Database Management Systems

Madhavan Mukund

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

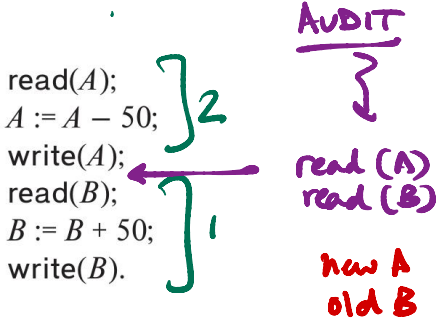
Sai University

Lecture 20, 8 November 2023

Transactions — ACID properties

- Atomicity
- Consistency
- Isolation
- Durability

All or nothing



Atomicity vs isolation

Transactions — ACID properties

- Atomicity
- Consistency
- Isolation
- Durability

Keys
Referential integrity
Value checks ...

Invariant →

read(A);
$A := A - 50$;
write(A);
read(B);
$B := B + 50$;
write(B).

Invariant →

Application constraints

AUDIT – Balance total must not change

Transactions — ACID properties

- Atomicity
- Consistency
- Isolation
- Durability

read(A);
$A := A - 50$;
write(A);
read(B);
$B := B + 50$;
write(B).

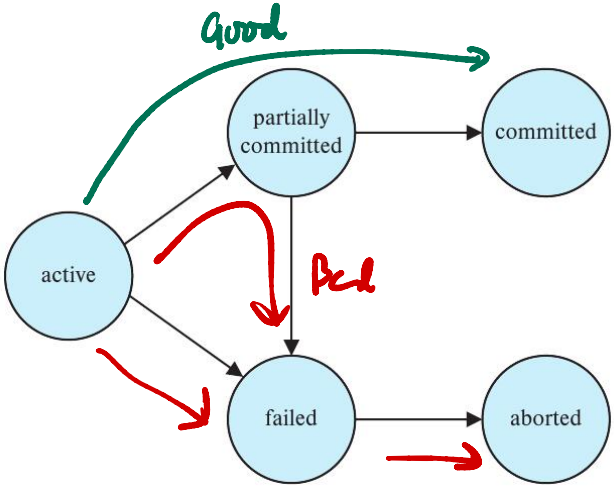
All updates on disk persist

"Committed"

States of a transaction

- After abort, restart or kill
- Be careful about observable external writes

Dispense cash
Send SMS
⋮



Transaction logs

- Log each update before it happens
- Rollback updates in case of failure

Concurrent execution and schedules

T_1: read(A);
$A := A - 50$; 1
write(A); 2
read(B); 4
$B := B + 50$; 6
write(B). 10

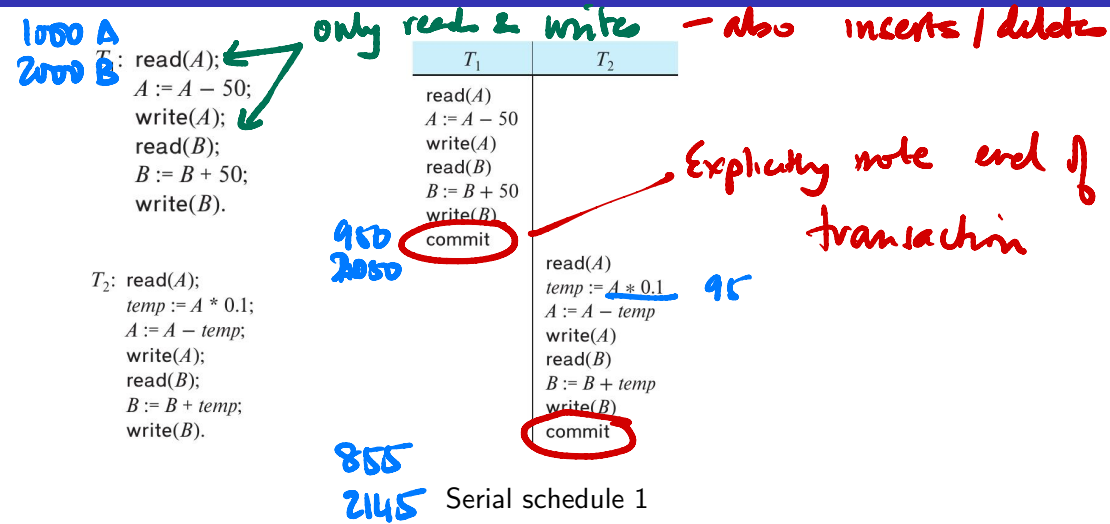
T_2: read(A);
$temp := A * 0.1$; 3
$A := A - temp$; 5
write(A); 7
read(B); 8
$B := B + temp$; 9
write(B). 11

Schedule

Sequence of 'low level' operations

How do ops interleave?

Concurrent execution and schedules



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).

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	

Serial schedule 2

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).

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

Serial schedule 1

equivalent

T_1	T_2
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)	
commit	
	read(B)
	$B := B + temp$
	write(B)
	commit

Consistent concurrent schedule

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).

