Lecture 4

Parallel Architectures

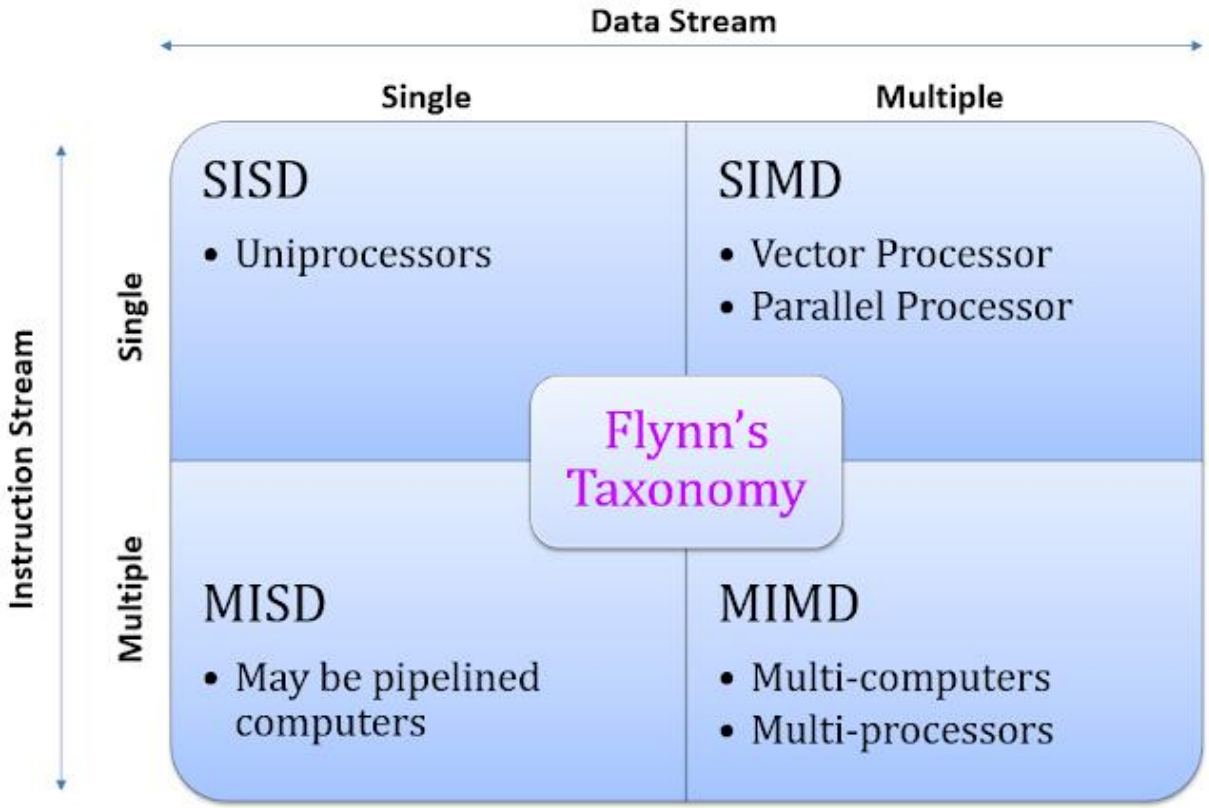
~different models of architectures that impose different models of computation~

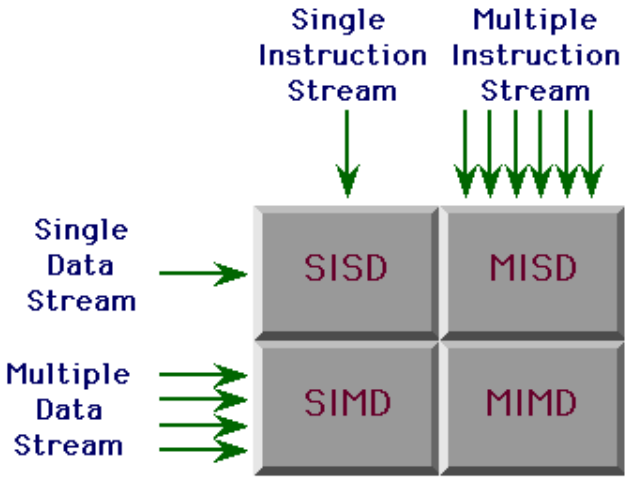
Reference: Introduction to Parallel Computing, Univ. of Oregon

Outline

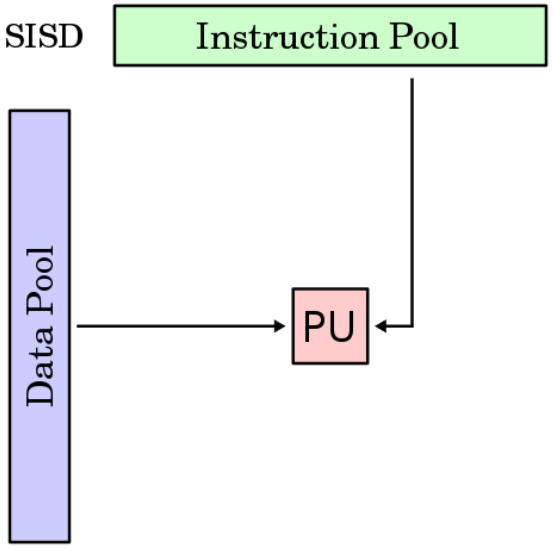
- Parallel architecture types – Flynn taxonomy
- Instruction-level parallelism
- Vector processing
- SIMD
- Shared memory
  - Memory organization: UMA, NUMA
  - Coherency: CC-UMA, CC-NUMA
- Distributed memory
  - Interconnection networks
- Clusters/Clusters of SMPs
- Heterogeneous clusters of SMPs
- Examples
- Cache coherence

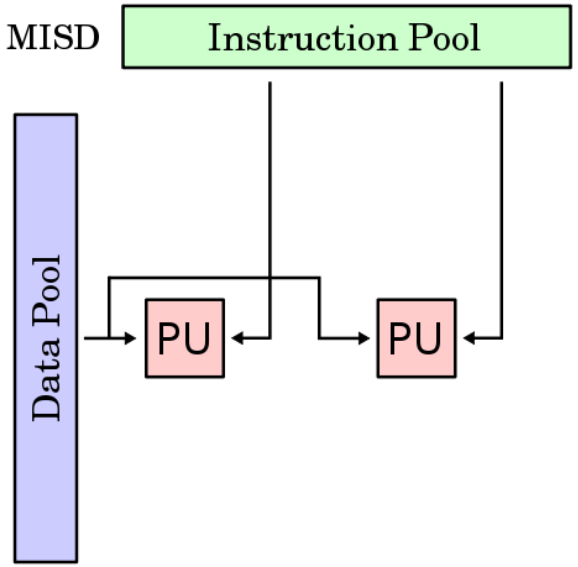
Flynn Taxonomy (M.J. Flynn -1966)

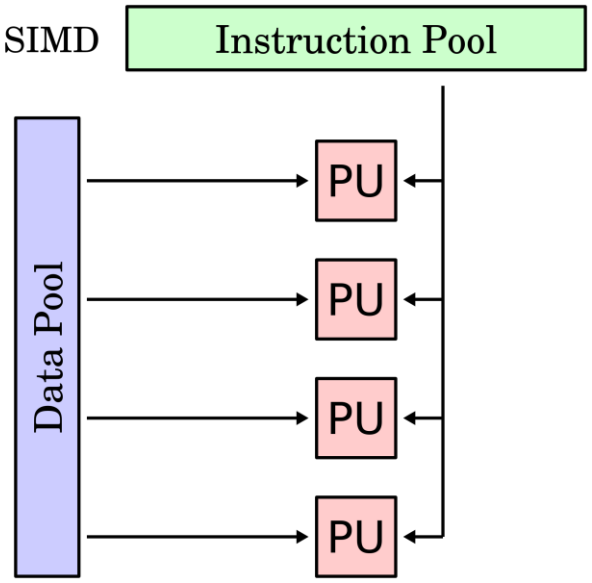


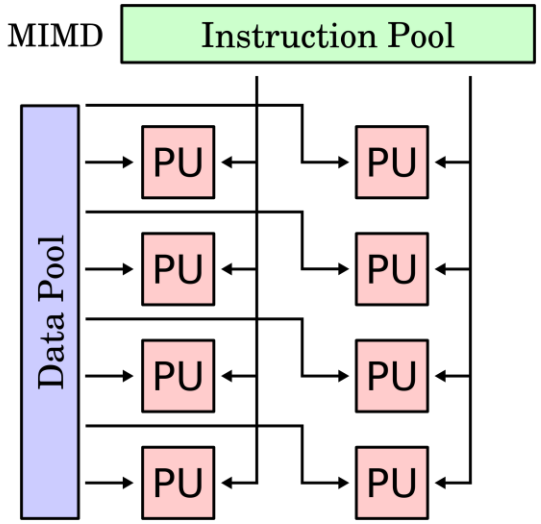


Flynn Taxonomy



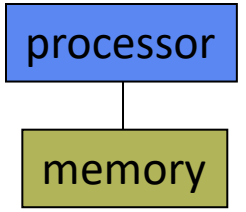




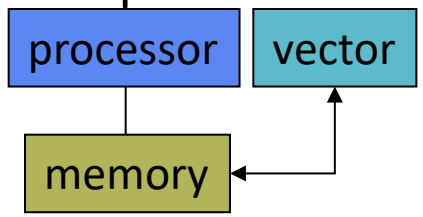


Parallel Architecture Types

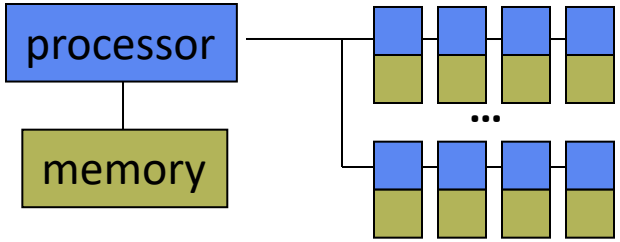
- Uniprocessor
  - Scalar processor



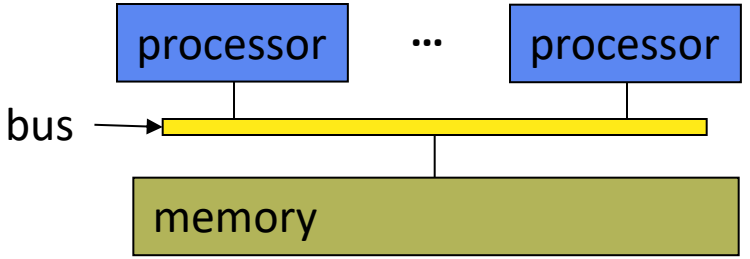
  - Vector processor



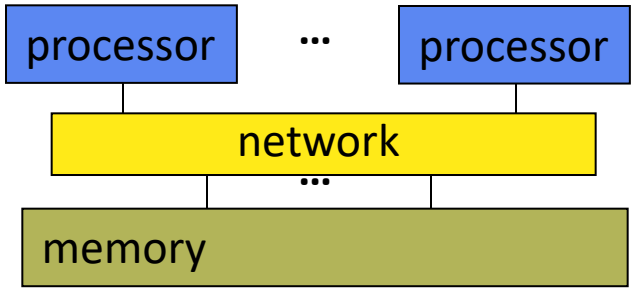
  - Single Instruction Multiple Data (SIMD)



- Shared Memory Multiprocessor (SMP)
  - Shared memory address space
  - Bus-based memory system

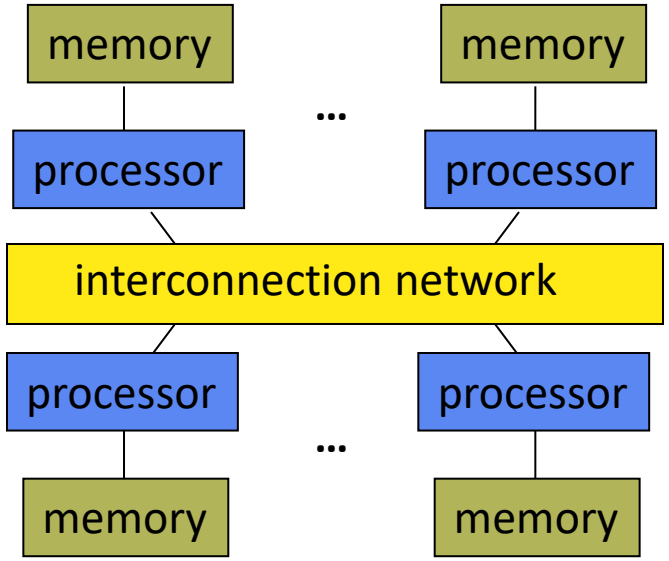


  - Interconnection network



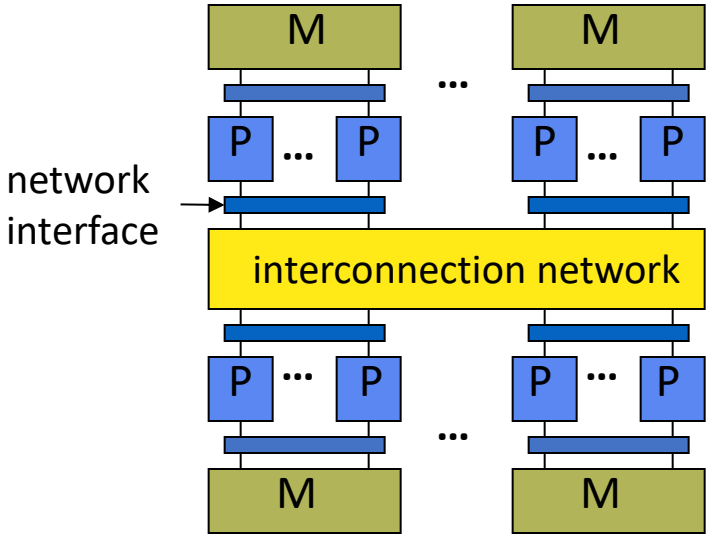
Parallel Architecture Types (2)

- Distributed Memory Multiprocessor
  - Message passing between nodes



  - Massively Parallel Processor (MPP)
    - Many, many processors

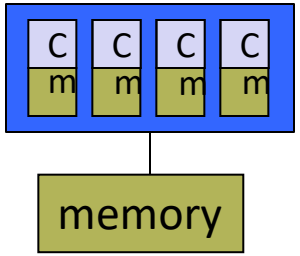
- Cluster of SMPs
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes



  - Can also be regarded as MPP if processor number is large

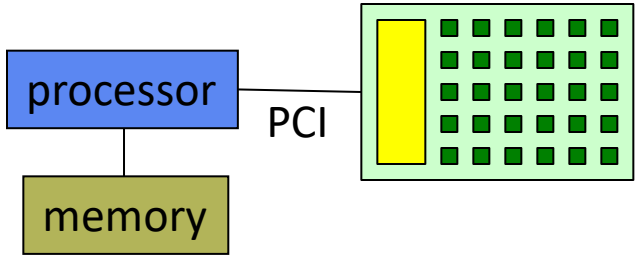
Parallel Architecture Types (3)

- Multicore
  - Multicore processor

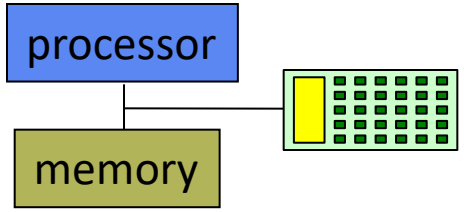


cores can be hardware multithreaded (hyperthread)

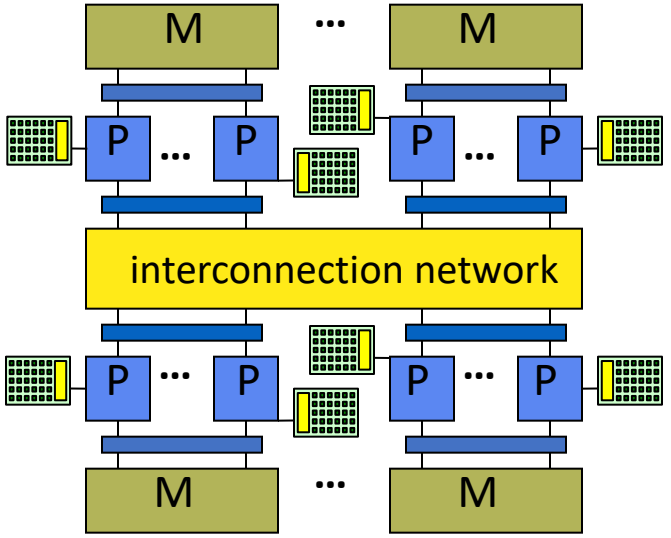
  - GPU accelerator



  - "Fused" processor accelerator



- Multicore SMP+GPU Cluster
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes
  - GPU accelerators attached



How do we get parallelism?

