SFU

CMPT 450/750: Computer Architecture
Fall 2024

Cache Coherence

Alaa Alameldeen & Arrvindh Shriraman

Shared Memory Multiprocessors

- **All processors can access all memory**
- **Processors share memory resources, but can operate independently**
- **One processor's memory changes are seen by all other processors**
- **Easier to program**
  - Communication through shared memory
  - Synchronization through locks stored in shared memory
- **Need cache coherence in hardware**
- **Need interconnection network between all processors and all memory**
- **Two Types:**
  - Uniform Memory Architectures (UMA): e.g., Symmetric Multiprocessors
  - Non-Uniform Memory Architectures (NUMA): Access & latency to memory is different

Interconnection Networks

- **In a shared memory MP, we need to connect different processors and memory modules**
- **Types of interconnect:**
  - Shared bus
  - Crossbar: Fully connected
  - Ring
  - Mesh
  - 2-D Torus
  - Hypercube
- **Number of hops vs. number of links: Compare N processors and M memory modules**

Shared Memory Multiprocessors: Memory Hierarchy

- **Problem: sharing memory means more than one processor can send requests to memory**
  - ➢High memory bandwidth required
- **To avoid sending lots of memory requests, processors use caches to:**
  - ➢Filter out many memory requests
  - ➢Reduce average memory latency
  - ➢Reduce memory bandwidth requirements
- **Typically more than one level of caches is used**
  - ➢L1 caches: Usually Split I & D caches, small and fast
  - ➢L2 caches: Usually on die, composed of SRAM cells
  - ➢L3 caches: On-die or off-die, SRAM or eDRAM cells

Cache Coherence

- **Problem: Using caches means multiple copies of the same memory location may exist**
  - ➢Updates to the same location may lead to bugs
- **Example:**

  Processor 1 reads A

  Processor 2 reads A

  Processor 1 writes to A

Now, processor 2's cache contains stale data

- **Cache coherence need to be implemented in hardware using a cache coherence protocol**

Conditions for Cache Coherence

- **Program Order.** **A read by processor P to location A that follows a write by P to A, with no writes to A by another processor in between, should always return the value of A written by P**
- **Coherent View of Memory.** **A read by processor P1 to location A that follows a write by another processor P2 to location A should return the written value by P2 if:**
  - The read and write are sufficiently separated in time
  - No other writes to A by another processor occur between the read and the write
- **Write Serialization.** **Writes to the same location are serialized: Two writes to the same location by any two processors are seen in the same order by all processors**

Cache Coherence Protocol Classification

- **Cache coherence defines behavior of reads and writes to the same memory location**
- **Compared to: Memory consistency models define the behavior of reads and writes with respect to accesses to other memory locations**
- **Two main types of cache coherence protocols:**
  - Snooping
    - Caches keep track of the sharing status of all blocks
    - No centralized state is kept
    - Cache controllers snoop shared interconnect (typically shared bus) to track when a requested block exists in the cache
  - Directory
    - Sharing status of any block in memory is kept in one location

Cache Coherence Example

ARCH Figure 5.3

Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

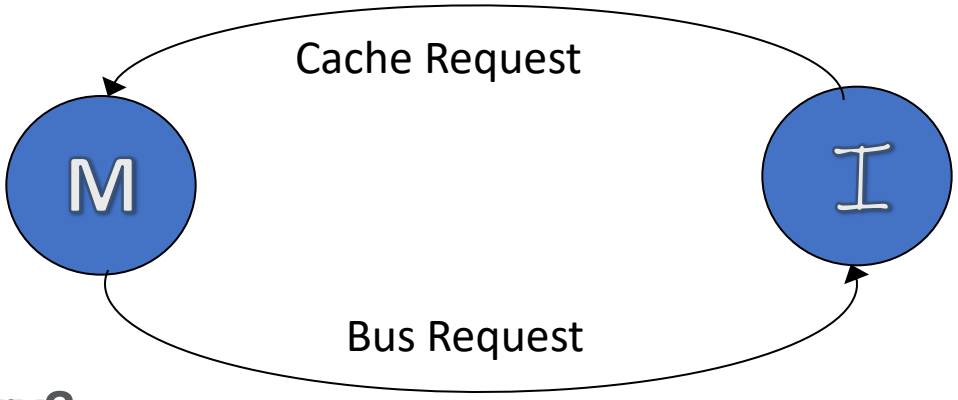
- **Without cache coherence, Processor B's cache has stale data**
- **How to enforce coherence?**
  - B's cache line is invalidated (write-invalidate protocols)
  - B's cache line is updated by A's write (write-update protocols)

Invalidate vs. Update Protocols

- **Write-invalidate protocols**
  - Guarantees only one writer has a valid copy of a block
  - Read requests issue a “Get Shared” (GetS) request for the block to other caches/memory
  - When a processor wants to write to a cache block, it issues a “Get Exclusive” (GetX) request to other processors, forcing them to invalidate any copies of the block
  - Subsequent writes from the same processor are done locally in the cache
- **Write-update protocols**
  - When a processor writes to a block, it sends data to all other processors with valid copies
- **Pros and cons?**

Very Simple Write-Invalidate Coherence Protocol

- **MI protocol**
  - Two states: M (Modified) and I (Invalid)
  - Only one cache contains a copy of a certain memory location
  - When another cache requests a block, the cache currently containing the block invalidates it
  - Protocol limits sharing and degrades performance



- **Why is sharing necessary?**
- **Optimization: MSI protocol allows read sharing**

Example 1: Finite-Buffer Producer/Consumer

Producer:

```
If (count <= N) {
        #mutex begin
            buffer [in] = item;
            in = (in +1) % N;
            count ++;
         #mutex end
}
```

Consumer:

```
If (count > 0) {
        #mutex begin
            item = buffer [out];
            out = (out +1) % N;
            count --;
         #mutex end
}
```

- Producer generates an item unless buffer is full
- Consumer removes an item unless buffer is empty
- **Read-write sharing** for buffer size and buffer elements

Example 2: Solving a Linear System of Equations

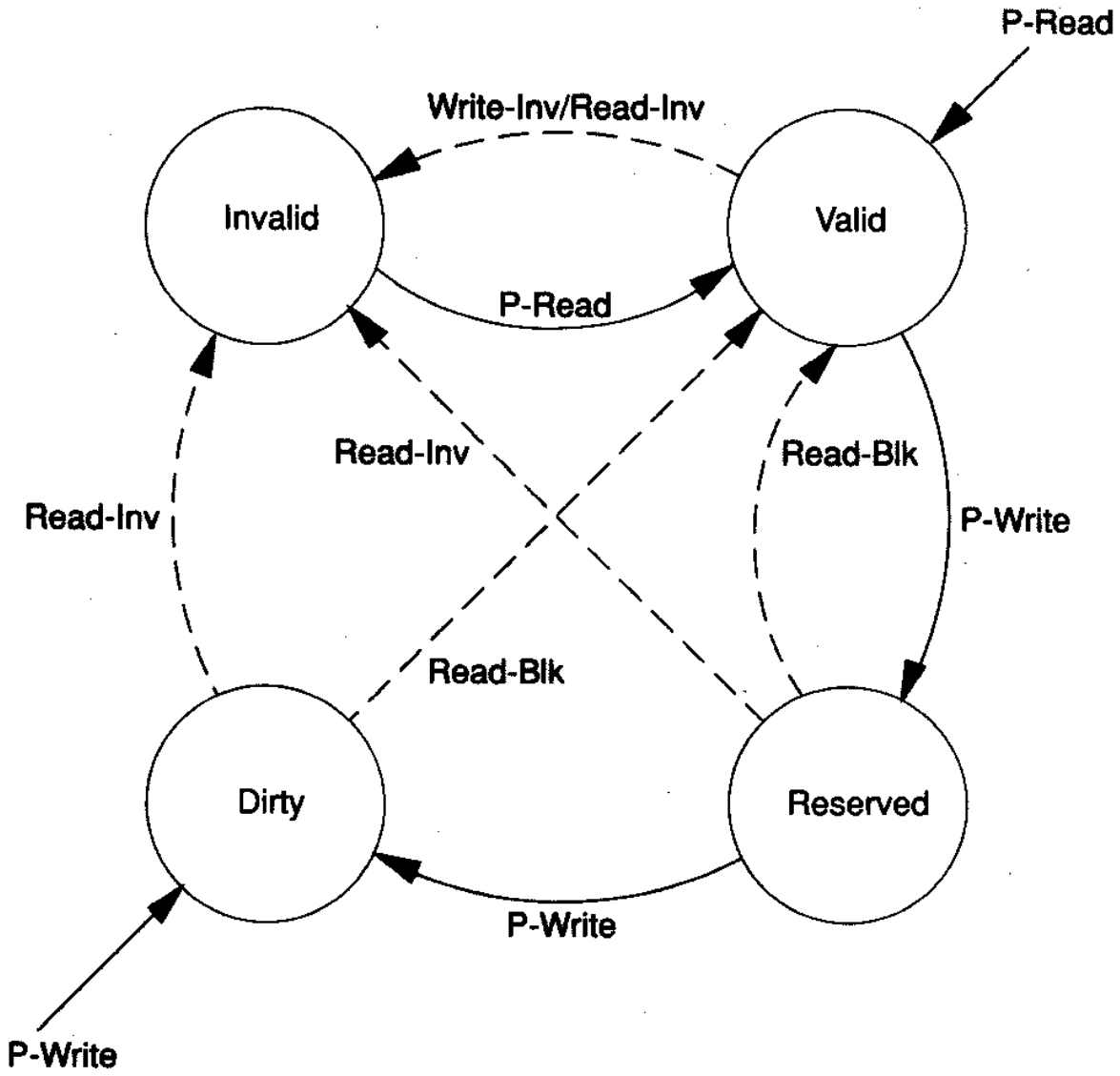
- Solving for x in system A x = b

```
diff = 1000;
while(diff > 0.01) {
  parallel for(j=0; j < N; j++) {
        xtemp[j] = b[j];
        for (k=0; k < N; k++)
              xtemp[j] += A[j][k]*x[k];
  }
  # Barrier Synchronization
  // compute diff = max(abs(x[i]-xtemp[i]));
  parallel for (j =0; j < N; j++)
        x[j] = xtemp[j];
  # Barrier Synchronization
}
```

