

Memory Consistency Models

Alistair Rendell

See “Shared Memory Consistency Models: A Tutorial”,

S.V. Adve and K. Gharachorloo

Chapter 8 pp 262-265 of Wilkinson and Allen

Parallel Programming

Dekkers Algorithm for Critical Sections

- “To write correct and efficient shared memory programs, programmers need a precise notion of how memory behaves with respect to read and write operations from multiple processors” (Adve and Gharachorloo)

```
initially flag1 = flag2 = 0
```

Process 0

```
flag1 = 1
if (flag2 == 0)
    critical section
```

Process 1

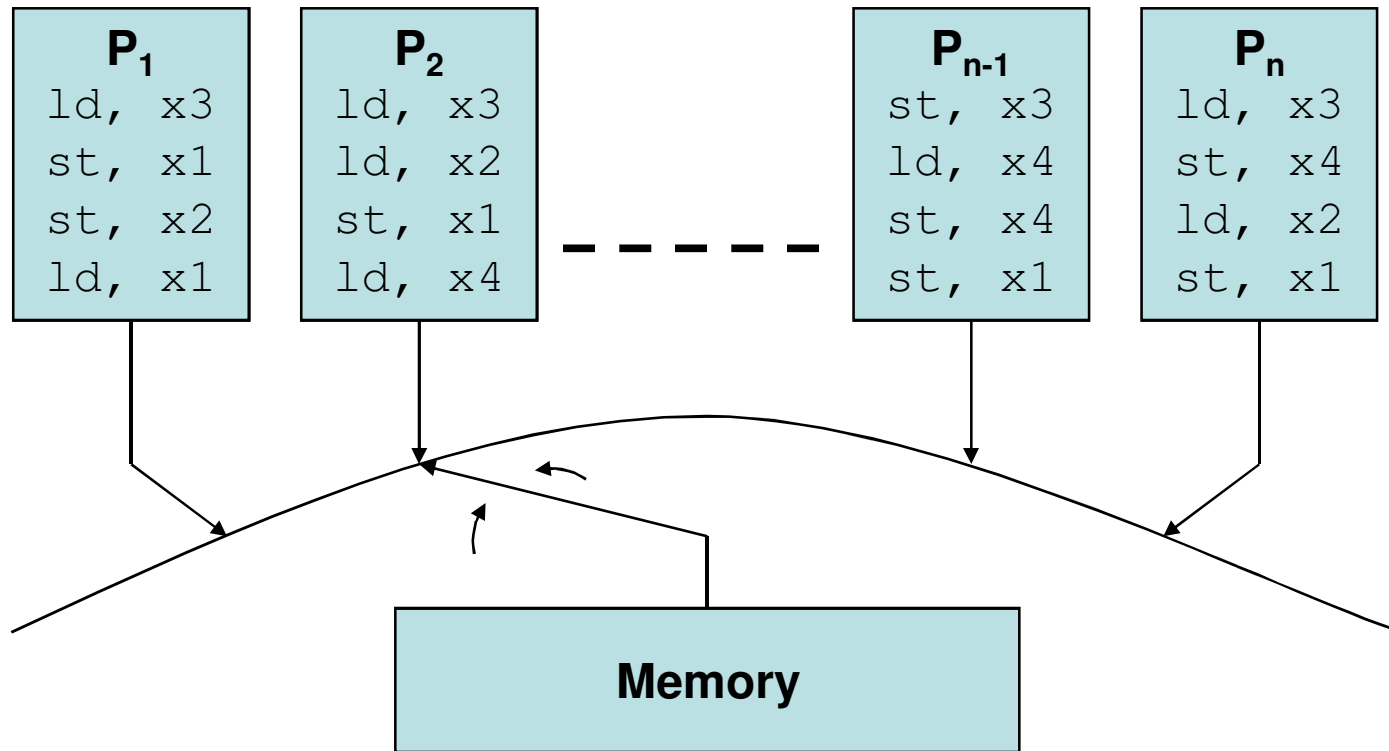
```
flag2 = 1
if (flag1 == 0)
    critical section
```

