LECTURE 8

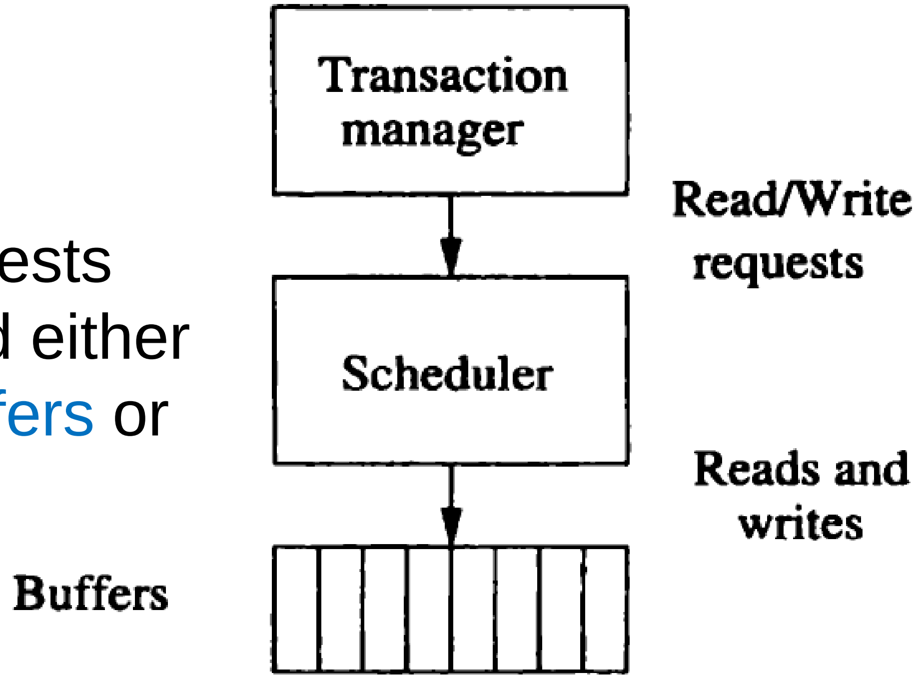
CONCURRENCY CONTROL (PART 1/3)

Concurrency Control

- Motivation
  - Interactions among concurrently executing transactions
    - can cause the database state to become inconsistent
    - even when the transactions individually preserve correctness of the state, and there is no system failure
  - The timing of individual steps of different transactions needs to be regulated

Concurrency Control

- Concurrency Control
  - The process of assuring that transactions preserve consistency when executing simultaneously.
- The scheduler
  - takes read/write requests from transactions and either executes them in buffers or delay them



Concurrency Control

- Concurrency Control
  - How to assure that concurrently executing transactions preserve correctness of the database state
  - Serializability and conflict-serializability
  - Locking
    - Two-phase locking

OUTLINES

18.1 **Serial and Serializable Schedules**

18.2 **Conflict-Serializability**

OUTLINES

18.1 Serial and Serializable Schedules

18.2 Conflict-Serializability

Recall: Correctness Principle

- *The Correctness Principle*: if a transaction executes in the absence of any other transactions or system errors, and it starts with the DB in a consistent state, then the DB is also in a consistent state when the transaction ends.
- In practice
  - Transactions often run concurrently with other transactions, so the correctness principle does not apply directly

Schedules

- Schedule
  - A sequence of the important actions taken by one or more transactions

T_1	T_2
READ(A,t)	READ(A,s)
t := t+100	s := s*2
WRITE(A,t)	WRITE(A,s)
READ(B,t)	READ(B,s)
t := t+100	s := s*2
WRITE(B,t)	WRITE(B,s)

Serial Schedules

- A schedule is serial
  - If its actions consist of all the actions of one transaction, then all the actions of another transaction, and so on.
  - No mixing of the actions is allowed.

