CS395T: Introduction to Scientific and Technical Computing

Instructors:

Dr. Karl W. Schulz, Research Associate, TACC

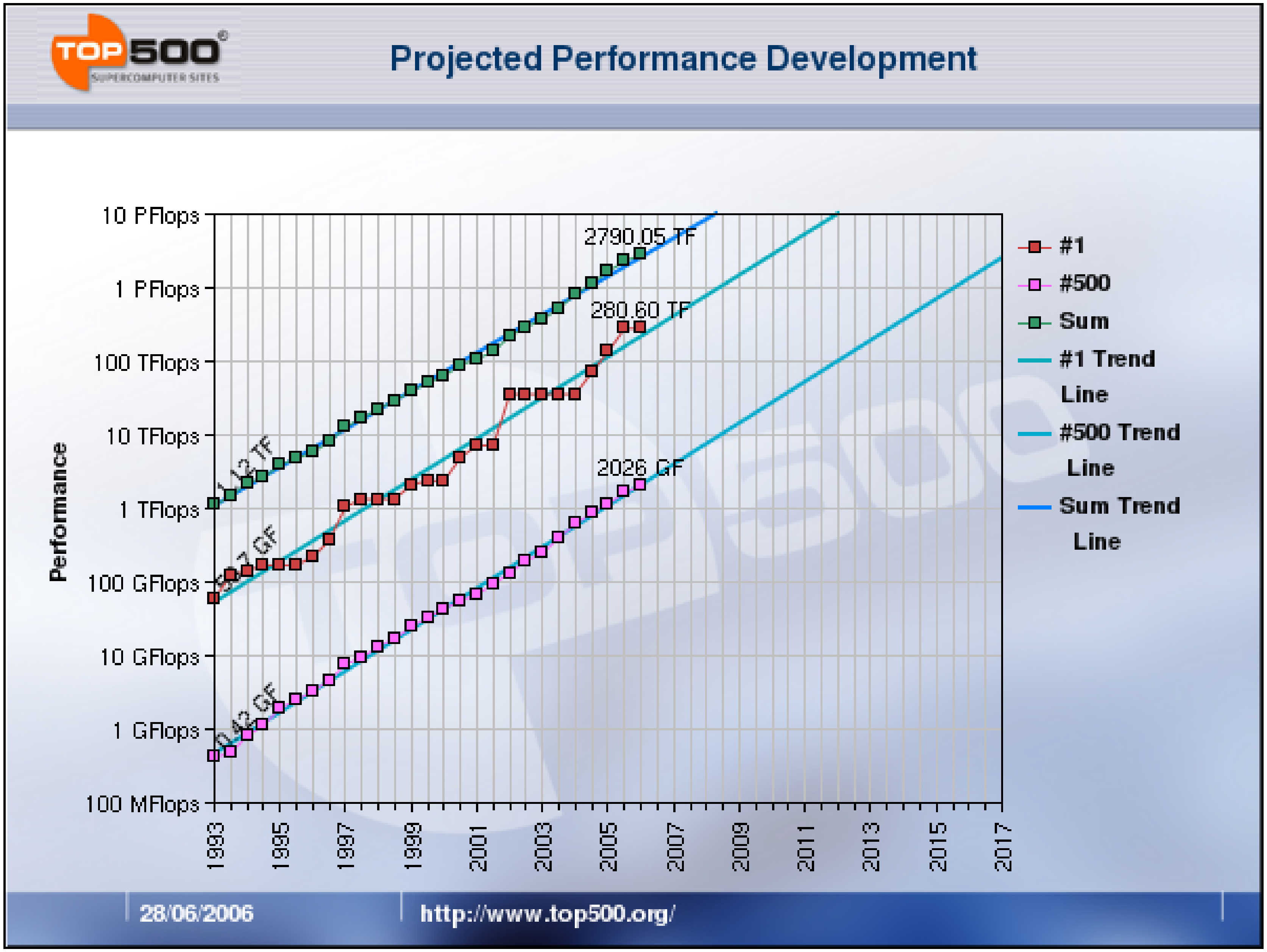
Dr. Bill Barth, Research Associate, TACC

Outline

- Parallel Computer Architectures
- Components of a Cluster
- Basic Anatomy of a Server/Desktop/Laptop/Cluster-node
  - Memory Hierarchy
    - Structure, Size, Speed, Line Size, Associativity
    - Latencies and Bandwidths
  - Intel vs. AMD platforms
    - Memory Architecture
  - Point-to-Point Communications between platforms.
- Node Communication in Clusters
  - Interconnects

Administrative Stuff

- Anybody need a syllabus
- Blackboard should be open to everyone
  - Please take the survey before Thursday



Projected Performance Development

28/06/2006 http://www.top500.org/

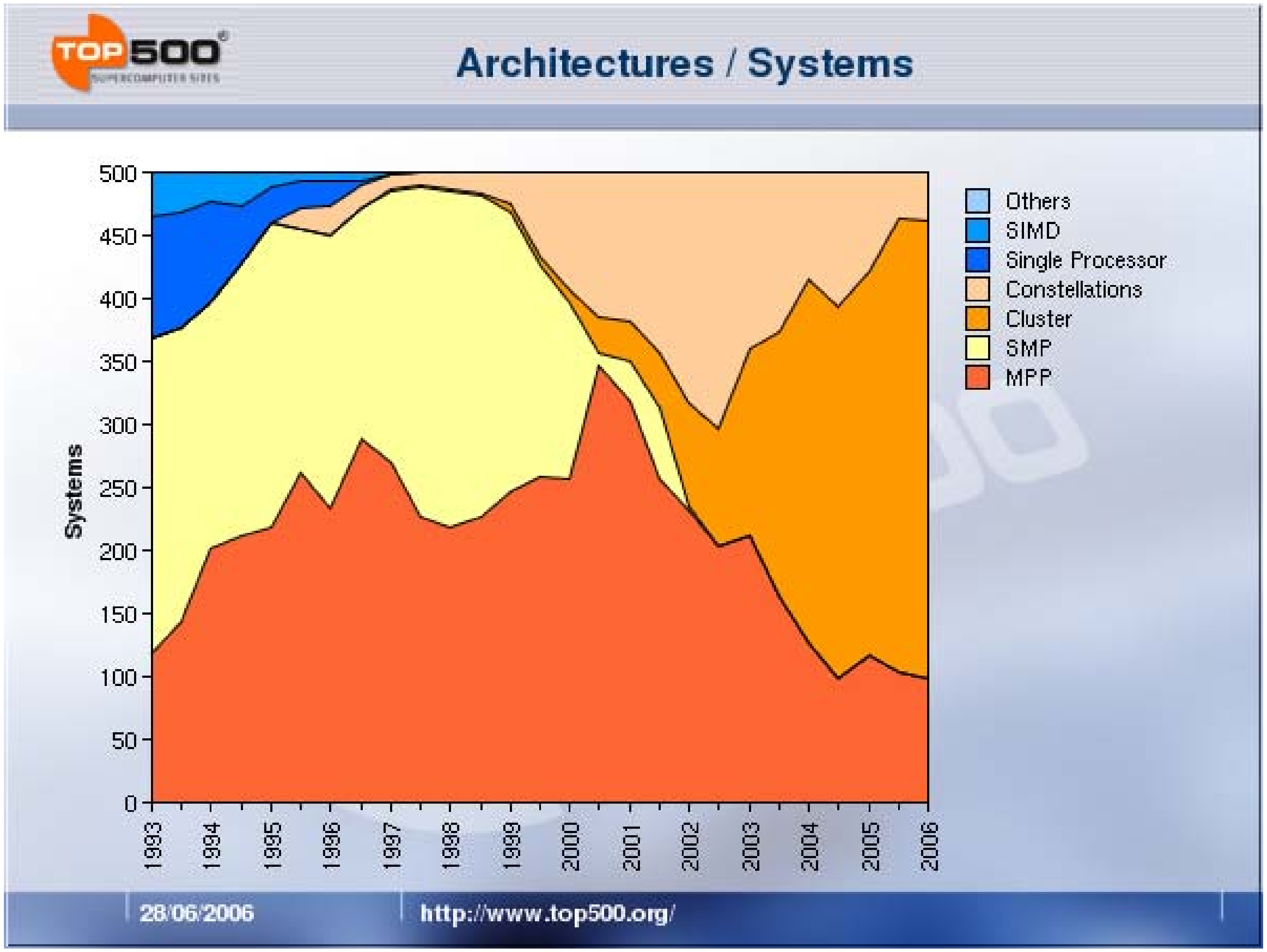
The Top500 List

- High-Performance LINPACK
  - Dense linear system solve with LU factorization
  - $2/3\ n^3 + O(n^2)$
  - Measure: MFlops
  - http://www.netlib.org/benchmark/hpl/
- The problem size can be chosen
  - fiddle with it until you find *n* to get the best performance
  - report *n*, maximum performace, and theoretical peak performance
- http://www.top500.org/

Parallel Computers Architectures

- **Parallel computing** means using multiple processors, possibly comprising multiple computers
- Until recently, Flynn's (1966) taxonomy was commonly used to classify parallel computers into one of four types:
  - (SISD) Single instruction, single data
    - Your desktop (unless you have a newer multiprocessor one)
  - (SIMD) Single instruction, multiple data:
    - Thinking machines CM-2
    - Cray 1, and other vector machines (there's some controversy here)
    - Parts of modern GPUs
  - (MISD) Multiple instruction, single data
    - Special purpose machines
    - No commercial, general purpose machines
  - (MIMD) Multiple instruction, multiple data
    - Nearly all of today's parallel machines

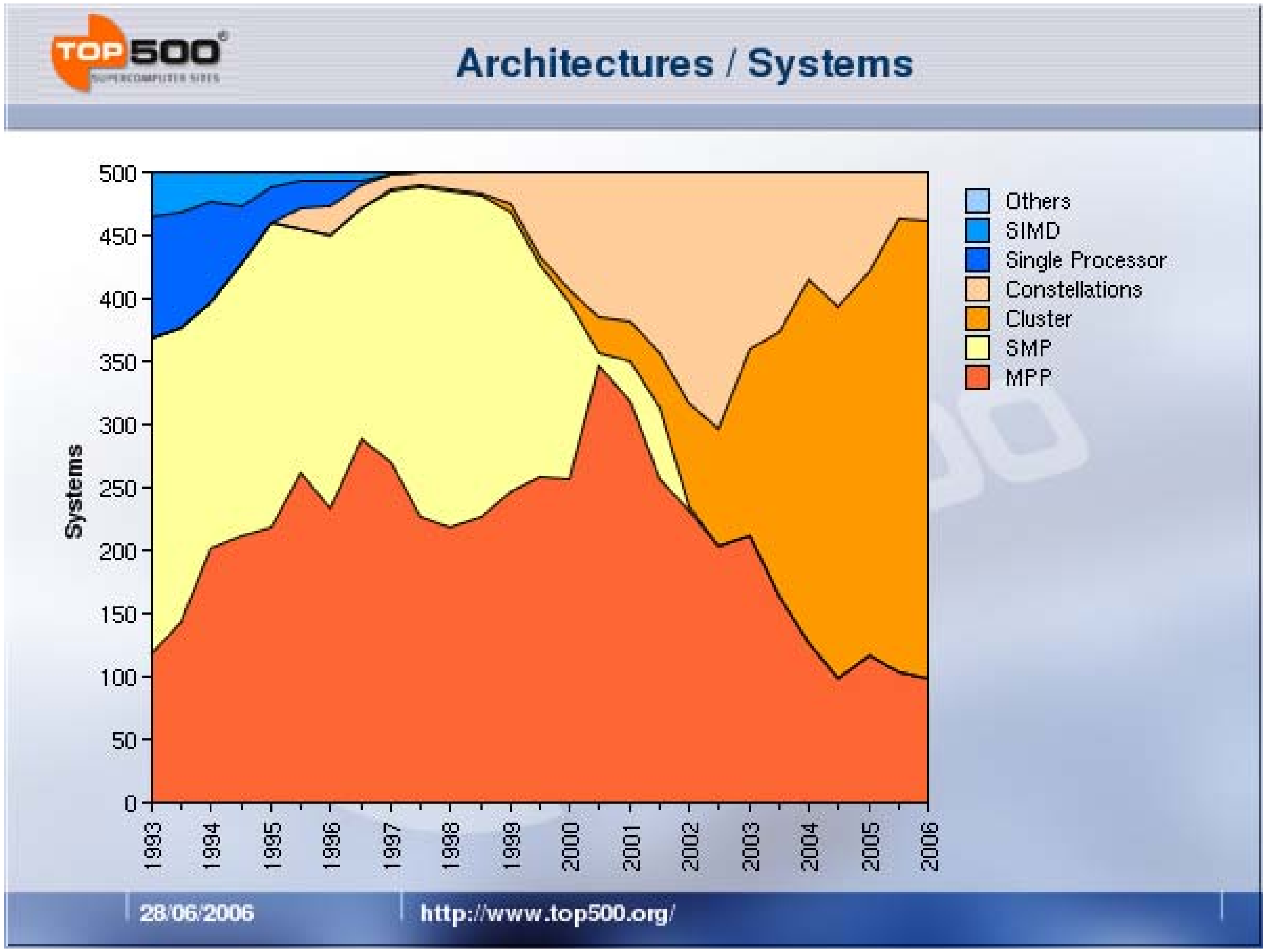
Top500 by Overall Architecture



Vector Machines

- Based on a single processor with:
  - – Multiple functional units
  - – Each performing the same operation
- Dominated early parallel market
  - – overtaken in the 90s by MPP, et al.
- Making a comeback (sort of)
  - – clusters/constellations of vector machines:
    - Earth Simulator (NEC SX6) and Cray X1/X1E
  - – modern micros have vector instructions
    - MMX, SSE, etc.
  - – GPUs