- Instruction-Level Parallelism (ILP)
- Data parallelism
  - Increase amount of data to be operated at the same time
- Processor parallelism
  - Increase number of processors
- Memory system parallelism
  - Increase number of memory units
  - Increase bandwidth to memory
- Communication parallelism
  - Increase amount of interconnection between elements
  - Increase communication bandwidth

Instruction-Level Parallelism (ILP)

- Opportunities for splitting up instruction processing
- Pipelining within instruction
- Pipelining between instructions
- Overlapped execution
- Multiple functional units
- Out of order execution
- Multi-issue execution
- Superscalar processing - can execute more than one instruction during a clock cycle
- Superpipelining - a technique of splitting instructions into many separate "pipelines"
- Very Long Instruction Word (VLIW) - instruction set architectures designed to allow instructions to execute in parallel.
- Hardware multithreading (hyperthreading)

Parallelism inside the CPU

(ref: https://www.tutorialspoint.com/cuda/cuda_tutorial.pdf)

- The five essential steps required for an instruction to finish:
  - Instruction fetch (IF)
  - Instruction decode (ID)
  - Instruction execute (Ex)
  - Memory access (Mem)
  - Register write-back (WB)
- This is a basic five-stage RISC architecture.
- There are multiple ways to achieve parallelism in the CPU.
  - one is ILP (Instruction Level Parallelism), also known as pipelining.

Instruction Level Parallelism

- how
  Instruction Level Parallelism works:

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

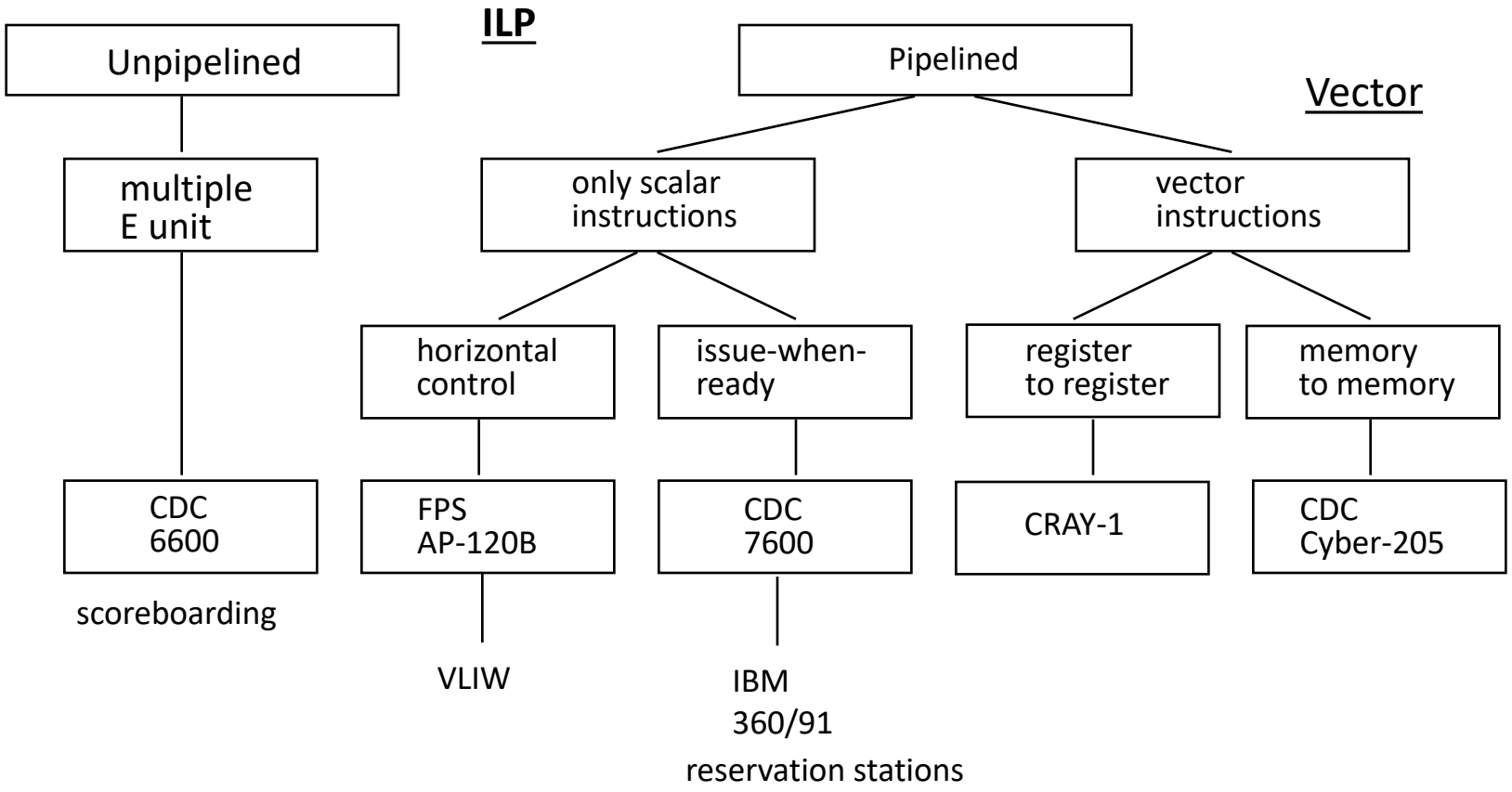
- **Using instruction pipelining, the instruction throughput increase (instructions per seconds).**
- **This way, we can process many instructions in one-clock cycle.**

ILP in a superscalar architecture

- The primary difference between a **superscalar** and a **pipelined** processor is (a superscalar processor is also pipeline) that the former uses **multiple execution units** (on the same chip) to achieve ILP whereas the latter divides the EU in multiple phases to do that.
  - This means that in superscalar, several instructions can simultaneously be in the same stage of the execution cycle. This is not possible in a simple pipelined chip.
  - Superscalar microprocessors can execute two or more instructions at the same time. They typically have at least 2 ALUs.
- **Superscalar processors can dispatch multiple instructions in the same clock cycle.**
  - This means that multiple instructions can be started in the same clock cycle.
  - In a pipelines architecture at any clock cycle, only one instruction is dispatched.
  - This is not the case with superscalars. But we have only one instruction counter (in-flight, multiple instructions are tracked). This is still just one process !

Parallelism in Single Processor Computers

- History of processor architecture innovation



Vector Processing

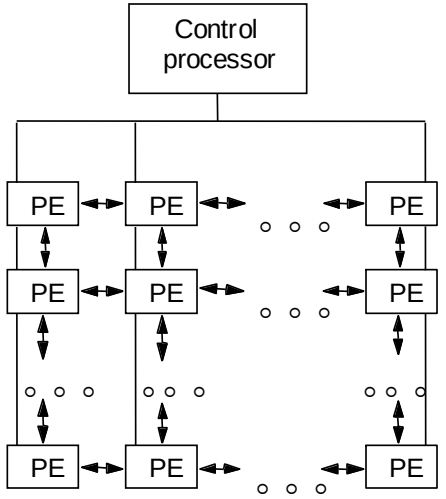
- Scalar processing
  - Processor instructions operate on scalar values
  - integer registers and floating point registers
- Vectors
  - Set of scalar data
  - ***Vector registers***
    - integer, floating point (typically)
  - *Vector instructions operate on vector registers* (SIMD)
- Vector unit pipelining
- Multiple vector units
- Vector chaining

Liquid-cooled with inert fluorocarbon. (That's a waterfall fountain!!!)

Cray 2

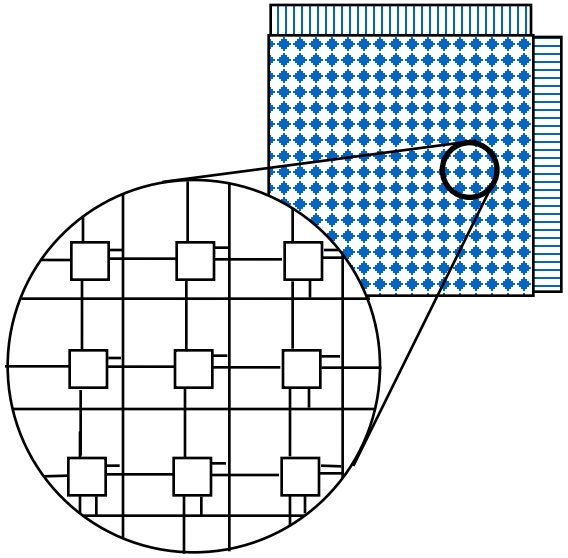
Data Parallel Architectures

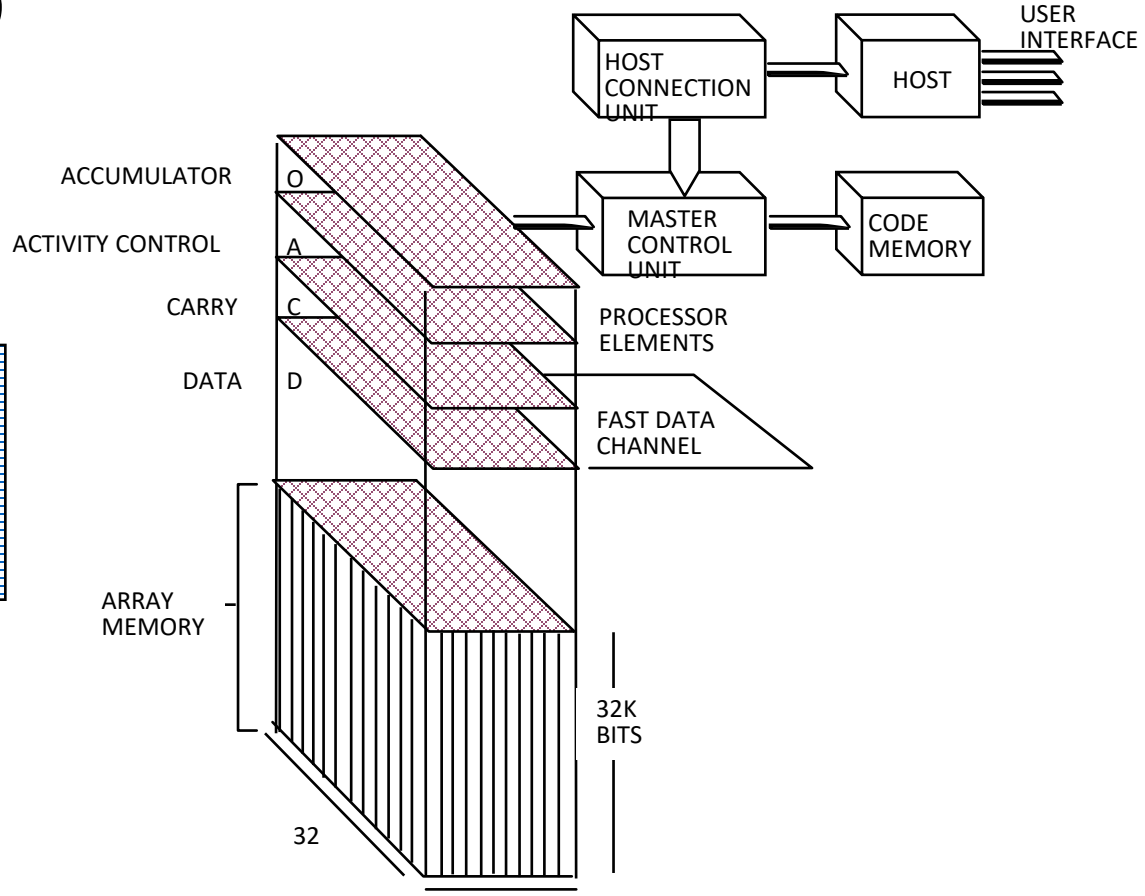
- SIMD (Single Instruction Multiple Data)
  - Logical single thread (instruction) of control
  - Processor associated with data elements
- Architecture
  - Array of simple processors with memory
  - Processors arranged in a regular topology
  - Control processor issues instructions
    - **All processors execute same instruction** (maybe disabled)
  - Specialized synchronization and communication
  - Specialized reduction operations
  - Array processing



AMT DAP 500

- Applied Memory Technology (AMT)
- Distributed Array Processor (DAP)

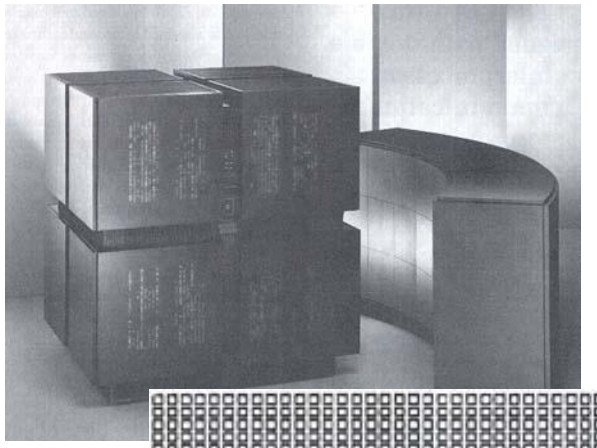


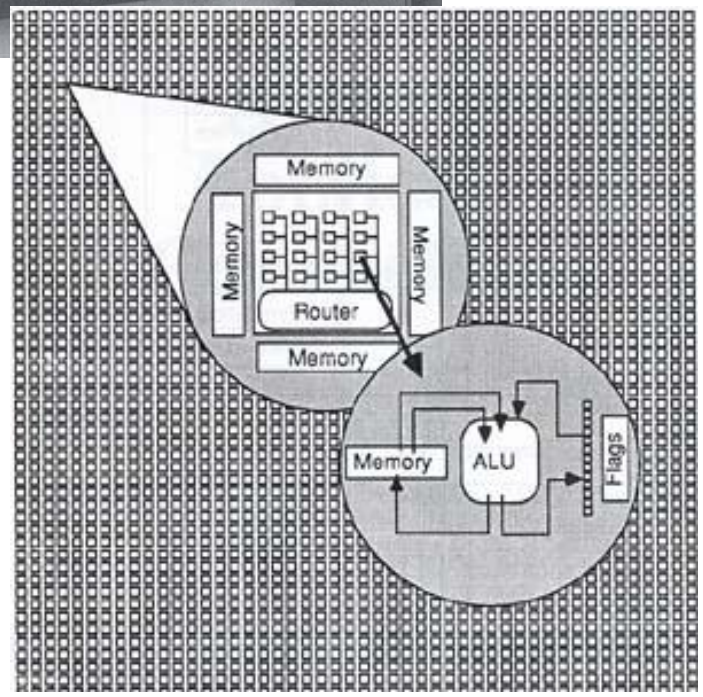


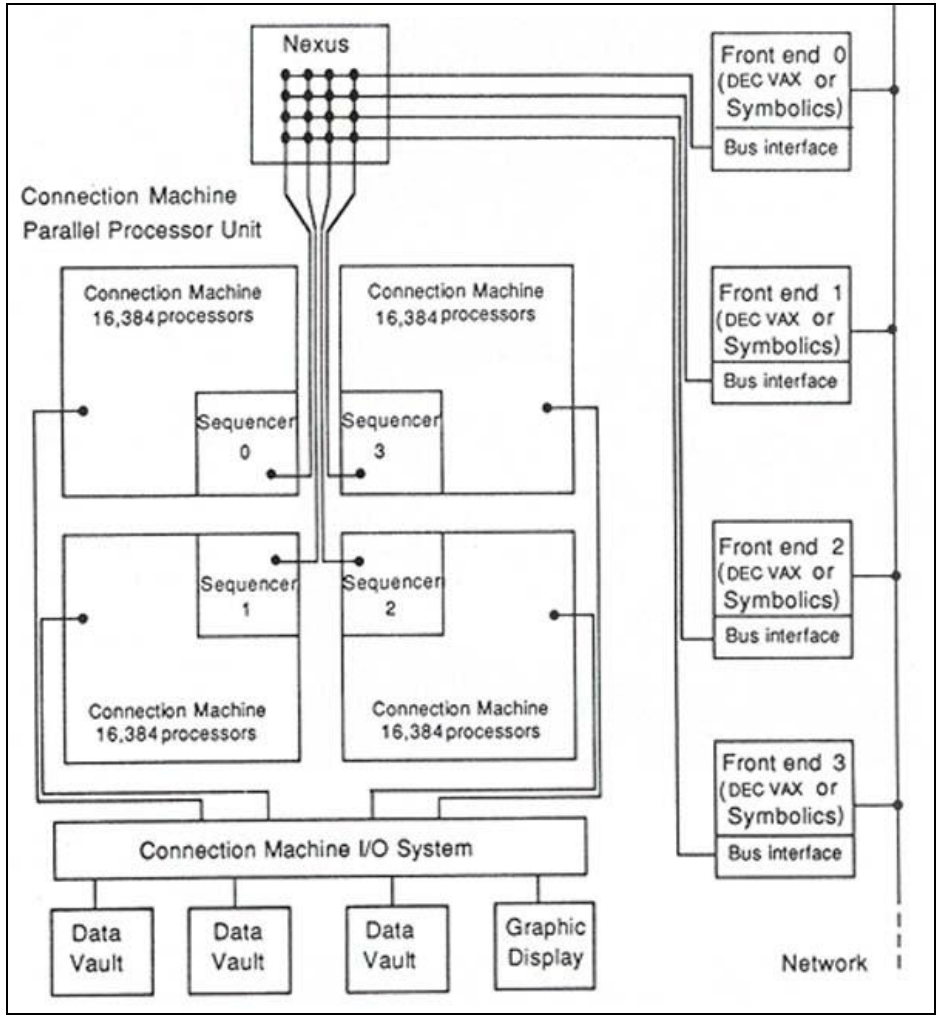
Thinking Machines Connection Machine

16,000 processors!!!

