Computer Architecture
计 算 机 体 系 结 构

第18讲：TLP（4）

张献伟

xianweiz.github.io

DCS3013, 12/5/2022

Review Questions

- Data X is shared in processors A and B. Steps for A to write X? (note: cache is write-back)

  Acquires bus, sends invalidate, then updates X (shared → modified)

- Next, processor B reads X. What will happen?

  Places a miss on bus, A responds data and also writes back to mem.

- Possible to have X being modified in both A and B?

  No way. Modified is exclusive, writes are serialized on bus.

- MSI protocol?

  Modified/Shared/Invalid. Invalidation protocol for write-back $.

- How does MESI improves MSI?

  Splits S into E(1 share) and S (2+ shares), to avoid unnecessary inv.

- False sharing miss?

  No real data sharing, can be avoided with 1B blocks.

Performance of SMPs: Misses (cont.)

- **True sharing misses**, in an invalidation-based protocol
  - The first write by a processor to a shared block causes an invalidation to establish ownership of that block (invalidate all)
  - When another processor tries to read a modified word in that block, a miss occurs and the resultant block is transferred (invalidated by the store)
- **False sharing misses**
  - Caused by the coherence alg. with a single valid bit per block
  - Occurs when a block is invalidated (and a subsequent reference causes a miss)
    - Some word in the block, other than the one being read, is written into

Shared:
x1
x2
x1
x2

Time	P1	P2
1	Write x1	
2		Read x1

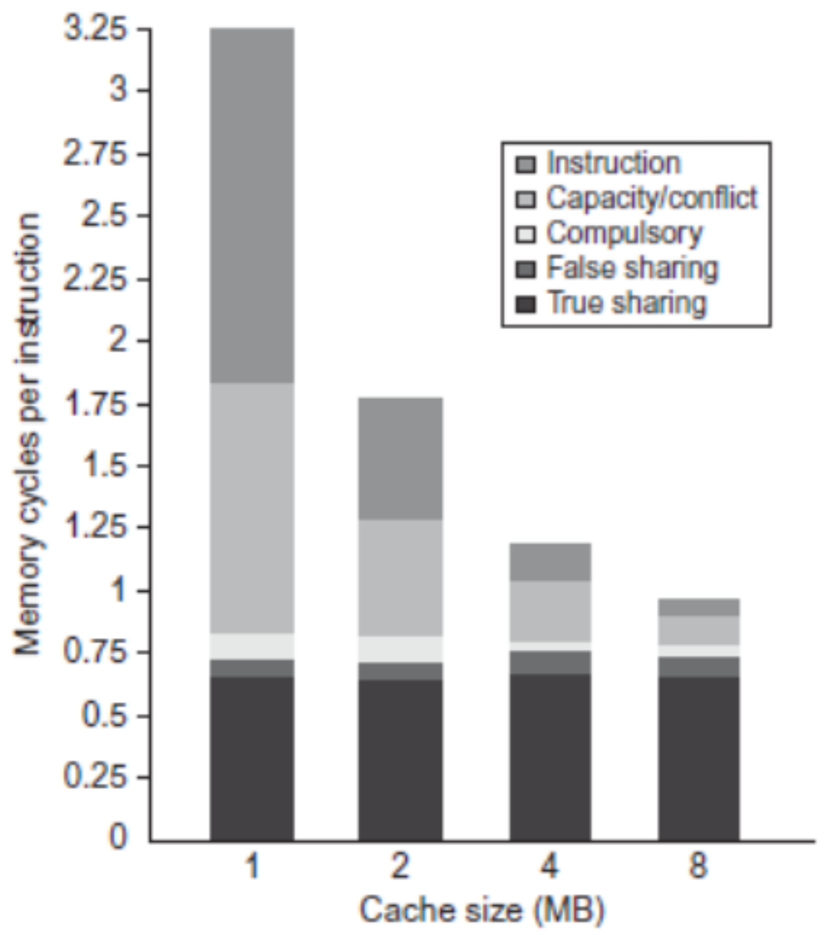
True sharing miss: x1 was invalidated by ‘write x1’ in P1

Shared:
x1
x2
x1
x2

Time	P1	P2
1	Write x1	
2		Read x2

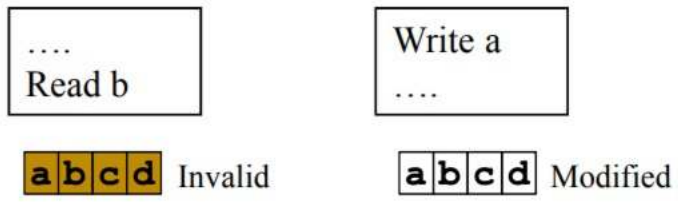
False sharing miss: x2 was invalidated by ‘write x1’ in P1

Performance of SMPs: Result



CPI for commercial benchmarks

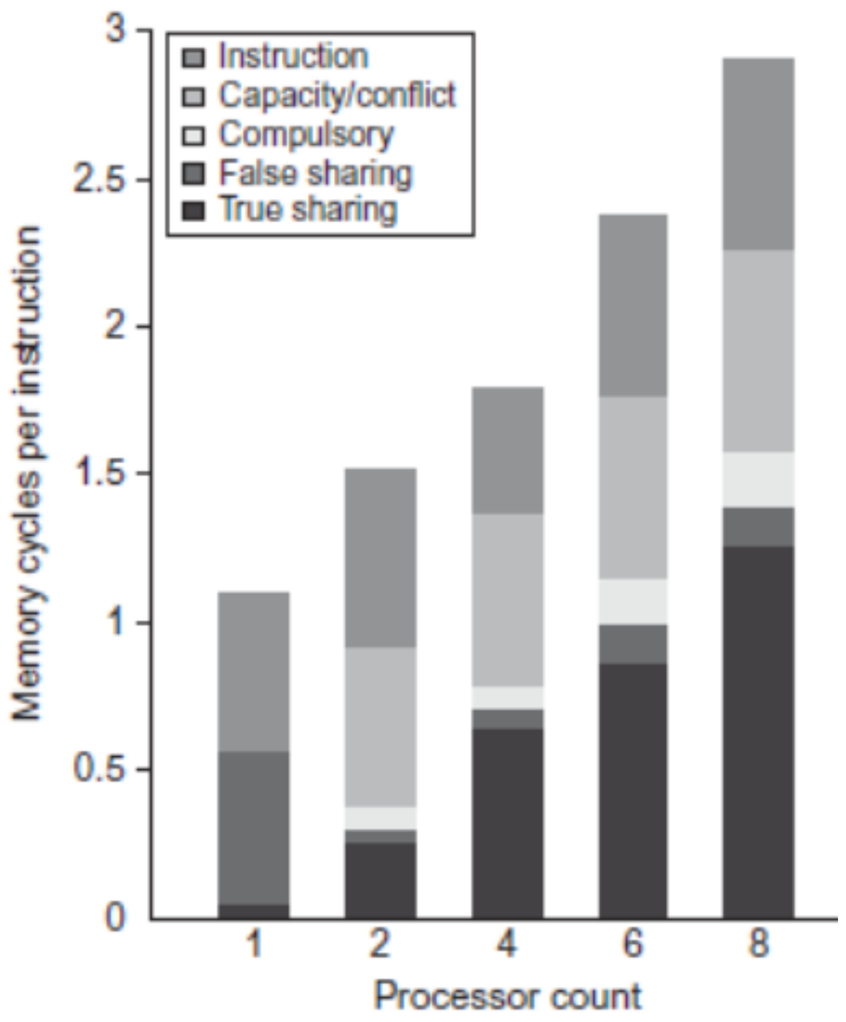
- Coherence misses:
  - True sharing misses
    - Write to a shared block
    - Read an invalidated block
  - False sharing misses
    - Read an unmodified word in an invalidated block

