CS3223 Tutorial 9

Gary Lim

Chapter Review

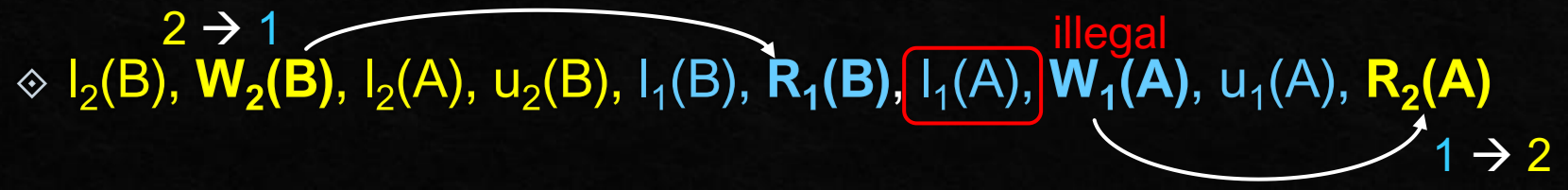
- 2PL
- Strict 2PL
- Lock granularity

2 Phase Locking

- Across the entire span of the transaction, the number of locks held by a transaction first increases, then it decreases
  - NOT ALLOWED to acquire new locks again after it has released some
  - At the highest point, it must have all the locks (of the correct type, on the correct objects) to accomplish its actions

2 Phase Locking

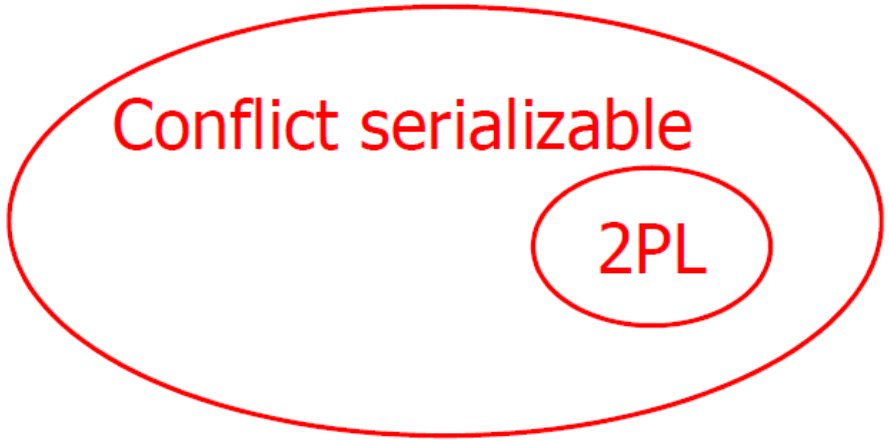
- **2PL schedules** (where each transaction acquires first, then releases their locks) **are Conflict Serializable** (2PL ➔ Conflict Serializable schedule)
  - Intuition:
    - T1 must have exclusive access to object A before it can write to it.
    - T2 must wait for T1 to release X-lock(A) before it can access it. **$W_1(A)$**, $u_1(A)$, $l_2(A)$, **$R_2(A)$**
    - If T2 comes before T1 due to a pair of conflicting actions on some other object B, e.g. **$W_2(B)$**, $u_2(B)$, $l_1(B)$, **$R_1(B)$**, then a 2PL schedule **would not** have been possible
      - T1 would not have been able to acquire X-lock(A) because T2 has acquired X-lock(B) and S-lock(A) and has not released S-lock(A)
- Non-conflict serializable schedule is not 2PL



2 Phase Locking

2PL → conflict serializable

Not conflict serializable → Not 2PL



Strict 2PL

- Same as 2PL with the addition that each Xact must hold on to locks until **Xact commits** or **aborts**
- **Strict 2PL schedules** are **strict** and **conflict serializable**
  - $R_1(X)$, $R_1(Y)$, $R_2(Y)$, $W_1(Z)$, c1, $W_3(X)$, $W_3(Z)$, c3, $R_2(X)$, $W_2(Z)$, c2
- Recall: A schedule S is a **strict** schedule if for every $W_i(O)$ in S, O is **not read** or **written** by another Xact until T_i either **aborts** or **commits**
  - If $W_1(Z) \rightarrow W_3(Z)$, then $W_1(Z)$, c1 … $W_3(Z)$
    - T2 only writes to Z after T1 commits $W_1(Z)$
  - If $W_3(X) \rightarrow R_2(X)$, then $W_3(X)$, c3 … $R_2(X)$
    - T2 only reads X after T3 commits $W_3(X)$

