Cache coherence part 4

Directory Coherence Protocols

- **Directories**:
  - Purpose to keep track of each cache block in the private caches at a centralized location
  - Typically, extend LLC (or memory) to track caching information

Directory Coherence Protocols

- **Directories**:
  - Purpose to keep track of each cache block in the private caches at a centralized location
  - Extend LLC (or memory) to track caching information
  - For each physical cache block, track:
    - **Owner**: which processor has a dirty copy (i.e., M state)
    - **Sharers**: which processors have clean copies (i.e., S state)

State	Owner (if M)	Sharer bit vector (if S)

Directory Coherence Protocols

- **Directories**:
  - Purpose to keep track of each cache block in the private caches at a centralized location
  - Extend LLC (or memory) to track caching information
  - For each physical cache block, track:
    - **Owner**: which processor has a dirty copy (i.e., M state)
    - **Sharers**: which processors have clean copies (i.e., S state)

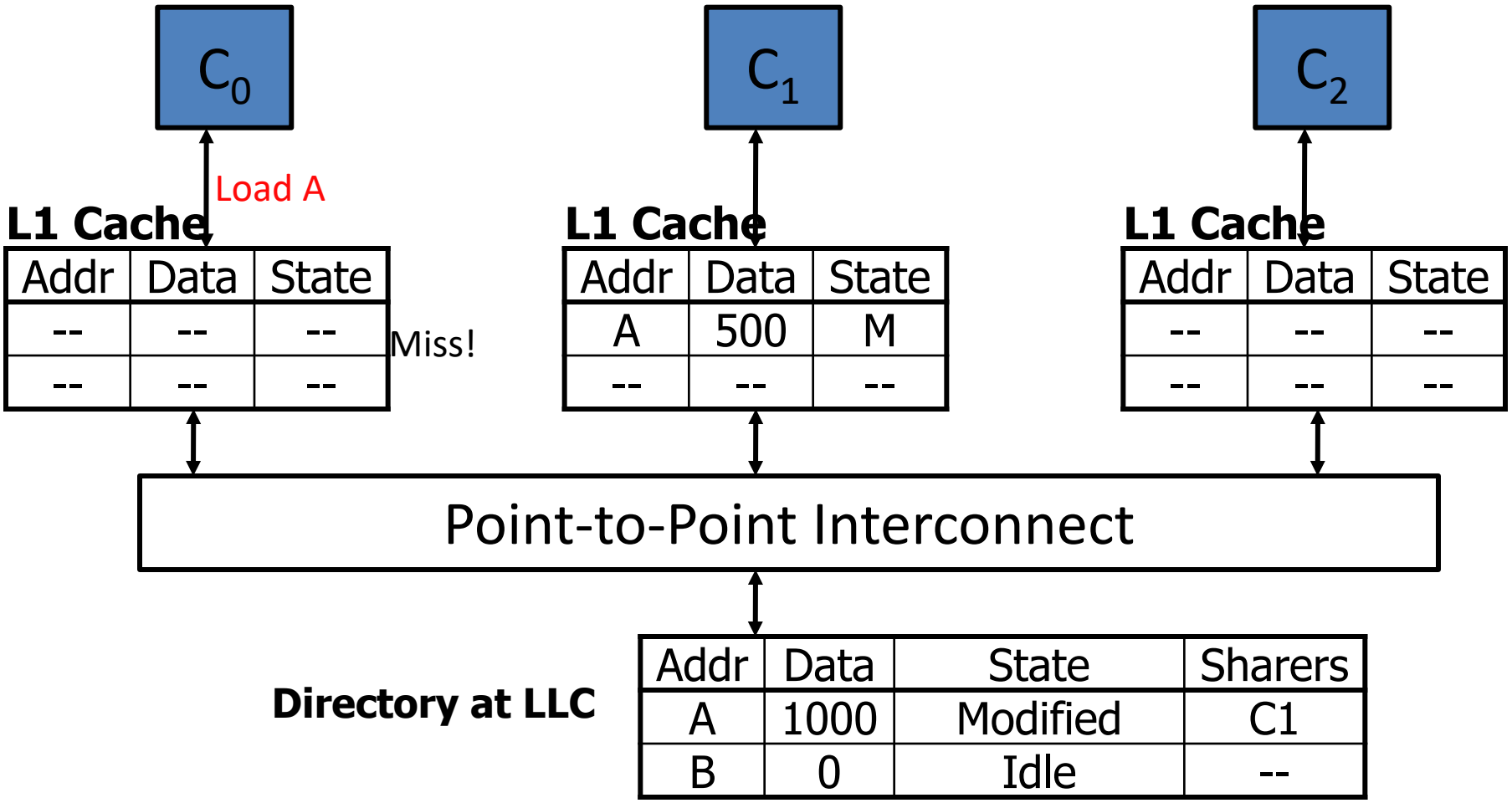
State	Owner (if M)	Sharer bit vector (if S)

  - Processor sends coherence requests to directory
    - Directory sends (point-to-point) messages only to processors **as needed**
  - Avoids non-scalable broadcast used by snooping protocols, does not require strong ordering properties from the network
    - Directory is the ordering point
  - For multicore with shared L3 cache, put directory info in cache tags

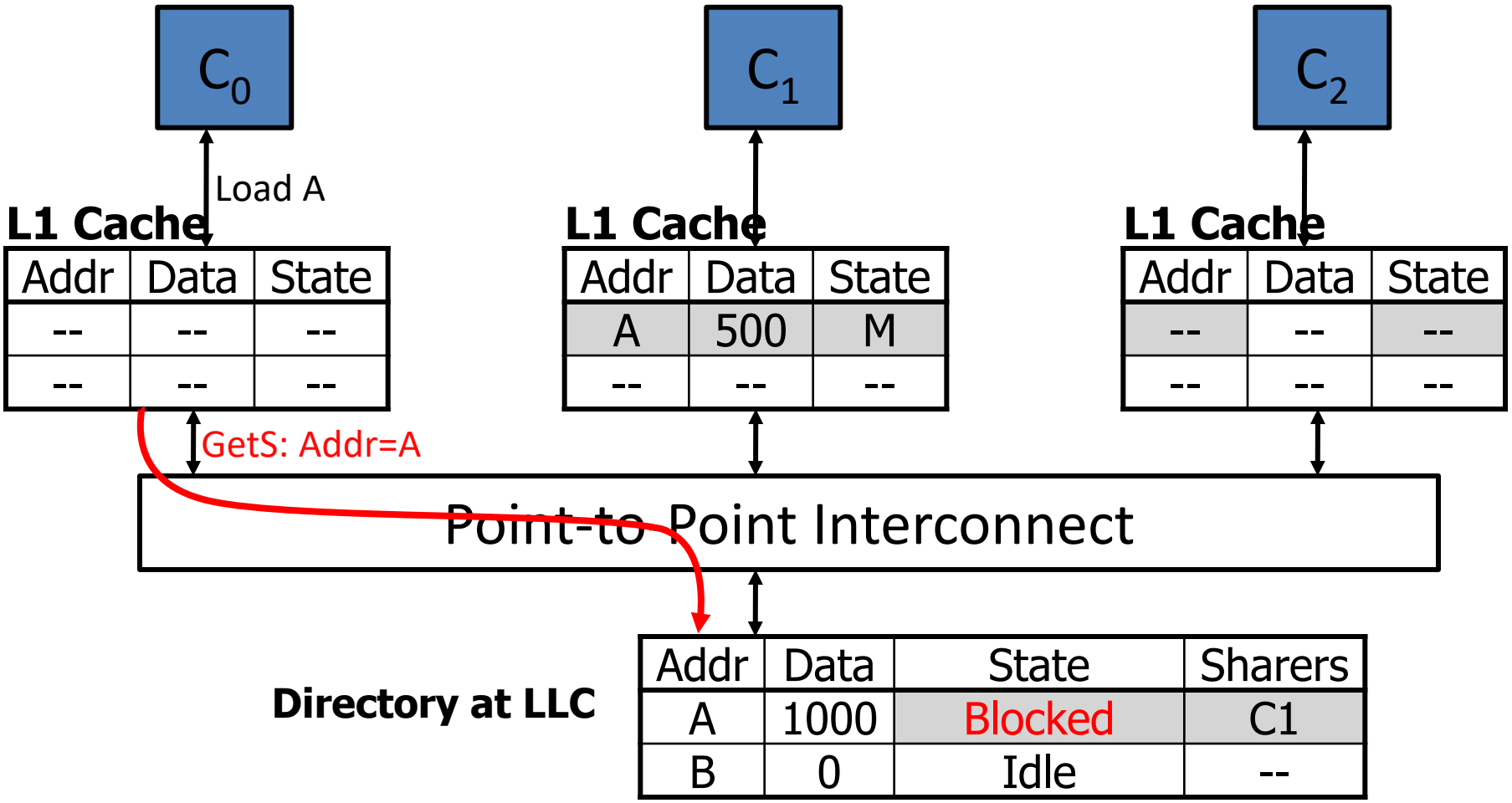
MSI Directory Protocol

- Two slightly different set of protocol (FSM)
  - All private caches follows one protocol
  - Directory has another protocol
- Similar to bus-based MSI (Modified, Shared, Invalid)
  - Same three primary states
  - New blocking states since unlike bus the network is un-ordered and can each message can take different time to travel

