

Lecture 13: Two Phase Locking

CREATING THE NEXT®

Today's Agenda

Two Phase Locking

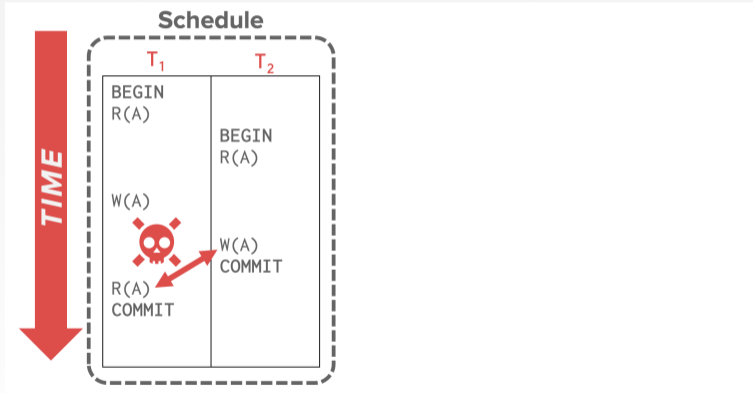
- 1.1 Recap
- 1.2 Lock Types
- 1.3 Two-Phase Locking
- 1.4 Deadlock Detection + Prevention
- 1.5 Hierarchical Locking
- 1.6 Locking in Practice
- 1.7 Conclusion

Recap

Formal Properties of Schedules

- Conflict Serializable
 - ▶ Verify using either the "swapping" method or dependency graphs.
 - ▶ Any DBMS that says that they support "serializable" isolation does this.
- View Serializable
 - ▶ No efficient way to verify.
 - ▶ No DBMS supports this.

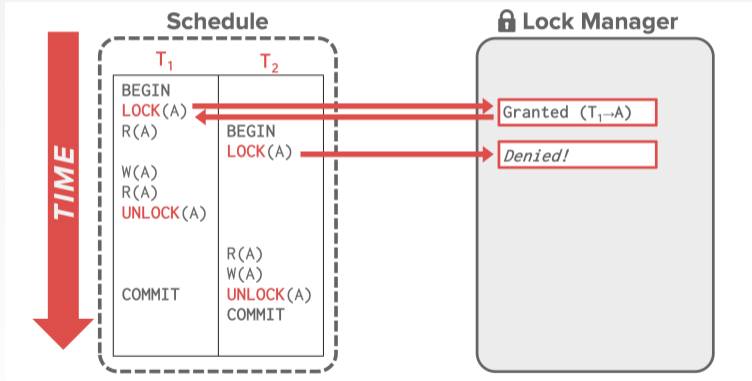
Example: Entire Schedule



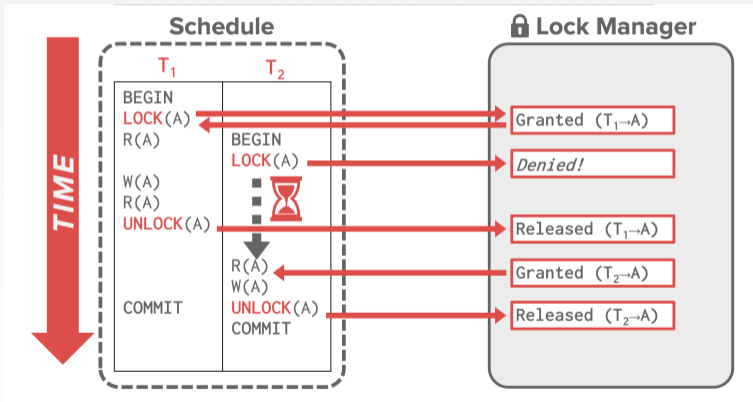
Observation

- We need a way to guarantee that all execution schedules are correct (*i.e.*, serializable) without knowing the entire schedule ahead of time.
- Solution: Use locks to protect database objects.