- Vector x is computed every cycle, is **shared "read-write"**
- Both array A and vector b are **shared "read-only"**, can be safely replicated

Write-Once Invalidate Protocol

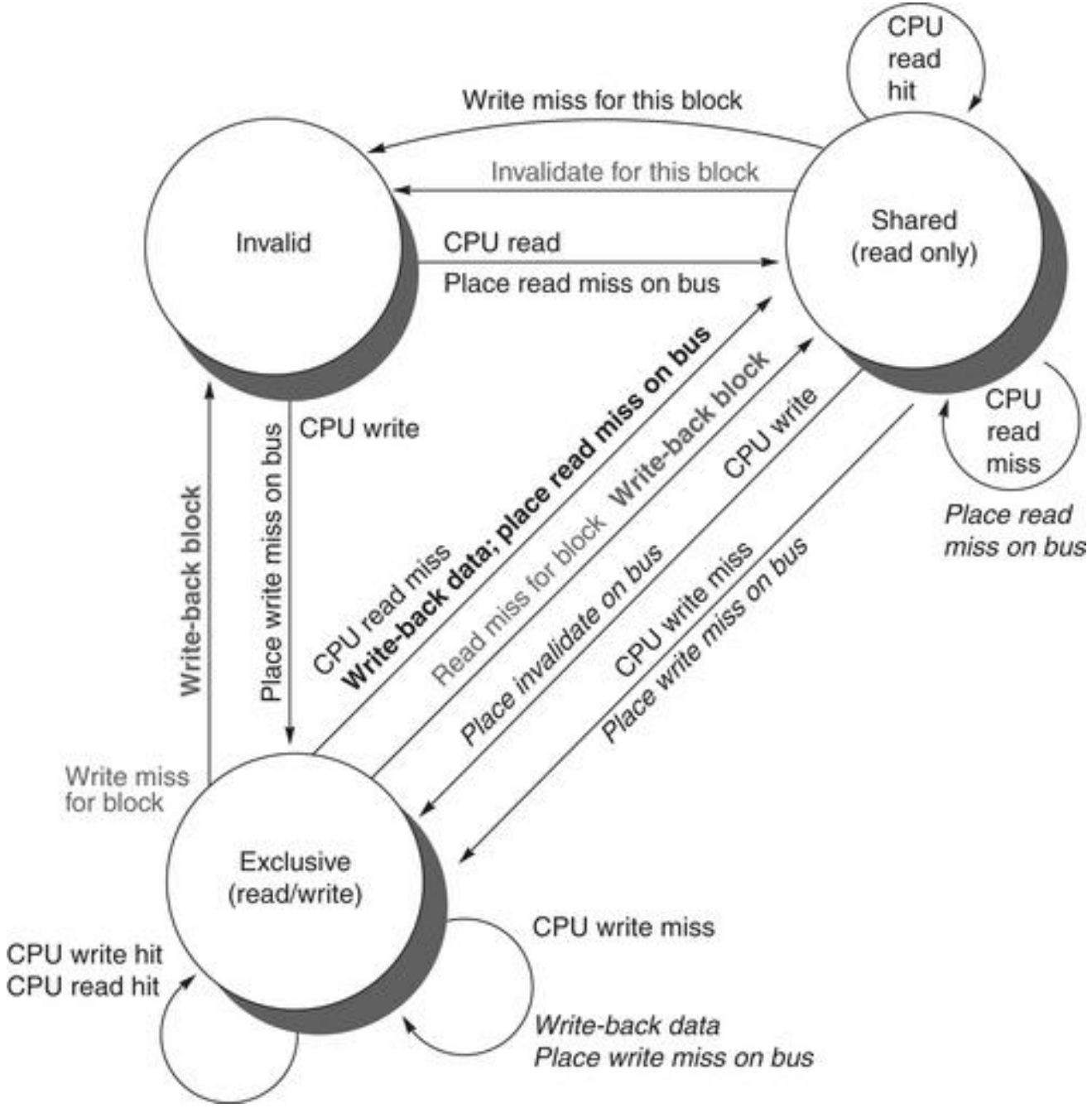
- **States**
  - Invalid
  - Valid: Copy is consistent with memory
  - Reserved: Data has been written **exactly once**, and copy is consistent with memory (the only other copy)
  - Dirty: Data modified more than once, only valid copy
- **Copy-back memory update policy: Block is written back to memory when replaced if the block is dirty**
- **Events:**
  - Processor read (P-Read) and Processor write (P-Write)
  - Memory read block (Read-Blk) and write block (Write-Blk)
  - Write-Inv: Invalidate all other copies of block
  - Read-Inv: Read a block and invalidate all other copies



Stenstrom 1990, Figure 6

MSI Protocol

- Allows read sharing
- Better performance when tasks on different CPUs have shared read-only data (e.g., System of Linear Equations)
- Exclusive State also called "Modified" State, so the protocol is referred to as "MSI"



