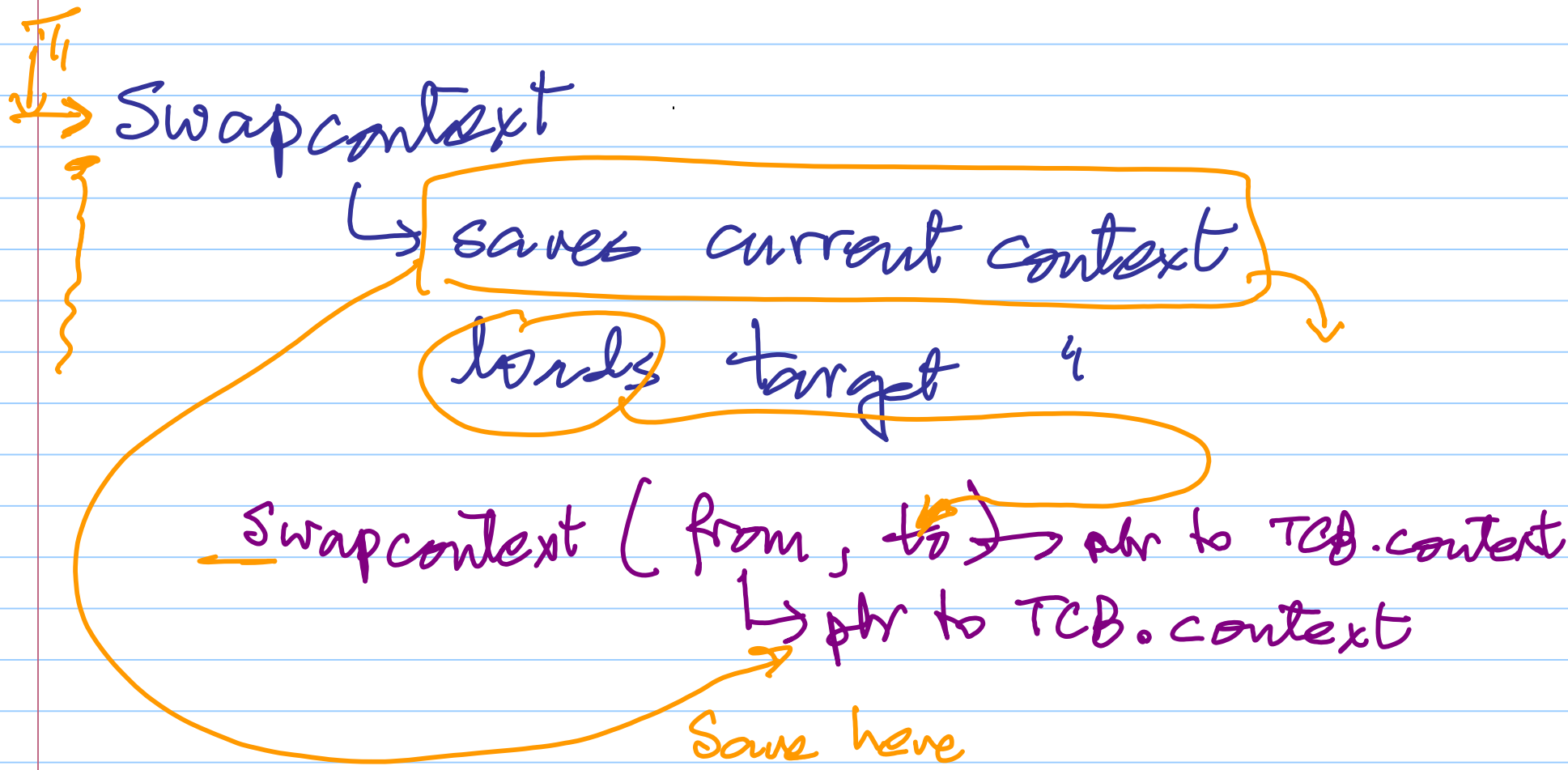
CSE 531: Distributed and Multiprocessor Operating Systems

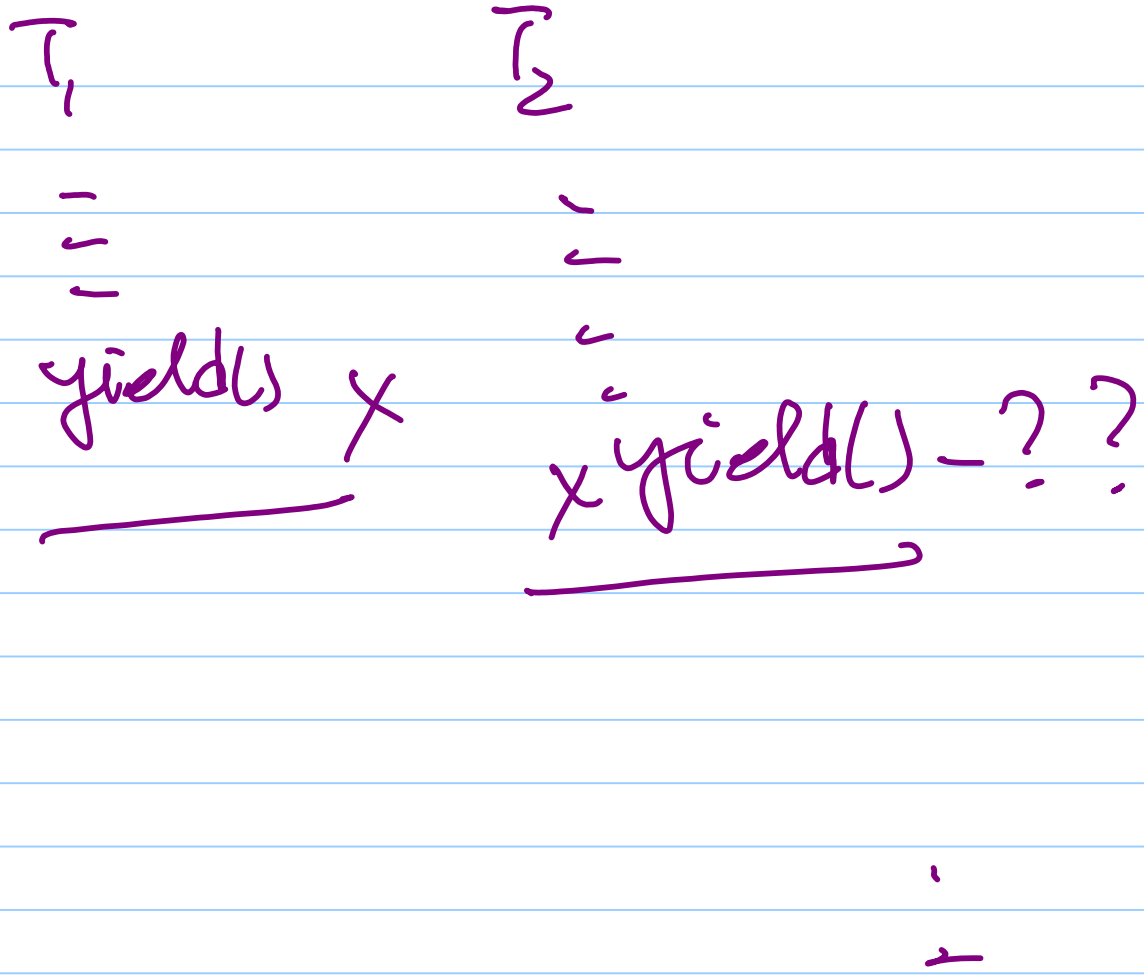
Class: 13 Date: 10/7

Fall 2020

Note Title



T1
swapcont(x, y)
from
swapcont(x, y)
T2
swapcont(y, x)

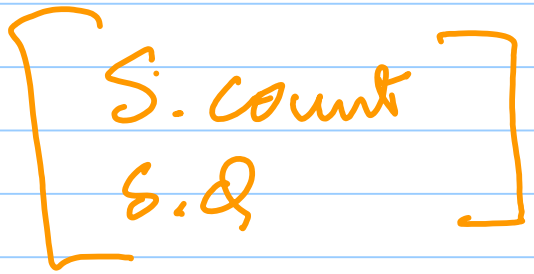

T1
T2
yield() X
X yield() - ??

