ISCAS NIST

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

2021/4

本章内容

- 并发与同步
- 临界区、锁、死锁
- 自旋锁的实现
- 其他同步技术

参考

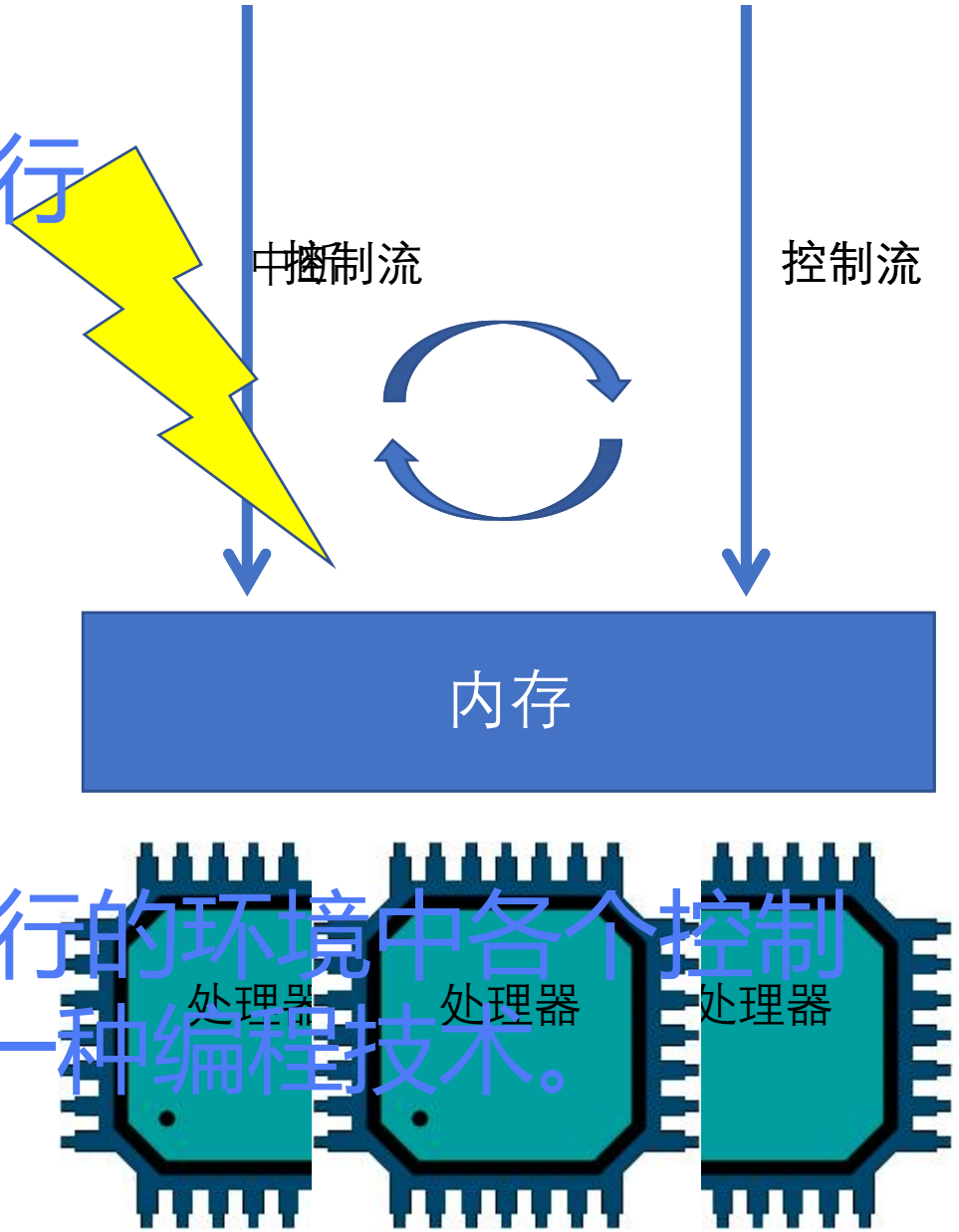
- 【参考 1】：The RISC-V Instruction Set Manual，Volume I: Unprivileged ISA，Document Version 20191213
- 【参考 2】: The RISC-V Instruction Set Manual，Volume II: Privileged Architecture，Document Version 20190608-Priv-MSU-Ratified

本章内容

- 并发与同步
- 临界区、锁、死锁
- 自旋锁的实现
- 其他同步技术

并发（Concurrency）与 同步

- **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的环境中各个控制流可以有效执行而采用的一种编程技术。

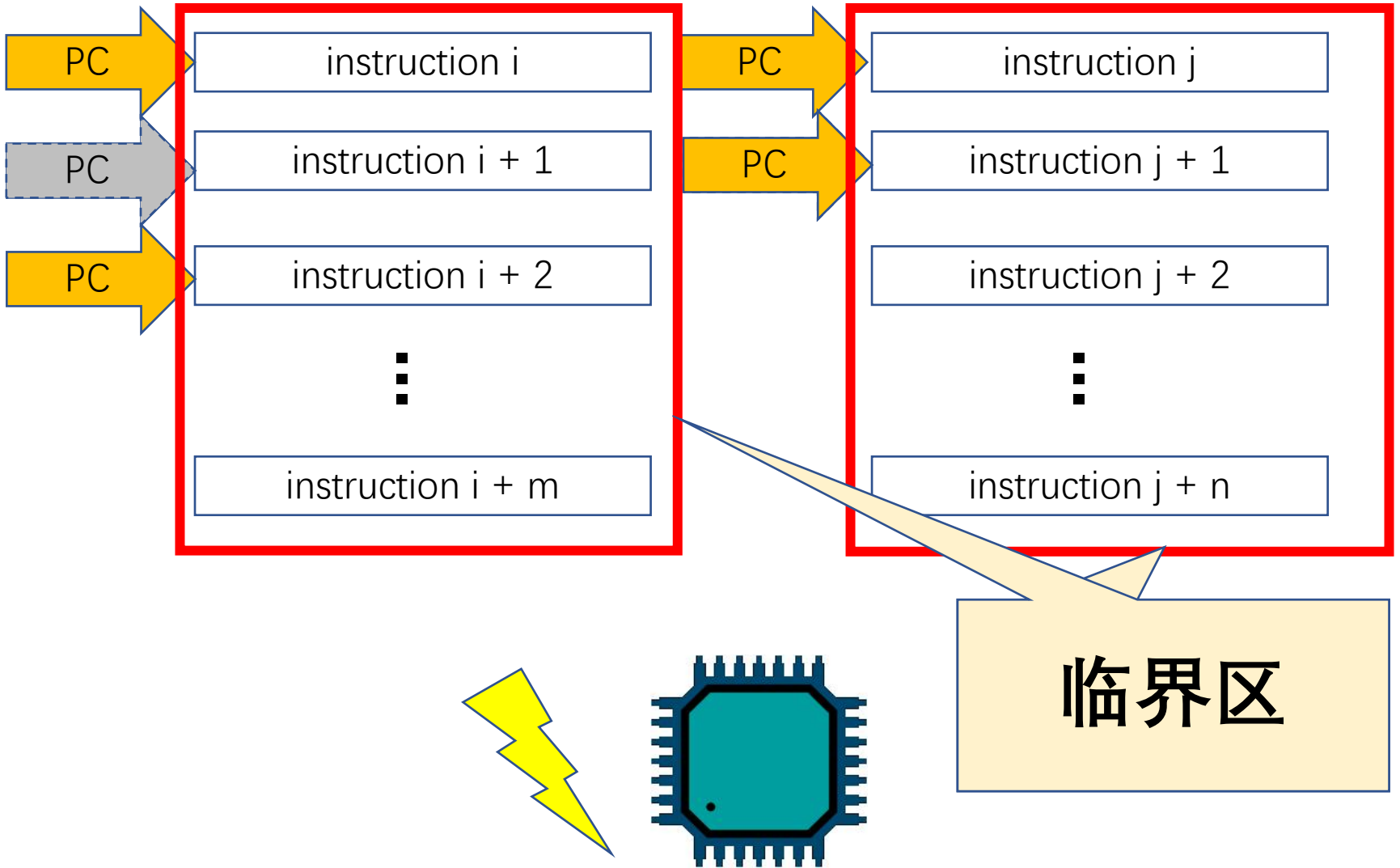
本章内容

- 并发与同步
- 临界区、锁、死锁
- 自旋锁的实现
- 其他同步技术

临界区

- **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存储内存）的**指令片段**，而且当这样的多个指令片段同时访问某个共享资源时可能会引发问题。

