CSE 531: Distributed and Multiprocessor Operating Systems

Class: 04 Date: 9/2

Fall 2020

Note Title

Reading

- Multicore architectures - NUMA
  hypercubes
- coherent caches
- race cond / critical sections

Recap

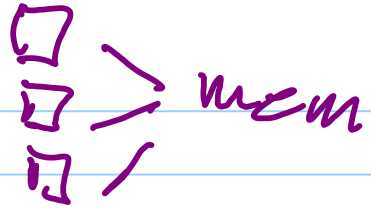
- Snoopy caches
- Cache coherence

- UMA → OS

→ NUMA → Barrelfish

- SISD ×
- MIMD -
- SIMD -

UMA → simple

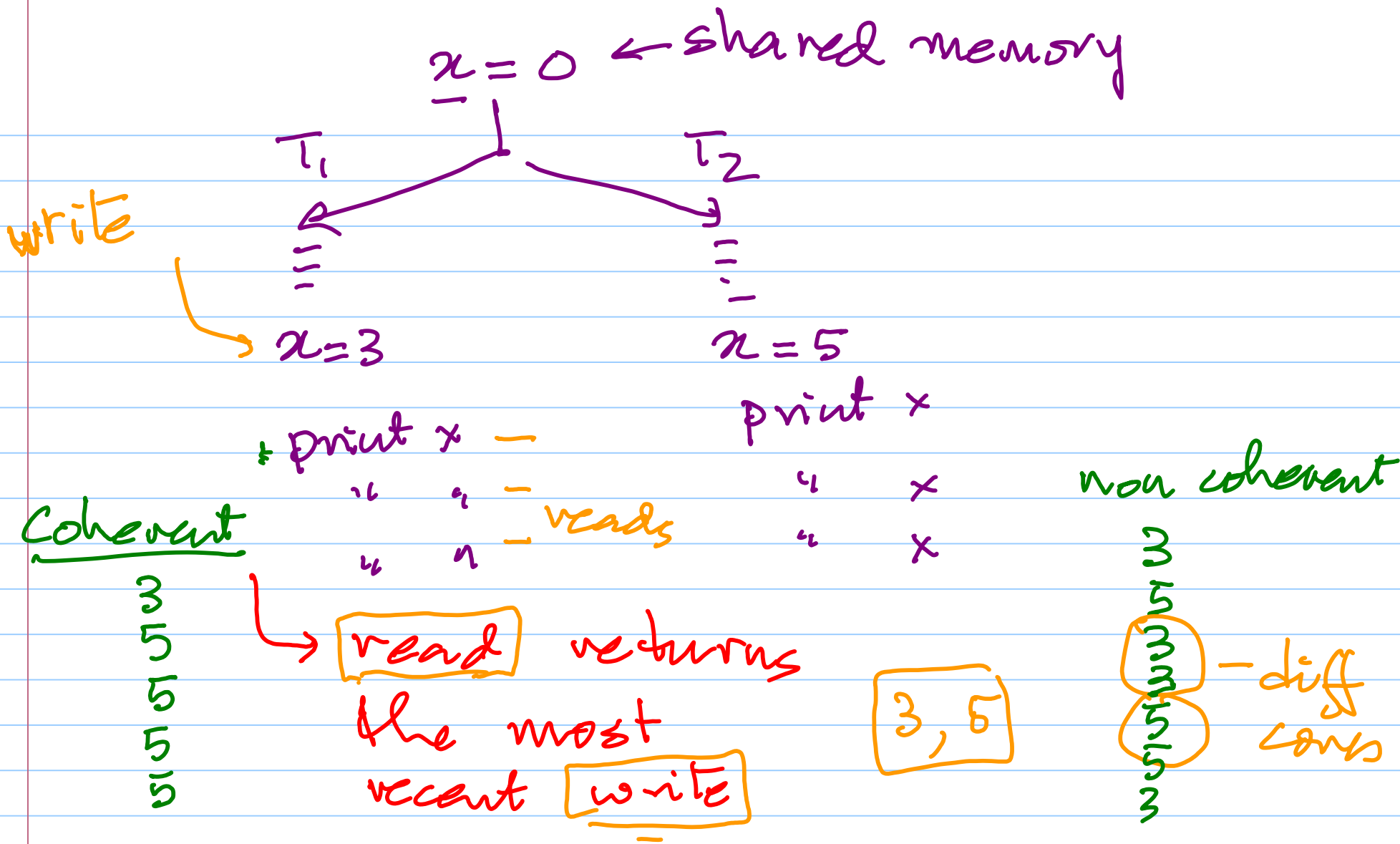


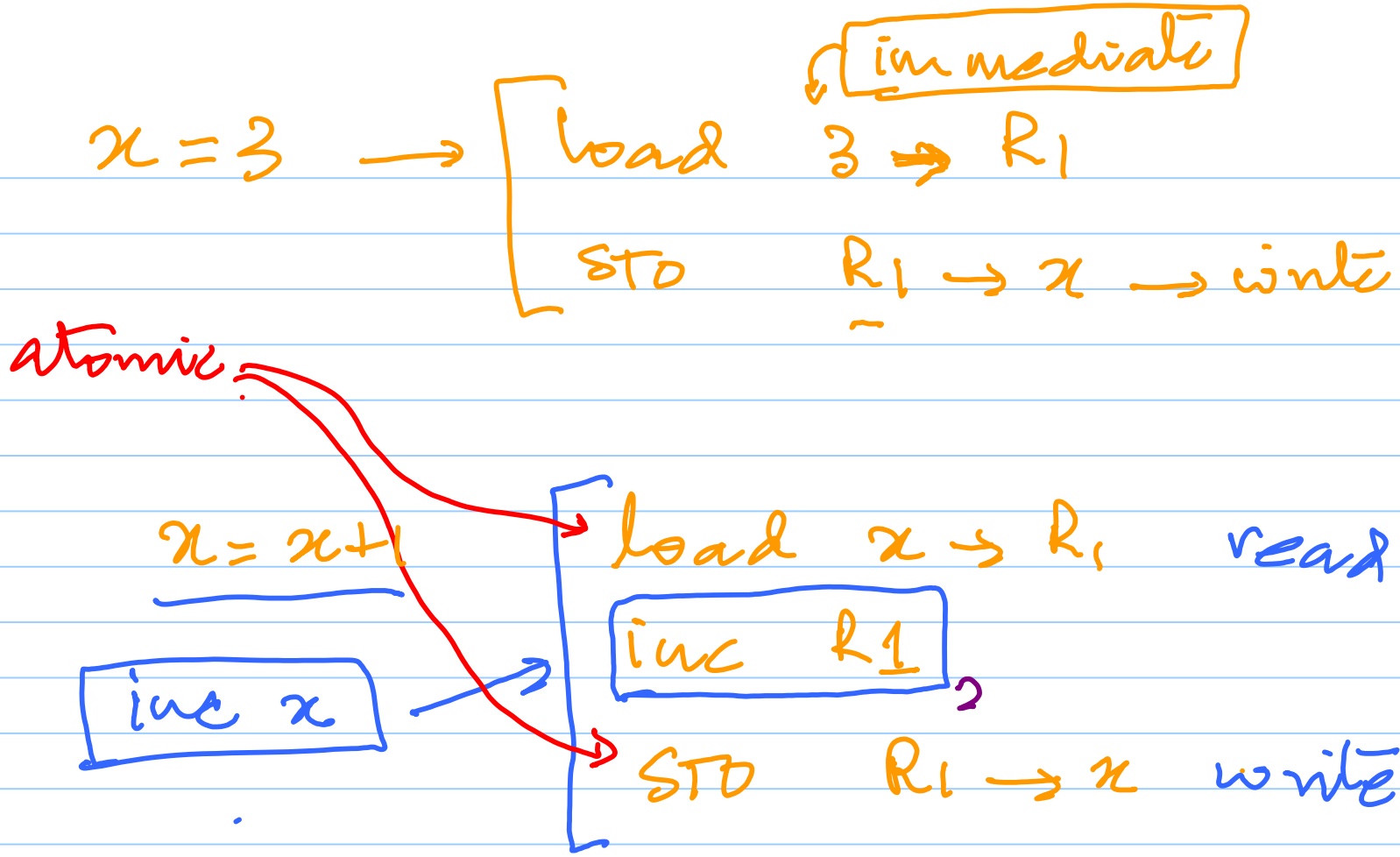
→ complex

↳ CC-NUMA → UMA

- caches
- pipelining
- hyperthreading

Caches — coherent
— non coherent


x = 0
shared memory
T1
T2
write
x = 3
x = 5
print x
" x
" x
reads
Coherent
3
5
5
5
5
read returns the most recent write
3, 5
non coherent
3
5
3
5
5
3
- diff cons


immediate
x = 3
load 3 ⇒ R1
STO R1 → x → write
atomic
x = x+1
inc x
load x → R1 read
inc R1
STO R1 → x write

Race condition

$x = 3$ $x = 5$

↓

$x = 3$ or 5

(not both)

