

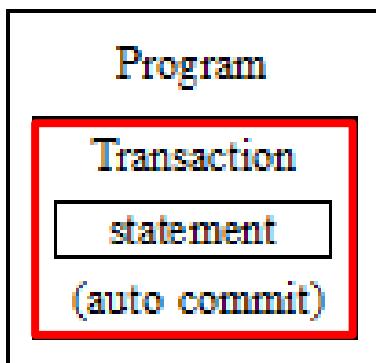
Transaction Management

DB programming style →
Ensure failure recovery

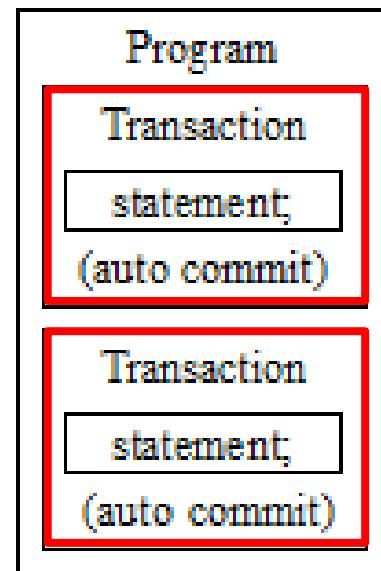
Transaction ?

- ✿ a special program unit embedded in DB applications.
- ✿ consist of many commands, just like any procedures or functions
- ✿ can be recovered from failure
 - IF all commands in a TS are not completed successfully.

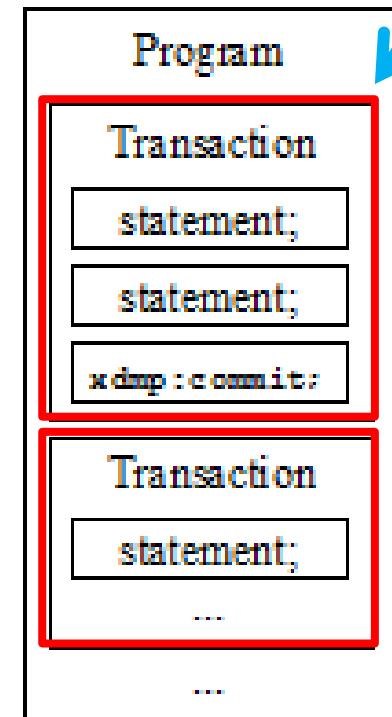
Program vs Transaction



Default model: A program containing one single statement, auto commit transaction.



Default model: A program containing multiple single statement transactions.



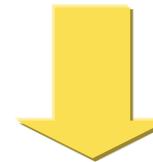
Multi-statement transactions:
A program containing multiple, multi-statement transactions.

- 1.Recovery process ?
- 2.Failure ?

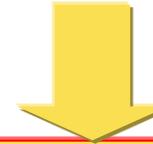
Withdraw Money



Enter PWD



Enter account Info



Withdraw 5000



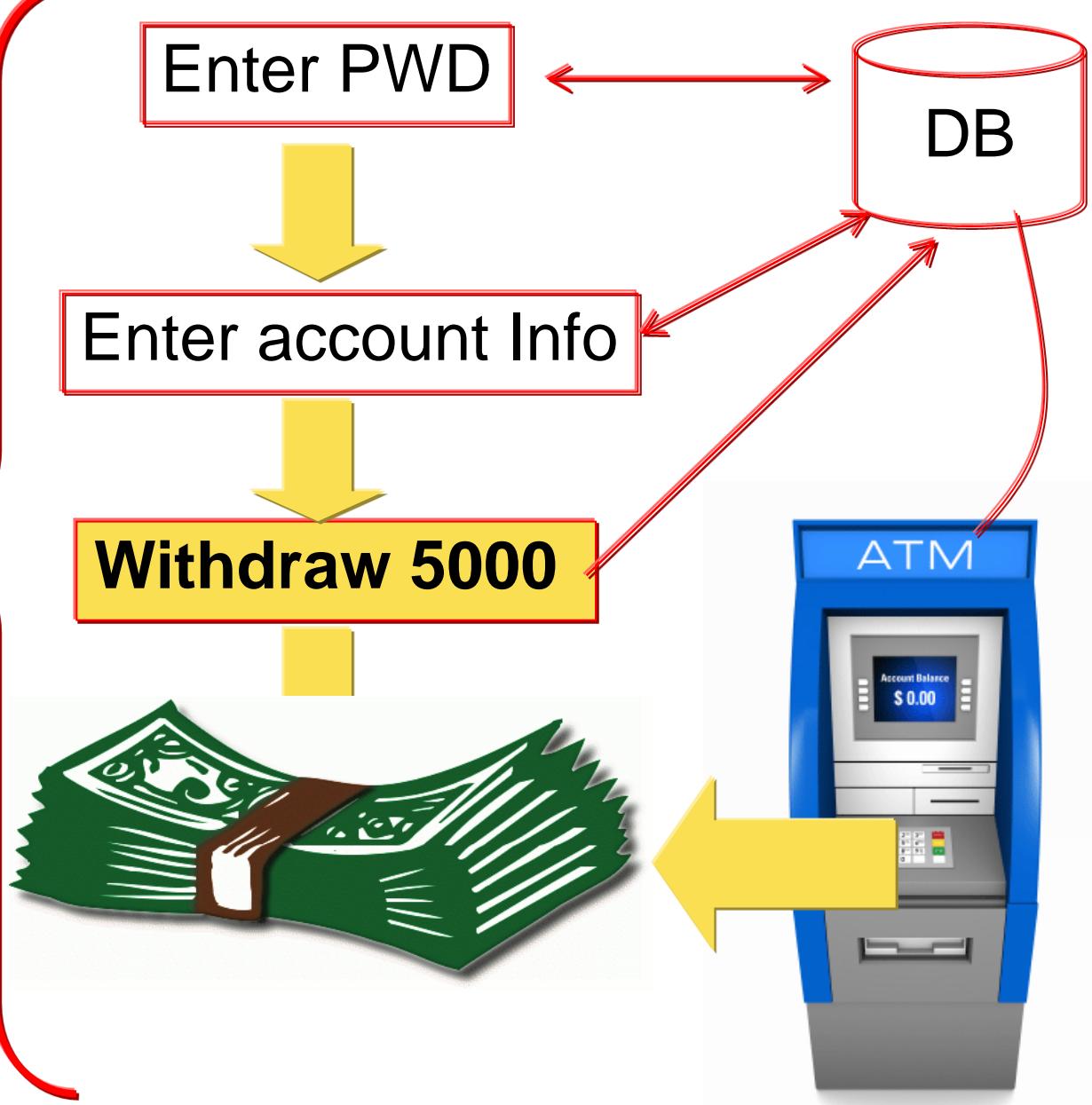
Failure

เงินไม่ออกมา

Balance updated ?



Transaction



Outline

- 1. Transaction basics**
- 2. Concurrency control**
- 3. Recovery management**
- 4. Transaction design issues**
- 5. Workflow management**

Transaction Definition

- ✿ Supports daily operations of an organization
- ✿ Become more important with the growth of the internet

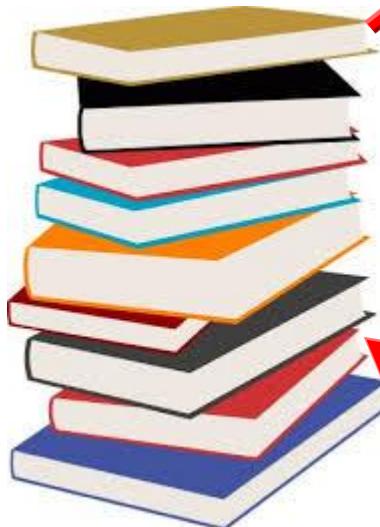


Transaction Definition

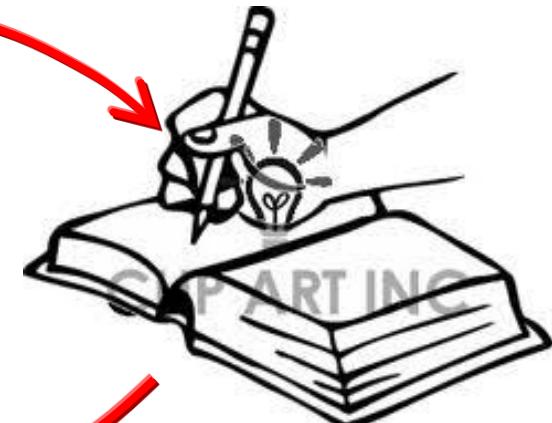
- ✿ Collection of database operations
- ✿ Reliably and efficiently processed as one unit of work
- ✿ No lost data
 - ✿ Interference among multiple users
 - ✿ Failures (operating system, program, disk, ...)

Library Example

10 copies of
a database book



1.borrow a book



3.return

4.Update all

Assume that a library allow people to borrow a book for updating its content! as well.

