CS5300
Database Systems

Transaction Processing

A.R. Hurson
323 CS Building
hurson@mst.edu

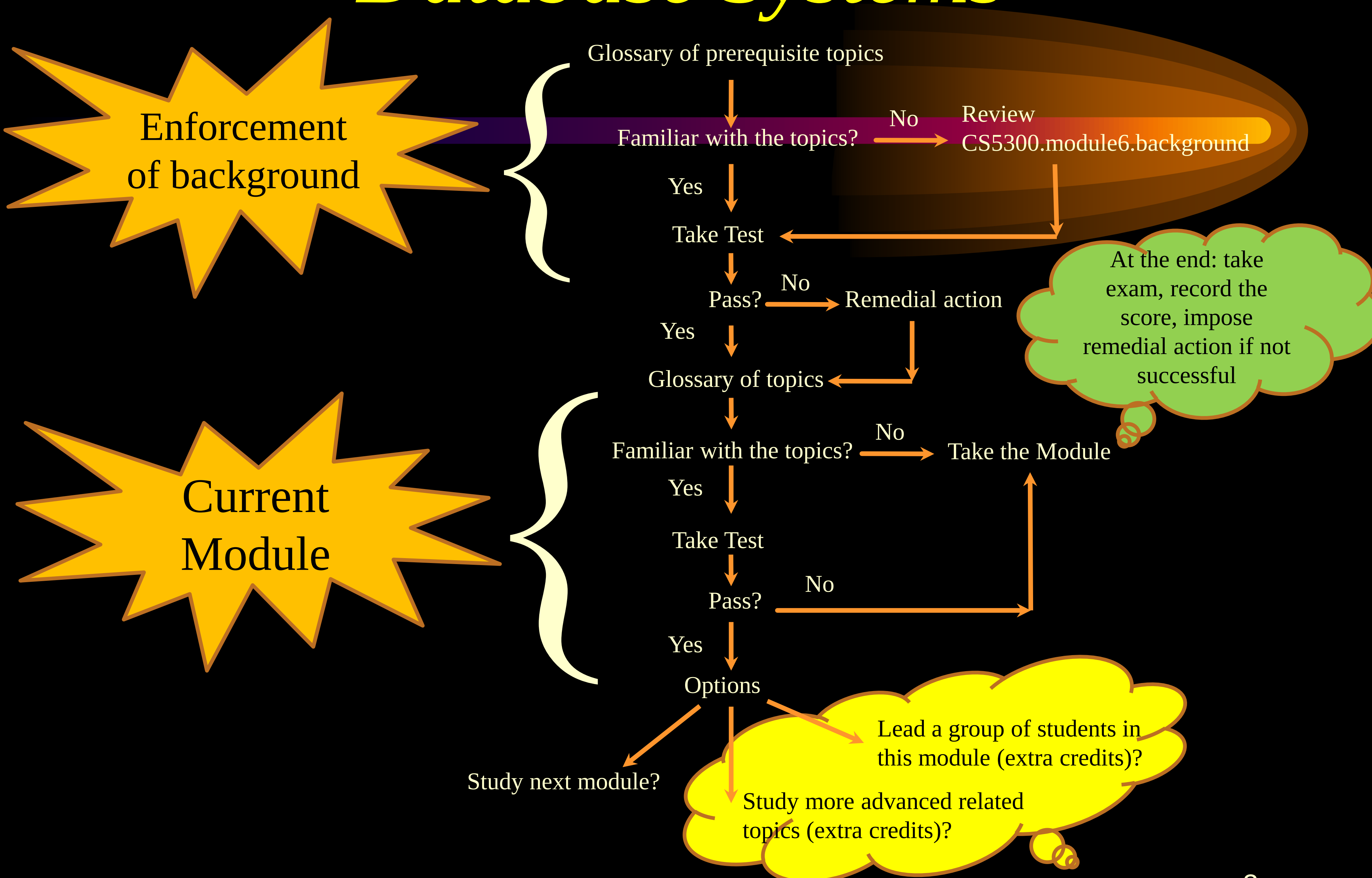
Database Systems

Note, this unit will be covered in six lectures. In case you finish it earlier, then you have the following options:

1) Take the early test and start CS5300.module7
2) Study the supplement module (supplement CS5300.module6)
3) Act as a helper to help other students in studying CS5300.module6

Note, options 2 and 3 have extra credits as noted in course outline.

Database Systems



Extra Curricular activities

Database Systems

- You are expected to be familiar with:
  - Relational database model,
  - SQL
  - Query processing and query optimization
- If not, you need to study CS5300.module6.background

Database Systems

- Previous module concentrated on query processing and query optimization. This module will concentrate on transaction processing. Note that we will distinguish transaction from query.
- A query does not change the data in base sources (e.g., relations), however, a transaction my do so. As a result, queries, initiated by several users, do not have conflicts with each other and can be executed in any order (including simultaneously). However, this is not true for transactions, since they may be in conflict with each other and hence needs to be executed in a proper sequence.

Database Systems

- In this module, we will talk about:
  - Transaction processing and management
  - Formal definition of transactions
  - ACID property
  - Serializability
  - Concurrency control
  - Concurrency control protocols
  - Transaction processing

Database Systems

- Two potential problems:
  - Two requests attempt to update the same data item, simultaneously, and
  - system fails during the execution of a request.
- In case of retrieve only request (e.g., query processing):
  - The 1st issue has no consequences, and
  - The 2nd issue is resolved by restarting the request.

Database Systems

- In the following few slides, we will define several terms that should motivate:
  - Issues of concern in transaction processing,
  - Capability of the database system in transaction processing, and
  - Characteristics of a transaction.

Database Systems

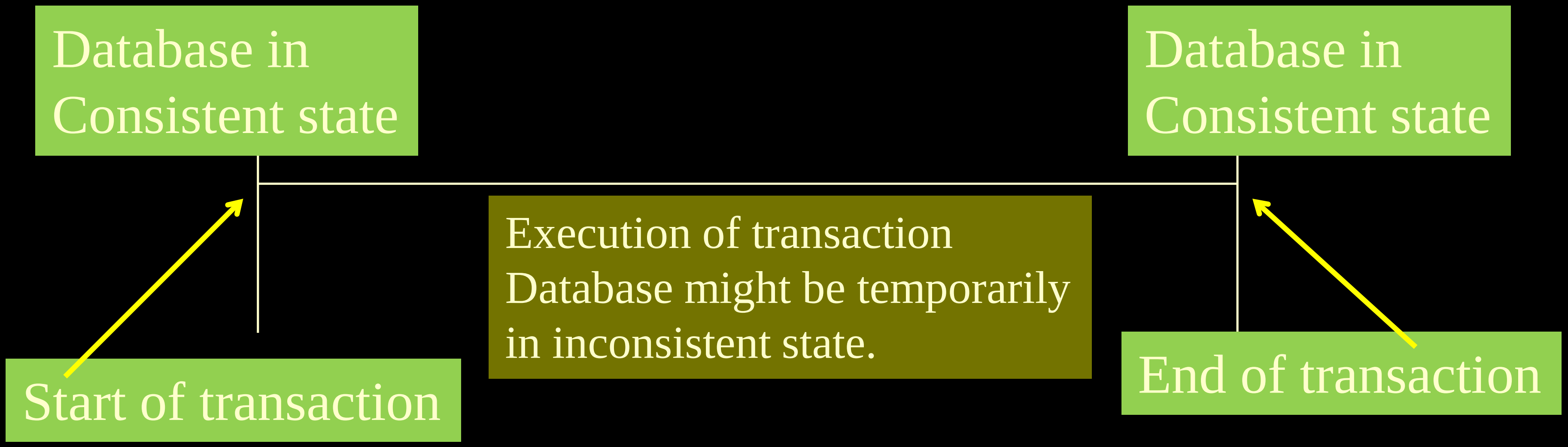
- In a query processing there is no notion of consistent execution or reliable execution. These issues are becoming a part of transaction processing.
- A transaction is a basic unit of consistent and reliable computing.
- We distinguish a difference between database consistency and transaction consistency.

Database Systems

- A database is in consistent state if it obeys all integrity constraints defined over it.
- State of a database changes due to the update operations — modifications, insertions, and deletions.
- Database can be temporarily inconsistent during the execution of a transaction. The important point is that the database should be in consistent state when the transaction terminates.
- Transaction consistency refers to the actions of concurrent transactions — we would like database remain in a consistent state even if there are a number of concurrent users' transactions.

Database Systems

- A transaction is a sequence of operations that transfers database from one consistent state to another consistent state.



Database Systems

- Several issues hinder transaction consistency:
  - Concurrent execution of transactions,
  - Replicated data, and
  - Failure.
- A replicated database is in a mutually consistent state if copies of every data item in it have identical values — one copy equivalence.

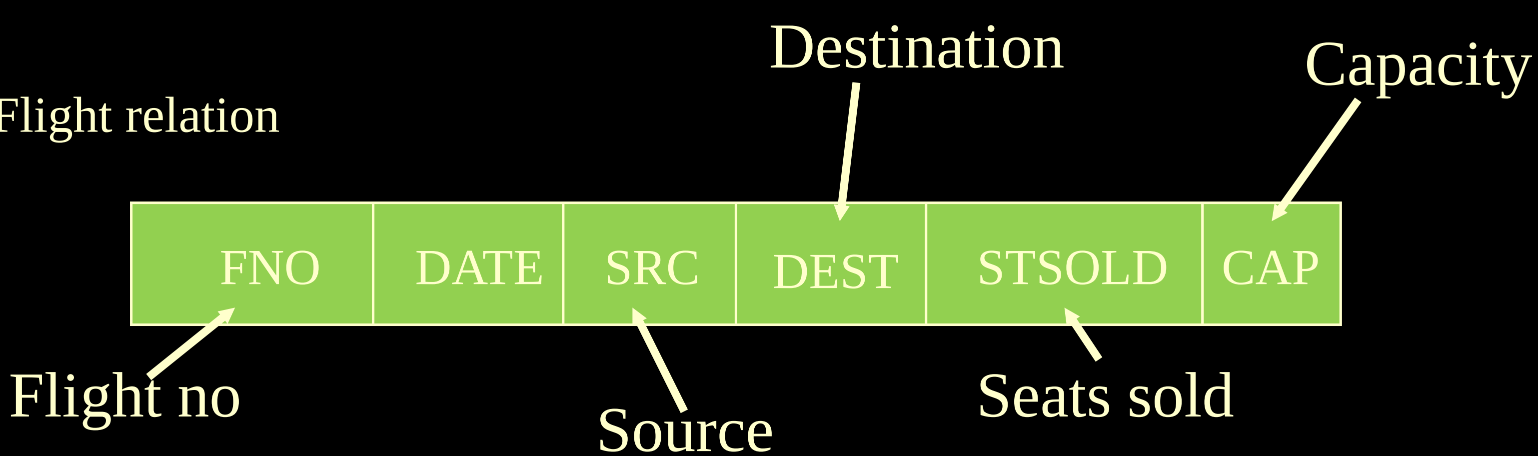
Database Systems

- Reliability refers to resiliency of a system to various types of failures and its ability to recover from it.
  - A resilient system tolerates system failure and continues to provide services,
  - A recoverable system is the one that can get to a consistent state under failure.

