③ Strict → First update → Commit or abort → Read and write

$w_2(x) \rightarrow c_2 \rightarrow r_1(x)$ / $w_1(x) \rightarrow c$

Strict

Read and write after commit or abort

$S_a: r_1(y), w_1(y), r_1(x), \ldots$

$S_2: r_2(x), w_2(x), r_2(y), w_2(y), a_2, r_1(x), w_1(x), c_1$

↳ old value → Before Image → BFIM

$S_n: r_1(y), r_1(x), w_1(x), w_1(y), w_2(x), c_1, c$

① Recoverable

First write → First commit

$w_1(x) \rightarrow c_1$

Recoverable

① Recoverable

First update → First commit

$w_2(x) \rightarrow c_2$

Recoverable

② Cascaders

Read only after commit

$w_1(x) \rightarrow w_2(x) \rightarrow c_2$

Visit Cascaders

② Cascaders

First update → Commit → Read

$w_1(x) \rightarrow c_1 \rightarrow r_1(x)$

Cascaders

Wait die

T_2 holds X

T_1 holds Y

① $T_3(T_1) < T_3(T_2)$

T_1 is allowed to wait

② Otherwise ($T_1(T_1) \geq T_1(T_2)$)

- T_1 is aborted
- T_1 is killed by T_2
- Start later with the same time stamp

$T_3(T_1) = 8$

$T_1(T_2) = 5$

$T_1(T_3) = 11$

$T_3(T_1) < T_1(T_2) < T_3(T_2)$

$S_a: r_1(y); w_1(y); r_1(x); w_1(x); C_1; r_2(x); w_2(x); C_2$

Read only after commit

Recoverable

First update → First commit

$w_1(x) \rightarrow C_1$

Cascading

Y_{es} it's Recoverable

T_1	T_2
read(y);	read(x);
y = y - 100;	x = x + 5;
write(y);	write(x);
read(x);	Commit
x = x - 100;	
write(x);	
Commit	

① From two different transaction

② Access the same item

③ at least one of the operation write

Conflict pairs

$w_1(x)$ $w_2(x)$

Cascadeless :- Read only after commit

$w_1(x) \rightarrow C_1 \rightarrow R_2(x)$

Yes, Cascadeless

S_a: $r_1(y), w_1(y), r_1(x), w_1(x), C_1$; $r_2(x), w_2(x), C_2$

: $r_2(x), \underline{w_2(x)}, r_1(x), w_1(x), C_2, C_1$

S_{11}: $r_1(y), r_1(x), \underline{w_1(x)}, w_1(y), w_2(x), C_1, C_2$

Recoverable

① Recoverable

update ⟶ First commit

First write ⟶ First Commit

$w_1(x) \longrightarrow C_2$

$w_1(x) \longrightarrow C_1$

it's Recoverable

Recoverable

cadless

② Cascadless

only after commit

Read only after commit

$x) \rightarrow r_1(x) \rightarrow C$

$w_1(x) \rightarrow w_2(x) \rightarrow C_2$

it's Not Cascadless

Yes it's Cascadless

S_a: $r_1(x); w_1(x); r_2(x); w_1(x); C_1; r_2(x)$

S_n: $r_2(x); w_1(x); r_1(x); w_1(x); C_2; C_1$

① Recoverable

First update → First commit

$w_1(x) \rightarrow C_2$

Yes it's Recoverable

② Cascadless:
Read only after commit

$w_2(x) \rightarrow r_1(x) \rightarrow C$

No it's Not Cascadless

① From the diffe
② Access the
item

③ at last
operation

Cascadless:- Read only after

$w_7(x) \rightarrow C_7 \rightarrow R_2(x)$

Yes Cascadless

T

read(x)
x = x + T;
write(x)
Commit

$S_a: r_1(x), w_1(y), r_1(x), w_1(x), C_1; r_2(x), w_2(x), C_2$

c

read only after commit

Recoverable

first update → first commit

d Cascading

T_1

T_2

@ From two different transaction
@ Access the same item
③ at least one of the operation write