Top500 by Overall Architecture



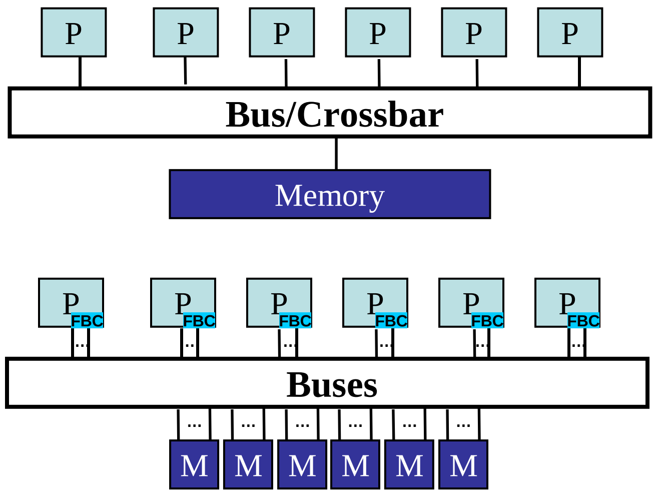
Parallel Computer Architectures

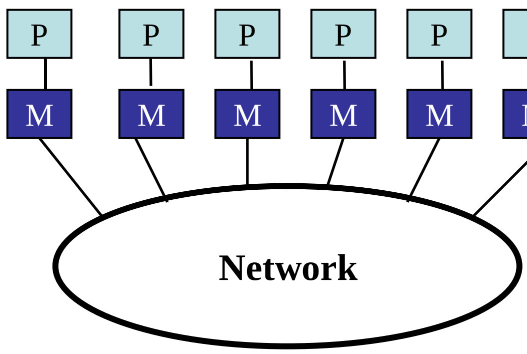
- Top500 List now dominated by MPPs, Constellations and Clusters
- The MIMD model “won”.
- A much more useful way to classification is by memory model
  - *shared* memory
  - *distributed* memory
- Note that the distinction is (mostly) logical, not physical: distributed memory systems could still be single systems (e.g. Cray XT3) or a set of computers (e.g. clusters)

Clusters and Constellations

- (Commodity) Clusters
  - collection of independent nodes connected with a communications network
  - each node a stand-alone computer
  - both nodes and interconnects available on the open market
  - each node may be have more than one processor (i.e., be an SMP)
- Constellations
  - clusters where there are more processors within the node than there are nodes interconnected
  - not very many of these any more (SGI Altix)

Shared and Distributed Memory





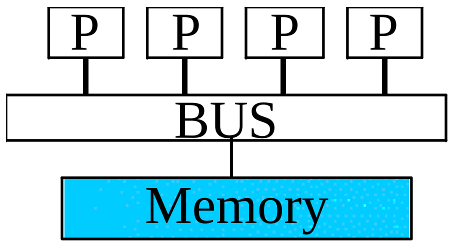
Shared memory: single address space. All processors have access to a pool of shared memory. (e.g., Single Cluster node (2-way, 4-way, ...), IBM Power5 node, Cray X1E)

Methods of memory access :
- Bus
- Distributed Switch
  (Fabric Bus Controller for each Processor)
- Crossbar

Distributed memory: each processor has it's own local memory. Must do message passing to exchange data between processors. (examples: Linux Clusters, Cray XT3)

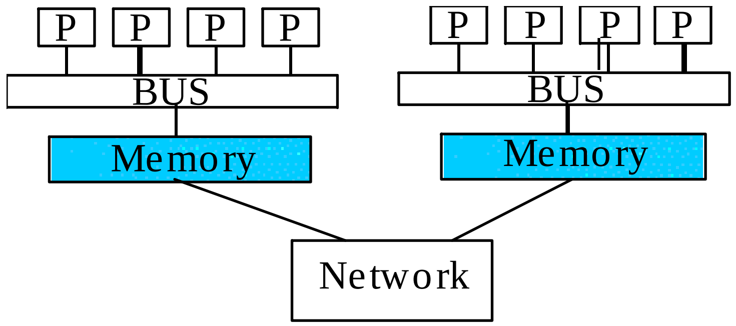
Methods of memory access :
- single switch or switch hierarchy
  with fat tree, etc. topology

Shared Memory: UMA and NUMA



Uniform Memory Access (UMA): Each processor has uniform access time to memory - also known as symmetric multiprocessors (SMPs) (example: Sun E25000 at TACC)

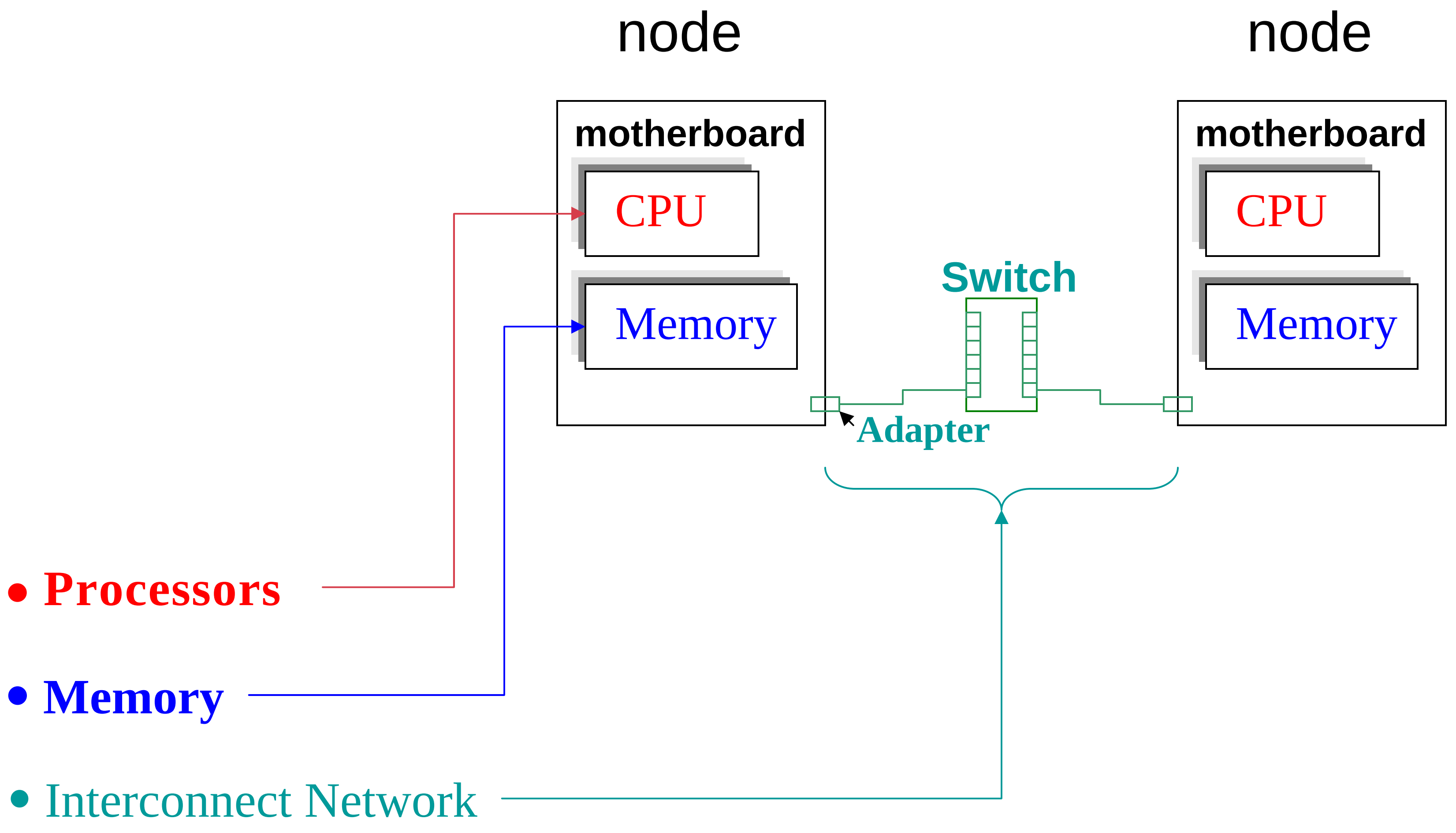
Non-Uniform Memory Access (NUMA): Time for memory access depends onlocation of data; also known as Distributed Shared memory machines. Local access is faster than non-local access. Easier to scale than SMPs (e.g.: SGI Origin 2000)



Memory Access Problems

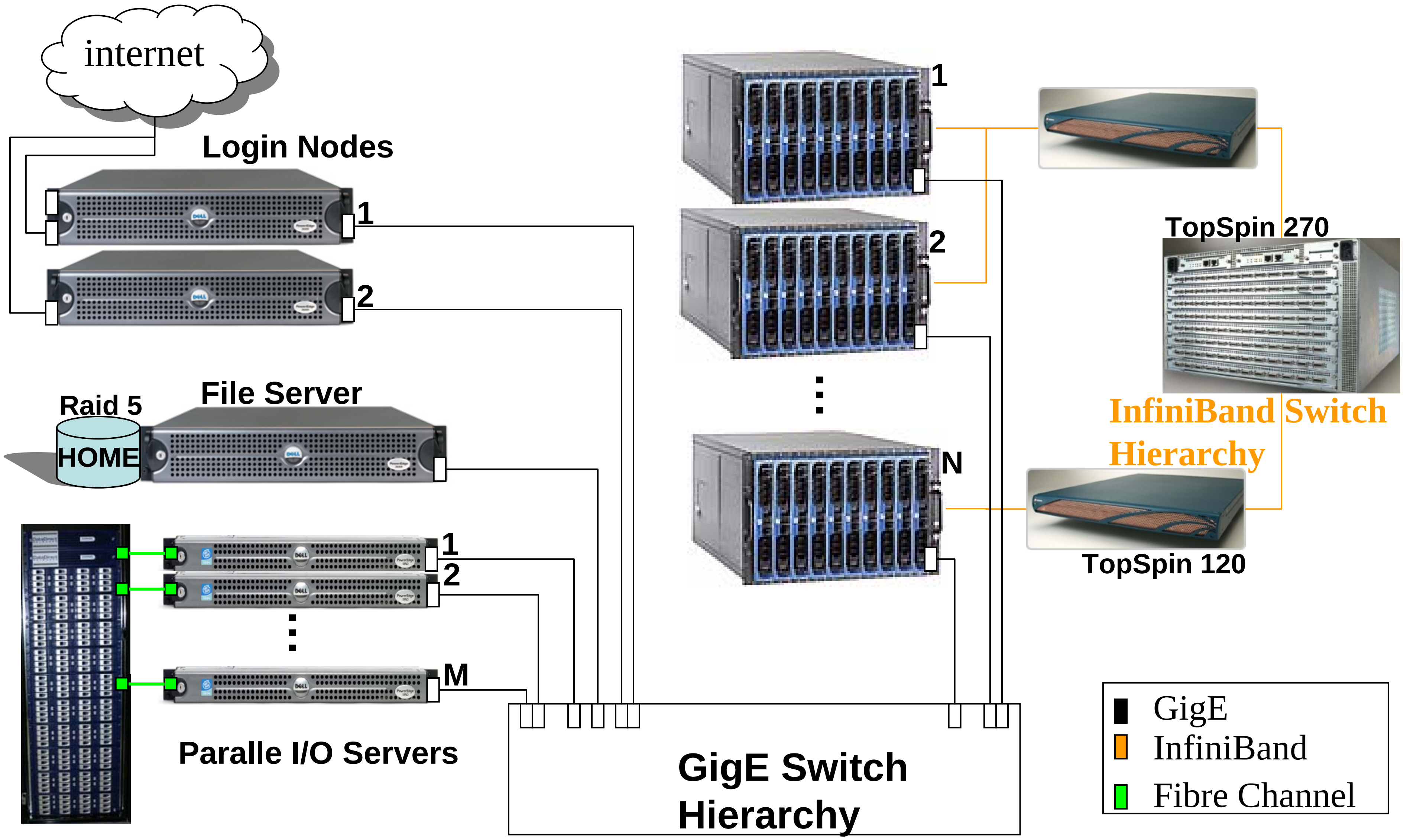
- SMP systems do not scale well
  - bus-based systems can become saturated
  - large, fast (high bandwidth, low latency) crossbars are expensive
  - cache-coherency is hard to maintain at scale (we'll get to what this means in a minute)
- Distributed systems scale well, but:
  - they are harder to program (message passing)
  - interconnects have higher latency
    - makes parallel algorithm development and programming harder