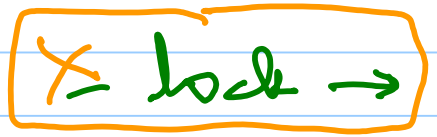
Semaphore - integer → 0 or >0

Semaphore - data type

↳ count ← integer

queue ← ptr to a TCB q.





initSem(&S, v)

[S.count = v

[S.Q ← initQ ← set to null

→

*TCB_t

→

Sem_t → [int count
          q q_t

P(Sem) → S.count --

V(Sem)

P(Sem)

{

sem.count --

if(sem.count < 0)

{

x = DelQ(RunQ)

AddQ(x, Sem.Q)

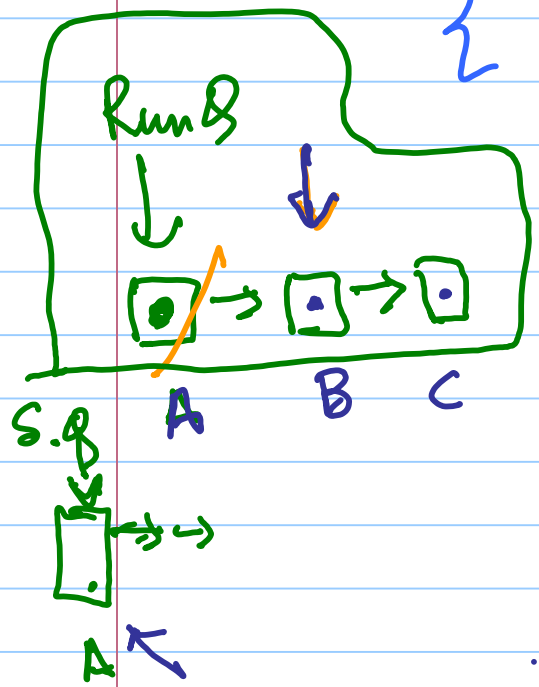
yield() → save to beg of RunQ
load from next