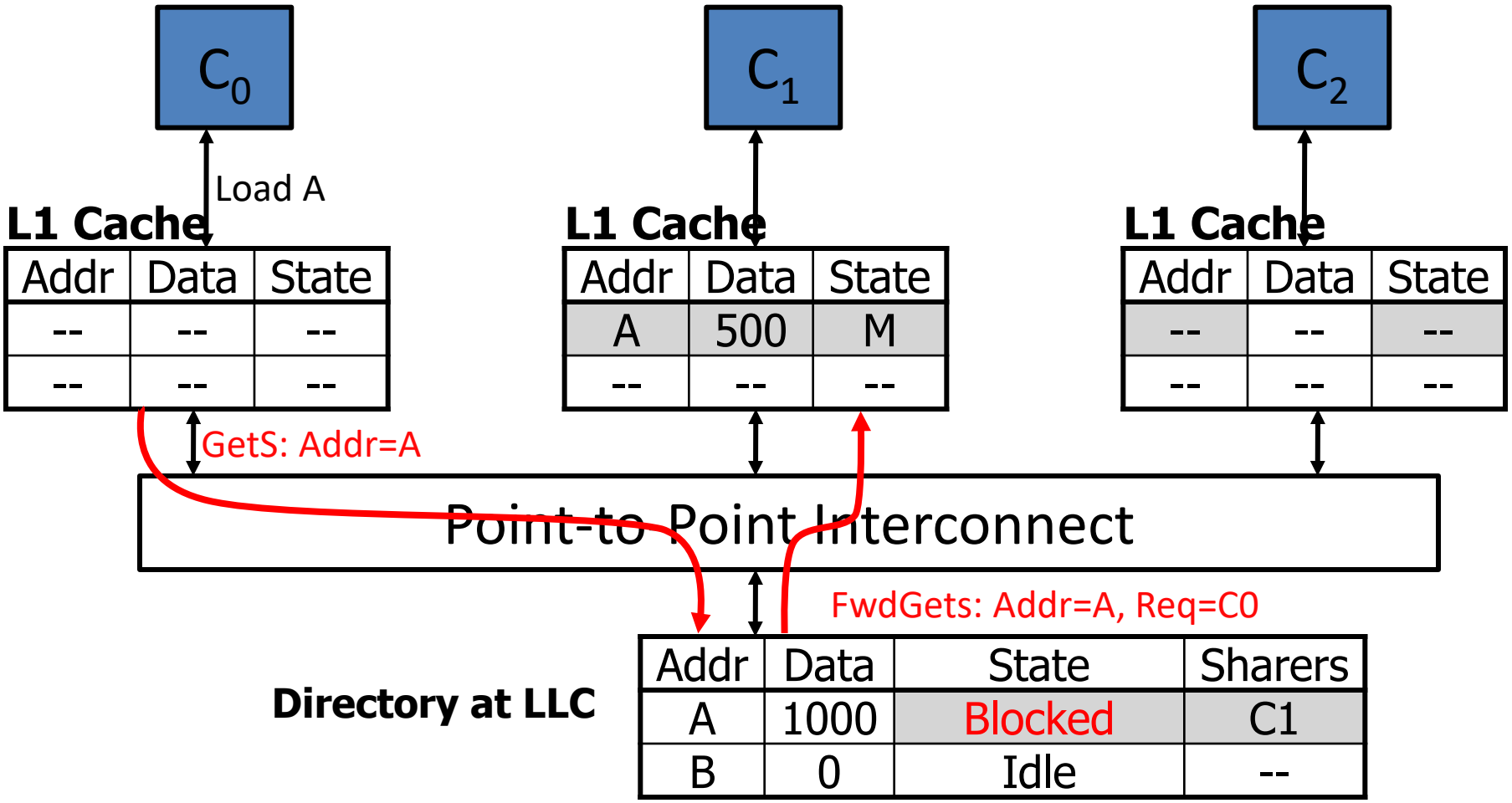
MSI Directory Example: Step #1



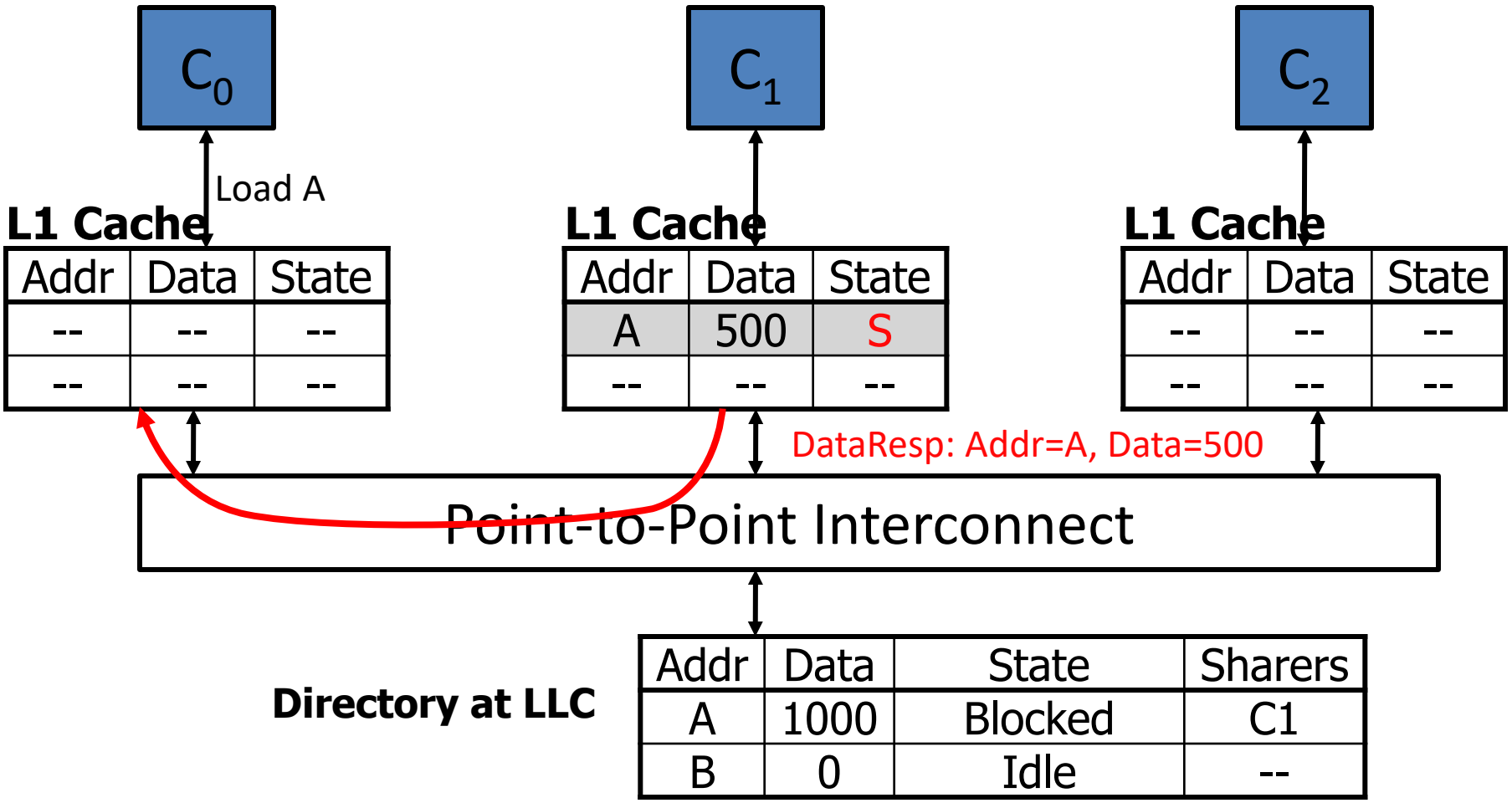
MSI Directory Example: Step #2



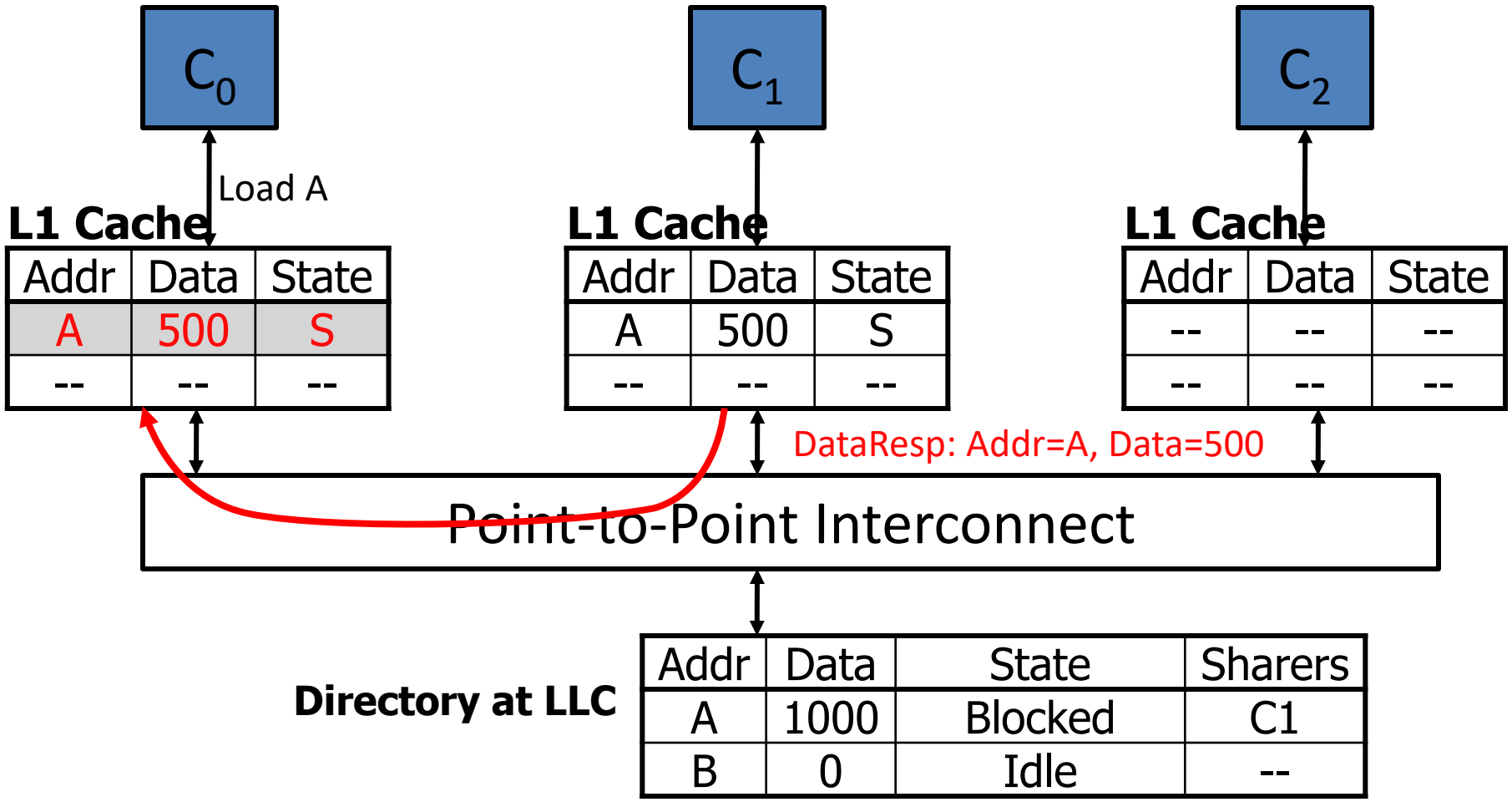
MSI Directory Example: Step #2



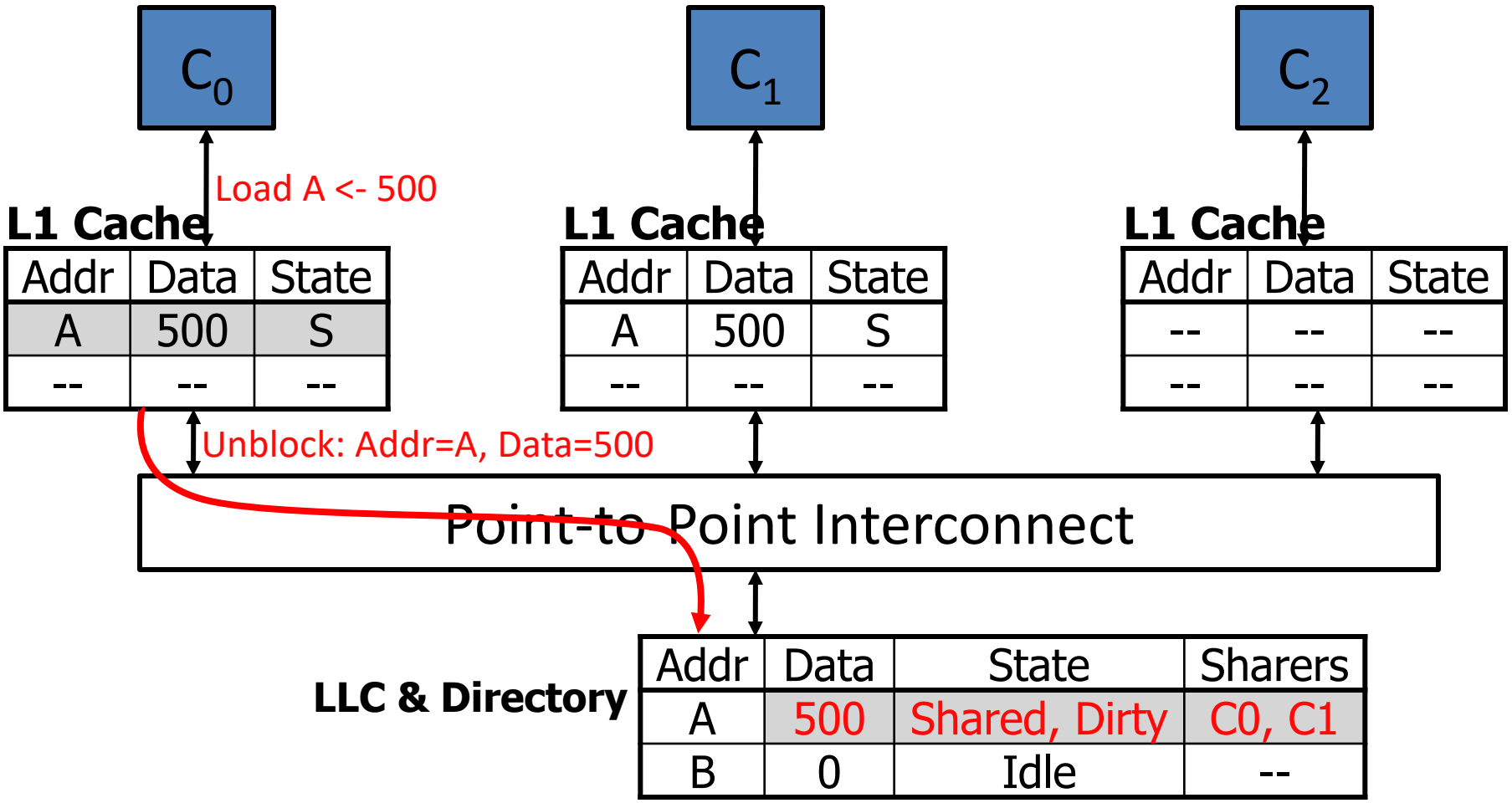
MSI Directory Example: Step #3



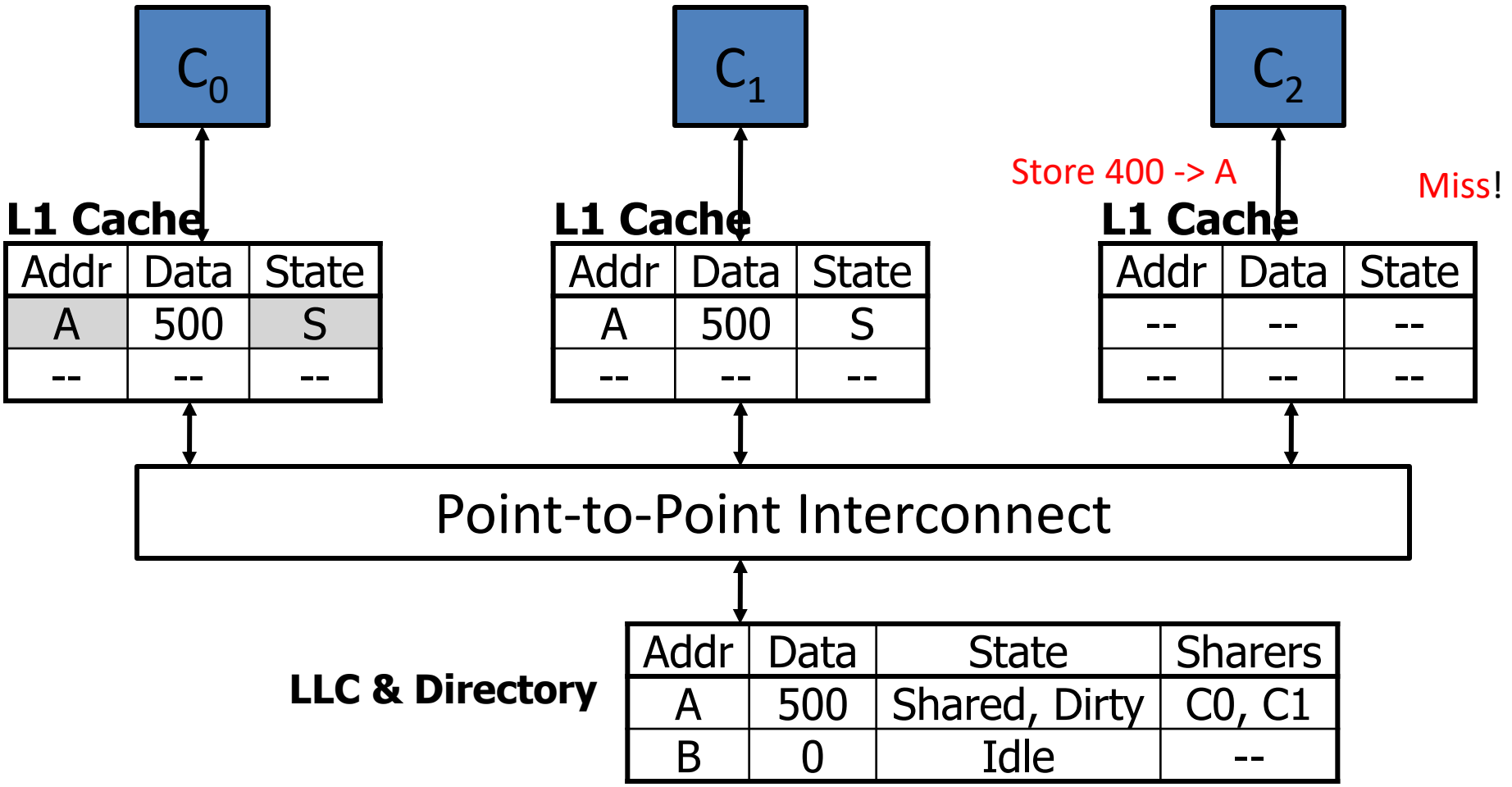
MSI Directory Example: Step #4



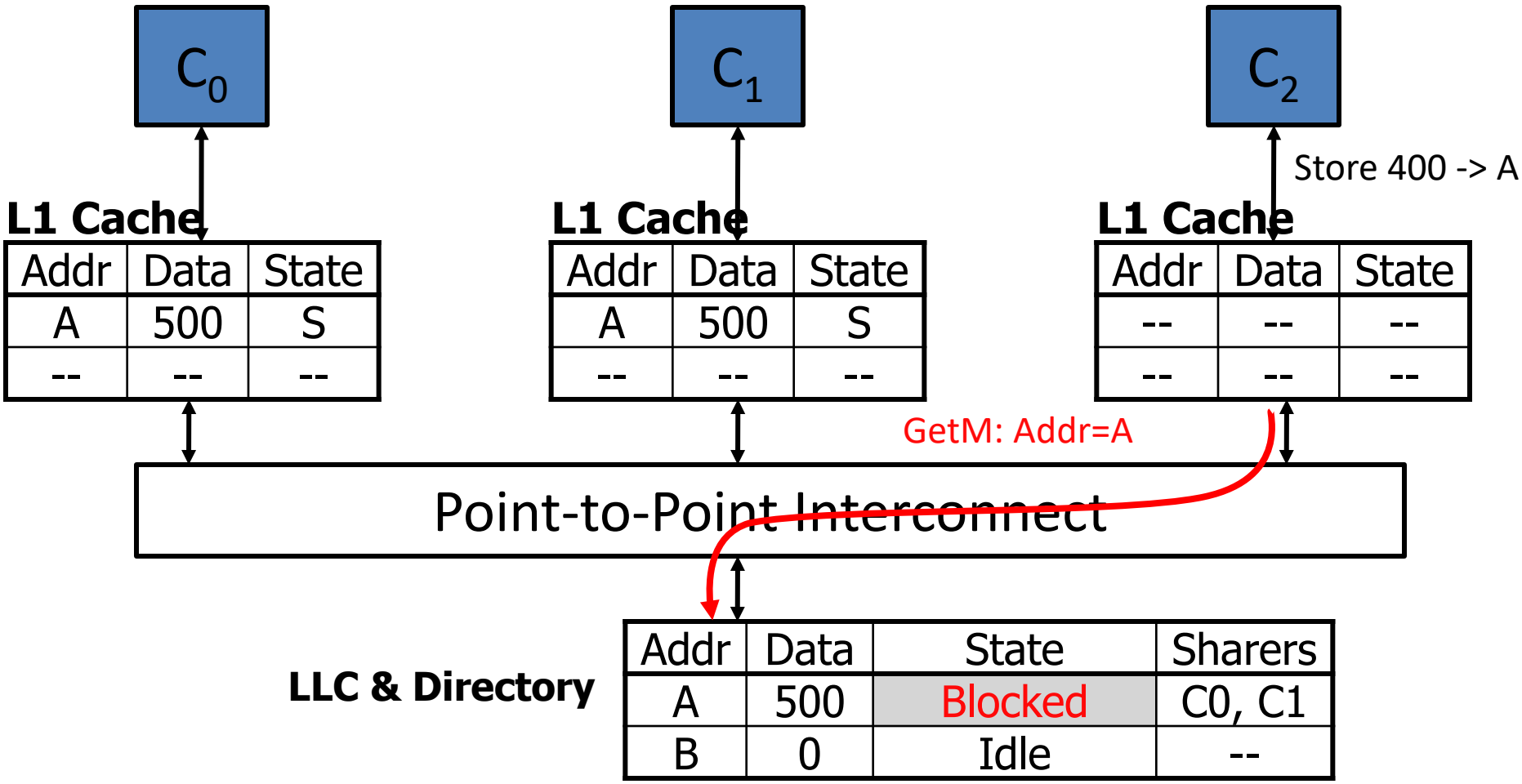
MSI Directory Example: Step #5



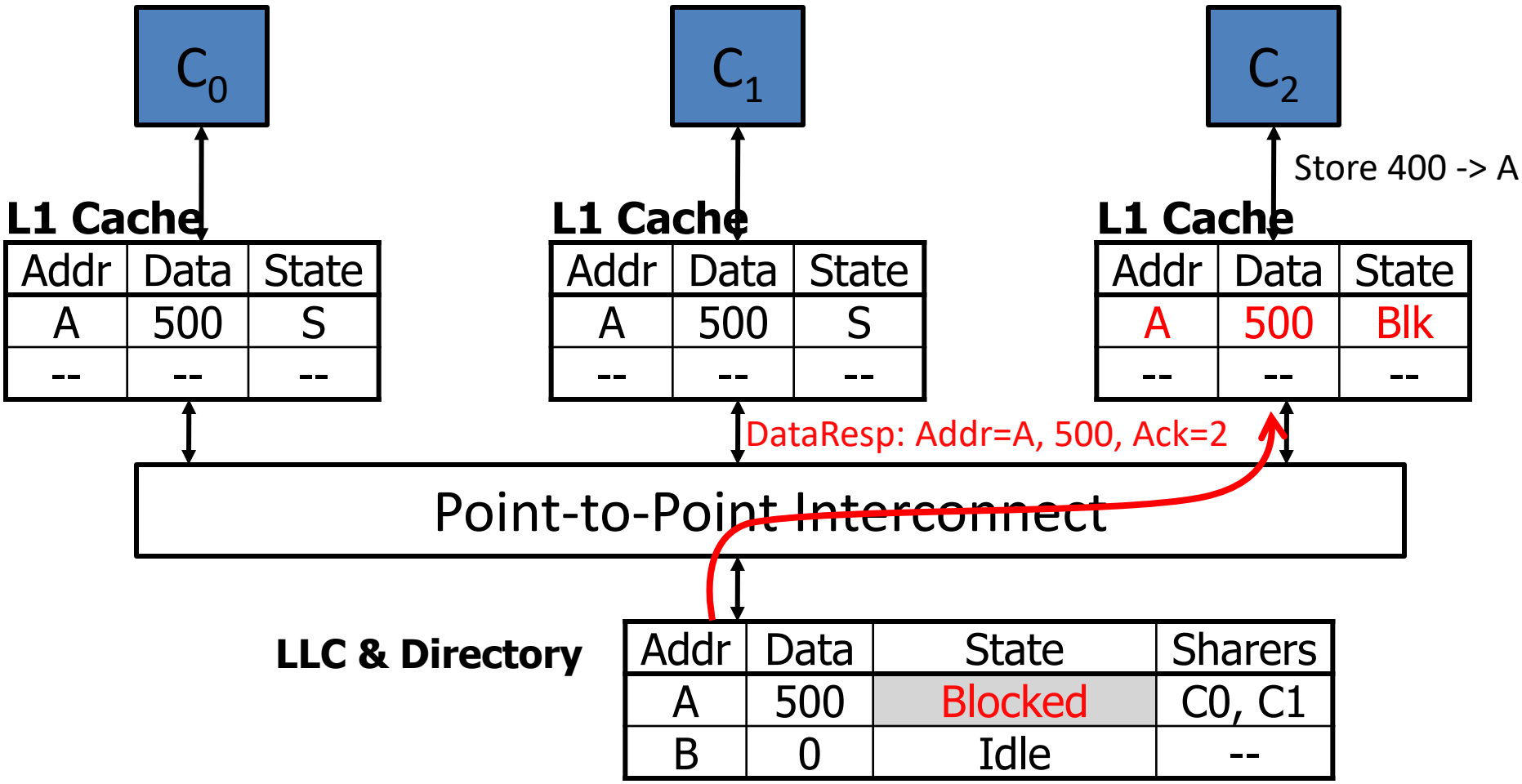
MSI Directory Example: Step #6



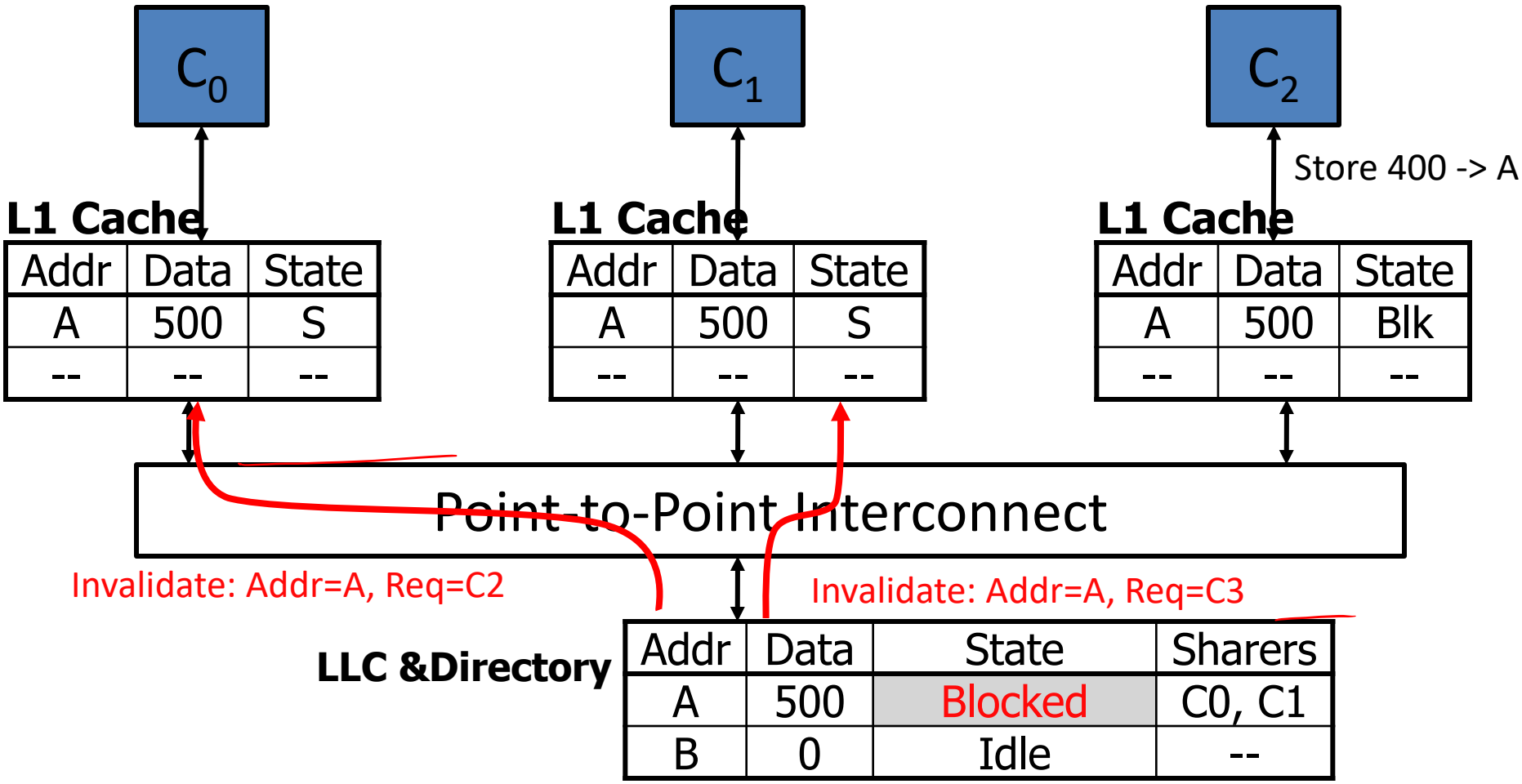
MSI Directory Example: Step #7



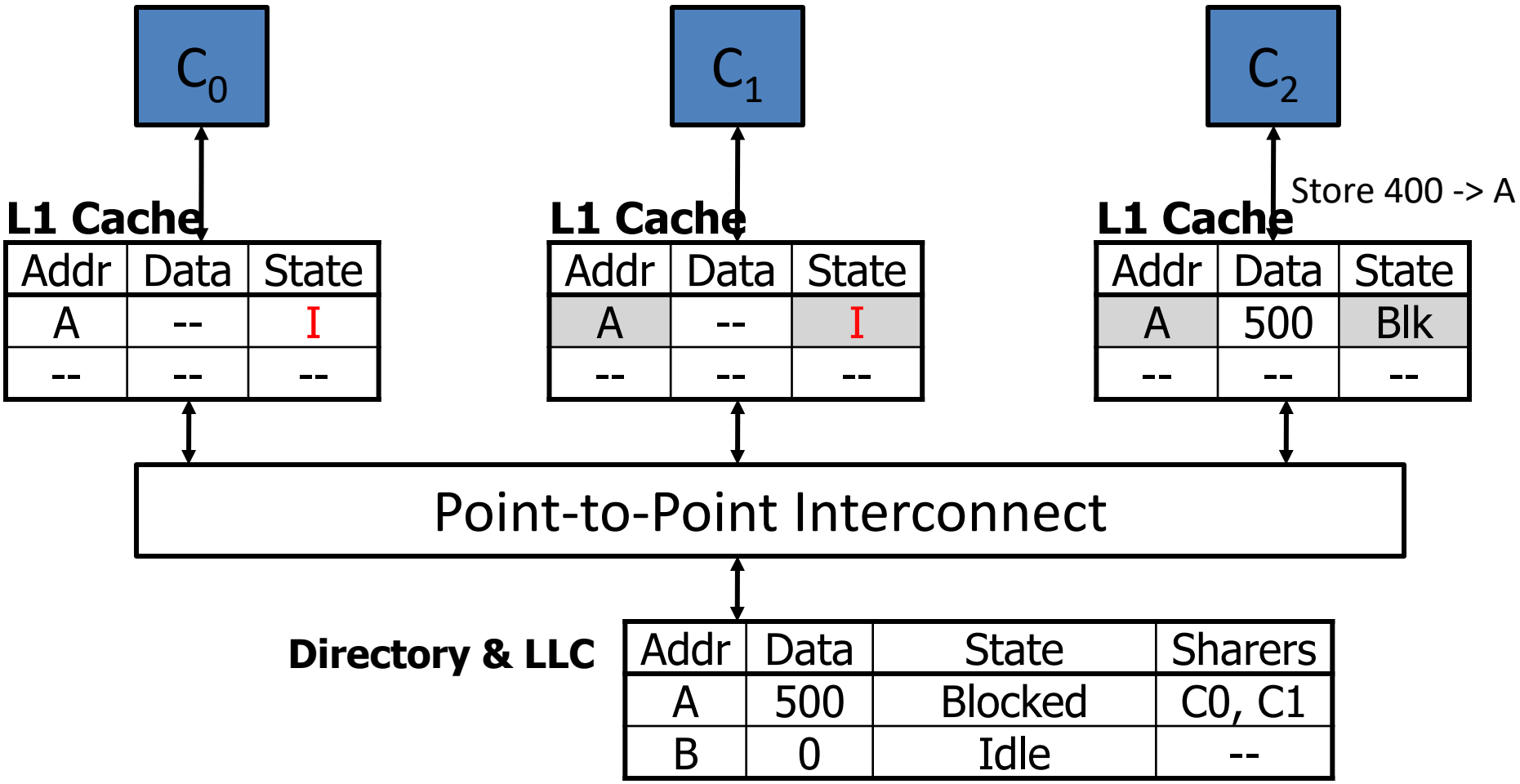
MSI Directory Example: Step #8



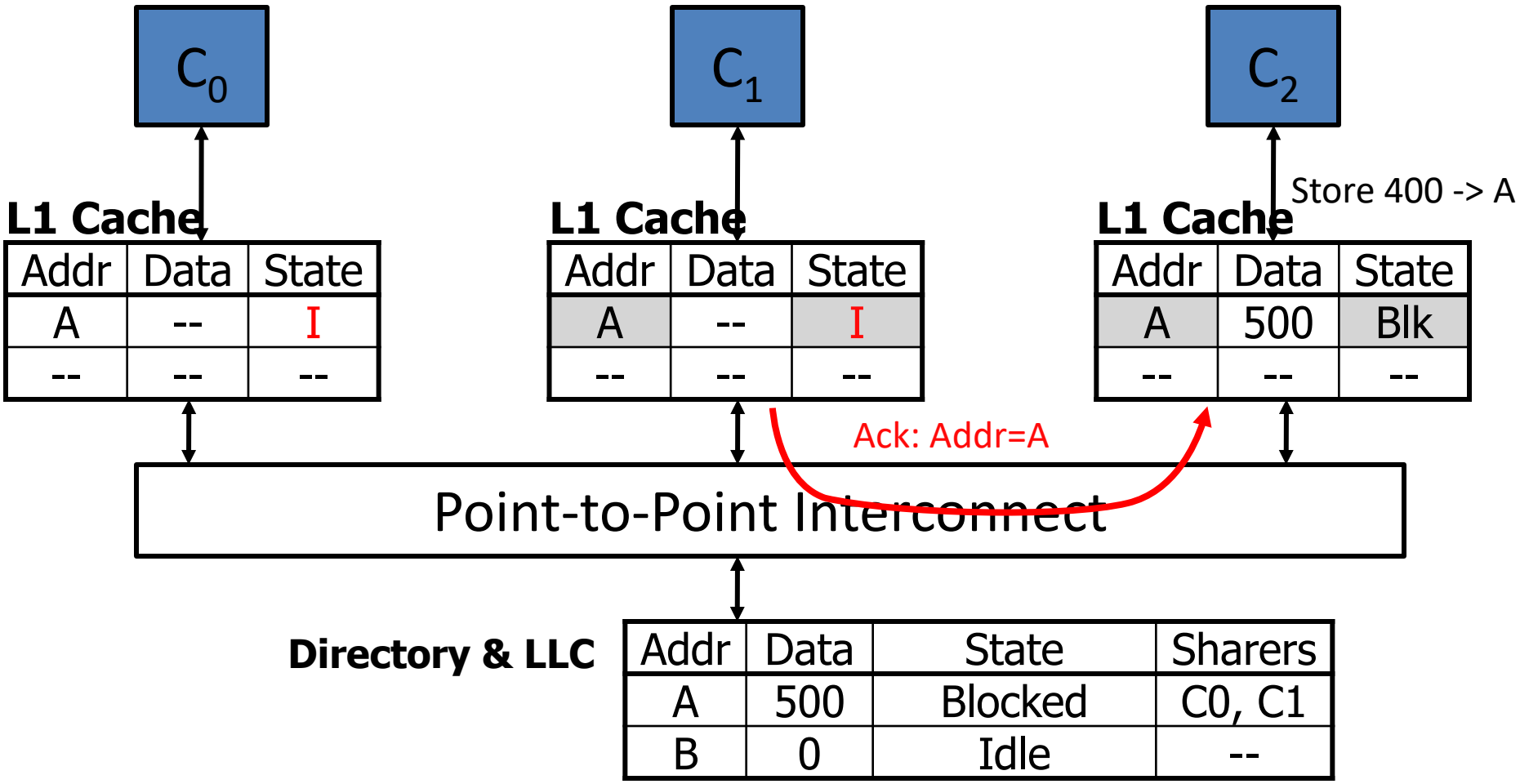
MSI Directory Example: Step #9



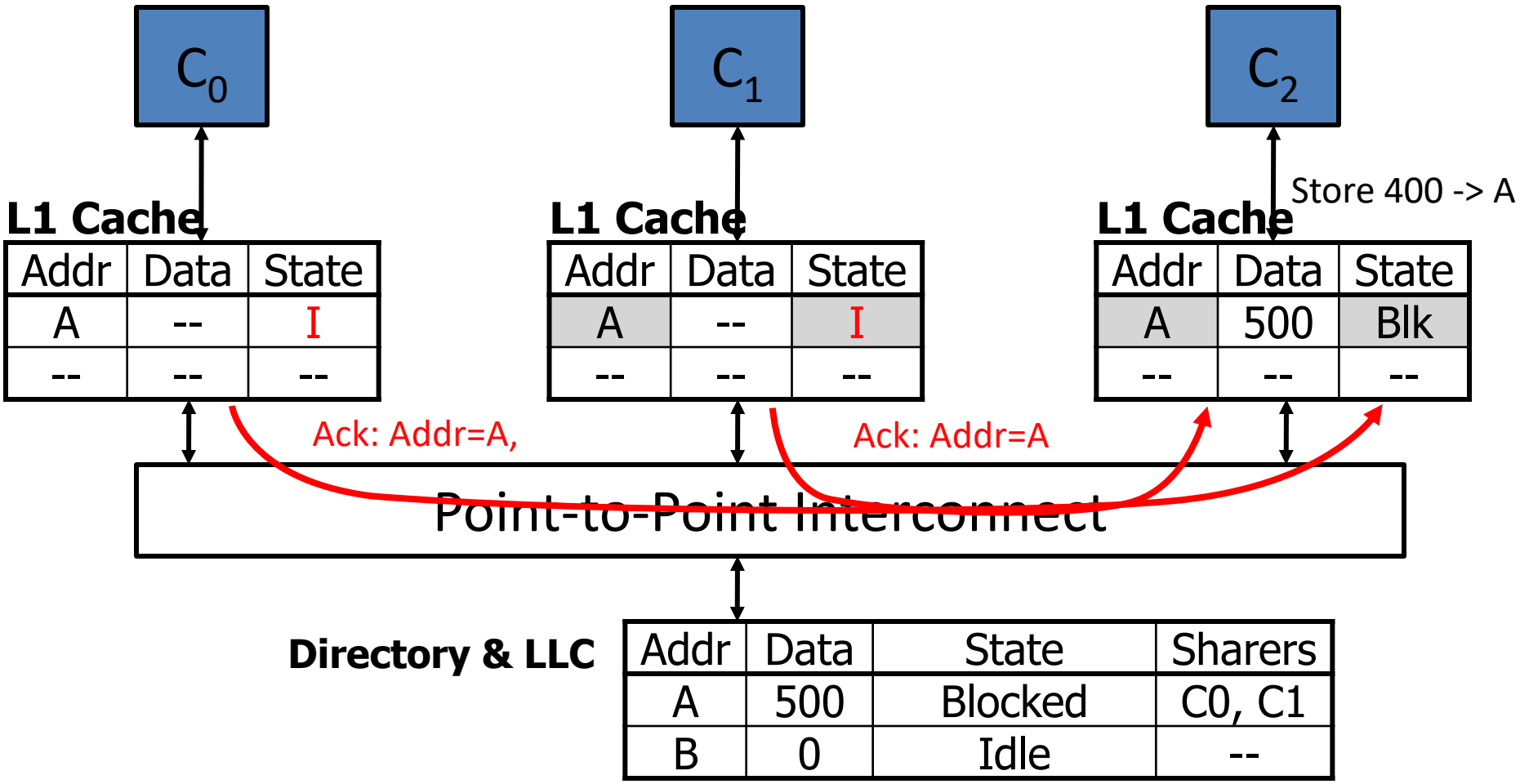
MSI Directory Example: Step #10



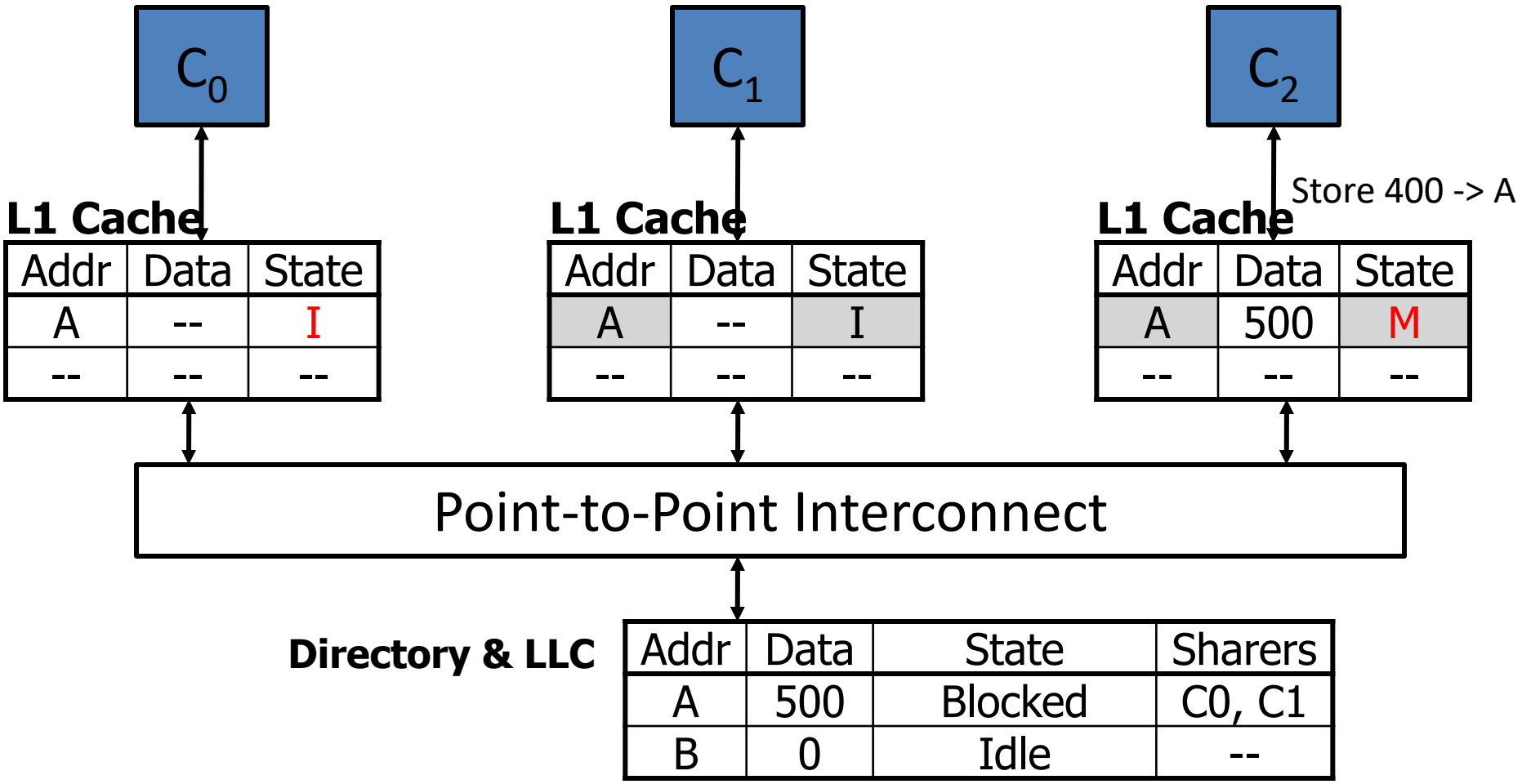
MSI Directory Example: Step #10



