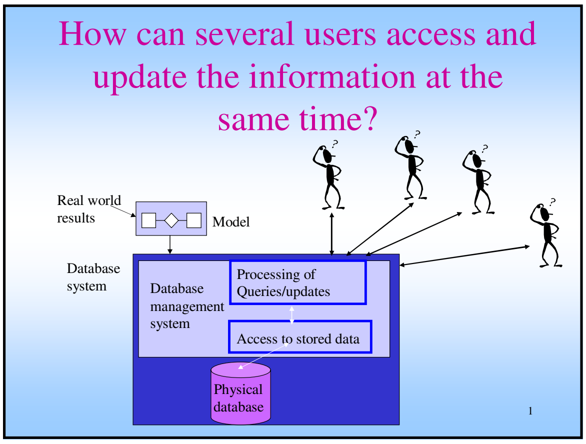


How can several users access and update the information at the same time?

Real world results

Model

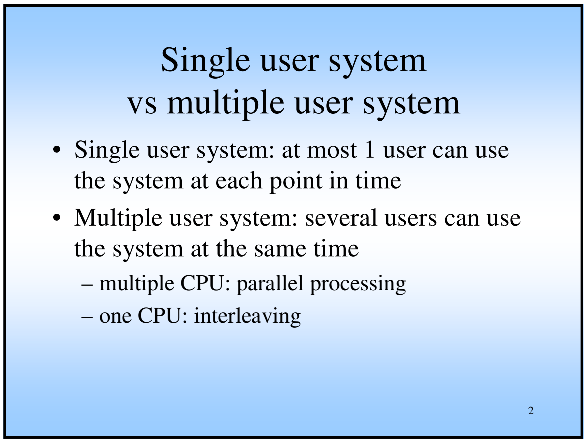
Database system

Database management system

Processing of Queries/updates

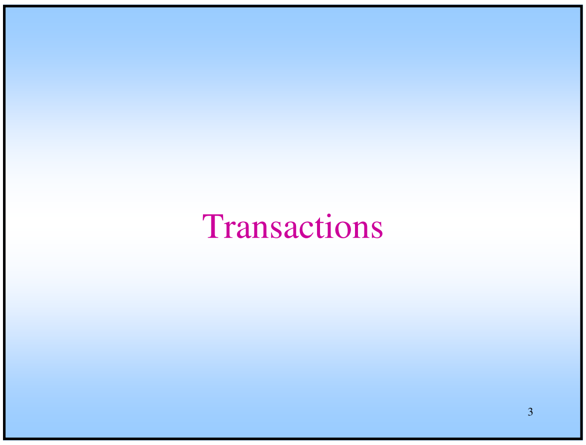
Access to stored data

Physical database

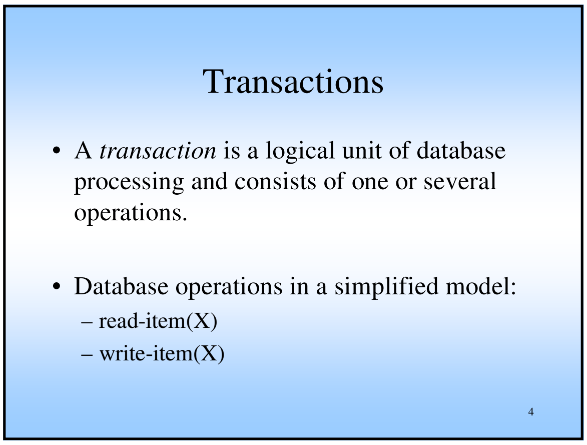


Single user system vs multiple user system

- Single user system: at most 1 user can use the system at each point in time
- Multiple user system: several users can use the system at the same time
  - multiple CPU: parallel processing
  - one CPU: interleaving