Save to B & load C

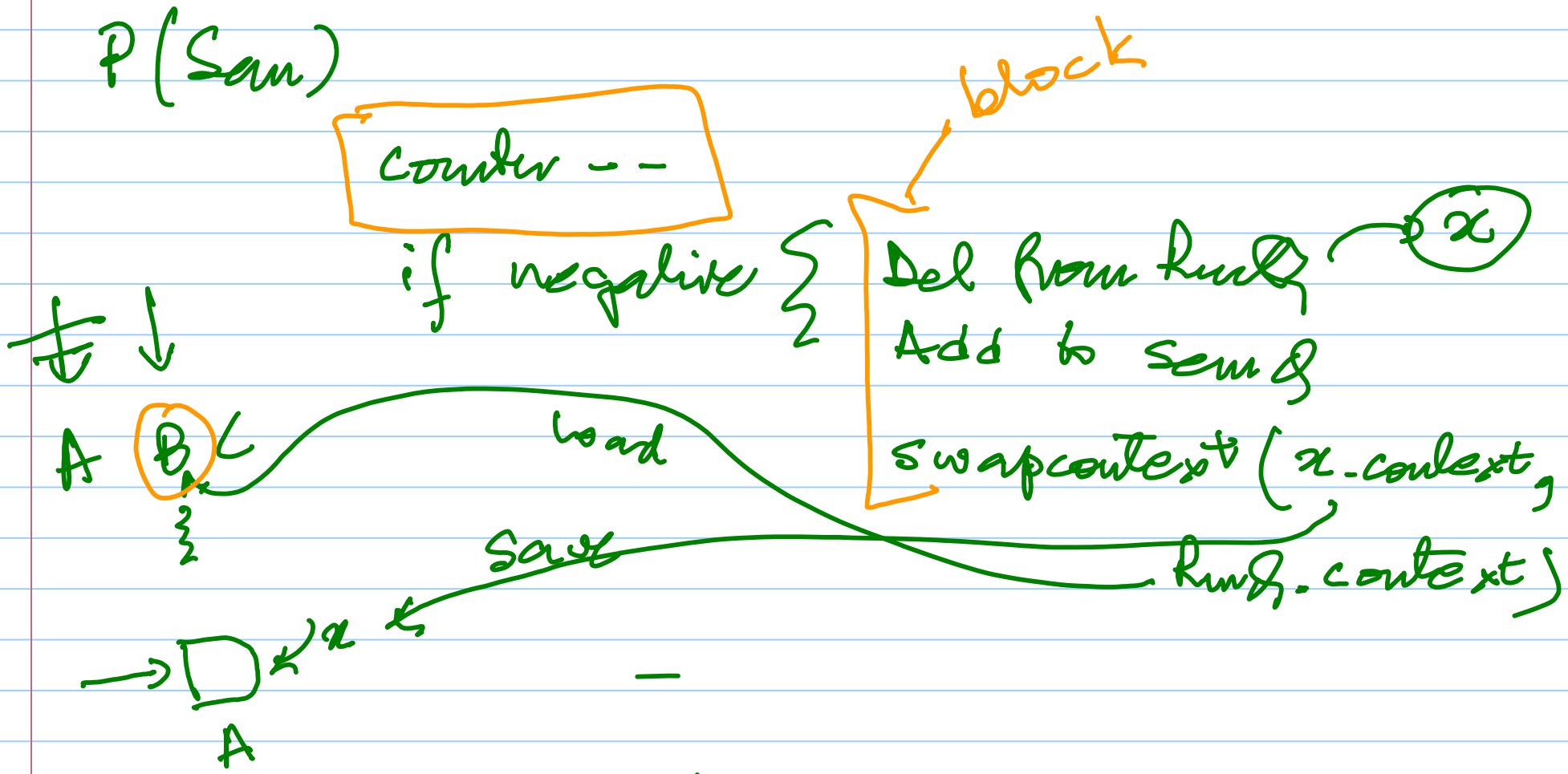
}

else → nothing

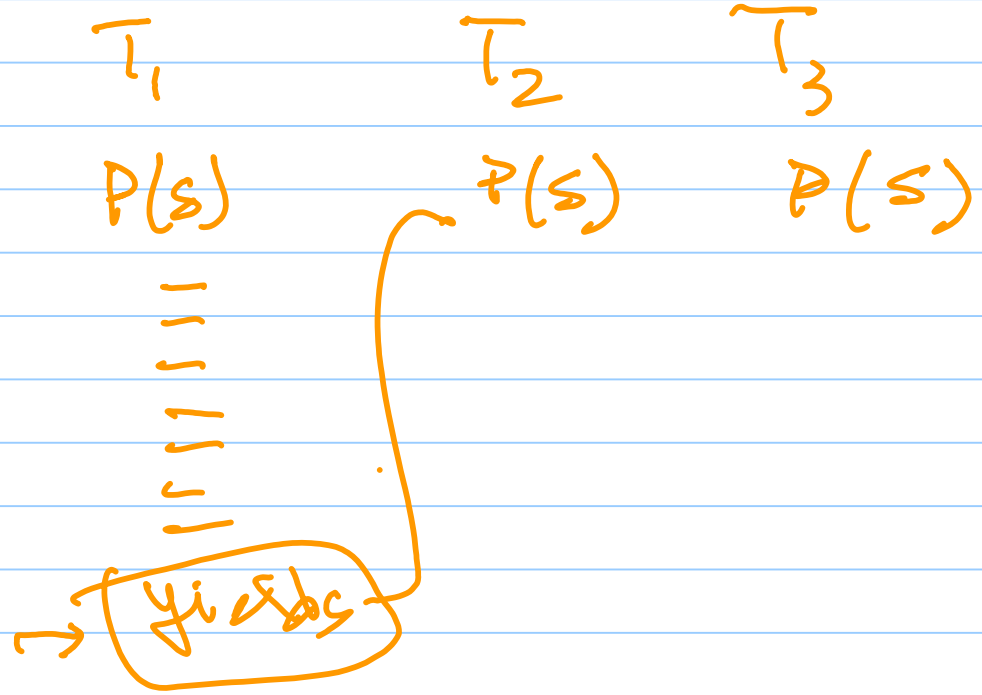
}



block this thread
↳ save context


P(Sem)
counter --
block
if negative
Del Running
Add to sem
x
swapcontext(x.context,
Running.context)
load
save
A
B
C
A
x

