Database Management Systems

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Lecture 10, 24 November 2023

Transactions

Sequence of actions that is "one operation"

Ticket booking

- Seat Availability
- Allocation of seat
- Status Update

Money transfer

- Debit one account
- Credit one account

Desirable properties

- Atomicity

All or nothing

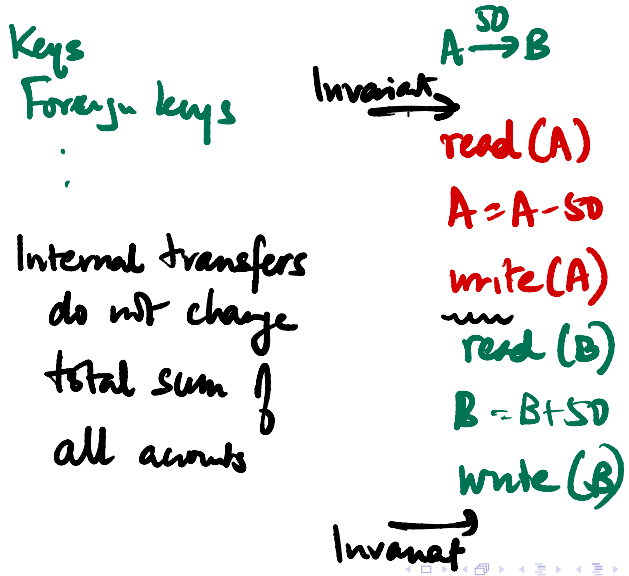
Wrt failures of any kind

"Roll back"

- Internal logical errors
  "Insufficient balance"
- External
  OTP timeout
  :

Desirable properties

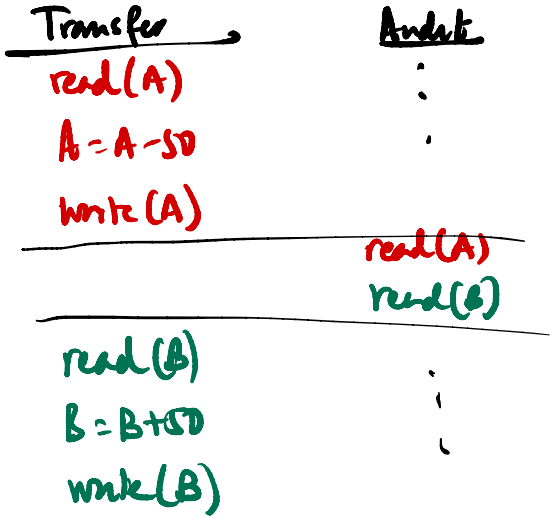
- Atomicity
- Consistency



Desirable properties

- Atomicity
- Consistency
- Isolation

Transactions should "appear" to execute sequentially



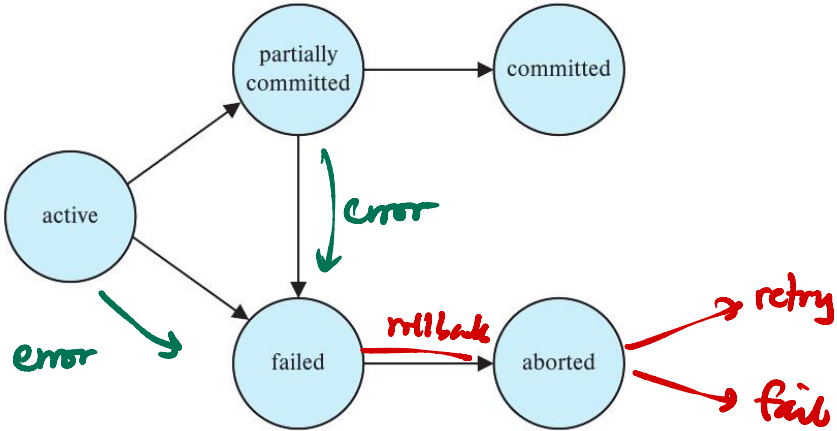
Desirable properties

- Atomicity
- Consistency
- Isolation
- Durability — Effect of a completed transaction is permanent