$x = 0$

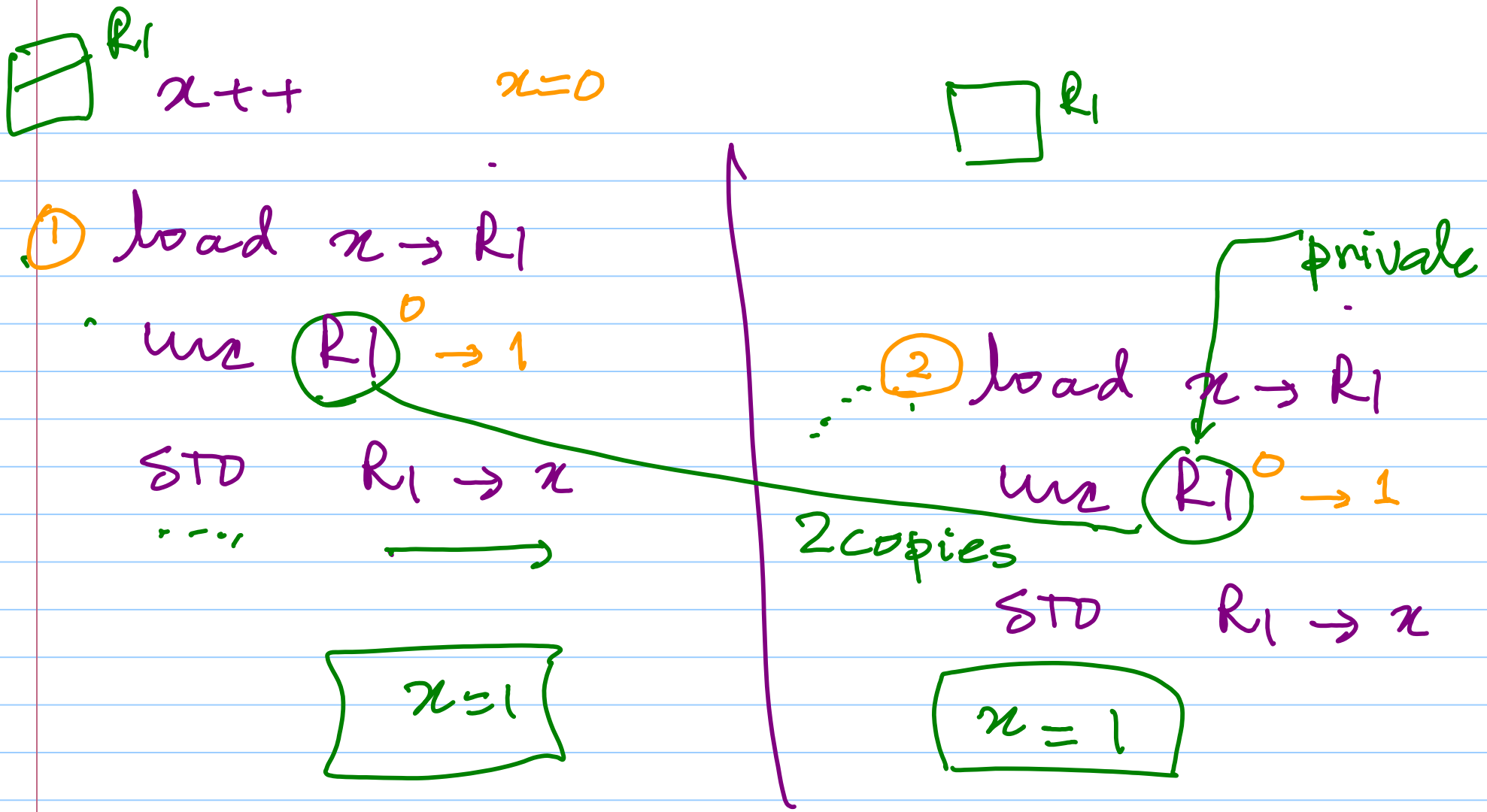
$x++$	$x++$
print x	print x
‘,	‘,
‘,	‘,

2	1		1	1			
2	2		1	1			
2	2		1	1			2
2	2		1	1			2
2	2		1	1			2

all possible

desired

always 1


R1
x++
x=0
R1
private
1 load x → R1
inc R1 0 → 1
STO R1 → x
2 load x → R1
inc R1 0 → 1
2 copies
STO R1 → x
x=1
x=1

program ⟶ 10 threads, 0 .. 9

each thread

{ id = getid()

