FlightTracker

Consistency across Read-Optimized Online Stores at Facebook

Xiao Shi, Scott Pruett, Kevin Doherty, Jinyu Han,
Dmitri Petrov, Jim Carrig, John Hugg, and Nathan Bronson

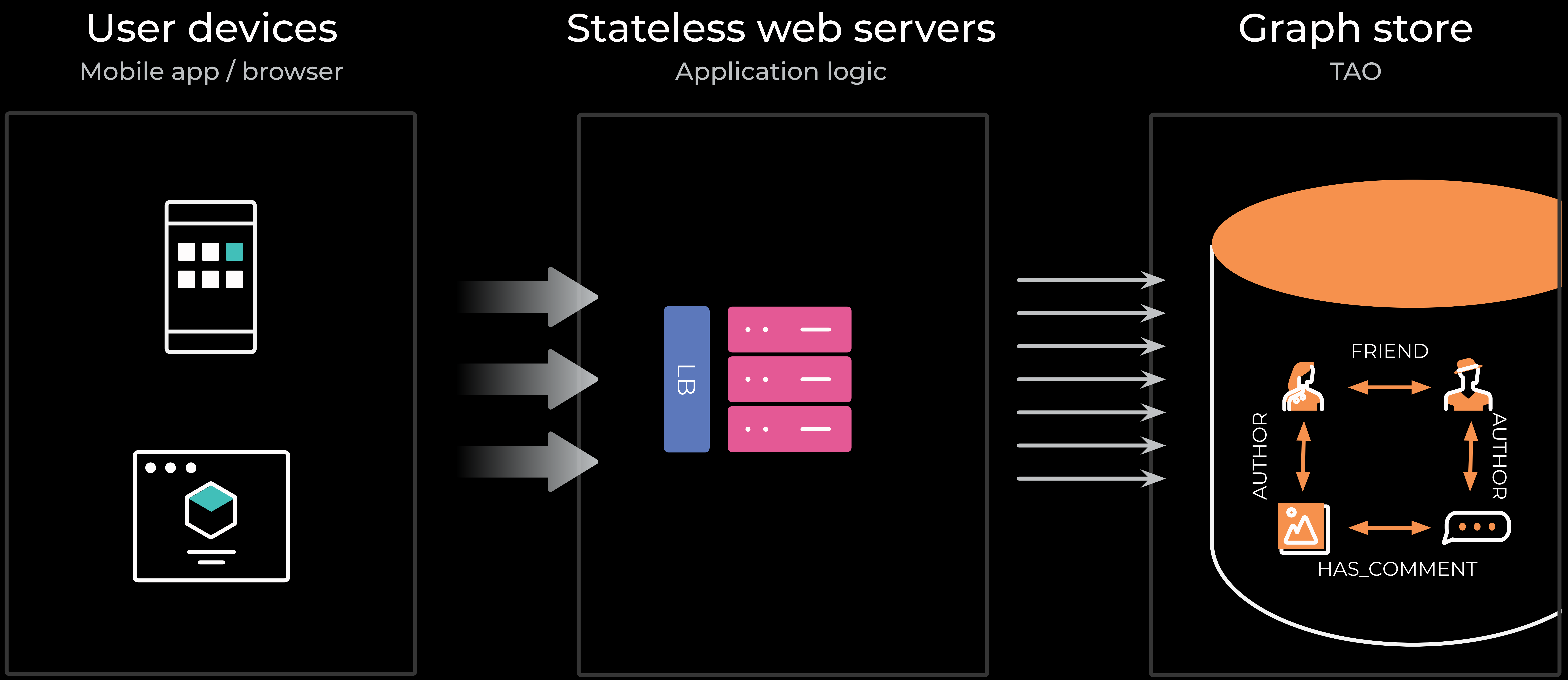
Software Engineer, Facebook Boston

Agenda

- The challenges of providing Read-Your-Writes (RYW) consistency for the social graph
- Our solution: FlightTracker
- Lessons learned and production experiences
- Q&A

TAO

Read-optimized data store for the social graph

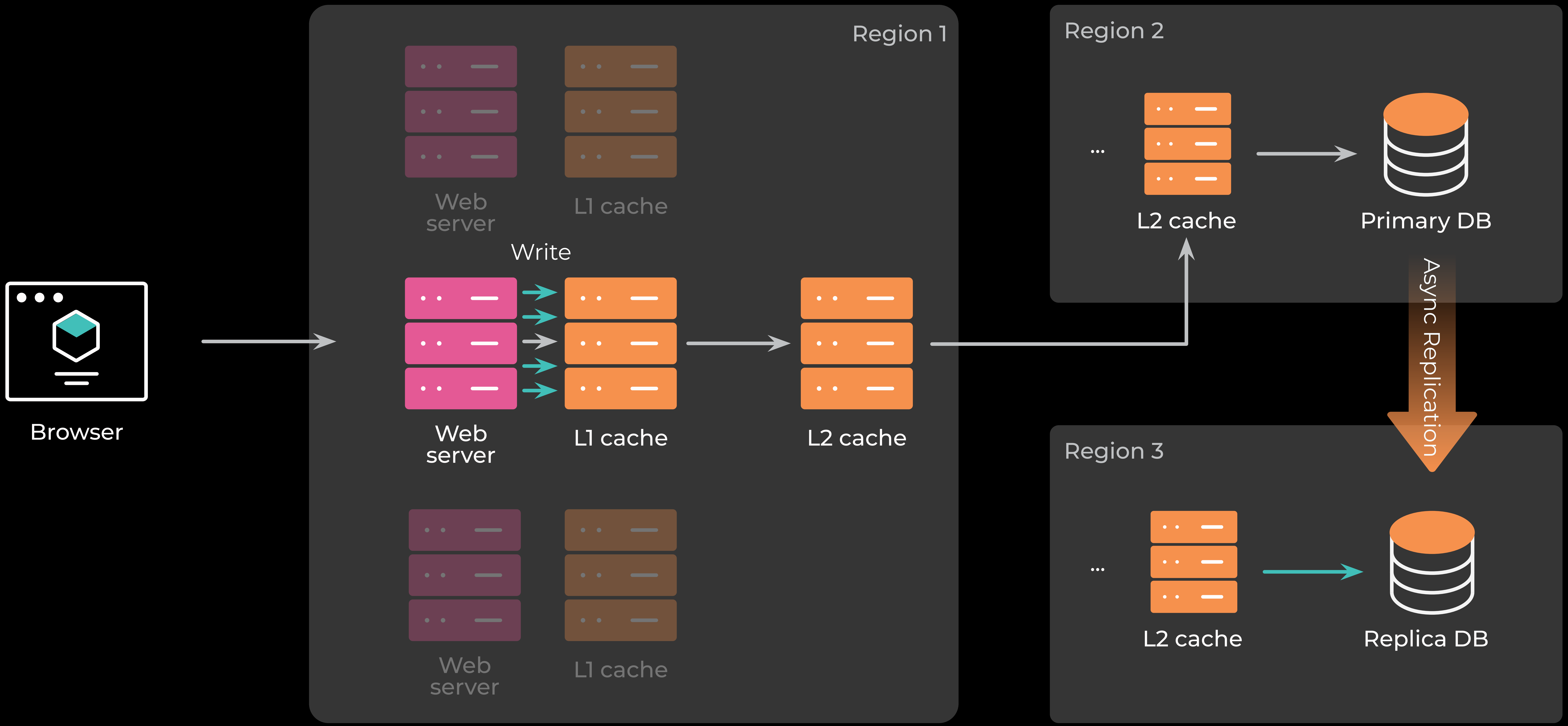


[Usenix ATC’13] TAO: Facebook’s Distributed Data Store for the Social Graph

Social graph consistency model

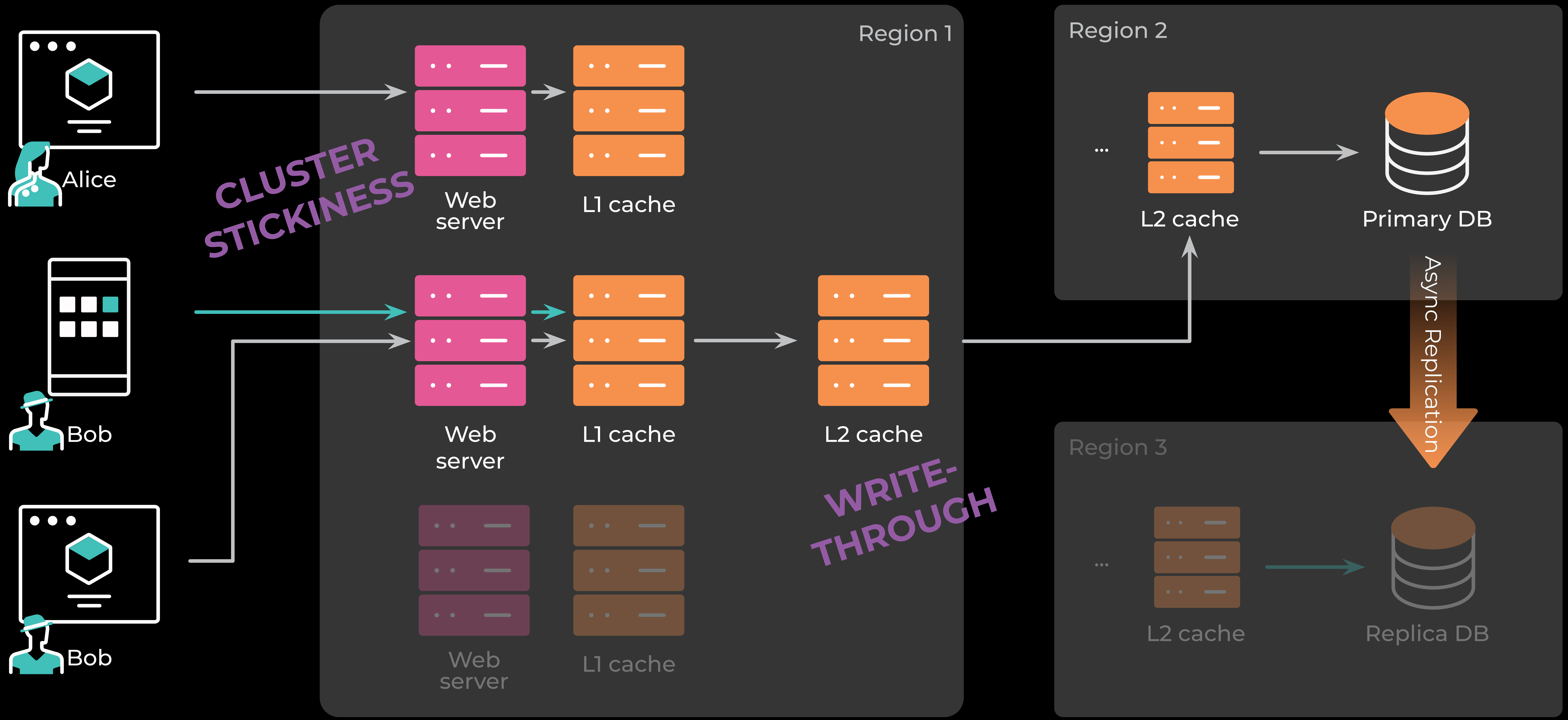
- Applications can query latest data if necessary
- Reading fresher data is OK
  i.e., per-item at-or-after / lower bound semantics
- End users get Read-Your-Writes
- Eventual consistency as a baseline

