Database backed Coherence cache

Tips, Tricks and Patterns

Alexey Ragozin

alexey.ragozin@gmail.com

May 2012

Power of read-write-backing-map

- Fetching data as needed
- Separation of concerns
- Gracefully handling concurrency
- Write-behind – removing DB from critical path
- Database operation bundling

… and challenges

- DB operations are order of magnitude slower
  - Less deterministic response time
  - Coherence thread pools issues
- How verify persistence with write behind?
- Data are written in DB in random order
- `read-write-backing-map` and expiry

TIPS

BinaryEntryStore, did you know?

`BinaryEntryStore` – an alternative to `CacheLoader` / `CacheStore` interface.

Works with `BinaryEntry` instead of objects.

- You can access binary key and value
  - Skip deserialization, if binary is enough
- You can access previous version of value
  - Distinguish inserts vs. updates
  - Find which fields were cached
- You cannot set entry TTL in cache loader ☹

When storeAll(...) is called?

- `cache.getAll(...)`
  - loadAll(...) will be called with partition granularity *(since Coherence 3.7)*
- `cache.putAll(...)`
  - write-behind scheme will use storeAll(...)
  - write-through scheme will use store(...) *(this could be really slow)*

When storeAll(...) is called?

- `cache.invokeAll(...)/aggregate(...)`
  - calling get() on entry will invoke load(...)
    (if entry is not cached yet)
  - calling set() on entry will invoke put(...)
    (in case of write-through)
  - you can check entry.isPresent() to avoid needless read-through
  - Coherence will never use bulk cache store operations for aggregators and entry processors

Warming up aggregator

```
public static void preloadValuesViaReadThrough(Set<BinaryEntry> entries) {
 CacheMap backingMap = null;
 Set<Object> keys = new HashSet<Object>();
 for (BinaryEntry entry : entries) {
  if (backingMap == null) {
   backingMap = (CacheMap) entry.getBackingMapContext().getBackingMap();
  }
  if (!entry.isPresent()) {
   keys.add(entry.getBinaryKey());
  }
 }
 backingMap.getAll(keys);
}
```

Code above will force all entries for working set to be preloaded using bulk loadAll(…).

Call it before processing entries.

Why load(...) is called on write?

Case:

- Entry processor is called on set of entries which is not in cache and assigns values to them

Question:

- Why read-through is triggered?

Answer:

- `BinaryEntry.setValue(Object)` returns old value
- Use `BinaryEntry.setValue(Object, boolean)`

Bulk put with write through

You can use same trick for updates.

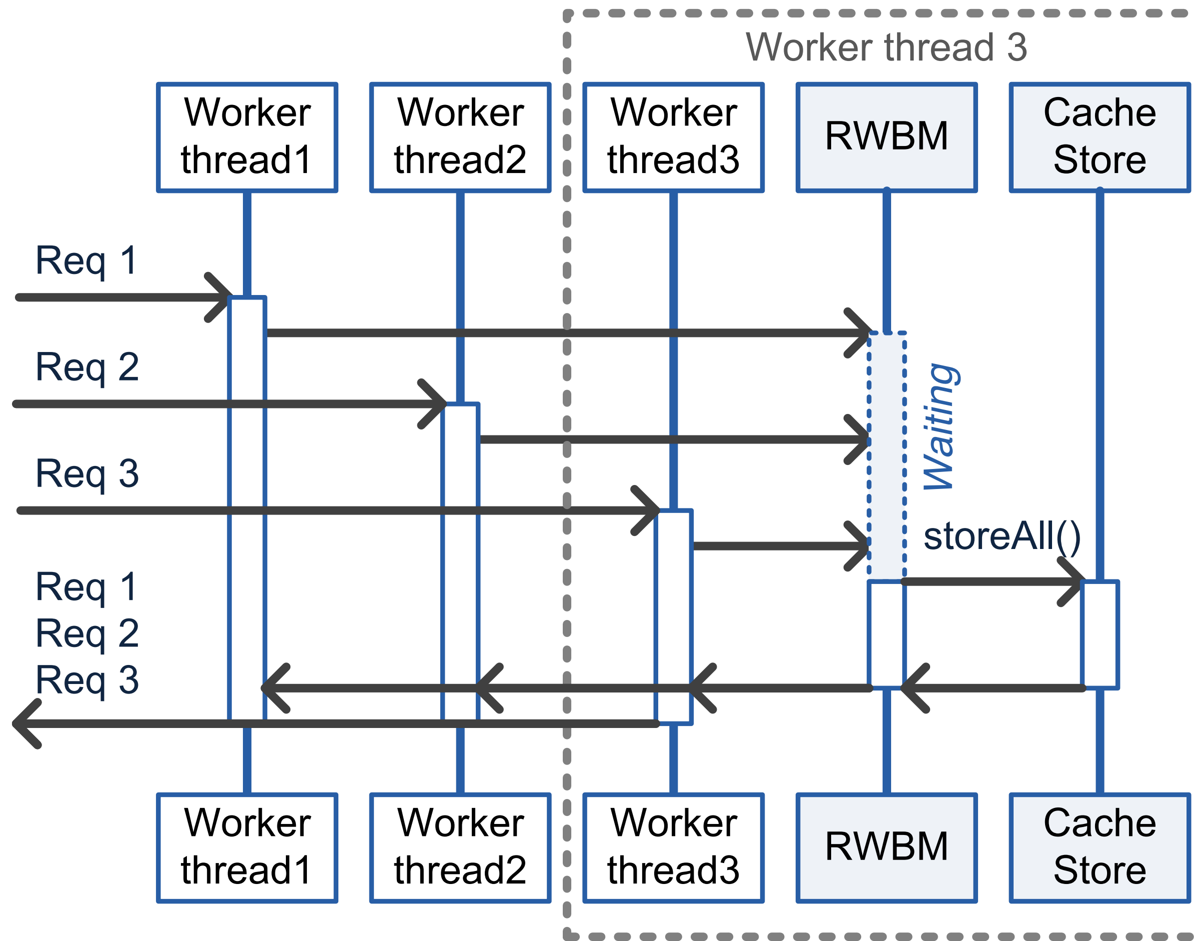
1. Pack your values in entry processor.
2. In entry processor obtain backing map reference.
3. Call `putAll(…)` on backing map.

Be careful !!!

- You should only put key for partition entry processor was called for.
- Backing map accepts serialized objects.

Hack alert

Using operation bundling



Using operation bundling

storeAll(…) with N keys could be called if

- You have at least N concurrent operations
- You have at least N threads in worker pool

```
<cachestore-scheme>
    <operation-bundling>
        <bundle-confing>
            <operation-name>store</operation-name>
            <delay-millis>5</delay-millis>
            <thread-threshold>4</thread-threshold>
        </bundle-config>
    </operation-bundling>
</cachestore-scheme>
```

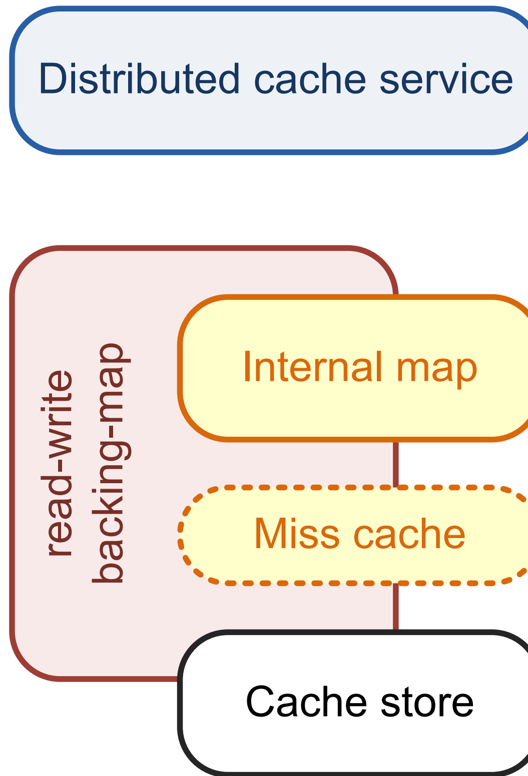
Checking STORE decoration

- Configure cache as “write-behind”
- Put data
- Wait until, STORE decoration become TRUE
  *(actually it will switch from FALSE to **`null`**)*

