Scalable Multiprocessors

[§10.1] A *scalable* system is one in which resources can be added to the system without reaching a hard limit.

What does scalability mean?

- Avoids inherent design limits on resources.
- Bandwidth increases with # of processors *p*.
- Latency does not.
- Cost increases slowly with *p*.

Why doesn't a bus-based design scale?

- Physical constraints

- Protocol constraints

- Contention everywhere: bus, snooper, memory

Scalability and coherence

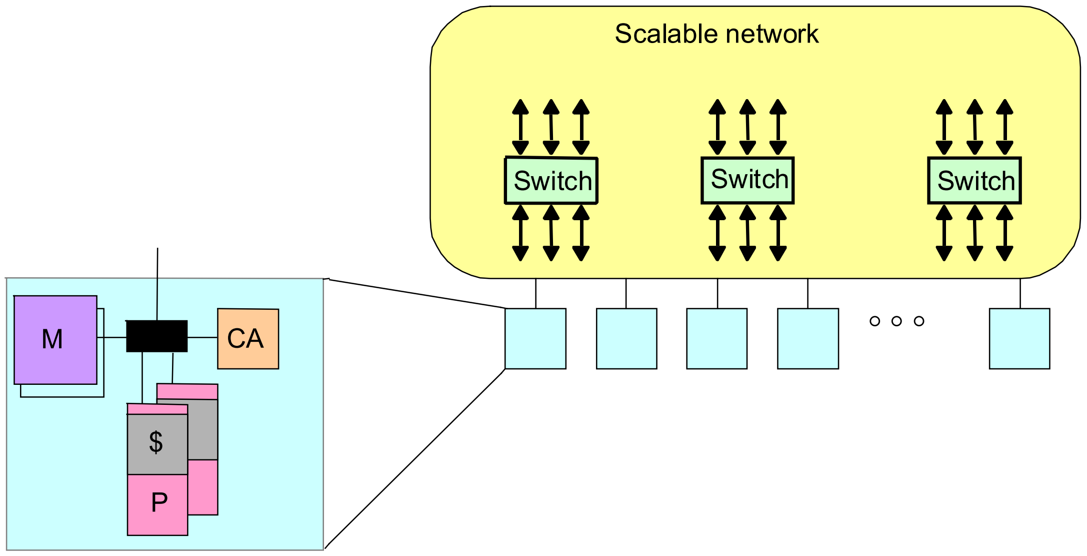
All of the cache-coherent systems we have talked about until now have had a bus.

Not only does the bus guarantee serialization of transactions; it also serves as a convenient *broadcast* mechanism to assure that each transaction is propagated to all other processors' caches.

How can cache coherence can be provided on a machine with physically distributed memory and no globally snoopable interconnect?

- To support a shared address space?
- To be able to satisfy a cache miss transparently from local or remote memory?

This means data is replicated widely. How can it be kept coherent?



Scalable distributed memory machines consist of P-C-M nodes connected by a network.

The *communication assist* interprets network transactions and forms the interface between the processor and the network.

A coherent system must do these things.

- Provide a set of states, a state-transition diagram, and actions.
- Manage the coherence protocol.
    - (0) Determine when to invoke the coherence protocol
    - (a) Find a source of information about the state of this block in other caches.
    - (b) Find out where the other copies are
    - (c) Communicate with those copies (invalidate/update)

(0) is done the same way on all systems

- The state of the line is maintained in the cache
- The protocol is invoked if an “access fault” occurs on the line.

The different approaches to scalable cache coherence are distinguished by their approach to (a), (b), and (c).

Bus-based coherence

In a bus-based coherence scheme, all of (a), (b), and (c) are done through broadcast on bus.

- The faulting processor sends out a "search."
- Other processors respond to the search probe and take necessary action.

We *could* do this in a scalable network too—broadcast to all processors, and let them respond. Why don't we?

Why not? On a scalable network, every fault leads to at least *p* network transactions.

