Transactional Consistency and Automatic Management in an Application Data Cache

Dan R. K. Ports
MIT CSAIL

joint work with
Austin Clements Irene Zhang
Samuel Madden Barbara Liskov

Applications are increasingly concurrent

- complex, scalable web applications require large-scale distributed implementations

…but programmers haven't gotten much better at dealing with concurrency

- need to ensure consistency of data in system despite race conditions

Facebook Outage, Sep. ‘10

Facebook entirely unavailable for several hours

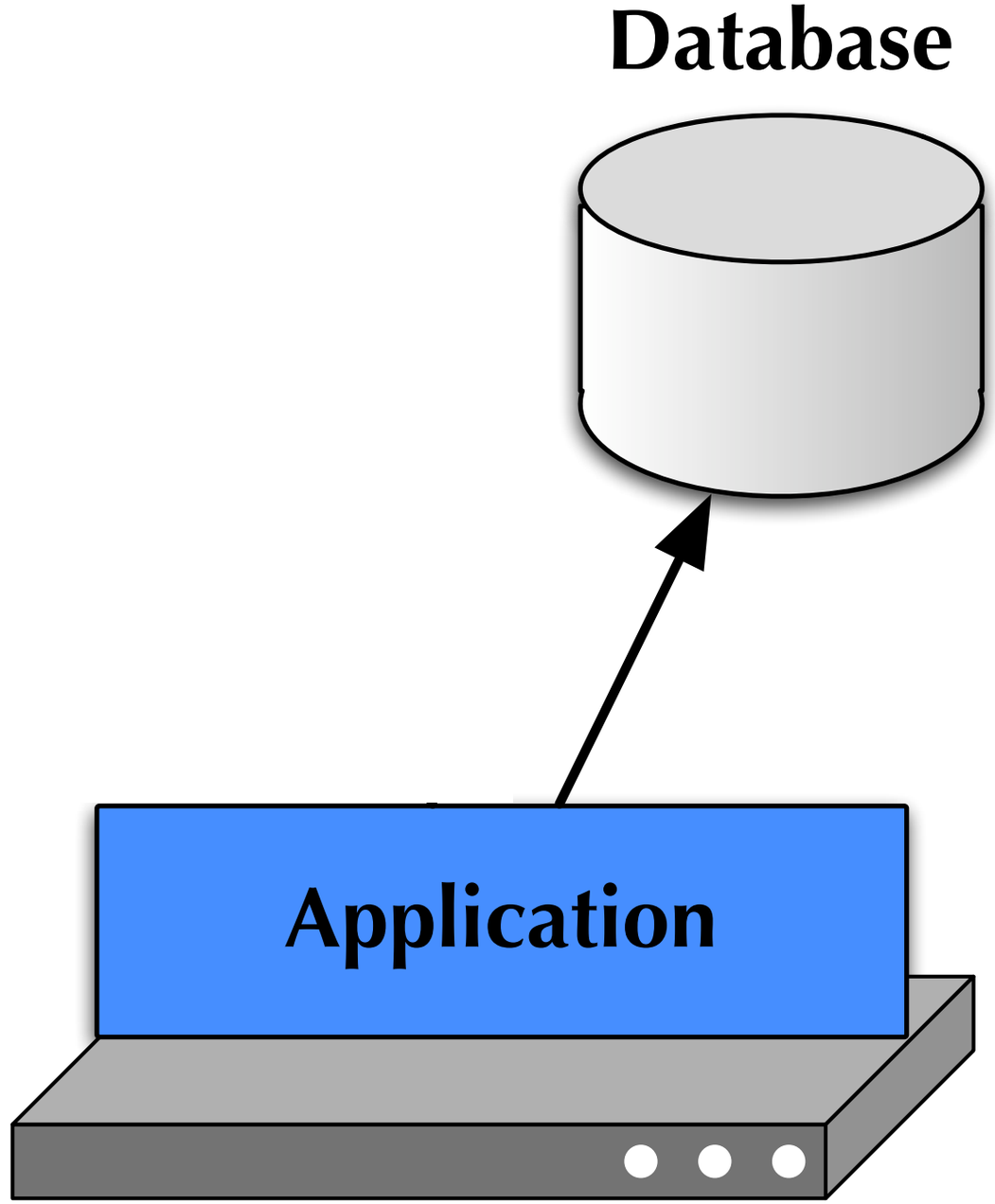
“worst outage in over four years”

Facebook Outage, Sep. ‘10

“The key flaw [...] was an unfortunate handling of an error condition. An automated system for verifying configuration values ended up causing much more damage than it fixed. The intent of the automated system is to check for configuration values that are invalid in the cache and replace them with updated values from the persistent store.”

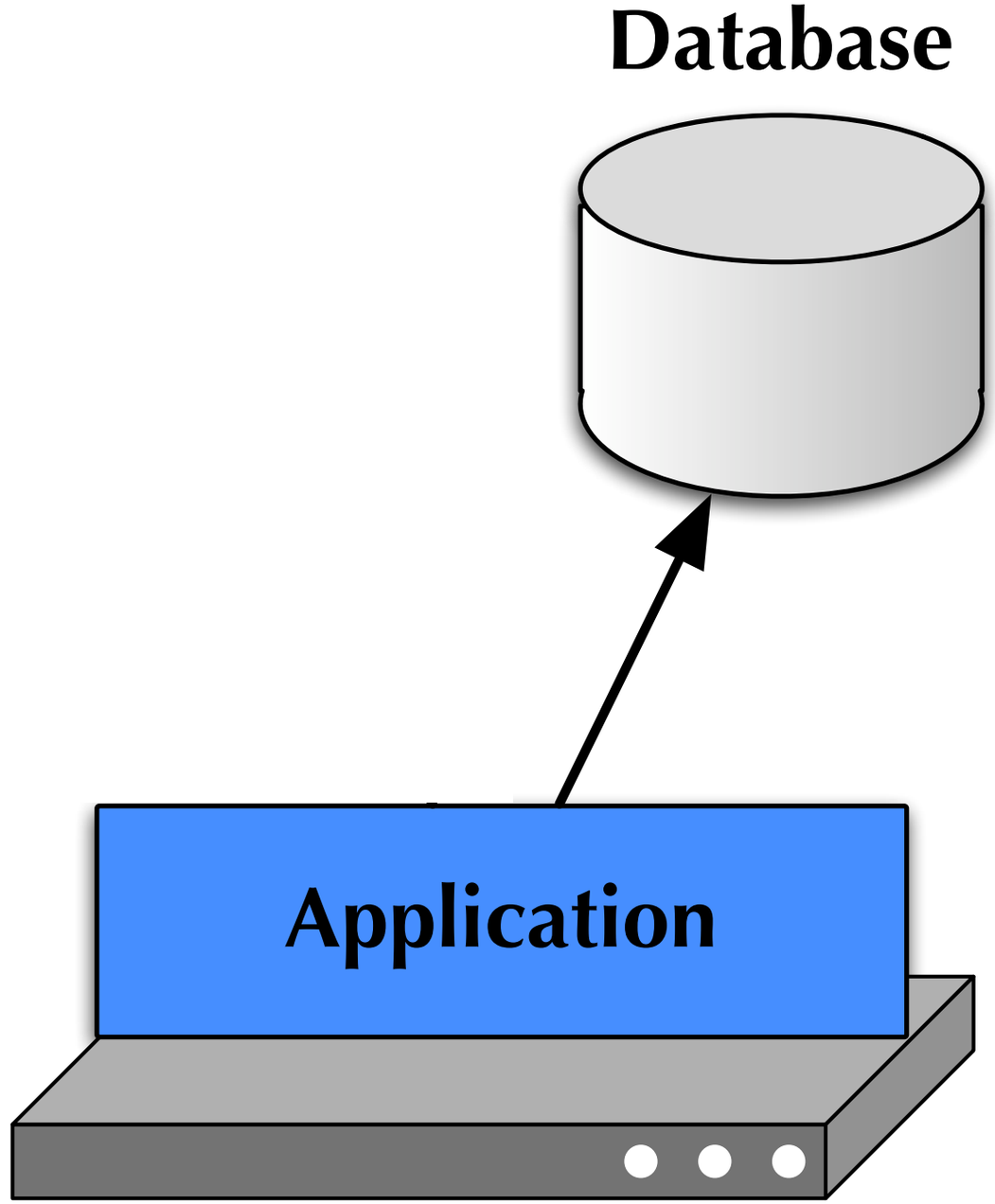
—Robert Johnson, Facebook

Application-Level Caching



Application-Level Caching

e.g. memcached,
Java object caches



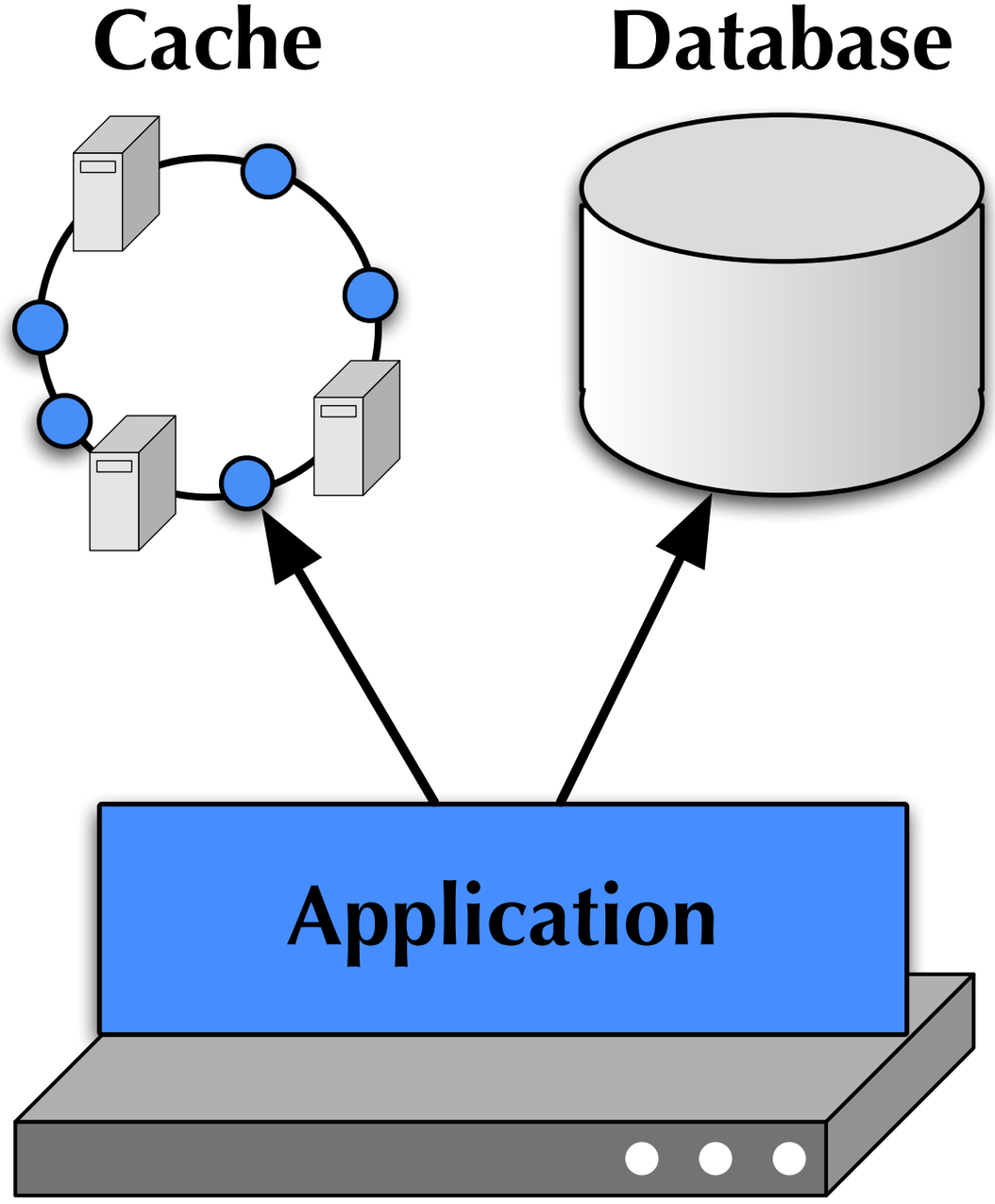
Application-Level Caching

e.g. memcached,
Java object caches

very lightweight
in-memory caches

stores ***application*** objects
(computations),
i.e.:

not a database replica
not a query cache
not a webpage cache



Existing Caches Add To Application Complexity

No transactional consistency

- violates isolation guarantees of underlying DB
- app. code must deal with transient anomalies

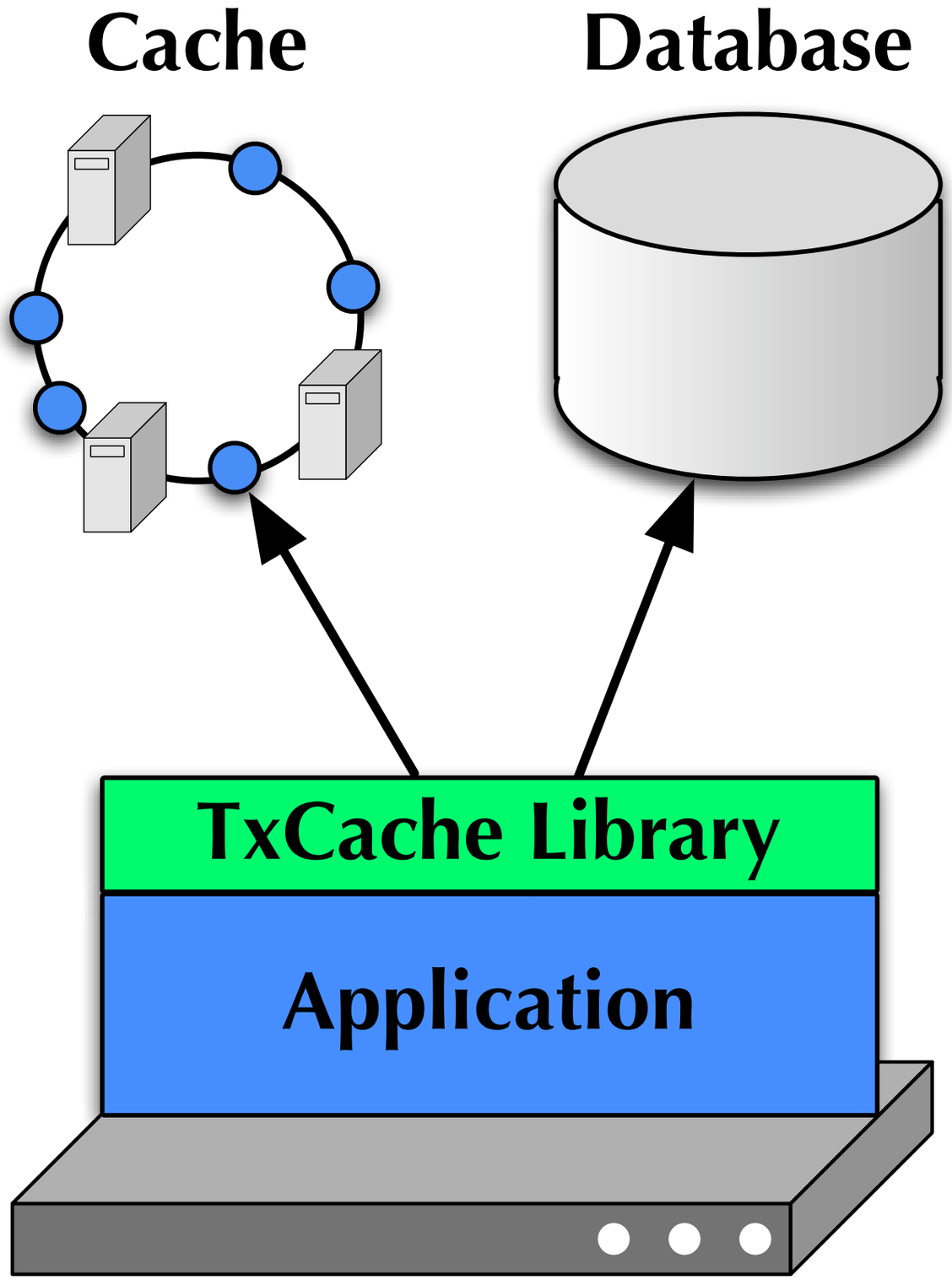
Hash table interface leaves apps responsible for:

