Memory Consistency Models

Mengjia Yan
Computer Science and Artificial Intelligence Lab
M.I.T.

Coherence vs Consistency

- Cache coherence makes private caches invisible to software
  - Concerns reads/writes to a single memory location

Coherence vs Consistency

- Cache coherence makes private caches invisible to software
  - Concerns reads/writes to a single memory location

- Memory consistency models precisely specify how memory behaves with respect to read and write operations from multiple processors
  - Concerns reads/writes to multiple memory locations

Why Consistency Matters

Initial memory contents

a: 0

flag: 0

Processor 1	*Processor 2*
Store (a), 10;	L: Load r1, (flag);
Store (flag), 1;	if r1 == 0 goto L;
	Load r2, (a);

- What value does r2 hold after both processors finish running this code?

Why Consistency Matters

Initial memory contents

a: 0

flag: 0

Processor 1   *Processor 2*

Store (a), 10;   L: Load r1, (flag);

Store (flag), 1;   if r1 == 0 goto L;

                                    Load r2, (a);

- What value does r2 hold after both processors finish running this code?

It depends on the order in which processor 2 observes processor 1's stores!

Why Consistency Matters

Initial memory contents

a: 0

flag: 0

Processor 1

Store (a), 10;

Store (flag), 1;

Processor 2

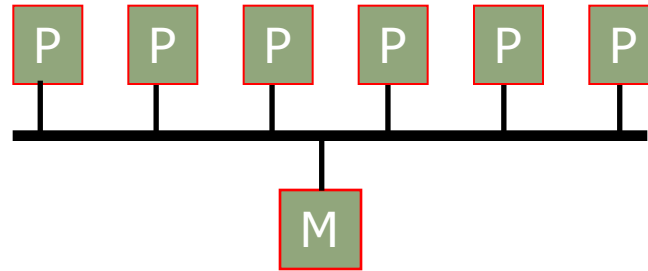L: Load r1, (flag);

    if r1 == 0 goto L;

    Load r2, (a);

- What value does r2 hold after both processors finish running this code?

It depends on the order in which processor 2 observes processor 1's stores!

10 if Store (flag) > Store (a); 0 or 10 otherwise

Sequential Consistency
A Straightforward Memory Model

P P P P P P

M

"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 the order specified by the program"

Leslie Lamport

Sequential Consistency =
        arbitrary *order-preserving interleaving*
        of memory references of sequential programs

Sequential Consistency

Processor 1

Store (a), 10;

Store (flag), 1;

Processor 2

L: Load r1, (flag);

    if r_1 == 0 goto L;

    Load r2, (a);

- In-order instruction execution
- Atomic loads and stores

Sequential Consistency

Processor 1

Store (a), 10;

Store (flag), 1;

Processor 2

L: Load r1, (flag);

if r_1 == 0 goto L;

Load r2, (a);

- In-order instruction execution
- Atomic loads and stores

Sequential Consistency

Processor 1

Store (a), 10;
Store (flag), 1;

Processor 2

L: Load r1, (flag);

if r_1 == 0 goto L;

Load r2, (a);

- In-order instruction execution
- Atomic loads and stores

Sequential Consistency

Processor 1

Store (a), 10;

Store (flag), 1;

Processor 2

L: Load r1, (flag);

if r_1 == 0 goto L;

Load r2, (a);

- In-order instruction execution
- Atomic loads and stores

Sequential Consistency

Processor 1

Store (a), 10;

Store (flag), 1;

Processor 2

L: Load r1, (flag);

if r_1 == 0 goto L;

Load r2, (a);

- In-order instruction execution
- Atomic loads and stores

Sequential Consistency

Processor 1

Store (a), 10;
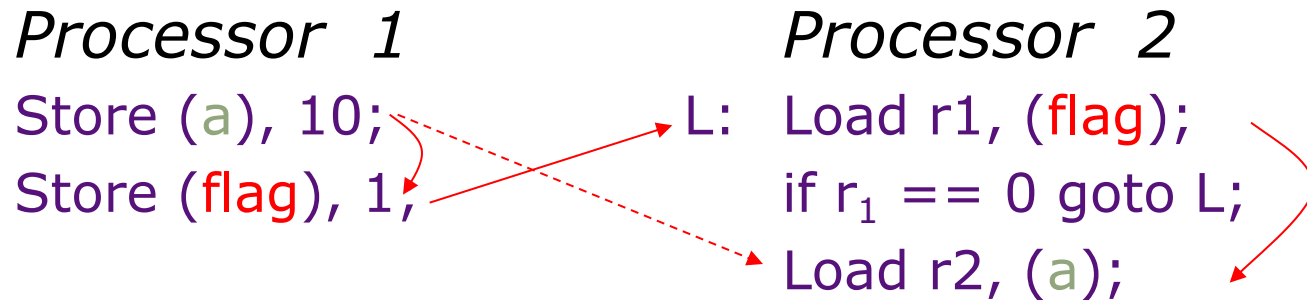
Store (flag), 1;

Processor 2

L: Load r1, (flag);
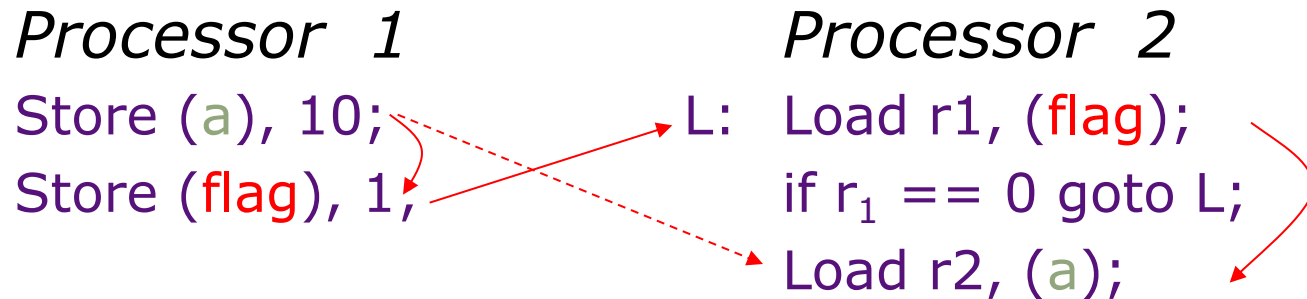
    if r_1 == 0 goto L;

    Load r2, (a);

- In-order instruction execution
- Atomic loads and stores

SC is easy to understand, but architects and compiler writers want to violate it for performance

Memory Model Issues

Architectural optimizations that are correct for uniprocessors often violate sequential consistency and result in a new memory model for multiprocessors

Consistency Models

- ## Sequential Consistency
  - All reads and writes in order

- ## Relaxed Consistency (one or more of the following)
  - Loads may be reordered after loads
    - e.g., PA-RISC, Power, Alpha
  - Loads may be reordered after stores
    - e.g., PA-RISC, Power, Alpha
  - Stores may be reordered after stores
    - e.g., PA-RISC, Power, Alpha, PSO
  - Stores may be reordered after loads
    - e.g., PA-RISC, Power, Alpha, PSO, TSO, x86
  - Other more esoteric characteristics
    - e.g., Alpha

Committed Store Buffers

- CPU can continue execution while earlier committed stores are still propagating through memory system

    – Processor can commit other instructions (including loads and stores) while first store is committing to memory

    – Committed store buffer can be combined with speculative store buffer in an out-of-order CPU

- Local loads can bypass values from buffered stores to same address
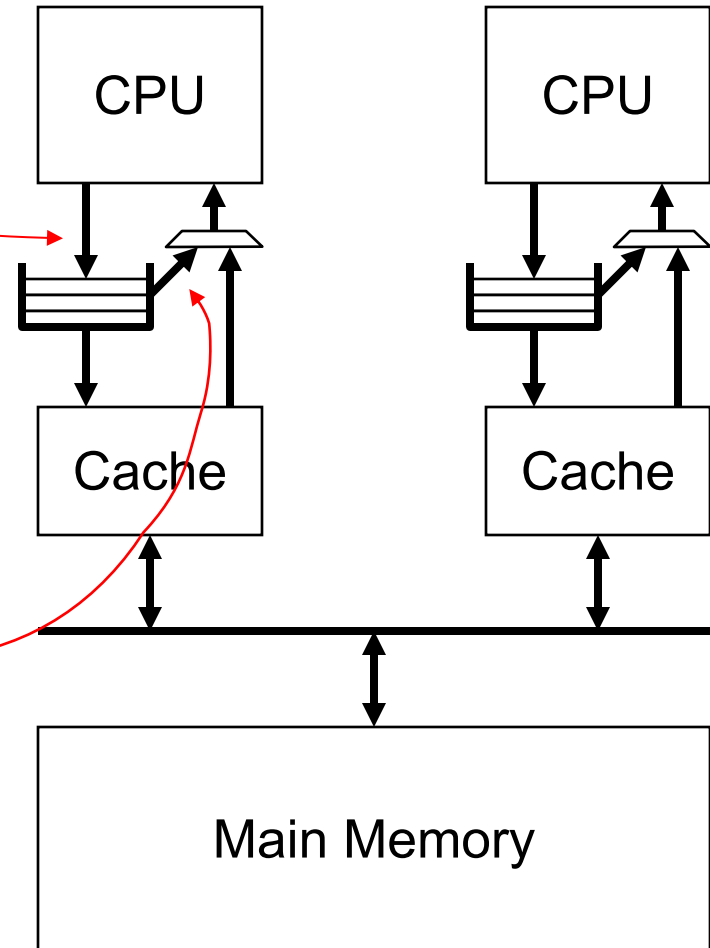
Example 1: Store Buffers

Process 1	*Process 2*
Store (flag$_1$),1;	Store (flag$_2$),1;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Initially, all memory locations contain zeros

Example 1: Store Buffers

Process 1	Process 2
Store (flag$_1$),1;	Store (flag$_2$),1;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Question: Is it possible that r$_1$=0 and r$_2$=0?