TAO 2013: two-layer write-through cache



As described in the TAO paper
[Usenix ATC'13]

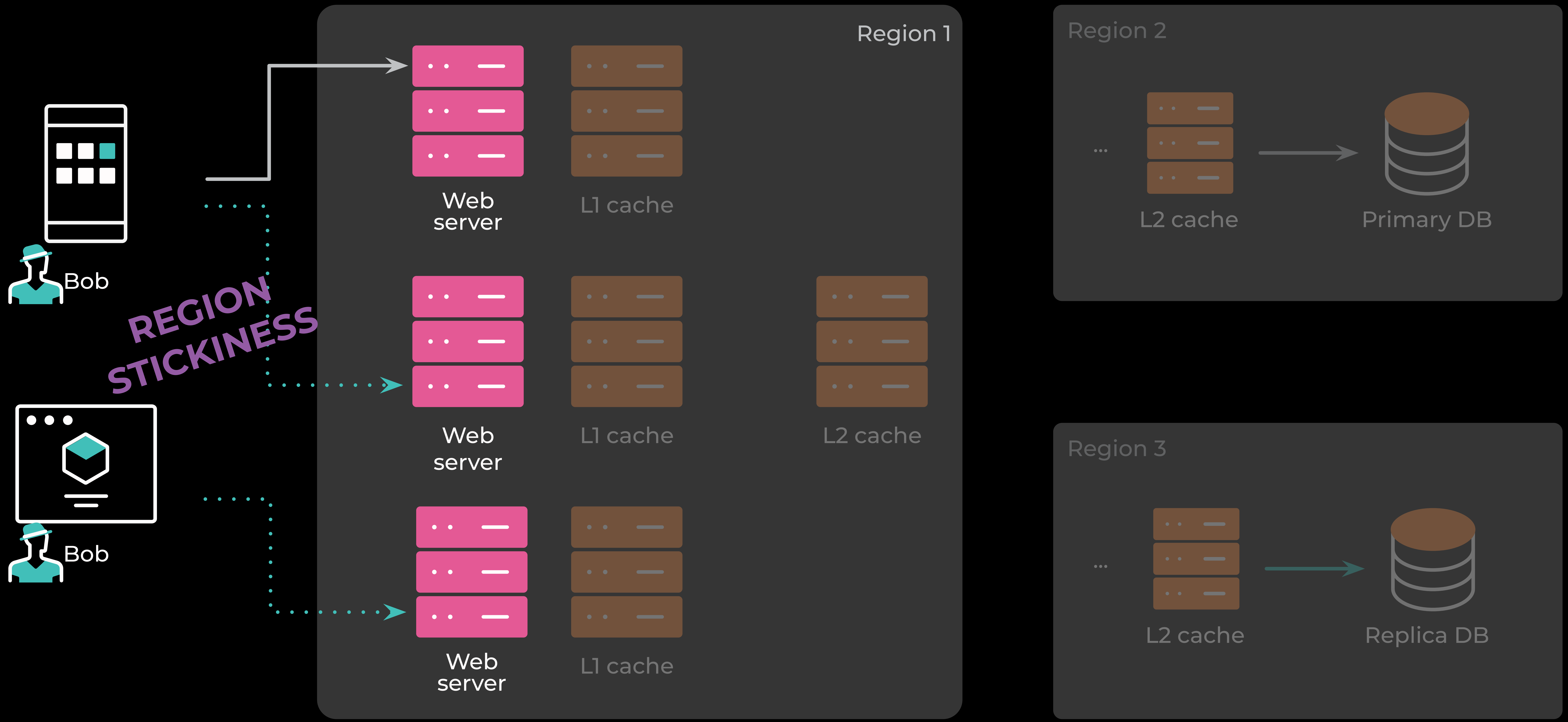
TAO 2013: fixed communication patterns for RYW



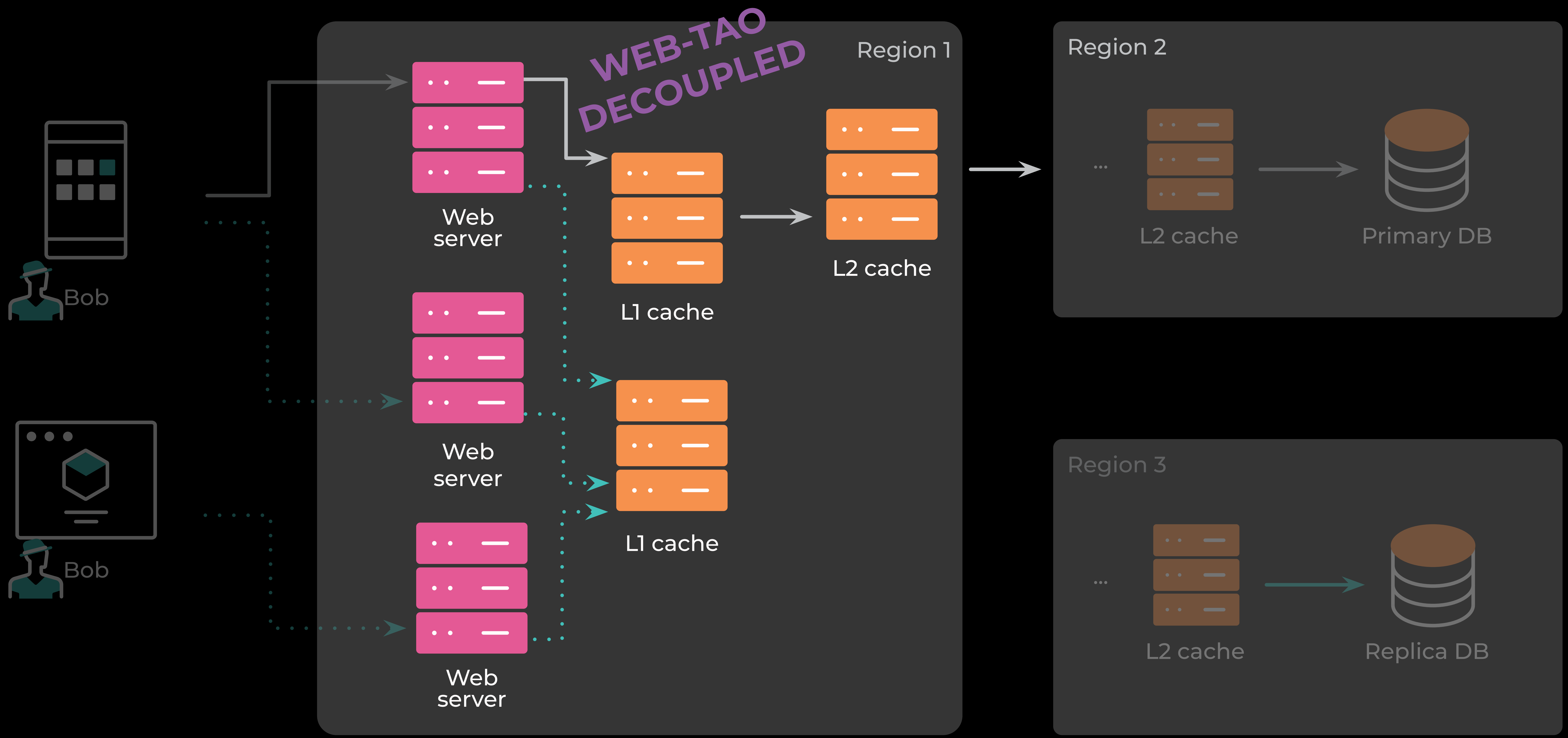
As described in the TAO paper
[Usenix ATC’13]

