# CPEG 852 — Advanced Topics in Computing Systems
# Memory Models
## Introduction to Memory Consistency Models

Stéphane Zuckerman

Computer Architecture & Parallel Systems Laboratory
Electrical & Computer Engineering Dept.
University of Delaware
140 Evans Hall Newark,DE 19716, United States
szuckerm@udel.edu

September 22, 2015

# Outline



1. Introduction to Memory Models
2. Overview of Shared Memory Systems
3. Memory Consistency Models
   - A Motivating Example
   - Uniform Memory Consistency Models
     - Strongest MCMs
     - Weaker Uniform MCMs
   - Non-Uniform Memory Consistency Models
     - Hardware-Oriented MCMs
   - Conclusion On MCMs

# Components of a Memory Model

UNIVERSITY OF DELAWARE

## A Short Intro

Traditionally, a memory model is described through two components:

- The addressing mode
- The memory consistency model

## Basics

- Traditionally associated with an architecture's given ISA
- Describe how memory can be accessed by a given process/thread
  - *e.g.*: Can the thread simply write at address `0×800FA` without supervision?

University of Delaware

## Physical (Flat) Memory Addressing

- No abstraction
- Often used in embedded systems (single-application-per-device)
- Usually: no hardware help to deal with memory isolation (*e.g.*, no TLB, *etc.*)
- Advantages:
  - Simple
  - Can specify specific memory ranges for specific applications at compile/link time
  - When you know what you are doing, probably leads to the most efficient memory/resource usage
- Shortcomings:
  - Makes it difficult to run more than one application at a time
  - The system software is in charge of ensuring processes and threads do not overlap when they are not collaborating

# Virtual Memory Addressing I

UNIVERSITY OF DELAWARE

## Principles

- Each process is given an address space (on Linux: 4GiB per process by default)
- The operating system (OS) decides which physical address ranges to assign to which process when it allocates memory
- Usually, the space is not contiguous: the physical memory is partitioned:
  - Into segments (the OS then computes the physical address using a base address + offset scheme)
  - In modern computer systems, pages are used (on x86 machines, one regular page = 4KiB)
- When a process/thread accesses a virtual memory address, it is then translated back to its corresponding physical address to access the content

# Virtual Memory Addressing II

UNIVERSITY OF DELAWARE

## Features

- Use of both hardware and software mechanisms (interaction between OS and TLB)
- Each process believes it has "all" of the memory to itself (in Linux/UNIX, the limit is usually 4GiB/process)
- Mostly implemented through paging and segmenting
- If the physical memory is too small for the process' needs, permanent storage (*i.e.*, HDDs, SSDs) are used
  - Unused pages are stored on disk
  - This is called swapping

# Virtual Memory Addressing III

University of Delaware

## Advantages

- Transparent to the user/programmer: no need to specify memory ranges at link time
- Enhances portability: no need to know what is "under the hood"
- Naturally provides process isolation thanks to segmentation and paging
  - If a thread or process tries to access a virtual address outside of its range, an exception is raised (`Segmentation Fault`)
  - Very efficient to allow multiple processes to access and compete for memory resources

# Virtual Memory Addressing IV

## Shortcomings

- Address translation is not free:
  - The system software (OS) is in charge of translating virtual pages to physical ones
  - There is a need for hardware assistance
  - The memory manager must also handle memory fragmentation
  -
- Translation Lookaside Buffers (TLBs) are small caches (often fully associative) that help the OS map virtual to physical addresses (and vice-versa)
  - As a result, a TLB miss incurs a much more severe penalty than a regular cache miss