- *What is Dekkers algorithm for critical sections?*
 - *Go and look this up!*
- *What fundamental assumption(s) is the above code making?*

Sequential Consistency

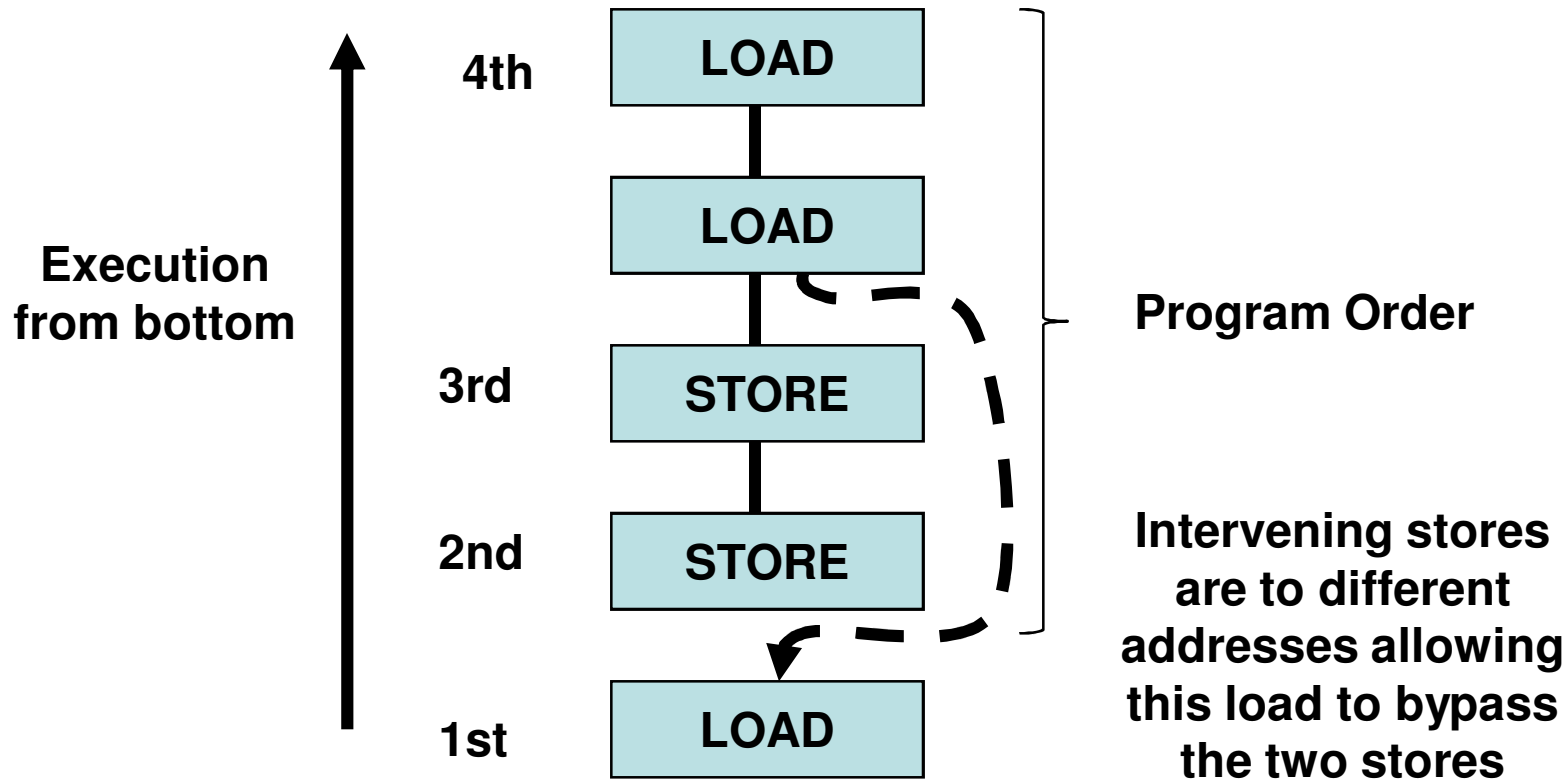
- Lamport's definition: [A multiprocessor system is *sequentially consistent* if] the result of any execution is the same as if the operation of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program
- Two aspects:
 - Maintaining program order among operations from individual processors
 - Maintaining a single sequential order among operations from all processors
- The latter aspect make it appear as if a memory operation executes *atomically* or *instantaneously* with respect to other memory operations

