Concurrency Control

Hector Garcia-Molina and
Mahmoud SAKR

Invited Lectures

Extensible Databases - 6/10/2021

Dimitri Fontaine

CITUS BLOG AUTHOR PROFILE

PostgreSQL major contributor & author of “The Art of PostgreSQL”. Contributed extension facility & event triggers feature in Postgres. Maintains pg_auto_failover. Speaker at so many conferences.

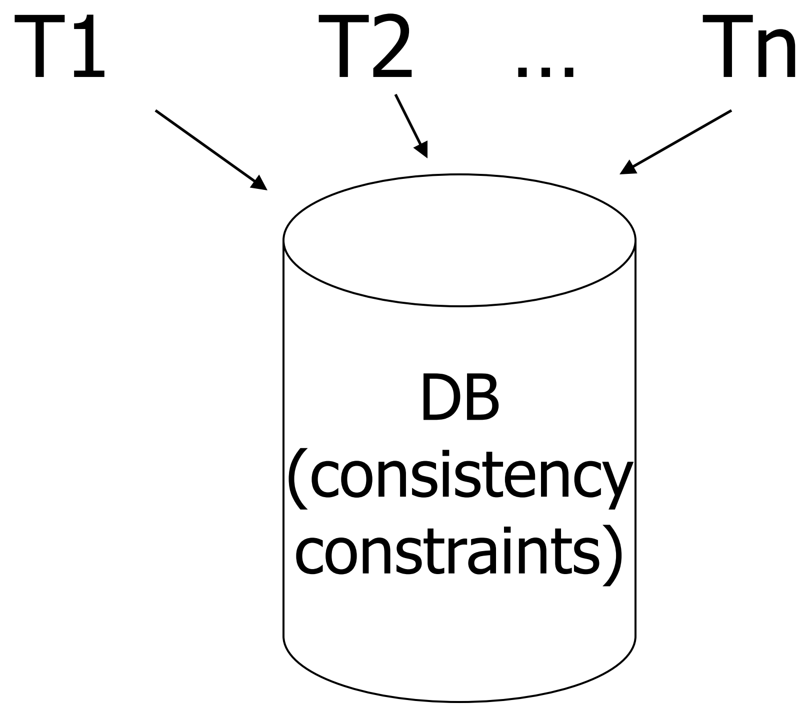
Distributed Databases - 8/12/2021

Marco Slot

CITUS BLOG AUTHOR PROFILE

Lead engineer on the Citus engine team at Microsoft. Speaker at Postgres Conf EU, PostgresOpen, pgDay Paris, Hello World, SIGMOD, & lots of meetups. PhD in distributed systems. Loves mountain hiking.

Chapter 18 [18] Concurrency Control



Example:

T1:	Read(A)	T2:	Read(A)
	$A \leftarrow A+100$		$A \leftarrow A\times2$
	Write(A)		Write(A)
	Read(B)		Read(B)
	$B \leftarrow B+100$		$B \leftarrow B\times2$
	Write(B)		Write(B)

Constraint: $A=B$

Schedule A

T1	T2
Read(A); $A \leftarrow A+100$	
Write(A);	
Read(B); $B \leftarrow B+100$;	
Write(B);	
	Read(A); $A \leftarrow A \times 2$;
	Write(A);
	Read(B); $B \leftarrow B \times 2$;
	Write(B);

Schedule A

T1	T2	A	B
		25	25
Read(A); A $\leftarrow$ A+100			
Write(A);		125	
Read(B); B $\leftarrow$ B+100;			
Write(B);			125
	Read(A);A $\leftarrow$ A$\times$2;		
	Write(A);	250	
	Read(B);B $\leftarrow$ B$\times$2;		
	Write(B);		250
		250	250

Schedule B

T1	T2
	Read(A);A $\leftarrow$ A$\times$2;
	Write(A);
	Read(B);B $\leftarrow$ B$\times$2;
	Write(B);
Read(A); A $\leftarrow$ A+100	
Write(A);	
Read(B); B $\leftarrow$ B+100;	
Write(B);	

Schedule B

T1	T2	A	B
		25	25
	Read(A);A $\leftarrow$ A$\times$2;		
	Write(A);	50	
	Read(B);B $\leftarrow$ B$\times$2;		
	Write(B);		50
Read(A); A $\leftarrow$ A+100			
Write(A);		150	
Read(B); B $\leftarrow$ B+100;			
Write(B);			150
		150	150

Schedule C

T1	T2
Read(A); A $\leftarrow$ A+100	
Write(A);	
	Read(A);A $\leftarrow$ A$\times$2;
	Write(A);
Read(B); B $\leftarrow$ B+100;	
Write(B);	
	Read(B);B $\leftarrow$ B$\times$2;
	Write(B);

Schedule C

T1	T2	A	B
		25	25
Read(A); A $\leftarrow$ A+100			
Write(A);		125	
	Read(A);A $\leftarrow$ A$\times$2;		
	Write(A);	250	
Read(B); B $\leftarrow$ B+100;			
Write(B);			125
	Read(B);B $\leftarrow$ B$\times$2;		
	Write(B);		250
		250	250

Schedule D

T1	T2
Read(A); A $\leftarrow$ A+100	
Write(A);	
	Read(A);A $\leftarrow$ A$\times$2;
	Write(A);
	Read(B);B $\leftarrow$ B$\times$2;
	Write(B);
Read(B); B $\leftarrow$ B+100;	
Write(B);	

Schedule D

T1	T2	A	B
		25	25
Read(A); A $\leftarrow$ A+100			
Write(A);		125	
	Read(A);A $\leftarrow$ A$\times$2;		
	Write(A);	250	
	Read(B);B $\leftarrow$ B$\times$2;		
	Write(B);		50
Read(B); B $\leftarrow$ B+100;			
Write(B);			150
		250	150

Schedule E

Same as Schedule D but with new T2′

T1	T2′
Read(A); A $\leftarrow$ A+100	
Write(A);	
	Read(A);A $\leftarrow$ A$\times$1;
	Write(A);
	Read(B);B $\leftarrow$ B$\times$1;
	Write(B);
Read(B); B $\leftarrow$ B+100;	
Write(B);	

Schedule E

Same as Schedule D
but with new T2′

T1	T2′	A	B
		25	25
Read(A); A $\leftarrow$ A+100			
Write(A);		125	
	Read(A);A $\leftarrow$ A$\times$1;		
	Write(A);	125	
	Read(B);B $\leftarrow$ B$\times$1;		
	Write(B);		25
Read(B); B $\leftarrow$ B+100;			
Write(B);			125
		125	125

- Want schedules that are “good”, regardless of
  - initial state and
  - transaction semantics
- Only look at order of read and writes

Example:

$Sc=r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$

Example:

$$Sc = r_1(A)w_1(A)\underbrace{r_2(A)w_2(A)}\underbrace{r_1(B)w_1(B)}r_2(B)w_2(B)$$

$$Sc' = \underbrace{r_1(A)w_1(A)\ r_1(B)w_1(B)}_{T_1}\underbrace{r_2(A)w_2(A)r_2(B)w_2(B)}_{T_2}$$

The Transaction Game

A								
B								
T1								
T2								

The Transaction Game

A	**r**	**w**	**r**	**w**				
B					**r**	**w**	**r**	**w**
T1	**r**	**w**			**r**	**w**		
T2			**r**	**w**			**r**	**w**

The Transaction Game

A	**r**	**w**	**r**	**w**				
B					**r**	**w**	**r**	**w**
T1	**r**	**w**			**r**	**w**		
T2			**r**	**w**			**r**	**w**

until column hits something

can move column

A	**r**	**w**	**r**	**w**				
B					**r**	**w**	**r**	**w**
T1	**r**	**w**			**r**	**w**		
T2			**r**	**w**			**r**	**w**

move

move

A	**r**	**w**			**r**	**w**		
B			**r**	**w**			**r**	**w**
T1	**r**	**w**	**r**	**w**				
T2					**r**	**w**	**r**	**w**

Schedule D

A	**r**	**w**	**r**	**w**				
B					**r**	**w**	**r**	**w**
T1	**r**	**w**					**r**	**w**
T2			**r**	**w**	**r**	**w**		

However, for Sd:

$$Sd = \underbrace{r_1(A)w_1(A)}r_2(A)w_2(A)\ r_2(B)w_2(B)\underbrace{r_1(B)w_1(B)}$$

- as a matter of fact,

  T_2 must precede T_1

  in any equivalent schedule,

  i.e., $T_2 \rightarrow T_1$

- $T_2 \rightarrow T_1$
- Also, $T_1 \rightarrow T_2$

$$\begin{tikzcd} T_1 \arrow[r, bend left] & T_2 \arrow[l, bend left] \end{tikzcd}$$

$\Rightarrow$ Sd cannot be rearranged into a serial schedule

$\Rightarrow$ Sd is not "equivalent" to any serial schedule

$\Rightarrow$ Sd is "bad"

Returning to Sc

$$Sc = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$$

$$T_1 \rightarrow T_2 \qquad\qquad T_1 \rightarrow T_2$$

Returning to Sc

$$Sc = r_1(A) w_1(A) r_2(A) w_2(A) r_1(B) w_1(B) r_2(B) w_2(B)$$

$$T_1 \rightarrow T_2 \qquad\qquad T_1 \rightarrow T_2$$

☛ no cycles $\Rightarrow$ Sc is "equivalent" to a serial schedule (in this case T_1,T_2)

Concepts

Transaction: sequence of $r_i(x)$, $w_i(x)$ actions

Conflicting actions: $r_1(A)$ / $w_2(A)$ $\quad$ $w_2(A)$ / $r_1(A)$ $\quad$ $w_1(A)$ / $w_2(A)$

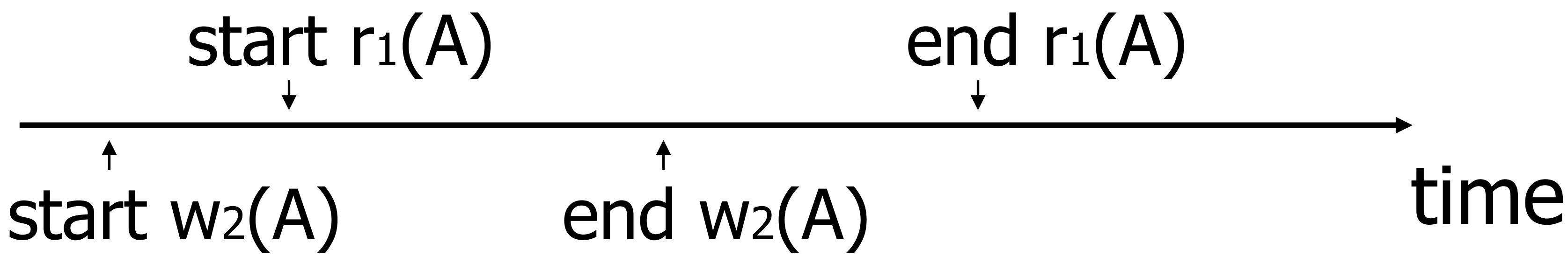
Schedule: represents chronological order in which actions are executed

