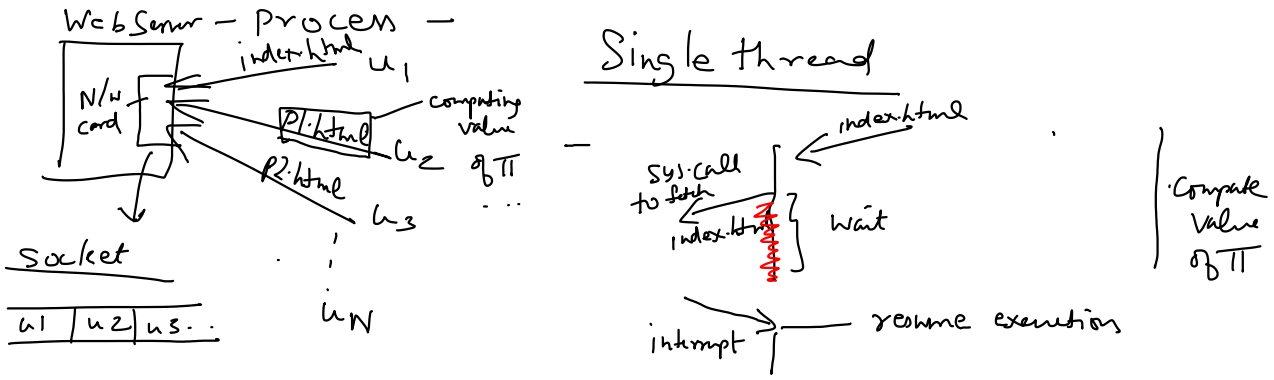
Concurrency

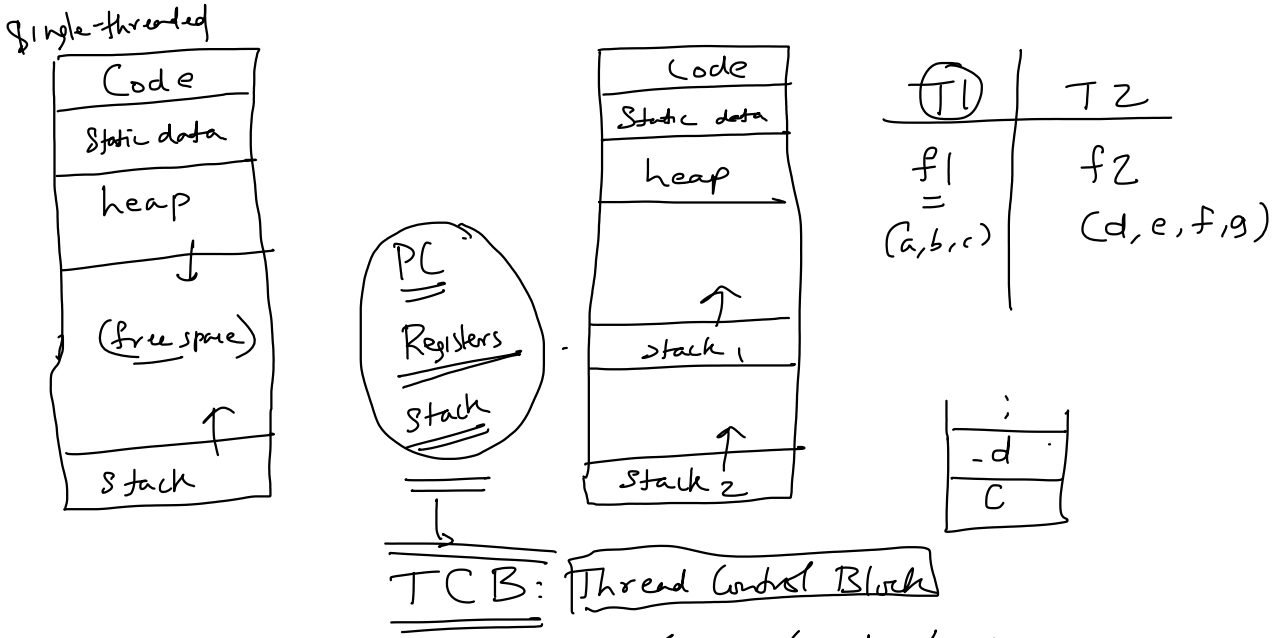
Process: Addr. space with 1 or more threads

Why do we need threads?