- naming and retrieving cache entries
- keeping cache up-to-date (invalidations)

Introducing TxCache

Our cache provides:

- **transactional consistency:** serializable, point-in-time view of data, whether from cache or DB
- **bounded staleness:** improves hit rate for applications that accept old (but consistent) data
- **simpler interface:**
  applications mark functions cacheable;
  TxCache caches their results,
  including naming and invalidations



- TxCache library hides complexity of cache management
- Integrates with new cache server, minor DB modifications (Postgres; <2K lines changed)
- Together, ensure whole-system transactional consistency

TxCache Interface

- beginRO(*staleness*), commit(),
  beginRW(), abort()
- make-cacheable(*fn*)
  where *fn* is a side-effect-free function that depends only on its arguments and the database state
  - ➔ *fn* returns cached result of previous call with same inputs if still consistent w/ DB

TxCache Interface

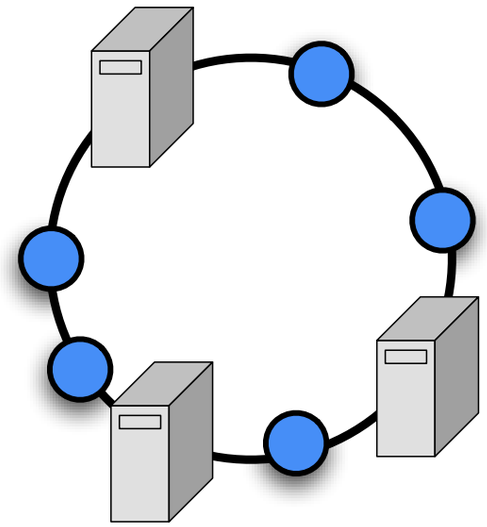
- beginRO(*staleness*), commit(),
  beginRW(), abort()
- make-cacheable(*fn*)
  where *fn* is a side-effect-free function that depends only on its arguments and the database state
  ➔ *fn* returns cached result of previous call with same inputs if still consistent w/ DB

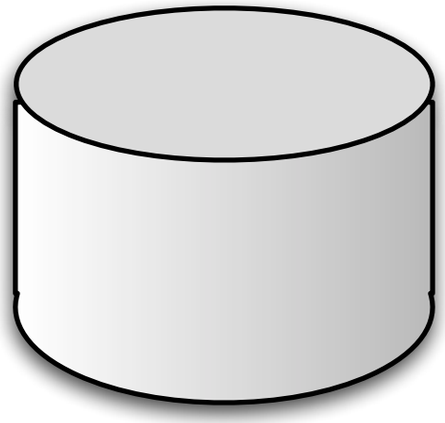
That's it.

TxCache Interface

- beginRO(*staleness*), commit(),
  beginRW(), abort()
- make-cacheable(*fn*)
  where *fn* is a side-effect-free function that depends only on its arguments and the database state
  ➔ *fn* returns cached result of previous call with same inputs if still consistent w/ DB

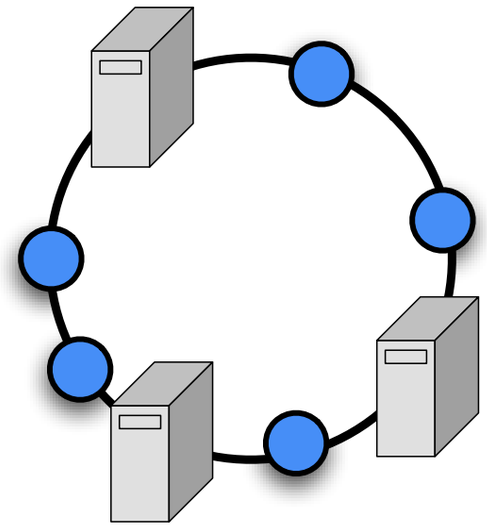
That's it.
Really!

