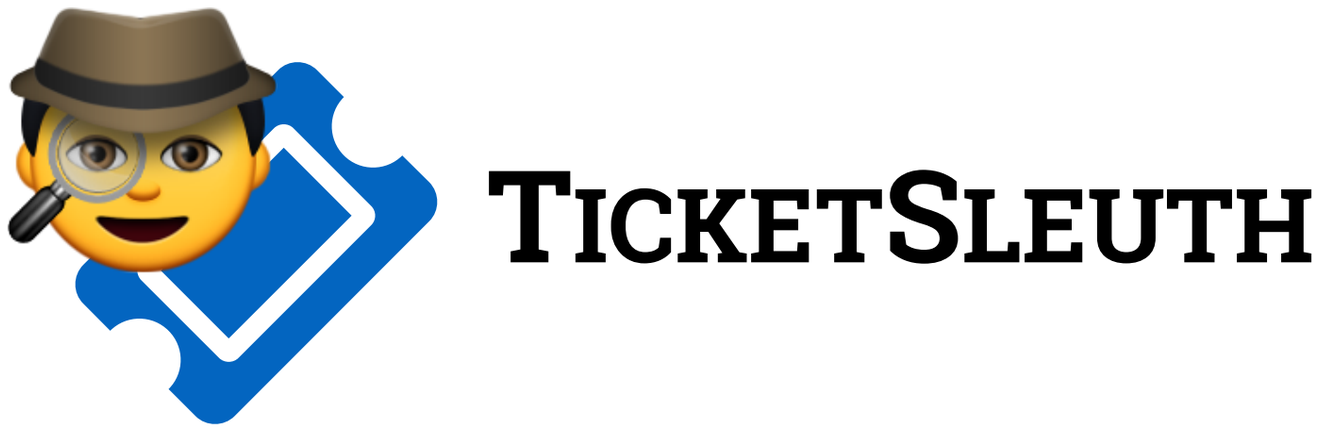
Mitigating Contention in Distributed Systems

Generals Exam

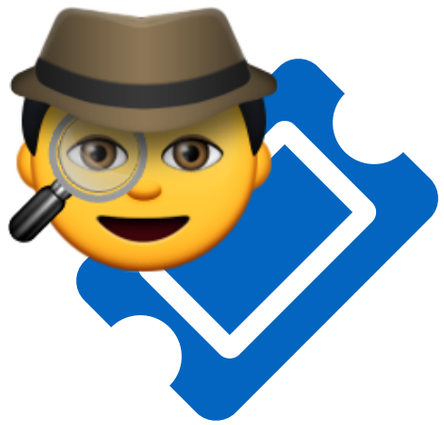
Brandon Holt

Committee

Luis Ceze, Mark Oskin, Dan Ports, Jevin West



TicketSleuth



Customer Ratings

Current Version | All Versions

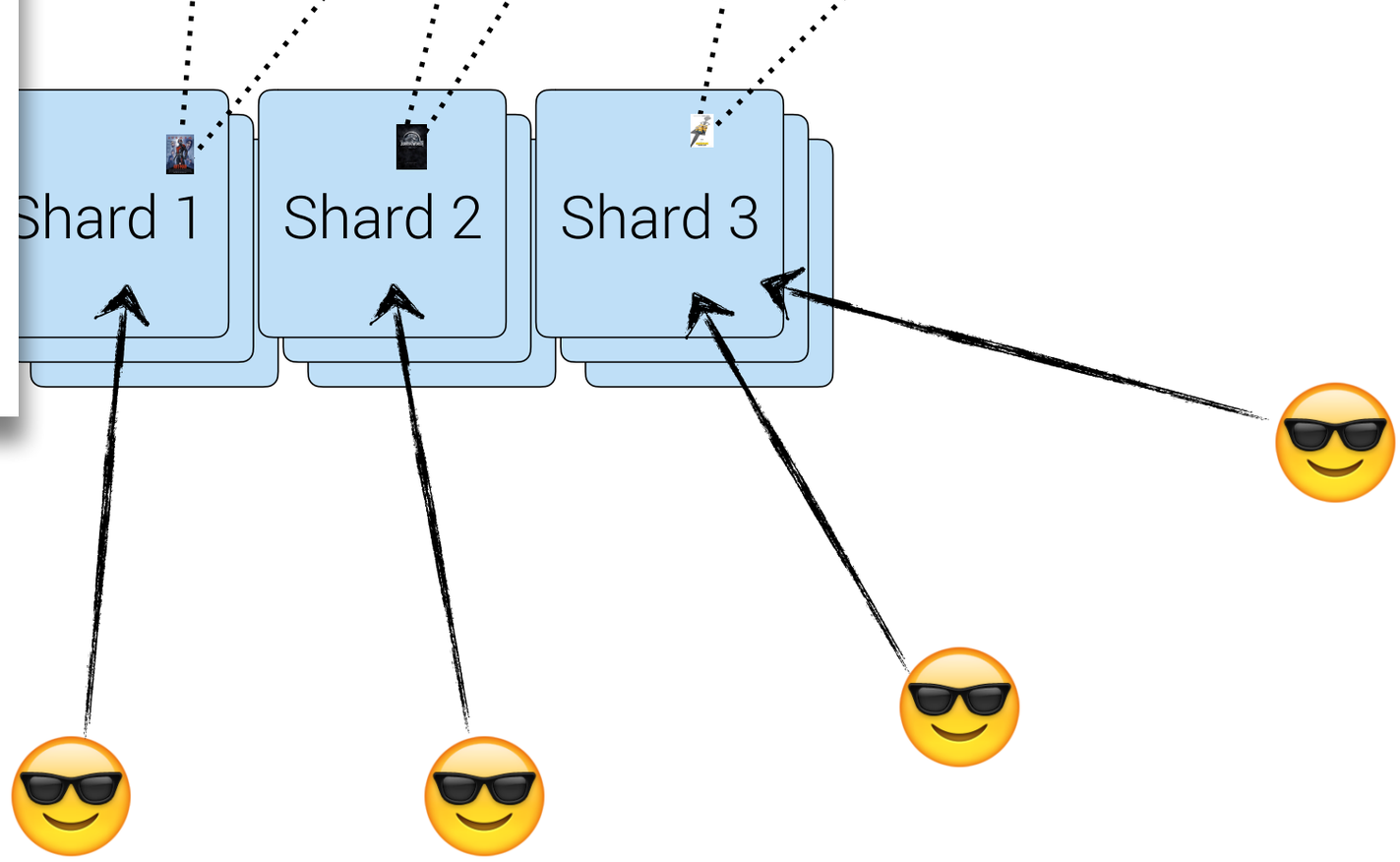
Average Rating: ★★★★★ 12,725 Ratings

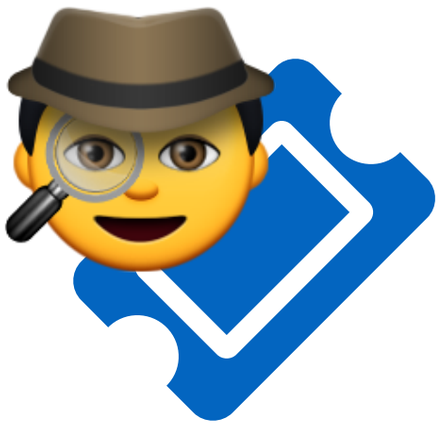
Click to rate: ★★★★★

Customer Reviews

Write a Review | App Support

TicketSleuth is #winning!! ★★★★★
by bholt1111 – Nov 13, 2015





FANDANGO

AMC THEATRES

'Star Wars' Presales Crash

www.forbes.com/sites/hayleycuccinello/2015/10/20/star-wars-presales-crash-ticketing-sites-sets-records-for-fandango/

holt.bg

Forbes / Media & Entertainment

The Little Black Book of Billionaire Secrets

OCT 20, 2015 @ 03:55 PM 138,858 VIEWS

'Star Wars' Presales Crash Ticketing Sites, Set Record for Fandango

Hayley C. Cuccinello, CONTRIBUTOR

I write about television and theater

FOLLOW ON FORBES (11)

Opinions expressed by Forbes Contributors are their own.

FULL BIO

COMMENT NOW

SHARE

What causes contention?

Power Laws

Focus on few records.

Realtime Events

Focus in time.

Interactivity

Heavy write load → synchronize or conflict

Introduction: Contention

✓ Sources of contention.

➢ 3 broad approaches to mitigating contention.

Background: Trading off consistency

Consistency for architects.

Ordering and visibility constraints.

Dealing with uncertainty: **converged state vs. staleness**

Programming with tradeoffs.

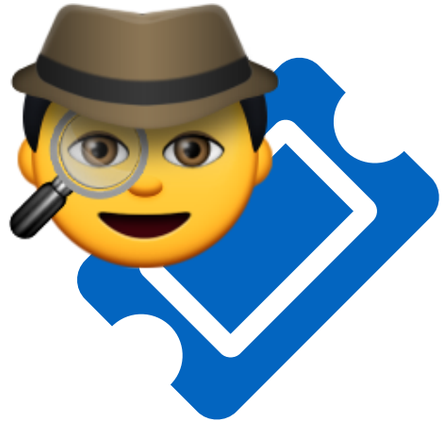
Proposal: Disciplined Inconsistency

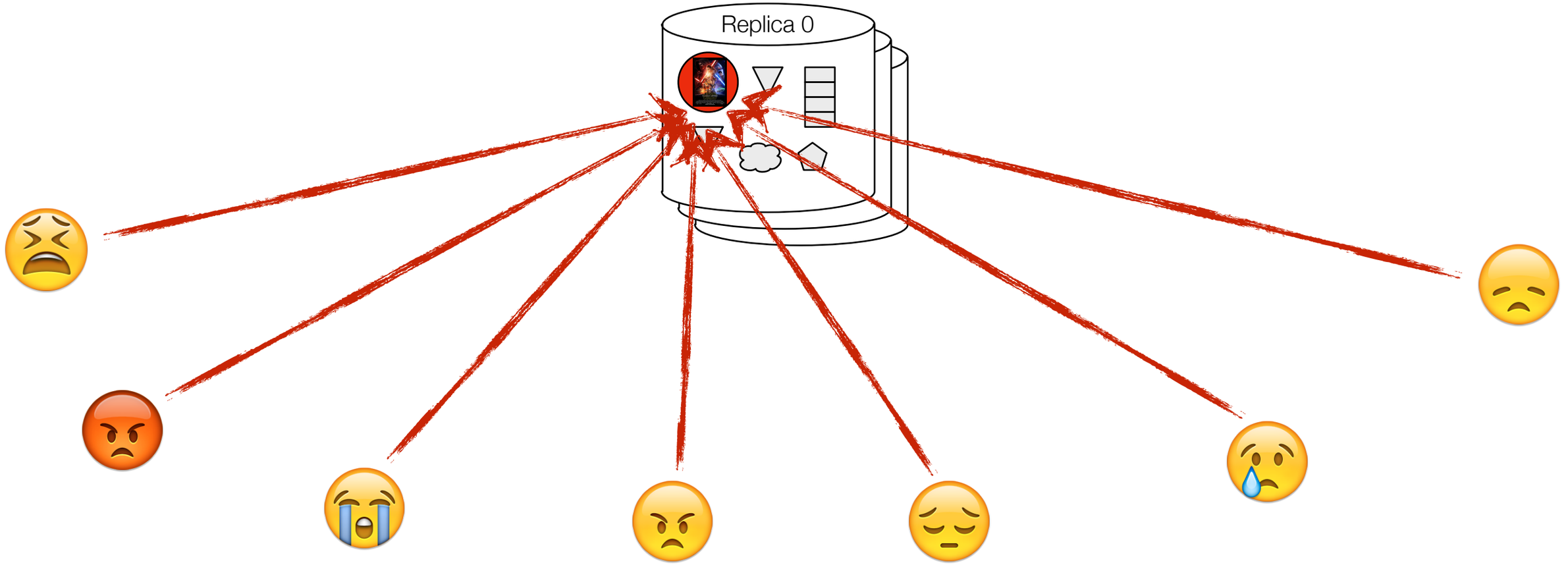
Make tradeoffs *explicit* and *safe* with types.

Reason about *values* rather than *ordering.*

Synthesize synchronization/enforcement logic.

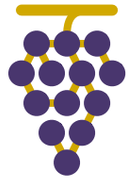
Mitigating contention



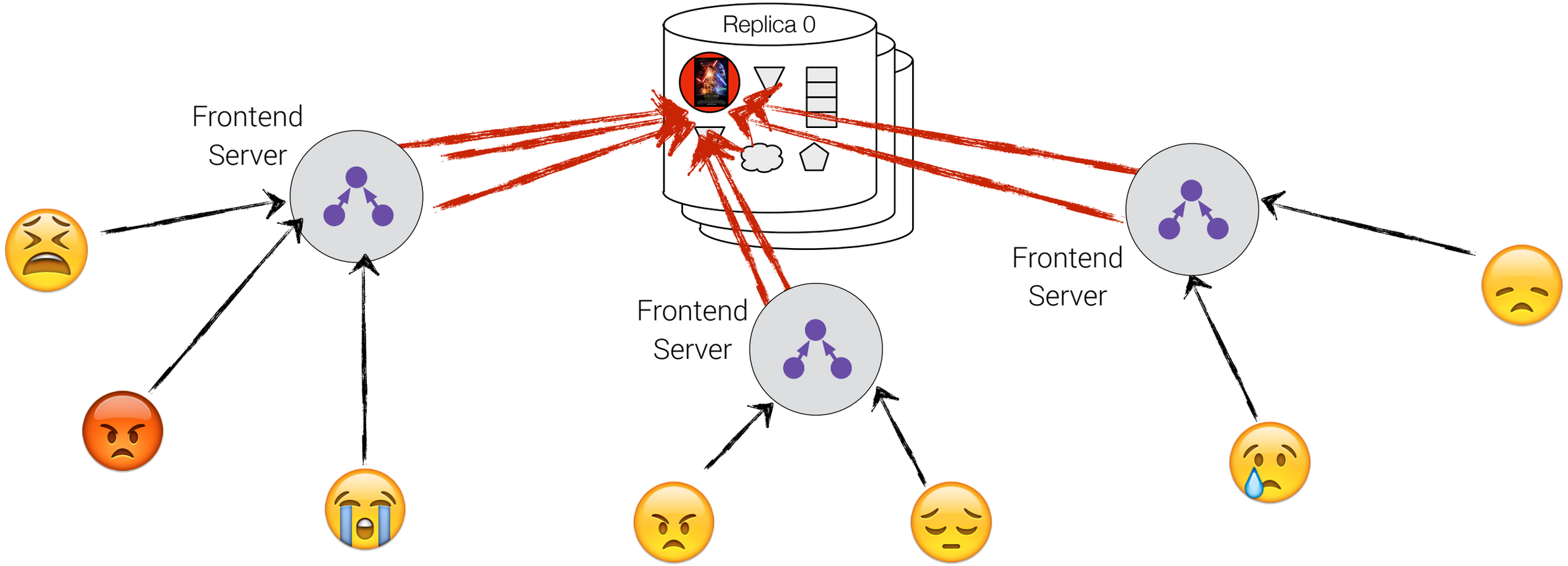


Mitigating contention

① Offload synchronization to clients.



Flat Combining Global Shared Data Structures
B. Holt, et al. — *PGAS'13*

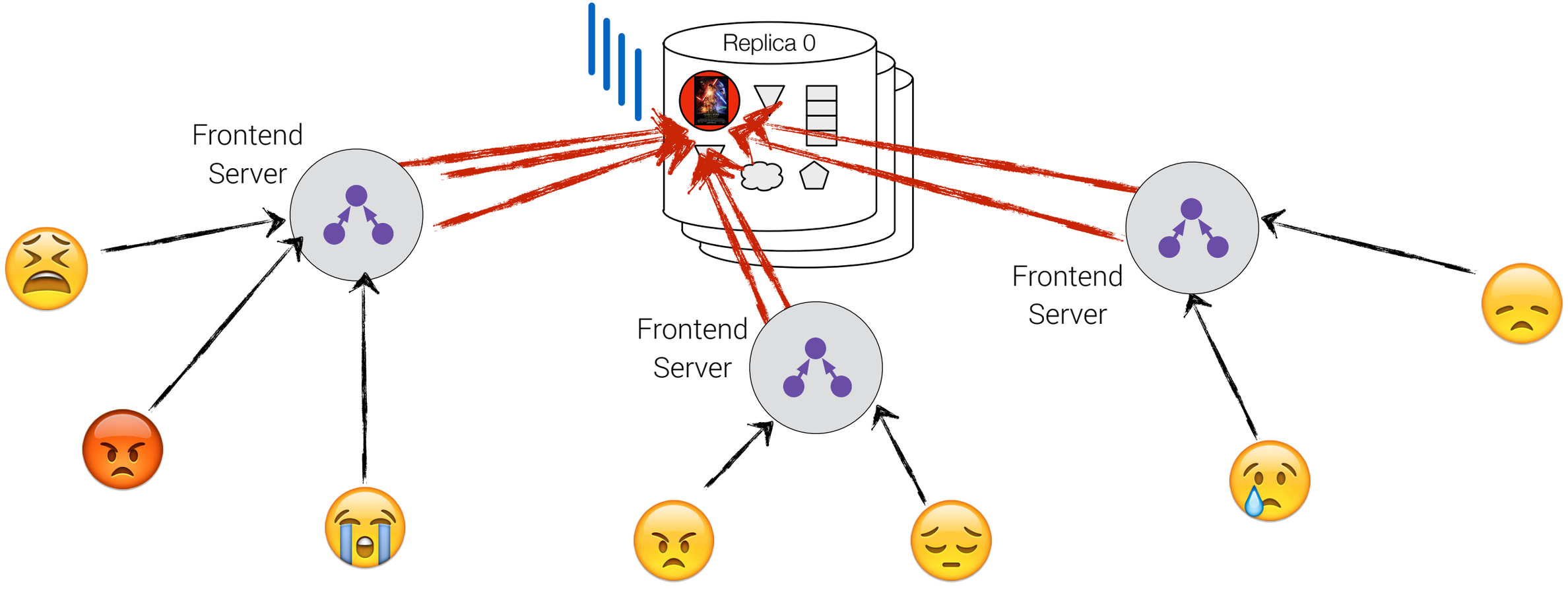


Mitigating contention

① Offload synchronization to clients.

