Databasteknik
Databaser och bioinformatik
Transaction

Fang Wei-Kleiner

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.

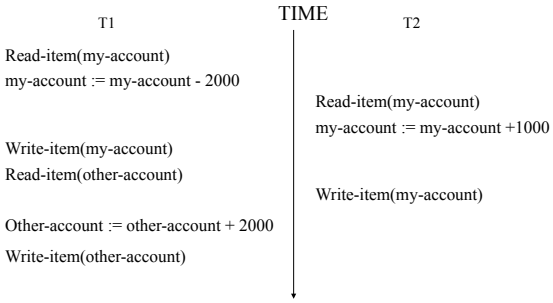
Write-item(X)

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.

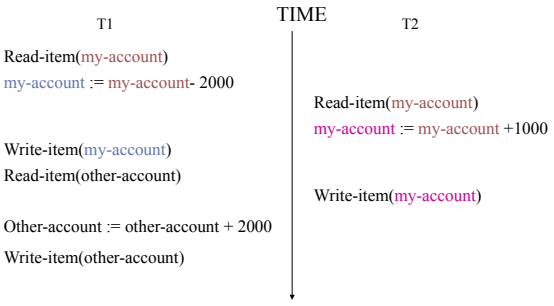
Schedule

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

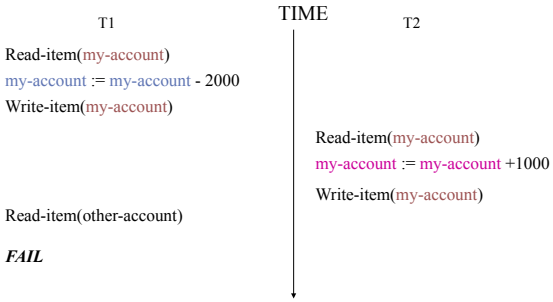
Schedule - example



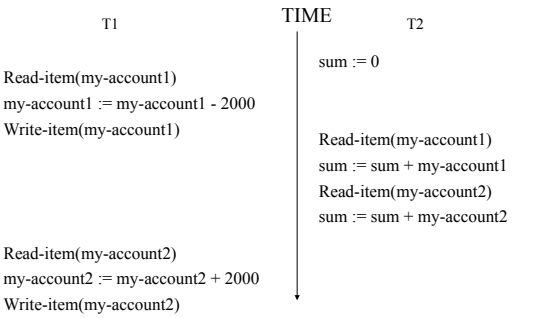
Lost update problem



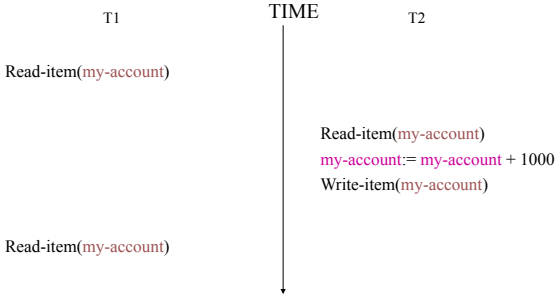
Dirty read problem



Incorrect summary problem



Unrepeatable read problem



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

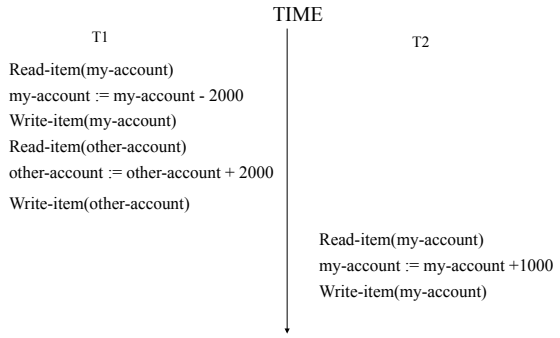
T1

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