ARCH Figure 5.7

Coherence Misses

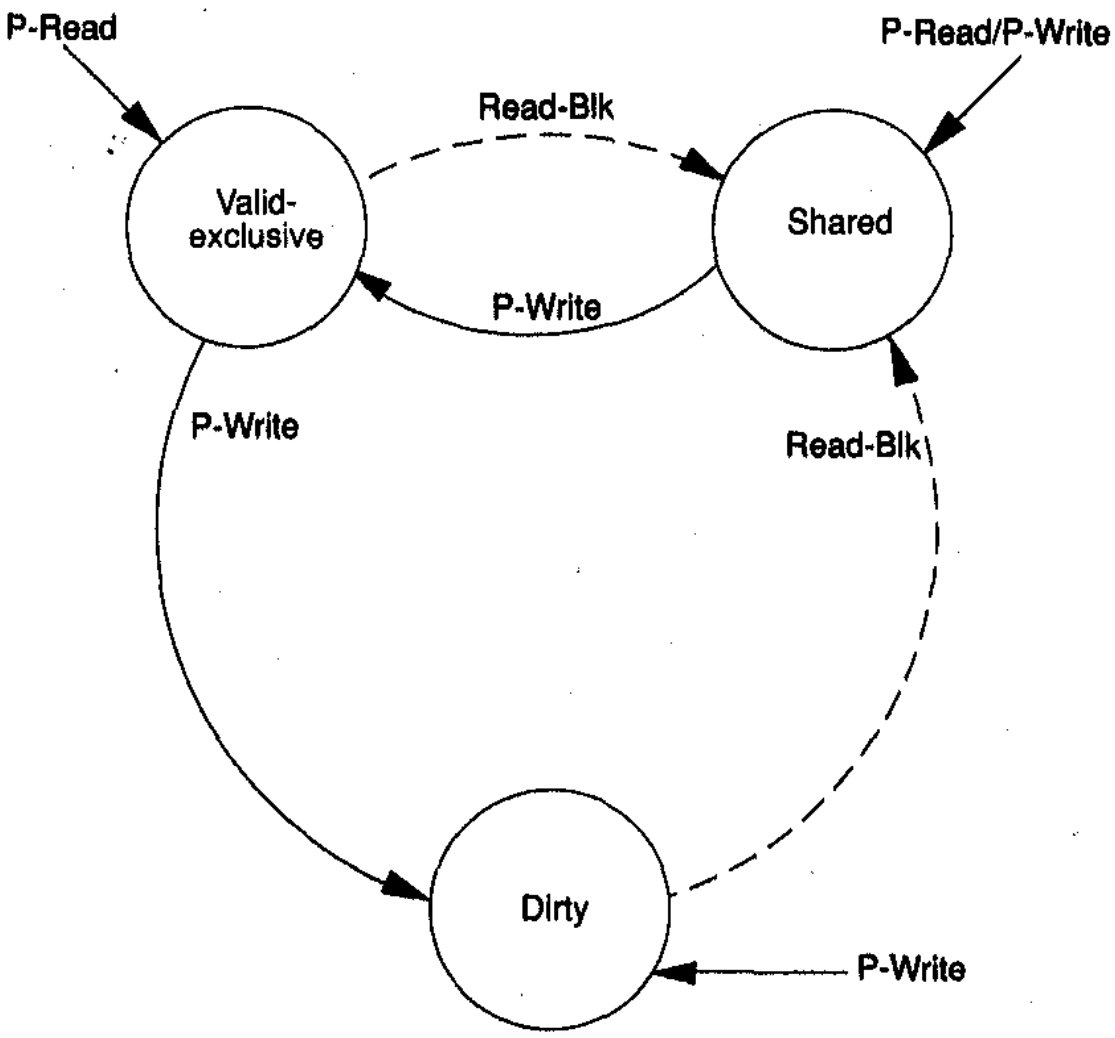
- **Recall the 3 C classification of cache misses: Compulsory, Capacity, Conflict**
- **Another type of misses: Coherence Misses**
  - Misses caused by coherence actions
  - That is, misses that will not occur in a single-processor system
- **In the MSI protocol, coherence misses are caused by:**
  - Read or Write requests to an invalid block which was invalidated by another processor's write request
  - Write requests to a shared block which was downgraded to S by another processor's read request
- **4 C classification of cache misses: Compulsory, Capacity, Conflict, Coherence**

Extensions for MSI Protocol

- **MESI**
  - Same as MSI but adds an E “Clean-Exclusive” state
  - E state is for read-only blocks that aren’t modified compared to memory
  - Read request by another CPU to an E-block: State changes to S
  - Write request by same CPU to an E-block: State silently upgraded to M
  - Advantage: Saving coherence bandwidth
    - Silent upgrade to M with no coherence requests
    - Evicting a block in E does not require writing data to memory
- **MOESI**
  - Same as MESI, but adds an O “Own” state
  - O state is for blocks that are different from memory and owned by cache. When another CPU requests a block in M, the cache sends it to the other CPU and changes state to O
  - Cache with O block is responsible for sending data to other read requesters, and updating memory when the block is evicted
  - Advantages:
    - Less memory traffic: Memory only updated on an eviction of the O-block, not on a read request for an M-block
    - Less coherence traffic: Only the cache with the O-block is responsible for sending shared copy on a read request

Firefly Write-Update Protocol

- **States**
  - Valid-exclusive: Only copy, consistent with memory copy
  - Shared: One of many valid copies
  - Dirty: Only valid copy, memory is inconsistent
- **Protocol uses copy-back update policy for private blocks**
- **Protocol uses write-through for shared blocks**
- **Used in the Firefly multiprocessor workstation from DEC**
- **Another update protocol: Dragon protocol proposed for the Dragon MP workstation from Xerox**
  - Avoids updating memory until a block is replaced



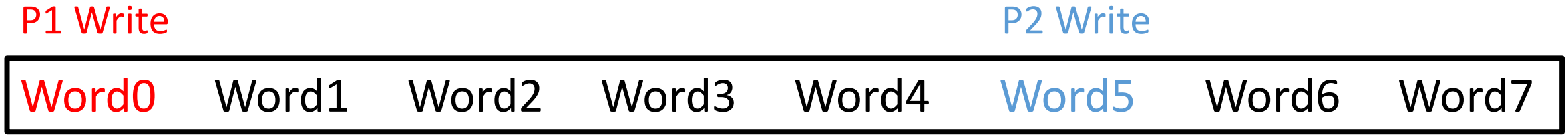
Stenstrom 1990, Figure 7

Implementation Issues for Snoopy Coherence Protocols

- **Lower complexity compared to directory protocols**
- **Hardware Components:**
  - Cache Controller: A finite state machine that implements coherence protocol state transition diagram
  - Cache Directory: Stores state for each block
  - Bus Controller: Implements bus snooping. Monitors every shared bus operation and takes action if needed (if the block is cached)
- **Implementation Issues**
  - Contention for cache directory between local and remote (bus) requests
  - Impact of block size (next slide)
  - Write-through (WT) vs. write-back (WB) caches
    - WT caches support update protocols while WB caches are more suitable for invalidate protocols
    - Most real systems caches are WB caches
  - Write-allocate vs. write no-allocate policies

False Sharing

- Occurs when non-shared data are co-located in the same cache line
- Example: Processor 1 writes to Word0 of Cache Block A, Processor 2 writes to Word 5 of Cache Block A



- Neither Word0 nor Word5 is shared, but the cache line needs to be in “M” in one cache and “I” in another (write-invalidate protocols)
- False sharing leads to increased coherence traffic
- False sharing increases with larger cache lines

Software Coherence Protocols

- **Compiler limits which blocks can be cached**
- **Types of data accesses**
  1. Shared read-only
  2. One writer, multiple readers
  3. One process read/write
  4. Shared read-write
- **Trivial solution: All shared read-write blocks are marked as uncacheable (types 2 and 4 above)**
- **Optimization: some shared read-write variables can be used by one processor for a long time, so may be cached**
- **Disadvantages vs. hardware protocols?**

Atomic Coherence Transactions

- **Previous discussion assumes that a coherence request will hold the shared bus until data comes back**
  - Example: A GetS “GetShared” request holds the shared bus from the time when the GetS request is sent till the requested data is received
  - During the time between the request and the response, no other processors can send any requests on the shared bus
  - This implies that all coherence requests are ***blocking requests, limiting MLP***
- **Atomic transactions cause significant execution delays especially for long-latency memory accesses**
  - Data could be in none of the caches and need to be read from memory. During this time, no other processors can send out requests
- **Disadvantage: Atomic transactions limit scaling to larger numbers of processors for snooping-based protocols**

Atomic Coherence Transactions: Example

- N processors, each with a private cache, run at a 2GHz frequency and are connected via a shared bus. All processors run a multi-threaded parallel program where each thread has an MPKI = 10 (7 MPKI from other caches needing 10 ns and 3 MPKI from memory needing 80 ns). All tasks have an IPC of 0.5. What is the value of N beyond which processors saturate the shared bus?

Cycle Time = 1/frequency = 0.5 ns; Cycles/sec = frequency = 2×10^9

Instructions/Sec (1 Proc.) = IPC x Cycles/sec = $0.5 \times (2 \times 10^9) = 10^9$

Memory Requests/Sec (1 Proc.) = IPS x MPKI(mem)/1000 = $10^9 \times 3/1000 = 0.003 \times 10^9$

Other Cache Requests/Sec (1 Proc.) = IPS x MPKI(cache)/1000 = $10^9 \times 7/1000 = 0.007 \times 10^9$

Bus Time Required/Sec (1 Proc.) = Memory Requests/Sec x Time/Mem_Request + Cache Requests/Sec x Time/OtherCache_Request

$= 0.003 \times 10^9 \times 80 \times 10^{-9} + 0.007 \times 10^9 \times 10 \times 10^{-9} = 0.31$ sec

