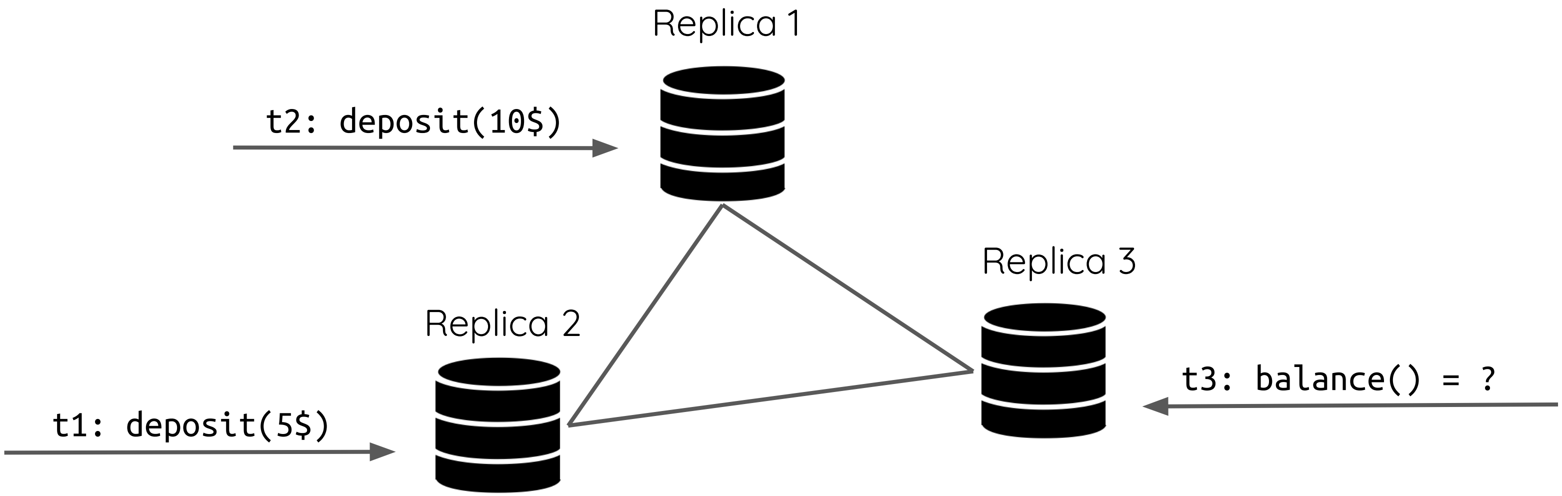
Configurable Consistency

Konstantinos Kallas -- WPE 2 Presentation

