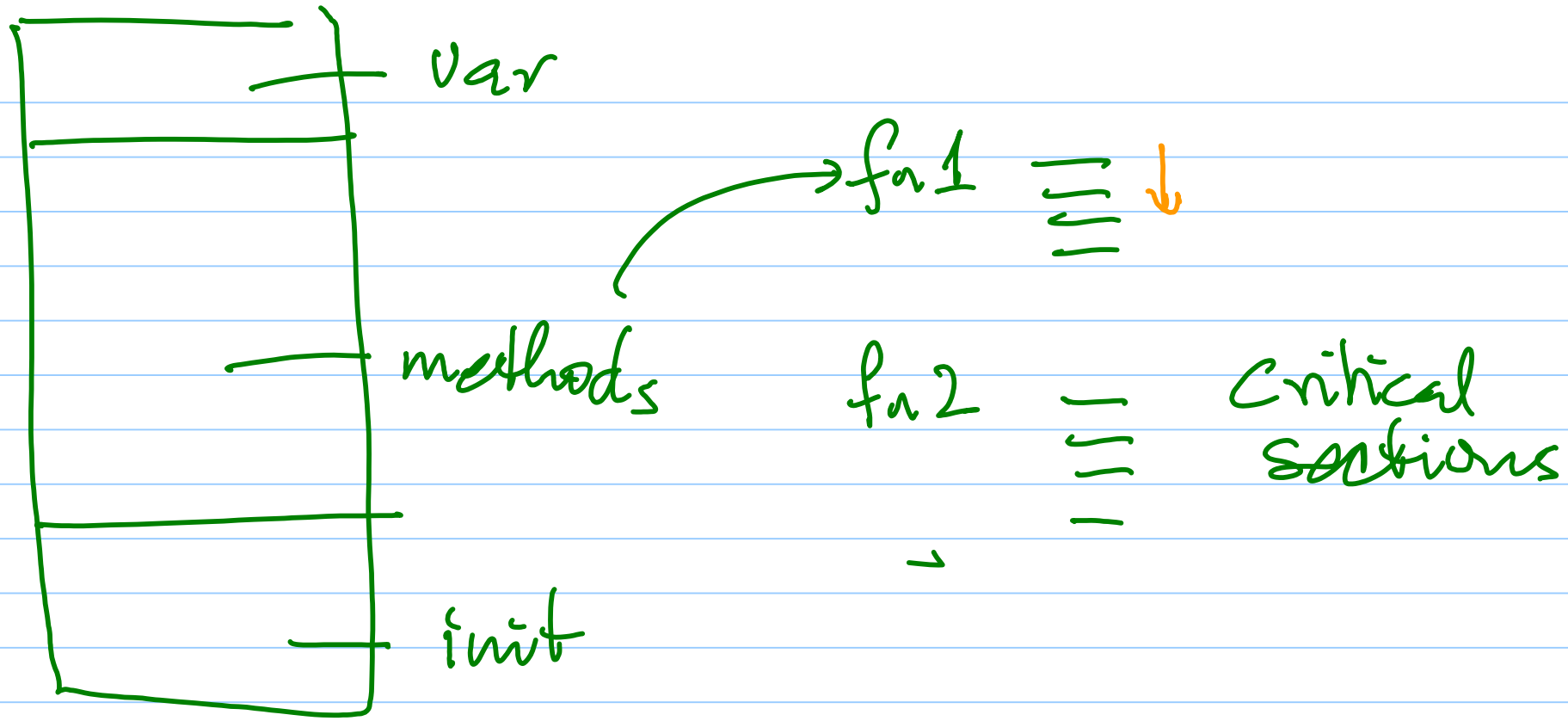
Monitors

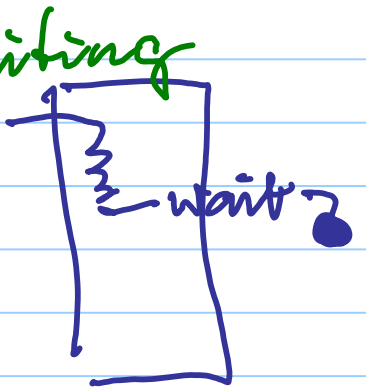
- monitor variables
- " methods
- " initialization
-
-


var
methods
init
fn1
fn2
critical
sections

unconditional block

wait (c) → block, outside the monitor (on condition c)

→ (unblocks one waiting thread if any)



signal (c)

↳ wake up 1 thread that T_2 if blocked on c (if any)

T_1

wait(c) → blocks on c

↳ unblock who?

