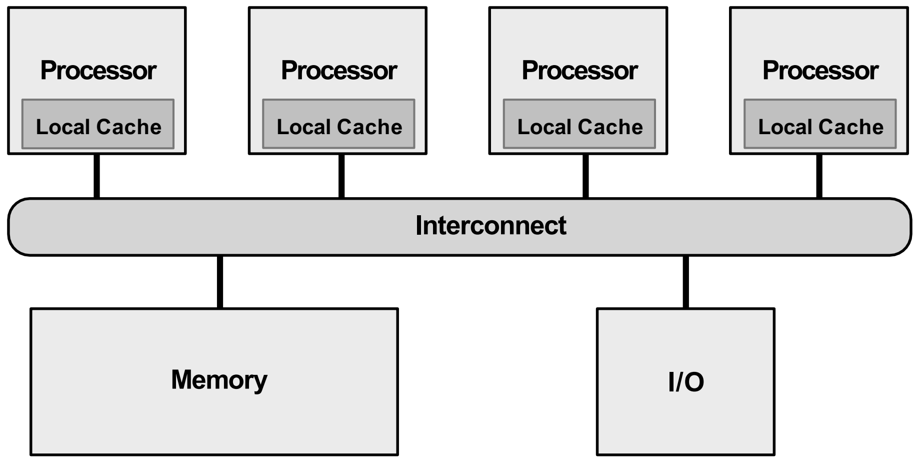
Directory Based Cache Coherence

CSCE 4213 Introduction to Computer Architecture

David Andrews

Slides are Adapted from
Parallel Computer Architecture and Programming CMU 15-418/15-618, Spring 2019

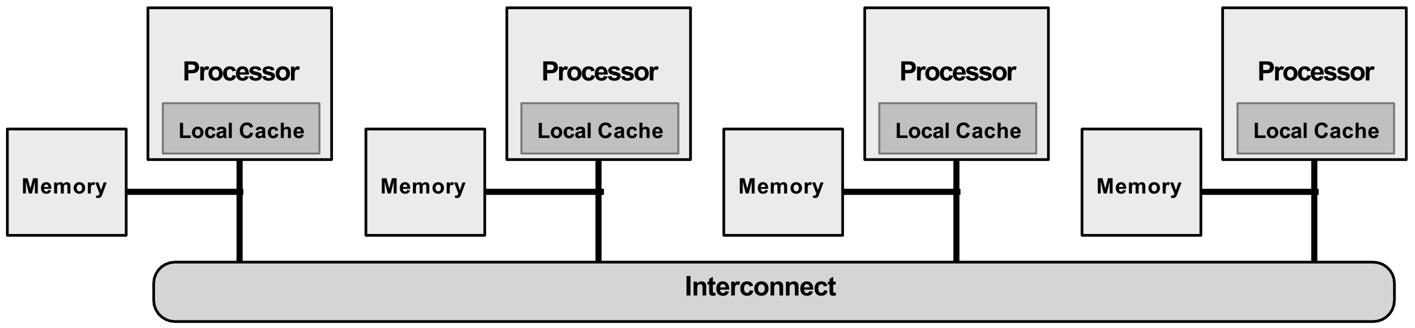
Implementing cache coherence



The snoopy cache coherence protocol relies on

1. **Single bus to <u>broadcast</u> coherence information to all processors**
2. **All caches monitoring all bus traffic "snoop" and take appropriate action**
3. **"Broadcast doesn't scale past a handful of processors**

Scaling cache coherence to large machines



Recall non-uniform memory access (NUMA) shared memory systems

Idea: locating regions of memory near the processors increases scalability: it yields higher aggregate bandwidth and reduced latency (especially when there is locality in the application)

But... efficiency of NUMA system does little good if the coherence protocol can't also be scaled!

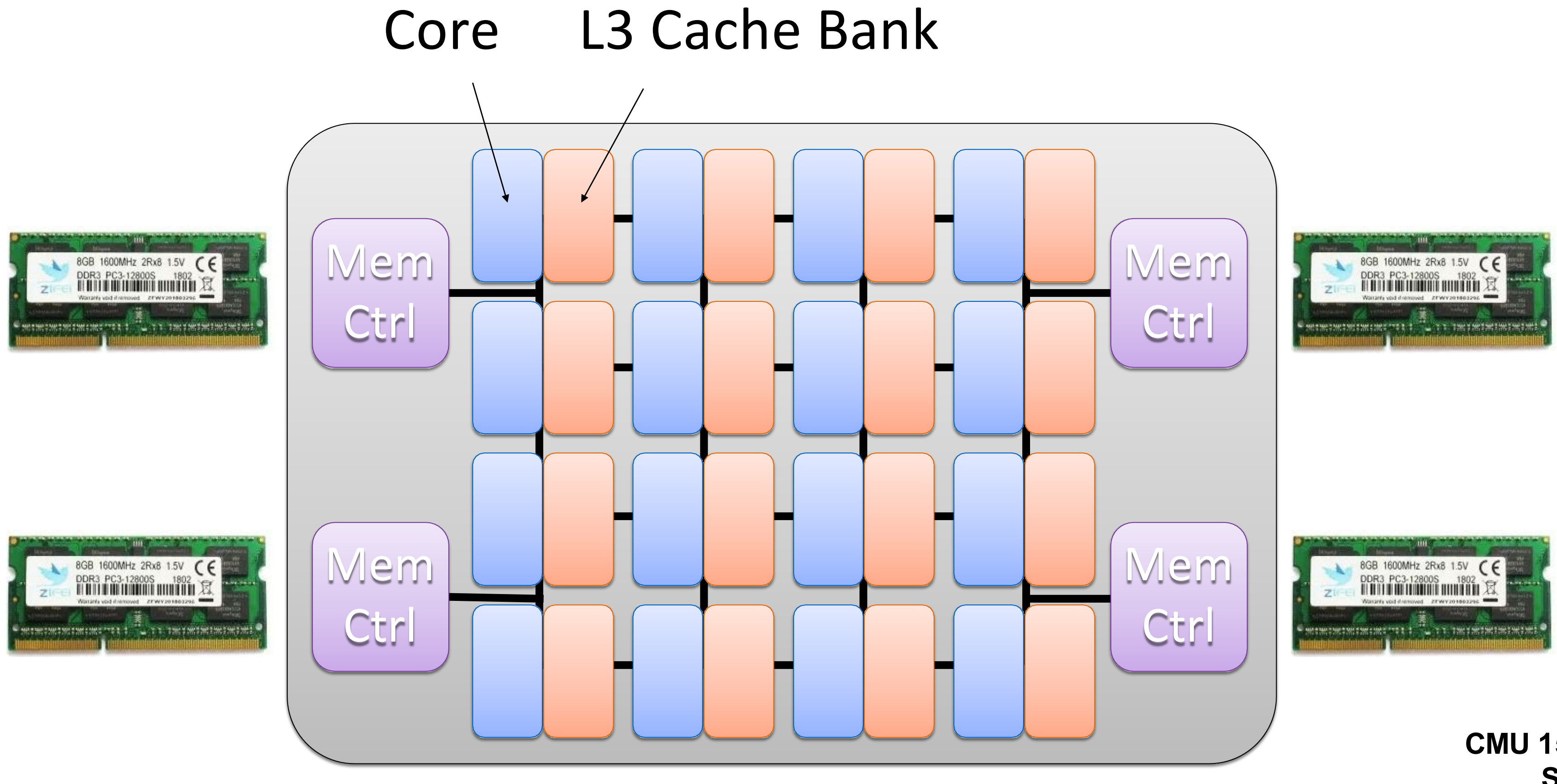
Consider this case: processor accesses nearby memory (good...), but to ensure coherence still must broadcast to all other processors it is doing so (bad...)

Some terminology:

- cc-NUMA = "cache-coherent, non-uniform memory access"
- Distributed shared memory system (DSM): cache coherent, shared address space, but architecture implemented by physically distributed memories

Scaling cache coherence in current multicores

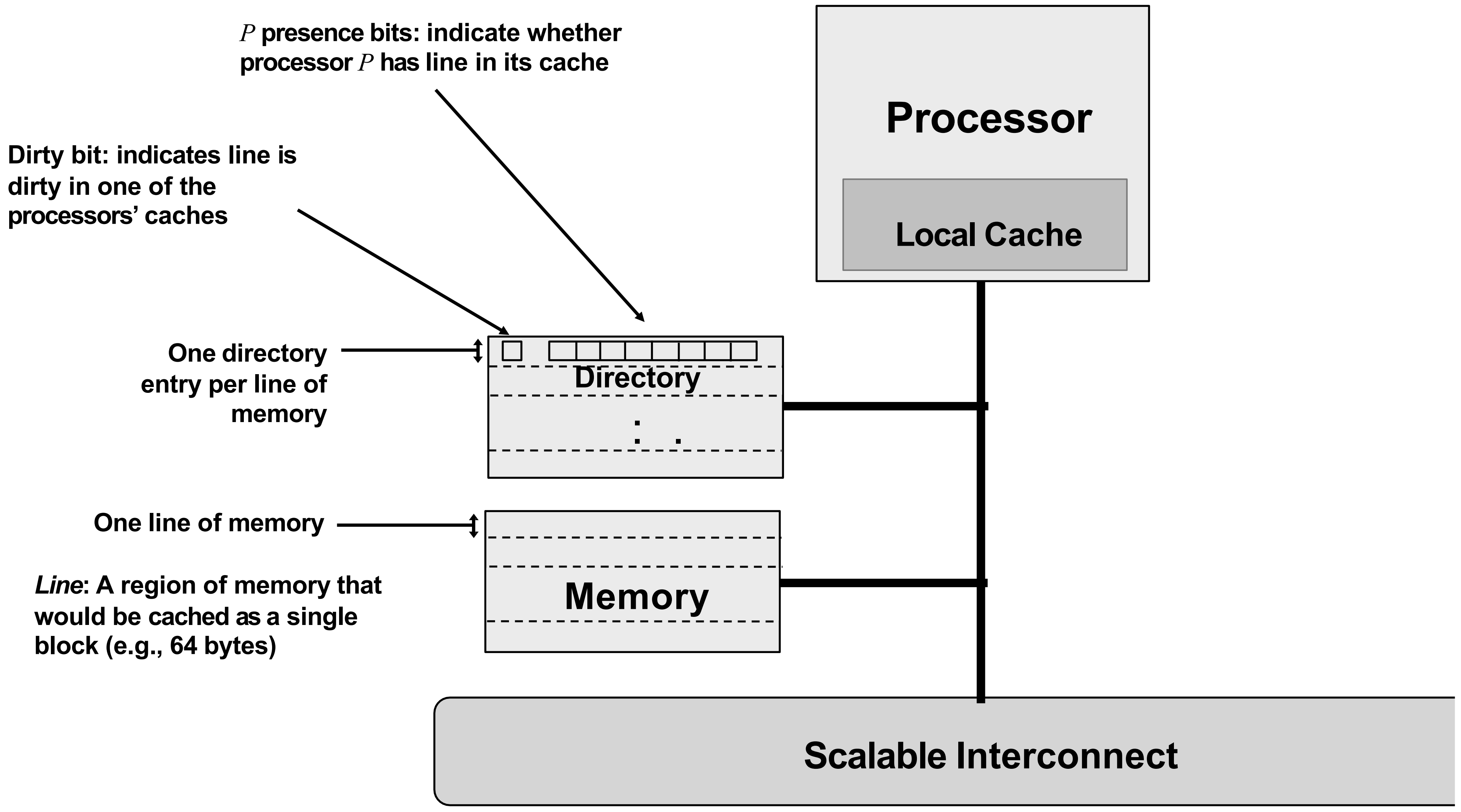
- ccNUMA typically refers to supercomputing/clusters
- Same issues appear in multicores
  - NUMA: Memory controllers distributed around chip
  - NUCA (non-uniform cache access): L3 banks distributed around the chip too



Scalable cache coherence using <u>directories</u>

- Snooping schemes <u>broadcast</u> coherence messages to determine the state of a line in the other caches
- Alternative idea: avoid broadcast by storing information about the status of the line in one place: a "directory"
  - The directory entry for a cache line contains information about the state of the cache line in all caches.
  - Caches look up information from the directory as necessary
  - Cache coherence is maintained by point-to-point messages between the caches on a "need to know" basis (not by broadcast mechanisms)

A very simple directory



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

Line: A region of memory that would be cached as a single block (e.g., 64 bytes)

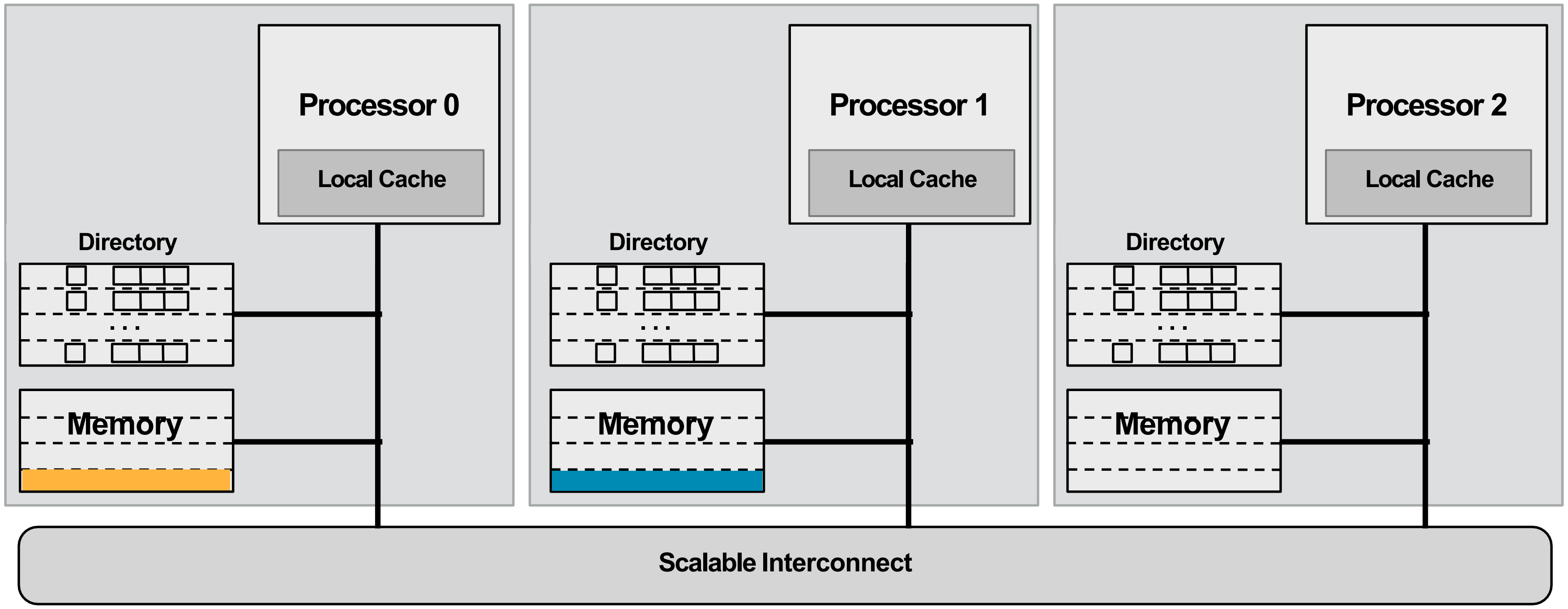
So we've traded a bus bottleneck for a memory bottleneck?

