CPEG 852 — Advanced Topics in Computing Systems
Memory Models
A Survey of 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 29, 2015

Outline

Outline

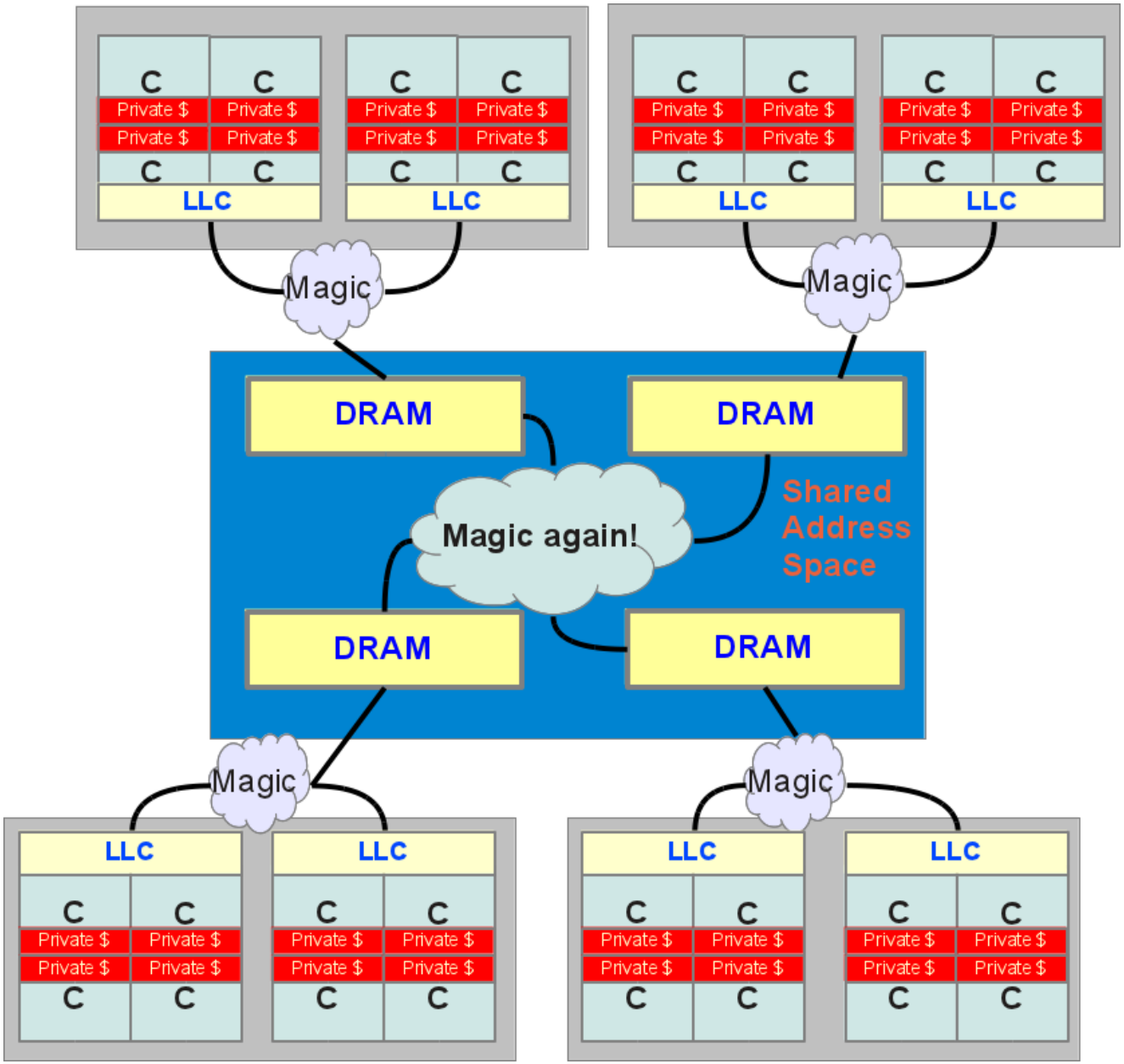
Outline

Components of a Memory Model

Traditionally, a memory model is described through two components:

- The addressing mode
- The memory consistency model

A Representative Picture of a (Distributed) Shared-Memory System



Outline

Two major addressing modes dominate the landscape:

- Physical (Flat) Memory Addressing
  - 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
    - 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

Memory Models II

Addressing Modes

- The physical (real) memory space is usually not contiguous, and is decomposed in segments and/or pages
- From the program/process' perspective, it is given a unique contiguous memory space
- Processes access virtual addresses, which are mapped to their physical counterpart
  - The hardware and system software collaborate: use of TLBs
- Features:
  - Processes' memory spaces are isolated from each other (cannot access another process' memory space)
  - Allows for swapping if the main memory is not big enough (but this is very slow)
- 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"

- Shortcomings:
  - Address translation is not free (OS overhead for managing physical/virtual address mapping)
  - Hardware assistance is required for efficient execution — and even then, a TLB miss incurs a high penalty
  - Memory management using paging means managing physical memory fragmentation

Addressing Physically Distributed Memory

Distributed Memory

- Fully distributed memory:
  - Different compute nodes have separate address spaces
  - Communications are explicit
- Distributed Shared Memory
  - Provides the illusion that memory is shared across physically separated nodes
  - The system software (compiler, runtime) implements a communication layer to transparently send data across nodes

Outline

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

Outline

A Motivating Example

$x \leftarrow 1$

$r1 \leftarrow y$

A Motivating Example

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

A Motivating 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 $r1 = r2 = 0$?