$S = 1 \quad 0 \quad -1 \quad -2$

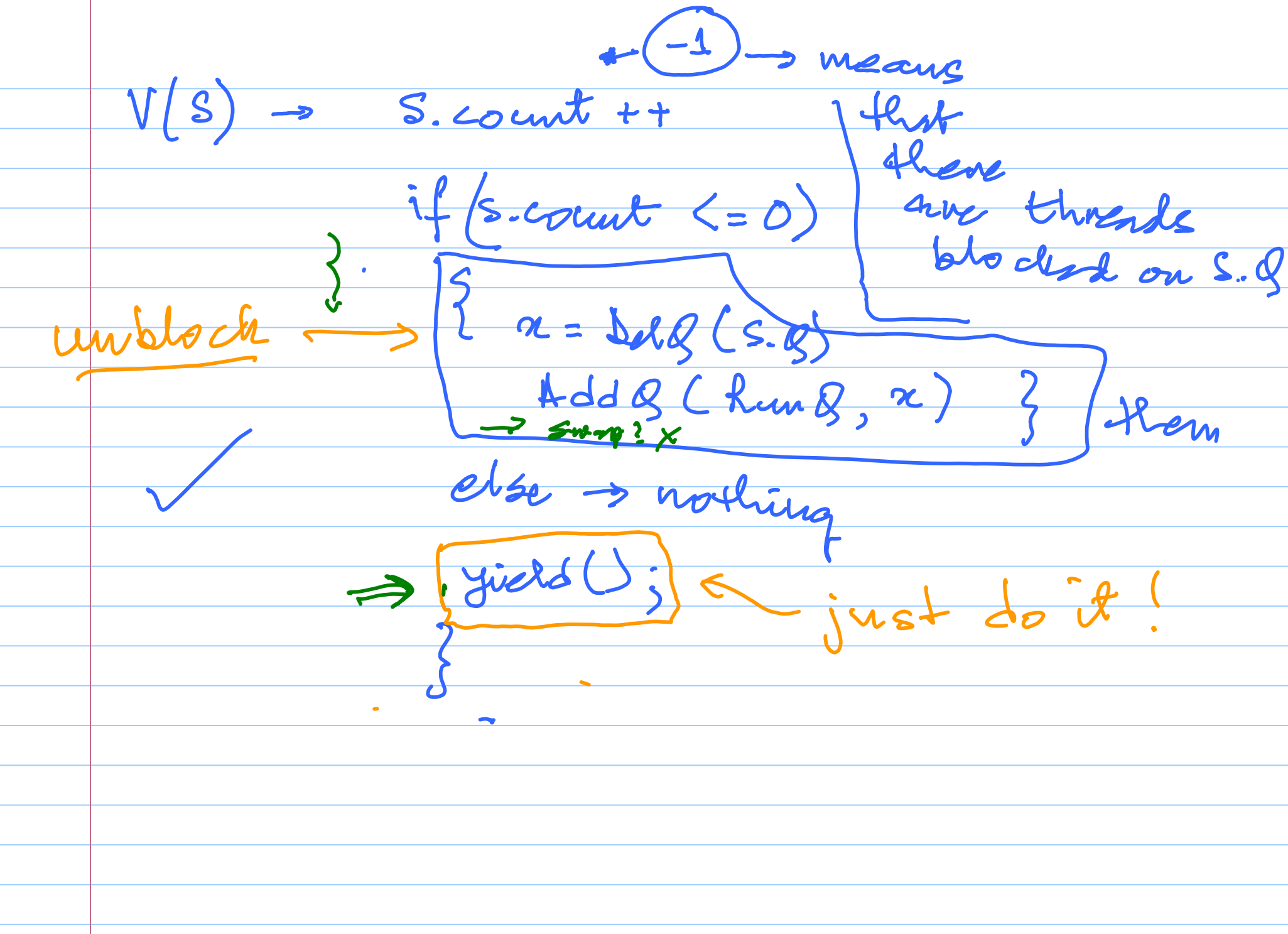


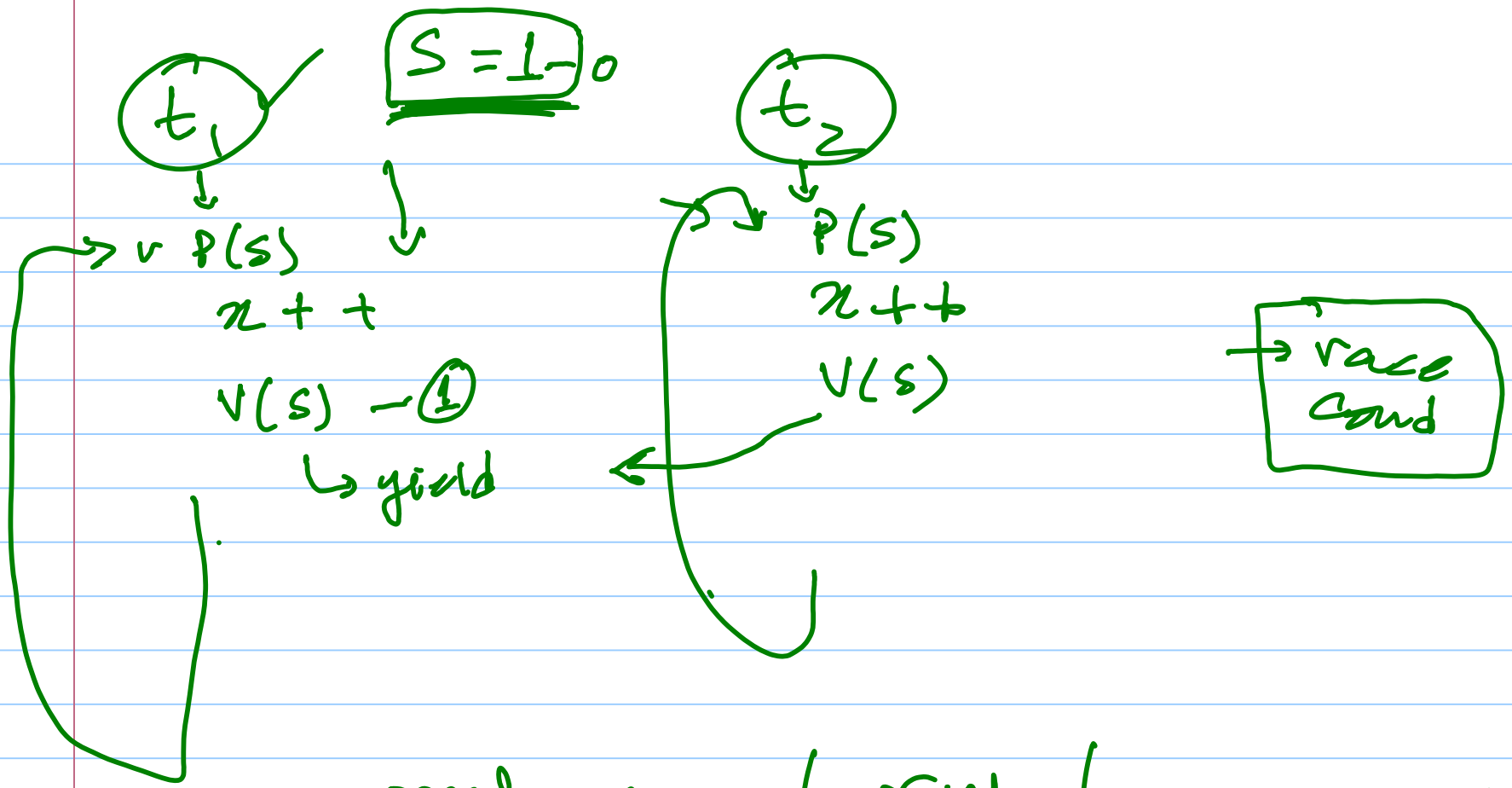
Prod | Cons

$S_1 = 0$ $S_2 = N$

P(S_1) → blocked

$= -1$


-1
means that there are threads blocked on S.Q
V(S) → S.count ++
if (S.count <= 0)
{ x = DelQ(S.Q)
AddQ(RunQ, x) }
then
unblock
else → nothing
yield();
just do it !



prod - cons / rw / ....

- Semaphore implementation for non-pre

- for preemptive → just add spin locks]

