# Transaction Correctness

Enterprise Scale Data Management

Divy Agrawal
Department of Computer Science
University of California at Santa Barbara

## Lost Update (REVIEW)



- History corresponding to lost update:
  - H=r1[x]r2[x]w1[x]w2[x]
  - Possible serializations: H1=r1[x]w1[x]r2[x]w2[x] OR H2=r2[x]w2[x]r1[x]w1[x]
- Construct D(H), D(H1) and D(H2) and see if this H is not FSE either to H1 or H2?

➔ Indeed the interleaved history H is not Final State Equivalent to either of the serial orders.

## Fund Transfer (REVIEW)

3

- Fund Transfer History:
  - H=r2[x]w2[x]r1[x]r1[y]r2[y]w2[y]
  - Final State Equivalent to both T1-T2 and T2-T1.
- Even if we can develop an efficient tool to enforce FSR executions, it is not good enough for our purpose.



## Key Insight

4

- We need to strengthen the notion of final state serializability:
  - By not only focusing on the state of the database alone
  - But also requiring that the "database view" observed by each transaction in the equivalent schedules is identical (THIS IS MISSED BY FSE SINCE IT TREATS READ-ONLY TRANSACTIONS DEAD).

THIS LECTURE.

# View Serializability

**Definition 3.9 (View Equivalence):**
Schedules s and s‘ are **view equivalent**, denoted $s \approx_v s'$, if the following hold:
(i) op(s)=op(s‘)
(ii) H[s] = H[s‘]
(iii) $H_s[p] = H_{s'}[p]$ for all (read or write) steps

Where H and Hs are the Herbrand Semantics.

Note in the case of Final State Equivalence we only considered Herbrand Semantics of Database objects.

Here we are considering the Herbrand Semantics of each individual operation.



# View Serializability

**Theorem 3.2:**
For schedules s and s‘ the following statements hold.
(i) $s \approx_v s'$ iff op(s)=op(s‘) and RF(s)=RF(s‘)
(ii) $s \approx_v s'$ iff D(s)=D(s‘)

Where RF is the reads-from relations.

D(s) is the step graph we used for FSE.

# View Serializability

**Corollary 3.2 (checking equality of Step graphs):**
View equivalence of two schedules s and s' can be decided in time that is polynomial in the length of the two schedules.

**Definition 3.10 (View Serializability):**
A schedule s is **view serializable** if there exists a serial schedule s' s.t. $s \approx_v s'$.
VSR denotes the class of all view-serializable histories.



# Inconsistent Read Reconsidered

- **Inconsistent read anomaly:**

$I = r_2(x)\ w_2(x)\ r_1(x)\ r_1(y)\ r_2(y)\ w_2(y)\ c_1\ c_2$

→ history is not VSR !

$RF(I) = \{(t_0,x,t_2), (t_2,x,t_1), (t_0,y,t_1), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$
$RF(t_1\ t_2) = \{(t_0,x,t_1), (t_0,y,t_1), (t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$
$RF(t_2\ t_1) = \{(t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_1), (t_2,y,t_1), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$

*Observation: VSR properly captures our intuition*

## Relationship Between VSR and FSR

**Theorem 3.3:**
VSR ⊂ FSR.

**Theorem 3.4:**
Let s be a history without dead steps. Then s ∈ VSR iff s ∈ FSR.

LEFT AS EXERCISES



## On the Complexity of Testing VSR

**Theorem 3.5:**
The problem of deciding for a given schedule s whether s ∈ VSR holds is NP-complete.

Our two attempts based on FSE and VE
Resulted in failures.

Need something else!!!

## Conflicting Operations

**Definition 3.12 (Conflicts and Conflict Relations):**
Let s be a schedule, t, t‘ ∈ trans(s), t ≠ t‘.

(i) Two data operations p ∈ t and q ∈ t‘ are in **conflict** in s if they access the same data item and at least one of them is a write.

(ii) {(p, q)} | p, q are in conflict and $p <_s q$} is the **conflict relation** of s.



