Shared Memory Systems

Haitao Wei

(Most of the slides are from Dr. Stephane Zuckerman)

University of Delaware

http://www.udel.edu

Computer Architecture and Parallel Systems Laboratory

http://www.capsl.udel.edu

UNIVERSITY OF DELAWARE

Outline

- Part 1: Basic Topics (2 sessions)
  - Memory Model of Shared Memory System
  - Cache
- Part 2: Advanced Topics (1 session)
  - Partitioned Global Address Space (PGAS)
  - Open Problems of Memory Models for Distributed

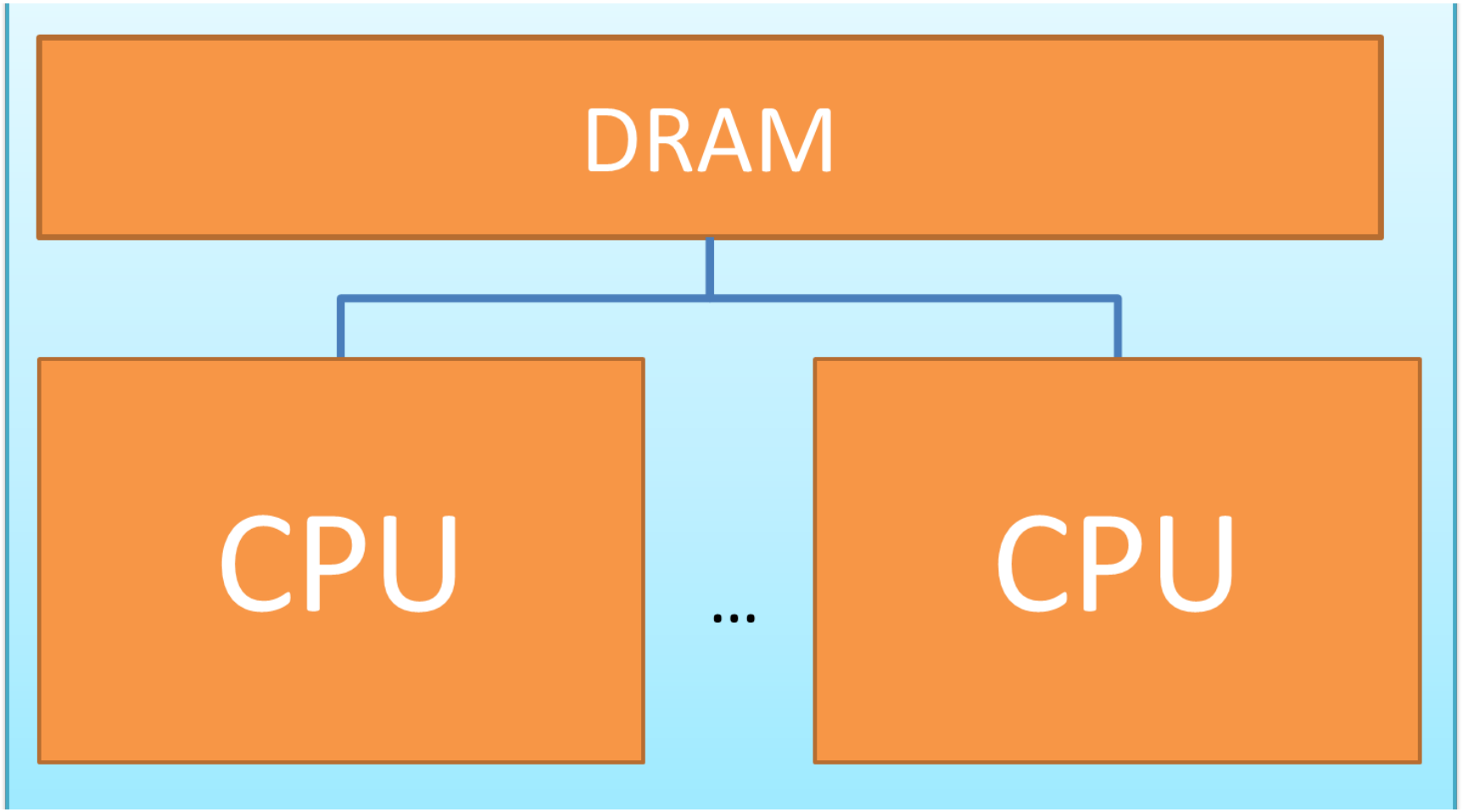
Reading List

- Hennessy and Patterson: Chapter 5 (optionally chapter 6)
- Culler and Singh: Chapter 5

Outline

- **Overview of Shared Memory Systems**
- Programming Execution Models
- Memory Consistency Models
  - A Motivating Example
  - Uniform Memory Consistency Models Strongest MCMs
    Weaker Uniform MCMs
  - Non-Uniform Memory Consistency Models Hardware-Oriented MCMs
    Software- and Programmer-Oriented MCMs
  - Conclusion On MCMs

A 100,000-Mile View



Advantages of Shared-Memory Systems

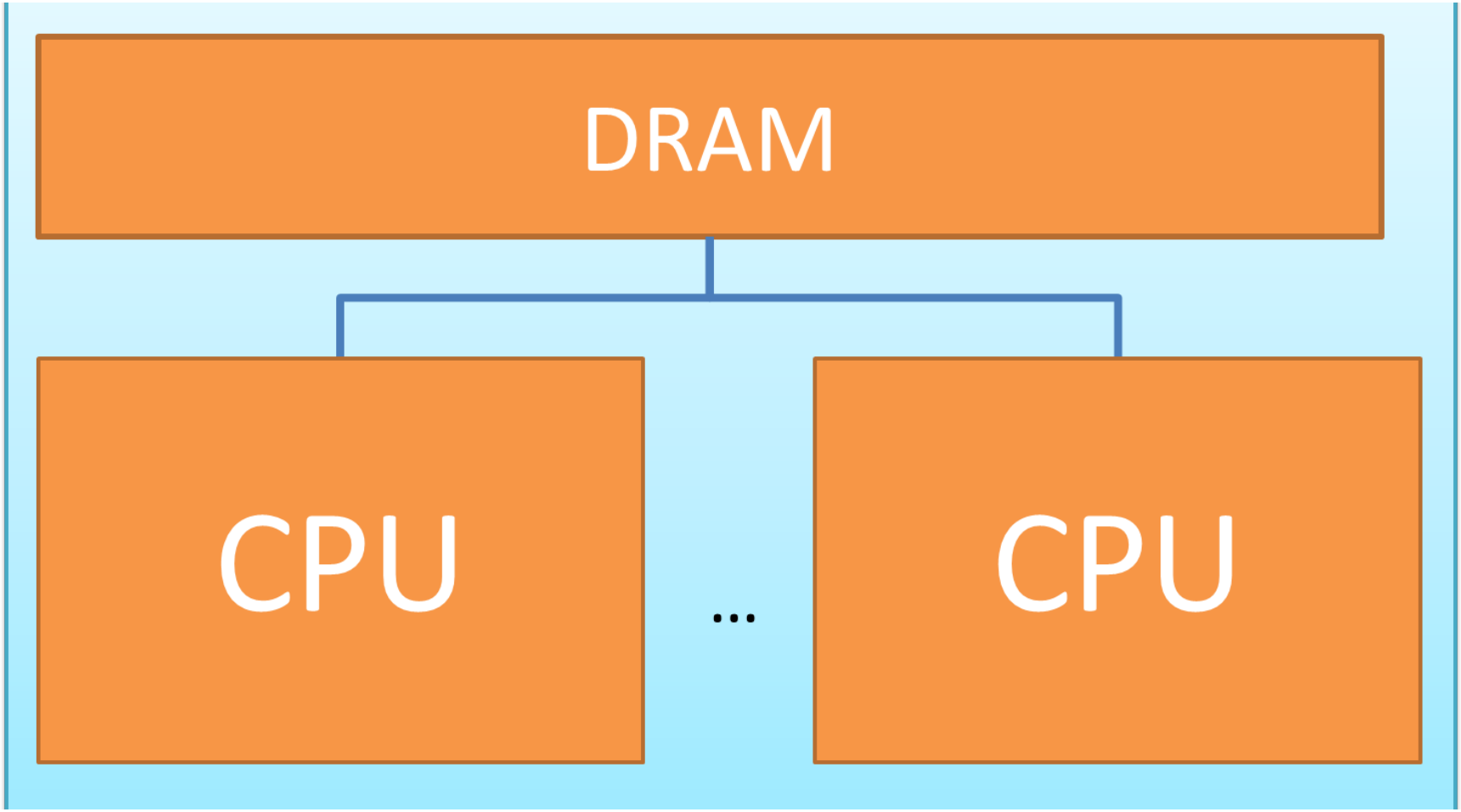
- No need to perform special operations to access memory locations
- State can be passed to multiple threads of execution implicitly
- Reduced overhead when read from/writing to memory