- Not quite; directories distributed across memory banks
  - *Different* ordering points for *different* addresses

- Can cache directory entries + avoid memory access
  - Caches are much faster + higher bandwidth than memory

A distributed directory in ccNUMA

Example: directory partition is co-located with memory it describes

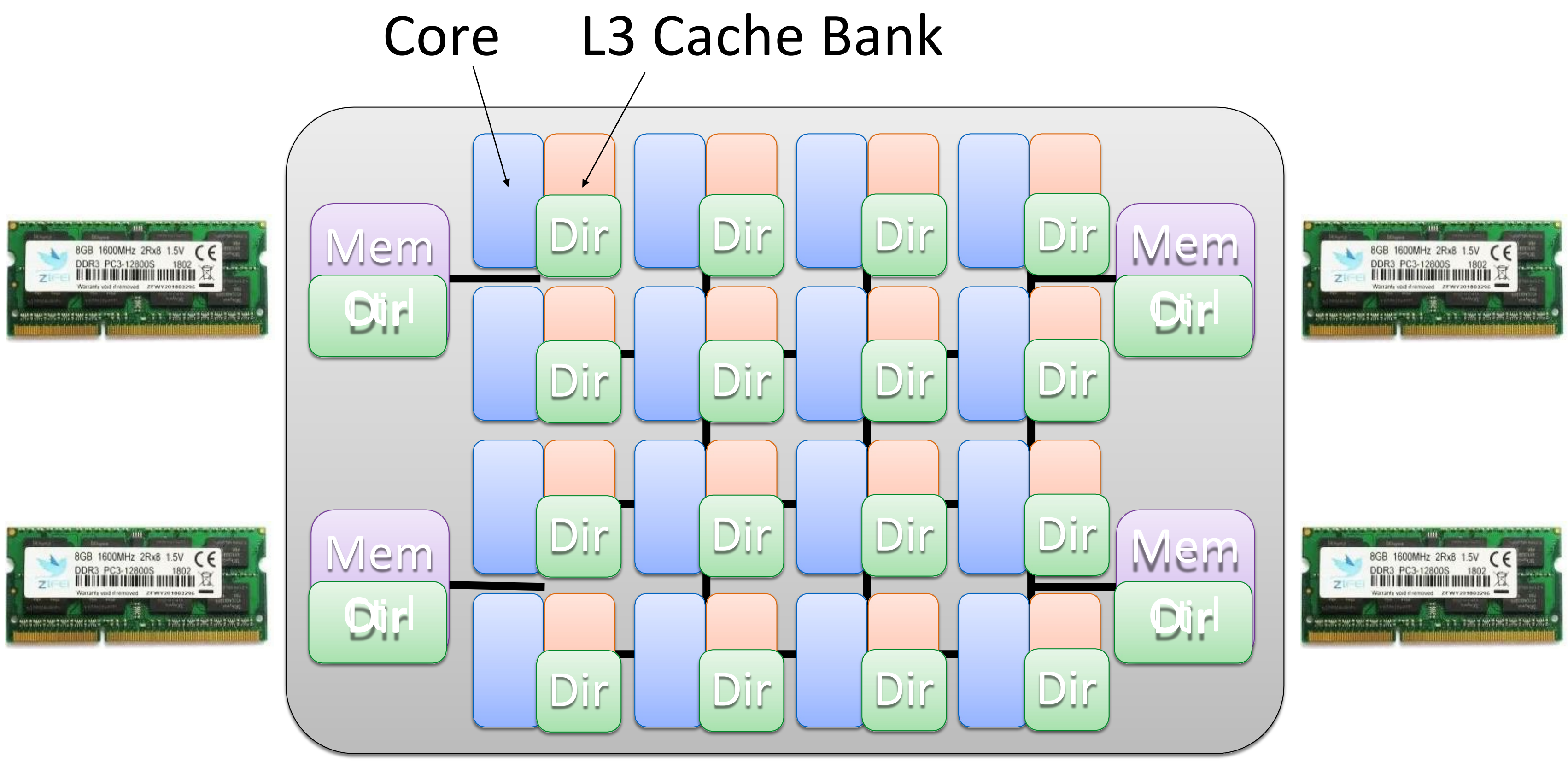


- **"Home node" of a line: node with memory holding the corresponding data for the line**

Example: node 0 is the home node of the yellow line, node 1 is the home node of the blue line

- **"Requesting node": node containing processor requesting line**

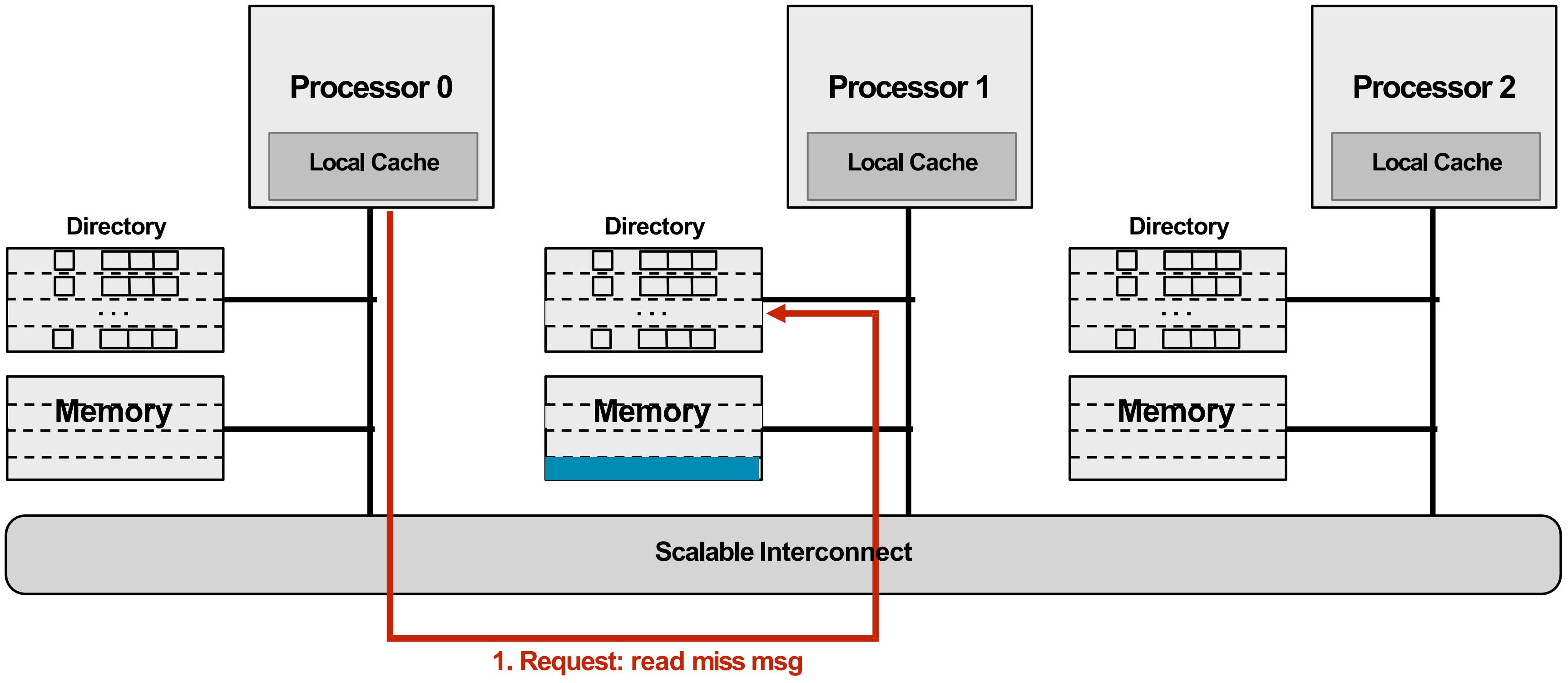
A distributed directory in a multicore



- **As we shall see, directories really live in each L3 bank**

Example 1: read miss to clean line

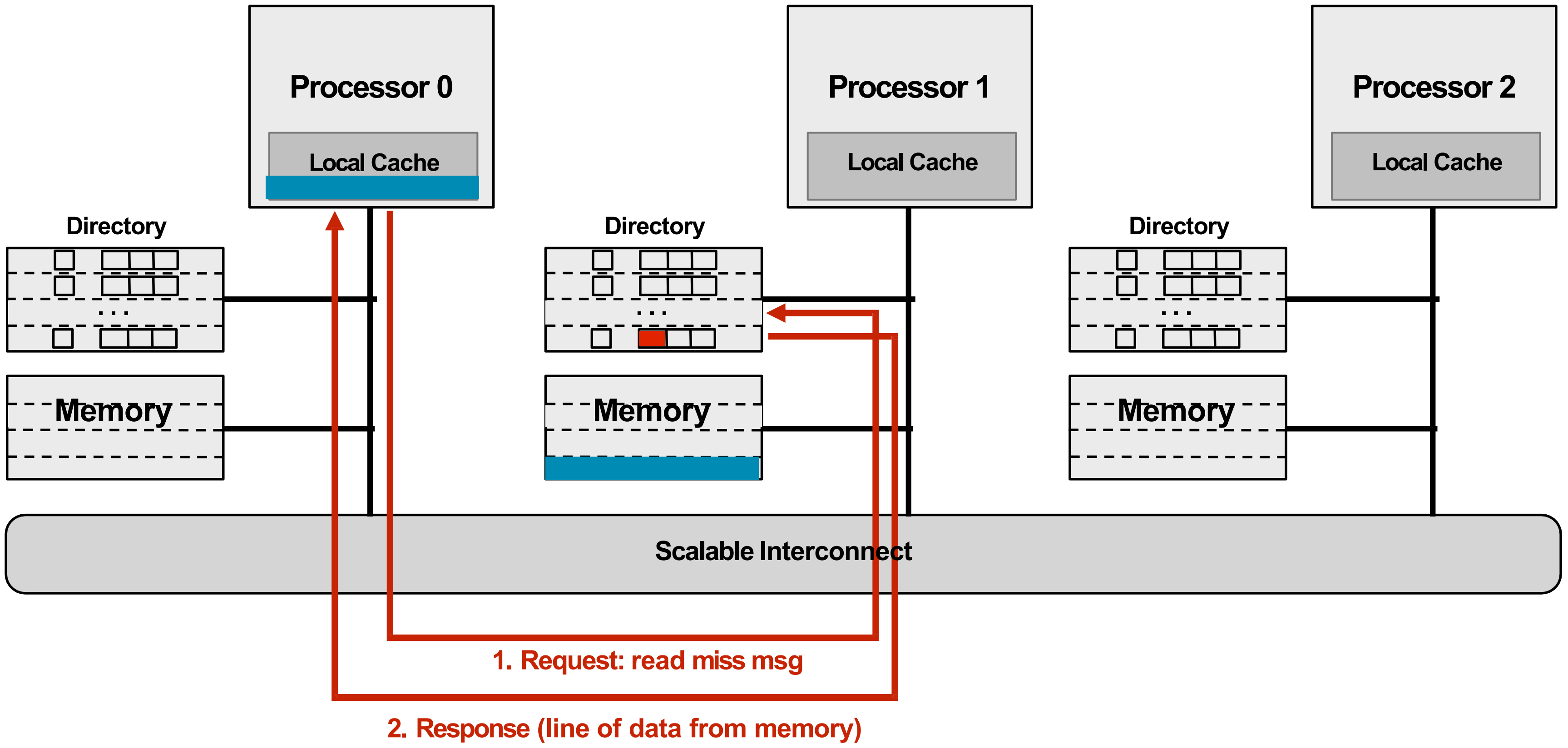
Read from main memory by processor 0 of the blue line: line is not dirty



- **Read miss message sent to home node of the requested line**
- **Home directory checks entry for line**

Example 1: read miss to clean line

Read from main memory by processor 0 of the blue line: line is not dirty

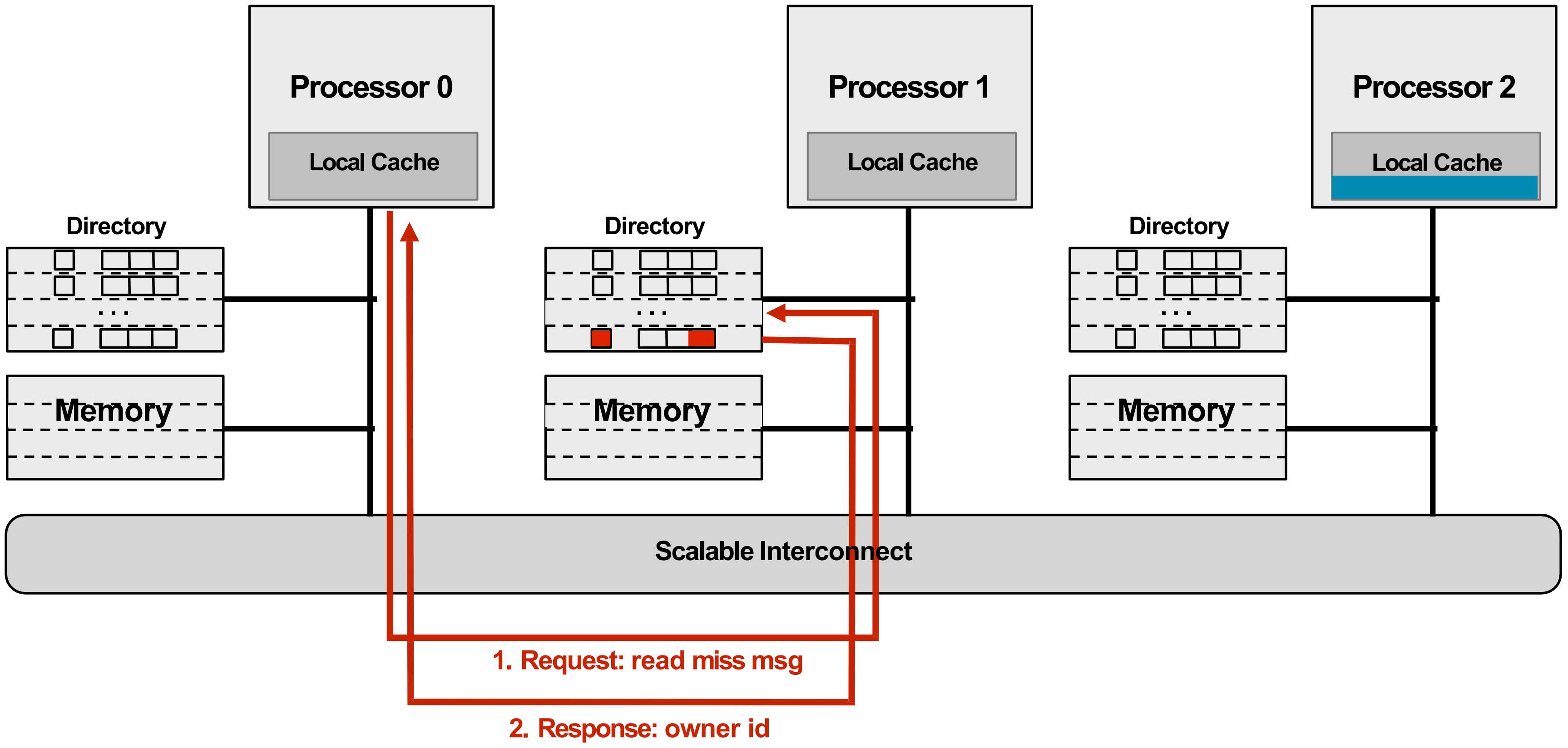


- Read miss message sent to home node of the requested line
- Home directory checks entry for line
  - If dirty bit for cache line is OFF, respond with contents from memory, set presence[0] to true

