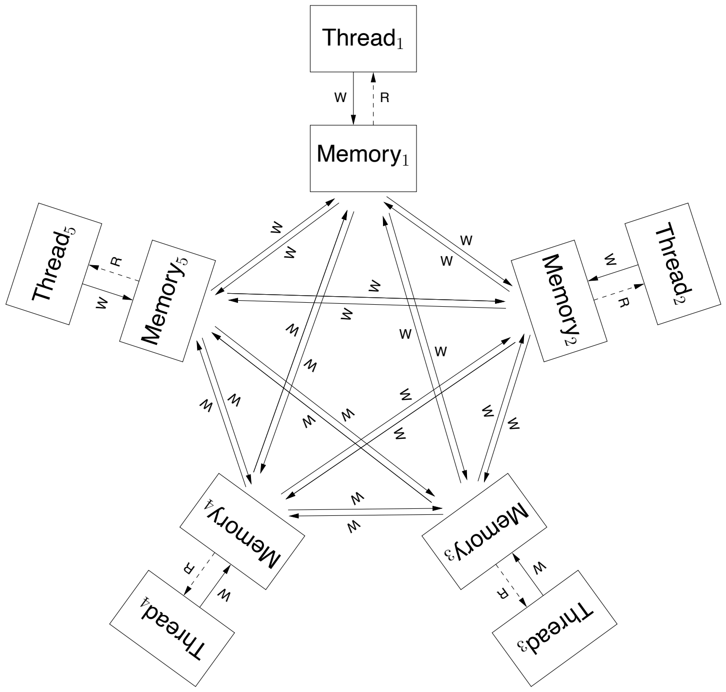
Principles of Concurrency

Lecture 11
Memory Models: Power and ARM

The IBM Power Memory Model

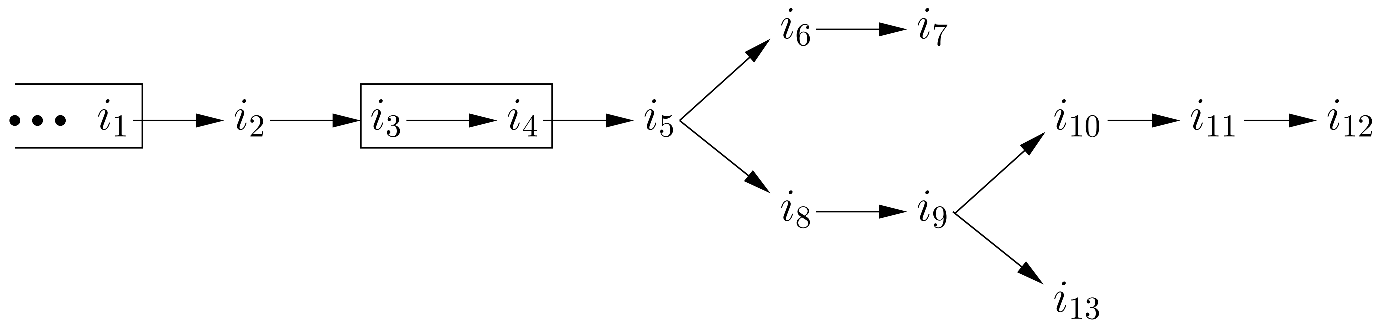
Highly relaxed, significantly more behaviors than possible under TSO

- Hardware threads can each perform reads and writes out-of-order, or even speculatively
- Arbitrary local reordering is allowed
- Does not support multi-copy atomicity: a write issued by a processor is not guaranteed to be visible to all other threads at the same time



Operational Model

Each thread, at each step in time, maintains a tree of committed and in-flight instruction instances



Instruction i5 and i9 are branches for which the thread has multiple possible successors

