Formalising the Foundation

Modelling and Verifying a Database Backend

Joint Work With:

Marc Shapiro (LIP6 & INRIA, Paris)
Annette Bieniusa (RPTU, Kaiserslautern)
Carla Ferreira (NOVA LINCS, Lisbon)

With Support From:

Saalik Hatia (Scaleway, Paris)
Emilie Ma (UBC, Canada)
Jaurel Fosset (Polytech, Dijon)
Gustavo Petri (Amazon Research)

Ayush Pandey, Telecom SudParis and INRIA

11.07.2025, IIIT Delhi

Outline

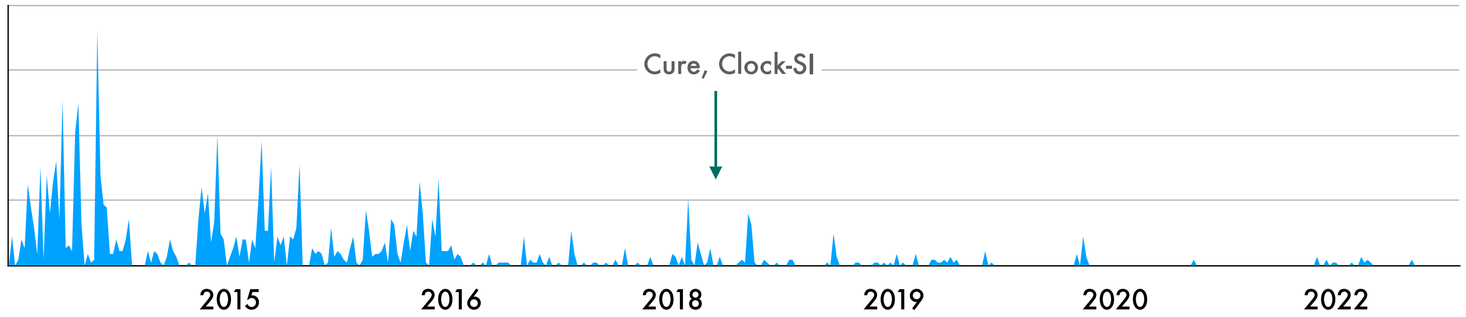
1. Background of this work and Motivation
2. Introduction
    1. Rich Datatypes
    2. Transactions
    3. Traces and their correctness (TCC)
    4. Database Backend
3. Stores
    1. Semantics
    2. Journal
    3. Map
4. Transaction Semantics
5. Correctness of transaction semantics
6. Improved definition of TCC
7. Proof Outline for correctness of the system specification
8. Specification to Implementation
9. Future

Learnings from AntidoteDB

Began ca. 2015

Clock-SI protocol in 2018

Bug fixing since then



What Happened?

Databases are large, complex

AntidoteDB: Planet-scale, highly available, replicated db

- ≈ 6 PhD Students => CRDTs, Clock-SI, Sharding, Lazy Replication, Lazy CC
- Heavily Engineered, Optimised. 60 KLoC (Erlang)
- Too much code + tired PhD students = Unknown Bugs = Data Loss!

Redis ≈ 100 KLoc (C)

RocksDB ≈ 300 KLoC (C++)

Overarching Research Theme

We know, formal methods help avoid bugs in systems.

Can we specify and verify a set of building blocks for a database which, when wrapped and composed correctly, can be used to build a correct database?

Aside: Rich Data Types

Conflict Free Replicated Datatypes (CRDT)

Concurrent updates without synchronisation + Guaranteed Convergence

Key Properties:

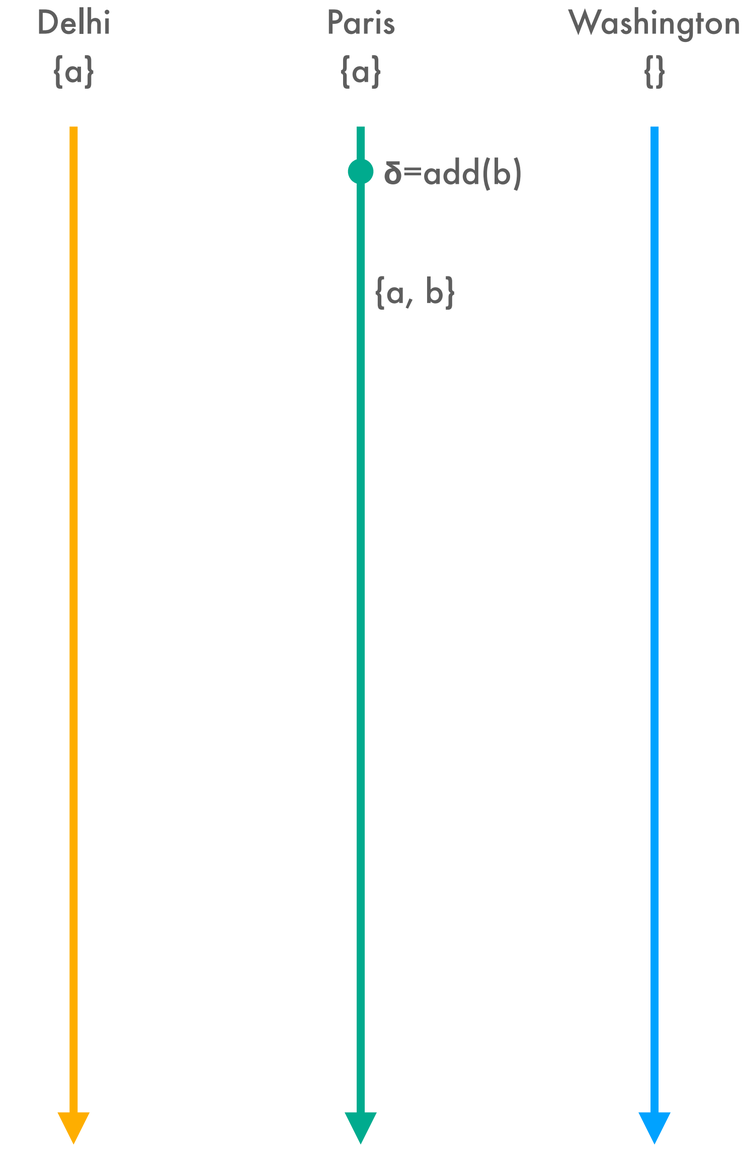
- Commutative, Associative and Idempotent Operations
- Mergeable States
- No Locks or consensus for updates

Implementation:

- **Effect (δ):** Takes the state of the CRDT and produces a new state
- **Merge:** Takes two CRDT states and **deterministically** produces a new state

Example: Add-Wins Set

- Effects: Add, Remove
- Merge: Between concurrent add and remove, add wins.



Aside: Rich Data Types

Conflict Free Replicated Datatypes (CRDT)

Concurrent updates without synchronisation + Guaranteed Convergence

Key Properties:

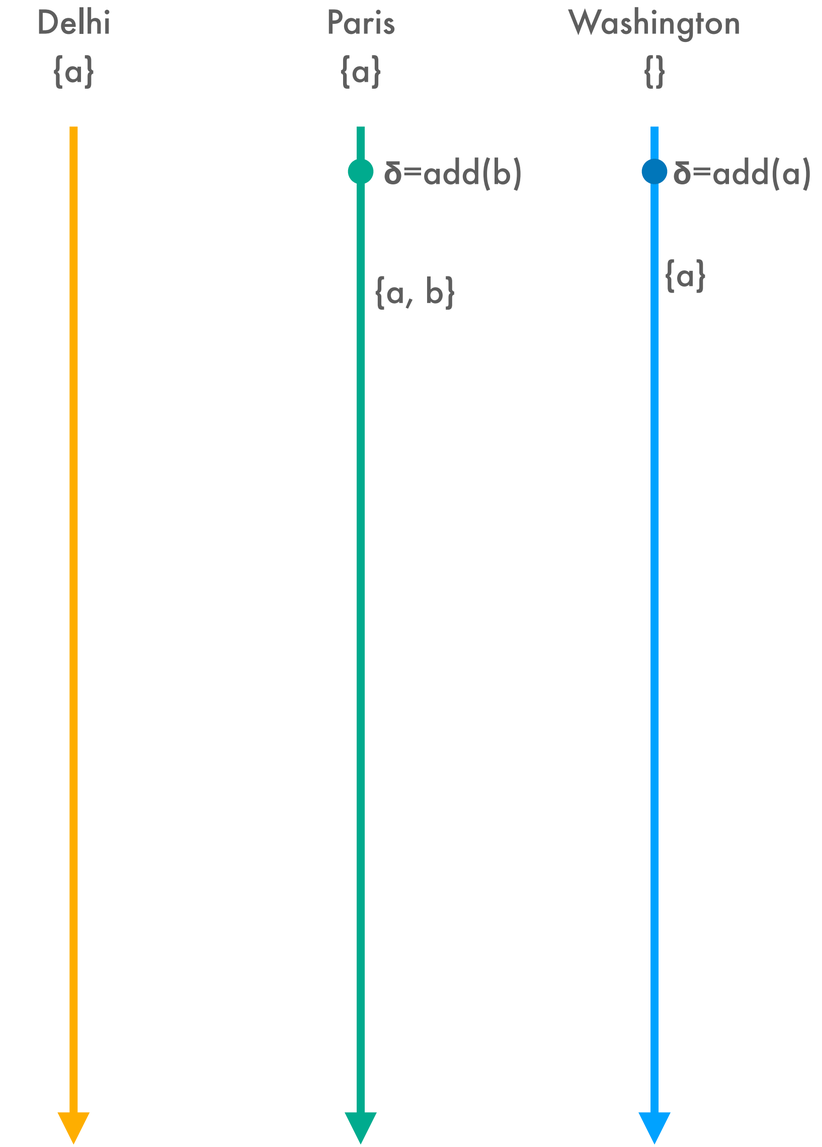
- Commutative, Associative and Idempotent Operations
- Mergeable States
- No Locks or consensus for updates

Implementation:

- **Effect (δ):** Takes the state of the CRDT and produces a new state
- **Merge:** Takes two CRDT states and **deterministically** produces a new state

Example: Add-Wins Set

- Effects: Add, Remove
- Merge: Between concurrent add and remove, add wins.



Aside: Rich Data Types

Conflict Free Replicated Datatypes (CRDT)

Concurrent updates without synchronisation + Guaranteed Convergence

Key Properties:

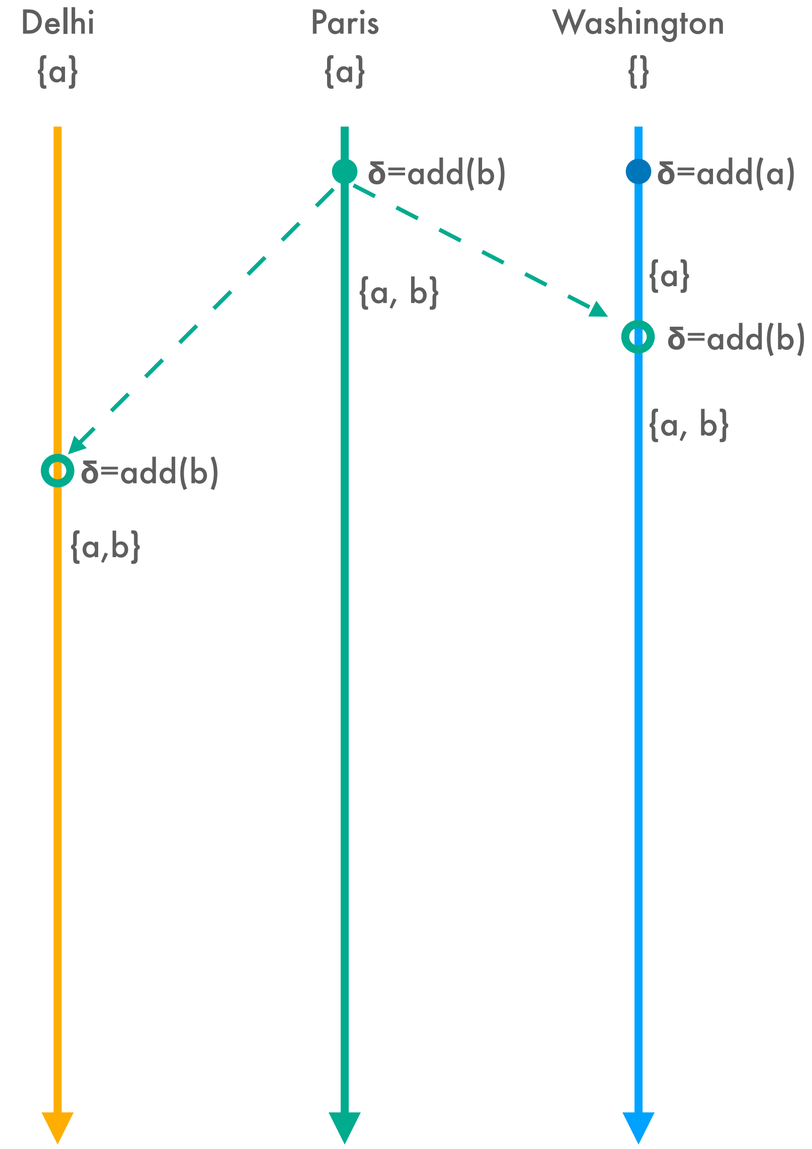
- Commutative, Associative and Idempotent Operations
- Mergeable States
- No Locks or consensus for updates

Implementation:

- **Effect (δ):** Takes the state of the CRDT and produces a new state
- **Merge:** Takes two CRDT states and **deterministically** produces a new state

Example: Add-Wins Set

- Effects: Add, Remove
- Merge: Between concurrent add and remove, add wins.



Aside: Rich Data Types

Conflict Free Replicated Datatypes (CRDT)

Concurrent updates without synchronisation + Guaranteed Convergence

Key Properties:

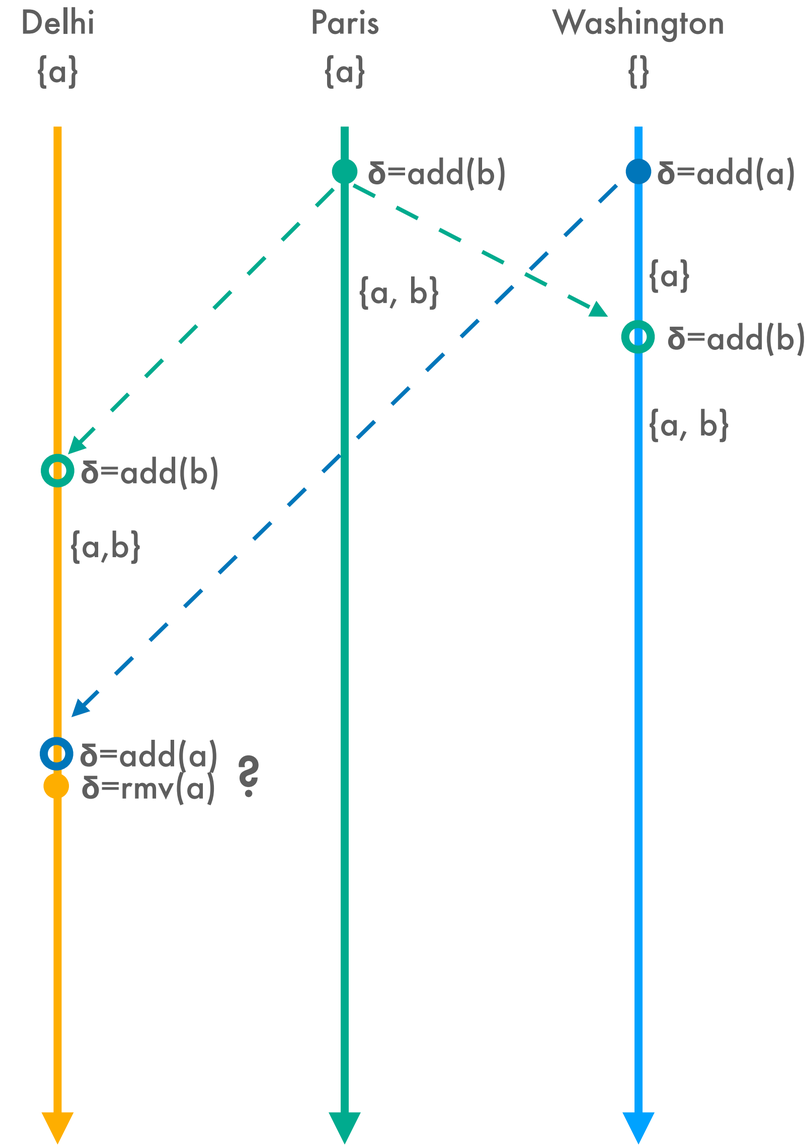
- Commutative, Associative and Idempotent Operations
- Mergeable States
- No Locks or consensus for updates

Implementation:

- **Effect (δ):** Takes the state of the CRDT and produces a new state
- **Merge:** Takes two CRDT states and **deterministically** produces a new state

Example: Add-Wins Set

- Effects: Add, Remove
- Merge: Between concurrent add and remove, add wins.



Aside: Rich Data Types

Conflict Free Replicated Datatypes (CRDT)

Concurrent updates without synchronisation + Guaranteed Convergence

Key Properties:

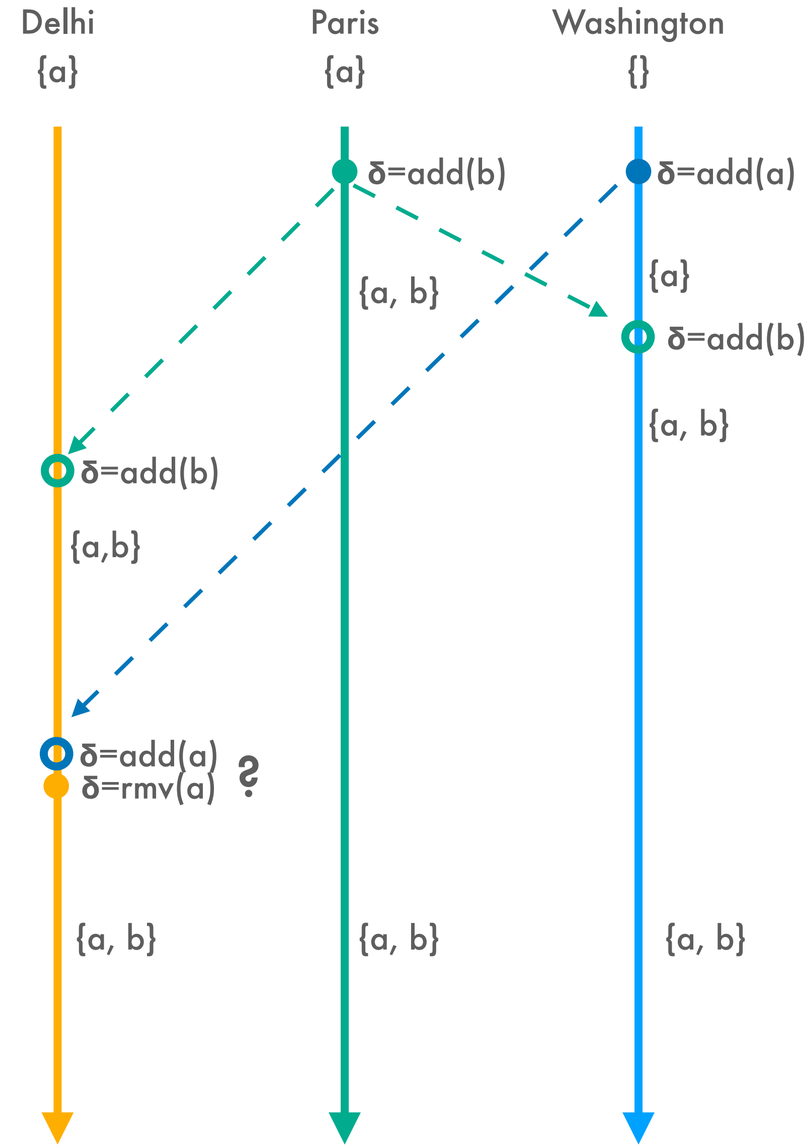
- Commutative, Associative and Idempotent Operations
- Mergeable States
- No Locks or consensus for updates

Implementation:

- **Effect (δ):** Takes the state of the CRDT and produces a new state
- **Merge:** Takes two CRDT states and **deterministically** produces a new state

Example: Add-Wins Set

- Effects: Add, Remove
- Merge: Between concurrent add and remove, add wins.



