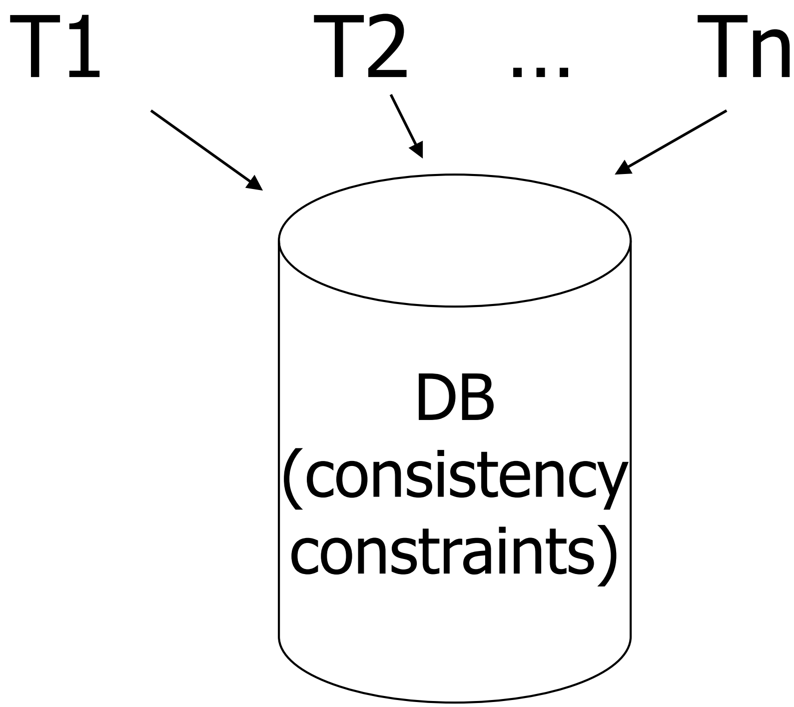
Concurrency Control

Concurrency Control



Transactions

A transaction is an execution of a program having the following properties:

Atomicity

A transaction either happens or doesn’t.

A transaction either completes and the results become visible or no results are visible.

Consistency

Transactions preserve correctness of database.

Isolation

Each transaction is unaware of other transactions executing concurrently

Durability

The results of a completed transaction are permanently installed within D.B.

ACID properties

A transaction has exactly one of two possible outcomes:

a) <u>Commit</u>

Program execution completes and the results become permanent in database.

b) <u>Abort</u>

Program execution was not successful. “Results” are not installed into database.

- Transactions may abort due to hardware failure, system error, bad input data, or as a means to ensure consistency.

Example:

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

Constraint: $A=B$

Main idea of concurrency control

1. An execution without any interleaving is OK

   serial: T1 ; T2 or T2 ; T1

2. If an execution has <u>the same effect</u> as a serial execution then it is also acceptable

   serializable

> Main goal of concurrency control:
>
> guarantee serializability

Serial Schedule A ("good" by definition)

T1	T2	A	B
		25	25
Read(A); A ← A+100			
Write(A);		125	
Read(B); B ← B+100;			
Write(B);			125
	Read(A);A ← A×2;		
	Write(A);	250	
	Read(B);B ← B×2;		
	Write(B);		250
		250	250

Serial Schedule B (equally “good”)

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

Interleaved Schedule C (good because it is equivalent to A)

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

A and C are equivalent because if they start from same initial values they end up with same results

Interleaved Schedule D (bad!)

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

Same as Schedule D
but with new T2′

Schedule E (good by "accident")

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

The accident being the particular semantics

- Want schedules that are “good”, I.e., equivalent to serial regardless of
  - initial state and
  - transaction semantics
- Only look at order of read and writes


What we say to dogs
Okay, Ginger! I've had it! You stay out of the garbage! Understand, Ginger? Stay out of the garbage, or else!
What they hear
blah blah GINGER blah blah blah blah blah blah blah blah GINGER blah blah blah blah blah...