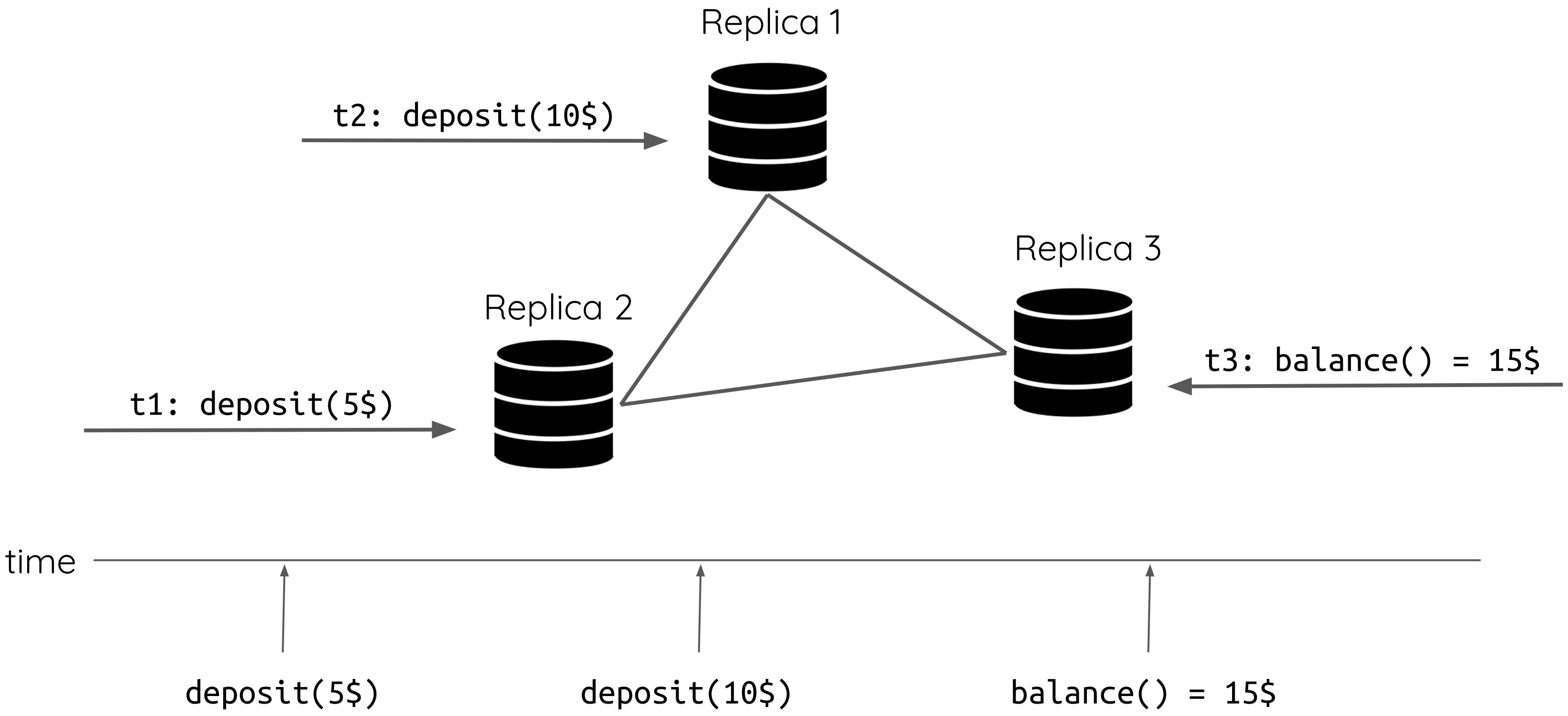
Motivation

Consistency Notions

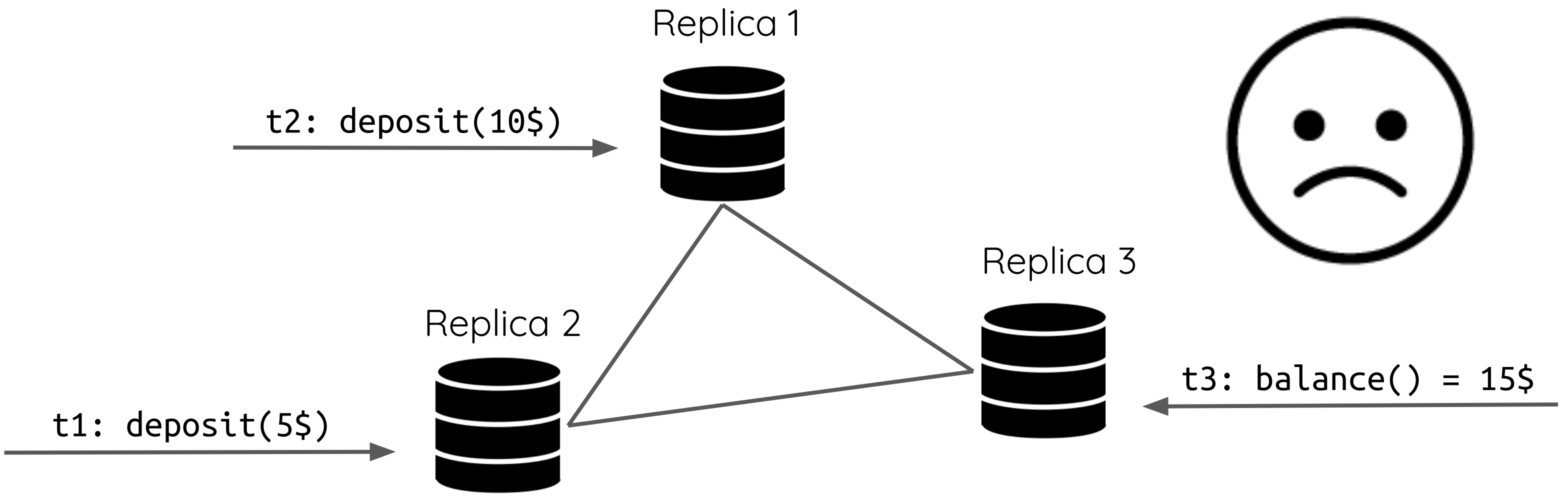