When a branch is committed, all alternative paths are discarded

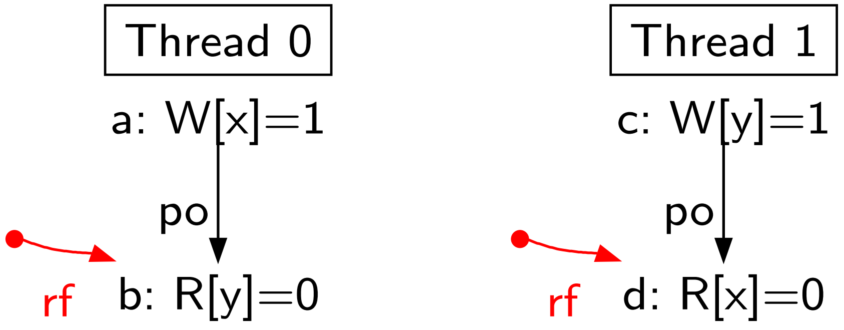
Actions become committed when the relevant address and value are determined ("satisfied" for reads, "committed" for writes)

Test Execution Diagrams

Thread 0	Thread 1

x=1
r1=y
	y=1
	r2=x

Initial shared state: x=0 and y=0	
Allowed final state: r1=0 and r2=0	



Relations:

- `po`: **program order**
- `rf`: **reads-from**

https://www.cl.cam.ac.uk/~pes20/ppcmem/

Message-Passing

MP-loop Pseudocode

Thread 0	Thread 1

x=1 // write data
y=1 // write flag
	while (y==0) {} // busy-wait for flag
	r2=x // read data

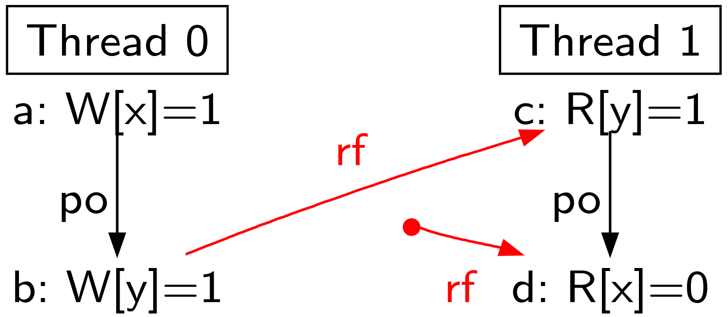
Initial state: $x=0 \wedge y=0$	
Forbidden?: Thread 1 register $r2 = 0$	

MP Pseudocode

Thread 0	Thread 1

x=1
y=1
	r1=y
	r2=x

Initial state: $x=0 \wedge y=0$	
Forbidden?: $1{:}r1=1 \wedge 1{:}r2=0$	



Allowed because (a) writes by Thread 0 are to distinct addresses and can be committed out-of-order; (b) reads performed by Thread 1 can be satisfied out-of-order

Is this outcome possible under SC?

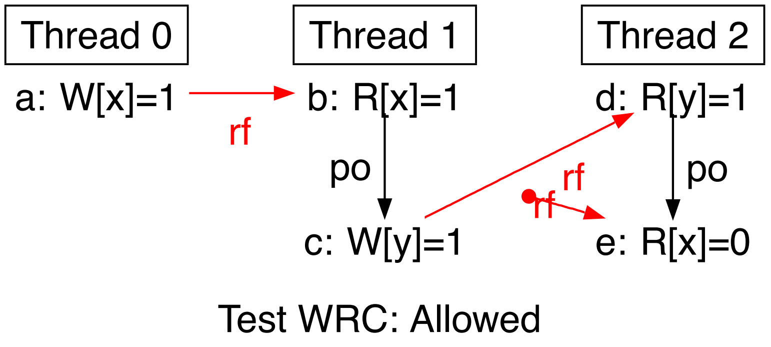
Iterated Message-Passing

WRC Pseudocode

Thread 0	Thread 1	Thread 2

x=1	r1=x
	y=1
		r2=y
		r3=x

Initial state: x=0 ∧ y=0		
Allowed: 1:r1=1 ∧ 2:r2=1 ∧ 2:r3=0		



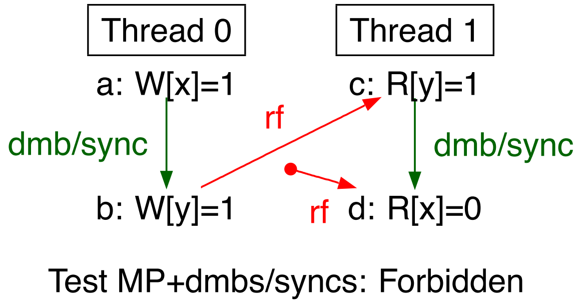
Thread 1 reads and writes to different addresses and can be thus reordered

