Coherence Implementation Patterns

Ben Stopford
The Royal Bank of Scotland

Some Ideas

Nothing More

Why do we use Coherence?

Fast?

Scalable?

Application layer?

Simplifying the Contract

- We don’t want ACID all of the time
- We want to pick the bits we need when we need them
- We want to use the context of our business requirement to work our way around the ones we don’t need.

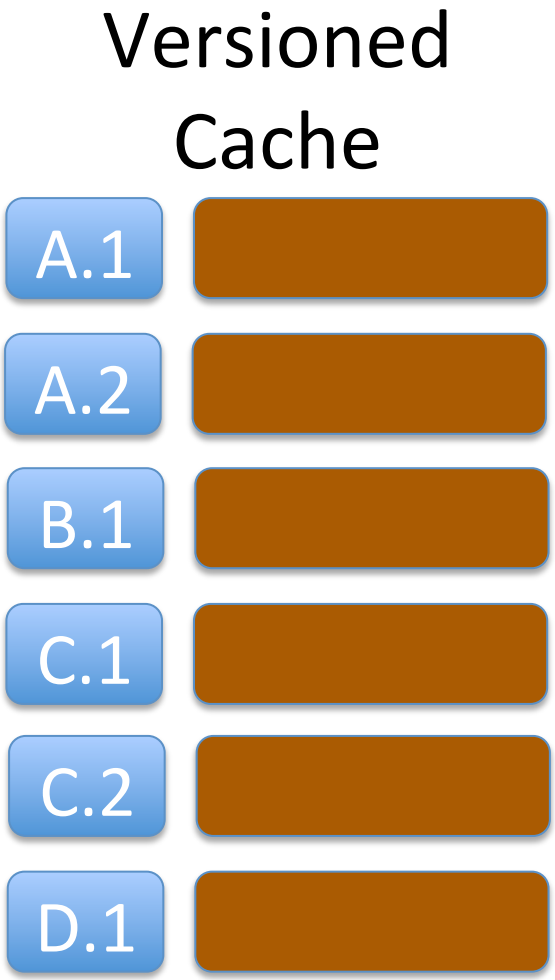
Version your Objects

Why do we care?

Without versioning it's a free-for-all.

- What changed?
- Was something overwritten?
- How can you prevent concurrent updates?
- What did the system look like 10 seconds ago.
- How can I provide a consistent view?
- How to I ensure ordering of updates in an asynchronous system?

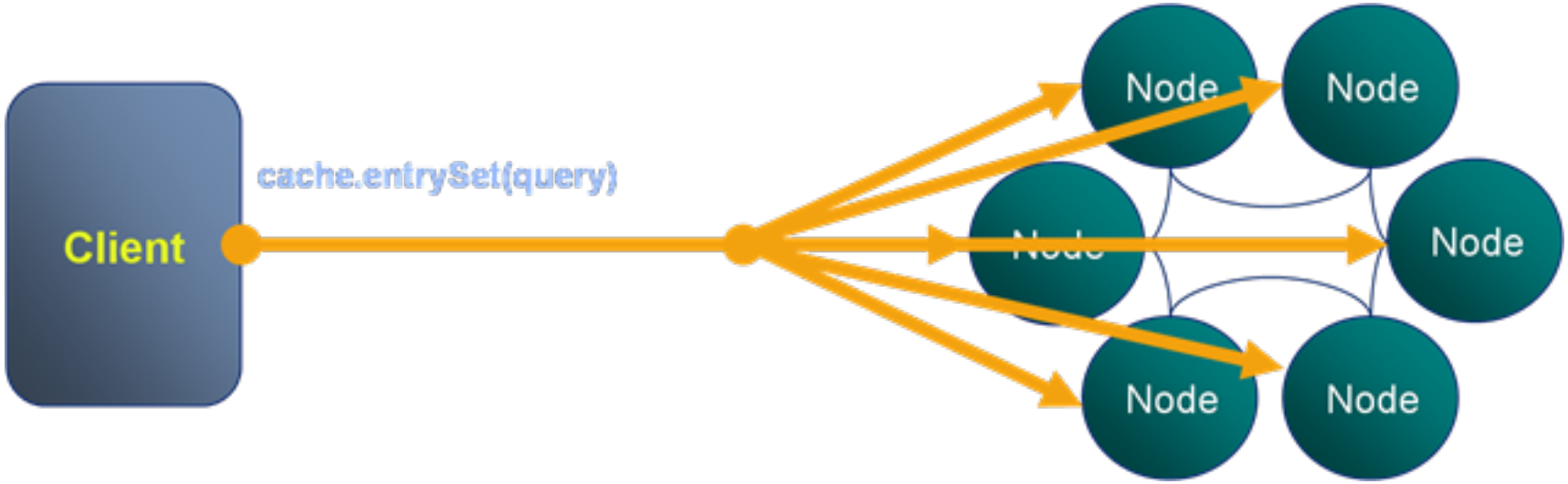
Versioning your Objects



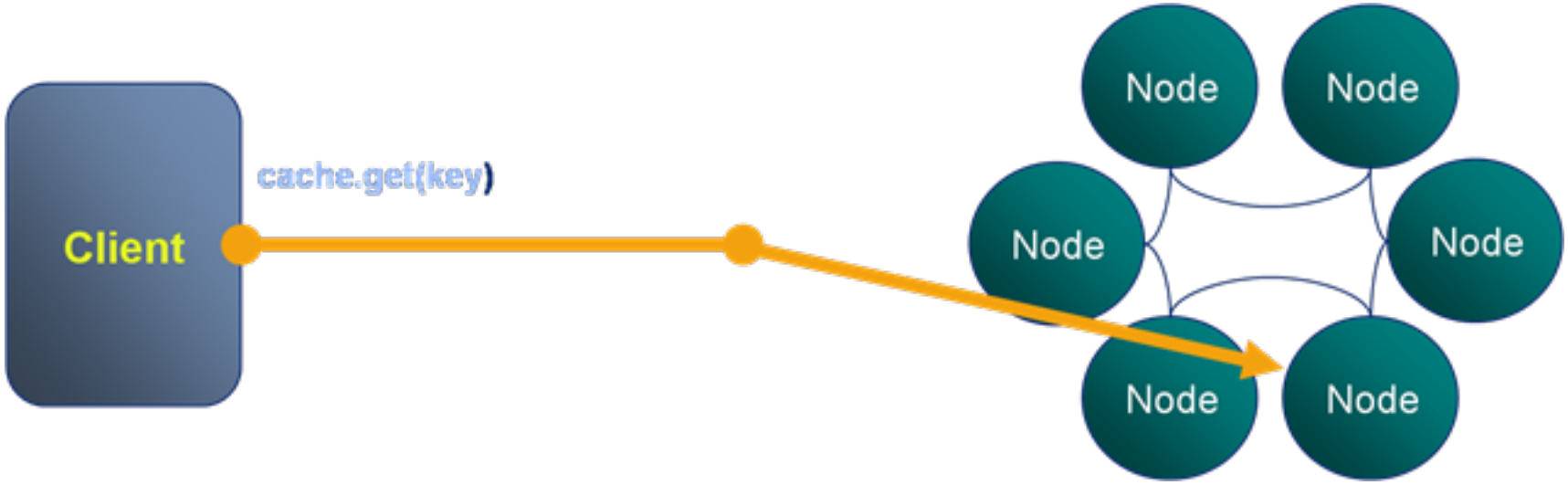
Versioning your Objects

New Version = Old Version + 1 ??

