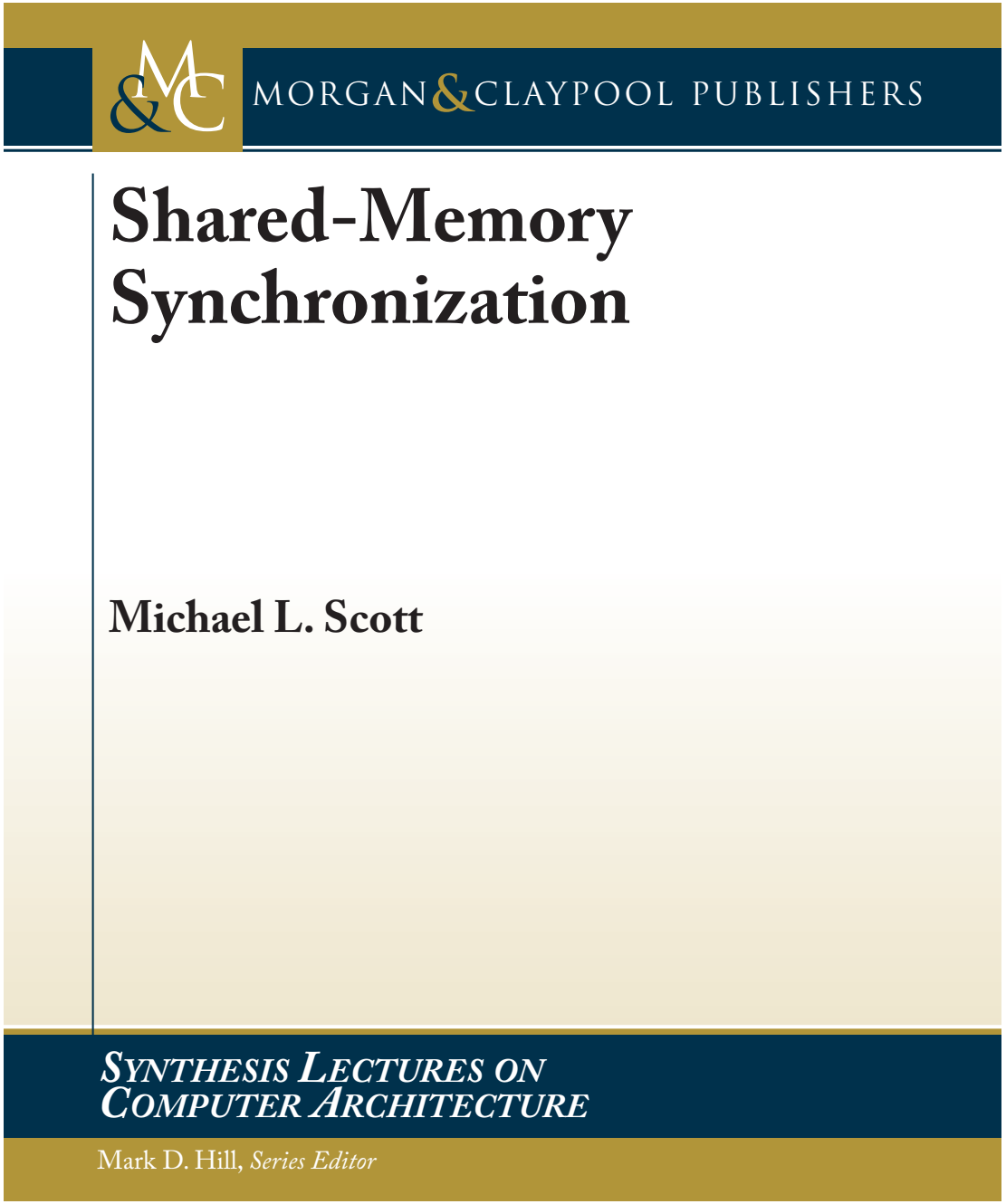
