A Framework for Transactional Consistency Models with Atomic Visibility

Daniel Solomon
30/05/18

Overview

- Introduction
- Notations and Definitions
- Transactional Consistency Models
- Models Relationship
- Optimizations
- Operational Model Equivalence

Overview

- **Introduction**
- Notations and Definitions
- Transactional Consistency Models
- Models Relationship
- Optimizations
- Operational Model Equivalence

Introduction

- Our main focus is databases
- What is a *database*?
  - Database is a organized collection of data
- There are many types of databases
  - We will talk about *replicated databases*

Introduction – Cont.

- Replicated database maintains shared data between several replicas
- A client may perform *transaction* in any replica
- Updates will propagate between all replicas
- Why replicated database?
  - *Availability*
  - *Low latency*
  - *Offline purpose*

Introduction – Cont.

- Ideally, we would like that the use of replicas will be transparent
- Formally, *serializability*
  - The database behaves as if it executed transactions serially on a non-replicated copy of the data
- Inefficient!
- Low latency and Availability properties may be affected

Transactions

- Transaction is a sequence of *events*, each event is a *read* or *write* operation
- Transaction may be committed or aborted
- Atomic Visibility
- We will use:
  - x, y as database objects
  - u, v, w as local variables
  - txn is a transaction

Anomalies

- In weaker consistency model than $Serializability$, non-serial behavior might appear, we will call them *anomalies*
- For example,
  - $txn_1 = \{x.write(post); y.write(empty)\} \mid\mid$
  - $txn_2 = \{u = x.read(\,); y.write(comment)\} \mid\mid$
  - $txn_3 = \{v = x.read(\,); w = y.read(\,)\}$
  - Under specific assumptions, $u = post, v = empty, w = comment$

Anomalies – Cont.

- The consistency model defines which anomalies might appear
- Different types of anomalies affects directly the semantic of the software that interacting with the database
- Up until now, the current consistency models are coupled with the internal implementation of the database
- Lack of generalization or rules when deciding which model to use

Declarative Models

- To deal with this problem, we propose a framework that is used to specify six different consistency models for replicated databases
- Specifications are *declarative* – do not refer to the db internals
- Allow reasoning at higher abstraction level

Atomic Visibility

- Usually *atomic visibility* is guaranteed, causing that for any transaction T:
  - All T events are visible at once
  - None of T events are visible
- Thanks to *atomic visibility,* transactions become our atomic unit so we may talk about relations on whole transactions

Overview

- Introduction
- **Notations and Definitions**
- Transactional Consistency Models
- Models Relationship
- Optimizations
- Operational Model Equivalence

Notations

- $Obj = \{x, y, \dots\}$, all of them integers
- $Op = \{read(x,n), write(x,n) | x \in Obj, n \in \mathbb{Z}\}$
- $EventId$ – a set of infinite indexes
- $historyevent = (i, o)$, $i \in EventId, o \in Op$
- $WEvent_x = \{(i, write(x,n) | i \in EventId, n \in \mathbb{Z}, x \in Obj\}$
- $REvent_x = \{(i, read(x,n) | i \in EventId, n \in \mathbb{Z}\ x \in Obj\}$
- $HEvent_x = WEvent_x \cup REvent_x$

Definition 1 – Transaction & History

- A transaction T is a pair (E, po), where $E \subseteq HEvent$ is a finite, non-empty set of events with distinct identifier. The program order po is a total order over E.
- A history H is a (finite or infinite) set of transactions with disjoint sets of event identifiers.
- All transactions in a history are assumed to be committed.

Definitions

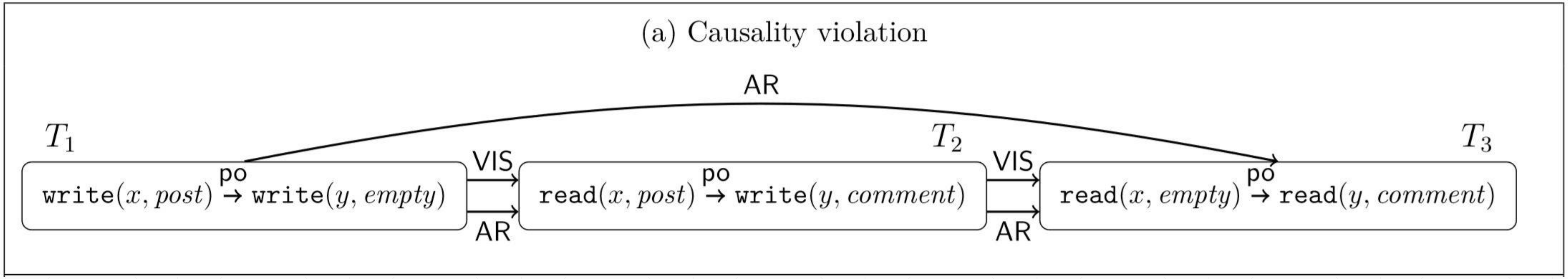
- Prefix-finite:
  - Relation is *prefix-finite* if every element has finitely many predecessors in the transitive closure of the relation ($\{a|(a,b)\epsilon Trans(R)\}$ is finite)
- VIS:
  - $T_1 \xrightarrow{VIS} T_2$ or $(T_1, T_2) \in VIS$, if the transaction T_2 is aware of the updates made by transaction T_1
- AR:
  - $T_1 \xrightarrow{AR} T_2$ or $(T_1, T_2) \in AR$, means that the version of objects written by T_2 supersede those written by T_1
- AR is a completion of VIS into a total order relation

Definition 2 – Abstract Execution

- An *abstract execution* is a triple $A = (H, VIS, AR)$ *where:*
  - H is a history
  - Visibility: $VIS \subseteq H \times H$
  - Arbitration: $AR \subseteq H \times H$ is a prefix-finite, total order relation
  - $AR \supseteq VIS$ ($\Rightarrow VIS$ is a prefix-finite, acyclic relation)

Example

- *Causality Violation* anomaly



Consistency Model

- A consistency model specification is a set of *consistency axioms* ϕ constraining executions.
- The model allows those histories for which there exists an execution that satisfies the axioms:
  - $Hist_{\phi} = \{H \mid \exists VIS, AR. (H, VIS, AR) \vDash \phi\}$
  - This set (or its complement) defines the anomalies in the consistency model ϕ

Overview

- Introduction
- Notations and Definitions
- **Transactional Consistency Models**
- Models Relationship
- Optimizations
- Operational Model Equivalence

Transactional Consistency Models

- We now describe 6 different consistency models
- Each model will be described by its axioms
- We start from the weakest model and we will strength them from one to another

(I) Read Atomic

- $\phi = \{Int, Ext\}$
- The weakest model we will see today

More Notations

- For a total order $R \subseteq A \times A$ and a set A, we let $\max_R(A)$ be the element $u \in A$ such that $\forall v \in A.\, v = u \vee (v, u) \in R$
- $R^{-1}(u) = \{v | (v, u) \in R\}$
- _ will be used for an irrelevant value