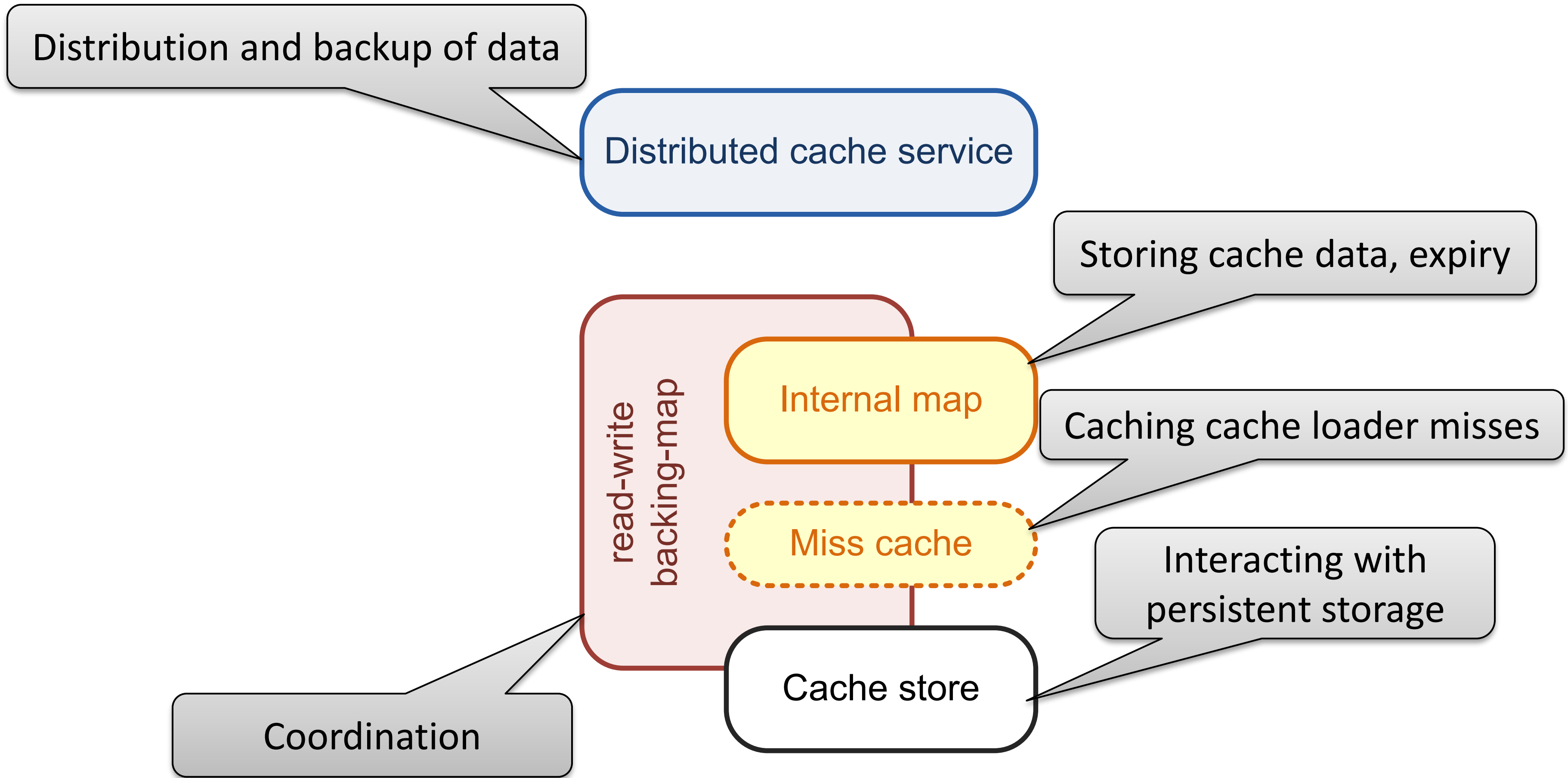
```
public class StoreFlagExtractor extends AbstractExtractor implements PortableObject {
    // ...
    private Object extractInternal(Binary binValue, BinaryEntry entry) {
        if (ExternalizableHelper.isDecorated(binValue)) {
            Binary store = ExternalizableHelper.getDecoration(binValue, ExternalizableHelper.DECO_STORE);
            if (store != null) {
                Object st = ExternalizableHelper.fromBinary(store, entry.getSerializer());
                return st;
            }
        }
        return Boolean.TRUE;
    }
}
```

BEHIND SCENES

How it works?

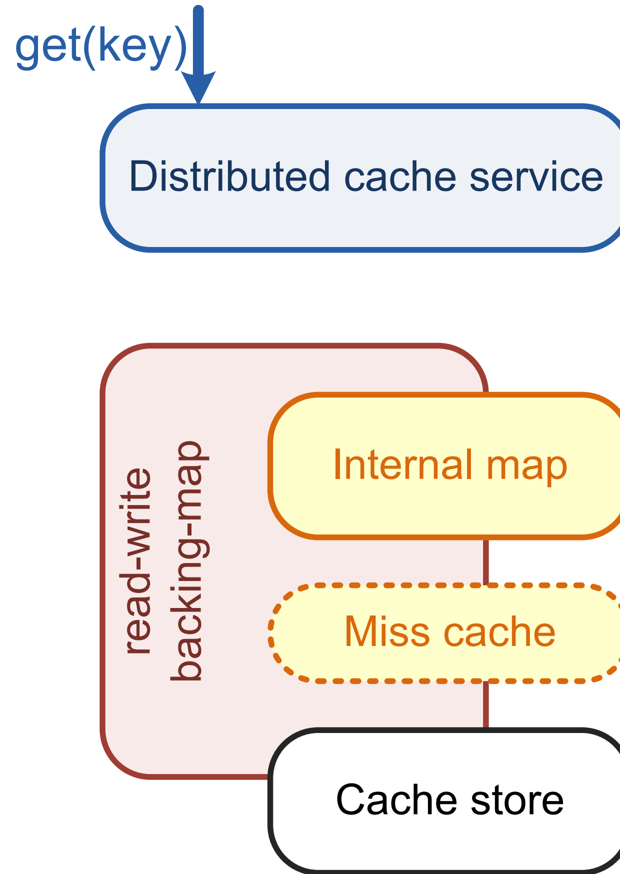


How it works?



How it works?

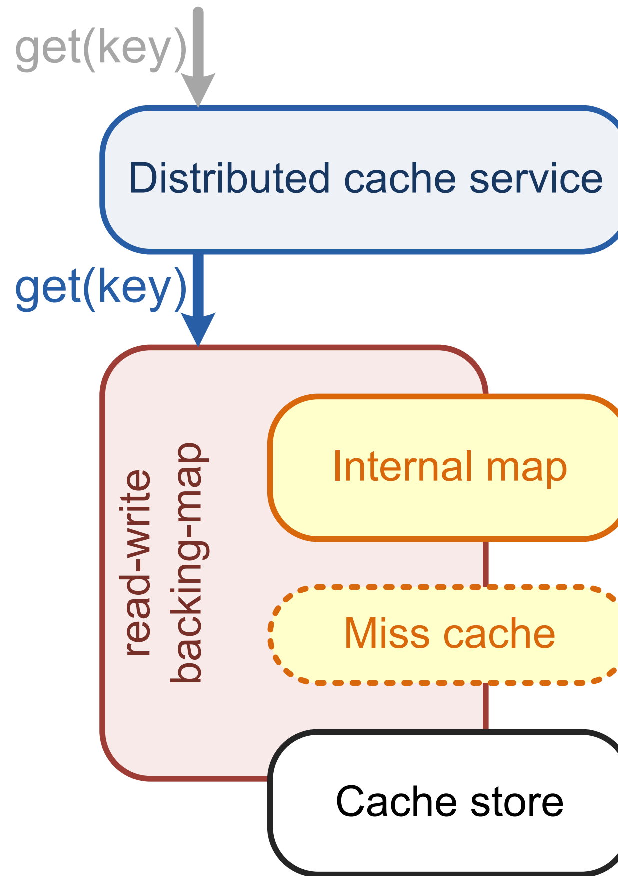
Cache service is receiving `get(…)` request.



read-through

How it works?

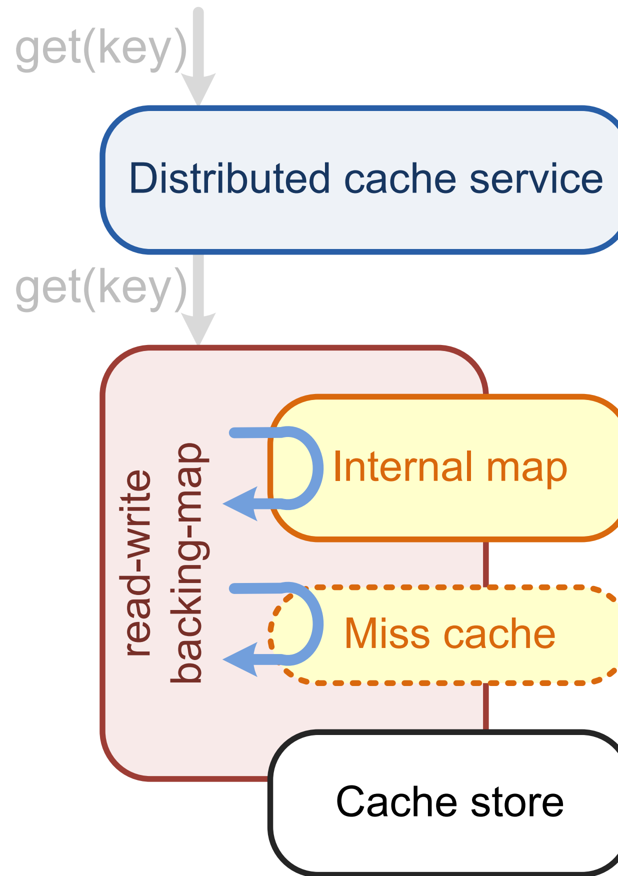
Cache service is invoking `get(…)` on backing map. Partition transaction is open.



read-through

How it works?

