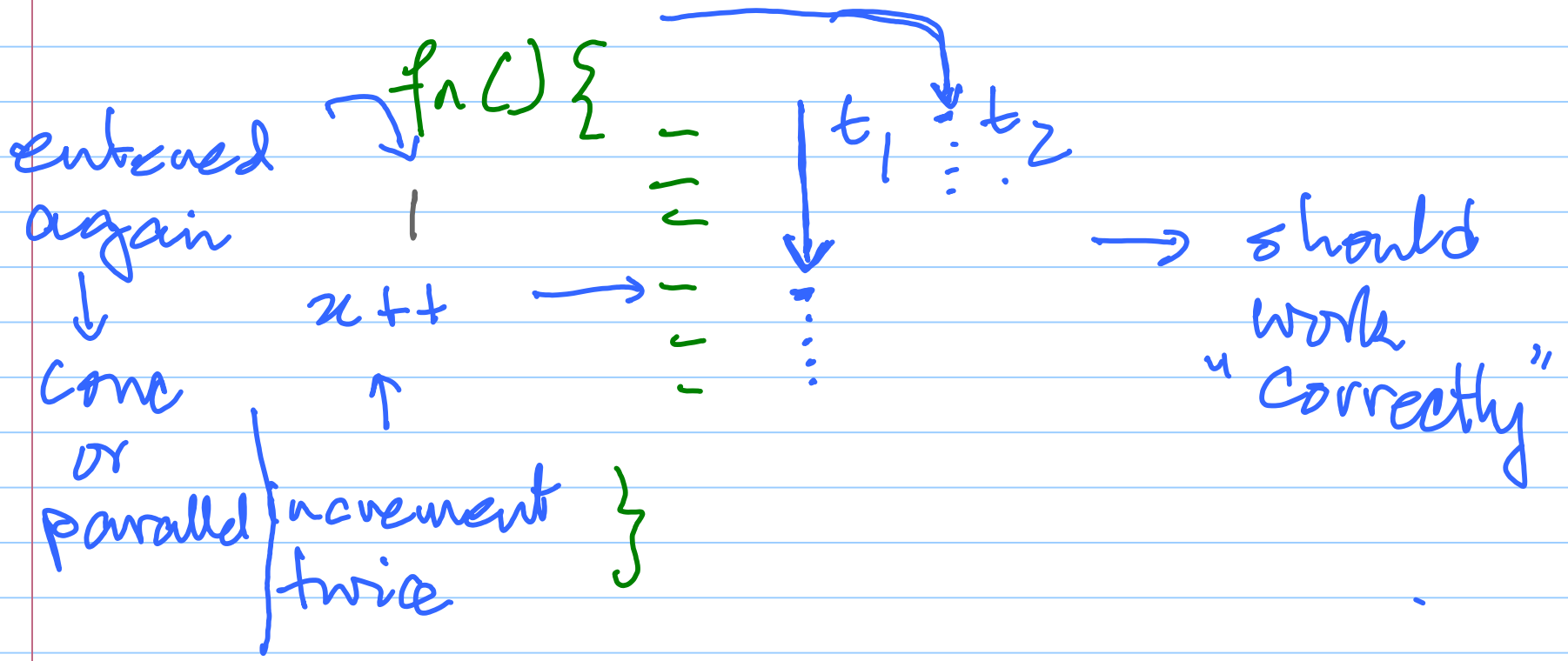
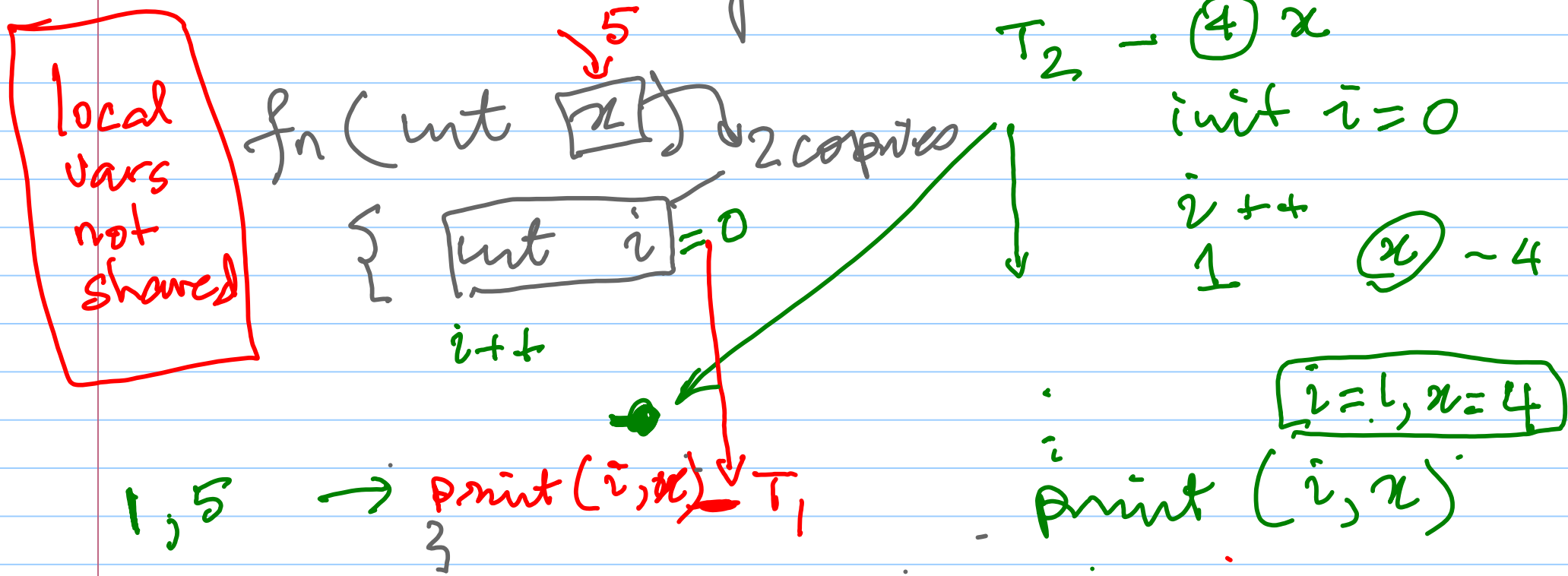
Reentrant code



- All kernel routines are reentrant

↓ code reentrant?