Bus Time Required/Sec (N Proc.) = 0.31 x N

Beyond N = 3 processors, Bus will saturate

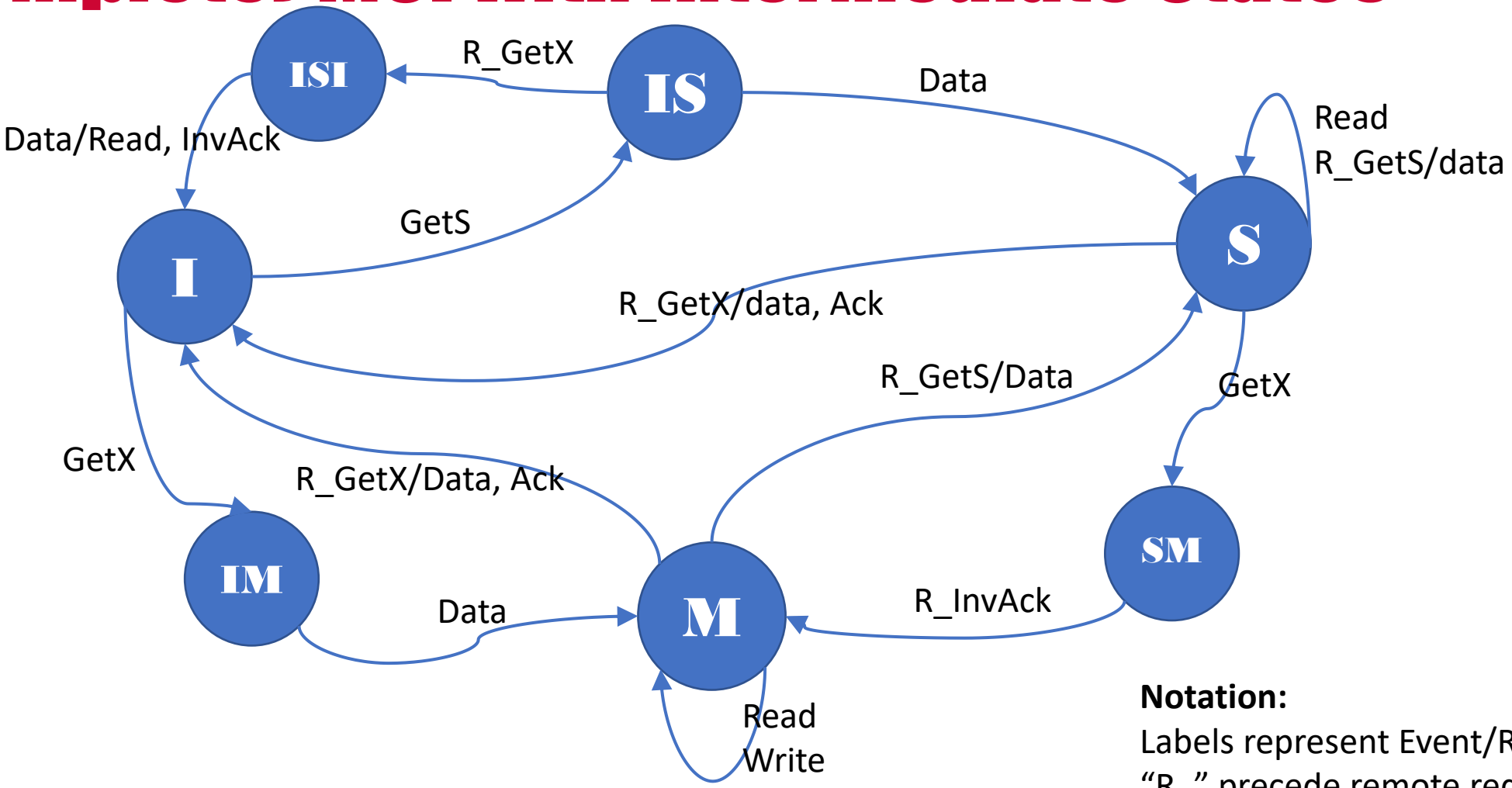
Non-Atomic Coherence Transactions

- **To avoid the scaling issues and lower MLP caused by atomic coherence transactions, systems use a "Split-Transaction" Bus (or other interconnect)**
- **Split-Transaction bus is acquired to send a request then released. It needs to be re-acquired before the response is sent out**
- **This implies that each coherence transaction is split into a request and a response, and the shared bus is released between the request and the response**
  - In previous example, actual time needed to send request and receive response from other caches/memory would be much lower, limited by cache or memory bandwidth
- **Split-Transaction Bus leads to implementation complexity with coherence protocols**

Intermediate States

- **Non-atomic transactions require additional (intermediate) states for cache lines that have sent out a request and are waiting for response**
  - Examples (MSI):
    - Intermediate state IS follows a GetS request from I while waiting for data
    - Intermediate state SM follows a GetX request from S while waiting for invalidation acknowledgments
- **Intermediate states may need to respond to other requests that occur between request and response**
  - Example (MSI): Block in SM may need to send a negative acknowledgment to a GetS request
- **Alternatively, more intermediate states are needed to indicate that another request was received while waiting for the original response**
  - Example (MSI): Block in IS may need to go to ISI to indicate that a GetX request was received. After data is returned, ISI completes the read, sends an invalidation ack then goes back to I (instead of S)

(Incomplete) MSI with Intermediate States



Notation:
Labels represent Event/Response
"R_" precede remote requests

- **Which other states/transitions are missing?**

Multi-Level Protocols

- **Inclusion/Exclusion policy for multi-level caches:**
  - Inclusive caches
  - Exclusive caches
  - Non-inclusive (non-exclusive) caches
- **Which caches need to snoop?**
- **CMP private vs. shared caches**
  - Private caches maintain coherence state
  - Shared L2/L3 caches may store coherence state of all lower-level private caches

Directory Coherence Protocols

Why Directory Protocols?

- **Snooping-based protocols may not scale**
  - All requests must be broadcast to all processors
  - Cache tag directory needs to handle both local and remote requests
  - All processors should monitor all requests on the shared interconnect
  - Shared interconnect utilization can be high, leading to very long wait times
- **Directory protocols**
  - Coherence state maintained in a directory associated with memory
  - Requests to a memory block do not need broadcasts
    - Served by local nodes if possible
    - Otherwise, sent to owning node
- **Note: Some snooping-based protocols do not require broadcast, and therefore are more scalable**

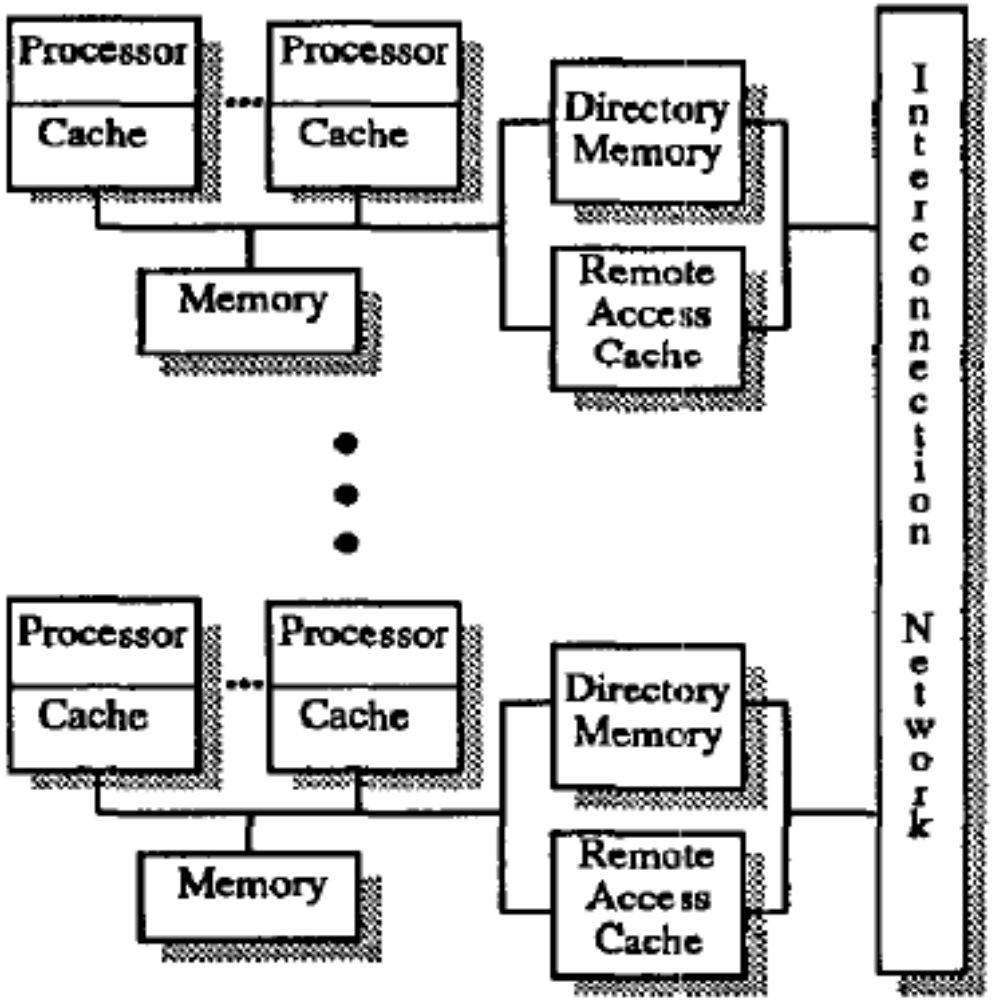
Design Issues for Distributed Coherence Protocols

