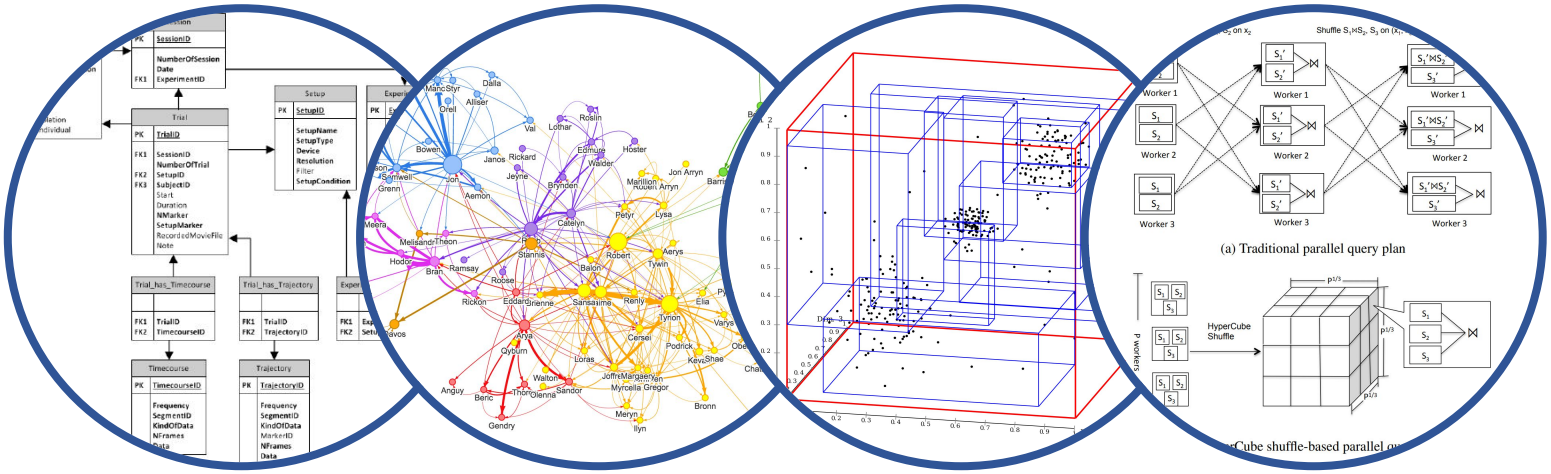


Introduction to Data Management

Transactions: Serializability

Alyssa Pittman
Based on slides by Jonathan Leang, Dan Suciu, et al

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Outline

- Concurrency control problems
- Transactions
- Serializable schedules
- Conflict serializability

Transactions

How do we support multiple people using a database at the same time?

- Multiple application users
- Multiple application programmers
- Multiple analysts

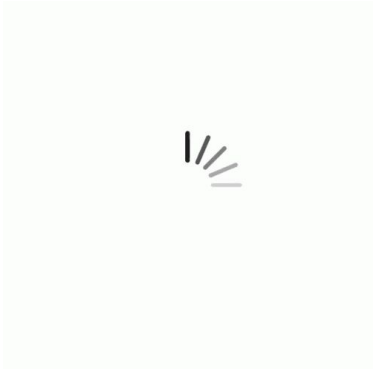
Imagine a world where each person had to wait in line to use your database ☹️

Concurrency Control Problems

- Non-Atomic Operations
- Lost Update
- Dirty/Inconsistent Read
- Unrepeatable Read
- Phantom Read

Non-Atomic Operations

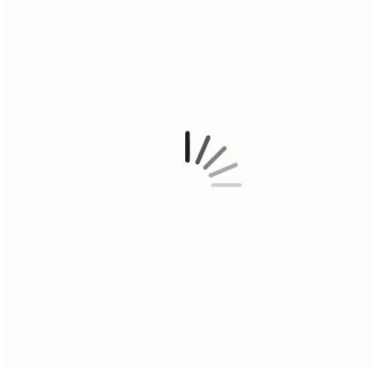
- I'll book this airline flight!
- ...submit credit card number
- loading.gif
- ...bank account goes down...
- no booking confirmation
- waiting.png
- browser crashes
- still no confirmation



Non-Atomic Operations

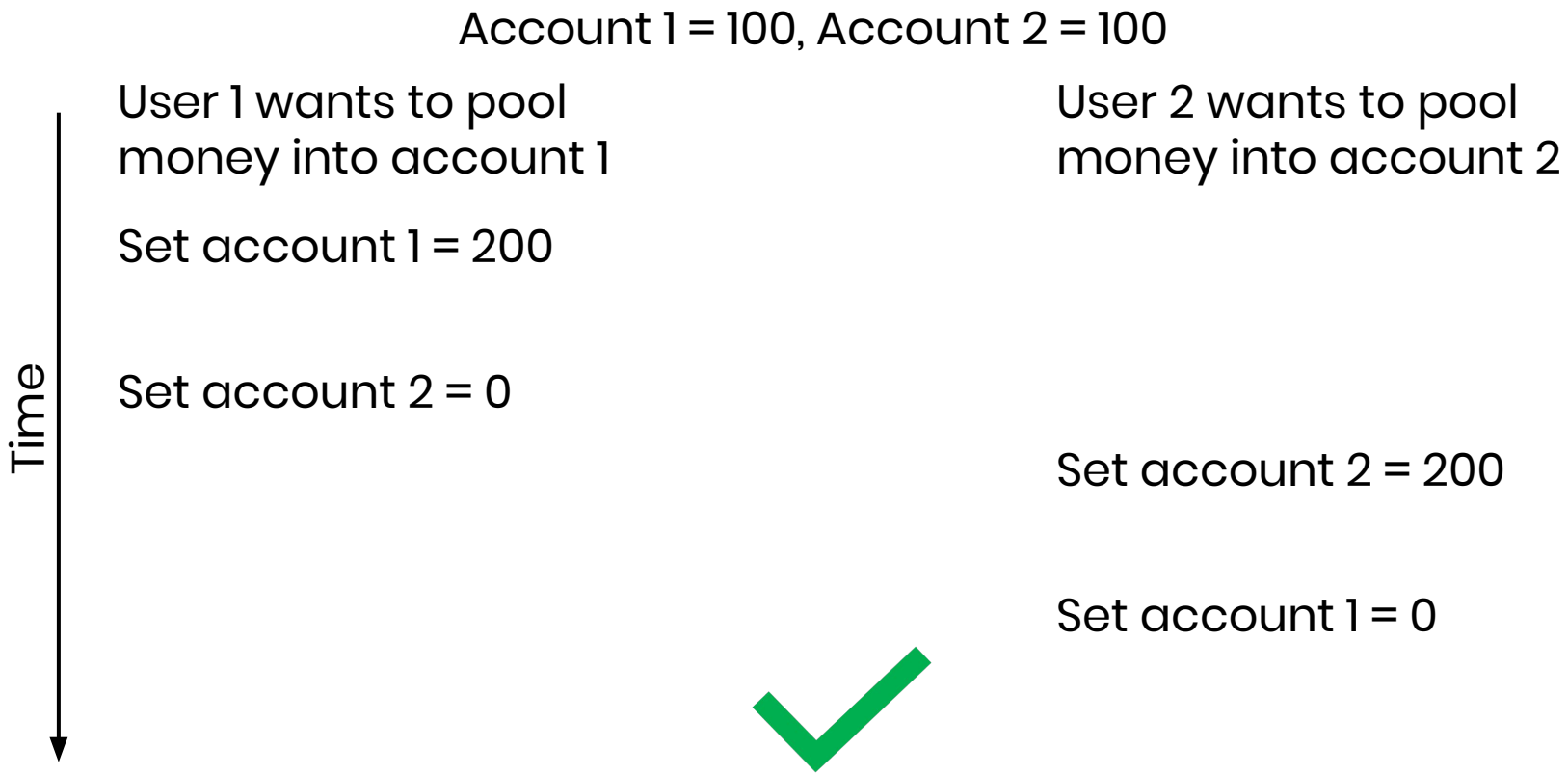
- I'll book this airline flight!
- ...submit credit card number
- loading.gif
- ...bank account goes down...
- no booking confirmation
- waiting.png
- browser crashes
- still no confirmation

I would have been fine if nothing happened or if everything worked!