Database Systems

- Transaction management – Example
  - Assume the following database (air line reservation system):

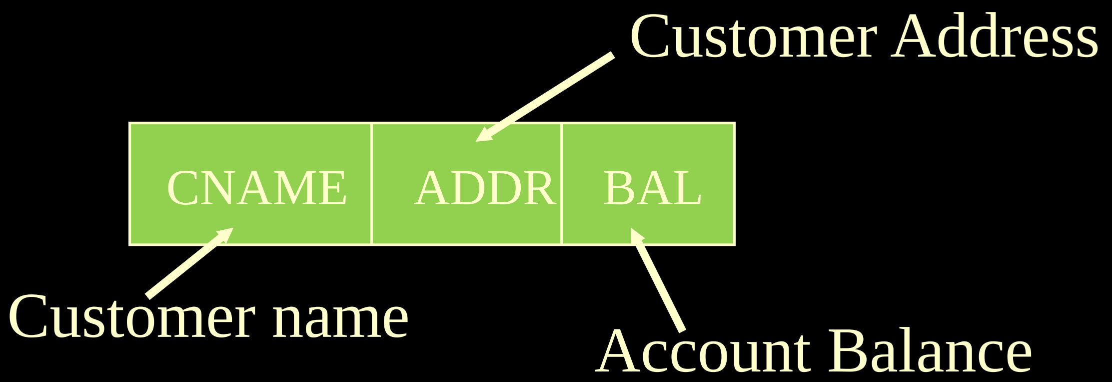
Flight relation



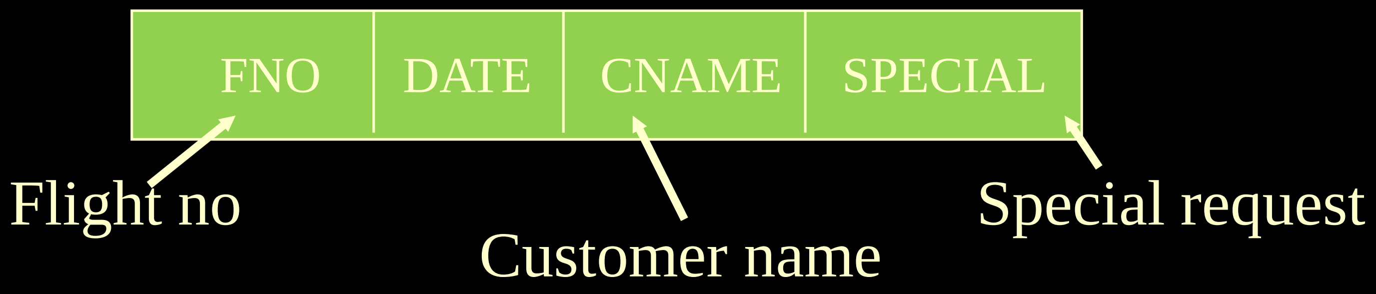
Database Systems

- Transaction management – Example

Customer relation



FC relation



Database Systems

- Transaction management – Example

```
Begin_transaction Reservation
Begin
    input (flight-no, date, customer-name)
    EXEC SQL UPDATE FLIGHT
        SET       STSOLD = STSOLD + 1
        WHERE     FNO = flight-no
        AND       DATE = date
    EXEC SQL INSERT
        INTO      FC(FNO,DATE,CNAME,SPECIAL)
        VALUES    (flight-no,date,customer-name,null);
    output ("reservation completed")
end
```

Database Systems

- Transaction management – Example
  - The previous example assumed that there will always be a free seat available. However, transaction might fail because the plane is full. A transaction must always terminate even if there is a failure.
  - If a transaction completes its task successfully, then the transaction must commit – its results will be available to other transactions.
  - If a transaction stops without completing its task then it must be aborted – all its already performed operations must be undone.

Database Systems

- Transaction management — Example

```
Begin_transaction Reservation
Begin
    input (flight-no, date, customer-name)
    EXEC SQL  SELECT  STSOLD,CAP
                      INTO     temp1,temp2
                      FROM     FLIGHT
                      WHERE  FNO = flight-no
                      AND      DATE = date;
    if temp1 = temp2 then
    begin
            output (“no available seats”);
    Abort
    end
    else begin
            EXEC SQL UPDATE FLIGHT
                 SET              STSOLD = STSOLD + 1
                 WHERE    FNO = flight-no
                 AND         DATE = date;
       EXEC SQL INSERT
                 INTO        FC(FNO,DATE,CNAME,SPECIAL)
                 VALUES  (flight-no,date,customer-name,null);
             Commit;
       output (“reservation completed”)
             end
    end-if
end
```

Database Systems

- Let us define a set of notations that allows us to formally represent a transaction:

Database Systems

- The data items read by a transaction is called read set (RS).
- The data item that a transaction writes are called write set (WS).
- The base set for a transaction is defined as:

$$BS = RS \cup WS$$

Database Systems

- In our previous example:

  RS = {STSOLD, CAP}

  WS = {STSOLD, FNO, DATE, CNAME, SPECIAL}

  BS = {STSOLD, CAP, FNO, DATE, CNAME, SPECIAL}

Database Systems

- For a transaction T_i:

$O_{ij}(x) \in T_i$ (operation O_j of transaction T_i operating on data item x)

$O_{ij}(x) \in \{read, write\}$ (operations are atomic)

$OS_i = \cup_j O_{ij}$

$N_i \in \{abort, commit\}$ (N_i the termination condition)

$T_i = \{\Sigma_i, \prec_i\}$

$\Sigma_i = OS_i \cup \{N_i\}$

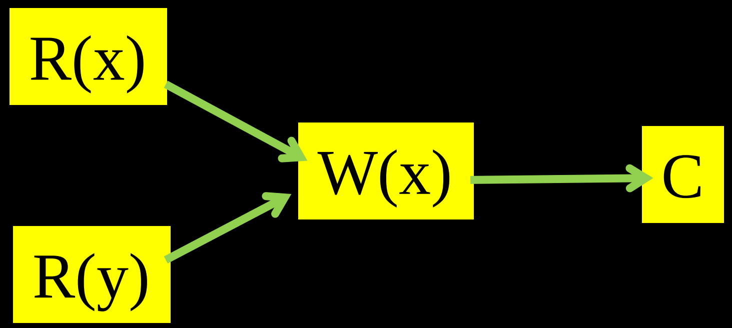
$\prec$ is a binary operator representing the execution order

$O_{ij}, O_{ik} \in OS_i$, if $O_{ij} = \{R(x) \text{ or } W(x)\}$ and $O_{ik} = W(x)$ for any data item x, then either $O_{ij} \prec_i O_{ik}$ or $O_{ik} \prec_i O_{ij}$

$\forall O_{ij} \in OS_i, O_{ij} \prec_i N_i$

- Consider the following transaction, its formal definition, and its graphical representation (T):

Read (x)
Read (y)
x $\leftarrow$ x + y
Write (x)
Commit



Σ = { R(x), R(y), W(x), C}
$\prec$ = { (R(x), W(x)), (R(y), W(x)),
(W(x), C), (R(x), C), (R(y), C)}
Where (O_i, O_j) as an element indicates that $O_i \prec O_j$

Database Systems

- As another example, recall our earlier reservation transaction. Also remember that the reservation transaction had two terminating conditions.
- First part can be formally defined as:

$$\Sigma = \{ R(STSOLD), R(CAP), A\}$$
$$\prec = \{ (O_1, A), (O_2, A)\}$$

Database Systems

- The second part can be represented as:

Σ = { R(STSOLD), R(CAP), W(STSOLD), W(FNO), W(DATE), W(CNAME), W(SPECIAL), C}

$\prec$ = { (O_1, O_3), (O_2, O_3), (O_1, O_4), (O_1, O_5), (O_1, O_6), (O_1, O_7), (O_2, O_4), (O_2, O_5), (O_2, O_6), (O_2, O_7), (O_1, C), (O_2, C), (O_3, C), (O_4, C), (O_5, C), (O_6, C), (O_7, C)}

Where O_1 = R(STSOLD), O_2 = R(CAP), O_3 = W(STSOLD), O_4 = W(FNO), O_5 = W(DATE), O_6 = W(CNAME), and O_7 = W(SPECIAL)

Database Systems

- Last lecture
  - Distinction between transaction and query
  - Issues of concern
    - Concurrent execution of transactions
    - Failure
  - Some terms
    - Consistent execution
    - Reliable execution
  - Formal definition of transaction

Database Systems

- In general, in a database system, one needs to ensure Atomicity, Consistency, Isolation, and Durability properties of transactions:

Database Systems

- Atomicity (all or nothing): either all operations of the transaction are reflected in database, or none are.
- Consistency (no violation of integrity rules): Execution of transaction in isolation preserves the consistency of the database.
- Isolation (Concurrent changes invisible and serializable): Even though multiple transactions may execute concurrently, each transaction assumes it is executed in isolation (it is unaware of other transactions executing concurrently in the system).
- Durability (Committed updates persist): After a transaction completes successfully, its results are becoming persistence.

Database Systems

- Atomicity
  - The database should always reflect a real state of the world.
  - A transaction must transfer the database from one consistent state to another.
  - If during the course of a transaction a failure occurs, then the database is in inconsistent state and it does not reflect a real world state. Therefore, the partial results must be undone.

Database Systems

- Atomicity
  - The activity of preserving the transaction's atomicity in the presence of aborts due to input data errors, system overheads, or deadlock is called transaction recovery.
  - The activity of ensuring atomicity in the presence of system crashes is called crash recovery.

Database Systems

- Consistency
  - If the database is consistent before execution of a transaction, the database remains consistent after the execution of the transaction.
  - Transactions are correct programs that do not violate database integrity constraints.

Database Systems

Consistency

- From consistency point of view four levels of consistency can be recognized:
  - Degree 3: a transaction sees degree 3 consistency if:
    - *T* does not overwrite dirty data of other transactions (preventing lost update),
    - *T* does not commit any writes until it completes all its operations — until the end of transaction,
    - *T* does not read dirty data from other transactions,
    - Other transactions do not dirty any data read by *T* before *T* completes.

