CSE 531: Distributed and Multiprocessor Operating Systems

Fall 2017

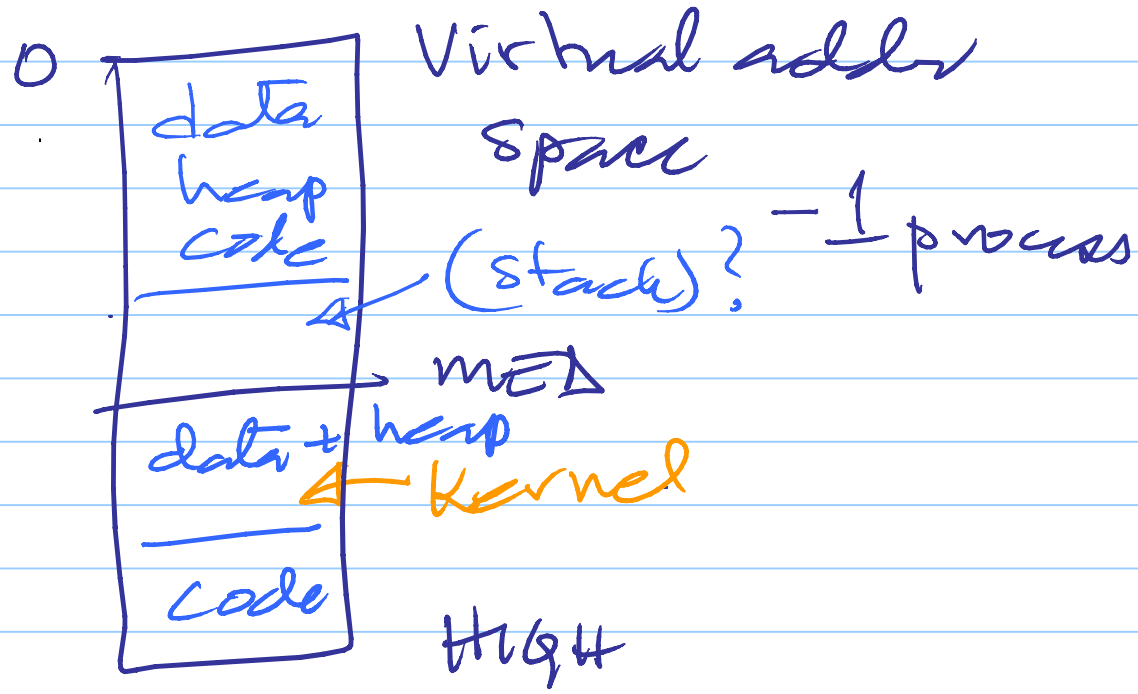
Class: 05 Date: 8/31

Note Title

Lock

user

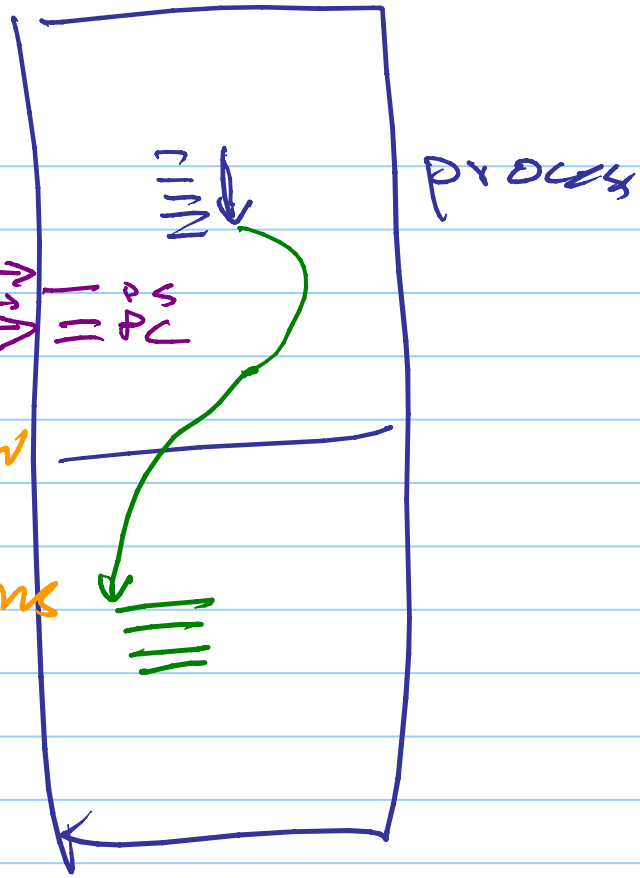
kernel



Interrupt

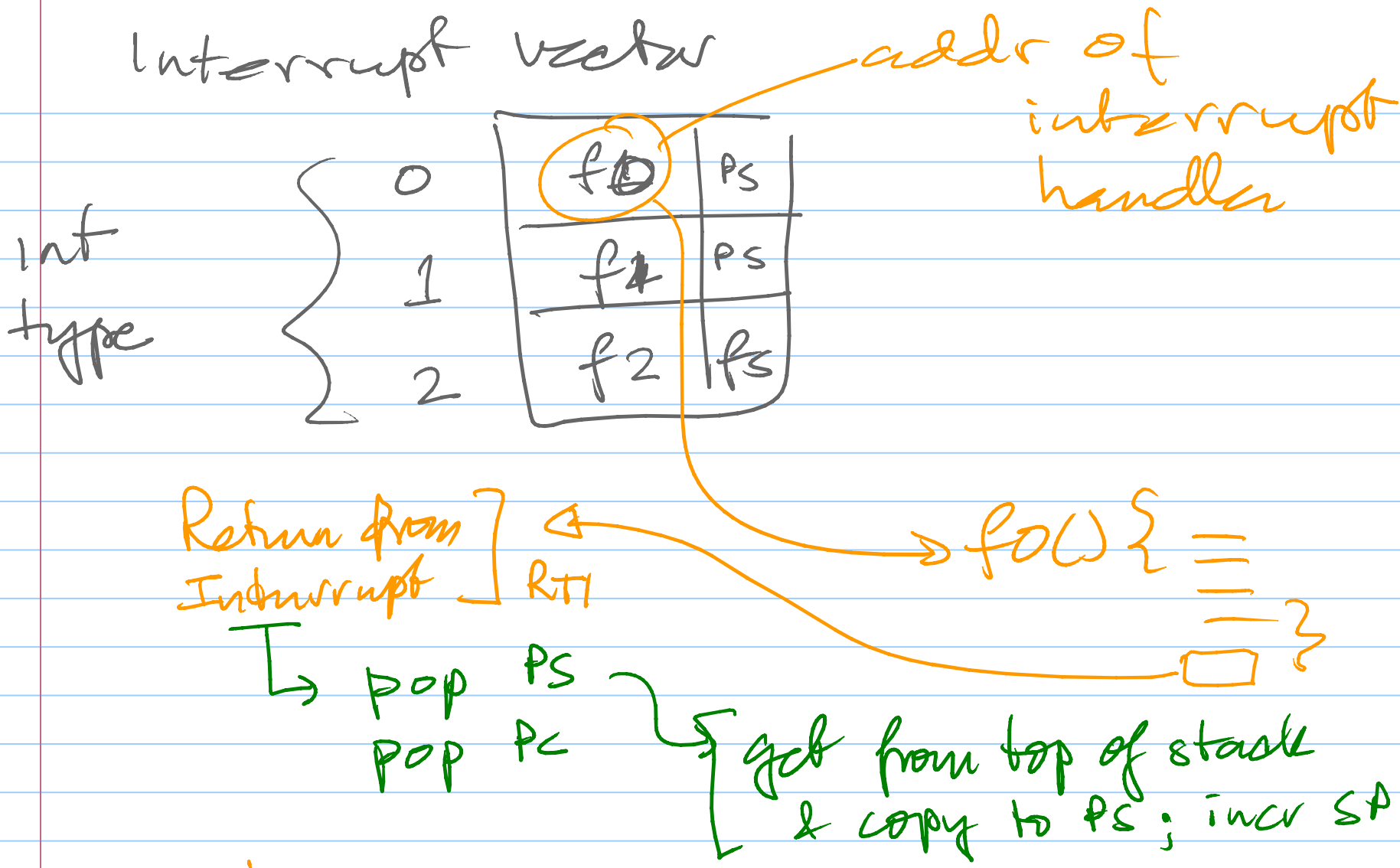