↓

someone waiting to enter

signal(c)

↓ @ wait(c)

T_1 will wake up T_2 if any

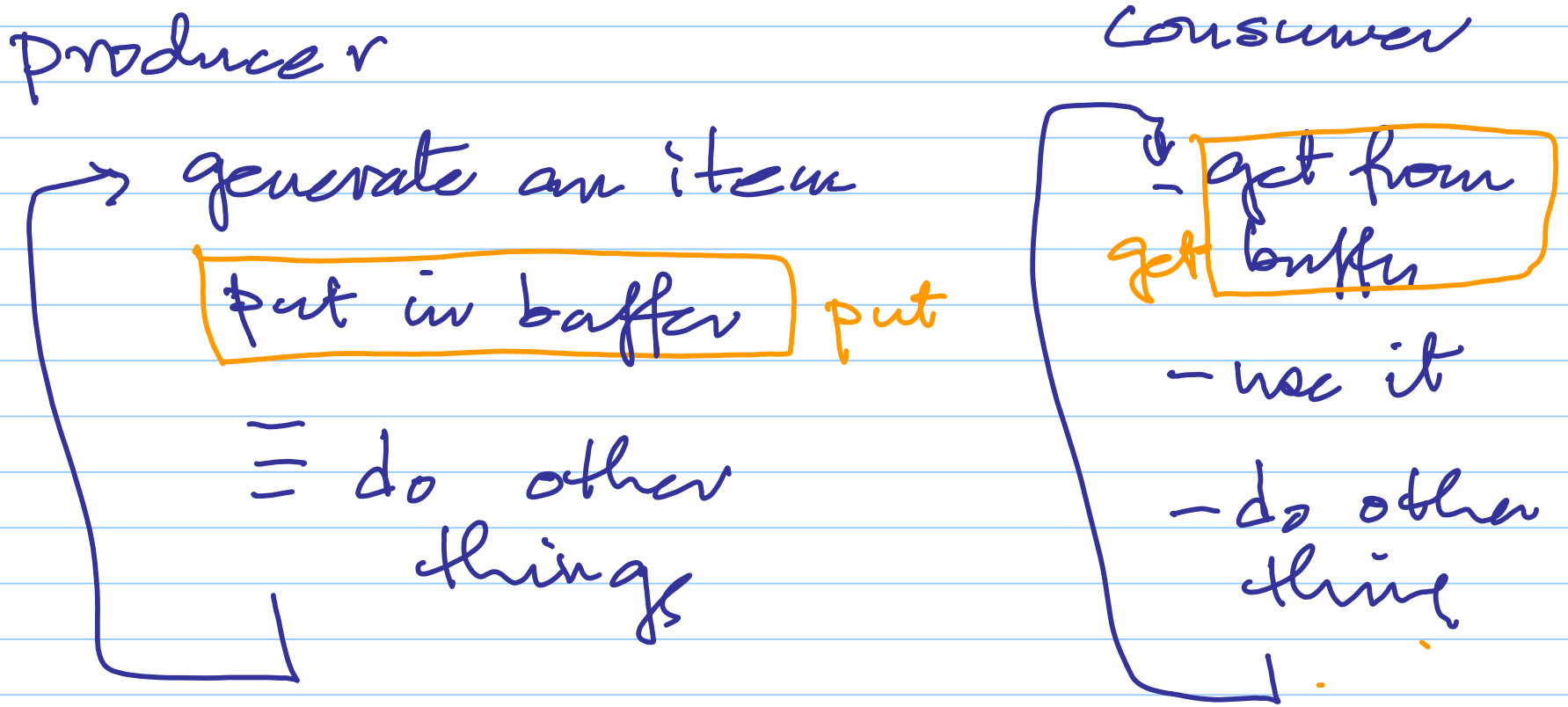
priority

better → ① T_1 goes to sleep, T_2 runs & then T_1 resumes when T_2 leaves or waits

② T_1 continues to execute, & T_2 ??