Evolution since 2013

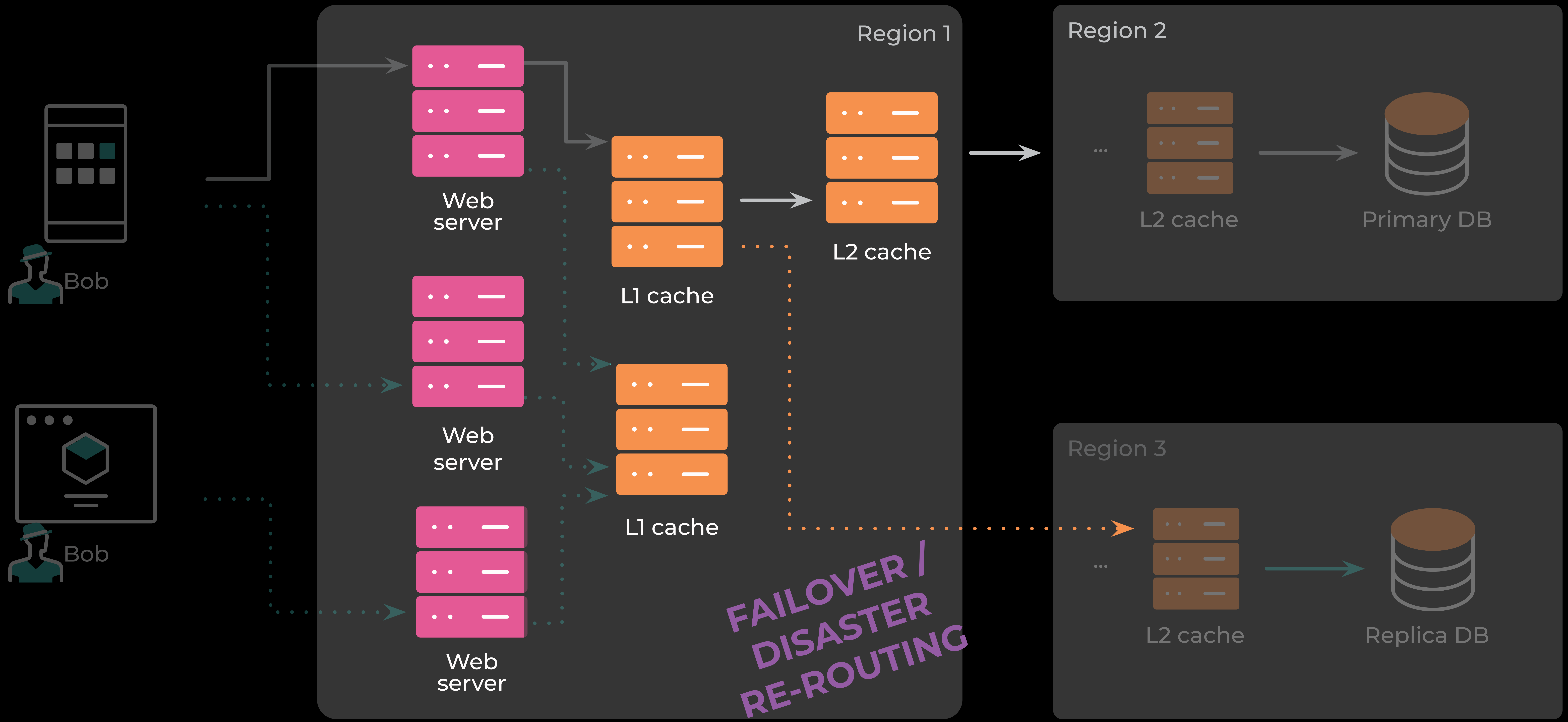
Scalability limit: routing must be more dynamic



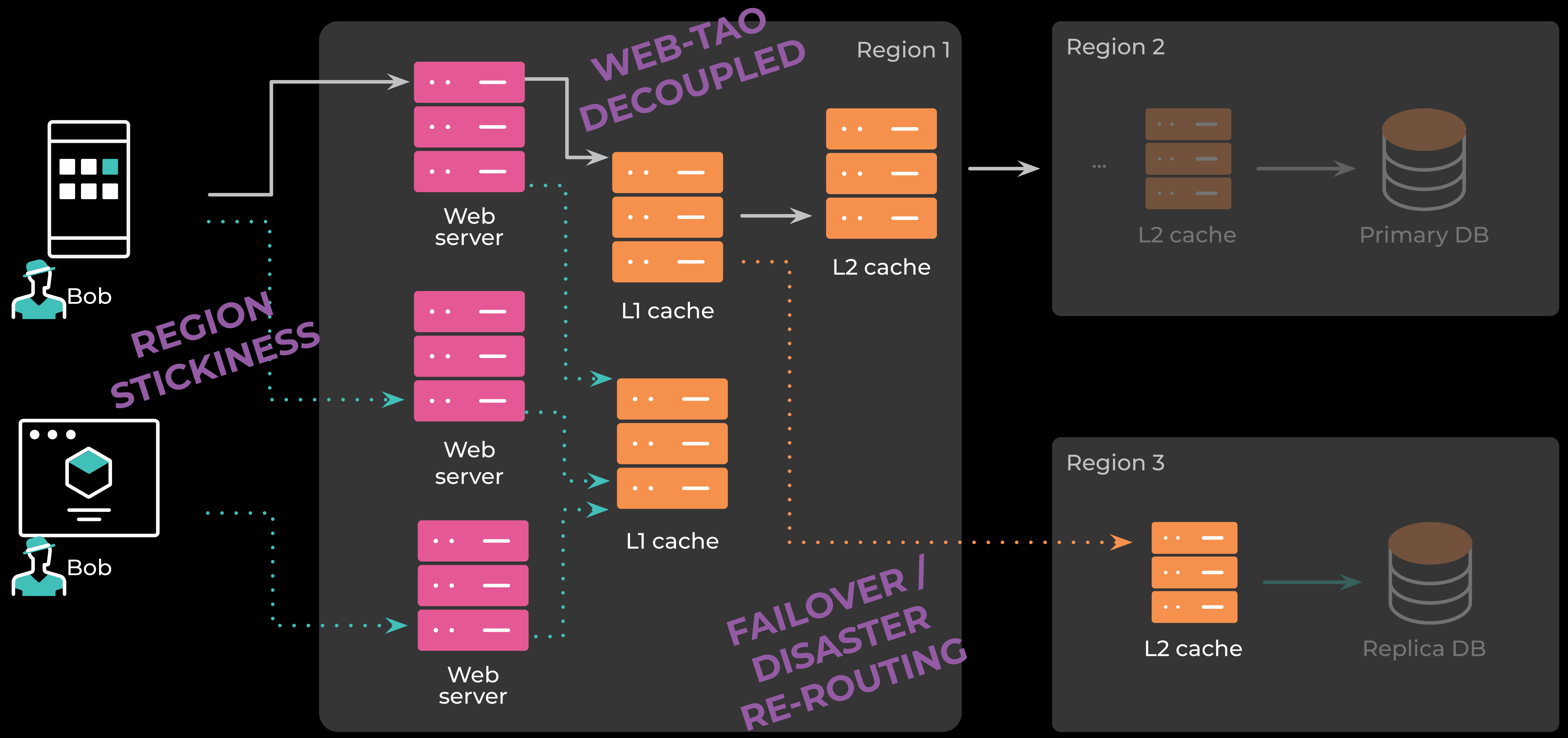
Scalability limit: routing must be more dynamic



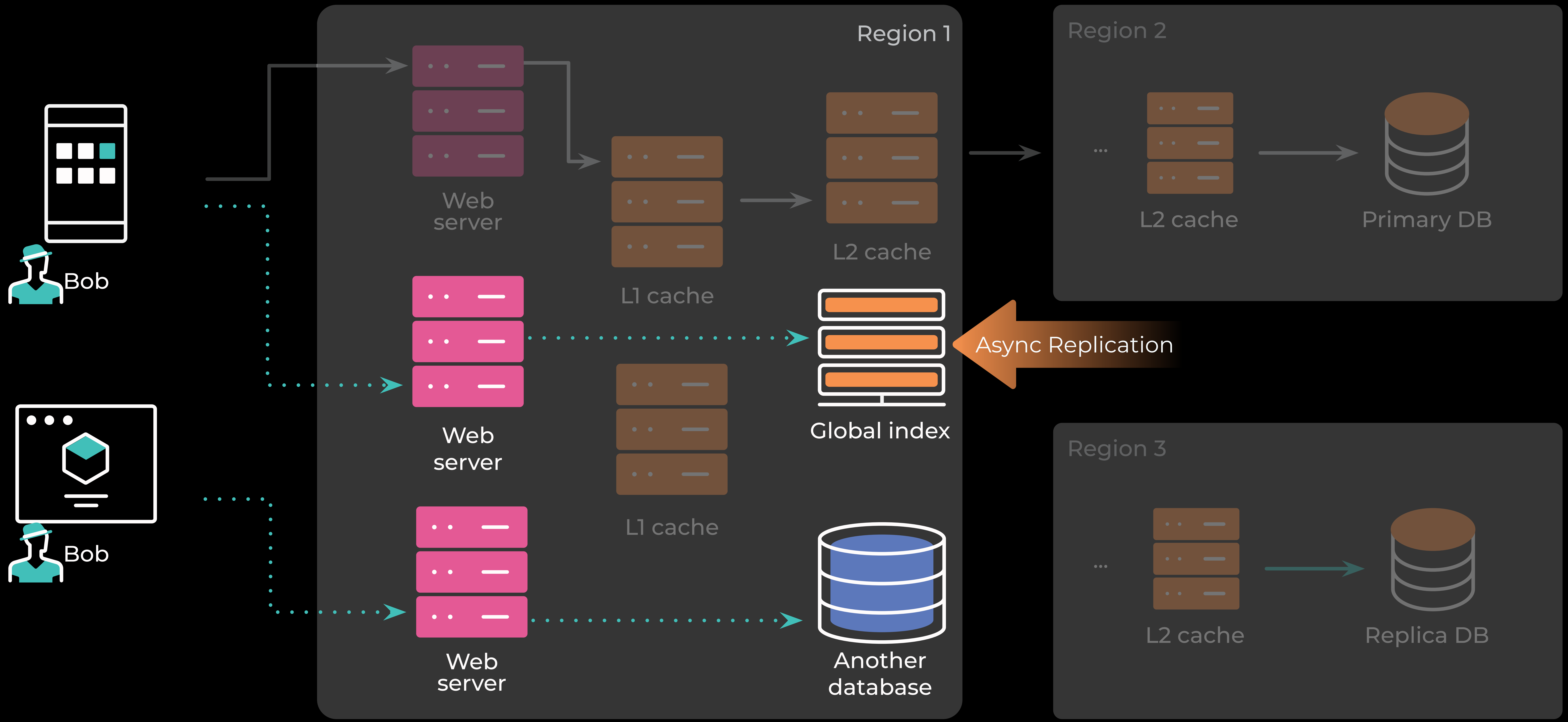
Scalability limit: routing must be more dynamic