Library Example

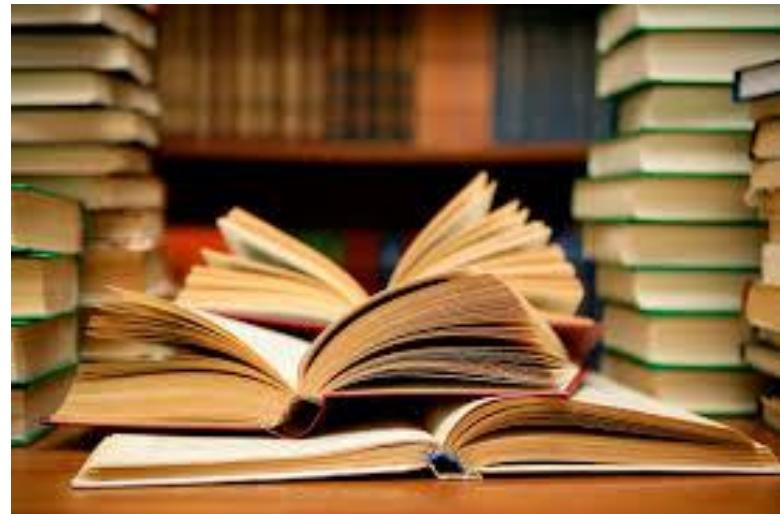
✿ Borrow for read only

✿ concurrently

✿ Borrow for update

✿ one person at a time

- ป้องกันการแก้สิ่งเดียวกัน



Airline Transaction Example

AirAsiaGo.com

หน้าแรก โรงแรม เที่ยวบิน + โรงแรม การเช่ารถ ข้อเสนอพิเศษ

ภาษาไทย | English

ไป-กลับ เที่ยวเดียว หลาย晚

กรุงเทพฯ - ดอนเมือง

โตเกียว

12/04/2018

17/04/2018

ค้นหา

ผู้เดินทาง 2 คน แสดงตัวเลือก

2. แสดงเที่ยวบินต่างๆ

เลือกเที่ยวบินขาไปโตเกียว

ราคาเป็นราคาไป-กลับต่อคน รวมภาษีและค่าธรรมเนียม

การ選擇 1 เริ่มที่:

สายพัสดุ 1 ครั้ง (5)

B18,788

สายการบิน 1 เริ่มที่:

แอร์เอเชียแอร์ (5)

B18,788

แอร์เอเชีย (4)

B18,788

ไทยแอร์เอเชีย (1)

B23,090

เวลาที่ออกเดินทาง - กรุงเทพฯ

ช่วงเช้า (05:00 น. - 11:59 น.)

ช่วงค่ำ (18:00 น. - 23:59 น.)

เวลาที่เดินทางถึง - โตเกียว

ช่วงค่ำ (18:00 น. - 23:59 น.)

20:20 น. - 22:30 น. +1

24ชม. 10น. (สายพัสดุ 1 ครั้ง) ☎

B18,788

หลายสายการบิน
ด้วย (3/10)

DMK - 14ชม. 55น. ที่ KUL - HND

ไป-กลับ

อาจมีค่าธรรมเนียมเพิ่มเติม ขึ้นอยู่กับวิธีการชำระเงินของคุณ ค่าธรรมเนียมจากสายการบินยังไม่รวมอยู่ในราคาตั๋ว

รายละเอียดเที่ยวบิน ✅ ไม่รวมส่วนการเช็คอิน

18:35 น. - 22:30 น. +1

25ชม. 55น. (สายพัสดุ 1 ครั้ง) ☎

B18,788

หลายสายการบิน
ด้วย (3/10)

DMK - 16ชม. 35น. ที่ KUL - HND

ไป-กลับ

อาจมีค่าธรรมเนียมเพิ่มเติม ขึ้นอยู่กับวิธีการชำระเงินของคุณ ค่าธรรมเนียมจากสายการบินยังไม่รวมอยู่ในราคาตั๋ว

รายละเอียดเที่ยวบิน ✅ ไม่รวมส่วนการเช็คอิน

23:10 น. - 22:30 น. +1

21ชม. 20น. (สายพัสดุ 1 ครั้ง) ☎

B19,788

หลายสายการบิน
ด้วย (3/10)

DMK - 12ชม. 5น. ที่ KUL - HND

ไป-กลับ

1. เลือกเมืองต้นทาง ปลายทาง และวันที่

Creating a transaction by SQL statements:

➤ START TRANSACTION

- เป็น key words กำหนดจุดเริ่มต้น

➤ COMMIT: คำสั่งสิ้นสุด transaction

คำสั่งอื่นๆเพื่อสื่อสารกับ a DBMS

✿ ROLLBACK สั่งให้ระบบ

✿ คือยกเลิกการกระทำการที่มีผล

✿ กรณีมี error เราจะใช้คำสั่งนี้เพื่อให้ระบบลบผลกระทบที่มีต่อฐานข้อมูล

✿ On Error: part of exception handling

Airline Transaction Example

START TRANSACTION

Display greeting

Get reservation preferences from user

- SELECT departure and return flight records
 - If reservation is acceptable then
 - UPDATE seats remaining of departure flight record
 - UPDATE seats remaining of return flight record
 - INSERT reservation record
 - Print ticket if requested
 - End If
 - On Error: ROLLBACK
- COMMIT**

If fail?

ATM Transaction Example

START TRANSACTION

Display greeting

Get account number,pin,type,amount

SELECT account number,type,balance

If **balance is sufficient** then

 UPDATE account by posting debit

 INSERT history record

 Display message and **dispense cash**

 UPDATE history record

 Print receipt if requested

End If

On Error: **ROLLBACK**

COMMIT

If fail?

Transaction Properties (ACID)

- ✿ **Atomic** ความเป็น unit เดียวกัน
- ✿ **Consistent** การรัน T ข้อมูลถูกต้องเสมอ
- ✿ **Isolated** แต่ละ T เป็นอิสระ ไม่เกี่ยวเนื่องกัน
- ✿ **Durable** ผลต่อฐานข้อมูลของ T ที่สำเร็จแล้วจะอยู่那儿 ถึงแม้ระบบจะ fails หลังจากนั้น และ recovery ได้

Transaction Properties (ACID)

✿ **Atomic:** all or nothing

✿ all changes are made or none are made

✿ **Consistent:** database must be consistent before and after a transaction

✿ by the time any transaction ends, each and every reference in the database must be valid.

example

Transaction Properties (ACID)

✿ Enforce referential integrity:

✿ If a transaction consisted of an attempt to delete a record referenced by another, each of the followings would maintain consistency:

- **abort** the transaction, rolling back to the consistent, prior state; or,
- **delete ALL records** that reference the deleted record (*cascade delete*); or,
- **nullify the relevant fields** in **ALL records** that point to the deleted record.

propagation
constraints

OfferNo	CourseNo	OffTerm	OffYear	OffLocation	OffTime	FacSSN	OffDays
1111	IS320	SUMMER	2006	BLM302	10:30:00		MW
1234	IS320	FALL	2005	BLM302			MW
2222	IS460	SUMMER	2005	BLM412			TH
3333	IS320	SPRING	2006	BLM214			MW
4321	IS320	FALL	2005	BLM214			TH
4444	IS320	WINTER	2006	BLM302	1	543-21-0987	TTH
5555	FIN300	WINTER	2006	BLM207	0	765-43-2109	MW
5678	IS480	WINTER	2006	BLM207	0	007-65-4221	MW
5679	IS480	SPRING	2006				
6666	FIN450	WINTER	2006				
7777	FIN480	SPRING	2006				
8888	IS320	SUMMER	2006				
9876	IS460	SPRING	2006				

Transaction:
delete IS320

Offering

Course

CourseNo	CrsDesc	CrsUnits
FIN300	FUNDAMENTALS OF FINANCE	4
FIN450	PRINCIPLES OF INVESTMENTS	4
FIN480	CORPORATE FINANCE	4
IS320	FUNDAMENTALS OF BUSINESS PROGRAMMING	4
IS460	SYSTEMS ANALYSIS	4
IS470	BUSINESS DATA COMMUNICATIONS	4
IS480	FUNDAMENTALS OF DATABASE MANAGEMENT	4

Transaction Properties(ACID)

- ✿ **Isolated:** no unwanted interference from other *users*
- ✿ requirement that other operations cannot access data that has been modified during a transaction that has not yet completed
- ✿ In case of concurrent transactions

Transaction Properties(ACID)

✿ Durable:

✿ the ability of the DBMS to **recover the committed transaction** updates against any kind of system failure (HW or SW)

- the data changes will survive system failure, and
- all integrity constraints have been satisfied so the DBMS won't need to reverse the transaction.

✿ implemented by writing transactions into a transaction log

Transaction Processing Services

1. Concurrency control
2. Recovery management
3. Service characteristics

✿ **Transparent**

- Application programmer does not write code except SQL statements

✿ **Consume significant resources**

✿ **Significant cost component**

✿ **Transaction design** important เกี่ยวกับ Programmers

Concurrency Control

✿ **Problem definition**

✿ **Concurrency control problems**

✿ **Concurrency control tools**

Concurrency Control Problem

✿ Objective:

✿ ให้ระบบทำงานได้มากที่สุด

✿ Throughput: จำนวน T ที่ทำงานสำเร็จต่อนาที

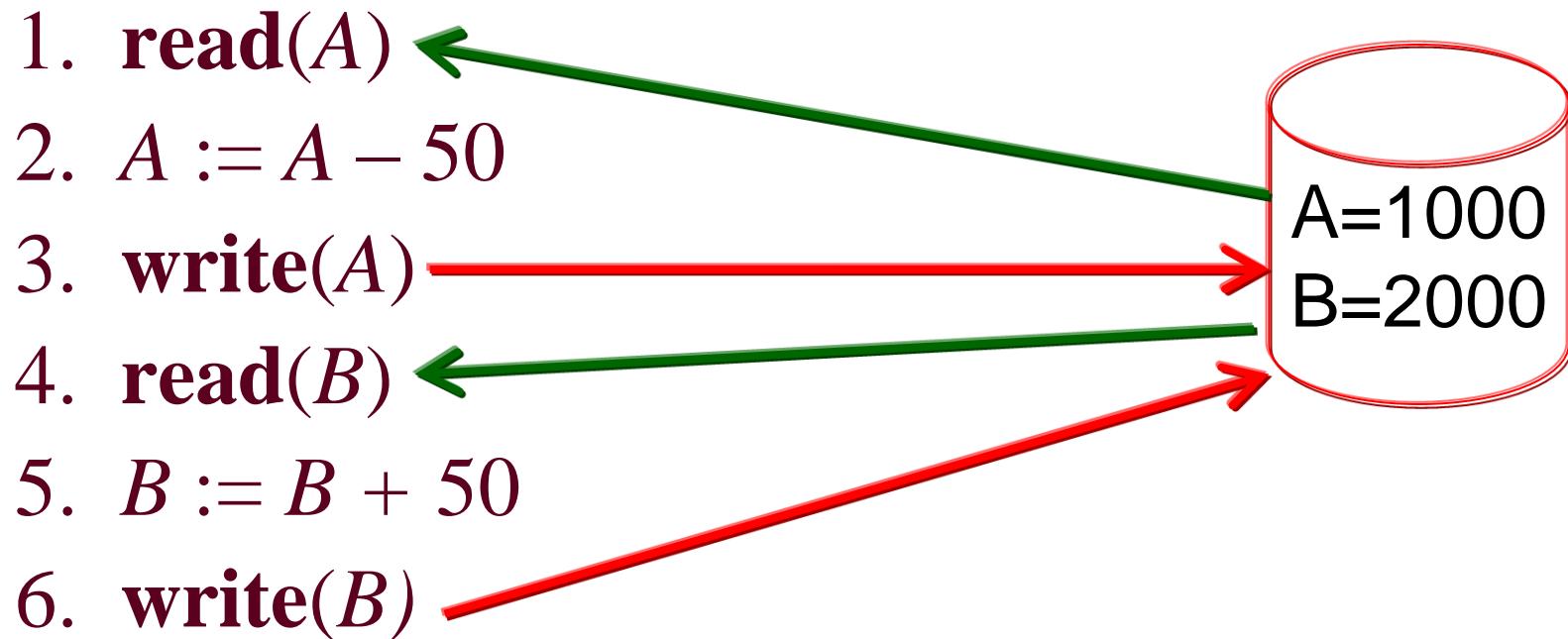
- Large organizations: thousands of transactions per minute

✿ จัด schedules การรัน T ให้มีลักษณะเป็น Serializability.