Executing with Locks



Executing with Locks



Today's Agenda

- Lock Types
- Two-Phase Locking
- Deadlock Detection + Prevention
- Hierarchical Locking
- Locking in Practice

Lock Types

Locks vs. Latches

	Locks	Latches
Separate...	User transactions	Threads
Protect...	Database Contents	In-Memory Data Structures
During...	Entire Transactions	Critical Sections
Modes...	Shared, Exclusive, Update, Intention	Read, Write (<i>a.k.a.</i> , Shared, Exclusive)
Deadlock	Detection & Resolution	Avoidance
...by...	Waits-for, Timeout, Aborts	Coding Discipline
Kept in...	Lock Manager	Protected Data Structure

Reference

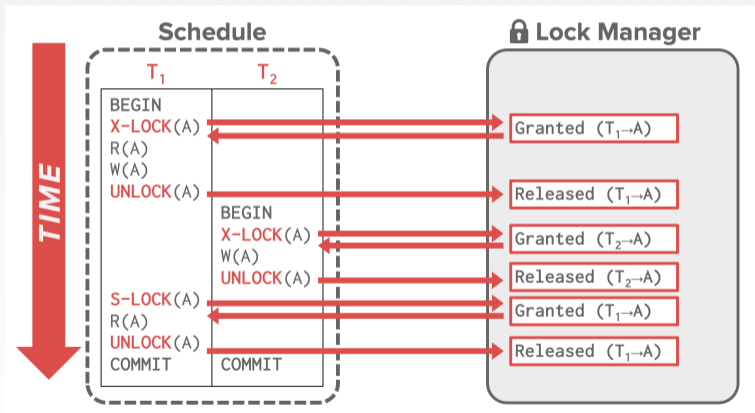
Basic Lock Types

- **S-LOCK**: Shared locks for reads.
- **X-LOCK**: Exclusive locks for writes.

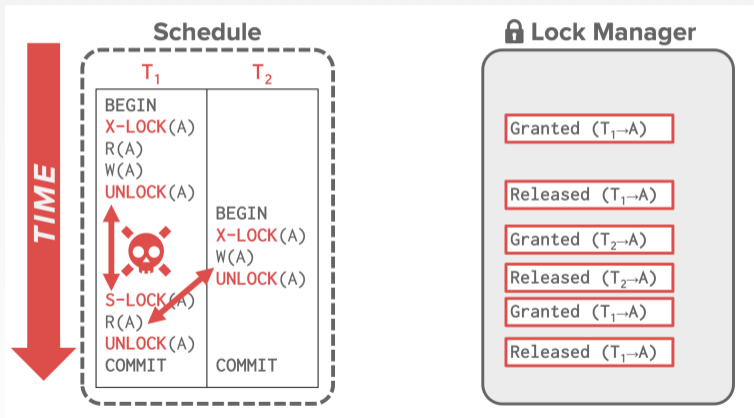
Executing with Locks

- Transactions request locks (or upgrades).
- Lock manager grants or blocks requests.
- Transactions release locks.
- Lock manager updates its internal lock-table.
 - ▶ It keeps track of what transactions hold what locks and what transactions are waiting to acquire any locks.

Executing with Locks: Not Sufficient



Executing with Locks: Not Sufficient



Two-Phase Locking

Concurrency Control Protocol

- Two-phase locking (2PL) is a concurrency control protocol that determines whether a txn can access an object in the database on the fly.
- The protocol does **not** need to know all the queries that a txn will execute ahead of time.

Two-Phase Locking

- **Phase 1: Growing**

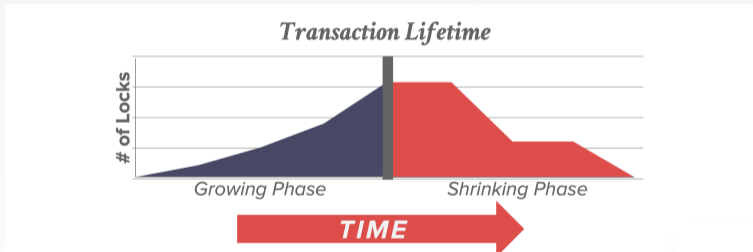
- ▶ Each txn requests the locks that it needs from the DBMS's lock manager.
- ▶ The lock manager grants/denies lock requests.

- **Phase 2: Shrinking**

- ▶ The txn is allowed to only release locks that it previously acquired. It cannot acquire new locks.