→ type (numeric code) → CPU

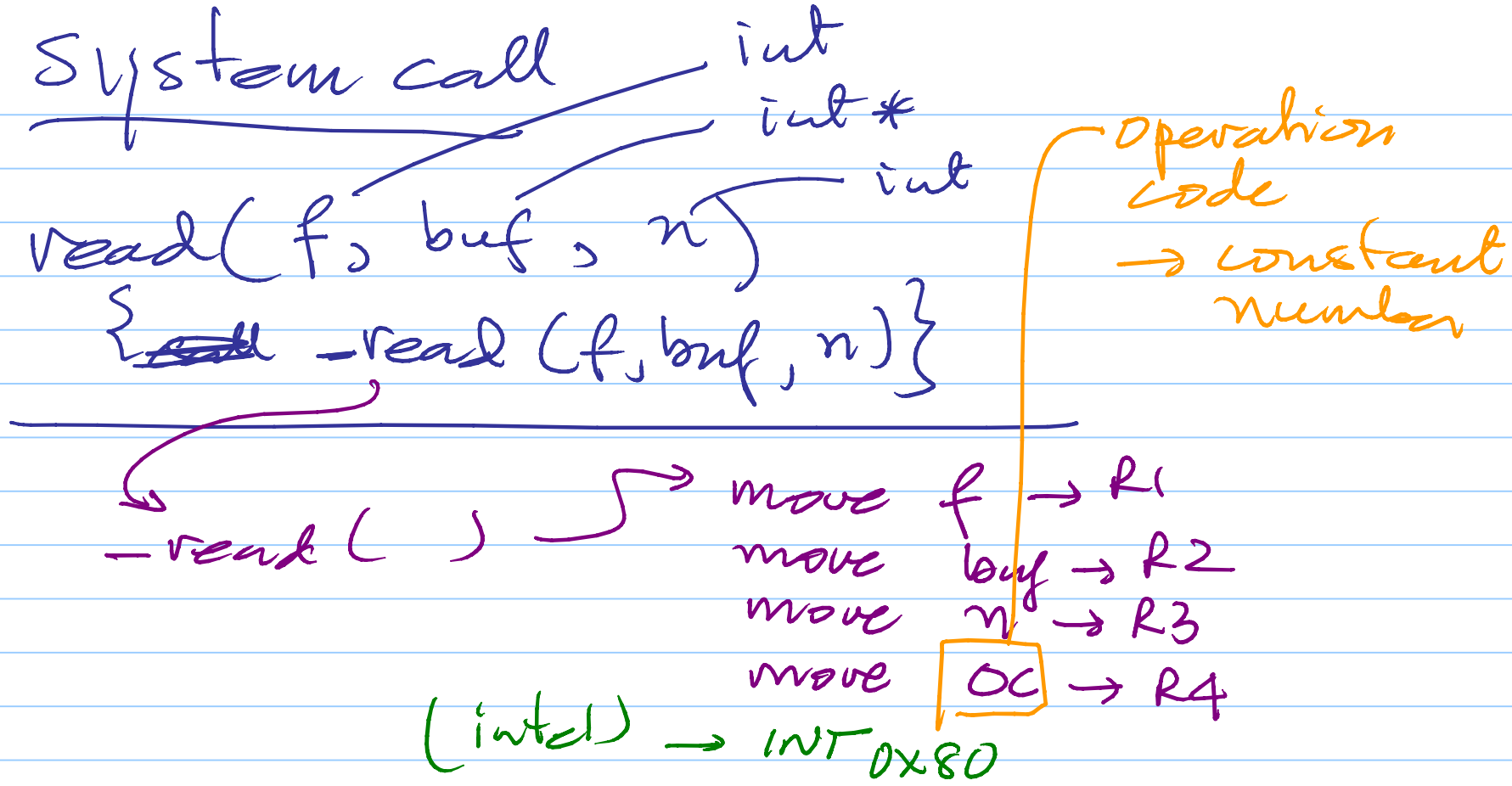
hardware



prog counter
instr pointer
program status
flags

on an interrupt {
push PC/IP
push PS/FL
Load PC, PS from interrupt vector


Interrupt vector
addr of interrupt handler
int type
0
1
2
f0
PS
f1
PS
f2
PS
Return from Interrupt
RTI
f0() { ... }
pop PS
pop PC
get from top of stack & copy to PS; incr SP


System call
int
int*
int
read(f, buf, n)
{ call _read(f, buf, n) }
operation code
→ constant number
_read( )
move f → R1
move buf → R2
move n → R3
move OC → R4
(intel) → INT 0x80