Conflict	pairs
$r_2(x)$	$w_1(x)$
$w_1(x)$	$r_2(x)$
$w_1(x)$	$w_2(x)$

T_2

read(x)
x = x + 10;
write(x)

T_1

read(y),
y = y - 50;
write(y),
read(x)
x = x - 10;
write(x);
Commit

read(x)
x = x + 10;
write(x)
Commit

T_1 T_2

read(x),
x = x + 5;
write(x)

S_a : $r_1(y)$, $w_1(y)$, $r_2(x)$, $w_1(x)$, C_1 ; r_2

c

b only after commit

Recoverable

first update → first commit

coding

Conflict pairs

$r_1(x)$	$w_2(x)$
$w_1(x)$	$r_2(x)$
$w_1(x)$	$w_2(x)$

T_1

read(y),
y = y - 100;
write(y);
read(x);
x = x - 100;
write(x);
Commit

T_2

read(x);
x = x + 1;
write(x);
Commit

① From two different transaction
② Access the same item
③ at least one of the operation write

T_0

③ Strict-

For update → Commit or abort → Read and write

$w_2(x) \rightarrow c_2 \rightarrow r_1(x)$ / $w_1(x) \rightarrow c_1$

Strict

Read and write after commit or abort

$S_2: r_2(x), w_2(x), r_2(y), w_2(y) c_2, r_1(x), w_1(x), c_1$

① Recoverable

First update → First commit

$w_2(x) \rightarrow c_2$

Recoverable

② Cascadless

First update → Commit → Read

$w_1(x) \rightarrow c_1 \rightarrow r_1(x)$

Cascadless

$S_a: r_1(x), w_1(x), r_1(x), w_1(x), c_1;$

$S_b: r_1(y), r_1(x), w_1(x), w_1(y), w_2(x), c_1$

① Recoverable

First write → First commit

$w_1(x) \rightarrow c_1$

Recoverable

② Cascadless

Read only after commit

$w_1(x) \rightarrow w_2(x) \rightarrow c_2$

Yes it is Cascadless

Read only after commit

Recoverable

First update → First commit

Cascadless

⇓

to avoid Cascading

Rollback

$\{ r_1(x), w_1(x), r_2(x), w_2(x), C_2, C_1 \}$

First update → First commit

$w_2(x) \rightarrow C_2$

Non Recoverable

T_2

Read(x)

x=x*15,

write(x)

Fail

T_0

read(x)

x=x+10,

write(x)

commit T_0

Read → R

Write → W

Commit → C

Abort → A

T_1

1 — Read(x);
2 — x=x-100;
3 — write(x);

T_2

$\sum_i r_1(X), w_1(X), r_2(X), w_2(X); r_1(Y), w_1(Y), C_1, C_2$

Read(x);
x=x-5;
write(x);
← 4, 5, 6

Read(y)
y=y*1.5;
write(y);
Commit
← 7, 8, 9, 10

Commit 11

$S: r_1(x), w_1(x), r_2(x), w_2(x), r_1(y), w_1(y), C_1, C_2$

Read → R

Write → W

Commit → C

Abort → A

T_1

1 ——— Read(x);

2 ——— x = x + 5;

3 ——— write(x);

T_2

4 ——— read(x);

5 ——— x = x - 5;

6 ——— write(x);

7 ——— Read(y)

8 ——— y = y × 1.5;

9 ——— write(y);

Commit

Commit

S_4: $r_2(x), r_2(y), r_3(x), w_2(x), w_3(x), c_2, c_3$

① Recoverable:-

S_1: $r_1(x), r_1(y), w_1(y), w_1(x), r_2(x), w_2(x), c_1, c_2$

First write → First commit

$w_1(x) \rightarrow c_1$

Recoverable

S_2: $r_2(x), w_2(x), r_1(x), w_1(x), c_1, c_2$;

First update → First commit

$w_2(x) \rightarrow c_1$

Non Recoverable

T_5	T_2
	read (x),
	read (y),
	x := x + y,
read (x),	
x := x + 5,	
	write (x),
write (x),	
	Commit
Commit	

