LECTURE 5

MULTIPROCESSOR SYSTEMS (II)

Miquel Pericàs

EDA284/DIT361 - 2019/2020

What's cooking

1. **Lectures**
   - Today: Shared memory multiprocessors
   - Friday 8h: Roofline Model
     - will check 9h is doable, TBA via Canvas
2. **Practice session Friday 10h**
   - Problems 1.2 (a-e), 5.2, 2019 Re-exam Problem 3
   - Try to solve the problems ahead of the session!
3. **EDA284 project**
   - Teams and selected topics are now in Canvas
   - https://chalmers.instructure.com/courses/8752/files/folder/Project#
   - Please contact your partners ASAP and make sure that they are available!

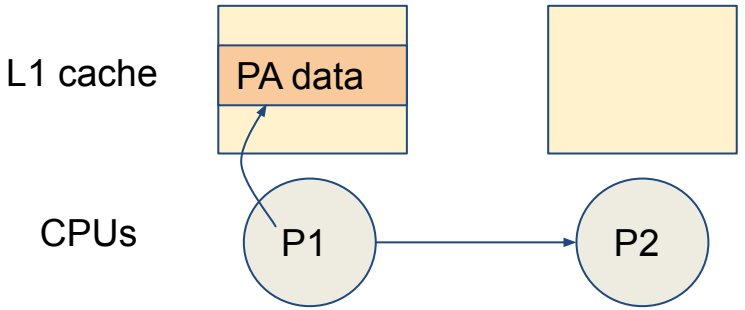
Why are update protocols not popular?

Follow up on yesterday's discussion:

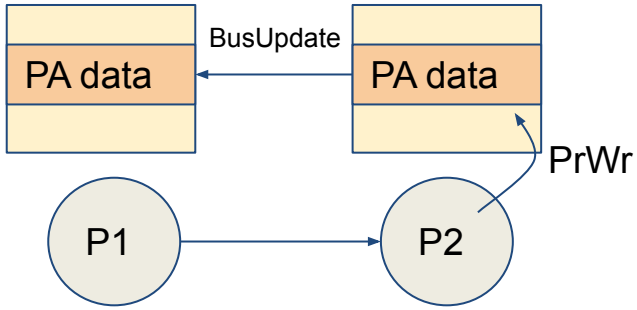
For multi-threaded applications, choice of write-update vs write-invalidate is governed by (producer-consumer) sharing amount vs length of write runs on shared data

- write-update is lower cost as long as written data is consumed often

For single-threaded workloads a problematic case arises. Why? Because of O/S-initiated process migration (e.g., to balance load across processors)



O/S migrates process A from P1 to P2



After migration keep on updating P1 cache but there are no consumers!

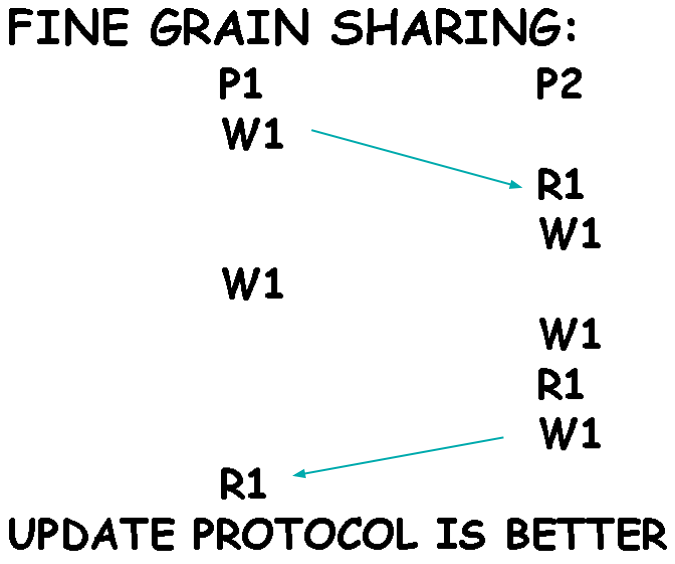
Remember: most workloads are single-threaded. Even MT workloads are occasionally migrated. This is my *guess* as to why update-protocols are not popular

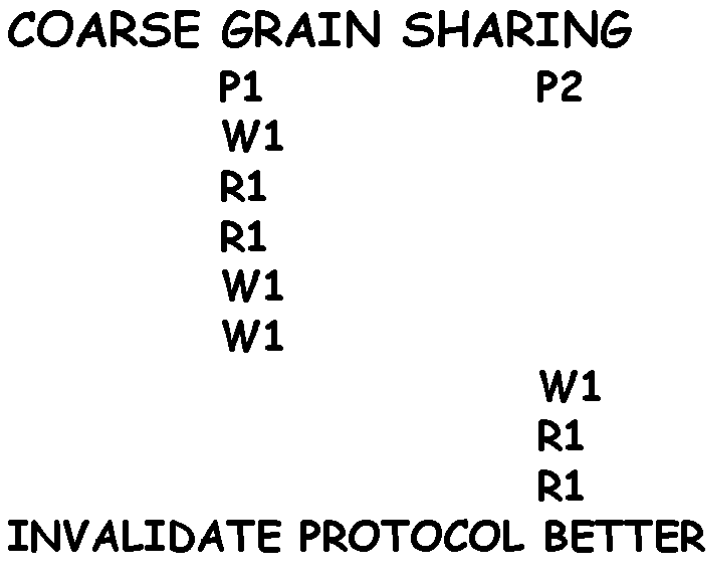
OUTLINE (Lectures 4+5)

- Shared-memory parallel programming
- Bus-based shared memory systems
  - architectures
  - coherence protocols
  - coherence optimizations
  - multi-level caches
- **Classification of Misses**
- **Scalable shared memory systems**

TRUE vs FALSE SHARING

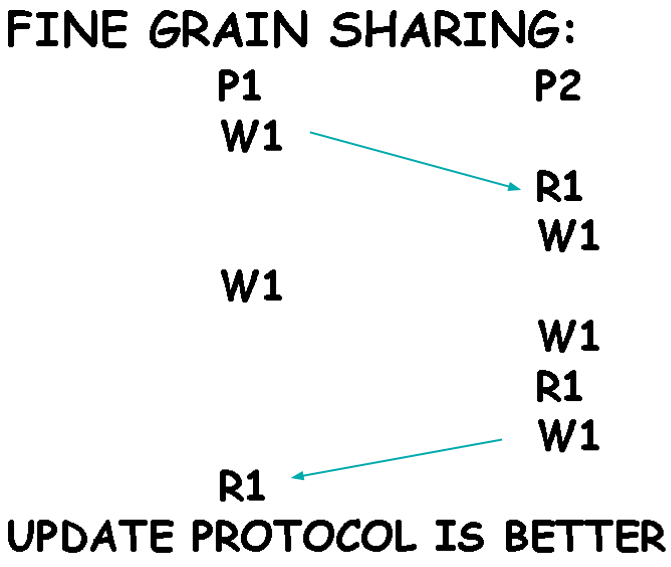
- **Assume that a block is shared by two processors**
  - the block contains two words, word1 and word2
- **True sharing:** the two processors access the same word