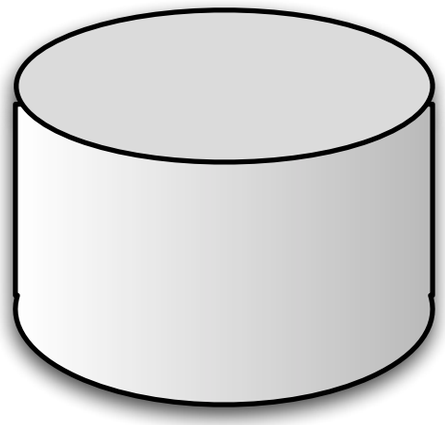


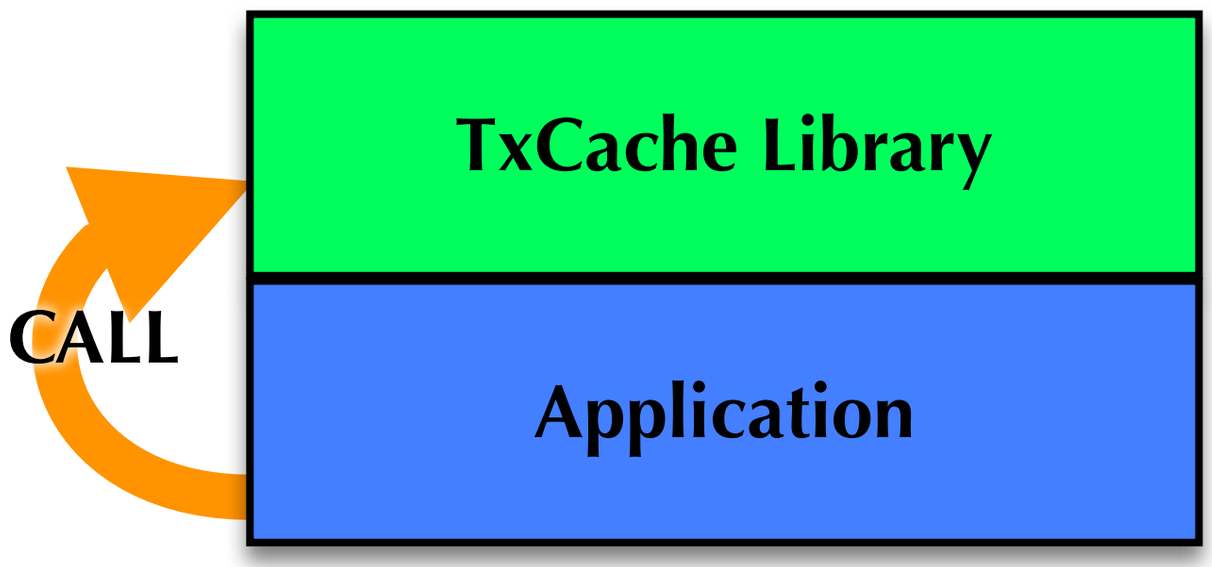


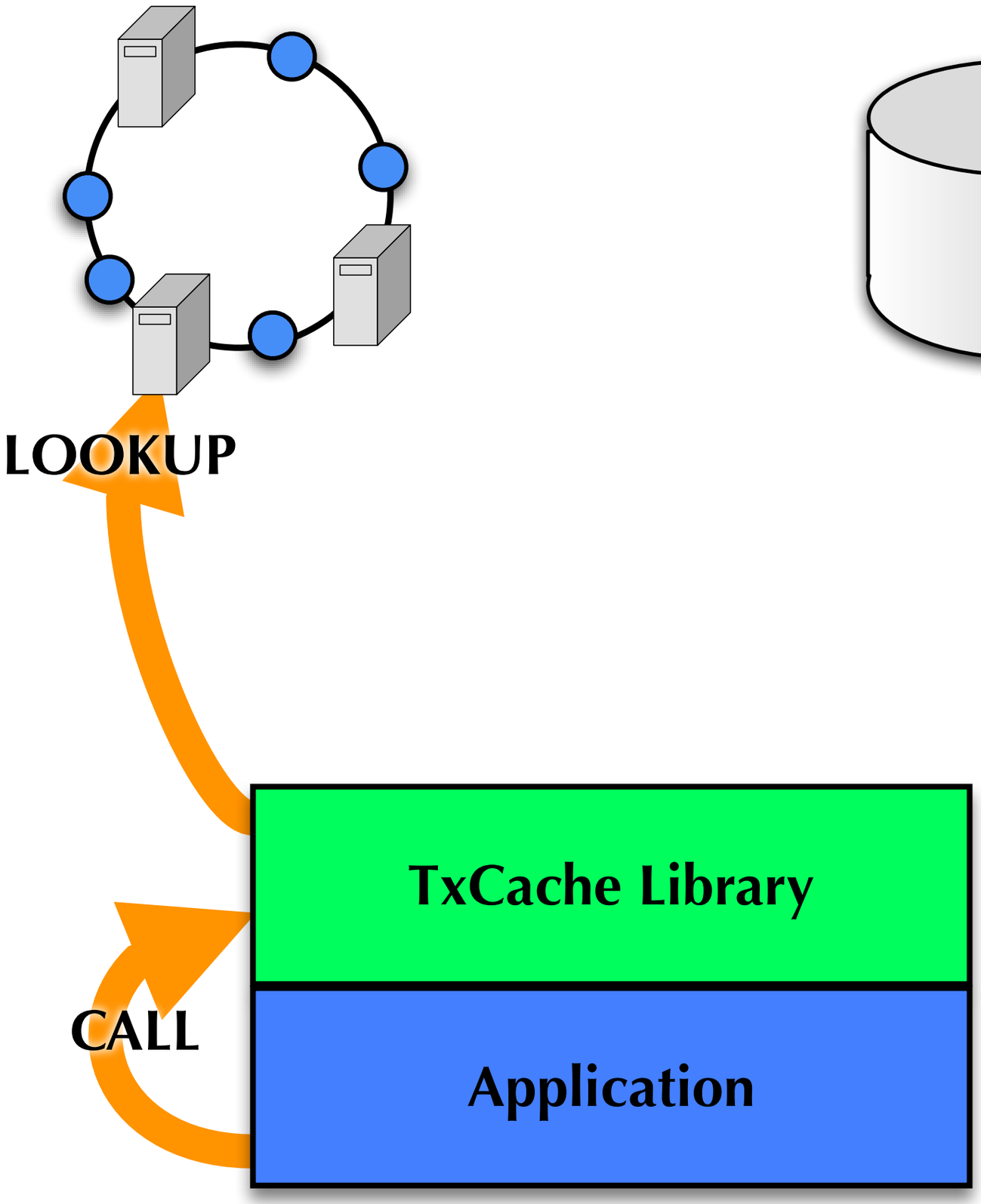
TxCache Library

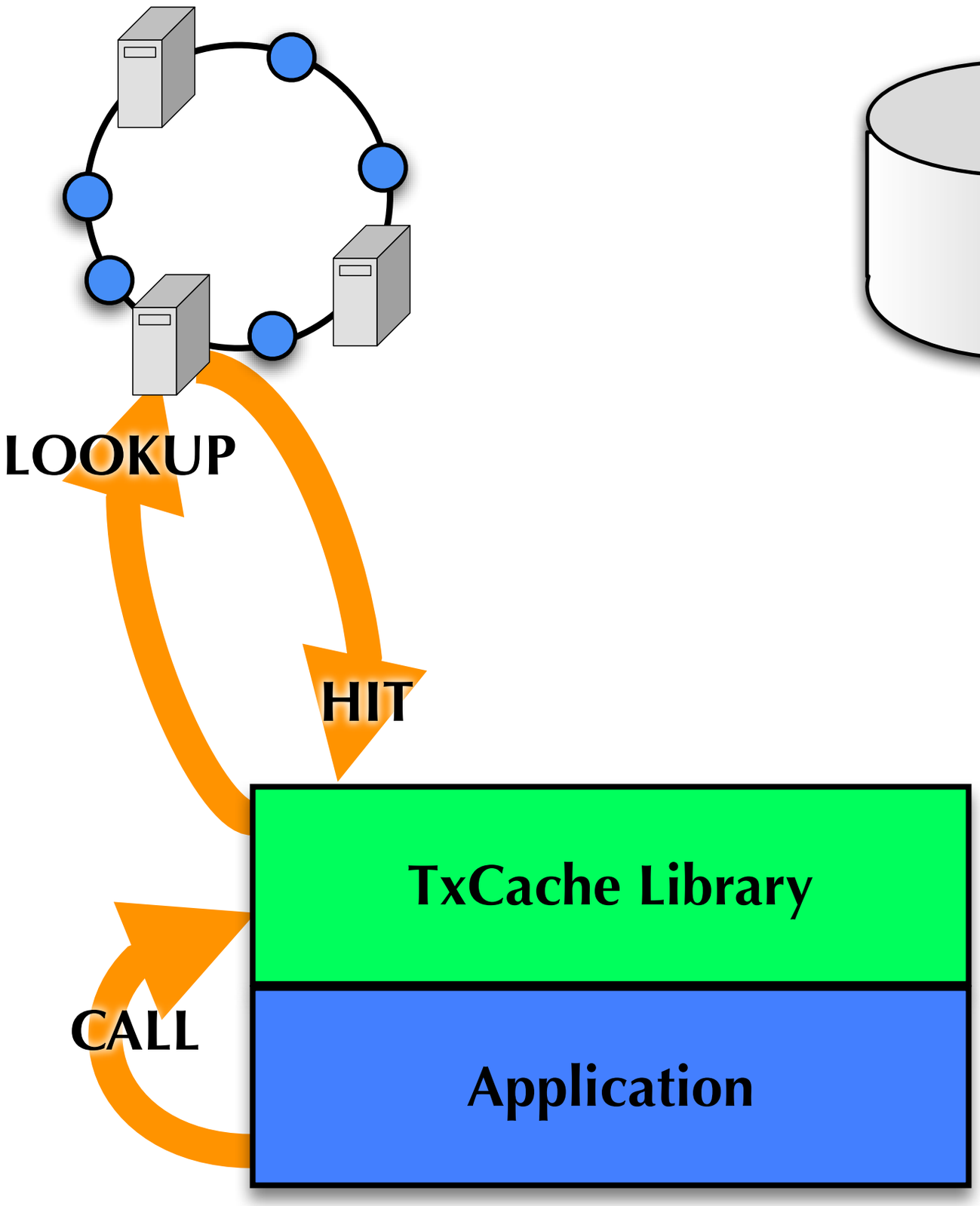
Application

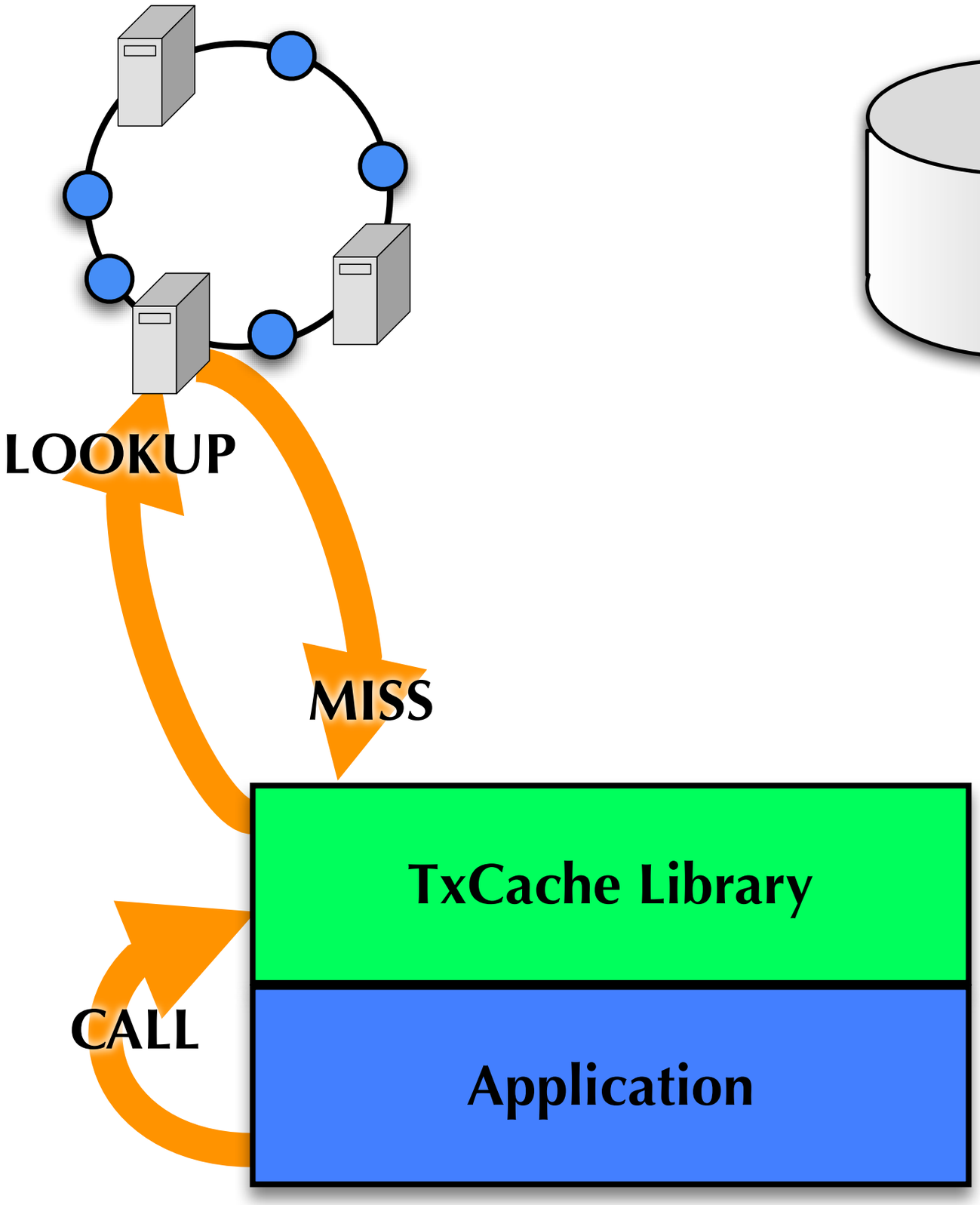


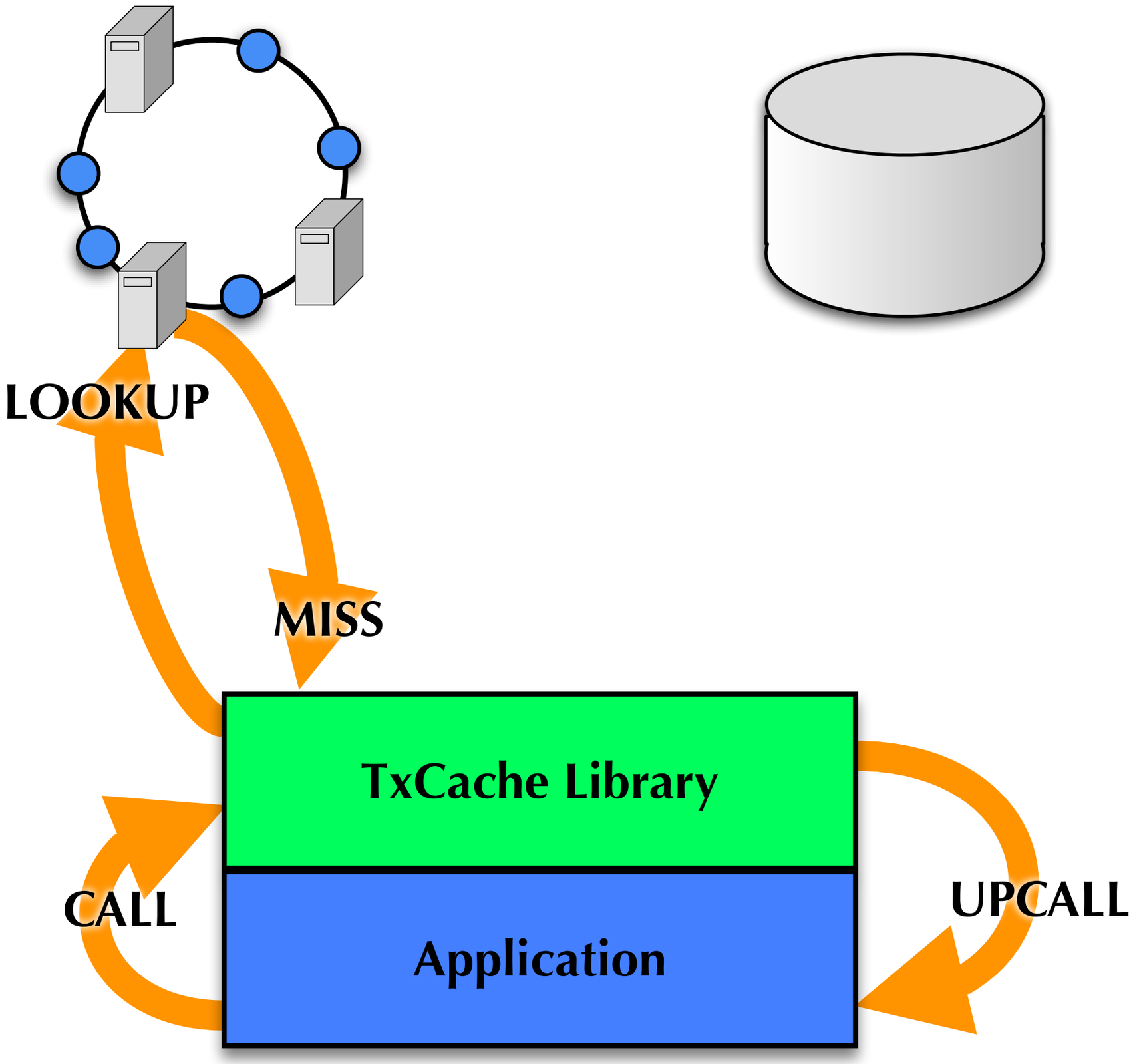


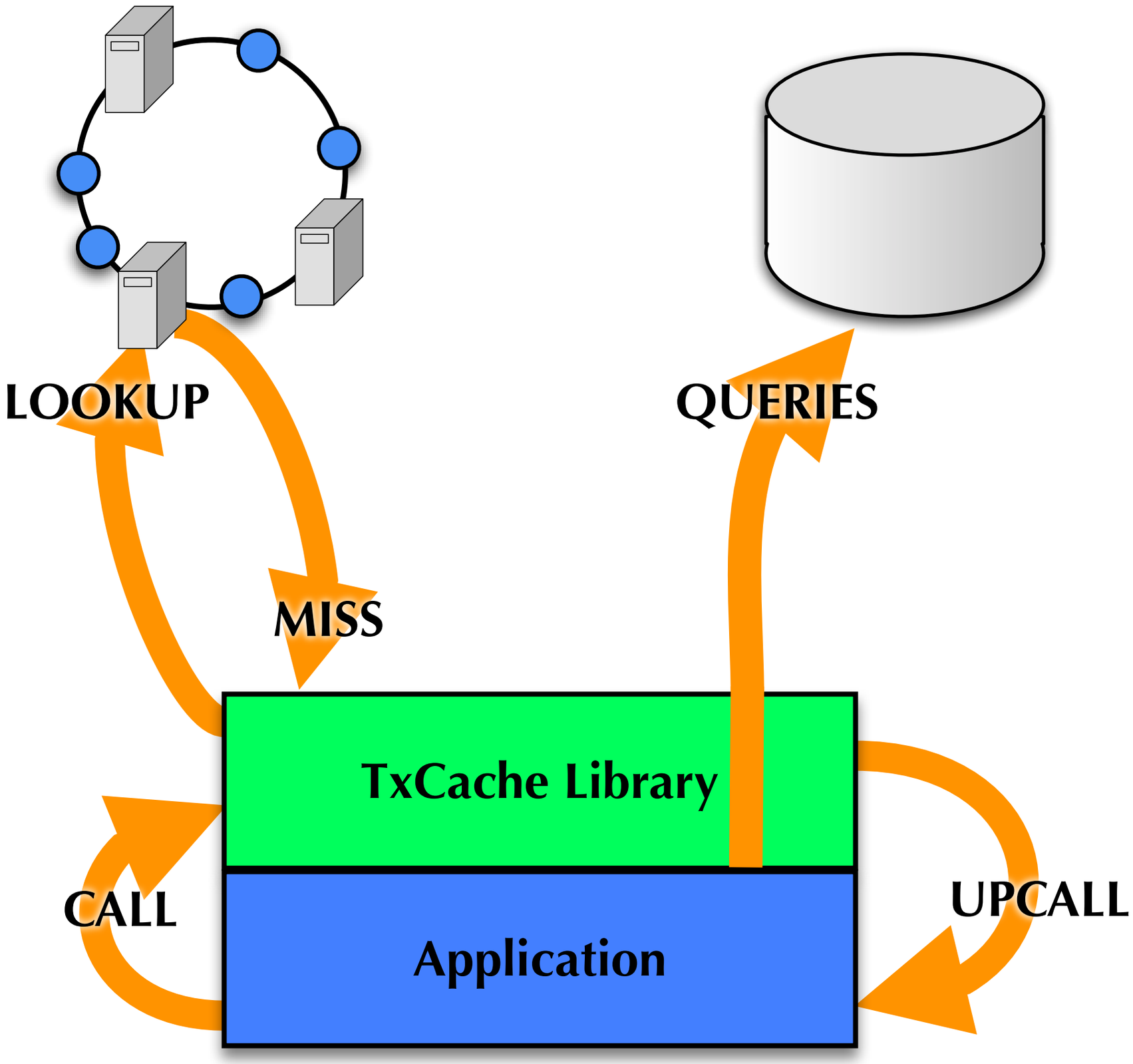

TxCache Library
Application
CALL

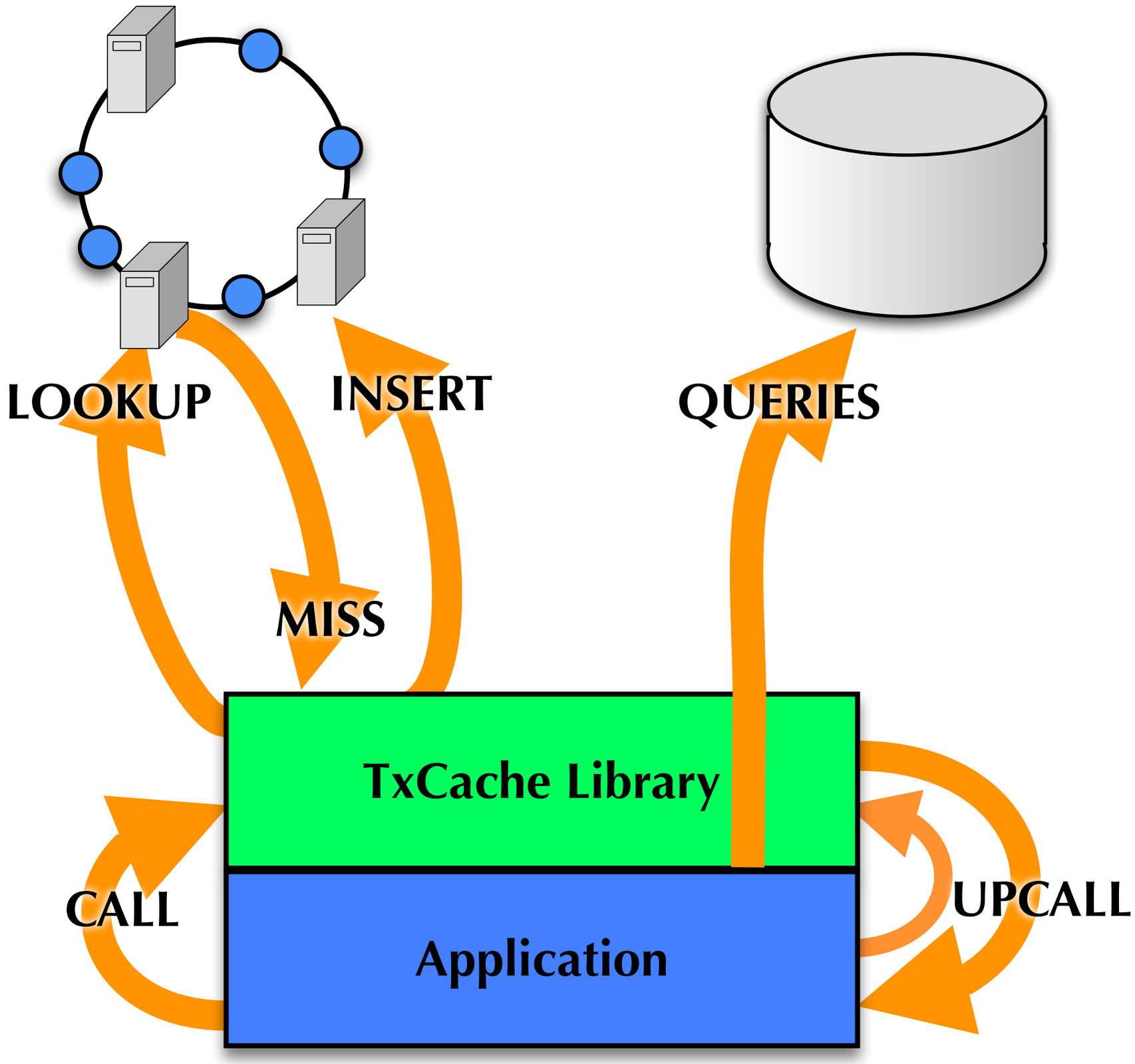

LOOKUP
TxCache Library
CALL
Application

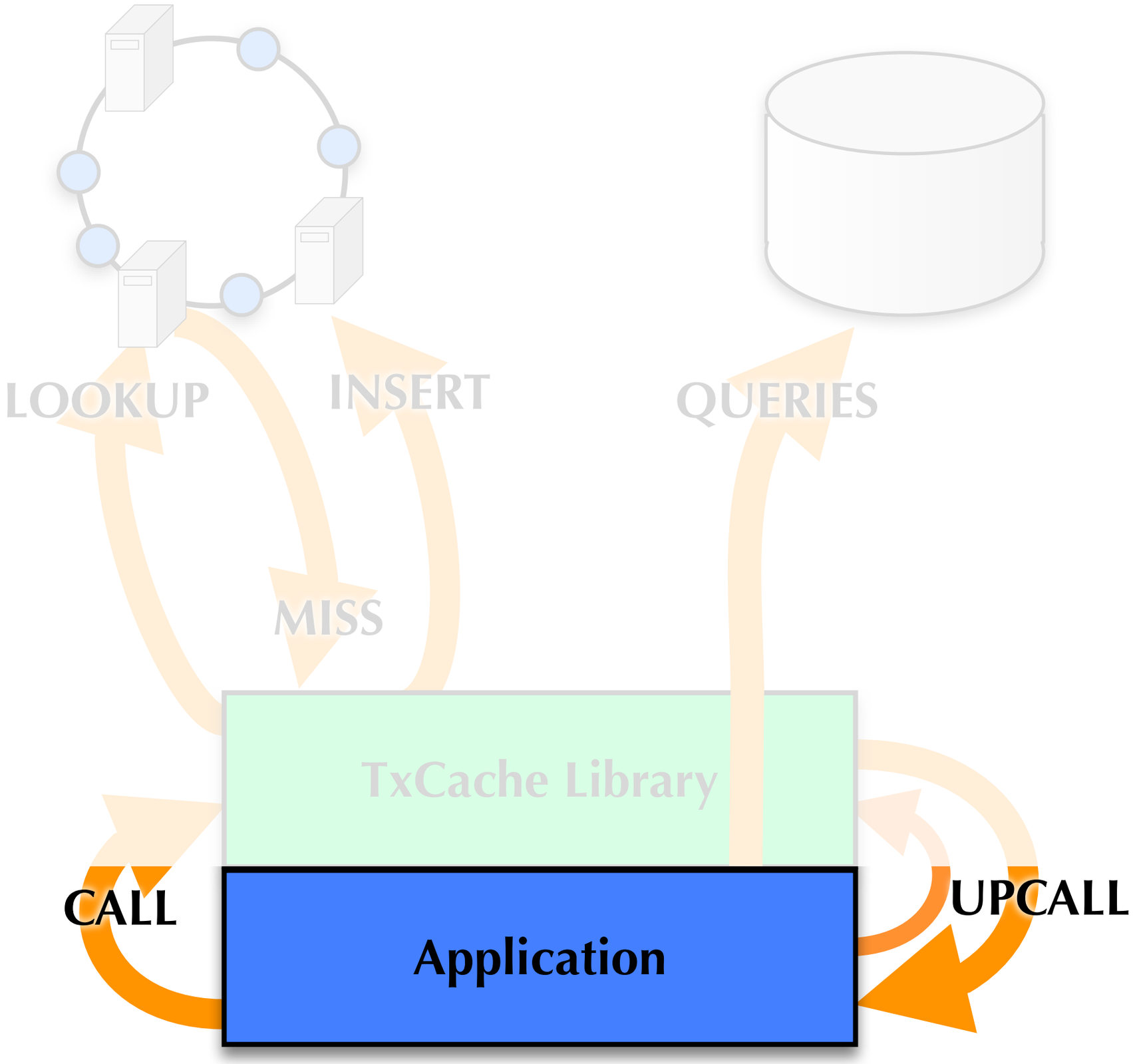

LOOKUP
HIT
CALL
TxCache Library
Application


LOOKUP
MISS
CALL
TxCache Library
Application


LOOKUP
MISS
TxCache Library
Application
CALL
UPCALL


LOOKUP
MISS
QUERIES
TxCache Library
Application
CALL
UPCALL


LOOKUP
INSERT
QUERIES
MISS
TxCache Library
Application
CALL
UPCALL


LOOKUP
INSERT
QUERIES
MISS
TxCache Library