Transactions

Transaction is a block of operations ≈ *Session Order*

- Begin (B), Update (U), Read, Commit (C), Abort (A)

Begins with a snapshot timestamp (st) and commits with a commit timestamp (ct)

Sees updates from other transactions ≈ *Visibility*

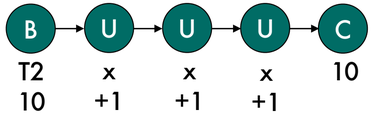
- $T_1 \prec_{vis} T_2 \Leftrightarrow T_1 . ct < T_2 . st$

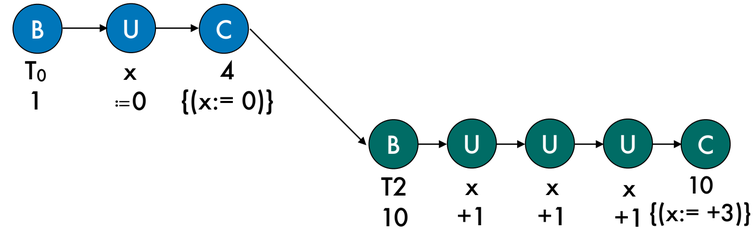
Reads in a transaction:

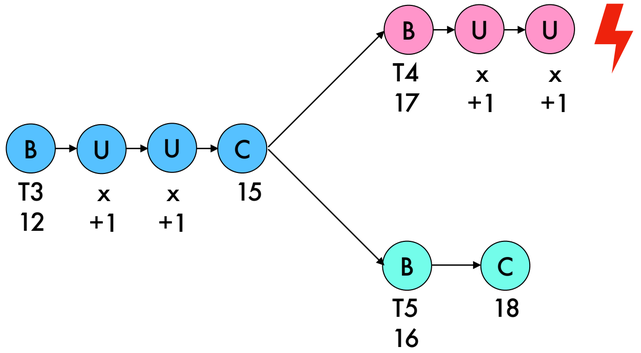
- read = snapshot ⊙ local updates

Crashes?

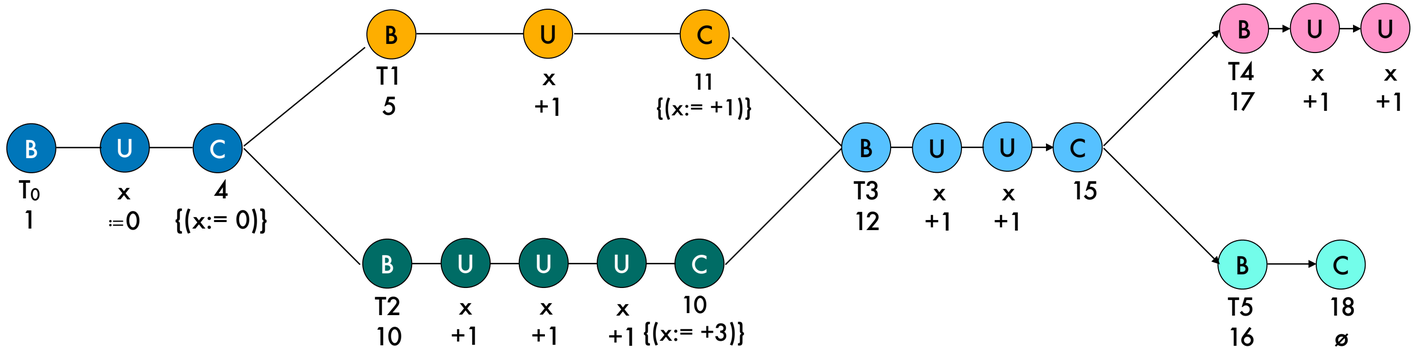
- Durable once committed, aborted otherwise







Traces - Histories of Transactions



Directed Acyclic Graph (DAG)

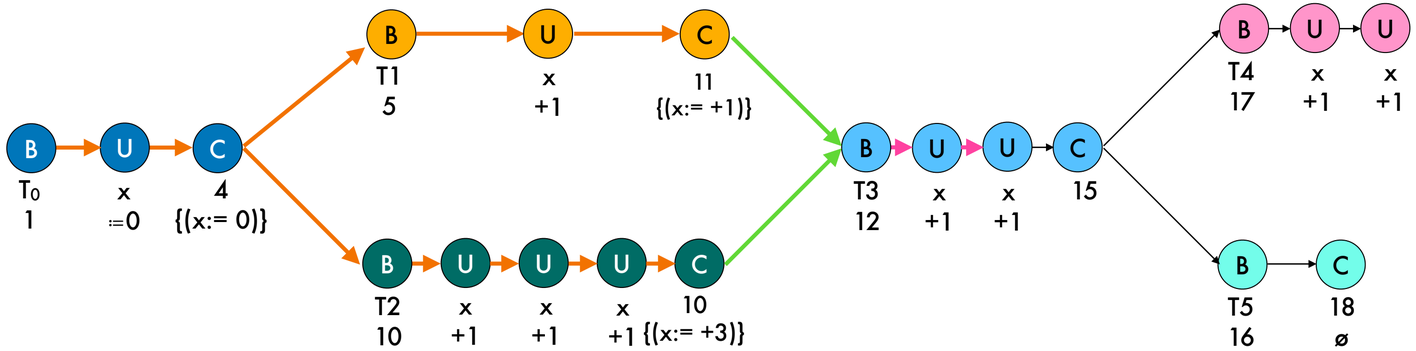
Vertices = Transaction events

Edges = Execution order

Some Metadata

Transaction IDs, Snapshot Timestamps, Datatypes, Effects etc.

Traces - Correctness



Correctness Criterion [Cerone et. al.] :

- Trace is Transactionally Causally Consistent (TCC)
- TCC = Ext ∧ Int ∧ TransVis

Why TCC?

TCC is the strongest consistency level which guarantees availability under partitions [CAP]

→ **Ext**: Snapshot Reads: First reads contains all previous transactions.

- Read rule: First read is a snapshot

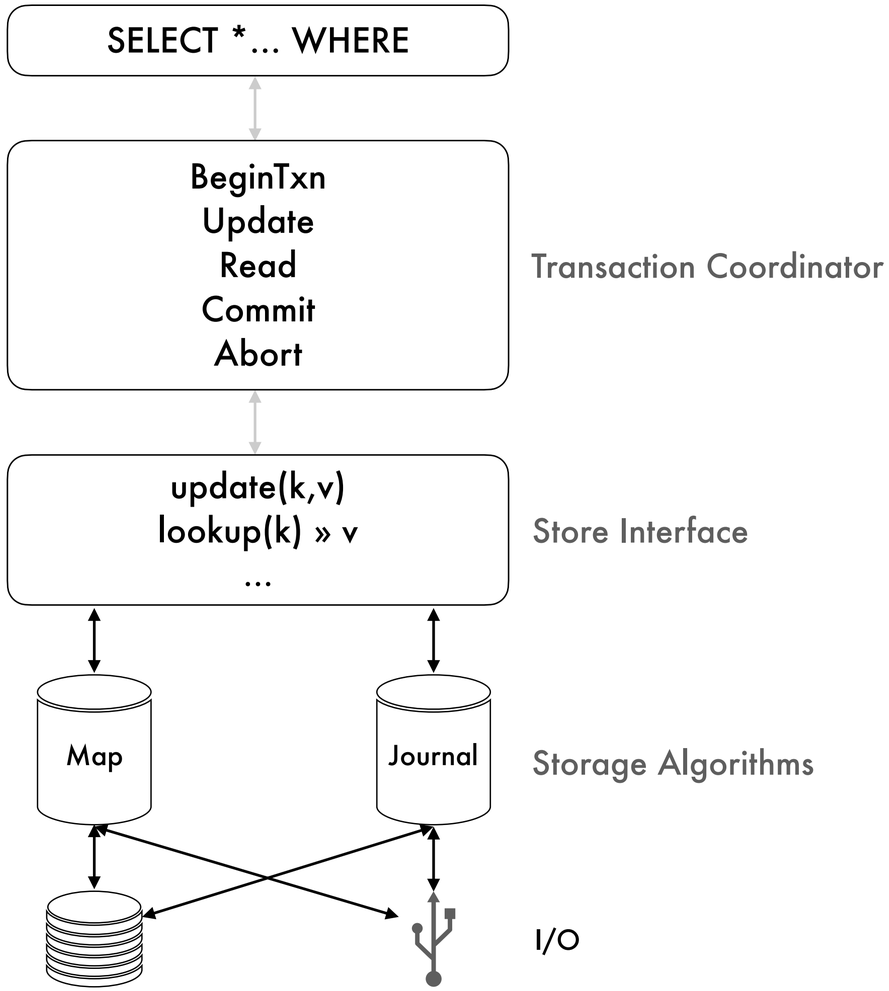
→ **Int**: Read-My-Writes

- Read rule: later reads apply local effects

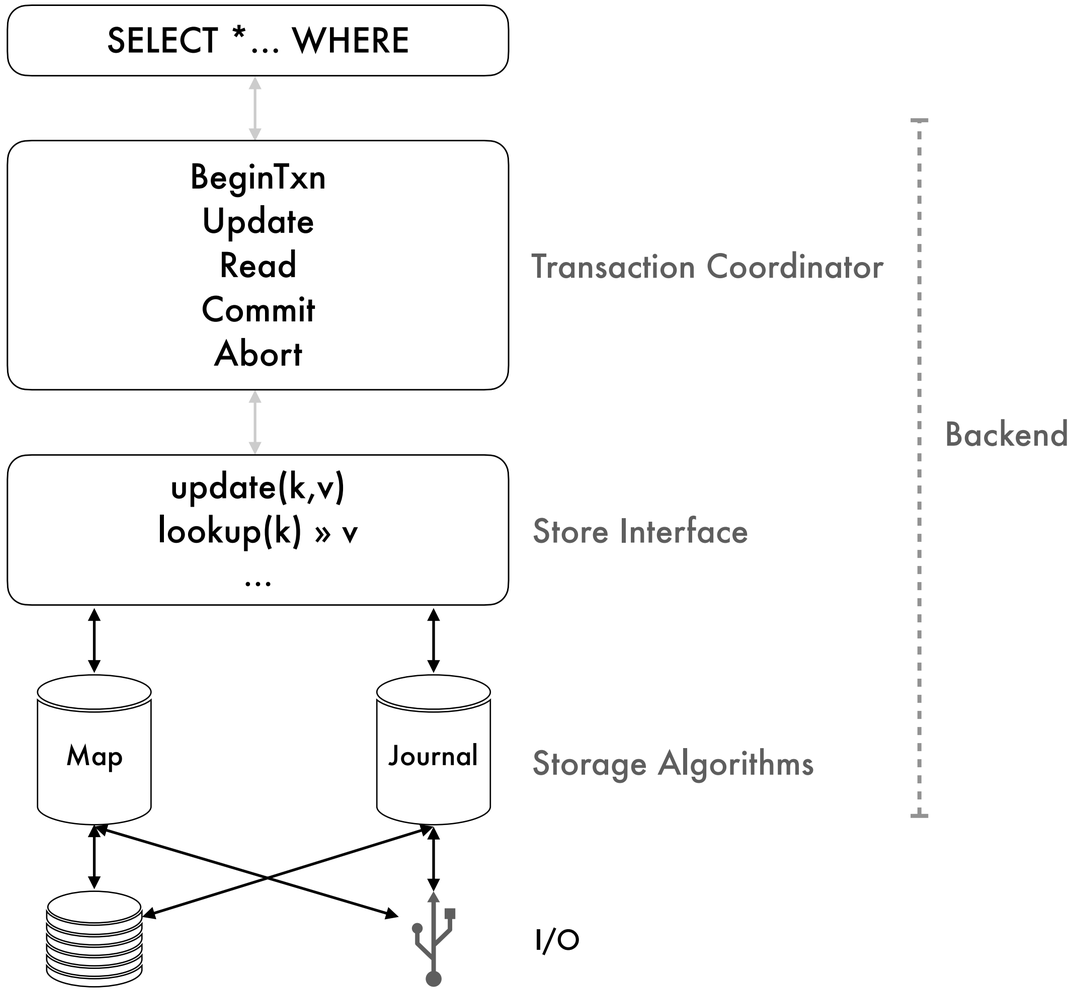
→ **TransVis**: Visibility is transitive

- By construction of a well formed trace

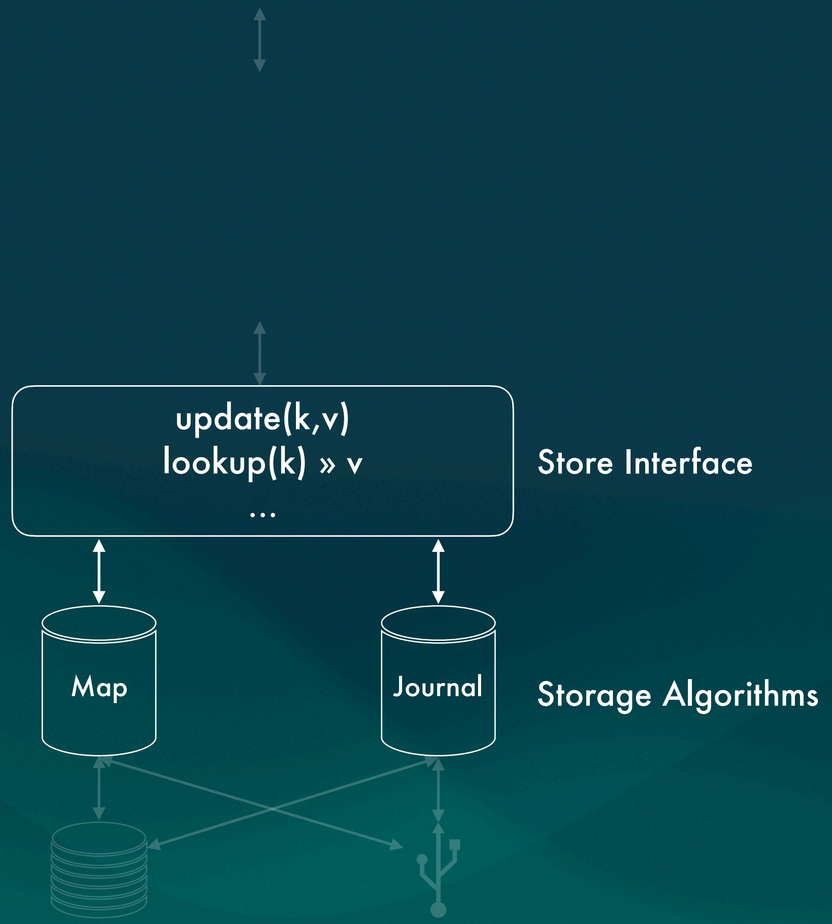
Database Backend?



Database Backend?



Stores

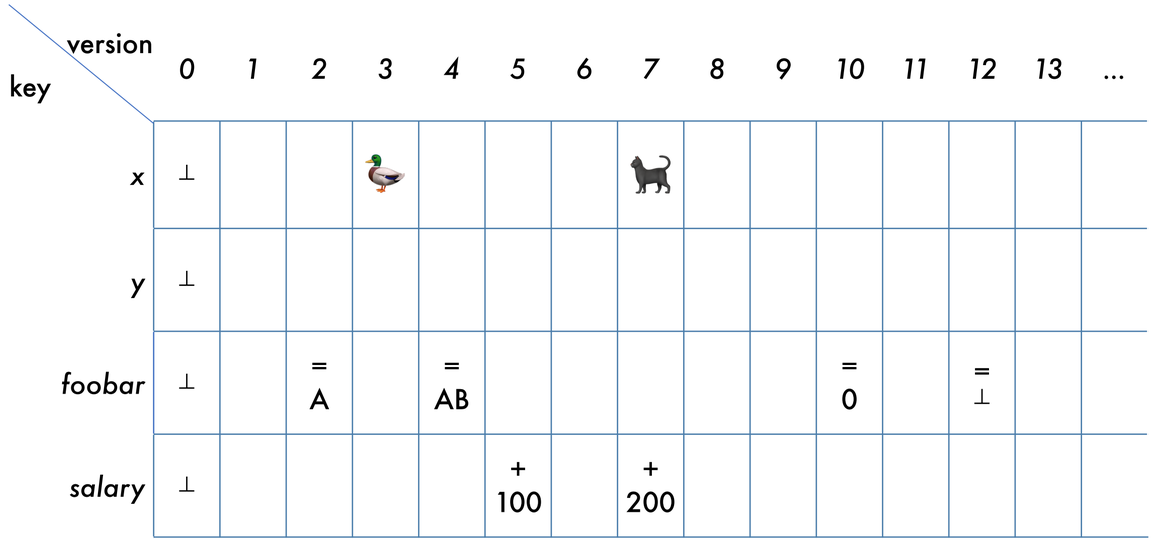

update(k,v)
lookup(k) » v
...
Store Interface
Map
Journal
Storage Algorithms

Store

Abstraction: Shared memory object (σ) that returns whatever is written to it.

