CS216: Data-Intensive Computing Systems

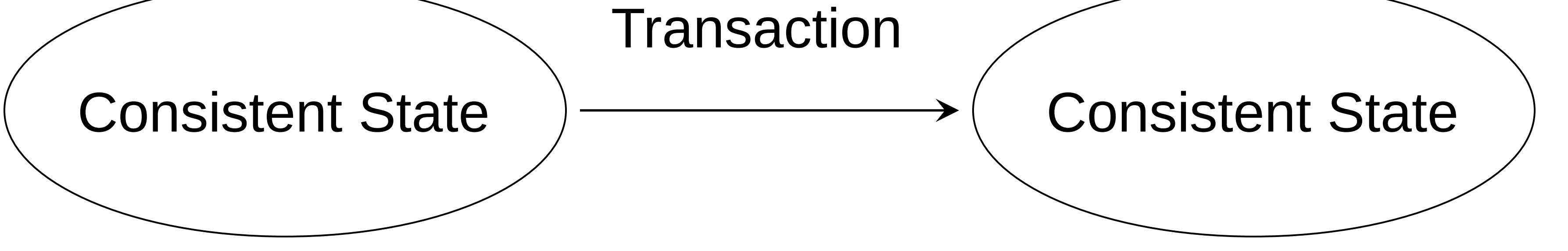
Concurrency Control

Shivnath Babu

Transaction

- Programming abstraction
- Implement real-world transactions
  - Banking transaction
  - Airline reservation

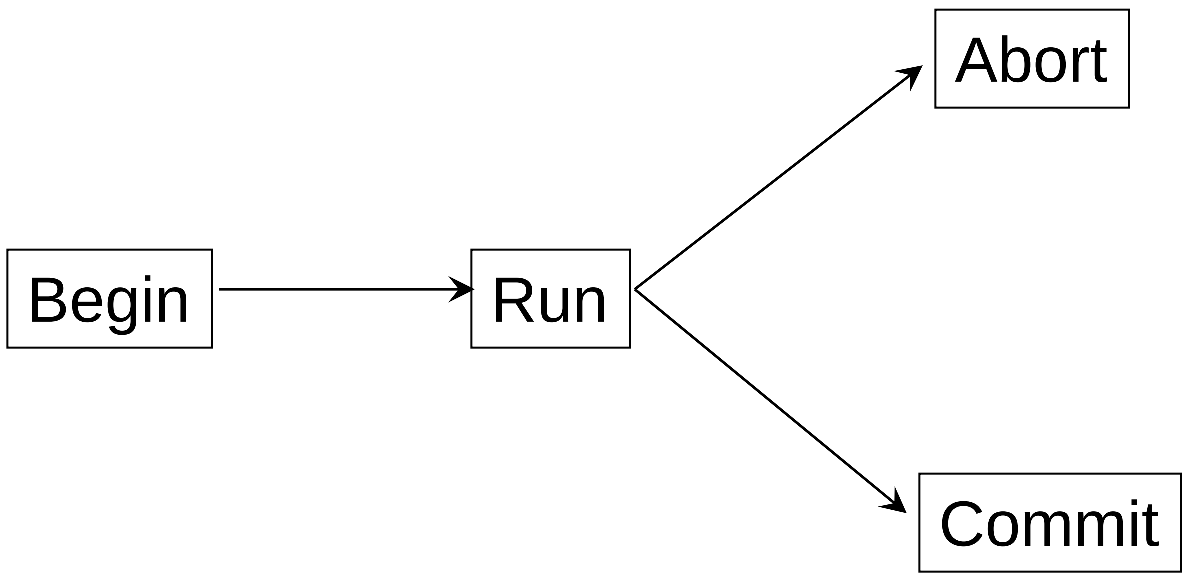
Transaction: Programmer's Role



Transaction: System's Role

- Atomicity
  - All changes of the transaction recorded or none at all
- Durability
  - All future transactions see the changes made by this transaction if it completes
- Isolation
  - Net effect as if the transaction executed in isolation

Transaction: States



Transactions

- Historical note:
  - Turing Award for Transaction concept
  - Jim Gray (1998)
- Interesting reading:

Transaction Concept: Virtues and Limitations
by Jim Gray

http://www.hpl.hp.com/techreports/tandem/TR-81.3.pdf

Context

- We have seen:
  - Ensure atomicity in presence of failures
- Next:
  - Ensure Isolation during concurrency

Issues with Concurrency: Example

Bank database: 3 Accounts

Account Balances

A = 500

B = 500

C = 500

Property: A + B + C = 1500

Money does not leave the system

Issues with Concurrency: Example

Transaction T1: Transfer 100 from A to B

A = 500, B = 500, C = 500 ⟶

Read (A, t)

t = t - 100

Write (A, t)

Read (B, t)

t = t + 100

Write (B, t)

A = 400, B = 600, C = 500 ⟶

Issues with Concurrency: Example

Transaction T2: Transfer 100 from A to C

Read (A, s)

s = s - 100

Write (A, s)

Read (C, s)

s = s + 100

Write (C, s)

Transaction T1	Transaction T2	A	B	C
		500	500	500
Read (A, t)				
t = t - 100				
	Read (A, s)			
	s = s - 100			
	Write (A, s)	400	500	500
Write (A, t)		400	500	500
Read (B, t)				
t = t + 100				
Write (B, t)		400	600	500
	Read (C, s)			
	s = s + 100			
	Write (C, s)	400	600	600