UNIVERSITY OF DELAWARE

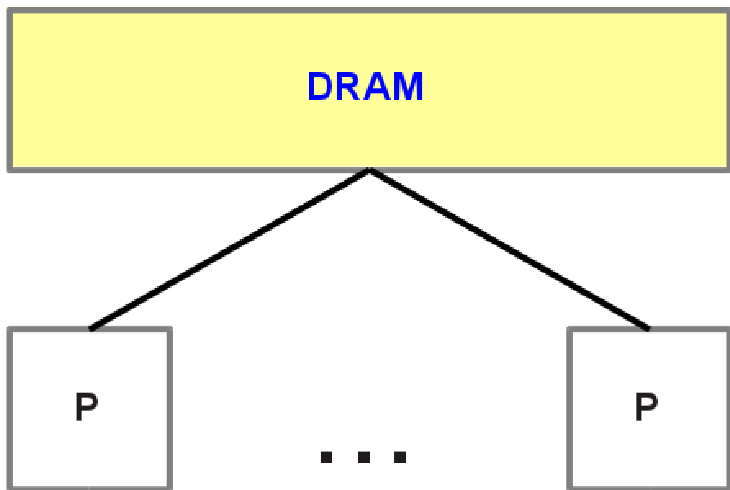
## Distributed Memory

- Fully distributed memory:
  - Different compute nodes have separate address spaces
  - To make nodes communicate, one needs to explicitly send data back and forth
  - Inside a given node, virtual memory may be used
- Distributed Shared Memory
  - Provides the illusion that memory is shared across physically separated nodes
  - The programming environment software layer provdes the abstraction
  - The system software (compiler, runtime) implements a communication layer to transparently send data across nodes

# Outline


1. Introduction to Memory Models
2. **Overview of Shared Memory Systems**
3. Memory Consistency Models
   - A Motivating Example
   - Uniform Memory Consistency Models
     - Strongest MCMs
     - Weaker Uniform MCMs
   - Non-Uniform Memory Consistency Models
     - Hardware-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

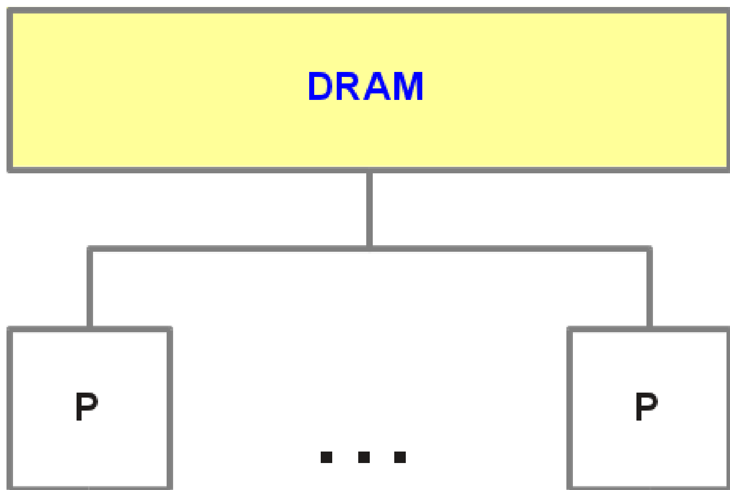
LOTS!

# Shortcomings of Shared-Memory Systems

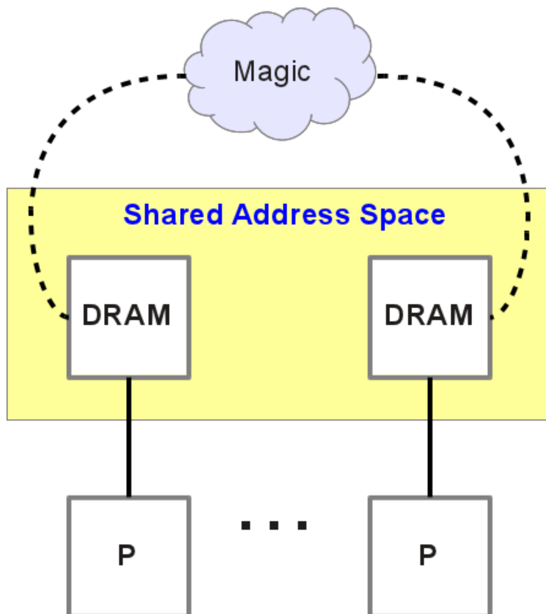
LOTS!

. . . But we will talk about them later.