Approach:

- [Key, Version] ⟶ *Value*$_\perp$
- Lookup(σ, k, t) ⟶ $\delta_\perp$



key \ version	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...
x	⊥			🦆				🐈‍⬛							
y	⊥														
foobar	⊥		= A		= AB						= 0		= ⊥		
salary	⊥					+ 100		+ 200							

Variants:

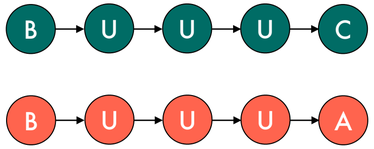
Trivial:

- Journal
- Map

Non-trivial compositions:

- Write Ahead Log (Map + Journal)
- LSM-Tree (Tree of Maps + Journal Blocks = WAL Memtables)

Store Interface



$doBegin(\sigma, \tau, st) \longmapsto \sigma'$ — B B — Start a transaction, add the transaction record to the store

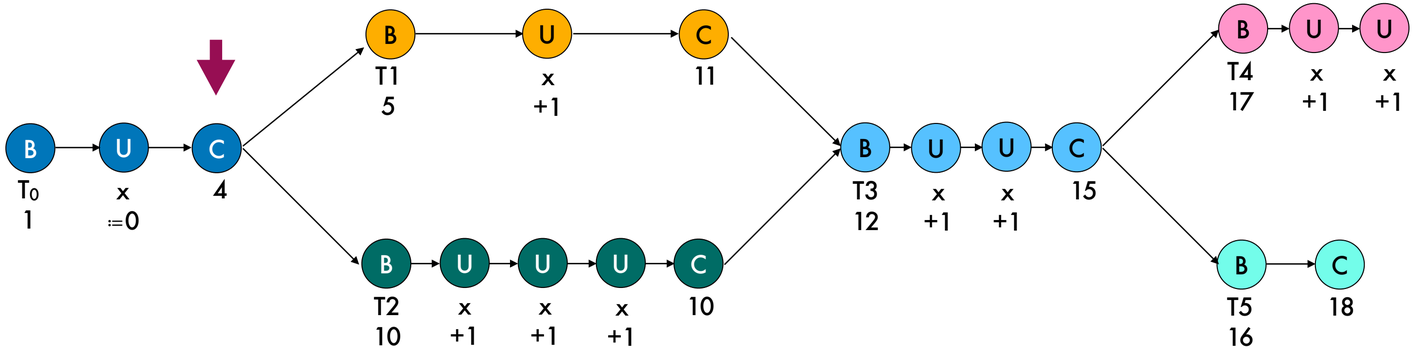
$lookup(\sigma, k, st) \longmapsto \sigma$ — ? — Fetch the value of key '*k*' at timestamp '*st*'

$doUpdate(\sigma, \tau, k, st, \delta) \longmapsto \sigma'$ — U U — Update the value of key 'k' by applying the effect 'delta', within transaction 'tau' at timestamp 'st'.

$doAbort(\sigma, \tau, st) \longmapsto \sigma'$ — A — Abort transaction 'tau' which began at timestamp 'st'

$doCommit(\sigma, \tau, st, \mathscr{B}, ct) \longmapsto \sigma'$ — C — Commit transaction 'tau', which began at 'st', with timestamp 'ct' and make effects from 'B' visible

Journal Store

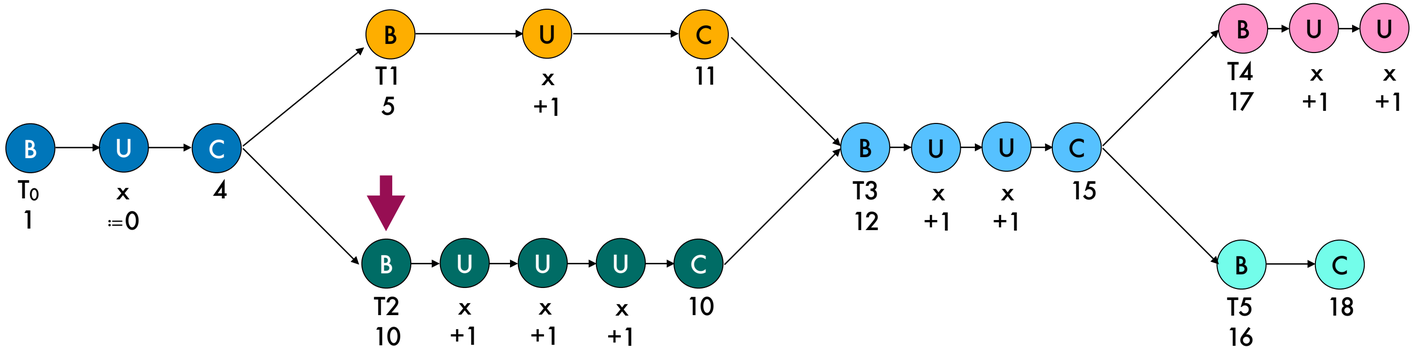


LSN	1	2	3
type	B	U	C
Txn ID	T0	T0	T0
st	1		
key		x	
effect		:= 0	
ct			4
poststate(x)	⊥	0	0

Every store operation is recorded in the journal as they occur.

Writing is fast, Reading is slow.

Journal Store

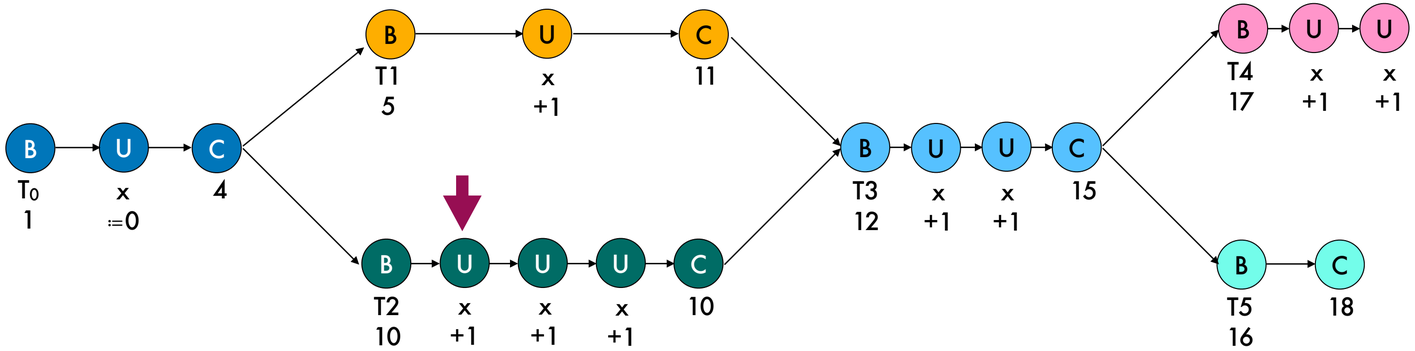


LSN	1	2	3	4
type	B	U	C	B
Txn ID	T0	T0	T0	T2
st	1			10
key		x		
effect		:= 0		
ct			4	
poststate(x)	⊥	0	0	0

Every store operation is recorded in the journal as they occur.

Writing is fast, Reading is slow.

Journal Store

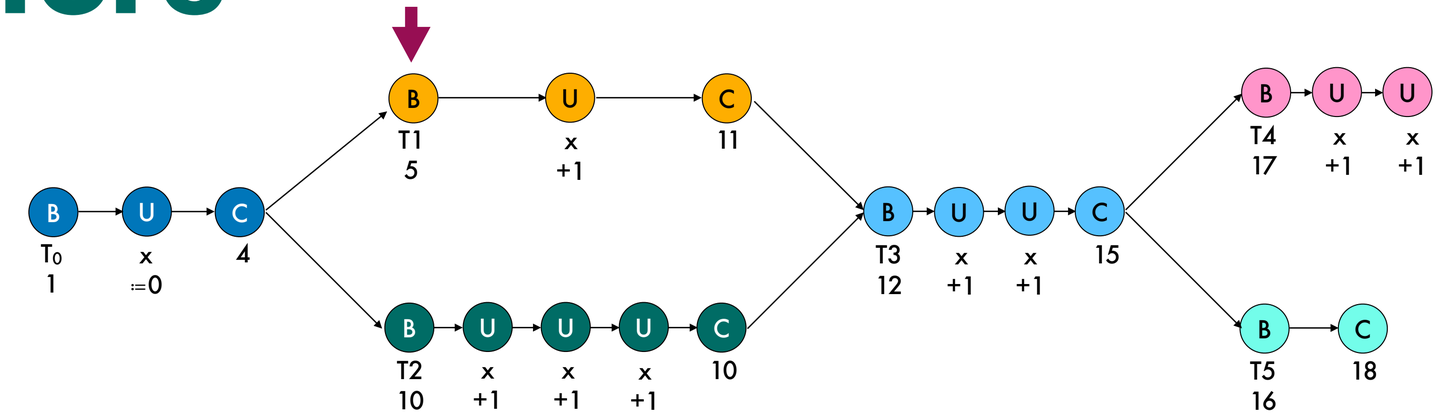


LSN	1	2	3	4	5
type	B	U	C	B	U
Txn ID	T0	T0	T0	T2	T2
st	1			10	
key		x			x
effect		:= 0			+1
ct			4		
poststate(x)	⊥	0	0	0	1

Every store operation is recorded in the journal as they occur.

Writing is fast, Reading is slow.

Journal Store

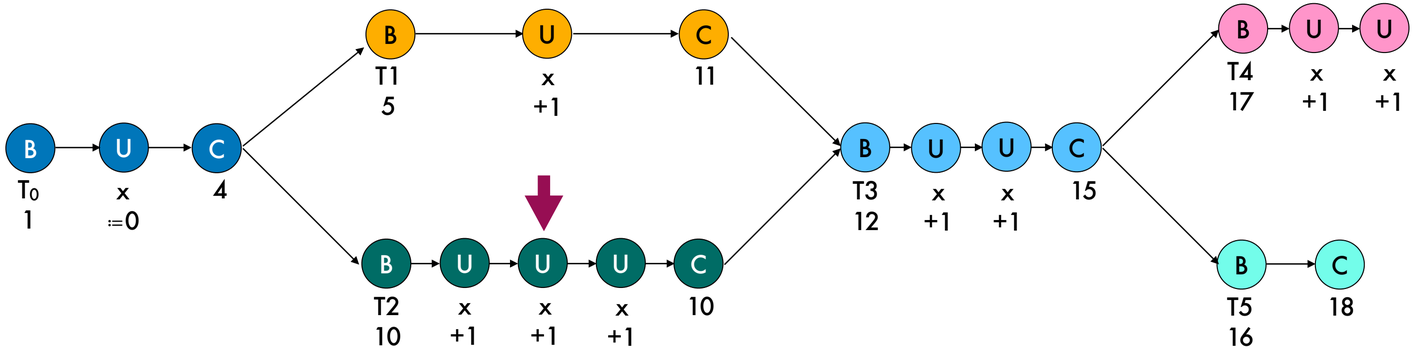


LSN	1	2	3	4	*5*	6
type	B	U	C	B	U	B
Txn ID	T0	T0	T0	T2	T2	T1
st	1			10		*5*
key		x			x	
effect		:= 0			+1	
ct			4			
poststate(x)	⊥	0	0	0	1	1

Every store operation is recorded in the journal as they occur.

Writing is fast, Reading is slow.

Journal Store

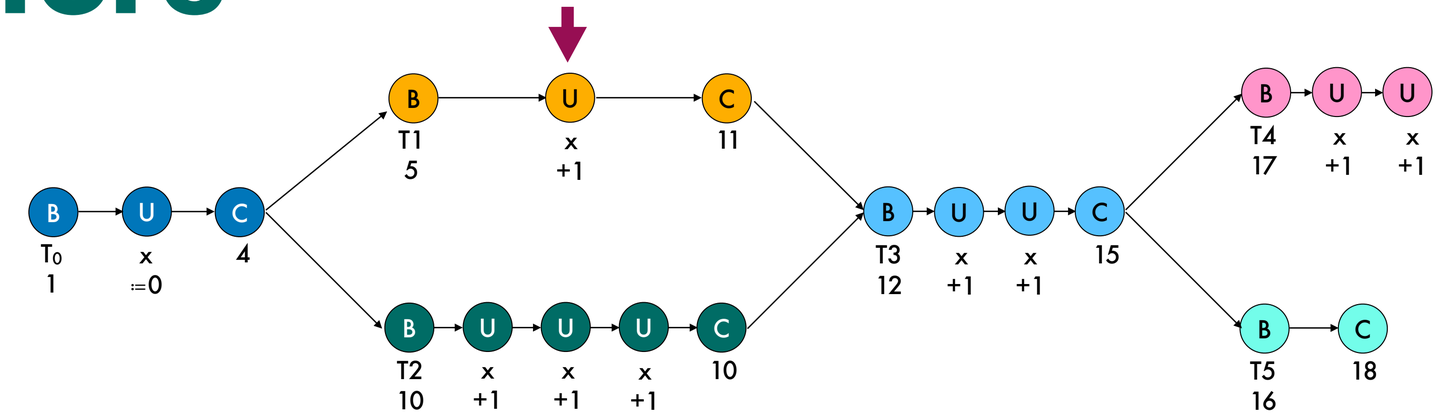


LSN	1	2	3	4	5	6	7
type	B	U	C	B	U	B	U
Txn ID	T0	T0	T0	T2	T2	T1	T2
st	1			10		5	
key		x			x		x
effect		:= 0			+1		+1
ct			4				
poststate(x)	⊥	0	0	0	1	1	2

Every store operation is recorded in the journal as they occur.

Writing is fast, Reading is slow.

Journal Store

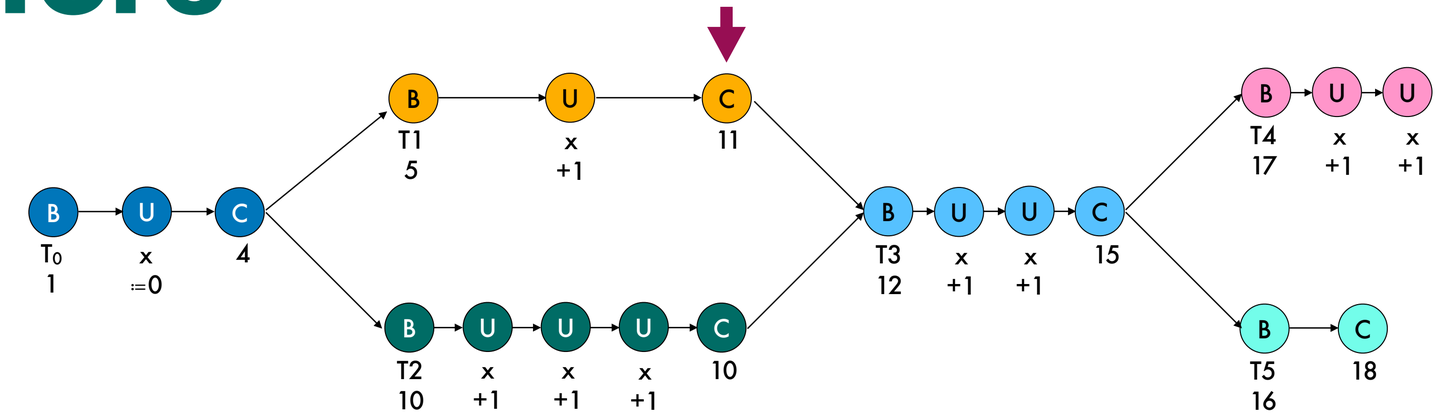


LSN	1	2	3	4	*5*	6	7	8
type	B	U	C	B	U	B	U	U
Txn ID	T0	T0	T0	T2	T2	T1	T2	T1
st	1			10		*5*		
key		x			x		x	x
effect		:= 0			+1		+1	+1
ct			4					
poststate(x)	⊥	0	0	0	1	1	2	1

Every store operation is recorded in the journal as they occur.

Writing is fast, Reading is slow.

Journal Store

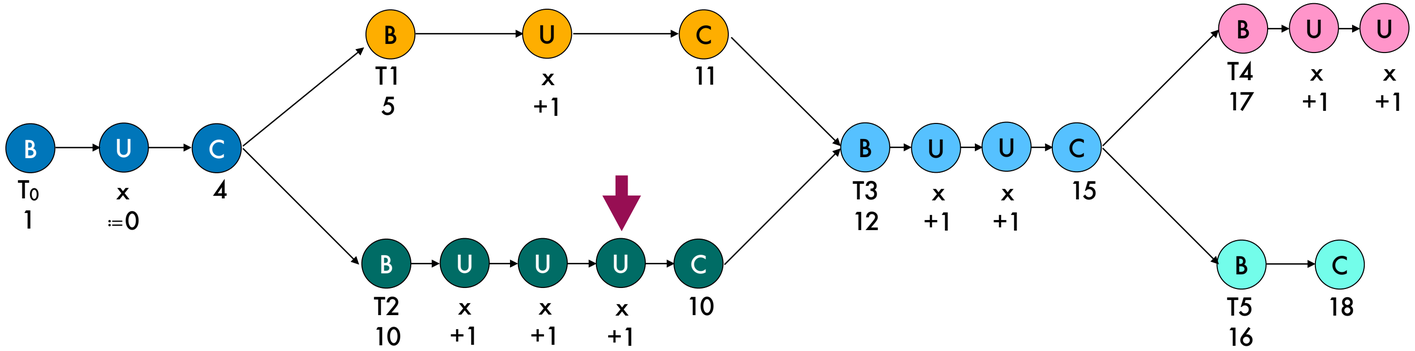


LSN	1	2	3	4	5	6	7	8	9
type	B	U	C	B	U	B	U	U	C
Txn ID	T0	T0	T0	T2	T2	T1	T2	T1	T1
st	1			10		*5*			
key		x			x		x	x	
effect		:= 0			+1		+1	+1	
ct			4						11
poststate(x)	⊥	0	0	0	1	1	2	1	1

Every store operation is recorded in the journal as they occur.

Writing is fast, Reading is slow.

Journal Store

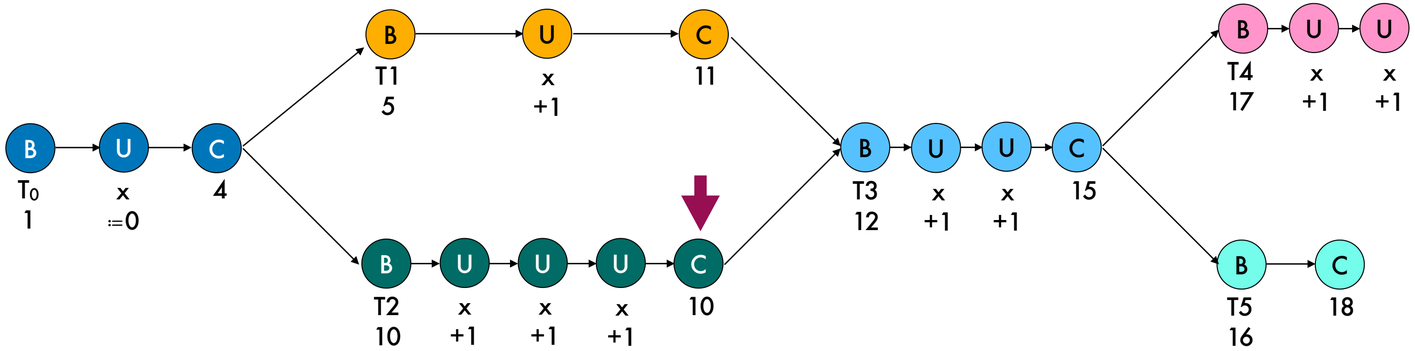


LSN	1	2	3	4	*5*	6	7	8	9	10
type	B	U	C	B	U	B	U	U	C	U
Txn ID	T0	T0	T0	T2	T2	T1	T2	T1	T1	T2
st	1			10		*5*				
key		x			x		x	x		x
effect		:= 0			+1		+1	+1		+1
ct			4						11	
poststate(x)	⊥	0	0	0	1	1	2	1	1	3

Every store operation is recorded in the journal as they occur.

