- scheduler (user level)
- locking & synch in user level scheduler
  - semaphores (one option)

Semaphores

Scheduler

- start_thread
- yield
- (run)

↓

type
variables that are global

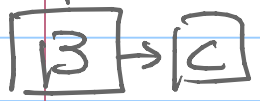
↳ sem_t → struct { int count;
q_t q;
}

```
Sem_t S1, S2;
initSem (S, i){  S.count = i;
                 initQ (S.Q); }

P(S) -> { S.count--
          if (S.count < 0)
                { AddQ(S.Q, DelQ(RunQ))
                  yield()
                }
          else
                }
```

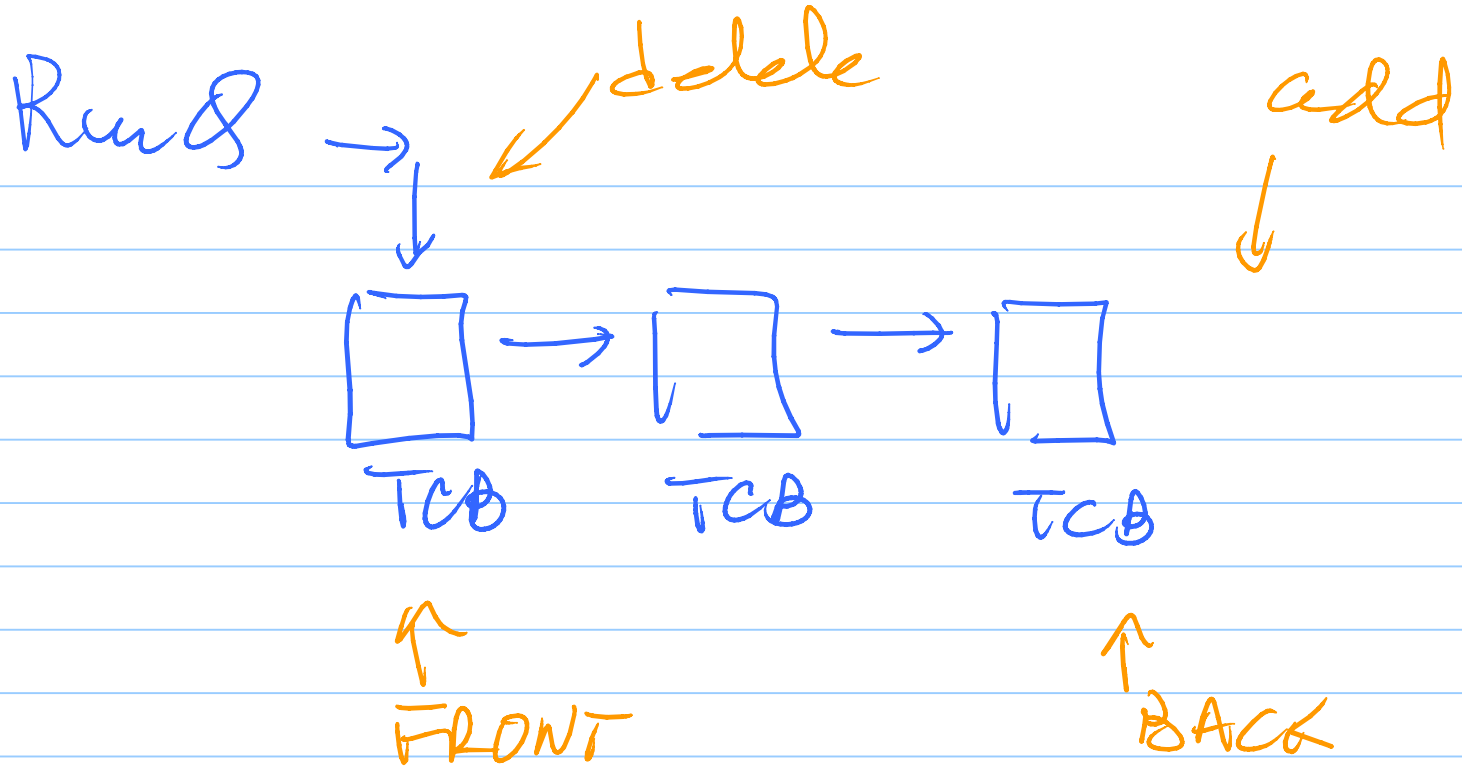
AddQ(S.Q, DelQ(RunQ))

yield()

