Computer Architecture
ELE 475 / COS 475
Slide Deck 15: Directory Cache Coherence

David Wentzlaff

Department of Electrical Engineering

Princeton University

PRINCETON UNIVERSITY

PRINCETON
School of Engineering and Applied Science

Coherency Misses

1. True sharing misses arise from the communication of data through the cache coherence mechanism
   - Invalidates due to 1st write to shared block
   - Reads by another CPU of modified block in different cache
   - Miss would still occur if block size were 1 word
2. False sharing misses when a block is invalidated because some word in the block, other than the one being read, is written into
   - Invalidation does not cause a new value to be communicated, but only causes an extra cache miss
   - Block is shared, but no word in block is actually shared $\Rightarrow$ miss would not occur if block size were 1 word

Example: True v. False Sharing v. Hit?

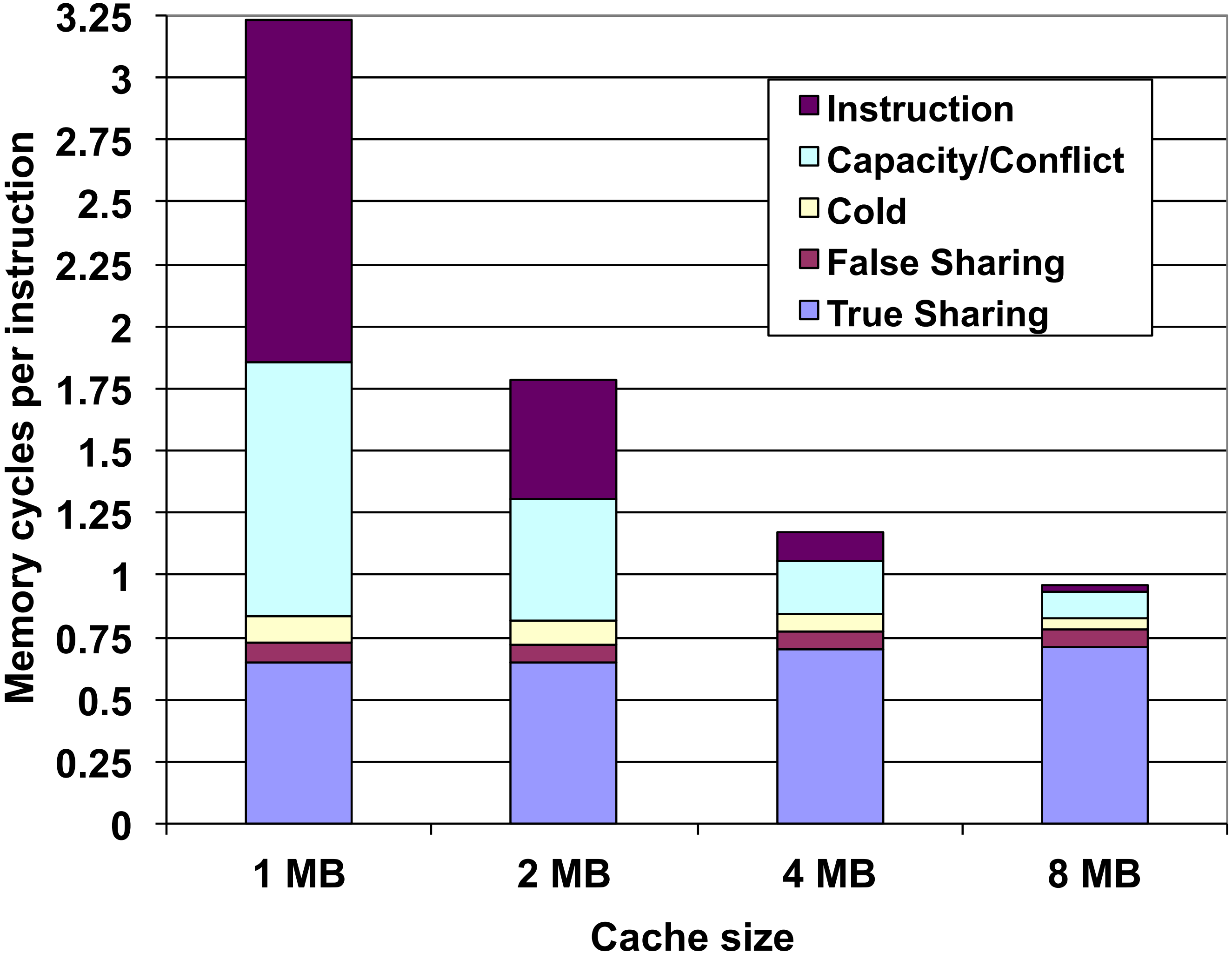
- **Assume x1 and x2 in same cache block.
P1 and P2 both read x1 and x2 before.**

Time	P1	P2	True, False, Hit? Why?
1	Write x1		True miss; invalidate x1 in P2
2		Read x2	False miss; x1 irrelevant to P2
3	Write x1		False miss; x1 irrelevant to P2
4		Write x2	False miss; x1 irrelevant to P2
5	Read x2		True miss; invalidate x2 in P1

MP Performance 4 Processor
Commercial Workload: OLTP, Decision Support (Database), Search Engine