MP+dmb/syncs Pseudocode

Thread 0	Thread 1

x=1
dmb/sync
y=1
	r1=y
	dmb/sync
	r2=x

Initial state: x=0 ∧ y=0	
Forbidden: 1:r1=1 ∧ 1:r2=0	



Memory fences (barriers) enforce ordering. Called "sync" on Power and "dmb" on ARM

maintains local ordering and fixes propagation order to other threads

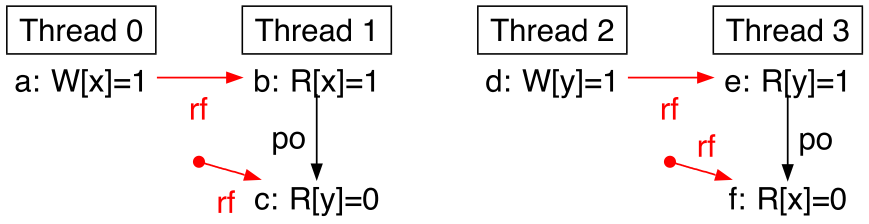
IRIW and Coherence Ordering

IRIW Pseudocode

Thread 0	Thread 1	Thread 2	Thread 3

x=1	r1=x
	r2=y
		y=1	r3=y
			r4=x

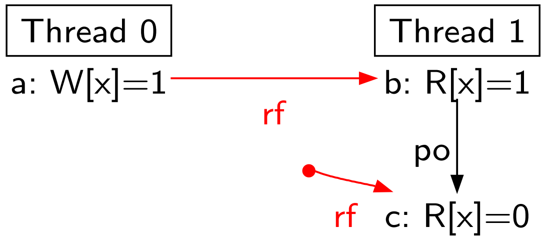
Initial state: x=0 $\wedge$ y=0			
Allowed: 1:r1=1 $\wedge$ 1:r2=0 $\wedge$ 3:r3=1 $\wedge$ 3:r4=0			



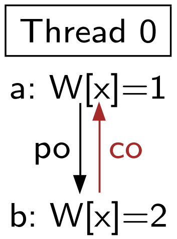
Test IRIW: Allowed

Writes can be propagated to different threads in different orders

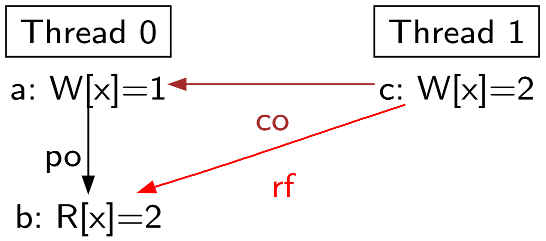
Coherence constraints



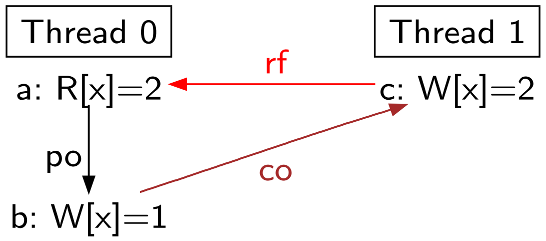
Test CoRR1 : Forbidden



Test CoWW : Forbidden



Test CoWR : Forbidden



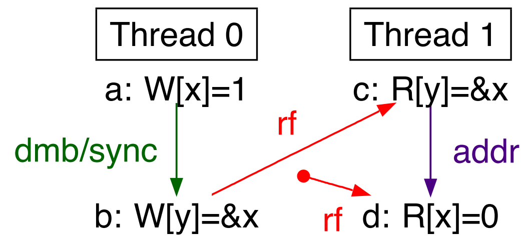
Test CoRW : Forbidden

Dependency Ordering

- Address dependency: value loaded by a read is used to compute the address used in a subsequent read or write

