

# VanillaCore Walkthrough

## Part 6

Introduction to Databases

DataLab

CS, NTHU

# Outline

- Lock-Based Concurrency Control
  - 2PL
  - S2PL
  - Conservative Locking
- Code Tracing
  - S2PL in VanillaDB

# Outline

- Lock-Based Concurrency Control
  - 2PL
  - S2PL
  - Conservative Locking
- Code Tracing
  - S2PL in VanillaDB

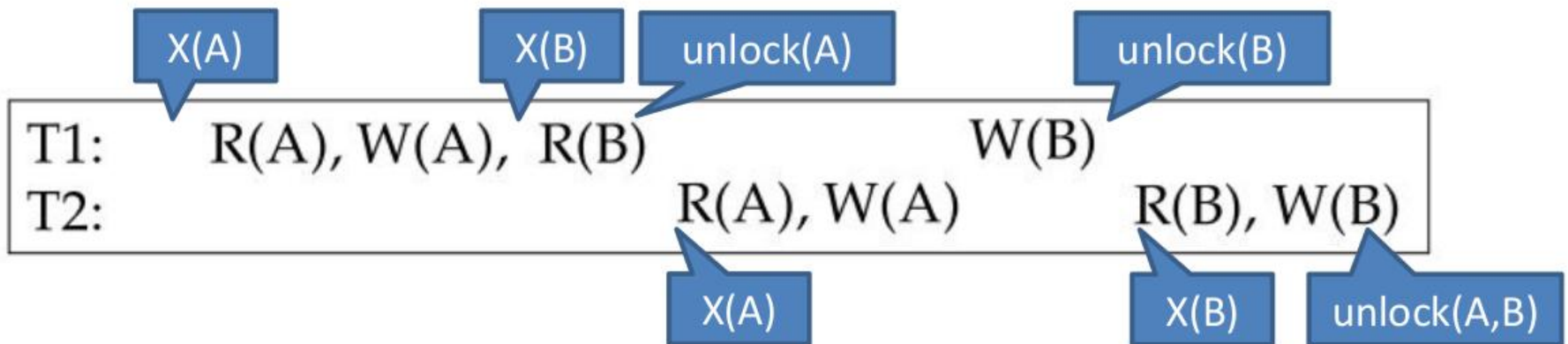
# Lock-Based Concurrency Control

- For isolation and consistency, a DBMS should only allow **serializable, recoverable** schedules
  - Uncommitted changes cannot be seen (no WR)
  - Ensure repeatable read (no RW)
  - Cannot overwrite uncommitted change (no WW)
- Locks are useful in this scenario

# 2 Phase Locking (2PL)

- 2 types of locks
  - Shared (S) lock
  - Exclusive (X) lock
- Phase 1: Growing Phase
  - Each tx must obtain an S (X) lock on an object before reading (writing) it
- Phase 2: Shrinking Phase
  - A transaction can not request additional locks once it releases any locks
- Ensures conflict serializability

