Chapter 5 - Synchronization

Single Processor Synchronization - Test and Set

```
function TestAndSet(boolean_ref lock) {
    boolean initial = lock;
    lock = true;
    return initial;
}
```

Processor Synchronization - Exchange

```
          addi  x2,R0,#1
lockit:   EXCH  x2,0(x1)      ;atomic exchange
          bnez  x2,lockit     ;already locked?
```

Load Reserved (lr) - Store Conditional (sc)

```
try:    mov    x3,x4      ;mov exchange value
        lr     x2,x1      ;load reserved from
        sc     x3,0(x1)   ;store conditional
        bnez   x3,try     ;branch store fails
        mov    x4,x2      ;put load value in x4
```

```
lockit:  ld    x2,0(x1)    ;load of lock
         bnez  x2,lockit   ;not available-spin
         addi  x2,R0,#1    ;load locked value
         EXCH  x2,0(x1)    ;swap
         bnez  x2,lockit   ;branch if lock wasn't 0
```

Step	P0	P1	P2	Coherence state of lock at end of step	Bus/directory activity
1	Has lock	Begins spin, testing if lock = 0	Begins spin, testing if lock = 0	Shared	Cache misses for P1 and P2 satisfied in either order. Lock state becomes shared.
2	Set lock to 0	(Invalidate received)	(Invalidate received)	Exclusive (P0)	Write invalidate of lock variable from P0.
3		Cache miss	Cache miss	Shared	Bus/directory services P2 cache miss; write-back from P0; state shared.
4		(Waits while bus/ directory busy)	Lock = 0 test succeeds	Shared	Cache miss for P2 satisfied.
5		Lock = 0	Executes swap, gets cache miss	Shared	Cache miss for P1 satisfied.
6		Executes swap, gets cache miss	Completes swap: returns 0 and sets lock = 1	Exclusive (P2)	Bus/directory services P2 cache miss; generates invalidate; lock is exclusive.
7		Swap completes and returns 1, and sets lock = 1	Enter critical section	Exclusive (P1)	Bus/directory services P1 cache miss; sends invalidate and generates write-back from P2.
8		Spins, testing if lock = 0			None

Figure 5.22 Cache coherence steps and bus traffic for three processors, P0, P1, and P2. This figure assumes write invalidate coherence. P0 starts with the lock (step 1), and the value of the lock is 1 (i.e., locked); it is initially exclusive and owned by P0 before step 1 begins. P0 exits and unlocks the lock (step 2). P1 and P2 race to see which reads

Using lr-sc instead of XCHG

```
lockit:  lr    x2,0(x1)    ;load reserved
         bnez  x2,lockit   ;not available-spin
         addi  x2,R0,#1    ;locked value
         sc    x2,0(x1)    ;store
         bnez  x2,lockit   ;branch if store fails
```

Consistency Models

```
P1:     A = 0;          P2:    B = 0;
        .....                  .....
        A = 1;                 B = 1;
L1:     if (B == 0)...  L2:    if (A == 0)...
```

Example Suppose we have a processor where a write miss takes 50 cycles to establish ownership, 10 cycles to issue each invalidate after ownership is established, and 80 cycles for an invalidate to complete and be acknowledged once it is issued. Assuming that four other processors share a cache block, how long does a write miss stall the writing processor if the processor is sequentially consistent? Assume that the invalidates must be explicitly acknowledged before the coherence controller knows they are completed. Suppose we could continue executing after obtaining ownership for the write miss without waiting for the invalidates; how long would the write take?

Answer When we wait for invalidates, each write takes the sum of the ownership time plus the time to complete the invalidates. Because the invalidates can overlap, we need only worry about the last one, which starts $10+10+10+10=40$ cycles after ownership is established. Therefore the total time for the write is $50+40+80=170$ cycles. In comparison, the ownership time is only 50 cycles. With appropriate write buffer implementations, it is even possible to continue before ownership is established.

