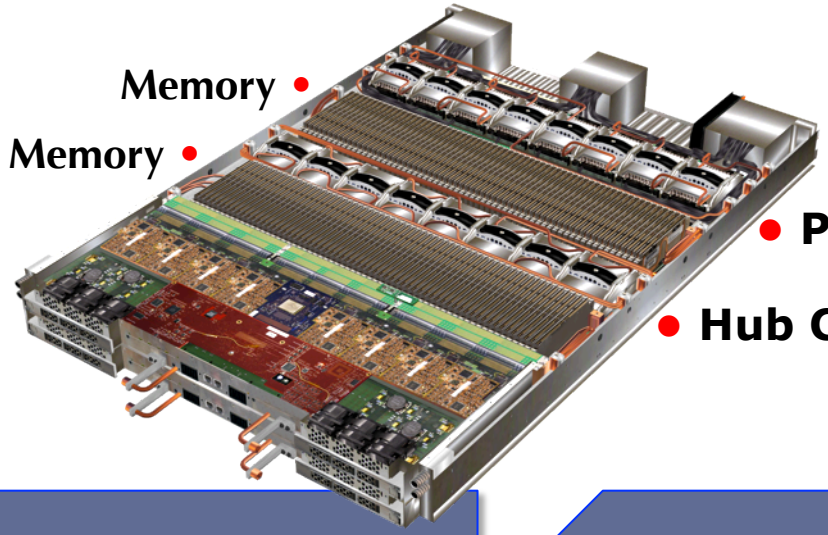
Features of Multilevel Networks

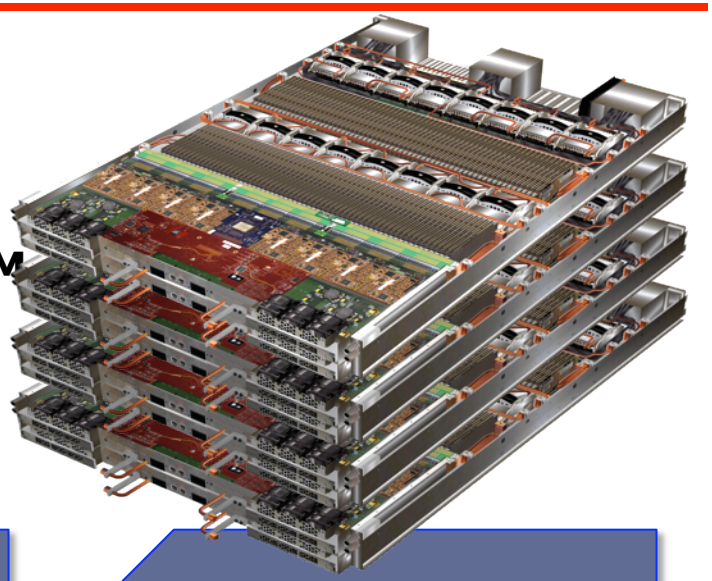
- Advantages
  - Bisection bandwidth can be high; increase by adding more switches
  - "Hops" between nodes small
- Disadvantages
  - Cost grows faster than linear with number of compute nodes
  - More complex; harder to reason about performance

Multilevel Networks Part 2

- As you move up in level (away from the compute nodes) it is common to need more bandwidth between the switches
  - Solution: Use higher bandwidth links, especially optical links
    - Electrical links are cheap, fast over short distances
    - Optical links have enormous bandwidth, even over long distances, but are expensive

IH Server Node (Drawer) and Supernode





Specifications:

8 Quad-chip Modules
Up to 8 Tflops
1 TByte of memory
4 TB/s memory
8 Hub Chips
9 TB/sec/Hub

Packaging:

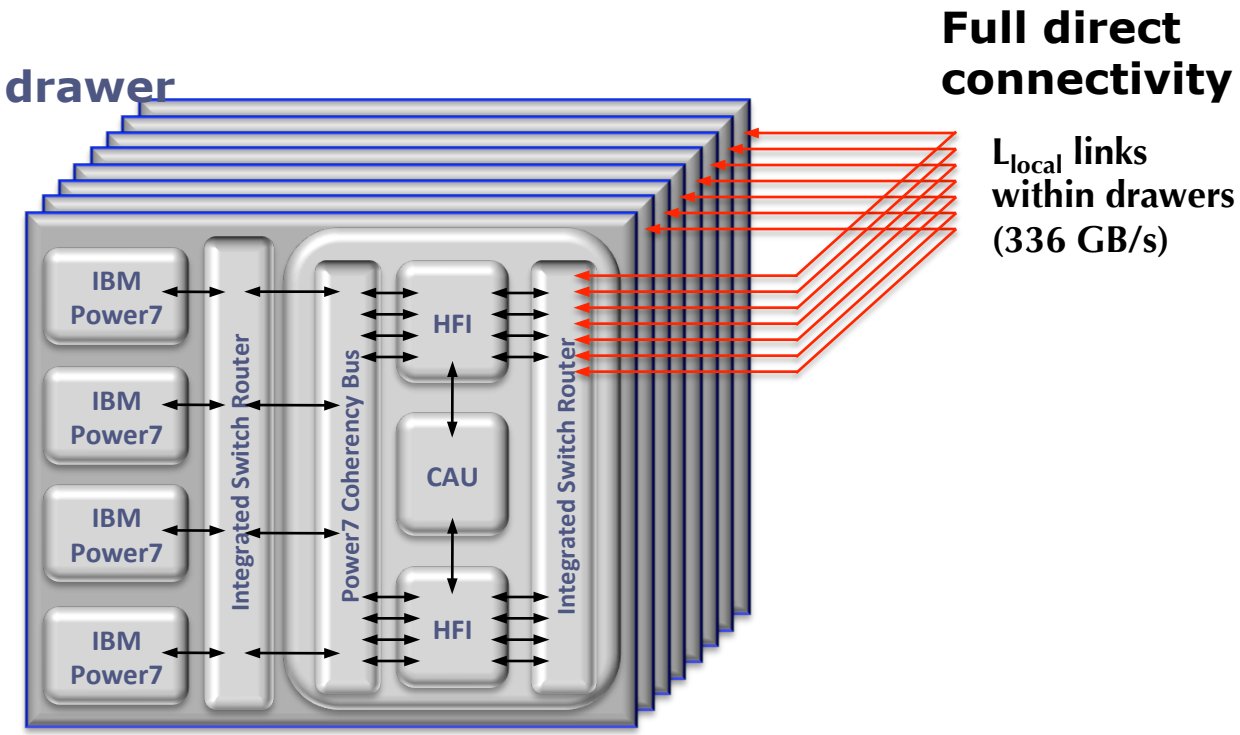
2U Drawer
39″w x 72″d
> 300 lbs.

Fully water cooled (QCMs, Hubs, and Memory)