T1	T2	T3
$R_1(X)$		
$R_1(Y)$		
	$R_2(Y)$	
$W_1(Z)$ Commit		
		$W_3(X)$
		$W_3(Z)$ Commit
	$R_2(X)$	
	$W_2(Z)$ Commit	

Strict 2PL

- Same as 2PL with the addition that each Xact must hold on to locks until **Xact commits** or **aborts**
- **Strict 2PL schedules** are **strict** and **conflict serializable**
  - $R_1(X)$, $R_1(Y)$, $R_2(Y)$, $W_1(Z)$, c1, $W_3(X)$, $W_3(Z)$, c3, $R_2(X)$, $W_2(Z)$, c2
- Recall: A schedule S is a **strict** schedule if for every $W_i(O)$ in S, O is **not read** or **written** by another Xact until T_i either **aborts** or **commits**
  - If $W_1(Z) \rightarrow W_3(Z)$, then $W_1(Z)$, c1 … $W_3(Z)$
    - T2 only writes to Z after T1 commits $W_1(Z)$
  - If $W_3(X) \rightarrow R_2(X)$, then $W_3(X)$, c3 … $R_2(X)$
    - T2 only reads X after T3 commits $W_3(X)$

T1	T2	T3
I-$S_1(X)$ $R_1(X)$ **I-$S_1(Y)$** $R_1(Y)$		
	I-$S_2(Y)$ $R_2(Y)$	
I-$X_1(Z)$ $W_1(Z)$ Commit **$U_1(X)$** **$U_1(Y)$** **$U_1(Z)$**		
		I-$X_3(X)$ $W_3(X)$ **I-$X_3(Z)$** $W_3(Z)$ Commit **$U_3(X)$** **$U_3(Z)$**
	I-$S_2(X)$ $R_2(X)$ **I-$X_2(Z)$** $W_2(Z)$ Commit **$U_2(X)$** **$U_2(Y)$** **$U_2(Z)$**	