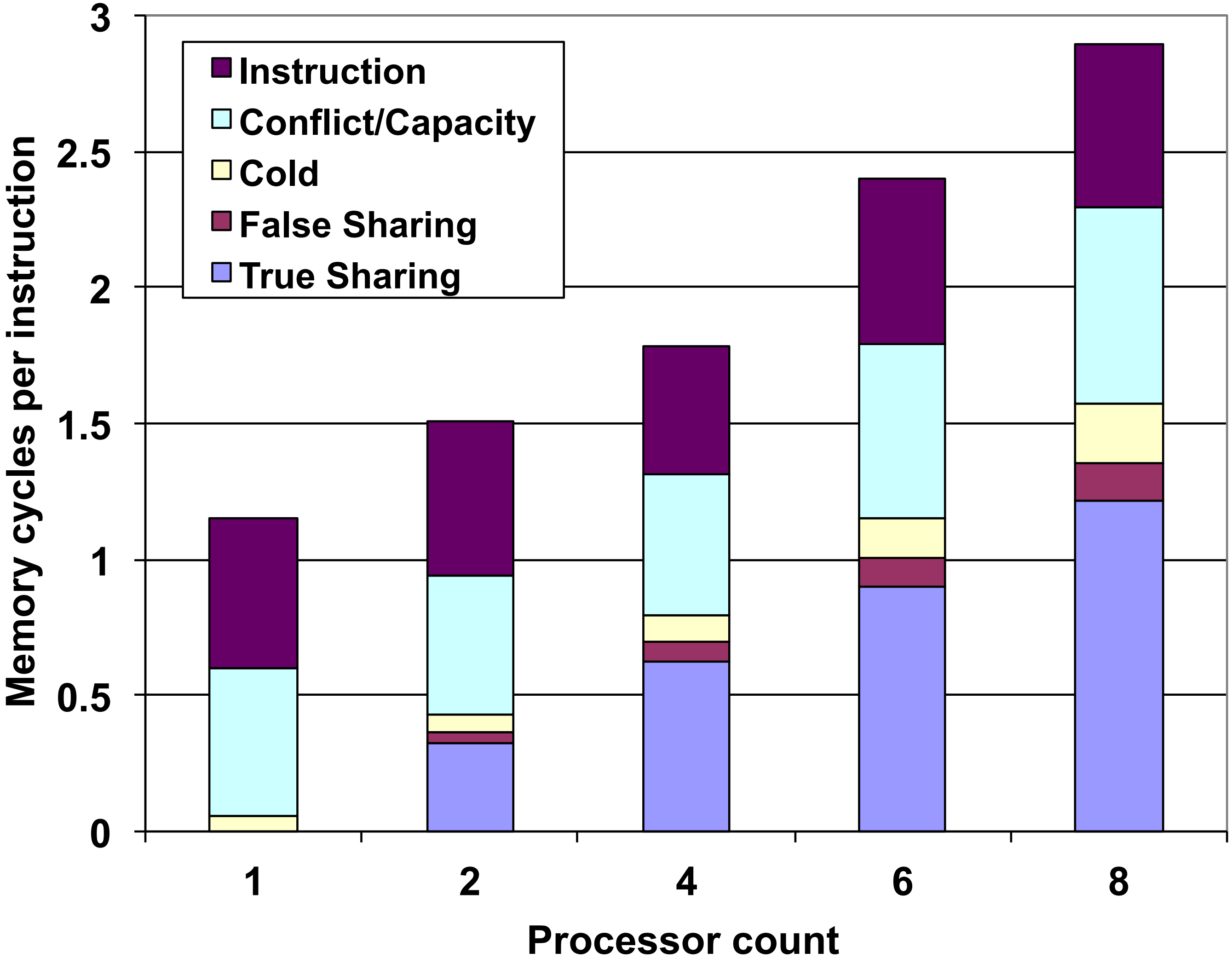
- True sharing and false sharing unchanged going from 1 MB to 8 MB (L3 cache)

- Uniprocessor cache misses improve with cache size increase (Instruction, Capacity/Conflict, Compulsory)



MP Performance 2MB Cache Commercial Workload: OLTP, Decision Support (Database), Search Engine

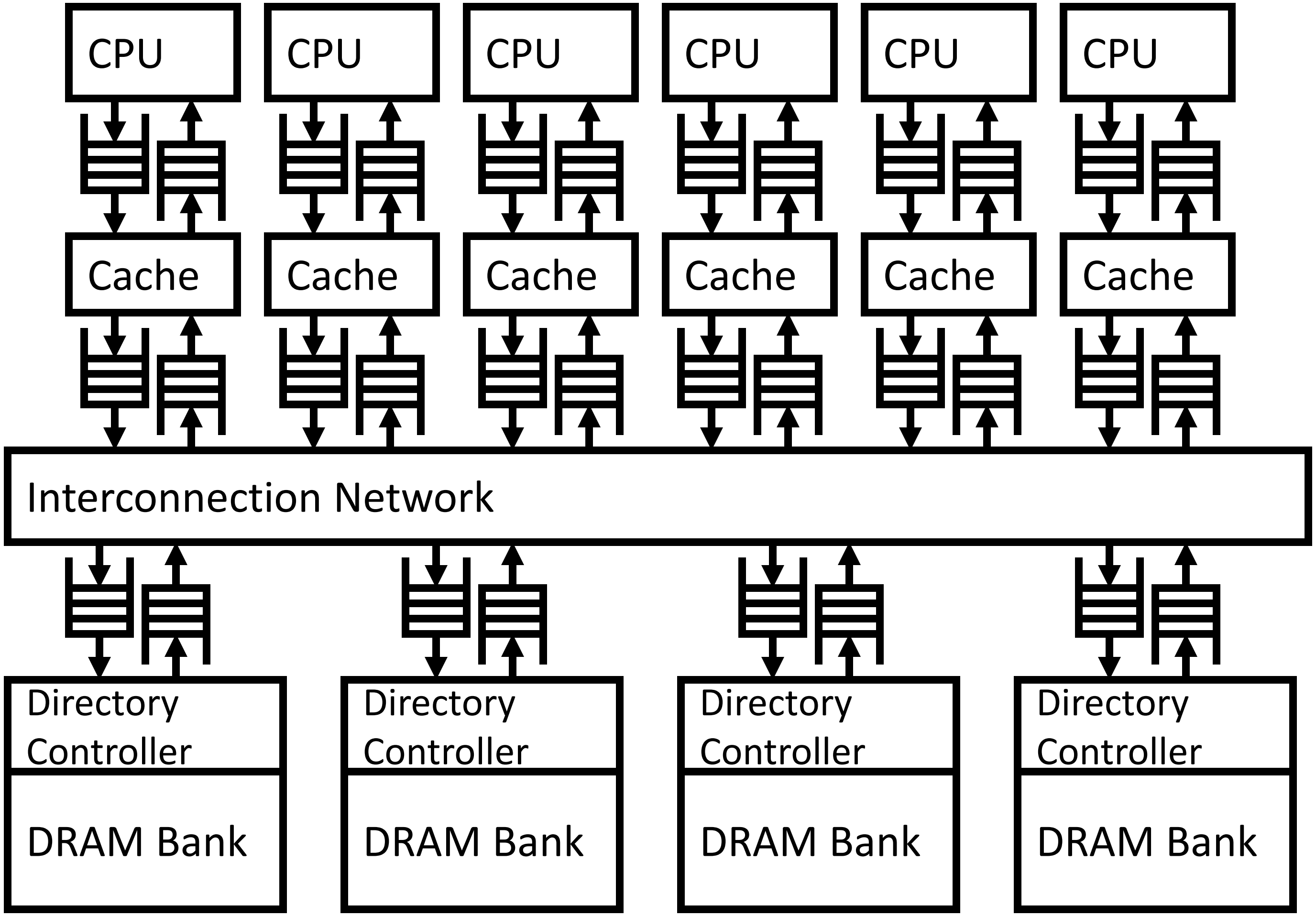
- True sharing, false sharing increase going from 1 to 8 CPUs