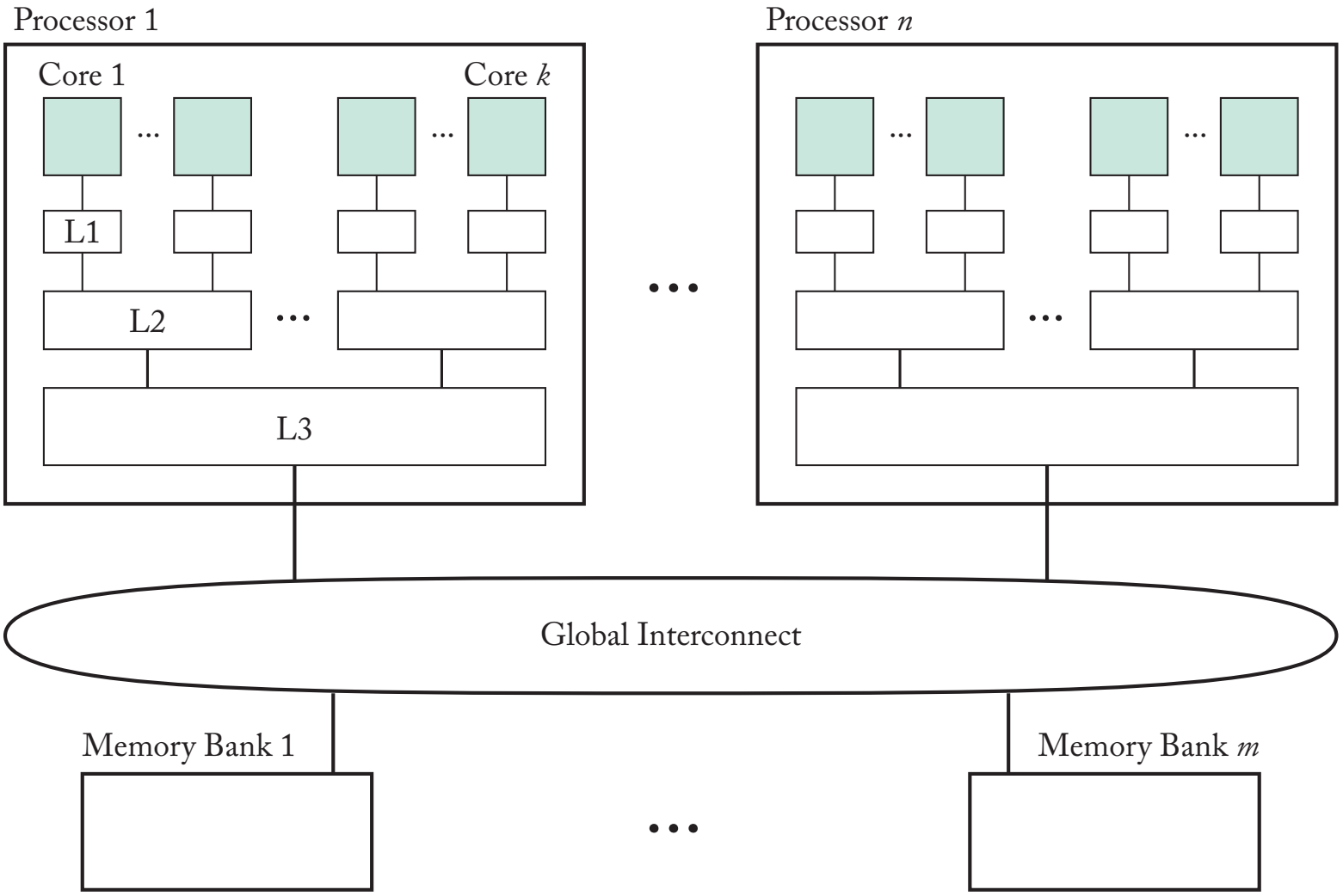
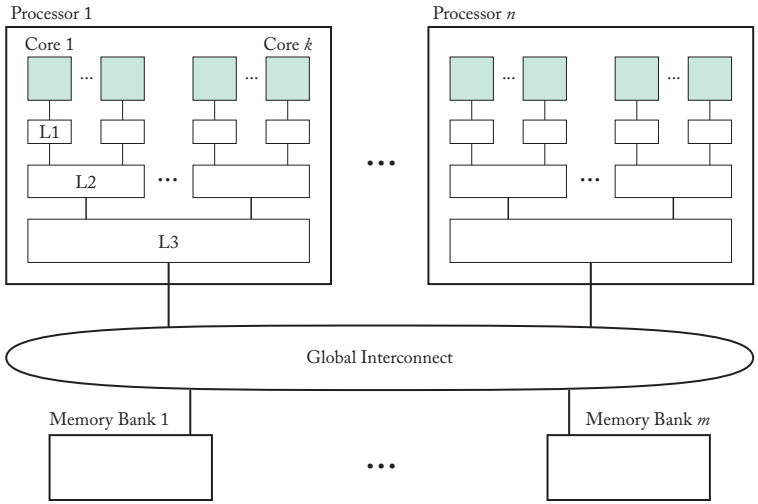
