Cluster Design Rules:
Effective Cluster Design With Commodity Parts

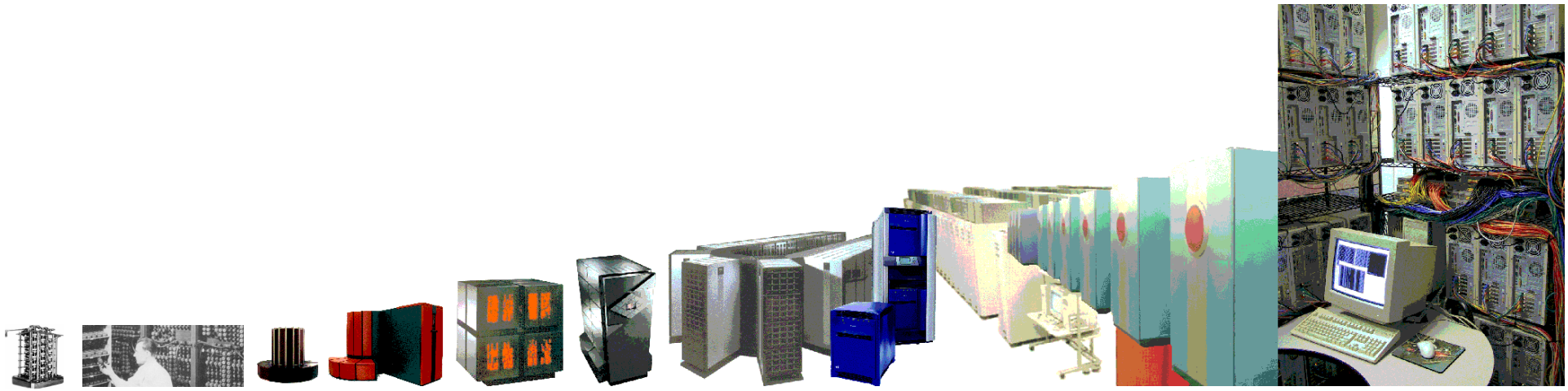
SC2002 Tutorial M13

Monday, November 18, 2002

Hank Dietz and Bill Dieter
Electrical and Computer Engineering Department
University of Kentucky
Lexington, KY 40506-0046
http://aggregate.org/

The Evolution Of Supercomputers

- Most fit survives, even if it's ugly
- Rodents outlast dinosaurs...
  and bugs will outlast us all!



Pre-Siliconian Era | **Vectorian Period** | **Early Massivian Period** | **Sharedmemian Period** | **Late Massivian Period** | **Clusterian Period**

When Does Parallel Supercomputing Make Sense?

- When you need results **now!**
  - Top500 speed-up 1.4X every 6 months!
    Just waiting might work....
  - Optimizing your existing code helps a lot;
    do that first!
- When your application takes enough time per run
  to justify the effort and expense
- Clusters don’t change the basics...
  they primarily reduce the expense

What Is A Cluster?

- Not a "traditional" supercomputer...?
- Is *The GRID* a cluster?
- Is a *Farm* a cluster?
- A *Beowulf*?
- A supercomputer made from **interchangeable parts**
  - What are interchangeable supercomputer parts?
  - Some PC parts you don’t need (and shouldn’t want)

Types Of Hardware Parallelism

Pipelining	overlapped execution	automatic HW
Superscalar/VLIW/EPIC	1 instruction stream feeds multiple pipelines	automatic HW/SW
SWAR	SIMD (vector) Within A Register	SW vectorization
SMP	Symmetric MultiProcessor using shared memory	SW threads, etc.
Cluster	Homogeneous, dedicated, tightly-coupled PCs	SW messaging, etc.
"Beowulf"	Cluster using mostly commodity parts	SW messaging, etc.
Farm	Mostly homogeneous, dedicated PCs	SW messaging, etc.
Grid	Heterogenous, intermittently available PCs	SW messaging, etc.

Parts... Vs. In A Traditional Supercomputer

- Processors: primarily *AMD Athlon* and *Intel Pentium 4* (within 2X of best available @ very low cost)
- Motherboards, Memory, Disks, Video, etc.: many options
- Physical Packaging: choices here too
- No special "hooks" for parallel supercomputing....
  - Parts are tuned for PC use, compatability
  - E.g., long circuit path to network hardware

When Does It Make Sense To Build A Cluster?

Only when you are ready to use it.... ;-)

When Am I Ready To Use It?

- Is your application ready?
  - If not, is it in your power to make it so?
  - Can do code development/optimization using:
    - PCs
    - A small cluster
    - OPC (Other People's Clusters ;-)
  - Know what your application does
- Must coordinate purchase, assembly, software, etc. (e.g., don't buy processors too soon!)

Better Price/Performance => Bigger Machines

- Clusters aren’t nescessarily cheap;
  they are made of relatively cheap parts
- Expensive clusters can have fantastic performance
  - 1000s of fast processors, TBs of memory, PBs of disk
  - Scaling up isn’t trivial, but good designs exist
  - High *availability* (not *reliability*) via spares
- Easy upgrades prolong useful lifespan
- Nice "retirement plan":
  Break into smaller clusters & give to users

Clusters As Dedicated Lab Equipment

- You can have your own dedicated supercomputer... but it is your responsibility!
- Not much harder to live with than other lab equipment (although usually not a very good officemate)
- Interchangable parts mean low cost configurability; Engineer an *application specific* design
- High availability (not reliability) via spares
- Easy upgrades prolong useful lifespan
- Impress your friends, scare your enemies

Engineer Cluster To Meet Application Needs

- Know your application(s)
- Tune your application(s)
- Know your budget (money, power, cooling, space)
- Hardware configuration options (our primary focus here)
- Software configuration options

And now for something completely different...

What are your hardware options?

Total number of nodes or processors

- Not really a fundamental parameter
- Problem topology (square, cube, power of two, etc)
  - A function of code parallelism structure
  - Don’t buy processors application will not use
- Hot or cold spares
  - **You need to buy spares!**
  - *Hot Spares* minimize *MTTR*,
    increase power use and network complexity
  - *Cold Spares* get physically swapped-in
- Superset-configured spares
  can serve other purposes until needed...

Node hardware configuration

- The "best" nodes...
  do not always yield the "best" supercomputer
- Motherboards
- Kind of processor
- Number of processors per node
- Memory

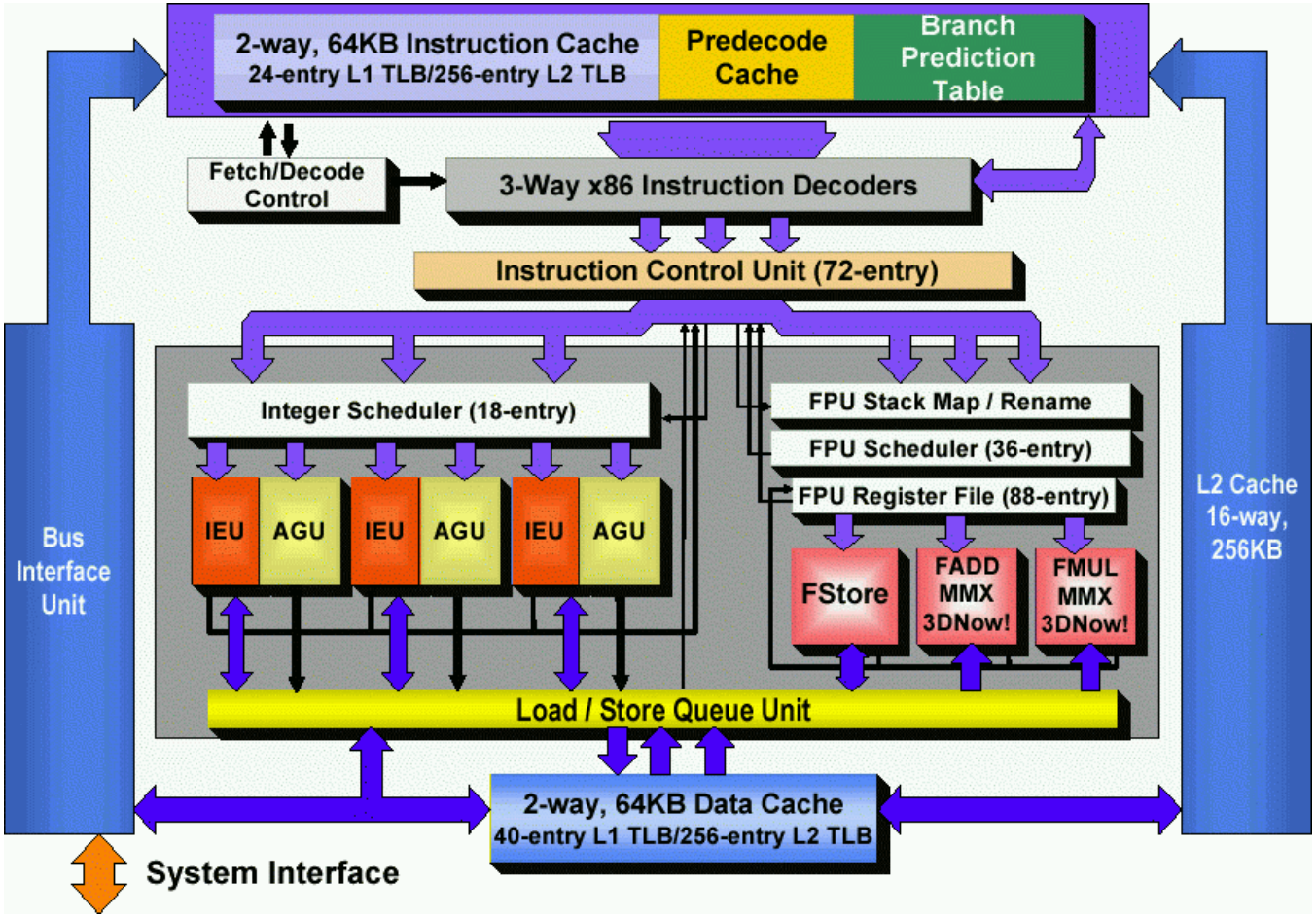
Motherboards

- Features you want:
  - Run without a video card, keyboard, & mouse
  - Temperature and fan RPM monitoring
- Features you do not want:
  - Overclocking support with "safe" reset
  - Extra stuff you will not use
    (e.g., SCSI RAID controller in diskless nodes)
- CMOS setup and other configuration issues
  - Don’t automatically reboot after power fail
  - Lots of performance-tuneable parameters