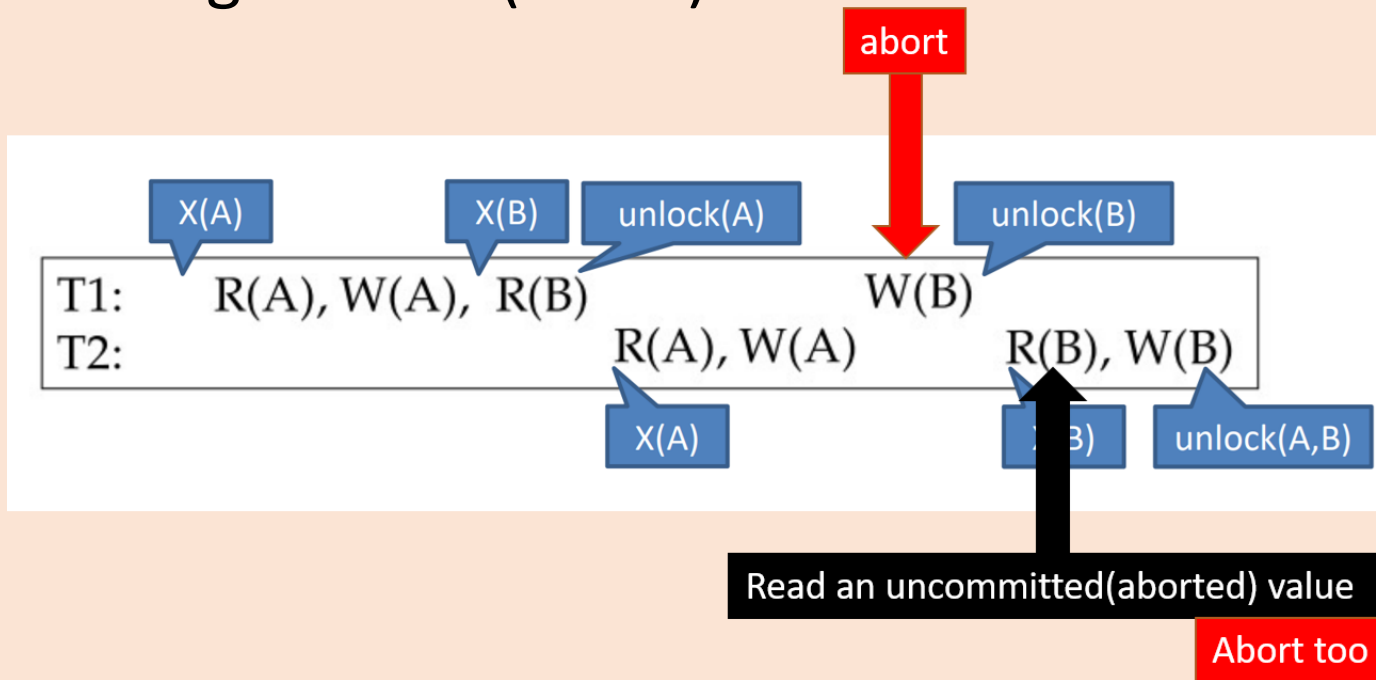
# Example



- Ensures conflict serializability

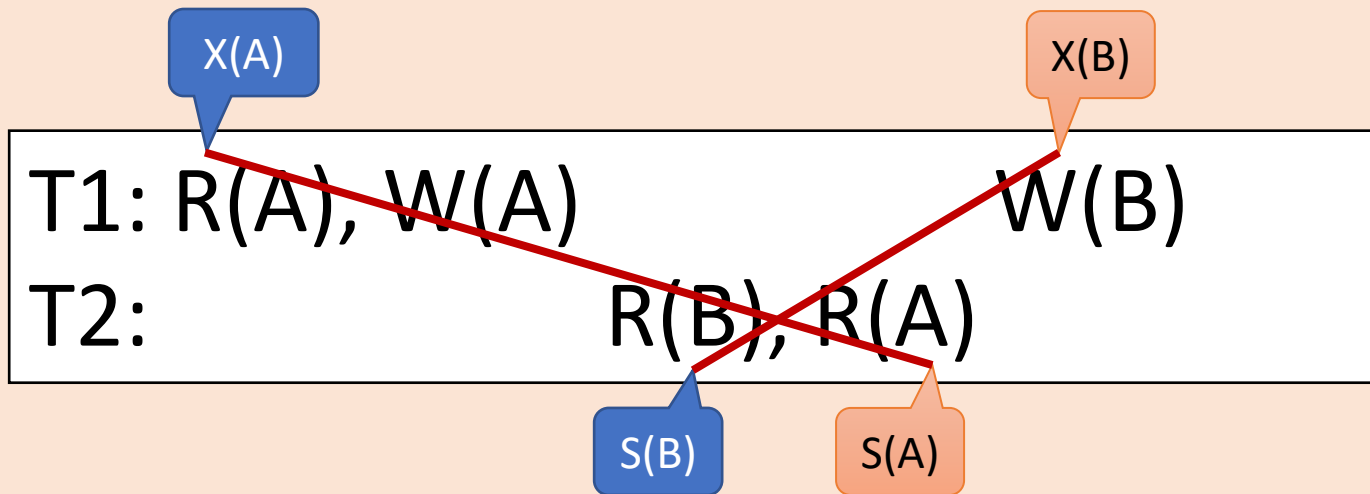
# Problems of 2PL

- Cascading rollback(abort)