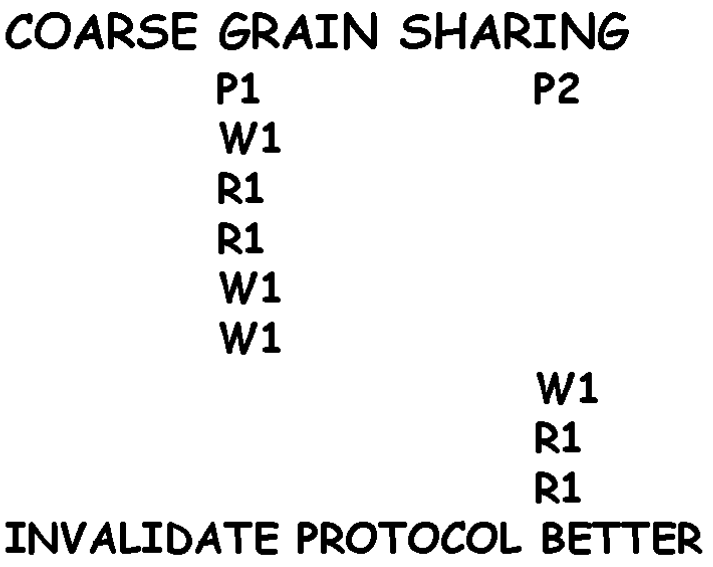




true sharing communicates values – essential

TRUE vs FALSE SHARING





- **False sharing:** the two processors access different words in the block
  - if P1 accesses W1 and P2 accesses W2. then sharing is useless
  - Write invalidate causes false sharing misses
  - Write update causes false sharing updates to dead copies

False sharing does not communicate values.

useless (non-essential), pure overhead

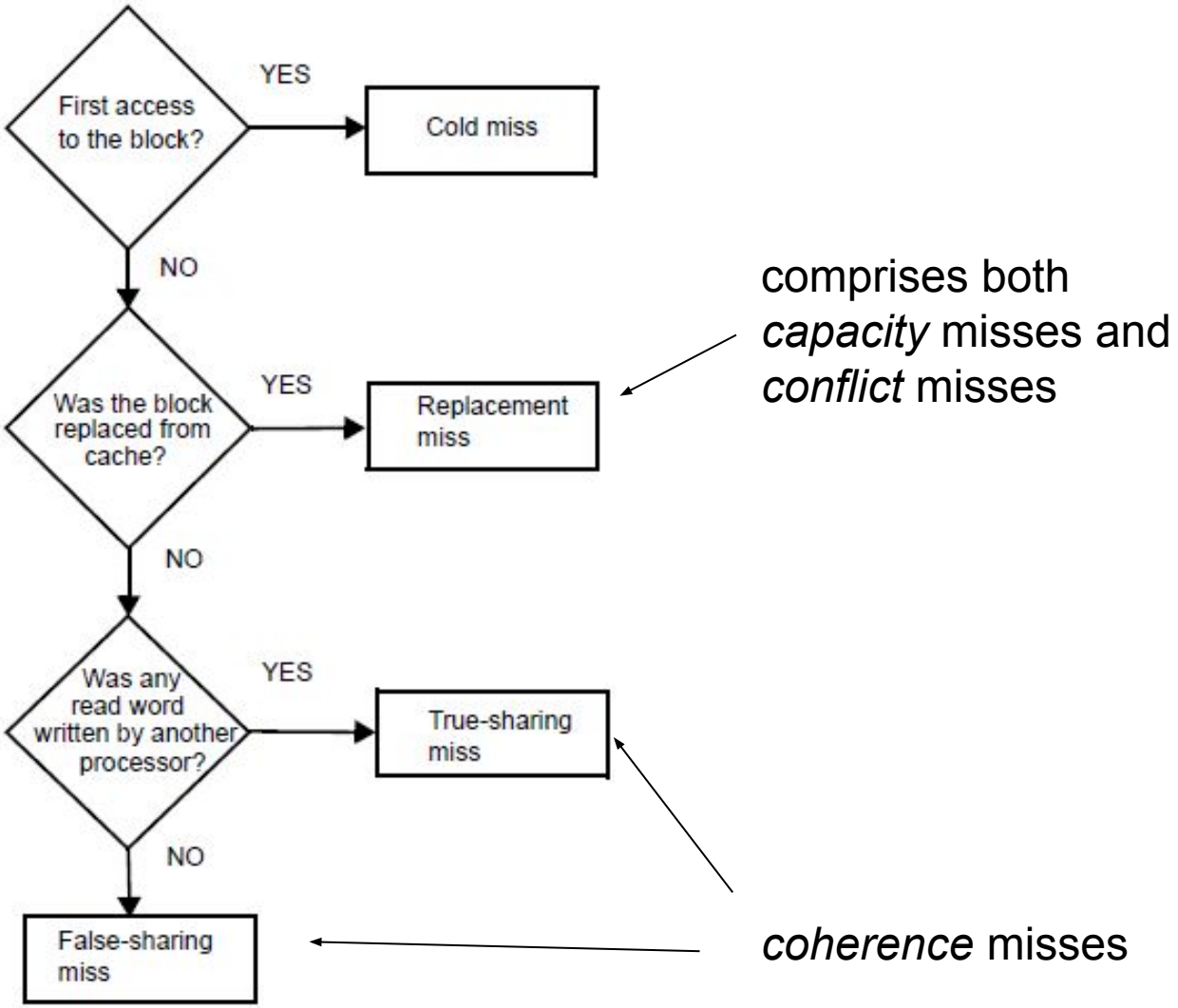
ESSENTIAL VS NON-ESSENTIAL MISSES

Example: assume **A**,**B** and **C** belong to same block (**B1**) and **D** to another

Time step	Processor 1	Processor 2	Processor 3	MISS CLASSIFICATION
1	R_A			COLD MISS
2		R_B		COLD MISS
3			R_C	COLD MISS
4			R_D (evict block B1)	COLD MISS
5	W_A			
6		R_A		TRUE SHARING MISS
7	W_B			
8		R_A		FALSE SHARING MISS
9			R_C	REPLACEMENT MISS

- **4Cs: cold**, **true sharing** (coherence) and **replacement** (conflict, capacity) are **essential;** whereas **false sharing** misses are **non-essential**; can be ignored
- A transferred word is essential if it carries a new value into a cache, and the value is accessed by the attached processor during the lifetime of the word in the cache