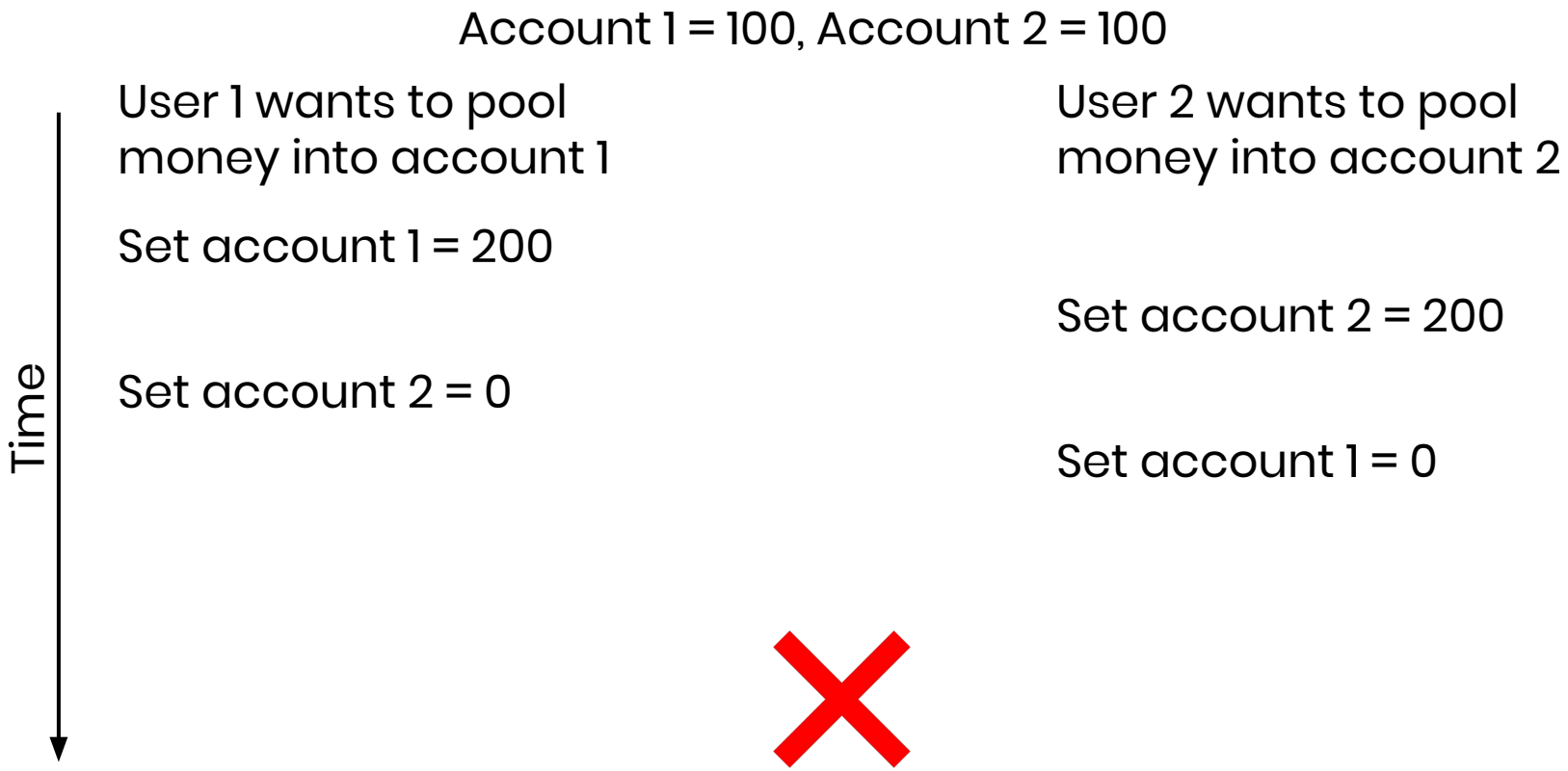
Lost Update

- Write-Write (WW) conflict
- Consolidation scenario:



Lost Update

- Write-Write (WW) conflict
- Consolidation scenario:



Dirty/Inconsistent Read

- Write-Read (WR) conflict
- Budget management scenario:

Time

Manager wants to balance project budgets

-$10mil from project A

+$7mil to project B

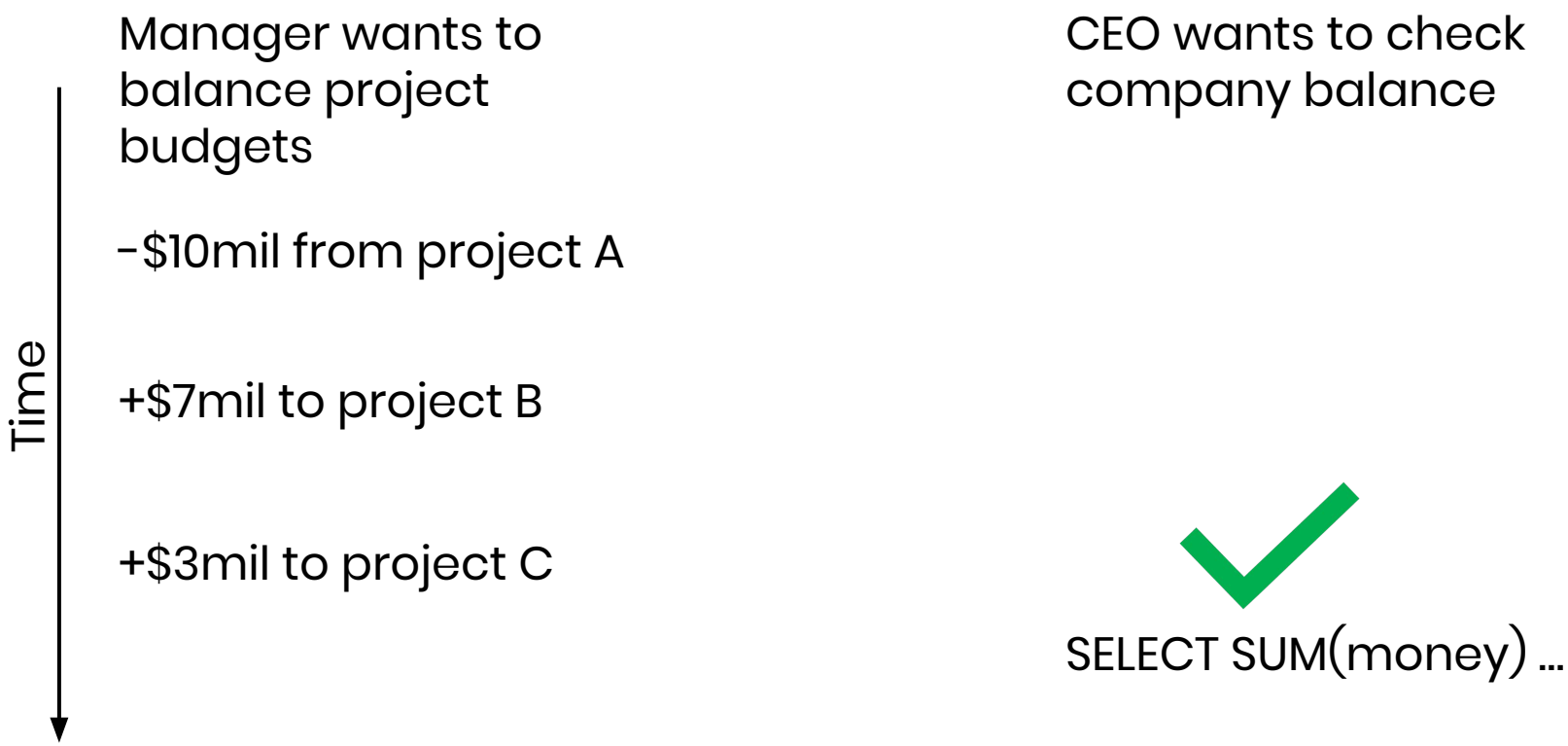
+$3mil to project C

CEO wants to check company balance

SELECT SUM(money) …

Dirty/Inconsistent Read

- Write-Read (WR) conflict
- Budget management scenario:



Dirty/Inconsistent Read

- Write-Read (WR) conflict
- Budget management scenario:

Time

Manager wants to balance project budgets

-$10mil from project A

+$7mil to project B

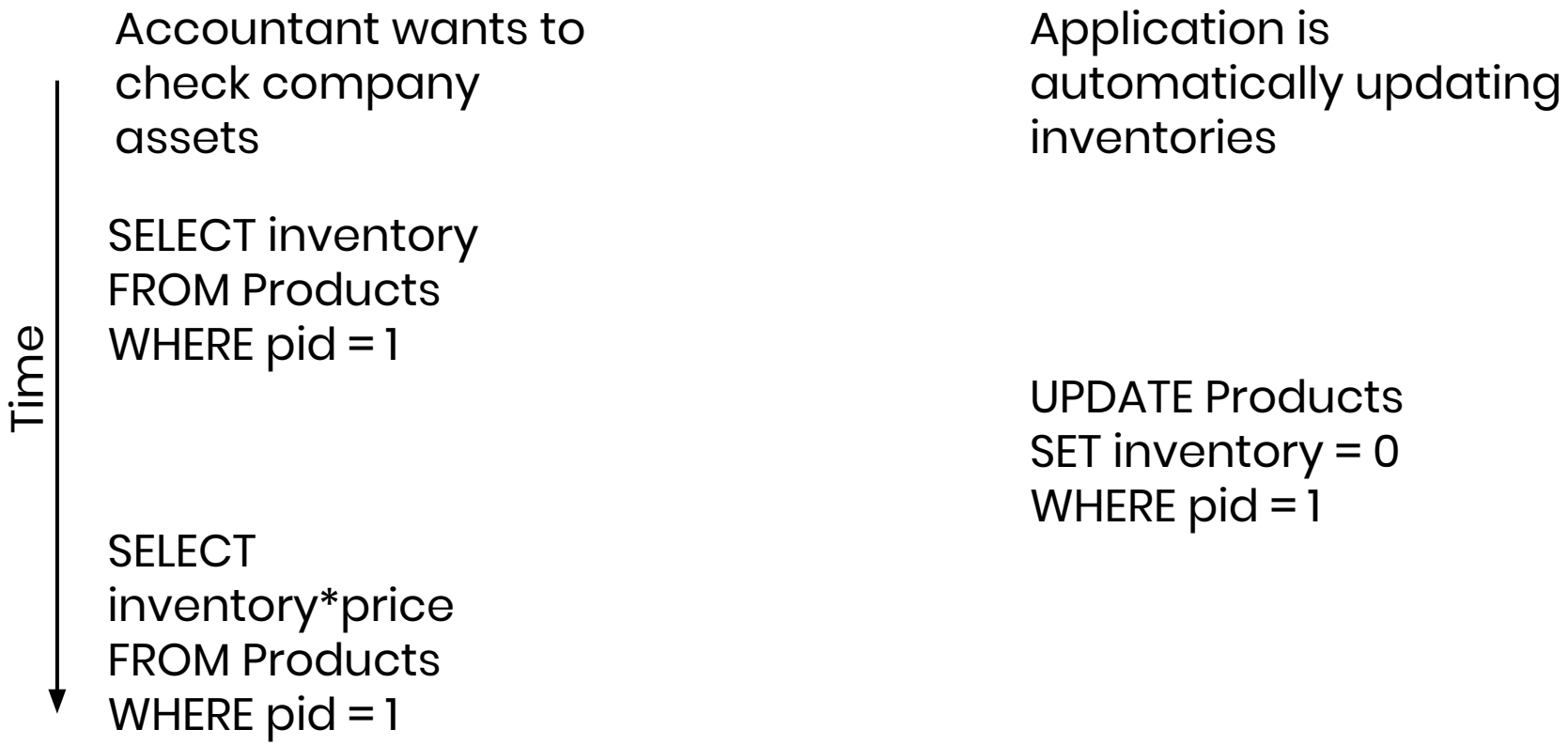
+$3mil to project C

CEO wants to check company balance

SELECT SUM(money) …

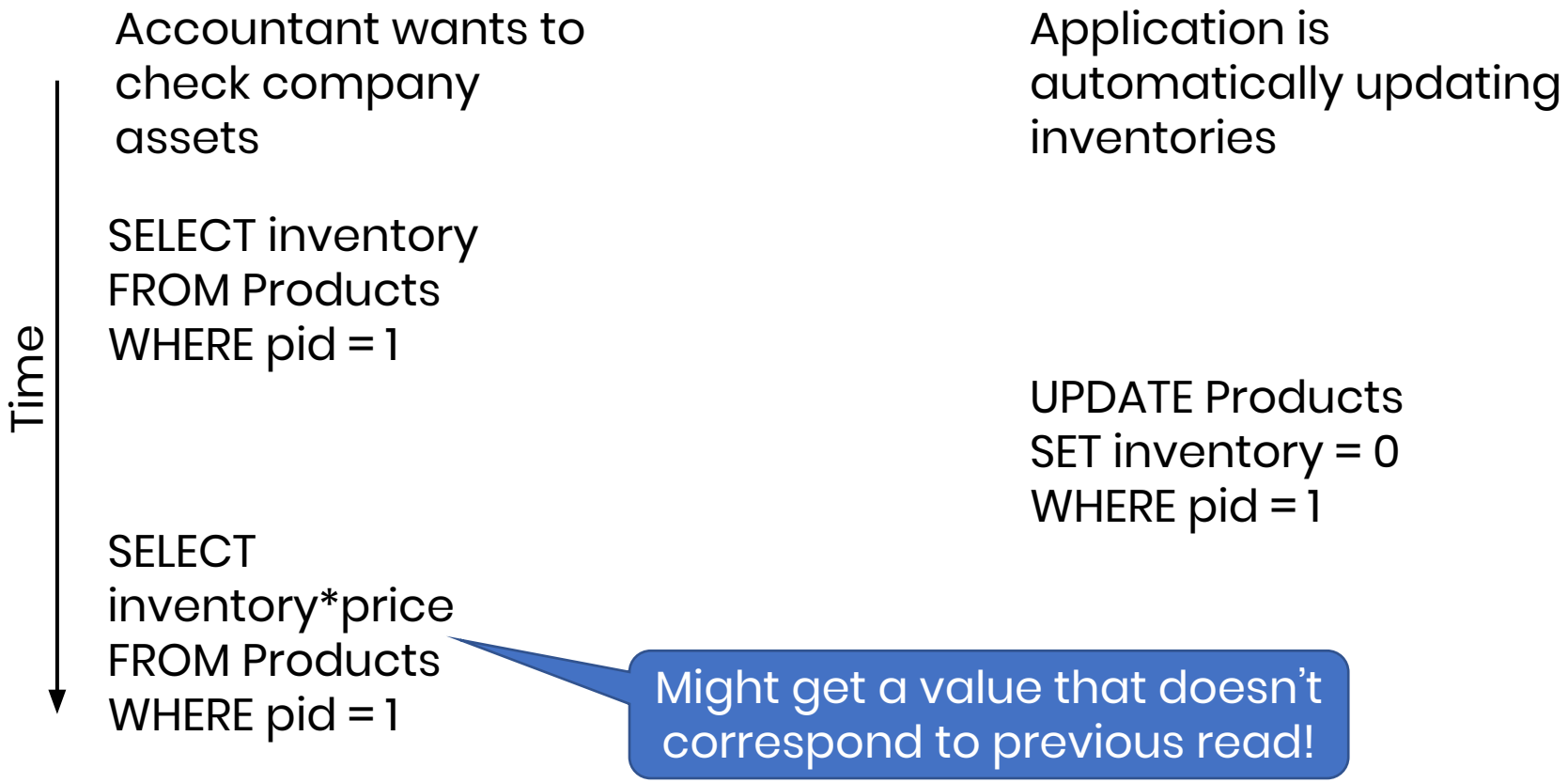
Unrepeatable Read

- Read-Write (RW) conflict
- Asset checking scenario:



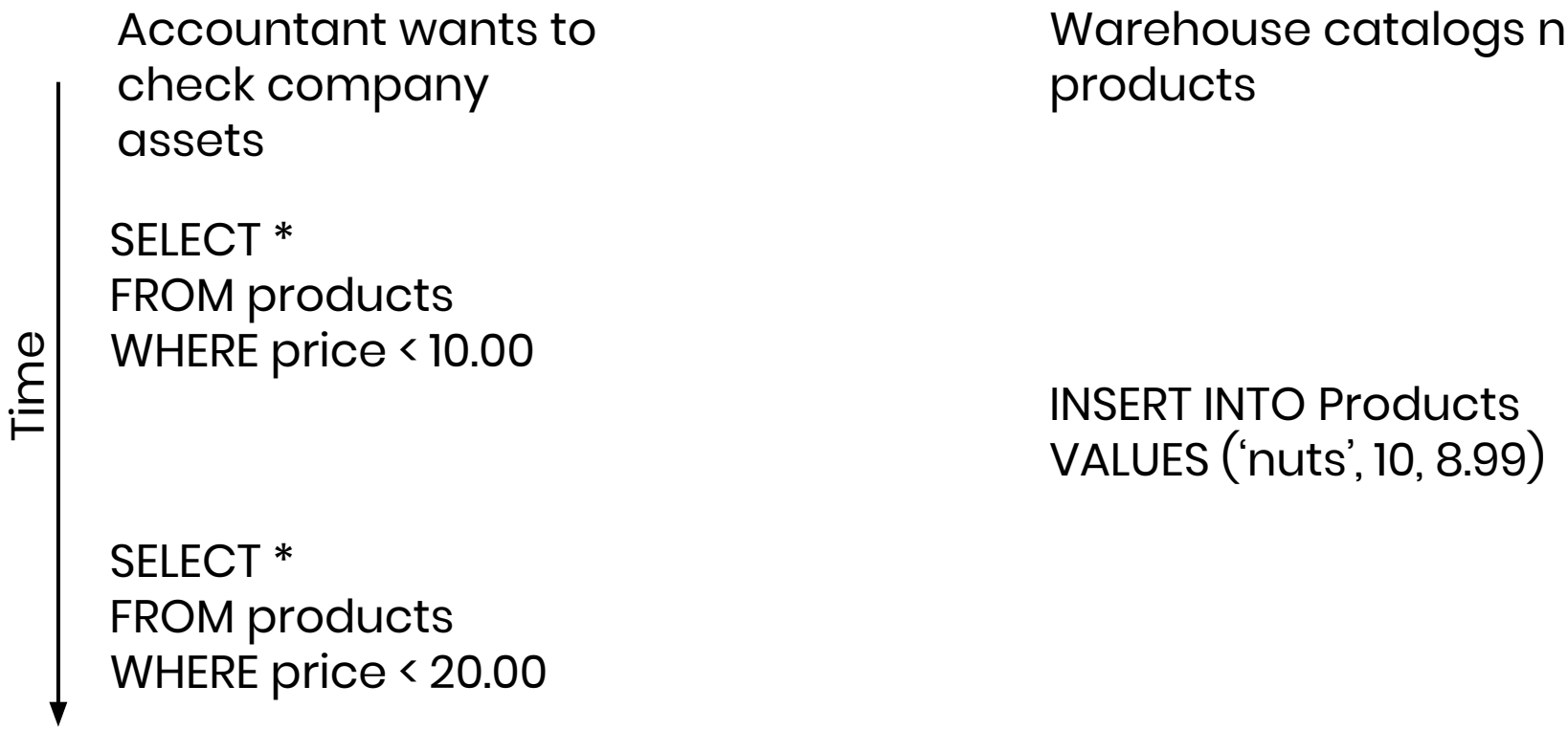
Unrepeatable Read

- Read-Write (RW) conflict
- Asset checking scenario:



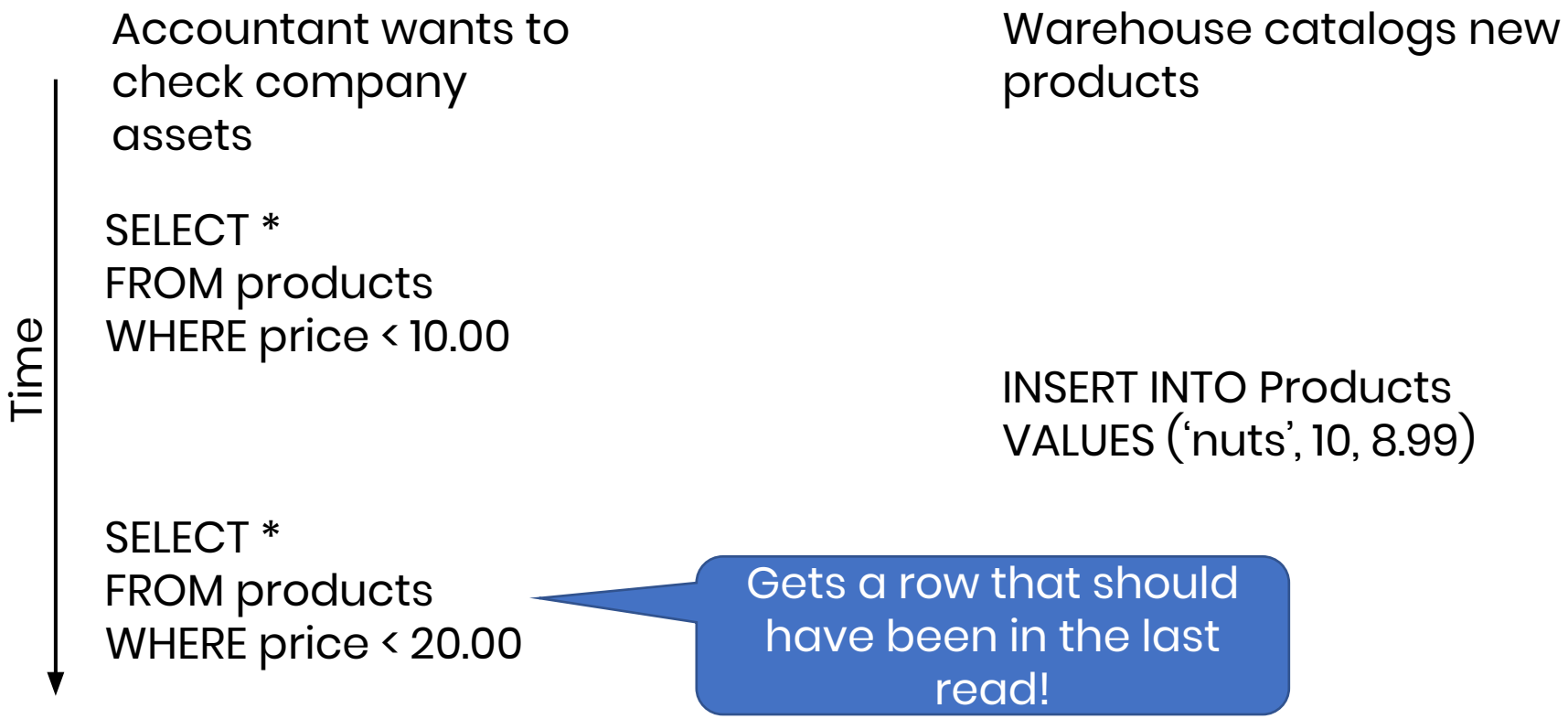
Phantom Read

- Same read has more rows
- Asset checking scenario:



Phantom Read

- Same read has more rows
- Asset checking scenario:



ACID

- Atomic
- Consistent
- Isolated
- Durable

- Ideally a DBMS follows these principles, but sacrificing some behavior for performance gains is common
- Definitely needs to follow these principles if you are dealing with $$$

Atomic

- Operation encapsulation
- A transaction is atomic if the state shows all the effects of a transaction, or none of them (all or nothing)

Consistent

- Integrity constraints and application specification
- A transaction begins with a valid database state and ends with a valid database state

Isolated

- Concurrency management
- Running transactions has the same effect as if we ran each transaction one after another

Durable

- Crash recovery
- Once a transaction has been committed, its effects remain in the database
- CSE 444 topic, not discussed in this class

Transactions

- A collection of statements that are executed atomically (logically speaking!)
  - A single application function may involve multiple different operations
  - Transactions let them execute properly **together as if it were a single action**

```
BEGIN TRANSACTION
 [SQL Statements]
COMMIT -- finalizes execution
```

```
BEGIN TRANSACTION
 [SQL Statements]
ROLLBACK -- undo everything
```

```
[Single SQL Statement]
```

Transactions

- A collection of statements that are executed atomically (logically speaking!)
  - A single application function may involve multiple different operations
  - Transactions let them execute properly **together as if it were a single action**

```
BEGIN TRANSACTION
 [SQL Statements]
COMMIT -- finalizes execution
```

```
BEGIN TRANSACTION
 [SQL Statements]
ROLLBACK -- undo everything
```

```
[Single SQL Statement]
```

By default, without BEGIN, each statement is its own transaction

Do I need to implement this?

- **DBMS concurrency control is all based on specification**
- Merely specifying what your transactions are is good enough for the DBMS to take care of it as a single unit

Transaction Modeling

- Logical perspective a database is a set of sets/bags of tuples
- Design perspective a database is a schema that models information
- Physical perspective a database is a catalog of organized files
- Transaction perspective a database is a **collection of elements** that can be **written to** or **read from**
  - Element granularity can vary depending on DBMS and/or user specification
  - Transactions are sequences of element reads and/or writes

Schedules

- Transactions are sequence of element reads and/or writes
  - $R_i(A)$ □ transaction i **reads** element A
  - $W_i(A)$ □ transaction i **updates** element A
  - $I_i(A)$ □ transaction i **inserts** an element A
  - $D_i(A)$ □ transaction i **deletes** an element A
- Schedules are a sequence of interleaved actions from all transactions

Schedules

- Transactions are sequence of element reads and/or writes
  - $R_i(A)$ □ transaction i **reads** element A
  - $W_i(A)$ □ transaction i **updates** element A
  - $I_i(A)$ □ transaction i **inserts** an element A
  - $D_i(A)$ □ transaction i **deletes** an element A
- Schedules are a sequence of interleaved actions from all transactions

Serial Schedules

- A **serial schedule** is a schedule where each transaction would be executed in some order

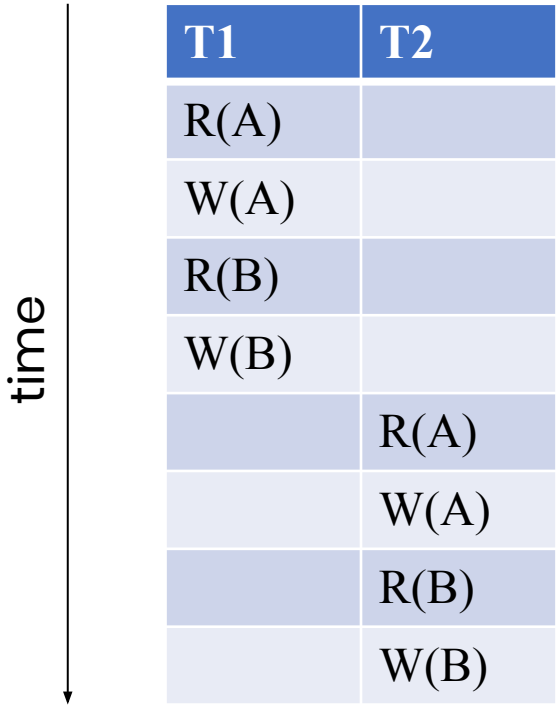
Transaction Schedule

T1	T2
R(A)	R(A)
W(A)	W(A)
R(B)	R(B)
W(B)	W(B)

Serial Schedule Example

- T1 then T2

$R_1(A), W_1(A), R_1(B), W_1(B), R_2(A), W_2(A), R_2(B), W_2(B)$



T1	T2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

Serial Schedule Example

- T2 then T1

$R_2(A), W_2(A), R_2(B), W_2(B), R_1(A), W_1(A), R_1(B), W_1(B)$

T1	T2
	R(A)
	W(A)
	R(B)
	W(B)
R(A)	
W(A)	
R(B)	
W(B)	

Serial Schedules

- A **serial schedule** is a schedule where each transaction would be executed in some order
- A **serializable schedule** is a schedule that is equivalent to a serial schedule
  - If the schedule were executed and you we given a before and after, you would not be able to tell if the transactions were interleaved