private

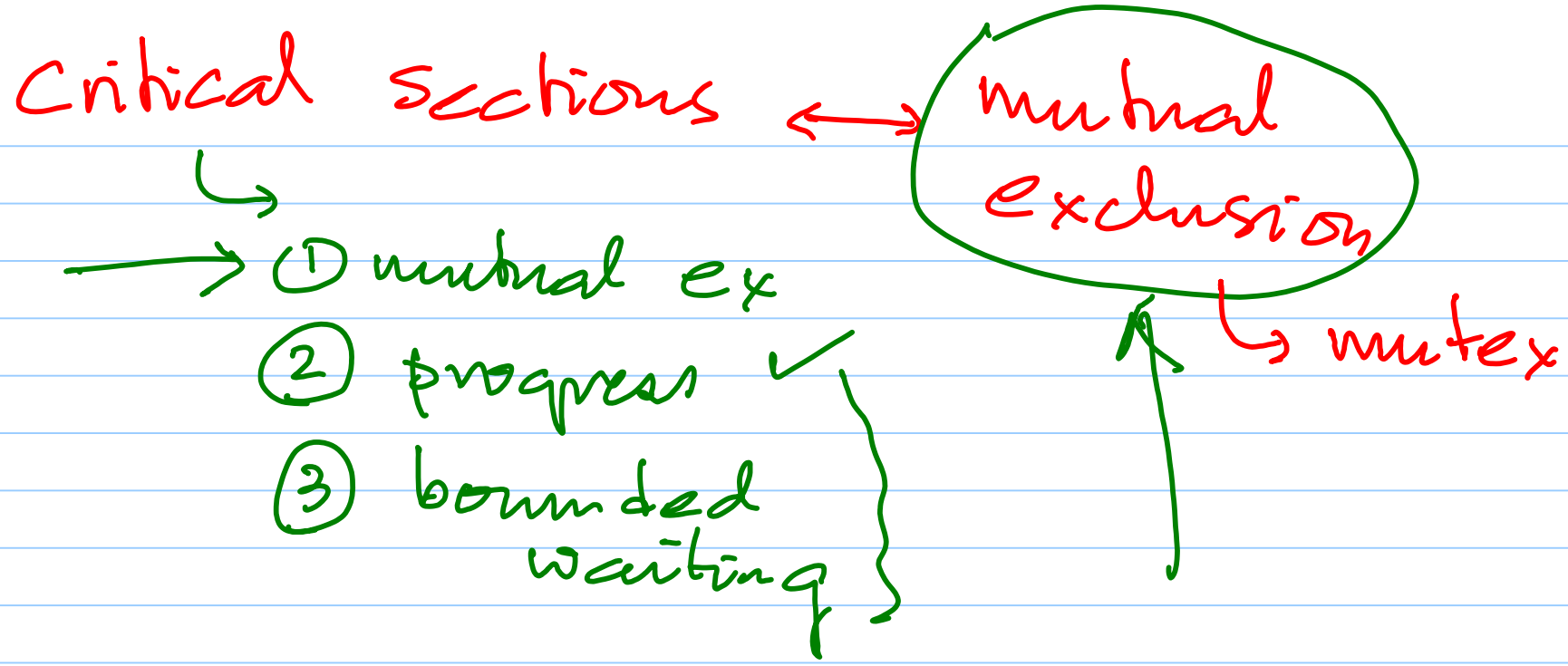
size = 5

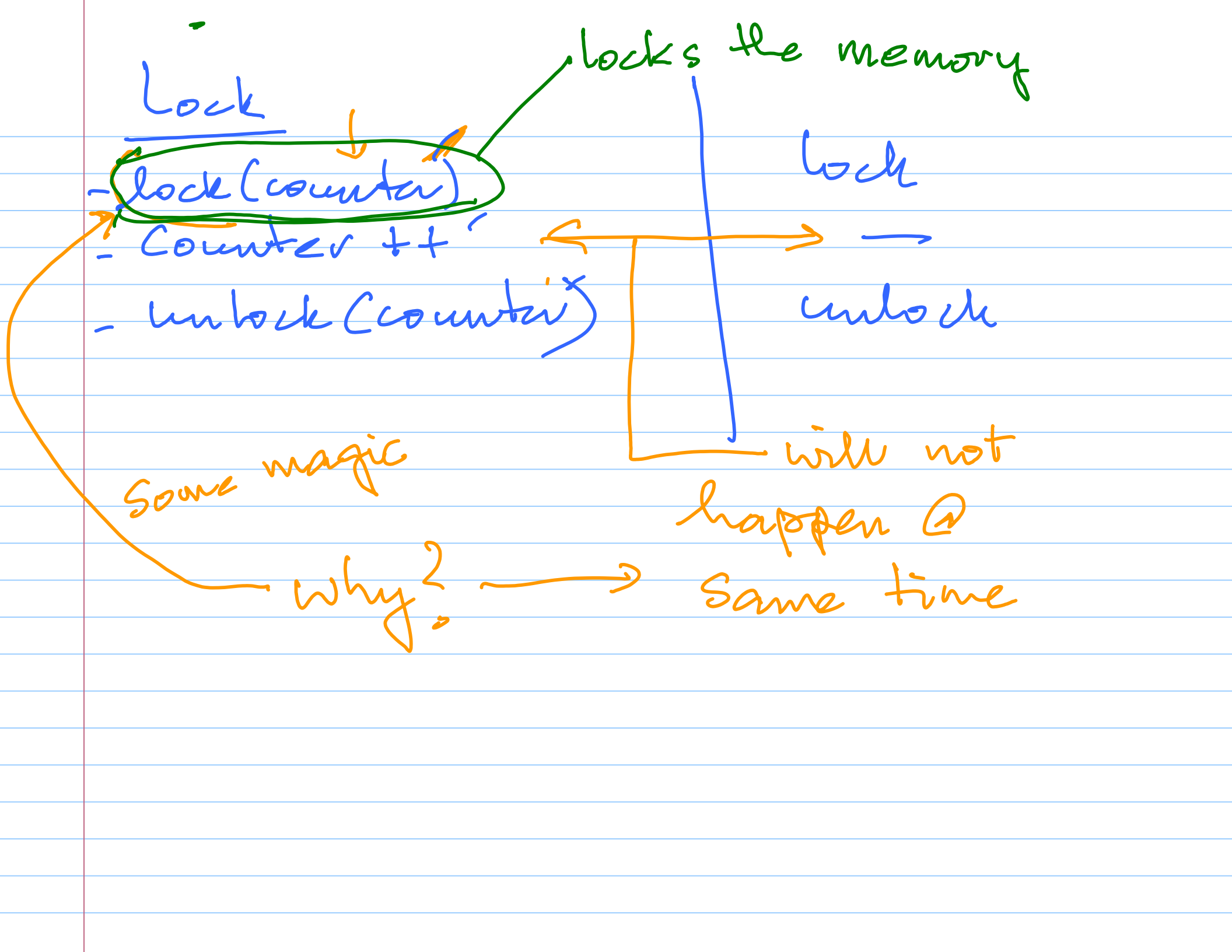
do work on

data[(i * size)] to data((i+1)size - 1)

counter ++ ← track of # of times

Atomic


Critical sections
mutual exclusion
1 mutual ex
2 progress
3 bounded waiting
mutex


Lock
locks the memory
lock(counter)
Counter ++
unlock(counter)
lock
unlock
will not happen @ same time
Some magic
why?

