Lecture 15
Multiprocessors

Why Multiprocessors?

- Application requirements
  - Uniprocessor speed improving fast
  - But there are things that need even more speed
- Microprocessors as the fastest CPUs
  - Collecting several much easier than redesigning one
- Complexity of current microprocessors
  - Do we have enough ideas to sustain 1.5X/yr?
  - Can we deliver such complexity on schedule?
- Slow (but steady) improvement in parallel software
  - (scientific apps, databases, OS)

Opportunities for Applications

- **Scientific computing**
  - Nearly unlimited demand (Grand challenge)

App	*Perf(GFLOPS)*	*Memory (GB)*
72 hour weather	3	1
Pharmaceutical design	100	10
Global Change,Genome	1000	1000

- **Successes in some real industries:**
  - Petroleum reservoir modeling
  - Automotive: crash simulation, drag analysis, engine
  - Aeronautics: airflow analysis, engine, structural mechanics
  - Pharmaceuticals: molecular modeling
  - Entertainment: full length movies ("Toy Story")
- **Commercial application**
  - Transaction processing, file servers, electronic CAD simulation,search engine
  - Examples: IBM RS6000, Tandem (Compaq) Himilaya

Parallel Processing

- Multiple processors working cooperatively on problems: *not* the same as multiprogramming
- Goals/Motivation
  - Performance: limits of uniprocessors

    ILP (branch prediction, RAW dependencies, memory)
  - Cost Efficiency: build big systems with commodity parts

    (uniprocessors)
  - Scalability: just add more processors to get more performance
  - Fault tolerance: One processor fails you still can continue processing

Parallel Computers

- Definition: "**A parallel computer is a collection of processiong elements that cooperate and communicate to solve large problems fast.**"

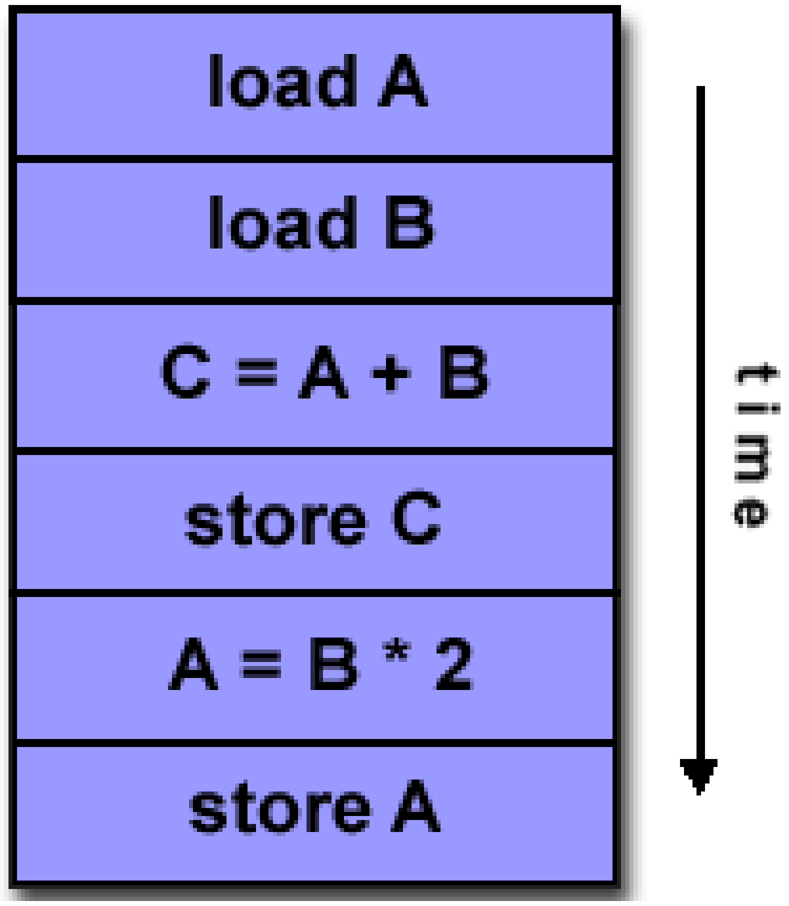
  Almasi and Gottlieb, *Highly Parallel Computing* ,1989

- Questions about parallel computers:
  – How large a collection?
  – How powerful are processing elements?
  – How do they cooperate and communicate?
  – How are data transmitted?
  – What type of interconnection?
  – What are HW and SW primitives for programmer?
  – Does it translate into performance?

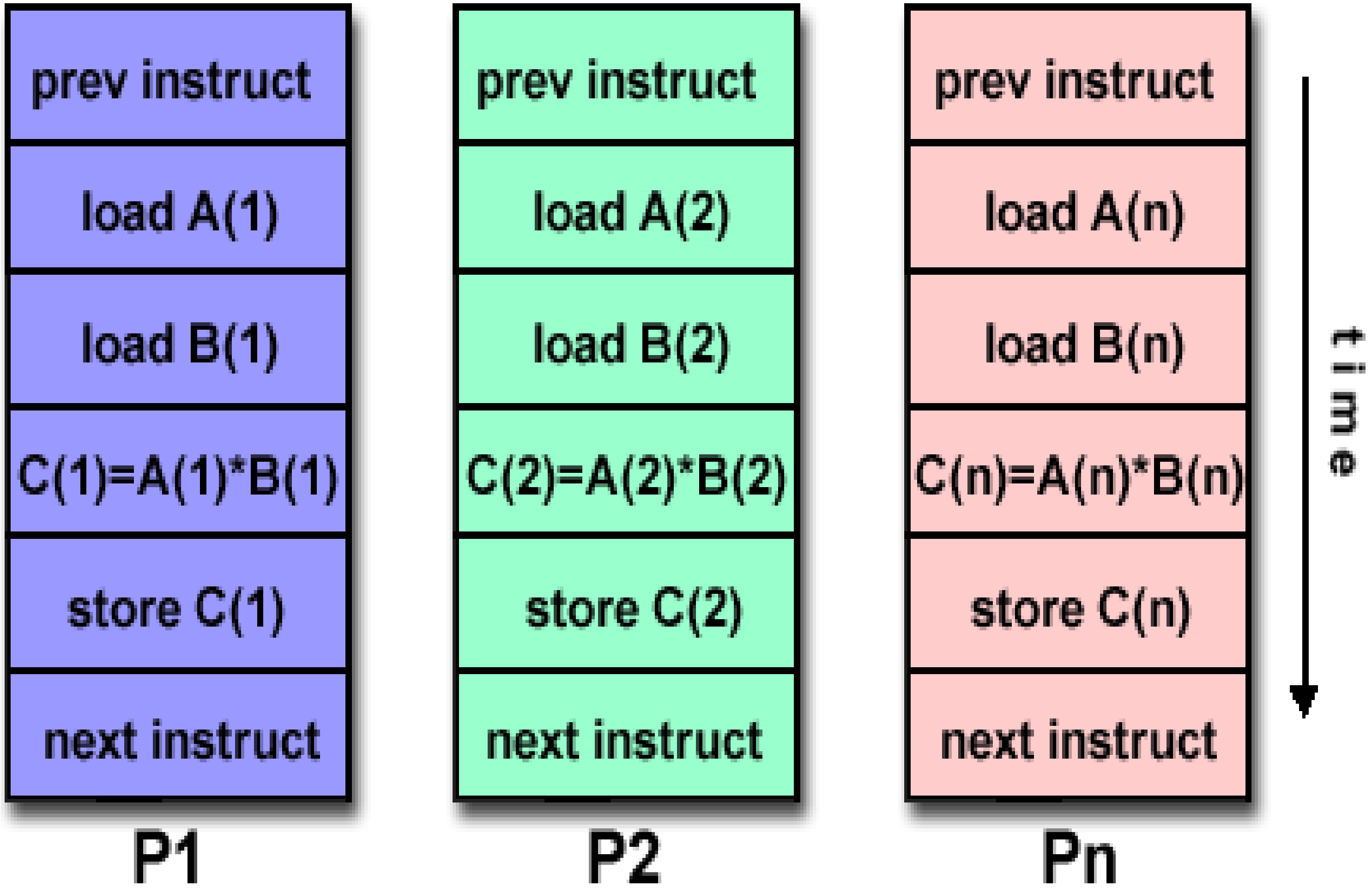
Popular Flynn Taxonomy

- Flynn's Taxonomy of Parallel Machines
  - How many Instruction streams?
  - How many Data streams?
- SISD (Single Instruction Single Data) ----Uniprocessors
- MISD (Multiple Instruction Single Data) ---- Not used much
- SIMD (Single Instruction Multiple Data)
  - Each "processor" works on its own data, but execute the same instr.
  - Examples: connection machine 2：65535个 1bit processors;
    Illiac IV： 64个 64bit processors;
  - Ad: Simple programming model; Low overhead; Flexibility;
- MIMD (Multiple Instruction Multiple Data)
  - Each processor executes its own instr. and operates on its own data
  - Examples: Sun Enterprise 5000, Cray T3D, SGI Origin
  - Ad: Flexible; Use off-the-shelf microprocessors;
  - *Not superscalar, each node is superscalar, lessons will apply to multi-core*