- Consistency
  - Degree 2: a transaction sees degree 2 consistency if:
    - *T* does not overwrite dirty data of other transactions,
    - *T* does not commit any writes until it completes all its operations — until the end of transaction,
    - *T* does not read dirty data from other transactions.

Database Systems

- Consistency
  - Degree 1: a transaction sees degree 1 consistency if:
    - *T* does not overwrite dirty data of other transactions,
    - *T* does not commit any writes until it completes all its operations – until the end of transaction.
  - Degree 0: a transaction sees degree 0 consistency if:
    - *T* does not overwrite dirty data of other transactions.

Dirty read: Data item whose value is modified by an un-committed transaction

Database Systems

- Isolation
  - To improve performance, we need to interleave operations of transactions running concurrently.
  - Even if consistency and atomicity properties are ensured, undesirable interleaving of operations results in an inconsistent state.
  - Isolation property, guarantees that concurrent transactions are interleaved correctly.

Database Systems

- Isolation
  - Serializability: If several transactions are executed concurrently, the result must be the same as if they were executed serially in an orderly fashion.
  - Incomplete results: Result of an incomplete transactions is not available to other transactions before it is committed.
  - Cascading aborts: In execution of concurrent transactions, attempts must be made to avoid cascading aborts. Cascading aborts happens if a transaction allows other transactions to see its incomplete result before committing and later on deciding to abort.

Database Systems

- Isolation
  - Consider the following two transactions and their possible scheduling orders:

T_1:	T_2:
Read (x)	Read (x)
$x \leftarrow x + 1$	$x \leftarrow x + 1$
Write (x)	Write (x)
Commit	Commit

Database Systems

- Isolation

Correct execution order	Incorrect execution order
T_1: Read (x)	T_1: Read (x)
T_1: x $\leftarrow$ x + 1	T_1: x $\leftarrow$ x + 1
T_1: Write (x)	T_2: Read (x)
T_1: Commit	T_1: Write (x)
T_2: Read (x)	T_2: x $\leftarrow$ x + 1
T_2: x $\leftarrow$ x + 1	T_2: Write (x)
T_2: Write (x)	T_1: Commit
T_2: Commit	T_2: Commit

Database Systems

- Durability
  - After successful termination of a transaction, no system failure should result in a loss of data.
  - The durability properly guarantees that, once a transaction completes successfully, all the updates that it carried out on the database persist.

Database Systems

- Classification
  - Based on different parameters, transactions can be classified:
    - Structure: flat transaction vs. nested transactions.
    - Timing (duration): short life (on-line) transactions vs. long life (batch) transactions, conversational transactions.
    - Application areas: centralized transactions vs. distributed transactions.
    - Organization of read and write actions.

Database Systems

- Classification
  - Flat Transaction: It is a sequence of primitive operations (read, write, commit).
  - Nested transaction: The operations of the transaction may themselves be transactions.

Database Systems

- Flat Transaction

  Begin_transaction Reservation

  ●

  ●

  ●

  end

Database Systems

- Nested transaction

  Begin_transaction Reservation
  •
  •
  •
  Begin_transaction Airline
  •
  •
  •
  end (airline)
  Begin_transaction Hotel
  •
  •
  •
  end (hotel)
  end (Reservation)

Database Systems

- Classification
  - Closed nesting
    - Sub-transactions begin after their parents and finish before the parents. Commitment of a sub-transaction is conditional upon the commitment of the parent.
  - Open nesting
    - Sub-transactions can execute and commit independently. In case of open nesting we may be needing compensating transaction.

Database Systems

- Classification
  - Two-step transaction: All read actions are performed before write actions.
  - Restricted: A data item has to be read before being updated.
  - Restricted two-step: A transaction that is both two-step and restricted.
  - Action Model: A restricted model with additional restriction that each <read, write> pair be executed atomically.

Database Systems

- Classification

General Transaction

T = { R(x), R(y), W(y), R(z), W(x), W(z), W(w), C}

Two-step Transaction

T = { R(x), R(y), R(z), W(x), W(z), W(y), W(w), C}

Restricted Transaction

T = { R(x), R(y), W(y), R(z), W(x), W(z), R(w), W(w), C}

Restricted Two-step Transaction

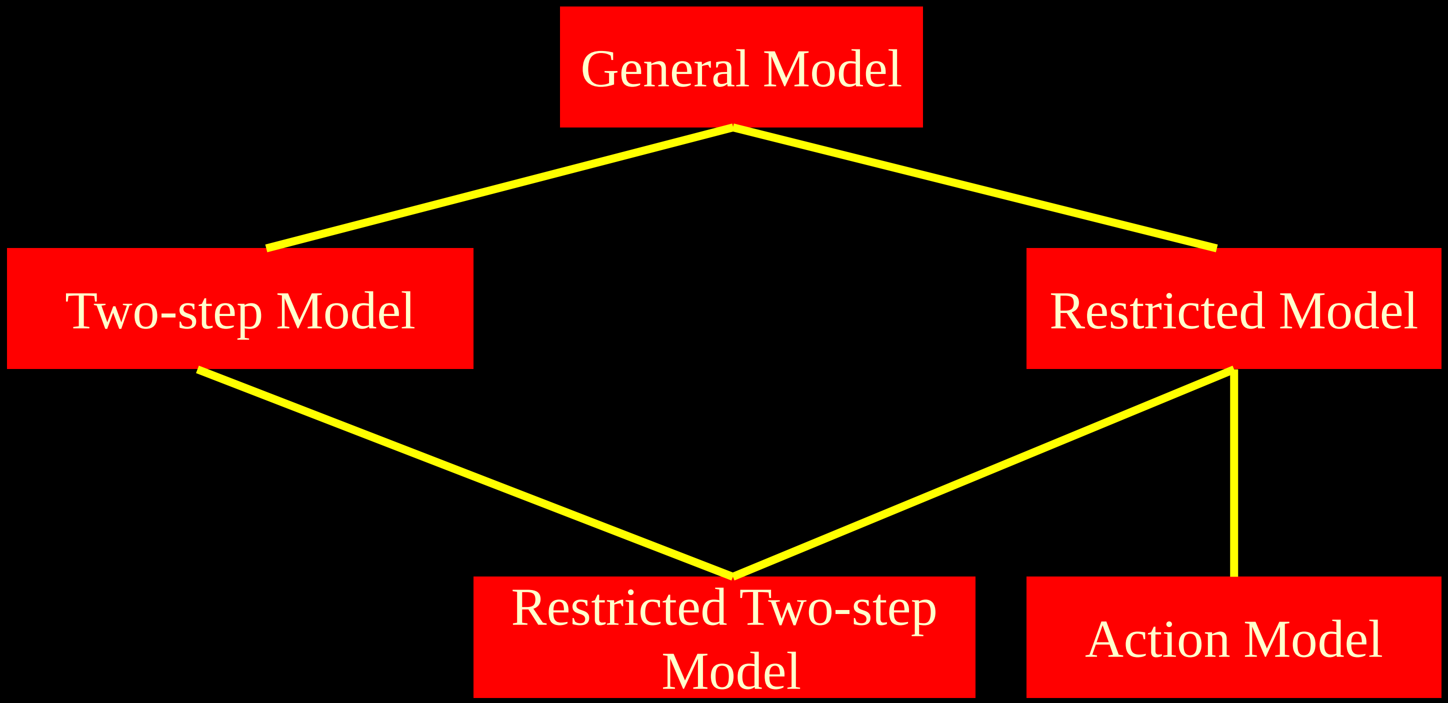
T = { R(x), R(y), R(z), R(w), W(x), W(z), W(y), W(w), C}

Action Transaction

T = { [R(x), W(x)], [R(y), W(y)], [R(z), W(z)], [R(w), W(w)], C}

Database Systems

- Classification



Database Systems

- Transaction states
  - In the absence of failures, we are expecting that a transaction completes successfully.
  - A transaction that completes its execution successfully is said to be committed.
  - A committed transaction, that has updated the database, has transferred database from one consistent state to a new consistent state which must be persisted, even if the system fails (database recovery).

Database Systems

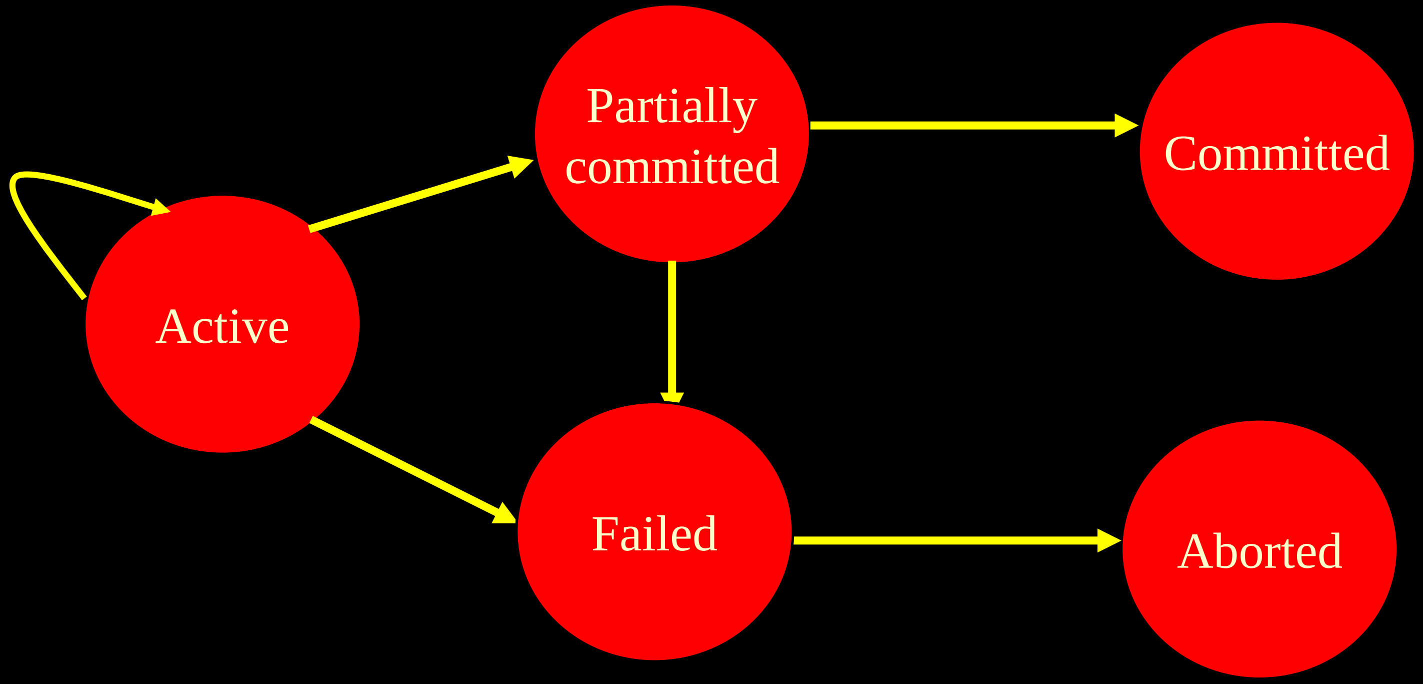
- Transaction states
  - Note, if a transaction is committed, we cannot undo its effect by aborting it – We need a compensating transaction to undo its effect.
  - If a transaction does not complete its execution successfully, to ensure atomicity, it must be aborted – and any change to database must be undone.
  - In case of failure, the transaction must be rolled back.