Serial schedule: no interleaving of actions or transactions

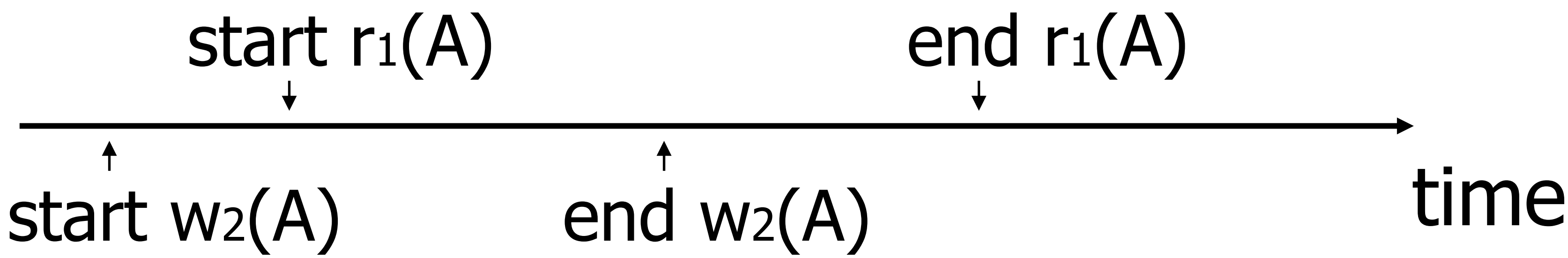
Is it OK to model reads & writes as occurring at a single point in time in a schedule?

- S=… $r_1(x)$ … $w_2(b)$ …

What about conflicting, concurrent actions on same object?



What about conflicting, concurrent actions on same object?



- Assume equivalent to either $r_1(A)\ w_2(A)$
  or $w_2(A)\ r_1(A)$
- $\Rightarrow$ low level synchronization mechanism
- Assumption called "atomic actions"

Definition

S_1, S_2 are conflict equivalent schedules if S_1 can be transformed into S_2 by a series of swaps on non-conflicting actions.

Definition

A schedule is conflict serializable if it is conflict equivalent to some serial schedule.

<u>Precedence graph</u> P(S) (S is schedule)

Nodes: transactions in S

Arcs: $Ti \rightarrow Tj$ whenever

- $p_i(A)$, $q_j(A)$ are actions in S
- $p_i(A) <_S q_j(A)$
- at least one of p_i, q_j is a write

Exercise:

- What is P(S) for
  $S = w_3(A)\ w_2(C)\ r_1(A)\ w_1(B)\ r_1(C)\ w_2(A)\ r_4(A)\ w_4(D)$

- Is S serializable?

Another Exercise:

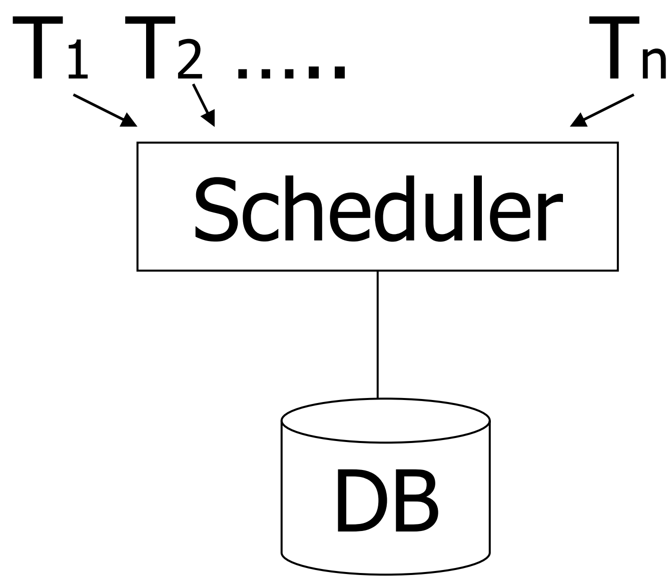
- What is P(S) for
  $S = w_1(A)\ r_2(A)\ \ r_3(A)\ w_4(A)$?

How to enforce serializable schedules?

Option 1: run system, recording P(S);
at end of day, check for P(S)
cycles and declare if execution
was good

How to enforce serializable schedules?

Option 2: prevent P(S) cycles from occurring

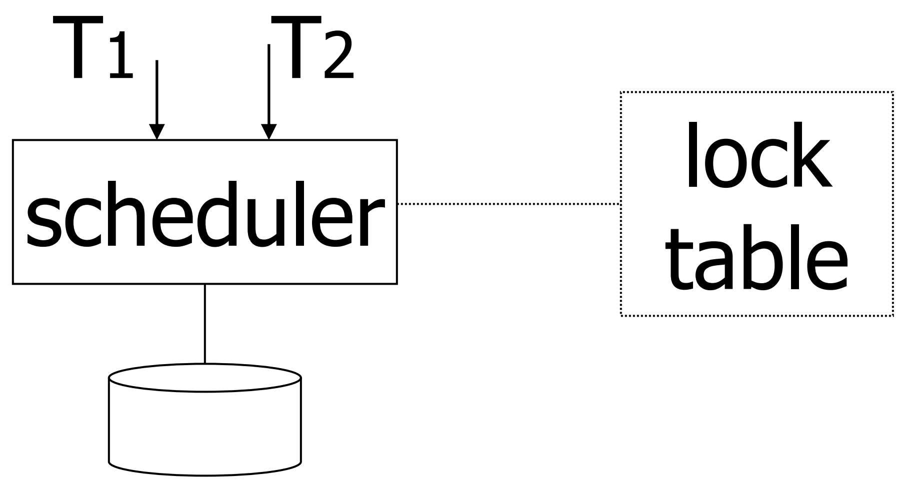


A locking protocol

Two new actions:

lock (exclusive): l_i (A)

unlock: u_i (A)



Rule #1: Well-formed transactions

T_i: … $l_i(A)$ … $p_i(A)$ … $u_i(A)$ …

Rule #2 Legal scheduler

$S = \ldots\ldots.. \; l_i(A) \; \ldots\ldots\ldots\ldots \; u_i(A) \; \ldots\ldots\ldots$

no $l_j(A)$

Exercise:

- What schedules are legal?
  What transactions are well-formed?

$S1 = l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B)$
$r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

$S2 = l_1(A)r_1(A)w_1(B)u_1(A)u_1(B)$
$l_2(B)r_2(B)w_2(B)l_3(B)r_3(B)u_3(B)$

$S3 = l_1(A)r_1(A)u_1(A)l_1(B)w_1(B)u_1(B)$
$l_2(B)r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

Exercise:

- What schedules are legal?
  What transactions are well-formed?

  S1 = $l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B)$
  $r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

  S2 = $l_1(A)r_1(A)w_1(B)u_1(A)u_1(B)$
  $l_2(B)r_2(B)w_2(B)l_3(B)r_3(B)u_3(B)$ $u_2(B)$?

  S3 = $l_1(A)r_1(A)u_1(A)l_1(B)w_1(B)u_1(B)$
  $l_2(B)r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

Schedule F

T1	T2
$l_1(A)$;Read(A)	
$A \leftarrow A+100$;Write(A);$u_1(A)$	
	$l_2(A)$;Read(A)
	$A \leftarrow Ax2$;Write(A);$u_2(A)$
	$l_2(B)$;Read(B)
	$B \leftarrow Bx2$;Write(B);$u_2(B)$
$l_1(B)$;Read(B)	
$B \leftarrow B+100$;Write(B);$u_1(B)$	

Schedule F

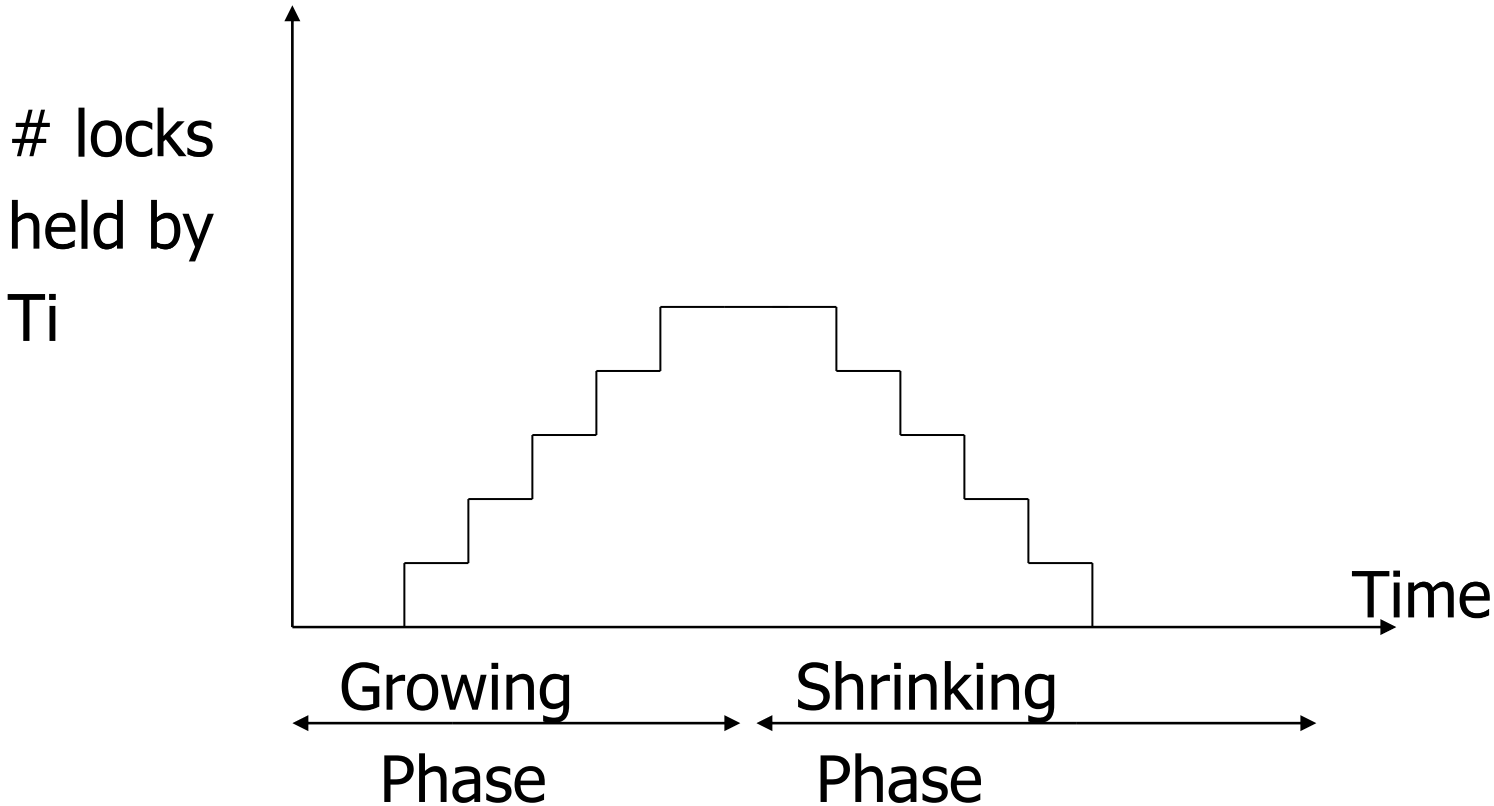
T1	T2	A	B
		25	25
$l_1(A)$;Read(A)			
$A \leftarrow A+100$;Write(A);$u_1(A)$		125	
	$l_2(A)$;Read(A)		
	$A \leftarrow Ax2$;Write(A);$u_2(A)$	250	
	$l_2(B)$;Read(B)		
	$B \leftarrow Bx2$;Write(B);$u_2(B)$		50
$l_1(B)$;Read(B)			
$B \leftarrow B+100$;Write(B);$u_1(B)$			150
		250	150