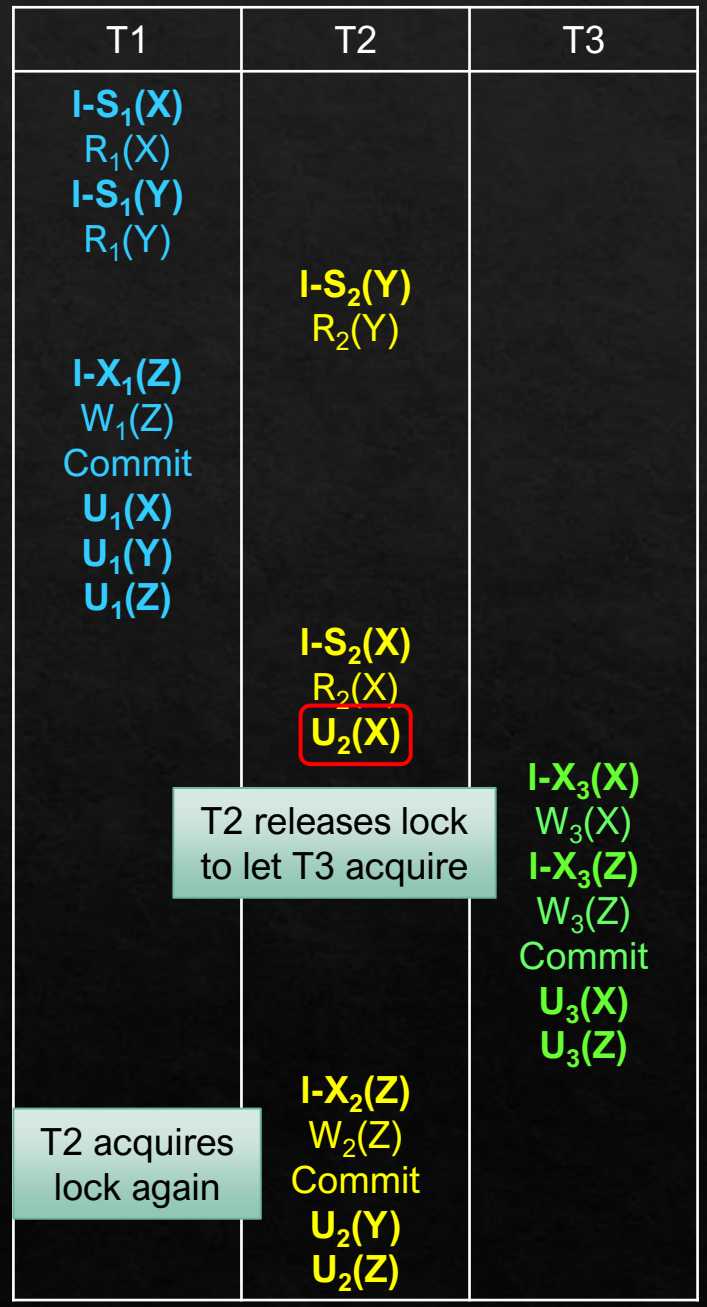
Strict 2PL

- Same as 2PL with the addition that each Xact must hold on to locks until **Xact commits** or **aborts**
- This is NOT 2PL

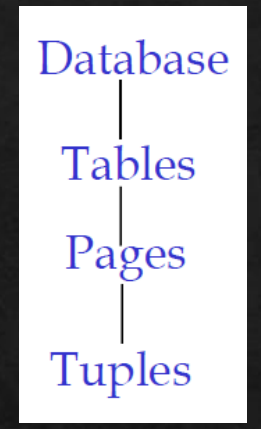
T1	T2	T3
I-S$_1$(X) R$_1$(X) **I-S$_1$(Y)** R$_1$(Y)		
	I-S$_2$(Y) R$_2$(Y)	
I-X$_1$(Z) W$_1$(Z) Commit **U$_1$(X)** **U$_1$(Y)** **U$_1$(Z)**		
	I-S$_2$(X) R$_2$(X) **U$_2$(X)**	
		I-X$_3$(X) W$_3$(X) **I-X$_3$(Z)** W$_3$(Z) Commit **U$_3$(X)** **U$_3$(Z)**
	I-X$_2$(Z) W$_2$(Z) Commit **U$_2$(Y)** **U$_2$(Z)**	

Strict 2PL

- Same as 2PL with the addition that each Xact must hold on to locks until **Xact commits** or **aborts**
- This is NOT 2PL



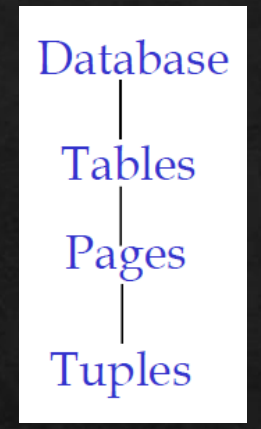
Locking Rules



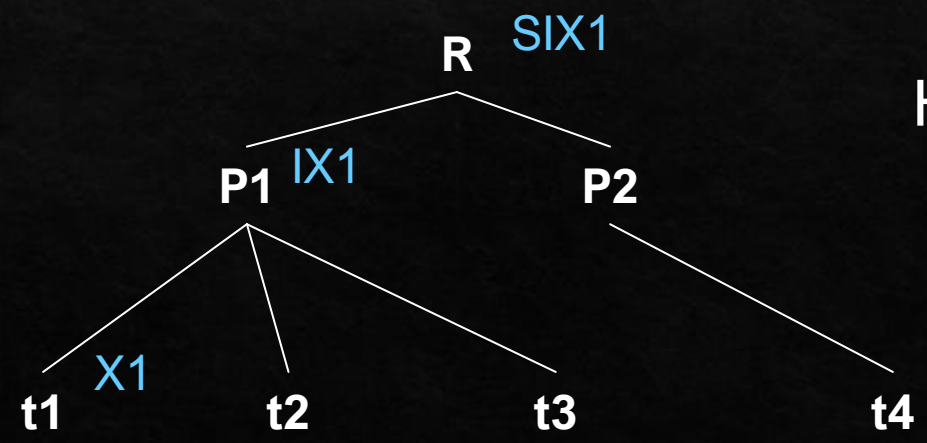
- IS – intent to read at a finer granularity
- IX – intent to write at a finer granularity
- SIX – will read this object + intent to write at a finer granularity
  - Stricter than S and IX, but less strict than X