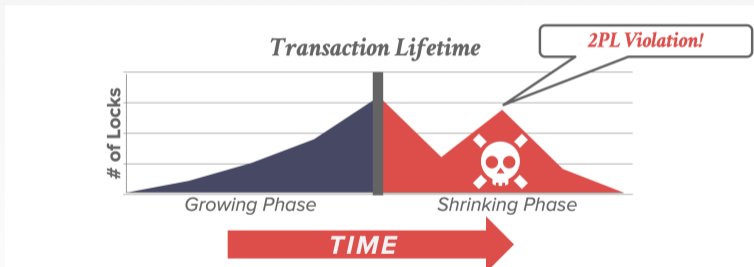
Two-Phase Locking

- The txn is not allowed to acquire/upgrade locks after the growing phase finishes.

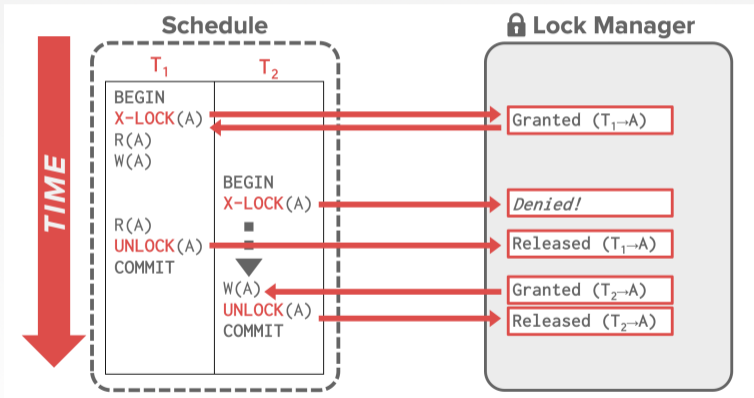


Two-Phase Locking

- The txn is not allowed to acquire/upgrade locks after the growing phase finishes.



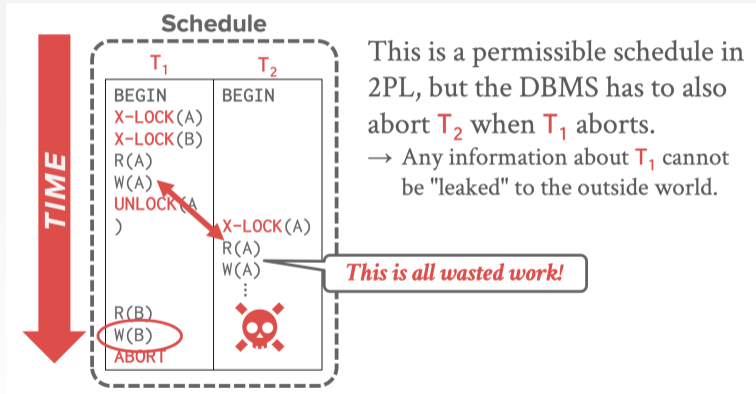
Executing with 2PL



Two-Phase Locking

- 2PL on its own is sufficient to guarantee conflict serializability.
 - ▶ It generates schedules whose precedence graph is acyclic.
- But it is subject to cascading aborts.

2PL – Cascading Aborts

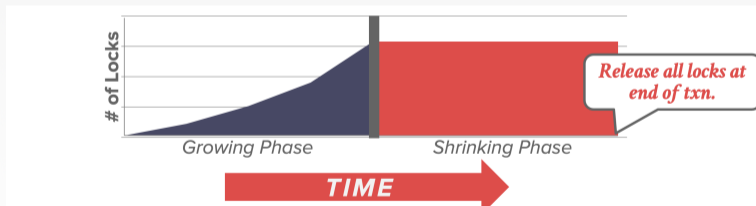


2PL: Observations

- There are potential schedules that are serializable but would not be allowed by 2PL.
 - ▶ Locking limits concurrency.
- May still have "dirty reads".
 - ▶ Solution: Strong Strict 2PL (aka Rigorous 2PL)
- May lead to deadlocks.
 - ▶ Solution: Detection or Prevention

Strong Strict Two-Phase Locking

- The txn is not allowed to acquire/upgrade locks after the growing phase finishes.
- Allows only conflict serializable schedules, but it is often stronger than needed for some apps.