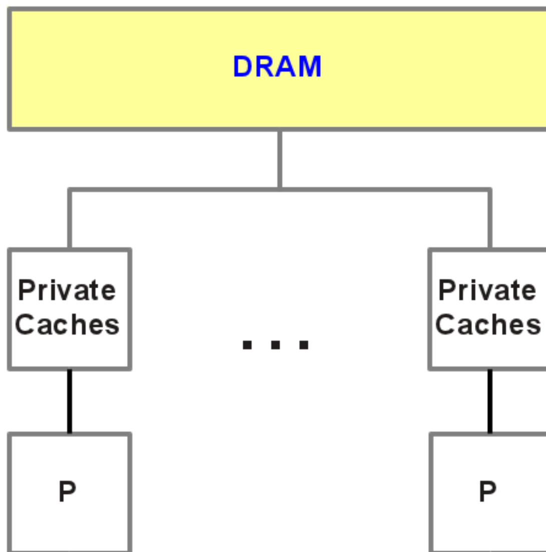
# Symmetric MultiProcessor Systems



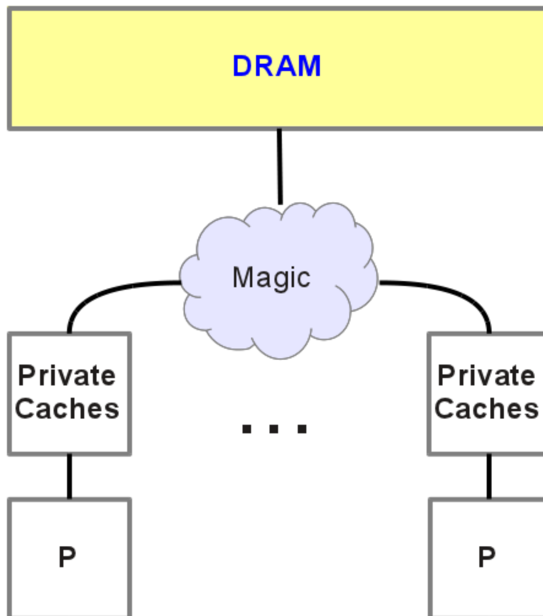
# Distributed Shared Memory Systems



# Why This Is More Complicated Than It Appears

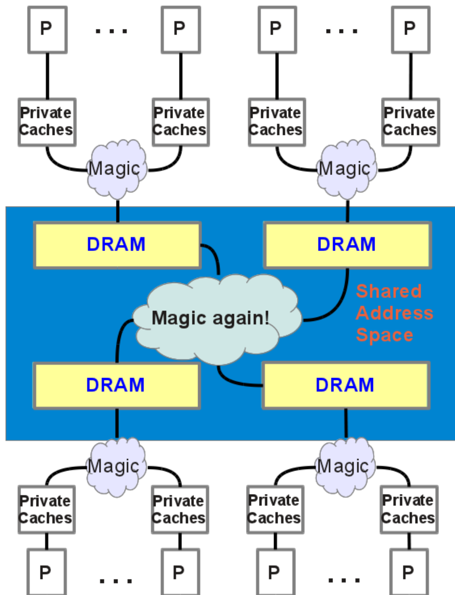


# Why This Is More Complicated Than It Appears

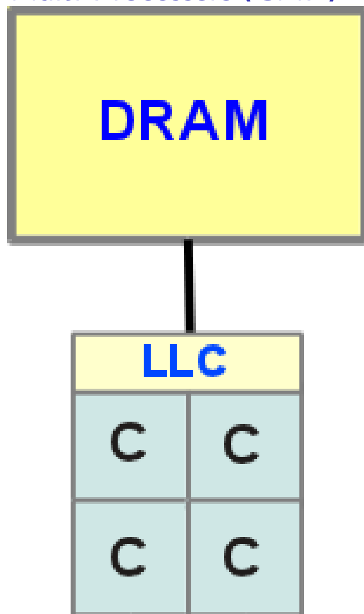


# Putting It All Together

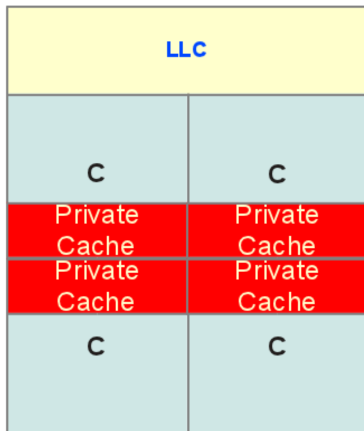
University of Delaware



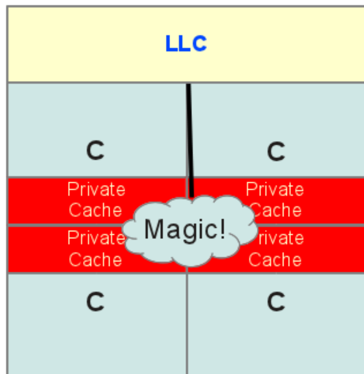
# The Advent of Chip Multi-Processors (CMP)



