**Transactions:**

-Used to support multiple users

**Common Concurrency Problems**

Schedule conflicts can be resolved

Conflict Serializable is stronger than Serializable

**Precedence Graph:**

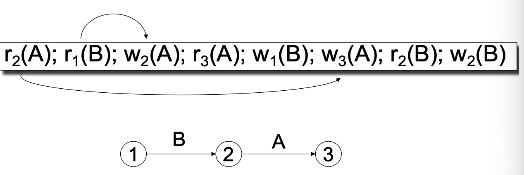
Schedule is conflict serializable if precedence graph is acyclic

**Conflict Serializable Example:**

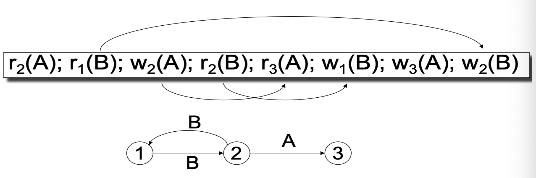
Not conflict serializable:

-**Non-Atomic:** Multiple users writing table at once

-**Lost Update (WW conflict):** One user’s write affects other user’s writes

-**Dirty Read(WR conflict):** User’s read affected by another user writing at the same time

**Unrepeatable Read (RW)**: User’s 2 reads one after another are different because another user writes data in the middle

**Phantom Read:** User’s 2 reads one after another are different because another user inserted data in the middle

**ACID (A + I = functional transactions):**

DBMS try to follow ACID, but performance gains can be done by sacrificing some parts of it

**Atomic (2nd most important):** Either everything in the transaction runs or nothing runs

**Transactions:  Locking**

-Locking is a “pessimistic” scheduler

-Assumes transaction executions will conflict

-Expensive overhead cost but cheap aborting process

-Locks allow specific transactions to do operations

-Must acquire lock before reading or writing → e.g.  L1(X)

-Must eventually release locks → e.g. U1(X)

-“Blocking” happens when a transaction finds another transaction with an existing lock

-2 Phase Locking (2PL) - Within every transaction, all lock requests must precede all unlock requests

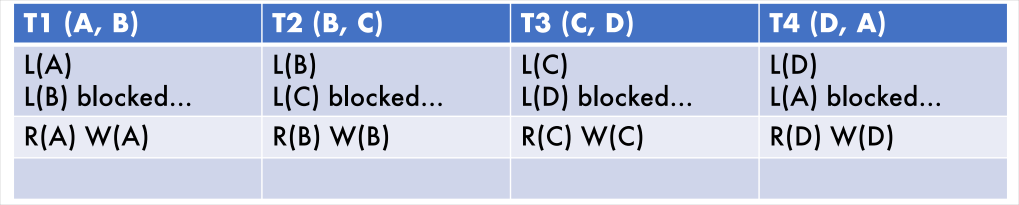
-Doesn’t guarantee ACID

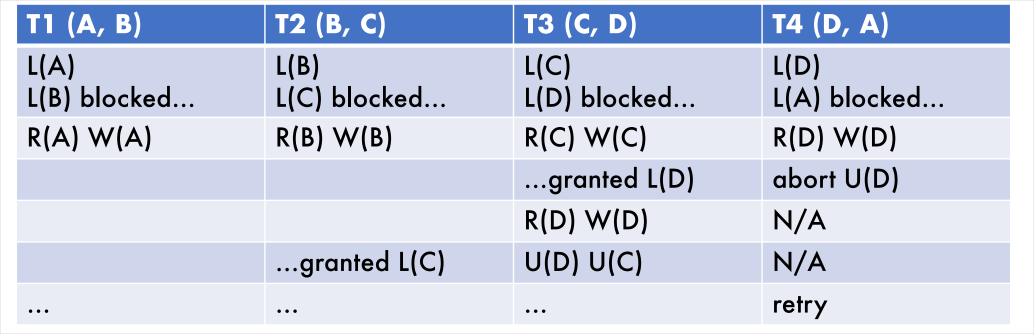
-Ensures conflict serializability

Regular locking vs 2PL

-Deadlock is a situation in which two or more transactions are waiting indefinitely for one another to give up locks