(to indicate line is cached by processor 0)

Example 2: read miss to dirty line

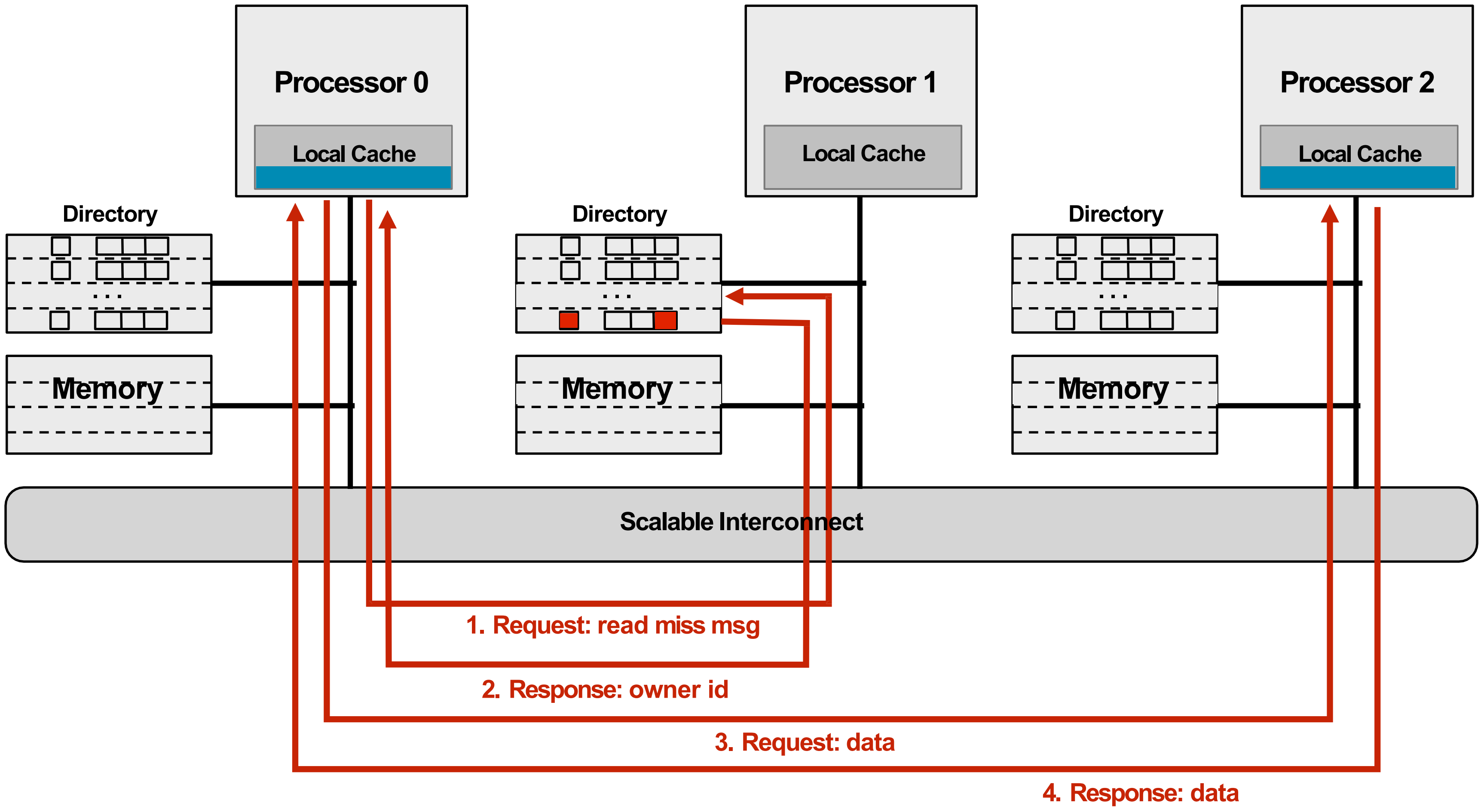
Read from main memory by processor 0 of the blue line: line is dirty (contents in P2's cache)



- **If dirty bit is ON, then data must be sourced by another processor (with the most up-to-date copy of the line)**
- **Home node must tell requesting node where to find data**
  - **Responds with message providing identity of line owner ("get it from P2")**

Example 2: read miss to dirty line

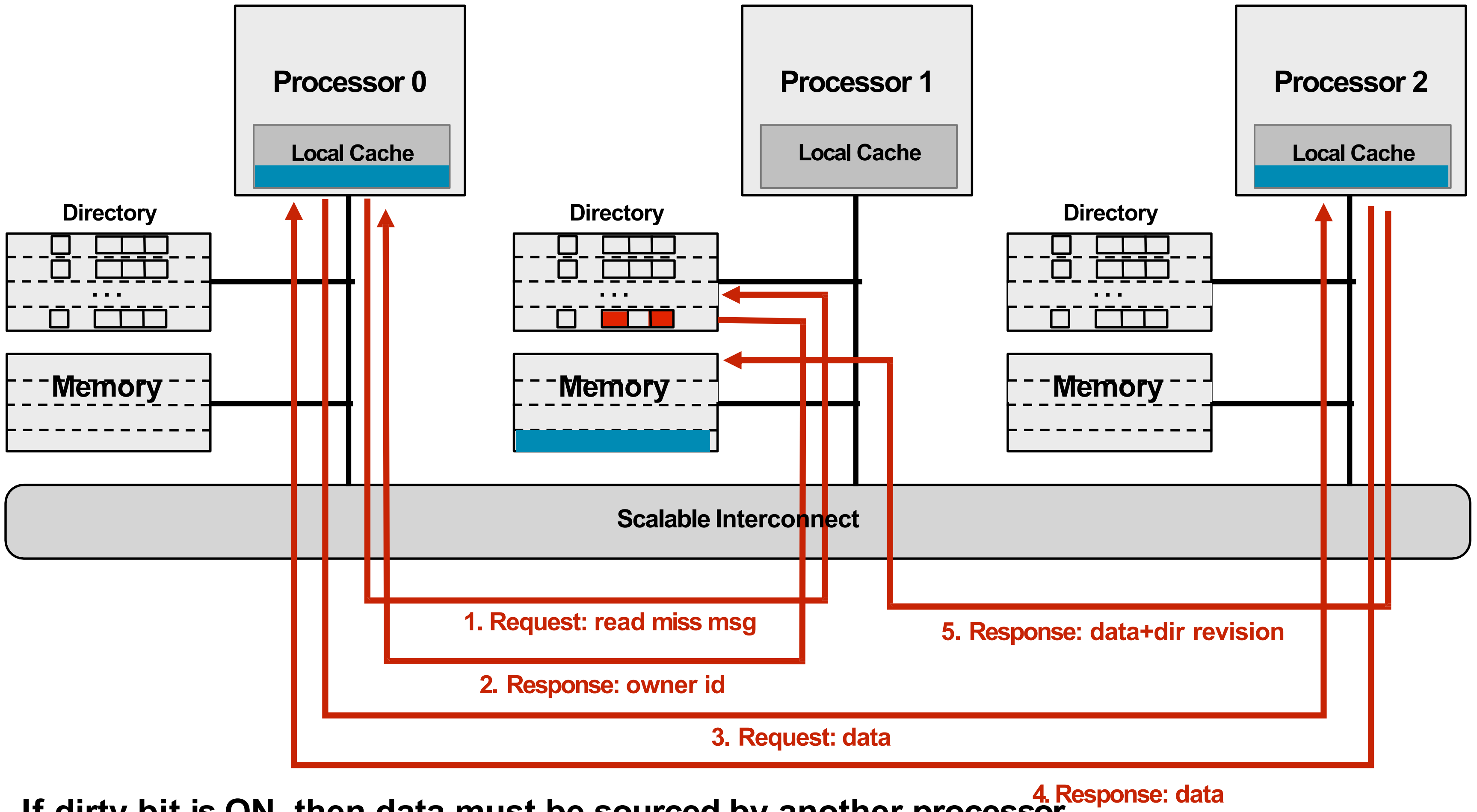
Read from main memory by processor 0 of the blue line: line is dirty (contents in P2's cache)



5. If dirty bit is ON, then data must be sourced by another processor
6. Home node responds with message providing identity of line owner
7. Requesting node requests data from owner
8. Owner changes state in cache to SHARED (read only), responds to requesting node

Example 2: read miss to dirty line

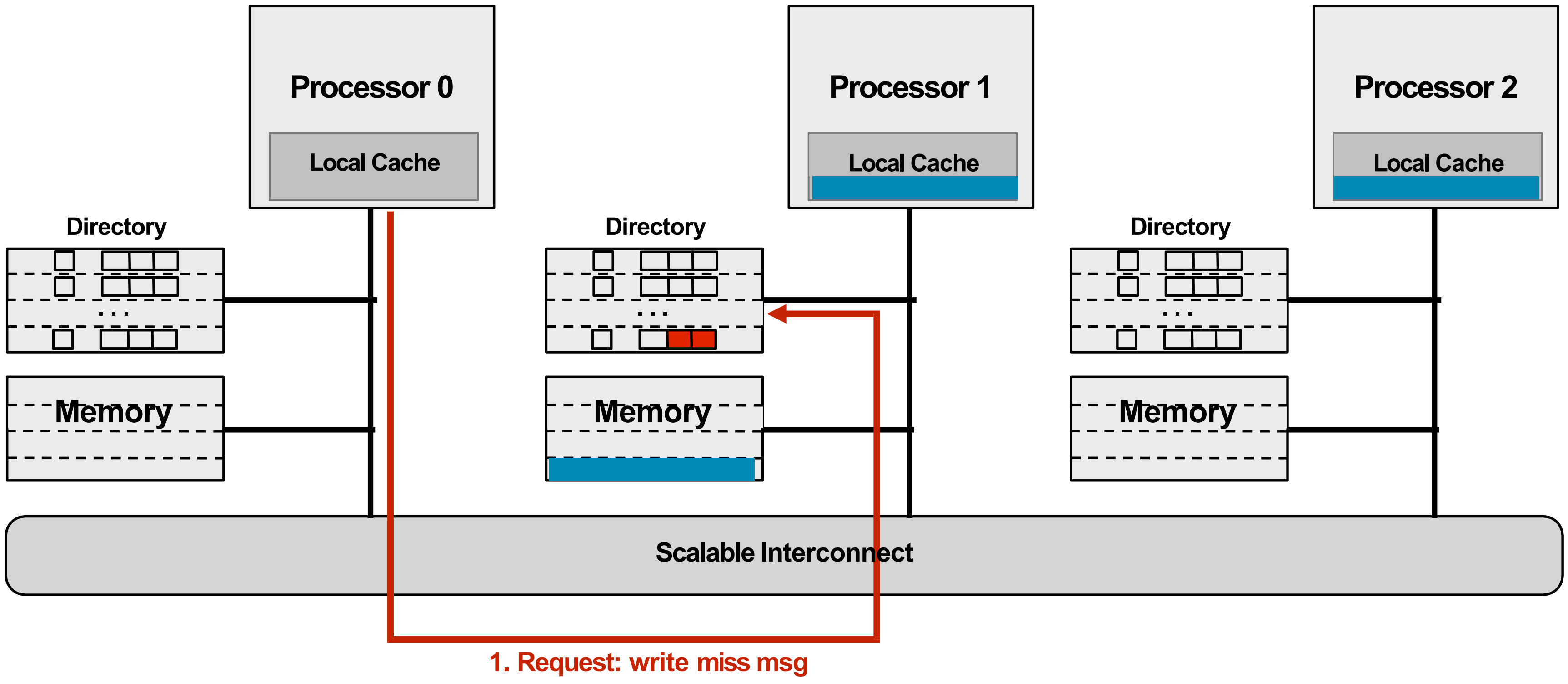
Read from main memory by processor 0 of the blue line: line is dirty (contents in P2's cache)



