CSC 261/461 – Database Systems
Lecture 23

Spring 2017
MW 3:25 pm – 4:40 pm
January 18 – May 3
Dewey 1101

Announcements

- Term Paper due on April 20
- Project 1 Milestone 4 will be out tonight.
- The last (mini) project would be out before Wednesday.

Topics for Today

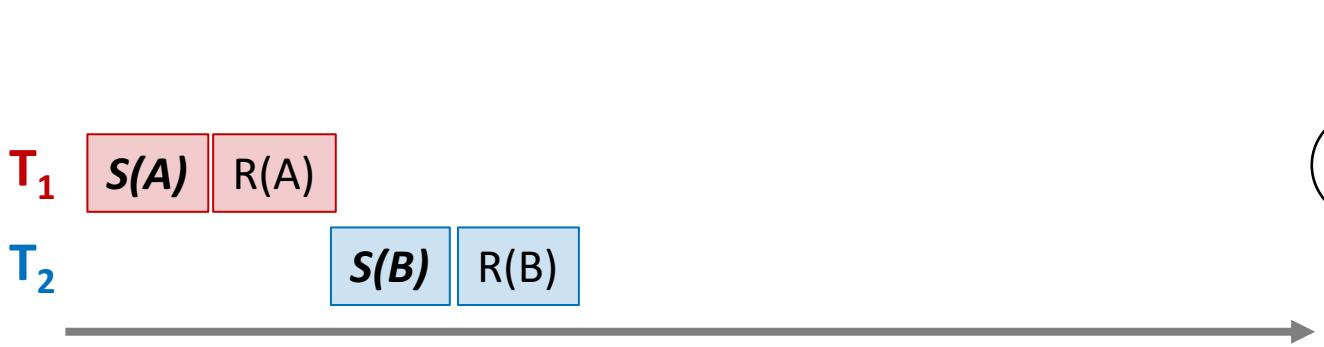
- Transactions (Deadlock and Recovery)
- Spark

DEADLOCK

Deadlock Detection: Example

First, T_1 requests a shared lock on A to read from it

Deadlock Detection: Example



Waits-for graph:

Next, T_2 requests a shared lock on B to read from it

Deadlock Detection: Example

T_2 then requests an exclusive lock on A to write to it- **now T_2 is waiting on T_1…**

Performance of Locking

- Resolve conflicts between transactions and use two basic mechanisms:
  - Blocking
  - Aborting
- Both incurs performance penalty.
  - Blocking: Other transactions need to wait)
  - Aborting: Wastes the work done thus far)

- Deadlock:
  - Extreme instance of blocking
  - A set of transactions are forever blocked unless one of the deadlocked transactions is aborted by the DBMS

Deadlocks

- **Deadlock**: Cycle of transactions waiting for locks to be released by each other.

- Two ways of dealing with deadlocks:

  1. Deadlock prevention

  2. Deadlock avoidance

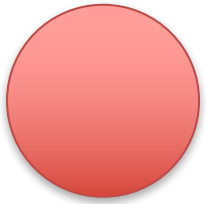
Deadlock Prevention

- Use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

- Wait-Die Scheme
- Wound-Wait Scheme

Timestamp Ordering

- Each **transaction** is **assigned** a ***unique*** **increasing timestamp**
- ***Earlier*** **transactions** receives a ***smaller*** **timestamp**
- T_1 (old) , T_2, T_3 (new), …
- Notation: Old Transaction T_{old} New Transaction T_{new}