② Unlock safe concurrency within the record.

Claret: Avoiding Contention in Distributed Transactions with Abstract Data Types
B. Holt, et al. – *In submission.*

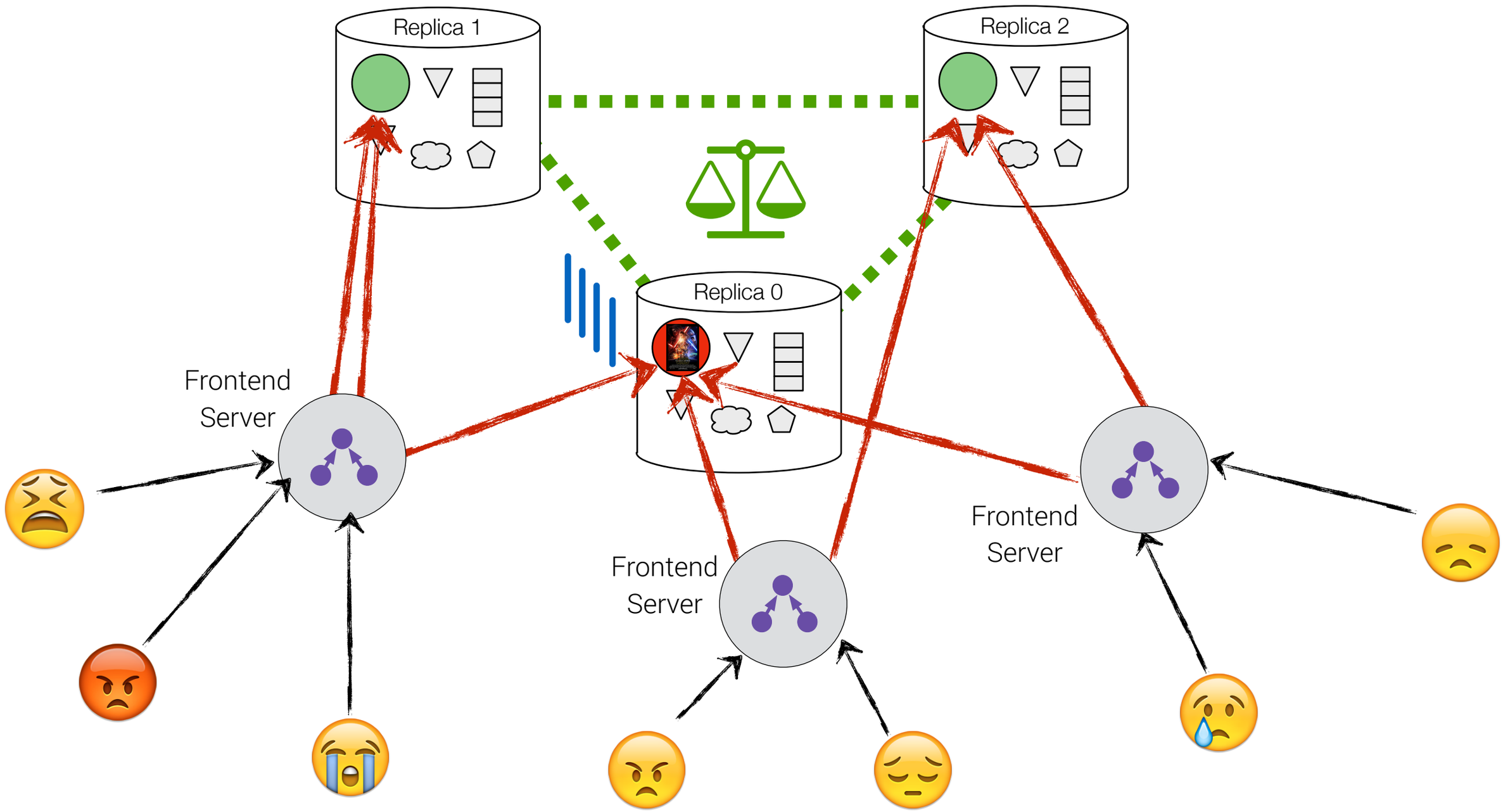


Mitigating contention

① Offload synchronization to clients.

② Unlock safe concurrency within the record.

③ Trade off consistency with weak replication.

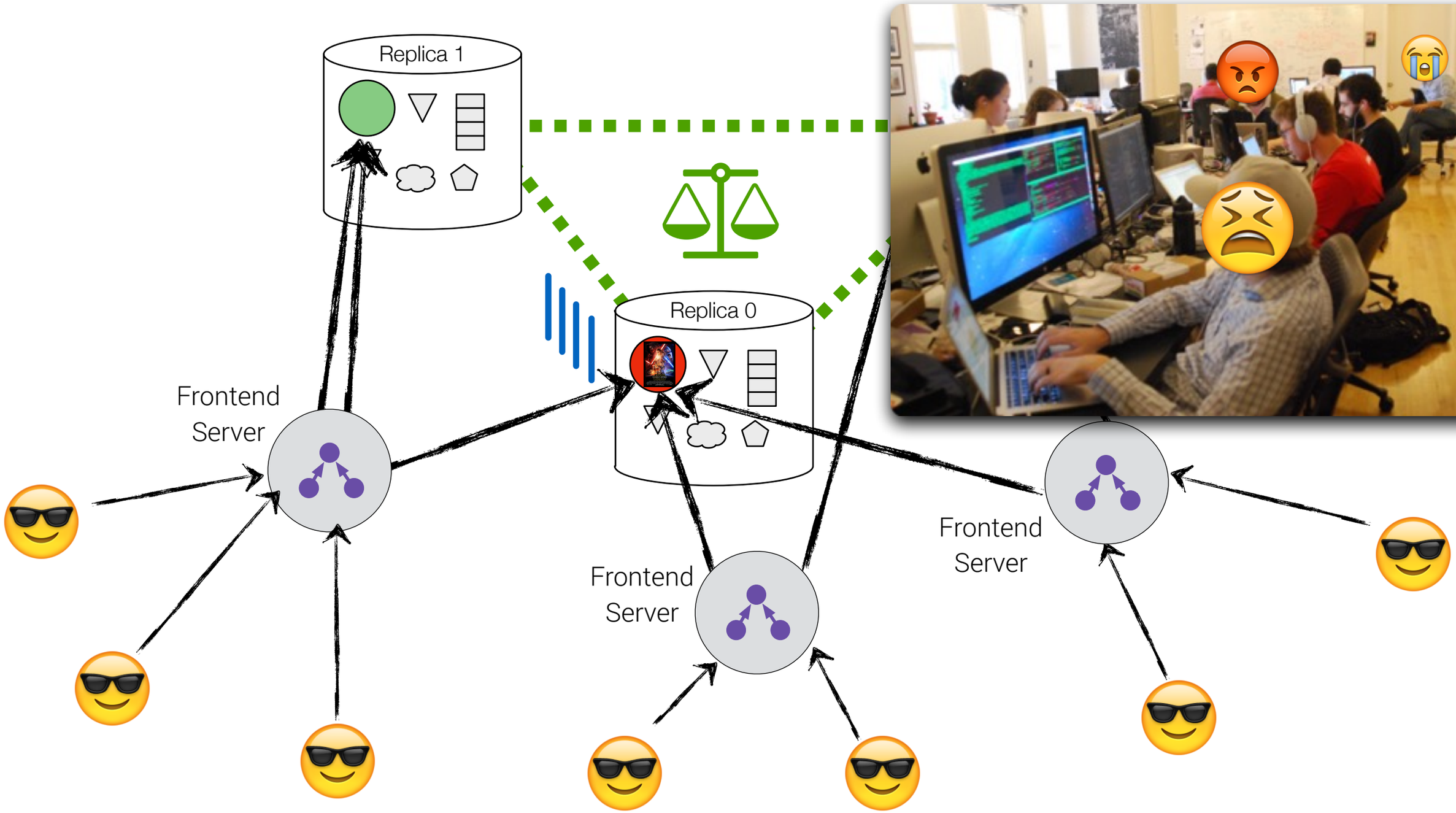


Mitigating contention

① Offload synchronization to clients.

② Unlock safe concurrency within the record.

③ Trade off consistency with weak replication.



Mitigating contention

③ Trade off consistency with weak replication.

How should coordination be specified and enforced?

How can programming models help make these trade-offs?

Introduction: Contention

- ✓ Sources of contention.
- ✓ 3 broad approaches to mitigating contention.

Background: Trading off consistency

- Consistency for architects.
- Ordering and visibility constraints.
- Dealing with uncertainty.
- Programming with tradeoffs.

Proposal: Disciplined Inconsistency

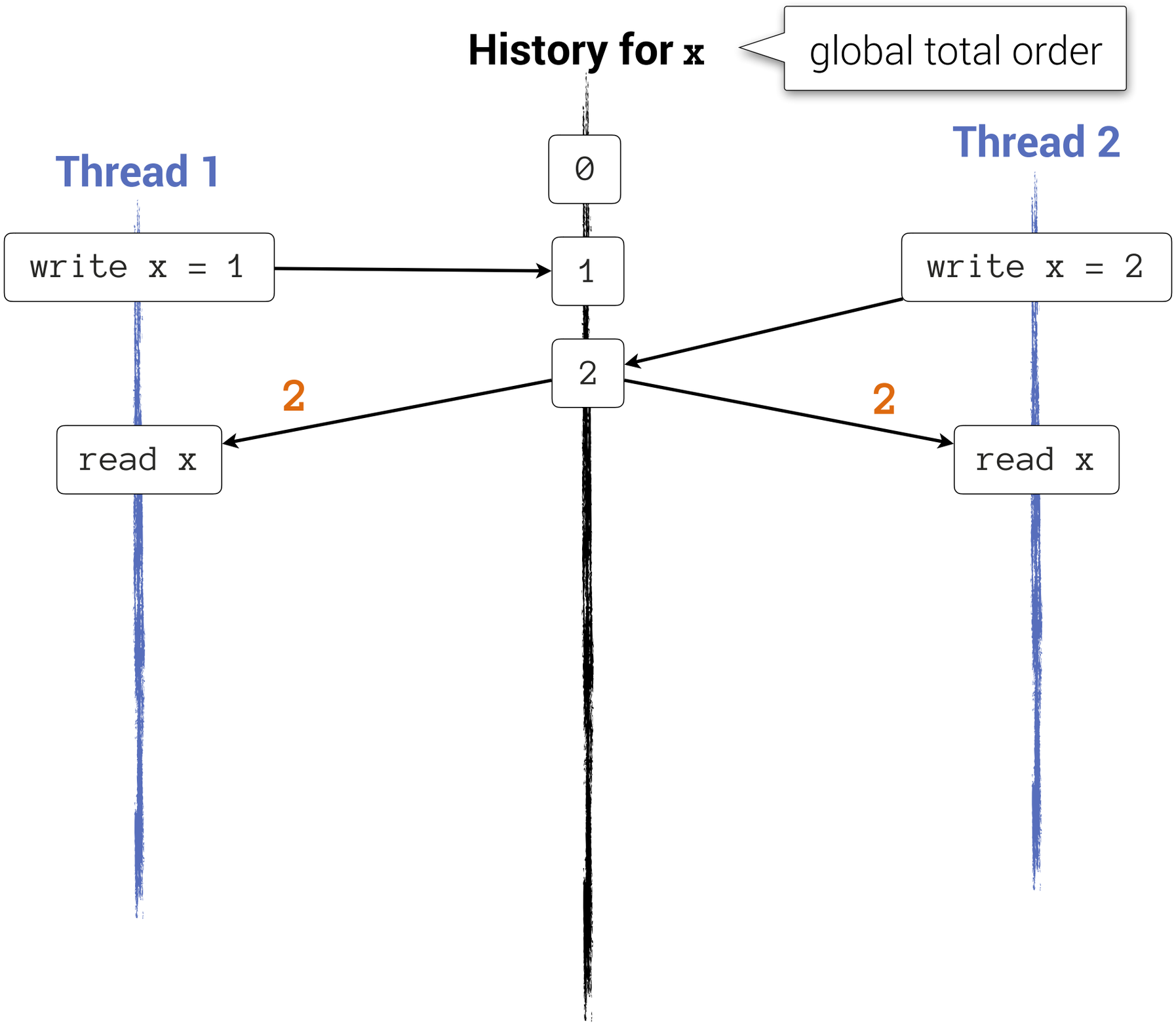
Make tradeoffs *explicit* and *safe* with types.

Reason about *values* rather than *ordering.*

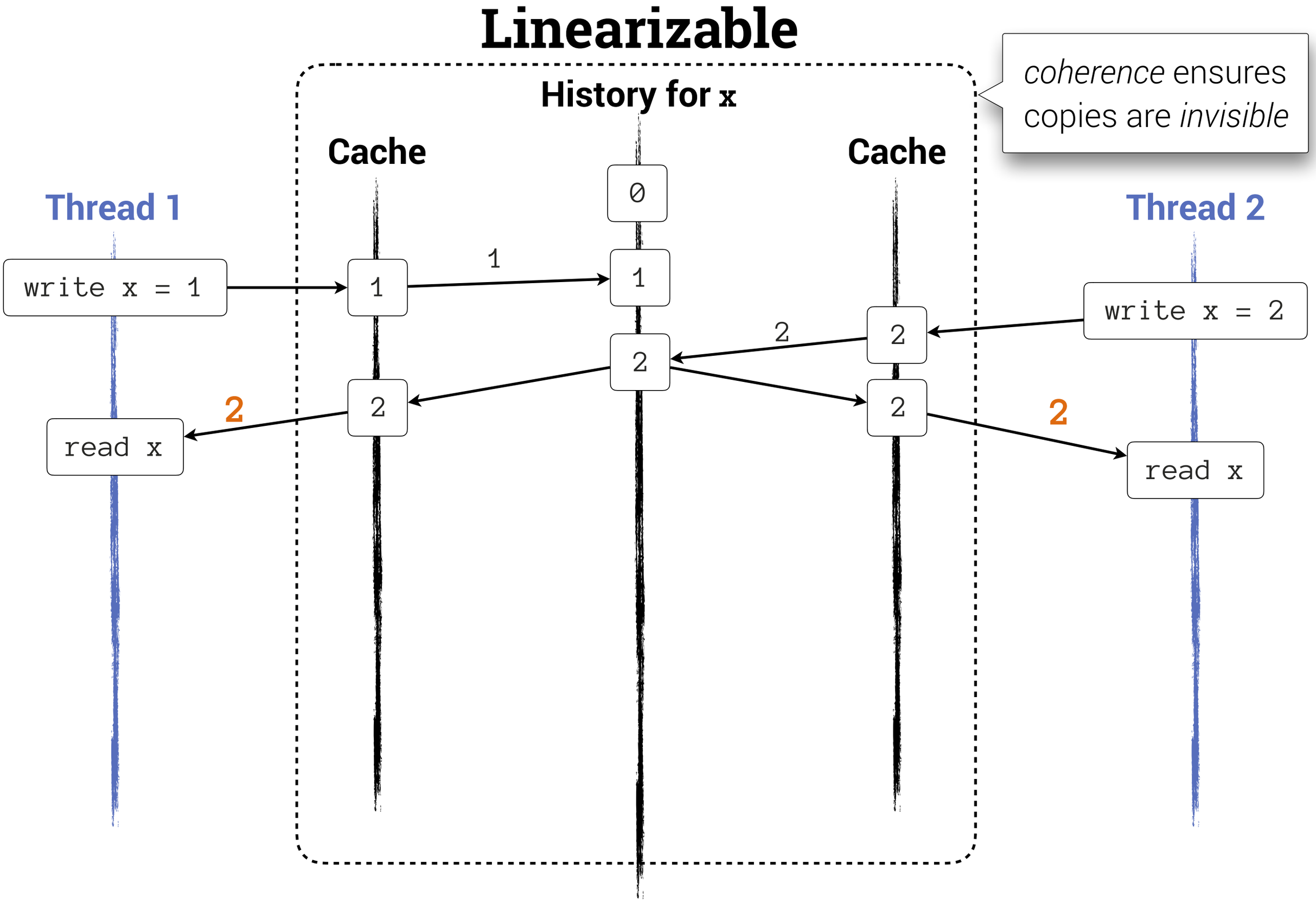
Synthesize synchronization/enforcement logic.