What we say to a database

"delete all movies not directed by Berto"

read, read, write, read, read….

What the database hears

Example of a read/write schedule:

SC=r1(A)w1(A)r2(A)w2(A)r1(B)w1(B)r2(B)w2(B)

Definition

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

Non-conflicting actions:

- actions on different data
- read/read on the same data

Definition

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

Example:

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

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

However, for SD:

$SD = r_1(A)w_1(A)r_2(A)w_2(A)\ r_2(B)w_2(B)r_1(B)w_1(B)$

- as a matter of fact,

  T_2 must precede T_1

  in any equivalent schedule,

  i.e., $T_2 \rightarrow T_1$
- And vice versa

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

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

⇨ SD cannot be rearranged into a serial schedule

⇨ SD is not "equivalent" to any serial schedule

⇨ SD is "bad"

Returning to SC

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

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

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

Precedence graph $P(S)$ (S is schedule)

Nodes: transactions in S

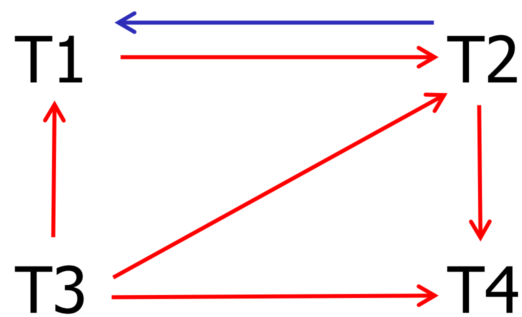
Arcs: $Ti \rightarrow Tj$ whenever

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

$<_S$: order of appearance in schedule

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?

Lemma

S_1, S_2 conflict equivalent $\Rightarrow P(S_1)=P(S_2)$

Proof:

Assume $P(S_1) \neq P(S_2)$

$\Rightarrow \exists\ T_i, T_j$: $T_i \rightarrow T_j$ in S_1 and not in S_2

$$\Rightarrow \begin{array}{l} S_1 = \ldots p_i(A) \ldots q_j(A) \ldots \\ S_2 = \ldots q_j(A) \ldots p_i(A) \ldots \end{array} \quad \left\{ \begin{array}{l} p_i, q_j \\ \text{conflict} \end{array} \right.$$

$\Rightarrow S_1, S_2$ not conflict equivalent

Lemma

S_1, S_2 conflict equivalent $\Rightarrow P(S_1)=P(S_2)$

Is the converse true?

A: yes B: no

Note: $P(S_1)=P(S_2) \not\Rightarrow S_1, S_2$ conflict equivalent

<u>Counter example:</u>

$S_1=w_1(A)\ r_2(A) \quad w_2(B)\ r_1(B)$

$S_2=r_2(A)\ w_1(A) \quad r_1(B)\ w_2(B)$

Theorem

$P(S_1)$ acyclic $\Longleftrightarrow$ S_1 conflict serializable

($\Leftarrow$) Assume S_1 is conflict serializable

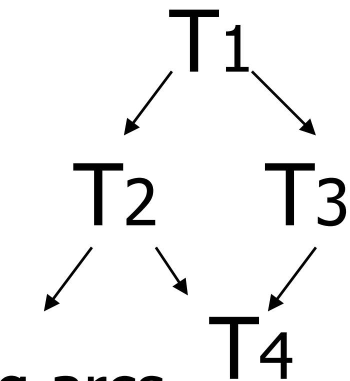
$\Rightarrow \exists\ S_s$: S_s, S_1 conflict equivalent

$\Rightarrow P(S_s) = P(S_1)$

$\Rightarrow P(S_1)$ acyclic since $P(S_s)$ is acyclic

Theorem

$P(S_1)$ acyclic $\Longleftrightarrow$ S_1 conflict serializable



($\Rightarrow$) Assume $P(S_1)$ is acyclic

Transform S_1 as follows:

(1) Take T_1 to be transaction with no incoming arcs

(2) Move all T_1 actions to the front

$$S_1 = \ldots\ldots\;\; q_j(A)\ldots\ldots p_1(A)\ldots..$$

(3) we now have S_1 = < T_1 actions >< ... rest ... >

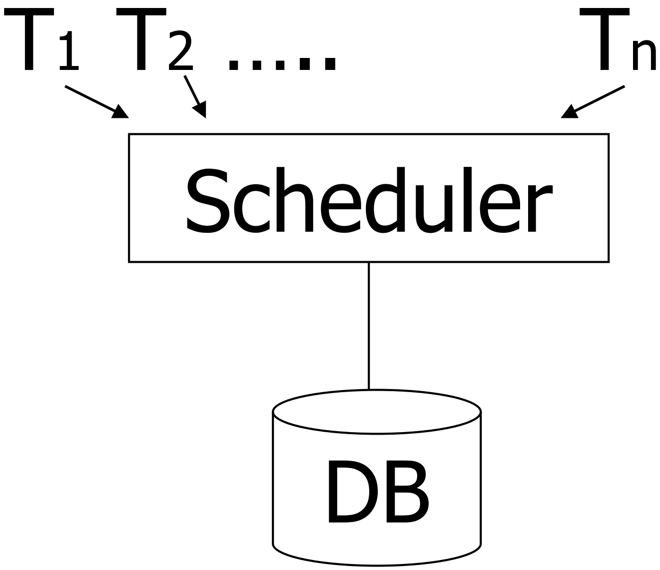
(4) repeat above steps to serialize rest!

How to enforce serializable schedules?

Option 1: run system, recording P(S);
check for P(S) cycles and
declare if execution was good;
or abort transactions as soon
as they generate a cycle

How to enforce serializable schedules?

Option 2: prevent P(S) cycles from occurring

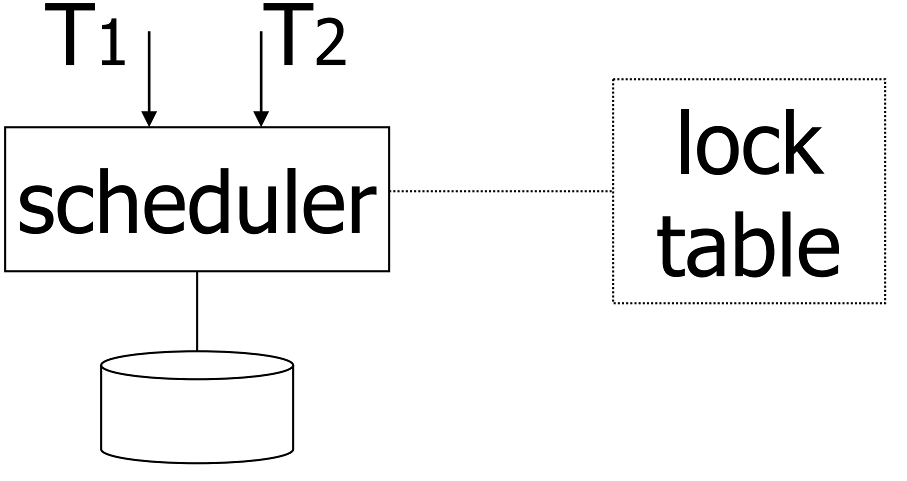


A locking protocol

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

$S = \ldots\ldots.. \; l_i(A) \; \underbrace{\ldots\ldots\ldots\ldots}_{\text{no } l_j(A)} \; u_i(A) \; \ldots\ldots\ldots$

Exercise:

A: S1 correct

B: S2 correct

C: S3 correct

- What schedules are correct?

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

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

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

C: S3 correct

Exercise:

- What schedules are correct?

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

Just having locks is not enough!

