Disciplined Inconsistency

with *Consistency Types*

Brandon Holt, James Bornholt, Irene Zhang,
Dan Ports, Mark Oskin, Luis Ceze

UNIVERSITY of WASHINGTON

UCSC – CMPS290S – 2018

Building **scalable** & **reliable** applications is **difficult**.

Constantly changing network conditions, traffic spikes…

NEWSFEED VIRAL

The Dress That Broke the Internet, and the Woman Who Started It All

Rishi Iyengar @Iyengarish | Feb. 27, 2015

Caitlin McNeill had no idea she would start something that would be tweeted by Taylor Swift

On Thursday, Scottish musician Caitlin McNeill posted a picture of a dress on micro-blogging website Tumblr with the caption "guys please help me — is this dress white and gold, or blue and black?

Building **scalable** & **reliable** applications is **difficult**.

Constantly changing network conditions, traffic spikes…

NEWSFEED VIRAL

The Dress That Broke the Internet, and the Woman Who Started It All

Rishi Iyengar @Iyengarish | Feb. 27, 2015

Caitlin McNeill had no idea she would start something that would be tweeted by Taylor Swift

On Thursday, Scottish musician Caitlin McNeill posted a picture of a dress on micro-blogging website Tumblr with the caption "guys please help me — is this dress white and gold, or blue and black?

Building **scalable** & **reliable** applications is **difficult**.

MOVIES | STAR WARS GALAXY

Star Wars: The Force Awakens presale crashes ticketing websites

BY OLIVER GETTELL · @OGETTELL

Share | 14064 Tweet | 0 | 15 | 0

(Lucasfilm)

Top Stories

Tom Hanks braces for impact in first 'Sully' trailer

'Lost in Space' reboot ordered by Netflix

'Star Trek' fan film guidelines defended by CBS official

'Ghostbusters' cast shows off new gadgets

Emma Watson's Tina Turner ringtone interrupts interview

What to Watch Wednesday: 'Barely Famous' is fully funny

'Night Of': Riz Ahmed held 'onto trauma for 6 months' during filming

Building **scalable** & **reliable** applications is **difficult**.

Programming languages can **help**:

- Prevent common mistakes *statically*
- Communicate application requirements

TicketSleuth

MOVIES | STAR WARS GALAXY

Star Wars: The Force Awakens presale crashes tic websites

BY OLIVER GETTELL · @OGETTELL

Share | 14064 Tweet | 0 | 15 | 0

(Lucasfilm)

Top Stories

Tom Hanks braces for impact in first 'Sully' trailer

'Lost in Space' reboot ordered by Netflix

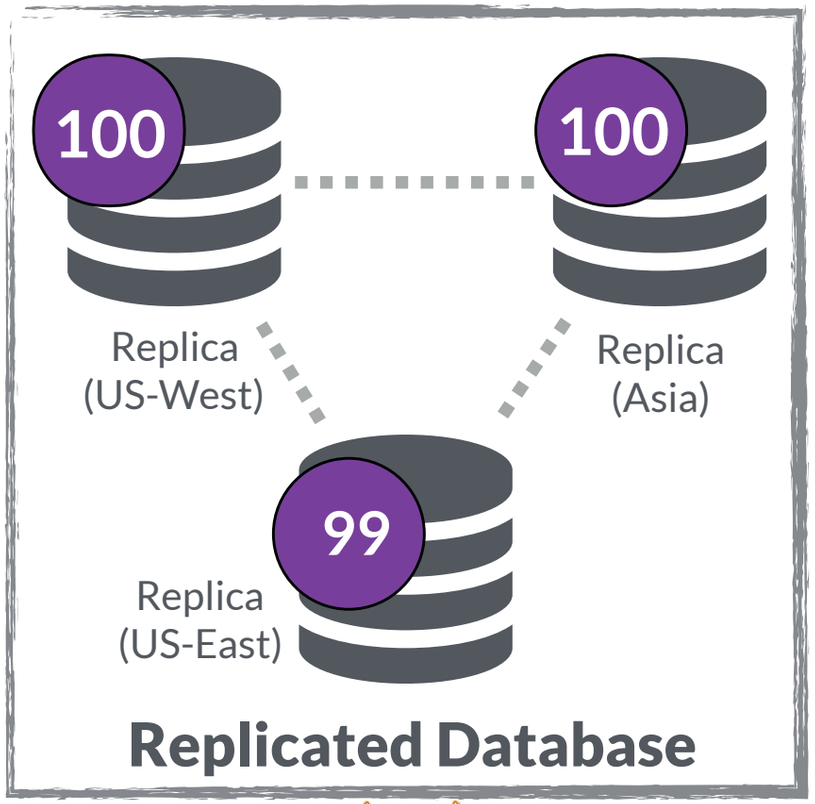
'Star Trek' fan film guidelines defended by CBS official

'Ghostbusters' cast shows off new gadgets

Emma Watson's Tina Turner ringtone interrupts interview

What to Watch Wednesday: 'Barely Famous' is fully funny

'Night Of': Riz Ahmed held 'onto trauma for 6 months' during filming

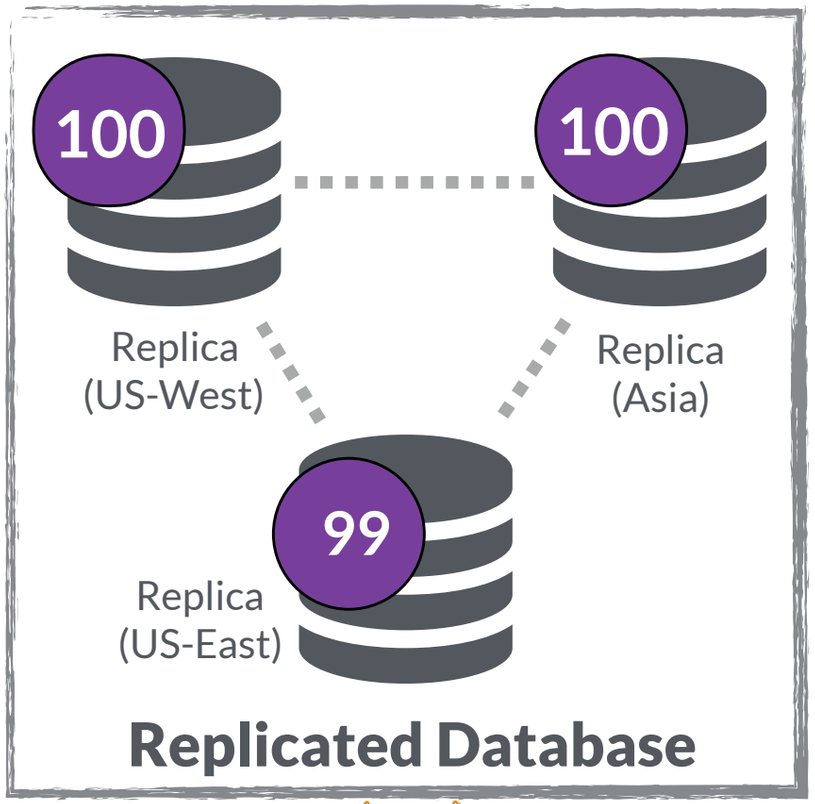


Strong consistency

Performance constraint:
< 50 ms latency

Hard constraint:
do not over-sell tickets

Error tolerance:
count does not need to be exact…



Performance constraint:
< 50 ms latency

Hard constraint:
do not over-sell tickets

Weak?

Strong consistency

Error tolerance:
count does not need to be exact…

Trading off consistency

Strong Consistency

Linearizable

Serializable

Causal

Read-your-writes

Local quorum

Eventual

Weak Consistency

easier to use,
slower

faster,
more **error-prone**

Inconsistent
Performance-bound
Approximate

} **Programming Model**

Trading off consistency

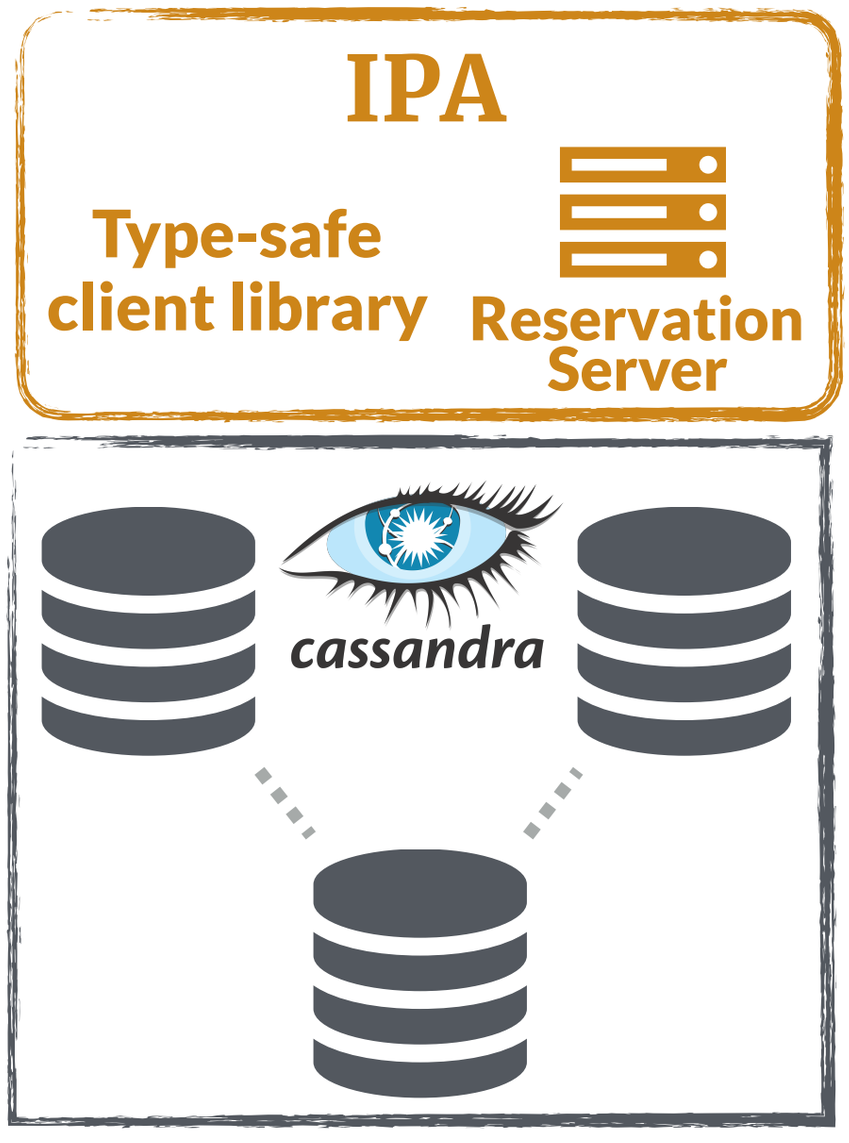
Consistency types

- Statically prevent consistency bugs

Performance & accuracy constraints

- Dynamically adjust consistency

Case study



Inconsistent
Performance-bound
Approximate

} **Programming Model**

Trading off consistency

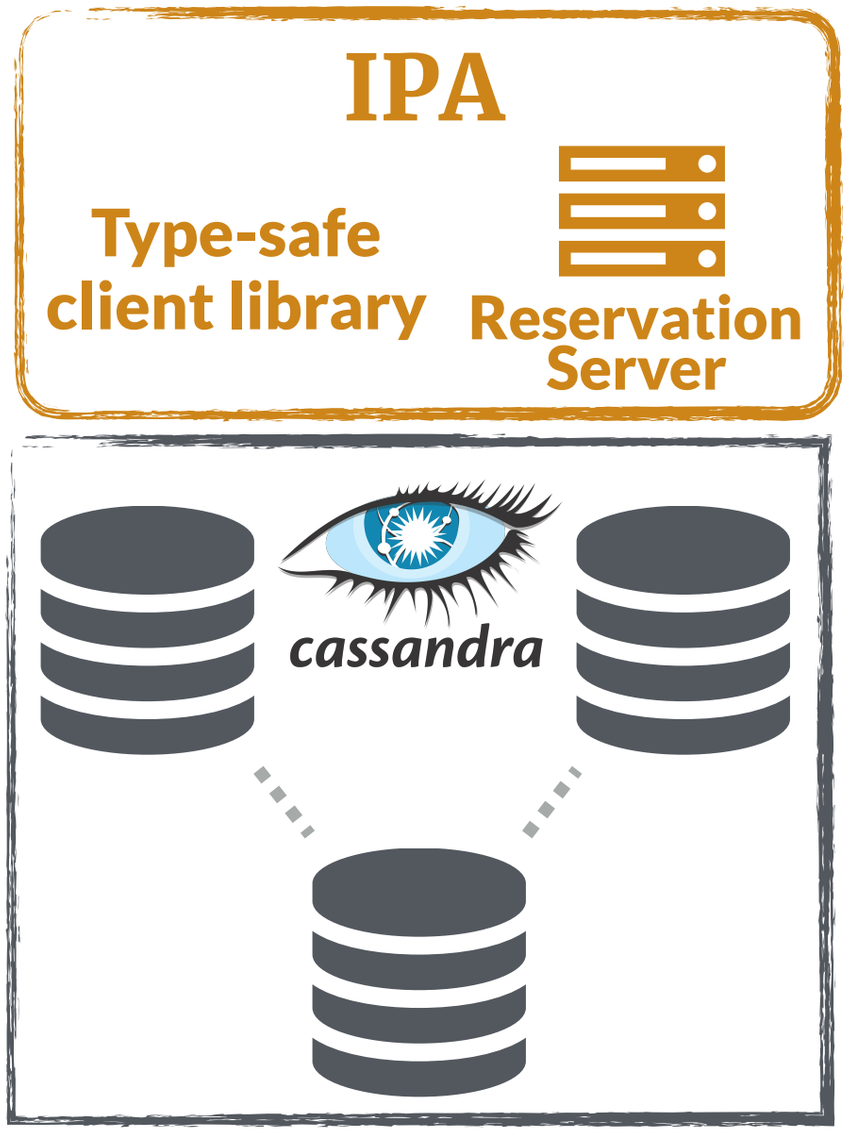
Consistency types

- Statically prevent consistency bugs

Performance & accuracy constraints

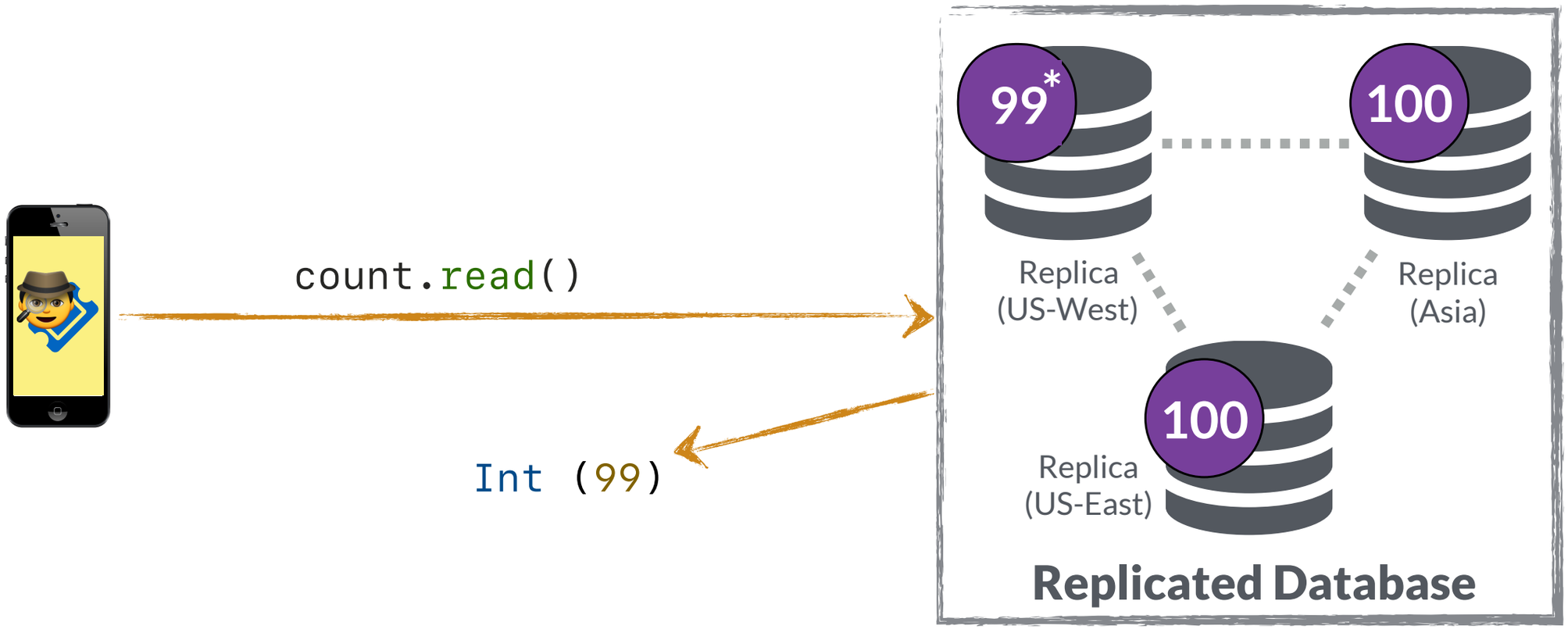
- Dynamically adjust consistency

Case study



Consistency Types

Every value returned from the datastore has a *consistency type*.

