CacheStore Lore

The ins & outs of Coherence DB Integration

Phil Wheeler

Credit Suisse

A few pointers on DB integration

caveats

- Hope this is useful
- There’s much I don’t (yet) understand ☺

schemes

QUIZ!

which schemes

support a CacheStore?

schemes

local	✓
distributed	✓
replicated	✗
transactional	✗

interfaces

QUIZ!

what are

the key Java interfaces to know?

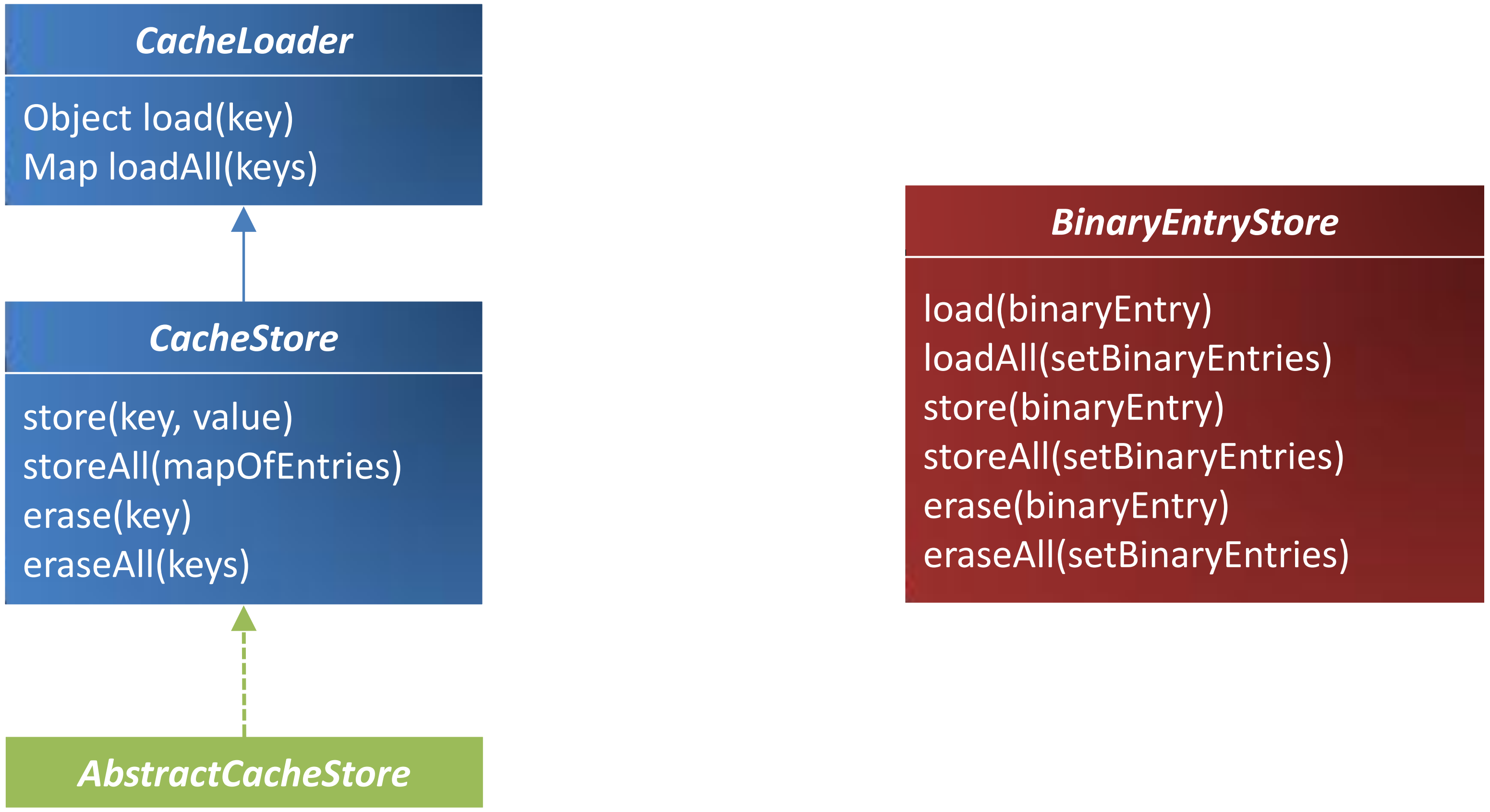
interfaces

CacheLoader

CacheStore

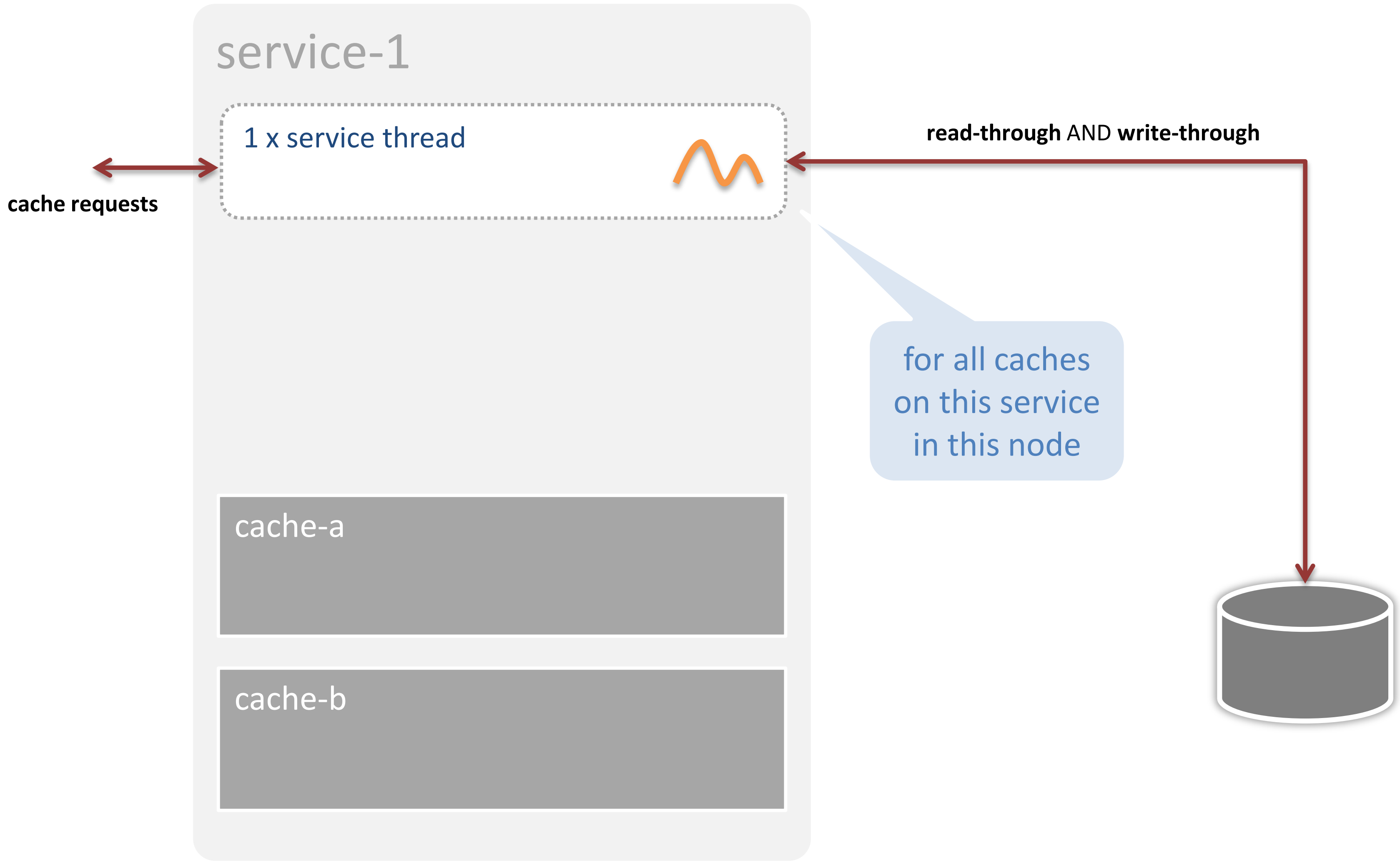
BinaryEntryStore

interfaces

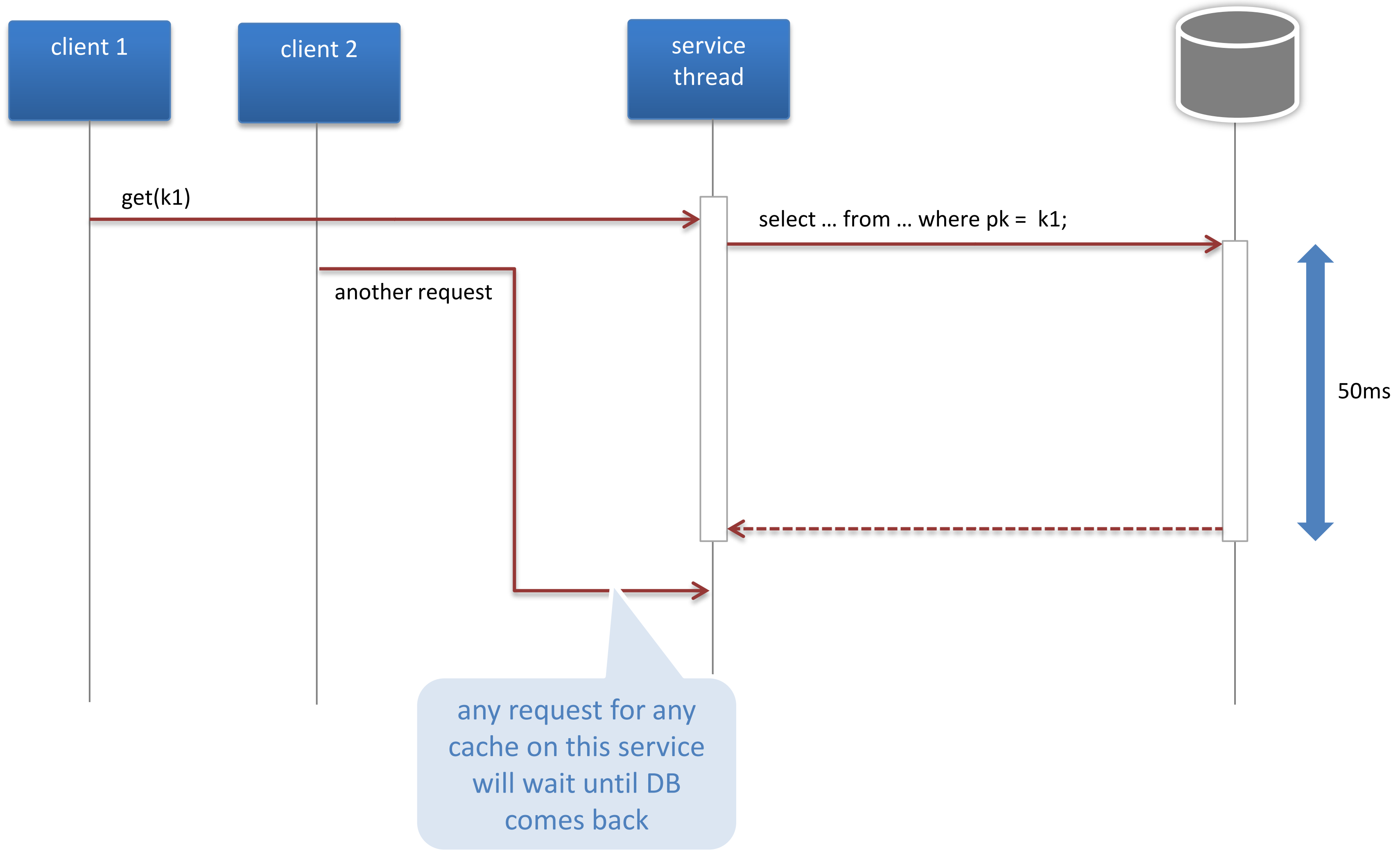

CacheLoader
Object load(key)
Map loadAll(keys)
CacheStore
store(key, value)
storeAll(mapOfEntries)
erase(key)
eraseAll(keys)
AbstractCacheStore
BinaryEntryStore
load(binaryEntry)
loadAll(setBinaryEntries)
store(binaryEntry)
storeAll(setBinaryEntries)
erase(binaryEntry)
eraseAll(setBinaryEntries)