Consistency for architects

Linearizable

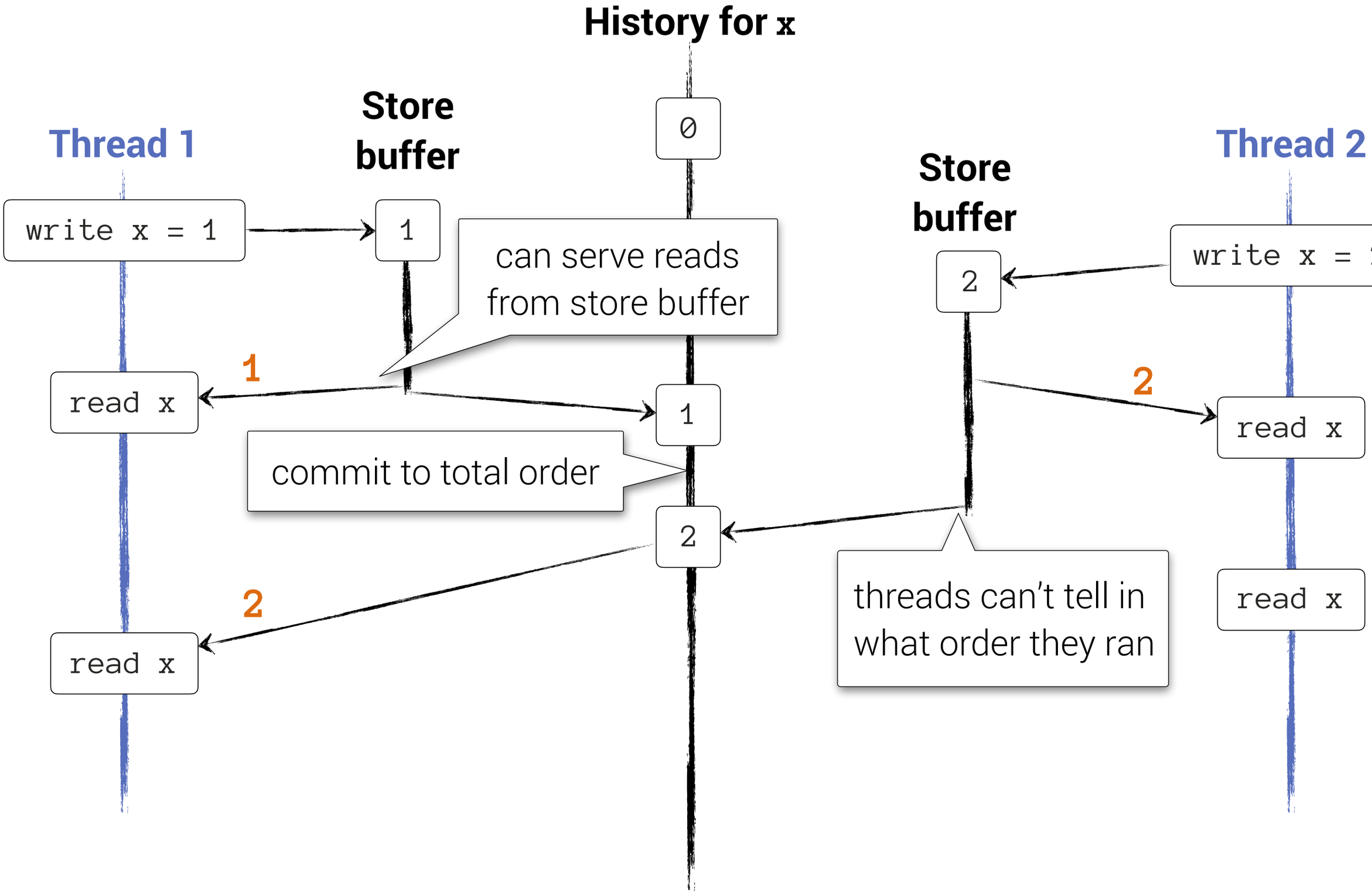


Consistency for architects



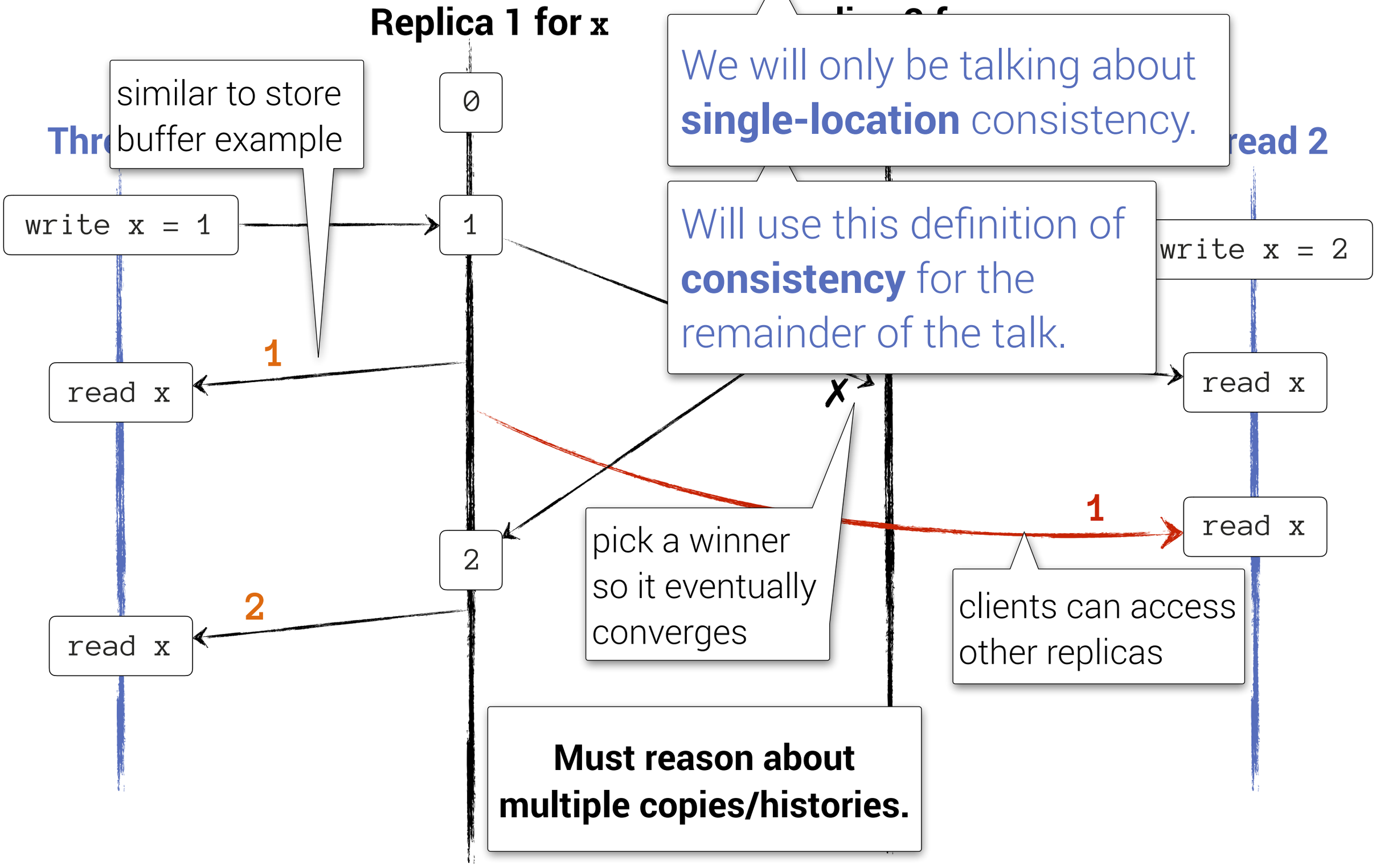
Consistency for architects

Linearizable



Consistency for architects

Weak consistency



Introduction: Contention

- ✓ Sources of contention.
- ✓ 3 broad approaches to mitigating contention.

Background: Trading off consistency

- ✓ Consistency for architects.
- ➢ Ordering and visibility constraints.

Dealing with uncertainty.

Programming with tradeoffs.

Proposal: Disciplined Inconsistency

Make tradeoffs *explicit* and *safe* with types.

Reason about *values* rather than *ordering.*

Synthesize synchronization/enforcement logic.

Trading off consistency

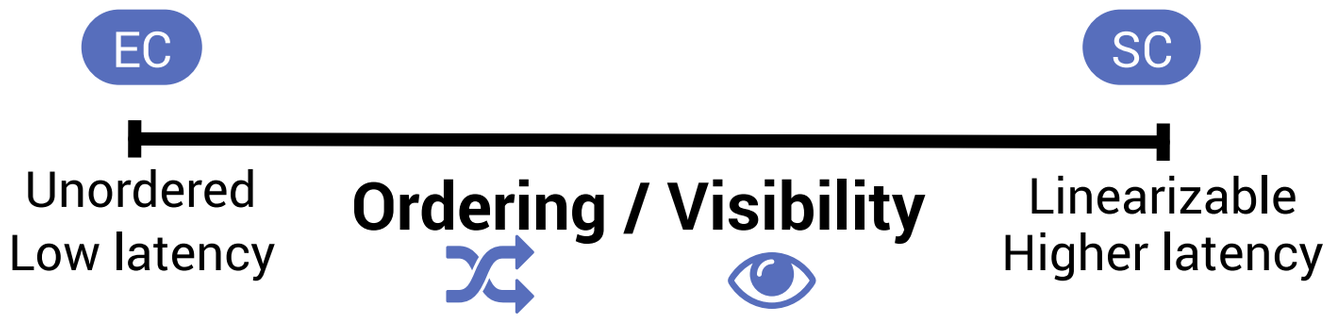
Ordering constraints

Eventual Consistency

- Only guarantee: if updates stop, replicas will eventually reflect the same state.
- Fastest response time.
- *High availability, lowest latency*

Strong Consistency

- Herlihy's *linearizability* (only talking about single-location)
- Total global order
- *Not* highly available



Ordering constraints

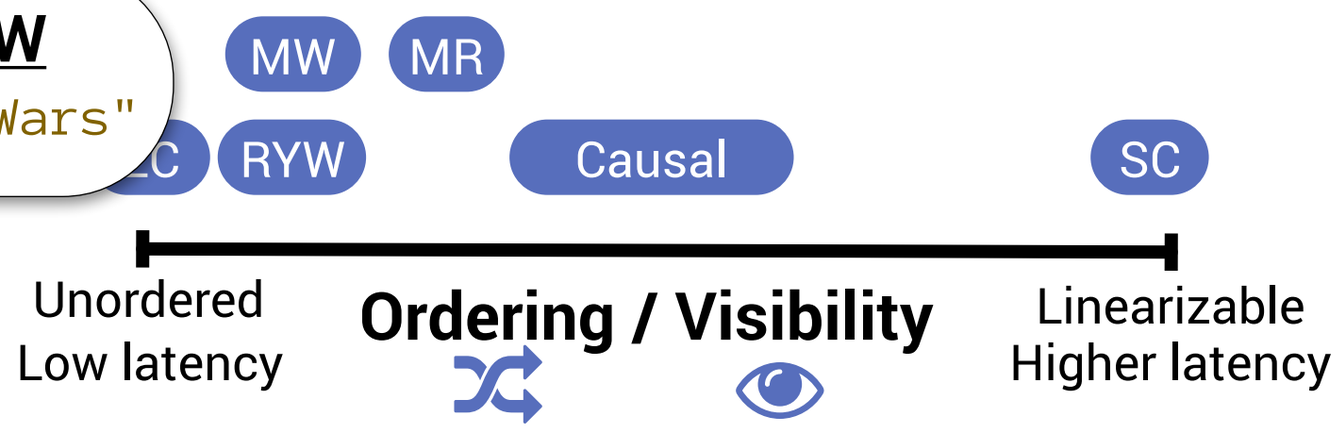
Other weak consistency

- Force coordination in certain cases,
  or restrict which replicas clients can talk to
  (e.g. "sticky sessions")
- But, easier to reason about:

```
# put ticket in my cart
put("bholt:cart", "Star Wars")
# view my cart
cart = get("bholt:cart")
```

EC
(empty)
"Star Wars"

RYW
"Star Wars"



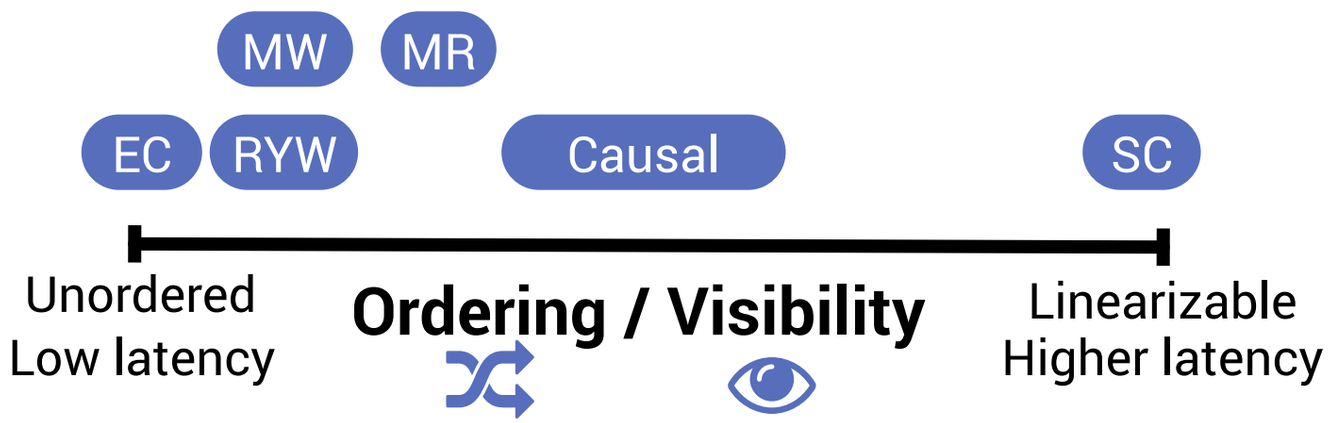
Trading off consistency

Ordering constraints

- Consistency models are **too coarse-grained**.
- Only enforce constraints where **necessary**.

68,144,938 followers

178 followers

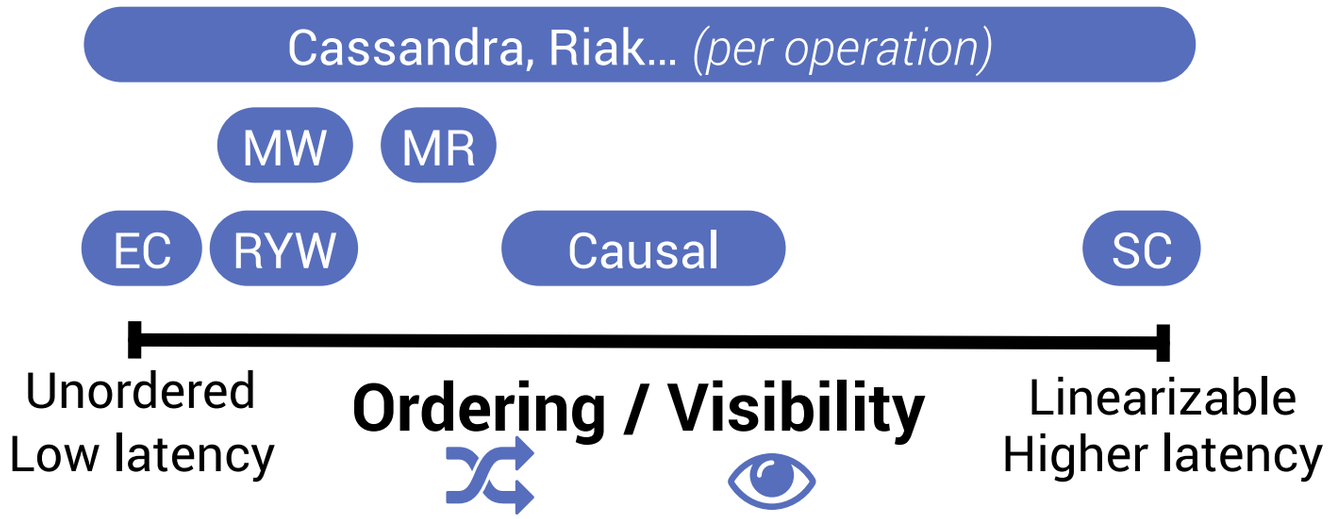


think about a better example? would be better to have something about different parts of app, since that's more typical.

Ordering constraints

Per-Operation Consistency Level

- Supported by many datastores (e.g. Cassandra, Riak…)
- Only use stronger consistency where necessary.
- Selecting the right one is *error-prone, non-modular.*



add example of difficulty of using multiple consistency levels? get one from Quelea paper.