Writing is fast, Reading is slow.

Journal Store

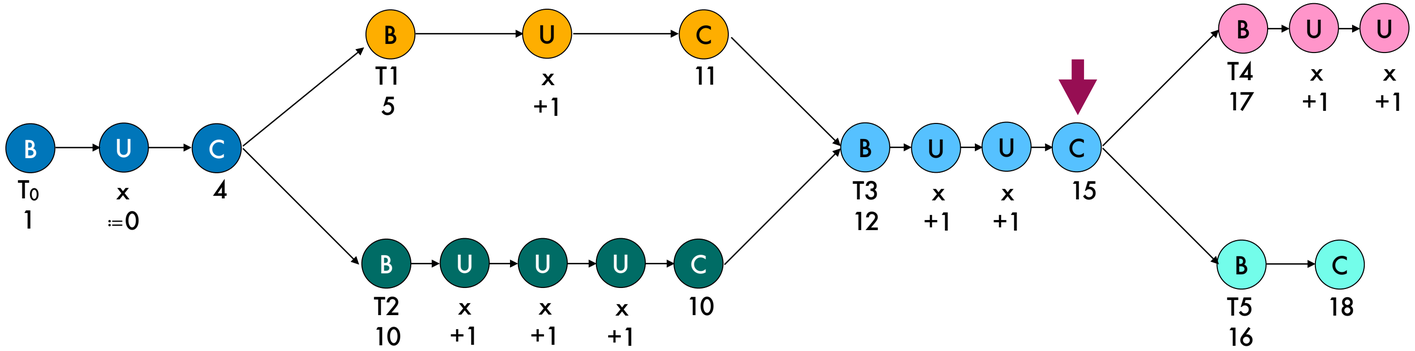


LSN	1	2	3	4	*5*	6	7	8	9	10	11
type	B	U	C	B	U	B	U	U	C	U	C
Txn ID	T0	T0	T0	T2	T2	T1	T2	T1	T1	T2	T2
st	1			10		*5*					
key		x			x		x	x		x	
effect		:= 0			+1		+1	+1		+1	
ct			4						11		10
poststate(x)	⊥	0	0	0	1	1	2	1	1	3	3

Every store operation is recorded in the journal as they occur.

Writing is fast, Reading is slow.

Journal Store

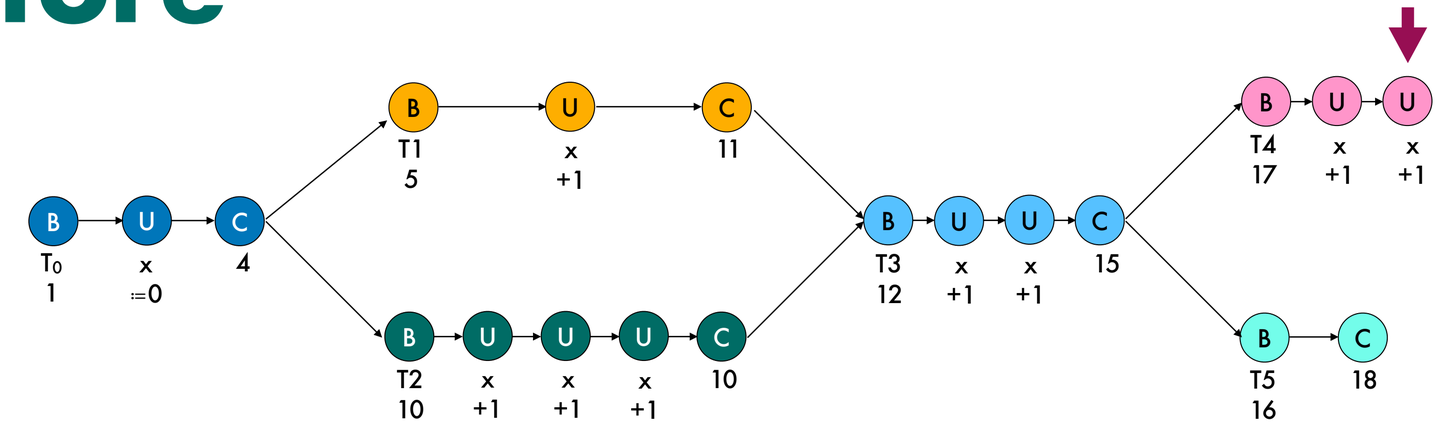


LSN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
type	B	U	C	B	U	B	U	U	C	U	C	B	U	U	C
Txn ID	T0	T0	T0	T2	T2	T1	T2	T1	T1	T2	T2	T3	T3	T3	T3
st	1			10		*5*						12			
key		x			x		x	x		x			x	x	
effect		:= 0			+1		+1	+1		+1			+1	+1	
ct			4						11		10				15
poststate(x)	⊥	0	0	0	1	1	2	1	1	3	3	4	5	6	6

Every store operation is recorded in the journal as they occur.

Writing is fast, Reading is slow.

Journal Store

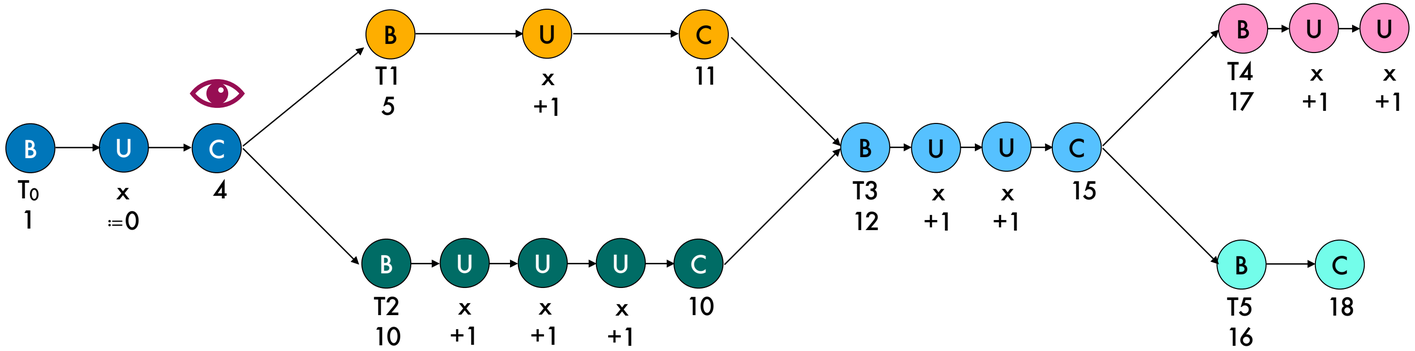


LSN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
type	B	U	C	B	U	B	U	U	C	U	C	B	U	U	C	B	U	B	C	U	
Txn ID	T0	T0	T0	T2	T2	T1	T2	T1	T1	T2	T2	T3	T3	T3	T3	T4	T4	T5	T5	T4	
st	1			10		5						12				17		16			
key		x			x		x	x		x			x	x			x			x	
effect		:= 0			+1		+1	+1		+1			+1	+1			+1			+1	
ct			4						11		10				15				18		
poststate(x)	⊥	0	0	0	1	1	2	1	1	3	3	4	5	6	6	6	7	6	6	7	

Every store operation is recorded in the journal as they occur.

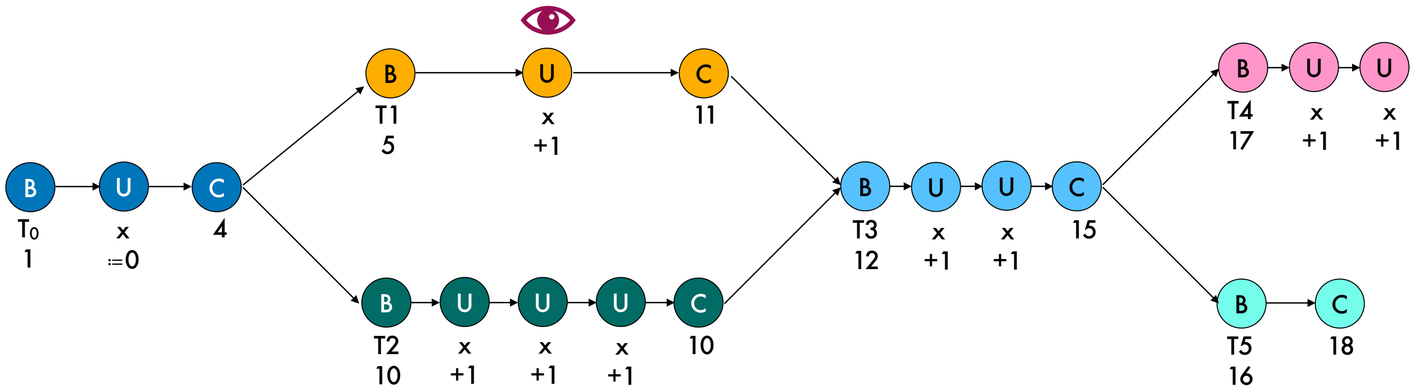
Writing is fast, Reading is slow.

Journal Store



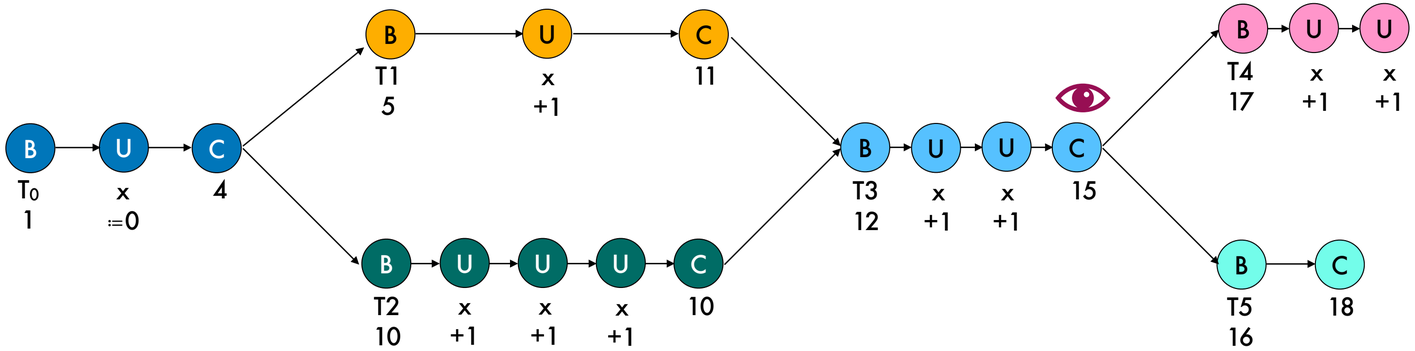
LSN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
type	B	U	C	B	U	B	U	U	C	U	C	B	U	U	C	B	U	B	C	U	
Txn ID	T0	T0	T0	T2	T2	T1	T2	T1	T1	T2	T2	T3	T3	T3	T3	T4	T4	T5	T5	T4	
st	1			10		5						12				17		16			
key		x			x		x	x		x			x	x			x			x	
effect		:= 0			+1		+1	+1		+1			+1	+1			+1			+1	
ct			4						11		10				15				18		
poststate(x)	⊥	0	0	0	1	1	2	1	1	3	3	4	5	6	6	6	7	6	6	7	

Journal Store



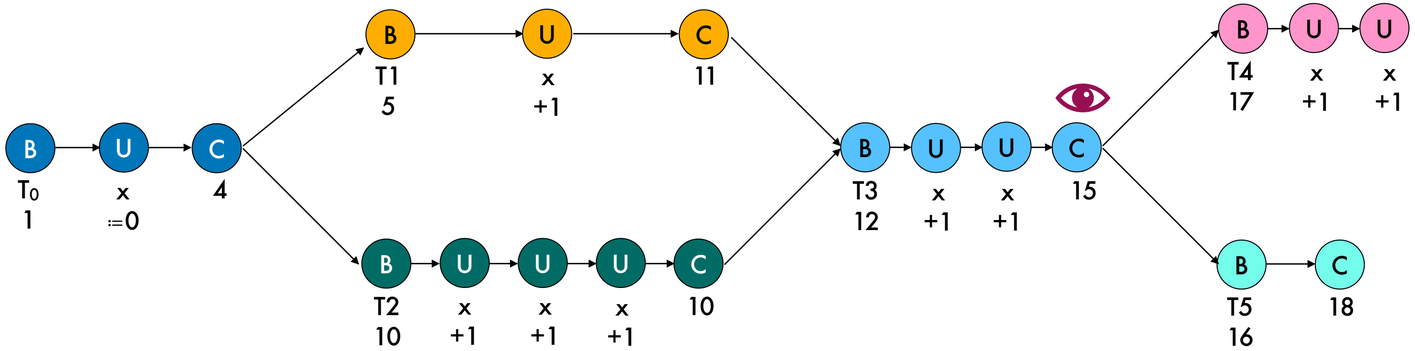
LSN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
type	B	U	C	B	U	B	U	U	C	U	C	B	U	U	C	B	U	B	C	U	
Txn ID	T0	T0	T0	T2	T2	T1	T2	T1	T1	T2	T2	T3	T3	T3	T3	T4	T4	T5	T5	T4	
st	1			10		5						12				17		16			
key		x			x		x	x		x			x	x			x			x	
effect		:= 0			+1		+1	+1		+1			+1	+1			+1			+1	
ct			4						11		10				15				18		
poststate(x)	⊥	0	0	0	1	1	2	1	1	3	3	4	5	6	6	6	7	6	6	7	

Journal Store



LSN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
type	B	U	C	B	U	B	U	U	C	U	C	B	U	U	C	B	U	B	C	U	
Txn ID	T0	T0	T0	T2	T2	T1	T2	T1	T1	T2	T2	T3	T3	T3	T3	T4	T4	T5	T5	T4	
st	1			10		5						12				17		16			
key		x			x		x	x		x			x	x			x			x	
effect		:= 0			+1		+1	+1		+1			+1	+1			+1			+1	
ct			4						11		10				15				18		
poststate(x)	⊥	0	0	0	1	1	2	1	1	3	3	4	5	6	6	6	7	6	6	7	

Journal Store

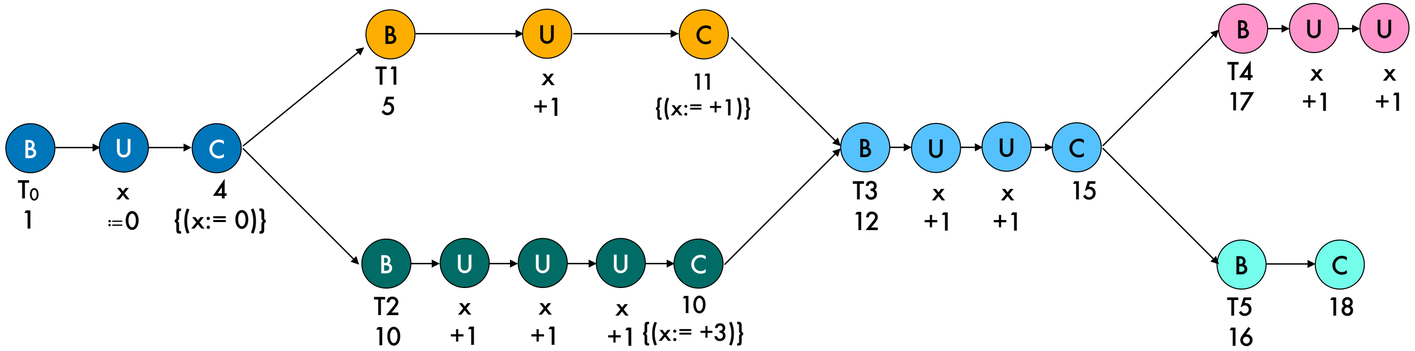


LSN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
type	B	U	C	B	U	B	U	U	C	U	C	B	U	U	C	B	U	B	C	U	
Txn ID	T0	T0	T0	T2	T2	T1	T2	T1	T1	T2	T2	T3	T3	T3	T3	T4	T4	T5	T5	T4	
st	1			10		5						12				17		16			
key		x			x		x	x		x			x	x			x			x	
effect		:= 0			+1		+1	+1		+1			+1	+1			+1			+1	
ct			4						11		10				15				18		
poststate(x)	⊥	0	0	0	1	1	2	1	1	3	3	4	5	6	6	6	7	6	6	7	

$$
\begin{aligned}
\text{doBegin}(\sigma_J, \tau, \text{st}) &= \sigma_J \triangleright (\text{beginTxn}, \tau, \text{st}) \\
\text{lookup}(\sigma_J, k, \text{st}) &= \mathit{poststate}_{\sigma_J}(r, k) \text{ where } r = (\text{beginTxn}, \_, \text{st}) \\
\text{doUpdate}(\sigma_J, \tau, k, \text{st}, \delta) &= \sigma_J \triangleright (\text{update}, \tau, k, \delta) \\
\text{doAbort}(\sigma_J, \tau, \text{st}) &= \sigma_J \triangleright (\text{abort}, \tau) \\
\text{doCommit}(\sigma, \tau, \text{st}, \mathcal{B}, \text{ct}) &= \sigma_J \triangleright (\text{commit}, \tau, \text{ct})
\end{aligned}
$$

$$
\mathit{poststate}_{\sigma_J}(r, k) \stackrel{\text{def}}{=} \begin{cases}
\text{merge}\ (\ \{\ \mathit{poststate}_{\sigma_J}(r', k) : r' \in \mathit{parent}_J(r)\ \}\ ) & \\
\qquad\qquad\qquad\qquad\qquad\qquad & \text{if } r = (\text{beginTxn}, \_, \_) \\
\mathit{poststate}_{\sigma_J}(\mathit{parent}_J(r), k) & \text{if } r = (\text{update}, \_, k', \_) \wedge k \neq k' \\
\mathit{poststate}_{\sigma_J}(\mathit{parent}_J(r), k) \odot \delta & \text{if } r = (\text{update}, \_, k, \delta) \\
\mathit{poststate}_{\sigma_J}(\mathit{parent}_J(r), k) & \text{if } r = (\text{commit}, \_, \_) \vee r = (\text{abort}, \_)
\end{cases}
$$

Map Store



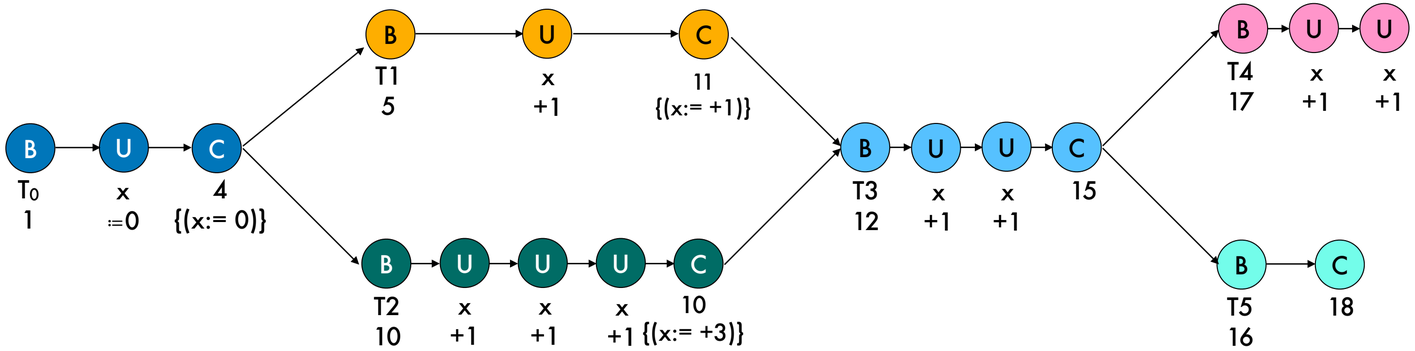
key	x	x	x	x
version	4	11	10	15
dependence	-	T_0 5 > 4	T_0 10 > 4	$\{T_1,T_2\}$ 12 > 11, 10
Effect	:= 0	:= 1	:= 3	:= 6

Only commits are recorded in the map, in no particular logical order.