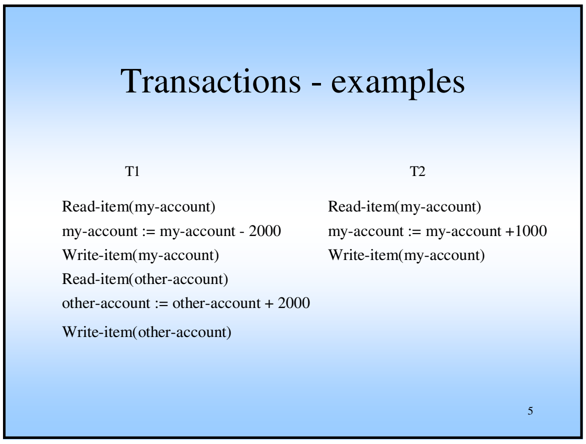


Transactions



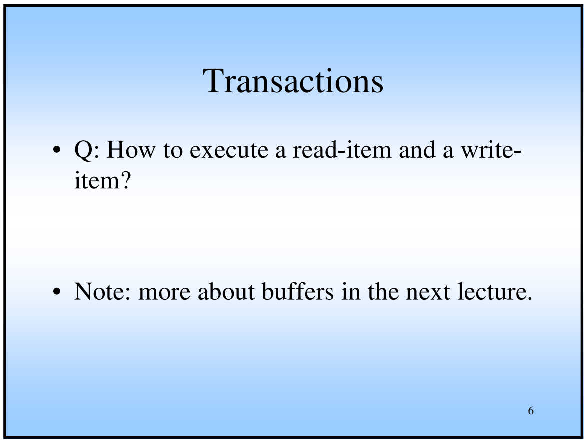
Transactions

- A *transaction* is a logical unit of database processing and consists of one or several operations.
- Database operations in a simplified model:
  - read-item(X)
  - write-item(X)



Transactions - examples

T1	T2
Read-item(my-account)	Read-item(my-account)
my-account := my-account - 2000	my-account := my-account +1000
Write-item(my-account)	Write-item(my-account)
Read-item(other-account)	
other-account := other-account + 2000	
Write-item(other-account)	



Transactions

- Q: How to execute a read-item and a write-item?
- Note: more about buffers in the next lecture.

Read-item(X)

- Locate the block on disk that contains X
- Copy the block to primary memory (a buffer)
- Copy X from the buffer to program variable X.

Write-item(X)

1. Locate the block on disk that contains X
2. Copy the block to primary memory (a buffer)
3. Copy the value of program variable X to the right place in the buffer
4. Store the modified block on disk.

Schedule

- A schedule defines the order between the operations in the different transactions.

Schedule - example

T1	TIME	T2
Read-item(my-account)		
my-account := my-account - 2000		
		Read-item(my-account)
		my-account := my-account +1000
Write-item(my-account)		
Read-item(other-account)		
		Write-item(my-account)
Other-account := other-account + 2000		
Write-item(other-account)		

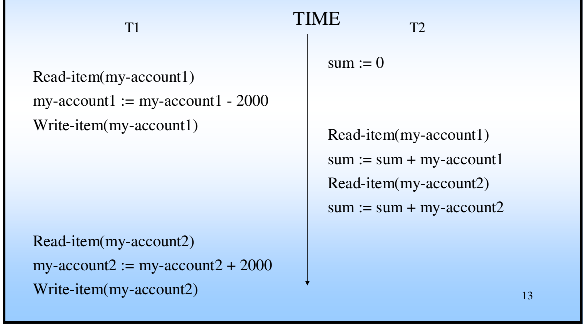
Lost update problem

T1	TIME	T2
Read-item(my-account)		
my-account := my-account - 2000		
		Read-item(my-account)
		my-account := my-account +1000
Write-item(my-account)		
Read-item(other-account)		
		Write-item(my-account)
Other-account := other-account + 2000		
Write-item(other-account)		

Dirty read problem

T1	TIME	T2
Read-item(my-account)		
my-account := my-account - 2000		
Write-item(my-account)		
		Read-item(my-account)
		my-account := my-account +1000
		Write-item(my-account)
Read-item(other-account)		
FAIL		

Incorrect summary problem



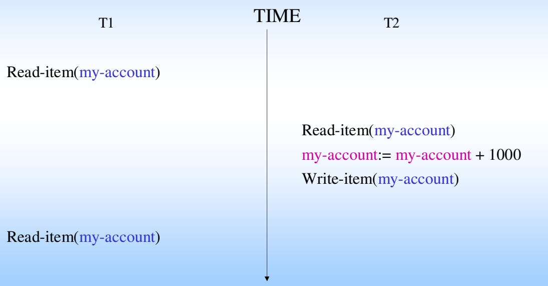
T1	T2
	sum := 0

Read-item(my-account1)
my-account1 := my-account1 - 2000
Write-item(my-account1)	

	Read-item(my-account1)
	sum := sum + my-account1
	Read-item(my-account2)
	sum := sum + my-account2