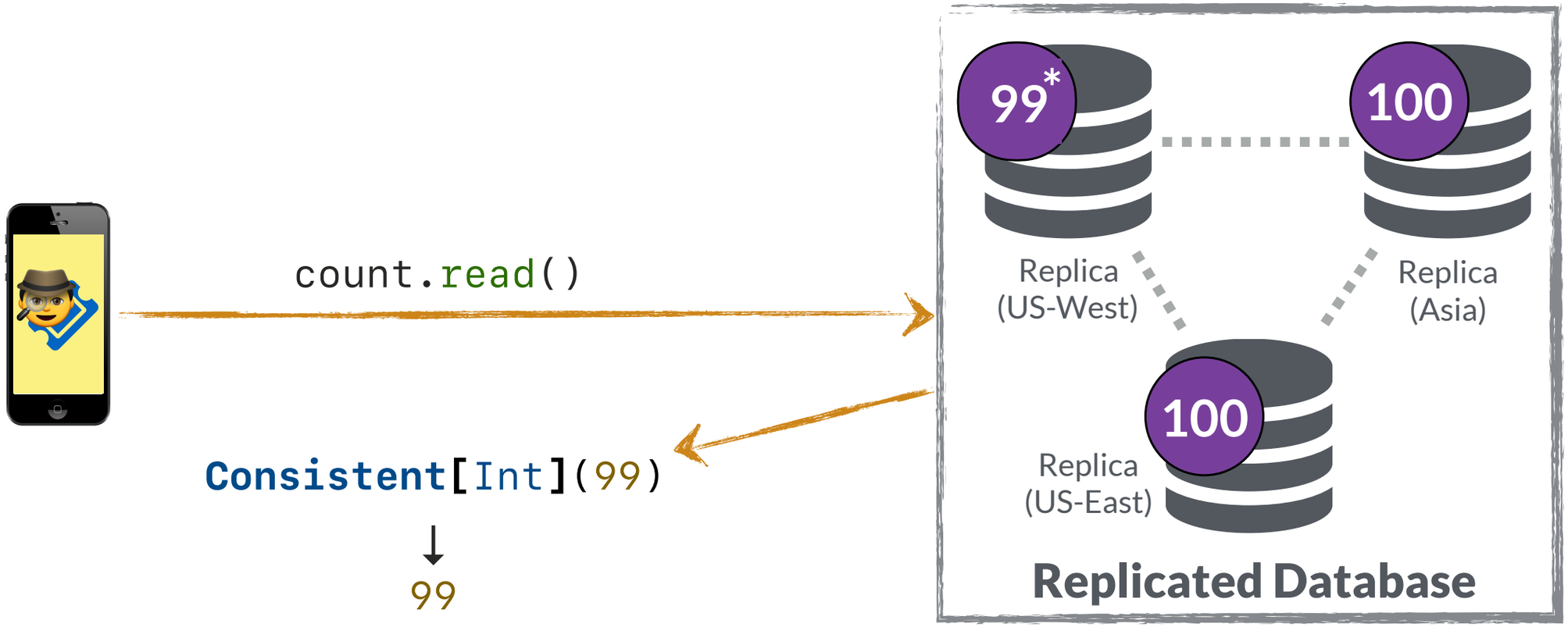


Consistency Types

Every value returned from the datastore has a *consistency type*.

<u>Strong</u> reads return `Consistent[T]`

- Can be implicitly converted to `T` (treated as a normal value)



Consistency Types

Every value returned from the datastore has a *consistency type*.

Strong reads return `Consistent[T]`

- Can be implicitly converted to `T` (treated as a normal value)

Weak reads return `Inconsistent[T]`

- Explicitly *endorse* to get the value:

```
endorse(Inconsistent(99)) → 99
```

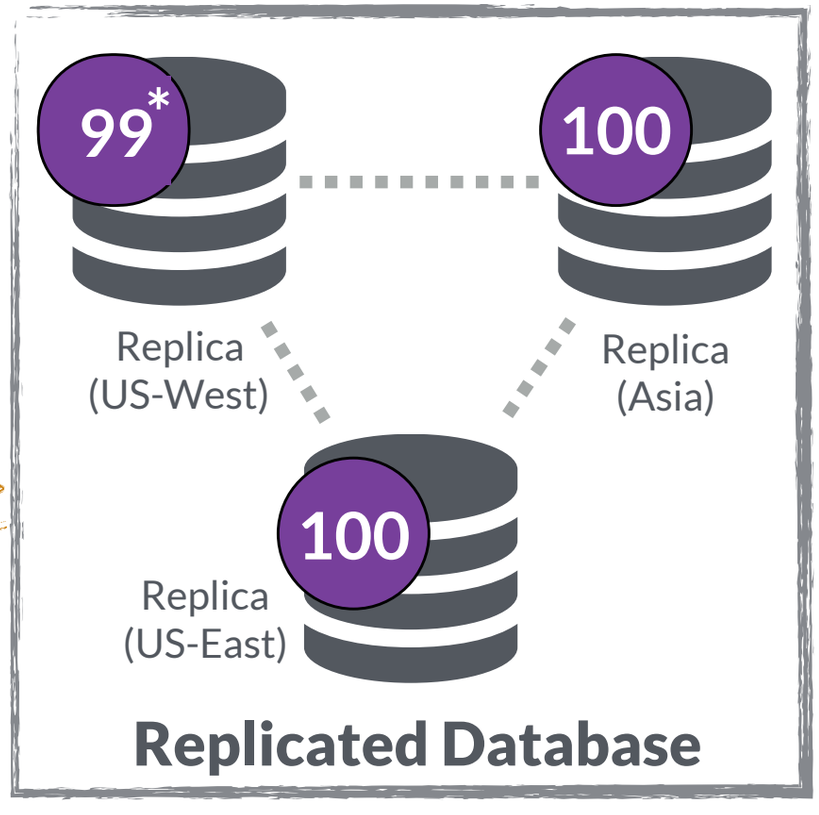
or with a case match:

```
case Inconsistent(x) ⇒ x:Int
```

`count.read_weak()`

`Inconsistent[Int](99)`

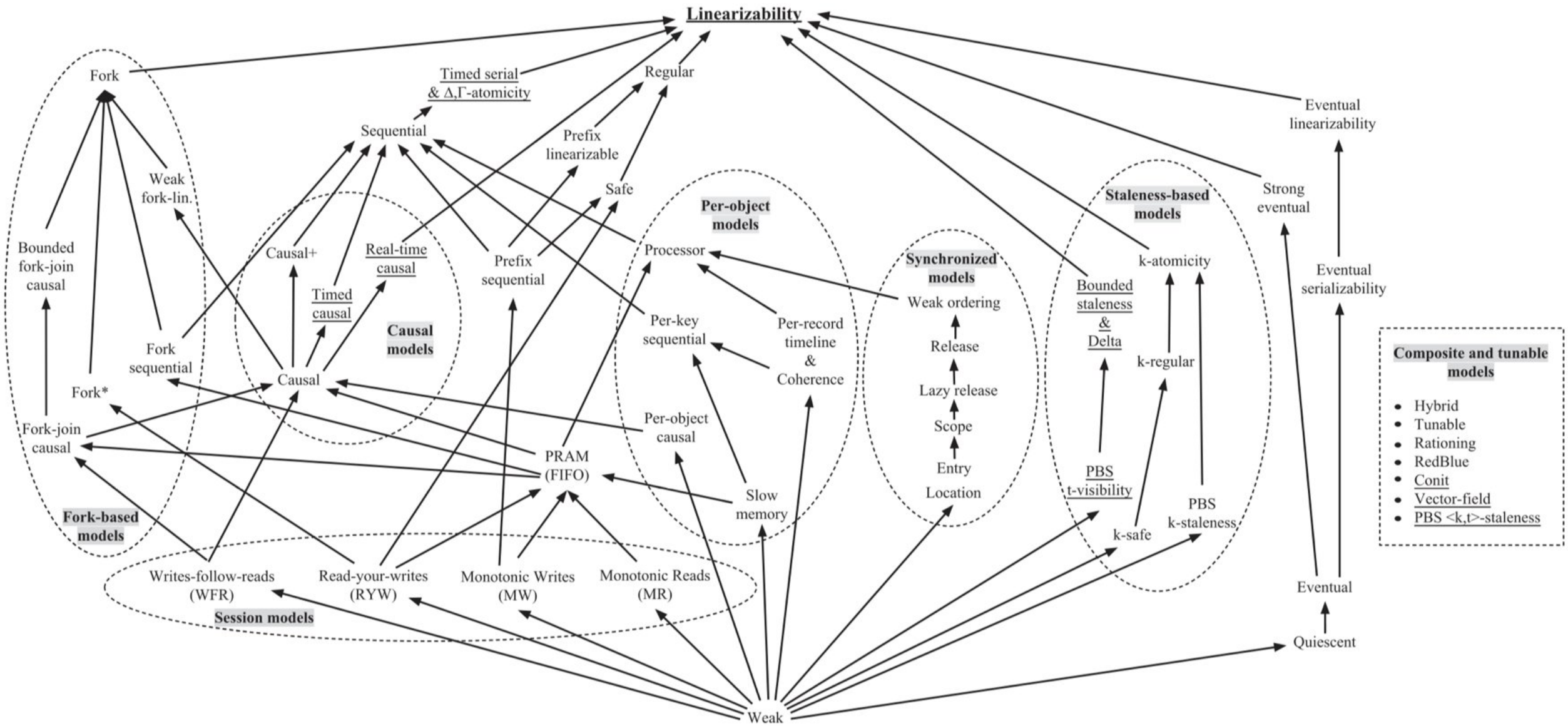
or… **`Inconsistent[Int](100)`**



Consistency Types

***Partially ordered*, forming a *lattice*:**

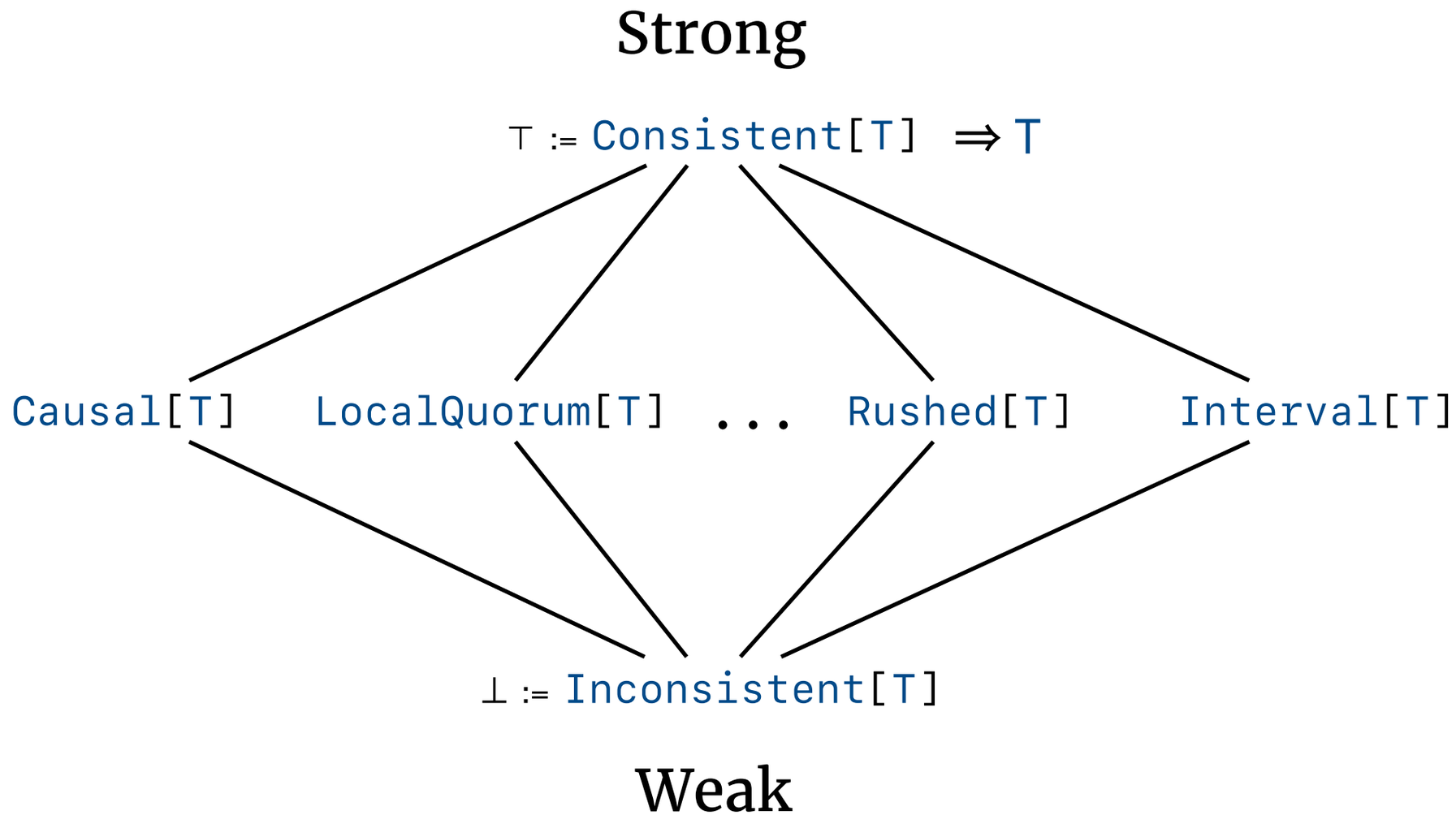
$$\top := \texttt{Consistent[T]} \Rightarrow \texttt{T}$$