Example: this allows Schedule F (bad)

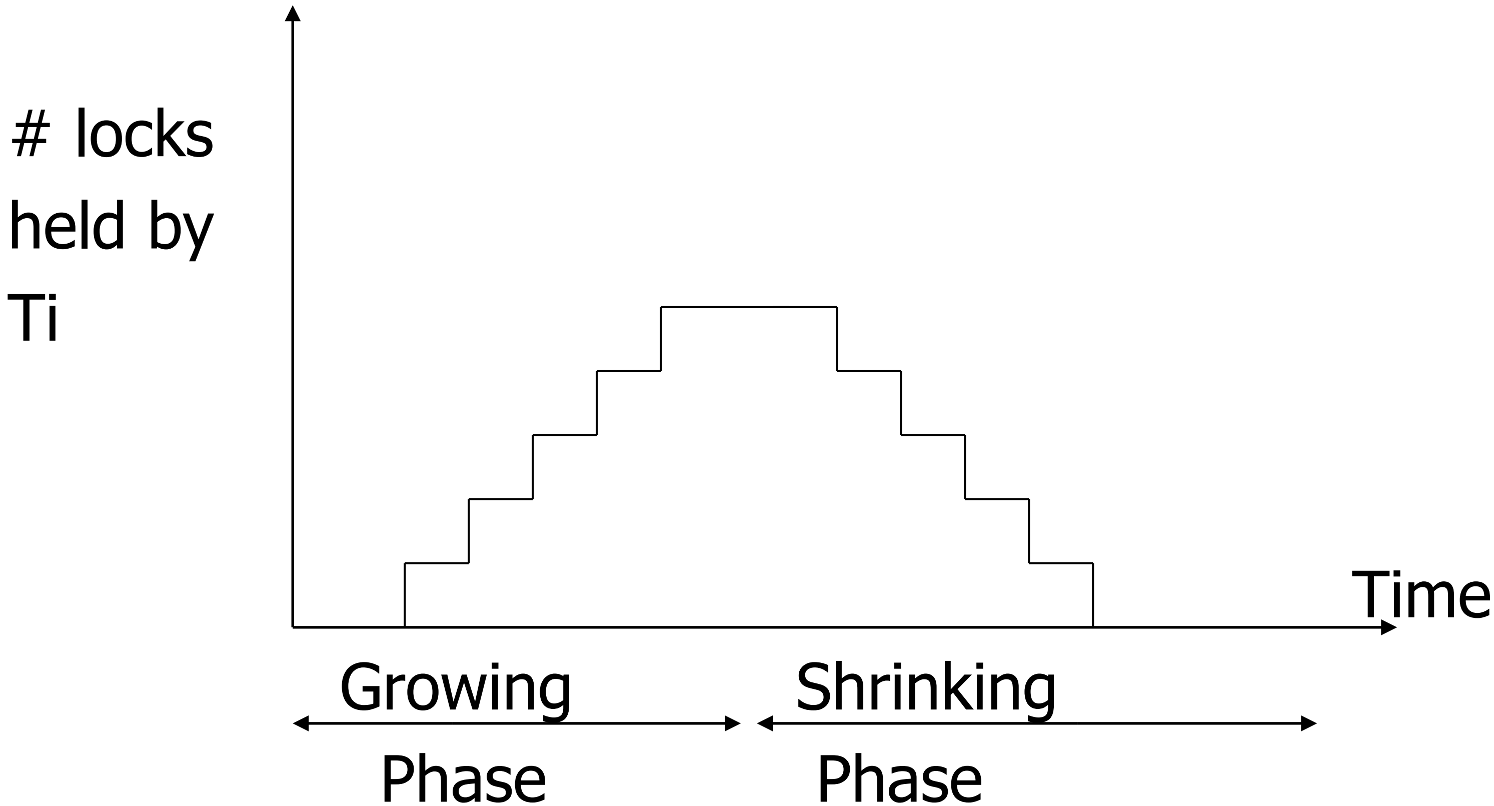
T1	T2
$l_1(A)$;Read(A)	
$A \leftarrow A+100$;Write(A);$u_1(A)$	
	$l_2(A)$;Read(A)
	$A \leftarrow Ax2$;Write(A);$u_2(A)$
	$l_2(B)$;Read(B)
	$B \leftarrow Bx2$;Write(B);$u_2(B)$
$l_1(B)$;Read(B)	
$B \leftarrow B+100$;Write(B);$u_1(B)$	

