Caching with Spring: Advanced Topics and Best Practices

Michael Plöd
@bitboss

I will talk about

Caching Types / Topologies
Best Practices for Caching in Enterprise Applications
Caching with Spring
JCache and Spring

I will NOT talk about

Latency / Synchronization discussion
What is the best caching product on the market
HTTP / Database Caching
Caching in JPA, Hibernate or other ORMs

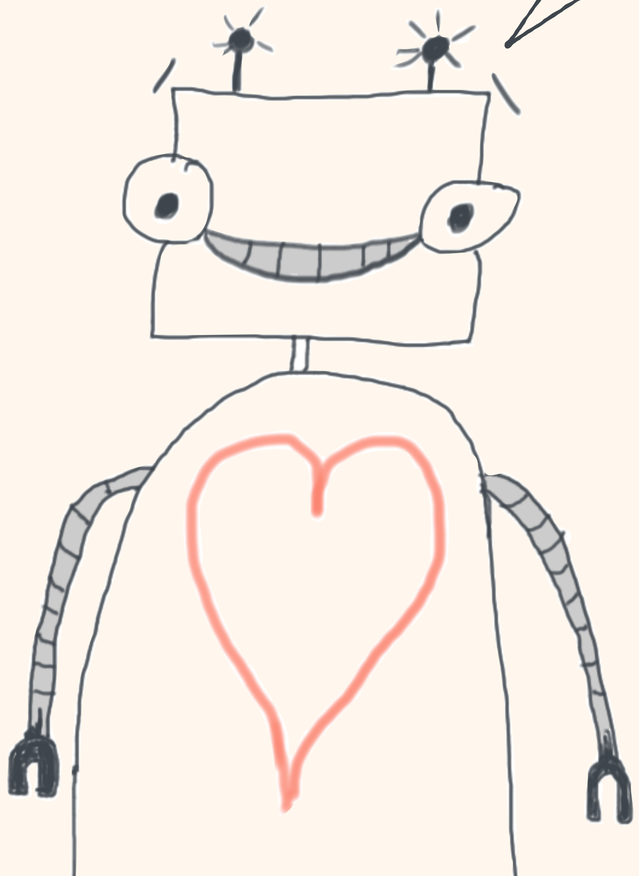
Cache

/ kæʃ /

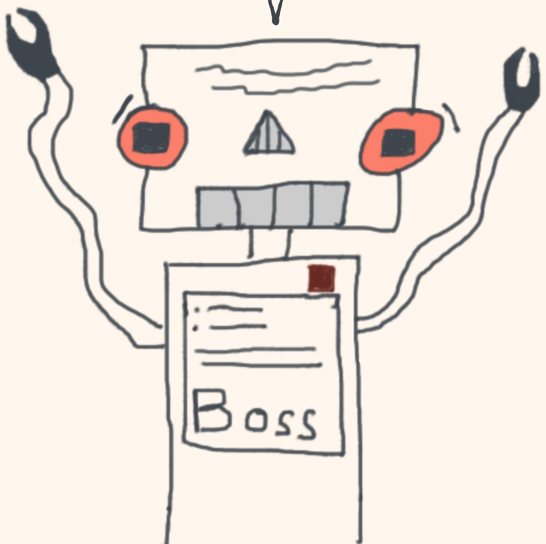
In computing, a cache is a component that **transparently stores data so that future requests for that data can be served faster**. The data that is stored within a cache might be values that have been computed earlier or duplicates of original values that are stored elsewhere. If requested data is contained in the cache (**cache hit**), this request can be served by simply reading the cache, which is comparatively faster. Otherwise (**cache miss**), the data has to be recomputed or fetched from its original storage location, which is comparatively slower. **Hence, the greater the number of requests that can be served from the cache, the faster the overall system performance becomes.**

Source: http://en.wikipedia.org/wiki/Cache_(computing)

That's awesome. Let's **cache everything** and **everywhere** and **distribute** it all in a **Cluster** in a **transactional manner**
ohhh by the way: Twitter has been doing that for ages



Are you crazy?



Business-Applications

Twitter / Facebook & co.

Many enterprise grade projects are adapting caching too defensive or too offensive and are running into consistency or performance issues because of that

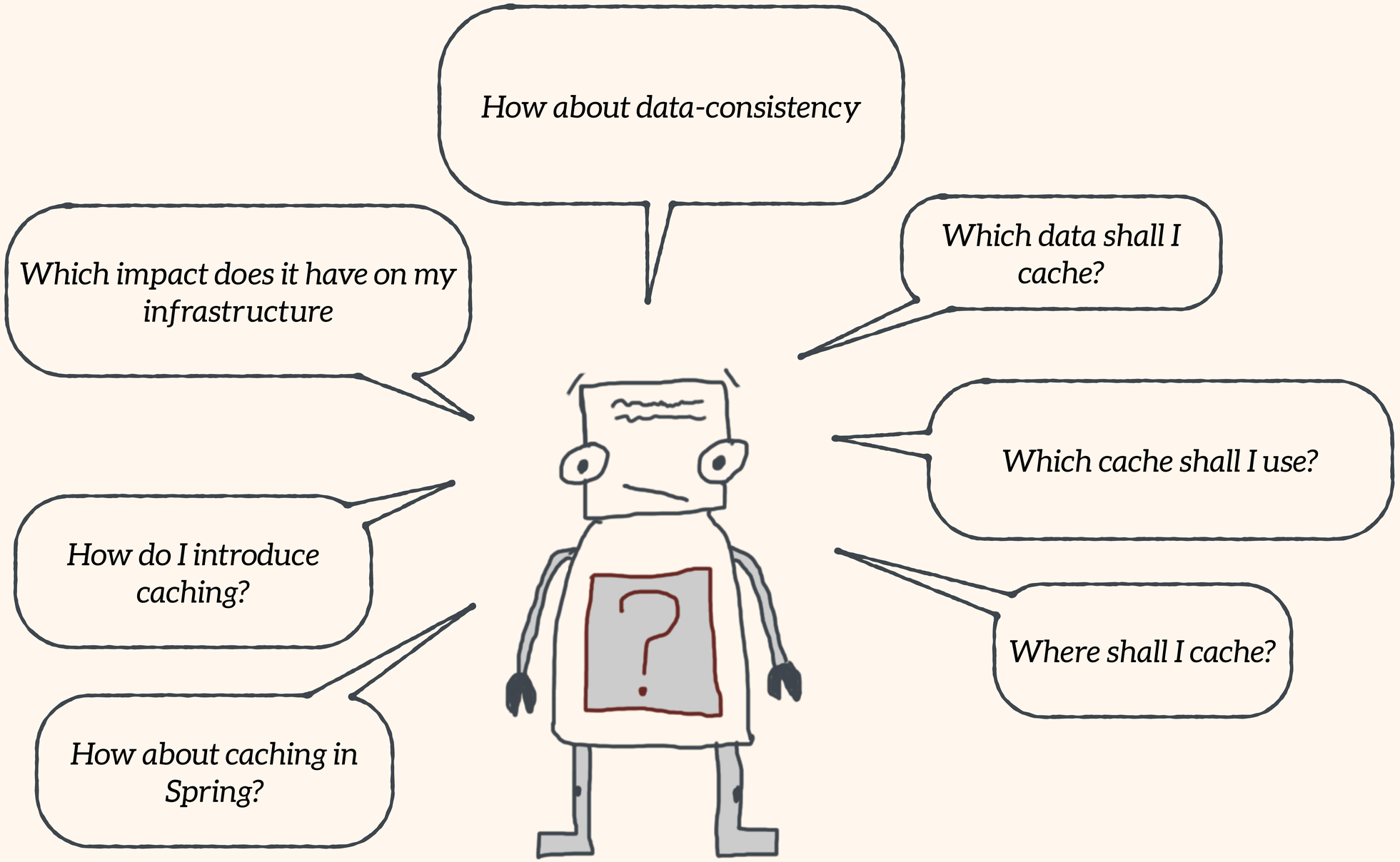
But with a well adjusted caching strategy you will make your application more scalable, faster and cheaper to operate.

Local Cache, Data Grid, Document Store, JPA First Level Cache, JPA Second Level Cache, Hybrid Cache

Types of CACHES
Places for CACHES

Database, Heap, HTTP Proxy, Browser, Prozessor, Disk, Off Heap, Persistence-Framework, Application

We will focus on local and distributed caching at the application level with the Spring Framework


How about data-consistency
Which impact does it have on my infrastructure
Which data shall I cache?
How do I introduce caching?
Which cache shall I use?
How about caching in Spring?
Where shall I cache?

How about data-consistency

Which impact does it have on my infrastructure

Which data shall I cache?

Which cache shall I use?

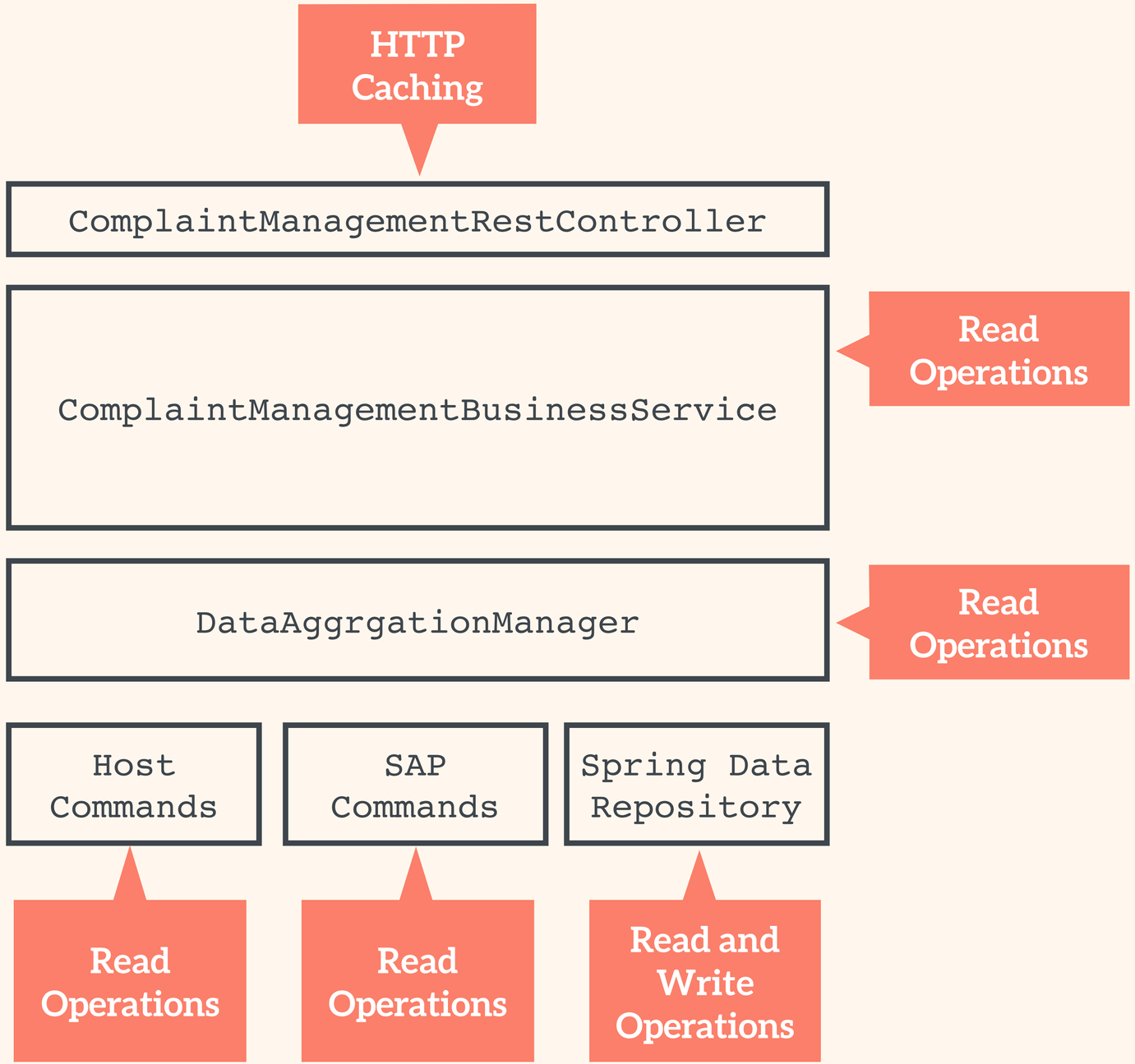
Where shall I cache?

How do I introduce caching?

How about caching in Spring?