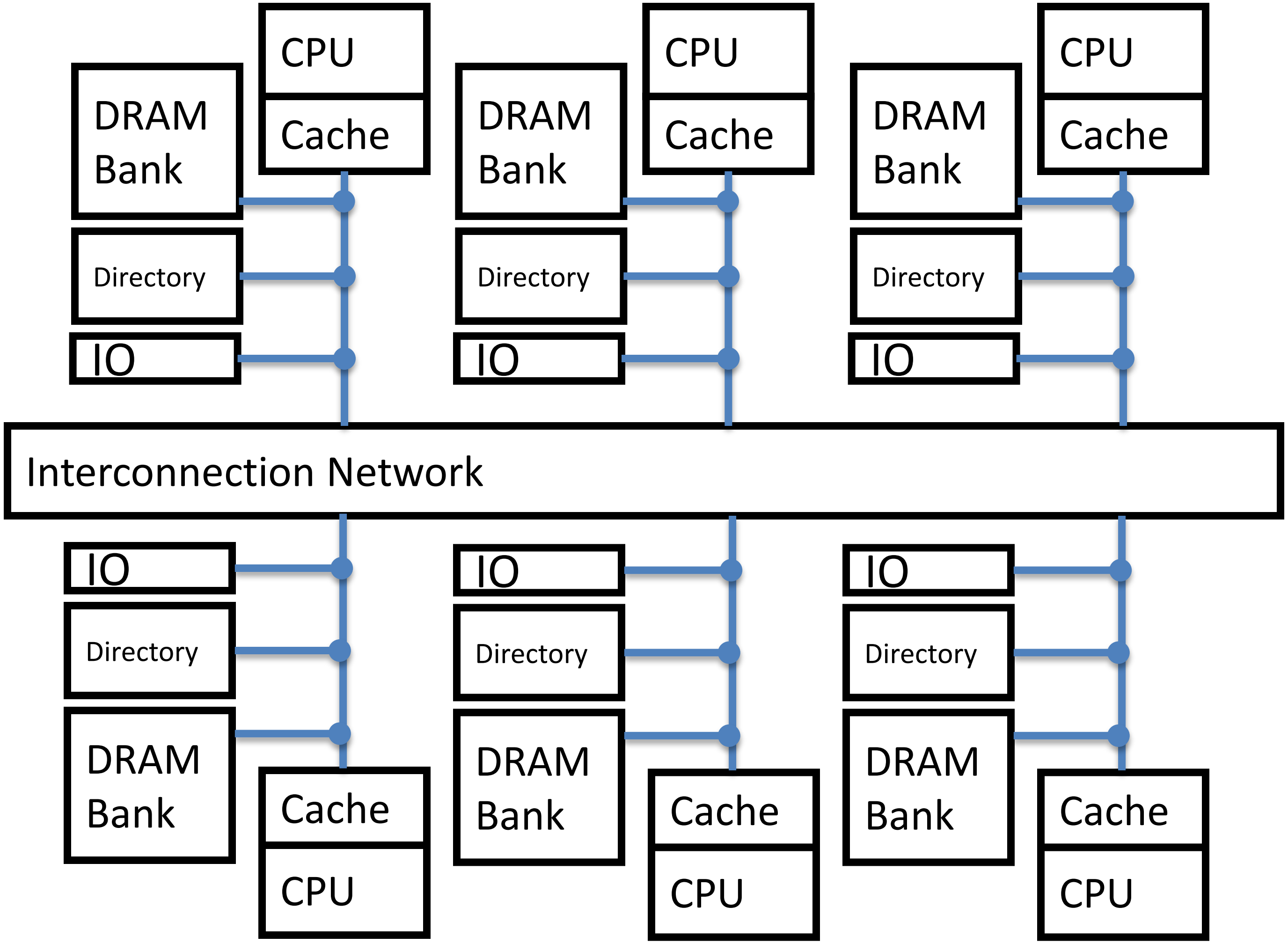
Directory Coherence Motivation

- Snoopy protocols require every cache miss to broadcast
  - Requires large bus bandwidth, O(N)
  - Requires large cache snooping bandwidth, O(N^2) aggregate
- Directory protocols enable further scaling
  - Directory can track all caches holding a memory block and use point-to-point messages to maintain coherence
  - Communication done via scalable point-to-point interconnect

