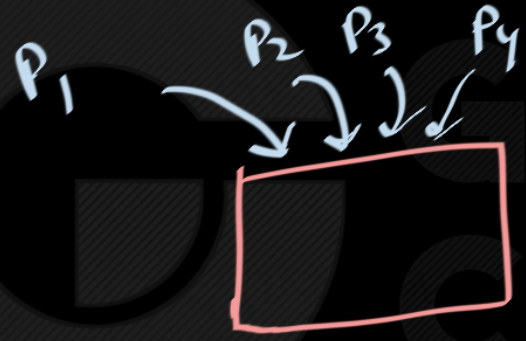
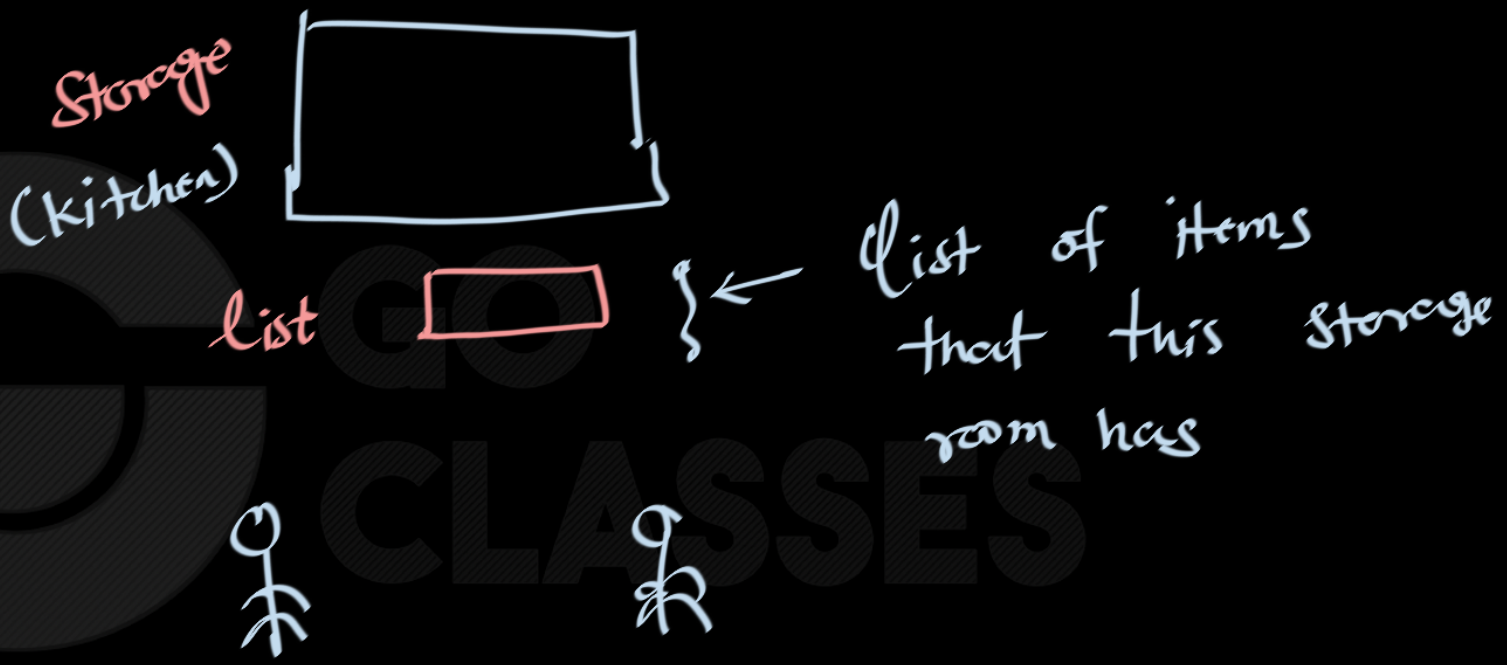
Next Topic:
Synchronization