		Interconnection	
		Bus	*Point-to-point*
Protocol	*Snoopy*	Least scalable	More scalable
	Directory	–	Most scalable

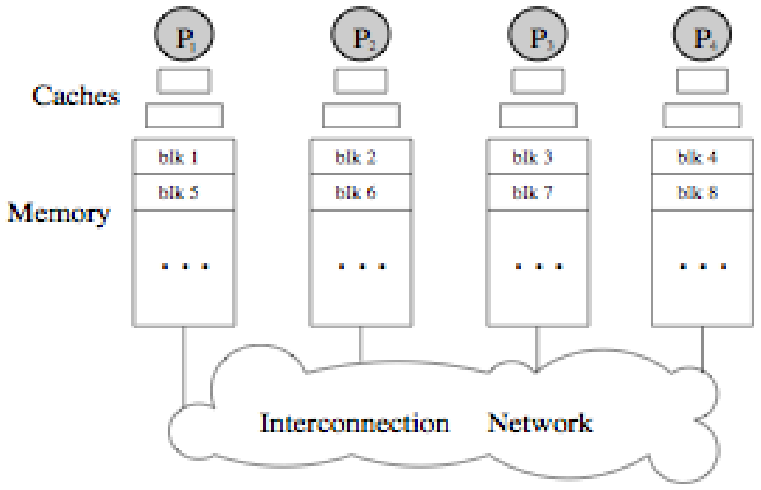
Directory-based protocol

- Instead of broadcasting to find out who has the block, *keep track* of copies in the directory.
- Invalidation requests must be sent (individually) to all sharers; can you explain why this doesn't render the protocol too slow?

- Used with distributed shared memory (DSM) multiprocessors
- Can scale to tens or hundreds of processors.

How to map memory on a DSM?

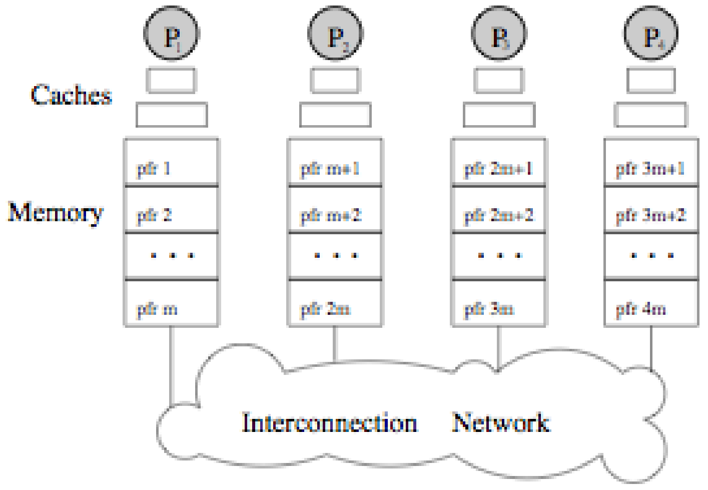
- Block interleaving?
  - distributes data around
  - hard to exploit spatial locality



- No interleaving?

  [pfr = page frame]

  Of course, the OS is responsible for placing pages in page frames.



- The OS must be involved in deciding where to allocate a page. Answer these questions …
- How are pages typically replaced on a uniprocessor?
- Why is the decision different on a multiprocessor?
- Why is "first touch" a sensible policy for many situations?
- Why is "first touch" grossly suboptimal for many parallel algorithms?

- What is an alternative allocation policy that often works well?

Handling misses in directory-based coherence

The basic idea of a directory-based approach is this.