400 + 600 + 600 = 1600

Transaction T1	Transaction T2	A	B	C
		500	500	500
Read (A, t)				
t = t - 100				
Write (A, t)		400	500	500
	Read (A, s)			
	s = s - 100			
	Write (A, s)	300	500	500
Read (B, t)				
t = t + 100				
Write (B, t)		300	600	500
	Read (C, s)			
	s = s + 100			
	Write (C, s)	300	600	600

300 + 600 + 600 = 1500

Terminology

- Schedule:
  - The exact sequence of (relevant) actions of one or more transactions

Problems

- Which schedules are "correct"?
  - Mathematical characterization
- How to build a system that allows only "correct" schedules?
  - Efficient procedure to enforce correctness

Correct Schedules: Serializability

- Initial database state is consistent
- Transaction:
  - consistent state $\rightarrow$ consistent state
- Serial execution of transactions:
  - Initial state $\rightarrow$ consistent state
- Serializable schedule:
  - A schedule equivalent to a serial schedule
  - Always "correct"

Serial Schedule

	T1	T2	A	B	C
	Read (A, t)		500	500	500
	t = t - 100				
T1	Write (A, t)				
	Read (B, t)				
	t = t + 100				
	Write (B, t)		400	600	500
		Read (A, s)			
		s = s - 100			
		Write (A, s)			
T2		Read (C, s)			
		s = s + 100			
		Write (C, s)	300	600	600

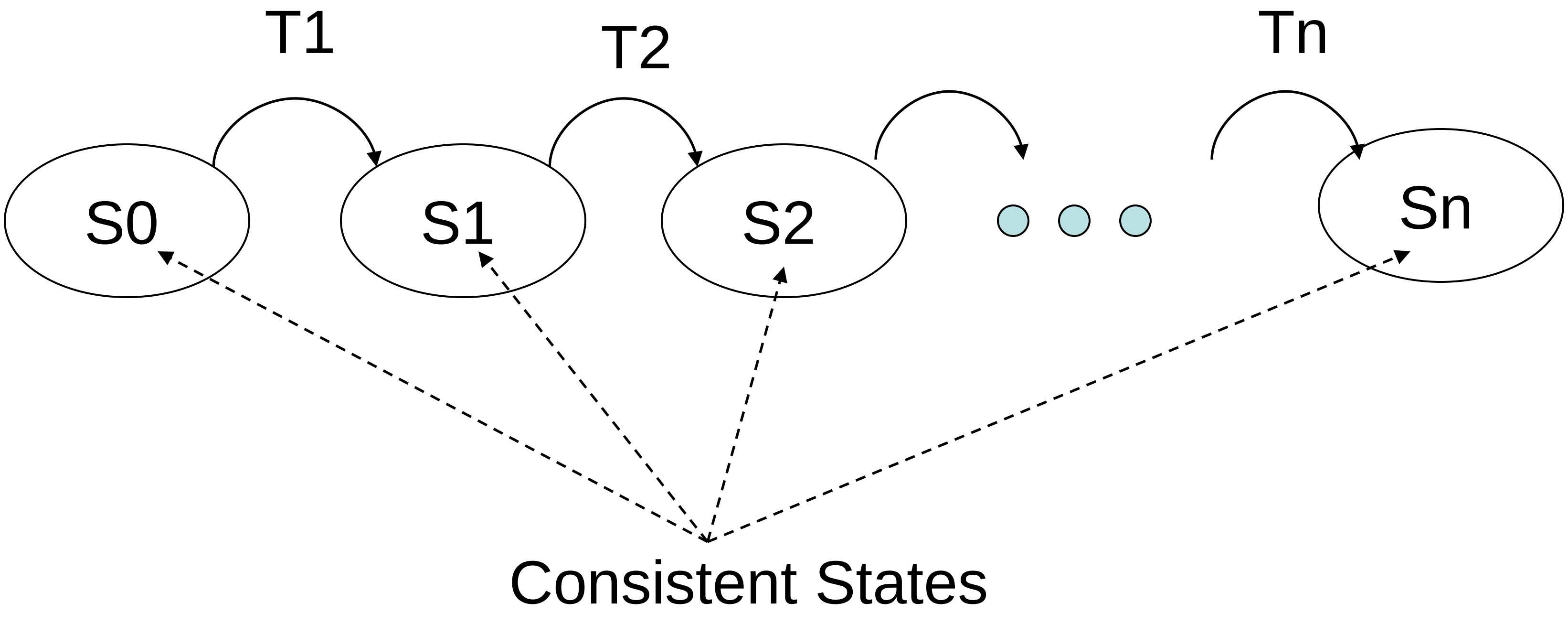
300 + 600 + 600 = 1500

Serial Schedule

	T1	T2	A	B	C
T2		Read (A, s)	500	500	500
		s = s - 100			
		Write (A, s)			
		Read (C, s)			
		s = s + 100			
		Write (C, s)	400	500	600
T1	Read (A, t)				
	t = t - 100				
	Write (A, t)				
	Read (B, t)				
	t = t + 100				
	Write (B, t)		300	600	600

300 + 600 + 600 = 1500

Serial Schedule



Is this Serializable?

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
Write (A, t)	
	Read (A, s)
	s = s - 100
	Write (A, s)
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (C, s)
	s = s + 100
	Write (C, s)

Equivalent Serial Schedule

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
Write (A, t)	
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (A, s)
	s = s - 100
	Write (A, s)
	Read (C, s)
	s = s + 100
	Write (C, s)

Is this Serializable?

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
	Read (A, s)
	s = s - 100
	Write (A, s)
Write (A, t)	
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (C, s)
	s = s + 100
	Write (C, s)

No. In fact, it leads to inconsistent state

Is this Serializable?

