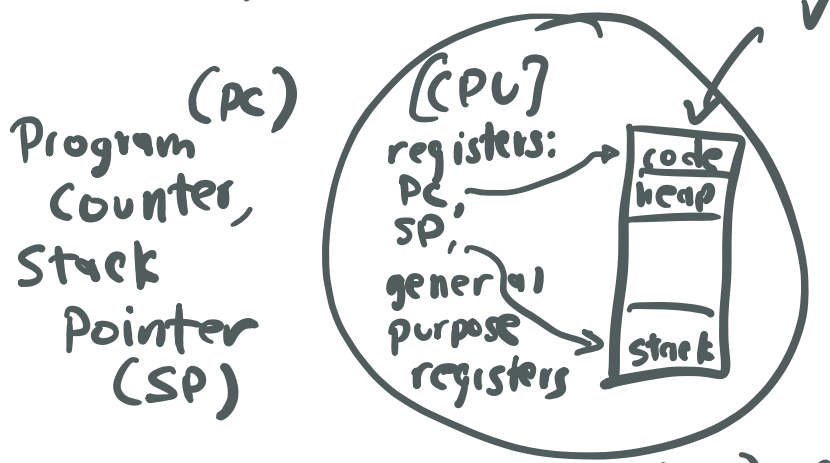
Today: OS, Part 2

What is concurrency?
What are the major issues?
(two: atomicity and ordering)

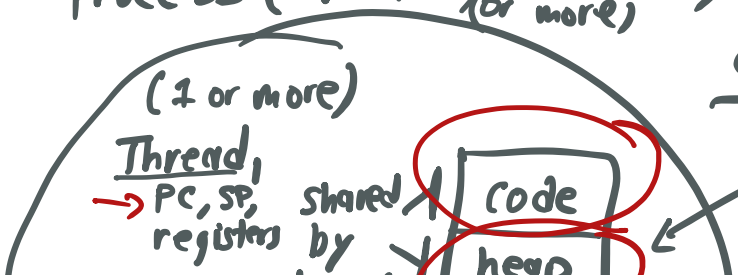
Part 1: Virtualization
Part 2: => (Concurrency)
Part 3: Persistence

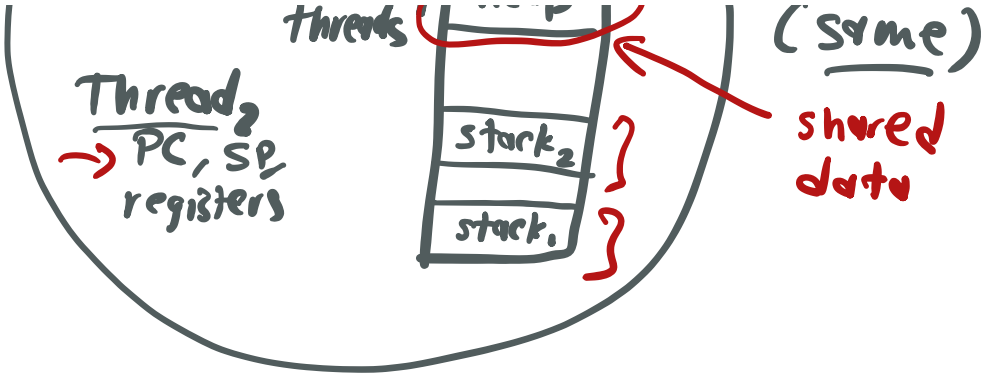
To begin:
abstraction of running program
=> Process: one thing at a time

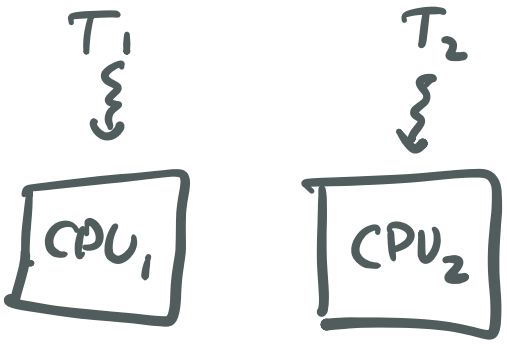


=> Multi-Threaded Process: to do many things at a time

Process (w/ two (or more) threads) (to do many things concurrently)







Why have multi-threaded program?