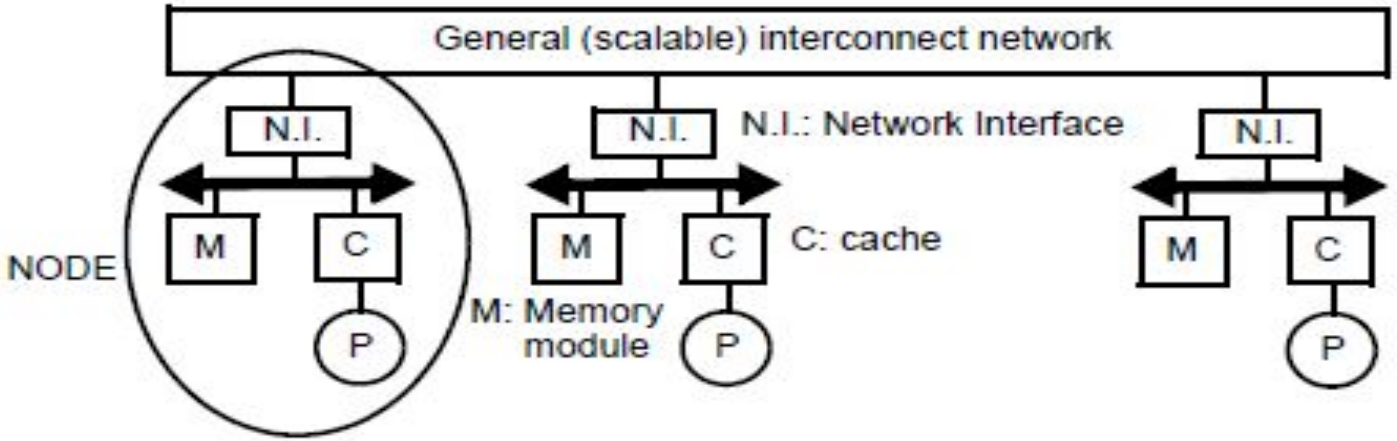
CLASSIFICATION OF MISSES



SCALABLE CACHE COHERENCE SOLUTIONS

Bus-based multiprocessors are inherently non-scalable

1) Bus BW capped by product of bus wires and clock rate
2) length & load on wires grows with #nodes (latency↑ BW↓)
3) memory access latency increases



cache-coherent, Non-Uniform Memory Architectures (cc-NUMA):

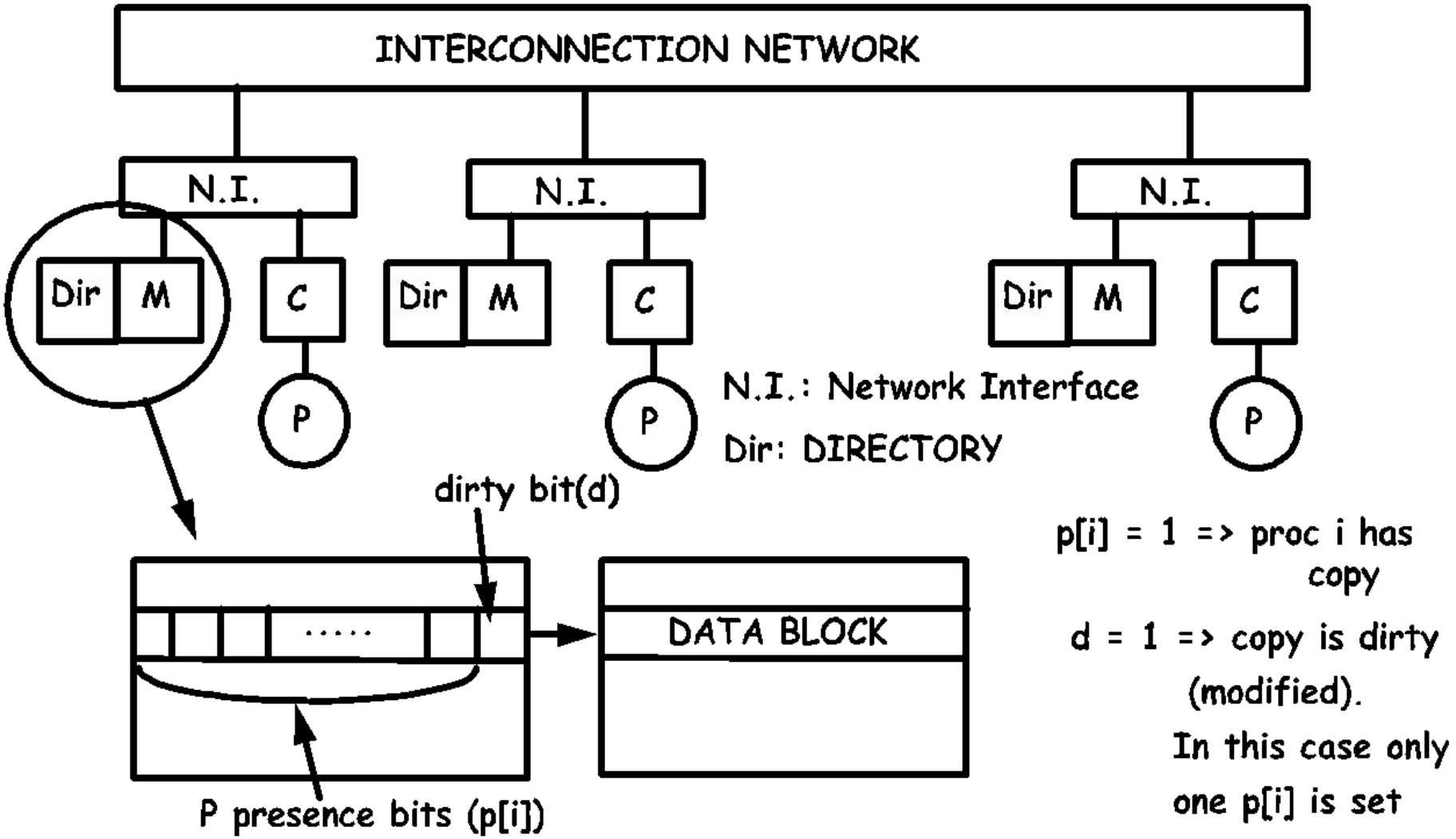
- distribute main memory across nodes to leverage locality (based on physical address bits; at page granularity, mapping can be tuned by O/S)
- use a general (scalable) interconnect to accommodate BW

snoopy (broadcast) based cache protocols are not scalable

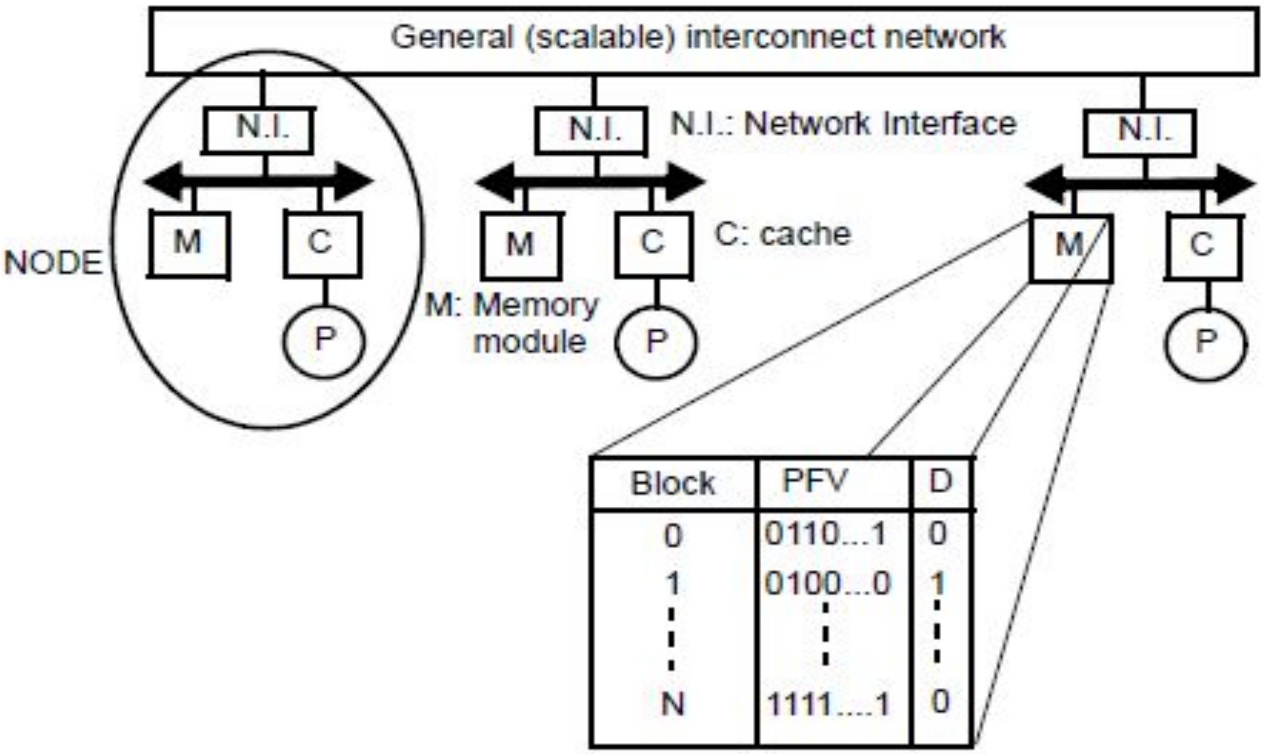
scalable cache protocols need to keep track of sharers