- **Serializability** of a schedule คือ schedule ของการรันหลาย T ที่อาจทำไปพร้อมๆกันแต่มีผลต่อฐานข้อมูลเหมือนกับ schedule ของการรัน T ต่างๆ ตามลำดับ โดยไม่มีการทำงานไปพร้อมๆกัน)

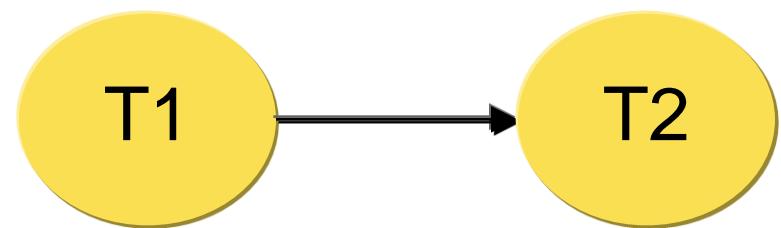
Example of Fund Transfer

Transaction to transfer \$50 from account A to account B :



Serial Schedule

T_1	T_2
read(A) $A := A - 50$ write (A) read(B) $B := B + 50$ write(B)	read(A) $temp := A * 0.1$ $A := A - temp$ write(A) read(B) $B := B + temp$ write(B)



รัน T_1 เสร็จจึงรัน T_2

Serial Schedule

Serializable Schedule

S1

T1	T2
read(A) $A := A - 50$ write(A)	read(A) $temp := A * 0.1$ $A := A - temp$ write(A)
read(B) $B := B + 50$ write(B)	read(B) $B := B + temp$ write(B)

S2

T1	T2
read(A) $A := A - 50$ write(A)	read(A) $temp := A * 0.1$ $A := A - temp$ write(A)
read(B) $B := B + 50$ write(B)	read(B) $B := B + temp$ write(B)
	read(B) $B := B + temp$ write(B)

Effects of T1 and T2 are the same

Concurrency Control Problem

✿ Constraint:

✿ No interference:

- effect is the same as **serial effect** (1-by-1 execution)
- slows down processing

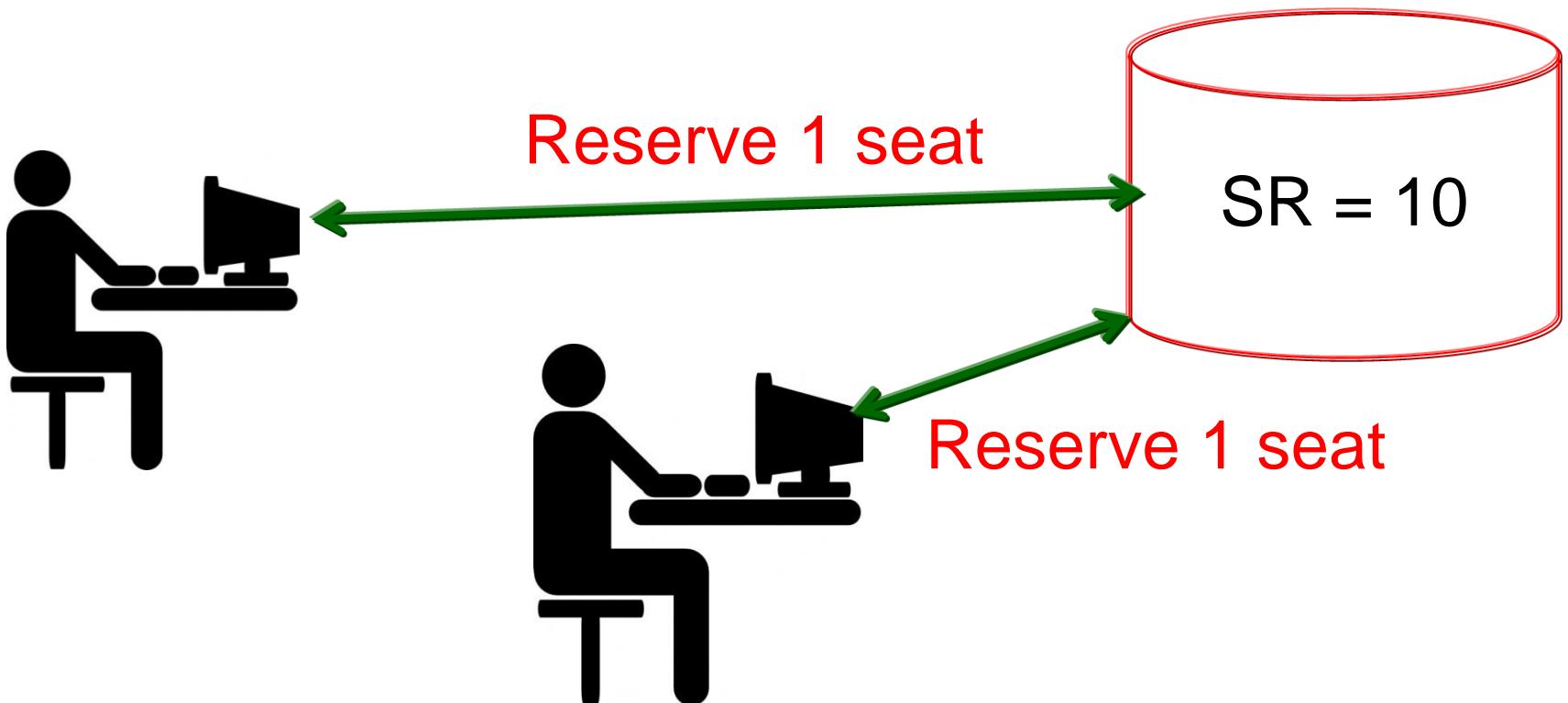
✿ Commonly manipulated data known as hot spots

- Example: seats remaining in a flight record

✿ Concurrency control must be applied to the entire database, not just the hot spots

Lost Update Problem

Two users trying to **reserve the same flight**:
need to change seats remaining (SR) field



Lost Update Problem

Two users changing the same part of the database

Transaction A	Time	Transaction B
Read SR (10)	T ₁	
	T ₂	Read SR (10)
If SR > 0 then SR = SR - 1	T ₃	
	T ₄	If SR > 0 then SR = SR - 1
Write SR (9)	T ₅	
	T ₆	Write SR (9)

SR: Seat Reserved column name

Write-Write problem

Uncommitted Dependency Problem

Transaction A	Time	Transaction B
Read SR (10)	T ₁	
SR = SR - 1	T ₂	
Write SR (9)	T ₃	
	T ₄	Read SR (9)
ROLLBACK ★	T ₅	

failure

the write-read problem

Inconsistent Retrieval Problems

✿ การรันหลายๆ T ที่เกี่ยวเนื่องกัน(Interference) พร้อมกัน ก่อให้เกิดปัญหาการอ่านที่ไม่ถูกต้องตรงกัน (inconsistency)

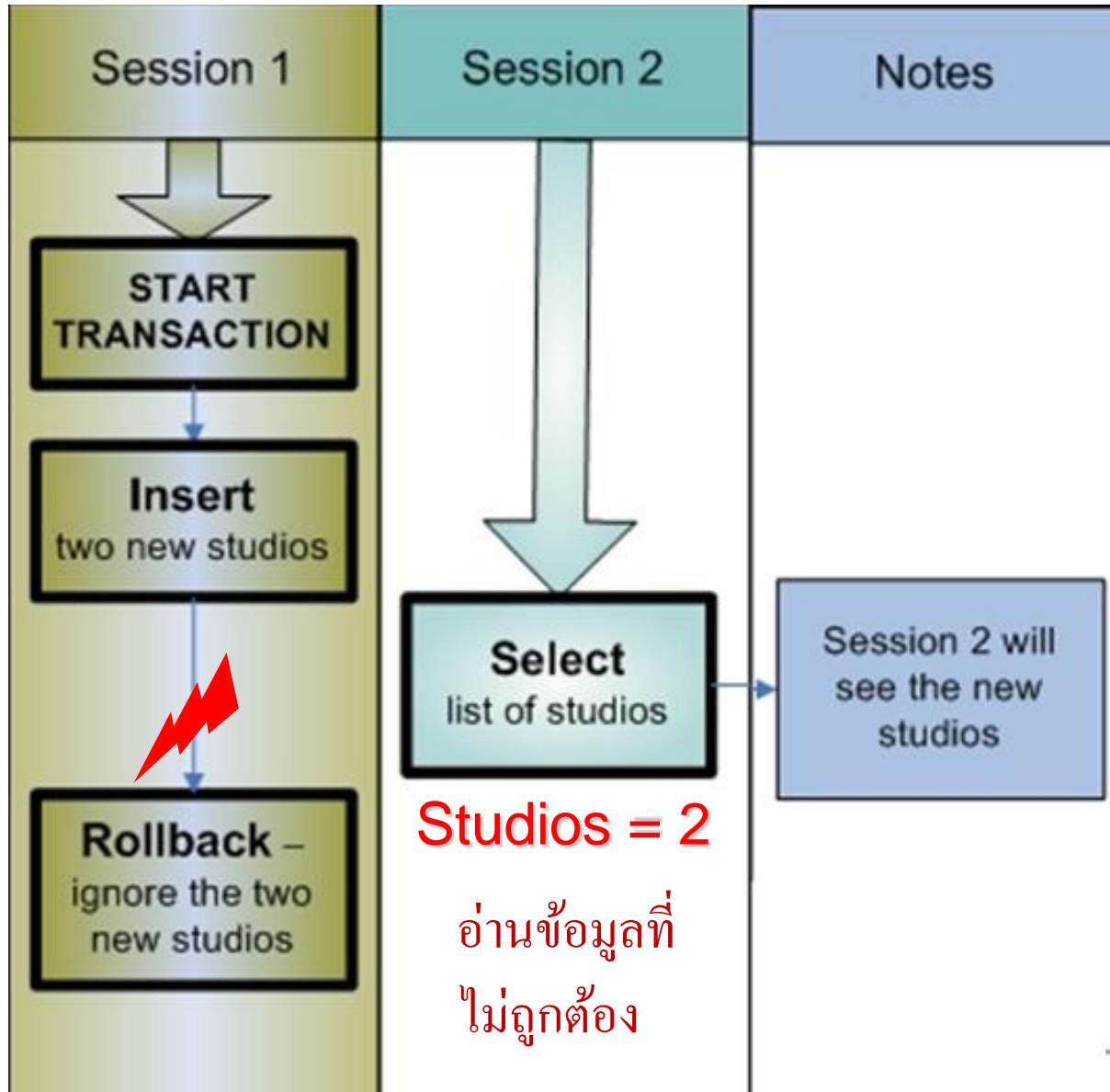
1. Phantom read
2. Non repeatable read
3. Incorrect summary