Shortcomings of Shared-Memory Systems

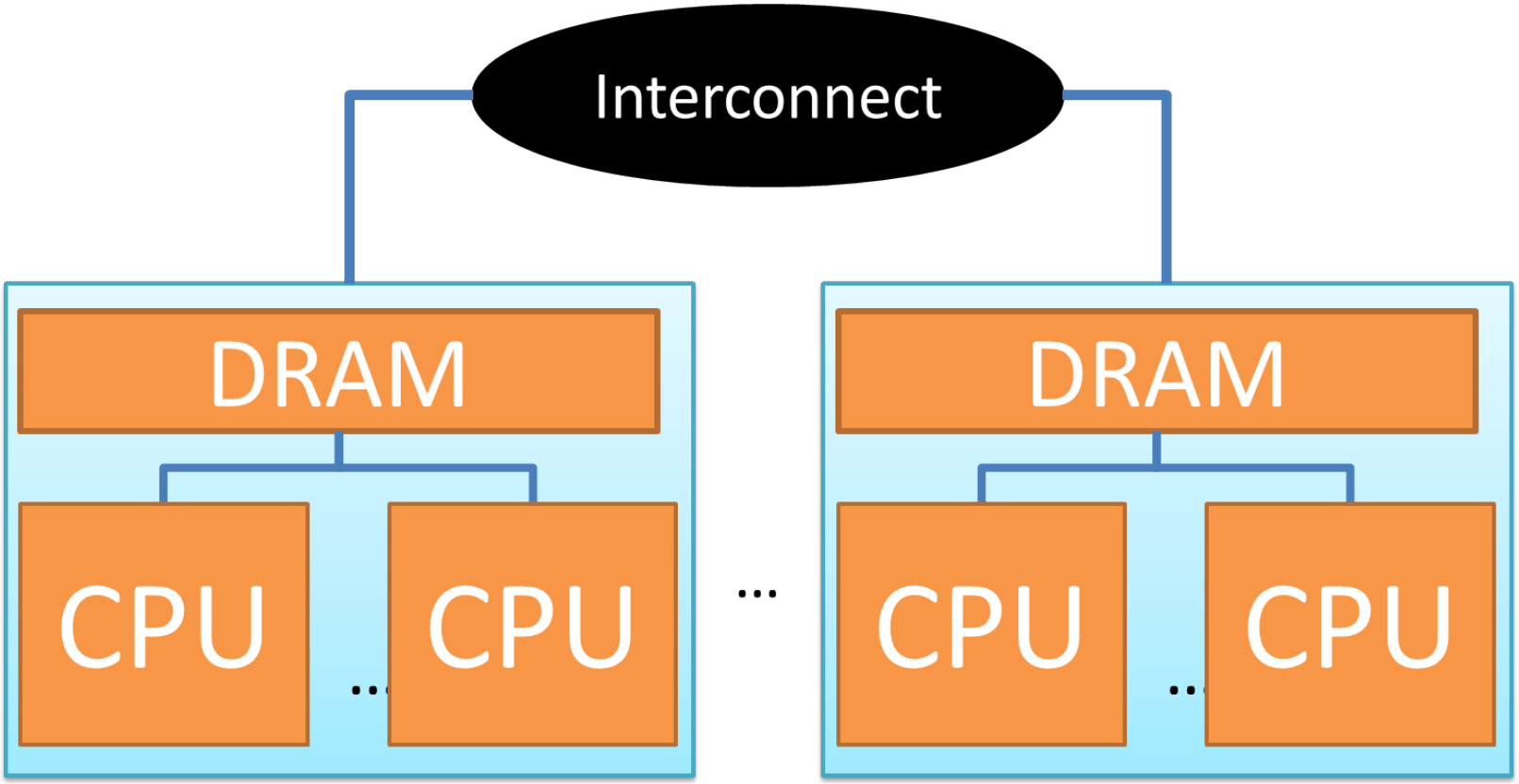
LOTS!

. . . But we will talk about them later.