Database Systems

- Transaction states
  - In general, a transaction is in one of the following states:
    - Active: the transaction stays in this state while executing,
    - Partially committed: the final statement of transaction has been executed,
    - Failed: it is discovered that normal execution can no longer proceed,
    - Aborted: the transaction has been rolled back and state of database has been restored.
    - Committed: successful completion of transaction.

Database Systems

- Transaction states



Database Systems

- It is much easier if internally consistent transactions are run serially — each transaction is executed alone, one after the another. However, there are two good motivations to allow concurrent execution of transactions:
  - Improved throughput and resource utilization
  - Improved average response time.

- Concurrent execution of transactions means that they should be scheduled in order to ensure consistency.

Database Systems

- The concurrency control mechanism attempts to find a suitable trade-off between maintaining the consistency of the database and maintaining a high level of concurrency.
- Note concurrency control deals with the isolation and consistency properties of transactions.

Database Systems

- Two operations (within a transaction or two transactions) are in conflict if their order of execution is important:
  - Read-write,
  - Write-read,
  - Write-write.

Database Systems

- A schedule (history) over a set of transactions $T = \{T_1, T_2, \ldots, T_n\}$ is an interleaved order of execution of these transactions.
- A schedule is a complete schedule, if it defines the execution order of all operations in its domain.

Database Systems

Formally a complete schedule S_T^C over a set of transactions $T = \{T_1, T_2, \ldots, T_n\}$ is a partial order $S_T^C = \{\Sigma_T, \prec_T\}$ where:

$$\Sigma_T = \bigcup_{i=1}^{n} \Sigma_i$$

$$\prec_T \supseteq \bigcup_{i=1}^{n} \prec_i$$

For any two conflicting operations $O_{ij}, O_{kl} \in \Sigma_T$ either $Oij \prec_T O_{kl}$ or $O_{kl} \prec_T O_{ij}$.

1st rule shows that the schedule must contain all operations in participating transactions.
2nd rule shows that the ordering relation on *T* is a superset of ordering relations of individual transactions.
3rd rule shows the execution order among conflicting operations.

Database Systems

- Consider the following two transactions:

T_1:

Read (x)
x $\leftarrow$ x + 1
Write (x)
Commit

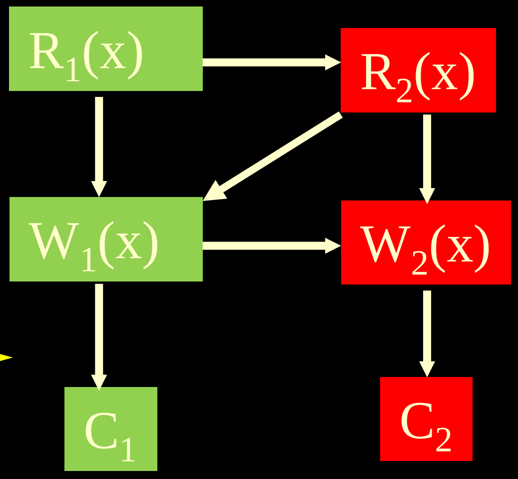
T_2:

Read (x)
x $\leftarrow$ x + 1
Write (x)
Commit

$$S_T^C = \{\sum_T, \prec_T\}$$

Database Systems

$\Sigma_1 = \{ R_1(x), W_1(x), C_1 \}$
$\Sigma_2 = \{ R_2(x), W_2(x), C_2 \}$
$\Sigma_T = \{ R_1(x), W_1(x), C_1, R_2(x), W_2(x), C_2 \}$
$\prec_T = \{(R_1, R_2), (R_1, W_1), (R_1, C_1), (R_1, W_2), (R_1, C_2),$
$(R_2, W_1), (R_2, C_1), (R_2, W_2), (R_2, C_2), (W_1, C_1),$
$(W_1, W_2), (W_1, C_2), (C_1, W_2), (W_2, C_2), (C_1, C_2)\}$



Transitive relationships are omitted for the sake of clarity.

Database Systems

Question

- Consider the following transactions:

T_1:	T_2:	T_3:
Read (x)	Write (x)	Read (x)
Write (x)	Write (y)	Read (y)
Commit	Read (z)	Read (z)
	Commit	Commit

- Define its complete schedule and its corresponding DAG.
- What is an action model transaction?

Database Systems

- Consider the following transactions:

T_1:	Read (A); A := A – 50; Write (A); Read (B); B := B + 50; Write (B);

T_2:	Read (A); temp := A * 0.1; A := A – temp; Write (A); Read (B); B := B + temp; Write (B);

Database Systems

- The following is the serial execution schedule of T_1 followed by 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);
```

Database Systems

- A schedule for a set of transactions must consists all instructions in those transactions.
- A serial schedule consists of a sequence of instructions in transactions, where instructions of one single transaction appear together in that schedule.

Database Systems

- For the following transactions:

T_1:	T_2:	T_3:
Read (x)	Write (x)	Read (x)
Write (x)	Write (y)	Read (y)
Commit	Read (z)	Read (z)
	Commit	Commit

$S = \{W_2(x), W_2(y), R_2(z), C_2, R_1(x), W_1(x), C_1,$
$R_3(x), R_3(y), R_3(z), C_3\}$

Is a serial schedule since T_2 is executed before T_1 and T_1 is executed before T_3:

$$T_2 \prec_S T_1 \prec_S T_3 \Longrightarrow T_2 \rightarrow T_1 \rightarrow T_3$$

Database Systems

- When interleaving instructions from different transactions, one can come up with a number of execution sequence (schedule).
- In this case, we can ensure consistency if the concurrent schedule has the same effect as a serial schedule of transactions – Concurrent schedule is equivalent to a serial schedule.

Database Systems

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);

Database Systems

- Conflict Serializability
  - In two operations of two transactions refer to two different data items, they can be executed in any order. We might have problem if these operations refer to the same data item.

Database Systems

- Conflict Serializability
  - Assume Two transactions T_i and T_j and two instruction $I_i \in T_i$ and $I_j \in T_j$:
    - If I_i = Read(Q) and I_j = Read(Q), the order of I_i and I_j does not matter.
    - If I_i = Read(Q) and I_j = Write(Q), the order of I_i and I_j matters.
    - If I_i = Write(Q) and I_j = Read(Q), the order of I_i and I_j matters.
    - If I_i = Write(Q) and I_j = Write(Q), the order of I_i and I_j matters.
  - I_i and I_j conflicts if they are operations of two different transactions on the same data item and at least one of them is a write operation.

Database Systems

- Conflict Serializability
  - A simplified version of previous transactions.

Read (A); Write (A);	
	Read (A); Write (A);
Read (B); Write (B);	
	Read (B); Write (B);

Database Systems

- Conflict Serializability
  - Two schedules *S* and *S'* are conflict equivalent if *S'* is generated by a series of swaps of non conflicting instructions in *S*.

Read (A);	
Write (A);	
	Read (A);
	Write (A);
Read (B);	
Write (B);	
	Read (B);
	Write (B);

Read (A);	
Write (A);	
	Read (A);
Read (B);	
	Write (A);
Write (B);	
	Read (B);
	Write (B);

Database Systems

- Conflict Serializability
  - Formally, two schedules *S* and *S'* over a set of transactions are conflict equivalent if for each pair of conflicting operations O_{ij}, O_{kl} $(i \neq k)$, whenever $O_{ij} \prec_S O_{kl}$, then $O_{ij} \prec_{S'} O_{kl}$.

Database Systems

- Conflict Serializability
  - Consider the following transactions:

T_1:
Read (x)
Write (x)
Commit

T_2:
Write (x)
Write (y)
Read (z)
Commit

T_3:
Read (x)
Read (y)
Read (z)
Commit

The schedule S' = $\{W_2(x), R_1(x), W_1(x), C_1, R_3(x), W_2(y), R_3(y), R_2(z), C_2, R_3(z), C_3\}$

Is conflict equivalence to schedule

$S = \{W_2(x), W_2(y), R_2(z), C_2, R_1(x), W_1(x), C_1, R_3(x), R_3(y), R_3(z), C_3\}$

Database Systems

- Conflict Serializability
  - Concept of conflict equivalent leads to the concept of conflict serailizability.
  - A schedule *S* is conflict serializable if it is conflict equivalent to a serial schedule.
  - Note that serializability is roughly equivalent to degree 3 consistency discussed before.

Database Systems

- Uncommitted Data (Dirty read (WR conflict))

Assume T_1 transfers 100 from A to B, and T_2 increments both A and B by 6%.

The sequence of operations as scheduled does not generate the same data in A and B as the serial execution of T_1 and T_2, regardless of the order.

T_1	T_2
Read (A); Write (A);	
	Read (A); Write (A); Read (B); Write (B); Commit
Read (B); Write (B); Commit	

Database Systems

- Assume initially A=500 and B=100 and T_1 is executed first. Serial schedule of T_1 and T_2 results in A=424 and B=212. However:

T_1	T_2	A	B
Read (A); Write (A);		400	
	Read (A); Write (A); Read (B); Write (B); Commit	424	106
Read (B); Write (B); Commit			206

Database Systems

- Unrepeatable reads (WR conflict)

T_1 reads two different values for A.

T_1	T_2
Read (A);	
	Read (A);
	Write (A);
Read (A);	

- Lost Update (WR conflict)

Assume T_1 increments A and T_2 decrements A

T_1	T_2
Read (A);	
	Read (A);
Write (A)	
	Write (A);

The sequence of operations as scheduled does not generate the same data in A as the serial execution of T_1 and T_2, regardless of the order.

- Overwriting Uncommitted data (WW conflict (blind write))

Assume T_1 and T_2 are intended to Keep the same values in A and B. Say T_1 sets A and B to 2000 and T_2 sets A and B to 1000.

The sequence of operations as scheduled does not generate the same data in A and B as the serial execution of T_1 and T_2, regardless of the order.

T_2	T_1
Read (A); Write (A);	
	Read (B); Write (B);
Read (B); Write (B);	
	Read (A); Write (A);

Database Systems

- Consider the following schedule:

T_1	T_2
Read (A);	
Write (A);	
	Read (A);
Read (B);	

- Assume we allow T_2 to commit after read(A). Therefore T_2 commits before T_1 does. Now suppose T_1 fails before it commits. Since T_2 has read the value of data item (A) written by T_1, we must abort T_2. However, T_2 has committed and cannot be aborted. Such a schedule is non-recoverable schedule.

Database Systems

- Definition: A recoverable schedule is a schedule that for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, T_i commits before T_j commits.

Database Systems

- Consider the following schedule:

T_1	T_2	T_3
Read (A);		
Read (B);		
Write (A);		
	Read (A);	
	Write (A);	
		Read (A);
abort		

Since T_1 failed, it needs to be rolled back, but T_2 is dependent on T_1 and T_3 is dependent on T_2, so they need to be rolled back, and hence cascading aborts.

Database Systems

- Definition: A schedule is a cascadeless schedule where each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before read operation of T_j.

Database Systems

- Definition: A schedule is a strict schedule in which transactions cannot read or write an item X until the last transaction that wrote X is committed (or aborted).

◆ Scheduling Involving Aborted Transactions

In this case, all actions of T_1 are to be undone. However, T_2 is already committed. If T_2 was not committed by cascading aborts we were able to resolve the situation. Such a schedule is called unrecoverable schedule.

T_1	T_2
Read (A);	
Write (A);	
	Read (A);
	Write (A);
	Read (B);
	Write (B);
	Commit
Abort	

- Serializable schedule

A serializable schedule over a set of T transactions is a schedule whose effect on any consistent database instance is guaranteed to be identical to that of some complete serial schedule over the set of committed transactions in T.

Database Systems

- Testing for Serializability
  - The concept of precedence graph can be used to test serializability.
  - A precedence graph for a schedule *S* is a directed graph $G = (V, E)$, where *V* is the set of vertices each representing a transaction and *E* is the set of directed edges between the vertices.

Database Systems

- Testing for Serializability
  - Assume T_i and $T_j \in V$, then there is an edge between $T_i \rightarrow T_j$ if one of the following conditions holds:
    - If T_i executes Read(Q) before T_j executes Write(Q),
    - If T_i executes Write(Q) before T_j executes Read(Q),
    - If T_i executes Write(Q) before T_j executes Write(Q),

Database Systems

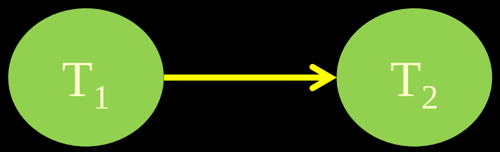
- Testing for Serializability
  - If the precedence graph for a schedule contains a cycle, then this schedule is not conflict serializable, otherwise it is.

Database Systems

- A schedule and its precedence graph

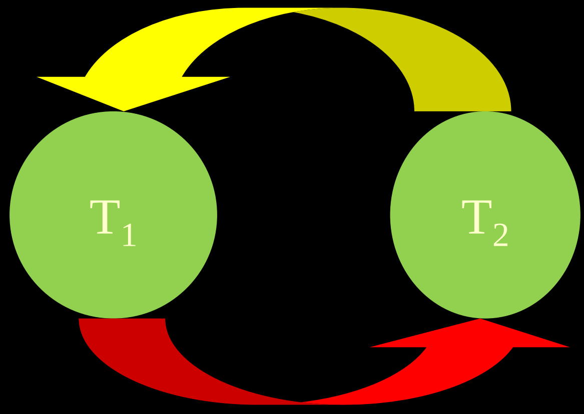
```
Read (A);
A := A – 50;
Write (A);
Read (B);
B := B + 50;
Write (B);
```

