Scalable Cache Coherent Systems

- Scalable distributed shared memory machines Assumptions:
  - Processor-Cache-Memory nodes connected by scalable network.
  - Distributed shared physical address space.
  - Communication assist must interpret network transactions, forming shared address space.
- For a system with shared physical address space:
  - A cache miss must be satisfied transparently from local or remote memory depending on address.
  - By its normal operation, cache replicates data locally resulting in a potential cache coherence problem between local and remote copies of data.
  - A coherency solution must be in place for correct operation.
- Standard snoopy protocols studied earlier do not apply for lack of a bus or a broadcast medium to snoop on.
- For this type of system to be scalable, in addition to latency and bandwidth scalability, the cache coherence protocol or solution used must also scale as well.

Functionality Expected In A Cache Coherent System

- **Provide a set of states, a state transition diagram, and actions representing the cache coherence protocol used.**
- **Manage coherence protocol:**
  - **(0) Determine when to invoke the coherence protocol**
  - **(a) Find source of information about state of cache line in other caches**
    - **Whether need to communicate with other cached copies**
  - **(b) Find out the location or locations of other copies if any.**
  - **(c) Communicate with those copies (invalidate/update).**
- **(0) is done the same way on all cache coherent systems:**
  - **State of the local cache 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-Based Coherence

- All of (a), (b), (c) done through broadcast on the bus:
  - Faulting processor sends out a “search”.
  - Others respond to the search probe and take necessary action.
- This approach could be done in a scalable network too:
  - Broadcast to all processors, and let them respond.
  - Conceptually simple, but broadcast doesn’t scale with p:
    - Bus bandwidth doesn’t scale.
    - On a scalable network (e.g MINs) , every fault may lead to at least p network transactions.

Scalable Cache Coherence

- A scalable cache coherence approach may have similar cache line states and state transition diagrams as in bus-based coherence protocols.
- However, different additional mechanisms other than broadcasting must be devised to manage the coherence protocol.
- Two possible approaches:
  - Approach #1: Hierarchical Snooping.
  - Approach #2: Directory-based cache coherence.
  - Approach #3: A combination of the above two approaches.

Approach #1: Hierarchical Snooping

- **Extend snooping approach: A hierarchy of broadcast media:**
  - **Tree of buses or rings (KSR-1).**
  - **Processors are in the bus- or ring-based multiprocessors at the leaves.**
  - **Parents and children connected by two-way snoopy interfaces:**
    - **Snoop both buses and propagate relevant transactions.**
  - **Main memory may be centralized at root or distributed among leaves.**
- **Issues (a) - (c) handled similarly to bus, but not full broadcast.**
  - **Faulting processor sends out "search" bus transaction on its bus.**
  - **Propagates up and down hierarchy based on snoop results.**
- **Problems:**
  - **High latency: multiple levels, and snoop/lookup at every level.**
  - **Bandwidth bottleneck at root.**
- **This approach has, for the most part, been abandoned.**

Hierarchical Snoopy Cache Coherence

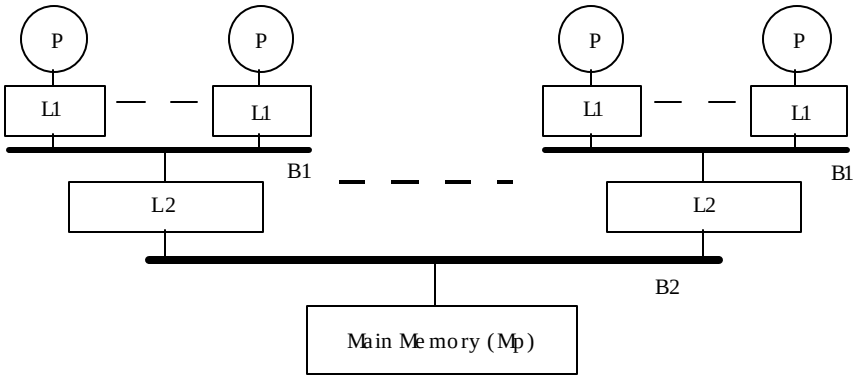
Simplest way: hierarchy of buses; snoopy coherence at each level.