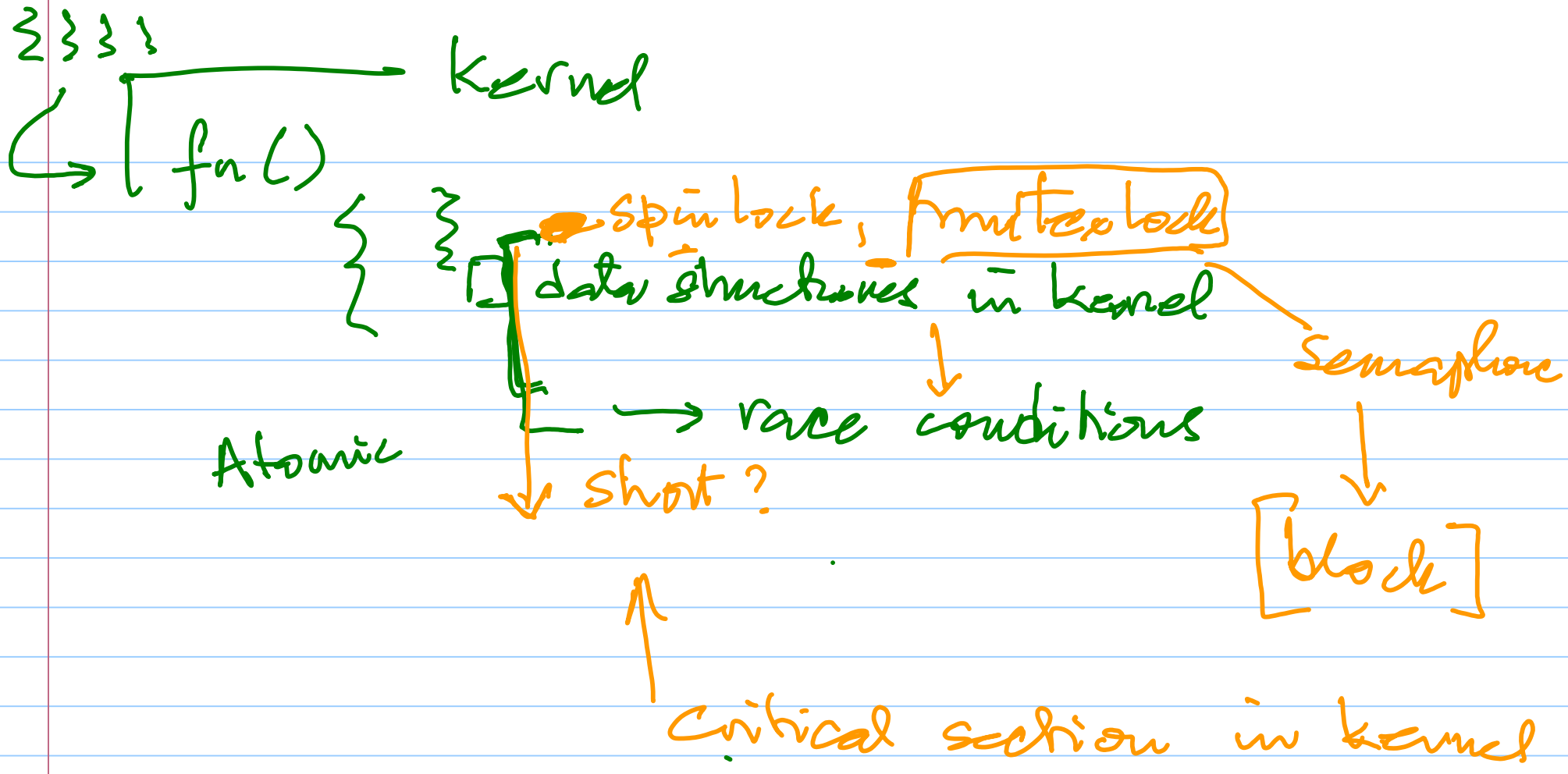
→

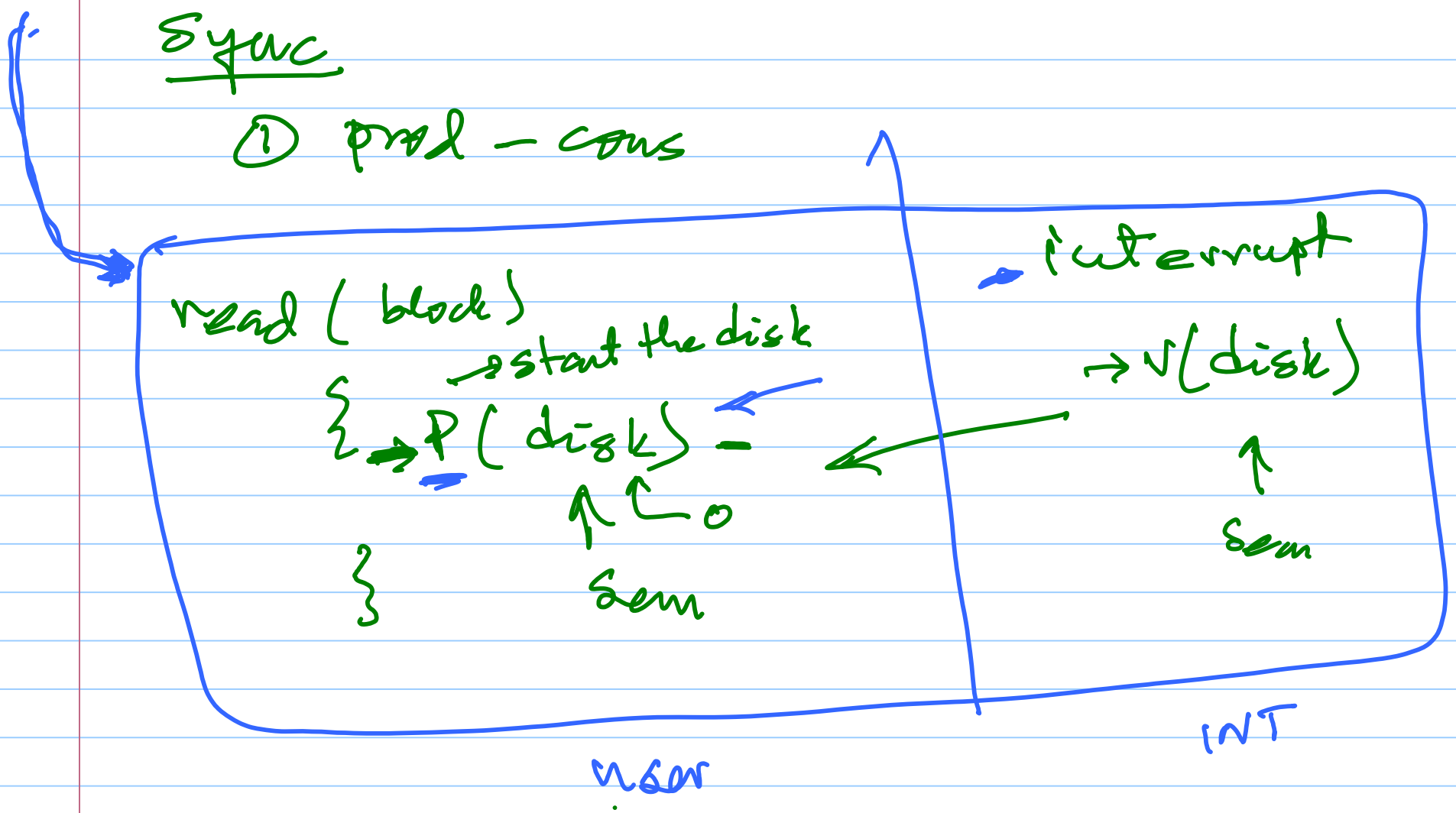
Semaphores

- → at user level
- → at kernel level

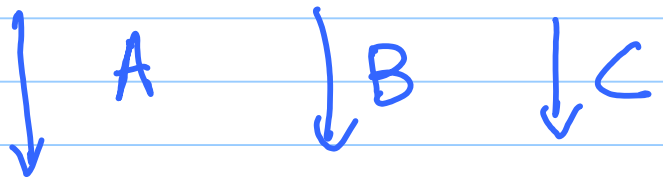
Kernel routines use semaphores

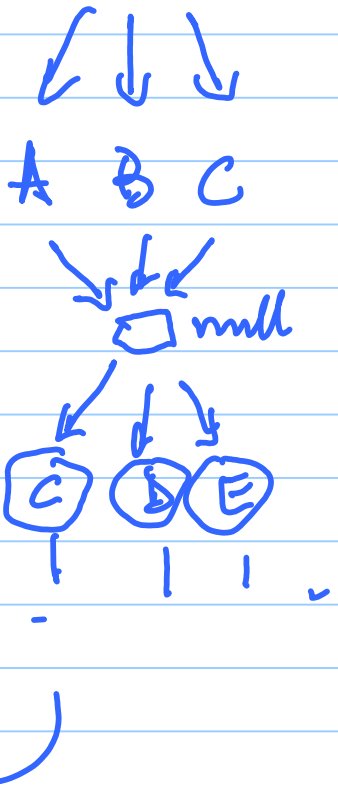
- → reentrancy
- → synchronization


kernel
fn()
spinlock, mutexlock
data structures in kernel
race conditions
Atomic
short ?
semaphore
block
Critical section in kernel


Sync
① prod – cons
read (block)
start the disk
P (disk)
0
Sem
interrupt
V (disk)
Sem
user
INT