Rule #3 Two phase locking (2PL)

for transactions

T_i = ... $l_i(A)$ .... $u_i(A)$ ....

no unlocks　　　　no locks



locks
held by
Ti
Time
Growing
Phase
Shrinking
Phase

Schedule G

T1	T2
$l_1(A)$;Read(A)	
$A \leftarrow A+100$;Write(A)	
$l_1(B)$; $u_1(A)$	
	$l_2(A)$;Read(A)
	$A \leftarrow Ax2$;Write(A); $l_2(B)$

delayed

Schedule G

T1	T2	
$l_1(A)$;Read(A)		
$A \leftarrow A+100$;Write(A)		
$l_1(B)$; $u_1(A)$		
	$l_2(A)$;Read(A)	delayed
	$A \leftarrow Ax2$;Write(A);$l_2(B)$	
Read(B);$B \leftarrow B+100$		
Write(B); $u_1(B)$		

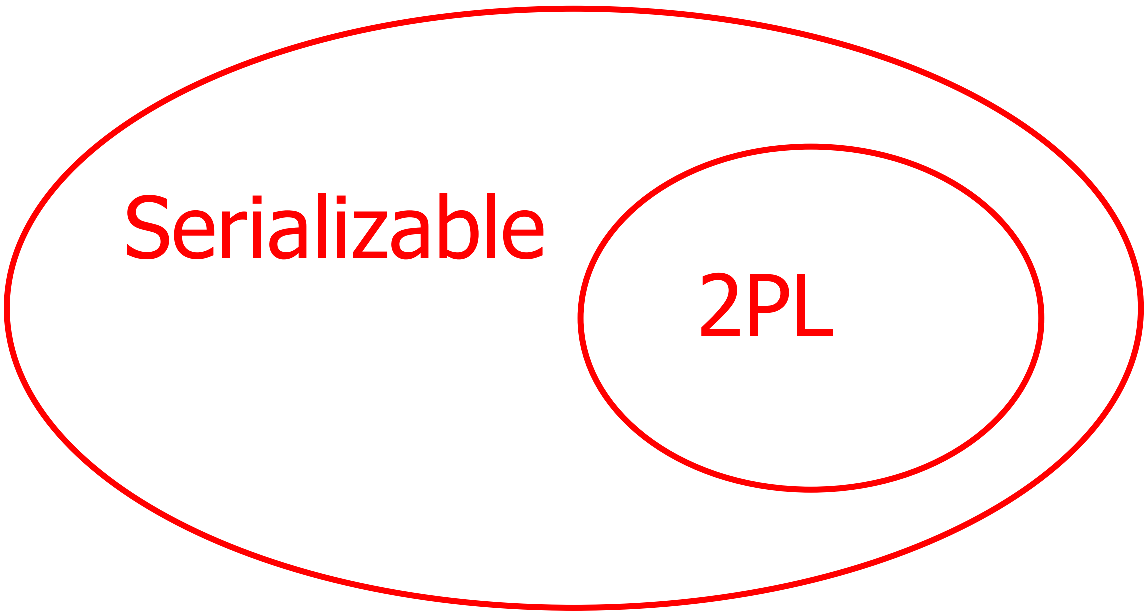
Schedule G

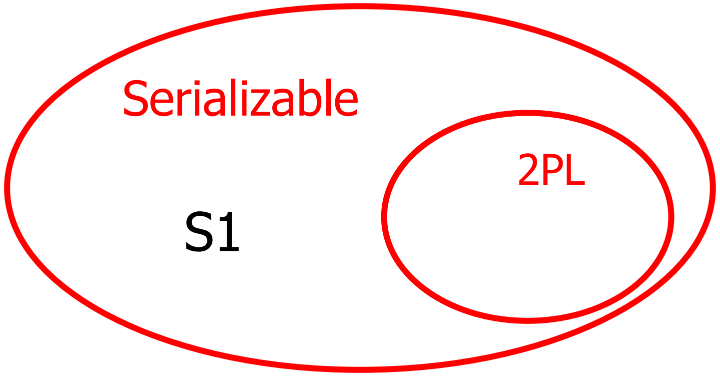
T1	T2
$l_1(A)$;Read(A)	
$A \leftarrow A+100$;Write(A)	
$l_1(B)$; $u_1(A)$	
	$l_2(A)$;Read(A)
	$A \leftarrow Ax2$;Write(A); $l_2(B)$ delayed
Read(B);$B \leftarrow B+100$	
Write(B); $u_1(B)$	
	$l_2(B)$; $u_2(A)$;Read(B)
	$B \leftarrow Bx2$;Write(B);$u_2(B)$;

Schedule H (T$_2$ reversed)

T1	T2
$l_1(A)$; Read(A)	$l_2(B)$;Read(B)
$A \leftarrow A+100$;Write(A)	$B \leftarrow Bx2$;Write(B)
$l_1(B)$ delayed	$l_2(A)$ delayed

2PL subset of Serializable





S1: w1(x) w3(x) w2(y) w1(y)

S1: w1(x) w3(x) w2(y) w1(y)

- S1 cannot be achieved via 2PL:
  The lock by T1 for y must occur after w2(y), so the unlock by T1 for x must occur after this point (and before w1(x)). Thus, w3(x) cannot occur under 2PL where shown in S1 because T1 holds the x lock at that point.
- However, S1 is serializable
  (equivalent to T2, T1, T3).

If you need a bit more practice:

Are our schedules S_C and S_D 2PL schedules?

S_C: w1(A) w2(A) w1(B) w2(B)

S_D: w1(A) w2(A) w2(B) w1(B)

- Beyond this simple 2PL protocol, it is all a matter of improving performance and allowing more concurrency….
  - Shared locks
  - Multiple granularity
  - Inserts, deletes and phantoms
  - Other types of C.C. mechanisms

Shared locks

So far:

$S = ... l_1(A)\ r_1(A)\ u_1(A)\ ...\ l_2(A)\ r_2(A)\ u_2(A)\ ...$

Do not conflict

Shared locks

So far:

$S = \ldots l_1(A)\ r_1(A)\ u_1(A)\ \ldots\ l_2(A)\ r_2(A)\ u_2(A)\ \ldots$

Do not conflict

Instead:

$S = \ldots\ ls_1(A)\ r_1(A)\ ls_2(A)\ r_2(A)\ \ldots.\ us_1(A)\ us_2(A)$

Lock actions (Shared, Exclusive)

$l\text{-}t_i(A)$: lock A in t mode (t is S or X)

$u\text{-}t_i(A)$: unlock t mode (t is S or X)

Shorthand:

$u_i(A)$: unlock whatever modes
T_i has locked A

Rule #1 Well formed transactions

T_i =... l-S_1(A) ... r_1(A) ... u_1 (A) ...

T_i =... l-X_1(A) ... w_1(A) ... u_1 (A) ...

- What about transactions that read and write same object?

Option 1: Request exclusive lock

T_i = ...l-X_1(A) ... r_1(A) ... w_1(A) ... u(A) ...

- What about transactions that read and write same object?

Option 2: Upgrade

(E.g., need to read, but don't know if will write…)

$T_i = \ldots\ l\text{-}S_1(A)\ \ldots\ r_1(A)\ \ldots\ l\text{-}X_1(A)\ \ldots w_1(A)\ \ldots u(A) \ldots$

Think of
- Get 2nd lock on A, or
- Drop S, get X lock

Rule #2 Legal scheduler

$S = \ldots . l\text{-}S_i(A) \ldots \quad \ldots u_i(A) \ldots$

no $l\text{-}X_j(A)$

$S = \ldots l\text{-}X_i(A) \ldots \quad \ldots u_i(A) \ldots$

no $l\text{-}X_j(A)$
no $l\text{-}S_j(A)$

A way to summarize Rule #2

Compatibility matrix

Comp

	S	X
S	true	false
X	false	false

Rule # 3 2PL transactions

No change except for upgrades:

(I) If upgrade gets more locks
(e.g., S $\rightarrow$ {S, X}) then no change!

(II) If upgrade releases read (shared)
lock (e.g., S $\rightarrow$ X)
- can be allowed in growing phase

Lock types beyond S/X

Examples:

(1) increment lock

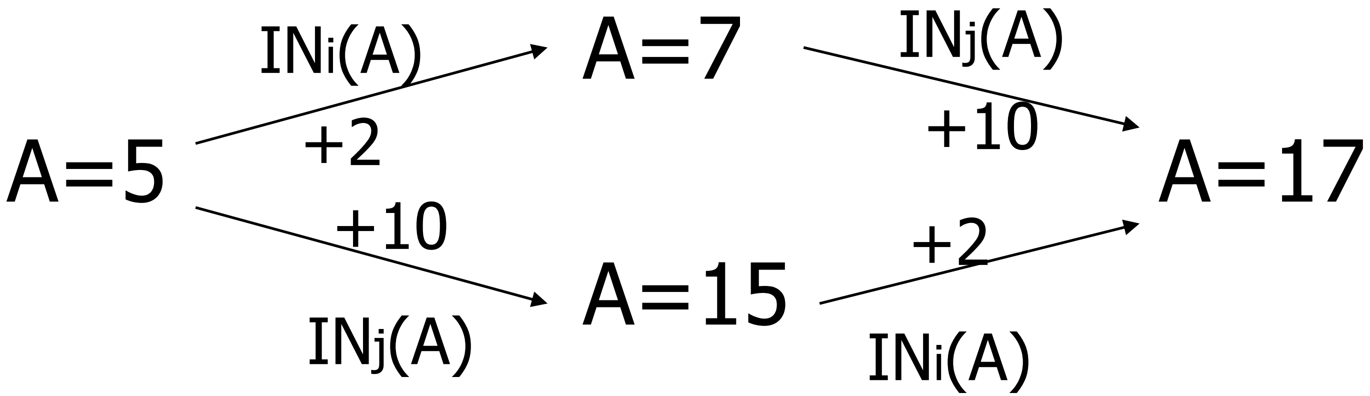
(2) update lock

Example (1): increment lock

- Atomic increment action: $IN_i(A)$

  $\{Read(A);\ A \leftarrow A+k;\ Write(A)\}$

- $IN_i(A)$, $IN_j(A)$ do not conflict!