- **Correctness**
  - Memory consistency model: Performance vs. ease of programming
  - Deadlock avoidance
  - Error handling (fault tolerance)
- **Performance**
  - Latency
  - Bandwidth
- **Distributed Control and Complexity**
- **Scalability**

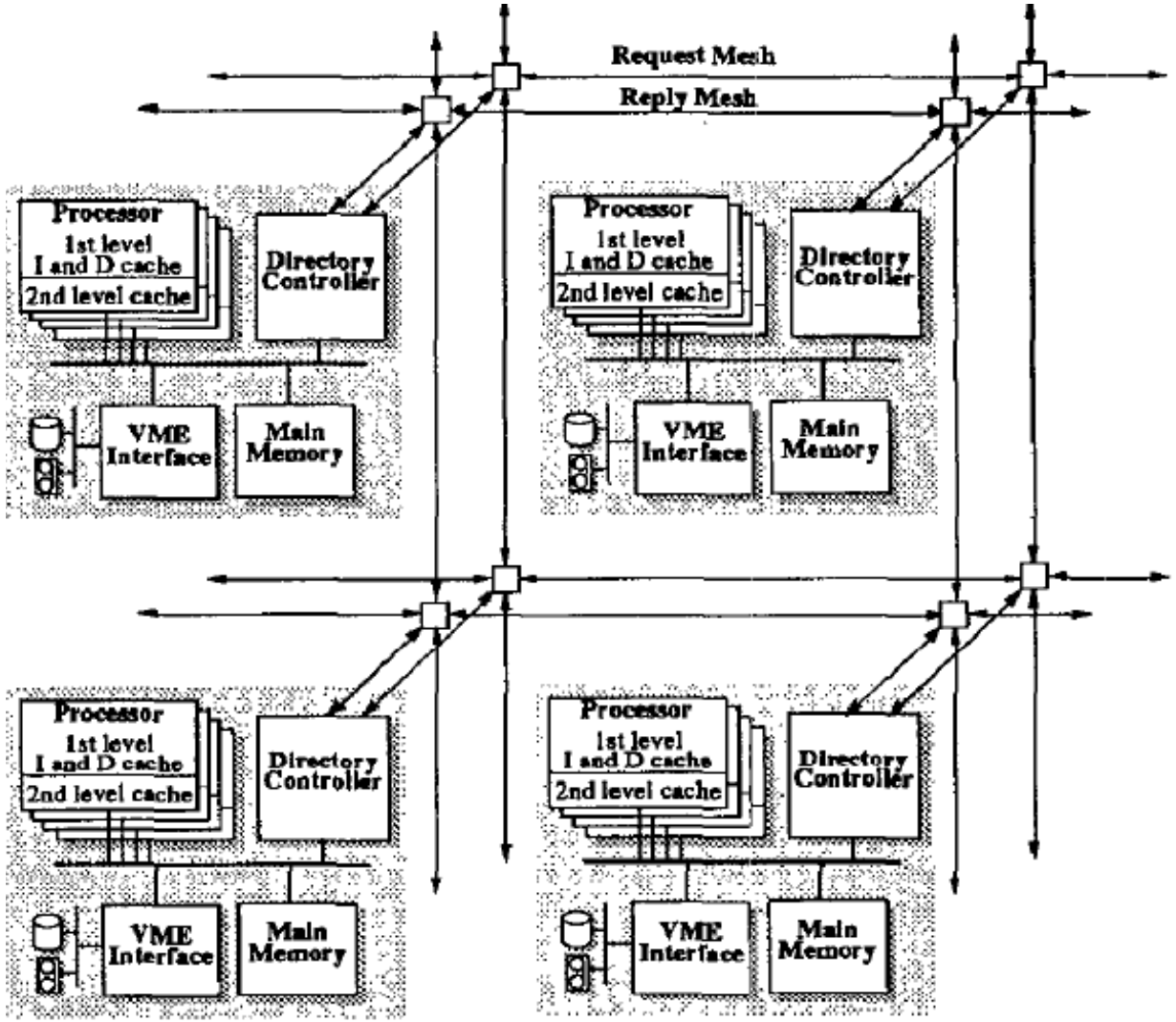
The Stanford DASH Prototype

- **DASH: Directory Architecture for SHared memory**
- **Architecture consists of many clusters**
  - Each cluster contains 4 processors
  - Processor caches
    - L1I: 64KB, direct mapped
    - L1D: 64KB, direct-mapped, write-through
    - L2: 256KB, direct-mapped, write-back
    - 4-word write buffer
  - Snooping implemented within a cluster (Illinois protocol, similar to MSI)

DASH Architecture



Lenoski et al 1990, Figure 1

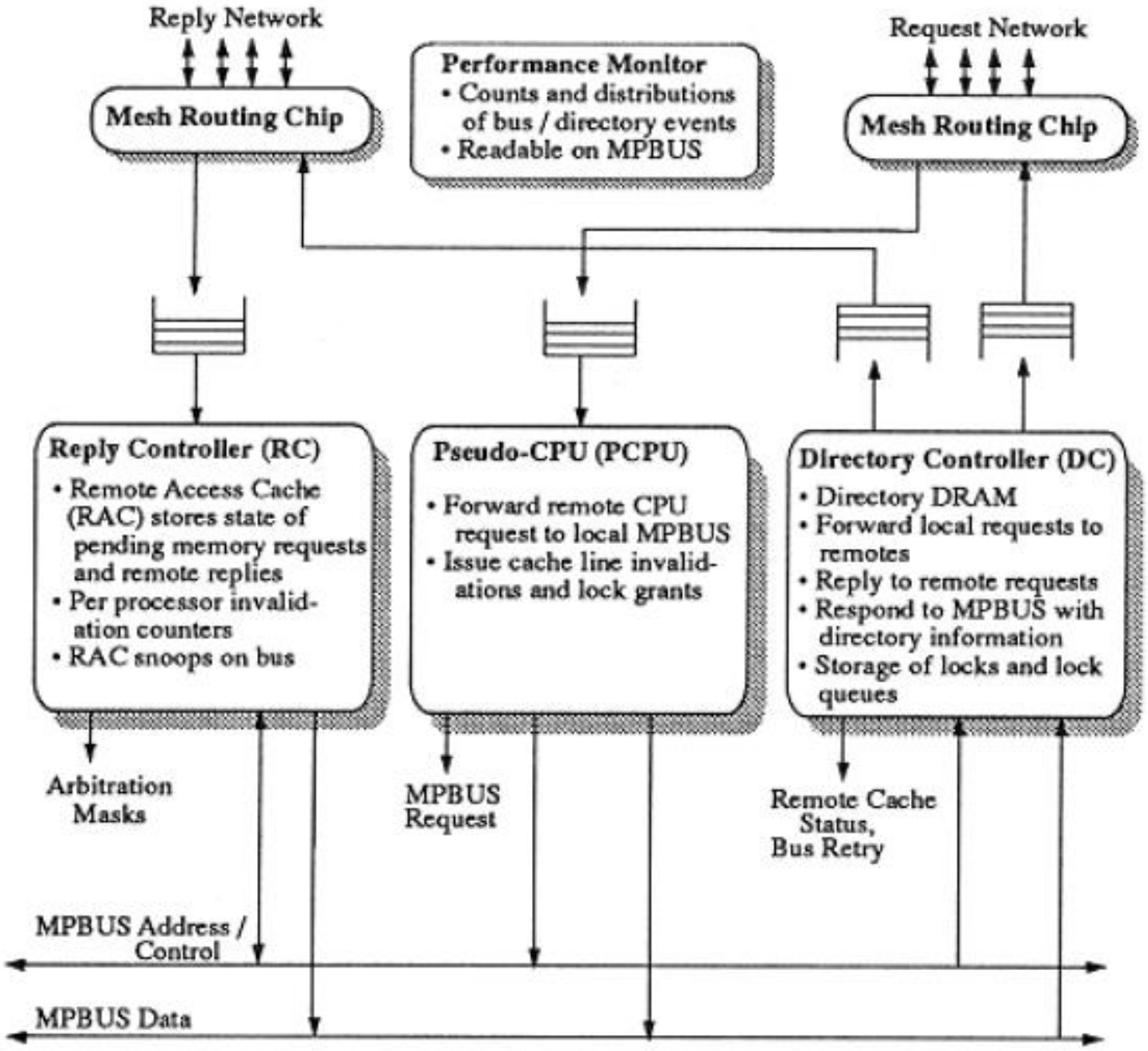


Lenoski et al 1990, Figure 2: 2x2 DASH System

DASH Directories

- **Directory controller (DC)**
  - Directory memory corresponding to cluster's main memory portion
  - Initiates out-bound network requests and replies
- **Pseudo-CPU (PCPU)**
  - Buffers incoming requests and issues them on cluster bus
  - Mimics a CPU on behalf of remote processors (except for bus replies sent by DC)
- **Reply Controller (RC)**
  - Remote Access Cache (RAC) tracks outstanding requests by local processors
  - Receives and buffers corresponding replies from remote clusters
  - RAC snoops on bus
- **Requests and replies sent on two different networks using wormhole routing**