1

Identify suitable layers for caching

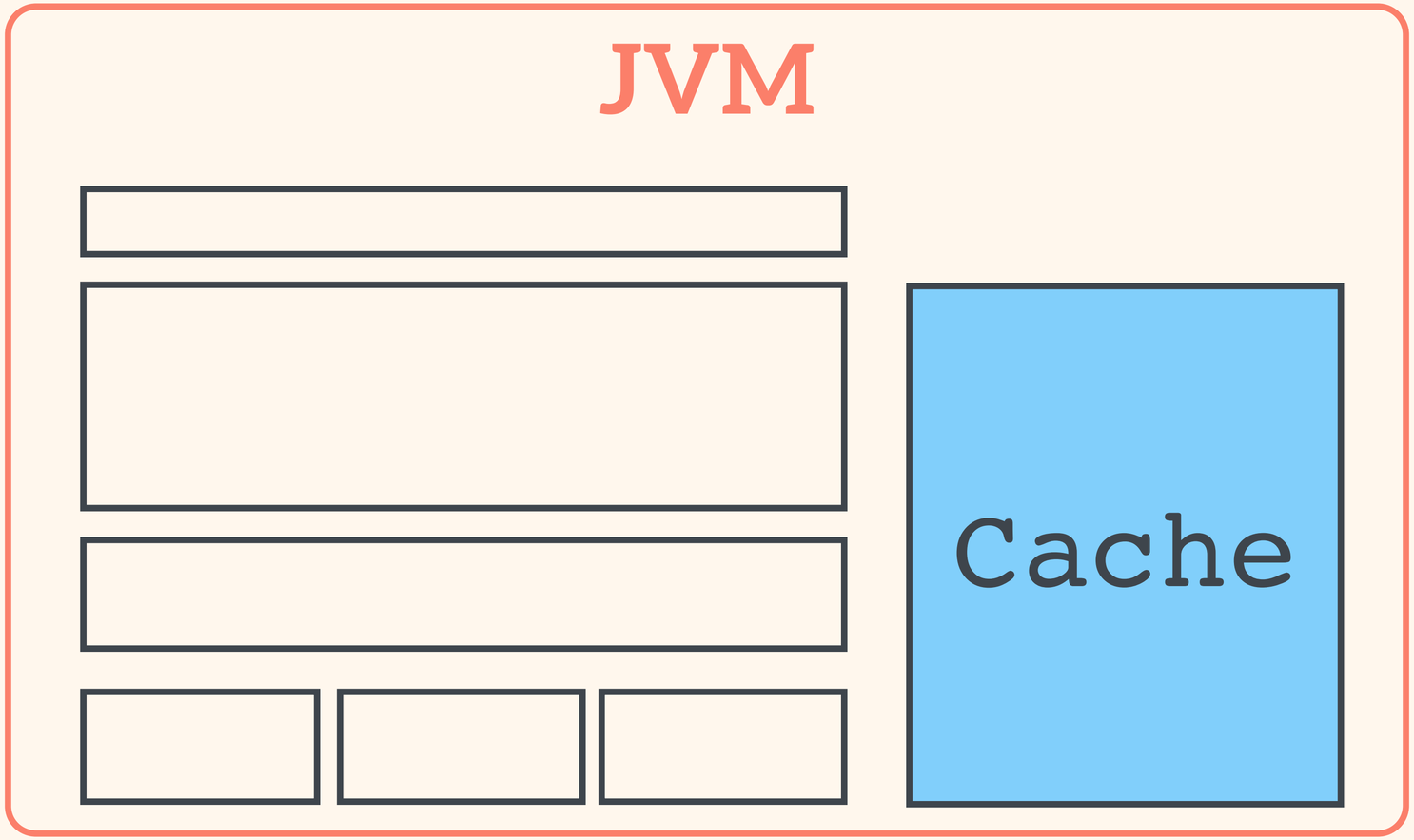


Suitable Layers for Caching

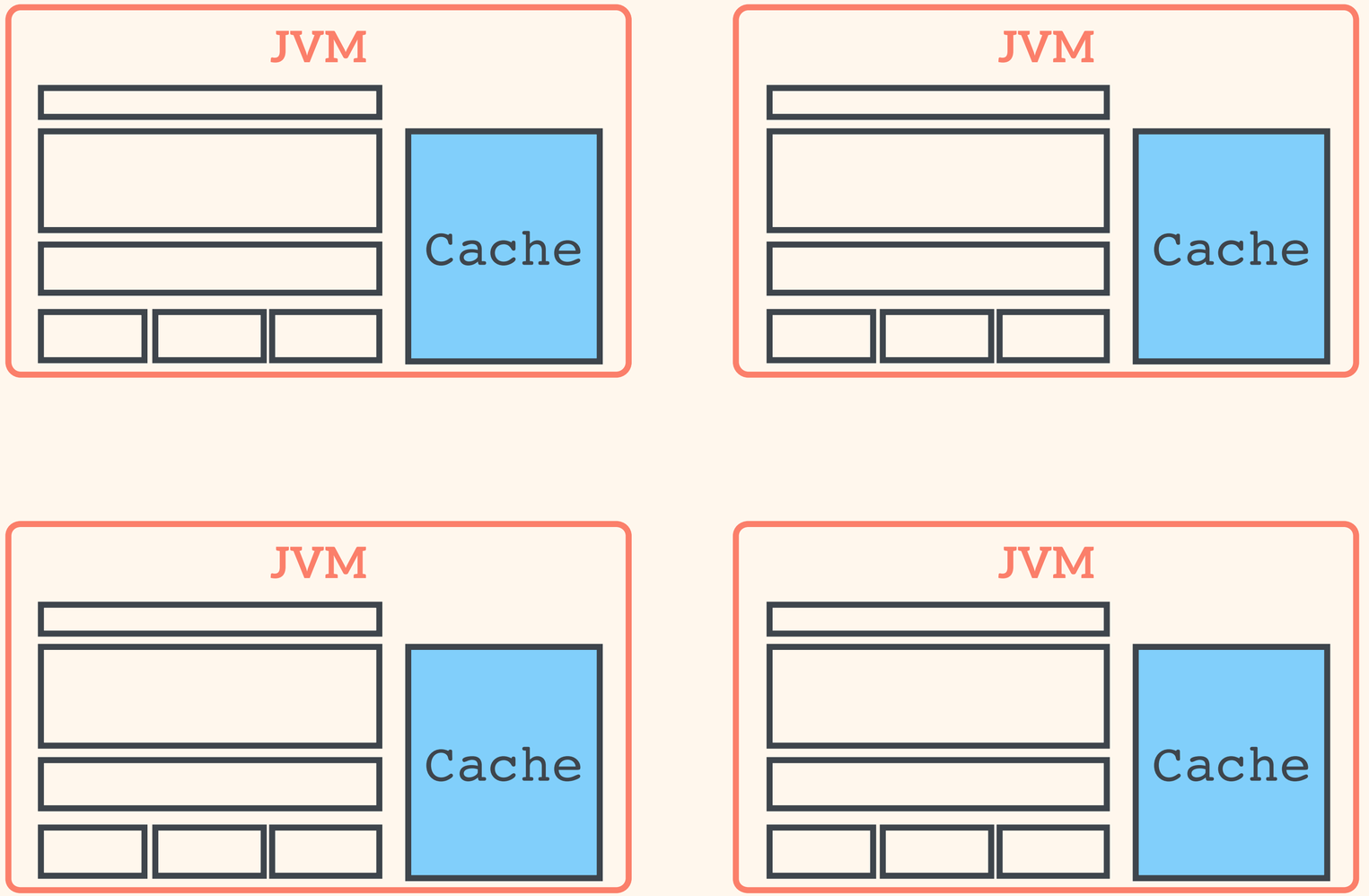
2

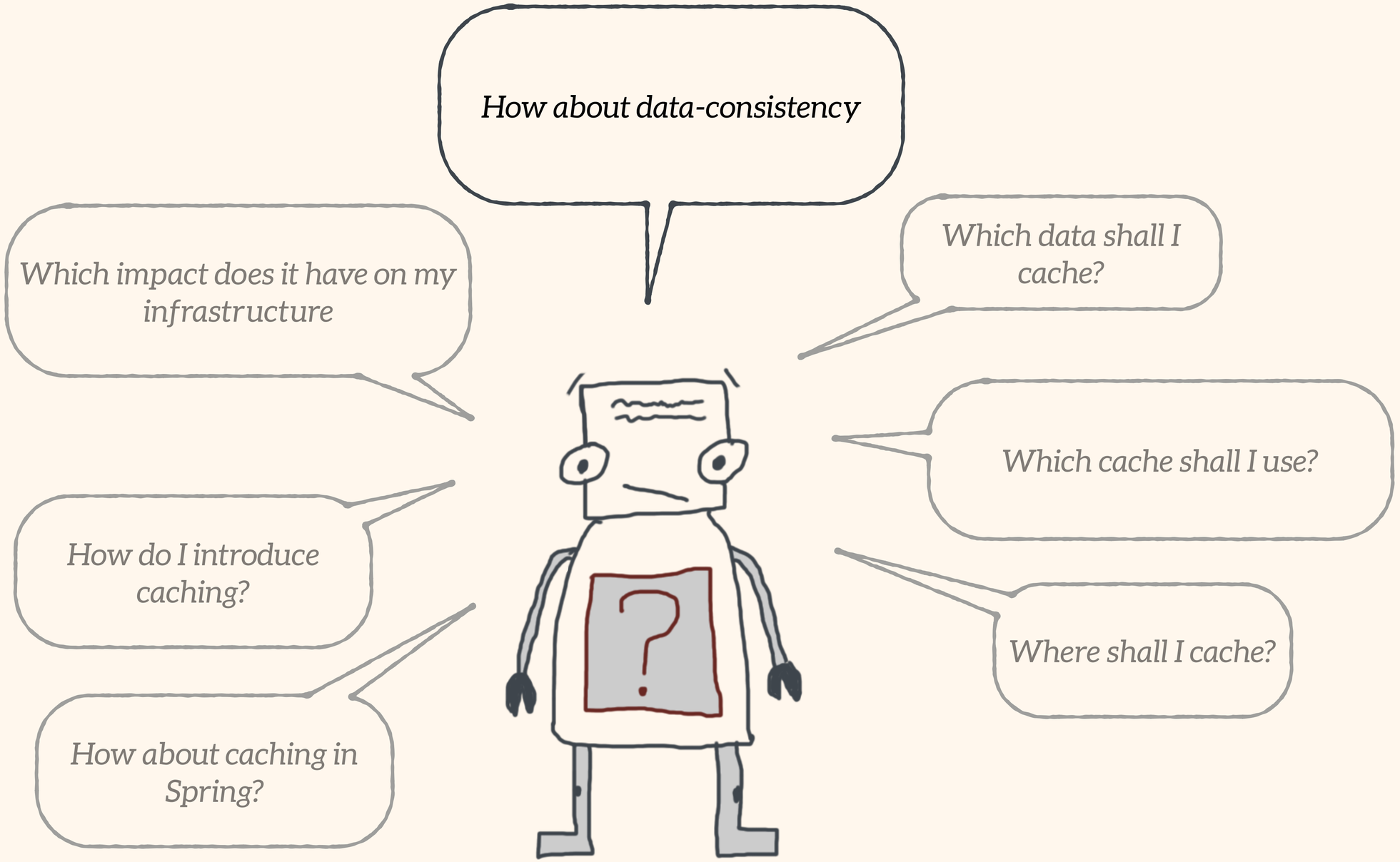
Stay local as long as possible

Lokal In-Memory

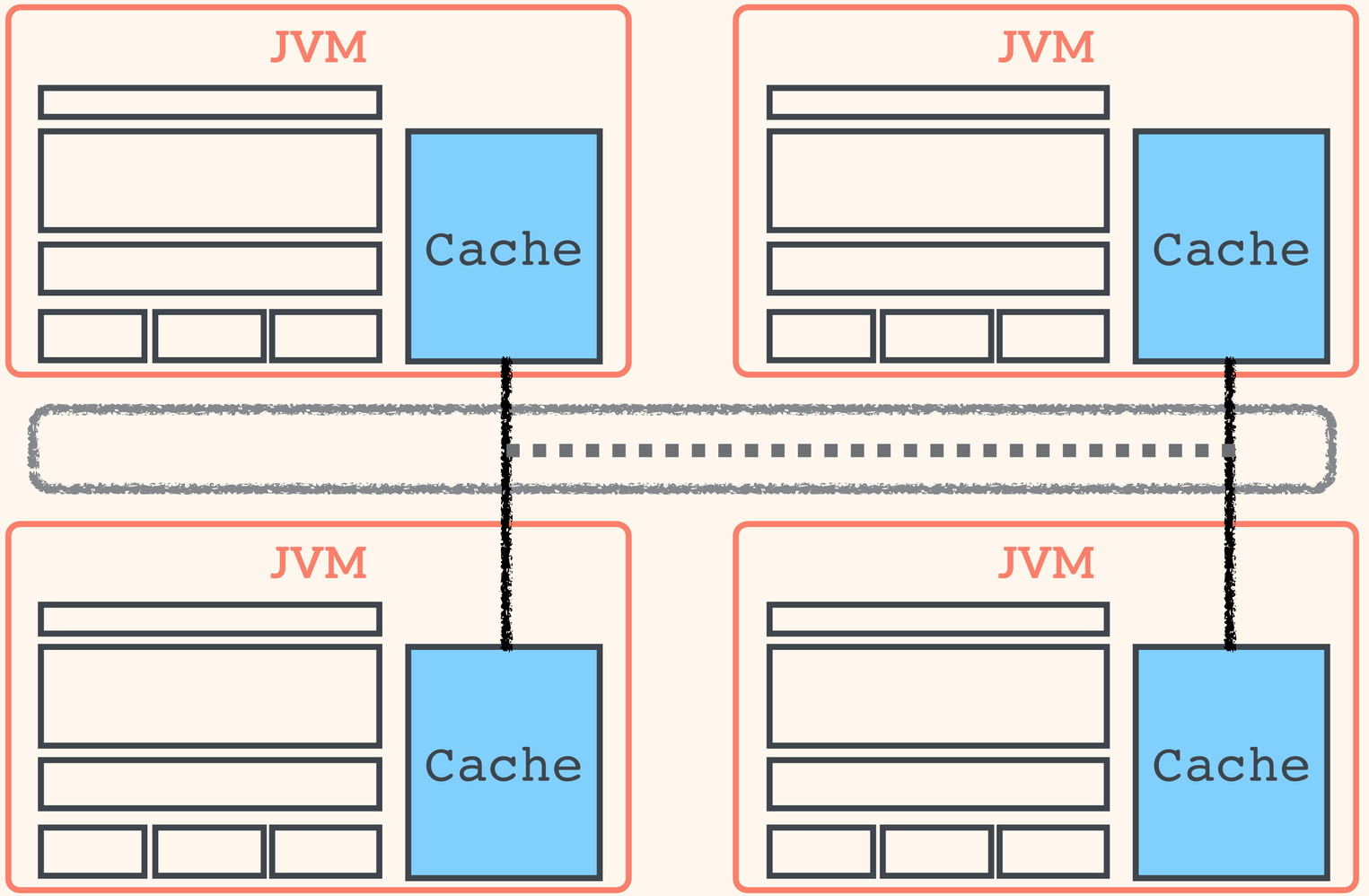


Clustered

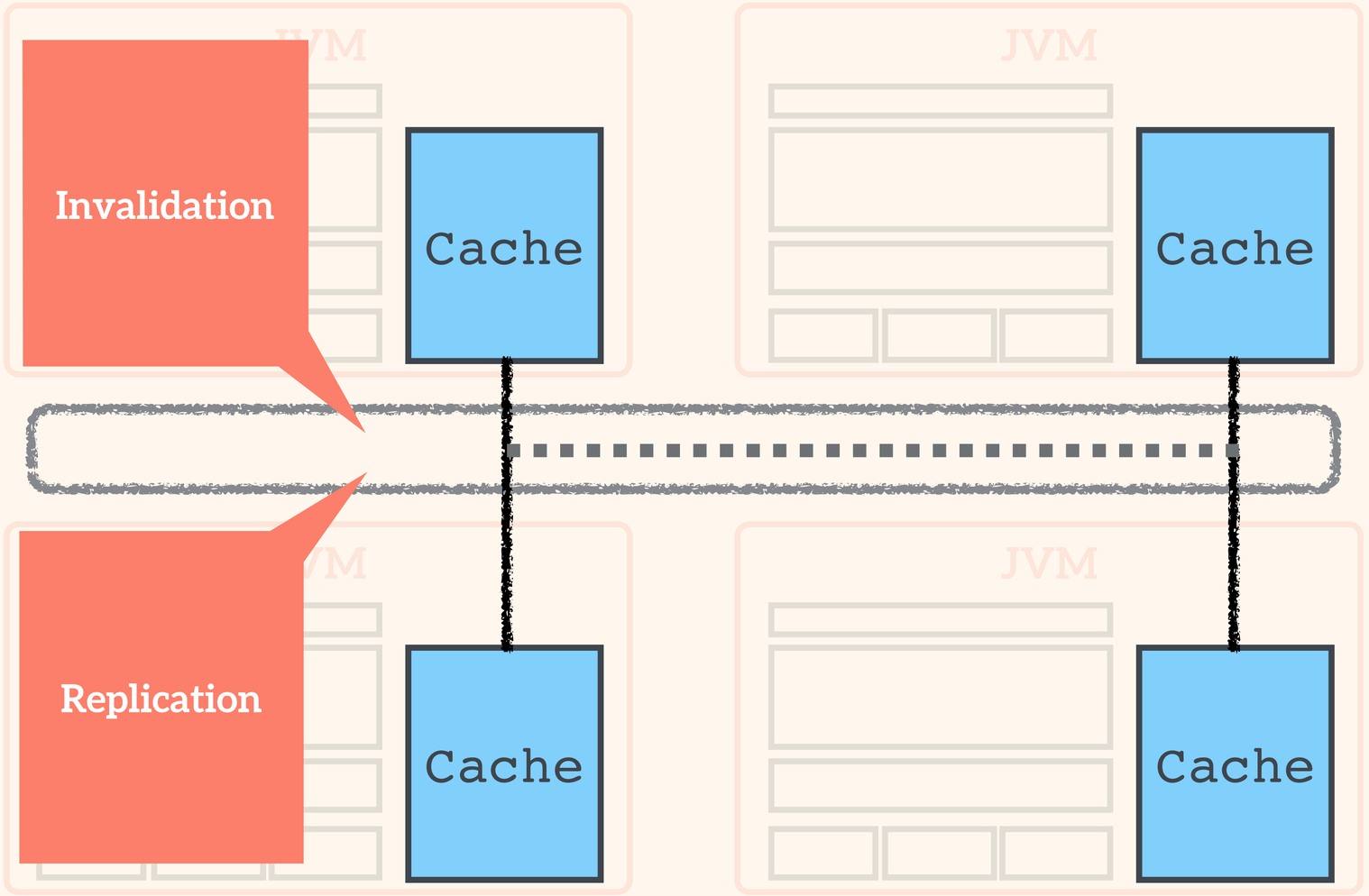



How about data-consistency
Which impact does it have on my infrastructure
Which data shall I cache?
How do I introduce caching?
Which cache shall I use?
How about caching in Spring?
Where shall I cache?