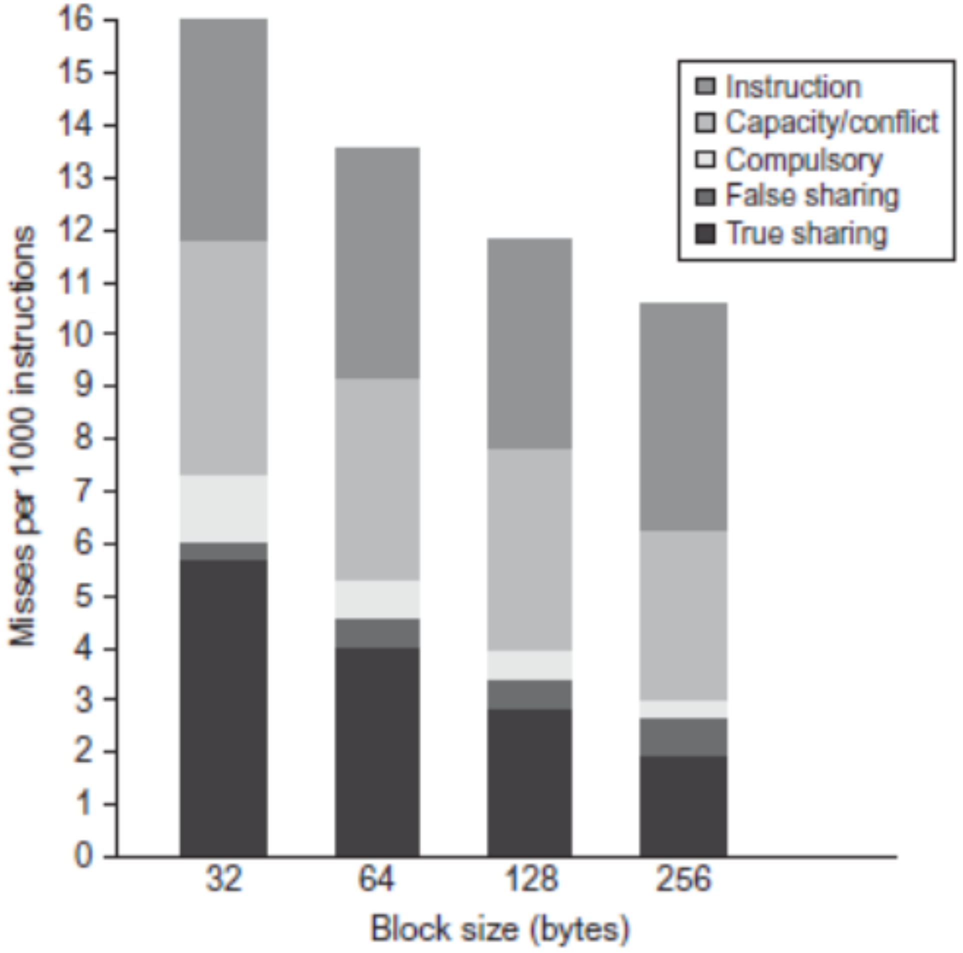


Increasing the cache size eliminates most of the uniprocessor misses while leaving the multiprocessor misses untouched.

https://people.cs.pitt.edu/~melhem/courses/2410p/ch5-2.pdf

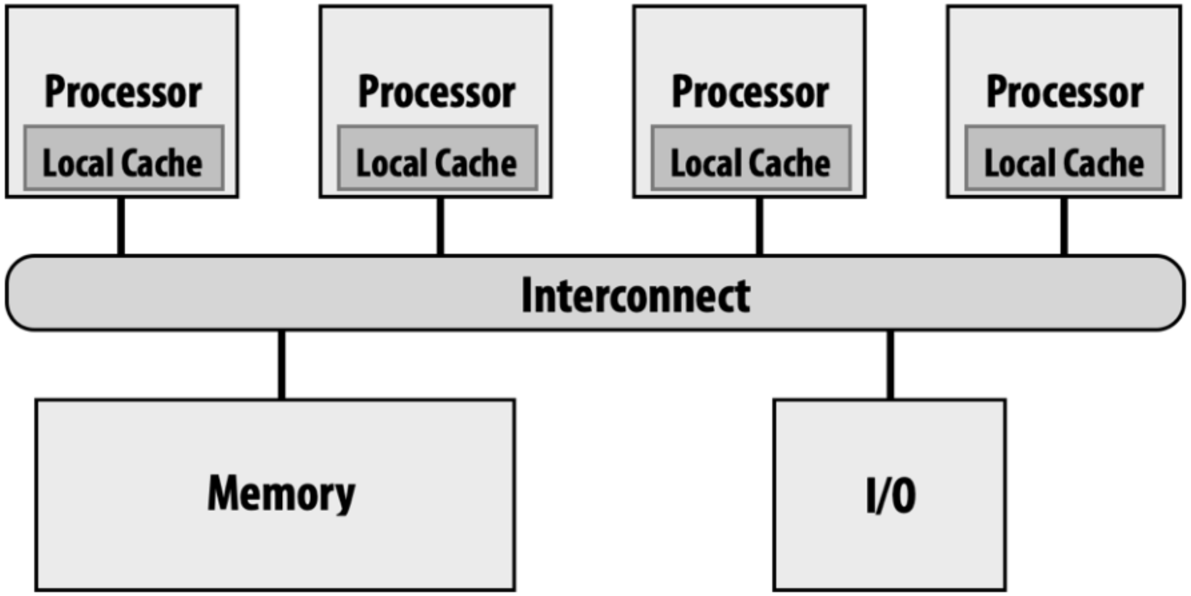
Performance of SMPs: Result (cont.)





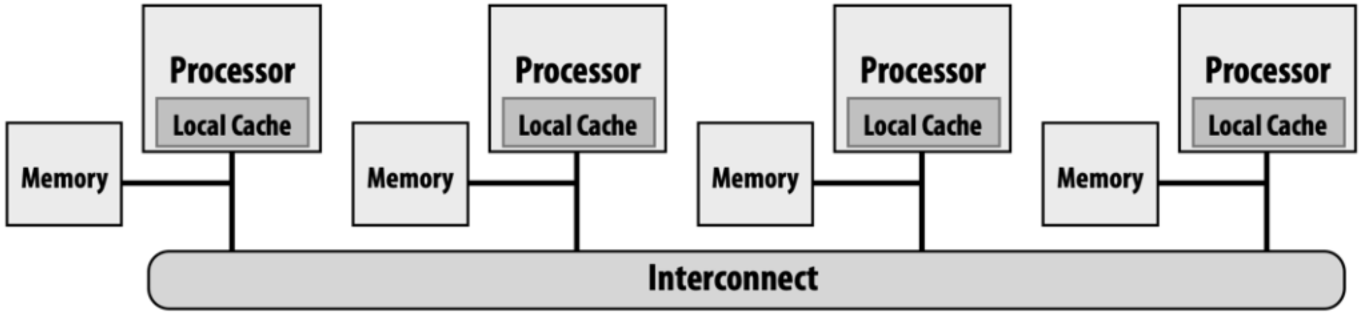
Limits of Snooping Protocol[局限]

- Snooping cache coherence protocols rely on broadcasting coherence info to all processors over the chip inter-connect[依赖于广播]
  - Cache miss occurred, triggering cache communicated with all other caches



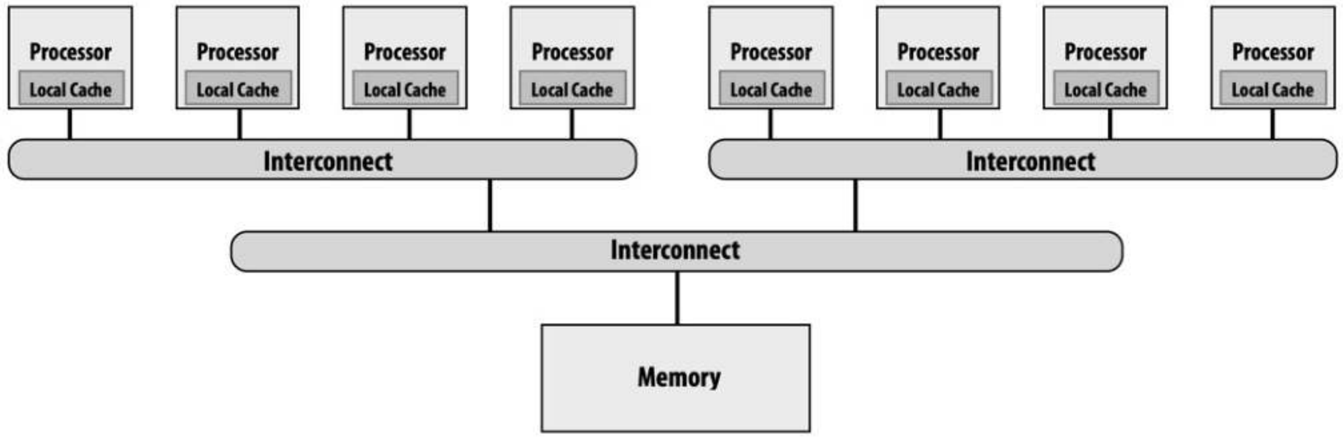
Limits of Snooping Protocol (cont.)