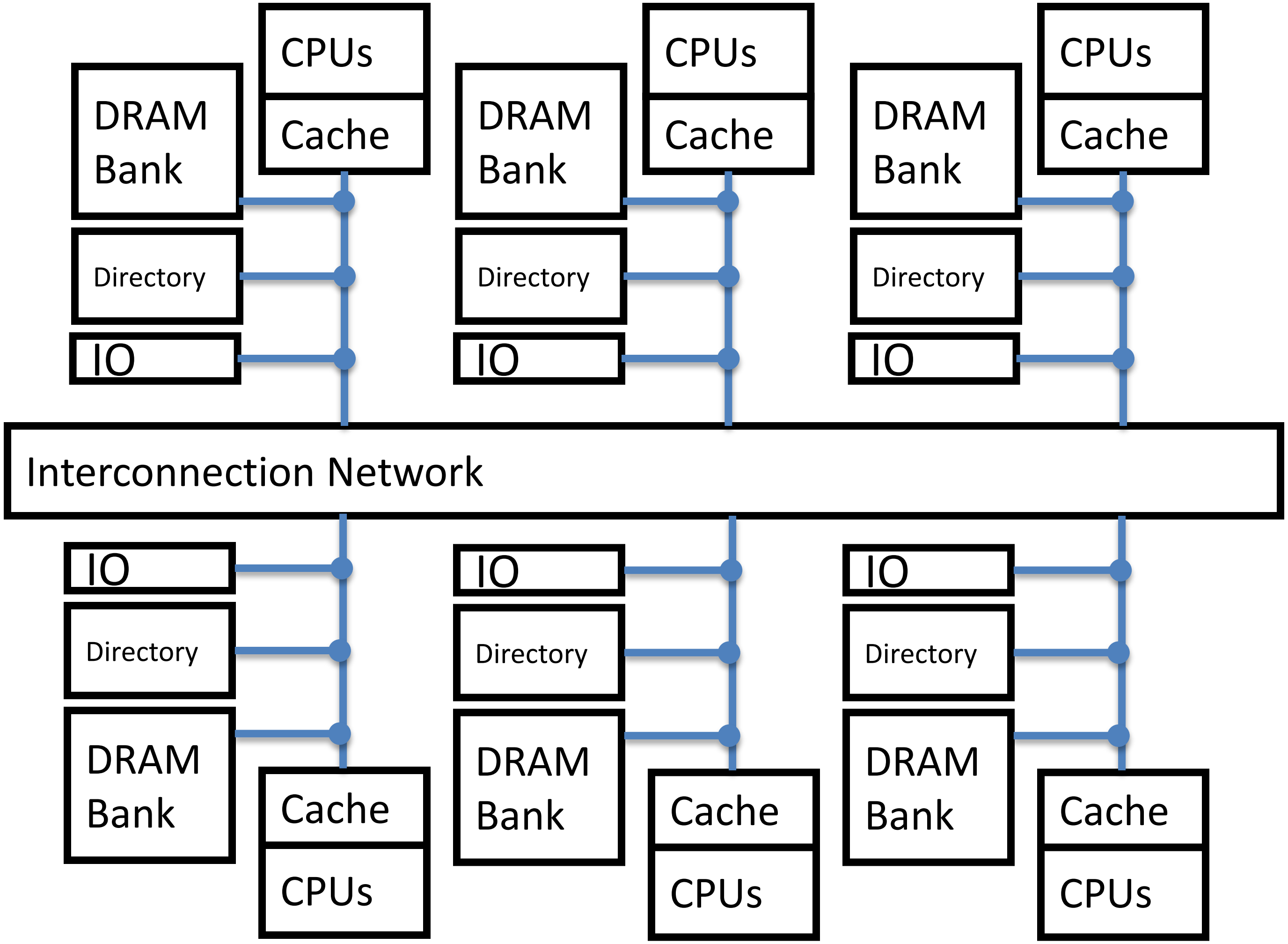
Directory Cache Coherence



Distributed Shared Memory



Multicore Chips in a Multi-chip System



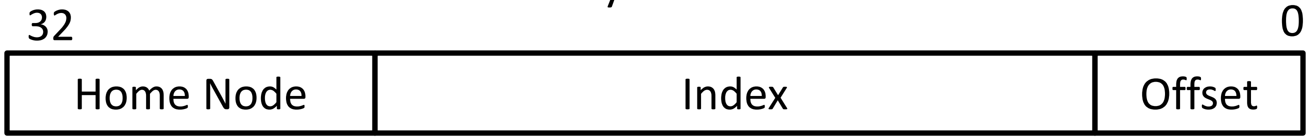
Non-Uniform Memory Access (NUMA)

- Latency to access memory is different depending on node accessing it and address being accessed
- NUMA does not necessarily imply ccNUMA

Address to Home Directory

High Order Bits Determine Home (Directory)

Physical Address



32

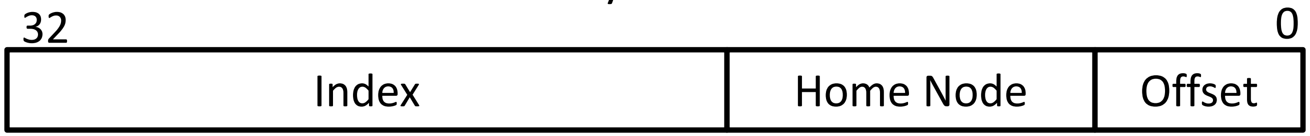
Home Node | Index | Offset

0

- OS can control placement in NUMA
- Homes can become hotspots

Low Order Bits Determine Home (Directory)

Physical Address



32

Index | Home Node | Offset

0

- OS looses control on placement
- Load balanced well

Basic Full-Map Directory

State	Sharers/Owner
S	1011001100011
U	xxxxxxxxxxxxxxxx
U	xxxxxxxxxxxxxxxx
E	0001000000000
…	

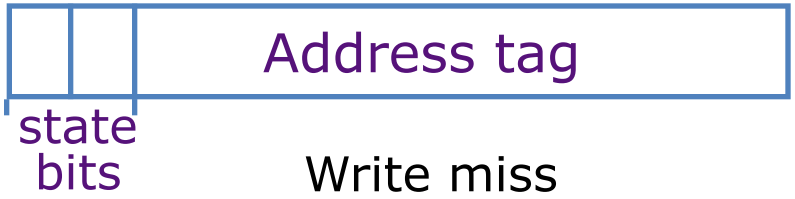
State: State that the directory believes cache line is in {Shared, Uncached, Exclusive, (Pending)}
Sharers/Owner: If State == Shared:
Bit vector of all of the nodes in system that have line in shared state.
If State == Modified:
Denotes which node has line in cache exclusively