Initially, all memory locations contain zeros

Example 1: Store Buffers

Process 1	*Process 2*
Store (flag$_1$),1;	Store (flag$_2$),1;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Question: Is it possible that r$_1$=0 and r$_2$=0?
- *Sequential consistency:*

Initially, all memory locations contain zeros

Example 1: Store Buffers

Process 1	*Process 2*
Store (flag$_1$),1;	Store (flag$_2$),1;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Question: Is it possible that r$_1$=0 and r$_2$=0?
- *Sequential consistency:*   *No*

Initially, all memory
locations contain zeros

Example 1: Store Buffers

Process 1	Process 2
Store (flag$_1$),1;	Store (flag$_2$),1;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Question: Is it possible that $r_1=0$ and $r_2=0$?
- *Sequential consistency:*   *No*

- *Suppose Loads can go ahead of Stores waiting in the store buffer:*

Initially, all memory locations contain zeros

Example 1: Store Buffers

Process 1	*Process 2*
Store (flag$_1$),1;	Store (flag$_2$),1;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Question: Is it possible that r$_1$=0 and r$_2$=0?

- *Sequential consistency:*   *No*

- *Suppose Loads can go ahead of Stores waiting in the store buffer: Yes!*

Initially, all memory locations contain zeros

Example 1: Store Buffers

Process 1	Process 2
Store (flag$_1$),1;	Store (flag$_2$),1;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Question: Is it possible that $r_1=0$ and $r_2=0$?
- *Sequential consistency:* *No*

- *Suppose Loads can go ahead of Stores waiting in the store buffer:* *Yes!*

Total Store Order (TSO):
Sun SPARC, IBM 370

Initially, all memory
locations contain zeros

Example 2: Store-Load Bypassing

	Process 1	_Process 2_
Store $(flag_1)$, 1;		Store $(flag_2)$, 1;
Load r_3, $(flag_1)$;		Load r_4, $(flag_2)$;
Load r_1, $(flag_2)$;		Load r_2, $(flag_1)$;

Example 2: Store-Load Bypassing

Process 1	_Process 2_
Store (flag$_1$), 1;	Store (flag$_2$), 1;
Load r$_3$, (flag$_1$);	Load r$_4$, (flag$_2$);
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Question: Do extra Loads have any effect?

Example 2: Store-Load Bypassing

Process 1	_Process 2_
Store (flag$_1$), 1;	Store (flag$_2$), 1;
Load r$_3$, (flag$_1$);	Load r$_4$, (flag$_2$);
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Question: Do extra Loads have any effect?
- *Sequential consistency: No*

Example 2: Store-Load Bypassing

Process 1	Process 2
Store (flag$_1$), 1;	Store (flag$_2$), 1;
Load r$_3$, (flag$_1$);	Load r$_4$, (flag$_2$);
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Question: Do extra Loads have any effect?

- *Sequential consistency: No*

- *Suppose Store-Load bypassing is permitted in the store buffer*

Example 2: Store-Load Bypassing

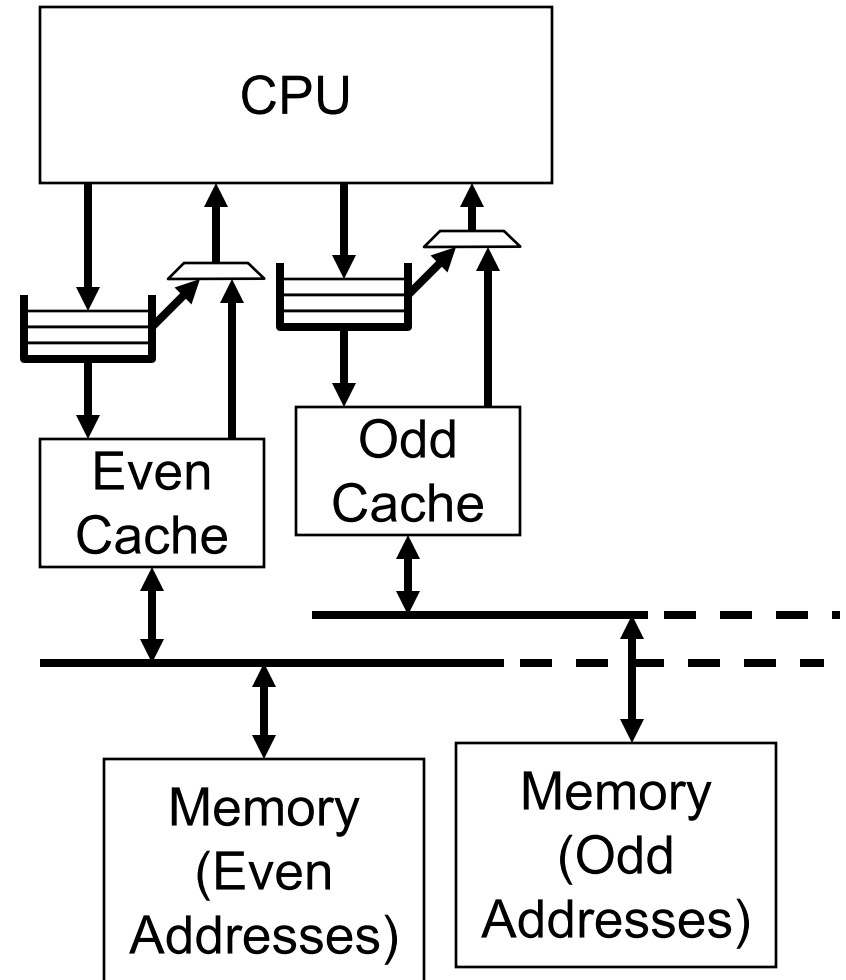| _Process 1_	_Process 2_
Store (flag$_1$), 1;	Store (flag$_2$), 1;
Load r$_3$, (flag$_1$);	Load r$_4$, (flag$_2$);
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Question: Do extra Loads have any effect?

- _Sequential consistency:_ _No_

- _Suppose Store-Load bypassing is permitted in the store buffer_
  - No effect in Sparc's TSO model, still not SC

Example 2: Store-Load Bypassing

Process 1	Process 2
Store (flag$_1$), 1;	Store (flag$_2$), 1;
Load r$_3$, (flag$_1$);	Load r$_4$, (flag$_2$);
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Question: Do extra Loads have any effect?

- *Sequential consistency: No*

- *Suppose Store-Load bypassing is permitted in the store buffer*
  - No effect in Sparc's TSO model, still not SC
  - In IBM 370, a load cannot return a written value until it is visible to other processors => implicitly adds a memory fence, looks like SC

Interleaved Memory System

- Achieve greater throughput by spreading memory addresses across two or more parallel memory subsystems

  - In snooping system, can have two or more snoops in progress at same time (e.g., Sun UE10K system has four interleaved snooping busses)

  - Greater bandwidth from main memory system as two memory modules can be accessed in parallel

CPU

Even Cache

Odd Cache

Memory (Even Addresses)

Memory (Odd Addresses)

Example 3: Non-FIFO Store buffers

Process 1	*Process 2*
Store (a), 1;	Load r_1, (flag);
Store (flag), 1;	Load r_2, (a);

Example 3: Non-FIFO Store buffers

Process 1	_Process 2_
Store (a), 1;	Load r_1, (flag);
Store (flag), 1;	Load r_2, (a);

Question: Is it possible that $r_1=1$ but $r_2=0$?

Example 3: Non-FIFO Store buffers

Process 1	Process 2
Store (a), 1;	Load r_1, (flag);
Store (flag), 1;	Load r_2, (a);

Question: Is it possible that $r_1=1$ but $r_2=0$?
- *Sequential consistency: No*

Example 3: Non-FIFO Store buffers

Process 1	Process 2
Store (a), 1;	Load r_1, (flag);
Store (flag), 1;	Load r_2, (a);

Question: Is it possible that $r_1=1$ but $r_2=0$?
- Sequential consistency: *No*
- *With non-FIFO store buffers: Yes*

Example 3: Non-FIFO Store buffers

Process 1	Process 2
Store (a), 1;	Load r_1, (flag);
Store (flag), 1;	Load r_2, (a);

Question: Is it possible that $r_1=1$ but $r_2=0$?
- *Sequential consistency: No*
- *With non-FIFO store buffers: Yes*

Sparc's PSO memory model

Example 4: Non-Blocking Caches

Process 1	*Process 2*
Store (a), 1;	Load r_1, (flag);
Store (flag), 1;	Load r_2, (a);

Example 4: Non-Blocking Caches

Process 1	_Process 2_
Store (a), 1;	Load r_1, (flag);
Store (flag), 1;	Load r_2, (a);

Question: Is it possible that $r_1=1$ but $r_2=0$?

Example 4: Non-Blocking Caches

Process 1	Process 2
Store (a), 1;	Load r_1, (flag);
Store (flag), 1;	Load r_2, (a);

Question: Is it possible that $r_1=1$ but $r_2=0$?
- *Sequential consistency: No*

Example 4: Non-Blocking Caches

Process 1	_Process 2_
Store (a), 1;	Load r_1, (flag);
Store (flag), 1;	Load r_2, (a);

Question: Is it possible that $r_1=1$ but $r_2=0$?

