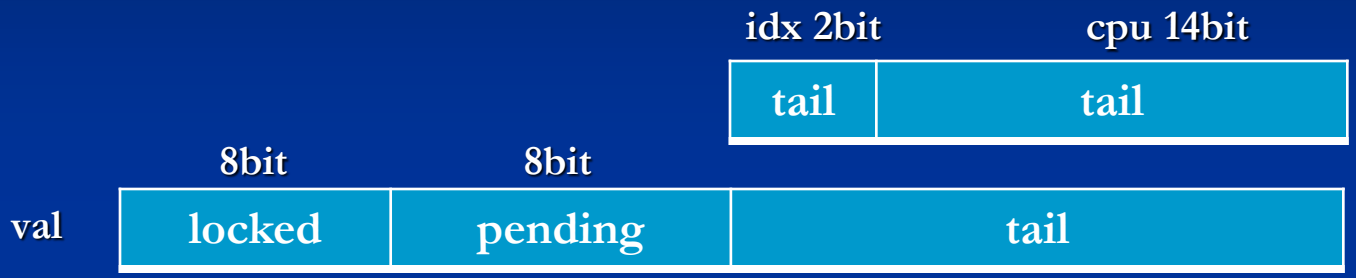
2.3.5 qspinlock

2.2.5.1 qspinlock的结构



locked:用于指示qspinlock是否加锁，0表示未加锁。
pending:第一个等待锁的CPU需要先设置pending位，后续等待锁的CPU则全部进入MCS spinlock队列自旋等待。
tail:由idx+cpu构成，标识等待队列最后一个节点。
idx:就是index，作为数组下标使用。
cpu:队尾的CPU编号，编号+1是为了和没有CPU排队的情况区分开来。

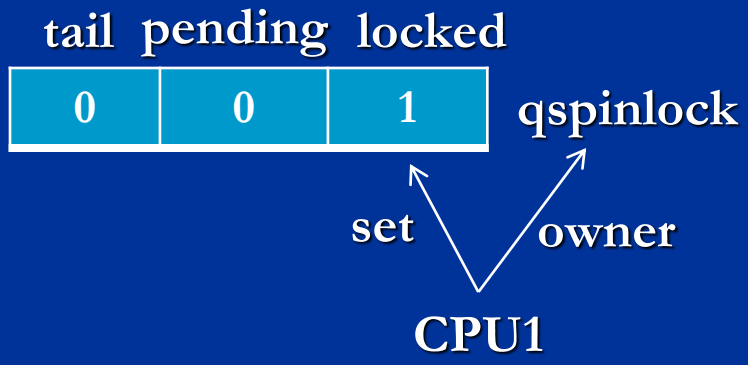
2.3.5 qspinlock

2.3.5.2 qspinlock加锁流程

(1) 第一个cpu获取锁

把一个val对应的locked,pending,tail作为一个三元组(x,y,z)来表示，三元组的初始值为(0,0,0)。

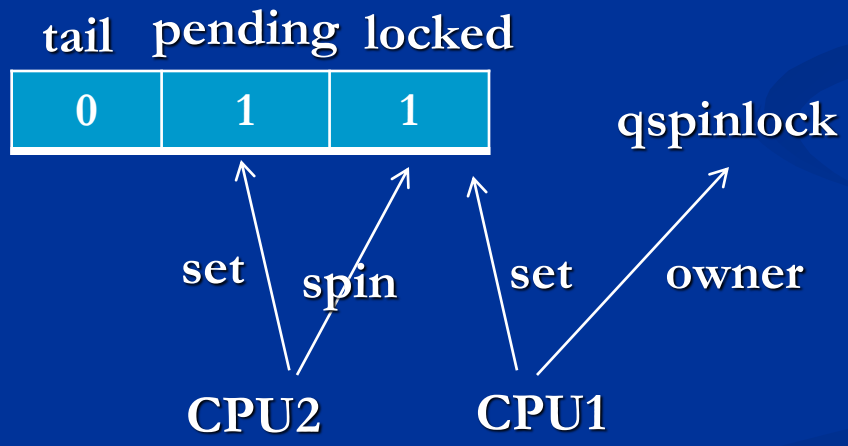
当第一个CPU试图获取锁时，locked的值从0变为1，三元组变成(0,0,1)。



2.3.5 qspinlock

(2) 第二个CPU获取锁

在第一个CPU未释放锁之前，第二个CPU获取锁必须等待。
设置pending，(0, 0, 1)->(0, 1, 1)，然后在qspinlock的locked值上自旋。
第一个CPU释放锁后，三元组(0, 1, 1)->(0, 1, 0)>(0,0,1)。