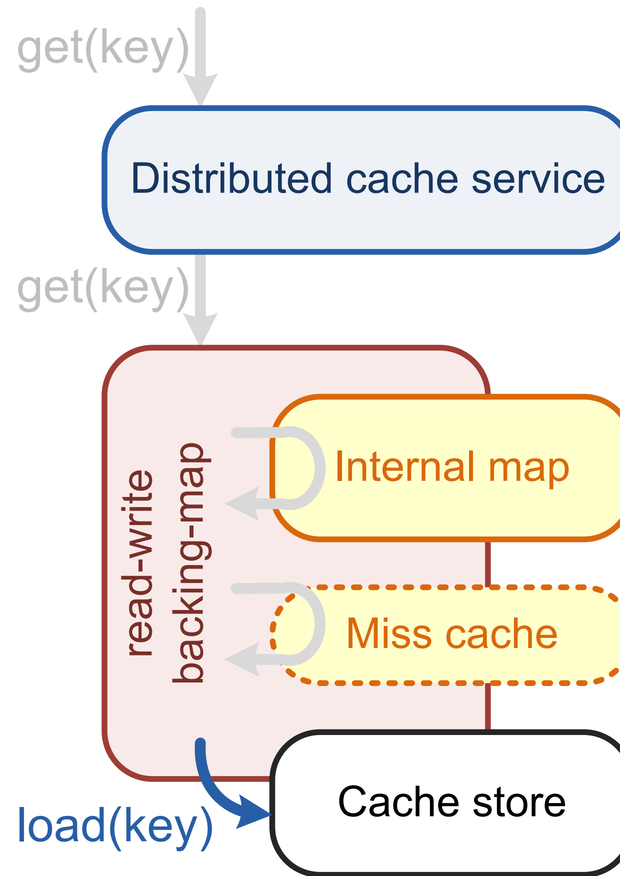
Backing map checks internal map and miss cache if present. Key is not found.



read-through

How it works?

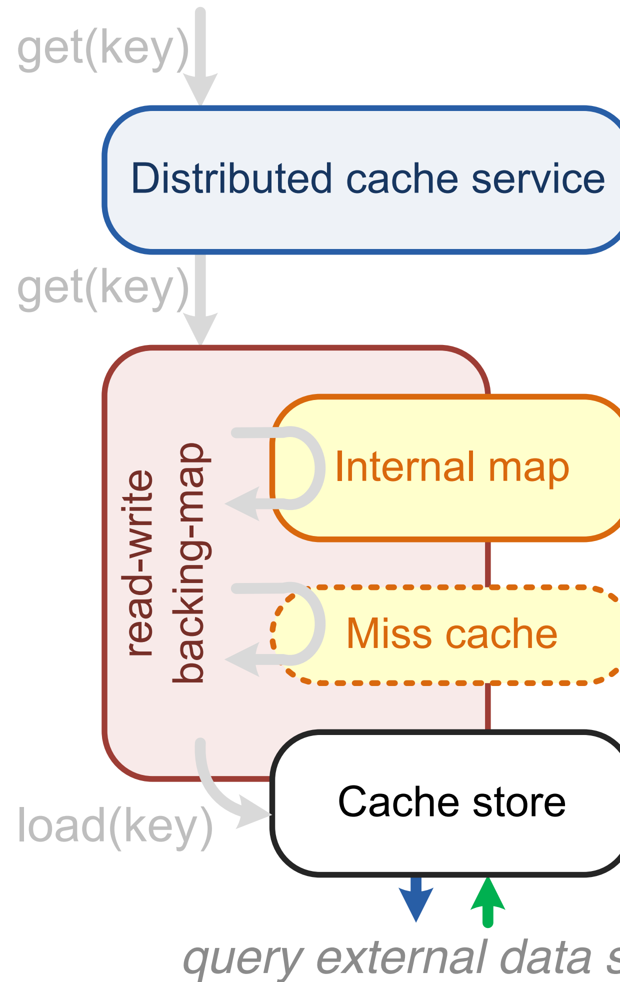
Backing map is invoking `load(…)` on cache loader.



read-through

How it works?

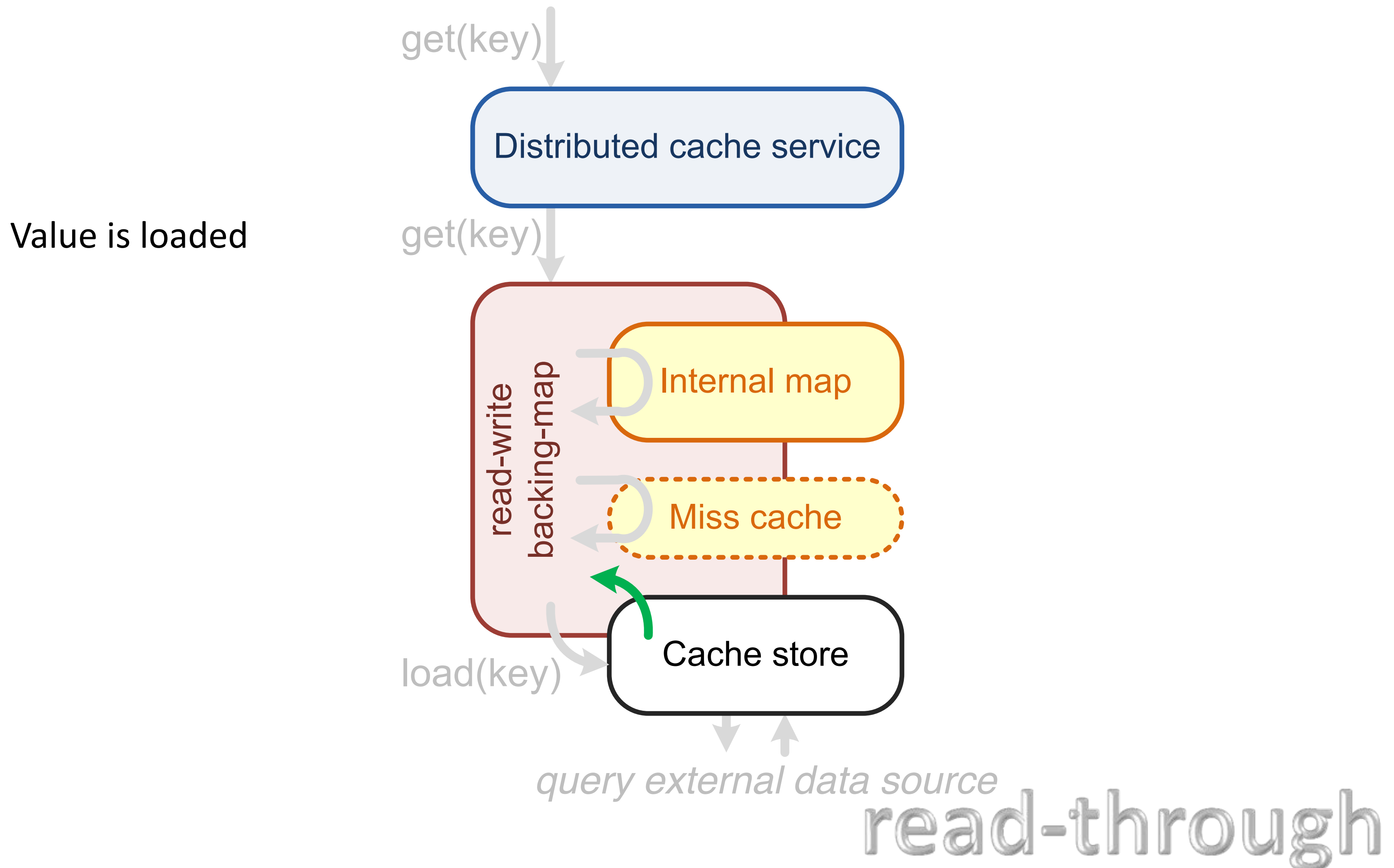
Cache loader is retrieving value for external source



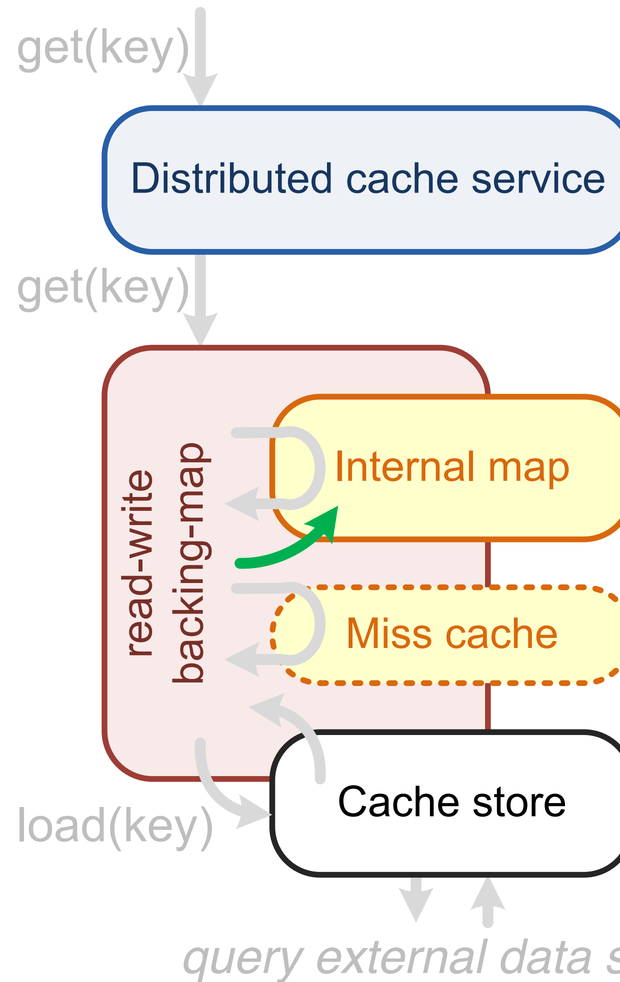
read-through

How it works?

Value is loaded



How it works?