Two phase locking (2PL)

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

no unlocks

no locks


locks
held by
Ti
Time
Growing
Phase
Shrinking
Phase

2PL prevents Schedule F:

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

2PL prevents Schedule F:

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); $u_1(B)$	

2PL prevents Schedule F:

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

2PL may however result in deadlock:

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

Usual fix: deadlocked transactions are aborted and rolled back

Next step:

Show that 2 Phase Locking $\Rightarrow$ conflict-serializable schedules

Theorem 2PL $\Rightarrow$ conflict serializable schedule

To help in proof:

Definition Shrink(Ti) = SH(Ti) = first unlock action of Ti

Lemma

$Ti \rightarrow Tj$ in $S \Rightarrow SH(Ti) <_S SH(Tj)$

Proof of lemma:

$Ti \rightarrow Tj$ means that

$S = \ldots p_i(A) \ldots q_j(A) \ldots;$ p,q conflict

By rules 1,2:

$S = \ldots p_i(A) \ldots u_i(A) \ldots l_j(A) \ldots q_j(A) \ldots$

By 2PL: SH(Ti) SH(Tj)

So, $SH(Ti) <_S SH(Tj)$

Theorem 2PL $\Rightarrow$ conflict serializable schedule

Proof:

(1) Assume P(S) has cycle

$$T_1 \rightarrow T_2 \rightarrow .... T_n \rightarrow T_1$$

(2) By lemma: $SH(T_1) < SH(T_2) < \ldots < SH(T_1)$

(3) Impossible, so P(S) acyclic

(4) $\Rightarrow$ S is conflict serializable

How does locking work in practice?