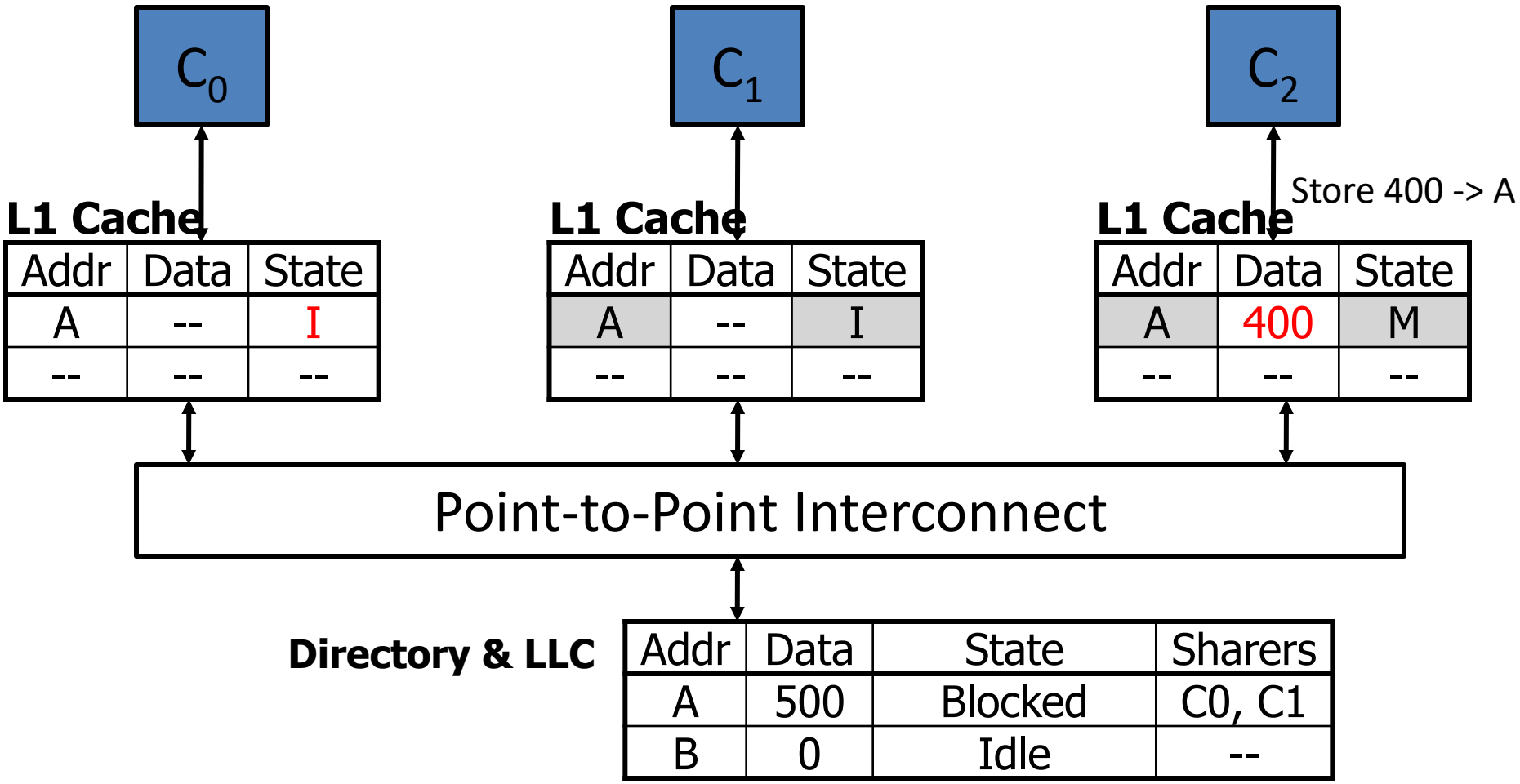
MSI Directory Example: Step #10



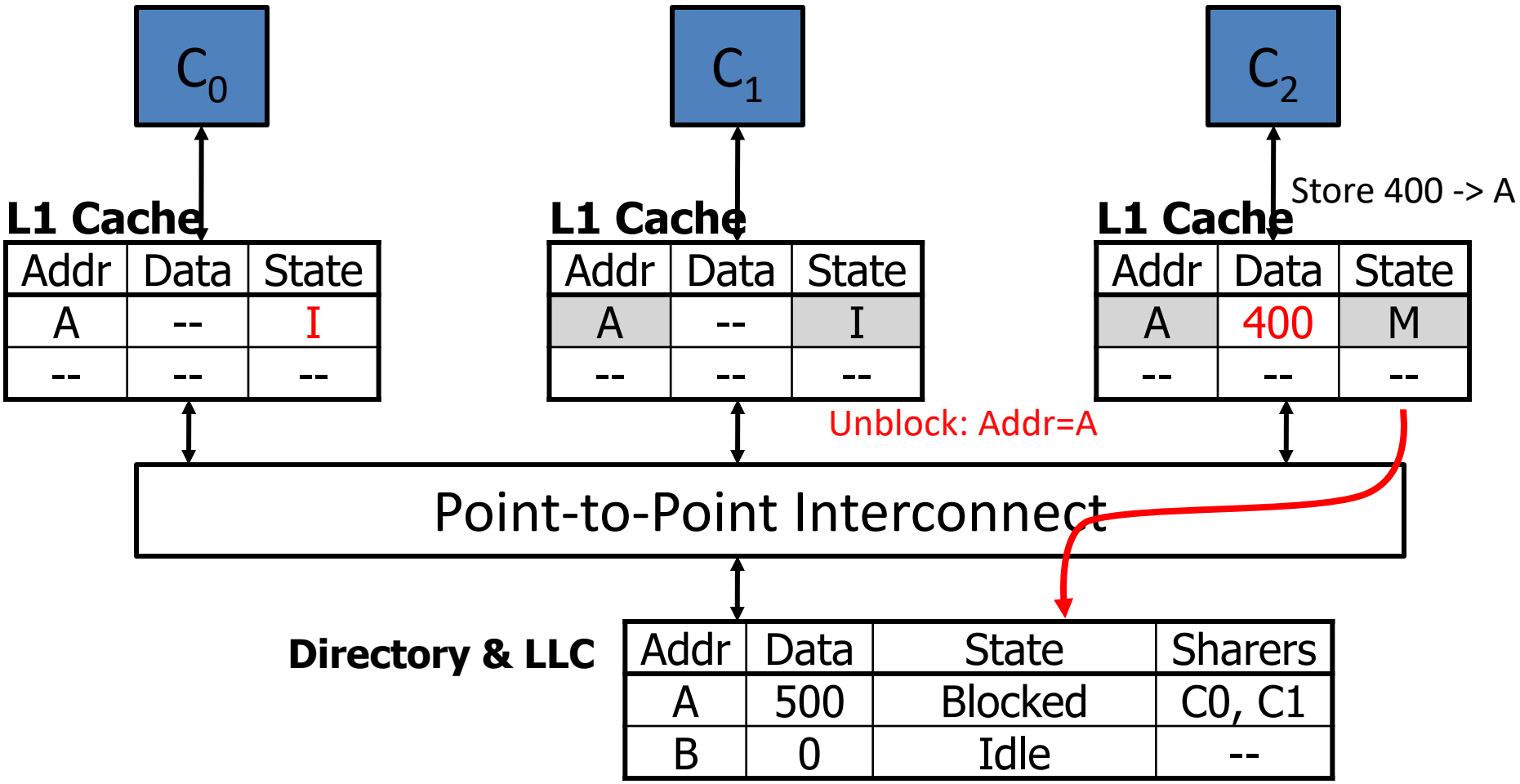
MSI Directory Example: Step #11



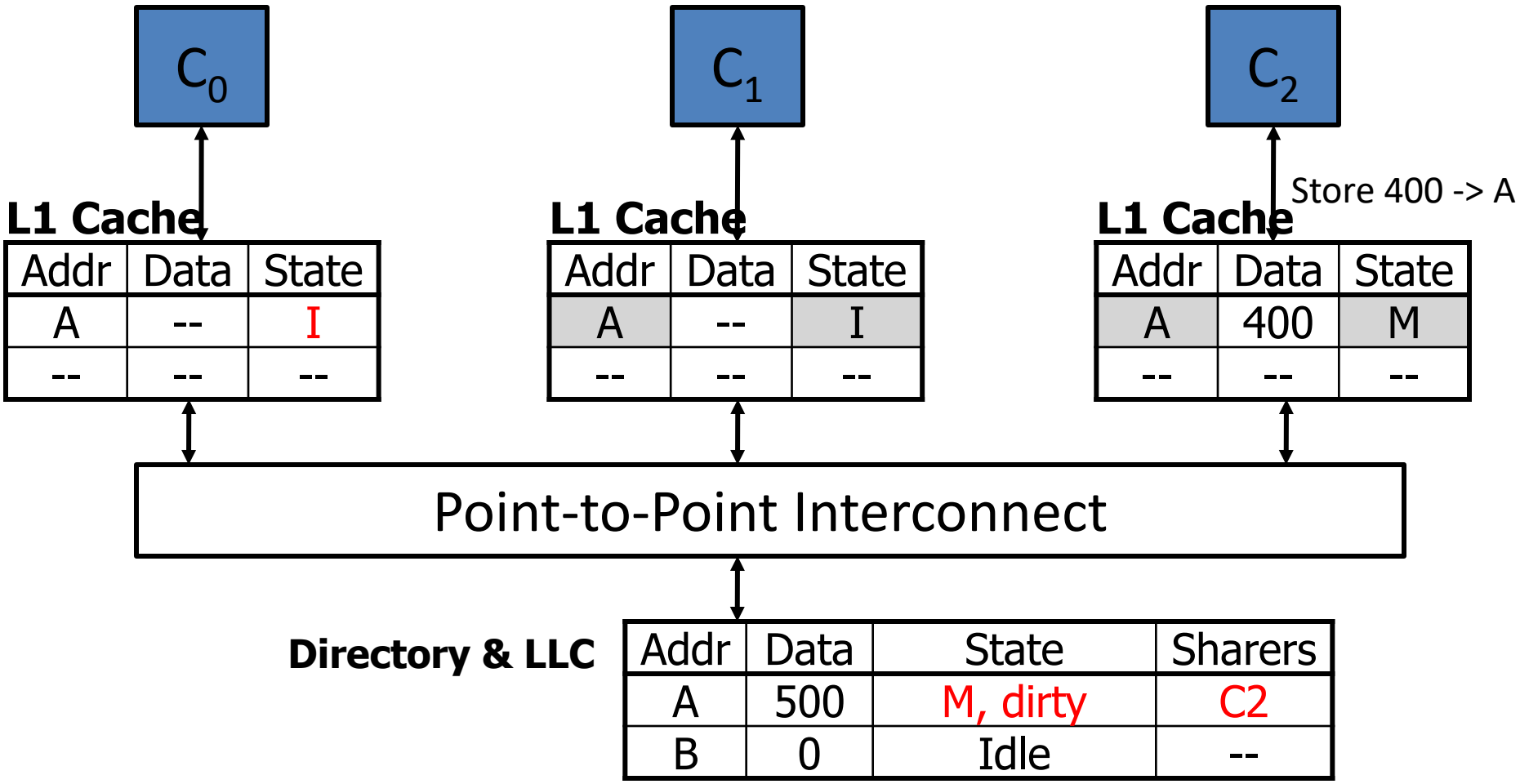
MSI Directory Example: Step #12



MSI Directory Example: Step #12

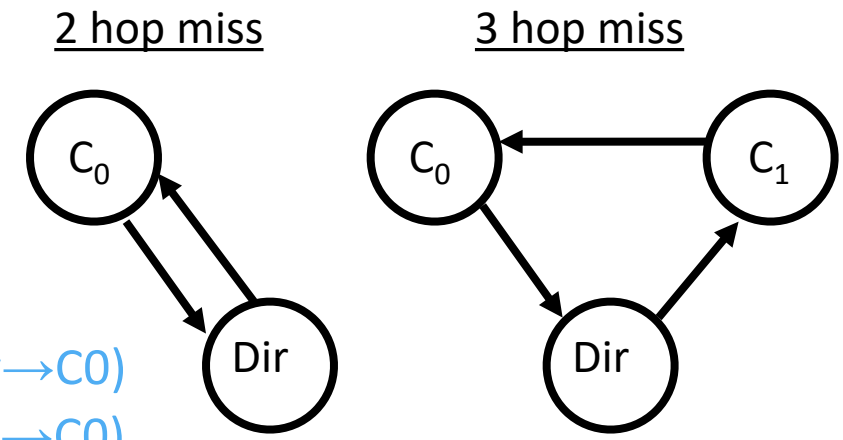


MSI Directory Example: Step #13



Directory Flip Side: Latency

- Directory protocols
  - + Lower bandwidth consumption → more scalable
  - ▸ Longer latencies



- Two read miss situations
- Unshared: get data from memory
  - ▸ Snooping: 2 hops (C0→LLC/memory→C0)
  - ▸ Directory: 2 hops (C0→LLC/memory→C0)
- Shared or exclusive: get data from other processor (P1)
  - ▸ Assume cache-to-cache transfer optimization
  - ▸ Snooping: 2 hops (C0→C1→C0)
  - ▸ Directory: **3 hops** (C0→Dir/LLC→C1→C0)
  - ▸ Common, with many processors high probability someone has it