CALL
Application
UPCALL

Consistency Approach

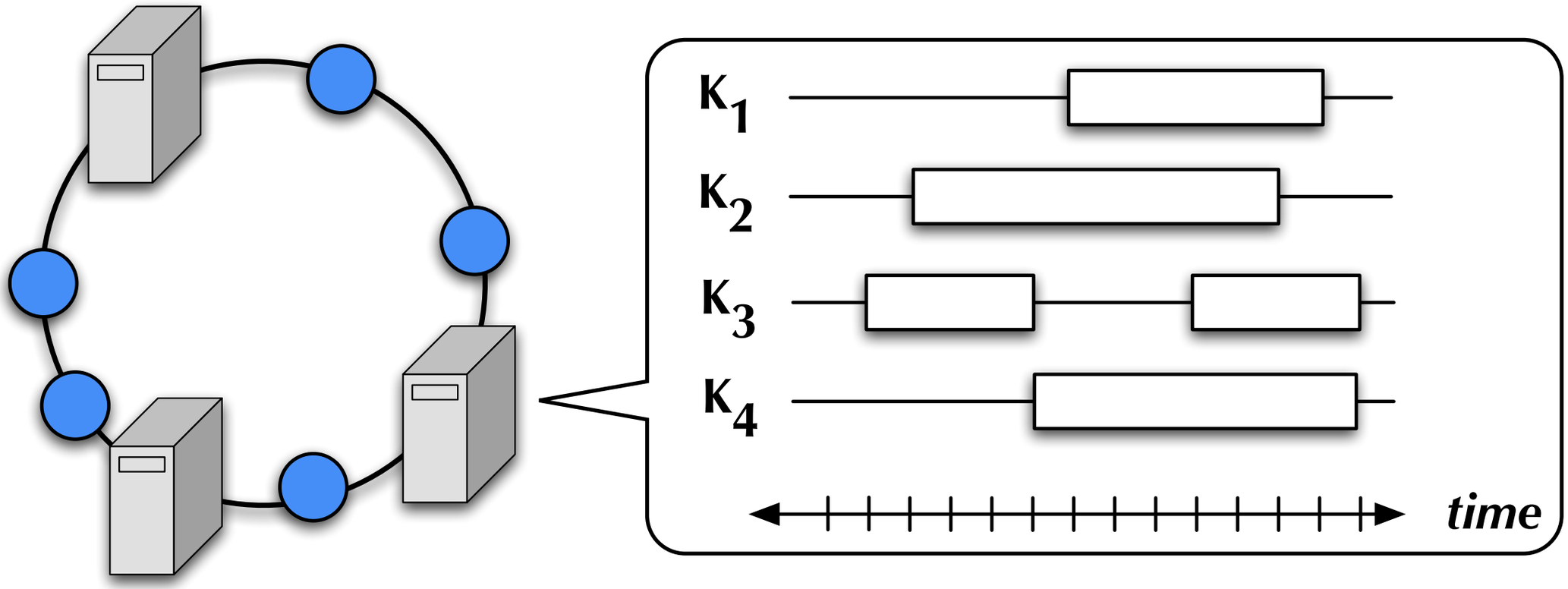
Goal: all data seen in a transaction reflects single point-in-time snapshot

- Assign timestamp to transaction
- Know the ***validity interval*** of each object in cache or database:
  set of timestamps when it was valid
- Then: transaction can read data if data's validity interval contains txn's timestamp

A Versioned Cache

Cache entries tagged with validity intervals

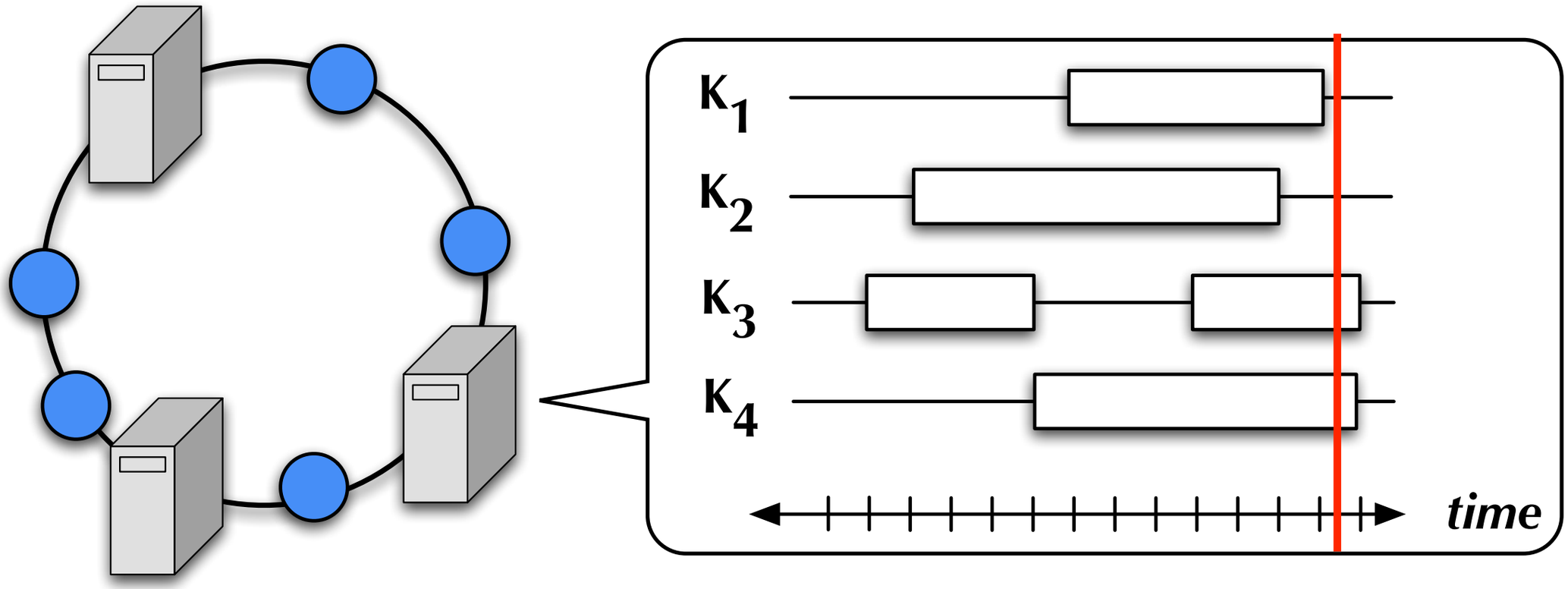
- each entry one immutable version of an object
- allows lookup for value valid at certain time



A Versioned Cache

Cache entries tagged with validity intervals

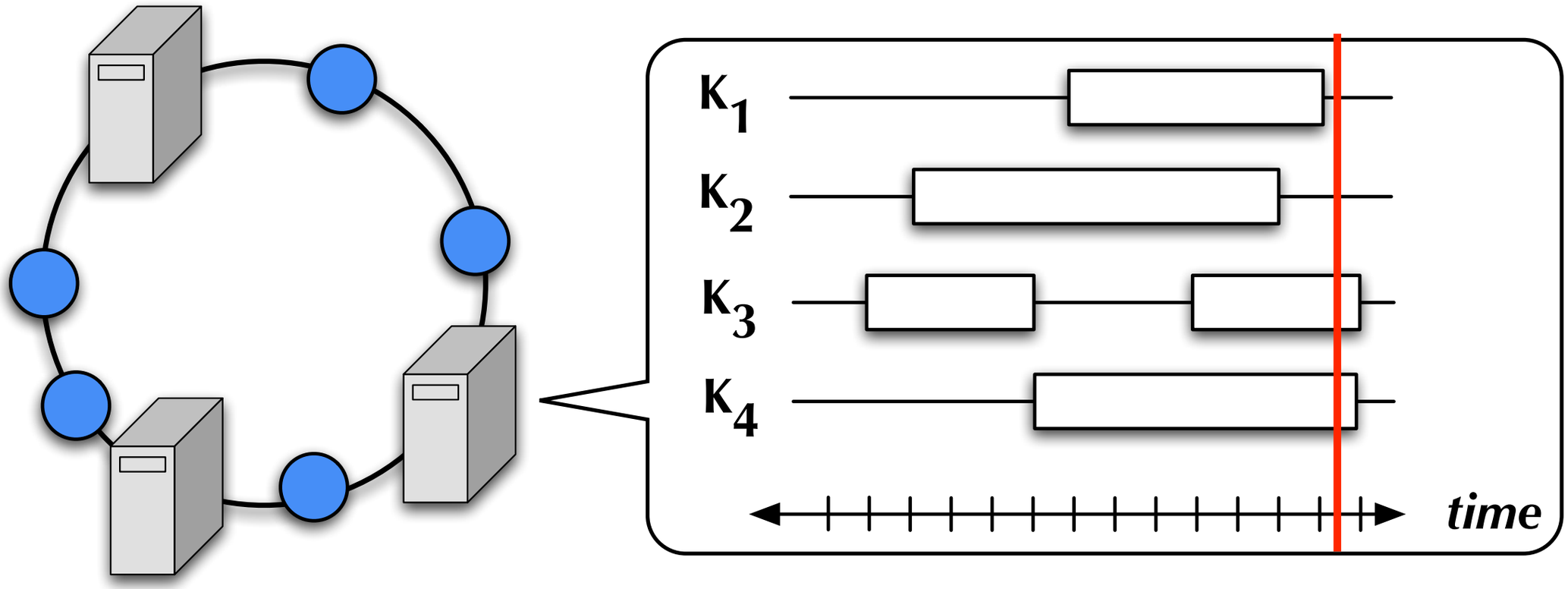
- each entry one immutable version of an object
- allows lookup for value valid at certain time