pthreads starts after T_1 leaves or waits

producer/consumer with monitors



PC monitor count

buffer, in, out → int

prod, cons → condition

put(item)

{ if (count == N) wait(prod)

count++

buff[in] ← item

: inc in

signal(cons)

}

get()

{

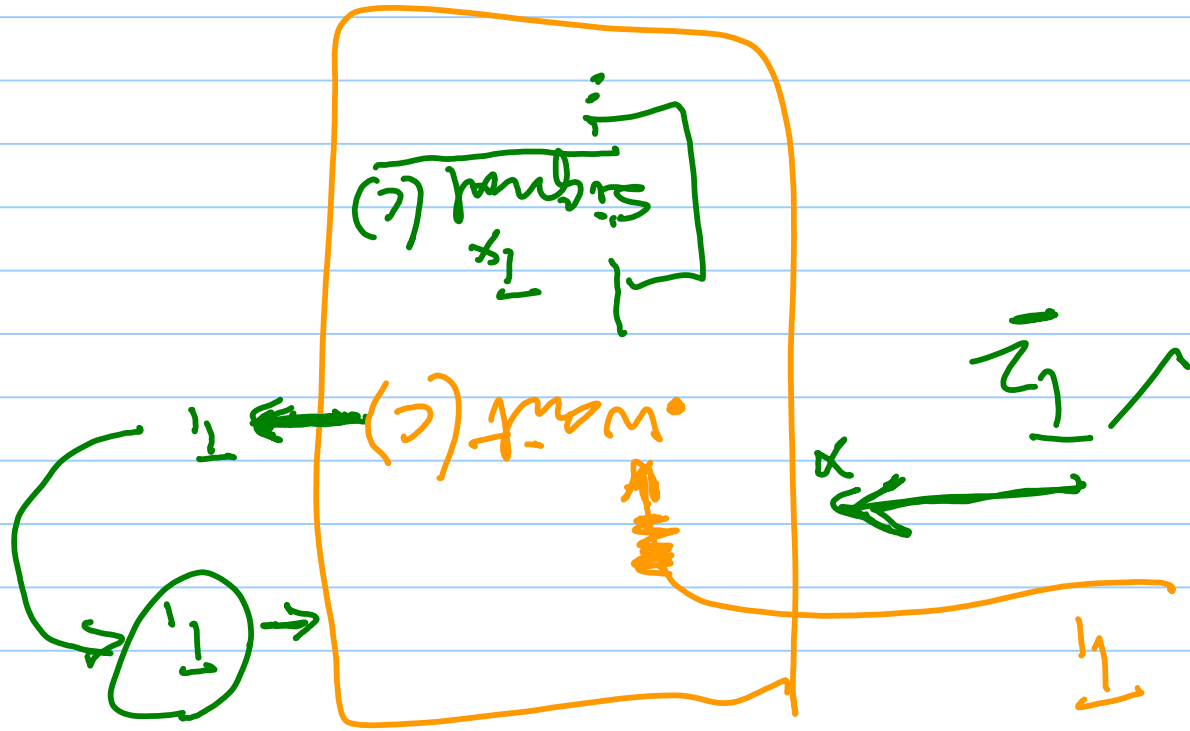
if (count == 0)