DIRECTORY-BASED PROTOCOLS

- Each memory block has a **Home** := the node where the block resides (in DRAM)
- A directory entry associated with the memory block points to nodes with a copy of the block (remote nodes) + tracks the state of the block



PRESENCE-FLAG VECTOR SCHEME

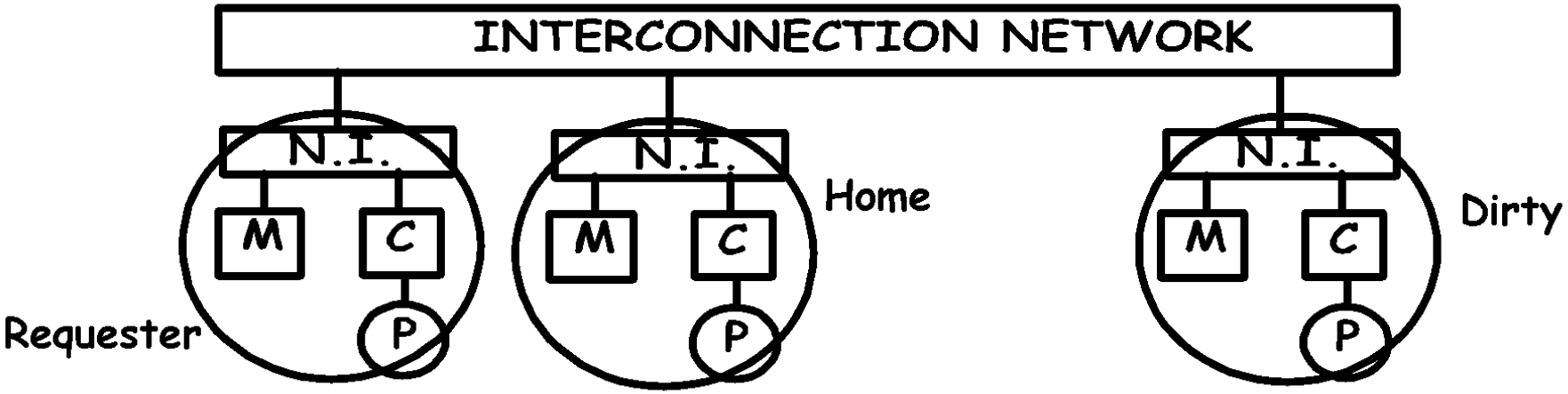


EXAMPLE

- Block 1 is cached by processor 2 only and is dirty
- Block N is cached by all processors and is clean
- Block 0 is cache by a subset of processors and is clean

cc-NUMA PROTOCOLS

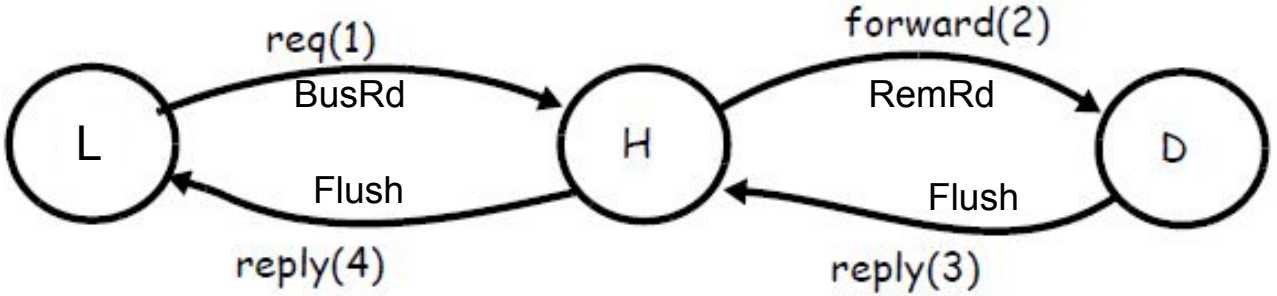
- based on snooping protocols (MSI-invalidate, MSI-update)
- **protocol agents:**
  - **home node (H)**: node where the memory block and its directory entry reside
  - **local node (L)**: node making the request
  - **dirty node (D)**: node holding the latest, modified copy
  - **shared node(s) (S)**: nodes holding a shared copy



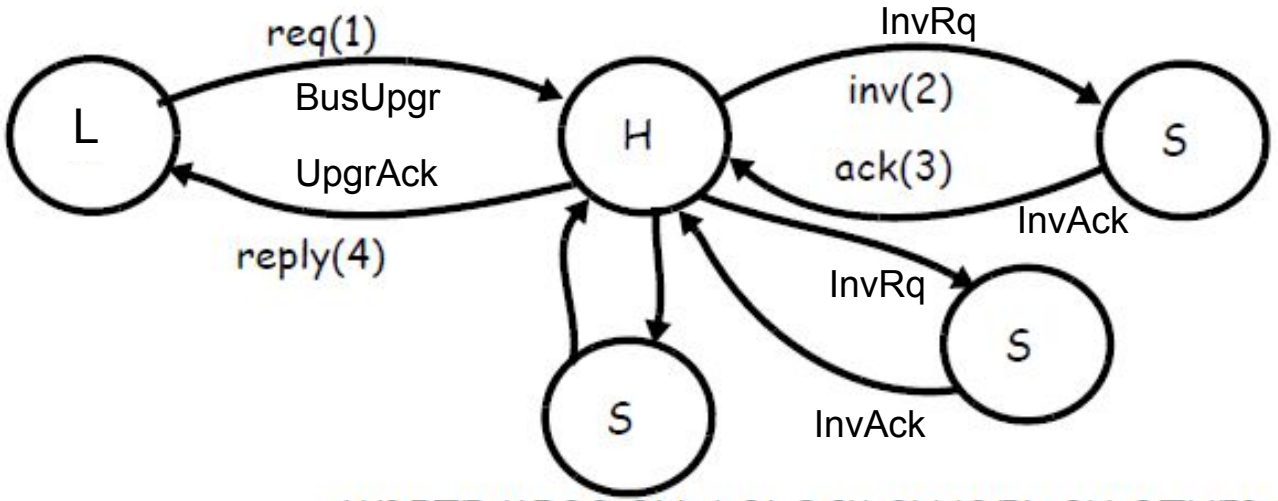
home may be the same node as requester or dirty

note: busy bit per entry

MSI INVALIDATE PROTOCOL IN cc-NUMAs



READ MISS ON A BLOCK DIRTY REMOTE



WRITE MISS ON A BLOCK SHARED BY OTHERS

note: in MSI-update the transitions are the same except that updates are sent instead of invalidations

Multiple concurrent transactions are avoided by busy bit:
Home node is central arbiter that serializes all requests to a memory block!