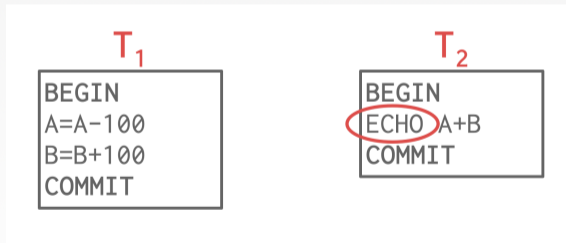


Strong Strict Two-Phase Locking

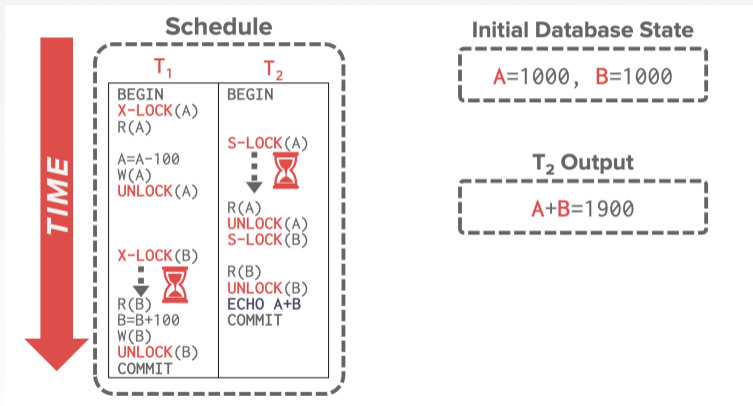
- A schedule is **strict** if a value written by a txn is not read or overwritten by other txns until that txn finishes.
- Advantages:
 - ▶ Does not incur cascading aborts.
 - ▶ Aborted txns can be undone by just restoring original values of modified tuples.

Examples

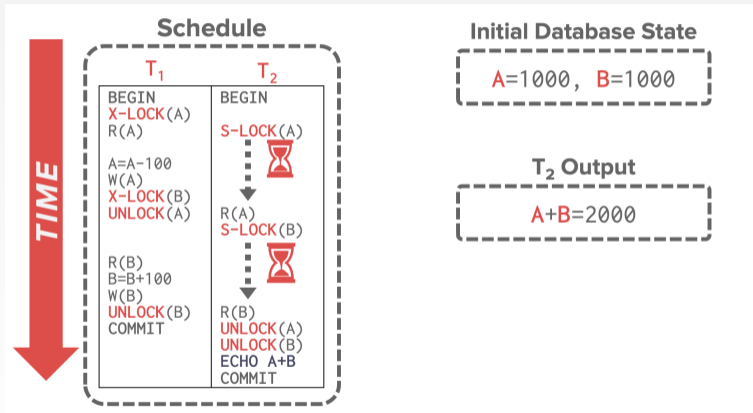
- T1 – Move \$100 from A's account to B's account.
- T2 – Compute the total amount in all accounts and return it to the application.