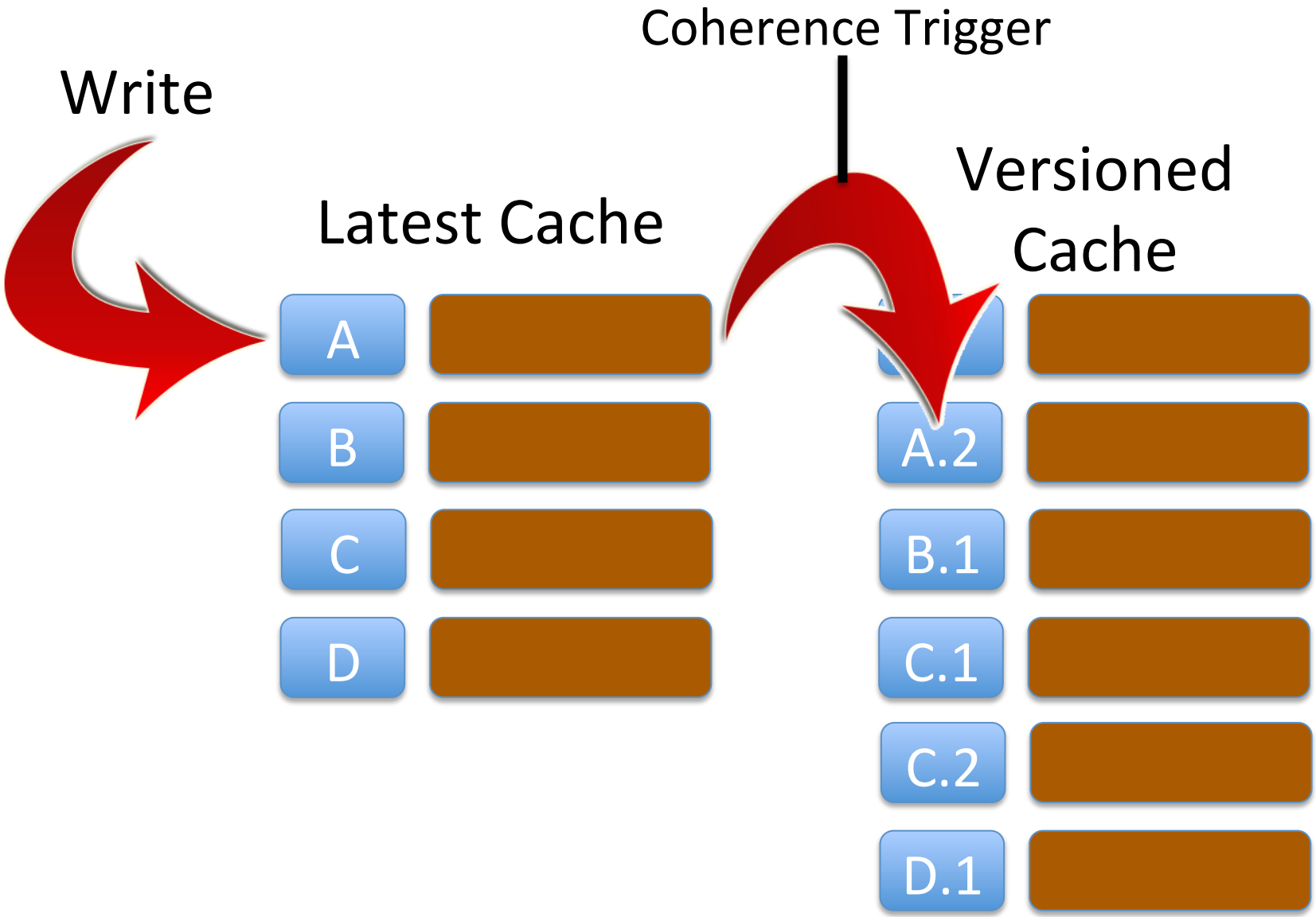
Running a Coherence Filter



Using Key-Based Access



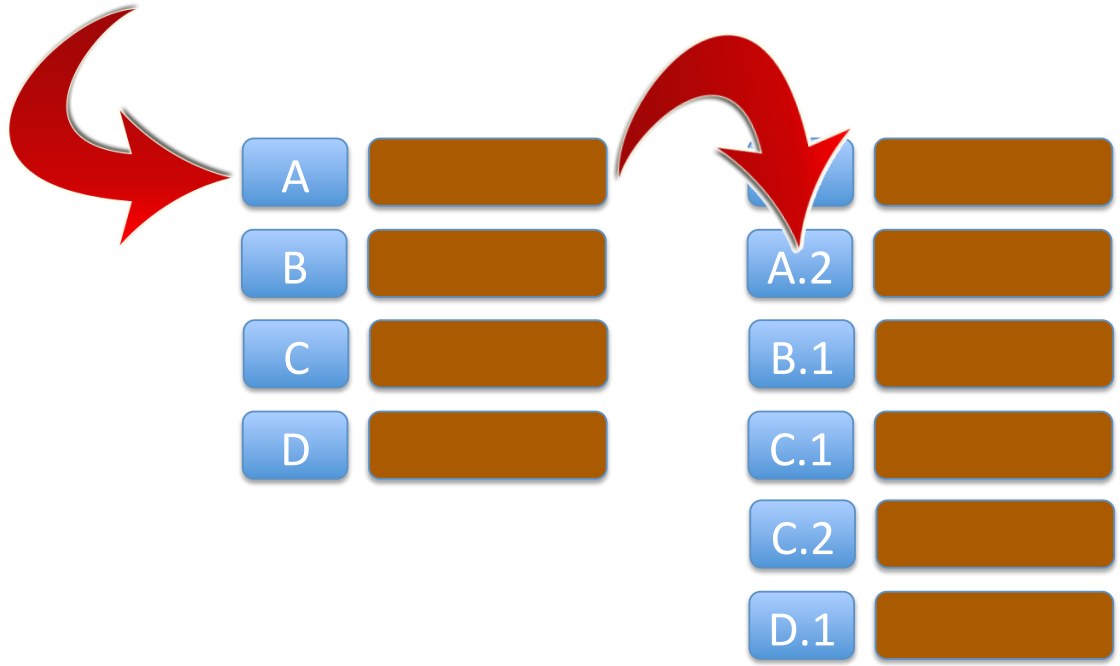
Latest / Versioned Pattern



Latest / Versioned Pattern

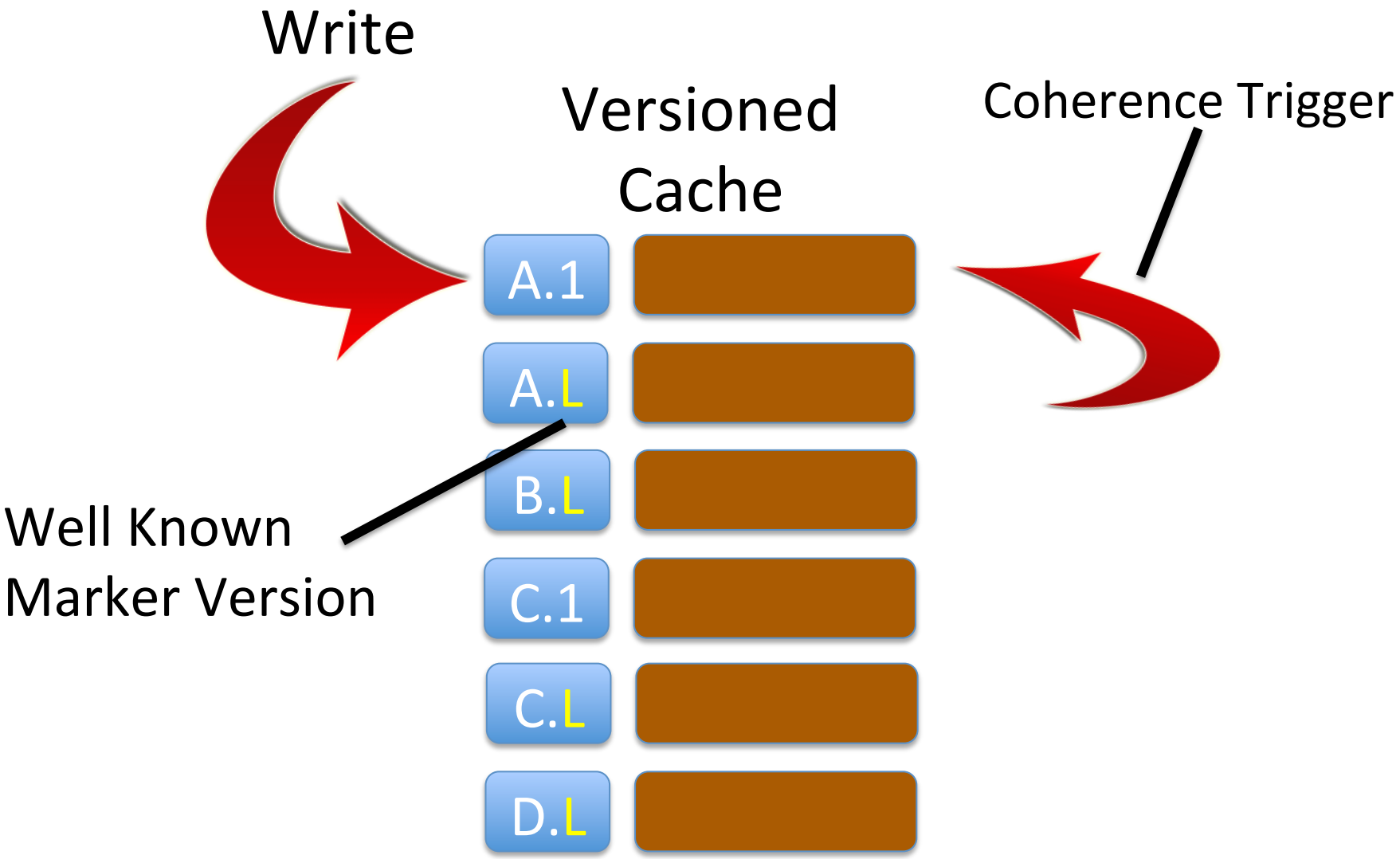
Latest Cache Key = [Business Key]

Versioned Cache Key = [Business Key][Version]



Suffers from data duplication

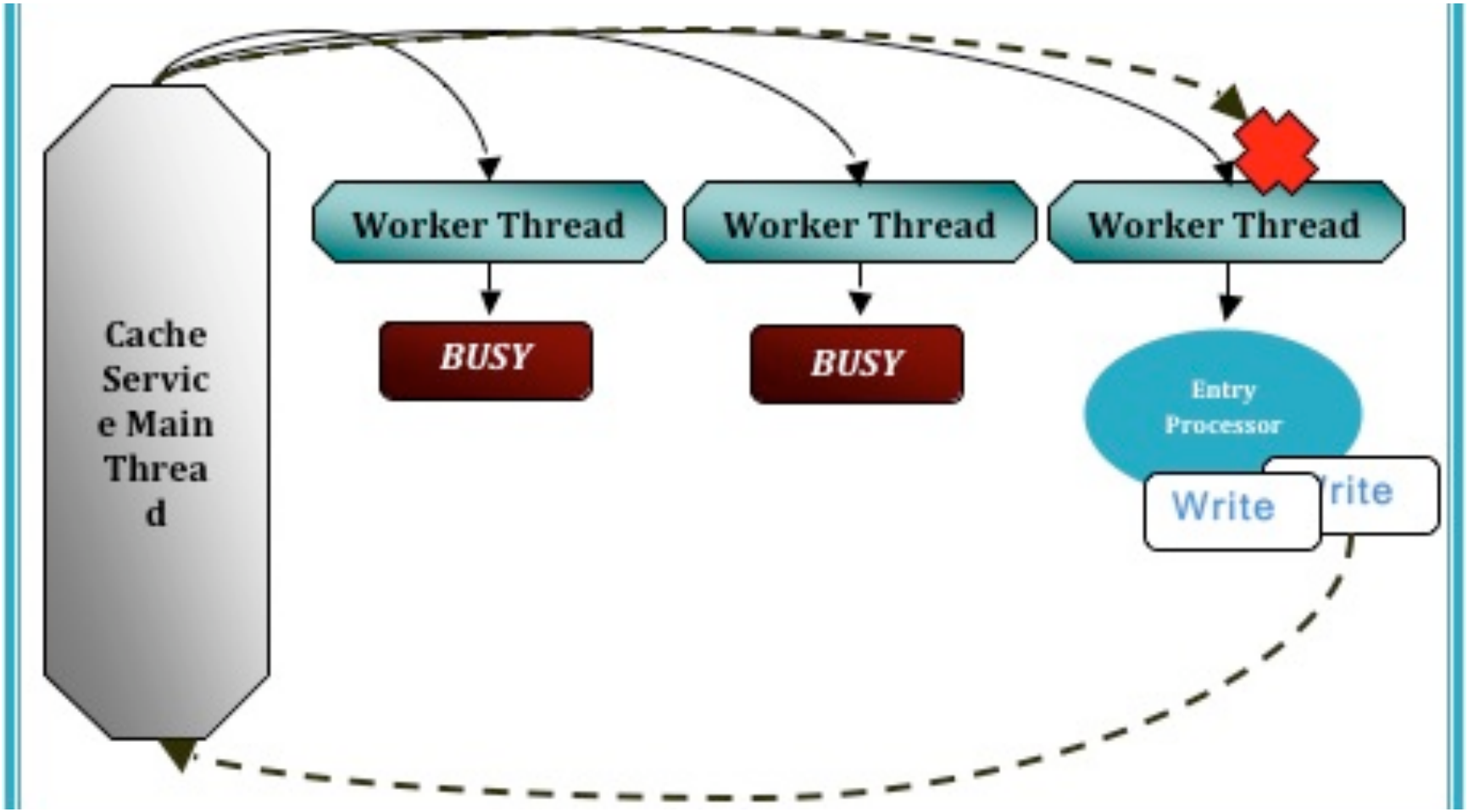
Latest Marker Pattern



However our trigger can’t use cache.put()

Why?

Need to consider the threading model



So we'll need to use the backing map directly

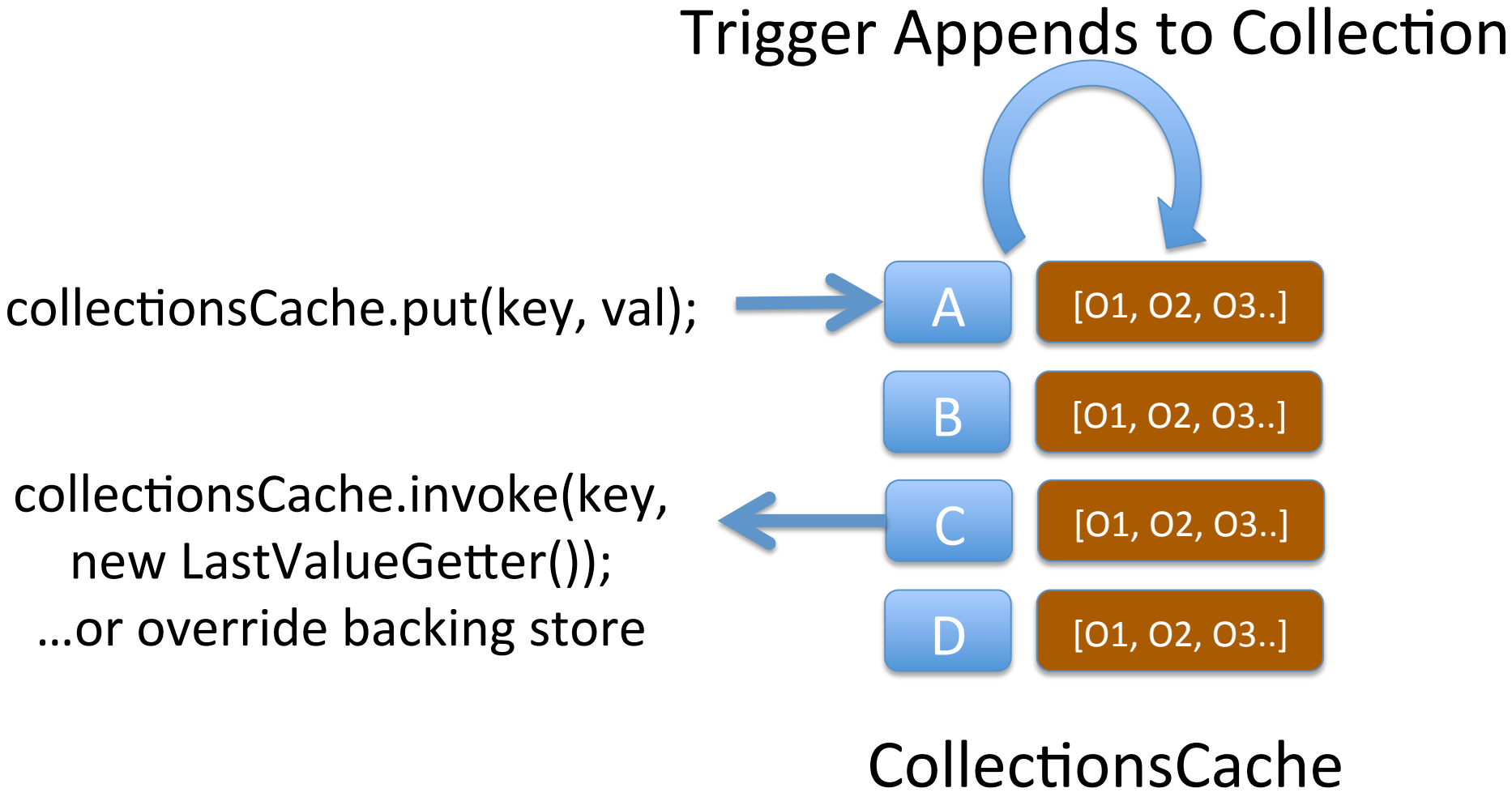
```
public void copyObjectToVersionedCacheAddingVersion(MapTrigger.Entry entry)
{
  MyValue value = (MyValue)entry.getValue();
  MyKey versionedKey = (MyKey)value.getKey();

  BinaryEntry binary = (BinaryEntry)entry;
  Binary binaryValue = binaryEntry.getBinaryValue();

  Map map = binary.getContext().getBackingMap("VersionedCacheName");
  map.put(toBinary(versionedKey), binaryValue);
}
```