Serial Schedules

- A schedule is serial
  - T_1 precedes T_2

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
READ(B,t)			
t := t+100			
WRITE(B,t)			125
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		250

Serial Schedules

- A schedule is serial
  - T_2 precedes T_1
  - In general, we would not expect the final state of a database to be independent of the order of transactions

T_1	T_2	A	B
		25	25
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	50	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		50
READ(A,t)			
t := t+100			
WRITE(A,t)		150	
READ(B,t)			
t := t+100			
WRITE(B,t)			150

Serializable Schedules

- The correctness principle tells us
  - Every serial schedule will preserve consistency of the database state.
  - Are there any other schedules that also are guaranteed to preserve consistency?

- Serializable schedule
  - A schedule *S* is *serializable* if there is a serial schedule *S’* such that for every initial database state, the effects of *S* and *S’* are the same

Serializable Schedules

- Serializable schedule
  - Example:

    Which is the same as for the serial schedule (T_1, T_2)

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
READ(B,t)			
t := t+100			
WRITE(B,t)			125
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		250

Serializable Schedules

- Serializable schedule
  - Example

which is not serializable

- This schedule is the sort of behavior that concurrency control mechanisms must avoid

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		50
READ(B,t)			
t := t+100			
WRITE(B,t)			150

The Effect of Transaction Semantics

- Example
  - Coincidentally, it also results from the serial schedule (T_2, T_1)
- Assume:
  - Any database element *A* that a transaction *T* writes is given a value that depends on the database state in such a way that no arithmetic coincidences occur.

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
	READ(A,s)		
	s := s+200		
	WRITE(A,s)	325	
	READ(B,s)		
	s := s+200		
	WRITE(B,s)		225
READ(B,t)			
t := t+100			
WRITE(B,t)			325

A Notation for Transactions and Schedules

- If we assume "no coincidences"
  - Then only the reads and writes performed by the transactions matter,
  - not the actual values involved
- Notation

$r_T(X)$ transaction T reads database element X

$w_T(X)$ transaction T writes database element X

$r_i(X)$ and $w_i(X)$ synonyms for $r_{Ti}(X)$ and $w_{Ti}(X)$

A Notation for Transactions and Schedules

- Example

T_1	T_2
READ(A,t)	READ(A,s)
t := t+100	s := s*2
WRITE(A,t)	WRITE(A,s)
READ(B,t)	READ(B,s)
t := t+100	s := s*2
WRITE(B,t)	WRITE(B,s)

T_1: $r_1(A)$; $w_1(A)$; $r_1(B)$; $w_1(B)$;
T_2: $r_2(A)$; $w_2(A)$; $r_2(B)$; $w_2(B)$;

A Notation for Transactions and Schedules

- Example

T_1	T_2	A	B
		25	25
`READ(A,t)`			
`t := t+100`			
`WRITE(A,t)`		125	
	`READ(A,s)`		
	`s := s*2`		
	`WRITE(A,s)`	250	
`READ(B,t)`			
`t := t+100`			
`WRITE(B,t)`			125
	`READ(B,s)`		
	`s := s*2`		
	`WRITE(B,s)`		250

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

- An schedule S
  - of a set of transactions $\mathcal{T}$ is a sequence of actions,
  - in which for each transaction T_i in $\mathcal{T}$, the actions of T_i appear in S in the same order that they appear in the definition of T_i itself

OUTLINES

Conflicts

- Conflict
  - A pair of consecutive actions in a schedule such that,
    - If their order is interchanged, then behavior of at least one of the transactions involved can change

Conflicts

- Most pairs of actions do *not* conflict
  1. $r_i(X); r_j(Y)$ is never a conflict, even if $X=Y$
  2. $r_i(X); w_j(Y)$ is not a conflict if $X \neq Y$
  3. $w_i(X); r_j(Y)$ is not a conflict if $X \neq Y$
  4. $w_i(X); w_j(Y)$ is not a conflict if $X \neq Y$

Conflicts

- May *not* swap the order of actions
  1. Two actions of the same transaction:

     $r_i(X); r_i(Y)$, $r_i(X); w_i(Y)$, $w_i(X); r_i(Y)$, $w_i(X); w_i(Y)$
  2. A read and a write of the same database element by different transactions

     $r_i(X); w_j(X)$, $w_i(X); r_j(X)$
  3. Two writes of the same database element by different transactions

     $w_i(X); w_j(X)$

Conflicts

- Any two actions of different transactions may be swapped unless
  1. They involve the same database element, and
  2. At least one is a write

Conflicts

- Conflict-equivalent
  - Two schedules are conflict-equivalent if they can be turned one into the other by a sequence of nonconflicting swaps of adjacent actions
- Conflict-serializable
  - A schedule is conflict-serializable if it is conflict-equivalent to a serial schedule

Conflicts

- Conflict-serializable
  - Conflict-serializability is a sufficient condition for serializability
  - A conflict-serializable schedule is a serializable schedule

Conflicts

- Example

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

$$r_1(A);\ w_1(A);\ r_2(A);\ \underline{w_2(A)};\ \underline{r_1(B)};\ w_1(B);\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ \underline{r_2(A)};\ \underline{r_1(B)};\ w_2(A);\ w_1(B);\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ r_1(B);\ r_2(A);\ \underline{w_2(A)};\ \underline{w_1(B)};\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ r_1(B);\ \underline{r_2(A)};\ \underline{w_1(B)};\ w_2(A);\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ r_1(B);\ w_1(B);\ r_2(A);\ w_2(A);\ r_2(B);\ w_2(B);$$

Precedence Graphs

- To decide whether or not it is conflict-serializable
- T_1 takes precedence of T_2
  - Given a schedule S, involving transactions T_1 and T_2
  - $T_1 <_S T_2$ if there are actions A_1 of T_1 and A_2 of T_2, such that
    - A_1 is ahead of A_2 in S,
    - Both A_1 and A_2 involve the same database elements
    - At least one of A_1 and A_2 is a write action

Precedence Graphs

- Precedence Graphs
  - Nodes
    - The transactions of a schedule *S*
    - Label the node for T_i by only integer *i*
  - Edges
    - There is an arc from node *i* to node *j* if $T_i <_S T_j$

Precedence Graphs

- Example

$$S\colon\ r_2(A);\ r_1(B);\ w_2(A);\ r_3(A);\ w_1(B);\ w_3(A);\ r_2(B);\ w_2(B);$$

The precedence graph for the schedule S

Test for Conflict-Serializability

- To tell whether a schedule *S* is conflict-serializable
  1. Construct the precedence graph for *S*
  2. Ask if there are any cycles
     - If so, then *S* is not conflict-serializable
     - If not, then *S* is conflict-serializable

       – Topological order of the nodes is a conflict-equivalent serial order

Conflicts

- Conflict-serializability is not necessary for serializability
  - Counterexample:

S_1: $w_1(Y)$; $w_1(X)$; $w_2(Y)$; $w_2(X)$; $w_3(X)$;   serial

S_2: $w_1(Y)$; $w_2(Y)$; $w_2(X)$; $w_1(X)$; $w_3(X)$;

S_2 is serializable

S_2 is not conflict -serializable

Test for Conflict-Serializability

- Example

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

The precedence graph for the schedule S

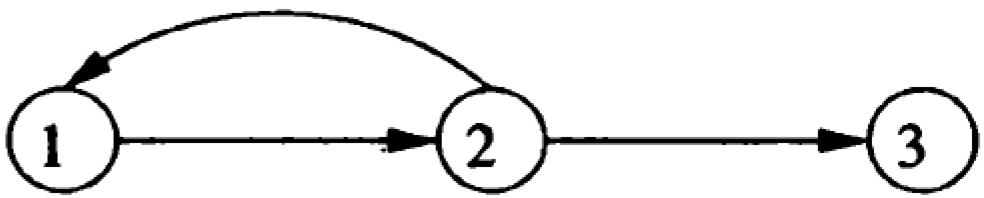
The precedence graph is acyclic, so the schedule *S* is conflict-serializable