Holder \ Requestor	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Locking Rules



- IS – intent to read at a finer granularity
- IX – intent to write at a finer granularity
- SIX – will read this object + intent to write at a finer granularity
  - Stricter than S and IX, but less strict than X

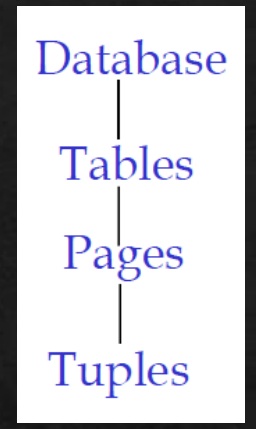


Holder

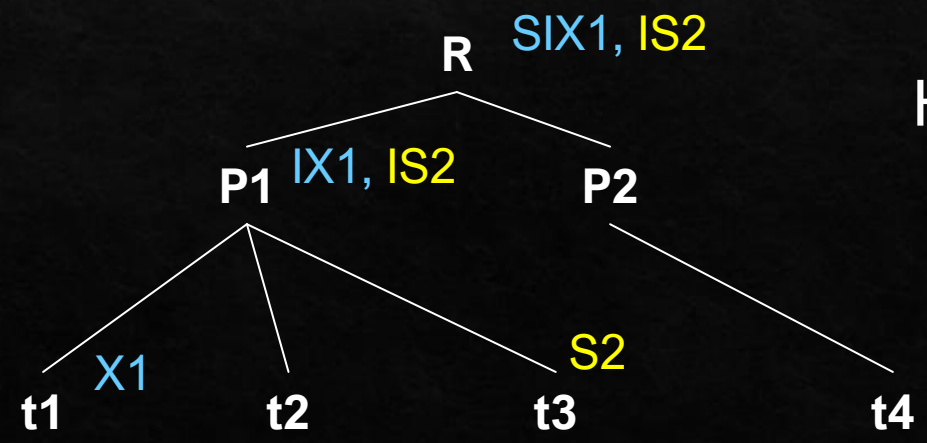
Requestor

	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Locking Rules



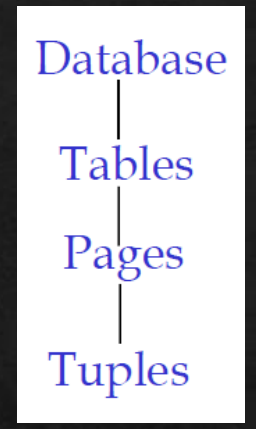
- IS – intent to read at a finer granularity
- IX – intent to write at a finer granularity
- SIX – will read this object + intent to write at a finer granularity
  - Stricter than S and IX, but less strict than X



Holder \ Requestor

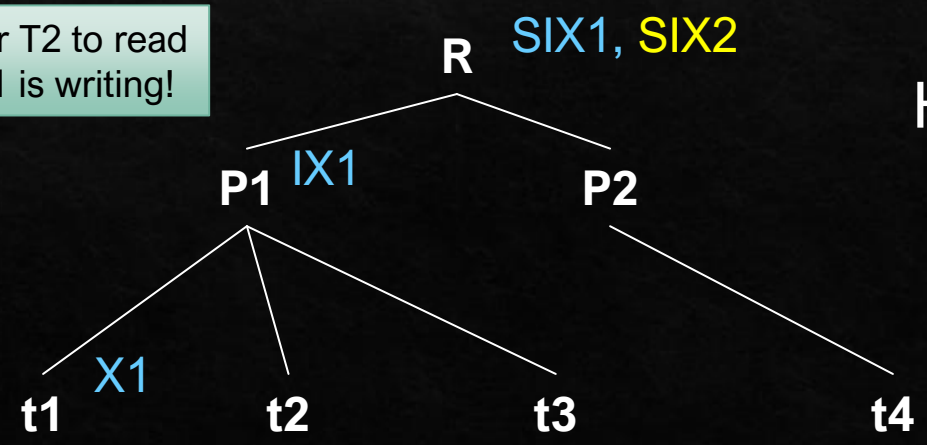
	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Locking Rules



- IS – intent to read at a finer granularity
- IX – intent to write at a finer granularity
- SIX – will read this object + intent to write at a finer granularity
  - Stricter than S and IX, but less strict than X

Not safe for T2 to read R when T1 is writing!