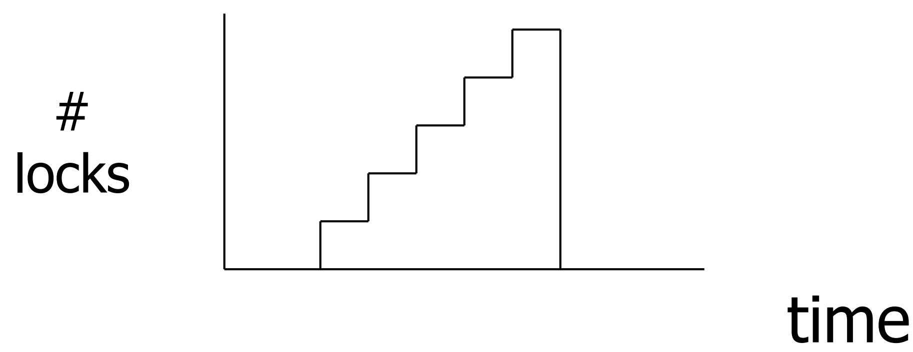
- Every system is different

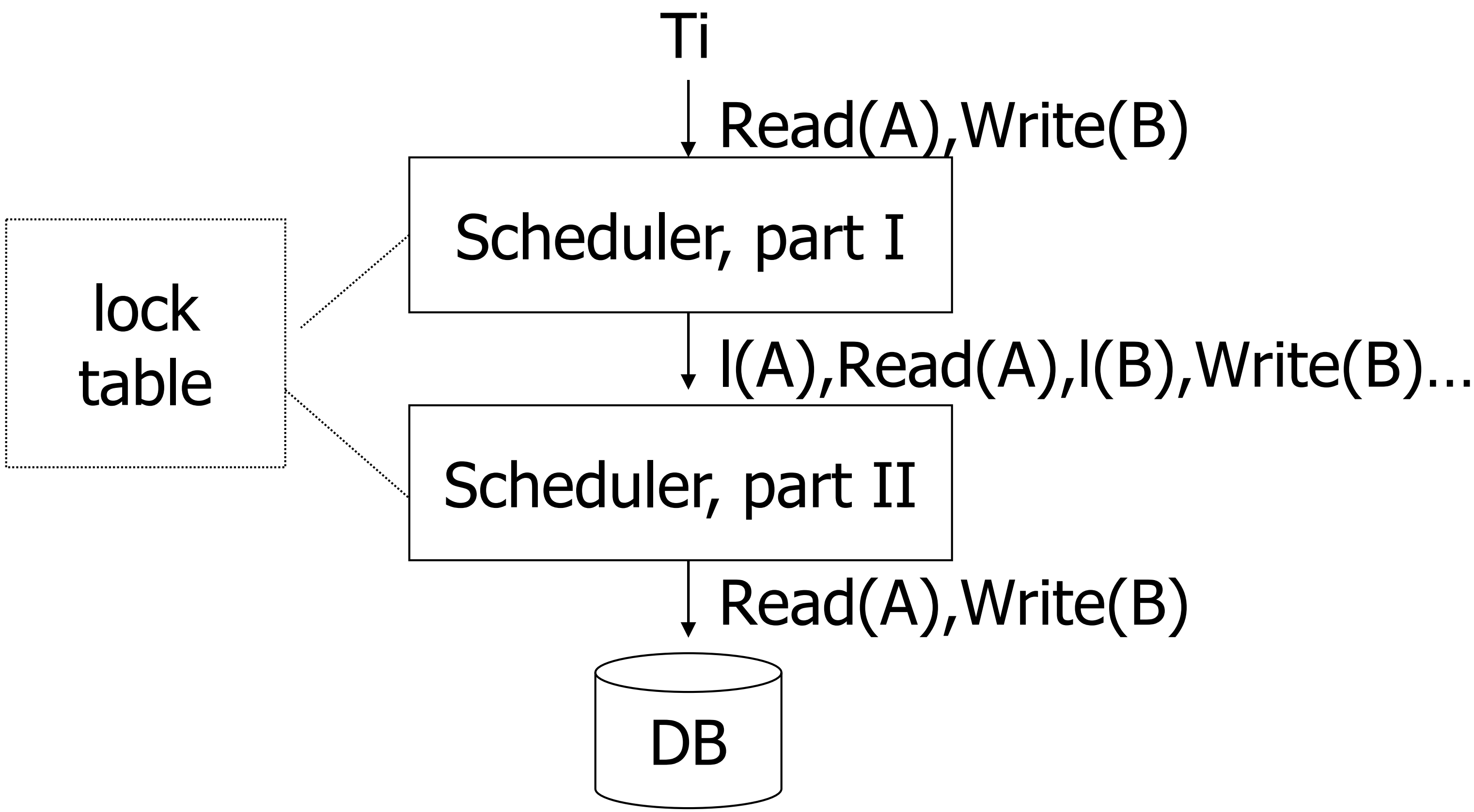
But here is one (simplified) way ...