- **or rings.**

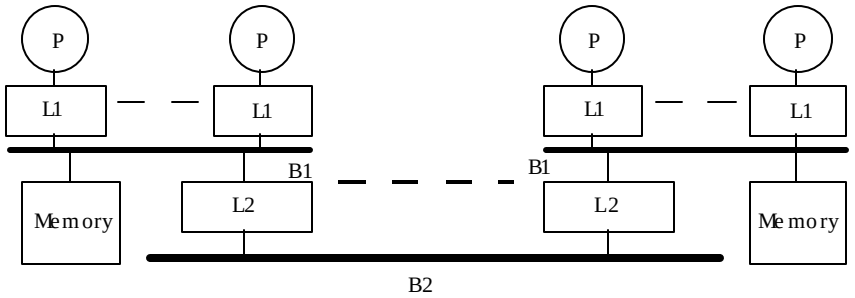
• Consider buses. Two possibilities:

(a) All main memory at the global (B2) bus.

(b) Main memory distributed among the clusters.

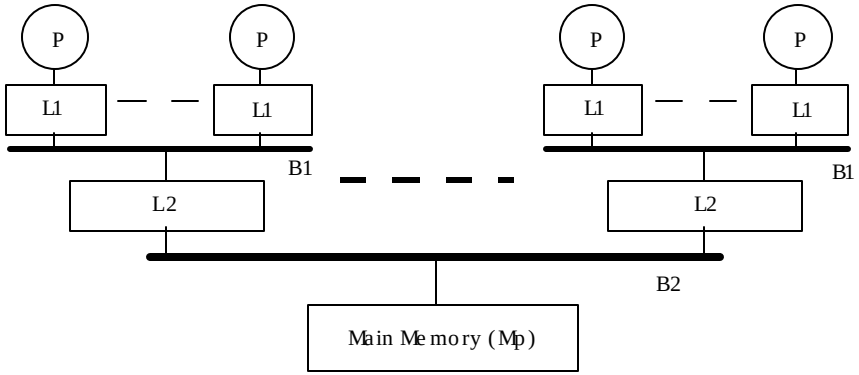


(a)



(b)

Bus Hierarchies with Centralized Memory



B1 follows standard snoopy protocol.

Need a monitor per B1 bus:

- **Decides what transactions to pass back and forth between buses.**
- **Acts as a filter to reduce bandwidth needs.**

Use L2 cache:

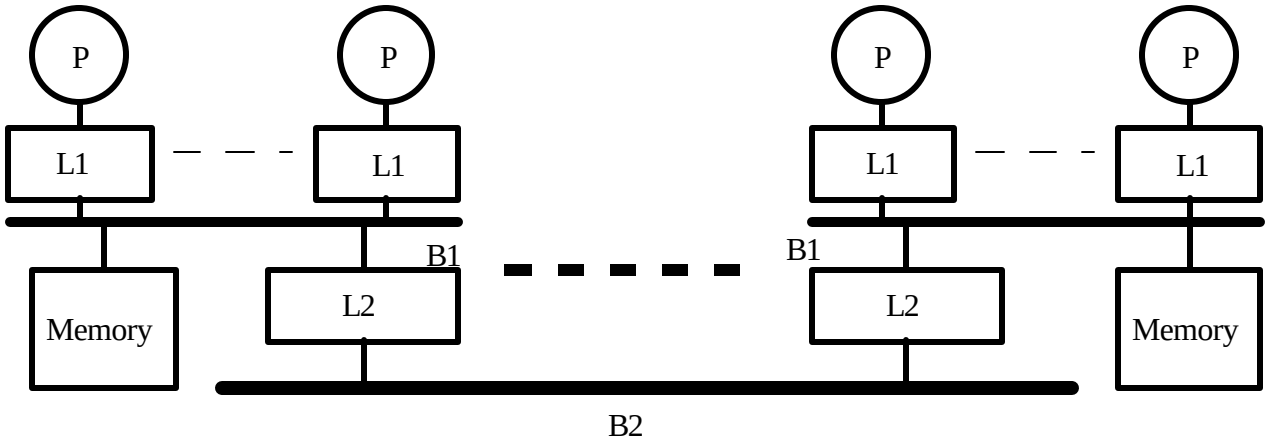
- **Much larger than L1 caches (set associative). Must maintain inclusion.**
- **Has dirty-but-stale bit per line.**
- **L2 cache can be DRAM based, since fewer references get to it.**

Bus Hierarchies with Centralized Memory
Advantages and Disadvantages

- Advantages:
  - Simple extension of bus-based scheme.
  - Misses to main memory require single traversal to root of hierarchy.
  - Placement of shared data is not an issue.

- Disadvantages:
  - Misses to local data (e.g., stack) also traverse hierarchy.
    - **Higher traffic and latency.**
  - Memory at global bus must be highly interleaved for bandwidth.