Clustered - with sync



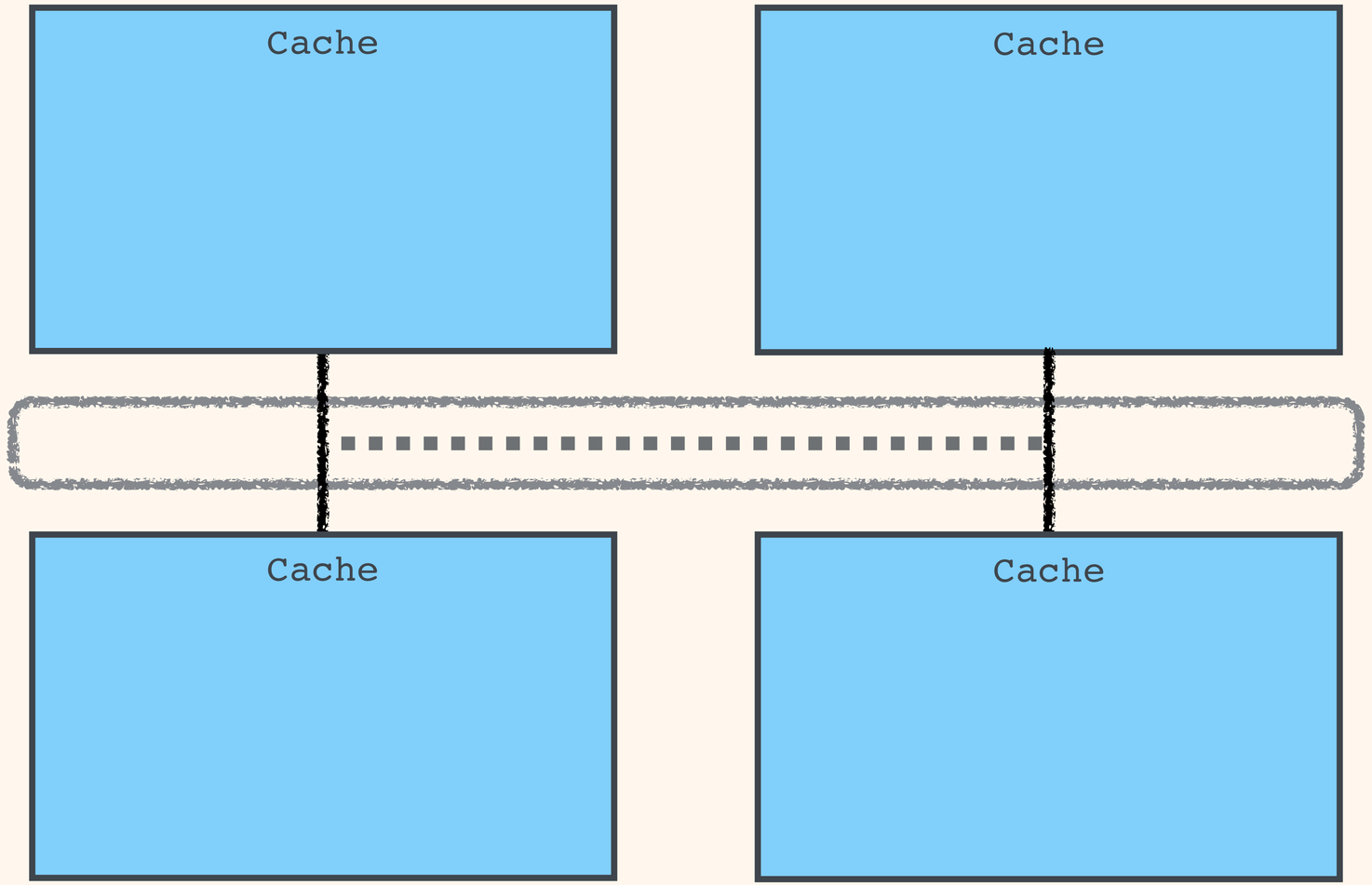
Clustered - with sync



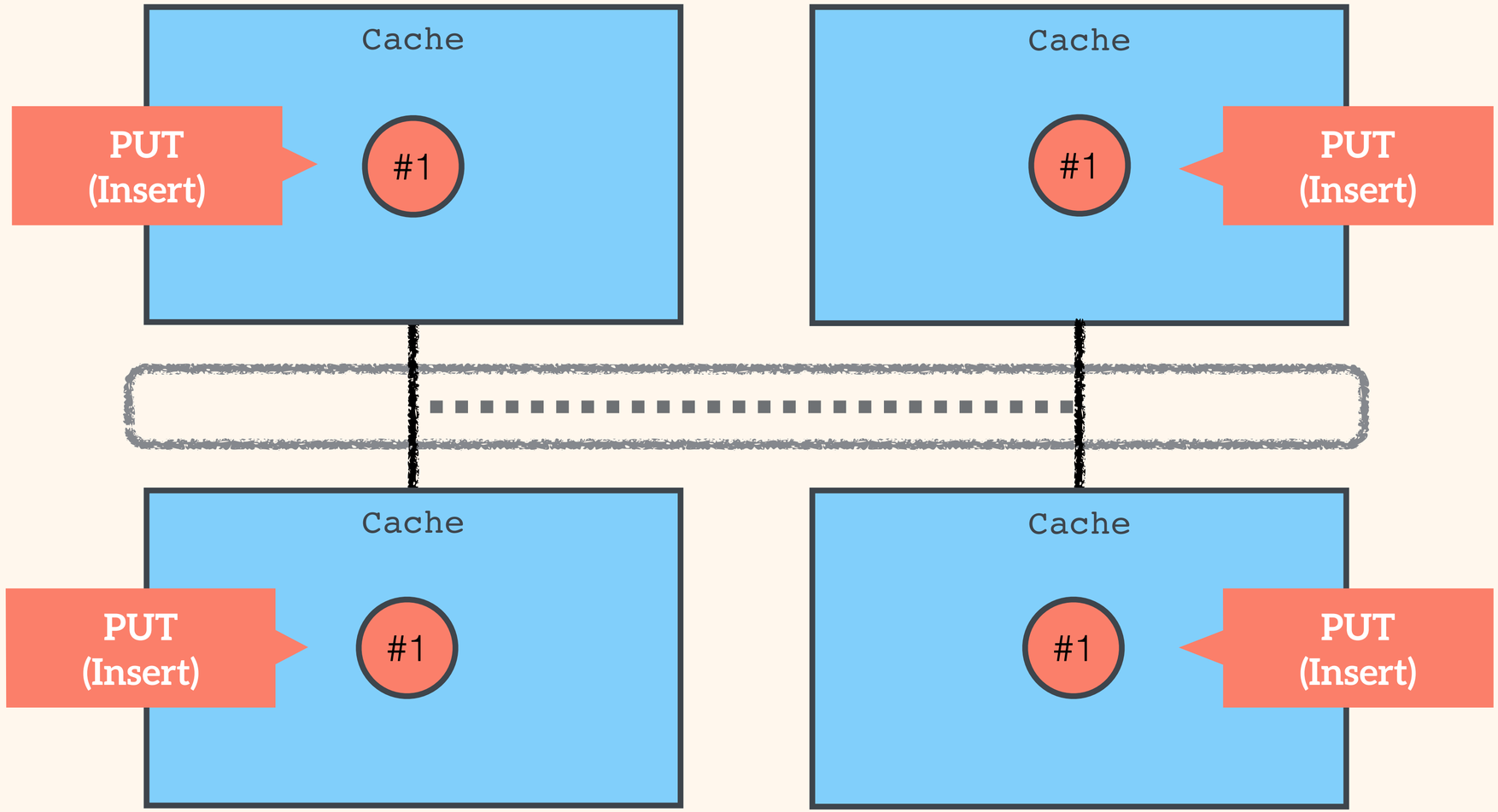
3

Avoid real replication where possible

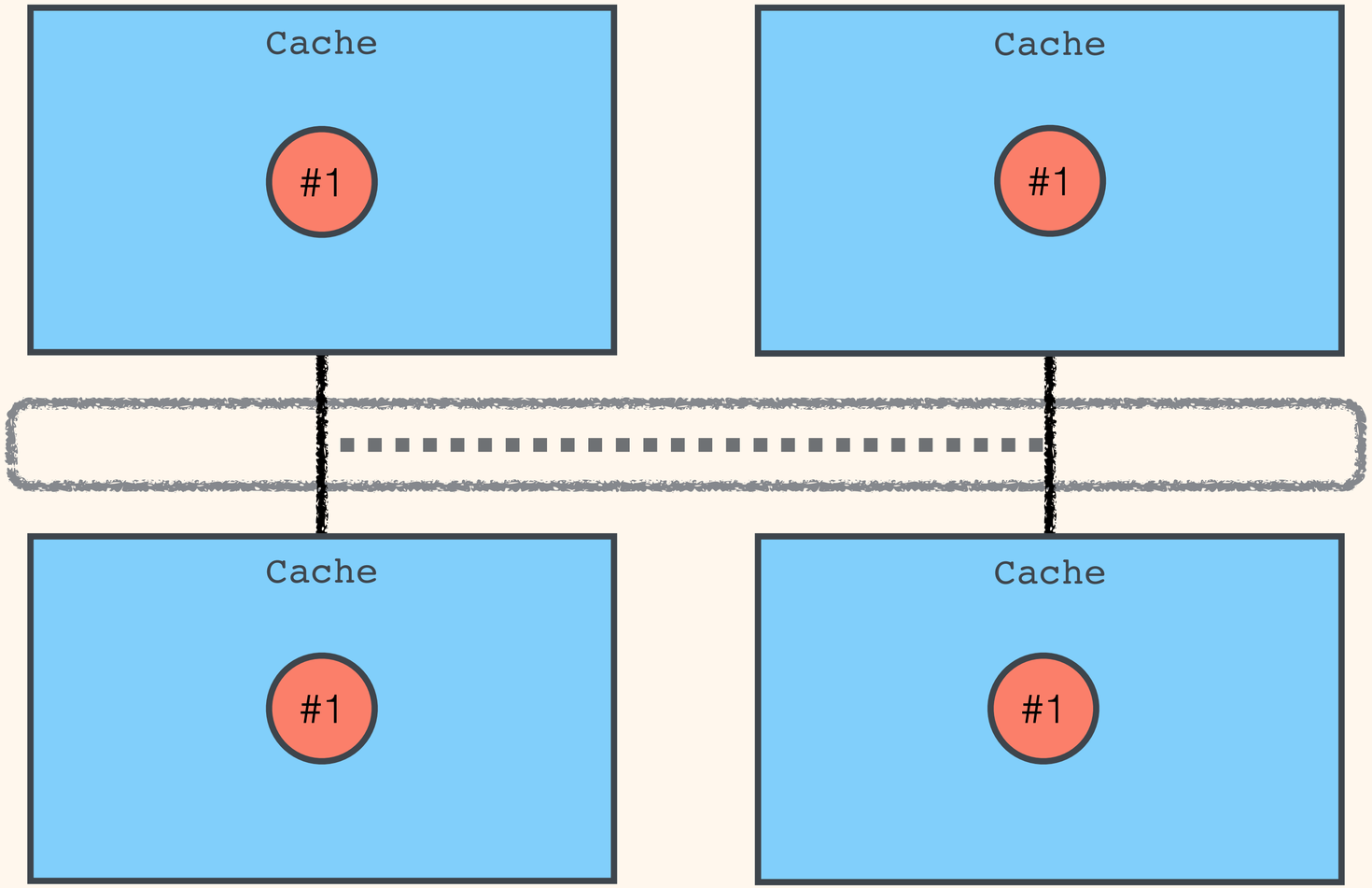
Invalidation - Option 1



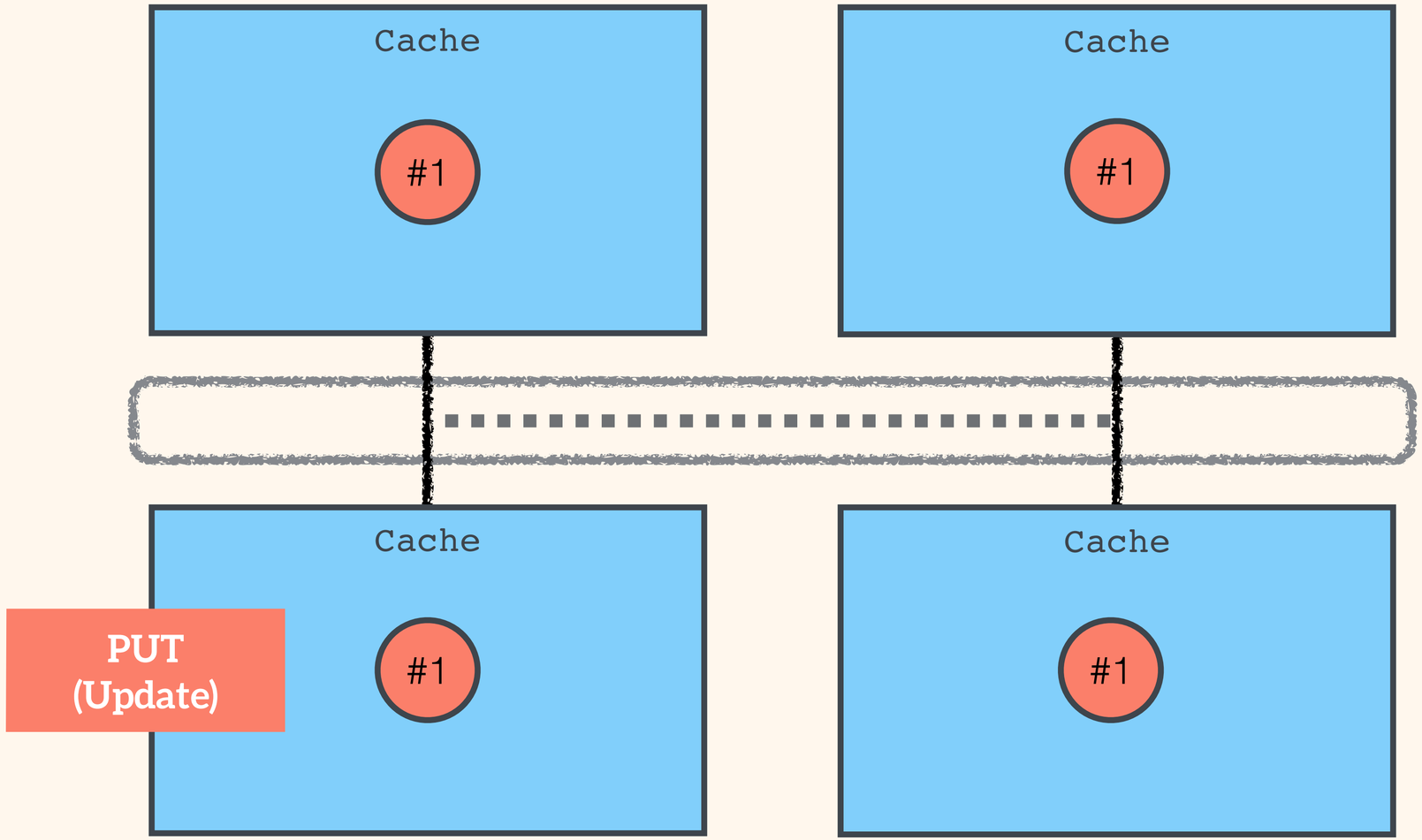
Invalidation - Option 1



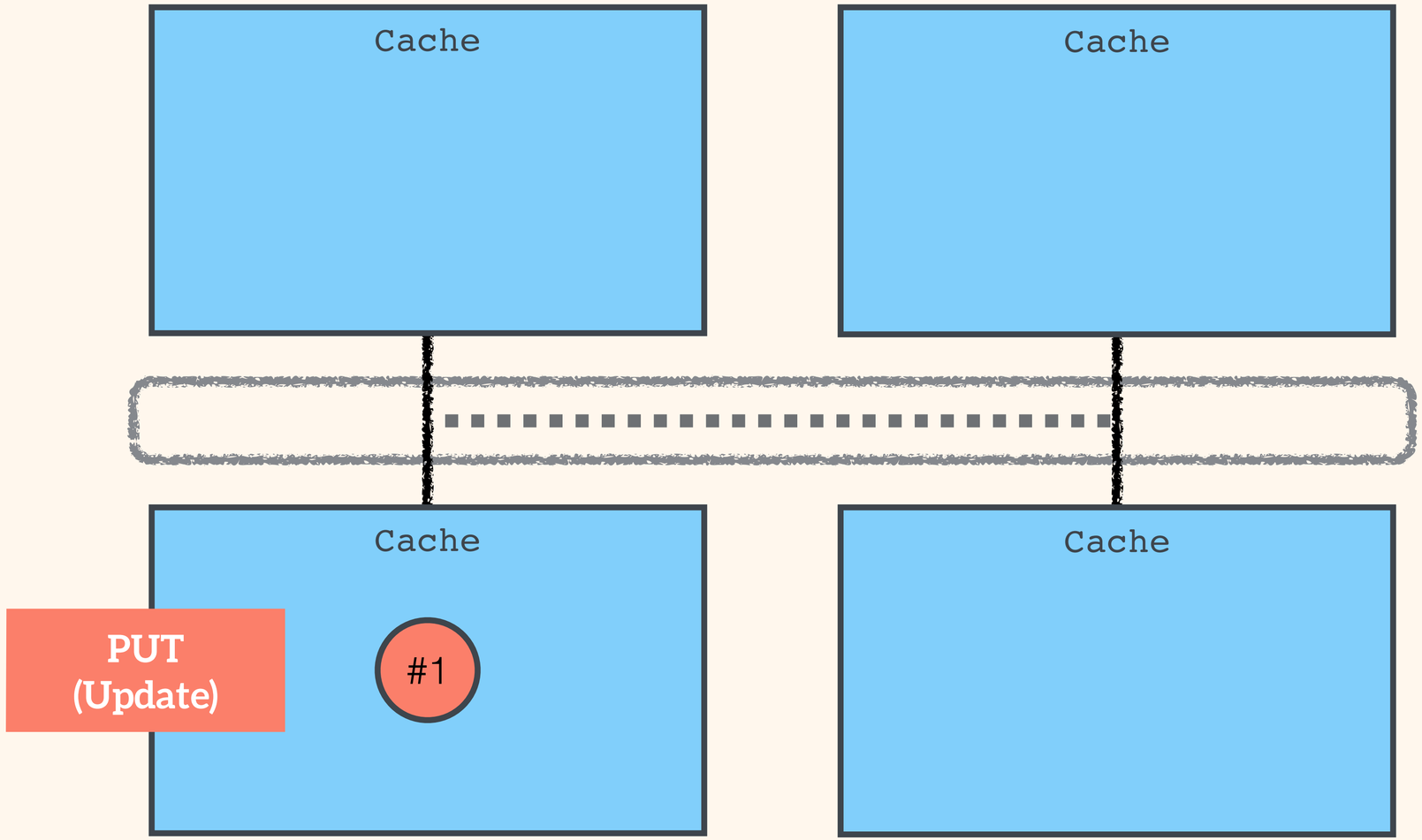
Invalidation - Option 1