Read-item(my-account2)
my-account2 := my-account2 + 2000
Write-item(my-account2)	

Unrepeatable read problem



T1	T2
Read-item(my-account)	

	Read-item(my-account)
	my-account:= my-account + 1000
	Write-item(my-account)

Read-item(my-account)	

Properties for transactions

ACID: Atomicity, Consistency preservation, Isolation, Durability

- A: A transaction is an atomic unit: it is either executed completely or not at all
- C: A database that is in a consistent state before the execution of a transaction (i.e. it fulfills the conditions in the schema and other conditions declared for the database), is also in a consistent state after the execution.

Properties for transactions

ACID: Atomicity, Consistency preservation, Isolation, Durability

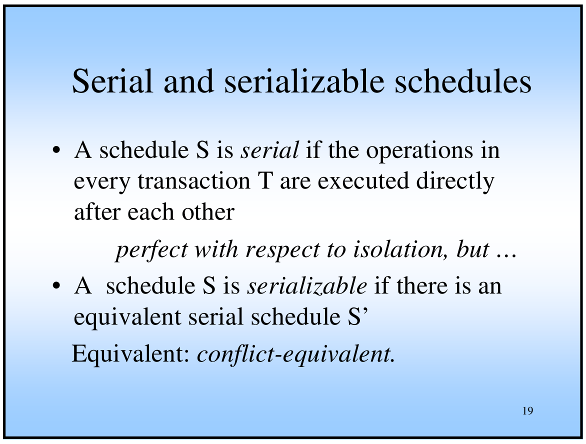
- I: A transaction should act as if it is executed isolated from other transactions.
- D: Changes in the database made by a committed transaction are permanent.

Properties for transactions

How are the ACID properties achieved?

- A: recovery system
- C: programmer + DBMS
- I: concurrency contol
- D: recovery system

Concurrency control (Isolation)



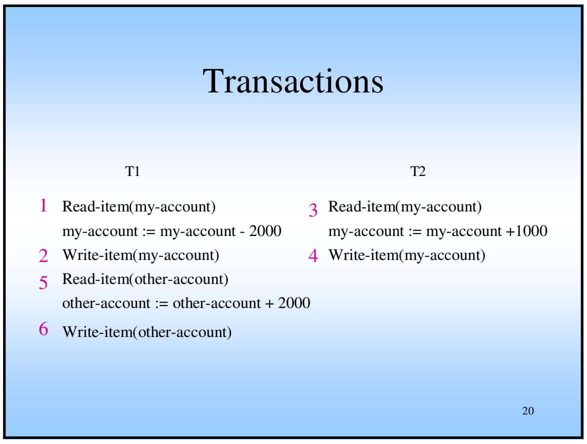
Serial and serializable schedules

- A schedule S is *serial* if the operations in every transaction T are executed directly after each other

  perfect with respect to isolation, but …

- A schedule S is *serializable* if there is an equivalent serial schedule S’

  Equivalent: *conflict-equivalent.*



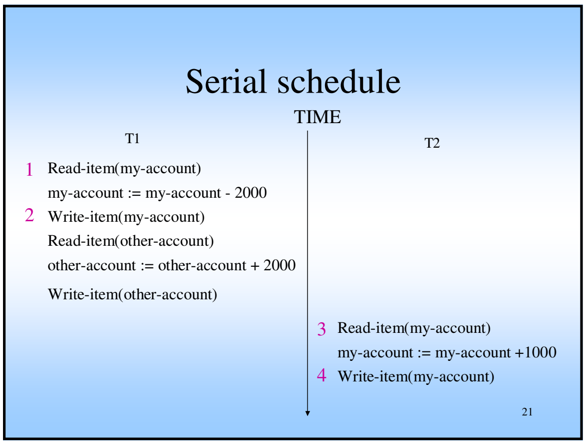
Transactions

T1

1 Read-item(my-account)
my-account := my-account - 2000
2 Write-item(my-account)
5 Read-item(other-account)
other-account := other-account + 2000
6 Write-item(other-account)