Programmers View of Sequential Consistency



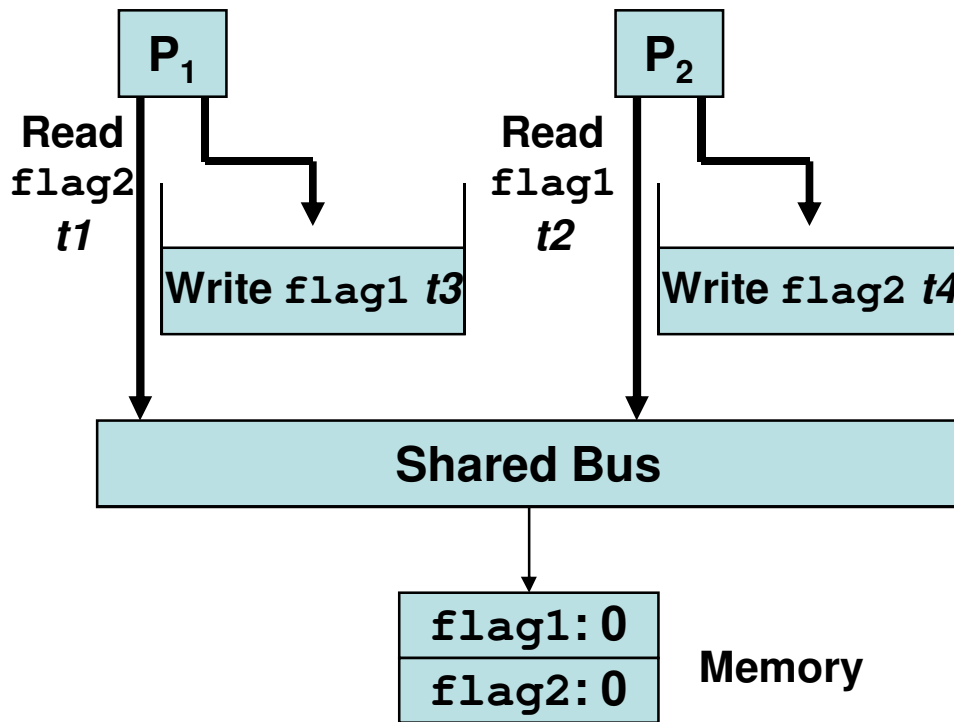
- Conceptually
 - There is a single global memory and a switch that connects an arbitrary processor to memory at any time step
 - Each process issues memory operations in program order and the switch provides the global serialization among all memory operations.

Processor Consistency



- Before a **LOAD** is allowed to perform wrt any processor, all previous **LOAD** accesses must be performed wrt everyone
- Before a **STORE** is allowed to perform wrt any processor, all previous **LOAD AND STORE** accesses must be performed wrt everyone.