$$\bot := \texttt{Inconsistent[T]}$$

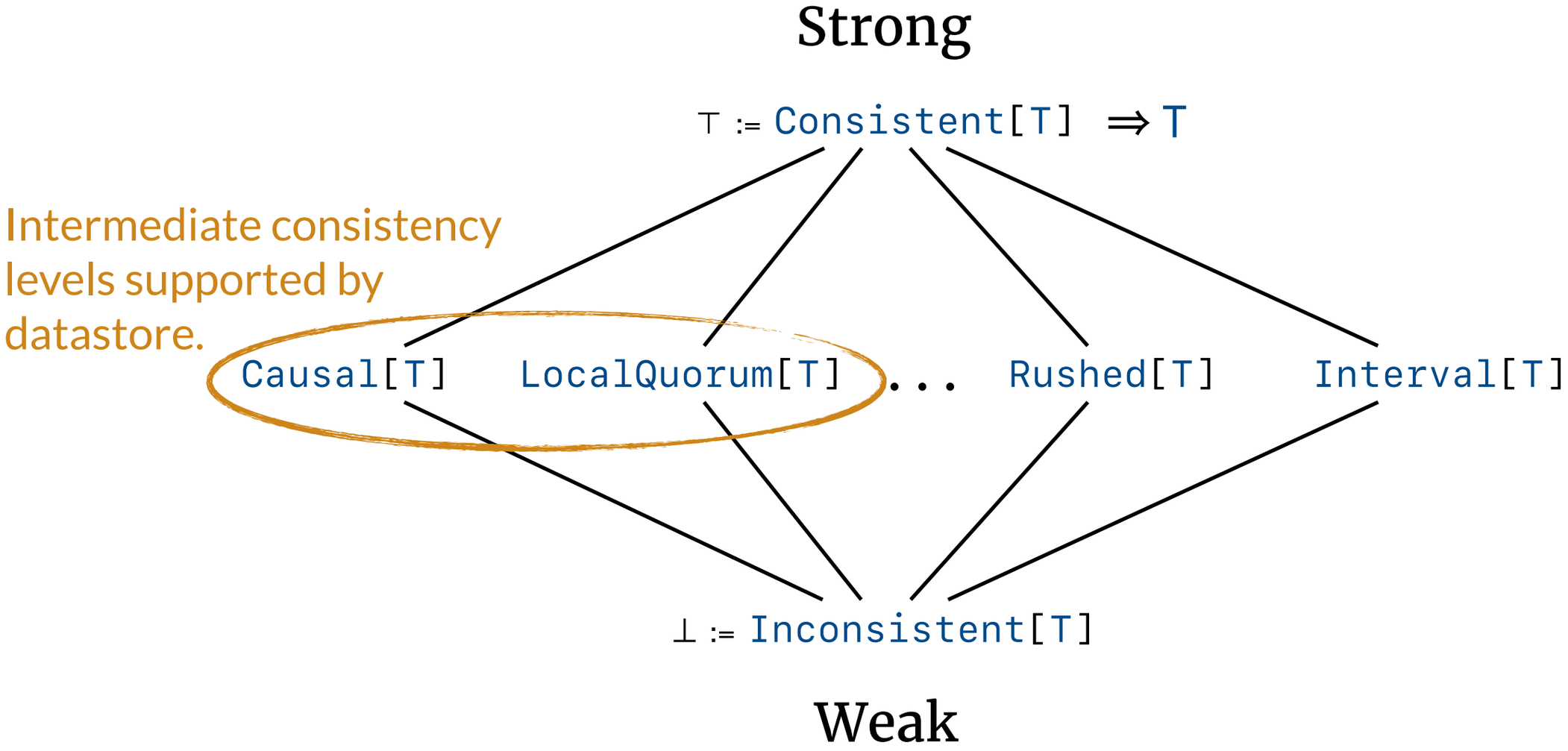
Consistency Types

***Partially ordered*, forming a *lattice*:**



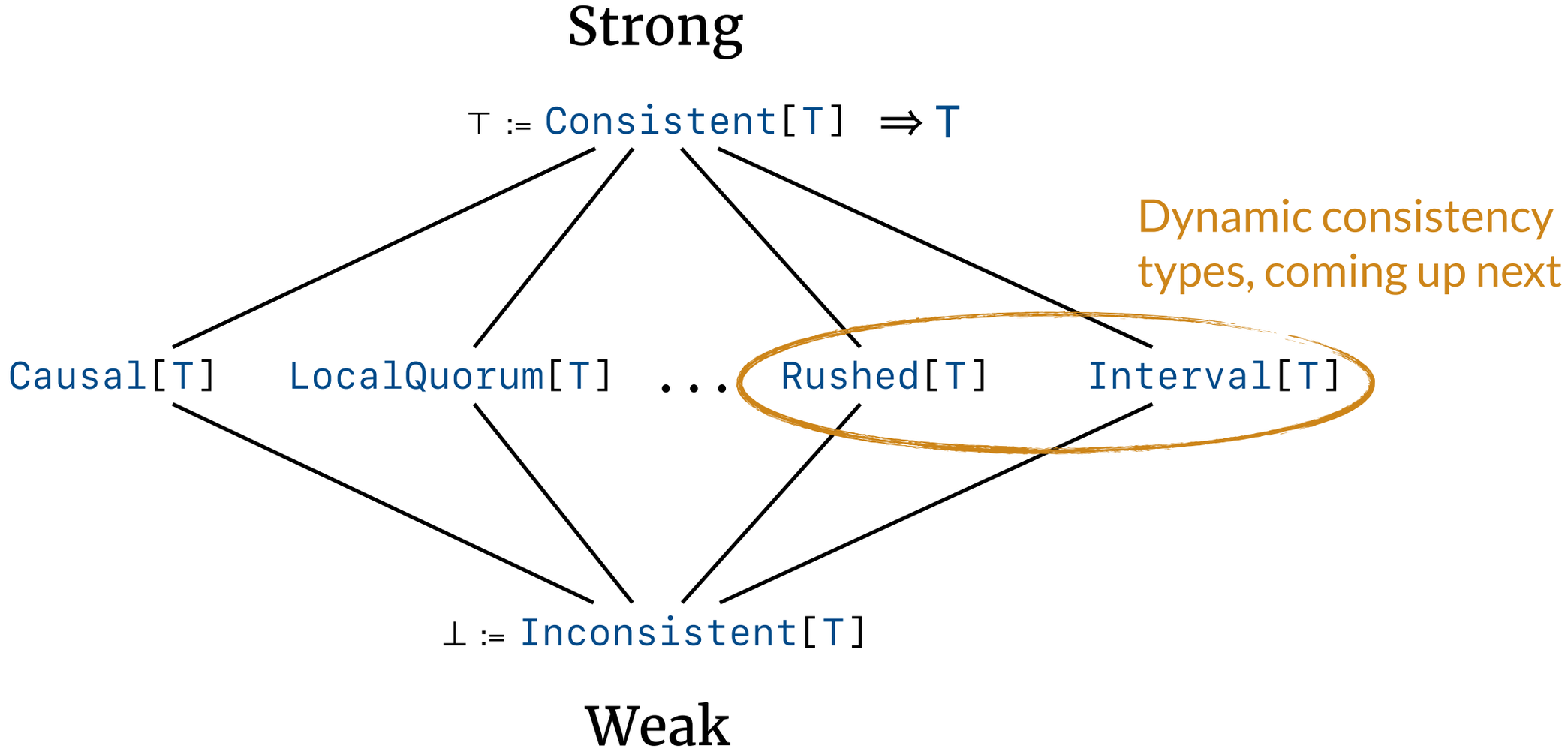
Consistency Types

***Partially ordered*, forming a *lattice*:**



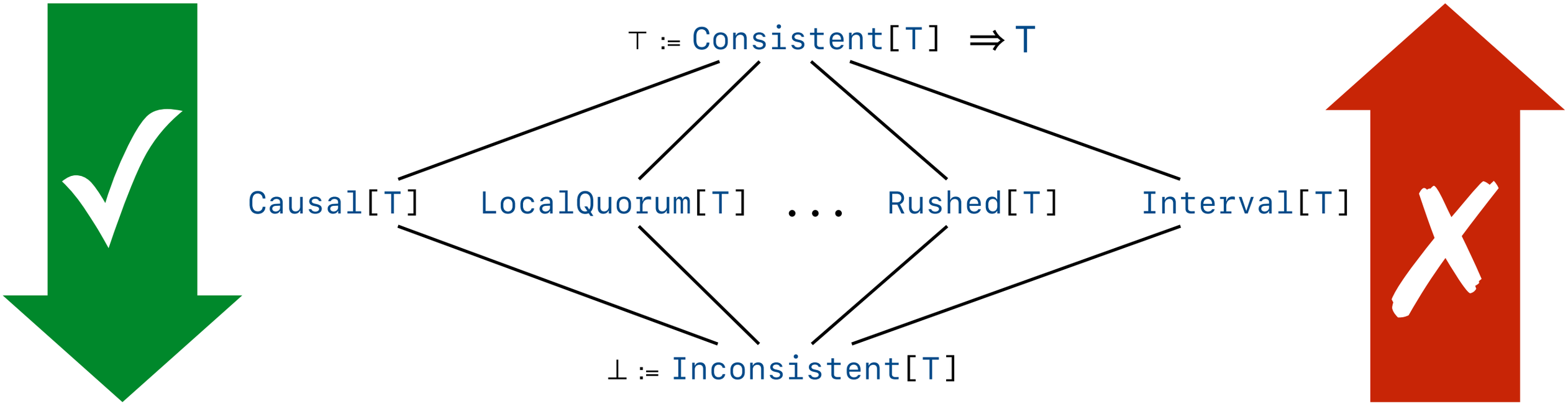
Consistency Types

***Partially ordered*, forming a *lattice*:**



Consistency Types

IPA type system enforces *consistency safety*.



***Consistency safety*:**
Weaker values cannot flow into stronger computations.

```
val n: Inconsistent[Int] = count.read()

if n < 100:
    print("Value is less than 100.")
```

Constraints

Every item in the datastore has a datatype with methods.

```
Counter:
   .increment(): Unit
   .decrement(): Unit
   .read():      Int
```

Constraints

Every item in the datastore has a datatype with methods.

Datatypes can be annotated with *consistency constraints*.

- Constraints determine the consistency type of results.

Constraints

Every item in the datastore has a datatype with methods.

Datatypes can be annotated with *consistency constraints*.

- Constraints determine the consistency type of results.

Static constraints:

```
Counter with Consistency(Strong):
  .increment(): Unit
  .decrement(): Unit
  .read():      Consistent[Int]

Counter with Consistency(Weak):
  .increment(): Unit
  .decrement(): Unit
  .read():      Inconsistent[Int]
```

Constraints

Every item in the datastore has a datatype with methods.

Datatypes can be annotated with *consistency constraints*.

- Constraints determine the consistency type of results.

Static constraints

- Enforced by the underlying datastore

```
Consistency(Strong) → Consistent[T]
Consistency(Weak) → Inconsistent[T]
```

Constraints

Every item in the datastore has a datatype with methods.

Datatypes can be annotated with *consistency constraints*.

- Constraints determine the consistency type of results.

Static constraints

- Enforced by the underlying datastore

```
Consistency(Strong) → Consistent[T]
Consistency(Weak) → Inconsistent[T]
```

Dynamic constraints

- Performance targets:

```
LatencyBound(50 ms) → Rushed[T]
```

- Approximation opportunity:

```
ErrorTolerance(10%) → Interval[T]
```

Constraints: Latency bounds

Need result in a fixed amount of time.

- Dynamically adjust consistency to meet the target latency.

```
Counter with LatencyBound(50 ms):
  .increment(): Unit
  .read():      Rushed[Int]
```

? `Consistent[Int]` `LocalQuorum[Int]` `Inconsistent[Int]`

Return value represents achieved consistency

- **Rushed** is a ***union*** of other consistency types

```
count.read() match {
  case Consistent(x)   => print("consistent")
  case LocalQuorum(x)  => print("locally consistent")
  case Inconsistent(x) => print("???")
}
```

Constraints: Latency bounds

Implementation:

- Send requests to # replicas needed for strong consistency

```
Counter with LatencyBound(50 ms):
  .increment(): Unit
  .read():      Rushed[Int]
```

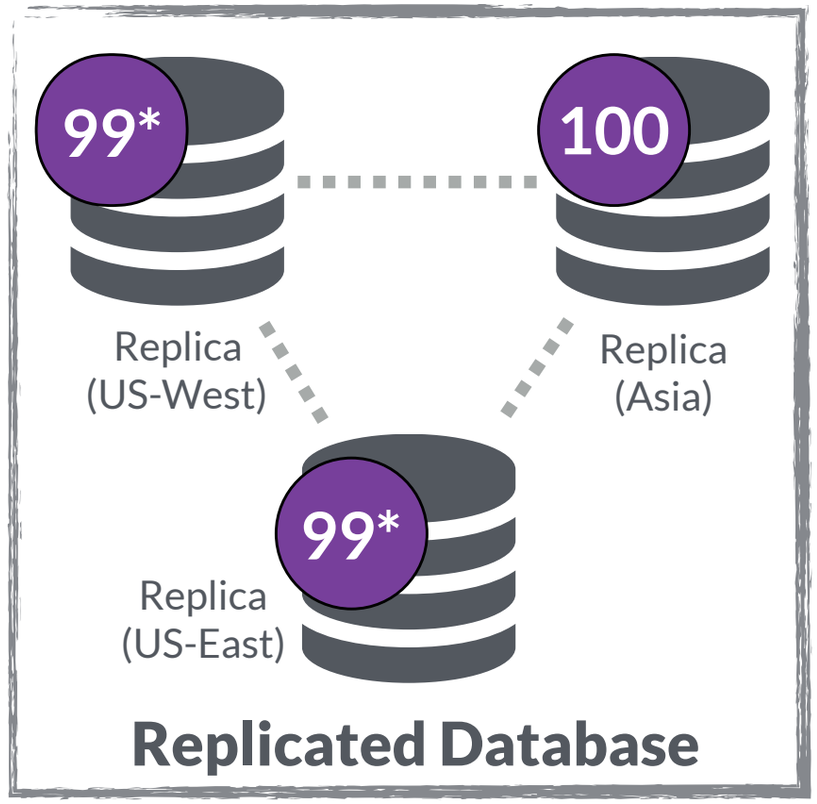
Calculating consistency

Strong consistency is achieved if:

$$W + R > N$$

(W = # replicas written to,
R = # replicas read from,
N = total # replicas)

cassandra



Constraints: Latency bounds

Implementation:

- Send requests to # replicas needed for strong consistency

```
Counter with LatencyBound(50 ms):
  .increment(): Unit
  .read():      Rushed[Int]
```