BARRIER SYNC

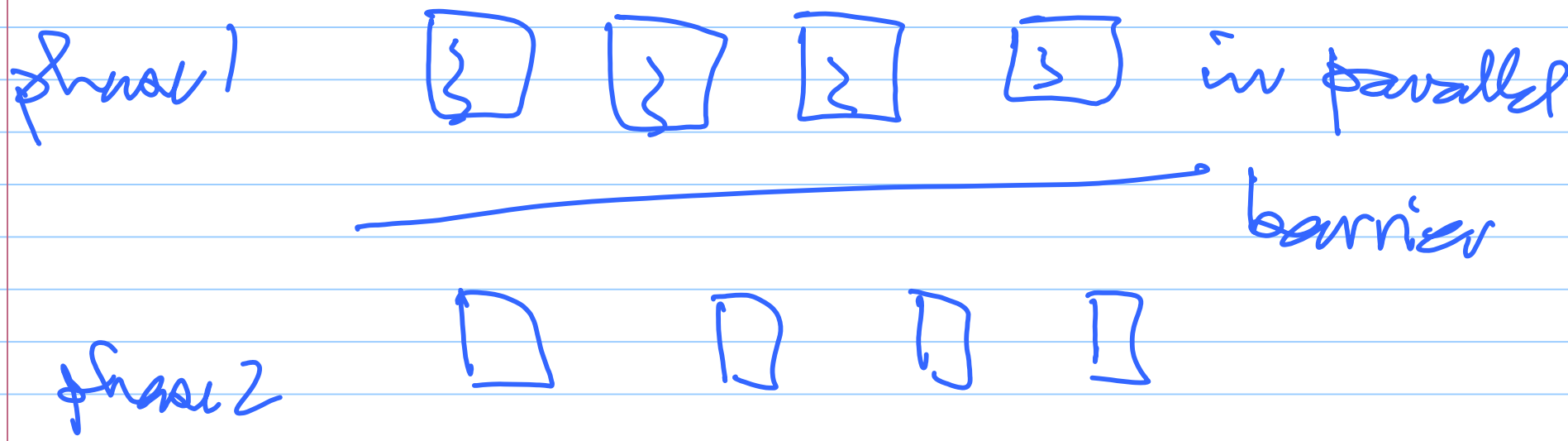


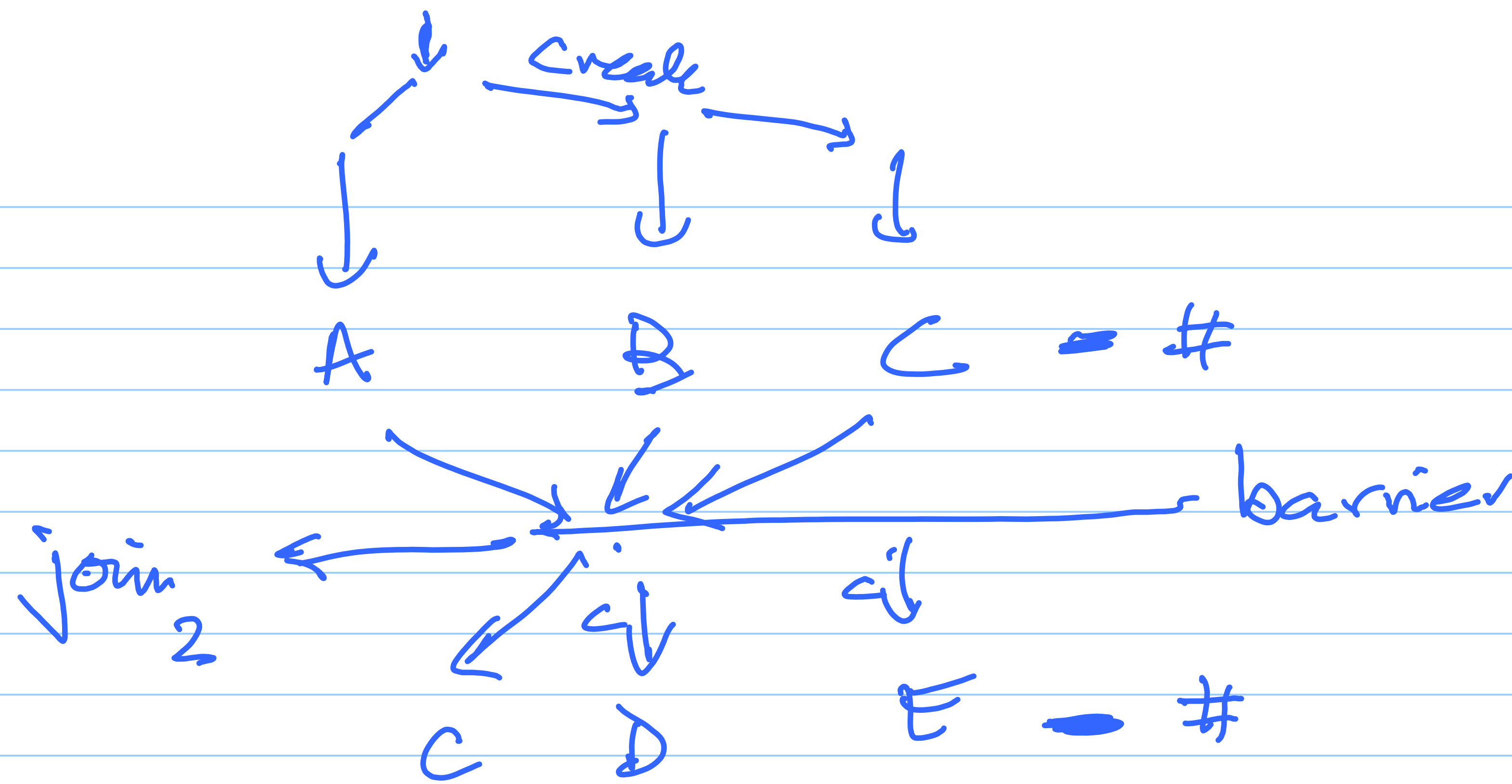


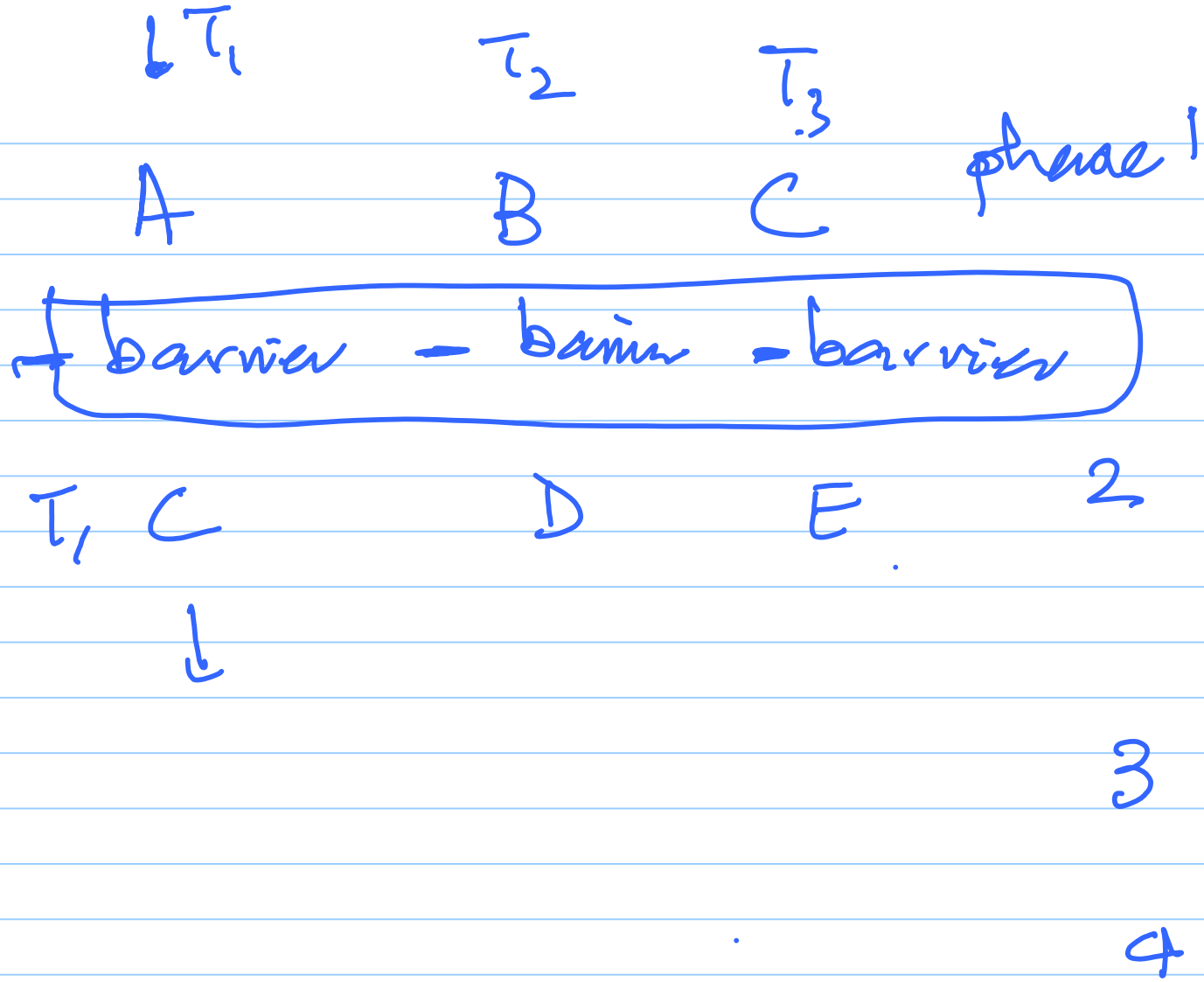
parallel

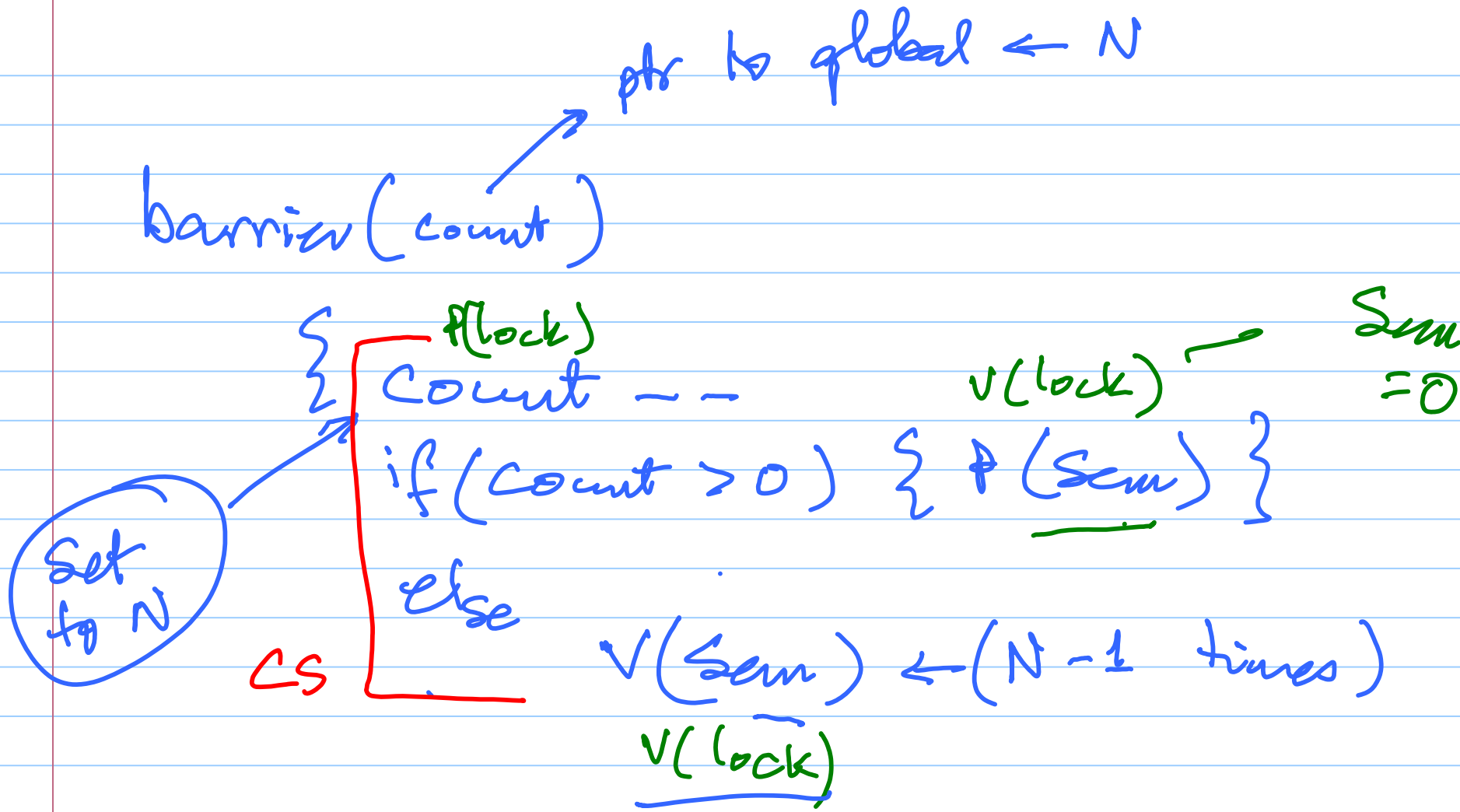
after ABC → do XYZ parallel

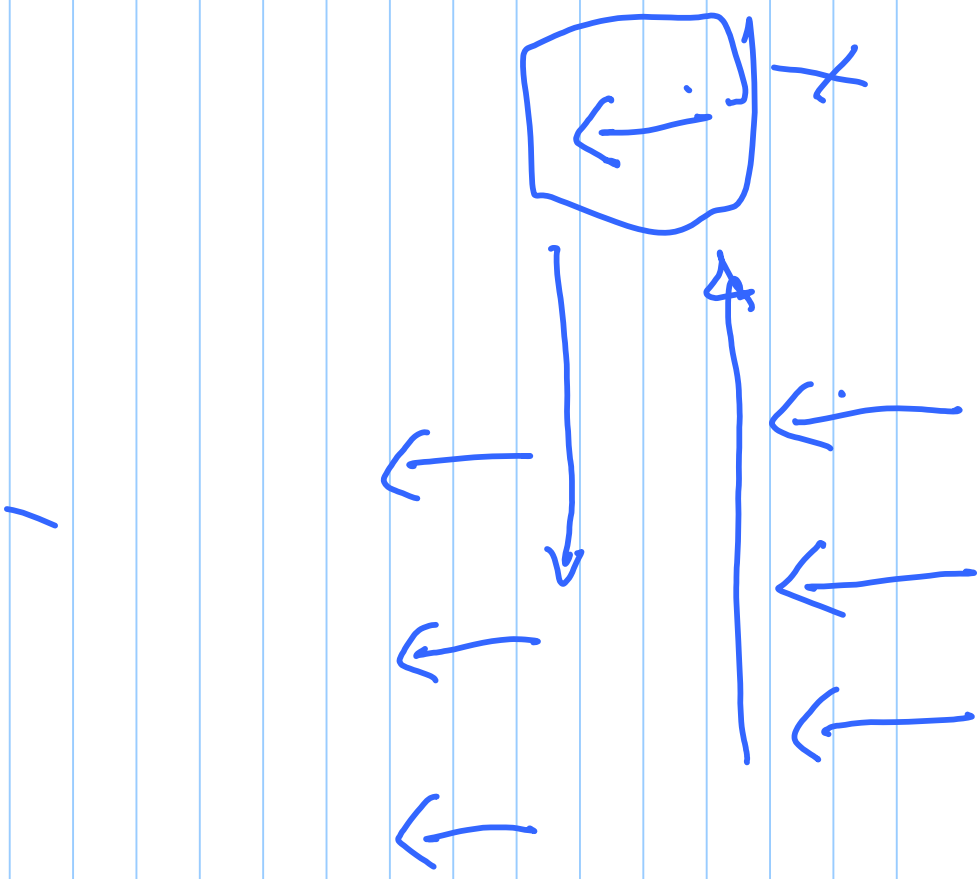