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

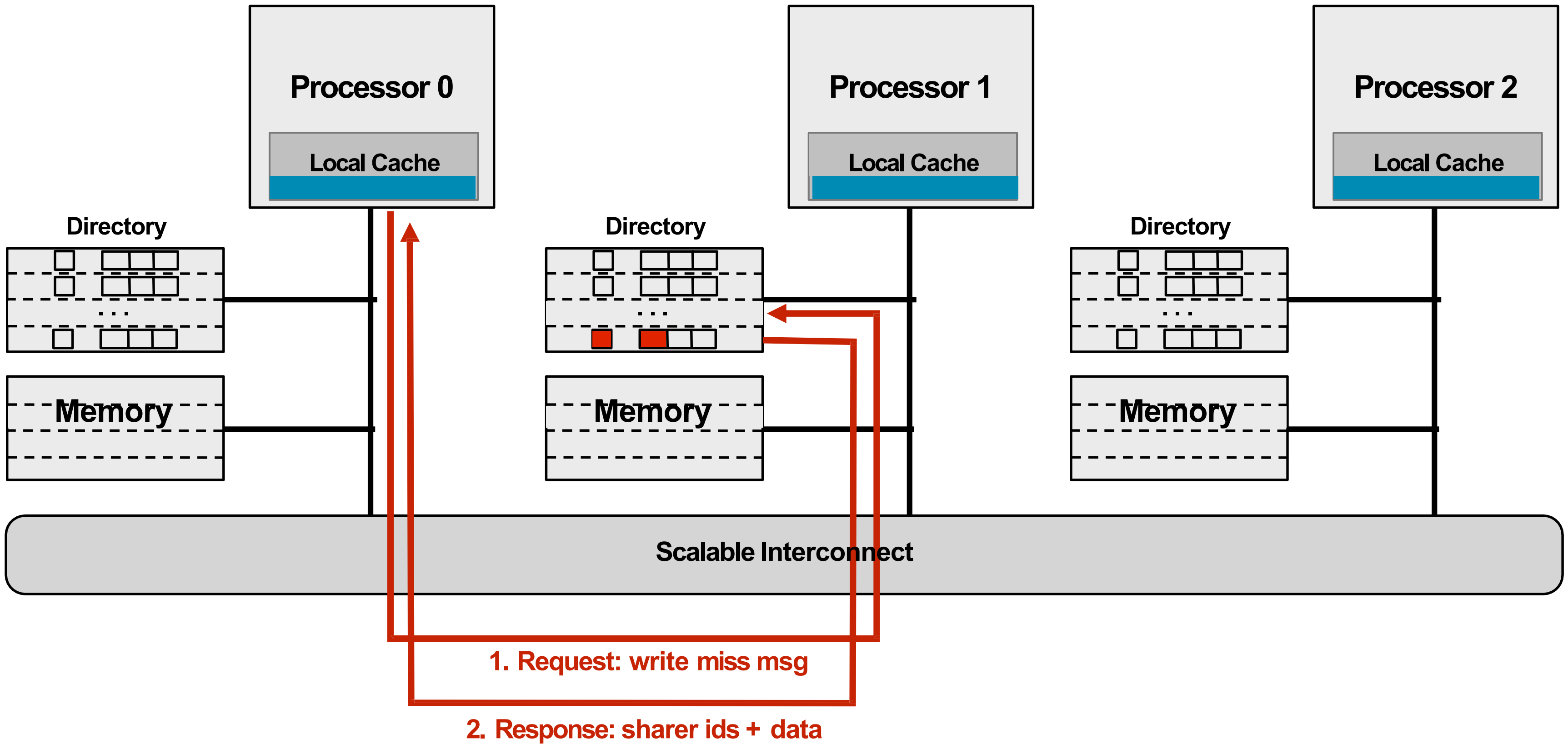
Example 3: write miss

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



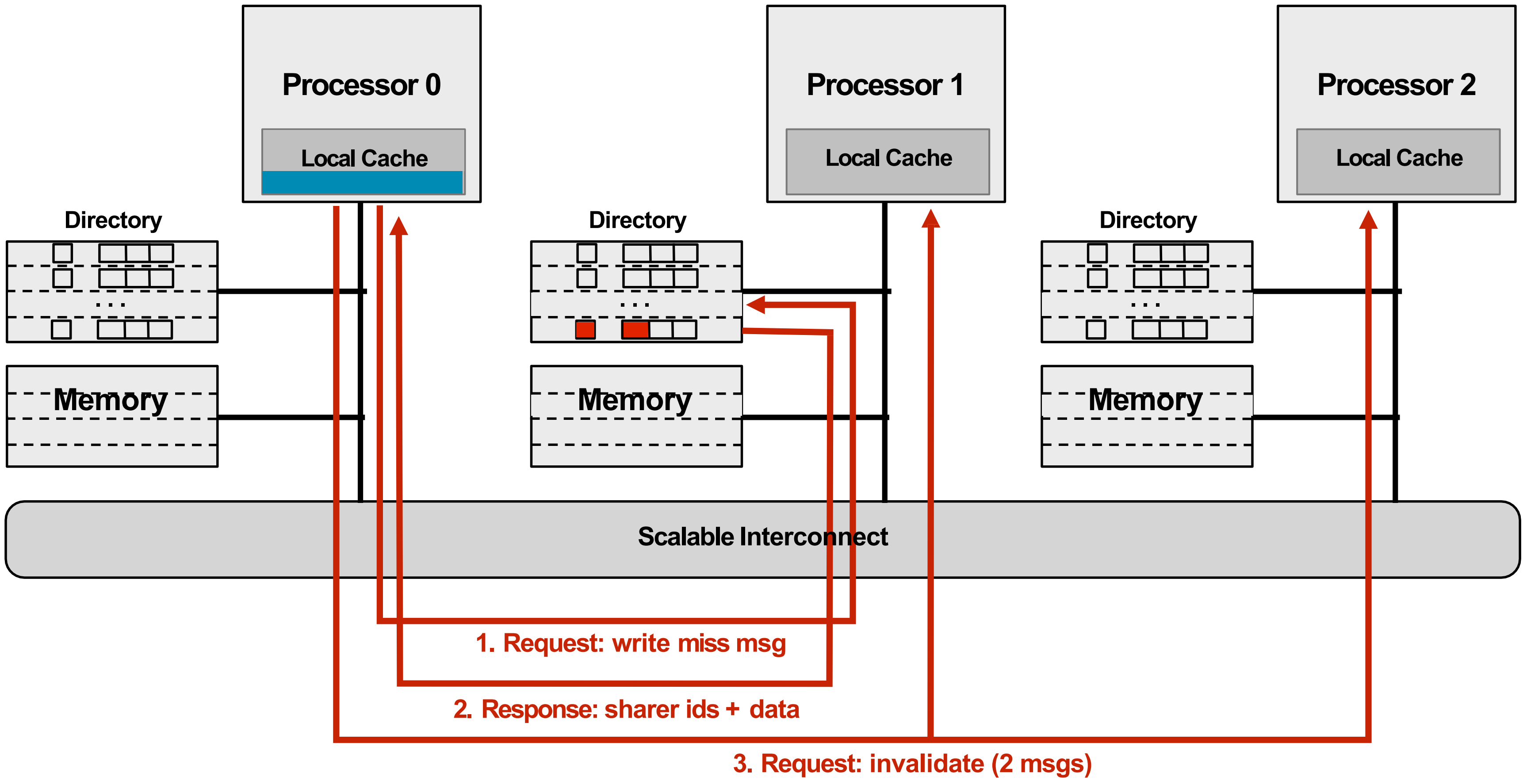
Example 3: write miss

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



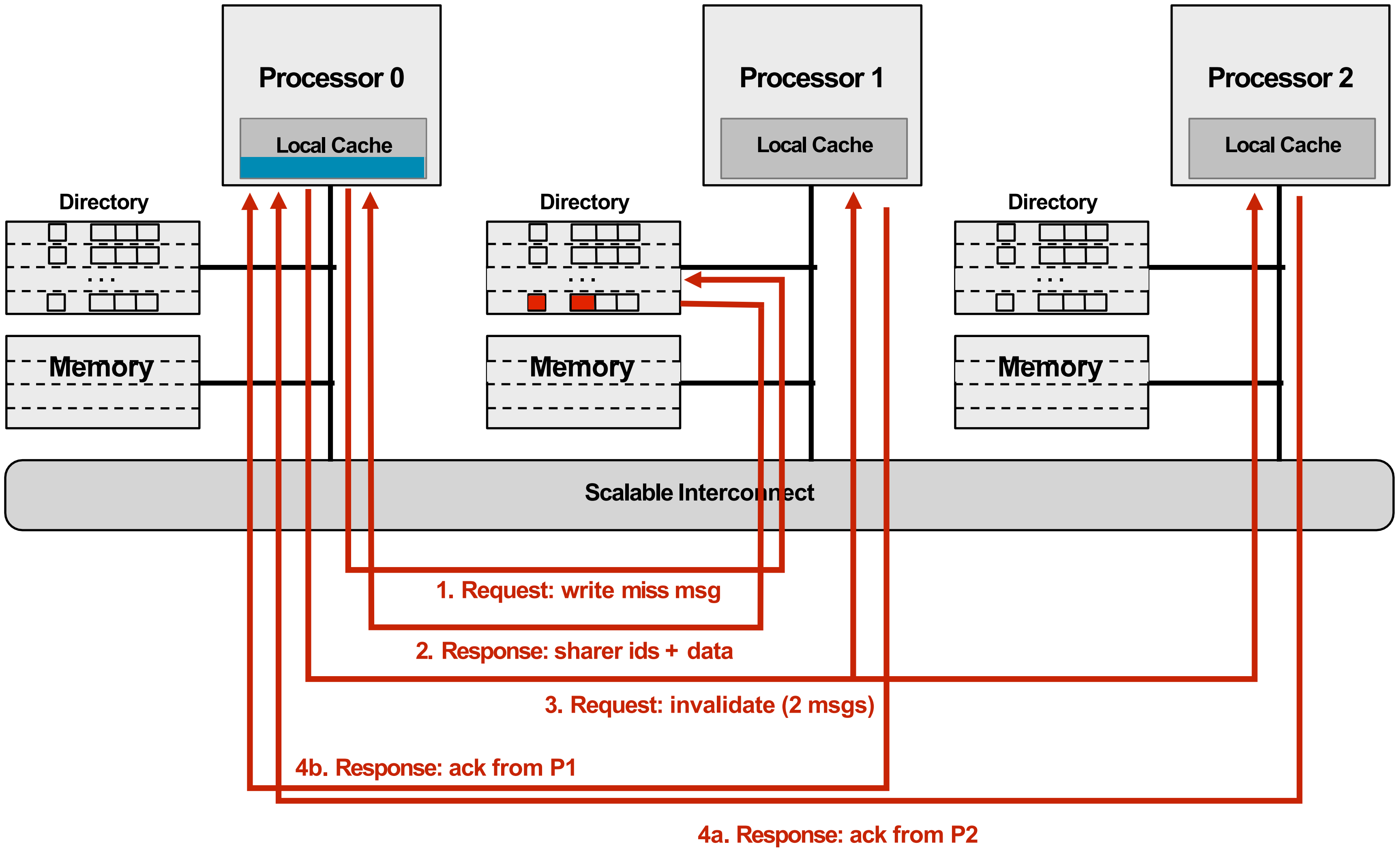
Example 3: write miss

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



Example 3: 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

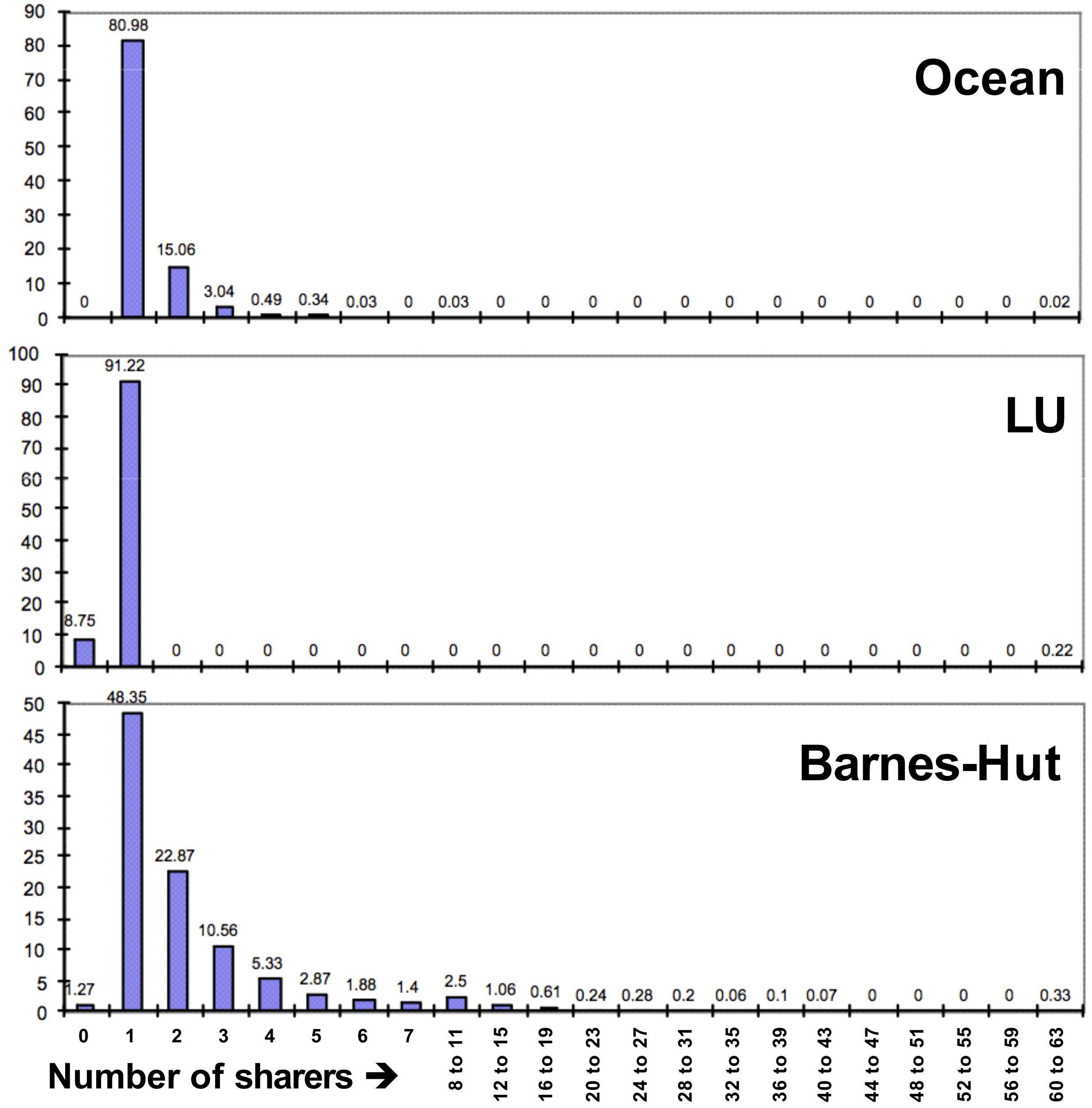
After receiving both invalidation acks, P0 can perform write

Advantage of directories

- 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 advantage 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)

Cache invalidation patterns

64 processor system



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

In general only a few processors share the line (only a few processors must be told of writes)

Not shown here, but the expected number of sharers typically increases slowly with P (good!)

