# CS150A Database

Wenjie Wang

School of Information Science and Technology

ShanghaiTech University

April. , 2024

Today:

- Transactions & Concurrency Control II:

Readings:

- Database Management Systems (DBMS), Chapters 16&17



*Acknowledgement: Joe Hellerstein@UCBerkeley's scourse notes*

# TWO PHASE LOCKING

# Two Phase Locking (2PL)

- **The most common scheme for enforcing conflict serializability**
- **A bit “pessimistic”**
  - Sets locks for fear of conflict… Some cost here.
  - Alternative schemes use multiple versions of data and “optimistically” let transactions move forward
    - Abort when conflicts are detected.
    - Some names to know/look up:
      - Optimistic Concurrency Control
      - Timestamp-Ordered Multiversion Concurrency Control
    - We will not study these schemes in this lecture

# Two Phase Locking (2PL), Part 2

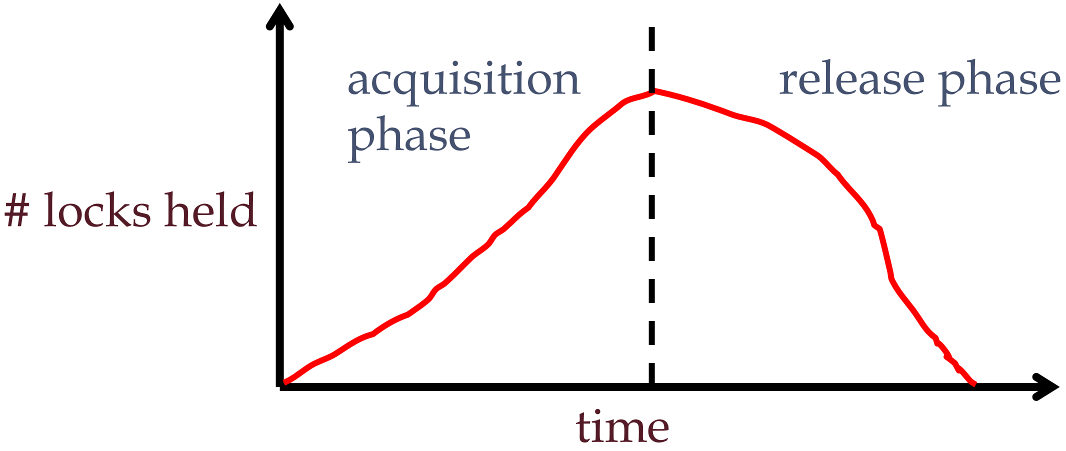
- Rules:
  - Xact must obtain a S (shared) lock before reading, and an X (exclusive) lock before writing.
  - **Xact cannot get new locks after releasing any locks**

Lock Compatibility Matrix

| | S | X |
|---|---|---|
| S | √ | – |
| X | – | – |

# Two Phase Locking (2PL), Part 3

- **2PL guarantees conflict serializability (why?)**
- But, does not prevent **cascading aborts**



# Why 2PL guarantees conflict serializability

- When a committing transaction has reached the end of its acquisition phase…
  - Call this the “lock point”
  - At this point, it has *everything it needs* locked…
  - … and any conflicting transactions either:
    - started release phase before this point
    - are blocked waiting for this transaction
- Visibility of actions of two conflicting transactions are ordered by their lock points
- The order of lock points gives us an equivalent serial schedule!

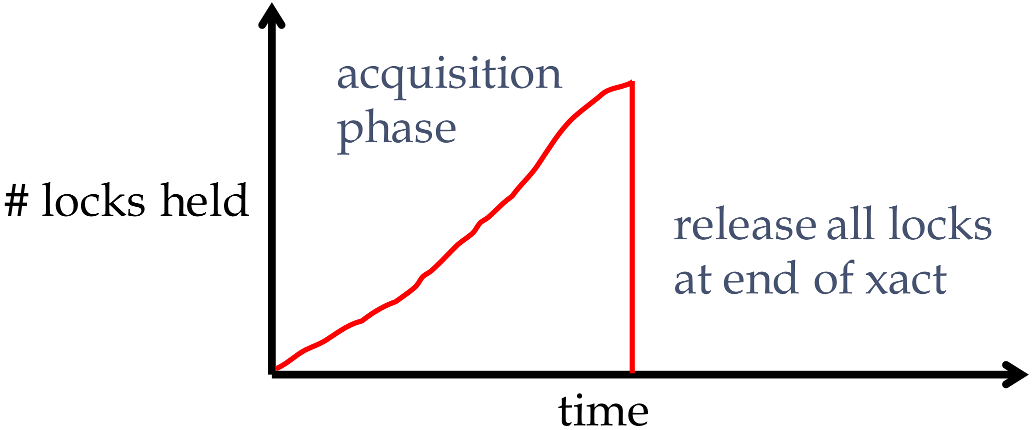
# Strict Two Phase Locking (2PL)

- **Problem: Cascading Aborts**
- Example: rollback of T1 requires rollback of T2!

| | | | |
|---|---|---|---|
| **T1:** | **R(A), W(A)** | | **Abort** |
| **T2:** | | **R(A), W(A)** | |

# Strict Two Phase Locking

- Same as 2PL, except all locks released together when transaction completes
  - (i.e.) either
    - Transaction has committed (all writes durable), OR
    - Transaction has aborted (all writes have been undone)



# Next ...

- A few examples

# Non-2PL, A = 1000, B = 2000, Output = ?

| T1 | T2 |
|---|---|
| Lock_X(A) | |
| Read(A) | |
| | Lock_S(A) |
| A: = A-50 | |
| Write(A) | |
| Unlock(A) | |
| | Read(A) |
| | Unlock(A) |
| | Lock_S(B) |
| Lock_X(B) | |
| | Read(B) |
| | Unlock(B) |
| | PRINT(A), PRINT(B), PRINT(A+B) |
| Read(B) | |
| B := B +50 | |
| Write(B) | |
| Unlock(B) | |

Output: 950, 2000, 2950

# Non-2PL, A = 1000, B = 2000, Output = ? cont

| T1 | T2 |
| --- | --- |
| Lock_X(A) | |
| Read(A): (A=1000) | |
| | Lock_S(A) |
| A: = A-50 (A=950) | |
| Write(A) A=950 | |
| Unlock(A) | |
| | Read(A) (A = 950) |
| | Unlock(A) |
| | Lock_S(B) |
| Lock_X(B) | |
| | Read(B) (B=2000) |
| | Unlock(B) |
| | PRINT(A), PRINT(B), PRINT(A+B) |
| Read(B) (B=2000) | |
| B := B +50 (B=2050) | |
| Write(B) B=2050 | |
| Unlock(B) | |

Output: 950, 2000, 2950

# 2PL, A = 1000, B = 2000, Output = ?

| T1 | T2 |
|---|---|
| Lock_X(A) | |
| Read(A) | |
| A: = A-50 | |
| Write(A) | |
| Lock_X(B) | |
| Unlock(A) | |
| | Lock_S(A) |
| | Read(A) |
| Read(B) | |
| B := B +50 | |
| Write(B) | |
| Unlock(B) | |
| | Lock_S(B) |
| | Unlock(A) |
| | Read(B) |
| | Unlock(B) |
| | PRINT(A), PRINT(B), PRINT(A+B) |

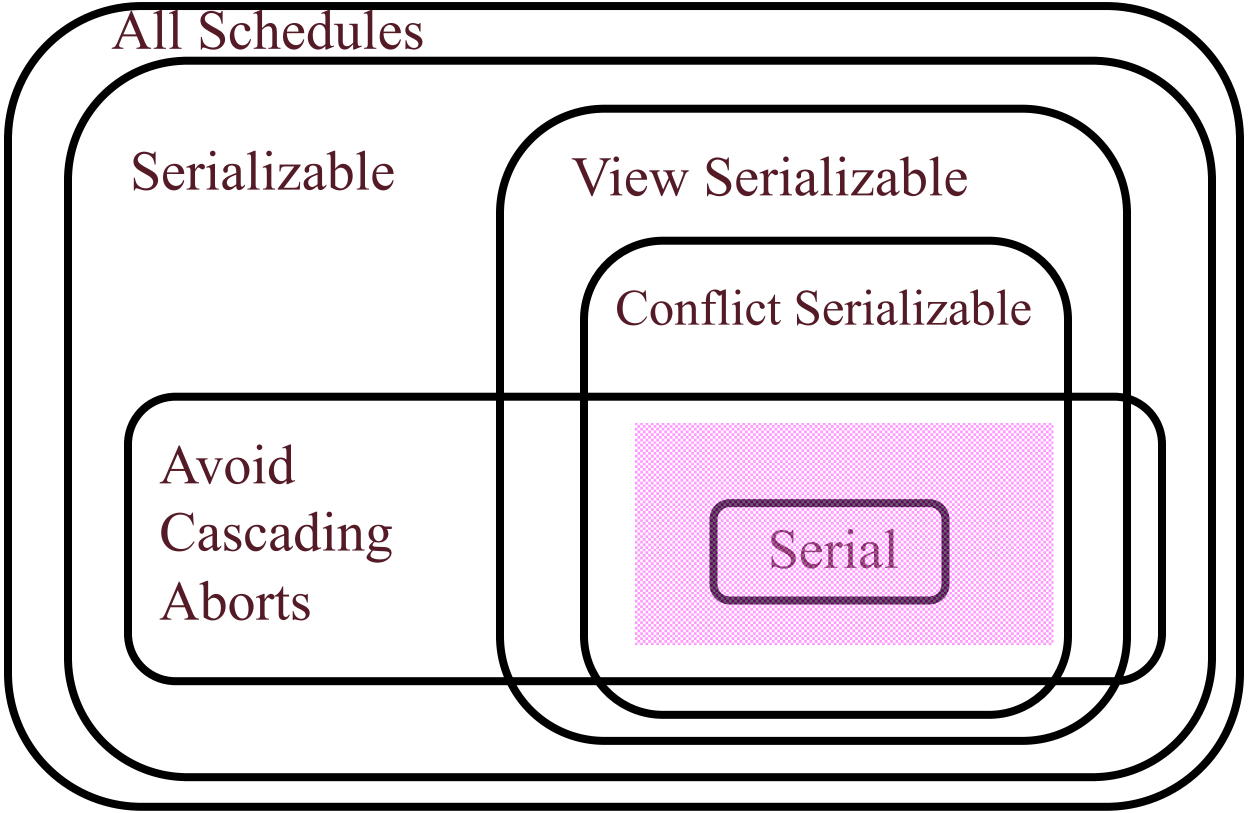
Output: 950, 2050, 3000

# Strict 2PL, A = 1000, B = 2000, Output = ?

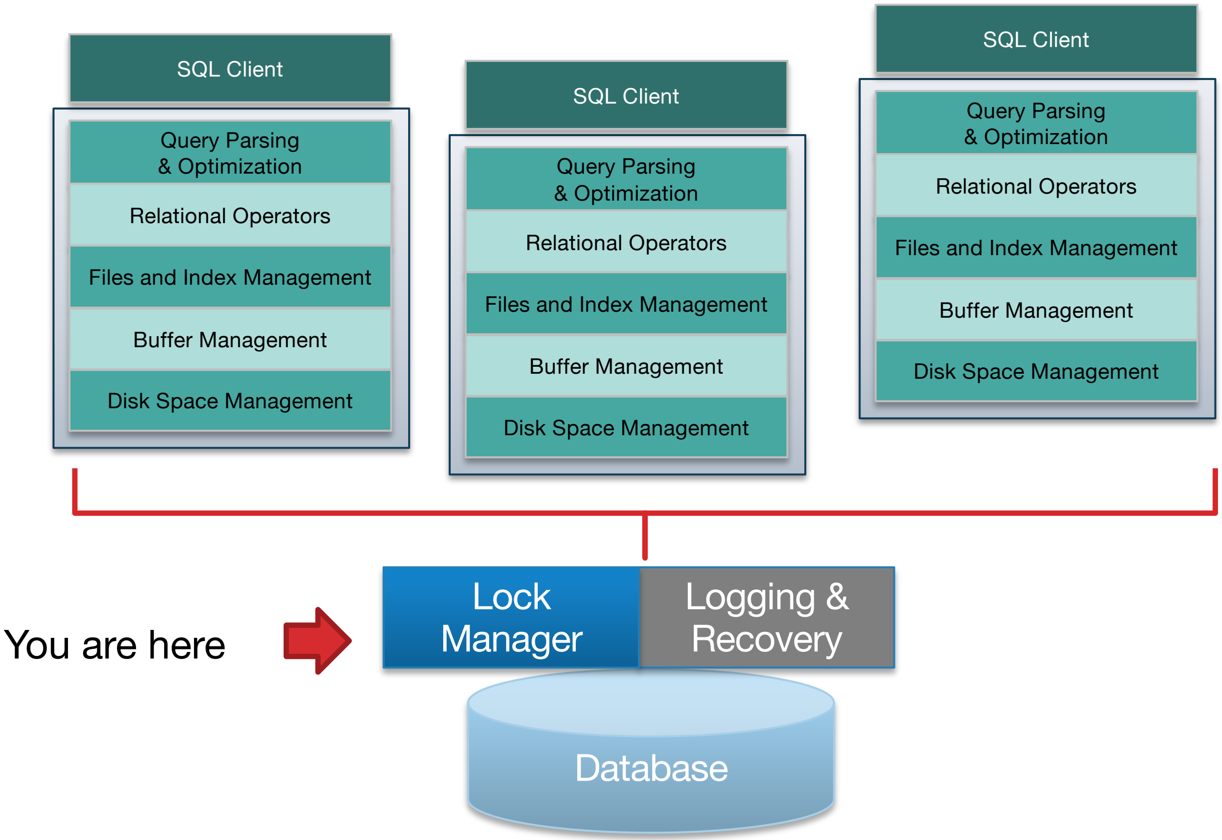
| T1 | T2 |
|---|---|
| Lock_X(A) | |
| Read(A) | |
| | Lock_S(A) |
| A: = A-50 | |
| Write(A) | |
| Lock_X(B) | |
| Read(B) | |
| B := B +50 | |
| Write(B) | |
| Unlock(A) | |
| Unlock(B) | |
| | Read(A) |
| | Lock_S(B) |
| | Read(B) |
| | PRINT(A), PRINT(B), PRINT(A+B) |
| | Unlock(A) |
| | Unlock(B) |

Output: 950, 2050, 3000

# Which schedules does Strict 2PL allow?



# Architecture



# How Do We Lock Data?


- Not by any crypto or hardware enforcement
  - There are no adversaries here … this is all within the DBMS

- We lock by simple convention:
  - Within DBMS internals, we observe a lock *protocol*
  - If your transaction *holds* a lock, and my transaction *requests* a conflicting lock, then I am queued up waiting for that lock.

# Lock Management

- Lock and unlock requests handled by Lock Manager
- LM maintains a hashtable, keyed on names of objects being locked.
- LM keeps an entry for each currently held lock
- Entry contains
  - Granted set: Set of xacts currently granted access to the lock
  - Lock mode: Type of lock held (shared or exclusive)
  - Wait Queue: Queue of lock requests

| | Granted Set | Mode | Wait Queue |
|---|---|---|---|
| **A** | {T1, T2} | S | T3(X) ← T4(X) |
| **B** | {T6} | X | T5(X) ← T7(S) |

# Lock Management (continued)

- **When lock request arrives:**
  - Does any xact in Granted Set or Wait Queue want a conflicting lock?
    - If no, put the requester into “granted set” and let them proceed
    - If yes, put requester into wait queue (typically FIFO)

- **Lock upgrade:**
  - Xact with shared lock can request to upgrade to exclusive

| | Granted Set | Mode | Wait Queue |
|---|---|---|---|
| **A** | {T1, T2} | S | **T2(X)** ← T3(X) ← T4(X) |
| **B** | {T6} | X | T5(X) ← T7(S) |

# Example

| | |
|---|---|
| Lock_X(A) | |
| | Lock_S(B) |
| | Read(B) |
| | Lock_S(A) |
| Read(A) | |
| A: = A-50 | |
| Write(A) | |
| Lock_X(B) | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

Final lock table state:

A:
X lock held by T1
wait queue = [ T2 wants S ]

B:
S lock held by T2
wait queue = [ T1 wants X ]

Uh-oh, T1 and T2 are waiting for each other!

# DEADLOCK

# Deadlocks, cont

- **Deadlock: Cycle of Xacts waiting for locks to be released by each other.**
- **Three ways of dealing with deadlocks:**
  - Prevention
  - Avoidance
  - Detection and Resolution
- **Many systems just punt and use timeouts**
  - What are the dangers with this approach?

# Deadlock Scenarios

- They can just happen (unavoidable)

| | Granted Set | Mode | Wait Queue |
|---|---|---|---|
| **A** | {T1} | S | T2(X) |
| **B** | {T2} | X | T1(S) |

- Bad implementation of Lock Upgrade (avoidable! prioritize upgrades)

| | Granted Set | Mode | Wait Queue |
|---|---|---|---|
| **A** | {T1, T2} | S | T3(X) ← T4(X) ←T2(X) |

- Multiple Lock Upgrades (unavoidable)

| | Granted Set | Mode | Wait Queue |
|---|---|---|---|
| **A** | {T1, T2} | S | **T2(X) ← T1(X) ←** T3(X) ← T4(X) |

# Deadlock Scenarios

- They can just happen (unavoidable)

| | Granted Set | Mode | Wait Queue |
|---|---|---|---|
| A | {T1} | S | T2(X) |
| B | {T2} | X | T1(S) |

- Bad implementation of Lock Upgrade (avoidable! prioritize upgrades)

| | Granted Set | Mode | Wait Queue |
|---|---|---|---|
| A | {T1, T2} | S | T2(X) ← T3(X) ← T4(X) |

- Multiple Lock Upgrades (unavoidable)

| | Granted Set | Mode | Wait Queue |
|---|---|---|---|
| A | {T1, T2} | S | **T2(X) ← T1(X) ←** T3(X) ← T4(X) |

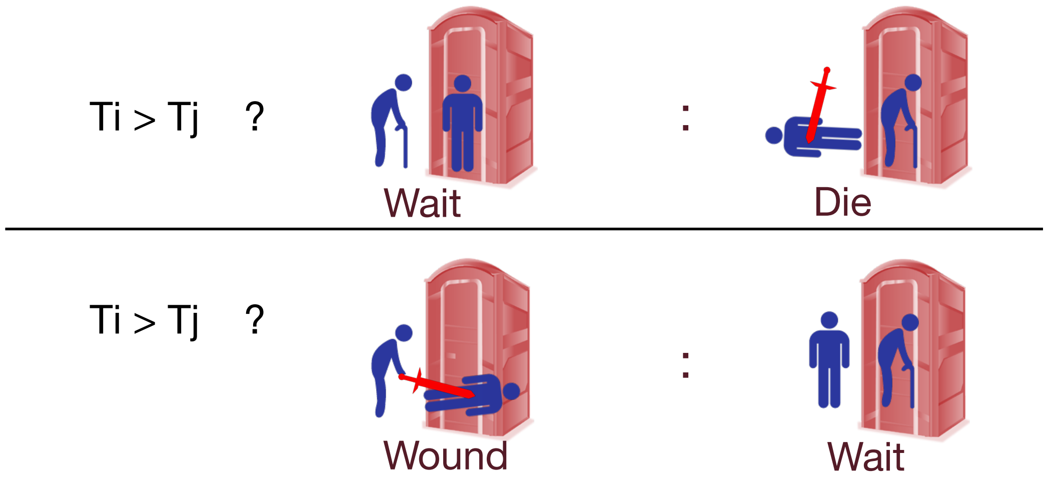
# Deadlock Prevention

- **Common technique in operating systems**
- **Standard approach: resource ordering**
  - Screen < Network Card < Printer
- **Why is this problematic for Xacts in a DBMS?**
  - What order would you impose?

# Deadlock Avoidance

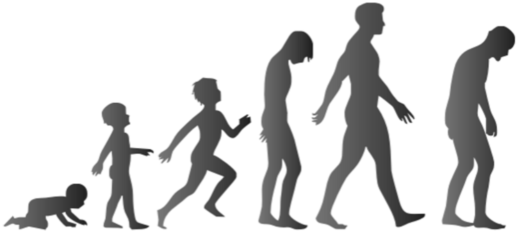
- **Assign priorities based on age: (now – start_time).**
- Say Ti wants a lock that Tj holds. Two possible policies:
  - **Wait-Die:** If Ti has higher priority, Ti waits for Tj; else Ti aborts
  - **Wound-Wait:** If Ti has higher priority, Tj aborts; else Ti waits

  - Read each of these like a ternary operator (C/C++/java/javascript)



# Deadlock Avoidance: Analysis

- Q: Why do these schemes guarantee no deadlocks?
  - Q: What do the previous images have in common?
- Important Detail: If a transaction re-starts, make sure it gets its original timestamp. Why?
- Note: other priority schemes make sense
  - E.g. measures of resource consumption, like #locks acquired



# Deadlock Detection

- Create and maintain a **“waits-for” graph**
- Periodically check for cycles in a graph

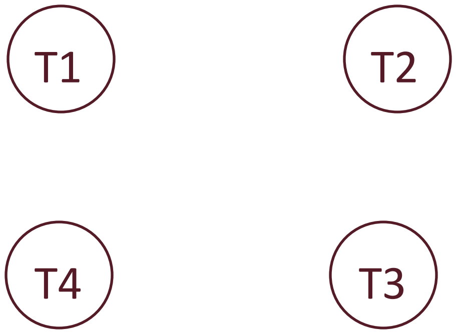
# Deadlock Detection, Part 2

**Example:**

**T1:**
**T2:**

**T3:**

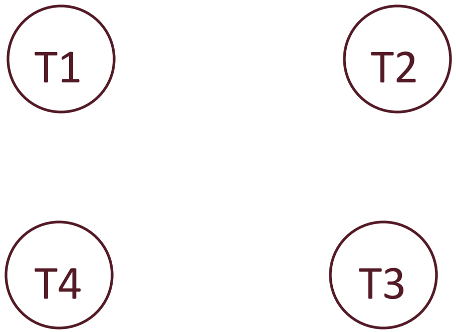
**T4:**



# Deadlock Detection, Part 3

**Example:**

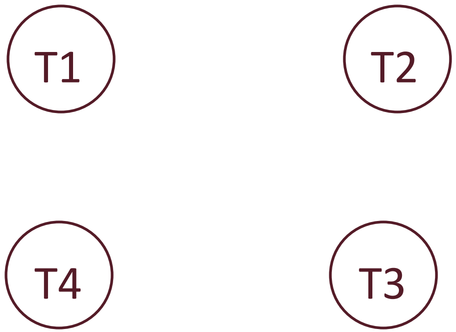
**T1: S(A)**
**T2:**
**T3:**
**T4:**



# Deadlock Detection, Part 4

**Example:**

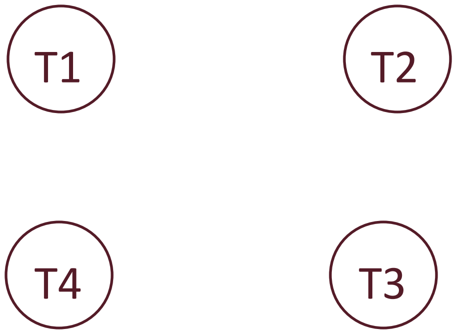
**T1: S(A) S(D)**
**T2:**
**T3:**
**T4:**



# Deadlock Detection, Part 5

**Example:**

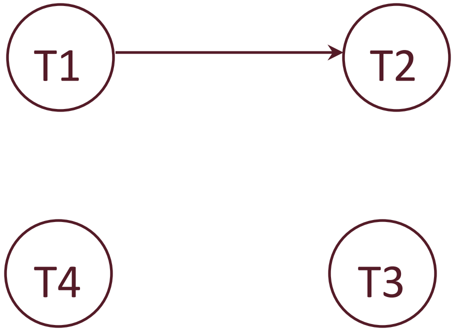
**T1: S(A) S(D)**
**T2: X(B)**
**T3:**
**T4:**



# Deadlock Detection, Part 6

**Example:**

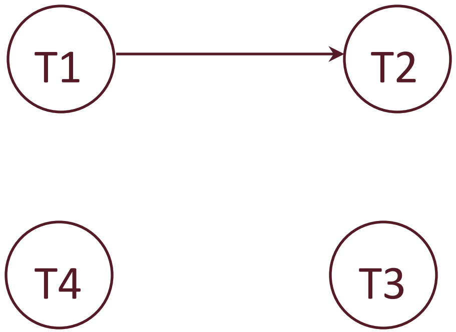
**T1: S(A) S(D) S(B)**
**T2: X(B)**
**T3:**
**T4:**



# Deadlock Detection, Part 7

**Example:**

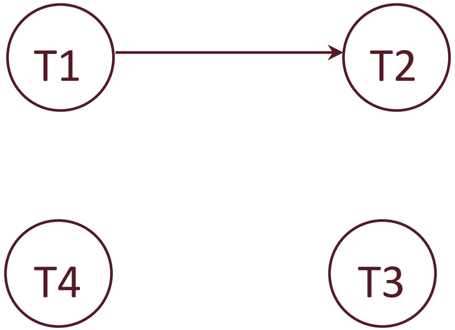
| | | | | | |
|---|---|---|---|---|---|
| **T1:** | **S(A)** | **S(D)** | | **S(B)** | |
| **T2:** | | | **X(B)** | | |
| **T3:** | | | | | **S(D)** |
| **T4:** | | | | | |



# Deadlock Detection, Part 8

**Example:**

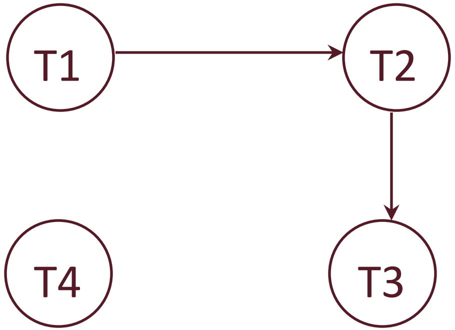
**T1: S(A) S(D) S(B)**
**T2: X(B)**
**T3: S(D), S(C)**
**T4:**



# Deadlock Detection, Part 9

**Example:**

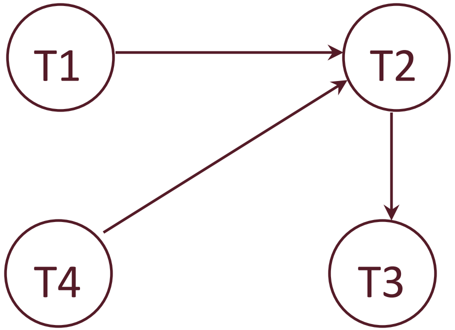
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **T1:** | **S(A)** | **S(D)** | | **S(B)** | | | |
| **T2:** | | | **X(B)** | | | | **X(C)** |
| **T3:** | | | | | **S(D)** | **S(C)** | |
| **T4:** | | | | | | | |



# Deadlock Detection, Part 10

**Example:**

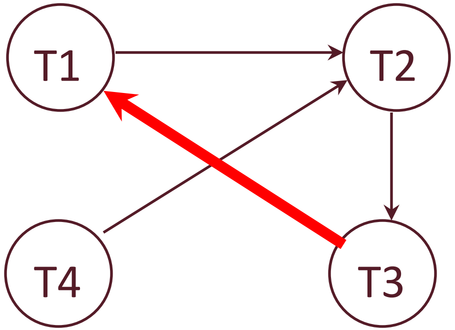
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **T1:** | **S(A)** | **S(D)** | | **S(B)** | | | | |
| **T2:** | | | **X(B)** | | | | **X(C)** | |
| **T3:** | | | | | **S(D)** | **S(C)** | | |
| **T4:** | | | | | | | | **X(B)** |



# Deadlock Detection, Part 11

**Example:**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **T1:** | **S(A)** | **S(D)** | | **S(B)** | | | | | |
| **T2:** | | | **X(B)** | | | | **X(C)** | | |
| **T3:** | | | | | **S(D)** | **S(C)** | | | **X(A)** |
| **T4:** | | | | | | | | **X(B)** | |



# Deadlock!

- **T1, T2, T3 are deadlocked**
  - Doing no good, and holding locks
- **T4 still cruising**
- **In the background, run a deadlock detection algorithm**
  - Periodically extract the waits-for graph
  - Find cycles
  - “Shoot” a transaction on the cycle
- **Empirical fact**
  - Most deadlock cycles are small (2-3 transactions)

# LOCK GRANULARITY

# Lock Granularity, cont

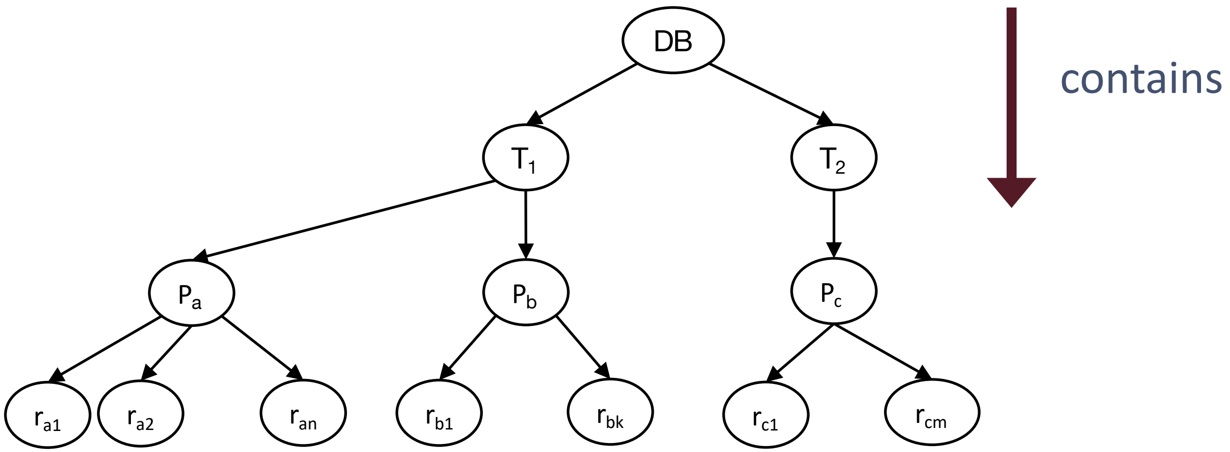
- Hard to decide what granularity to lock
  - Tuples vs Pages vs Tables?
- What is the tradeoff?
  - Fine-grained availability of resources would be nice (e.g. lock per tuple)
  - Small # of locks to manage would also be nice (e.g. lock per table)
  - Can’t have both!
    - Or can we???

# Multiple Locking Granularity

- **Shouldn’t have to make same decision for all transactions!**
- Allow data items to be of various sizes
- Define a hierarchy of data granularities, small nested within large
    - Can be represented graphically as a tree.

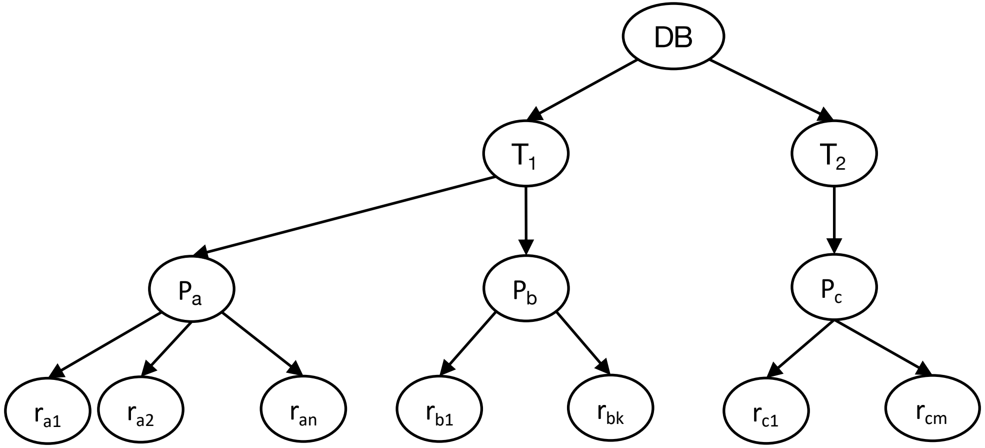
# Example of Granularity Hierarchy (RDBMS)

- Data “containers” can be viewed as nested.
- The levels, starting from the coarsest (top) level are
  - Database, Tables, Pages, Records
- When a transaction locks a node in the tree ***explicitly***, it ***implicitly*** locks all the node’s descendants in the same mode.



# Multiple Locking Granularity

- Granularity of locking (level in tree where locking is done):
  - **Fine granularity** (lower in tree): High concurrency, lots of locks (high overhead)
  - **Coarse granularity** (higher in tree): Few locks (low overhead), lost concurrency
    - Lost potential concurrency if you don’t need everything inside the coarse grain



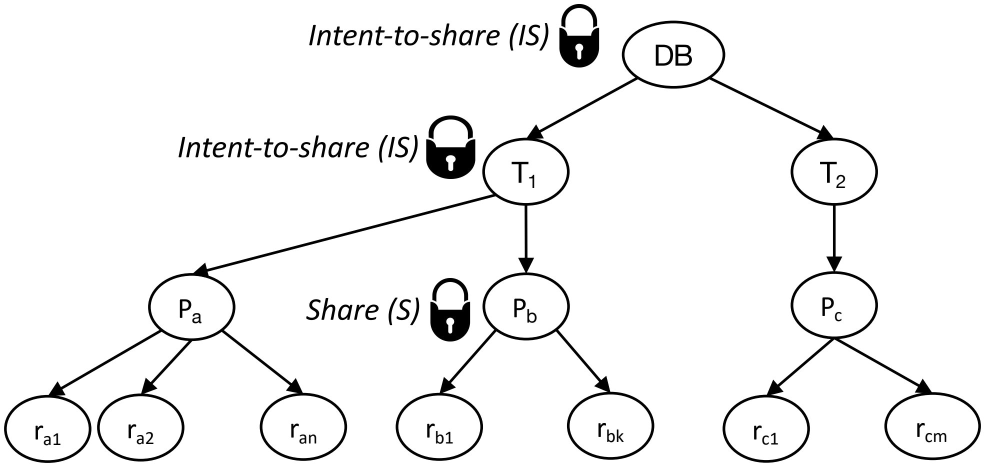
# Real-World Locking Granularities

| Resource | Description |
|---|---|
| RID | A row identifier used to lock a single row within a heap. |
| KEY | A row lock within an index used to protect key ranges in serializable transactions. |
| PAGE | An 8-kilobyte (KB) page in a database, such as data or index pages. |
| EXTENT | A contiguous group of eight pages, such as data or index pages. |
| HoBT | A heap or B-tree. A lock protecting a B-tree (index) or the heap data pages in a table that does not have a clustered index. |
| TABLE | The entire table, including all data and indexes. |
| FILE | A database file. |
| APPLICATION | An application-specified resource. |
| METADATA | Metadata locks. |
| ALLOCATION_UNIT | An allocation unit. |
| DATABASE | The entire database. |

From MS SQL Server
https://technet.microsoft.com/en-us/library/jj856598(v=sql.110).aspx

# Solution: New Lock Modes, Protocol

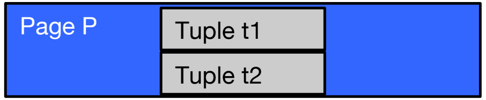
- Allow xacts to lock at each level, but with a special protocol using new “intent” locks:
- Before getting S or X lock, Xact must have proper intent locks on all its ancestors in the granularity hierarchy.



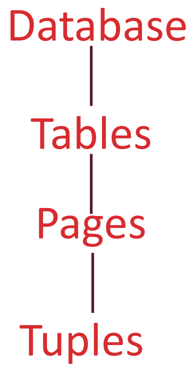
# New Lock Modes – Intention Lock Modes

- 3 additional lock modes:
  - ***IS:*** *Intent to get S lock(s) at finer granularity.*
  - ***IX:*** *Intent to get X lock(s) at finer granularity.*
  - ***SIX:*** *Like S & IX at the same time. Why useful?*

- Intention locks allow a higher level node to be locked in S or X mode without having to check all descendent nodes



# Multiple Granularity Locking Protocol



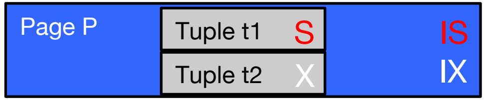

- Each Xact starts from the root of the hierarchy.
- To get S or IS lock on a node, must hold IS or IX on parent node.
  - What if Xact holds S on parent? SIX on parent?
- To get X or IX or SIX on a node, must hold IX or SIX on parent node.
- Must release locks in bottom-up order.

- 2-phase and lock compatibility matrix rules enforced as well
- Protocol is correct in that it is *equivalent to directly setting locks at leaf levels of the hierarchy*.

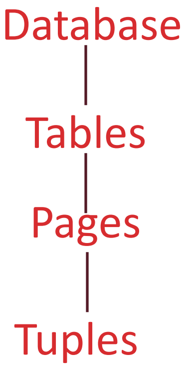
# Lock Compatibility Matrix

- IS – Intent to get S lock(s) at finer granularity.
- IX – Intent to get X lock(s) at finer granularity.
- SIX mode: Like S & IX at the same time.

Handy simple case to remember:
Could 2 intent locks be compatible?




| | IS | IX | S | SIX | X |
|---|---|---|---|---|---|
| IS | | | | | |
| IX | | | | | |
| S | | | true | | false |
| SIX | | | | | |
| X | | | false | | false |



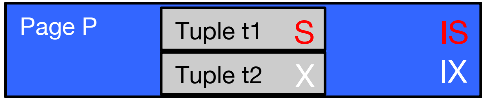
# Lock Compatibility Matrix, Cont

- IS – Intent to get S lock(s) at finer granularity.
- IX – Intent to get X lock(s) at finer granularity.
- SIX mode: Like S & IX at the same time.

| | IS | IX | S | SIX | X |
|---|---|---|---|---|---|
| **IS** | true | true | true | true | false |
| **IX** | true | true | false | false | false |
| **S** | true | false | true | false | false |
| **SIX** | true | false | false | false | false |
| **X** | false | false | false | false | false |

Database
Tables
Pages
Tuples

Handy simple case to remember:
Could 2 intent locks be compatible?



