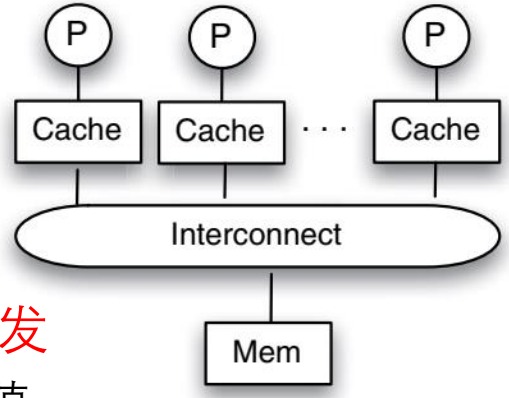
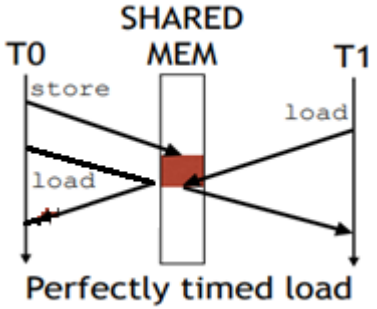
多线程多核系统中的
数据一致性、同步、存储同一性

$2.11，$5.10，$5.12，$5.14

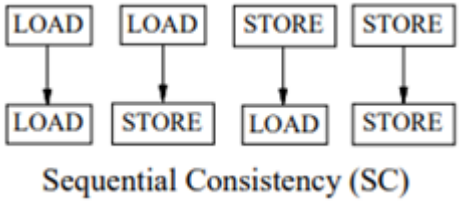
共享存储架构的挑战：多核/多处理器



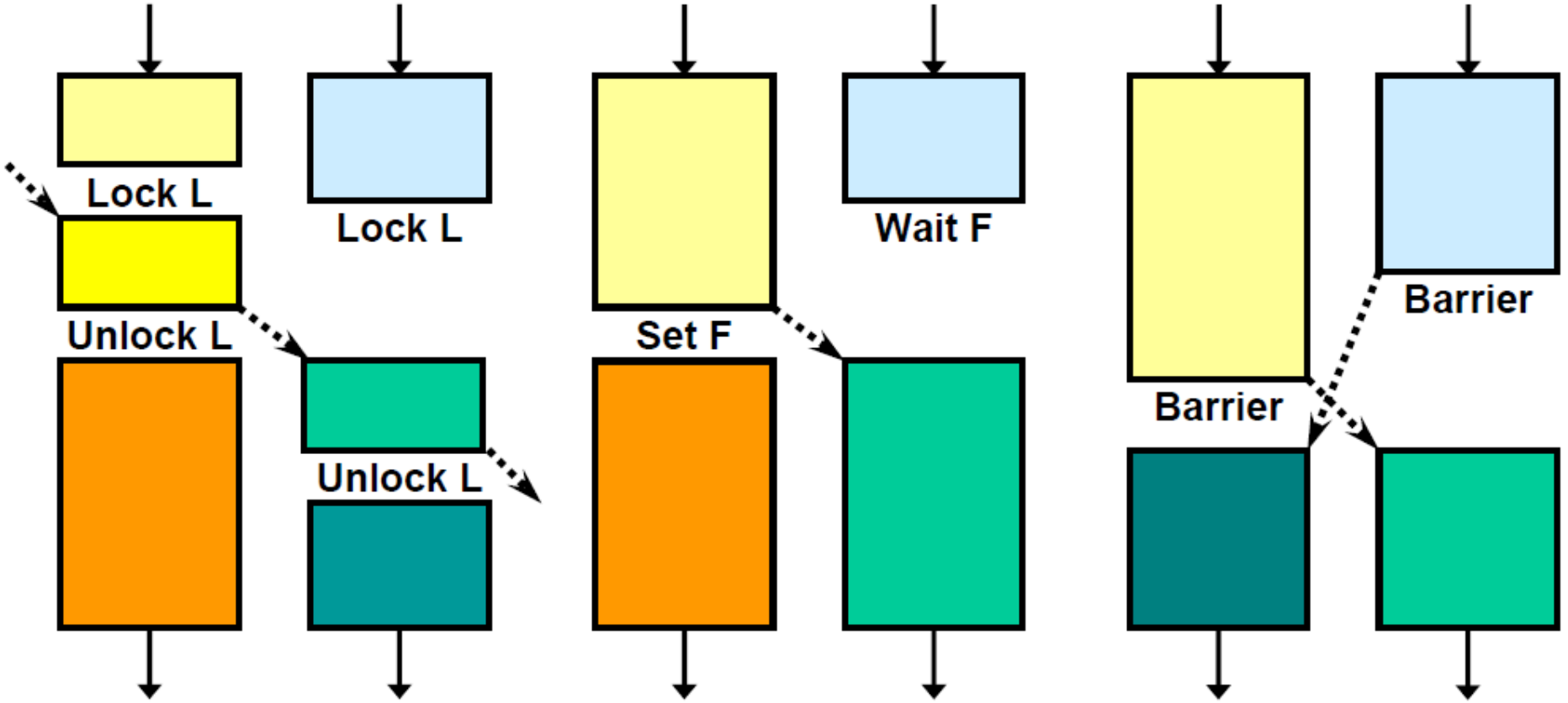
- 程序员的单处理器单线程顺序执行模型假设：程序序【线程序】，原子性，无并发
  - 1）相同的输入，产生相同的输出，2）对任一数据的读操作都能返回其最近被写入的值。
- 多线程SM的关键问题【CRC】：Coherence，Race conditions，Consistency【RV2“连续性”】
  - Coherence：单个data的multi-copy的一致性。写传播，串行化（不一定按程序序）。如CCP【MSI】
  - Race Conditions：多线程并发/并行访问共享资源，且最终执行结果依赖于它们的**执行顺序**时的系统状态
    - data race：并发读写同一data，且至少有一个write，无同步。一种race condition。“相同输入，输出不确定”
    - 原因与解决：线程交错不确定。原子性，串行化；禁止并发（关中断，关调度器）
      - 原子操作：RMW，CAS，LL/SC（无ABA问题）