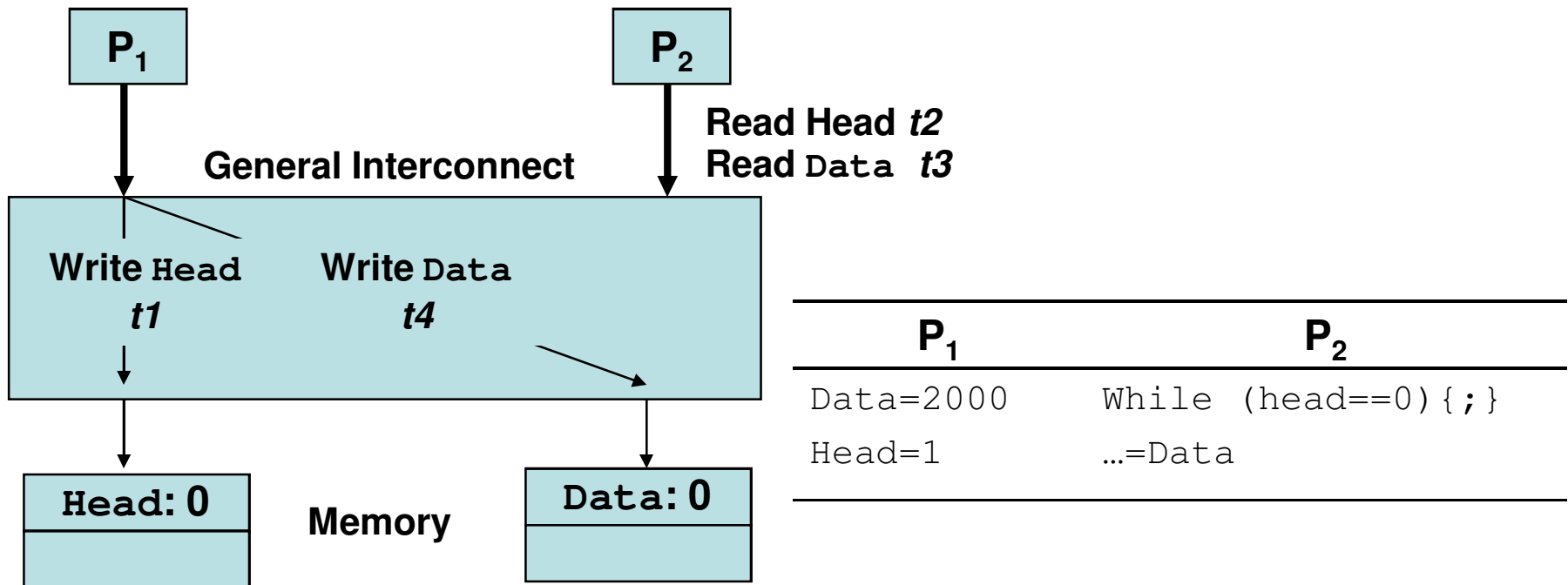
Breaking Sequential Consistency: Write Buffers



| P ₁ | P ₂ |
|----------------|----------------|
| flag1 = 1 | flag2 = 1 |
| if (flag2==0) | if (flag1==0) |
| Critical | Critical |
| Section | Section |