Directory Flip Side: Complexity

- Latency not the only issue for directories
  - Subtle correctness issues as well
  - Stem from unordered nature of underlying inter-connect
- Individual requests to single cache must be ordered
  - Point-to-point network: requests may arrive in different orders
    - Directory has to enforce ordering explicitly
    - Cannot initiate actions on request B…
    - Until all relevant processors have completed actions on request A
    - Requires directory to collect acks, queue requests, etc.
- Directory protocols
  - Obvious in principle
  - Complicated in practice
  - **State space explosion due to unordered network**
  - **Need to consider various possible coherence message races**

More realistic MSI directory state table (Cache side)

	load	store	replacement	Fwd-GetS	Fwd-GetM	Inv	Put-Ack	Data from Dir (ack=0)	Data from Dir (ack>0)	Data from Owner	Inv-Ack	Last-Inv-Ack
I	send GetS to Dir/IS^D	send GetM to Dir/IM^{AD}										
IS^D	stall	stall	stall			stall		-/S		-/S		
IM^{AD}	stall	stall	stall	stall	stall			-/M	-/IM^A	-/M	ack--	
IM^A	stall	stall	stall	stall	stall						ack--	-/M
S	hit	send GetM to Dir/SM^{AD}	send PutS to Dir/SI^A			send Inv-Ack to Req/I						
SM^{AD}	hit	stall	stall	stall	stall	send Inv-Ack to Req/IM^{AD}		-/M	-/SM^A	-/M	ack--	
SM^A	hit	stall	stall	stall	stall						ack--	-/M
M	hit	hit	send PutM+data to Dir/MI^A	send data to Req and Dir/S	send data to Req/I							
MI^A	stall	stall	stall	send data to Req and Dir/SI^A	send data to Req/II^A		-/I					
SI^A	stall	stall	stall			send Inv-Ack to Req/II^A	-/I					
II^A	stall	stall	stall				-/I					

From Computer architecture synthesis lecture series on Coherence and consistency

More realistic MSI directory state table (Cache side)

Transient states