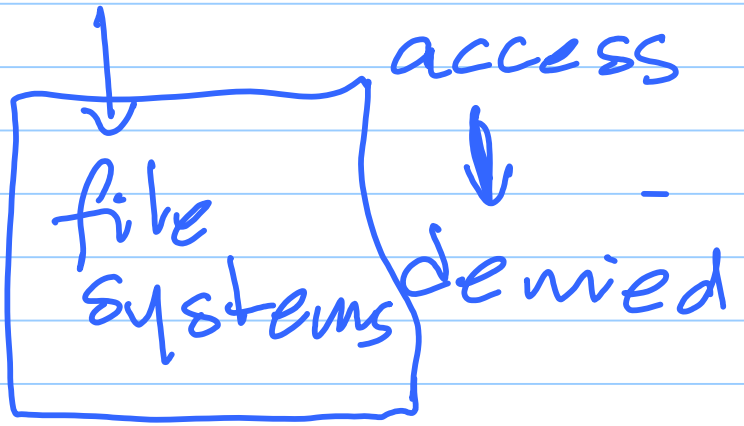
lock → advisory, mandatory

Built @ os level

← lock

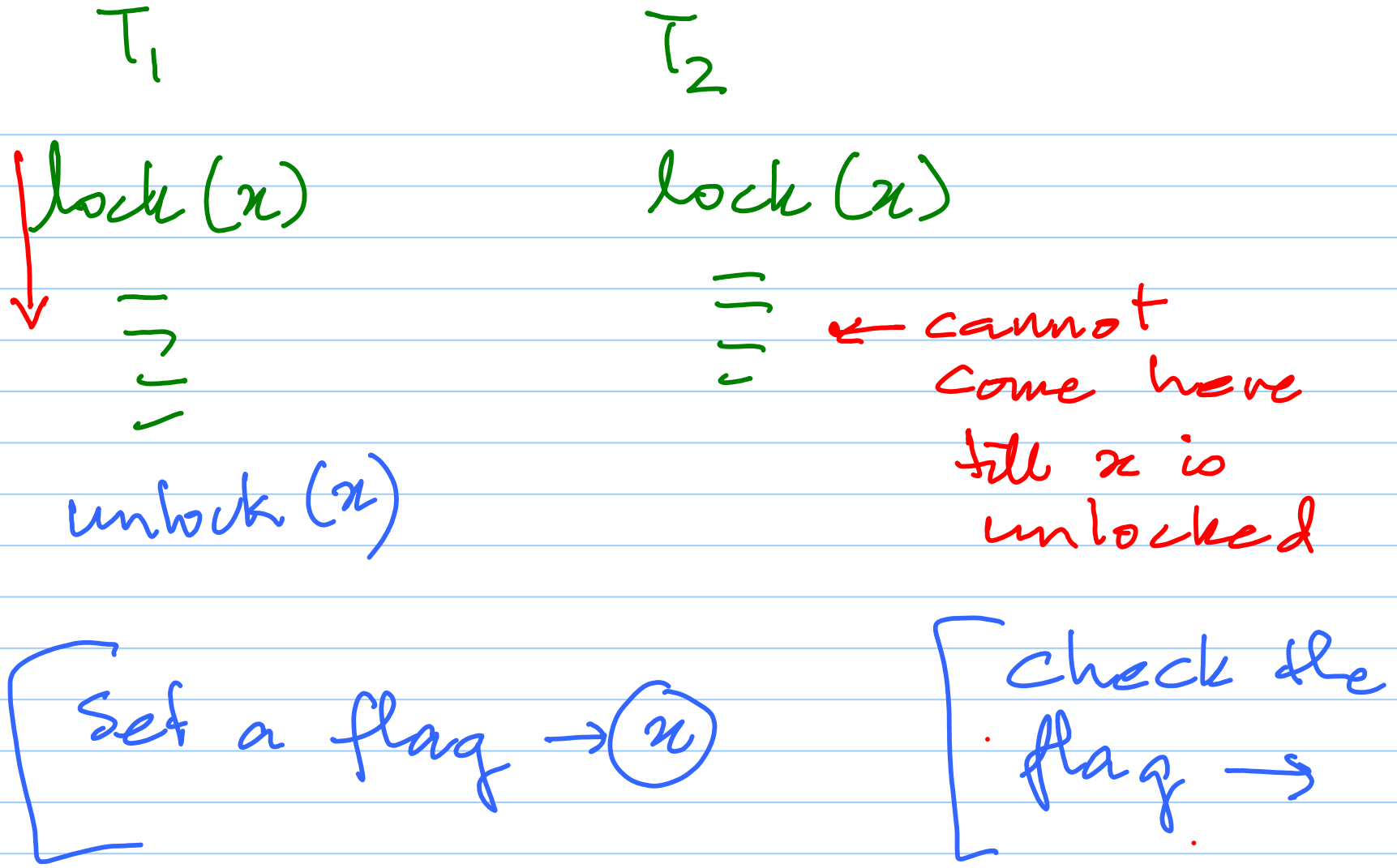
lock (x)