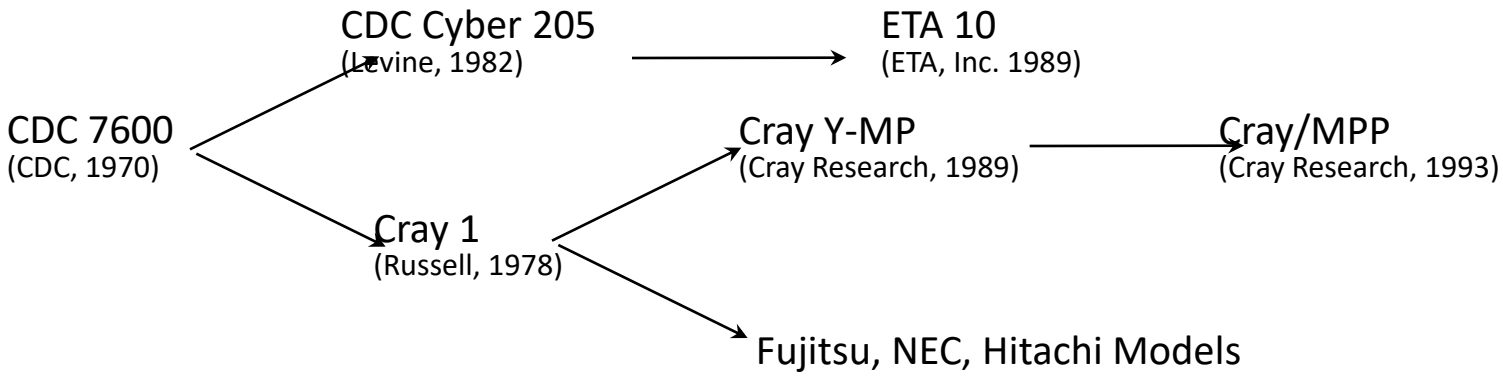
(Tucker, IEEE Computer, Aug. 1988)



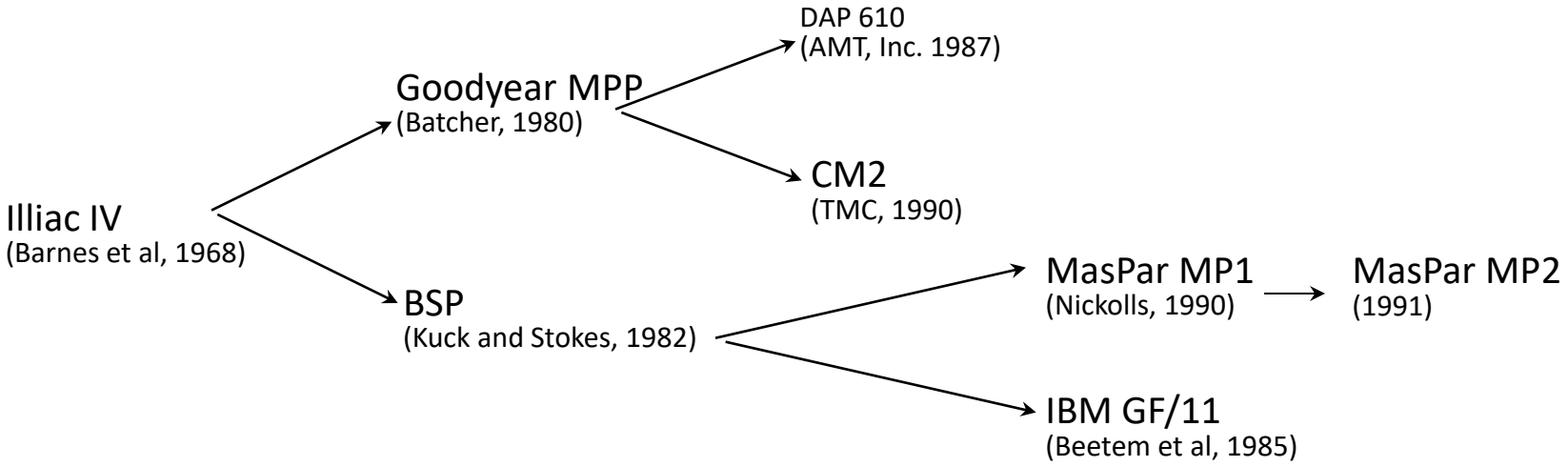




Vector and SIMD Processing Timeline



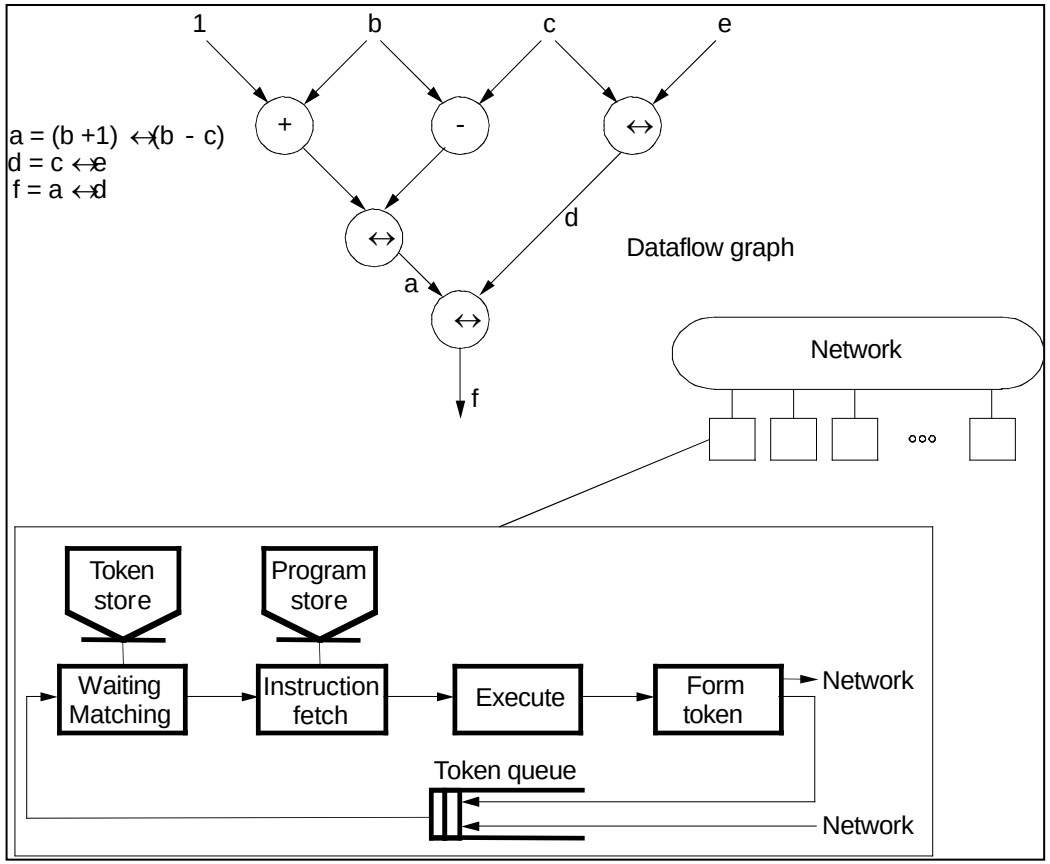
(a) Multivector track



(b) SIMD track

Dataflow Architectures

- ***Represent computation as graph of dependencies***
- Operations stored in memory until operands are ready
- Operations can be dispatched to processors
- Tokens carry tags of next instruction to processor
- Tag compared in matching store
- ***A match fires execution***
- Machine does the hard parallelization work
- Hard to build correctly !!!!



Shared Physical Memory

- Add processors to single processor computer system
- Processors *share* computer system resources
  - Memory, storage, …
- Sharing physical memory
  - Any processor can reference any memory location
  - Any I/O controller can reference any memory address
  - ***Single physical memory address space***
- Operating system runs on any processor, or all
  - OS see single memory address space
  - Uses shared memory to coordinate
- ***Communication*** occurs as a result of ***loads and stores***

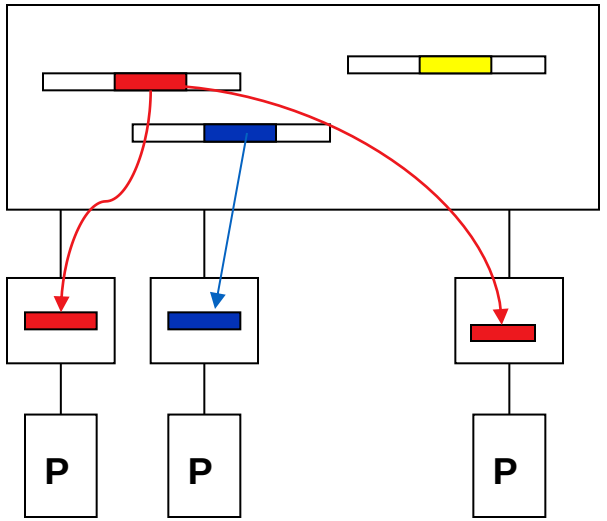
Latency vs. Bandwidth

- *Latency* = time needed for one date(byte or word) to arrived to processor
  (after the request was sent)

- *Bandwidth* = the rate at which data can be read from or stored into memory by a processor.
  - expressed in units of *bytes/second*

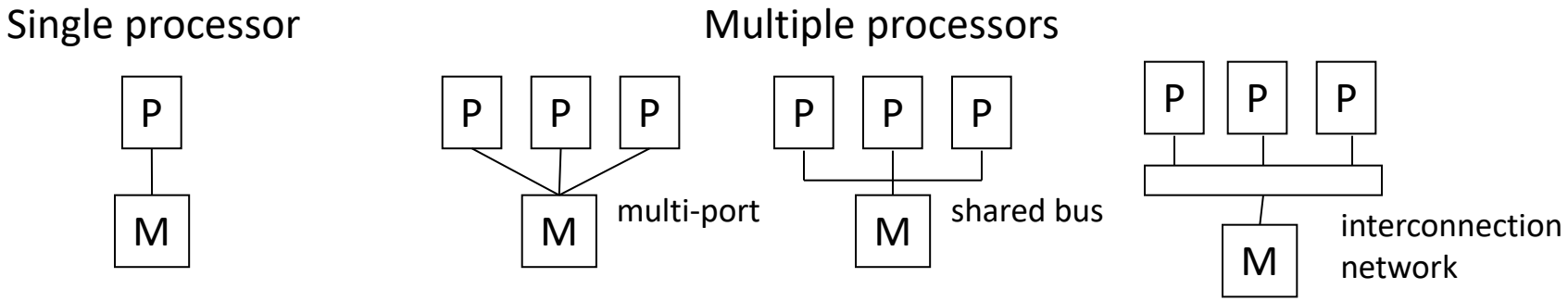
Caching in Shared Memory Systems

- Reduce average latency
  - automatic replication closer to processor
- Reduce average bandwidth
- Data is logically transferred from producer to consumer to memory
  - store reg $\rightarrow$ mem
  - load reg $\leftarrow$ mem
- Processors can share data efficiently
- What happens when store and load are executed on different processors?
- Cache coherence problems

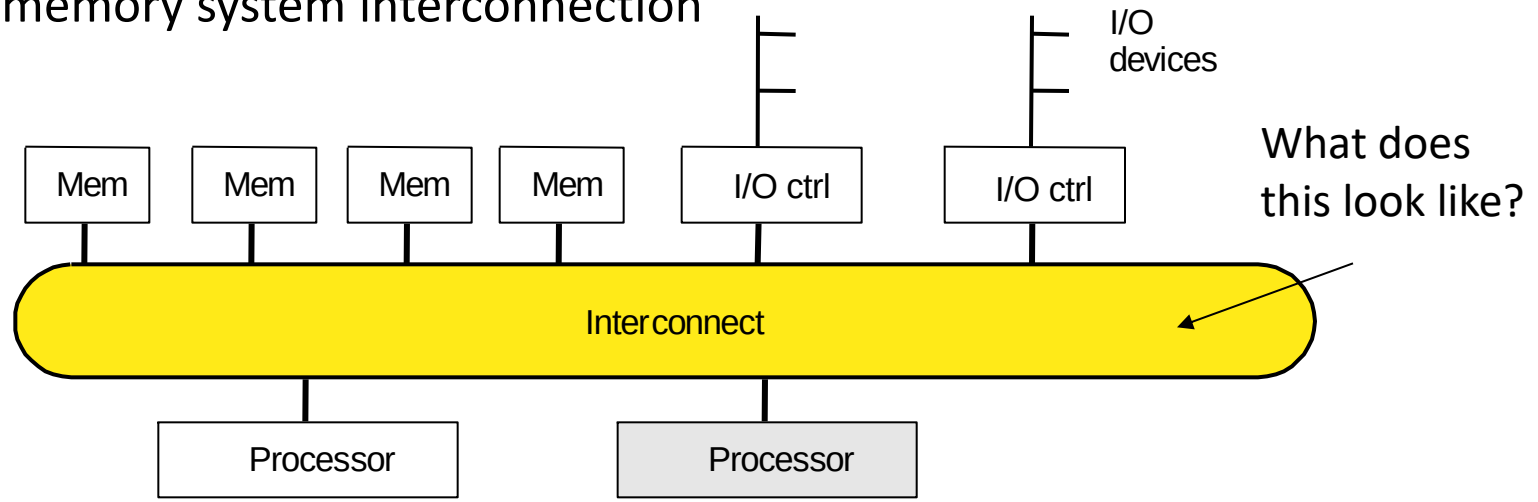


Shared Memory Multiprocessors (SMP)

- Architecture types

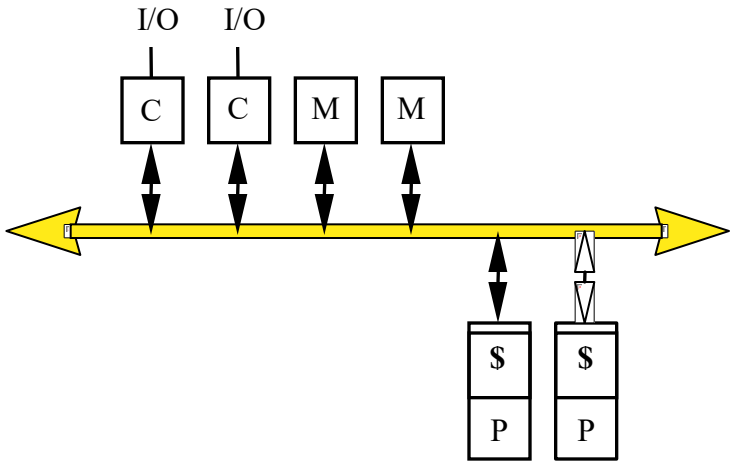


- Differences lie in memory system interconnection



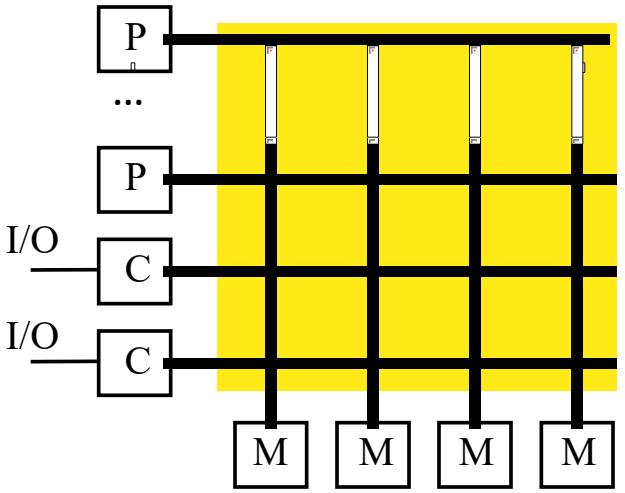
Bus-based SMP

- Memory bus handles all memory read/write traffic
- Processors share bus
- *Uniform Memory Access* (*UMA*)
  - Memory (not cache) uniformly equidistant
  - Take same amount of time (generally) to complete
- May have multiple memory modules
  - Interleaving of physical address space
- Caches introduce memory hierarchy
  - Lead to data consistency problems
  - Cache coherency hardware necessary (*CC-UMA*)