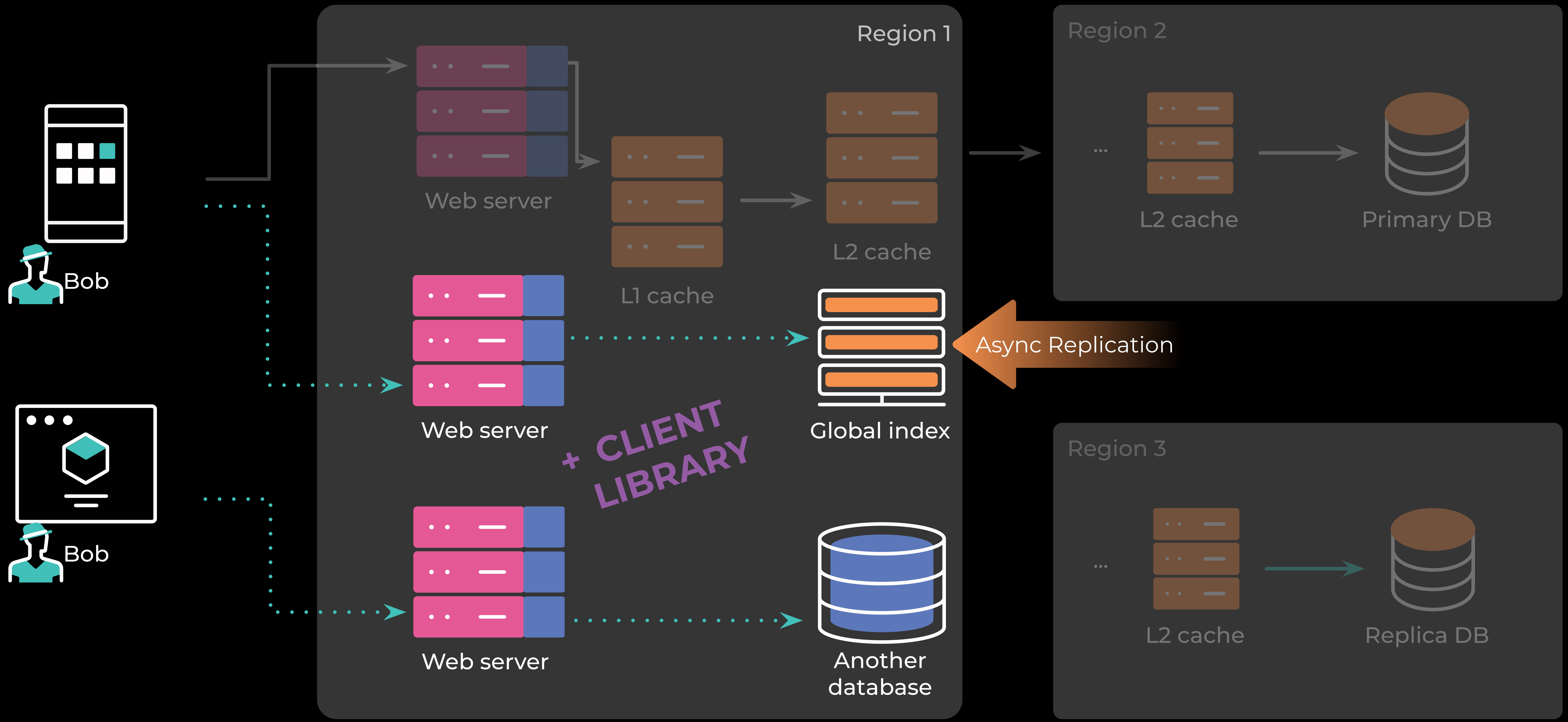
Scalability limit: routing must be more dynamic



Scalability: add global indexes and other data stores



Scalability: add global indexes and other data stores



Rethinking social graph consistency

RETAIN

- User-centric RYW consistency
- Read efficiency and hot spot tolerance
- High availability, low latency, and loose-coupling (async-replication)

ENABLE

- Dynamic communication paths
- Extend uniform semantics to global indexes and new Database types

Our solution: FlightTracker

Our solution: decompose the consistency problem

Consistency: what writes are visible to a read?

FlightTracker
Identify missing writes

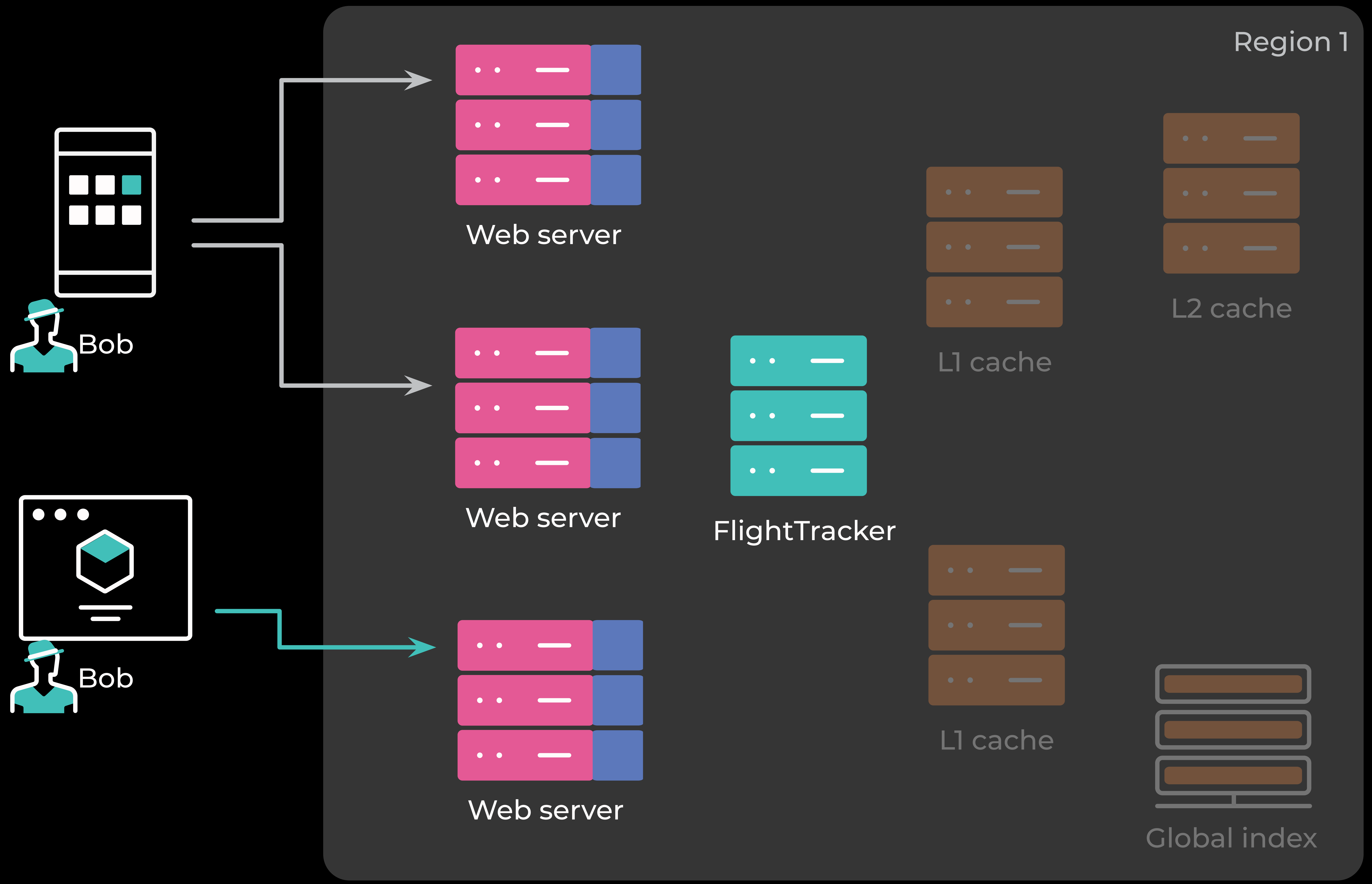
- Data-store agnostic
- Reusable and extensible
- Write metadata only