- _Sequential consistency:_ _No_

- _Assuming stores are ordered: Yes because Loads can be reordered_

Example 4: Non-Blocking Caches

Process 1	*Process 2*
Store (a), 1;	Load r_1, (flag);
Store (flag), 1;	Load r_2, (a);

Question: Is it possible that $r_1=1$ but $r_2=0$?
- *Sequential consistency: No*

- *Assuming stores are ordered: Yes because Loads can be reordered*

Alpha, Sparc's RMO, PowerPC's WO

Example 5: Store Atomicity

Process 1	*Process 2*	*Process 3*	*Process 4*
Store (a),1;	Store (a),2;	Load r_1, (a);	Load r_3, (a);
		Load r_2, (a);	Load r_4, (a);

Example 5: Store Atomicity

Process 1	Process 2	Process 3	Process 4
Store (a),1;	Store (a),2;	Load r_1, (a);	Load r_3, (a);
		Load r_2, (a);	Load r_4, (a);

Question: Is it possible that $r_1=1$ and $r_2=2$ but $r_3=2$ and $r_4=1$?

Example 5: Store Atomicity

Process 1   _Process 2_   _Process 3_   _Process 4_

Store (a),1;   Store (a),2;   Load r_1, (a);   Load r_3, (a);

Load r_2, (a);   Load r_4, (a);

Question: Is it possible that $r_1=1$ and $r_2=2$ but $r_3=2$ and $r_4=1$?

- _Sequential consistency:_

Example 5: Store Atomicity

Process 1   _Process 2_   _Process 3_   _Process 4_

Store (a),1;   Store (a),2;   Load r_1, (a);   Load r_3, (a);

Load r_2, (a);   Load r_4, (a);

Question: Is it possible that $r_1=1$ and $r_2=2$ but $r_3=2$ and $r_4=1$?

- _Sequential consistency:_ _No_

Example 5: Store Atomicity

Process 1	_Process 2_	_Process 3_	_Process 4_
Store (a),1;	Store (a),2;	Load r_1, (a);	Load r_3, (a);
		Load r_2, (a);	Load r_4, (a);

Question: Is it possible that $r_1=1$ and $r_2=2$ but $r_3=2$ and $r_4=1$?

- *Sequential consistency:* No
- *Even if Loads on a processor are ordered, the different ordering of stores can be observed if the Store operation is not atomic.*

Example 6: Causality

Process 1	*Process 2*	*Process 3*
Store (flag$_1$), 1;	Load r$_1$, (flag$_1$);	Load r$_2$, (flag$_2$);
	Store (flag$_2$), 1;	Load r$_3$, (flag$_1$);

Example 6: Causality

Process 1	_Process 2_	_Process 3_
Store (flag$_1$), 1;	Load r$_1$, (flag$_1$);	Load r$_2$, (flag$_2$);
	Store (flag$_2$), 1;	Load r$_3$, (flag$_1$);

Question: Is it possible that $r_1=1$ and $r_2=1$ but $r_3=0$?

Example 6: Causality

Process 1	Process 2	Process 3
Store (flag$_1$), 1;	Load r$_1$, (flag$_1$);	Load r$_2$, (flag$_2$);
	Store (flag$_2$), 1;	Load r$_3$, (flag$_1$);

Question: Is it possible that r$_1$=1 and r$_2$=1 but r$_3$=0 ?

- *Sequential consistency:* *No*

-

Example 6: Causality

Process 1	*Process 2*	*Process 3*
Store (flag$_1$), 1;	Load r$_1$, (flag$_1$);	Load r$_2$, (flag$_2$);
	Store (flag$_2$), 1;	Load r$_3$, (flag$_1$);

Question: Is it possible that $r_1=1$ and $r_2=1$ but $r_3=0$?

- *Sequential consistency: No*

- *With load/load reordering:*

-

Example 6: Causality

Process 1	Process 2	Process 3
Store (flag$_1$), 1;	Load r$_1$, (flag$_1$);	Load r$_2$, (flag$_2$);
	Store (flag$_2$), 1;	Load r$_3$, (flag$_1$);

Question: Is it possible that $r_1=1$ and $r_2=1$ but $r_3=0$?

- *Sequential consistency: No*

- *With load/load reordering: Yes*

    Alpha

Example 7: Register Renaming

	Process 1		_Process 2_

Process 1

Store (flag$_1$), r$_1$;

Load r$_1$, (flag$_2$);

Process 2

Store (flag$_2$), r$_2$;

Load r$_2$, (flag$_1$);

Initially both r$_1$ and r$_2$ contain 1.

Example 7: Register Renaming

Process 1	Process 2
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_2$;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Initially both r$_1$ and r$_2$ contain 1.

Question: Is it possible that r$_1$=0 but r$_2$=0?

Example 7: Register Renaming

Process 1	_Process 2_
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_2$;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Initially both r$_1$ and r$_2$ contain 1.

Question: Is it possible that r$_1$=0 but r$_2$=0?
- _Sequential consistency: No_

Example 7: Register Renaming

Process 1	Process 2
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_2$;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Initially both r$_1$ and r$_2$ contain 1.

Question: Is it possible that r$_1$=0 but r$_2$=0?
- *Sequential consistency: No*

- *Register renaming: Yes because it removes anti-dependencies*

Example 7: Register Renaming

Process 1	*Process 2*

Store (flag$_1$), r$_1$;   Store (flag$_2$), r$_2$;

Load r$_1$, (flag$_2$);   Load r$_2$, (flag$_1$);

> Initially both r$_1$ and r$_2$ contain 1.

Question: Is it possible that r$_1$=0 but r$_2$=0?

- *Sequential consistency: No*

- *Register renaming: Yes because it removes anti-dependencies*
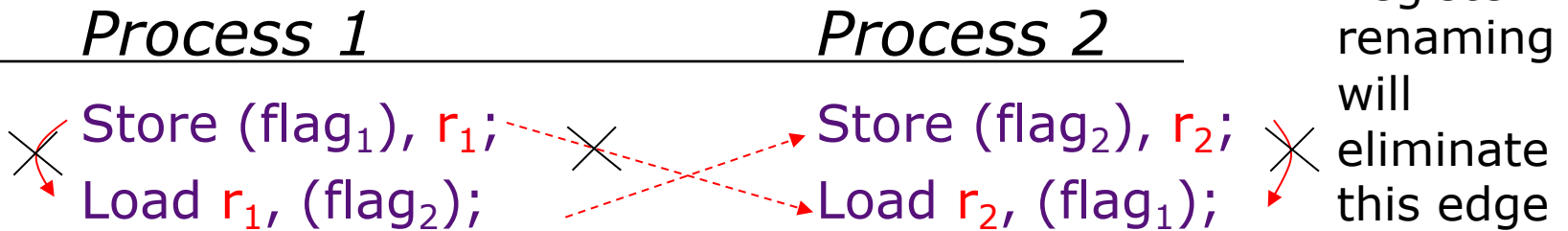
Example 7: Register Renaming

	Process 1		*Process 2*

Store (flag$_1$), r$_1$;   Store (flag$_2$), r$_2$;

Load r$_1$, (flag$_2$);   Load r$_2$, (flag$_1$);

Initially both r$_1$ and r$_2$ contain 1.

Question: Is it possible that r$_1$=0 but r$_2$=0?
- *Sequential consistency: No*
- *Register renaming: Yes because it removes anti-dependencies*
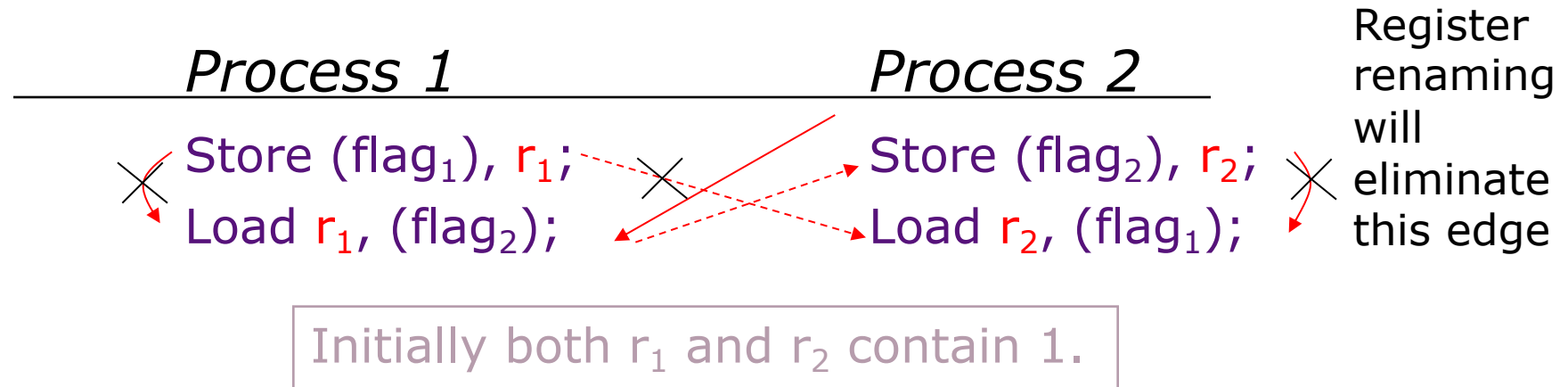
Example 7: Register Renaming

	Process 1	_Process 2_

Process 1 | _Process 2_

Store (flag$_1$), r$_1$;   Store (flag$_2$), r$_2$;

Load r$_1$, (flag$_2$);   Load r$_2$, (flag$_1$);

Initially both r$_1$ and r$_2$ contain 1.