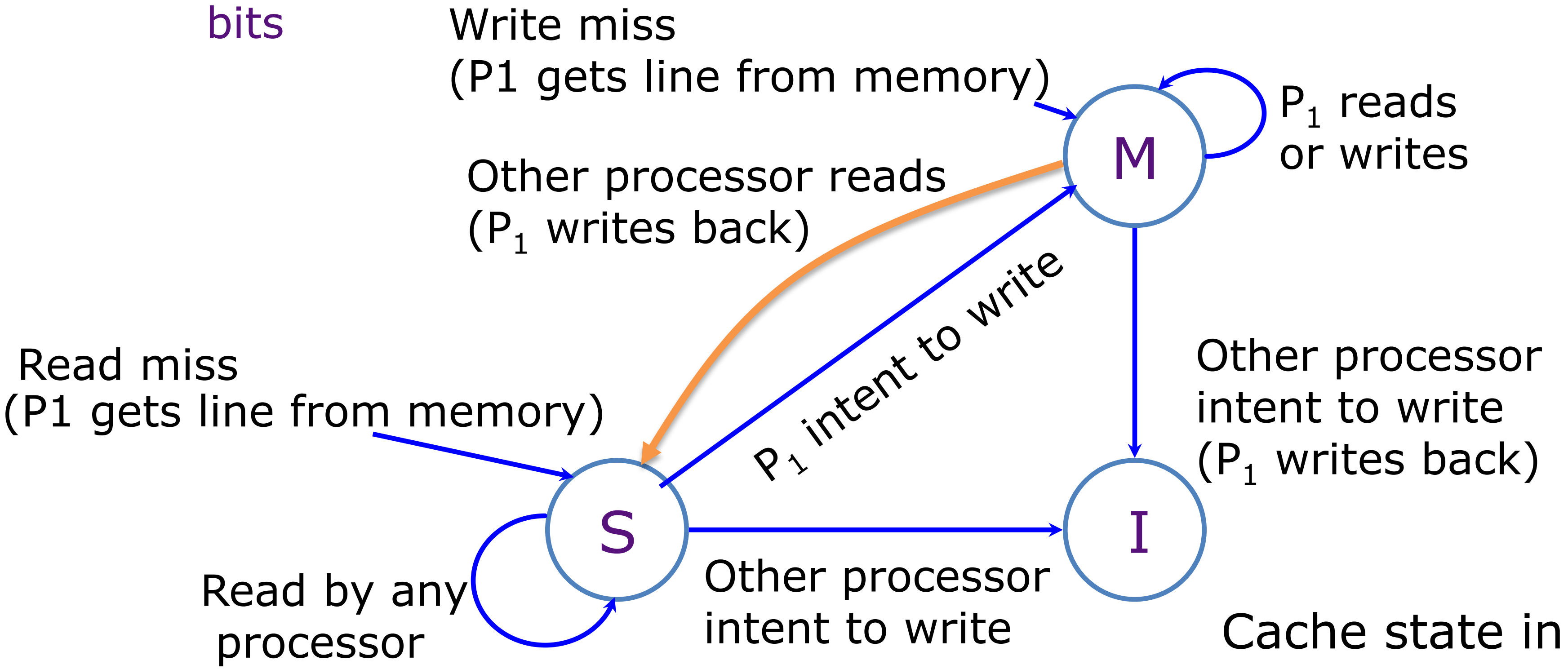
Cache State Transition Diagram

The MSI protocol

Each cache line has state bits

M: Modified
S: Shared
I: Invalid





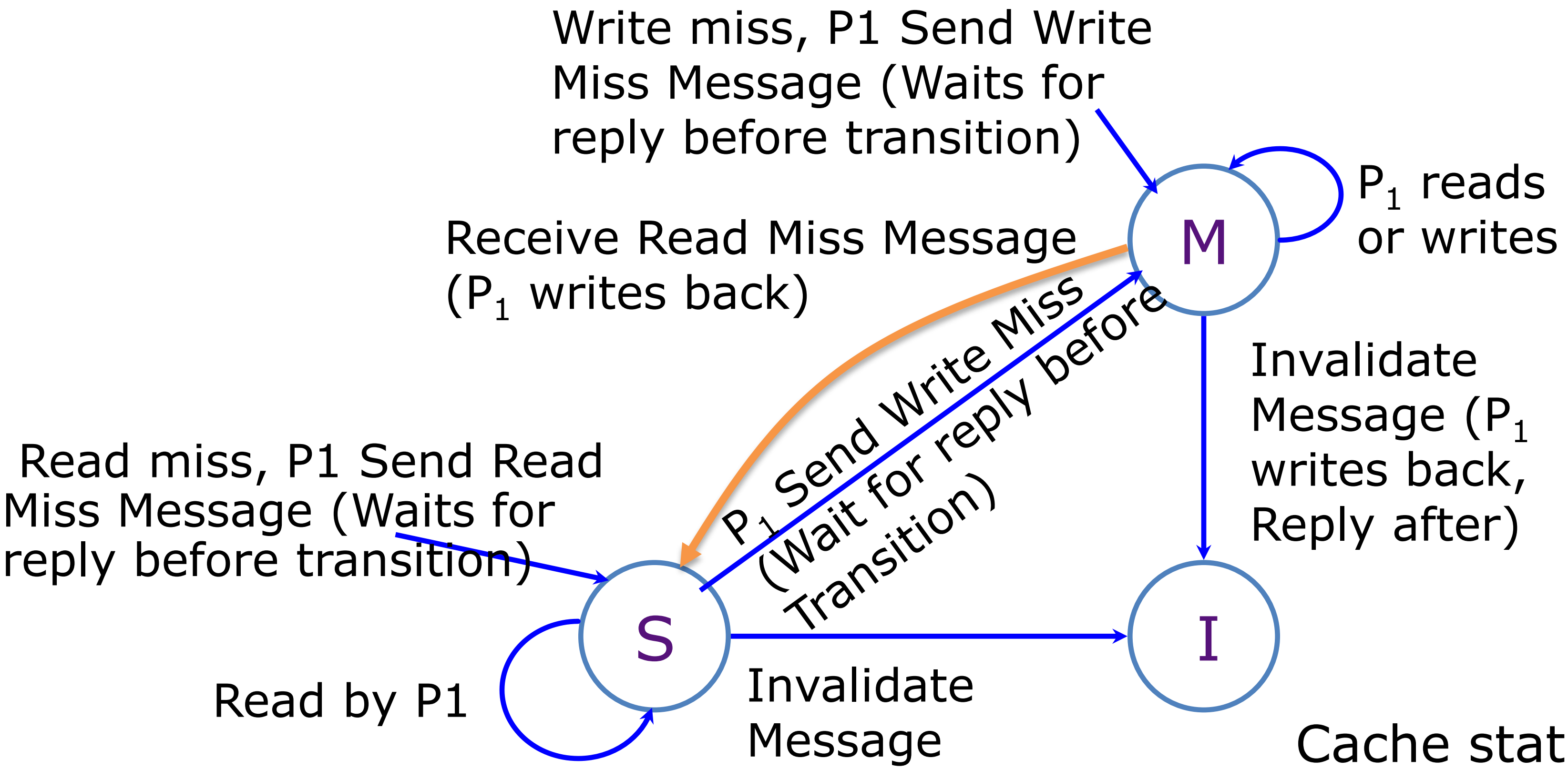
Write miss (P1 gets line from memory)
M
P_1 reads or writes
Other processor reads (P_1 writes back)
P_1 intent to write
Read miss (P1 gets line from memory)
Other processor intent to write (P_1 writes back)
S
I
Read by any processor
Other processor intent to write

Cache state in processor P_1

Cache State Transition Diagram

For Directory Coherence

M: Modified
S: Shared
I: Invalid



M

S

I

Write miss, P1 Send Write Miss Message (Waits for reply before transition)

P_1 reads or writes

Receive Read Miss Message (P_1 writes back)

P_1 Send Write Miss (Wait for reply before Transition)

Invalidate Message (P_1 writes back, Reply after)

Read miss, P1 Send Read Miss Message (Waits for reply before transition)