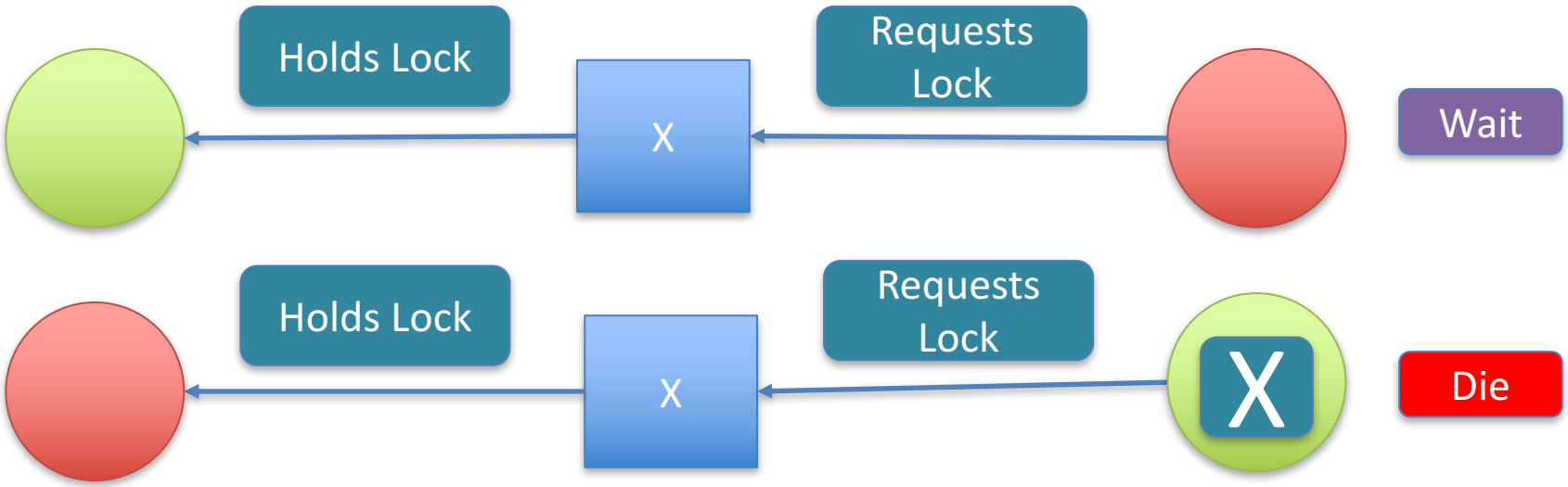
T_{old}

T_{new}

Wait-Die

T_{old} is allowed to ***wait*** for T_{new}

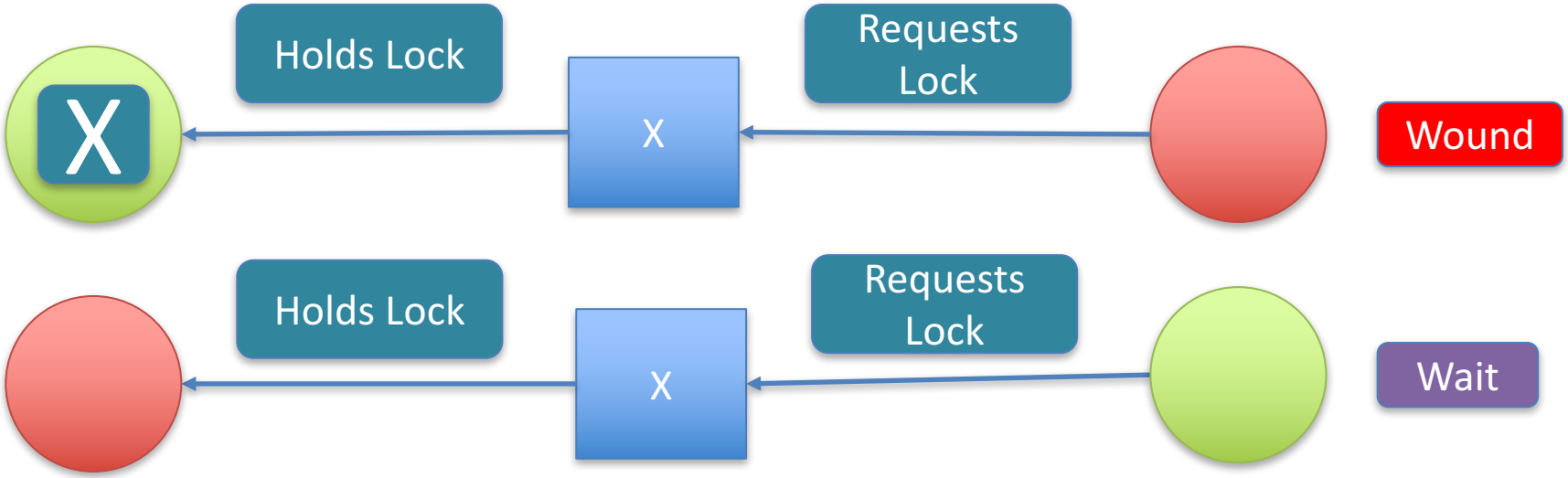
T_{new} will ***die*** when it waits for T_{old}