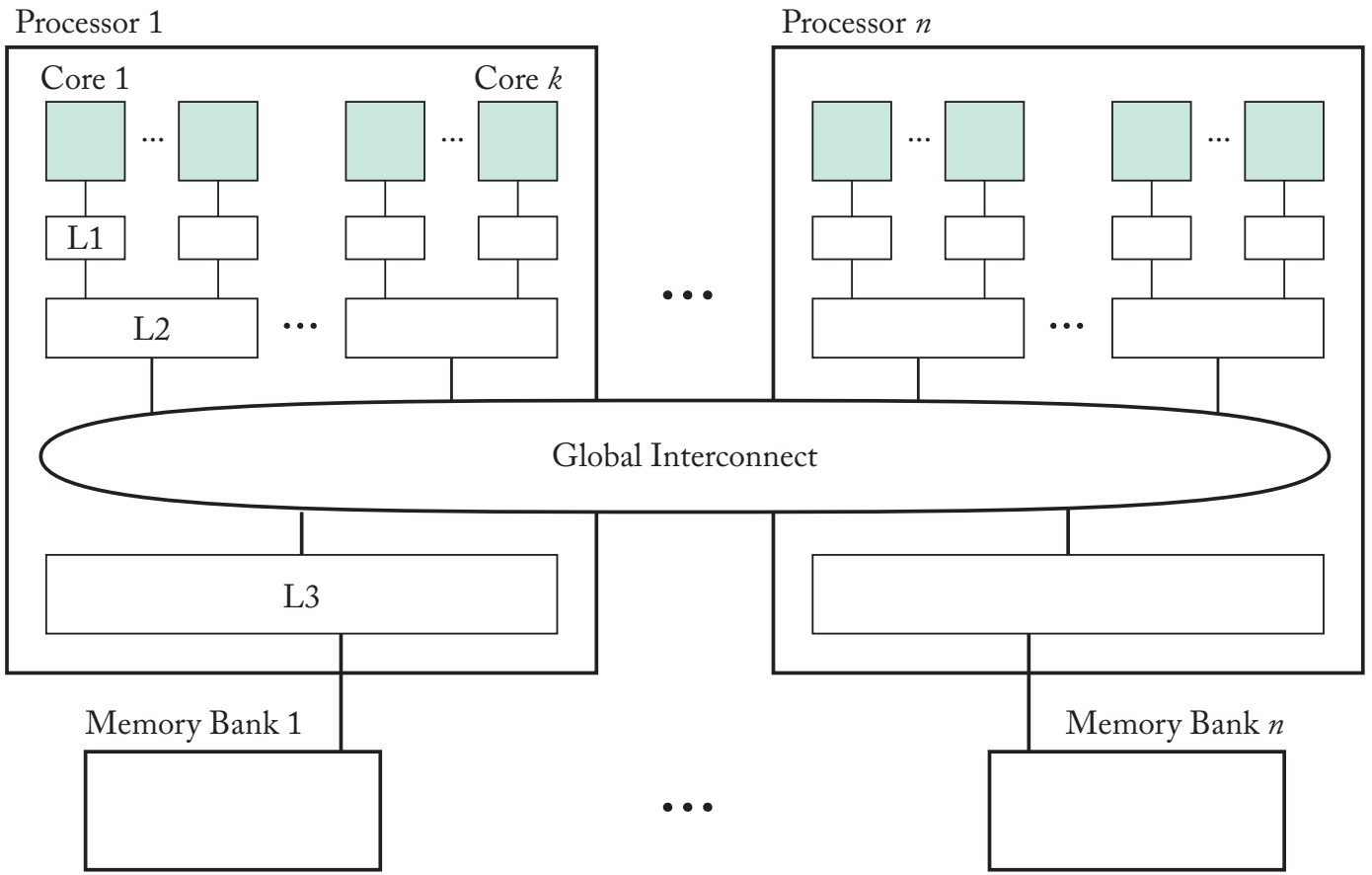