Serializable Schedules

- Serializable to T1 then T2

$R_1(A), W_1(A), R_2(A), W_2(A), R_1(B), W_1(B), R_2(B), W_2(B)$

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

Serializable Schedules

- Serializable to T1 then T2

$R_1(A), W_1(A), R_2(A), W_2(A), R_1(B), W_1(B), R_2(B), W_2(B)$

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

Looks like T2 finished after T1 for each element

Serializable Schedules

- Not serializable to either order

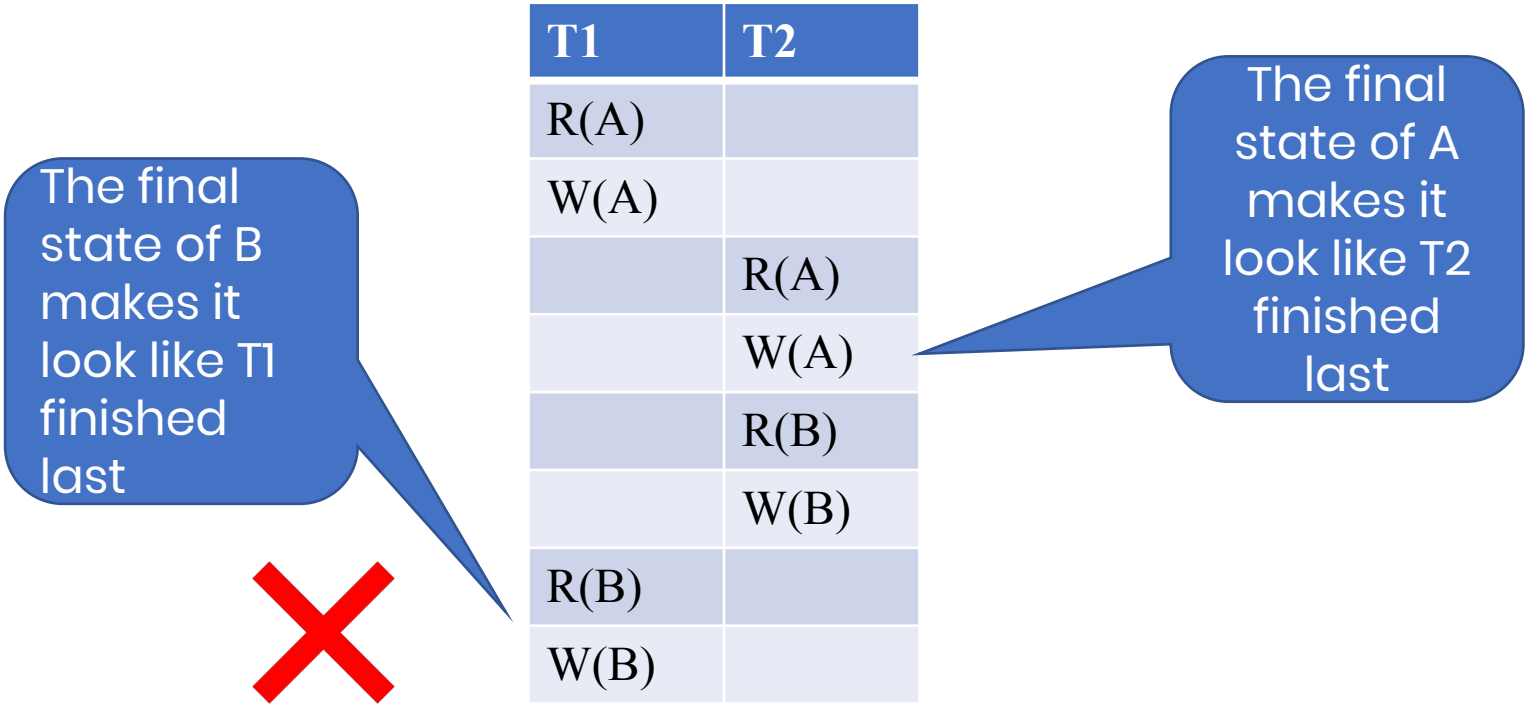
$R_1(A), W_1(A), R_2(A), W_2(A), R_2(B), W_2(B), R_1(B), W_1(B)$

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
R(B)	
W(B)	

Serializable Schedules

- Not serializable to either order

$R_1(A), W_1(A), R_2(A), W_2(A), R_2(B), W_2(B), R_1(B), W_1(B)$



Checking Serializability

- How does the DBMS tell if some schedule is serializable?

Conflicts

Conflict

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

Conflicts

- In terms of the application concurrency problems we saw:
- Lost Update Write-Write (WW) conflict
- Dirty Read Write-Read (WR) conflict
- Unrepeatable Read Read-Write (RW) conflict
- Phantom Read
  - We'll talk about this later

Individual conflicts aren't "bad"! We expect that different transactions have conflicting actions. The problem comes with interleaving the conflicts.

Types of Conflicts

- Changing the order of things in conflict will cause program behavior to behave badly
- **Intra-transaction conflicts**
  - Operations within a transaction cannot be swapped (you would be literally changing the program)
- **Inter-transaction conflicts**
  - WW conflicts □ W1(X), W2(X)
  - WR conflicts □ W1(X), R2(X)
  - RW conflicts □ R1(X), W2(X)

Types of Conflicts

- Changing the order of things in conflict will cause program behavior to behave badly
- **Intra-transaction conflicts**
  - Operations within a transaction cannot be swapped (you would be literally changing the program)
- **Inter-transaction conflicts**
  - WW conflicts □ W1(X), W2(X)
  - WR conflicts □ W1(X), R2(X)
  - RW conflicts □ R1(X), W2(X)

Note what's missing: actions on different elements or reading the same element. are NOT conflicts.

Serial Schedules

- A **serial schedule** is a schedule where each transaction would be executed in some order
- A **serializable schedule** is a schedule that is equivalent to a serial schedule
  - If the schedule were executed and you we given a before and after, you would not be able to tell if the transactions were interleaved
- A **conflict serializable schedule** is a schedule that can be transformed into a serial schedule by performing a series of swaps of adjacent non-conflicting actions

Conflict Serializable Schedule Example

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

Conflict Serializable Schedule Example

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

Conflict Serializable Schedule Example

T1	T2
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)
W(B)	
	R(B)
	W(B)

Conflict Serializable Schedule Example

T1	T2
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)
W(B)	
	R(B)
	W(B)

Conflict Serializable Schedule Example

T1	T2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)
W(B)	
	R(B)
	W(B)

Conflict Serializable Schedule Example

T1	T2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)
W(B)	
	R(B)
	W(B)

Conflict Serializable Schedule Example

T1	T2
R(A)	
W(A)	
R(B)	
	R(A)
W(B)	
	W(A)
	R(B)
	W(B)

Conflict Serializable Schedule Example

T1	T2
R(A)	
W(A)	
R(B)	
	R(A)
W(B)	
	W(A)
	R(B)
	W(B)

Conflict Serializable Schedule Example

T1	T2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

Non Conflict Serializable Schedule Example

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
R(B)	
W(B)	

Non Conflict Serializable Schedule Example

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
R(B)	
	W(B)
W(B)	

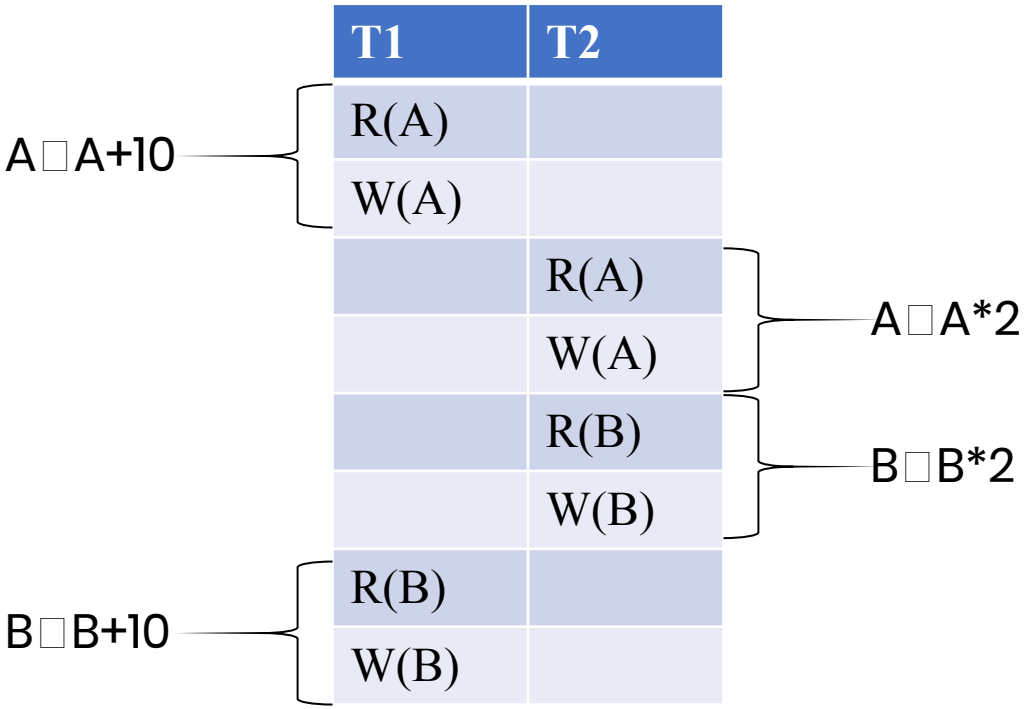
Conflict rule broken!