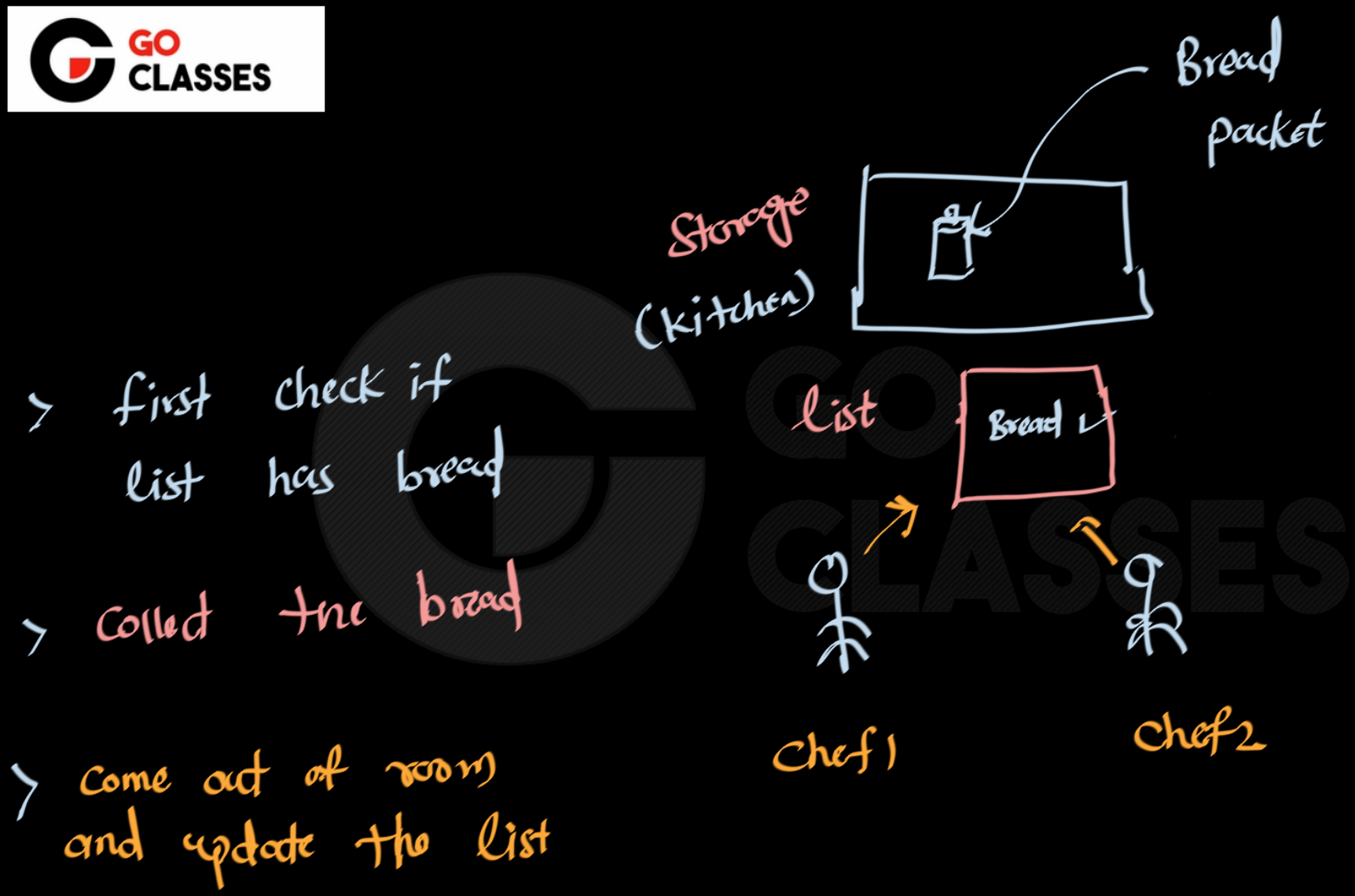
GO CLASSES

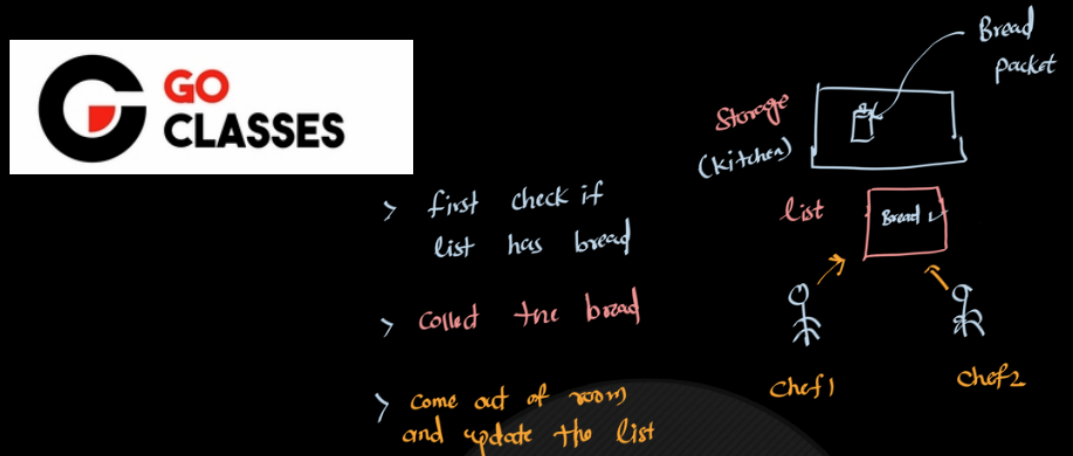
what we have seen :-



if multiple processes try to read / write in a shared memory then we should be careful.


Storage
(kitchen)
list
list of items
that this storage
room has


GO CLASSES
Bread packet
Storage (kitchen)
> first check if list has bread
> collect the bread
> come out of room and update the list
list
Bread
Chef 1
Chef 2



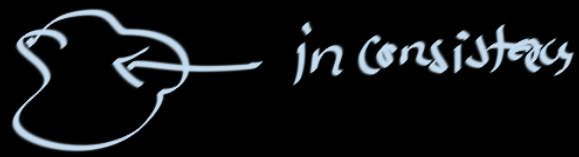
1) Chef1 checks that breads are available