parallel prog




create
A
B
C = #
barrier
join 2
C
D
E = #

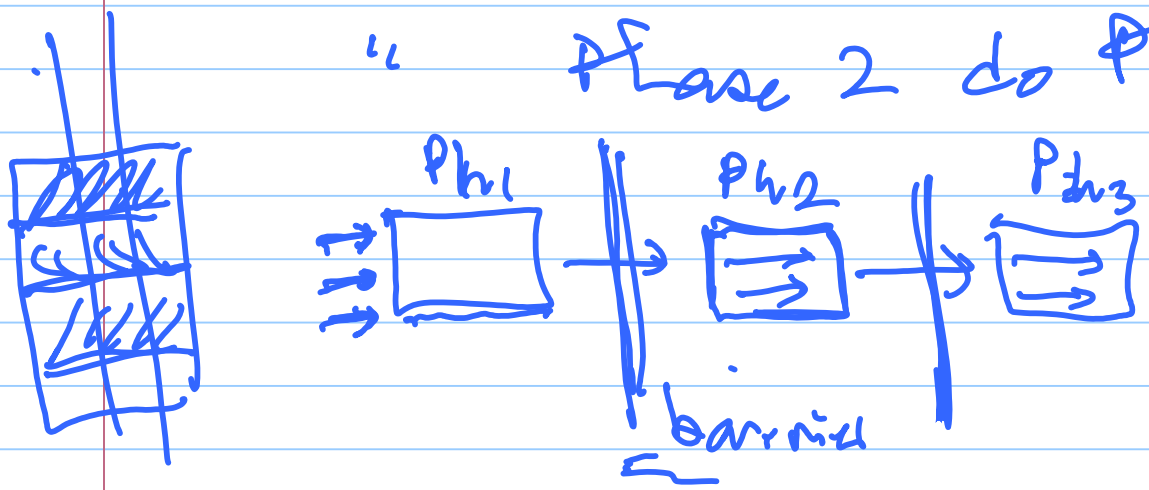

T1
T2
T3
A
B
C
phase 1
barrier
barrier
barrier
T1 C
D
E
2
3
4

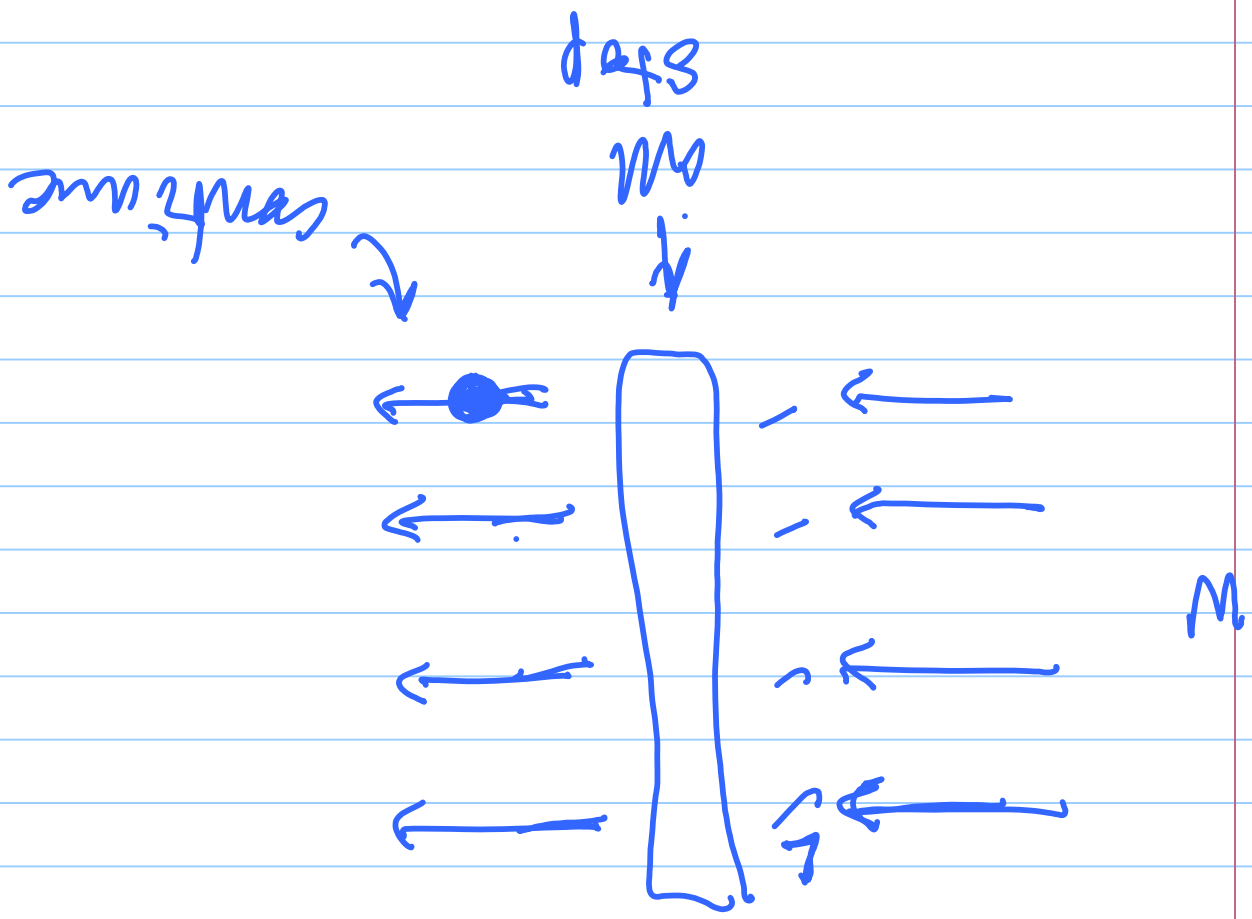

ptr to global ← N
barrier(count)
{
P(lock)
count --
if(count > 0) { P(sem) }