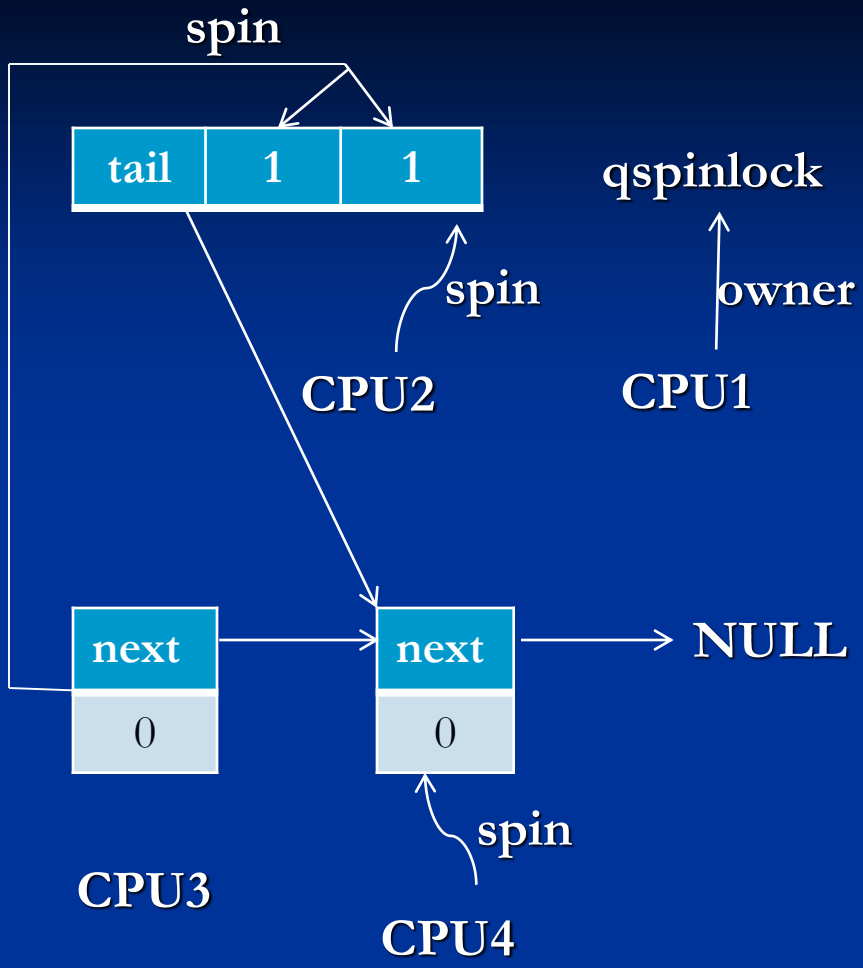
2.3.5 qspinlock

(3) N个CPU获取锁

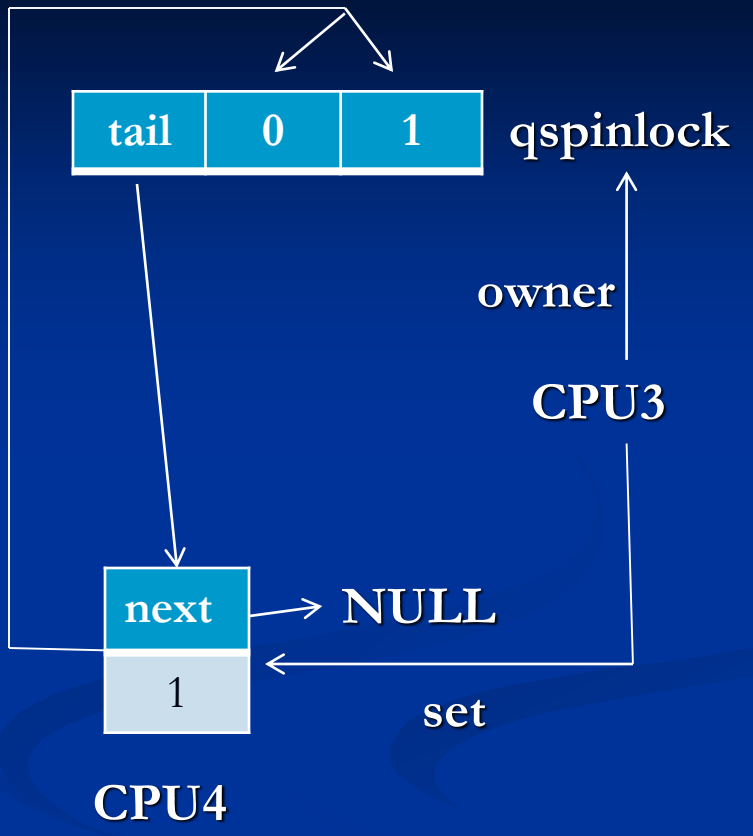
当第二个CPU还在等待时，第三个CPU需创建一个MCS节点，tail指向该节点。

若N>3则会创建多个MCS节点，这些节点构成一个等待队列。队头于qspinlock的locked和pending进行自旋。队中节点基于自身自旋。

当前两个CPU释放锁后，第三个CPU结束自旋，获取qspinlock。队中第二个节点结束自身自旋，切换为队头进行自旋应该基于的对象。



等待队列状态



队头获取锁时队列状态

2.3.5 qspinlock

2.3.5.3 qspinlock解锁流程

解锁只需要将locked byte置为0即可。