```
Read (A);
temp := A * 0.1;
A := A – temp;
Write (A);
Read (B);
B := B + temp;
Write (B);
```



- A schedule and its precedence graph

Read (A);	
A := A – 50;	
	Read (A);
	temp := A * 0.1;
	A := A – temp;
	Write (A);
	Read (B);
Write (A);	
Read (B);	
B := B + 50;	
Write (B);	
	B := B + temp;
	Write (B);



Database Systems

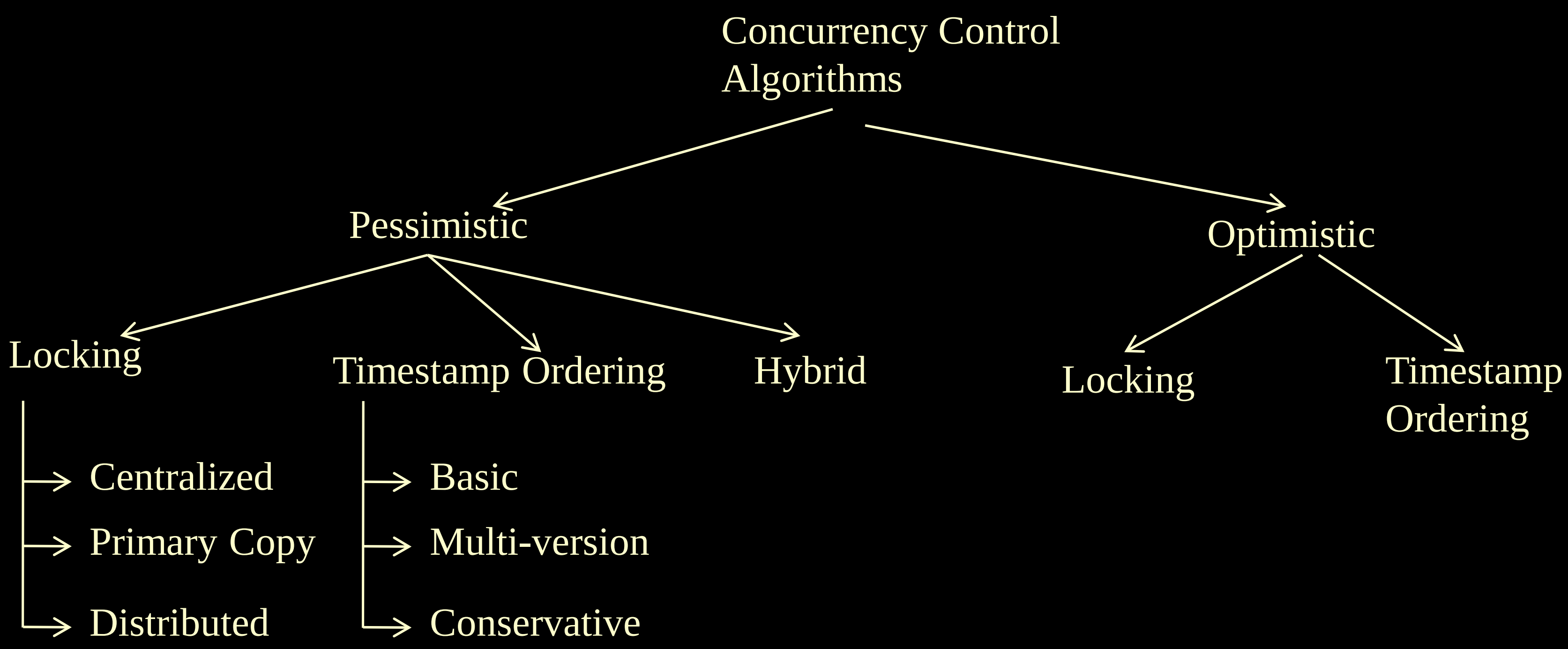
- The primary function of concurrency controller is to generate a serializable schedule for execution of a sequence of transactions – to devise algorithms that guarantee the generation of serializable schedules.

Database Systems

- Concurrency control algorithms' taxonomy
  - Pessimistic algorithms — Synchronizes The concurrent execution early.
  - Optimistic algorithms — delays synchronization until termination.

Database Systems

- Concurrency control algorithms' taxonomy



Database Systems

- Lock-Based Protocol
  - One way to ensure serializability is to require data items to be accessed in a mutual exclusive fashion — while a transaction is accessing the data item, no other transaction can access that data item, i.e., being Locked.

Database Systems

- Lock-Based Protocol
  - There are various type of locks:
    - Shared: if a transaction T_i has obtained a shared-mode lock (lock-s(Q)) on item Q, then T_i can read Q, but cannot write Q.
    - Exclusive: if a transaction T_i has obtained an exclusive-mode lock (lock-x(Q)) on item Q, then T_i can read and write Q.

Database Systems

Lock-Based Protocol

- The following matrix shows the compatibility between different lock modes:

	S	X
S	true	false
X	false	false

Database Systems

- Lock-Based Protocol
  - To access a data item, transaction T_i must first request for a lock on that data item. If the data item is already locked by another transaction in an incompatible mode, the concurrency control manager will not grant the lock until all incompatible locks held by other transactions are released.

Database Systems

- Lock-Based Protocol
  - Assume the following two transactions:

Lock-X (B); Read (B); B := B – 50; Write (B); Unlock (B); Lock-X (A); Read (A); A := A + 50; Write (A); Unlock (A);	Lock-S (A); Read (A); Unlock (A); Lock-S (B); Read (B); Unlock (B); Display (A+B);

◆ Lock-Based Protocol

Incorrect Schedule, why?

Lock-X (B);		
		Grant-X (B, T_1);
Read (B);		
B := B – 50;		
Write (B);		
Unlock (B);		
	Lock-S (A);	
		Grant-S (A, T_2);
	Read (A);	
	Unlock (A);	
	Lock-S (B);	
		Grant-S (B, T_2);
	Read (B);	
	Unlock (B);	
	Display (A+B);	
Lock-X (A);		
		Grant-X (A, T_1);
Read (A);		
A := A + 50;		
Write (A);		
Unlock (A);		

Database Systems

- Lock-Based Protocol
  - Now assume the following similar transactions to the previous ones:

Scheduling these two will not result in a wrong sequence of operations.

Lock-X (B);	Lock-S (A);
Read (B);	Read (A);
B := B – 50;	Lock-S (B);
Write (B);	Read (B);
Lock-X (A);	Display (A+B);
Read (A);	Unlock (A);
A := A + 50;	Unlock (B);
Write (A);	
Unlock (B);	
Unlock (A) ;	

Database Systems

Lock-Based Protocol

- Unfortunately, locking can lead to deadlock, consider the partial schedule of previous transactions.

Lock-X (B);	
Read (B);	
B := B – 50;	
Write (B);	
	Lock-S (A);
	Read (A);
	Lock-S (B);
Lock-X (A);	

Database Systems

- Lock-Based Protocol
  - We will define a set of rules, called locking protocol, to indicate when a transaction may lock and unlock a data item.

Database Systems

Lock-Based Protocol

- Let $\{T_0, T_1, \ldots, T_n\}$ be a set of transactions in a schedule *S*. T_i proceeds T_j in *S*, written $T_i \rightarrow T_j$, if there exist a common data item *Q* such that T_i has held a lock mode *A* on *Q*, and T_j has held a lock mode *B* on *Q* later, and comp (A,B) = false. Then in any equivalent serial schedule T_i must appear before T_j.

- In another words, the precedence rule, implies data dependence between the two transactions. Conflicts between instructions implies incompatibility of lock modes.

Database Systems

- Lock-Based Protocol
  - Within the scope of locking protocol, one has to be concern about starvation. Starvation can be avoided by the concurrency control manager.
  - Assume T_i request a lock on a data item Q in a particular mode M, the lock is granted if:
    - There is no other transaction holding a lock on Q in a mode that conflicts with M.
    - There is no other transaction waiting for a lock on Q and made its lock request before T_i.
  - In short, a lock request will never get blocked by a lock request that is made later.

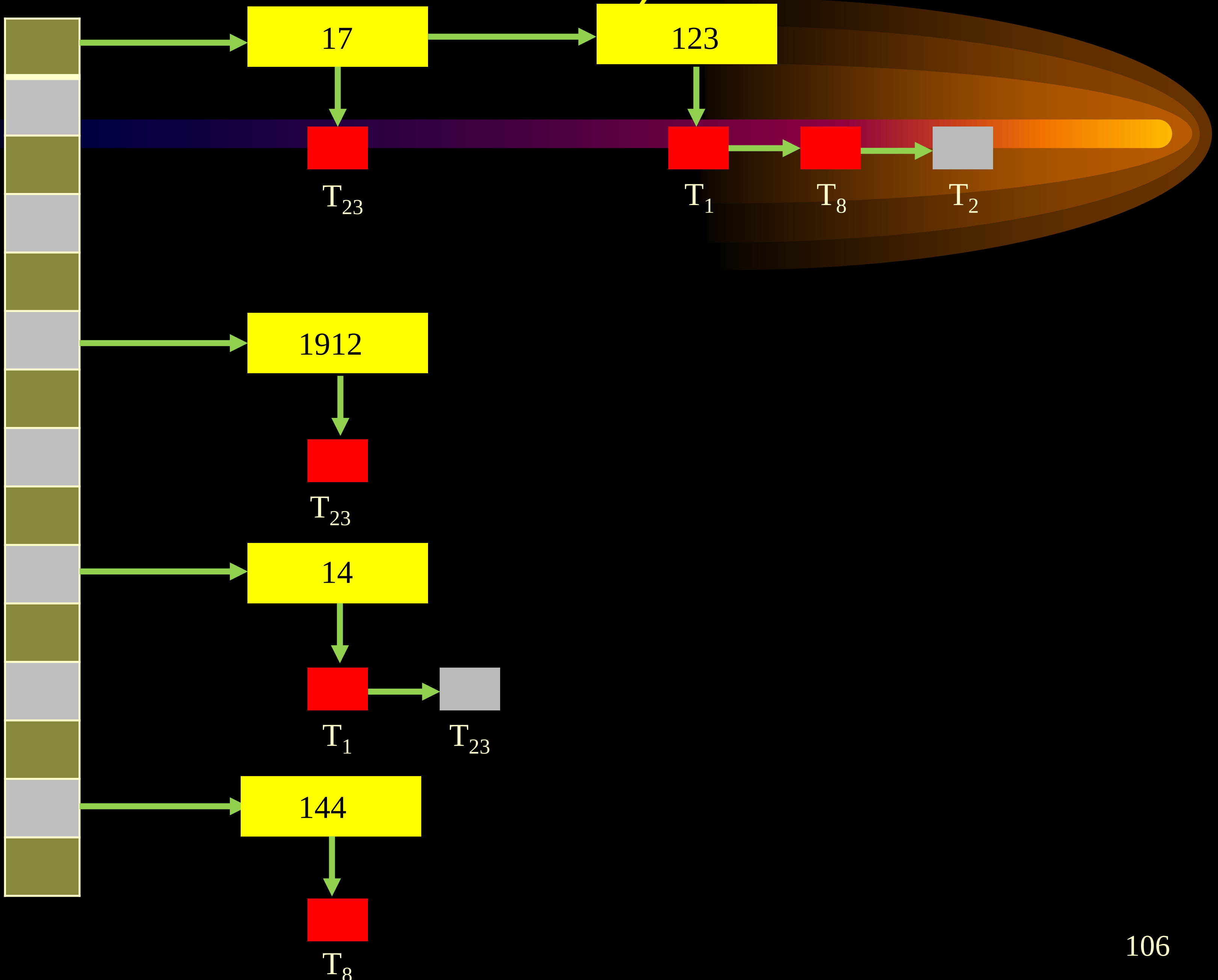
Database Systems

- Implementation
  - A lock manager can be implemented as a process that receives/sends messages from/to transactions.
  - Lock-request messages are responded with lock-grant messages, or messages requesting rollback (in case of deadlock).
  - Un-lock messages will be acknowledged in respond, but may results in a lock-grant message.