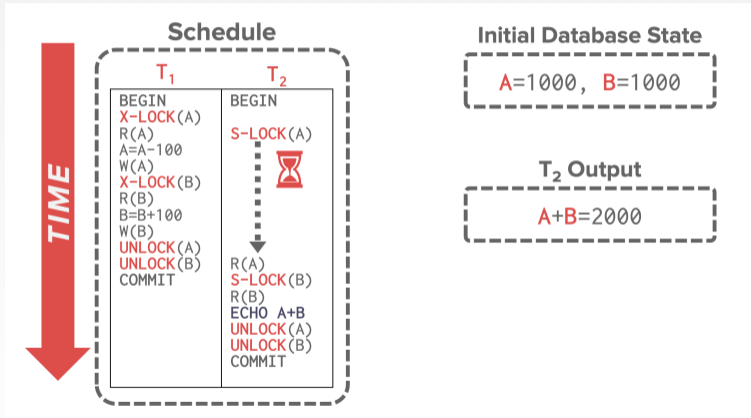
Non-2PL Example



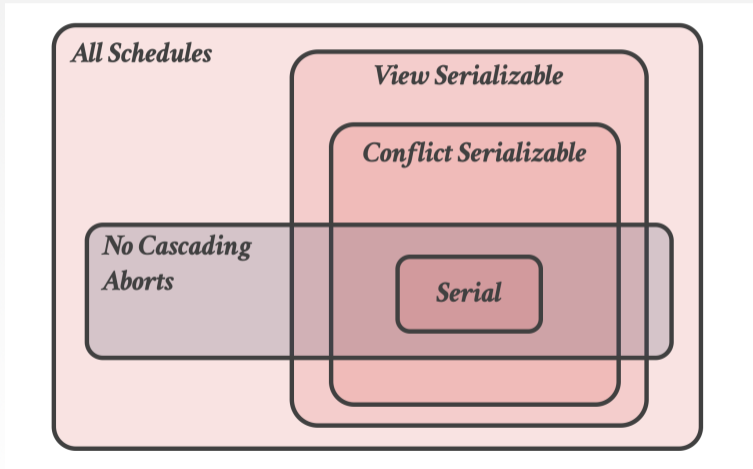
2PL Example



Strong Strict 2PL Example



Universe of Schedules

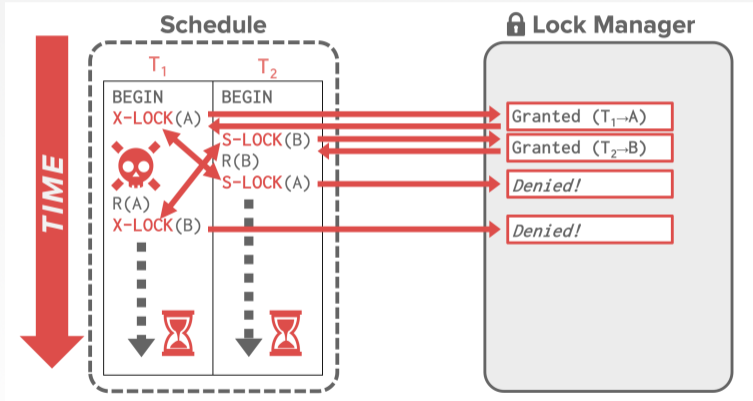


2PL: Observations

- There are potential schedules that are serializable but would not be allowed by 2PL.
 - ▶ Locking limits concurrency.
- May still have "dirty reads".
 - ▶ Solution: Strong Strict 2PL (Rigorous)
- May lead to deadlocks.
 - ▶ Solution: Detection or Prevention

Deadlock Detection + Prevention

Deadlocks



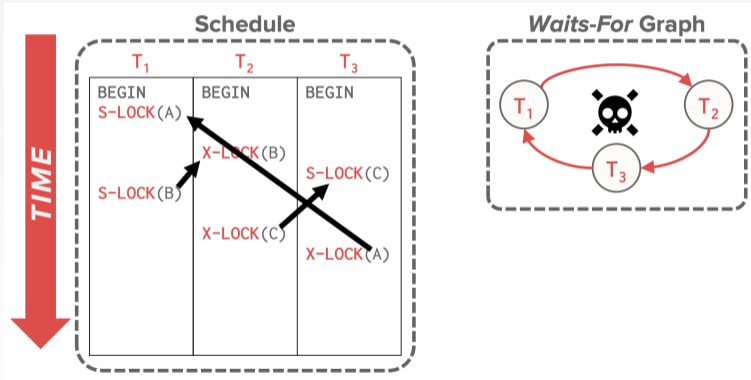
2PL Deadlocks

- A deadlock is a cycle of transactions waiting for locks to be released by each other.
- Two ways of dealing with deadlocks:
 - ▶ Approach 1: Deadlock Detection
 - ▶ Approach 2: Deadlock Prevention

Deadlock Detection

- The DBMS creates a waits-for graph to keep track of what locks each txn is waiting to acquire:
 - ▶ Nodes are transactions
 - ▶ Edge from T_i to T_j if T_i is waiting for T_j to release a lock.
- The system periodically checks for cycles in waits-for graph and then decides how to break it.

Deadlock Detection



Deadlock Handling

- When the DBMS detects a deadlock, it will select a "victim" txn to rollback to break the cycle.
- The victim txn will either restart or abort(more common) depending on how it was invoked.
- There is a trade-off between the frequency of checking for deadlocks and how long txns have to wait before deadlocks are broken.

Deadlock Handling: Victim Selection

- Selecting the proper victim depends on a lot of different variables. . . .
 - ▶ By age (lowest timestamp)
 - ▶ By progress (least/most queries executed)
 - ▶ By the of items already locked
 - ▶ By the of txns that we have to rollback with it
- We also should consider the of times a txn has been restarted in the past to prevent starvation.