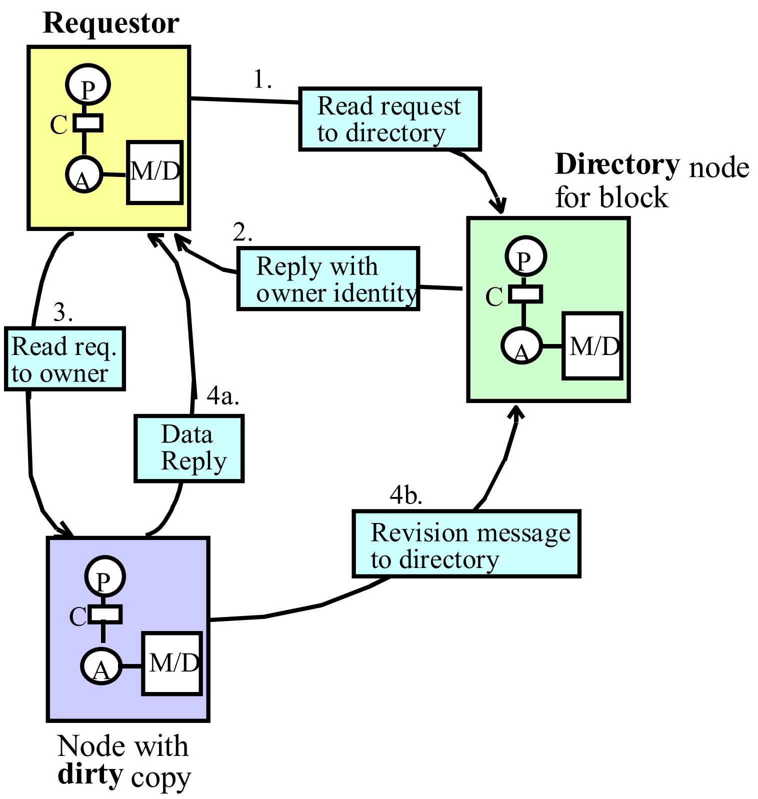
- Every memory block has associated directory information; it keeps track of copies of cached blocks and their states.
- On a miss, it finds the directory entry, looks it up, and communicates only with the nodes that have copies (if necessary).

In scalable networks, communication with the directory and with copies occurs through *network transactions*.

Let us assume that the directory is distributed, with each node holding directory information for the blocks it contains.

This node is called the *home node* for these blocks.

What happens on a read miss?

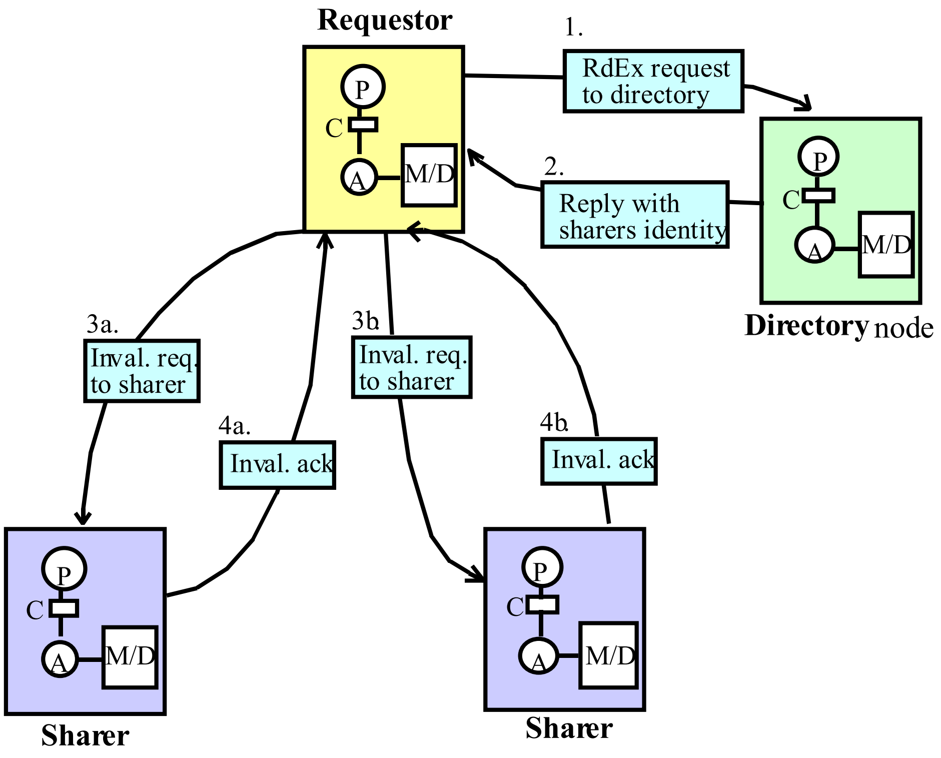


The requesting node sends a request transaction over the network to the home node.