Holder

Requestor

	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Q1A

- Illustrate with an example how allowing upgrading of S to X locks can potentially lead to deadlocks

Q1A

- Illustrate with an example how allowing upgrading of S to X locks can potentially lead to deadlocks

T1	T2
I-S_1(A)	
	I-S_2(A)
I-X_1(A)	
	I-X_2(A)
Deadlock!	

Neither T1 nor T2 can proceed to write safely

Q1A

- Given the following locks, how would the compatibility matrix look like?
  - S – read only
  - X – write
  - U – read but MAY write

		Requestor		
		S	X	U
Holder	S	T	F	
	X	F	F	
	U			

Q1A

- Given the following locks, how would the compatibility matrix look like?
  - S – read only
  - X – write
  - U – read but MAY write

		Requestor		
		S	X	U
Holder	S	T	F	T
	X	F	F	F
	U	F	F	F

Q1A

- Given the following locks, how would the compatibility matrix look like?
  - S – read only
  - X – write
  - U – read but MAY write

Holder \ Requestor	S	X	U
S	T	F	T
X	F	F	F
U	F	F	F

U lock: there is a possibility that Xact may not update, just read only. If this Xact attempt to acquire X lock, it cannot even acquire at all! (i.e. it can't even read)

If a Xact acquires U lock while others are holding S lock, it can still read. But it cannot update until all other S locks have been released

Q1A

- Given the following locks, how would the compatibility matrix look like?
  - S – read only
  - X – write
  - U – read but MAY write

Once a Xact has U lock, we don't allow additional S-locks to be acquired so that this Xact can eventually get to write. If we allow more S-locks while U is held, then the transaction may never get to update because Xacts requiring the S-lock keeps entering

T would still be correct though

If this is T, it could lead to starvation

Holder \ Requestor	S	X	U
S	T	F	T
X	F	F	F
U	T	F	F

Q1B

- **copy(x,y)** – atomically copies value stored in record x (read x) to record y (write y)

		Requestor			
		R(x)	W(x)	C(x,?)	C(?,x)
Holder	R(x)	T	F		
	W(x)	F	F		
	C(x,?)				
	C(?,x)				

Q1B

- **copy(x,y)** – atomically copies value stored in record x (read x) to record y (write y)

Holder \ Requestor	R(x)	W(x)	C(x,?)	C(?,x)
R(x)	T	F	T	F
W(x)	F	F	F	F
C(x,?)	T	F	T	F
C(?,x)	F	F	F	F

Q2

- Compatibility matrix for INCREMENT operation (x = x + 1)

Holder \ Requestor	S	X	I
S	T	F	
X	F	F	
I			

Q2

- Compatibility matrix for INCREMENT operation (x = x + 1)
  - “Special” kind of INCREMENT operation that is independent of ordering; outcome is still the same regardless of ordering
  - Many transactions can hold an I lock concurrently

Holder \ Requestor	S	X	I
S	T	F	F
X	F	F	F
I	F	F	T

Q2

- Is the schedule serializable? If so, what is the equivalent serial schedule?
- S = $r_1(A)$; $r_2(B)$; $r_3(C)$; $incr_2(C)$; $r_3(B)$; $incr_1(C)$; $incr_2(A)$;

Holder \ Requestor	S	X	I
S	T	F	F
X	F	F	F
I	F	F	T

Q2

- Is the schedule serializable? If so, what is the equivalent serial schedule?
- S = $r_1(A)$; $r_2(B)$; $r_3(C)$; $incr_2(C)$; $r_3(B)$; $incr_1(C)$; $incr_2(A)$;
- Conflicting actions
  - Object A: 1→2
  - Object B: none
  - Object C: 3→2, 3→1
  - Serial order: 3→1→2
- S_{serial} = $r_3(C)$; $r_3(B)$; $r_1(A)$; $incr_1(C)$; $r_2(B)$; $incr_2(C)$; $incr_2(A)$;