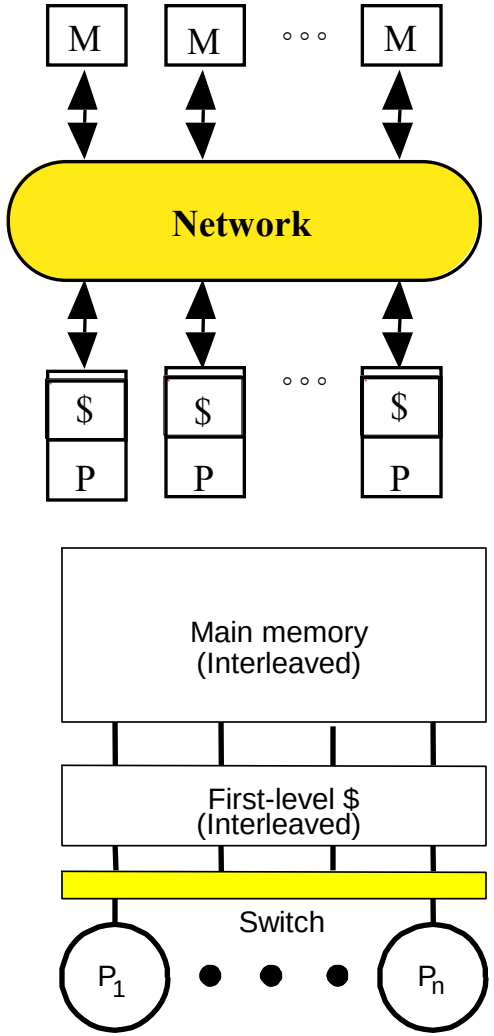
Crossbar SMP

- Replicates memory bus for every processor and I/O controller
  - Every processor has direct path
- UMA SMP architecture
- Can still have cache coherency issues
- Multi-bank memory or interleaved memory
- Advantages
  - Bandwidth scales linearly (no shared links)
- Scalability Problems
  - High incremental cost (cannot afford for many processors)
  - Use switched multi-stage interconnection network



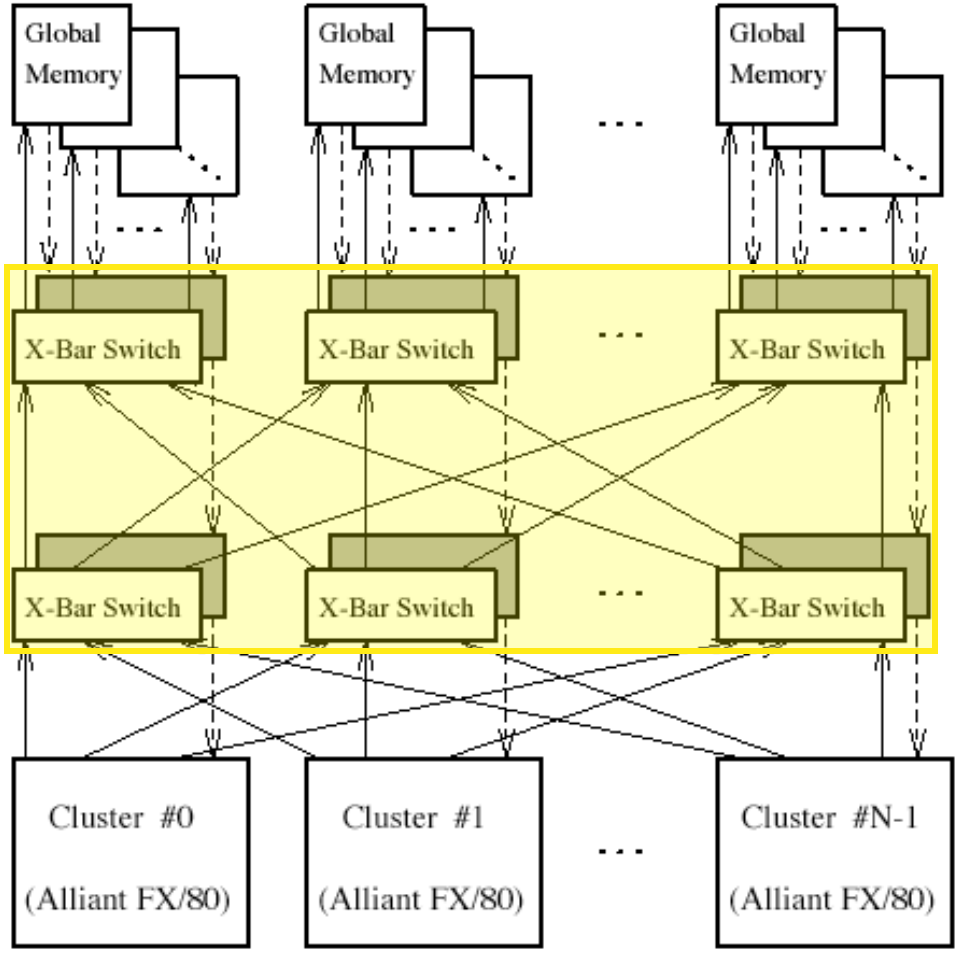
SMP and Shared Cache

- *Interconnection network connects processors to memory*
- Centralized memory (UMA)
- Network determines performance
  - Continuum from bus to crossbar
  - Scalable memory bandwidth
- Memory is physically separated from processors
- *Could have cache coherence problems*
- *Shared cache* reduces coherence problem and provides fine grained data sharing

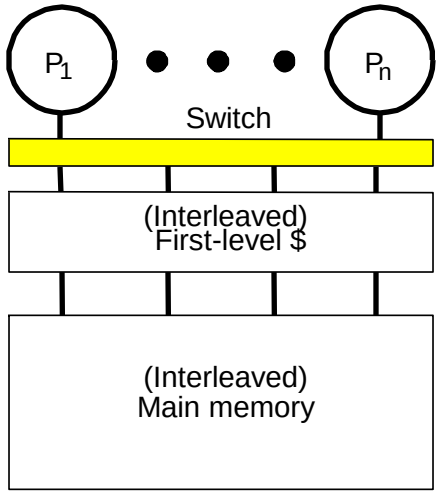


Example: University of Illinios CSRD Cedar Machine

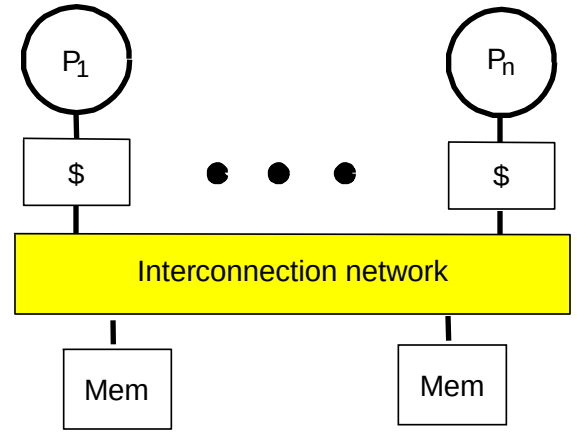
- Center for Supercomputing Research and Development
- Multi-cluster scalable parallel computer
- Alliant FX/80
  - 8 processors w/ vectors
  - Shared cache
  - HW synchronization
- Omega switching network
- Shared global memory
- SW-based global memory coherency



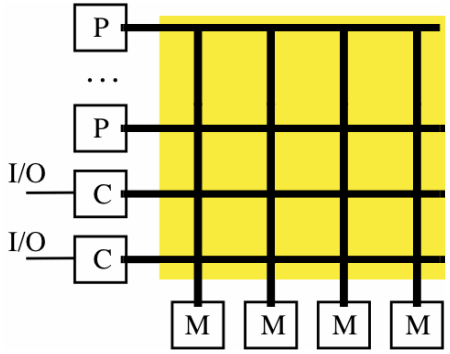
Natural Extensions of the Memory System



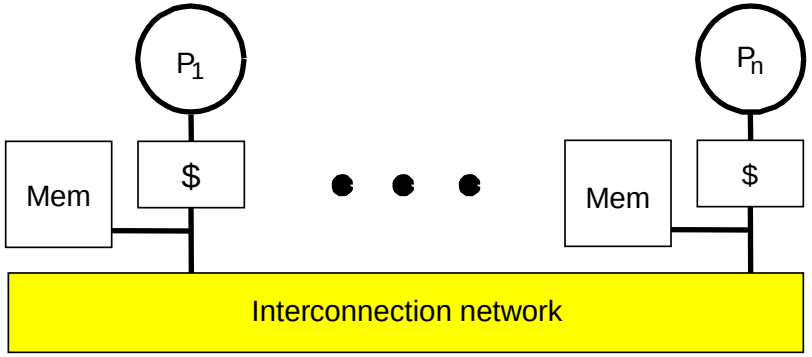
Shared Cache



Centralized Memory
Dance Hall, UMA

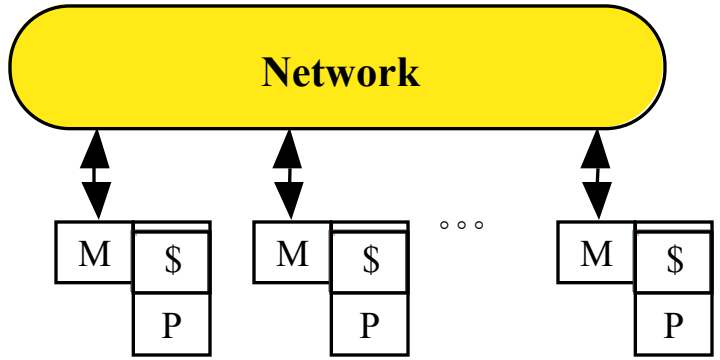


Crossbar, Interleaved



Distributed Memory (NUMA)

Non-Uniform Memory Access (NUMA) SMPs



- Distributed memory
- ***Memory is physically resident close to each processor***
- ***Memory is still shared!***
- *Non-Uniform Memory Access* (NUMA)
  - Local memory and remote memory
  - Access to local memory is faster, remote memory slower
  - Access is non-uniform
  - Performance will depend on data locality
- Cache coherency is still an issue (more difficult)
- Interconnection network architecture is more scalable!!!

Cache Coherency and SMPs

- Caches play key role in SMP performance
  - Reduce average data access time
  - Reduce bandwidth demands placed on shared interconnect
- Private processor caches create a problem
  - Copies of a variable can be present in multiple caches
  - A write by one processor may not become visible to others
    - they'll keep accessing stale value in their caches

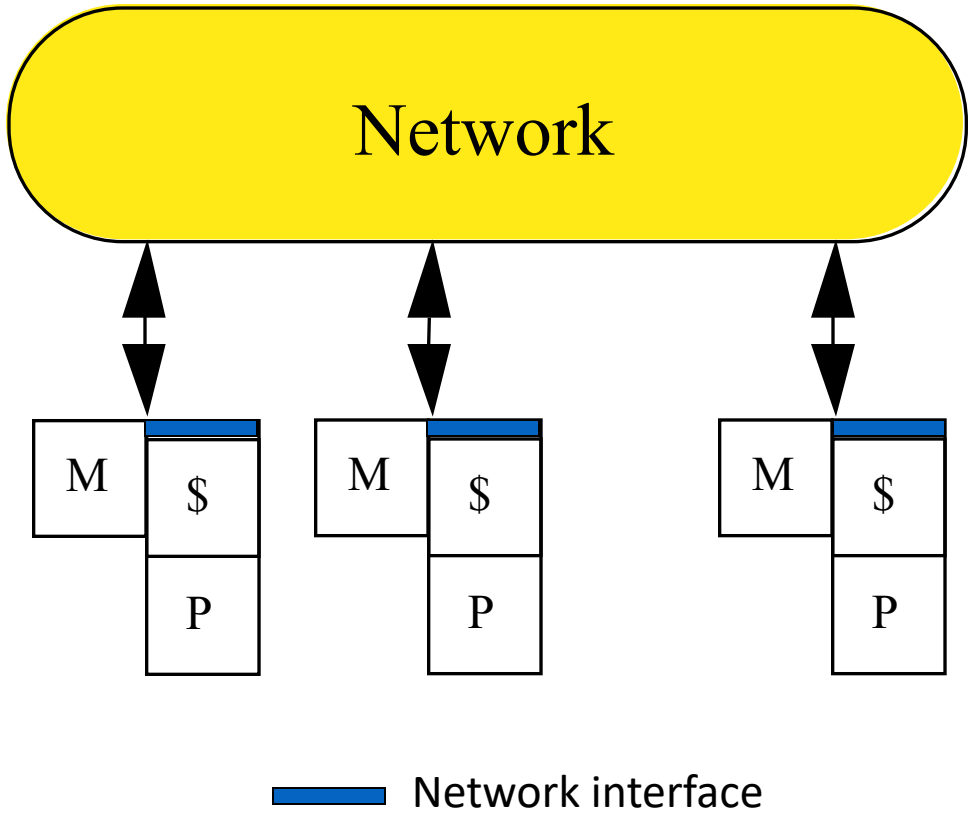
  $\Rightarrow$ *Cache coherence* problem
- What do we do about it?
  - *Organize the memory hierarchy*
  - Detect and take actions to eliminate the problem

Distributed Memory Multiprocessors

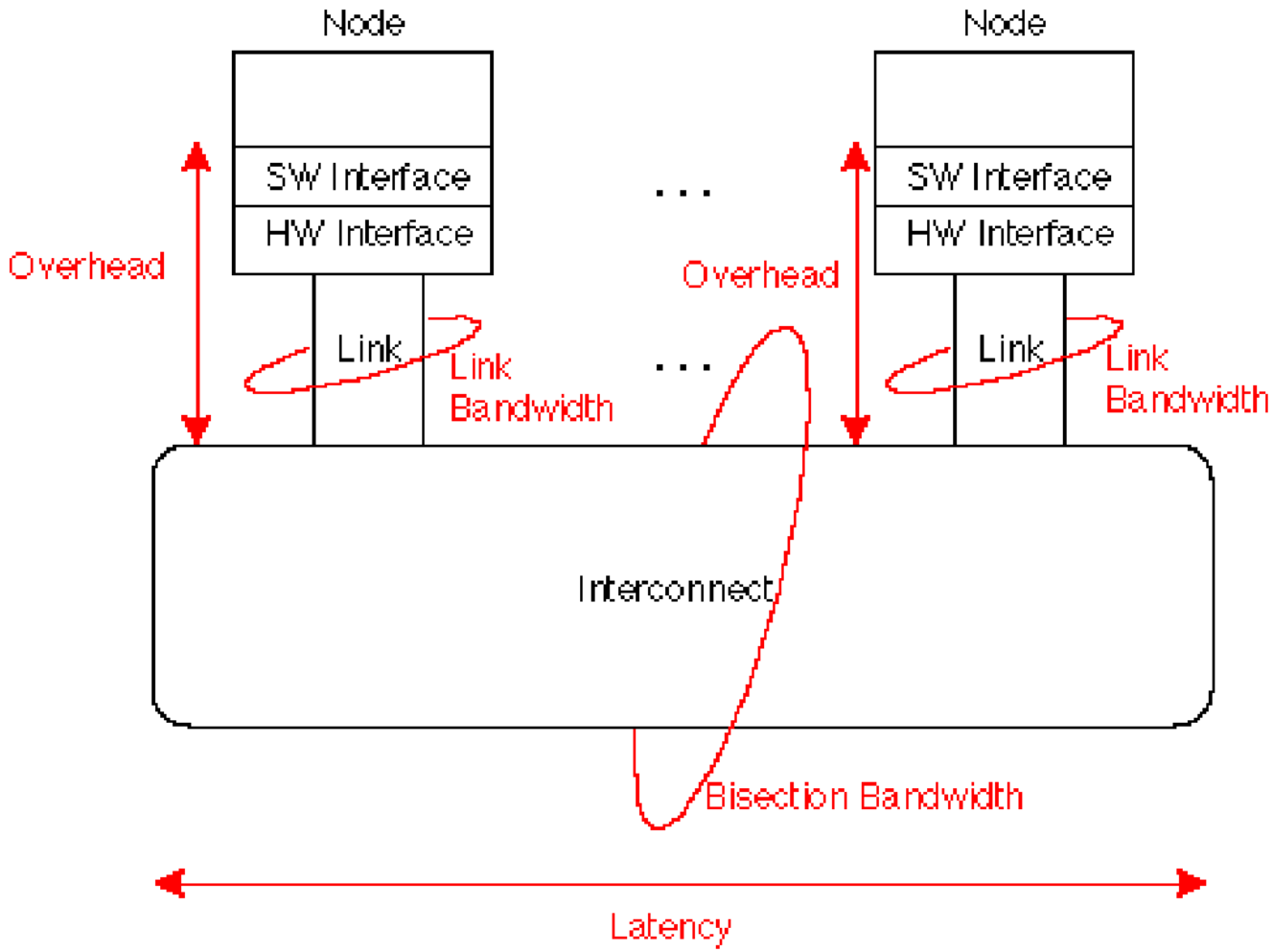
- Each processor has a local memory
  - ***Physically separated memory address space***
- Processors must communicate to access non-local data
  - Message communication (message passing)
    - *Message passing architecture*
  - Processor interconnection network
- ***Parallel applications must be partitioned across***
  - ***Processors: execution units***
  - ***Memory: data partitioning***
- Scalable architecture
  - Small incremental cost to add hardware (cost of node)