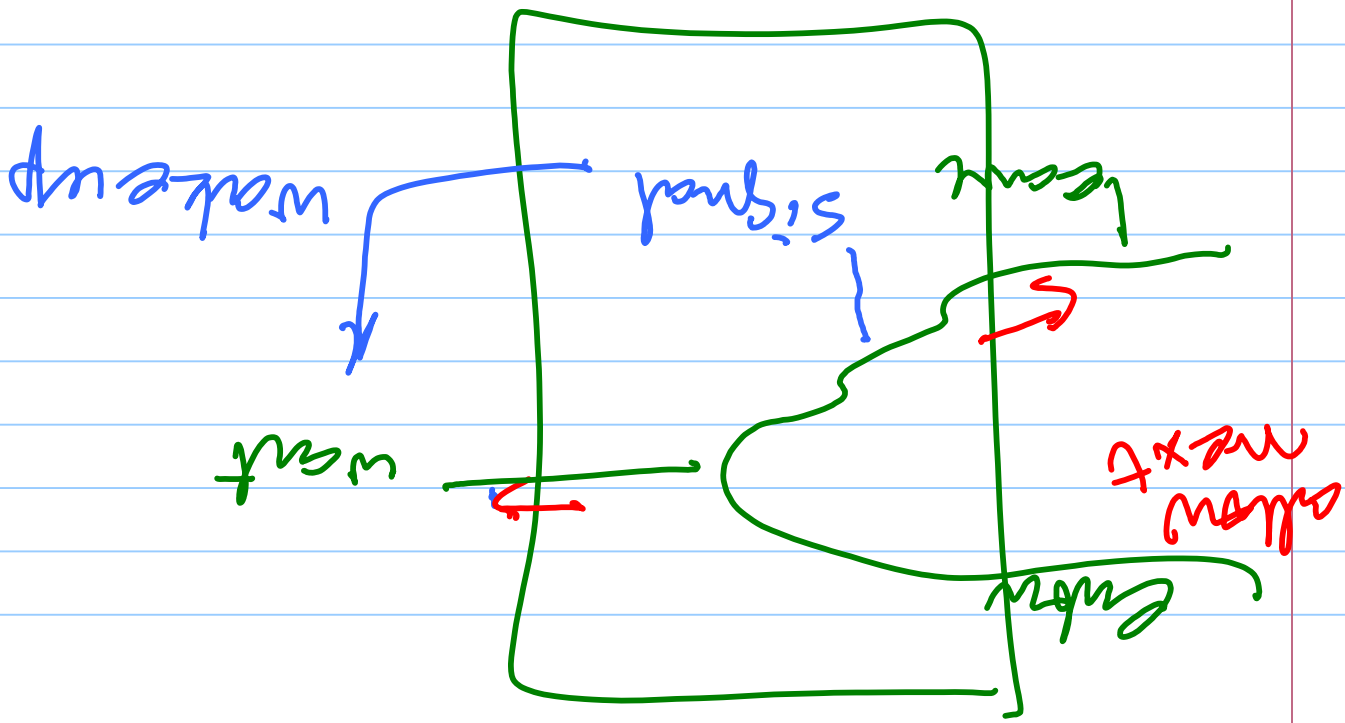
wait(cons)

count--

signal(prod)

}





Can monitors implement semaphores?

→ A monitor S for every semaphore

Var → S_count, S_cond

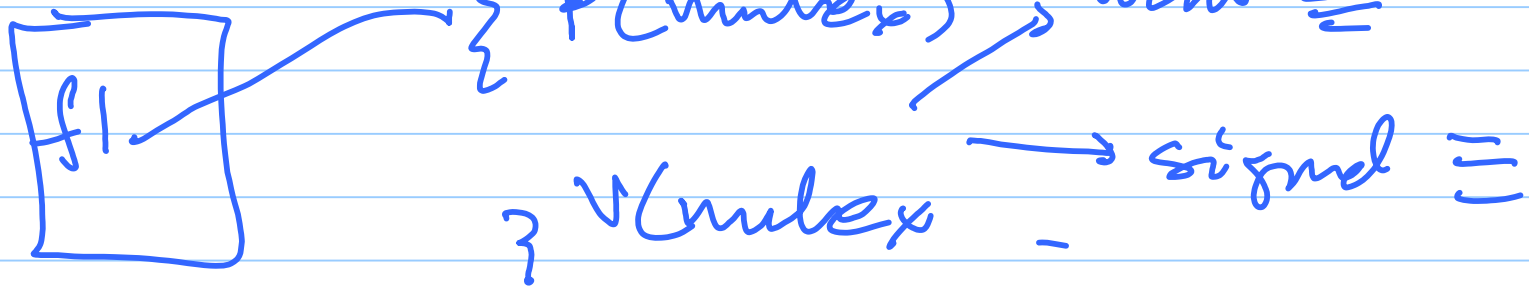
init to sem value

P(S) → [S_count - -
if (S.count < 0) wait(S_cond)]

V(S) → [S.count + +
if (S.count <= 0) signal(S_cond)]