Desirable properties

- Atomicity
- Consistency
- Isolation
- Durability
- ACID properties

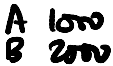
States of a transaction



Transaction logs

- Log each update before it happens — has to be on disk
- Rollback updates in case of failure

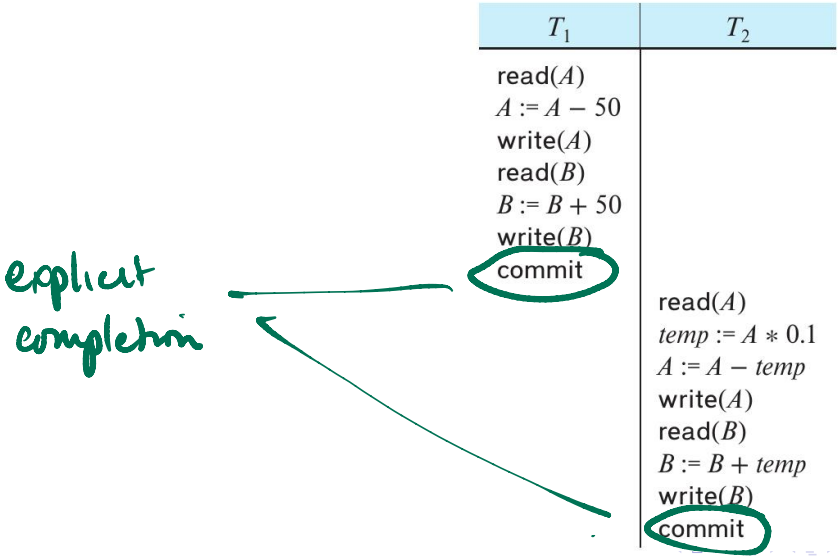
Concurrent execution and schedules



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).

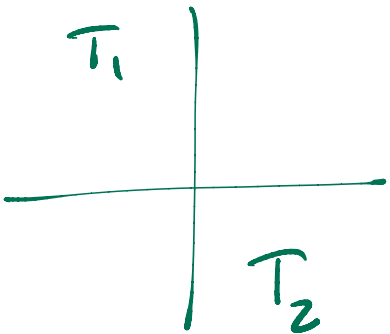
Concurrent execution and schedules



Concurrent execution and schedules

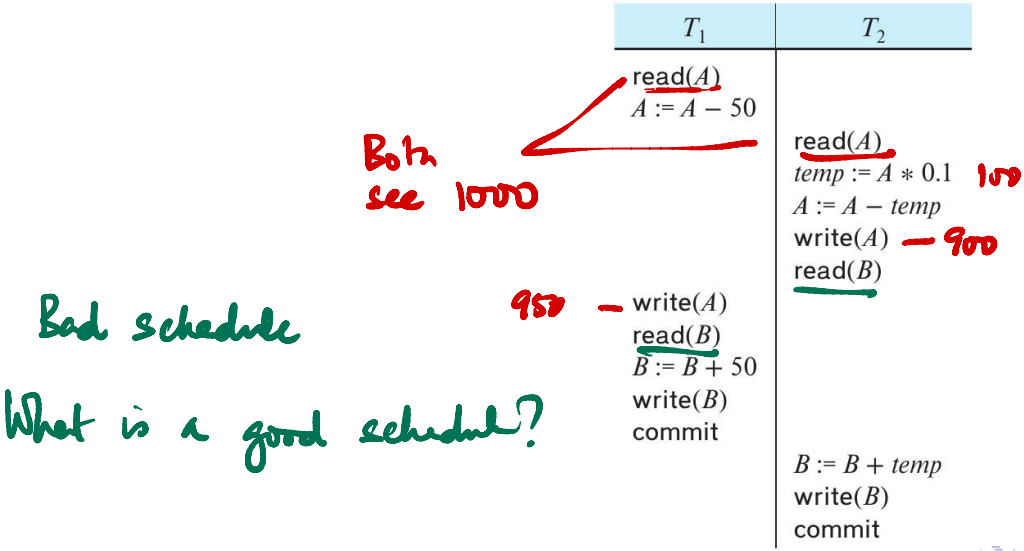
T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	

Concurrent execution and schedules



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

Concurrent execution and schedules



Serializability

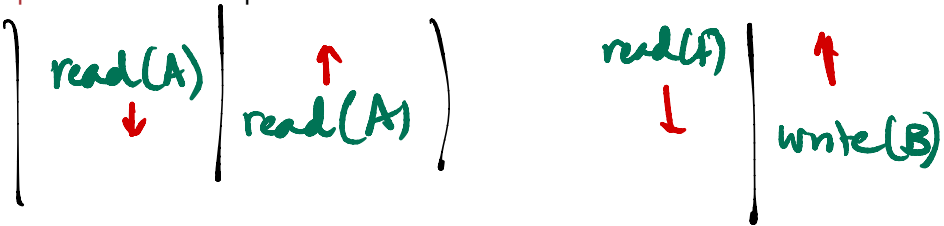
- Serial schedule — each transaction executes as a block, no interleaving

Serializability

- Serial schedule — each transaction executes as a block, no interleaving
- Serializable schedule — equivalent to *some* serial schedule

Serializability

- Serial schedule — each transaction executes as a block, no interleaving
- Serializable schedule — equivalent to *some* serial schedule
- Conflicting operations — two operations on the *same* value where *at least one is a write*



Serializability

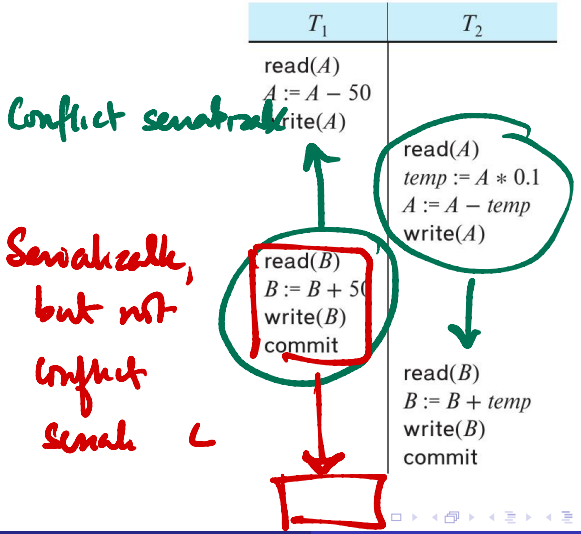
- Serial schedule — each transaction executes as a block, no interleaving
- Serializable schedule — equivalent to *some* serial schedule
- Conflicting operations — two operations on the *same* value where *at least one is a write*
- Conflict equivalence — one schedule can be transformed into the other by reordering non-conflicting operations

Serializability

- Serial schedule — each transaction executes as a block, no interleaving
- Serializable schedule — equivalent to *some* serial schedule
- Conflicting operations — two operations on the *same* value where *at least one is a write*
- Conflict equivalence — one schedule can be transformed into the other by reordering non-conflicting operations
- Conflict serializable — can be reordered to a conflict-equivalent serial schedule

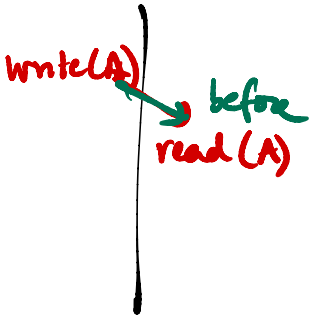
Conflict equivalence

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



Testing for conflict serializability

- Start with a schedule — interleaved sequence of operations from multiple transactions



Testing for conflict serializability

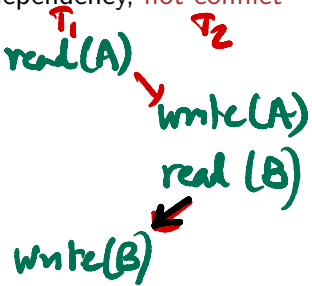
- Start with a schedule — interleaved sequence of operations from multiple transactions
- Build a graph, with transactions as nodes

Testing for conflict serializability

- Start with a schedule — interleaved sequence of operations from multiple transactions
- Build a graph, with transactions as nodes
- Edge $T_i \rightarrow T_j$ if an earlier operation in T_i conflicts with a later operation in T_j

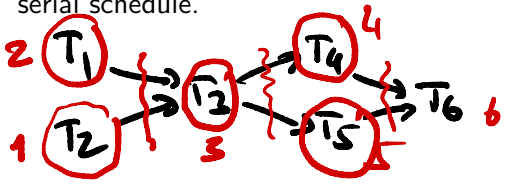
Testing for conflict serializability

- Start with a schedule — interleaved sequence of operations from multiple transactions
- Build a graph, with transactions as nodes
- Edge $T_i \rightarrow T_j$ if an earlier operation in T_i conflicts with a later operation in T_j
- If this conflict graph has cycles, there is a circular dependency, not conflict serializable

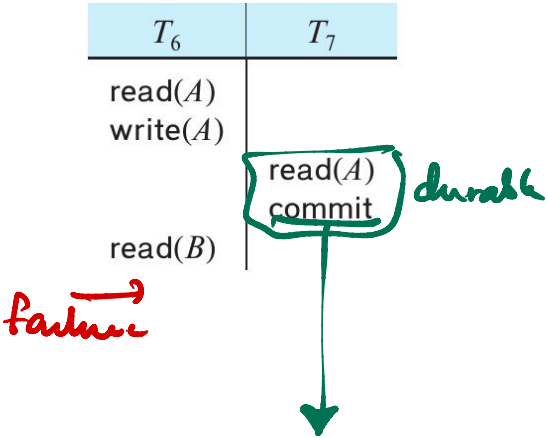


Testing for conflict serializability

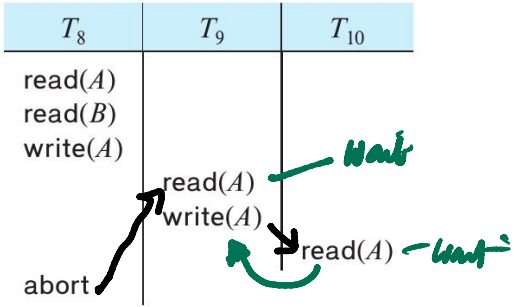
- Start with a schedule — interleaved sequence of operations from multiple transactions
- Build a graph, with transactions as nodes
- Edge $T_i \rightarrow T_j$ if an earlier operation in T_i conflicts with a later operation in T_j
- If this conflict graph has cycles, there is a circular dependency, not conflict serializable
- If the conflict graph is acyclic, use topological sort to order the transactions into a serial schedule.



Recoverable schedules



Cascading rollbacks



Cascadeless schedules

- If T_j reads data written by T_i, T_i commits before the read of T_j

"Dirty write"

Transactions in SQL

- `START TRANSACTION`, `COMMIT`, `ROLLBACK`

Transactions in SQL

- `START TRANSACTION`, `COMMIT`, `ROLLBACK`
- Isolation levels

Transactions in SQL

- `START TRANSACTION`, `COMMIT`, `ROLLBACK`
- Isolation levels
  - Serializable

Transactions in SQL

- `START TRANSACTION`, `COMMIT`, `ROLLBACK`
- Isolation levels
  - Serializable
  - Read committed

Transactions in SQL

- `START TRANSACTION`, `COMMIT`, `ROLLBACK`
- Isolation levels
  - Serializable
  - Read committed
  - Read uncommitted

Transactions in SQL

- `START TRANSACTION`, `COMMIT`, `ROLLBACK`
- Isolation levels
  - Serializable
  - Read committed
  - Read uncommitted
  - Repeatable read

Transactions in SQL

- `START TRANSACTION`, `COMMIT`, `ROLLBACK`
- Isolation levels
  - Serializable
  - Read committed
  - Read uncommitted
  - Repeatable read
  - `SET TRANSACTION ISOLATION LEVEL READ COMMITTED`

Concurrency control

- Ensure that only serializable schedules are generated
- Allow concurrency
- Control access to data to avoid conflicts

"Lock"

Good locking protocols that (1) Allow concurrency (2) Only permit serializable schedules