In general, only a few sharers during a write

- Access patterns
  - "Mostly-read" objects: lots of sharers but writes are infrequent, so minimal impact on performance (e.g., root node in Barnes-Hut)
  - Migratory objects (one processor reads/writes for while, then another, etc.): very few sharers, count does not scale with number of processors
  - Frequently read/written objects: frequent invalidations, but sharer count is low because count cannot build up in short time between invalidations (e.g, shared task queue)
  - Low-contention locks: infrequent invalidations, no performance problem
  - High-contention locks: can be a challenge, because many readers present when lock released

- Implication 1: directories are useful for limiting coherence traffic
  - Don't need a broadcast mechanism to "tell everyone"

- Implication 2: lets us limit directory storage overhead (how?)

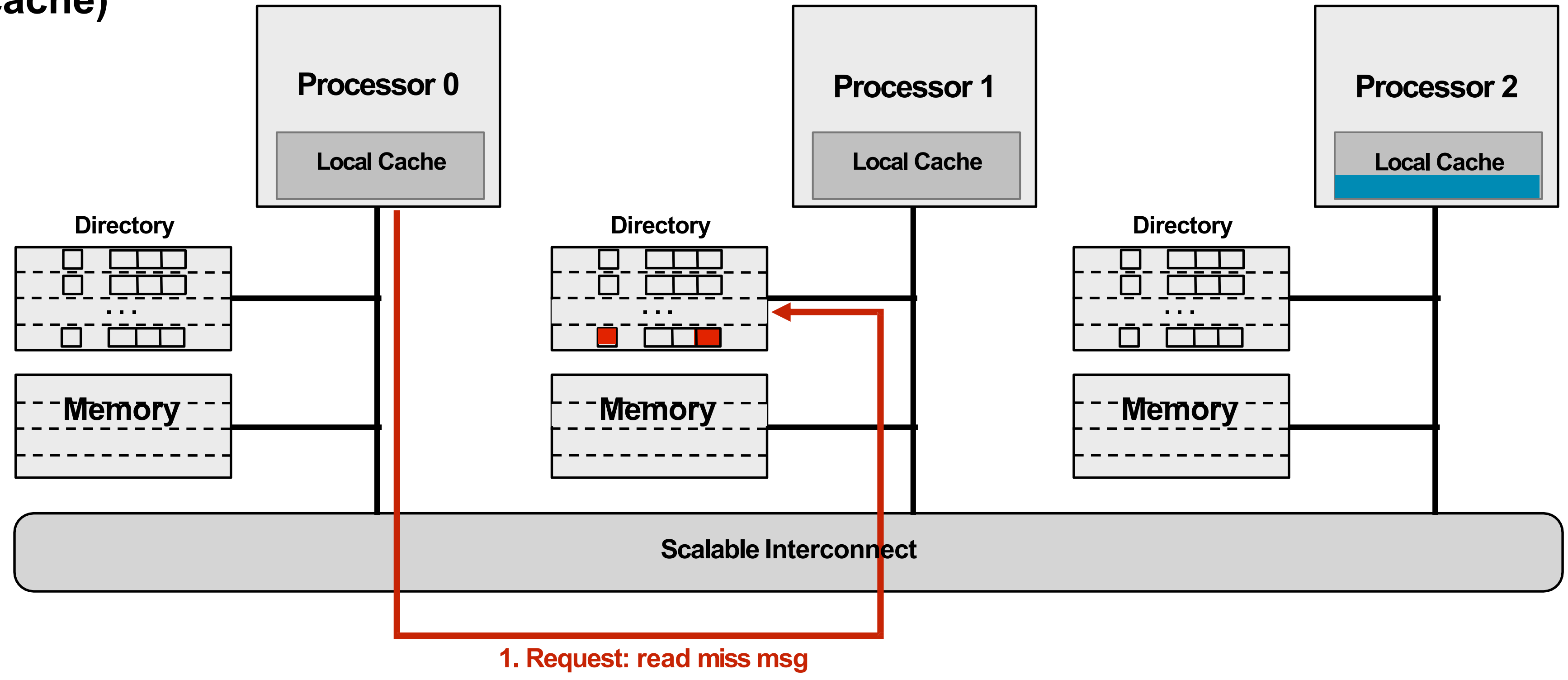
Optimizing directory-based coherence

- Reducing storage overhead of directory data structure
  - Limited pointer schemes
  - Sparse directories

- Reducing number of messages sent to implement coherence protocol

Recall: read miss to dirty line

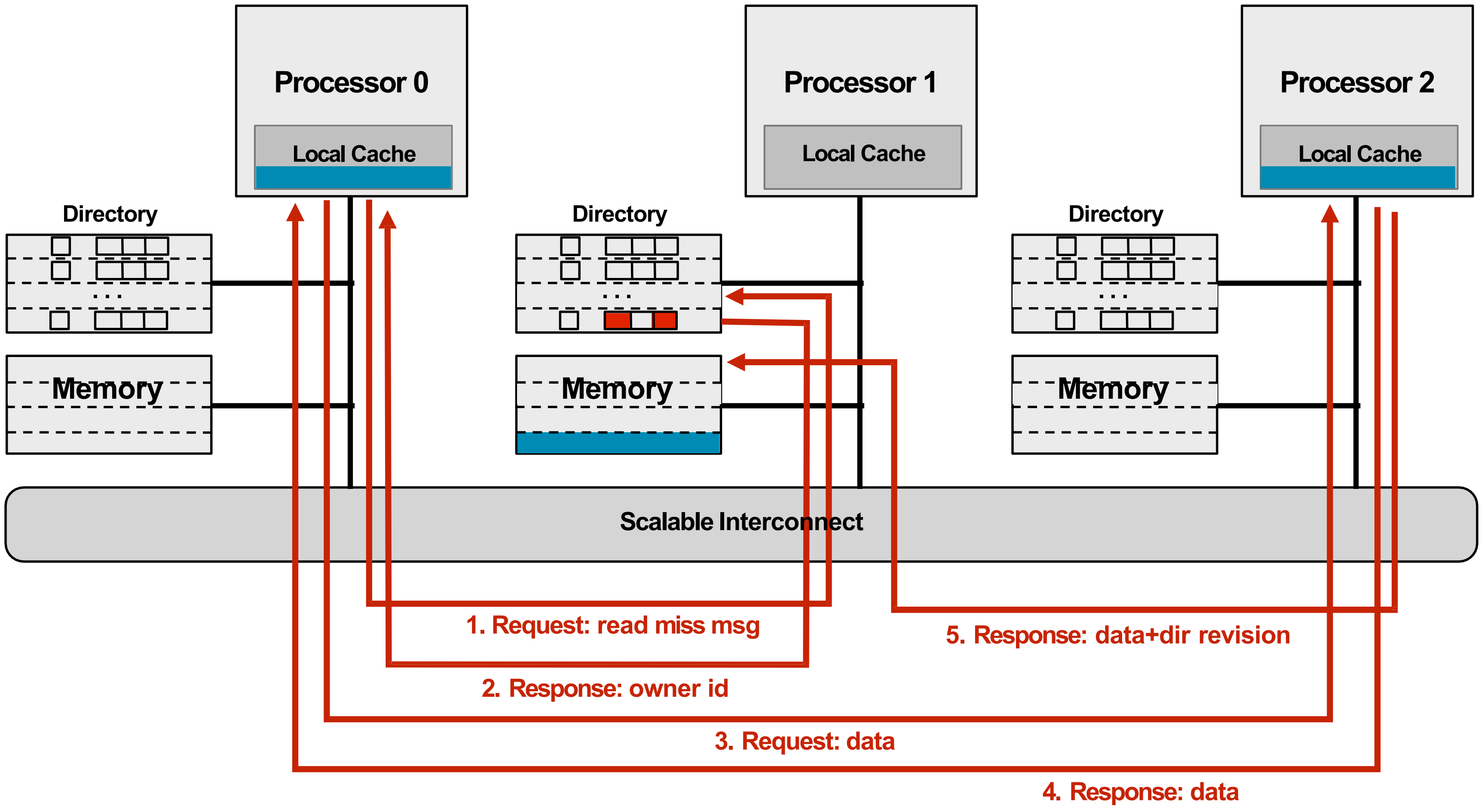
Read from main memory by processor 0 of the blue line: line is dirty (contained in P2's cache)



Recall: read miss to dirty line

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2's cache)

(Note: figure below shows final state of system after operation is complete)



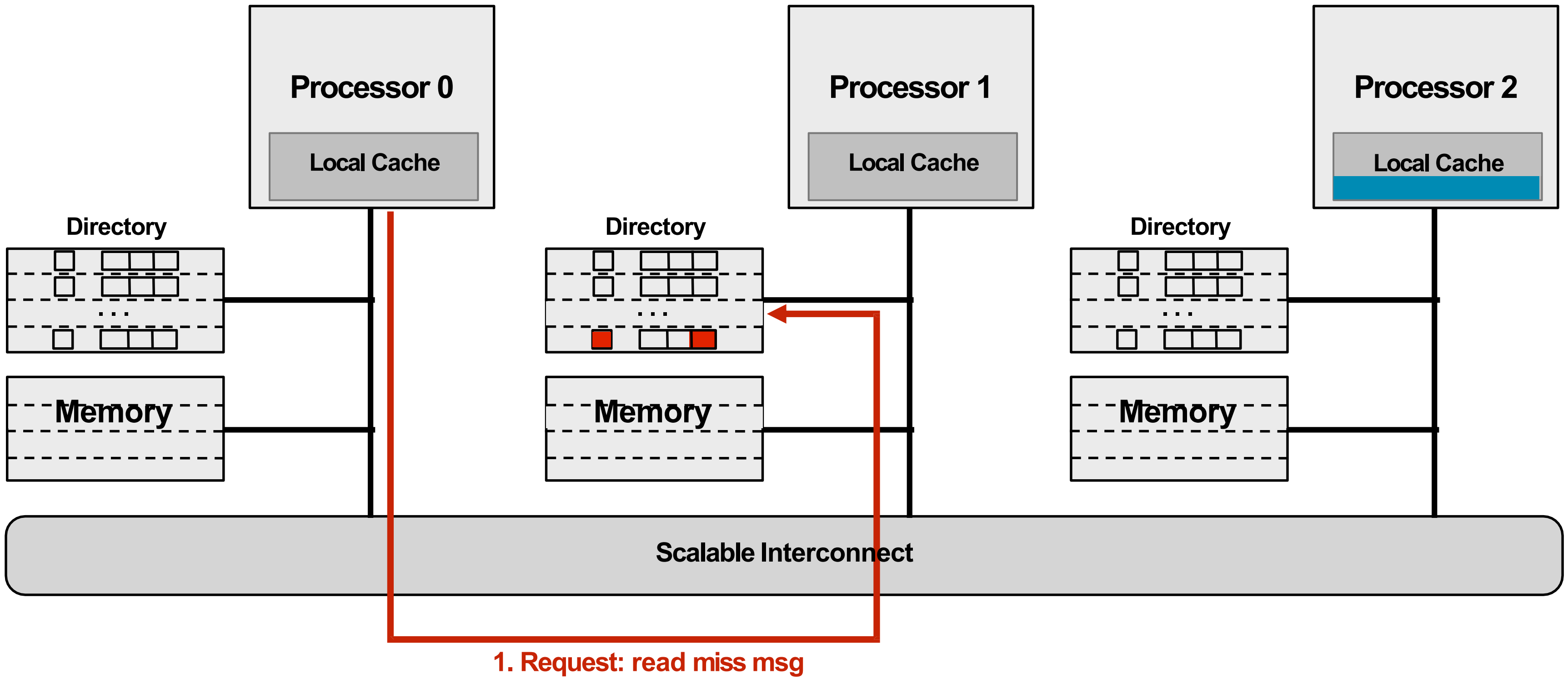
Five network transactions in total

Four of the transactions are on the "critical path" (transactions 4 and 5 can be done in parallel)

- Critical path: sequence of dependent operations that must occur to complete operation

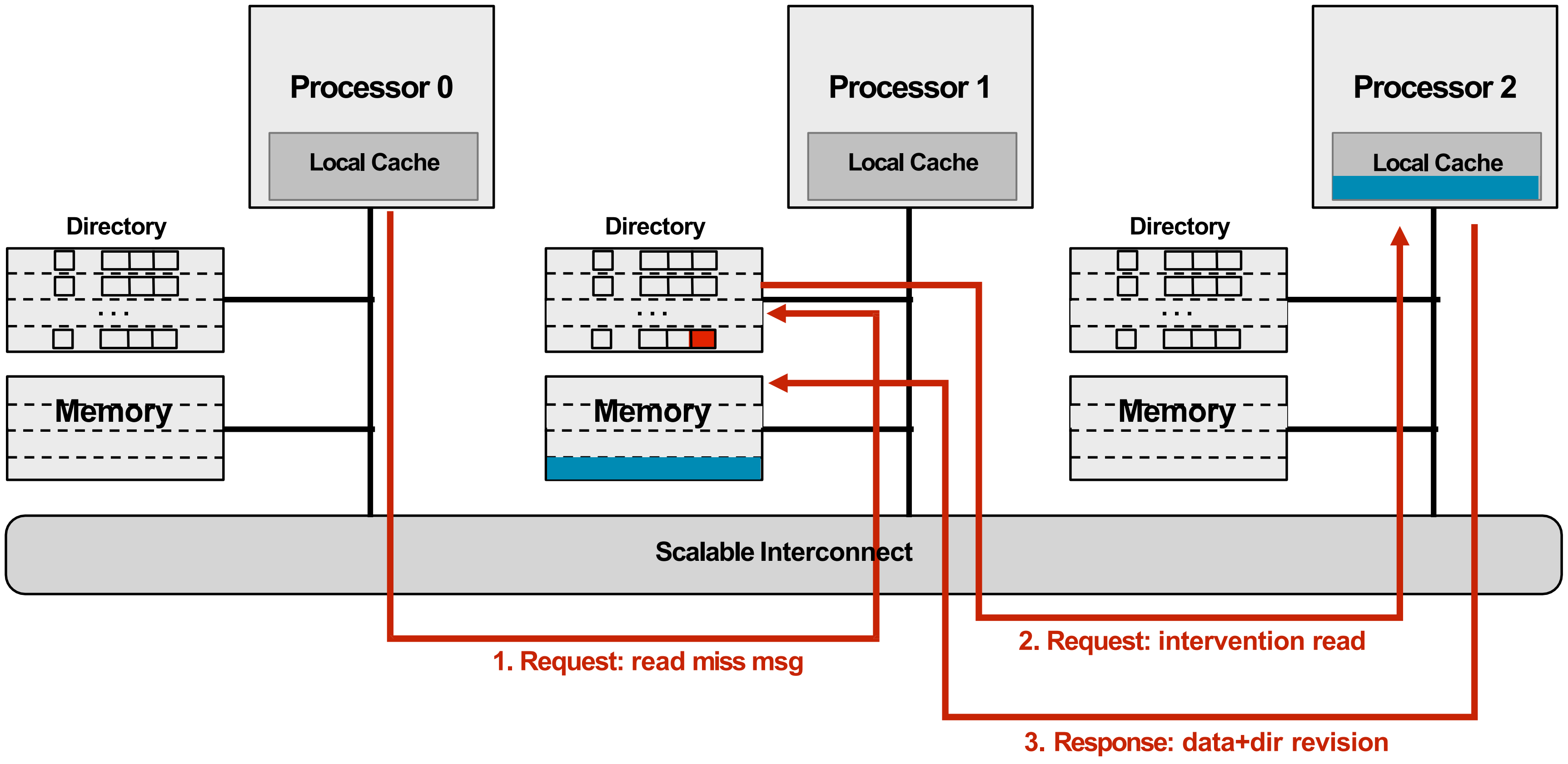
Intervention forwarding

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2's cache)