else
V(lock)
V(sem) ← (N - 1 times)
V(lock)
}
CS
Set to N
Sem = 0

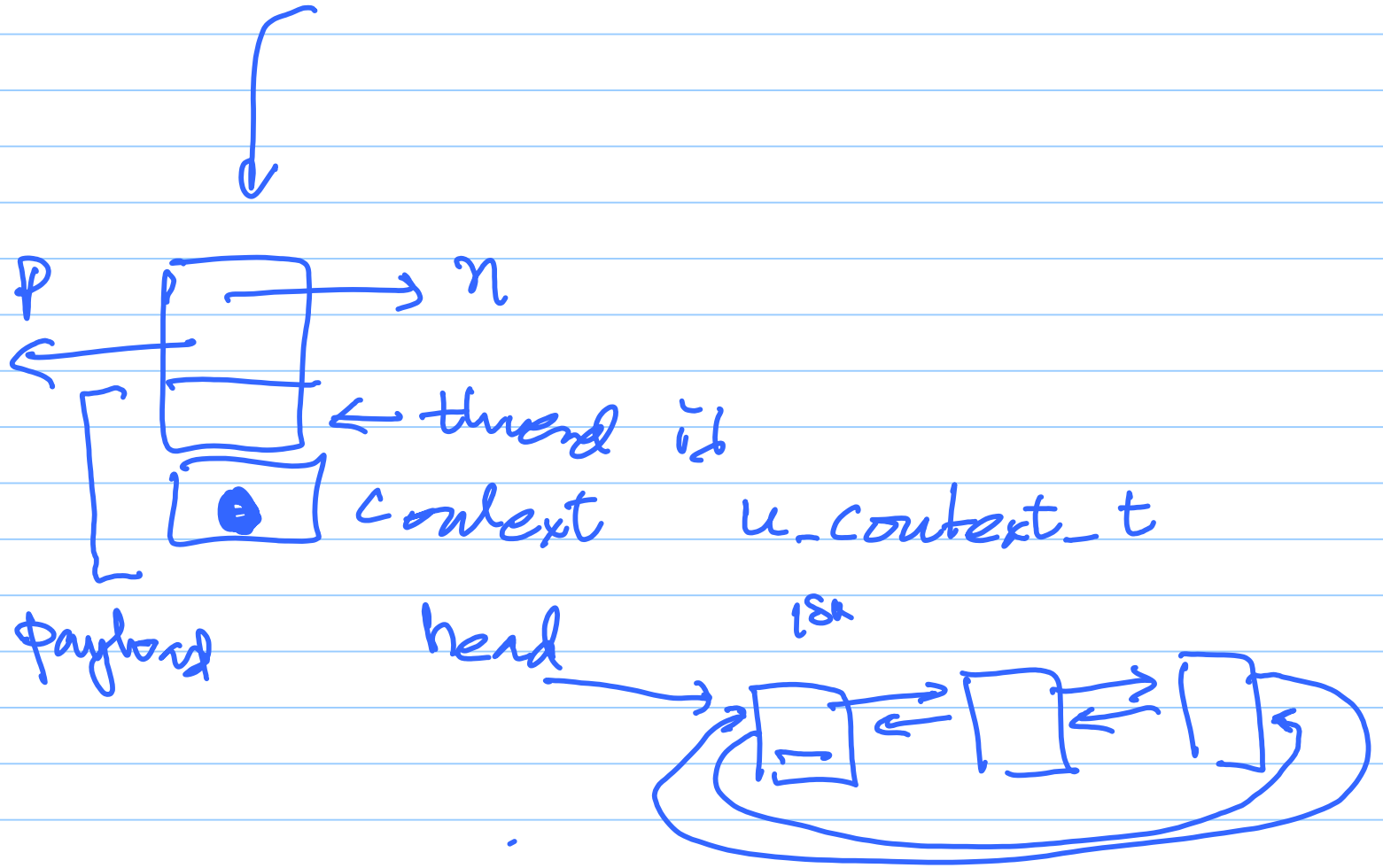


Barrier

- N threads
  all Do phase 1
  after Phase 1 do Phase 2
  " Phase 2 do Phase 3






p
n
thread id
context
u_context_t
payload
head
isa

TCB-t