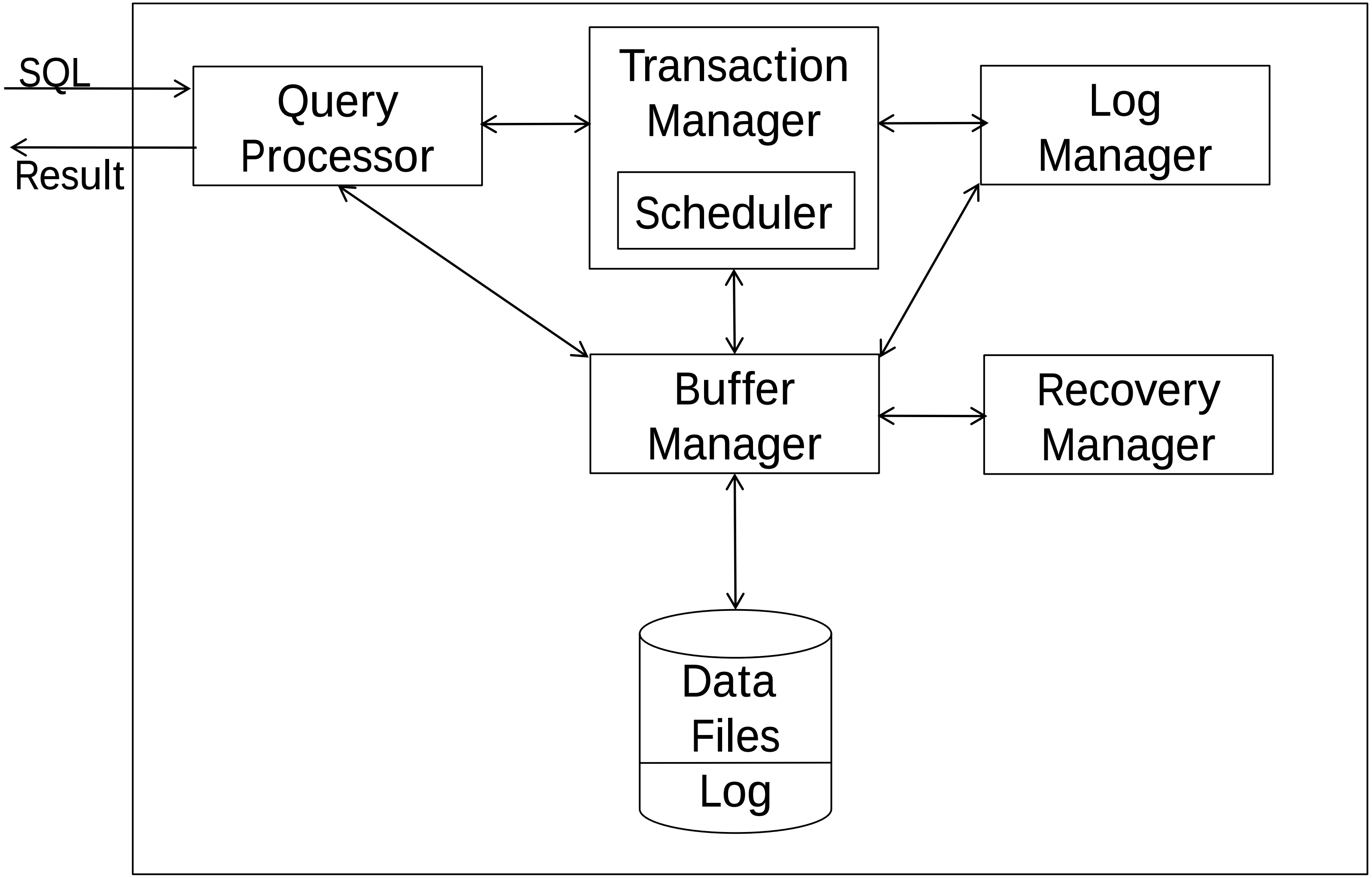
CSCD43: Database Systems Technology

Lecture 12

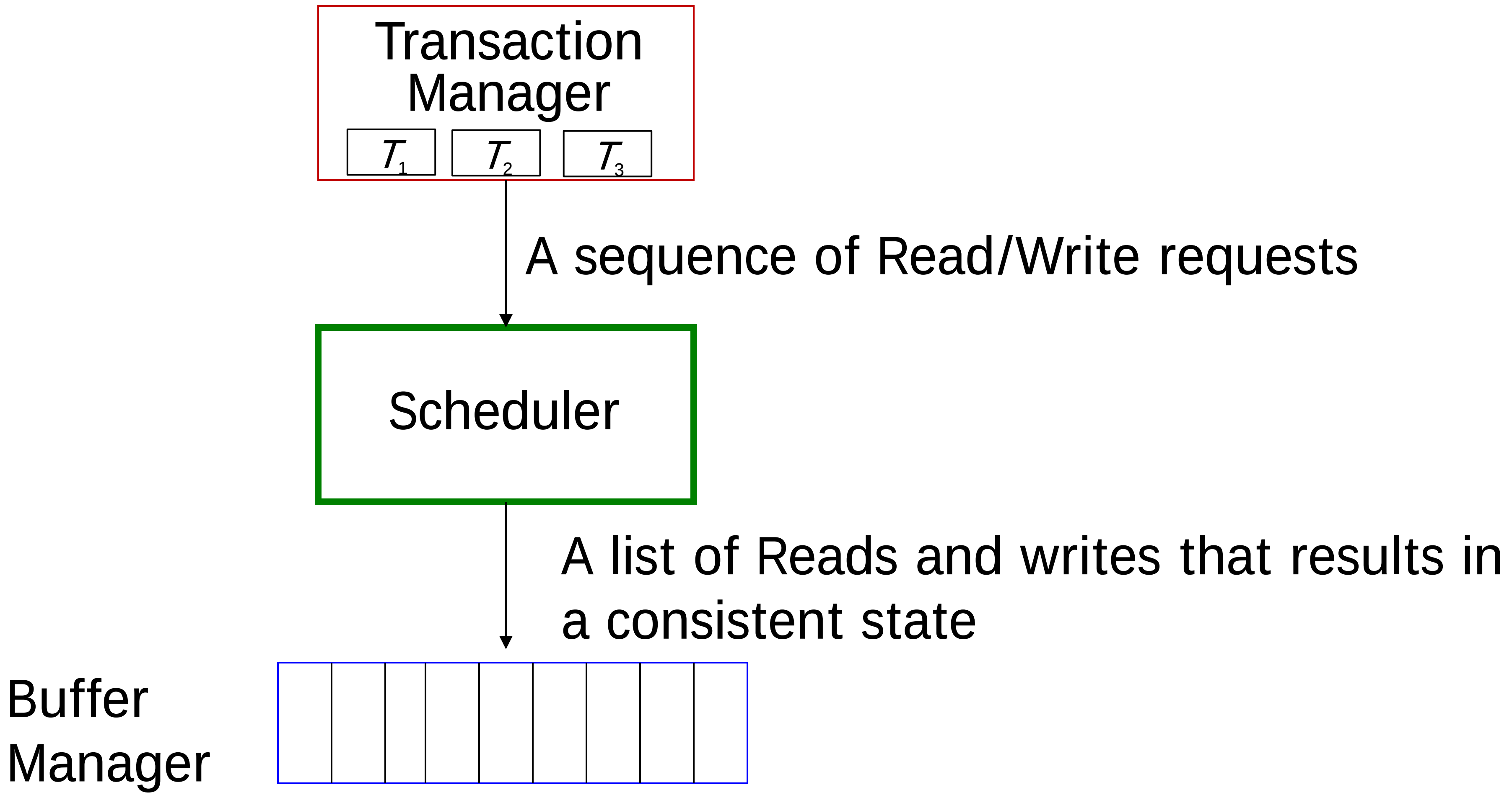
Wael Aboulsaadat

Acknowledgment: these slides are based on Prof. Garcia-Molina & Prof. Ullman slides accompanying the course's textbook.