Basic Anatomy of a Server/Desktop/Laptop/Cluster-node



- **Processors**
- **Memory**
- Interconnect Network

Lonestar @ TACC

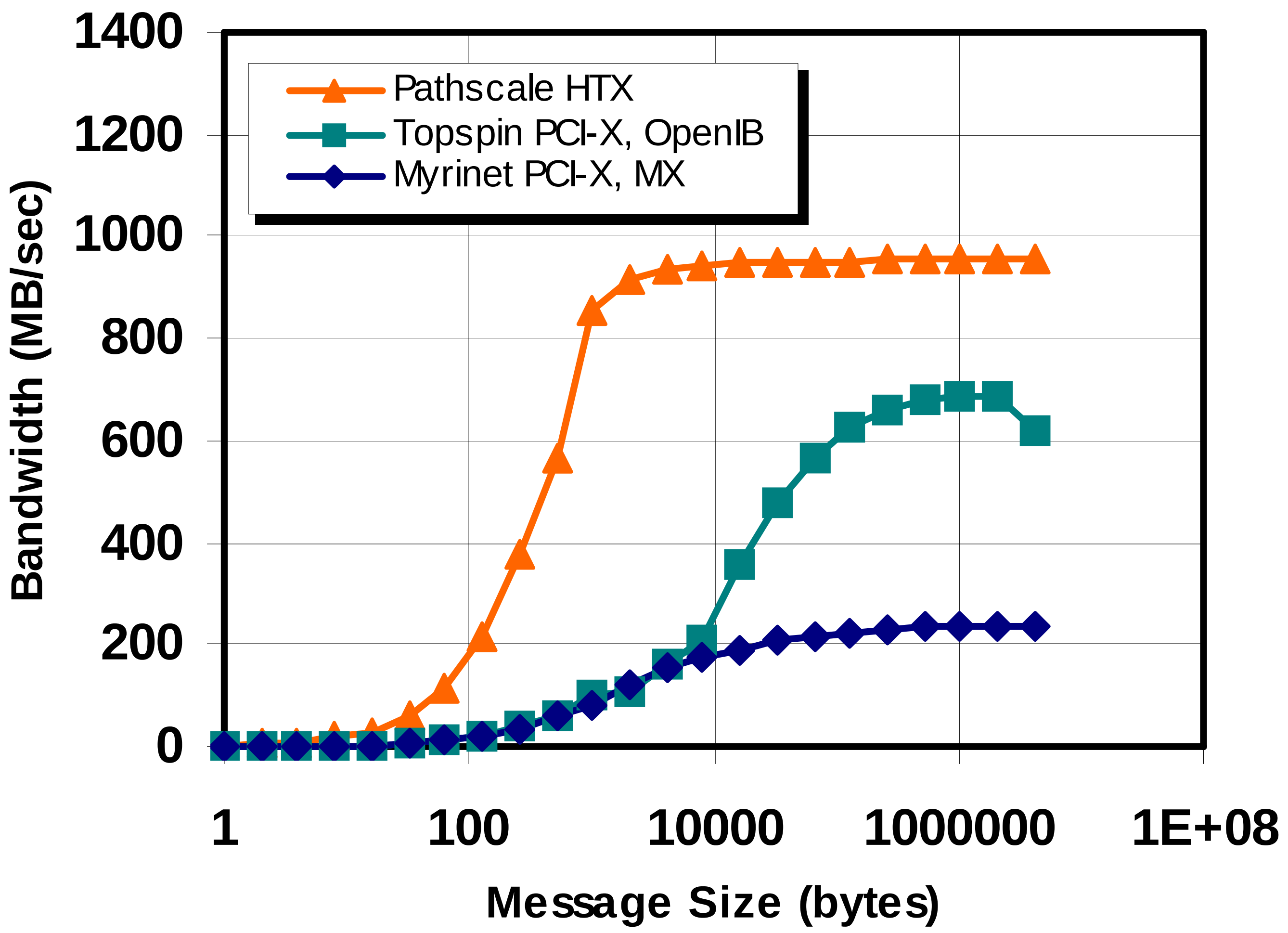


TACC

Interconnects

- Started with FastEthernet (*Beowulf* @ NASA)
  - 100 Mb/s, 100 μs latency
  - quickly transitioned to higher bandwidth, lower latency solutions
- Now
  - Ethernet/IP network for administrative work
  - InfiniBand, Myrinet, Quadrics for MPI traffic

Interconnect Performance



RAID

- Was: Redundant Array of Inexpensive Disks
- Now: Redundant Array of Independent Disks
- Multiple disk drives working together to:
  - increase capacity of a single logical volume
  - increase performance
  - improve reliability/add fault tolerance
- 1 Server with RAIDed disks can provide disk access to multiple nodes with NFS

Parallel Filesystems

- Use multiple servers together to aggregate disks
  - utilizes RAIDed disks
  - improved performance
  - even higher capacities
  - may use high-performance network
- Vendors/Products
  - CFS/Lustre
  - IBM/GPFS
  - IBRIX/IBRIXFusion
  - RedHat/GFS
  - ...

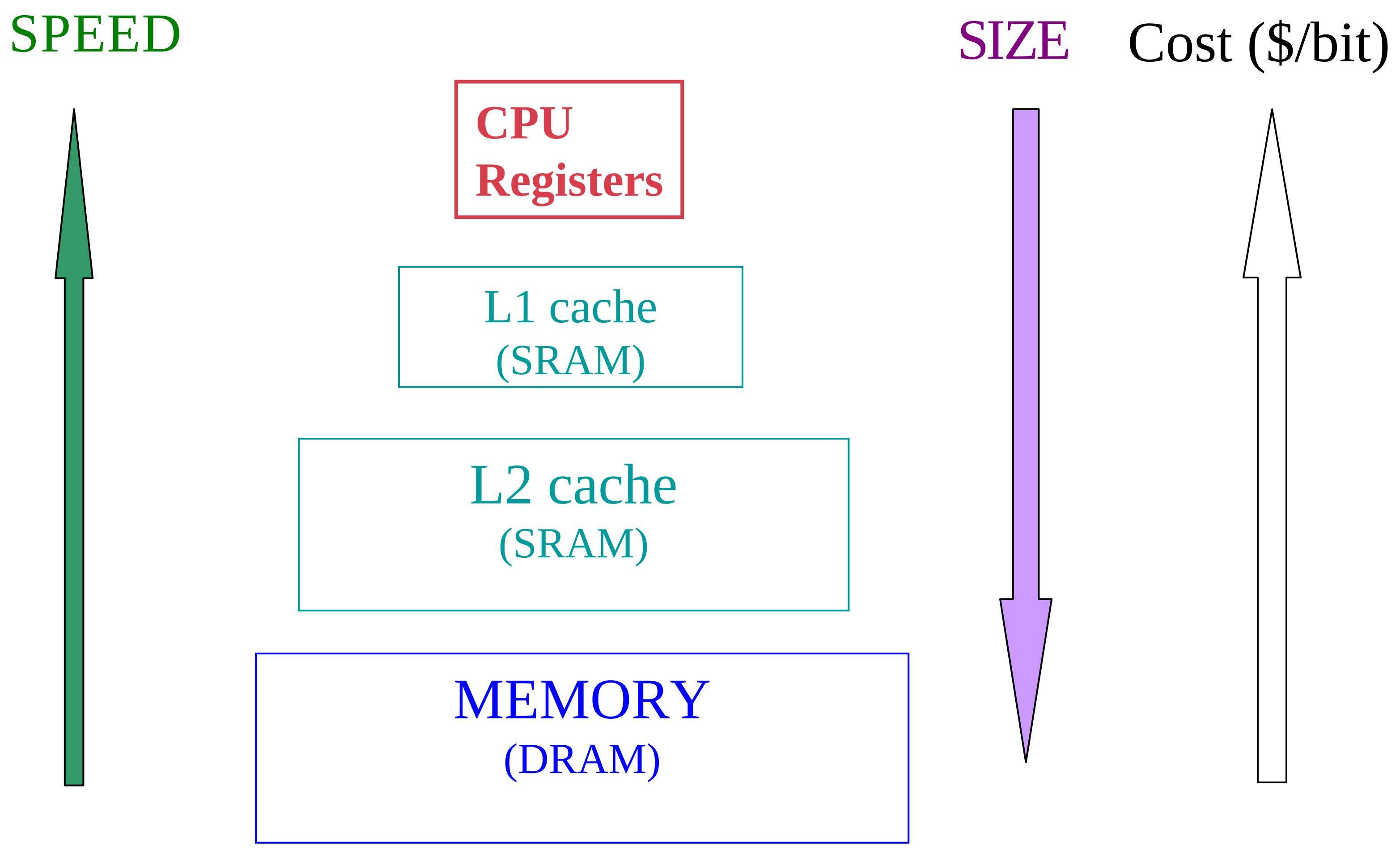
Microarchitecture

- Memory hierarchies
- Commodity CPUs
  - theoretical performance
  - piplining
  - superscaling
- Interconnects
  - Different topologies
  - Performance

Memory Hierarchies

- Due primarily to cost, memory is divided into different levels:
  - Registers
  - Caches
  - Main Memory
- Memory is accessed through the hierarchy
  - registers where possible
  - ... then the caches
  - ... then main memory