Coherence Transitions

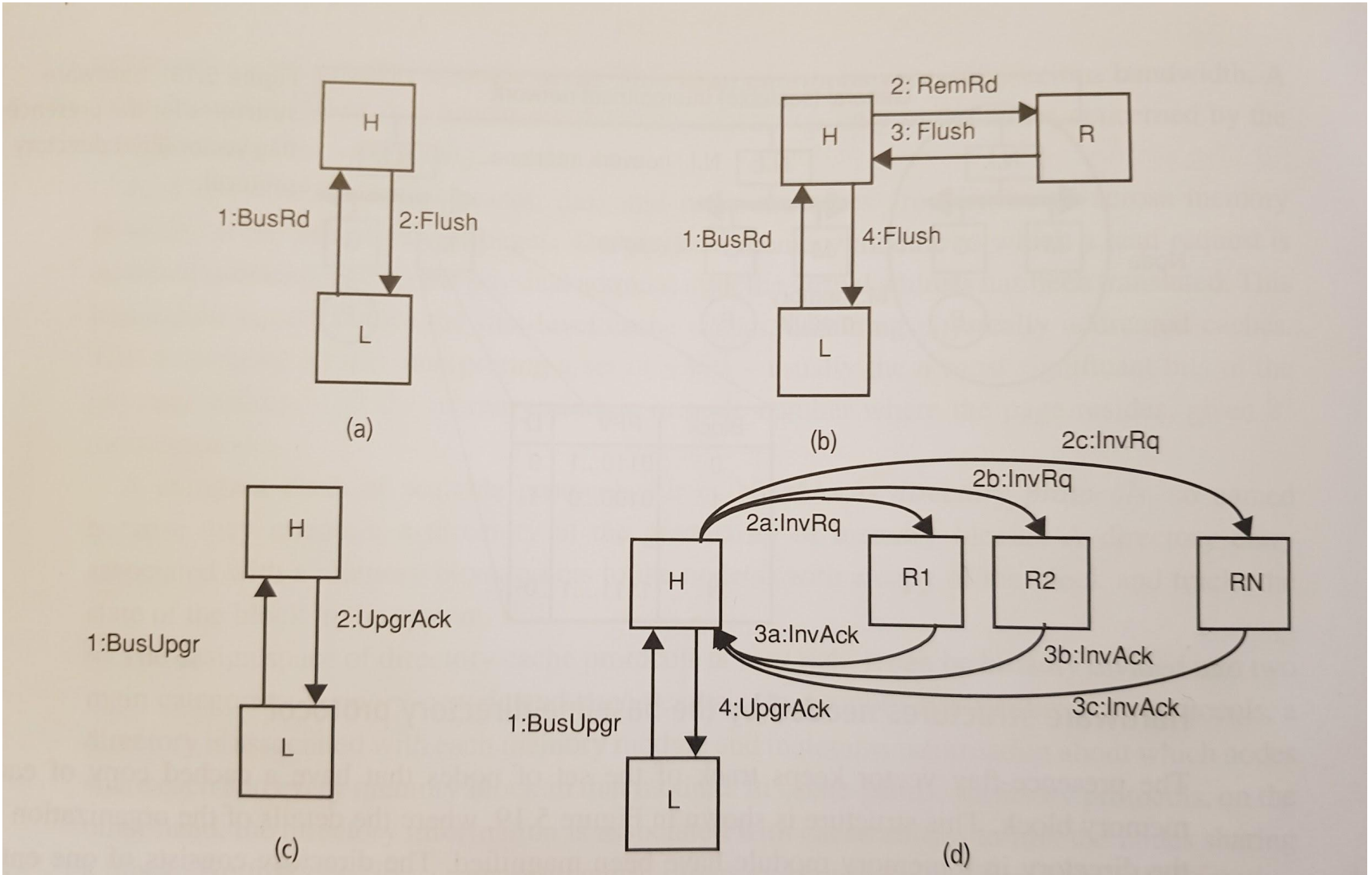
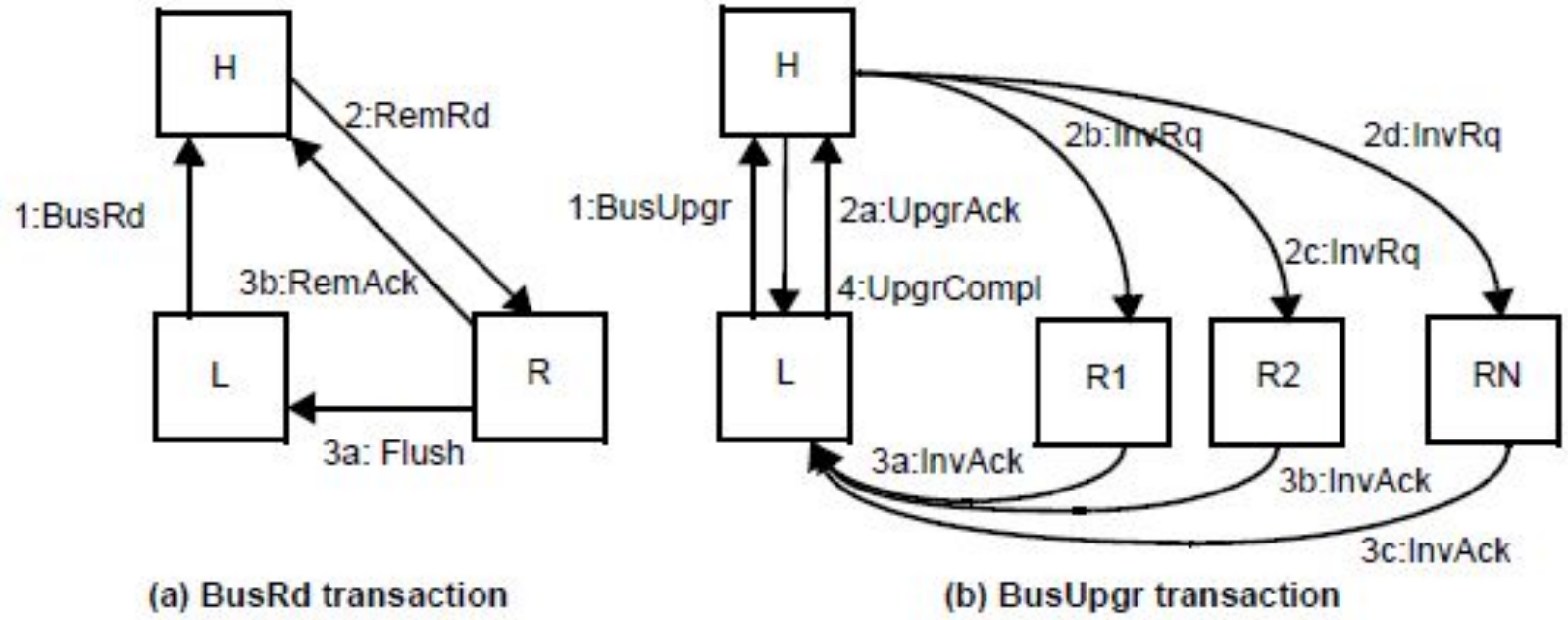


Figure 5.20. Coherence transactions in the presence-flag vector protocol. (a) BusRd on a clean miss; (b) BusRd on a dirty miss; (c) BusUpgr with single copy at home; (d) BusUpgr with multiple copies.