Phantom Read

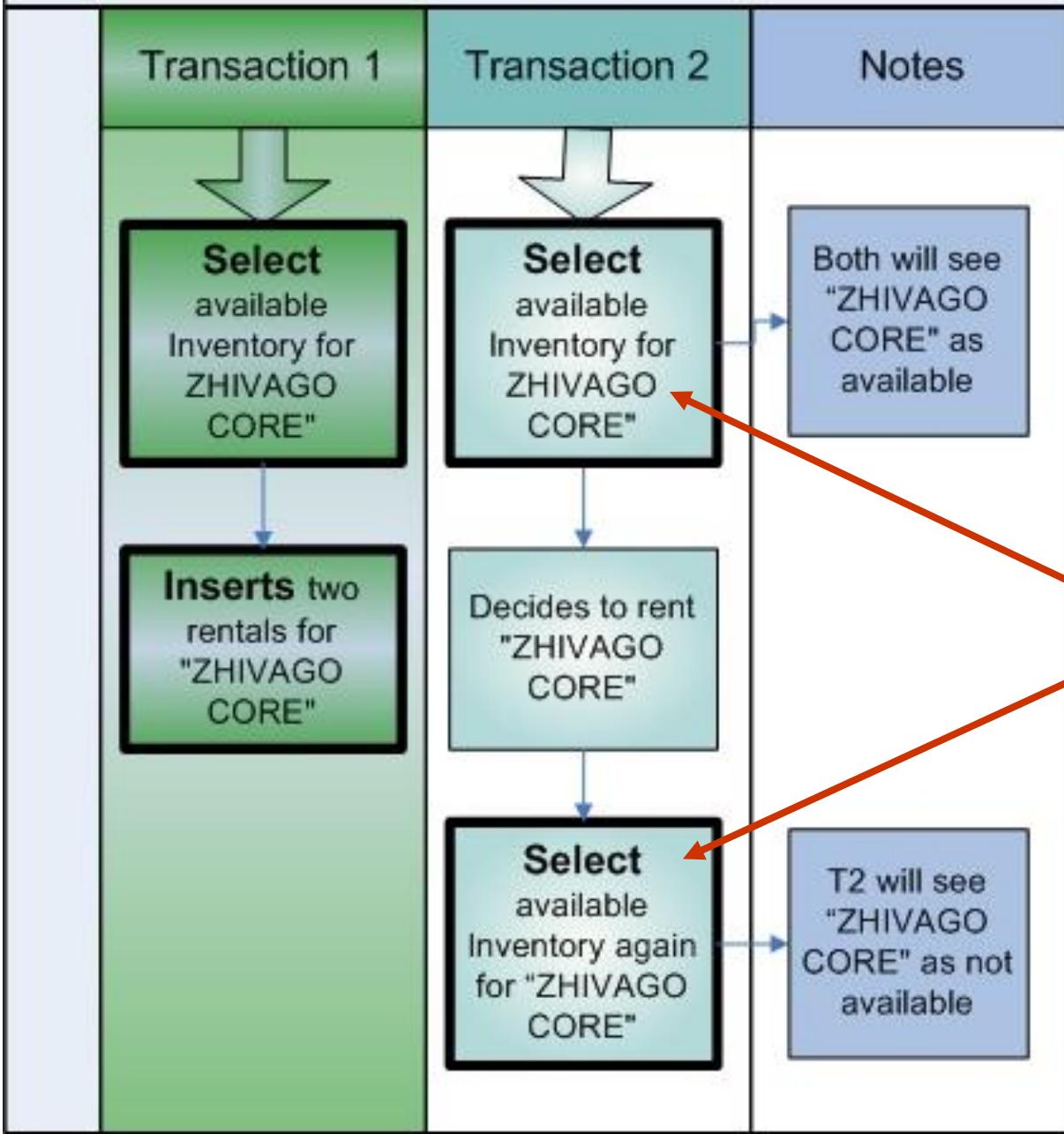
Studios=0

Studios=2

Studios=0



Non-Repeatable Reads



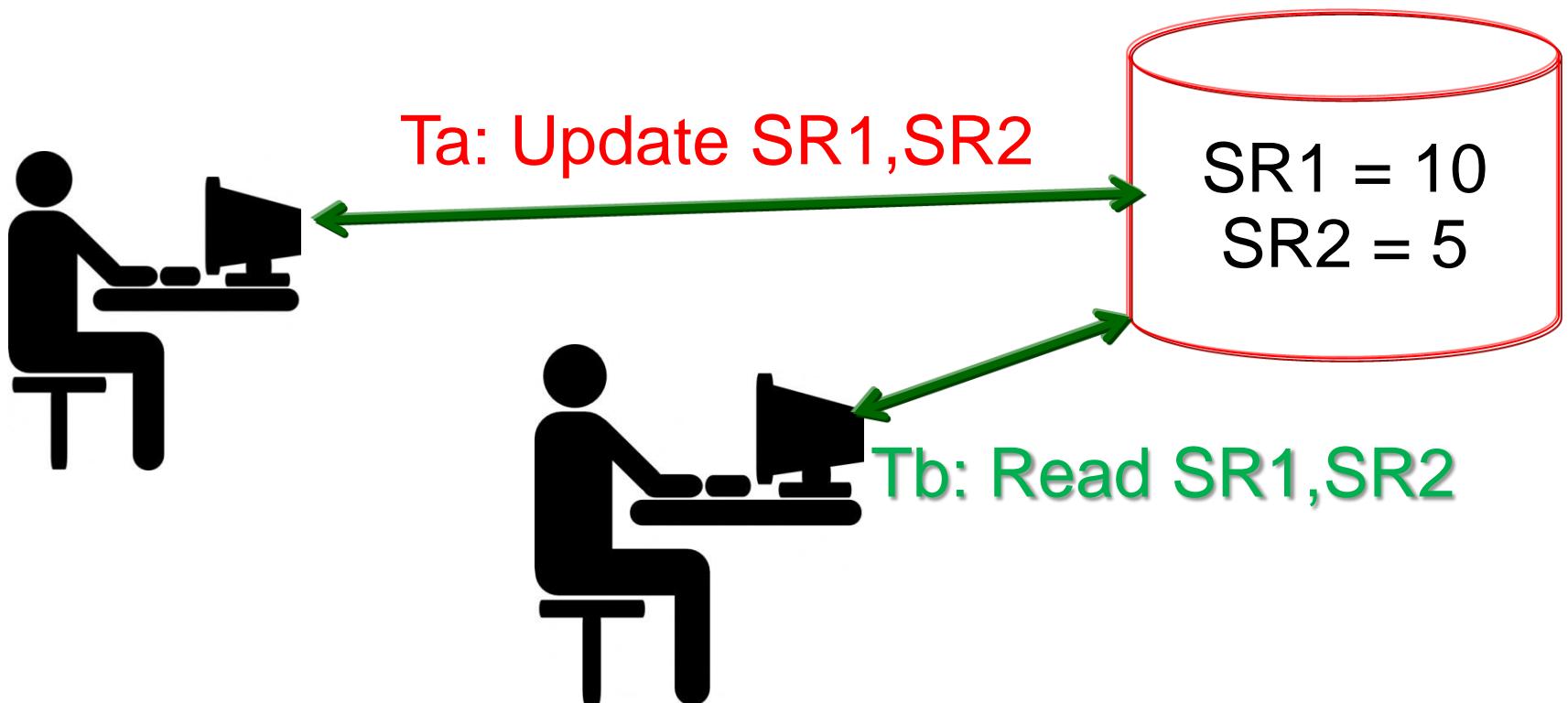
ปัญหาแบบนี้เกิดเมื่อ T มี
การอ่านข้อมูลเดิม มากกว่า
1 ครั้ง