Intervention forwarding

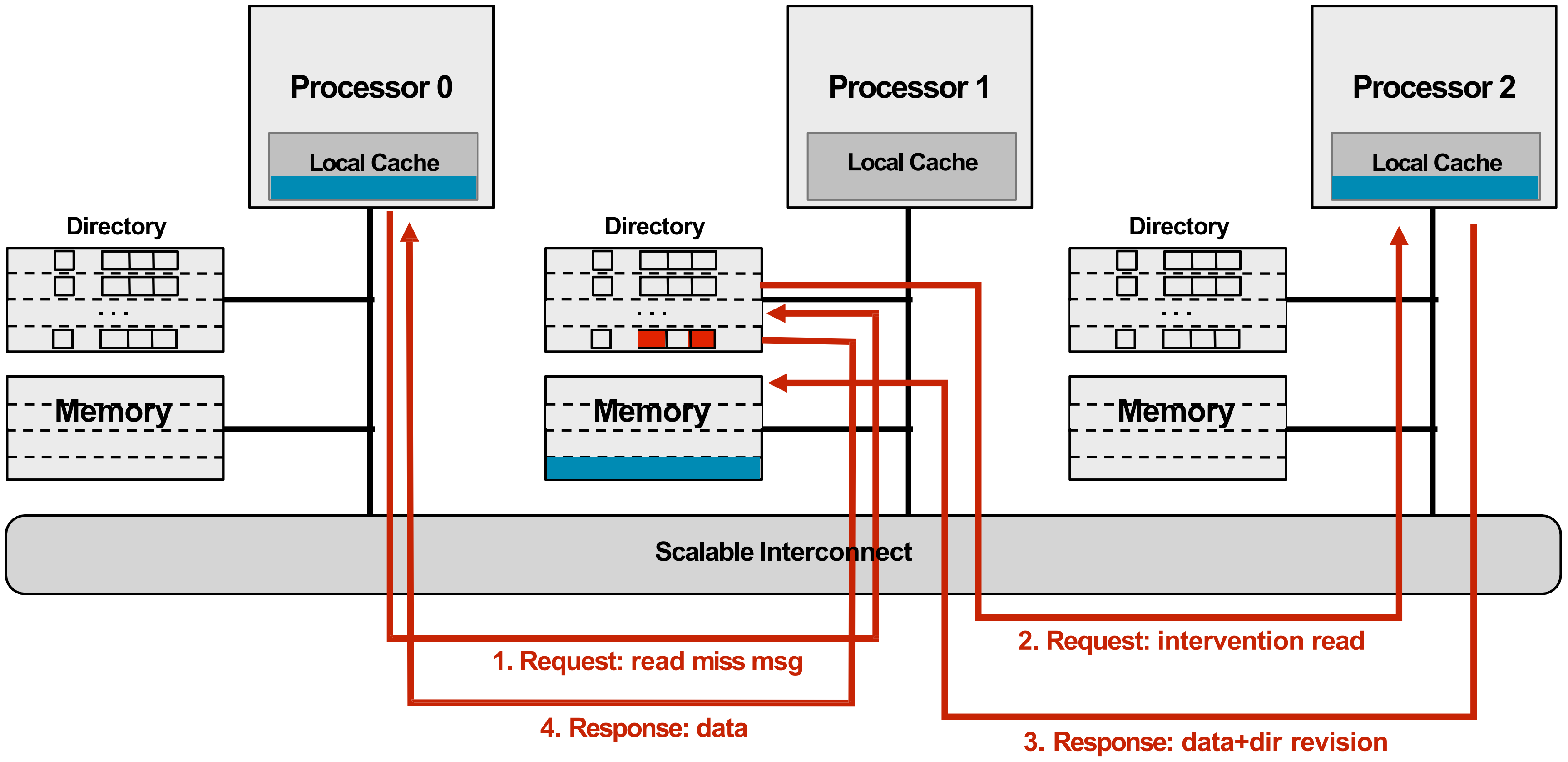
Read from main memory by processor 0 of the blue line: line is dirty (contained in P2's cache)



2. **Home node requests data from owner node (processor 2)**
3. **Owning node responds**

Intervention forwarding

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2's cache)



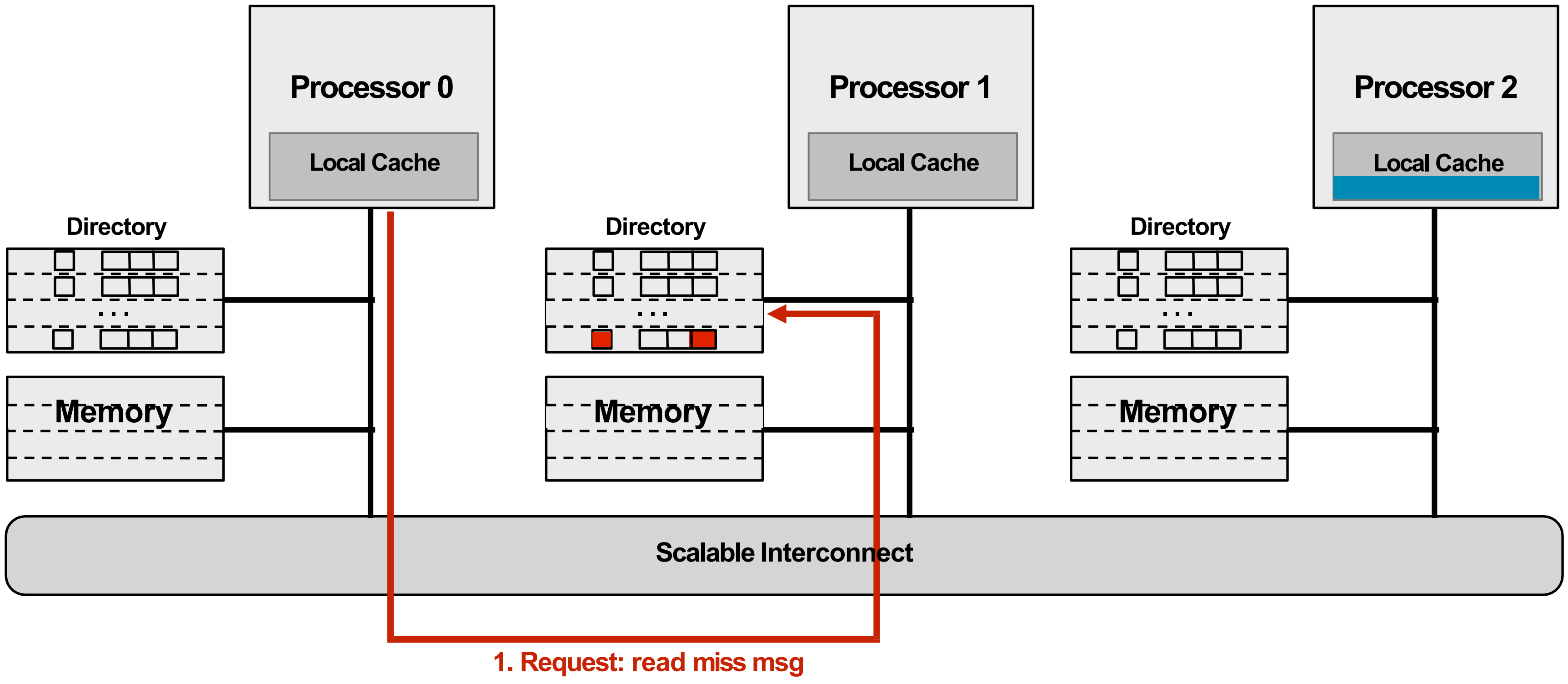
4. Home node updates directory, and responds to requesting node with data

Four network transactions in total (less traffic)

But all four of the transactions are on the "critical path." Can we do better?

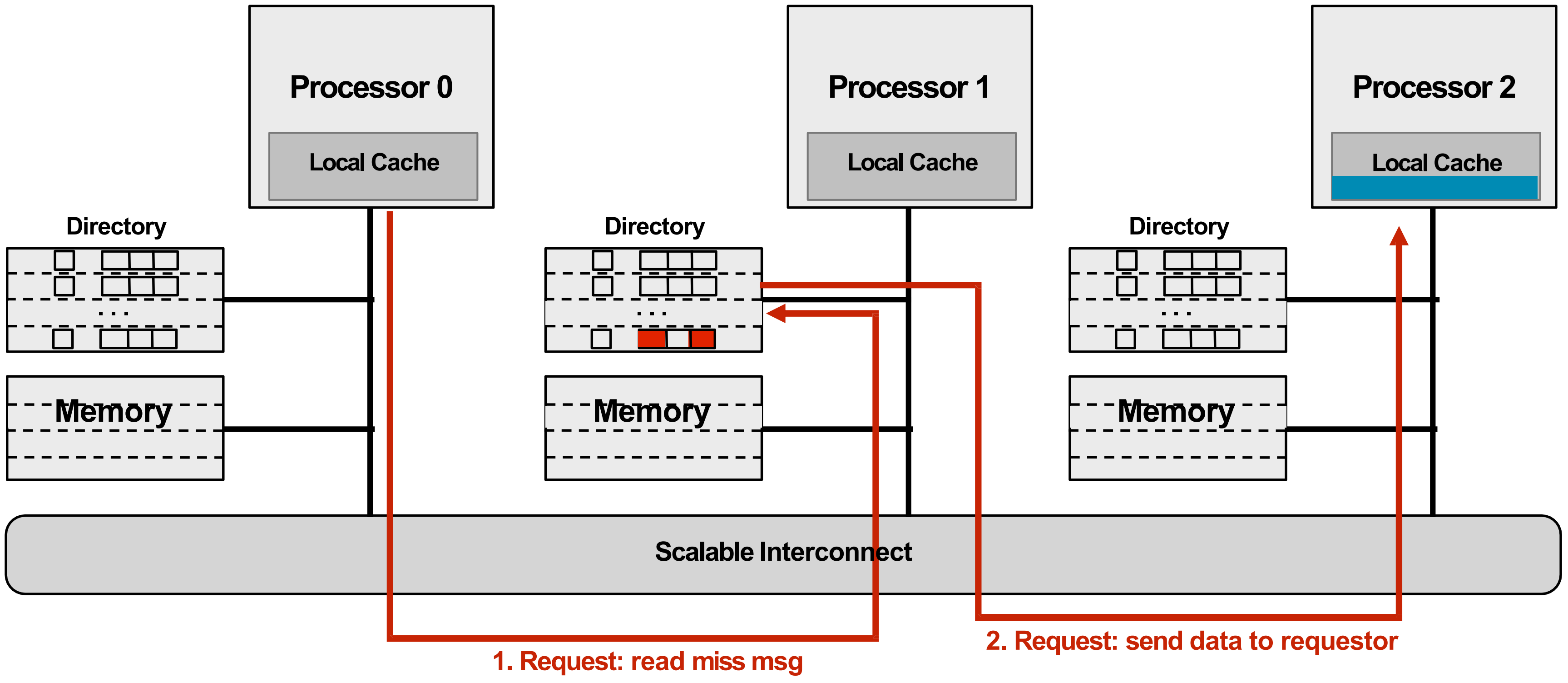
Request forwarding

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2's cache)



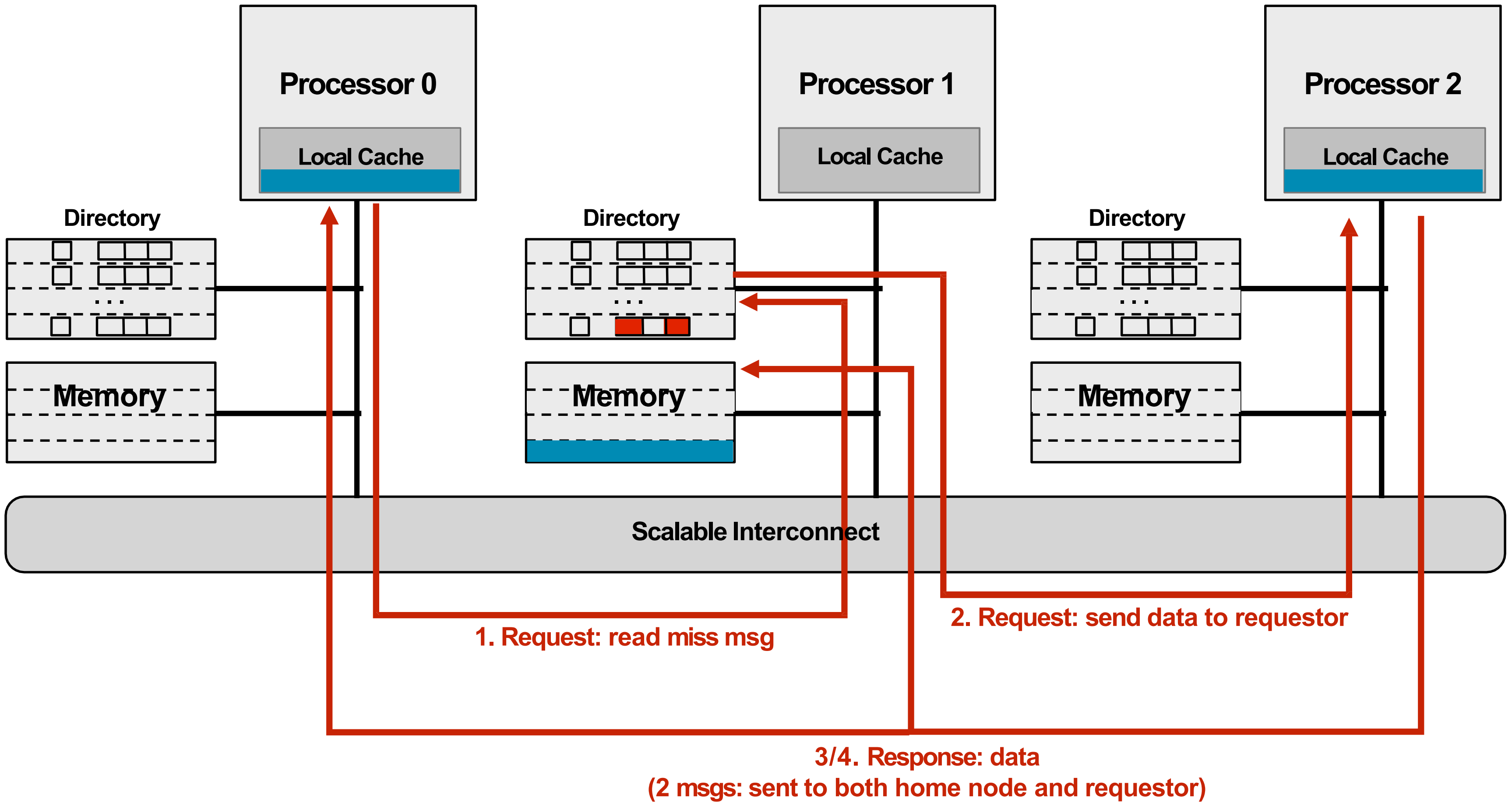
Request forwarding

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2's cache)