	load	store	replacement	Fwd-GetS	Fwd-GetM	Inv	Put-Ack	Data from Dir (ack=0)	Data from Dir (ack>0)	Data from Owner	Inv-Ack	Last-Inv-Ack
I	send GetS to Dir/IS^D	send GetM to Dir/IM^{AD}										
IS^D	stall	stall	stall			stall		-/S		-/S		
IM^{AD}	stall	stall	stall	stall	stall			-/M	-/IM^A	-/M	ack--	
IM^A	stall	stall	stall	stall	stall						ack--	-/M
S	hit	send GetM to Dir/SM^{AD}	send PutS to Dir/SI^A			send Inv-Ack to Req/I						
SM^{AD}	hit	stall	stall	stall	stall	send Inv-Ack to Req/IM^{AD}		-/M	-/SM^A	-/M	ack--	
SM^A	hit	stall	stall	stall	stall						ack--	-/M
M	hit	hit	send PutM+data to Dir/MI^A	send data to Req and Dir/S	send data to Req/I							
MI^A	stall	stall	stall	send data to Req and Dir/SI^A	send data to Req/II^A		-/I					
SI^A	stall	stall	stall			send Inv-Ack to Req/II^A	-/I					
II^A	stall	stall	stall				-/I					

More realistic MSI directory state table (Cache side)