Staleness

Assign transaction an earlier timestamp

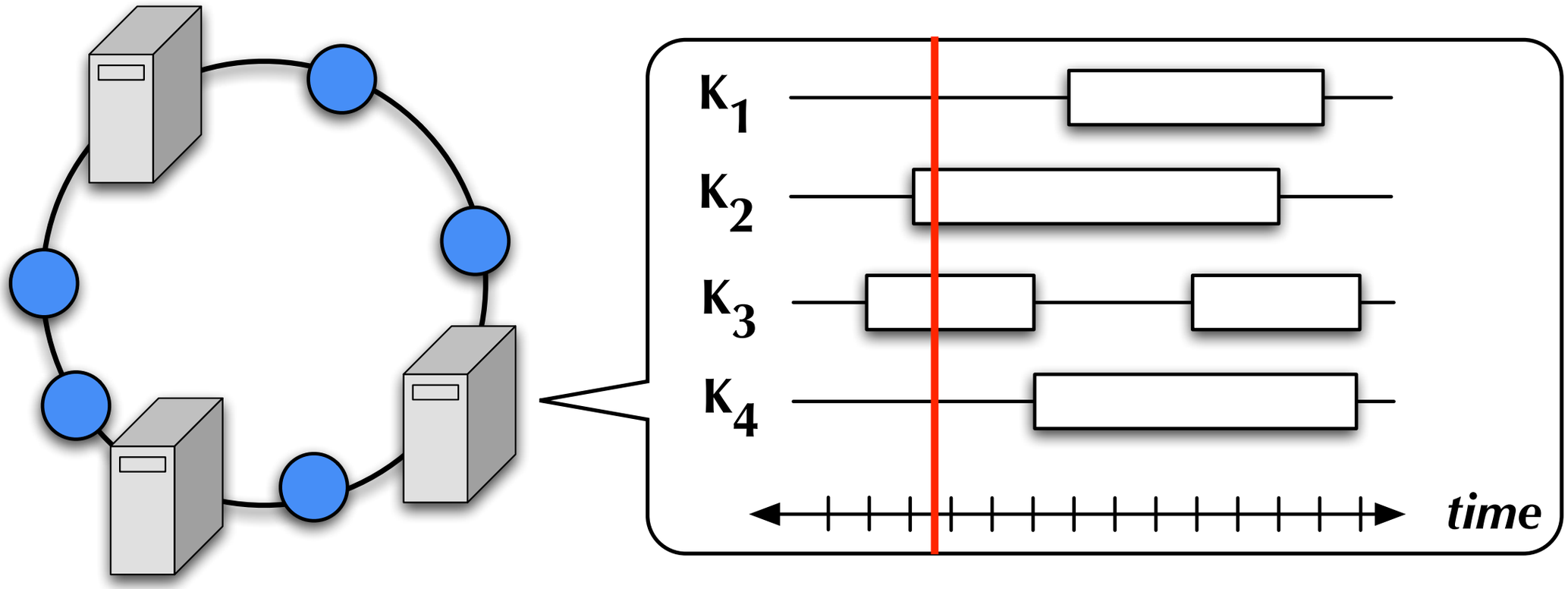
- if consistent with application requirements
- allows cached data to be used longer



Staleness

Assign transaction an earlier timestamp

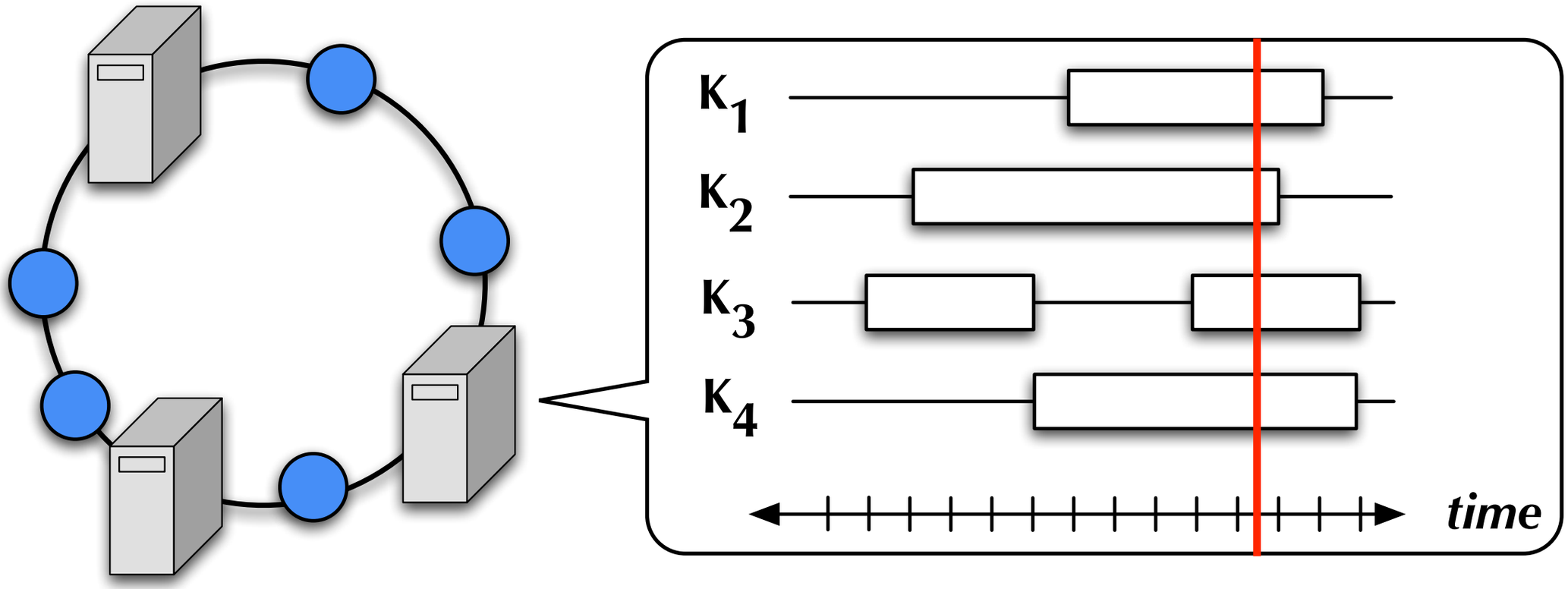
- if consistent with application requirements
- allows cached data to be used longer



Staleness

Assign transaction an earlier timestamp

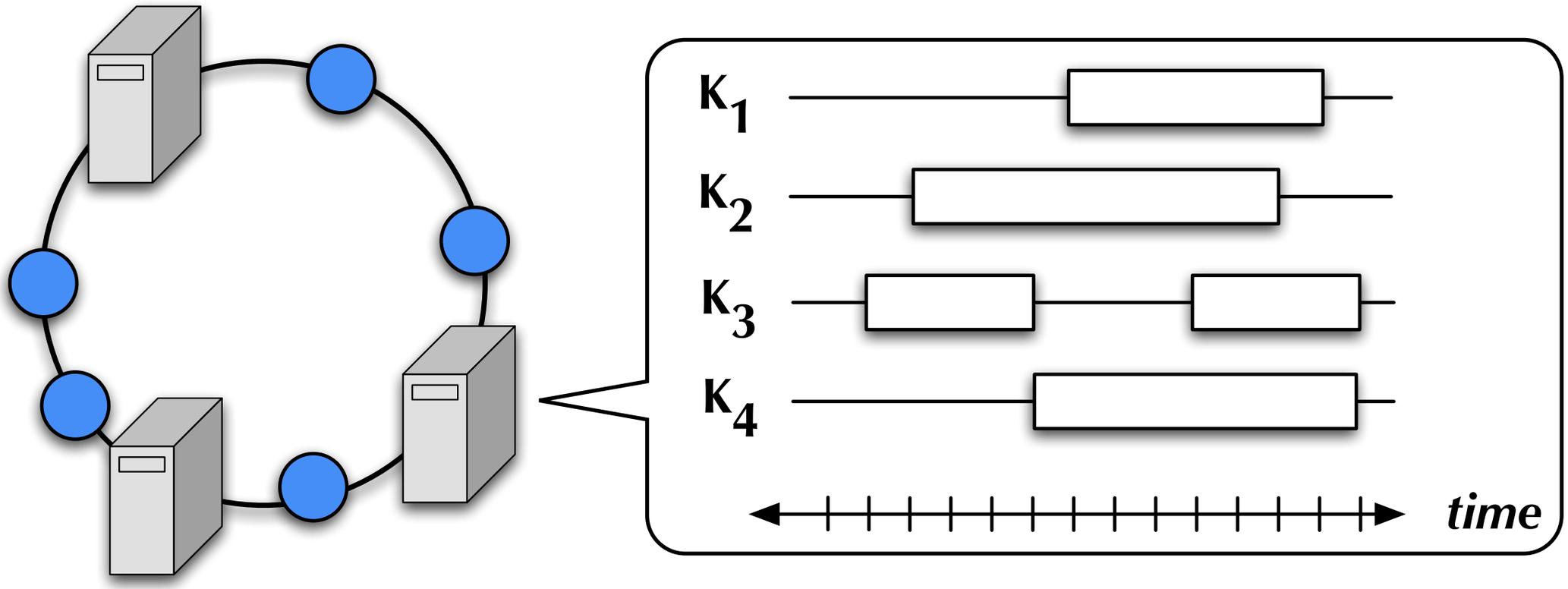
- if consistent with application requirements
- allows cached data to be used longer



Lazy Timestamp Selection

Hard to choose timestamp *a priori*

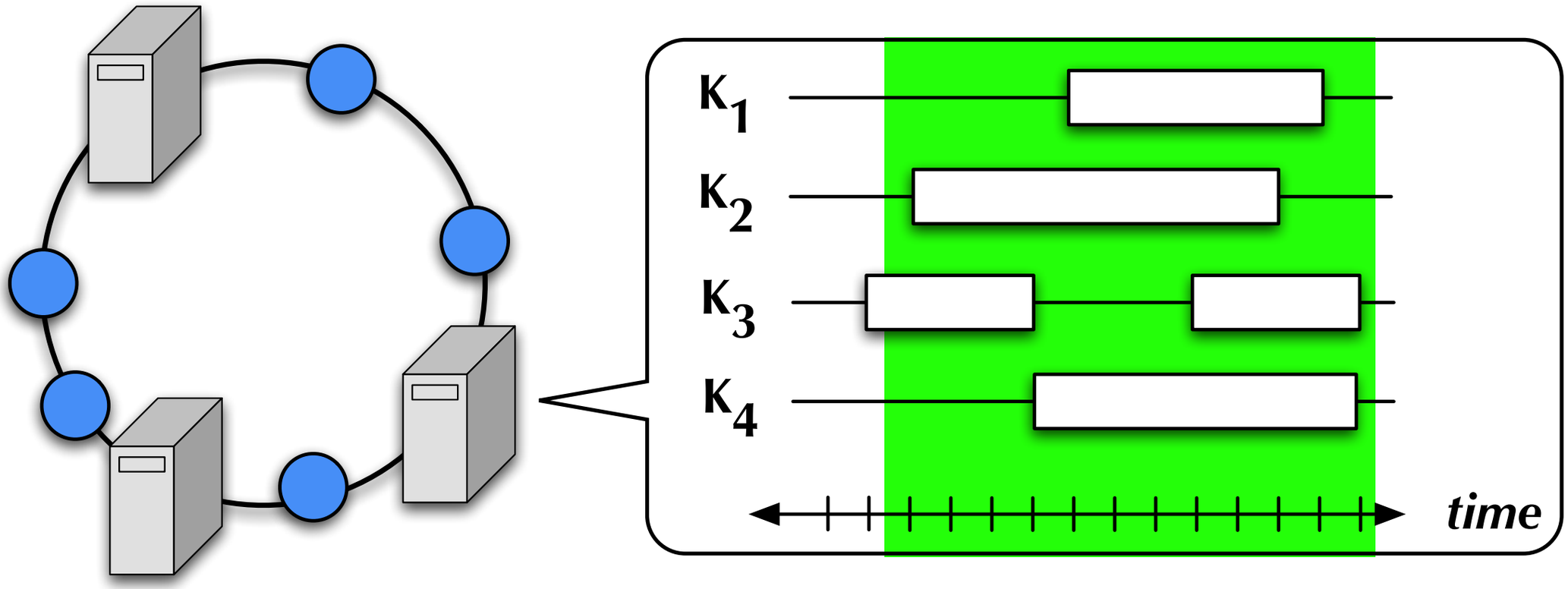
- Don't know access pattern *or* cache contents
- **Insight: don't have to choose right away!**



Lazy Timestamp Selection

Hard to choose timestamp *a priori*

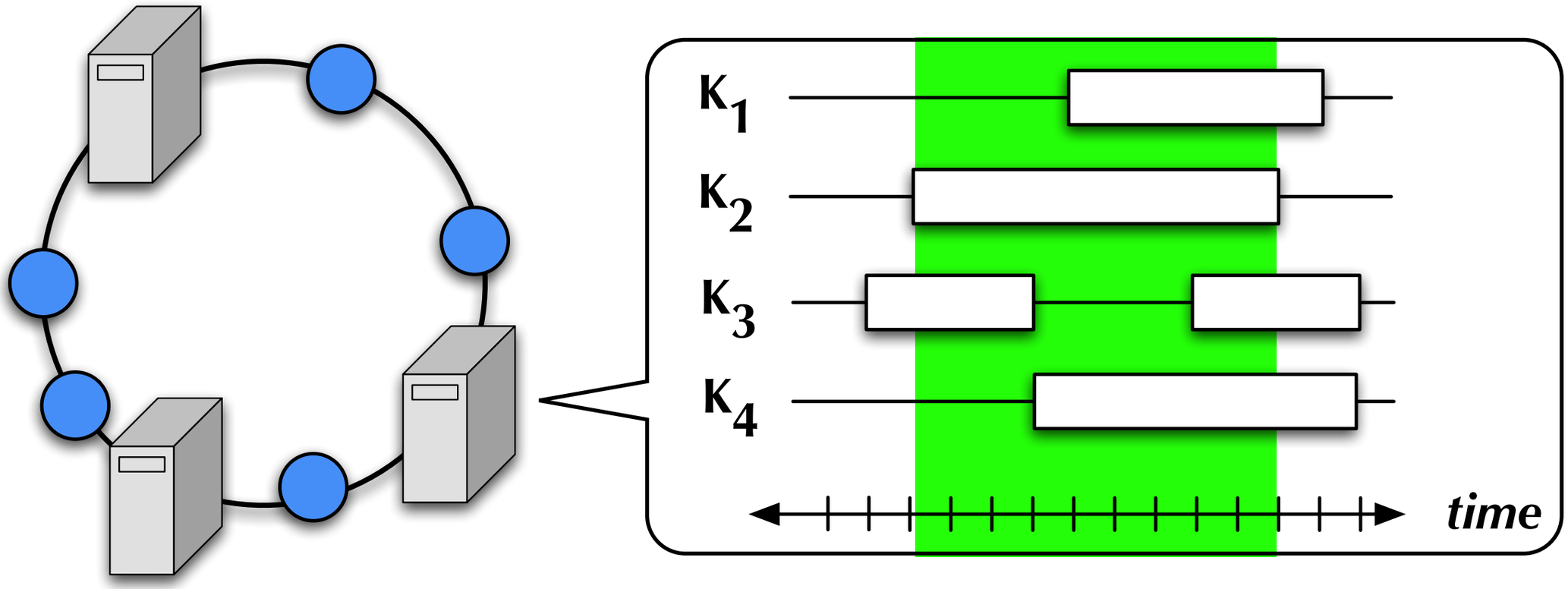
- Don't know access pattern *or* cache contents
- **Insight: don't have to choose right away!**