- On a non-uniform memory access (NUMA) shared memory system, regions of memory are located near the processors increases scalability[非一致内存访问]
  - Yield higher aggregate bandwidth and reduced latency
- NUMA does little good if the coherence protocol can't be scaled
  - Processor can access nearby memory, but need to broadcast to all other processors (overhead)



Scaling Cache Coherence

- One possible solution: hierarchical snooping[多层级]
  - Use snooping coherence at each level



  - Advantages
    - Relatively simple to build (already have to deal with similar issues due to multi-level caches)
  - Disadvantages
    - The root of network may become a performance bottleneck
    - Larger latencies than direct communication
    - Doesn’t apply to more general network topologies

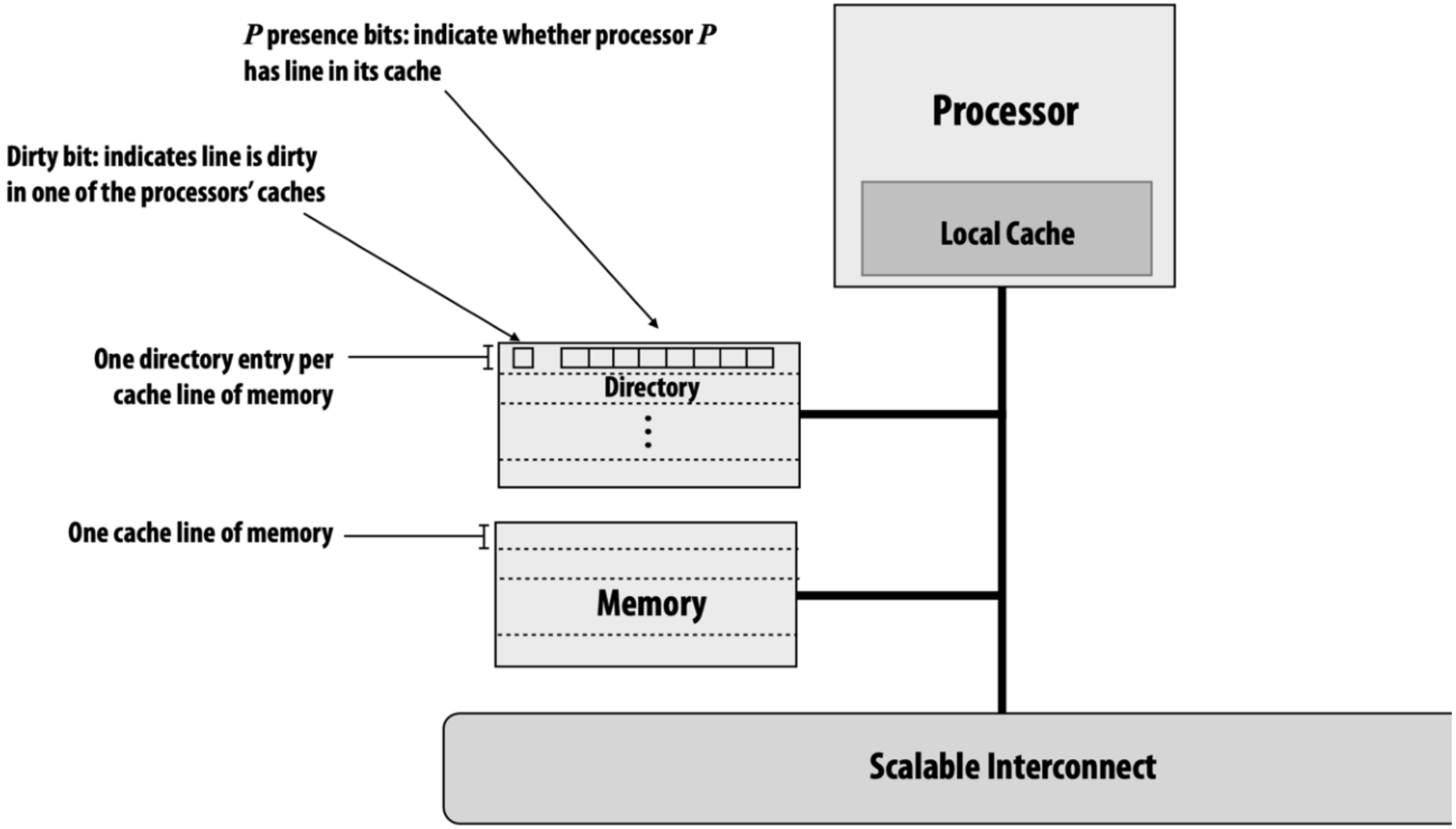
Scalable Coherence using Directories

- To avoid broadcast by storing info about status of the line in one place: directory[目录]
  - The directory entry for a cache line contains information about the state of the cache line in all caches[保存状态]
  - Caches look up information from the directory as necessary[查询目录]
  - Cache coherence is maintained by point-to-point messages between the caches (not by broadcast mechanisms)[点对点通信]

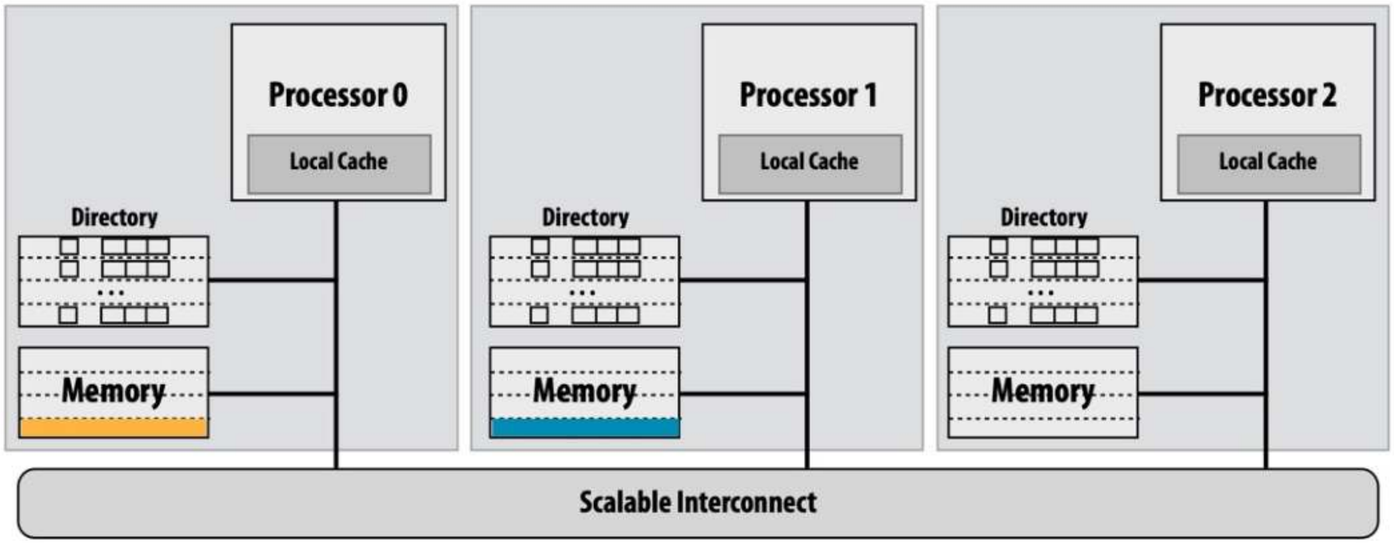
- Theoretical advantages of directory-based approach
  - The root of network won't be the performance bottleneck
  - Can apply to more general network topologies(e.g. meshes, cubes)

http://15418.courses.cs.cmu.edu/spring2017/lecture/directorycoherence

Simple Directory Protocol Impl.


P presence bits: indicate whether processor *P* has line in its cache
Dirty bit: indicates line is dirty in one of the processors' caches
One directory entry per cache line of memory
Directory
One cache line of memory
Memory
Processor
Local Cache
Scalable Interconnect

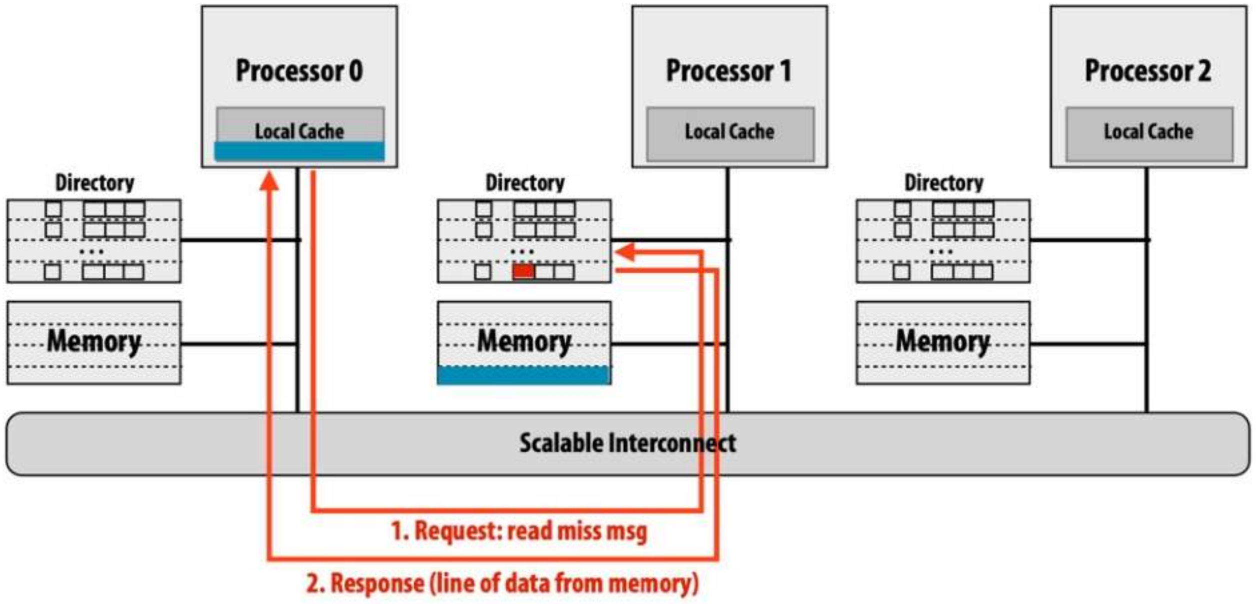
Distributed Directory: Partition[分区]