Memory Relativity

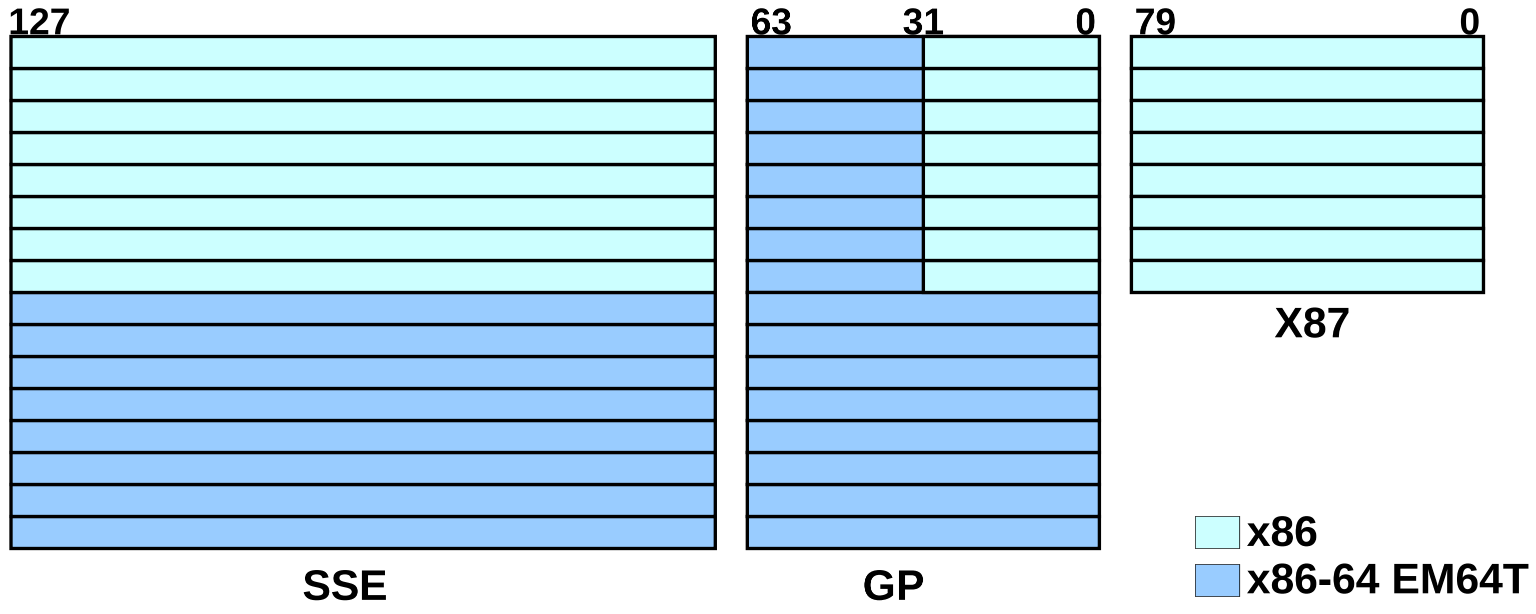


Latency and Bandwidth

- The two most important terms related to performance for memory subsystems and for networks are:
  - Latency
    - How long does it take to retrieve a word of memory?
    - Units are generally nanoseconds or clock periods (CP).
  - Bandwith
    - What data rate can be sustained once the message is started?
    - Units are B/sec (MB/sec, GB/sec, etc.)

Registers

- Highest bandwidth, lowest latency memory that a modern processor can acces
  - built into the CPU
  - often a scarce resource
  - not RAM
- AMD x86-64 and Intel EM64T Registers



Registers

- Processors instructions operate on registers directly
  - have names like: eax, ebx, ecx, etc.
  - sample instruction:

```
addl  %eax, %edx
```

- Separate instructions and registers for floating-point operations

Cache

- Between the CPU Registers and main memory
- L1 Cache : Data cache closest to registers (on die)
- L2 Cache: Secondary data cache, stores both data and instructions (on die)
  - Data from L2 has to go through L1 to registers
  - L2 is 10 to 100 times larger than L1
  - Some systems have a off-die L3 cache, ~10x larger than L2
- Cache line
  - The smallest unit of data transferred between main memory and the caches (or between levels of cache)
  - *N* sequentially-stored, multi-byte words (usually N=8 or 16).

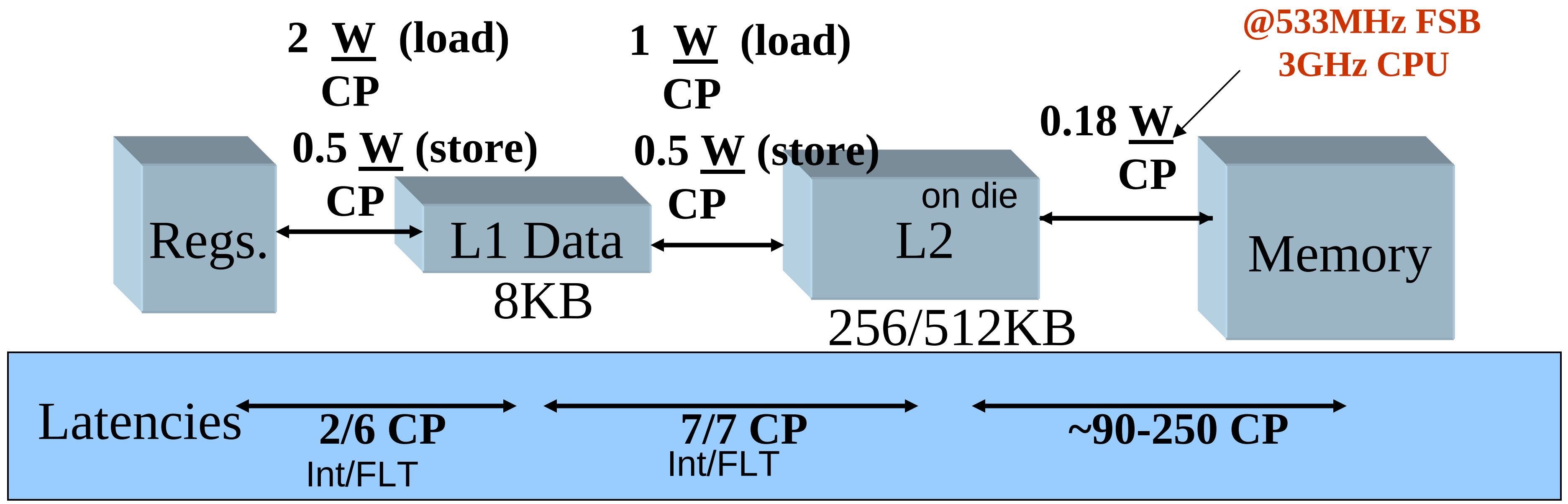
Main Memory

- Cheapest form of RAM
- Also the slowest
  - lowest bandwidth
  - highest latency
- Unfortunately most of our data lives out here

Approximate Latencies and Bandwidths in a Memory Hierarchy

	Latency	Bandwidth
Registers		
L1 Cache	~5 CP ↑	~2 W/CP
L2 Cache	~15 CP ↑	~1 W/CP
Memory	~300 CP ↑	~0.25 W/CP
Dist. Mem.	~10000 CP ↑	~0.01 W/CP