Internal Consistency

- Within the transaction, the database provides sequential semantics:
  - A read from an object returns the same value as the last write or read in this very transaction

$$\forall (E, \mathsf{po}) \in \mathcal{H}.\, \forall e \in E.\, \forall x, n.\, (e = (\_, \mathtt{read}(x, n)) \wedge (\mathsf{po}^{-1}(e) \cap \mathsf{HEvent}_x \neq \emptyset)) \implies \max_{\mathsf{po}}(\mathsf{po}^{-1}(e) \cap \mathsf{HEvent}_x) = (\_, \_(x, n)) \quad (\textsc{Int})$$

- *Unrepeatable reads* is disallowed as well:
  - if a transaction reads an object twice without writing to it in-between, it will read the same value in both cases

External Consistency

- We let $T \vdash Write\ x\!:n$ if T writes to x and the last value written is n:

$$\max_{po}(E \cap WEvent_x) = (\_, write(x, n))$$

- We let $T \vdash Read\ x\!:n$ if T makes an external read from x, before writing to x and n is the first value returned:

$$\min_{po}(E \cap REvent_x) = (\_, read(x, n))$$

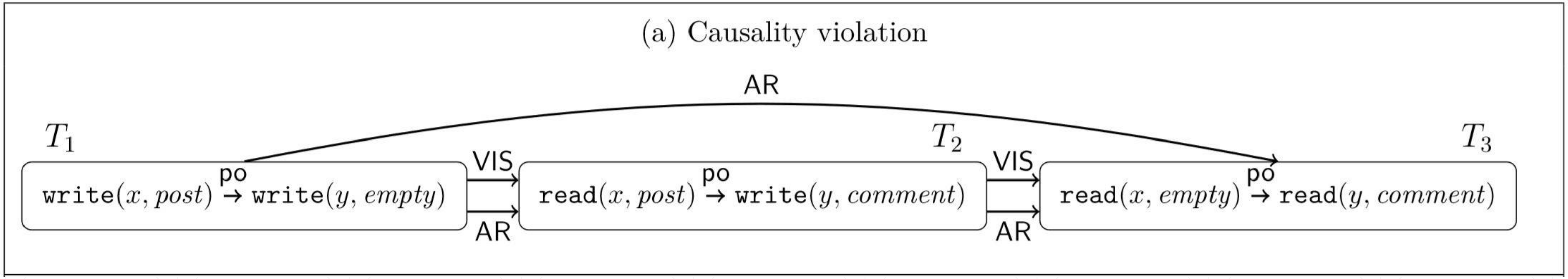
- The value returned by an external read in T is determined by the transactions VIS-preceding T that write to x
  - If none exists, T reads the initial value 0

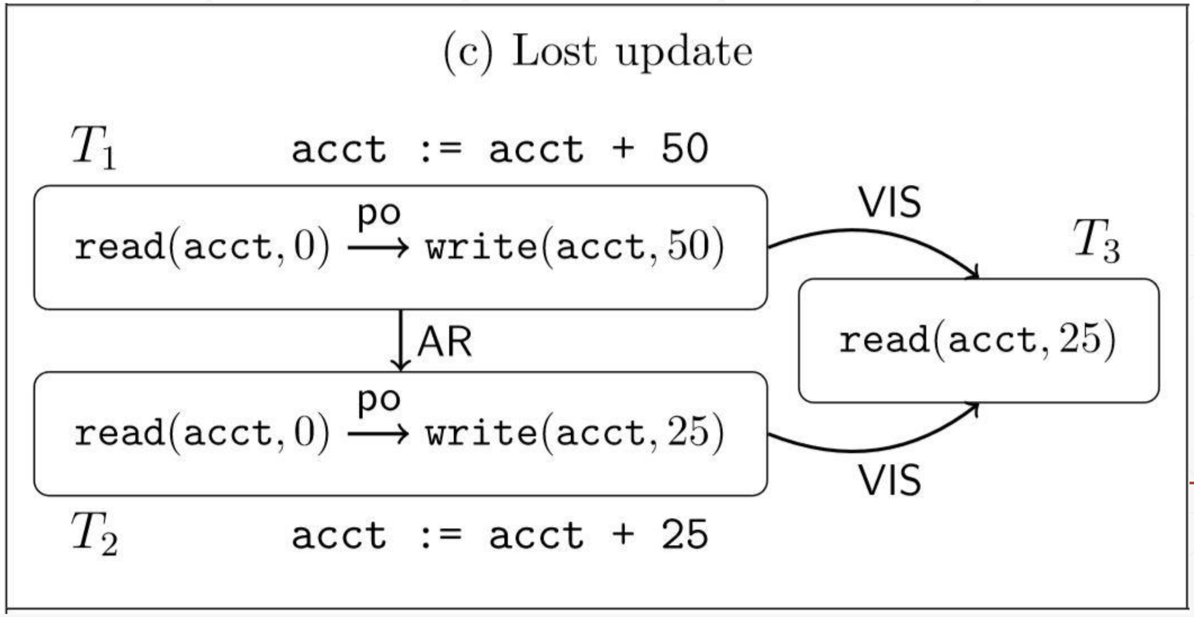
$$\begin{aligned}
&\forall T \in \mathcal{H}. \forall x, n. T \vdash \mathsf{Read}\ x : n \implies \\
&((\mathsf{VIS}^{-1}(T) \cap \{S \mid S \vdash \mathsf{Write}\ x : _\} = \emptyset \wedge n = 0) \vee \\
&\max_{\mathsf{AR}}(\mathsf{VIS}^{-1}(T) \cap \{S \mid S \vdash \mathsf{Write}\ x : _\}) \vdash \mathsf{Write}\ x : n) \qquad (\text{EXT})
\end{aligned}$$

Example – Internal Consistency

$$Write(x,1) \overset{po}{\rightarrow} Read(x,1) \xrightarrow{AR} Read(x,0) \overset{po}{\rightarrow} Read(x,0)$$

Example – External Consistency





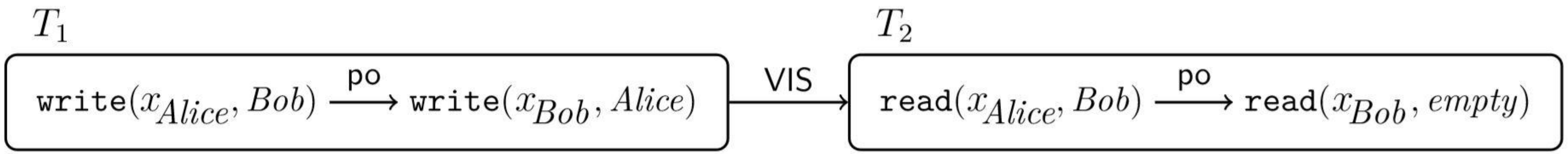
External Consistency – Cont.