Writing is slow, Reading is fast.

Effects are buffered by the transactions until commit i.e. Correctness only when transactional.

Map Store



key	x	x	x	x
version	4	11	10	15
dependence	-	T_0 5 > 4	T_0 10 > 4	$\{T_1, T_2\}$ 12 > 11, 10
Effect	:= 0	:= 1	:= 3	:= 6

$\sigma_M \in (Key \times Timestamp \to Timestamp \times Assign) \subseteq \Sigma$, initially, $\sigma_M[k,t] = \bot$ for all k, t

$$\mathsf{lookup}(\sigma_M, k, \mathsf{st}) \quad = \quad \mathsf{merge}(\max_{\prec_M}(\{\sigma_M[k,w] \neq \bot : w < \mathsf{st}\}).\delta)$$

$$\mathsf{doCommit}(\sigma_M, \_, \mathsf{st}, \mathcal{B}, \mathsf{ct})[k,t] \quad = \quad \begin{cases} (\mathsf{st}, \mathcal{B}[k]) & \text{if } \mathcal{B}[k] \neq \bot \wedge t = \mathsf{ct} \\ \sigma_M[k,t] & \text{otherwise} \end{cases}$$

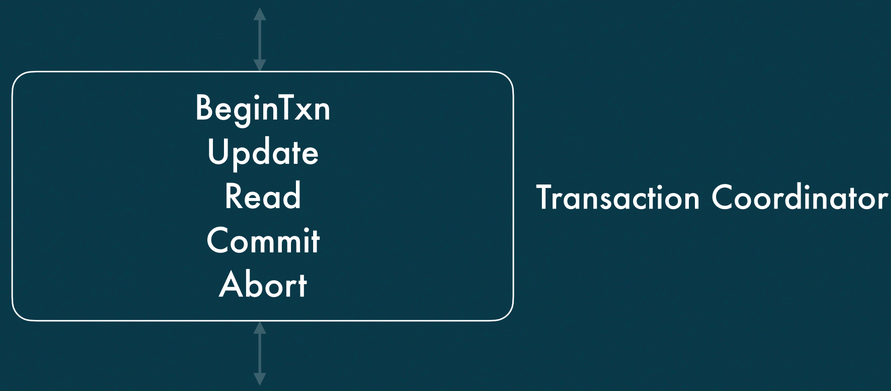
doBegin, doUpdate, doAbort: no-ops

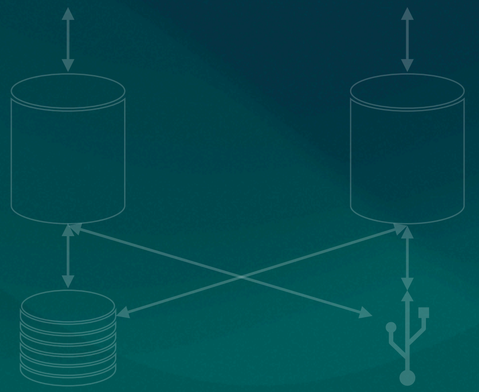
Only commits are recorded in the map, in no particular logical order.

Writing is slow, Reading is fast.

Effects are buffered by the transactions until commit.

Transactions





Recap

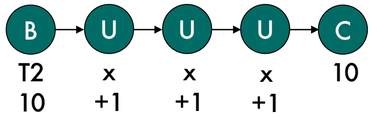
Block of Operations ≈ *Session Order*

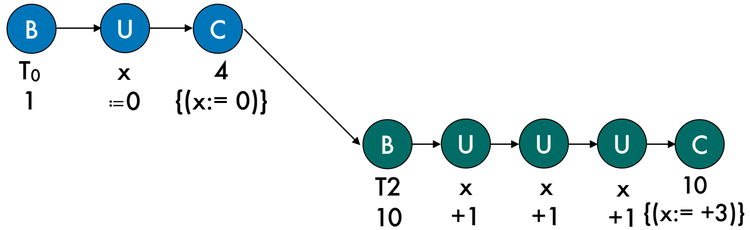
Has a st and a ct (if commits)

Sees updates from other transactions ≈ *Visibility*

Reads values from other transactions and self

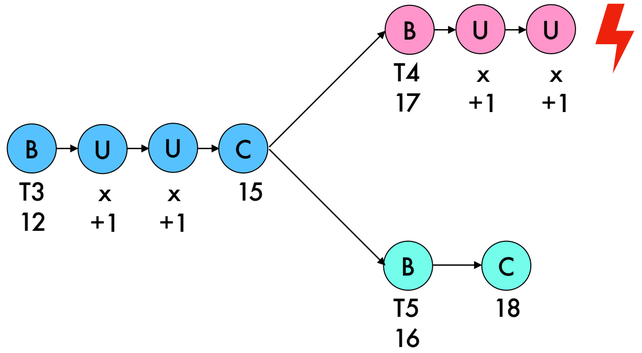
Durable once committed, aborted otherwise



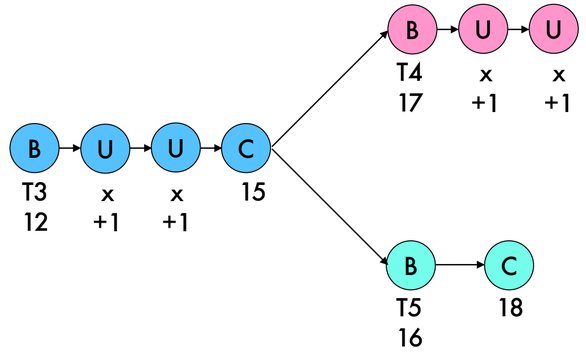


API:

$beginTxn(st)$	B
$read(k) \rightarrow value$	
$update(k, st, \delta)$	U
$abort()$	
$commit(ct)$	C



Transactions ↔ Stores



Transaction event…	…calls store method
beginTxn (st)	*doBegin (σ, τ, st)*
*update (k, **δ**)*	*doUpdate (σ, τ, k, st, **δ**)*
read (k) ⟶ v	*lookup (σ, k, st) ⟶ **δ***
abort ()	*doAbort (σ, τ, st)*
commit (ct)	*doCommit (σ, τ, st, Δ, ct)*

σ	store
τ	Transaction ID
st	snapshot timestamp
k	key
δ	effect
Δ	buffered effects
ct	commit timestamp

Transaction Semantics

BeginTxn

$$\frac{\forall T \in \mathcal{X}_\mathrm{a} \uplus \mathcal{X}_\mathrm{c} \uplus \mathcal{X}_\mathrm{r}, T.\tau \neq \tau \qquad \sigma' = \mathsf{doBegin}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}'_\mathrm{r} = \mathcal{X}_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \varnothing, \varnothing, \varnothing, \_)\}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{beginTxn}(\mathsf{st})} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Update

$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \\ \delta \neq \bot \qquad \mathcal{B}' = \mathcal{B}[k \leftarrow \mathcal{B}[k] \odot \delta] \qquad \sigma' = \mathsf{doUpdate}(\sigma, \tau, k, \mathsf{st}, \delta) \\ \mathcal{R}' = (\text{ if } k \notin \mathcal{I} \text{ then } \mathcal{R} \text{ else } \mathcal{R}[k \leftarrow \mathcal{R}[k] \odot \delta] \,) \qquad \mathcal{X}'_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}', \_)\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{update}(k,\delta)} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

SnapshotRead

$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad k \notin \mathcal{I} \qquad \mathsf{lookup}(\sigma, k, \mathsf{st}) = \delta \\ \delta' = \delta \odot \mathcal{B}[k] \qquad \mathcal{I}' = \mathcal{I} \cup \{k\} \qquad \mathcal{R}' = \mathcal{R}[k \leftarrow \delta'] \qquad \mathcal{X}'_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}', \mathcal{R}', \mathcal{B}, \_)\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \underset{\tau}{\rightarrow} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Read

$$\frac{\mathcal{X}_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \_, \mathcal{I}, \mathcal{R}, \_, \_)\} \qquad k \in \mathcal{I} \qquad \mathcal{R}[k] = \delta \qquad \delta \in \mathit{Assign}_\bot \qquad v = \delta(\bot)}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{read}(k) \rightarrow v} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r})}$$

Abort

$$\frac{\mathcal{X}_\mathrm{r} = \mathcal{X}'_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \sigma' = \mathsf{doAbort}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}'_\mathrm{a} = \mathcal{X}_\mathrm{a} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{abort}()} (\sigma', \mathcal{X}'_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Commit

$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}'_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \forall T \in \mathcal{X}_\mathrm{c}, T.\mathsf{ct} \neq \mathsf{ct} \qquad \mathsf{st} \leq \mathsf{ct} \\ \mathsf{noInversion}(\mathsf{ct}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \qquad \sigma' = \mathsf{doCommit}(\sigma, \tau, \mathsf{st}, \mathcal{B}, \mathsf{ct}) \qquad \mathcal{X}'_\mathrm{c} = \mathcal{X}_\mathrm{c} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \mathsf{ct})\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{commit}(\mathsf{ct})} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}'_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Transaction Semantics

BeginTxn

$$\frac{\forall T \in \mathcal{X}_a \uplus \mathcal{X}_c \uplus \mathcal{X}_r, T.\tau \neq \tau \qquad \sigma' = \mathsf{doBegin}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}_r' = \mathcal{X}_r \uplus \{(\tau, \mathsf{st}, \varnothing, \varnothing, \varnothing, \_)\}}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \xrightarrow[\tau]{\mathsf{beginTxn(st)}} (\sigma', \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r')}$$

Aborted, Committed, Running

Update

$$\frac{\mathcal{X}_r = \mathcal{X}_r'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \delta \neq \bot \qquad \mathcal{B}' = \mathcal{B}[k \leftarrow \mathcal{B}[k] \odot \delta] \qquad \sigma' = \mathsf{doUpdate}(\sigma, \tau, k, \mathsf{st}, \delta) \qquad \mathcal{R}' = (\text{ if } k \notin \mathcal{I} \text{ then } \mathcal{R} \text{ else } \mathcal{R}[k \leftarrow \mathcal{R}[k] \odot \delta] \ ) \qquad \mathcal{X}_r' = \mathcal{X}_r'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}', \_)\}}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \xrightarrow[\tau]{\mathsf{update}(k,\delta)} (\sigma', \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r')}$$

SnapshotRead

$$\frac{\mathcal{X}_r = \mathcal{X}_r'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad k \notin \mathcal{I} \qquad \mathsf{lookup}(\sigma, k, \mathsf{st}) = \delta \qquad \delta' = \delta \odot \mathcal{B}[k] \qquad \mathcal{I}' = \mathcal{I} \cup \{k\} \qquad \mathcal{R}' = \mathcal{R}[k \leftarrow \delta'] \qquad \mathcal{X}_r' = \mathcal{X}_r'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}', \mathcal{R}', \mathcal{B}, \_)\}}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \underset{\tau}{\rightarrow} (\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r')}$$

Read

$$\frac{\mathcal{X}_r = \mathcal{X}_r'' \uplus \{(\tau, \_, \mathcal{I}, \mathcal{R}, \_, \_)\} \qquad k \in \mathcal{I} \qquad \mathcal{R}[k] = \delta \qquad \delta \in \mathit{Assign}_\bot \qquad v = \delta(\bot)}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \xrightarrow[\tau]{\mathsf{read}(k) \rightarrow v} (\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r)}$$

Abort

$$\frac{\mathcal{X}_r = \mathcal{X}_r' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \sigma' = \mathsf{doAbort}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}_a' = \mathcal{X}_a \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\}}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \xrightarrow[\tau]{\mathsf{abort()}} (\sigma', \mathcal{X}_a', \mathcal{X}_c, \mathcal{X}_r')}$$

Commit

$$\frac{\mathcal{X}_r = \mathcal{X}_r' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \forall T \in \mathcal{X}_c, T.\mathsf{ct} \neq \mathsf{ct} \qquad \mathsf{st} \leq \mathsf{ct} \qquad \mathsf{noInversion}(\mathsf{ct}, \mathcal{X}_c, \mathcal{X}_r) \qquad \sigma' = \mathsf{doCommit}(\sigma, \tau, \mathsf{st}, \mathcal{B}, \mathsf{ct}) \qquad \mathcal{X}_c' = \mathcal{X}_c \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \mathsf{ct})\}}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \xrightarrow[\tau]{\mathsf{commit(ct)}} (\sigma', \mathcal{X}_a, \mathcal{X}_c', \mathcal{X}_r')}$$

Transaction Semantics

BeginTxn

$$\frac{\forall T \in \mathcal{X}_a \uplus \mathcal{X}_c \uplus \mathcal{X}_r, T.\tau \neq \tau \qquad \sigma' = \mathsf{doBegin}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}_r' = \mathcal{X}_r \uplus \{(\tau, \mathsf{st}, \varnothing, \varnothing, \varnothing, \_)\}}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \xrightarrow[\tau]{\mathsf{beginTxn}(\mathsf{st})} (\sigma', \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r')}$$

Aborted, Committed, Running

Update

$$\frac{\begin{array}{c}\mathcal{X}_r = \mathcal{X}_r'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \\ \delta \neq \bot \qquad \mathcal{B}' = \mathcal{B}[k \leftarrow \mathcal{B}[k] \odot \delta] \qquad \sigma' = \mathsf{doUpdate}(\sigma, \tau, k, \mathsf{st}, \delta) \\ \mathcal{R}' = (\text{ if } k \notin \mathcal{I} \text{ then } \mathcal{R} \text{ else } \mathcal{R}[k \leftarrow \mathcal{R}[k] \odot \delta] \,) \qquad \mathcal{X}_r' = \mathcal{X}_r'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}', \_)\}\end{array}}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \xrightarrow[\tau]{\mathsf{update}(k,\delta)} (\sigma', \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r')}$$

SnapshotRead

$$\frac{\begin{array}{c}\mathcal{X}_r = \mathcal{X}_r'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad k \notin \mathcal{I} \qquad \mathsf{lookup}(\sigma, k, \mathsf{st}) = \delta \\ \delta' = \delta \odot \mathcal{B}[k] \qquad \mathcal{I}' = \mathcal{I} \cup \{k\} \qquad \mathcal{R}' = \mathcal{R}[k \leftarrow \delta'] \qquad \mathcal{X}_r' = \mathcal{X}_r'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}', \mathcal{R}', \mathcal{B}, \_)\}\end{array}}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \underset{\tau}{\rightarrow} (\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r')}$$

Read

$$\frac{\mathcal{X}_r = \mathcal{X}_r'' \uplus \{(\tau, \_, \mathcal{I}, \mathcal{R}, \_, \_)\} \qquad k \in \mathcal{I} \qquad \mathcal{R}[k] = \delta \qquad \delta \in \mathit{Assign}_\bot \qquad v = \delta(\bot)}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \xrightarrow[\tau]{\mathsf{read}(k) \rightarrow v} (\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r)}$$

Abort

$$\frac{\mathcal{X}_r = \mathcal{X}_r' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \sigma' = \mathsf{doAbort}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}_a' = \mathcal{X}_a \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\}}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \xrightarrow[\tau]{\mathsf{abort}()} (\sigma', \mathcal{X}_a', \mathcal{X}_c, \mathcal{X}_r')}$$

Commit

$$\frac{\begin{array}{c}\mathcal{X}_r = \mathcal{X}_r' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \forall T \in \mathcal{X}_c, T.\mathsf{ct} \neq \mathsf{ct} \qquad \mathsf{st} \leq \mathsf{ct} \\ \mathsf{noInversion}(\mathsf{ct}, \mathcal{X}_c, \mathcal{X}_r) \qquad \sigma' = \mathsf{doCommit}(\sigma, \tau, \mathsf{st}, \mathcal{B}, \mathsf{ct}) \qquad \mathcal{X}_c' = \mathcal{X}_c \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \mathsf{ct})\}\end{array}}{(\sigma, \mathcal{X}_a, \mathcal{X}_c, \mathcal{X}_r) \xrightarrow[\tau]{\mathsf{commit}(\mathsf{ct})} (\sigma', \mathcal{X}_a, \mathcal{X}_c', \mathcal{X}_r')}$$

Transaction descriptor

Transaction Semantics

BeginTxn

$$\frac{\forall T \in \mathcal{X}_\mathrm{a} \uplus \mathcal{X}_\mathrm{c} \uplus \mathcal{X}_\mathrm{r}, T.\tau \neq \tau \qquad \sigma' = \mathsf{doBegin}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}'_\mathrm{r} = \mathcal{X}_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \varnothing, \varnothing, \varnothing, \_)\}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{beginTxn(st)}} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Update