Question: Is it possible that r$_1$=0 but r$_2$=0?
- _Sequential consistency:_   _No_

- _Register renaming: Yes because it removes anti-dependencies_

Example 7: Register Renaming

Process 1	*Process 2*
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_2$;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Register renaming will eliminate this edge

Initially both r$_1$ and r$_2$ contain 1.

Question: Is it possible that r$_1$=0 but r$_2$=0?

- *Sequential consistency:* *No*

- *Register renaming:* *Yes because it removes anti-dependencies*

Example 7: Register Renaming

Process 1	Process 2	Register renaming will eliminate this edge
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_2$;	
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);	

Initially both r$_1$ and r$_2$ contain 1.

Question: Is it possible that r$_1$=0 but r$_2$=0?
- *Sequential consistency: No*
- *Register renaming: Yes because it removes anti-dependencies*

Example 7: Register Renaming

Register renaming will eliminate this edge

Process 1	*Process 2*
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_2$;
Load r$_1$, (flag$_2$);	Load r$_2$, (flag$_1$);

Initially both r$_1$ and r$_2$ contain 1.

Question: Is it possible that r$_1$=0 but r$_2$=0?
- *Sequential consistency: No*
- *Register renaming: Yes because it removes anti-dependencies*

Example 8: Speculative Execution

	Process 1		Process 2
	Store (a), 1;	L:	Load r_1, (flag);
	Store (flag), 1;		if r_1 == 0 goto L;
			Load r_2, (a);

Example 8: Speculative Execution

Process 1	_Process 2_
Store (a), 1;	L: Load r_1, (flag);
Store (flag), 1;	if r_1 == 0 goto L;
	Load r_2, (a);

Question: Is it possible that $r_1=1$ but $r_2=0$?

Example 8: Speculative Execution

Process 1	_Process 2_
Store (a), 1;	L: Load r_1, (flag);
Store (flag), 1;	if r_1 == 0 goto L;
	Load r_2, (a);

Question: Is it possible that $r_1=1$ but $r_2=0$?
- _Sequential consistency: No_

Example 8: Speculative Execution

Process 1	Process 2

Store (a), 1;   L: Load r_1, (flag);

Store (flag), 1;   if r_1 == 0 goto L;

                                     Load r_2, (a);

Question: Is it possible that $r_1=1$ but $r_2=0$?

- *Sequential consistency: No*

- *With speculative loads: Yes even if the stores are ordered*

Example 9: Address Speculation

	Process 1	*Process 2*
	Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_3$;
	Load r$_2$, (flag$_2$);	Load r$_4$, (flag$_1$);

Initially both r_1 and r_3 contain 1.

Flag$_1$ and flag$_2$ are registers
pointing at memory locations

Example 9: Address Speculation

	Process 1	*Process 2*
	Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_3$;
	Load r$_2$, (flag$_2$);	Load r$_4$, (flag$_1$);

Initially both r$_1$ and r$_3$ contain 1.

Question: Is it possible that r$_2$=0 but r$_4$=0?

Flag$_1$ and flag$_2$ are registers pointing at memory locations

Example 9: Address Speculation

Process 1	Process 2
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_3$;
Load r$_2$, (flag$_2$);	Load r$_4$, (flag$_1$);

Initially both r$_1$ and r$_3$ contain 1.

Question: Is it possible that r$_2$=0 but r$_4$=0?
- *Sequential consistency: No*
-

Flag$_1$ and flag$_2$ are registers
pointing at memory locations

Example 9: Address Speculation

Process 1	Process 2
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_3$;
Load r$_2$, (flag$_2$);	Load r$_4$, (flag$_1$);

Initially both r$_1$ and r$_3$ contain 1.
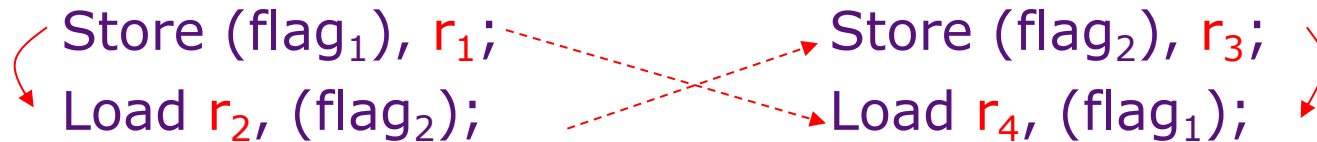
Question: Is it possible that r$_2$=0 but r$_4$=0?
- *Sequential consistency: No*
- *Address speculation: Yes because it removes the dependencies between the stores and loads*

Flag$_1$ and flag$_2$ are registers pointing at memory locations

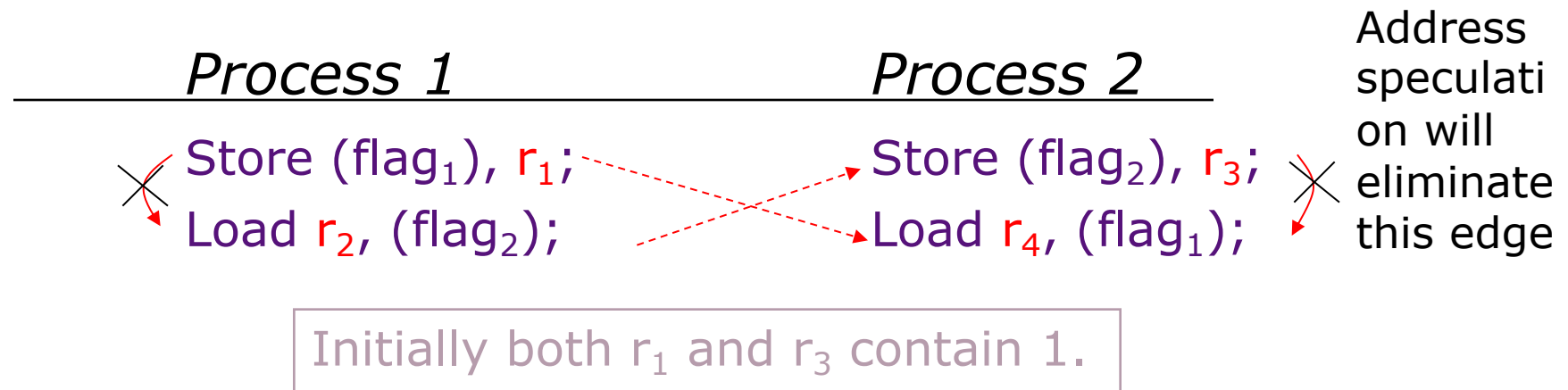# Example 9: Address Speculation

Process 1	_Process 2_
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_3$;
Load r$_2$, (flag$_2$);	Load r$_4$, (flag$_1$);

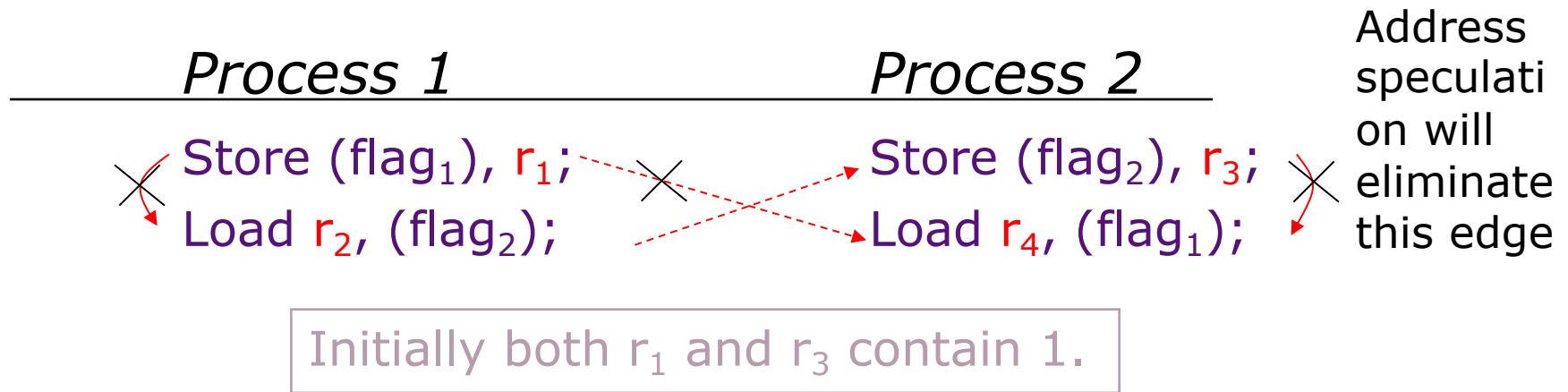Initially both r$_1$ and r$_3$ contain 1.