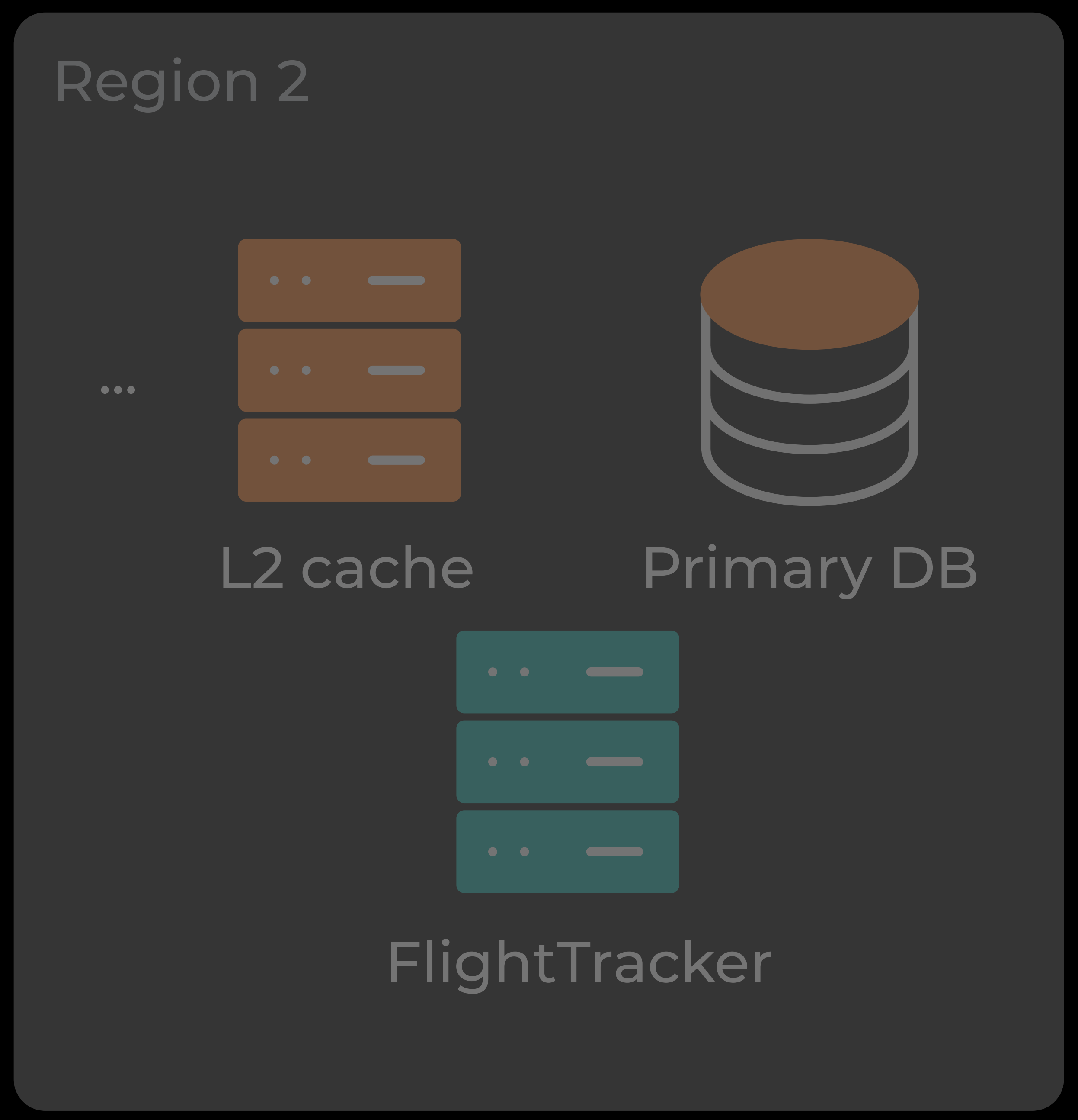
Ticket
Encapsulated set of writes

Ticket-inclusive reads
Ensure visibility: read results reflect missing writes

- Data-store specific strategies
- Ticket attached on each query

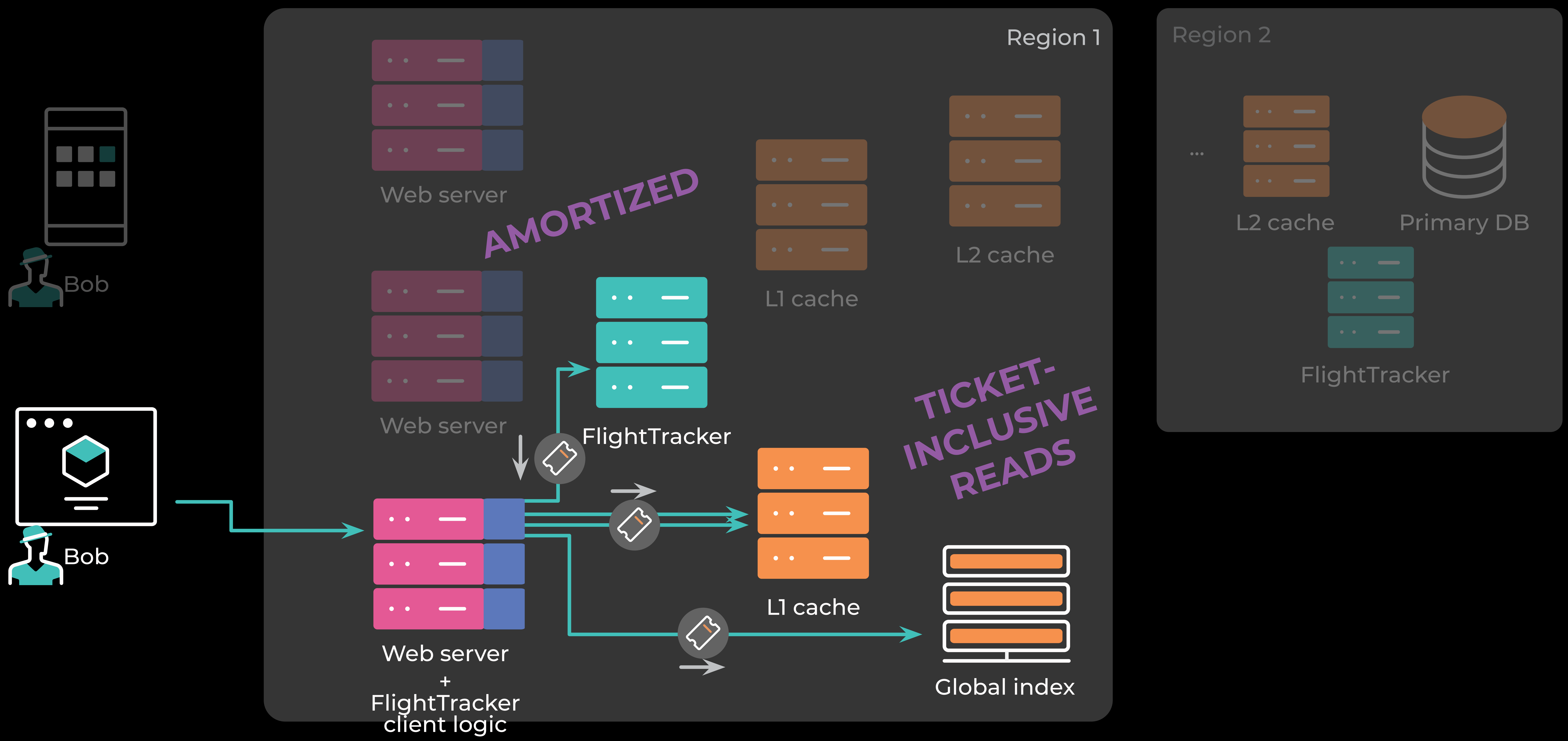
RYW: User writes span web requests



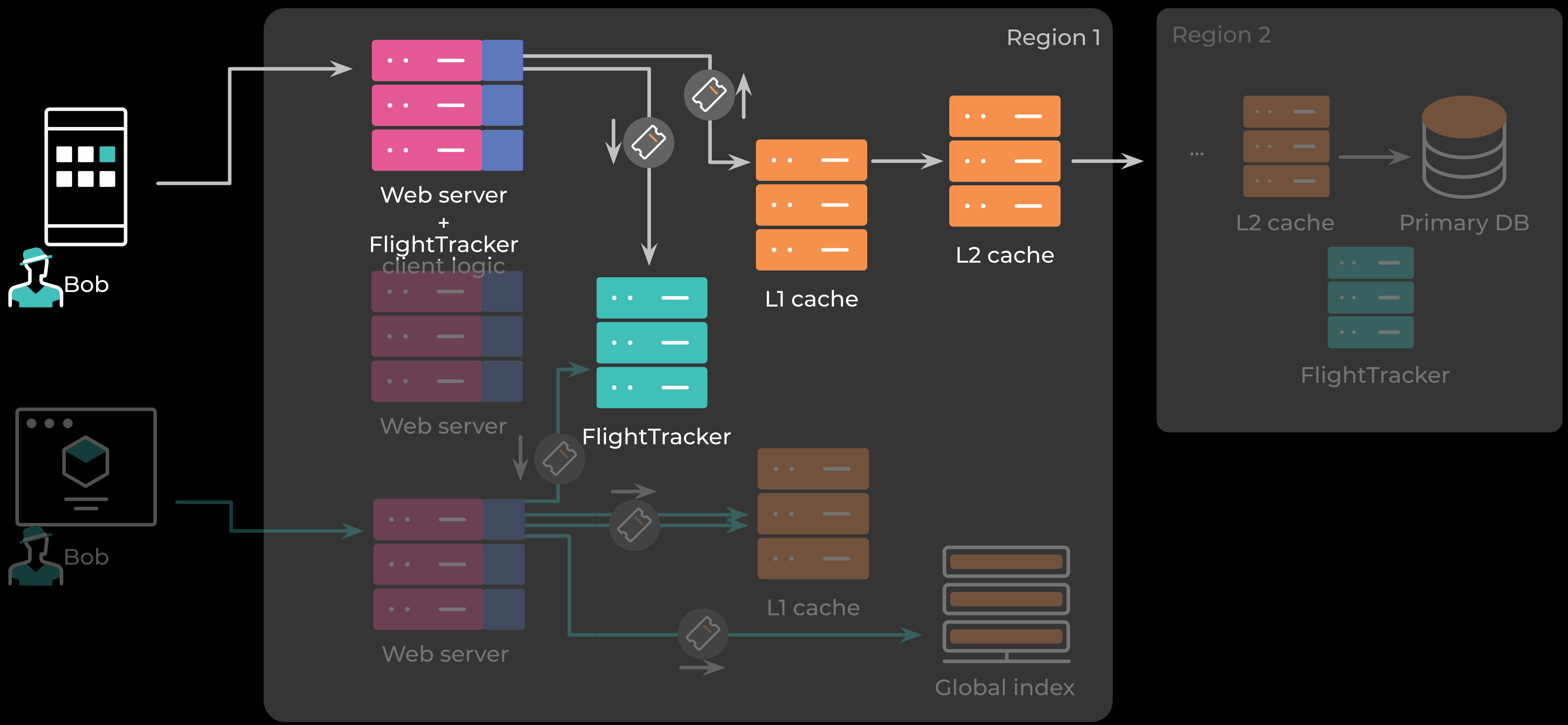


FlightTracker
maps user_ids to
recent write metadata

RYW: Read flow using FlightTracker



RYW: Write flow using FlightTracker



Ticket

- Write set: metadata that identifies a set of writes
  - Joinable, i.e., set union
- Encapsulated
  - Most code paths treat Tickets as opaque tokens
  - Serialized and compressed on the wire
- Named “Ticket” (vs. timestamp / version) to reduce potential preconception about its semantics

Ticket representation

```
Ticket {
  RepForDatabaseA databaseA;
  RepForDatabaseB databaseB;
  ...
  Timestamp globalTs;
}

// Example database-specific representation
RepForDatabaseA {
  map<WriteKey, pair<Version, Timestamp>> perKeyMap;
  map<ShardId, pair<TxnId, Timestamp>> perShardMap;
}
```

```
{
  databaseA: {
    “node123”:
      {v: 2, ts: 1603237337483},
    “edge456”:
      {v: 42, ts: 1603237338021}
  }
}
```