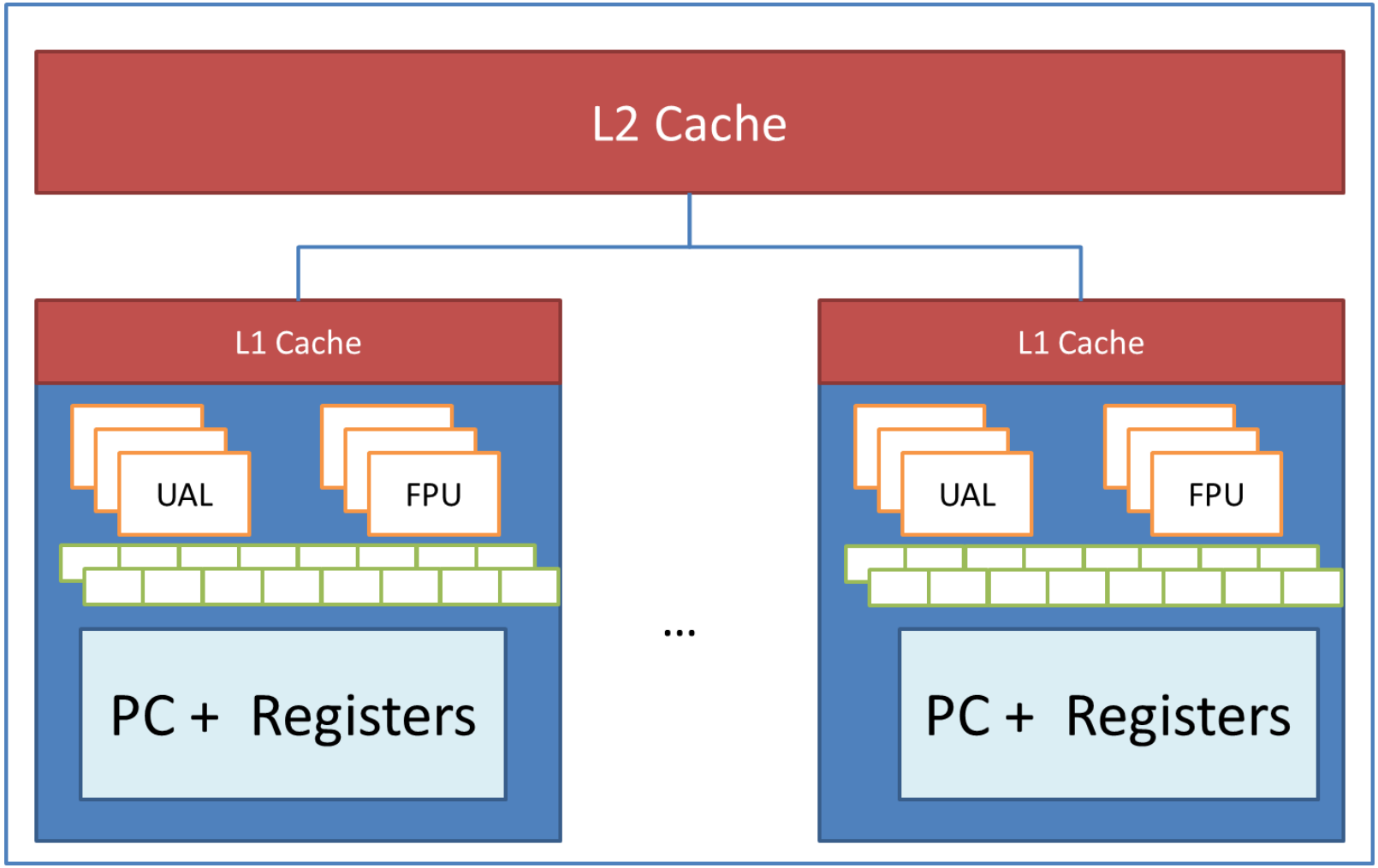
Symmetric MultiProcessor Systems



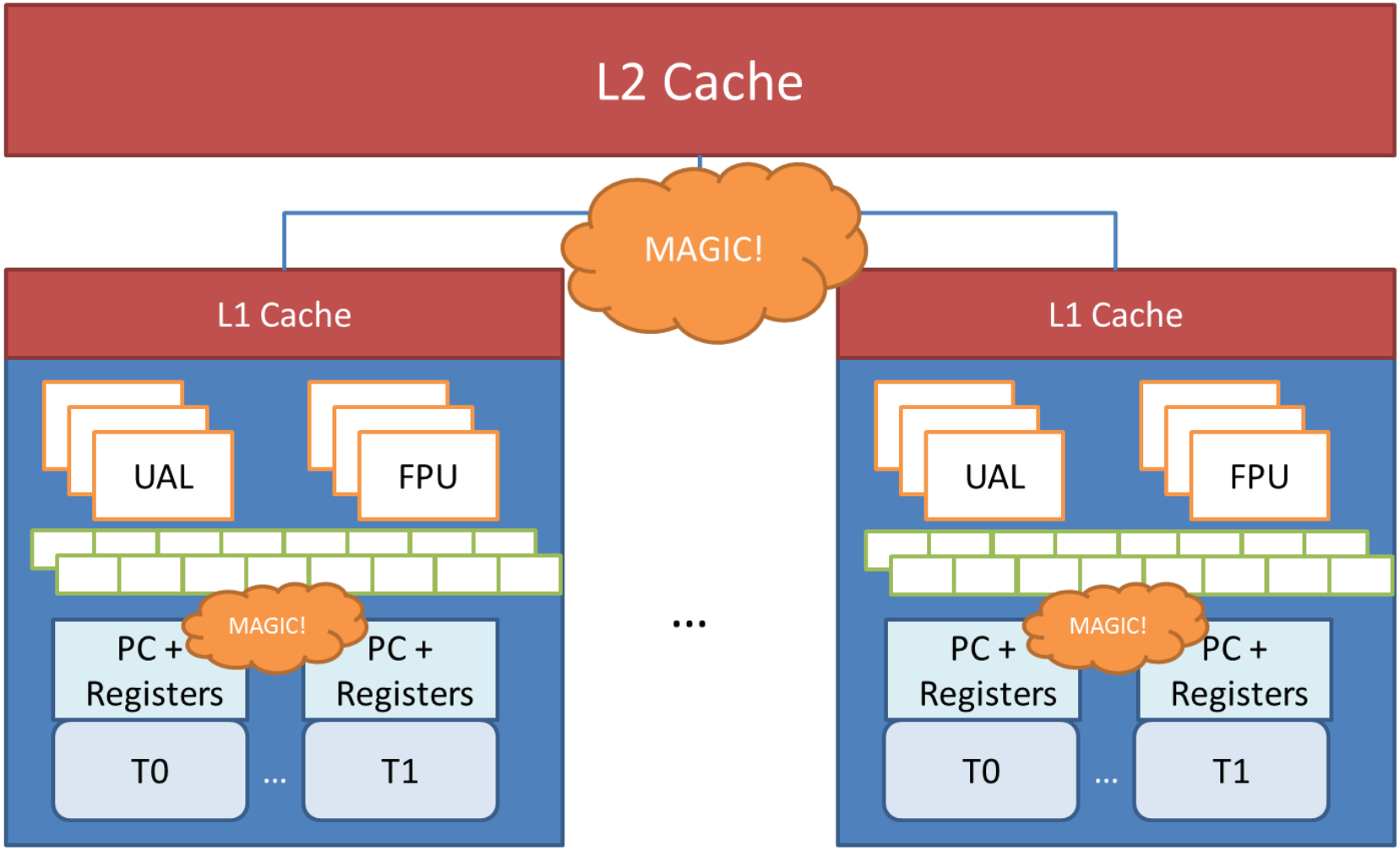
Distributed Shared Memory Systems



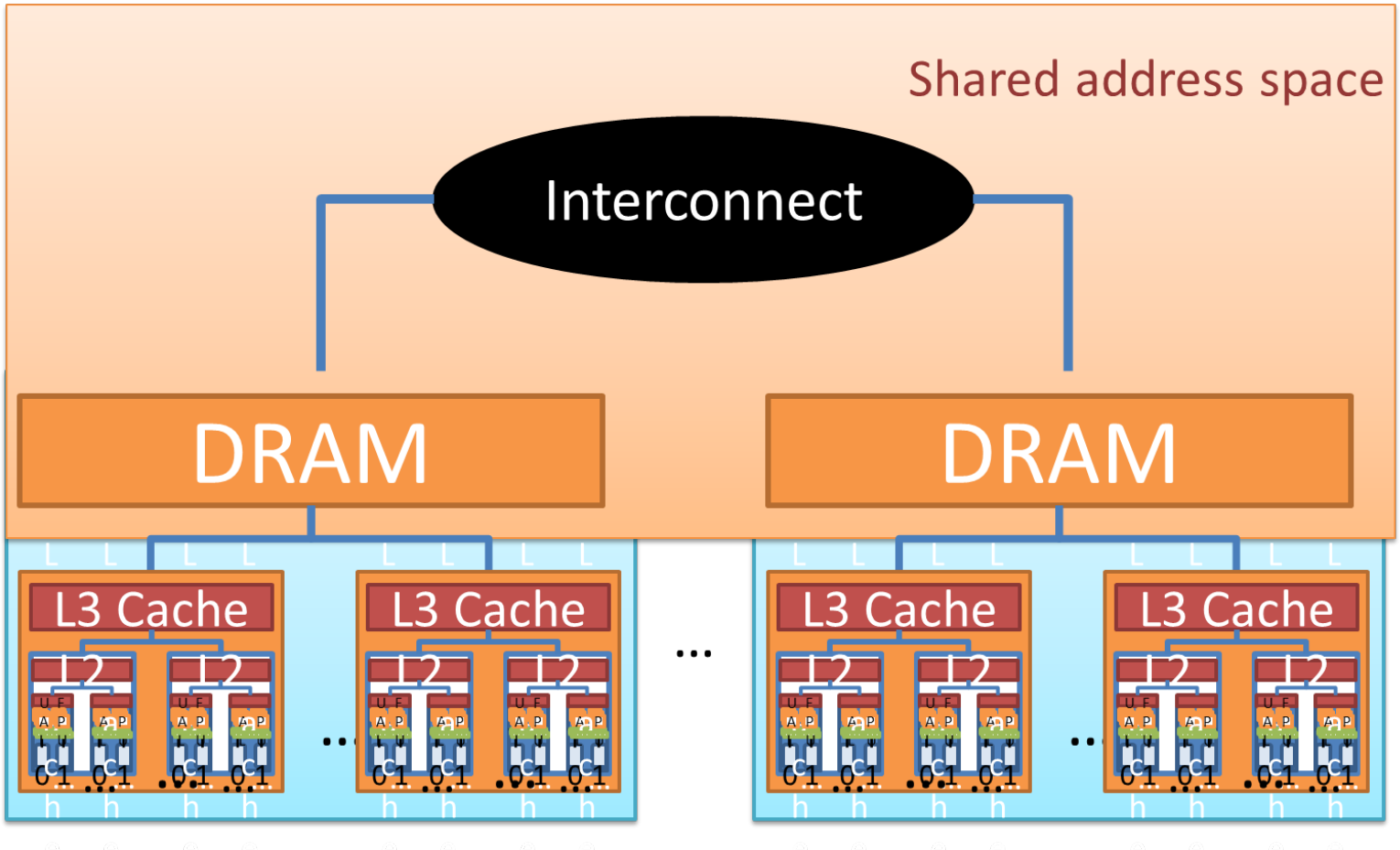
Why This Is More Complicated Than It Appears