      - Critical Sections：一次仅允许一个线程访问【执行】的共享资源区【代码区】，SYNC（lock）
      - 只读共享：如果共享数据在初始化后是只读的，则并发读取不会引发Data Race。
      - 线程本地存储：将数据变为线程私有的，从根本上移除“共享”。复制replication



  - Consistency：访存的顺序（普通/同步）与程序序的同一性，SC/TSO/RC等，SYNC（fence）
- 关键机制：同步synchronization（coordination mechanism）
  - 数据同步：互斥（lock，acq/rel，串行化）
  - 事件同步：任务协同（点对点（flag，wait-signal，rel/acq），全局（barrier））

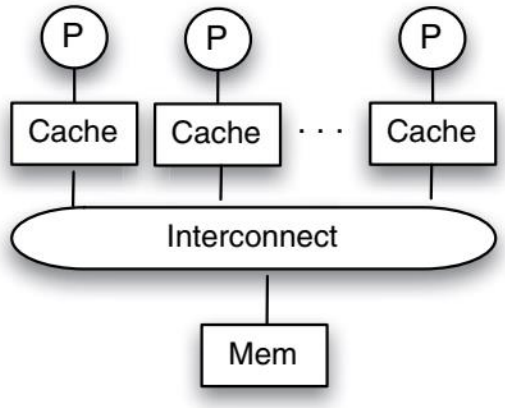


Different Synchronization Ops：两个线程



内容

- 数据一致性coherence：$5.10（$5.12 实现CCP ）
  - MESI：写传播，事务串行化，
- 原子指令：$2.11
  - LR/SC：load-bnz-store语义。load≠lock，store≠unlock
  - AMO
- 同步指令：$5.14
  - fence
  - 访存顺序一致性consistency（连续性，同一性），任务协作