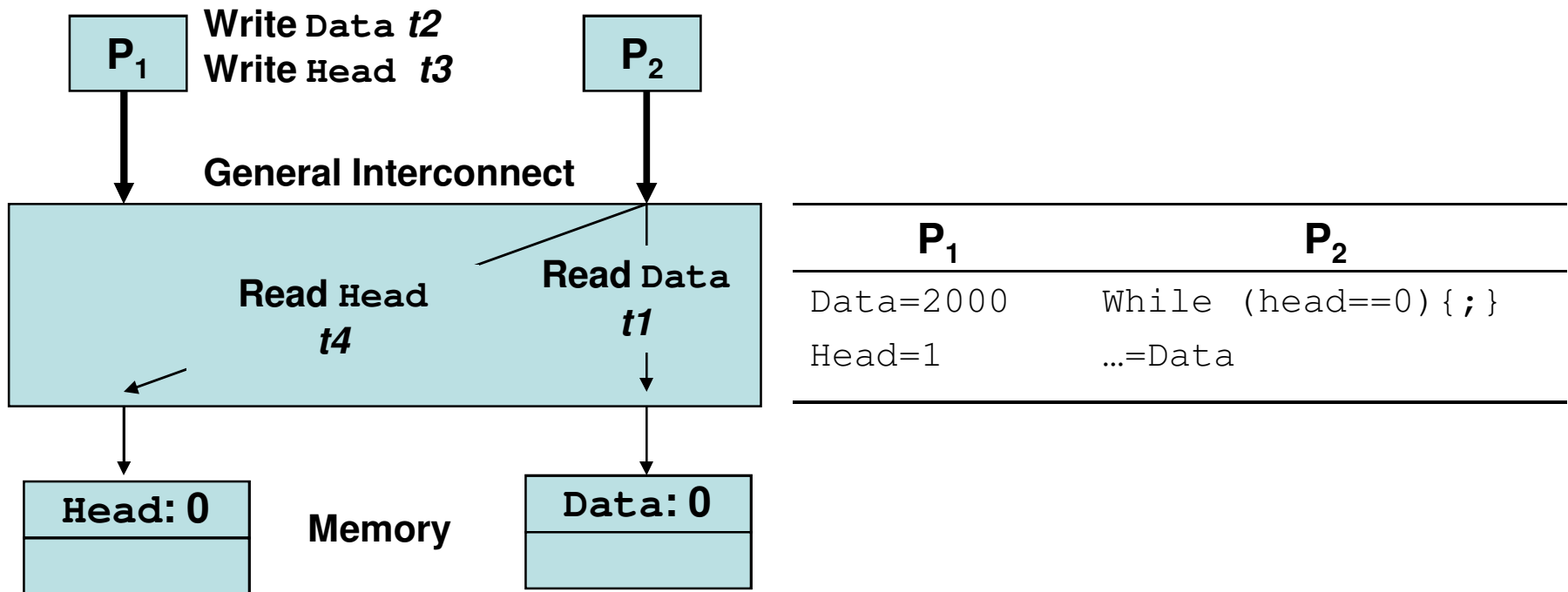
- Write buffer (very common)
 - Process inserts writes in write buffer and proceeds assuming it completes in due course
 - Subsequent reads bypass previous writes as long as read address does not overlap with those in write buffer

Breaking Sequential Consistency: Overlapped Writes



- General (non-bus) Interconnect with multiple memory modules
 - Different memory operations issued by same processor are serviced by different memory modules
 - Writes from P_1 are injected into the memory system in program order, but they may complete out of program order
 - Digital Alpha processors coalesced write to the same cache line in a write buffer, and could lead to similar effects

Breaking Sequential Consistency: Non-Blocking Reads



- Effect of Non-Blocking Read
 - Early RISC machines stalled for value of a read
 - Many current generation machines have capability to proceed past a read using non-blocking caches, speculative execution, and dynamic scheduling

Cache Coherency and Sequential Consistency

- One definition of Cache Coherency:
 - A write is eventually made visible to all processors
 - Writes to the same location appear to be seen in the same order by all processors
- Whereas sequential consistency requires:
 - Writes to ALL memory locations appear to be seen in the same order by all processors
 - Also require operations of a single processor appear to execute in program order

Multiple Caches and Writes

- Sequential consistency requires memory operations to appear atomic or instantaneous
 - But propagating changes to multiple cache copies is inherently non-atomic!
- One option is to prohibit a read from returning a newly written quantity until all cached copies have acknowledged receipt of the invalidation or update messages
 - Easier to ensure for invalidate protocol