Lazy Timestamp Selection

Hard to choose timestamp *a priori*

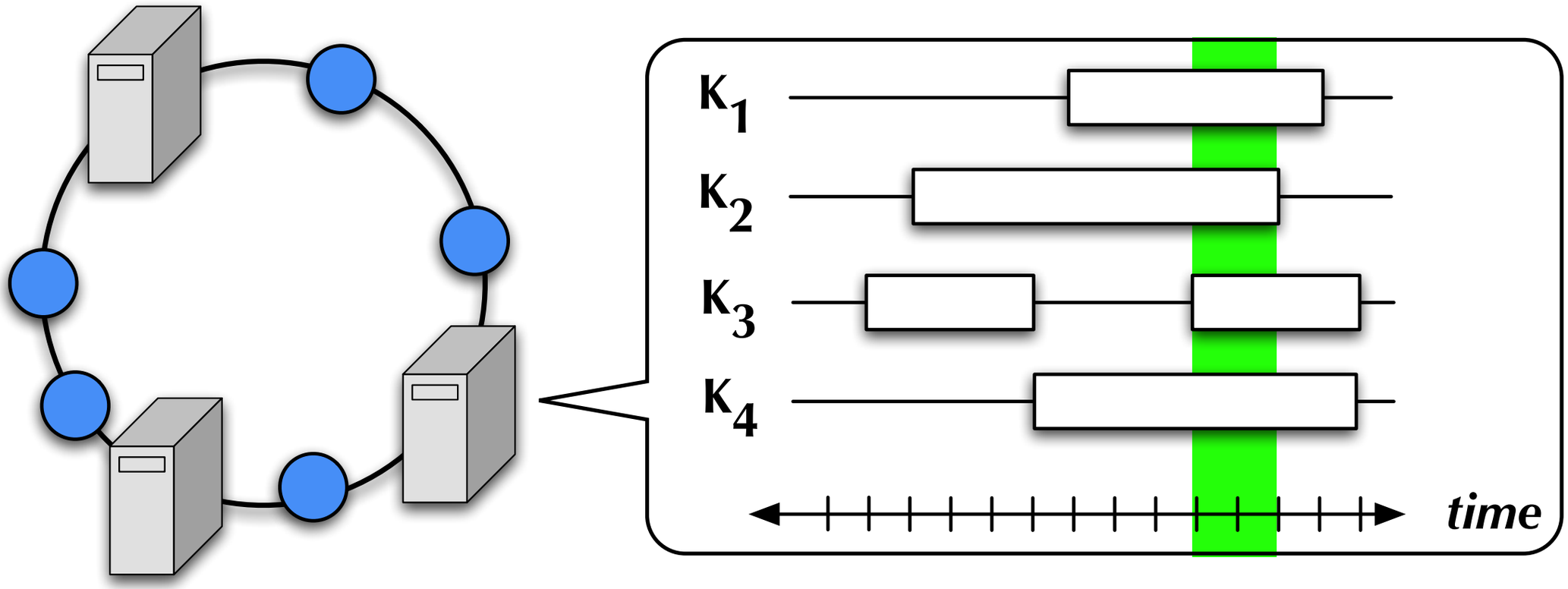
- Don't know access pattern *or* cache contents
- **Insight: don't have to choose right away!**



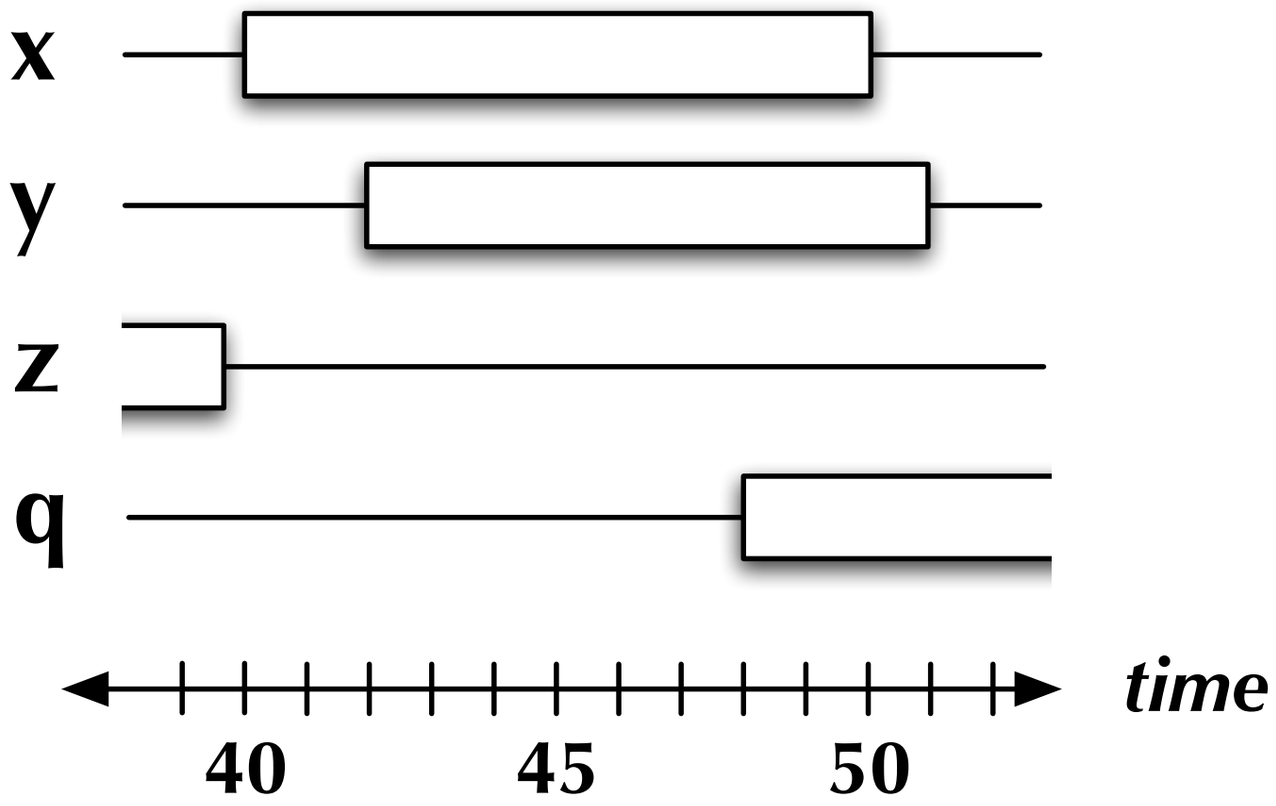
Lazy Timestamp Selection

Hard to choose timestamp *a priori*

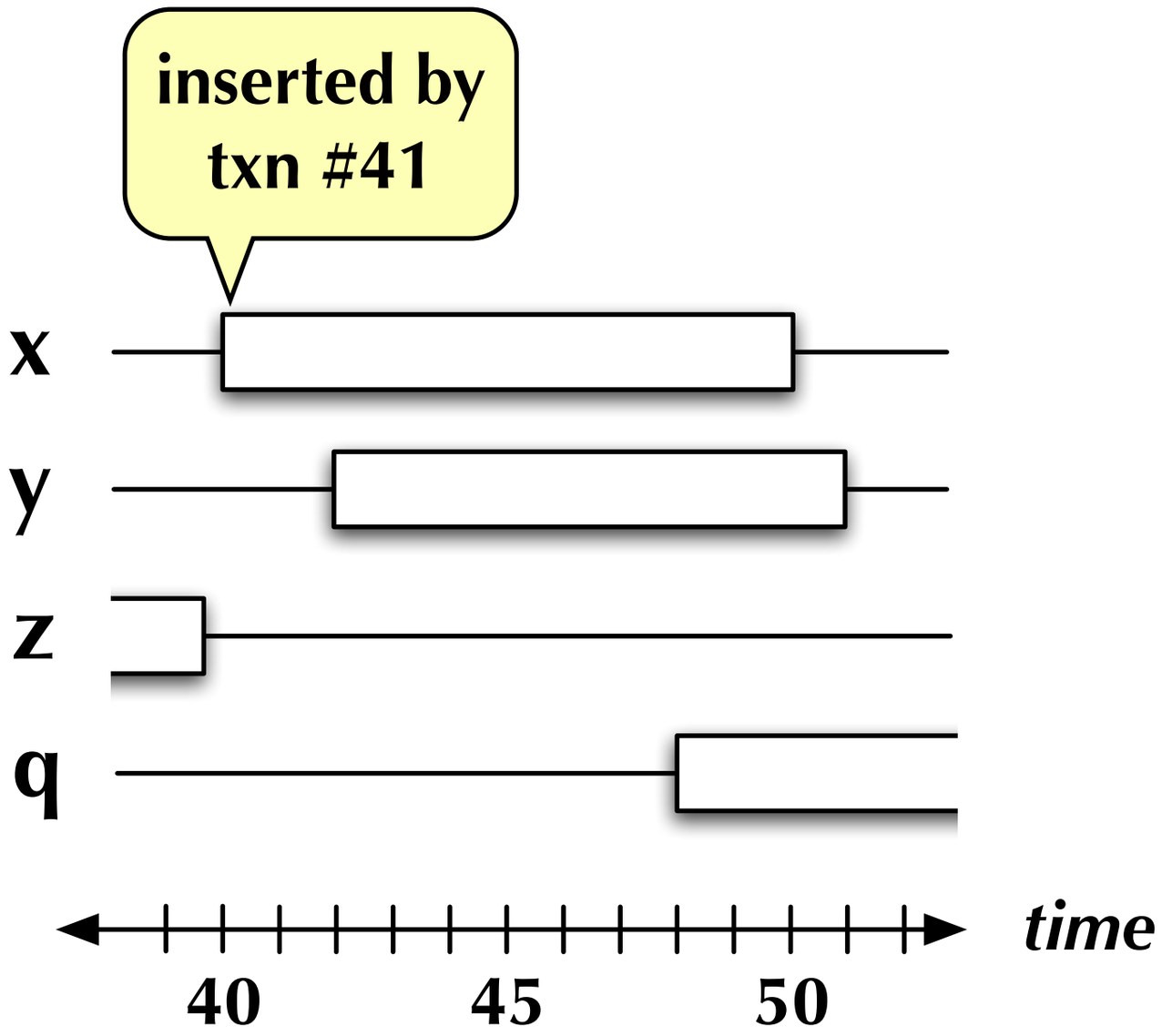
- Don't know access pattern *or* cache contents
- **Insight: don't have to choose right away!**



