Problem with Locks

DEADLOCK and DIRTY READ

Rules of Locks

• Several transactions can hold read locks on same data item
• Only one holds write lock on any one data item
• Transaction which tries for exclusive lock and cannot get it goes to sleep
• When exclusive lock is released, some sleeping transaction wakened
Deadlock

- **Transaction A**
  - Read X
  - Writelock X
  - Read Y
  - Readlock Y
  - Write X
  - Cannot get lock due to B’s Writelock -- A sleeps
  - Each sleeps, waiting for the other

- **Transaction B**
  - Read Y
  - Writelock Y
  - Readlock X
  - Write Y
  - Cannot get lock due to A’s Writelock -- B sleeps

---

**Deadlock Prevention Strategy**

- Change some rule about locks
- Normally: Can’t get lock ⇒ Sleep on Queue
- New rule: Can’t get lock ⇒ Some transaction may die
- Strategy: find something (X) that always happens when there is going to be a deadlock:
  - If **deadlock** about to happen Then **X** is true
- Turn it around to make the rule:
  - If **X** is true Then **deadlock** is about to happen
  - And someone must die to prevent it.
A 2PL Solution

- New Rule: Get all locks at the beginning of transaction
- If a needed lock is not available
  - Abort transaction
  - wait a little
  - Restart
- Notice change in rule about locks
- Problem: reduces concurrency
- Called Conservative 2PL
- Will see other solutions later

Second Problem: Dirty Read

- Source: DB policy may allow time between last unlock and DB promise to remember all.
- So this may occur
  - $T_1$ writes A
  - $T_1$ last unlock
  - $T_2$ reads A
  - $T_1$ fails
- Value of A is now INVALID
- $T_2$ must fail as well.
Dirty Read

Transaction A
Read X  Writelock X
Write X  Unlock X
ABORT and ROLL BACK

Transaction B
Read X  Readlock X
Read X  Unlock X
Cascading ROLL BACK

A 2PL Solution

• New Rule: Keep all locks until commit
• No uncommitted write can be read
• Called Strict 2PL
• Problem: reduces concurrency
• Will see other solutions later