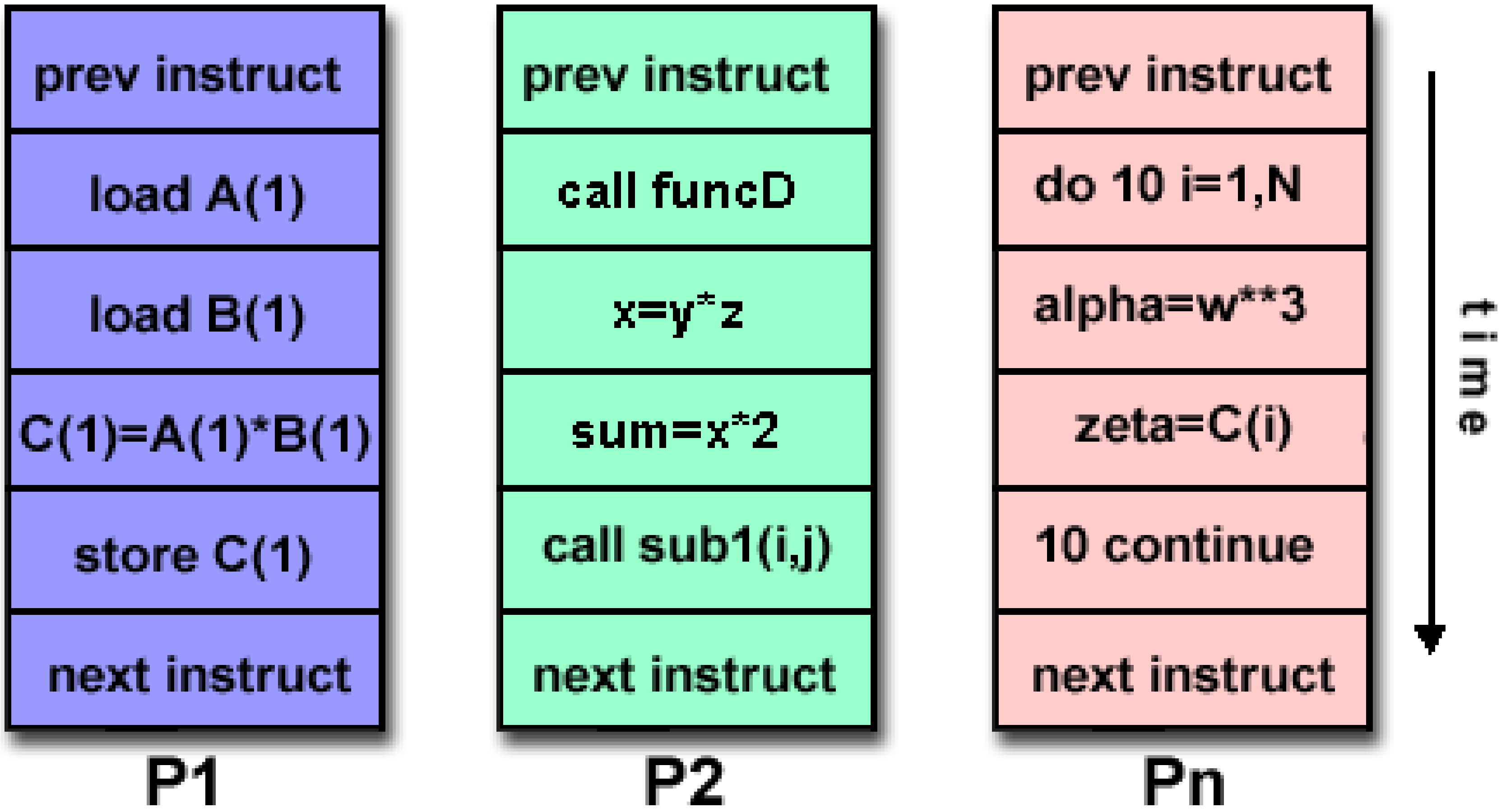
Program execute on SISD



Program execute on SIMD



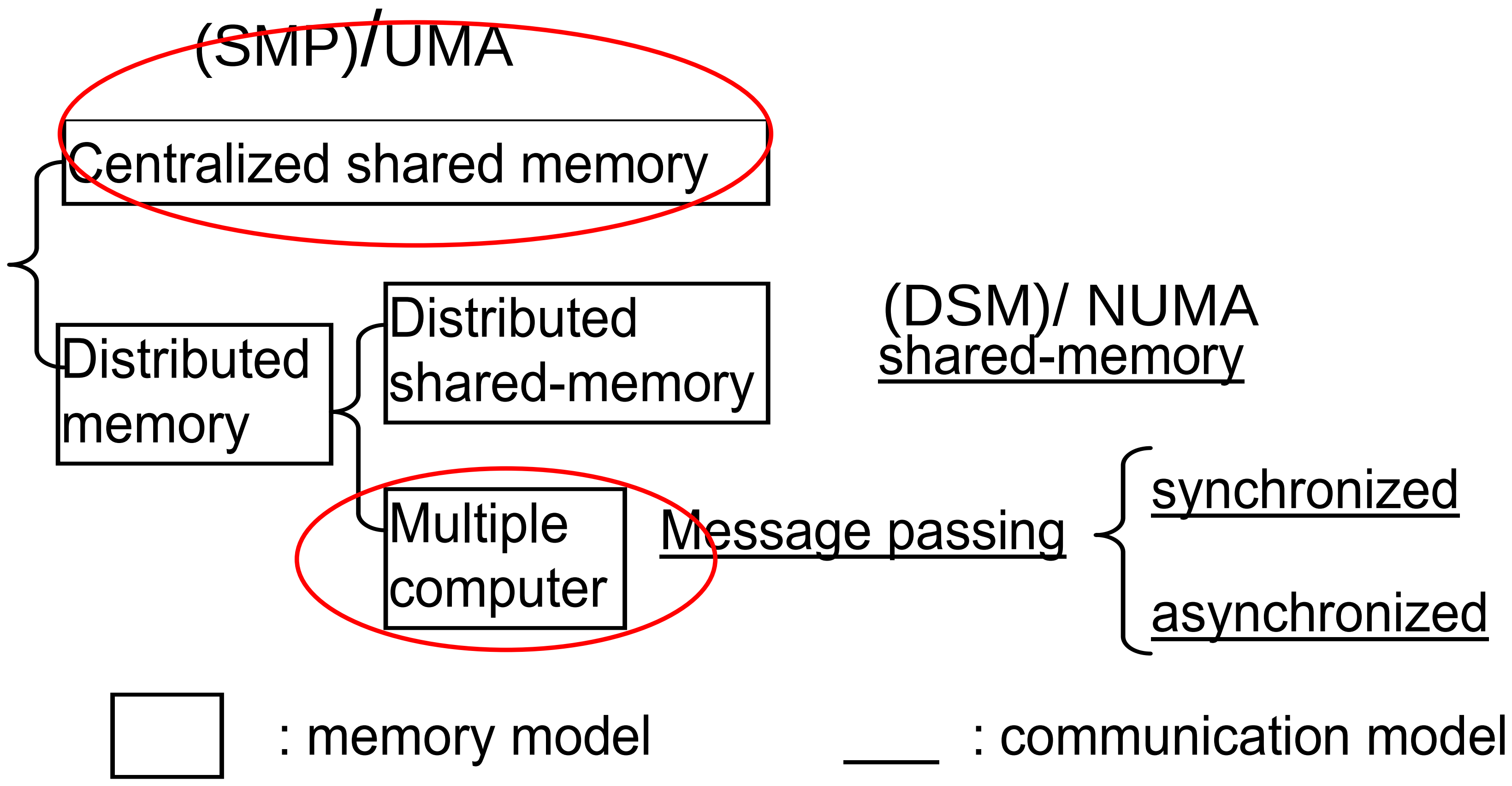
Program execute on MIMD



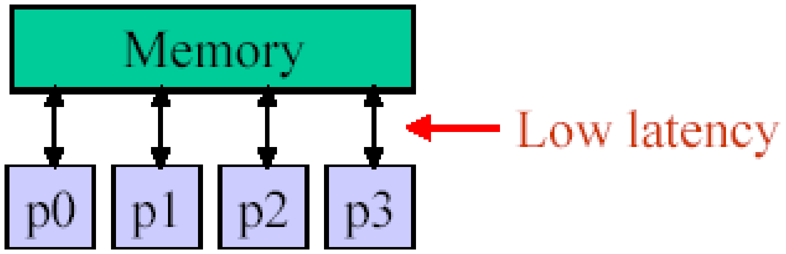
Catalogue the Parallel (MIMD) processors

- Center on organization of main memory
  - Shared vs. Distributed
- Appearance of memory to hardware
  - Q1: Memory access latency uniform?
  - Shared : yes, doesn’t matter where data goes
  - Distributed: no, makes a big difference
- Appearance of memory to software
  - Q2: Can processors communicate directly via memory?
  - Shared (shared memory): yes, communicate via load/store
  - Distributed (message passing): no, communicate via messages
- Dimensions are orthogonal
  - e.g. DSM: (physically) distributed, (logically) shared memory

Dimensions are orthogonal