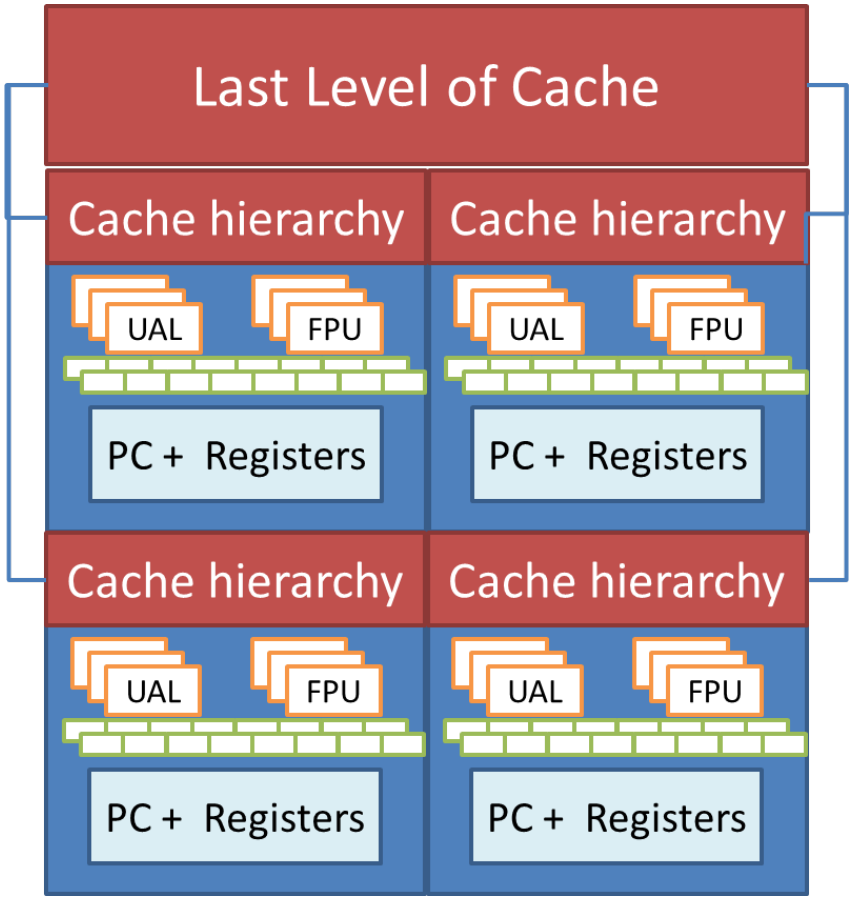
Why This Is More Complicated Than It Appears (cont'd)



Putting It All Together (1)



The Advent of Chip Multi-Processors (CMP)



Hardware Implementations of Threads

- Types of Multithreading
  - Fine-grain
    - Switch between threads on each instruction
    - Interleaved thread execution
    - Hide throughput losses in both short and long stalls
    - Slowdown execution of single-threaded applications
  - Coarse-grain
    - Switch threads only when a high-latency or stall is found
    - Limited ability to overcome throughput losses
      - Pipeline startup
- Simutaneous MultiThreading (SMT)
  - Better utilization of resources
  - Normal multithreading: pipeline is “locked” by the thread

Comparison of Multithreading Hardware

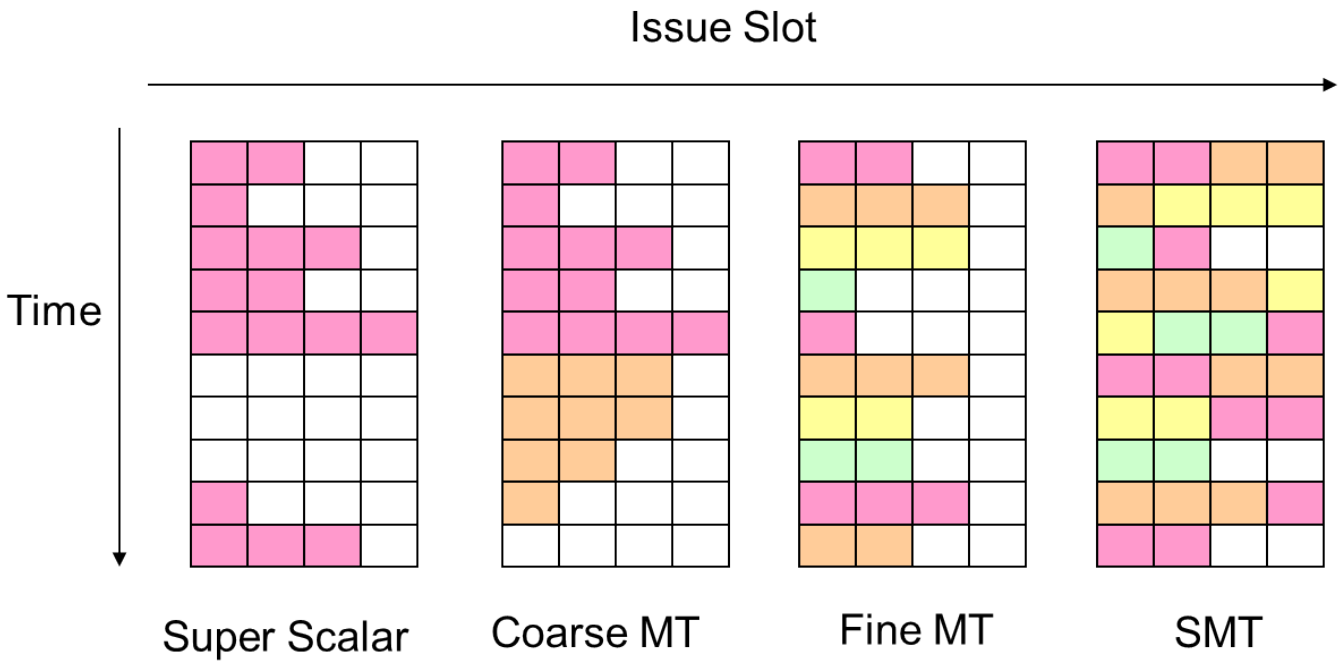
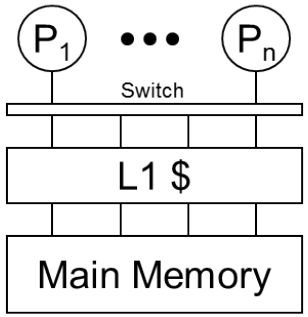
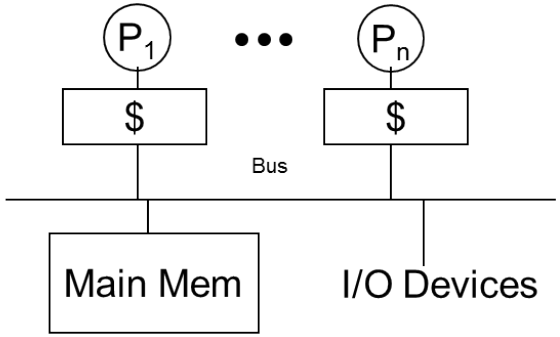