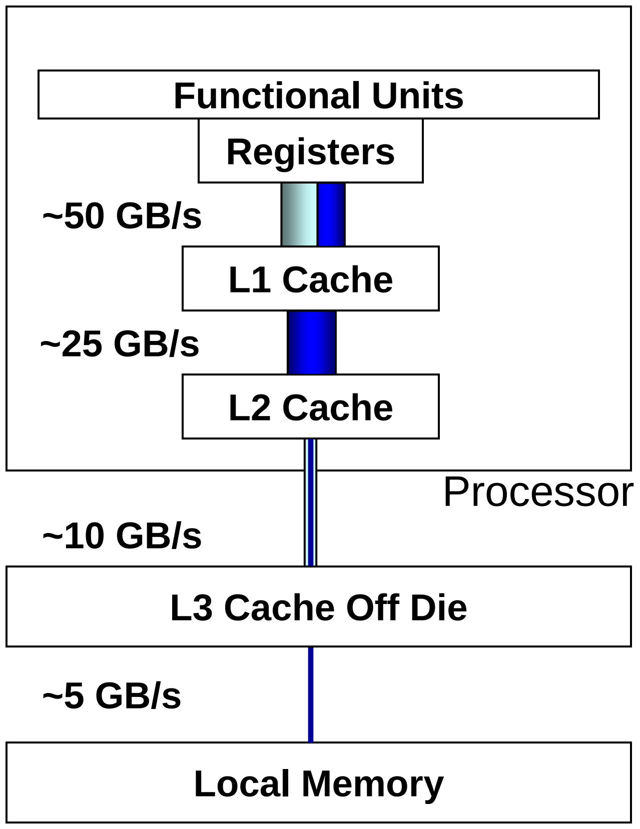
Example: Pentium 4



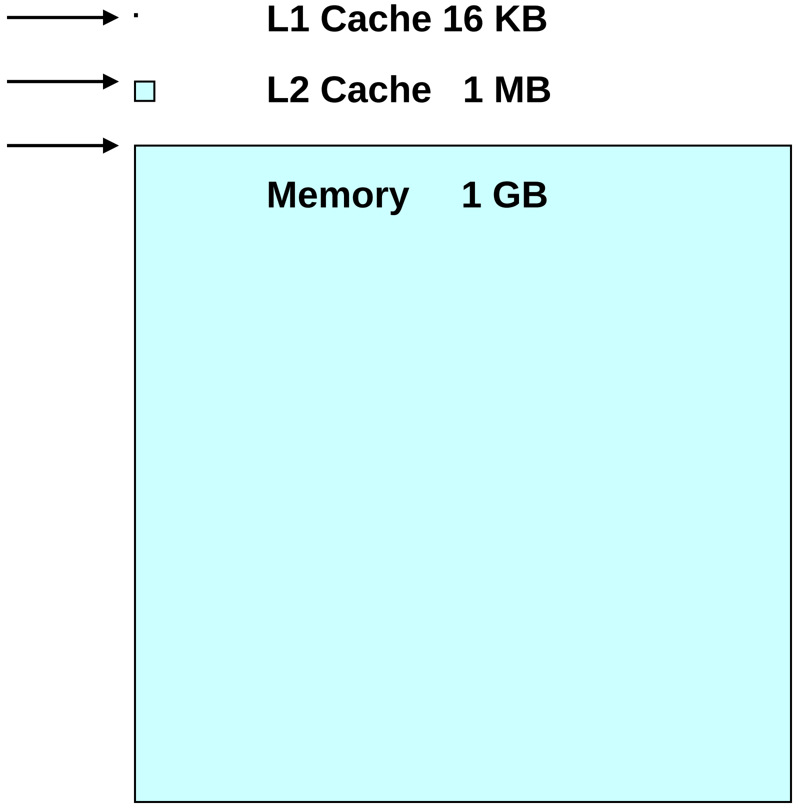
Line size L1/L2 =8W/16W

Memory Bandwidth and Size Diagram

Relative Memory Bandwidths

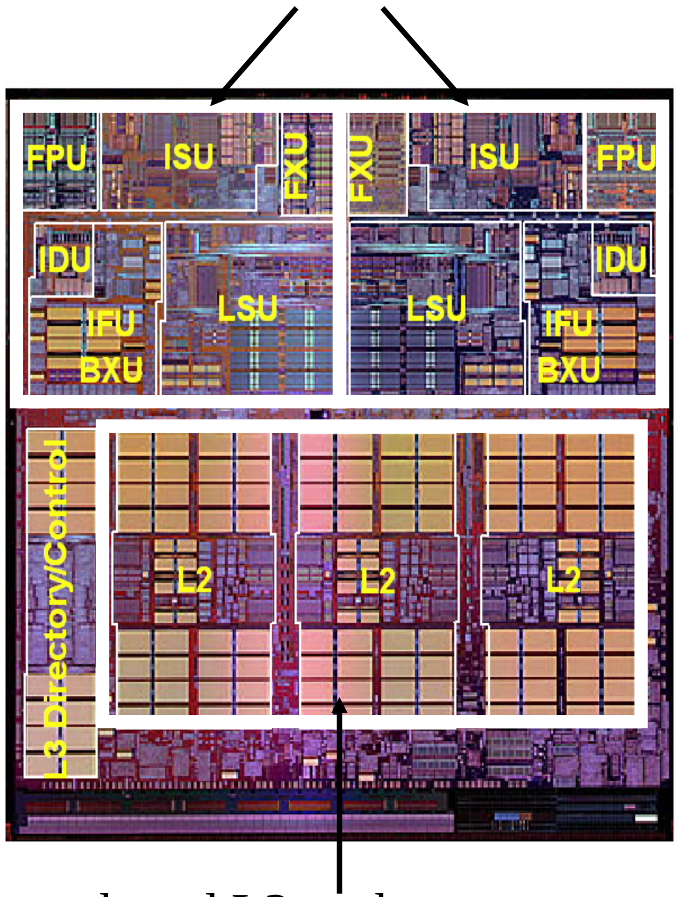


Relative Memory Sizes



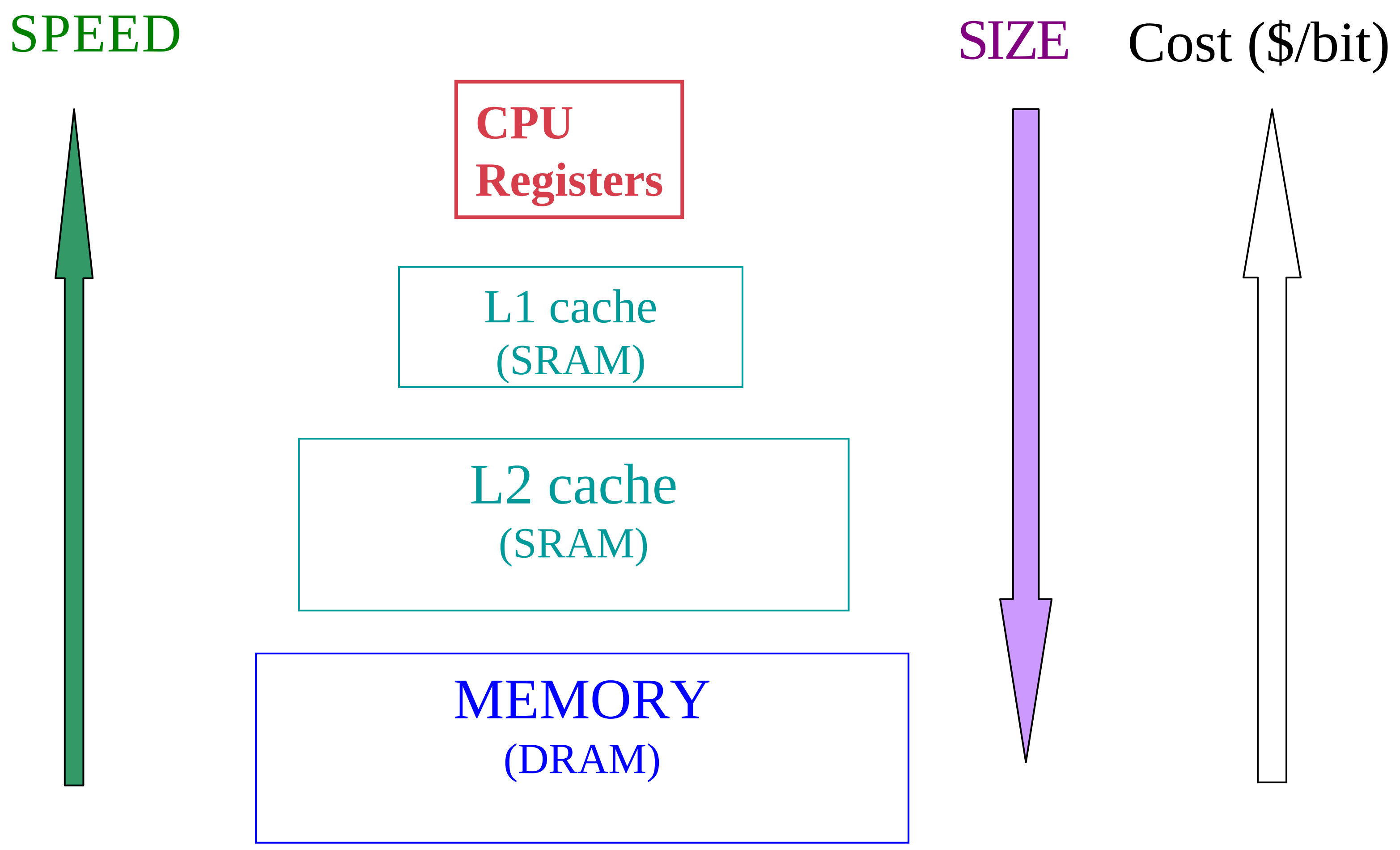
IBM Power4 Chip Layout

cores



shared L2 cache

Memory/Cache Related Terms



Why Caches?

- Since registers are expensive
- ... and main memory slow
- Caches provide a buffer between the two
- Access is transparent
  - either it’s in a register or
  - it’s in a memory location
  - processor/cache controller/MMU hides cache access from the programmer

Memory Access Example

```
#include <stdlib.h>
#include <stdio.h>
#define N 1234
int main()
{
  int i;
  int *buf=malloc(N*sizeof(int));
  buf[0]=1;
  for (i=1; i < N; ++i)
    buf[i]=i;
  printf("%d\n",buf[N-1]);
}
```

```
        movl    $1, (%eax)
        movl    $1, %edx
.L2:
        movl    %edx, (%eax,%edx,4)
        addl    $1, %edx
        cmpl    $1234, %edx
        jne     .L2
```

Hits, Misses, Thrashing

Cache hit
  – location referenced is found in the cache
- Cache miss
  – location referenced is not found in cache
  – triggers access to the next higher cache or memory
- Cache thrashing
  – a thrashed cache line (TCL) much be repeatedly recalled in the process of accessing its elements
  – caused when other cache lines, assigned to the same location, are simultaneous accessing data/instructions that replace the TCL with their content.

Design Considerations

- Data cache designed with two key concepts in mind
- Spatial Locality
  - when an element is referenced, its neighbors will be referenced, too
  - all items in the cache line are fetched together
  - work on consecutive data elements in the same cache line gives performance boost
- Temporal Locality
  - when an element is referenced, it will be referenced again soon
  - arrange code so that date in cache is reused as often as possible

Cache Line Size vs Access Mode

Access Line-size	Sequential data	Random data
Short Line	more fetches (overhead)	Best—low “latency”
Long Line	Best--Fewer fetches, but higher probability for cache trashing.	Longer “latency”, effectively smaller cache

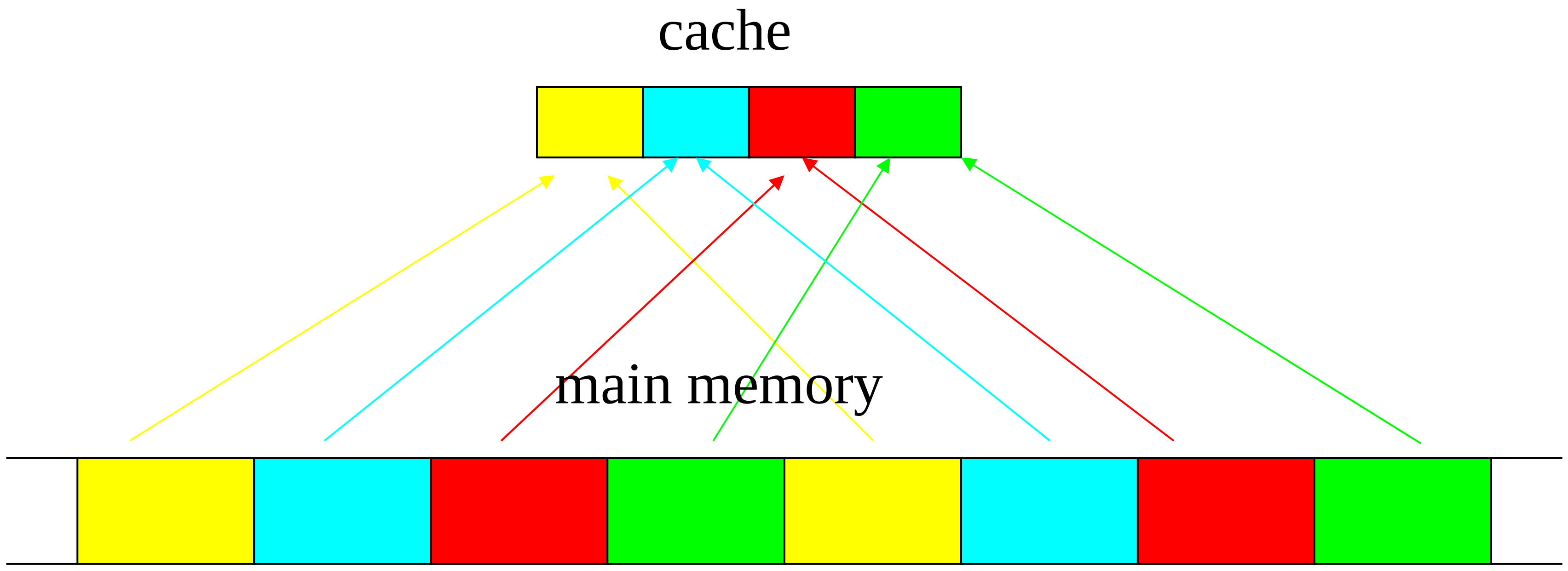
Cache Mapping

- Because each memory subsystem is smaller than the next-closer level, data must be mapped

- Types of mapping
  - Direct
  - Set associative
  - Fully associative

Direct Mapped Caches

Direct mapped cache: A block from main memory can go in exactly one place in the cache. This is called direct mapped because there is direct mapping from any block address in memory to a single location in the cache.

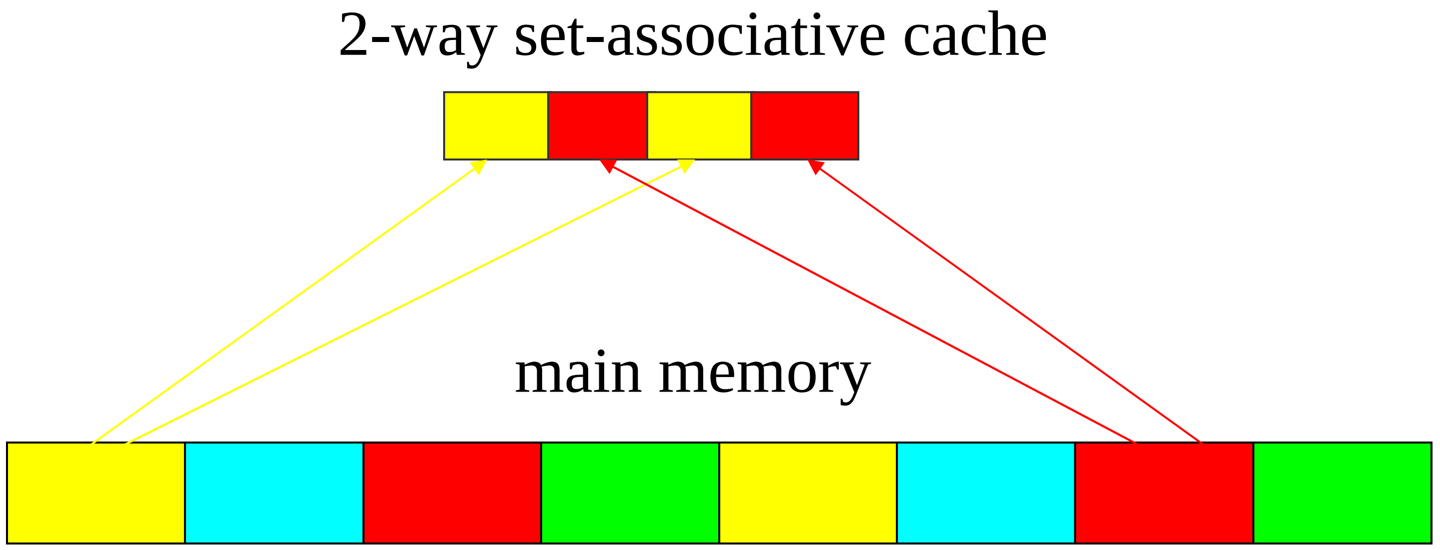


Direct Mapped Caches

- If the cache size is N_c and it is divided into k lines, then each cache line is N_c/k in size
- If the main memory size is N_m, memory is then divided into $N_m/(N_c/k)$ blocks that are mapped into each of the k cache lines
- Means that each cache line is associated with particular regions of memory

Set Associative Caches

<u>Set associative cache</u> **:** The middle range of designs between direct mapped cache and fully associative cache is called set-associative cache. In a n-way set-associative cache a block from main memory can go into n (n at least 2) locations in the cache.