Bus Hierarchies with Distributed Memory

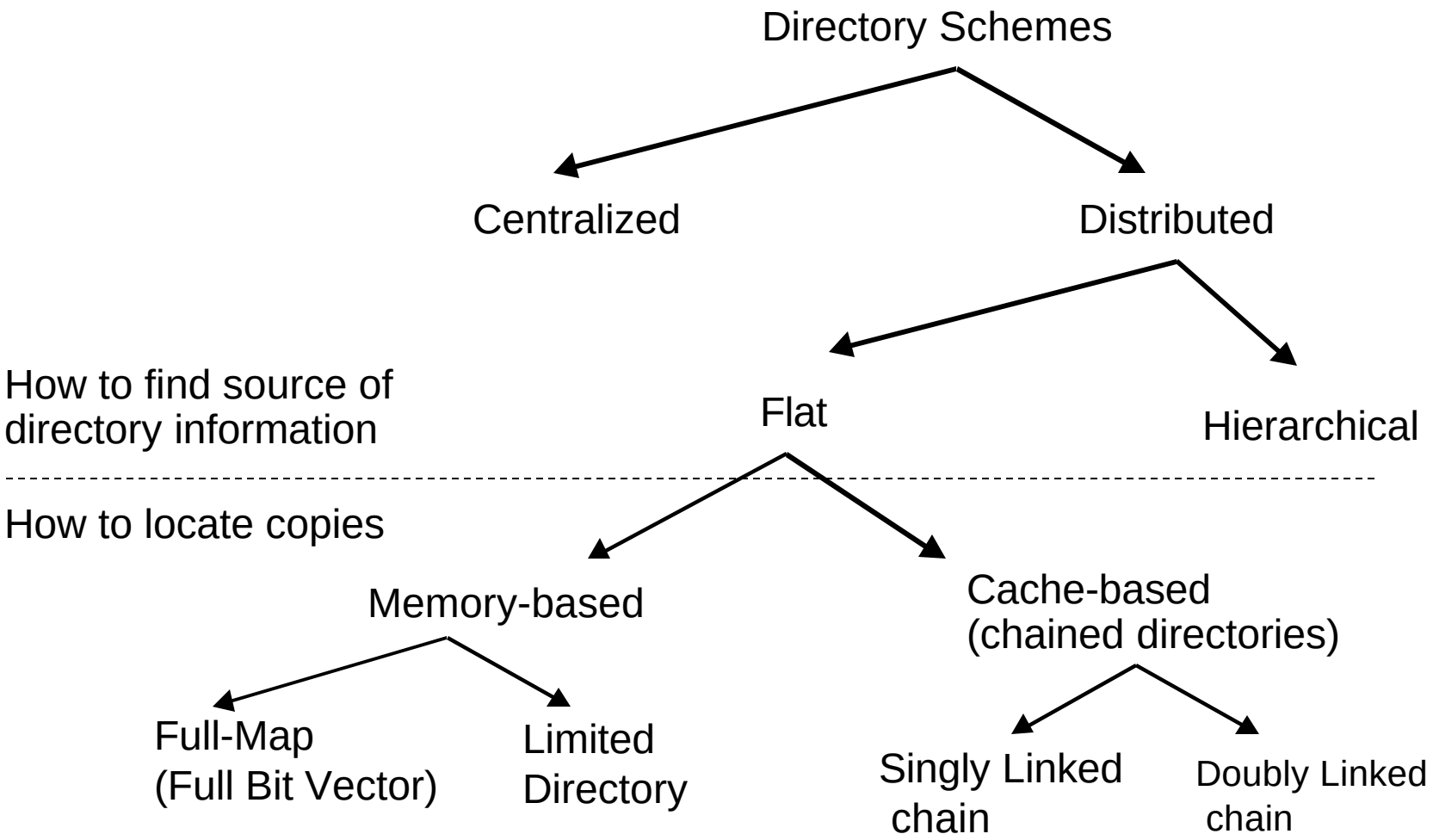


- **Main memory distributed among clusters.**
  - **Cluster is a full-fledged bus-based machine, memory and all.**
  - **Automatic scaling of memory (each cluster brings some with it).**
  - **Good placement can reduce global bus traffic and latency.**
    - **But latency to far-away memory is larger.**