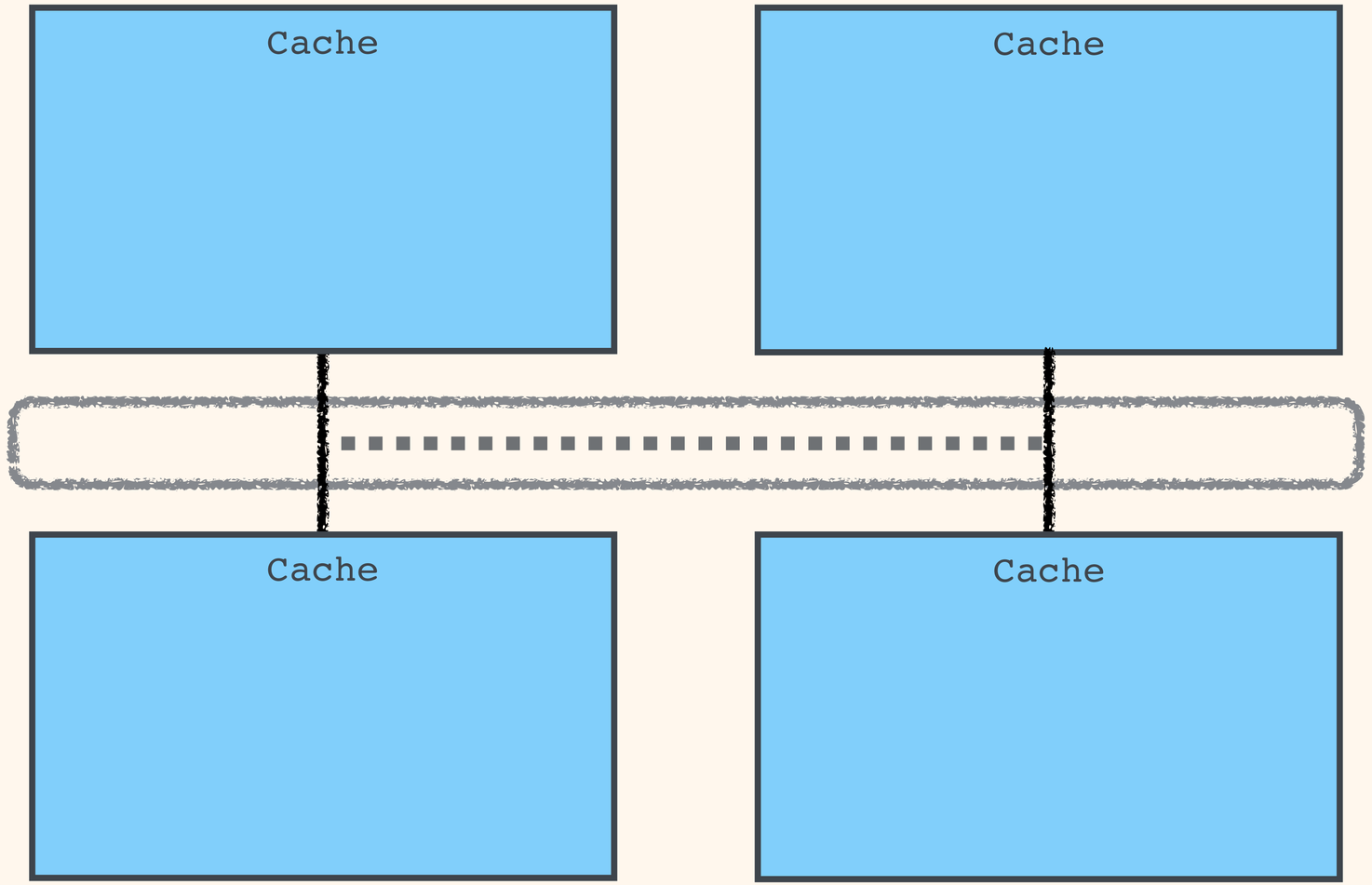
Invalidation - Option 1



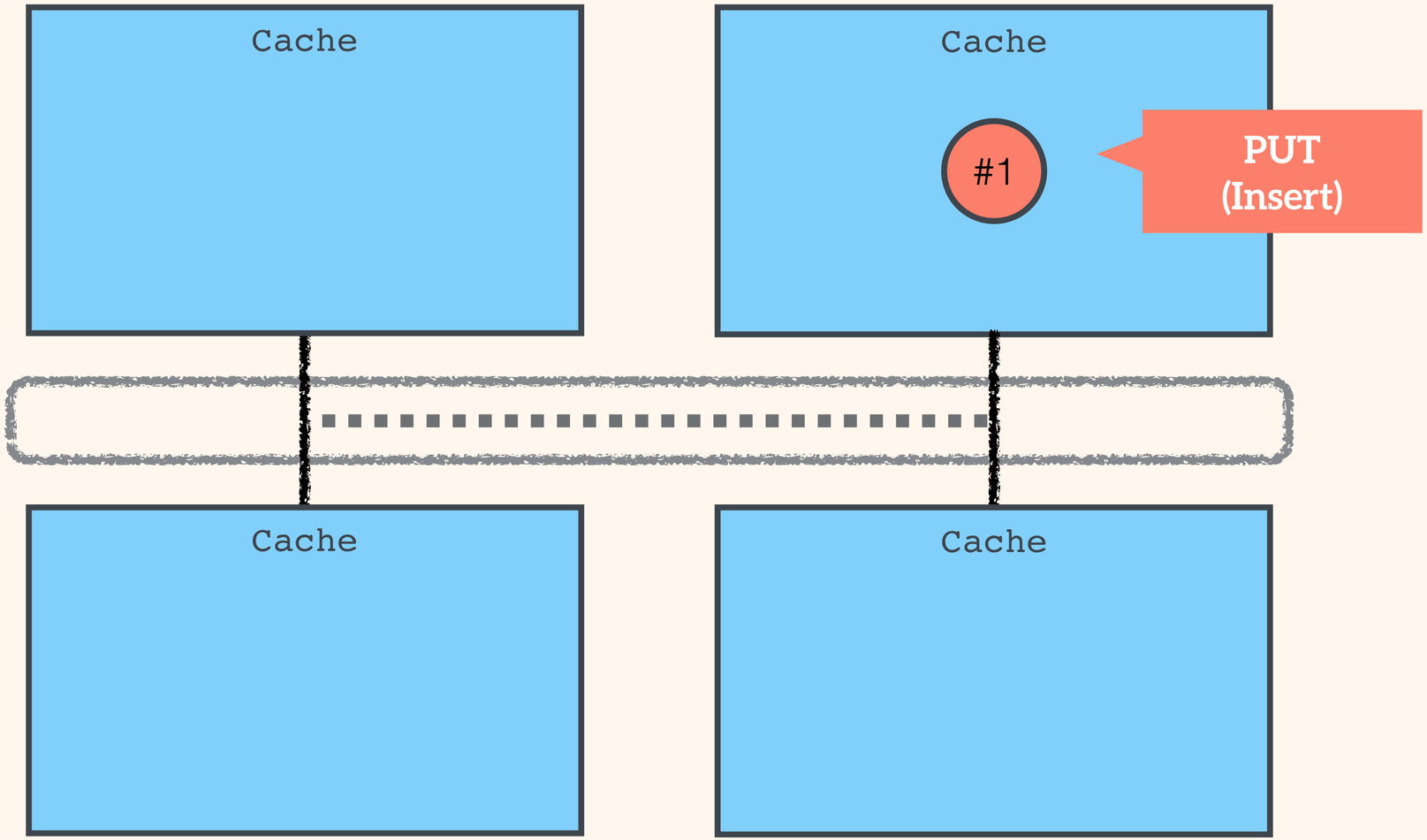
Invalidation - Option 1



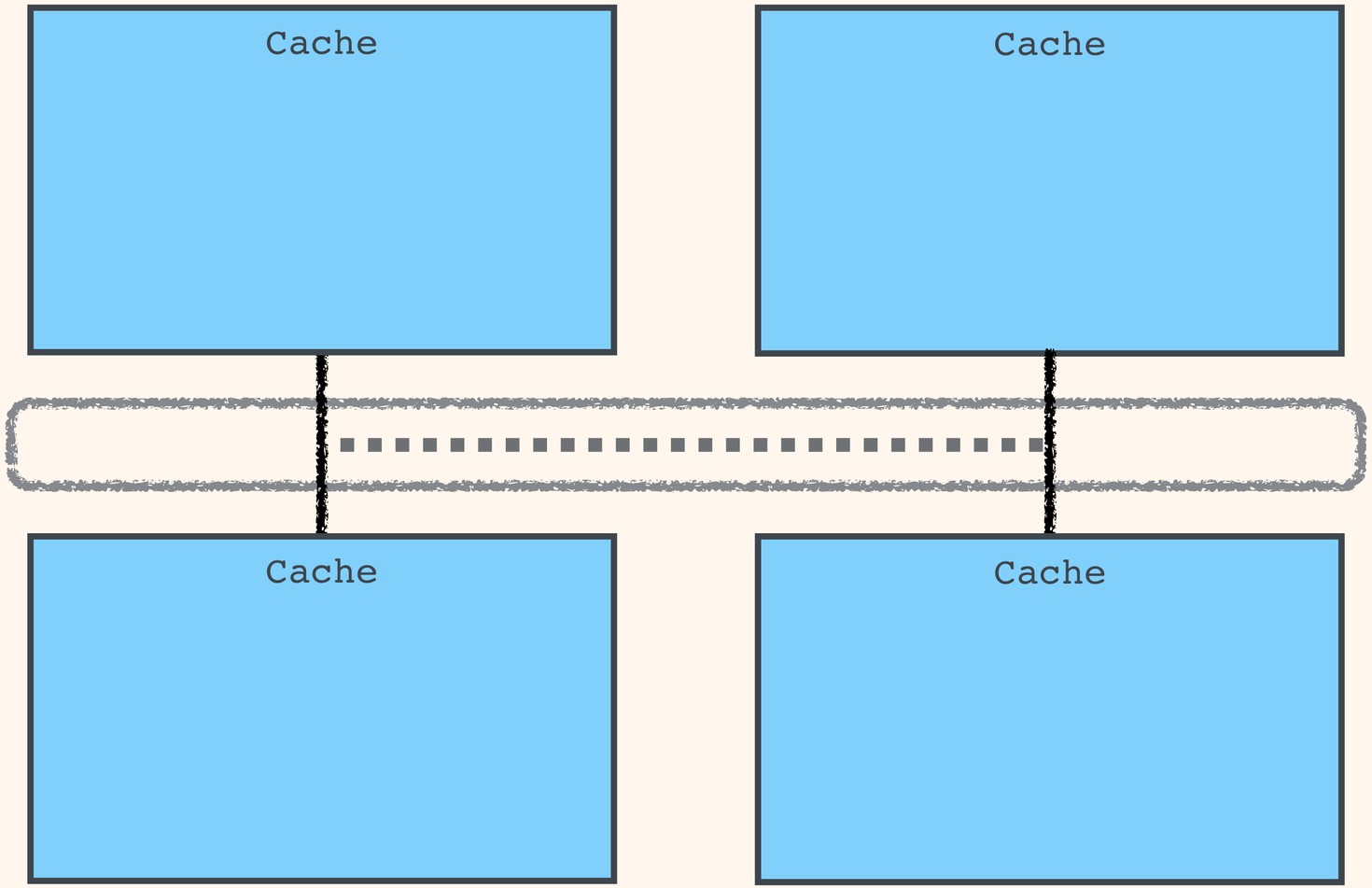
Invalidation - Option 2



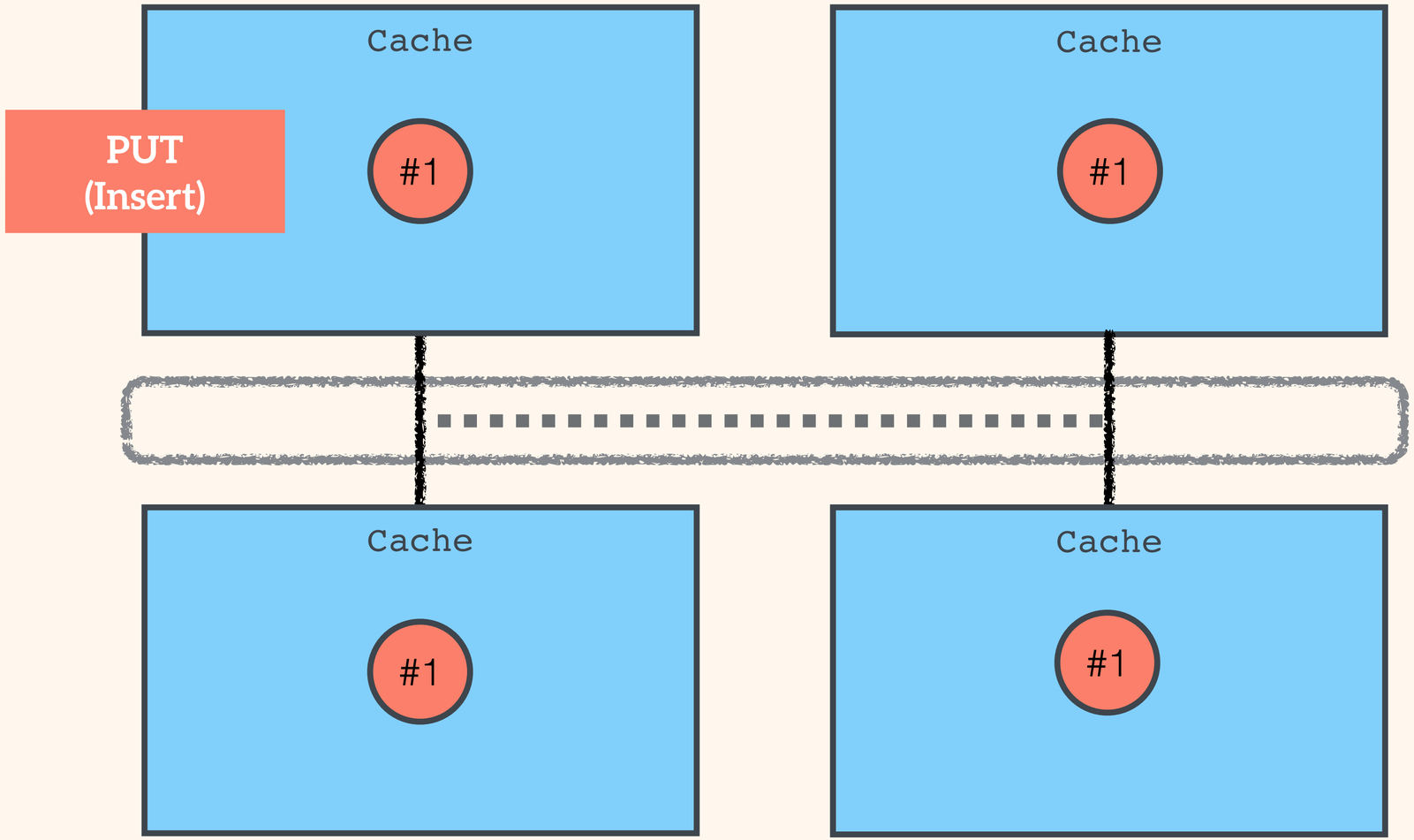
Invalidation - Option 2



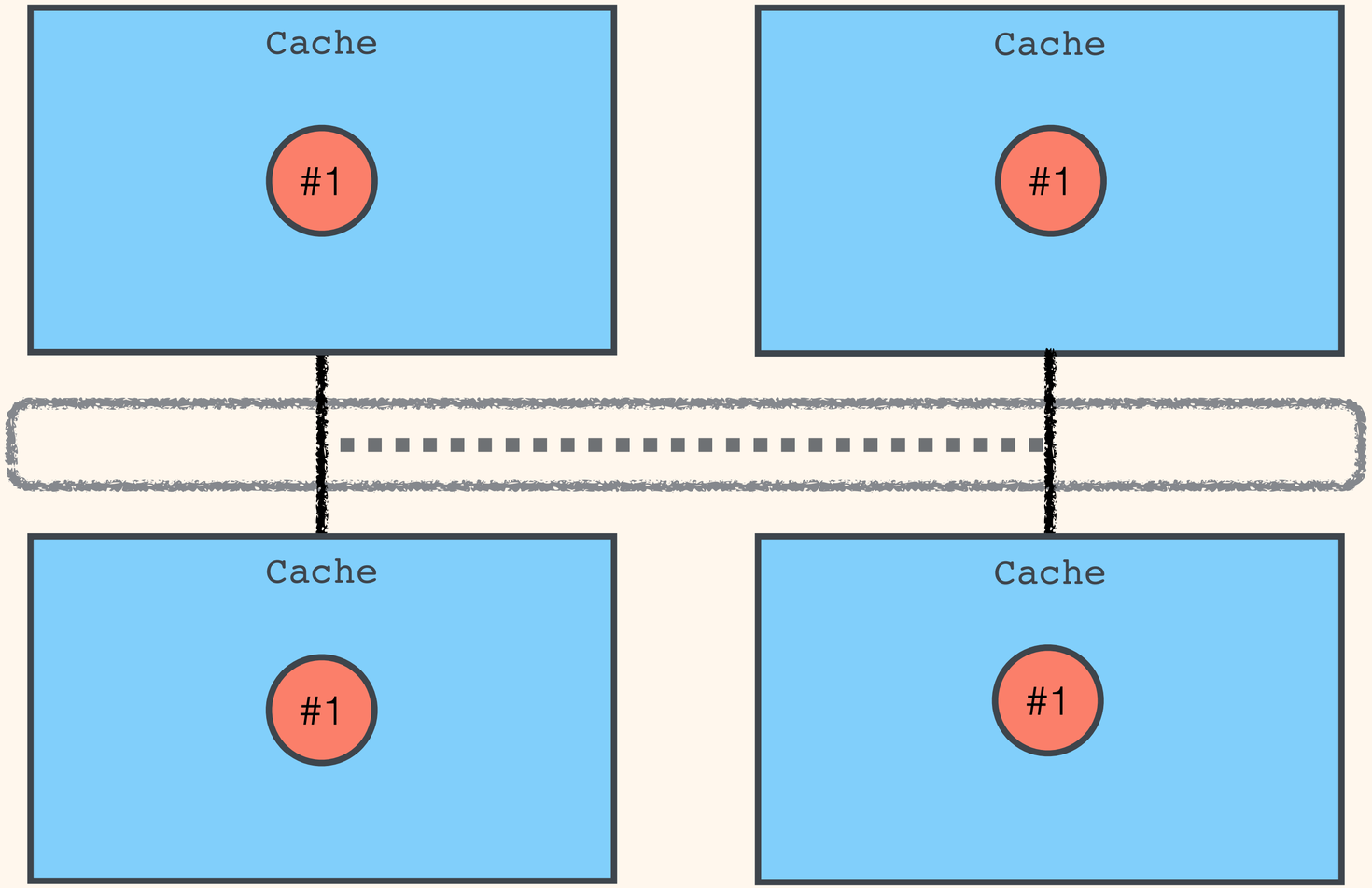
Replication



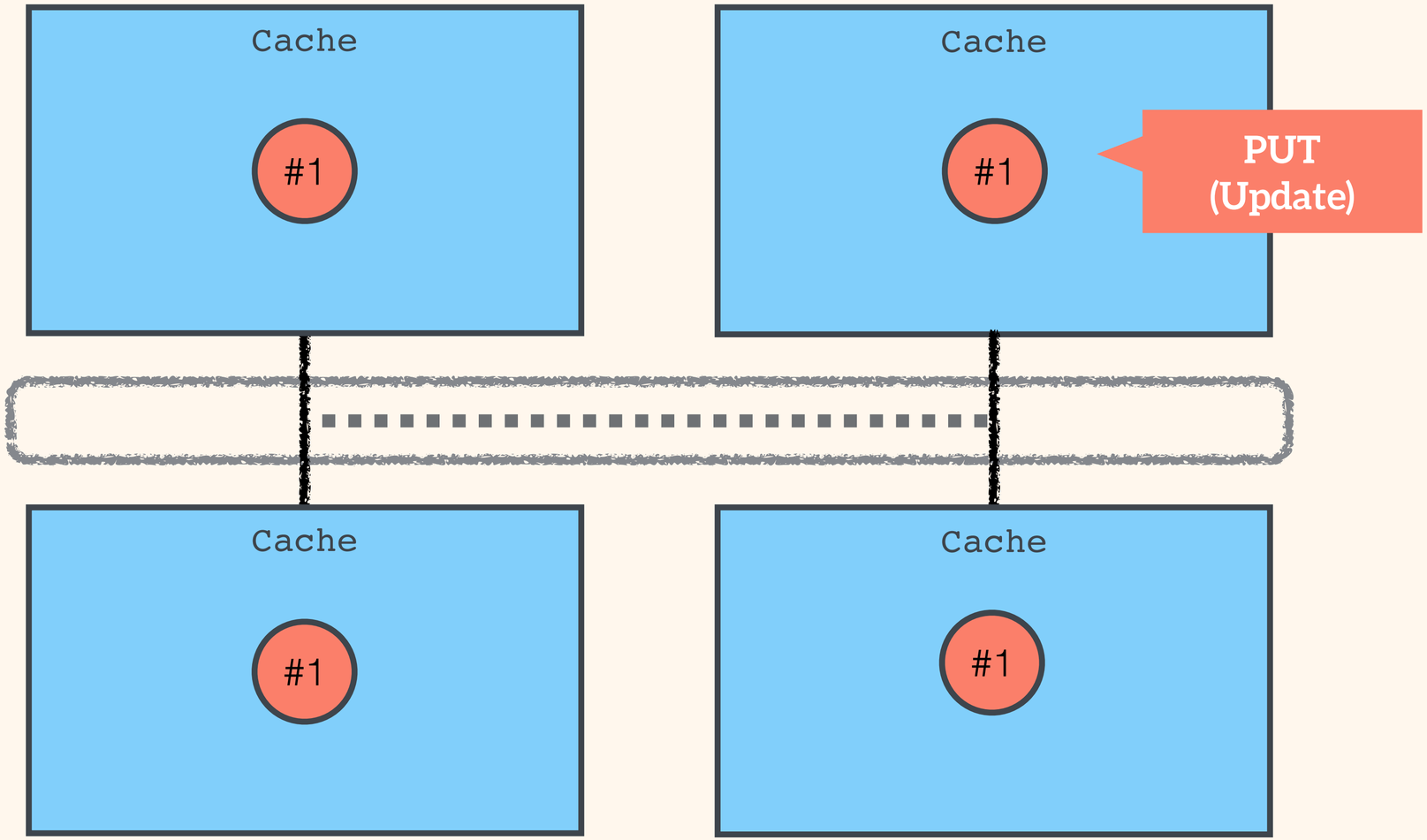