Replicated Datastore

Linearizability



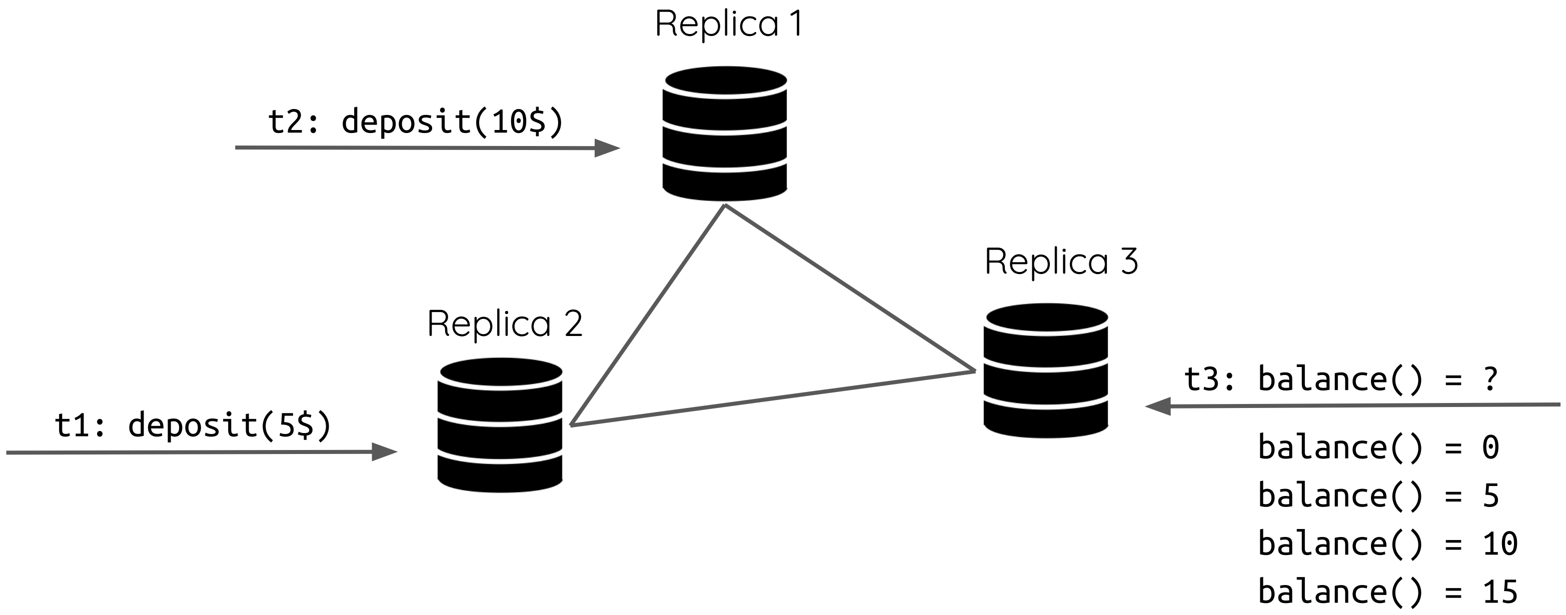
Linearizability too strong?