Question: Is it possible that r$_2$=0 but r$_4$=0?
- _Sequential consistency: No_
- _Address speculation: Yes because it removes the dependencies between the stores and loads_

Flag$_1$ and flag$_2$ are registers pointing at memory locations

Example 9: Address Speculation

Process 1	_Process 2_
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_3$;
Load r$_2$, (flag$_2$);	Load r$_4$, (flag$_1$);

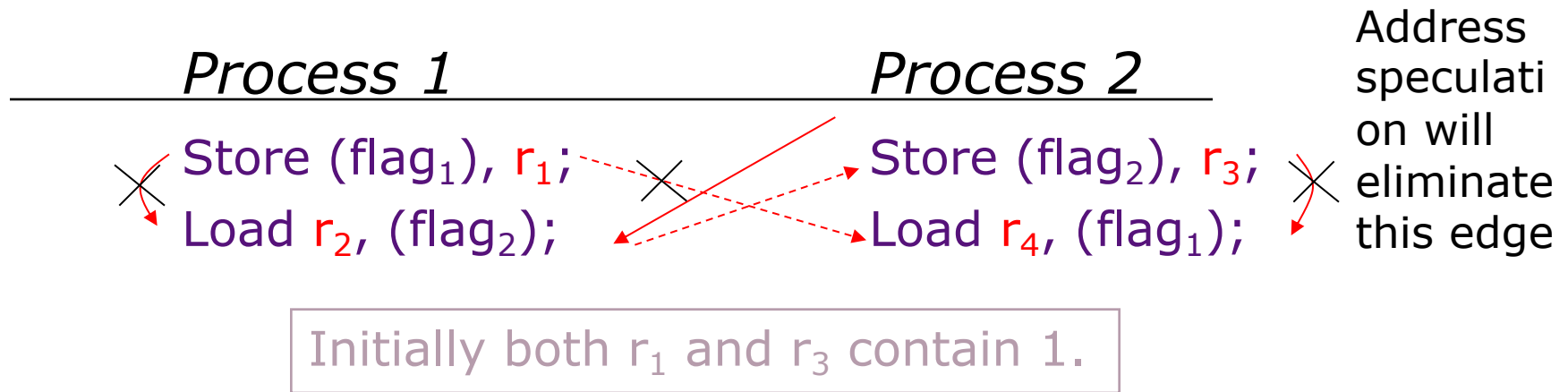Initially both r$_1$ and r$_3$ contain 1.

Question: Is it possible that r$_2$=0 but r$_4$=0?
- _Sequential consistency: No_
- _Address speculation: Yes because it removes the dependencies between the stores and loads_

Flag$_1$ and flag$_2$ are registers pointing at memory locations

Example 9: Address Speculation

Process 1	*Process 2*
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_3$;
Load r$_2$, (flag$_2$);	Load r$_4$, (flag$_1$);

Initially both r$_1$ and r$_3$ contain 1.

Question: Is it possible that r$_2$=0 but r$_4$=0?
- *Sequential consistency: No*
- *Address speculation: Yes because it removes the dependencies between the stores and loads*

Flag$_1$ and flag$_2$ are registers pointing at memory locations

Example 9: Address Speculation

	Process 1		*Process 2*

Store (flag$_1$), r$_1$;   Store (flag$_2$), r$_3$;

Load r$_2$, (flag$_2$);   Load r$_4$, (flag$_1$);

Initially both r$_1$ and r$_3$ contain 1.

Question: Is it possible that r$_2$=0 but r$_4$=0?
- *Sequential consistency: No*
- *Address speculation: Yes because it removes the dependencies between the stores and loads*

Flag$_1$ and flag$_2$ are registers pointing at memory locations

Example 9: Address Speculation

	Process 1	*Process 2*	Address speculation will eliminate this edge
	Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_3$;	
	Load r$_2$, (flag$_2$);	Load r$_4$, (flag$_1$);	

Initially both r$_1$ and r$_3$ contain 1.

Question: Is it possible that r$_2$=0 but r$_4$=0?
- *Sequential consistency: No*
- *Address speculation: Yes because it removes the dependencies between the stores and loads*

Flag$_1$ and flag$_2$ are registers pointing at memory locations

Example 9: Address Speculation

Process 1	Process 2
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_3$;
Load r$_2$, (flag$_2$);	Load r$_4$, (flag$_1$);

Initially both r$_1$ and r$_3$ contain 1.

Question: Is it possible that r$_2$=0 but r$_4$=0?
- *Sequential consistency: No*
- *Address speculation: Yes because it removes the dependencies between the stores and loads*

Flag$_1$ and flag$_2$ are registers pointing at memory locations

Example 9: Address Speculation

Address speculation will eliminate this edge

Process 1	Process 2
Store (flag$_1$), r$_1$;	Store (flag$_2$), r$_3$;
Load r$_2$, (flag$_2$);	Load r$_4$, (flag$_1$);

Initially both r$_1$ and r$_3$ contain 1.

Question: Is it possible that r$_2$=0 but r$_4$=0?
- *Sequential consistency: No*
- *Address speculation: Yes because it removes the dependencies between the stores and loads*

Flag$_1$ and flag$_2$ are registers pointing at memory locations

Weaker Memory Models & Memory Fence Instructions

- Architectures with weaker memory models provide memory fence instructions to prevent otherwise permitted re-orderings of loads and stores

Store (a_1), r2;

Load r1, (a_2);

> The Load and Store can be reordered if a_1 =/= a_2. Insertion of $Fence_{wr}$ will disallow this reordering

Similarly:

Weaker Memory Models & Memory Fence Instructions

- Architectures with weaker memory models provide memory fence instructions to prevent otherwise permitted re-orderings of loads and stores

Store (a_1), r2;

Fence$_{wr}$

Load r1, (a_2);

> **The Load and Store can be reordered if a_1 =/= a_2. Insertion of Fence$_{wr}$ will disallow this reordering**

Similarly:

Weaker Memory Models & Memory Fence Instructions

- Architectures with weaker memory models provide memory fence instructions to prevent otherwise permitted re-orderings of loads and stores

Store (a_1), r2;

Fence$_{wr}$

Load r1, (a_2);

> **The Load and Store can be reordered if $a_1 =/= a_2$. Insertion of Fence$_{wr}$ will disallow this reordering**

Similarly: Fence$_{rr}$;   Fence$_{rw}$;   Fence$_{ww}$;

Weaker Memory Models & Memory Fence Instructions

- Architectures with weaker memory models provide memory fence instructions to prevent otherwise permitted re-orderings of loads and stores

Store (a_1), r2;
Fence$_{wr}$
Load r1, (a_2);

> The Load and Store can be reordered if $a_1 =/= a_2$. Insertion of Fence$_{wr}$ will disallow this reordering

Similarly: Fence$_{rr}$;   Fence$_{rw}$;   Fence$_{ww}$;

SUN's Sparc: MEMBAR;
     MEMBARRR; MEMBARRW; MEMBARWR; MEMBARWW
PowerPC: Sync; EIEIO

Enforcing Ordering using Fences

<div>

Processor 1

Store (a),10;

Store (flag),1;

</div>

<div>

Processor 2

L: Load r_1, (flag);

    if r_1 == 0 goto L;

    Load r_2, (a);

</div>

Enforcing Ordering using Fences

Processor 1
Store (a),10;
Store (flag),1;

Processor 2
L: Load r_1, (flag);
    if r_1 == 0 goto L;
    Load r_2, (a);

Processor 1
Store (a),10;
Fence$_{ww}$;
Store (flag),1;

Processor 2
L: Load r_1, (flag);
    if r_1 == 0 goto L;
Fence$_{rr}$;
Load r_2, (a);

Weaker (Relaxed) Memory Models

Weaker (Relaxed) Memory Models

Alpha, Sparc PowerPC, ...

Store is globally performed

TSO, PSO, RMO, ...

Write-buffers

RMO=WO?

SMP, DSM, CMP

Weaker (Relaxed) Memory Models

Alpha, Sparc PowerPC, ...

Store is globally performed

TSO, PSO, RMO, ...

Write-buffers

RMO=WO?

SMP, DSM, CMP

- Hard to understand and remember

Weaker (Relaxed) Memory Models

Alpha, Sparc PowerPC, ...

Store is globally performed

TSO, PSO, RMO, ...

Write-buffers

RMO=WO?

SMP, DSM, CMP

- Hard to understand and remember
- Unstable - *Modèle de l'année*

Weaker (Relaxed) Memory Models

- Hard to understand and remember
- Unstable - *Modèle de l'année*
- Abandon weaker memory models in favor of implementing SC

Implementing SC

1. The memory operations of each individual processor appear to all processors in the order the requests are made to the memory

   – *Provided by cache coherence*, which ensures that all processors observe the same order of loads and stores to an address

2. Any execution is the same as if the operations of all the processors were executed in some sequential order

   – Provided by enforcing a **dependence** between each memory operation and the following one

SC Data Dependence

SC Data Dependence

- *Stall*
  - *Use in-order execution and blocking caches*
    - *Cache coherence plus allowing a processor to have only one request in flight at a time will provide SC*

SC Data Dependence

- *Stall*
  - *Use in-order execution and blocking caches*
    - *Cache coherence plus allowing a processor to have only one request in flight at a time will provide SC*

- *Change architecture ⇒ Relaxed memory models*
  - *Use OOO and non-blocking caches*
    - *Cache coherence and allowing multiple concurrent requests (to different addresses) gives high performance*
    - *Add fence operations to force ordering when needed*

SC Data Dependence

- *Stall*
  - *– Use in-order execution and blocking caches*
    - *Cache coherence plus allowing a processor to have only one request in flight at a time will provide SC*

- *Change architecture $\Rightarrow$ Relaxed memory models*
  - *– Use OOO and non-blocking caches*
    - *Cache coherence and allowing multiple concurrent requests (to different addresses) gives high performance*
    - *Add fence operations to force ordering when needed*

- *Speculate…*

Sequential Consistency Speculation

- Local load-store ordering uses standard OOO mechanism

- Globally <u>non-speculative</u> stores
  - Stores execute at commit -> stores are in-order!

Sequential Consistency Speculation

- Local load-store ordering uses standard OOO mechanism

- Globally <u>non-speculative</u> stores
  - Stores execute at commit -> stores are in-order!

- Globally <u>speculative</u> loads
  - **Guess** at issue that the memory location used by a load will not change between issue and commit of the instruction
    - this is equivalent to loads happening in-order at commit

  - **Check** at commit by remembering all loads addresses starting at issue and watching for writes to that location.

  - **Data Management** for rollback relies on the basic out-of-order speculative data management used for uni-processor rollback and instruction re-execution.