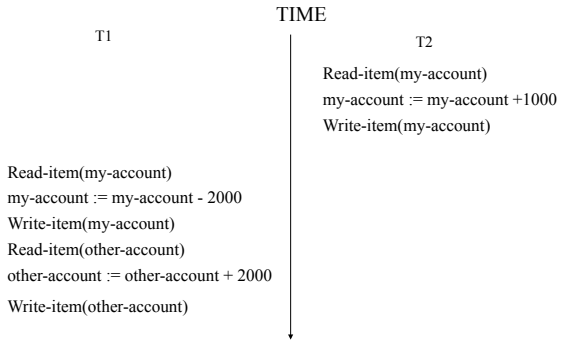
T2

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

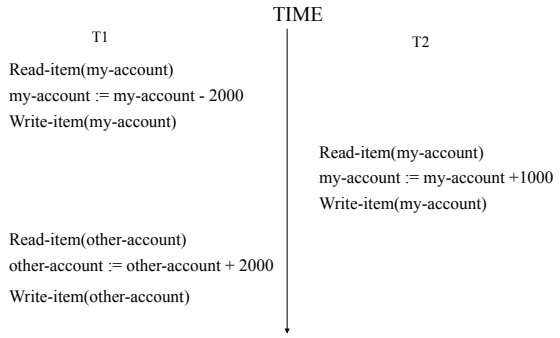
Serial schedule



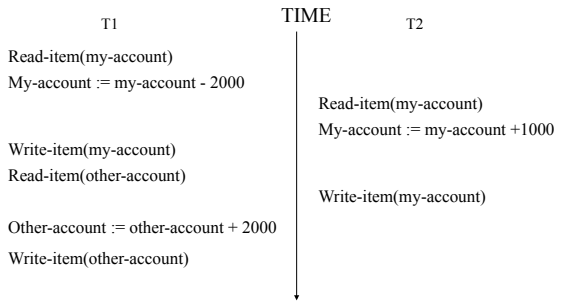
Serial schedule



Non-serial schedule



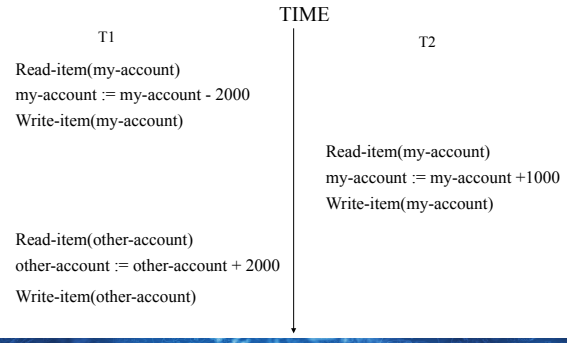
Non-serial schedule (2)



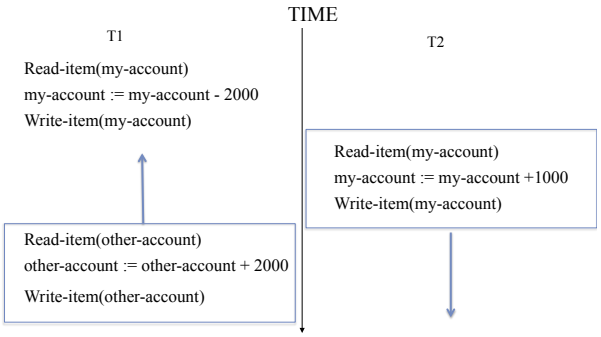
What is a good schedule?

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

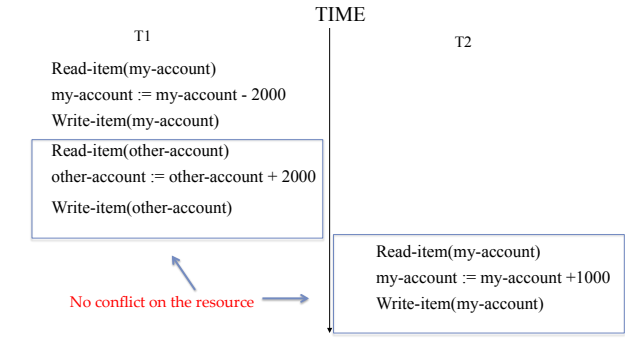
Non-serial schedule (S)



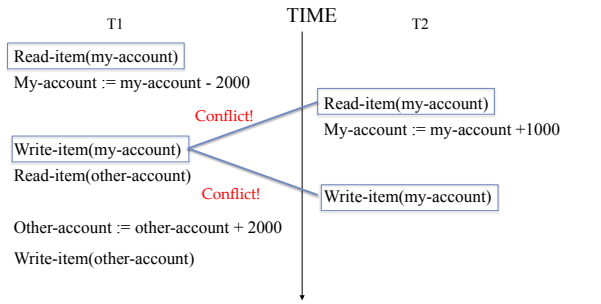
Non-serial schedule (S')



S equivalent to S'



Non-serial schedule (2)



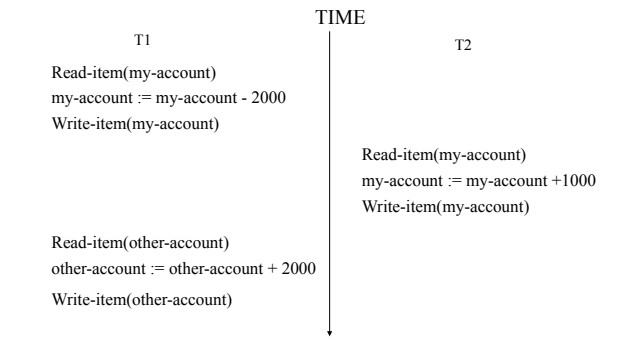
Conflicts

- 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)