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
	Read (A, s)
	s = s - ~~100~~ 0
	Write (A, s)
Write (A, t)	
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (C, s)
	s = s + ~~100~~ 0
	Write (C, s)

Is this Serializable?

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
	Read (A, s)
	s = s - 0
	Write (A, s)
Write (A, t)	
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (C, s)
	s = s + 0
	Write (C, s)

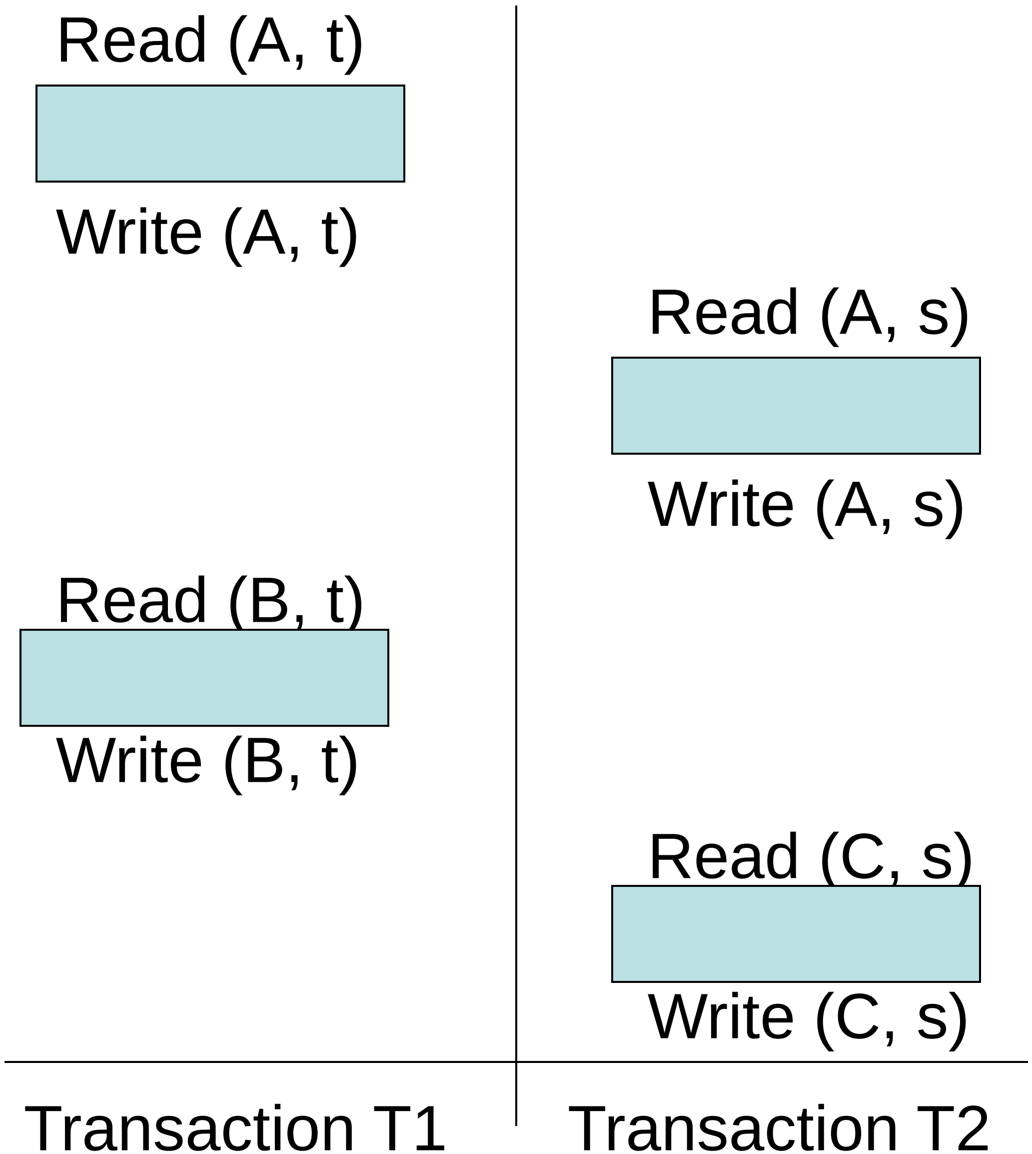
Yes, T2 is no-op

Serializable Schedule

Transaction T1	Transaction T2
Read (A, t)	
t = t - 100	
	Read (A, s)
	s = s - 0
	Write (A, s)
Write (A, t)	
Read (B, t)	
t = t + 100	
Write (B, t)	
	Read (C, s)
	s = s + 0
	Write (C, s)

Serializability depends on code details

Serializable Schedule



Still Serializable!

Serializability

- General Serializability:
  – Hard to determine
- Goal: weaker serializability
  – Determined from database operations alone
- Database Operations:
  – Reads, Writes, Inserts, …

Simpler Notation

$r_T(X)$ Transaction T reads X

$w_T(X)$ Transaction T writes X

What is X in r (X)?

- X could be any component of a database:
  - Attribute of a tuple
  - Tuple
  - Block in which a tuple resides
  - A relation
  - …

New Notation: Example Schedule

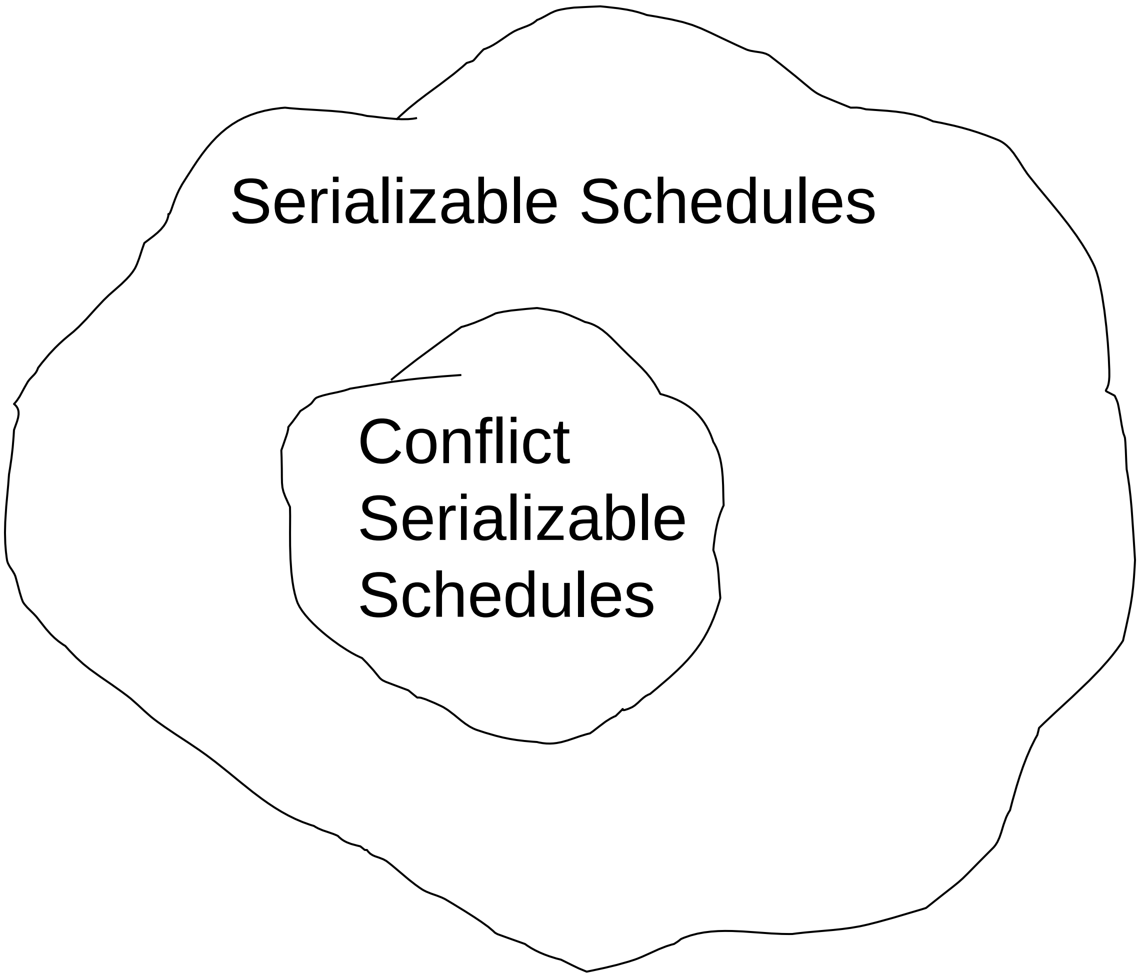
r1(A) w1(A) r2(A) w2(A) r1(B) w1(B) r2(B) w2(B)