SC Speculative Behavior

CPU A   CPU B

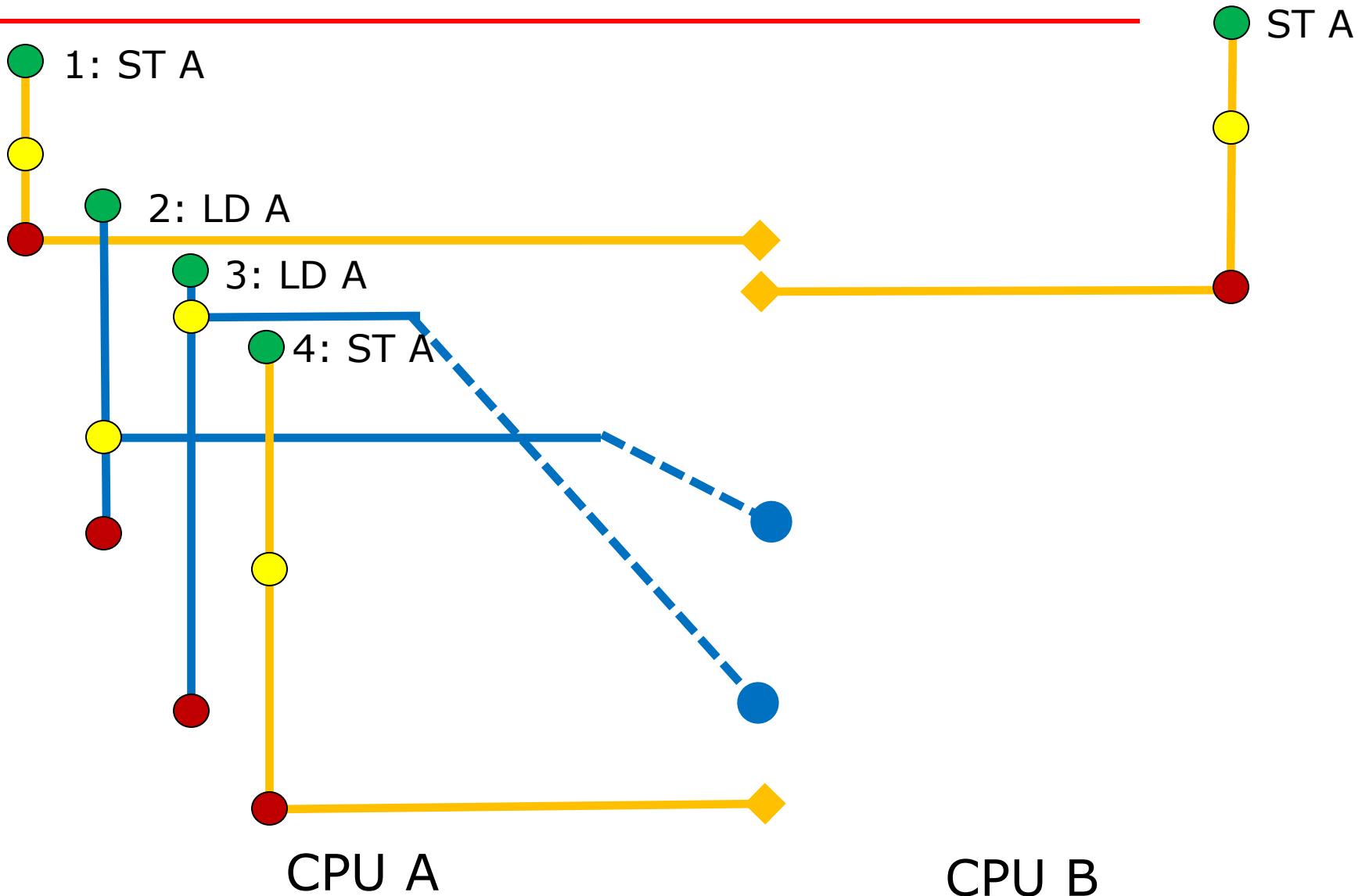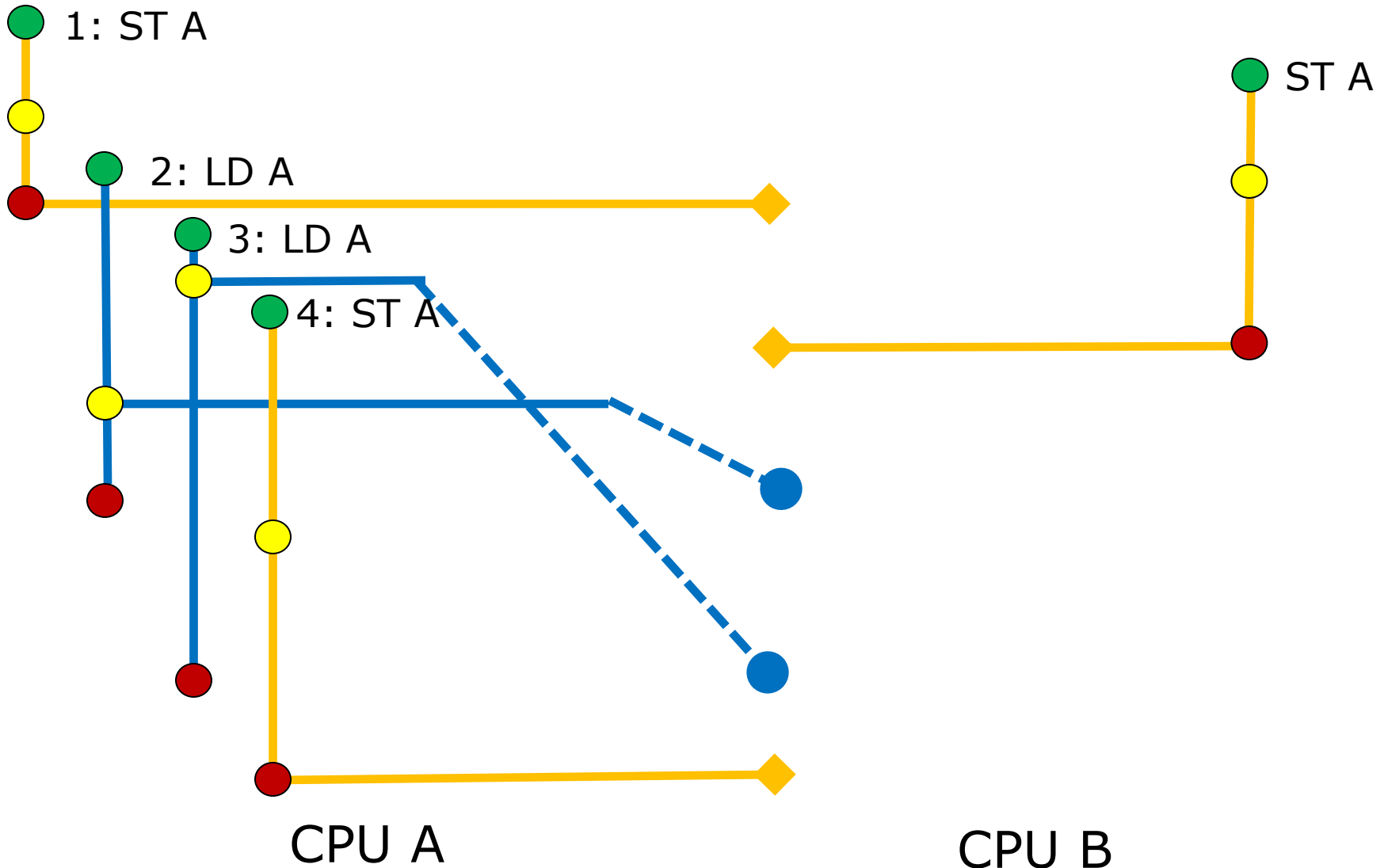# SC Speculative Behavior