- Ext implies two more properties:
  - *No Dirty reads*:
    - A committed transaction cannot read a value written by an aborted or an ongoing transaction
    - A transaction cannot read a value that was overwritten by the transaction that wrote it
  - *Atomic Visibility:*
    - *Either all or none of the transaction writes can be visible to another transaction*

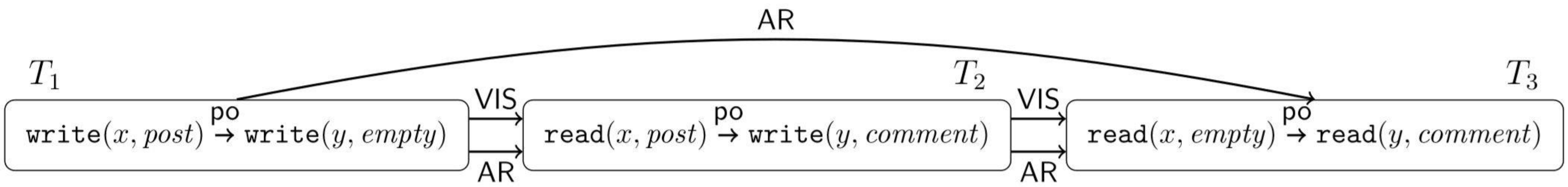
Read Atomic – Use Case

- Symmetric relation
- *Fractured Reads* anomaly

(b) Fractured reads

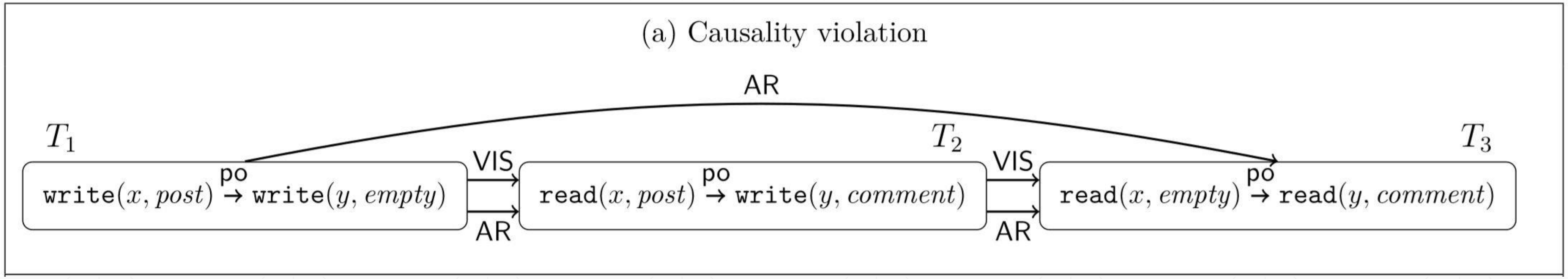


(a) Causality violation



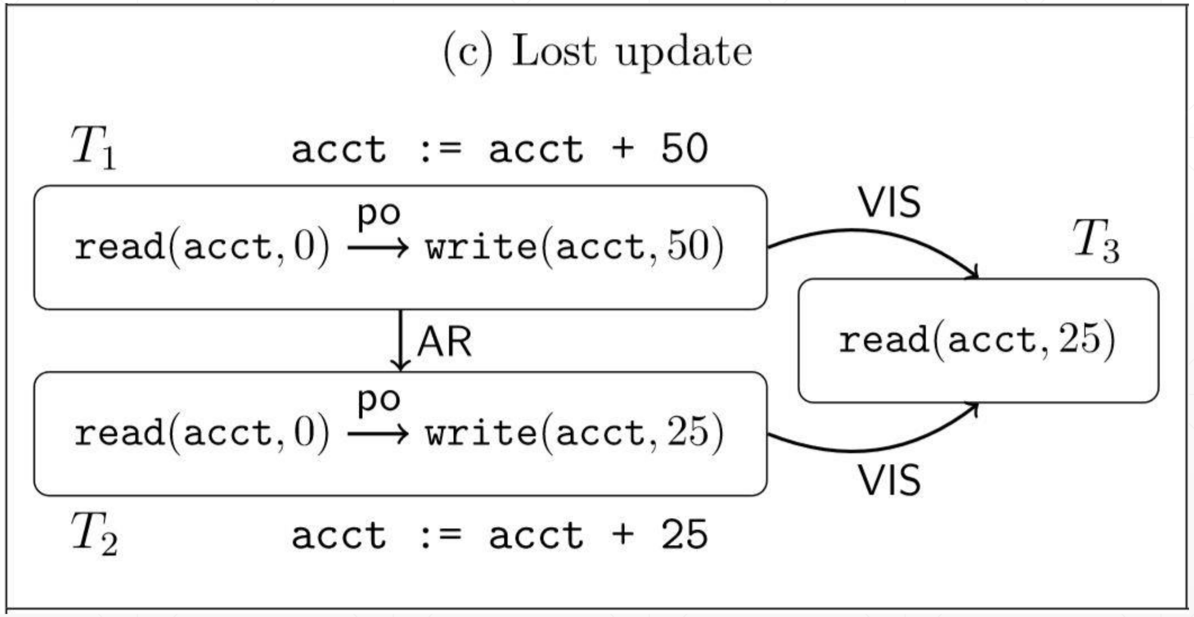
(II) Causal Consistency

- $\phi = \{Int, Ext, TransVis\}$
- *TransVis*:
  - Requiring VIS to be transitive



Read Atomic & Causal Consistency

- Both can be implemented without requiring any coordination among replicas:
  - A replica can decide to commit a transaction without consulting others
  - Advantage: *availability*
- *Lost Update:*
  An anomaly they both can't prevent