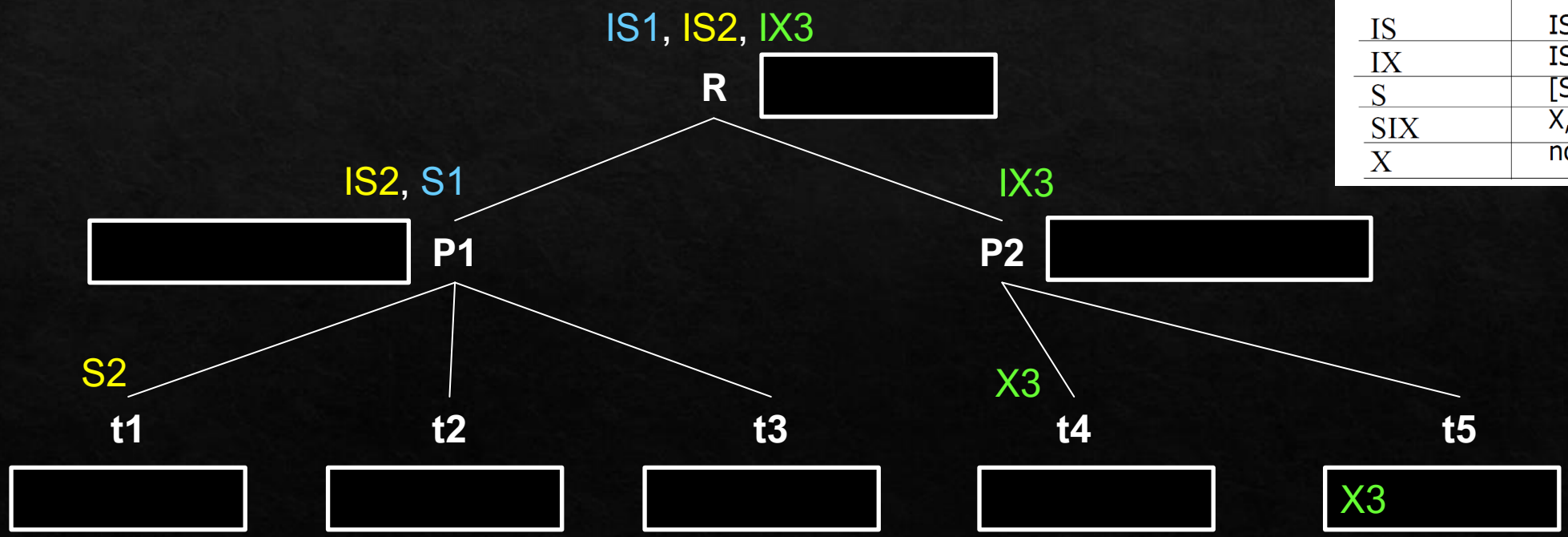
Holder \ Requestor	S	X	I
S	T	F	F
X	F	F	F
I	F	F	T

Q4

Show what are the **next** possible locks to be acquired by T1, T2, T3 and T4

Holder \ Requestor	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Parent locked in	Child can be locked in
IS	IS, S
IX	IS, S, IX, X, SIX
S	[S, IS] not necessary
SIX	X, IX, [SIX]
X	none

