Locks in XV6

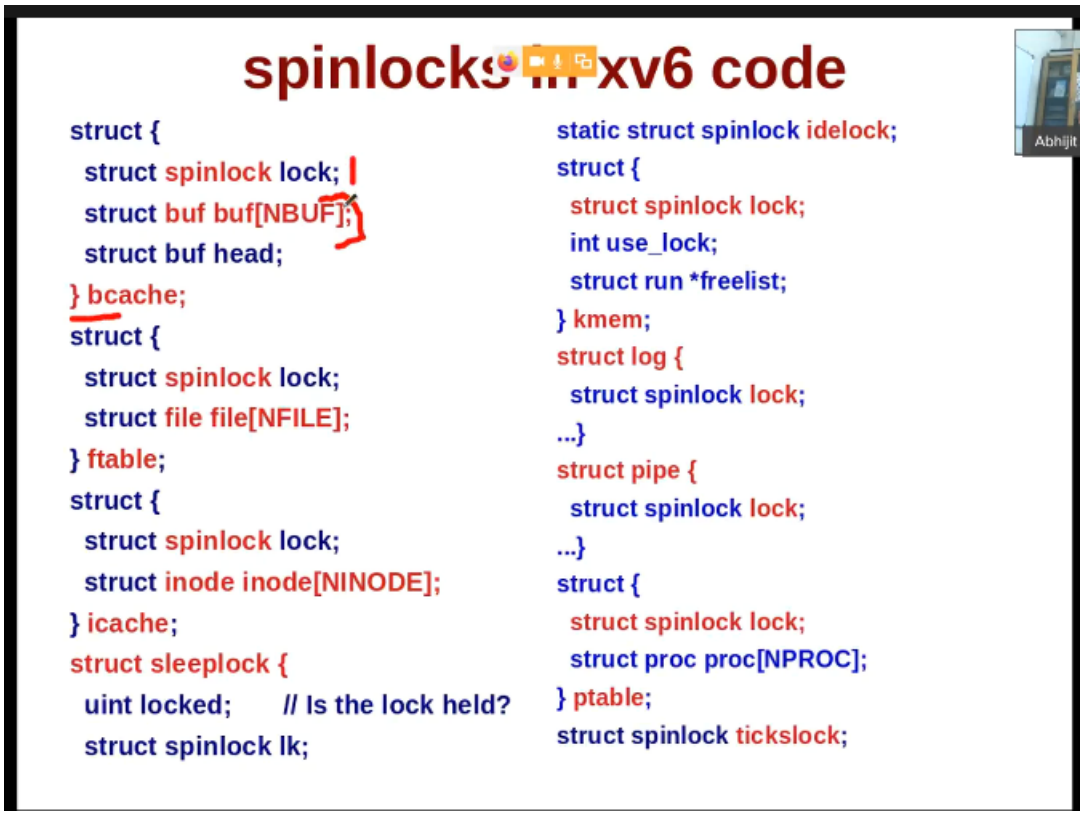
Has both locks : spinlock and sleeplock

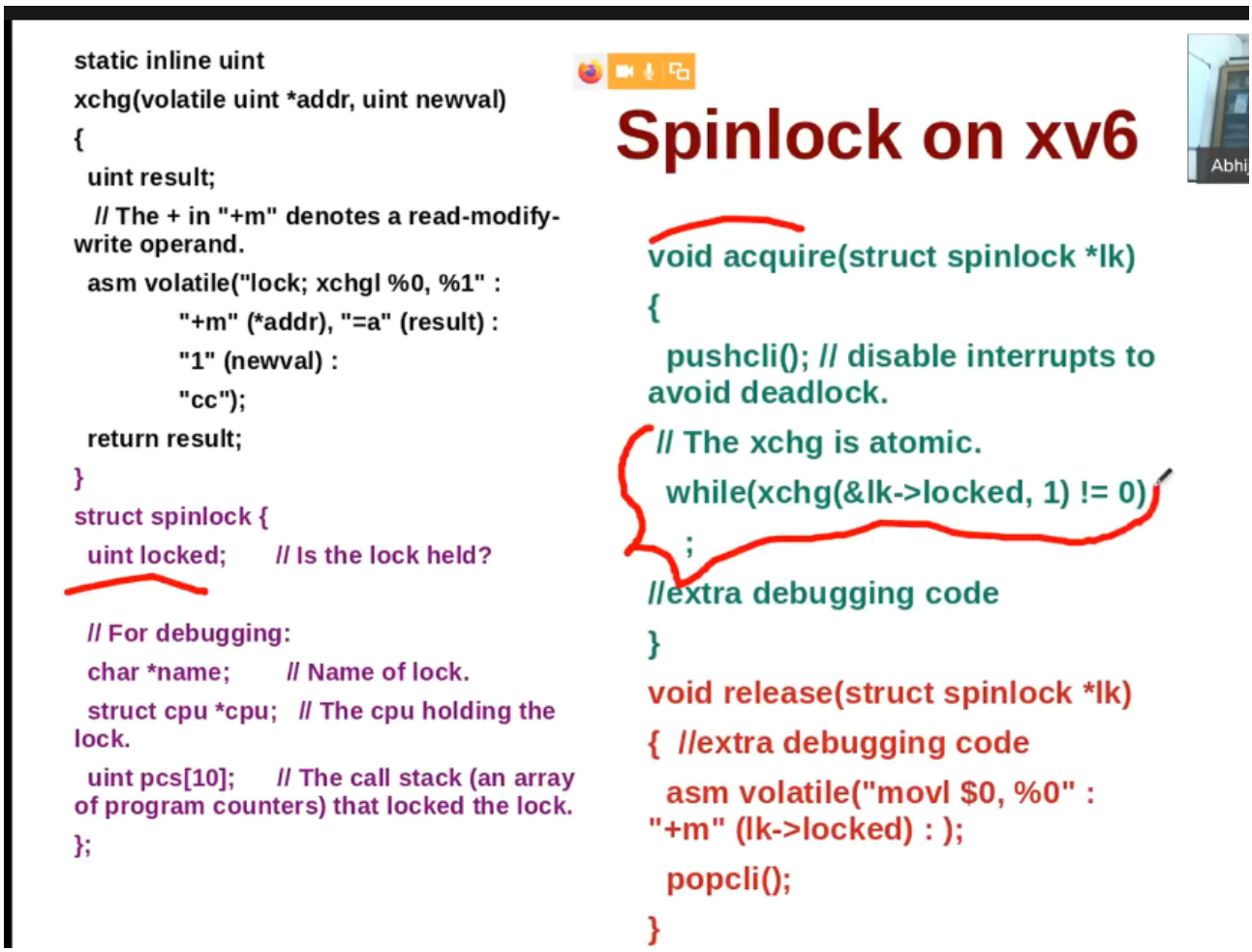
```
/*
struct spinlock {
  uint locked;       // Is the lock held?

  // For debugging:
  char *name;        // Name of lock.
  struct cpu *cpu;   // The cpu holding the lock.
  uint pcs[10];      // The call stack (an array of program counters)
                     // that locked the lock.
};

*/
```