Non-repeatable read

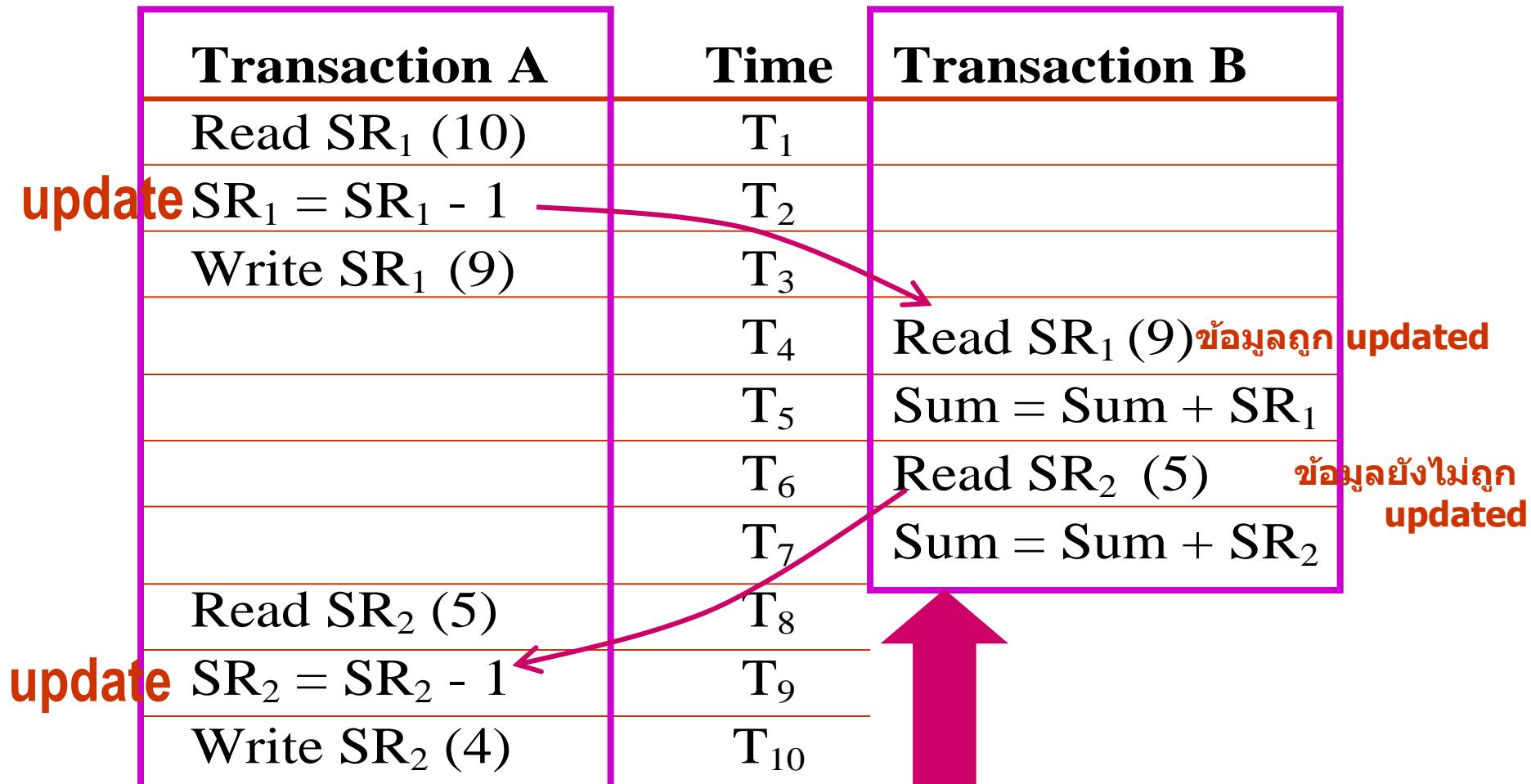
ใน T2 มีคำสั่งอ่านข้อมูล
ห้องว่าง 2 ครั้ง
แต่ได้ข้อมูลไม่ตรงกัน
นั่นคือการจัด schedule
แบบนี้ไม่ดี

Incorrect Summary Problem

Interference causes inconsistency



Incorrect Summary Problem



TB ต้องการหาค่าเฉลี่ยของ SR1 & SR2 แต่อ่านข้อมูลที่ update บ้างไม่ update บ้าง

HOW TO solve concurrency problems ?

Locking Fundamentals

- ✿ ใช้แก้ปัญหาทรานแซคชันที่เกี่ยวเนื่องกัน (interference)
 - ✿ ใช้ข้อมูลตัวเดียว กัน
- ✿ หลักการคือในโปรแกรมต้องเขียนคำสั่ง
 - ✿ ขอ lock ข้อมูล ก่อนคำสั่งใช้งานข้อมูล
 - ✿ ปลด lock ข้อมูลหลังใช้งานข้อมูลเสร็จ
- ✿ ระบบจะให้รอ Wait ถ้าเกิด conflicting lock
- ✿ โปรแกรม concurrency control manager จะจัดการกับ lock table ซึ่งบันทึกการล็อกข้อมูลต่างๆ

Locking data

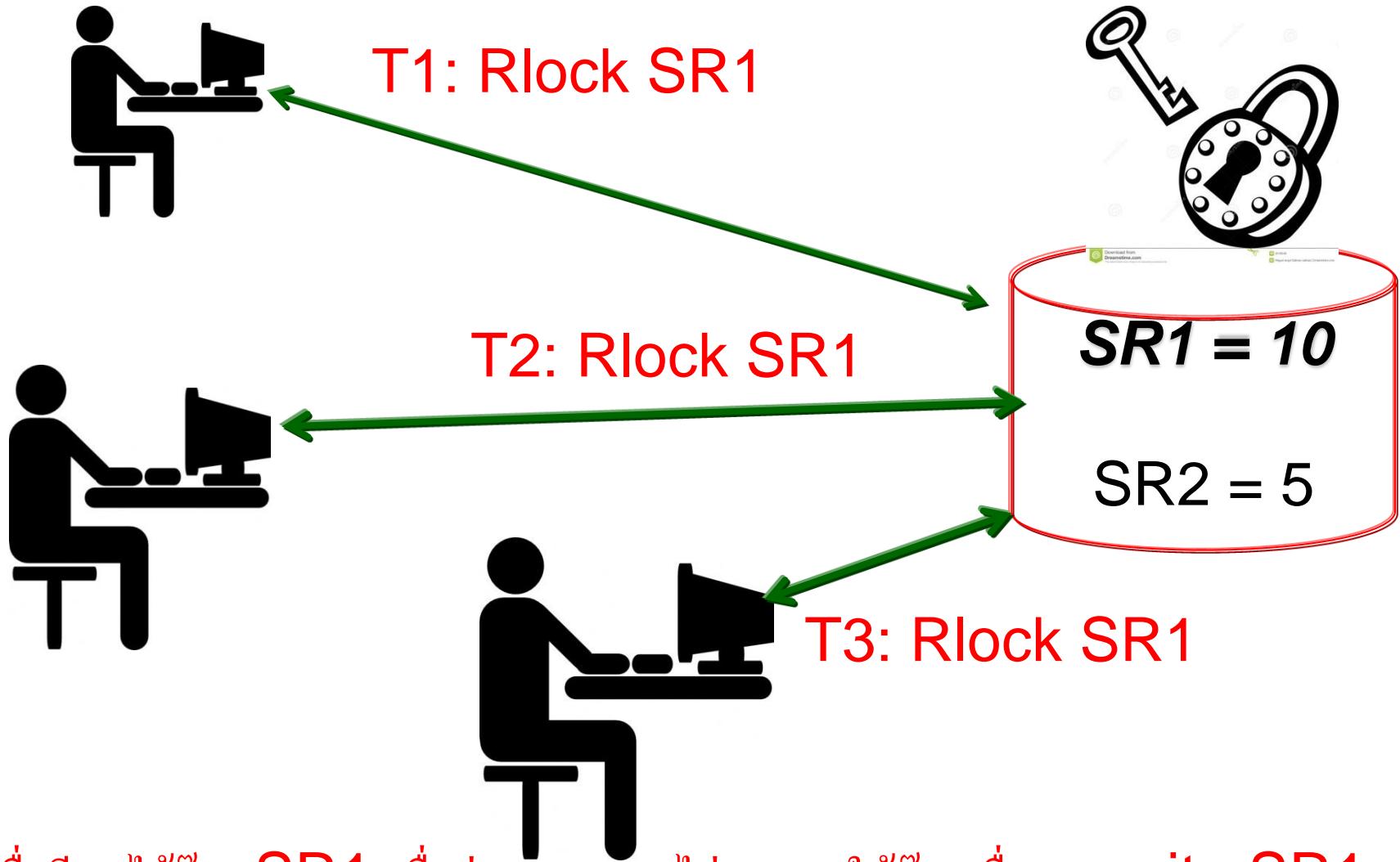
การล็อกมีหลายประเภท เพื่อให้ระบบทำงานเร็ว

✿ **Shared lock:** ล็อกข้อมูลเพื่อการอ่าน จึงล็อกได้พร้อมๆ กันโดยหมาย T

✿ **Exclusive lock:** ล็อกเพื่อการ write ข้อมูล
✿ ไม่อนุญาตให้ล็อกข้อมูล ตัวเดียวกัน ได้พร้อมๆ กัน



Shared lock: a read lock



เมื่อมีคนได้ล็อก SR1 เพื่ออ่าน ระบบจะไม่อนุญาตให้ล็อกเพื่อการ write SR1

Locking Fundamentals

✿ Conflicting locks

✿ **Shared lock:** conflicts with exclusive locks:

- เมื่อข้อมูลถูก **Read Lock** ระบบจะไม่ให้ **Write lock** พร้อมๆกัน นั่นคือต้องรอ

✿ **Exclusive lock:** conflicts with all other kinds of locks (**Write lock/Read lock**):

- เมื่อข้อมูลถูก **Write Lock** ระบบจะไม่ให้ T อื่นใดได้ lock ข้อมูลไม่ว่าแบบใดก็ตาม นั่นคือ T อื่นๆ ต้องรอนกว่าจะมีการปลดล็อก