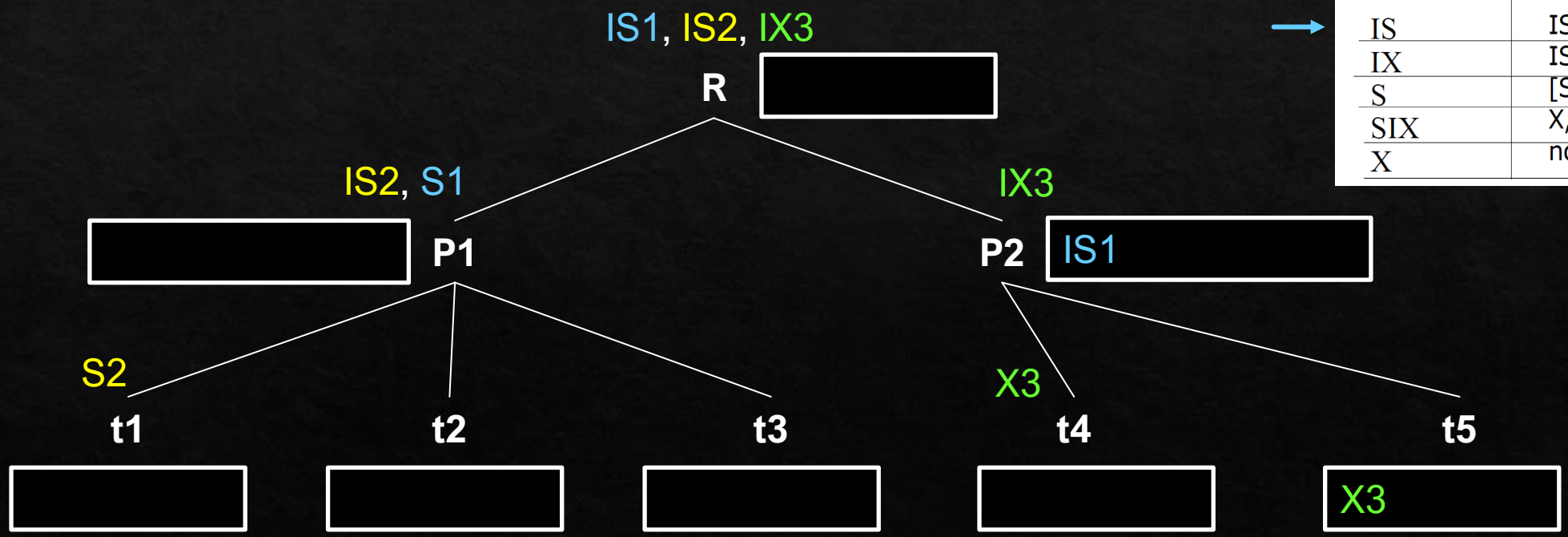


Q4

Show what are the **next** possible locks to be acquired by T1, T2, T3 and T4

Holder \ Requestor	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Parent locked in	Child can be locked in
IS	IS, S
IX	IS, S, IX, X, SIX
S	[S, IS] not necessary
SIX	X, IX, [SIX]
X	none