Synchronization overhead ↑↑↑

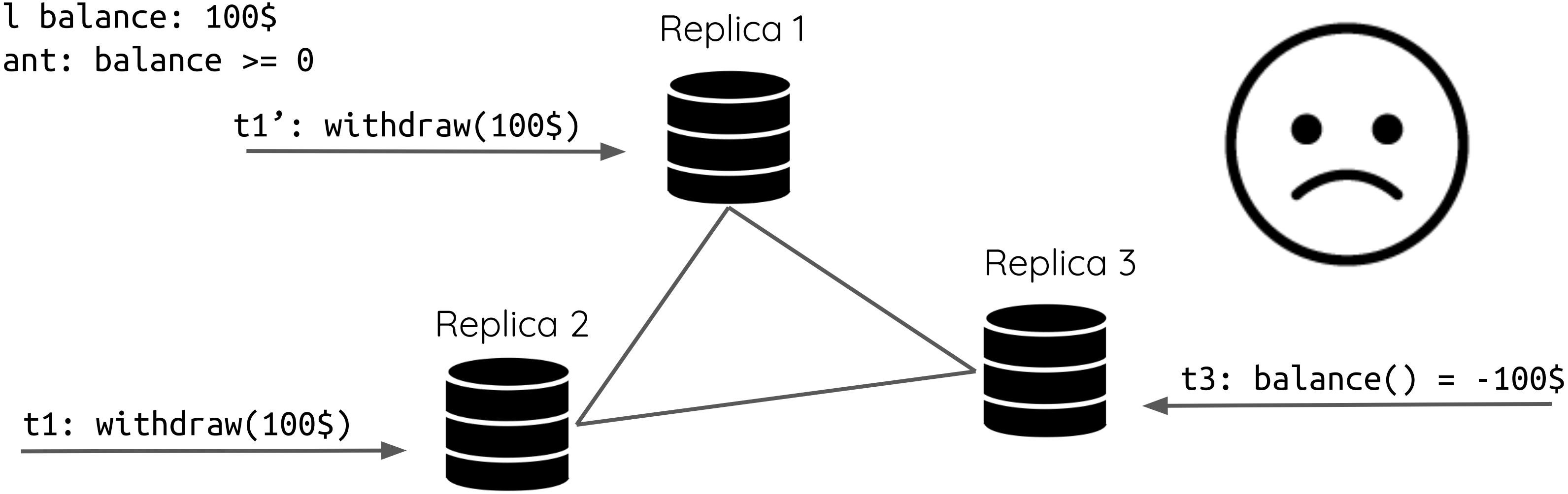
Availability ↓↓↓

Eventual Consistency



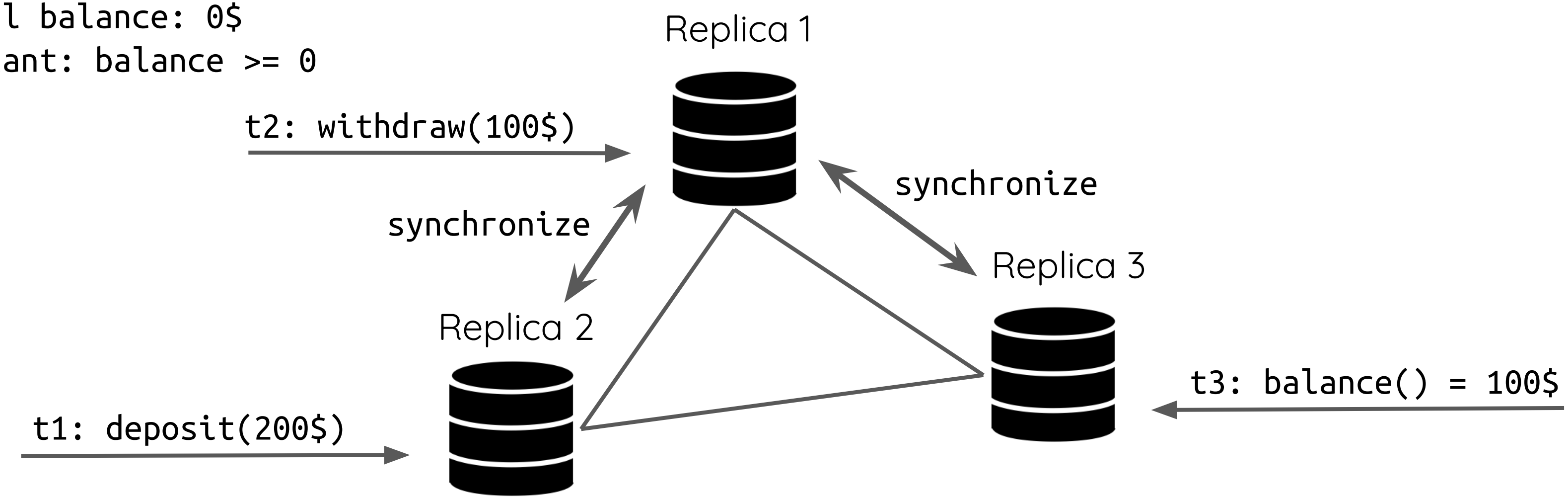
Eventual Consistency too weak?

```
Initial balance: 100$
Invariant: balance >= 0
```



Why not both?

```
Initial balance: 0$
Invariant: balance >= 0
```



Configurable Consistency Notions

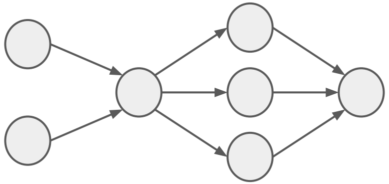
Talk Outline

I. Common Model
II. Configurable Consistency Notions:
    A. RedBlue Consistency -- Li et al. -- OSDI 2012
    B. Explicit Consistency -- Balegas et al. -- EuroSys 2015
    C. Reasoning about Consistency -- Gotsman et al. -- POPL 2016
III. Comparison
IV. Future Work

Common Model

Model

Executions are partial orders (PO) of events:



Each event is an operation execution, e.g. `balance() = 100$`

Each replica sees a consistent serialization of the PO.