- Directory partition is co-located with memory it describes
- "**Home node**" of a line: node with memory holding the corresponding data for the line
  - For example: node 0 is the home node of orange line, node 1 is the home node of blue line
- "**Requesting node**": node containing processor requesting line

Example: read miss to clean line

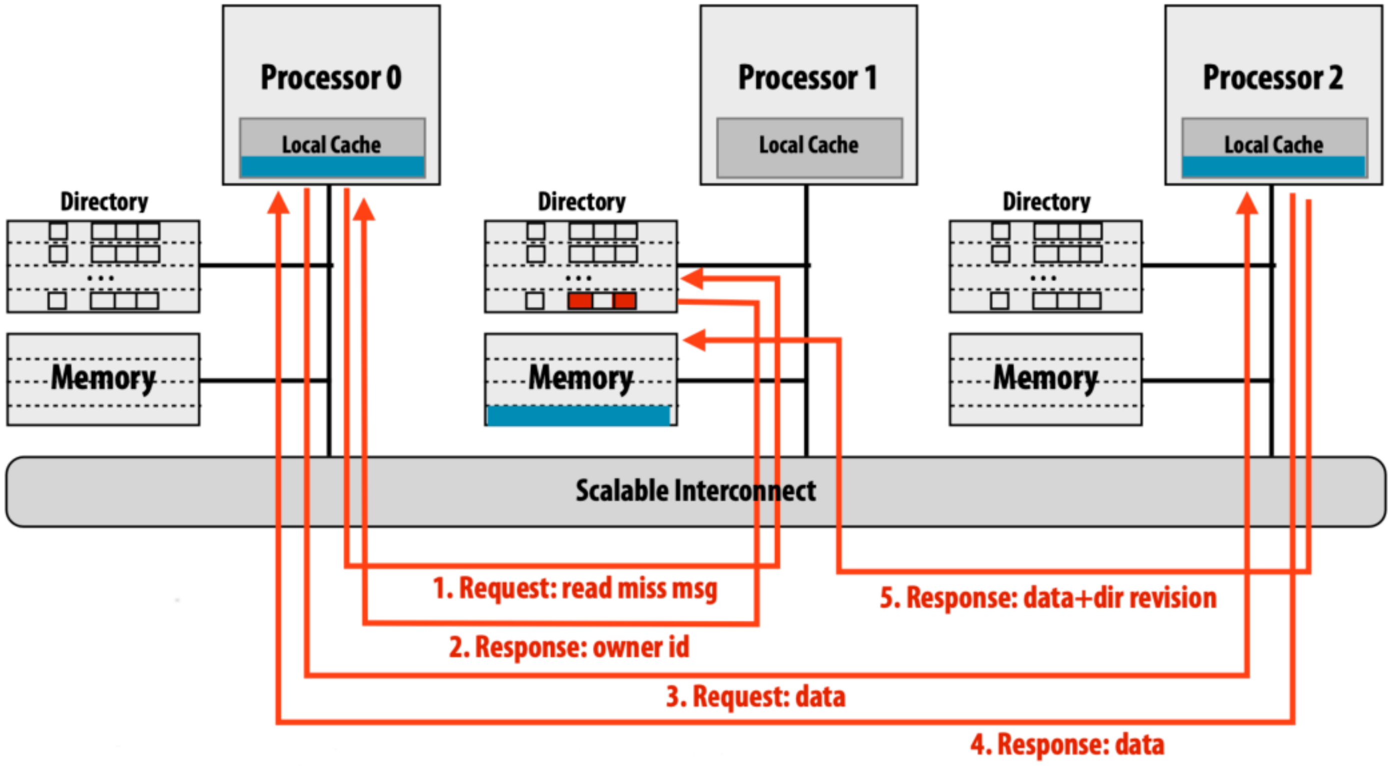
- Read from main memory by processor 0 of blue line (not dirty)



- Read miss message sent to home node of requested line
- Home directory checks entry for line
  - If dirty bit of line is OFF, respond with contents from memory, set presence[0] to true (to indicate line is cached by processor 0)

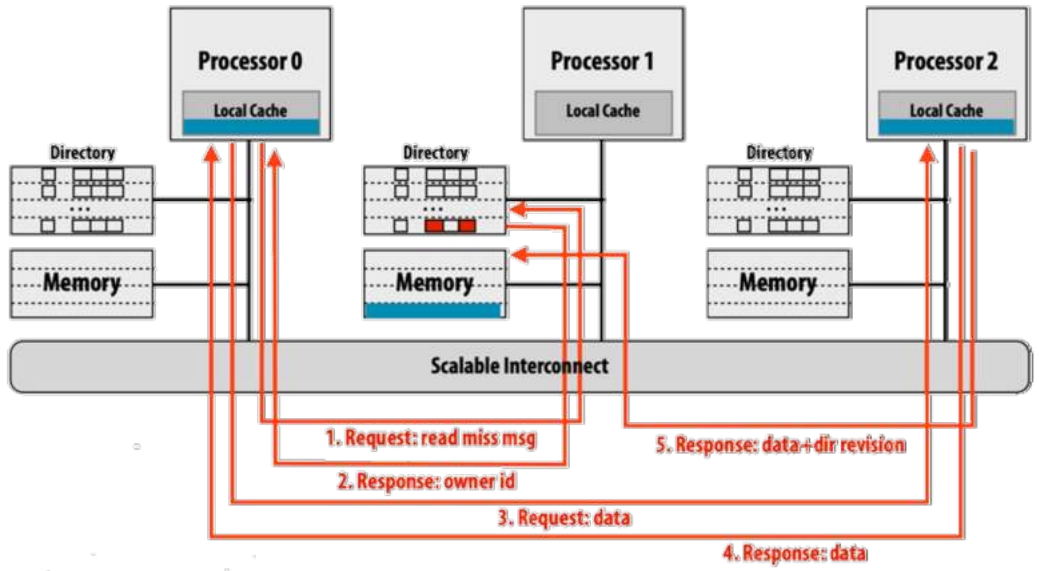
Example: read miss to dirty line

- Read from main memory by processor 0 of blue line
  - Dirty and its content is in P2’s cache



http://15418.courses.cs.cmu.edu/spring2017/lecture/directorycoherence

Example: read miss to dirty line (cont.)

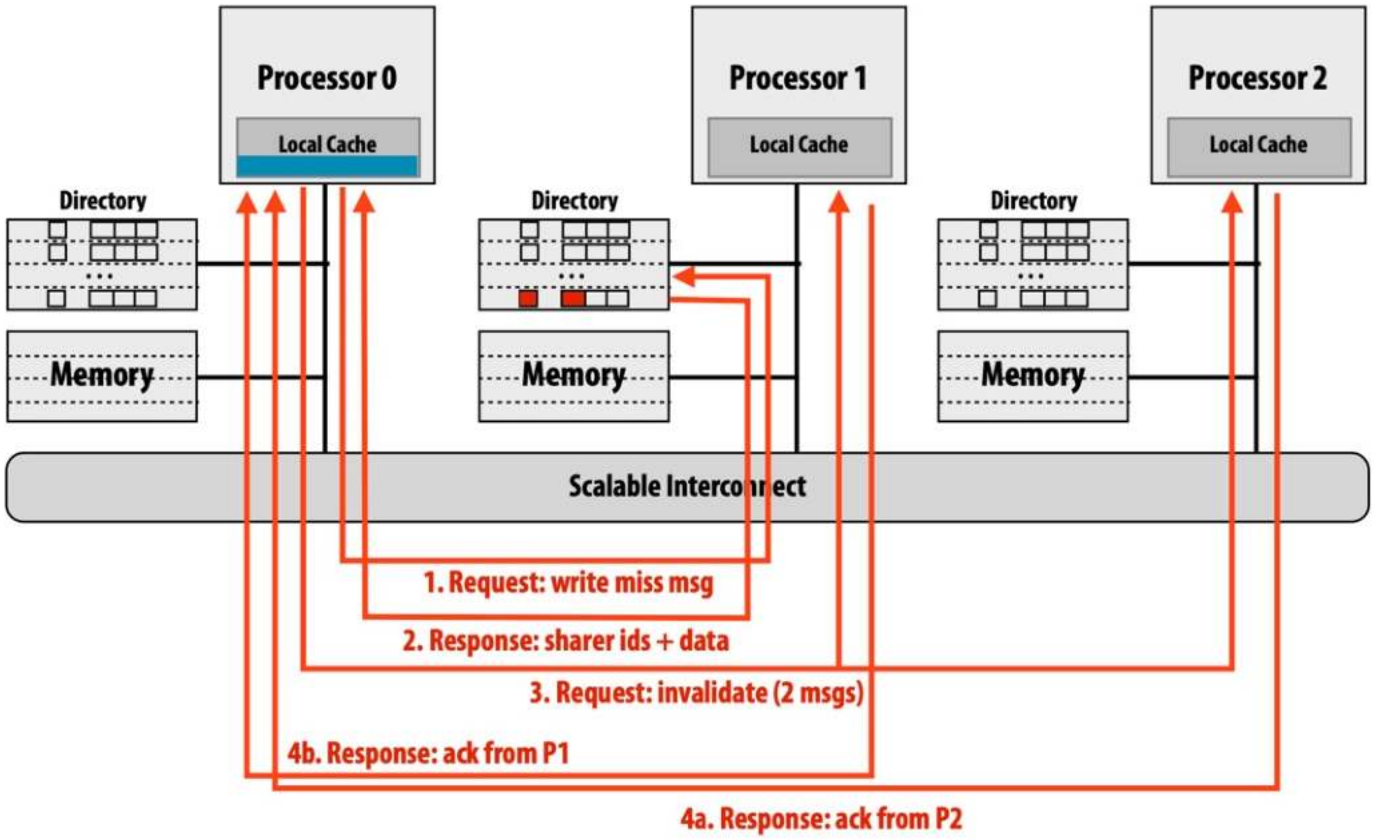


1. If dirty bit is ON, data must be sourced by another processor
2. <u>Home node</u> responds with <u>id</u> of line owner
3. <u>Requesting node</u> requests data from owner
4. <u>Owner</u> responds to <u>requesting node</u>
    - changes state in cache to SHARED (read only)
5. <u>Owner</u> also responds to <u>home node</u>, home clears dirty
    - updates presence bits, updates memory