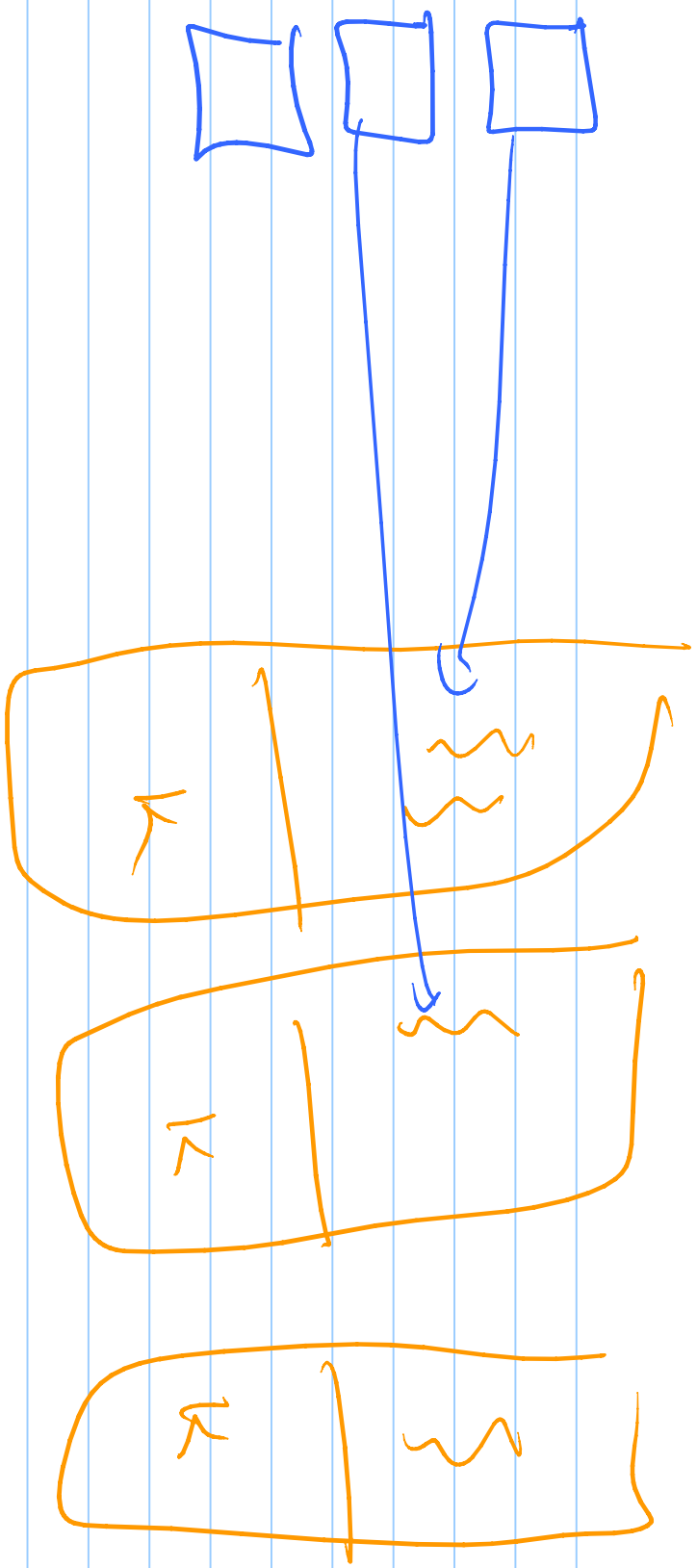
int handler 0x80

`syscall()