Comp

	S	X	I
S			
X			
I			

Comp

	S	X	I
S	T	F	F
X	F	F	F
I	F	F	T

Update locks

A common deadlock problem with upgrades:

T1	T2
l-S_1(A)	
	l-S_2(A)
l-X_1(A)	
	l-X_2(A)

--- Deadlock ---

Solution

If T_i wants to read A and knows it may later want to write A, it requests <u>update</u> lock (not shared)

New request

Comp

Lock already held in

	S	X	U
S			
X			
U			

New request

Comp

Lock already held in

	S	X	U
S	T	F	T
X	F	F	F
U	F	F	F

Note: object A may be locked in different modes at the same time...

$$S_1 = \ldots l\text{-}S_1(A) \ldots l\text{-}S_2(A) \ldots l\text{-}U_3(A) \ldots \begin{cases} l\text{-}S_4(A) \ldots ? \\ l\text{-}U_4(A) \ldots ? \end{cases}$$

Note: object A may be locked in different modes at the same time...

$$S_1 = \ldots l\text{-}S_1(A) \ldots l\text{-}S_2(A) \ldots l\text{-}U_3(A) \ldots \begin{cases} l\text{-}S_4(A) \ldots ? \\ l\text{-}U_4(A) \ldots ? \end{cases}$$

- To grant a lock in mode t, mode t must be compatible with all currently held locks on object

How does locking work in practice?

- Every system is different

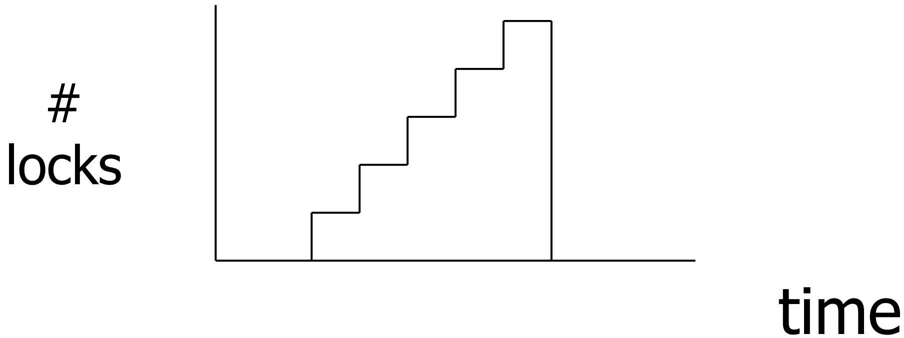
  (E.g., may not even provide
  CONFLICT-SERIALIZABLE schedules)

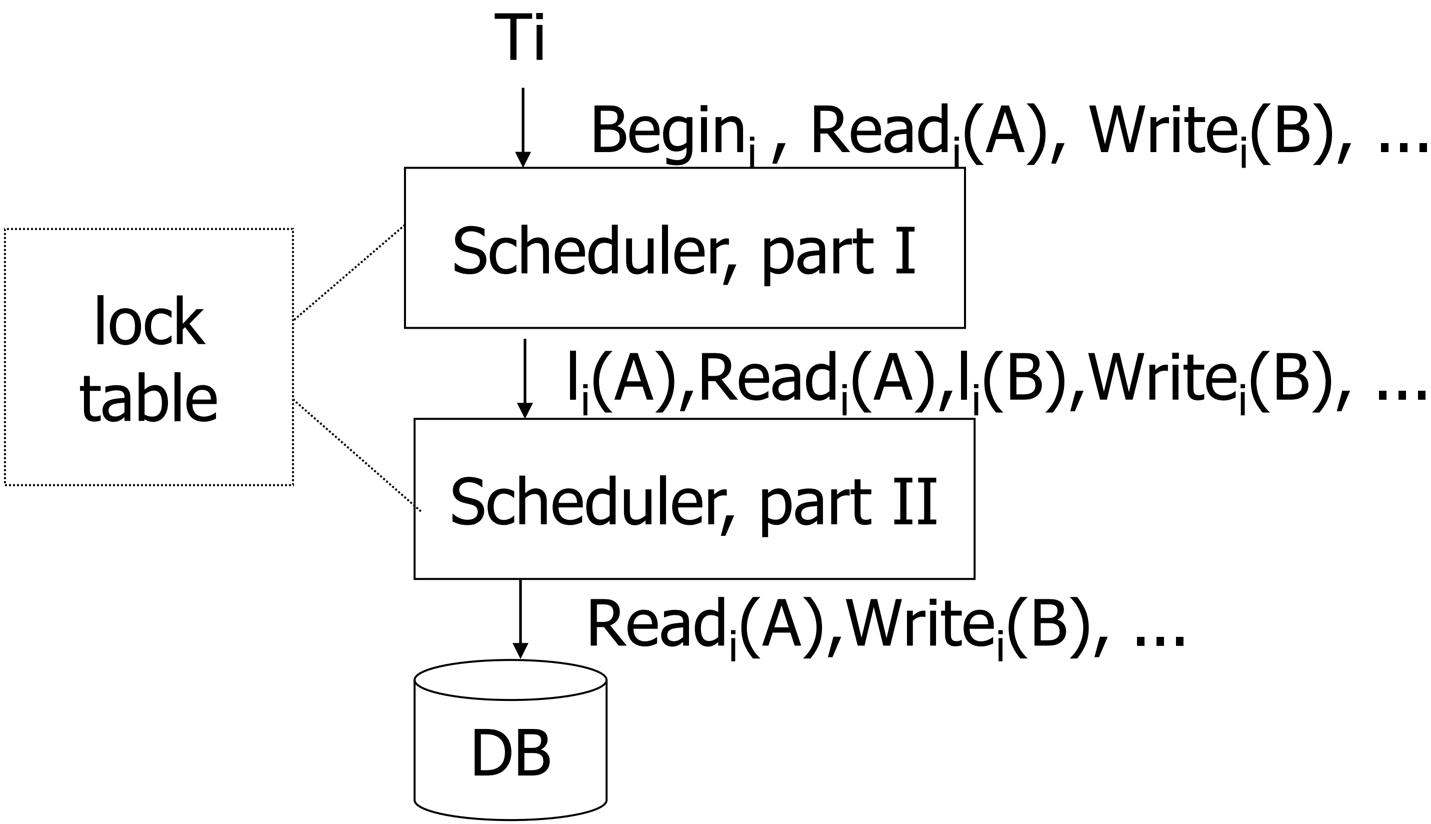
- But here is one (simplified) way ...

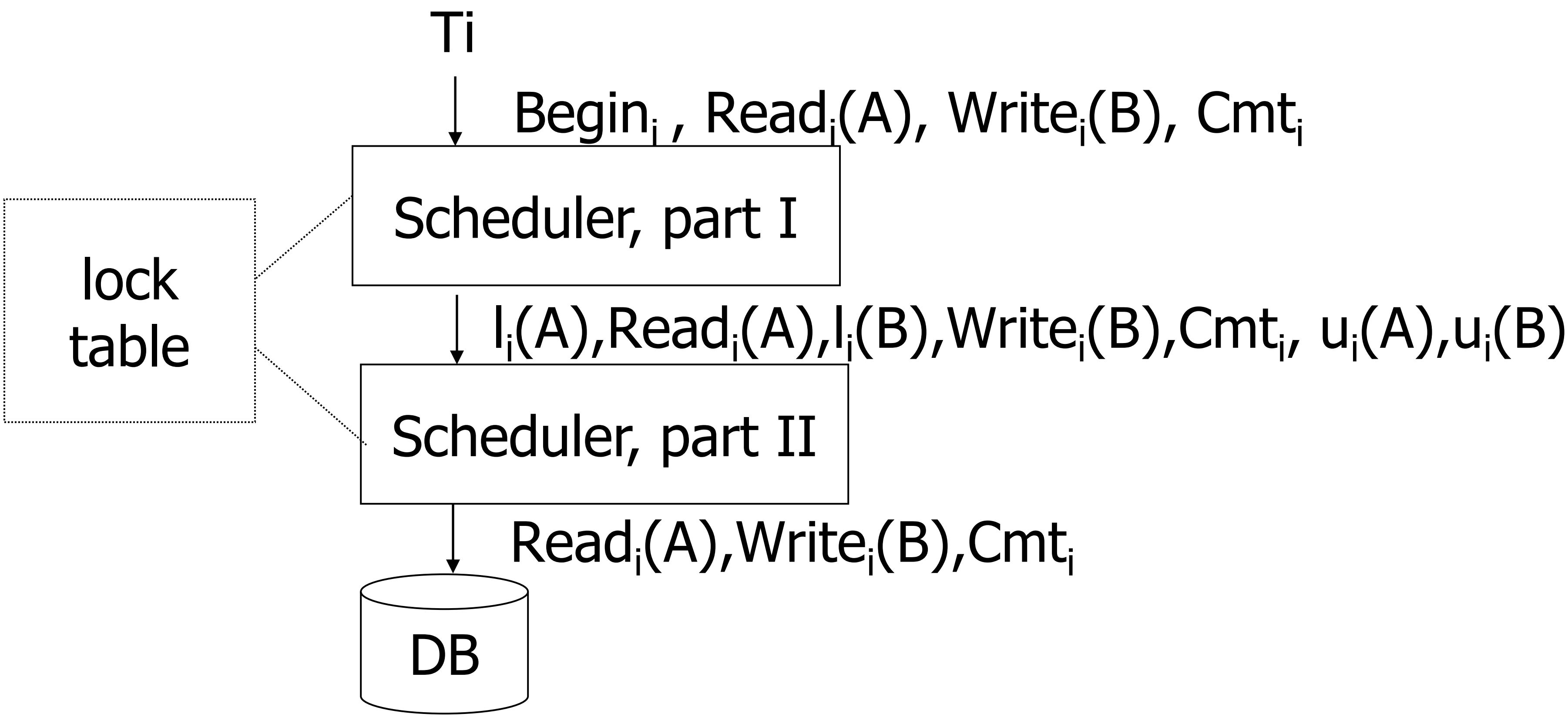
Sample Locking System:

(1) Don’t trust transactions to request/release locks

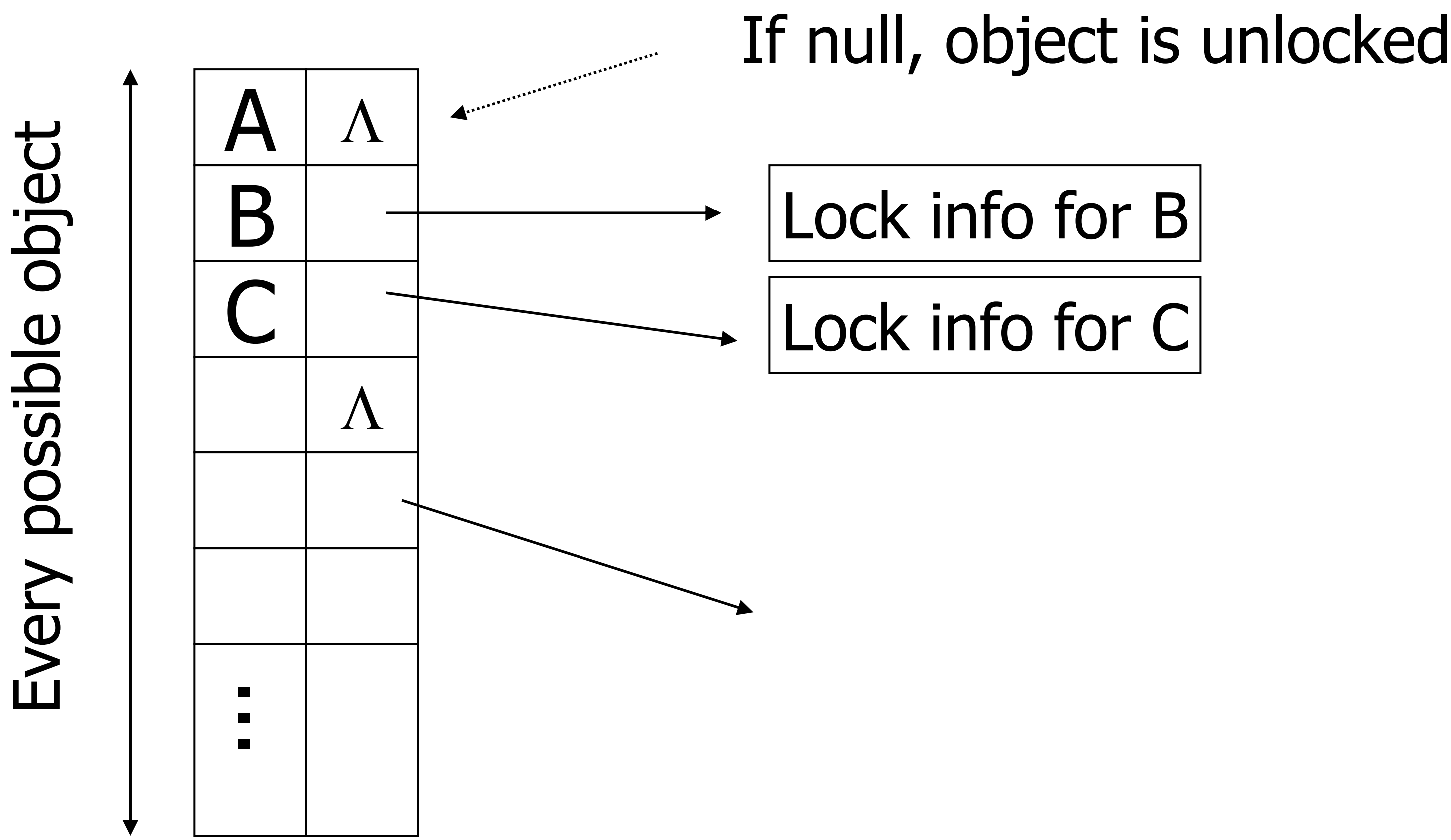
(2) Hold all locks until transaction commits



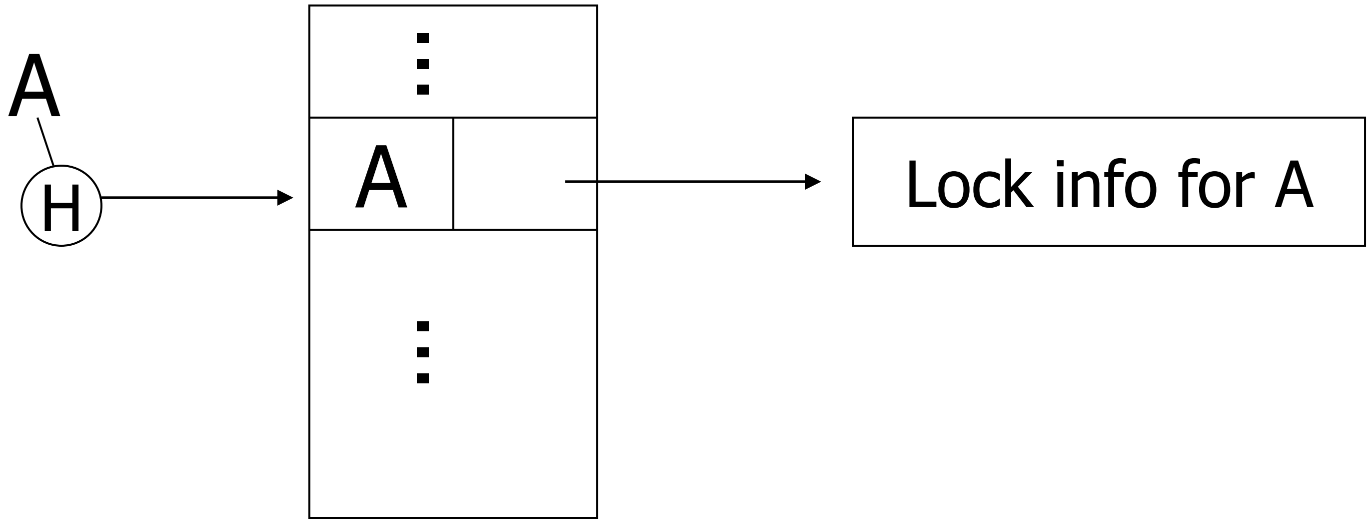

Ti
Begin$_i$, Read$_i$(A), Write$_i$(B), ...
Scheduler, part I
lock
table
l$_i$(A),Read$_i$(A),l$_i$(B),Write$_i$(B), ...
Scheduler, part II
Read$_i$(A),Write$_i$(B), ...
DB


Ti
$Begin_i$, $Read_i(A)$, $Write_i(B)$, Cmt_i
Scheduler, part I
lock table
$l_i(A)$,$Read_i(A)$,$l_i(B)$,$Write_i(B)$,Cmt_i, $u_i(A)$,$u_i(B)$
Scheduler, part II
$Read_i(A)$,$Write_i(B)$,Cmt_i
DB

Lock table   Conceptually

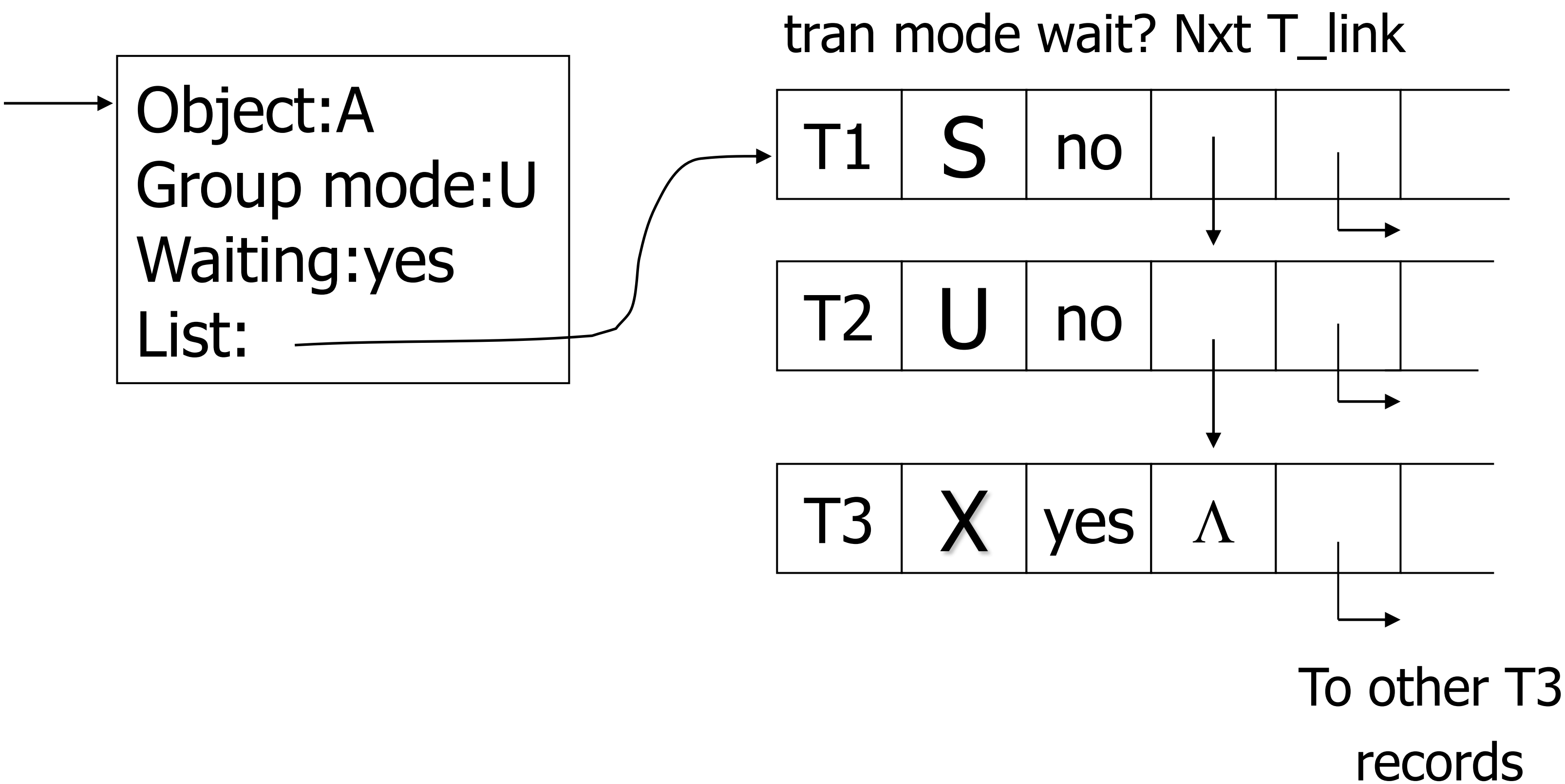


But use hash table:

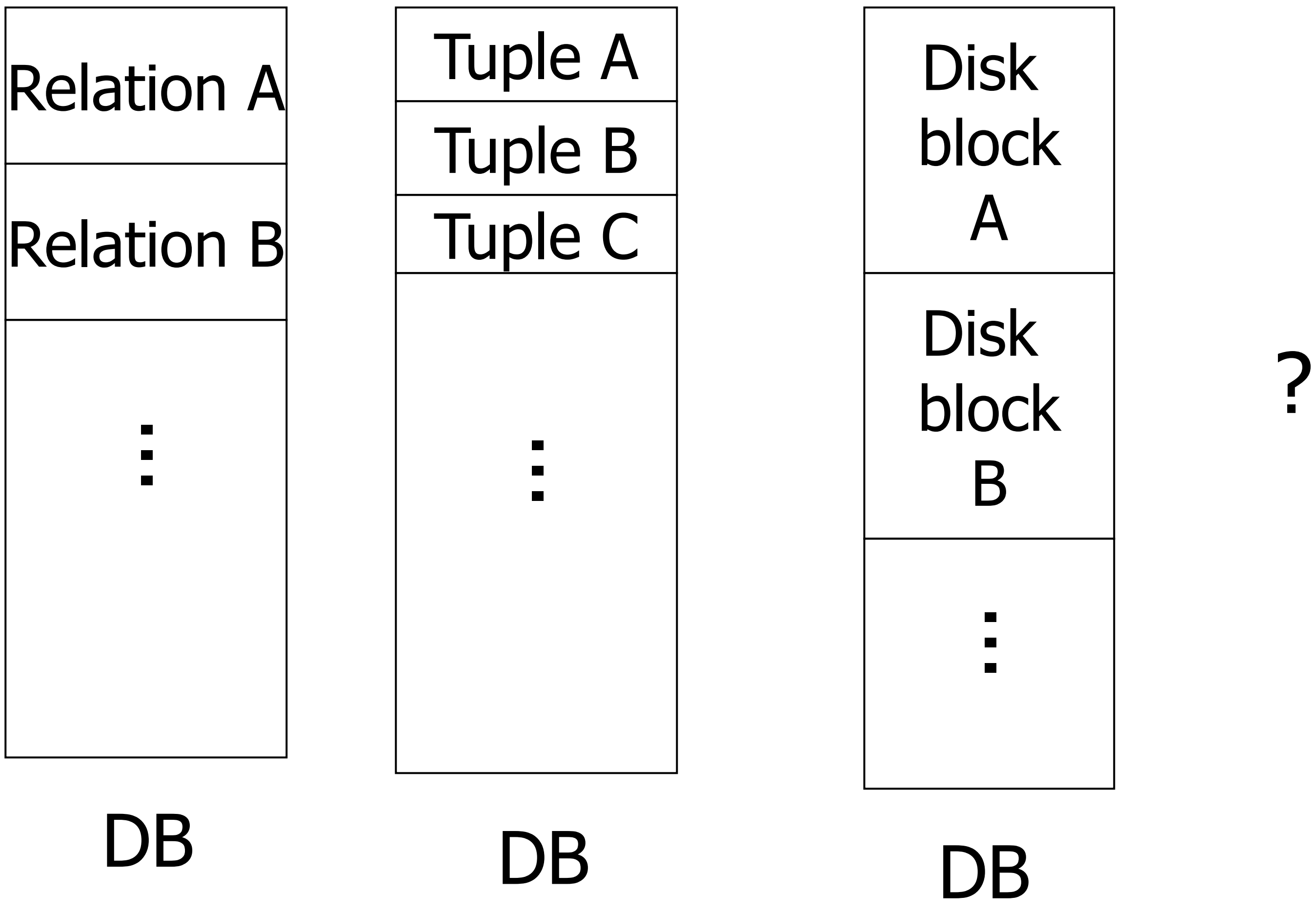


If object not found in hash table, it is unlocked

Lock info for A - example



What are the objects we lock?

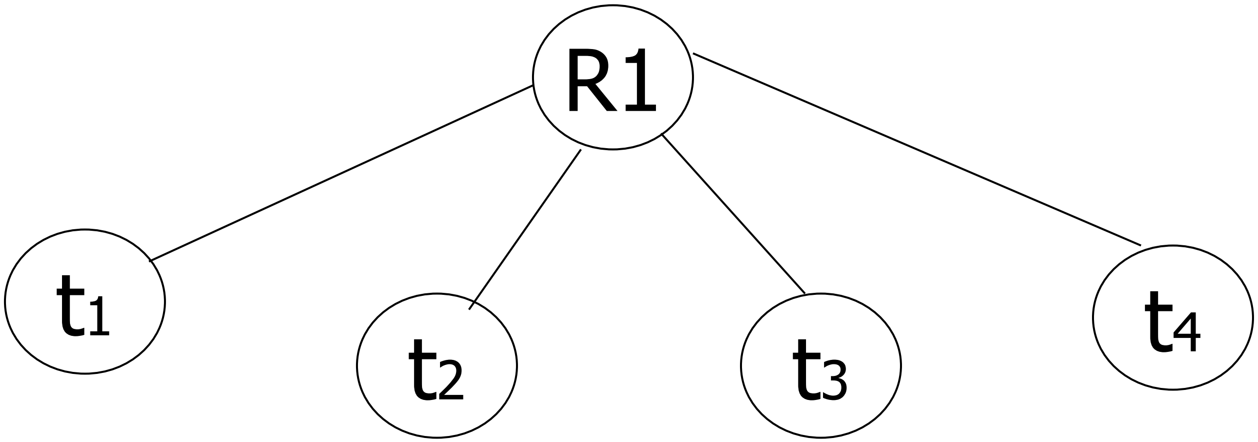


- Locking works in any case, but should we choose small or large objects?

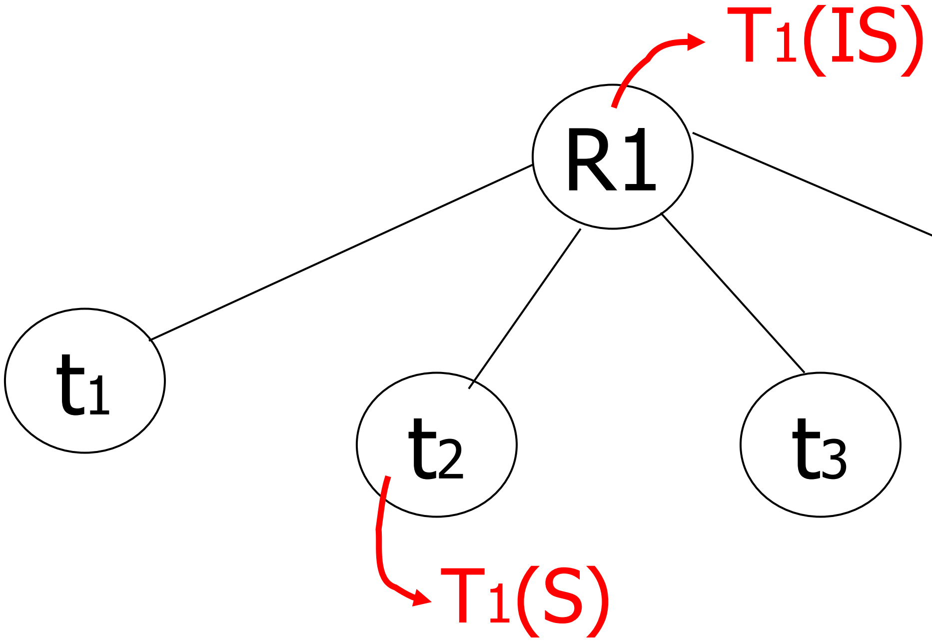
- Locking works in any case, but should we choose <u>small</u> or <u>large objects?</u>

- If we lock <u>large</u> objects (e.g., Relations)
  - Need few locks
  - Low concurrency
- If we lock small objects (e.g., tuples,fields)
  - Need more locks
  - More concurrency