REDUCING LATENCIES IN DIRECTORY PROTOCOLS



Baseline directory protocol invokes four hops (in worst case) on a remote miss. Three-hop protocol is also possible:

1. request is sent to home (H)
2. home (H) redirects the request to remote(s) (R)
3. remote (R) responds to local (L) and notifies home (e.g. BusRd), or
4. local (L) notifies home (off the critical access path) (e.g. BusUpgr)

proposed by Stanford DASH Multiprocessor (late 1980s)

MEMORY REQUIREMENTS OF DIRECTORY PROTOCOLS

Memory requirement of a Presence-Flag Vector protocol

- **N** processors (nodes)
- **M** memory blocks per node
- **B** block size

size of directory = M x $\mathbf{N}^2$ (one directory at each processor)

- Directory scales with the square of number of processors; a scalability concern!
- **Mem. overhead:** size(dir)/(size(memory) + size(dir)) = N / (N+B)

Eg: 128 nodes, block size 16 bytes. What is the overhead?

Mem. overhead = 128 / (128 + 128) = 50%!

REDUCING MEMORY REQUIREMENTS OF DIRECTORY PROTOCOLS

Option 1: limited pointer protocol

- maintain *i* pointers (each log **N** bits) instead of **N** as in presence flag vectors
- if pointer overflows, resort to broadcast
- good if sharing is limited

Example: Mem. overhead for limited pointer scheme:

$$M \times N \times i \log N / (M \times N \times B + M \times N \times i \log n) = i \log N / (B + i \log N)$$

Option 2: coarse vector scheme

- presence flags identify groups of processors rather than individual nodes.
- Eg, each bit can identify clusters of four processors. Within the cluster, a broadcast protocol can be used.

Hybrid Scheme

- Coarse-grain vector can be combined with limited pointer scheme
  - start by using limited pointer scheme. If overflow, switch to coarse vector scheme
  - similar option is used in SGI Origin 2000
- E.g. 32-bit vector, 128 nodes
  - $\log_2(128) = 7$ bits, can hold four pointers (4x7 = 28 bits)
  - when fifth pointer is about to be allocated, switch to coarse scheme
  - 128 nodes / 32 bits = 4 nodes per bit, i.e can track 32 clusters of 4 nodes

OTHER SCALABLE PROTOCOLS

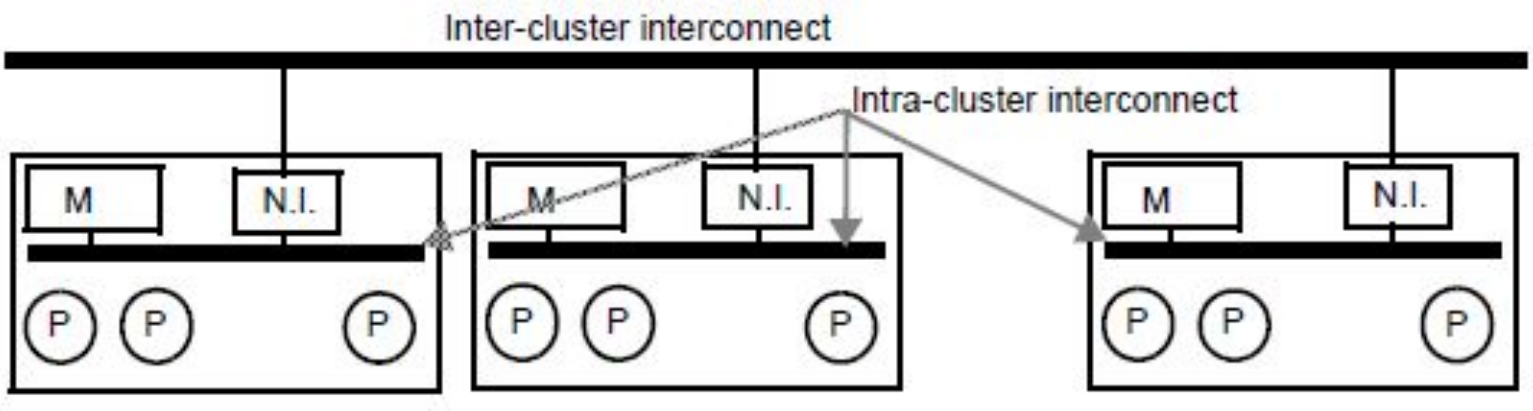
Alternative:

- instead of memory, make number of entries proportional to cache size

Directory cache

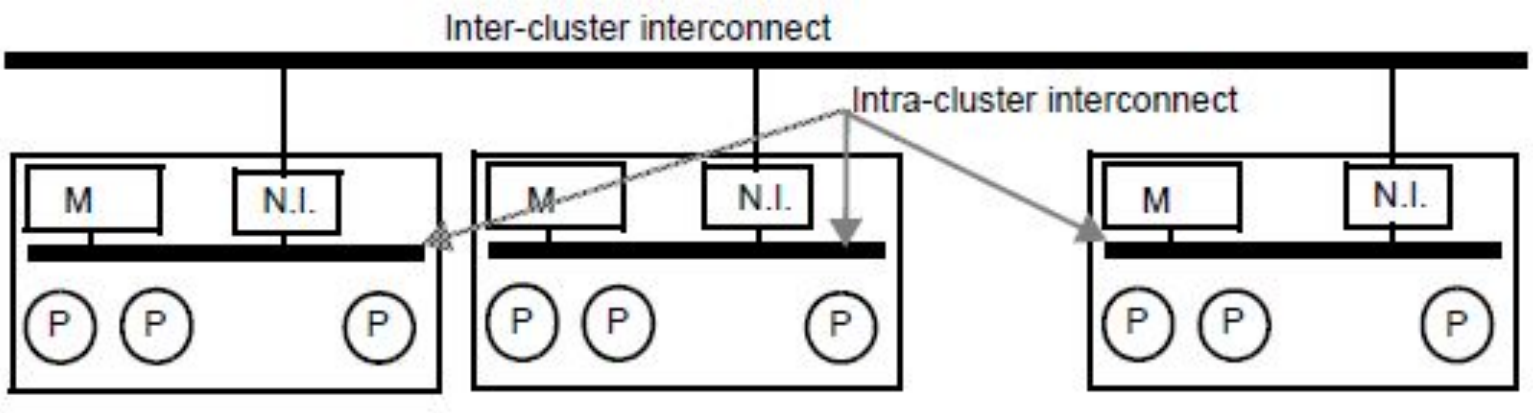
- Most blocks are private or never used at all! Allocate directory entry only when a memory block is cached in a new node
- Directory cache overhead is proportional to cache size instead of main memory size (leverages locality)
- When an entry is replaced, remotes need to be invalidated

HIERARCHICAL SYSTEMS



- Instead of flat configuration, form clusters with multiple processors in a hierarchical organization
- Requests are first serviced intra-cluster. If not possible, then the inter-cluster scheme is used.
- Renewed interest due to chip-multiprocessors