Figure : From Hennessy and Patterson

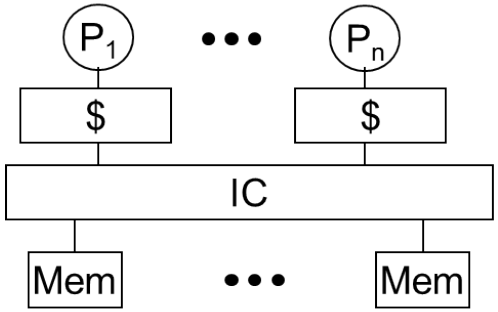
Putting It All Together



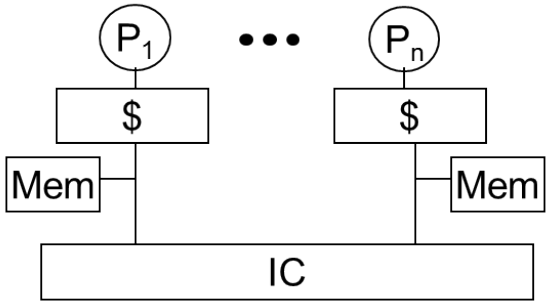
Shared Cache



Bus Based Shared Memory



Dance Hall



Distributed Memory

Outline

- Overview of Shared Memory Systems
- **Programming Execution Models**
- Memory Consistency Models
  - A Motivating Example
  - Uniform Memory Consistency Models Strongest MCMs
    Weaker Uniform MCMs
  - Non-Uniform Memory Consistency Models Hardware-Oriented MCMs
    Software- and Programmer-Oriented MCMs
  - Conclusion On MCMs

Programming Execution Models

PXMs define a set of rules to describe how programs run:

- Message Passing Model
  - De facto multi-computer programming model
  - Multiple address spaces
  - Explicit communications / implicit synchronization

- Shared Memory Model
  - De facto multi-processor programming model
  - Single address space
  - Implicit communications / explicit synchronization

Distributed Memory MIMD

Advantages:

- Less contention
- Highly scalable
- Simplified synchronization
  ⇒ sync+comms

Disadvantages:

- Load balancing
- Deadlock / Livelock prone
- Waste of bandwidth
- Overhead of small messages

Shared Memory MIMD

Advantages:

- No Partitioning
- No (explicit) data movement
- Minor modifications (or not at all) of toolchains and compilers

Disadvantages:

- Synchronization
- Scalability
  - High-throughput, low-latency network
  - Memory hierarchies
  - Distributed shared-memory (DSM)

Shared-Memory Execution Models

Shared-memory PXMs are usually described according to three criteria:

- The threading (or task) model
  ⇒ How do I create parallel work to speed-up my computation?
- The memory model

  ⇒ In which order are loads and stores seen by all threads?
- The synchronization model
  ⇒ How is memory ordering enforced when needed?

The Challenges of Shared-Memory

- Shared-memory multiprocessors
  ⇒ Effective at a number of thousand units
- How to optimize and compile parallel applications
- Main areas: assumptions about

  Coherence
  Memory consistency

Outline

- Overview of Shared Memory Systems
- Programming Execution Models
- **Memory Consistency Models**
  - A Motivating Example
  - Uniform Memory Consistency Models Strongest MCMs
    Weaker Uniform MCMs
  - Non-Uniform Memory Consistency Models Hardware-Oriented MCMs
    Software- and Programmer-Oriented MCMs
  - Conclusion On MCMs

A Motivating Example

$x \leftarrow 1$

$r1 \leftarrow y$

A Motivating Example

$y \leftarrow 1$

$r2 \leftarrow x$

A Motivating Example

Thread0	*Thread1*
$x \leftarrow 1$	$y \leftarrow 1$
$r1 \leftarrow y$	$r2 \leftarrow x$

Table: Initially, x =y =0. Is it possible to have r1=r2=0?

What Memory Consistency is All About

Q What happens when at least two concurrent memory operations arrive at the same memory location *x*?

→ What happens when a data-race (i.e. at least one of the two memory operations is a write) occurs at some memory location *x*?

- Memory Consistency Models try to answer that question.

The Answer of the Message Passing Crowd

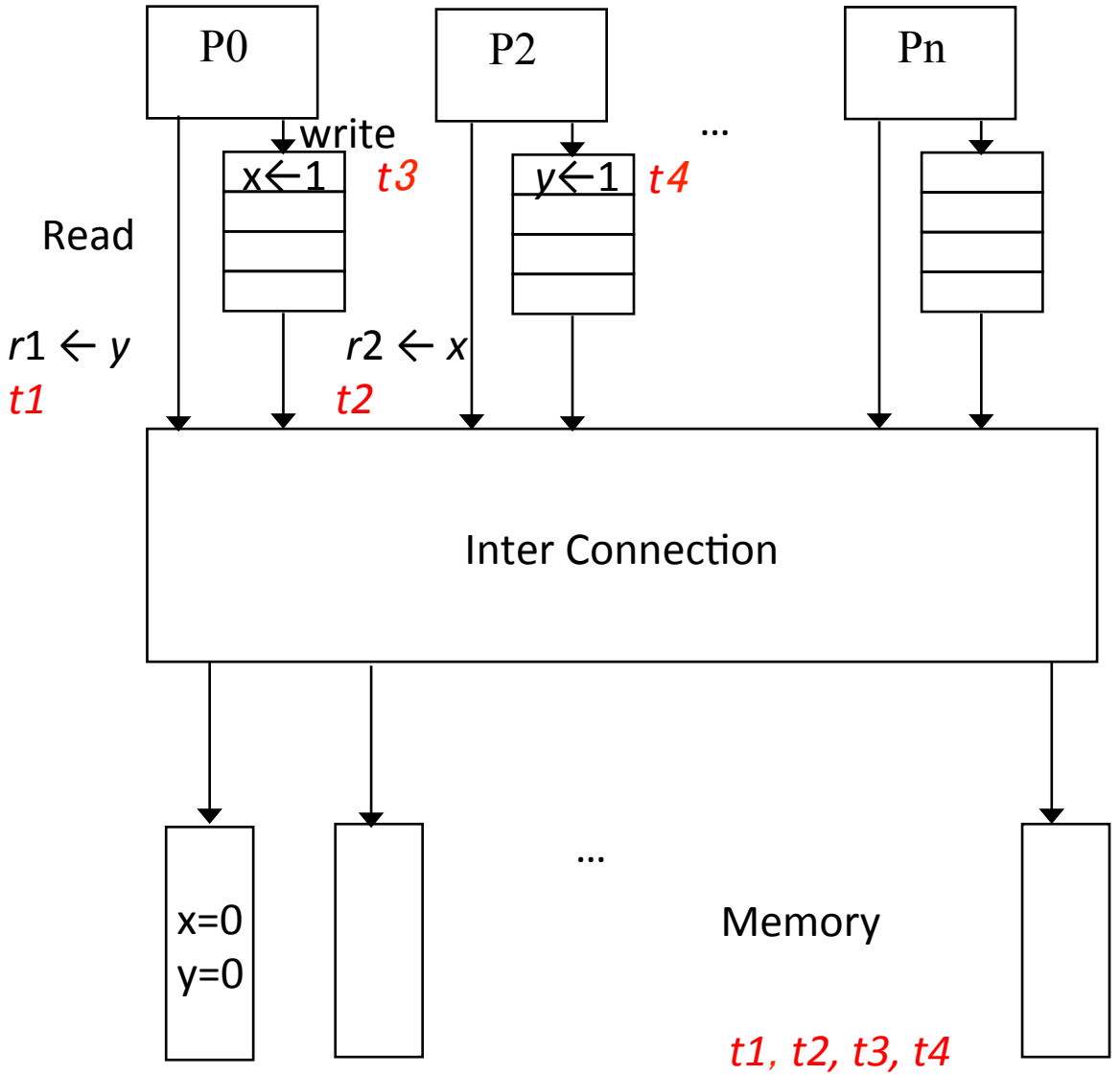
It can never happen: data is explicitly sent and received. This answer is fine, but. . .