A third approach

The Collections Cache



So we have 3 patterns for managing versioning whilst retaining key based access

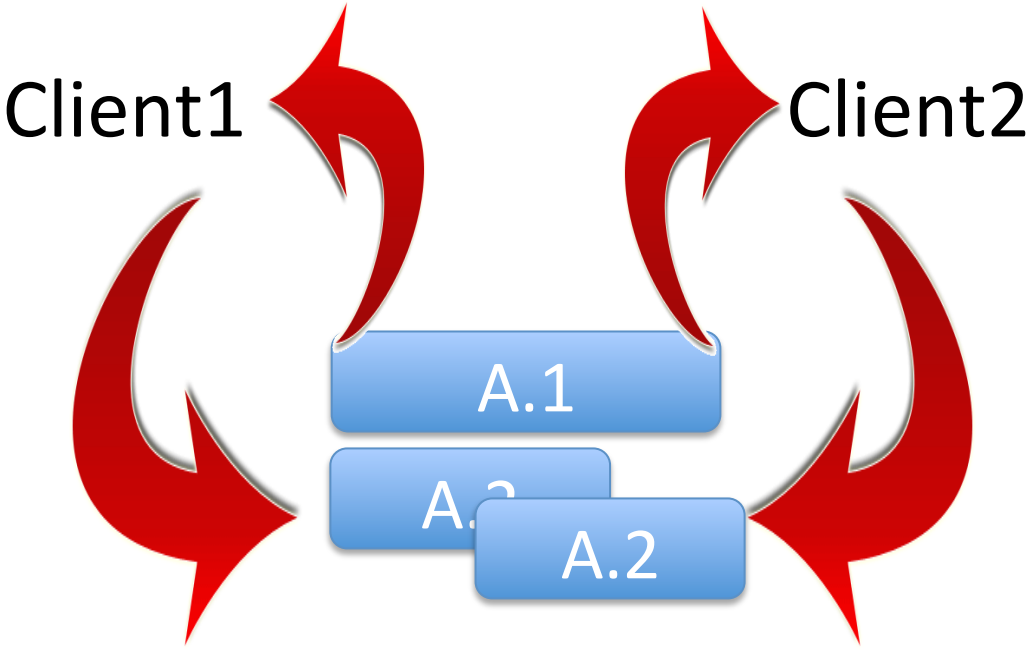
Using versioning to manage concurrent changes

Multi Version Concurrency Control (MVCC)

New Version = Old Version + 1 ??

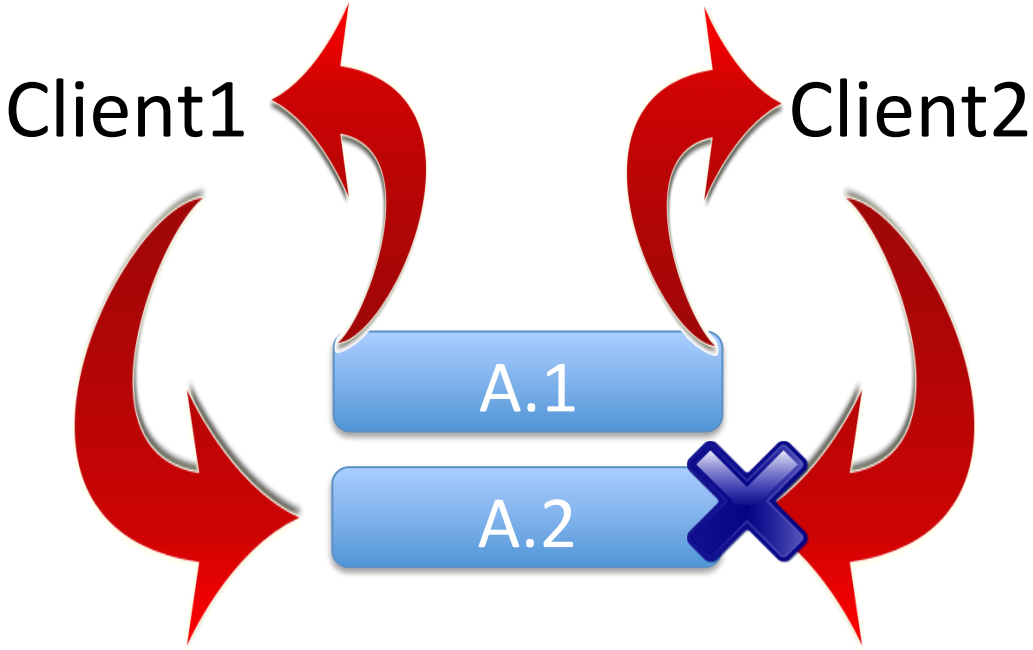
Concurrent Object Update

(2 Clients update the same object at the same time)



Concurrent Object Update

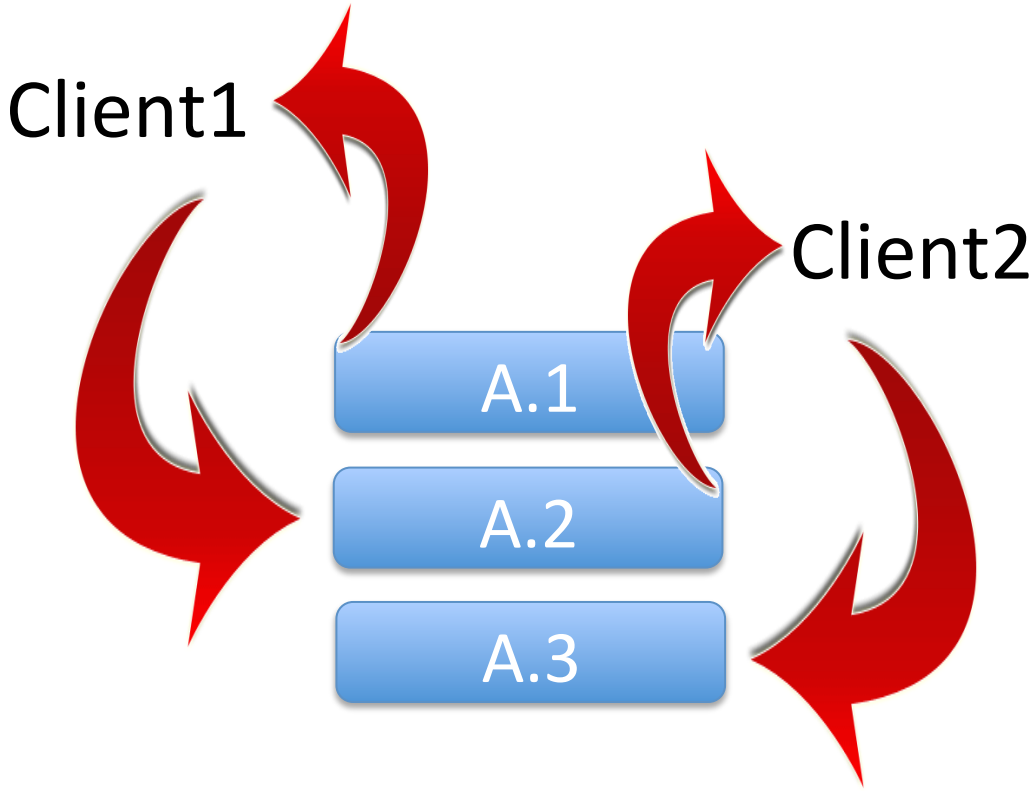
(Client2 fails to update dirty object)



Versioned
Cache

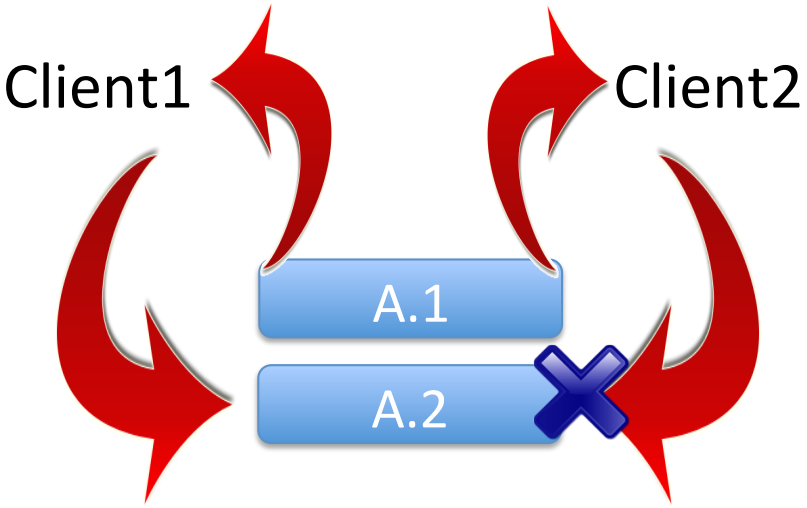
Concurrent Object Update

(Client 2 updates clean object)



Versioned Cache

So a concurrent update results in an error and must be retried.

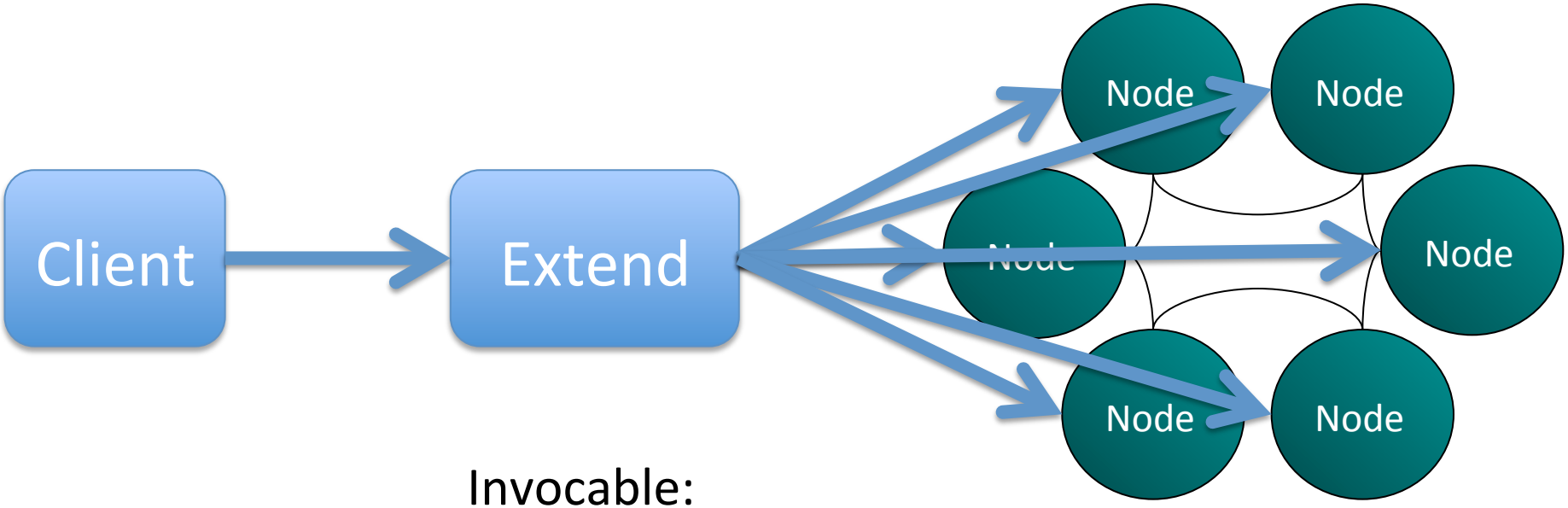


What’s going to happen if we are using putAll?

