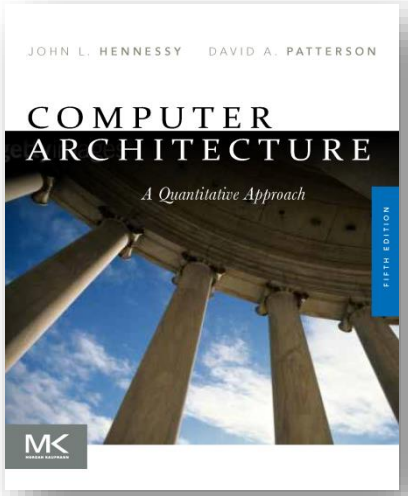


Chapter 5

Thread-Level Parallelism

Adapted by Prof. Gheith Abandah

Contents

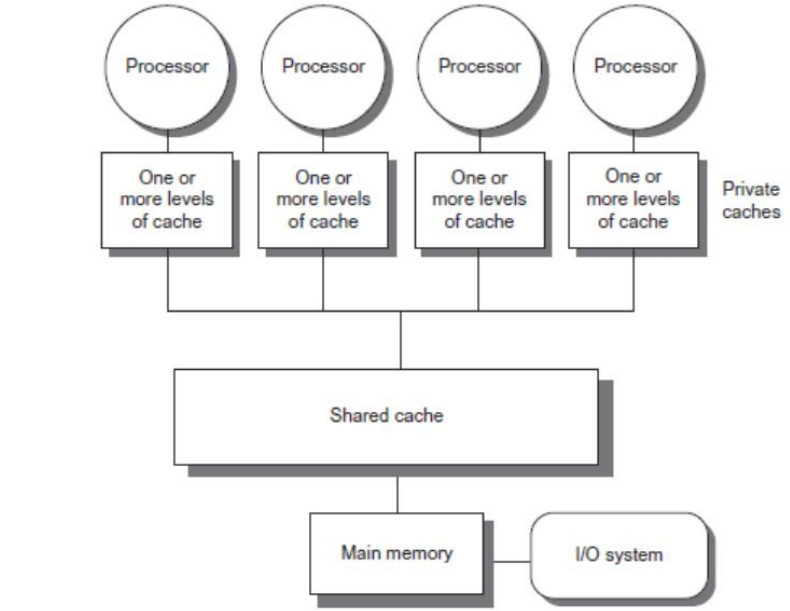
- Introduction
- Centralized Shared-Memory Architectures
- Performance of Symmetric Shared-Memory Multiprocessors
- Distributed Shared-Memory and Directory-Based Coherence
- Synchronization
- Models of Memory Consistency
- Example Multicore Processors
- Fallacies and Pitfalls

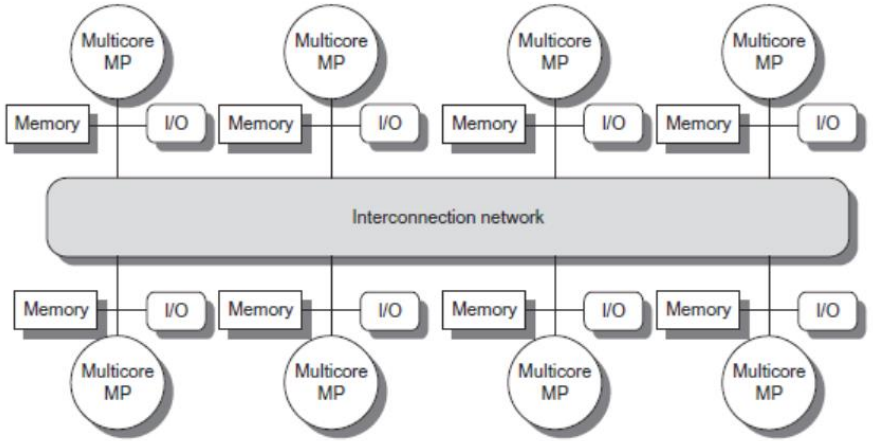
Introduction

- Thread-Level parallelism
  - Have multiple program counters
  - Uses MIMD model
  - Targeted for tightly-coupled shared-memory multiprocessors
- For *n* processors, need *n* threads
- Amount of computation assigned to each thread = grain size
  - Threads can be used for data-level parallelism, but the overheads may outweigh the benefit

Multiprocessor Types

- Symmetric multiprocessors (SMP)
  - Small number of cores
  - Share single memory with uniform memory latency
- Distributed shared memory (DSM)
  - Memory distributed among processors
  - Non-uniform memory access/latency (NUMA)
  - Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks





Contents

- Introduction
- Centralized Shared-Memory Architectures
- Performance of Symmetric Shared-Memory Multiprocessors
- Distributed Shared-Memory and Directory-Based Coherence
- Synchronization
- Models of Memory Consistency
- Example Multicore Processors
- Fallacies and Pitfalls

Contents

- Centralized Shared-Memory Architectures
  - Cache Coherence Problem
  - Enforcing Coherence Schemes
  - Snooping Coherence Protocols
  - Basic Implementation Techniques
  - Extensions
  - Limitations of Snooping Protocols

Cache Coherence Problem

- Processors may see different values through their caches:

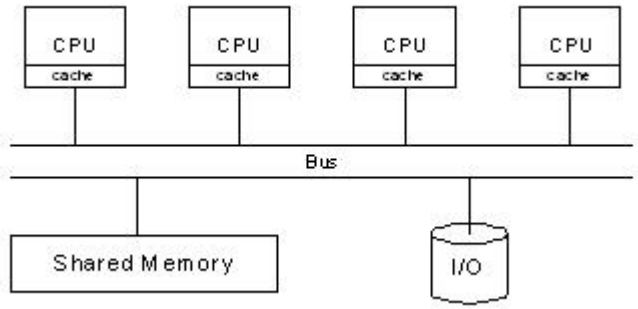
Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

- Cache coherence requirement:
  - All reads by any processor must return the most recently written value
  - Writes to the same location by any two processors are seen in the same order by all processors

Enforcing Coherence Schemes

- Coherent caches provide:
  - *Migration*: movement of data
  - *Replication*: multiple copies of data

- Cache coherence protocols
  1. Snooping
     - Each core tracks sharing status of each block



  2. Directory based
     - Sharing status of each block is kept in one location

Snooping Coherence Protocols

1. Write invalidate protocols
   - On write, invalidate all other copies
   - Use bus itself to serialize
     - Write cannot complete until bus access is obtained

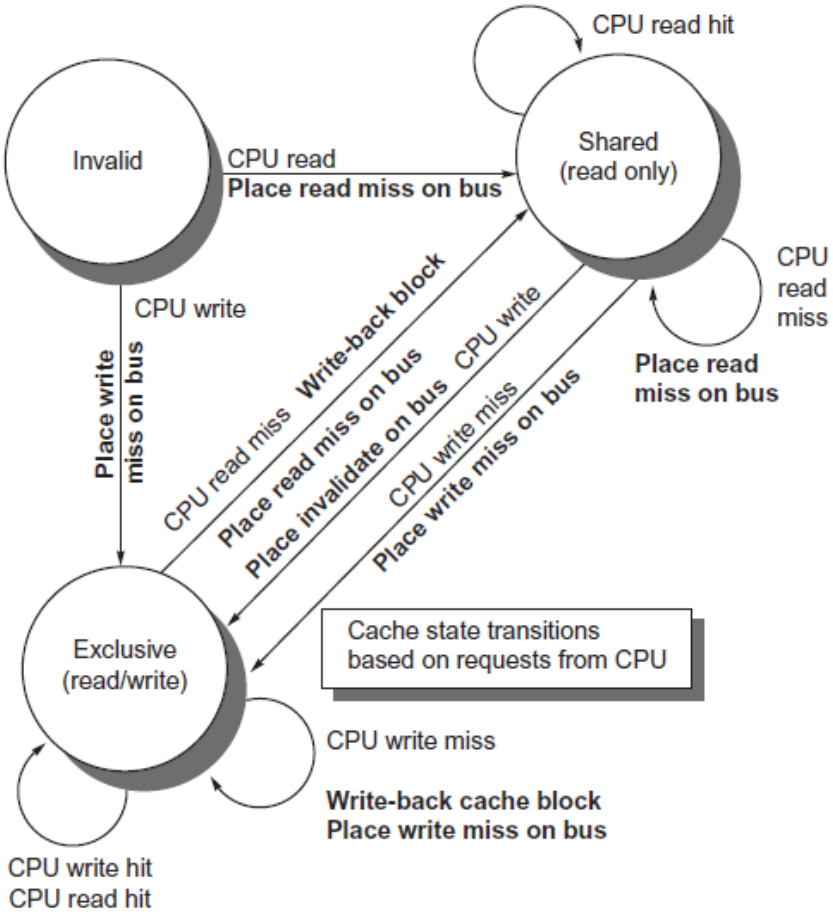
Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
				0
Processor A reads X	Cache miss for X	0		0
Processor B reads X	Cache miss for X	0	0	0
Processor A writes a 1 to X	Invalidation for X	1		0
Processor B reads X	Cache miss for X	1	1	1