## Conflicts: What's the deal?

12

- Now that we have defined the notion of conflicts:
  - The intuition is if two histories maintain the order of conflicting operations they must influence the database and the transactions in the same way.
  - This is definition is operation and not grounded in semantics as was the case with FSE and CE.
- Ready to define EQUIVALENCE based on conflicts.

## Conflict Serializability

**Definition 3.13 (Conflict Equivalence):**
Schedules s and s‘ are **conflict equivalent**, denoted
$s \approx_c s'$, if
op(s) = op(s‘) and conf(s) = conf(s‘).



## Conflict Serializability

**Definition 3.14 (Conflict Serializability):**
Schedule s is **conflict serializable** if there is a serial schedule s‘ s.t. $s \approx_c s'$.
CSR denotes the class of all conflict serializable schedules.

**Example a:** $r_1(x)\ r_2(x)\ r_1(z)\ w_1(x)\ w_2(y)\ r_3(z)\ w_3(y)\ c_1\ c_2 w_3(z)\ c_3$ $\rightarrow \in$ CSR

**Example b:** $r_2(x)\ w_2(x)\ r_1(x)\ r_1(y)\ r_2(y)\ w_2(y)\ c_1\ c_2$ $\rightarrow \notin$ CSR

## Properties of CSR

**Theorem 3.8:**
CSR ⊂ VSR

**Example:** s = $w_1(x)\ w_2(x)\ w_2(y)\ c_2\ w_1(y)\ c_1\ w_3(x)\ w_3(y)\ c_3$
s ∈ VSR, but s ∉ CSR.



## Efficient Ways to Recognize CSR Executions

16

- What is a directed graph?
- Think of ways to associate a graph with a schedule!

# Conflict Graph

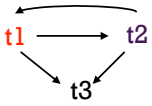
**Definition 3.15 (Conflict Graph):**
Let s be a schedule. The **conflict graph** G(s) = (V, E) is a directed graph with vertices V := commit(s) and
edges E := {(t, t') | t ≠ t' and there are steps p ∈ t, q ∈ t' with (p, q) ∈ conf

**Theorem 3.10:**
Let s be a schedule. Then s ∈ CSR iff G(s) is acyclic.

**Corollary 3.4:**
Testing if a schedule is in CSR can be done in time polynomial to the schedule's number of transactions.

**Example 3.12:**
s = $r_1(y)$ $r_3(w)$ $r_2(y)$ $w_1(y)$ $w_1(x)$ $w_2(x)$ $w_2(z)$ $w_3(x)$ $c_1$ $c_3$ $c_2$

G(s):






# Activity

18

- What is a characterization (in a mathematical sense)?
- How do you prove necessary and sufficient condition?
- What needs to be shown for the serializability theorem?

## Proof of the Conflict-Graph Theorem

(i) Let s be a schedule in CSR. So there is a serial schedule s‘ with conf(s) = conf(s‘).
Now assume that G(s) has a cycle $t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_k \rightarrow t_1$.
This implies that there are pairs $(p_1, q_2), (p_2, q_3), \ldots, (p_k, q_1)$ with $p_i \in t_i$, $q_i \in t_i$, $p_i <_s q_{(i+1)}$, and $p_i$ in conflict with $q_{(i+1)}$.
Because s‘ $\approx_c$ s, it also implies that $p_i <_{s'} q_{(i+1)}$.
Because s‘ is serial, we obtain $t_i <_{s'} t_{(i+1)}$ for i=1, ..., k-1, and $t_k <_{s'} t_1$.
By transitivity we infer $t_1 <_{s'} t_2$ and $t_2 <_{s'} t_1$, which is impossible.
This contradiction shows that the initial assumption is wrong. So G(s) is acyclic.



## Proof of the Conflict-Graph Theorem

(ii) Let G(s) be acyclic. So it must have at least one source node.
The following topological sort produces a total order < of transactions:
a) start with a source node (i.e., a node without incoming edges),
b) remove this node and all its outgoing edges,
c) iterate a) and b) until all nodes have been added to the sorted list.
The total transaction ordering order < preserves the edges in G(s); therefore it yields a serial schedule s‘ for which s‘$\approx_c$ s.