Deadlock Handling: Rollback Length

- After selecting a victim txn to abort, the DBMS can also decide on how far to rollback the txn's changes.
- Approach 1: Completely
- Approach 2: Minimally (*i.e.*, release a subset of locks)

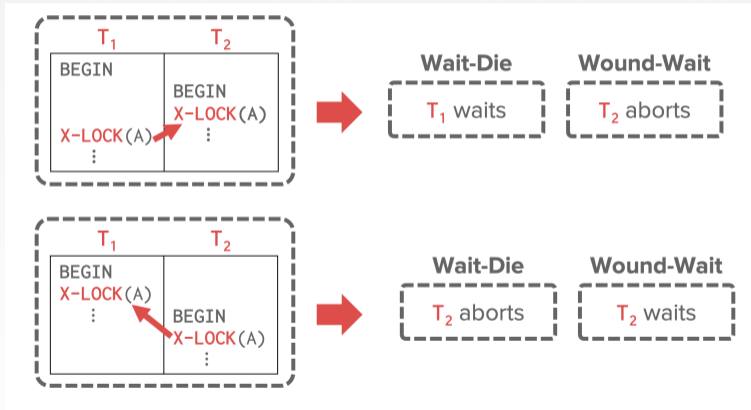
Deadlock Prevention

- When a txn tries to acquire a lock that is held by another txn, the DBMS kills one of them to prevent a deadlock.
- This approach does **not** require a waits-for graph or detection algorithm.

Deadlock Prevention

- Assign priorities based on timestamps:
 - ▶ Older Timestamp = Higher Priority (e.g., $T1 > T2$)
- **Wait-Die** ("Old Waits for Young")
 - ▶ If requesting txn has higher priority than holding txn, then requesting txn waits for holding txn.
 - ▶ Otherwise requesting txn aborts.
- **Wound-Wait** ("Young Waits for Old")
 - ▶ If requesting txn has higher priority than holding txn, then holding txn aborts and releases lock.
 - ▶ Otherwise requesting txn waits.

Deadlock Prevention



Deadlock Prevention

- Why do these schemes guarantee no deadlocks?
- Only one "type" of direction allowed when waiting for a lock.
- When a txn restarts, what is its (new) priority?
- Its original timestamp. Why?

Observation

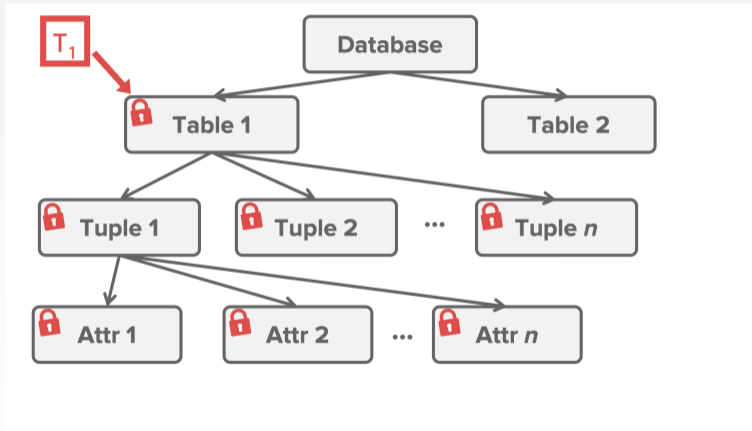
- All of these examples have a one-to-one mapping from database objects to locks.
- If a txn wants to update one billion tuples, then it has to acquire one billion locks.

Hierarchical Locking

Lock Granularities

- When we say that a txn acquires a “lock”, what does that actually mean?
 - ▶ On an Attribute? Tuple? Page? Table?
- Ideally, each txn should obtain fewest number of locks that is needed. . .

Database Lock Hierarchy



Example

- T1 – Get the balance of A's account.
- T2 – Increase B's bank account balance by 1%.
- What locks should these txns obtain?
- Multiple:
 - ▶ Exclusive + Shared for leafs of lock tree.
 - ▶ Special Intention locks for higher levels.

Intention Locks

- An intention lock allows a higher level node to be locked in shared or exclusive mode without having to check all descendent nodes.
- If a node is in an intention mode, then explicit locking is being done at a lower level in the tree.

Intention Locks

- **Intention-Shared** (IS)
 - ▶ Indicates explicit locking at a lower level with shared locks.
- **Intention-Exclusive** (IX)
 - ▶ Indicates locking at lower level with exclusive or shared locks.

Intention Locks

- **Shared+Intention-Exclusive** (SIX)
 - ▶ The subtree rooted by that node is locked explicitly in **shared mode** and explicit locking is being done at a lower level with **exclusive-mode** locks.