T2

3 Read-item(my-account)
my-account := my-account +1000
4 Write-item(my-account)



Serial schedule

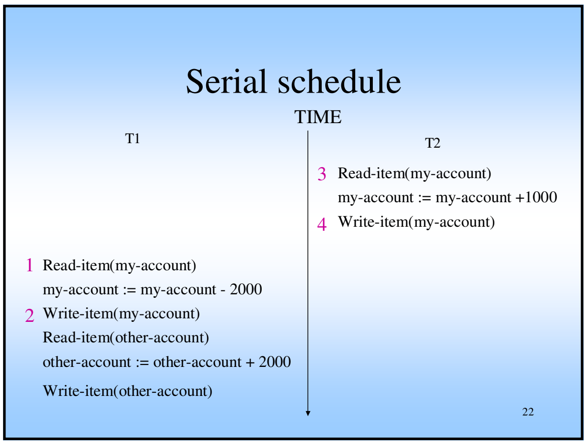
TIME

T1

1 Read-item(my-account)
my-account := my-account - 2000
2 Write-item(my-account)
Read-item(other-account)
other-account := other-account + 2000
Write-item(other-account)

T2

3 Read-item(my-account)
my-account := my-account +1000
4 Write-item(my-account)



Serial schedule

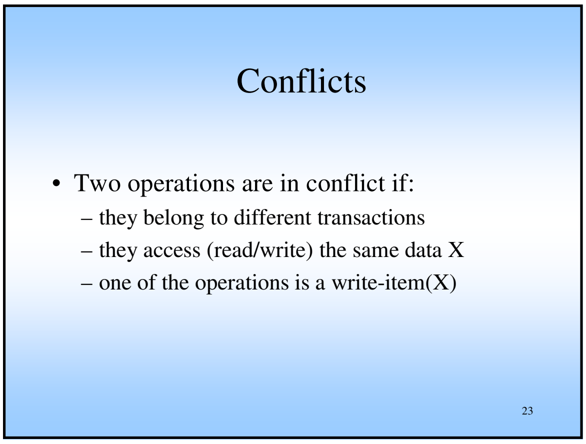
TIME

T1

T2

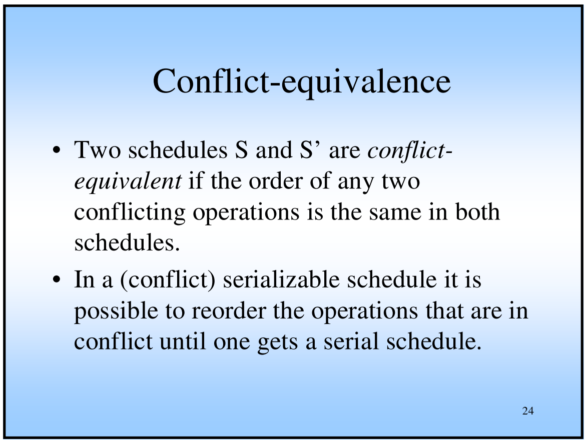
3 Read-item(my-account)
my-account := my-account +1000
4 Write-item(my-account)

1 Read-item(my-account)
my-account := my-account - 2000
2 Write-item(my-account)
Read-item(other-account)
other-account := other-account + 2000
Write-item(other-account)



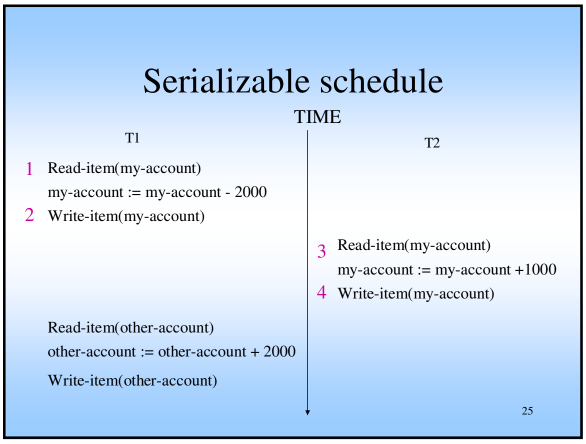
Conflicts

- Two operations are in conflict if:
  - they belong to different transactions
  - they access (read/write) the same data X
  - one of the operations is a write-item(X)



Conflict-equivalence

- Two schedules S and S’ are *conflict-equivalent* if the order of any two conflicting operations is the same in both schedules.
- In a (conflict) serializable schedule it is possible to reorder the operations that are in conflict until one gets a serial schedule.



