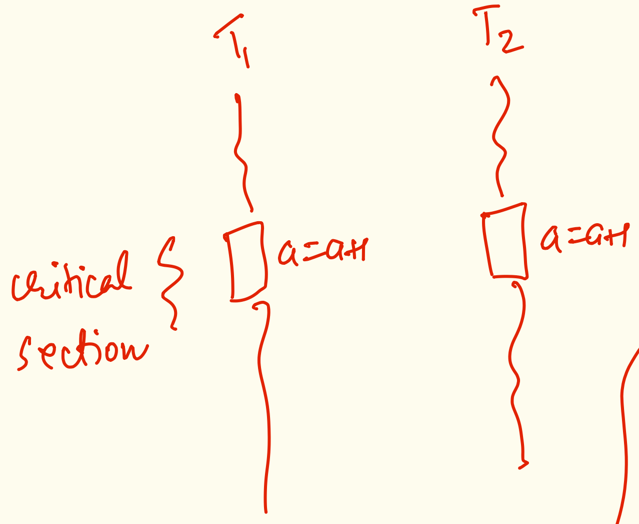
CS 219 Lecture 25

(1) race condition



(i) multiple threads of execution

(ii) "race" among threads to read - modify - write values in shared memory region.

compare

check

⇒ non-deterministic outputs / updates

⇒ depends on timing of accesses!

critical section

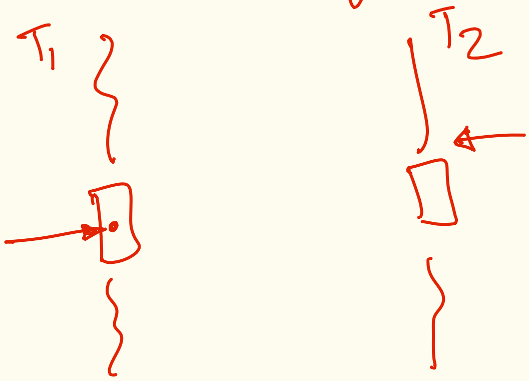
atomicity property to hold

- all or nothing property
- serializable or mutual exclusion manner

synchronization primitives

aim to provide mutual exclusion.

Q how to design these primitives.



wait till T1 is out of critical section

①

```
system      +  system
  call            call
  {                {
read()
```

- on read Kernel sends request to disk
- switches to another process which makes a system call

⇒ can lead to violation of mutual exclusion.

```
T1 {
sys_read()
  {
      //num. of bytes to read
  nbr = 1024;
  for( i=0; i< #nbr/512; i++)
      issueread(i, device);
hit disk
```

```
T2 {
  sys-read()
    nbr= 56;        (*)

read( fd, buf, size)
```

① # disable pre-emption

~ no switching

~ run system call to completion.

+ve : will work on single CPUs

- efficient for "quick" system calls.

-ves: ~ non work-conserving / inefficient

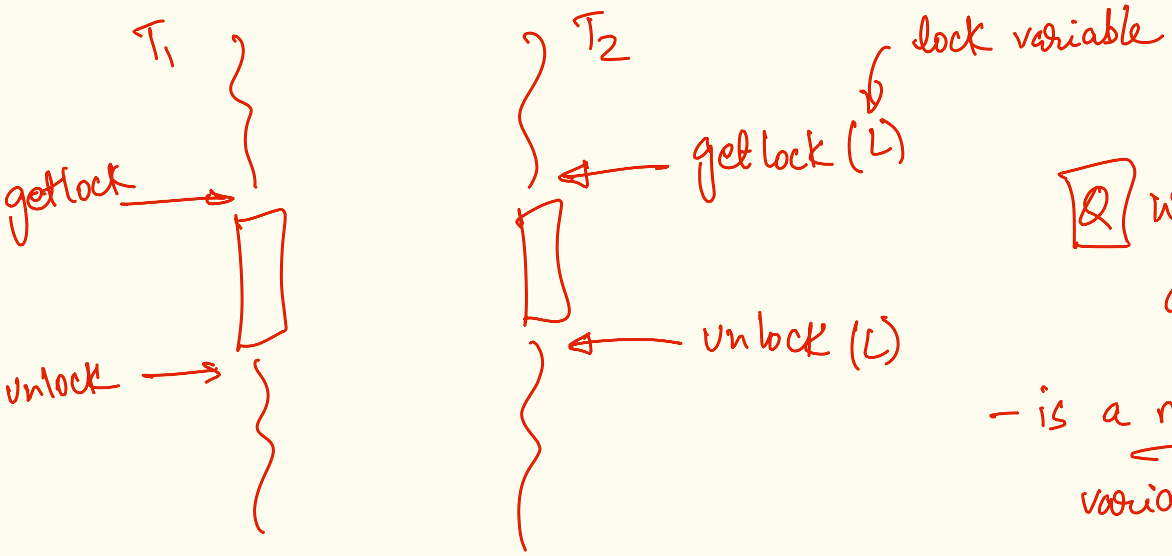
~ does not apply to multi-CPUs!

② disable interrupts

system call + (device) interrupts

- disable interrupts
- execute system calls to completion.
- same -ves as ①
  - does not apply to multi-CPUs
  - disabling interrupts
    - loosing interrupts!

③ Build primitives w/ ISA+OS



Q what is a lock?

- is a memory variable/loc^n.
- L=0 — available
- L=1 — taken

~ ~~wt~~ how to design a lock?

```
getlock (L) {
    if (L==0) {      ← T1    ] critical
        L=1;                 ] section
        return TRUE;
    else
        return false;
}
```

one-shot

needs some primitive support for retry.

read — modify — update
cmp
↑ interrupt ↑ interrupt

T1 T2

{ {

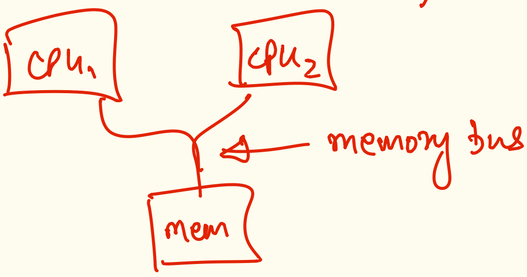
B B L=0!

```
unlock (L) {
    L=0;
}
```

(i) multiple 'C' instructions } multiple ISA instructions } can be non-atomic!

(ii) single C instruction

(iii) single ISA instruction → fetch-decode-execute pipeline



x86: provides a primitive to serialize the address bus.

lock; inc (%addr) ← locking address bus till instruction retires finishes.