## Order Preserving Conflict Serializability

**Definition 3.18 (Order Preservation):**
Schedule s is **order preserving conflict serializable** if it is conflict equivalent to a serial schedule s‘ and
for all t, t‘ ∈ trans(s): if t completely precedes t‘ in s, then the same holds i
OCSR denotes the class of all schedules with this property.

**Theorem 3.12:**
OCSR ⊂ CSR.

**Example 3.13:**
$s = w_1(x)\ r_2(x)\ c_2\ w_3(y)\ c_3\ w_1(y)\ c_1$ → ∈ CSR
→ ∉ OCSR



## Commit-order Preserving Conflict Serializability

**Definition 3.19 (Commit Order Preservation):**
Schedule s is **commit order preserving conflict serializable** if
for all $t_i, t_j \in$ trans(s): if there are $p \in t_i$, $q \in t_j$ with $(p,q) \in$ conf(s) then $c_i <_s$
COCSR denotes the class of all schedules with this property.

**Theorem 3.13:**
COCSR ⊂ CSR.

**Theorem 3.14:**
Schedule s is in COCSR iff there is a serial schedule s‘ s.t. $s \approx_c$ s‘ and
for all $t_i, t_j \in$ trans(s): $t_i <_{s'} t_j \Leftrightarrow c_i <_s c_j$.

**Theorem 3.15:**
COCSR ⊂ OCSR.

**Example:**
$s = w_3(y)\ c_3\ w_1(x)\ r_2(x)\ c_2\ w_1(y)\ c_1$ → ∈ OCSR
→ ∉ COCSR

# Commit Serializability (SKIP)

**Definition 3.20 (Closure Properties of Schedule Classes):**
Let E be a class of schedules.
For schedule s let CP(s) denote the projection $\Pi_{commit(s)}$ (s).
E is **prefix-closed** if the following holds: $s \in E \Leftrightarrow p \in E$ for each prefix of
E is **commit-closed** if the following holds: $s \in E \Rightarrow CP(s) \in E$.

**Theorem 3.16:**
CSR is prefix-commit-closed, i.e., prefix-closed and commit-closed.

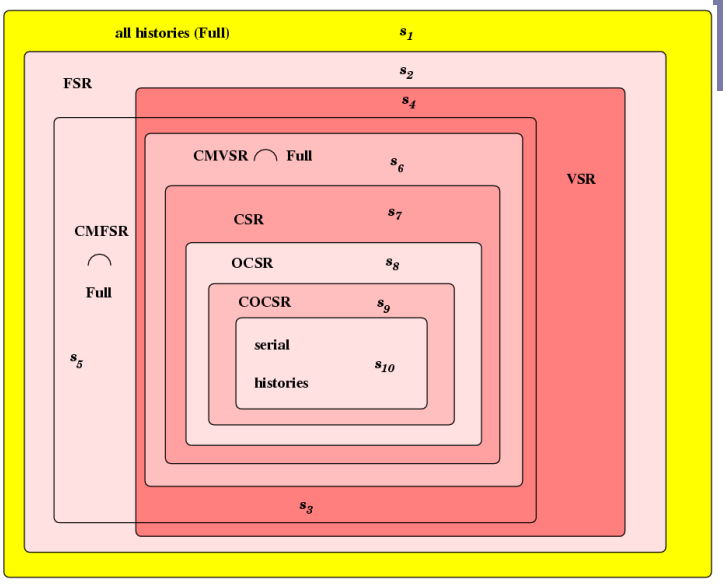
**Definition 3.21 (Commit Serializability):**
Schedule s is **commit-Θ-serializable** if CP(p) is Θ-serializable for each prefix p of s, where Θ can be FSR, VSR, or CSR.
The resulting classes of commit-Θ-serializable schedules are denoted CMFSR, CMVSR, and CMCSR.

**Theorem 3.17:**
(i) CMFSR, CMVSR, CMCSR are prefix-commit-closed.
(ii) CMCSR ⊂ CMVSR ⊂ CMFSR



# Landscape of History Classes





# Lessons Learned

- Equivalence to serial history is a natural correctness criterion
- CSR, albeit less general than VSR,
  is most appropriate for
  - complexity reasons
  - its generalizability to semantically rich operations
- OCSR and COCSR have additional beneficial properties (LATER)