threads

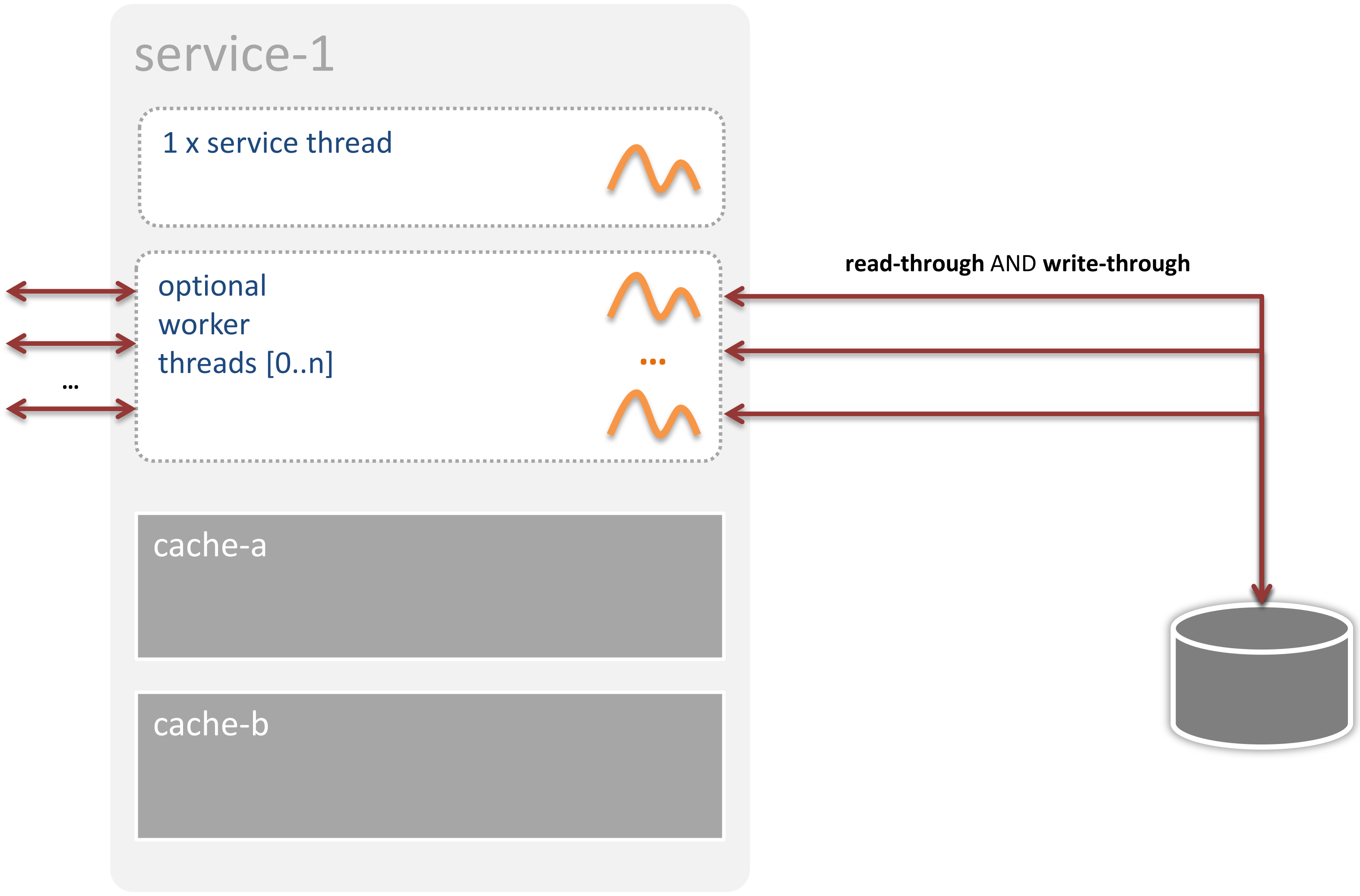
default situation


service-1
1 x service thread
cache requests
read-through AND write-through
for all caches on this service in this node
cache-a
cache-b