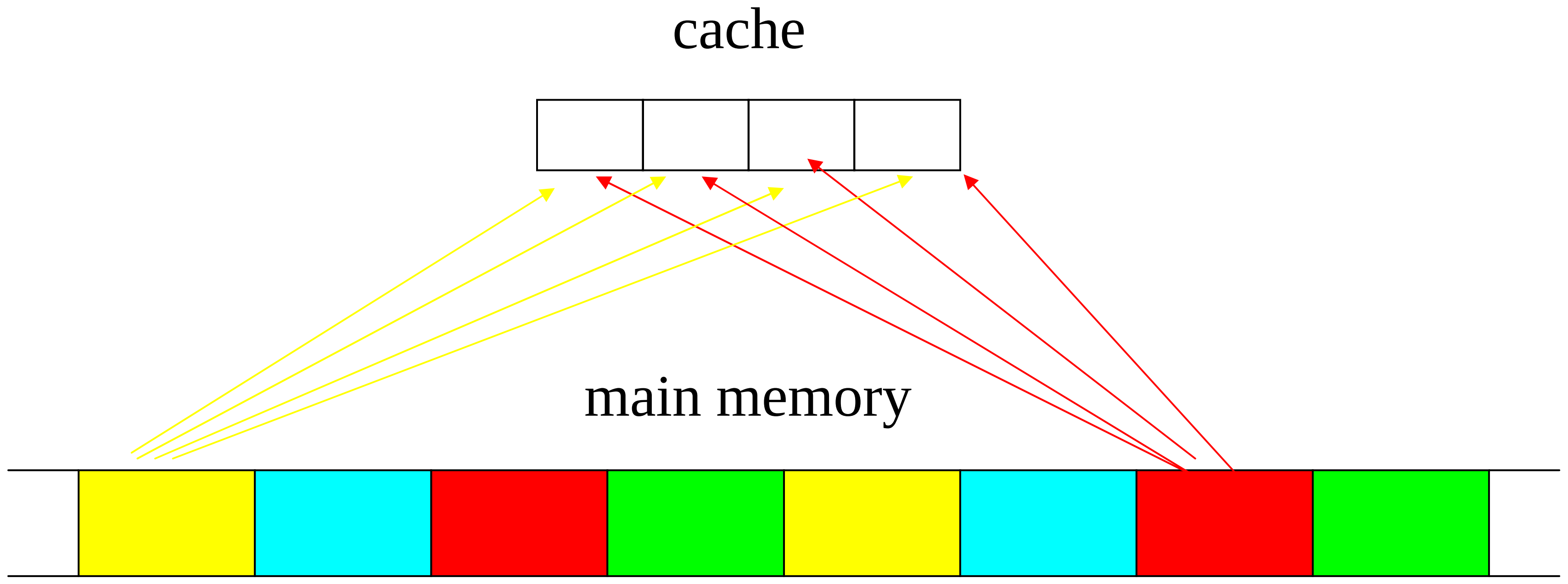


Set Associative Caches

- Direct-mapped caches are 1-way set-associative caches
- For a k-way set-associative cache, each memory region can be associated with *k* cache lines

Fully Associative Caches

<u>Fully associative cache :</u> A block from main memory can be placed in any location in the cache. This is called fully associative because a block in main memory may be associated with any entry in the cache.



Fully Associative Caches

- Ideal situation
- Any memory location can be associated with any cache line
- Cost prohibitive

Intel Woodcrest Caches

- L1
  - 32 KB
  - 8-way set associative
  - 64 byte line size
- L2
  - 4 MB
  - 8-way set associative
  - 64 byte line size

Theoretical Performance

- How many operations per clock cycle?
  - Intel Woodcrest: 4 Flop/cp
- Clock rate?
  - 2.66 GHz
- 4 Flop/cp * 2.66 Gcp/s = 10.64 GFlops
- 2 W/cp (loaded from L1 cache)
  - 2 * 2.66 * 8 = 42.56 GB/s
- ~ 0.25 W/cp (from main memory)
  - 0.25 * 2.66 * 8 = .665 GB/s
- 42.56 / 10.64 / 8 = 0.5 W/Flop for data in L1
- 0.665 / 10.64 / 8 = ~0.0625 W/Flop for data in main memory

Theoretical Performance

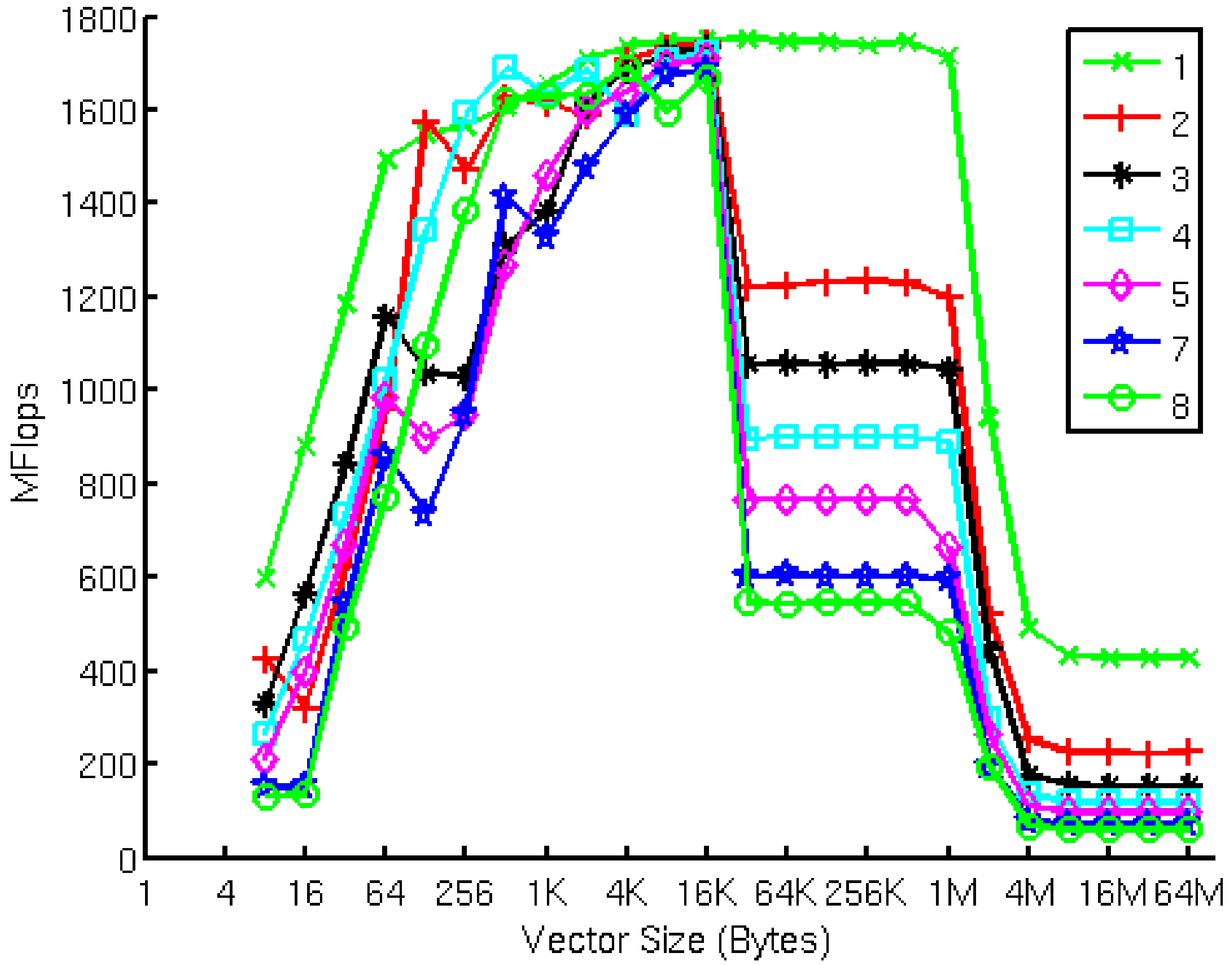
- Dot product:
  - *sum=sum+a[i]*b[i];*
  - 2 words loaded, 1 add, 1 multiply for each *i*
  - = 2 Words/2 Flops = 1 W/Flop
  - *sum* is in a register
- Should run OK (~50% of peak) if data fits in L1 cache (0.5 W/Flop)
- Will run poorly (< 10% of peak) out of main memory (~0.07 W/Flop)

Strided Dot Product

```
sum=0.;
for (j=0; j < stride; ++j)
  for(i=j; i < n; i+=stride)
    sum += a[i]*b[i];
```

- Strided access common on vector machines
  - *stride* = *vector length*
  - vector machine/compiler can remove the outer loop and compute a stride's worth of $a \cdot b$ at once
- Not likely to be useful on a scalar machine, but it's an easy way to cause cache misses

Dot Product Performance

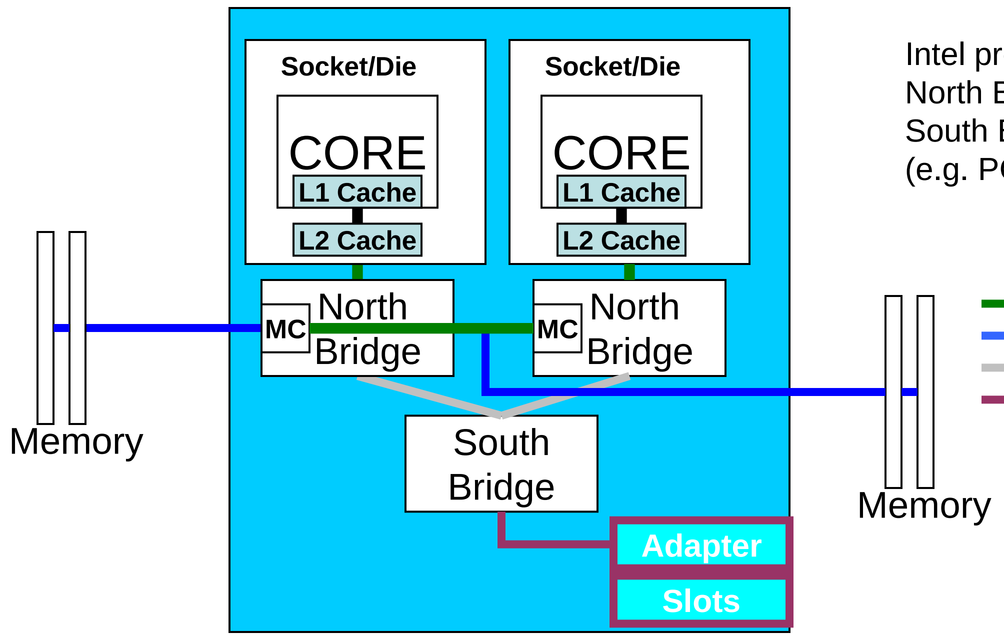


What’s Going On?

- Small vectors are noisy
  - probably not enough work to be measuring well
  - even still non-stride-1 access foil the plans of the hardware prefetcher
- Eventually everyone gets to the peak
  - ~1.8 GFlops = ~20% of peak
  - not far from what we predicted
  - probably some improvement yet to be had
- For *stride != 1* we see the L1 (32K) cache size boundary
- For *stride == 1* prefetching and other latency hiding tricks let the processor maintain performance
- Everybody hits the L2 (4MB) cache size boundary pretty hard

Intel System Architecture

- Basic Components of a Compute Node
  - Function Units: perform operations (e.g. Floating Point/Integer ops, load,stores, etc.)
  - Cores: (1-4) contains a set of functional units that function as an independent processor.
  - Caches: on-die
  - Bridges: Interconnects two different busses (may contain a “controller”)



Intel products:
North Bridge has memory controller.
South Bridge interfaces to adapters (e.g. PCI, PCI-X PCI-E)

🟩 FSB (Front Side Bus)
🟦 Memory Bus
⬜ Inter-Bridge Bus
🟪 PCI Bus

Memory Bandwidth

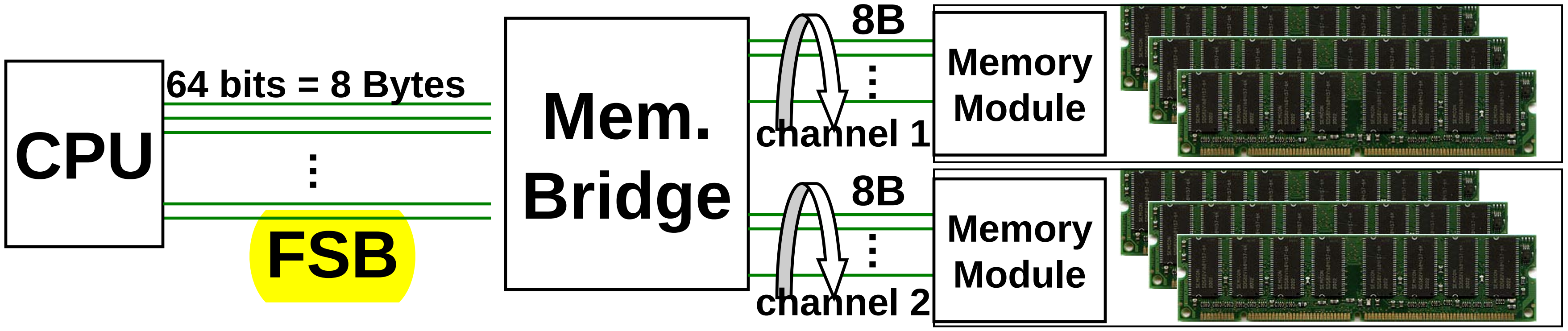
- Component Bandwidth (BW):

  The bus between the CPU and the Memory controller is known as the front-side bus (FSB). Multiply the frequency times the bus width to obtain bandwidth.

