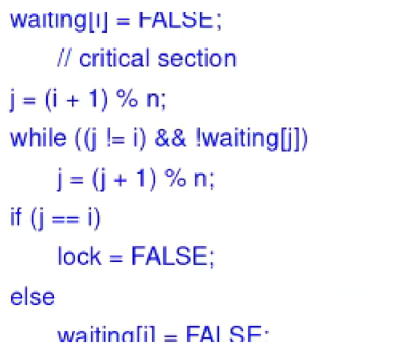
SpinLOCK

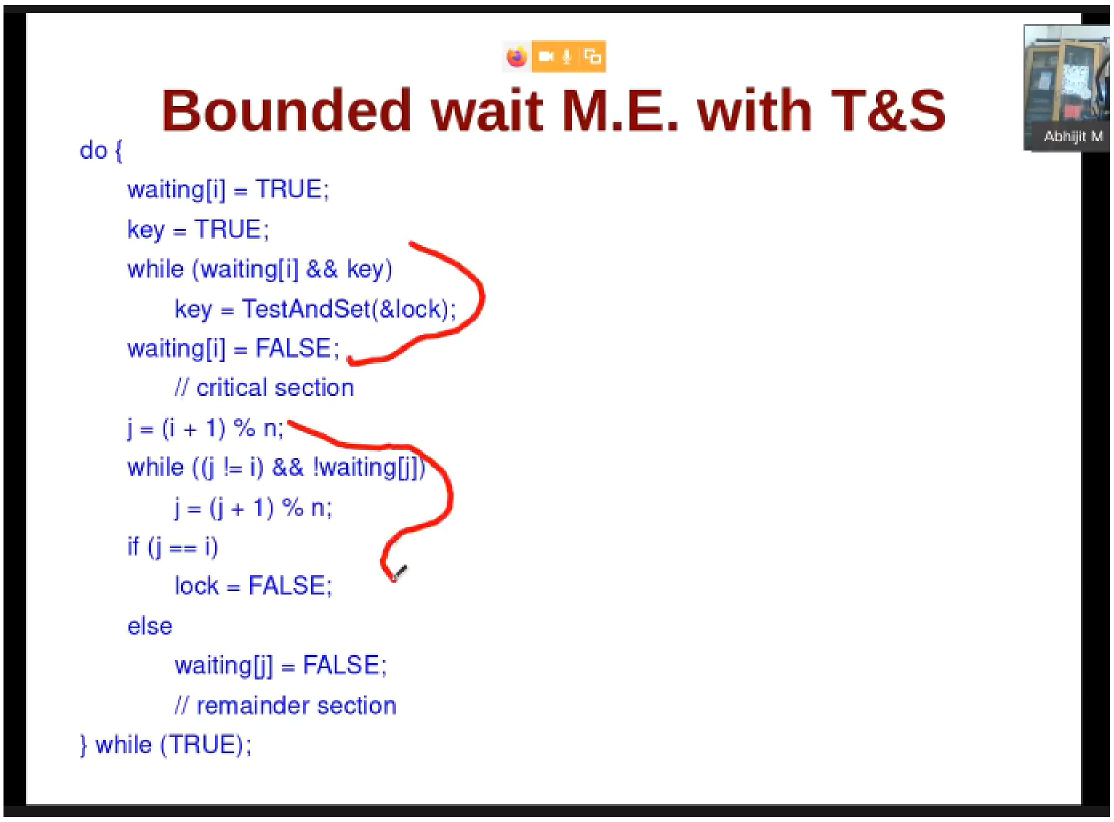
Lock implemented to do busy-wait

using instr like Test and set or swap



```
waiting[i] = FALSE;
    // critical section
j = (i + 1) % n;
while ((j != i) && !waiting[j])
    j = (j + 1) % n;
if (j == i)
    lock = FALSE;
else
    waiting[j] = FALSE;
```

Bounded wait with Mutual Exclusion with test and set



Bounded wait M.E. with T&S

```
do {
    waiting[i] = TRUE;
    key = TRUE;
    while (waiting[i] && key)
        key = TestAndSet(&lock);
    waiting[i] = FALSE;
        // critical section
    j = (i + 1) % n;
    while ((j != i) && !waiting[j])
        j = (j + 1) % n;
    if (j == i)
        lock = FALSE;
    else
        waiting[j] = FALSE;
        // remainder section
} while (TRUE);
```