Database Systems

- Implementation
  - Lock manager maintains the lock table.
  - Lock table is a hash table that maintains a linked list of records, one for each request, in the order the requests arrive.
  - Each record of the linked list for a data item contains:
    - The transaction identifier,
    - The type of the requested lock mode, and
    - The indicator of whether or not the request is granted.

Database Systems



Database Systems

- In the previous example:
  - Lock table contains locks for five data items (14, 17, 123, 144, and 1912).
  - Granted locks are represented as red squares and waiting locks are represented as grey squares.
  - T_{23} has been granted lock on 1912 and 17, and waiting on 14.

Database Systems

- Implementation
  - When a lock request arrives, a record will be added to the end of the linked list, if it exists, for the data item. Otherwise, a linked list is created.
  - The 1st lock request for a data item is always granted. However, if the data item is already locked, the compatibility between the lock requests is checked by the lock manager. If they are compatible, the request is granted, otherwise it has to wait.

Database Systems

- Implementation
  - When the lock manager receives an unlock message, the record corresponding to that transaction is deleted. Then the lock manager checks to see whether or not the next request can be granted. If so, the request is granted and the next record, if any, is checked for compatibility, and so on.

Database Systems

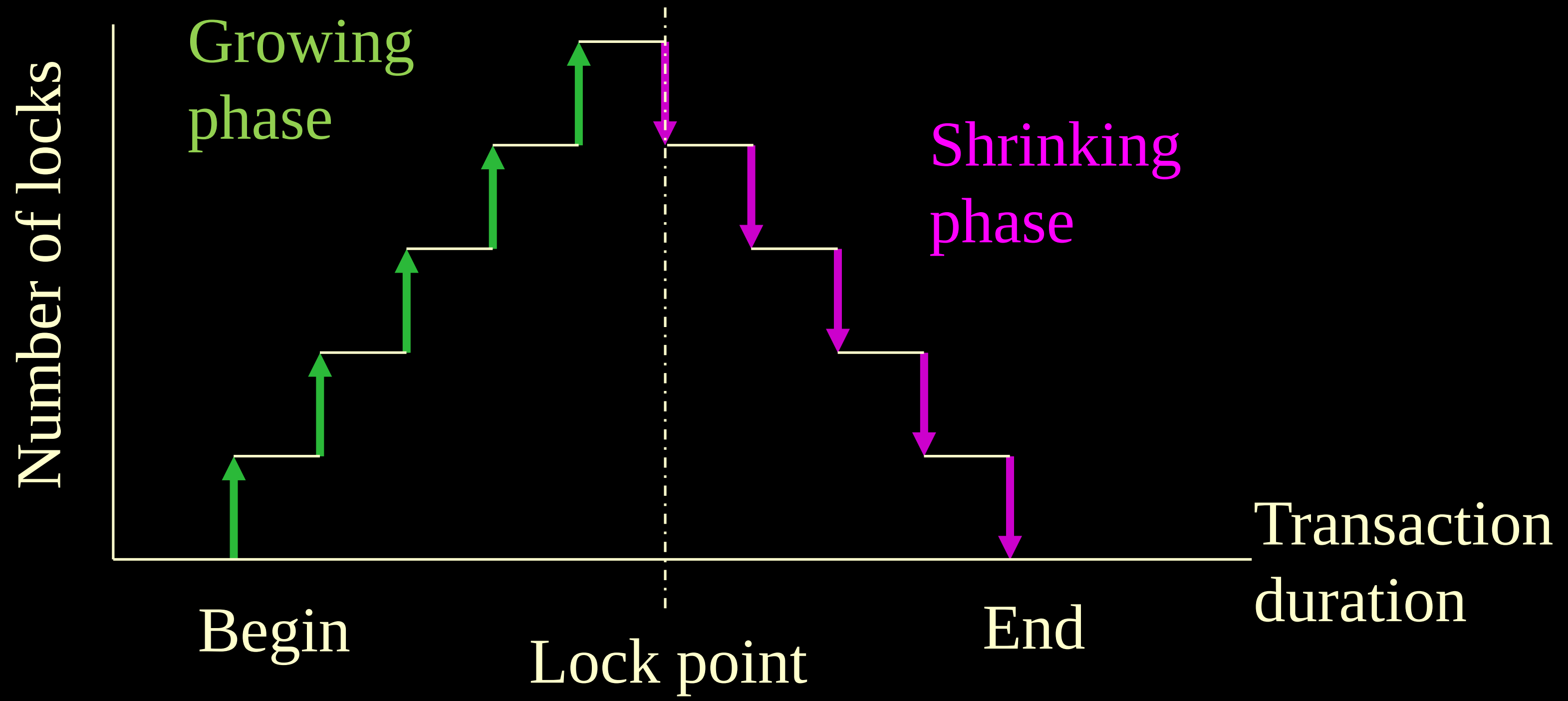
- Implementation
  - If a transaction is aborted, the lock manager deletes any waiting requests made by the transaction. Once the database system took appropriate actions to undo the transaction, all locks held by the aborted transaction is released.

Database Systems

- Two-phase Locking Protocol
  - This protocol ensures serializability, however, it requires that each transaction issue lock and unlock requests in two phases:
    - Growing Phase: A transaction may obtain locks, but may not release any lock.
    - Shrinking phase: A transaction may release locks, but may not obtain any new locks.
  - Initially, a transaction is in the growing phase. It acquires locks as needed. Once it releases a lock, it enters the shrinking phase, and can issue no more lock requests.

Database Systems

- Two-phase Locking Protocol



Database Systems

- Two-phase Locking Protocol
  - The following two transactions are not 2-phase:

Lock-X (B); Read (B); B := B – 50; Write (B); Unlock (B); Lock-X (A); Read (A); A := A + 50; Write (A); Unlock (A);	Lock-S (A); Read (A); Unlock (A); Lock-S (B); Read (B); Unlock (B); Display (A+B);

Database Systems

- Two-phase Locking Protocol
  - The following two transactions are 2-phase:

Lock-X (B);	Lock-S (A);
Read (B);	Read (A);
B := B – 50;	Lock-S (B);
Write (B);	Read (B);
Lock-X (A);	Display (A+B);
Read (A);	Unlock (A);
A := A + 50;	Unlock (B);
Write (A);	
Unlock (B);	
Unlock (A) ;	

Database Systems

- Two-phase Locking Protocol
  - For a 2-phase transaction, the point where the transaction obtains its last lock is called the lock point of the transaction.
  - In a two phase locking protocol, transactions can be scheduled (ordered) based on their lock points.

Database Systems

Two-phase Locking Protocol

- Two phase locking protocol does not ensure freedom from deadlock.

These two transactions are 2-phase, but in this schedule they are deadlocked.

Lock-X (B);	
Read (B);	
B := B – 50;	
Write (B);	
	Lock-S (A);
	Read (A);
	Lock-S (B);
Lock-X (A);	

Database Systems

- Two-phase Locking Protocol
  - A “good” schedule should also be cascadeless. Cascading rollback may occur under two phase locking protocol.
  - Look at the following schedule and the reason why rollback cascading must be enforced.

Database Systems

- Two-phase Locking Protocol

If the first transaction fails after this point

Then the other two Transactions have to be rolled back.

Lock-X (A);		
Read (A);		
Lock-S (B);		
Read (B);		
Write (A);		
Unlock (A);		
	Lock-X (A);	
	Read (A);	
	Write (A);	
	Unlock (A);	
		Lock-S (A);
		Read (A);
•••		

Database Systems

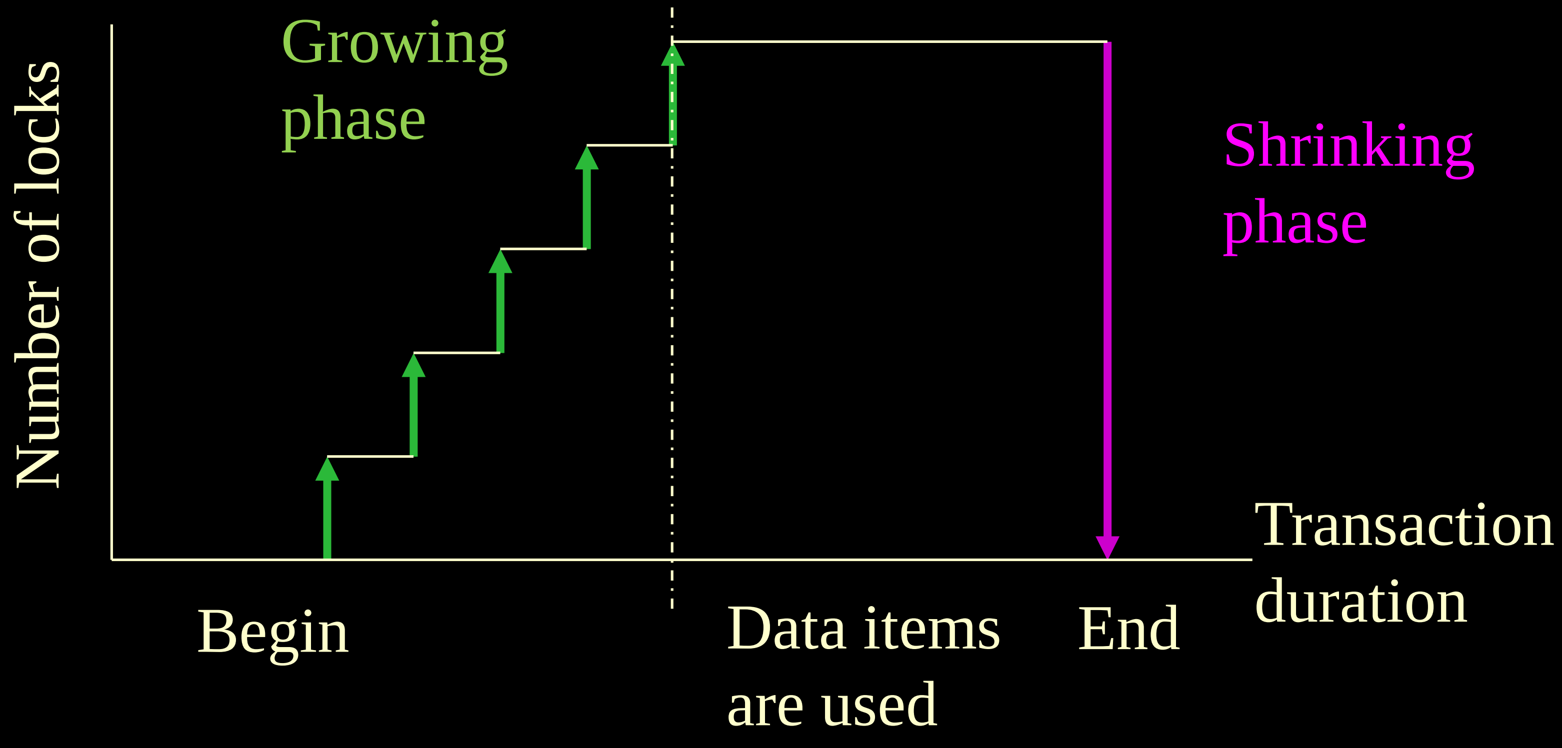
- Two-phase Locking Protocol
  - Two-phase locking protocol can be modified to avoid cascading rollback:
    - Strict two-phase locking protocol,
    - Rigorous two-phase locking protocol.

Database Systems

- Strict two-phase locking protocol
  - Within a two-phase locking protocol, this protocol requires all exclusive-locks be held until transaction commits.
  - Any data written by an uncommitted transaction are locked and unaccessible to any other transactions to read it.

Database Systems

- Strict two-phase locking protocol



Database Systems

- Rigorous two-phase locking protocol