2) chef1 goes to collect the bread

1) Chef2 will also check that bread is available

2) chef2 will also go to collect the bread

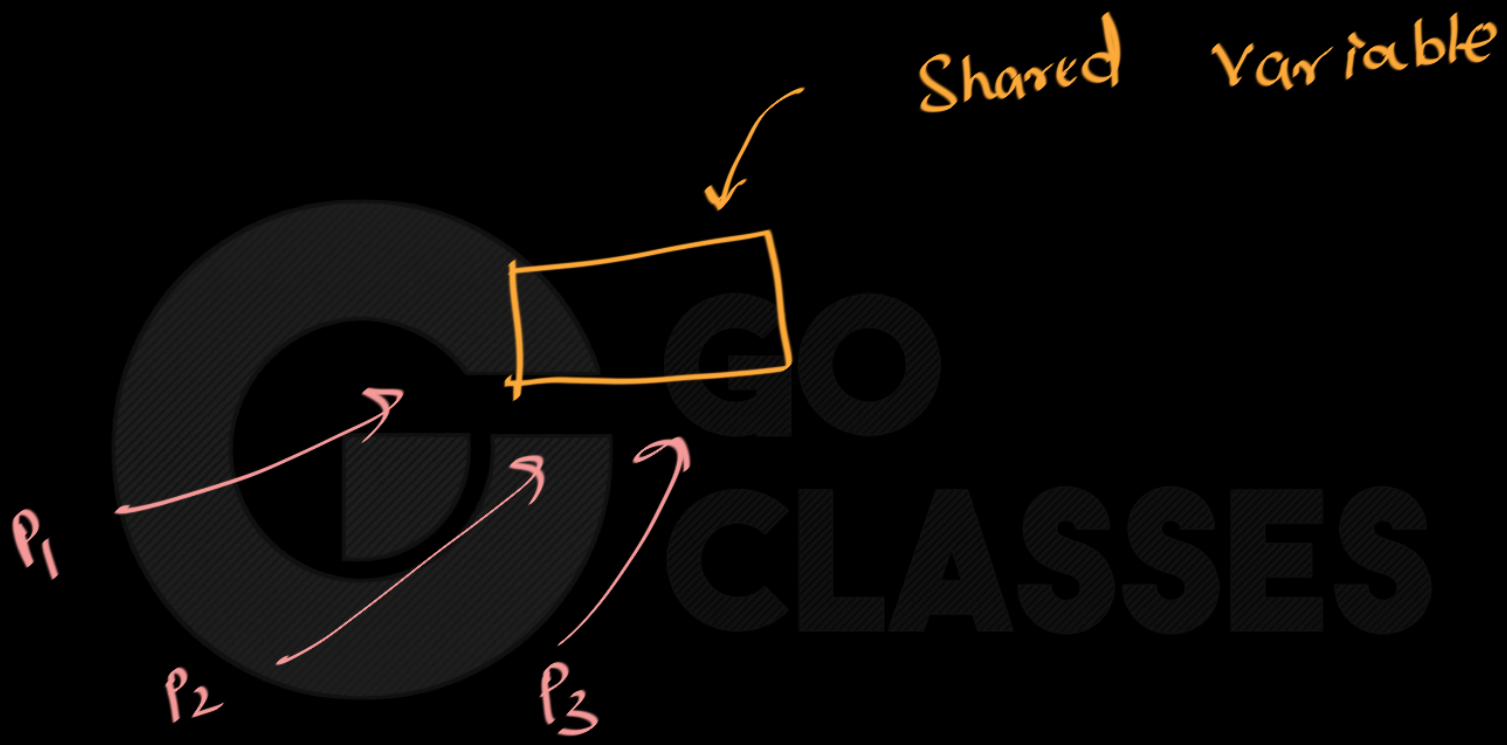
3) chef1 has collected the bread and going to update the list

3) chef2 will not be able to find bread.  inconsistency

GO CLASSES

- "The *too much milk* problem"

time	You (Me)	Your Roommate (Deepak)
3:00	Arrive home	
3:05	Look in fridge, no milk	
3:10	Leave for grocery	
3:15		Arrive home
3:20	Arrive at grocery	Look in fridge, no milk
3:25	Buy milk	Leave for grocery
3:35	Arrive home, put milk in fridge	
3:45		Buy Milk
3:50		Arrive home, put up milk
3:50		Oh no!


Shared Variable
P1
P2
P3

GO CLASSES

x: global variable

```
foo()
{
    x = x+1
}

int main() {
    T1 = Pcreate_thread(foo);
    T2 = "          "
    wait(T1, T2)
    Pf(x)
}
```

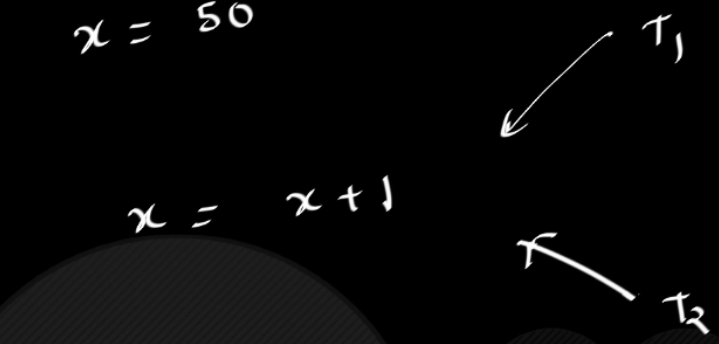
Suppose x is shared among ? threads

$x = 50$

$x = x + 1$

Once T_1 and T_2 both are done then what will be the value of x?