How to swap? S': $r_1(B)$; $w_1(B)$; $r_2(A)$; $w_2(A)$; $r_2(B)$; $w_2(B)$; $r_3(A)$; $w_3(A)$;

Test for Conflict-Serializability

- Example

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



The precedence graph is cyclic, so the schedule S_1 is not conflict-serializable

Why the Precedence-Graph Test Works

- Why the Precedence-Graph Test Works? Proof
  1. If there is a cycle involving n transactions $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n \rightarrow T_1$
  2. Then in the hypothetical serial order, the actions of T_1 must precede those of T_2, and so on, up to T_n
  3. But the actions of T_n are also required to precede those of T_1
  - If there is a cycle in the precedence graph, then the schedule is not conflict-serializable

Why the Precedence-Graph Test Works

- Why the Precedence-Graph Test Works?

  Proof: If the precedence graph has no cycles, then the schedule is conflict-serializable

  BASIS:

  - If n=1, i.e., there is only one transaction in the schedule, then the schedule is already serial, and therefore surely conflict-serializable

Why the Precedence-Graph Test Works

- Why the Precedence-Graph Test Works?

  Proof: If the precedence graph has no cycles, then the schedule is conflict-serializable

  INDUCTION:

  - Let the schedule S consist of the actions of n transactions T_1, T_2, …, T_n
  - We suppose that S has an acyclic precedence graph.
    - If a finite graph is acyclic, then there is at least one node that has no arcs in;
    - Let the node i corresponding to transaction T_i be such a node

Why the Precedence-Graph Test Works

- Why the Precedence-Graph Test Works?

  Proof: If the precedence graph has no cycles, then the schedule is conflict-serializable

  INDUCTION:

  - Since there are no arcs into node i, there can be no action A in S that
    1. Involves any transaction Tj other than Ti,
    2. Precedes some action of Ti, and
    3. Conflicts with that action
  - For if there were, we should have put an arc from node j to node i in the precedence graph

Why the Precedence-Graph Test Works

- Why the Precedence-Graph Test Works?

  Proof: If the precedence graph has no cycles, then the schedule is conflict-serializable

  INDUCTION:

  - It is thus possible to swap all the actions of Ti, keeping them in order, but moving them to the front of S
  - The schedule has now taken the form

  (Actions of Ti) (Actions of the other n-1 transactions)

Why the Precedence-Graph Test Works

- Why the Precedence-Graph Test Works?

  Proof: If the precedence graph has no cycles, then the schedule is conflict-serializable

  INDUCTION:

  - Let us now consider the tail of S – the actions of all transactions other than Ti
  - Since these actions maintain the same relative order that they did in S
    - The precedence graph for the tail is the same as the precedence graph for S, except that the node for Ti and any arcs out of that node is missing

Why the Precedence-Graph Test Works

- Why the Precedence-Graph Test Works?

  Proof: If the precedence graph has no cycles, then the schedule is conflict-serializable

  INDUCTION:

  - Since the original precedence graph was acyclic, and deleting nodes and arcs cannot make it cyclic, we conclude that the tail's precedence graph is acyclic
  - Moreover, since the tail involves n-1 transactions, the inductive hypothesis applies to it

Why the Precedence-Graph Test Works

- Why the Precedence-Graph Test Works?

  Proof: If the precedence graph has no cycles, then the schedule is conflict-serializable

  INDUCTION:

  - Thus, we know we can reorder the actions of the tail using legal swaps of adjacent actions to turn it into a serial schedule.
  - Now, S itself has been turned into a serial schedule, with the actions of Ti first and the actions of the other transactions following in some serial order

Why the Precedence-Graph Test Works

- Why the Precedence-Graph Test Works?

  Proof: If the precedence graph has no cycles, then the schedule is conflict-serializable

  INDUCTION:

  - The induction is complete, and we conclude that every schedule with an acyclic precedence graph is conflict-serializable.