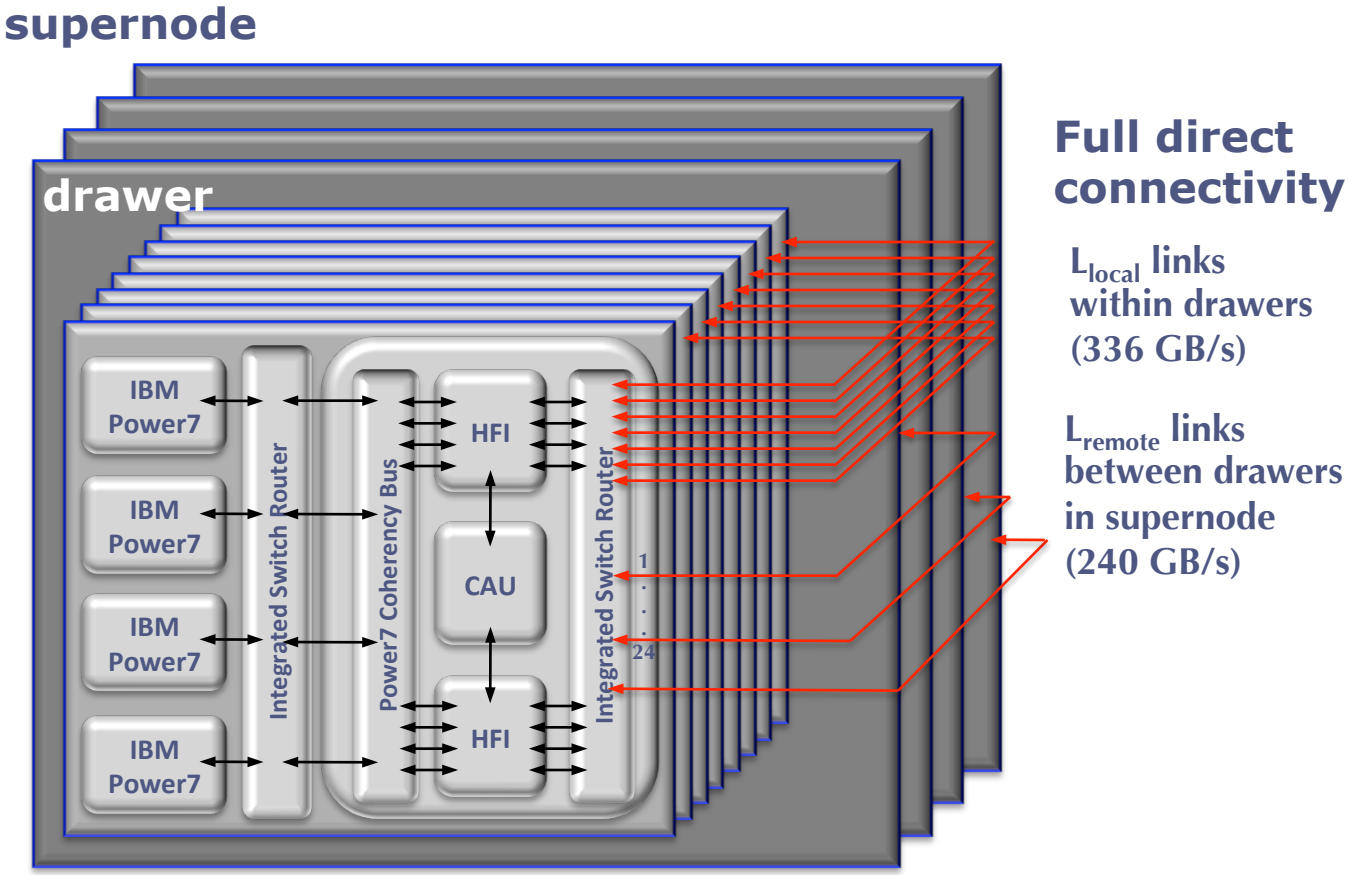
Specifications:

4 IH Server Nodes
Up to 32 Tflops
4 TBytes of memory
16 TB/s
32 Hub Chips
36 TB/sec

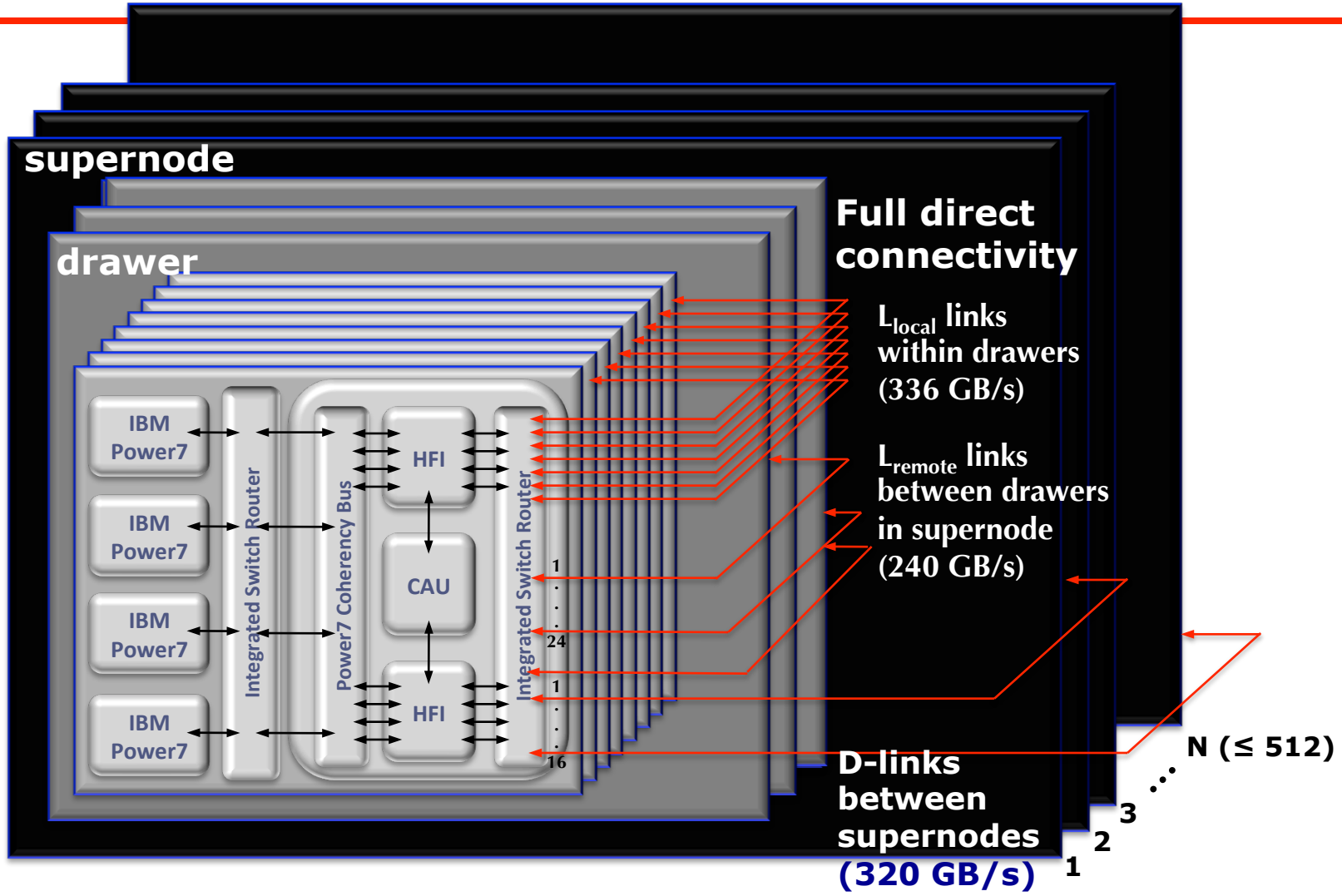
Logical View of IBM PERCS Interconnect



Logical View of IBM PERCS Interconnect

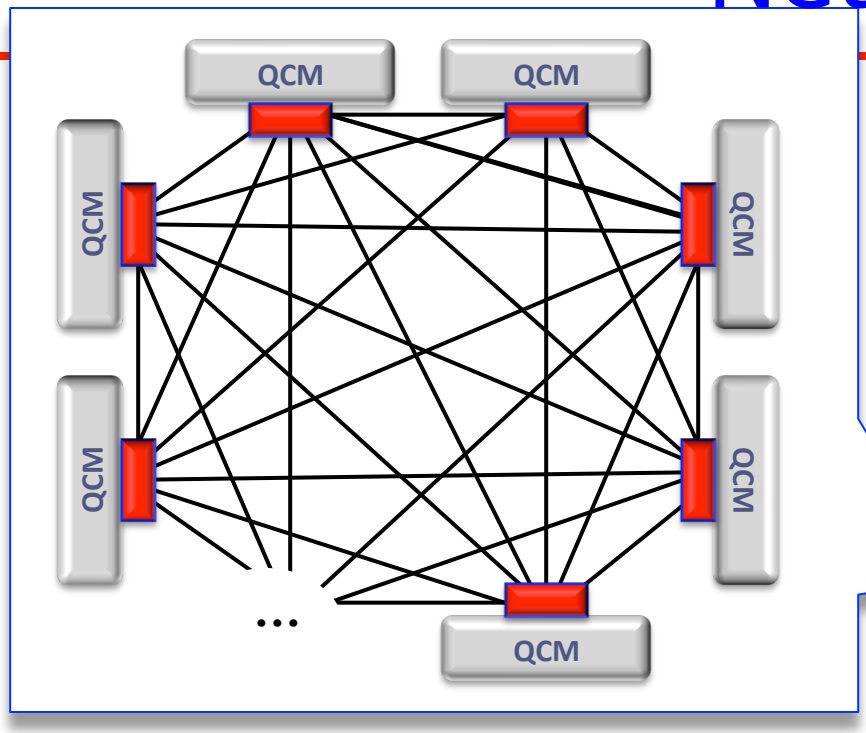


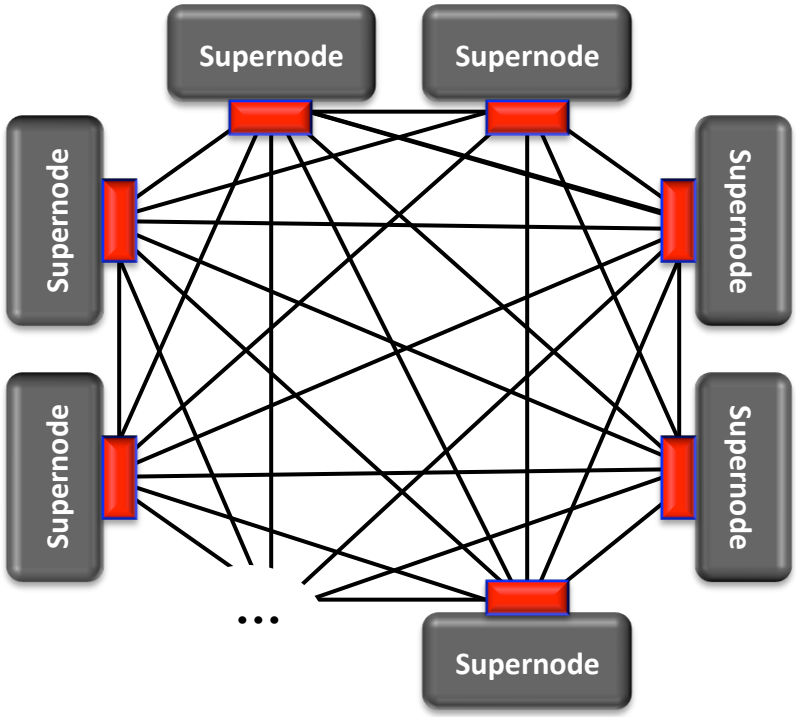
Logical View of IBM PERCS Interconnect



PARALLEL@ILLINOIS

Two-level (L, D) Direct-connect Network





Each Supernode = 32 QCMs
(4 Drawers x 8 SMPs/Drawer)

Fully Interconnected with L_{local} and L_{remote} Links

Original Blue Waters Plan = 320 Supernodes
(40 BBs x 8 SNs/BB)