Conflict Serializability

- Showing program serializability is hard
  - Needs lots of extra information besides R, W, I, D
- Observation: Enforce something something simpler but stronger than serializability
- **Conflict serializability implies serializability**
- Serializability does not imply conflict serializability

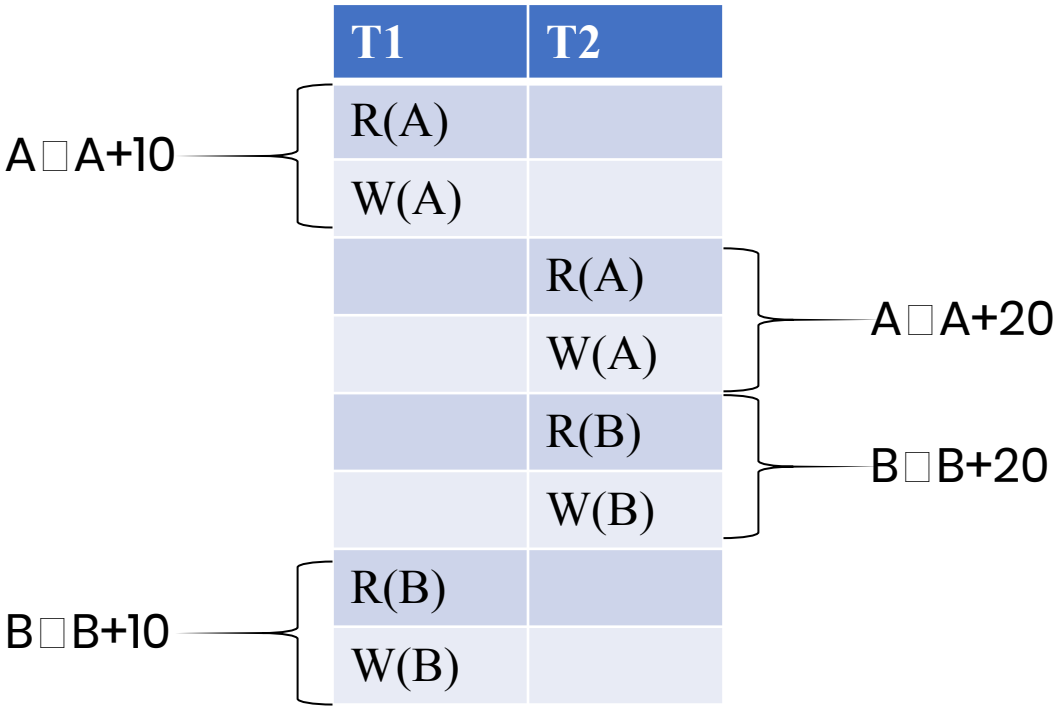
Serializable vs Conflict Serializable

Not serializable nor conflict serializable



Serializable vs Conflict Serializable

Serializable but not conflict serializable



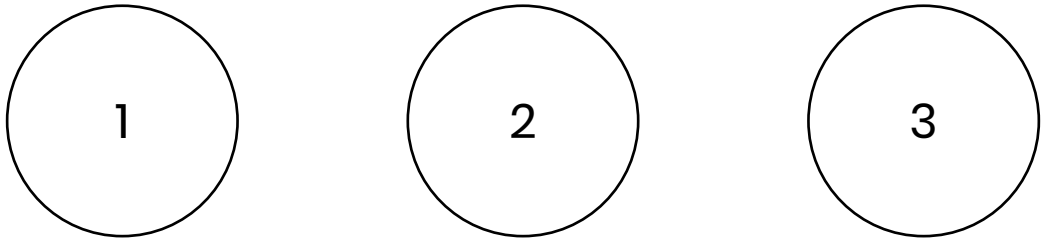
Enforcing Conflict Serializability

- We only care if some conflict rule would be broken
- But we need an efficient algorithm

- Method:
  - Model each transaction as a node
  - Model a inter-transaction conflict as a directed edge
  - If the resulting graph is a DAG then there is a serial order
  - Conflict serializability enforcement turns into the graph cycle detection problem

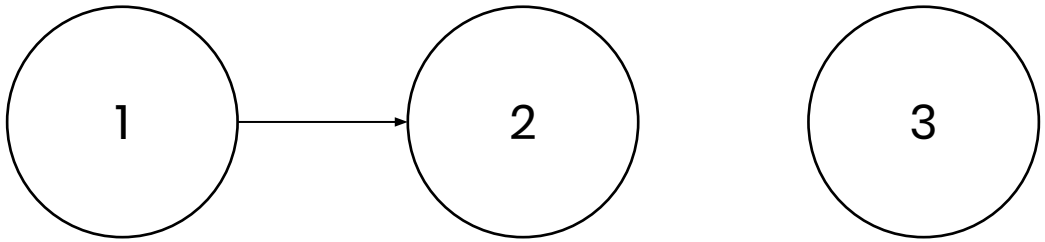
Example of Checking Conflict Serializability

$R_1(A), W_2(A), W_2(C), R_3(A), W_3(A), R_3(B), W_1(B)$



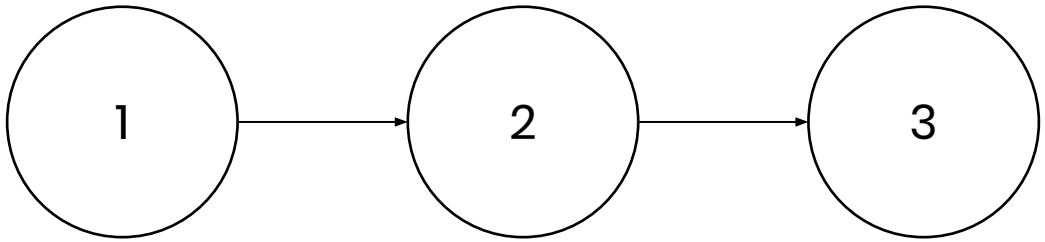
Example of Checking Conflict Serializability

$R_1(A), W_2(A), W_2(C), R_3(A), W_3(A), R_3(B), W_1(B)$



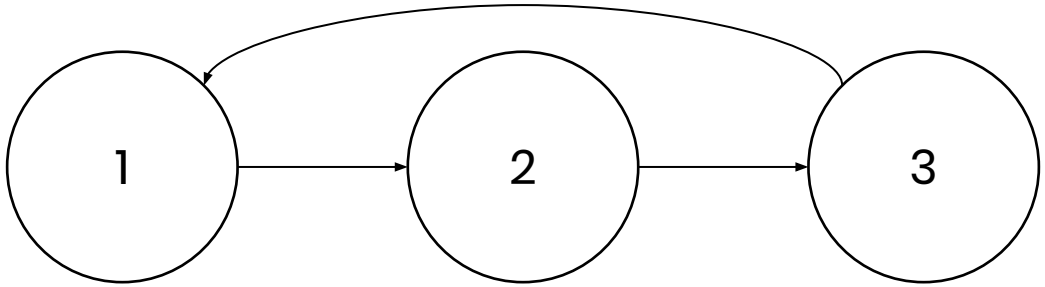
Example of Checking Conflict Serializability