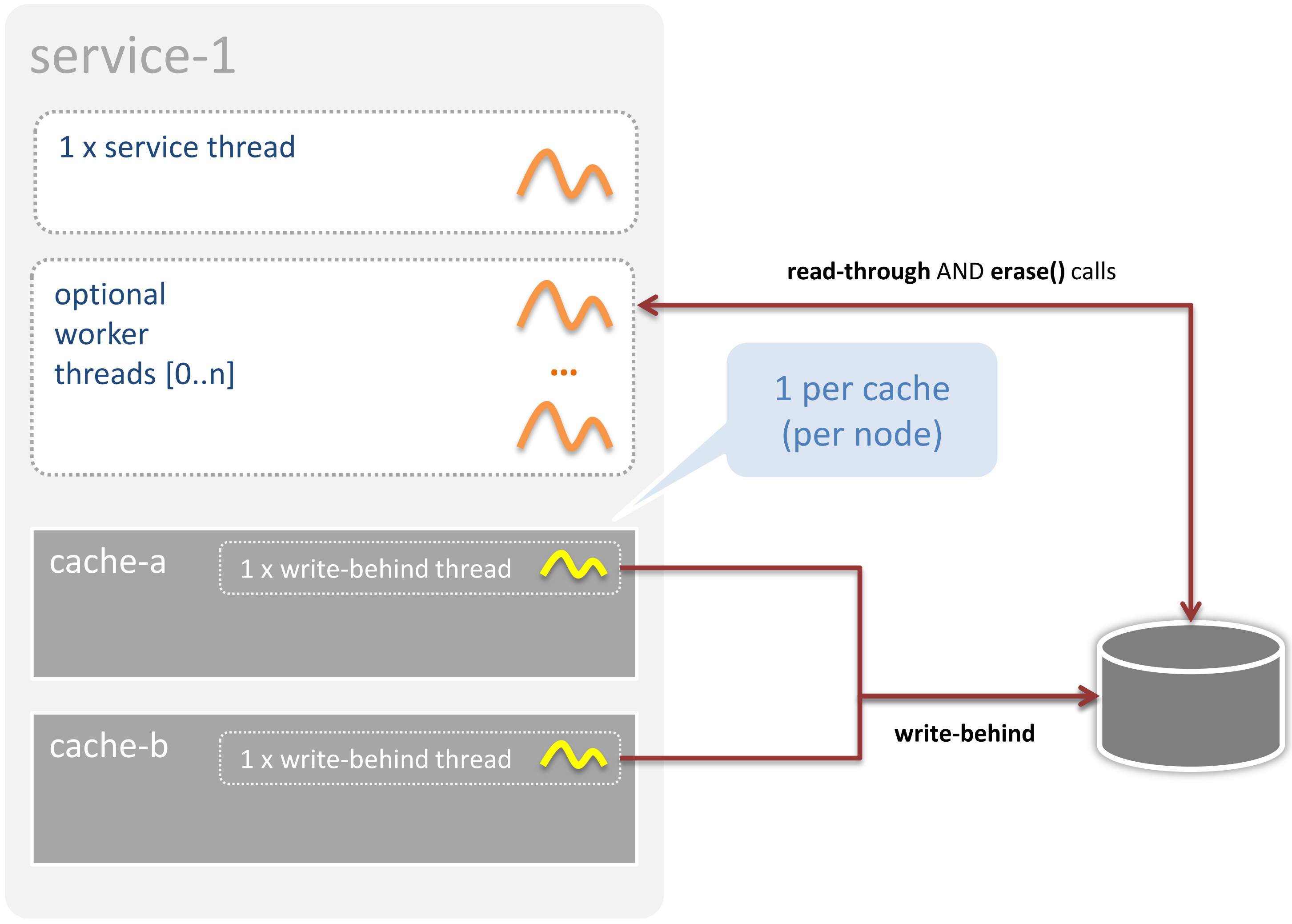
blocking


client 1
client 2
service thread
get(k1)
select … from … where pk = k1;
another request
50ms
any request for any cache on this service will wait until DB comes back

worker threads



threads: write-behind

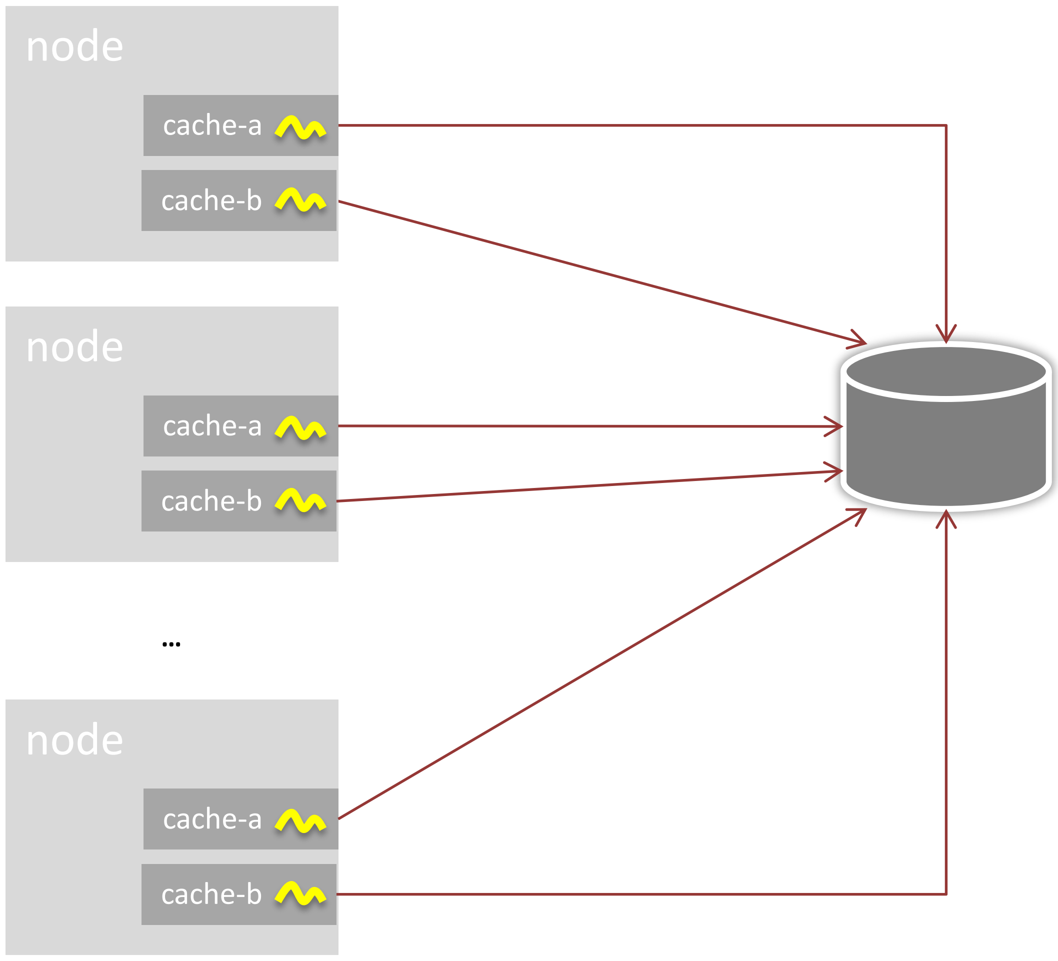


QUIZ!