The home node responds with the identity of the *owner*—the node that currently holds a valid copy of the block.

The requesting node then gets the data from the owner, and revises the directory entry accordingly.

On a write miss, the directory identifies copies of the block, and invali-dation or update messages may be sent to the copies.



Now, see if you can tell how many directory messages are needed in each of several cases.

One major difference from bus-based schemes is that we can't assume that a write has

What information will be held in the directory?

- There will be a dirty bit telling if the block is dirty in some cache.
- Not all state information (MESI, etc.) needs to be kept in the directory, only enough to determine what actions to take.

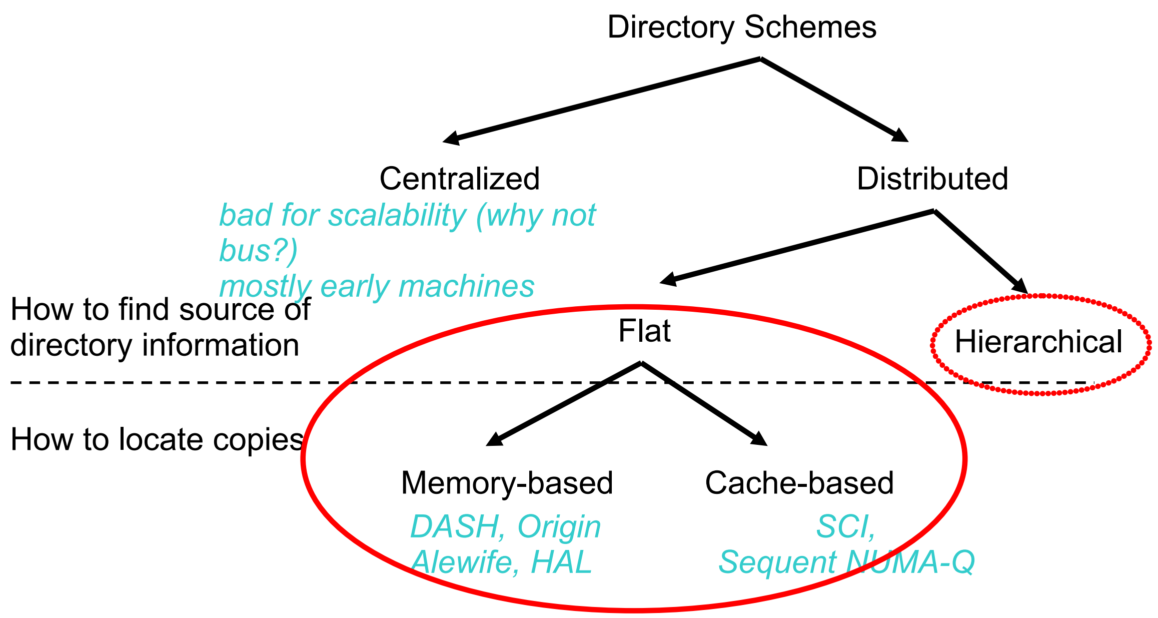
  Sometimes the state information in the directory will be out of date. Why?

  So, sometimes a directory will send a message to the cache that is no longer correct when it is received.

Flat vs. hierarchical directories

When a miss occurs, how do we find the directory information? There are two main alternatives.