{ 0 → f0()
  1 → f1
  :
  :
}

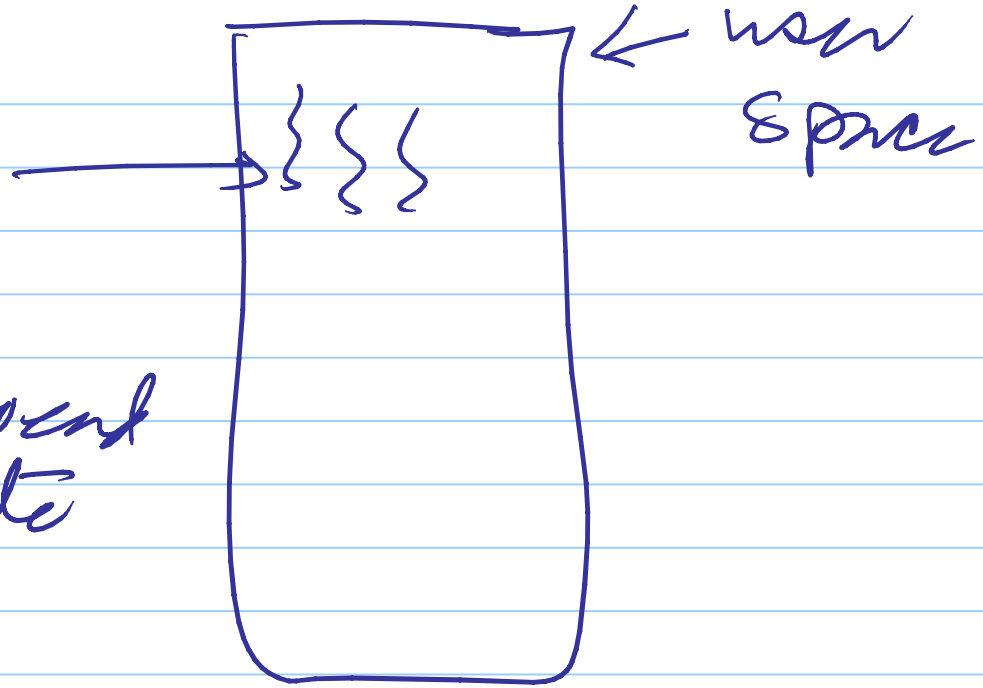
operation codes

all the syscalls



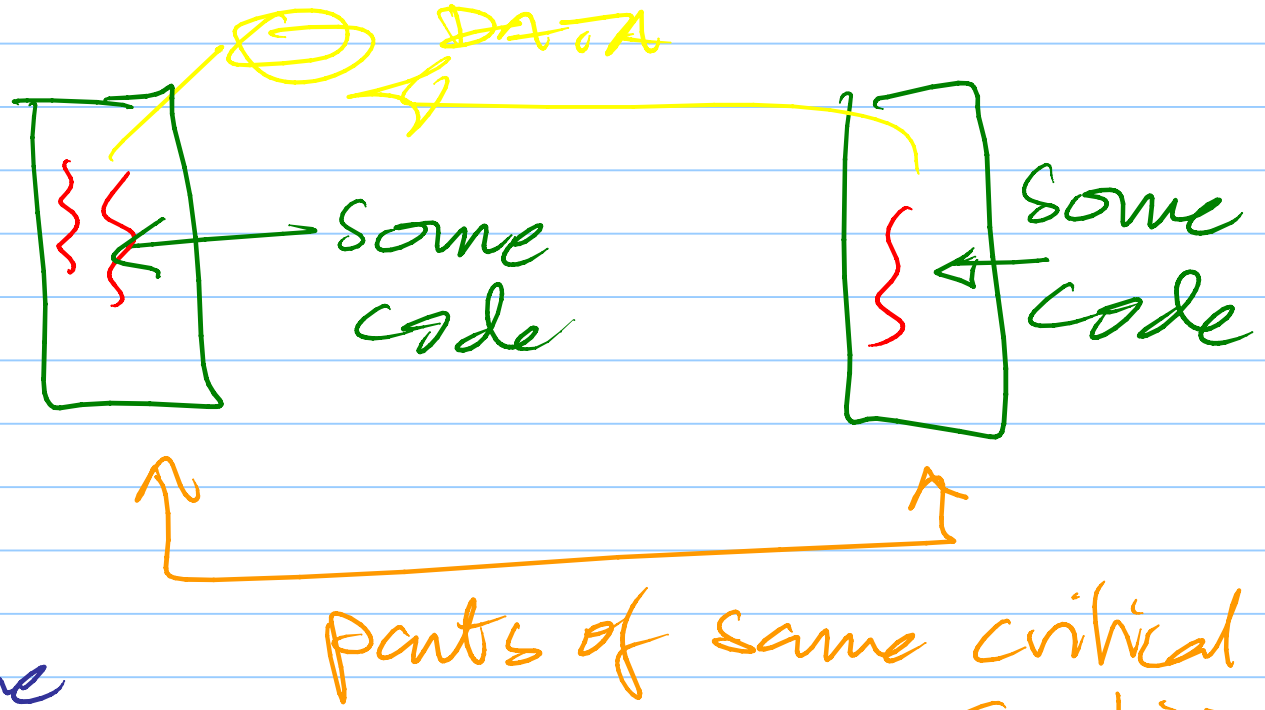
Thread

multiple threads (each thread has private PC, SP, Registers)