临界区


PC
PC
PC
instruction i
instruction i + 1
instruction i + 2
instruction i + m
PC
PC
instruction j
instruction j + 1
instruction j + 2
instruction j + n
临界区

实现同步的技术 - 锁

- 在并发环境下为了有效控制临界区的执行（同步），我们要做的是当有一个控制流进入临界区时，其他相关控制流必须 **等待**。
- 锁是一种最常见的用来实现同步的技术。
  - 不可睡眠的锁
  - 可睡眠的锁

实现同步的技术 - 锁

PC aquire (lock)

PC instruction i

PC instruction i + 1

PC instruction i + 2

instruction i + m

PC release (unlock)

PC aquire (lock)

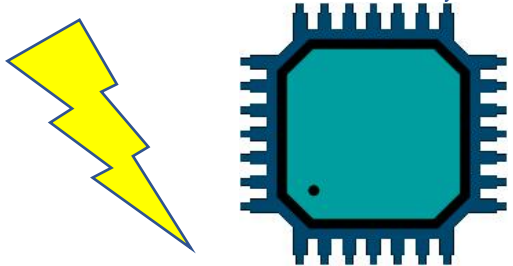
PC instruction j

instruction j + 1

instruction j + 2

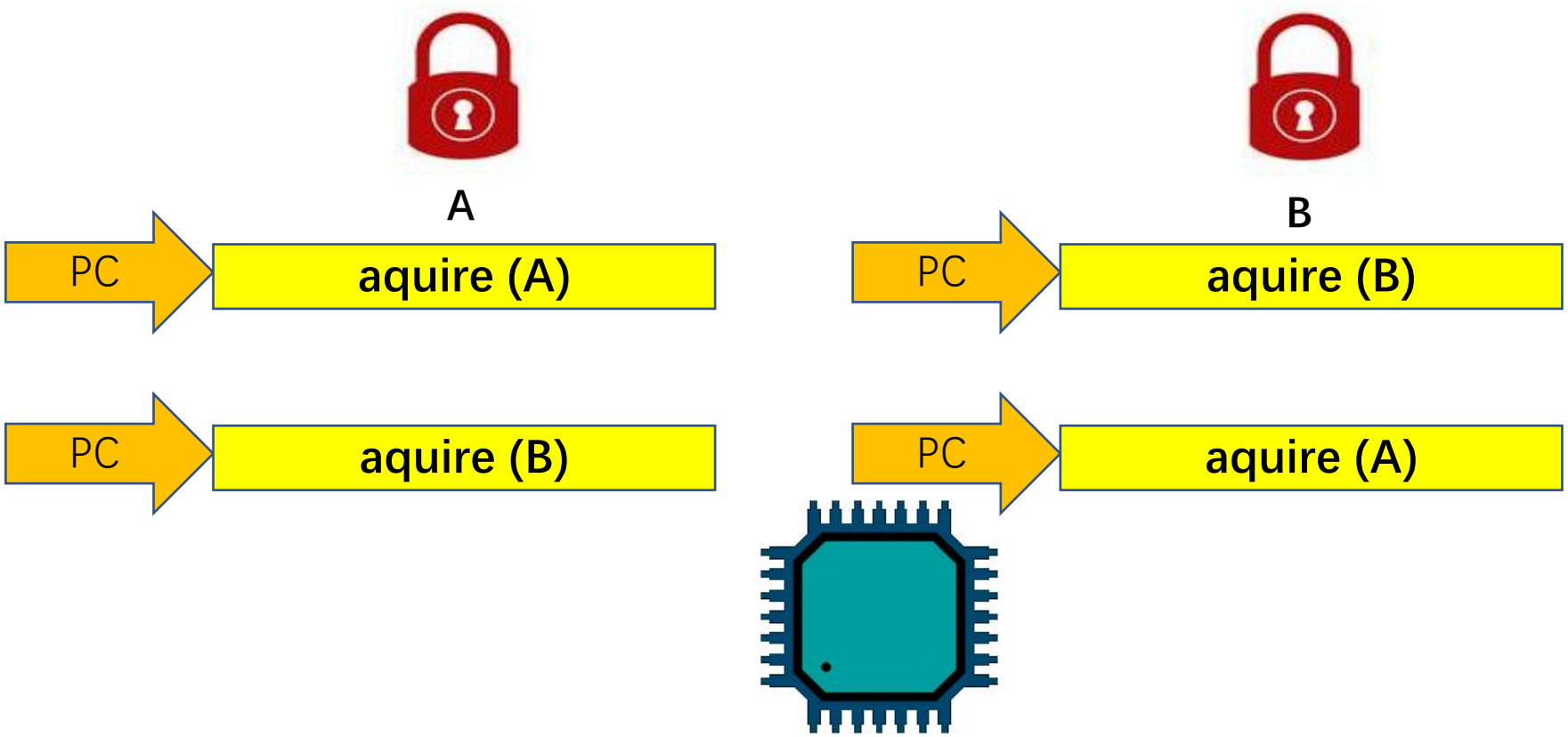
instruction j + n

PC release (unlock)