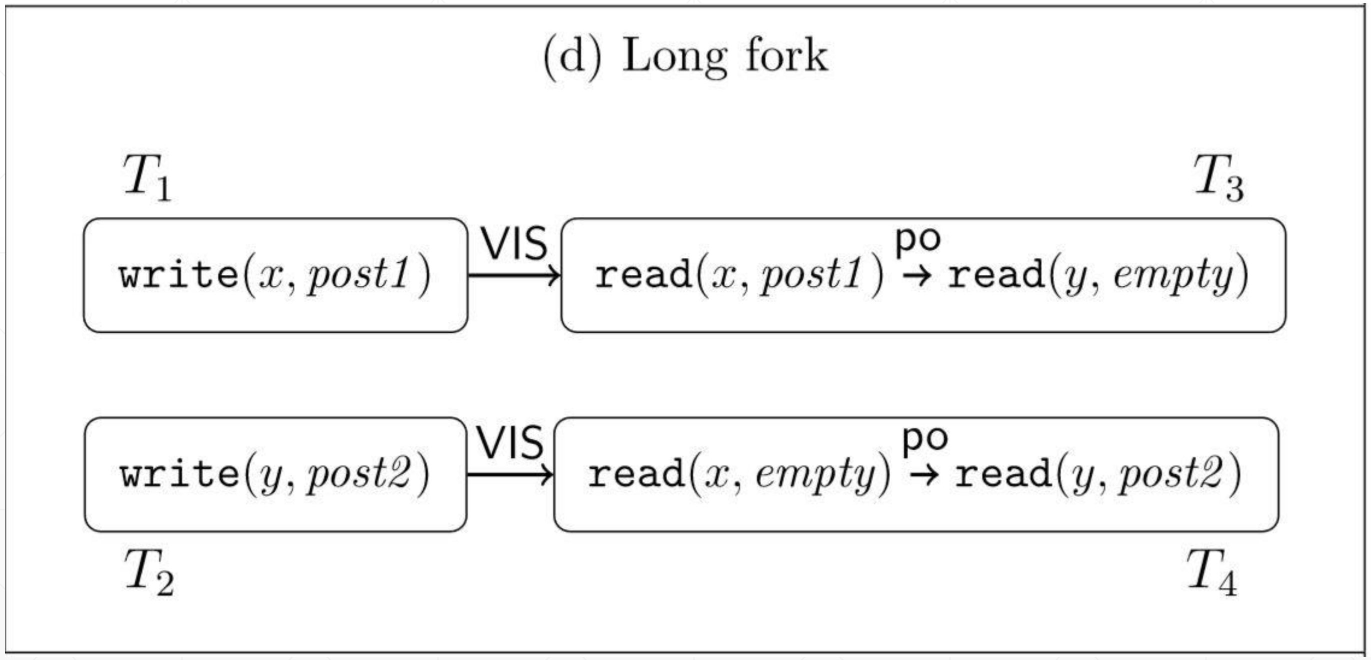
(III) Parallel Snapshot Isolation

- $\phi = \{Int, Ext, TransVis, NoConflict\}$
- *NoConflict:*
  - Disallows different transactions writing to the same object to be concurrent (prohibits *Lost Update* anomaly)
  - If two transactions write concurrently to an object, there must be a VIS relation between them

$$\forall T, S \in \mathcal{H}. (T \neq S \wedge T \vdash \mathsf{Write}\ x : \_ \wedge S \vdash \mathsf{Write}\ x : \_) \implies (T \xrightarrow{\mathsf{VIS}} S \vee S \xrightarrow{\mathsf{VIS}} T) \qquad (\textsc{NoConflict})$$

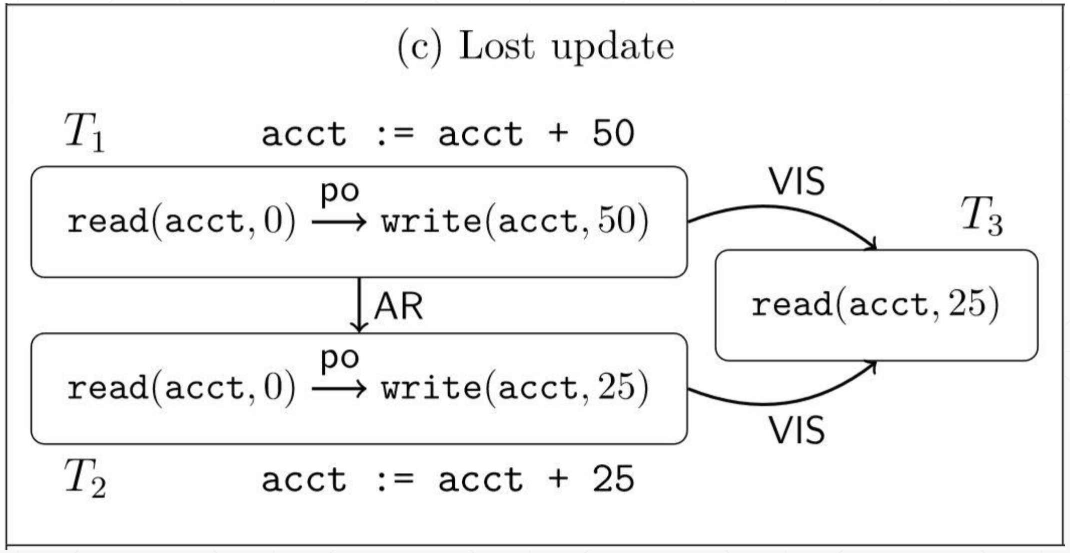
RA & CC & PSI

- Two concurrent transactions may be observed in different orders
- *Long Fork:*



(IV) Prefix Consistency

- $\phi = \{Int, Ext, TransVis, Prefix\}$
- *Prefix:*
  - If T observes S, then it also observes all AR-predecessors of S
  - $AR; VIS \subseteq VIS$



(V) Snapshot Isolation

- $\phi = \{Int, Ext, TransVis, NoConflict, Prefix\}$
- Prevents *Long Fork & Lost Update* anomalies
- Adopted by some major DB systems such as MongoDB, PostgreSQL, Oracle, MSSQL and many others.
- *Write Skew* anomaly:

(e) Write skew. Initially acct1 = acct2 = 60.

```
if (acct1 + acct2 > 100)
    acct1 := acct1 - 100
```

$\texttt{read}(\texttt{acct1}, 60) \xrightarrow{\texttt{po}} \texttt{read}(\texttt{acct2}, 60) \xrightarrow{\texttt{po}} \texttt{write}(\texttt{acct1}, -40)$ T_1

```
if (acct1 + acct2 > 100)
    acct2 := acct2 - 100
```

$\texttt{read}(\texttt{acct1}, 60) \xrightarrow{\texttt{po}} \texttt{read}(\texttt{acct2}, 60) \xrightarrow{\texttt{po}} \texttt{write}(\texttt{acct2}, -40)$ T_2

(VI) Serializability

- $\phi = \{Int, Ext, TotalVis\}$
- *TotalVis:*
  - VIS relation must be total

(e) Write skew. Initially `acct1` = `acct2` = 60.

```
if (acct1 + acct2 > 100)
    acct1 := acct1 - 100
```

$$\texttt{read}(\texttt{acct1}, 60) \xrightarrow{\texttt{po}} \texttt{read}(\texttt{acct2}, 60) \xrightarrow{\texttt{po}} \texttt{write}(\texttt{acct1}, -40) \quad T_1$$

```
if (acct1 + acct2 > 100)
    acct2 := acct2 - 100
```

$$\texttt{read}(\texttt{acct1}, 60) \xrightarrow{\texttt{po}} \texttt{read}(\texttt{acct2}, 60) \xrightarrow{\texttt{po}} \texttt{write}(\texttt{acct2}, -40) \quad T_2$$

Overview

- Introduction
- Notations and Definitions
- Transactional Consistency Models
- Models Relationship
- Optimizations
- Operational Model Equivalence