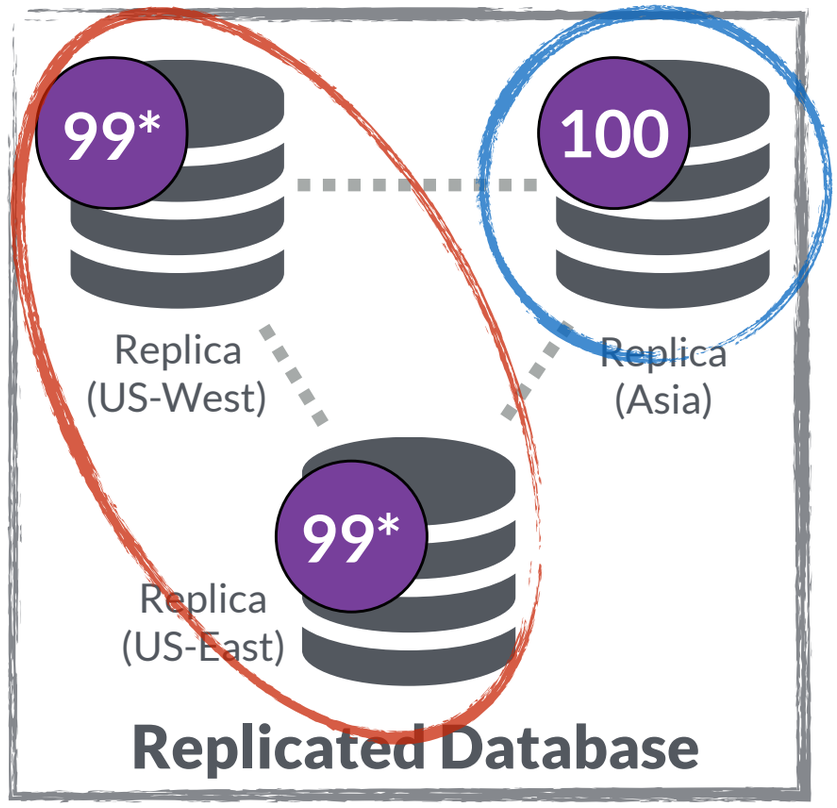
Calculating consistency

Strong consistency is achieved if:

$$W + R > N$$

(W = # replicas written to,
R = # replicas read from,
N = total # replicas)

cassandra



Constraints: Latency bounds

Implementation:

- Send requests to # replicas needed for strong consistency

```
Counter with LatencyBound(50 ms):
  .increment(): Unit
  .read():      Rushed[Int]
```

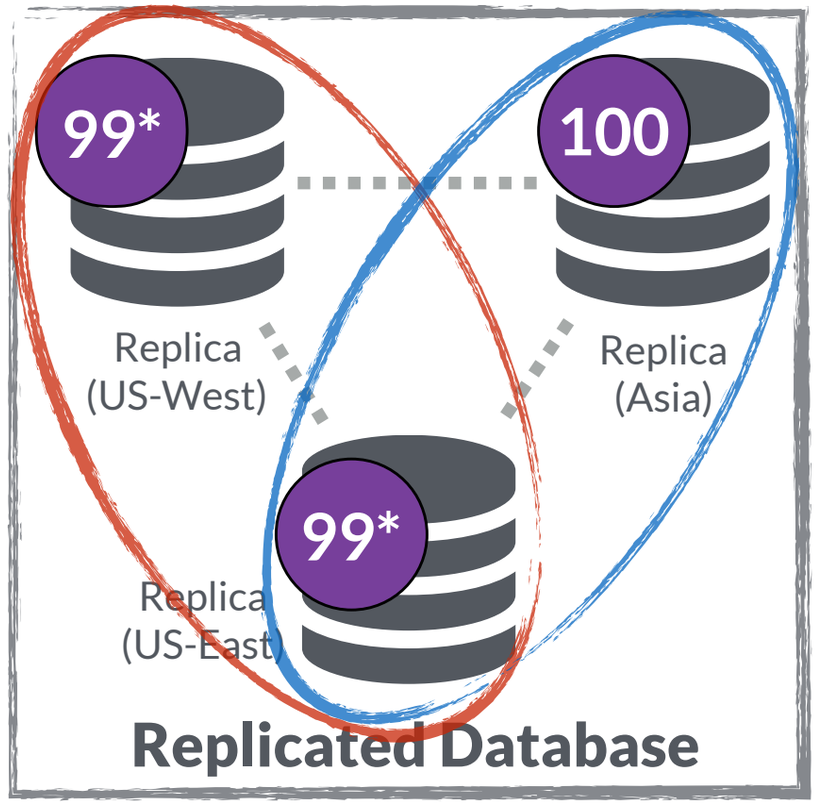
Calculating consistency

Strong consistency is achieved if:

$$W + R > N$$

(W = # replicas written to,
R = # replicas read from,
N = total # replicas)

cassandra

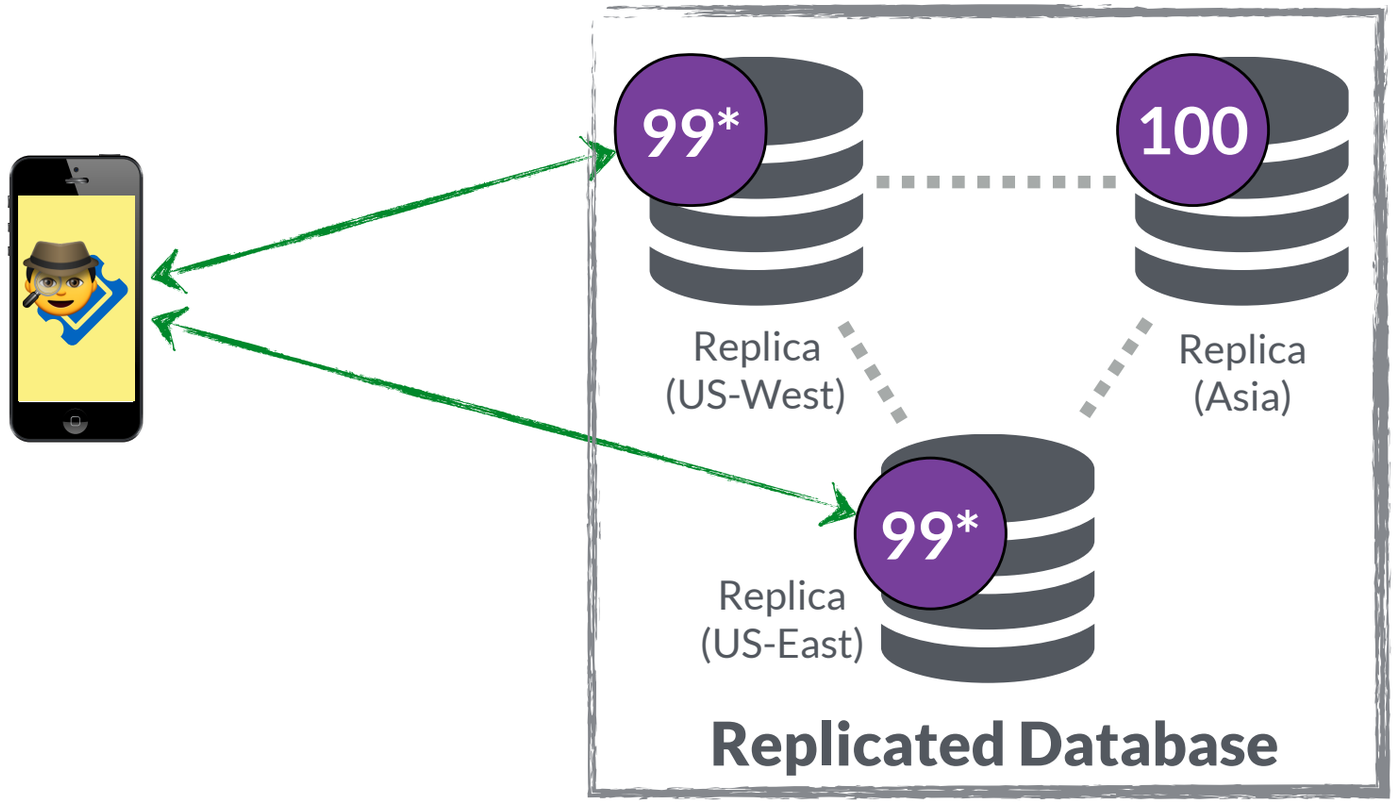


Constraints: Latency bounds

Implementation:

- Send requests to # replicas needed for strong consistency

```
Counter with LatencyBound(50 ms):
  .increment(): Unit
  .read():      Rushed[Int]
```

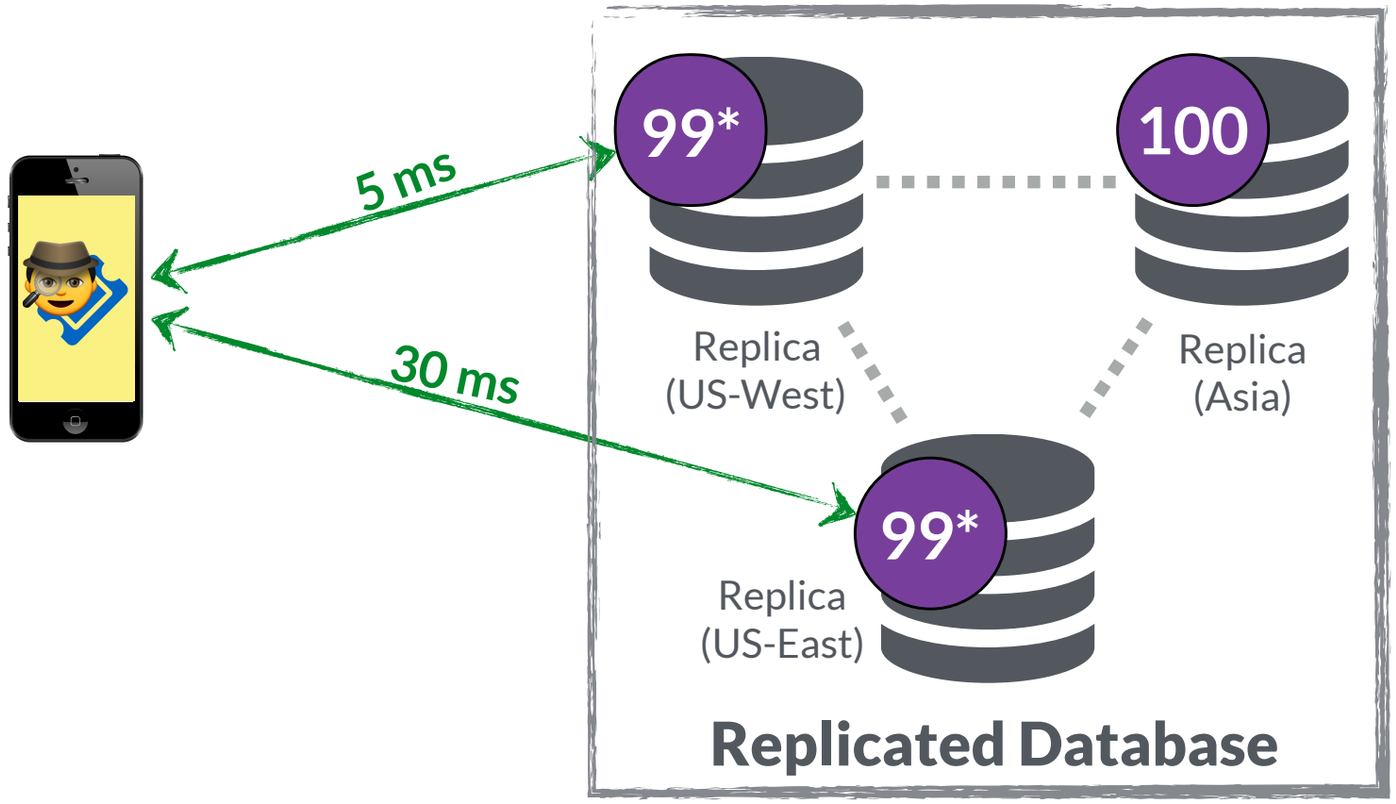


Constraints: Latency bounds

Implementation:

- Send requests to # replicas needed for strong consistency
- Wait for time limit

```
Counter with LatencyBound(50 ms):
  .increment(): Unit
  .read():      Rushed[Int]
```

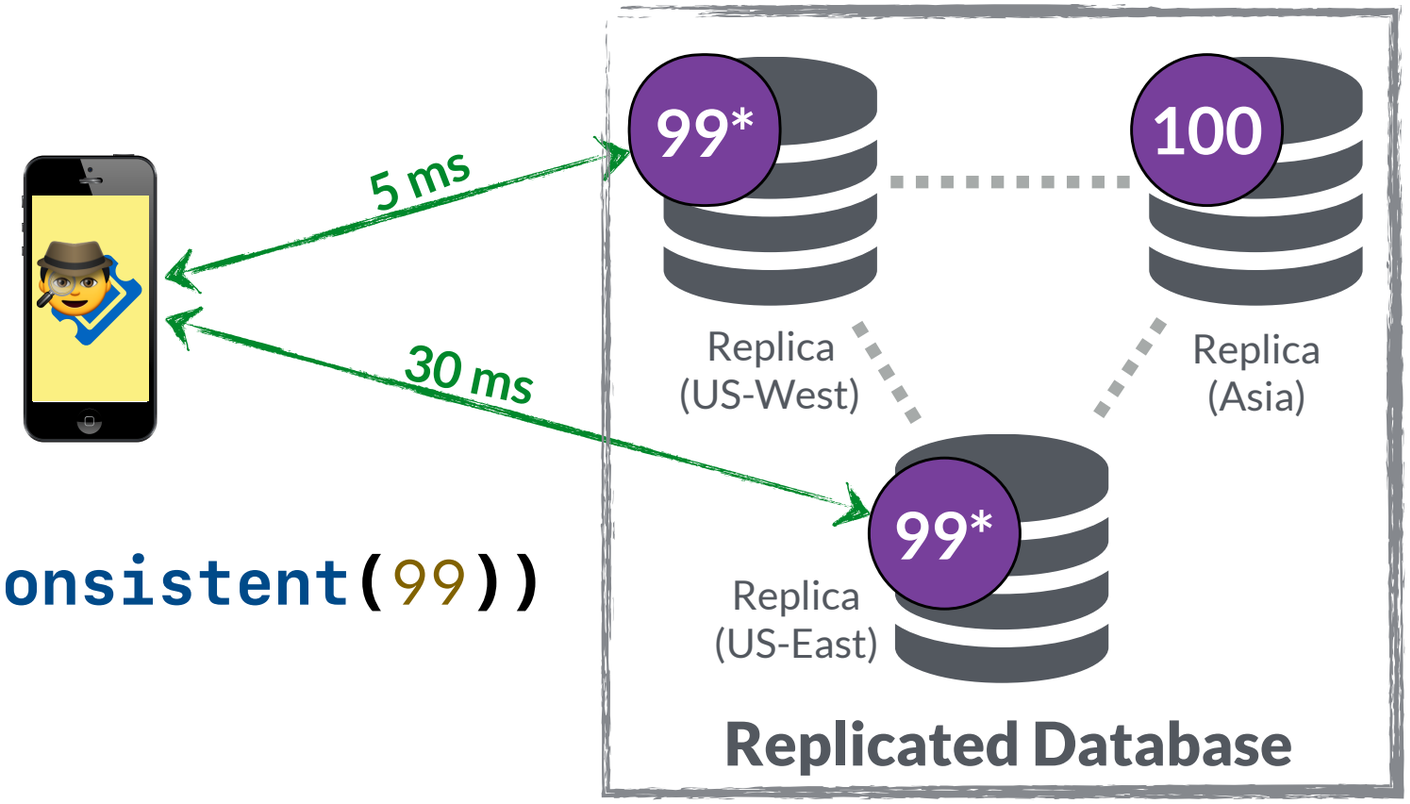


Constraints: Latency bounds

Implementation:

- Send requests to # replicas needed for strong consistency
- Wait for time limit
- Resulting consistency determined by number of on-time responses

```
Counter with LatencyBound(50 ms):
  .increment(): Unit
  .read():      Rushed[Int]
```



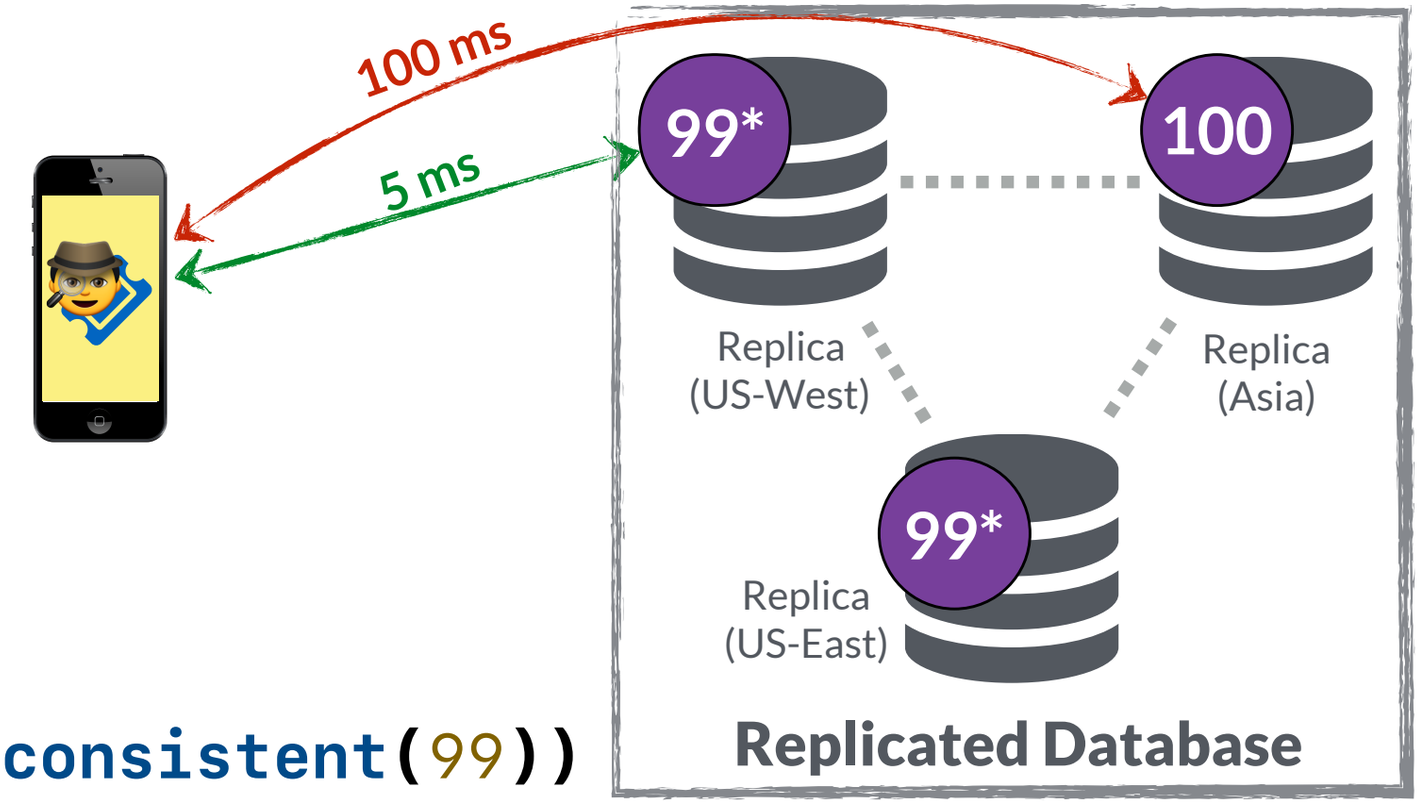

```
Rushed[Int](Consistent(99))
```

Constraints: Latency bounds

Implementation:

- Send requests to # replicas needed for strong consistency
- Wait for time limit
- Resulting consistency determined by number of on-time responses

```
Counter with LatencyBound(50 ms):
  .increment(): Unit
  .read():      Rushed[Int]
```




```
Rushed[Int](Inconsistent(99))
```

Constraints: Error bounds

Tolerate a fixed amount of *inaccuracy* in value.

```
Counter with ErrorTolerance(10%):
  .decrement(): Unit
  .read(): Interval[Int]
```

Return value represents a range of possible consistent values

- Programmer must decide what to do with the `Interval` value

```
val remaining = ticketCount.read()
if (remaining.min > 100) {
  display("At least 100 tickets left.")
}
```

Works by limiting outstanding writes

- Requires knowing the semantics of each operation

`decrement() × (0..10)` ⇒ `read() = Interval[Int](90,100)`

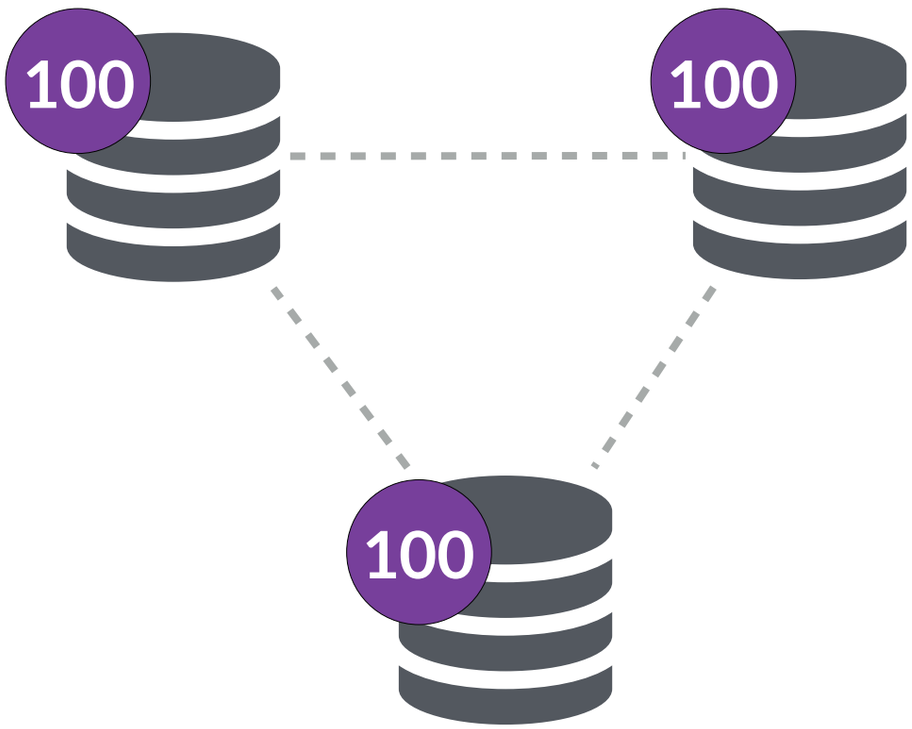
Constraints: Error bounds

Implementation:

- **Reservations:** pre-allocate permission to do updates, distribute among replicas.

```
Counter with ErrorTolerance(10%):
  .decrement(): Unit
  .read(): Interval[Int]
```

10 × `decr()`

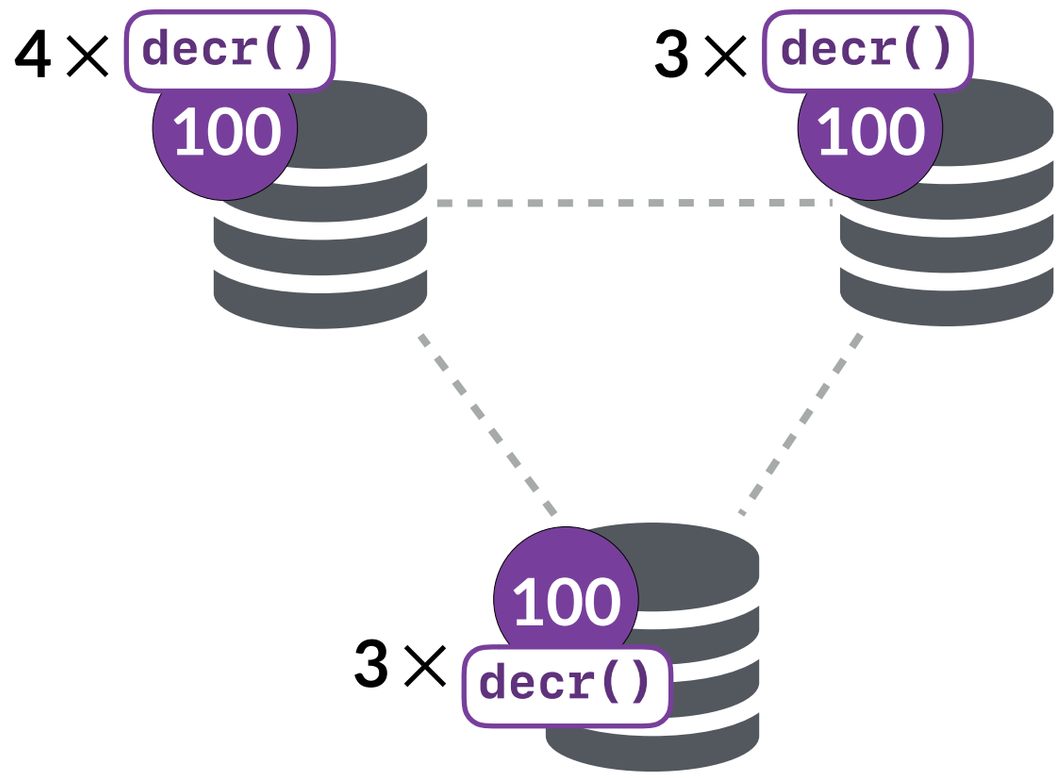


Constraints: Error bounds

Implementation:

- **Reservations:** pre-allocate permission to do updates, distribute among replicas.

```
Counter with ErrorTolerance(10%):
  .decrement(): Unit
  .read(): Interval[Int]
```

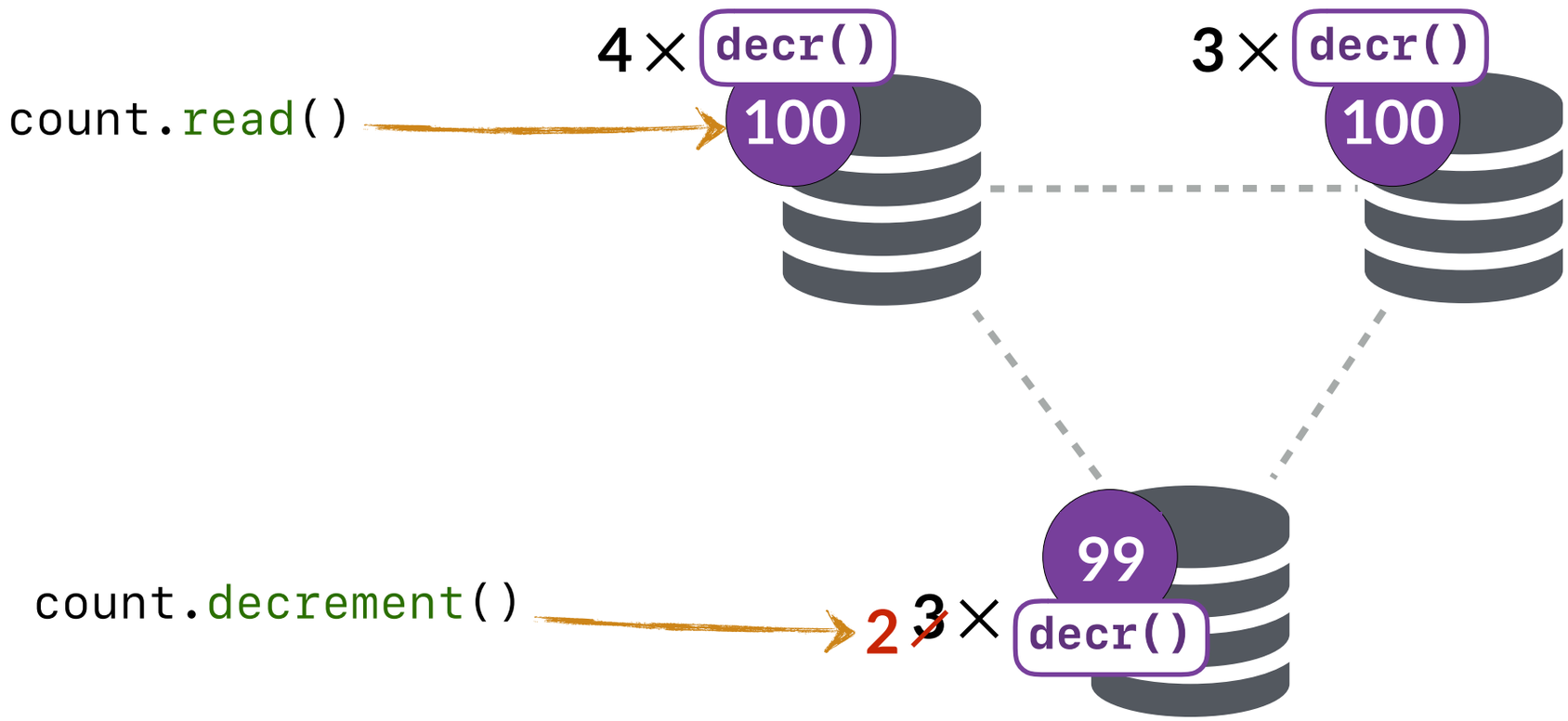


Constraints: Error bounds

Implementation:

- **Reservations:** pre-allocate permission to do updates, distribute among replicas.

```
Counter with ErrorTolerance(10%):
  .decrement(): Unit
  .read(): Interval[Int]
```