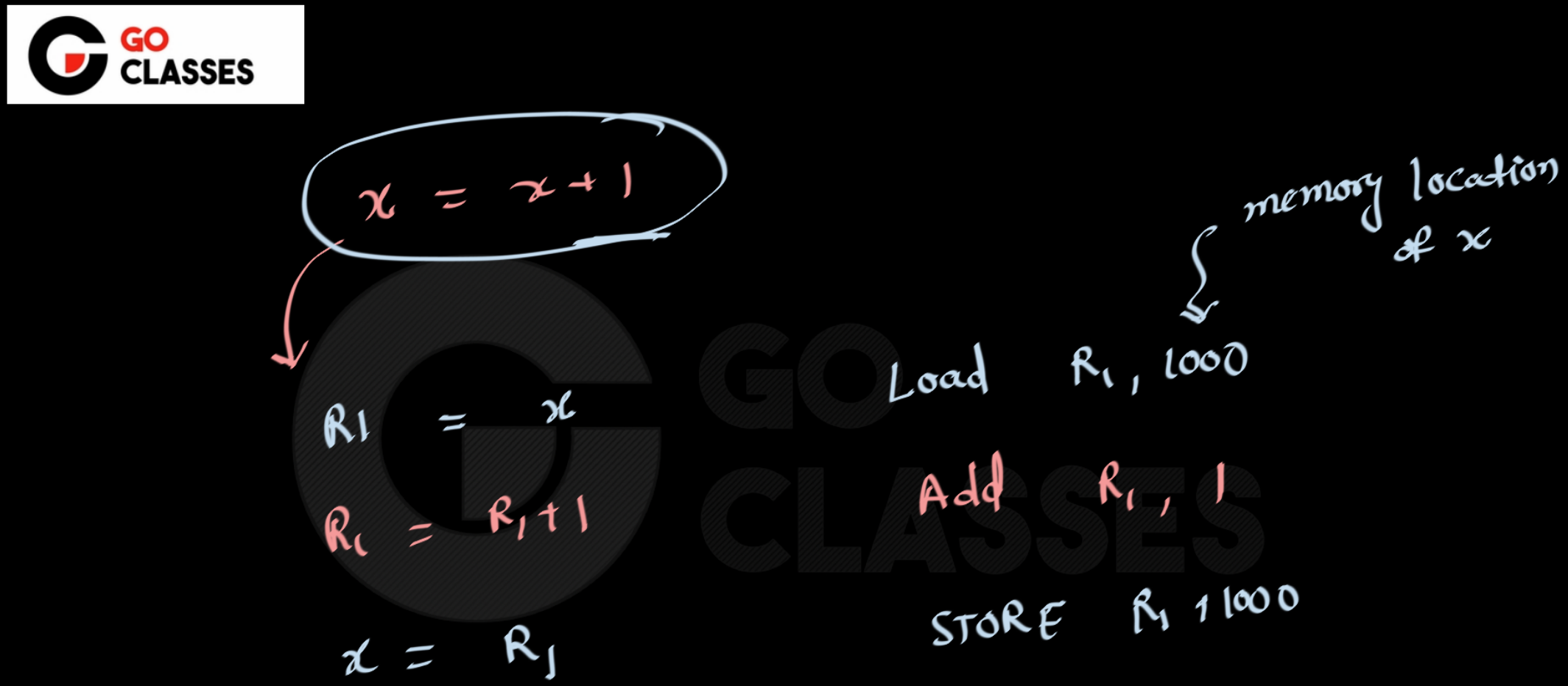
53 ← final value ← either T_1 or T_2 is first



Once T_1 and T_2 both are done then what will be the value of x?

52 ← final value ← either T_1 or T_2 is first

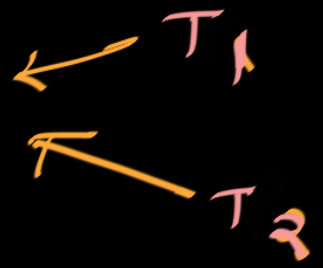
the final answer ⇒ Could be 51 or 52.

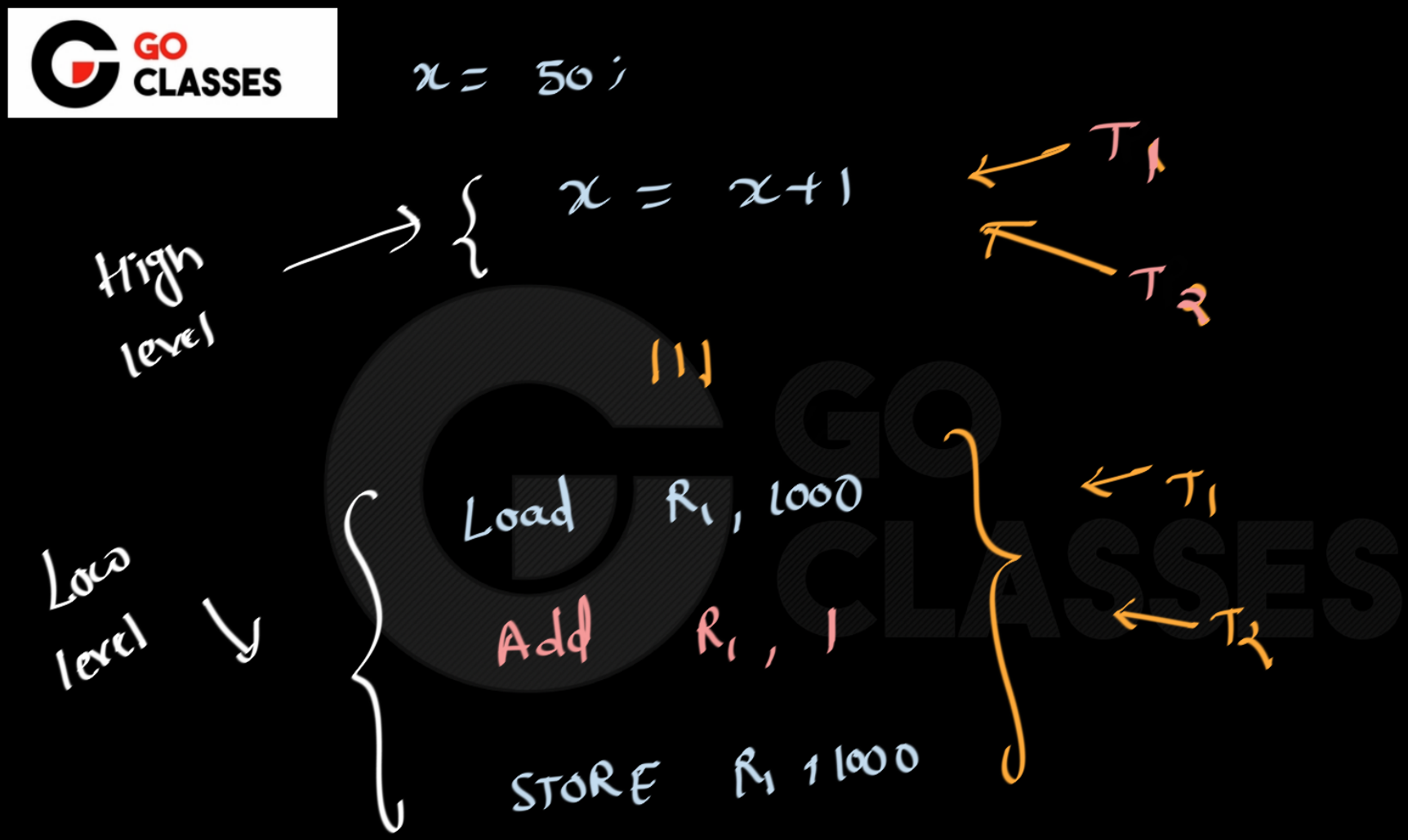

x = x + 1
R1 = x
R1 = R1 + 1
x = R1
memory location of x
Load R1, 1000
Add R1, 1
STORE R1, 1000