Compatibility Matrix

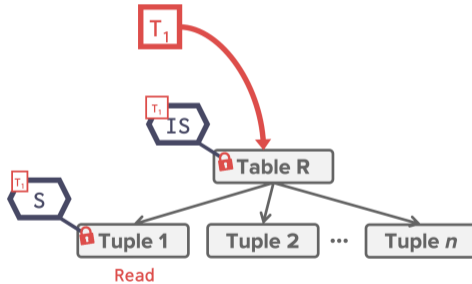
		T_2 Wants				
		IS	IX	S	SIX	X
T_1 Holds	IS	✓	✓	✓	✓	✗
	IX	✓	✓	✗	✗	✗
	S	✓	✗	✓	✗	✗
	SIX	✓	✗	✗	✗	✗
	X	✗	✗	✗	✗	✗

Hierarchical Locking Protocol

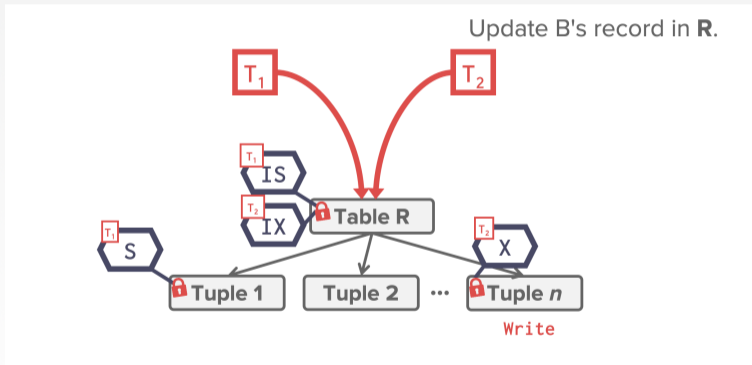
- Each txn obtains appropriate lock at highest level of the database hierarchy.
- To get S or IS lock on a node, the txn must hold at least IS on parent node.
- To get X, IX, or SIX on a node, must hold at least IX on parent node.

Example – Two-Level Hierarchy

Read A's record in R.



Example – Two-Level Hierarchy

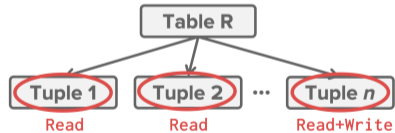


Example – Three Transactions

- Assume three txns execute at same time:
 - ▶ T1 – Scan R and update a few tuples.
 - ▶ T2 – Read a single tuple in R .
 - ▶ T3 – Scan all tuples in R .

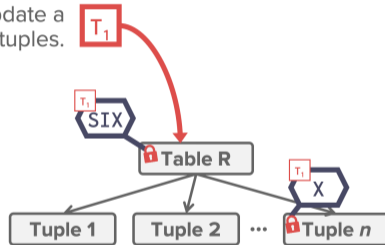
Example – Three Transactions

Scan **R** and update a few tuples. T_1

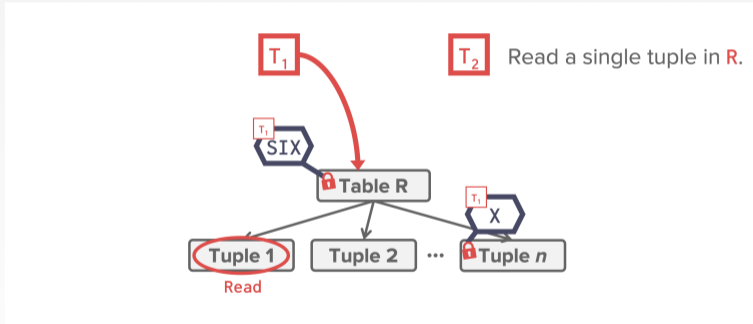


Example – Three Transactions

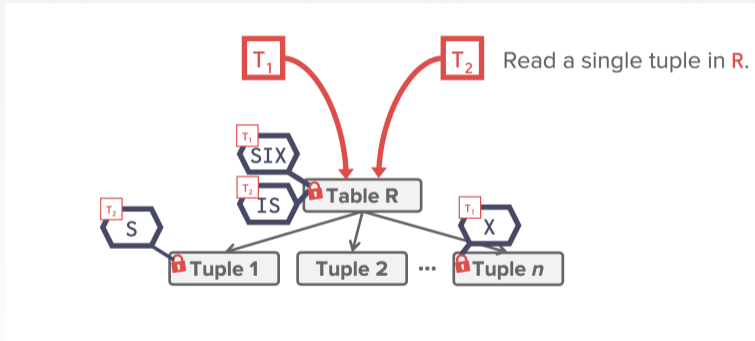
Scan R and update a few tuples.