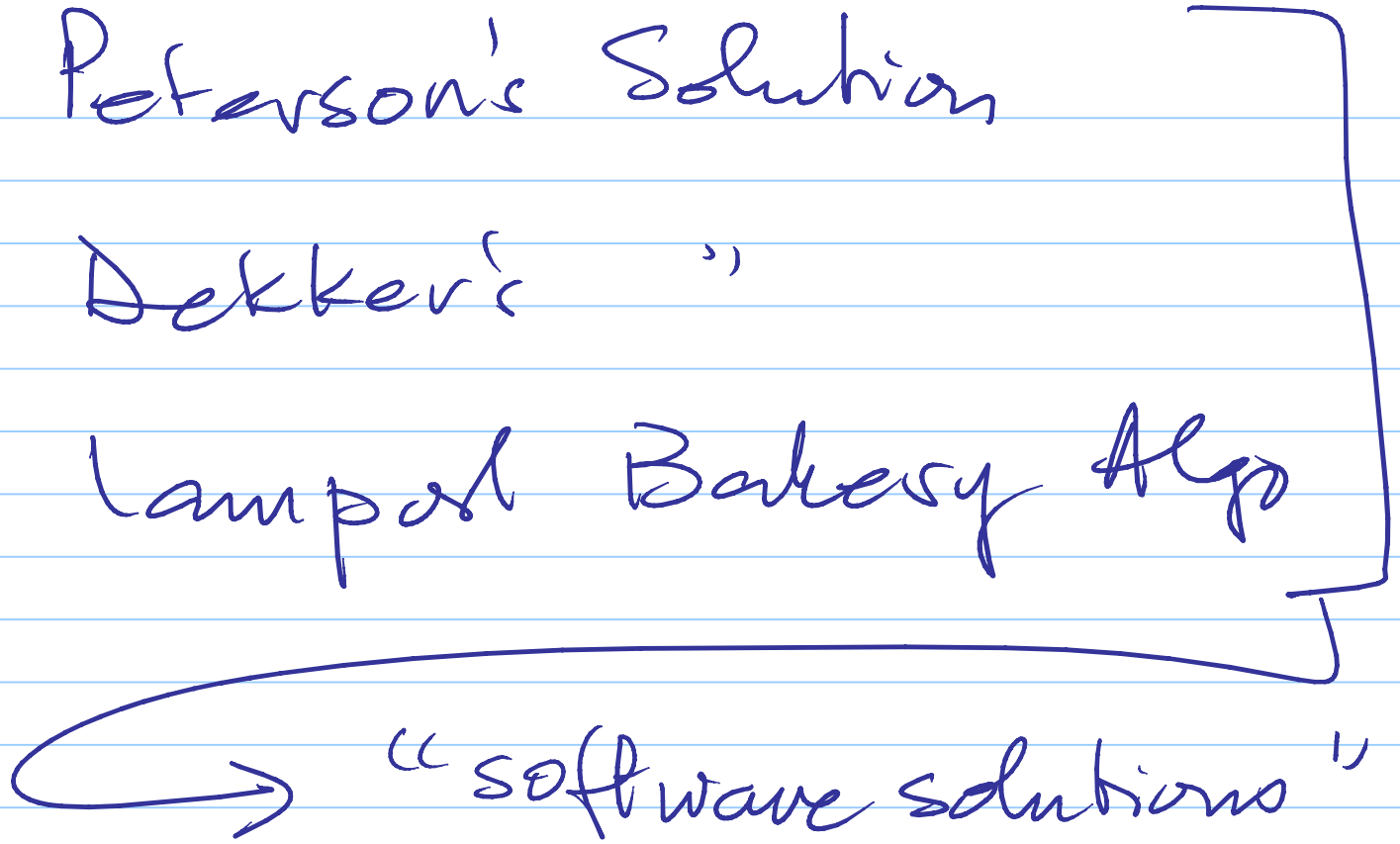


Critical sections

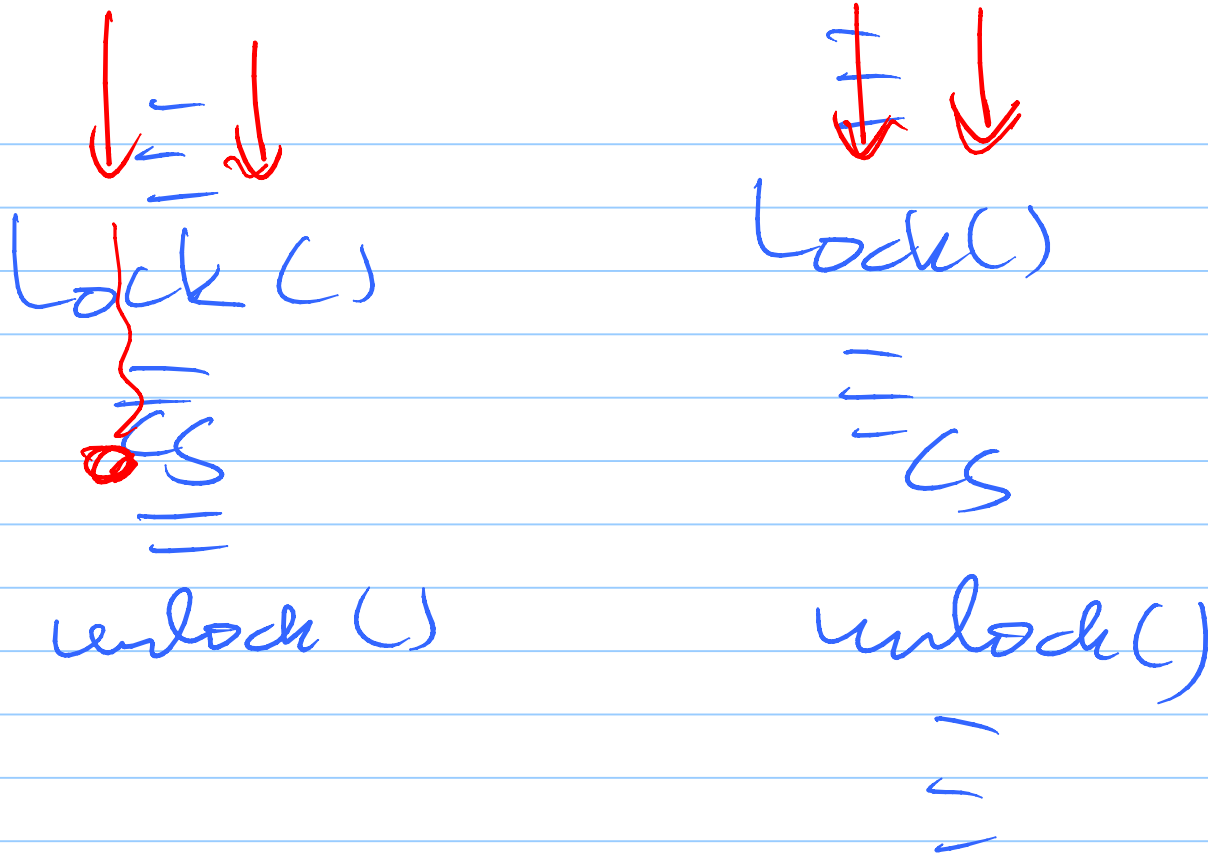
ONLY
1 (or zero)
threads
can be
in a
c.s. @ one
point in time