Models Relationship

Φ	Consistency model	Axioms (Figure 2)	Fractured reads	Causality violation	Lost update	Long fork	Write skew
RA	Read Atomic [6]	INT, EXT	✘	✓	✓	✓	✓
CC	Causal consistency [19, 12]	INT, EXT, TRANSVIS	✘	✘	✓	✓	✓
PSI	Parallel snapshot isolation [24, 21]	INT, EXT, TRANSVIS, NOCONFLICT	✘	✘	✘	✓	✓
PC	Prefix consistency [13]	INT, EXT, PREFIX	✘	✘	✓	✘	✓
SI	Snapshot isolation [8]	INT, EXT, PREFIX, NOCONFLICT	✘	✘	✘	✘	✓
SER	Serialisability [20]	INT, EXT, TOTALVIS	✘	✘	✘	✘	✘

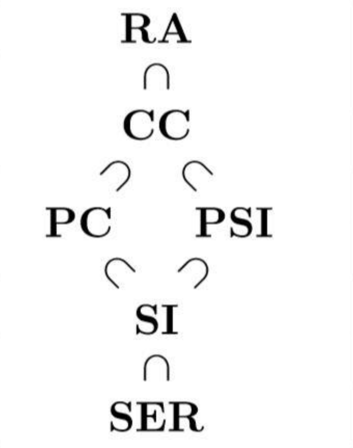


Figure 1 Consistency model definitions, anomalies and relationships.

Framework Benefits

- Declarative specifications
- High level relations
- Strengthening consistency is easy

Overview

- Introduction
- Notations and Definitions
- Transactional Consistency Models
- Models Relationship
- **Optimizations**
- Operational Model Equivalence

Optimizations

- Can we optimize an *abstract execution?*
- Since we speak about transactions and not low-level events, two different transactions may cause the same external behaviour
- *Observationally Refines:*
  - T observationally refines S, if we can replace T with S in the execution without invalidating the consistency axioms

Observationally Refines – Cont.

- Context:
  - *Abstract execution* with a “hole”
  - $\chi = (H \cup \{[\,]\}, VIS, AR), VIS, AR \subseteq (H \cup \{[\,]\}) \times (H \cup \{[\,]\})$
  - $\chi[T] = (H \cup \{[T]\}, VIS[[\,] \to T], AR[[\,] \to T])$
- Formal definition*:*
  - T_1 *observationally refines* T_2 on the consistency model ϕ ($T_1 \sqsubseteq_\phi T_2$) if

$$\forall \chi.\, \chi[T_1] \vDash \phi \Longrightarrow \chi[T_2] \vDash \phi$$

Optimizations – Cont.

- *Theorem 4:* Let T_1, T_2 be such that $(\{T_1, T_2\}, \emptyset, \emptyset) \vDash \mathrm{Int}$
  - *RA:* We have $\mathrm{T}_1 \sqsubseteq_{\boldsymbol{RA}} T_2$ if and only if for all x, n:
    $$\big(\neg(T_1 \vdash Read\ x{:}n) \Rightarrow \neg(T_2 \vdash Read\ x{:}n)\big) \land (T_1 \vdash Write\ x{:}n \Leftrightarrow T_2 \vdash Write\ x{:}n)$$
  - *CC/PC/SER:* We have $\mathrm{T}_1 \sqsubseteq_{\boldsymbol{\phi}} T_2$ if and only if for all x, n, m, l:
    $$\big(\neg(T_1 \vdash Read\ x{:}n) \Rightarrow (\neg(T_2 \vdash Read\ x{:}n) \land (T_1 \vdash Write\ x{:}n \Leftrightarrow T_2 \vdash Write\ x{:}n))\big) \land \big((T_1 \vdash Read\ x{:}n) \land (T_1 \vdash Write\ x{:}m \Rightarrow m = n)\big) \Rightarrow (T_2 \vdash Write\ x{:}l \Rightarrow l = n))$$
  - *SI/PSI:* We have $\mathrm{T}_1 \sqsubseteq_{\boldsymbol{\phi}} T_2$ if and only if for all x, n:
    $$\mathrm{T}_1 \sqsubseteq_{\boldsymbol{CC}} T_2 \ \land \big(\neg(T_1 \vdash Write\ x{:}\ n) \Rightarrow (\neg(T_2 \vdash Write\ x{:}n))\big)$$

Optimizations – Cont.

- Notice that since we defined *external reads* by $\mathrm{T} \vdash Read\ x$: ... and $\mathrm{T} \vdash Write\ x$: ..., two transactions that have the same *last writes* and the same *initial reads* are considered as equivalent since their *external behavior* is exactly the same

Overview

- Introduction
- Notations and Definitions
- Transactional Consistency Models
- Models Relationship
- Optimizations
- **Operational Model Equivalence**

Operational Models Equivalence

- Without any practical implementation, our axiomatic specifications may not describe a real database behavior
- We now prove that our abstract models are equivalent to operational ones
- It will be done by showing algorithms that are very close to actual implementations

The System

- The database consists of a set of *replicas*, $RId = \{r_0, r_1, \ldots\}$
- We assume that the system is fully connected
- All client operations in the same transaction are being executed in a specific replica
- Any transaction eventually terminates
  - Then the replica decides to abort or commit it
  - On commit, a *transaction log* broadcast message with the updates will be sent by the replica

Transaction Log

- t: ρ
  - $\rho \in \{write(x, n) | x \in Obj, n \in \mathbb{Z}\}^* \triangleq UpdateList$
  - $t \in \mathbb{N}$ is the unique *timestamp*
- $LogSet \triangleq \bigcup_{unique\ t} TransactionLog_t$

Replica State

- $RState \triangleq LogSet \times (UpdateList \uplus \{idle\})$
- The replica state is a pair (D, l)
  - D is the database copy of r, represented by the set of logs of committed transactions
  - l is either the sequence of updates done so far by a single running transaction or *idle*

System Configuration

- $Config \triangleq (RId \rightarrow RState) \times LogSet$
- The configuration of the whole system is $(R, M) \in Config$
  - $R(r)$ is the state of replica r
  - M is the pool of messages which are in transit among the replicas
- $\rightarrow$ transition relation is defined by $Config \times LEvent \times Config$
- $LEvent$ consists triples (i, r, o) $i \in EventId, r \in RId, o \in COp$
- COp is the set of all low level operations:
  - $COp = \{start, read(x, n), write(x, n), commit(t), abort, receive(t\colon \rho) | x \in Obj, n \in \mathbb{Z}, t \in \mathbb{N}, \rho \in UpdateList\}$

System Configuration – Transitions

- We now describe how each low-level operations changes the system configuration
- *Start*
  - *Start* may be operated only if the transaction is in *idle* state
  - In order to signify that the replica is executing a transaction we change *idle* to { }

$$\text{(Start)} \qquad \frac{\mathbf{e} = (\_, r, \texttt{start})}{(R[r \mapsto (D, \mathsf{idle})], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \varepsilon)], M)}$$

System Configuration – Transitions

- *Write*
  - The record $write(x, n)$ is appended to the current sequence of updates

$$(\text{Write}) \qquad \frac{\mathbf{e} = (\_, r, \texttt{write}(x, n))}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \rho \cdot \texttt{write}(x, n))], M)}$$