① Write the schedule
② Find conflict pair
③ check if this should Recoverable

First update → First commit

$w_2(x) \rightarrow c_2$

Recoverable

$r_2(x)$	$w_5(x)$
$r_5(x)$	$w_2(x)$
$w_2(x)$	$w_5(x)$

Transfer 100 JD from account A to Account B

T_1 (Transaction)

```
Begin
Read(A),
A := A - 100,
write(A),
Read(B),
B := B + 100,
write(B)
end,
```

A = 500
B = 700

read(A)
write(A)

Read(A),
A := A - 200,
write(A),

Transfer 50 JD from account A to Account B

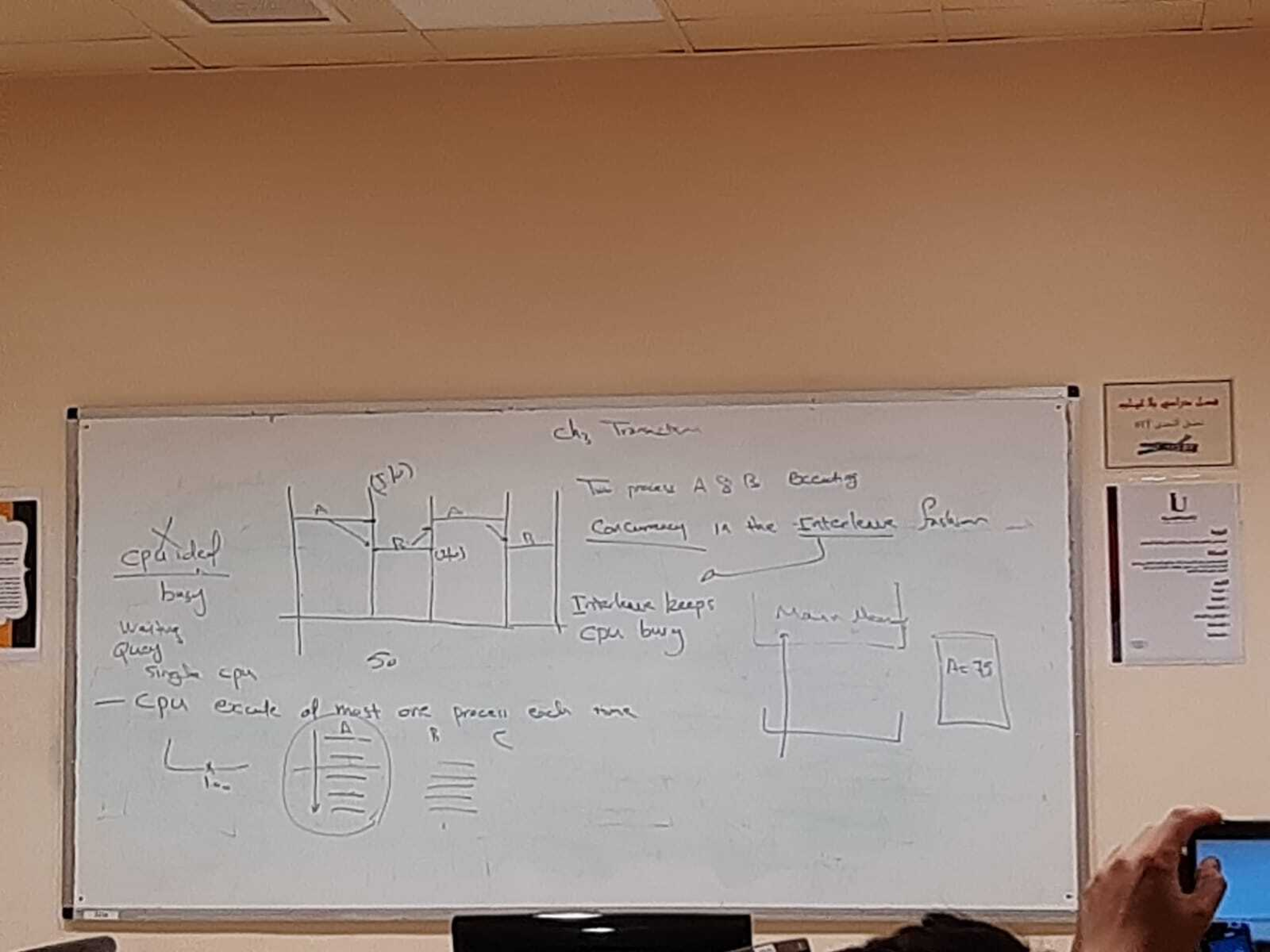
T_1 (Transaction)