Locking Fundamentals

✿ Lock usage: เวลาเขียนโปรแกรม

✿ Obtain appropriate kind of lock before accessing database item

✿ Wait if another transaction holds a conflicting lock

✿ Lock table: ใช้โดยระบบ

✿ Details about locks held by transactions

✿ Concurrency control manager maintains

✿ Lock: insert a lock record

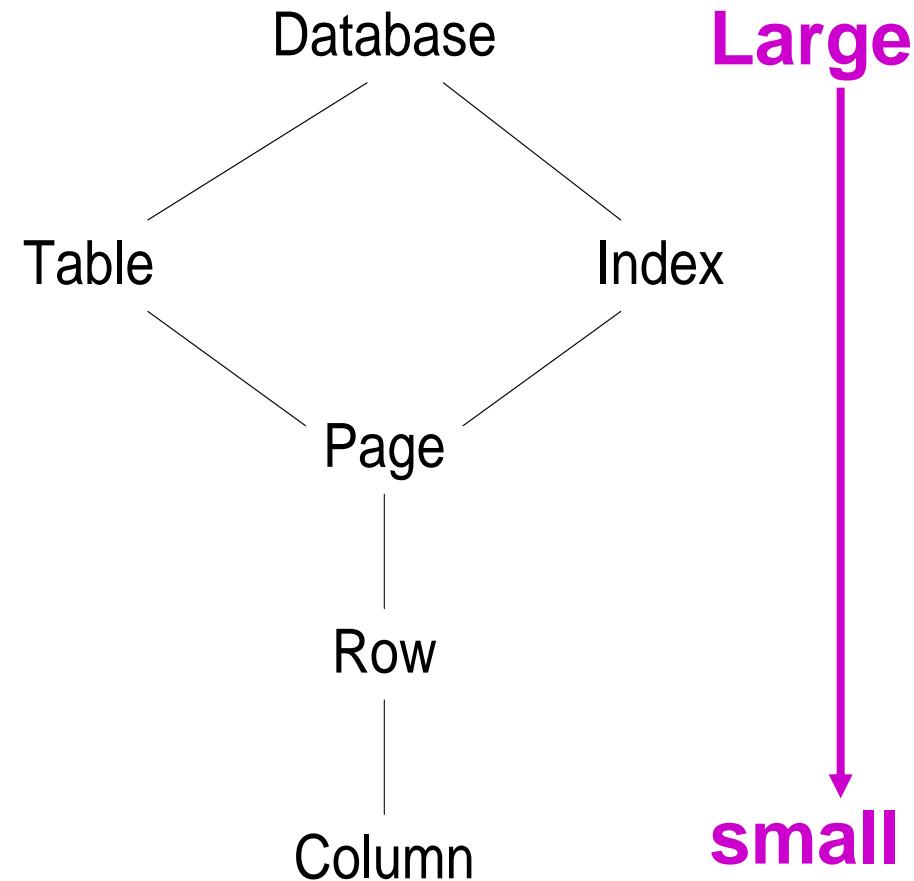
✿ Release (unlock): delete a lock record

Locking Granularity

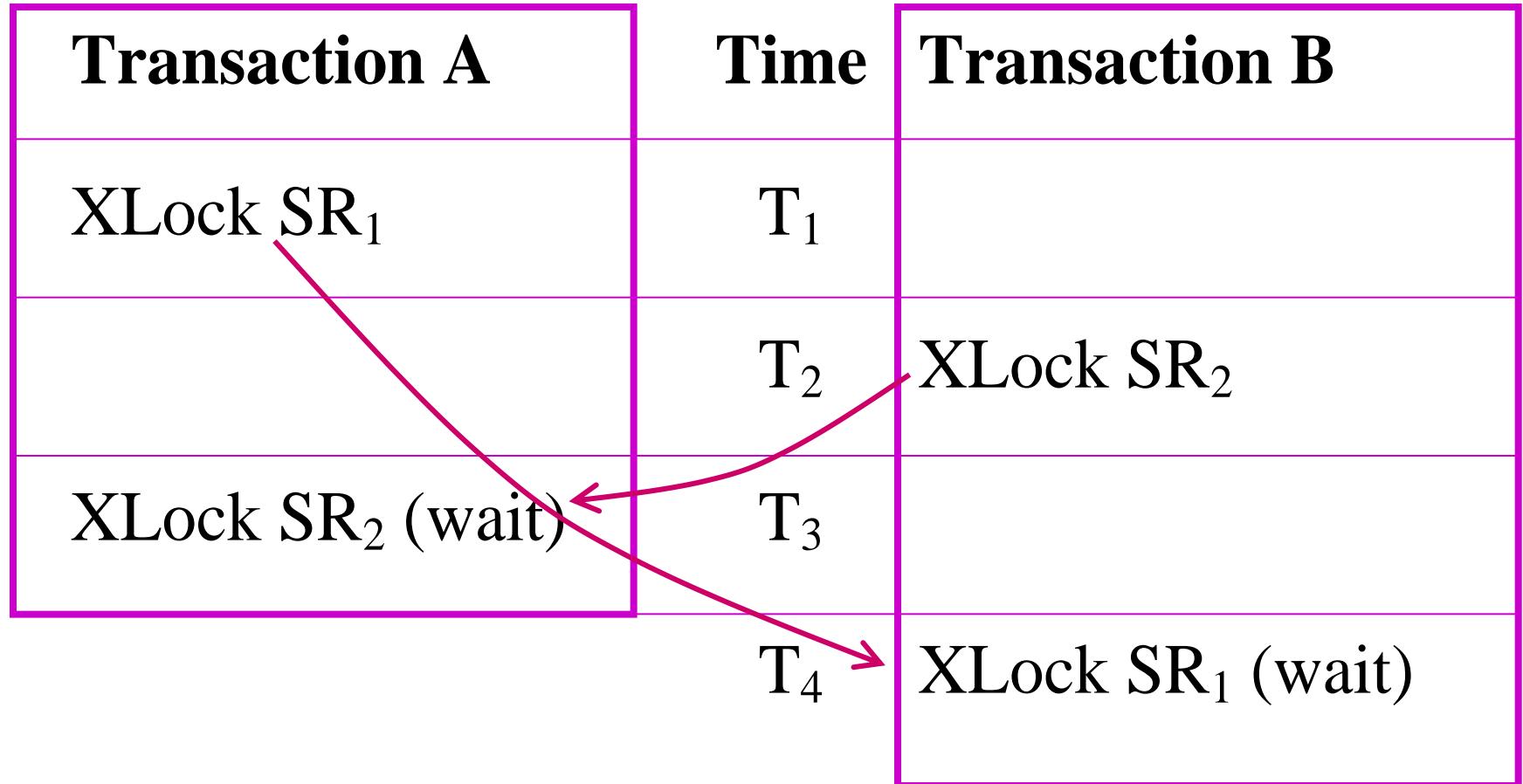
(size of database item locked)

Tradeoff:

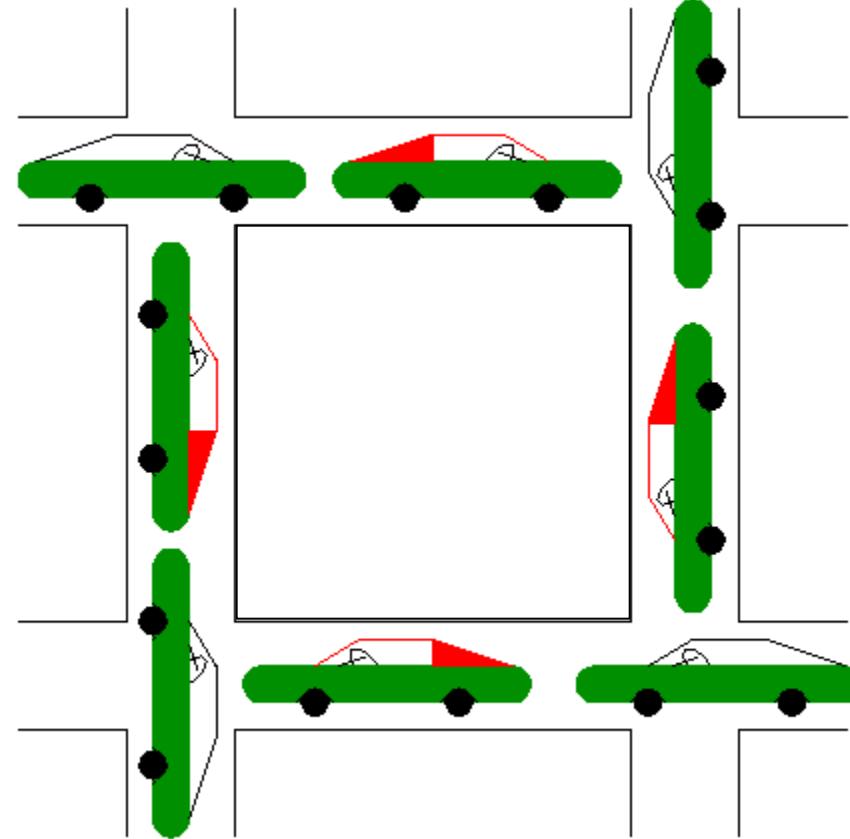
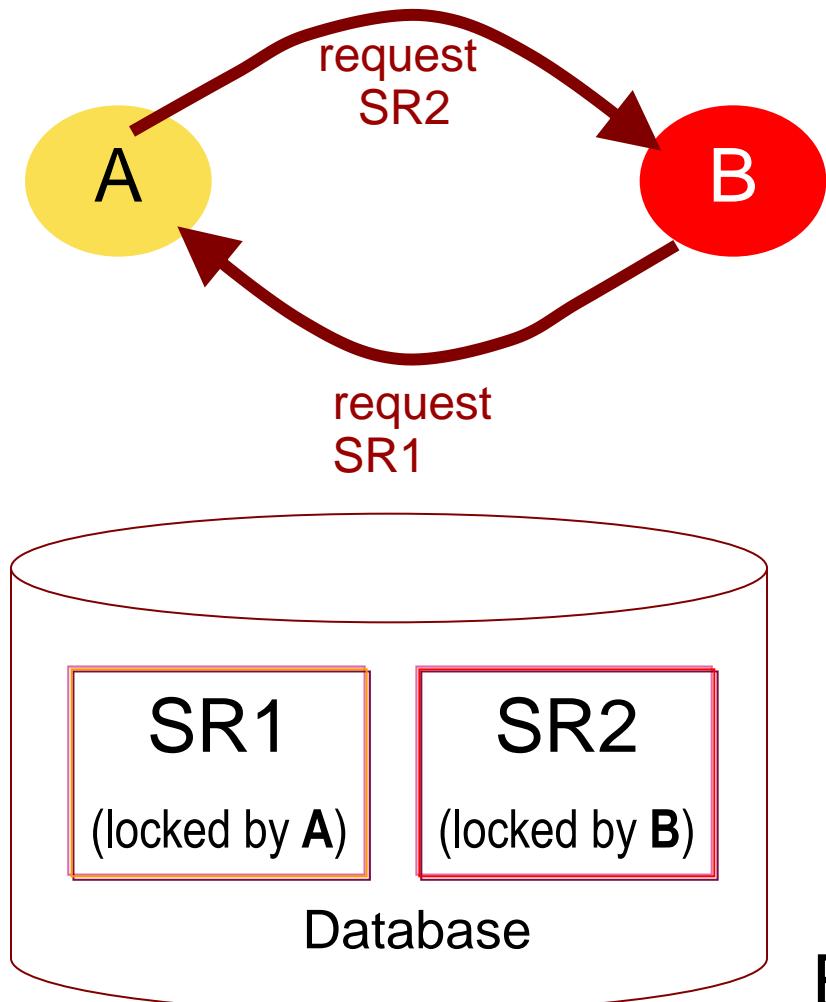
Overhead (#of locks)
versus waiting