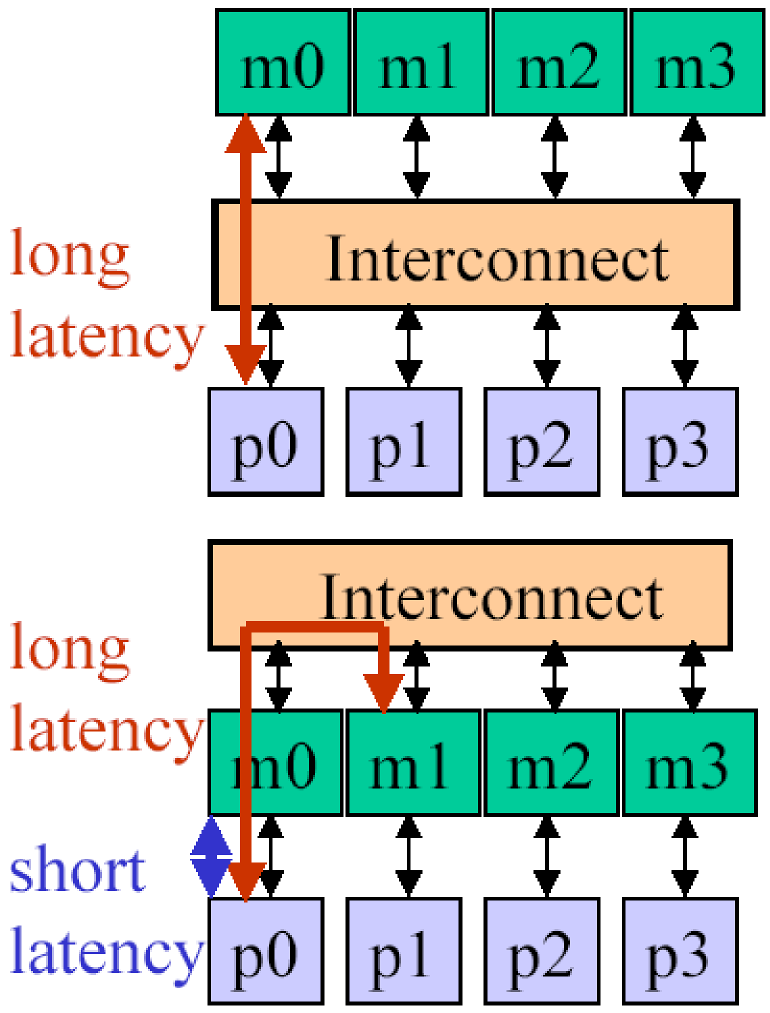


UMA vs. NUMA: Why it matters



- Ideal model:
  - Perfect (single-cycle) memory latency
  - Perfect (infinite) memory bandwidth
- Real systems:
  - Latencies are long and grow with system size
  - Bandwidth is limited
  - Add memory banks, interconnect to hook up (latency goes up)

UMA vs. NUMA



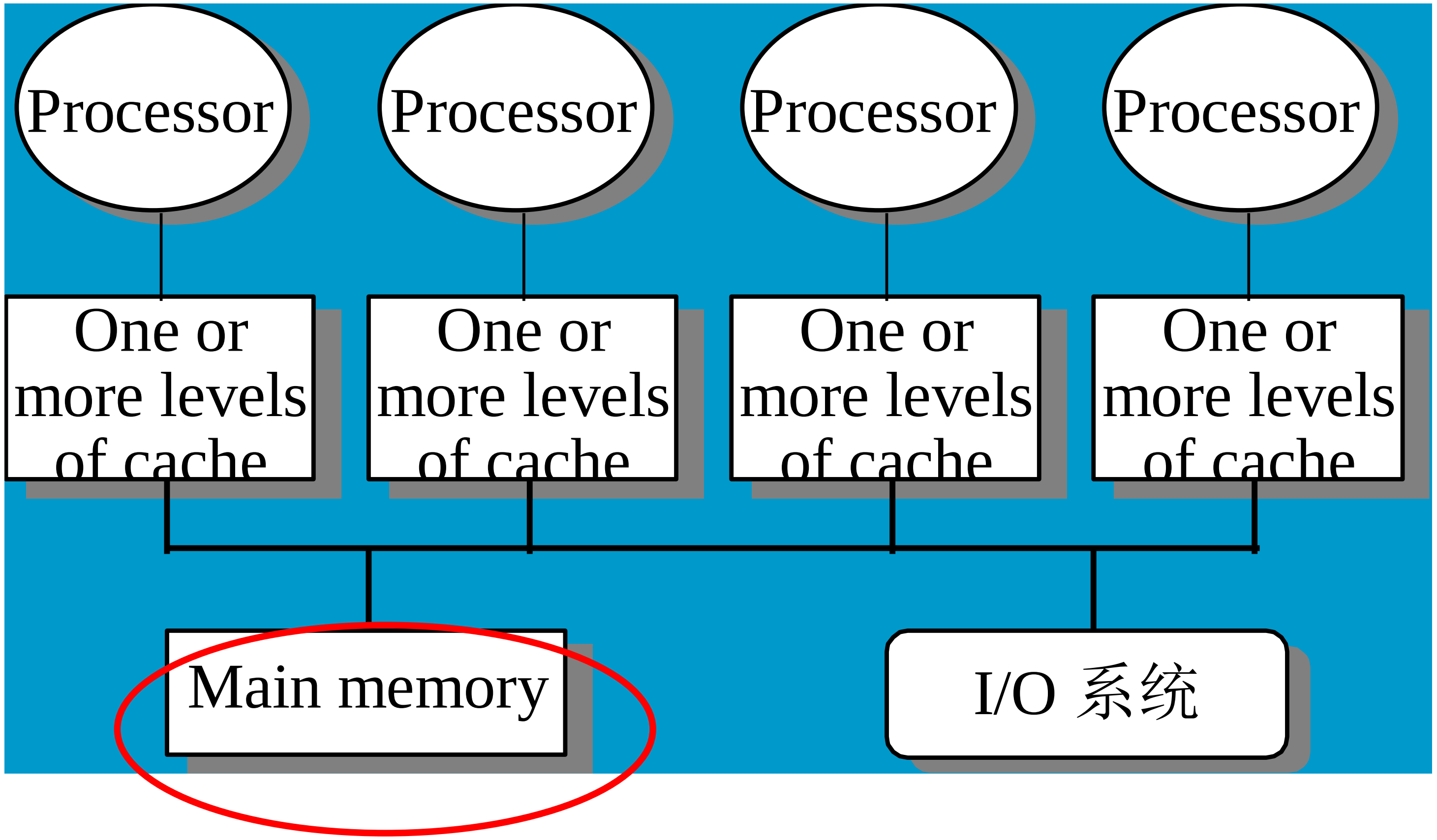
- UMA: uniform memory access
  - From p0 same latency to m0-m3
  - Data placement doesn’t matter
  - Latency worse as system scales
  - Interconnect contention restricts bandwidth
  - Small multiprocessors only
- NUMA: non-uniform memory access
  - From p0 faster to m0 than m1-m3
  - Low latency to local memory helps performance
  - Data placement important (software!)
  - Less contention => more scalable
  - Large multiprocessor systems