Sample Locking System:

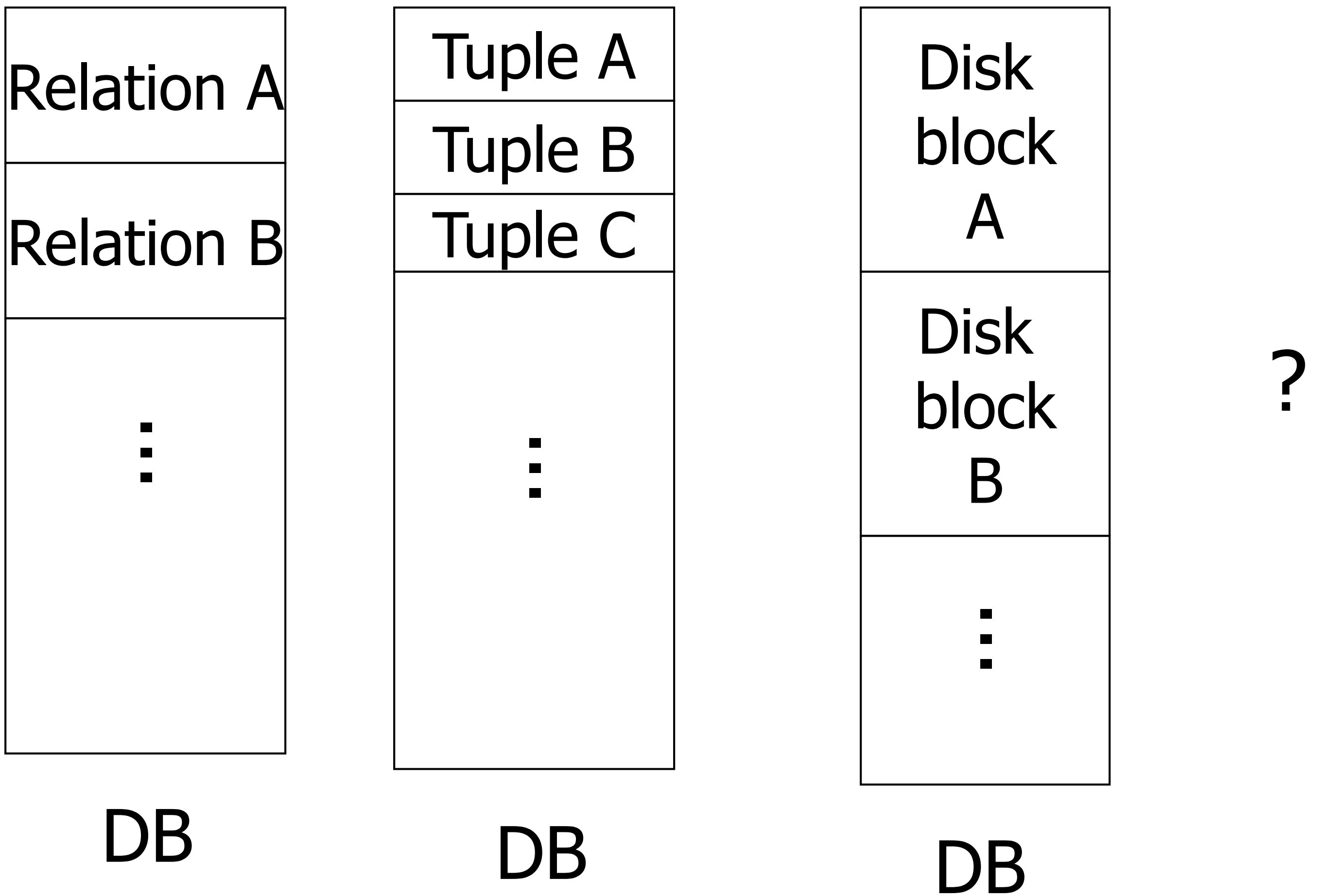
(1) Don't trust transactions to request/release locks

(2) Hold all locks until transaction commits




Ti
Read(A),Write(B)
Scheduler, part I
lock
table
l(A),Read(A),l(B),Write(B)…
Scheduler, part II
Read(A),Write(B)
DB

What are the objects we lock?



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

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

- Beyond this simple 2PL protocol, it is all a matter of improving performance and allowing more concurrency….
  - Shared locks
  - Multiple granularity
  - Inserts, deletes
  - Other types of C.C. mechanisms
  - Weaker guarantees (e.g. in the cloud)