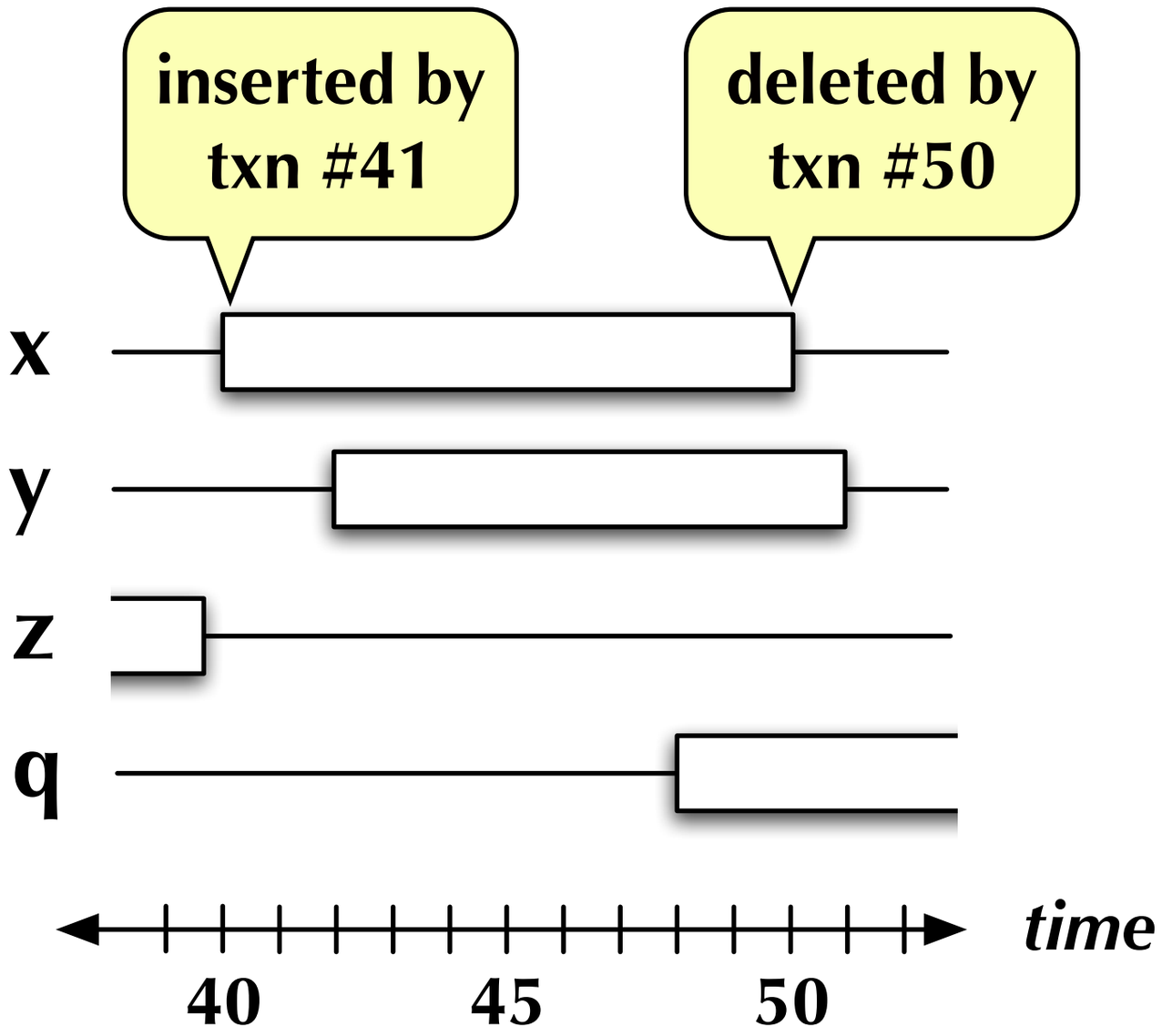
DB Tracks Validity Intervals



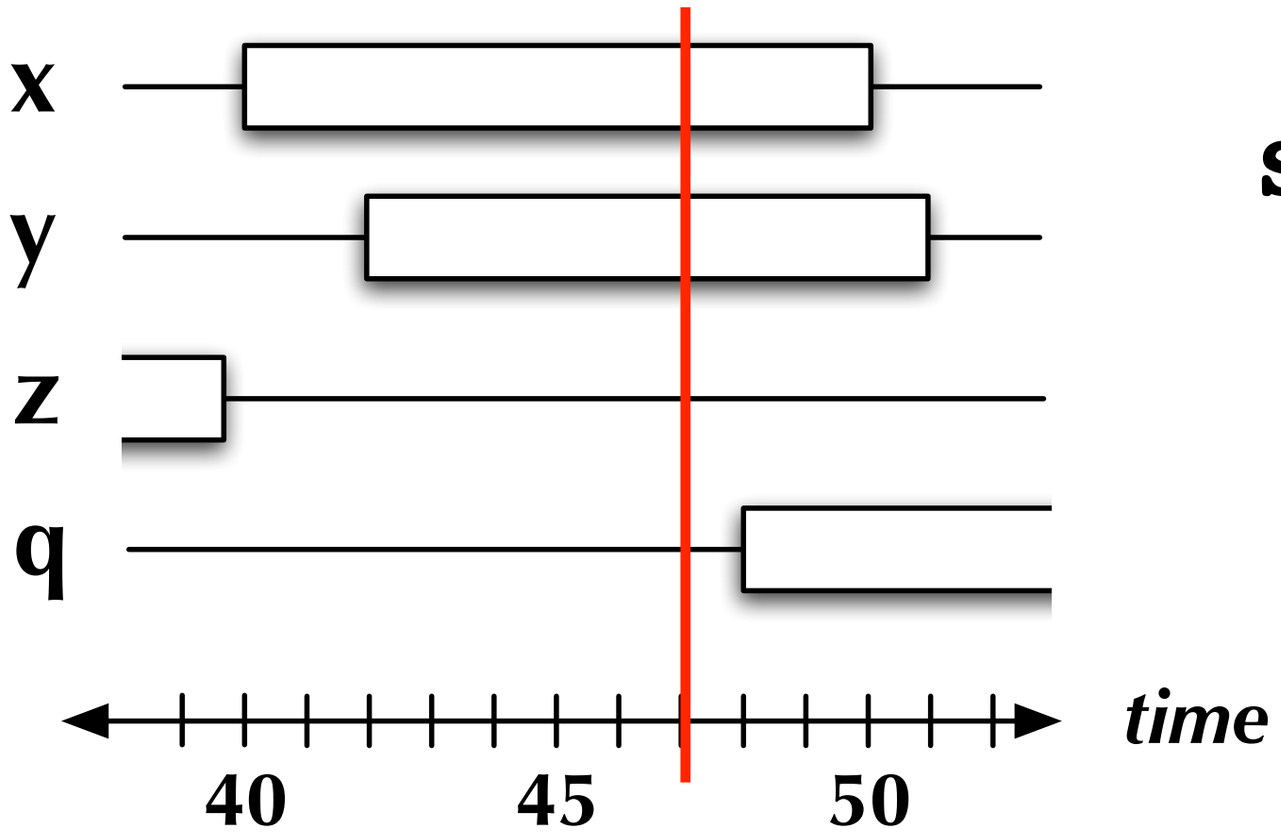
DB Tracks Validity Intervals



DB Tracks Validity Intervals

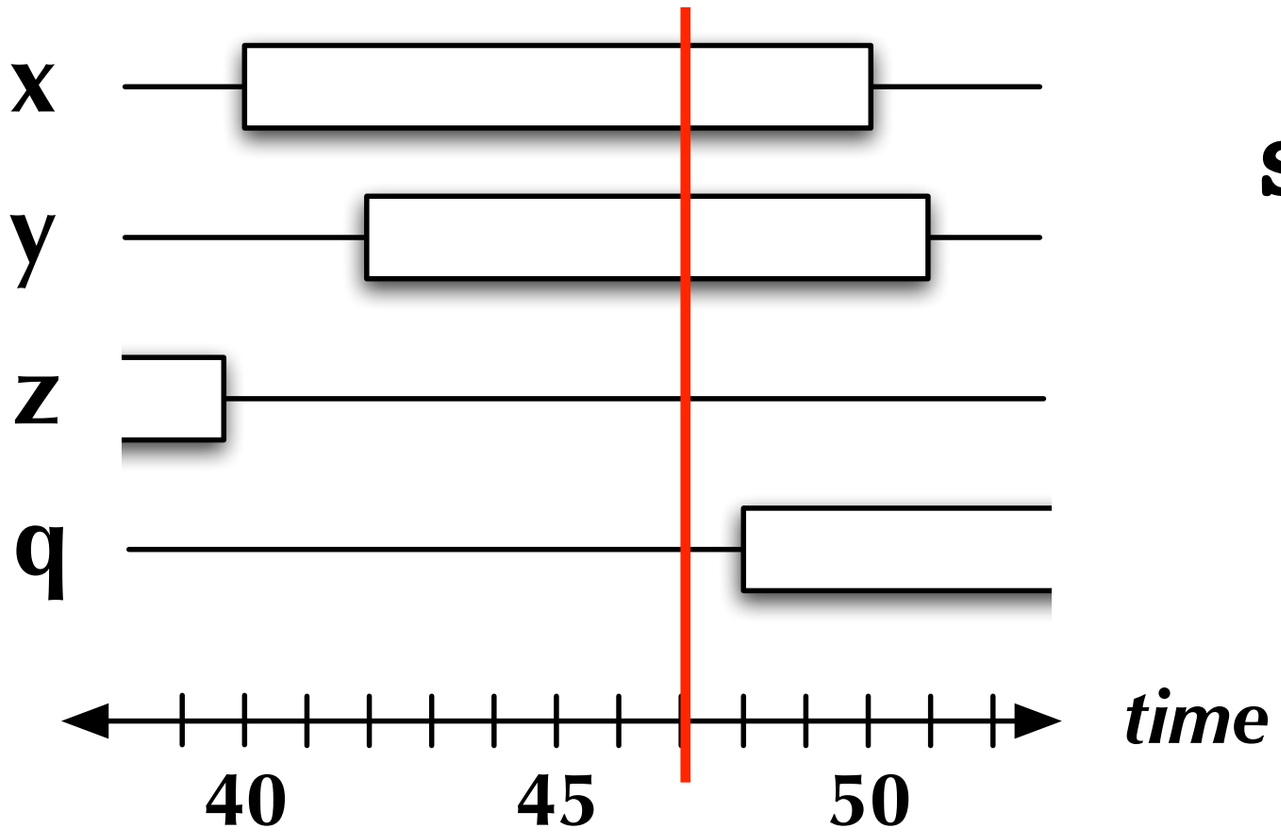


DB Tracks Validity Intervals




```
SELECT * FROM ...;
```

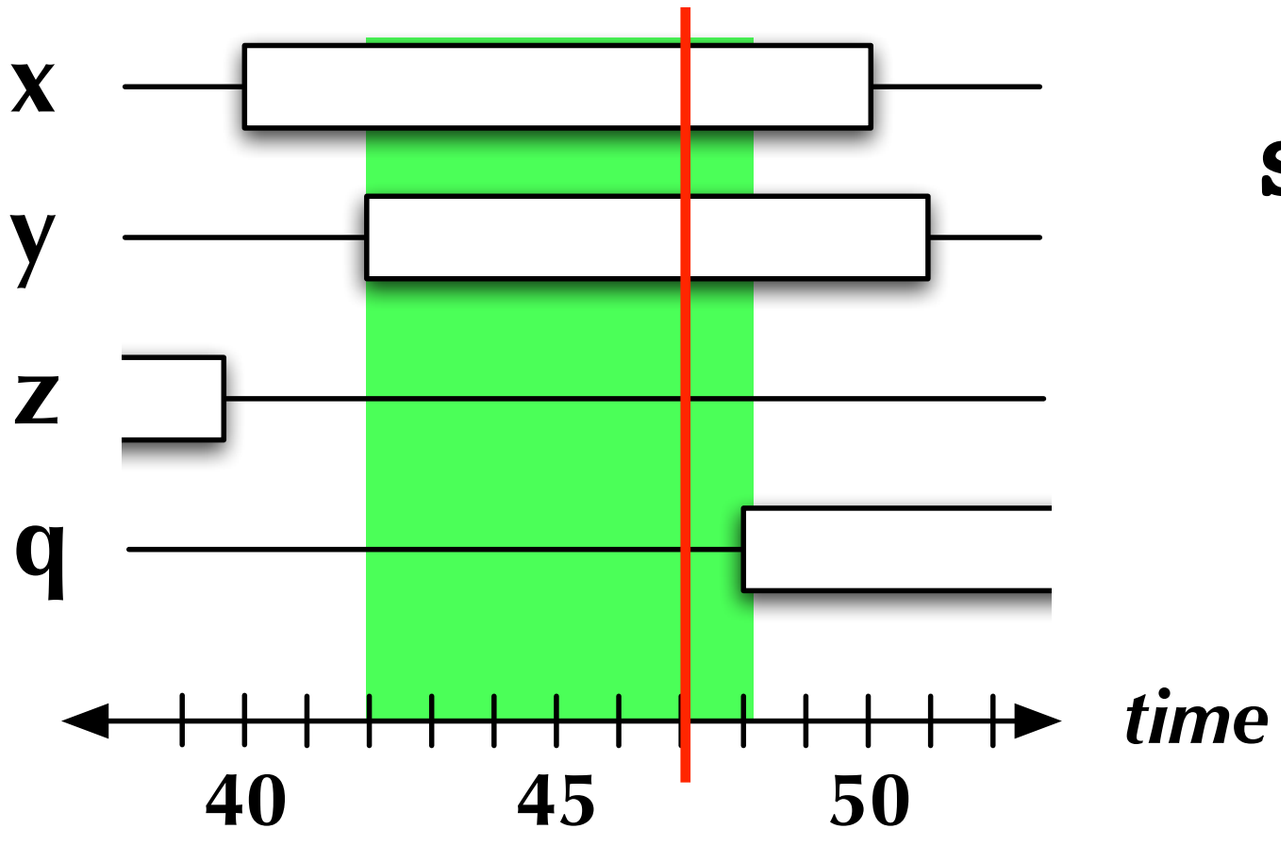
DB Tracks Validity Intervals




```
SELECT * FROM ...;
  result = {x, y}
```

DB Tracks Validity Intervals

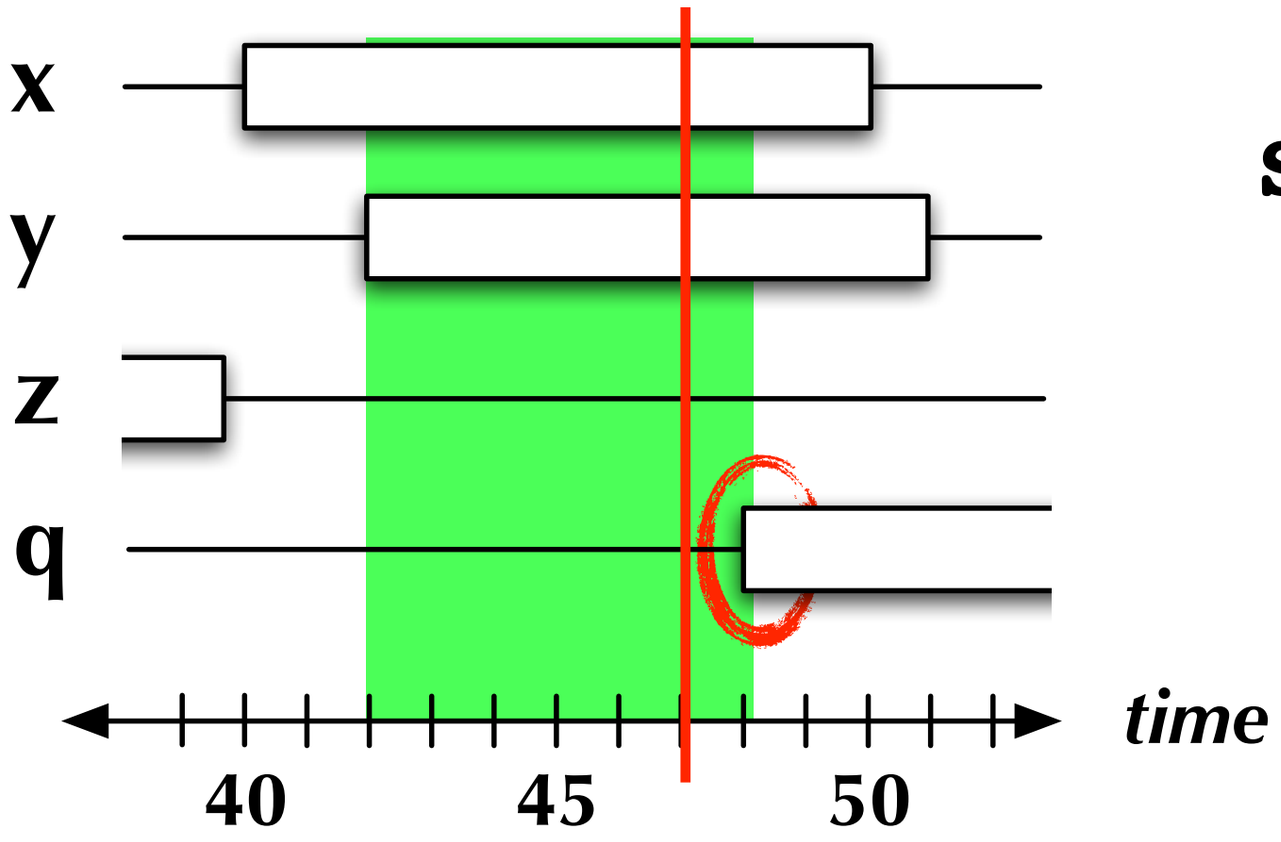
Intersect validity intervals of tuples accessed




```
SELECT * FROM ...;
  result = {x, y}
  VALIDITY [41, 48)
```

DB Tracks Validity Intervals

Intersect validity intervals of tuples accessed

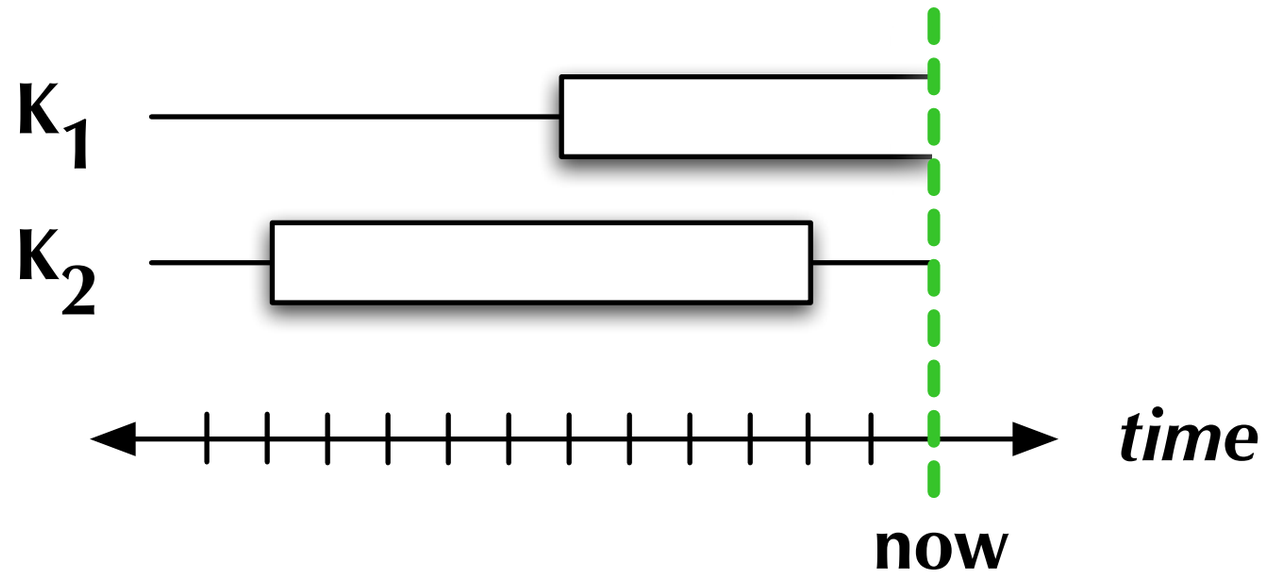



```
SELECT * FROM ...;
  result = {x, y}
  VALIDITY [41, 48)
```

Invalidations

What about objects that are still valid?