→ time

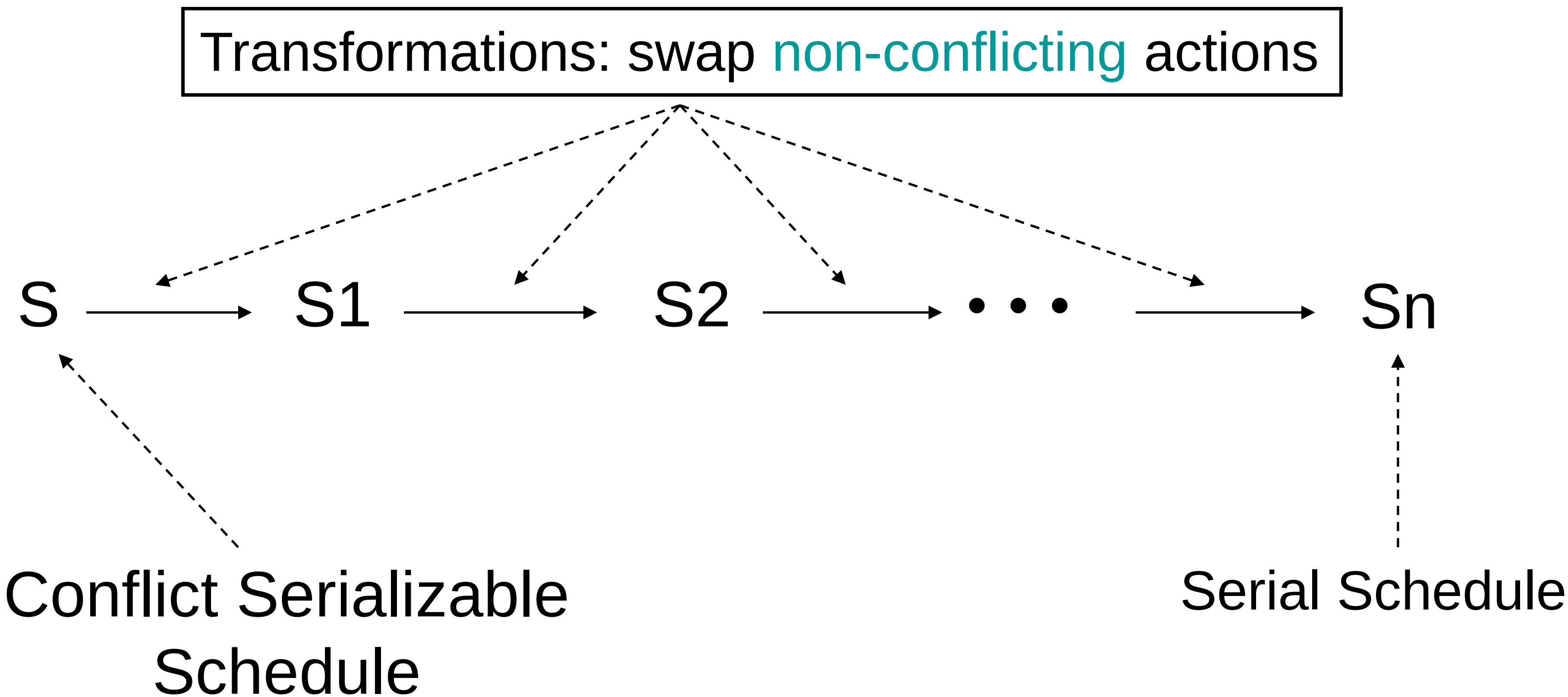
Conflict Serializability

- Weaker notion of serializability
- Depends only on reads and writes

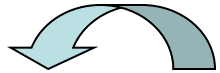
Conflict Serializability



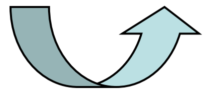
Conflict Serializable Schedule

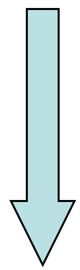


Transformation: Example



r1(A) w1(A) r2(A) w2(A) r1(B) w1(B) r2(B) w2(B)





r1(A) w1(A) r2(A) r1(B) w2(A) w1(B) r2(B) w2(B)

Non-Conflicting Actions

Two actions are non-conflicting if whenever they occur consecutively in a schedule, swapping them does not affect the final state produced by the schedule. Otherwise, they are conflicting.

Conflicting or Non-Conflicting?

(Work on paper: Example 1)

Conflicting Actions: General Rules

- Two actions of the same transaction conflict:
  - r1(A) w1(B)
  - r1(A) r1(B)
- Two actions over the same database element conflict, if one of them is a write
  - r1(A) w2(A)
  - w1(A) w2(A)

Conflict Serializability Examples

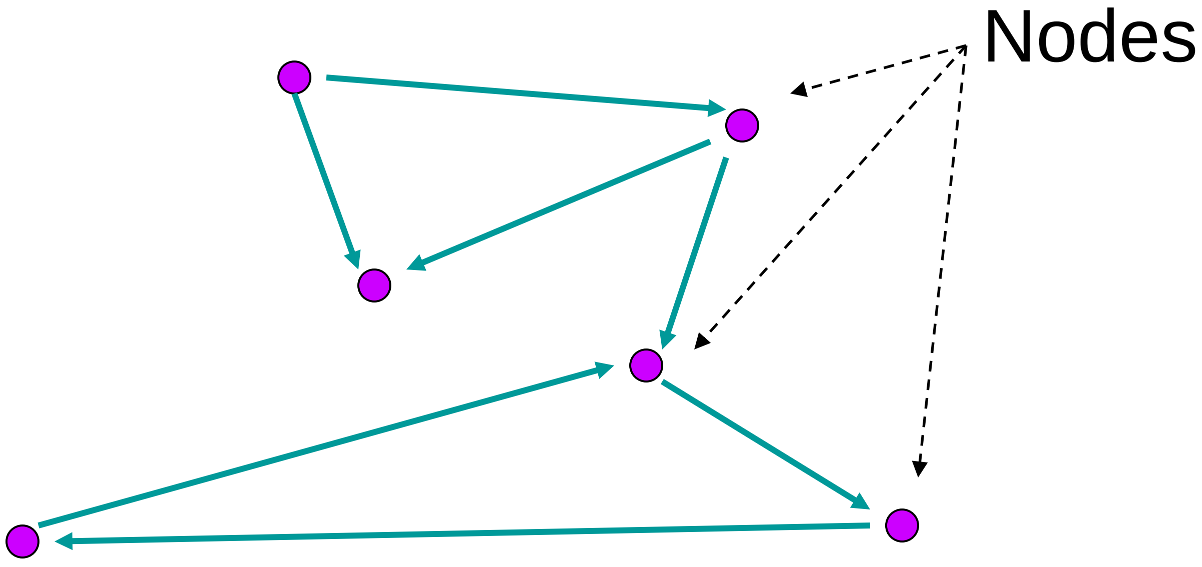
(Work on paper: Example 2 and 3)

Testing Conflict Serializability

- Construct precedence graph G for given schedule S
- S is conflict-serializable iff G is acyclic