DASH Directory



Lenoski et al 1990, Figure 3

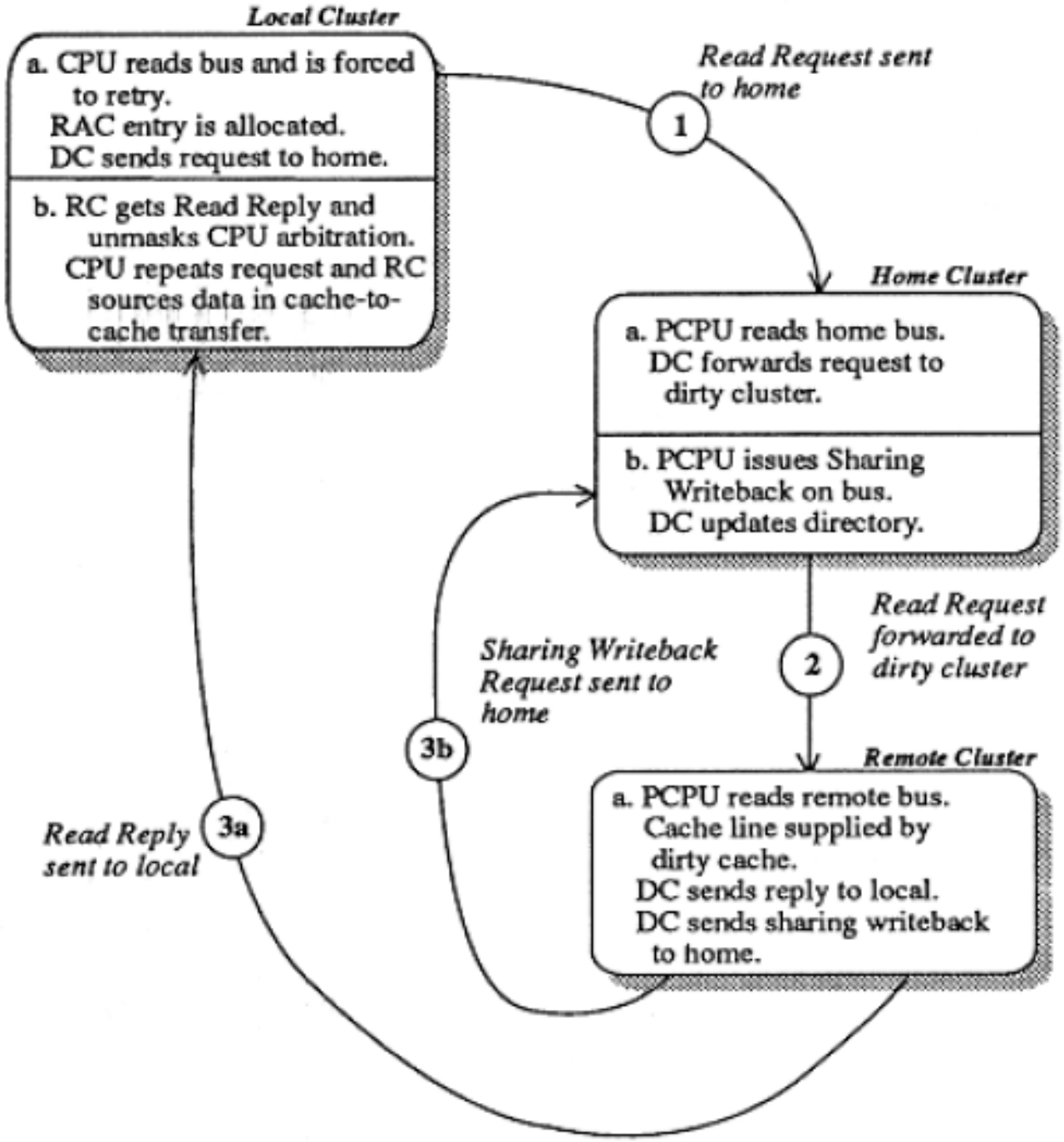
DASH Coherence Protocol

- **Terminology**
  - Local cluster: cluster containing the processor originating a request
  - Home cluster: cluster containing the main memory and directory for a given memory address
  - Remote cluster: Any cluster other than local and home clusters
  - Local memory: main memory associated with the local cluster
  - Remote memory: Any memory whose home is not the local cluster
- **Invalidation-based protocol**
  - Cache states: invalid, shared, and dirty
- **Directory state (for all local memory blocks)**
  - Uncached-remote: not cached by any remote cluster
  - Shared-remote: Cached, unmodified, by one or more remote clusters
  - Dirty-remote: Cached, modified, by one remote cluster
- **Owning cluster for a block is the home cluster except if *dirty-remote***
- **Owning cluster responds to requests and updates directory state**

Read Requests

- **Initiated by CPU load instruction**
- **If address is in L1 cache, L1 supplies data – otherwise, fill request sent to L2**
- **If address is in L2, L2 supplies data – otherwise, read request sent on bus**
- **If address is in the cache of another processor in the cluster or in the RAC, that cache responds**
  - Shared: data transferred over the bus to requester
  - Dirty: data transferred over bus to requester, RAC takes ownership of cache line
- **If address not in local cluster, processor retries bus operation, and request is sent to home cluster, RAC entry is allocated**