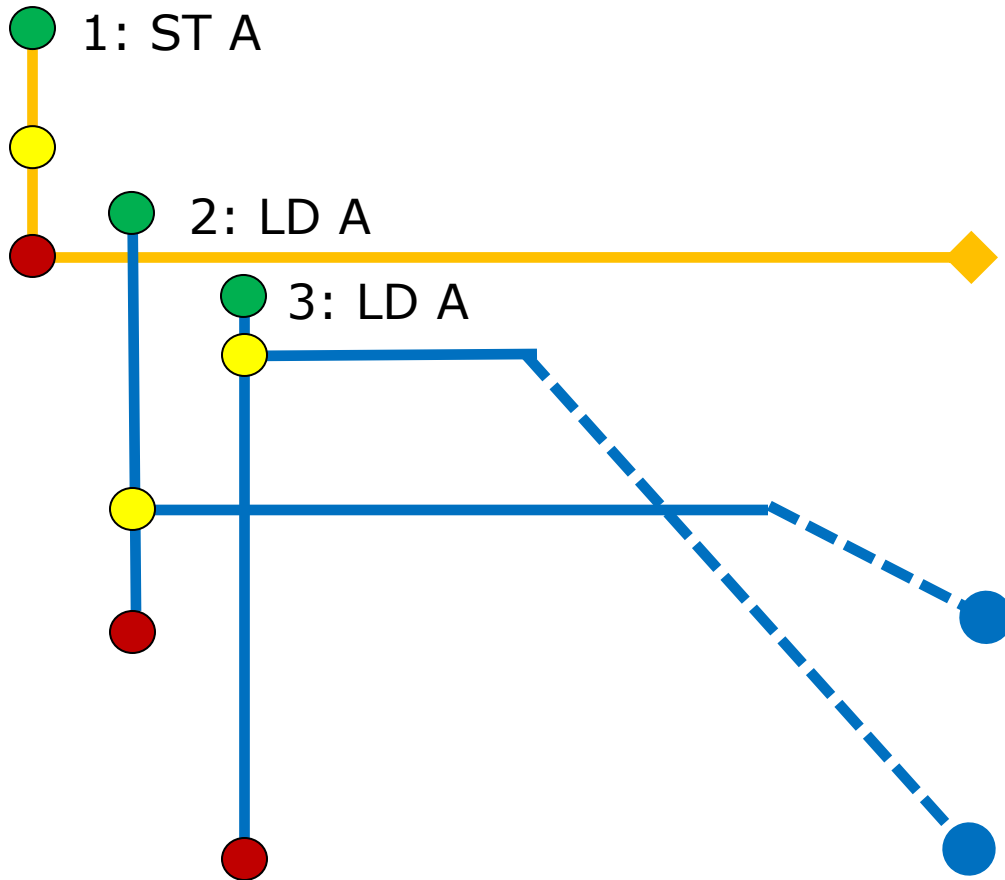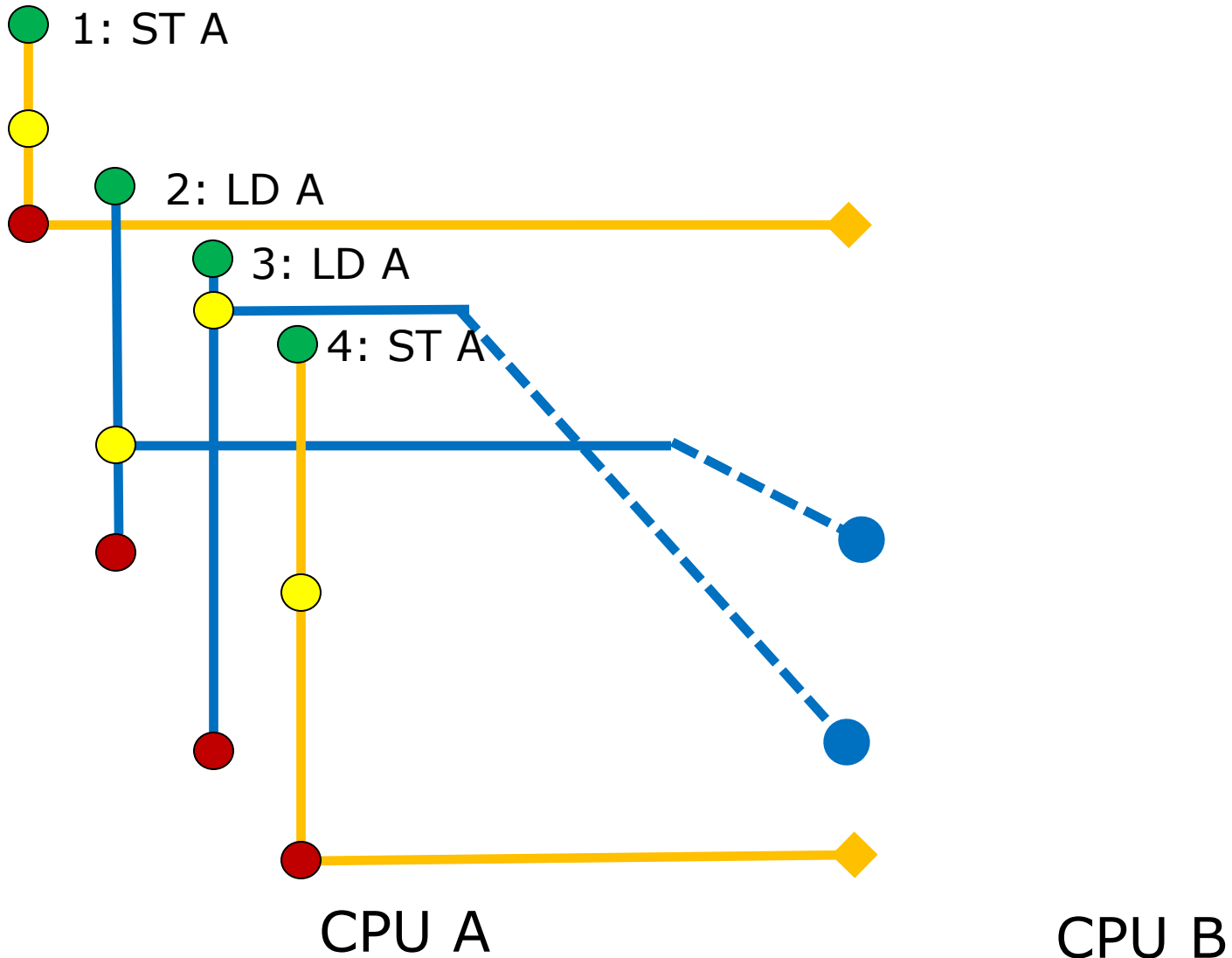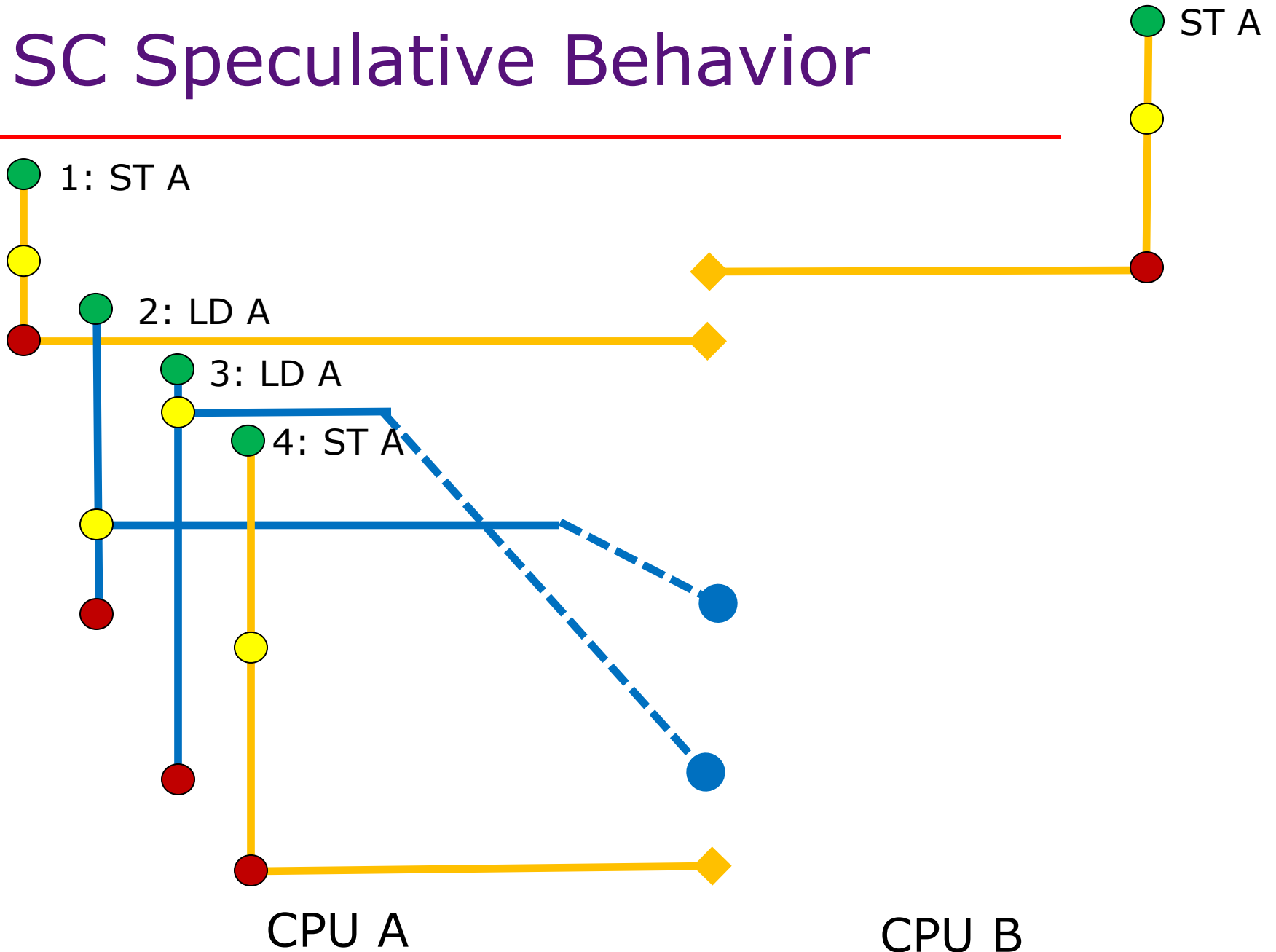
1: ST A