(1) use local variables only



Shared by threads

- global
- heap

not shared → stack (local)

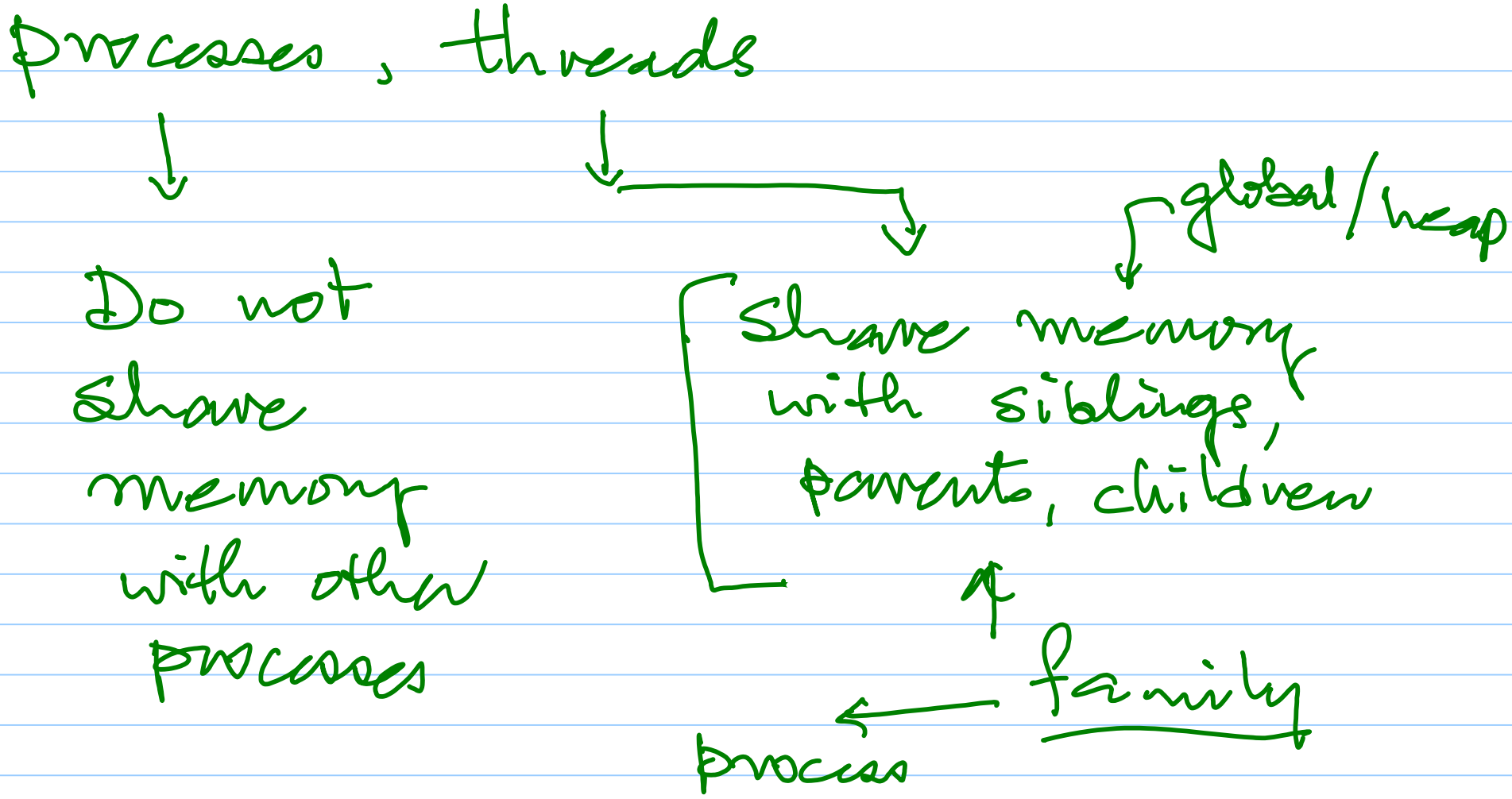
local variables only

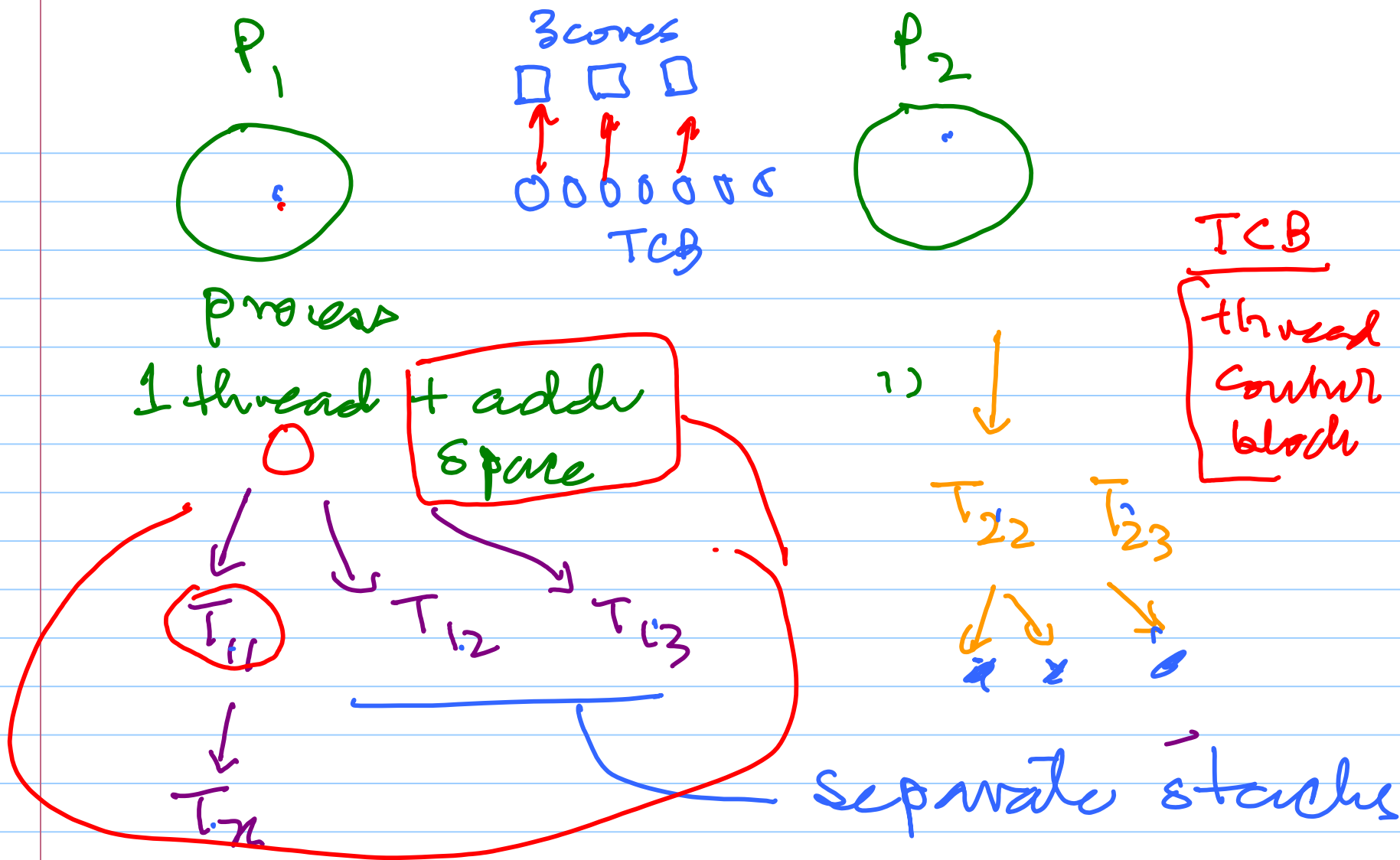
- not possible

lots of global

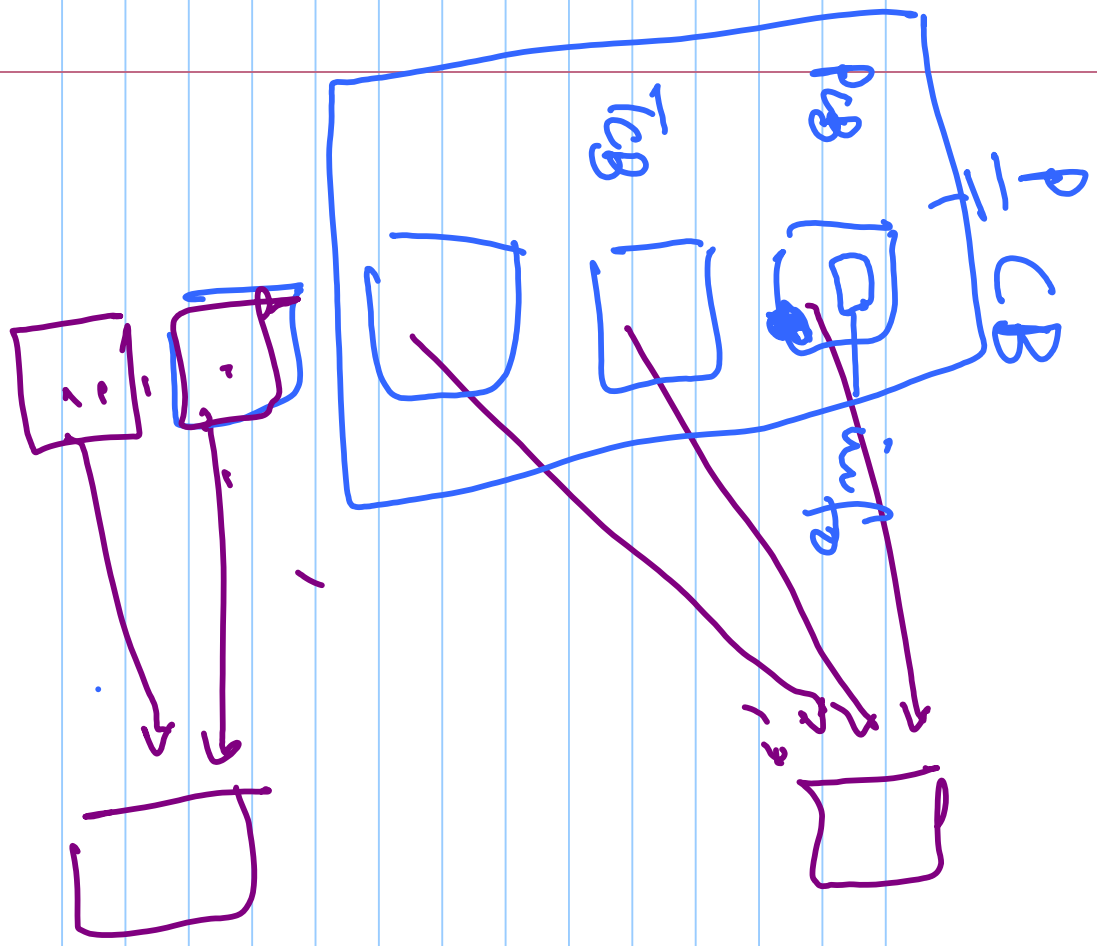
↳ shared

② → use locking → reentrant


processes , threads
Do not share memory with other processes
global/heap
share memory with siblings, parents, children
family
process


P1
3 cores
P2
TCB
process
1 thread
+ addr space
TCB
thread control block
T11
T12
T13
Tn
T22
T23
separate stacks