数据一致性问题

- 同一data（X）的多个copy
  - 写传播
  - 读写操作的原子性（写操作）
- 访存操作的全局序不确定性
  - P1的wr与P2的wr先后顺序?
  - 事务串行化

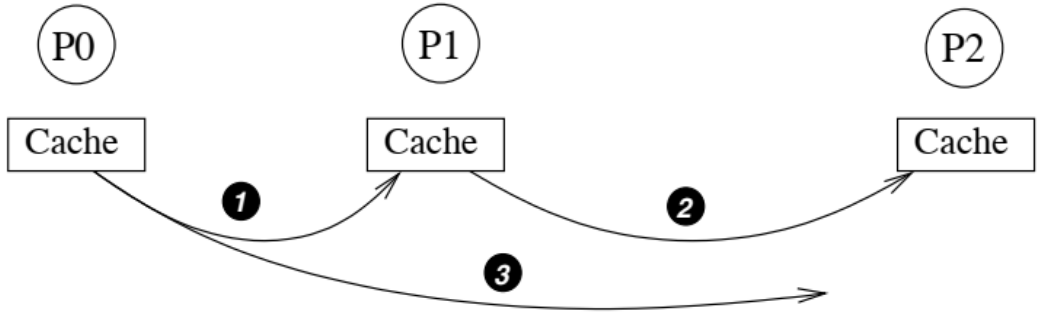
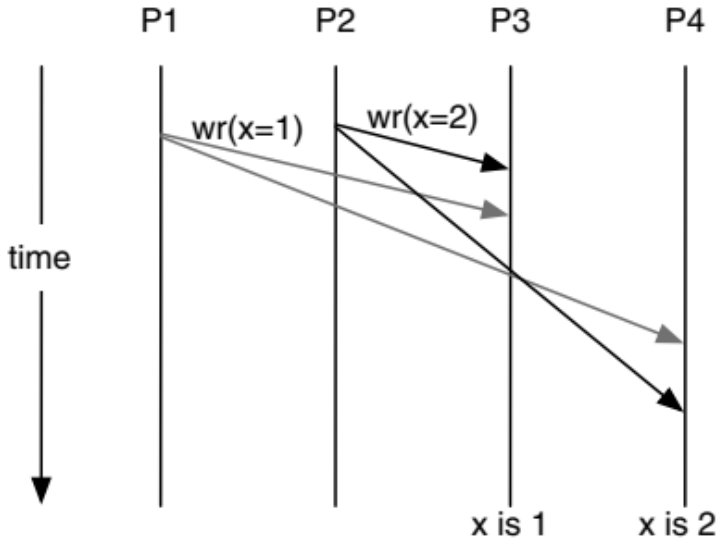
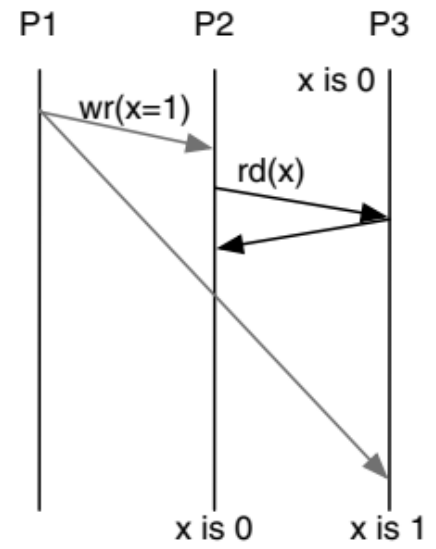


图5.39：写直达，T3时CPU_A写1，A$=1,而B$=0

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0		inv	inv	0
1	CPU A reads X	0	inv	0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