Major MIMD Styles

- Centralized shared memory ("Uniform Memory Access" time or "Shared Memory Processor")

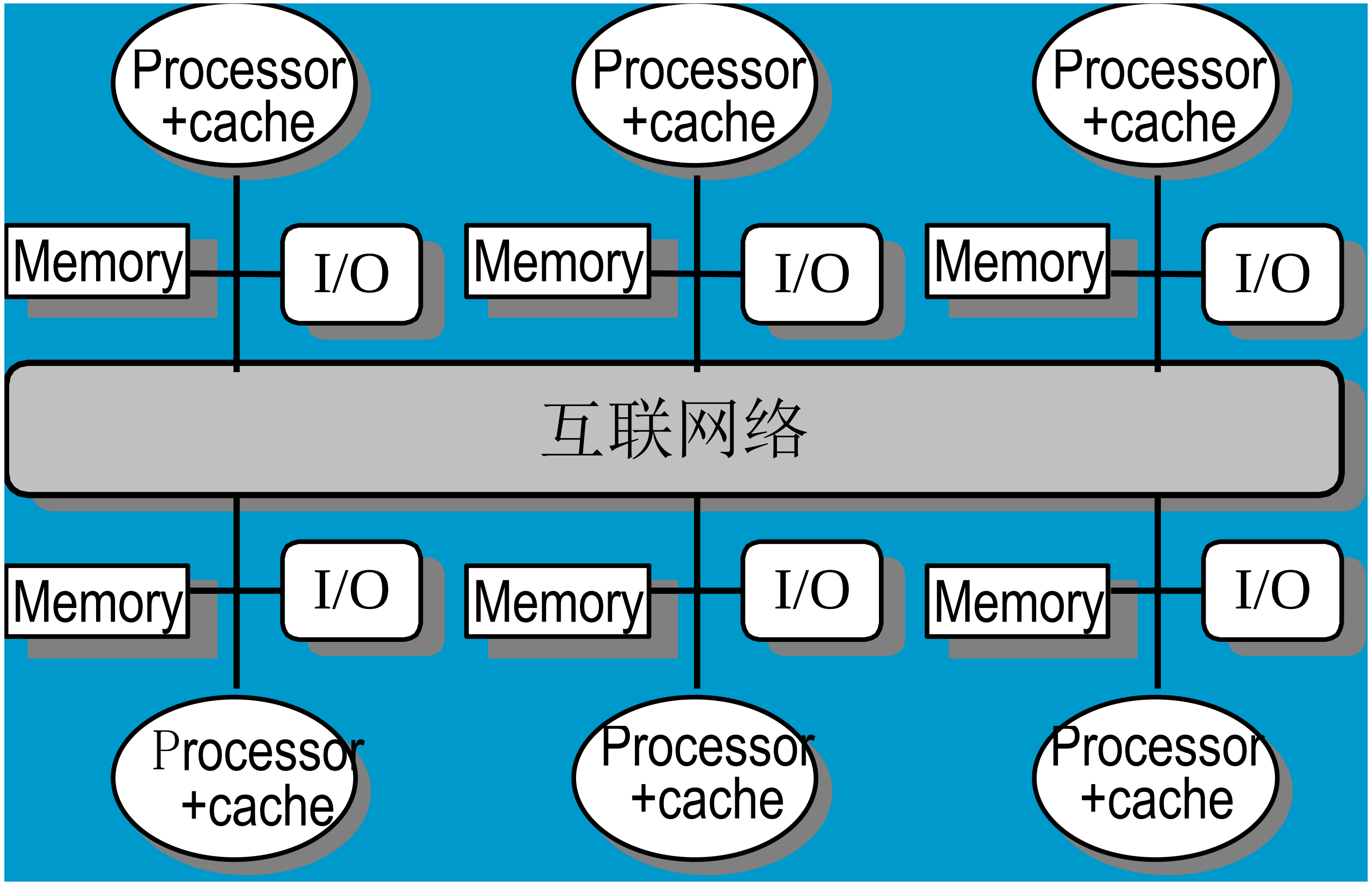
- Decentralized memory (memory module with CPU)
    - get more memory bandwidth, lower memory latency
    - Drawback: Longer communication latency
    - Drawback: Software model more complex

Structure of centralized shared-memory multiprocessor


Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Main memory
I/O 系统

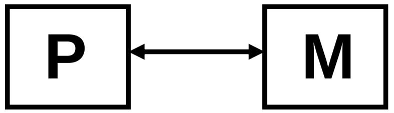
< 100 processor nodes in 2006, normal few dozen

Structure of distributed-memory multiprocessor

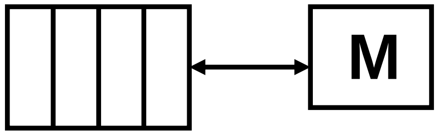


Comparison in graph

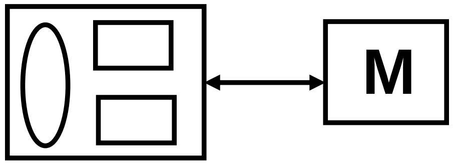
- Uniprocessor



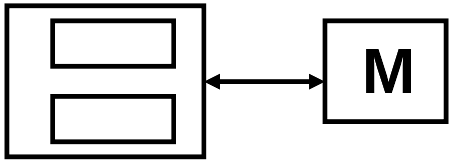
- Pipelined



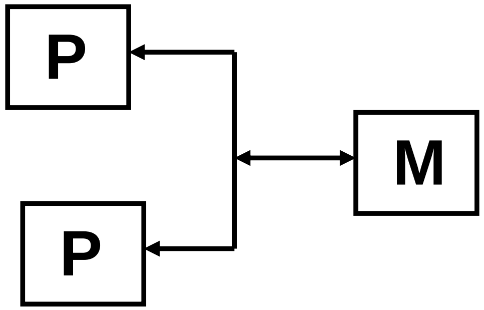
-
- Superscalar



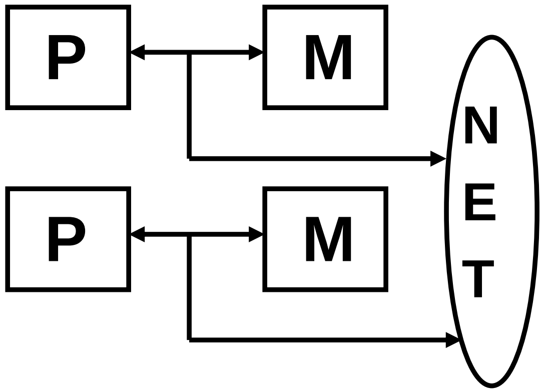
- VLIW



- SMP(Symmetric) or CSM(Centralized)



- Distributed



Distributed-memory multiprocessor(1)

- Distributed shared memory
  （DSM or scalable shared memory)
  - logical uniform address space but physical distributed memory, so any one of the processors can access any one of the memories.
  - Shared memory means sharing the address space, which is different from centralized shared memory.