Reliable PutAll

We want putAll to tell us which objects failed the write process

Reliable PutAll



Invocable:

- Split keys by member
- Send appropriate values to each member
- Collect any exceptions returned

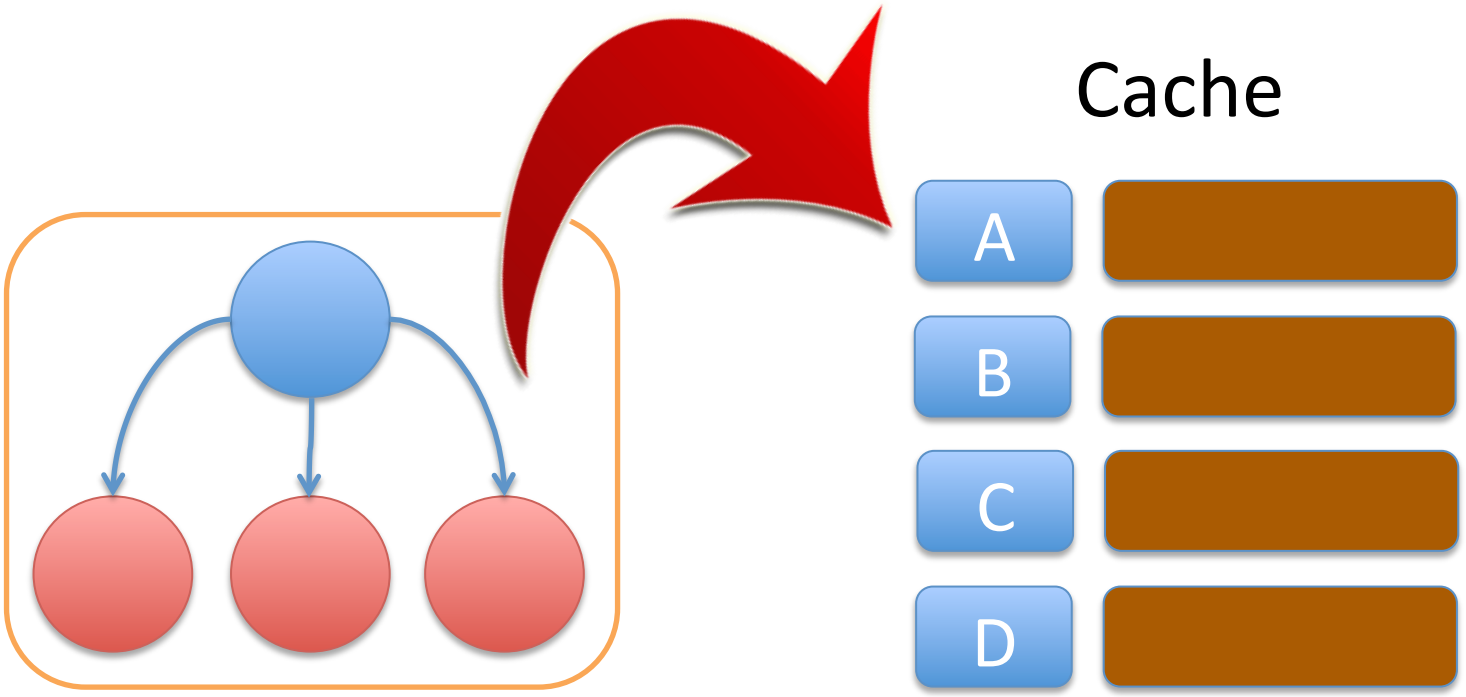
Invocable:

- Write entries to backing map (we use an EP for this)

This gives us a reliable mechanism for knowing what worked and what failed

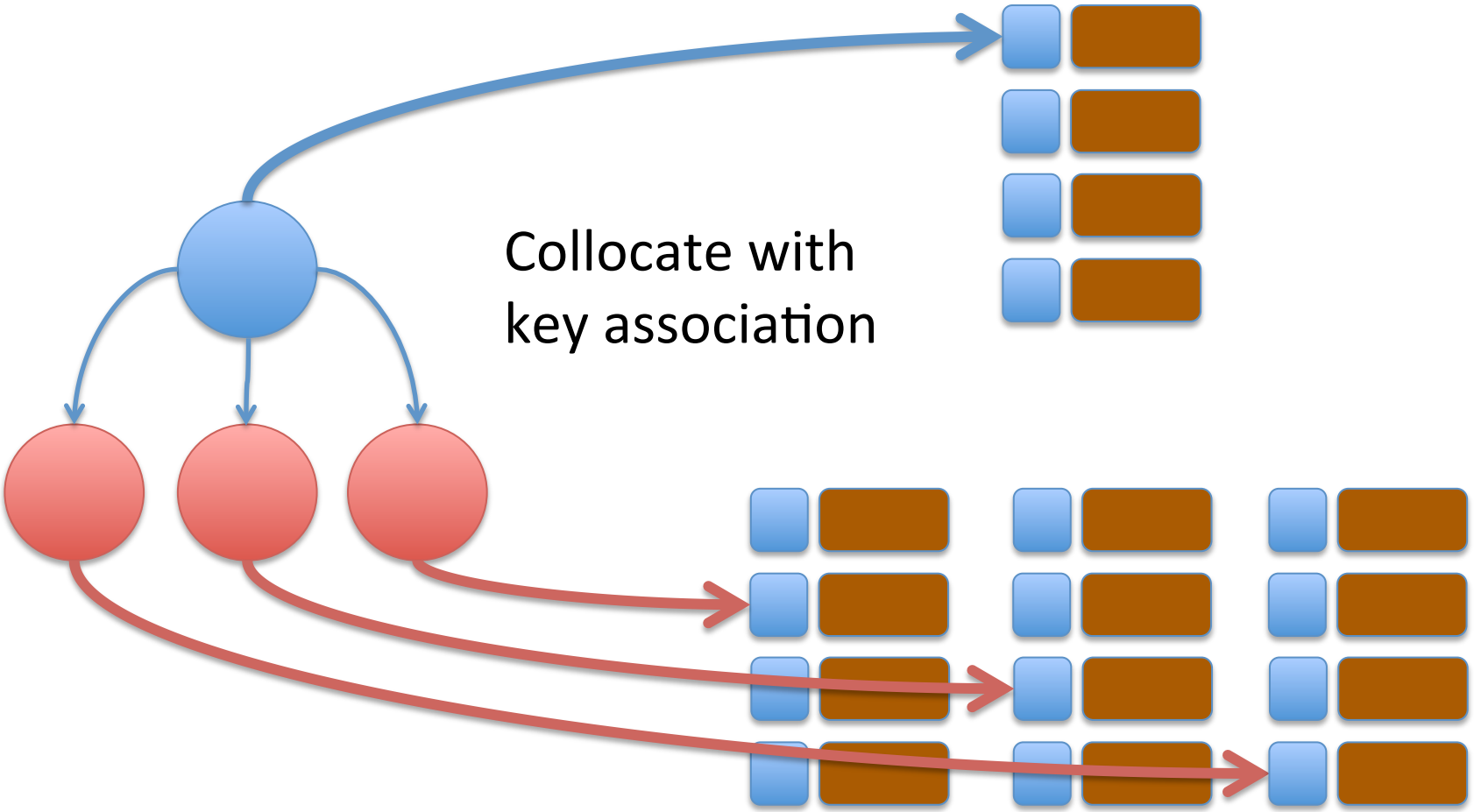
Synthesising Transactionality

The Fat Object Method

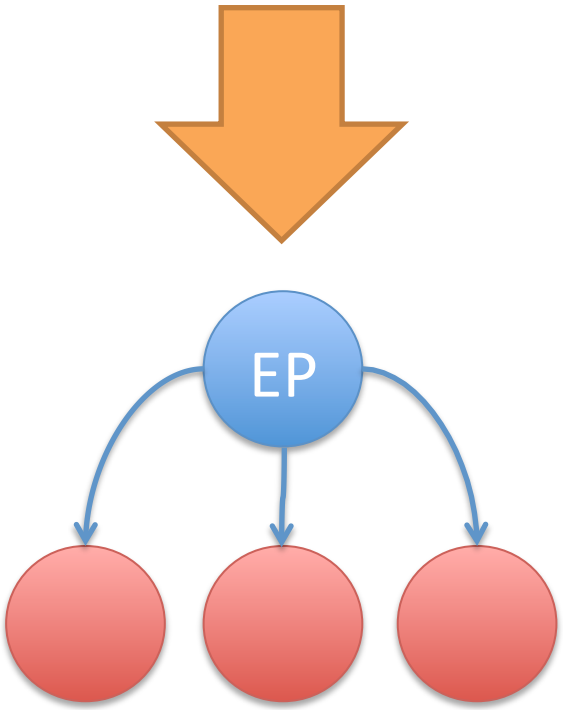


The Single Entry Point Method

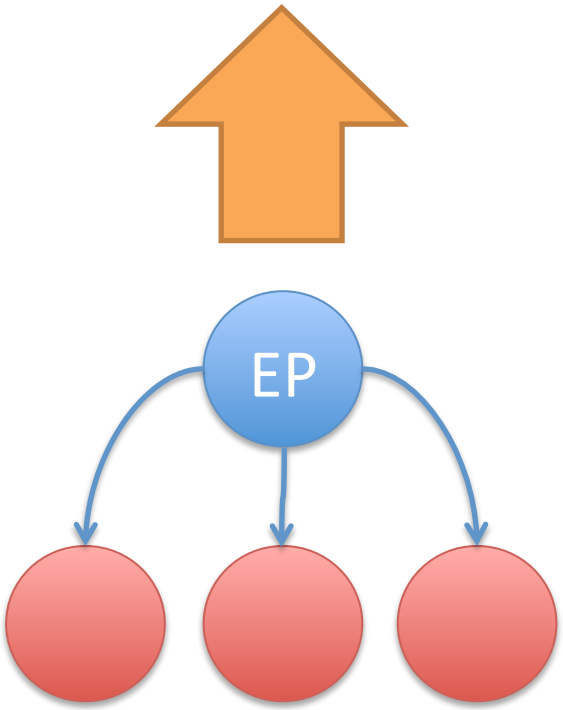
(objects are stored separately)



All writes synchronize on the primary object.

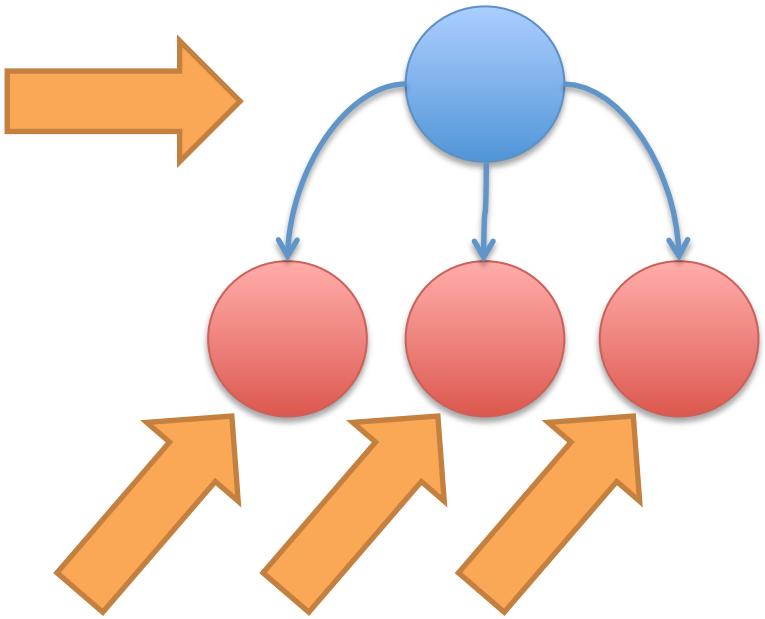


All reads synchronize on the primary object.