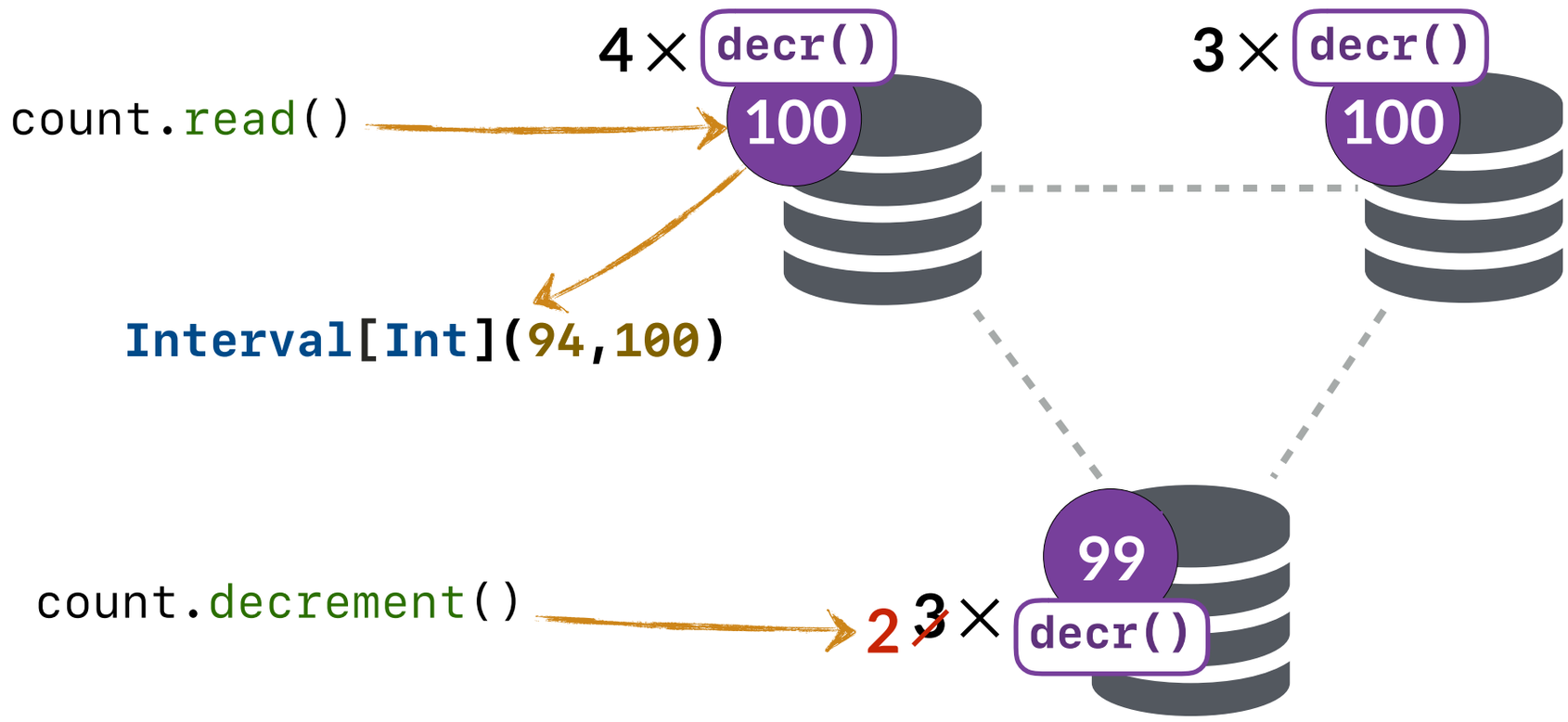


Constraints: Error bounds

Implementation:

- **Reservations:** pre-allocate permission to do updates, distribute among replicas.

```
Counter with ErrorTolerance(10%):
  .decrement(): Unit
  .read(): Interval[Int]
```

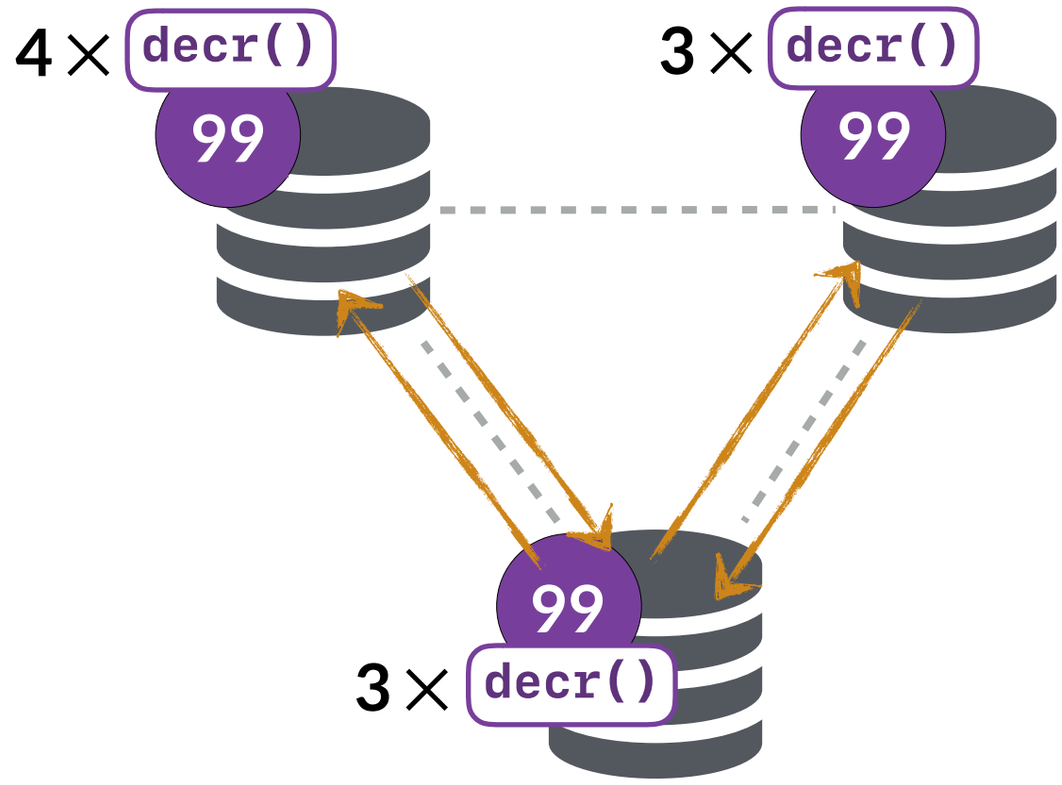


Constraints: Error bounds

Implementation:

- **Reservations:** pre-allocate permission to do updates, distribute among replicas.
- Refresh permissions on sync.

```
Counter with ErrorTolerance(10%):
  .decrement(): Unit
  .read(): Interval[Int]
```

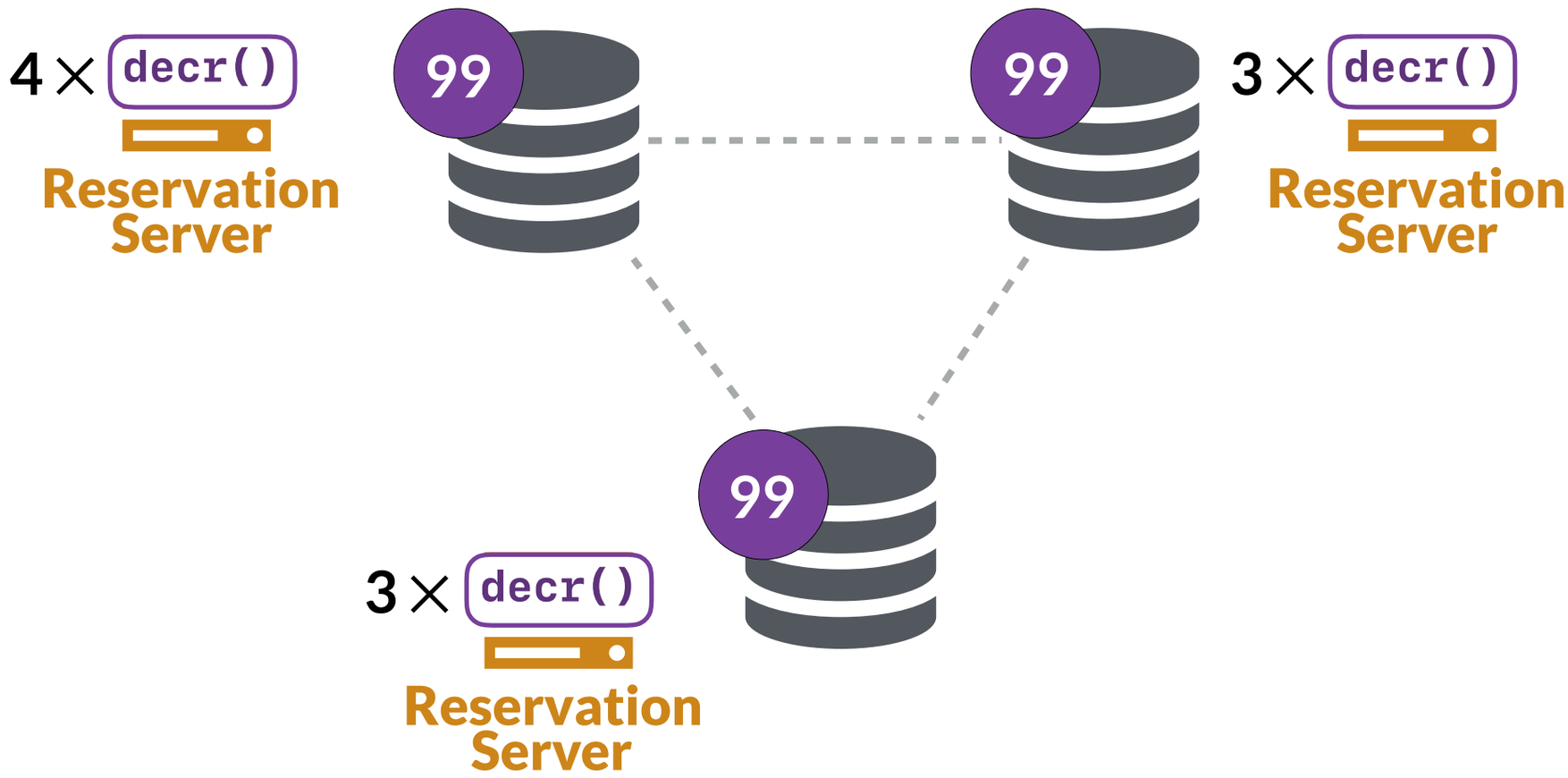


Constraints: Error bounds

Implementation:

- **Reservations:** pre-allocate permission to do updates, distribute among replicas.
- Refresh permissions on sync.
- **Reservation servers** mediate requests.

```
Counter with ErrorTolerance(10%):
  .decrement(): Unit
  .read(): Interval[Int]
```



Inconsistent
Performance-bound
Approximate

} **Programming Model**

Trading off consistency

Consistency types

- Statically prevent consistency bugs

Performance & accuracy constraints

- Dynamically adjust consistency

Case study

Case Study: TicketSleuth

Performance constraint:
< 50 ms latency

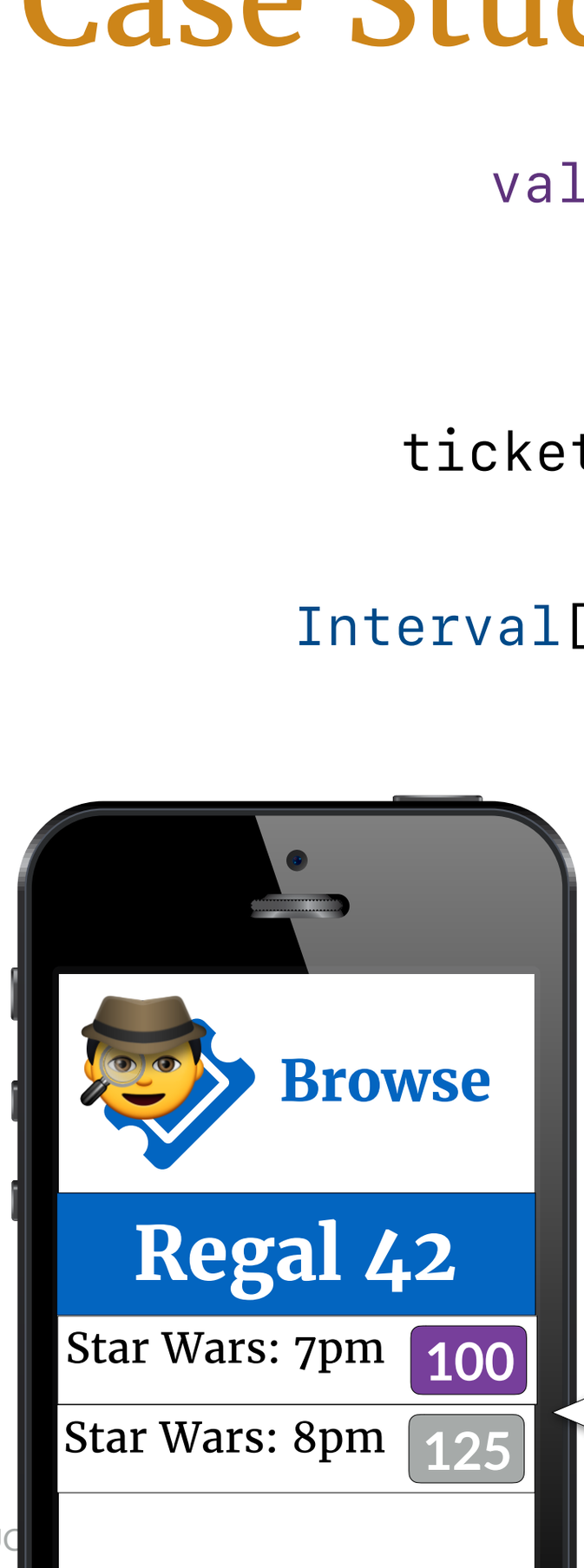

```
val events = List[EventID]()
                    with LatencyBound(50 ms)

events("Regal42:StarWars").range(0,2)

Rushed(Inconsistent(List("7pm","8pm")))

endorse
```

Case Study: TicketSleuth

```
val tickets = UUIDPool()
                with ErrorTolerance(count,5%)
```

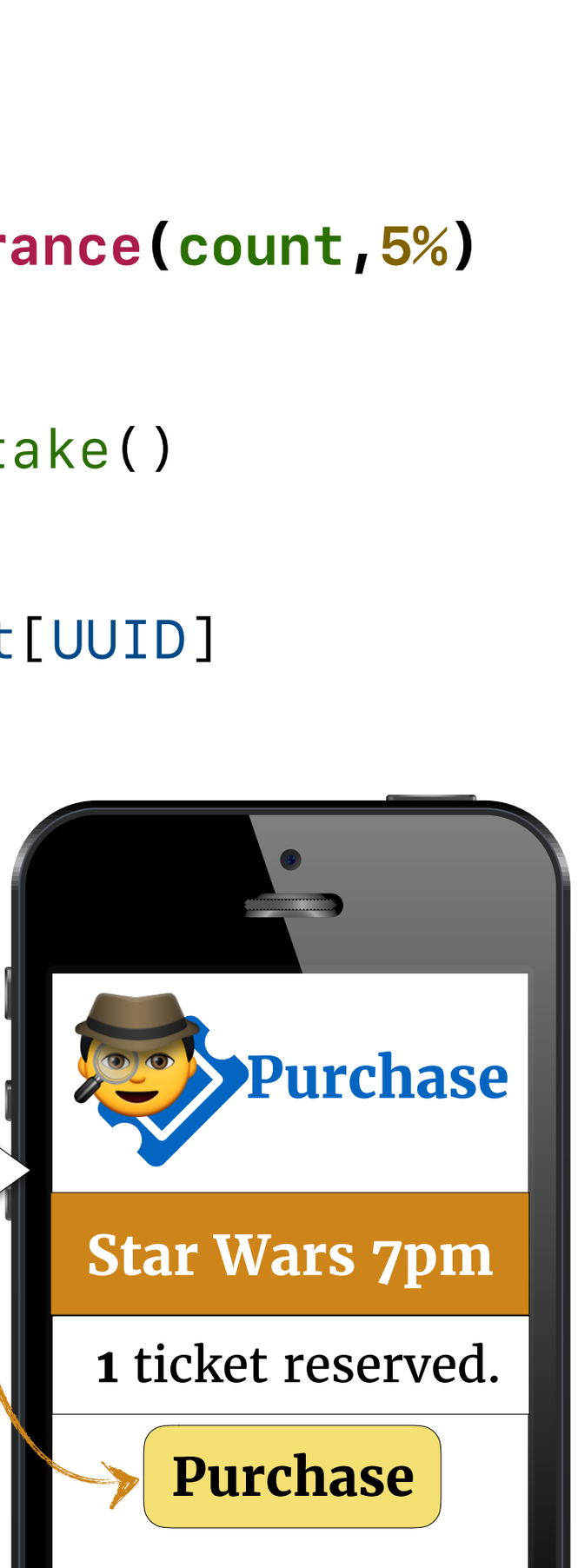
`tickets.count()` → `Interval[Int](96,100)` → `.max`