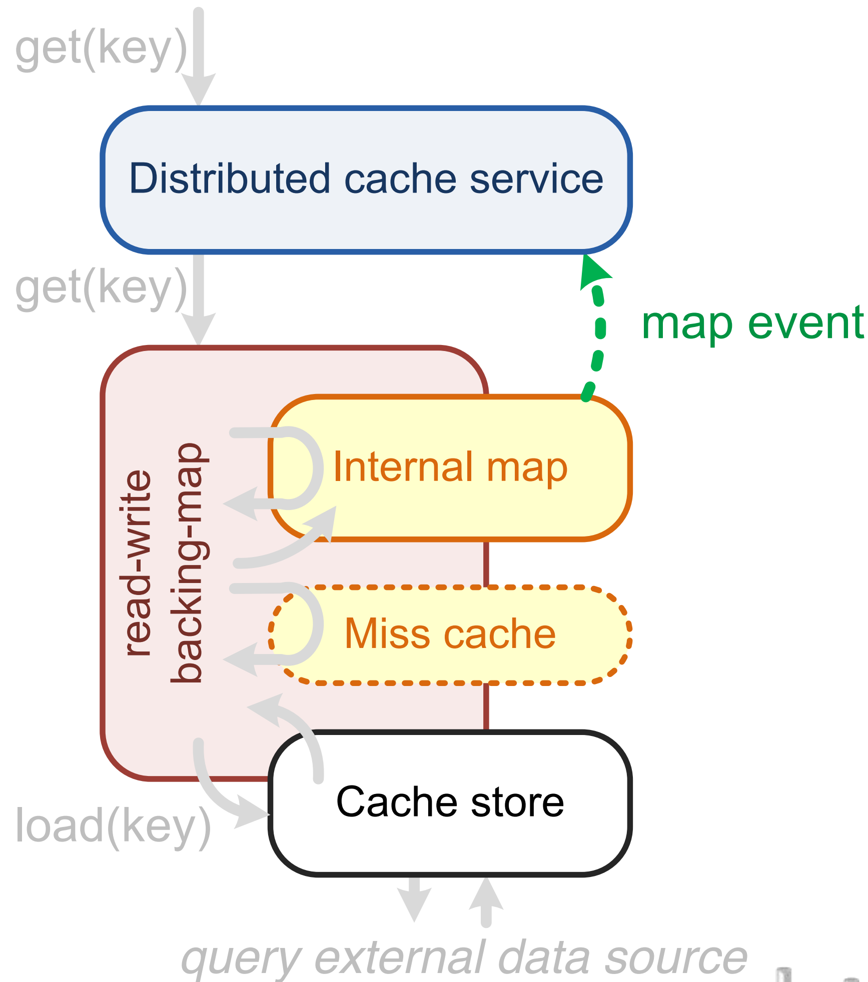


Backing map is updating internal map

read-through

How it works?

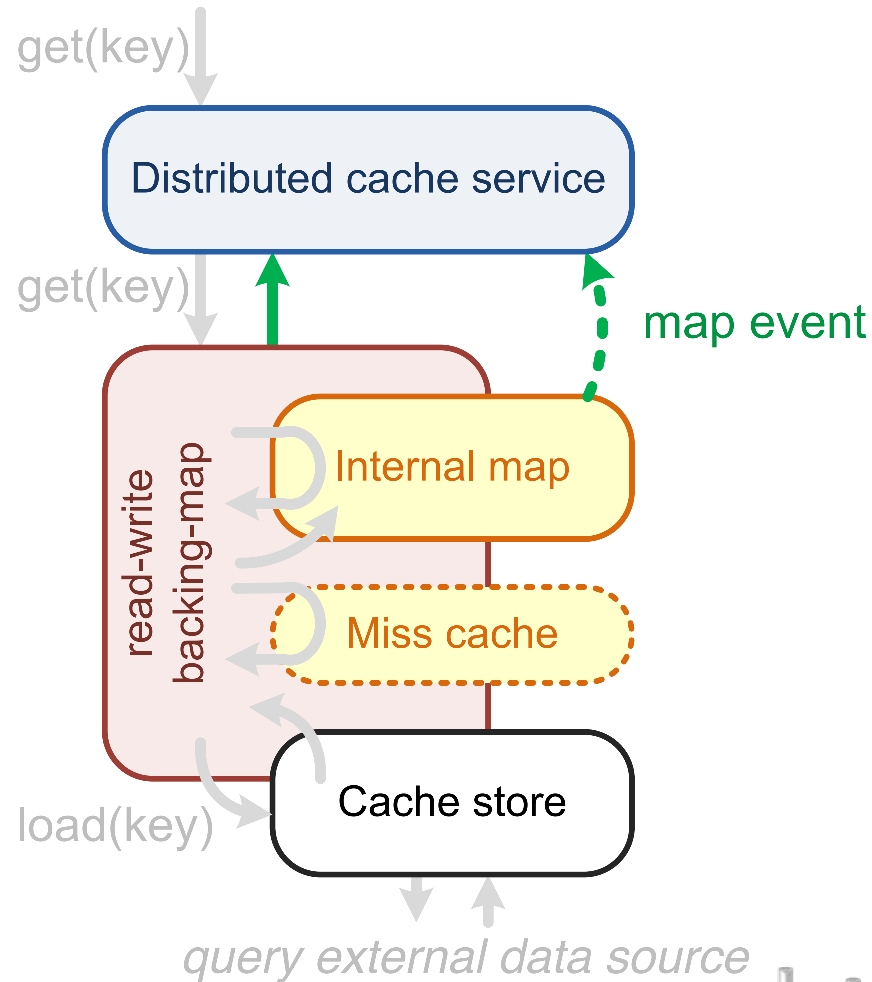
Internal map is observable and cache service is receiving event about new entry in internal map.



read-through

How it works?

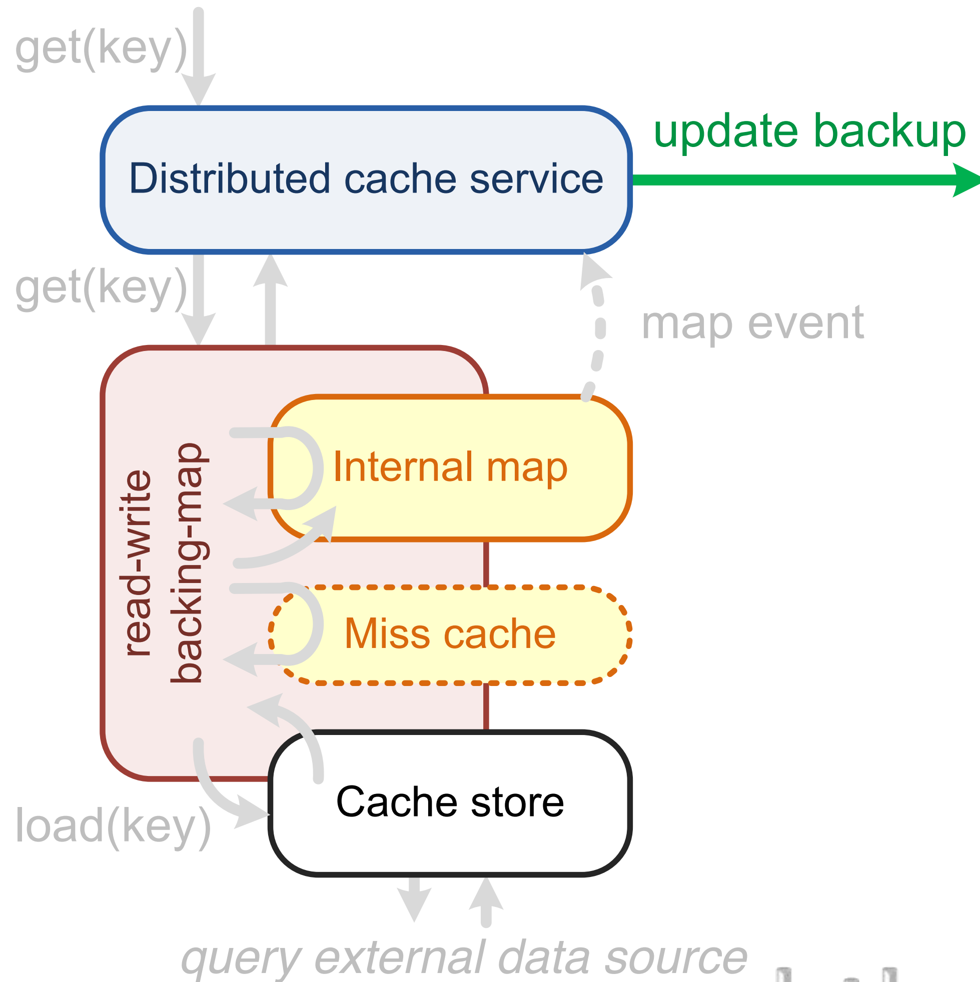
Call to backing map returns. Cache service is ready to commit partition transaction.