Writing Orphaned Objects

Write read entry point object last



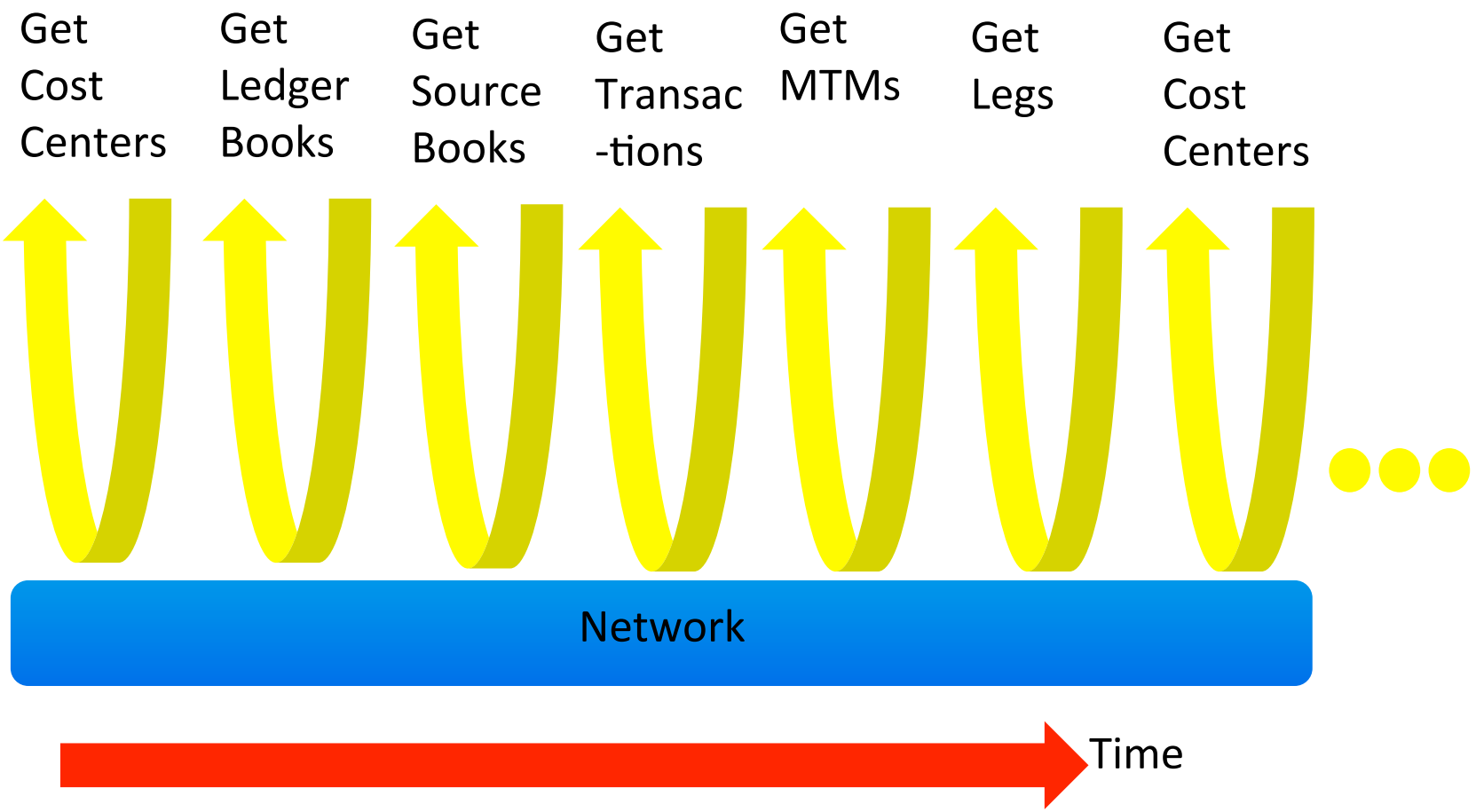
Write orphaned objects first

This mechanism is subtly flawed

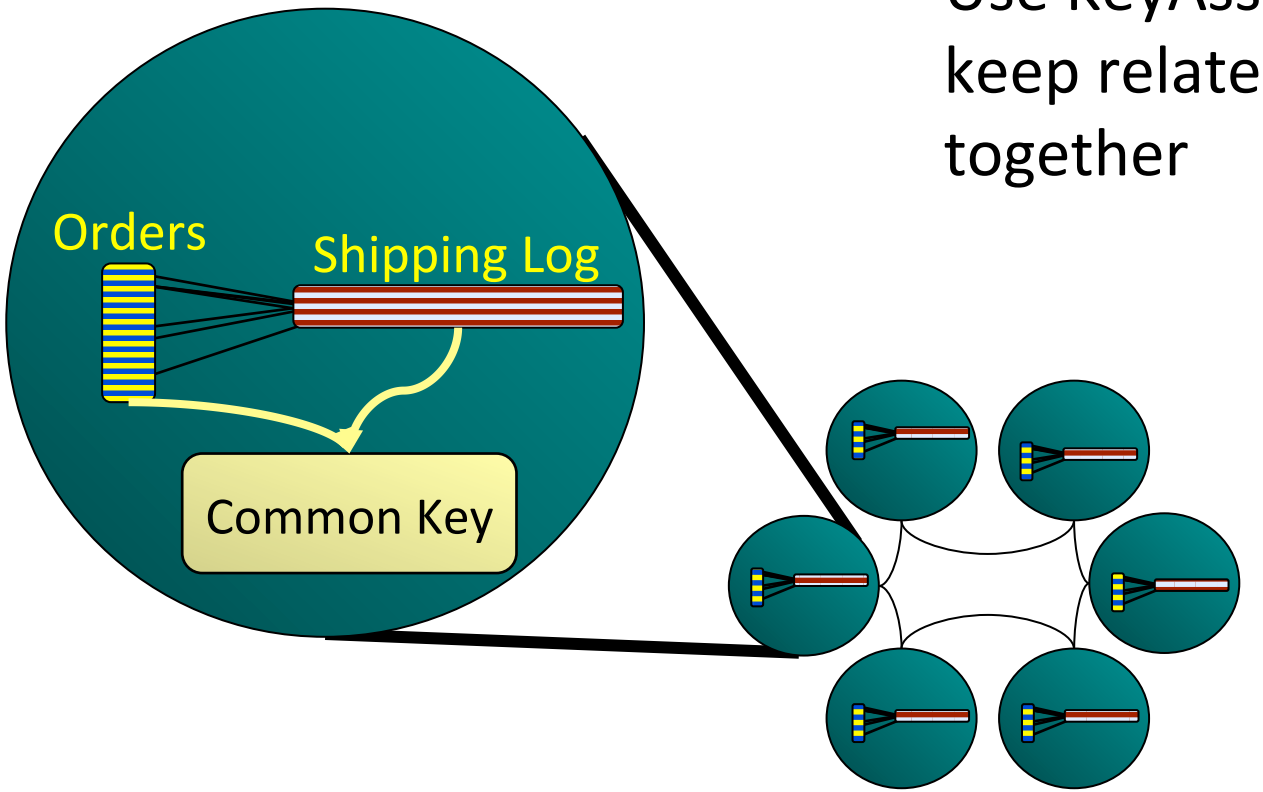
Reading several objects as an atomic unit

aka Joins

The trivial approach to joins

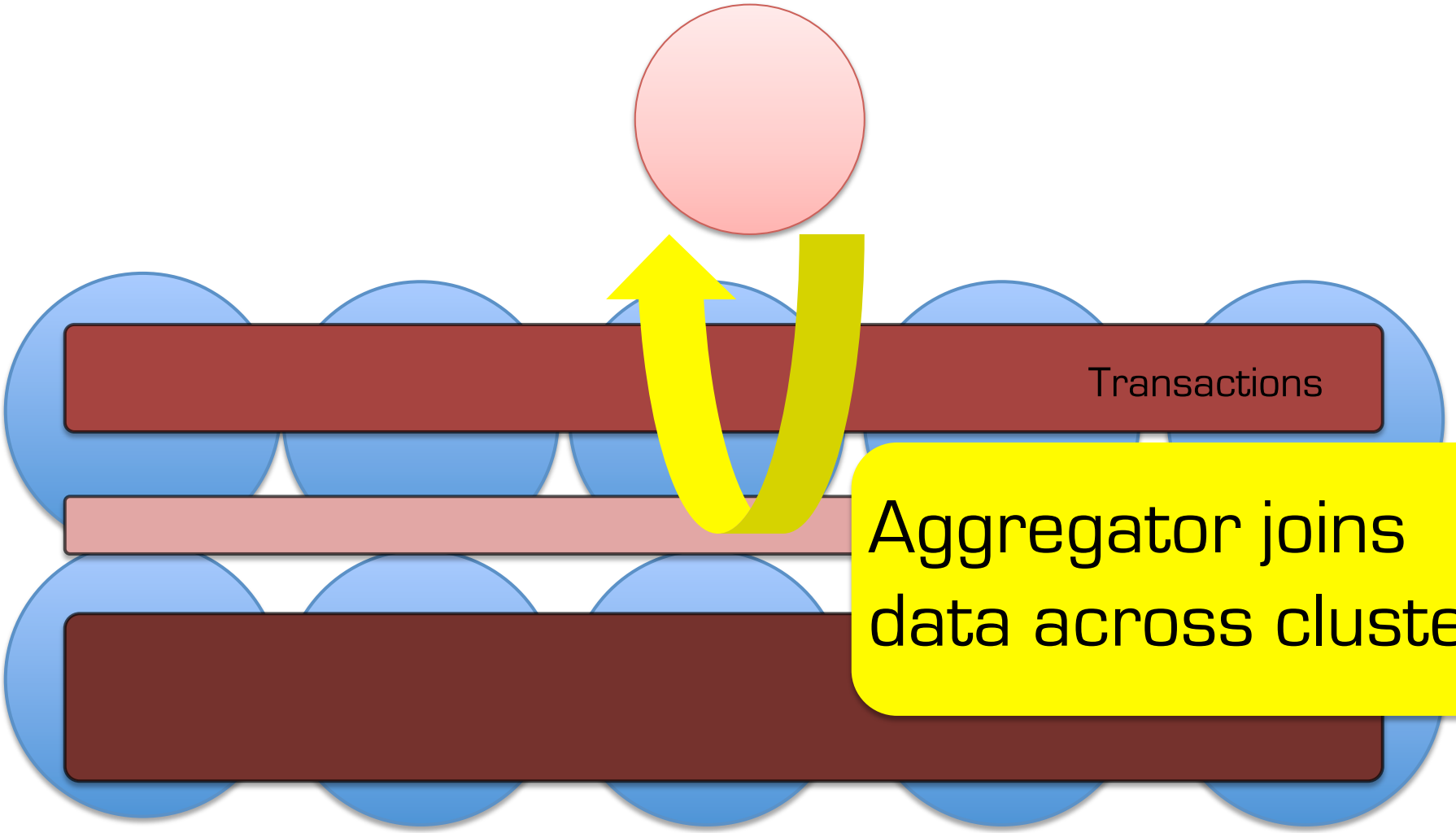


Server Side, Sharded Joins



Use KeyAssociation to keep related entities together

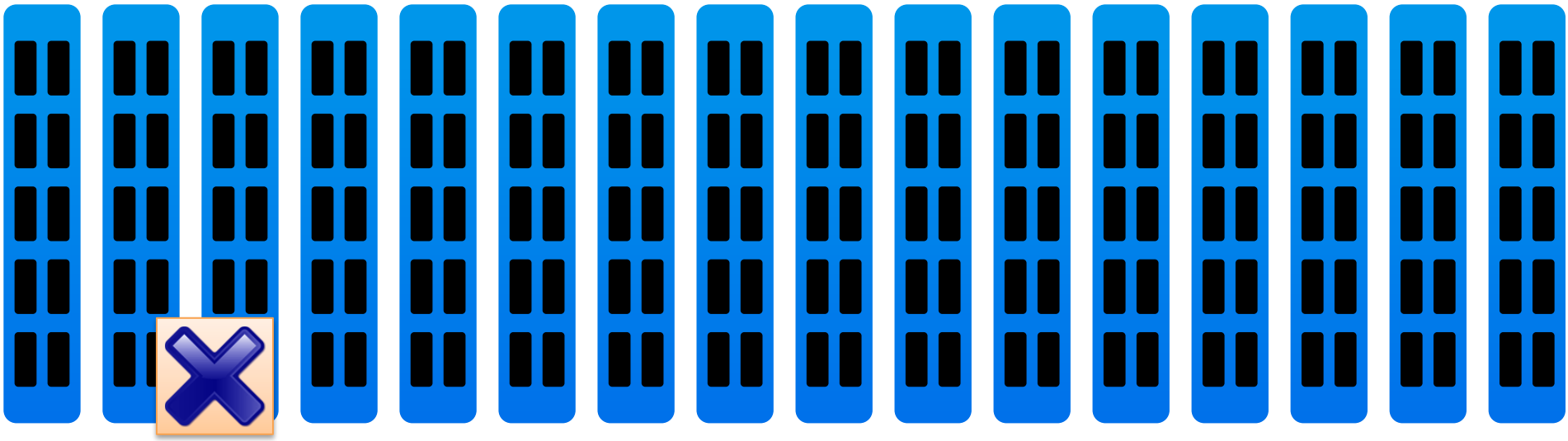
Server Side, Sharded Joins



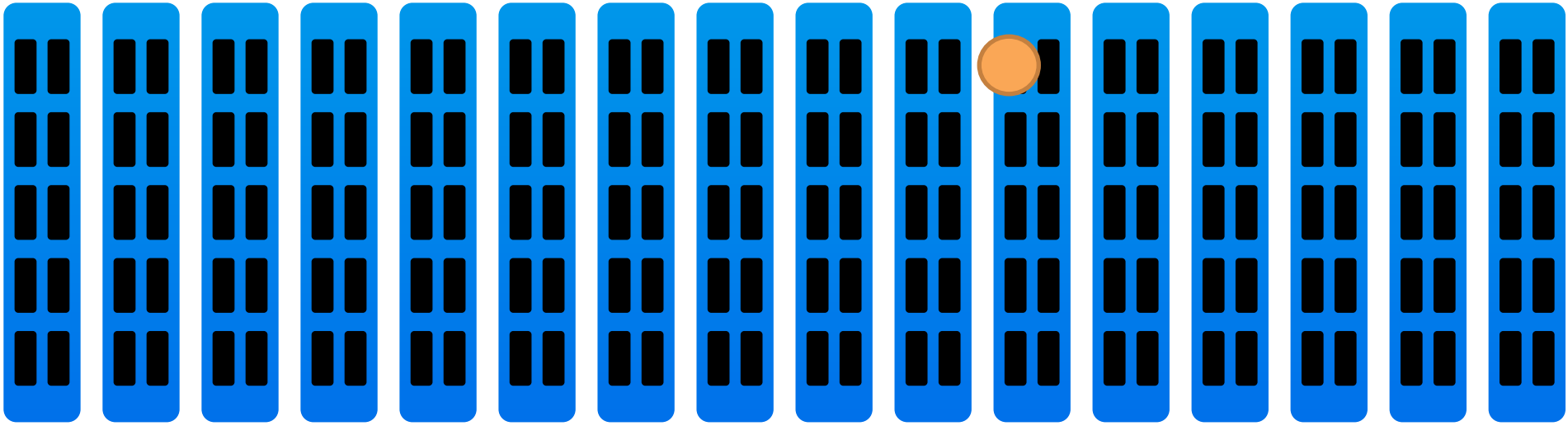
So we have a set of mechanisms for reading and writing groups of related objects.