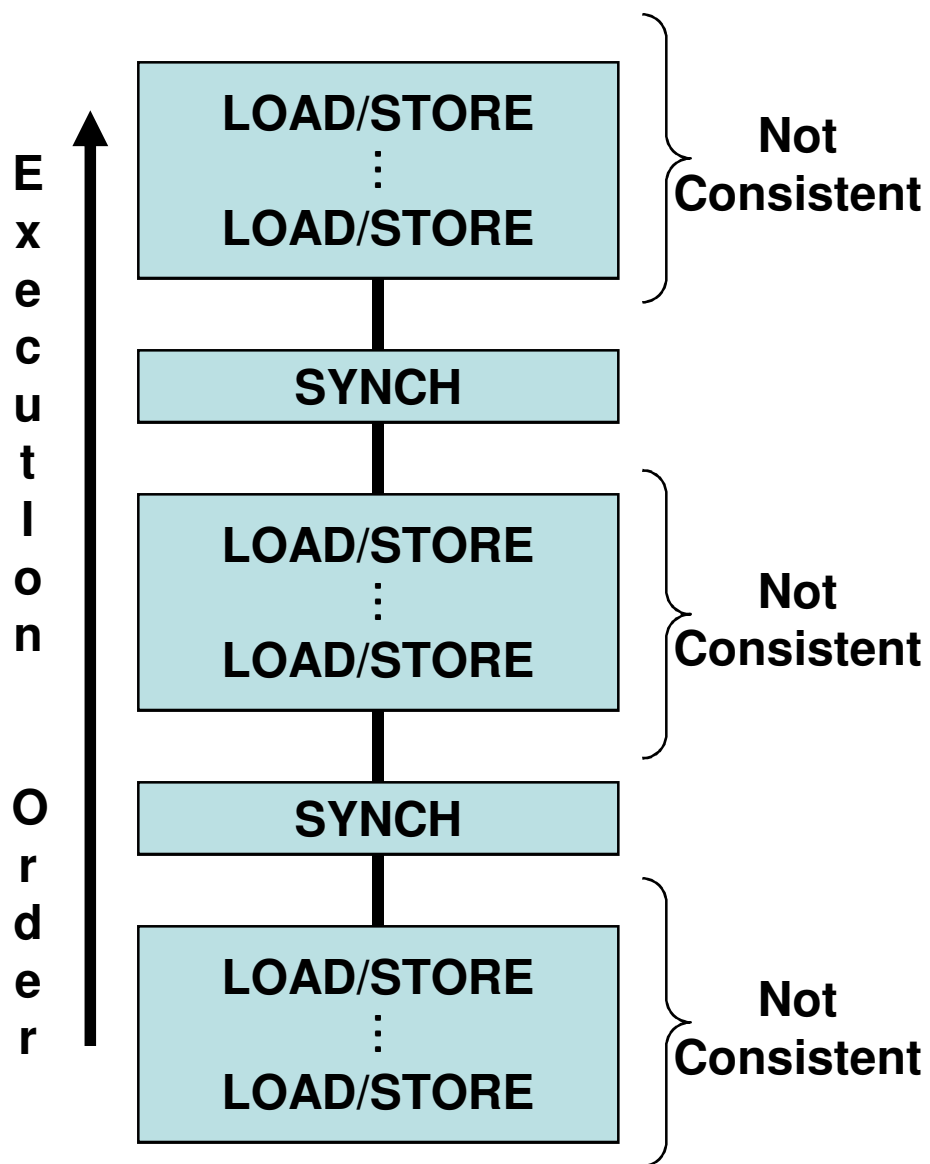
Relaxed Memory Consistency Models

- How they relax
 - The program order requirement
 - The write atomicity requirement
- Relaxing the write to read program order
 - This is store buffer
- Relaxing write to read and write to write
 - Writes to different locations can be pipelined or overlapped
- Relaxing everything!
 - Weak consistency
 - Release/Acquire consistency models
- Write atomicity
 - Allowing a read to return another processors write before all cached copies have been updated