Begin
Read(A),
A := A - 50,
write(A)
Read(B),
B := B + 50,
write(B)
end.

A = 500
B = 700

read(A)
write(A)

Read(A),
A := A - 20,
write(A),



ch3 Transaction
Two process A & B Executing
Concurrency in the Interleave fashion
Interleave keeps cpu busy
cpu idel
busy
Waiting Query
Single cpu
— cpu execute at most one process each time
A
B
C
50
Main Mem
A=75

100,000

dept

ID	d_Name	d_location
	IT	Amman
	Sales	Amman
	H.R	Amman

emp

one table two tables

the same type varchar(15)

P.K emp_ID	EName	Address	Sal	Mobile	deptID
1000	Ali	Amman	700	0795555	1
2000	Mai	Madaba	900	0786666	2
3000	Mona	Amman	600	0771234	3
4000	Hani	Madaba	1000	0785678	1
5000	Sami	Amman	700	0793232	2

Select emp_ID, EName, dname
from emp, dept
where
emp.dept_ID = dept.dID
AND
D_Name like 'IT';

Need All Information of employee who Number is 1000,

→ Select * from emp
Read only where emp_ID = 1000;

Ms. Access

Kuwait
Jordan

10,000

dept

d_ID	d_Name	d_location
1	IT	Amman
2	Sale	Amman
3	H.R	Amman

emp

emp_ID	EName	Address	Sal	Mobile	DeptID
1000	Ali	Amman	700	079 5555	1
2000	Mai	Madaba	900	07 66666	2
3000	Maha	Amman	600	077 1234	3
4000	Hani	Madaba	1000	078 5678	1
5000	Sami	Amman	700	079 323232	2

Need All IT department Name, EName, emp_ID, d_Name,.

Need All Information of employee who Number is 1000,

→ Select * from emp

PL/Sql

Read only where emp_ID = 1000;

Ms. Acces

Kuwait

Jordan

Select emp_ID, EName, dname from emp, dept where emp.dept_ID = dept.d_ID AND D_Name like 'IT';

T_1	T_2
read(x)	
	read(x)
read(y)	
	read(y)
y = y + x	
	z = y + x
write(y)	
	write(z)

$T_1 \xrightarrow[\text{write}(y)]{\text{issue}}$ R-timestamp(y), W-timestamp(y)

$TS(T_1) > W\text{-timestamp}(y)$

$1 > 0$

$TS(T_1) < R\text{-timestamp}(y)$

$1 < 2$

- write(y) operation is Rejected
- Roll back for T_1
- assign a new timestamp for T_1
- Restart

$TS(T_1) = 1$

$TS(T_2) = 2$

	X	Y	Z	TW
R-timestamp	~~0~~ 2	~~0~~ 2	0	T_1 to T_1T_2
W-timestamp	0	0	0	

$TS(T_1) = 5:00$

Timestamp protocol / Basic

Suppose T_i issues a read(Q)

if $TS(T_i) < W\text{-timestamp}(Q)$

- read operation is rejected
- Rolled back for T_i
- Assign a new timestamp for T_i
- restart

T_i

read(x)
y = x + low
write(y)

Abort

read(y) < write timestamp(y)

Reject

T_i

$TS(T_i) = 8:00$

T_i issues a read(Q) → Q ← T_5 W-timestamp = 9

$TS(T_5) = 9:00$

- Rejected read
- Rolled back
- Assign new timestamp to T_i
- restart

T_i reads a Q → R-timestamp, W-timestamp

Conflict

① from two different transaction
② Access the same item
③ at least one operation is write.