Cluster Singleton Service

A service that automatically restarts after failure



A service that automatically restarts after failure



What is the cluster singleton good for

- Adding indexes
- Loading data
- Keeping data up to date
- Updating cluster time
- You can probably think of a bunch of others yourselves.

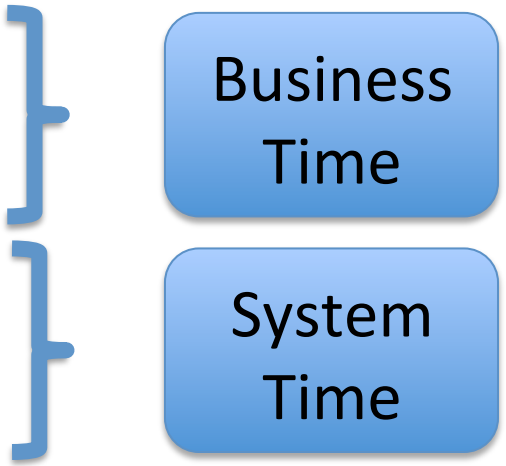
Code for Cluster Singleton

```
//run in a new thread on every Cache Server
while (true) {
    boolean gotLock = lockCache.lock("singletonLock", -1);
    if (gotLock) {
        //Start singletons
        wait();
    }
}
```

Implementing Consistent Views and Repeatable Queries

Bi-temporal

```
public interface MyBusinessObject{
   //data
   public Date getBusinessDate();
   public Date validFrom();
   public Date validTo();
}
```

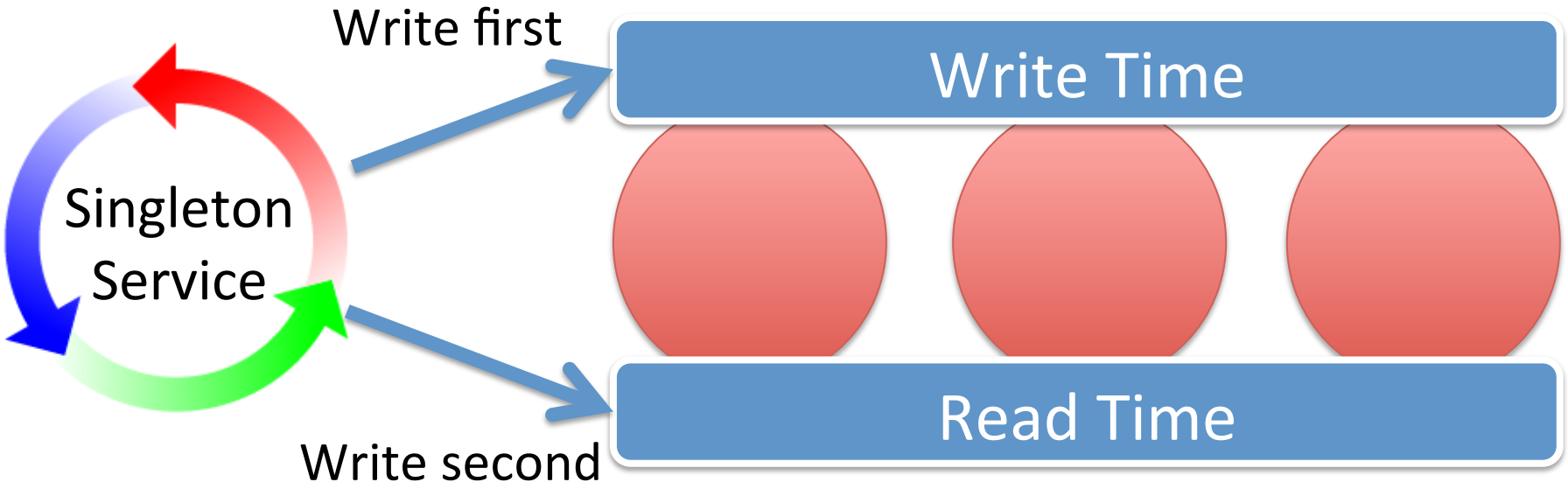


Where does the System Time come from?

You can’t use the System.currentTimeMillis() in a distributed environment!

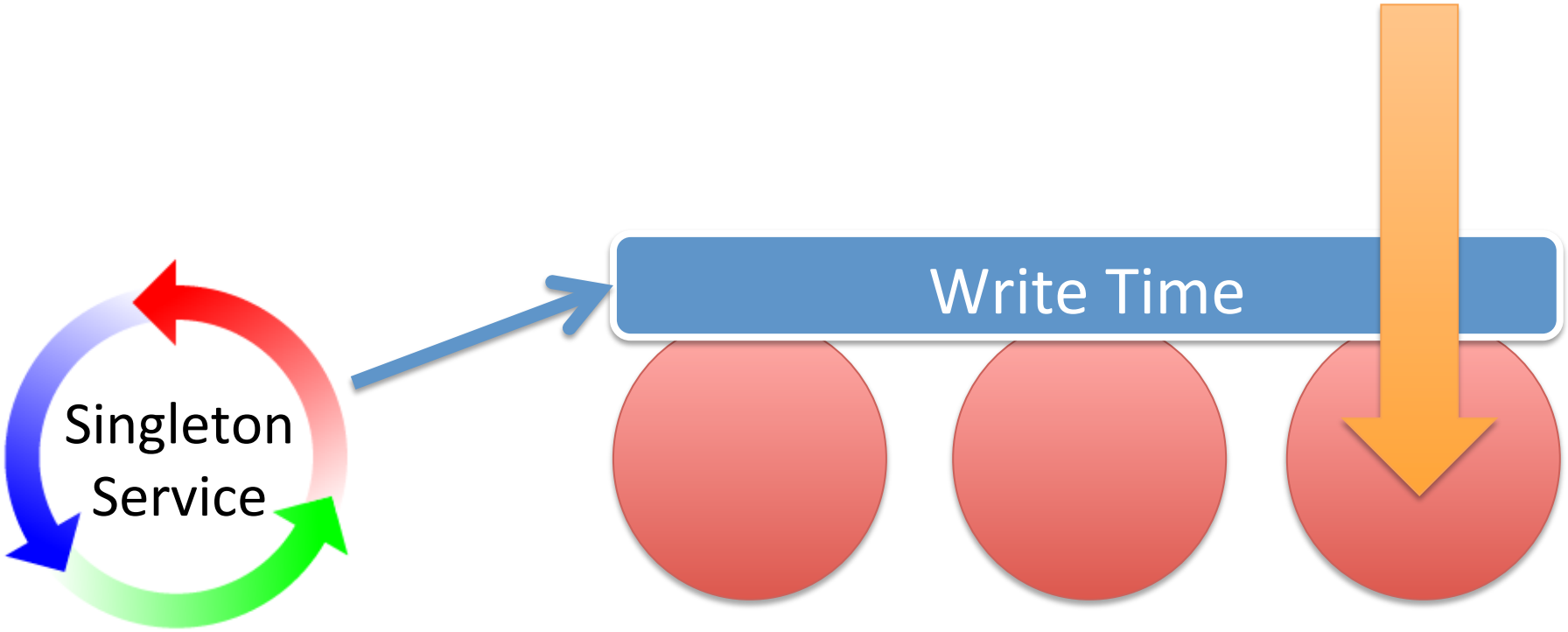
You need a cluster synchronised clock

Repeatable Time: A guaranteed Tick

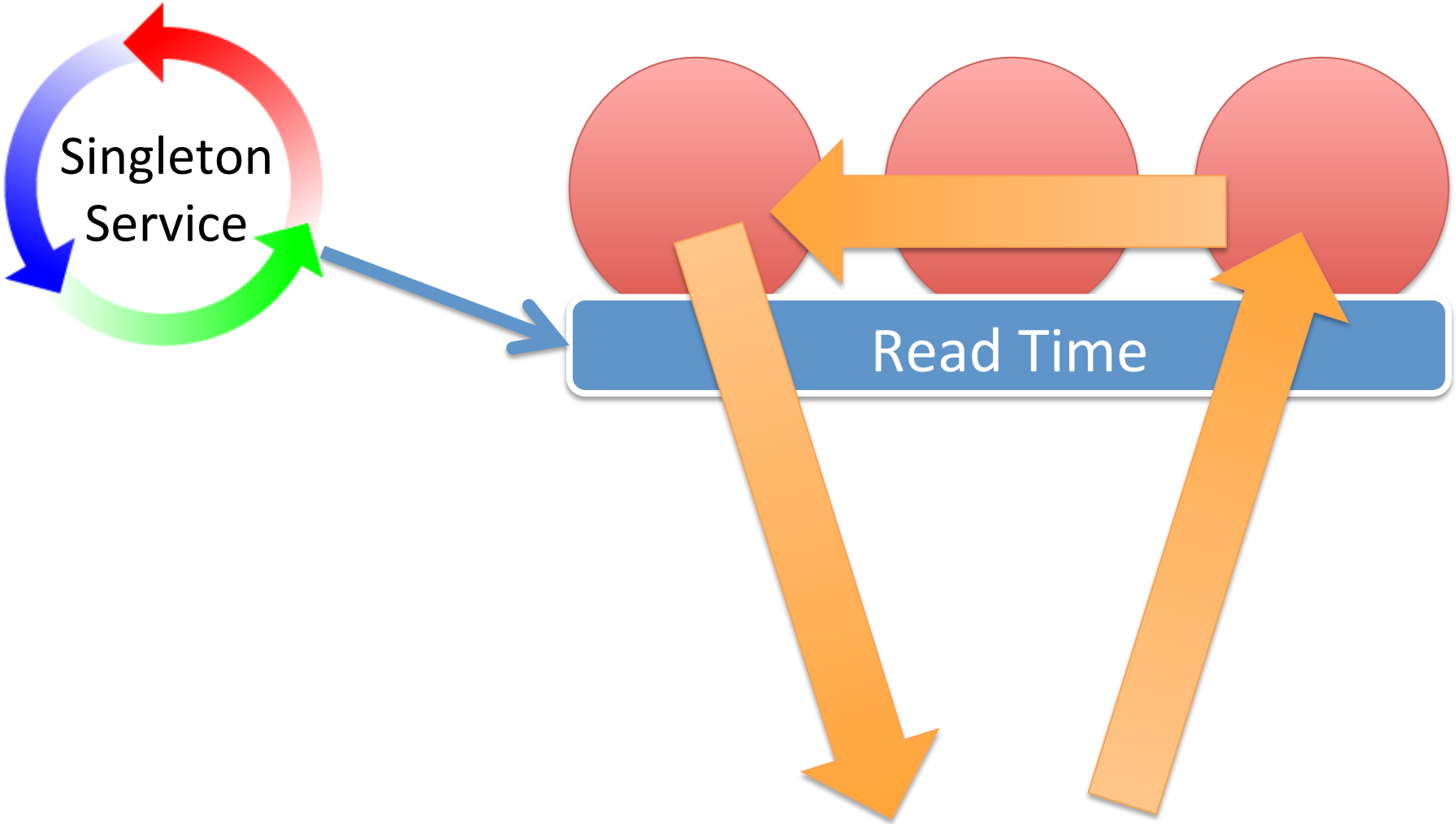


Replicated Caches
(pessimistic)