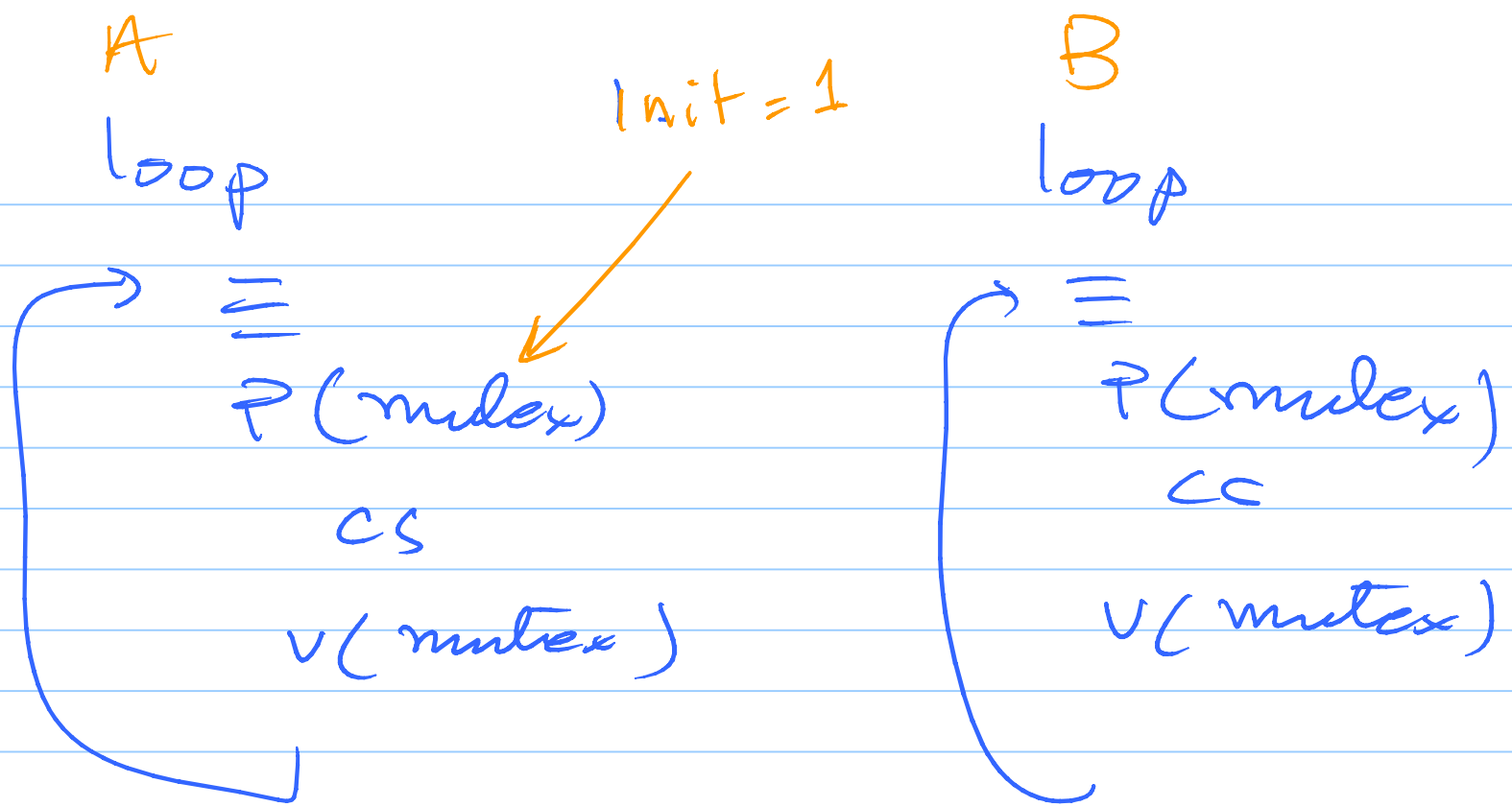


from = lastTCB of S.Q
to = 1st TCB of RunQ
swapcontext(from, to)

from = 1st thread in RunQ
to → 2nd thread
Rotate
swapcontext(from, to)