T_i read(Q) → Q ← T_j W-timestamp (Y:00)

$TS(T_i) = 9:0$

$9 < 11$

Suppose T_i issues a read(Q)

① if $TS(T_i) < W\text{-timestamp}(Q)$

- Read operation is rejected
- Rolled back for T_i
- Assign a new time stamp for T_i
- restart

② if $TS(T_i) \geq W\text{-time stamp}$

* Read operation is executed
* R-timeStamp(Q) is set to the time stamp Read(T_i)
(update R-timestamp to T_i)

Suppose T_i issues a write (Q) — conflict: R-timestamp, W-time stamp

if $TS(T_i) < R\text{-timestamp}(Q)$

- write operation is Rejected
- Rollback for T_i
- Assign a new time stamp to T_i
- Restart

if $T_{os}(T_i) < W\text{-timestamp}$

- Rejected
- Rolled back
- assign a new time stamp
- restart

otherwise ($T_s(T_i) \geq R\text{-timestamp}$, $T_s(T_i) \geq W\text{-timestamp}$)

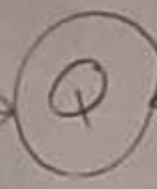
operation is executed

W-timestamp is updated to t_s of $T_s(T_i)$

$T_s(T_5) = 9.00$

$T_s(T_i) = 9:00$

T_i issues a write (Q)



R-timestamp(Q) = 8a
W-timestamp(Q) = 8a

write operation is executed
W-timestamp(Q) = 9.0-

T_1	T_2
Read(x)	
	read(x)
read(y)	read(y)
y := y + x	z := y + x
write(y)	write(z)

$T_s(T_1) = 100$

$T_s(T_2) = 200$

	X	Y	Z	T_{ID}
R-timestamp	~~0~~ 2	~~0~~ 2	0	T_1 T_2
W-timestamp	0	0	0	

T_1

Read(y)

$T_s(T_1) > W\text{-timestamp}(y)$

$2 > 0$

* Read(y) operation for T_2 is executed

- R-timestamp(y) = 2

1

$T_s(T_2) > W\text{-timestamp}(x)$

$2 > 0$

* Read operation for T_2 is executed

R-timestamp(x) = 2

(1) R-timestamp = 0, W-timestamp = 0

Timestamp protocol (Basic

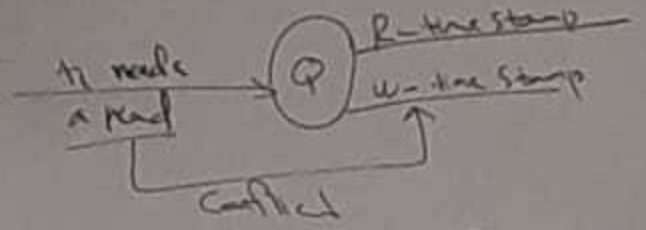
Suppose T_i issues a read(Q)

① if $TS(T_i) < W\text{-timestamp}(Q)$

- read operation is rejected
- Rolled back for T_i
- Assign a new time stamp for T_i
- restart

② if $TS(T_i) \geq W\text{-timestamp}$

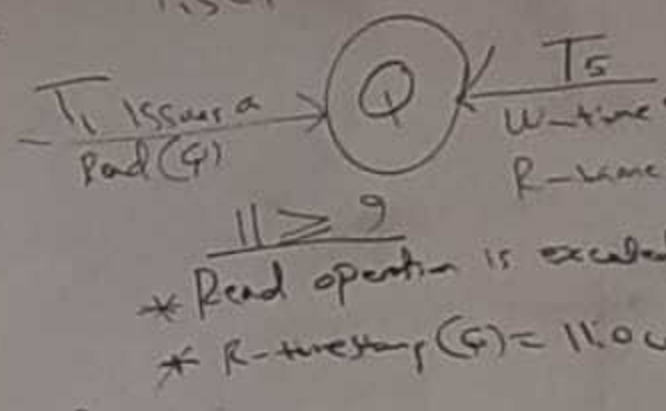
* Read operation is executed
* R-timestamp(Q) is set
  ≥ the time stamp Read $T_{(i)}$
  (update R-timestamp to be (T_i) timestamp



① from two different transaction
② Access the same item
③ at least one operation is write.