Wound Wait

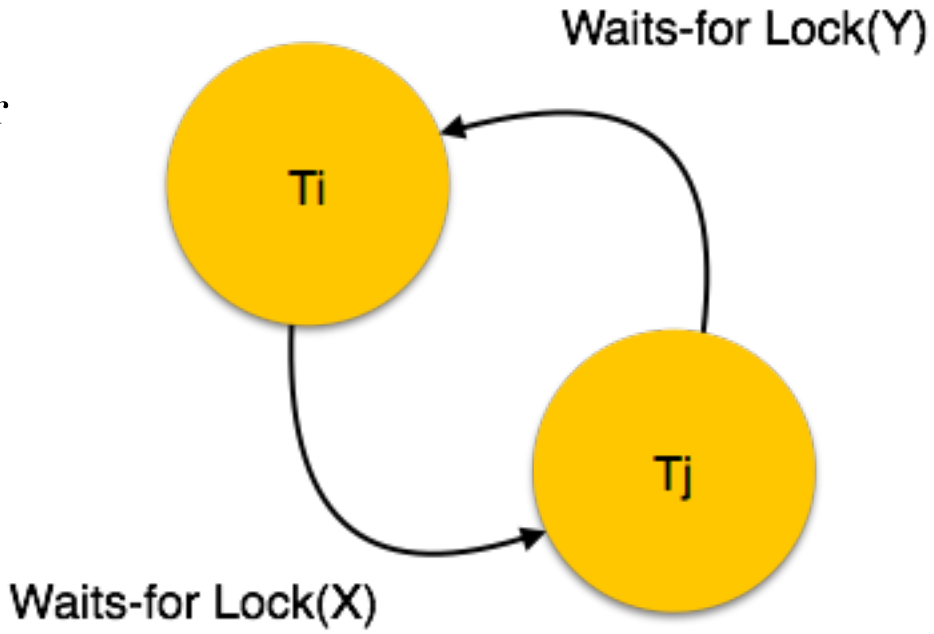
T_{old} will wound T_{new}

T_{new} waits for T_{old}



Deadlock Avoidance

- Waits-for graph:
  - For each transaction entering into the system, a node is created.
  - When a transaction T_i requests for a lock on an item, say X, which is held by some other transaction T_j, a directed edge is created from T_i to T_j.
  - If T_j releases item X, the edge between them is dropped and T_i locks the data item.
- The system maintains this wait-for graph for every transaction waiting for some data items held by others. The system keeps checking if there's any cycle in the graph.



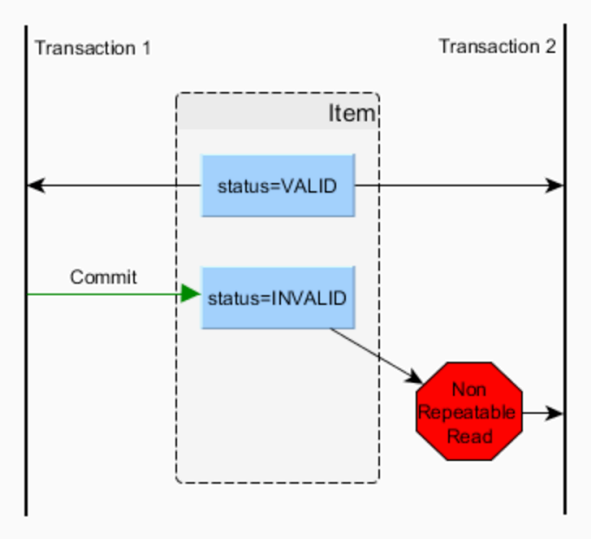
- Here, we can use any of the two following approaches –
- First, do not allow any request for an item, which is already locked by another transaction.
  - This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.
- Second, roll back one of the transactions.
  - It is not always feasible to roll back the younger transaction, as it may be important than the older one.
  - With the help of some relative algorithm, a transaction is chosen, which is to be aborted.
  - This transaction is known as the **victim** and the process is known as **victim selection**.

Transaction Characteristics in SQL

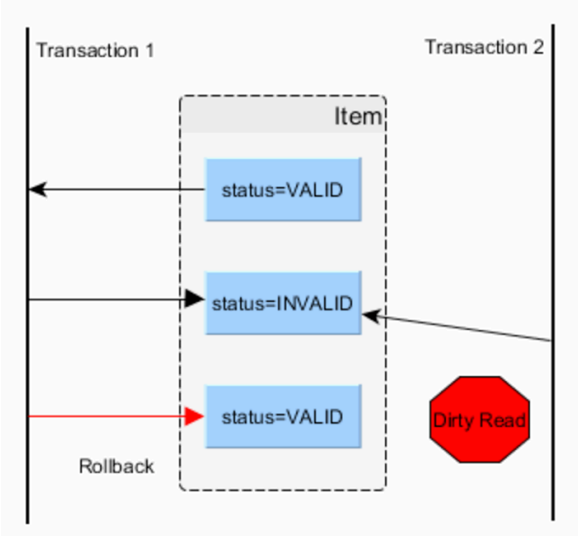
- SQL allows to specify three (3) characteristics of a transaction
  - Access mode
    - READ ONLY and READ WRITE
  - Diagnostics size
    - (Determines # of error conditions.)
  - Isolation level
    - Controls the extent to which a given transaction is exposed to actions of other transactions