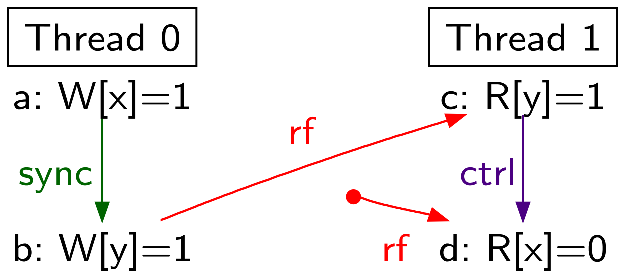
MP+dmb/sync+addr′ Pseudocode

Thread 0	Thread 1
x=1	r1=y
dmb/sync	
y=&x	r2=*r1
Initial state: x=0 ∧ y=0	
Forbidden: 1:r1=&x ∧ 1:r2=0	



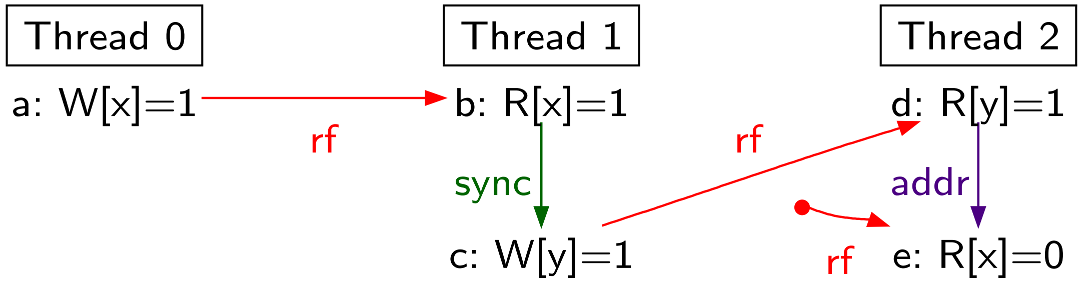
Test MP+dmb/sync+addr’: Forbidden

- Control dependency: value loaded by a read is used to compute the value of a conditional that is program-order-before another read or write

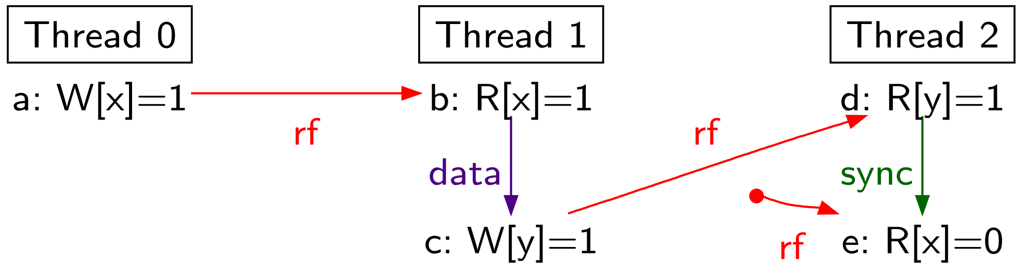


Test MP+sync+ctrl : Allowed

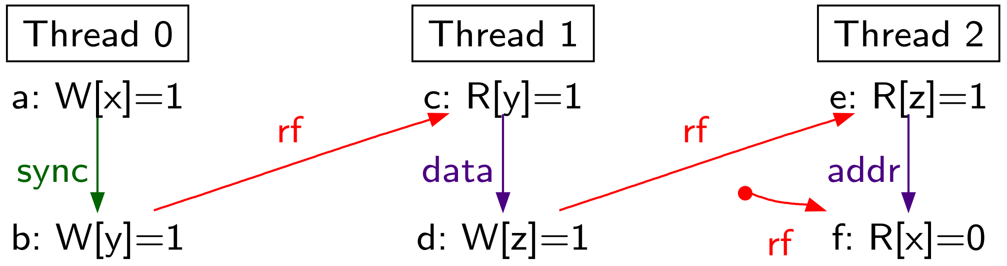
Cumulativity



Test WRC+sync+addr : Forbidden



Test WRC+data+sync : Allowed



Test ISA2+sync+data+addr : Forbidden