Read by P1

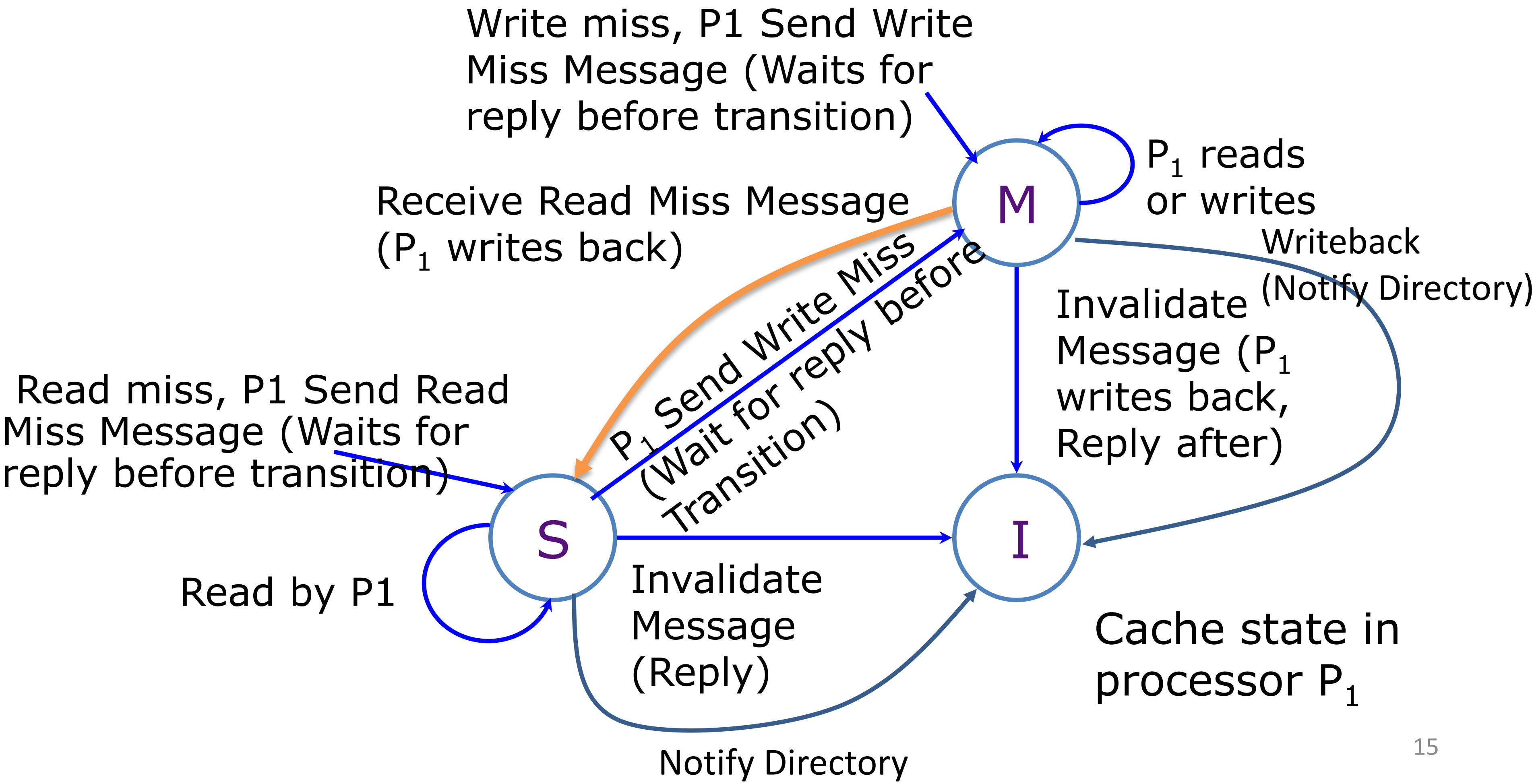
Invalidate Message (Reply)

Cache state in processor P_1

Cache State Transition Diagram

For Directory Coherence

M: Modified
S: Shared
I: Invalid



Write miss, P1 Send Write Miss Message (Waits for reply before transition)

P_1 reads or writes

Receive Read Miss Message (P_1 writes back)

P_1 Send Write Miss (Wait for reply before Transition)

Writeback (Notify Directory)

Invalidate Message (P_1 writes back, Reply after)

Read miss, P1 Send Read Miss Message (Waits for reply before transition)

Read by P1

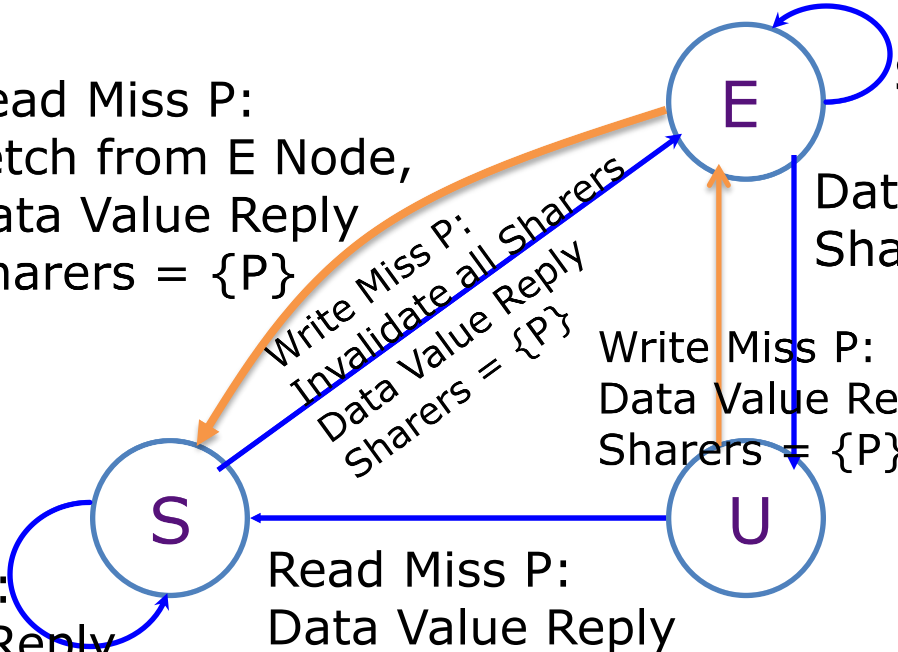
Invalidate Message (Reply)