Replication



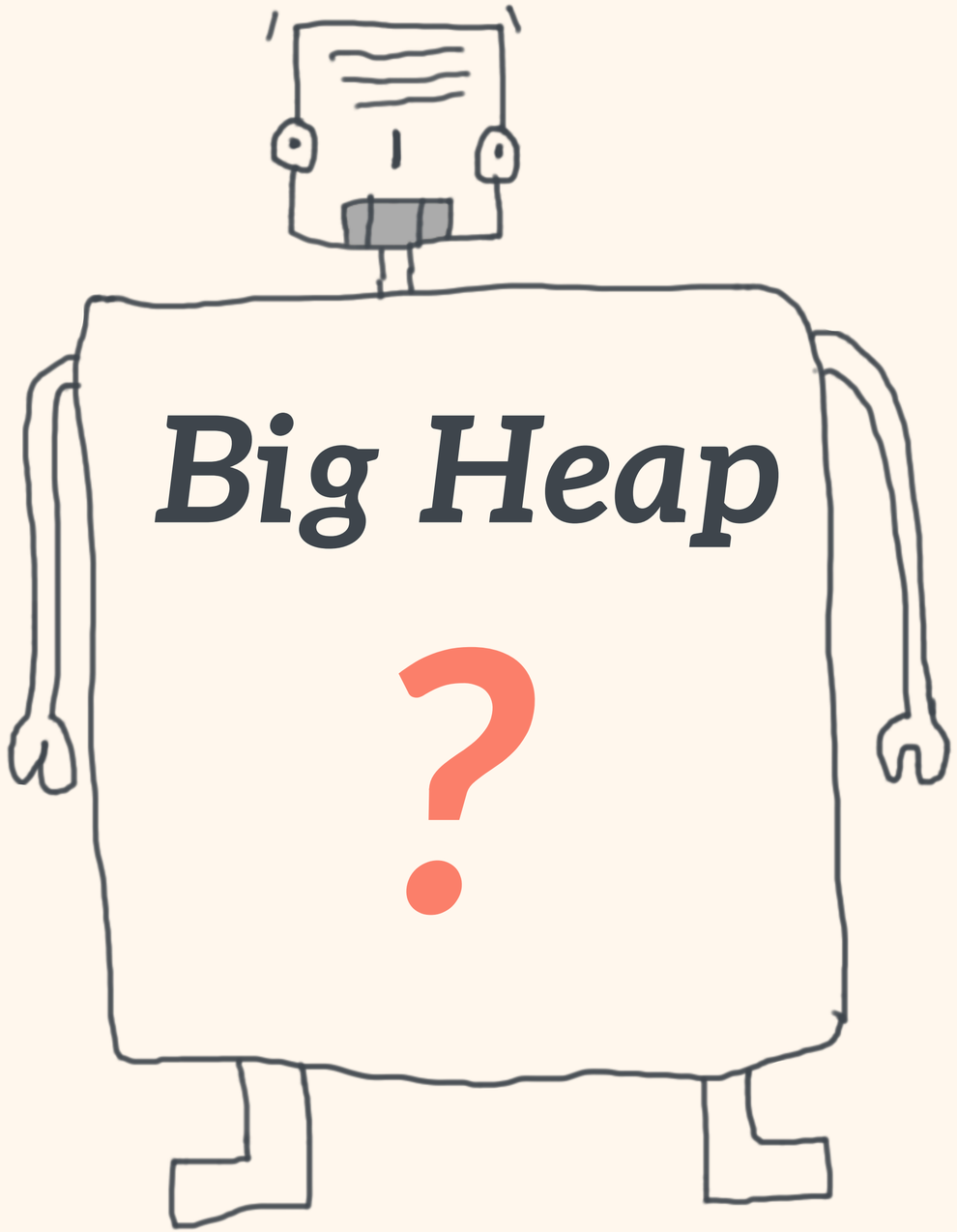
Replication



Replication



As of now every cache could potentially hold every data which consumes heap memory


Big Heap
?

How about data-consistency

Which impact does it have on my infrastructure

Which data shall I cache?

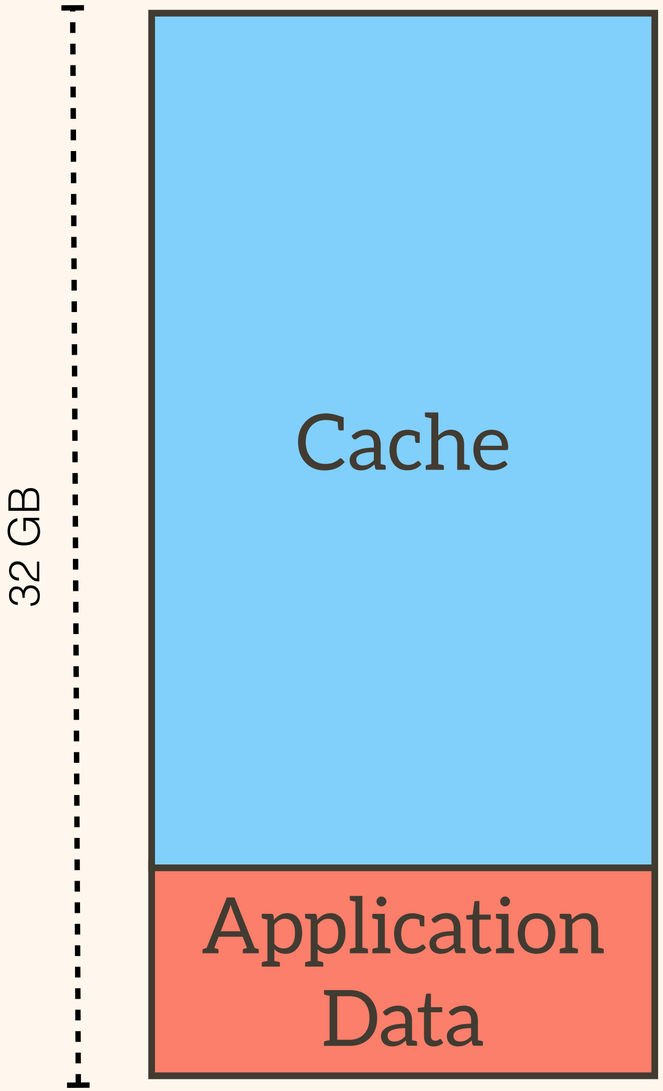
Which cache shall I use?

How do I introduce caching?