- p. ∃T2 ⇒ stack contain-

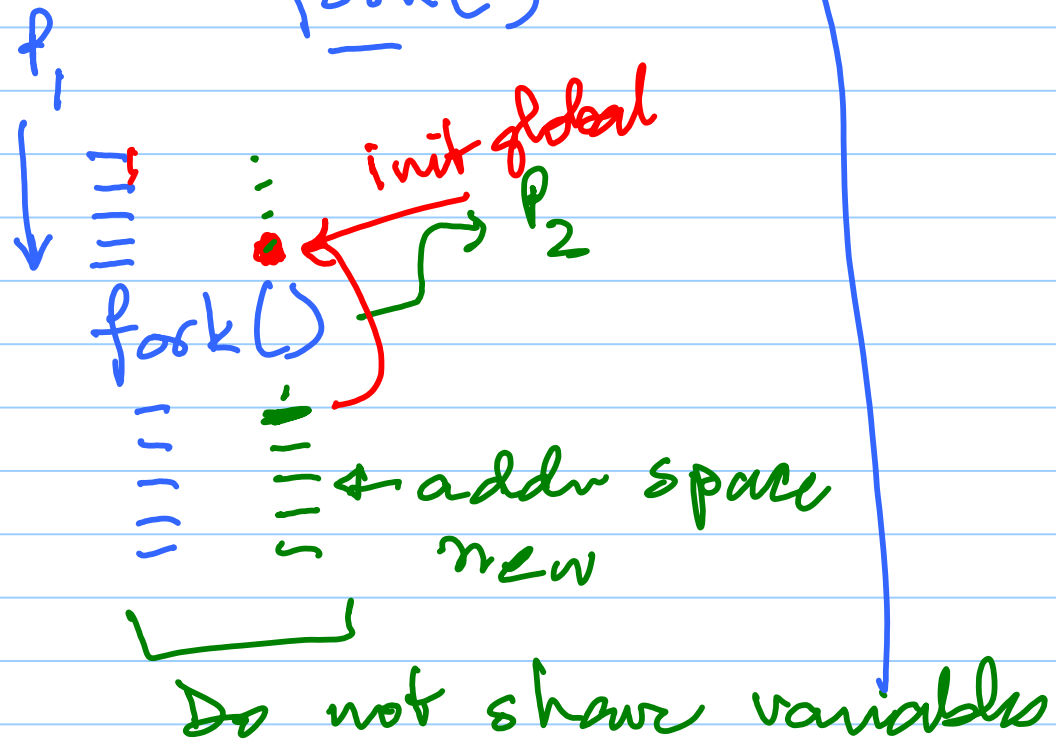

PSB
CB

Starting processes & threads

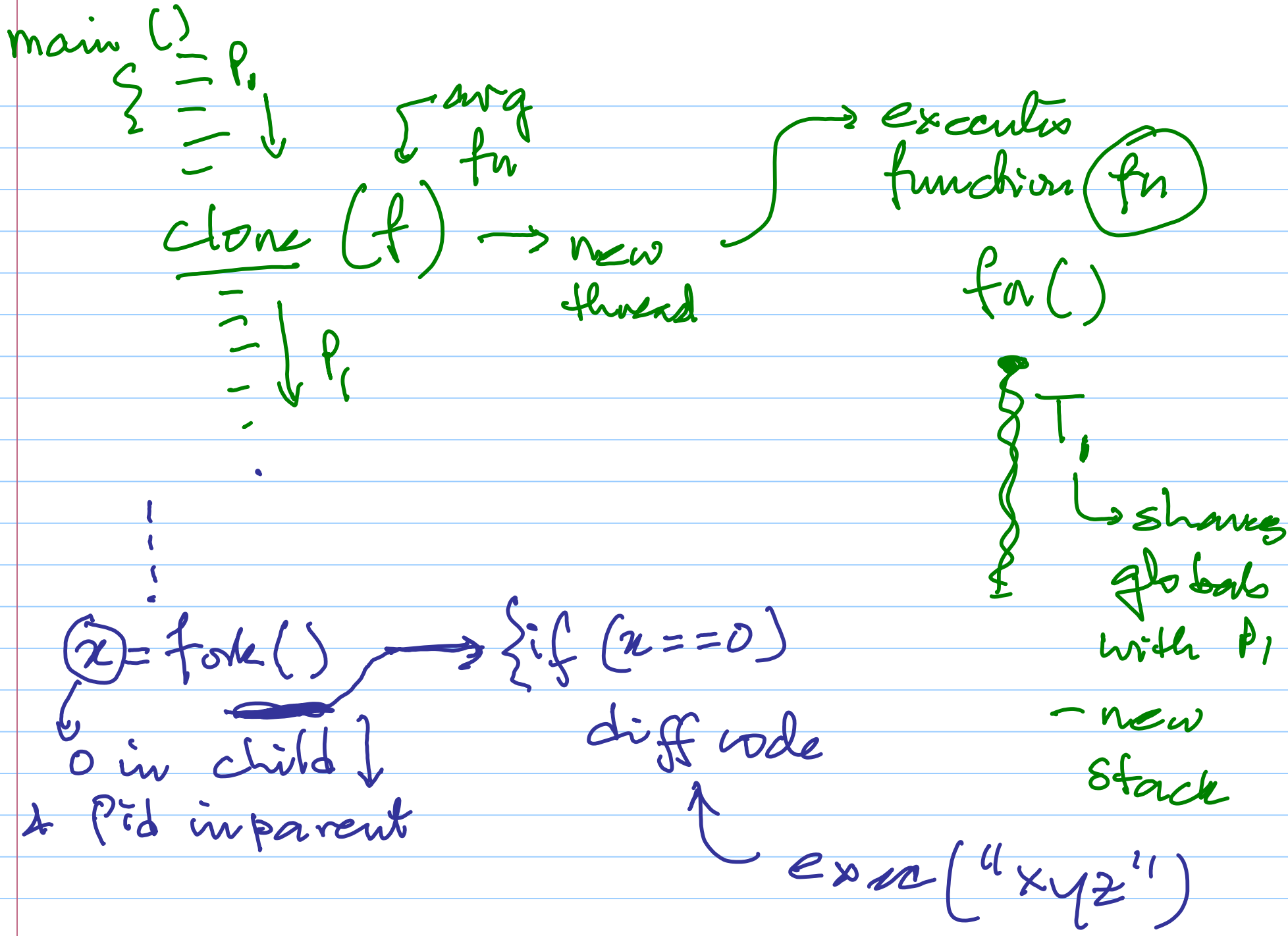
Linux

fork()