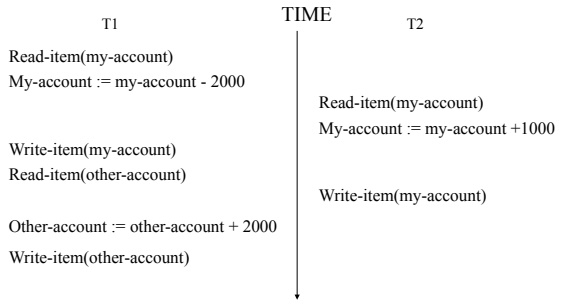
Conflict-equivalence

- 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



Not serializable schedule



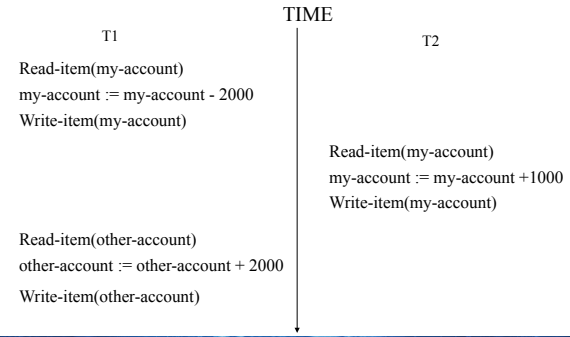
Algorithm: Serializability test

With a directed graph:

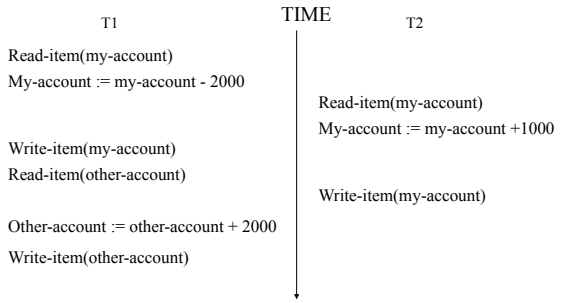
1. Create a node for each transaction
2. If Tj executes a read-item(X) after Ti executes a write-item(X), create an arch $T_i \rightarrow T_j$
3. If Tj executes a write-item(X) after Ti executes a read-item(X), create an arch $T_i \rightarrow T_j$
4. If Tj executes a write-item(X) after Ti executes a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

→S is serializable if the graph does not contain any cycles.

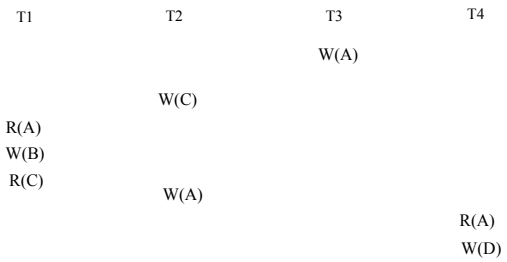
Serializable schedule



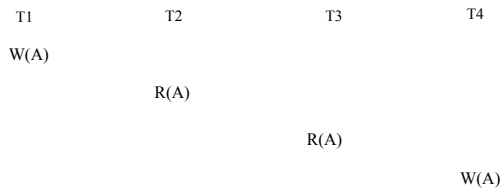
Not serializable schedule



Example



Example



- 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.

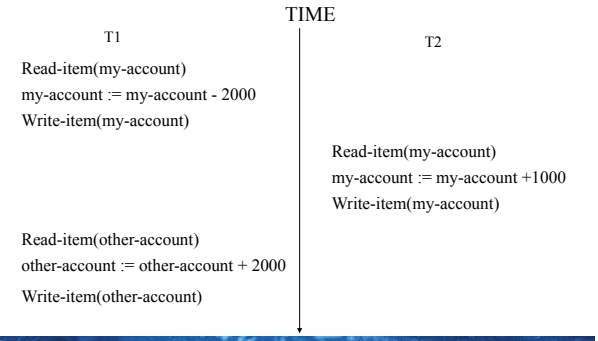
Two-phase locking – allowed transactions?

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)

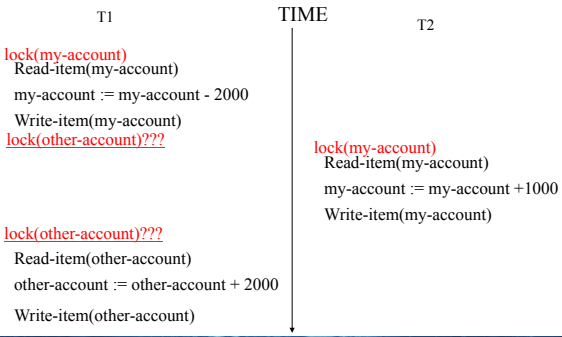
Two-phase locking – allowed transactions?