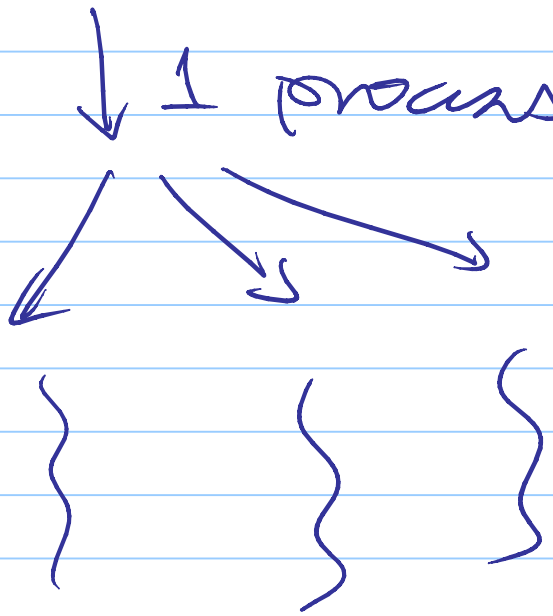

Run
delete
add
TCB
TCB
TCB
FRONT
BACK

```
V(s)
{ s.count++
  if(s.count <= 0)
      { AddQ(RunQ, DelQ(S.Q)}
  → yield() // not needed?
}
```


A
loop
P(mutex)
CS
V(mutex)
Init = 1
B
loop
P(mutex)
CC
V(mutex)

- Code should be written with assumption threads are preempted
  - critical sections needed, use semaphores init to 1
  - synchronization → use sem
    - NO 'yield' necessary