http://15418.courses.cs.cmu.edu/spring2017/lecture/directorycoherence

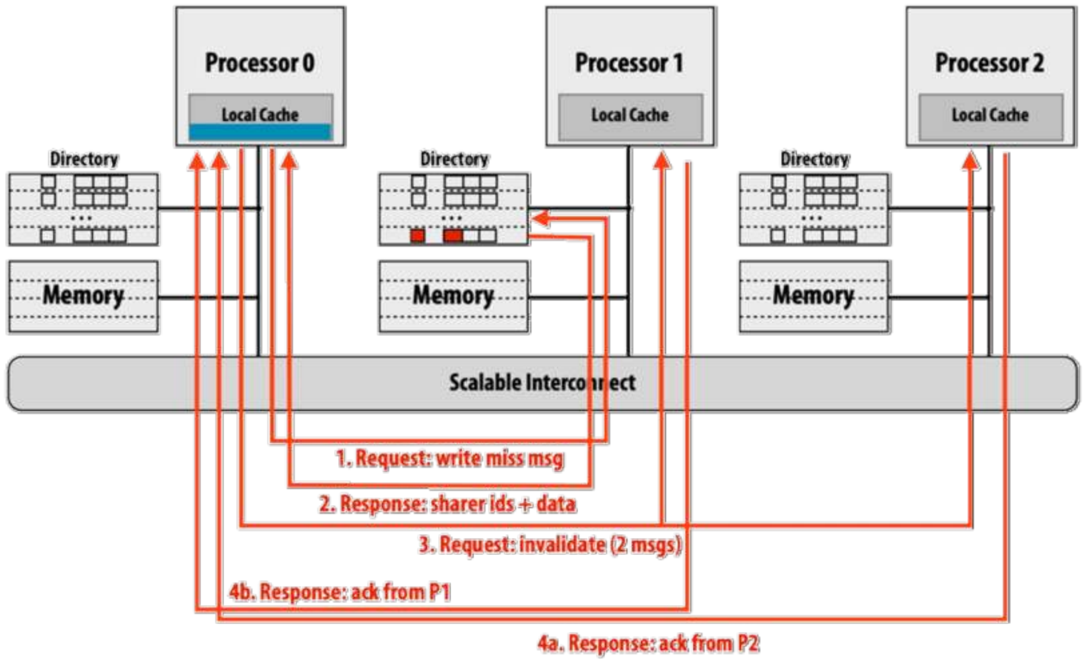
Example: write miss

- Write to memory by processor 0
  - Line is clean, but resident in P1's and P2's caches


Processor 0
Local Cache
Processor 1
Local Cache
Processor 2
Local Cache
Directory
Directory
Directory
Memory
Memory
Memory
Scalable Interconnect
1. Request: write miss msg
2. Response: sharer ids + data
3. Request: invalidate (2 msgs)
4b. Response: ack from P1
4a. Response: ack from P2

http://15418.courses.cs.cmu.edu/spring2017/lecture/directorycoherence

Example: write miss (cont.)


Processor 0
Local Cache
Processor 1
Local Cache
Processor 2
Local Cache
Directory
Directory
Directory
Memory
Memory
Memory
Scalable Interconnect
1. Request: write miss msg
2. Response: sharer ids + data
3. Request: invalidate (2 msgs)
4b. Response: ack from P1
4a. Response: ack from P2

1. Requesting node sends the write miss to home node
2. Home node responds with ids of nodes containing this data (sharer) and data
3. Requesting sharer to invalidate corresponding data
4. Get response from P1 and P2
   - After receiving both invalidation acks, P0 can write

http://15418.courses.cs.cmu.edu/spring2017/lecture/directorycoherence

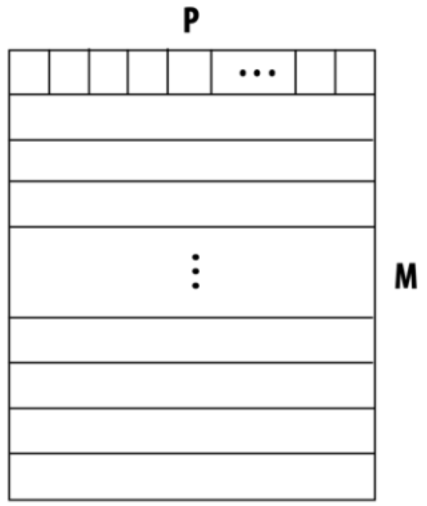
Pros of Directory Protocol

- On reads, directory tells requesting node exactly where to get the line from
  - Either from home node (if the line is clean)
  - Or from the owning node (if the line is dirty)
  - Either way, retrieving data involves only point-to-point communication

- On writes, the advantages of directories depends on the number of sharers
  - In the limit, if all caches are sharing data, all caches must be communicated with (just like broadcast in a snooping protocol)

Cons of Directory Protocol

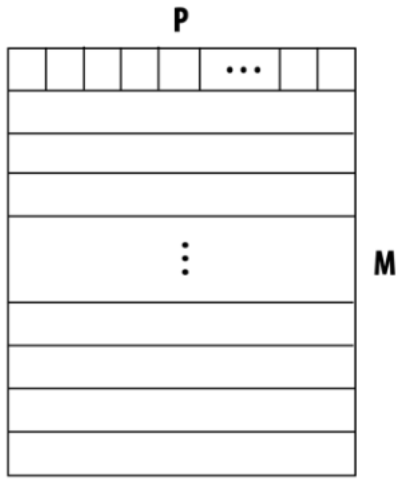
- Full bit vector directory representation
  - One presence bit per node



- Storage proportional to P * M
  - P = number of nodes (e.g., processors)
  - M = number of lines in memory
- Storage overhead rises with P
  - Assume 64 byte cache line size (512 bits)
  - 64 nodes (P=64) -> 12.5% overhead
  - 256 nodes (P=256) -> 50% overhead
  - 1024 nodes (P=1024) -> 200% overhead

Reducing Storage Overheads

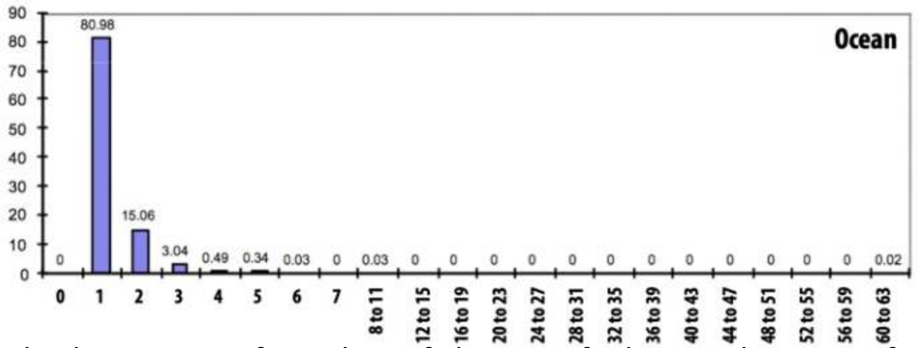
- Optimizations on full-bit vector scheme
  - Increase cache line size (reduce M term)
  - Group multiple processors into a single directory “node” (reduce P term)
    - Need only one directory bit per node, not one bit per processor
    - Hierarchical: could use snooping protocol to maintain coherence among processors in a node, directory across nodes



- Two alternative schemes
  - **Limited pointer** schemes (reduce P)
  - **Sparse directories** (reduce M)

Limited Pointer Schemes (LPS)[有限指针]

- Since data is expected to only be in a few caches at once, storage for a limited number of pointers per directory entry should be sufficient (only need a list of the nodes holding a valid copy of the line)[数据通常小范围内共享]
  - Example:
    - In a 1024 processor system
    - Full bit vector scheme needs 1024 bits per line
    - Using limited pointer scheme, 1024 bits can store approximately 100 pointers to nodes holding the line (log(1024) = 10 bits per pointer)
    - In practice, we can get by with far less than this (20-80 principle)