Distributed Memory (MP) Architecture

- Nodes are complete computer systems
  - Including I/O
- Nodes communicate via interconnection network
  - Standard networks
  - Specialized networks
- Network interfaces
  - Communication integration
- Easier to build



Network Performance Measures



Overhead: latency of interface vs. Latency: network

Performance Metrics: Latency and Bandwidth

- Bandwidth
  - Need high bandwidth in communication
  - Match limits in network, memory, and processor
  - Network interface speed vs. network bisection bandwidth
- Latency
  - Performance affected since processor may have to wait
  - Harder to overlap communication and computation
  - Overhead due to communication is a problem in many machines
- Latency hiding
  - Increases programming system burden
  - Examples: communication/computation overlaps, prefetch

Scalable, High-Performance Interconnect

- Interconnection network is the core of parallel architecture (... next lecture...)
- Requirements and tradeoffs at many levels
  - Elegant mathematical structure
  - Deep relationship to algorithm structure
  - Hardware design sophistication
- Little consensus
  - Performance metrics?
  - Cost metrics?
  - Workload?
  - …

Networks of Real Machines (circa 2000)

Machine	Topology	Speed	Width	Delay	Flit
nCUBE/2	hypercube	25 ns	1	40 cycles	32
CM-5	fat-tree	25 ns	4	10 cycles	4
SP-2	banyan	25 ns	8	5 cycles	16
Paragon	2D mesh	11.5 ns	16	2 cycles	16
T3D	3D torus	6.67 ns	16	2 cycles	16
DASH	torus	30 ns	16	2 cycles	16
Origin	hypercube	2.5 ns	20	16 cycles	160
Myrinet	arbitrary	6.25 ns	16	50 cycles	16

A message is broken up into *flits* for transfer.

Message Passing Model

- Hardware maintains send and receive message buffers
- Send message (synchronous)
  - Build message in local message send buffer
  - Specify receive location (processor id)
  - Initiate send and wait for receive acknowledge
- Receive message (synchronous)
  - Allocate local message receive buffer
  - Receive message byte stream into buffer
  - Verify message (e.g., checksum) and send acknowledge
- Memory to memory copy with acknowledgement and pairwise synchronization

Advantages of Shared Memory Architectures

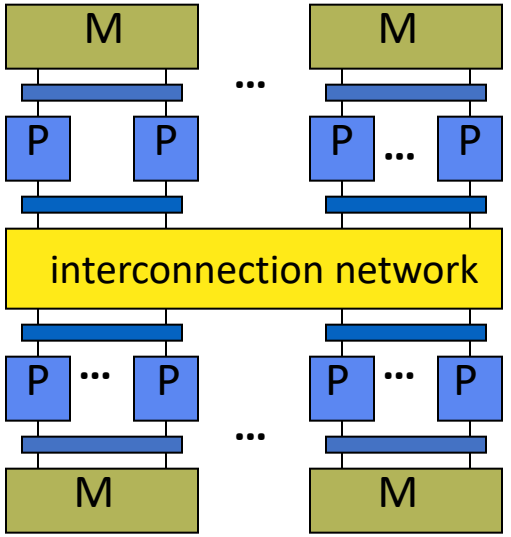
- Compatibility with SMP hardware
- Ease of programming when communication patterns are complex or vary dynamically during execution
- Ability to develop applications using familiar SMP model, attention only on performance critical accesses
- Lower communication overhead, better use of BW for small items, due to implicit communication and memory mapping to implement protection in hardware, rather than through I/O system
- HW-controlled caching to reduce remote communication by caching of all data, both shared and private

Advantages of Distributed Memory Architectures

- The hardware can be simpler (especially versus NUMA) and is more scalable
- Communication is explicit and simpler to understand
- Explicit communication focuses attention on costly aspect of parallel computation
- Synchronization is naturally associated with sending messages, reducing the possibility for errors introduced by incorrect synchronization
- Easier to use sender-initiated communication, which may have some advantages in performance

Clusters of SMPs

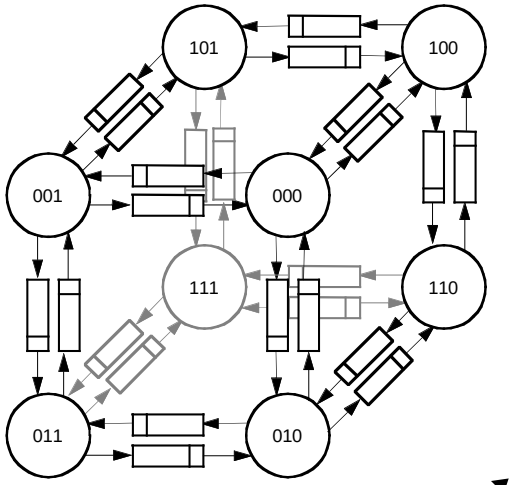
- Clustering
  - Integrated packaging of nodes
- Motivation
  - Amortize node costs by sharing packaging and resources
  - Reduce network costs
  - Reduce communications bandwidth requirements
  - Reduce overall latency
  - More parallelism in a smaller space
  - Increase node performance
- Scalable parallel systems today are built as SMP clusters !!!



Examples...

Example:CalTech Cosmic Cube

- First distributed memory message passing system
- Hypercube-based communications network



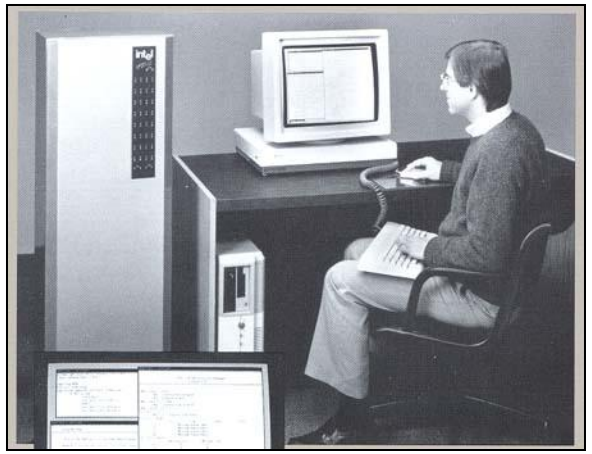
- Chuck Seitz, Geoffrey Fox

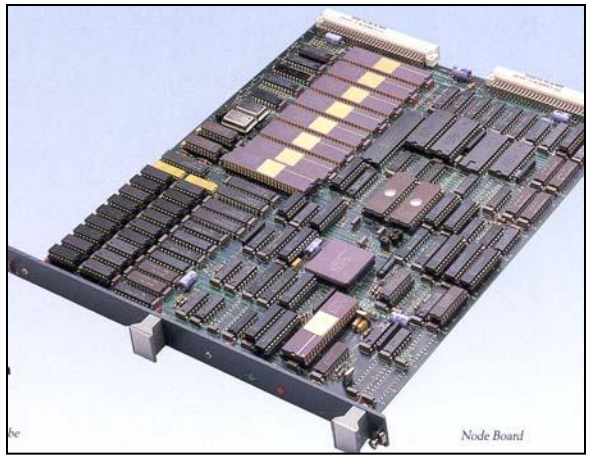
FIFO on each link
- store and forward

Compute node

Example:Intel iPSC/1, iPSC/2, iPSC/860

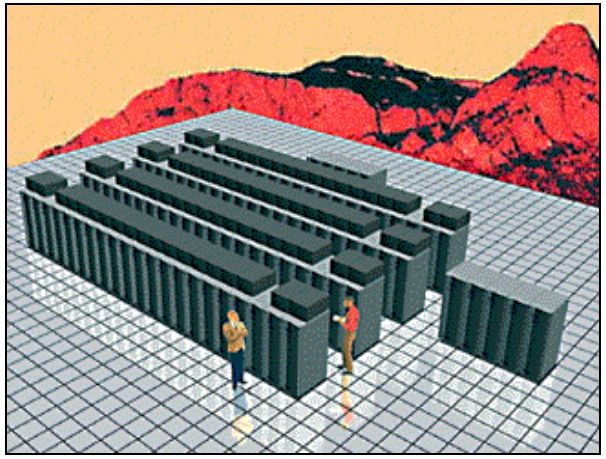
- Shift to general links
  - DMA, enabling non-blocking ops
    - Buffered by system at destination until recv
  - Store&forward routing
- Diminishing role of topology
  - Any-to-any pipelined routing
  - node-network interface dominates communication time
  - Simplifies programming
  - Allows richer design space

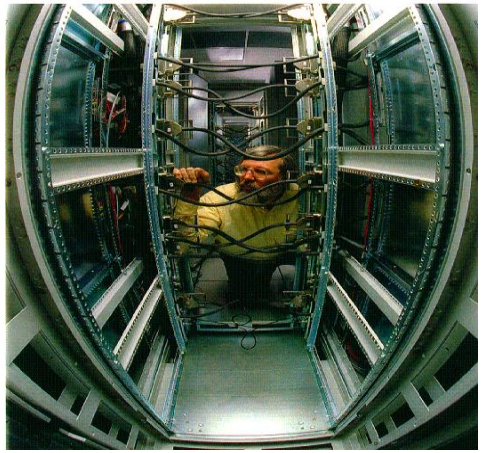


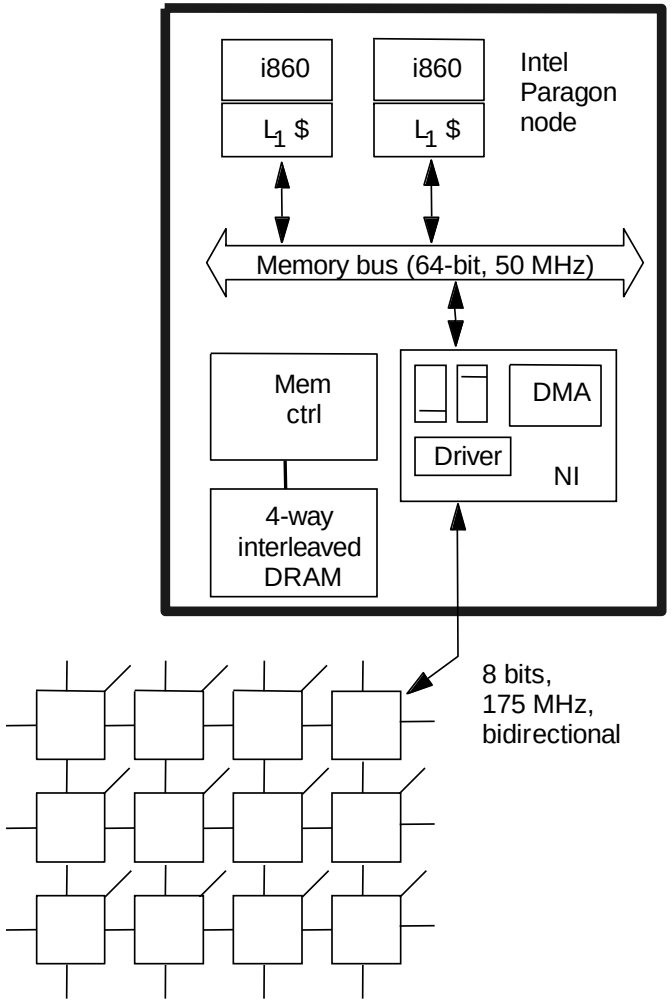


Example: Intel Paragon and ASCI Red

- DARPA project machine
  - Intel i860 processor
  - 2D grid network with processor node attached to every switch
  - 8bit, 175 MHz bidirectional links
- Forerunner design for ASCI Red
  - First Teraflop computer

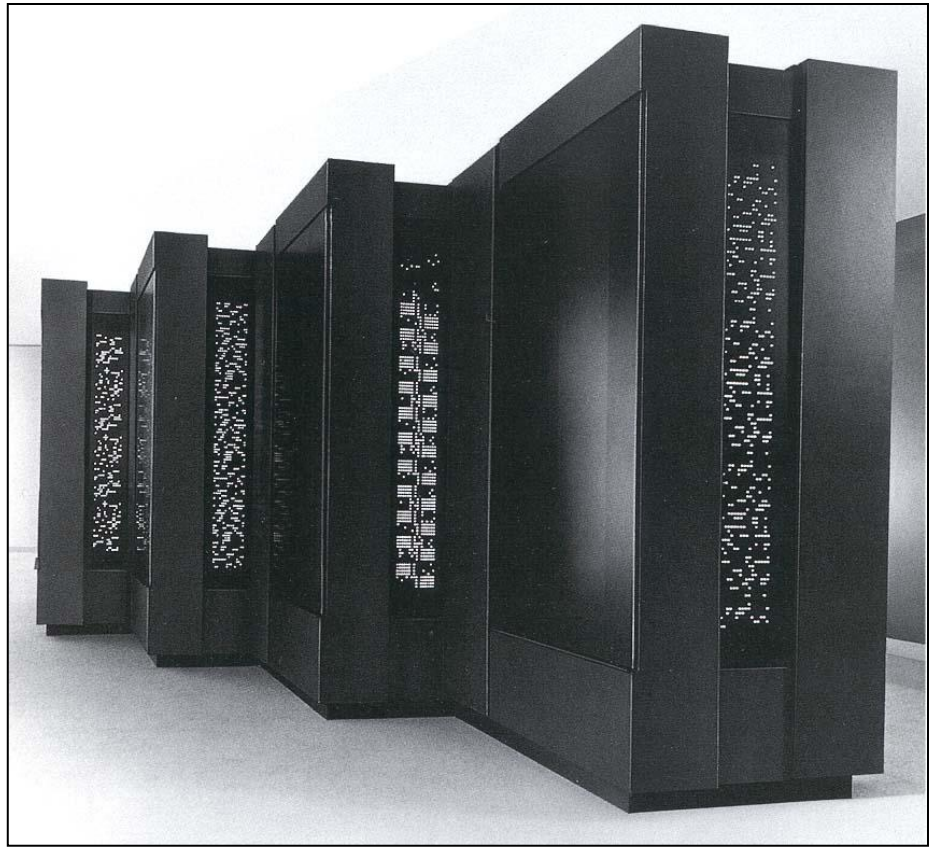






Example: Thinking Machine CM-5

- Repackaged SparcStation
  - 4 per board
- Fat-Tree network
- Control network for global synchronization
- Suffered from hardware design and installation problems