Weak Consistency

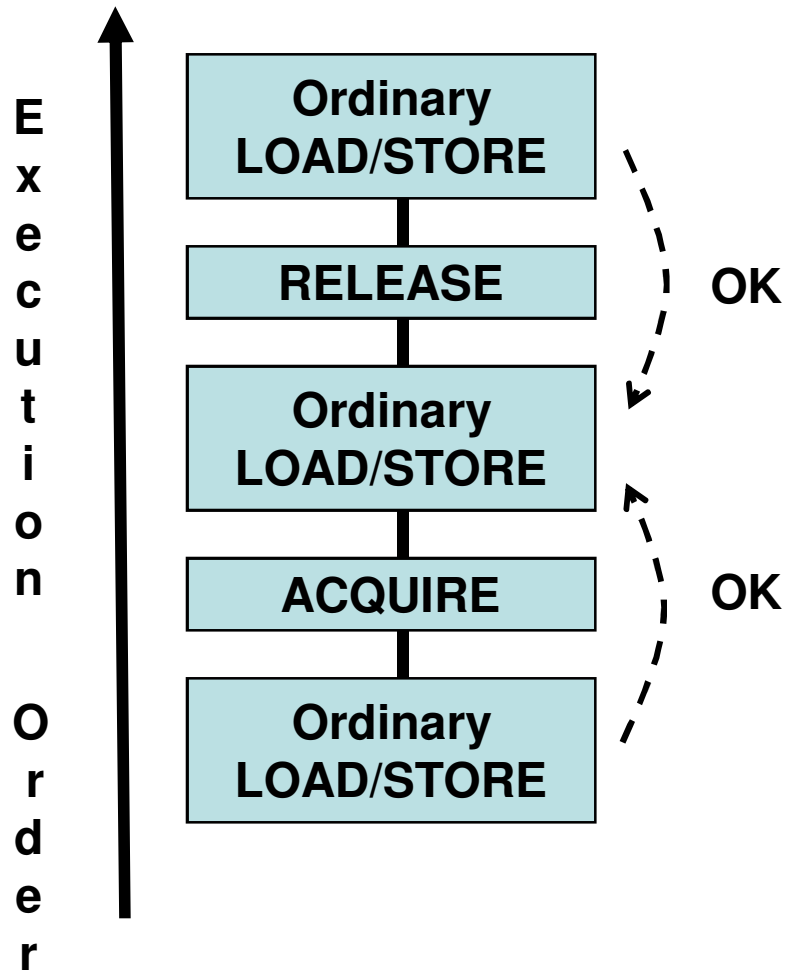
- Relies on the programmer having used critical sections to control access to shared variables
 - Within the critical section no other process can rely on that data structure being consistent until the critical section is exited
- We need to distinguish critical points when the programmer enters or leaves a critical section
 - Distinguish standard load/stores from synchronization accesses

Weak Consistency



- Before an ordinary LOAD/STORE is allowed to perform wrt any processor, all previous SYNCH accesses must be performed wrt everyone
- Before a SYNCH access is allowed to perform wrt any processor, all previous ordinary LOAD/STORE accesses must be performed wrt everyone
- SYNCH accesses are sequentially consistent wrt one another

Release Consistency



- Before any ordinary LOAD/STORE is allowed to perform wrt any processor, all previous ACQUIRE accesses must be performed wrt everyone
- Before any RELEASE access is allowed to perform wrt any processor, all previous ordinary LOAD/STORE accesses must be performed wrt everyone
- Acquire/Release accesses are processor consistent wrt one another

Enforcing Consistency

- The hardware provides underlying instructions that are used to enforce consistency
 - Fence or Memory Bar instructions
- Different processors provide different types of fence instructions

Some Examples

| | W→R Order | W→W Order | W→RW Order | Rd Others' Wt Earl | Read Own Write Early | Safety Net |
|---------------------|--------------|--------------|---------------|-----------------------|-------------------------|--------------------------------|
| Sequential | | | | | X | |
| IBM 370 | X | | | | | Serialisation Inst |
| Total Store Order | X | | | | X | ReadModifyWrite |
| Processor (PC) | X | | | X | X | RMW |
| Partial Store Order | X | X | | | X | RMW, STBAR |
| Weak | X | X | X | | X | Synchronization |
| Release (SC) | X | X | X | | X | Release,acquire, nsync, RMW |
| Release (PC) | X | X | X | X | X | Release,acquire, nsync, RMW |
| Alpha | X | X | X | | X | MB, WMB |
| SPARC v9 | X | X | X | | X | Various MBAR |
| PowerPC | X | X | X | X | X | SYNC |

Final Thoughts

- What consistency model best describes pthreads?
- How does all this relate to the OpenMP flush directive?
- We will talk more about this in the context of software distributed shared memory systems.