  The bus between the Memory Controller and the DIMMS determines the "Memory" speed. Multiply the frequency, bus width and number of channels to obtain an "aggregate" bandwidth.

  BW (memory) = 533 MHz (S) * 8B (W) * 2 = 8.5GB/s

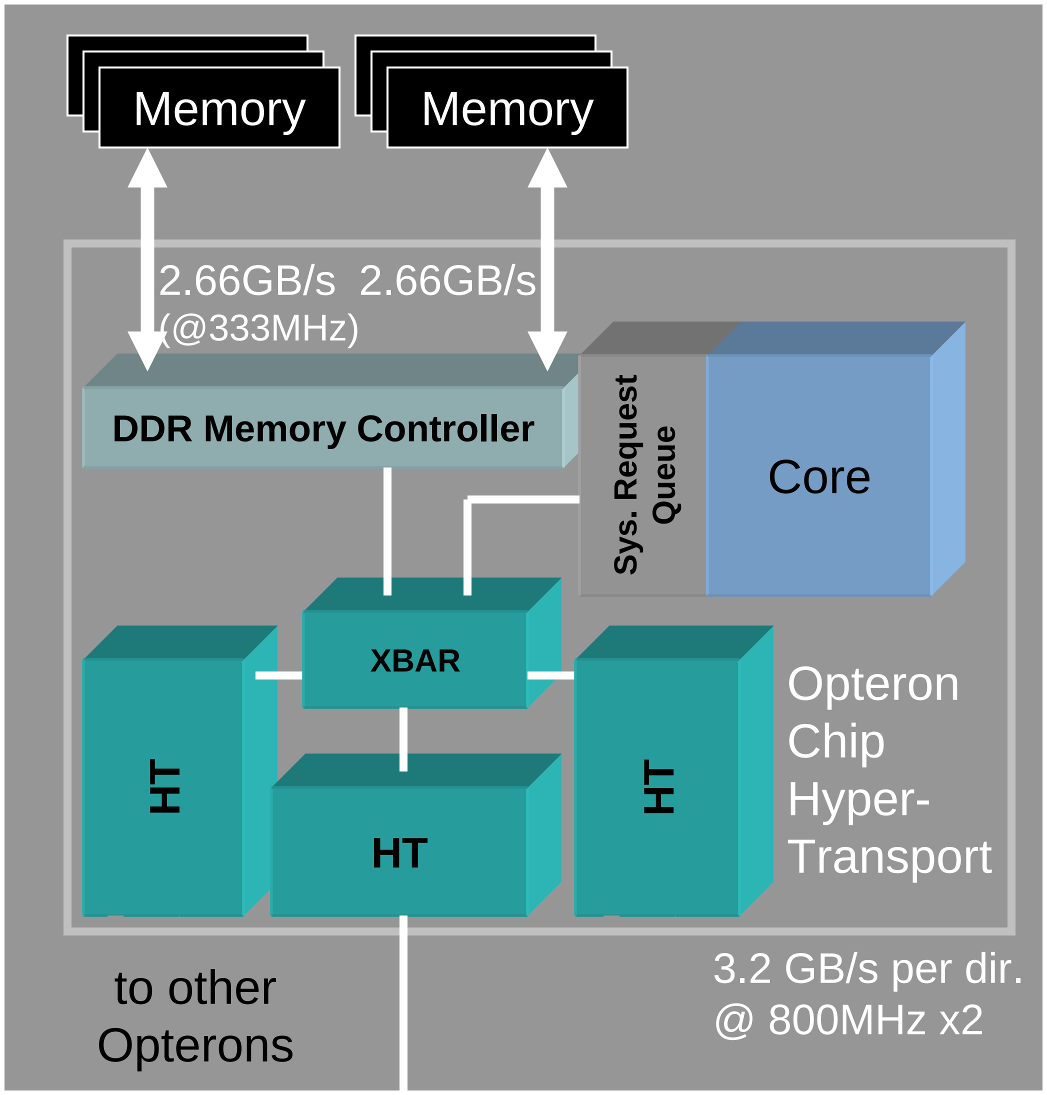
  BW (FSB) = 1.33 GHz (S) * 8B (W) = 10.7GB/s



AMD System Architecture

HyperTransport: New technology & protocol for data transfer—point-to-point links (chip-to-chip)

Crossbar switches between memory and HyperTransport (effectively, the Front Side Bus)



http://www.hypertransport.org/tech/index.cfm

Pipelining

Pipeline

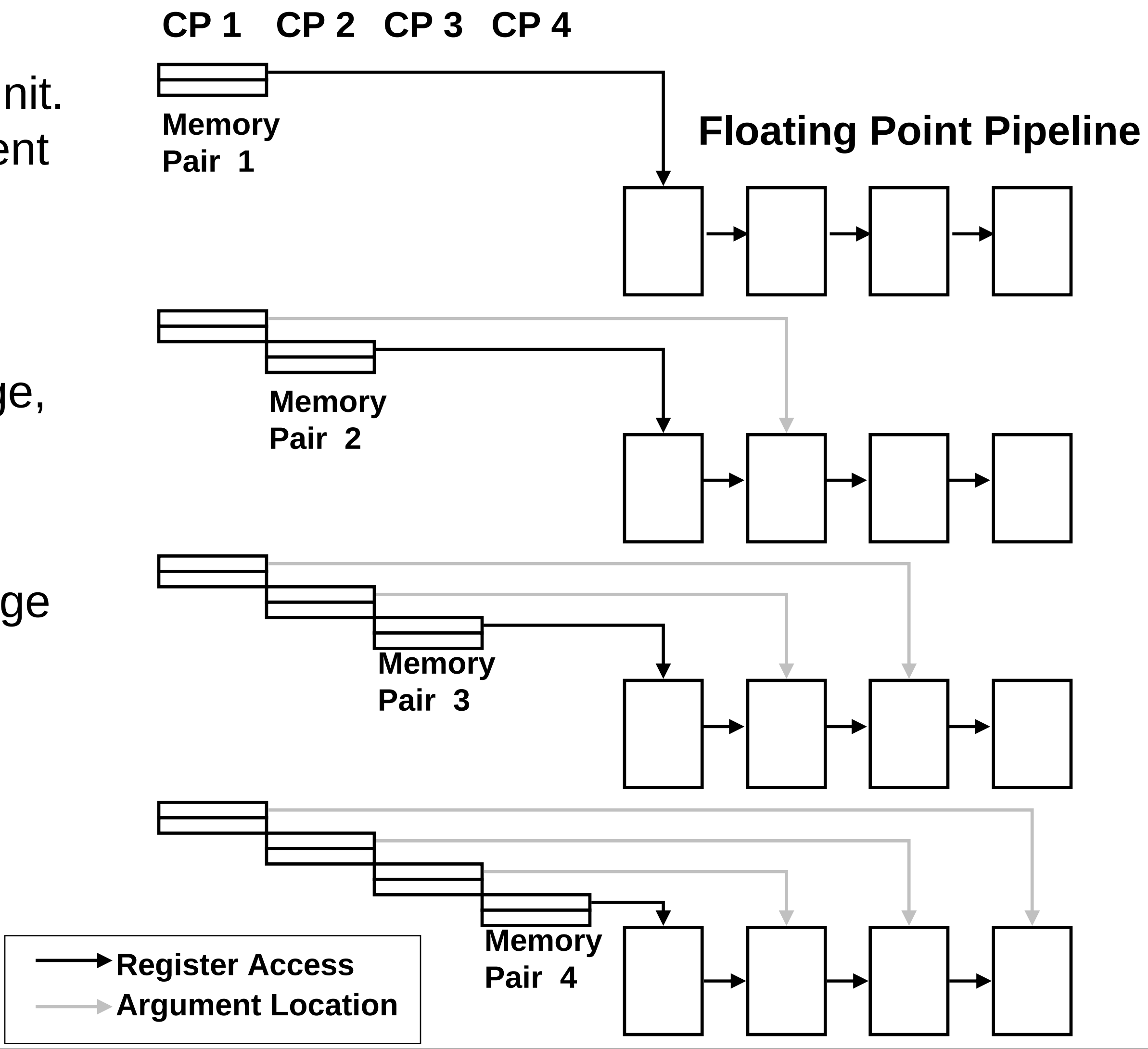
A serial multistage functional unit. Each stage can work on different sets of independent operands simultaneously.

After execution in the final stage, first result is available.

Latency = # of stages * CP/stage

CP/stage is the same for each stage and usually 1.

4-Stage FP Pipe



Branch Prediction

- The “instruction pipeline” is all of the processing steps (also called segments) that an instruction must pass through to be “executed”.
- Higher frequency machines have a larger number of segments.
- Branches are points in the instruction stream where the execution may jump to another location, instead of executing the next instruction.
- For repeated branch points (within loops), instead of waiting for the loop to branch route outcome, it is predicted.

Pentium III processor pipeline

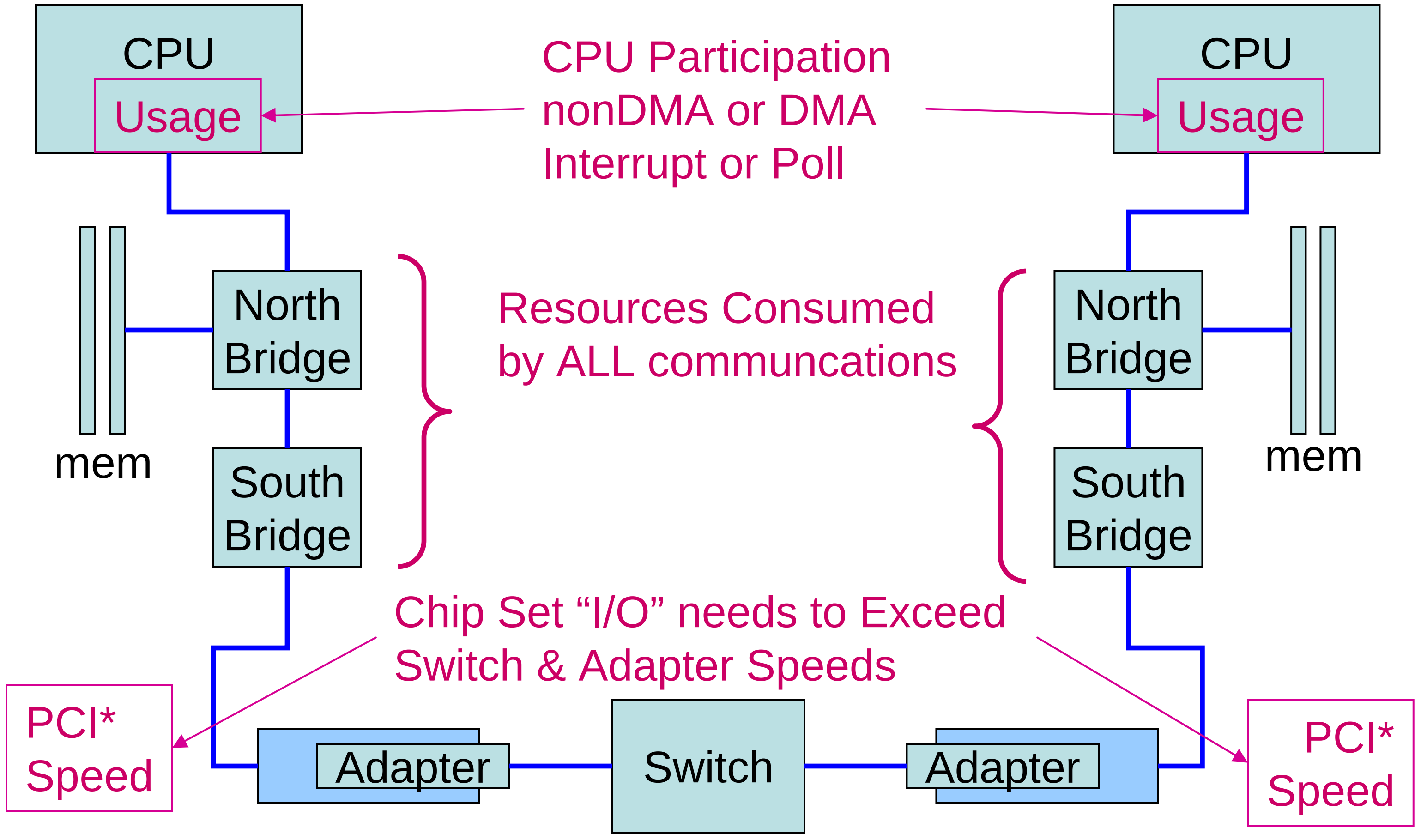
1	2	3	4	5	6	7	8	9	10

Pentium 4 processor pipeline

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Misprediction is more “expensive” on Pentium 4’s.