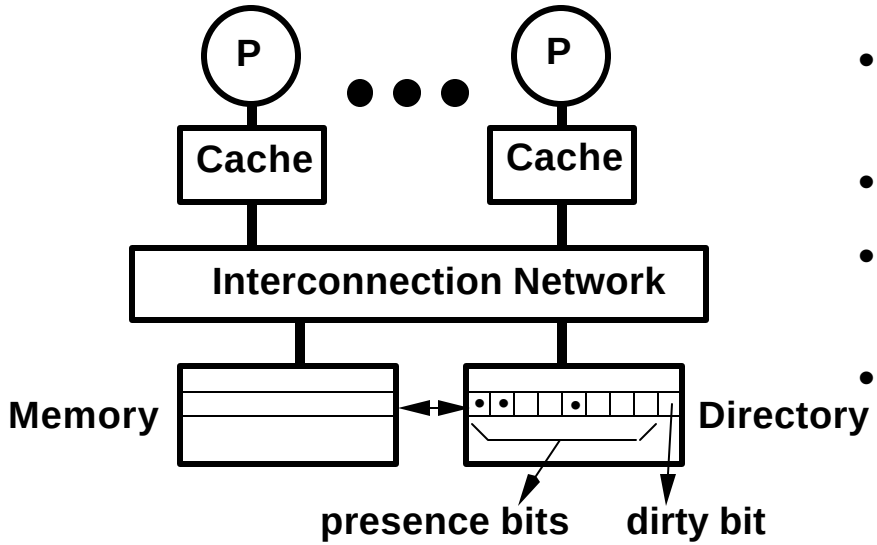
Scalable Approach #2: Directories

- A directory is composed of a number of directory entries.
- Every memory block has an associated directory entry:
  - Keeps track of the nodes or processors that have cached copies of the memory block 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 nodes that have copies is through *network transactions.*
- Many alternatives exist for organizing directory information.

Organizing Directories



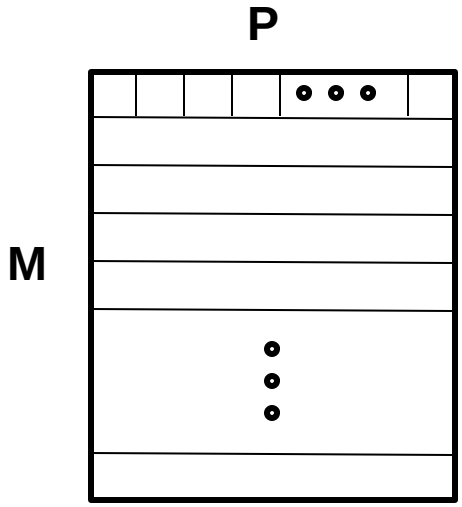
Basic Operation of Centralized Directory



- Both memory and directory are centralized.
- P processors.
- With each cache-block in memory: P presence-bits p[i], 1 dirty-bit.
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit.

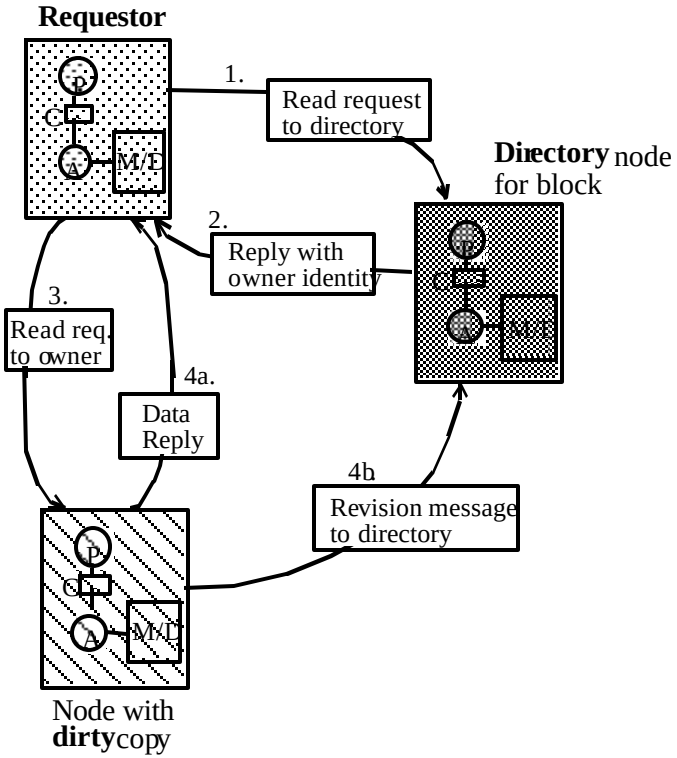
- **Read from main memory (read miss) by processor i:**
  - **If dirty-bit OFF then { read from main memory; turn p[i] ON; }**
  - **if dirty-bit ON then { recall line from dirty proc j (cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to i;}**
- **Write miss to main memory by processor i:**
  - **If dirty-bit OFF then { supply data to i; send invalidations to all caches that have the block; turn dirty-bit ON; turn p[i] ON; ... }**
  - **if dirty-bit ON then { recall line from dirty proc (with p[j] on); update memory; block state on proc j invalid ; turn p[i] ON; supply recalled data to i;}**