read-through

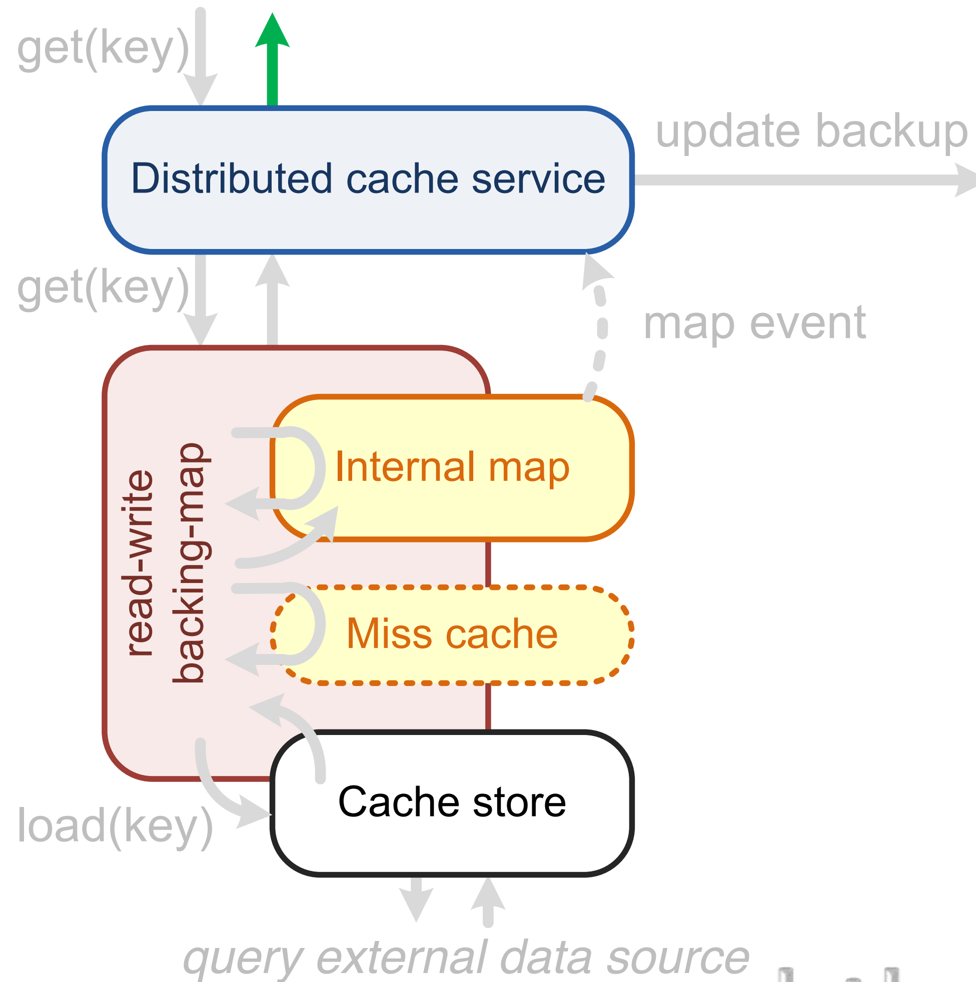
How it works?

Partition transaction is being committed. New value is being sent to backup node.



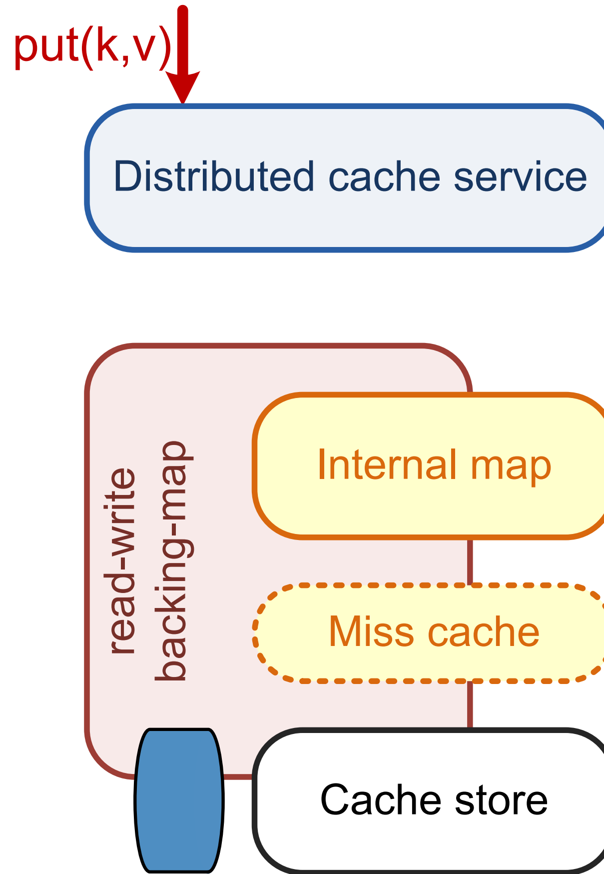
How it works?

Response for `get(…)` request is sent back as backup has confirmed update.



read-through

How it works?



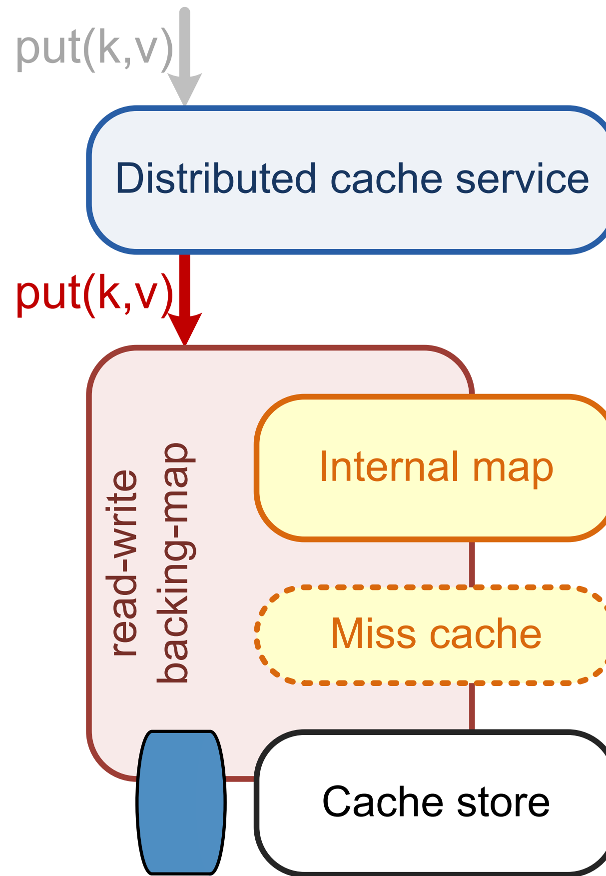
Cache service is receiving `put(…)` requiest.

write-behind

#1

How it works?

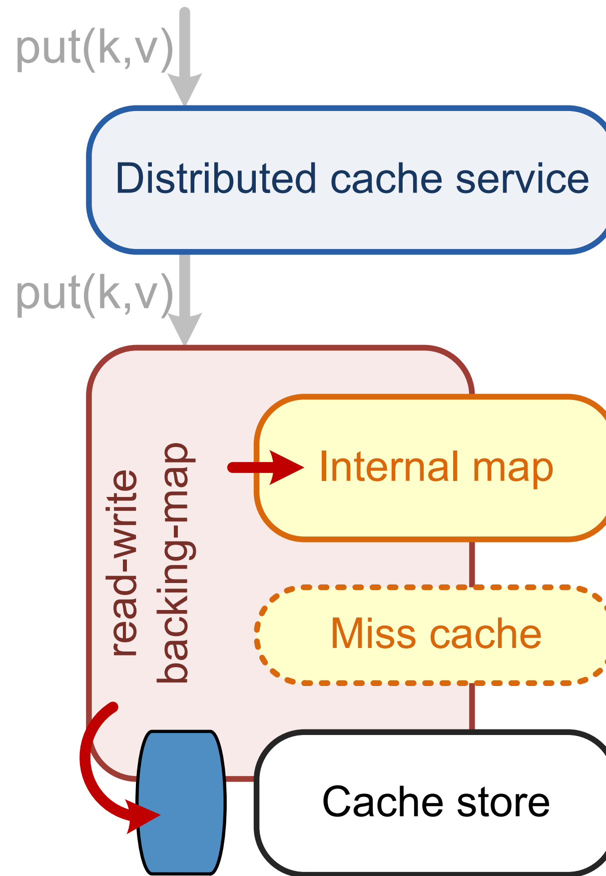
Cache service is invoking `put(...)` on backing map. Partition transaction is open.



write-behind

How it works?

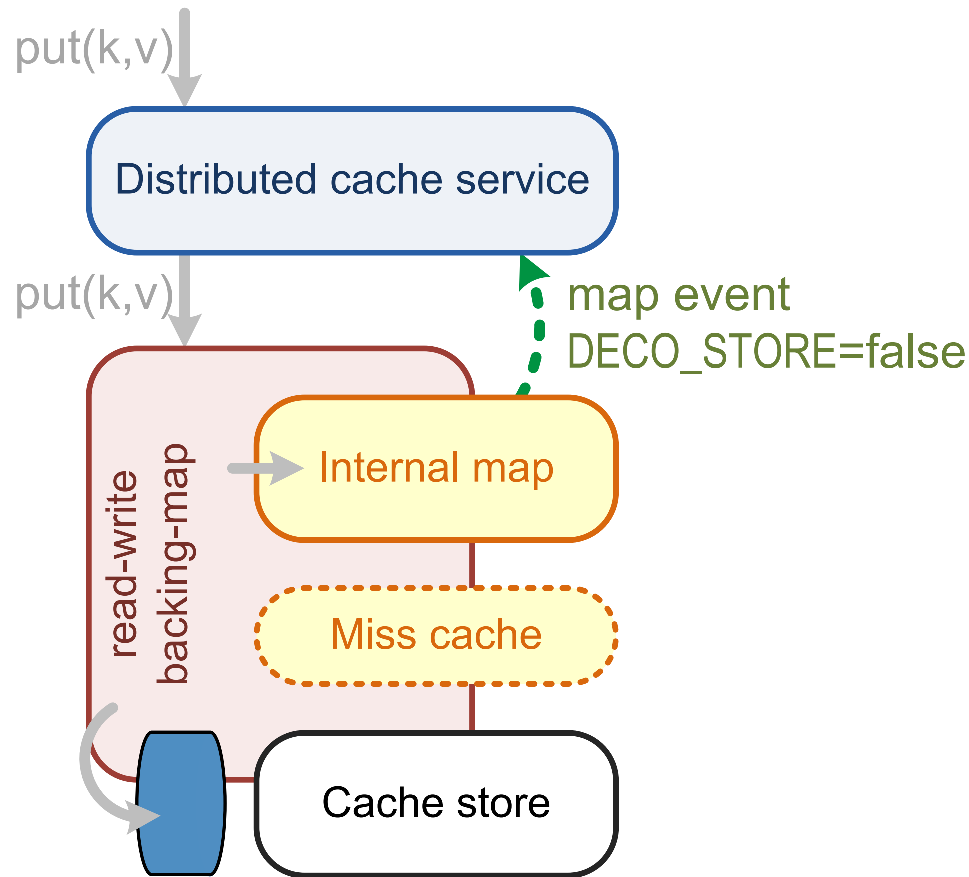
Value is immediately stored in internal map and put to write-behind queue.



write-behind

How it works?