Serializable schedule

TIME

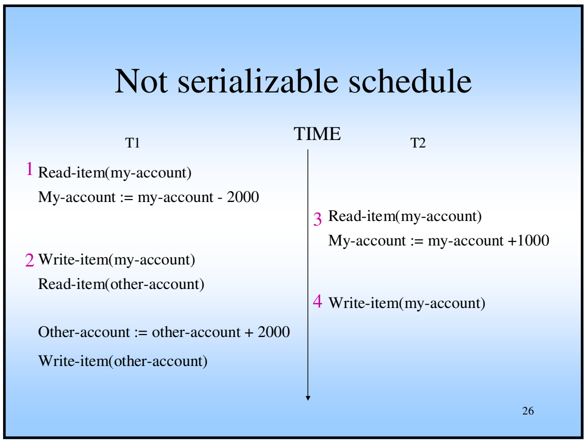
T1

1 Read-item(my-account)
my-account := my-account - 2000
2 Write-item(my-account)

T2

3 Read-item(my-account)
my-account := my-account +1000
4 Write-item(my-account)

Read-item(other-account)
other-account := other-account + 2000
Write-item(other-account)



Not serializable schedule

TIME

T1

1 Read-item(my-account)
My-account := my-account - 2000

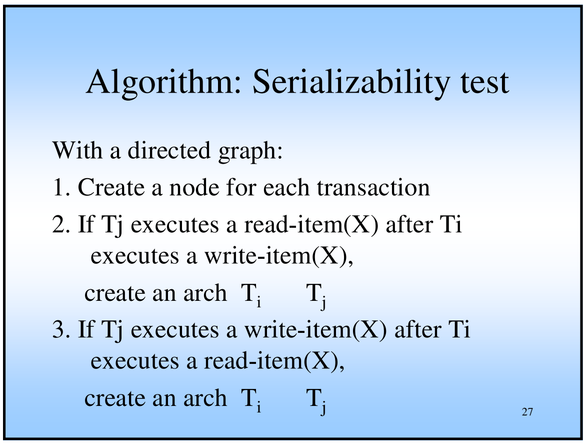
T2

3 Read-item(my-account)
My-account := my-account +1000

2 Write-item(my-account)
Read-item(other-account)

4 Write-item(my-account)

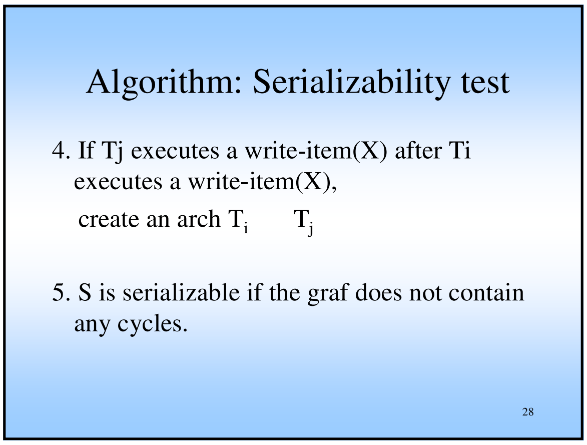
Other-account := other-account + 2000
Write-item(other-account)



Algorithm: Serializability test

With a directed graph:

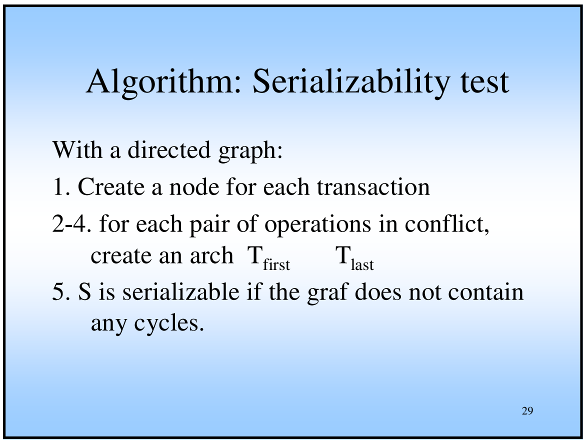
1. Create a node for each transaction
2. If Tj executes a read-item(X) after Ti executes a write-item(X),
   create an arch T_i T_j
3. If Tj executes a write-item(X) after Ti executes a read-item(X),
   create an arch T_i T_j