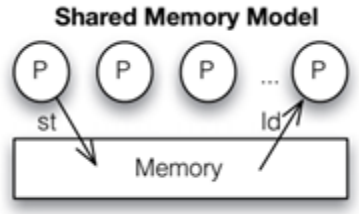


违反写操作串行化

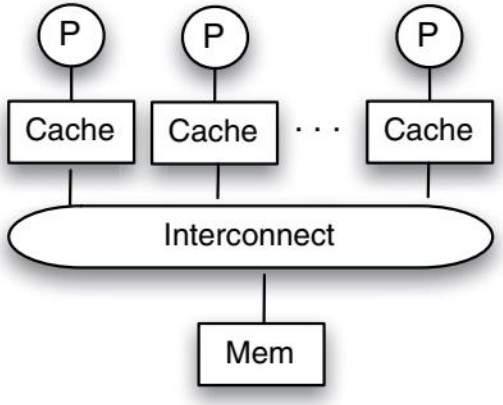


违反读写操作串行化

RV$5.10：并发、并行和存储层次结构，CC

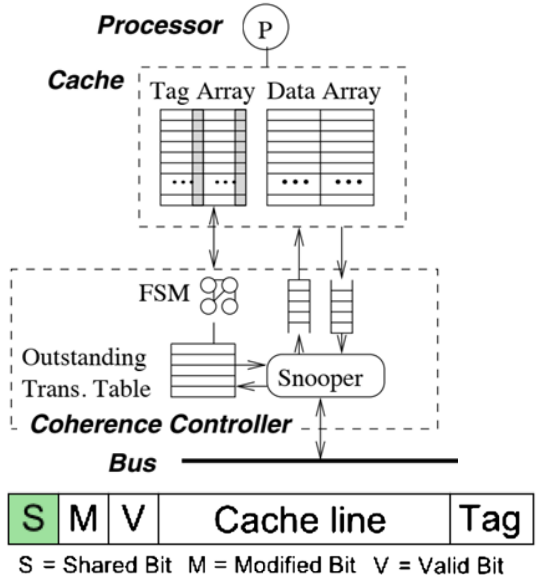


- 多线程：单核多线程或多核多线程，共享一个物理地址空间
- memory coherence and consistency问题：data，读值，写值，副本
  - （数据）一致性coherence：读操作将返回什么值
  - （访存）连续性consistency：写值何时被读操作返回（“看到”）
- 一致性存储系统：对任一数据的读操作都能返回其最近被写入的值。
  - 自写自读：程序序
  - 他写我读：各处值相同（一定的时间间隔之后），原子性，全局序
    - 写串行化：全局序。同一位置，避免不同处理器的读值有新有旧
- Cache一致性：跟踪每个共享数据块的状态（MSI）
  - Cache的作用：迁移（减少远程访问延迟），复制（减少共享竞争）



Cache Coherence：$5.10，$5.12

- 嗅探协议snoopy，目录协议dir：使无效式，更新式
- Snooping Protocols：a write invalidate protocol
  - 分布式一致性状态维护（VI，MSI，MESI）
    - 写传播：写操作独占访问【Bus】，并使其他副本无效【eager/lazy】
    - 写串行化：对同一data，写竞争时只允许有一个写（SWMR）
      - Bus广播协议保证



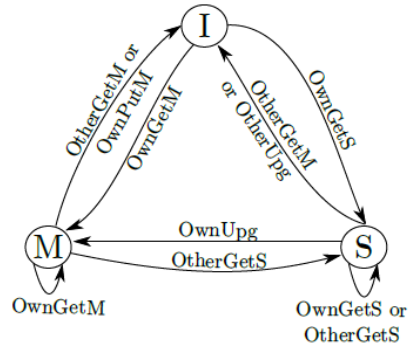


FIGURE 5.40 write-back， invalidation protocol， snooping bus

图5.39：写直达，T3时CPU_A写1，A$=1,而B$=0

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0		inv	inv	0
1	CPU A reads X	0	inv	0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
		inv	inv	0
CPU A reads X	Cache miss for X	0	inv	0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1	inv	0
CPU B reads X	Cache miss for X	1	1	1 写回