GO CLASSES

$x = 50\ i$

$x = x + 1$




GO CLASSES
x = 50 ;
High level
{ x = x+1
T1
T2
|||
Low level
{ Load R1, 1000
Add R1, 1
STORE R1 1000
T1
T2

GO CLASSES

```
Load   R1, 1000
Add    R1, 1
STORE  R1 1000
```

T_1

```
Load   R1, 1000
Add    R1, 1
STORE  R1 1000
```

T_2

```
Load   R2, 1000
Add    R2, 1
STORE  R2 1000
```

GO CLASSES

$x = 50$

T

1 Load R_1, 1000

2 Add R_1, 1

3 STORE R_1, 1000

Z

1 Load R_2, 1000

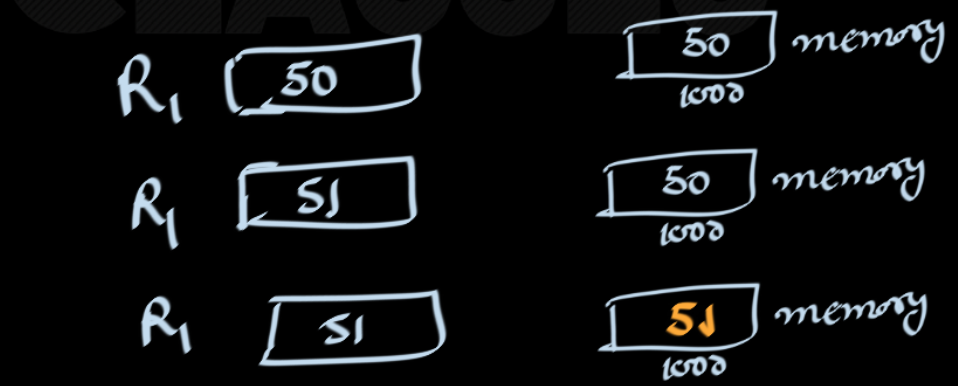
2 Add R_2, 1

3 STORE R_2, 1000

T1 : Load R_1, 1000

T2 : Add R_1, 1

T3 : STORE R_1, 1000



GO CLASSES

$x = 50$

T

1 Load R_1, 1000
2 Add R_1, 1
3 STORE R_1, 1000

Z

1 Load R_2, 1000
2 Add R_2, 1
3 STORE R_2, 1000

final value of x is **52**

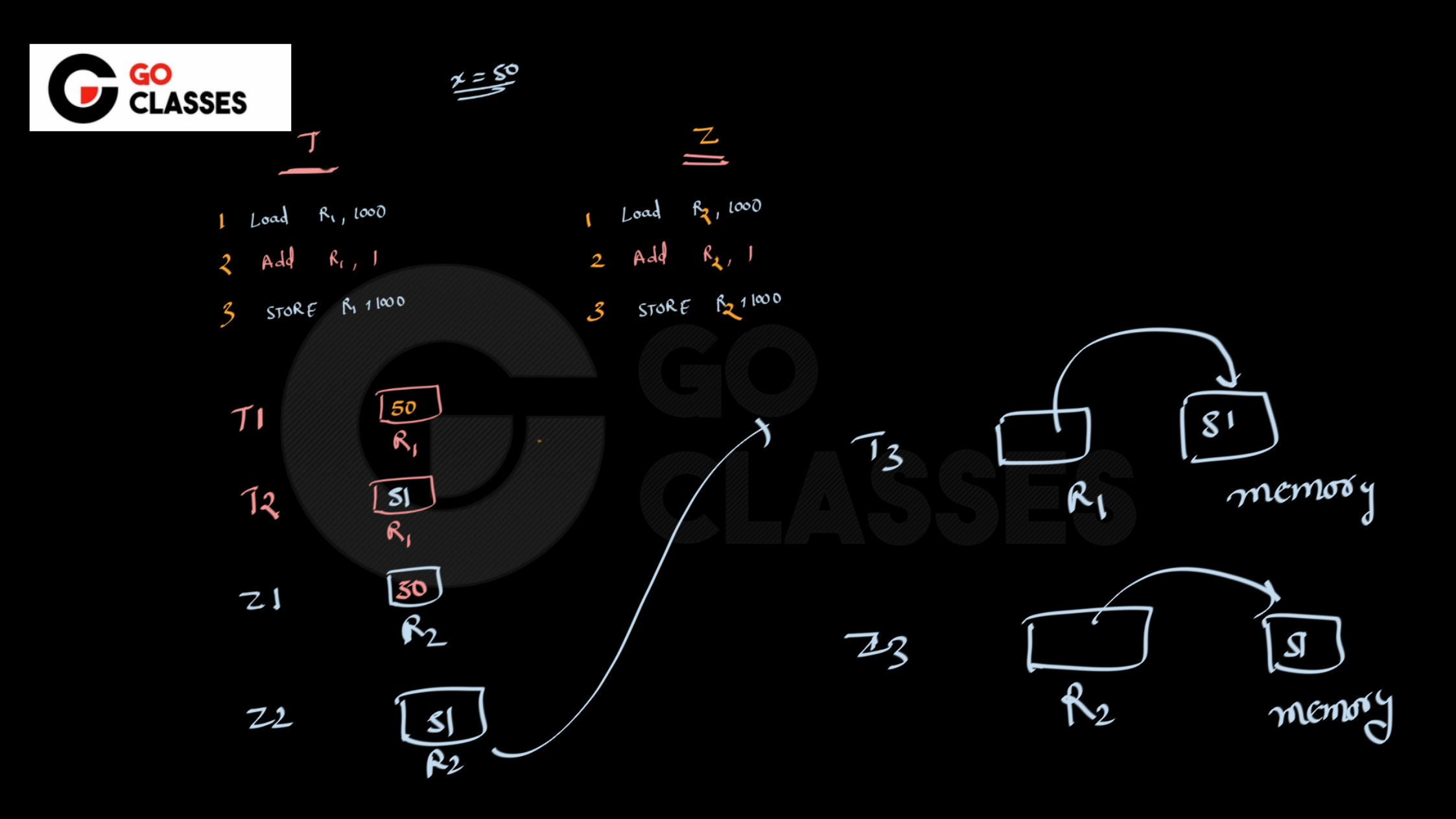
T

T1 : Load R_1, 1000 — R_1 50 — 50 memory 1000
T2 : Add R_1, 1 — R_1 51 — 50 memory 1000
T3 : STORE R_1, 1000 — R_1 51 — 51 memory 1000

Z

Z1 : Load R_2, 1000 — R_2 51 — 51 memory 1000
Z2 : Add R_2, 1 — R_2 52 — 51 memory 1000
Z3 : STORE R_2, 1000 — R_2 52 — 52 memory 1000

SEQUENTIAL EXECUTION


GO CLASSES
x = 50
T
1 Load R1, 1000
2 Add R1, 1
3 STORE R1, 1000
Z
1 Load R2, 1000
2 Add R2, 1
3 STORE R2, 1000
T1
50
R1
T2
51
R1
Z1
50
R2
Z2
51
R2
T3
R1
51
memory
Z3
R2
51
memory

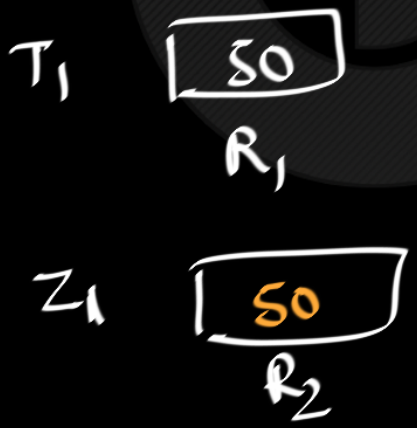
$x = 50$

T

```
1  Load   R1, 1000
2  Add    R1, 1
3  STORE  R1, 1000
```

Z

```
1  Load   R2, 1000
2  Add    R2, 1
3  STORE  R2, 1000
```



Now whatever be the situation final value of x will be 51.

Question:

There are two processor named producer and consumer.

And their code are also given below.

P1

Code for Producer

```
counter++;
```

P2

Code for Consumer

```
counter--;
```

Analyze what can happen If we run these two processes concurrently.

`counter++;`

$register_1$ = `counter`
$register_1 = register_1 + 1$
`counter` = $register_1$

`counter--;`

$register_2$ = `counter`
$register_2 = register_2 - 1$
`counter` = $register_2$

`counter++;`

$register_1 = \texttt{counter}$
$register_1 = register_1 + 1$
$\texttt{counter} = register_1$

`counter--;`

$register_2 = \texttt{counter}$
$register_2 = register_2 - 1$
$\texttt{counter} = register_2$

producer	execute	$register_1 = \texttt{counter}$	$\{register_1 = 5\}$
producer	execute	$register_1 = register_1 + 1$	$\{register_1 = 6\}$
consumer	execute	$register_2 = \texttt{counter}$	$\{register_2 = 5\}$
consumer	execute	$register_2 = register_2 - 1$	$\{register_2 = 4\}$