1. Relaxing only the W → R ordering yields a model known as *total store ordering* or *processor consistency*. Because this model retains ordering among writes, many programs that operate under sequential consistency operate under this model, without additional synchronization.
2. Relaxing both the W → R ordering and the W → W ordering yields a model known as *partial store order*.
3. Relaxing all four orderings yields a variety of models including *weak ordering*, the PowerPC consistency model, and *release consistency*, the RISC V consistency model.

Model	Used in	Ordinary orderings	Synchronization orderings
Sequential consistency	Most machines as an optional mode	$R \rightarrow R$, $R \rightarrow W$, $W \rightarrow R$, $W \rightarrow W$	$S \rightarrow W$, $S \rightarrow R$, $R \rightarrow S$, $W \rightarrow S$, $S \rightarrow S$
Total store order or processor consistency	IBMS/370, DEC VAX, SPARC	$R \rightarrow R$, $R \rightarrow W$, $W \rightarrow W$	$S \rightarrow W$, $S \rightarrow R$, $R \rightarrow S$, $W \rightarrow S$, $S \rightarrow S$
Partial store order	SPARC	$R \rightarrow R$, $R \rightarrow W$	$S \rightarrow W$, $S \rightarrow R$, $R \rightarrow S$, $W \rightarrow S$, $S \rightarrow S$
Weak ordering	PowerPC		$S \rightarrow W$, $S \rightarrow R$, $R \rightarrow S$, $W \rightarrow S$, $S \rightarrow S$
Release consistency	MIPS, RISC V, Armv8, C, and C++ specifications		$S_A \rightarrow W$, $S_A \rightarrow R$, $R \rightarrow S_R$, $W \rightarrow S_R$, $S_A \rightarrow S_A$, $S_A \rightarrow S_R$, $S_R \rightarrow S_A$, $S_R \rightarrow S_R$

Figure 5.23 The orderings imposed by various consistency models are shown for both ordinary accesses and synchronization accesses. The models grow from most restrictive (sequential consistency) to least restrictive (release