Chapters 2 (background) and 3.4 (memory models).

Hardware Basics


Processor 1
Core 1
Core k
L1
L2
L3
Processor n
Global Interconnect
Memory Bank 1
Memory Bank m


Processor 1
Core 1
Core k
L1
L2
L3
Processor n
Global Interconnect
Memory Bank 1
Memory Bank m


Processor 1
Core 1
Core k
L1
L2
L3
Processor n
Global Interconnect
Memory Bank 1
Memory Bank n

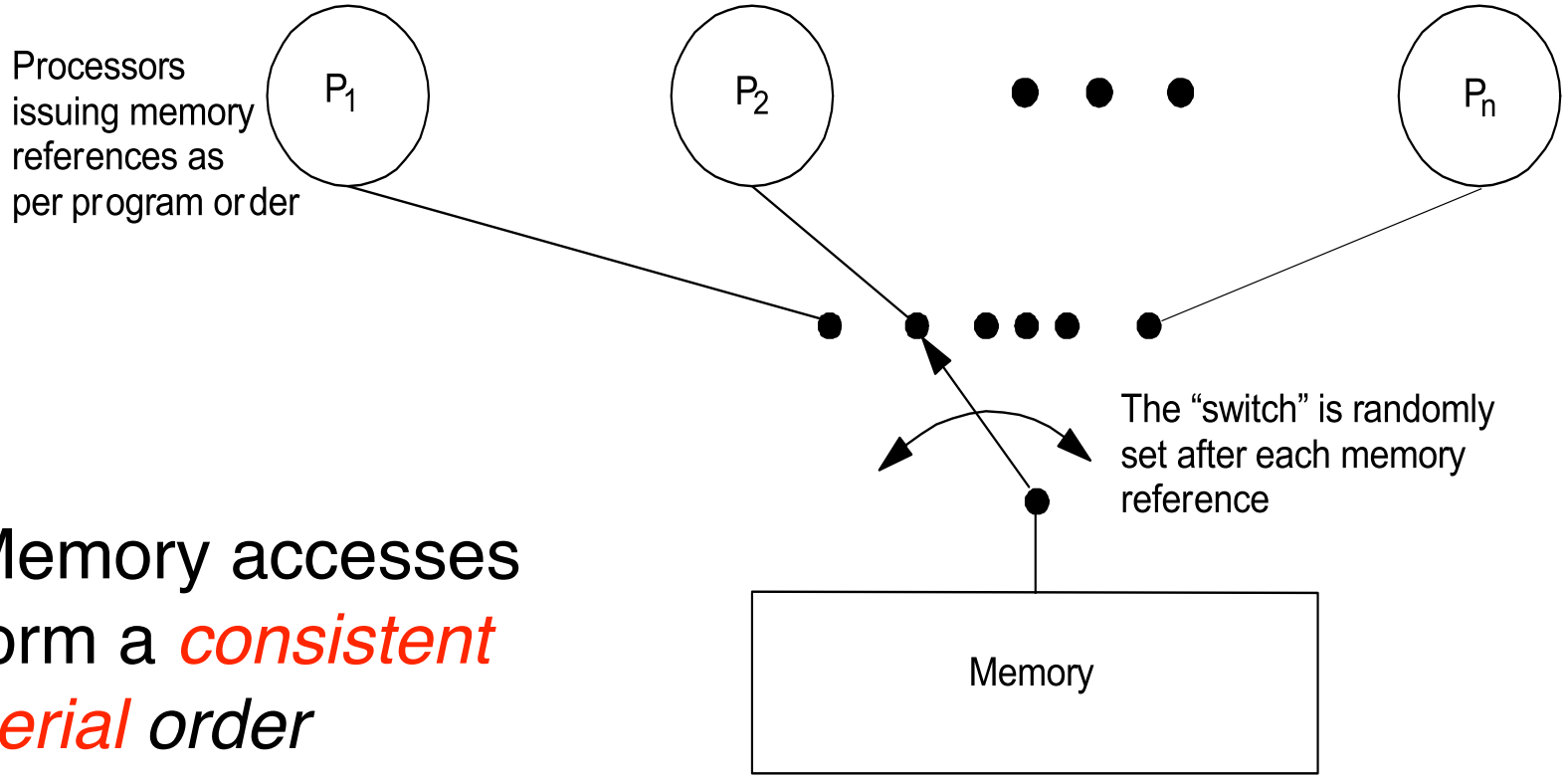
Cache Coherence

Rajeev Balasubramonian

Associate Professor
School of Computing
University of Utah

A memory system is coherent if:

- P writes to X; no other processor writes to X; P reads X and receives the value previously written by P
- P1 writes to X; no other processor writes to X; sufficient time elapses; P2 reads X and receives value written by P1
- Two writes to the same location by two processors are seen in the same order by all processors – write serialization

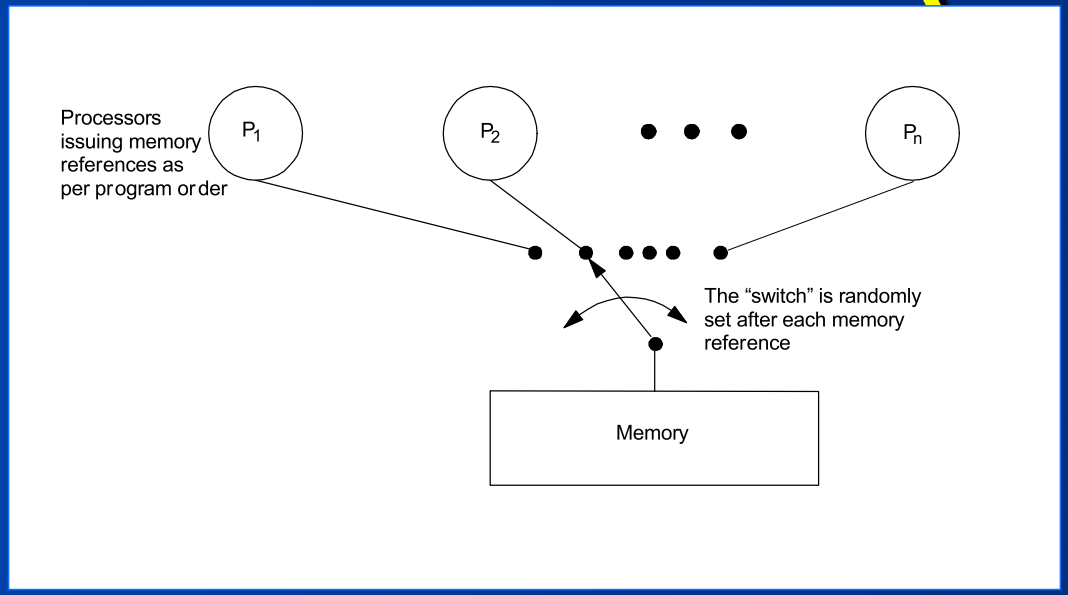


Memory accesses form a *consistent serial order*

Implications:

- Reads and writes are carried out *atomically*
- Memory accesses from each processor appear in the serial order in the order of the program

Sequential Consistency (SC)