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

Serial schedule 1

T_1	T_2
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)	
commit	
	read(B)
	$B := B + temp$
	write(B)
	commit

Interleave operations
Internally preserve order

Consistent concurrent schedule

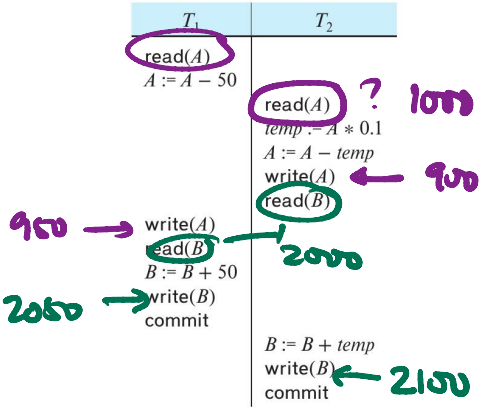
Concurrent execution and schedules

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).

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

Serial schedule 1



Inconsistent concurrent schedule

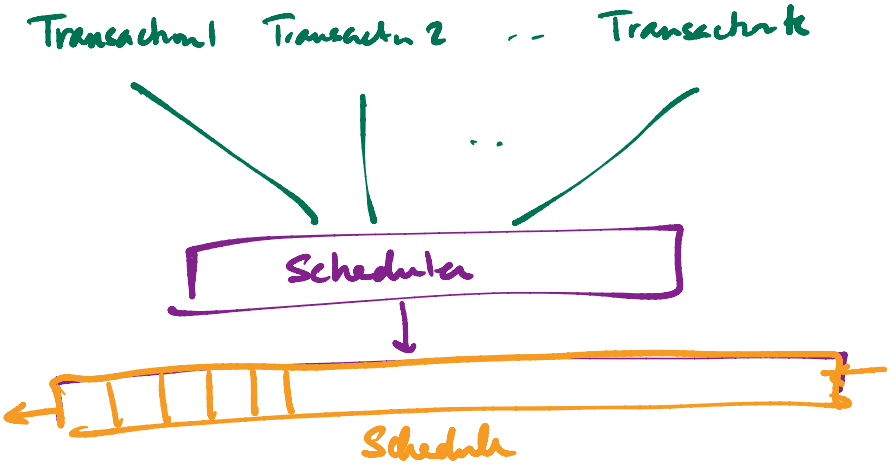
Serializability

A concurrent schedule is consistent if it is equivalent to some serial schedule

Said to Serializable

Given a concurrent schedule — is it serializable?

Serializability



Conflict serializability

Detecting if a schedule is serializable

Problem? Interference of reads & writes of same value

Two concurrent reads do not interfere

T1: write(A)
T2: read(A)
$\neq$
T2: read(A)
T1: write(A)
$\parallel$ conflict

Conflict serializability

Also

T1: write(A)
T2: write(A)
overwrite

vs

T1: read(A)
T2: read(A)
interchangeable

Conflicting pairs of actions

T1: read(A) T1: write(A) T1: write(A)
T2: write(A) T2: read(A) T2: write(A)

Conflict serializability

Swap adjacent operations that do not conflict

- read(A) read(A)
- read(A) write(B) – different data

Repeatedly swap non conflicting ops to get a serial schedule

Conflict serializability

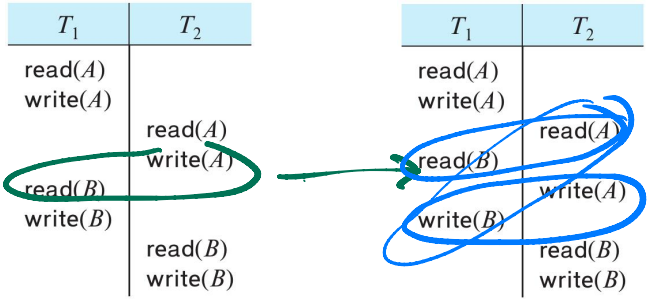
Transfer 5% Transfer 10%

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

conflict

? Interchangeable

Conflict serializability



Conflict serializability

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

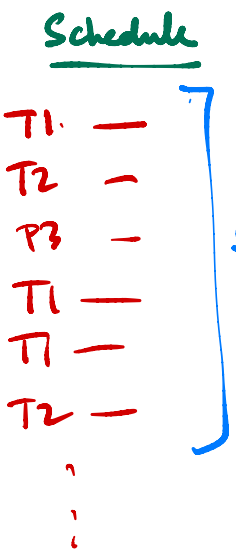
3 swaps

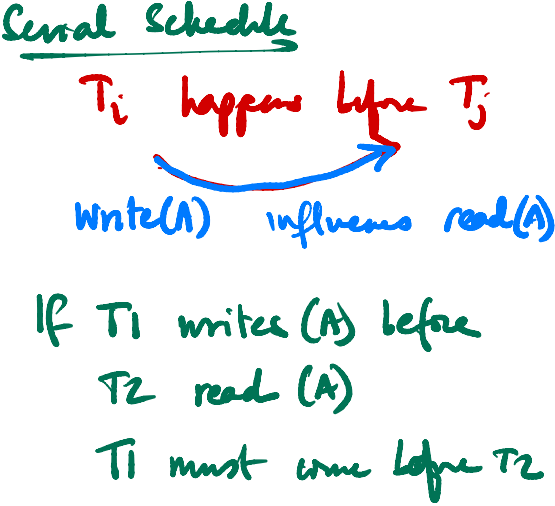
T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Sufficient, not necessary