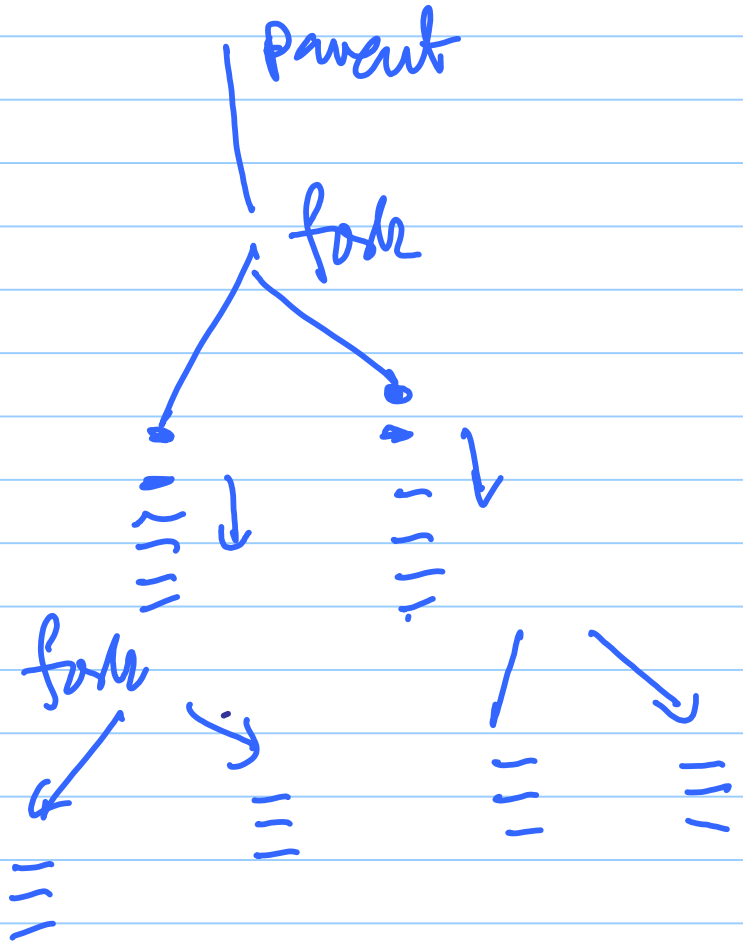
x clone()

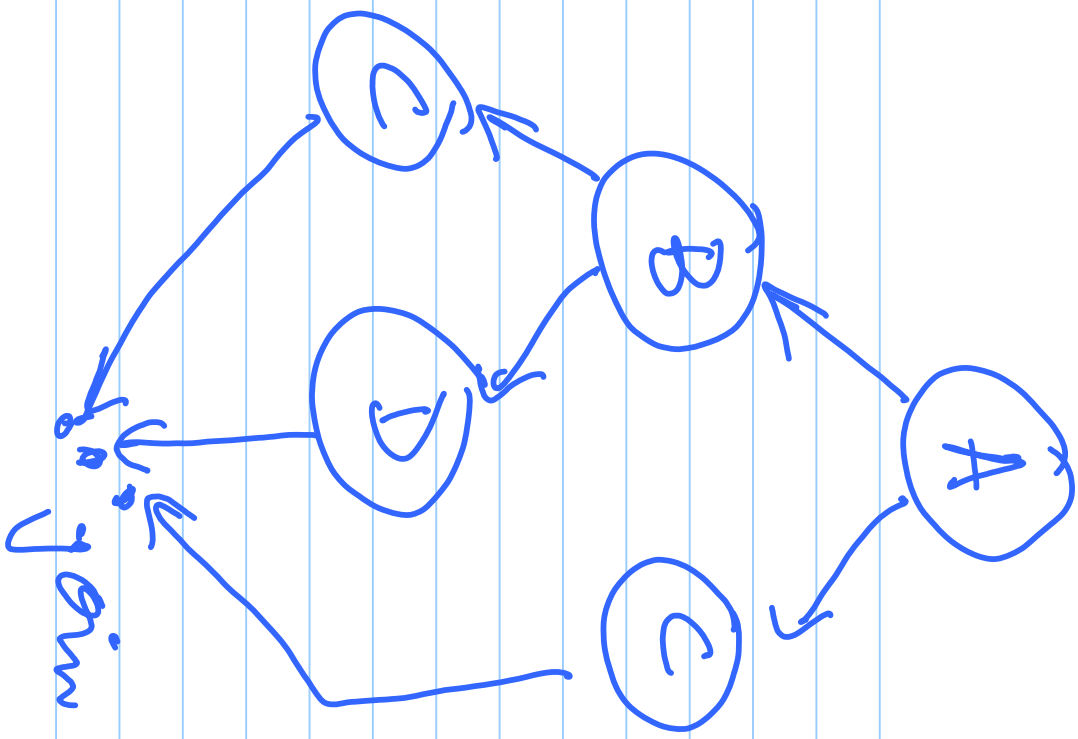


P1 process
fork
P2
P3


main ()
P0
clone (f)
arg fn
new thread
executes function fn
fn()
T1
shares global with P1
- new stack
x = fork()
0 in child
& Pid in parent
{if (x==0)
diff code
exec("xyz")