Where shall I cache?

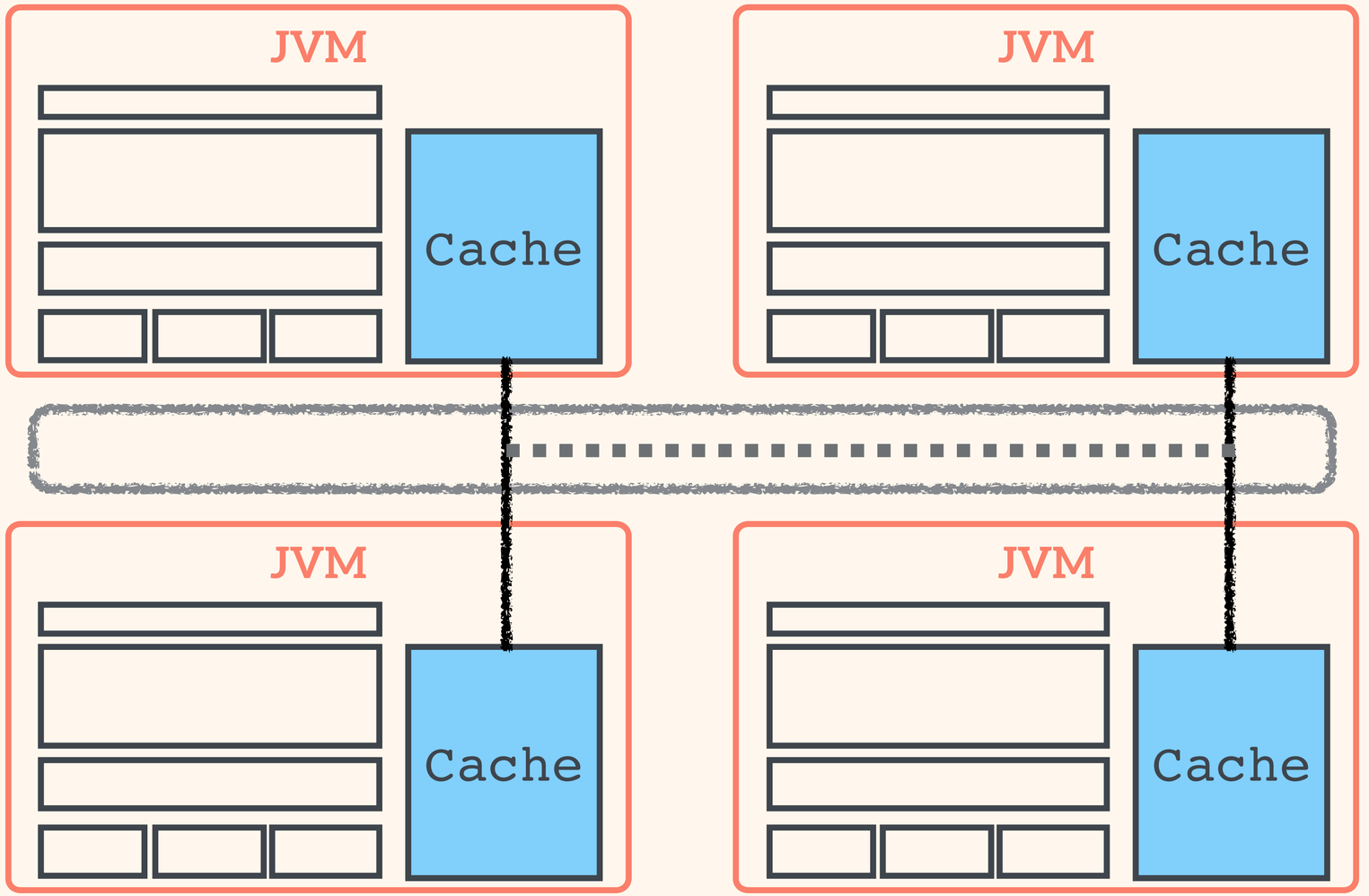
How about caching in Spring?

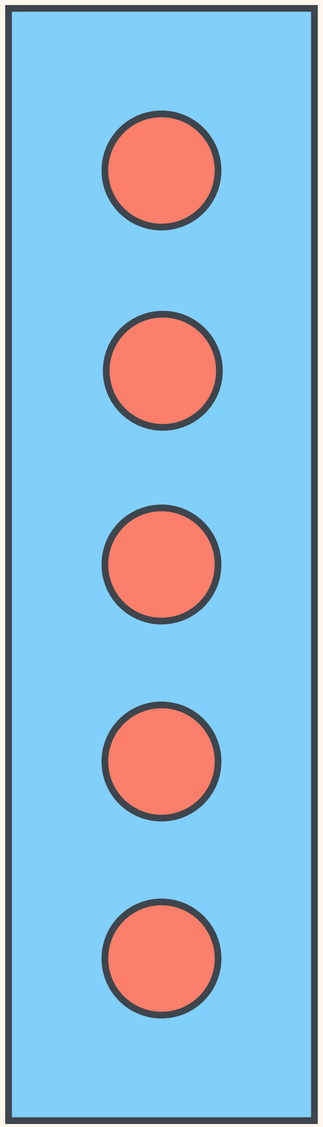
Avoid big heaps just for caching



Big heap leads to long major GCs

Long GCs can destabilize your cluster





Small caches are a bad idea!

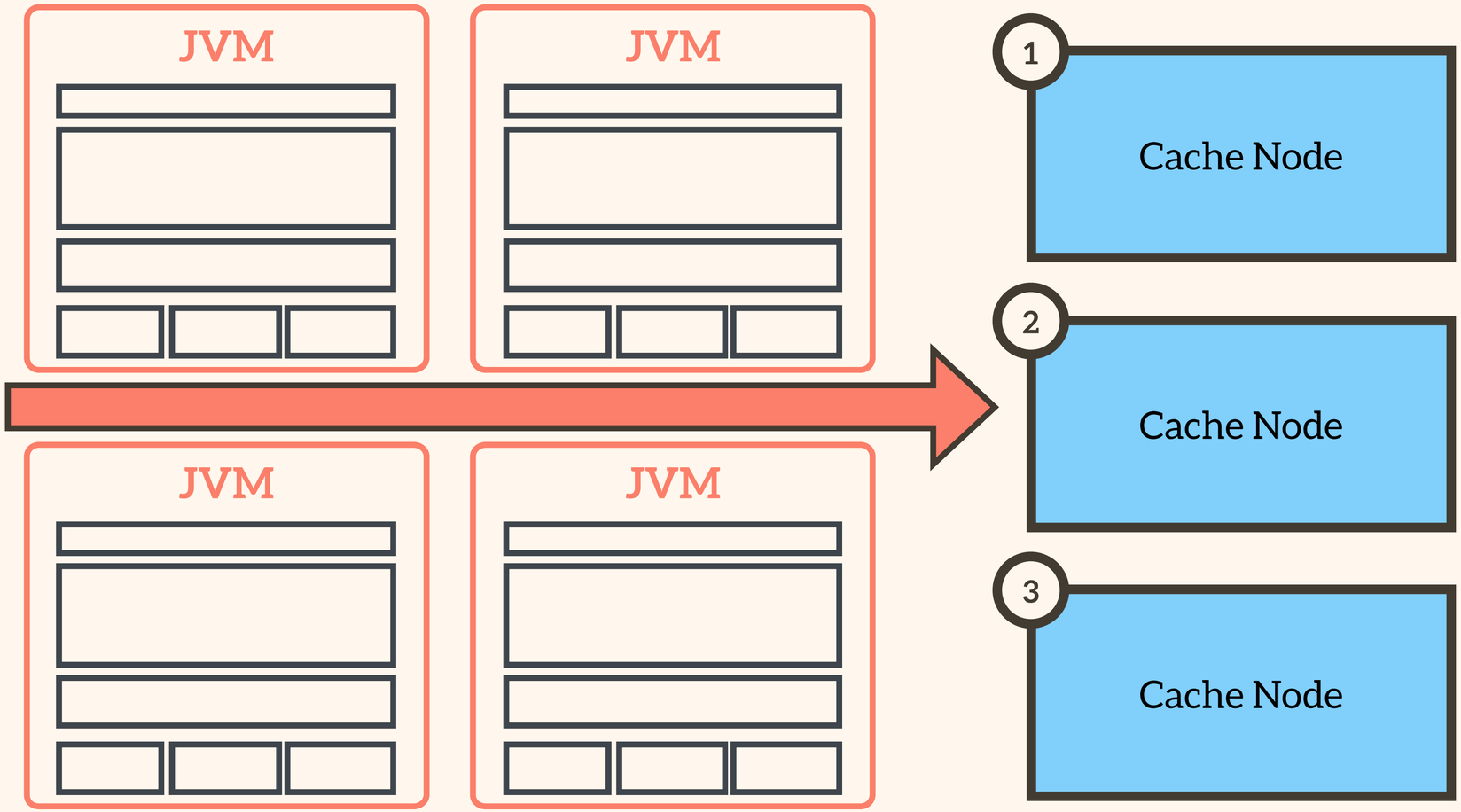
Many evictions, fewer hits,
no „hot data“.