- A *flat* directory scheme. Directory information is in a fixed place, usually at the *home* (where the memory is located).
  - On a miss, a transaction is sent to the home node.
- A *hierarchical* directory scheme. Directory information is organized as a tree, with the processing nodes at the leaves.
  - Each node keeps track of which, if any, of its (immediate) children have a copy of the block.
  - When a miss occurs, the directory information is found by traversing up the hierarchy level until the block is found (in the "appropriate state").
  - The state indicates, e.g., whether copies of the block exist outside the subtree of this directory.

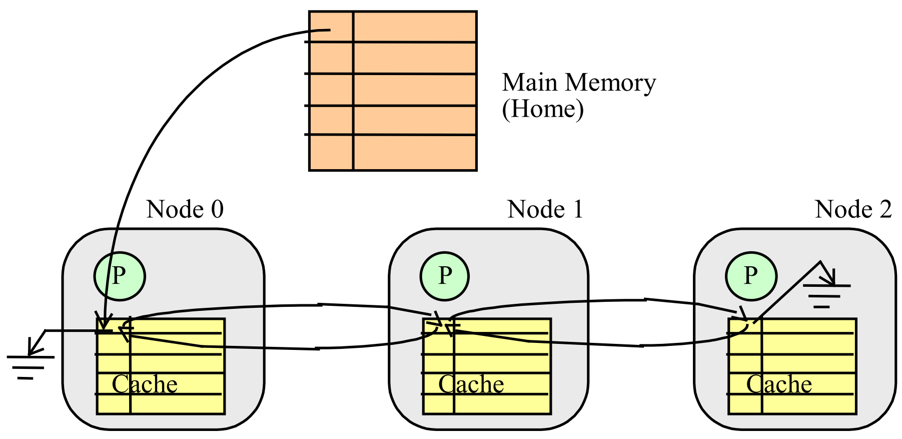


How do flat schemes store information about copies?

- *Memory-based schemes* store the information about all cached copies at the home node of the block. E.g., Dash, Alewife, SGI Origin.

- *Cache-based schemes* distribute information about copies among the copies themselves. E.g., IEEE SCI, Sequent NUMA-Q.
    - The home contains a pointer to one cached copy of the block.
    - Each copy contains the identity of the next node that has a copy of the block.

This means that the copies are located through network transactions.



When do hierarchical schemes outperform flat schemes?

Why might hierarchical schemes be slower than flat schemes?

Summary

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 explict list of sharers
- only direct communication is between parents and children

Distributing the directory

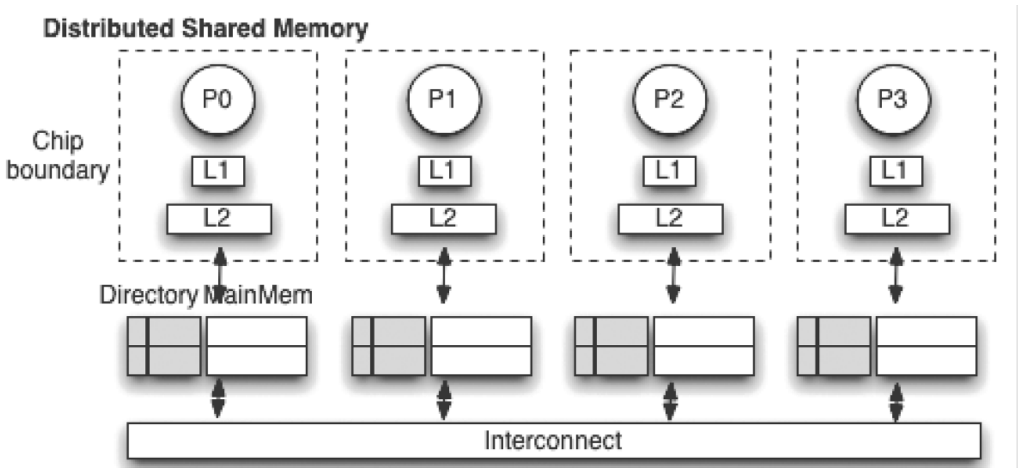
The directory needs to be distributed, but how many "pieces" should there be, and where should they be located?

Classical DSM

P-C-M nodes (p. 2, above) are connected to form a distributed shared memory system.