Fully Interconnected with D Links

But complex, nonuniform network

Result: Very low hardware latency Very high bandwidth

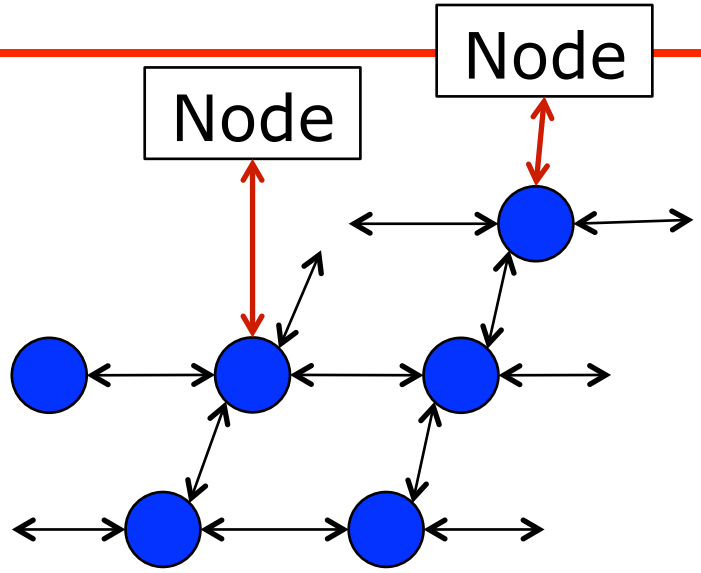
PARALLEL@ILLINOIS

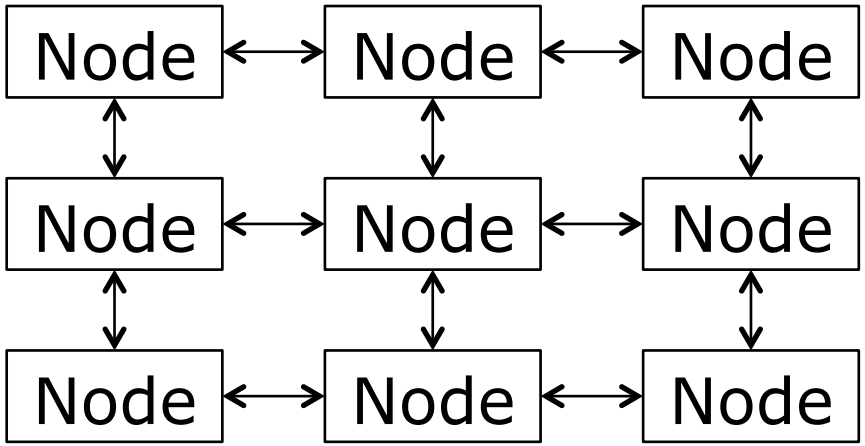
Another Multilevel Network

- Cray Ares network uses a similar approach called a *dragonfly* network
  - ♦ Uses mixed electrical/optical connections
  - ♦ Roughly, a hierarchy of completely connected networks; no more than 4 hops for the sizes considered
- Presentation on Cray XC30
  - ♦ http://www.nersc.gov/assets/Uploads/NERSC.XC30.overview.pdf
- Patent for Dragonfly Network
  - ♦ http://www.faqs.org/patents/app/20100049942