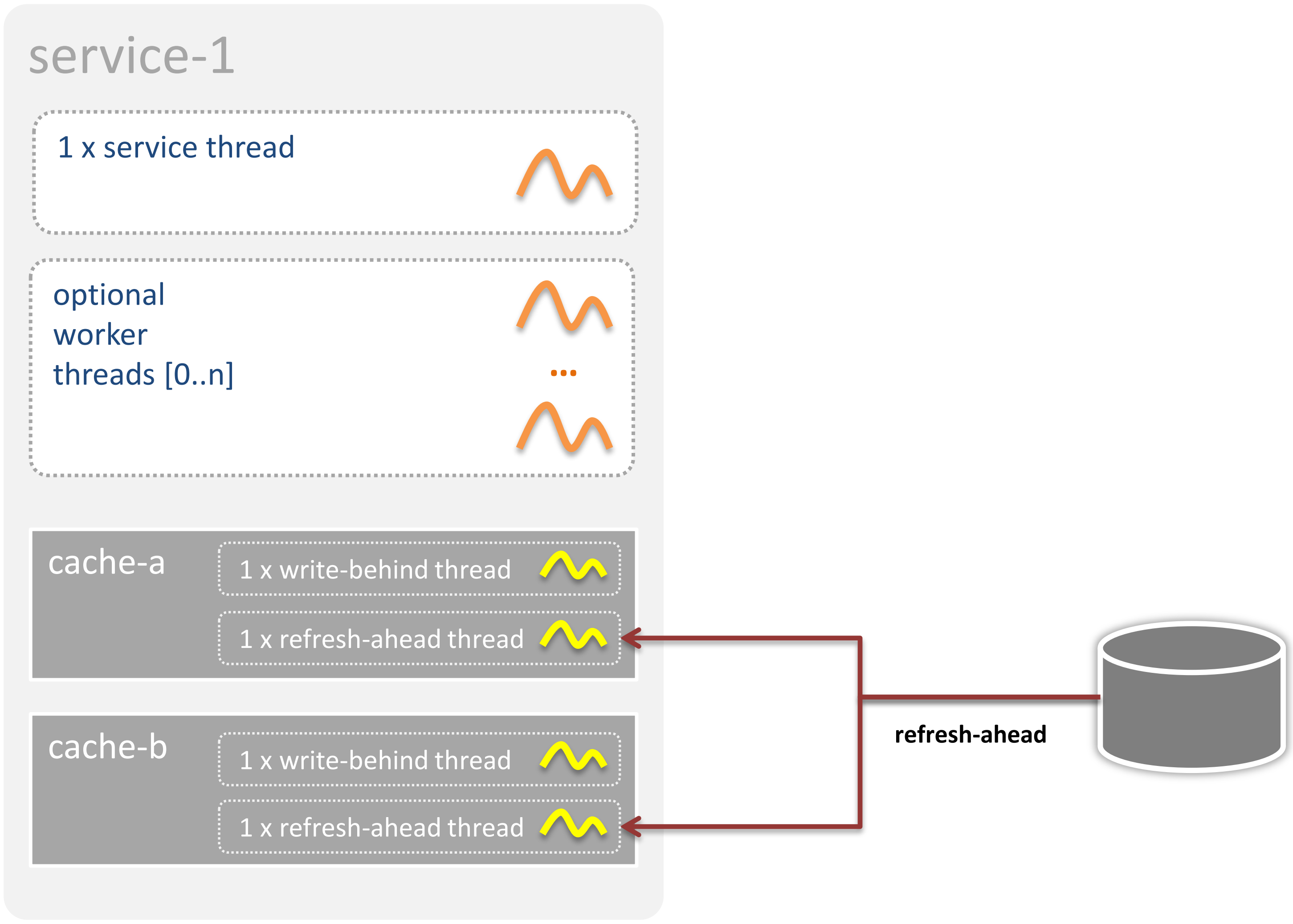
write-behind

how do we scale out write-behind?

write-behind scale-out



all service threads



thread names

per service threads	
service thread	DistributedCache:<service-name>
worker threads	<service-name>Worker:0
	<service-name>Worker:n

not unique!

per cache threads	
write behind	WriteBehindThread:CacheStoreWrapper(<cache-store-class-name>):<service-name>
refresh-ahead	ReadThread:CacheStoreWrapper(<cache-store-class-name>):<service-name>

handling exceptions

DB constraints

- write-through ✓
- write-behind ✗

write-through exceptions

`<rollback-cachestore-failures>`

- leave this true
- pass the exception back to the caller

write-behind exceptions

rule #1

- avoid DB exceptions in the first place!

treat the write-behind DB tables like an append log that can’t fail

acceptable DB exceptions

- DB unavailable ✓
- Out of space ✓

- constraint violations no ✗
- business rules, etc. NO! ✗

write-behind exceptions

rule #2

- catch SQLExceptions
- log the problem
- throw a RuntimeException

retries

write-behind exceptions

rule #3

- enable retries

but beware stuck items

Retrying

<write-requeue-threshold> : *> 0*

retrying

is the entire batch

not individual entries within a batch

QUIZ!

how many times

- will Coherence call CacheStore.store() if you throw a RuntimeException?

QUIZ!

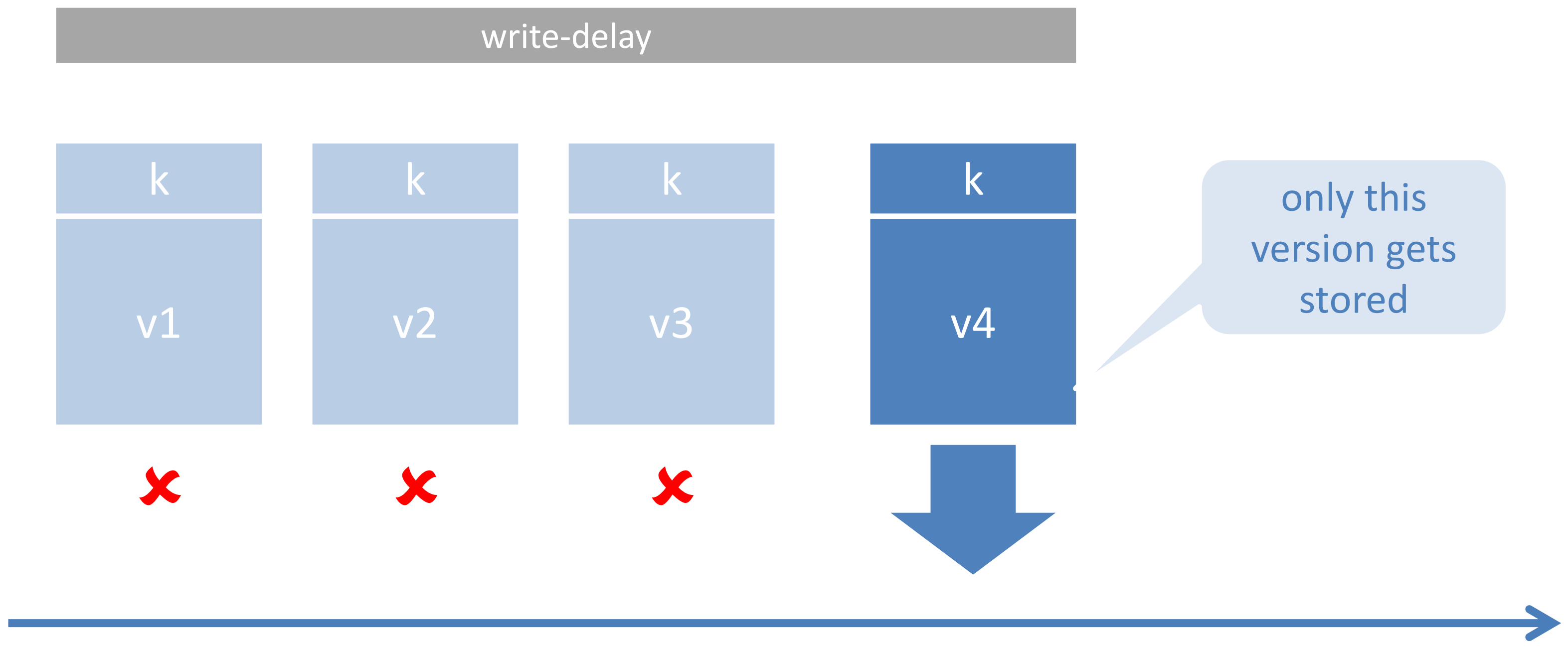
how often

- will Coherence call CacheStore#store() if you throw a RuntimeException?

efficiency

coalescing

<write-delay>



QUIZ!

cache miss cache

<miss-cache-scheme>

What is it?

backups

do you need them after a write?

```
<backup-count-after-writebehind>
```

batching

rule #3

Don’t write one entry at a time

storeAll()

implement it

don’t do what AbstractCacheStore does…

batching

Inserts? Updates?