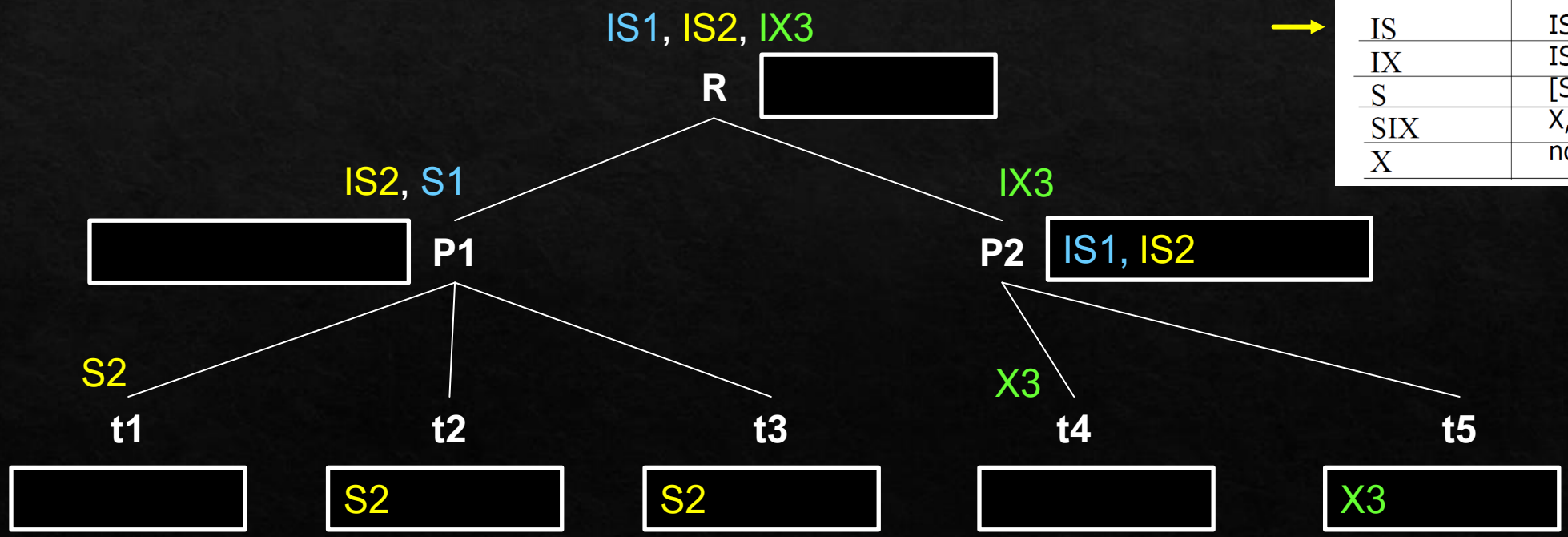


Q4

Show what are the **next** possible locks to be acquired by T1, T2, T3 and T4

Holder \ Requestor	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Parent locked in	Child can be locked in
IS	IS, S
IX	IS, S, IX, X, SIX
S	[S, IS] not necessary
SIX	X, IX, [SIX]
X	none

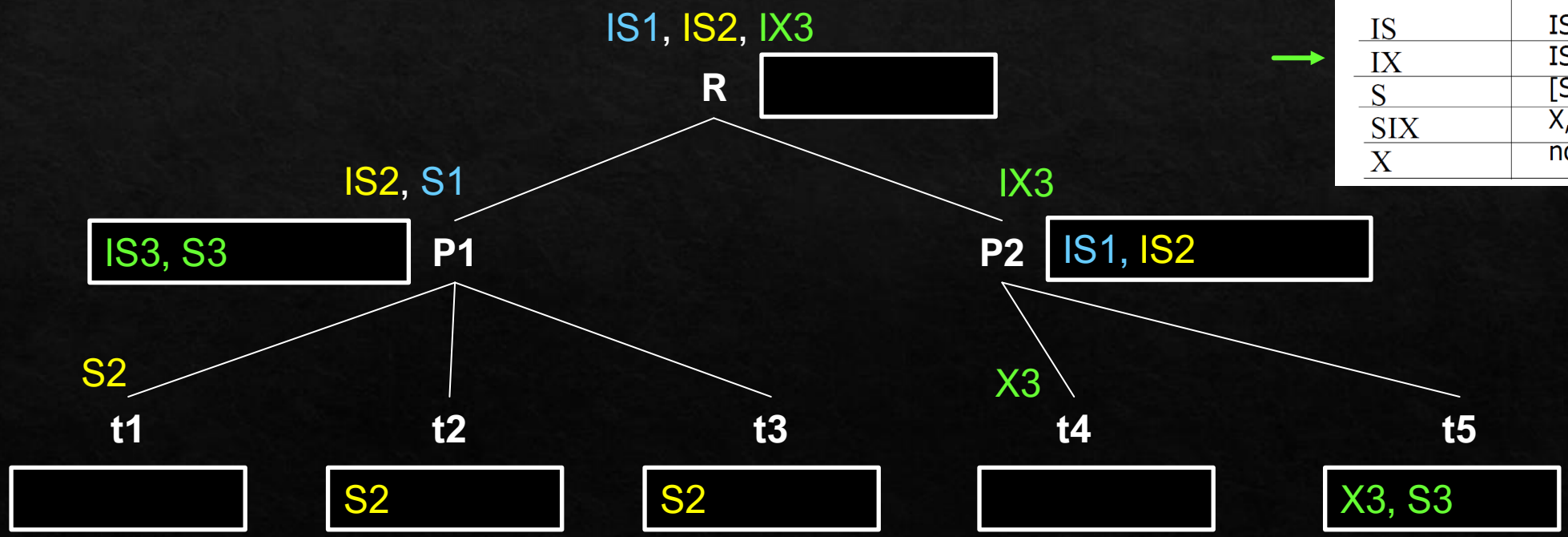


Q4

Show what are the **next** possible locks to be acquired by T1, T2, T3 and T4

Holder \ Requestor	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Parent locked in	Child can be locked in
IS	IS, S
IX	IS, S, IX, X, SIX
S	[S, IS] not necessary
SIX	X, IX, [SIX]
X	none



Q4

Show what are the **next** possible locks to be acquired by T1, T2, T3 and T4

Holder \ Requestor	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Parent locked in	Child can be locked in
IS	IS, S
IX	IS, S, IX, X, SIX
S	[S, IS] not necessary
SIX	X, IX, [SIX]
X	none