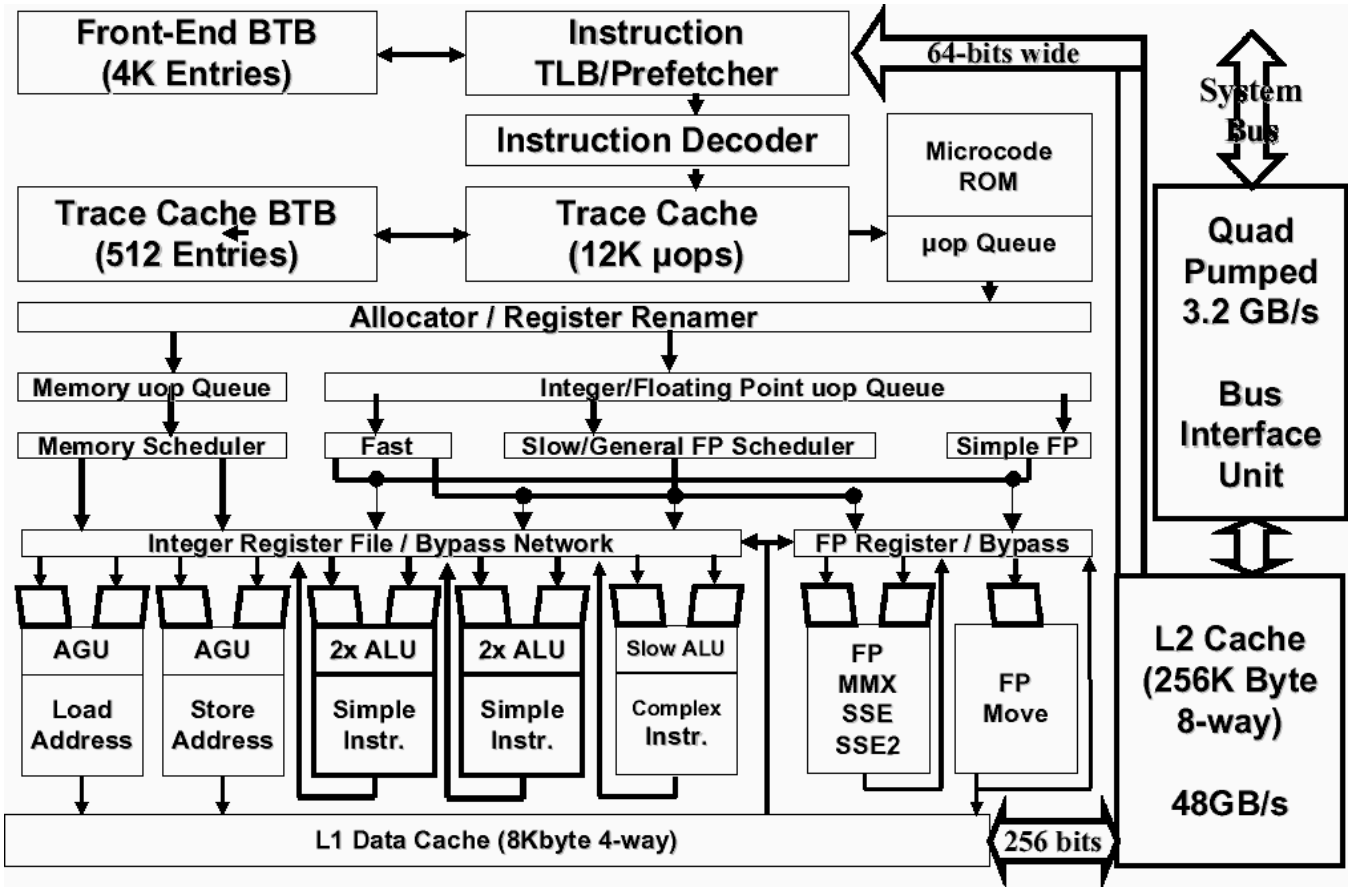
Kind of CPU

- Unless you have software licensing constraints, you really don’t care unless it’s slow or expensive
- Current top choices:
  - AMD Athlon: IA32, very aggressive ILP, cheap
  - AMD Opteron: x86-64, 64-bit Athlon upgrade
  - Intel Pentium 4: IA32, trace cache, fast clock
  - Intel Itanium 2: IA64, innovative but **very pricey**
  - Motorola G4: PowerPC, RISC, few vendors
  - Transmeta Crusoe: IA32, low power with lousy FP

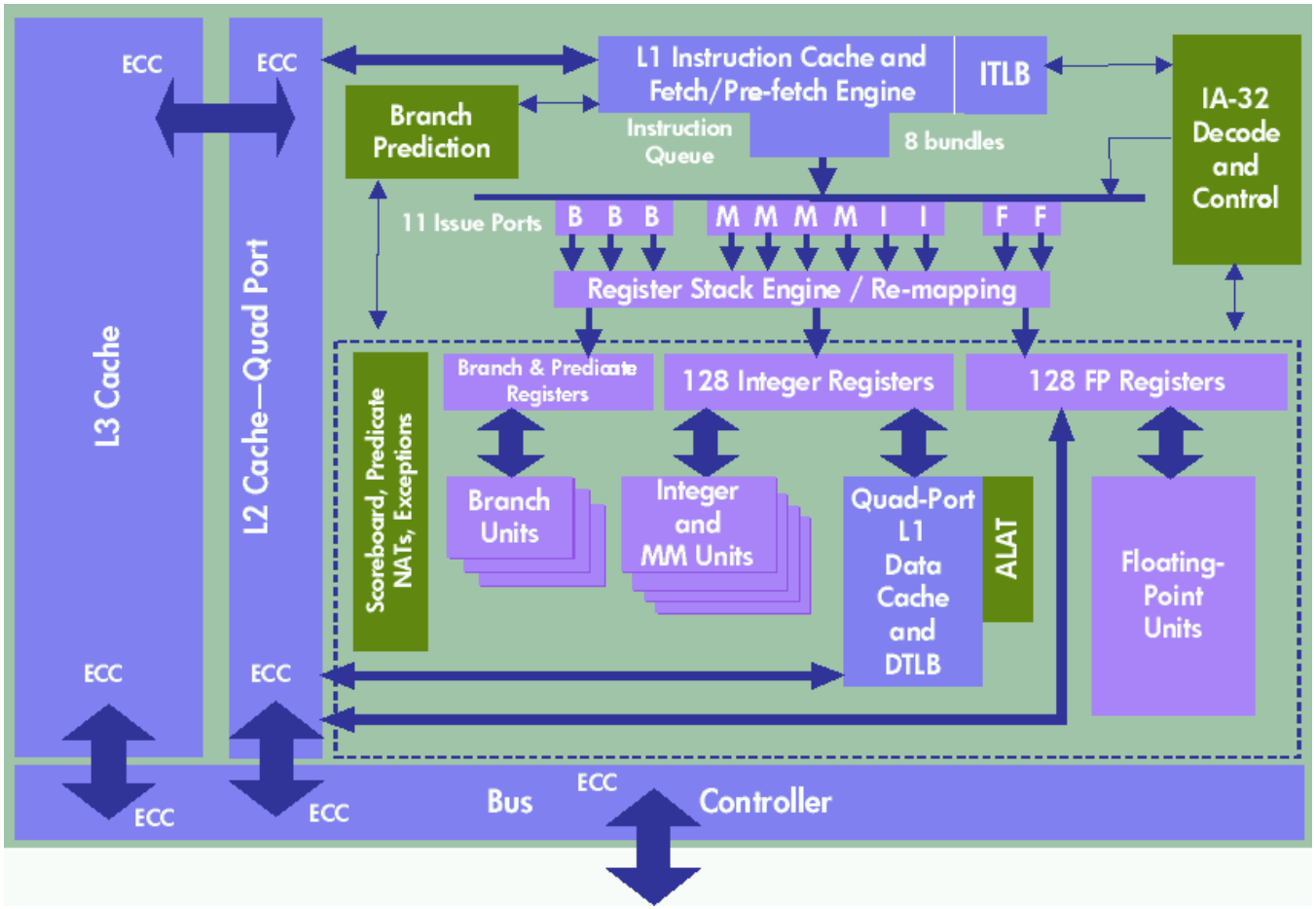
AMD Athlon


2-way, 64KB Instruction Cache
24-entry L1 TLB/256-entry L2 TLB
Predecode Cache
Branch Prediction Table
Fetch/Decode Control
3-Way x86 Instruction Decoders
Instruction Control Unit (72-entry)
Integer Scheduler (18-entry)
IEU
AGU
IEU
AGU
IEU
AGU
FPU Stack Map / Rename
FPU Scheduler (36-entry)
FPU Register File (88-entry)
FStore
FADD MMX 3DNow!
FMUL MMX 3DNow!
Load / Store Queue Unit
Bus Interface Unit
L2 Cache 16-way, 256KB
2-way, 64KB Data Cache
40-entry L1 TLB/256-entry L2 TLB
System Interface

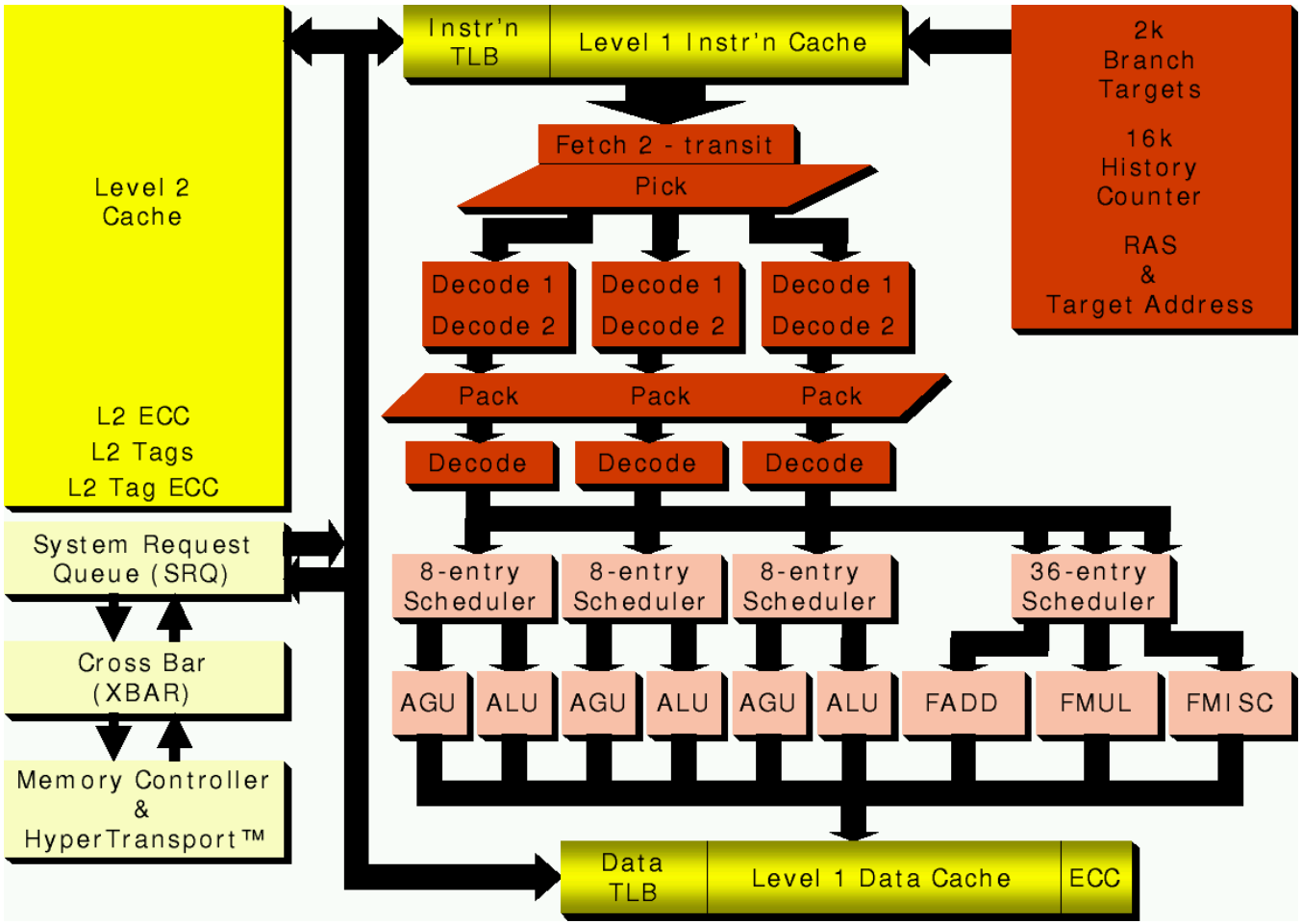
Intel Pentium 4



Intel Itanium 2



AMD Hammer (Opteron)