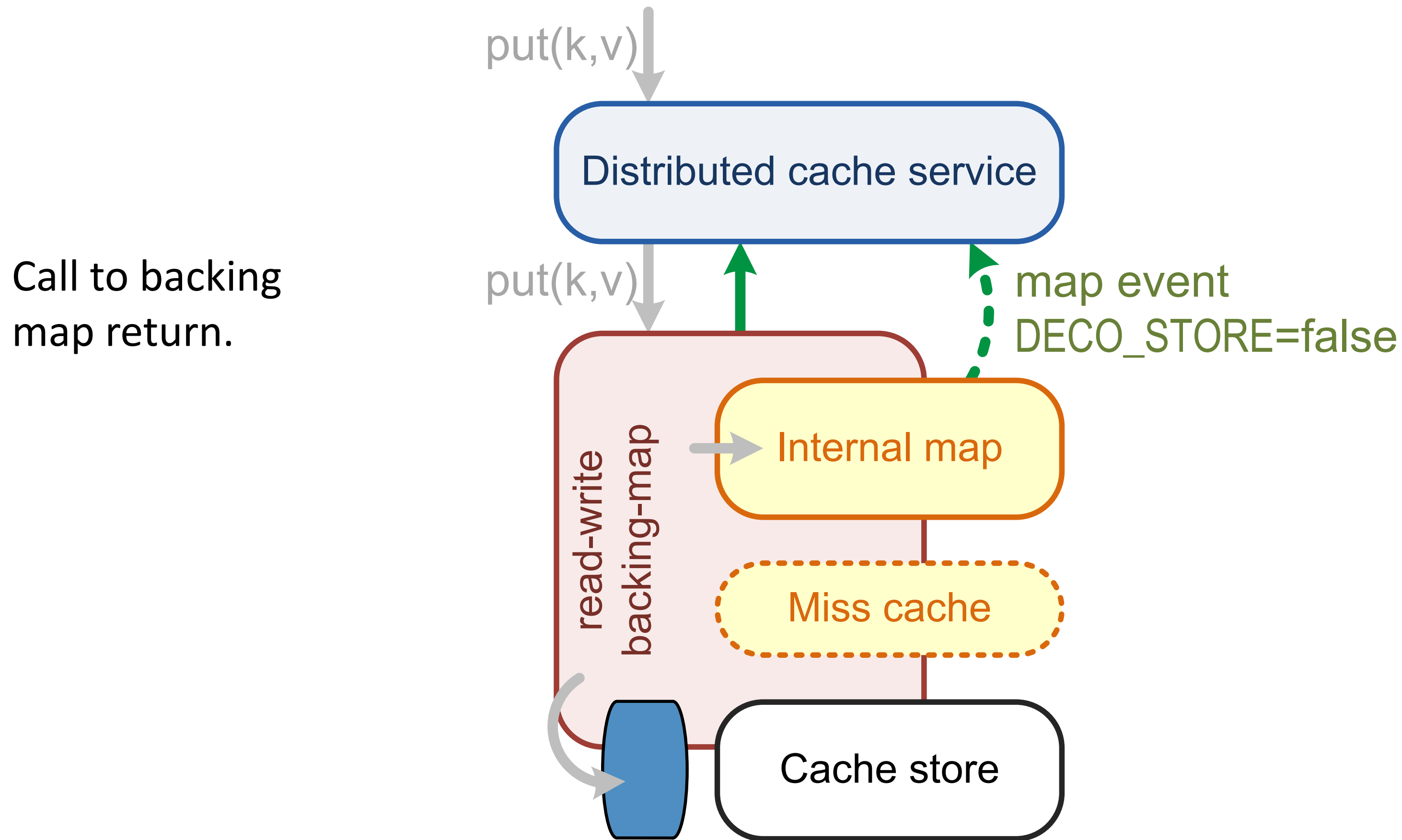
Cache service is receiving event, but backing map is decorating value with DECO_STORE=false flag to mark that value is yet-to-stored.



write-behind

How it works?

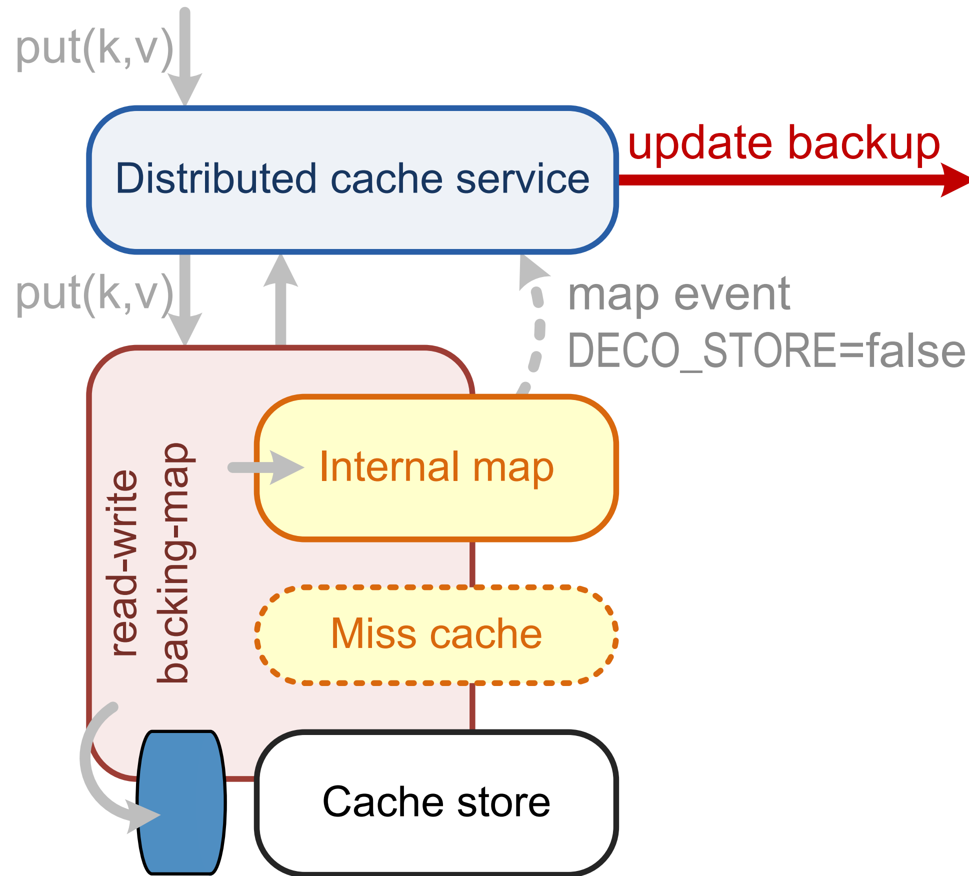
Call to backing map return.



write-behind

How it works?

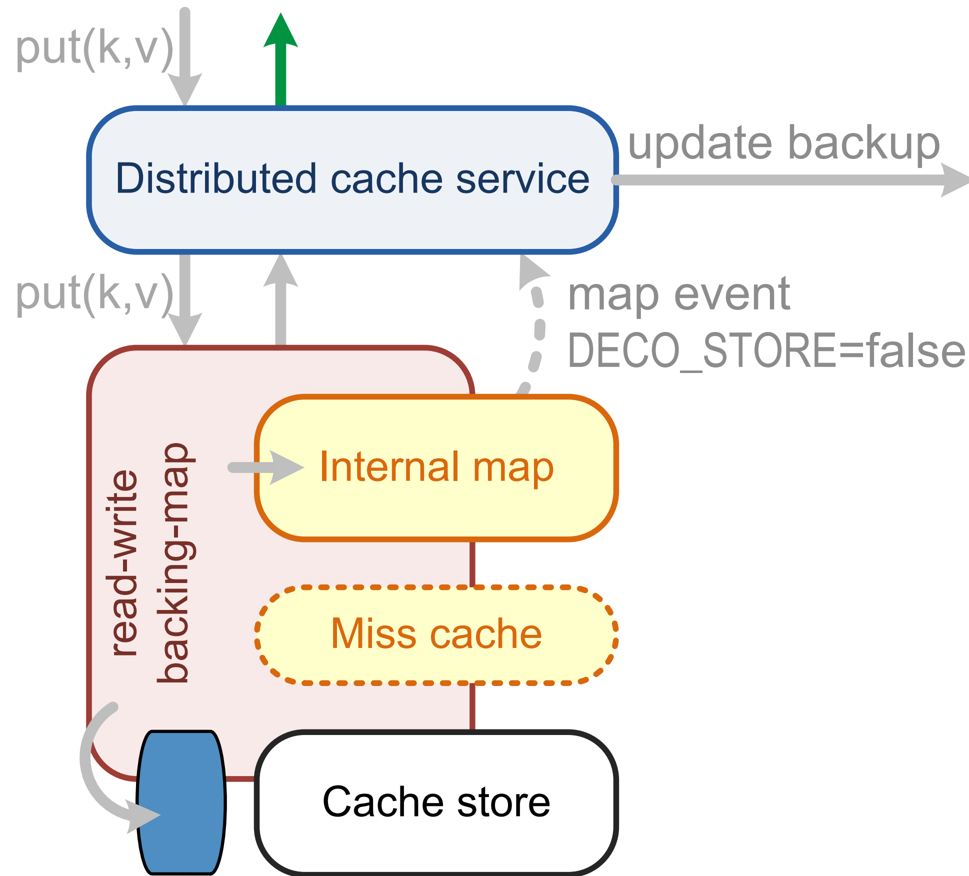
Partition transaction is being committed. Backup will receive value decorated with DECO_STORE=false.



write-behind

How it works?