死锁（Deadlock）问题

- **什么是死锁以及死锁为何会发生：**
  - 当控制流执行路径中会涉及多个锁，并且这些控制流执行路径获取（aquire）锁的顺序不同时就可能会发生死锁问题。



死锁（Deadlock）问题

- **如何解决死锁：**
  - 调整获取（aquire）锁的顺序，譬如保持一致。
  - 尽可能防止任务在持有一把锁的同时申请其它的锁。
  - 尽可能少用锁，尽可能少并发。

本章内容

- 并发与同步
- 临界区、锁、死锁
- **自旋锁的实现**
- 其他同步技术

➢ 锁的分类

- 不可睡眠的锁
- 可睡眠的锁

自旋锁
（Spin Lock）

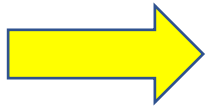
自旋锁

```c
struct spinlock {
        int locked;
};

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

// aquire
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;
}
```



```c
struct spinlock lk;

initlock(&lk);
```

控制流

```c
spin_lock(&lk);

// critical section
......

spin_unlock(&lk);
```

控制流

```c
spin_lock(&lk);

// critical section
......

spin_unlock(&lk);
```

自旋锁

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

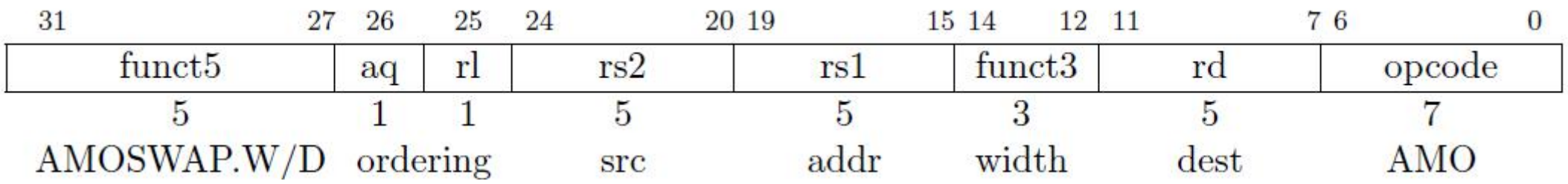
读取锁状态和上锁的操作必须是原子性的

自旋锁

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

控制流 控制流

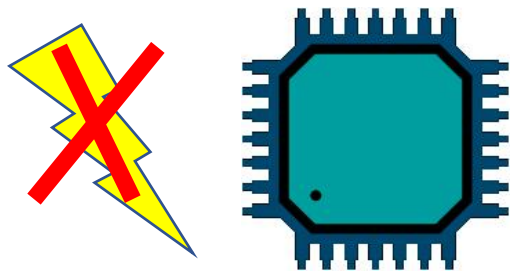
```
loop:
        lw      a4,-20(s0)
        li      a5,1
        amoswap.w.aq    a5,a5,(a4)
        mv      a3,a5
        bnez    a3, loop
```



【参考 1】8.4 Atomic Memory Operations

自旋锁

PC → **aquire (lock)**	**aquire (lock)**
instruction i	instruction j
instruction i + 1	instruction j + 1
instruction i + 2	instruction j + 2
⋮	⋮
instruction i + m	instruction j + n
release (unlock)	**release (unlock)**



- **自旋锁的使用：**
  - 自旋锁可以防止多个任务同时进入临界区（critical region）
  - 在自旋锁保护的临界区中不能执行长时间的操作。
  - 在自旋锁保护的临界区中不能主动放弃CPU

本章内容

- 并发与同步
- 临界区、锁、死锁
- 自旋锁的实现
- 其他同步技术

其他同步技术

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。
……	……

练习 14-1

谢 谢

欢迎交流合作