spinlocks in xv6





in acquire (spinlock *lk):

pushcli() : get eflags : readflags() which pushes flag onto stack
cli : diable interr
increment ncli in mycpu

if holding(lk) :
holding : pushcli
check if locked is set and check if other proc is is holding a lock : then it panics
while (xchg(&lk->locked,1) : actual locking
syn_schronise :
lk->cpu = mycpu() // note down where lock is being used
getcallerpcs : setting values in pcs array
pcs gets ebp values .i.e. return values
which is used in panic

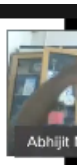
in release
if !holding(lk) :
panic
lk->pcs[0] = 0
lk-> cpu = 0
asm volatile("movl $0, %0" : "+m" (lk->locked) :); // actual release code

```
Void acquire(struct spinlock *lk)
{
  pushcli(); // disable interrupts to avoid deadlock.
  if(holding(lk))
    panic("acquire");
    ......
void pushcli(void)
{
  int eflags;

  eflags = readeflags();
  cli();
  if(mycpu()->ncli == 0)
    mycpu()->intena = eflags & FL_IF;
  mycpu()->ncli += 1;
}
static inline uint
readeflags(void)
{
  uint eflags;
  asm volatile("pushfl; popl %0" : "=r" (eflags));
  return eflags;
}
```

spinlocks

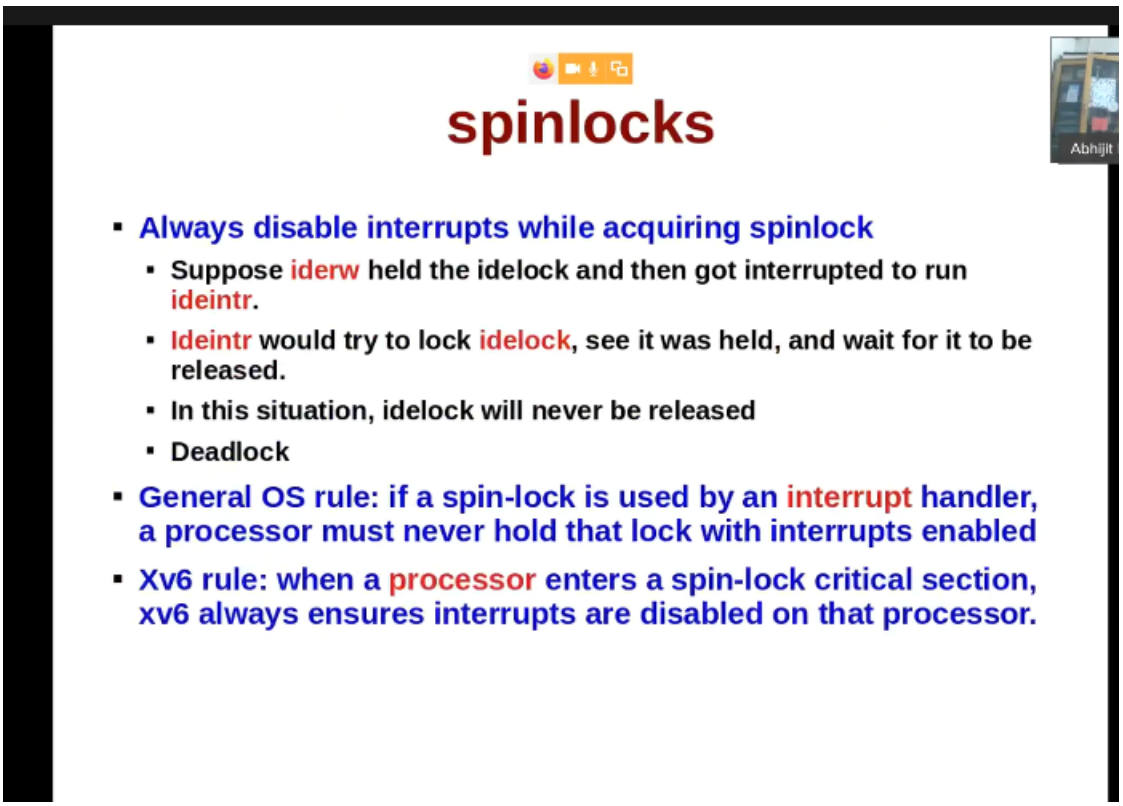
- Pushcli() - disable interrupts on that processor
- One after another many acquire() can be called on different spinlocks
  - Keep a count of them in mycpu()->ncli



```
void
release(struct spinlock *lk)
{
...
    asm volatile("movl $0, %0" : "+m" (lk->locked) : );
    popcli();
}
.
Void popcli(void)
{
  if(readeflags()&FL_IF)
    panic("popcli - interruptible");
  if(--mycpu()->ncli < 0)
    panic("popcli");
  if(mycpu()->ncli == 0 && mycpu()->intena)
    sti();
}
```