producer	execute	$\texttt{counter} = register_1$	$\{counter = 6\}$
consumer	execute	$\texttt{counter} = register_2$	$\{counter = 4\}$

`counter++;`

$register_1 = \texttt{counter}$
$register_1 = register_1 + 1$
$\texttt{counter} = register_1$

`counter--;`

$register_2 = \texttt{counter}$
$register_2 = register_2 - 1$
$\texttt{counter} = register_2$

producer	execute	$register_1 = \texttt{counter}$	$\{register_1 = 5\}$
producer	execute	$register_1 = register_1 + 1$	$\{register_1 = 6\}$
consumer	execute	$register_2 = \texttt{counter}$	$\{register_2 = 5\}$
consumer	execute	$register_2 = register_2 - 1$	$\{register_2 = 4\}$
producer	execute	$\texttt{counter} = register_1$	$\{counter = 6\}$
consumer	execute	$\texttt{counter} = register_2$	$\{counter = 4\}$

producer	execute	$register_1 = \texttt{counter}$	$\{register_1 = 5\}$
producer	execute	$register_1 = register_1 + 1$	$\{register_1 = 6\}$
consumer	execute	$register_2 = \texttt{counter}$	$\{register_2 = 5\}$
consumer	execute	$register_2 = register_2 - 1$	$\{register_2 = 4\}$
consumer	execute	$\texttt{counter} = register_2$	$\{counter = 4\}$
producer	execute	$\texttt{counter} = register_1$	$\{counter = 6\}$

final value

Operation	counter = counter+1;	counter = counter+1;
on CPU	reg_1 = counter; reg_1 = reg_1 + 1; counter = reg_1;	reg_2 = counter; reg_2 = reg_2 + 1; counter = reg_2;
Interleaved execution	reg_1 = counter; reg_1 = reg_1 + 1;	
		reg_2 = counter; reg_2 = reg_2 + 1;
	counter = reg_1;	
		counter = reg_2;

`counter++;`

$register_1$ = `counter`
$register_1 = register_1 + 1$
`counter` = $register_1$

`counter--;`

$register_2$ = `counter`
$register_2 = register_2 - 1$
`counter` = $register_2$

This is called Race Condition

T_0:	*producer*	execute	$register_1$ = `counter`	$\{register_1 = 5\}$
T_1:	*producer*	execute	$register_1 = register_1 + 1$	$\{register_1 = 6\}$
T_2:	*consumer*	execute	$register_2$ = `counter`	$\{register_2 = 5\}$
T_3:	*consumer*	execute	$register_2 = register_2 - 1$	$\{register_2 = 4\}$
T_4:	*producer*	execute	`counter` = $register_1$	$\{counter = 6\}$
T_5:	*consumer*	execute	`counter` = $register_2$	$\{counter = 4\}$

Race Condition:

A situation where several processes access and manipulate the same data concurrently and the outcome of execution depends on the particular order in which the access takes place, is called a Race Condition.

