Locking

- Advisory
- Locks do not fail (no error return)
- " " not have timeouts
- Locking & atomicity → ?

Atomicity — [indivisible execution

locking → does not provide atomicity
→ but looks like it
→ provides mutual excl
or critical sections

How to lock?

use → Lock → spin ✓

→ mutex → a lock that blocks

Semaphores → mutex lock + more

P(Sem) → while(Sem == 0);
Sem--

Sem = int

Implementation ?

Semaphore data structure
- → counter
- → queue (of PCBs or TCBs → process control block or thread CB)
- → spin lock // optional

P(Sem) { → spinlock(sched_lock)
Sem.count--;
if (Sem.count < 0)

"block" ← { ask scheduler to block me, put my PCB in Sem.Q & run something else & unlock(sched_lock)

(not sleep)
(not yield)

} else { <nothing>; → unlock(sched_lock) }

[sched_lock] ← global

block → block forever

sleep → block for some time

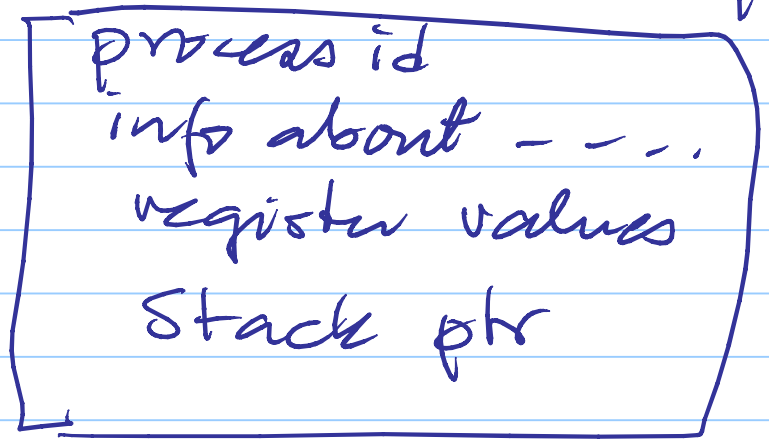
yield → block, till next scheduling turn comes

```
V(Sem) {  spin_lock(sem_lock)
          Sem.count++
          if (Sem.count <= 0)
             { wake up one process
               or thread from Sem.Q }
             unlock(sem_lock)
          else → unlock(sem_lock)
                 ~~nothing~~ }
```

Scheduler —

↳ context switch .. [function]

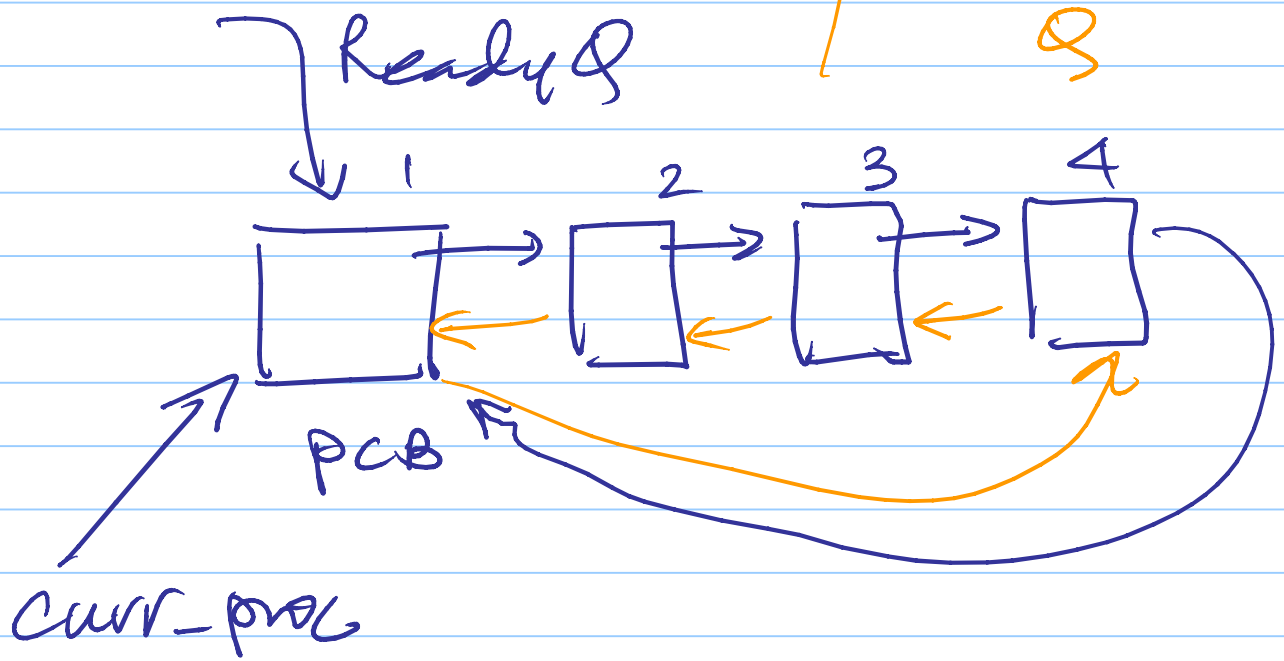
→ PCB
TCB]