Ordering constraints

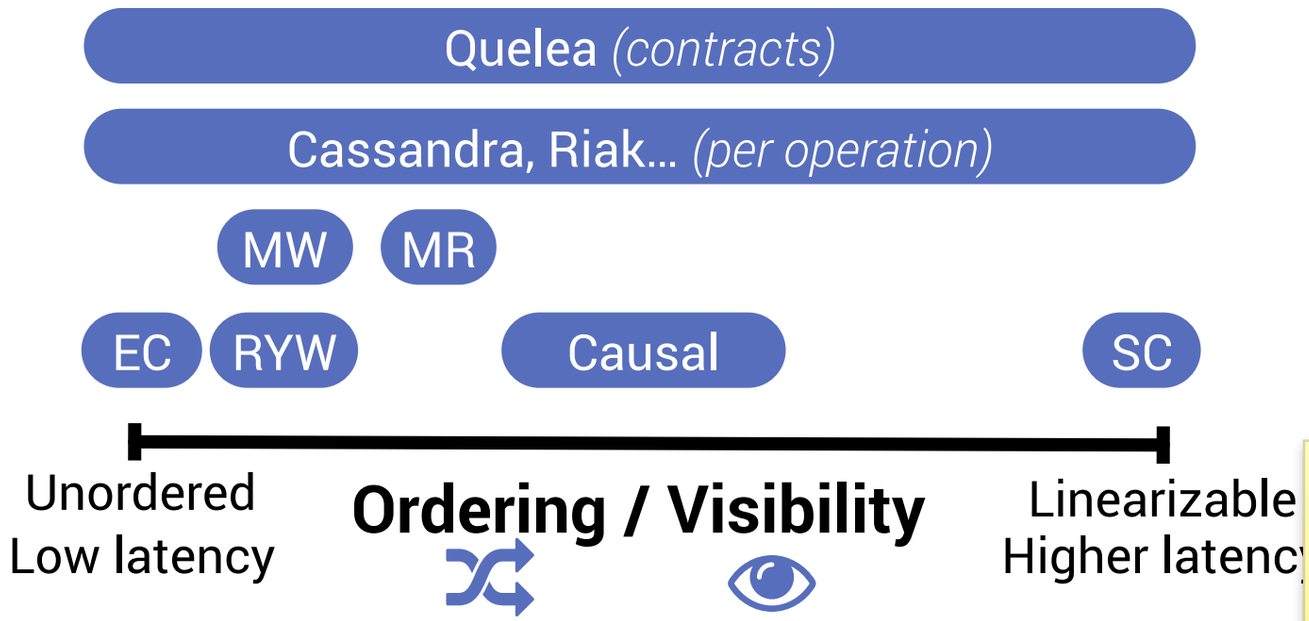
Quelea

- Explicit ordering & visibility *contracts:*

$$\forall(a : \texttt{sellTicket}).\ \texttt{sameobj}(a, \eta) \Rightarrow a = \eta \ \vee\ \mathsf{vis}(a, \eta) \ \vee\ \mathsf{vis}(\eta, a)$$

- *Automatically* choose correct consistency level.
- But low-level constraints not much better.

Declarative programming over eventually consistent data stores. (PLDI'15)
KC Sivaramakrishnan, Gowtham Kaki, and Suresh Jagannathan.



Add example from Fig. 3 from Quelea paper showing how this prevents over-selling.

Ordering constraints

Indigo

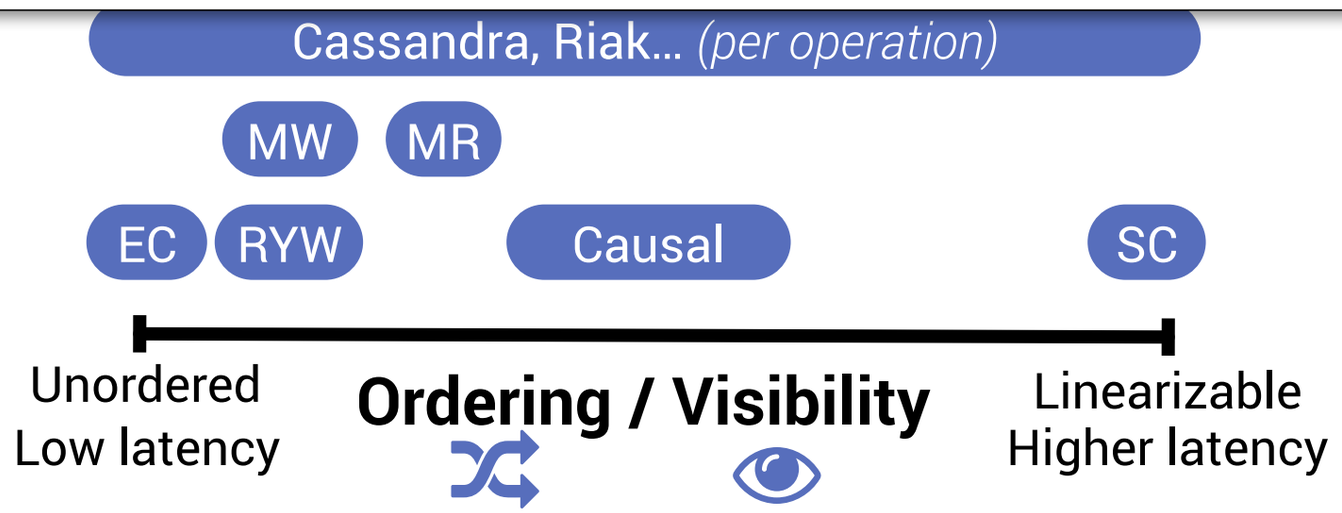
- Annotate *application-level invariants*:

```
@Invariant("forall(Movie : m) :- tickets(m) >= 0")
public interface MovieTickets {
    @PostCondition_Decrements("tickets(m, 1)")
    void sellTicket(Movie m);
}
```

- Synthesize coordination logic (*reservations*).

- But: manually annotate abstract state & behavior.

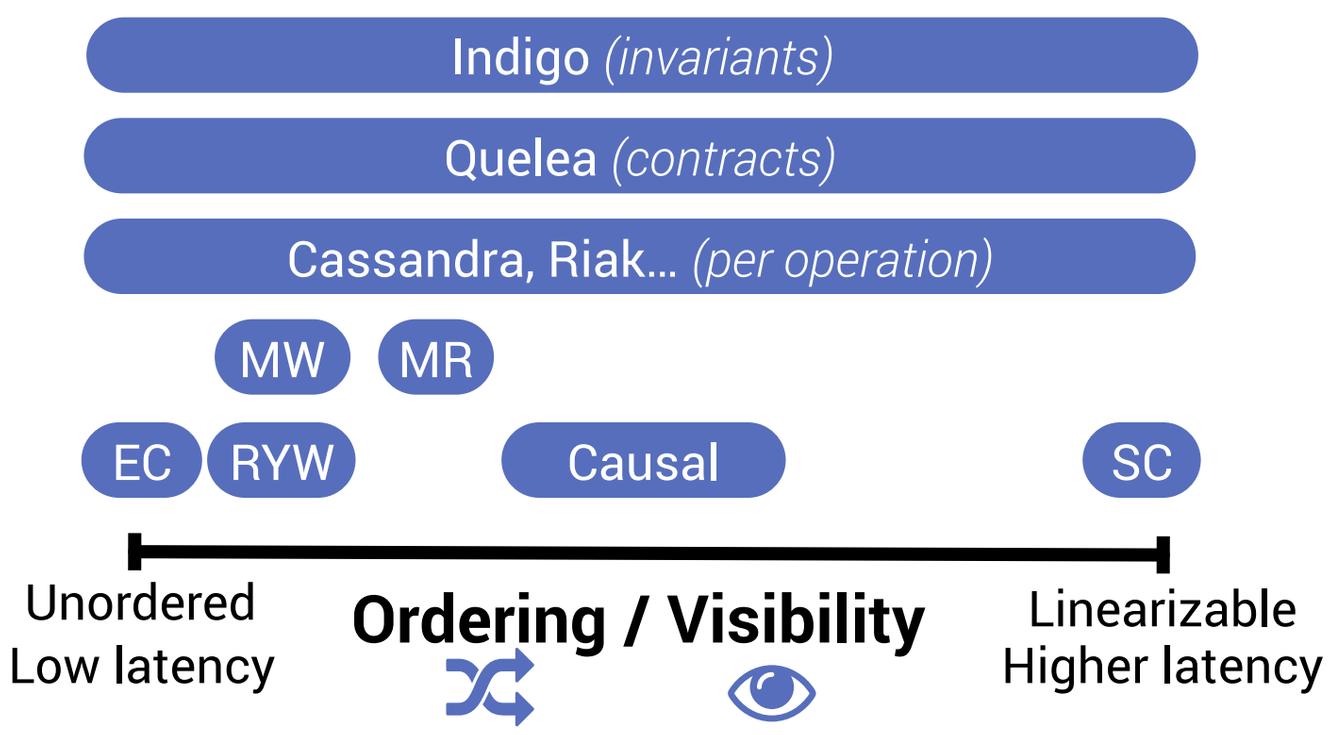
Putting consistency back into eventual consistency. (EuroSys'15)
V. Balegas, … , M. Shapiro.



Trading off consistency

Ordering constraints

- Consistency models are **too coarse-grained**.
- Only enforce constraints where **necessary**.
- **Synthesize** fine-grained synchronization.



Introduction: Contention

✓ Sources of contention.

✓ 3 broad approaches to mitigating contention.

Background: Trading off consistency

✓ Consistency for architects.

✓ Ordering and visibility constraints.

➢ Dealing with uncertainty: **convergence vs. staleness**

Programming with tradeoffs.

Proposal: Disciplined Inconsistency

Make tradeoffs *explicit* and *safe* with types.

Reason about *values* rather than *ordering.*

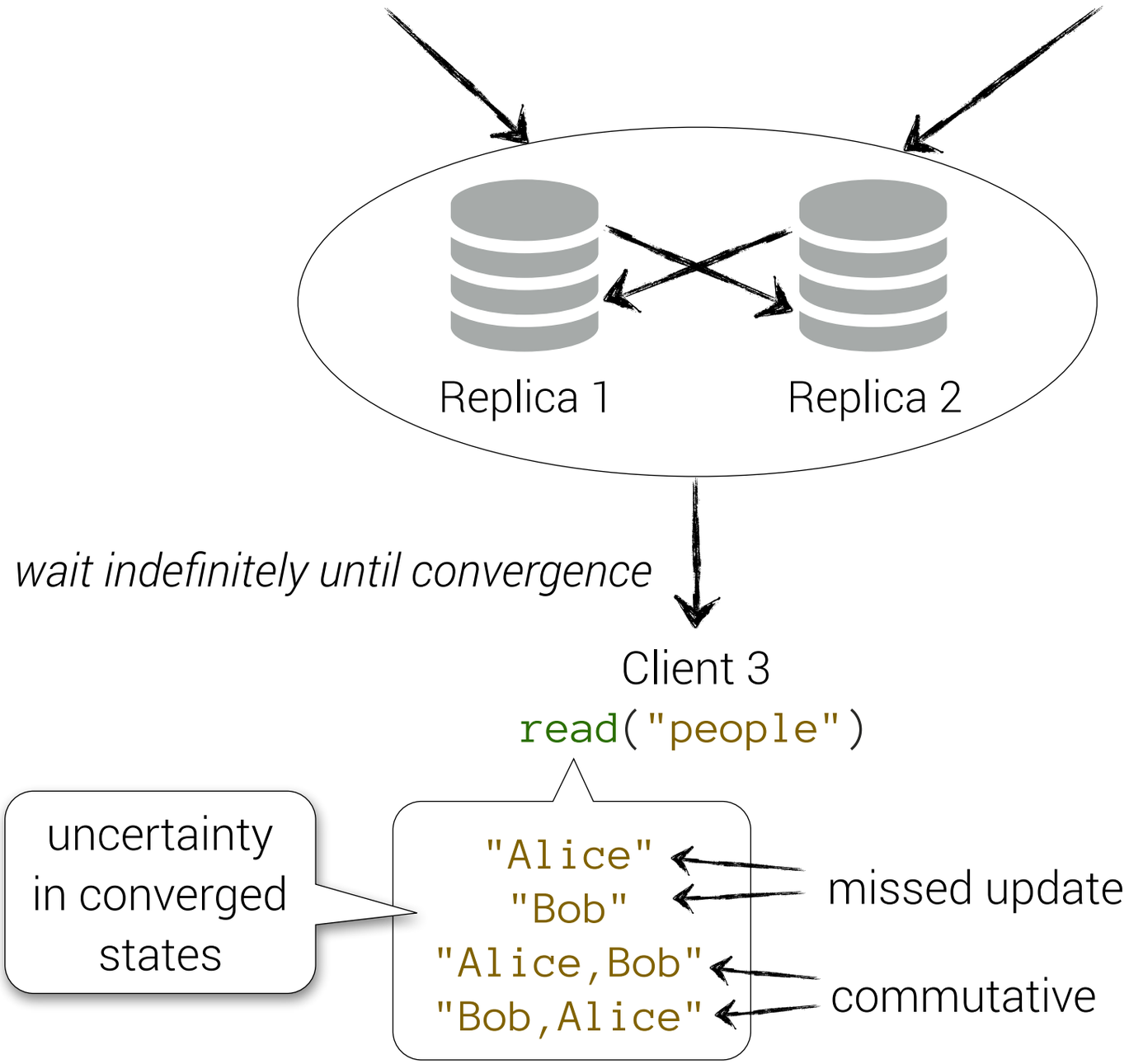
Synthesize synchronization/enforcement logic.

Uncertainty

Converged states

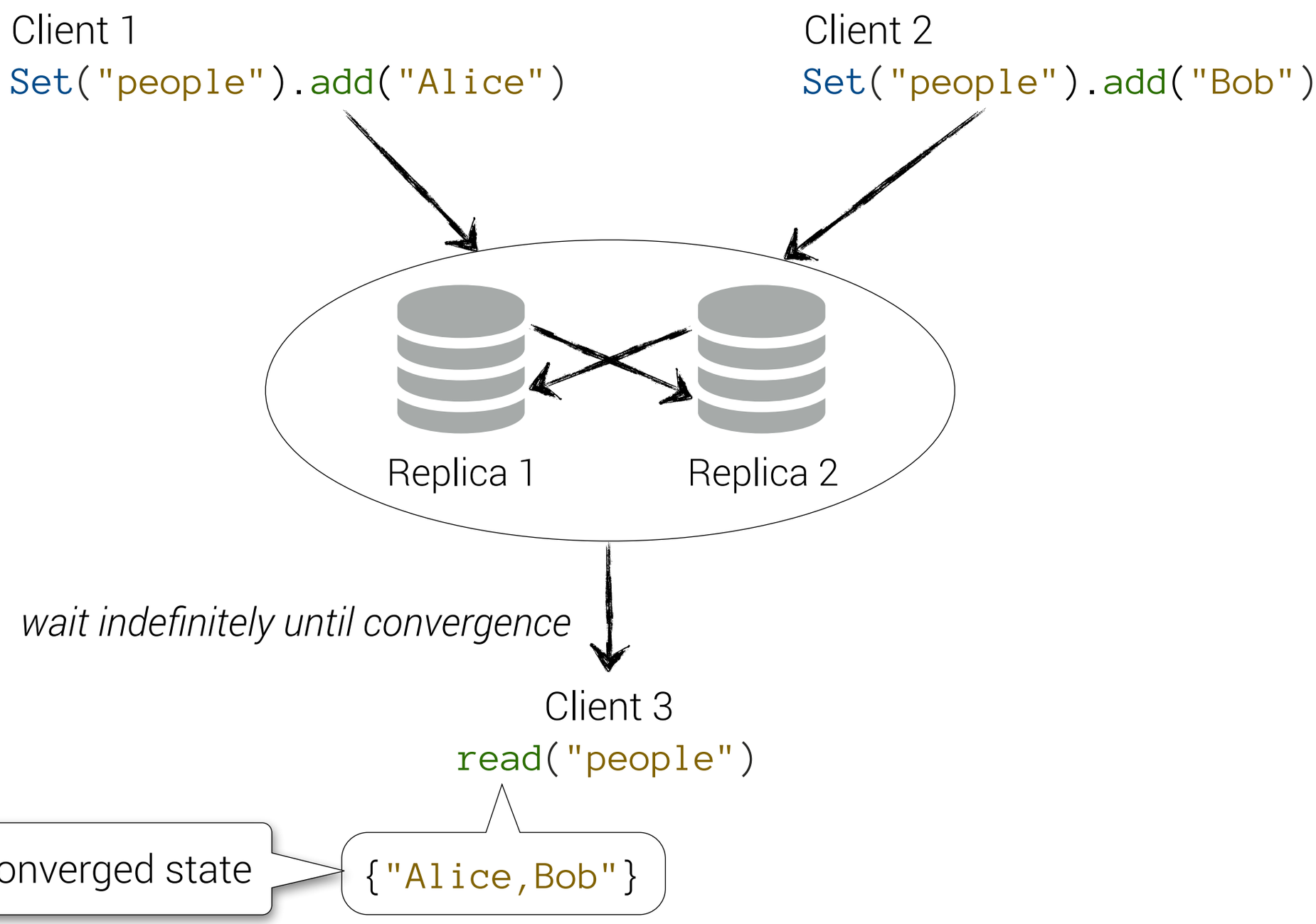
Client 1
```
x = read("people")
write("people", x+"Alice")
```