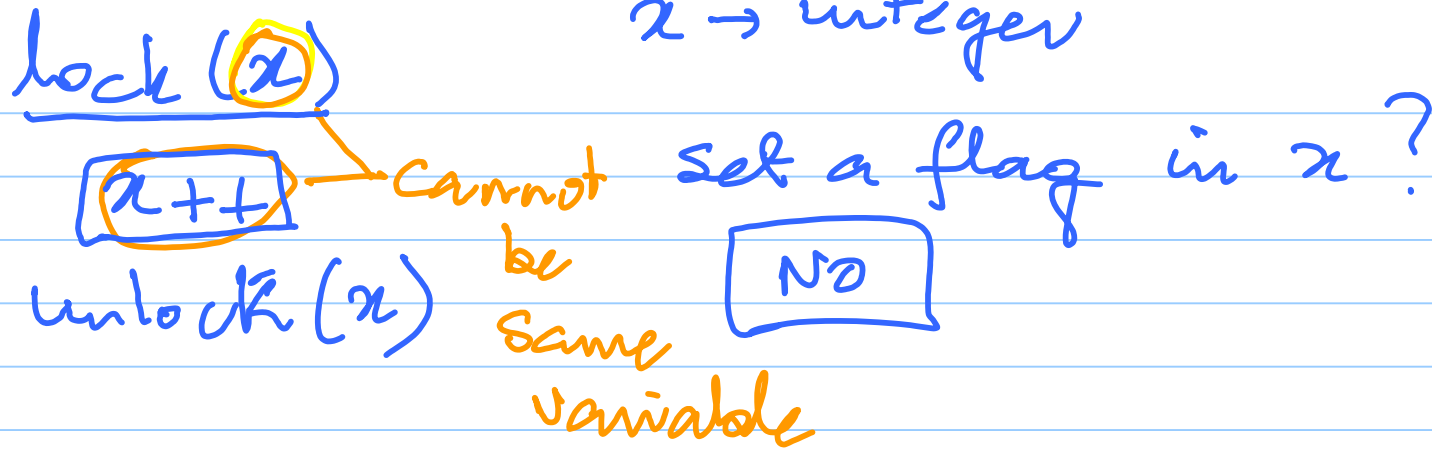
⋮



T_1 . T_2

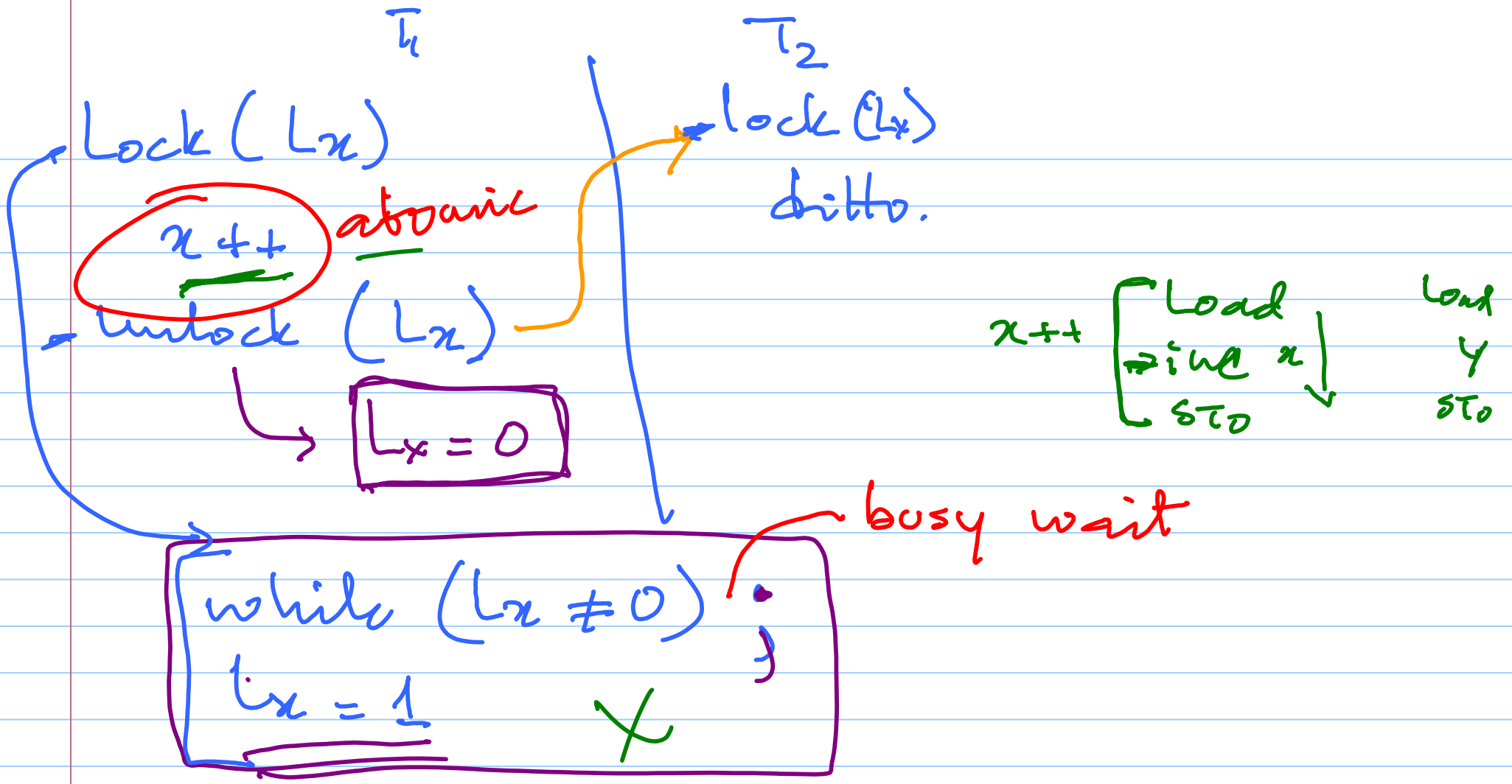
↳ can access x while it is locked


T_1
T_2
lock (x)
lock (x)
cannot come here till x is unlocked
unlock (x)
Set a flag → (x)
check the flag →