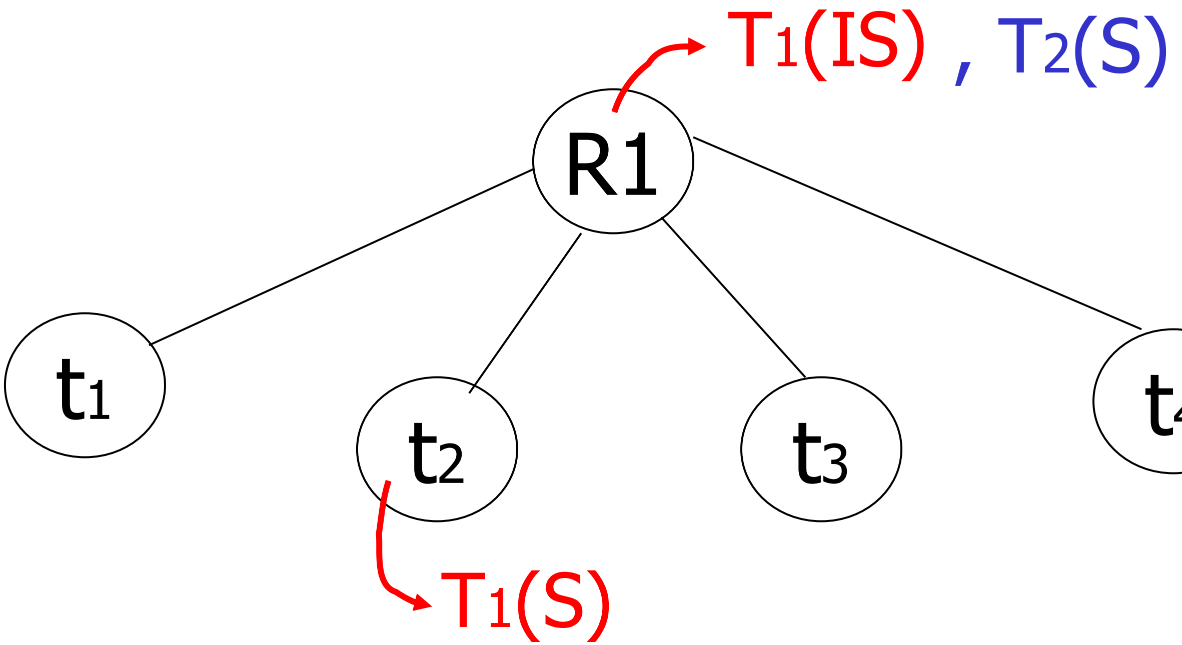
Example



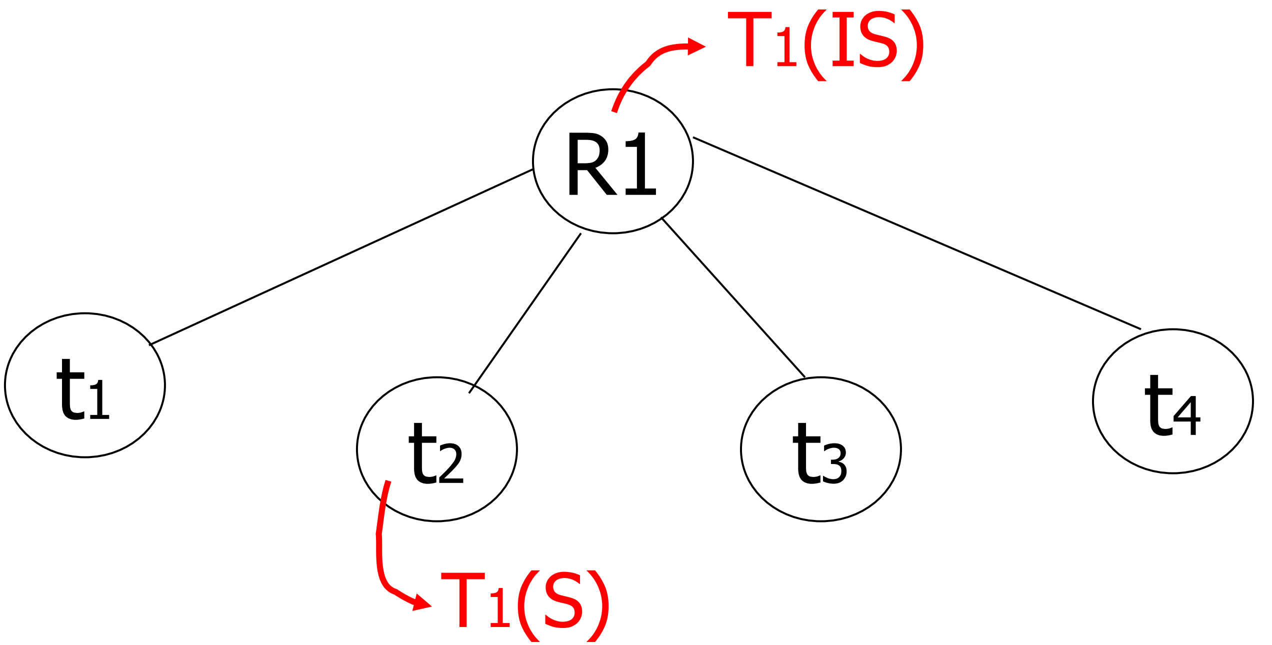
Example



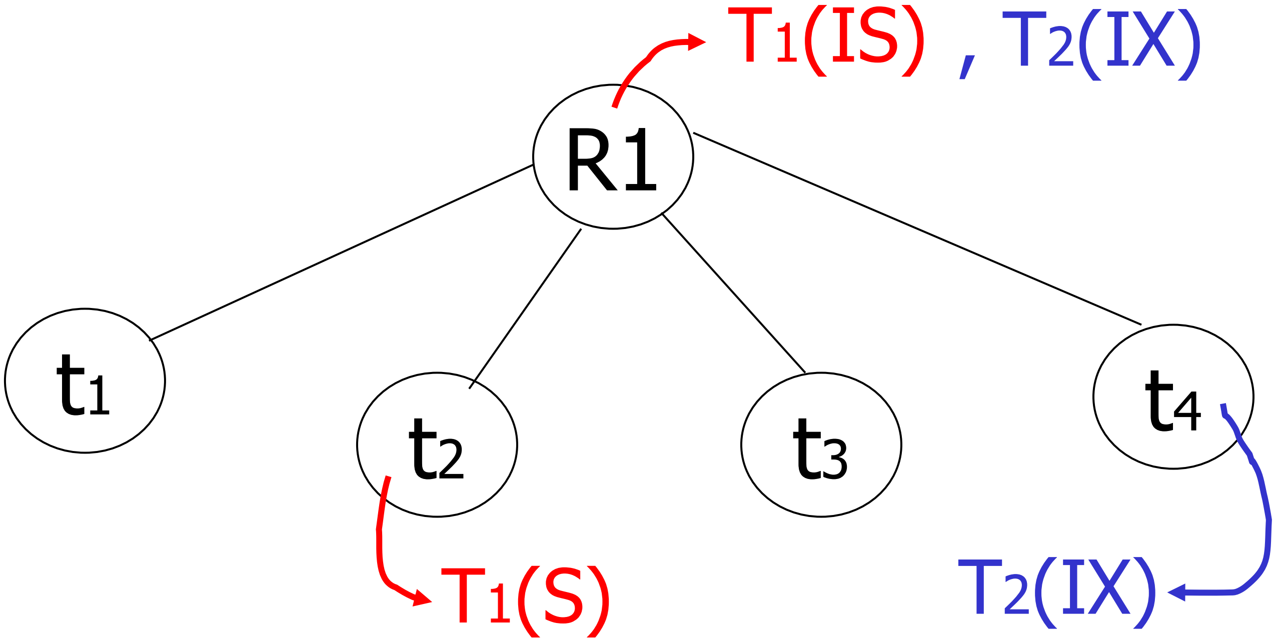
Example



Example (b)



Example



Multiple granularity

Comp (Holder \ Requestor)	IS	IX	S	SIX	X
IS					
IX					
S					
SIX					
X					

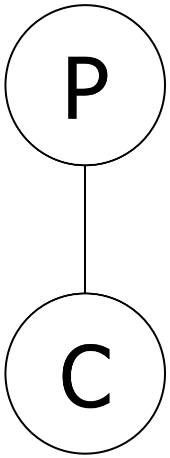
Multiple granularity

Comp Requestor

Holder

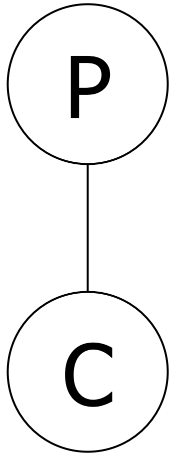
	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Parent locked in	Child can be locked in
IS	
IX	
S	
SIX	
X	



Parent locked in	Child can be locked by same transaction in
IS	IS, S
IX	IS, S, IX, X, SIX
S	none
SIX	X, IX, [SIX]
X	none

not necessary

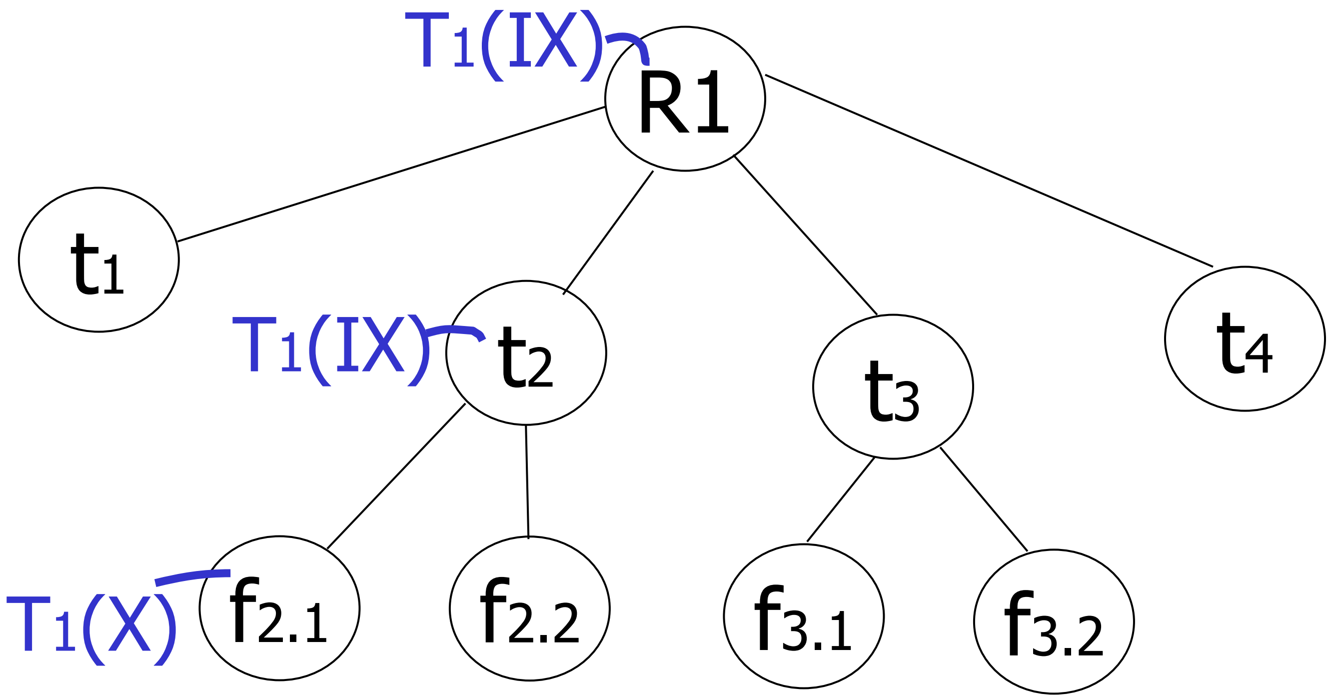


Rules

(1) Follow multiple granularity comp function
(2) Lock root of tree first, any mode
(3) Node Q can be locked by Ti in S or IS only if parent(Q) locked by Ti in IX or IS
(4) Node Q can be locked by Ti in X,SIX,IX only if parent(Q) locked by Ti in IX,SIX
(5) Ti is two-phase
(6) Ti can unlock node Q only if none of Q's children are locked by Ti

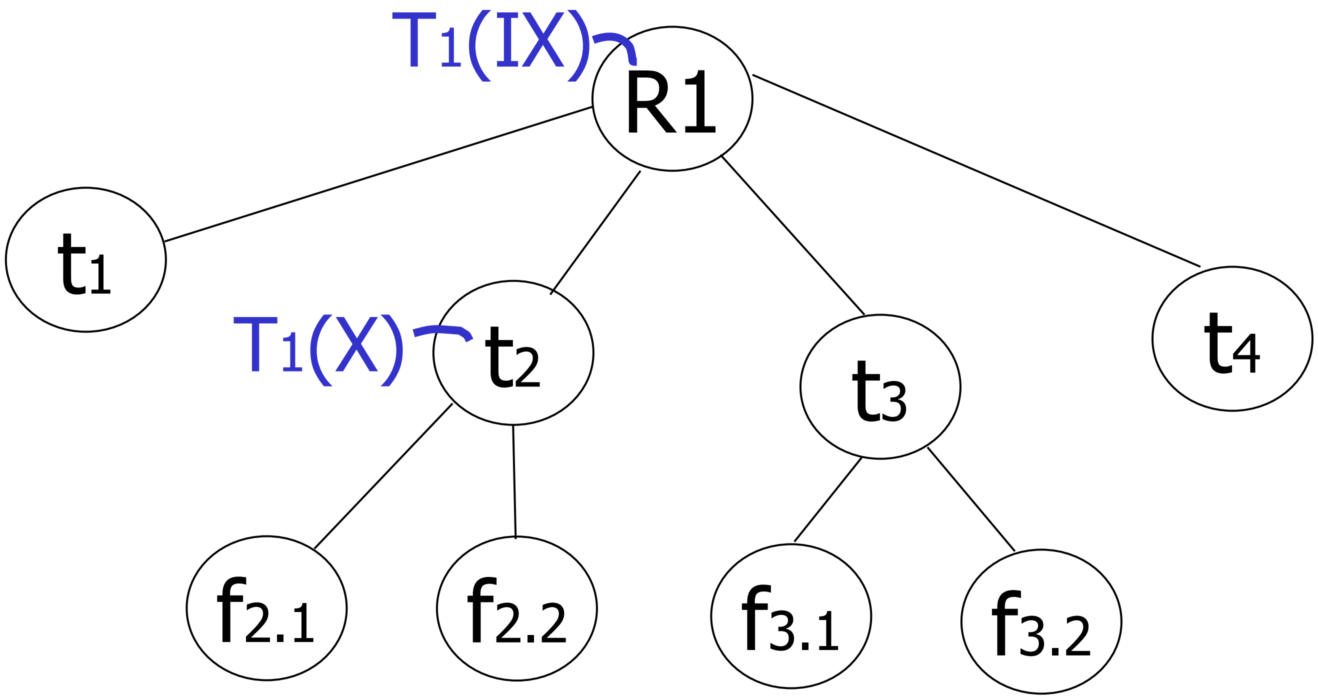
Exercise:

- Can T_2 access object $f_{2.2}$ in X mode? What locks will T_2 get?