System Configuration – Transitions

- *Read*
  - The returned value is determined by a *lastval* function
  - *lastval* function is based on the maintained database copy or replica r and the current $UpdateList$
    - Search in $UpdateList$ for $write(x, \_)$ in reverse order
    - Search in D for $write(x, \_)$ by descending order of the timestamps
    - If no such $write$, a value 0 is returned

$$(\text{Read}) \qquad \frac{\mathbf{e} = (\_, r, \mathtt{read}(x, n)) \qquad n = \mathsf{lastval}(x, D \cup \{\infty : \rho\})}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \rho)], M)}$$

System Configuration – Transitions

- *Abort*
  - If a transaction aborts at replica r, the current sequence of updates is in r is cleared

$$(\text{Abort}) \qquad \frac{\mathbf{e} = (\_, r, \texttt{abort})}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \mathsf{idle})], M)}$$

System Configuration – Transitions

- *Commit*
  - If a transaction commits, it gets assigned a *timestamp* t and its transaction log is added to the message pool
  - t must be a distinct timestamp and must be greater than all timestamps that r is aware of
  - A single message is sent for each commit, which ensures *atomic visibility* property

$$\text{(Commit)} \quad \frac{\mathbf{e} = (\_, r, \texttt{commit}(t)) \qquad (\forall r', D'.\, R(r') = (D', \_) \implies (t : \_) \notin D') \qquad (\forall t'.\, (t' : \_) \in D \implies t > t')}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D \cup \{t : \rho\}, \mathsf{idle})], M \cup \{t : \rho\})}$$

System Configuration – Transitions

- *Receive*
  - A replica r may receive a transaction log from the message pool, only if it is in *idle* state
  - The received transaction log is added to the database copy

$$\text{(Receive)} \quad \frac{\mathbf{e} = (\_, r, \mathtt{receive}(t : \rho))}{(R[r \mapsto (D, \mathsf{idle})], M \cup \{(t : \rho)\}) \xrightarrow{\mathbf{e}} (R[r \mapsto (D \cup \{(t : \rho)\}, \mathsf{idle})], M \cup \{t : \rho\})}$$

System Configuration – Transitions – Cont.

- We define the semantics of the operational model by considering all sequences of transitions generated by $\rightarrow$ starting from an initial configuration
  - Log sets of all replicas are empty
  - The message pool is empty

Concrete Execution

- *Concrete execution:*
  - Let $(R_0, M_0) = (\forall r.(\emptyset, idle), \emptyset)$. A *concrete execution* is a pair $C = (E, \prec)$
  - $E \subseteq LEvent$, $\prec$ is a prefix-finite, total order over E
  - let $(e_1, e_2, \dots)$ events in E ordered by $\prec$, then for some configurations $(R_1, M_1), (R_2, M_2), \dots \in Config$, we have
  - $(R_0, M_0) \xrightarrow{e_1} (R_1, M_1) \xrightarrow{e_2} (R_2, M_2) \xrightarrow{e_3} \dots$

Equivalence – Read Atomic

- We want to show that the *operational model* defined by the transition function indeed defines the semantics of *Read Atomic* model
- TS_C:
  - Function that maps *read/write* event to its committed transaction

$$\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = \begin{cases} t, & \text{if } \exists r.\, \mathbf{e} \in \{(\_, r, \mathtt{read}(\_, \_)), (\_, r, \mathtt{write}(\_, \_))\} \land \\ & \quad \exists \mathbf{g} \in \mathbf{E}.\, \mathbf{g} = (\_, r, \mathtt{commit}(t)) \land \\ & \quad \neg(\exists \mathbf{f} \in \{(\_, r, \mathtt{commit}(\_)), (\_, r, \mathtt{abort})\}.\, (\mathbf{e} \prec \mathbf{f} \prec \mathbf{g})) \\ \text{undefined}, & \text{otherwise} \end{cases}$$

History

- We first map *concrete execution* into a history
- The history of $C = (E, \prec)$ is defined as follows:
  - $history(C) = \{T_t | \{e \in E | TS_C(e) = t\} \neq \emptyset\}\ where\ T_t = (E_t, po_t)$
  - $E_t = \{(i, o) | \exists e \in E.\, e = (i, \_, o) \wedge TS_C(e) = t\}$
  - $po_t = \{(i_1, o_1), (i_2, o_2) | (i_1, o_1), (i_2, o_2) \in E_t \wedge (i_1, \_, o_1) \prec (i_2, \_, o_2)\}$

Equivalence – Read Atomic – Cont.

- $history(ConcExec_{RA}) = Hist_{RA}$
- $ConcExec_{RA}$ is the set of *concrete executions* satisfying the *Read Atomic* model constraints

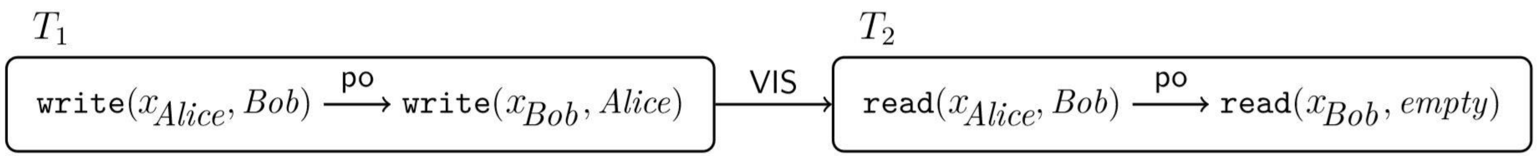
Equivalence – Read Atomic – Proof Outline

- $history(ConcExec_{RA}) \subseteq Hist_{RA}$
- Let $C = (E, \prec) \in ConcExec_{RA}$, our goal is to show that $history(C) \in Hist_{RA}$
- We build an *abstract execution* from C:
  - $A = (history(C), VIS, AR)$
  - $AR = \{(T_{t_1}, T_{t_2}) | t_1 < t_2\}$
  - $VIS = \left\{ (T_{t_1}, T_{t_2}) \middle| \begin{array}{c} \exists e_1, e_2 \in E. \exists r. \\ e_1 \in \{(\_, r, commit(t_1)), (\_, r, receive(t_1 : \_))\} \wedge e_2 = (\_, r, commit(t_2)) \wedge e_1 \prec e_2 \end{array} \right\}$

Equivalence – Read Atomic – Proof Outline – Cont.