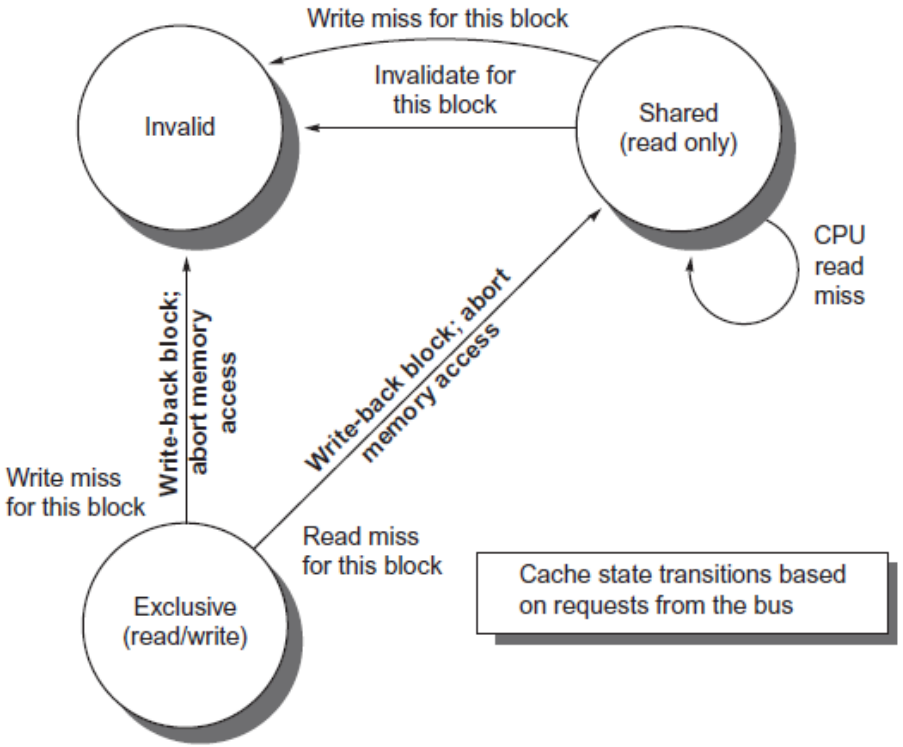
2. Write update protocols
   - On write, update all copies

Basic Implementation Techniques

- Locating an item when a read miss occurs
  - In write-back cache, the updated value must be sent to the requesting processor
- Cache lines marked as shared or exclusive/modified
  - Only writes to shared lines need an invalidate broadcast
    - After this, the line is marked as exclusive

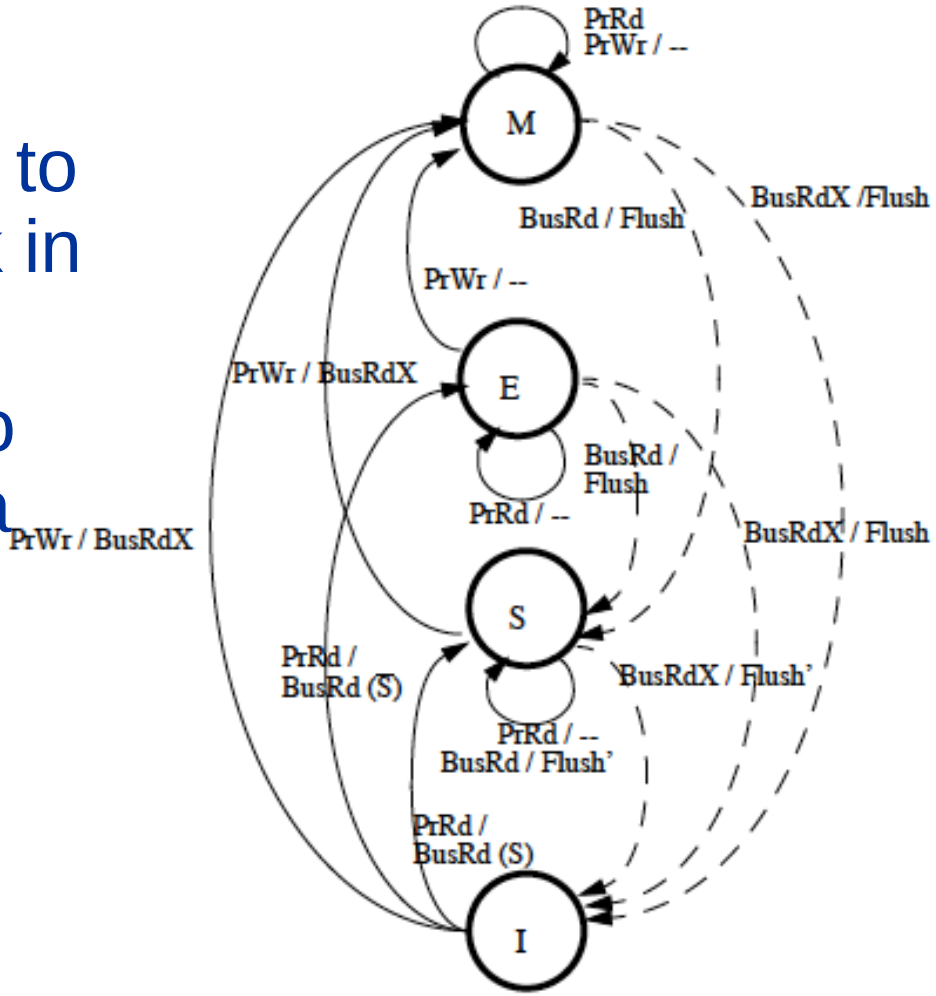
Example Protocol: MSI





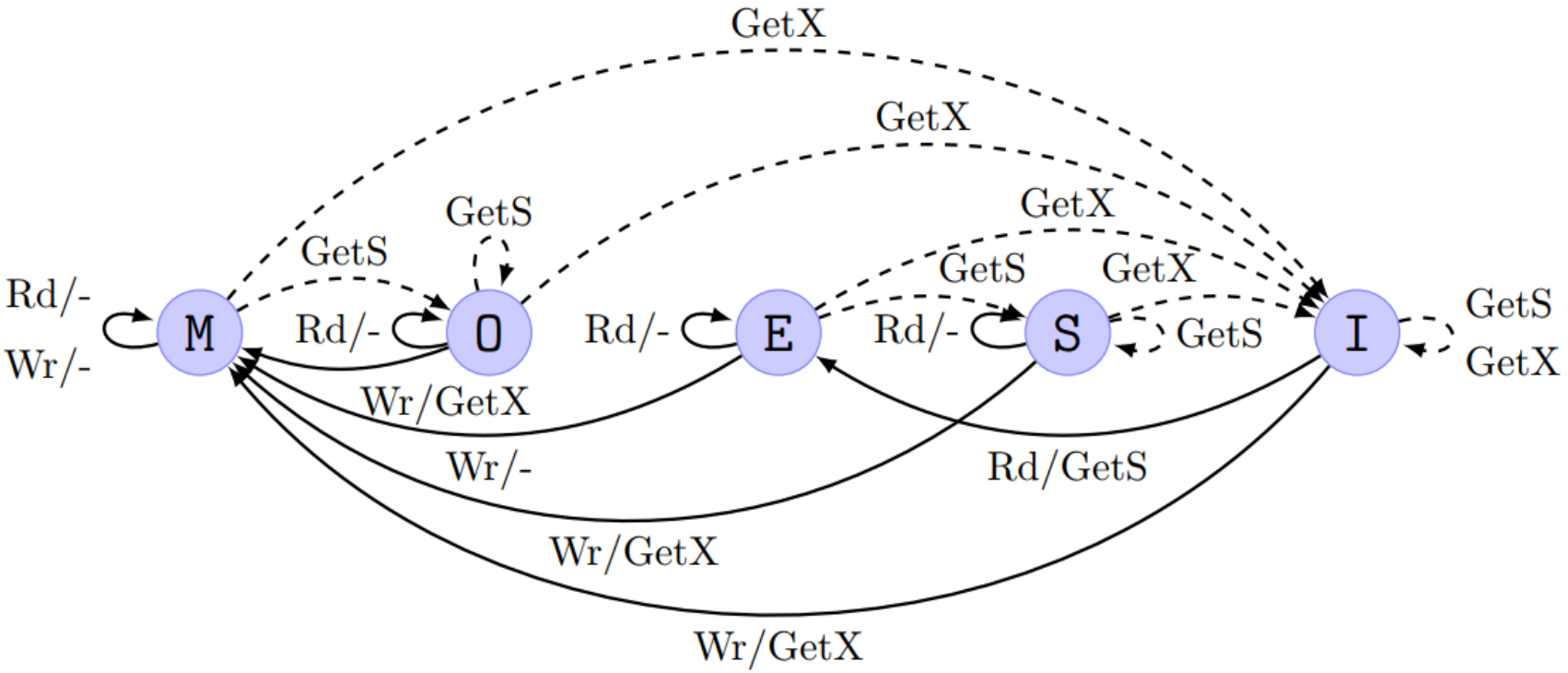
Extensions: The MESI Protocol

- Add exclusive state to indicate clean block in only one cache.
- Prevents needing to write invalidate on a write.



