Timestamp Ordering

- Using timestamps
  - any conflicting read/write operations are executed in their timestamp order
  - simple and aggressive: schedule immediately and reject requests that arrive too late
  - each data object maintains read-timestamp (rts) and write-timestamp (wts)

- Timestamp ordering rule
  
  if \( p_i(x) \) and \( q_j(x) \) are conflicting operations
  then \( p_i(x) \) is processed before \( q_j(x) \) iff \( ts(T_i) < ts(T_j) \)

Theorem: If TO rule is enforced, then all the executions generated by the scheduler is serializable

Proof: If \( T_1 \rightarrow \ldots \rightarrow T_n \rightarrow T_1 \) exists in SG(H)
then \( ts(T_1) < \ldots < ts(T_n) < ts(T_1) \) due to TO rule.
A contradiction.
Timestamp Ordering Protocol

- Basic timestamp ordering protocol
  1) $r_i(x)$: if $ts(T_i) < wts(x)$ then reject it
     otherwise $(ts(T_i) \geq wts(x))$, schedule $r_i(x)$
     and set $rts(x) = \max (rts(x), ts(T_i))$
  2) $w_i(x)$: if $ts(T_i) < rts(x)$ then reject it
     if $ts(T_i) < wts(x)$ then reject it
     otherwise, schedule $w_i(x)$
     and set $wts(x) = \max (wts(x), ts(T_i))$

- when restarted, $T_i$ is assigned a new timestamp

- Thomas write rule (TWR)
  if $ts(T_i) < wts(x)$ for $w_i(x)$ and $ts(T_i) \geq rts(x)$
  then $w_i(x)$ can be ignored, instead of being rejected

--- delete obsolete write
Strict Timestamp Ordering

- SR but not RC
  - basic timestamp ordering is SR but not necessarily RC
    \[ w_1(x) \ r_2(x) \ w_2(y) \ c_2 \]

- Strict timestamp ordering
  - does not schedule any operation conflicting with \( w_i(x) \) until \( T_i \) terminates
    
    --- same as 2PL?

- Non-equivalence of 2PL and timestamp ordering

  \[ H_1 = r_2(x) \ w_3(x) \ c_3 \ w_1(y) \ c_1 \ r_2(y) \ w_2(y) \ c_2 \]

  - is this possible with timestamp ordering? with 2PL?

  - \( T_2 \) must release lock on \( x \) for \( T_3 \) to access, but then gets lock on \( y \) --- violation of two-phasedness

  - it is legal in strict timestamp ordering, equivalent to a serial schedule \( T_1 \ T_2 \ T_3 \)
Relationship between 2PL and TO

- Schedules generated by 2PL and timestamp ordering
  - they are all correct (i.e., serializable)
  - they are not the same sets: \( H_1 \) shows it
  - is the relationship inclusive?
    \[
    \{ S \in 2PL \} \subset \{ S \in TO \}?
    \]
    \[
    \{ S \in TO \} \subset \{ S \in 2PL \}?
    \]
    \[w_3(x) \preceq_3 w_2(x) \preceq_2 r_1(x)\]
    --- legal in 2PL, not possible using timestamp ordering

- Relationship between the two sets of schedules
Certifier Approach

● Optimistic approach
  - aggressive scheduling
  - three phases: read, validation, write phase
  - conflict resolution during validation phase
  - when c_i comes from T_i, check
    \[ RS(T_i) \cap WS(T_j) = \emptyset \]
    \[ WS(T_i) \cap RS(T_j) = \emptyset \]
    \[ WS(T_i) \cap WS(T_j) = \emptyset \]
● Validation: forward or backward
  - forward: validating transaction against active ones
  - backward: validating transaction against committed ones
● Non-blocking approach
  - abort and restart instead of blocking (deadlock-free)
  - problems of wasted resources and wasted aborts
Hybrid Approach

● Objective
  
  practical concurrency control protocols that allow transactions to meet the deadlines without reducing the concurrency level of the system

● Combine pessimistic and optimistic approaches
  - effective control of blocking and aborting
  - avoid unnecessary blocking and aborting

● Approach
  - adjust the serialization order of active transactions dynamically in favor of high priority transactions
  - relax the relationship between the serialization order and the past execution history
Multiversion Data Objects

- Objectives
  - improved system responsiveness by providing multiple versions (increased degree of concurrency)
  - reduce the probability of conflicts and rejection of tardy transactions by successive views of data objects

- Maintenance of multiple versions
  - each write creates a new version
  - system selects an appropriate version to read

- Potential problems
  - coordination for consistency
  - storage and processing overhead
Multiversion Timeestamp Ordering

MVTO scheduler translates operations on data objects into version operations to make it appear as if operations are processed in timestamp order on a single-version database

1) $r_i(x) \Rightarrow r_i(x_k)$

   where $x_k$ is the version of $x$ with largest timestamp
   not greater than $ts(T_i)$: $ts(T_k) \leq ts(T_i)$

2) $w_i(x)$:

   case 1: if $r_j(x_k)$ such that $ts(T_k) < ts(T_i) < ts(T_j)$
   is already processed, reject $w_i(x)$

   case 2: otherwise, translate $w_i(x)$ into $w_i(x_i)$
   and send it to data manager

3) $c_i$:

   delay processing of $c_i$ for recoverability until $c_j$
   of all $T_j$ that wrote versions read by $T_i$ has processed
Database Systems: Review

- Objectives of database systems
  - convenience and efficiency
  - name of the game: data abstraction

- Data models
  - object-based: ER model, OO model
  - record-based: relational, network, hierarchical

- Relational model
  - DML: relational algebra and calculus (tuple/domain)
  - query languages: SQL, QUEL, QBE
  - properties of good/bad design
  - functional dependencies and normal forms
  - MVD and higher normal forms
Database Systems: Review

- Physical organization
  - file organization and disk configuration (RAID)
  - indexing by B-tree and $B^+$-tree
  - hashing function (static and extendible)

- System issues
  - transaction: atomicity, serializability, recoverability
  - scheduling and concurrency control
    - two-phase locking and timestamp ordering
    - optimistic (certifier or validation) approach
  - recovery
    - undo/redo
    - log and checkpoint

- Other models: object-oriented, network, hierarchical