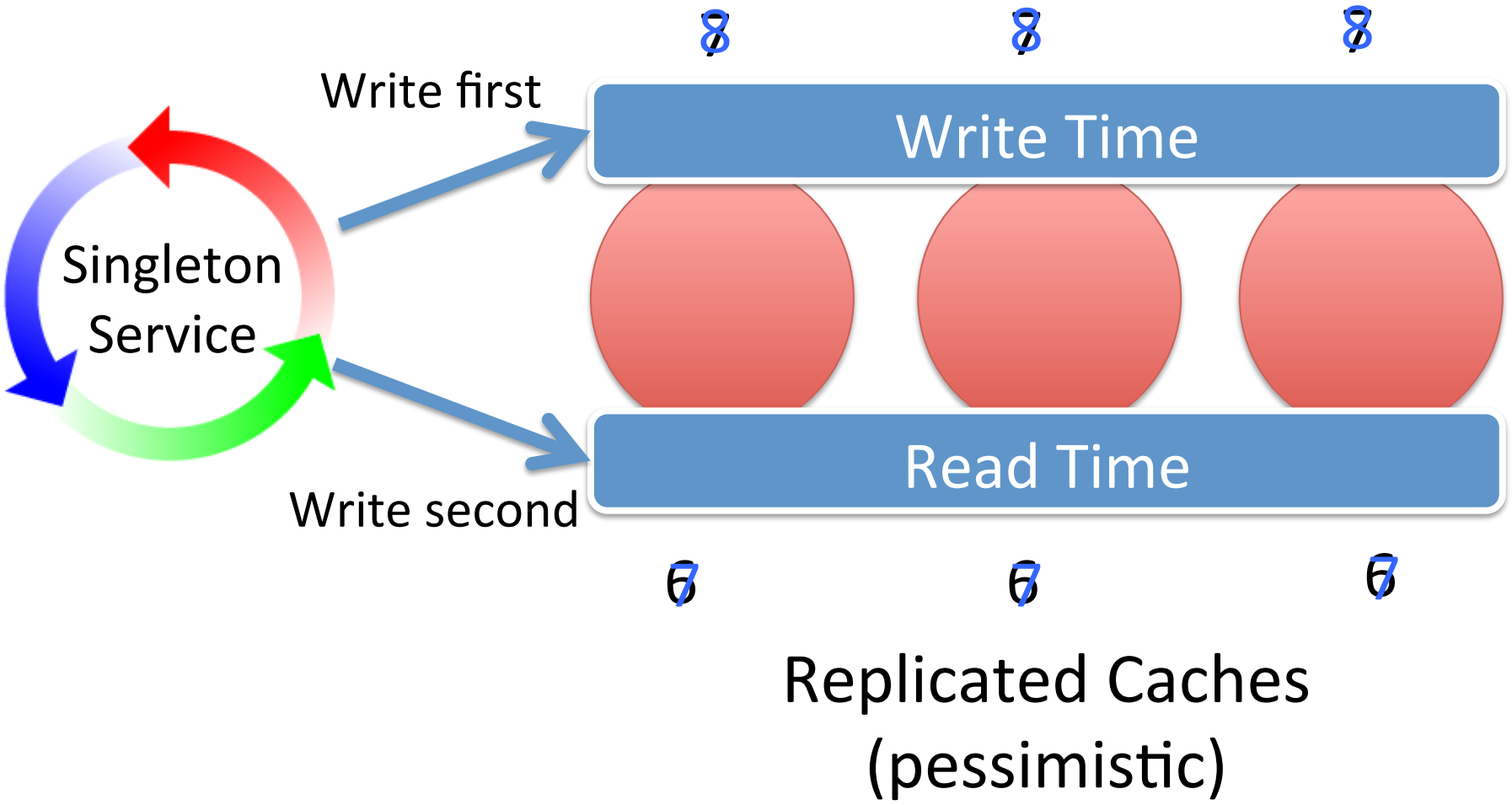
As we add objects we timestamp them with Write Time