Distributed-memory multiprocessor(2)

- multiple computers
  - Address space consists of multiple private (separate) address spaces。 A processor can NOT access the remote memory node directly.
  - Every node (processor-memory pair） is a independent computer.
  - NOW(Network of Workstation)is consisted of multiple node(PC or workstation) connected by LAN
  - PC cluster

Parallel Architecture

- Parallel Architecture extends traditional computer architecture with a communication architecture
  - abstractions (HW/SW interface)
  - organizational structure to realize abstraction efficiently

Parallel Framework

- Programming Model:
  - Multiprogramming : lots of jobs, no communication
  - Shared address space: communicate via memory
  - Message passing: send and recieve messages
  - Data Parallel: several agents operate on several data sets simultaneously and then exchange information globally and simultaneously (shared or message passing)
- Communication Abstraction:
  - Shared address space: e.g., load, store, atomic swap
  - Message passing: e.g., send, receive library calls

Shared Address Model-1

- Each processor can name every physical location in the machine
- Each process can name all data that shares with other processes
- Data transfer via load and store
- Data size: byte, word, ... or cache blocks
- Uses virtual memory to map virtual space to local or remote physical space
- Memory hierarchy model applies: now communication moves data to local processor cache (as load moves data from memory to cache)
  – Latency, Bandwidth, scalability when communicate?
- For distributed memory architecture, a layer (software or hardware) is generally added to allow transparent address mapping

Shared Address Model-2

- **Significant research has been conducted to make the translation transparent and scalable for many node**
- **Handling data consistency and protection is typically a challenge**
- **For multi-computer systems, address mapping has to be performed by software modules, typically added as part of the operating system**
- **Latency depends on the underlined hardware architecture (bus bandwidth, memory access time and support for address translation)**
- **Scalability is limited given that the communication model is so tightly coupled with process address space***

Message Passing Model-1

- Whole computers (CPU, memory, I/O devices) communicate as explicit I/O operations
  - Essentially NUMA but integrated at I/O devices vs. memory system
- Send specifies local buffer + receiving process on remote computer
- Receive specifies sending process on remote computer + local buffer to place data
  - Usually send includes process tag
    and receive has rule on tag: match 1, match any
  - Synch: when send completes, when buffer free, when request accepted, receive wait for send
- Send + receive => memory-memory copy, where each supplies local address, AND does pairwise sychronization!

Message Passing Model-2

- History of message passing:
  - Network topology important because could only send to immediate neighbor
  - Typically synchronous, blocking send & receive
  - Later DMA with non-blocking sends, DMA for receive into buffer until processor does receive, and then data is transferred to local memory
  - Later SW libraries to allow arbitrary communication
- Example:
  - IBM SP-2, RS6000 workstations in racks
  - Network Interface Card has Intel 960
  - 8X8 Crossbar switch as communication building block
  - 40 MByte/sec per link

Shared Memory vs. Message Passing

- Shared Memory (multiprocessors)
  - One shared address space
  - Processors use conventional load/stores to access shared data
  - Communication can be complex/dynamic
  - Simpler programming model (compatible with uniprocessors)
  - Hardware controlled caching is useful to reduce latency contention
  - Has drawbacks
    - Synchronization (discussed later)
    - More complex hardware needed

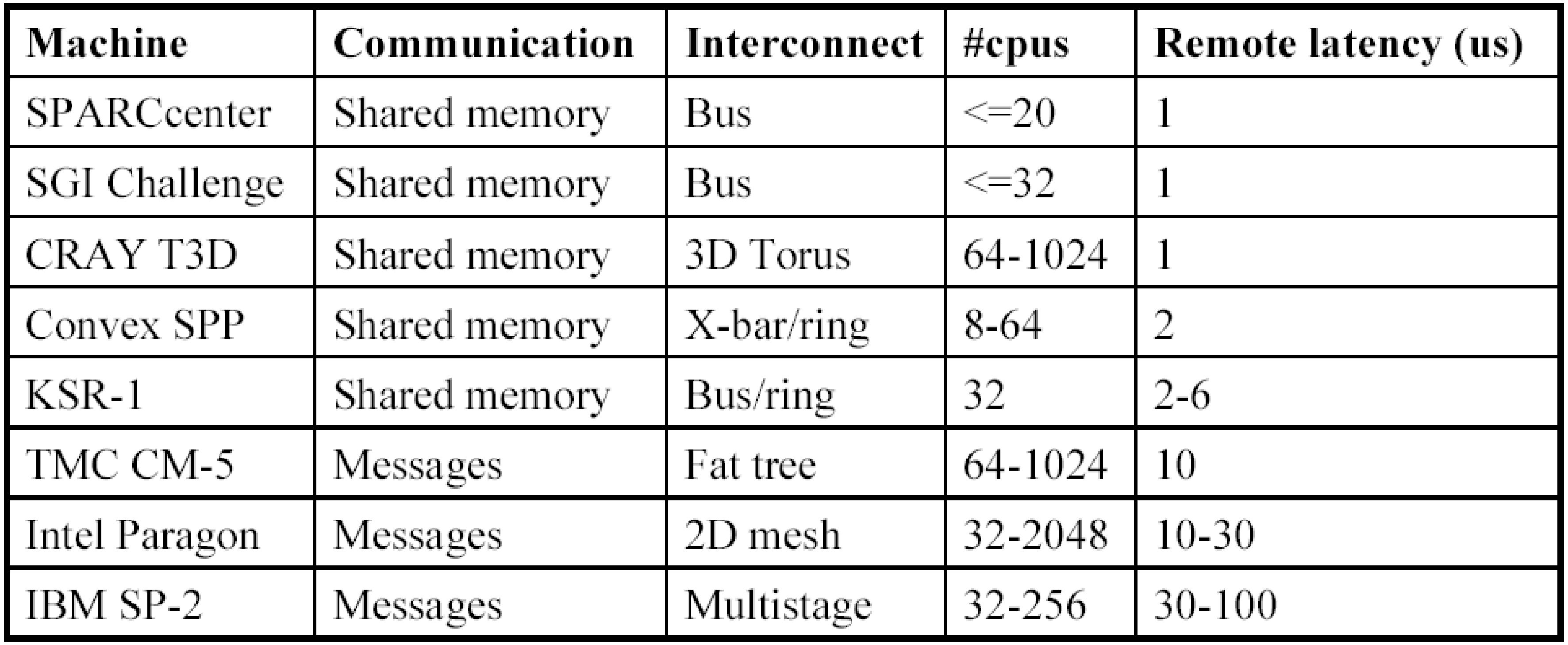
Shared Memory vs. Message Passing

- MIMD (appearance of memory to software)
- Message Passing (multicomputers)
  - Each processor has its own address space
  - Processors send and receive messages to and from each
  - other
  - Communication patterns explicit and precise
  - Explicit messaging forces programmer to optimize this
  - Used for scientific codes (explicit communication)
  - Message passing systems: PVM, MPI, OpenMP
  - Simple Hardware
  - Difficult programming Model

Communication Models

- Shared Memory
  - Processors communicate with shared address space
  - Easy on small-scale machines
  - Advantages:
    - Model of choice for uniprocessors, small-scale MPs
    - Ease of programming
    - Lower latency
    - Easier to use hardware controlled caching
- Message passing
  - Processors have private memories, communicate via messages
  - Advantages:
    - Less hardware, easier to design
    - Focuses attention on costly non-local operations
- Can support either SW model on either HW base

Parallel Systems (80s and 90s)