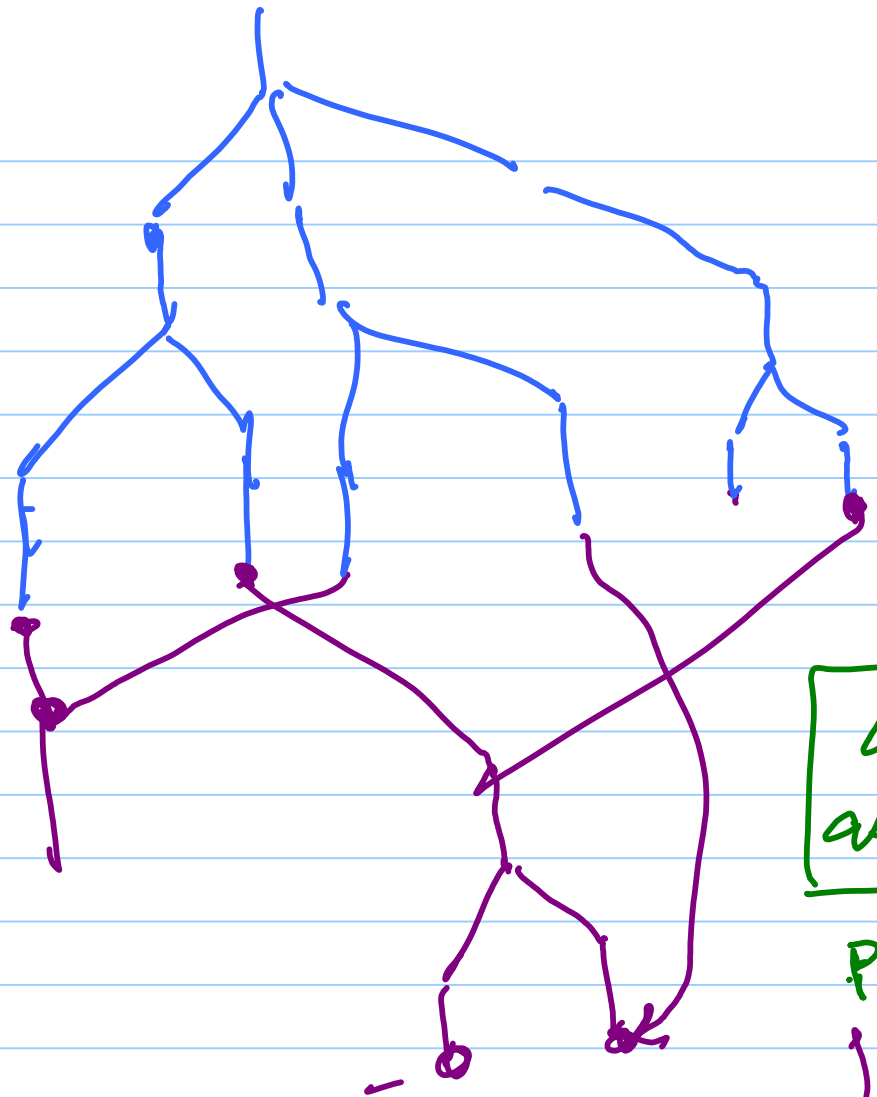
theory fork-join model
(for threads & processes)





join
Any 2 threads
↓

fork-join



Q, R after P

P

Q R

A B

C

precedence graph

A x B will execute b4 C

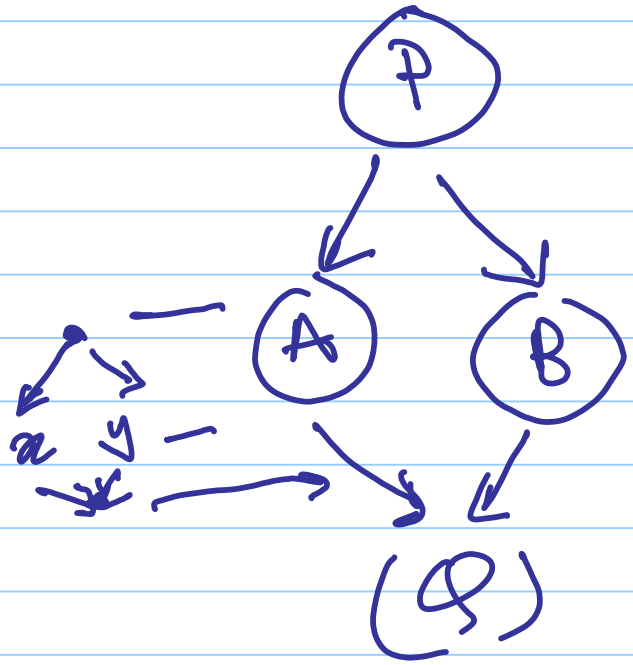
mc ss

par model

$\equiv P$

$\equiv$ par (A , B)