Distributed, Flat, Memory-based Schemes

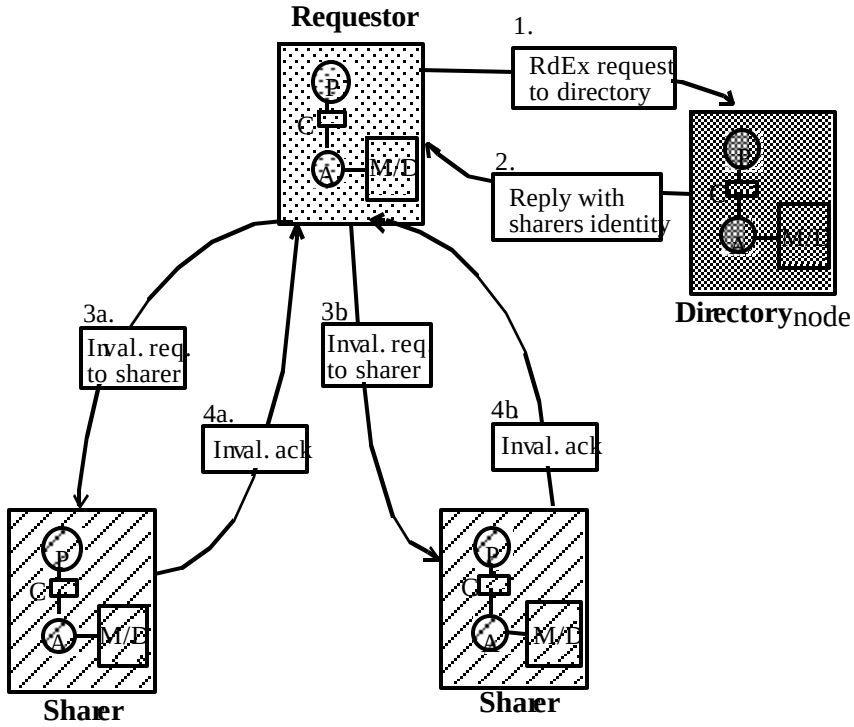


- All info about copies of a memory blocks co-located with block itself at home node (directory node of block).
  - Works just like centralized scheme, except distributed.
- Scaling of performance characteristics:
  - Traffic on a write: proportional to number of sharers.
  - Latency a write: Can issue invalidations to sharers in parallel.
- Scaling of storage overhead:
  - Simplest representation: Full-Map (*full bit vector)*, i.e. one presence bit per node: P presence bits, 1 dirty bit per block directory entry.
  - Storage overhead doesn't scale well with P; a 64-byte cache line implies:
    - 64 nodes: 65/(64 x 8) = 12.7% overhead.
    - 256 nodes: 50% overhead.; 1024 nodes: 200% overhead.
  - For M memory blocks in memory, storage overhead is proportional to P*M

Basic Operation of Distributed, Flat, Memory-based Directory



(a) Read miss to a block in dirty state



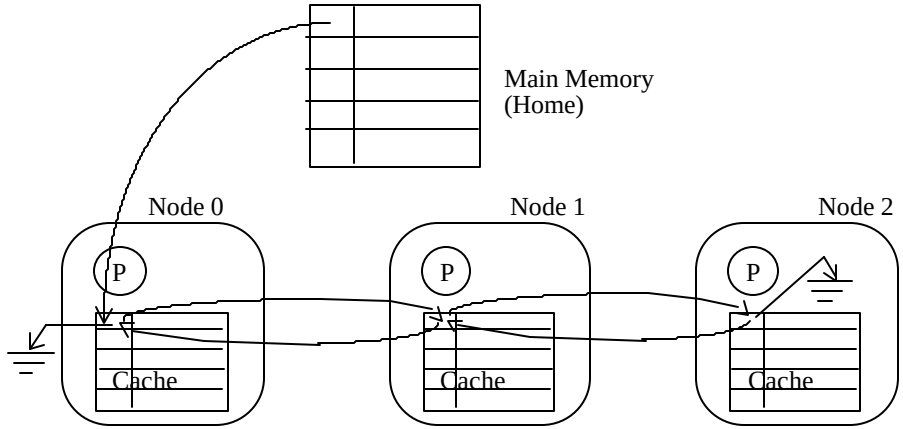
(b) Write miss to a block with two sharers