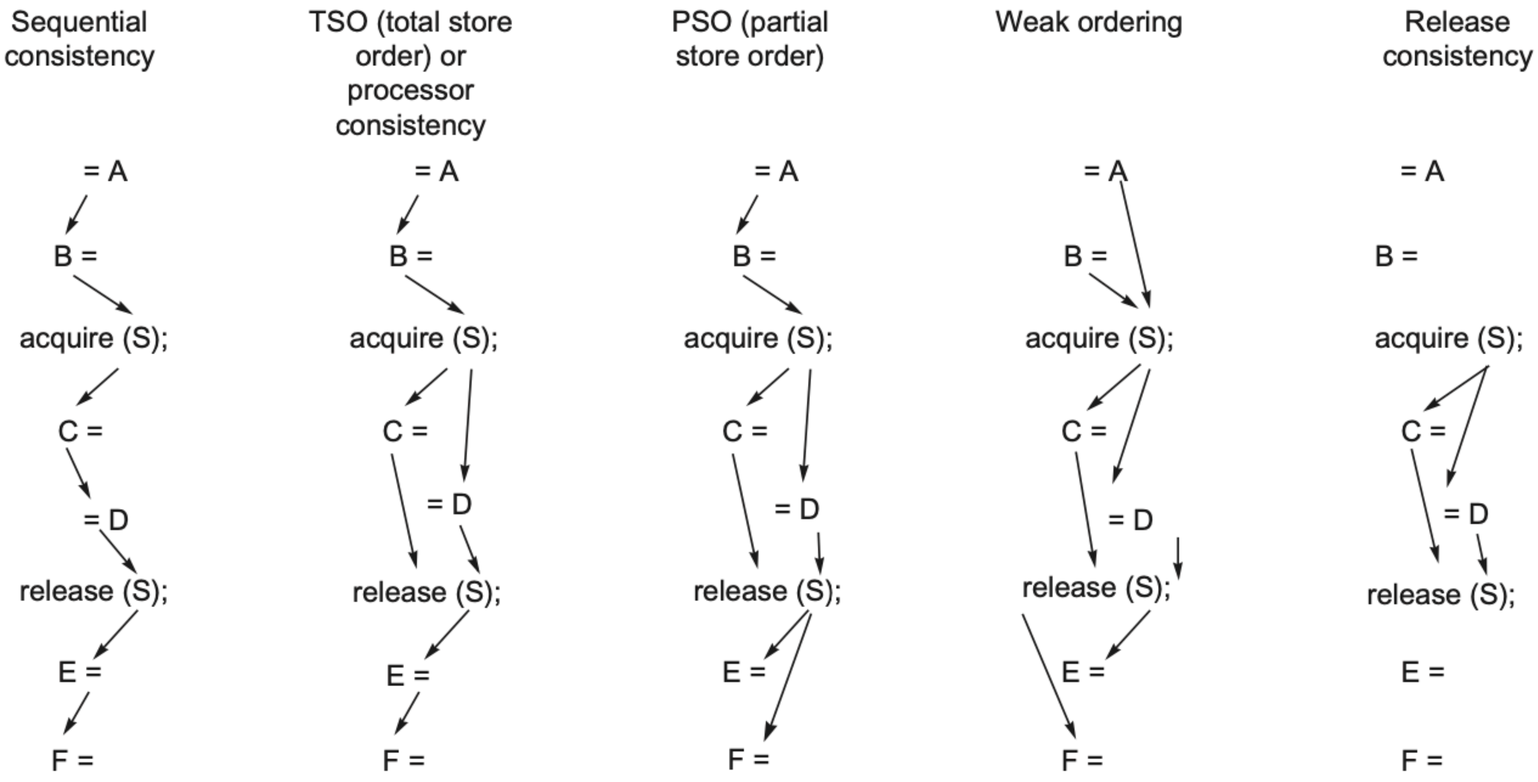

Figure 5.24 These examples of the five consistency models discussed in this section show the reduction in the number of orders imposed as the models become more relaxed. Only the minimum orders are shown with arrows.

Feature	IBM Power8	Intel Xeon E7	Fujitsu SPARC64 X+
Cores/chip	4, 6, 8, 10, 12	4, 8, 10, 12, 22, 24	16
Multithreading	SMT	SMT	SMT
Threads/core	8	2	2
Clock rate	3.1–3.8 GHz	2.1–3.2 GHz	3.5 GHz
L1 I cache	32 KB per core	32 KB per core	64 KB per core
L1 D cache	64 KB per core	32 KB per core	64 KB per core
L2 cache	512 KB per core	256 KB per core	24 MiB shared
L3 cache	L3: 32–96 MiB: 8 MiB per core (using eDRAM); shared with nonuniform access time	10–60 MiB @ 2.5 MiB per core; shared, with larger core counts	None
Inclusion	Yes, L3 superset	Yes, L3 superset	Yes
Multicore coherence protocol	Extended MESI with behavioral and locality hints (13-states)	MESIF: an extended form of MESI allowing direct transfers of clean blocks	MOESI
Multichip coherence implementation	Hybrid strategy with snooping and directory	Hybrid strategy with snooping and directory	Hybrid strategy with snooping and directory
Multiprocessor interconnect support	Can connect up to 16 processor chips with 1 or 2 hops to reach any processor	Up to 8 processor chips directly via Quickpath; larger system and directory support with additional logic	Crossbar interconnect chip, supports up to 64 processors; includes directory support
Processor chip range	1–16	2–32	1–64
Core count range	4–192	12–576	8–1024

Figure 5.26 Summary of the characteristics of three recent high-end multicore processors (2015–2017 releases) designed for servers. The table shows the range of processor counts, clock rates, and cache sizes within each pro-