- We do not live in a pure message-passing world
  - Memory is shared on most super-computers, e.g.:
    - Efficient MPI runtime systems make the distinction between intra-node and inter-node communications
    - Inter-node communications work as advertised, but...
    - *Efficient* intra-node communications make the use of shared-memory segments, i.e. shared memory

Issue order and perform order

- Issue order (program order if there is on out of order opt.): the operation order that is issued by the processor
- Perform order: the operation order that is happened (executed) to the memory

Back to our Example



Thread0	*Thread1*
x←1	*y*←1
*r*1 ← *y*	*r*2 ← *x*

r1=r2=0

Outline

- Overview of Shared Memory Systems
- Programming Execution Models
- **Memory Consistency Models**
  - A Motivating Example
  - Uniform Memory Consistency Models Strongest MCMs
    Weaker Uniform MCMs
  - Non-Uniform Memory Consistency Models Hardware-Oriented MCMs
    Software- and Programmer-Oriented MCMs
  - Conclusion On MCMs

Atomic Consistency [Lamport(1986)]

A system is AC if

- All memory operations are *issued* and *performed* in some total order

  → Real time constraint: time slots are allocated, and mem ops must be performed according to them.

- Memory operations must follow program order
- Strongest MCM that was conceived

  → Never implemented

Sequential Consistency [Lamport(1978)]

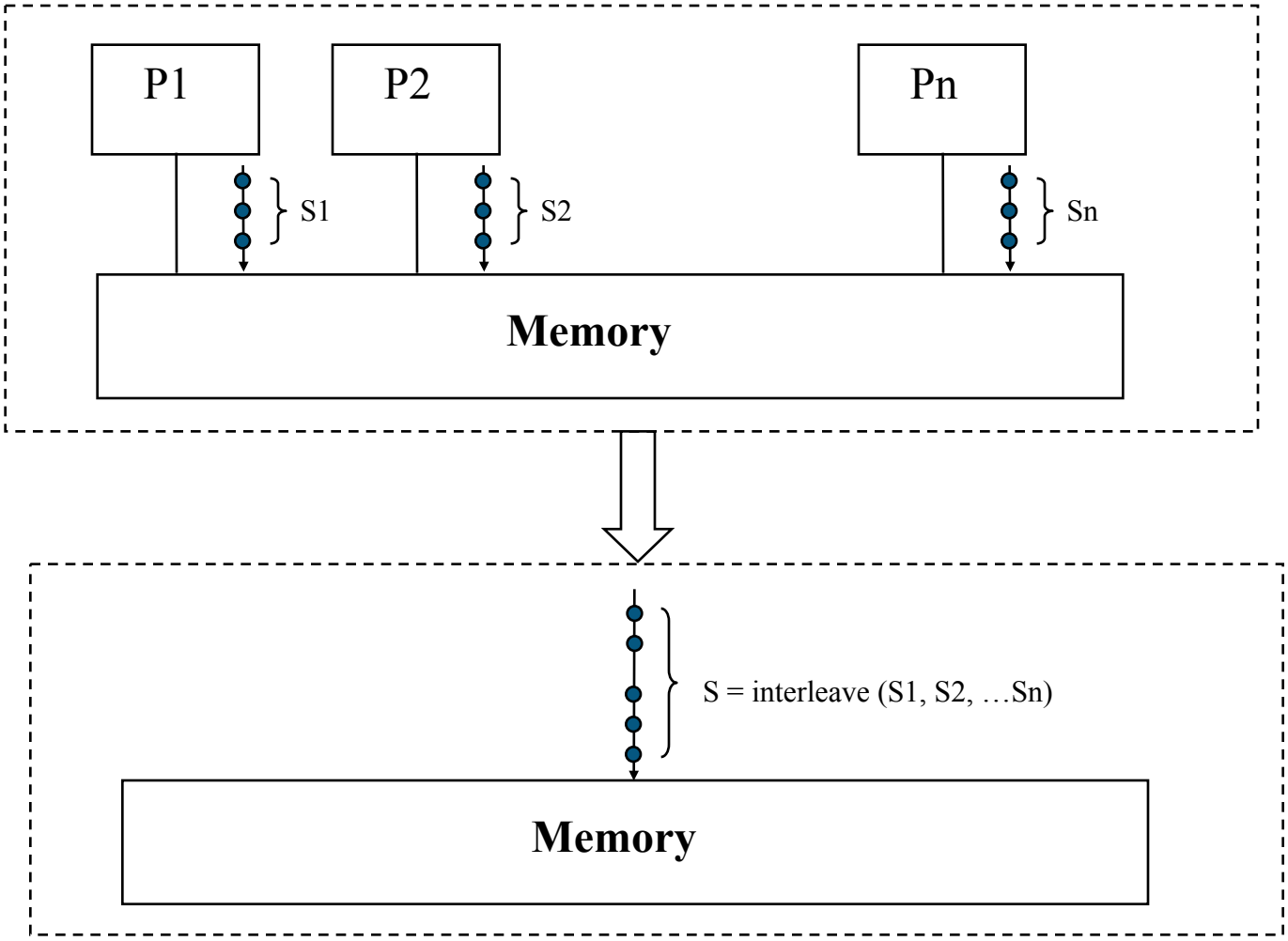
A system is SC if

- All memory operations *appear* to follow some total order
- Memory operations (*appear to*) follow program order

Definition: Sequential Consistency

A system is sequentially consistent if

. . . the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.



Sequential Consistency Model

Back to our Example

Thread0	*Thread1*
$x \leftarrow 1$	$y \leftarrow 1$
$r1 \leftarrow y$	$r2 \leftarrow x$

Table: Initially, x =y =0.

Is it possible to have r1 = r2 = 0 ?