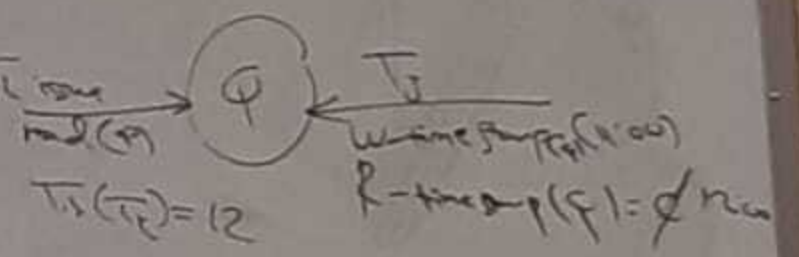
$TS(T_5) = 9:00$

$TS(T_i) = 11:00$



$11 \geq 9$

* Read operation is executed
* R-timestamp(Q) = 11:00



legend

T_1, T_2, T_5, T_{10}.

	X	y	z
Producer Sleep	0	0	0
Consumer Sleep	0	0	0

T_1 T_2 T_5

read(x)

x = x + 1;

read(x);

x = x + 1

write(x)

read(y);

y = y * 5;

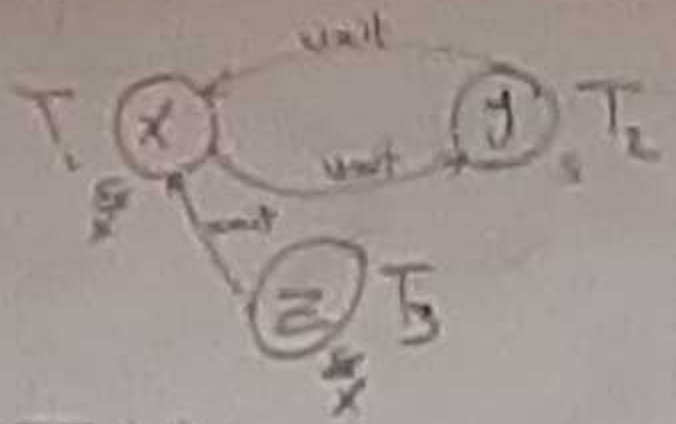
write(y);

read(x);

read(z);

z = z + x;

write(z);


T1
T2
T3
x
y
z

Worst case

T_2 locked(x)

T_1 wait to lock(x)

① $T_2(T_1) < T_1(T_2)$

T_1 is allowed to wait

② otherwise ($T_2(T_2) \geq T_1(T_2)$)

T_1 is aborted

Worst case

T_2 locked(x)

T_1 wait to lock(x)

① $T_2(T_1) < T_1(T_2)$

T_1 is allowed to wait

② otherwise ($T_2(T_2) \geq T_1(T_2)$)

