

循序渐进，学习开发一个 RISC-V 上的操作系统



第 14 章 任务同步和锁

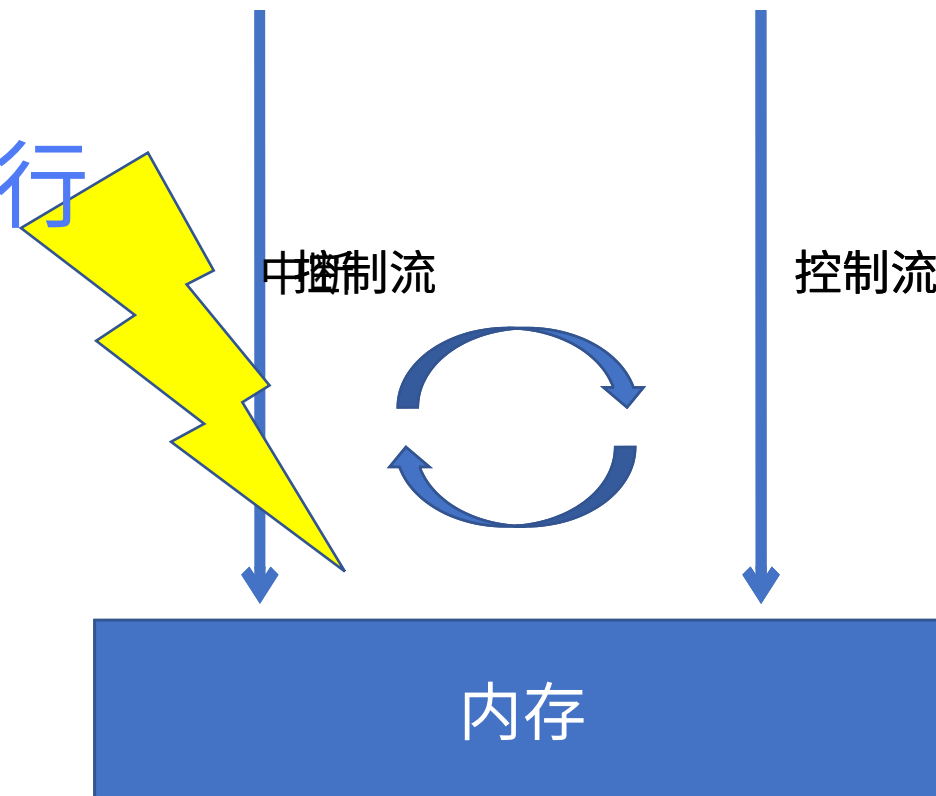
汪辰

- 并发与同步
- 临界区的概念
- 同步的实现 - 自旋锁
- 其他同步技术

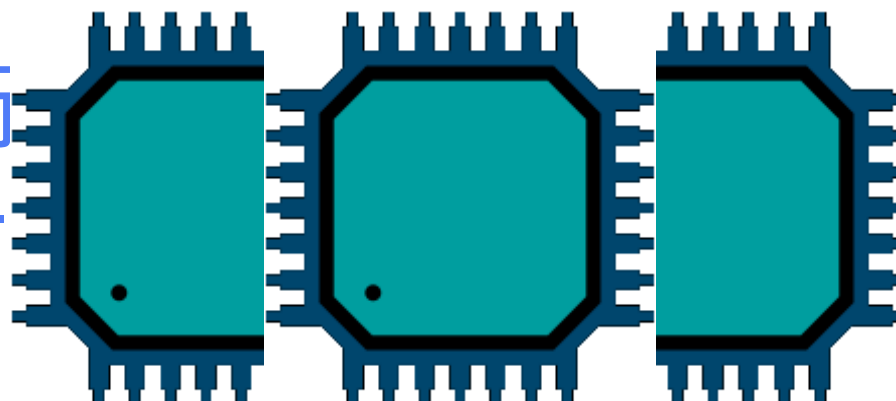
- 【参考 1】： The RISC-V Instruction Set Manual , Volume I: Unprivileged ISA , Document Version 20191213
- 【参考 2】： The RISC-V Instruction Set Manual , Volume II: Privileged Architecture , Document Version 20190608-Priv-MSU-Ratified