RV原子指令1：LR/SC指令，lr=ll，RV$2.11

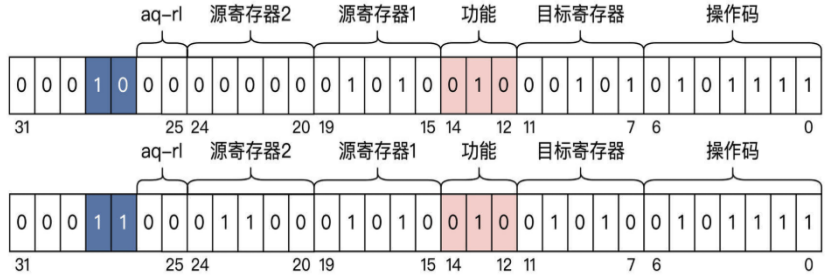
- 功能：实现原子操作RMW和锁
- lr：装入同步变量（a0）到t0
  - 其后可跟任何对t0的操作指令
    - 即进行RMW的“M”操作
- sc：写回同步变量
  - iff在lr之后没有其他处理器对该单元或缓存块进行写
  - 否则改写a0作废，重新尝试lock
- lr本身不是加锁，sc也不是解锁
  - lr完成并不意味着获得了排他访问权
  - 成功的lr-sc保证的是两者之间没有发生对同步变量的写
    - 两者间的操作只是针对同步变量，失败时不可见
  - lock()：同步变量（a0），如图
  - unlock()

    unlock: st (a0), 0

    ret
- 多个处理器可能同时执行lock

```
1  .globl lrsc_ins
2  #a0内存地址
3  #a1预期值
4  #a2所需值
5  #a0返回值，如果成功，则为0！否则为1
6  lrsc_ins:
7  cas:
8      lr.w t0, (a0)        #加载以前的值
9      bne t0, a1, fail     #不相等则跳转到fail
10     sc.w a0, a2, (a0)    #尝试更新
11     jr ra                #返回
12 fail:
13     li a0, 1             #a0 = 1
14     jr ra                #返回
15
```

RV原子指令1：LR/SC指令，RV$2.11



- LR/SC指令：不同 hart 之间同步【DR，CS】
- lr.{w/d}.{aqrl} rd，(rs1);
  - {可选内容}：w（32位）、d（64位），aqrl为内存顺序，一般使用默认的【下页】
  - rd为目标寄存器
  - rs1为源寄存器1；内存地址，加载后会设置当前CPU hart读取该地址的保留位
- sc.{w/d}.{aqrl} rd，rs2，(rs1)；先判断rs1对应地址的保留位有没有被设置
  - rd目标寄存器，成功则为0，否则为1（非0）
  - rs1源寄存器1，含保留位，与lr相同
  - rs2源寄存器2，被写入（rs1）
- 指令格式：R-type（图2.18）
  - 上图lr指令，下图sc指令

只有满足以下四个条件，SC才能执行成功：
1.LR和SC访问相同的地址。
2.LR和SC之间没有任何其它的写操作访问同一地址（来自任何一个hart）。
3.LR和SC指令之间没有任何中断与异常发生。
4.LR和SC指令之间没有执行MRET指令。

.aqrl：内存序约束标记

- 便于在原子指令上施加额外的内存顺序限制
  - .aq，约束为acquire内存序
  - .rl，约束为release内存序
  - .aqrl，约束为顺序一致（Sequential Consistency，SC）内存序

Acquire	Release	含义
0	0	没有顺序限制
0	1	该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到
1	0	该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行
1	1	该指令前序所有访问存储的指令的结果必须在该指令执行之前被观察到，该指令后序所有访问存储的指令必须等该指令执行完成后才开始执行