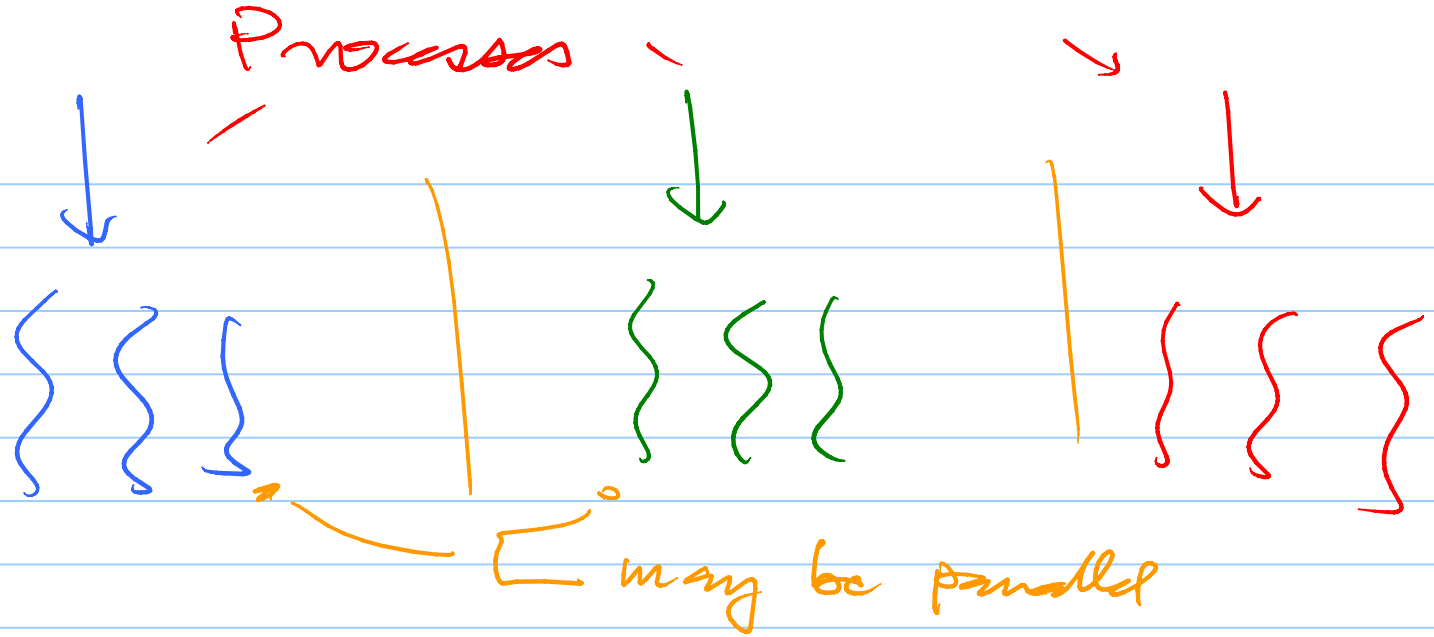
User level scheduling



multiple threads

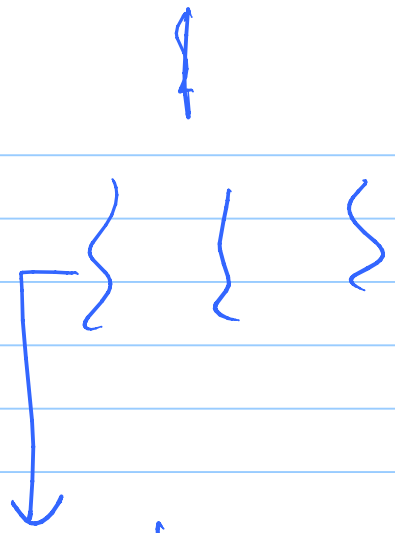
← all run by a single process

→ switch by yield



user scheduler

kernel "



system call

↳ block (I/O wait)

User level threads

- 1 process
- if a thread blocks (in kernel) all threads block
- No parallel execution → even on multicores
- fast switching, no kernel calls

Kernel level threads

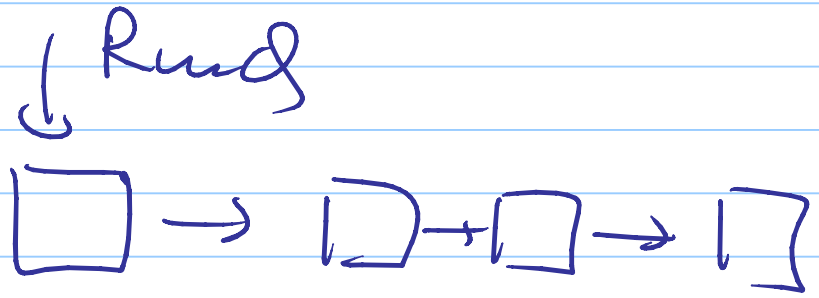
- → 1 scheduler in kernel
- → startthread (or pthread_create or clone) → system call
  - ↳ creates TCB in kernel
    - → add to Run Q
    - → use same address space as parent