	load	store	replacement	Fwd-GetS	Fwd-GetM	Inv	Put-Ack	Data from Dir (ack=0)	Data from Dir (ack>0)	Data from Owner	Inv-Ack	Last-Inv-Ack
I	send GetS to Dir/IS^D	send GetM to Dir/IM^{AD}										
IS^D	stall	stall	stall			stall		-/S		-/S		
IM^{AD}	stall	stall	stall	stall	stall			-/M	-/IM^A	-/M	ack--	
IM^A	stall	stall	stall	stall	stall						ack--	-/M
S	hit	send GetM to Dir/SM^{AD}	send PutS to Dir/SI^A			send Inv-Ack to Req/I						
SM^{AD}	hit	stall	stall	stall	stall	send Inv-Ack to Req/IM^{AD}		-/M	-/SM^A	-/M	ack--	
SM^A	hit	stall	stall	stall	stall						ack--	-/M
M	hit	hit	send PutM+data to Dir/MI^A	send data to Req and Dir/S	send data to Req/I							
MI^A	stall	stall	stall	send data to Req and Dir/SI^A	send data to Req/II^A		-/I					
SI^A	stall	stall	stall			send Inv-Ack to Req/II^A	-/I					
II^A	stall	stall	stall				-/I					

More realistic MSI directory state table (Directory side)

	GetS	GetM	PutS-NotLast	PutS-Last	PutM+data from Owner	PutM+data from NonOwner	Data
I	send data to Req, add Req to Sharers/S	send data to Req, set Owner to Req/M	send Put-Ack to Req	send Put-Ack to Req		send Put-Ack to Req	
S	send data to Req, add Req to Sharers	send data to Req, send Inv to Sharers, clear Sharers, set Owner to Req/M	remove Req from Sharers, send Put-Ack to Req	remove Req from Sharers, send Put-Ack to Req/I		remove Req from Sharers, send Put-Ack to Req	
M	Send Fwd-GetS to Owner, add Req and Owner to Sharers, clear Owner/S^D	Send Fwd-GetM to Owner, set Owner to Req	send Put-Ack to Req	send Put-Ack to Req	copy data to memory, clear Owner, send Put-Ack to Req/I	send Put-Ack to Req	
S^D	stall	stall	remove Req from Sharers, send Put-Ack to Req	remove Req from Sharers, send Put-Ack to Req		remove Req from Sharers, send Put-Ack to Req	copy data to memory/S