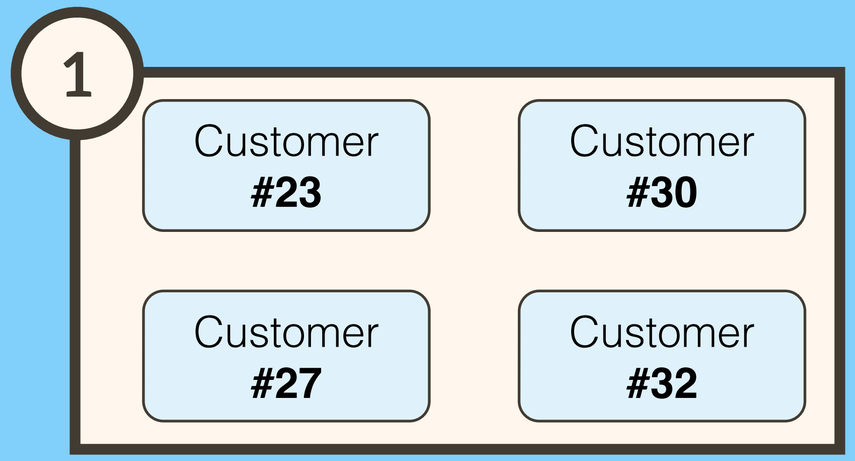
This is especially critical for
replicating caches.

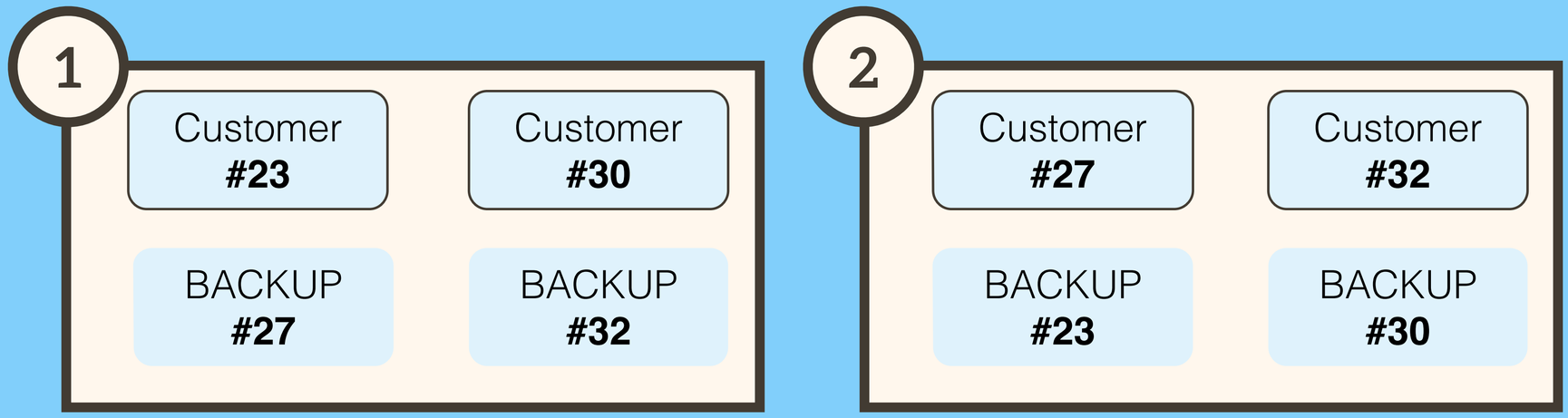
5

Use a distributed cache for big amounts of data

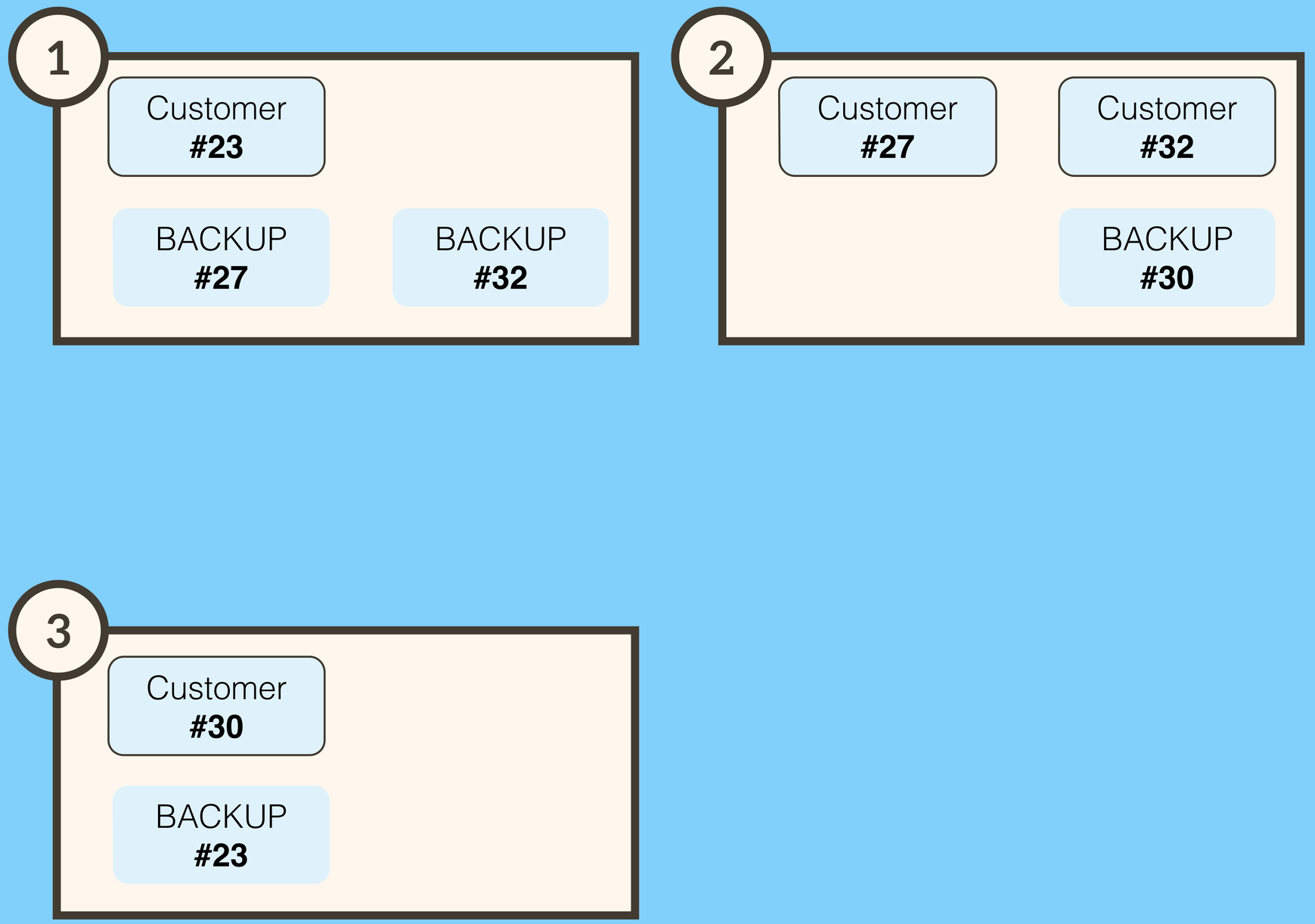
Distributed Caches

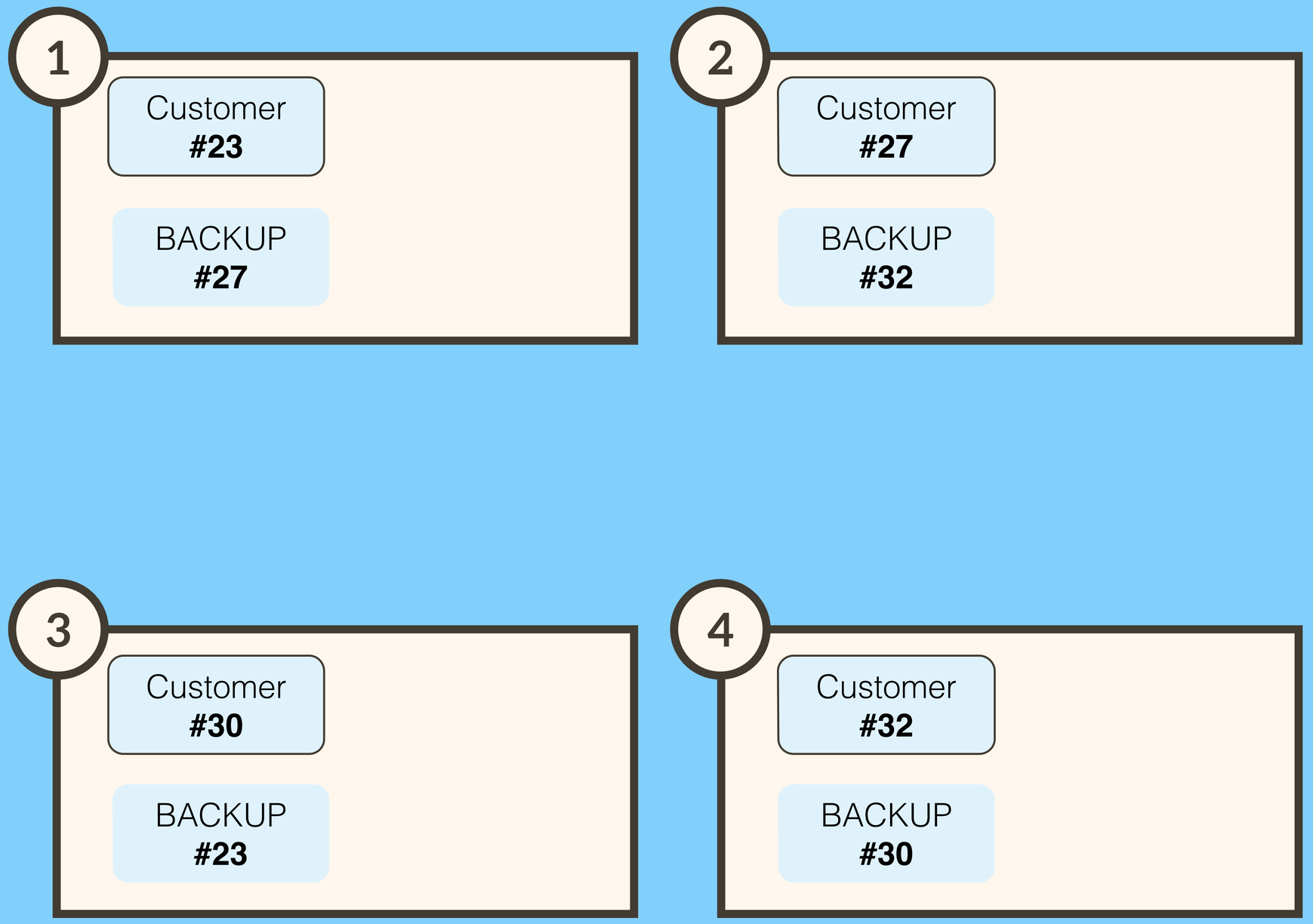



1
Customer
#23
Customer
#30
Customer
#27
Customer
#32



Data is being distributed and backed up

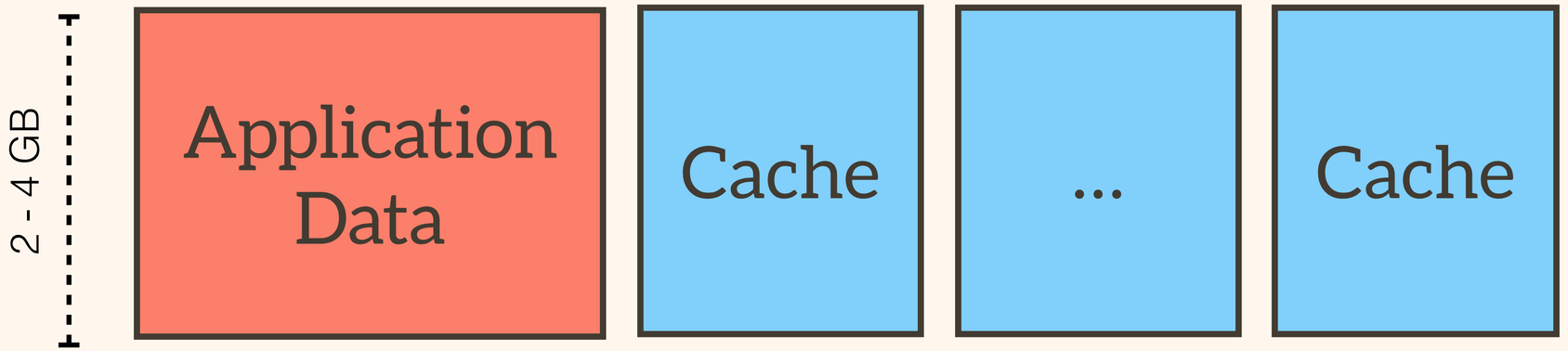

1
Customer
#23
BACKUP
#27
BACKUP
#32
2
Customer
#27
Customer
#32
BACKUP
#30
3
Customer
#30
BACKUP
#23


1
Customer
#23
BACKUP
#27
2
Customer
#27
BACKUP
#32
3
Customer
#30
BACKUP
#23
4
Customer
#32
BACKUP
#30

DEMO

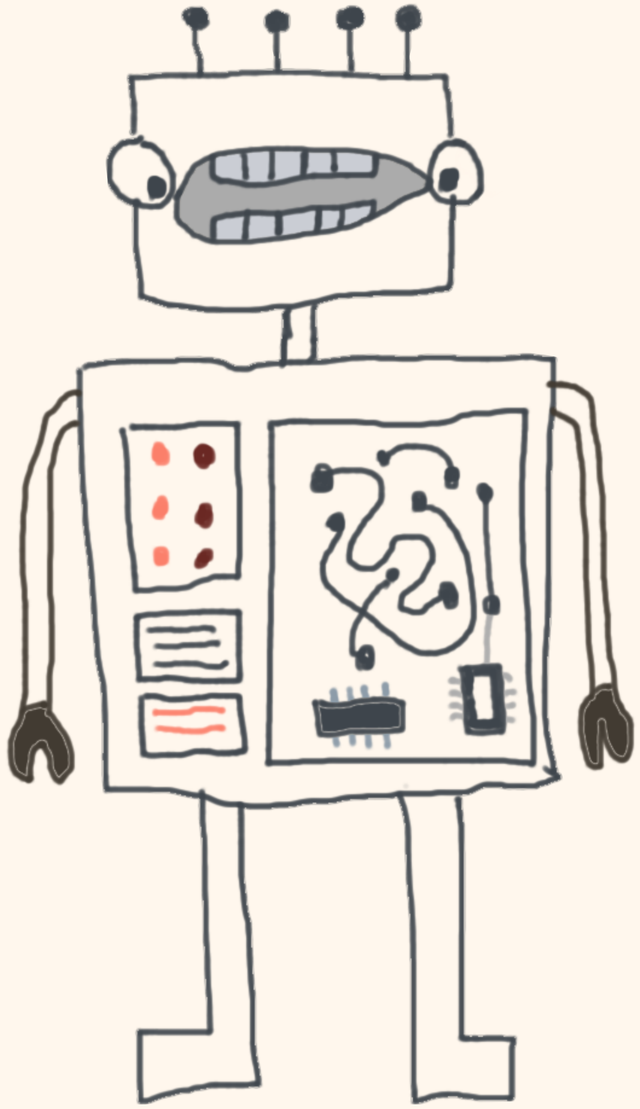
Hazelcast

A distributed cache leads to smaller heaps, more capacity and is easy to scale

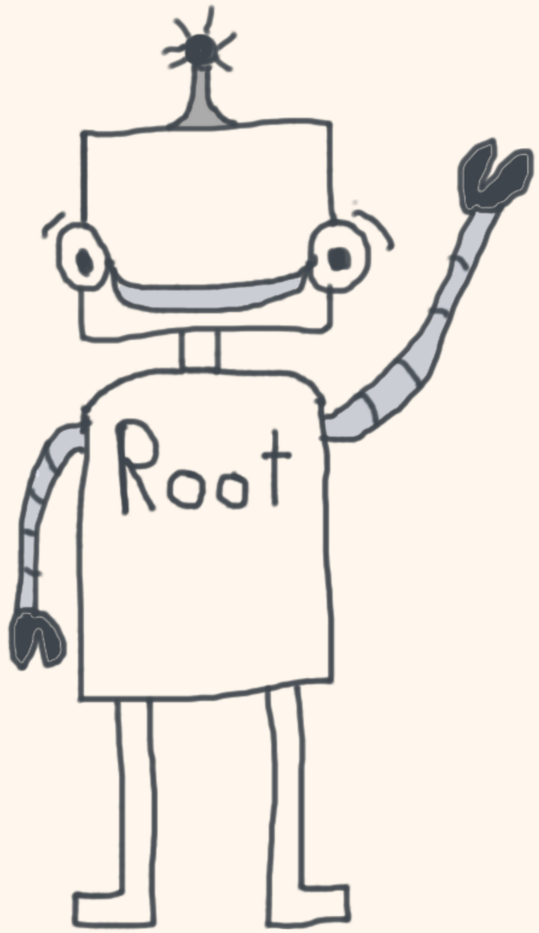


6

The operations specialist is your new best friend



Clustered caches are complex. Please make sure that operations and networking are involved as early as possible.