$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \\ \delta \neq \bot \qquad \mathcal{B}' = \mathcal{B}[k \leftarrow \mathcal{B}[k] \odot \delta] \qquad \sigma' = \mathsf{doUpdate}(\sigma, \tau, k, \mathsf{st}, \delta) \\ \mathcal{R}' = (\text{ if } k \notin \mathcal{I} \text{ then } \mathcal{R} \text{ else } \mathcal{R}[k \leftarrow \mathcal{R}[k] \odot \delta] \,) \qquad \mathcal{X}'_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}', \_)\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{update}(k,\delta)} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

SnapshotRead

$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad k \notin \mathcal{I} \qquad \mathsf{lookup}(\sigma, k, \mathsf{st}) = \delta \\ \delta' = \delta \odot \mathcal{B}[k] \qquad \mathcal{I}' = \mathcal{I} \cup \{k\} \qquad \mathcal{R}' = \mathcal{R}[k \leftarrow \delta'] \qquad \mathcal{X}'_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}', \mathcal{R}', \mathcal{B}, \_)\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \underset{\tau}{\rightarrow} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Read

$$\frac{\mathcal{X}_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \_, \mathcal{I}, \mathcal{R}, \_, \_)\} \qquad k \in \mathcal{I} \qquad \mathcal{R}[k] = \delta \qquad \delta \in \mathit{Assign}_\bot \qquad v = \delta(\bot)}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{read}(k) \rightarrow v} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r})}$$

Abort

$$\frac{\mathcal{X}_\mathrm{r} = \mathcal{X}'_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \sigma' = \mathsf{doAbort}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}'_\mathrm{a} = \mathcal{X}_\mathrm{a} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{abort}()} (\sigma', \mathcal{X}'_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Commit

$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}'_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \forall T \in \mathcal{X}_\mathrm{c}, T.\mathsf{ct} \neq \mathsf{ct} \qquad \mathsf{st} \leq \mathsf{ct} \\ \mathsf{noInversion}(\mathsf{ct}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \qquad \sigma' = \mathsf{doCommit}(\sigma, \tau, \mathsf{st}, \mathcal{B}, \mathsf{ct}) \qquad \mathcal{X}'_\mathrm{c} = \mathcal{X}_\mathrm{c} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \mathsf{ct})\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{commit(ct)}} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}'_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Aborted, Committed, Running

Transaction buffers

Transaction descriptor

Transaction Semantics - BeginTxn

$$\textsc{BeginTxn}\quad \frac{\forall T \in \mathcal{X}_{\mathrm{a}} \uplus \mathcal{X}_{\mathrm{c}} \uplus \mathcal{X}_{\mathrm{r}}, T.\tau \neq \tau \qquad \sigma' = \mathsf{doBegin}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}'_{\mathrm{r}} = \mathcal{X}_{\mathrm{r}} \uplus \{(\tau, \mathsf{st}, \varnothing, \varnothing, \varnothing, \_)\}}{(\sigma, \mathcal{X}_{\mathrm{a}}, \mathcal{X}_{\mathrm{c}}, \mathcal{X}_{\mathrm{r}}) \xrightarrow[\tau]{\mathsf{beginTxn}(\mathsf{st})} (\sigma', \mathcal{X}_{\mathrm{a}}, \mathcal{X}_{\mathrm{c}}, \mathcal{X}'_{\mathrm{r}})}$$

Pre Conditions:

1. Transaction not already running, aborted or committed

State Changes:

1. Add do begin record to the store
2. Add transaction to the set of running transactions

Transaction Semantics - Commit

COMMIT

$$\frac{\mathcal{X}_\mathrm{r} = \mathcal{X}'_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \quad \forall T \in \mathcal{X}_\mathrm{c}, T.\mathsf{ct} \neq \mathsf{ct} \quad \mathsf{st} \leq \mathsf{ct} \quad \mathsf{noInversion}(\mathsf{ct}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \quad \sigma' = \mathsf{doCommit}(\sigma, \tau, \mathsf{st}, \mathcal{B}, \mathsf{ct}) \quad \mathcal{X}'_\mathrm{c} = \mathcal{X}_\mathrm{c} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \mathsf{ct})\}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{commit}(\mathsf{ct})} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}'_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Pre Conditions for Update:

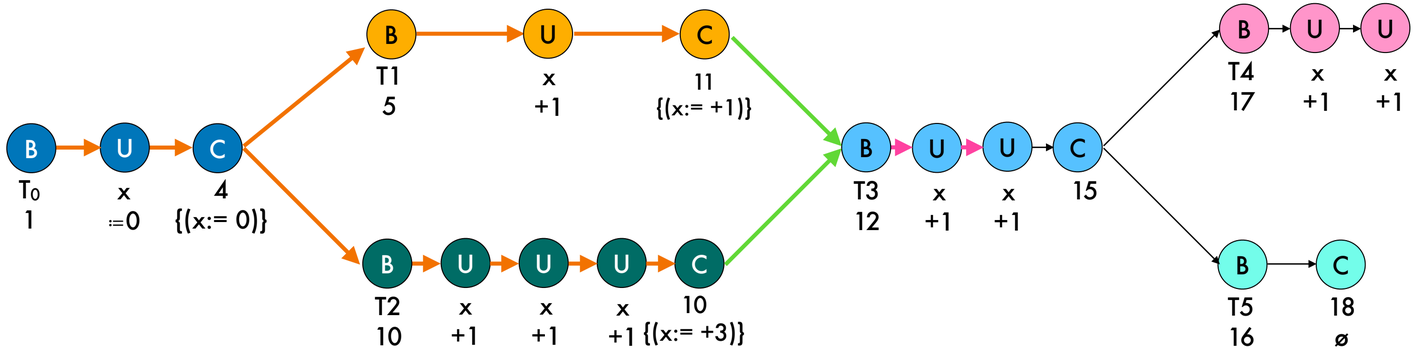
1. Transaction should be running
2. Commit timestamp should be unique
3. Snapshot timestamp ≤ commit timestamp
4. Dependency inversion should not occur

State Changes:

1. Add commit record to the store
2. Update buffers in the Txn descriptor

Execution traces and Consistency

Traces - Correctness?



Correctness Criterion [Cerone et. al.] :

- Trace is Transactionally Causally Consistent (TCC)
- TCC = Ext ∧ Int ∧ TransVis

SnapshotRead

$$\frac{\mathcal{X}_\mathrm{r} = \mathcal{X}_\mathrm{r}'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad k \notin \mathcal{I} \qquad \mathsf{lookup}(\sigma, k, \mathsf{st}) = \delta \qquad \delta' = \delta \odot \mathcal{B}[k] \qquad \mathcal{I}' = \mathcal{I} \cup \{k\} \qquad \mathcal{R}' = \mathcal{R}[k \leftarrow \delta'] \qquad \mathcal{X}_\mathrm{r}' = \mathcal{X}_\mathrm{r}'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}', \mathcal{R}', \mathcal{B}, \_)\}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \underset{\tau}{\rightarrow} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}')}$$

Read

$$\frac{\mathcal{X}_\mathrm{r} = \mathcal{X}_\mathrm{r}'' \uplus \{(\tau, \_, \mathcal{I}, \mathcal{R}, \_, \_)\} \qquad k \in \mathcal{I} \qquad \mathcal{R}[k] = \delta \qquad \delta \in \mathit{Assign}_\perp \qquad v = \delta(\perp)}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{read}(k) \rightarrow v} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r})}$$

→ **Ext**: Snapshot Reads: First reads contains all previous transactions.

- Read rule: First read is a snapshot

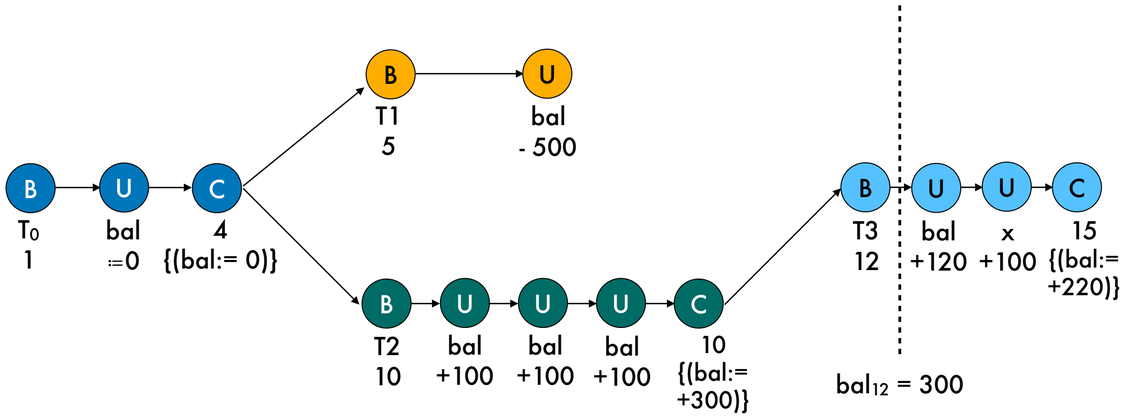
→ **Int**: Read-My-Writes

- Read rule: later reads apply local effects

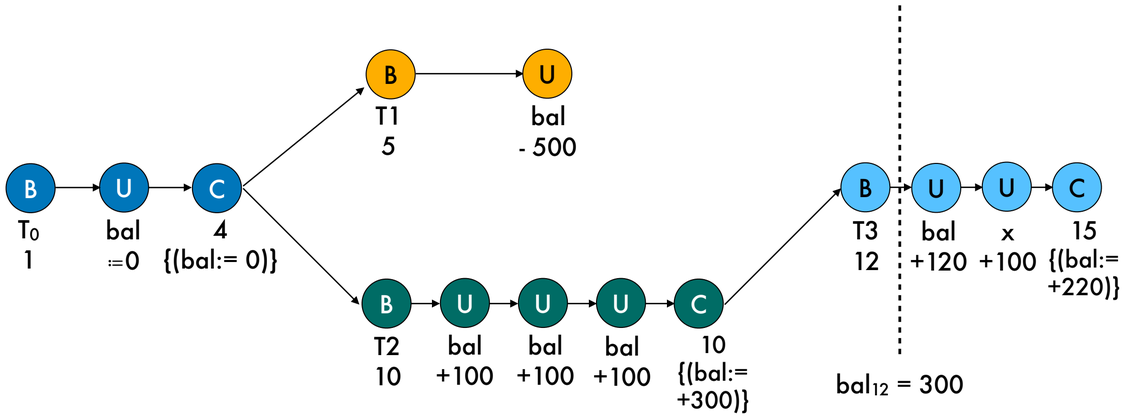
→ **TransVis**: Visibility is transitive

- By construction of a well formed trace

Ext + Int + TransViz ≠ TCC

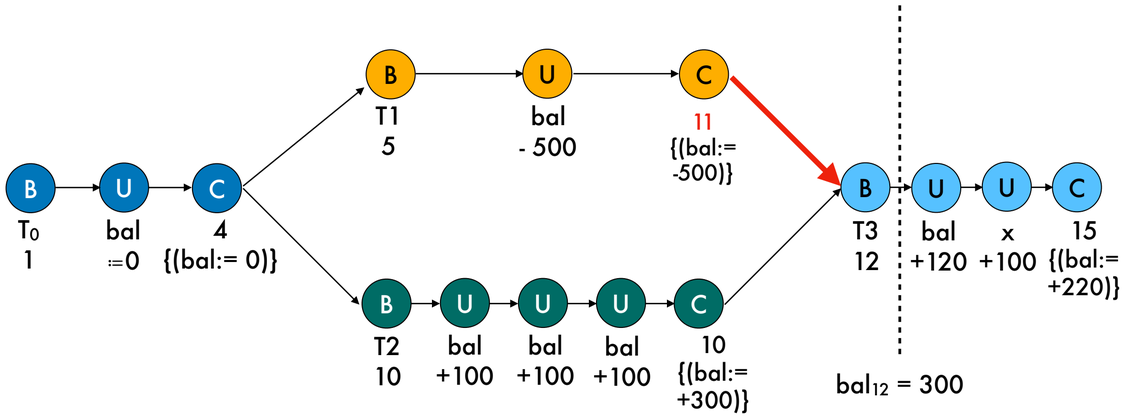


Ext + Int + TransViz ≠ TCC



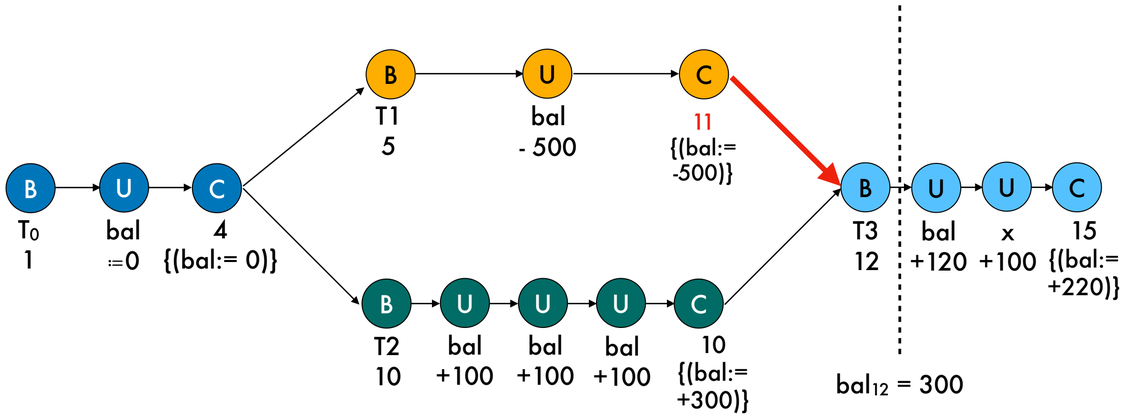
T3 (possibly) reads from its snapshot

Ext + Int + TransViz ≠ TCC



T3 (possibly) reads from its snapshot

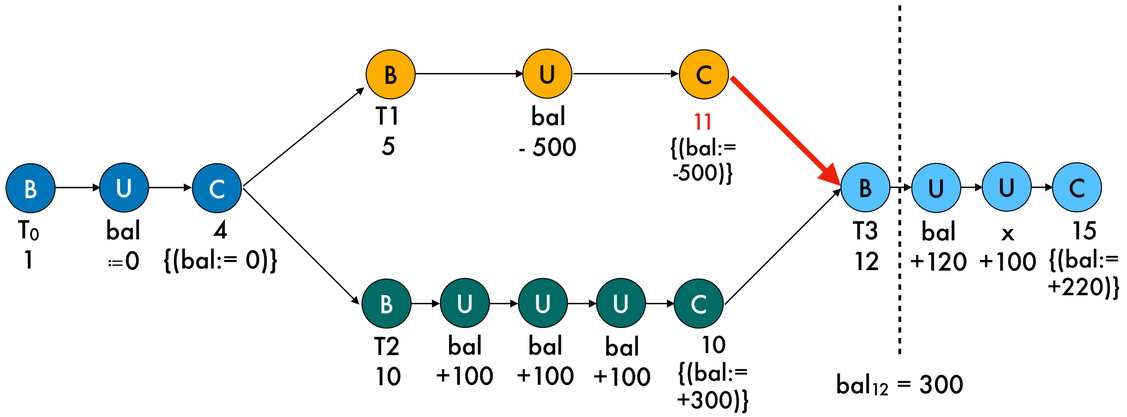
Ext + Int + TransViz ≠ TCC



T3 (possibly) reads from its snapshot

T1 becomes visible to T3 and changes its causal past: **anomaly!**
The balance i read has changed, Money is lost!!

Ext + Int + TransViz ≠ TCC

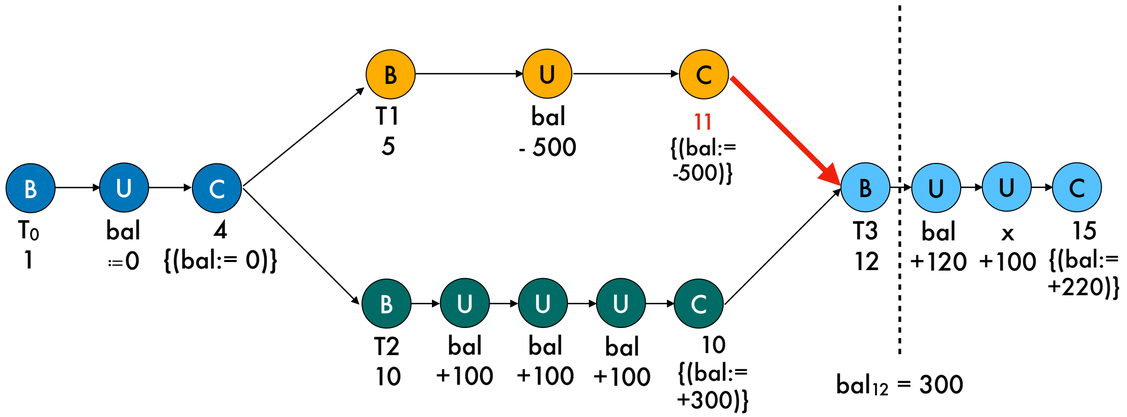


T3 (possibly) reads from its snapshot

T1 becomes visible to T3 and changes its causal past: **anomaly!**
The balance i read has changed, Money is lost!!

Does not violate the wellformedness rules of a trace

Ext + Int + TransViz ≠ TCC



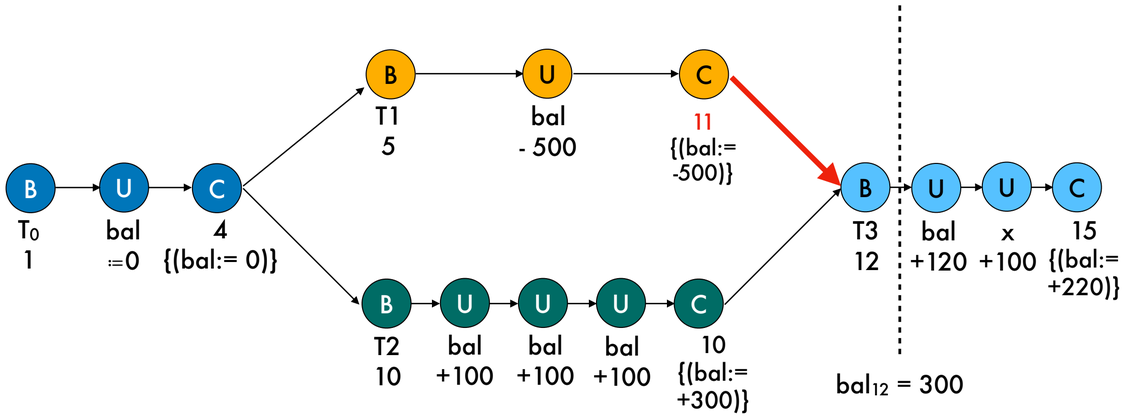
T3 (possibly) reads from its snapshot

T1 becomes visible to T3 and changes its causal past: **anomaly!**
The balance i read has changed, Money is lost!!