Deadlock (Mutual Waiting)



Deadlock (Mutual Waiting)



Both Ta,Tb want to access
SR1,SR2

Deadlock Resolution

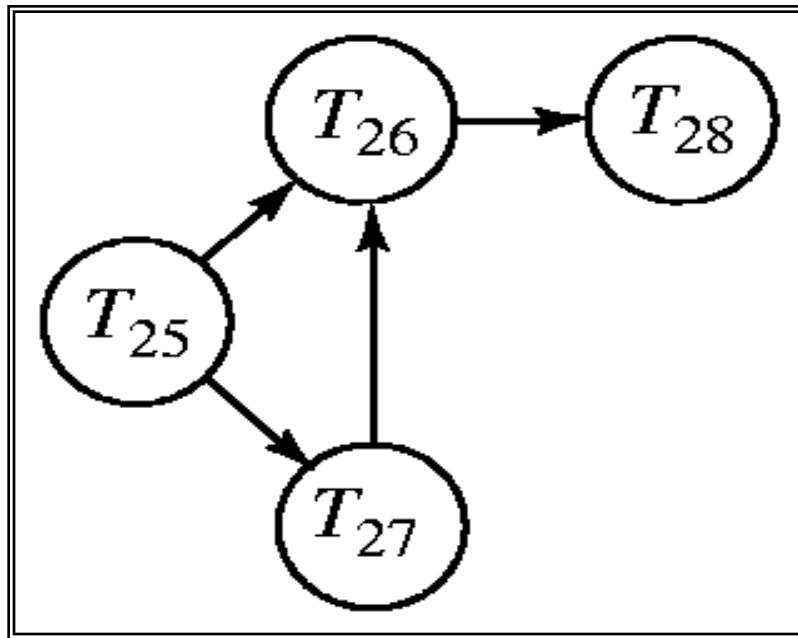
✿ Detection

- ✿ Create a graph showing waiting dependencies
 - **Search the graph for cycles**
- ✿ Empirically, most deadlocks involve 2 or 3 transactions
- ✿ Used by enterprise DBMSs

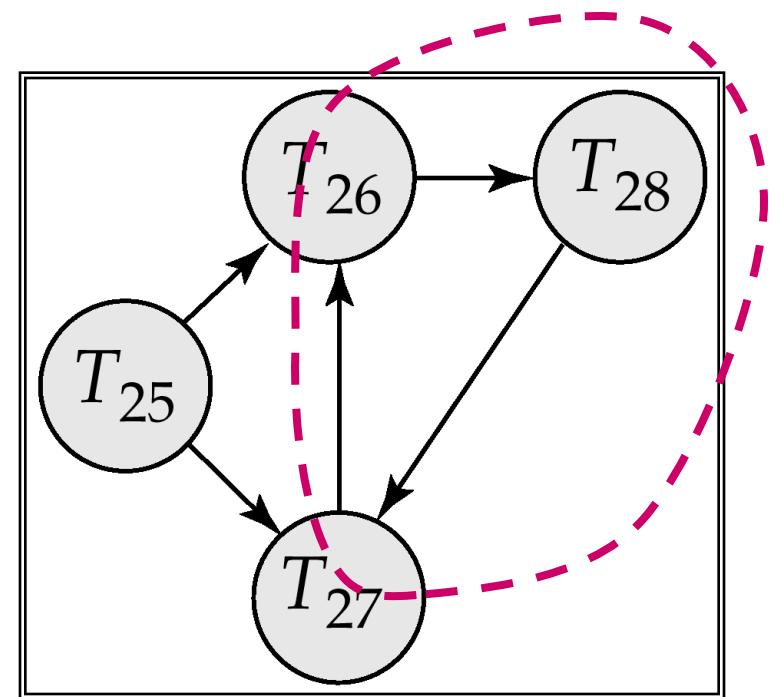
✿ Prevention: Timeout

- ✿ Waiting limit
- ✿ Can abort transactions that are not deadlocked
- ✿ Timeout interval is difficult to determine
- ✿ Used by desktop DBMSs

Deadlock Detection (Cont.)



Wait-for graph without a cycle



Wait-for graph with a cycle

Two Phase Locking (2PL)

- ❖ Protocol to prevent *lost update problems*
 1. Locking phase
 2. Unlocking phase

Lock A

....

Lock B

Lock C

....

Unlock B

Unlock A

Unlock C

Two Phase Locking (2PL)

✿ Protocol to prevent *lost update problems*

✿ ปัญหาการอ่านจำนวนที่ว่างบนเครื่องบินเพื่อจอง*update*

✿ All transactions must follow conditions

✿ Obtain lock before accessing item

- ระบบจะให้รอ ถ้าข้อล็อกแล้วไม่ได้ (conflicting lock)

✿ Cannot obtain new locks after releasing locks (within a transaction)

Lost Update Problem

Two users changing the same part of the database

Transaction A	Time	Transaction B
Read SR (10)	T ₁	
	T ₂	Read SR (10)
If SR > 0 then SR = SR - 1	T ₃	
	T ₄	If SR > 0 then SR = SR - 1
Write SR (9)	T ₅	
	T ₆	Write SR (9)

SR: Seat Reserved column name

Write-Write problem

Two distinct consecutive phases

during the transaction's execution:

✿ Expanding phase (Growing phase):

✿ locks are acquired and no locks are released
(the number of locks can only increase).

✿ Shrinking phase:

✿ locks are released and no locks are acquired.

ระบบสามารถจัดให้มีการเริ่มต้นรันหลายๆ T ไปพร้อมกันได้โดย

Guarantee Serializability

2PL

Lock..

...

Lock..

.

.

.

Unlock..

...

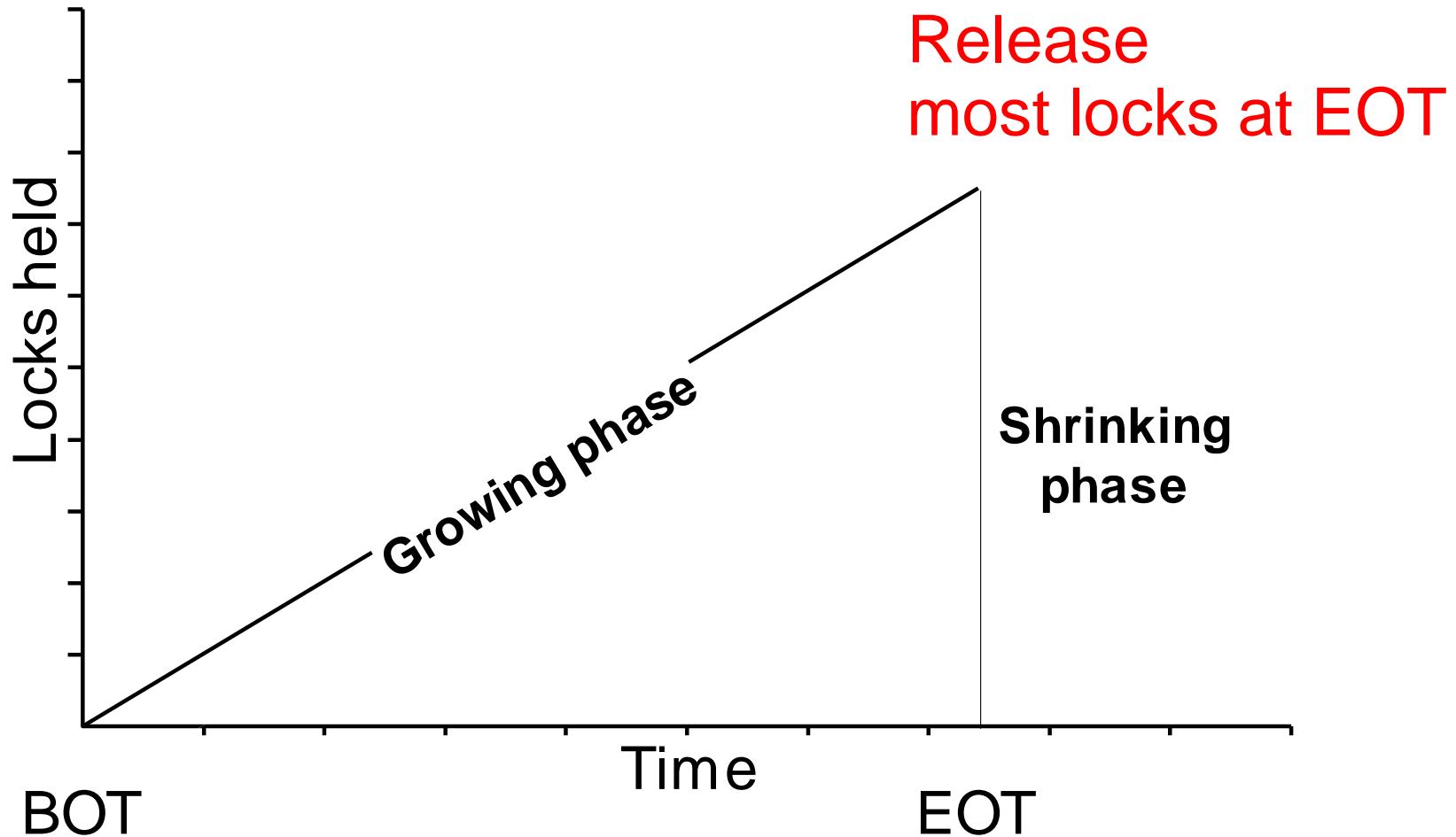
Unlock..