The Critical-Section Problem

- The important feature of the system is that, when one process is executing in its critical section, no other process is allowed to execute in its critical section.

area which is having shared variable

CS for P_1

$x = x + 1$

CS for P_2

$x = x - 1$

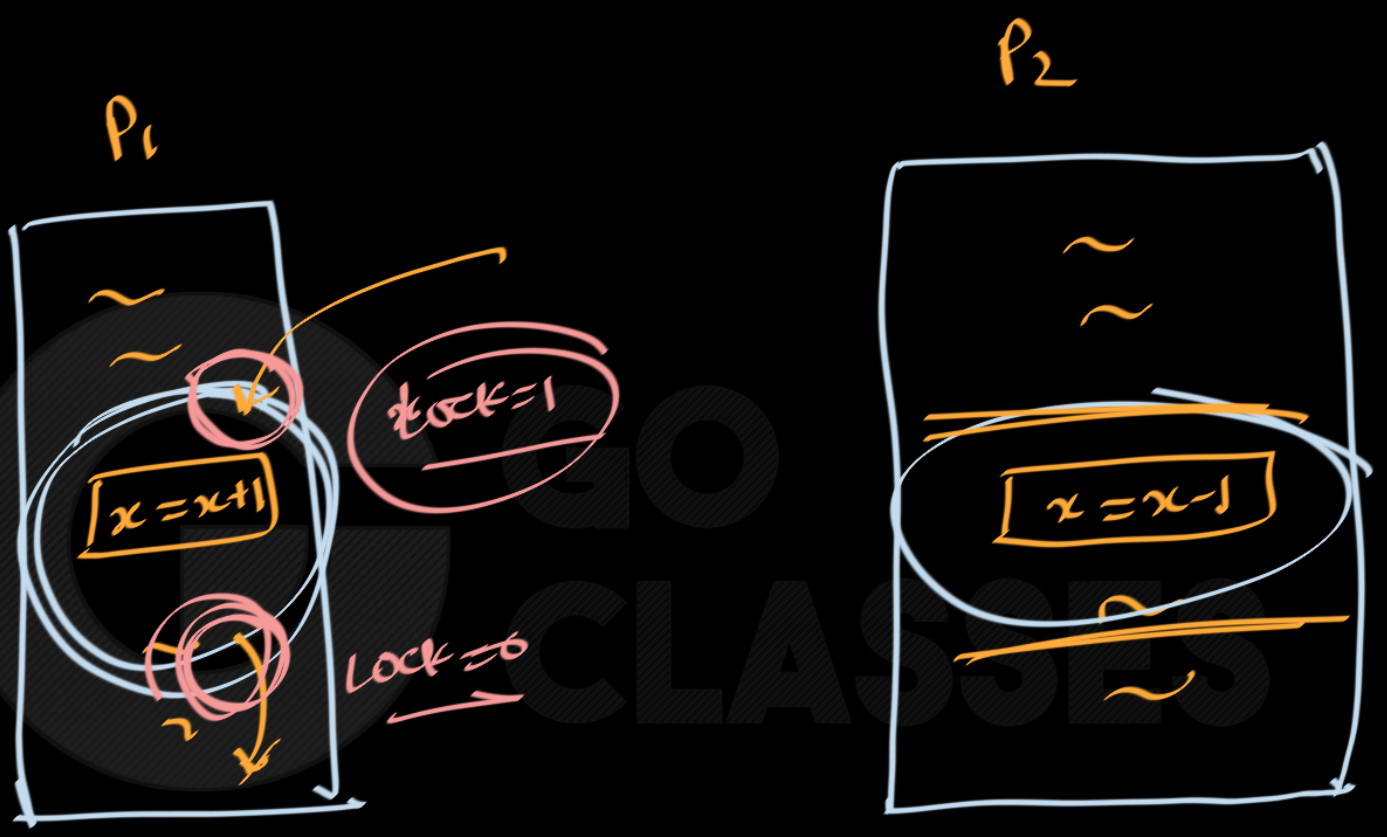
GO CLASSES

Very naive soln :

$P_1 \longrightarrow P_2$

$P_2 \longrightarrow P_1$

} Sequential execution (not a good idea for preemptive scheduling algorithms)


P1
P2
lock=1
x=x+1
lock=0
x=x-1

Washroom

≡ critical section

The Critical-Section Problem

Informally, a critical section is a code segment that accesses shared variables and has to be executed as an atomic action.

The critical section problem refers to the problem of how to ensure that at most one process is executing its critical section at a given time.

Important: Critical sections in different threads are not necessarily the same code segment!