Client 2
```
x = read("people")
write("people", x+"Bob")
```



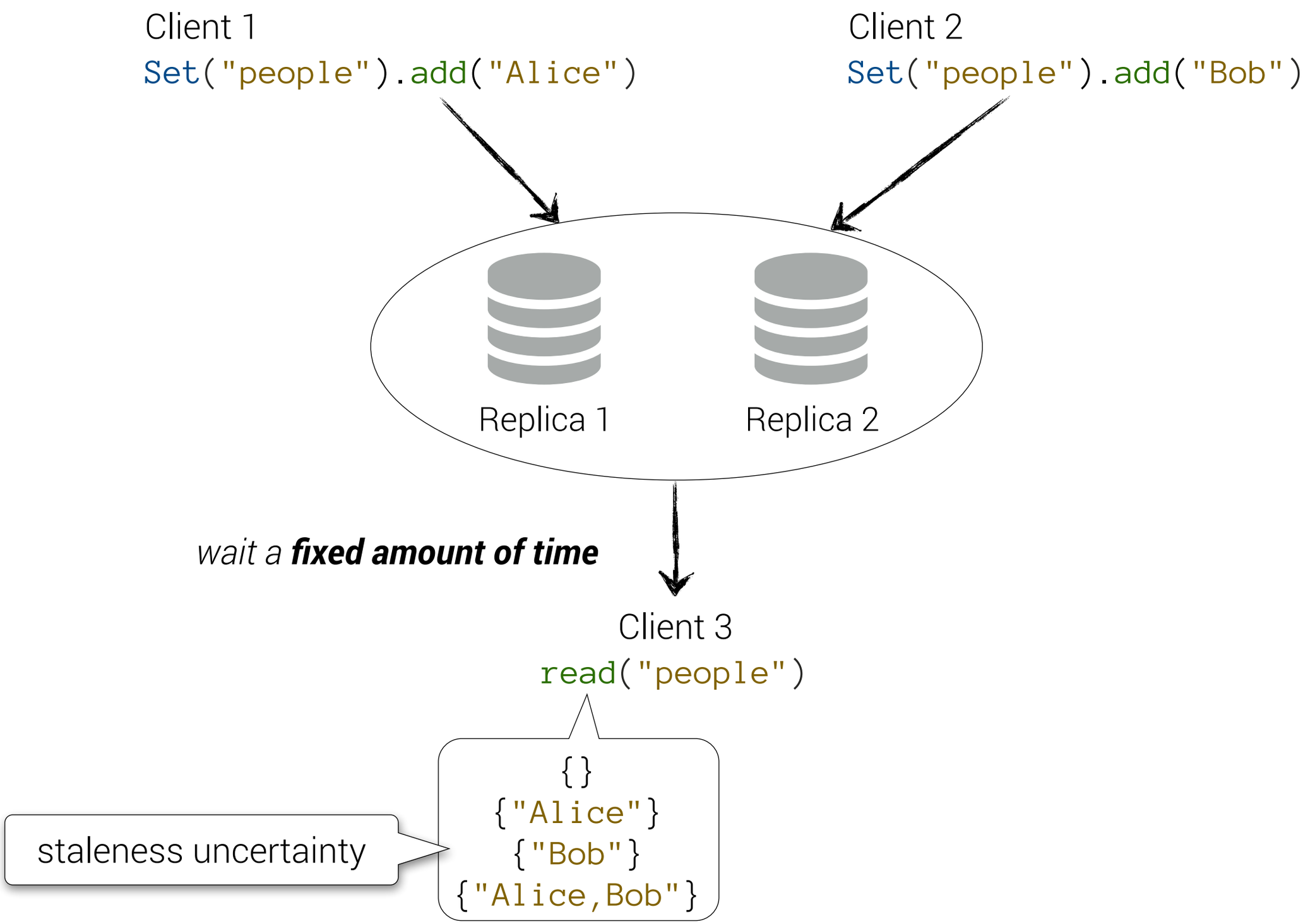
Uncertainty

Converged states



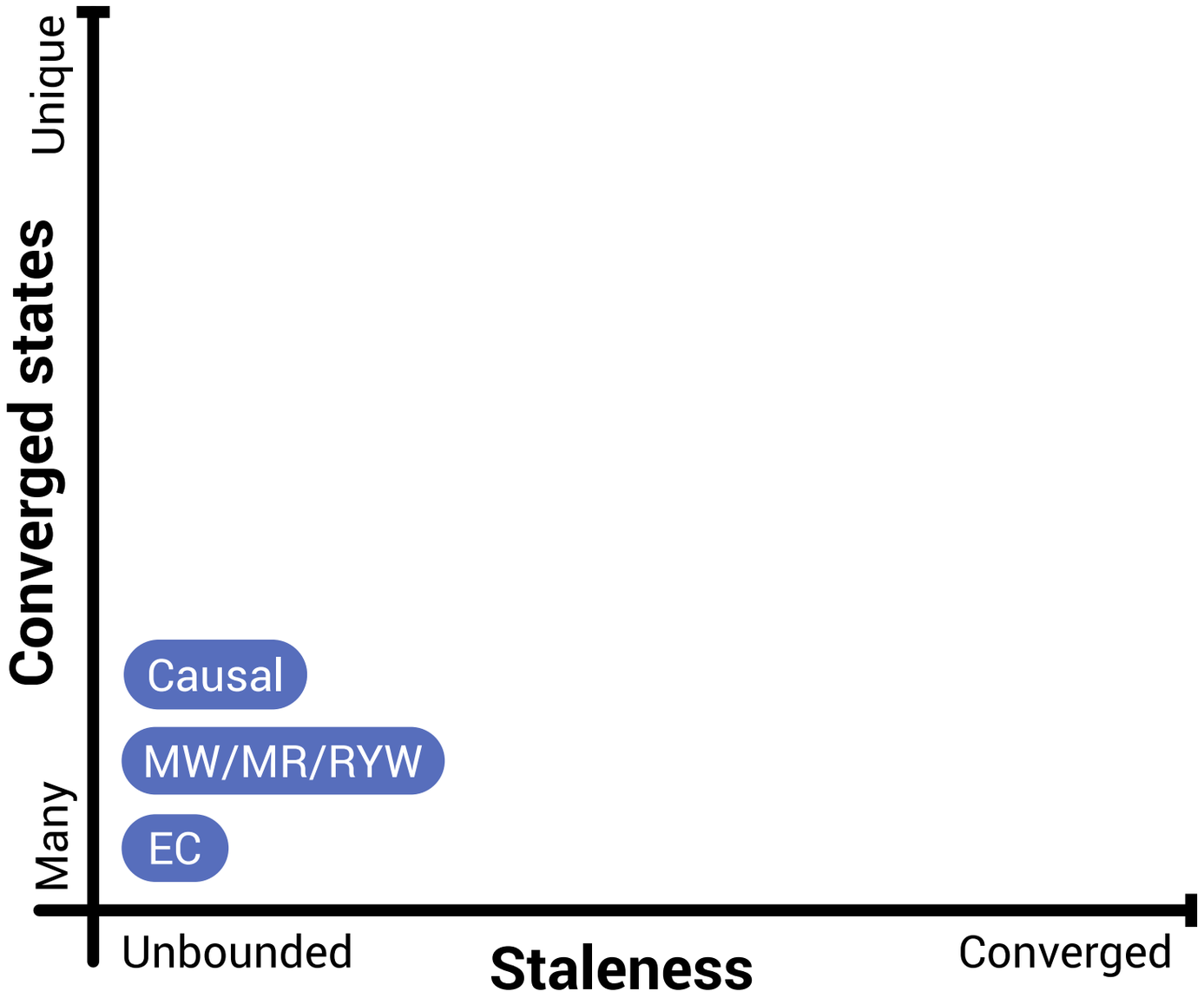
Uncertainty

Staleness

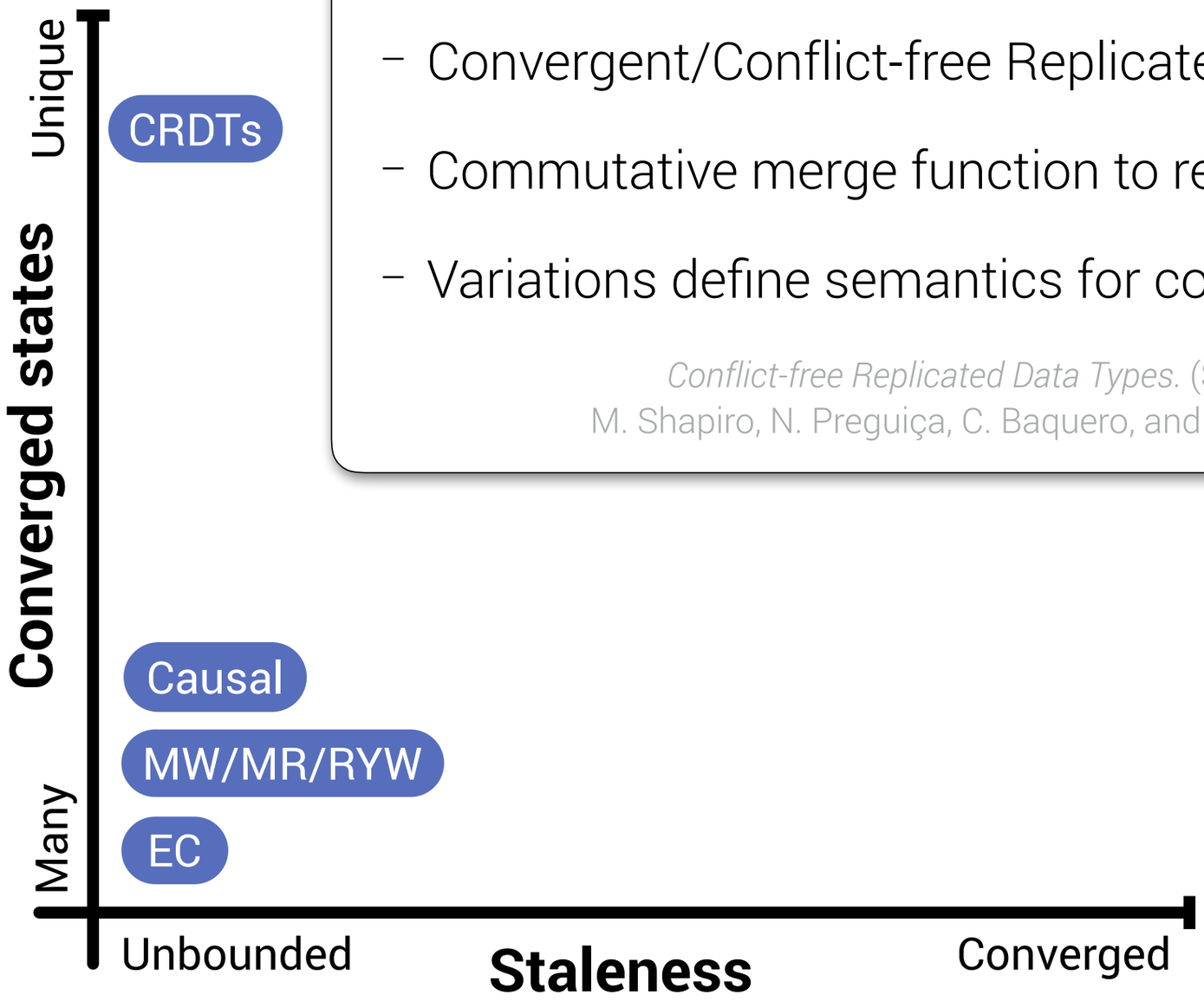


Trading off consistency

Uncertainty



Trading off consistency Uncertainty

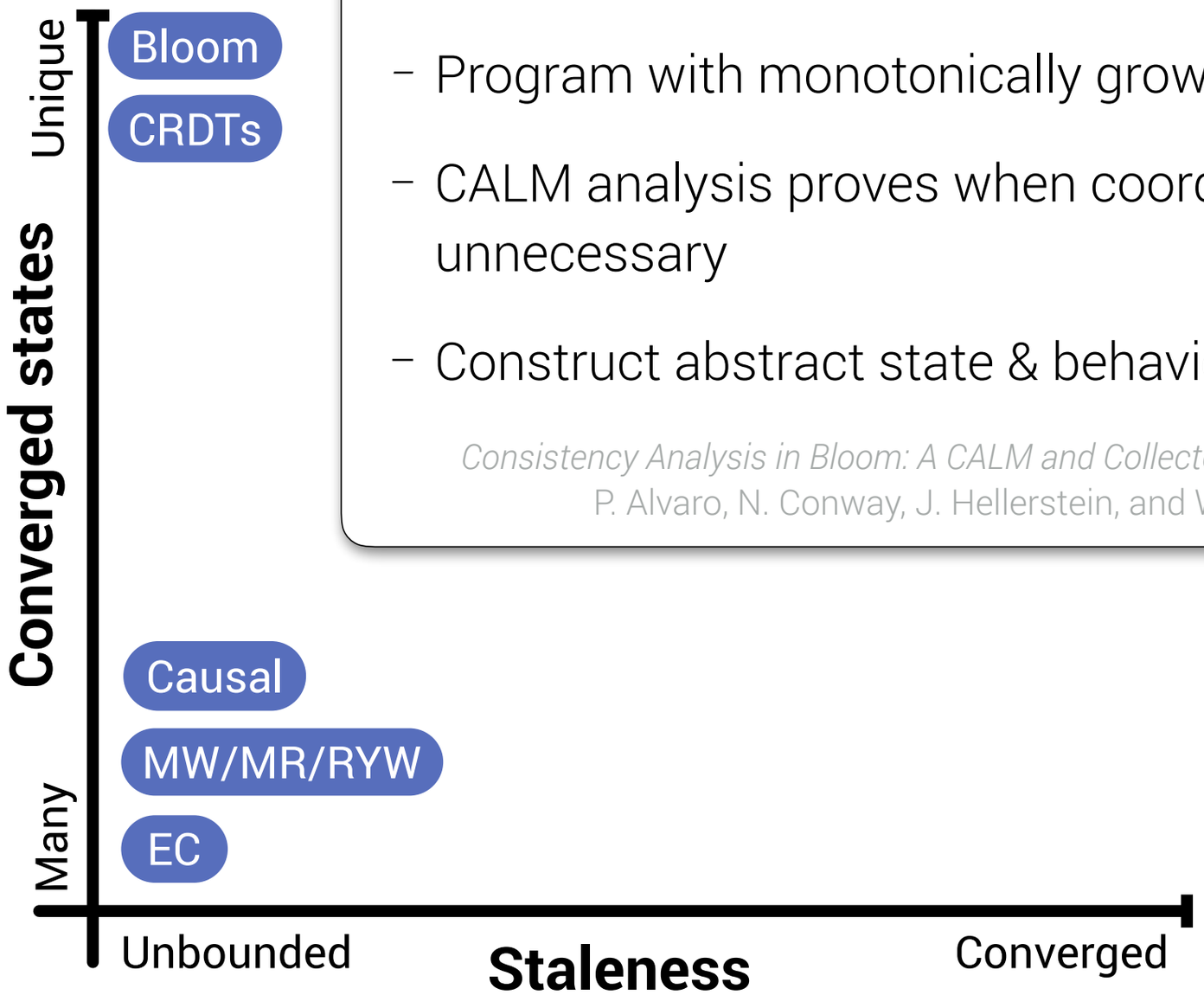


CRDTs

- Convergent/Conflict-free Replicated Data Types
- Commutative merge function to resolve conflicts
- Variations define semantics for conflicts differently

Conflict-free Replicated Data Types. (SSS'11)
M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski.