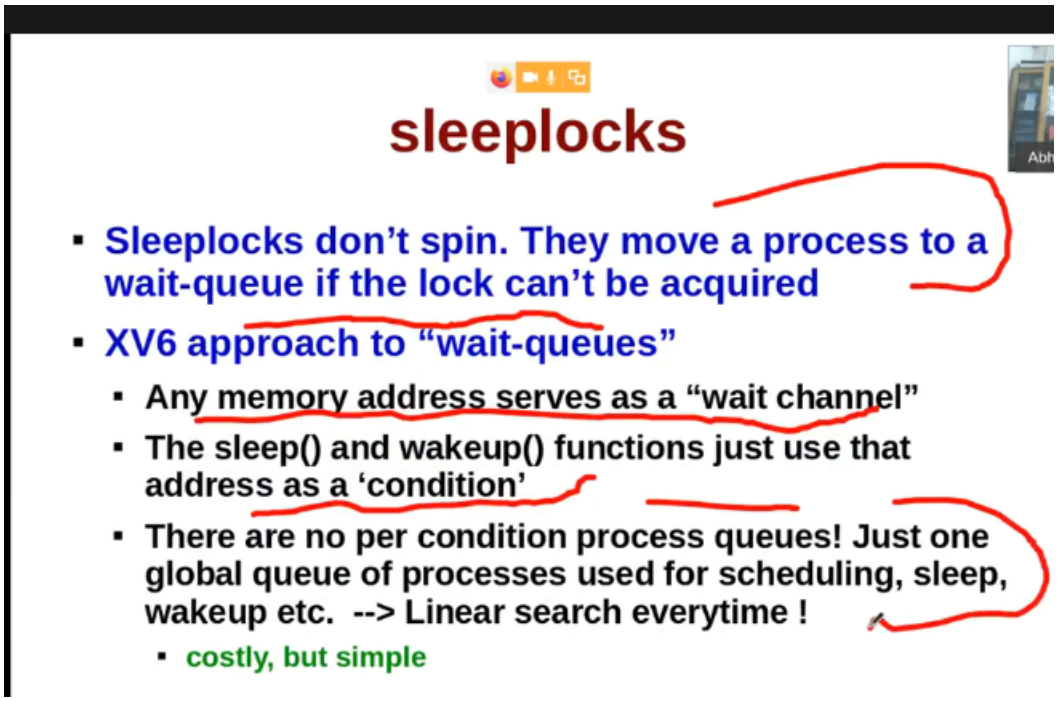
spinlocks

- Popcli()
  - Restore interrupts if last popcli() call restores ncli to 0 & interrupts were enabled before pushcli() was called



idelock is never released : because

When chan is not NULL
state has to be WAITING



sleep()


```
void
sleep(void *chan, struct spinlock *lk)
{
  struct proc *p = myproc();
....
  if(lk != &ptable.lock){
    acquire(&ptable.lock);
    release(lk);
  }
  p->chan = chan;
 p->state = SLEEPING;
  sched();
  // Reacquire original lock.
   if(lk != &ptable.lock){
    release(&ptable.lock);
    acquire(lk);
  }
```

- At call must hold lock on the resource on which you are going to sleep
- since you are going to change p-> values & call sched(), hold ptable.lock if not held
- p->chan = given address remembers on which condition the process is waiting
- call to sched() blocks the process

actually implemented
so sleep is called on a particular