CPU A   CPU B

SC Speculative Behavior

1: ST A

2: LD A

CPU A   CPU B

SC Speculative Behavior

1: ST A

2: LD A

CPU A   CPU B

SC Speculative Behavior

1: ST A

2: LD A

3: LD A

CPU A

CPU B

SC Speculative Behavior

1: ST A

2: LD A

3: LD A

CPU A   CPU B

MIT 6.5900 Fall 2023

SC Speculative Behavior

1: ST A

2: LD A

3: LD A

4: ST A

CPU A

CPU B

SC Speculative Behavior

ST A

1: ST A

2: LD A

3: LD A

4: ST A

CPU A

CPU B

SC Speculative Behavior

1: ST A

2: LD A

3: LD A

4: ST A

ST A

CPU A

CPU B

SC Speculative Behavior

1: ST A

2: LD A

3: LD A

4: ST A

ST A

CPU A

CPU B

SC Speculative Behavior

1: ST A

2: LD A

3: LD A

4: ST A

ST A

CPU A

CPU B

SC Speculative Behavior

1: ST A

2: LD A

3: LD A

4: ST A

ST A

CPU A

CPU B

SC Speculative Behavior

1: ST A

2: LD A

3: LD A

4: ST A

ST A

CPU A

CPU B

SC Speculative Behavior

1: ST A

2: LD A

3: LD A

4: ST A

CPU A

CPU B
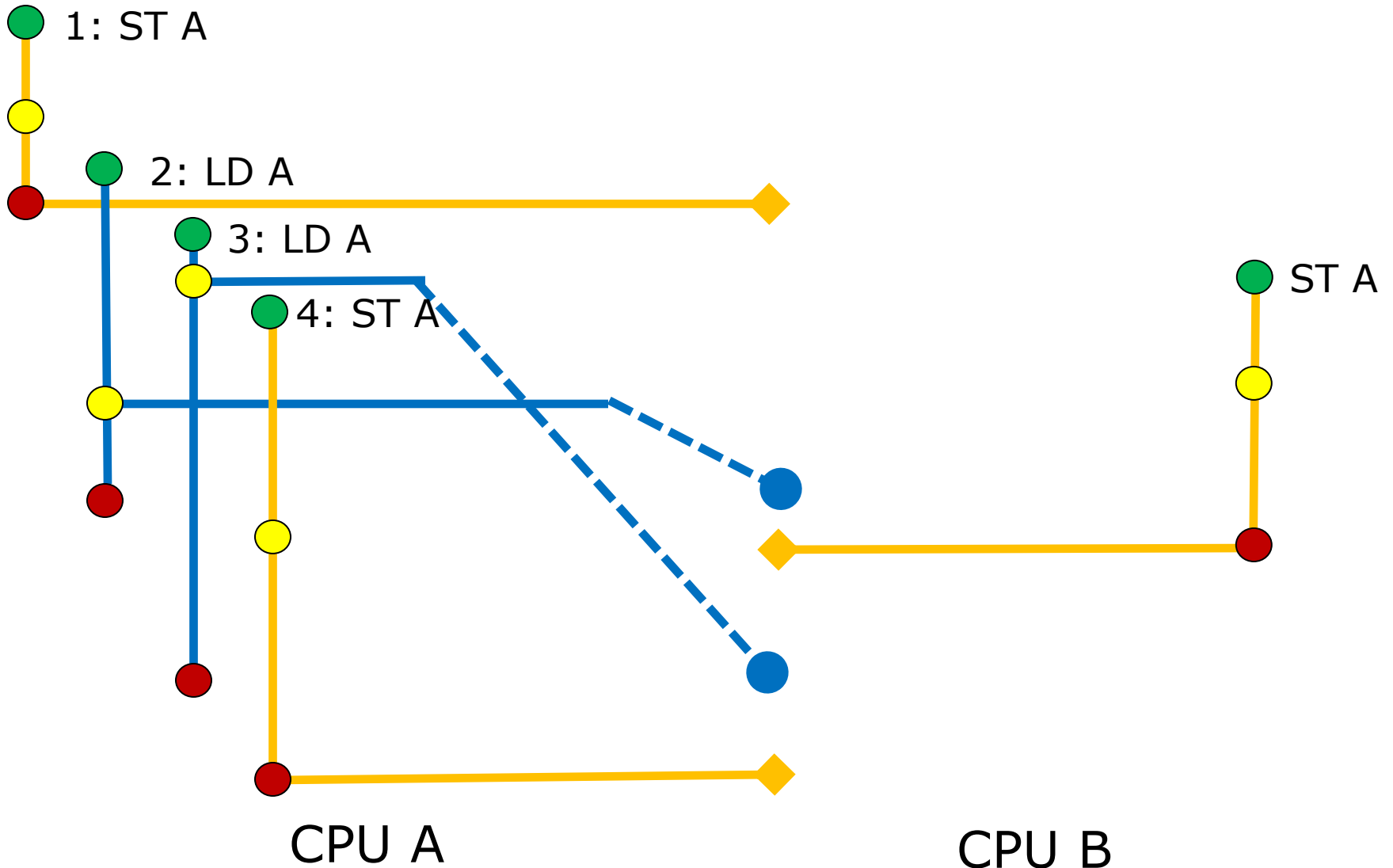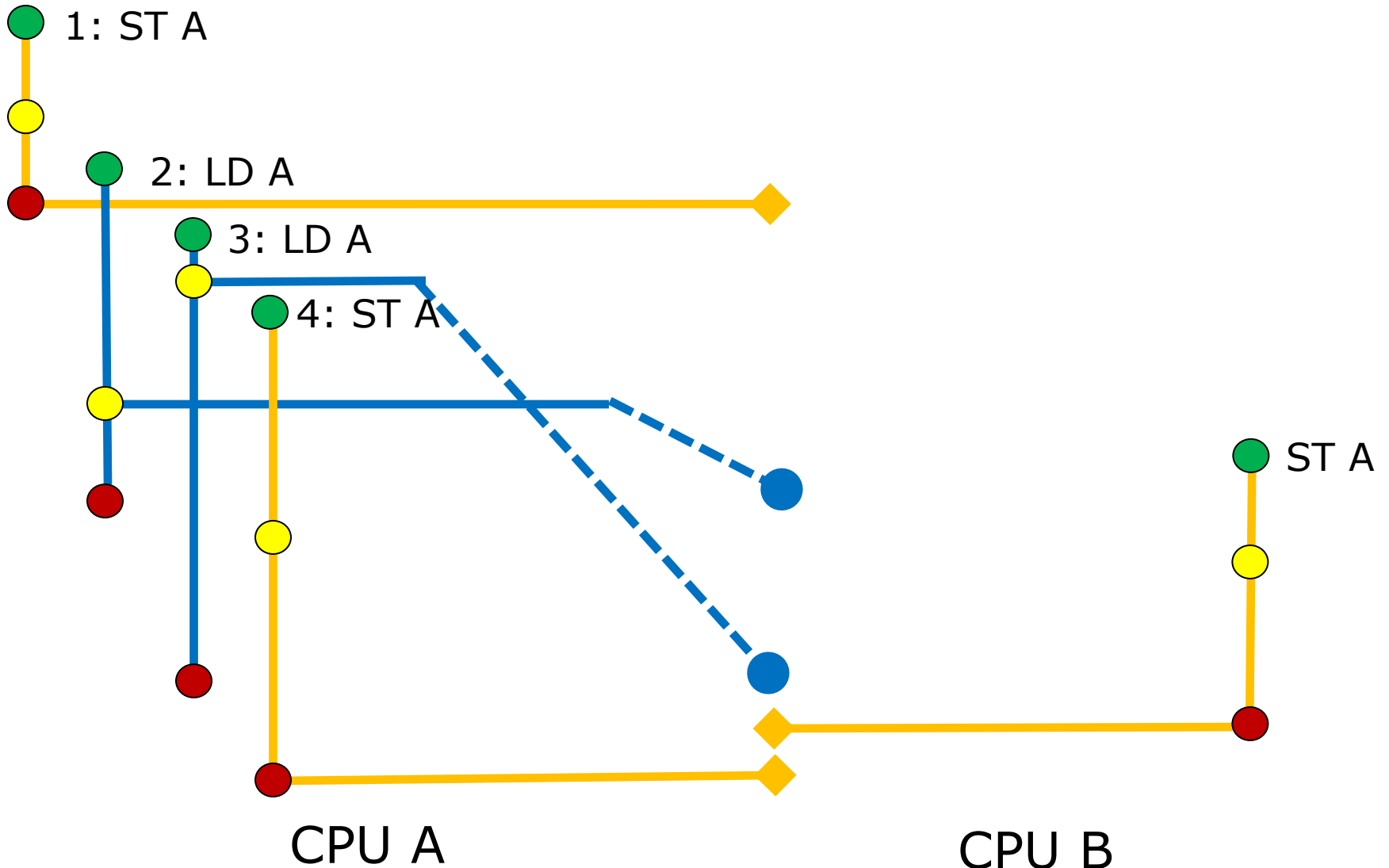
Properly Synchronized Programs

- Very few programmers do programming that relies on SC; instead, they use higher-level synchronization primitives

  – locks, semaphores, monitors, atomic transactions

Properly Synchronized Programs

- Very few programmers do programming that relies on SC; instead, they use higher-level synchronization primitives

  - locks, semaphores, monitors, atomic transactions

- A "properly synchronized program" is one where each shared writable variable is protected (say, by a lock) so that there is no race in updating the variable

  - There is still race to get the lock
  - There is no way to check if a program is properly synchronized

- For properly synchronized programs, instruction reordering does not matter as long as updated values are committed before leaving a locked region

Release Consistency
[Garachorloo 1990]

- Only care about inter-processor memory ordering at thread synchronization points, not in between
- Can treat all synchronization instructions as the only ordering points

> …
>
> Acquire(lock) // All following loads
>   // get most recent written values
>
>
> … Read and write shared data ..
>
>
> Release(lock) // All preceding writes are globally visible
>   // before lock is freed.
>
> …

Takeaways

- SC is too low level a programming model. High-level programming should be based on critical sections & locks, atomic transactions, monitors, ...

- High-level parallel programming should be oblivious of memory model issues
  - Programmer should not be affected by changes in the memory model

- ISA definition for Load, Store, Memory Fence, synchronization instructions should
  - Be precise
  - Permit maximum flexibility in hardware implementation
  - Permit efficient implementation of high-level parallel constructs

Thank you!

Next Lecture:
On-Chip Networks