for every x (lockable item)

declare Lx → a lock variable


T1
T2
lock (Lx)
x++
atomic
unlock (Lx)
lock (Lx)
ditto.
Lx = 0
while (Lx ≠ 0)
busy wait
Lx = 1
X
x++
Load
inc x
sto
Load
y
sto

to get an atomic section

↳ we need a small atomic section

↳ [read followed by write → atomic

Atomic instructions →

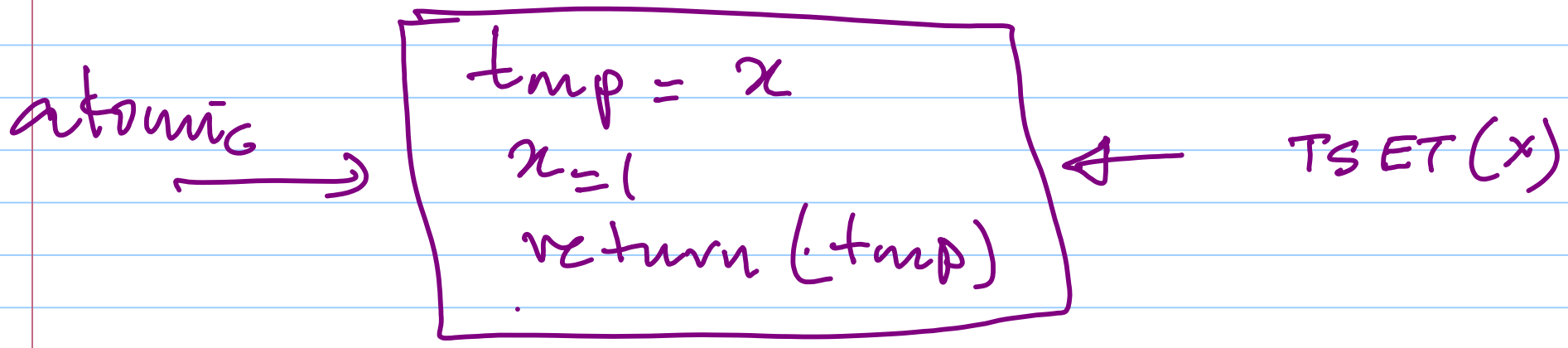
Atomic instructions

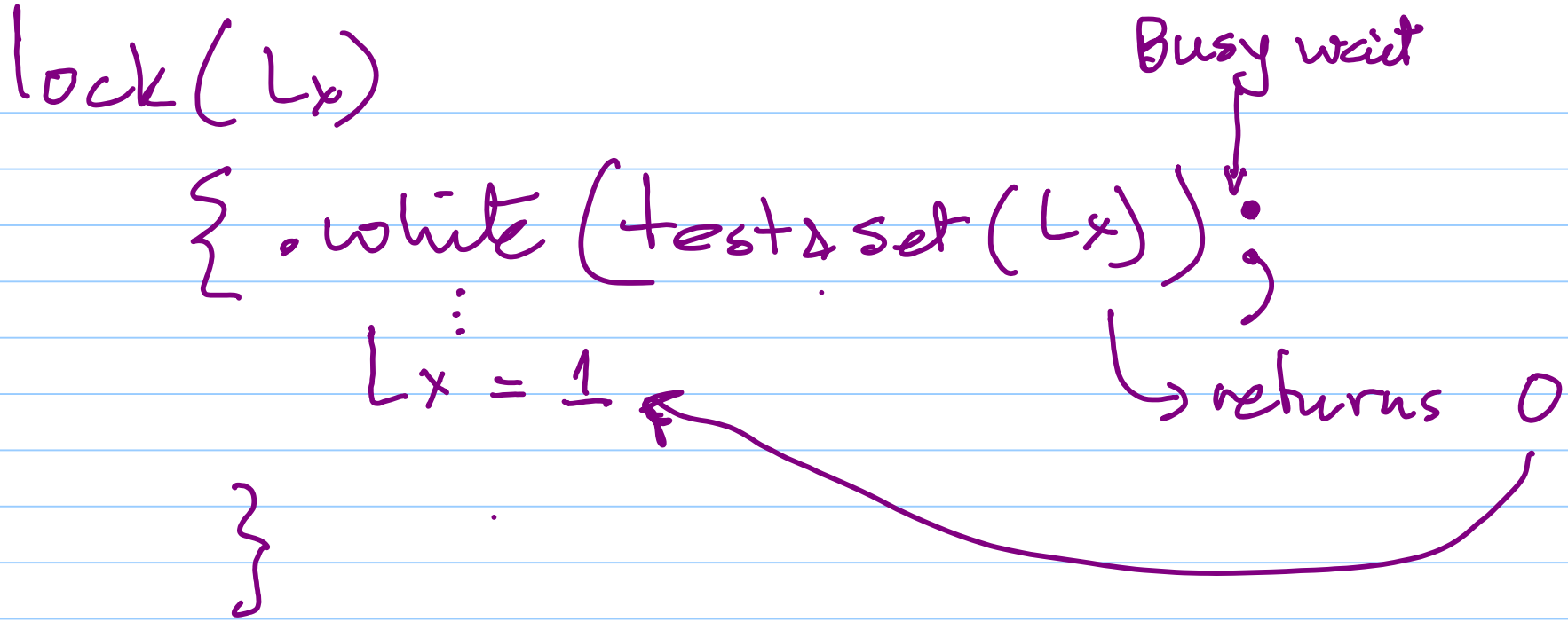
1. test & set
2. exchange → XCHG
3. Compare & swap
   → CMPXCHG