Algorithm: Serializability test

4. If Tj executes a write-item(X) after Ti executes a write-item(X),
   create an arch T_i T_j

5. S is serializable if the graf does not contain any cycles.



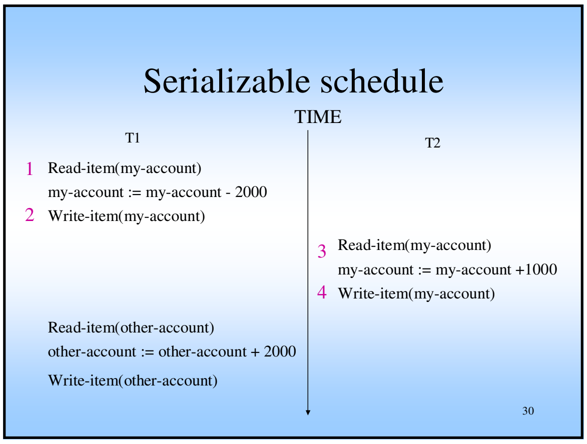
Algorithm: Serializability test

With a directed graph:

1. Create a node for each transaction

2-4. for each pair of operations in conflict, create an arch T_{first} T_{last}

5. S is serializable if the graf does not contain any cycles.



Serializable schedule

TIME

T1

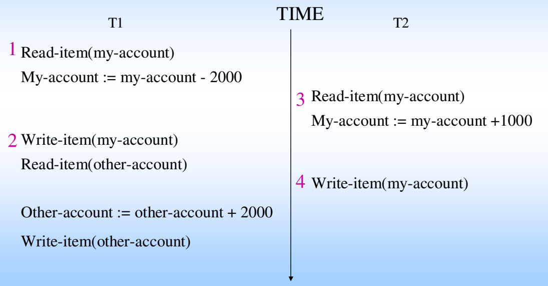
1 Read-item(my-account)
my-account := my-account - 2000
2 Write-item(my-account)

T2

3 Read-item(my-account)
my-account := my-account +1000
4 Write-item(my-account)

Read-item(other-account)
other-account := other-account + 2000
Write-item(other-account)

Not serializable schedule



T1

TIME

T2

1 Read-item(my-account)

My-account := my-account - 2000

3 Read-item(my-account)

My-account := my-account +1000

2 Write-item(my-account)

Read-item(other-account)

4 Write-item(my-account)

Other-account := other-account + 2000

Write-item(other-account)

- Can we make sure that we only get serializable schedules?

Locking

- Locking: to control access to data
- Shared/Exclusive lock or read/write lock
  - read-lock(X) (shared lock)
    - If X is unlocked or locked by a shared lock, lock it, otherwise wait until it is possible to lock it
  - write-lock(X) (exclusive lock)
    - If X is unlocked, lock it, otherwise wait until X is unlocked
  - unlock(X).

Shared/Exclusive locking

1. A transaction T should lock X with a read-lock(X) or a write-lock(X) before executing a read-item(X).
2. A transaction T should lock X with a write-lock(X) before executing a write-item(X).
3. A transaction T should unlock X with a unlock(X) after all read-item(X) and write-item(X) in T have been executed.

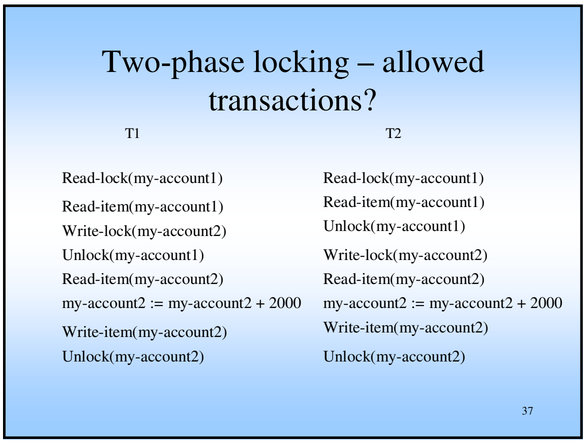
Shared/Exclusive locking

4. A transaction T should not use a read-lock(X) if it already has a read or write lock on X.
5. A transaction T should not use a write-lock(X) if it already has a read or write lock on X.

4 and 5 can sometimes be replaced by up- and downgrading of locks.