Notify Directory

Cache state in processor P_1

Directory State Transition Diagram

U: Uncached
S: Shared
E: Exclusive



Write Miss P:
Fetch/Invalidate
From E Node,
Data Value Reply
Sharers = {P}

Read Miss P:
Fetch from E Node,
Data Value Reply
Sharers = {P}

Write Miss P:
Invalidate all Sharers
Data Value Reply
Sharers = {P}

Data Write-Back P:
Sharers = {}

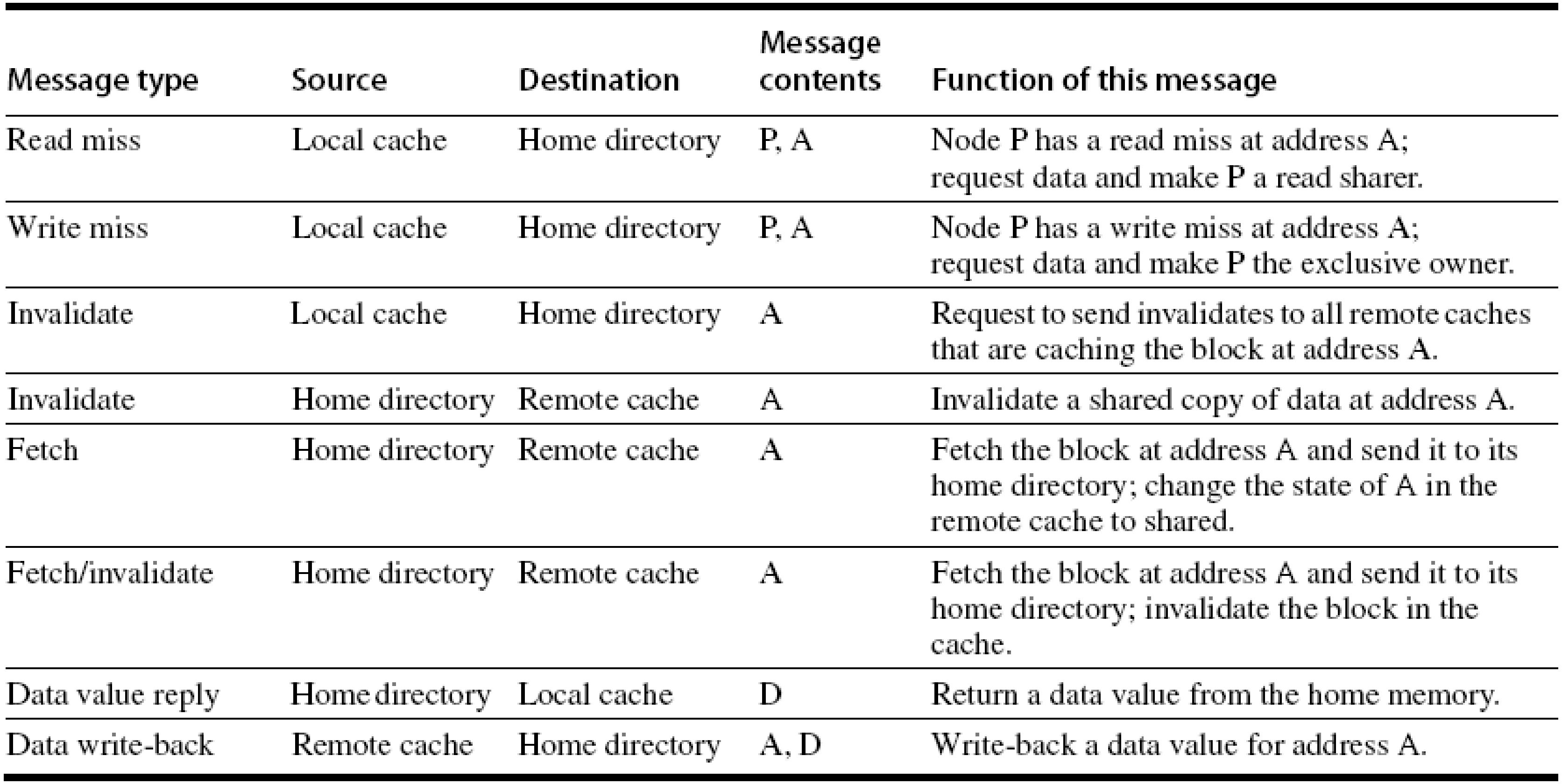
Write Miss P:
Data Value Reply
Sharers = {P}

Read Miss P:
Data Value Reply
Sharers =
Sharers +{P}

Read Miss P:
Data Value Reply
Sharers = {P}

State of Cache
Line in Directory

Message Types



Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Node P has a read miss at address A; request data and make P a read sharer.
Write miss	Local cache	Home directory	P, A	Node P has a write miss at address A; request data and make P the exclusive owner.
Invalidate	Local cache	Home directory	A	Request to send invalidates to all remote caches that are caching the block at address A.
Invalidate	Home directory	Remote cache	A	Invalidate a shared copy of data at address A.
Fetch	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared.
Fetch/invalidate	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; invalidate the block in the cache.
Data value reply	Home directory	Local cache	D	Return a data value from the home memory.
Data write-back	Remote cache	Home directory	A, D	Write-back a data value for address A.

Multiple Logical Communication Channels Needed

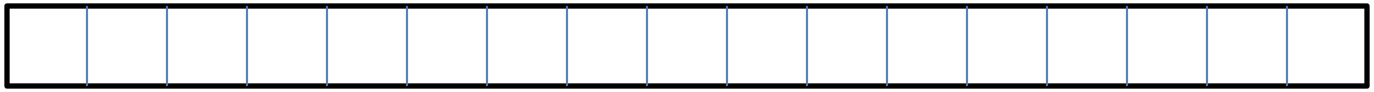
- Responses queued behind requests can lead to deadlock
- Many different message types, need to determine which message type can create more messages
- Segregate flows onto different logical/physical channels

Memory Ordering Point

- Just like in bus based snooping protocol, need to guarantee that state transitions are atomic
- Directory used as ordering point
  - Whichever message reaches home directory first wins
  - Other requests on same cache line given negative acknowledgement (NACK)
- NACK causes retry from other node
- Forward progress guarantee needed
  - After node acquires line, need to commit at least one memory operation before transitioning invalidating line

Scalability of Directory Sharer List Storage

- Full-Map Directory (Bit per cache)



- Limited Pointer (Keep list of base-2 encoded sharers)

Overflow	0x26	0x10			

- LimitLess (Software assistance)

Overflow	0x26	0x10			

Beyond Simple Directory Coherence

- On-chip coherence (Leverage fast on-chip communications to speed up or simplify protocol)
- Cache Only Memory Architectures (COMA)
- Large scale directory systems (Scalability of directory messages and sharer list storage)

SGI UV 1000 (Origin Descendant)