A consistency notion restricts the execution POs that can be observed.

All three papers support causal consistency as the weakest notion.

(i) RedBlue Consistency

Main Idea

Label operations as:

Red: Strong Consistency

`withdraw(x)`

Blue: Weak Consistency

`deposit(x)`
`balance()`
`accrue_interest()`

Also:

- Ensuring convergence
- Conditions for labelling operations

Model -- RedBlue Order

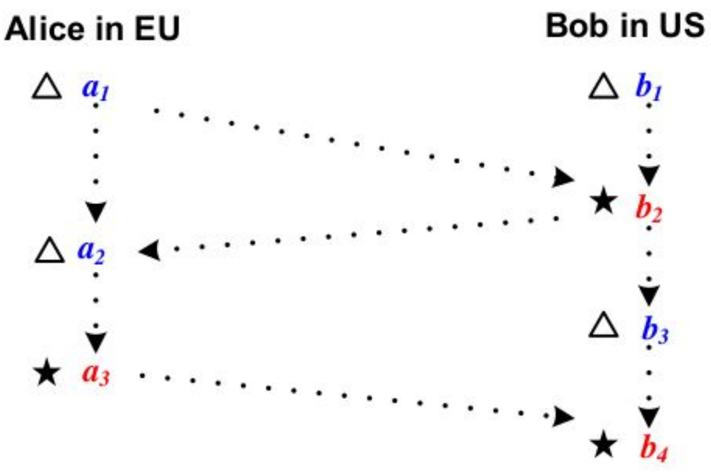
Definition 1 (RedBlue order) *Given a set of operations* $U = R \cup B$, *where* R *and* B *denote the red and blue operation set, respectively, and* $R \cap B = \emptyset$, *a* RedBlue order *is a partial order* $O = (U, \prec)$ *with the restriction that* $\forall u, v \in R$ *such that* $u \neq v$, $u \prec v$ *or* $v \prec u$ *(i.e., red operations are totally ordered).*

Model -- Causal Serializations

Definition 3 (Causal legal serialization) *Given a site i, $O_i = (U, <)$ is an i-*causal legal serialization *(or short, a* causal serialization*) of RedBlue order $O = (U, \prec)$ if*

- *O_i is a legal serialization of O, and*
- *for any two operations $u, v \in U$, if $site(v) = i$ and $u < v$ in O_i, then $u \prec v$.*

Model -- Example



(a) RedBlue order O of operations

(b) Causal serializations of O

Model -- RedBlue Consistency

Definition 3 (RedBlue consistency)**.** *A replicated system is O*-RedBlue consistent *(or short, RedBlue consistent) if each site i applies operations according to an i-causal serialization of RedBlue order O.*

State Convergence

(a) RedBlue order O of operations issued by Alice and Bob

(b) Causal serializations of O leading to diverged state

Shadow Operations

```
deposit(amount):
  lambda balance:
    return (balance + amount)

accrue_interest():
  lambda balance:
    return (balance * 1.05)
```

Instantly performed

```
deposit_gen(amount):
  lambda balance:
    return deposit_shadow(amount)

deposit_shadow(amount):
  lambda balance:
    return (balance + amount)

accrue_interest_gen():
  lambda balance:
    return accrue_interest_shadow(balance)

accrue_interest_shadow(balance):
  lambda balance’:
    return (balance’ + balance * 0.05)
```

Addition with a constant commutes with deposit

Invariant preservation

Definition 7 (Invariant safe). *Shadow operation $h_u(S)$ is invariant safe if for all valid states S and S', the state $S' + h_u(S)$ is also valid.*

Conditions

1. Label any pair of non-commutative ops **Red**
2. Label all non invariant-safe ops **Red**
3. Label all other ops **Blue**

Summary

- Main Idea: **Red** and **Blue** operations
- Shadow operations to improve commutativity
- Conditions for labelling operations

Pros:

- Clean model
- Easy to use and configure

Cons:

- No automation
- Very coarse-grained

(ii) Explicit Consistency

Main Idea

Finer-grained control of synchronization using reservations

- Reservations are types of locks
- Reduce synchronization for specific invariants

Also:

- Static analysis to identify unsafe pairs of operations

Model -- Serializations

Definition 2.1 (I-valid serialization)**.** Given a set of transactions T and its associated happens-before partial order $\prec$, $O_i = (T, <)$ is an *I-valid serialization* of $O = (T, \prec)$ if O_i is a valid serialization of O, and I holds in every state that results from executing some prefix of O_i.