# Problems of 2PL

- Deadlock





**Let's fix cascading rollback**

# Strict 2PL

1. Each tx obtains locks as in the growing phase in 2PL
2. But the tx holds all locks until it completes

# S2PL Example

## 2PL Cascading rollback

T1: R(A), W(A), R(B)

W(B)

T2: R(A), R(A) R(B), W(B)

## S2PL

T1: R(A), W(A), R(B), W(B)



Release X(A), X(B)

T2: R(A), R(A), R(B), W(B)

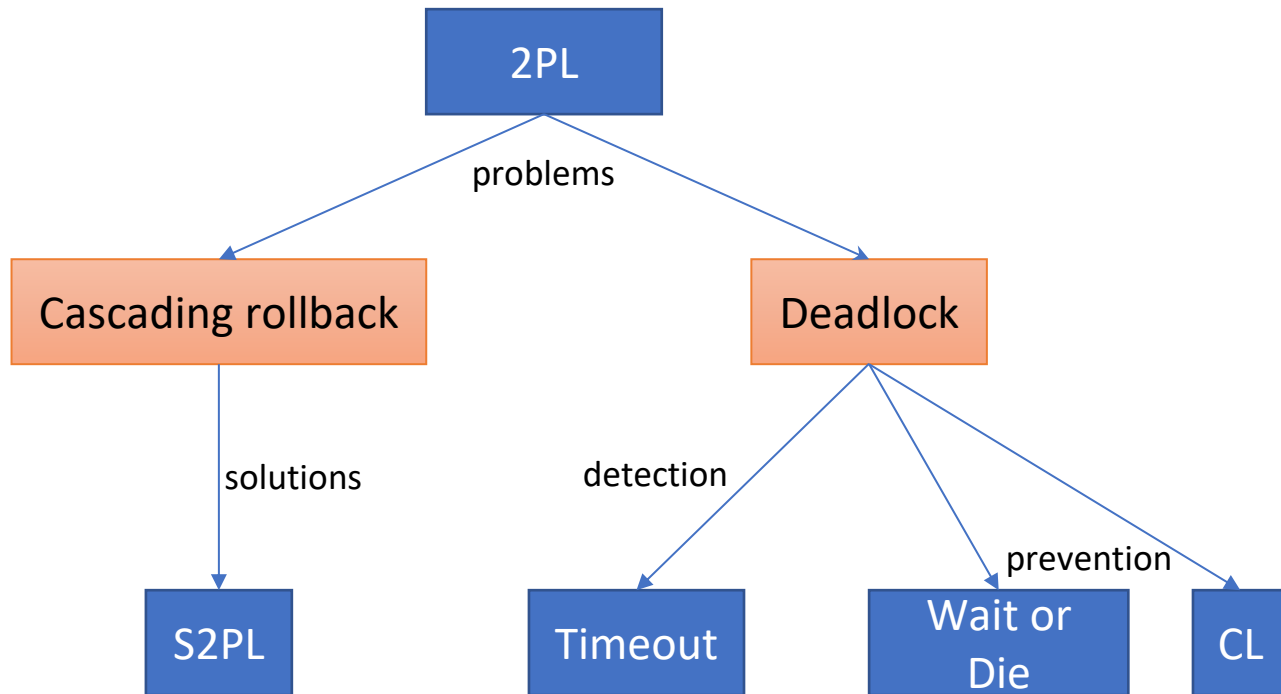
Do NOT release  
X(A)

**Let's fix deadlock**

# Deadlock detection & prevention

- Timeout (deadlock detection)
- Wait-Die (deadlock prevention)
  - Tx number as ages
    - Old man wait
    - Young men go die (abort)
- **Conservative locking** (deadlock prevention)
  - Locks all objects at once
  - However, we may not know which objects to lock
    - Stored procedure 
    - we've known the read/write set
    - Ad-hoc queries 