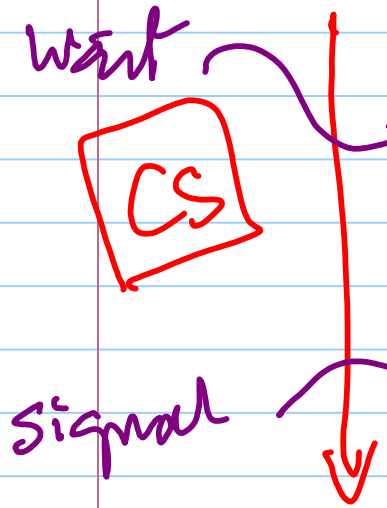
Implement monitors with Semaphores

↳ possible but complex



variables
pthread_mutex_t lock
pthread_cond_t cond

```
f1(arg)
{
    pthread_mutex_lock(lock)
    pthread_cond_wait(cond, lock)
    pthread_mutex_unlock(lock)
}
```

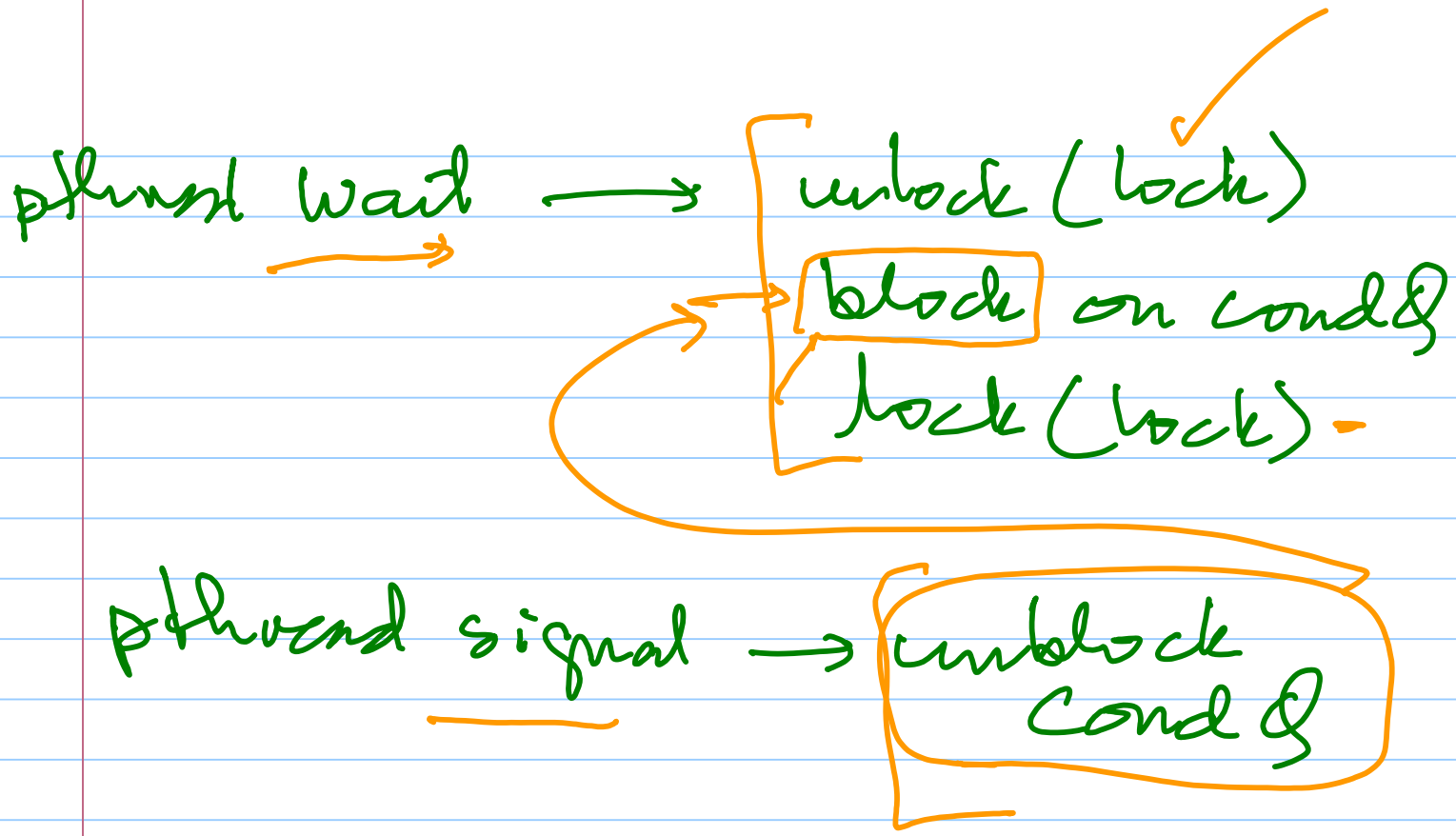


?

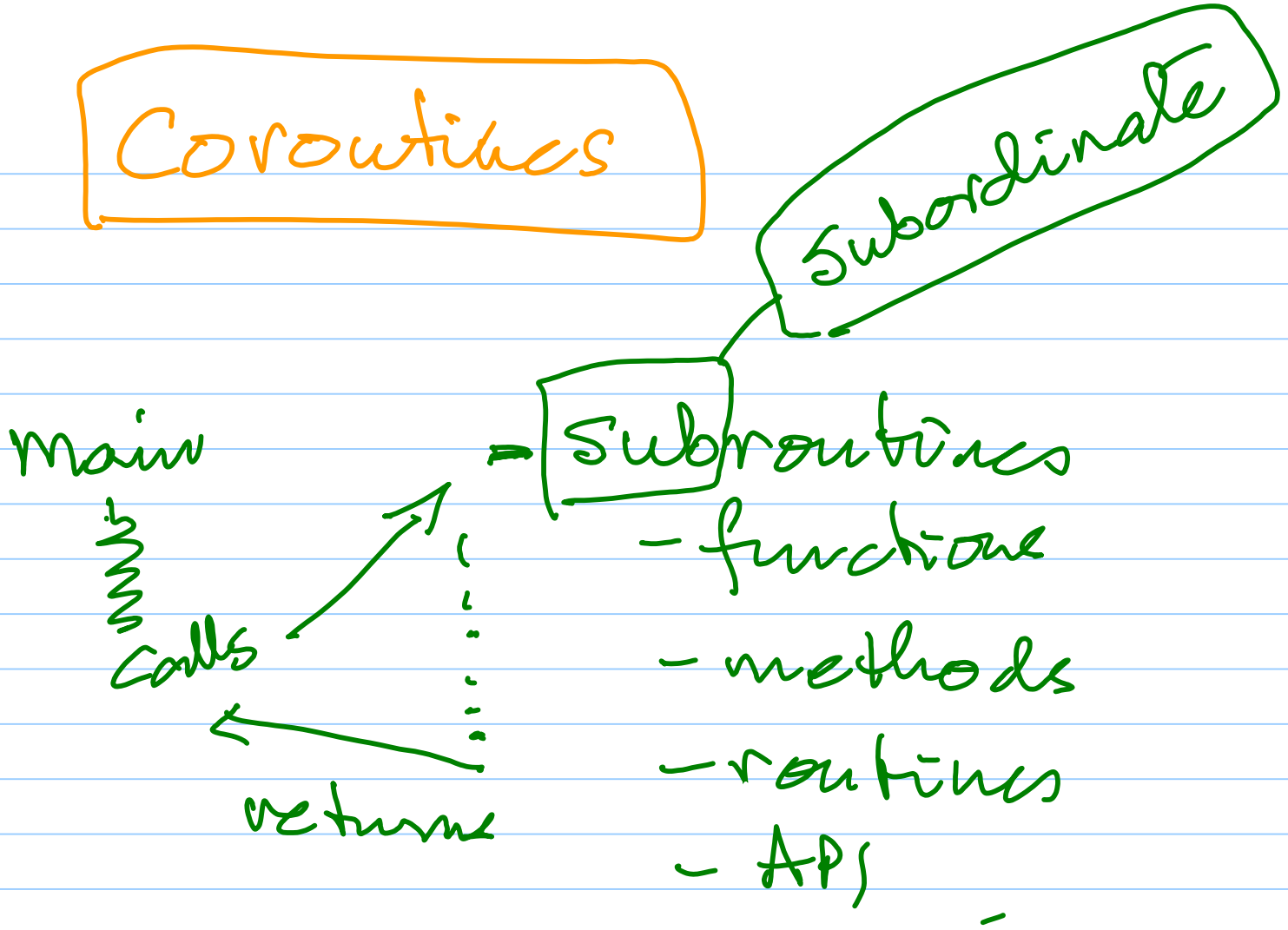
unlock(lock)
block
lock(lock)