=> [Parallelism: to use N CPUs (N>1) to work on one problem faster

=> [I/O: waiting for an I/O => thread blocked (but others can run)

Problems: Synchronization

=> Atomicity: sequence of actions: happen "all at once"

(not interrupted)

⇒ **Ordering** : one thing happens before the other

sequence of instructions:

inst 1 → load addr, reg (mov in x86)

i 2 → add 1, reg

i 3 → store reg, addr

Example run : 2 CPUs

critical section : update to shared data

→ CPU$_1$	→ CPU$_2$	mem
		balance:
⇒ load ⇒ reg (100)	⇒ load ⇒ reg (100)	~~100~~
add 1 (101)	add 2 (101)	~~101~~
store ⇒ 101	store ⇒ 101	101

i 1

2

3

i 1

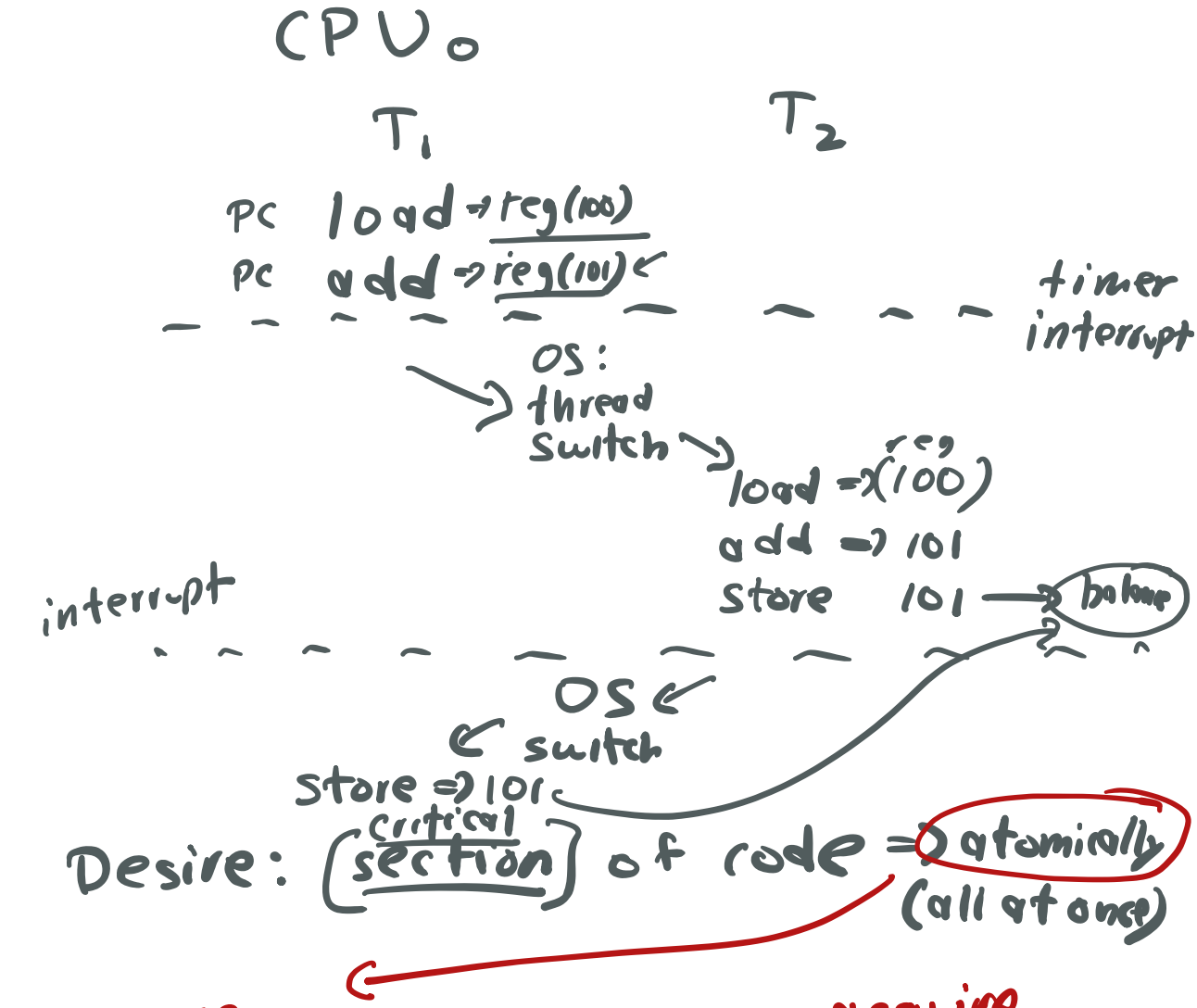
2

3

Problem: data race (or race condition)

outcome: indeterminant (outcome is not deterministic)

can even happen on 1 CPU:



Desire: [critical section] of code ⇒ atomically (all at once)

use (mutex) lock : lock, unlock OR acquire, release

[how to build?]

⇒ what is it?

⇒ lock ensures that only 1 thread is running w/in critical section

First Try: Spin Lock

```
int mutex = 0;        0 → free
                      1 → held

SpinLock() {
  while (mutex == 1)
    ; //spin
  mutex = 1;
```

test mutex branch