HIERARCHICAL SYSTEMS



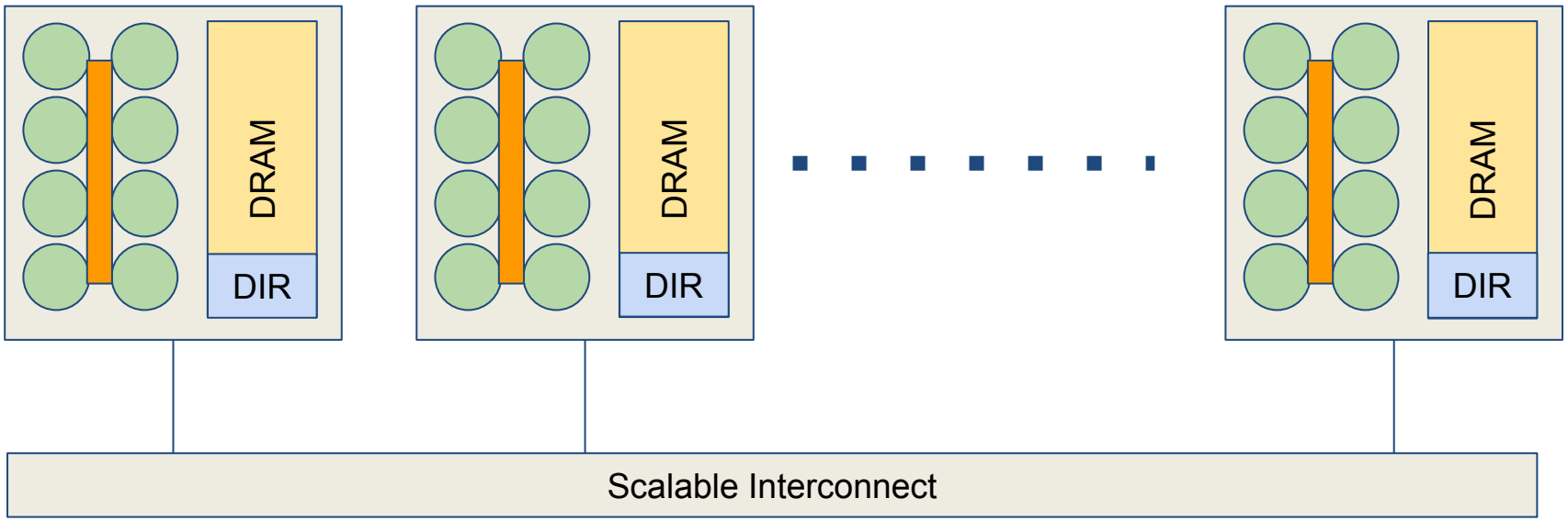
Coherence options:

- **intra-cluster coherence**: snoopy/directory
- **inter-cluster coherence**: snoopy/directory
- CMPs with large inclusive LLCs can filter external snoops
- intra-CMP coherence via bit vector (PFV) directory held directly in inclusive LLC blocks is popular option

tradeoffs affect memory overhead and performance to maintain coherence

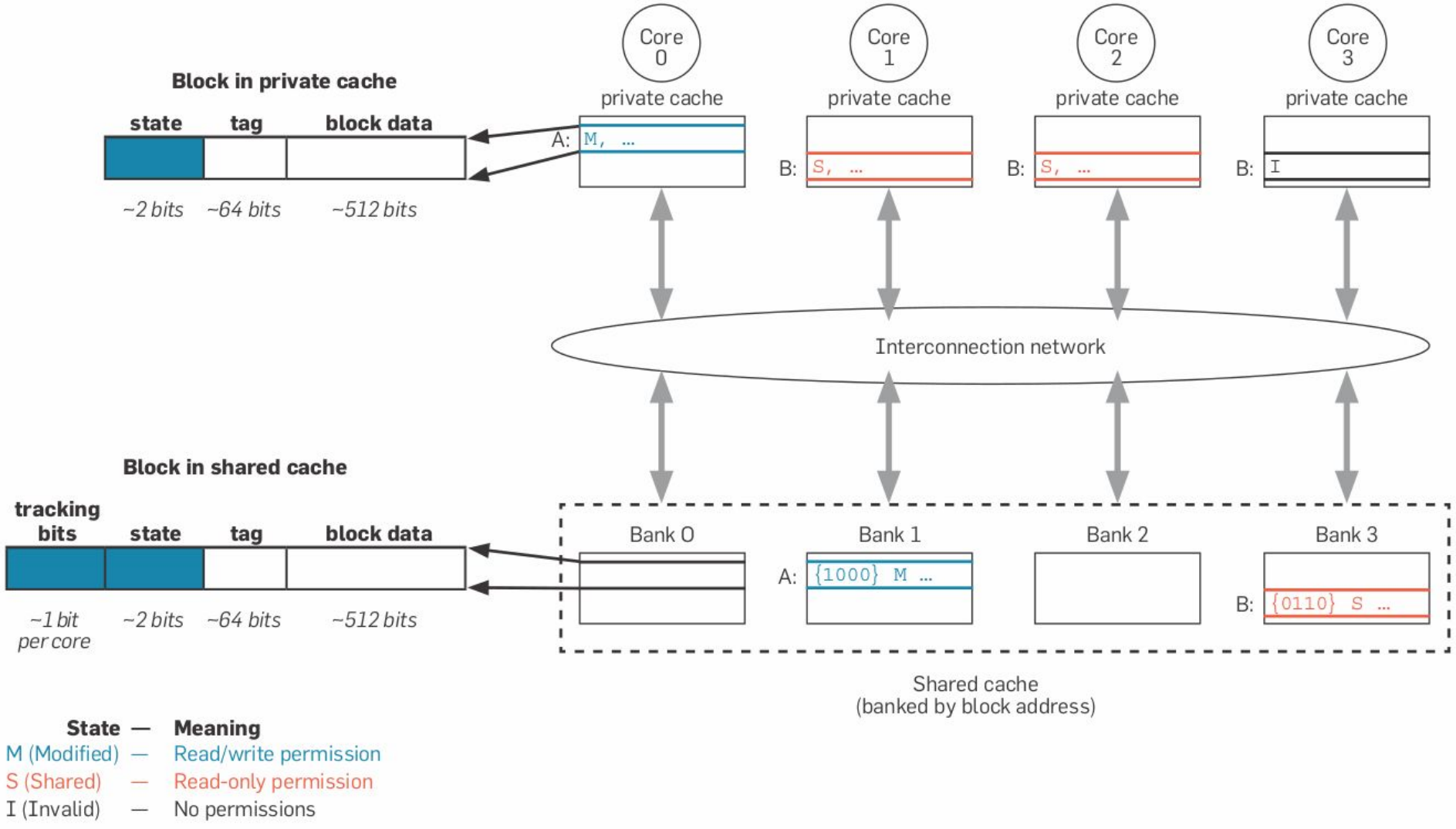
Example

- 16-byte memory blocks, 128 node system with directory based protocol
- traditional directory overhead is 128 / (128 + 128) = 50%
- What is the memory overhead if:
  - 8 processors per cluster,
  - intra-cluster coherence uses snoopy protocol
  - inter-cluster coherence uses directory protocol?