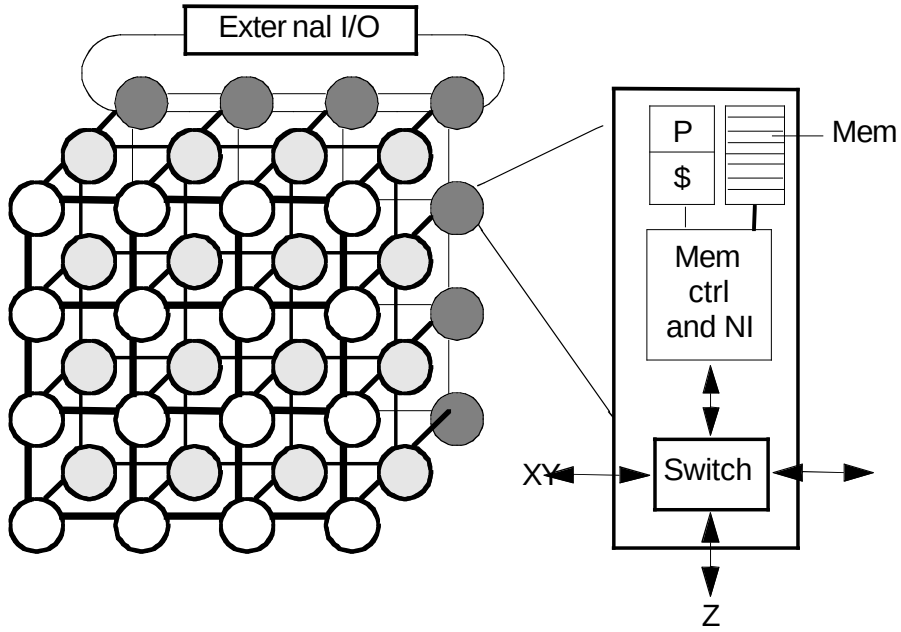


Example: Berkeley Network Of Workstations (NOW)

- 100 Sun Ultra2 workstations
- Inteligent network interface
  - proc + mem
- Myrinet network
  - 160 MB/s per link
  - 300 ns per hop

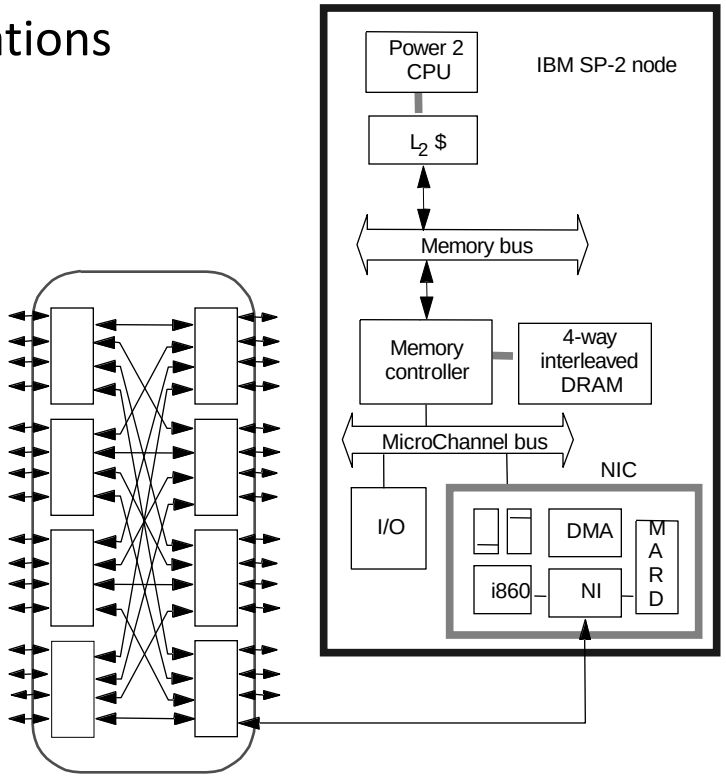
Example: Cray T3E

- Up to 1024 nodes
- 3D torus network
  - 480 MB/s links
- No memory coherence
- Access remote memory
  - Converted to messages
  - SHared MEMory communication
    - *put* / *get* operations
- Very successful machine



Example: IBM SP-2

- Made out of essentially complete RS6000 workstations
- Network interface integrated in I/O bus
- SP network very advanced
  - Formed from 8-port switches
- Predecessor design to
  - ASCI Blue Pacific (5856 CPUs)
  - ASCI White (8192 CPUs)



Example: NASA Columbia

- System hardware
  - 20 SGI Altix 3700 superclusters
    - 512 Itanium2 processors (1.5 GHz)
    - 1 TB memory
  - 10,240 processors (now 13,312)
  - NUMAflex architecture
  - NUMAlink “fat tree” network
  - Fully shared memory!!!
- Software
  - Linux with PBS Pro job scheduling
  - Intel Fortran/C/C++ compilers

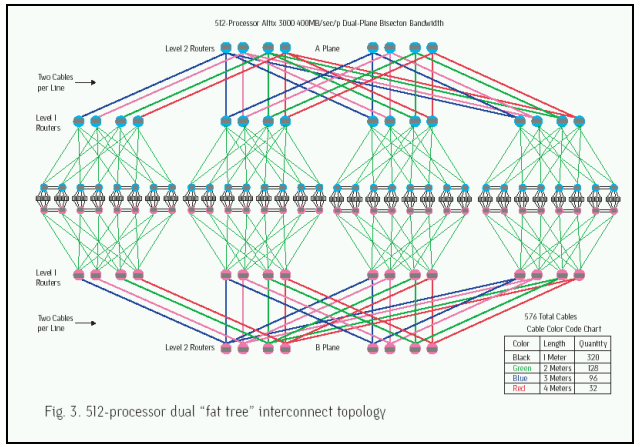
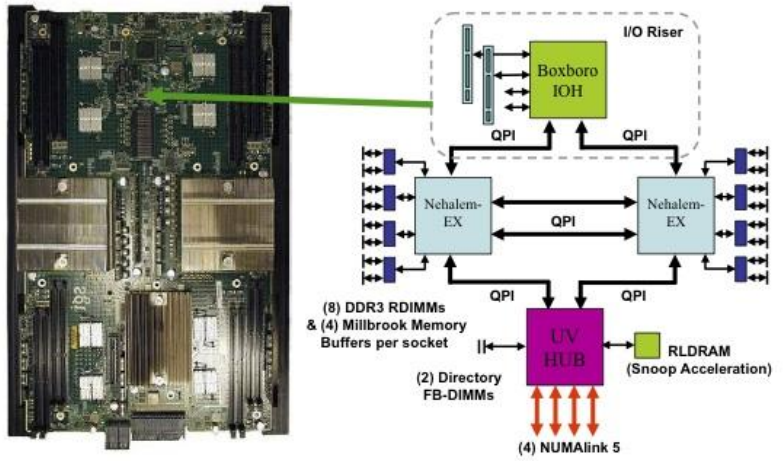
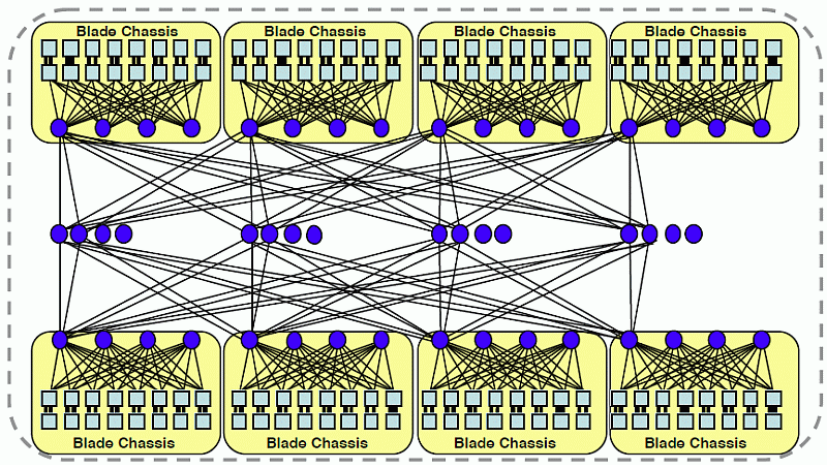


Fig. 3. 512-processor dual “fat tree” interconnect topology

Example: SGI Altix UV

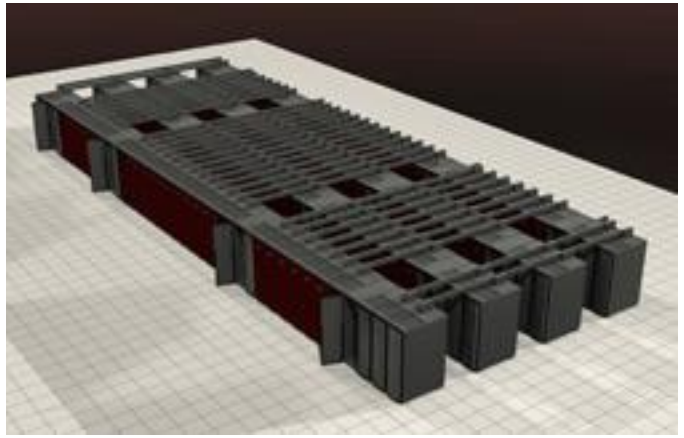
- Latest generation scalable shared memory architecture
- Scaling from 32 to 2,048 cores
  - Intel Nehalem EX
- Architectural provisioning for up to 262,144 cores
- Up to 16 terabytes of global shared memory in a single system image (SSI)
- High-speed 15GB per second interconnect NUMAlink 5

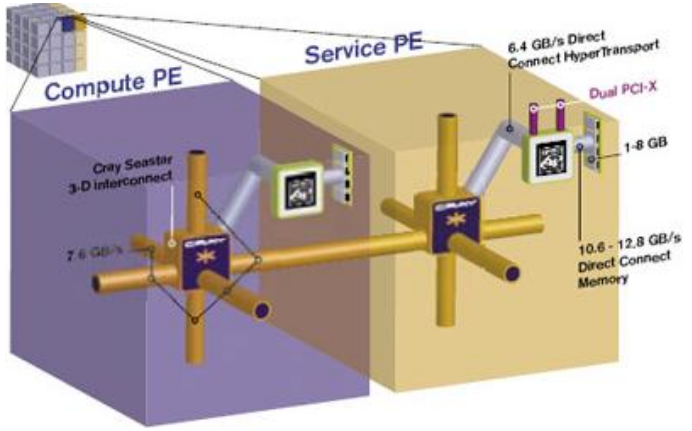




Example: Sandia Red Storm

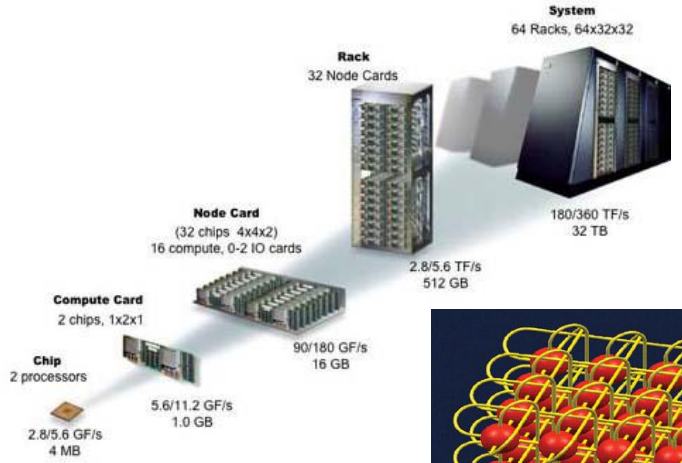
- System hardware
  - Cray XT3
  - 135 compute node cabinets
  - 12,960 processors
    - AMD Opteron dual-core
  - 320 / 320 service / I/O node processors
  - 40 TB memory
  - 340 TB disk
  - 3D mesh interconnect
- Software
  - Catamount compute node kernel
  - Linux I/O node kernel
  - MPI

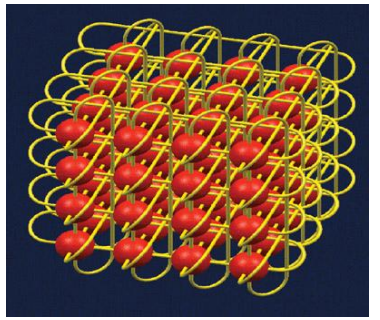




Example: LLNL BG/L

- System hardware
  - IBM BG/L (BlueGene)
  - 65,536 dual-processor compute nodes
    - PowerPC processors
    - “double hummer” floating point
  - I/O node per 32 compute nodes
  - 32x32x64 3D torus network
  - Global reduction tree
  - Global barrier and interrupt networks
  - Scalable tree network for I/O
- Software
  - Compute node kernel (CNK)
  - Linux I/O node kernel (ION)
  - MPI
  - Different operating modes





Example: Tokyo Institute of Technology TSUBAME

- System hardware
  - 655 Sun Fire X4600 servers
  - 11,088 processors
    - AMD Opteron dual-core
  - ClearSpeed accelerator
  - InfiniBand network
  - 21 TB memory
  - 42 Sun Fire X4500 servers
  - 1 PB of storage space
- Software
  - SuSE Linux Enterprise Server 9 SP3
  - Sun N1 Grid Engine 6.0
  - Lustre Client Software

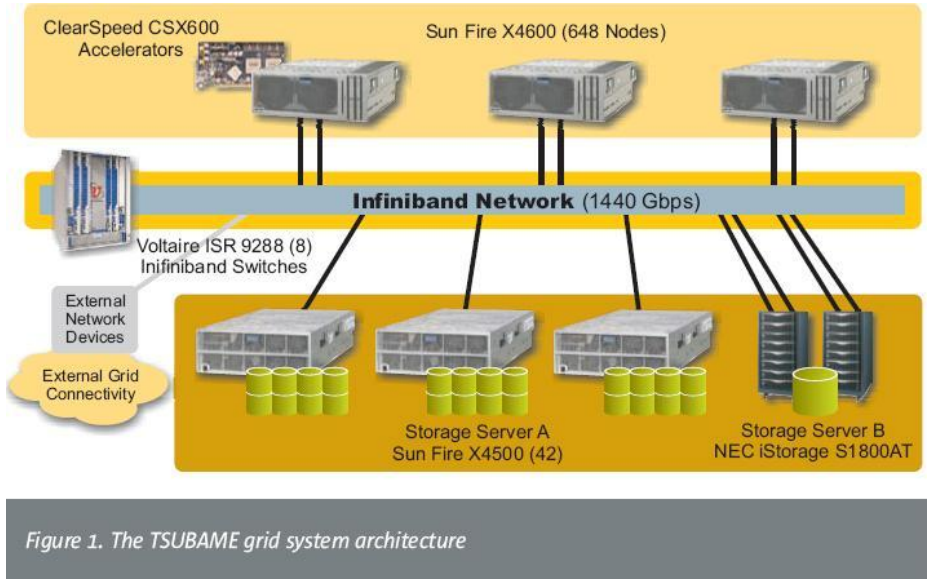
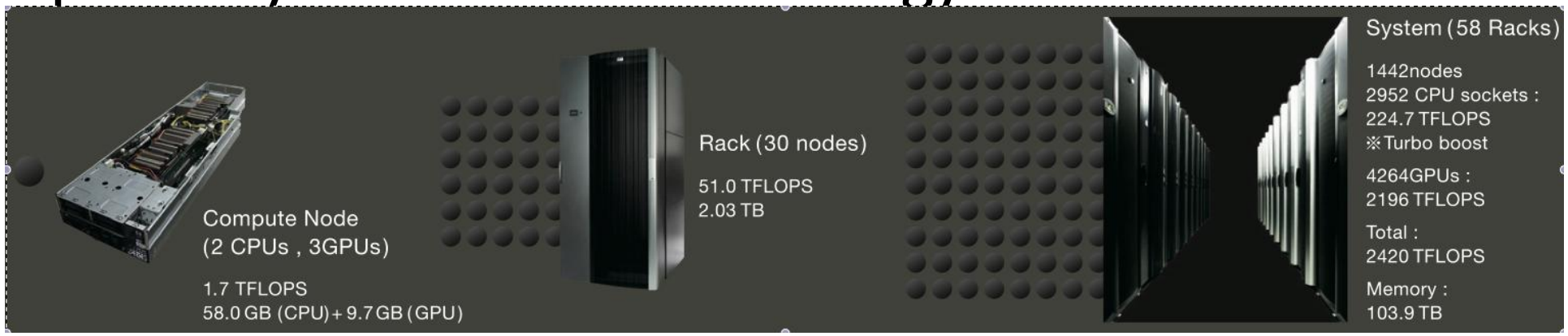


Figure 1. The TSUBAME grid system architecture

Example: Tokyo Institute of Technology TSUBAME2



Thin Node 1408 nodes

HP ProLiant SL390s

GPU : NVIDIA Tesla M2050 (Fermi Core)×3 515GFLOPS VRAM 3GB/GPU

CPU : Intel Xeon X5670 2.93GHz ×2
6 core/socket 76.7 GFLOPS (12cores/node) ※Turbo boost : 3.196GHz

Memory : 58GB DDR3 1333MHz (partly 103GB)

SSD : 60GB ×2 (120GB/node) (partly 120GB ×2 (240GB/node))