LL cache miss → request to directory determined by PFA of block

Directory is located at the same node as the block. Why?



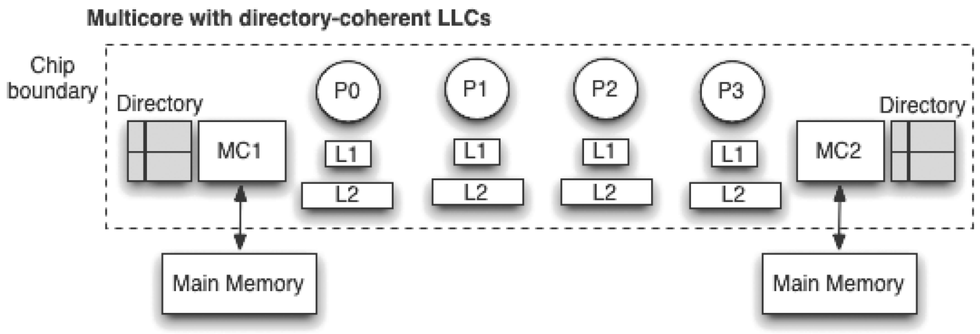
Multicore with coherent LLCs

Directory entries point to cache blocks, not main memory!

If the LLC misses, block can be fetched from another cache.

If it's not cached, then it needs to go through a memory controller (MC) to fetch it from main memory.

The number of memory controllers is limited by pin count, which may cause bottlenecks.



Multicore with coherent non-LLCs

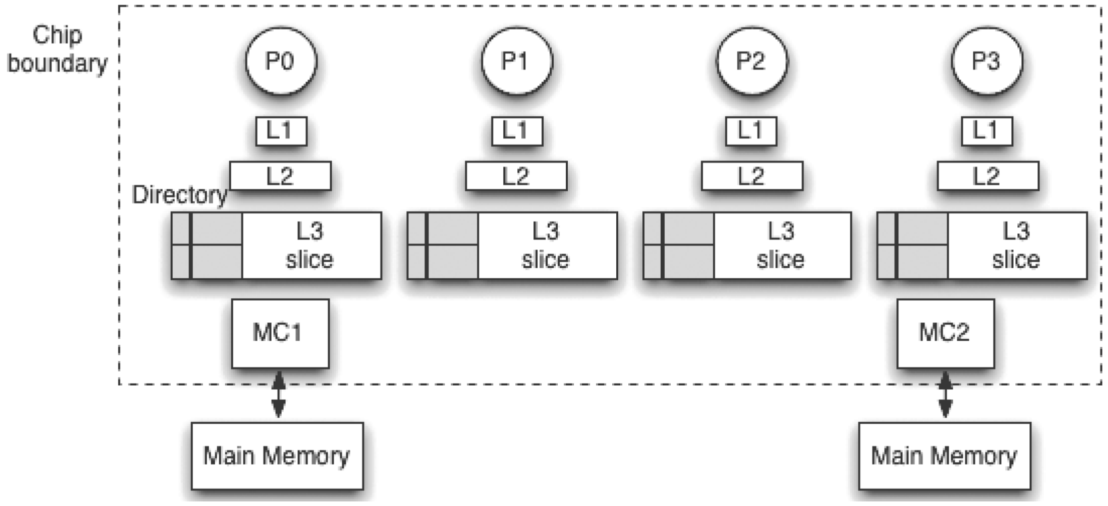
In the diagram below,

- the L3 cache is "physically distributed but logically shared," and

- the L2 caches are kept coherent.

L2 miss → L3 directory searched, block retrieved from L3 or memory

Multicore with directory-coherent Non-LLCs (L2)



In this case, the directory can be merged with the L3 tag array!

Not only does the L3 tag tell which block the L3 line holds, but also

Benefit: Lower miss latency for L2 and L3.

Drawback: Directory can hold only as many entries as there are lines in the L3.

So the L3 cache has to include all blocks cached in the L2. Why?