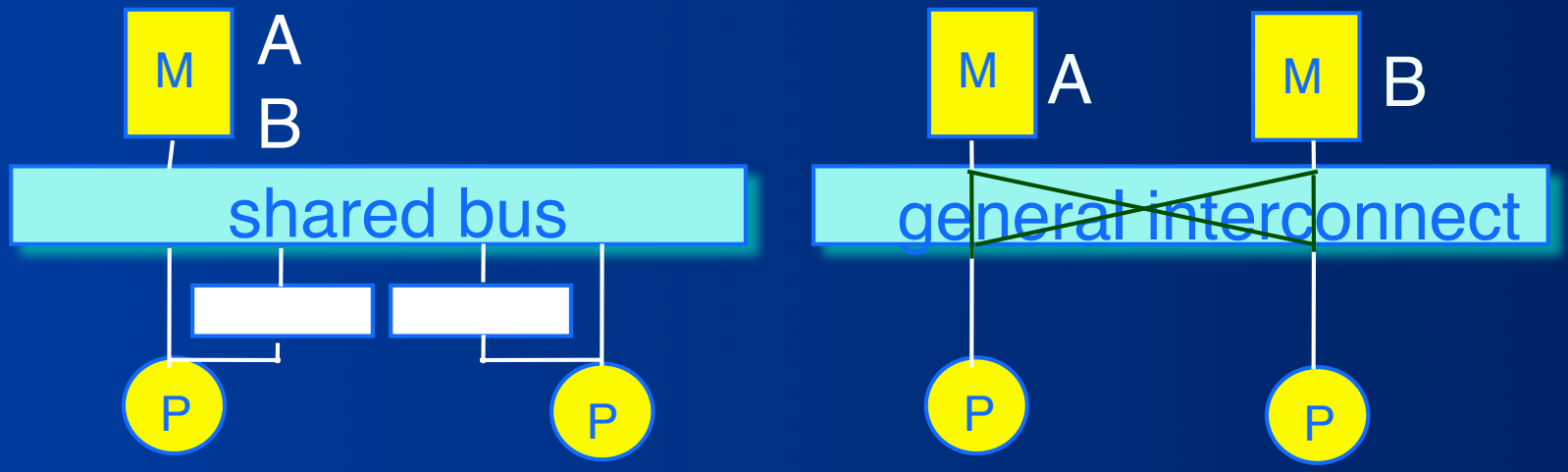
"the result of the execution is the same as if the operations of all processors are executed in some sequential order and the operations of each individual processor appear in the order specified by its program." (Lamport, 1979)

- *SC Execution*: An execution of a program is SC if the results it produces conform to any possible interleaving of program orders
- *SC System*: A system is SC if *all* possible executions on that system is an SC execution

Violations of SC: Examples

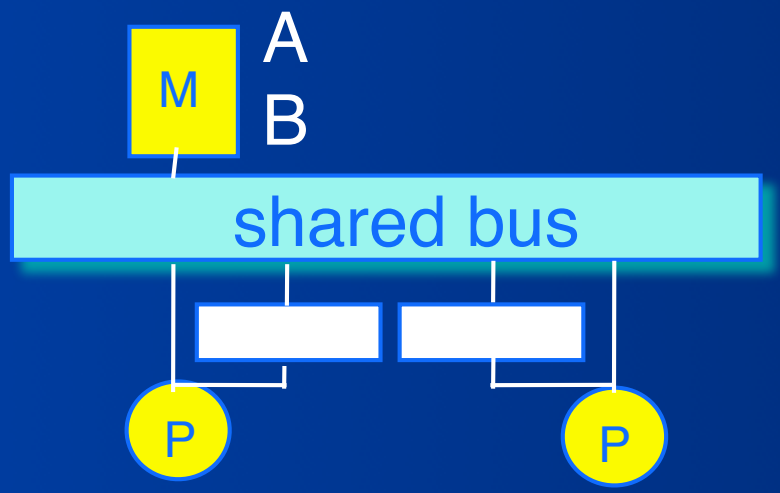
P0:	P1:
A:=1	B:=1
if B=0 then	if A=0 then
<critical section>	<critical section>