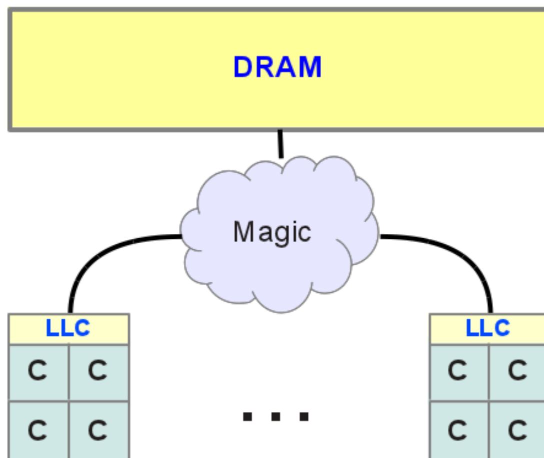
# The Advent of Chip Multi-Processors (CMP)



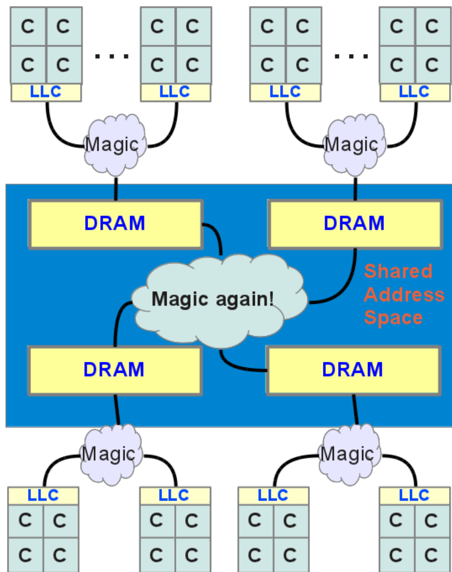
# The Advent of Chip Multi-Processors (CMP)



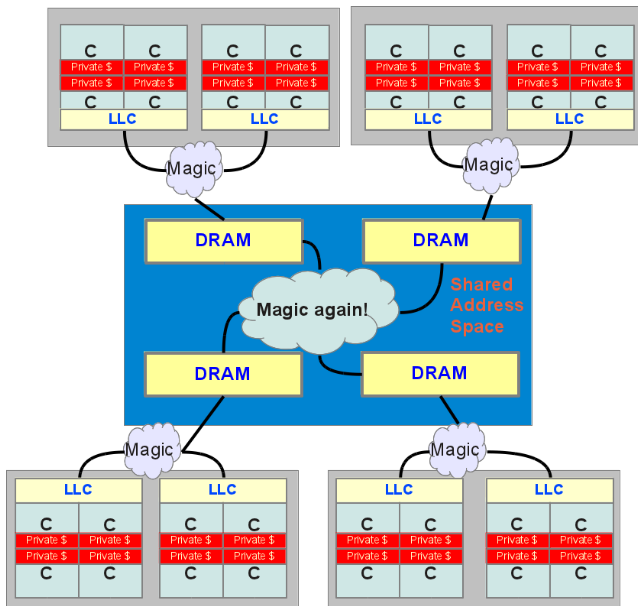
# Classical Compute Node in a Supercomputer



# Classical Compute Node in a Supercomputer



# Putting It All Together (Again!)



# Outline

UNIVERSITY OF DELAWARE


1. Introduction to Memory Models
2. Overview of Shared Memory Systems
3. Memory Consistency Models
   - A Motivating Example
   - Uniform Memory Consistency Models
     - Strongest MCMs
     - Weaker Uniform MCMs
   - Non-Uniform Memory Consistency Models
     - Hardware-Oriented MCMs
   - Conclusion On MCMs

# A Motivating Example



$$x \leftarrow 1$$
$$r1 \leftarrow y$$

# A Motivating Example

University of Delaware

$$y \leftarrow 1$$
$$r2 \leftarrow x$$

# A Motivating Example

University of Delaware

| Thread 0 | Thread 1 |
| --- | --- |
| $x \leftarrow 1$ | $y \leftarrow 1$ |
| $r1 \leftarrow y$ | $r2 \leftarrow x$ |