- Peterson's Solution
- Dekker's "
- Lamport Bakery Algo



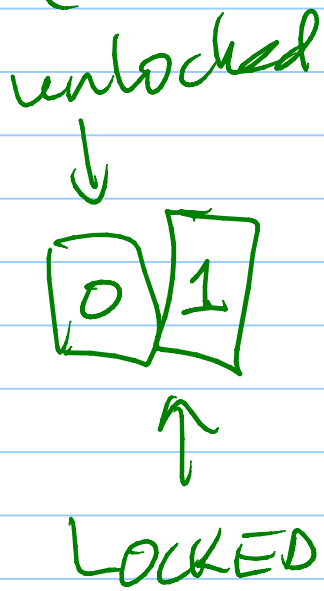
lock ?

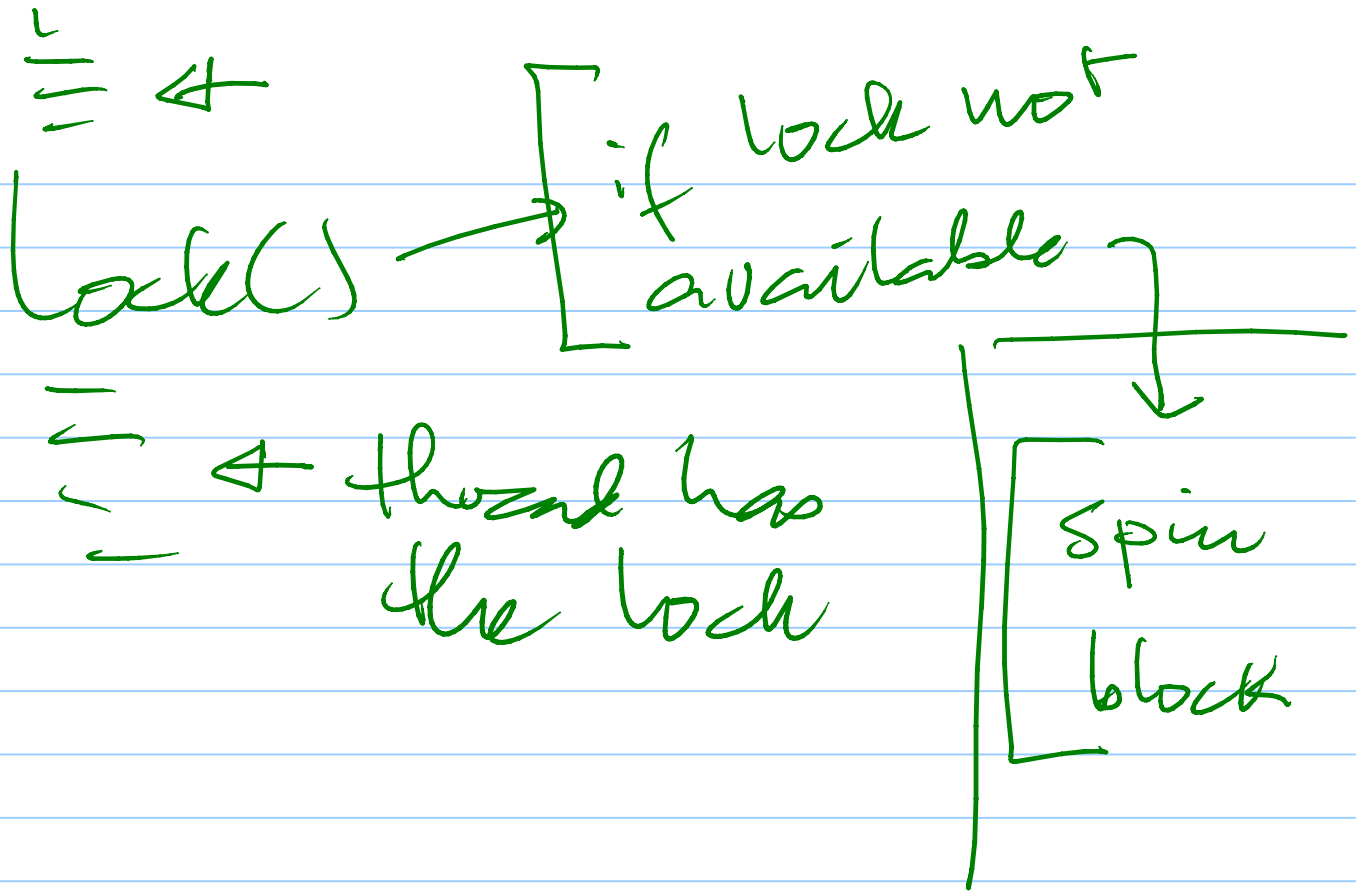


check if lock is available

→ yes → get it

→ NO → ??