LR/SC指令实现：基于Cache

- 执行LR时：将对应Cache行标记为“保留”状态；加载数据到寄存器
  - 多核支持：需通过总线或 Cache 一致性协议确保全局可见性
- 执行SC时：检查Cache行是否仍处于“保留”状态
  - 是 → 执行存储，清除保留标记，返回成功；否 → 返回失败
  - 无论成功失败，都通过总线发送 invalidate 消息，确保其他CPU的Cache副本失效
- 保证顺序一致性：LR 和 SC 之间的指令不能重排序
  - LR 和 SC 之间插入隐式的内存屏障，防止指令重排序
- 异常处理：在发生中断、异常或上下文切换时需清除保留状态
- 一般不允许LR/SC 对嵌套，第二次 LR 会清除前一次的保留状态

原子指令2：原子内存操作AMO，RV$2.11

- 对于单个变量，LR/SC使用不方便
  - 每个AMO指令完成的操作不可分割，不能被外部事件打断
  - 分为：原子交换指令、原子加法指令、原子逻辑指令和原子取大小值指令
    - amoswap.{w/d}.{aqrl} rd,rs2,(rs1)
    - 把内存地址rs1中的数据加载到rd寄存器中，然后把rs2中的数据写入到rs1指向的内存单元中
- 可用LR/SC实现AMO
- 临界区：两种方法
  - 可用amoswap【左】，或
  - fence【右，强】

```
    li          t0, 1        # Initialize swap value.
again:
    amoswap.w.aq t0, t0, (a0) # Attempt to acquire lock.
    bnez        t0, again    # Retry if held.
    # ...
    # Critical section.
    # ...
    amoswap.w.rl x0, x0, (a0) # Release lock by storing 0.
```

```
1          sd          x1, (a1)     # Arbitrary unrelated store
2          ld          x2, (a2)     # Arbitrary unrelated load
3          li          t0, 1        # Initialize swap value.
4      again:
5          amoswap.w   t0, t0, (a0) # Attempt to acquire lock.
6          fence       r, rw        # Enforce "acquire" memory ordering
7          bnez        t0, again    # Retry if held.
8          # ...
9          # Critical section.
10         # ...
11         fence       rw, w        # Enforce "release" memory ordering
12         amoswap.w   x0, x0, (a0) # Release lock by storing 0.
13         sd          x3, (a3)     # Arbitrary unrelated store
14         ld          x4, (a4)     # Arbitrary unrelated load
```

RV的fence：所有指令，不仅访存？$5.14

- 内存访问顺序（fence）强调的是同一个 hart 内的执行顺序
- fence [iorw], [iorw]：iorw分别表示fence要约束的前后指令的类型
  - PR/PW/SR/SW位：限制前导集/后续集中所包含指令的类型
  - 为了性能，FENCE可以进一步限制前导集和后续集为较小的内存访问集
    - FENCE RW,RW；相当于TSO
    - FENCE RW,W；之前的所有RW不能后移，之后的W不能前移。acquire语义
    - FENCE R,RW；之前的所有R【load】不能后移，之后的RW不能前移。release语义
    - FENCE R,R
    - FENCE W,W
- 实现fence
  - 冲刷本地Write Buffer到主存
  - 通知其他CPU失效其缓存副本

图5-47

Type	Mnemonic	Name
Mem. Ordering	FENCE.I	Instruction Fence
	FENCE	Fence
	SFENCE.VMA	Address Translation Fence

原子指令实现

- 在PentiumPro之前通过锁Bus阻止其他CPU核的内存访问实现lock
- 从PentiumPro开始，lock只会阻塞其他CPU核对相关的缓存块访问
  - LOCK的实现，只需要保持cache line的M和E状态，就可以阻止其他cpu核对该块内存的修改，而不用去锁住整个总线。
  - CAS的实现同样无需锁住总线，只须阻塞其他cpu核对相关内存的缓存块的访问。
- 在MIPS和ARM架构下，支持LL/SC的实现。
  - LL/SC不会出现CAS中的ABA问题，而且基于LL/SC可以实现CAS等原子操作。

休息是为了走更远的路!