Machine	Communication	Interconnect	#cpus	Remote latency (us)
SPARCcenter	Shared memory	Bus	<=20	1
SGI Challenge	Shared memory	Bus	<=32	1
CRAY T3D	Shared memory	3D Torus	64-1024	1
Convex SPP	Shared memory	X-bar/ring	8-64	2
KSR-1	Shared memory	Bus/ring	32	2-6
TMC CM-5	Messages	Fat tree	64-1024	10
Intel Paragon	Messages	2D mesh	32-2048	10-30
IBM SP-2	Messages	Multistage	32-256	30-100

Multiprocessor Trends

- Shared Memory
  - Easier, more dynamic programming model
  - Can do more to optimize the hardware
- Small-to-medium size UMA systems (2-8 processors)
  - Processor + memory + switch on single board (4x pentium)
  - Single-chip multiprocessors (POWER4)
  - Commodity parts soon – glueless MP systems
- Larger NUMAs built from smaller UMAs
  - Use commodity small UMAs with commodity interconnects (ethernet, myrinet)
  - NUMA clusters

Fundamental Issues_1

- *Naming*: how to solve large problem fast
  - what data is shared
  - how it is addressed
  - what operations can access data
  - how processes refer to each other
- Choice of naming affects code produced by a compiler; via load where just remember address or keep track of processor number and local virtual address for message passing
- Choice of naming affects replication of data; via load in cache memory hierarchy or via SW replication and consistency

Fundamental Issues_2

- Global physical address space: any processor can generate, address and access it in a single operation
- Global virtual address space: if the address space of each process can be configured to contain all shared data of the parallel program
- memory can be anywhere: virtual address translation handles it
- Segmented shared address space: locations are named <process number, address>
- uniformly for all processes of the parallel program

Fundamntal Issues_3

- *Synchronization*:
- To cooperate, processes must coordinate
- Message passing is implicit coordination with transmission or arrival of data
- Shared address => additional operations to explicitly coordinate: e.g., write a flag, awaken a thread, interrupt a processor

Fundamntal Issues_4
Latency and Bandwidth

1. Bandwidth
   - Need high bandwidth in communication
   - Match limits in network, memory, and processor
   - Challenge is link speed of network interface vs. bisection bandwidth of network
2. Latency
   - Affects performance, since processor may have to wait
   - Affects ease of programming, since requires more thought to overlap communication and computation
   - Overhead to communicate is a problem in many machines
3. Latency Hiding
   - How can a mechanism help hide latency?
   - Increases programming system burdern
   - Examples: overlap message send with computation, prefetch data, switch to other tasks

Challenge:
limited program parallism

$$\text{Speedup}_{\text{Overall}} = \frac{1}{\left((1 - \text{Fraction}_{\text{enhanced}}) + \dfrac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}\right)}$$

- Example: Achieve speedup of 80 x using 100 processors
  - 80 = 1/[Fracparallel/100+1-Fracparallel]
  - Frac parallel = 0.9975

    ==> only 0.25% of the work can be serial!

Can parallized part be imporved to 100 times considering the overheads ?

Challenge ：
long communication latency

- Given： (P203)
  - 32-processor machine, with each processor cycle time=0.5ns (2GHz);
  - remote reference time= 200ns;
  - all references except those involving communication hit in local memory;
  - base CPI=0.5 (all reference hit in the cache);
  - Processors are stalled on a remote request.
  - 0.2% of the instructions involve a remote reference.

The multiprocessor with all local references is 1.3/0.5 = 2.6 times faster.

- Answer:
  - **CPI = Base CPI + Remote request rate × Remote request cost**
    =0.5 + 0.2% × Remote request cost

    The remote request cost is:

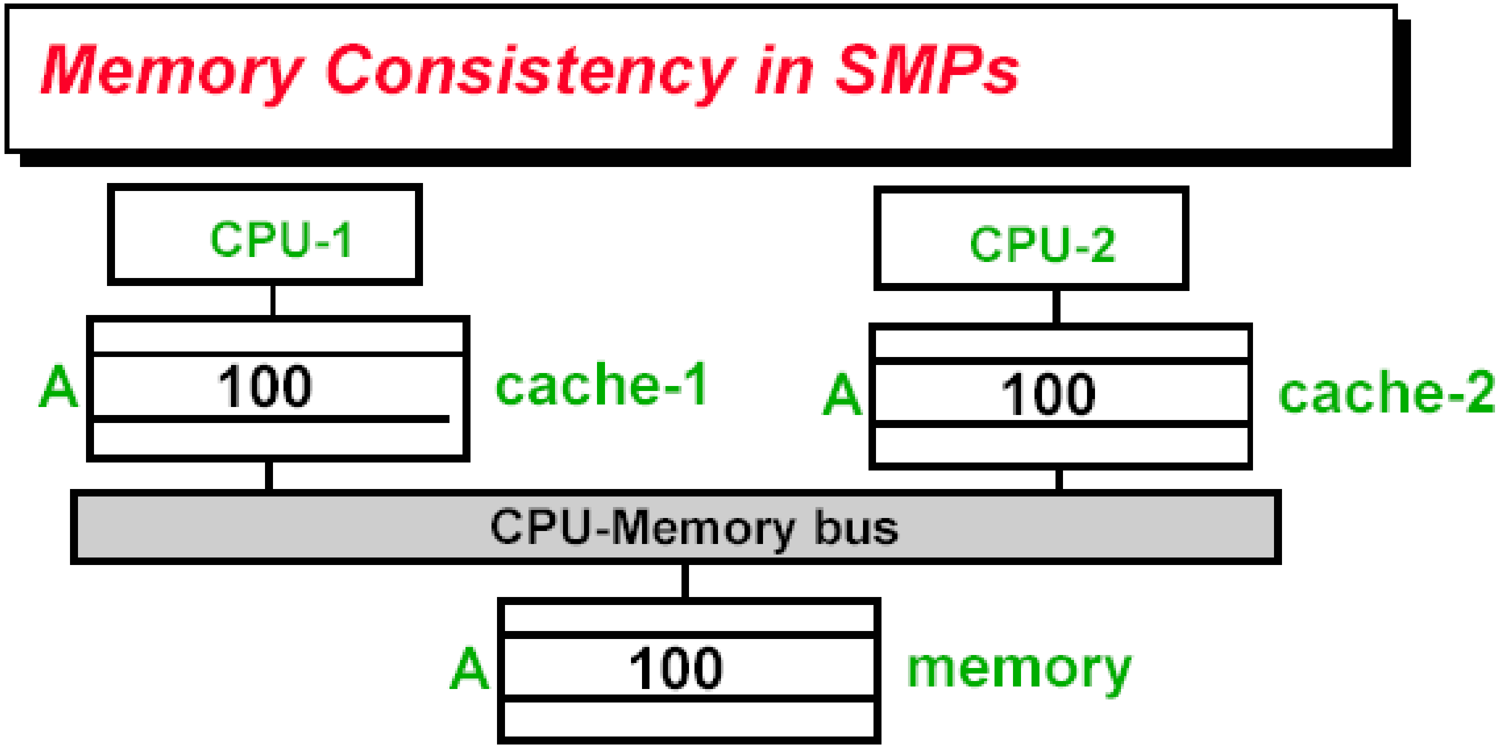
$$\frac{\text{Remote access cost}}{\text{Cycle time}} = \frac{200\text{ns}}{0.5\text{ns}} = 400\text{cycles}$$

CPI=0.5 + 0.8 = 1.3

What can we do ?