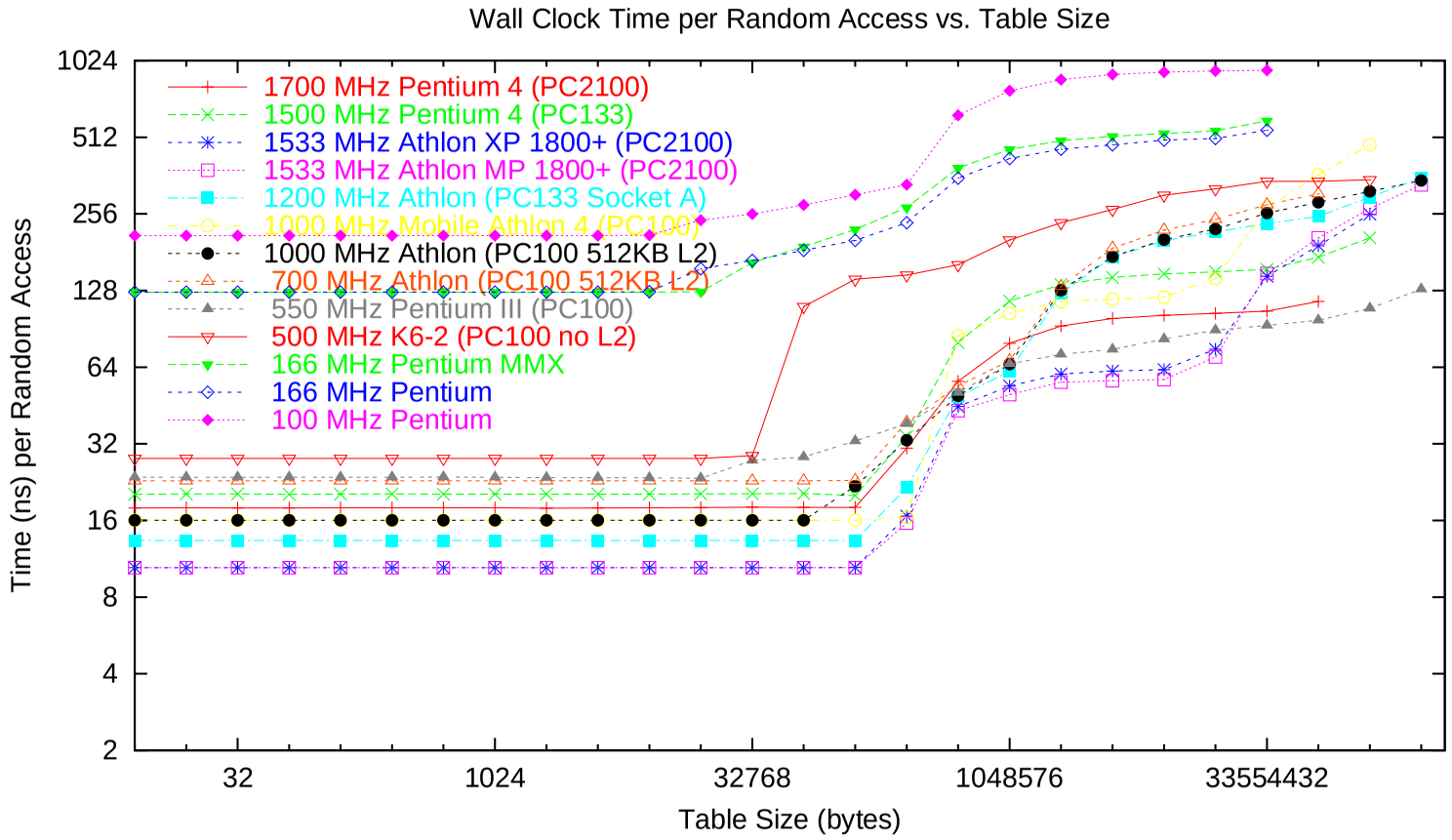
Number Of Processors Per Node

- Shared memory *SMP (Symmetric MultiProcessor)*
  - Physically vs. logically shared memory
  - Memory, I/O, network performance often divided, but so is the cost
  - *MPS* systems could be cheap, but are for servers
- Current 2-4 "head" systems are physically shared
- Intel Xeon and AMD MP not really "SMP processors"
- Pentium 4 soon will have *hyperthreading* (Intel’s name for MPS-compatible 2-way *multithreading*)
- AMD Hammer will have scalable memory bandwidth, latency for "remote" memory about 2X local

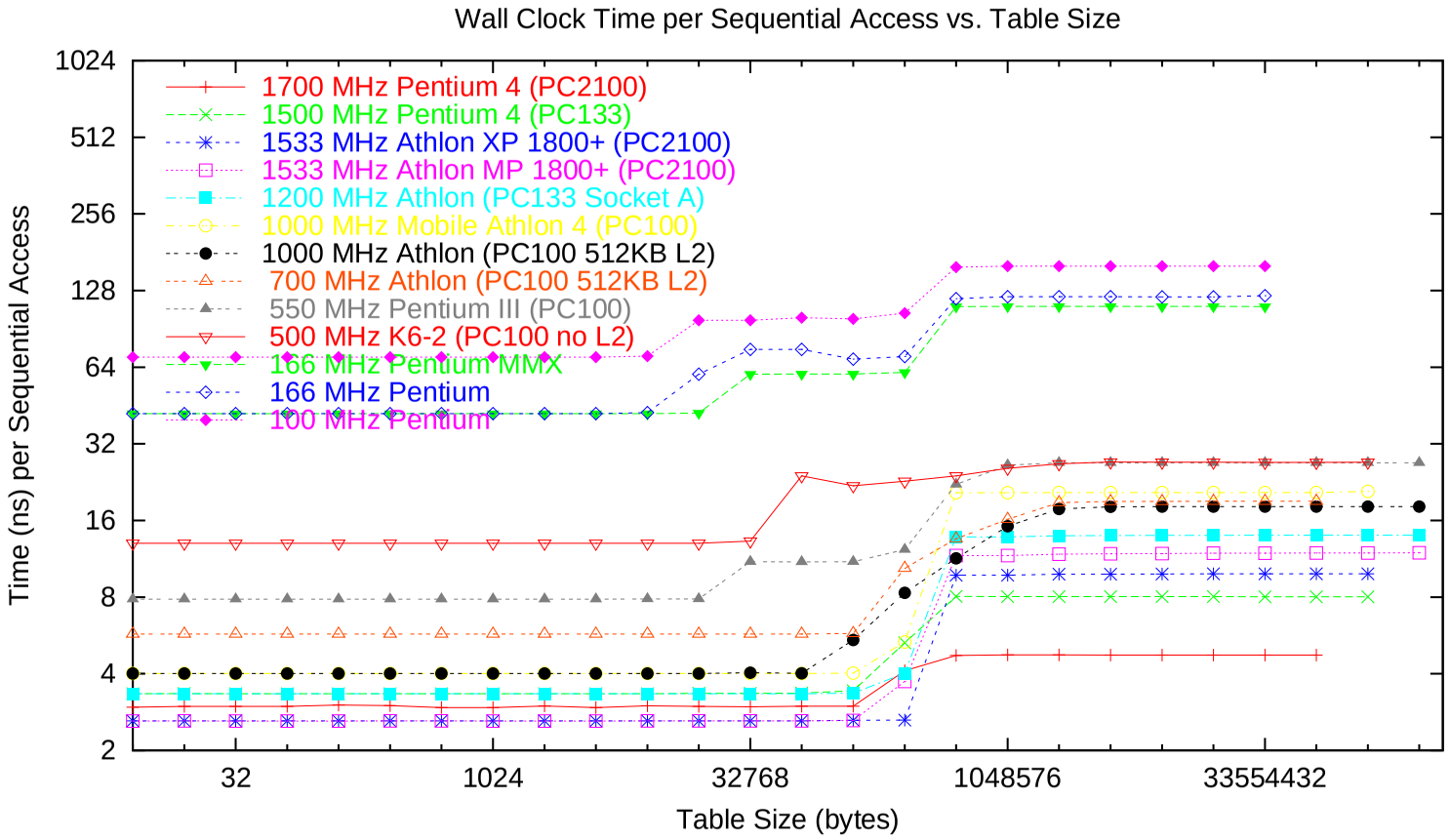
Memory

- 32-bit processors can address 4GB;
  most motherboards can hold 2GB or 3GB max
- Memory is hierarchical; **tune your code!**
  - Try to live in L1 or L2 Cache
  - Minimize TLB traffic
    (IA32 TLBs hold only 100's of 4KB pages)
  - Avoid referencing virtual memory
- Memory is probably the single weakest point in modern computer architecture and software technology...

How Slow Is Memory?



How Slow Is Memory?



Memory Parts

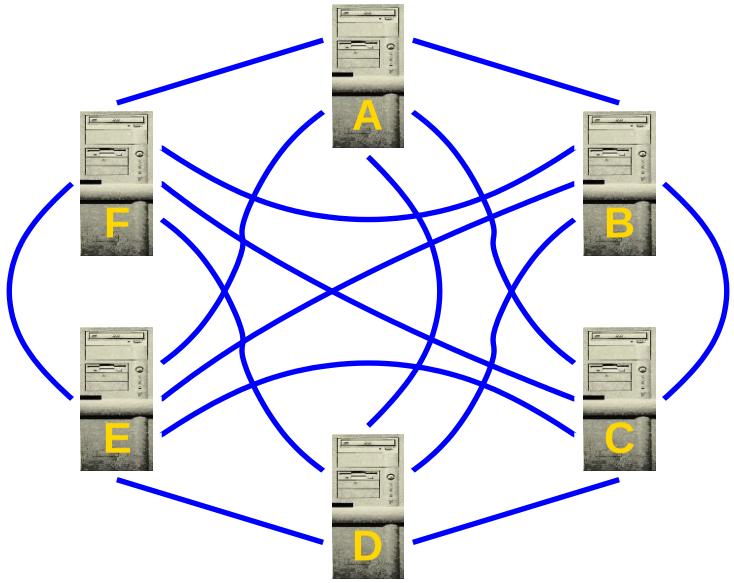
- The current major contenders:
  SDRAM, DDR, RDRAM
- Slow *FSB* can limit a fast memory;
  Slow memory limits a fast FSB
- *CAS* numbers: memory latency
- Registered vs. unbuffered
- *ECC?*

Network Design

- Assumptions
  - Links are bidirectional
  - Fixed maximum number of network interfaces per node
- Topologies
- Hardware
- Software

No Network

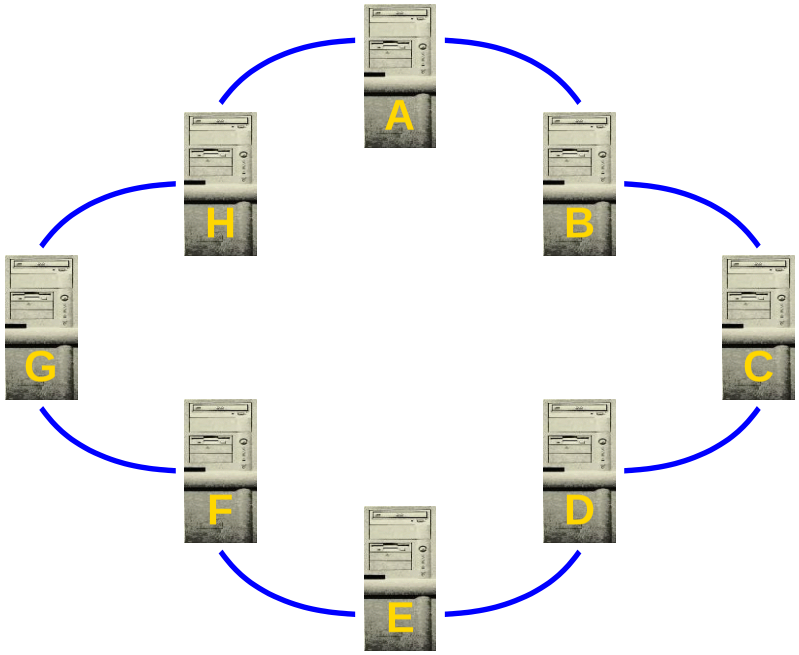
Direct Connections (Fully Connected)



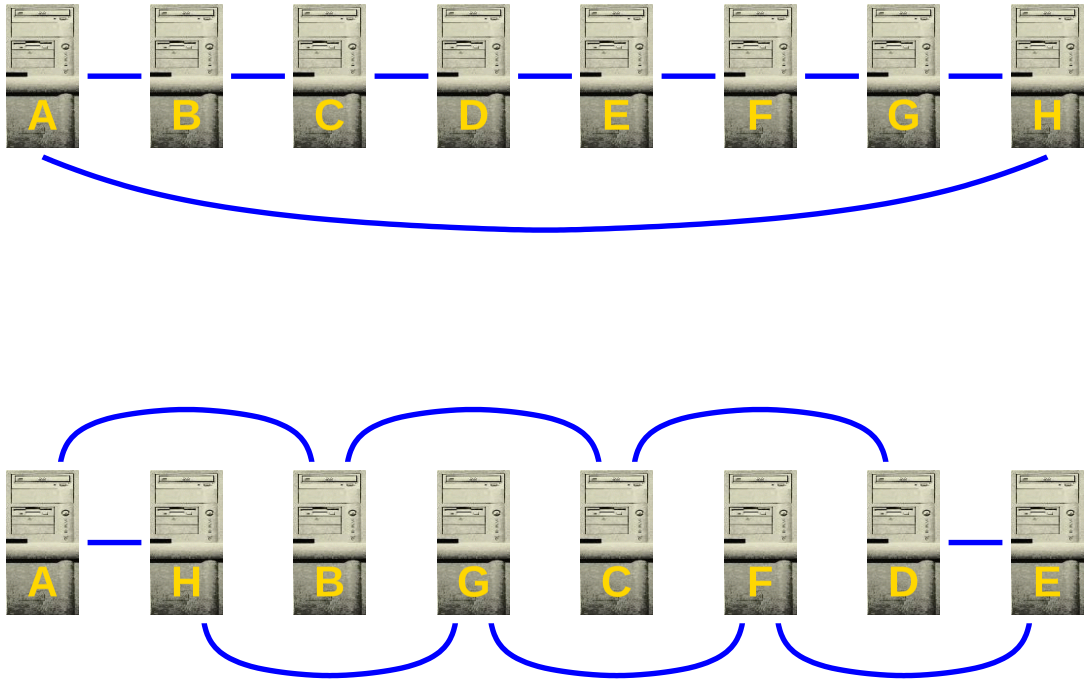
Toroidal Hyper-Meshes

- A generic term for the most common network topologies that use nodes as through-routing elements
- Includes rings, meshes, and hypercubes
- Can have any dimensionality: 1D, 2D, 3D, 4D, etc.
- Optionally have toroidal "wrap around" links
- Through-routing tends to imply high latency, and some processor overhead