Transaction Characteristics in SQL

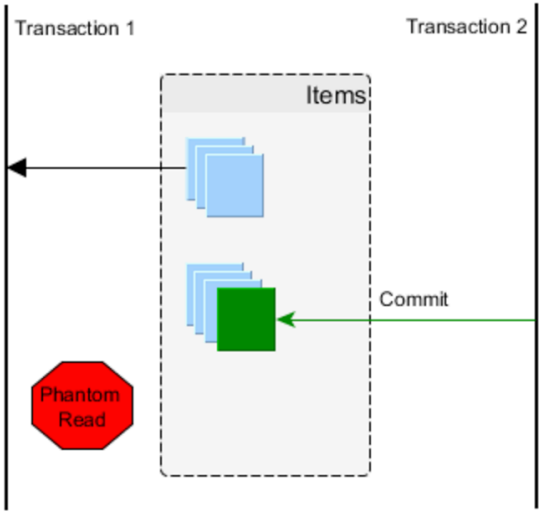
Unrepeatable Read



Dirty read



Phantom read



Transaction Characteristics in SQL

Isolation Level	Dirty Read	Unrepeatable Read	Phantom
READ UNCOMMITED	Maybe	Maybe	Maybe
READ COMMITED	No	Maybe	Maybe
REPEATABLE READ	No	No	Maybe
SERIALIZABLE	No	No	No

Example: SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ ONLY

Crash Recovery

- The recovery manager of a DBMS is responsible for ensuring transaction atomicity and durability
  - Atomicity by undoing the actions of transitions that do not commit
  - Durability by making sure that all actions of committed transactions survive system crashes.

- The transaction manager of a DBMS controls the execution of transactions.
  - Before reading and writing objects during normal execution, locks must be acquired (and released at some later time) according to a chosen locking protocol.

Stealing Frames and Forcing Pages

- Writing objects rises two important questions:
  - Can changes to an Object O made by a Transaction T be written to disk before T commits?
    - Other Transaction way 'steal' the page.
    - Steal approach
  - When a transaction commits, must we ensure that all changes it has made to objects are immediately forced to disk?
    - If yes, we call that a force approach.

    - Easy: No Steal, force approach
    - But have drawbacks

Drawbacks

- No-steal assumes all pages modified by ongoing transactions can be accommodated in the buffer pool
- Force approach results in excessive page I/O

- Most system uses a steal, no force approach.
  - Allows writing dirty frames to disk
  - Do not enforce immediate writing back after commit.

Who handles recovery?

- Recovery Manager
- Handles:
  - Atomicity
    - By undoing actions that do not commit
  - Durability
    - Making sure committed transactions survive system crashes

Solution?

- WAL (Write-Ahead Log)
- Enables the recovery manager to:
  - Undo the actions of aborted and incomplete transactions
  - Redo the actions of committed transactions.
- Example:
  - Changes of transactions that did not commit prior to crash might have written to the disk (due to steal approach)
    - Changes can be identified from the log and undone.
  - A transactions that committed before the crash may have updates not written to the disk. (due to no-force)
    - Changes can be identified from the log and written to disk

Example of Recovery Algorithm: ARIES

Overview of ARIES

- Algorithms for Recovery and Isolation Exploiting Semantics, or ARIES
  - A recovery algorithm designed to work with a no-force, steal database approach
  - Used by IBM DB2, Microsoft SQL Server and many other database systems

3 main principles

- Three main principles lie behind ARIES:
  - Write-ahead logging:
    - Any change to an object is first recorded in the log, and the log must be written to stable storage before changes to the object are written to disk.Repeating history during
  - Redo:
    - On restart after a crash, ARIES retraces the actions of a database before the crash and brings the system back to the exact state that it was in before the crash. Then it undoes the transactions still active at crash
  - Undo:
    - Changes made to the database while undoing transactions are logged to ensure such an action isn't repeated in the event of repeated restarts.

- (Refer: https://en.wikipedia.org/wiki/Algorithms_for_Recovery_and_Isolation_Exploiting_Semantics)

Summary

- Concurrency achieved by **interleaving TXNs** such that **isolation** & **consistency** are maintained
  - We formalized a notion of **serializability** that captured such a "good" interleaving schedule
- We defined **conflict serializability**, which implies serializability
- **Locking** allows only conflict serializable schedules
  - If the schedule completes… (it may deadlock!)

Acknowledgement

- Some of the slides in this presentation are taken from the slides provided by the authors.
- Many of these slides are taken from cs145 course offered by Stanford University.

- https://vladmihalcea.com/2014/01/05/a-beginners-guide-to-acid-and-database-transactions/

- Spark slides and material: Jonathan Carroll-Nellenback (CIRC, UofR)