- multiple threads of same application can run in parallel

- thread → system call → block
  blocks only the thread & not other threads

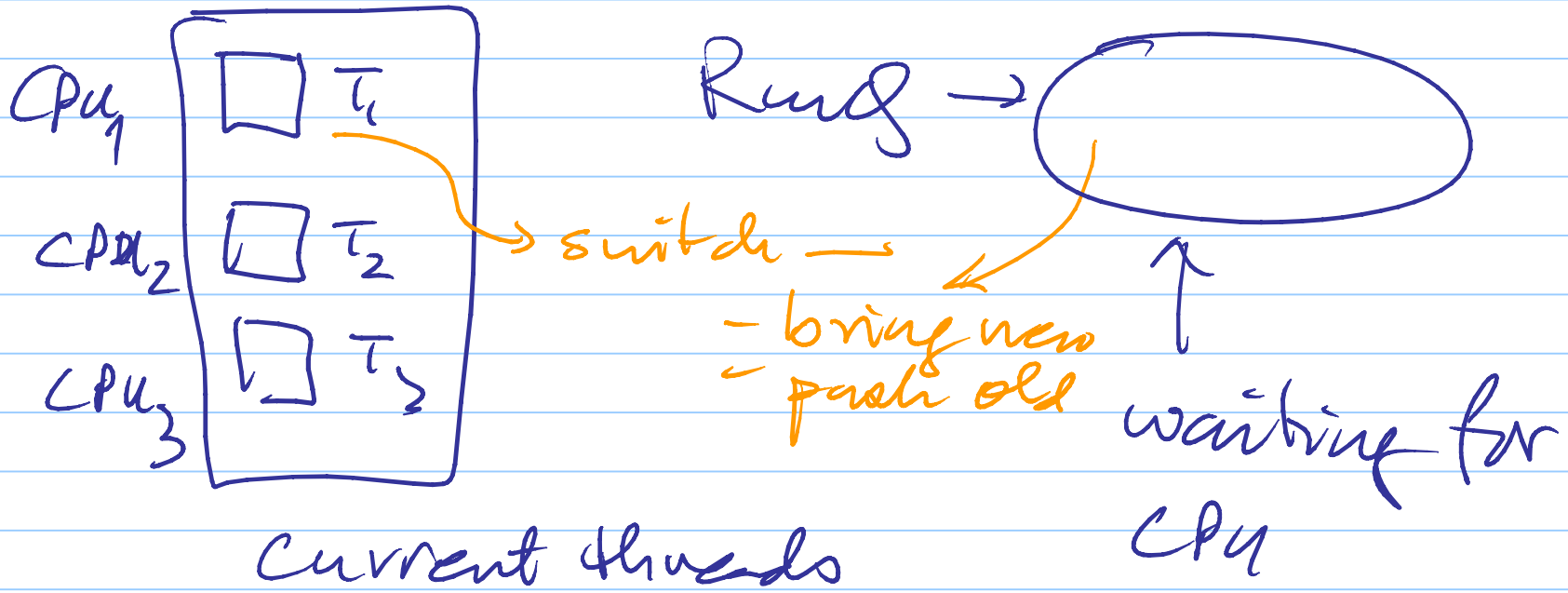
-

kernel scheduler (uniprocessor)

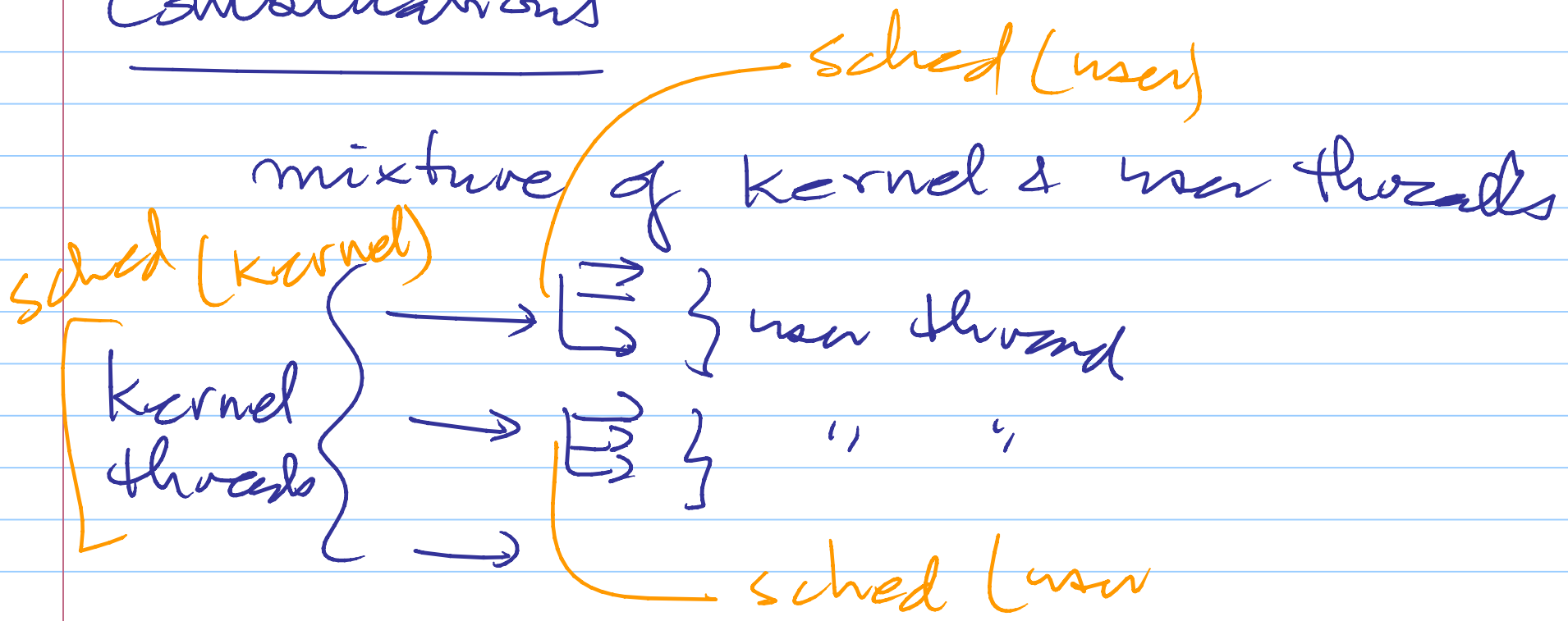


- yield = same, but in kernel
  → called on interrupt
- semaphores → in kernel

Multicore scheduling



Combinations



1 x N scheduling – 1 kernel thread runs N user threads

– 1 process may have multiple kernel threads

user kernel

N > M CPU

100 10 3

M x N scheduling

in each process

→ M kernel threads

→ N user threads

→ 1 scheduler

Tasks / Processes = threads (kernel)

↑ many diff