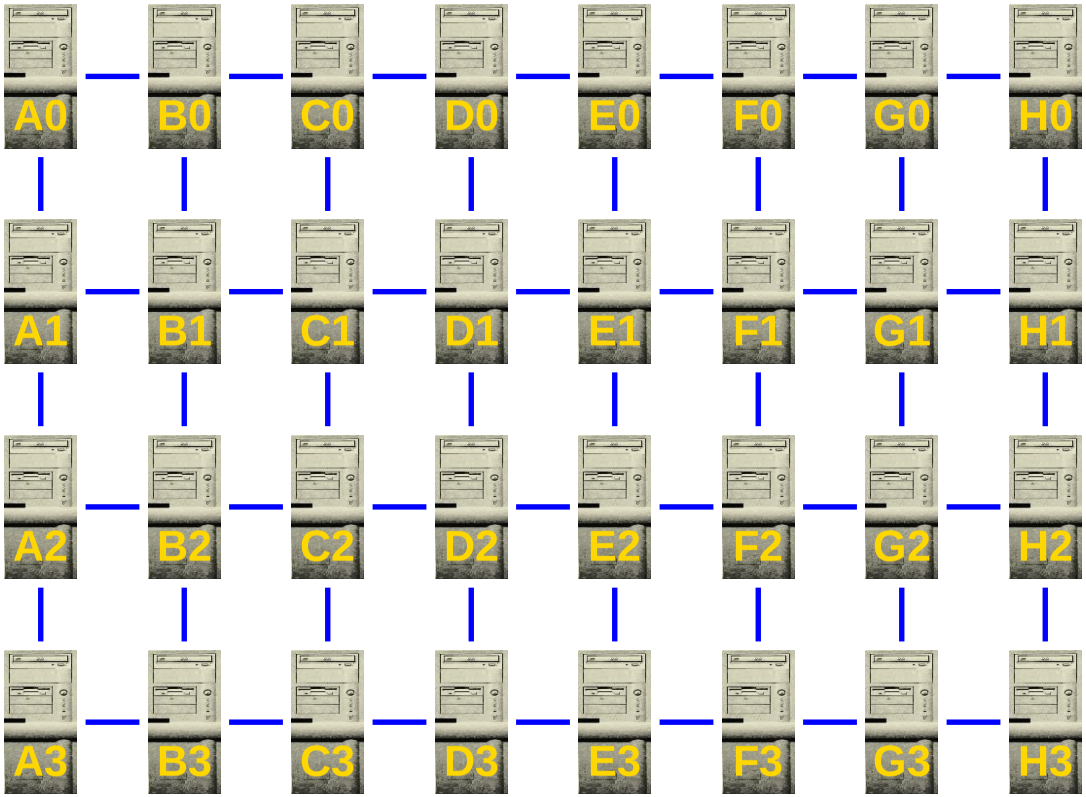
Toroidal 1D Mesh, AKA Ring



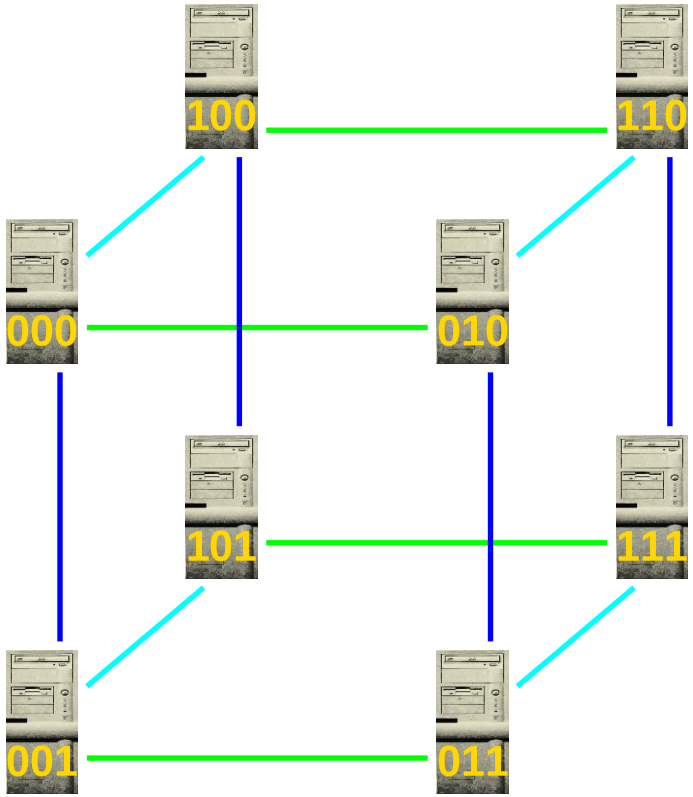
Physical Layout of a Ring



Non-Toroidal 2D Mesh



Non-Toroidal 3D Mesh or 3-Cube



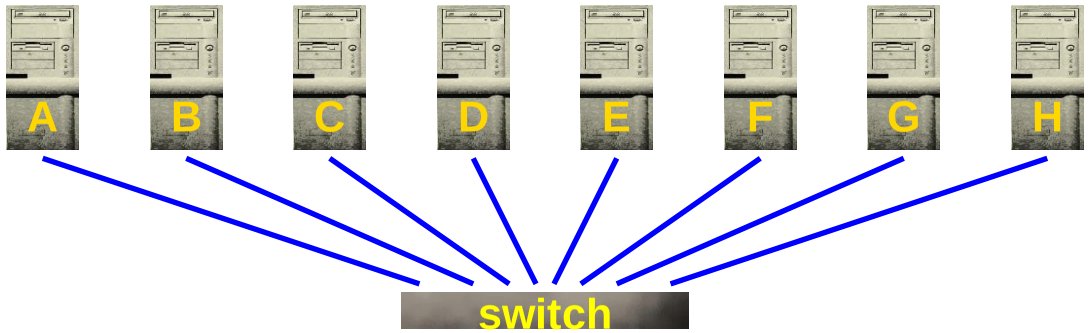
Switch Networks: What's a Switch?

- Hub: a shared bus in a box
- Full Duplex Repeater (FDR): two buffered busses
- Switch: interconnected switching elements
  - KxK switch chips interconnected (usually in a ring)
  - Bandwidth depends on interconnect;
    want *wire speed* and *non-blocking*
  - Latency a little higher than a hub or FDR
- Router: an expensive switch with smart routing abilities
  - Bandwidth can be helped by *trunking*
  - Latency often higher than dumb switch
    (due to routing processor)

The Ideal Switched Network

- No pair of nodes are separated by more than the latency of a single switch
- *Bisection Bandwidth* of each switch contributes to the total bisection bandwidth

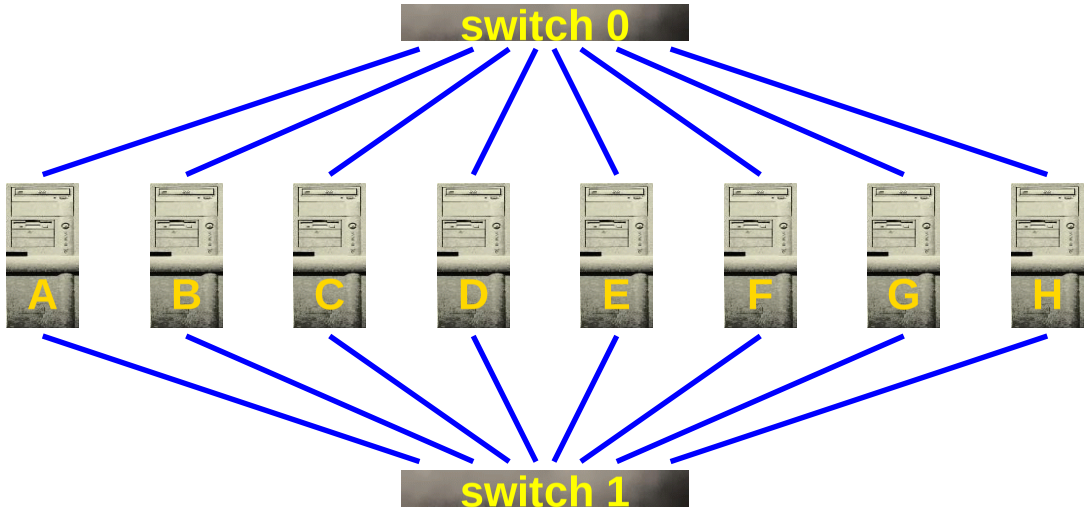
Simple Switch (8-port)



Channel Bonding

- The original "Beowulf" technology
- Use multiple, parallel, ethernet NICs
  - Dynamically load share between paths;
    Often, 2-5 100Mb/s NICs can match one Gb/s
  - Latency is not seriously degraded
- Beowulf implementation clones MAC addresses
  - Confuses switches if two NICs reachable
  - We are building a version that doesn’t
- Bonding performance limited by:
  NIC overhead, PCI bus, interrupt traffic

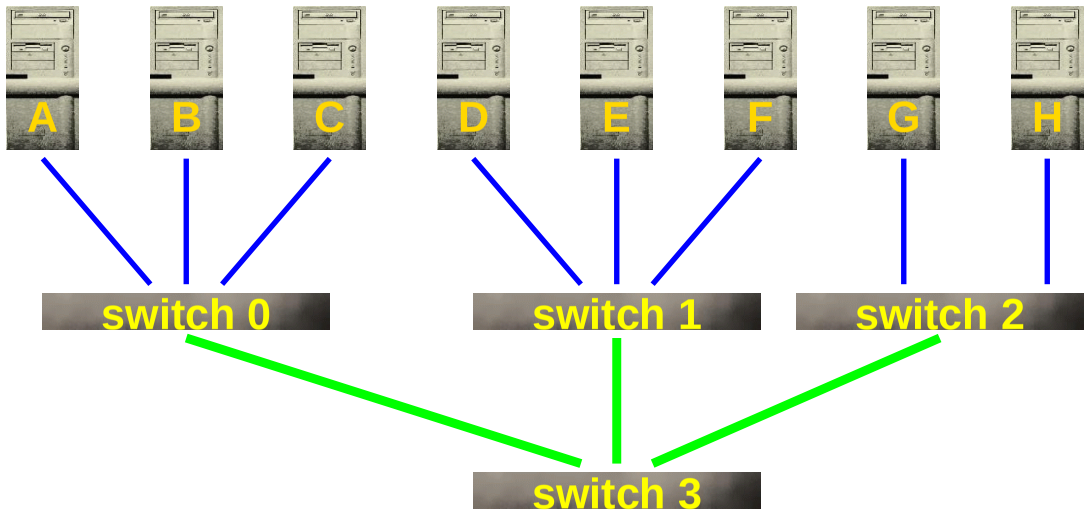
Channel Bonding (2-way, 8-port switches)