Ticket-inclusive read

Data-store specific implementation strategies

1

Fix data store first

e.g., consistency miss for caches

2

Fix stale results

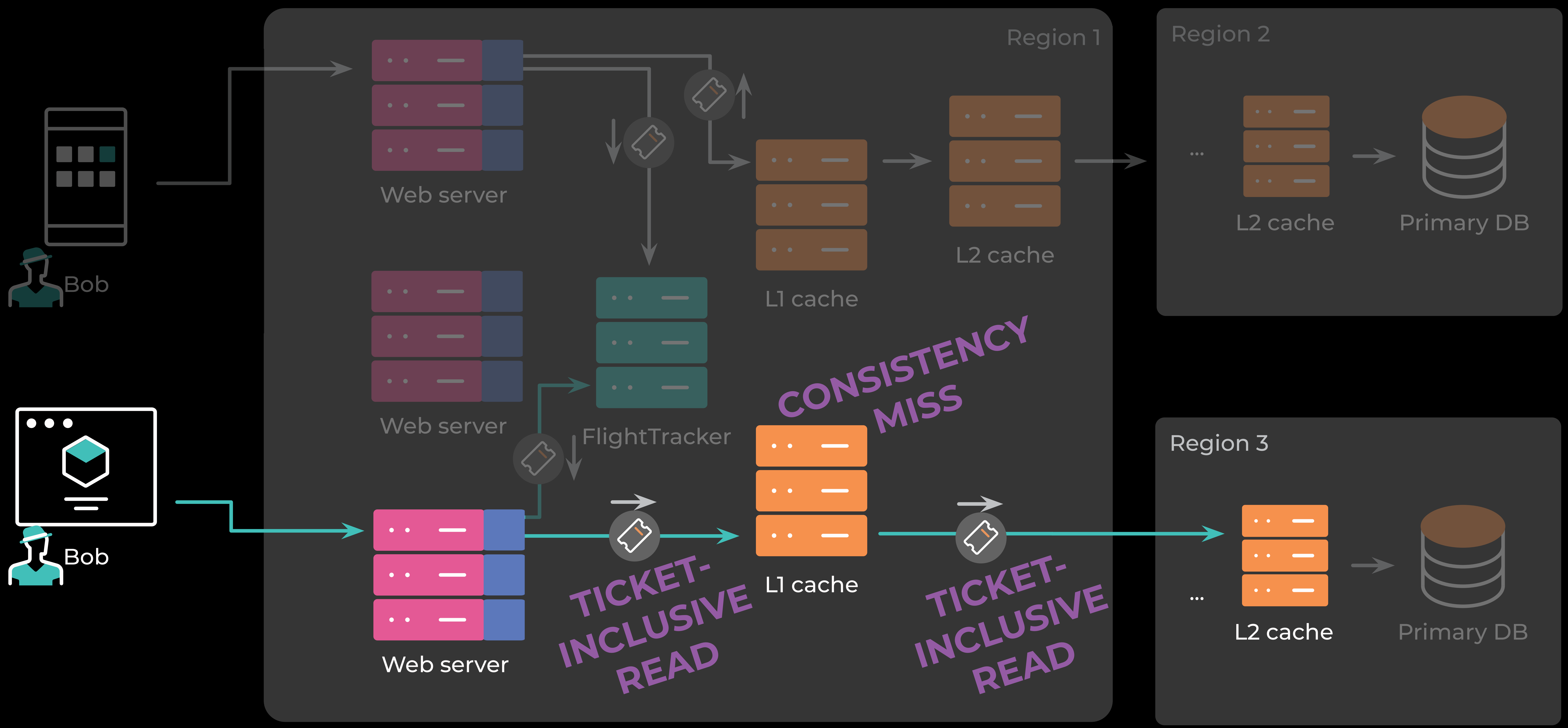
e.g., client read repair for indexes

3

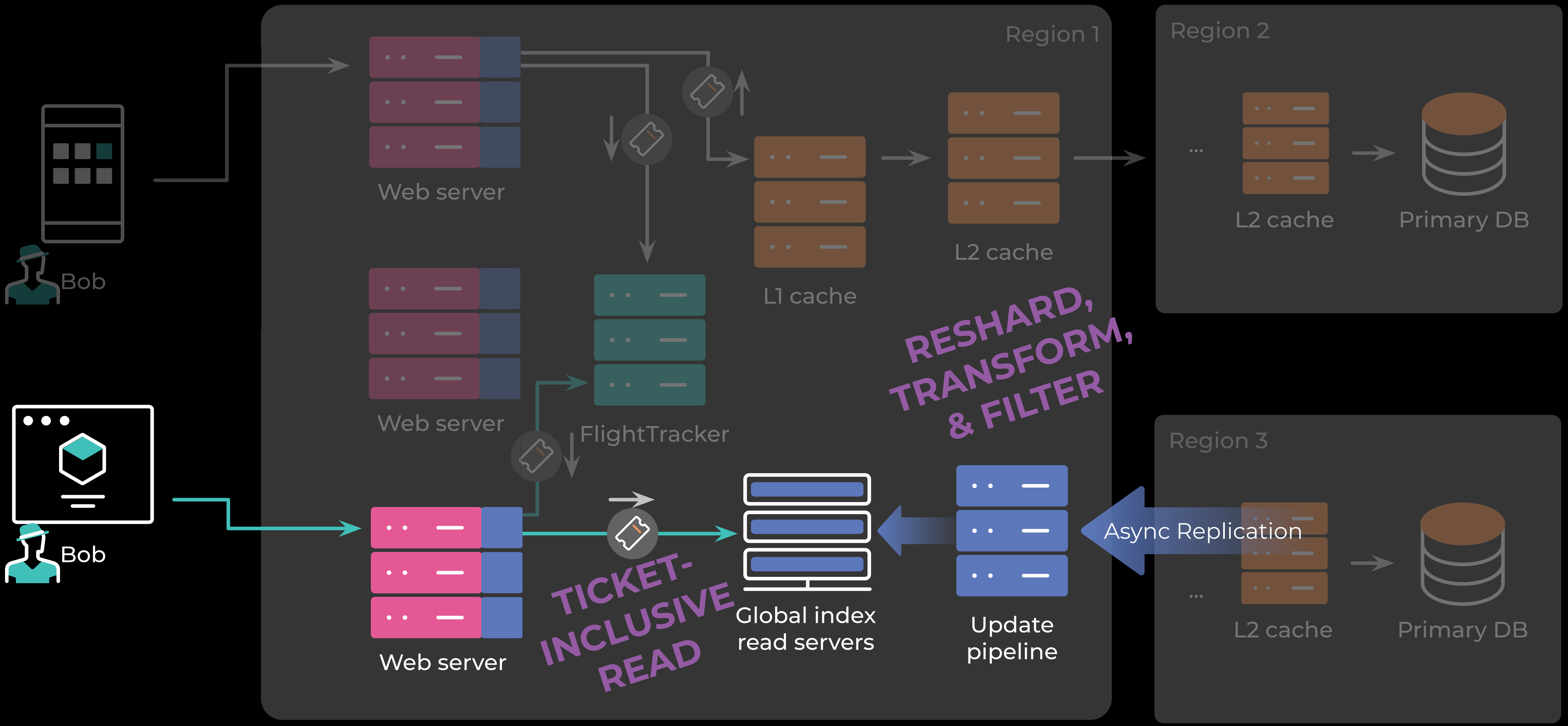
Reevaluate query

e.g., on a diff replica; at a later time

Ticket-inclusive read for caches



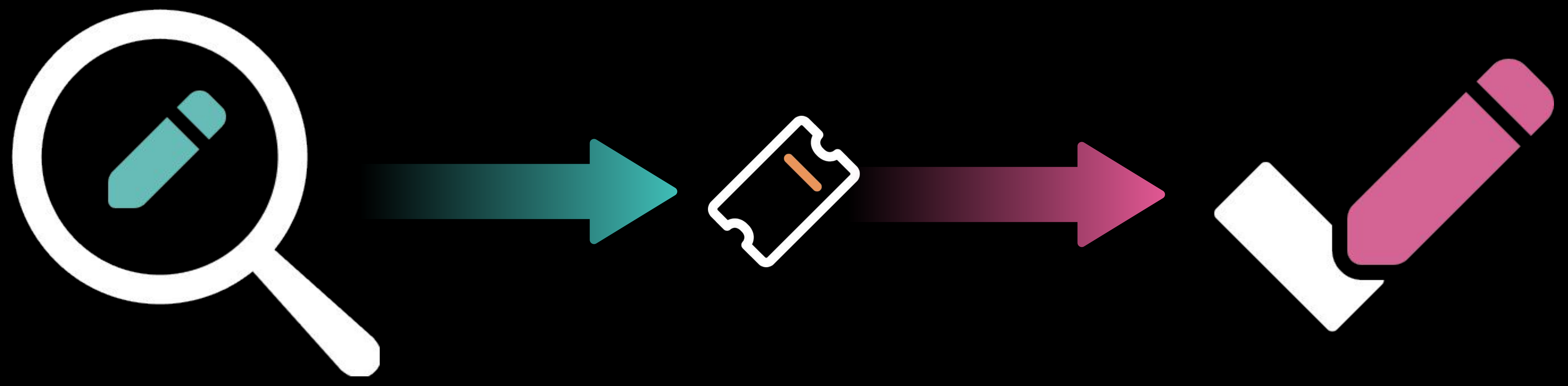
Challenges for global indexes



Beyond RYW

Beyond RYW: additional FlightTracker session types

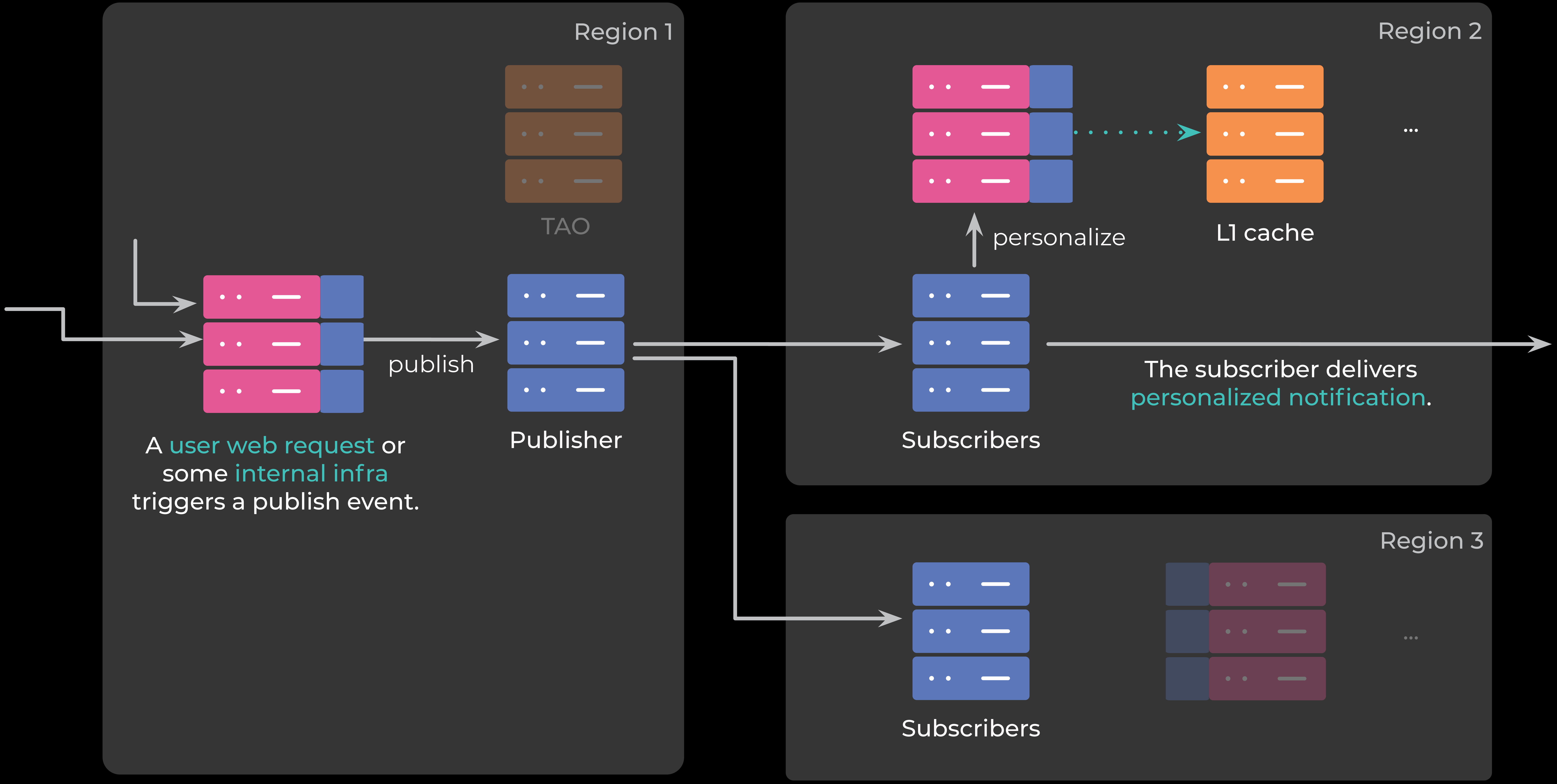
- The default session is an end user, which is sticky to a region.
- Select applications need write visibility guarantees other than user-centric RYW.
- Flexible definition of “session”