# Real-World Lock Compatibility Matrix

| | NL | SCH-S | SCH-M | S | U | X | IS | IU | IX | SIU | SIX | UIX | BU | RS-S | RS-U | RI-N | RI-S | RI-U | RI-X | RX-S | RX-U | RX-X |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NL | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N | N |
| SCH-S | N | N | C | N | N | N | N | N | N | N | N | N | N | I | I | I | I | I | I | I | I | I |
| SCH-M | N | C | C | C | C | C | C | C | C | C | C | C | C | I | I | I | I | I | I | I | I | I |
| S | N | N | C | N | N | C | N | N | C | N | C | C | C | N | N | N | N | N | C | N | N | C |
| U | N | N | C | N | C | C | N | C | C | C | C | C | C | N | C | N | N | C | C | N | C | C |
| X | N | N | C | C | C | C | C | C | C | C | C | C | C | C | C | N | C | C | C | C | C | C |
| IS | N | N | C | N | N | C | N | N | N | N | N | N | C | I | I | I | I | I | I | I | I | I |
| IU | N | N | C | N | C | C | N | N | N | N | N | C | C | I | I | I | I | I | I | I | I | I |
| IX | N | N | C | C | C | C | N | N | N | C | C | C | C | I | I | I | I | I | I | I | I | I |
| SIU | N | N | C | N | C | C | N | N | C | N | C | C | C | I | I | I | I | I | I | I | I | I |
| SIX | N | N | C | C | C | C | N | N | C | C | C | C | C | I | I | I | I | I | I | I | I | I |
| UIX | N | N | C | C | C | C | N | C | C | C | C | C | C | I | I | I | I | I | I | I | I | I |
| BU | N | N | C | C | C | C | C | C | C | C | C | C | N | I | I | I | I | I | I | I | I | I |
| RS-S | N | I | I | N | N | C | I | I | I | I | I | I | I | N | N | C | C | C | C | C | C | C |
| RS-U | N | I | I | N | C | C | I | I | I | I | I | I | I | N | C | C | C | C | C | C | C | C |
| RI-N | N | I | I | N | N | N | I | I | I | I | I | I | I | C | C | N | N | N | N | C | C | C |
| RI-S | N | I | I | N | N | C | I | I | I | I | I | I | I | C | C | N | N | N | C | C | C | C |
| RI-U | N | I | I | N | C | C | I | I | I | I | I | I | I | C | C | N | N | C | C | C | C | C |
| RI-X | N | I | I | C | C | C | I | I | I | I | I | I | I | C | C | N | C | C | C | C | C | C |
| RX-S | N | I | I | N | N | C | I | I | I | I | I | I | I | C | C | C | C | C | C | C | C | C |
| RX-U | N | I | I | N | C | C | I | I | I | I | I | I | I | C | C | C | C | C | C | C | C | C |
| RX-X | N | I | I | C | C | C | I | I | I | I | I | I | I | C | C | C | C | C | C | C | C | C |

**Key**

| | | | |
|---|---|---|---|
| N | No Conflict | SIU | Share with Intent Update |
| I | Illegal | SIX | Shared with Intent Exclusive |
| C | Conflict | UIX | Update with Intent Exclusive |
| | | BU | Bulk Update |
| NL | No Lock | RS-S | Shared Range-Shared |
| SCH-S | Schema Stability Locks | RS-U | Shared Range-Update |
| SCH-M | Schema Modification Locks | RI-N | Insert Range-Null |
| S | Shared | RI-S | Insert Range-Shared |
| U | Update | RI-U | Insert Range-Update |
| X | Exclusive | RI-X | Insert Range-Exclusive |
| IS | Intent Shared | RX-S | Exclusive Range-Shared |
| IU | Intent Update | RX-U | Exclusive Range-Update |
| IX | Intent Exclusive | RX-X | Exclusive Range-Exclusive |

From MS SQL Server
https://technet.microsoft.com/en-us/library/jj856598(v=sql.110).aspx

# For Your Information: Indexes

- **2PL on B+ tree pages is a rotten idea.**
  - Think about the first thing you would lock, and how that affects other xacts!
- **Instead, do short locks (latches) in a clever way**
  - Idea: Upper levels of B+ tree just need to direct traffic correctly. Don't need serializability or 2PL!
  - Different tricks to exploit this
    - The *B-link* tree is elegant
    - The *Bw-tree* is a recent variant for main memory DBs
- **Note: this is pretty complicated!**

# Summary, cont.

- **Correctness criterion for isolation is “serializability”.**
  - In practice, we use “conflict serializability” which is conservative but easy to enforce
- **Two Phase Locking and Strict 2PL: Locks implement the notions of conflict directly**
  - The lock manager keeps track of the locks issued.
  - **Deadlocks** may arise; can either be prevented or detected.
- **Multi-Granularity Locking:**
  - Allows flexible tradeoff between lock “scope” in DB, and # of lock entries in lock table
- **More to the story**
  - Optimistic/Multi-version/Timestamp CC
  - Index “latching”, phantoms
  - Actually, there’s much much more :-)