- This construction provides:
  - AR – lifts the order of timestamps to transactions
  - VIS – reflects message delivery
- We can show that any *abstract execution* constructed from a *concrete execution* as above, satisfies Int, Ext and hence $\in Hist_{RA}$

Example – Read Atomic



(b) Fractured reads

(Start)
$$\frac{\mathbf{e} = (\_, r, \texttt{start})}{(R[r \mapsto (D, \mathsf{idle})], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \varepsilon)], M)}$$

(Write)
$$\frac{\mathbf{e} = (\_, r, \texttt{write}(x, n))}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \rho \cdot \texttt{write}(x, n))], M)}$$

(Read)
$$\frac{\mathbf{e} = (\_, r, \texttt{read}(x, n)) \qquad n = \mathsf{lastval}(x, D \cup \{\infty : \rho\})}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \rho)], M)}$$

(Abort)
$$\frac{\mathbf{e} = (\_, r, \texttt{abort})}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \mathsf{idle})], M)}$$

(Commit)
$$\frac{\begin{array}{l}\mathbf{e} = (\_, r, \texttt{commit}(t)) \\ (\forall r', D'.\, R(r') = (D', \_) \implies (t : \_) \notin D') \qquad (\forall t'.\, (t' : \_) \in D \implies t > t')\end{array}}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D \cup \{t : \rho\}, \mathsf{idle})], M \cup \{t : \rho\})}$$

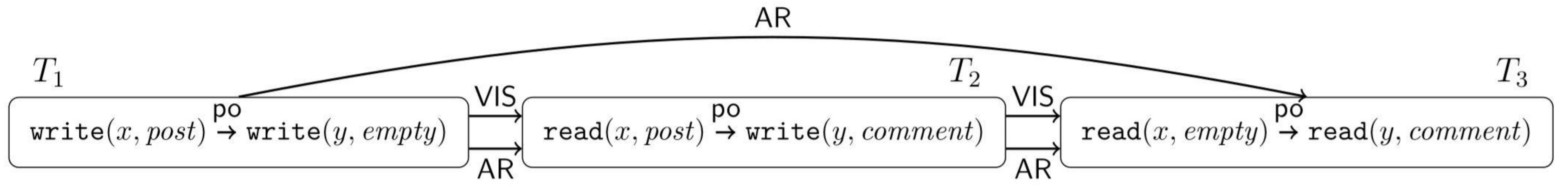
(Receive)
$$\frac{\mathbf{e} = (\_, r, \texttt{receive}(t : \rho))}{(R[r \mapsto (D, \mathsf{idle})], M \cup \{(t : \rho)\}) \xrightarrow{\mathbf{e}} (R[r \mapsto (D \cup \{(t : \rho)\}, \mathsf{idle})], M \cup \{t : \rho\})}$$

Stronger Operational Models – Causal Consistency

- For the stronger models, we will explain how to fulfill the axioms by constraining the communication protocol between the replicas
- *CausalDeliv:*
  - Implies TransVis, ensures that the message delivery is causal
  - If a replica r sends the transaction log of t_2 after it sends or receives the transaction log of t_1, then every other replica r' will receive the log t_2 only after it receives or sends the log t_1

$$\big(\mathbf{e}_1 \in \{(\_, r, \mathtt{receive}(t_1 : \_)), (\_, r, \mathtt{commit}(t_1))\} \land \mathbf{e}_2 = (\_, r, \mathtt{commit}(t_2)) \land \mathbf{e}_1 \prec \mathbf{e}_2 \land r \neq r' \land \mathbf{f}_2 = (\_, r', \mathtt{receive}(t_2 : \_))\big) \implies \big(\exists \mathbf{f}_1 \in \{(\_, r', \mathtt{receive}(t_1 : \_)), (\_, r', \mathtt{commit}(t_1))\}.\, \mathbf{f}_1 \prec \mathbf{f}_2\big) \qquad \text{(CausalDeliv)}$$

Example – Causal Consistency



(a) Causality violation

$$\big(\mathbf{e}_1 \in \{(\_, r, \texttt{receive}(t_1 : \_)), (\_, r, \texttt{commit}(t_1))\} \land \mathbf{e}_2 = (\_, r, \texttt{commit}(t_2)) \land \mathbf{e}_1 \prec \mathbf{e}_2 \land r \neq r' \land \mathbf{f}_2 = (\_, r', \texttt{receive}(t_2 : \_))\big) \implies \big(\exists \mathbf{f}_1 \in \{(\_, r', \texttt{receive}(t_1 : \_)), (\_, r', \texttt{commit}(t_1))\}.\ \mathbf{f}_1 \prec \mathbf{f}_2\big) \quad \text{(CausalDeliv)}$$

Stronger Operational Models – Prefix Consistency

- *MonTS:*
  - Timestamps must agree with the order in which transactions commit

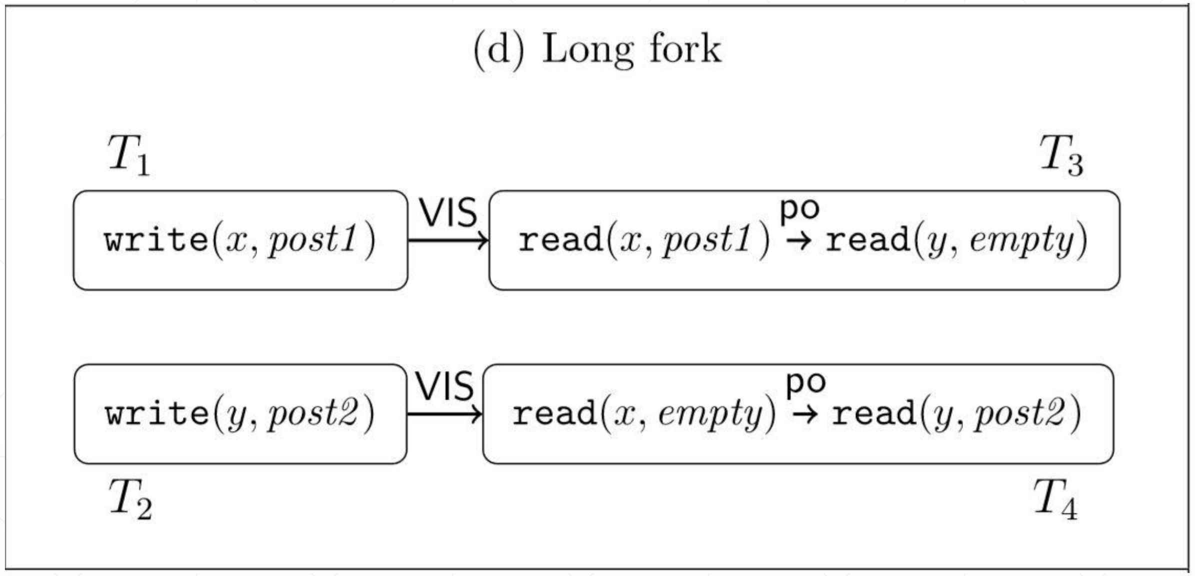
$$\big(\mathbf{e}_1 = (\_, \_, \texttt{commit}(t_1)) \land \mathbf{e}_2 = (\_, \_, \texttt{commit}(t_2)) \land \mathbf{e}_1 \prec \mathbf{e}_2\big) \implies t_1 < t_2 \qquad \text{(MonTS)}$$