Trading off consistency Uncertainty



Bloom

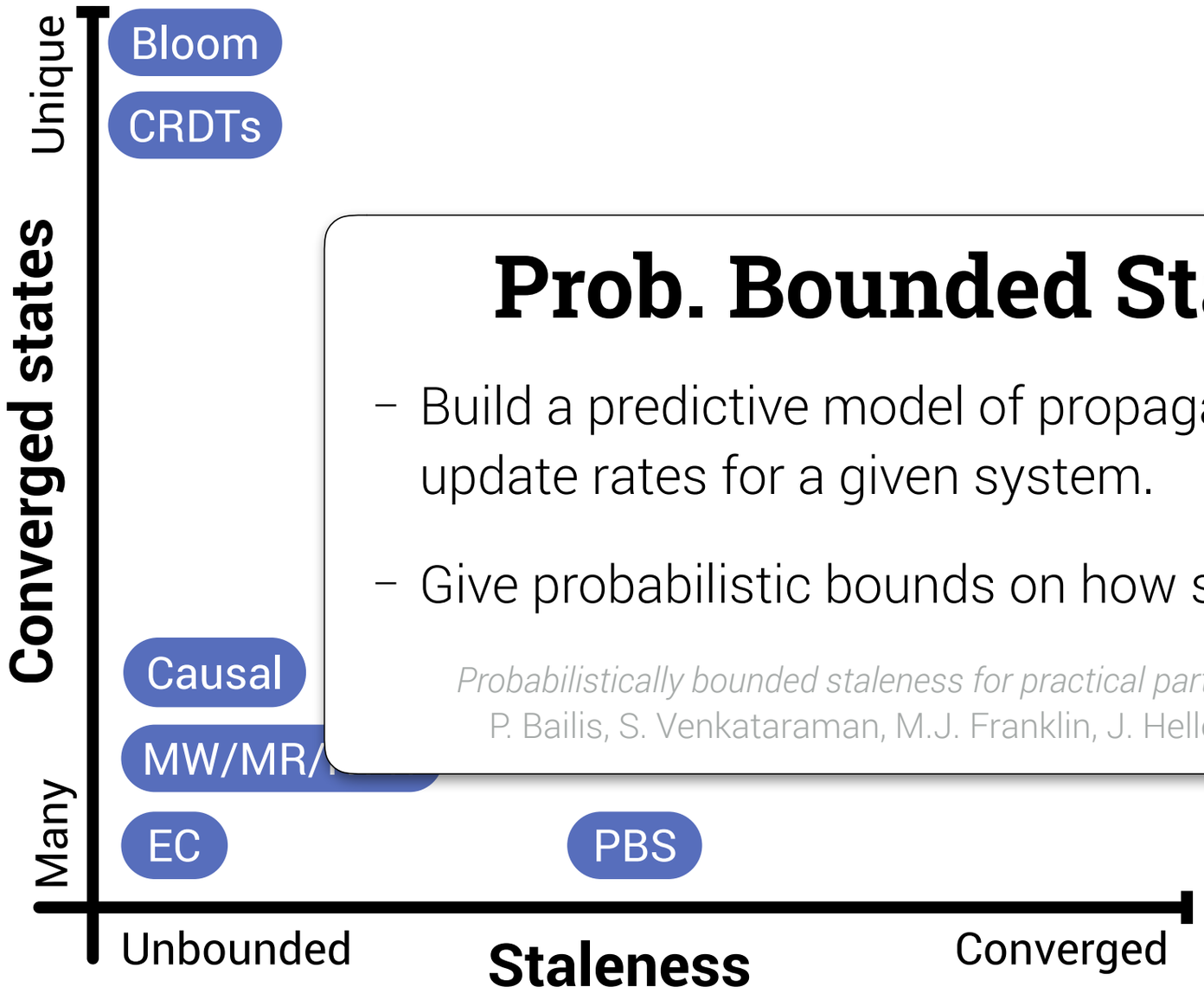
- Program with monotonically growing sets of facts
- CALM analysis proves when coordination is unnecessary
- Construct abstract state & behavior out of CRDTs

Consistency Analysis in Bloom: A CALM and Collected Approach. (CIDR'11)
P. Alvaro, N. Conway, J. Hellerstein, and W.R. Marczak.

Specify abstract state and behavior **by-construction**.

Trading off consistency

Uncertainty



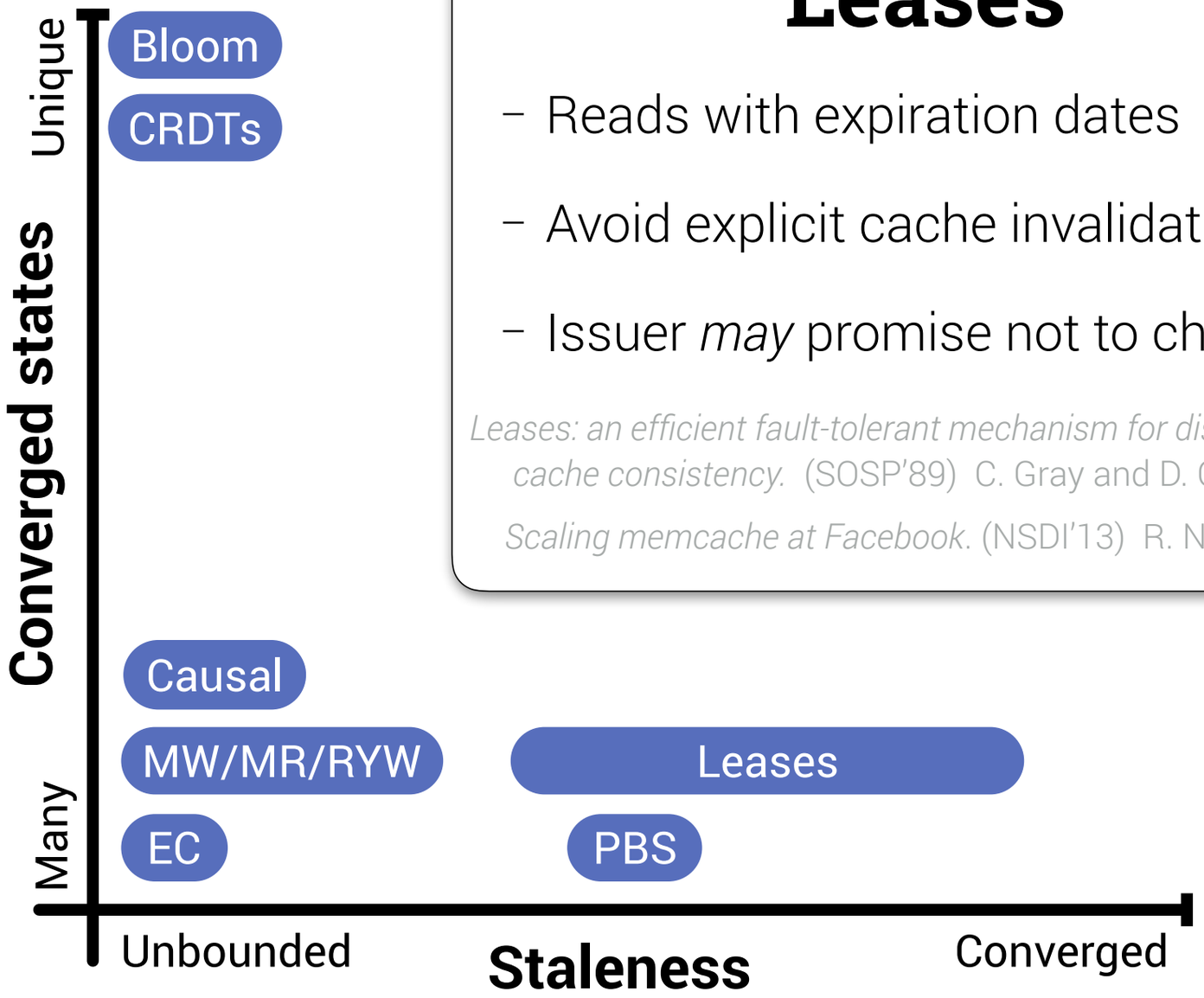
Prob. Bounded Staleness

- Build a predictive model of propagation latencies and update rates for a given system.
- Give probabilistic bounds on how stale reads will be.

Probabilistically bounded staleness for practical partial quorums. (VLDB'12)
P. Bailis, S. Venkataraman, M.J. Franklin, J. Hellerstein, and I. Stoica.

Trading off consistency

Uncertainty



Leases

- Reads with expiration dates
- Avoid explicit cache invalidation
- Issuer *may* promise not to change

Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. (SOSP'89) C. Gray and D. Cheriton.

Scaling memcache at Facebook. (NSDI'13) R. Nishtala, …

Uncertainty

Consistency-based SLAs

- System-enforced SLAs in a system called *Pileus*
- SLAs contain *target latency* and *consistency level*
- System *monitors* replicas, *predicts* which SLA it can meet.
- Specify SLA on reads, return achieved consistency level.

Consistency	Latency	Utility
Strong	100 ms	1.0
Eventual	100 ms	0.5

Movie Ticket SLA

```
# add ticket to cart
put("bob:cart", "Star Wars Ticket")

# get current cart
cart, consistency = get("bob:cart", MovieTicketSLA)
```

Consistency-based service level agreements for cloud storage. (SOSP'13)
D. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. Aguilera, and H. Abu-Libdeh.

Staleness

⚡ Explicit performance **targets**, **feedback** about results.

Introduction: Contention

✓ Sources of contention.

✓ 3 broad approaches to mitigating contention.

Background: Trading off consistency

✓ Consistency for architects.

✓ Ordering and visibility constraints.

✓ Dealing with uncertainty.

➢ **Review:** Programming tradeoffs.

Proposal: Disciplined Inconsistency

Make tradeoffs *explicit* and *safe* with types.

Reason about *values* rather than *ordering.*

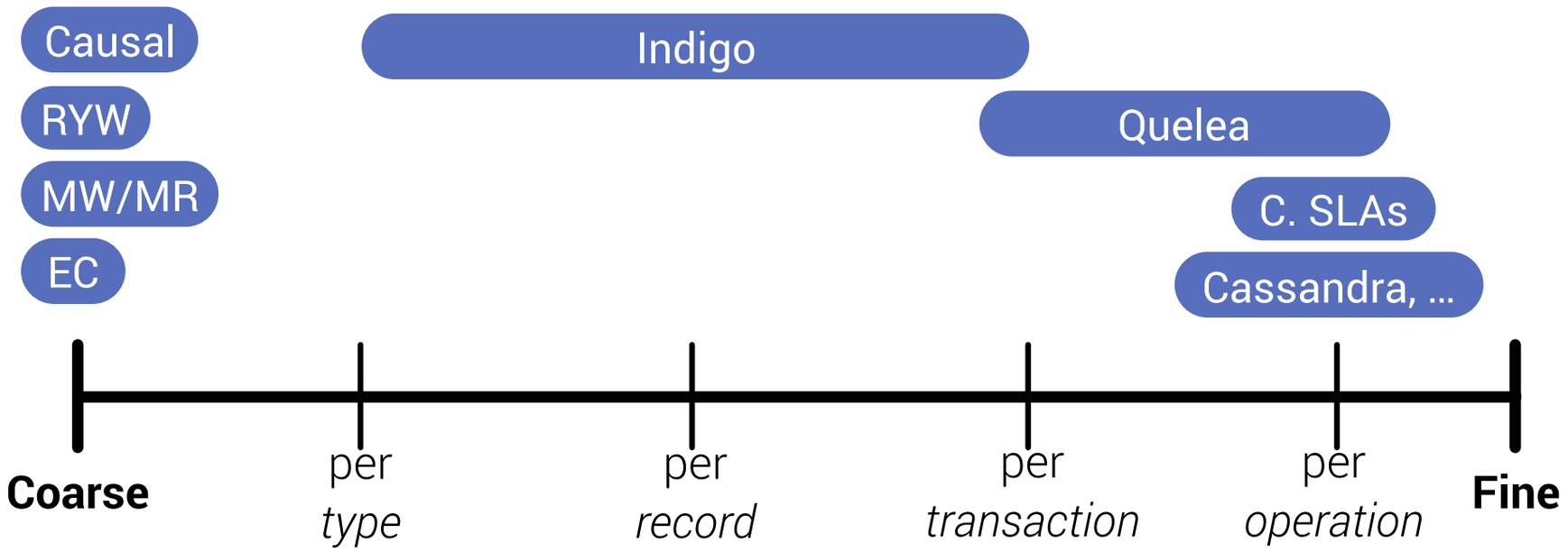
Synthesize synchronization/enforcement logic.

Trading off consistency

Programming tradeoffs

How should coordination be specified and enforced?

- Only enforce constraints where **necessary**.
- Consistency models are **too coarse-grained**.
- **Synthesize** fine-grained synchronization.

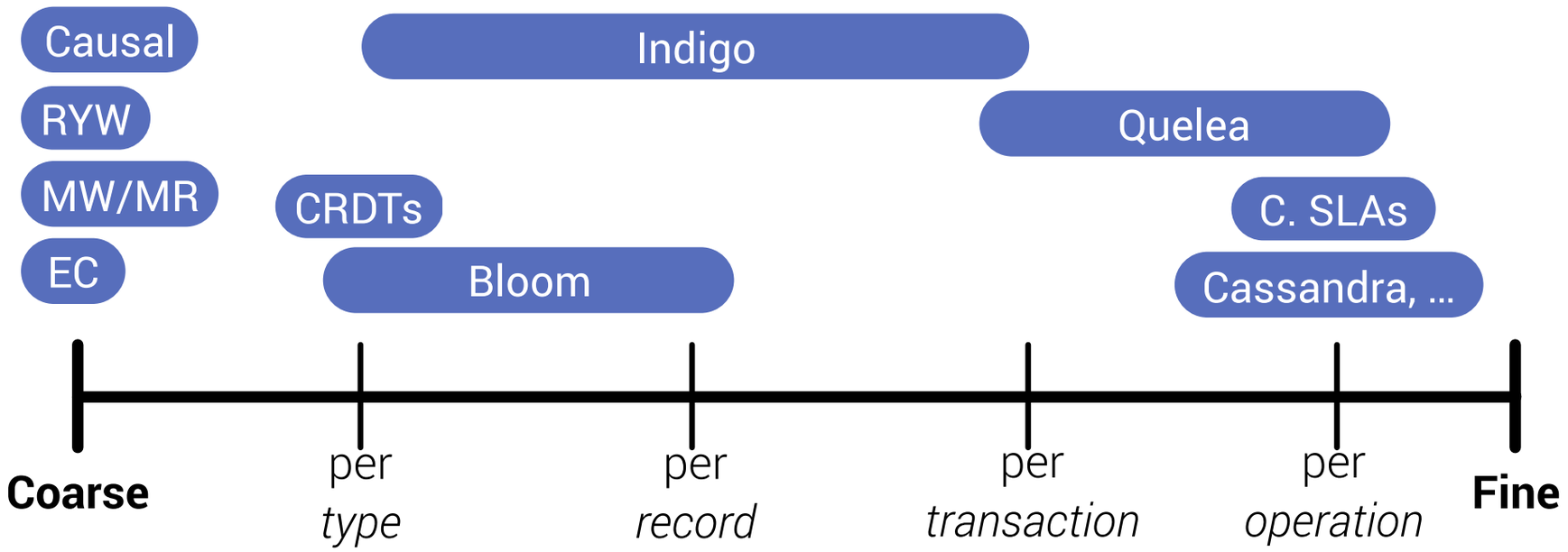


Trading off consistency

Programming tradeoffs

How can programming models help make these trade-offs?

- Specify abstract state and behavior **by-construction.**
- Explicit performance **targets**, **feedback** about results.



Introduction: Contention

- ✓ Sources of contention.
- ✓ 3 broad approaches to mitigating contention.

Background: Trading off consistency

- ✓ Consistency for architects.
- ✓ Ordering and visibility constraints.
- ✓ Dealing with uncertainty.
- ✓ **Review:** Programming tradeoffs.

➢ **Proposal:** Disciplined Inconsistency

Make tradeoffs *explicit* and *safe* with types.

Synthesize synchronization/enforcement logic.

Bounds ⇔ Uncertainty

- *Bounds* on reads determine the *uncertainty* of results

System Design

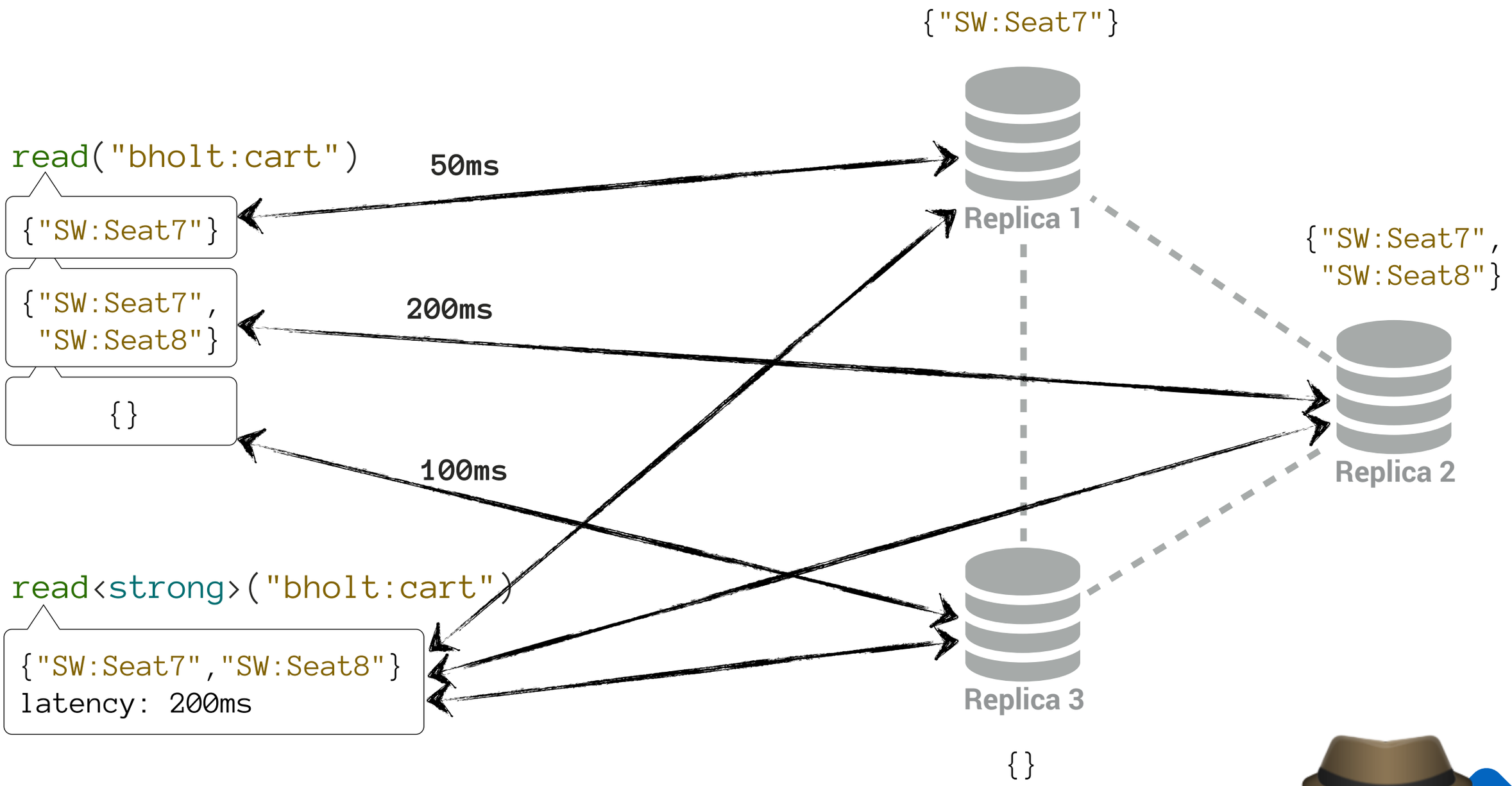
- Statically determine the type of uncertainty implied by bounds
  - IPA Types
  - Enforcement System