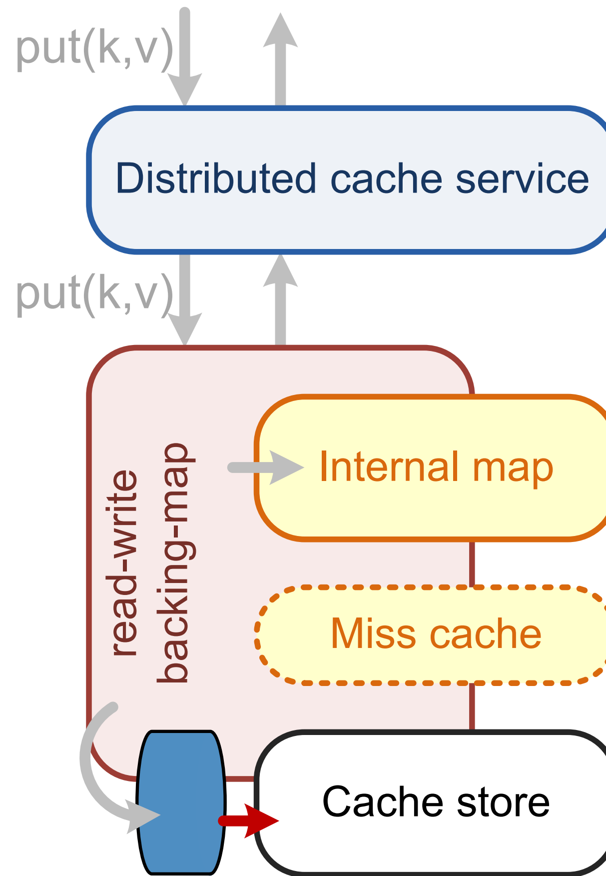
Cache service is sending response back as soon as backup is confirmed.



write-behind

How it works?

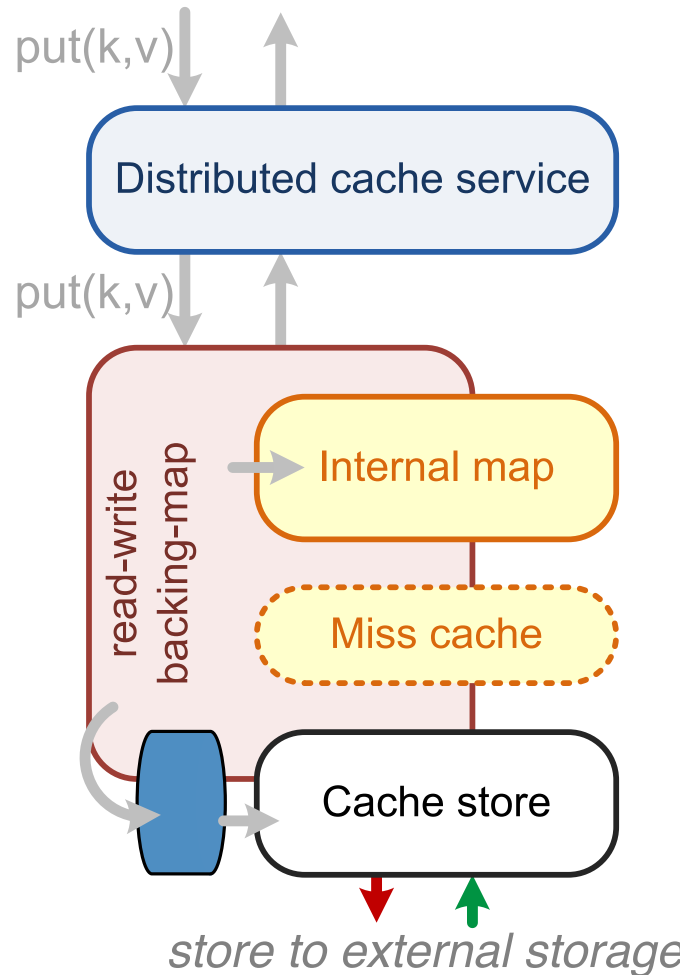
Eventually, cache store is called to persist value.
It is done on separate thread.



write-behind

How it works?

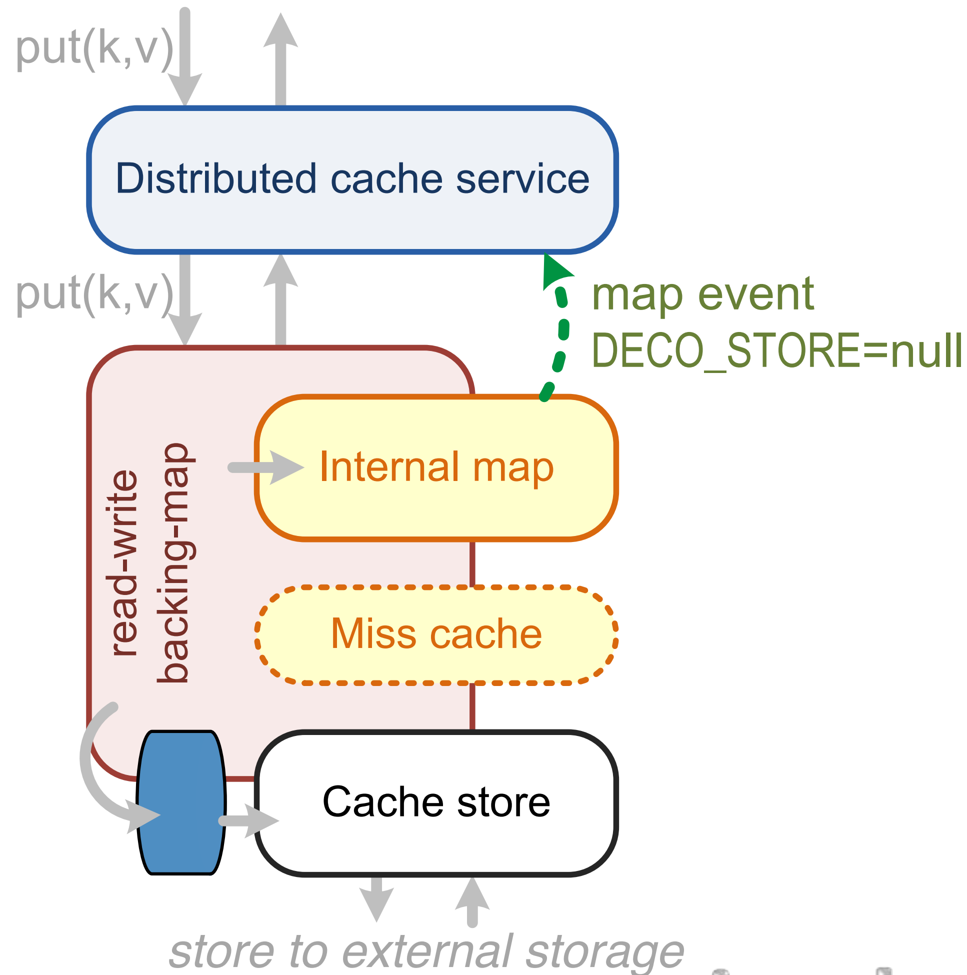
Value is stored in external storage by cache store.



write-behind

How it works?