Model -- Explicit Consistency

Definition 2.2 (Explicit consistency). A system provides Explicit Consistency if all serializations of $O = (T, \prec)$ are I-valid serializations, where T is the set of transactions executed in the system and $\prec$ their associated partial order.

Performing two withdraw(x) operations concurrently would lead to I-invalid serializations

Identifying unsafe operations

- User specifies invariant
- User writes postconditions for each operation
- Static analysis identifies and reports unsafe pairs

Invariant Specification

Some example invariants:

- Bounds: forall A, account(A) => balance(A) >= 0
- Uniqueness: forall A, account(A) => nrOwners(A) = 1
- Integrity: forall A, hasField(A, “balance”) => account(A)

Postconditions

Operations are uninterpreted by the static analysis.

Example operations and their postconditions:

- withdraw(A, x): decrements(balance(A), x)
- deposit(A, x): increments(balance(A), x)
- addAccount(A): true(account(A))
- removeAccount(A): false(account(A))

Static Analysis

- First finds all pairs of operations that produce contradicting effects
- Then for all other pairs query an SMT solver
- Reports pairs that are unsafe to execute concurrently

Handling unsafe operations

Two methods to handle unsafe operation pairs:

- Violation Repair (e.g. using CRDTs)
- Violation Avoidance (using reservations)

Reservations

Reservations are like locks.

There are several different types:

- Multi-level lock reservation
- Escrow reservation
- Multi-level mask reservation
- Partition lock reservation

Multi-level Lock Reservation

Their base lock mechanism:

- It refers to specific operations
- It allows for finer synchronization

Three types:

- Exclusive Allow (EA): Similar to labelling an operation **Red**
- Shared Allow (SA): Similar to EA, but many replicas can perform the op
- Shared Forbid (SF): Disallows any replica from performing an op

Multi-level Lock Reservation -- Example

Auction application with operations:

- place_bid
- close_auction
- query

Invariant:

- Auction closes once
- Highest bid at close time wins

With RedBlue:

- **Red**: place_bid, close_auction
- **Blue**: query

With Explicit Consistency:

- place_bid: SA, SF on close_auction
- close_auction: EA
- query: No lock

Escrow Reservation

- Useful for numeric bound invariants:
- For invariants x >= k
  - and initial value of x = x0
  - initial decrement rights: x0 - k
- Performing decrement(y) consumes y rights
- Replicas ask other replicas for rights to perform operations
- They have a technique for not “leaking” rights

Summary

- Main Idea: Reservations for fine grained synchronization
- Static analysis to identify unsafe operation pairs

Pros:

- Finer grain than other two
- Semi-automatic static analysis

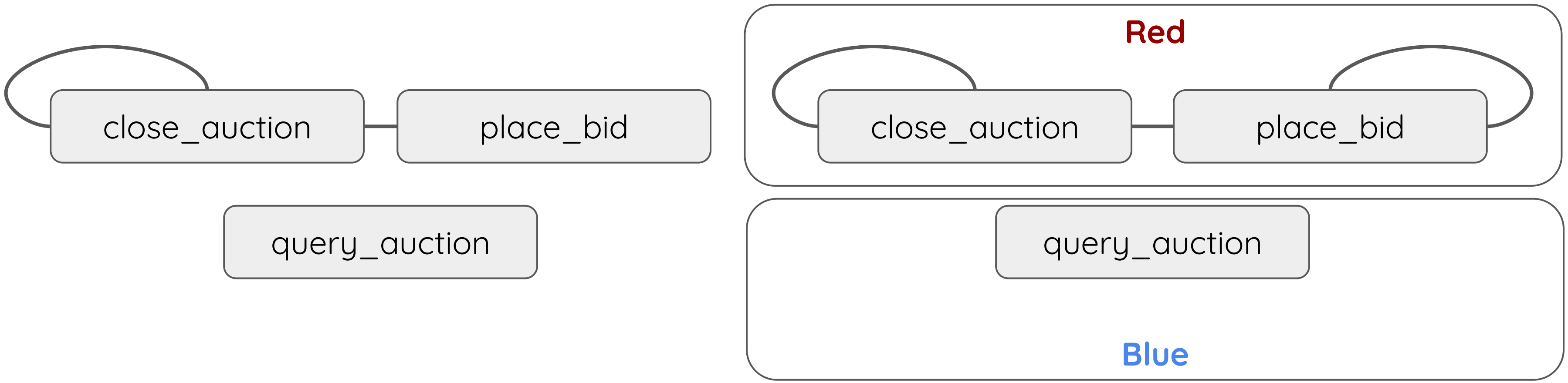
Cons:

- Reservations not formalized
- Analysis requires manual effort

(iii) Reasoning about Consistency

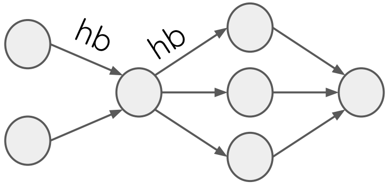
Main Idea

Token system $\mathcal{T} = (\mathsf{Token}, \bowtie)$, to model dependencies between operations.



Model

Executions are partial orders of events:



Operation semantics:

$$\forall o, \sigma.\, \mathcal{F}_o(\sigma) = (\mathcal{F}_o^{\mathsf{val}}(\sigma), \mathcal{F}_o^{\mathsf{eff}}(\sigma), \mathcal{F}_o^{\mathsf{tok}}(\sigma)).$$

where:

$$\mathcal{F}_o^{\mathsf{tok}}(\sigma) \;\in\; \mathcal{P}(\mathsf{Token})$$

$$\forall e, f \in E.\, \mathsf{tok}(e) \bowtie \mathsf{tok}(f) \implies (e \xrightarrow{\mathsf{hb}} f \vee f \xrightarrow{\mathsf{hb}} e).$$

Proving invariant preservation

We are given an invariant I over database states.

To show that invariant is preserved sequentially:

Generator

$$\forall \sigma.\ (\sigma \in I \implies \mathcal{F}_o^{\text{eff}}(\sigma)(\sigma) \in I).$$

To show that invariant is preserved in general:

Too imprecise!

$$\forall \sigma, \sigma'.\ (\sigma, \sigma' \in I \implies \mathcal{F}_o^{\text{eff}}(\sigma)(\sigma') \in I).$$

What about the tokens?

Guarantee relations

- Associate each token with a guarantee relation G(token)
- G(token) describes any state change that token can cause
- G0 relation of operations that don’t acquire any token

Guarantee relations -- Example

The standard banking example:

$$
\begin{aligned}
\mathcal{F}_{\mathsf{deposit}(a)}(\sigma) &= (\bot, (\lambda\sigma'.\,\sigma' + a), \emptyset) \\
\mathcal{F}_{\mathsf{interest}}(\sigma) &= (\bot, (\lambda\sigma'.\,\sigma' + 0.05 * \sigma), \emptyset) \\
\mathcal{F}_{\mathsf{query}}(\sigma) &= (\sigma, \mathsf{skip}, \emptyset) \\
\mathcal{F}_{\mathsf{withdraw}(a)}(\sigma) &= \text{if } \sigma \geq a \text{ then } (\checkmark, (\lambda\sigma'.\,\sigma' - a), \{\tau\}) \\
&\quad\ \text{else } (\text{✗}, \mathsf{skip}, \{\tau\})
\end{aligned}
$$

Has the following guarantee relations:

$$
\begin{aligned}
G(\tau) &= \{(\sigma, \sigma') \mid 0 \leq \sigma' < \sigma\}; \\
G_0 &= \{(\sigma, \sigma') \mid 0 \leq \sigma \leq \sigma'\}.
\end{aligned}
$$

State Based Proof rule

Given invariant, there exist G for all tokens and G0:

$$\frac{\begin{array}{l} \text{S1. } \sigma_{\mathsf{init}} \in I \\ \text{S2. } G_0(I) \subseteq I \wedge \forall \tau.\ G(\tau)(I) \subseteq I \\ \text{S3. } \forall o, \sigma, \sigma'.\ (\sigma \in I \wedge (\sigma, \sigma') \in (G_0 \cup G((\mathcal{F}_o^{\mathsf{tok}}(\sigma))^{\perp}))^*) \\ \qquad\qquad \implies (\sigma', \mathcal{F}_o^{\mathsf{eff}}(\sigma)(\sigma')) \in G_0 \cup G(\mathcal{F}_o^{\mathsf{tok}}(\sigma)) \end{array}}{\mathsf{Exec}(\mathcal{T}, \mathcal{F}) \subseteq \mathsf{eval}_{\mathcal{F}}^{-1}(I)}$$