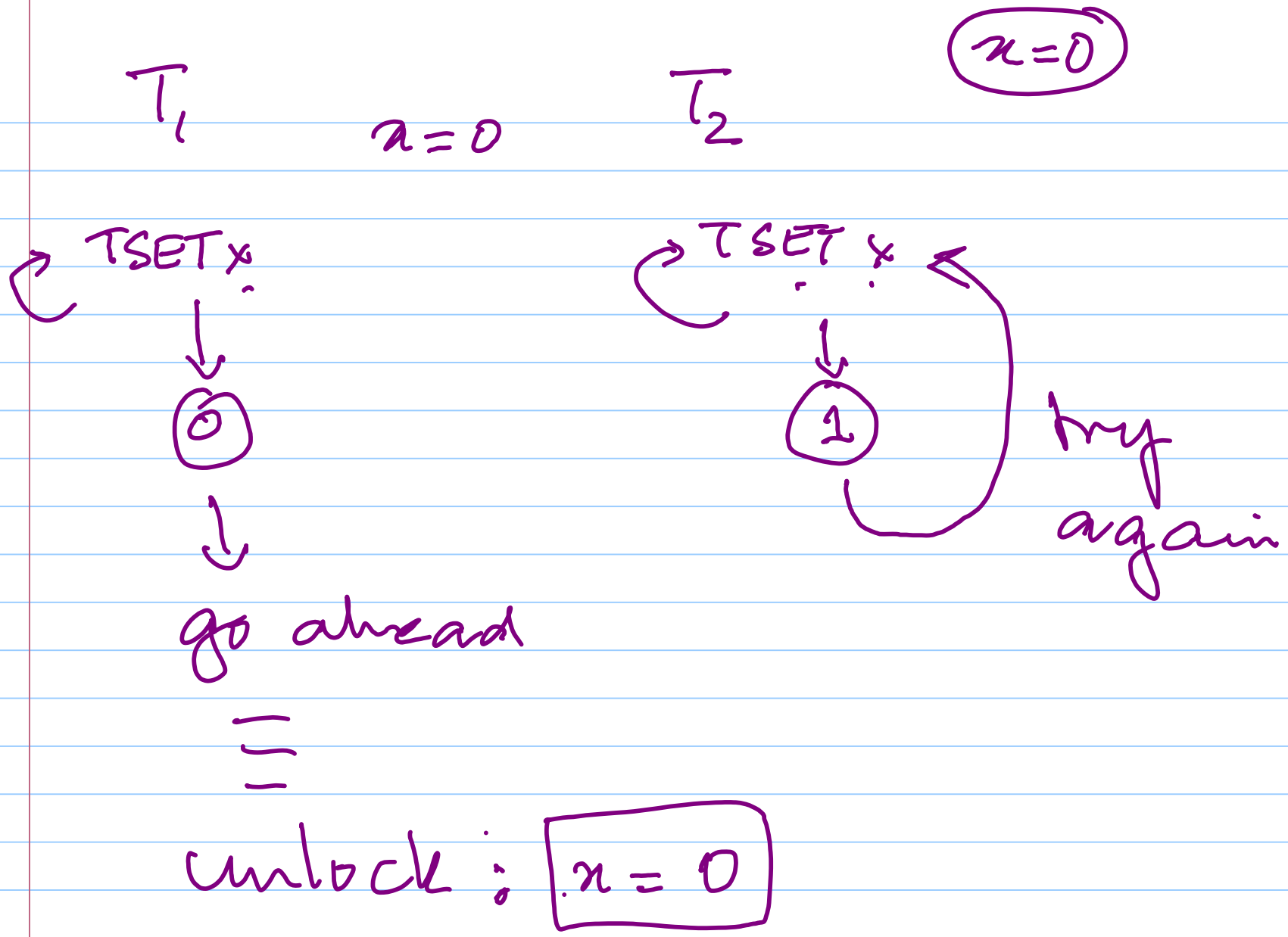
test & set (x)

→ 1 instruction

→ set x to 1 & return prev value




lock(Lx)
{ while (test&set(Lx));
Lx = 1
}
Busy wait
returns 0


x=0
T1
x=0
T2
TSET x
0
go ahead
=
unlock : x = 0
TSET x
1
try
again

test & set

↳ used to build

spin locks

↳ used heavily

ans = Tset(Lx) ans = Tset (Lx)

↳ 1 gets 0 as ans
all others get 1