$x = x+1$ $x = x-1$

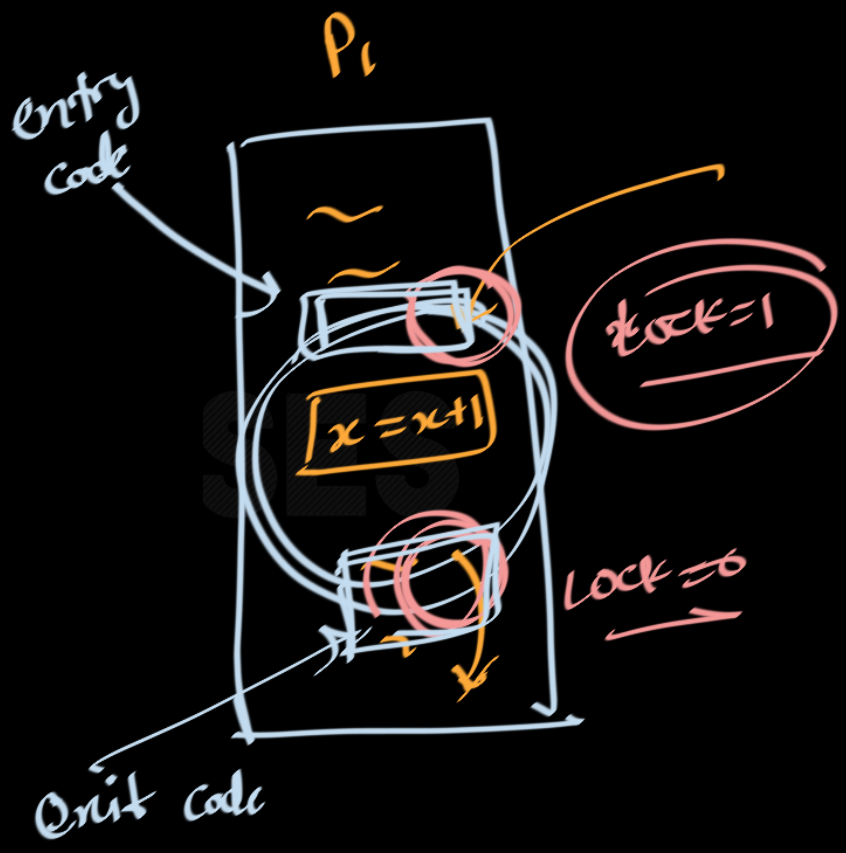
GO CLASSES

[CS ⇒ area of code which has Shared variable]

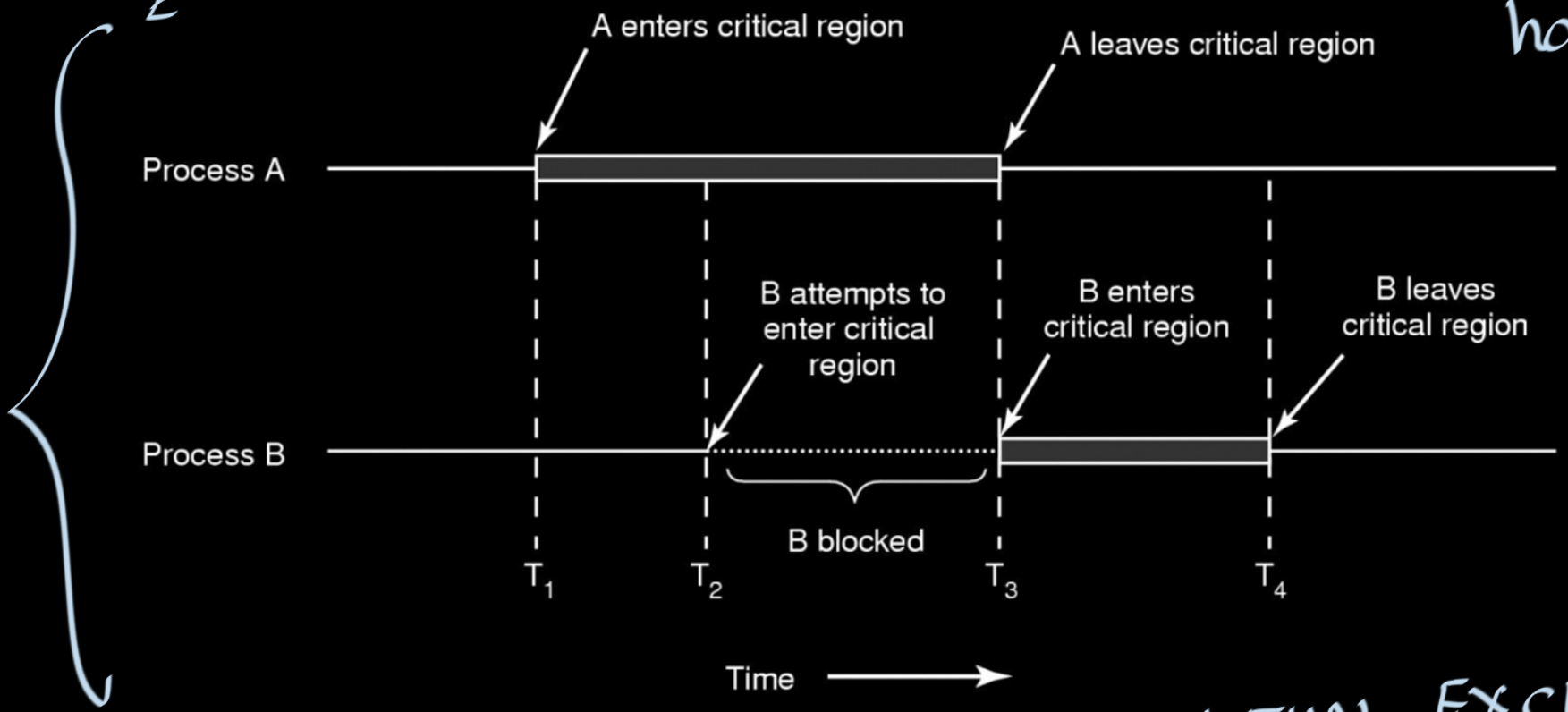
x = x+1

General structure of process P_i

```
do {
        entry section
                critical section
        exit section
                remainder section
} while (true);
```



this is how we want interleaving to happen.



MUTUAL EXCLUSION

The Critical-Section Solution