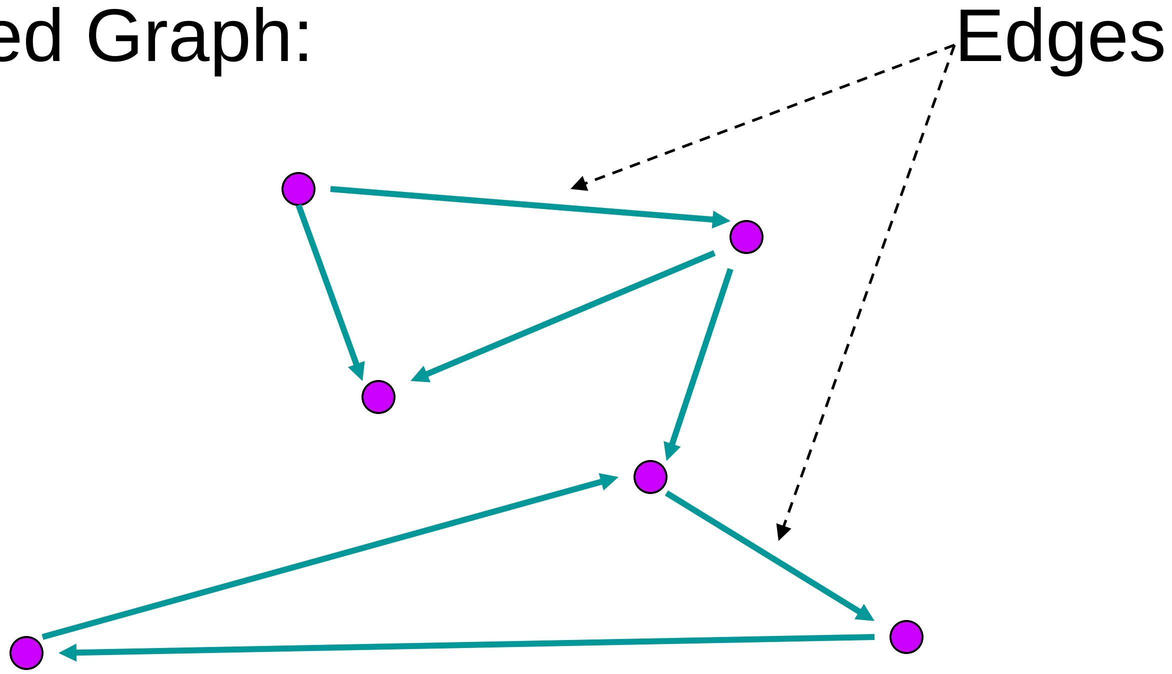
Graph Theory 101

Directed Graph:



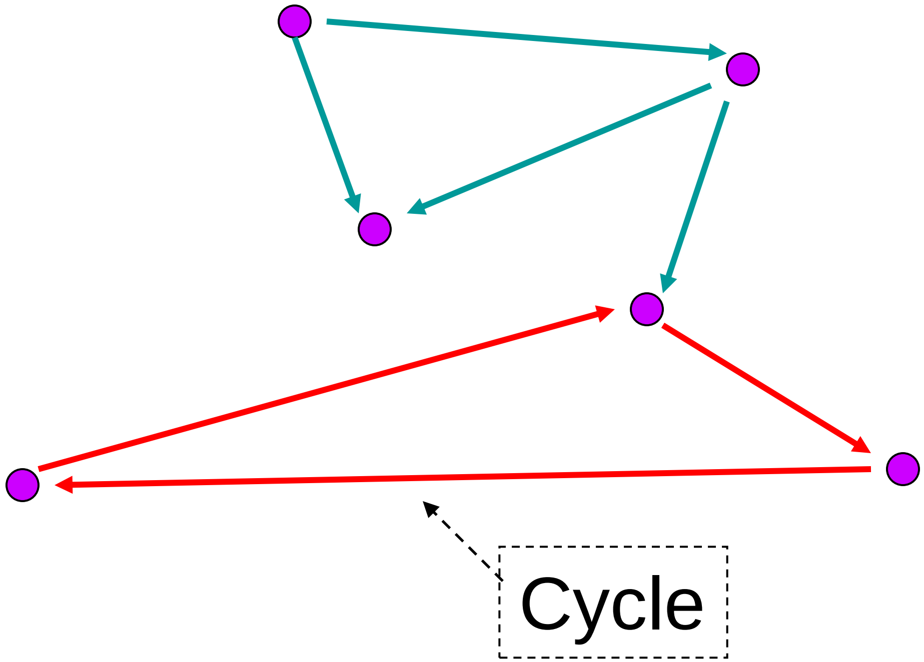
Graph Theory 101

Directed Graph:



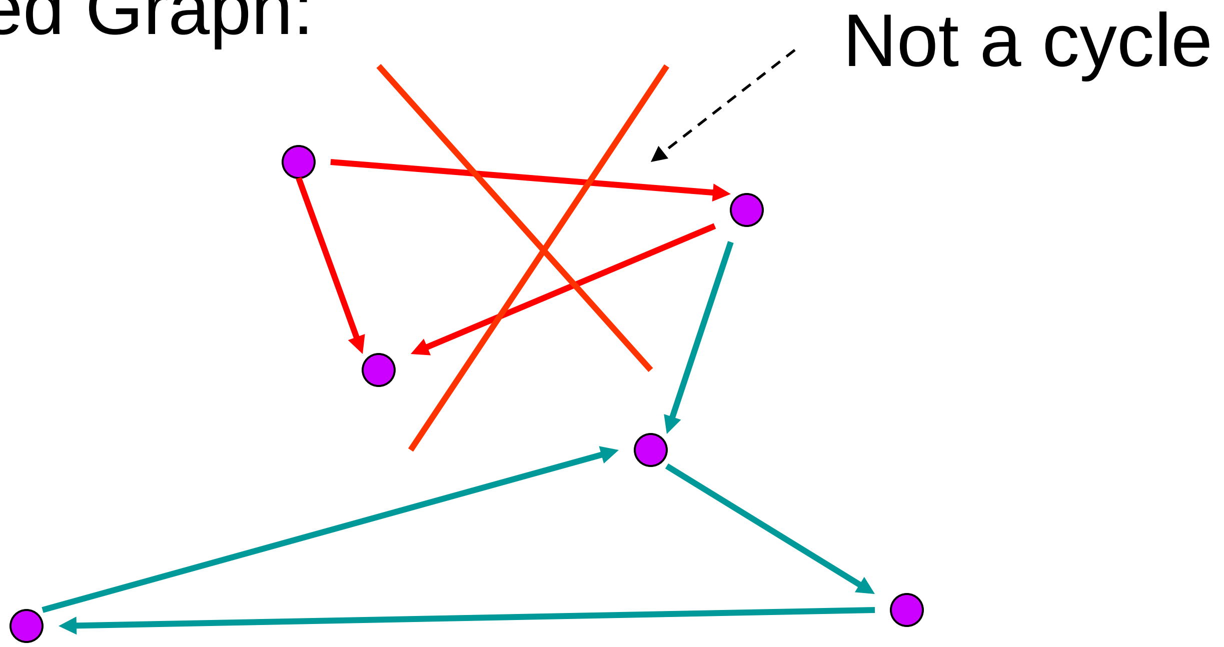
Graph Theory 101

Directed Graph:



Graph Theory 101

Directed Graph:

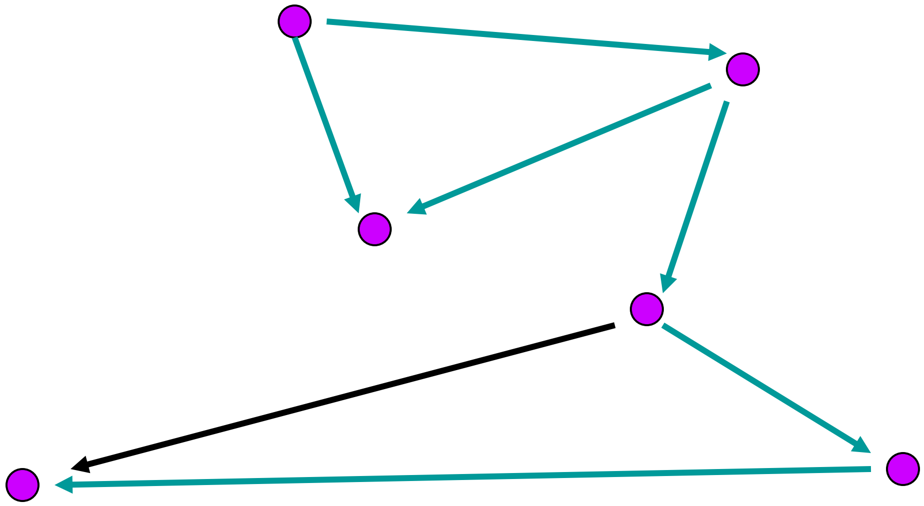


Graph Theory 101

Acyclic Graph: A graph with no cycles

Graph Theory 101

Acyclic Graph:



Testing Conflict Serializability

- Construct precedence graph G for given schedule S
- S is conflict-serializable iff G is acyclic

Precedence Graph

- Precedence graph for schedule S:
  - Nodes: Transactions in S
  - Edges: $T_i \rightarrow T_j$ whenever
    - S: … $r_i(X)$ … $w_j(X)$ …
    - S: … $w_i(X)$ … $r_j(X)$ …
    - S: … $w_i(X)$ … $w_j(X)$ …

Note: not necessarily consecutive

Precedence Graph

- Ti → Tj whenever:
  - There is an action of Ti that occurs before a conflicting action of Tj.

Precedence Graph Example

(Work on paper: Example 4)

Testing Conflict Serializability

- Construct precedence graph G for given schedule S
- S is conflict-serializable iff G is acyclic

Correctness of precedence graph method

(Work on paper)

Serializability vs. Conflict Serializability

(Work on paper: Example 5)

View Serializability

- A schedule S is view serializable if there exists a serial schedule S’, such that the source of all reads in S and S’ are the same.

View Serializability Example

View Serializable Schedule

r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)

Serial Schedule

r2(B) w2(A) w2(B) r1(A) w1(B) r3(A) w3(B)

View Serializability Example

View Serializable Schedule

r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)

Serial Schedule

r2(B) w2(A) w2(B) r1(A) w1(B) r3(A) w3(B)

View Serializability Example

View Serializable Schedule

r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)

Serial Schedule

r2(B) w2(A) w2(B) r1(A) w1(B) r3(A) w3(B)