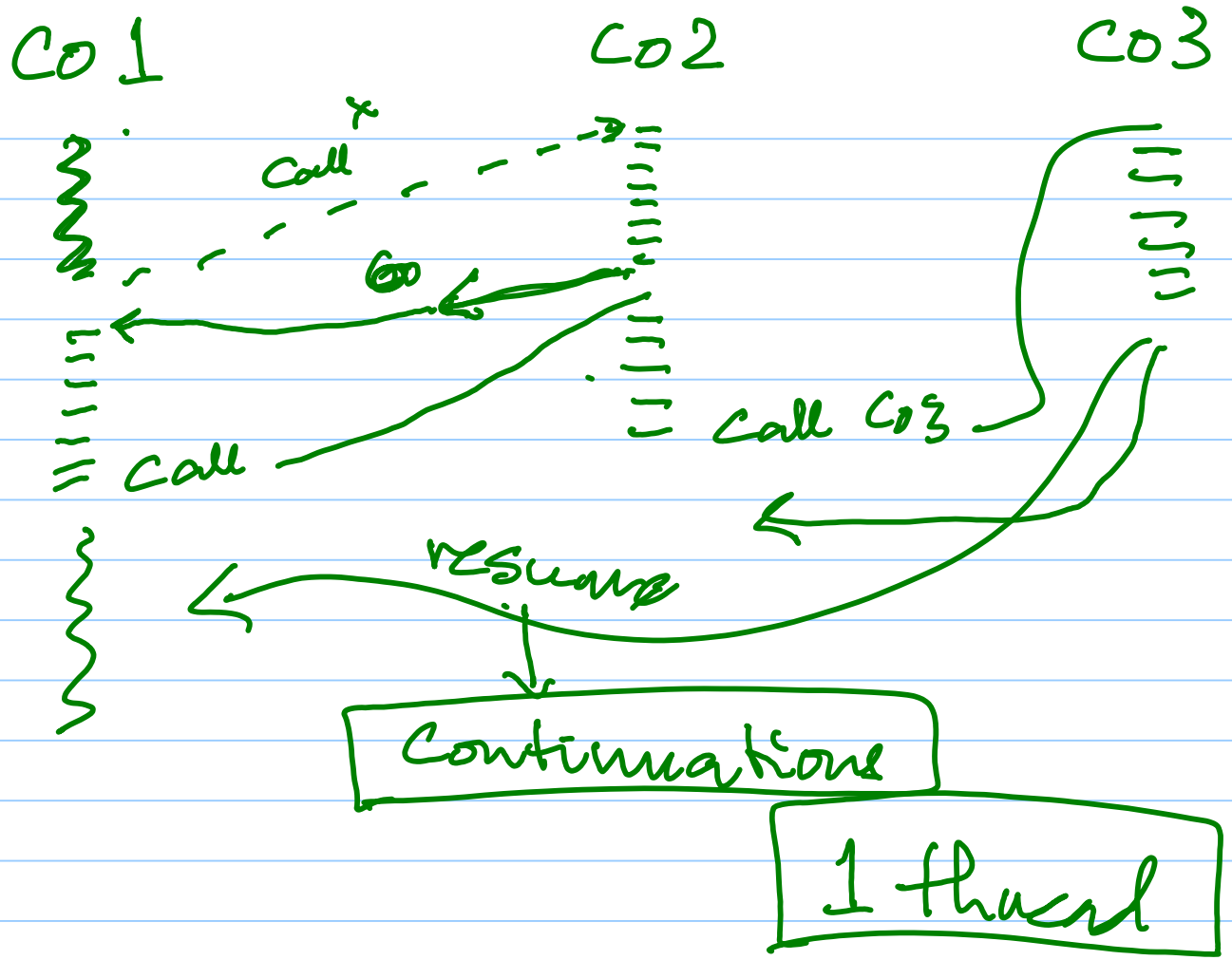
pthread_cond_signal(cond)

NO LOCK


pthread_wait
unlock (lock)
block on cond
lock (lock)
pthread_signal
unblock
cond

Coroutines


Subordinate
Main
calls
returns
Subroutines
- functions
- methods
- routines
- APIs


Co 1
Co2
Co3
call
call Co3
resume
Continuations
1 thread