Mesh and Torus Revisited

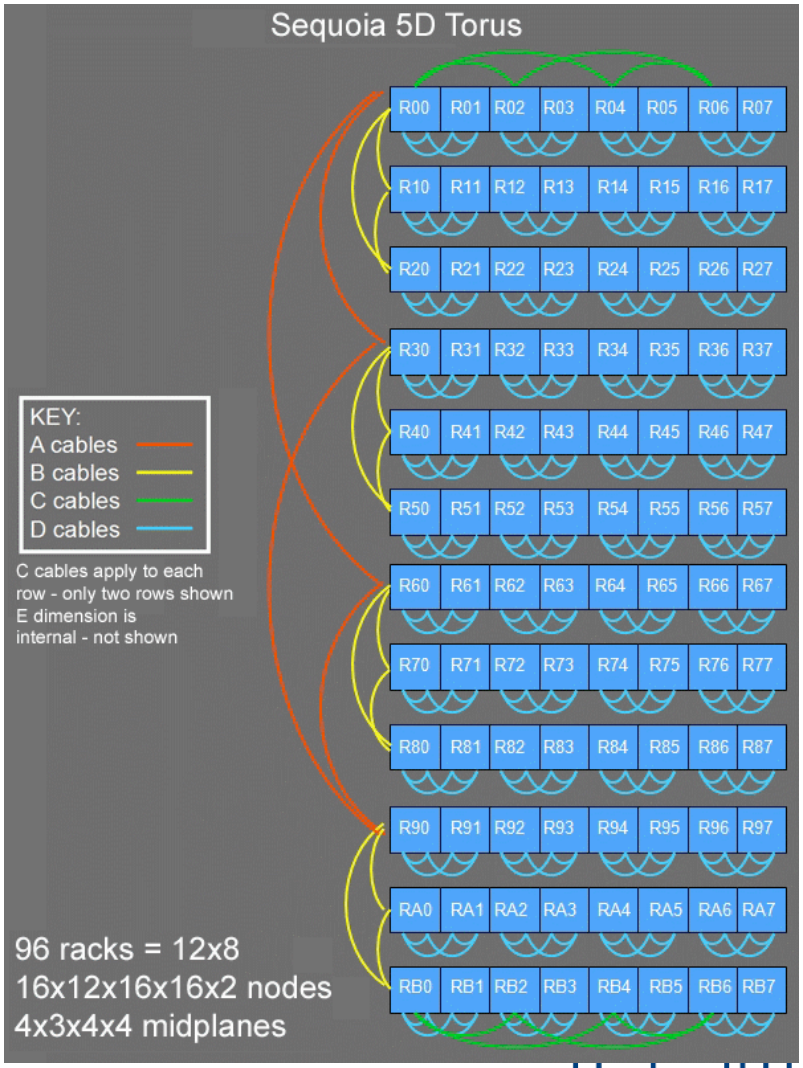
- We've assumed that the node communicates with the mesh through a vertex in the mesh
- Instead, the node could communicate with the mesh directly, through the edges (links) from the vertex



