Aborting
and
Crash Recovery

• Why transactions abort?
  – Deadlock avoidance
  – System error
  – user command
• Dependent transactions could be forced to abort too:
  1. Ti aborts
  2. Tj reads what Ti wrote
     =>
  3. Tj must abort (re-execute) EVEN IF Tj has committed!
     – What does “COMMIT” mean?
Concurrency details

2PH:
T1:  W(V)  Release E(V)                      ...  ABORT
T3:                                               R(V)  COMMIT

Strict 2PH:
• Ti releases locks and commits as atomic action
• Eliminates above problem

Choice of Restrictions:
• Strict:  Tj does not read or write until Ti commits
• Avoid cascaded abort:  Tj does not read until Ti commits
• Recoverable:  Tj only commits after Ti commits
  – CANNOT ABORT after COMMIT

Summary of 2-phase locking variations

• 2PH:  guarantees conflict serializable
• Strict 2PH: guarantees no cascaded aborts
• Conservative 2PH:  guarantees no deadlock
• Strict + conservative 2PH:  only allows reads of shared objects by uncommitted transactions.
Other consistency issues

Dynamics of DB can cause consistency problems even with Strict 2PL

**Example:**

T1
1. lock all pages containing records with property P
2. Take an aggregate of those records
3. Lock all pages containing records with property Q
4. Take an aggregate of those records

T2
1. Lock new page
2. Insert new record with property P on new page
3. Lock new page
4. Insert new record with property Q on new page

**Schedule:** T1:1 T1:2 T2:1,2,3,4 T1:3 T1:4

Aggregate for P before T2 inserts; aggregate for Q after T2 inserts

=> not serializable and not consistent

---

**Solutions?**

- Need to lock all now and future records
- **How?**
  - Lock whole file: pages and access - **COSTLY**
  - Predicate locking: lock all records satisfying predicate (e.g. salary > 100K)
  - **How?**
    - Special case: if only using index to reach records satisfying predicate
    - Lock pages in index which contain or **would contain** data entries to records satisfying predicate
Aborting – HOW?

See as part of algorithm for

**Crash recovery**

Goals of crash recovery
- Either transaction commits and is correct or aborts
- Commit means all actions of transaction have been executed
- Error model:
  - lose contents main memory
  - disk contents intact and correct

### Surviving crash

- **Crash recovery requirements**
  - If transaction has committed then still have results (on disk)
  - If transaction in process, either
    1. Transaction completely aborts
    OR
    2. Transaction can continue after restore as if no crash
  - Get serializable schedule such that transactions that committed before crash still commit and in same order

- => NEED LOG
ARIES algorithm

• Assumptions
  – Strict 2PL => no cascaded aborts
  – “in place” disk updates: data overwritten on disk
    • Page read into buffer, changed in buffer, written out again
    • Write of page to disk is atomic
      – How achieve?

• Log:
  – Sequential writes on separate disk
  – Write differences only
    • Multiple updates on single log page
    • Each log record has unique Log Sequence Number (LSN)
      – Strictly sequential