- Limited program parallelism
  - New algorithm
- Long communication latency
  - HW: **caching shared data to lower the remote access frequency**
    - **Problem:** cache coherence,
      cache consistence
  - SW**:**
    - **restructuring the data to make more accesses local**
    - Synchronization
    - **latency hiding techniques**

Cache Coherence in Multiprocessor



Suppose CPU-1 updates A to 200.

write-back: memory and cache-2 have stale values

write-through: cache-2 has a stale value

HW Coherence Protocols

- Snooping Solution (Snoopy Bus):
  - Send all requests for data to all processors
  - Processors snoop to see if they have a copy and respond accordingly
  - Requires broadcast, since caching information is at processors
  - Works well with bus (natural broadcast medium)
  - Dominates for small scale machines (most of the market)
- Directory-Based Schemes (discuss later)
  - Keep track of what is being shared in 1 centralized place (logically)
  - Distributed memory => distributed directory for scalability (avoids bottlenecks)
  - Send point-to-point requests to processors via network
  - Scales better than Snooping
  - Actually existed BEFORE Snooping-based schemes

Basic Snoopy Protocols

- Write Invalidate Protocol:
  - Multiple readers, single writer
  - Write to shared data: an invalidate is sent to all caches which snoop and *invalidate* any copies
  - Read Miss:
    - Write-through: memory is always up-to-date
    - Write-back: snoop in caches to find most recent copy
- Write Broadcast Protocol (typically write through):
  - Write to shared data: broadcast on bus, processors snoop, and *update* any copies
  - Read miss: memory is always up-to-date
- Write serialization: bus serializes requests!
  - Bus is single point of arbitration

EX: write back Cache, write invalidate

- Mechanics
  - Broadcast address of cache line to invalidate
  - All processor snoop, then invalidate if in local cache
  - policy can be used to service cache misses in write-back caches

Processor Activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of Memory Location X
				0
CPU A Reads X	Cache miss for X	0		0
CPU B Reads X	Cache miss for X	0	0	0
CPU A writes A 1 to X	Invalidation for X	1		0
CPU B Reads X	Cache miss for X	1	1	1

Ex: Write back Cache, update(Broadcast)

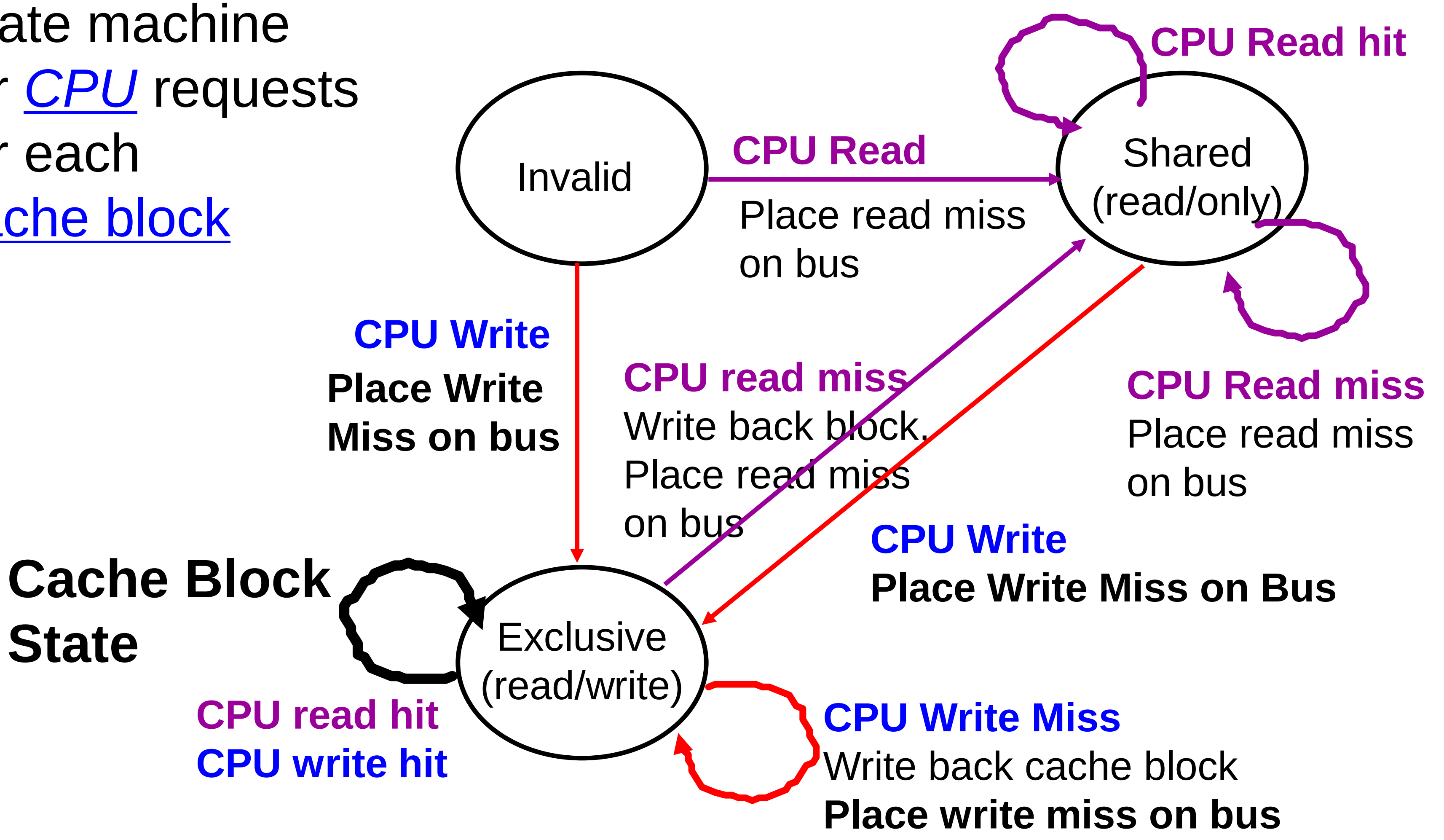
Processor Activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of Memory Location X
				0
CPU A Reads X	Cache miss for X	0		0
CPU B Reads X	Cache miss for X	0	0	0
CPU A writes A 1 to X	Write broadcast Of X	1	1	1
CPU B Reads X		1	1	1

Simple write-invalidate protocol

- Three states
  - Invalid, Shared, exclusive
- Events
  - CPU-R, CPU-W
  - BUS-R, BUS-W