Reducing Storage Overhead of Distributed Memory-based Directories

- **Optimizations for full bit vector schemes:**
  - **Increase cache block size (reduces storage overhead proportionally)**
  - **Use multiprocessor (SMP) nodes (one presence bit per multiprocessor node, not per processor)**
  - **still scales as P*M, but not a problem for all but very large machines**
    - **256-processors, 4 per node, 128 Byte block : 6.25% overhead.**
- **Limited Directories: Addressing entry width P**
  - **Observation: most blocks cached by only few nodes.**
  - **Don't have a bit per node, but directory entry contains a few pointers to sharing nodes (each pointer has $\log_2 P$ bits, e.g P=1024 => 10 bit pointers).**
  - **Sharing patterns indicate a few pointers should suffice (five or so)**
  - **Need an overflow strategy when there are more sharers.**
  - **Storage requirements: O(M $\log_2 P$).**
- **Reducing "height": addressing the M term**
  - **Observation: number of memory blocks >> number of cache blocks**
  - **Most directory entries are useless at any given time**
  - **Organize directory as a cache, rather than having one entry per memory block.**

Flat, Cache-based Schemes

- **How they work:**
  - **Memory block at home node only holds pointer to rest of directory info.**
  - **Distributed linked list of copies, weaves through caches:**
    - **Cache tag has pointer, points to next cache with a copy.**
  - **On read, add yourself to head of the list (comm. needed).**
  - **On write, propagate chain of invalidations down the list.**

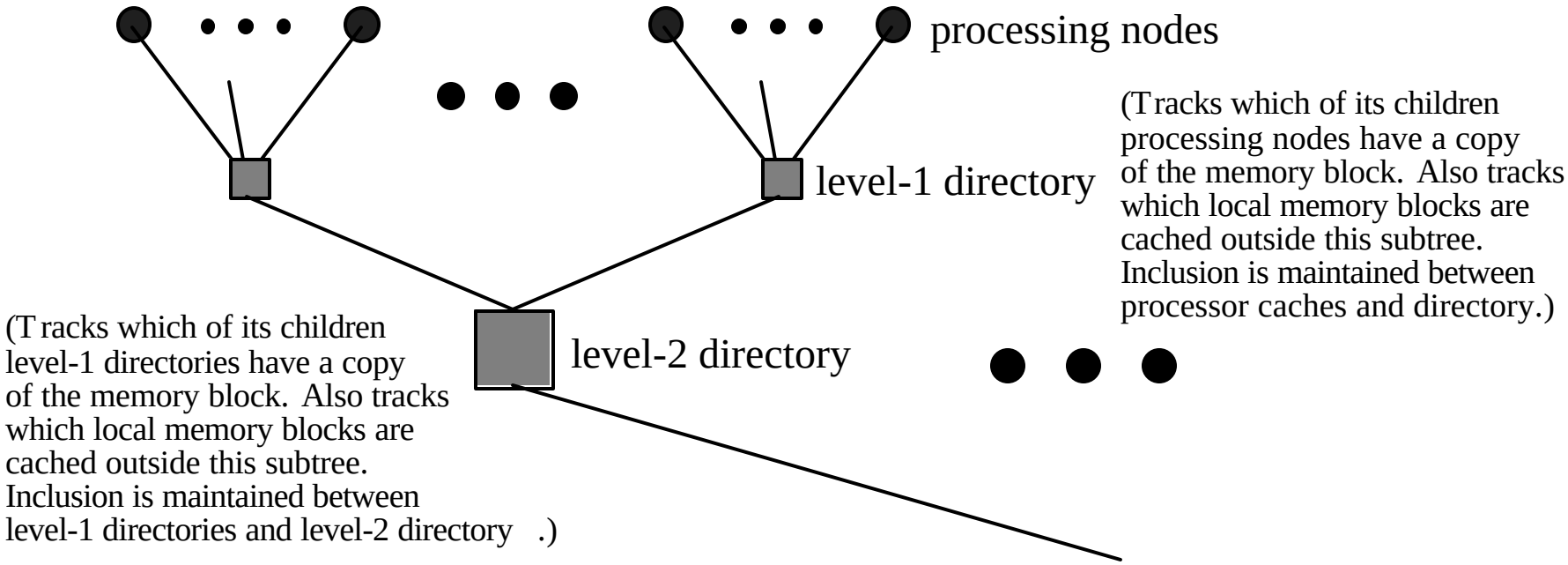


- **Utilized in Scalable Coherent Interface (SCI) IEEE Standard:**
  - **Uses a doubly-linked list.**