View Serializability Example

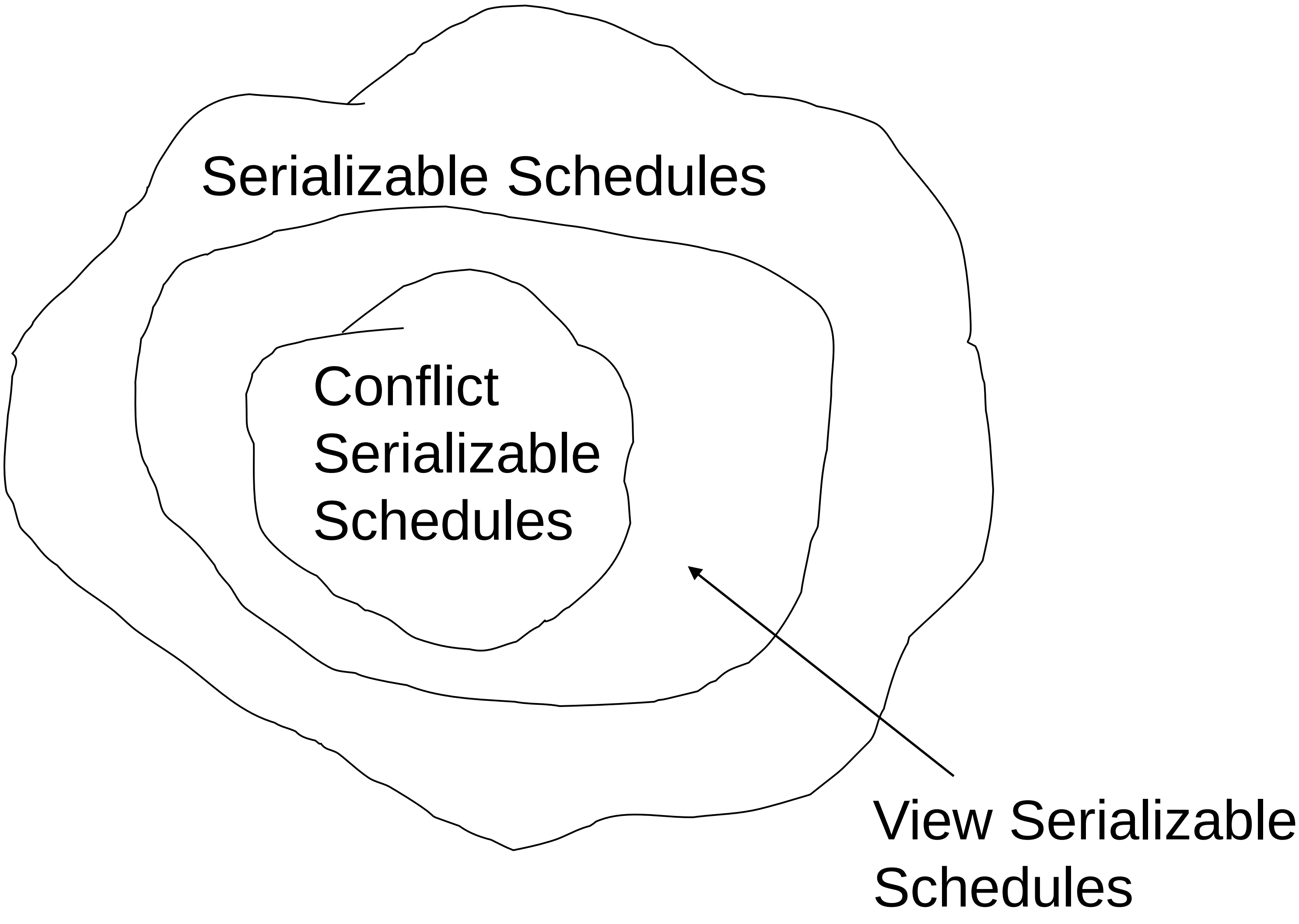
View Serializable Schedule

→ r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)

Serial Schedule

→ r2(B) w2(A) w2(B) r1(A) w1(B) r3(A) w3(B)

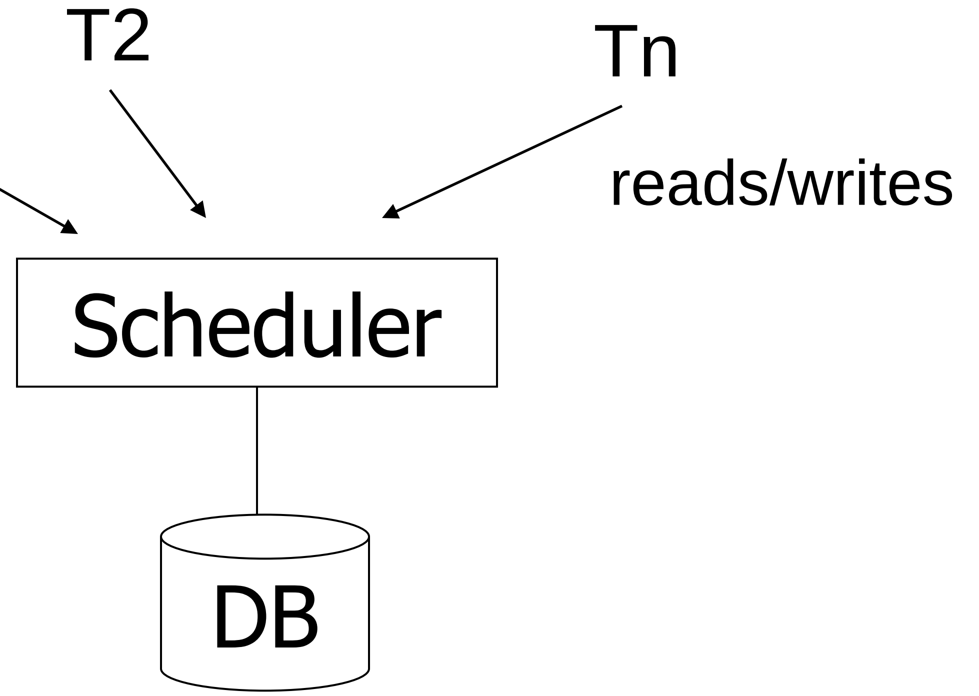
View Serializability



Problems

- Which schedules are "correct"?
  - Serializability theory
- How to build a system that allows only "correct" schedules?
  - Efficient procedure to enforce ~~correctness~~ serializable schedules

Enforcing Serializability



Strategy:
Prevent precedence graph cycles?

Next

- Enforcing serializability
  - Locking-based techniques
  - Timestamp-based techniques
  - Validation-based techniques