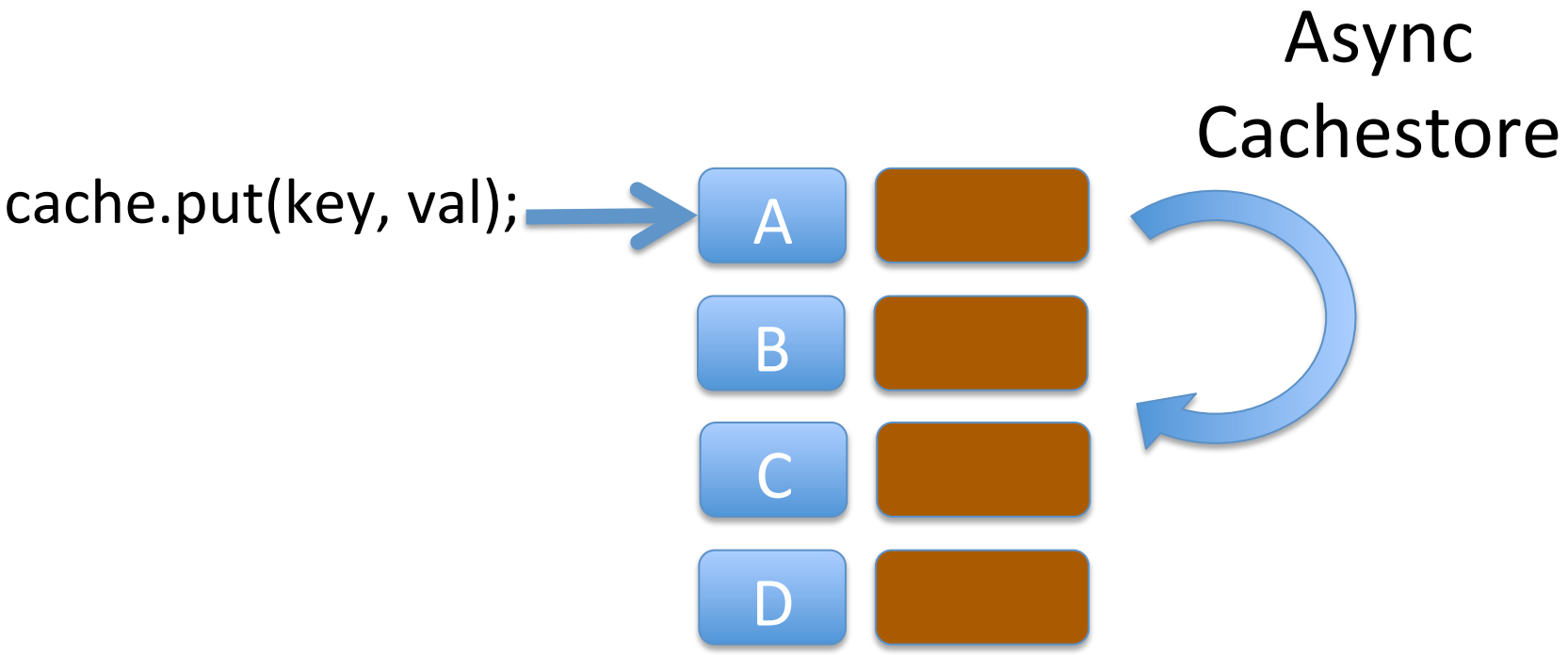
When we read objects we use Read Time



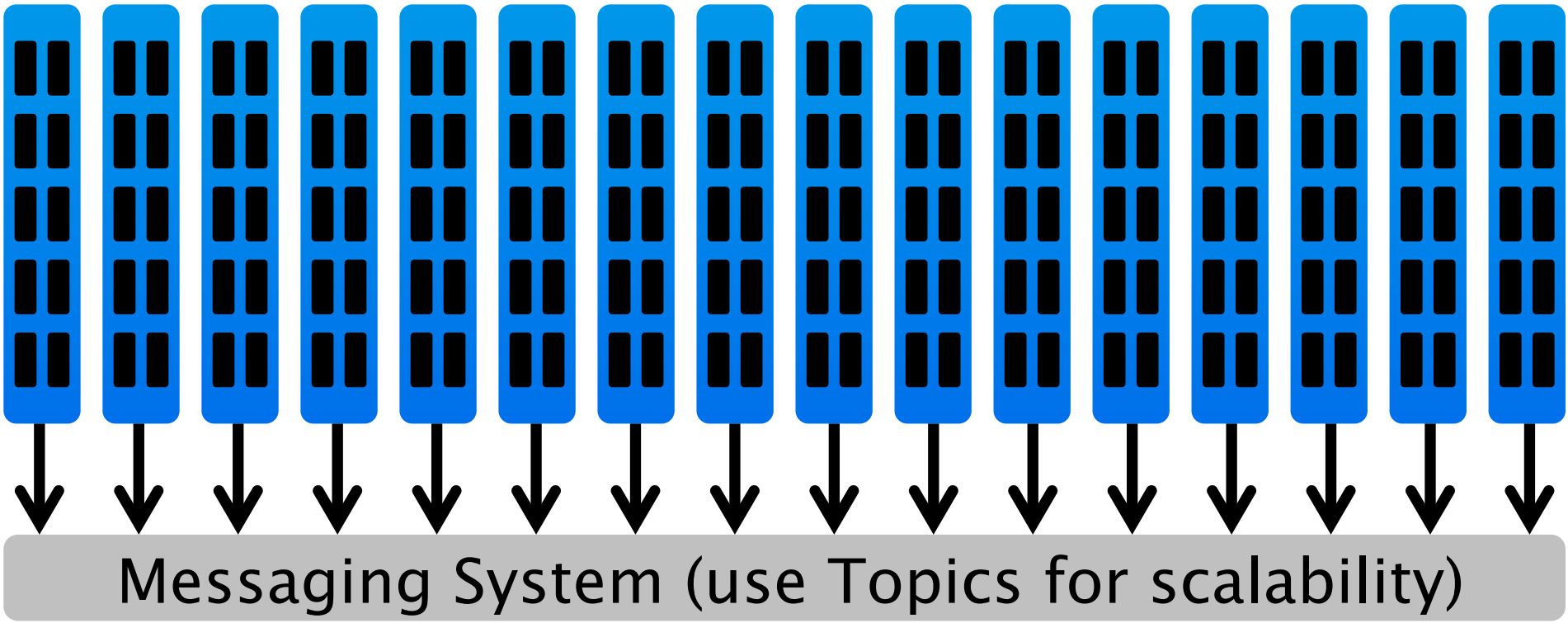
Repeatable Time: A guaranteed Tick



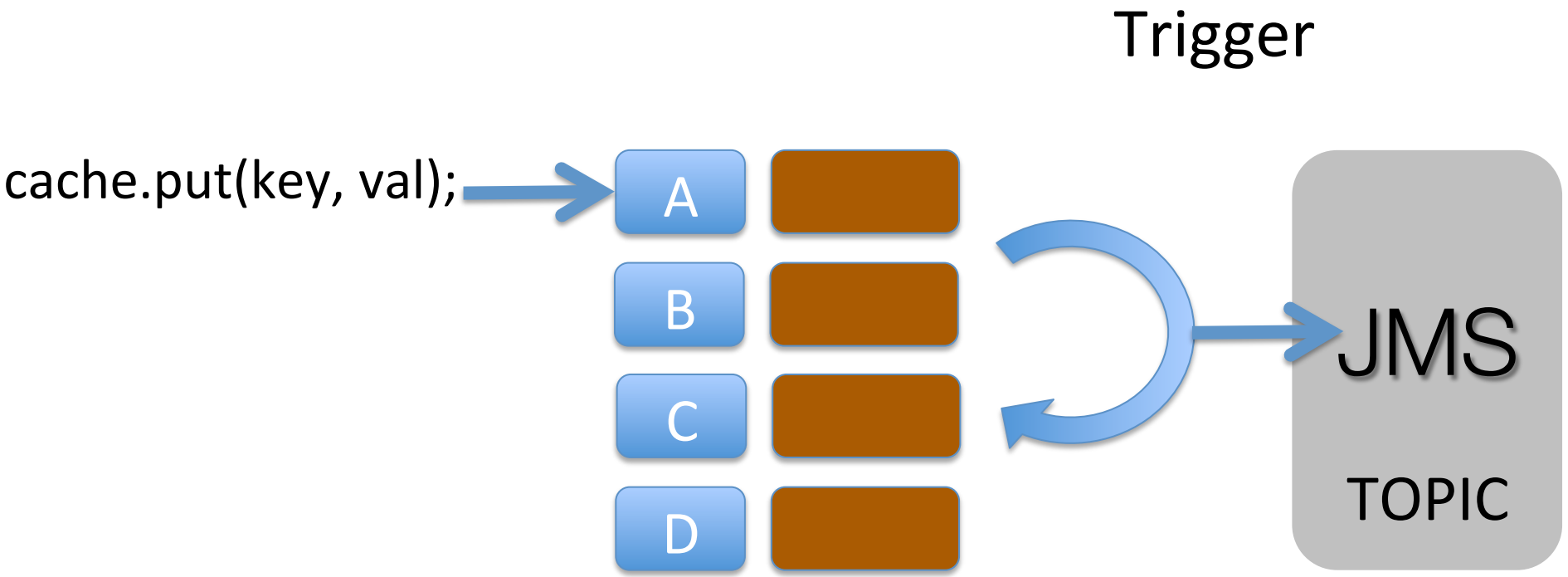
Event Based Processing



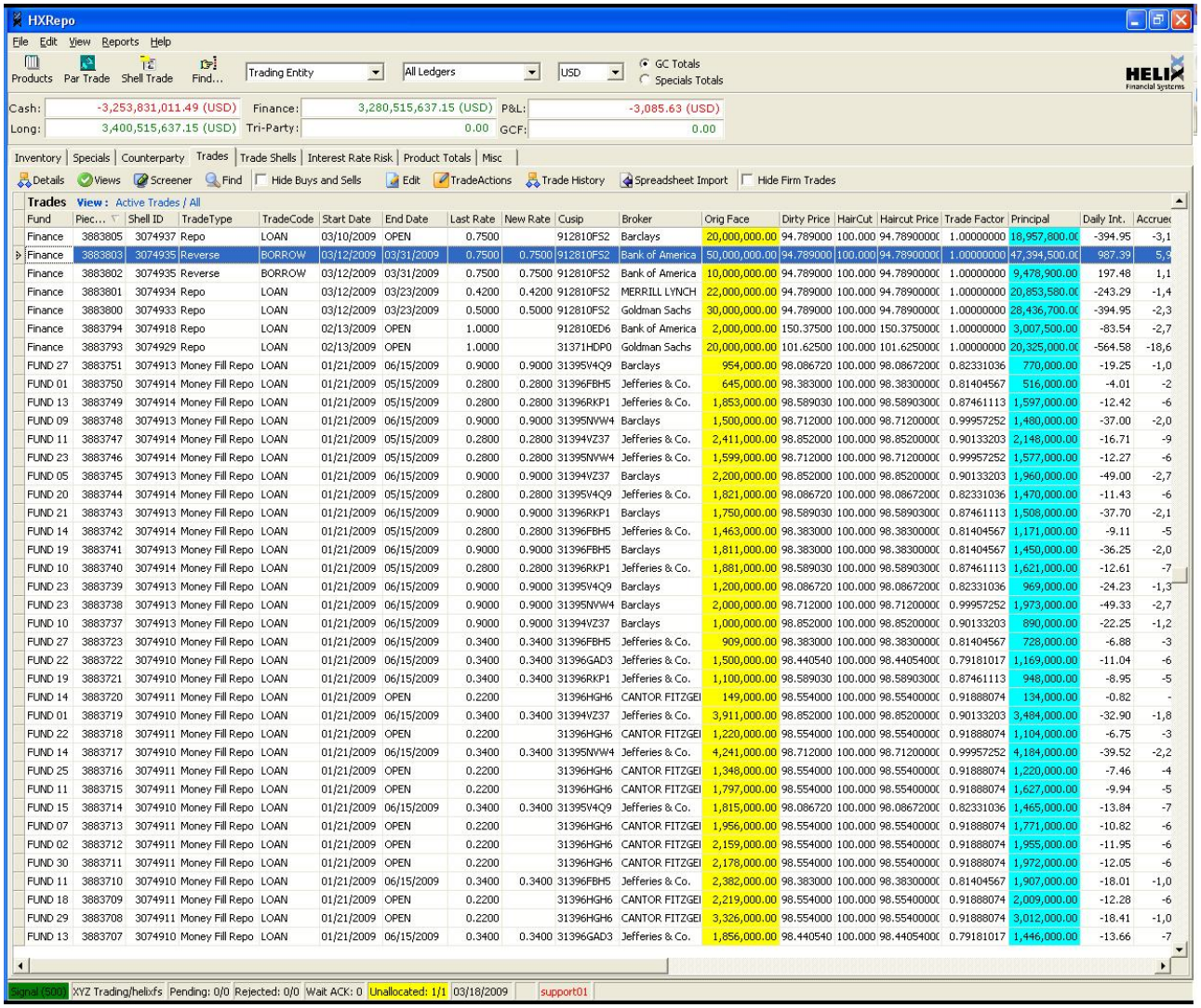
Messaging as a System of Record



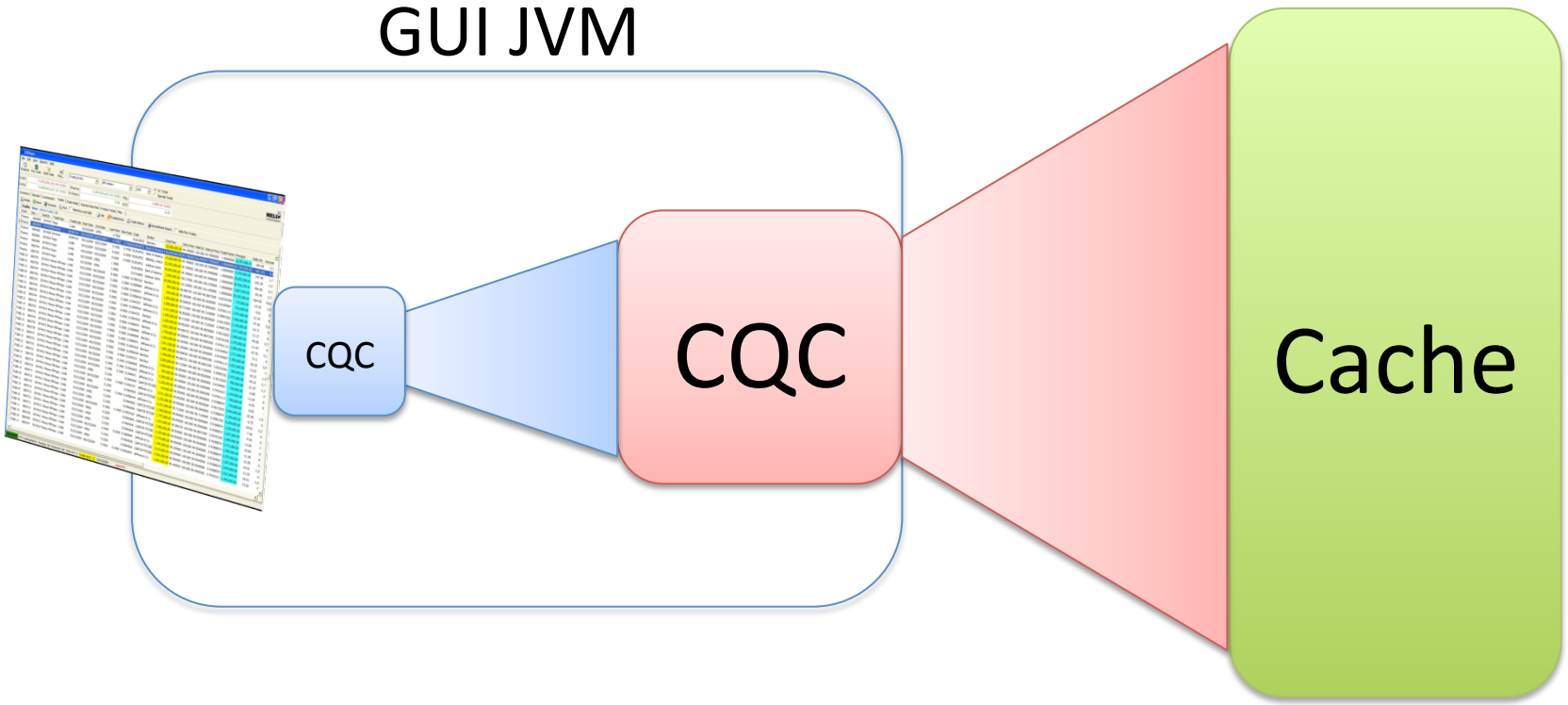
Messaging as a System of Record



Easy Grid Implementation in GUIs



CQCs on a CQC

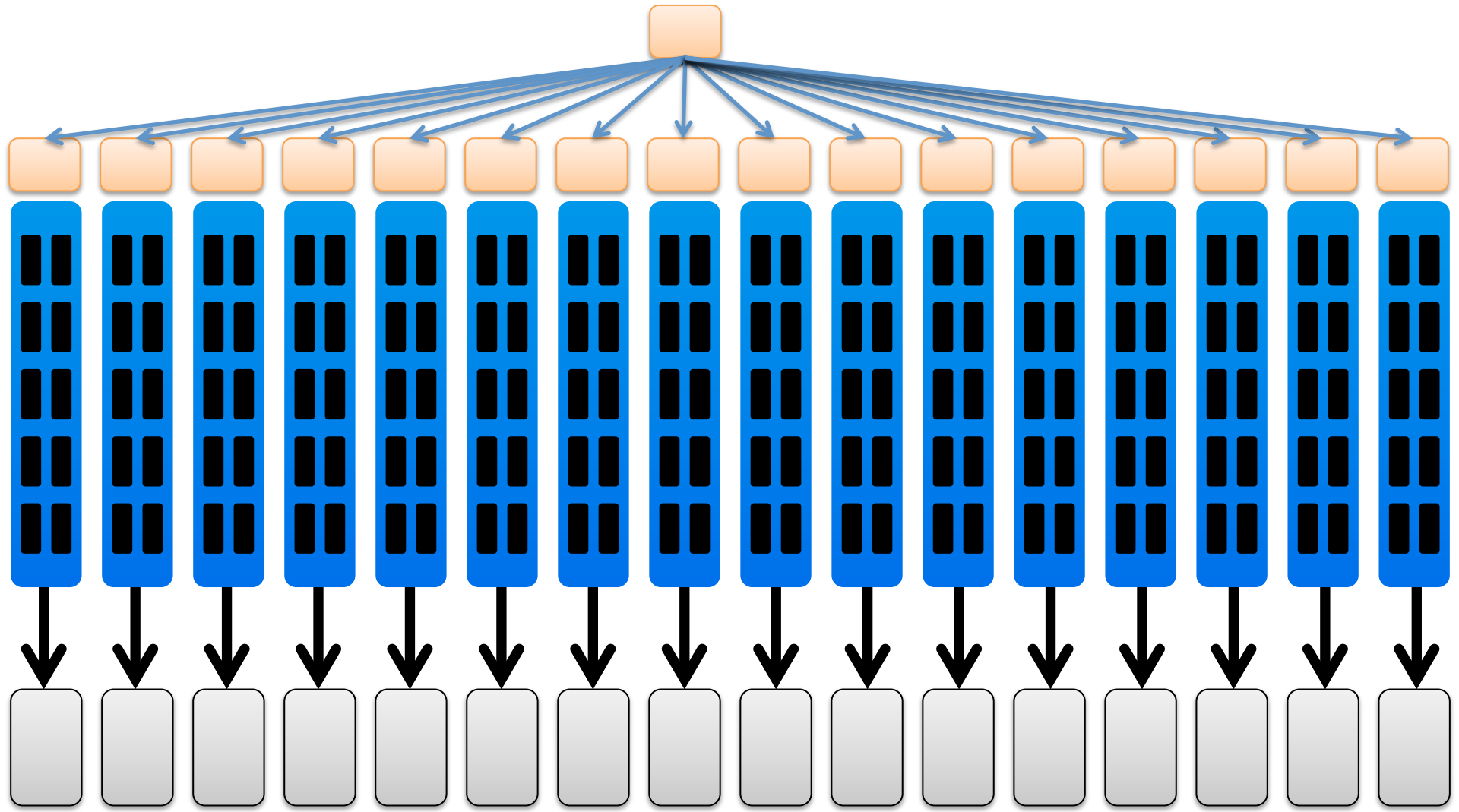


Define this in config

How do you release quickly to a Coherence cluster?

Rolling Restart?

Disk-Persist



Final Thoughts

Data is the most important commodity that you have

Keep it safe

Use a Partition Listener

Have Proactive Monitoring of Memory

Version your Objects

Thanks

Slides & related articles available at:

http://www.benstopford.com