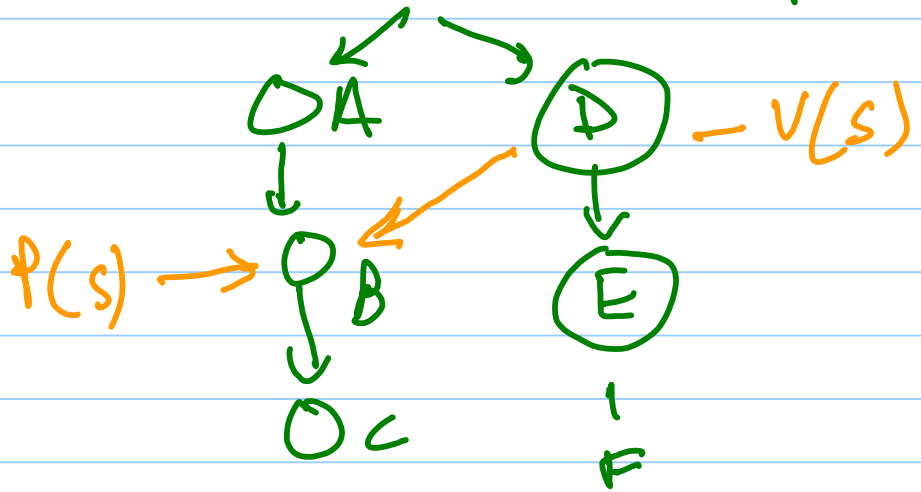
$\equiv Q$

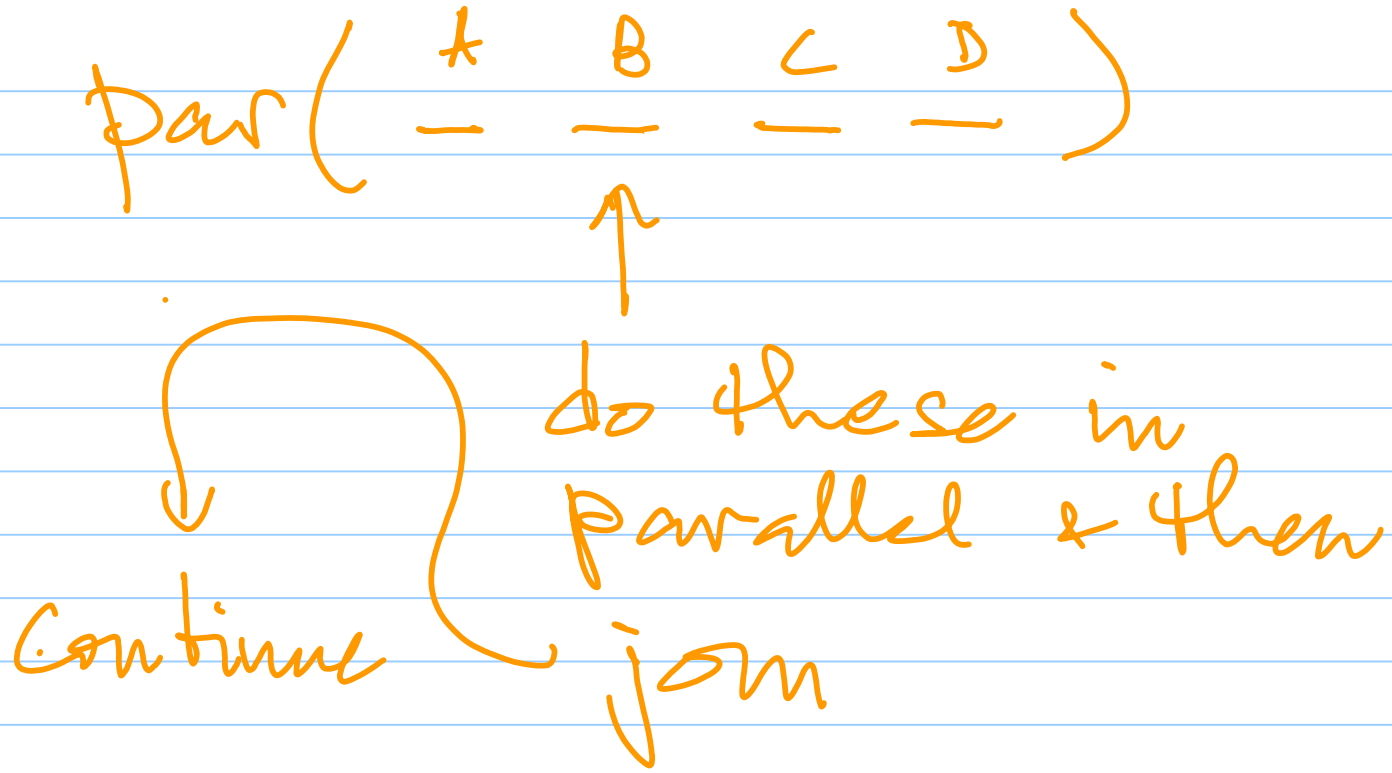


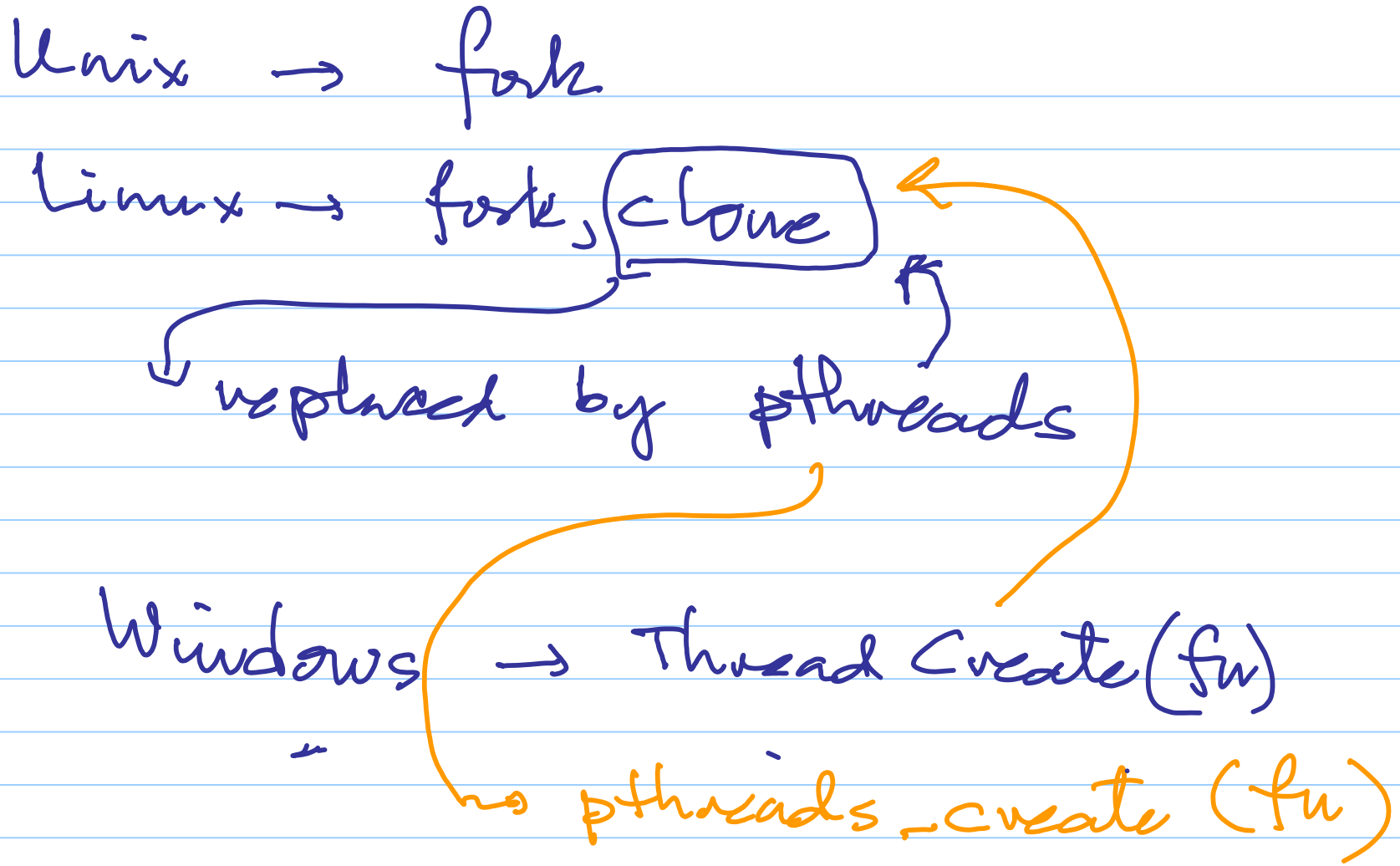
A → par (x , y) → series parallel graph

threads + semaphores

⇒ equiv to a fork join program




par(A B C D)
do these in parallel & then join
continue


Unix → fork
Linux → fork, clone
replaced by pthreads
Windows → ThreadCreate(fn)
pthreads_create(fn)

if (Y==0)
exit
Atomic
same

pthread_create

pthread_join X

pthread_wait

pthread_exit

pthread_mutex_t

→ pthread_mutex_lock (m)