# Summary of lock-based CC

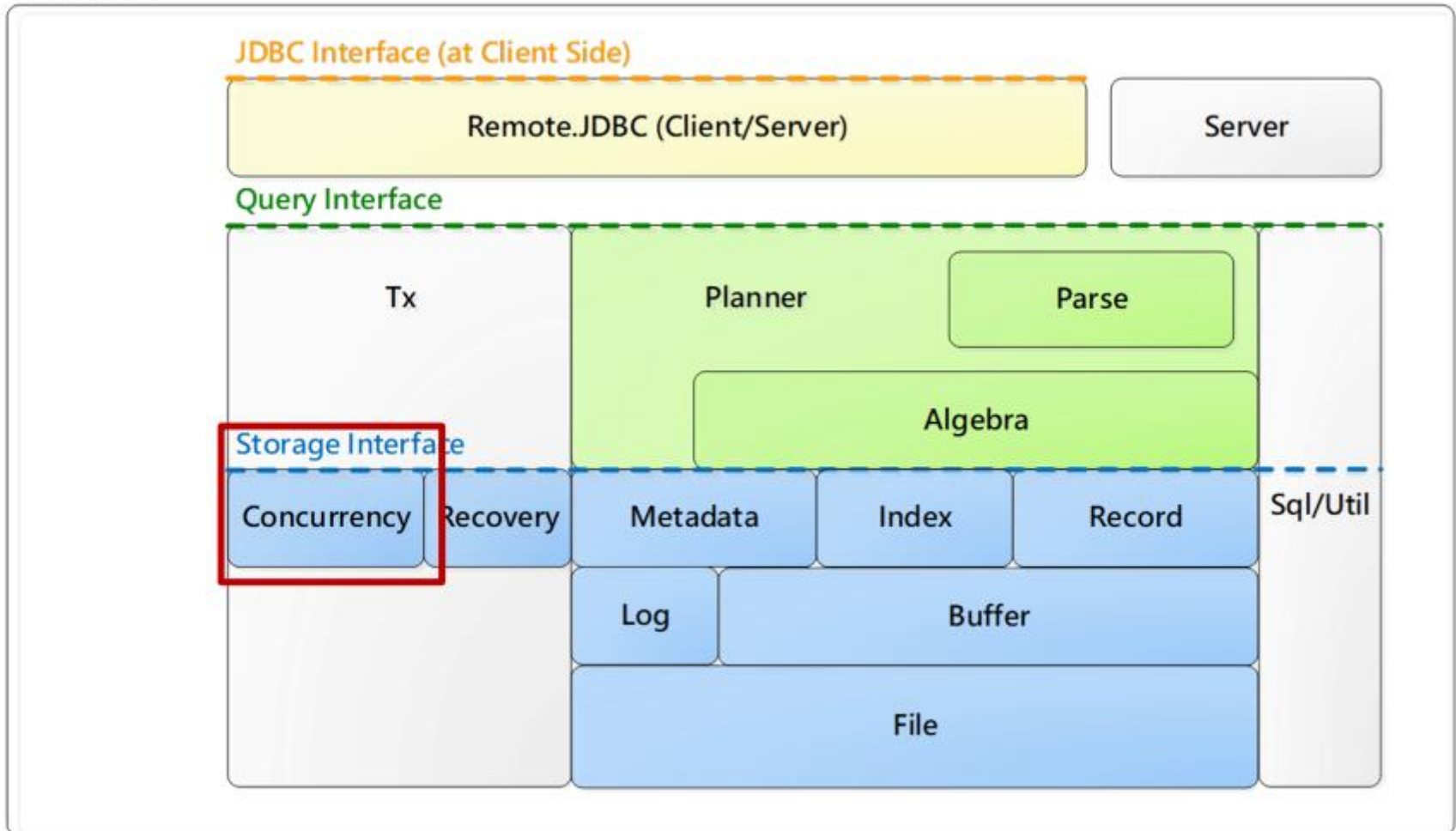


# Outline

- Lock-Based Concurrency Control
  - 2PL
  - S2PL
  - Conservative Locking
- Code Tracing
  - S2PL in VanillaDB

# Code Tracing

VanillaCore





# Package Structure



tx package

Concurrency package

implement

TransactionLifecycleListener

ConcurrencyMgr

has a

LockTable

extends

ReadCommittedConcurrencyMgr

RepeatableReadConcurrencyMgr

SerializableConcurrencyMgr

throw

LockAbortException

- org.vanilladb.core.storage.tx
  - Transaction.java
  - TransactionLifecycleListener.java
  - TransactionMgr.java

- org.vanilladb.core.storage.tx.concurrency
  - ConcurrencyMgr.java
  - LockAbortException.java
  - LockTable.java
  - ReadCommittedConcurrencyMgr.java
  - RepeatableReadConcurrencyMgr.java
  - SerializableConcurrencyMgr.java

# Isolation Levels

Isolation level	Dirty reads	Unrepeatable reads	Phantoms
Read Uncommitted	Maybe	Maybe	Maybe
Read Committed	No	Maybe	Maybe
Repeatable Read	No	No	Maybe
Serializable	No	No	No

Isolation level	Shared Lock	Predicate Lock
Read Uncommitted	No	No
Read Committed	Released early	No
Repeatable Read	Held to completion	No
Serializable	Held to completion	Held to completion

# readBlock

ReadCommitted

```
@Override
public void readBlock(BlockId blk) {
    lockTbl.isLock(blk.fileName(), txNum);
    // releases IS lock to allow phantoms
    lockTbl.release(blk.fileName(), txNum, LockTable.IS_LOCK);

    lockTbl.sLock(blk, txNum);
    // releases S lock at the end of statement to allow unrepeatable
    Read
    toReleaseSLockAtEndStatement.add(blk);
}
```

RepeatableRead

```
@Override
public void readBlock(BlockId blk) {
    lockTbl.isLock(blk.fileName(), txNum);
    // releases IS lock to allow phantoms
    lockTbl.release(blk.fileName(), txNum, LockTable.IS_LOCK);
    lockTbl.sLock(blk, txNum);
}
```

Serializable

```
@Override
public void readBlock(BlockId blk) {
    lockTbl.isLock(blk.fileName(), txNum);
    lockTbl.sLock(blk, txNum);
}
```

# TransactionLifecycleListener

```
16 package org.vanilladb.core.storage.tx;
17
18 public interface TransactionLifecycleListener {
19
20     void onTxCommit(Transaction tx);
21
22     void onTxRollback(Transaction tx);
23
24     void onTxEndStatement(Transaction tx);
25
26 }
27
```

Release all locks on tx commit

```
public class SerializableConcurrencyMgr extends ConcurrencyMgr {
    public SerializableConcurrencyMgr(long txNumber) {
        txNum = txNumber;
    }
    @Override
    public void onTxCommit(Transaction tx) {
        lockTbl.releaseAll(txNum, false);
    }
    @Override
    public void onTxRollback(Transaction tx) {
        lockTbl.releaseAll(txNum, false);
    }
}
```

# Event-Driven Architecture

```
public Transaction(TransactionMgr txMgr, TransactionLifecycleListener concurMgr,
    TransactionLifecycleListener recoveryMgr, TransactionLifecycleListener bufferMgr, boolean readOnly,
    long txNum) {
    this.concurMgr = (ConcurrencyMgr) concurMgr;
    this.recoveryMgr = (RecoveryMgr) recoveryMgr;
    this.bufferMgr = (BufferMgr) bufferMgr;
    this.txNum = txNum;
    this.readOnly = readOnly;

    lifecycleListeners = new LinkedList<TransactionLifecycleListener>();
    addLifecycleListener(txMgr);
    addLifecycleListener(recoveryMgr);
    addLifecycleListener(concurMgr);
    addLifecycleListener(bufferMgr);
}

    public void commit() {
        for (TransactionLifecycleListener l : lifecycleListeners)
            l.onTxCommit(this);

        if (logger.isLoggable(Level.FINE))
            logger.fine("transaction " + txNum + " committed");
    }
```

# How to use ConcurrencyMgr?

- RecordPage

```
private Constant getVal(int offset, Type type) {  
    if (!isTempTable())  
        tx.concurrencyMgr().readRecord(new RecordId(blk, currentSlot));  
    return currentBuff.getVal(offset, type);  
}
```

- SerializableConcurrencyMgr

```
public void readRecord(RecordId recId) {  
    lockTbl.isLock(recId.block().fileName(), txNum);  
    lockTbl.isLock(recId.block(), txNum);  
    lockTbl.sLock(recId, txNum);  
}
```