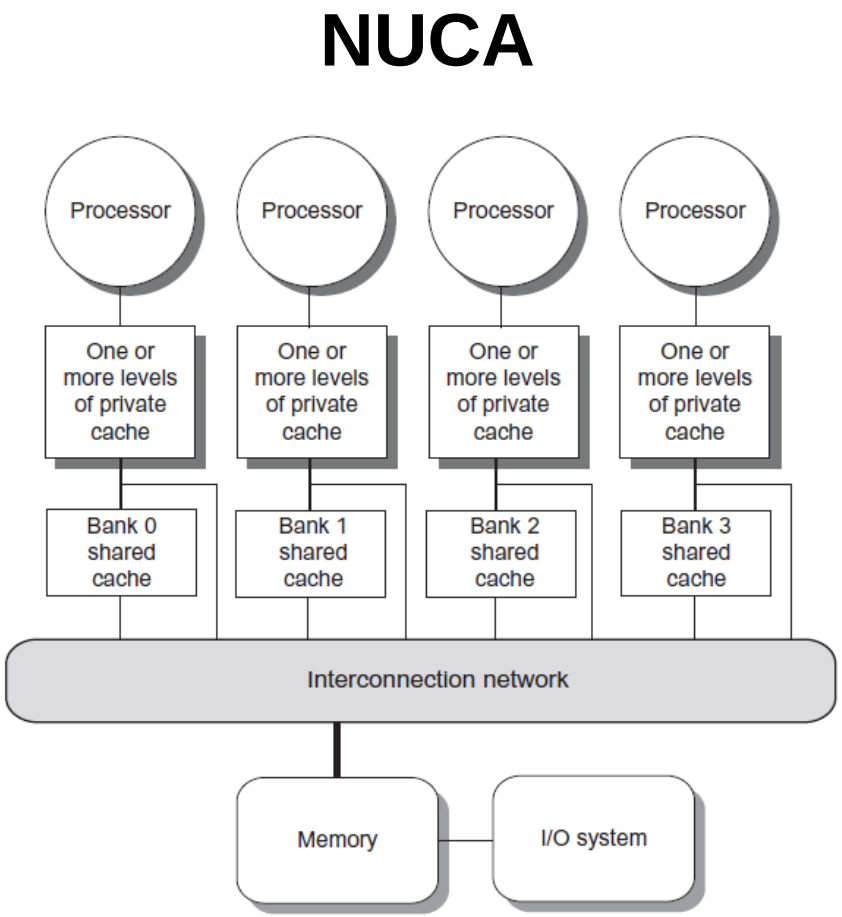
Extensions: The MEOSI Protocol

- Add owned state to indicate block owned by the cache and out-of-date in memory.
- The owner keeps providing it on misses.



Limitations of Snooping Protocols

- Shared memory bus and snooping bandwidth is bottleneck for scaling symmetric multiprocessors.
- Solutions:
  - Duplicating tags
  - Partition the inclusive, shared L3$ on the cores.
  - Place directory in outermost cache
  - Use crossbars or point-to-point networks with banked memory



Contents

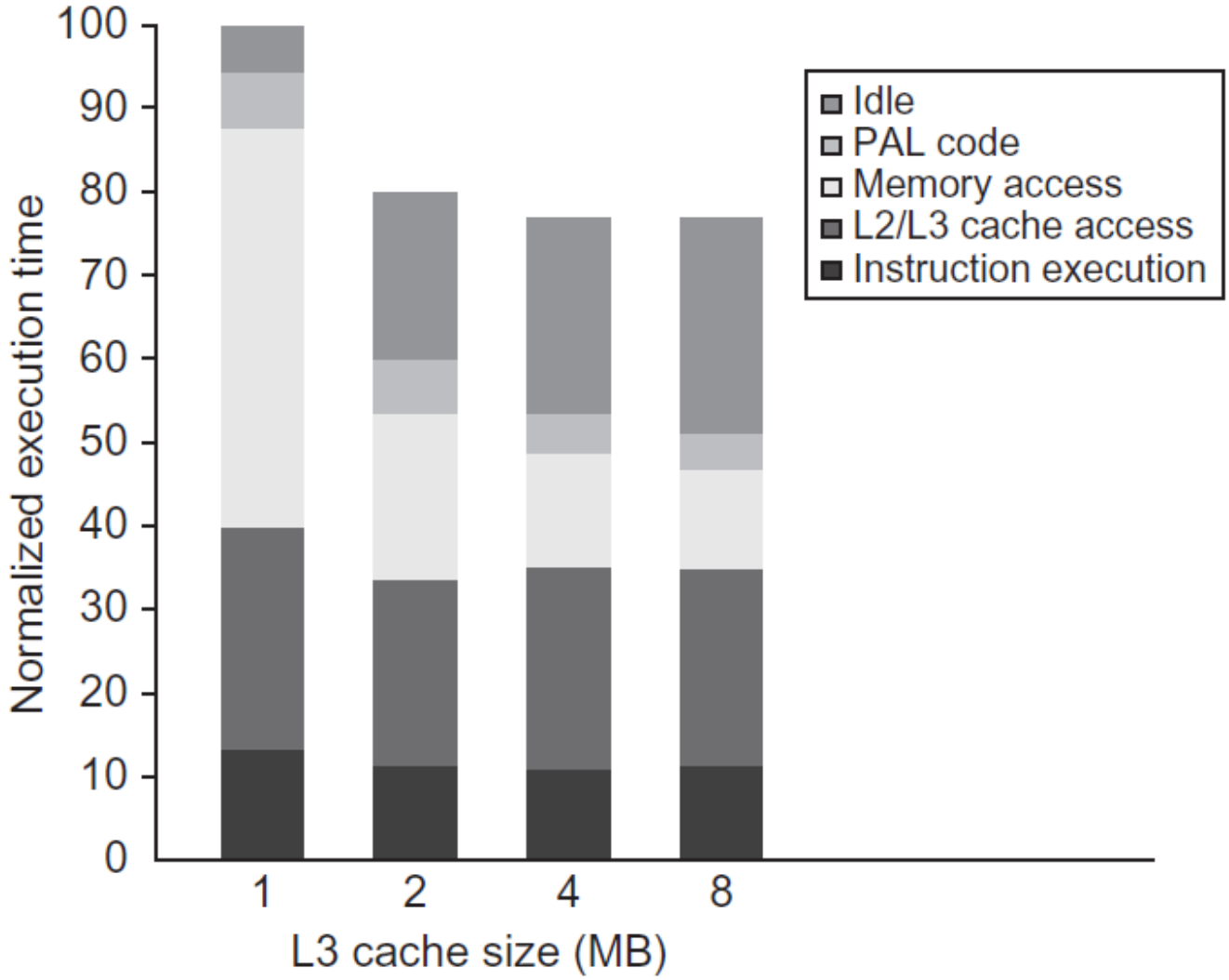
- Introduction
- Centralized Shared-Memory Architectures
- Performance of Symmetric Shared-Memory Multiprocessors
- Distributed Shared-Memory and Directory-Based Coherence
- Synchronization
- Models of Memory Consistency
- Example Multicore Processors
- Fallacies and Pitfalls

SMP Performance

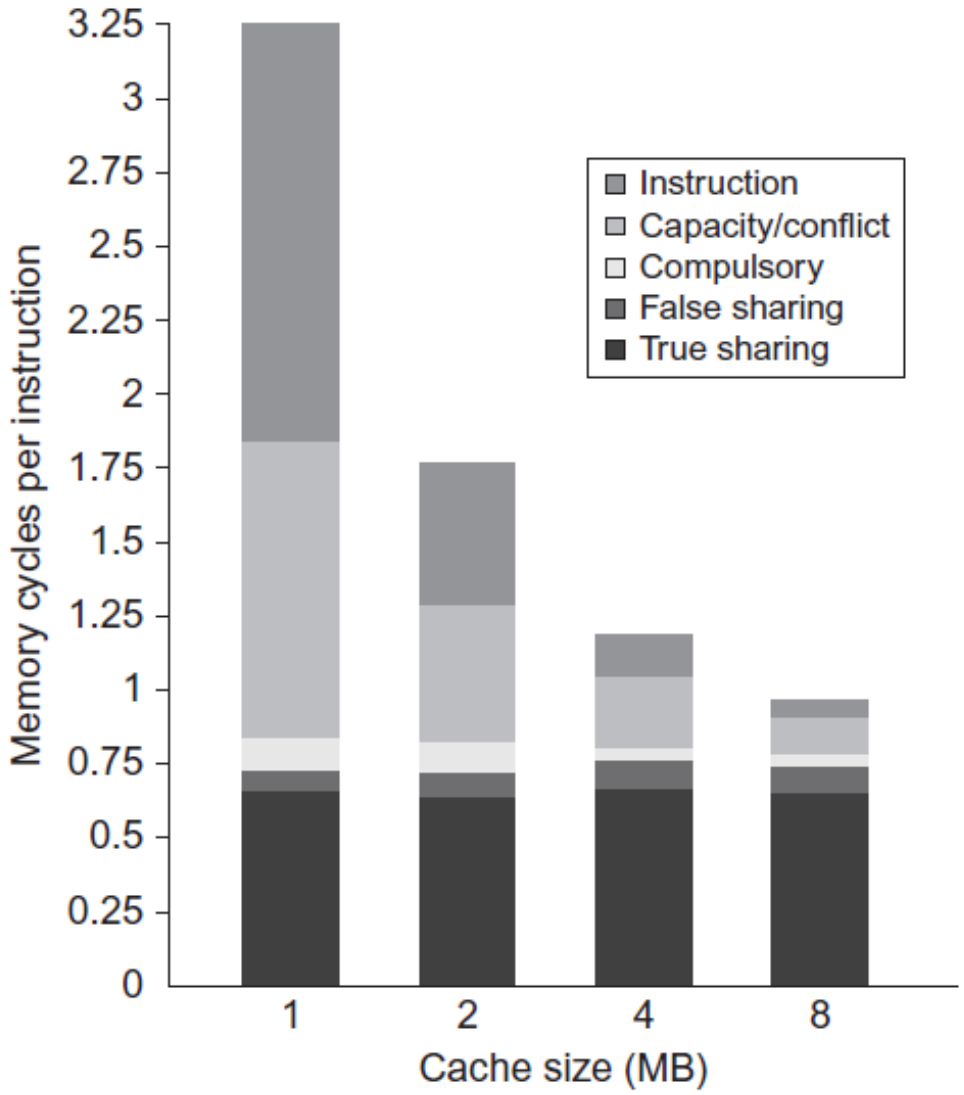
- Coherence misses are misses due the coherence protocol.
  - True sharing misses
    - Write to shared block (transmission of invalidation)
    - Read an invalidated block
  - False sharing misses
    - Read an unmodified word in an invalidated block
- Example: Assume z1 and z2 are in one shared block.

Time	P1	P2
1	Write z1	
2		Read z2
3	Write z1	
4		Write z2
5	Read z2	

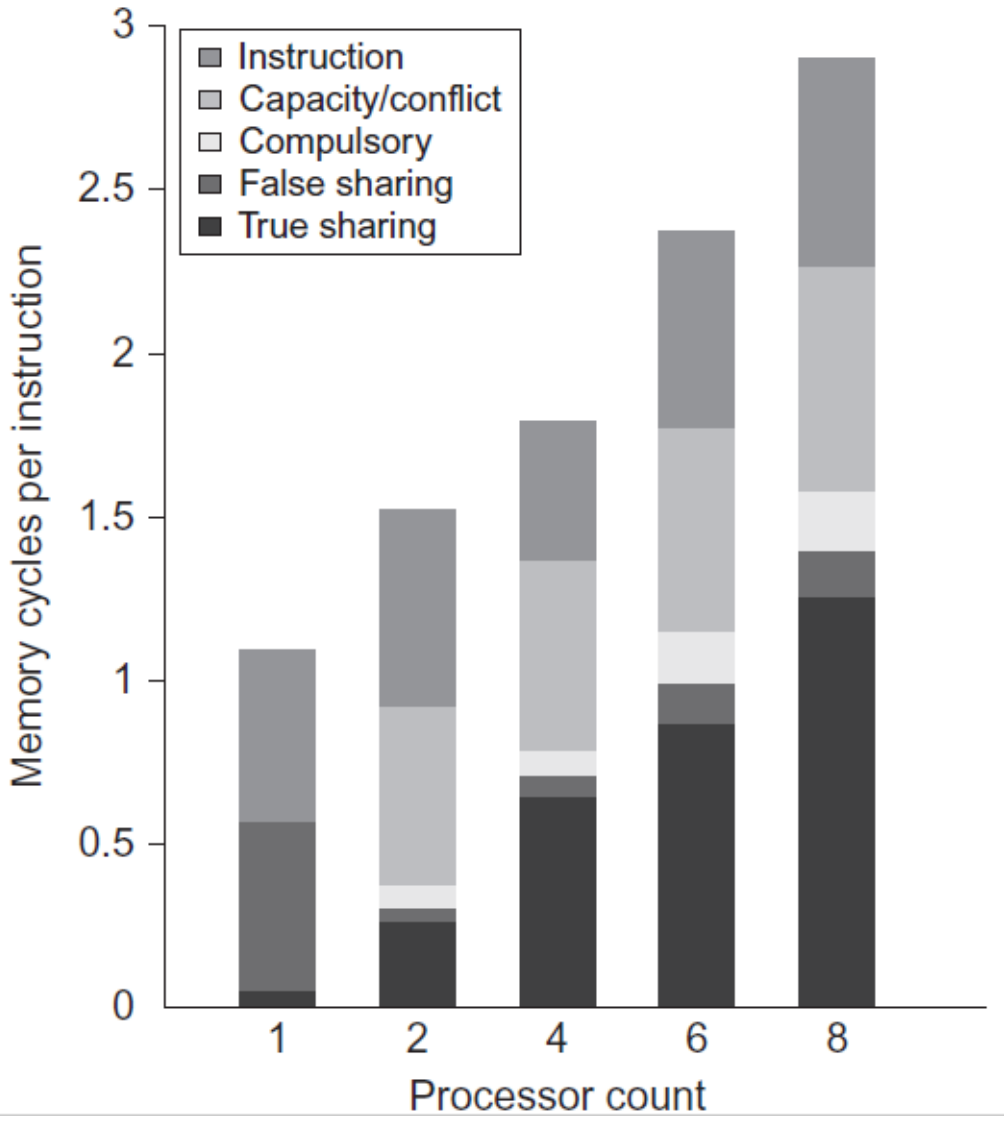
Execution Time and Cache Size



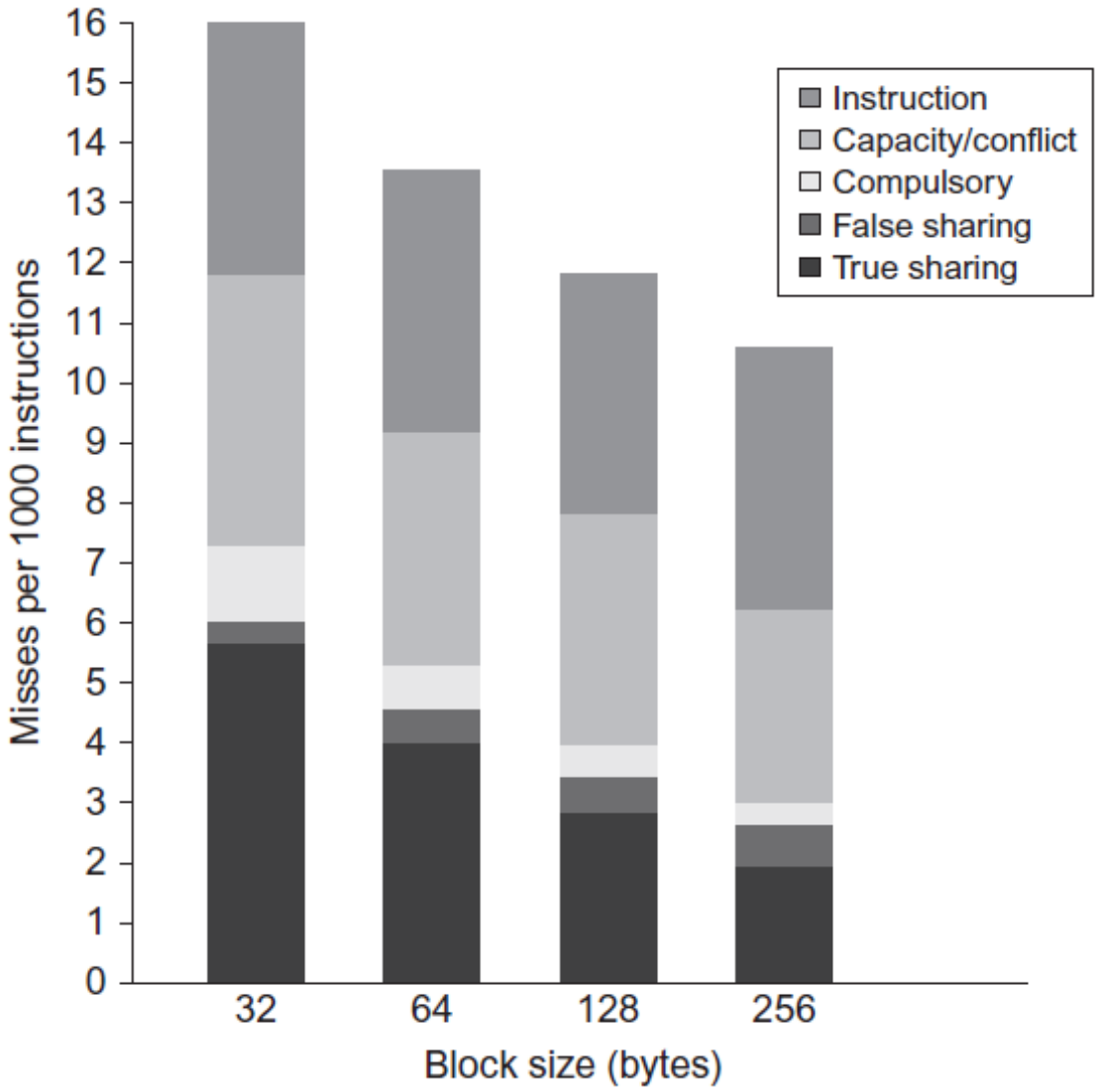
Memory Time and Cache Size



Memory Time and Processors



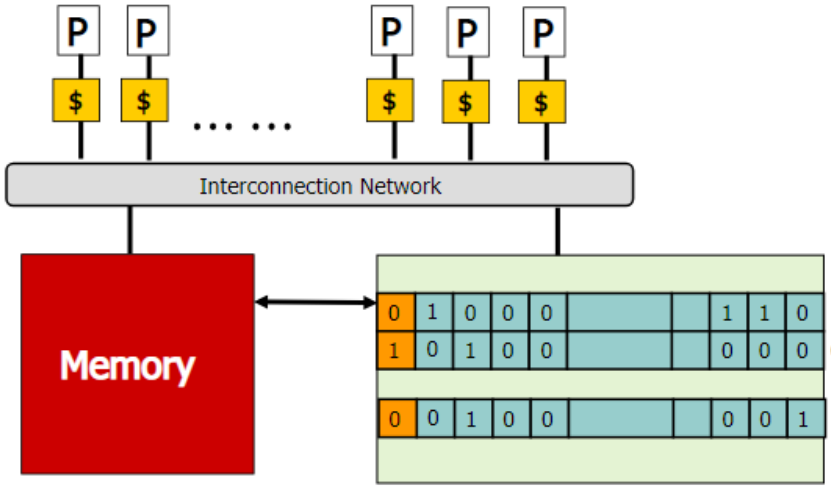
Misses and Block Size



Contents

Directory Protocols

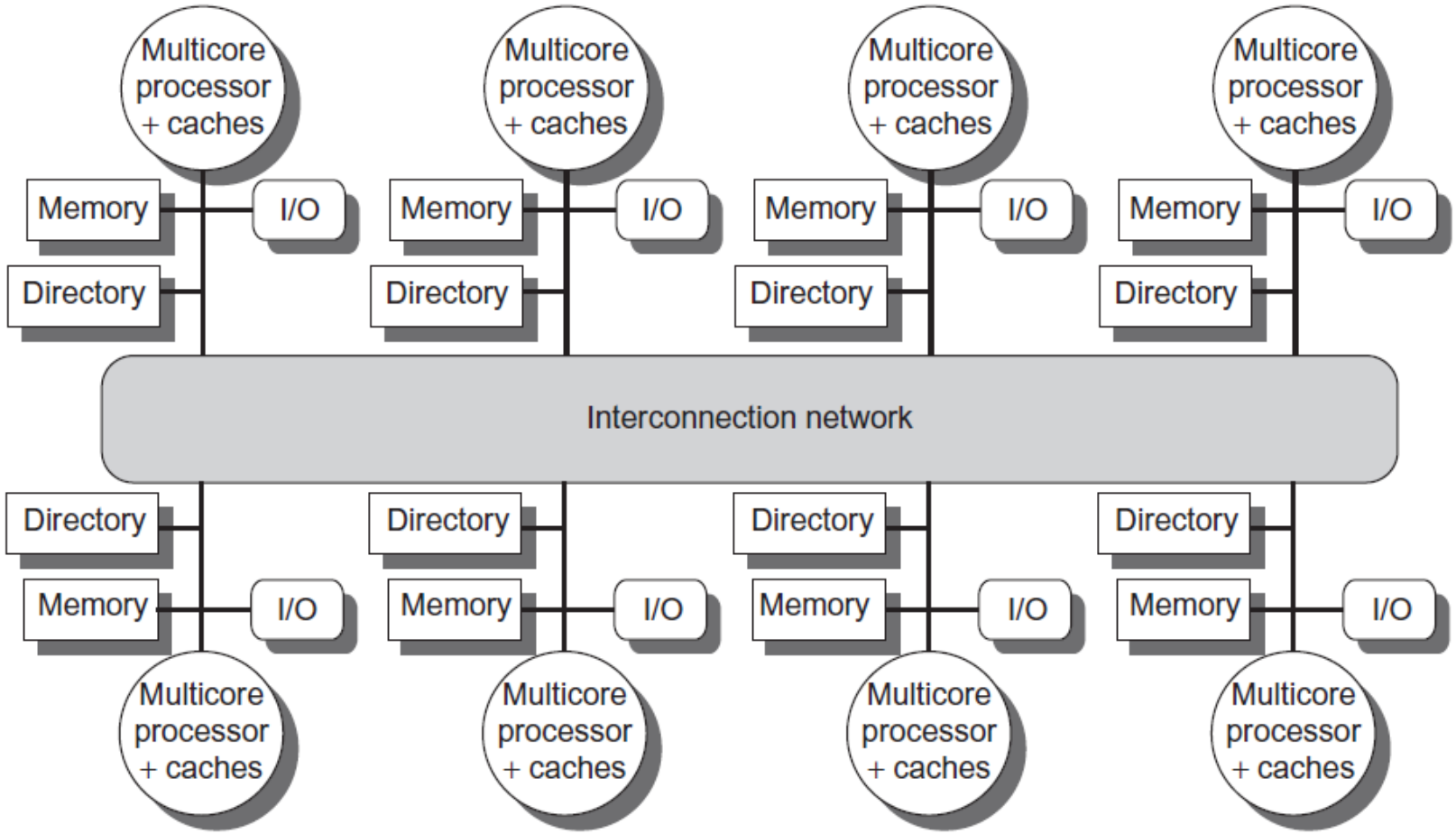
- Snooping schemes require communication among all caches on every cache miss
  - Limits scalability
  - Another approach: Use centralized directory to keep track of every block
    - Which caches have each block
    - Dirty status of each block



- Can implement it in shared L3 cache
  - Keep bit vector of size = # cores for each block in L3
  - Not scalable beyond shared L3

Directory Protocols

- Distributed memory with directories



Directory Protocol Basics

- For each block, maintain state:
  - Uncached
  - Shared
    - One or more nodes have the block cached, value in memory is up-to-date
    - Set of node IDs
  - Modified
    - Exactly one node has a copy of the cache block, value in memory is out-of-date
    - Owner node ID
- Directory maintains block states and sends invalidation messages

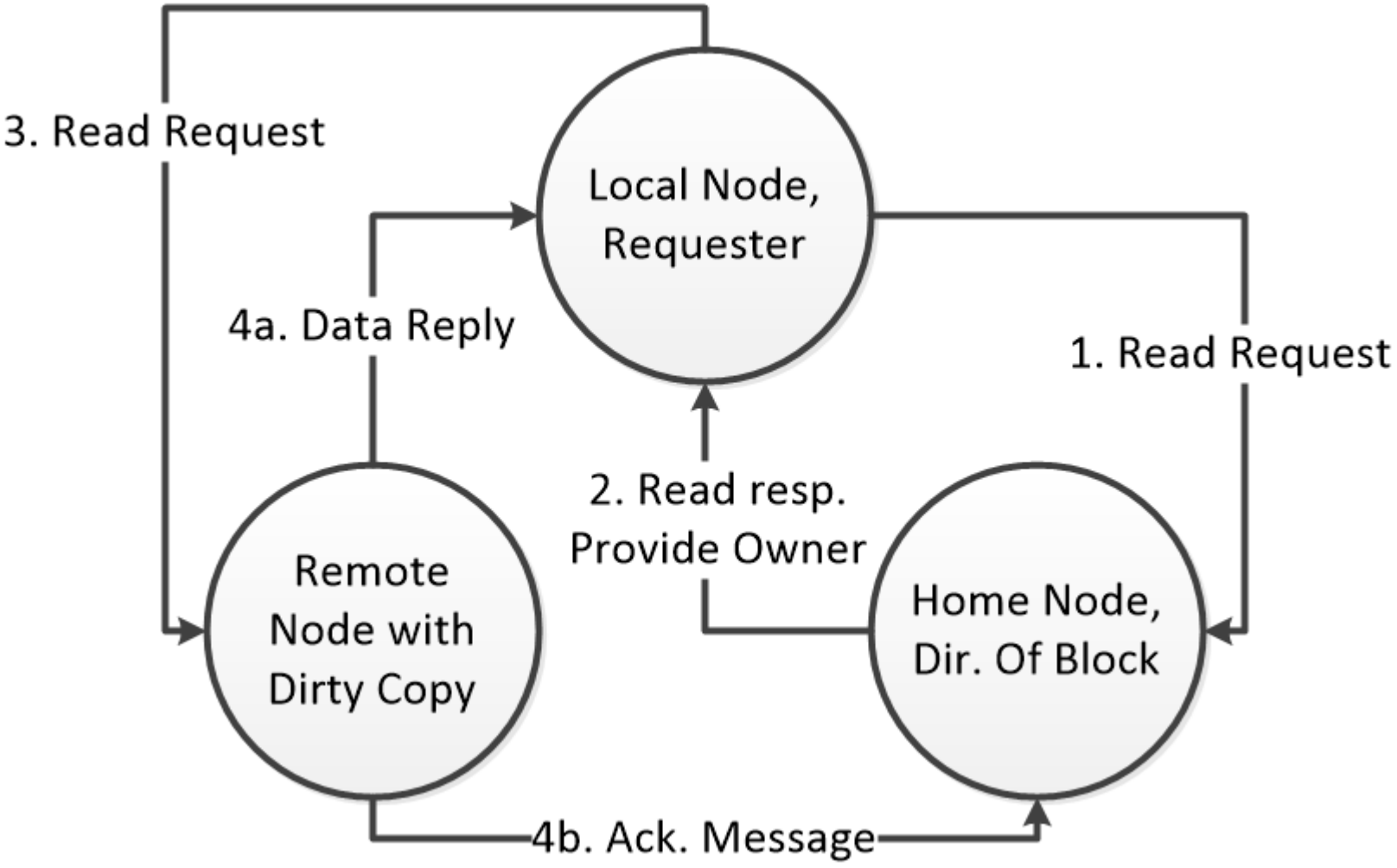
Directory Protocols Actions

1. For uncached block:
   - Read miss
     - Requesting node is sent the requested data and is made the only sharing node, block is now shared
   - Write miss
     - The requesting node is sent the requested data and becomes the sharing node, block is now exclusive
2. For shared block:
   - Read miss
     - The requesting node is sent the requested data from memory, node is added to sharing set
   - Write miss
     - The requesting node is sent the value, all nodes in the sharing set are sent invalidate messages, sharing set only contains requesting node, block is now exclusive

Directory Protocol Actions

3. For exclusive block:
   - Read miss
     - The owner is sent a data fetch message, block becomes shared, owner sends data to the directory, data written back to memory, sharers set contains old owner and requestor
   - Data write back
     - Block becomes uncached, sharer set is empty
   - Write miss
     - Message is sent to old owner to invalidate and send the value to the directory, requestor becomes new owner, block remains exclusive

Example Messages



Contents

- Introduction
- Centralized Shared-Memory Architectures
- Performance of Symmetric Shared-Memory Multiprocessors
- Distributed Shared-Memory and Directory-Based Coherence
- Synchronization
- Models of Memory Consistency
- Example Multicore Processors
- Fallacies and Pitfalls

Synchronization

- Basic building blocks:
  - Atomic exchange
    - Swaps register with memory location
  - Test-and-set
    - Sets under condition
  - Fetch-and-increment
    - Reads original value from memory and increments it in memory
  - Requires memory read and write in uninterruptable instruction

- RISC-V: load reserved/store conditional
  - If the contents of the memory location specified by the load linked are changed before the store conditional to the same address, the store conditional fails

Implementing Locks

- Atomic exchange (EXCH):
  - on the memory location specified by the contents of `x1` with the value in `x4`

```
try:  mov  x3,x4        ;move exchange value
      lr   x2,x1        ;load reserved from
      sc   x3,0(x1)     ;store conditional
      bnez x3,try       ;branch store fails
      mov  x4,x2        ;put load value in x4?
```

Implementing Locks

- Lock (not efficient)

```
        addi x2,x0,#1
lockit: EXCH x2,0(x1)       ;atomic exchange
        bnez x2,lockit      ;already locked?
```

- Lock (efficient):

```
lockit: ld   x2,0(x1)       ;load of lock
        bnez x2,lockit      ;not available-spin
        addi x2,x0,#1       ;load locked value
        EXCH x2,0(x1)       ;swap
        bnez x2,lockit      ;branch if lock isn't 0
```

Contents

Models of Memory Consistency

- Memory consistency models deal with when a processor must see a value that has been updated by another processor

- Sequential consistency:
  - Result of execution should be the same as long as:
    - Accesses on each processor were kept in order
    - Accesses on different processors were arbitrarily interleaved

- To implement, delay completion of all memory accesses until all invalidations caused by the access are completed.
  - Reduces performance!