`tickets.take()` → `Consistent[UUID]`

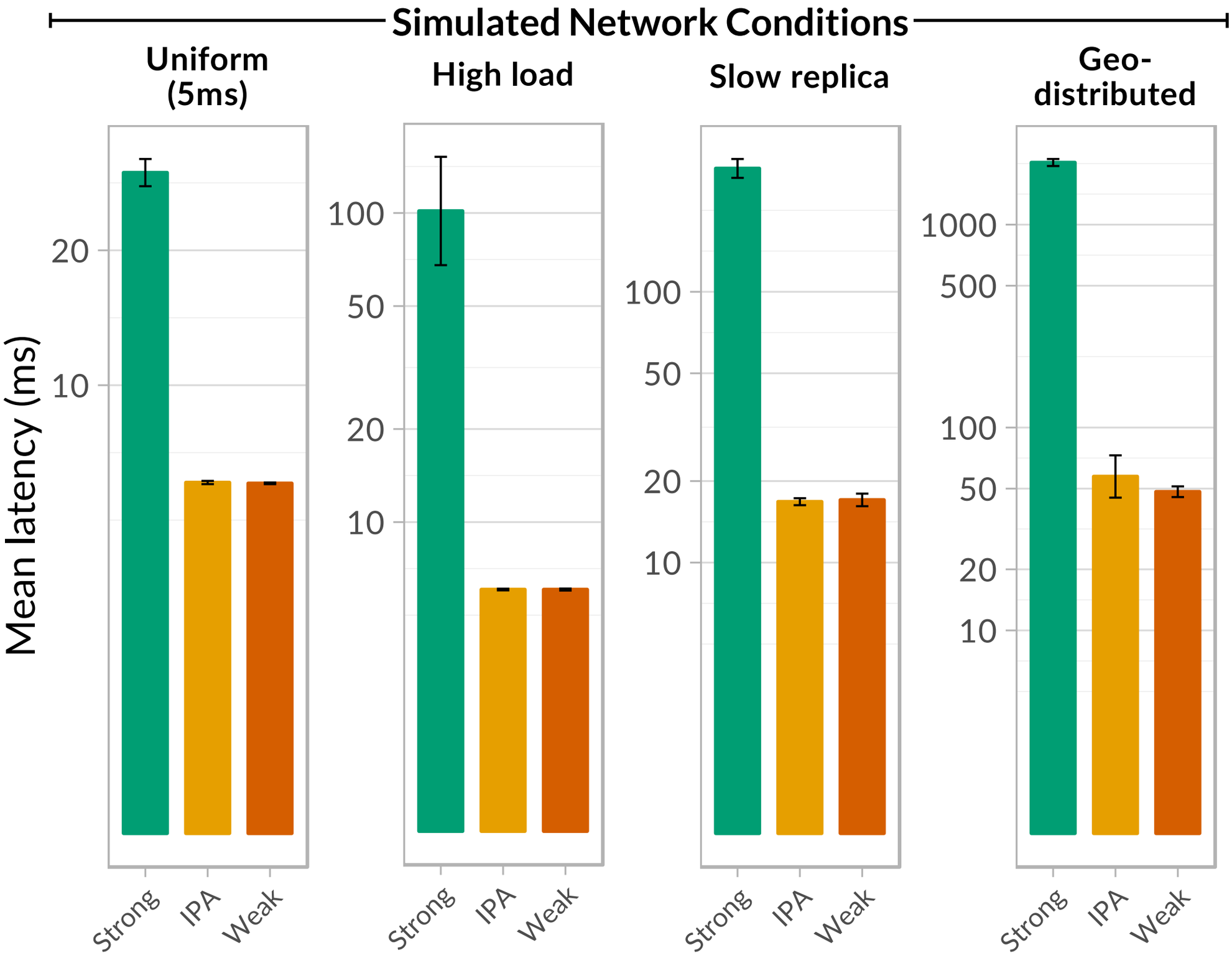


Error tolerance:
count does not need to be exact…

Hard constraint:
do not over-sell tickets



Case Study: TicketSleuth



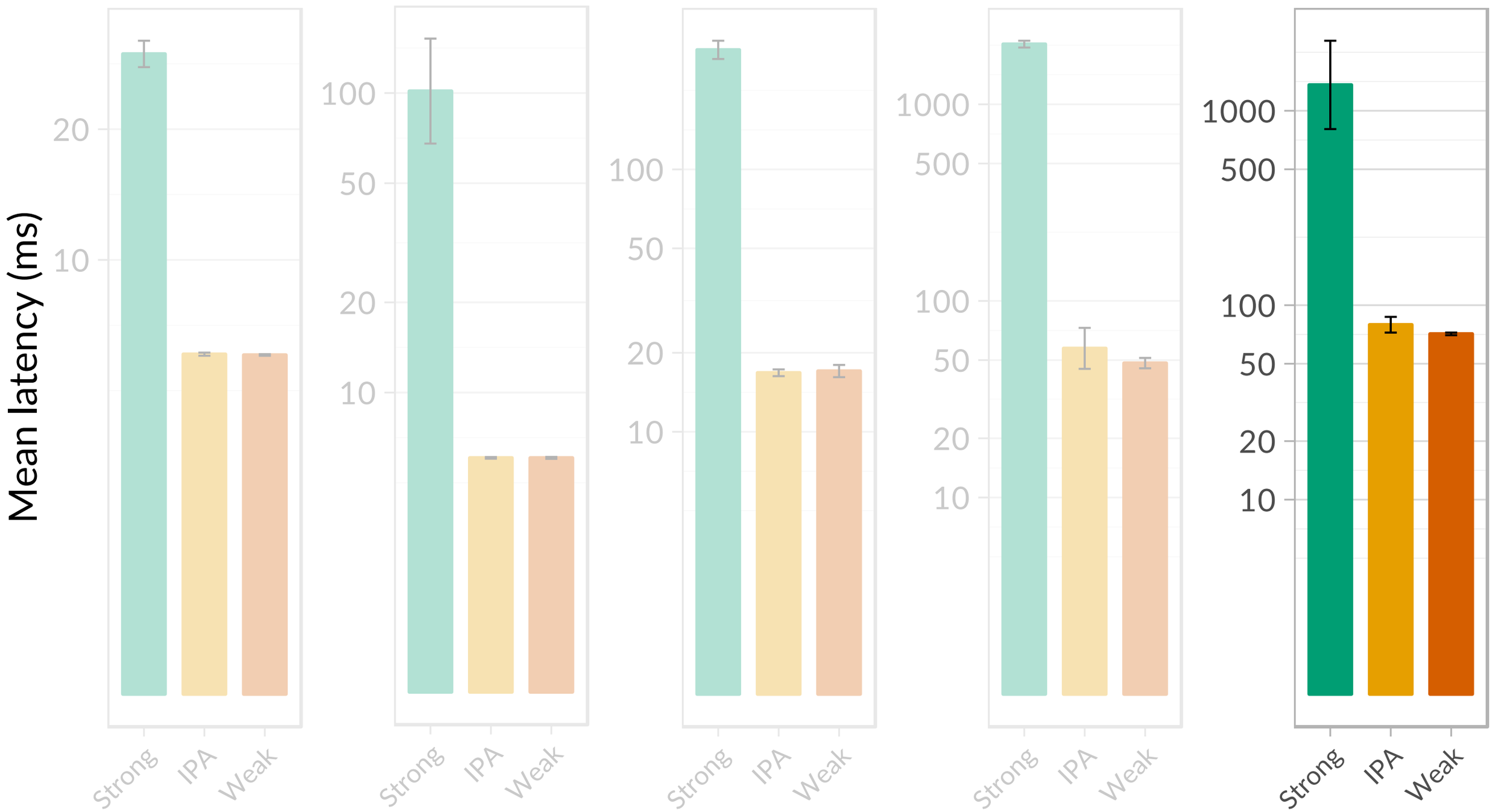
Case Study: TicketSleuth

VMs in US, Europe, and Asia

Simulated Network Conditions

Uniform (5ms) | High load | Slow replica | Geo-distributed | Google Compute Engine

Mean latency (ms)

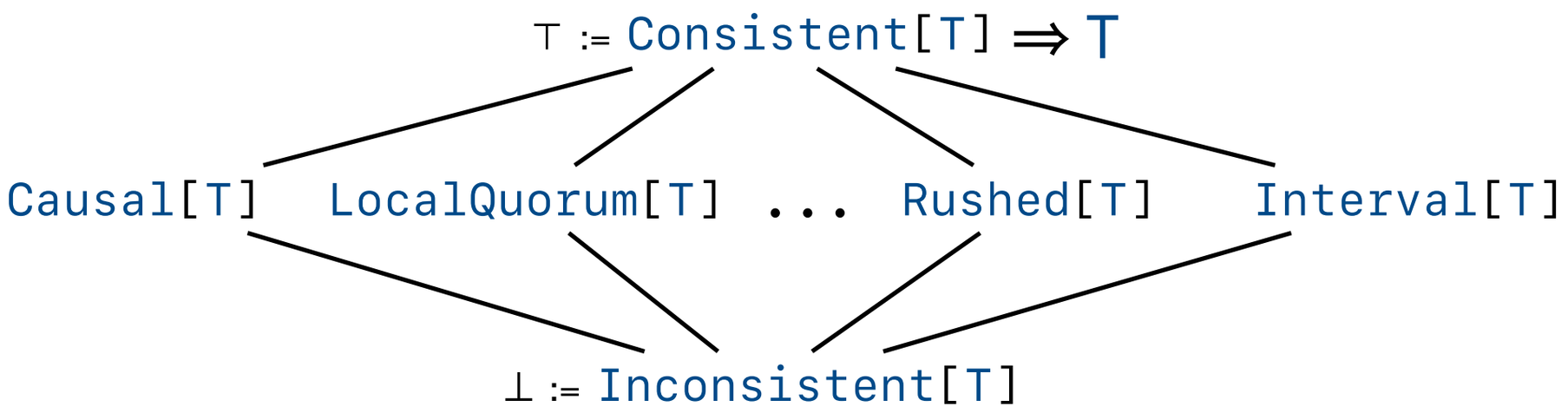


Disciplined Inconsistency

with *Consistency Types*

Inconsistent
Performance-bound
Approximate

} **Programming Model**



Discussion

Mixed consistency systems

- Geo-distributed databases
- Multiple layers of caches and downstream services
- Example: Facebook's TAO graph store

Data-type- vs operation-centric consistency

- Why do we usually annotate individual operations with consistency rather than annotating data types?
- How does this relate to CRDTs?

Other uses for reservations

Verifying or synthesizing ADT/constraint implementations

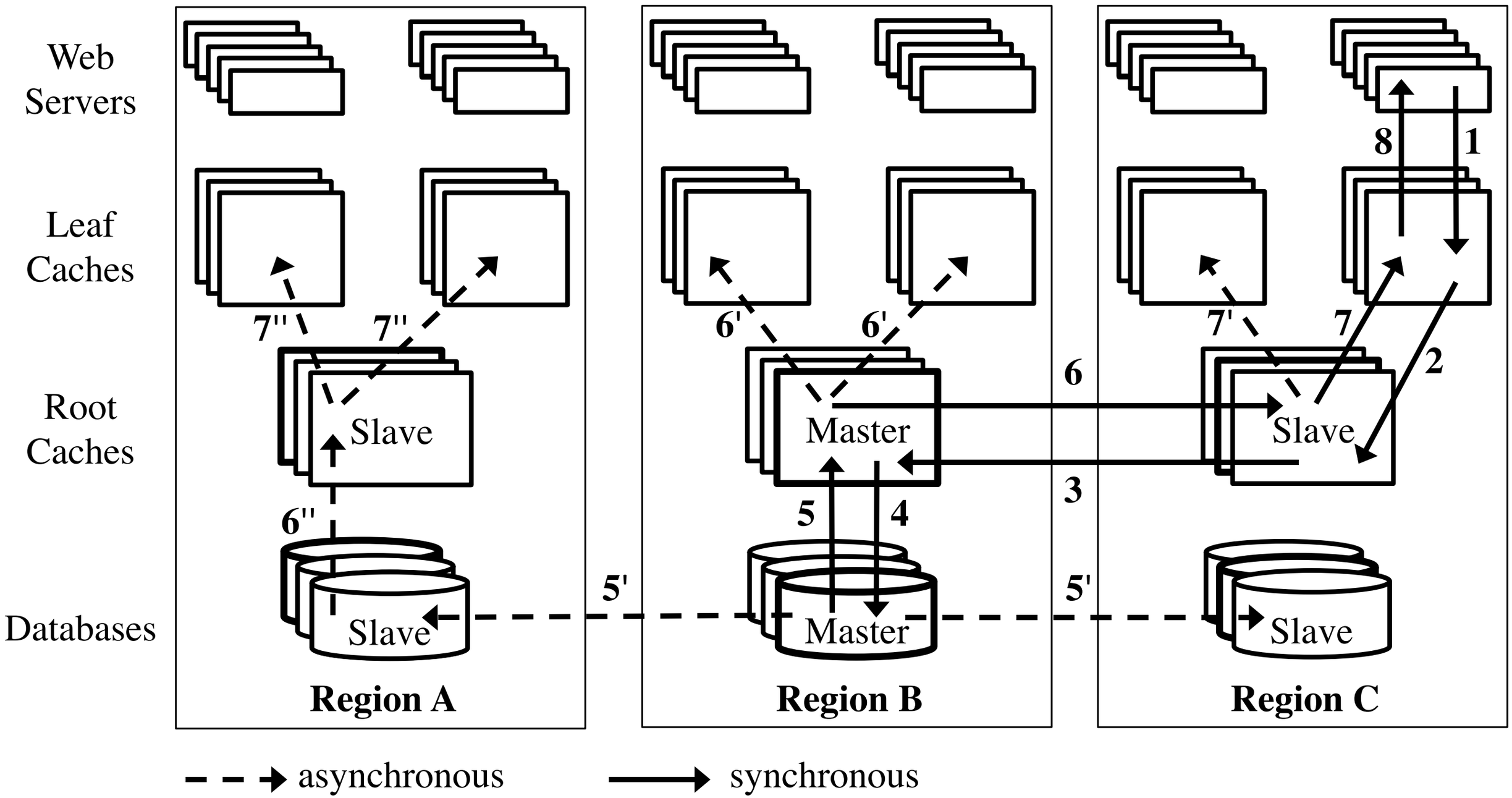
Composition of multiple operations or datatypes

Facebook TAO

Haonan Lu*†, Kaushik Veeraraghavan†, Philippe Ajoux†, Jim Hunt†,
Yee Jiun Song†, Wendy Tobagus†, Sanjeev Kumar†, Wyatt Lloyd*†

*University of Southern California, †Facebook, Inc.

(SOSP'15)



System architecture of TAO graph store.