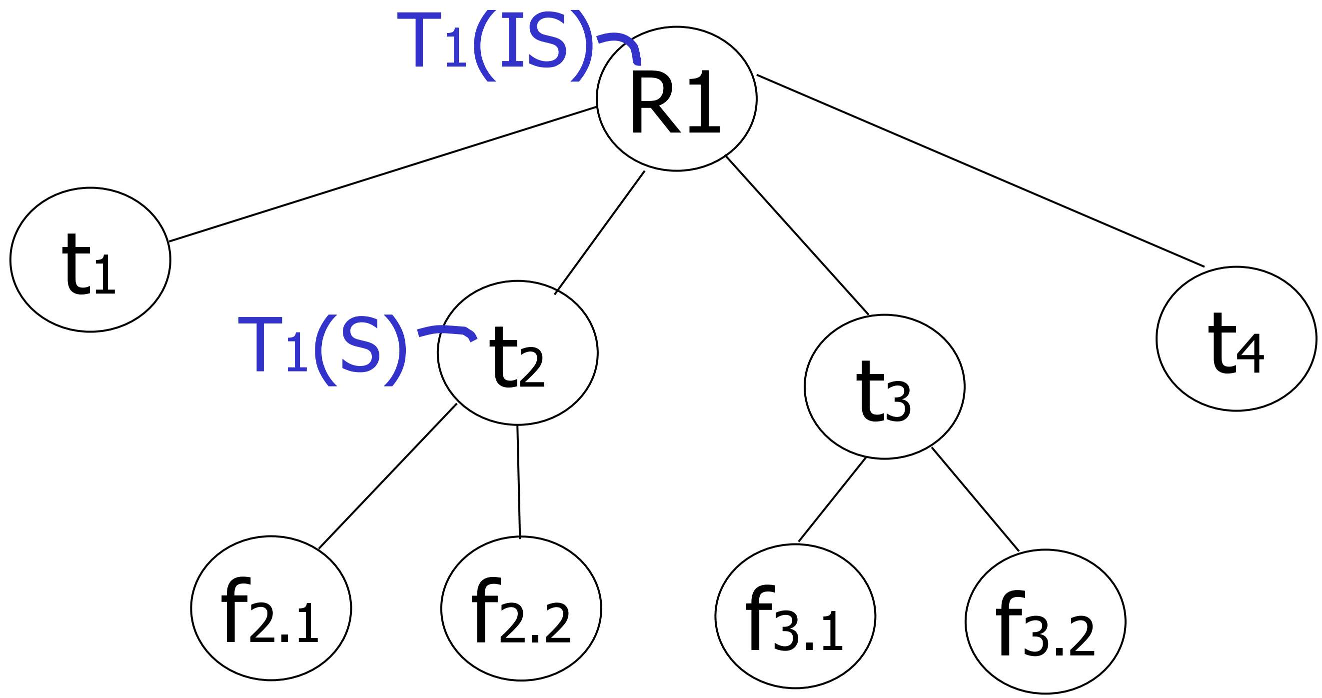
Exercise:

- Can T_2 access object $f_{2.2}$ in X mode? What locks will T_2 get?



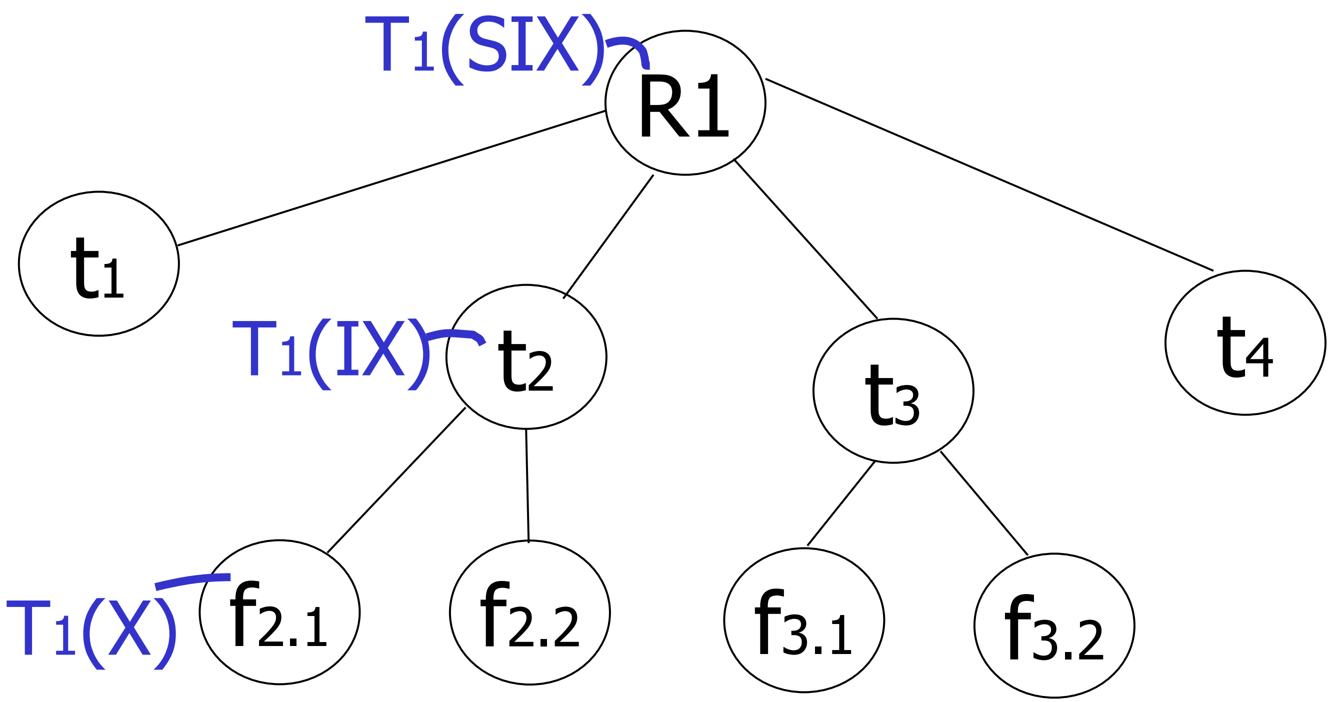
Exercise:

- Can T_2 access object $f_{3.1}$ in X mode? What locks will T_2 get?



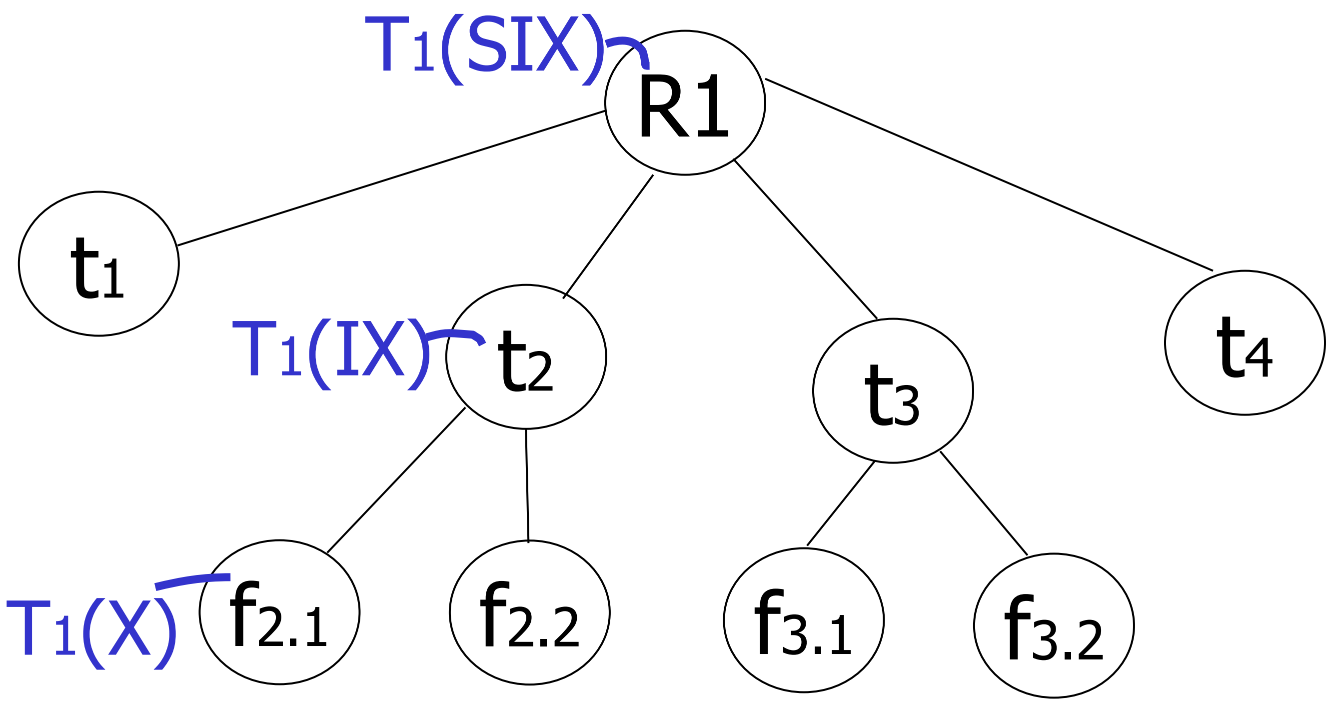
Exercise:

- Can T_2 access object $f_{2.2}$ in S mode? What locks will T_2 get?



Exercise:

- Can T_2 access object $f_{2.2}$ in X mode? What locks will T_2 get?



Reading

Ch18 Concurrency Control

Héctor García-Molina, Jeffrey Ullman, and Jennifer Widom. Database Systems: The Complete Book.