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

# What Memory Consistency is All About

UNIVERSITY OF DELAWARE

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

## 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 $x$?

# 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...

## 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

University of Delaware

1. Introduction to Memory Models
2. Overview of Shared Memory Systems
3. Memory Consistency Models
   - A Motivating Example
   - Uniform Memory Consistency Models
     - Strongest MCMs
     - Weaker Uniform MCMs
   - Non-Uniform Memory Consistency Models
     - Hardware-Oriented MCMs
   - Conclusion On MCMs

# 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.*

| Thread 0 | Thread 1 |
| --- | --- |
| $x \leftarrow 1$ | $y \leftarrow 1$ |
| $r1 \leftarrow y$ | $r2 \leftarrow x$ |

**Table:** Initially, $x = y = 0$.

**Is it possible to have $r1 = r2 = 0$ ?**

## Back to our Example

| Thread 0 | Thread 1 |
| --- | --- |
| $x \leftarrow 1$ | $y \leftarrow 1$ |
| $r1 \leftarrow y$ | $r2 \leftarrow x$ |

**Table:** Initially, $x = y = 0$.

**Is it possible to have $r1 = r2 = 0$ ?**

NO $\longrightarrow$ 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 $r1 = r2 = 0$

## Sequential Consistency and its Popularity

University of Delaware

- It behaves pretty much as one would expect in the context of a uniprocessor-multithread execution
  $\longrightarrow$ It is considered very intuitive
- It offers strong guarantees: a modification to memory *must* be seen by all other threads in a given program

# The Drawbacks of Sequential Consistency

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

$\longrightarrow$ How complicated is it to implement such a system in hardware ?

- $\rightarrow$ What about caches? Write buffers? etc.

$\longrightarrow$ How scalable is it ?

$\longrightarrow$ How expensive is it to implement that kind of consistency model?

# Coherence (Cache Consistency) (Kourosh Gharachorloo et al., 1990)

University of Delaware

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?

| Thread 0 | Thread 1 |
|---|---|
| $x \leftarrow 1$ | $y \leftarrow 1$ |
| $r1 \leftarrow y$ | $r2 \leftarrow x$ |

**Table:** Initially, $x = y = 0$

**Is it possible to get $r1 = r2 = 0$?**

UNIVERSITY OF DELAWARE

| Thread 0 | Thread 1 |
|---|---|
| $x \leftarrow 1$ | $y \leftarrow 1$ |
| $r1 \leftarrow y$ | $r2 \leftarrow x$ |

**Table:** Initially, $x = y = 0$

**Is it possible to get $r1 = r2 = 0$?**

YES!

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

# The Difference with Previous Models

University of Delaware

- 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, Scheurich, and Briggs, 1986) Weak Ordering (S. V. Adve and Hill, n.d.)

UNIVERSITY OF DELAWARE

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

# Yet Another Example

| Thread 0 | Thread 1 |
| --- | --- |
| $\cdots \leftarrow y,\ y = 2$ | $\cdots \leftarrow x,\ x = 1$ |
| $z \overset{sync}{\leftarrow} 3$ | $z \overset{sync}{\leftarrow} 4$ |
| $x \leftarrow 1$ | $y \leftarrow 2$ |

**Table:** Is this WC?

# Yet Another Example

| Thread 0 | Thread 1 |
| --- | --- |
| $\cdots \leftarrow y,\ y = 2$ | $\cdots \leftarrow x,\ x = 1$ |
| $z \stackrel{sync}{\leftarrow} 3$ | $z \stackrel{sync}{\leftarrow} 4$ |
| $x \leftarrow 1$ | $y \leftarrow 2$ |

**Table:** Is this WC? No.

## Release Consistency (Kourosh 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:

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

## More Examples (See (S. Adve, Pai, and Ranganathan, 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? PC? WC? RC?

## More Examples (See (S. Adve, Pai, and Ranganathan, 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? PC? WC? RC?

**Solution**

SC,PC `reg1 = 64` ; `reg2 = 55`

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

## More Examples (See (S. Adve and K. Gharachorloo, 1996))

UNIVERSITY OF DELAWARE

| 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? PC? WC? RC?