" " unlock (m)

---

like a semaphore

pthread_mutex_init (m)

↑ inits to 1

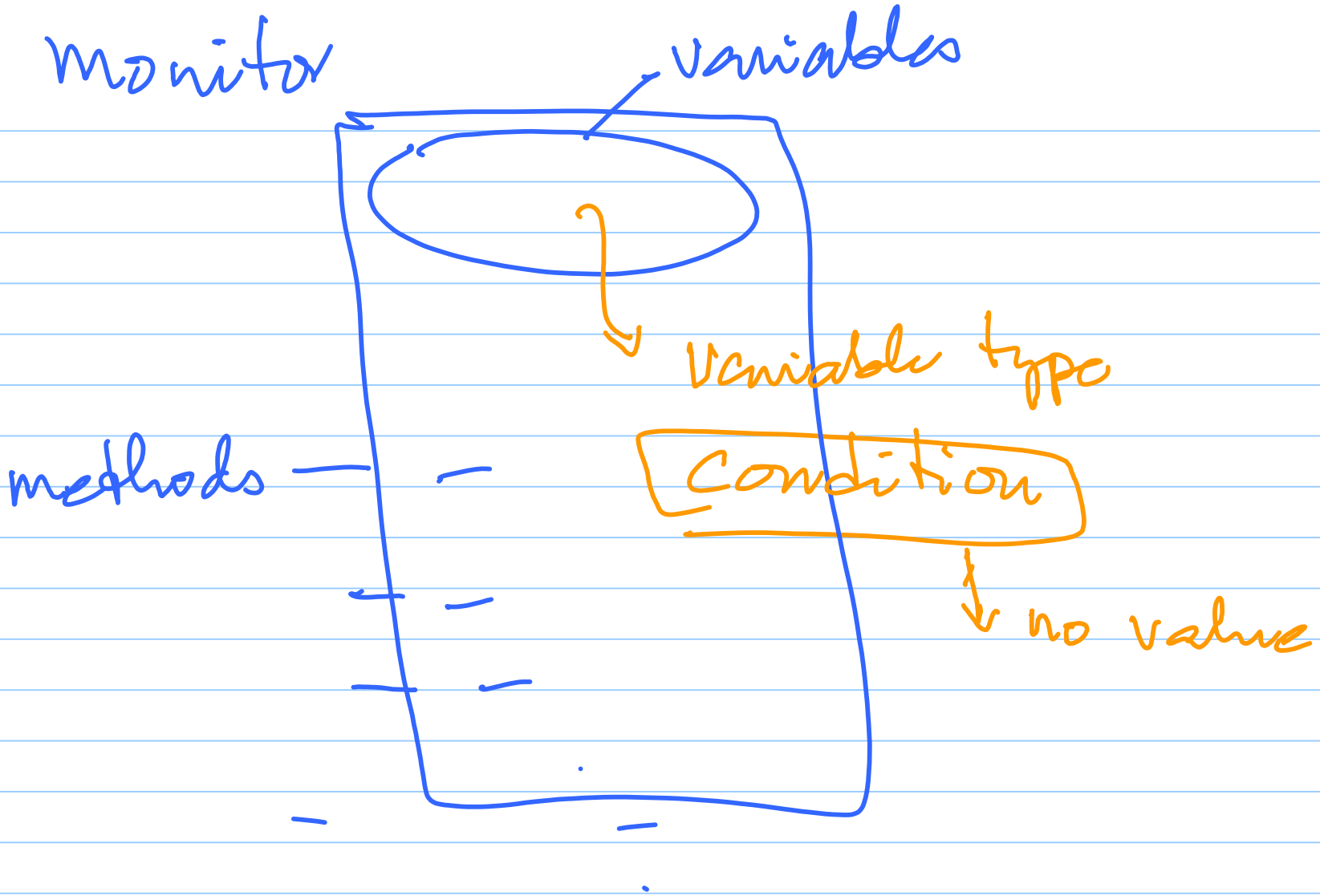
pthreads + others

└ Monitors

↑ ? → do not monitor anything!

↓

ADT (class)


Monitor
variables
Variable type
methods
condition
no value

Monitor →

(1) — condition variables

(2) → two defined functions

(1) WAIT

(2) SIGNAL

(3) All monitor methods are mutually exclusive → NO concurrency

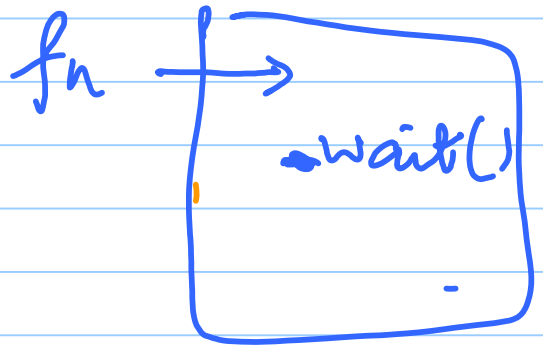
W
measurement
control panel
W. ft 1
W. ft 2
C1 C2 C3
cond
No
race
conditions

wait (c) ← arg → condition

↑

unconditional wait

⇒ thread will get blocked → But will exit the monitor



(T) Signal (cond)

(A) if no one waiting
→ nothing happens

(B) if Tx is waiting then
Tx is unblocked (!!)

fun()

Ty

Signal (c)

Tx

wait

Tm is blocked.