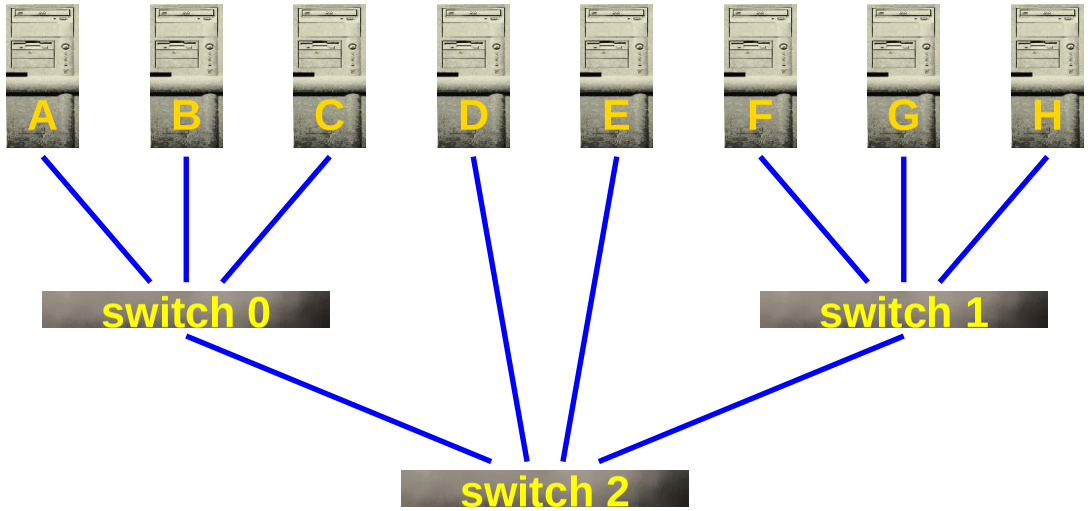
Switch Fabrics

- Number of nodes exceeds ports per switch...
  use multiple switches for connectivity
- Big switches tend to use internal ring-of-rings
  (often with mediocre bandwidth & latency)
- Most supercomputers, especially large clusters, use *hierarchical* switch fabrics
  - Most common are *trees* and *fat trees*
  - Want bisection bandwidth preserved at all tree levels

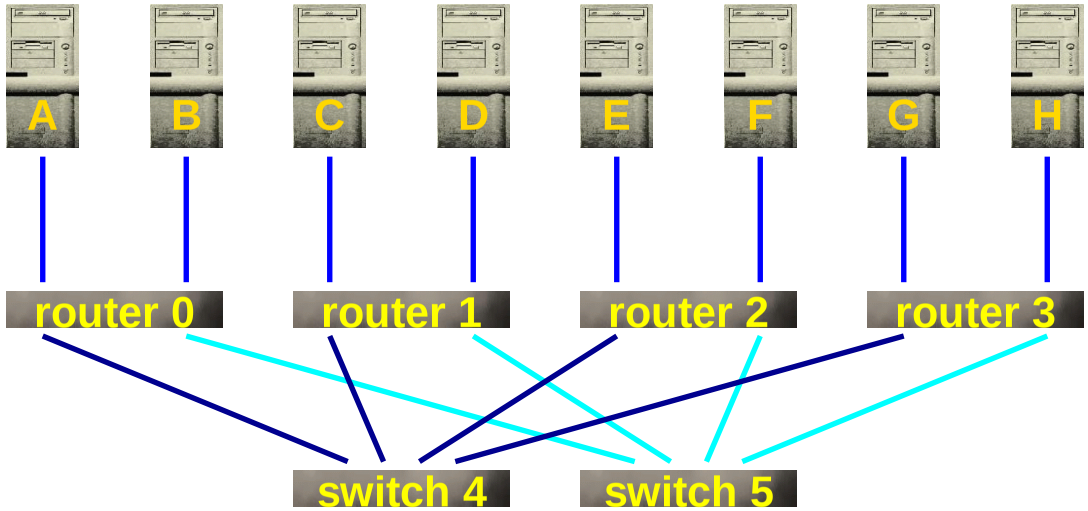
Tree (4-port switches)



A Better Tree (4-port switches)



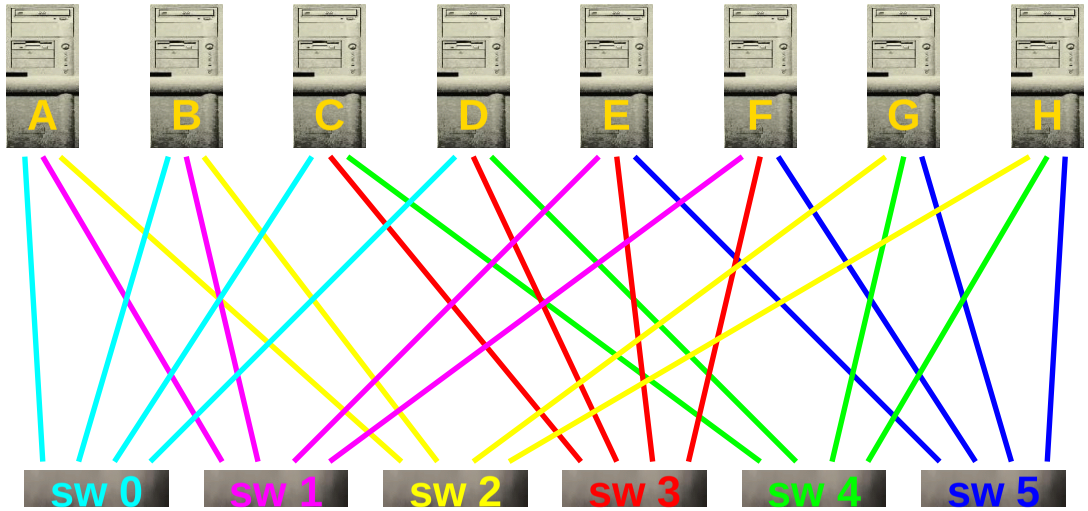
Fat Tree (4-port switches)



The Flat Neighborhood Network (FNN) Insight

- The ideal switch network uses one wide switch,
  but all PCs don’t have to share the *same* switch
- In a FNN:
  - Every pair of PCs shares at least one switch
    (each switch is a network "neighborhood")
  - Multiple NICs/PC connect to multiple neighborhoods
  - Topology is flat
    (no switch is connected to another switch)

Universal FNN (4-port switches)



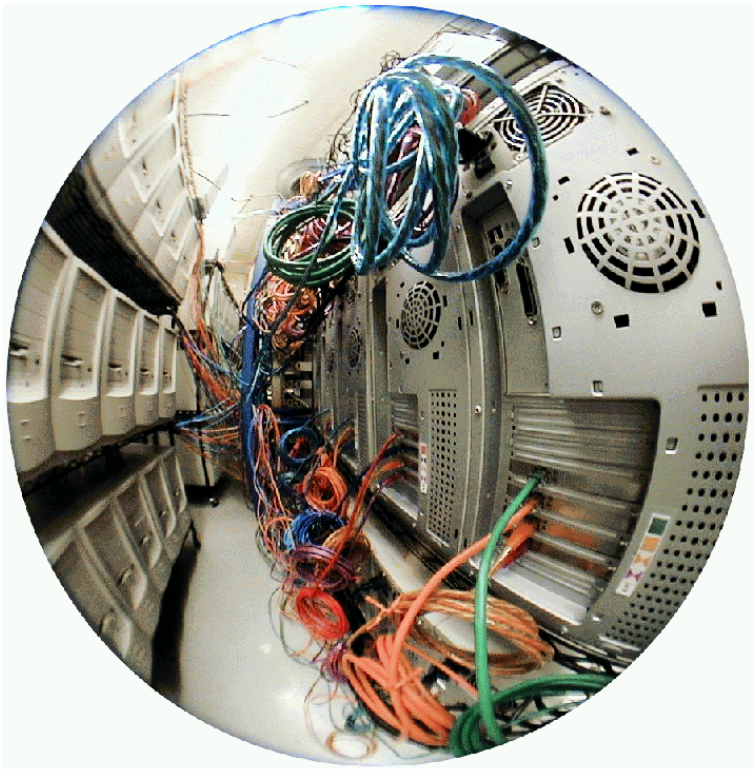
Flat Neighborhood Networks (FNNs)

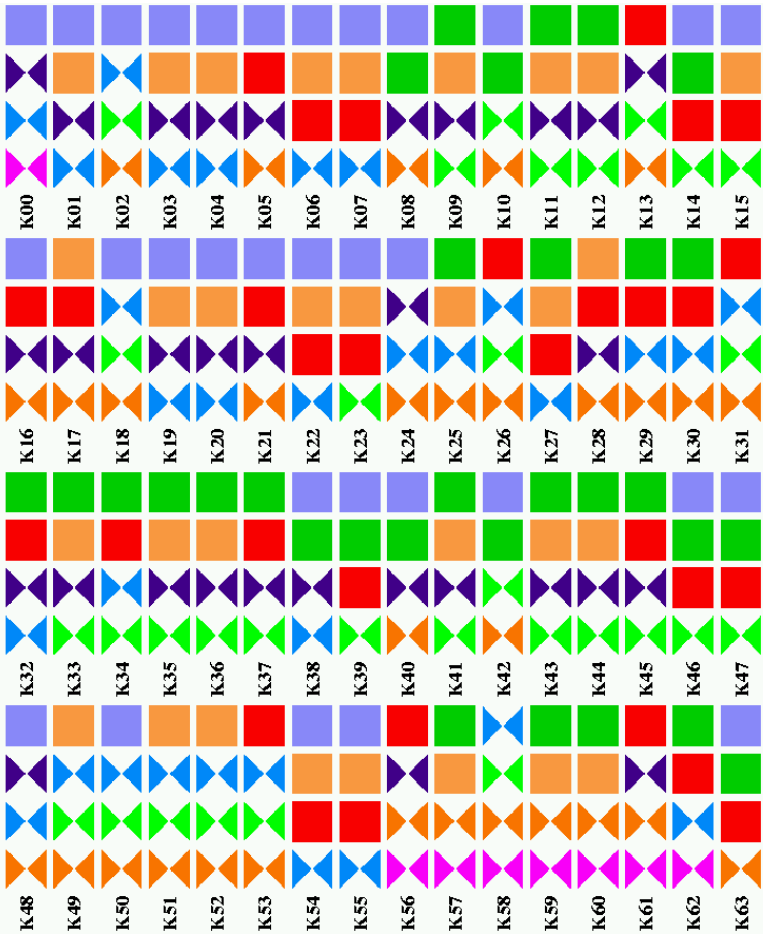
- Two flavors of FNN:
  - Universal FNN: all node pairs share neighborhoods
  - Application-Specific FNN:
    Only selected node pairs, very scalable
- FNN design is very difficult...
  - A GA creates FNN designs, configuration info
  - Optimize for specific communication patterns, sequences of patterns
- FNNs are usually asymmetric
- Node software is a superset of channel bonding

Flat Neighborhood vs. Fat Tree

- Typically, FNN uses no more switches than Fat Tree, but uses dumb switches and more NICs
- Flat vs. Fat Latency
  - 8 PCs, 4 port: 1.0 vs. (1.0+3.0*6)/7 = 2.7 switch delays
  - 64 PCs, 32 port: 1.0 vs. 2.5 switch delays
- Flat vs. Fat Pairwise Bandwidth
  - 8 PCs, 4 port: 1.29 vs. 1.0 NIC bandwidth units
  - 64 PCs, 32 port: 1.48 vs. 1.0 NIC bandwidth units
- Incremental improvement using FNNs: E.g., a 4th NIC/PC yields 1.86 NIC bandwidth units

The First FNN: KLAT2's Original Wiring





KLAT2's flat neighborhood network
Above: physical wiring
Right: neighborhood pattern