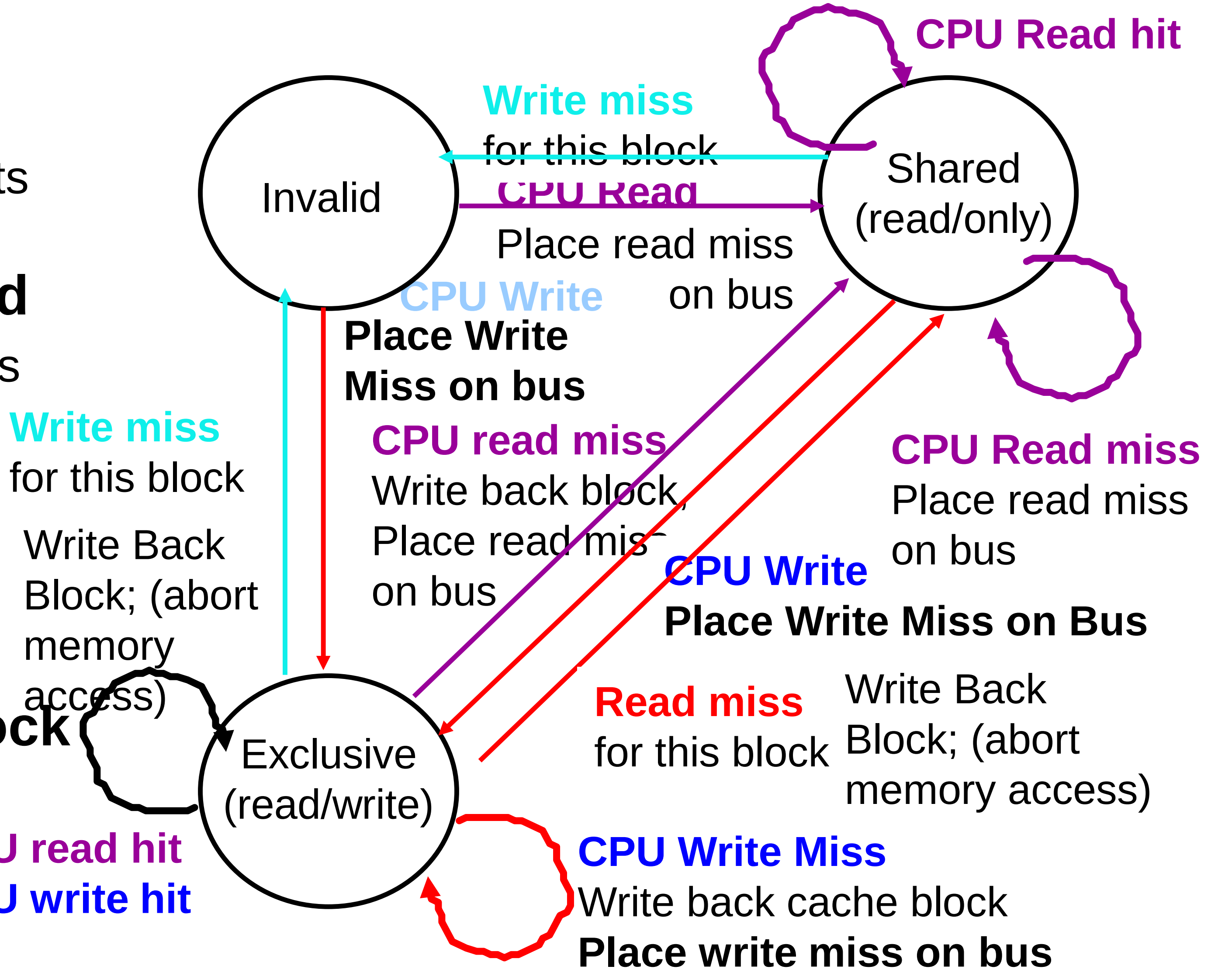
Snoopy-Cache State Machine-I

- State machine for CPU requests for each cache block



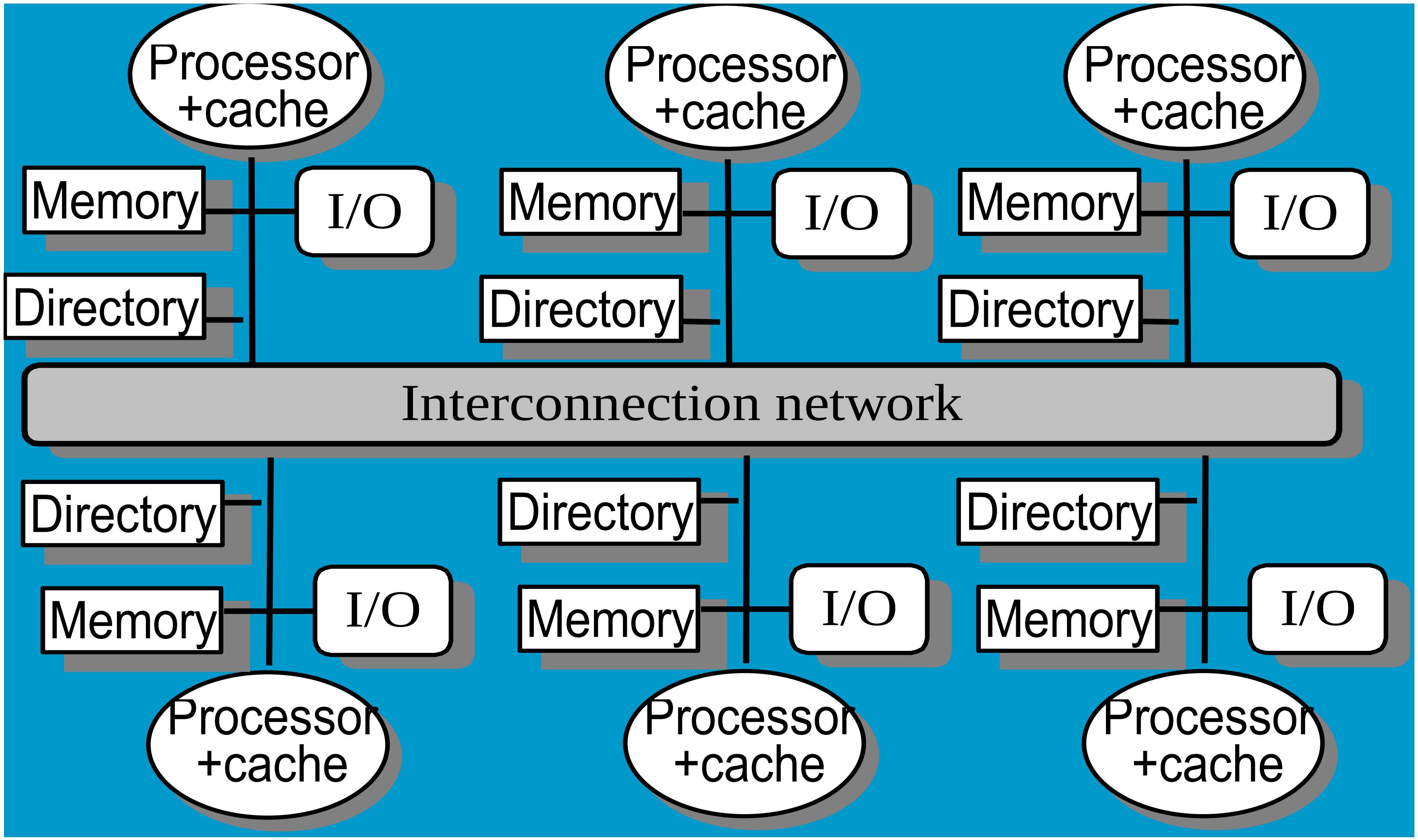
Snoopy-Cache State Machine-III

- State machine for CPU requests for each cache block **and** for bus requests for each cache block



Cache Block State

Distributed Directory MPs



Directory protocol

- Directory: track state of every block in memory, and change the state of block in cache according to directory.
- Information in directory
  - Status of Every block: shared/uncached/exclusive
  - Which processors have copies of the block: bit vector
  - Whether the block is dirty or clean
- Directory protocol can be implemented with a distributed memory
- Directory protocol can be applied to a centralized memory organized into banks

Directory protocol implementation

- Block status
  - Shared: ≥ 1 processors have data, memory up-to-date
  - Uncached (no processor hasit; not valid in any cache)
  - Exclusive: 1 processor (owner) has data; memory out-of-date
- Directory size = f (entry number * entry size)
  - Each memory block has an entry in directory / only keep the entries for cached blocks
  - Every processor has one bit / Limited processor bits in bit vector
- Directory can be distributed along with the memory to avoid becoming the bottleneck
- Assumptions to Keep it simple:
  - Writes to non-exclusive data => write miss
  - Processor blocks until access completes
  - Assume messages received and acted upon in order as sent

Directory Protocol

- No bus and don't want to broadcast:
  - Interconnect means no longer single arbitration point
  - all messages have explicit responses
- Terms: typically 3 processors involved
  - Local node where a request originates
  - Home node where the memory location of an address resides
  - Remote node has a copy of a cache block, whether exclusive or shared
- Example messages on next slide:
  P = processor number, A = address

Message type	***Source***	***Destination***	***Msg Content***
Read miss	Local cache	Home directory	P, A
– *Processor P reads data at address A; make P a read sharer and arrange to send data back*			
Write miss	Local cache	Home directory	P, A
– *Processor P has a write miss at address A; make P the exclusive owner and arrange to send data back*			
Invalidate	Local cache	Home directory	A
– *Request to send invalidates to all remote caches that are caching the block at address A*			
Invalidate	Home directory	**Remote caches**	A
– *Invalidate a shared copy at address A.*			
Fetch	Home directory	**Remote cache**	A
– *Fetch the block at address A and send it to its home directory*			
Fetch/Invalidate	Home directory	**Remote cache**	A
– *Fetch the block at address A and send it to its home directory; invalidate the block in the cache*			
Data value reply	Home directory	Local cache	Data
– *Return a data value from the home memory (read miss response)*			
Data write-back	Remote cache	Home directory	A, Data
– *Write-back a data value for address A (invalidate response)*			

- The End.