Graphs plot histogram of number of sharers of a line at the time of a write

http://15418.courses.cs.cmu.edu/spring2017/lecture/directorycoherence

Managing Overflow in LPS[管理溢出]

If too many pointers (sharers) are required…

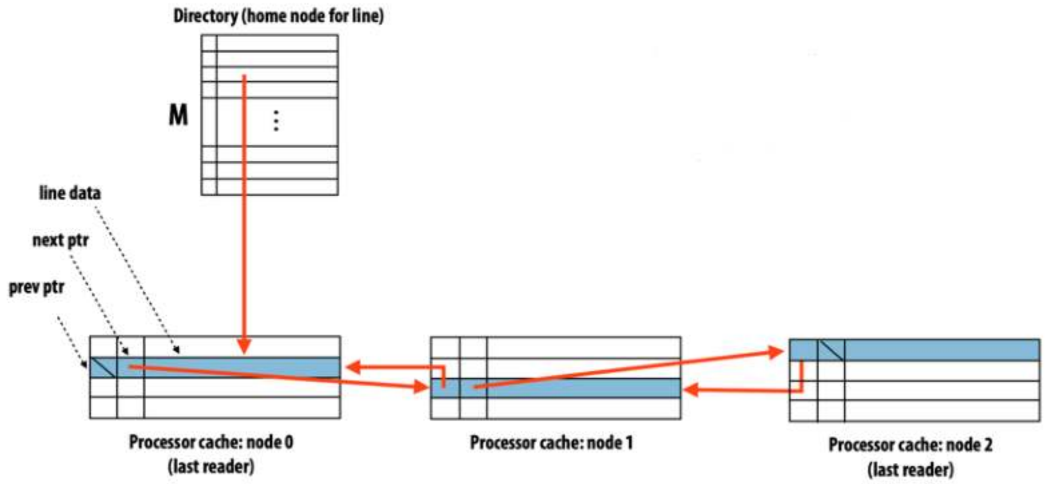
- Fallback to broadcast (if broadcast mechanism exists)[广播]
  - When more than max number of sharers, revert to broadcast
- If no broadcast mechanism present on machine[阈值]
  - Don’t allow more than a max number of sharers
  - On overflow, newest sharer replaces an existing one (must invalidate line in the old sharer’s cache)
- Coarse vector fallback[粗粒度]
  - Revert to 'bit' vector representation
  - Each bit corresponds to K nodes
  - On write, invalidate all nodes a bit corresponds to

Summary of Limited Pointer Schemes

- LPS reduces directory storage overhead caused by large P
  - By adopting a compact representation of a list of shares
- But do we really need to maintain storage for a list for each cache-line chunk of data in memory?
- Key observation: the majority of memory is NOT resident in cache. And to carry out coherence protocol the system only needs sharing information for lines that are currently in cache[仅小部分数据被缓存]
  - Most directory entries are empty most of the time
  - 1 MB cache, 1 GB memory per node -> 99.9% of directory entries are idle

Sparse Directories[稀疏目录]

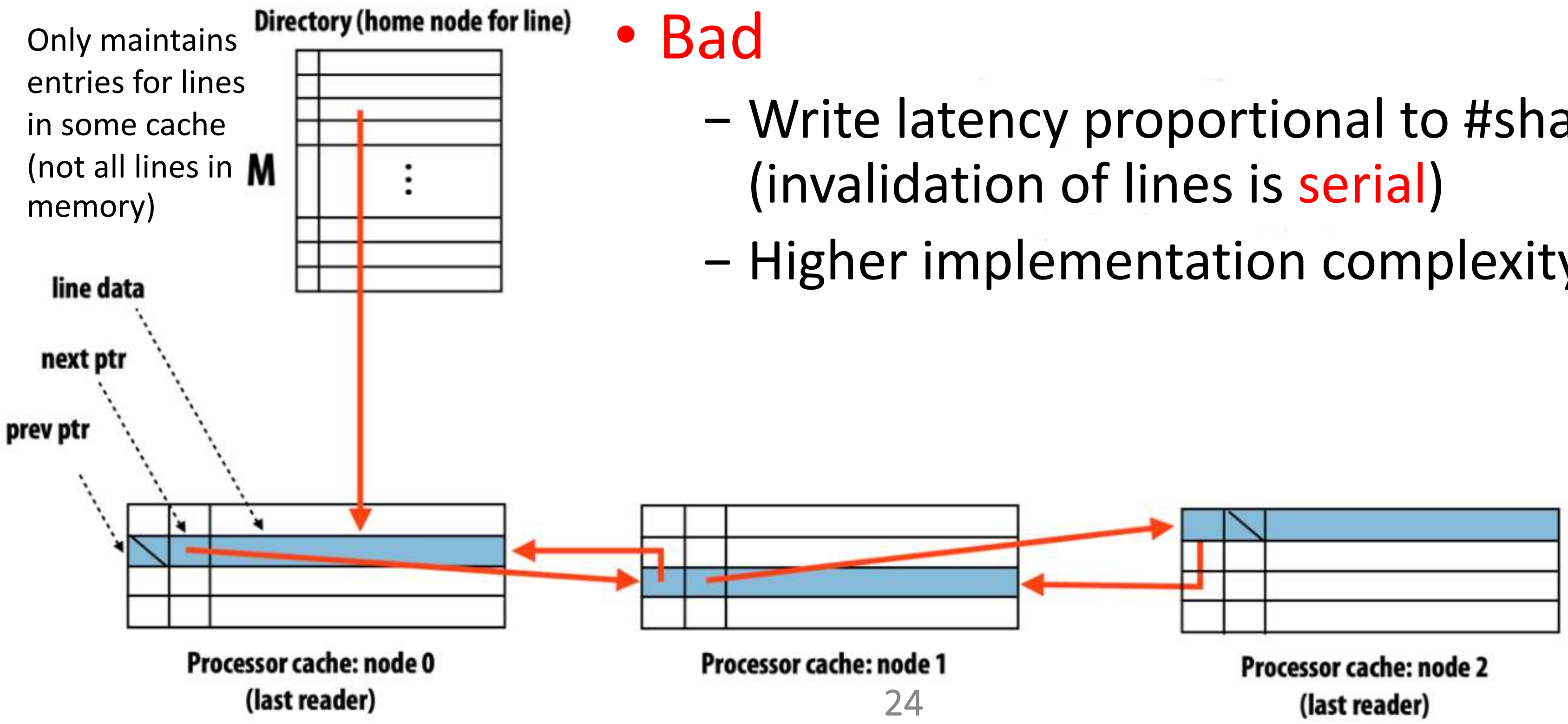
- Directory at home node maintains pointer to only one node caching line (not a list of sharers)[仅指向一个]
- Pointer to next node in list is stored as extra information in the cache line (like the line's tag, dirty bits, etc.)[链表]
- On read miss: add requesting node to head of list
- On write miss: propagate invalidations along list
- On evict: need to patch up list (linked list removal)



http://15418.courses.cs.cmu.edu/spring2017/lecture/directorycoherence

Scaling Properties of Sparse Directories

- Good
  - Low memory storage overhead (one pointer to list head per line)
  - Additional directory storage is proportional to cache size (the list stored in SRAM)
  - Traffic on write is still proportional to number of sharers