Some thumb rules of Spinlock :

1. never block proc holding a spinlock
2. Hold a spinlock for a short duration of time
   - prefered on multi-proc
   - cost of context swtch is concern on slp-wt lock
   - short =< 2 contxt swtch

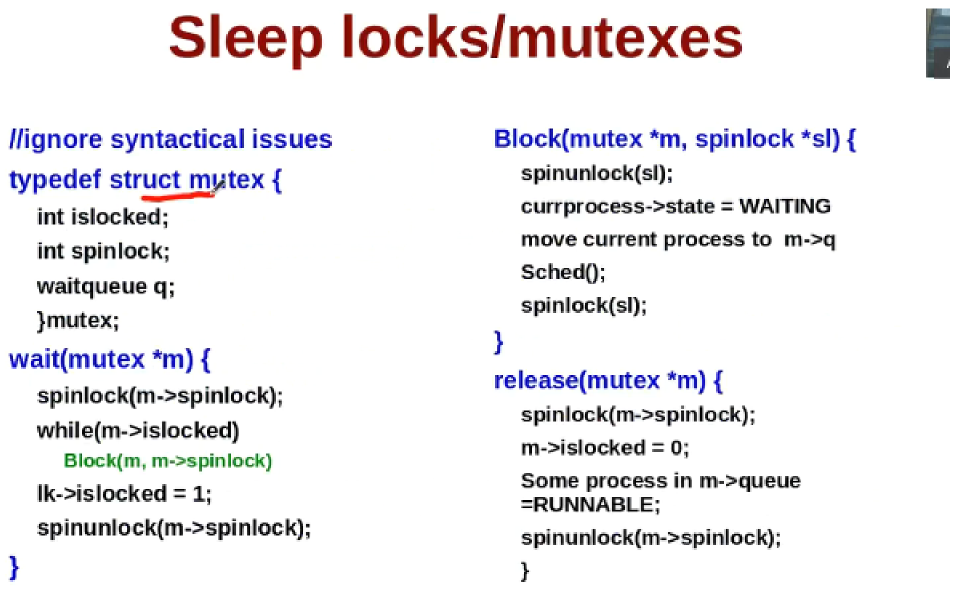
cost of contxt switch is actually reduced as other proc may doing busy-wait on proc, then when one leaves critical sect other may enter
// preferable when **multi – proc system**

```
while () {
        unlock; schedule(); lock;
}
```

Sleep-locks:

spin locks results on busy-wait
CPU cycle wasted by waiting proc/thrd

solution: thrd wait till lock is available. Thrd in wait Q. Thrd holding lck wake up one of them when critical sect is over.



Sleep locks/mutexes

```
//ignore syntactical issues
typedef struct mutex {
    int islocked;
    int spinlock;
    waitqueue q;
    }mutex;
wait(mutex *m) {
    spinlock(m->spinlock);
    while(m->islocked)
        Block(m, m->spinlock)
    lk->islocked = 1;
    spinunlock(m->spinlock);
}
```

```
Block(mutex *m, spinlock *sl) {
    spinunlock(sl);
    currprocess->state = WAITING
    move current process to  m->q
    Sched();
    spinlock(sl);
}
release(mutex *m) {
    spinlock(m->spinlock);
    m->islocked = 0;
    Some process in m->queue
    =RUNNABLE;
    spinunlock(m->spinlock);
    }
```

Islocked : 0 or 1 // if proc is holding lock or not

spinlock :
whenever there is race it has to be protected.

Any synchronisation soltuion involve use of orimitive atomic test and set

1st level solution we have using T&s is spinlock.

When proc is wanting to enter critical section, it calls wait. For multiple proc calling wait, only one of them should be waited.
Here it checks if islock is true. If true then it will block the process. And set the isLocked to 1
Here we have a race condition, so have to protect isLock. Thus we have to use spinlock in the wait to protect isLock variable.

In block, first spinlock is released as defined in the thumbrules. spinunlock()
Change state to waiting, then move pro to wait queue.

Then schedule
before exiting then we should get spinlock again as we checking it in the wait.

When proc is about to exit critical section then it calls release.

```
    Get spinlock
    Set isLocked to 0
    make proc runnable
    release lock . spinunlock()
```