T1	T2
lock(A)	
read(A)	
	lock(A)
write(A)	
lock(B)	
unlock(A)	read(A)
	write(A)
	lock(B)
	unlock(A)
read(B)	
write(B)	
unlock(B)	read(B)
	write(B)
	unlock(B)

wait A

wait B

2PL Implementation



Optimistic Approaches

- ✿ ใช้กรณีที่มีการใช้ข้อมูลเดียวกันในเวลาเดียวกัน (**conflict**) น้อย
- ✿ ไม่ต้องมีการใช้วิธีล็อคข้อมูล

✿ Check for conflicts (version/time control)

- ✿ After each read and write
- ✿ At the end of transaction

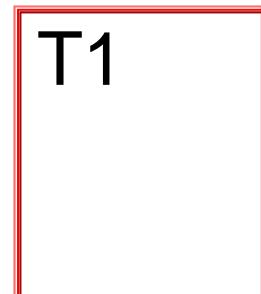
✿ Evaluation

- ✿ Less overhead: *no locks, deadlocks, ...*
- ✿ More variability: *penalty for conflict is rollback*
- ✿ Locking penalty: *waiting (less penalty than rollback)*

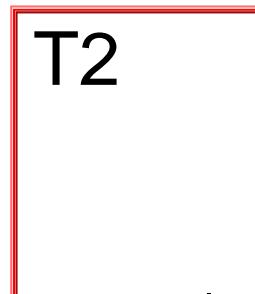
Timestamp-Based Protocols

- ✿ ทุกๆ transaction จะได้รับ timestamp ประจำเวลาเข้า คนมาก่อนจะได้ค่า timestamp น้อยกว่า คนมาทีหลัง
- ✿ Timestamps จะถูกใช้กำหนดลำดับการทำงาน ตามคำสั่งใน transaction เพื่อให้เป็นแบบ serializability order.

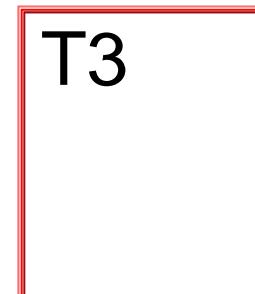
Timestamp 101



102



103



transaction

Timestamp-Based Protocols

- ✿ to assure **serializability**, the protocol maintains **for each data Q two timestamp values:**

✿ **W-timestamp(Q)**

- largest time-stamp of any T that executed write(Q) successfully.

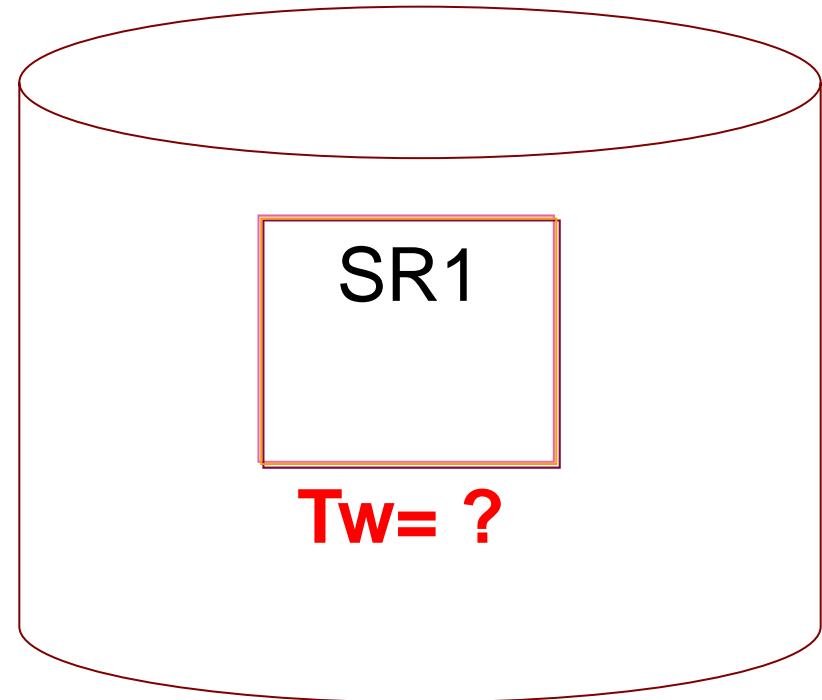
✿ **R-timestamp(Q)**

- largest time-stamp of any T that executed read(Q) successfully.

- เวลาบน data Q ไม่ใช่เวลาของ T ที่มาอ่าน/เขียน data Q หลังสุด แต่เป็นเวลาของ T ที่มี timestamp สูงสุด (เข้าสู่ระบบหลังสุด) ที่ได้เคยมาอ่าน/เขียน Q

Time-stamps

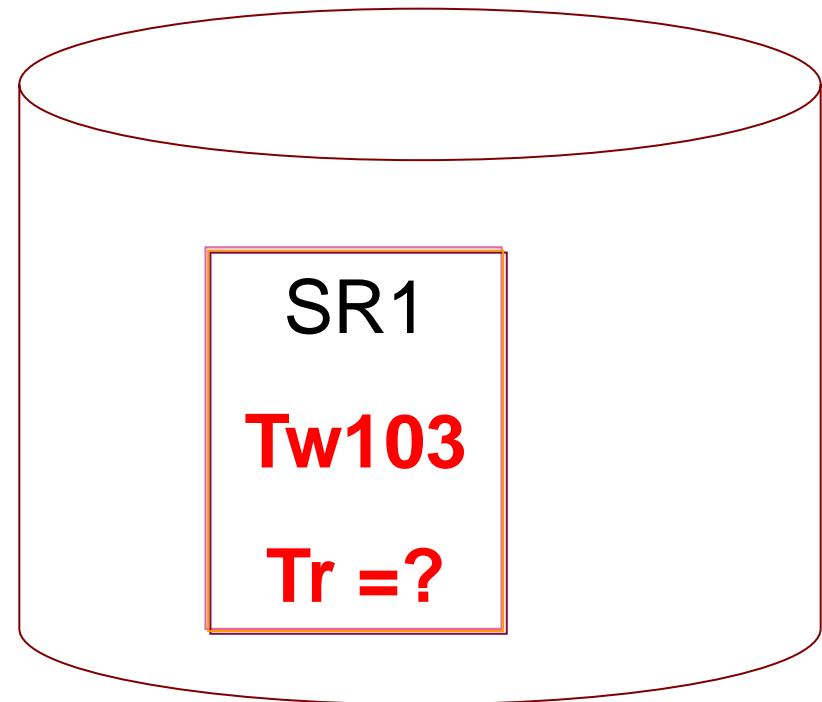
	101	102	103
1	T1		
2		T2	
3	W(SR1)	W(SR1)	T3 W(SR1)



1. ทราบแซดชั้นที่ write SR1 หลังสุดคือใคร?
2. Time-write บน SR1 = ?

Time-stamps

	101	102	103
1	T1		
2		R(SR1)	
3	RSR1)		



1. ทราบแซดชั้นที่ read SR1 หลังสุดคือใคร?
2. Time-read บน SR1 = ?

Timestamp Protocol

หลักการคือ read/write ตามลำดับเวลาประทับ
สมมุติ T_1 เข้าระบบก่อน T_2

- ✿ ถ้า T_1 จะ read data(A) ต้องตรวจสอบ $Tw(A)$
 - ✿ ถ้าพบว่า T_2 ได้ write ลงบน A แล้ว, ต้องยกเลิก/abort transaction T_1 และ rollback คือล้างผลกระทบต่อฐานข้อมูล
- ✿ ถ้า T_1 จะ write data(A) ต้องตรวจสอบ $Tw(A)$ & $Tr(A)$
 - ✿ ถ้าพบว่า T_2 read A หรือ พบร่วมกับ T_2 write A ไปก่อนแล้ว
 - ต้องยกเลิก/abort transaction T_1
 - ✿ ถ้าไม่ใช่กรณีดังกล่าว T_1 จึงจะสามารถ write A ได้
- ✿ After read/write, update timestamp $Tw(A)$ & $Tr(A)$

ยกตัวอย่าง ของตัวเครื่องบิน

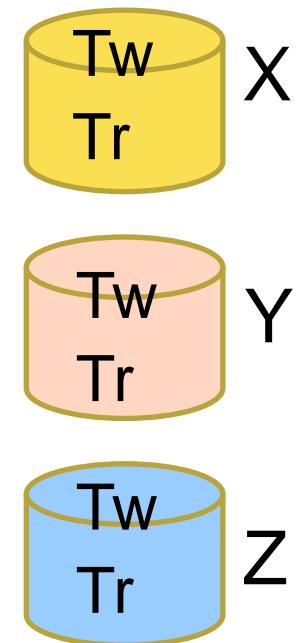
Timestamp rules

- ✿ Maintain the serial order
 - ✿ T1 must write data before T2 read, write
 - ✿ T1 can not read data that written by T2.

Example : *Timestamp Approach*

A partial schedule for several data items for transactions with timestamps 1, 2, 3, 4, 5

	T_1	T_2	T_3	T_4	T_5	
1						read(X)
2		read(Y)				
3	read(Y)					
4			write(Y)			
5			write(Z)			
6				read(Z)		
7		read(X)				
8	read(X)					
9			write(Z)			
10			abort			write(Y)
11						write(Z)



Locking strategies by table type.

Table Type	Examples	Suggested Locking Strategy
Live-High Volume	• Account	• <u>Optimistic</u> (first choice) • <u>Pessimistic</u> (second choice)
Live-Low Volume	• Customer • Insurance Policy	• <u>Pessimistic</u> (first choice) • <u>Optimistic</u> (second choice)
Log (<i>typically append only</i>)	• AccessLog • AccountHistory • TransactionRecord	• <u>Overly Optimistic</u>
Lookup/Reference (<i>typically read only</i>)	• State • PaymentType	• <u>Overly Optimistic</u>

Overly Optimistic is suitable for single user systems