Hardware View of Communication (Intel)



Performance

- Software

  High
  Ease of Use $\updownarrow$
  Low

  Low
  Performance $\updownarrow$
  High

- Hardware

  High
  Adapter Control $\updownarrow$
  Low

  High
  Performance $\updownarrow$
  Low

1.) Latency $\longrightarrow$ Direct Memory Access -- DMA
2.) Bandwidth $\longrightarrow$ Switch, Adapter, Host Bus
3.) Host Overhead $\longrightarrow$ µprocessors

Message Cost = Latency + Bandwidth * Message Size

Polling vs interrupt
user API vs direct access

Node Communication in Clusters

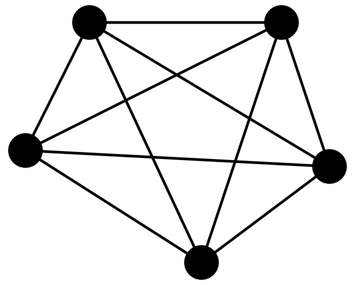
- Latency : How long does it take to start sending a "message"? Units are generally microseconds or milliseconds.
- Bandwidth : What data rate can be sustained once the message is started? Units are Mbytes/sec or Gbytes/sec.
- Topology: What is the actual ‘shape’ of the interconnect? Are the nodes connect by a 2D mesh? A ring? Something more elaborate?

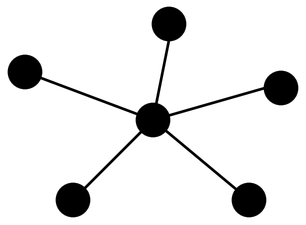
Node Communication in Clusters

- Processors can be connected by a variety of interconnects
- Static/Direct
  - point-to-point, processor-to-processor
  - no switch
  - major types/topologies
    - completely connected
    - star
    - linear array
    - ring
    - n-d mesh
    - n-d torus
    - n-d hyper cube
- Dynamic
  - processors connect to switches
  - major types
    - crossbar
    - fat tree

Completely Connected and Star Networks

- Completely Connected : Each processor has direct communication link to every other processor

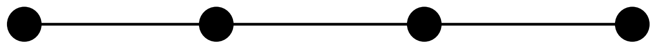




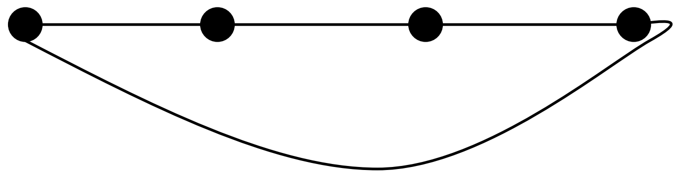
- Star Connected Network : The middle processor is the central processor. Every other processor is connected to it. Counter part of Cross Bar switch in Dynamic interconnect.

Arrays and Rings

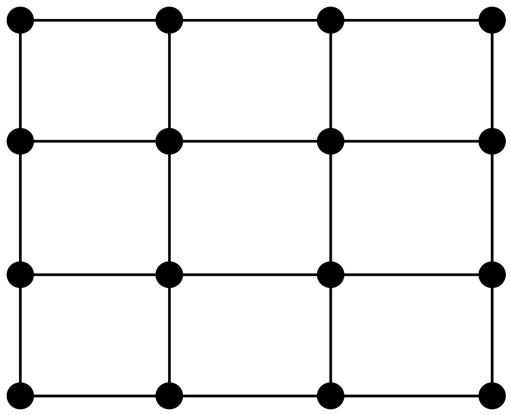
- Linear Array :



- Ring :

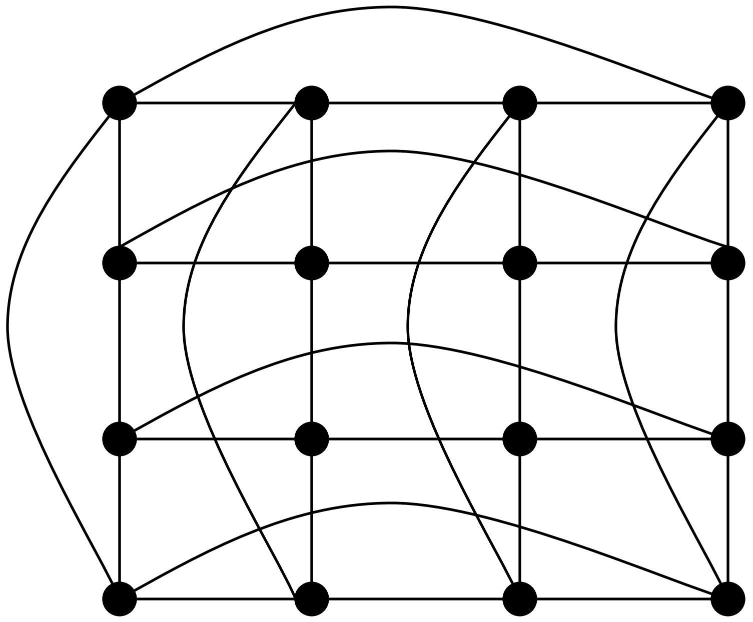


- Mesh Network (e.g. 2D-array)



Torus

2-d Torus (2-d version of the ring)

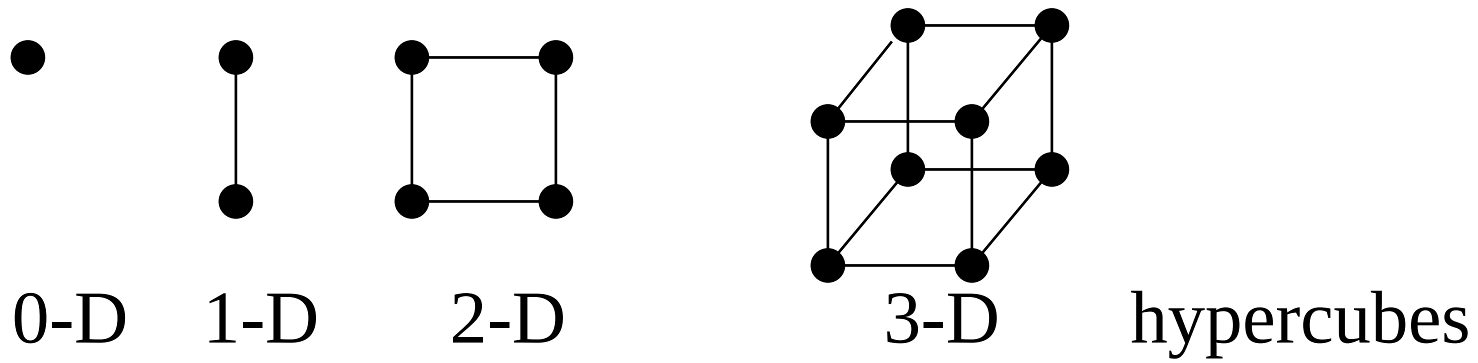


Hypercubes

- Hypercube Network : A multidimensional mesh of processors with exactly two processors in each dimension. A d dimensional processor consists of

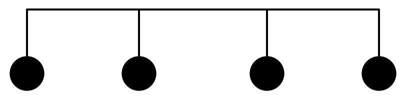
$$p = 2^d \text{ processors}$$

- Shown below are 0, 1, 2, and 3D hypercubes

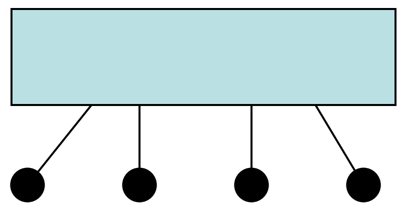


Busses/Hubs and Crossbars

Hub/Bus: Every processor shares the communication links

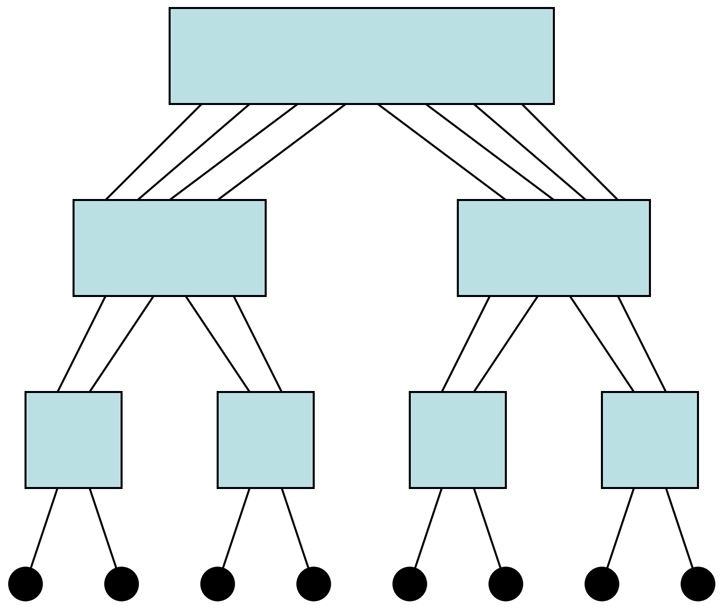


Crossbar Switches: Every processor connects to the switch which routes communications to their destinations



Fat Trees

- Multiple switches
- Each level has the same number of links in as out
- Increasing number of links at each level
- Gives full bandwidth between the links
- Added latency the higer you go



Interconnects

- Diameter
  - maximum distance between any two processors in the network.
  - The distance between two processors is defined as the shortest path, in terms of links, between them.
  - completely connected network is 1, for star network is 2, for ring is p/2 (for p even processors)
- Connectivity
  - measure of the multiplicity of paths between any two processors (# arcs that must be removed to break the connection).
  - high connectivity is desired since it lowers contention for communication resources.
  - 1 for linear array, 1 for star, 2 for ring, 2 for mesh, 4 for torus
  - technically 1 for traditional fat trees, but there is redundancy in the switch infrastructure

Interconnects

- Bisection width
  - Minimum # of communication links that have to be removed to partition the network into two equal halves. Bisection width is
  - 2 for ring, sq. root(p) for mesh with p (even) processors, p/2 for hypercube, (p*p)/4 for completely connected (p even).
- Channel width
  - of physical wires in each communication link
- Channel rate
  - peak rate at which a single physical wire link can deliver bits
- Channel BW
  - peak rate at which data can be communicated between the ends of a communication link
  - = (channel width) * (channel rate)
- Bisection BW
  - minimum volume of communication found between any 2 halves of the network with equal # of procs
  - = (bisection width) * (channel BW)