Medium Node 24 nodes

HP ProLiant DL580 G7

CPU: Intel Xeon X7550
(Nehalem-EX)
2.0 GHz ×4 sockets
(32cores/node)

GPU: NVIDIA Tesla S1070

Memory: 137 GB (DDR3 1066MHz)

SSD: 120GB ×4 (480GB/node)

Infiniband: QDR

Fat Node 10 nodes

HP ProLiant DL580 G7

CPU: Intel Xeon X7550
(Nehalem-EX)
2.0 GHz ×4 sockets
(32cores/node)

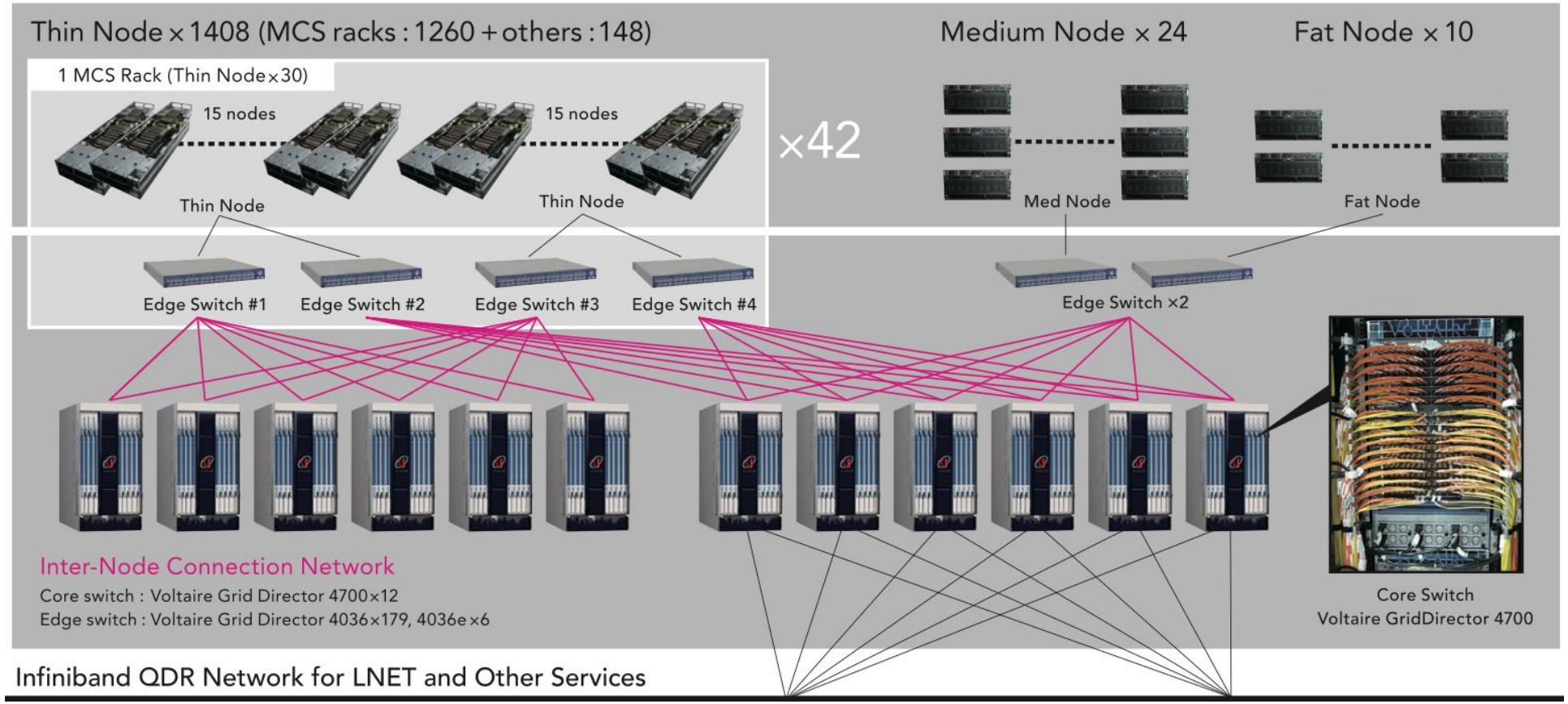
GPU: NVIDIA Tesla S1070

Memory: 274 GB (8 nodes),
548 GB (2 nodes)
DDR3 1066MHz

SSD: 120GB ×5 (600GB/node)

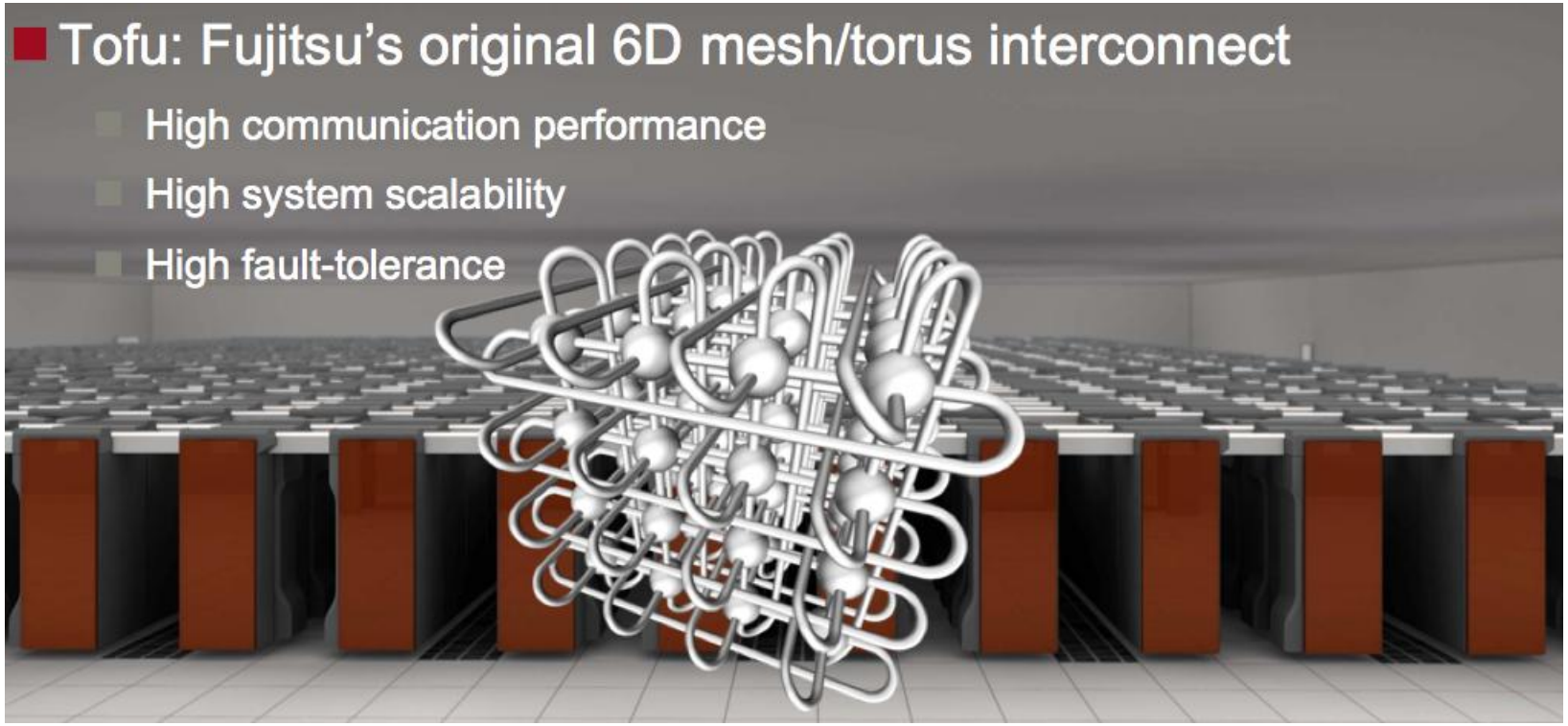
Infiniband: QDR

TSUBAME2 – Interconnect



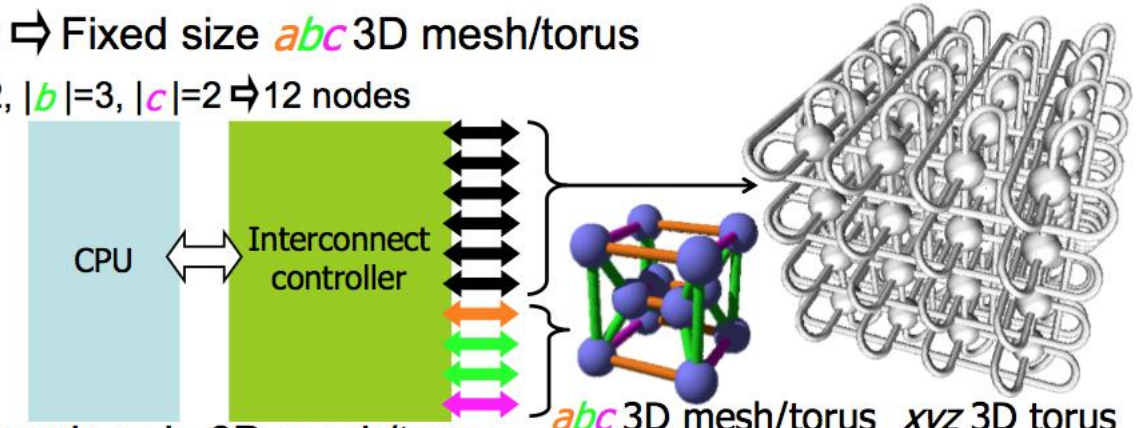
Example: Japanese K Computer – Interconnect

- 80,000 CPUs (SPARC64 VIIIfx), 640,000 cores
- 800 racks
- 8.6 Petaflops (Linpack)



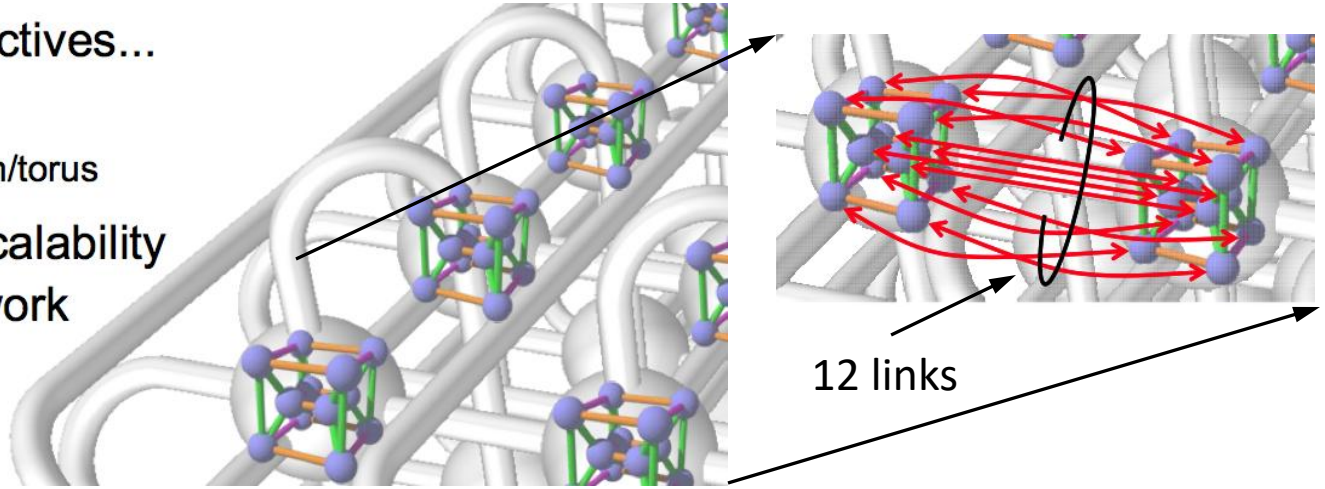
Japanese K Computer – Interconnect

- 6 links ⇨ Scalable *xyz* 3D torus
- 4 links ⇨ Fixed size *abc* 3D mesh/torus
  - |*a*|=2, |*b*|=3, |*c*|=2 ⇨ 12 nodes



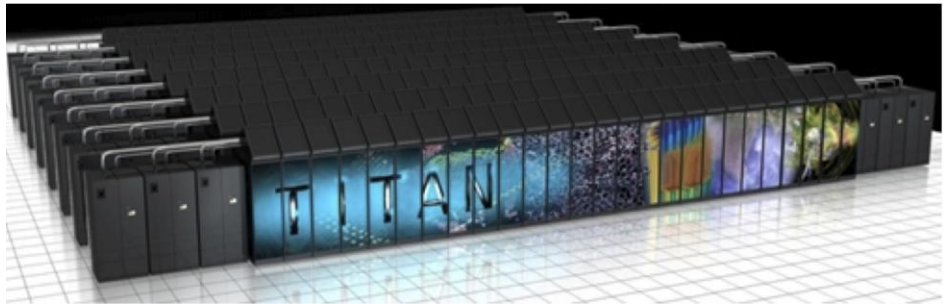
- Total topology is 6D mesh/torus
  - Cartesian product of *xyz* and *abc* mesh/torus

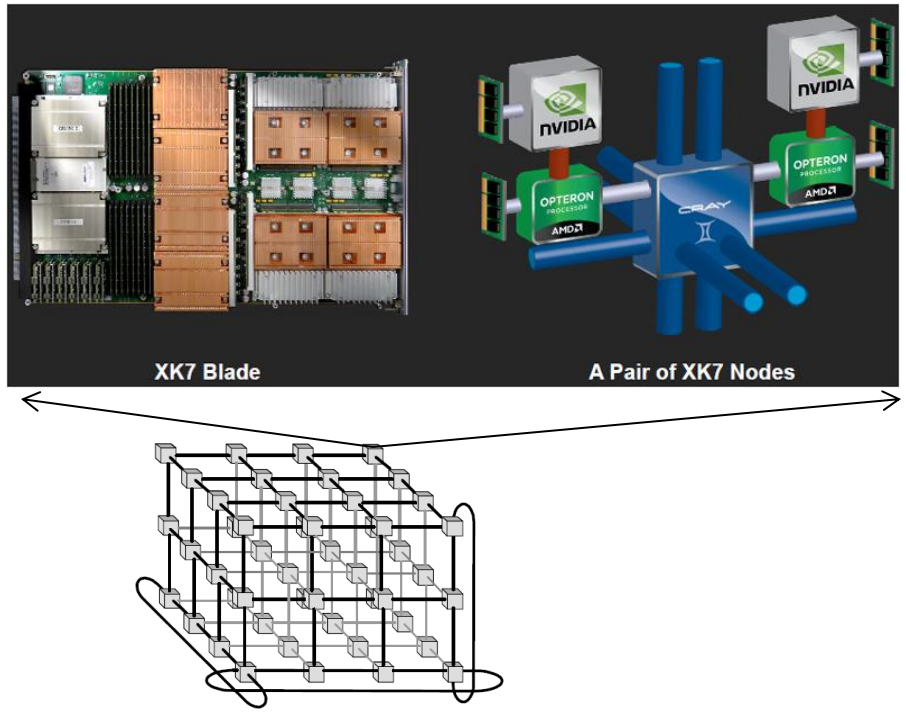
- From the other perspectives...
  - Overlaid twelve *xyz* torus
  - X x Y x Z array of *abc* mesh/torus
- Twelve times higher scalability than the 3D torus network



Example: ORNL Titan (http://www.olcf.ornl.gov/titan)

- Cray XK7
  - 18,688 nodes
  - AMD Opteron
    - 16-core Interlagos
    - 299,008 Opteron cores
  - NVIDIA K20x
    - 18,688 GPUs
    - 50,233,344 GPU cores
- Gemini interconnect
  - 3D torus
- 20+ petaflops



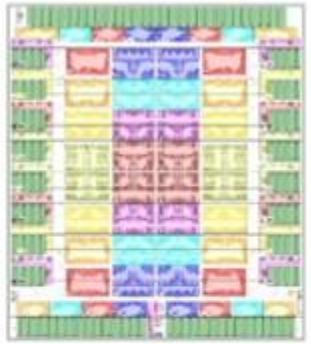


FRONTIER - HPE CRAY EX235A, AMD OPTIMIZED 3RD GENERATION EPYC 64C 2GHZ, AMD INSTINCT MI250X, SLINGSHOT-11 => 1ST TOP500

Slingshot Interconnect

Rosetta Switch

- Multiple QoS levels
- Aggressive adaptive routing
- Advanced congestion control
- Very low average and tail latency
- High performance multicast and reduction