Scaling Properties of Cache-based Schemes

- Traffic on write: proportional to number of sharers.
- Latency on write: proportional to number of sharers.
  - Don't know identity of next sharer until reach current one
  - also assist processing at each node along the way.
  - (even reads involve more than one other assist: home and first sharer on list)
- Storage overhead: quite good scaling along both axes
  - Only one head pointer per memory block
    - rest of storage overhead is proportional to cache size.
- Other properties:
  - Good: mature, IEEE Standard, fairness.
  - Bad: complex.

How Hierarchical Directories Work



- **Directory is a hierarchical data structure:**
  - **Leaves are processing nodes, internal nodes just directories.**
  - **Logical hierarchy, not necessarily physical (can be embedded in general network).**

How to Find Directory Information

- Centralized memory and directory - easy: go to it
  - But not scalable.
- Distributed memory and directory
  - Flat schemes:
    - Directory distributed with memory: at the cache block *home node.*
    - Location based on address: network transaction sent directly to home.
  - Hierarchical schemes:
    - Directory organized as a hierarchical data structure.
    - Leaves are processing nodes, internal nodes have only directory state.
    - Node's directory entry for a block says whether each subtree caches the block
    - To find directory info, send "search" message up to parent
      - Routes itself through directory lookups.
    - Similar to hierarchical snooping, but point-to-point messages are sent between children and parents.

How Is Location of Copies Stored?

- **Hierarchical Schemes:**
  - **Through the hierarchy.**
  - **Each directory has presence bits for its children (subtrees), and dirty bit.**
- **Flat Schemes:**
  - **Varies a lot (memory-based vs. Cache-based).**
  - **Different storage overheads and performance characteristics.**
  - **Memory-based schemes:**
    - **Info about copies stored all at the home with the memory block.**
    - **Examples: Dash, Alewife , SGI Origin, Flash.**
  - **Cache-based schemes:**
    - **Info about copies distributed among copies themselves.**
      - **Each copy points to next.**
    - **Example: Scalable Coherent Interface (SCI, an IEEE standard).**

Summary of Directory Organizations

Flat Schemes:

- Issue (a): finding source of directory data:
  - Go to home, based on address.
- Issue (b): finding out where the copies are.
  - Memory-based: all info is in directory at home.
  - Cache-based: home has pointer to first element of distributed linked list.
- Issue (c): communicating with those copies.
  - memory-based: point-to-point messages (perhaps coarser on overflow).
    - Can be multicast or overlapped.
  - Cache-based: part of point-to-point linked list traversal to find them.
    - serialized.

Hierarchical Schemes:

- All three issues through sending messages up and down tree.
- No single explicit list of sharers.
- Only direct communication is between parents and children.

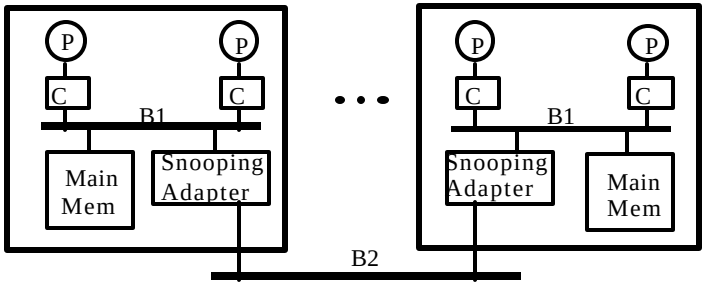
Summary of Directory Approaches

- Directories offer scalable coherence on general networks.
  - No need for broadcast media.
- Many possibilities for organizing directories and managing protocols.
- Hierarchical directories not used much.
  - High latency, many network transactions, and bandwidth bottleneck at root.
- Both memory-based and cache-based distributed flat schemes are alive:
  - For memory-based, full bit vector suffices for moderate scale.
    - Measured in nodes visible to directory protocol, not processors.