Request forwarding

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2's cache)

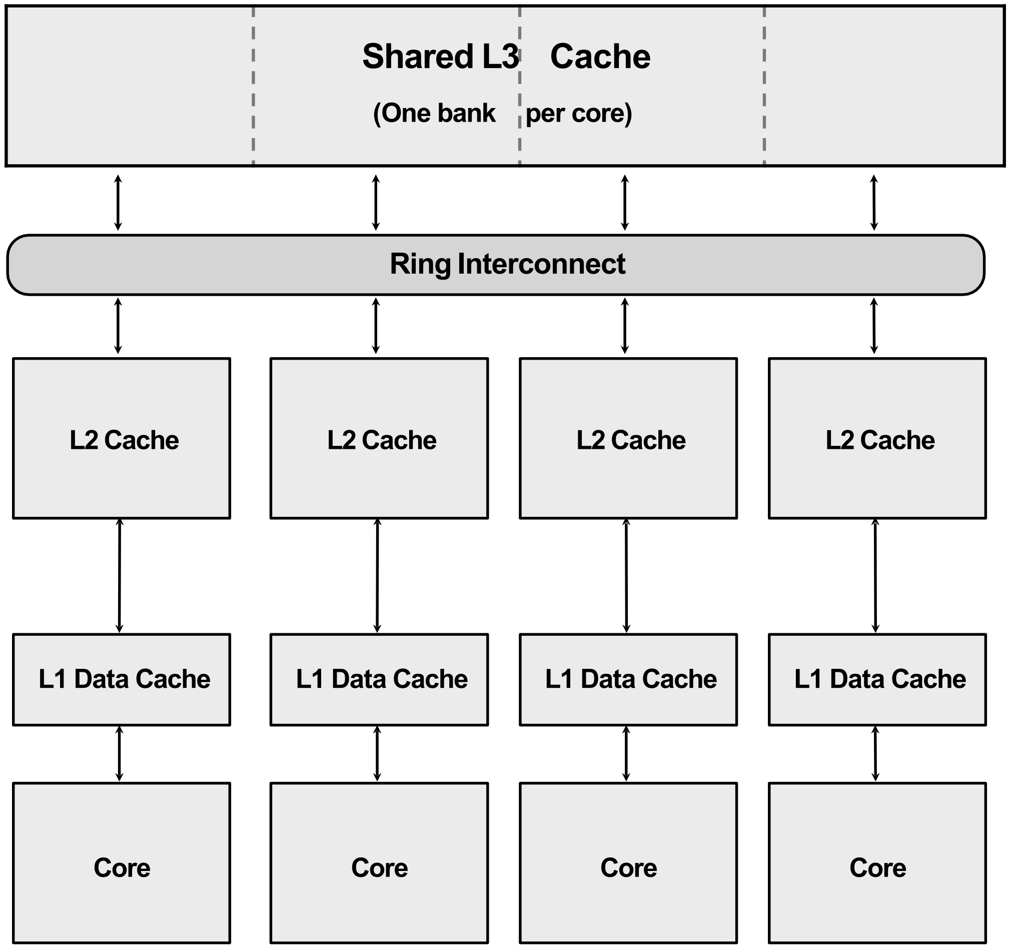


Four network transactions in total

Only three of the transactions are on the critical path (transactions 3 and 4 can be done in parallel)

Note: system is no longer pure request/response (since P0 sent request to P1, but receives response from P2)

Directory coherence in Intel Core i7 CPU

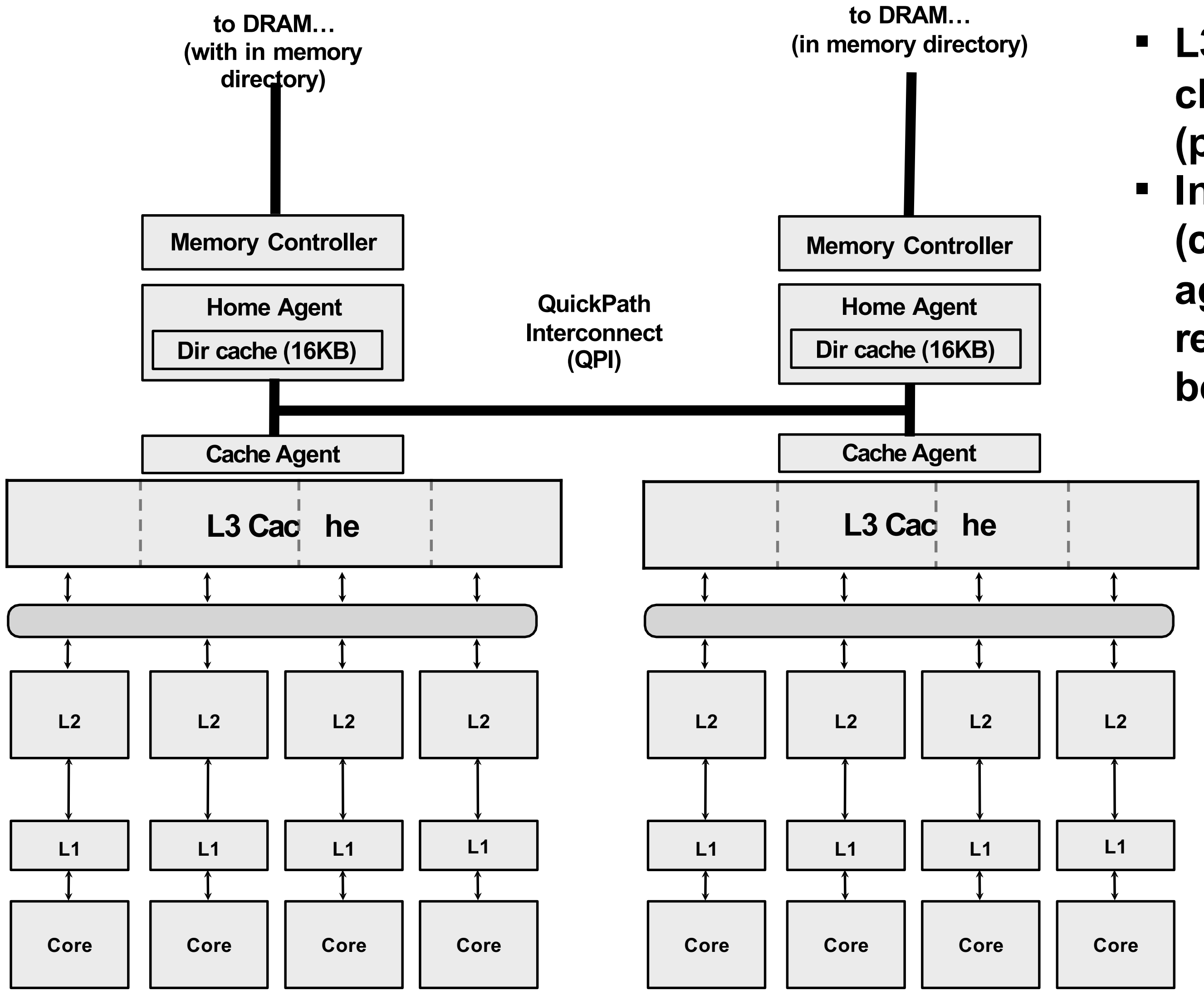


- **L3 hosts in-cache directory (and is inclusive)**
- **Directory maintains list of L2 caches containing line**
- **Instead of broadcasting coherence traffic to all L2's, only send coherence messages to L2's that contain the line**

(Core i7 interconnect is a ring, it is not a bus)

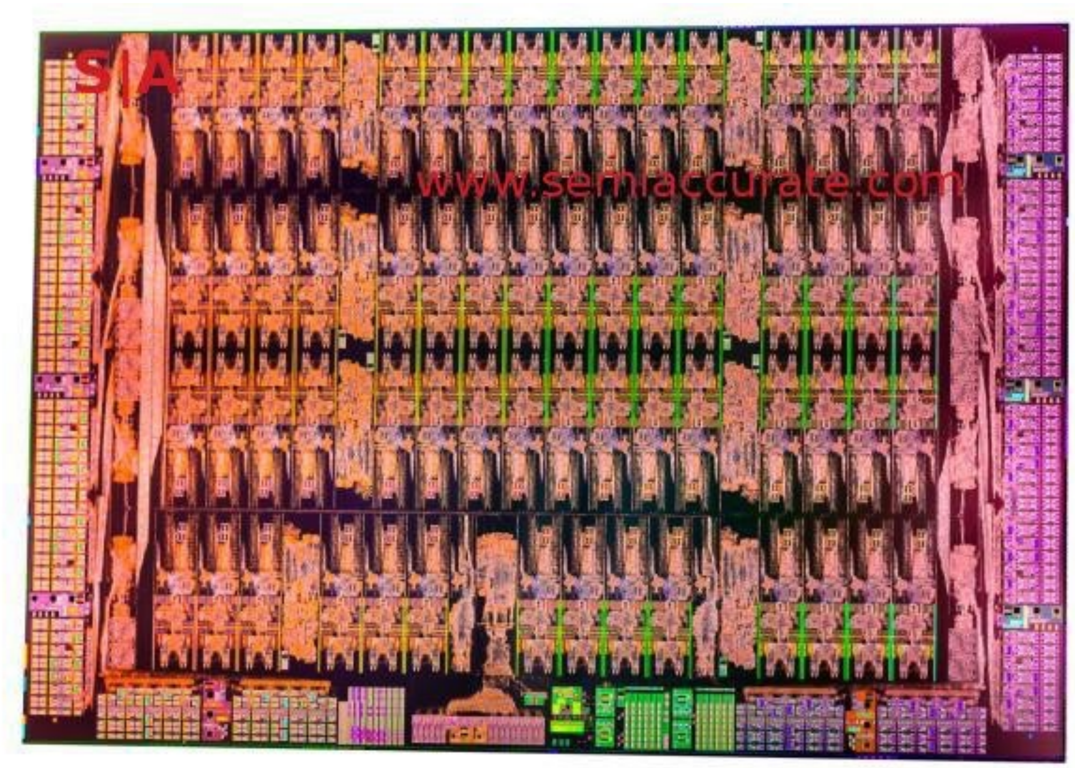
- **Directory dimensions:**
  - **P=4**
  - **M = number of L3 cache lines**

Coherence in multi-socket Intel systems



- L3 directory reduces on-chip coherence traffic (previous slide)
- In-memory directory (cached by home agent/memory controller) reduces coherence traffic between cores

Xeon Phi

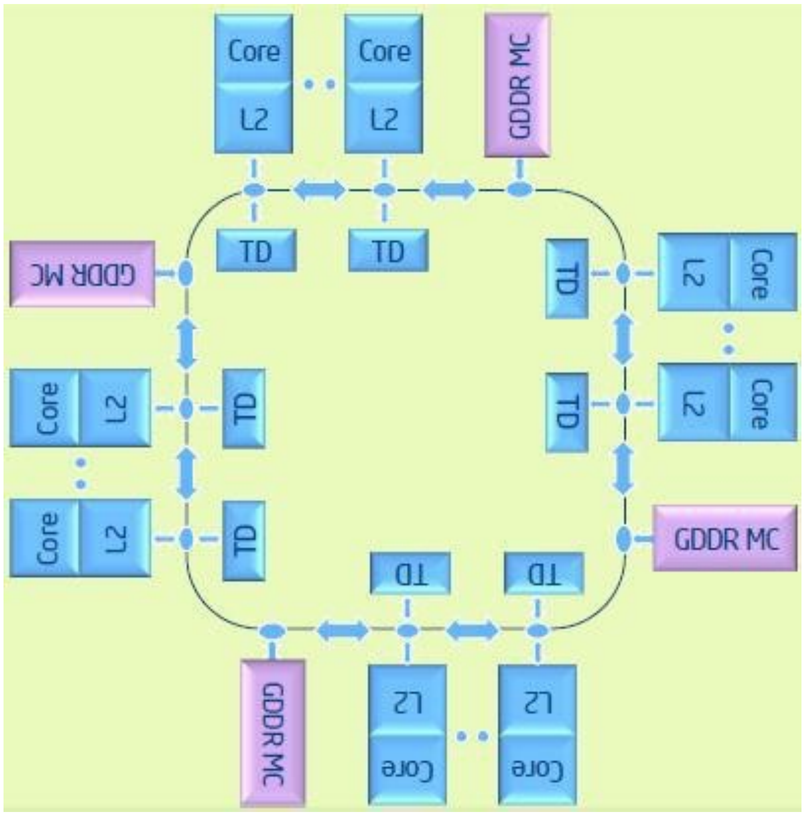


Prototype / internal names include:
Larrabee, Knight's Ferry, Knight's Corner, Knight's Landing, Knight's Hill

China's Tianhe-2 has 48,000 Knight's Corner chips

- Intel's NUMA on a chip
- Many (50+) x86 cores
  - Ours have 61
  - "Knight's Corner"
  - 4-way hyper threading
  - Each with 1–2 vector units
- Cache-coherent memory system
- Knight's Corner overall system:
  - Max. 8GB memory
  - Max. 2 TFLOPS
  - 0.004 bytes/flop
    - not balanced
  - 300 Watts