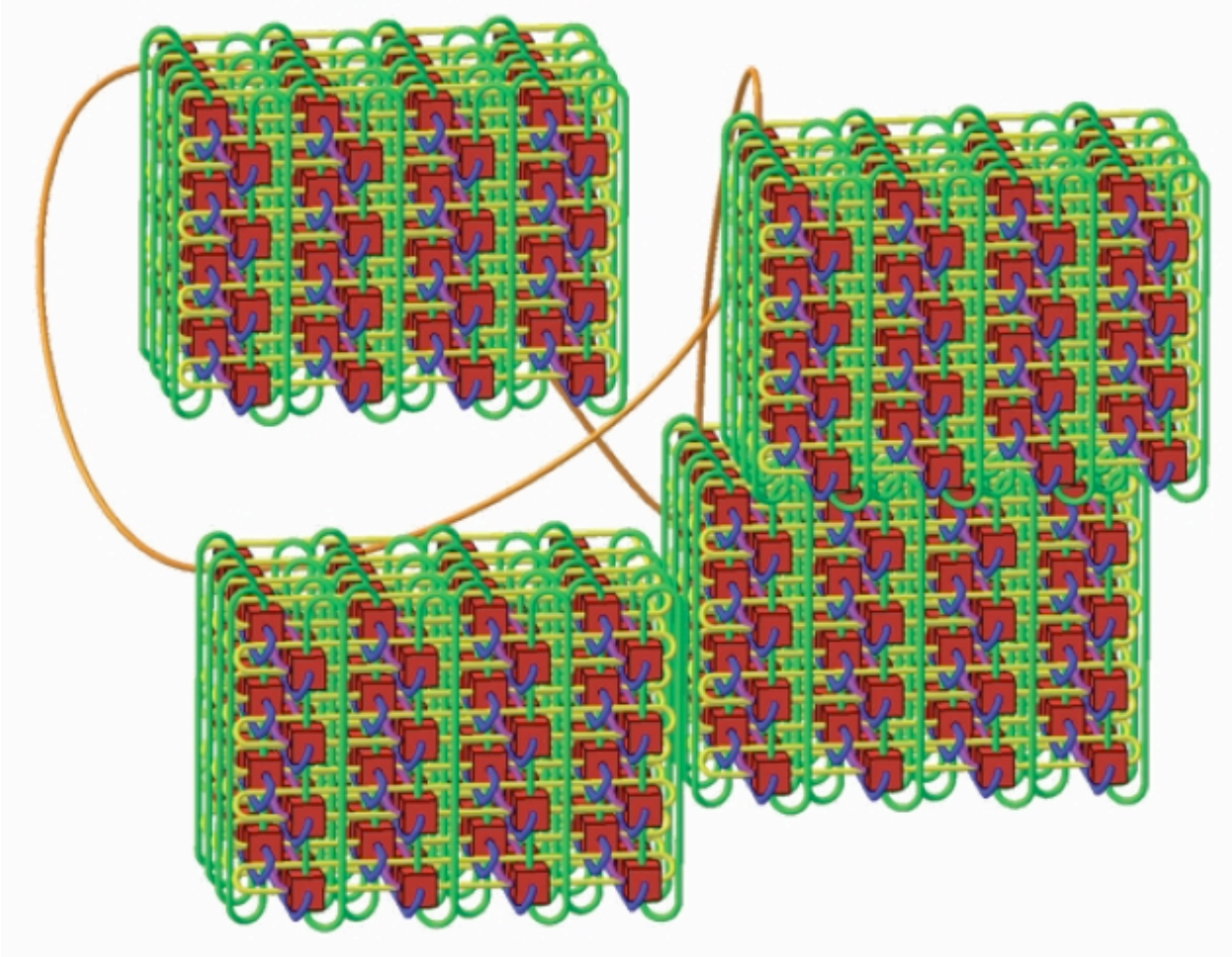


IBM BlueGene Series

- IBM BlueGene use mesh
- Node occupies a vertex, directly manages each link (edge) from that vertex separately
- Each link of modest speed; must drive them all to get comparable bandwidth to other systems



BG/Q Midplane, 512 nodes, 4x4x4x4x2 Torus



More Information on BG/Q

- Nice introduction to ANL's BG/Q, including details of memory system
- See esp. slide 18 on network
- http://extremecomputingtraining.anl.gov/files/2014/01/20140804-atpesc-parker-bgq-arch.pdf

Why This Focus on the Interconnect?

- Distributed memory parallel computers are just regular computers, nodes programmed like any other
- Designing for and programming the distributed memory means thinking about how data moves between the compute nodes

Performance Model

- A simple and often adequate performance model for the time to move data between "nodes" is
  - T = latency + length / bandwidth
  - T = s + r n , r = 1/bandwidth
- On modern HPC systems, latency is 1-10usec and bandwidths are 0.1 to 10 GB/sec
- This model has *many* limitations but is often adequate
  - E.g., does not include the effect of distance or of contention with other messages
  - We'll discuss other models later

Questions

- What sort of network does your system have?
- Read up on the Blue Waters (Cray Gemini) interconnect

Questions

- Consider a 1-dimensional torus where each link has bandwidth b. Consider the following cases:
  1. Each node sends n bytes to the node to the immediate left
  2. Each node sends n bytes to the node k links away to the left
- All communication starts at the same time. Ignore latency. In each of the above cases, how long does it take the communication to complete, in terms of n and b?
- In the second case, what is the effective bandwidth b' for the communication (e.g., if you wanted to use the T=s+n/b model, what value of b should you pick as a function of k)?