Maximum Memory: 16TB

Maximum Processors: 256

Maximum Cores: 2560

Topology: 2D Torus

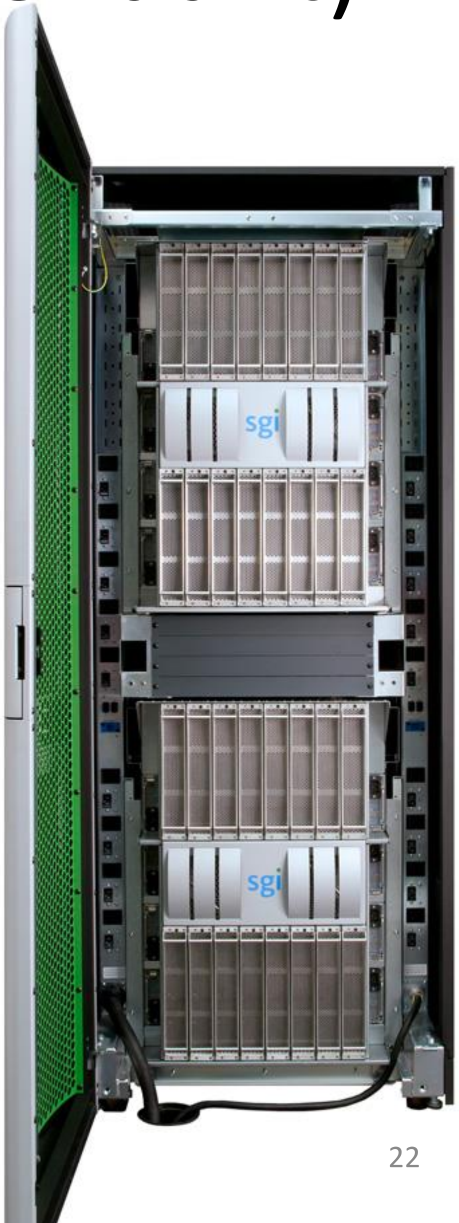
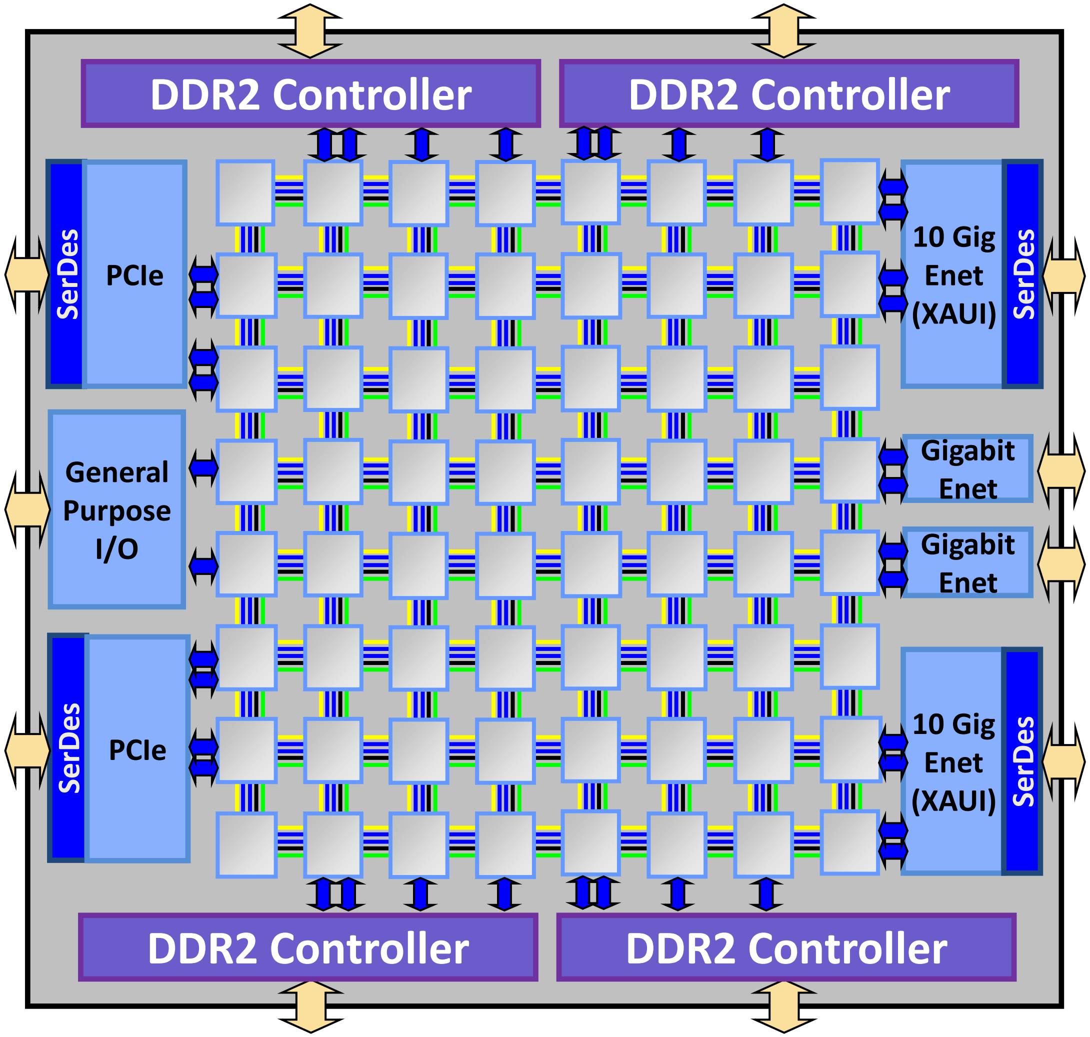


Image Credit: SGI

TILE64Pro

- 64 Cores
- 2D Mesh
- 4 Memory Controllers
- 3 Memory Networks
  - Divide different flows of traffic
- Each node can be a home



Beyond ELE 475

- Computer Architecture Research
  - International Symposium on Computer Architecture (ISCA)
  - International Symposium on Microarchitecture (MICRO)
  - Architectural Support for Programing Languages and Operating Systems (ASPLOS)
  - International Symposium on High Performance Computer Architecture (HPCA)
- Build some chips / FPGA
- Parallel Computer Architecture
- ELE 580A Parallel Computation (Princeton Only)
  - Graduate Level, Using Primary Sources

Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - Christopher Batten (Cornell)

- MIT material derived from course 6.823
- UCB material derived from course CS252 & CS152
- Cornell material derived from course ECE 4750