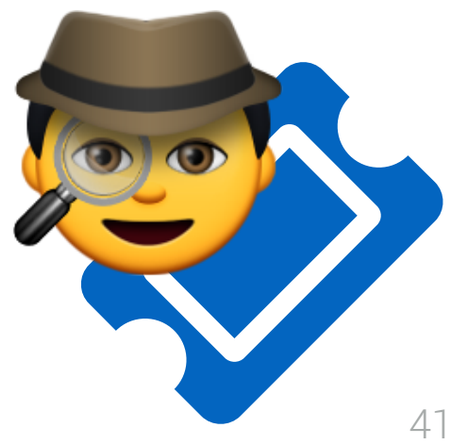
Case Studies

- TicketSleuth
- Twitter

Disciplined Inconsistency

Bounds ⇔ Uncertainty





Disciplined Inconsistency

Bounds ⇔ Uncertainty

Bounds on reads determine **uncertainty** of result.

```
read("bholt:cart")
```

{"SW:Seat7"}

{"SW:Seat7",
 "SW:Seat8"}

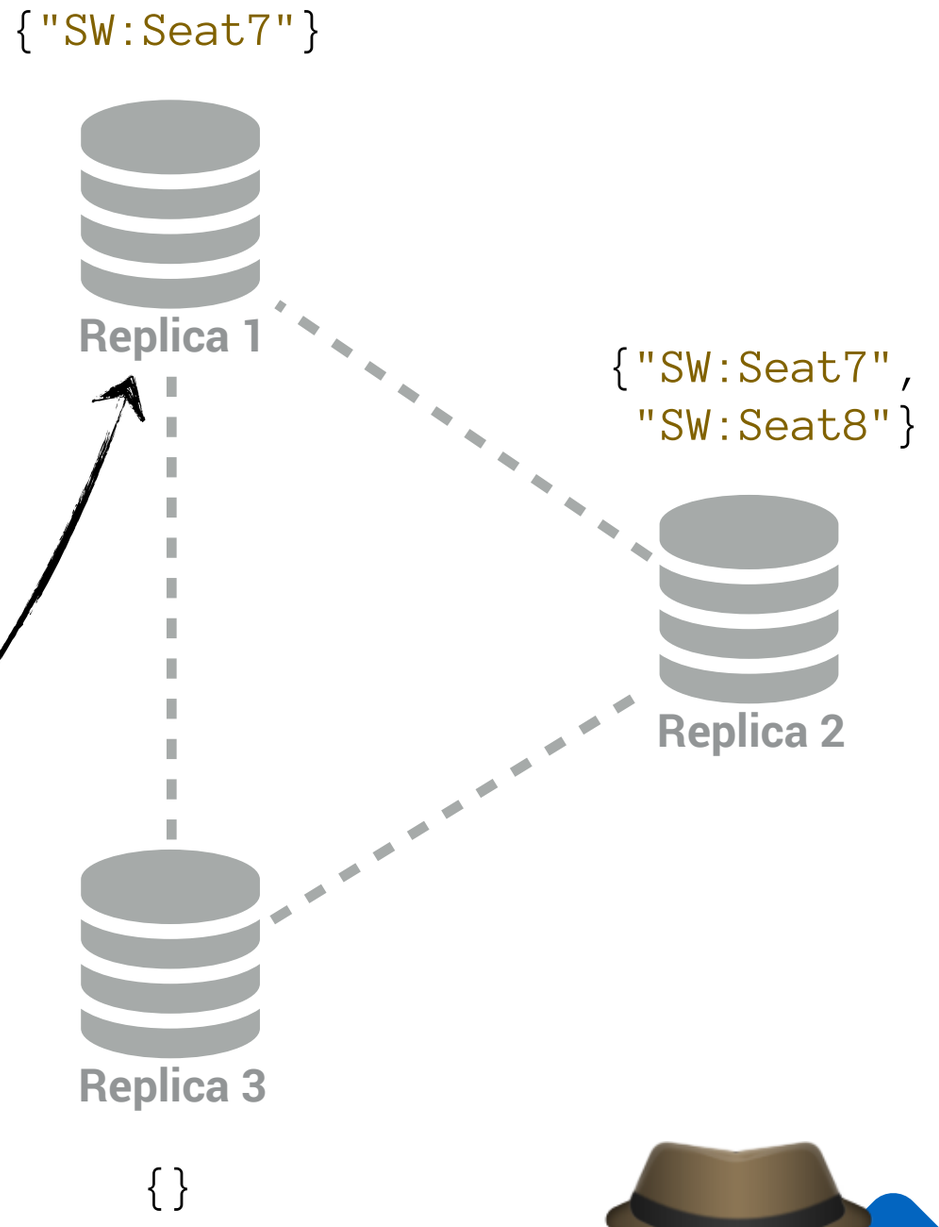
{}

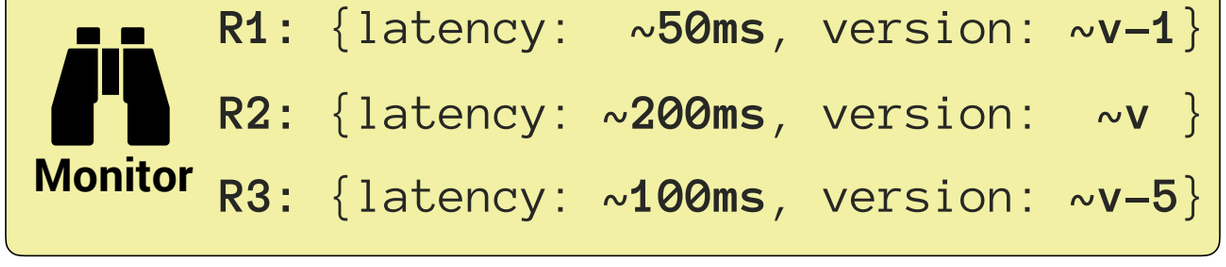
```
read<strong>("bholt:cart")
```

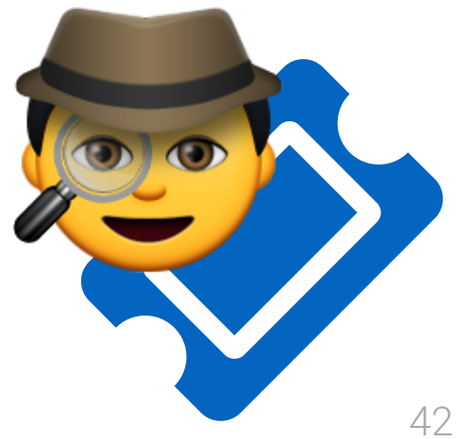
{"SW:Seat7","SW:Seat8"}
latency: 200ms

```
read<latency:100ms>("bholt:cart")
```

{"SW:Seat7"}
latency: 51ms







Disciplined Inconsistency

Bounds ⇔ Uncertainty

⚡ **Bounds** on reads determine **uncertainty** of result.

Bounds		Uncertainty
`read<strong>`	⇒	`T`
`read<ec>`	⇒	`Inconsistent<T>`
`read<available>`		

`latency < 100ms`	⇒	`Stale<T>`
		`Probabilistic<T>`
		`Interval<T>`

`value ≤ 0`
`value ± 5%`
`unique(_)`
	⇒	`Interval<T>`
		`Leased<T>`

Specified with abstract data types

IPA Types

Disciplined Inconsistency System Design

Abstract Data Types

encode **semantics** and
value & performance **bounds**

IPA Types

represent **uncertainty**

Type System

- **statically check** or **infer** that IPA types for return values match the specified bounds
- determine which techniques to use to enforce the bounds

Enforcement System

library of coordination/synchronization strategies

IPA Types

Inconsistent

`Inconsistent<T>`

`Stale<T>`

`Leased<T>`

- if you don't know anything more, at least prevent bad data flow (a la EnerJ)

```
// initial load: as fast as possible
Inconsistent<List> x = Cart.read()
display_cart(x)

// on checkout, be precise
List y = Cart.read<strong>()
display_cart(y)

checkout(x) // catch bug on compile
```

Probabilistic

`Probabilistic<T>`

Approximate

`Interval<T>`

IPA Types

Inconsistent

`Inconsistent<T>`

`Stale<T>`

`Leased<T>`

- when did the replica last synchronize?
- may want to handle older data differently

```
// initial load: as fast as possible
Stale<List> x = Cart.get()

if (x.older_than(Seconds(5))) {
    display_warning("May be out of date!")
}
display_cart(x)
```

Probabilistic

`Probabilistic<T>`

Approximate

`Interval<T>`

IPA Types

Inconsistent

`Inconsistent<T>`

`Stale<T>`

`Leased<T>`

- value associated with a *lease* promising no updates for the duration

```
// initial load: as fast as possible
Leased<List> x = Cart.get()

if (x.expired()) {
    display_warning("Out of date!")
} else {
    display_cart("Current: " + x)
}
```

Probabilistic

`Probabilistic<T>`

Approximate

`Interval<T>`

IPA Types

Inconsistent

`Inconsistent<T>`

`Stale<T>`

`Leased<T>`

Probabilistic

`Probabilistic<T>`

Approximate

`Interval<T>`

- probabilistic model (like PBS) that predicts staleness & update rate
- either a *single value* with a *confidence* (staleness domain)
- or a distribution in the *value domain*

```
// bounded latency (e.g. < 200ms)
Prob<Int> x = Tickets("StarWars").remaining()

if ( (x < 10).confidence(0.5) ) {
    display_warning("Purchase soon!")
}
display_tickets(x)
```

Uncertain<T>: A First Order Type for Uncertain Data (ASPLOS'14)
J. Bornholt, T. Mytkowicz, K.S. McKinley

IPA Types

Inconsistent

`Inconsistent<T>`

`Stale<T>`

`Leased<T>`

Probabilistic

`Probabilistic<T>`

Approximate

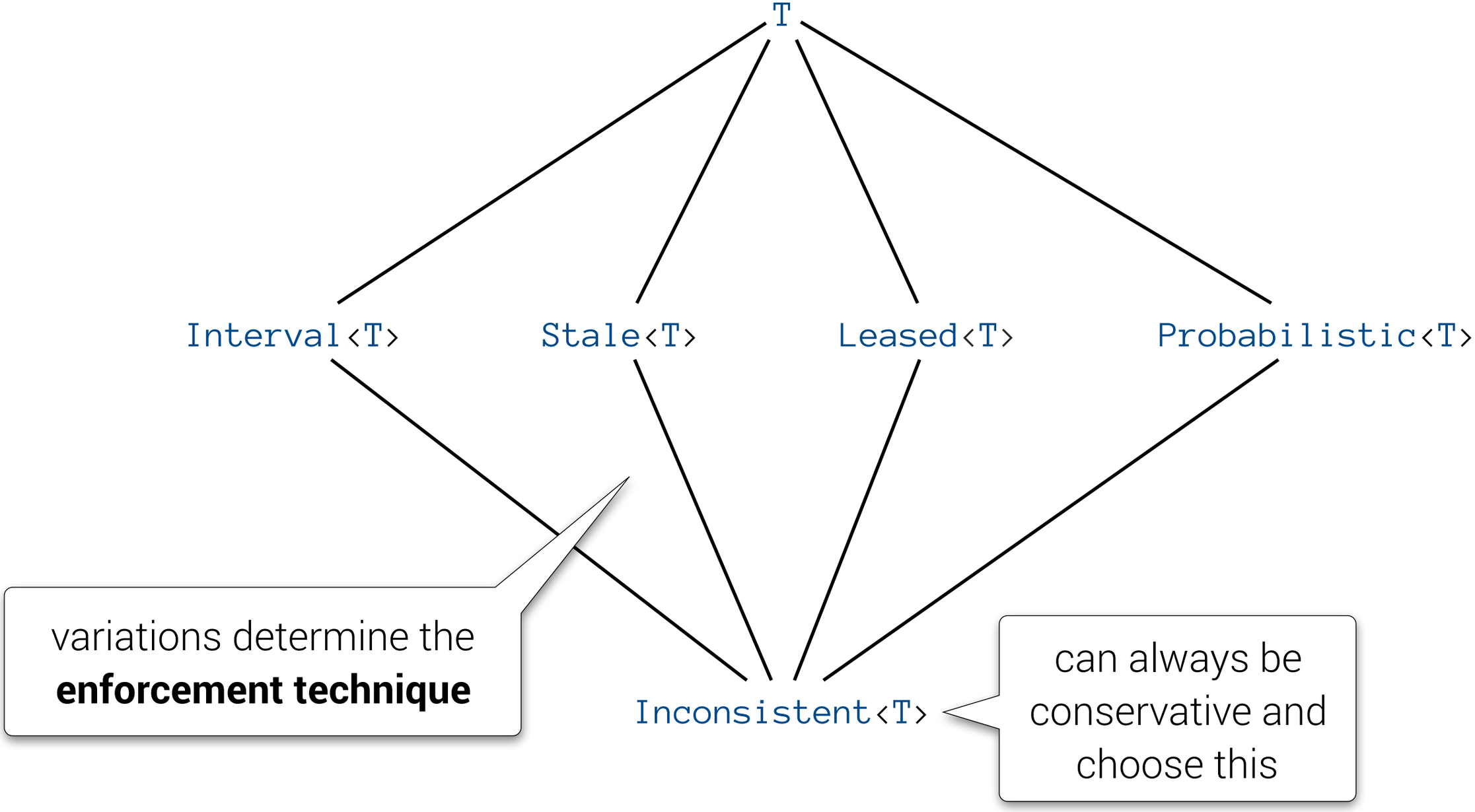
`Interval<T>`

- *range* of possible values
- any type with a *partial order over values*

```
// bounded latency (e.g. < 200ms)
Interval<Int> x = Tickets("StarWars").remaining()

if (x.min > 1M) {
    // e.g. '1.4M'
    display_tickets((x/100k) + "M")
} else if (x.min > 1000) {
    // e.g. '2.7k'
    display_tickets((x.min/100) + "k")
} else {
    // e.g. '5-20'
    display_tickets(x.min + "-" + x.max)
}
```

IPA Type Hierarchy

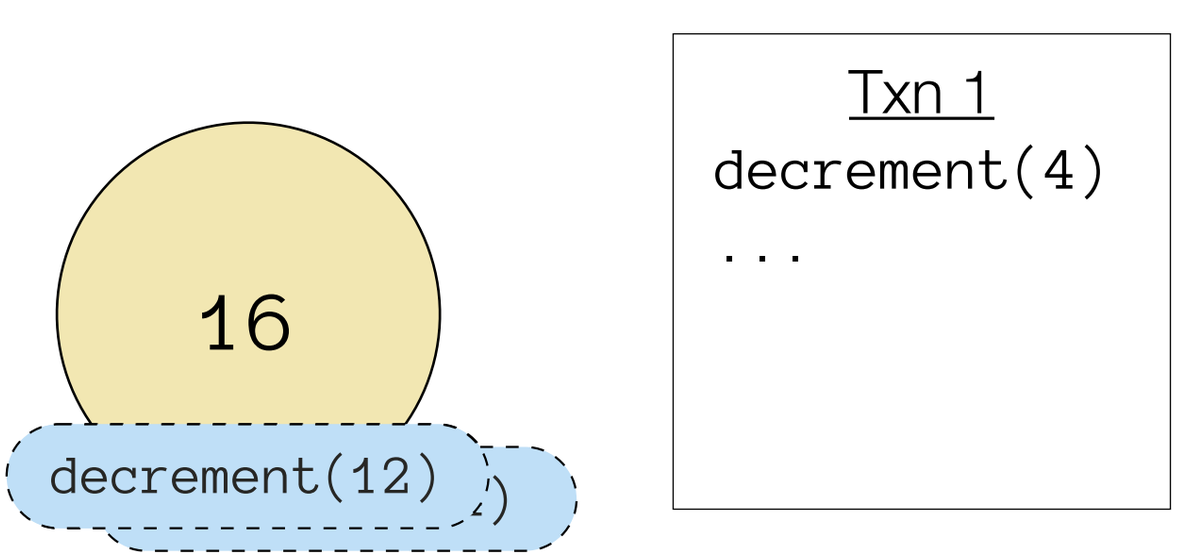


Enforcement System

*How can you enforce **hard value bounds** on weak replication?*

Escrow

- Fragment value, hold parts *in escrow* until ready to commit.
- Take from *pool* of available updates / fragments.