- don't know their upper validity bound yet!
- represent as open-ended validity intervals



Later, database notifies cache if object changes; cache truncates interval

Invalidation Tags

How to identify which objects changed?

- DB doesn't know which app-level objects are cached

Objects in cache have ***invalidation tags***

- Modified DB to assign invalidation tags to each query
  - `TABLE:KEY=VALUE` for queries that use index lookups
  - `TABLE:*` for non-indexed queries (rare)
- DB broadcasts list of tags affected by each update
- Cache finds affected objects and updates interval

Evaluation

- How much benefit from adding caching?
- Does consistency hurt performance?

Also [see paper, OSDI'10]:

- How much does using stale data help?
- Ease of adding caching to existing apps

RUBiS Benchmarks

RUBiS: simulated eBay-like auction site

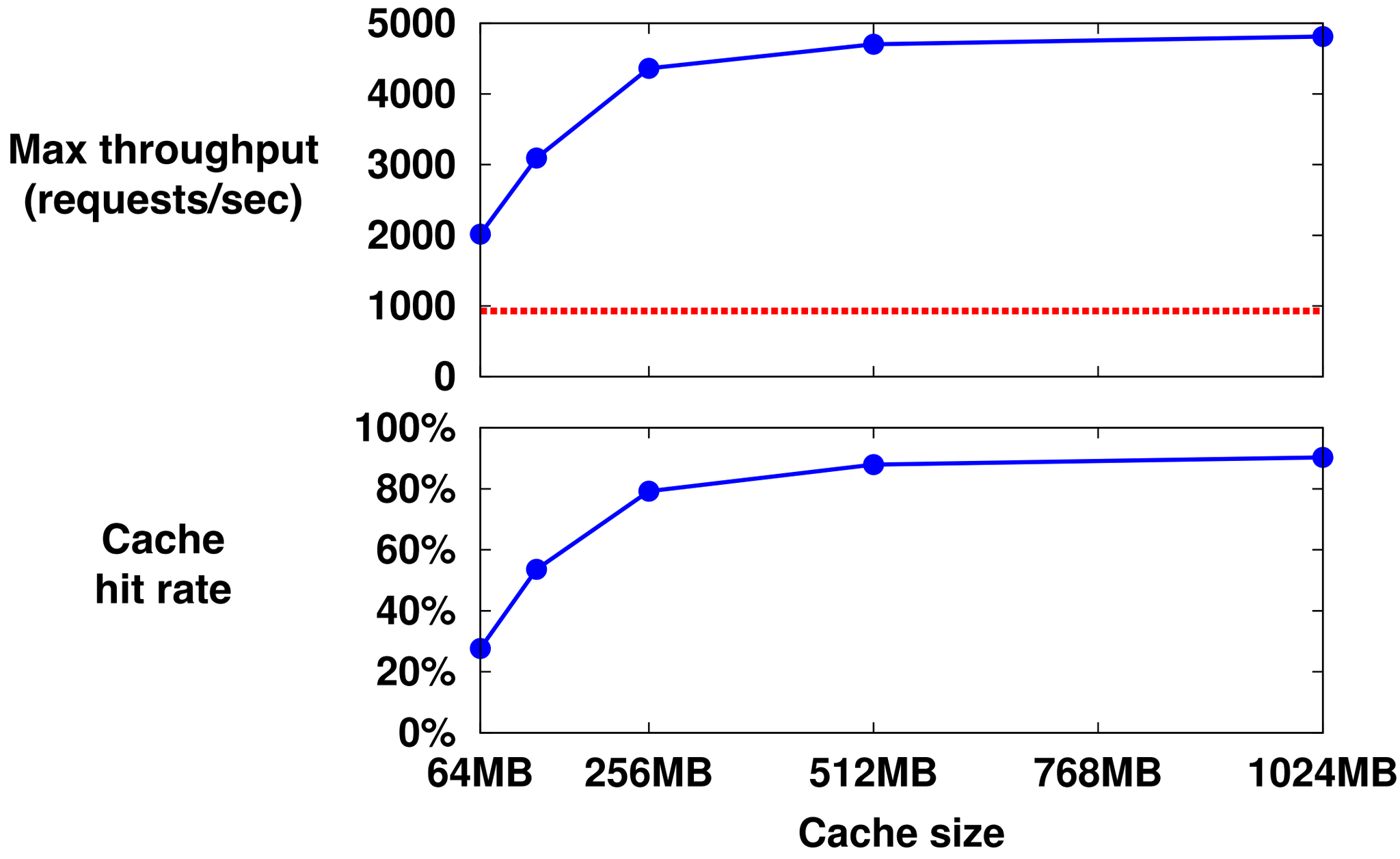
- standard browsing & bidding workload; 85% read-only
- two datasets: 850 MB (in-memory), 6 GB (disk-bound)

All servers 2x 3.20 GHz Xeon, 2 GB RAM

- 1 DB server (modified Postgres 8.2.11)
- 9 frontend/cache servers (Apache 2 / PHP 5)

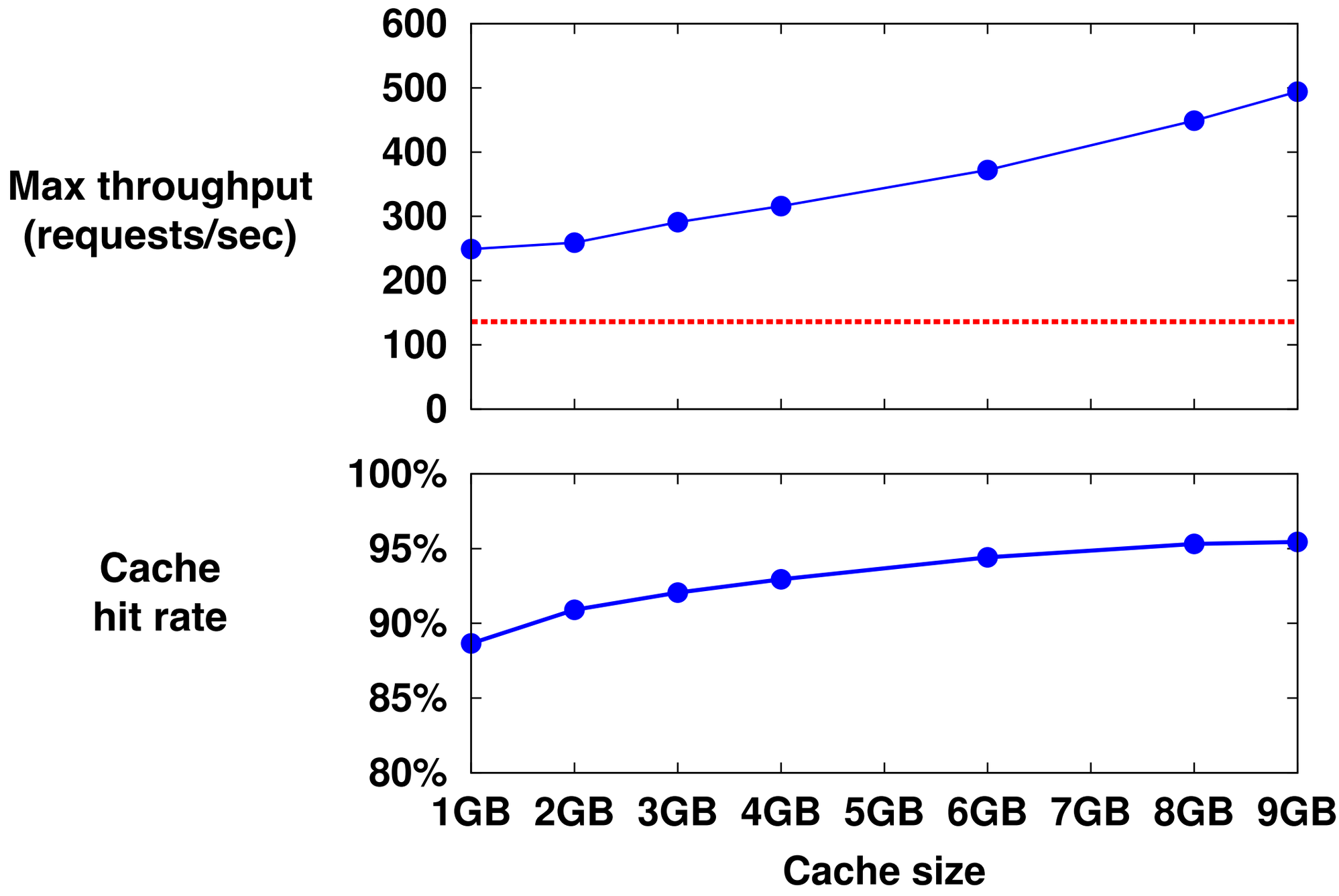
Cache Performance

(in-memory DB; 2 cache nodes)



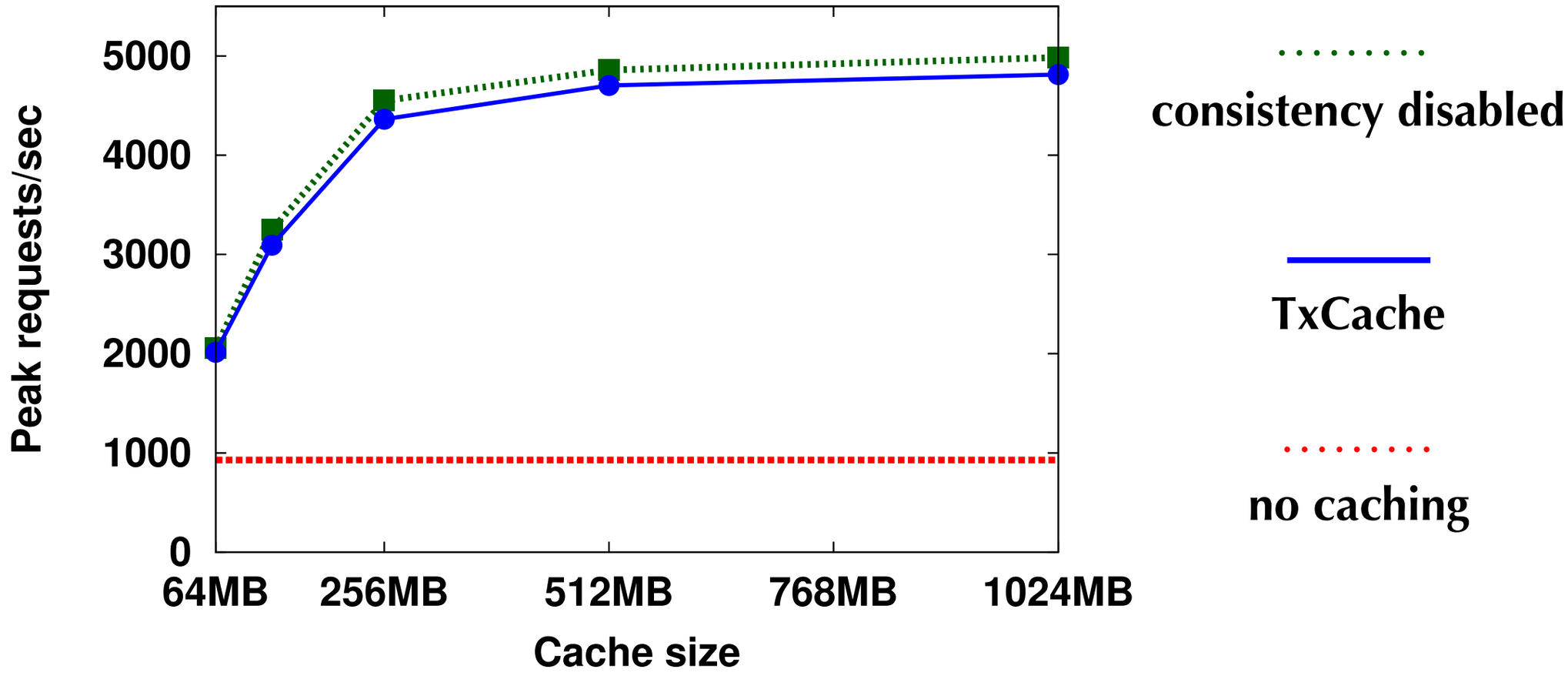
Cache Performance

(disk-bound DB; 8 shared web/cache nodes)



Costs of Consistency

0.2 – 7.8% cache misses caused by consistency



Ongoing Work

- Improved invalidations:
  more accurate tracking of depenencies
  and efficient distribution to many caches
- Efficient distributed transactions in
  partitioned/replicated databases
- Serializability for snapshot isolation DBs
  (available in PostgreSQL 9.1)

Conclusion

TxCache: application-layer caching with a simpler programming model

- provides transactional consistency across both cache and database
- automatic management: applications not responsible for lookups, updates, invalidations

New mechanisms:

- consistency ensured by tracking object validity intervals
- automatic database-generated invalidations

Consistency imposes little performance cost