$R_1(A), W_2(A), W_2(C), R_3(A), W_3(A), R_3(B), W_1(B)$



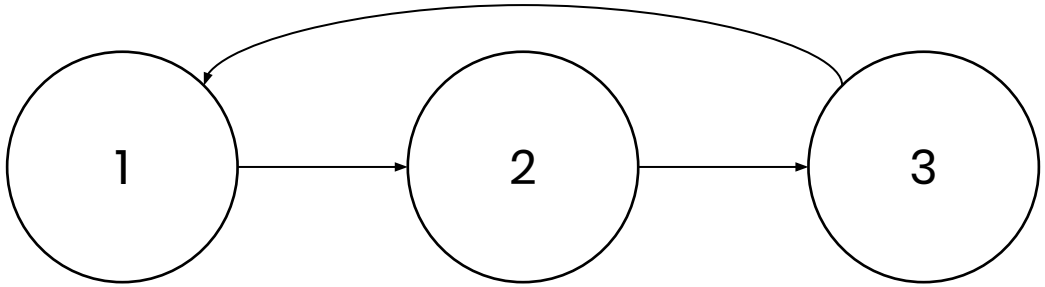
Example of Checking Conflict Serializability

$R_1(A), W_2(A), W_2(C), R_3(A), W_3(A), R_3(B), W_1(B)$



Example of Checking Conflict Serializability

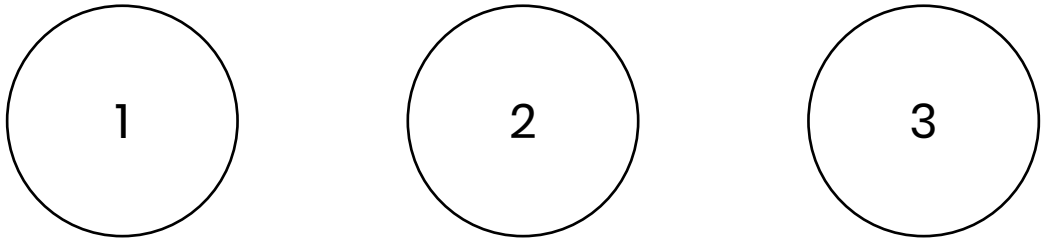
$R_1(A), W_2(A), W_2(C), R_3(A), W_3(A), R_3(B), W_1(B)$



Cycle □ Not conflict serializable

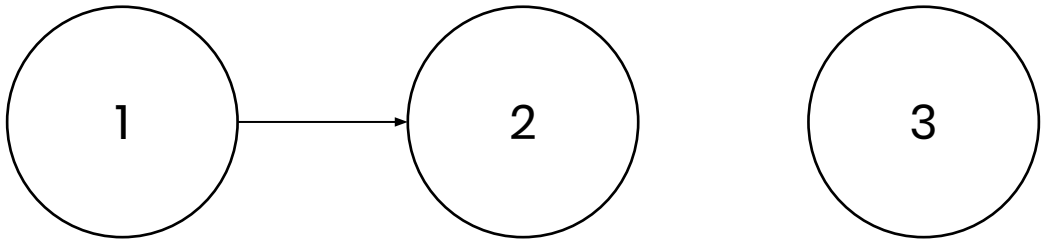
Example of Checking Conflict Serializability

$R_1(X), W_2(X), W_1(Y), W_2(Y), W_3(Y), R_3(X), W_3(X)$



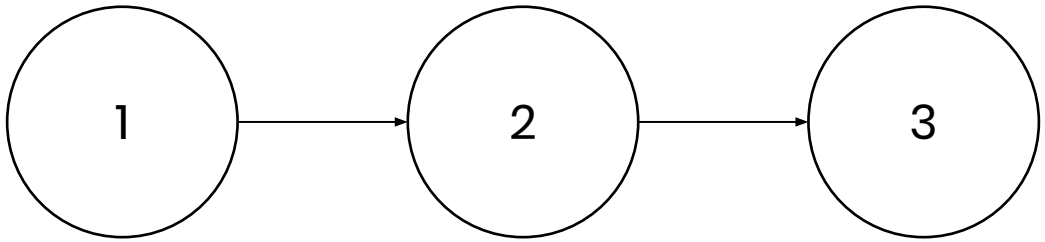
Example of Checking Conflict Serializability

$R_1(X), W_2(X), W_1(Y), W_2(Y), W_3(Y), R_3(X), W_3(X)$



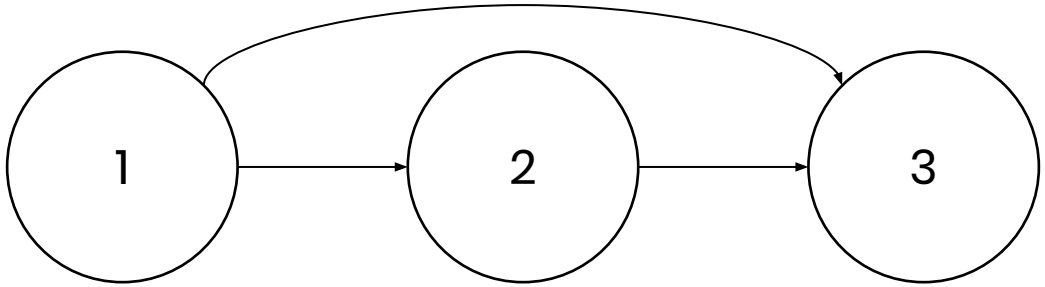
Example of Checking Conflict Serializability

$R_1(X), W_2(X), W_1(Y), W_2(Y), W_3(Y), R_3(X), W_3(X)$



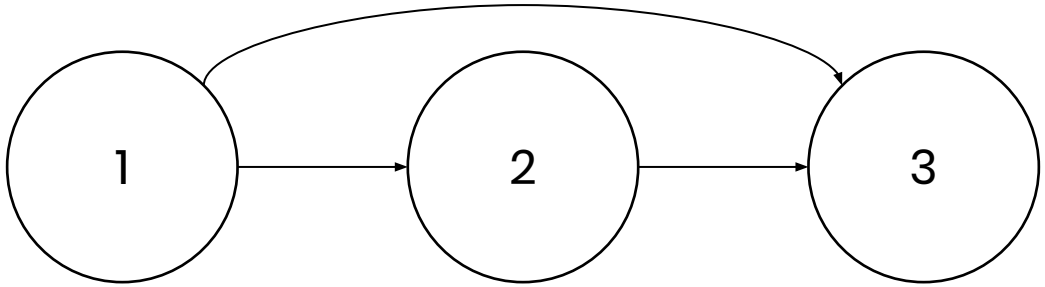
Example of Checking Conflict Serializability

$R_1(X), W_2(X), W_1(Y), W_2(Y), W_3(Y), R_3(X), W_3(X)$



Example of Checking Conflict Serializability

$R_1(X), W_2(X), W_1(Y), W_2(Y), W_3(Y), R_3(X), W_3(X)$



DAG □ Conflict serializable □ Serializable

Takeaways

- When a database has multiple concurrent users, a variety of conflicts can happen.
- DBMSs implement transactions with ACID properties to avoid anomalies while allowing concurrent users.