  - E.g., async job, particular TAO object (see paper)
  - Reads and writes can belong to multiple sessions.
- Customizable FlightTracker quorum config
  - E.g., write to FlightTracker in all regions, read locally



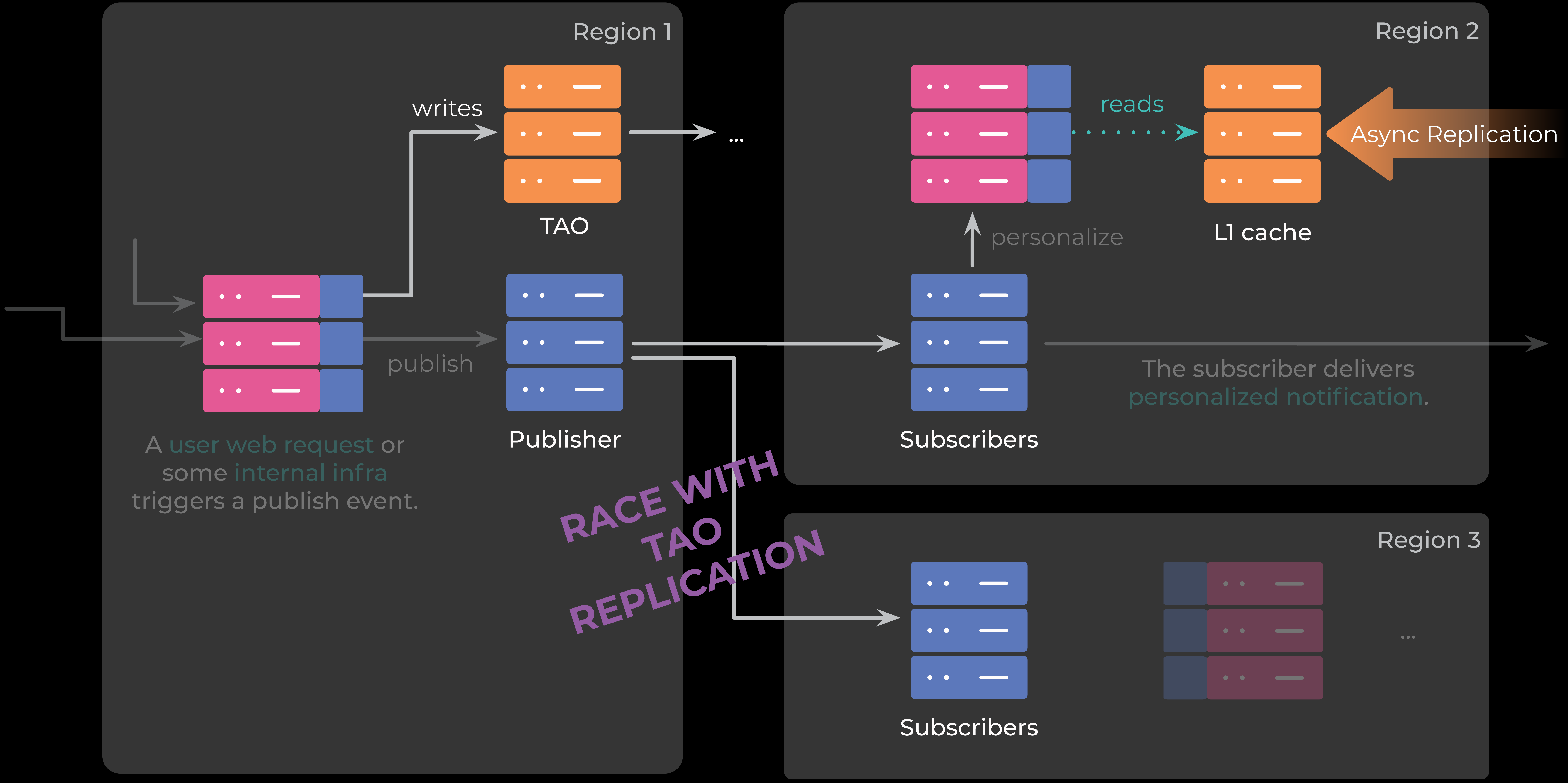
Beyond RYW: external Ticket handling

- Systems at the product infrastructure layer may handle Tickets explicitly
  - Especially when we can piggyback on existing communication
  - Still hidden from applications

Example: pub-sub notification system

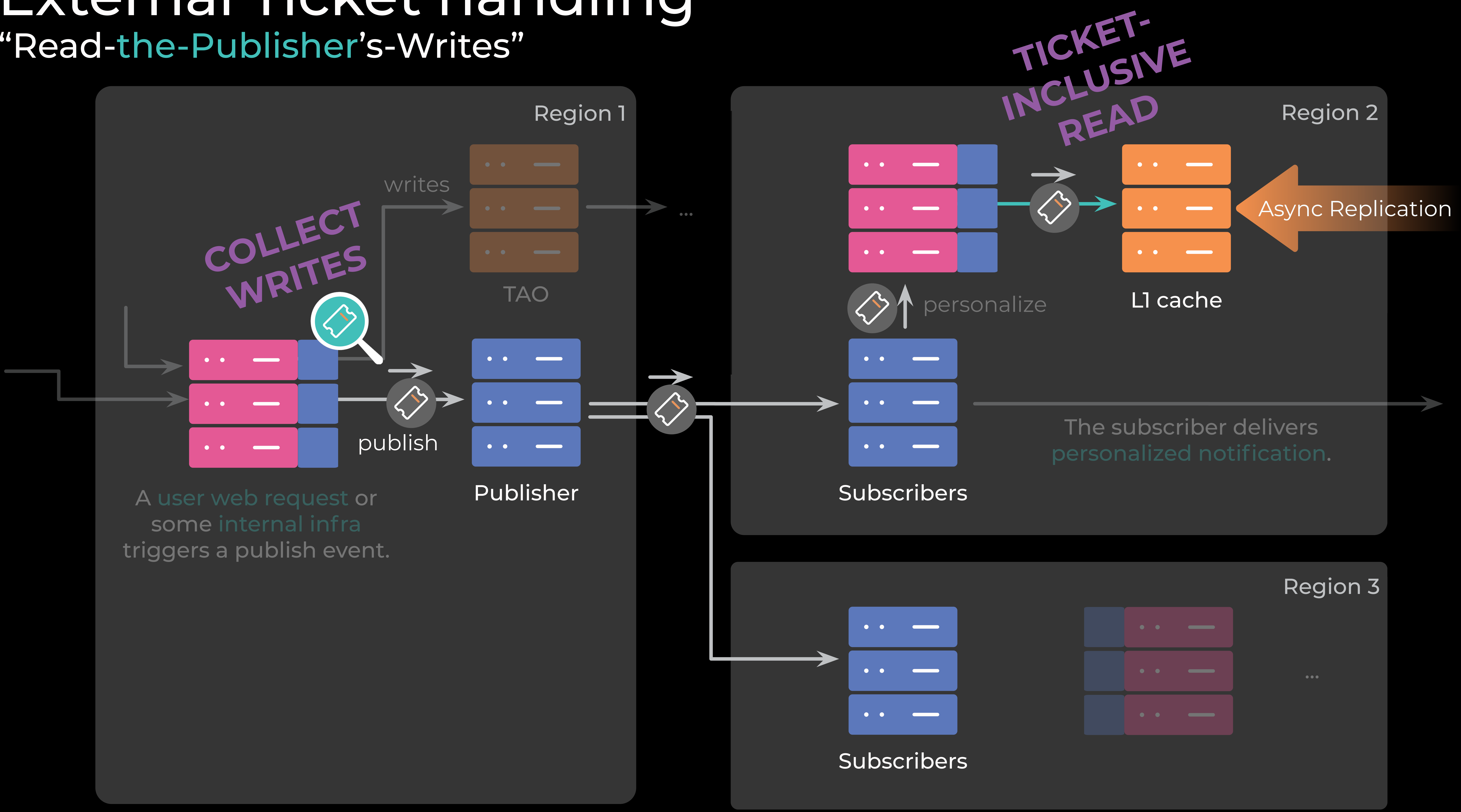


Pub-sub notification system: the problem



External Ticket handling

"Read-the-Publisher's-Writes"



Lessons learned
&
production experiences

Ticket internals are encapsulated from applications.