T_1 is aborted

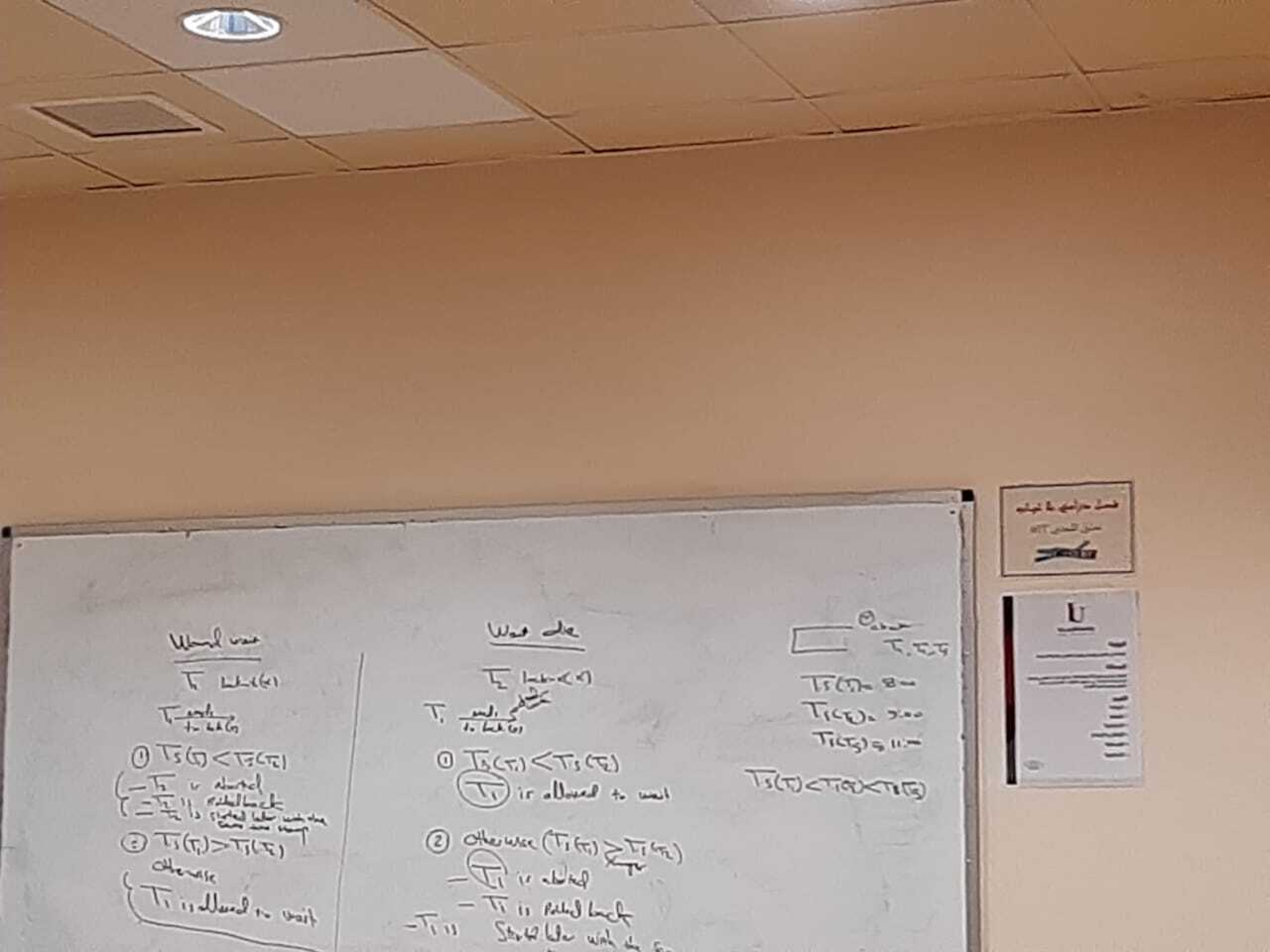
T_1, T_2, T_3

$T_3(T_1) = 8$

$T_1(T_2) = 9$

$T_1(T_3) = 11$

$T_3(T_1) < T_1(T_2) < T_2(T_3)$


Wait die
① $T_S(T_1) < T_S(T_2)$
T_1 is allowed to wait
② Otherwise $(T_S(T_1) \geq T_S(T_2))$
— T_1 is aborted
— T_1 is rolled back
— T_1 is started later with the
① $T_S(T_1) < T_S(T_2)$
— T_2 is aborted
— T_2 is rolled back
② $T_S(T_1) > T_S(T_2)$
Otherwise
T_1 is allowed to wait
$T_S(T_1) < T_S(T_2) < T_S(T_3)$