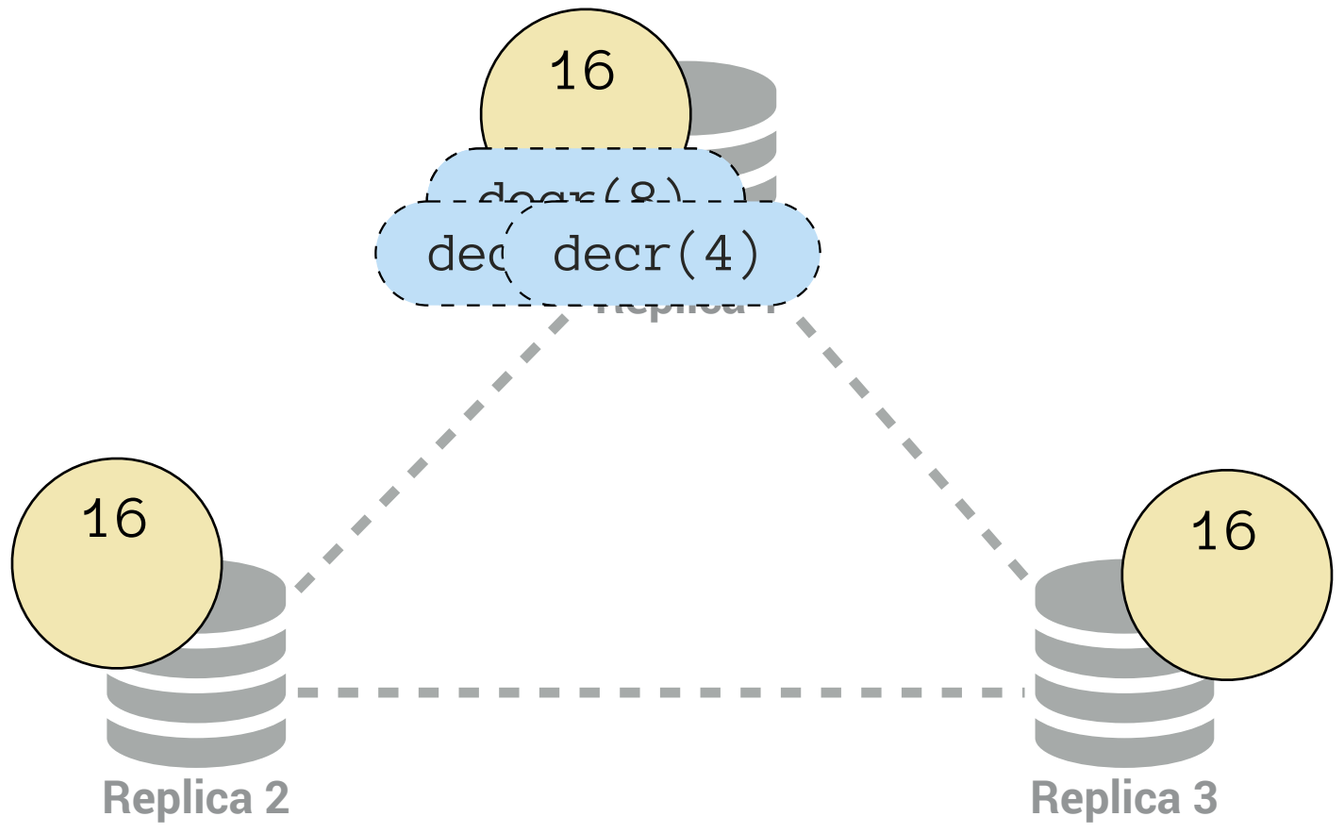
The Escrow Transactional Method.
(Transactions on Database Systems, 1986)
Patrick E. O'Neil

Enforcement System

*How can you enforce **hard value bounds** on weak replication?*

Reservations

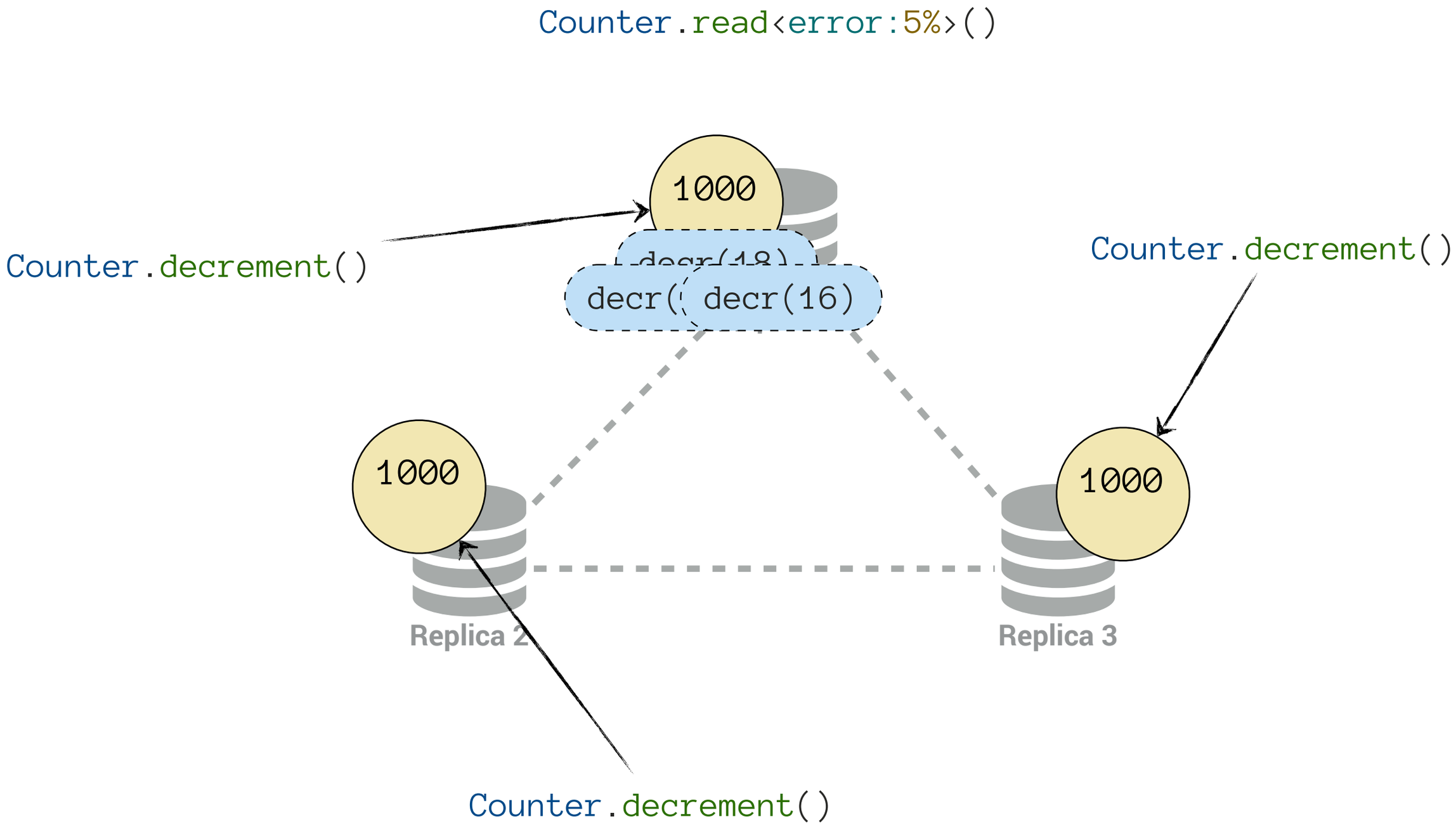
- Pre-allocate permission to do updates and distribute to replicas.
- Moves coordination off the critical path.



Reservations for conflict avoidance in a mobile database system.
N. Preguiça, J.L. Martins, M. Cunha, and H. Domingos.

Disciplined Inconsistency Enforcement System

*How can you enforce **hard value bounds** on weak replication?*



Escrow / Reservations

- provide `Interval<T>`, `Leased<T>`

Monitoring & prediction system

- like those in PBS/Consistency-based SLAs
- provide `Probabilistic<T>`, `Stale<T>`

More…

Bounds ⇔ Uncertainty

- *Bounds* on reads determine the *uncertainty* of results

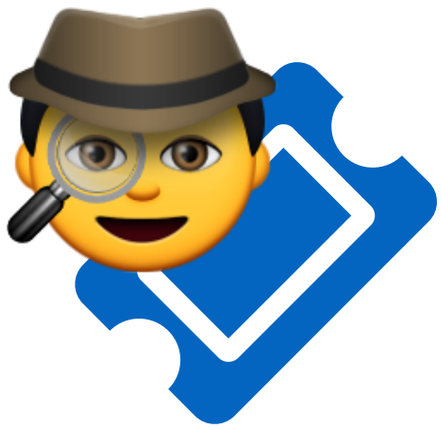
System Design

- Statically determine the type of uncertainty implied by bounds
  - IPA Types
  - Enforcement System

Case Studies

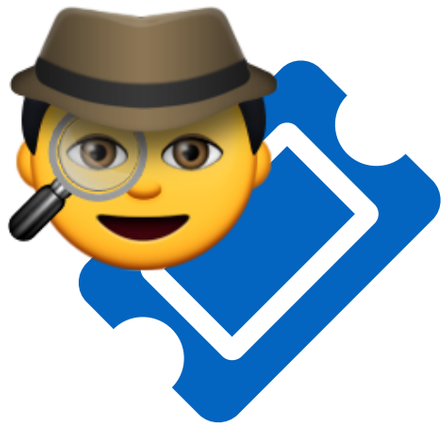
- TicketSleuth
- Twitter

Case Studies



TicketSleuth

Disciplined Inconsistency Case Studies

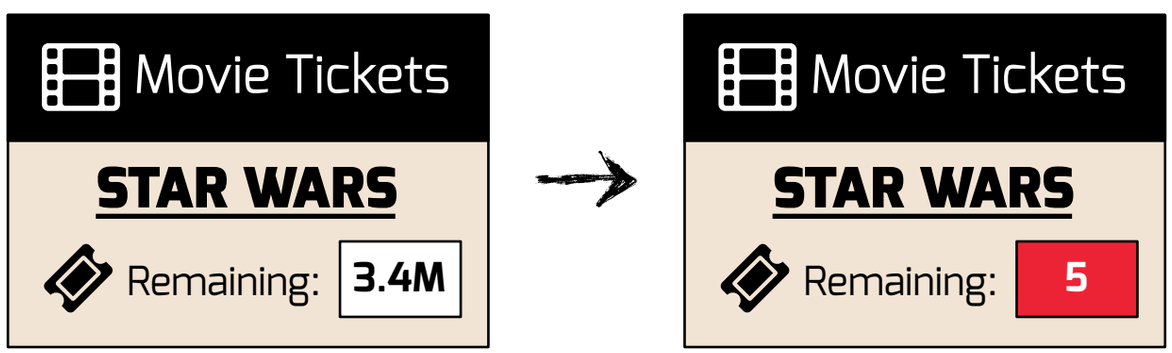


hard value constraint

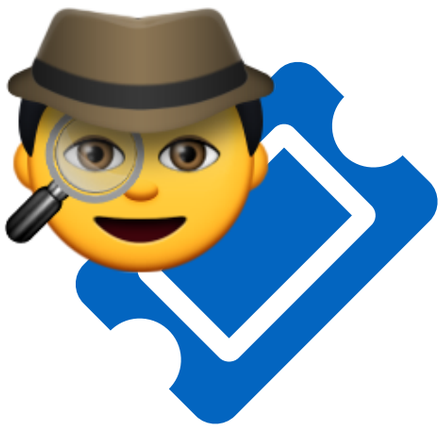
IPA type

1. *Never* over-sell tickets.
2. Show *estimate* of remaining tickets.
3. Initial page must load in *100ms* (99th percentile)

performance constraint



Disciplined Inconsistency Case Studies



hard value constraint

IPA type

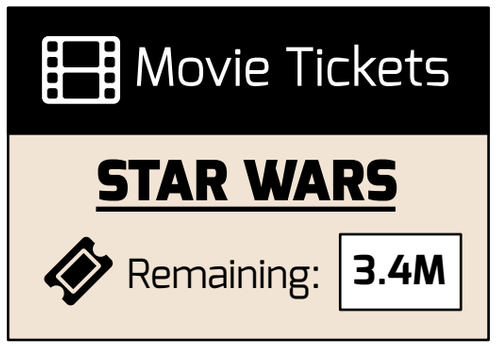
1. *Never* over-sell tickets.
2. Show *estimate* of remaining tickets.
3. Initial page must load in *100ms* (99th percentile)

performance constraint

```
// Pool supporting unique `take`s
// bounded by initial size, and
// reading an approximate size.
class Pool< type T, Latency L > {

  fn take() → T

  // return the range of possible
  // current sizes
  fn size<latency: L>() → Interval<int>
};
```




```
// derive app-specific type from generic IPA type,
// select constraints: 100 ms latency
type MovieTickets = Pool<TicketID, Latency(100ms)>;
```

Disciplined Inconsistency **Case Study:** Twitter

```
Timeline <: List<TweetID>
    fn slice<latency:100ms>(int start, int end) → Stale<List<TweetID>>

Retweets <: Set<UserID>
    fn size<tolerance:1%>() → Interval<int>
```

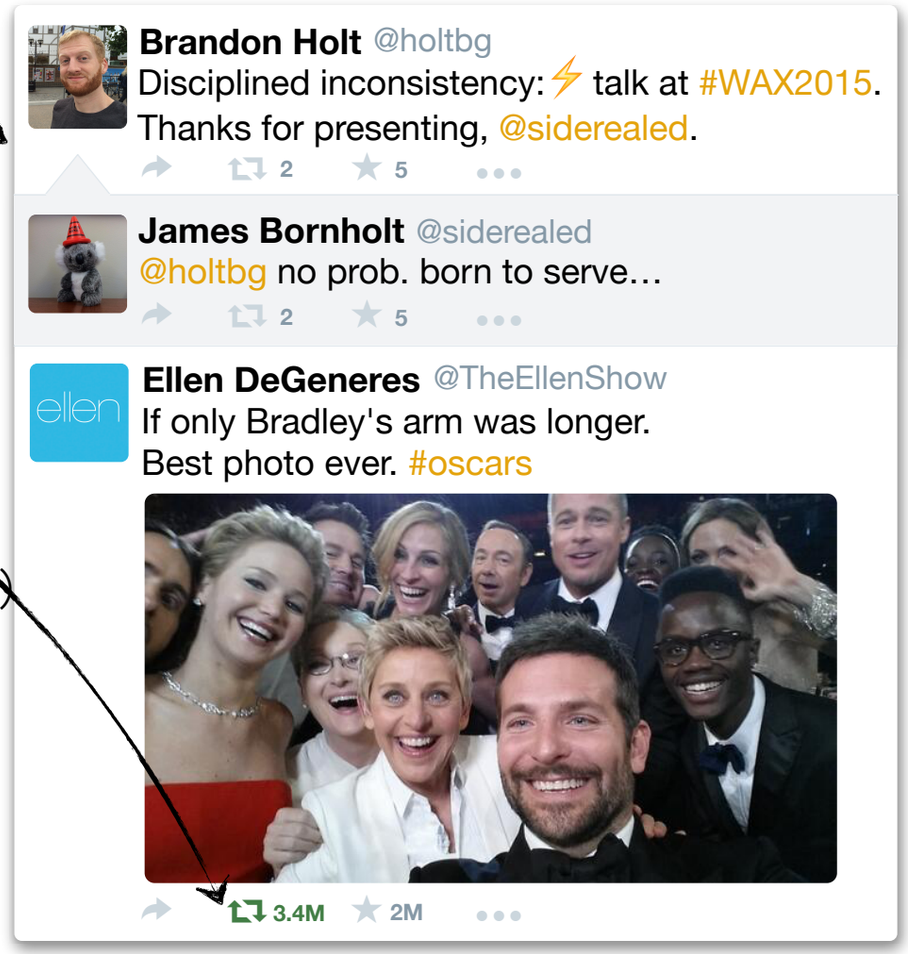
```
def timeline(user):
    # Timeline.range: 100 ms latency bound
    tweets: Stale<List<TweetID>> = Timeline(user).slice(0, 10)

    # check if timeline may be missing tweets older than 5s
    if posts.older_than(Seconds(5)):
        display_warning("May be out of date.")

    for t in tweets:
        tweet: Map = Tweet(t).get()

        # Retweeters.size: 1% tolerance
        retweets: Interval<Int> = Retweets(t).size()

        display_tweet(tweet, retweets)
```



Introduction: Contention

- Sources of contention.
- 3 broad approaches to mitigating contention.

Background: Trading off consistency

- Consistency for architects.
- Ordering and visibility constraints.
- Dealing with uncertainty.
- Programming tradeoffs.

Proposal: Disciplined Inconsistency

- Make tradeoffs *explicit* and *safe* with types.
- Reason about *values* rather than *ordering.*
- Synthesize synchronization/enforcement logic.

Mitigating Contention in Distributed Systems

Generals Exam

Brandon Holt

Committee

Luis Ceze, Mark Oskin, Dan Ports, Jevin West