DBMS Architecture



Component for Concurrency Control



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

Issues with Transactions: 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 Transactions: 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 Transactions: Example

Transaction T2: Transfer 100 from A to C

A = 500, B = 500, C = 500 →

Read (A, s)

s = s - 100

Write (A, s)

Read (C, s)

s = s + 100

Write (C, s)

A = 400, B = 500, C = 600 →

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

300 + 600 + 600 = 1500

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

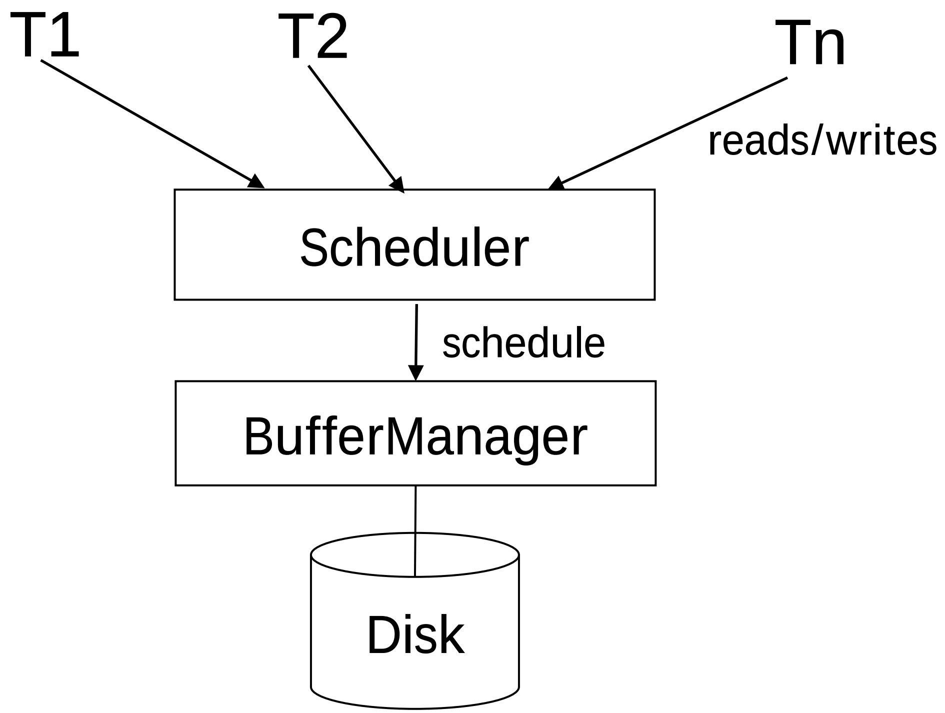
300 + 600 + 600 = 1500

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

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

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

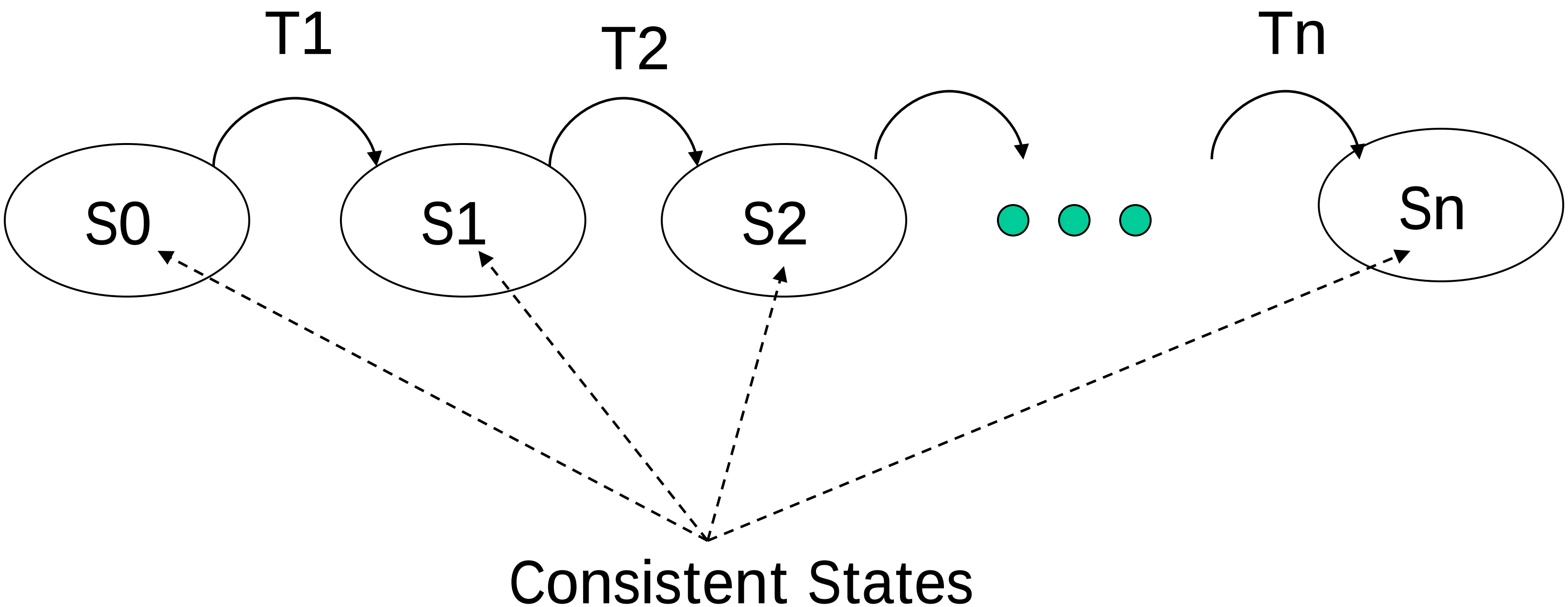
300 + 600 + 600 = 1500

Serial Schedule

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

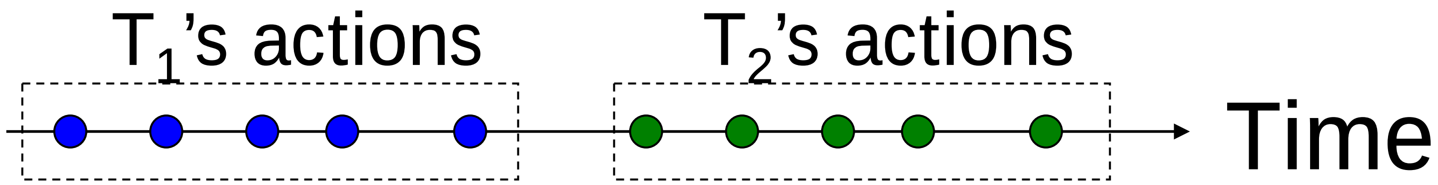
300 + 600 + 600 = 1500

Serial Schedule



Serial Schedule

- If any action of transaction T_1 precedes any action of T_2, then all action of T_1 precede all action of T_2
- The correctness principle tells us that every serial schedule will preserve consistency of the database state



- What's the problem with a Serial Schedule?

Serializability

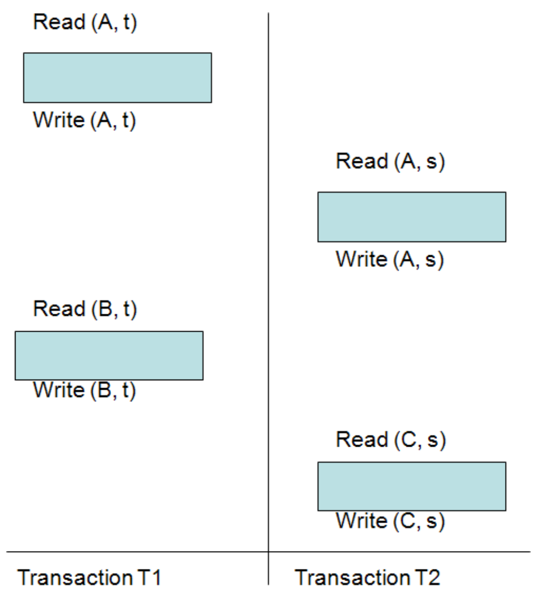
- A schedule is called *serializable* if its final effect is the same as that of a *serial schedule*
- Serializability → schedule is fine and does not result in inconsistent database
  - Since serial schedules are fine
- Non-serializable schedules are unlikely to result in consistent databases
- Scheduler ensures serializability

Serializability

- Not possible to look at all *n!* serial schedules to check if the effect is the same
  - Instead we ensure serializability by allowing or not allowing certain schedules

Conflict Serializability

- Weaker notion of serializability
- Depends only on reads and writes
- Which steps can be interleaved and which cannot



Conflict Serializability

- Recall from OS course:
  - Multitasking
  - context switch

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
read(B)	
B=B+50	
write(B)	
	read(B)
	B = B+ tmp
	write(B)

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
read(B)	
	write(A)
B=B+50	
write(B)	
	read(B)
	B = B+ tmp
	write(B)

Effect:

	Before	After
A	100	45
B	50	105

==

Effect:

	Before	After
A	100	45
B	50	105

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
read(B)	
B=B+50	
write(B)	
	read(B)
	B = B+ tmp
	write(B)

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
read(B)	
B=B+50	
	read(B)
write(B)	
	B = B+ tmp
	write(B)

Effect:	Before	After
A	100	45
B	50	105

! ==

Effect:	Before	After
A	100	45
B	50	55

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
read(B)	
B=B+50	
write(B)	
	read(B)
	B = B+ tmp
	write(B)

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
read(B)	
B=B+50	
	write(A)
write(B)	
	read(B)
	B = B+ tmp
	write(B)

Effect:	Before	After
A	100	45
B	50	105

==

Effect:	Before	After
A	100	45
B	50	105

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
read(B)	
B=B+50	
write(B)	
	read(B)
	B = B+ tmp
	write(B)

Effect:	Before	After
A	100	45
B	50	105

==

T1	T2
read(A)	
A = A -50	
write(A)	
read(B)	
B=B+50	
write(B)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
	read(B)
	B = B+ tmp
	write(B)

Effect:	Before	After
A	100	45
B	50	105

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
  - ...

Non-Conflicting Steps

- Two Reads
  - E.g., $r_i(X); r_i(Y)$
- Read and write of different database element
  - E.g., $r_i(X); w_j(Y)$
- Two writes of different database elements
  - E.g., $w_i(X); w_j(Y)$

Conflicting Steps

- Two actions of the same transaction
  - E.g., $r_i(X); w_i(Y)$

- Two writes of the same database element
  - E.g., $w_i(X); w_j(X)$

- A read and a write of the same database element
  - E.g., $r_i(X); w_j(X)$

Conflict Serializability

- Conflict-equivalent schedules:
  - If S can be transformed into S' through a series of swaps, S and S' are called *conflict-equivalent*
  - *conflict-equivalent guarantees same final effect on the database*

- A schedule S is conflict-serializable if it is conflict-equivalent to a serial schedule

Conflict-Serializability

- Commercial systems generally support *conflict-serializability*
  - Stronger notion than serializability

- Turn a given schedule to a serial one by make as many nonconflicting swaps as we wish

Testing for conflict-serializability

- Given a schedule, determine if it is conflict-serializable
- Construct a *precedence-graph* over the transactions
  - A directed edge from T1 and T2, if they have conflicting instructions, and T1's conflicting instruction comes first
- If there is a cycle in the graph, not conflict-serializable
  - Can be checked in at most $O(n+e)$ time, where n is the number of vertices, and e is the number of edges
- If there is none, conflict-serializable

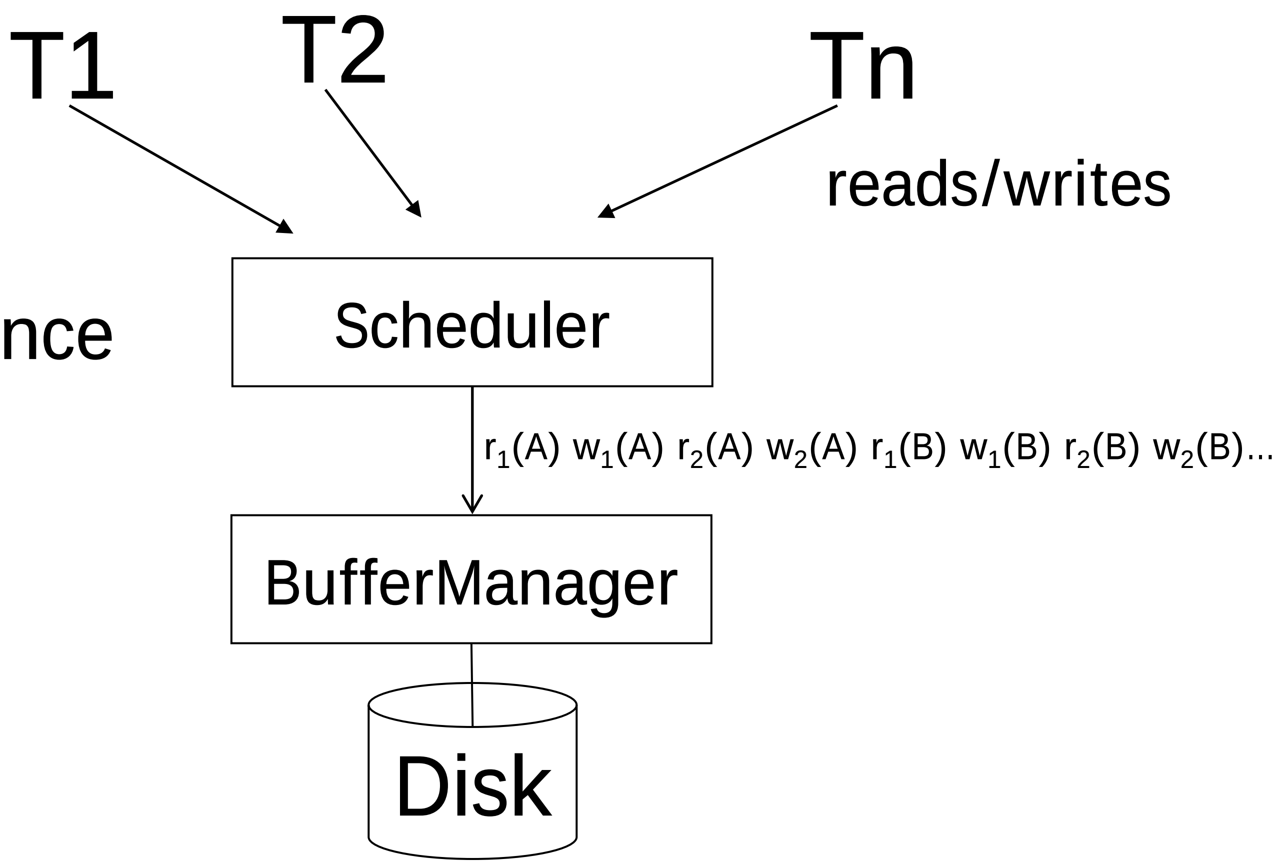
Precedence Graph

- Precedence graph for schedule S:
  - Nodes: Transactions in S
  - Edges: Ti → Tj whenever
    - S: ... r_i (X) ... w_i(X) ...
    - S: ... w_i (X) ... w_j (X) ...
    - S: ... r_i(X) ... w_j (X) ...

Note: not necessarily consecutive

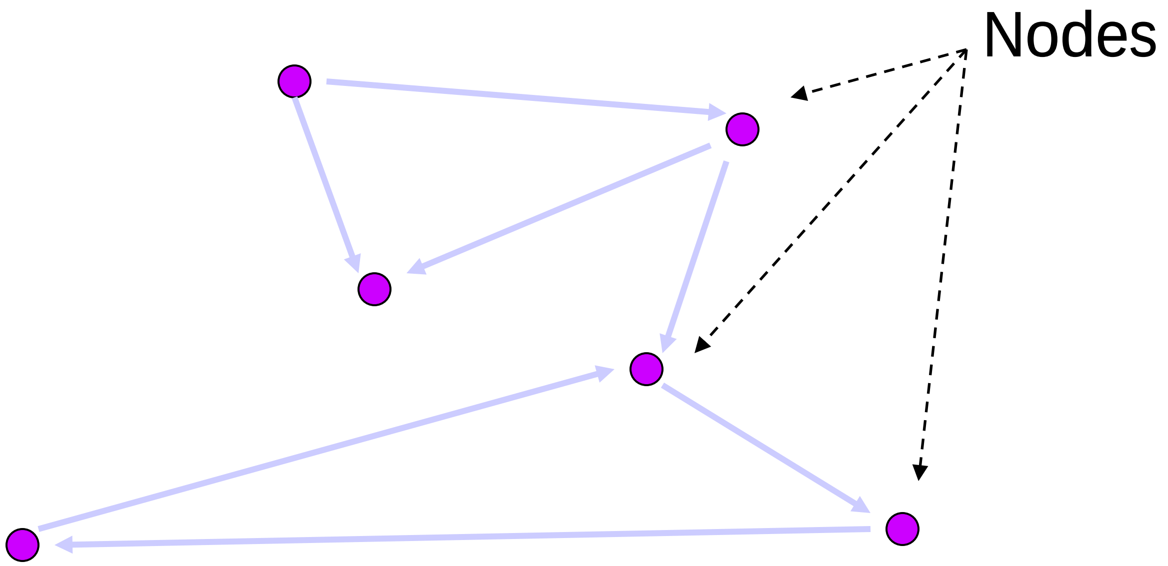
Enforcing Serializability

Strategy:
Prevent precedence graph cycles



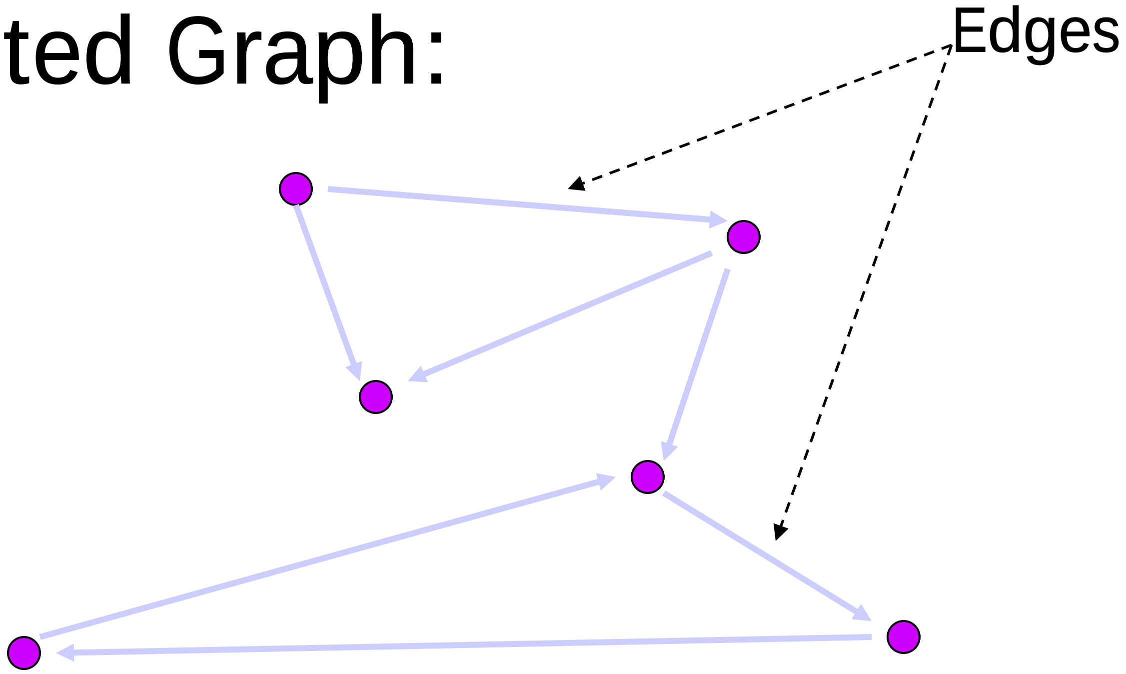
Graph Theory 101

Directed Graph:



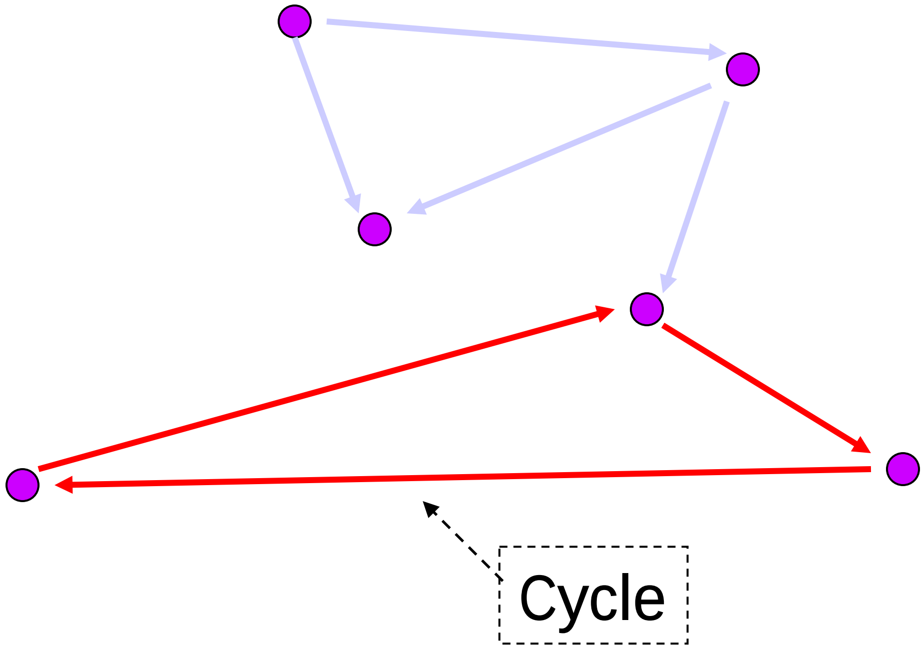
Graph Theory 101

Directed Graph:



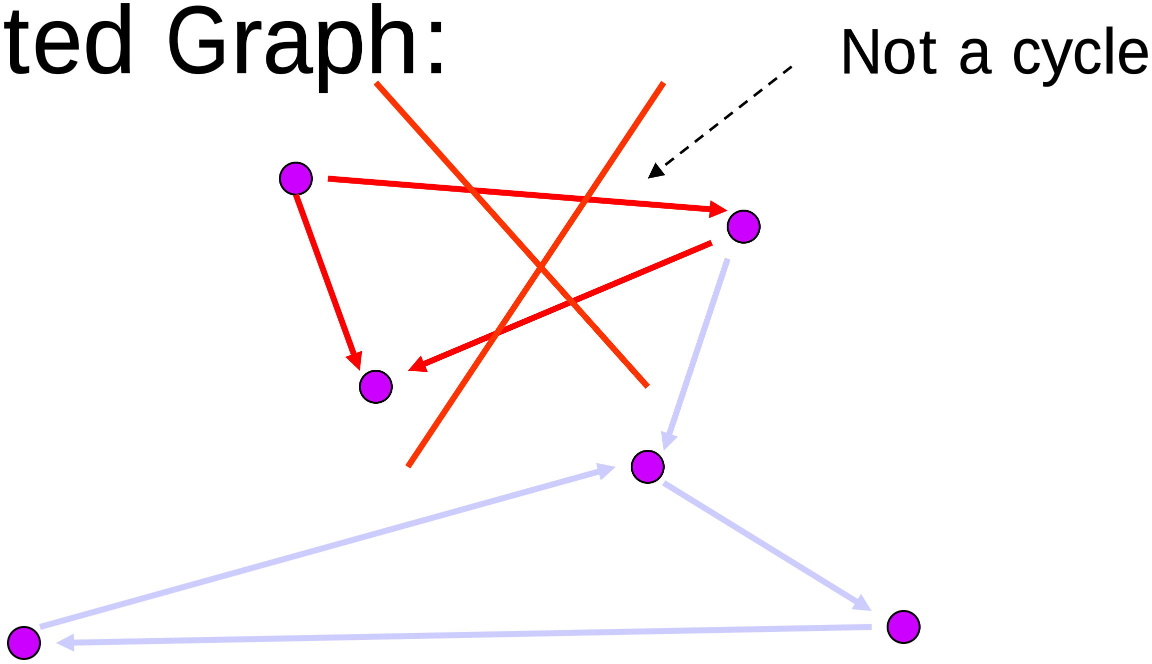
Graph Theory 101

Directed Graph:



Graph Theory 101

Directed Graph:

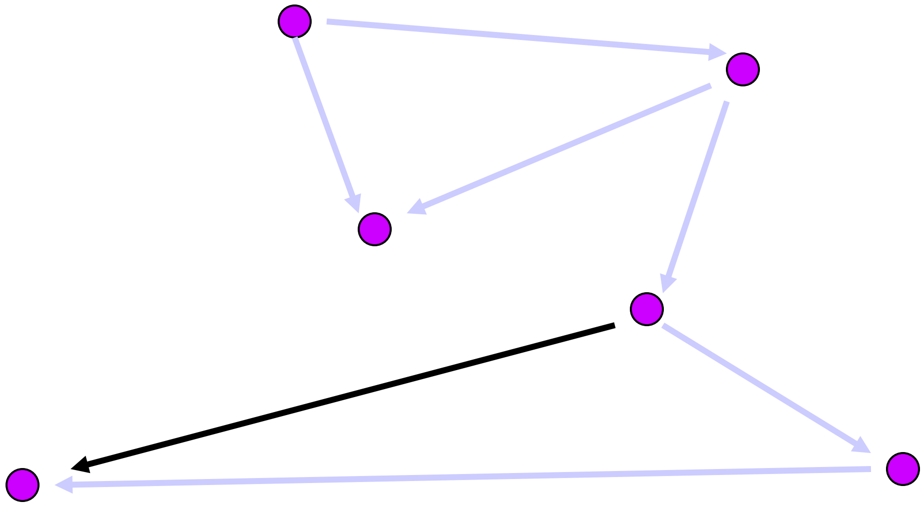


Graph Theory 101

Acyclic Graph: A graph with no cycles

Graph Theory 101

Acyclic Graph:



Precedence Graph - Example

- $T_i \rightarrow T_j$ whenever:
  - There is an action of T_i that occurs before a conflicting action of T_j.

$r_i(X)$; $w_j(Y)$
$w_i(X)$; $w_j(X)$
$r_i(X)$; $w_j(X)$

Precedence Graph – Example 1

- $T_i \rightarrow T_j$ whenever:
  - There is an action of T_i that occurs before a conflicting action of T_j.

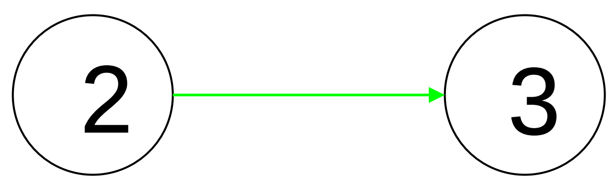
S_1: $r_2(A)$; $r_1(B)$; $w_2(A)$; $r_3(A)$; $w_1(B)$; $w_3(A)$; $r_2(B)$; $w_2(B)$;

$r_i(X)$; $w_i(Y)$
$w_i(X)$; $w_j(X)$
$r_i(X)$; $w_j(X)$

Precedence Graph – Example 1

- $T_i \rightarrow T_j$ whenever:
  - There is an action of T_i that occurs before a conflicting action of T_j.

S_1: $r_2(A)$; $r_1(B)$; $w_2(A)$; $r_3(A)$; $w_1(B)$; $w_3(A)$; $r_2(B)$; $w_2(B)$;

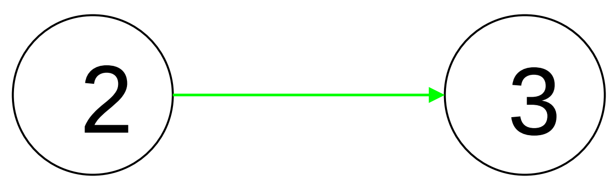


$r_i(X)$; $w_i(Y)$
$w_i(X)$; $w_j(X)$
$r_i(X)$; $w_j(X)$

Precedence Graph – Example 1

- $T_i \rightarrow T_j$ whenever:
  - There is an action of T_i that occurs before a conflicting action of T_j.

S_1: $r_2(A)$; $r_1(B)$; $w_2(A)$; $r_3(A)$; $w_1(B)$; $w_3(A)$; $r_2(B)$; $w_2(B)$;

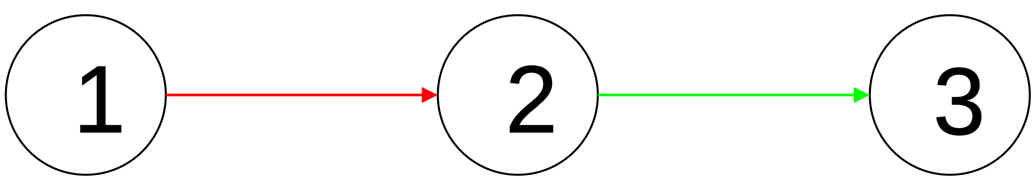


$r_i(X)$; $w_j(Y)$
$w_i(X)$; $w_j(X)$
$r_i(X)$; $w_j(X)$

Precedence Graph – Example 1

- $T_i \rightarrow T_j$ whenever:
  - There is an action of T_i that occurs before a conflicting action of T_j.

S_1: $r_2(A)$; $r_1(B)$; $w_2(A)$; $r_3(A)$; $w_1(B)$; $w_3(A)$; $r_2(B)$; $w_2(B)$;



$r_i(X)$; $w_j(Y)$
$w_i(X)$; $w_j(X)$
$r_i(X)$; $w_j(X)$

Precedence Graph – Example 2

- $T_i \rightarrow T_j$ whenever:
  - There is an action of T_i that occurs before a conflicting action of T_j.

S_2: $r_2(A)$; $r_1(B)$; $w_2(A)$; $r_2(B)$; $r_3(A)$; $w_1(B)$; $w_3(A)$; $w_2(B)$;

$r_i(X)$; $w_j(Y)$
$w_i(X)$; $w_j(X)$
$r_i(X)$; $w_j(X)$

Precedence Graph – Example 2

- $T_i \rightarrow T_j$ whenever:
  - There is an action of T_i that occurs before a conflicting action of T_j.

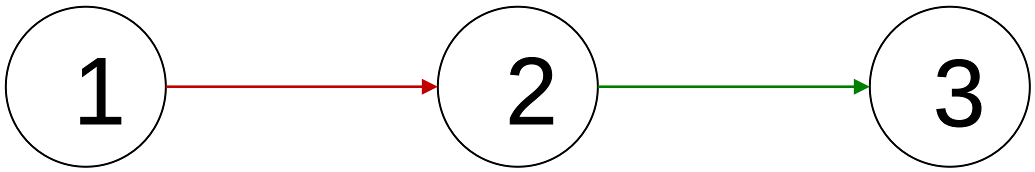
S_2: $r_2(A)$; $r_1(B)$; $w_2(A)$; $r_2(B)$; $r_3(A)$; $w_1(B)$; $w_3(A)$; $w_2(B)$;

$r_i(X)$; $w_j(Y)$
$w_i(X)$; $w_j(X)$
$r_i(X)$; $w_j(X)$

Precedence Graph – Example 2

- $T_i \rightarrow T_j$ whenever:
  - There is an action of T_i that occurs before a conflicting action of T_j.

S_2: $r_2(A)$; $r_1(B)$; $w_2(A)$; $r_2(B)$; $r_3(A)$; $w_1(B)$; $w_3(A)$; $w_2(B)$;

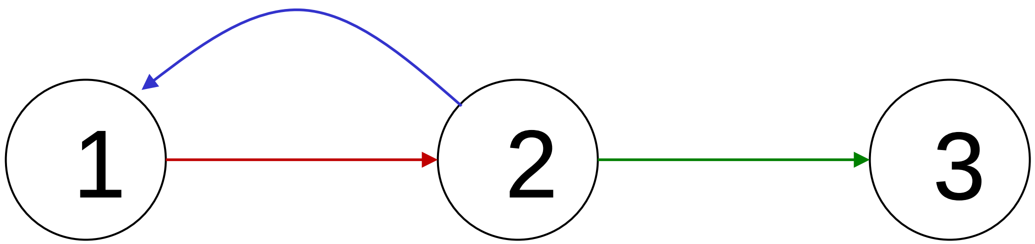


$r_i(X)$; $w_j(Y)$
$w_i(X)$; $w_j(X)$
$r_i(X)$; $w_j(X)$

Precedence Graph – Example 2

- $T_i \rightarrow T_j$ whenever:
  - There is an action of T_i that occurs before a conflicting action of T_j.

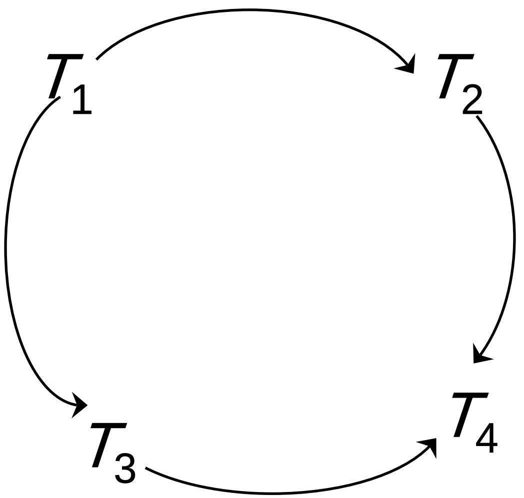
S_2: $r_2(A)$; $r_1(B)$; $w_2(A)$; $r_2(B)$; $r_3(A)$; $w_1(B)$; $w_3(A)$; $w_2(B)$;



$r_i(X)$; $w_j(Y)$
$w_i(X)$; $w_j(X)$
$r_i(X)$; $w_j(X)$

Precedence Graph – Example 3

T_1	T_2	T_3	T_4	T_5
	read(X)			
read(Y)				
read(Z)				
				read(V)
				read(W)
				read(W)
	read(Y)			
	write(Y)			
		write(Z)		
read(U)				
			read(Y)	
			write(Y)	
			read(Z)	
			write(Z)	
read(U)				
write(U)				



$r_i(X)$; $w_j(Y)$
$w_i(X)$; $w_j(X)$
$r_i(X)$; $w_j(X)$

$r_2(X);r_1(Y);r_1(Z);r_5(V);r_5(W);r_5(W);r_2(Y);w_2(Y);w_3(Z);r_1(U);r_4(Y);w_4(Y);r_4(Z);w_4(Z);r_1(U);w_1(U)$

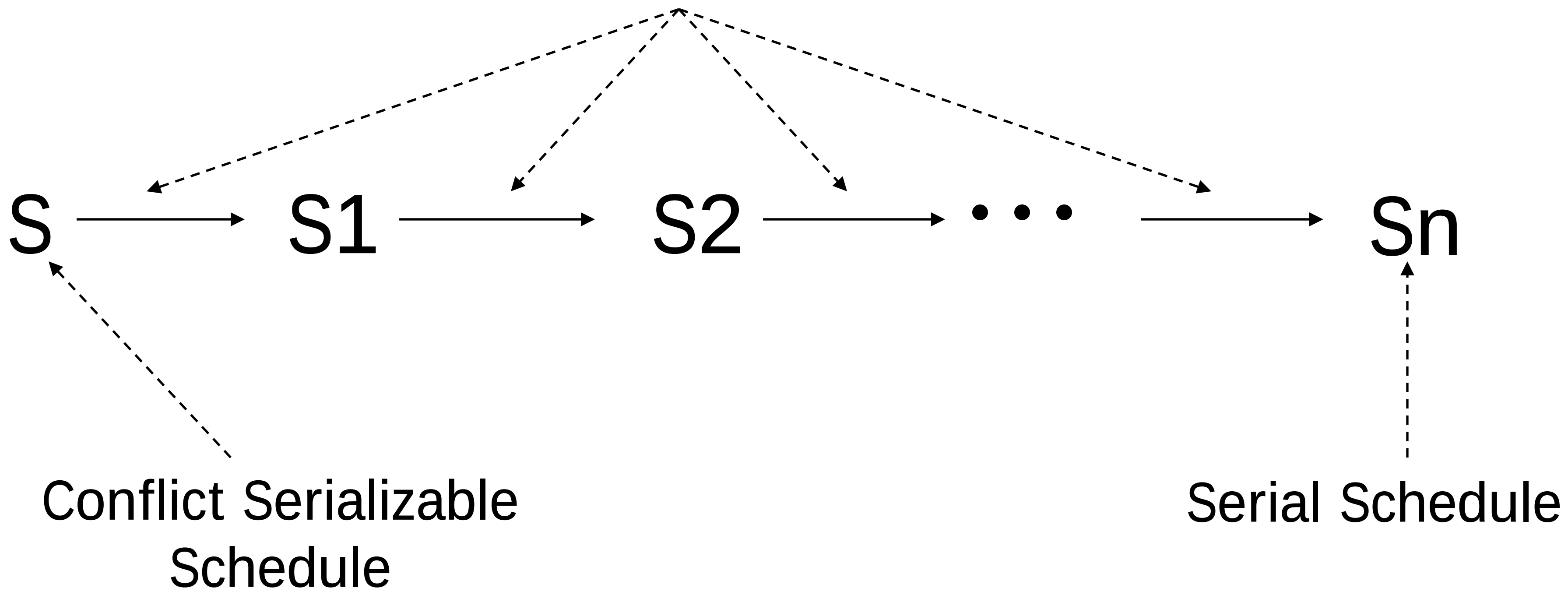
Conflict-serializable

- Two schedules are *conflict-equivalent* if they can be turned one into the other by a sequence of nonconflicting swaps of adjacent actions

- A schedule is *conflict-serializable* if it is conflict-equivalent to a serial schedule

Conflict Serializable Schedule

Transformations: swap non-conflicting actions



Transformation

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)

$r_i(X);\ r_i(Y)$
$r_i(X);\ r_i(Y)$
$w_i(X);\ w_j(Y)$

Transformation: Example

$r_1(A);\ w_1(A);\ r_2(A);\ w_2(A);\ r_1(B);\ w_1(B);\ r_2(B);\ w_2(B)$

Transformation: Example

$r_i(X); r_i(Y)$
$r_i(X); r_j(Y)$
$w_i(X); w_j(Y)$

$r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B)$

Transformation: Example

$r_i(X);\ r_j(Y)$
$r_i(X);\ r_j(Y)$
$w_i(X);\ w_j(Y)$

$r_1(A);\ w_1(A);\ r_2(A);\ w_2(A);\ r_1(B);\ w_1(B);\ r_2(B);\ w_2(B)$

$r_1(A);\ w_1(A);\ r_2(A);\ r_1(B);\ w_2(A);\ w_1(B);\ r_2(B);\ w_2(B)$

$r_i(X);\ r_i(Y)$
$r_i(X);\ r_i(Y)$
$w_i(X);\ w_j(Y)$

Transformation: Example

$r_1(A);\ w_1(A);\ r_2(A);\ w_2(A);\ r_1(B);\ w_1(B);\ r_2(B);\ w_2(B)$

$r_1(A);\ w_1(A);\ r_2(A);\ r_1(B);\ w_2(A);\ w_1(B);\ r_2(B);\ w_2(B)$

$r_i(X)$; $r_i(Y)$
$r_i(X)$; $r_j(Y)$
$w_i(X)$; $w_j(Y)$

Transformation: Example

$r_1(A)$; $w_1(A)$; $r_2(A)$; $w_2(A)$; $r_1(B)$; $w_1(B)$; $r_2(B)$; $w_2(B)$

$r_1(A)$; $w_1(A)$; $r_2(A)$; $r_1(B)$; $w_2(A)$; $w_1(B)$; $r_2(B)$; $w_2(B)$

$r_1(A)$; $w_1(A)$; $r_1(B)$; $r_2(A)$; $w_2(A)$; $w_1(B)$; $r_2(B)$; $w_2(B)$

Transformation: Example

$r_i(X); r_i(Y)$
$r_i(X); r_j(Y)$
$w_i(X); w_j(Y)$

$r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_2(A); r_1(B); w_2(A); w_1(B); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_1(B); r_2(A); w_2(A); w_1(B); r_2(B); w_2(B)$

Transformation: Example

$r_i(X); r_i(Y)$
$r_i(X); r_i(Y)$
$w_i(X); w_j(Y)$

$r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_2(A); r_1(B); w_2(A); w_1(B); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_1(B); r_2(A); w_2(A); w_1(B); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_1(B); r_2(A); w_1(B); w_2(A); r_2(B); w_2(B)$

$r_i(X);\ r_j(Y)$
$r_i(X);\ r_j(Y)$
$w_i(X);\ w_j(Y)$

Transformation: Example

$r_1(A);\ w_1(A);\ r_2(A);\ w_2(A);\ r_1(B);\ w_1(B);\ r_2(B);\ w_2(B)$

$r_1(A);\ w_1(A);\ r_2(A);\ r_1(B);\ w_2(A);\ w_1(B);\ r_2(B);\ w_2(B)$

$r_1(A);\ w_1(A);\ r_1(B);\ r_2(A);\ w_2(A);\ w_1(B);\ r_2(B);\ w_2(B)$

$r_1(A);\ w_1(A);\ r_1(B);\ r_2(A);\ w_1(B);\ w_2(A);\ r_2(B);\ w_2(B)$

Transformation: Example

$r_i(X); r_i(Y)$
$r_i(X); r_j(Y)$
$w_i(X); w_j(Y)$

$r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_2(A); r_1(B); w_2(A); w_1(B); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_1(B); r_2(A); w_2(A); w_1(B); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_1(B); r_2(A); w_1(B); w_2(A); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_1(B); w_1(B); r_2(A); w_2(A); r_2(B); w_2(B)$

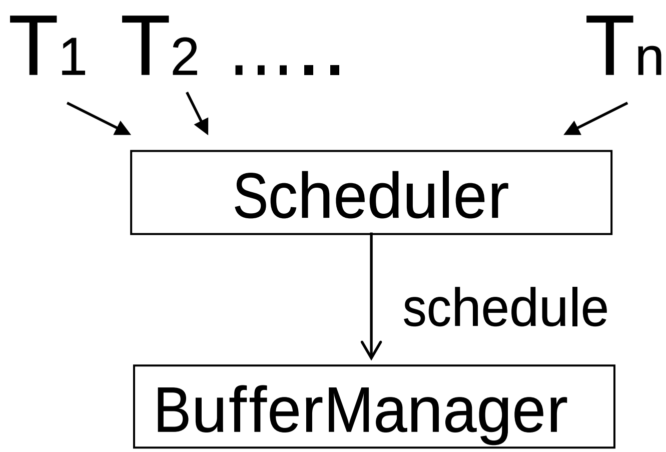
Enforcing Serializability

How to enforce serializable schedules?

Option 1: run system, recording P(S); at end of day, check for P(S) cycles and declare if execution was good! Unrealistic...!

How to enforce serializable schedules?

Option 2: prevent P(S) cycles from occurring



But how ???

2.A) Buffer transactions during n seconds, stop DBMS, make schedule, execute schedule, repeat...

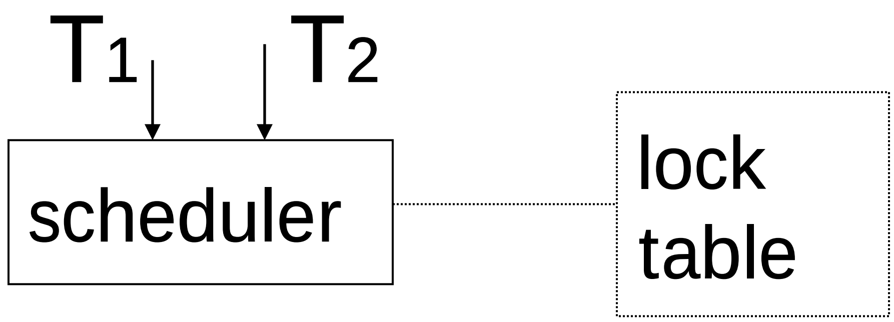
unrealistic...!

2.B) Use a locking protocol!

Two new actions:

lock (exclusive): l_i (A)

unlock: u_i (A)



Rule #1: Well-formed transactions

T_i: ... $l_i(A)$ ... $p_i(A)$ ... $u_i(A)$ ...

Rule #2 Legal scheduler

$S = \ldots\ldots\ l_i(A)\ \ldots\ldots\ldots\ u_i(A)\ \ldots\ldots$

no $l_j(A)$

Exercise:

- What schedules are legal?
  What transactions are well-formed?

  $S1 = l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B)$
  $r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

Exercise:

- What schedules are legal?
  What transactions are well-formed?

  $S1 = l_1(A) l_1(B) r_1(A) w_1(B) l_2(B) u_1(A) u_1(B)$
  $r_2(B) w_2(B) u_2(B) l_3(B) r_3(B) u_3(B)$

Exercise:

- What schedules are legal?
  What transactions are well-formed?

  $S1 = l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B)$
  $r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

  $S2 = l_1(A)r_1(A)w_1(B)u_1(A)u_1(B)$
  $l_2(B)r_2(B)w_2(B)l_3(B)r_3(B)u_3(B)$

Exercise:

- What schedules are legal?
  What transactions are well-formed?

  S1 = $l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B)$
  $r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

  S2 = $l_1(A)r_1(A)w_1(B)u_1(A)u_1(B)$
  $l_2(B)r_2(B)w_2(B)l_3(B)r_3(B)u_3(B)$

Exercise:

- What schedules are legal?
  What transactions are well-formed?

$S1 = l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B)$
$r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

$S2 = l_1(A)r_1(A)w_1(B)u_1(A)u_1(B)$
$l_2(B)r_2(B)w_2(B)l_3(B)r_3(B)u_3(B)$

$S3 = l_1(A)r_1(A)u_1(A)l_1(B)w_1(B)u_1(B)$
$l_2(B)r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

Exercise:

- What schedules are legal?
  What transactions are well-formed?

$S1 = l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B)$
$r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

$S2 = l_1(A)r_1(A)w_1(B)u_1(A)u_1(B)$
$l_2(B)r_2(B)w_2(B)l_3(B)r_3(B)u_3(B)$

$S3 = l_1(A)r_1(A)u_1(A)l_1(B)w_1(B)u_1(B)$
$l_2(B)r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

Locking Example

T1: Read(A); A =A+100; Write(A);
Read(B); B=B+100; Write(B);

T2: Read(A); A =A*2; Write(A);
Read(B); B=B*2; Write(B);

Serial Schedule

T1	T2
Read(A)	
A←A+100;Write(A)	
Read(B);	
B ← B+100; Write(B);	
	Read(A)
	A←Ax2;Write(A);
	Read(B)
	B ← Bx2;Write(B);

A	B
25	25
125	
	125
250	
	250
250	250

Schedule A

T1	T2
$l_1(A)$;Read(A)	
$A \leftarrow A+100$;Write(A);$u_1(A)$	
	$l_2(A)$;Read(A)
	$A \leftarrow Ax2$;Write(A);$u_2(A)$
	$l_2(B)$;Read(B)
	$B \leftarrow Bx2$;Write(B);$u_2(B)$
$l_1(B)$;Read(B)	
$B \leftarrow B+100$;Write(B);$u_1(B)$	

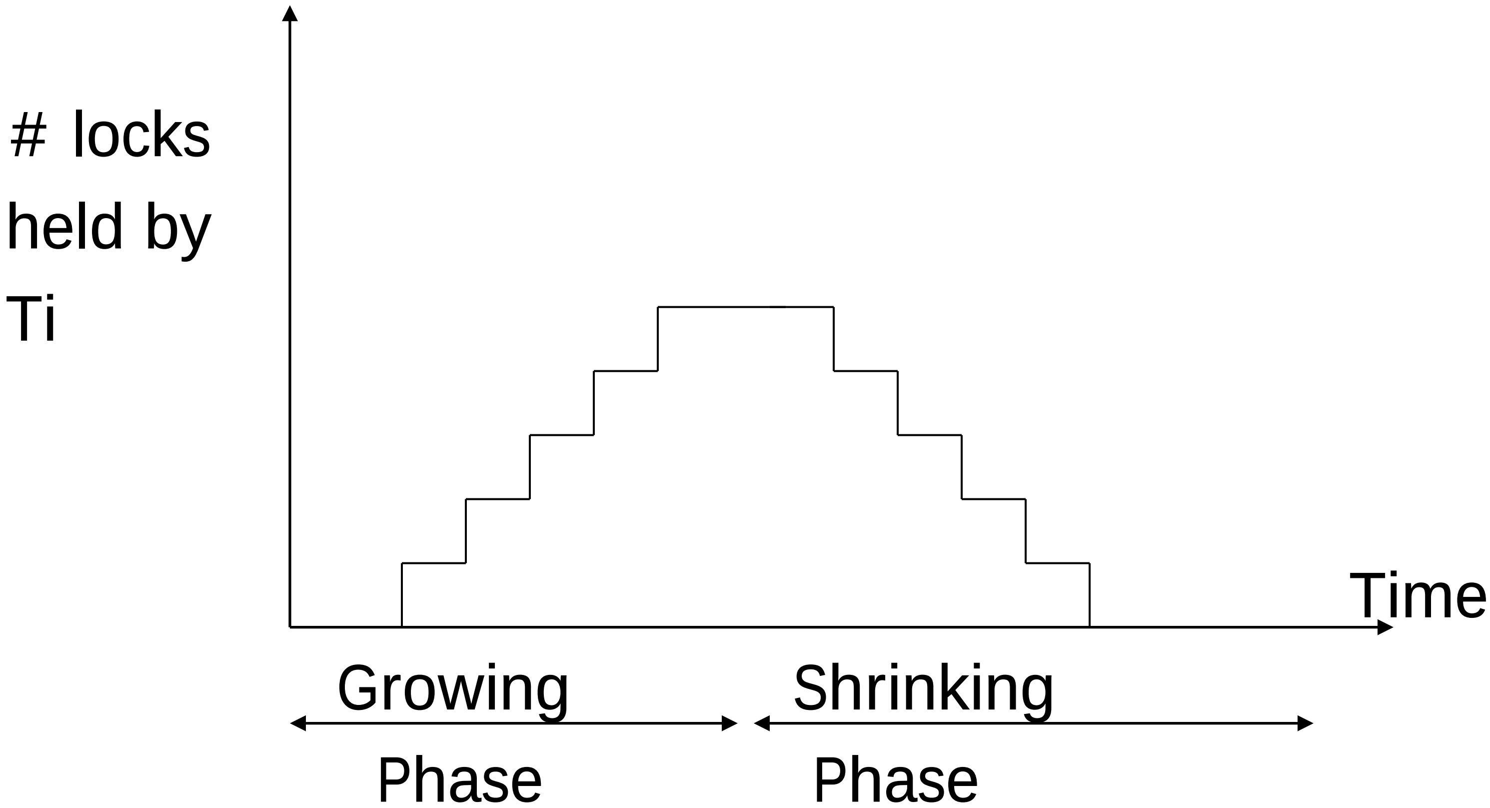
Schedule A

T1	T2	A	B
		25	25
$l_1(A)$;Read(A)			
$A \leftarrow A+100$;Write(A);$u_1(A)$		125	
	$l_2(A)$;Read(A)		
	$A \leftarrow Ax2$;Write(A);$u_2(A)$	250	
	$l_2(B)$;Read(B)		
	$B \leftarrow Bx2$;Write(B);$u_2(B)$		50
$l_1(B)$;Read(B)			
$B \leftarrow B+100$;Write(B);$u_1(B)$			150
		250	150

Rule #3 Two phase locking (2PL)

for transactions

T_i = ... l_i(A) .... u_i(A) ....


locks
held by
Ti
Time
Growing
Phase
Shrinking
Phase

Schedule B

T1	T2
$l_1(A)$;Read(A)	
A←A+100;Write(A)	
$l_1(B)$; $u_1(A)$	
	$l_2(A)$;Read(A)
	A←Ax2;Write(A);$l_2(B)$ (delayed)

Schedule B

T1	T2
$l_1(A)$;Read(A)	
$A \leftarrow A+100$;Write(A)	
$l_1(B)$; $u_1(A)$	
	$l_2(A)$;Read(A)
	$A \leftarrow Ax2$;Write(A);**$l_2(B)$** (delayed)
Read(B);$B \leftarrow B+100$	
Write(B); $u_1(B)$	

Schedule B

T1	T2	A	B
		25	25
l_1(A);Read(A)			
A←A+100;Write(A)			
l_1(B); u_1(A)		125	
	l_2(A);Read(A)		
	A←Ax2;Write(A);**l_2(B)** (delayed)	250	
Read(B);B←B+100			
Write(B); u_1(B)			125
	l_2(B); u_2(A);Read(B)		
	B←Bx2;Write(B);u_2(B);		250
		250	250