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)

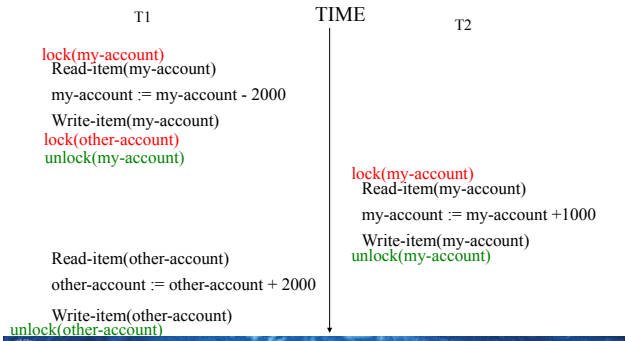
Follow 2PL Protocol?



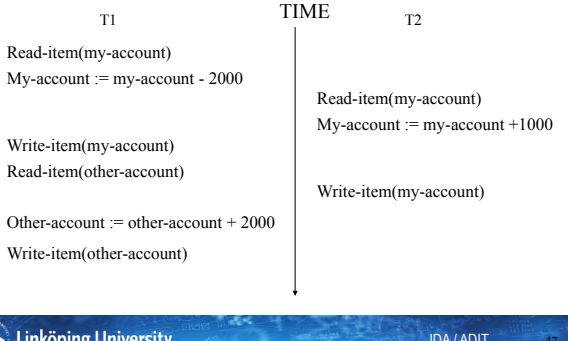
Follow 2PL Protocol?



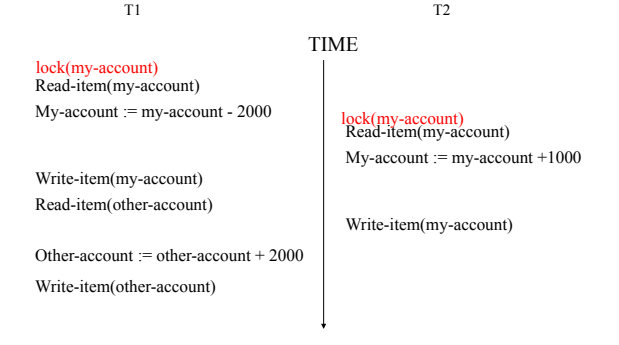
Follow 2PL Protocol? (yes)



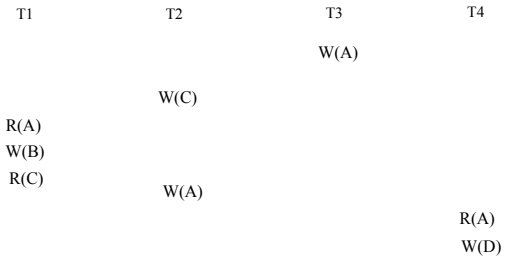
Follow 2PL Protocol?



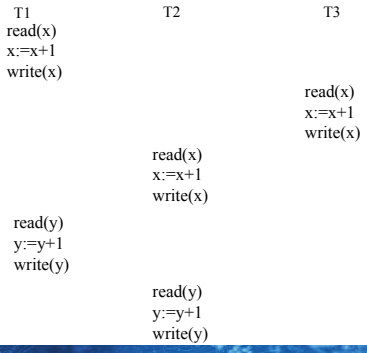
Follow 2PL Protocol? (no!)



Follow 2PL Protocol? (no!)



Follow 2PL Protocol? (yes)



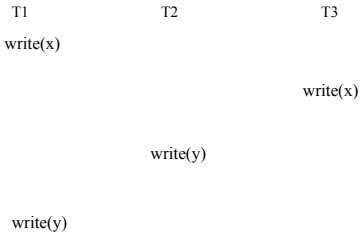
Serializability through 2PL

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

- Follow (aka permit) the two-phase locking protocol means we can apply the protocol so that the transactions interleave as it is.

→ However, there are serializable schedules which do not follow two-phase locking.

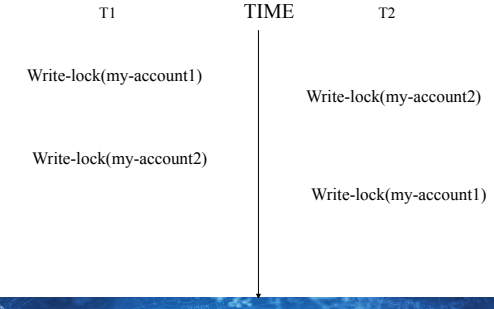
Serializable, but not follow 2PL Protocol



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