Does not violate the wellformedness rules of a trace

Does not violate the definition of TCC (remains a DAG)

Ext + Int + TransViz ≠ TCC



T3 (possibly) reads from its snapshot

T1 becomes visible to T3 and changes its causal past: **anomaly!**
The balance i read has changed, Money is lost!!

Does not violate the wellformedness rules of a trace

Does not violate the definition of TCC (remains a DAG)

Definition assumes correctness on a history

In practice, execution traces are dynamic

How can we guarantee that a visible prefix is immutable?

No Inversion Condition

Prevents transactions from choosing a commit timestamp which change causal past.

When adding a commit node, we ensure the following:

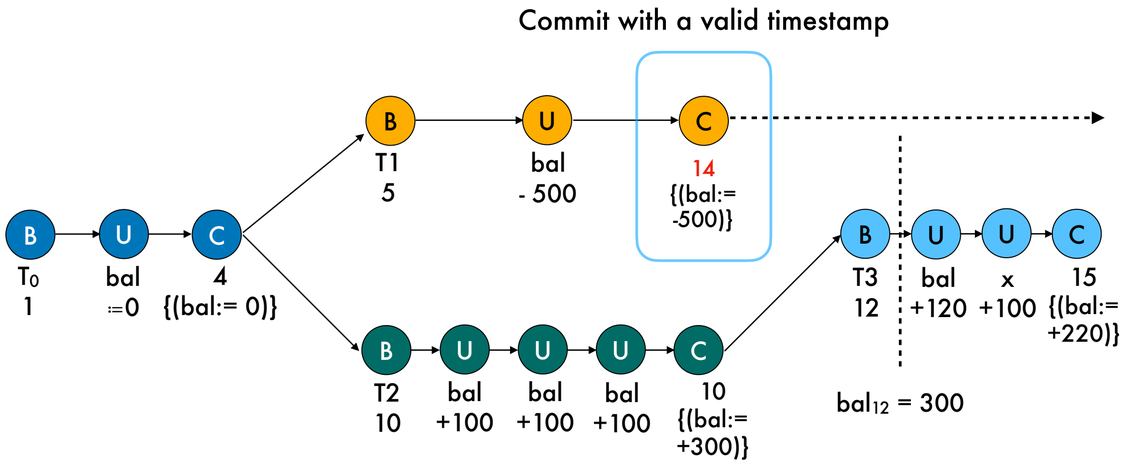
$$noInversion(n) \triangleq \forall B : n.ct \not\leq B.st$$

No Inversion Condition

Prevents transactions from choosing a commit timestamp which change causal past.

When adding a commit node, we ensure the following:

$$noInversion(n) \triangleq \forall B : n.ct \not\leq B.st$$



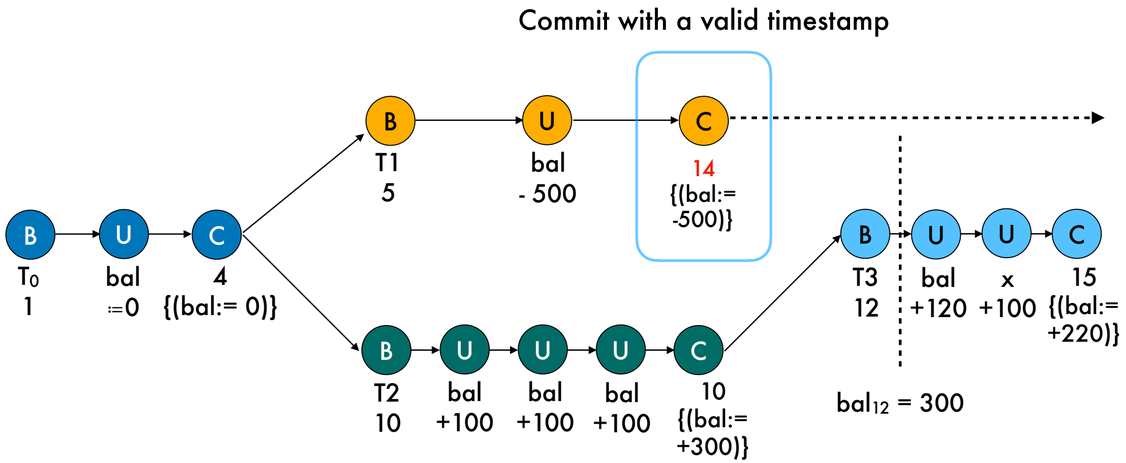
Solution 1

No Inversion Condition

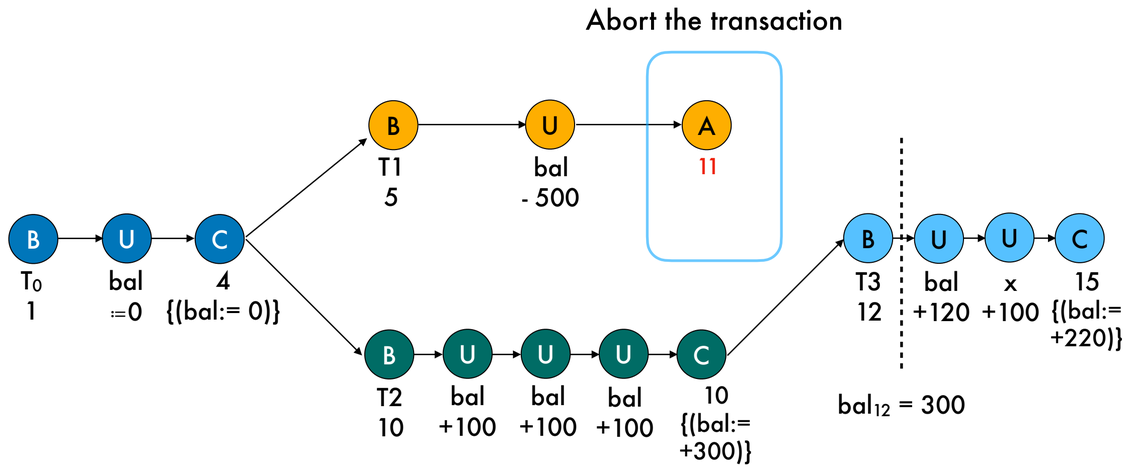
Prevents transactions from choosing a commit timestamp which change causal past.

When adding a commit node, we ensure the following:

$$noInversion(n) \triangleq \forall B : n.ct \not\leq B.st$$



Solution 1



Solution 2

Correctness

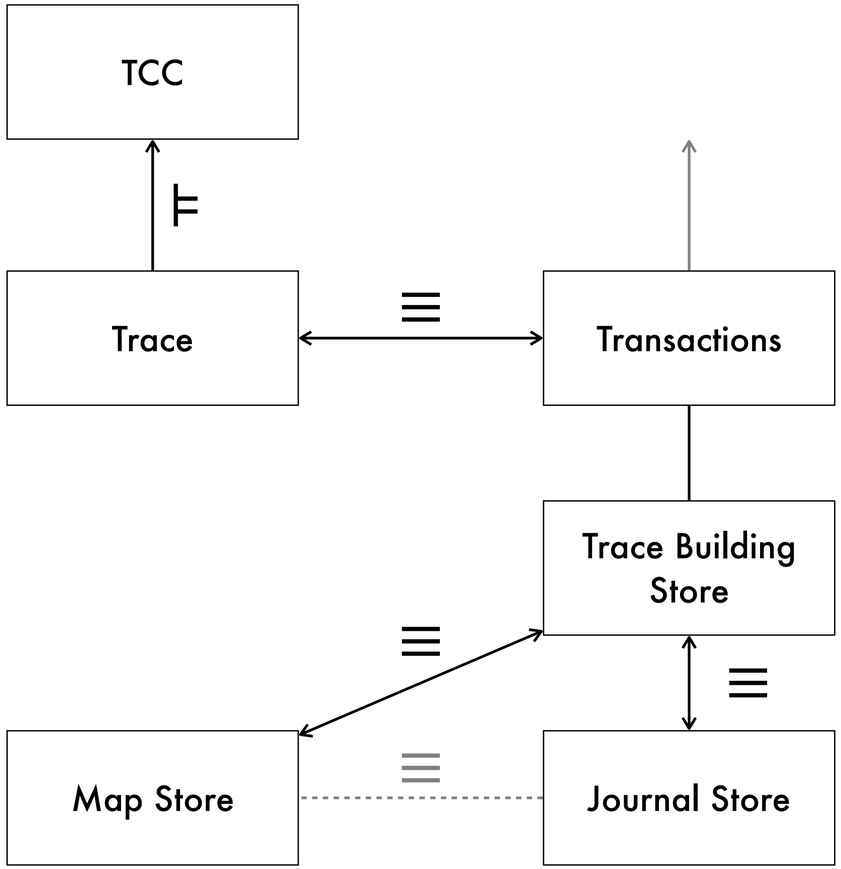
Correctness Proofs

Correct Transactions = TCC + NoInversion

1. Show that Transaction Semantics bisimulates Trace Semantics
2. Show that Trace semantics ⊨ TCC + NoInversion
3. Using 1, Transaction semantics ⊨ TCC + No inversion

Store Equivalence = Map and Journal are interchangeable

1. Valuation of Trace ⊨ Lookup in Map
2. Valuation of Trace ⊨ Lookup in Journal
3. Since valuation is the same, we get observational equivalence.



Proof Results:

1. **Transaction semantics preserve TCC + NoInversion**
2. **Journal Store and Map store are equivalent under transaction semantics**

Bisimulation example - BeginTxn

Trace semantics	Transaction semantics
Preconditions: • τ unique • $\mathsf{st} \neq \bot$ • $p \in parents_{\mathsf{G}}(n) \implies p \in \mathbb{C} \wedge p \in visited$	• **Preconditions:** • τ unique • $\mathsf{st} \neq \bot$ • $\{(\tau, \mathsf{st}, \_, \_, \_, \_)\} \notin \mathcal{X}_{\mathrm{r}}$ $\wedge \nexists \mathbb{T}(\tau', \mathsf{st}', \_, \_, \_, ct')$ $: st \geq ct' \wedge \mathbb{T} \notin \mathcal{X}_{\mathrm{c}}$
State Change: • $\sigma' = \mathsf{doBegin}(\sigma, \tau, \mathsf{st})$ • $visited' = visited \cup \{\mathbb{B}(\tau, \mathsf{st})\}$	**State Change:** • $\sigma' = \mathsf{beginTxn}(\mathsf{st})$ • $\mathcal{X}'_{\mathrm{r}} = \mathcal{X}_{\mathrm{r}} \cup \{(\tau, \mathsf{st}, \_, \_, \_, \_)\}$
Postconditions: • $\{\mathbb{B}(\tau, \mathsf{st})\} \in visited'$ • $B \in \sigma'$	**Postconditions:** • $\{(\tau, \mathsf{st}, \_, \_, \_, \_)\} \in \mathcal{X}'_{\mathrm{r}}$ • $\mathsf{beginTxn} \in \sigma'$
Enables: • UPDATE • COMMIT • ABORT	**Enables:** • UPDATE • COMMIT • ABORT

From Model to Implementation

Specification as Pseudocode

BeginTxn

$$\frac{\forall T \in \mathcal{X}_\mathrm{a} \uplus \mathcal{X}_\mathrm{c} \uplus \mathcal{X}_\mathrm{r}, T.\tau \neq \tau \qquad \sigma' = \mathsf{doBegin}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}'_\mathrm{r} = \mathcal{X}_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \varnothing, \varnothing, \varnothing, \_)\}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{beginTxn}(\mathsf{st})} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Update

$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \\ \delta \neq \bot \qquad \mathcal{B}' = \mathcal{B}[k \leftarrow \mathcal{B}[k] \odot \delta] \qquad \sigma' = \mathsf{doUpdate}(\sigma, \tau, k, \mathsf{st}, \delta) \\ \mathcal{R}' = (\text{ if } k \notin \mathcal{I} \text{ then } \mathcal{R} \text{ else } \mathcal{R}[k \leftarrow \mathcal{R}[k] \odot \delta] \,) \qquad \mathcal{X}'_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}', \_)\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{update}(k,\delta)} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

SnapshotRead

$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad k \notin \mathcal{I} \qquad \mathsf{lookup}(\sigma, k, \mathsf{st}) = \delta \\ \delta' = \delta \odot \mathcal{B}[k] \qquad \mathcal{I}' = \mathcal{I} \cup \{k\} \qquad \mathcal{R}' = \mathcal{R}[k \leftarrow \delta'] \qquad \mathcal{X}'_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}', \mathcal{R}', \mathcal{B}, \_)\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \underset{\tau}{\rightarrow} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Read

$$\frac{\mathcal{X}_\mathrm{r} = \mathcal{X}''_\mathrm{r} \uplus \{(\tau, \_, \mathcal{I}, \mathcal{R}, \_, \_)\} \qquad k \in \mathcal{I} \qquad \mathcal{R}[k] = \delta \qquad \delta \in \mathit{Assign}_\bot \qquad v = \delta(\bot)}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{read}(k) \rightarrow v} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r})}$$

Abort

$$\frac{\mathcal{X}_\mathrm{r} = \mathcal{X}'_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \sigma' = \mathsf{doAbort}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}'_\mathrm{a} = \mathcal{X}_\mathrm{a} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{abort}()} (\sigma', \mathcal{X}'_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Commit

$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}'_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \forall T \in \mathcal{X}_\mathrm{c}, T.\mathsf{ct} \neq \mathsf{ct} \qquad \mathsf{st} \leq \mathsf{ct} \\ \mathsf{noInversion}(\mathsf{ct}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \qquad \sigma' = \mathsf{doCommit}(\sigma, \tau, \mathsf{st}, \mathcal{B}, \mathsf{ct}) \qquad \mathcal{X}'_\mathrm{c} = \mathcal{X}_\mathrm{c} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \mathsf{ct})\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{commit}(\mathsf{ct})} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}'_\mathrm{c}, \mathcal{X}'_\mathrm{r})}$$

Specification as Pseudocode

BeginTxn

$$\frac{\forall T \in \mathcal{X}_\mathrm{a} \uplus \mathcal{X}_\mathrm{c} \uplus \mathcal{X}_\mathrm{r}, T.\tau \neq \tau \qquad \sigma' = \mathsf{doBegin}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}_\mathrm{r}' = \mathcal{X}_\mathrm{r} \uplus \{(\tau, \mathsf{st}, \varnothing, \varnothing, \varnothing, \_)\}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{beginTxn}(\mathsf{st})} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}')}$$

Update

$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}_\mathrm{r}'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \\ \delta \neq \bot \qquad \mathcal{B}' = \mathcal{B}[k \leftarrow \mathcal{B}[k] \odot \delta] \qquad \sigma' = \mathsf{doUpdate}(\sigma, \tau, k, \mathsf{st}, \delta) \\ \mathcal{R}' = (\text{ if } k \notin \mathcal{I} \text{ then } \mathcal{R} \text{ else } \mathcal{R}[k \leftarrow \mathcal{R}[k] \odot \delta] \,) \qquad \mathcal{X}_\mathrm{r}' = \mathcal{X}_\mathrm{r}'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}', \_)\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{update}(k,\delta)} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}')}$$

SnapshotRead

$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}_\mathrm{r}'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad k \notin \mathcal{I} \qquad \mathsf{lookup}(\sigma, k, \mathsf{st}) = \delta \\ \delta' = \delta \odot \mathcal{B}[k] \qquad \mathcal{I}' = \mathcal{I} \cup \{k\} \qquad \mathcal{R}' = \mathcal{R}[k \leftarrow \delta'] \qquad \mathcal{X}_\mathrm{r}' = \mathcal{X}_\mathrm{r}'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}', \mathcal{R}', \mathcal{B}, \_)\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \underset{\tau}{\rightarrow} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}')}$$

Read

$$\frac{\mathcal{X}_\mathrm{r} = \mathcal{X}_\mathrm{r}'' \uplus \{(\tau, \_, \mathcal{I}, \mathcal{R}, \_, \_)\} \qquad k \in \mathcal{I} \qquad \mathcal{R}[k] = \delta \qquad \delta \in \mathit{Assign}_\bot \qquad v = \delta(\bot)}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{read}(k) \rightarrow v} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r})}$$

Abort

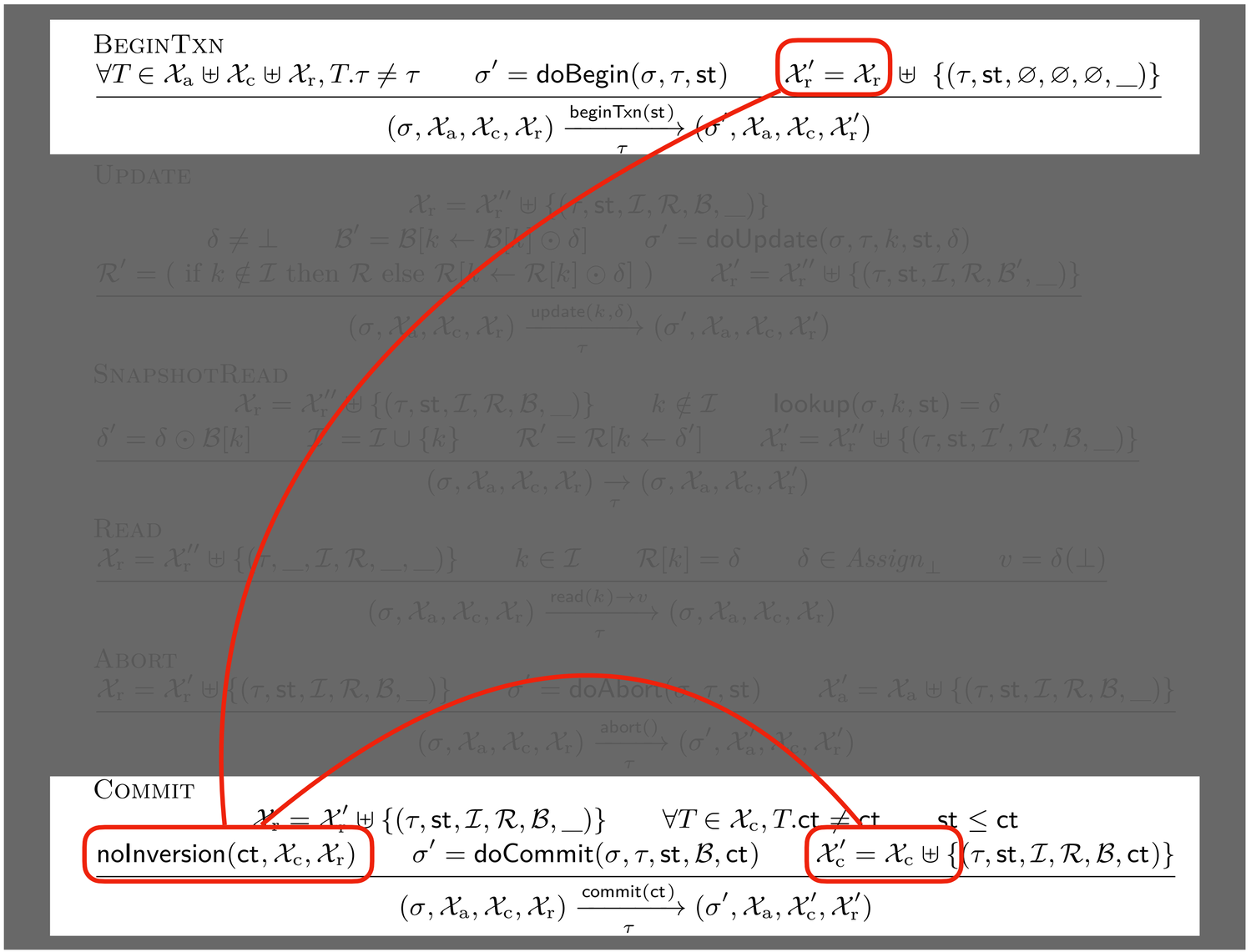
$$\frac{\mathcal{X}_\mathrm{r} = \mathcal{X}_\mathrm{r}' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \sigma' = \mathsf{doAbort}(\sigma, \tau, \mathsf{st}) \qquad \mathcal{X}_\mathrm{a}' = \mathcal{X}_\mathrm{a} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{abort}()} (\sigma', \mathcal{X}_\mathrm{a}', \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}')}$$