64 ports x 200 Gbps

SS-10 (100Gb)
Injection: ~14 TB/s
Bisection: ~24 TB/s

SS-11 (200Gb)
Injection: ~28 TB/s
Bisection: ~24 TB/s

Mellanox ConnectX NIC

Slingshot 10

- HPE Cray MPI stack
- Ethernet functionality
- RDMA offload

Cassini NIC

Slingshot 11

- MPI hardware tag matching
- MPI progress engine
- One-sided operations
- Collectives
- 2X injection bandwidth

Cache coherency

Cache Coherency and SMPs

- Caches play key role in SMP performance
  - Reduce average data access time
  - Reduce bandwidth demands placed on shared interconnect
- Private processor caches create a problem
  - Copies of a variable can be present in multiple caches
  - A write by one processor may not become visible to others
    - they'll keep accessing stale value in their caches

  $\Rightarrow$ *Cache coherence* problem
- What do we do about it?
  - *Organize the memory hierarchy*
  - Detect and take actions to eliminate the problem

Definitions

- Memory operation (load, store, read-modify-write, …)
- Memory issue is operation presented to memory system
- Processor perspective
  - Write: subsequent reads return the value
  - Read: subsequent writes cannot affect the value
- *Coherent memory system*
  - There exists a serial order of memory operations on each location such that
    - operations issued by a process appear in order issued
    - value returned by each read is that written by previous write

  $\Rightarrow$ write propagation + write serialization

Motivation for Memory Consistency

- Coherence implies that writes to a location become visible to all processors in the same order
- But when does a write become visible?
- How do we establish orders between a write and a read by different processors?
  - Use event synchronization
- Implement hardware protocol for cache coherency
- Protocol will be based on model of memory consistency

P_1	P_2
/* Assume initial value of A and flag is 0 */	
`A = 1;`	`while (flag == 0);` /* spin idly */
`flag = 1;`	`print A;`

Memory Consistency

- Specifies constraints on the order in which memory operations (from any process) can appear to execute with respect to each other
  - What orders are preserved?
  - Given a load, constrains the possible values returned by it
- Implications for both programmer and system designer
  - Programmer uses to reason about correctness
  - System designer can use to constrain how much accesses can be reordered by compiler or hardware
- Contract between programmer and system
- Need coherency systems to enforce memory consistency

Sequential Consistency

- Total order achieved by interleaving accesses from different processes
  - Maintains *program order*
  - Memory operations (from all processes) appear to issue, execute, and complete atomically with respect to others
  - As if there was a single memory (no cache)

*"A multiprocessor is sequentially consistent if the result of **any execution** is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program." [Lamport, 1979]*

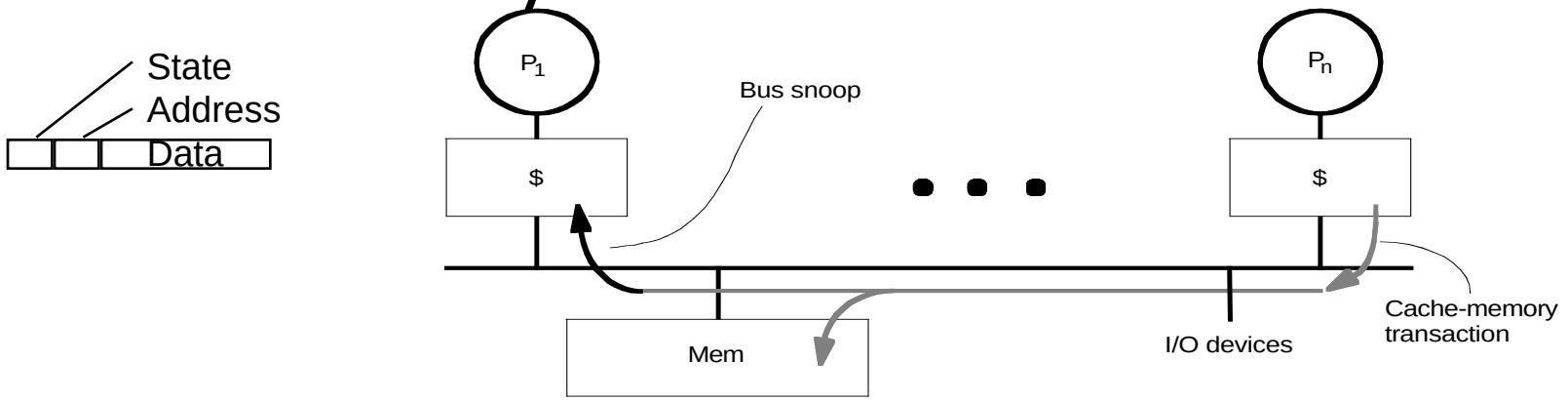
Sequential Consistency (Sufficient Conditions)

- There exist a total order consistent with the memory operations becoming visible in program order
- Sufficient Conditions
  - every process issues memory operations in program order
  - after write operation is issued, the issuing process waits for write to complete before issuing next memory operation (atomic writes)
  - after a read is issued, the issuing process waits for the read to complete and for the write whose value is being returned to complete (globally) before issuing its next memory operation
- Cache-coherent architectures implement consistency

Bus-based Cache-Coherent (CC) Architecture

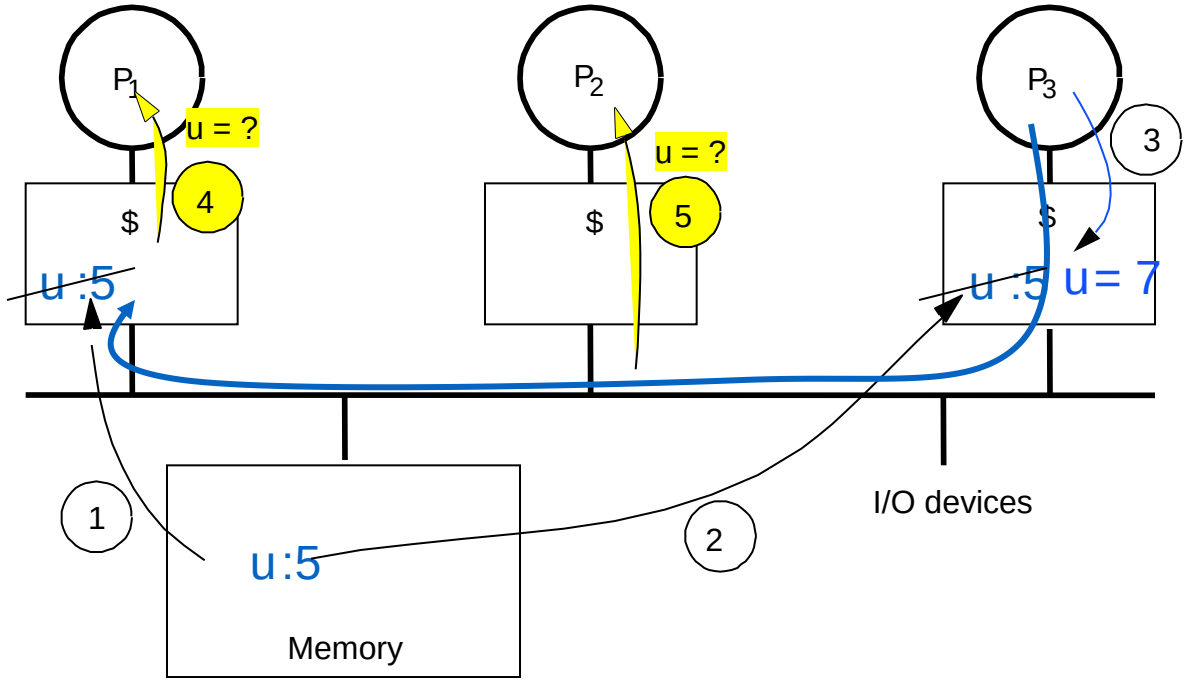
- Bus Transactions
  - Single set of wires connect several devices
  - Bus protocol: arbitration, command/addr, data
  - Every device observes every transaction
- Cache block state transition diagram
  - FSM specifying how disposition of block changes
    - invalid, valid, dirty
  - *Snoopy protocol*
- Basic Choices
  - Write-through vs Write-back
  - Invalidate vs. Update

Snoopy Cache-Coherency Protocols



- Bus is a broadcast medium
- Caches know what they have
- ***Cache controller*** "snoops" all transactions on shared bus
  - relevant transaction if looks for a block its cache contains
  - take action to ensure coherence
    - invalidate, update, or supply value
  - depends on state of the block and the protocol

Example: Write-back Invalidate



Requirements of a Cache Coherent System

- Provide set of states, state transition diagram, and actions
- Manage coherence protocol
  - (0) Determine when to invoke coherence protocol
  - (a) Find info about state of block in other caches to determine action
    - whether need to communicate with other cached copies
  - (b) Locate the other copies
  - (c) Communicate with those copies (invalidate/update)
- (0) is done the same way on all systems
  - state of the line is maintained in the cache
  - protocol is invoked if an "access fault" occurs on the line
- Different approaches distinguished by (a) to (c)

Bus-base Cache Coherence

- All of (a), (b), (c) done through broadcast on bus
  - faulting processor sends out a “search”
  - others respond to the search probe and take necessary action
- Could do it in scalable network too
- Conceptually simple, but broadcast doesn’t scale
  - on bus, bus bandwidth doesn’t scale
  - on scalable network, every fault leads to at least p network transactions
- Scalable coherence:
  - can have same cache states and state transition diagram
  - different mechanisms to manage protocol

Basic Snoop Protocols

- Write Invalidate :
  - Multiple readers, single writer
  - Write to shared data: an invalidate is sent to all caches
  - Read Miss:
    - Write-through: memory is always up-to-date
    - Write-back: snoop in caches to find most recent copy
- Write Broadcast (typically write through):
  - Write to shared data: broadcast on bus, snoop and update
- Write serialization: bus serializes requests!
- Write Invalidate versus Broadcast

Snooping Cache Variations

Basic Protocol	Berkeley Protocol	Illinois Protocol	MESI Protocol
Exclusive	Owned Exclusive	Private Dirty	Modified (private,!=Memory)
Shared	Owned Shared	Private Clean	Exclusive (private,=Memory)
Invalid	Shared	Shared	Shared (shared,=Memory)
	Invalid	Invalid	Invalid

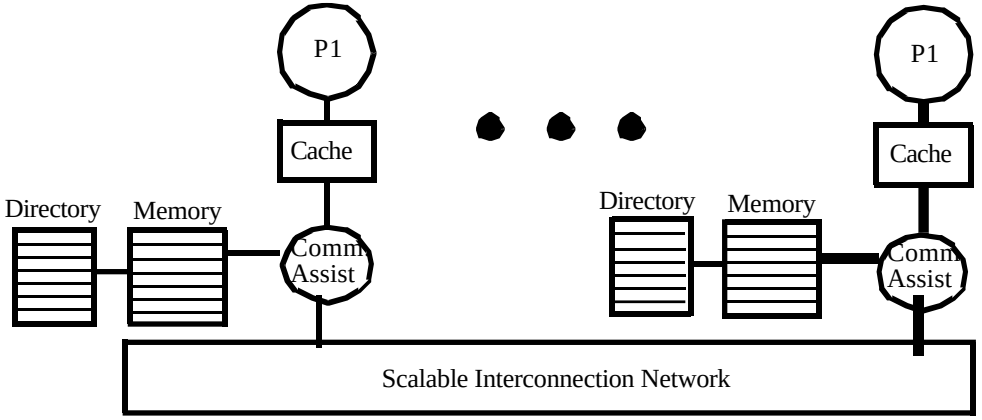
Owner can update via bus invalidate operation
Owner must write back when replaced in cache

If read sourced from memory, then Private Clean
if read sourced from other cache, then Shared
Can write in cache if held private clean or dirty

Scalable Approach: Directories

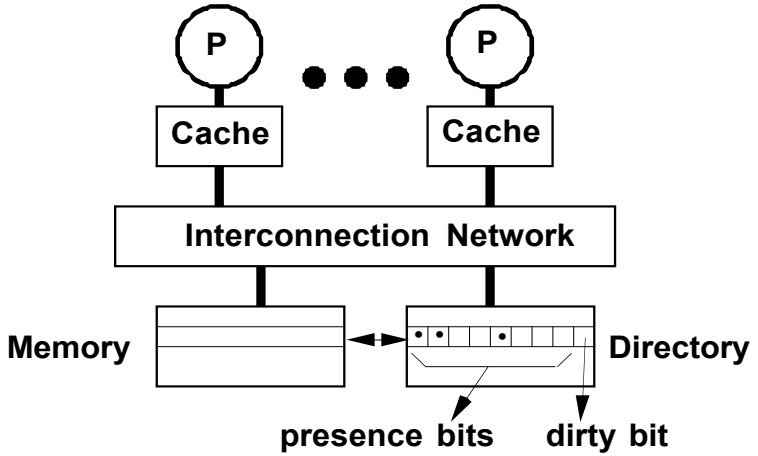
- Every memory block has associated directory information
  - keeps track of copies of cached blocks and their states
  - on a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary
  - in scalable networks, communication with directory and copies is through network transactions
- Many alternatives for organizing directory information

Generic Solution: Directories



- Maintain state vector explicitly
  - associate with memory block
  - records state of block in each cache
- On miss, communicate with directory
  - determine location of cached copies
  - determine action to take
  - conduct protocol to maintain coherence

Basic Operation of Directory

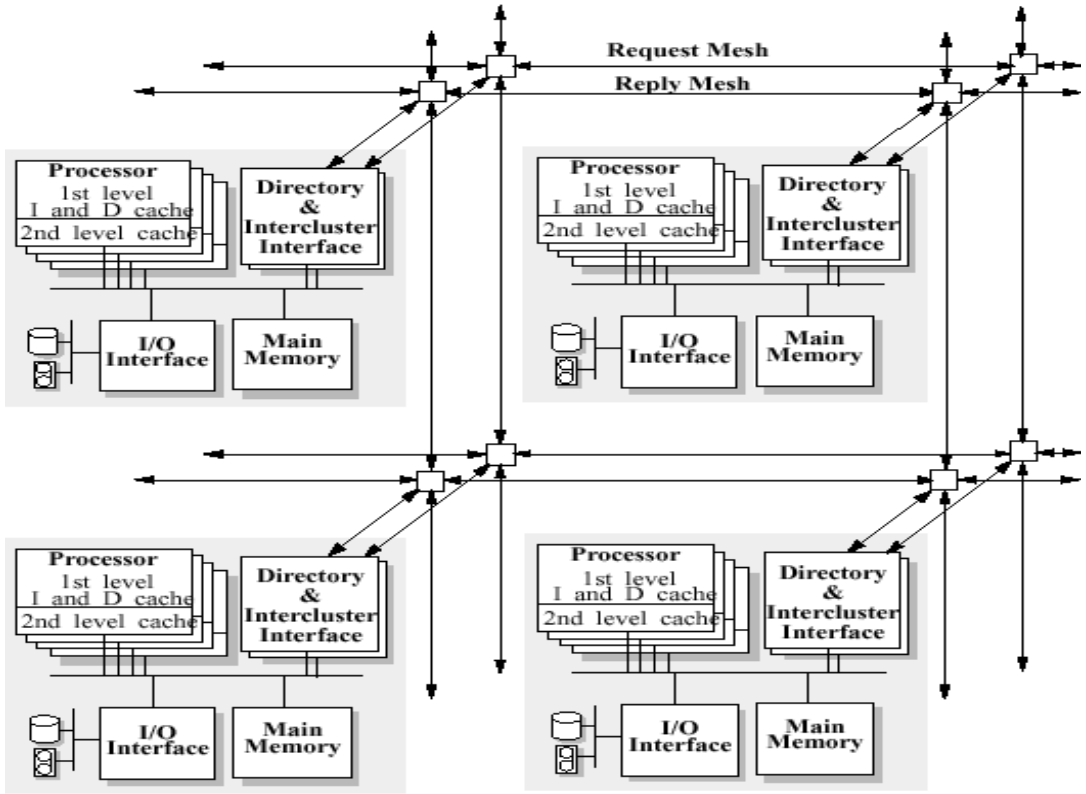


- k processors
- Each cache-block in memory
  - k presence bits and 1 dirty bit
- Each cache-block in cache
  - 1 valid bit and 1 dirty (owner) bit

- Read from memory
  - Dirty bit OFF
  - Dirty bit ON
- Write to memory
  - Dirty bit OFF

DASH Cache-Coherent SMP

- Directory Architecture for Shared Memory
- Stanford research project (early 1990s) for studying how to build cache-coherent shared memory architectures
- Directory-based cache coherency



- D. Lenoski, et al., “The Stanford Dash Multiprocessor,” IEEE Computer, Volume 25 Issue 3, pp: 63-79, March 1992