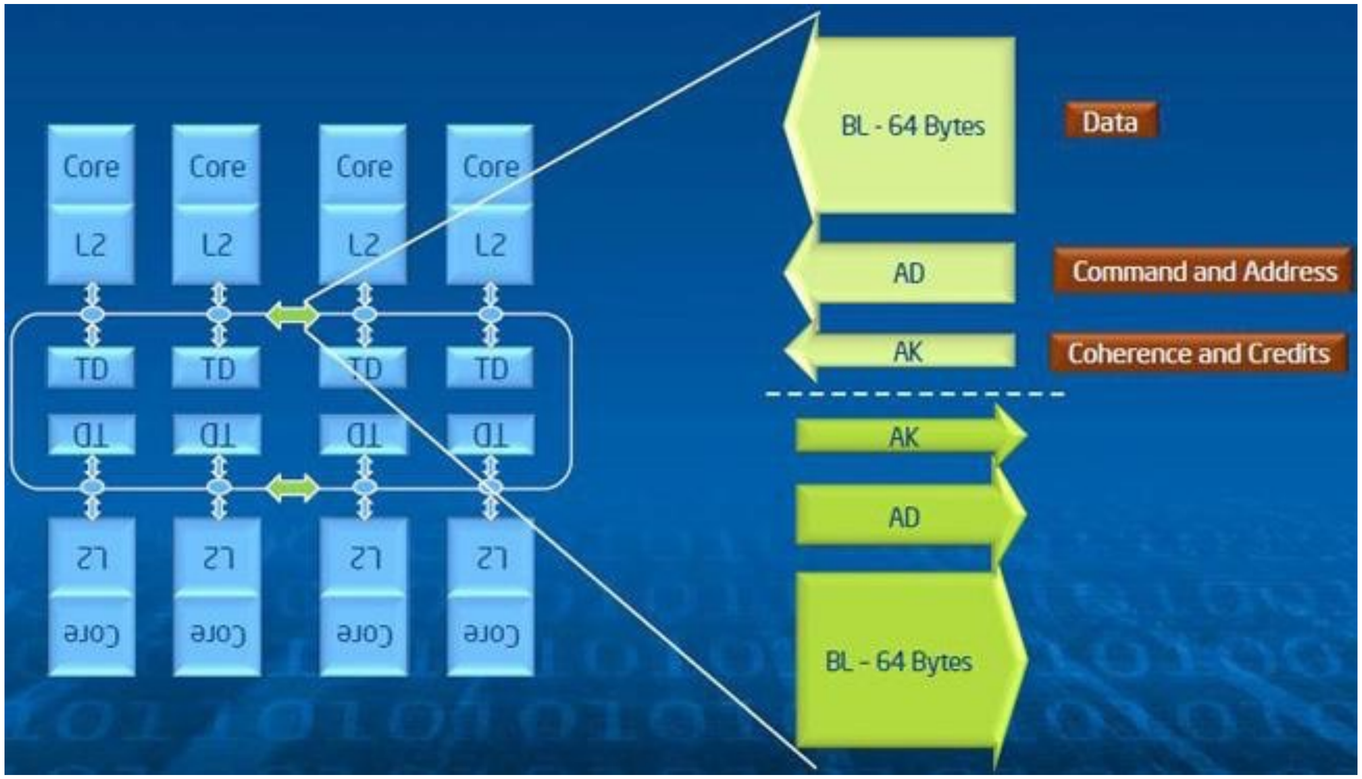
Knight's Corner Xeon Phi Cache Coherence



- 512KB L2 caches
- 8 memory controllers
    - Total 8GB max.

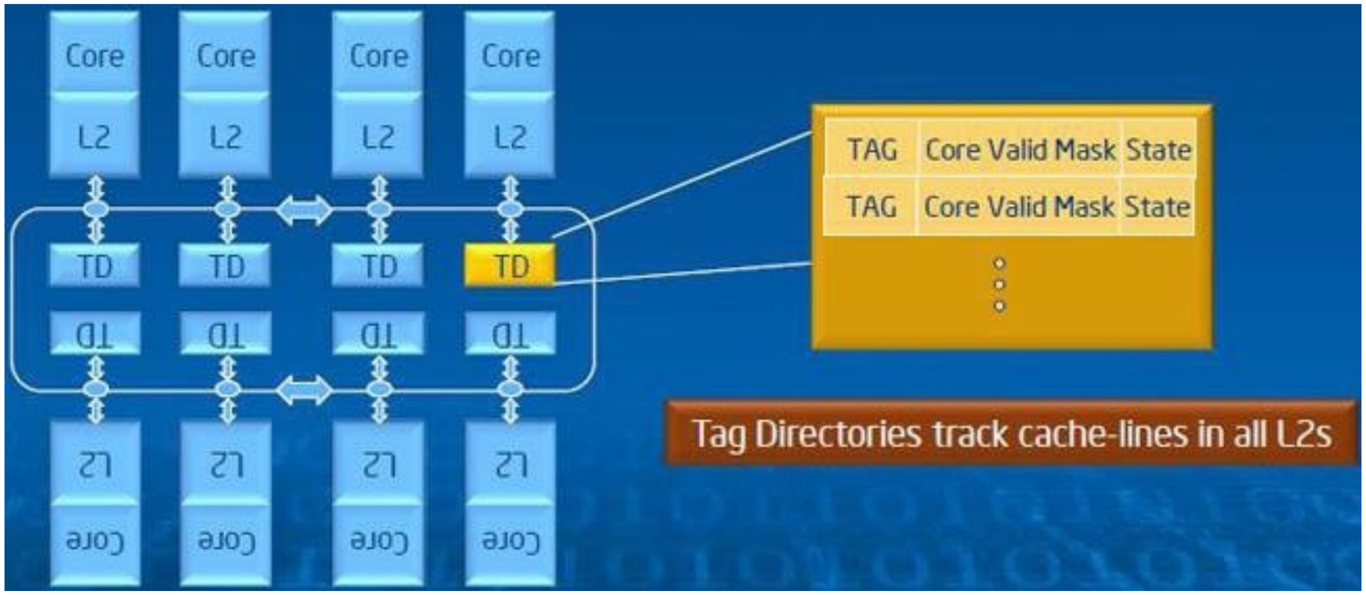
Prototype / internal names include: Larrabee, Knight's Ferry, Knight's Corner, Knight's Landing, Knight's Hill

KC Xeon Phi Ring Communication



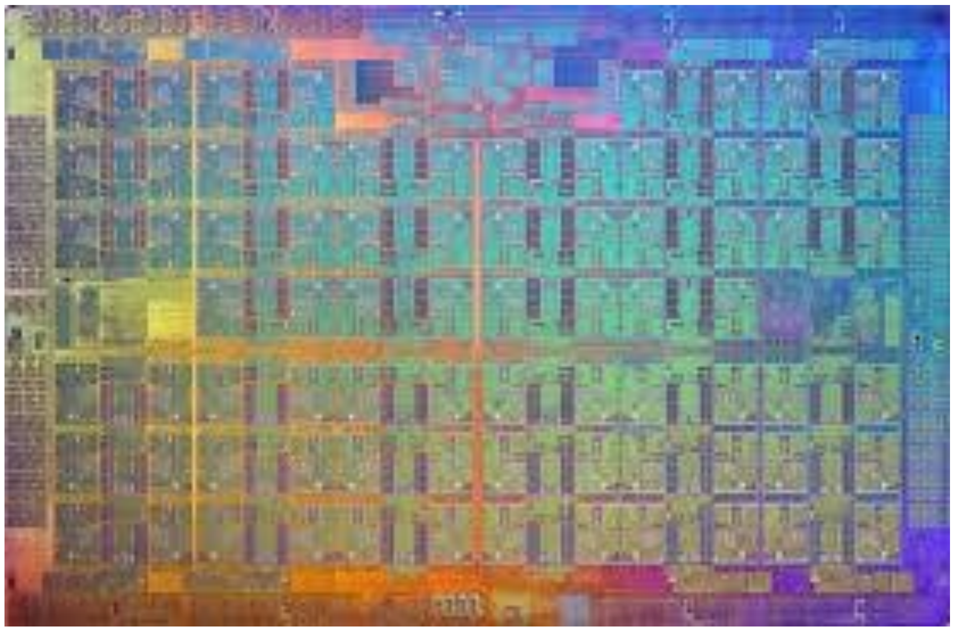
- Messages sent around bidirectional ring
  - Having everything on single chip enables very wide communication paths
  - Can get effect of broadcast by circulating message around entire ring
    - Advantage over point-to-point

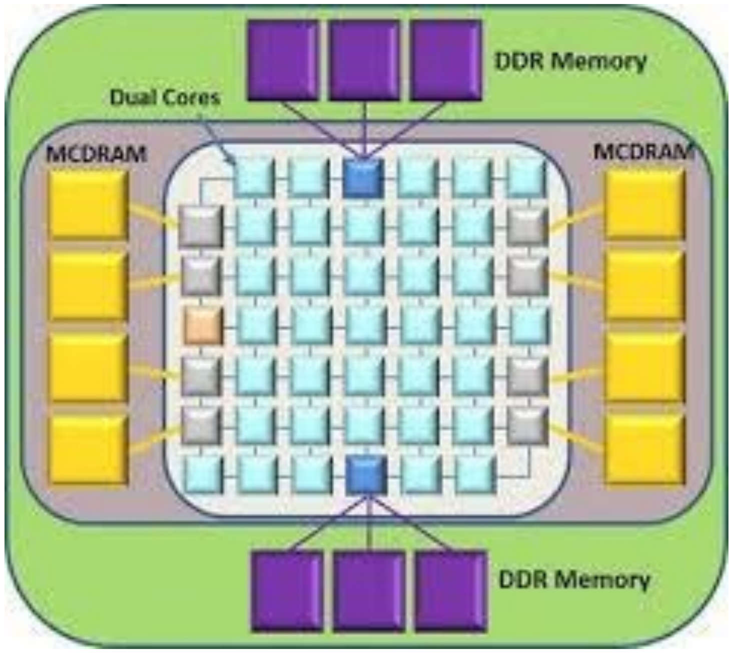
KC Xeon Phi Directory Structure


Core
L2
TD
TAG Core Valid Mask State
TAG Core Valid Mask State
Tag Directories track cache-lines in all L2s

- Directory keeps track of which lines are resident in local L2
  - Same as with single-node system
- Worst-case memory read or write by P:
  1. Check local cache
  2. Circulate request around ring for line in some cache
  3. Send request around ring to memory controller

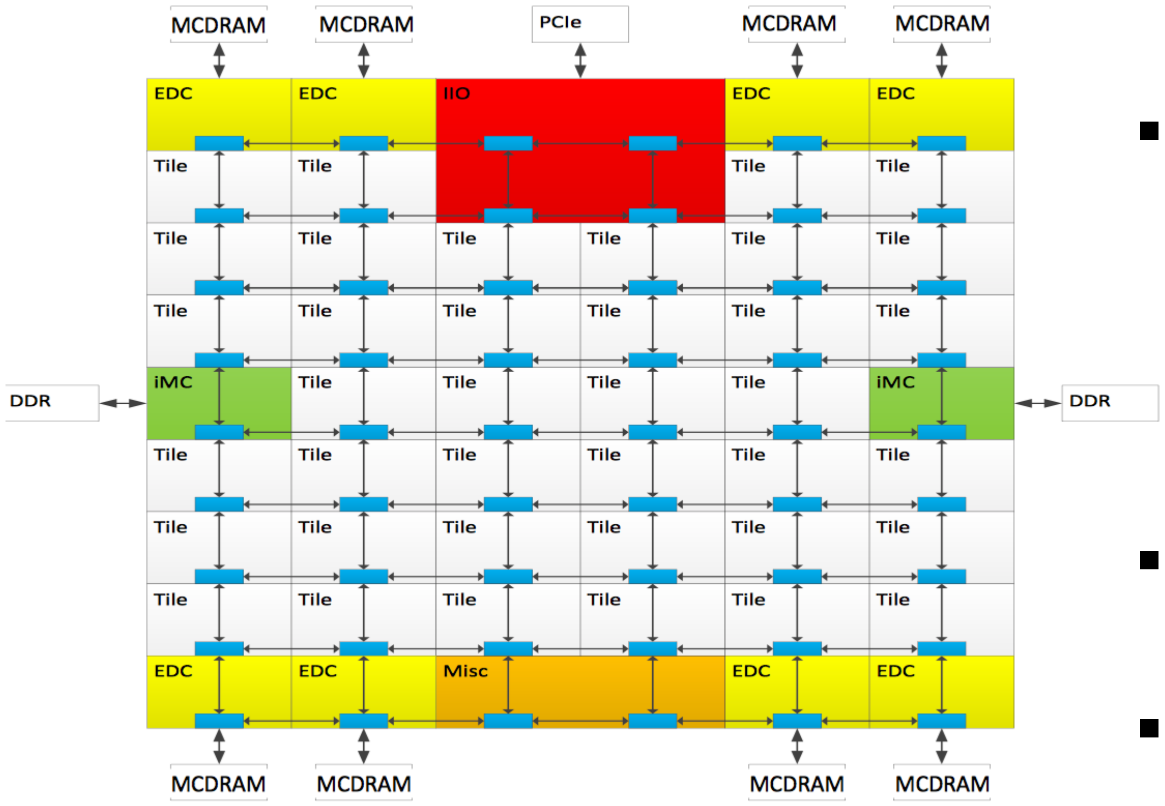
Next Generation Xeon Phi





- "Knight's Landing"
- 72 cores
  - Each with 4-way hyper threading
  - Each with 2 vector units
- Grouped into pairs to give 36 compute nodes
- Peak 6 SP TFLOPS
- 16 GB on package RAM
- Access to up to 384 GB off-package RAM

Knight’s Landing Xeon Phi Cache Coherence



- Nodes organized as 2-D mesh
  - Some for computation
  - Some for memory interfaces
- Use X/Y routing to send messages
- Must use more traditional directory-based scheme

Summary: directory-based coherence

- Primary observation: broadcast doesn’t scale, but luckily 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 as necessary
- One challenge: reducing overhead of directory storage
  - Use hierarchies of processors or larger line sizes
  - Limited pointer schemes: exploit fact the most processors not sharing line
  - Sparse directory schemes: exploit fact that most lines are not in cache
- Another challenge: reducing the number of messages sent (traffic) and critical path (latency) of message chains needed to implement coherence operations
- Ring-based schemes can be much simpler than point-to-point communication