  - Within a two-phase locking protocol, this protocol requires all locks be held until transaction commits.
  - Transactions can be serialized in the order in which they commit.

Database Systems

- Lock conversions
  - The basic two-phase locking can be extended to allow a better performance.
  - Consider the following two transactions:

Read (a_1); Read (a_2); • • • Read (a_n); Write (a_1);	Read (a_1); Read (a_2); Display (a_1+a_2);

Database Systems

Lock conversions

- In a normal two-phase locking protocol, first transaction locks a_1 in exclusive mode, as a result the second transaction must be scheduled after execution of the first one (serial schedule).
- However, first transaction needs exclusive lock on a_1 towards the end of its operations.
- If we allow a_1 to be locked in the shared mode initially, then the second transaction can be scheduled concurrent with the first one.

Database Systems

- Lock conversions
  - The two-phase locking protocol can be extended by allowing the lock conversion.
  - We will allow to upgrade a lock to exclusive mode and downgrade a lock from the exclusive mode.
  - We also impose the following restriction, upgrading can be done during the growing phase and downgrading can be done during the shrinking phase.

Database Systems
Distributed Databases

- Lock conversions

Lock-S (a_1);

Lock-S (a_2);

Lock-S (a_3);
Lock-S (a_4);
•
•
•
Lock-S (a_n);
Upgrade (a_1);

Lock-S (a_1);

Lock-S (a_2);
•
•
•
Unlock (a_1);
Unlock (a_2);

- Note a transaction attempting to upgrade a lock on a data item may be forced to wait if the data item is currently locked by another transaction in shared mode.

Database Systems

- Lock conversions
  - A two-phase locking protocol enhanced by lock conversion generates only conflict serializable schedules (transactions can be serialized based on their lock points).

Database Systems

- Two-phase Locking Protocol
  - For a set of transactions, there may be conflict serializable schedules that cannot be obtained through the two-phase locking protocol.

Database Systems

- Two-phase Locking Protocol
  - A simple automated scheme can be used to generate lock and unlock instructions for an arbitrary transaction:
    - When a transaction issues a read (Q), the system issues a lock-s (Q) instruction followed by the read (Q) instruction.
    - When a transaction issues a write (Q), the system check to see whether the same transaction holds a shared lock on *Q*. If it does, then an upgrade (Q) instruction followed by the write (Q) instruction is issued. Otherwise, a lock-x (Q) followed by write (Q) is issued.
    - All locks obtained by the transaction are unlocked after the transaction commit or aborts.

Database Systems

- Timestamp-based Protocol
  - To each transaction T_i a unique fixed timestamp $TS(T_i)$ is associated. Timestamp could be:
    - The system clock,
    - A logical counter that is incremented each time a timestamp is associated to a transaction.
  - The timestamps of transactions determine the serializability order — if $TS(T_i) < TS(T_j)$ then the system must ensure that the generated schedule is equivalent to a serial schedule in which transaction T_i appears before T_j.

Database Systems

Timestamp-based Protocol

- To implement timestamp-based protocol, two timestamp values are associated with each data item:
  - W-timestamp(Q) denotes the largest timestamp of any transaction that executed Write(Q) successfully.
  - R-timestamp(Q) denotes the largest timestamp of any transaction that executed Read(Q) successfully.
- These values are updated whenever a new Read(Q) or Write(Q) is executed.

Database Systems

- Timestamp-based Protocol
  - In case T_i issues a read(Q):
    - If $TS(T_i) < W\text{-timestamp}(Q)$, T_i should have read the old value of *Q* that has been modified. Hence, read operation is rejected and T_i is rolled back.
    - If $TS(T_i) \geq W\text{-timestamp}(Q)$, the read operation is executed and R-timestamp(Q) is set to the maximum of R-timestamp(Q) and $TS(T_i)$.

- Timestamp-based Protocol
  - In case T_i issues a write(Q):
    - If $TS(T_i) < R\text{-timestamp}(Q)$, the value of *Q* generated by T_i is relatively old, write is rejected and T_i is rolled back.
    - If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is trying to write an outdated value to *Q*, write is rejected and T_i is rolled back.
    - Otherwise, write is executed and W-timestamp(Q) is set to $TS(T_i)$.

Database Systems

- Timestamp-based Protocol
  - Consider the following transactions:

Green transaction	Red transaction
Read (B);	Read (B);
Read (A);	B := B – 50;
Display (A+B);	Write (B);
	Read (A);
	A := A + 50;
	Write (A);
	Display (A+B);

    - Note the timestamp of the green transaction is less than the timestamp of the red transaction.

Database Systems

- Timestamp-based Protocol

Read (B);	
	Read (B);
	B := B – 50;