Commit

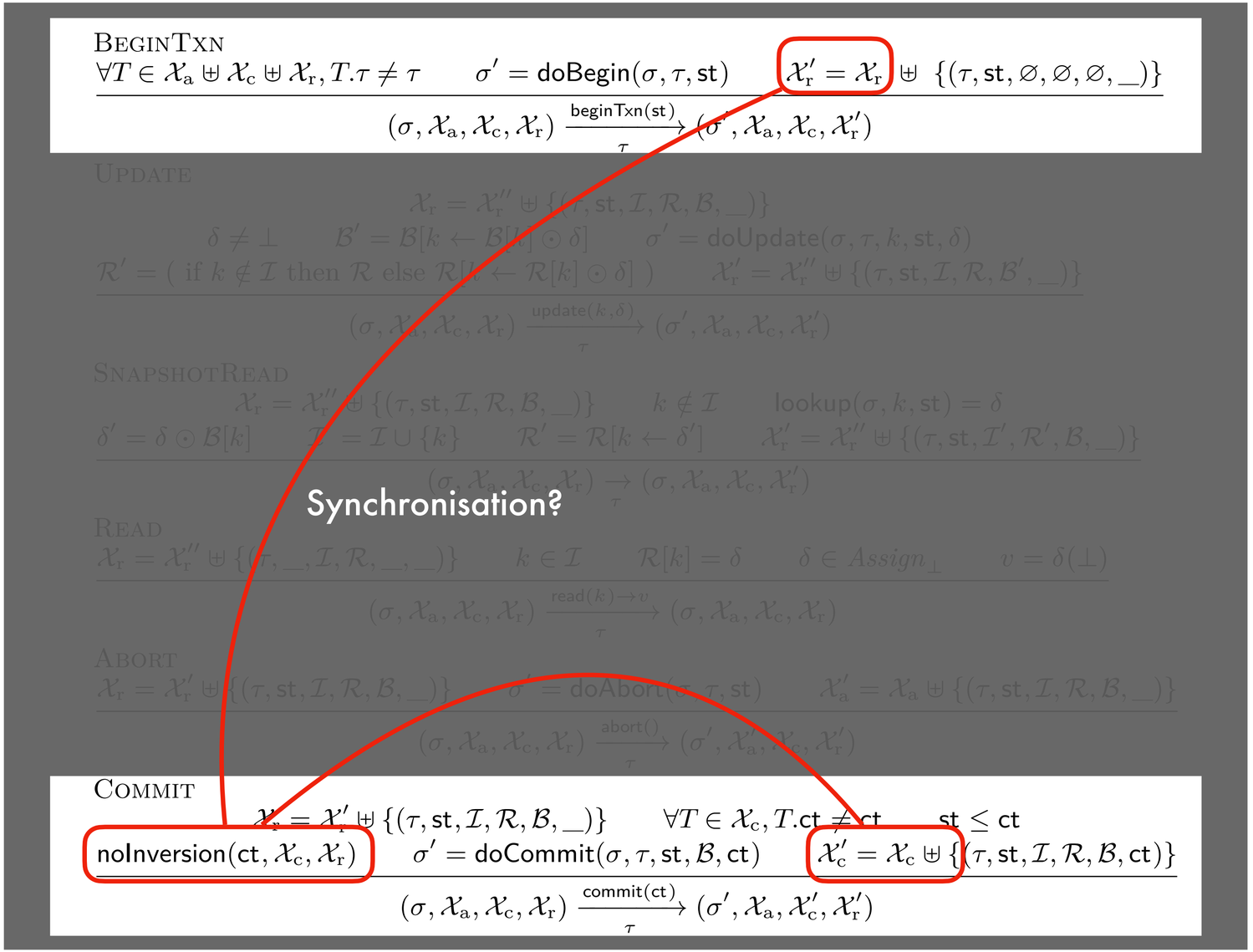
$$\frac{\begin{array}{c}\mathcal{X}_\mathrm{r} = \mathcal{X}_\mathrm{r}' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad \forall T \in \mathcal{X}_\mathrm{c}, T.\mathsf{ct} \neq \mathsf{ct} \qquad \mathsf{st} \leq \mathsf{ct} \\ \mathsf{noInversion}(\mathsf{ct}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \qquad \sigma' = \mathsf{doCommit}(\sigma, \tau, \mathsf{st}, \mathcal{B}, \mathsf{ct}) \qquad \mathcal{X}_\mathrm{c}' = \mathcal{X}_\mathrm{c} \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \mathsf{ct})\}\end{array}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{commit}(\mathsf{ct})} (\sigma', \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}', \mathcal{X}_\mathrm{r}')}$$

Atomic / all-or-nothing

Specification as Pseudocode



Specification as Pseudocode



Implementing the Specification

Academic Importance:

Faithful translation of specification into code => Map and Journal Stores + Composition logic

- A lot of asserts and tests with validation and fault injection

Real World Use:

LSMTree: Emulating RocksDB => CobbleDB

- Compose multiple Map and Journal stores to produce caches, checkpoints, memtables etc.
- 4K LoC total of Java << RocksDB ≈ 300 KLoC
- Confidence in Correctness
- Sub-par Performance

Lessons Learned

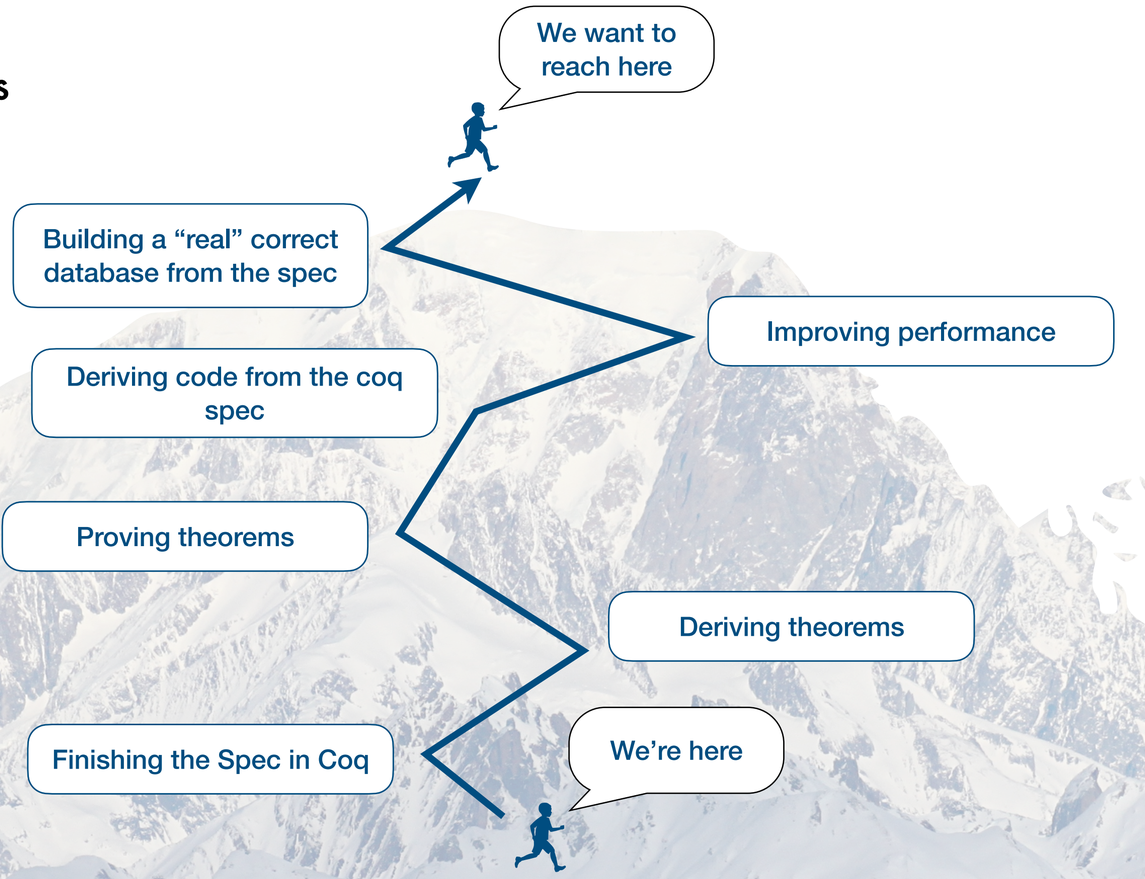
High and steep mountain!

Specification is useful for system implementors:

- Compact, aids reasoning; no forgetting edge cases
- Read as pseudocode
- Identify concurrency, crash-tolerance issues

New: rigorous store composition

- Explanatory value
- Ease of implementation



Mont Blanc - Highest Mountain in Europe

Contributions

Formalised:

- Rich Datatypes: Incremental, Convergent
- Transactional trace, valuation
- Store behaviour: map, journal
- Safety: transactions ∩ store ⟶ valuation
  - Satisfied for map, journal
- Client-driven transaction semantics
  - Bisimulates traces
  - Map, Journal safe

Ongoing:

- Specification in Coq and Dafny to extract correct by design code into OCaml (functional) and Java (procedural)
- Correct-by-construction implementation via asserts and test suites
- Better performance, within model improving data structures used to implement spec.

Appendices and References

RocksDB by Composition

Live level

Jrnl$_1$ (immutable): *[t3, t6]*

(k_4, δ_)	(k_3, δ_)	(k_1, δ_)	(k_2, δ_)
t_3	t_4	t_5	t_6

Jrnl$_2$ (immutable): *[t_7, t_{10}]*

(k_7, δ_)	(k_2, δ_)	(k_4, δ_)	(k_2, δ_)
t_7	t_8	t_9	t_{10}

Jrnl$_3$ (active): *[t_{11}, now]*

(k_1, δ_)	(k_1,δ_)		
t_{11}	t_{12}	t_{13}	t_{14}

Memtable$_1$ (immutable): *[t_3, t_6]*

(k_1, δ_)	(k_2, δ_)	(k_3, δ_)	(k_4, δ_)

Memtable$_2$ (immutable): *[t_7, t_{10}]*

(k_2, δ_)	(k_4, δ_)	(k_7, δ_)

Memtable$_3$ (active): *[t_{11}, now]*

(k_1, δ_)		

Level 0

SST$_1$: *[t_1, t_2)*

(k_1, δ_)	(k_3, δ_)	(k_5, δ_)	(k_6, δ_)	(k_7, δ_)	(k_8, δ_)

SST$_2$: *[t_2, t_3)*

(k_3, δ_)	(k_4, δ_)	(k_5, δ_)	(k_8, δ_)

Level 1: *[t_0 = 0, t_1)*

SST$_1$: keys 0–10

(k_0, :=_)	(k_1, :=_)	(k_2, :=_)	(k_3, :=_)	(k_8, :=_)	(k_9, :=_)

SST$_1$: keys 11+

(k_{17}, :=_)	(k_{19}, :=_)	(k_{20}, :=_)	(k_{21}, :=_)	(k_{22}, :=_)	(k_{77}, :=_)

Indices indicate order: t_1 older than t_2, k_1 less than k_2.
Journals and SSTs are persistent; Memtables are volatile.

Transaction Semantics - Read

$$\textsc{SnapshotRead}\quad \frac{\mathcal{X}_\mathrm{r} = \mathcal{X}_\mathrm{r}'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \qquad k \notin \mathcal{I} \qquad \mathsf{lookup}(\sigma, k, \mathsf{st}) = \delta \qquad \delta' = \delta \odot \mathcal{B}[k] \qquad \mathcal{I}' = \mathcal{I} \cup \{k\} \qquad \mathcal{R}' = \mathcal{R}[k \leftarrow \delta'] \qquad \mathcal{X}_\mathrm{r}' = \mathcal{X}_\mathrm{r}'' \uplus \{(\tau, \mathsf{st}, \mathcal{I}', \mathcal{R}', \mathcal{B}, \_)\}}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \underset{\tau}{\rightarrow} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}')}$$

$$\textsc{Read}\quad \frac{\mathcal{X}_\mathrm{r} = \mathcal{X}_\mathrm{r}'' \uplus \{(\tau, \_, \mathcal{I}, \mathcal{R}, \_, \_)\} \qquad k \in \mathcal{I} \qquad \mathcal{R}[k] = \delta \qquad \delta \in \mathit{Assign}_\perp \qquad v = \delta(\perp)}{(\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r}) \xrightarrow[\tau]{\mathsf{read}(k) \rightarrow v} (\sigma, \mathcal{X}_\mathrm{a}, \mathcal{X}_\mathrm{c}, \mathcal{X}_\mathrm{r})}$$

Pre Conditions for Read:

1. Transaction should be running
2. The key should be initialised, otherwise perform snapshot read

State Changes: None

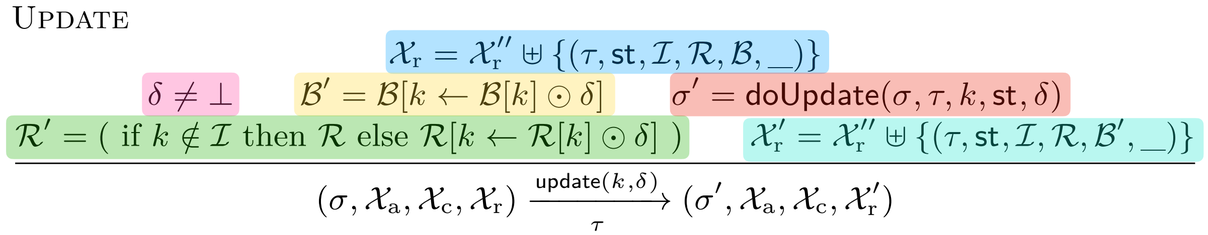
Pre Conditions for Snapshot read:

1. Transaction should be running
2. The key should not be initialised

State Changes:

1. Add key to the set of initialised keys
2. Add value of the key to the read buffer
3. Update buffers in Txn Descriptor

Transaction Semantics - Update



Pre Conditions for Update:

1. Transaction should be running
2. Effect should not be null

State Changes:

1. Add the effect of update to write buffer
2. Add the effect of update to read buffer
3. Add update record to the store
4. Update buffers in the Txn descriptor

Bisimulation example - Update

UPDATE

Trace semantics	Transaction semantics
Preconditions:	**Preconditions:**
• $p \in parents_{\mathsf{G}}(n)$ $\wedge\, p.\tau = n.\tau \wedge p.\mathsf{st} = n.\mathsf{st}$ $\implies (p \in \mathbb{U} \cup \mathbb{B}) \wedge p \in visited$	• $\{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \in \mathcal{X}_{\mathrm{r}}$
	• $\delta \neq \bot$
State Change:	**State Change:**
• $\sigma' = \mathsf{doUpdate}(\sigma, \tau, k, \mathsf{st}, \delta)$	• $\sigma' = \mathsf{doUpdate}(\sigma, \tau, k, \mathsf{st}, \delta)$ • $\mathcal{B}' = \mathcal{B}[k \leftarrow \mathcal{B}[k] \odot \delta]$ • $\mathcal{R}' = (\text{ if } \ k \notin \mathcal{I}$ then $\ \mathcal{R}$ else $\ \mathcal{R}[k \leftarrow \mathcal{R}[k] \odot \delta]\ )$
• $visited' = visited \cup \{\mathbb{U}(\tau, \mathsf{st})\}$	• $\mathcal{X}'_{\mathrm{r}} = \mathcal{X}_{\mathrm{r}} \cup \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}', \mathcal{B}', \_)\}$
Postconditions:	**Postconditions:**
• $\{\mathbb{U}(\tau, \mathsf{st})\} \in visited'$ • $U \in \sigma'$	• $\{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}', \mathcal{B}', \_)\} \in \mathcal{X}'_{\mathrm{r}}$ • $\mathsf{update} \in \sigma'$

Bisimulation example - Abort

UPDATE

Trace semantics	**Transaction semantics**
Preconditions:	**Preconditions:**
• $p \in \mathit{parents}_{\mathsf{G}}(n)$ $\wedge\, p.\tau = n.\tau \wedge p.\mathsf{st} = n.\mathsf{st}$ $\implies (p \in \mathbb{U} \cup \mathbb{B}) \wedge p \in \mathit{visited}$	• $\{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}, \mathcal{B}, \_)\} \in \mathcal{X}_{\mathrm{r}}$
	• $\delta \neq \bot$
State Change:	**State Change:**
• $\sigma' = \mathsf{doUpdate}(\sigma, \tau, k, \mathsf{st}, \delta)$	• $\sigma' = \mathsf{doUpdate}(\sigma, \tau, k, \mathsf{st}, \delta)$
	• $\mathcal{B}' = \mathcal{B}[k \leftarrow \mathcal{B}[k] \odot \delta]$
	• $\mathcal{R}' = (\text{ if } \ k \notin \mathcal{I}$ then $\mathcal{R}$ else $\mathcal{R}[k \leftarrow \mathcal{R}[k] \odot \delta]$ $)$
• $\mathit{visited}' = \mathit{visited} \cup \{\mathbb{U}(\tau, \mathsf{st})\}$	• $\mathcal{X}'_{\mathrm{r}} = \mathcal{X}_{\mathrm{r}} \cup \{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}', \mathcal{B}', \_)\}$
Postconditions:	**Postconditions:**
• $\{\mathbb{U}(\tau, \mathsf{st})\} \in \mathit{visited}'$	• $\{(\tau, \mathsf{st}, \mathcal{I}, \mathcal{R}', \mathcal{B}', \_)\} \in \mathcal{X}'_{\mathrm{r}}$
• $U \in \sigma'$	• $\mathsf{update} \in \sigma'$