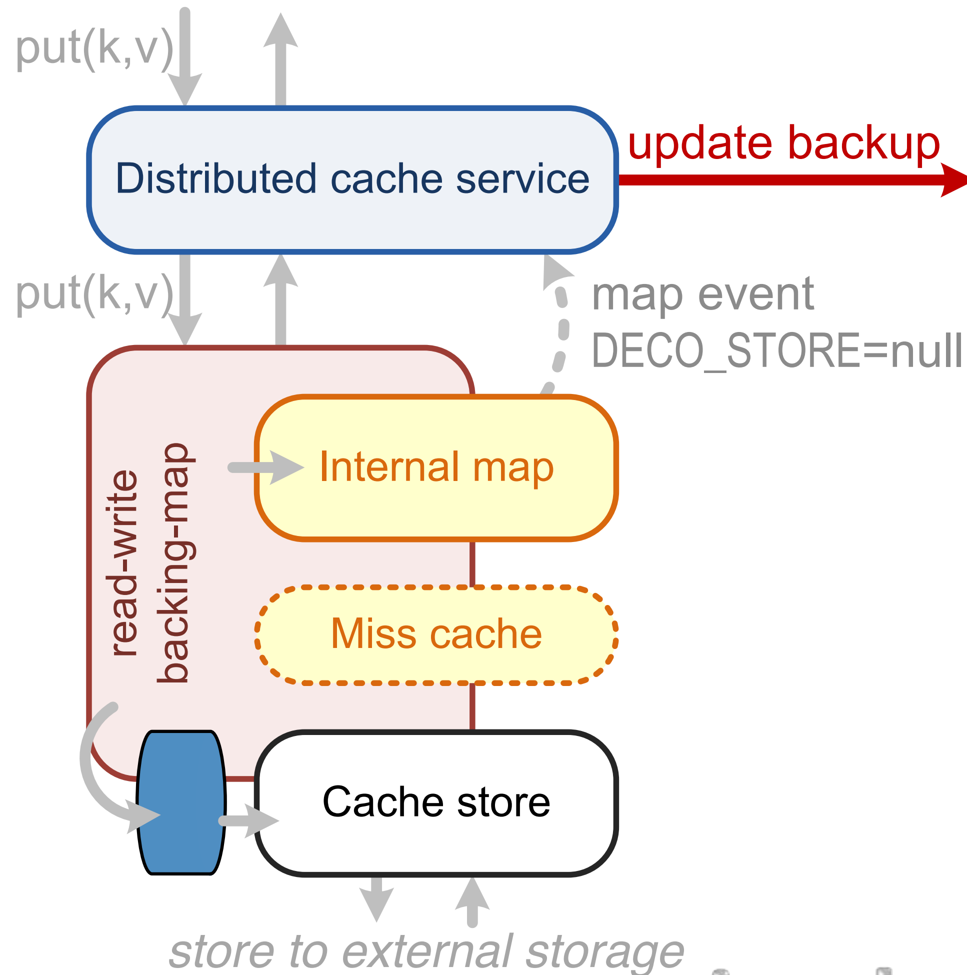
Once call to cache store has returned successfully. Backing map is removing DECO_STORE decoration from value is internal map. Cache service is receiving map event



write-behind

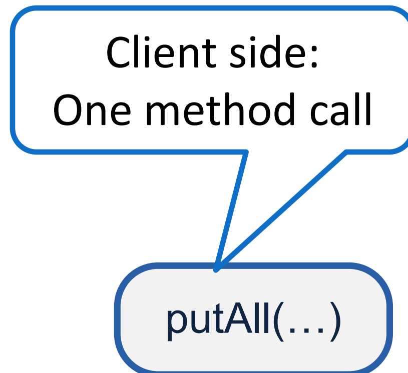
How it works?

Map event was received by cache service outside of service thread. It will be put to OOB queue and eventually processed.
Update to backup will be sent once event is processed.

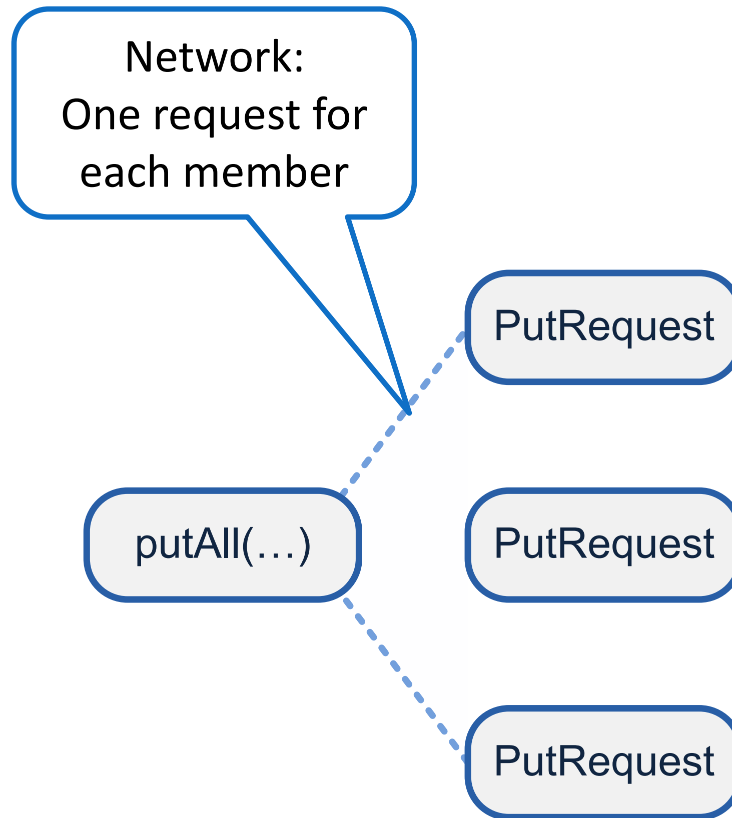


THREADING

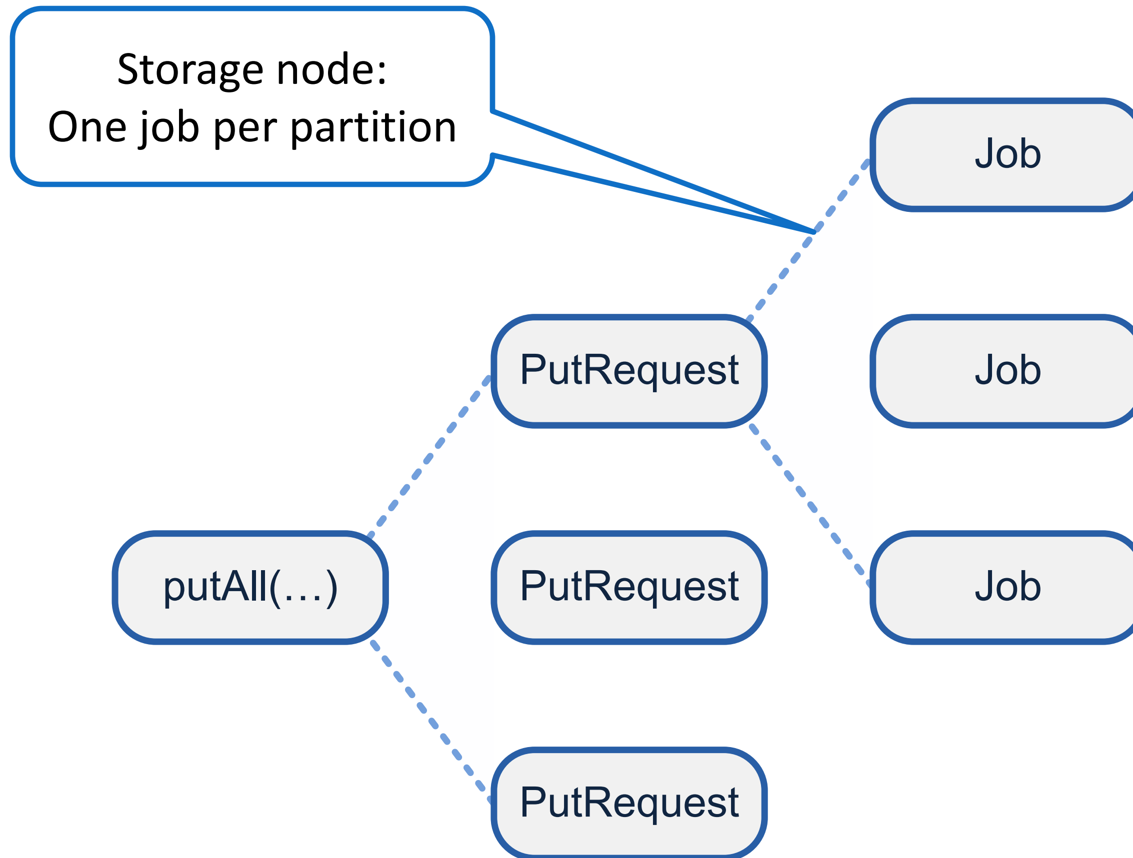
Requests and jobs



Requests and jobs



Requests and jobs



Requests and jobs

Problem

- Single API call may produce hundreds of jobs for worker threads in cluster (limited by partition count).
- Write-through and read-through jobs could be time consuming.
- While all threads are busy by time consuming jobs, cache is unresponsive.

Requests and jobs

Workarounds

- Huge thread pools
- Request throttling
  - By member (one network request at time)
  - By partitions (one job at time)
- Priorities
  - Applicable only to EP and aggregators

“UNBREAKABLE CACHE” PATTERN

“Canary” keys

- Canary keys – special keys (one per partitions) ignored by all cache operations.
- Canary key is inserted once “recovery” procedure have verified that partition data is complete.
- If partition is not yet loaded or lost due to disaster, canary key will be missing.

Recovery procedure

- Store object hash code in database
- Using hash you can query database for all keys belonging to partition
- Knowing all keys, can use read-through to pull data to a cache

---

- Cache is writable during recovery!
- Coherence internal concurrency control will ensure consistency

“Unbreakable cache”

read/write-trough + canary keys + recovery

- Key based operations rely on read-through
- Filter based operations are checking “canary” keys (and activate recovery is needed)
- Preloading = recovery
- Cache is writable at all times

Checking “canary” keys

- Option 1
  - check “canary” keys
  - perform query
- Option 2
  - perform query
  - check “canary” keys

Checking "canary" keys

- Option 1
  - ✓ check "canary" keys
  - ✓ perform query

Wrong

- Option 2
  - ✓ perform query
  - ✓ check "canary" keys

Wrong

- **Right way**
  - ✓ **check "canaries" inside of query!**

“Unbreakable cache”

Motivation

- Incomplete data set would invalidate hundred of hours of number crunching
- 100% complete data or exception
- Persistent DB is requirement anyway

Summary

- Transparent recovery (+ preloading for free)
- Always writable (i.e. feeds are not waiting for recovery)
- Graceful degradation of service in case of “disastrous conditions”

Thank you

http://blog.ragozin.info
- my articles
http://code.google.com/p/gridkit
- my open source code

Alexey Ragozin
alexey.ragozin@gmail.com