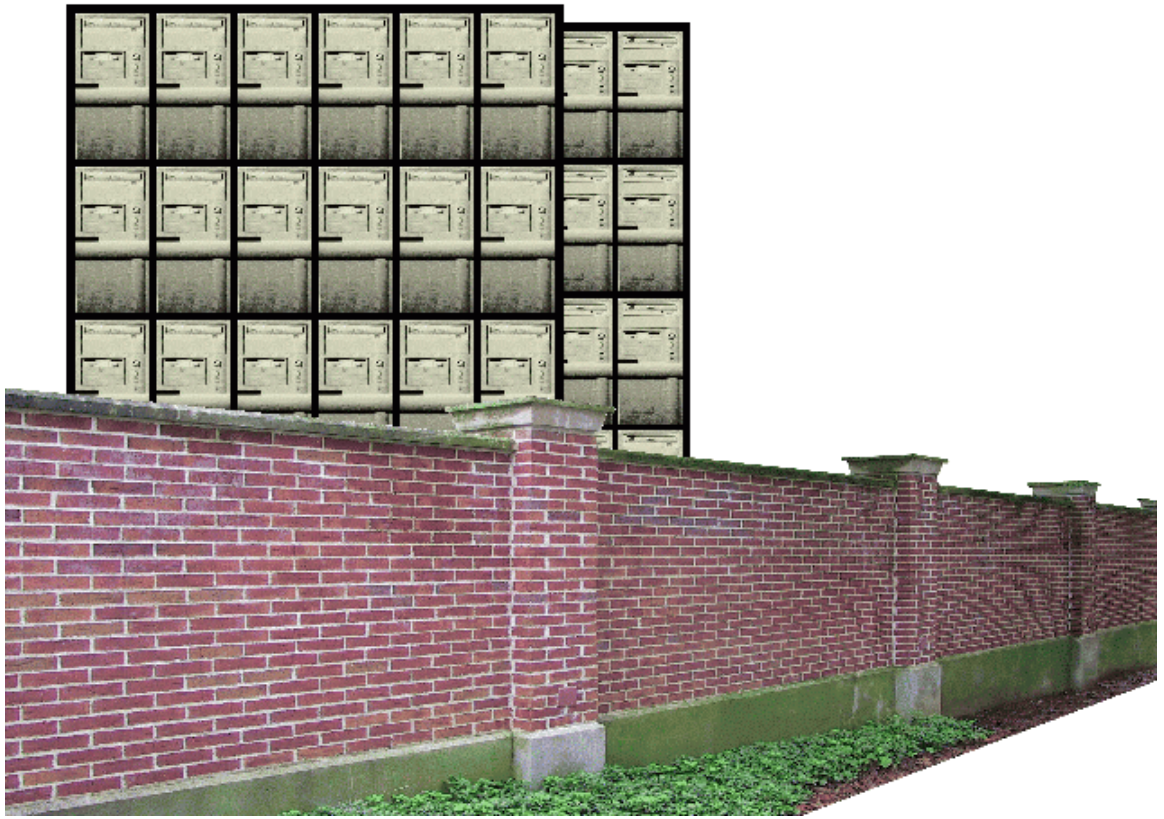
Using Multiple Networks

- Match networks to communication patterns:
  - a Mesh and a switch Tree
  - an application-specific FNN
  - a universal FNN and an Aggregate Function Network
- Match networks to service needs;
  - a SAN and a separate service LAN
  - a SAN, a LAN, and *Firewall(s)*

Topology Summary

- No Network: Easiest & best; pitty it doesn't scale
- Direct Connection:
  Easy, lots of NICs+cables, scaling problems
- Toroidal Hyper-Mesh:
  Researchy, cheap & scalable, lousy latency
- Simple Switch or Tree Fabric:
  Easy, bandwidth doesn't scale
- Fat Tree: Hard, need routers
- Channel Bonding (on any of the above):
  Easy, but configuration errors destroy performance
- FNNs: Researchy, best scalable topology

Put Clusters Behind Firewalls!



Why Put Clusters Behind Firewalls?

- Do you really want stray traffic pestering your cluster?
- A well-advertised large system is an exciting target for semi-skilled hackers
- Systems like Linux may be relatively secure, but only if you apply frequent security patches and upgrades...
- In short, it is better to have frequent upgrades of firewall software than to disrupt cluster service
- Use of `ssh` implies encryption CPU overhead;
  do you really want that on internal communications?

Network Hardware: LAN Technologies

- Cat5 100Mb/s Ethernet is a reliable commodity ($10 for a NIC, $300 for 32-port switch)
- Cat5 Gigabit Ethernet soon will be commodity
- Other choices: ATM, Firewire, USB2, etc.
- Heavyweight standard protocols (IP), but you don't have to use them (e.g., Gamma)

Network Hardware: SAN Technologies

- Custom and/or pricey (most >$1K per node)
- Myricom Myrinet: expensive smart NICs, dumb switches
- Dolphin SCI: 4/6 port NICs with 2D/3D mesh routing
- Other choices: cLAN, FC, HiPPI (Serial HiPPI), PAPERS, QsNet, dual-ended SCSI, etc.
- Latency 2-10X better than LANs
- Bandwidth 1-10X better than LANs

Software Configuration Issues

- Many cluster software packages...
  Look before you leap
- Remember to KISS:
  - Most cluster nodes don’t need email, httpd, etc.;
    They don’t need cron stuff like slocate either;
    Daemons suck resources even when they’re idle
  - Does a dedicated cluster need a job scheduler?
- How do nodes know who they are?
- Booting can be from local media or server(s)
- System can come from local media or server(s)

Booting & Loading The System

- From Local media:
  - A disk or CDROM can have boot and/or full system
  - A single floppy might do it
    (e.g., like the single-floppy Linux firewalls)
  - Easy to make local media customized per node
- From server(s):
  - Netboot from $8 PROM or $11 floppy drive
  - Much easier to keep things consistently updated
  - Large clusters are too much for one server;
    may need to skew requests, etc.

Performance Issues

- Local media operates in parallel
- Local media can hold network routing info, etc.;
  Nodes shouldn’t be ARPing their way around
- Actually, you don’t want anything ARPing,
  You also don’t want switches to unlearn routes
- Software reconfigure to replace a failed node
  can be nasty (and isn’t really automated)

Cluster Storage

- Disk attached to server
- Disk attached to each compute node
- *Storage Area Network*
  (the other SAN, the original definition ;-)

Why Do You Need Disks At All?

- You have disks somewhere, right?
- System configuration and boot information (netboot PROM, floppy, or CDROM can suffice)
- Swap space
- Application libraries and programs
- Application data

Remote/Parallel File System Features

- What you want:
  - Any nodes can access any file using the same filename
  - Access to files is fast
  - Multiple writers to different parts of a file is allowed and yields a coherent result
  - Files survive and remain available during node crashes
- Pitty it isn’t widely available yet....

NFS (Network File System)

- One server, many clients
- OK for infrequently accessed data and code
- Contention problems with large number of clients
- Coherency issues when writing files
- Commonly used in production systems
- Widely available as part of operating system kernel

PVFS (Parallel Virtual File System)

- Single metadata server
- File data striped across disks in cluster (user controllable)
- Supports normal Unix I/O (with kernel patch)
- User-level library to access PVFS directly
  (PVFS API or ROMIO MPI-IO)
- When a node fails data on it is lost until it returns
- Beta software
- Available under GPL
  `http://parlweb.parl.clemson.edu/pvfs/`

GFS (Global File System)

- Distributed metadata and file data, cached at clients
- Locking allows local writes
- Supports normal Unix I/O
- Journalling for recovery
- Commercial product (price negotiable?)
  `http://www.sistina.com/products_gfs.htm`

Distributed File Systems: AFS, Coda, InterMezzo, etc

- Designed for distributed applications
- Not really appropriate as a cluster file system
- Assumes one writer per file at a time is normal
- Some handle disconnected operation
- Good security support

Other Remote/Parallel File Systems

- Lustre - open source cluster file system in development, looks promising
- GPFS - cluster file system from IBM, open source, but only works on eServer cluster hardware
- Numerous research projects
  - Too many to name
  - Not ready, yet

Storage Hardware Options

- Hardware/software RAID
  - Many motherboards have RAID controllers
  - Linux software RAID works very well
    (IDE limits number of outstanding accesses)
- Backups
  (hard disks are the best backup for hard disks?)
- Disk drive choices
  - Latest IDE stuff is huge, fast, and cheap
  - SCSI has a better warranty, bigger price, spin faster
  - There are attached SAN disks too...

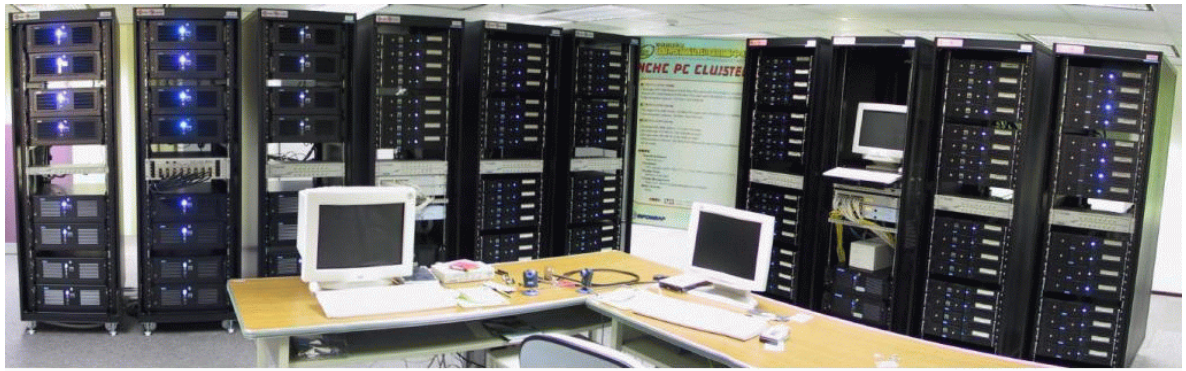
Physical Design Issues

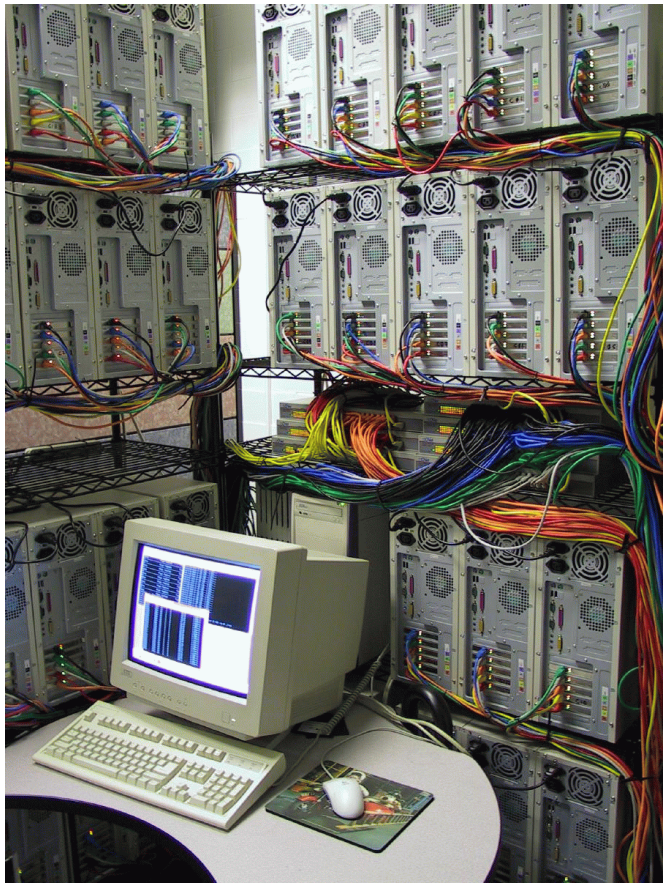
- Budget issues
  - Money
  - Power
  - Space
  - Cooling
- Packaging
  - Cases
  - Racks/shelving
  - Physical layout

Budget Issues

- Money is not the only limiting factor
- Often difficult or expensive to get more power in a room
- Most cooling systems designed for human occupants
  - Clusters dissipate lots more heat than people do
  - Clusters run when it’s cold outside
  - Clusters run during nights and weekends
- Need to have space for hardware and access to it

Rack Mount Vs. PCs On Shelves





Rack Mount Packaging

+ Rack can take less space per node

+ Racked clusters look like supercomputers

+ Often can hide most wiring

- Less space inside rackmount cases

- Cooling issues
(failure rates of high-speed fans, other components)

- More expensive with some specialized parts
(PCI slot-turning cards, cable mount extenders,
special power supplies, etc.)