← data structure in kernel

Scheduling Q

→ circular doubly linked Q



Timer

→ int → local vars:

PCB_type from, to;

from = current_proc

to = 2nd proc in Q

rotate the Q

[Context switch

<inline code>

Ret from int

Context switch

hardware registers

```
move regs to from^.registers;
move SP to from.SP
move to^.registers -> registers
move to^.SP -> SP
return from interrupt
```

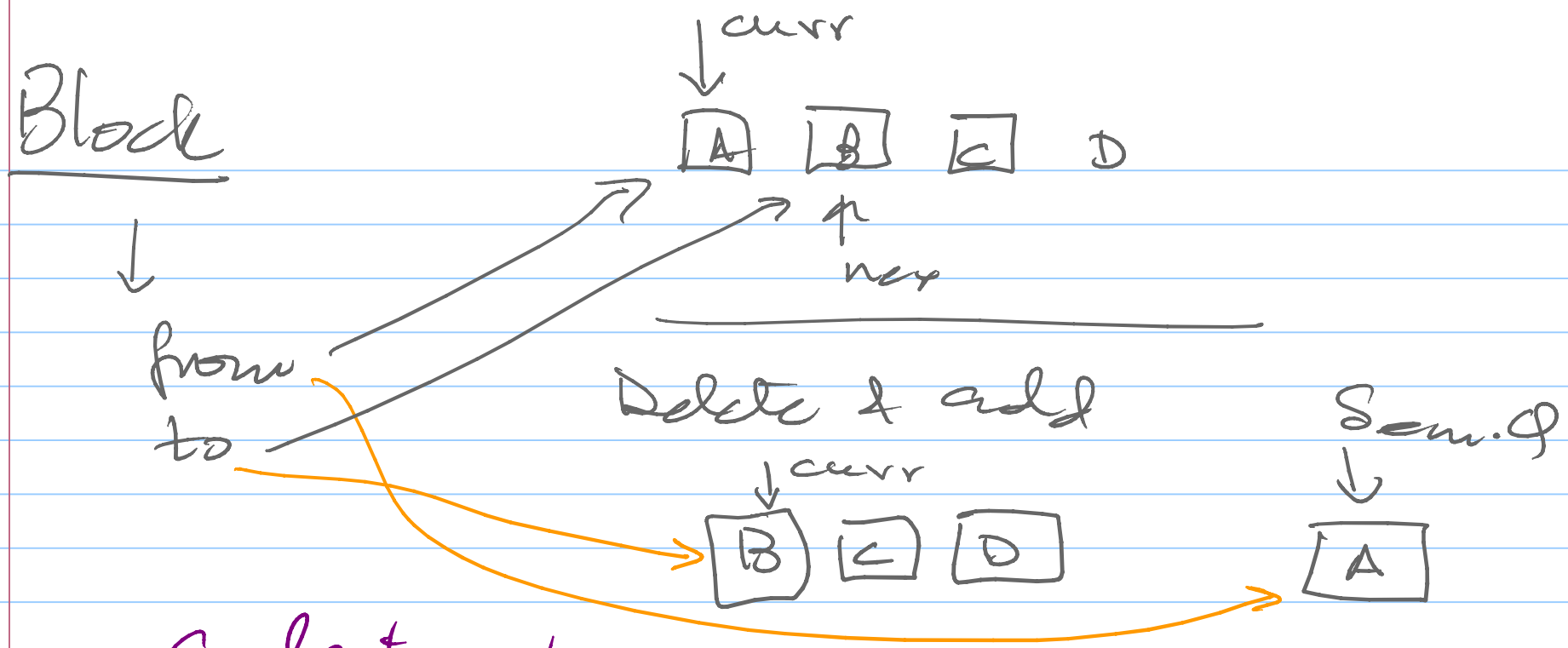
P(s) { lock
S.count--
if (C) {

block

from = current
to = next
tempPCB = delete from ready Q
Add tempPCB to Sem.Q
Context_switch_code

put the unlock before return from INT

} else unlock() }



Context switch from → to

↳ save [regs & SP] → from PCB

load ← to PCB

unlock

return

```
V(Sem) { sp_lock(sem_lock)
         Sem.count ++
         if ( )
            { temp ← delete 1 PCB
                     from Sem.Q
              Add temp to ready Q
            }
         unlock(sem_lock); }
```

P(sem)
V(sem)
CS
P(sem)
V(sem)
CS
Ready
Running