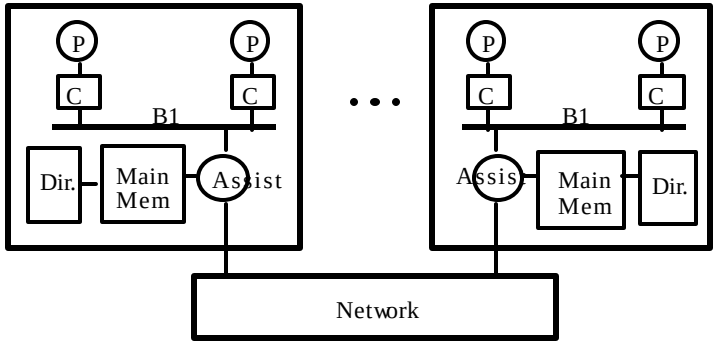
Approach #3: A Popular Middle Ground Two-level "Hierarchy"

- Individual nodes are multiprocessors, connected non-hierarchically.
  - e.g. mesh of SMPs.
- Coherence across nodes is directory-based.
  - Directory keeps track of nodes, not individual processors.
- Coherence within nodes is snooping or directory.
  - Orthogonal, but needs a good interface of functionality.
- Examples:
  - Convex Exemplar: directory-directory.
  - Sequent, Data General, HAL: directory-snoopy.

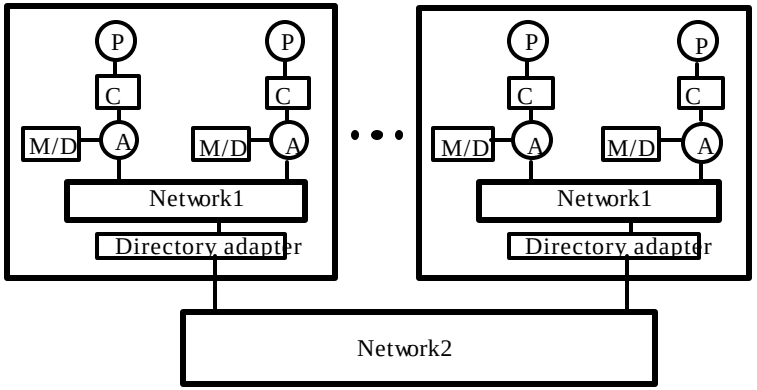
Example Two-level Hierarchies



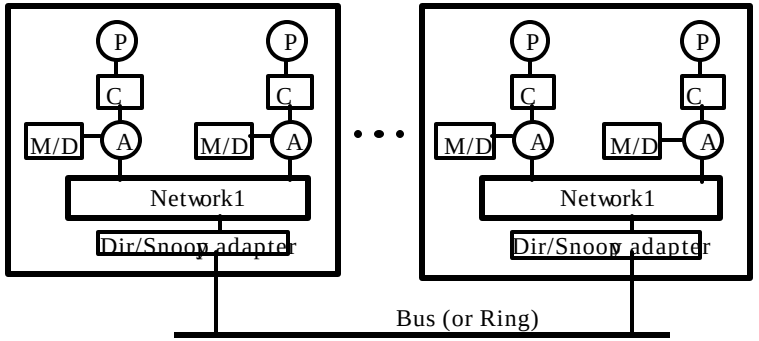
(a) Snooping-snooping



(b) Snooping-directory



(c) Directory-directory



(d) Directory-snooping

Advantages of Multiprocessor Nodes

- **Potential for cost and performance advantages:**
  - **Amortization of node fixed costs over multiple processors**
    - **Applies even if processors simply packaged together but not coherent.**
  - **Can use commodity SMPs.**
  - **Less nodes for directory to keep track of.**
  - **Much communication may be contained within node (cheaper).**
  - **Nodes prefetch data for each other (fewer "remote" misses).**
  - **Combining of requests (like hierarchical, only two-level).**
  - **Can even share caches (overlapping of working sets).**
  - **Benefits depend on sharing pattern (and mapping):**
    - **Good for widely read-shared: e.g. tree data in Barnes-Hut**
    - **Good for nearest-neighbor, if properly mapped**
    - **Not so good for all-to-all communication.**

Disadvantages of Coherent MP Nodes

- Bandwidth shared among nodes.
- Bus increases latency to local memory.
- With local node coherence in place, a CPU typically must wait for local snoop results before sending remote requests.
- Snoopy bus at remote node increases delays there too, increasing latency and reducing bandwidth.
- Overall, may hurt performance if sharing patterns don't comply with system architecture.