Read Requests to Remote Nodes

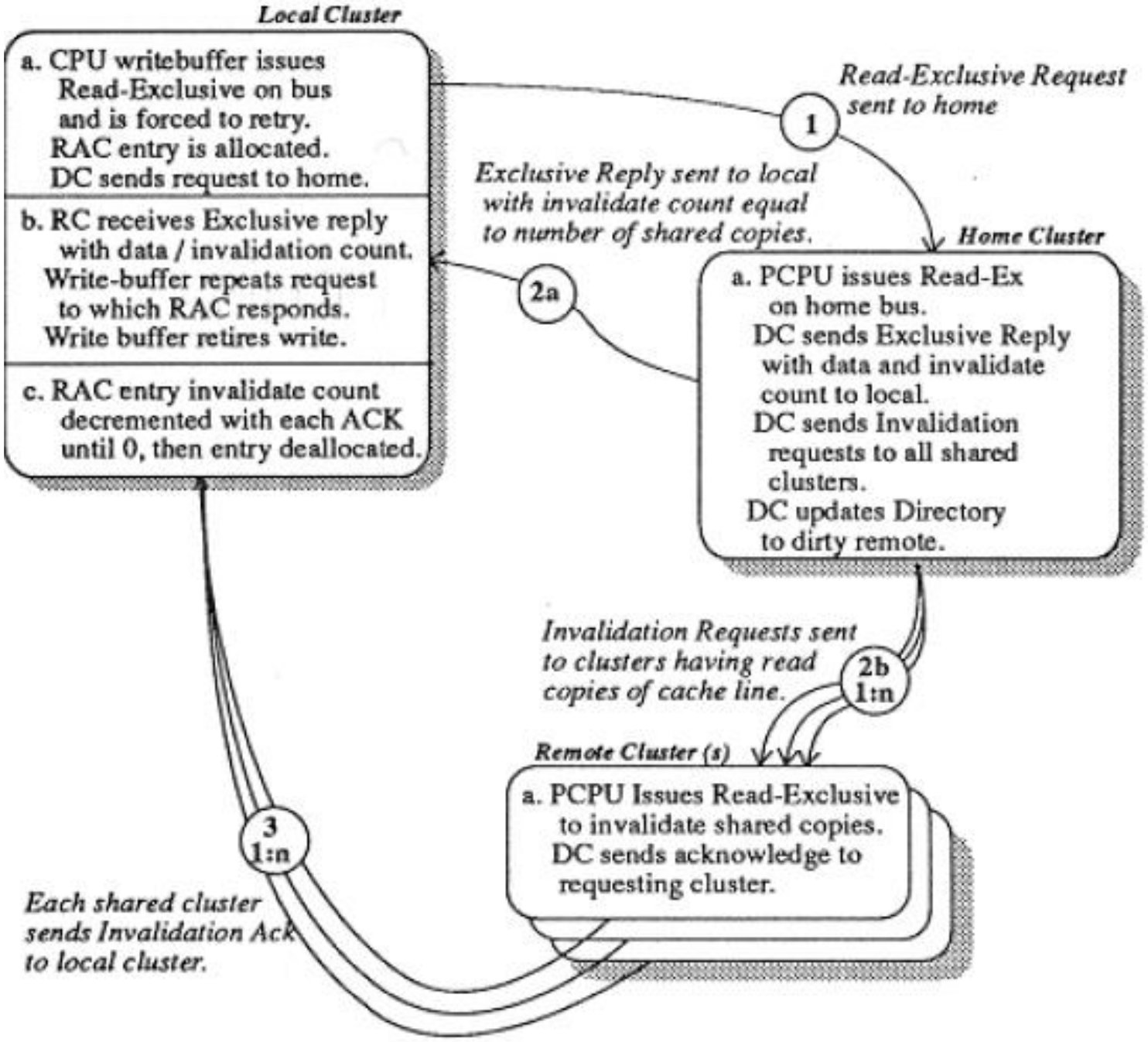


Lenoski et al 1990, Figure 4

Read-Exclusive Requests

- **Initiated by CPU store instruction**
- **Data written through L1 and buffered in a write buffer**
- **If L2 has ownership permission, write is retired – otherwise, read-exclusive request sent on local bus**
  - Write buffer is stalled
- **If address is in "dirty" in one of the caches in the cluster or in the RAC**
  - Owning cache sends data and ownership to requester
  - Owning cache invalidates its copy
- **If address not in local cluster**
  - Processor retries bus operation
  - Request is sent to home cluster
  - RAC entry is allocated

Remote Read-Exclusive Requests



Lenoski et al 1990, Figure 5

Other Implementation Details

- **Writeback requests: When a dirty block is replaced**
  - Home is local cluster: Write data to main memory
  - Home is a remote cluster: Send data to home which updates memory and state as “uncached-remote”
- **Exception conditions**
  - Request to a dirty block of a remote cluster after it gave up ownership
  - Ownership bouncing back and forth between two remote clusters while a third cluster requests block
  - Multiple paths in the system lead to requests being received out of order
- **Amount of information stored in directory affects scalability**
  - For each memory block, DASH stores state and bit vector for other processors
  - For a more scalable system, overhead needs to be lower

Read Operations	
Hit in 1st Level Cache	1 pclock
Fill from 2nd Level Cache	12 pclock
Fill from Local Cluster	22 pclock
Fill from Remote Cluster	61 pclock
Fill from Dirty Remote, Remote Home	80 pclock
Fill operations fetch 16 byte cache blocks and empty the write-buffer before fetching the read-miss cache block.	
Write Operations	
Hit on 2nd Level Owned Block	3 pclock
Owned by Local Cluster	18 pclock
Owned in Remote Cluster	57 pclock
Owned in Dirty Remote, Remote Home	76 pclock
Write operations only stall the write-buffer, not the processor, while the fill is outstanding. *Write delays assume Release Consistency (i.e. they do not wait for remote invalidates to be acknowledged).*	

Latency for Memory Operations:

Lenoski et al 1990, Figure 5

The SGI Origin

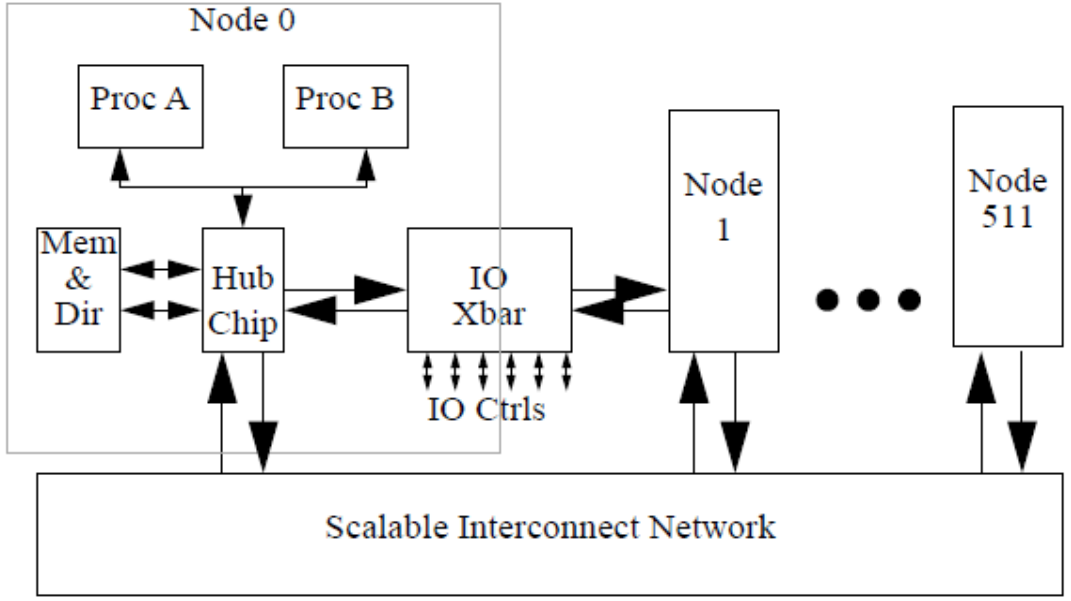
- Cache coherent non-uniform memory access
- Up to 512 nodes
- Scalable Cray link network (hypercube)
- 1 or 2 R10000 MIPS processors per node
- Up to 4G bytes per node
- Node connects to a portion of the IO subsystem
- No snooping within node

SGI Origin: Key Goals

- **Scale to large number of processors**
- **Provide higher performance per processor**
- **Maintain cache-coherent globally addressable memory model**
  - For ease of programming
- **Entry level and incremental cost of the system lower than a high performance SMP**

Origin Architecture

- Distributed shared memory (DSM)
- Directory based cache coherence
- Designed to minimize latency difference between local and remote memory
- Hardware and software provided to insure most memory references are local



Laudon&Lenoski 1997, Figure 1

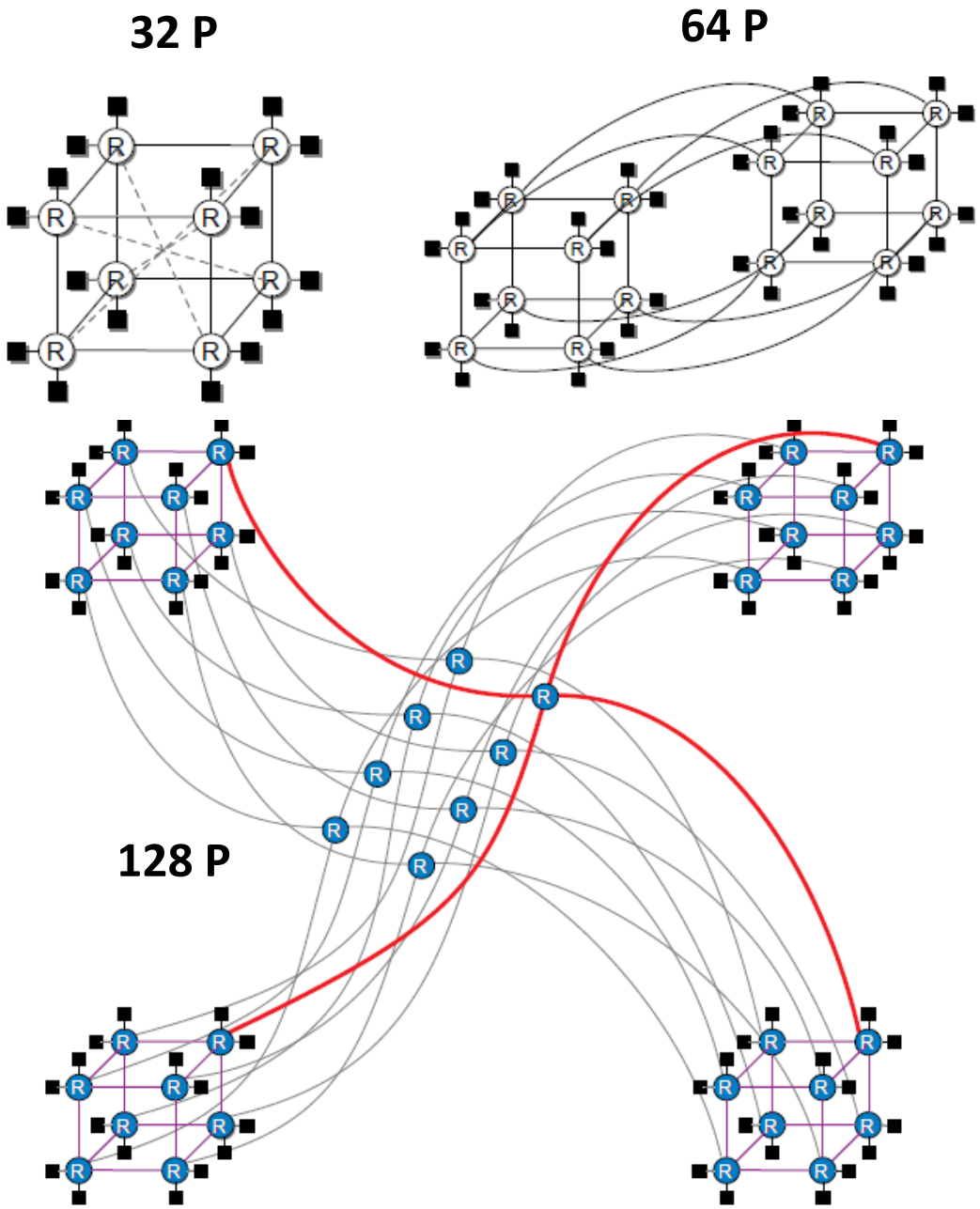
- Cache coherence does not require in-order message delivery
- I/O subsystem is also distributed and globally addressable
- I/O can DMA to and from all memory in the system
- Cluster bus is multiplexed but is not a snoopy bus
  - ➢ Reduce local and remote memory latency
    - ❑ Fewer processors on the bus
    - ❑ Remote request does not need to wait for snoop response