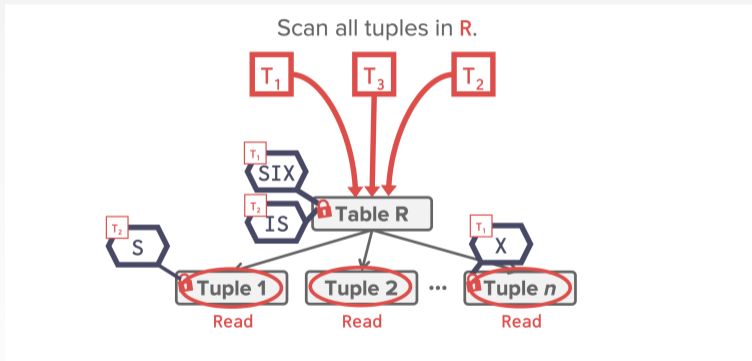
Example – Three Transactions



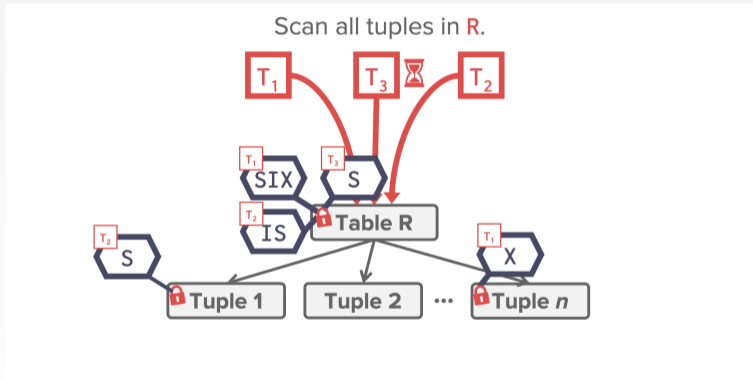
Example – Three Transactions



Example – Three Transactions



Example – Three Transactions



Multiple Lock Granularities

- Hierarchical locks are useful in practice as each txn only needs a few locks.
- Intention locks help improve concurrency:
 - ▶ **Intention-Shared** (IS): Intent to get S lock(s) at finer granularity.
 - ▶ **Intention-Exclusive** (IX): Intent to get X lock(s) at finer granularity.
 - ▶ **Shared+Intention-Exclusive** (SIX): Like S and IX at the same time.

Lock Escalation

- Lock escalation dynamically asks for coarser-grained locks when too many low level locks acquired.
- This reduces the number of requests that the lock manager has to process.

Locking in Practice

Locking in Practice

- You typically don't set locks manually in txns.
- Sometimes you will need to provide the DBMS with hints to help it to improve concurrency.
- Explicit locks are also useful when doing major changes to the database.

Lock Table

- Explicitly locks a table.
- Not part of the SQL standard.
 - ▶ Postgres/DB2/Oracle Modes: **SHARE**, **EXCLUSIVE**
 - ▶ MySQL Modes: **READ**, **WRITE**

 PostgreSQL

 ORACLE

 IBM DB2

 Microsoft SQL Server

 MySQL

```
LOCK TABLE <table> IN <mode> MODE;
```

```
SELECT 1 FROM <table> WITH (TABLOCK, <mode>);
```

```
LOCK TABLE <table> <mode>;
```

Select... For Update

- Perform a select and then sets an exclusive lock on the matching tuples.
- Can also set shared locks:
 - ▶ Postgres: **FOR SHARE**
 - ▶ MySQL: **LOCK IN SHARE MODE**

```
SELECT * FROM <table>  
WHERE <qualification> FOR UPDATE;
```

Conclusion

Parting Thoughts

- 2PL is used in almost all DBMSs.
- Automatically generates correct interleaving:
 - ▶ Locks + protocol (2PL, SS2PL ...)
 - ▶ Deadlock detection + handling
 - ▶ Deadlock prevention

Next Class

- Timestamp Ordering Concurrency Control