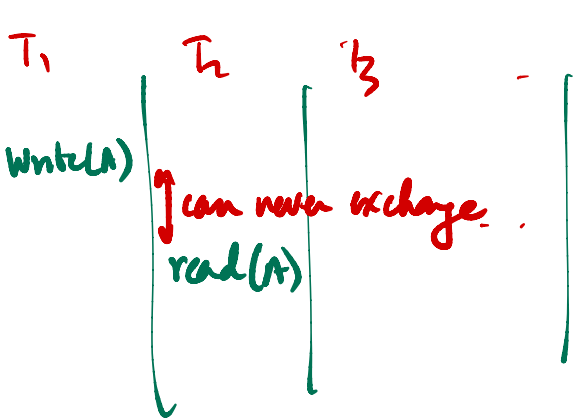
Serial

Testing for conflict serializability





Testing for conflict serializability



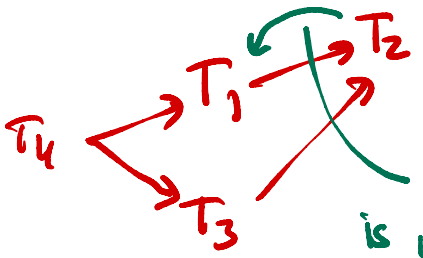
Record

$T_1 \longrightarrow T_2$

Testing for conflict serializability

Edge $T_i \rightarrow T_j$ if some earlier op in T_i conflicts some later op in T_j

Combine all such constraints in a graph



T_1 read(A)

T_2 write(A)

T_2 read(B)

T_3 write(B)

Testing for conflict serializability

Conflict graph G

- If G has a cycle – no consistent serial schedule exists

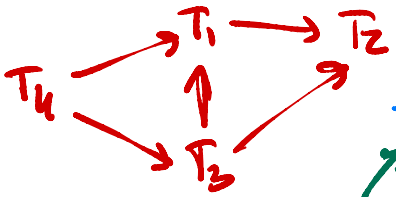
- If G is acyclic – it is conflict serializable

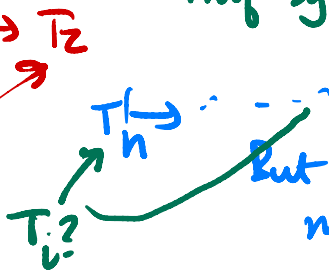
Testing for conflict serializability

Directed acyclic graph

There must be a node with no incoming edge

Proof by contradiction





$T_i' \to T_2' \to T_1'$

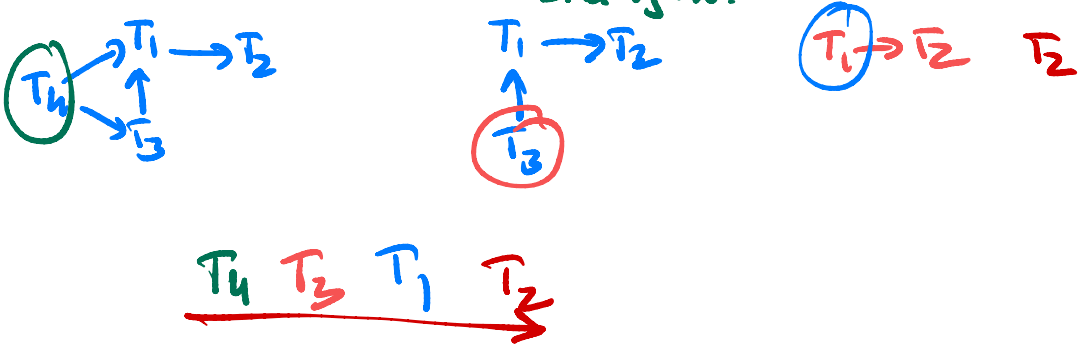
But there a fixed number of nodes, n

Testing for conflict serializability

DAG - directed acyclic graph

Enumerate a "starting" node & delete it

DAG again!



Testing for conflict serializability

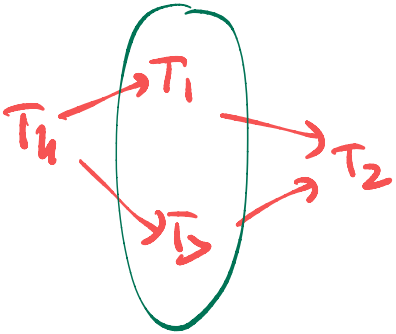
Any DAG can be "sequentialized" respecting dependencies

"Topological Sort"

Algorithms

1. Check if a directed graph has cycles
2. If DAG, generate a topological sort

Testing for conflict serializability



T_4 T_3 T_1 T_2

T_4 T_1 T_3 T_2