lock()
if lock not available
thread has the lock
spin
block

Uniprocessor system

Does not work for multiprocessor

{

lock() ⟶ Disable all interrupts

⋮

unlock() ⟶ Enable ''

Multiproc

need a lock variable or flag

lock(x) →

- test & set
- compare & swap
- exchange (Intel)

x = 0

memory variable

```
Test+set (x)
{ int oldx;
  oldx ← x
  x ← 1
  return (oldx)
}
```

private variable

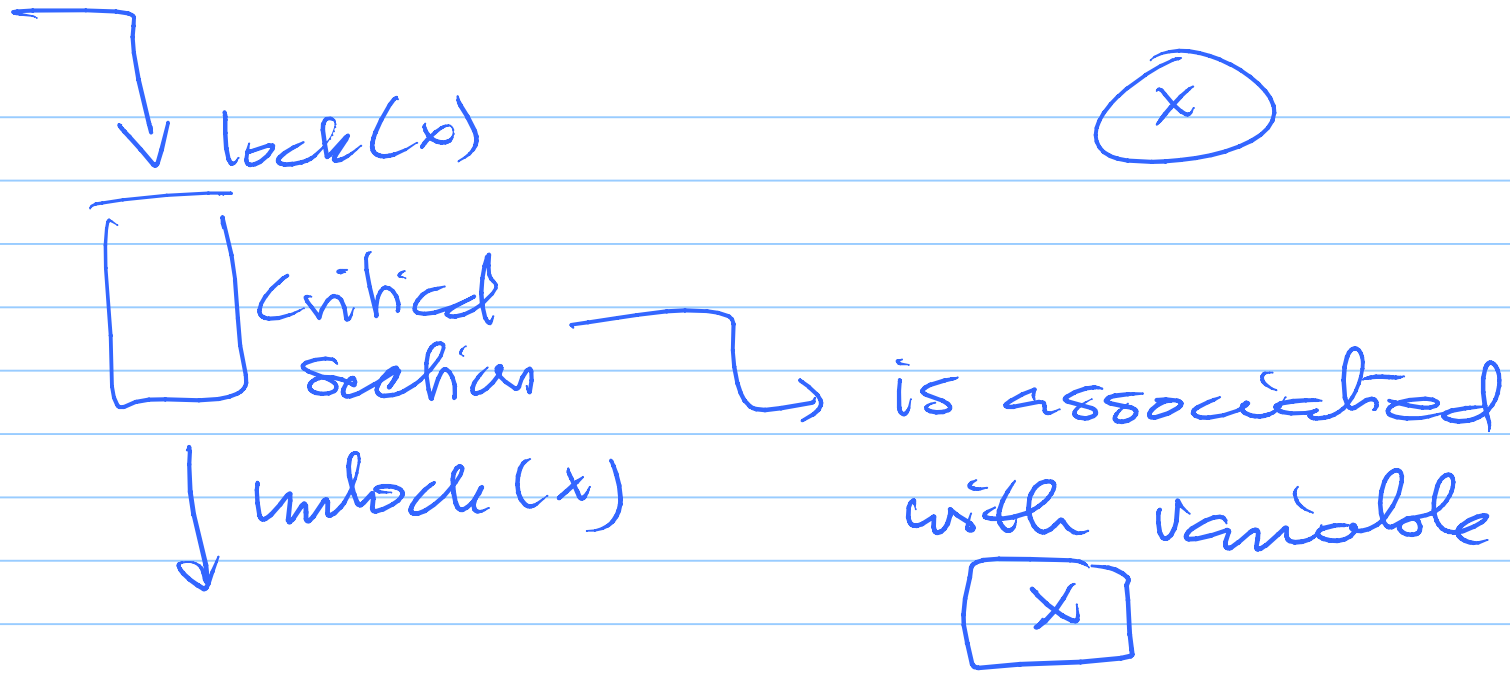
SPIN LOCK

```
lock(x)
{ while (test&set(x) == 1);
}

unlock(x) { x = 0 }
```

lock variables → the arg to lock(x)

- 1 per critical section
- value ⇒ do NOT check
- only use lock/unlock functions



Good → fast for small C.Sections

→ usable in user space

BAD Spin locks — Do not use

① on uniprocessor ⟹ ?!

② Long critical sections cause too much spin

UNIPROC

Thread starts spinning --.

MUTEX Locks (semaphores)

↳ does not spin

↳ uses spin locks to implement (or int Disable on UNIPROC)