	Write (B);
Read (A);	
	Read (A);
Display (A+B);	
	A := A + 50;
	Write (A);
	Display (A+B);

Database Systems

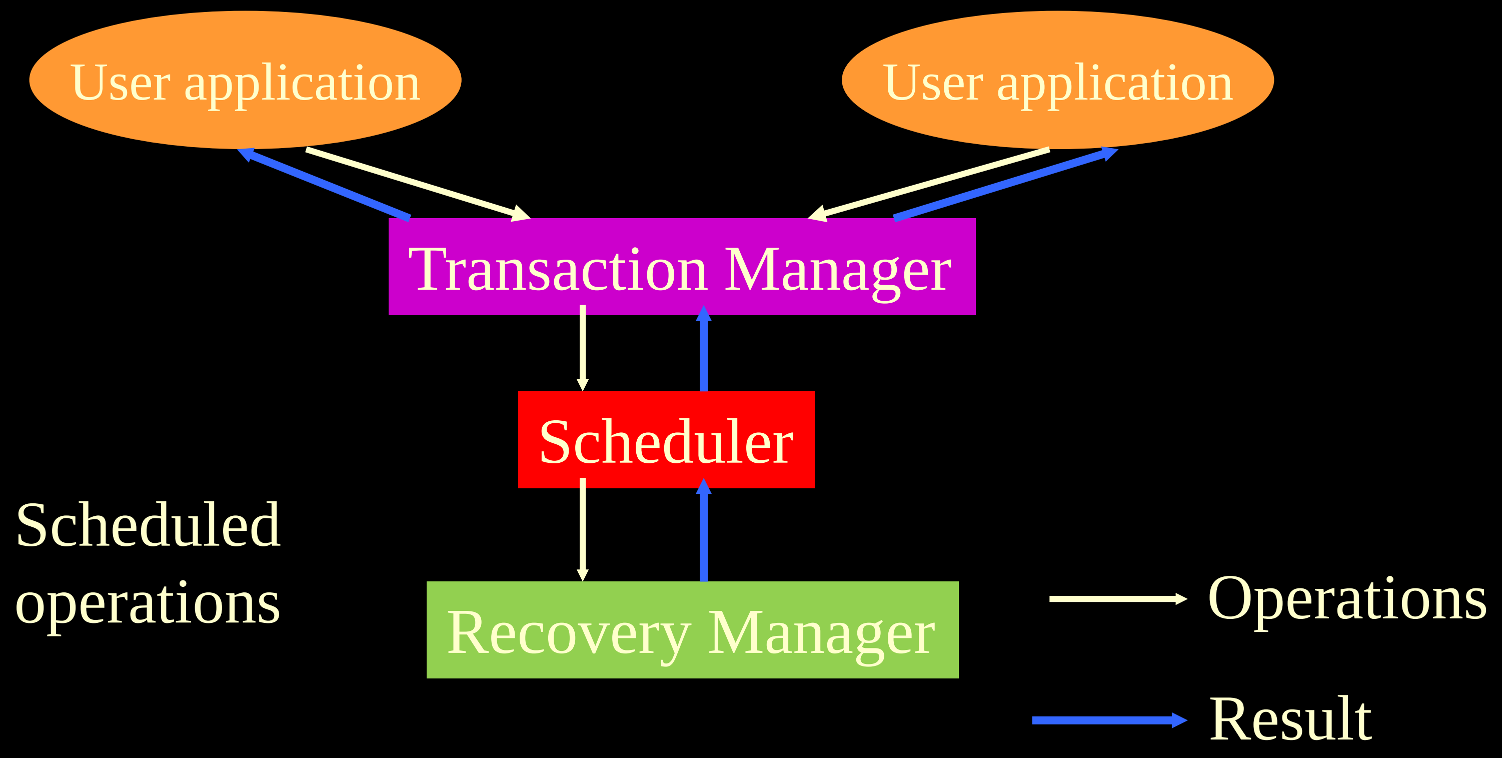
- Timestamp-based Protocol
  - Note there are schedules that are possible under two-phase locking that are not possible under timestamp protocol and vice versa.
  - Timestamp protocol ensures conflict serializability and freedom from deadlock, but it may cause starvation of long transactions by conflicting short transactions.

Database Systems

- Timestamp-based Protocol
  - Timestamp-ordering does not cause deadlock, since transactions never wait while they have access rights to data items.
  - The penalty of deadlock free comes at the expense of potential restart of a transaction again and again.
  - Operations from the scheduler is sent to the database processor one at a time. As long as an operation is not terminated a new one will not be passed on to the processor.

Database Systems

- Centralized Transaction Execution



Database Systems

- Centralized Transaction Execution
  - Transaction Manager is responsible for coordinating the execution of the database operations on behalf of an application.
  - Scheduler is responsible for the implementation of a specific concurrency control algorithm.
  - Recovery manager is responsible to implement procedures that transform database into a consistent state after a failure.

Database Systems

- A schedule is a strict schedule in which transactions cannot read or write an item X until the last transaction that wrote X is committed (or aborted).
- Is S_1: $r_1(x)$; $r_2(z)$; $r_1(z)$; $r_3(x)$; $r_3(y)$; $w_1(x)$; c_1; $w_3(y)$; c_3; $r_2(y)$; $w_2(z)$; $w_2(y)$; c_2; strict?
  - No, T_3 reads x before T_1 commits.
- Is S_2: $r_1(x)$; $r_2(z)$; $r_1(z)$; $r_3(x)$; $r_3(y)$; $w_1(x)$; $w_3(y)$; $r_2(y)$; $w_2(z)$; $w_2(y)$; c_1; c_2; c_3; strict?
  - No, T_3 reads x before T_1 commits.
- Is S_3: $r_1(x)$; $r_2(z)$; $r_3(x)$; $r_1(z)$; $r_2(y)$; $r_3(y)$; $w_1(x)$; c_1; $w_2(z)$; $w_3(y)$; $w_2(y)$; c_3; c_2; strict?
  - No, T_3 reads x before T_1 commits.

Database Systems

- A schedule is a cascadeless schedule where each pair of transactions T_i and T_j such that T_j reads a data item written by T_i, the commit operation of T_i appears before read operation of T_j.
- Is S_1: $r_1(x)$; $r_2(z)$; $r_1(z)$; $r_3(x)$; $r_3(y)$; $w_1(x)$; c_1; $w_3(y)$; c_3; $r_2(y)$; $w_2(z)$; $w_2(y)$; c_2; cascadeless?
  - No, T_3 reads x before T_1 commits.
- Is S_2: $r_1(x)$; $r_2(z)$; $r_1(z)$; $r_3(x)$; $r_3(y)$; $w_1(x)$; $w_3(y)$; $r_2(y)$; $w_2(z)$; $w_2(y)$; c_1; c_2; c_3; cascadeless?
  - No, T_3 reads x before T_1 commits.
- Is S_3: $r_1(x)$; $r_2(z)$; $r_3(x)$; $r_1(z)$; $r_2(y)$; $r_3(y)$; $w_1(x)$; c_1; $w_2(z)$; $w_3(y)$; $w_2(y)$; c_3; c_2; cascadeless?
  - No, T_3 reads x before T_1 commits, or T_2 reads y before T_3 commits.

Database Systems

- A recoverable schedule is a schedule that for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, T_i commits before T_j commits.
- Is S_1: $r_1(x)$; $r_2(z)$; $r_1(z)$; $r_3(x)$; $r_3(y)$; $w_1(x)$; c_1; $w_3(y)$; c_3; $r_2(y)$; $w_2(z)$; $w_2(y)$; c_2; recoverable?
  - If T_1 aborts in highlighted area, then S_1 is recoverable
- S_1: $r_1(x)$; $r_2(z)$; $r_1(z)$; $r_3(x)$; $r_3(y)$; $w_1(x)$; c_1; $w_3(y)$; c_3; $r_2(y)$; $w_2(z)$; $w_2(y)$; c_2;
  - If T_3 aborts in highlighted area, then S_1 is recoverable
- S_1: $r_1(x)$; $r_2(z)$; $r_1(z)$; $r_3(x)$; $r_3(y)$; $w_1(x)$; c_1; $w_3(y)$; c_3; $r_2(y)$; $w_2(z)$; $w_2(y)$; c_2;
  - If T_2 aborts in highlighted area, then S_1 is recoverable

Database Systems

- Is S_2: $r_1(x); r_2(z); r_1(z); r_3(x); r_3(y); w_1(x); w_3(y); r_2(y); w_2(z); w_2(y);$ $c_1; c_2; c_3;$ recoverable?
  - If T_1 aborts in highlighted area, then S_2 is recoverable
- Is S_2: $r_1(x); r_2(z); r_1(z); r_3(x); r_3(y); w_1(x); w_3(y); r_2(y); w_2(z); w_2(y);$ $c_1;$ $c_2; c_3;$ recoverable?
  - If T_2 aborts in highlighted area, then S_2 is recoverable
- Is S_2: $r_1(x); r_2(z); r_1(z); r_3(x); r_3(y); w_1(x); w_3(y); r_2(y); w_2(z); w_2(y); c_1;$ $c_2;$ $c_3;$ recoverable?
  - If T_3 aborts after highlighted area, then S_2 is not recoverable

Database Systems

- Is S_3: $r_1(x)$; $r_2(z)$; $r_3(x)$; $r_1(z)$; $r_2(y)$; $r_3(y)$; $w_1(x)$; c_1; $w_2(z)$; $w_3(y)$; $w_2(y)$; c_3; c_2; recoverable?
  - If T_1 aborts in highlighted area, then S_3 is recoverable
- Is S_3: $r_1(x)$; $r_2(z)$; $r_3(x)$; $r_1(z)$; $r_2(y)$; $r_3(y)$; $w_1(x)$; c_1; $w_2(z)$; $w_3(y)$; $w_2(y)$; c_3; c_2; recoverable?
  - If T_3 aborts in highlighted area, then S_3 is recoverable.
- Is S_3: $r_1(x)$; $r_2(z)$; $r_3(x)$; $r_1(z)$; $r_2(y)$; $r_3(y)$; $w_1(x)$; c_1; $w_2(z)$; $w_3(y)$; $w_2(y)$; c_3; c_2; recoverable?
  - If T_2 aborts after highlighted area, then S_3 is not recoverable since T_2 modifies y to the value before T_3's modification.