Shelving Unit Packaging

+ Plenty of space inside mid-tower PC cases

+ Better cooling, access for maintenance
(open wire racks help airflow around cases)

+ Less expensive, wider selection of parts

- Takes more space per node
(about 24 nodes + network per shelving unit)

- Look distinctively "homebuilt" and "unprofessional";
all the wiring is exposed

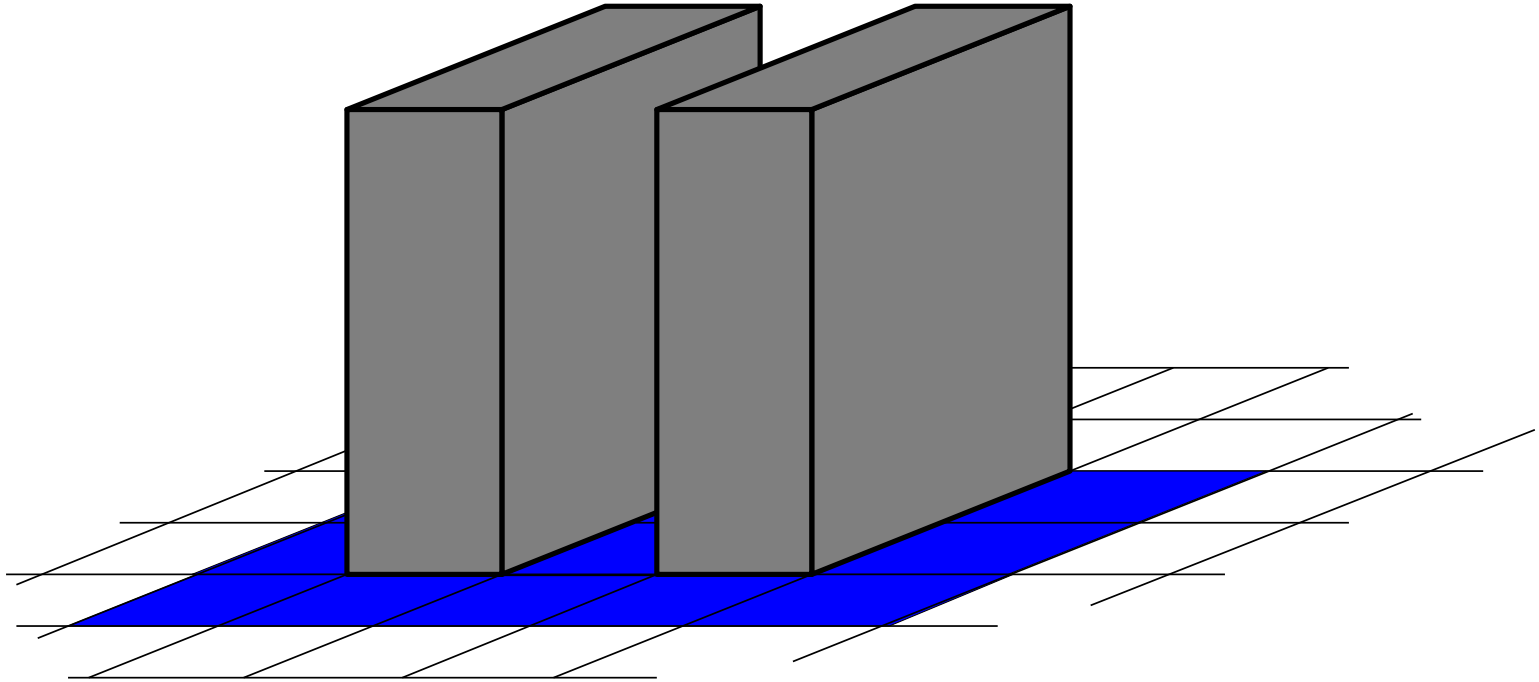
Space: The Final Frontier

- Think in terms of standard 2’x2’ raised floor tiles:
  - Even a 36u rack mount rack covers only 1
  - Standard 4’ wide shelving unit covers about 2
- Node cases are 1u-3u for racks,
  with some "blade" cases up to 24 nodes/3u;
  about 24 ATX PC cases fit on a shelving unit
- Typical network switch is 1u-2u rackmount;
  easily fit between rows of PCs on shelves
- Summary: racks look better and are ˜2X denser,
  but shelving units are much more practical

Physical Layout

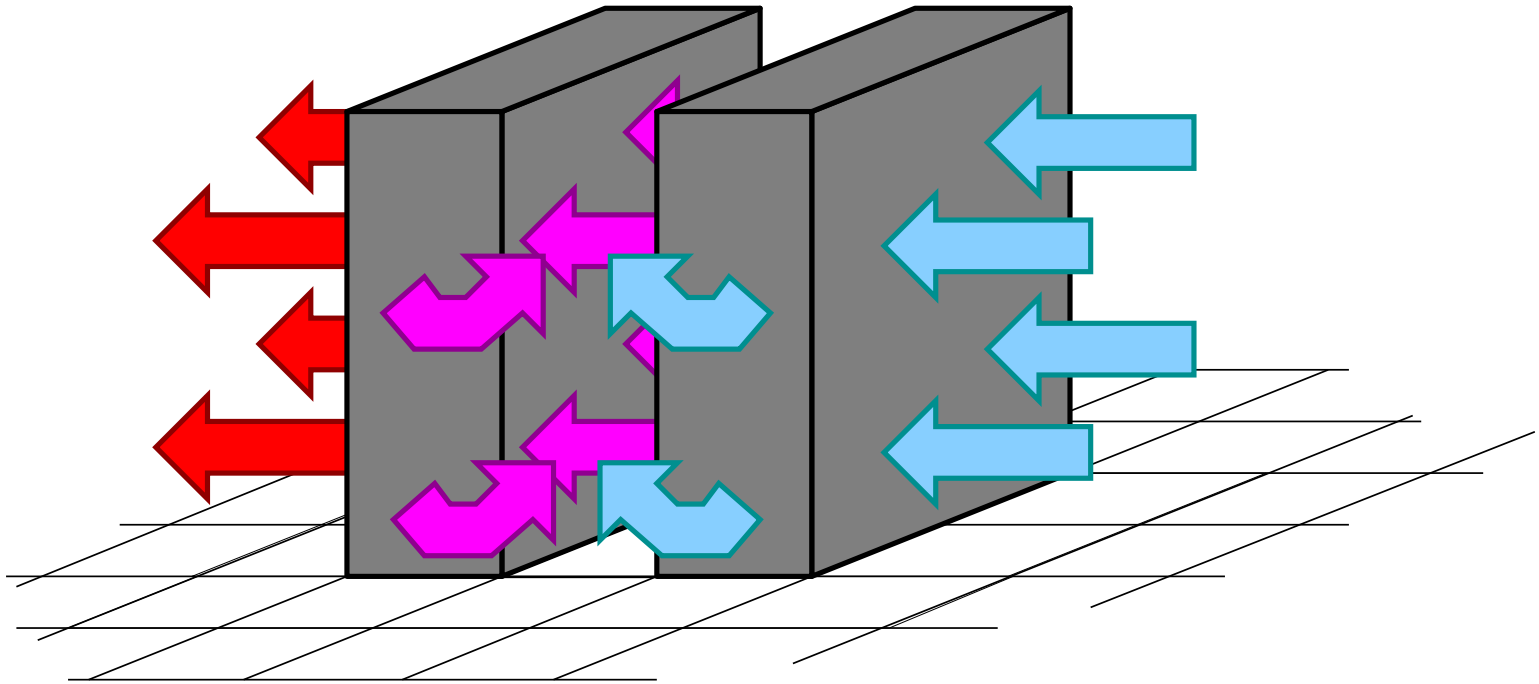
- Allow space for air flow and access
- Stack so that failed nodes can be removed
- Make maintenance easy, **neatness counts!**

Allow Space For Access



Minimum:
Connected 2’ wide walkways between & around

Allow Space For Airflow

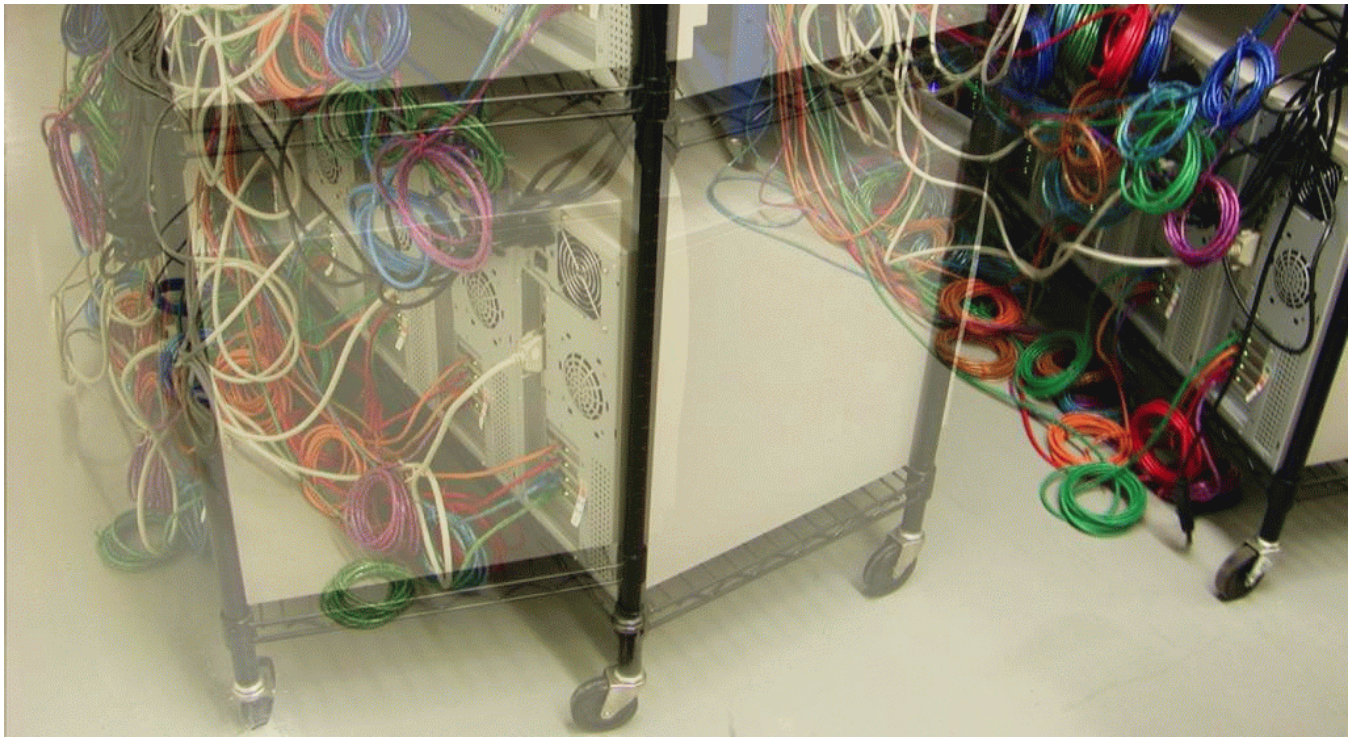


- Remove heat by airflow, air conditioning
- A typical modern processor dissipates about 70W!

Stacking For Easy Node Removal

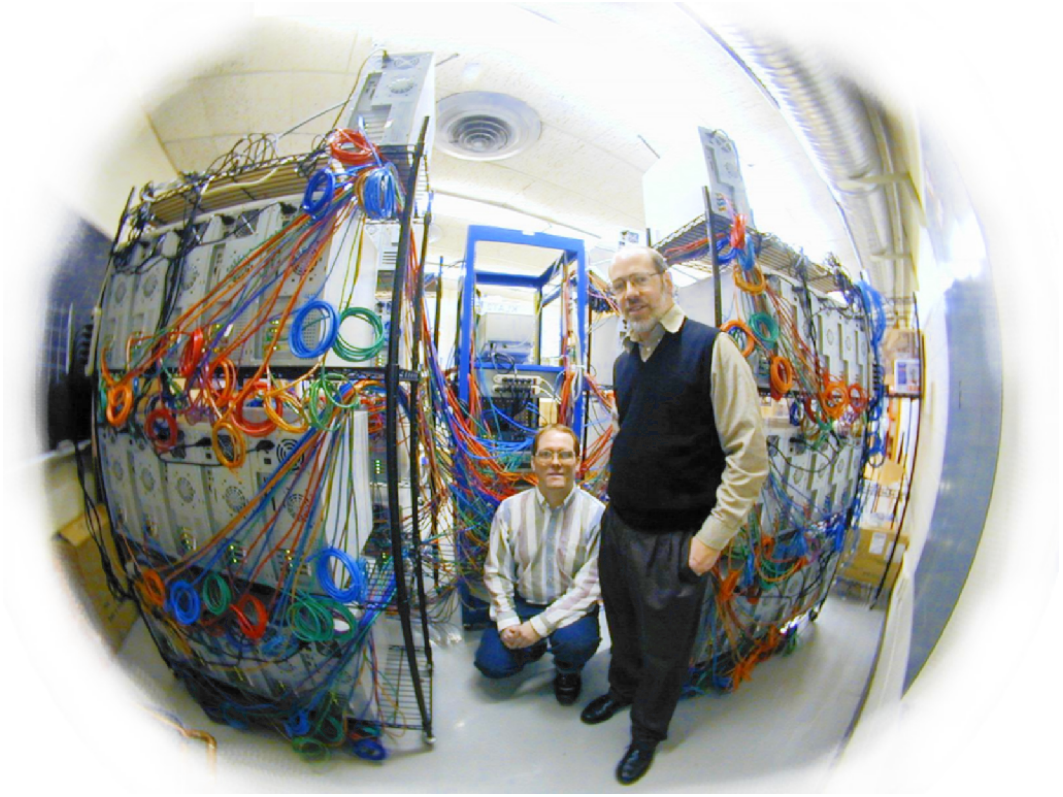
- Use shelves and leave enough space to get hands in
- High density & screw mounts complicate rack node access

The Importance Of Wheels...