Models of Memory Consistency

- Example:

Processor 1:	Processor 2:
A=0	B=0
…	…
A=1	B=1
if (B==0) …	if (A==0) …

- Should be impossible for both if-statements to be evaluated as true with sequential consistency.

- Possible with delayed write invalidate.

Relaxed Consistency Models

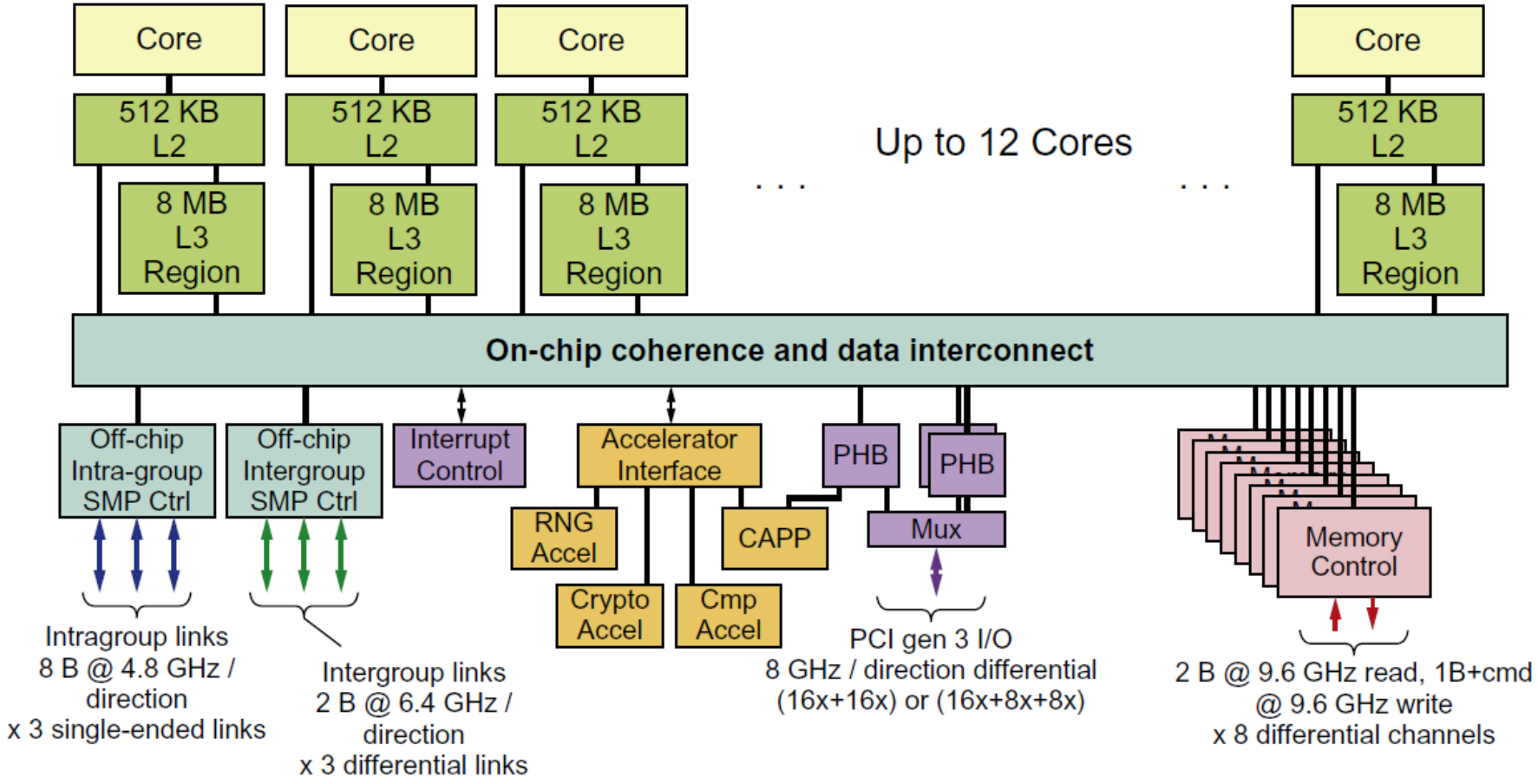
- The key idea in relaxed consistency models is to allow reads and writes to complete out of order, but to use synchronization operations to enforce ordering so that a synchronized program behaves as though the processor were sequentially consistent when needed.
- Program-enforced synchronization to force write on processor to occur before read on the other processor
  - Requires synchronization object for A and another for B
    - “Unlock” after write
    - “Lock” after read

Contents

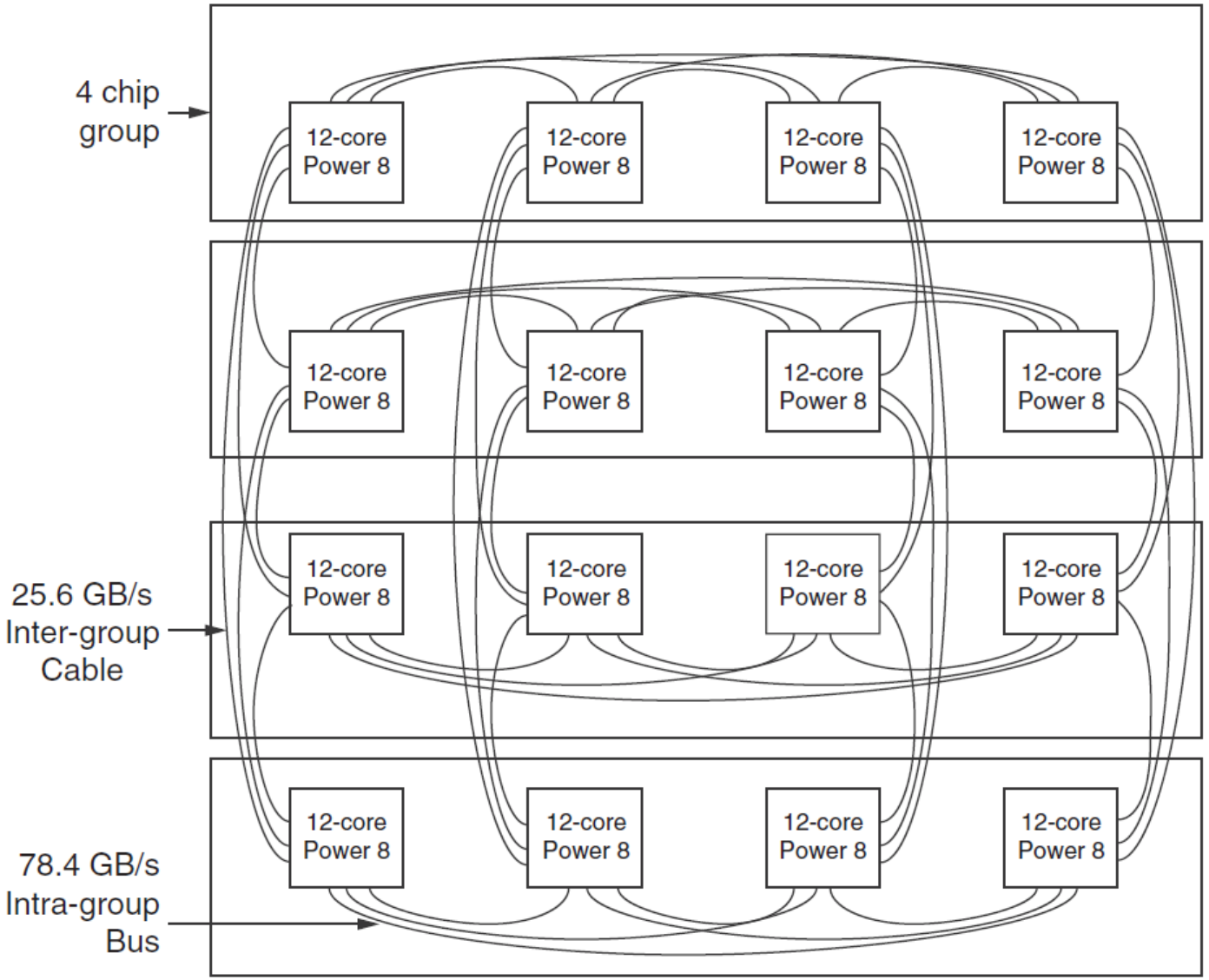
Example Multicore Processors

Feature	IBM Power8	Intel Xeon E7	Fujitsu SPARC64 X+
Cores/chip	4, 6, 8, 10, 12	4, 8, 10, 12, 22, 24	16
Multithreading	SMT	SMT	SMT
Threads/core	8	2	2
Clock rate	3.1–3.8 GHz	2.1–3.2 GHz	3.5 GHz
L1 I cache	32 KB per core	32 KB per core	64 KB per core
L1 D cache	64 KB per core	32 KB per core	64 KB per core
L2 cache	512 KB per core	256 KB per core	24 MiB shared
L3 cache	L3: 32–96 MiB: 8 MiB per core (using eDRAM); shared with nonuniform access time	10–60 MiB @ 2.5 MiB per core; shared, with larger core counts	None
Inclusion	Yes, L3 superset	Yes, L3 superset	Yes
Multicore coherence protocol	Extended MESI with behavioral and locality hints (13-states)	MESIF: an extended form of MESI allowing direct transfers of clean blocks	MOESI
Multichip coherence implementation	Hybrid strategy with snooping and directory	Hybrid strategy with snooping and directory	Hybrid strategy with snooping and directory
Multiprocessor interconnect support	Can connect up to 16 processor chips with 1 or 2 hops to reach any processor	Up to 8 processor chips directly via Quickpath; larger system and directory support with additional logic	Crossbar interconnect chip, supports up to 64 processors; includes directory support
Processor chip range	1–16	2–32	1–64
Core count range	4–192	12–576	8–1024