LLC extension for coherence

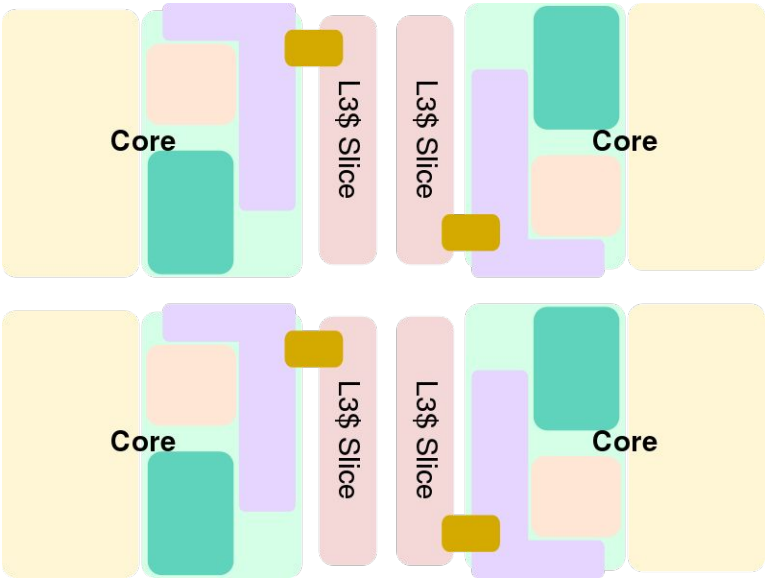
System model; additions for coherence are shaded.

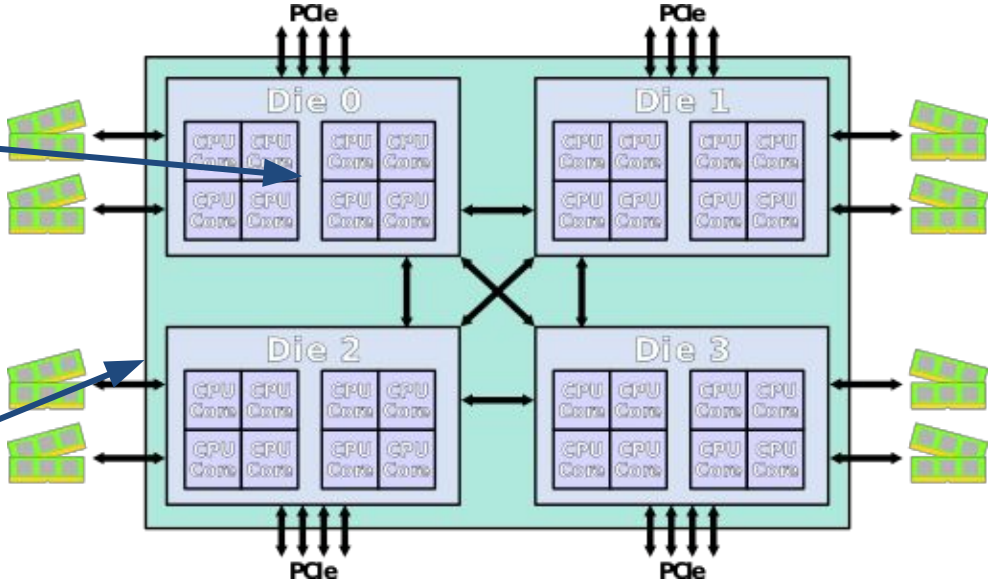


Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin. 2012. Why on-chip cache coherence is here to stay. Commun. ACM 55, 7 (July 2012), 78–89. DOI:https://doi.org/10.1145/2209249.2209269

Putting it all together: Zen

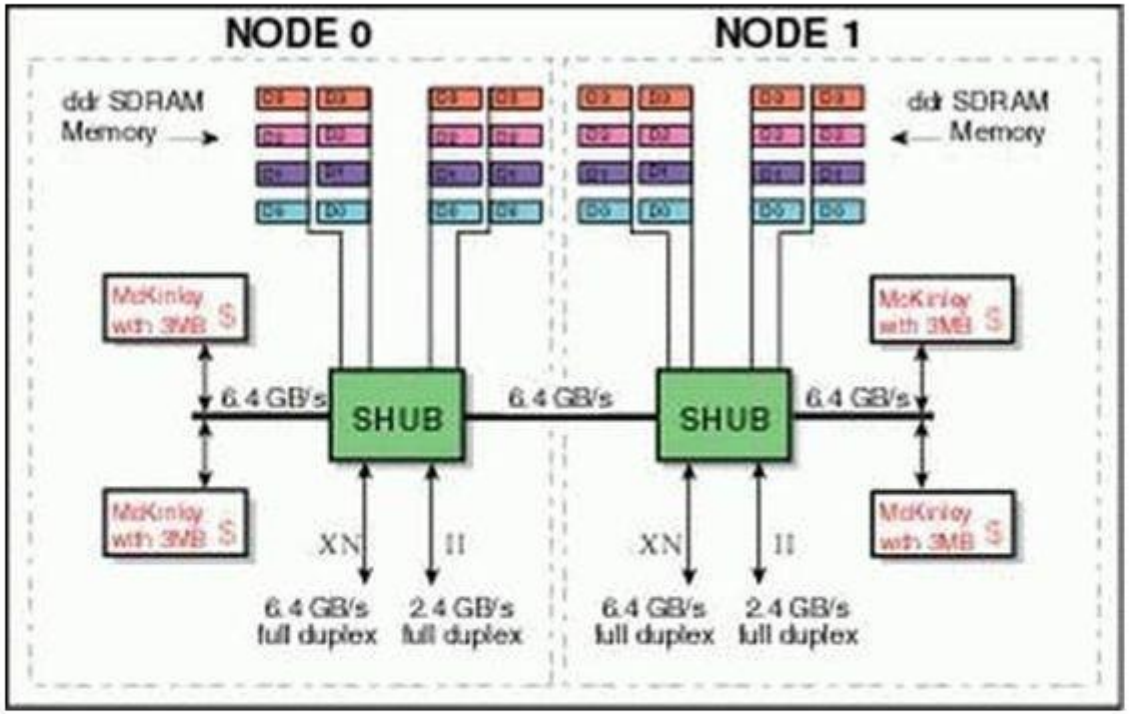
- Zen organized in groups of cores called a **CPU Complex (CCX)**.
- Each CCX consists of four cores connected to an L3.
- L3 cache is an 8 MiB 16-way set associative victim cache and is mostly exclusive of the L2
- The L3 cache is made of four slices (providing 2 MiB L3 slice/core) organized by low-order address interleaved.
- First fill up the L2 and and spillover to L3
- There may be one or more CCXs joined together.
- The separate CCX communicate with each other via **Infinity Fabric (IF)**
- Multiprocessors can also be built via IF (Global Memory Interconnect)
- Distributed MOESI cache coherence Directory (no extra details published...)





SGI NUMALINK

SGI Altix 3000



"Best of Show"
–LinuxWorld 2003

- A node contains up to 4 Itanium 2 processors and 32GB of memory
- Network is SGI's NUMAlink, the NUMAflex interconnect technology.
- Uses a mixture of snoopy and directory-based coherence
- Up to 512 processors that are cache coherent (global address space is possible for larger machines)

how much can we scale? numalink latencies are not negligible (~µs)
larger systems typically use distributed memory with message passing

SUMMARY

- Classification of misses
- Scalable cache coherence protocols
- Hierarchical Systems