- Can move for construction staging, general access
- Wheels must handle load, floor roughness

Neatness Counts!



Note use of twist ties, careful cable routing, etc.

Tuning For Your Application

- The cluster design space is **much larger** than what traditional supercomputers could support
- Can tune to a wide range of application characteristics
- Know your application...
  Well, at least make good guesses about what it does
- Apply common-sense analysis,
  Make approximate measurements if possible
- Search the complete design space with the CDR tool
  (don’t just look in your favorite corner ;-)

What Do You Need To Know?

- About your application(s)
- For the CDR tool
- For "tweaking" of the CDR tool’s results
- For impressing your friends
  (and scaring your enemies)

How Much Memory Do You Need?

- Code and data size (including libraries and OS)
- Simple Linux environments need about 16 MB
- Executable file size + size of dynamic libraries
- Data size depends on your program
- Estimate based on problem size
  (often from analysis or sequential code)
- Measure data + code size with "`ps -u`" (look at VSZ)

Memory Bandwidth Requirements

- *Memory Bandwidth*: rate at which data can be read from or written to memory (Mbytes/sec)
- Many applications are limited by memory bandwidth
- Want a way to compare memory bandwidth across machines with different processors and/or clock speeds
- Need a reference unit of computational effort
  - CDR calls them *GFLOPS*, but...
  - You can use any appropriate unit (e.g., Doom FPS)
- Memory bandwidth / GFLOPS,
  or bytes accessed / FLOP

Approximating Memory Bandwidth

```
double x[N], y[N], a[N][N];
register int i, j;

for (i=0; i<N; ++i) {
    y[i] = 0;
    for (j=0; j<N; ++j) {
        y[i] += a[j][i] * x[j];
    }
}
```

- First approximation:

$$\frac{4 \text{ accesses} \times 8 \text{ bytes/access}}{2 \text{ flops}} = 16 \text{ bytes/flop}$$

Accounting For The Compiler

```
for (i=0; i<N; ++i) {
    y[i] = 0;
    for (j=0; j<N; ++j) {
        y[i] += a[j][i] * x[j];
    }
}
```

- The compiler can keep y[i] in registers in the inner loop:

$$\frac{2\ accesses \times 8\ bytes/access}{2\ flops} = 8\ bytes/flop$$

Don’t Forget The Cache Memory

```
for (i=0; i<N; ++i) {
    y[i] = 0;
    for (j=0; j<N; ++j) {
        y[i] += a[j][i] * x[j];
    }
}
```

- References to cached data do not touch main memory
- If x[] is small enough to fit in cache:

$$\frac{1\ access \times 8\ bytes/access}{2\ flops} = 4\ bytes/flop$$

More On Memory Bandwidth

- Swapping i and j loop indices changes cache behavior
- Dividing matrices into blocks may improve performance, but complicates analysis
- In complex cases, measure L2 cache misses and number of flops with hardware performance counters
- Test "real" data sizes to get more realistic cache behavior

Local Disk Usage

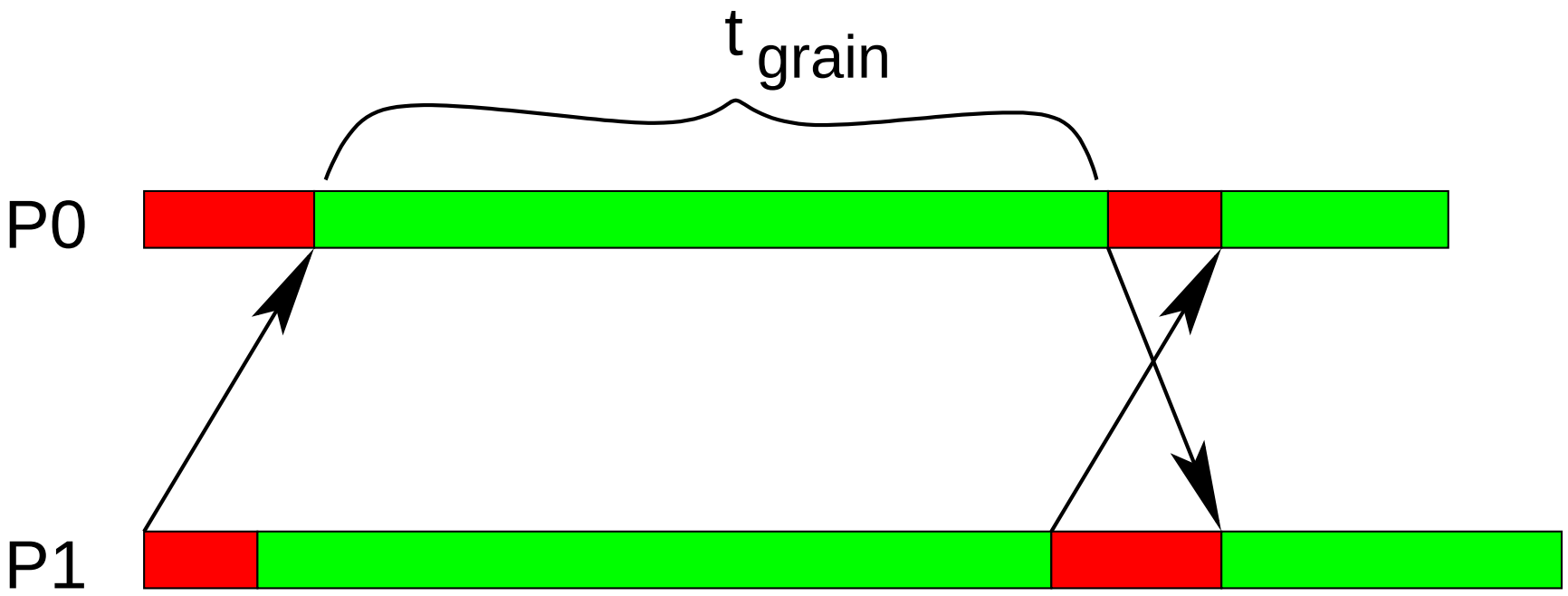
- Swap space rule of thumb:
  2x main memory size or none
- Might want swap space for:
  - Datasets too large to fit in main memory
  - Leaky programs or libraries
    (disturbingly common!)
- Other local data storage:
  - Caching local data
  - Checkpoint storage

Network Requirements

- *Bandwidth:* amount of data that can be sent per unit time
- *Latency:* delay between when data is sent and received
- To determine network requirements, need to know:
  - How often nodes will communicate
  - With how many others each node will communicate
  - How much data will each node send per communication

How Often do Nodes Communicate?

Granularity: the amount of time spent computing between receiving a message and sending a message



Determining Grain Size

- Grain size usually depends on:
  - Number of processors (*parallelism width*)
  - Processor speed
- How to determine t_{grain}
  - Estimate by examining code between communications
  - Measure in your running application
- Effectively, **t_{grain} is the maximum tolerable latency**

Affect of Granularity on Speedup

$$\textit{Single CPU execution time} = E_{single} = t_{grain} \times n_{grain}$$

$$\textit{Parallel execution time} = E_{parallel} = \frac{(t_{grain} + t_{comm}) \times n_{grain}}{n_{proc}}$$

$$\textit{Speedup} = \frac{E_{single}}{E_{Parallel}} = \frac{n_{proc}}{1 + \frac{t_{comm}}{t_{grain}}}$$

Need $t_{comm} << t_{grain}$ for good speedup

Estimating Communication Time

- Roughly speaking:

$$t_{comm} = latency + \frac{msg\ size}{bandwidth}$$

- Where latency and bandwidth are average values that a particular system design might yield for your application
- *Coordinality:* the number of nodes with which each node communicates regularly
- *Load balance:* do some nodes have to wait for others?
- t_{comm} is a very complex thing...

Notes on Latency and Bandwidth

- Raw network hardware specs. are not attainable
- Gigabit Ethernet sends 1 billion bits per second, but quite a few aren't your data bits
- PCI bus puts an upper limit on bandwidth
- Higher layer protocols add packet headers and processing
- Everything that touches data adds latency
  - Communication libraries (e.g., MPI)
  - Operating system
  - Network interfaces
  - Network switches

Physical parameters

- Space: go measure your room and the computing stuff (measure twice, build cluster once ;-)
- Power: what do you have, what does hardware want?
- Cooling: you don’t have enough :-(

A Few Words About Space...

- Your room was too small, right?
- You thought you could afford 8 processors...
  but the CDR suggests a 24+4 cluster!
- You can trade-off things to make it fit:
  - Rack mounts can be denser than shelving
  - SMP boxes can be as small as uniprocessor ones
  - One word: wheels.
- We suggest you ask for more space now.

Power: A Lot Of PC's Use A Lot

- What you have: count **dedicated** circuits
- What PCs want:
  - Power supply rating is not always helpful (max. rating)
  - Power used depends on what the program is doing; Make a bunch of measurements to find worst case
  - Use current meter to read actual current used
  - Measure startup power, use **power sequencing** (i.e., a bunch of separate power switches)

Keeping Your Cluster Cool

- Electronic components fail sooner when they run at high temperatures
- Room temperature is not the same as CPU temperature; e.g., 1 Athlon XP equals 2 "easy-bake" ovens
- Room temperature varies throughout room
- CPU temperature depends on case position and air flow
- Small (1u) fans run faster for the same air flow
- Monitoring software can watch temperature and fan speed

Systematic Cluster Design Using the CDR Tool

- Quickly getting a "reasonable" design
- Designing for incompletely-specified problems
- We just told you how to get the things you need to tell the CDR, and even a bit about how the CDR uses them....

Example Design

- Use of the CDR will be demonstrated live....
- If you’re here live, look up!
- If not, look at:
  `http://aggregate.org/CDR/`
  and go play....

What Problems Have Good Cluster Solutions?

- Any problem that can be split up and run efficiently on commodity nodes
- Individual processes limited to 4 GB
  (until a commodity 64-bit processor is available)
- Message latency >= 2 microsec for the best networks
  Message latency >= 5,600 clocks for Pentium4 2.8GHz
  Message latency >= 9KFLOP, 80-bit, for Athlon XP2800+
  (not very fine grain, eh?)
- Message bandwidth < PCI bus bandwidth
  (for 33MHz PCI, a little over 100MB/s)

Now That You've Got A Design, How Do You Get A Machine?

- Have a cluster vendor build it for you
- Buy and assemble the parts yourself
- Use reputable vendors
- Do not accept a mix of "equivalent" parts
- Getting through purchasing - you are not buying PC's
- Buy CPU's last (the price is most likely to drop)

Know When To Hold’Em And When To Fold’Em

- Normal component failures
  - Some "infant mortality" is normal:
    Send back parts for replacement or repair
    (just ignore dead parts too cheap to ship)
  - Some stuff "wears out" - especially fans;
    Get dual ball-bearing fans, etc.
- Incorrectly designed/constructed; 100% will fail soon
  - Run away;
    Switch to a different part as fast as possible
  - May need to redesign portions of the cluster...

What Do I Do With It Two Years From Now?

- Keep using it
- Upgrade it
  - Upgrades should keep the system homogeneous
  - Don’t waste time & money on trivial upgrades;
    Any upgrade should give significant new abilities
- Find somebody who actually wants your old cluster
  (or several who want pieces of it)
- Discard, I mean, Recycle it
  (disposable supercomputing!)

Any Questions?

http://aggregate.org/CDR/