NO → There is no total linear order which allows both Thread 0 and Thread 1 to see memory operations happening in the same order such that r1 = r2 = 0

The Drawbacks of Sequential Consistency

It offers strong guarantees: a modification to memory *must* be seen by all other threads in a given program

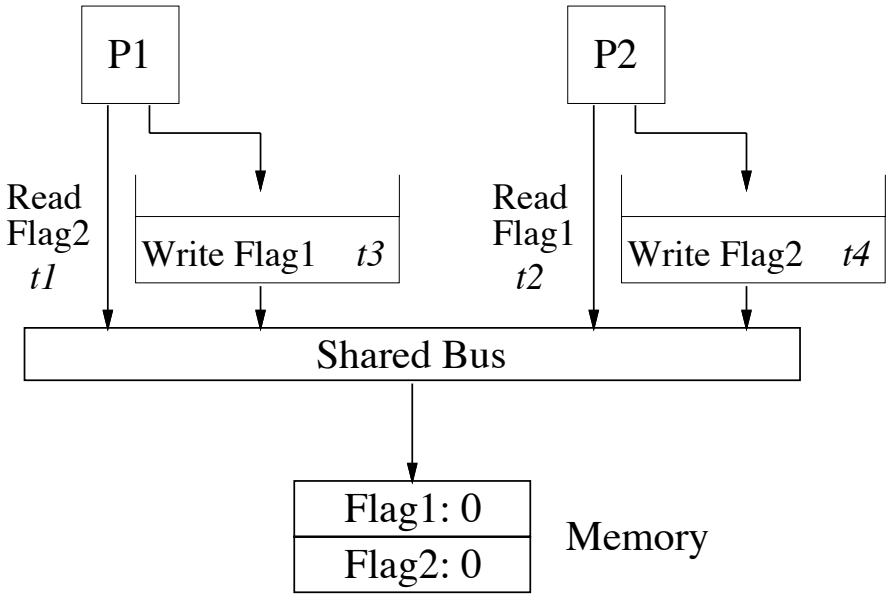
−→ How complicated is it to implement such a system in hardware ?

→ What about caches? Write buffers? etc.

−→ How scalable is it ?

−→ How expensive is it to implement that kind of consistency model?

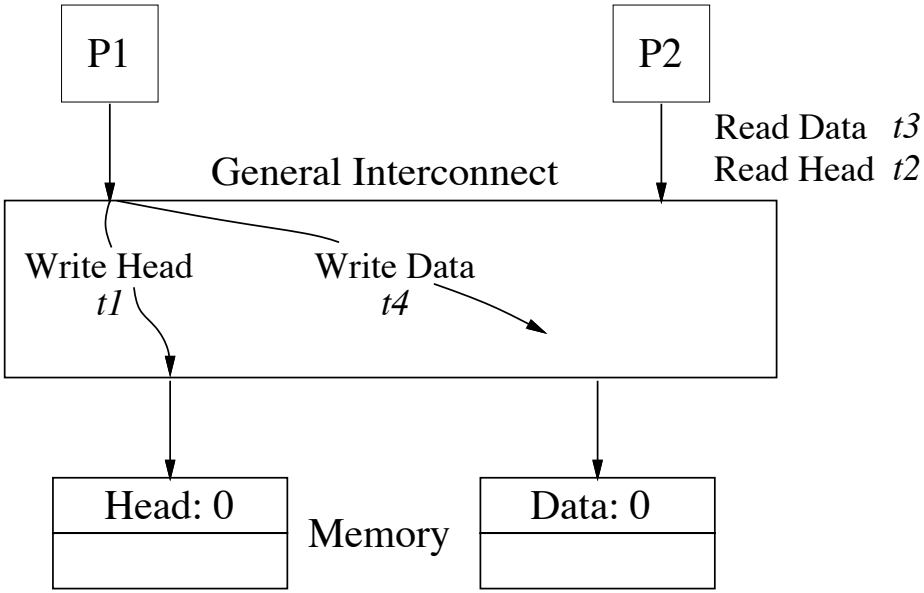
Write Buffer Breaks SC



P1	P2
Flag1 = 1	Flag2 = 1
if (Flag2 == 0)	if (Flag1 == 0)
critical section	*critical section*

From [Sarita V. Adve and Kourosh Gharachorloo paper 1995: Shared Memory Consistency Models: A Tutorial]
http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-95-7.pdf

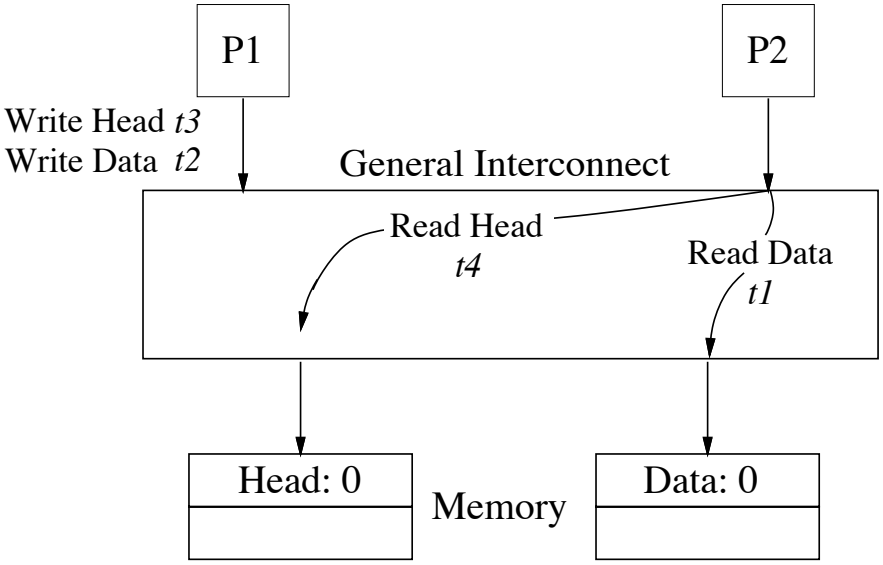
Overlapped Writes Breaks SC



P1	P2
Data = 2000	while (Head == 0) {;}
Head = 1	... = Data

From [Sarita V. Adve and Kourosh Gharachorloo paper 1995: Shared Memory Consistency Models: A Tutorial]
http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-95-7.pdf

Non-Blocking Reads Breaks SC



P1	P2
Data = 2000	while (Head == 0) {;}
Head = 1	... = Data

From [Sarita V. Adve and Kourosh Gharachorloo paper 1995: Shared Memory Consistency Models: A Tutorial]
http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-95-7.pdf

Coherence (Cache Consistency) [Gharachorloo et al.(1990)]

Coherence is achieved if

- for each memory location *x*, there is a total order of all the memory operations dealing with *x*
- Memory operations on *x* follow the program order

Is our First Example Coherent?

Thread0	*Thread1*
$x \leftarrow 1$	$y \leftarrow 1$
$r1 \leftarrow y$	$r2 \leftarrow x$

Table: Initially, x =y =0.

Is it possible to have r1 = r2 = 0 ?

YES!

$\Rightarrow r1 \leftarrow y, y \leftarrow 1, r2 \leftarrow x, x \leftarrow 1$

The Difference with Previous Models

- Previous models tried to define an order for memory operations, regardless of their role in a program whatsoever
- Non-uniform MCMs make a difference between *synchronizing* memory operations and *ordinary* ones

Weak Consistency [Dubois et al.(1986)Dubois, Scheurich, Weak Ordering [Adve and Hill()]]

A system is WC/WO if

- all *synchronizing* accesses have performed before any *ordinary* access (load or store) is allowed to perform, and
- all *ordinary* accesses (load or store) have performed before any *synchronizing* access is allowed to perform
- *synchronizing* accesses are SC

Release Consistency [Gharachorloo et al.(1990)]

RC refines synchronizing accesses into two types: *acquire* and *release*. They are used to label instructions (Gharachorloo speaks about *properly labeled* programs). A system is RC if:

- *acquire* accesses must have performed before any ordinary operation is performed
- all ordinary memory operations have performed before an *release* operation is performed
- Synchronizing accesses (*acquire* or *release*) are SC

The Critical Section Example

Thread 0

```
while(true){
  a=…
  while(lock==0)[acquire]
  {x+=1;
  a=x+a;}
  lock=0[Release]
  read a;
}
```

Thread 1

```
while(true){
  b=…
  while(lock==1)[acquire]
  {x+=2;
  b=x+b;}
  lock=1[Release]
  read b;
}
```

More Examples (See [Adve et al.(1999)]

Thread 0	Thread 1
Data1 = 64	while(Flag != 1) ;
Data2 = 55	reg1 = Data1
Flag = 1	reg2 = Data2

Table : Ex1: What are the legal values in SC? WC? RC?