-Abort (rollback) to bypass deadlock and retry later





Strict 2PL



**Consistent:** Integrity constraints and app specs are followed

**Isolated (Most Important):** Operations runs are not affected by other operations

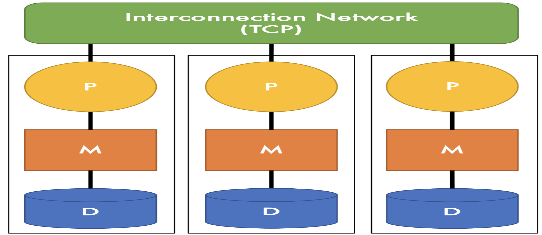
**Durable:** Crash Recovery

-Expensive to scale on, Easy to implement

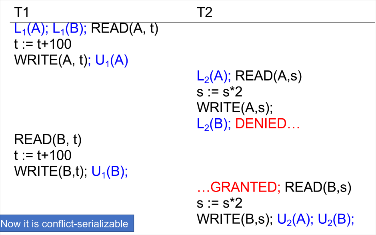
Shared-Nothing Architecture

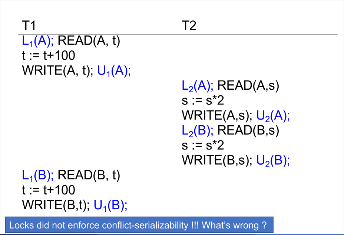
Transactions combine multiple operations as one action

Transactions either commit all operations or revert back to original data (Atomic)

DBMS concurrency control is based on user transaction specifications

Database perspectives

**Logical:** A set of sets of tuples

**Design:** A schema that models information

**Physical:** A catalog of organized files

-Theoretically can scale infinitely, hard to implement on

**Inter-Query Parallelism** (Shared-memory Architecture)

-Each transaction is processed on a separate node

-Scales very well for lots of simple transactions

**Intra-Query Parallelism (Shared-Nothing Architecture)**

-Specifically, intra-operator parallelism

-Each operator is processed by multiple nodes

-Scales well in general

**Unpartitioned Table** - Entire table on just one node in the system

**Block Partitioning** - Tuples are horizontally partitioned by raw size (B(R) / N nodes = # of blocks on each node)

**Hash Partitioning** - Node contains tuples with chosen attribute hashes (hash % N decides which node)

**Range Partitioning** - Node contains tuples in a chosen attribute’s range

**Transaction:** A collections of elements that can be written to or read from

Possible Actions:

**R(A): Read element A**

**W(A): Update element A**

**I(A): Insert element A**

**D(A): Delete element A**

Schedules are a series of actions

**Serial Schedule:** Transactions are executed one after another

**Serializable Schedule:** Transactions are executed at the same time but can be converted to a Serial Schedule and have the same result

**Inter-transaction Conflicts:**

**WW conflict: Lost Update**

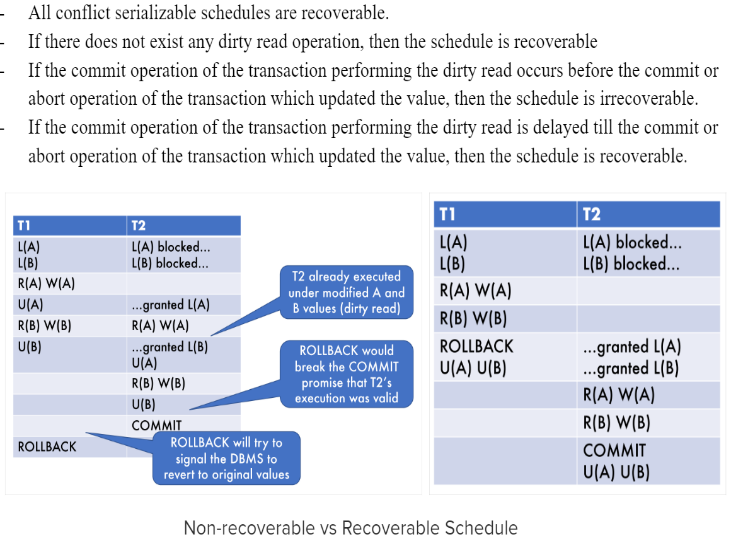
**WR conflict: Dirty Read**

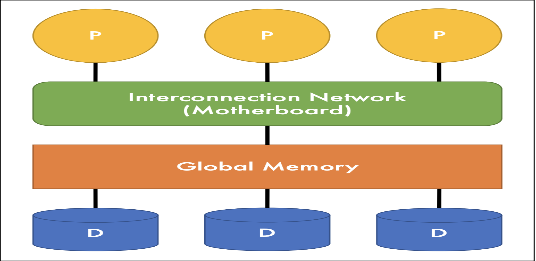
**RW conflict: Unrepeatable Read**

Phantom Read (RR conflict but don’t check for it)

Conflict Serializability:

Recoverbility:

Implies serializability



**Parallel Processing:**

**Common reasons for sublinear performance:**

-Overhead cost--(Starting and coordinating operations on many nodes)

-Interference/Contention--(Shared resources are not perfectly split)

-Skew: Process is only as fast as the slowest node

-Shared-memory Architecture: