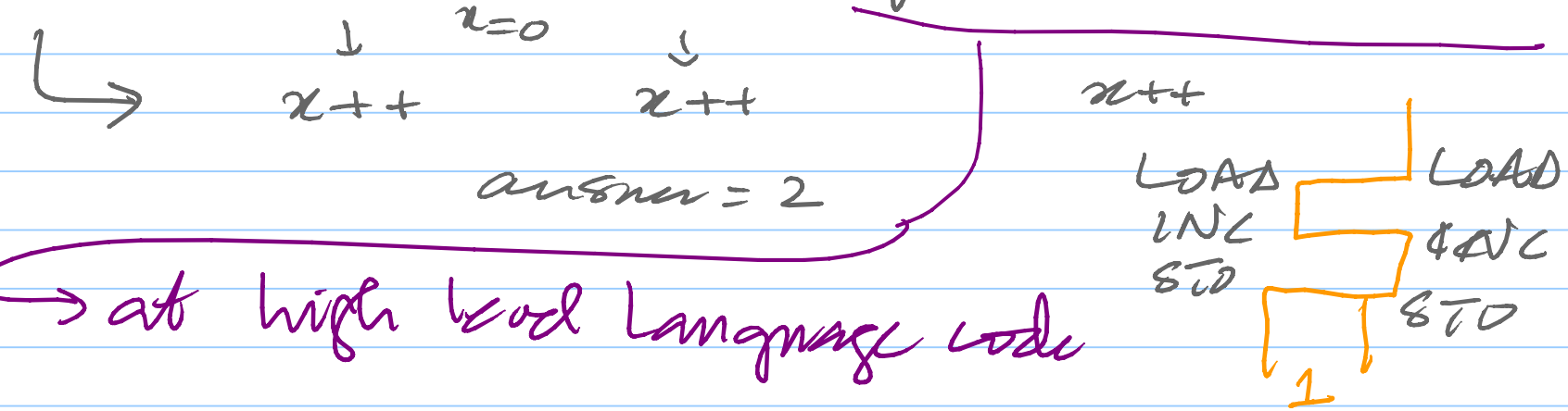
Concurrency / Parallelism

- happens
- multiple processes
  - multiple cores
- in kernel
- threads → applications

Reentrant or thread-safe code

- A concurrent execution will be equivalent to some sequential execution



Do it

- no shared memory
- locks (or interrupt disable
(- synchronization) ← different

Locks

→ set a lock
→ release a lock

- variable
- shared var
- not used for any other purpose

int a, b, c

→ want to lock "a"

— why?

— use another var ⇒ La

Lock(La)
@
unlock(L)

T_1

:

lock(La)

[a = ...
a = 1 + a + 1

unlock(La)

T_2

- Lock(La)

print(a)

- unlock(La)

Lock(variable) → implementation?

→ do not implement your own locks

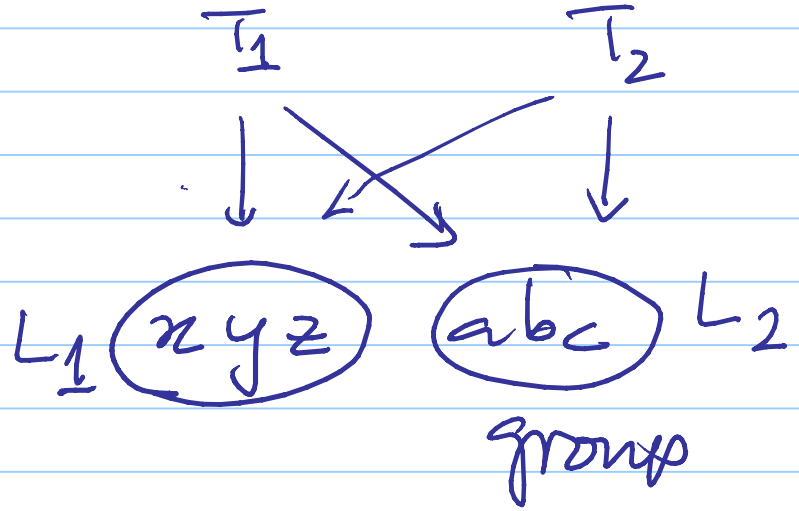
→ do not modify any existing impl

→ do not look at lock variables

Disable interrupt (uniproc)
— global

Spin lock
— not global

many typos



Spin lock

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

unlock(x) → x = 0;

```
int test_and_set (int *lock_pointer);
    .globl test_and_set

test_and_set:               # old val return in %eax
    movl $1,%eax            # get new_value into %eax
    movl 4(%esp),%edx       # get lock_pointer into %edx
    lock                    # next instruction is locked
    xchgl %eax,(%edx)       # swap %eax with what is stored in (%edx)
                            # ... and don't let any other cpu touch that
                            # ... memory location while you're swapping
    ret                     # return,..old value in %eax
```

spin locks

- fast
- kernel / user same
- can waste time if a thread spins — hopefully rare
- starvation possible

mutex locks

- does not spin*
- waits when resource is locked → not true in pthreads.
- FCFS*
- no starvation
- expensive

* NOT TRUE

Mutex lock

- 'mutex' in pthreads
- Semaphores → general method for OS academic classes

↓

- a data structure
- supports → init, P, V

Semaphore (Dijkstra) → integer

Variables are of type semaphore (int)

init(s, v) ⇒ s = v

0 or more

decr

P(s) { while(!(s==0)); atomic

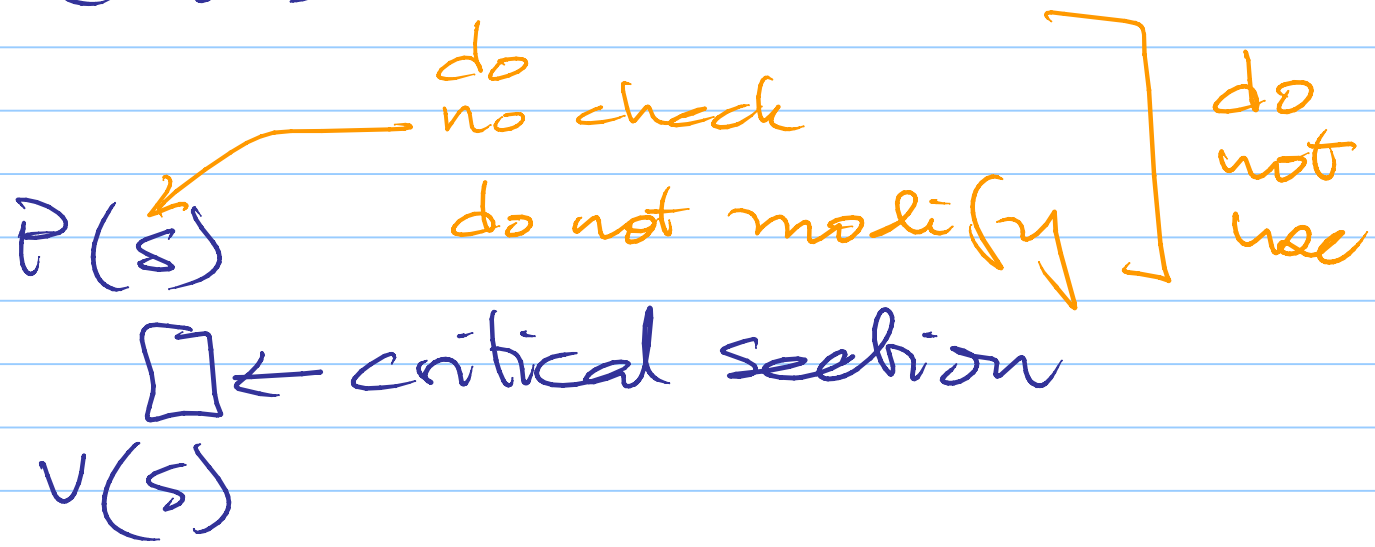
{s = s - 1; } }

V(s) { atomic { s = s+1; } }

DEFINITION

Critical Sections (lock)

init (s, 1)



while (s==0); → if(s==0)

if (s==0)

if (s==0)

else

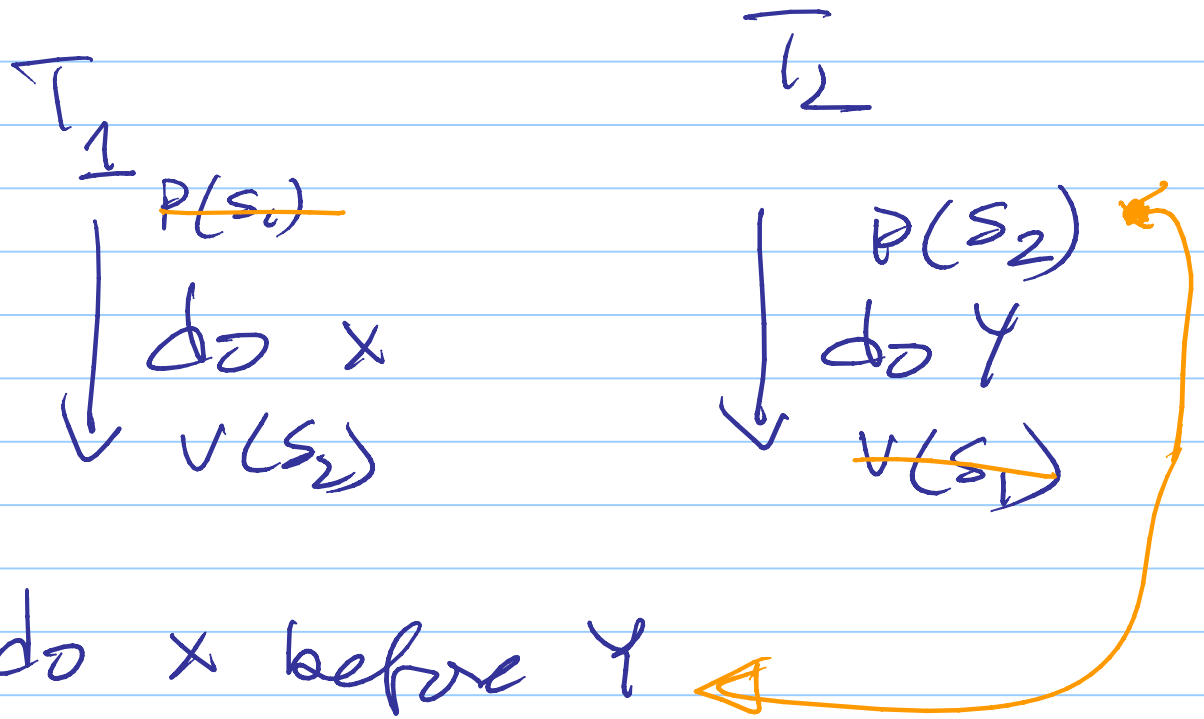
s--

S_1, S_2 - init to 1

T_1	T_2	T_3	T_4
$P(S_1)$	$P(S_2)$	$P(S_1)$	$P(S_2)$
=	=	$P(S_2)$	$P(S_1)$
=	=	=	
=	=	<	
$V(S_1)$	$V(S_2)$	$V(S_2)$	$V(S_2)$
		$V(S_1)$	$V(S_1)$

$S_1 \leftarrow 1$ $S_2 \leftarrow 0$

Synchronization



always do x before Y