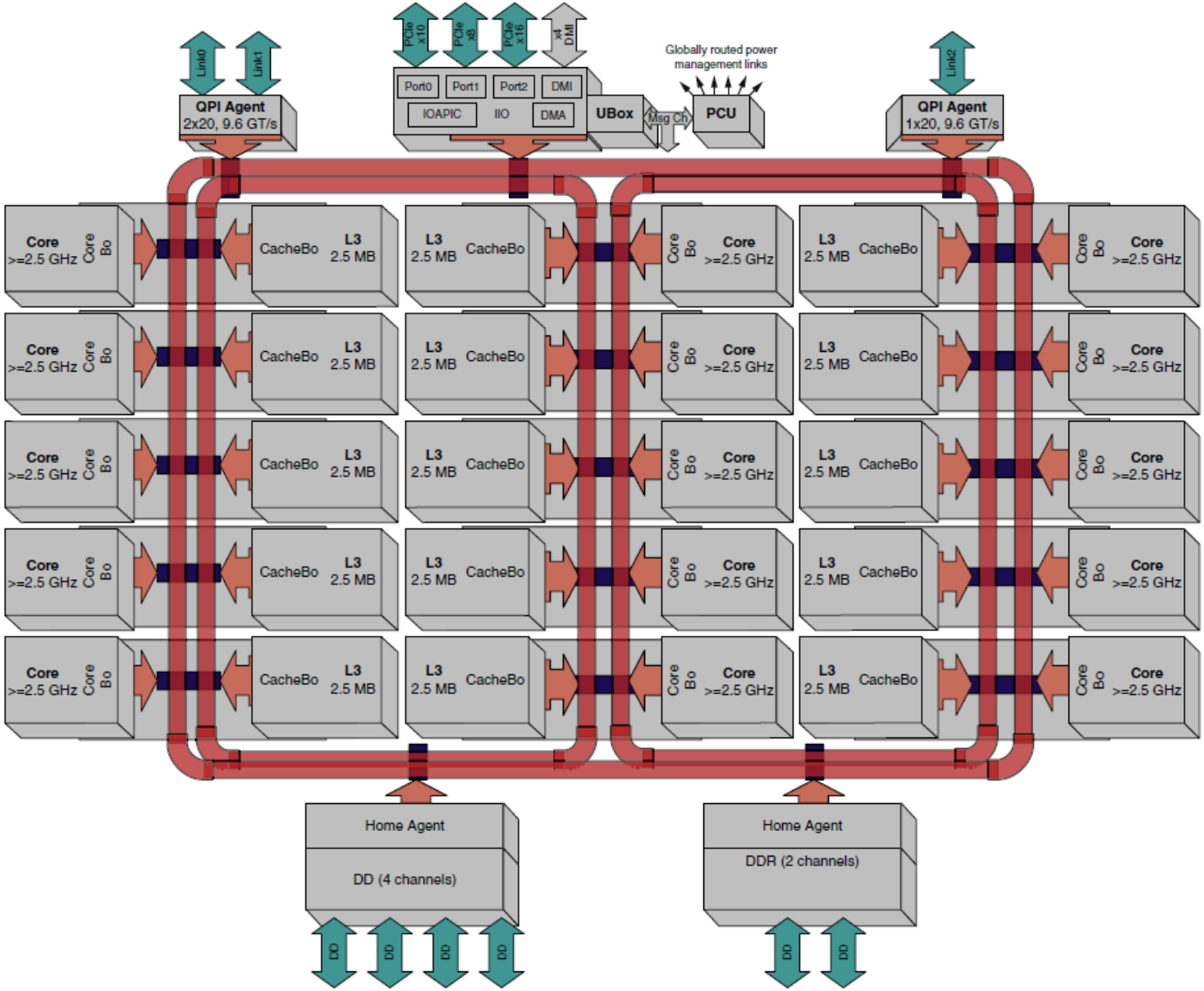
IBM Power 8



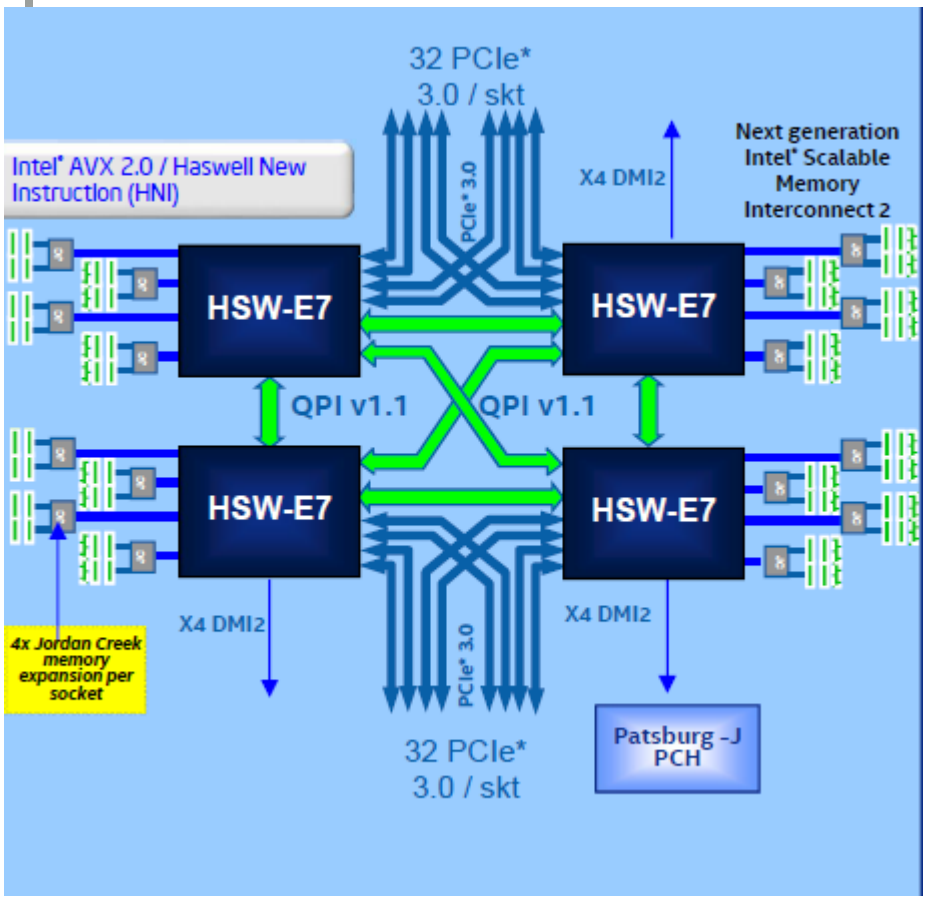
Power 8 System with 16 Chips

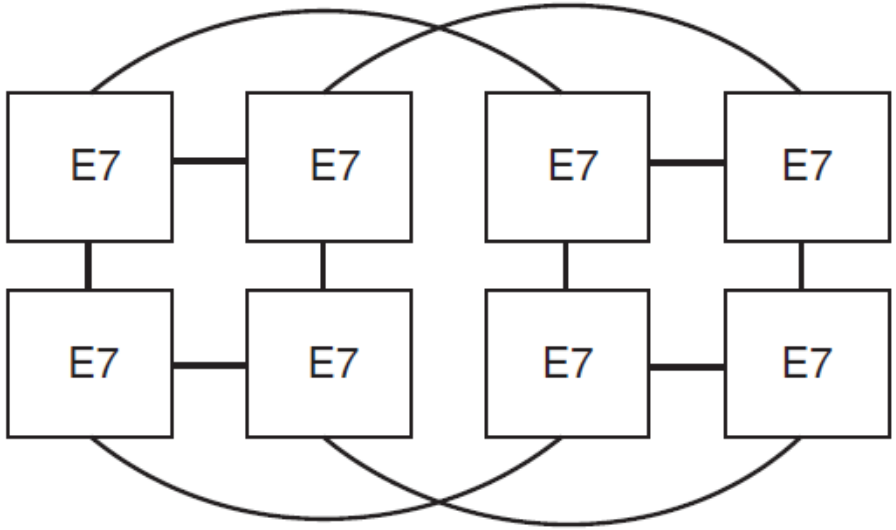


Intel Xeon E7

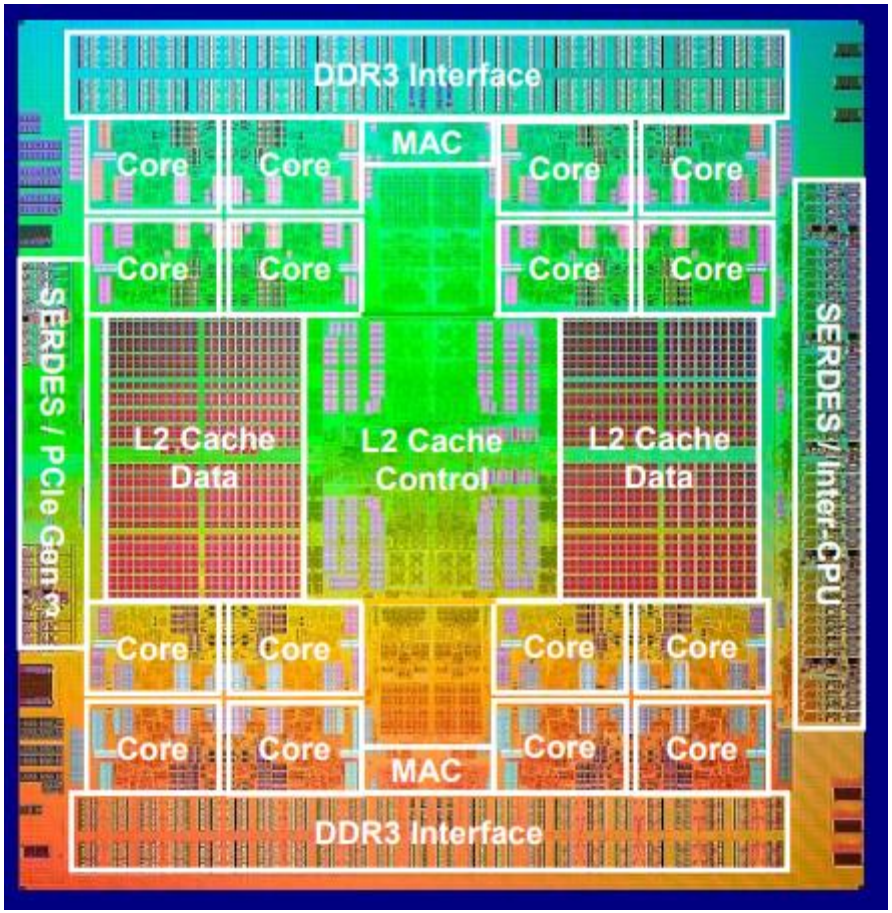


E7 System with 4 and 8 Chips

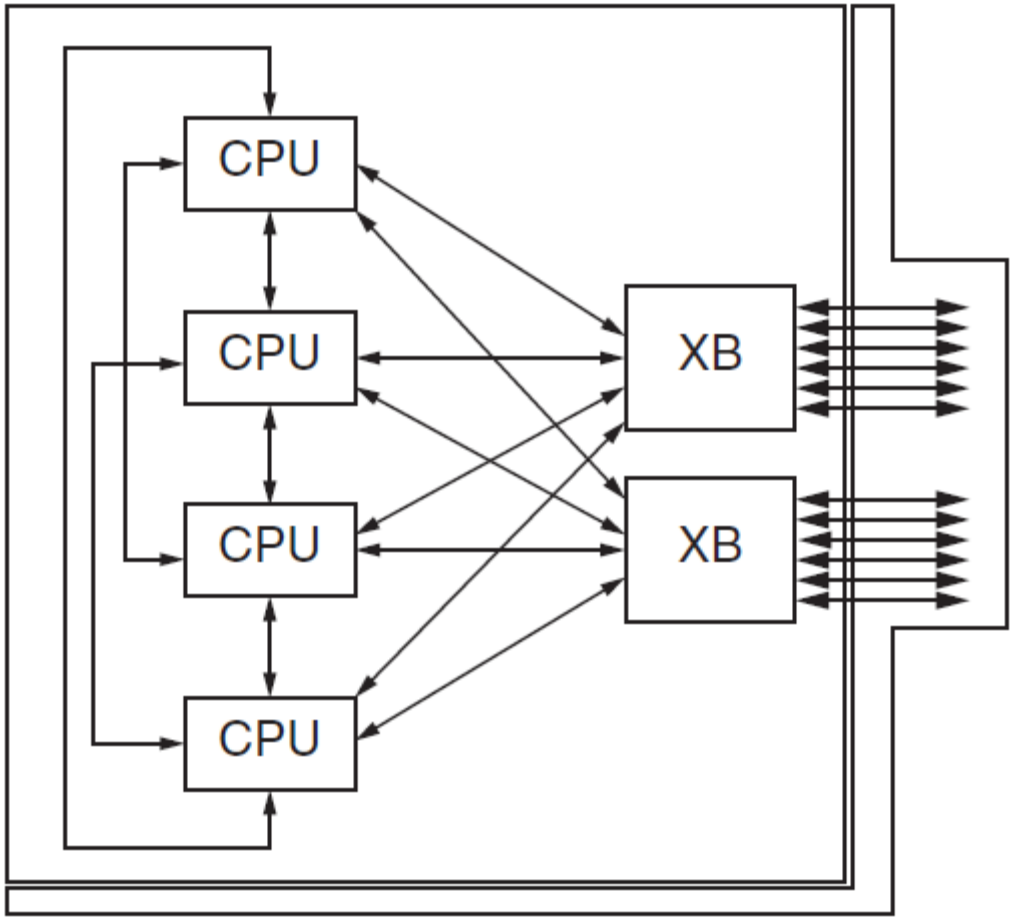




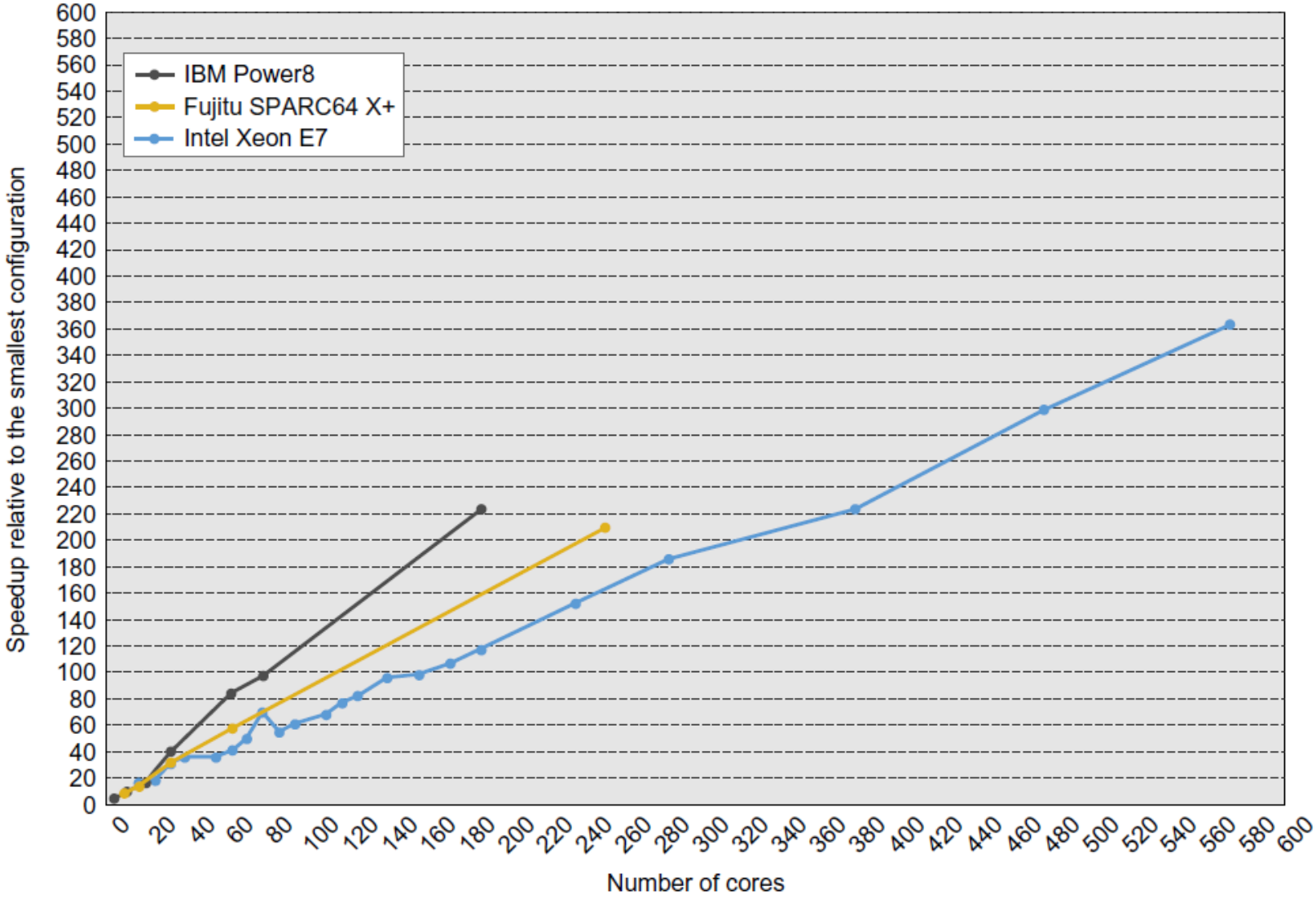
SPARC64 X+



X+ 4-chip Building Block



SPECintRate Speedup



Contents

Fallacies and Pitfalls

- P: Measuring performance of multiprocessors by linear speedup versus execution time
- F: Amdahl's Law doesn't apply to parallel computers
- F: Linear speedups are needed to make multiprocessors cost-effective
  - Doesn't consider cost of other system components
- P: Not developing the software to take advantage of, or optimize for, a multiprocessor architecture