What Memory Consistency is All About

What Memory Consistency is All About

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.

Outline

Atomic Consistency (L. 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

Atomic Consistency (L. 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 (Leslie 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.*

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 $r1 = r2 = 0$?

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

- 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)

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 $r1 = r2 = 0$?

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 $r1 = r2 = 0$?

YES!

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

Pipelined RAM (P-RAM)

P-RAM is achieved if

- all memory operations follow program order (w.r.t. process P), and
- all memory writes appear in some order to all processors of the system, but all memory writes issued by the same process P are in-order.

$\rightarrow$ Each process P_i must issue memory operations w.r.t. its program order, and sees other processes' writes in any interleaved order (but all the memory writes issued by a given process P_j, $i \neq j$ are seen in-order).

Is our First Example P-RAM?

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 $r1 = r2 = 0$?

Is our First Example P-RAM?

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!

T0 $x \leftarrow 1, r1 \leftarrow y, y \leftarrow 1$

T1 $y \leftarrow 1, r2 \leftarrow x, x \leftarrow 1$

Is P-RAM Weaker than Coherence (or vice-versa)?

Thread 0	Thread 1
$x \leftarrow 0$	$y \leftarrow 0$
$x \leftarrow 1$	$y \leftarrow 1$
$\cdots \leftarrow y,\ y = 0$	$\cdots \leftarrow x,\ x = 0$

Table: Initially, $x = y = 0$. Is this program history SC? Coherent? P-RAM?

Is P-RAM Weaker than Coherence (or vice-versa)?

Thread 0	Thread 1
$x \leftarrow 0$	$y \leftarrow 0$
$x \leftarrow 1$	$y \leftarrow 1$
$\cdots \leftarrow y,\ y = 0$	$\cdots \leftarrow x,\ x = 0$

Table: Initially, $x = y = 0$. Is this program history SC? Coherent? P-RAM?

Not SC, but it is P-RAM and Coherent.

Is P-RAM Weaker than Coherence (or vice-versa)?

Thread 0	Thread 1
$x \leftarrow 0$	
$x \leftarrow 1$	
$y \leftarrow 2$	$\cdots \leftarrow y,\ y = 2$
	$\cdots \leftarrow x,\ x = 0$

Table: Initially, $x = y = 0$. Is this program history SC? Coherent? P-RAM?

Is P-RAM Weaker than Coherence (or vice-versa)?

Thread 0	Thread 1
$x \leftarrow 0$	
$x \leftarrow 1$	
$y \leftarrow 2$	$\cdots \leftarrow y,\ y = 2$
	$\cdots \leftarrow x,\ x = 0$

Table: Initially, $x = y = 0$. Is this program history SC? Coherent? P-RAM?

Not SC, not P-RAM.

Is P-RAM Weaker than Coherence (or vice-versa)?

Thread 0	Thread 1
$x \leftarrow 0$	$x \leftarrow 1$
$\cdots \leftarrow x,\ x = 1$	$\cdots \leftarrow x,\ x = 0$

Table: Initially, $x = y = 0$. Is this program history SC? Coherent? P-RAM?

Is P-RAM Weaker than Coherence (or vice-versa)?

Thread 0	Thread 1
$x \leftarrow 0$	$x \leftarrow 1$
$\cdots \leftarrow x,\ x = 1$	$\cdots \leftarrow x,\ x = 0$

Table: Initially, $x = y = 0$. Is this program history SC? Coherent? P-RAM?

Not SC, not coherent.

Processor Consistency
(Goodman, 1989; Ahamad et al., 1993)

A system is PC if

- it is coherent
- it is P-RAM
- Both conditions must be true *simultaneously*

PC is supposedly easier to implement than SC

Another Example

Thread 0	Thread 1	Thread 2
$x \leftarrow 0$		
$y \leftarrow 0$	$\cdots \leftarrow y,\ y = 0$	
	$x \leftarrow 1$	$\cdots \leftarrow x,\ x = 1$
		$\cdots \leftarrow x,\ x = 0$

Table: Initially, $x = y = 0$. Is this program history SC? Coherent? P-RAM? PC?

Another Example

Thread 0	Thread 1	Thread 2
$x \leftarrow 0$		
$y \leftarrow 0$	$\cdots \leftarrow y,\ y = 0$	
	$x \leftarrow 1$	$\cdots \leftarrow x,\ x = 1$
		$\cdots \leftarrow x,\ x = 0$

Table: Initially, $x = y = 0$. Is this program history SC? Coherent? P-RAM? PC?