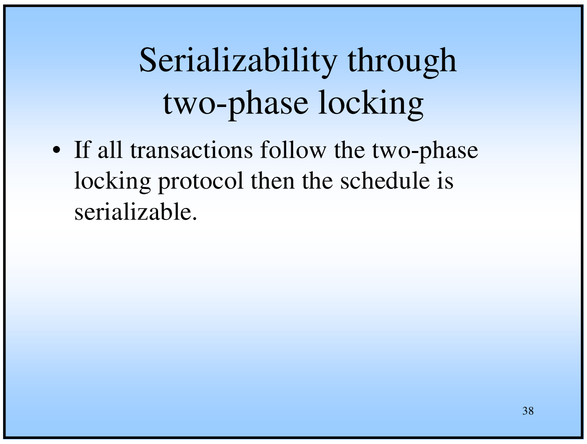
Two-phase locking

- A transaction follows the two-phase locking protocol if *all* locking operations (read-lock and write-lock) for all data items come before the first unlock operation in the transaction
- A transaction that follows the two-phase locking protocol has an expansion phase and a shrinking phase.



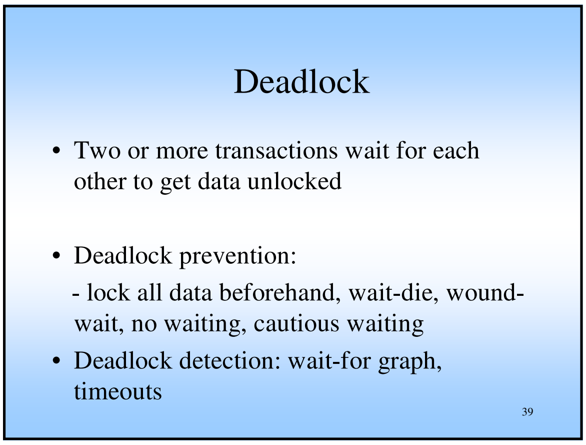
Two-phase locking – allowed transactions?

T1	T2
Read-lock(my-account1)	Read-lock(my-account1)
Read-item(my-account1)	Read-item(my-account1)
Write-lock(my-account2)	Unlock(my-account1)
Unlock(my-account1)	Write-lock(my-account2)
Read-item(my-account2)	Read-item(my-account2)
my-account2 := my-account2 + 2000	my-account2 := my-account2 + 2000
Write-item(my-account2)	Write-item(my-account2)
Unlock(my-account2)	Unlock(my-account2)



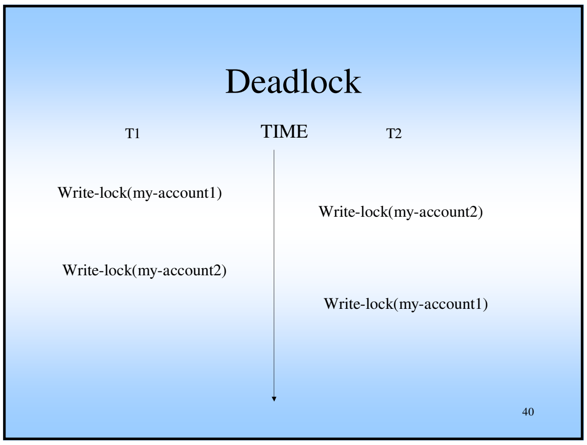
Serializability through two-phase locking

- If all transactions follow the two-phase locking protocol then the schedule is serializable.

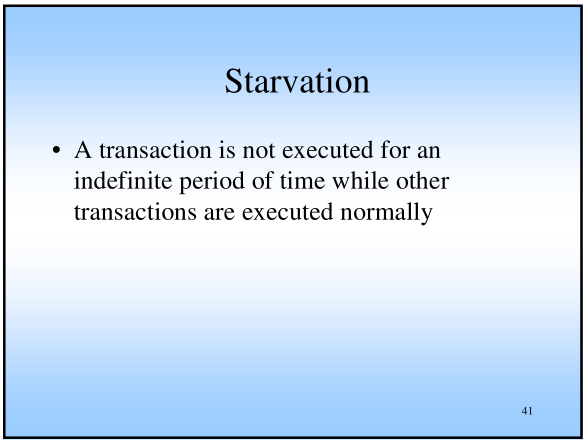


Deadlock

- Two or more transactions wait for each other to get data unlocked
- Deadlock prevention:
  - lock all data beforehand, wait-die, wound-wait, no waiting, cautious waiting
- Deadlock detection: wait-for graph, timeouts



Deadlock



Starvation

- A transaction is not executed for an indefinite period of time while other transactions are executed normally