Origin Architecture (Cont.)

- **Non-snoopy node bus tradeoff**
  - Disadvantage: remote bandwidth needs to match local bandwidth, unlike in SMP node systems
  - Advantage: easier migration path for existing SMP software
- **Page migration and replication insures most references are local**
  - Memory reference hardware counters
  - Copy engine to copy at near peak memory bandwidth
- **Rich synchronization primitives**
- **Fetch and op primitives are not cached and performed at memory**
  - Useful in highly contended locks
- **HUB implements 4-way full crossbar between processors, memory and I/O-network**
- **RAS features**
  - ECC in external cache and memory
  - Faulty packets automatic retries
  - Modular design provides highly available hardware

SGI Origin: Network

- Six ported router chip
- Fat, Bristled hypercube
- Low latency wormhole routing
- Four virtual channels per physical channel
- Congestion control to allow messages to adaptively switch between two virtual channels
- Support for 256 levels of message priority
- Increased priority via packet aging
- Automatic packet retries
- Software programmable routing tables



Laudon&Lenoski 1997, Figures 3-4

Cache Coherence Protocol

- **Similar to DASH protocol but with significant improvements**
  - ➢MESI protocol is fully supported
    - ❑Single fetch from memory for read-modify-writes
    - ❑Permits processor to replace E block in cache without informing directory
    - ❑ Requests from processors that had replaced E blocks can be immediately satisfied from memory
  - ➢Support of upgrade requests from S to E without data transfer
- **Coherence protocol supports Read, Read-Exclusive and Writeback requests**

Origin Coherence Protocol: Read Requests

1. Processor issues read request.
2. Read request goes across network to home memory (requests to local memory only traverse Hub).
3. Home memory does memory read and directory lookup.
4. If directory state is Unowned or Exclusive with requestor as owner, transitions to Exclusive and returns an exclusive reply to the requestor. *Go to 5a*.
   If directory state is Shared, the requesting node is marked in the bit vector and a shared reply is returned to the requestor. *Go to 5a*.
   If directory state is Exclusive with another owner, transitions to Busy-shared with requestor as owner and send out an intervention shared request to the previous owner and a speculative reply to the requestor. *Go to 5b*.
   If directory state is Busy, a negative acknowledgment is sent to the requestor, who must retry the request. QED

5a. Processor receives exclusive or shared reply and fills cache in CEX or shared (SHD) state respectively. QED

5b. Intervention shared received by owner. If owner has a dirty copy it sends an shared response to the requestor and a sharing writeback to the directory. If owner has a clean-exclusive or invalid copy it sends an shared ack (no data) to the requestor and a sharing transfer (no data) to the directory.

6a. Directory receives shared writeback or shared transfer, updates memory (only if shared writeback) and transitions to the shared state.

6b. Processor receives both speculative reply and shared response or ack. Cache filled in SHD state with data from response (if shared response) or data from speculative reply (if shared ack). QED

Laudon&Lenoski 1997

Origin Coherence Protocol: Read-Exclusive Requests

1. Processor issues read-exclusive request.
2. Read-exclusive request goes across network to home memory (only traverses Hub if local).
3. Home memory does memory read and directory lookup.
4. If directory state is Unowned or Exclusive with requestor as owner, transitions to Exclusive and returns an exclusive reply to the requestor. *Go to 5a.*
   If directory state is Shared, transitions to Exclusive and a exclusive reply with invalidates pending is returned to the requestor. Invalidations are sent to the sharers. *Go to 5b.*
   If directory state is Exclusive with another owner, transitions to Busy-Exclusive with requestor as owner and sends out an intervention exclusive request to the previous owner and a speculative reply to the requestor. *Go to 5c.*
   If directory state is Busy, a negative acknowledgment is sent to the requestor, who must retry the request. QED

5a. Processor receives exclusive reply and fills cache in dirty exclusive (DEX) state. QED

5b. Invalidates received by sharers. Caches invalidated and invalidate acknowledgments sent to requestor. *Go to 6a.*

5c. Intervention shared received by owner. If owner has a dirty copy it sends an exclusive response to the requestor and a dirty transfer (no data) to the directory. If owner has a clean-exclusive or invalid copy it sends an exclusive ack to the requestor and a dirty transfer to the directory. *Go to 6b.*

6a. Processor receives exclusive reply with invalidates pending and all invalidate acks. (Exclusive reply with invalidates pending has count of invalidate acks to expect.) Processor fills cache in DEX state. QED

6b. Directory receives dirty transfer and transitions to the exclusive state with new owner.

6c. Processor receives both speculative reply and exclusive response or ack. Cache filled in DEX state with data from response (if exclusive response) or data from speculative reply (if exclusive ack). QED

Laudon&Lenoski 1997

Origin Coherence Protocol: Writeback Requests

1. Processor issues writeback request.
2. Writeback request goes across network to home memory (only traverses Hub if local).
3. Home memory does memory write and directory lookup.
4. If directory state is Exclusive with requestor as owner, transitions to Unowned and returns a writeback exclusive acknowledge to the requestor. *Go to 5a.*
   If directory state is Busy-shared, transitions to Shared, a shared response is returned to the owner marked in the directory. A writeback busy acknowledgment is also sent to the requestor. *Go to 5b.*
   If directory state is Busy-exclusive, transitions to Exclusive, an exclusive response is returned to the owner marked in the directory. A writeback busy acknowledgment is also sent to the requestor. *Go to 5b.*

5a. Processor receives writeback exclusive acknowledgment. QED

5b. Processor receives both a writeback busy acknowledgment and an intervention. QED

Laudon&Lenoski 1997

Configuration and Performance

- **CPU Configuration**
  - MIPS R10000
  - 195 MHz
  - 4-way out-of-order
  - 4 M byte L2 cache
  - Bus connected to the HUB chip
- **Latency variation:**

Memory level	Latency (ns)
L1 cache	5.1
L2 cache	56.4
local memory	310
4P remote memory	540
8P avg. remote memory	707
16P avg. remote memory	726
32P avg. remote memory	773
64P avg. remote memory	867
128P avg. remote memory	945

Laudon&Lenoski 1997, Table 4

Reading Assignments

- ARCH Chapter 5.2, 5.3, 5.4 (Read)
- P. Stenstrom, “A Survey of Cache Coherence Schemes for Multiprocessors,” IEEE Computer 1990 (Skim)
- D. Lenoski et al., "The Directory-based Cache Coherence Protocol for the DASH Multiprocessor," ISCA 1990 (Read)
- J. Laudon and D.Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," ISCA 1997 (Skim)