Solution

SC: reg1 = 64;reg2 = 55

WC,RC: reg1 = 64 or 0; reg2 = 55 or 0

More Examples (See [Adve and Gharachorloo(1996)])

Thread 0	Thread 1
Flag1 = 1	Flag2 = 1
reg1 = Flag2	reg2 = Flag1
if reg1 == 0	if reg2 == 0
critical section	*critical section*

Table : Ex2: What are the legal values in SC? WC? RC?

Solution

SC: Both reg1 and reg2 cannot be 0 (at the same time)

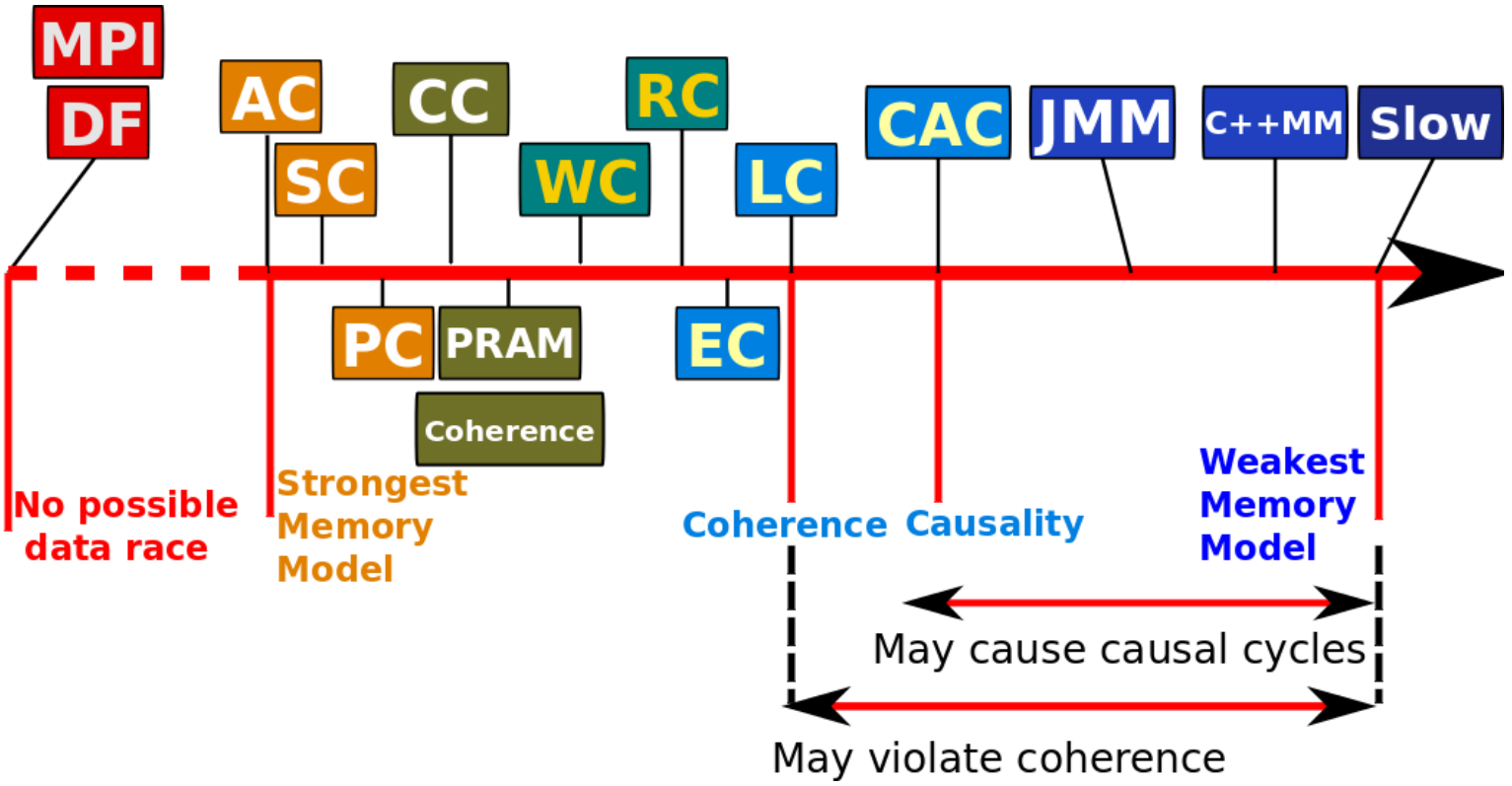
WC,RC: reg1 = 0 or 1;reg2 = 0 or 1

The C++ Memory Model

Very easy to understand:

- Synchronizing accesses (through the atomic keyword) are SC
- any incorrectly synchronized behavior implies an *undefined behavior*,

A Brief Recap



What to take home

A memory consistency model defines which memory operations are allowed, in which order

It concerns both hardware and software points of view

The weaker the MCM,

- the more optimizations can be performed
- the more scalable it is
- the heavier it is on a programmer's shoulders

The MCMs I Did Not Talk About

- SPARC processors' memory consistency models:
  - Total Store Order (TSO)
  - Partial Store Order (PSO)
- Location Consistency [Gao and Sarkar(2000)]
- Others(Localconsistency,...)

If You Want to Know More. . .

- S.Adve, K.Gharachorloo: *Shared Memory Consistency Models: a Tutorial* [Adve and Gharachorloo(1996)]
- D.Mosberger: *Memory Consistency Models* [Mosberger(1993)]
- J.Hennessy and D.Patterson“ *Computer Architecture: A Quantitative Approach*

Bibliography I

- S. Adve and K. Gharachorloo. Shared memory consistency models: a tutorial. *Computer*, 29(12):66 –76, Dec. 1996. ISSN 0018-9162. doi: 10.1109/2.546611.
- S. Adve, V. Pai, and P. Ranganathan. Recent advances in memory consistency models for hardware shared memory systems. *Proceedings of the IEEE*, 87(3):445 –455, Mar. 1999. ISSN 0018-9219. doi: 10.1109/5.747865.
- S. V. Adve and M. D. Hill. Weak ordering—a new definition. pages 2–14.
- M. Dubois, C. Scheurich, and F. Briggs. Memory access buffering in multiprocessors. In *Proceedings of the 13th Annual International Symposium on Computer Architecture*, pages 434–442, Tokyo, Japan, June 1986.

Bibliography II

- G. R. Gao and V. Sarkar. Location consistency-a new memory model and cache consistency protocol. IEEE Trans. Comput., 49:798–813, August 2000. ISSN 0018-9340.
- K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In Proceedings of the 17th Annual International Symposium on Computer Architecture, pages 15–26, Seattle, Washington, May 1990.
- L. Lamport. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7): 558–565, July 1978.

Bibliography III

- D. Mosberger. Memory consistency models. SIGOPS Oper. Syst. Rev., 27:18–26, January 1993.ISSN 0163-5980.