Facebook TAO

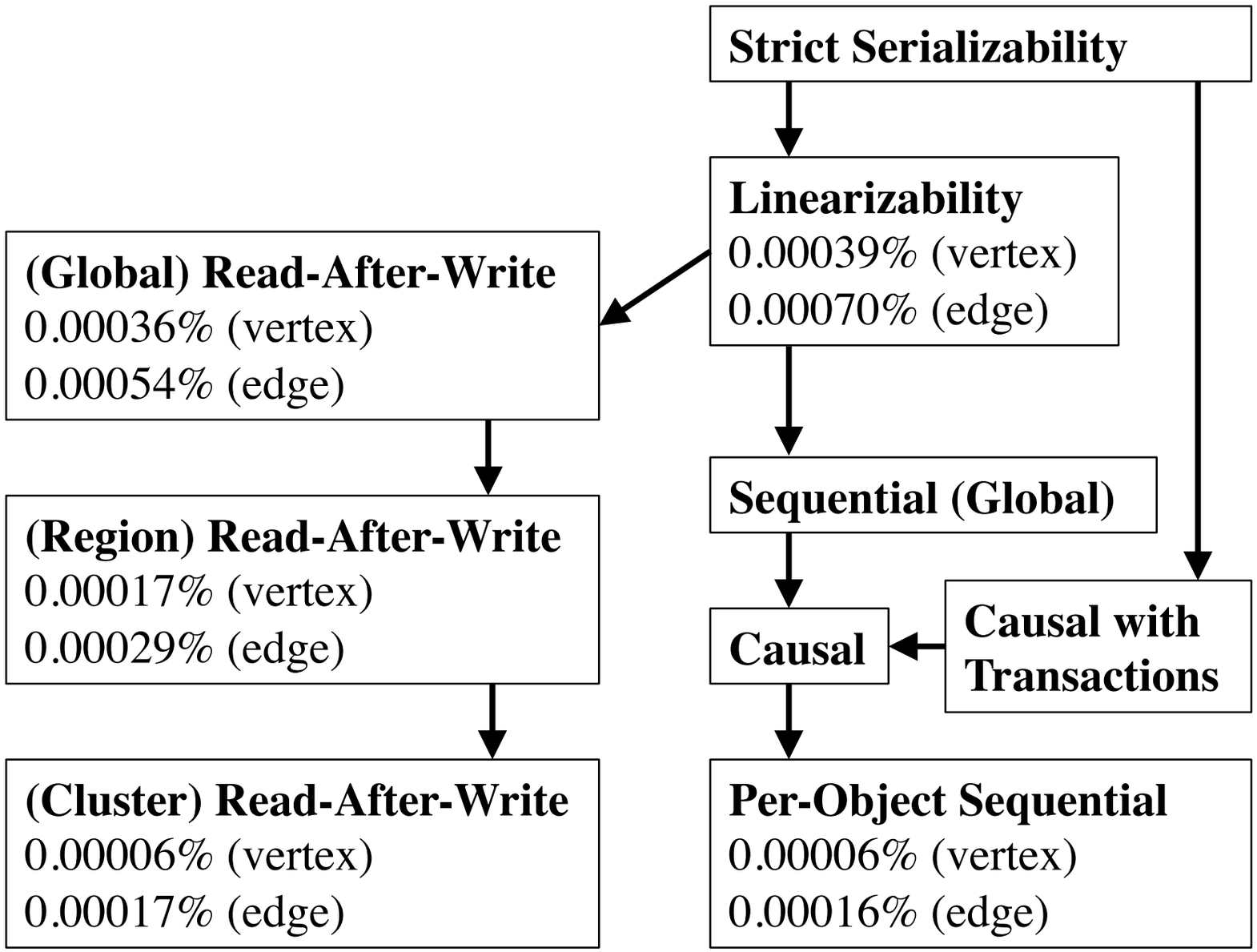


Figure 8: Spectrum of consistency models with arrows from stronger to weaker models. Measured anomaly rates are shown. Bounds on the anomaly rate in unmeasureable non-local consistency models can be inferred.

Case Study: Twitter clone

```
class Tweet(
  id: TweetID,
  user: UserID,
  body: String,
  retweets: Set[UserID] with ErrorTolerance(size,5%)
)
```

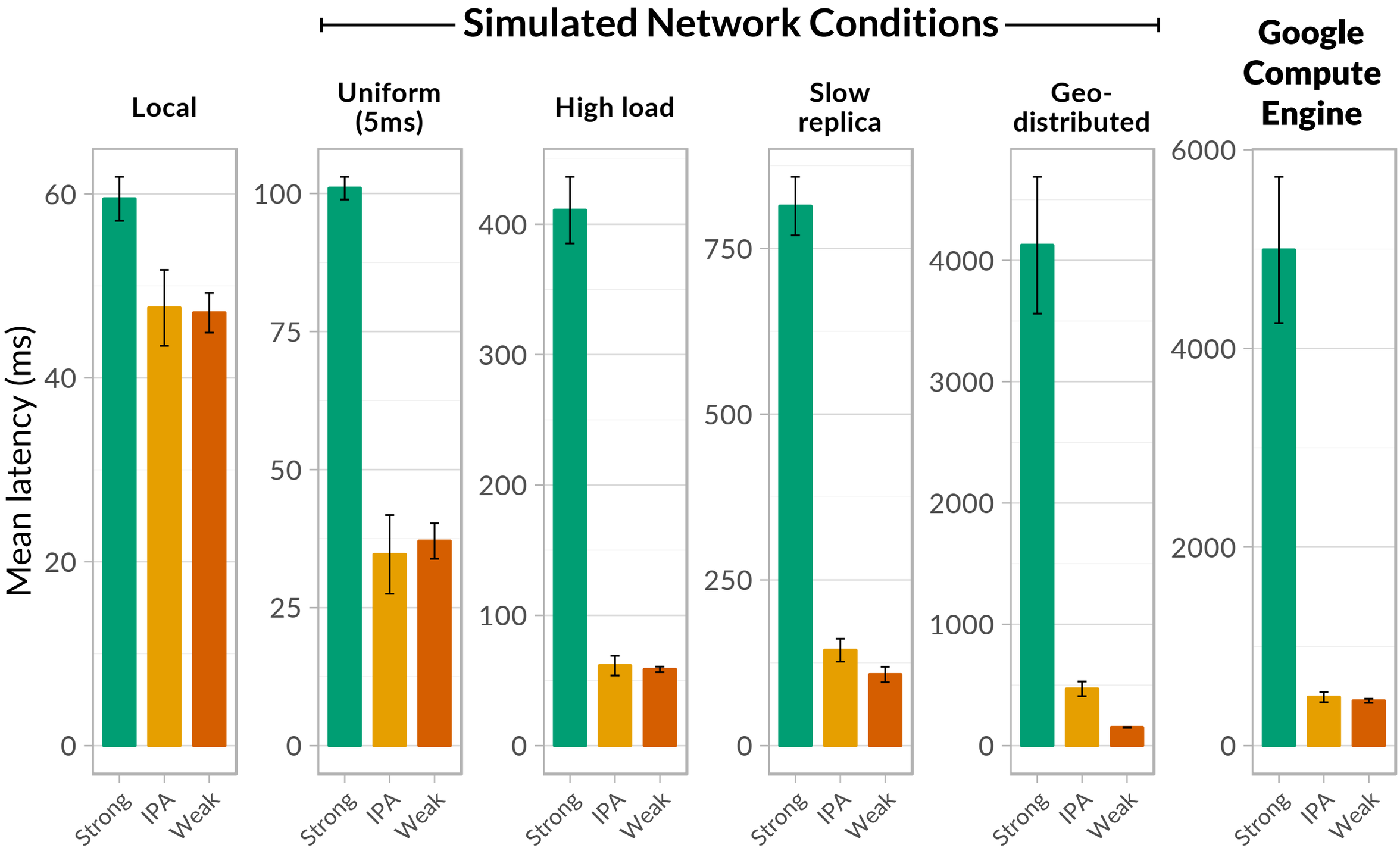
Exact count for average tweets.

Approximate count for popular tweets

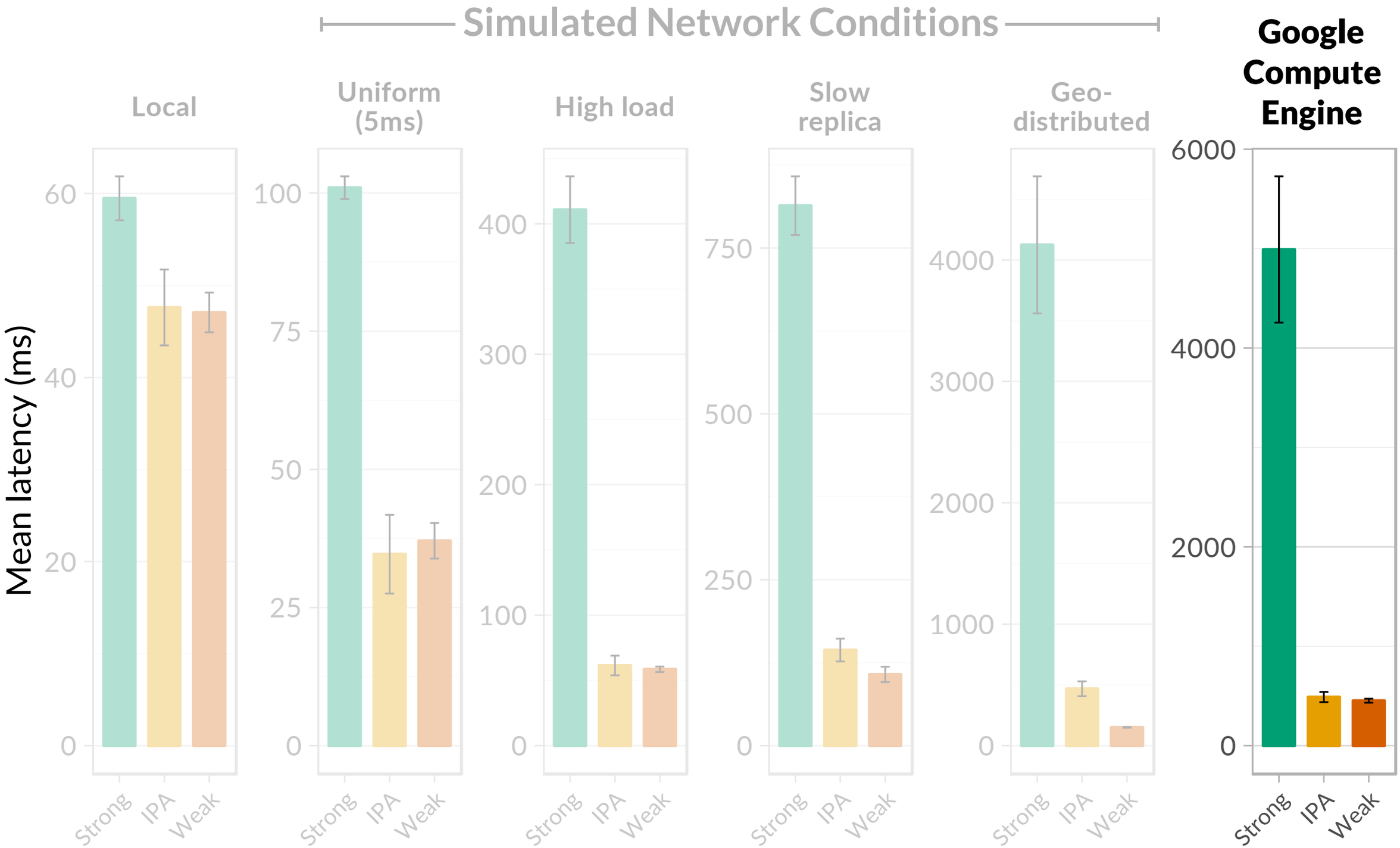
Case Study: Twitter clone

```
class User(
  id: UserID,
  name: String,
  followers: Set[UserID] with LatencyBound(20 ms),
  timeline: List[TweetID] with LatencyBound(20 ms)
)
```

Case Study: Twitter clone



Case Study: Twitter clone



Consistency-based SLAs

(SOSP'13)

Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, Hussam Abu-Libdeh

- SLAs specify consistency and target latency.
- Operations specify a set of SLAs with utility values.
- System monitors and predicts which SLA can be met, tries to maximize utility.
- Speculative execution

Consistency Levels
Strong
Eventual
Read-My-Writes
Monotonic
Bounded
Causal

```
CreateTable (name);
DeleteTable (name);
tbl = OpenTable (name);

s = BeginSession(tbl, sla);
EndSession(s);

Put (s, key, value);
value, cc = Get (s, key, sla);
```

Figure 2. The Pileus API

Consistency	Latency	Utility
Strong	300 ms	1.0
Eventual	300 ms	0.5

Example: Shopping Cart

A Romp Through the Consistency Zoo

by Brandon Holt


The **Strongest**
The **Safest**
The (least) Freedom
Strong Serializable

~~Linearizable~~

Lionizable

~~Linearizable~~

Lionizable

~~Seri~~alizable

Siri

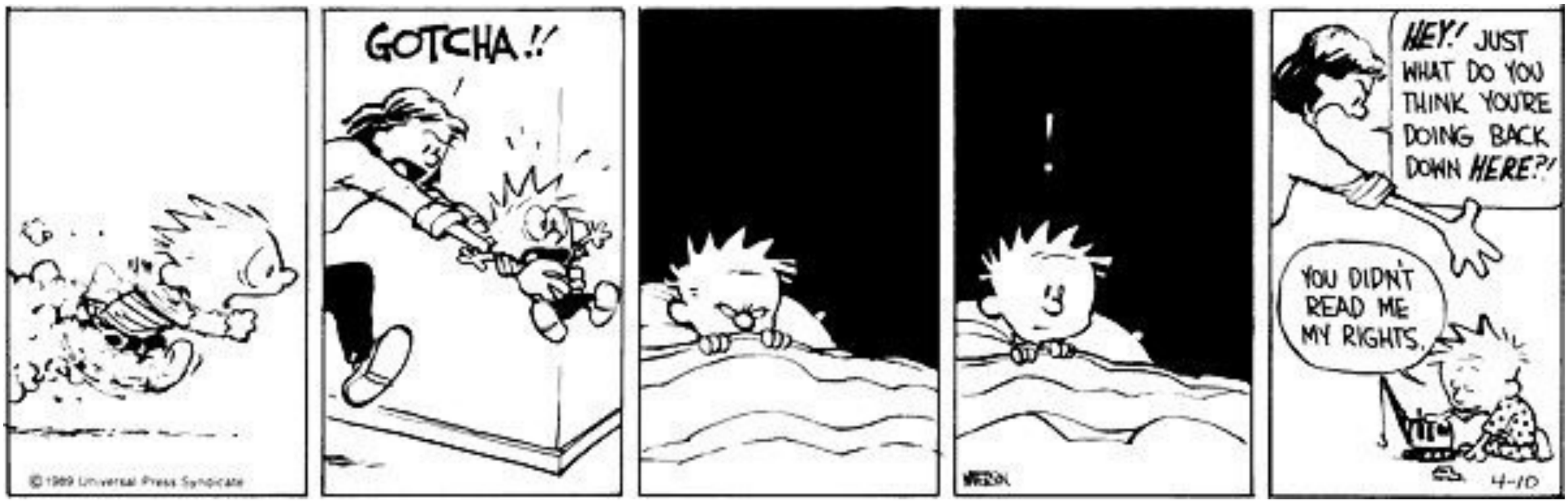
~~Serializable~~

Siri

Bet you thought I was gonna say:

Cerealizable

Read-Your-~~Writes~~
^ Rights



Monotonic consistency

PRAM

~~Causal~~ Consistency

Casual

Causal Consistency