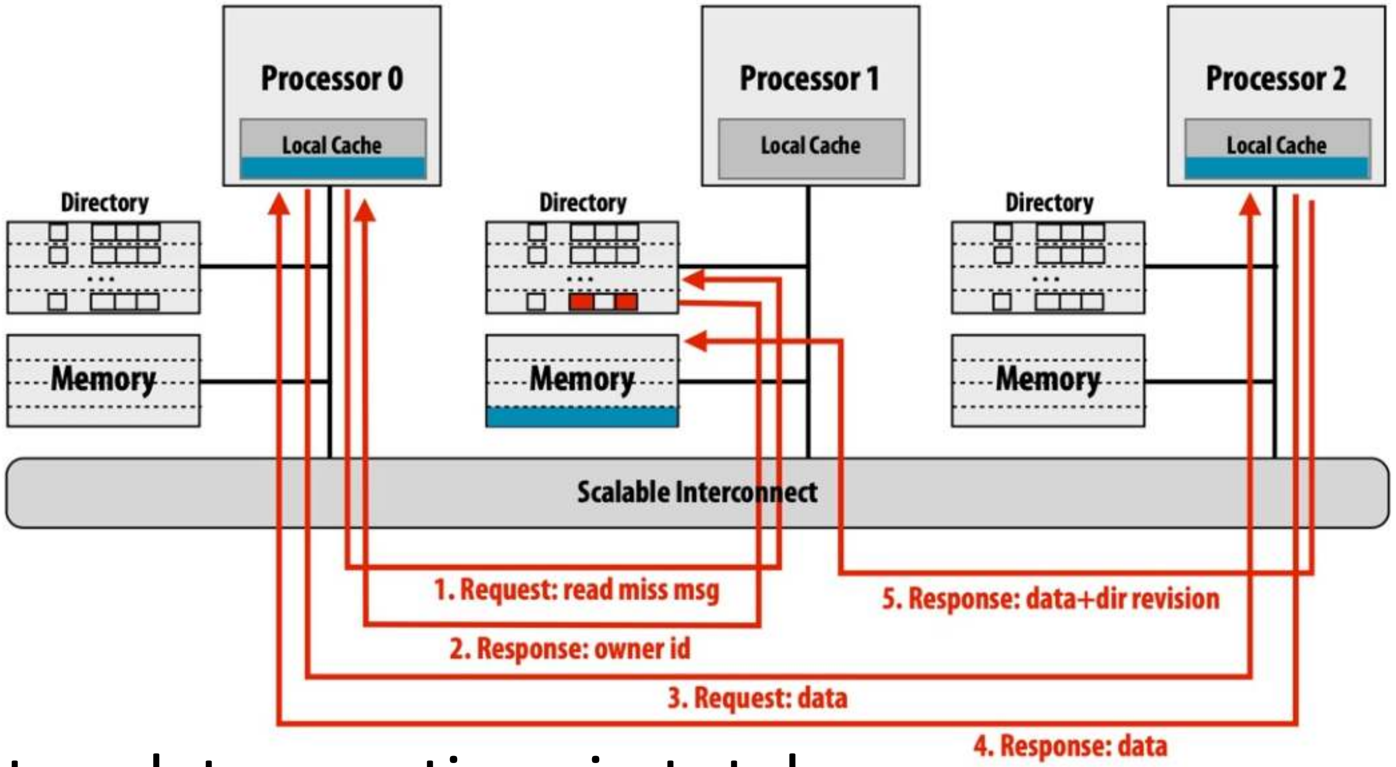

Only maintains entries for lines in some cache (not all lines in memory)
Directory (home node for line)
M
line data
next ptr
prev ptr
Processor cache: node 0 (last reader)
Processor cache: node 1
Processor cache: node 2 (last reader)

- Bad
  - Write latency proportional to #sharers (invalidation of lines is serial)
  - Higher implementation complexity

http://15418.courses.cs.cmu.edu/spring2017/lecture/directorycoherence

Reduce #msg. Sent

Read from main memory by P0 of the blue line: line is dirty (contained in P2’s cache)

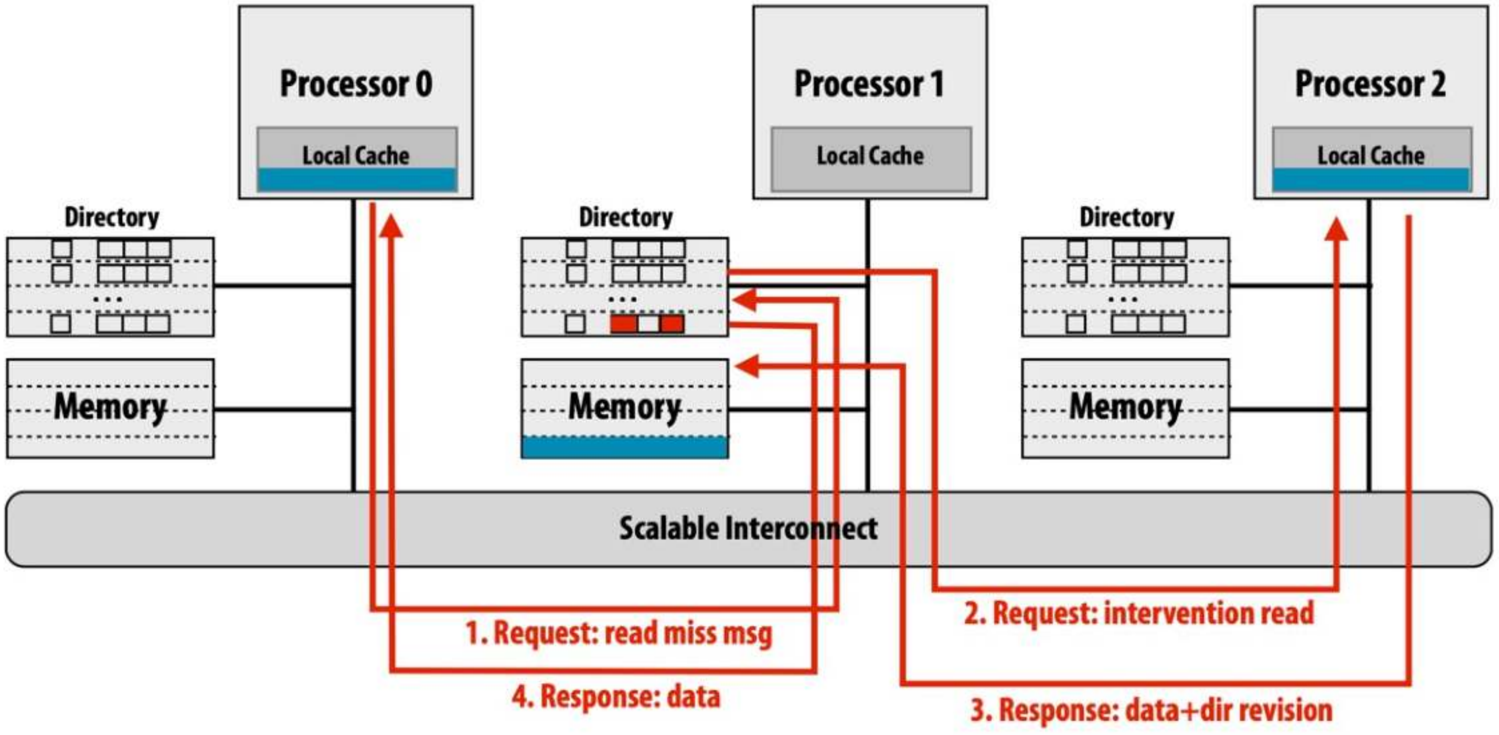


Five network transactions in total

Four of them are sequential (transaction 4 & 5 can parallel)

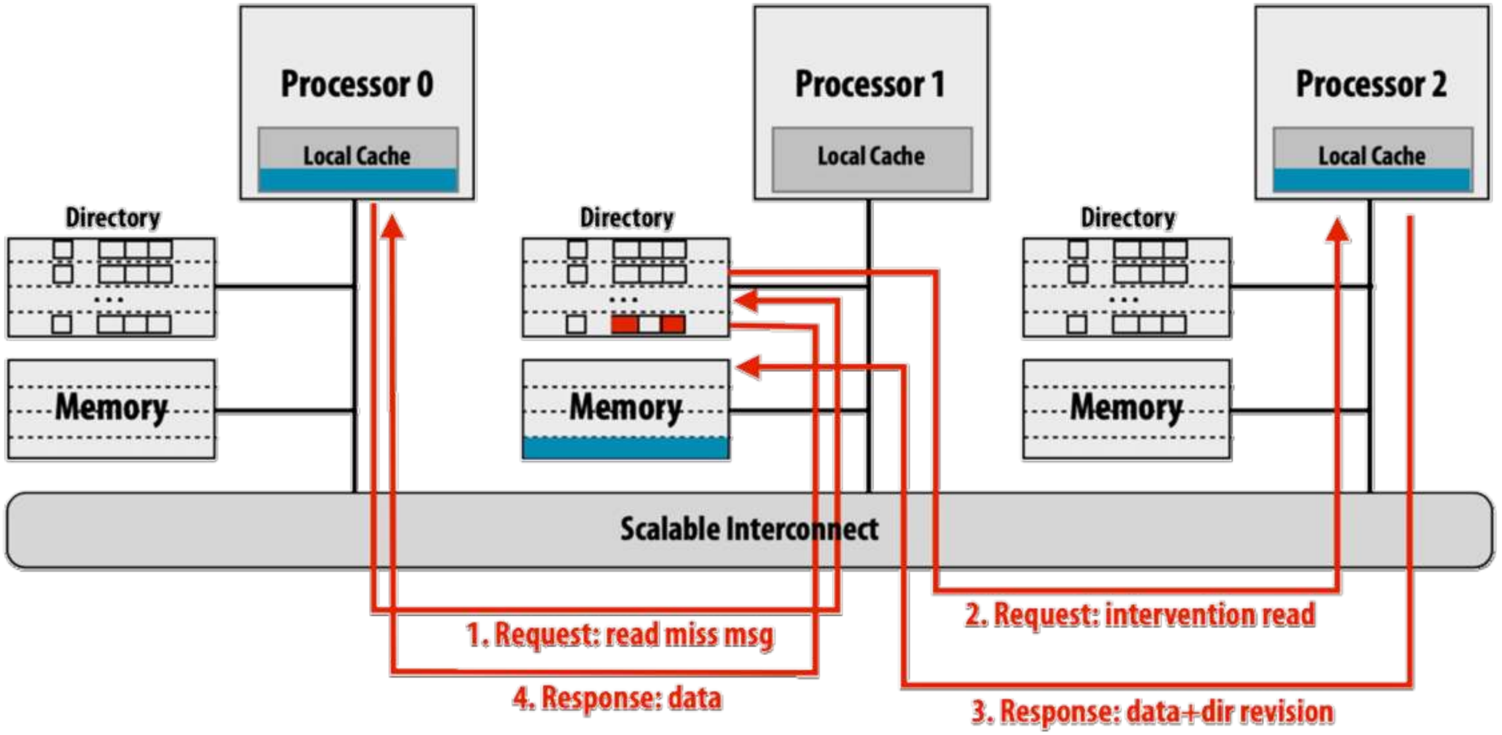
Intervention Forwarding[干预转发]

Read from main memory by P0 of the blue line: line is dirty (contained in P2's cache)



Total 4 transactions are needed

Intervention Forwarding (cont.)