## More Examples (See (S. Adve and K. Gharachorloo, 1996))

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

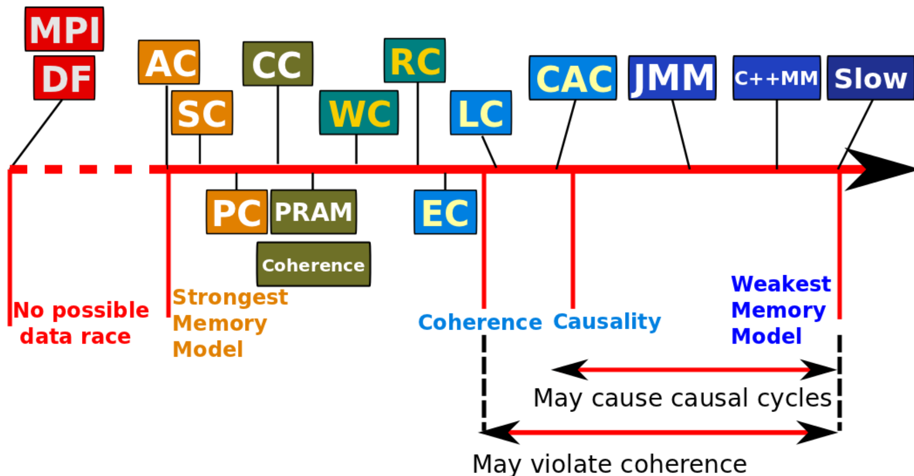
**Table:** Ex2: What are the legal values in SC? PC? WC? RC?

**Solution**

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

PC,WC,RC `reg1 = 0` or `1` ; `reg2 = 0` or `1`

# 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

- S.V. Adve and K. Gharachorloo (1996). "Shared memory consistency models: a tutorial". In: *Computer* 29.12, pp. 66–76. ISSN: 0018-9162. DOI: 10.1109/2.546611
- David Mosberger (1993). "Memory consistency models". In: *SIGOPS Oper. Syst. Rev.* 27 (1), pp. 18–26. ISSN: 0163-5980. DOI: http://doi.acm.org/10.1145/160551.160553. URL: http://doi.acm.org/10.1145/160551.160553
- John L Hennessy and David A Patterson (2011). *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann. ISBN: 9780123838728

# References I


- S.V. Adve and K. Gharachorloo (1996). "Shared memory consistency models: a tutorial". In: *Computer* 29.12, pp. 66–76. ISSN: 0018-9162. DOI: 10.1109/2.546611
- David Mosberger (1993). "Memory consistency models". In: *SIGOPS Oper. Syst. Rev.* 27 (1), pp. 18–26. ISSN: 0163-5980. DOI: http://doi.acm.org/10.1145/160551.160553. URL: http://doi.acm.org/10.1145/160551.160553
- Jeremy Manson, William Pugh, and Sarita V. Adve (2005). "The Java memory model". In: *SIGPLAN Not.* 40 (1), pp. 378–391. ISSN: 0362-1340. DOI: http://doi.acm.org/10.1145/1047659.1040336. URL: http://doi.acm.org/10.1145/1047659.1040336
- Hans-J. Boehm and Sarita V. Adve (2008). "Foundations of the C++ concurrency memory model". In: *Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation*. PLDI '08. Tucson, AZ, USA: ACM, pp. 68–78. ISBN: 978-1-59593-860-2. DOI: http://doi.acm.org/10.1145/1375581.1375591. URL: http://doi.acm.org/10.1145/1375581.1375591
- Phillip W. Hutto and Mustaque Ahamad (1990). "Slow Memory: Weakening Consistency to Enchance Concurrency in Distributed Shared Memories". In: *ICDCS*, pp. 302–309
- Guang R. Gao and Vivek Sarkar (1995). "Location Consistency: Stepping Beyond the Memory Coherence Barrier". In: *ICPP (2)*, pp. 73–76

# References II


- Guang R. Gao and Vivek Sarkar (1997). "On the Importance of an End-To-End View of Memory Consistency in Future Computer Systems". In: *Proceedings of the International Symposium on High Performance Computing*. London, UK: Springer-Verlag, pp. 30–41. ISBN: 3-540-63766-4. URL: http://portal.acm.org/citation.cfm?id=646346.690059
- Guang R. Gao and Vivek Sarkar (2000). "Location Consistency-A New Memory Model and Cache Consistency Protocol". In: *IEEE Trans. Comput.* 49 (8), pp. 798–813. ISSN: 0018-9340. DOI: 10.1109/12.868026. URL: http://portal.acm.org/citation.cfm?id=354862.354865
- Chen Chen et al. (2010). "A Study of a Software Cache Implementation of the OpenMP Memory Model for Multicore and Manycore Architectures". In: *Euro-Par (2)*, pp. 341–352