→ Overlap I/O operations with Compute optn —

→ Multiple processors – 1 thread cannot exploit the parallelism



TCB: Thread Control Block

/ proc / pid / tids

PCB_1 – PCB_2

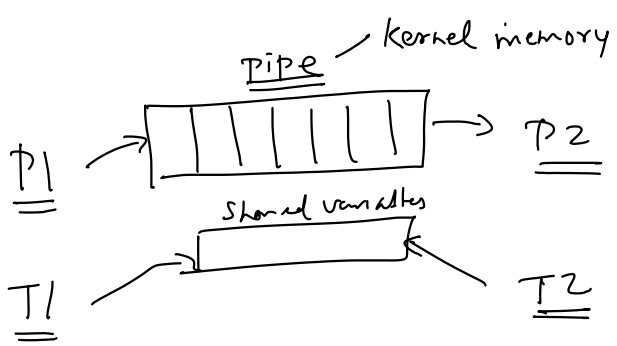
↳ TCB_1 ↳ TCB_2

T1 shares some data with T2 – Sharing with threads is easy // give rise to sync. issues.

P1 share some data with P2 –

Inter-process comm^n

pipes




```
i = 10;
fork(); {
```

P (i = 100) C (i = 200)

Issues that arise because of sharing state

thread_fork()

P C

{ non-reproducible, non-deterministic }

↳ hard to debug programs with threads

Q. arg = 200 — o/p — 400

arg = 20000000 — o/p — Non-deterministic?

} Context-switching

Fundamental building block

atomic operations — bal = bal + 1 } atomic opn

without any interruption { load bal, store } CS { bal = bal + 1 }

T1: lock.acquire() … lock.release()

T2: lock.acquire() wait

2 threads cannot hold the same lock at the same time.

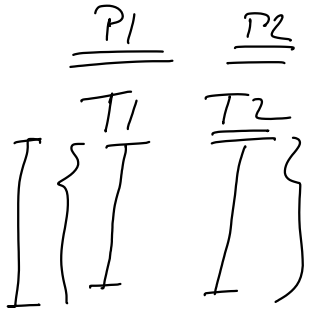
{ Critical section, mutual exclusion }

time: → real — exec. time

user — time spent in user mode

sys — " in kernel mode

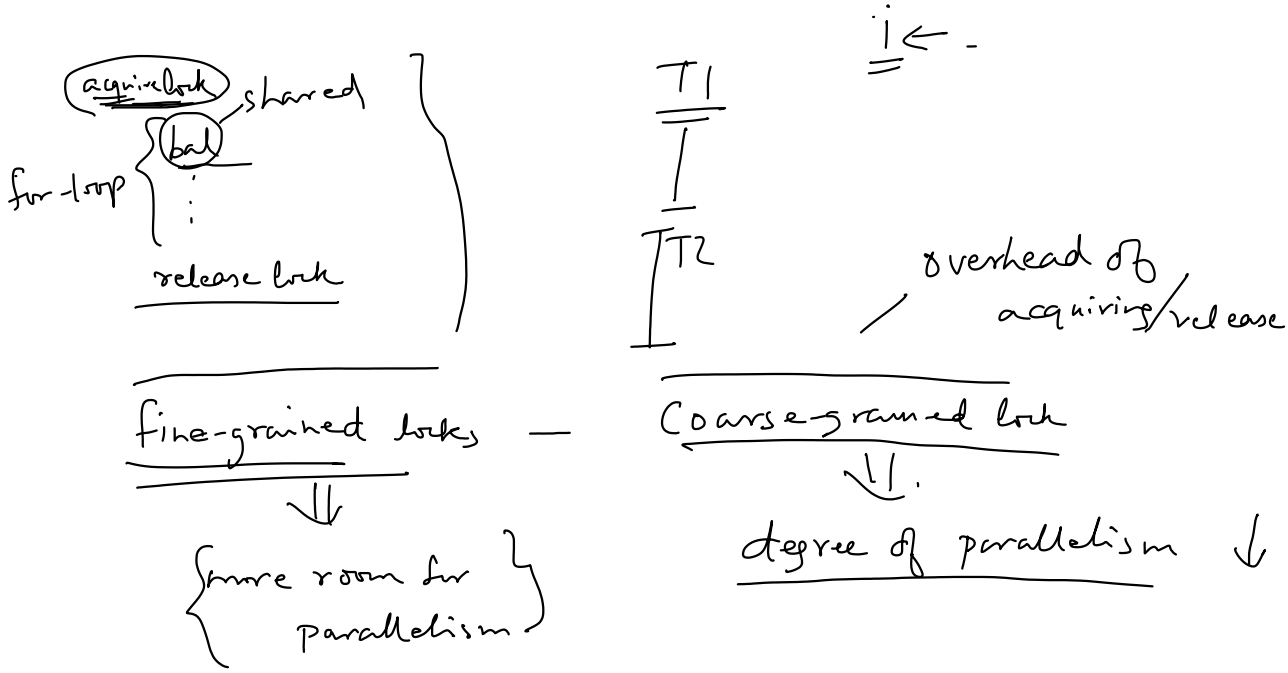
[user < real, sys < real]



Parallelism

real = max(T1, T2)

[user = T1.user + T2.user]


acquire lock
shared
bal
for-loop
release lock
T1
T2
overhead of acquiring/release
fine-grained locks
Coarse-grained lock
more room for parallelism
degree of parallelism