- TotalDeliv
  - Each transaction access a database snapshot that is closed under adding transactions with timestamps smaller than the ones already present in the snapshot

$$\big(\mathbf{g} = (\_, r, \texttt{start}) \land \mathbf{e}_2 \in \{(\_, r, \texttt{commit}(t_2)), (\_, r, \texttt{receive}(t_2 : \_))\} \land \mathbf{f} = (\_, \_, \texttt{commit}(t_1)) \land t_1 < t_2 \land \mathbf{e}_2 \prec \mathbf{g}\big) \implies \big(\exists \mathbf{e}_1 \in \{(\_, r, \texttt{commit}(t_1)), (\_, r, \texttt{receive}(t_1 : \_))\}. \mathbf{e}_1 \prec \mathbf{g}\big) \qquad \text{(TotalDeliv)}$$

- Both can be implemented via a central server
- Together guarantee *Prefix*

Example – Prefix Consistency



$$\big(\mathbf{e}_1 = (\_, \_, \mathtt{commit}(t_1)) \land \mathbf{e}_2 = (\_, \_, \mathtt{commit}(t_2)) \land \mathbf{e}_1 \prec \mathbf{e}_2\big) \implies t_1 < t_2 \qquad \text{(MonTS)}$$

$$\big(\mathbf{g} = (\_, r, \mathtt{start}) \land \mathbf{e}_2 \in \{(\_, r, \mathtt{commit}(t_2)), (\_, r, \mathtt{receive}(t_2 : \_))\} \land \mathbf{f} = (\_, \_, \mathtt{commit}(t_1)) \land t_1 < t_2 \land \mathbf{e}_2 \prec \mathbf{g}\big) \implies \big(\exists \mathbf{e}_1 \in \{(\_, r, \mathtt{commit}(t_1)), (\_, r, \mathtt{receive}(t_1 : \_))\}. \mathbf{e}_1 \prec \mathbf{g}\big) \qquad \text{(TotalDeliv)}$$

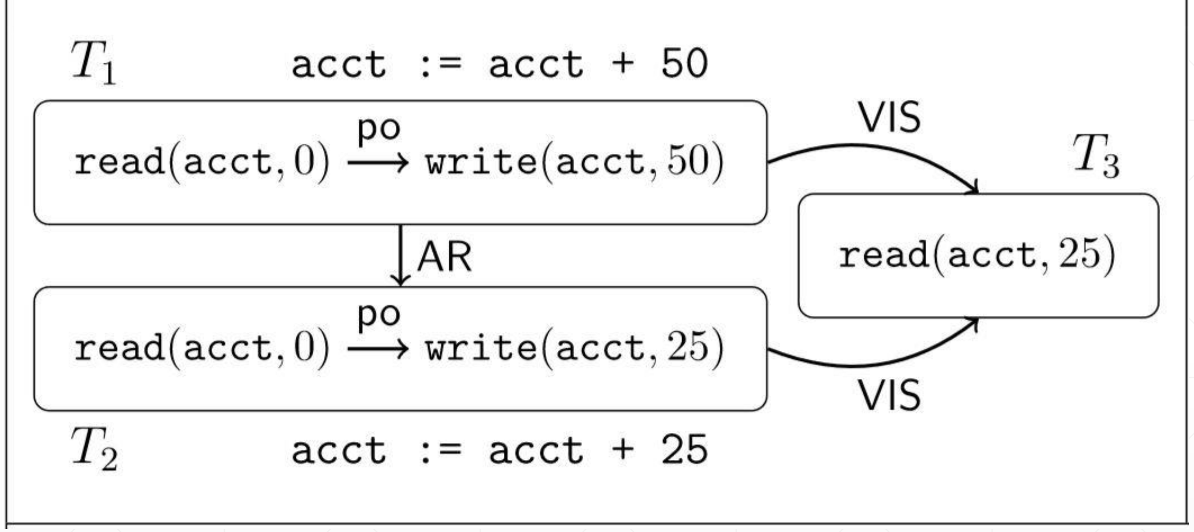
Stronger Operational Models – Parallel Snapshot Isolation

- *ConfictCheck:*
  - Allows transaction T_1 to commit at replica r only if it passes a *conflict detection check*:
  - if T_1 updates an object x that is also updated by a transaction T_2 committed at replica r', then the replica r must have received the log of T_2
  - If the check fails, r must abort the transaction

$$\Big(\mathbf{e}_1 = (\_, r, \mathtt{write}(x, \_)) \wedge \mathbf{f}_1 = (\_, r, \mathtt{commit}(t_1)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_1) = t_1 \wedge$$
$$\mathbf{e}_2 = (\_, r', \mathtt{write}(x, \_)) \wedge \mathbf{f}_2 = (\_, r', \mathtt{commit}(t_2)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_2) = t_2 \wedge \mathbf{f}_2 \prec \mathbf{f}_1 \wedge r \neq r'\Big)$$
$$\Longrightarrow \Big(\exists \mathbf{g} \in \mathbf{E}.\, \mathbf{g} = (\_, r, \mathtt{receive}(t_2 : \_)) \wedge \mathbf{g} \prec \mathbf{f}_1\Big), \qquad \text{(ConflictCheck)}$$

- May be implemented by requiring replica to coordinate with others before a commit

Example – Parallel Snapshot Isolation



(c) Lost update

$$\Big(\mathbf{e}_1 = (\_, r, \mathtt{write}(x, \_)) \wedge \mathbf{f}_1 = (\_, r, \mathtt{commit}(t_1)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_1) = t_1 \wedge$$

$$\mathbf{e}_2 = (\_, r', \mathtt{write}(x, \_)) \wedge \mathbf{f}_2 = (\_, r', \mathtt{commit}(t_2)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_2) = t_2 \wedge \mathbf{f}_2 \prec \mathbf{f}_1 \wedge r \neq r'\Big)$$

$$\implies \Big(\exists \mathbf{g} \in \mathbf{E}.\, \mathbf{g} = (\_, r, \mathtt{receive}(t_2 : \_)) \wedge \mathbf{g} \prec \mathbf{f}_1\Big), \qquad \text{(ConflictCheck)}$$

Stronger Operational Models

Φ	Constraints	Φ	Constraints	Φ	Constraints
RA	None	**PSI**	(CausalDeliv), (ConflictCheck)	**SI**	(MonTS), (TotalDeliv), (ConflictCheck)
CC	(CausalDeliv)	**PC**	(MonTS), (TotalDeliv)		

Conclusion

- We have proposed a framework for specifying transactional consistency models of replicated databases
- We derived 6 different models using the framework
- The models are declarative which gives us a better understanding (?) of the database behaviour and allows us to discuss about the relations between the transactions
- The declarative framework may be used to prove correctness and specify optimizations in a more elegant and simpler way
- Using this framework we may create some new consistency models
- For database architecture designer, the framework helps to determine which model to use for maximum efficiency

Thank You!