➤ 并发 指多个控制流同时执行

- 多处理器多任务
- 单处理器多任务
- 单处理器任务 + 中断

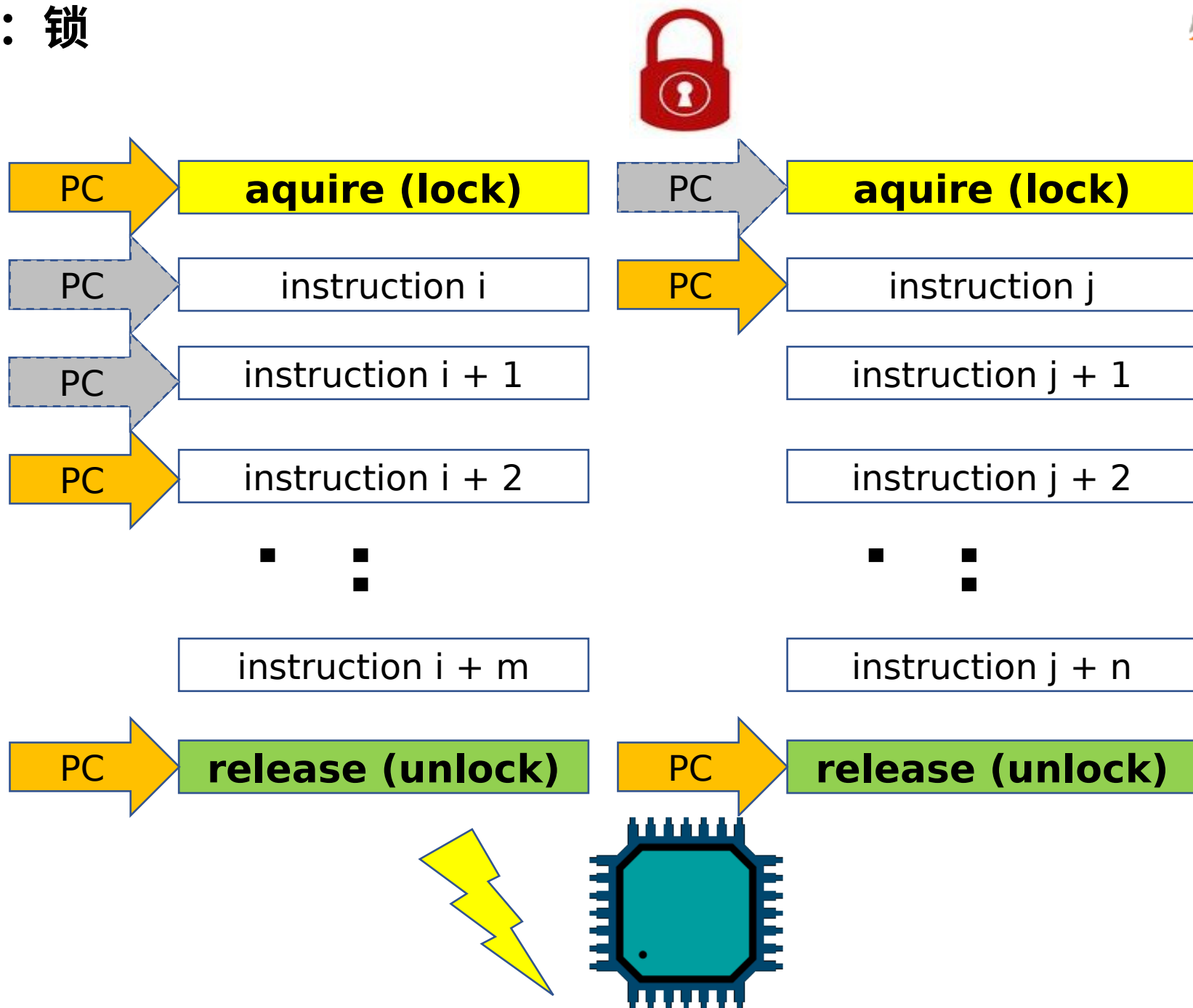


➤ 同步 是为了保证在并发执行流可以有效执行而采用的一



- **临界区**：在并发的程序执行环境中，所谓临界区（Critical Section）指的是一个会访问共享资源（例如：一个共享设备或者一块共享内存）的**指令片段**，且满足当这样的多个指令片段同时访问该共享资源时可能会引发问题。我们要做的是当有一个控制流进入临界区时，其他控制流必须等待。

同步的实现：锁



自旋锁

```
struct spinlock {
    int locked;
};

void initlock(struct spinlock *lk)
{
    lk->locked = 0;
}

// acquire
void spin_lock(struct spinlock *lk)
{
    for(;;) {
        if (lk->locked == 0) {
            lk->locked = 1;
            break;
        }
    }
}

// release
void spin_unlock(struct spinlock *lk)
{
    lk->locked = 0;
}
```



控制流

```
spin_lock(&lk);

// critical section
.....

spin_unlock(&lk);
```

控制流

```
spin_lock(&lk);

// critical section
.....

spin_unlock(&lk);
```

自旋锁

```
// acquire
void spin_lock(struct spinlock *lk)
{
    for(;;) {
        if (lk->locked == 0) {
            lk->locked = 1;
            break;
        }
    }
}
```



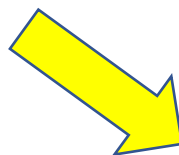
控制流 控制流

loop:

```
lw    a5, -20(s0)
lw    a4, 0(a5)
bnez  a4, loop
lw    a5, -20(s0)
li    a4, 1
sw    a4, 0(a5)
```


自旋锁

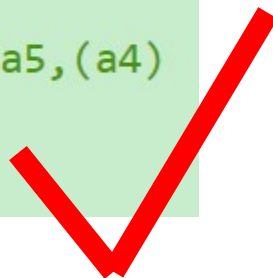
```
// acquire
void spin_lock(struct spinlock *lk)
{
    while(__sync_lock_test_and_set(&lk->locked, 1) != 0)
}
```



control flow

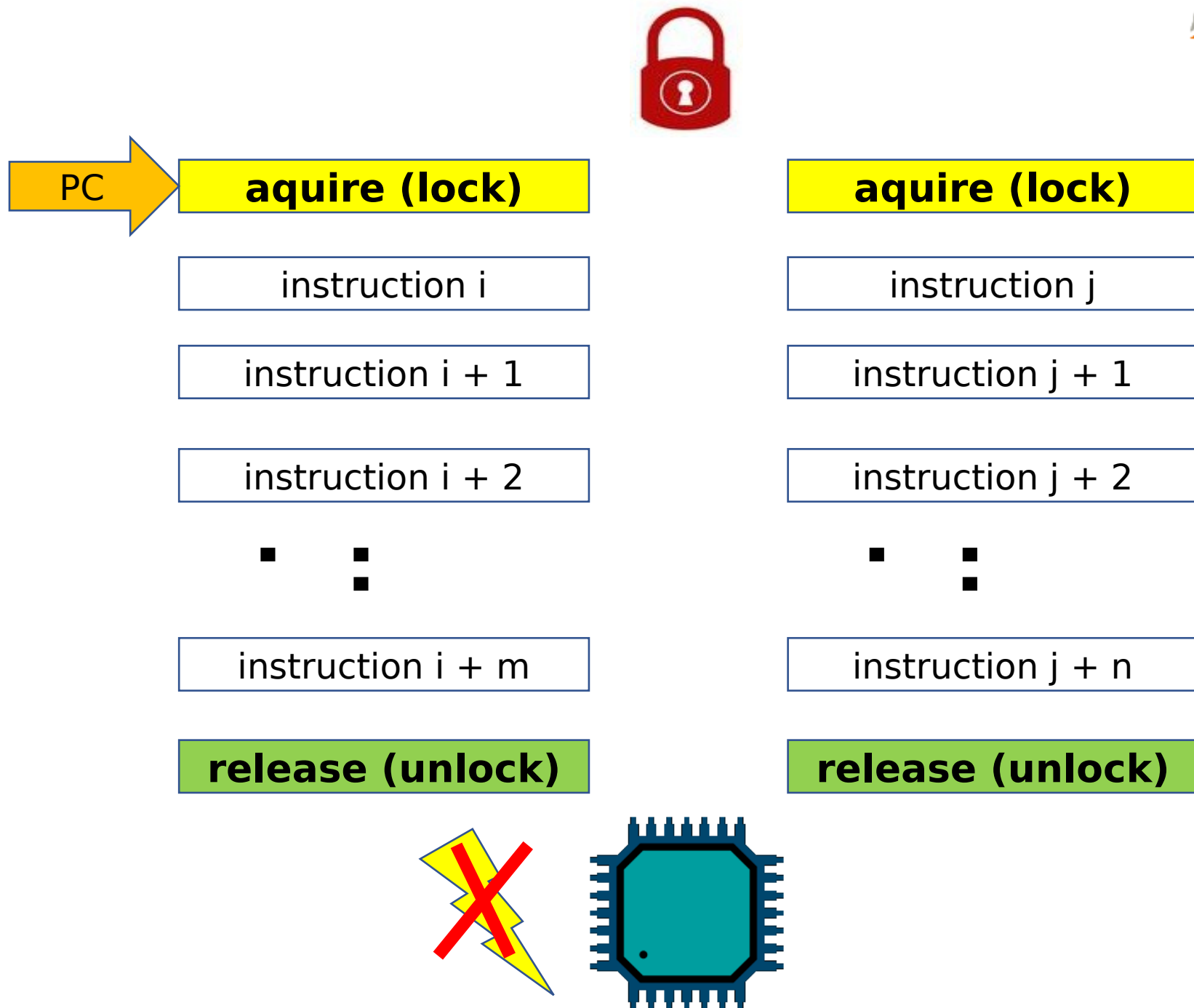
control flow

```
loop:
    lw      a4, -20(s0)
    li      a5, 1
    amoswap.w.aq    a5, a5, (a4)
    mv      a3, a5
    bnez    a3, loop
```



31	27	26	25	24	20	19	15	14	12	11	7	6	0
funct5					rs2		rs1		funct3		rd		opcode
5					5		5		3		5		7
AMOSWAP.W/D					src		addr		width		dest		AMO

自旋锁



同步技术	描述
自旋锁（Spin Lock）	如果一个任务试图获得一个已经被持有的自旋锁，这个任务就会进入忙循环（busy loops，即自旋）并等待，直到该锁可用，否则该任务就可以立刻获得这个锁并继续执行。自旋锁可以防止多个任务同时进入临界区（critical region）
信号量（Semaphore）	信号量是一种睡眠锁，当任务请求的信号量无法获得时，就会让任务进入等待队列并且让任务睡眠。当信号量可以获得时，等待队列中的一个任务就会被唤醒并获得信号量。
互斥锁（Mutex）	互斥锁可以看作是对互斥信号量（count 为 1）的改进，是一种特殊的信号量处理机制
完成变量（Completion Variable）	一个任务执行某些工作时，另一个任务就在完全变量上等待，当前者完成工作，就会利用完全变量来唤醒所有在这个完全变量上等待的任务。
.....

谢谢

欢迎交流合作