+

Ticket-inclusive reads only targets per-item at-or-after / lower-bound semantics.

Can safely include additional write metadata while honoring RYW.

e.g., FlightTracker server or client are free to join Tickets whenever new writes happen.

Can safely include
additional write metadata
while honoring RYW.

e.g., joining Tickets

Ticket compaction

e.g., can replace write metadata
with a single global timestamp
for writes older than 60s

Can safely include
additional write metadata
while honoring RYW.

e.g., joining Tickets

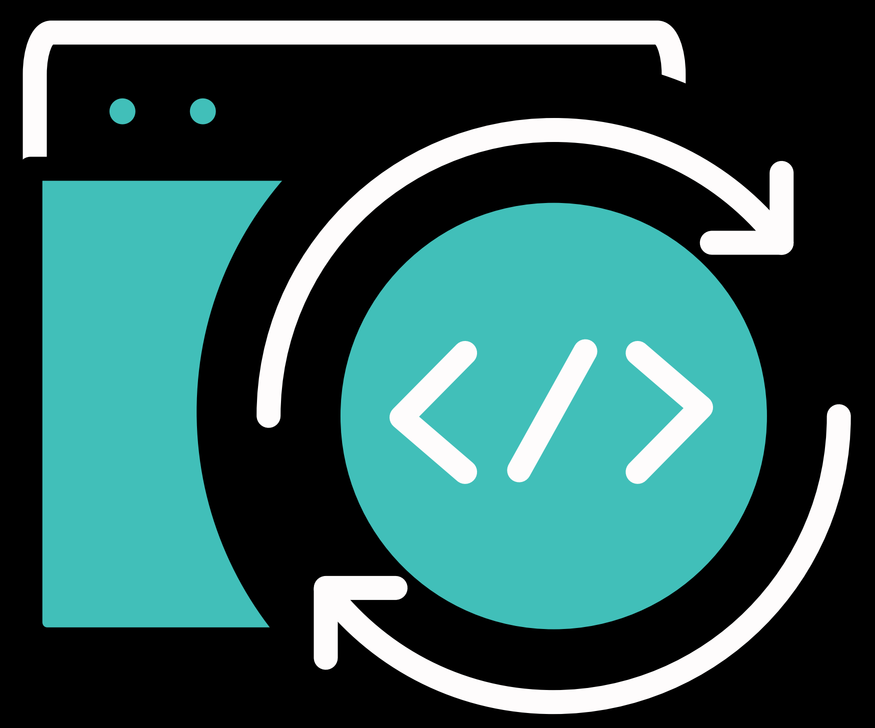
Single-round protocol
for FlightTracker

Only need to provide **durability**
but NOT **atomicity**

Lessons learned (Cont'd)

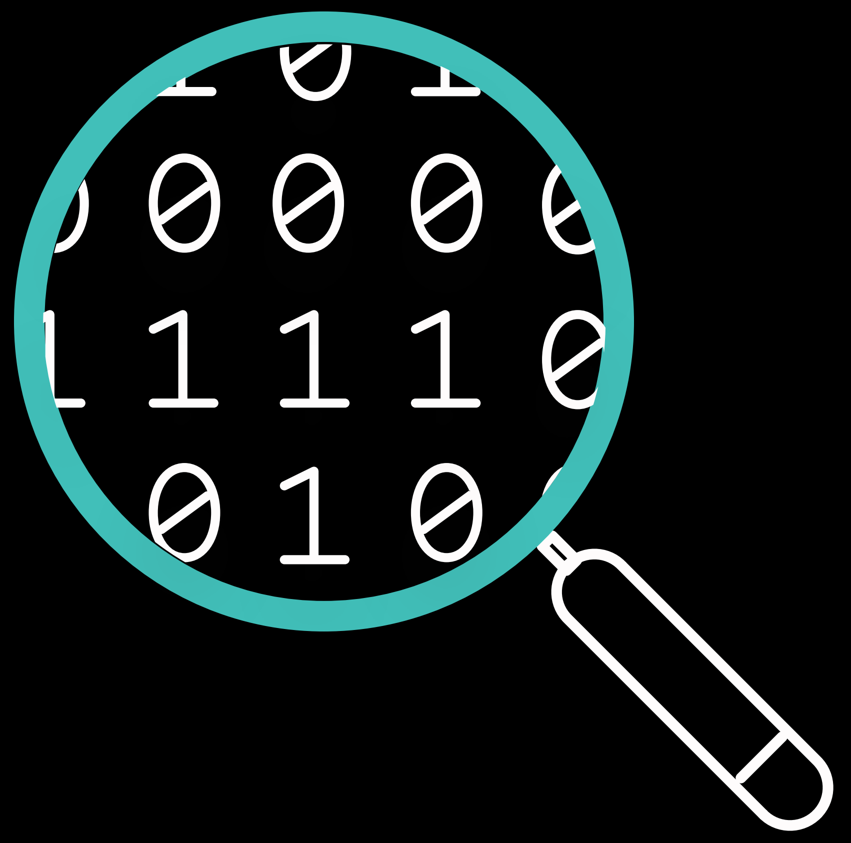
Identifying logged-in user_id was more difficult than we expected.

Constraints on FlightTracker design are not based on the average case, but the extreme ones, such as hot spots or disaster scenarios.

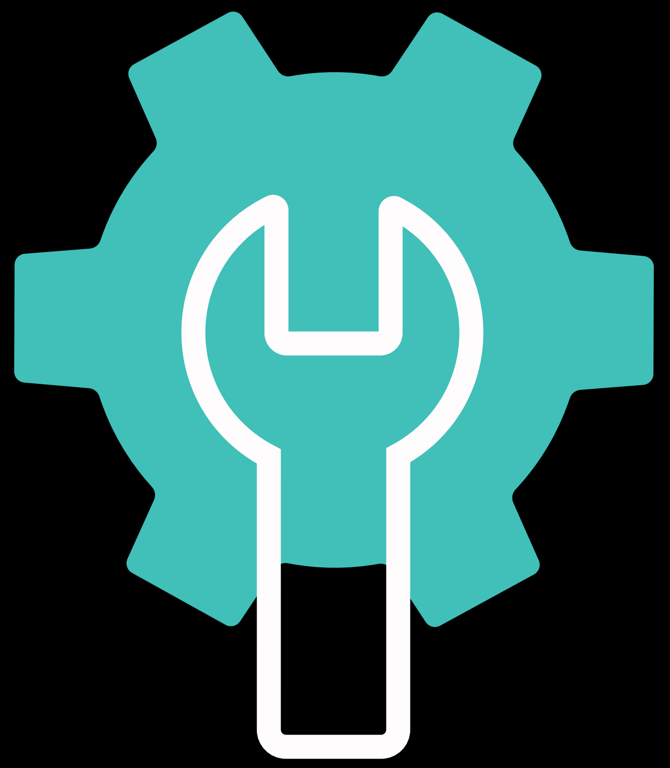


The ability to opt into alternative write visibility guarantees late in product dev cycle enabled us to make RYW a good default.

Lessons learned (Cont'd)



Ticket-inclusive reads established a contract that revealed latent bugs in our existing eventual consistency protocols.



The applications that cause the most operational trouble often need RYW the least.

The decomposition in the FlightTracker design allowed us to incrementally provide RYW for 2 caches, 3 global indexes, and 2 database technologies.

It’s real and it works

99.9999%

FlightTracker read availability
measured from the client

10X

FlightTracker write availability
compared with underlying data stores

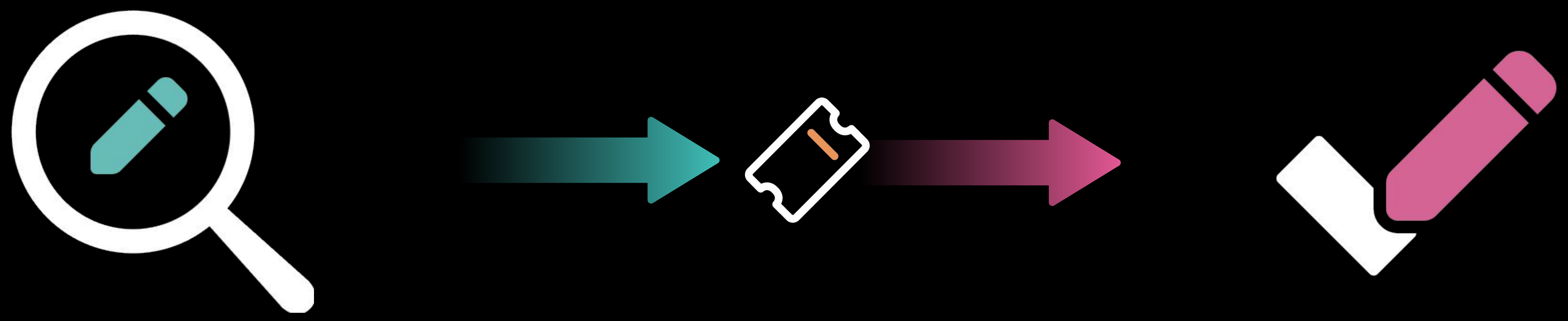
<2%

CPU/RAM overhead
on existing data stores and web servers

It’s real and it works

4 Yrs	20M	100M	10^{15}
In production	FlightTracker write QPS	FlightTracker read QPS	Social graph queries per day

Thank you!



FlightTracker: Consistency across Read-Optimized Online Stores at Facebook

Xiao Shi
xshi@fb.com

Scott Pruett

Kevin Doherty

Jinyu Han

Dmitri Petrov

Jim Carrig

John Hugg

Nathan Bronson