Not SC, not PC, but P-RAM or coherent.

Outline

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

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

Table: Initially, $x = y = 0$. Is this WC?

Yet Another Example

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

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

- Ordinary operations issued before an acquire operation can bypass it and perform (or *complete*) after it. Ordinary operations that were issued after an acquire must also perform after .
- Ordinary operations issued before a release operation must perform (or *complete*) before it does. Ordinary operations that were issued after a release can perform before it does.
- 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)

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`

The C++ Memory Model

Very easy to understand:

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

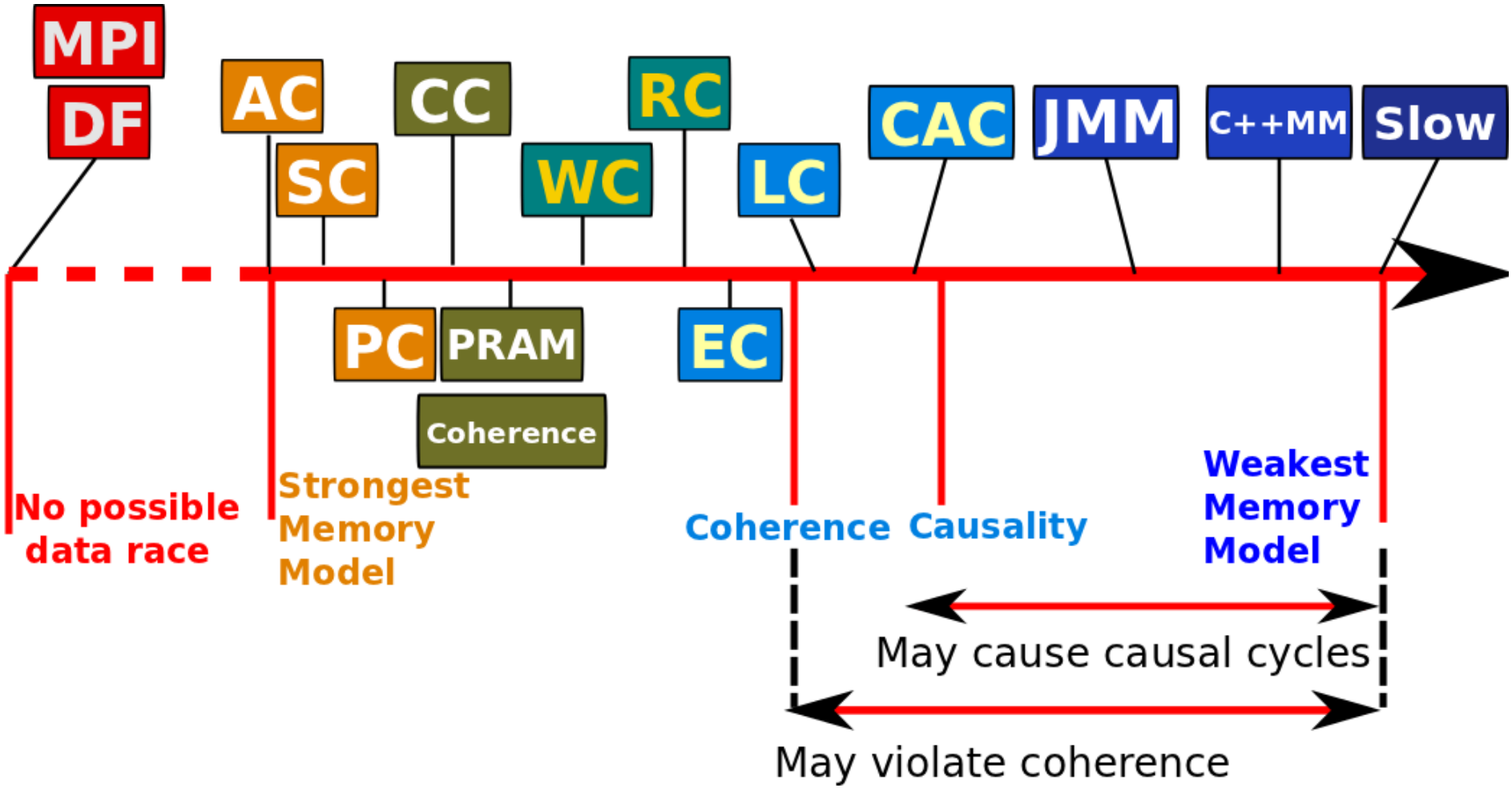
The C++ Memory Model

Very easy to understand:

- Synchronizing accesses (through the `atomic` keyword) are SC
- any incorrectly synchronized behavior implies an *undefined behavior*,
  - . . . which really means by issuing a data-race you can have initiated a new TCP connection in order to order 20 elephants to be delivered by next Saturday

Outline

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 (or the more clever the compiler and runtime systems must be)

If You Want to Know More...

- 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