Proof Rule Soundness

- They generalize the state based rule to refer to events
- They prove that:
  - the event-based rule is sound
  - the state-based rule is a specialization of it
- It follows that the state-based rule is sound

Summary

- Main idea: Conflict relation for fine grained synchronization control
- Sound proof rule that establishes invariant preservation

Pros:

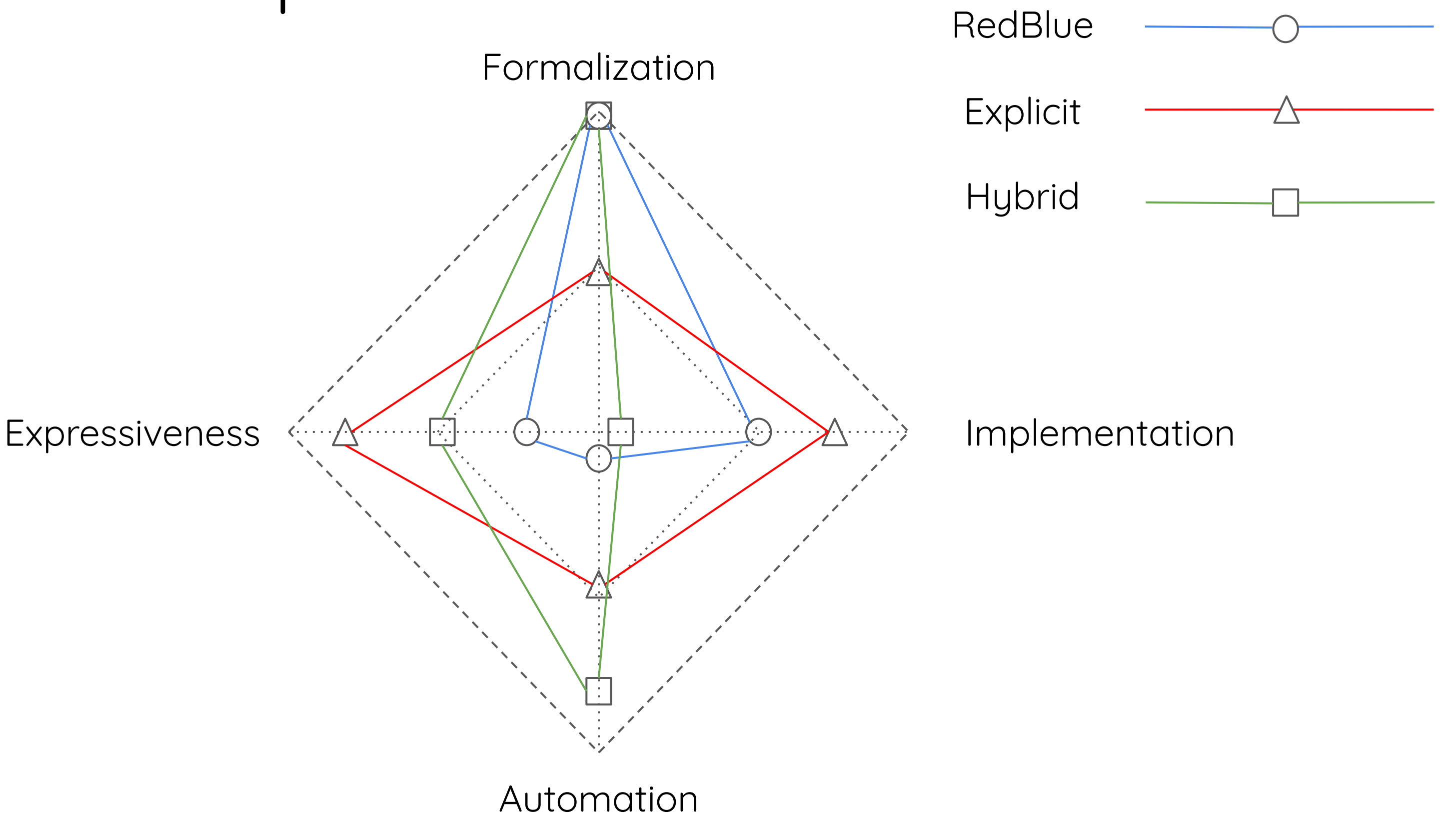
- Finer grain than RedBlue
- Fully formalized
- Automatic

Cons:

- Guarantee relation manual
- Less general than Explicit

Conclusion

Qualitative Comparison



Future Work

- Better Automation/Reduced user input
- More expressive correctness conditions
- Dropping the causality assumption
- Hybrid Consistency Data Types