1. Requests to read miss message on home node (P1)

2. Home node requests data from owner node (P2)

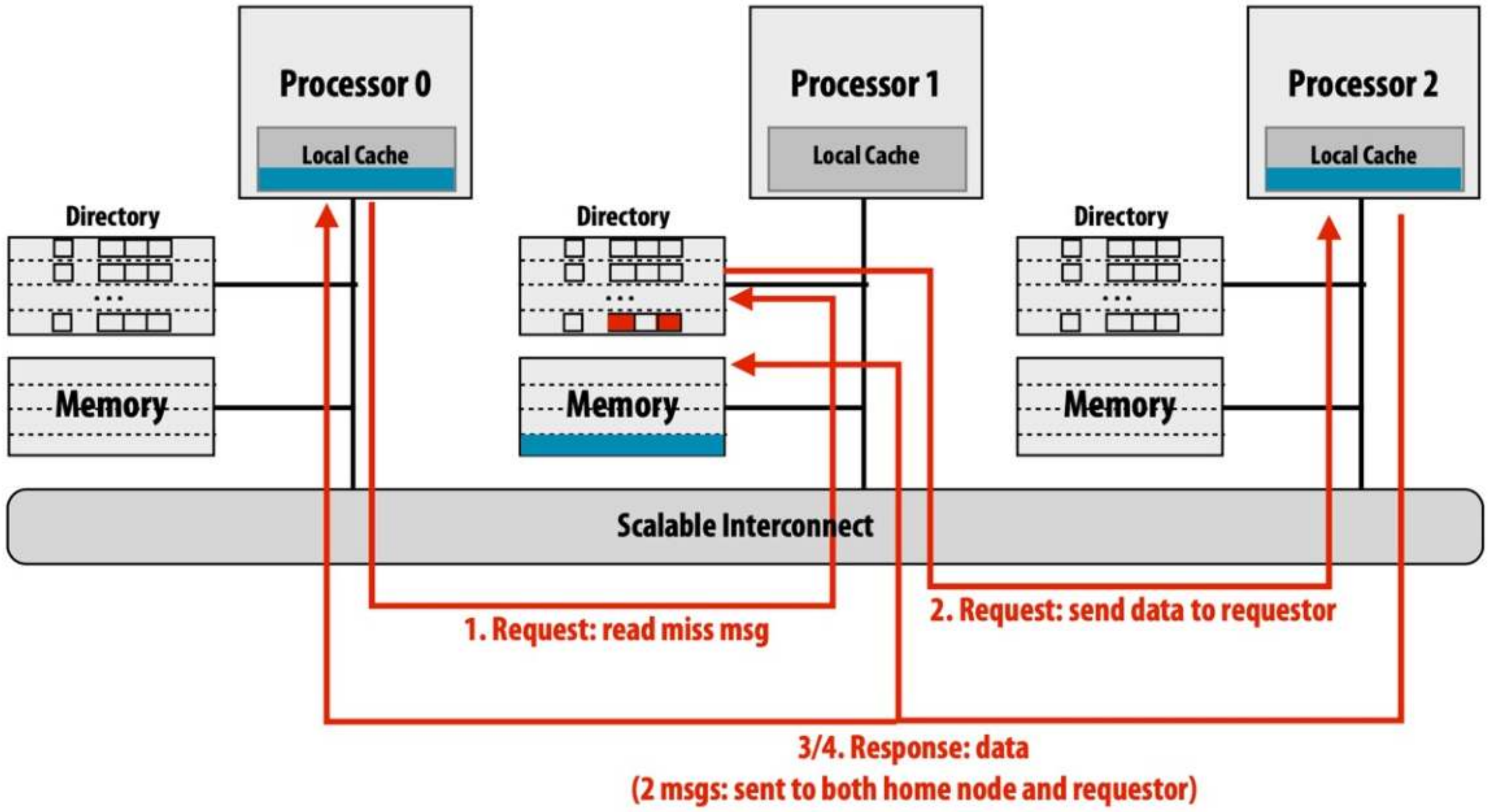
3. Owning node response

4. Home node updates directory, responds to requesting node with requested data

All transactions are sequential, can they be parallel?

Request Forwarding[请求转发]

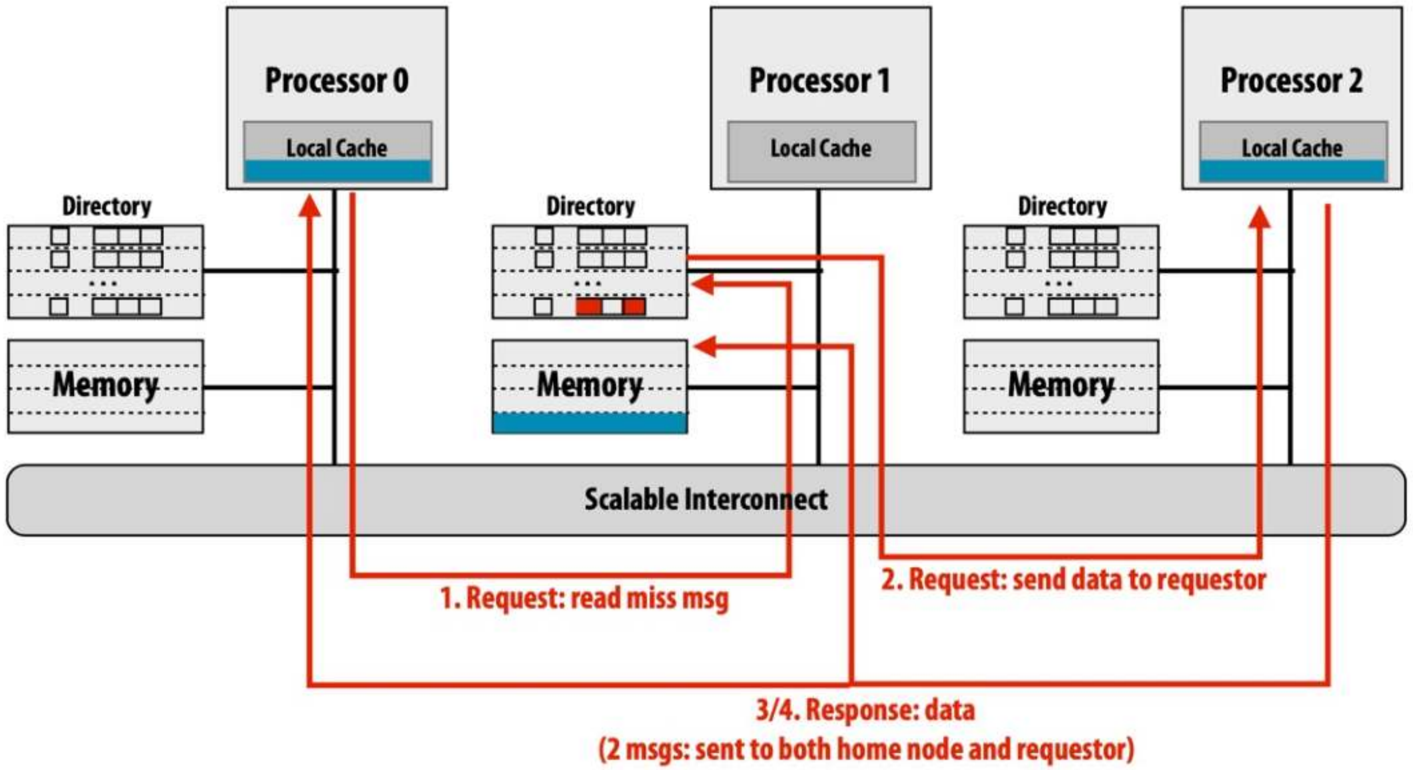
Read from main memory by P0 of the blue line: line is dirty (contained in P2's cache)



Only 3 transactions are in serial

Transaction 3 & 4 can be parallel

Request Forwarding (cont.)



1. Requests to read miss message on home node (P1)
2. Home node sends target data to owner
3. Owning node responses data to the home node
4. Owning node responses data to the requesting node

Summary of Directory-base Coherence

- Primary observation: broadcast doesn’t scale, but we don’t need to broadcast to ensure coherence because often the number of caches containing a copy of a line is small
- Instead of snooping, just store the list of sharers in a directory and check the list when necessary
- One challenge on storage[存储]
  - Use hierarchies of processors or larger cache size
  - Limited pointer schemes: exploit fact that most processors not sharing line
  - Sparse directory schemes: exploit fact that most lines not in cache
- Another challenge on communication[通信]
  - Reduce messages sent (traffic) and parallelize trans (latency)