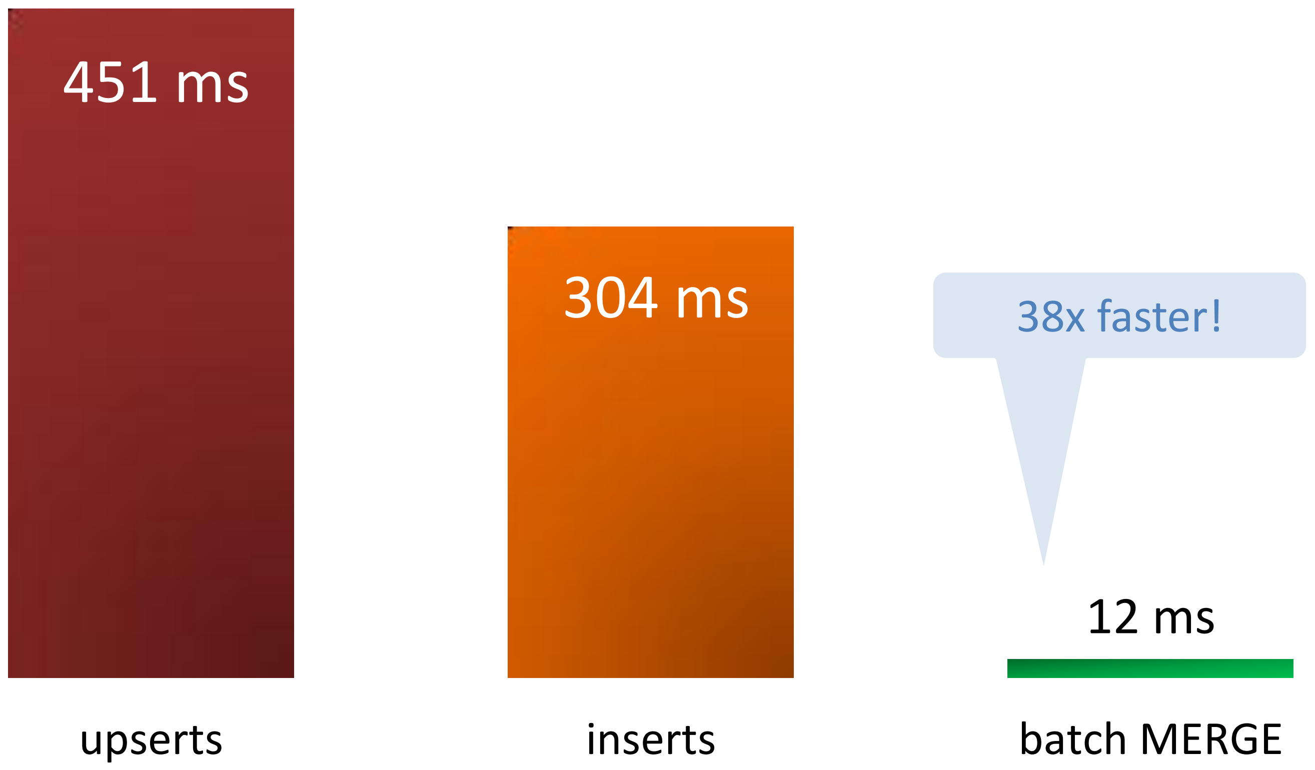
MERGE!

(or call a stored proc to do this)

merge

```
merge
    into MY_TABLE a
    using DUAL b
on (a.pk=?)
when matched
    then update set a.col1=?, a.col2=?, …
when not matched
    then insert values (?, ?, ?, …)
```

batch performance



time to persist 128 entries to Oracle DB

batch size

`<write-max-batch-size>`

otherwise `storeAll()` gets 128 entries

(or fewer)

transactions

transactions

- write-through ✓
  maybe (e.g. need to update multiple tables)

- write-behind ✗
  can’t retry part of a batch

hints and tips

idempotency

rule #4

Make your store methods idempotent

beware

`<cachestore-timeout>` DON'T ✘

Use the guardian – and change that

(to log & continue)

What does your JDBC driver do when a thread gets interrupted?

beware

write-behind is resilient

BUT

you can still lose data obviously

deletions

QUIZ!

when is this called?

erase()

erase

- optional
- synchronous
  (even for write behind)

erase

`UnsupportedOperationException`

gets logged first time

still gets called though

handly binary entries

interface

BinaryEntryStore

load(binaryEntry)
loadAll(setBinaryEntries)
store(binaryEntry)
storeAll(setBinaryEntries)
erase(binaryEntry)
eraseAll(setBinaryEntries)

BinaryEntryStore

- avoids deserialisation
- handy to publish entries to other clusters
- recovery DB

BinaryEntryStore

access to previous value

`BinaryEntry#getOriginalBinaryValue()`

`BinaryEntry#getOriginalValue()`

previous value

- memory hit
- where is this stored?

monitoring

CacheMBean

- QueueSize
  The size of the write-behind queue

- StoreFailures
  The total number of cache store failures

- StoreAverageBatchSize
  StoreAverageReadMillis
  StoreAverageWriteMillis

logs

- log SQLExceptions

e.g. tablespace is full

that’s all