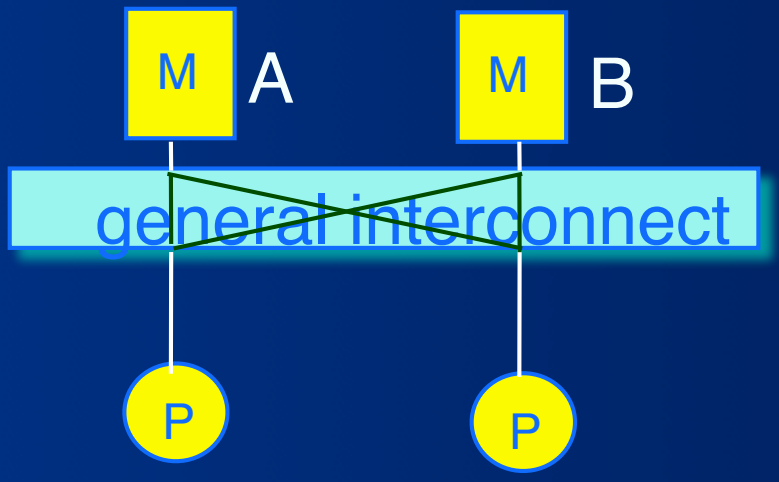
Two example systems:



Sequential consistency is preserved if memory operations are carried out in program order

Enforcing SC





Disallow bypassing in the write buffer

Do not issue a new request into the interconnect until the previous one is performed (acknowledged)

The Cache Coherence (CC) Problem

Most machines have multiple cached copies of data

The cache coherence problem: *how to maintain the illusion of a coherent shared address space*

Cache Coherence Protocols

- Directory-based: A single location (directory) keeps track of the sharing status of a block of memory

- Snooping: Every cache block is accompanied by the sharing status of that block – all cache controllers monitor the shared bus so they can update the sharing status of the block, if necessary

  - Write-invalidate: a processor gains exclusive access of a block before writing by invalidating all other copies
  - Write-update: when a processor writes, it updates other shared copies of that block

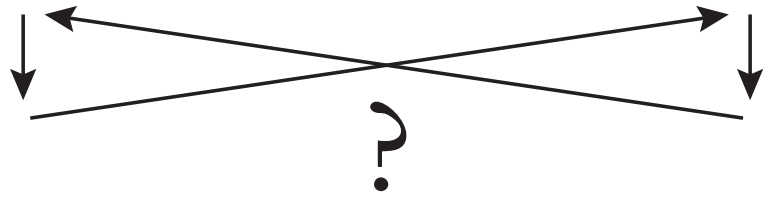
Ordering Loop 1

// initially x == y == 0

thread 1:

1: x := 1

2: i := y



thread 2:

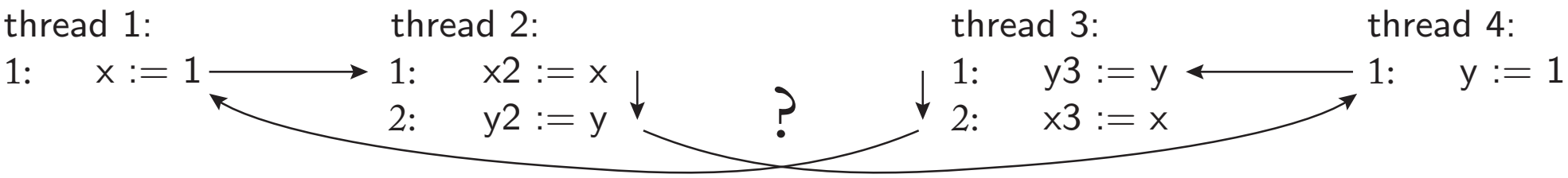
1: y := 1

2: j := x

// finally i == j == 0

Ordering Loop 2

// initially x == y == 0



// finally y2 == x3 == 0 and x2 == y3 == 1

Memory Reordering

- Software
  - compiler optimizations
- Processor
  - pipelining -- out-of-order completion
  - write buffers
  - out-of-order issue
- System
  - split-transaction buses
  - multiple buses
  - interleaved memory
- All of these obey single-threaded data dependences (only)

http://geekandpoke.typepad.com/.a/6a00d8341d3df553ef013485f6be4d970c-800wi