How about data-consistency

Which impact does it have on my infrastructure

Which data shall I cache?

Which cache shall I use?

How do I introduce caching?

Where shall I cache?

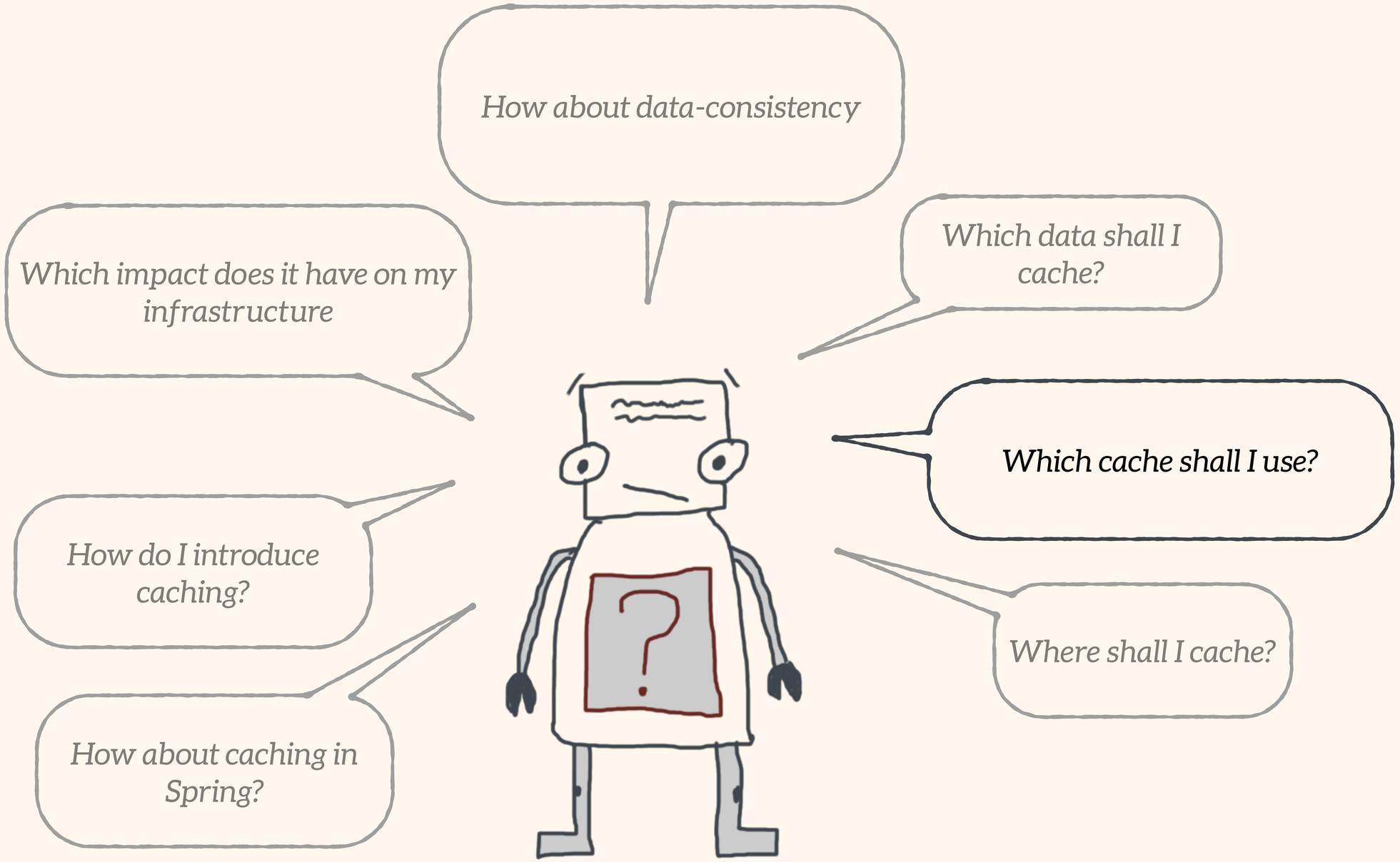
How about caching in Spring?

7

Make sure that only suitable data gets cached

The best cache candidates are read-mostly data, which are expensive to obtain

If you urgently must cache write-intensive data make sure to use a distributed cache and not a replicated or invalidating one


How about data-consistency
Which impact does it have on my infrastructure
Which data shall I cache?
Which cache shall I use?
How do I introduce caching?
Where shall I cache?
How about caching in Spring?

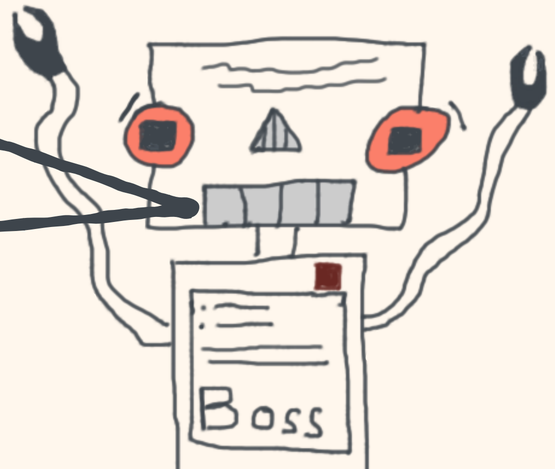
8

Only use existing cache implementations

NEVER

write your own cache implementation

EVER



Infinispan, EHCache, Hazelcast, Couchbase, Memcache, OSCache, SwarmCache, Xtreme Cache, Apache DirectMemory

CACHE

Implementations

Terracotta, Coherence, Gemfire, Cacheonix, WebSphere eXtreme Scale, Oracle 12c In Memory Database

How about data-consistency

Which impact does it have on my infrastructure

Which data shall I cache?

How do I introduce caching?

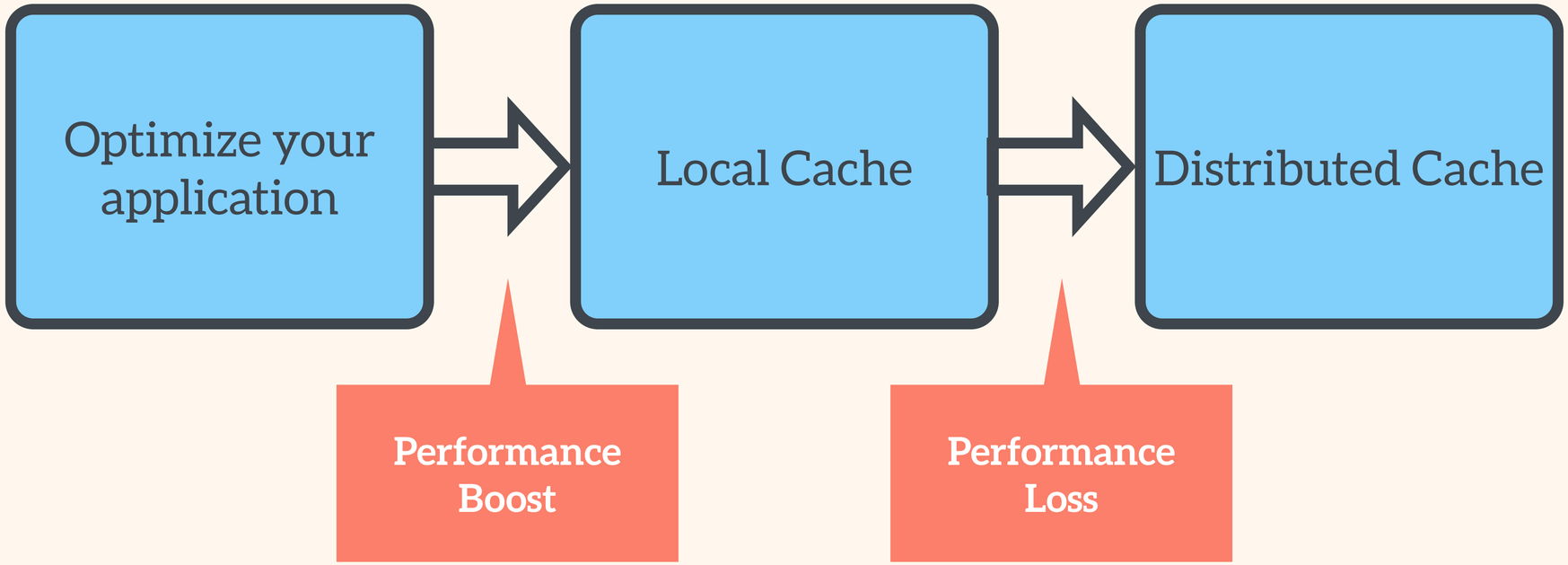
Which cache shall I use?

How about caching in Spring?

Where shall I cache?

9

Introduce Caching in three steps


Optimize your application
Local Cache
Distributed Cache
Performance Boost
Performance Loss

Optimize Serialization

Example: Hazelcast

putting and getting 10.000 objects locally

	GET Time	PUT Time	Payload Size
Serializable	?	?	?
Data Serializable	?	?	?
Identifier Data Serializable	?	?	?

DEMO
Serialization

Example: Hazelcast

putting and getting 10.000 objects locally

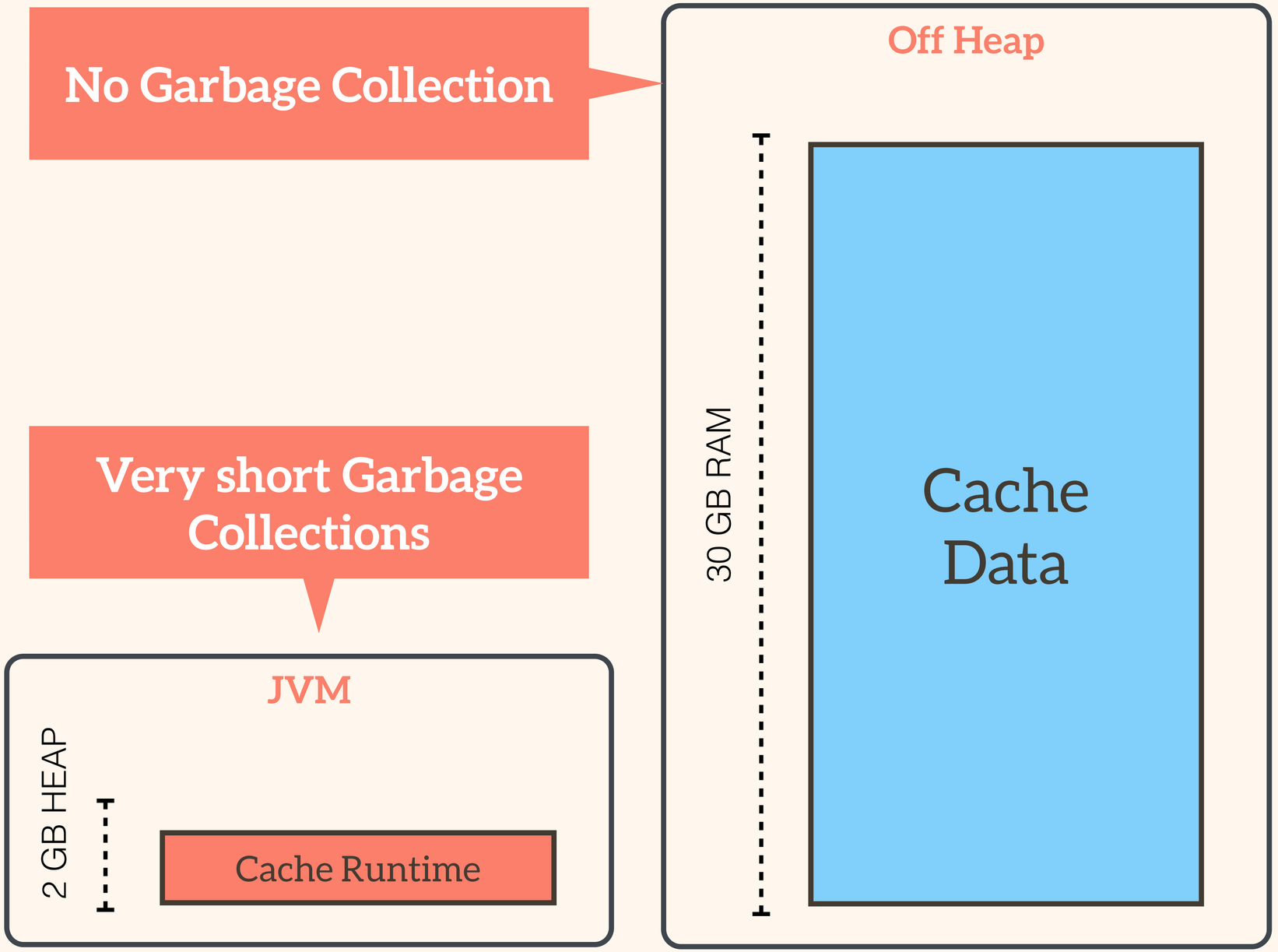
	GET Time	PUT Time	Payload Size
Serializable	1287 ms	1220 ms	1164 byte
Data Serializable	443 ms	408 ms	916 byte
Identifier Data Serializable	264 ms	207 ms	882 byte

JAVA *SERIALIZATION* SUCKS

for Caching if alternatives are present

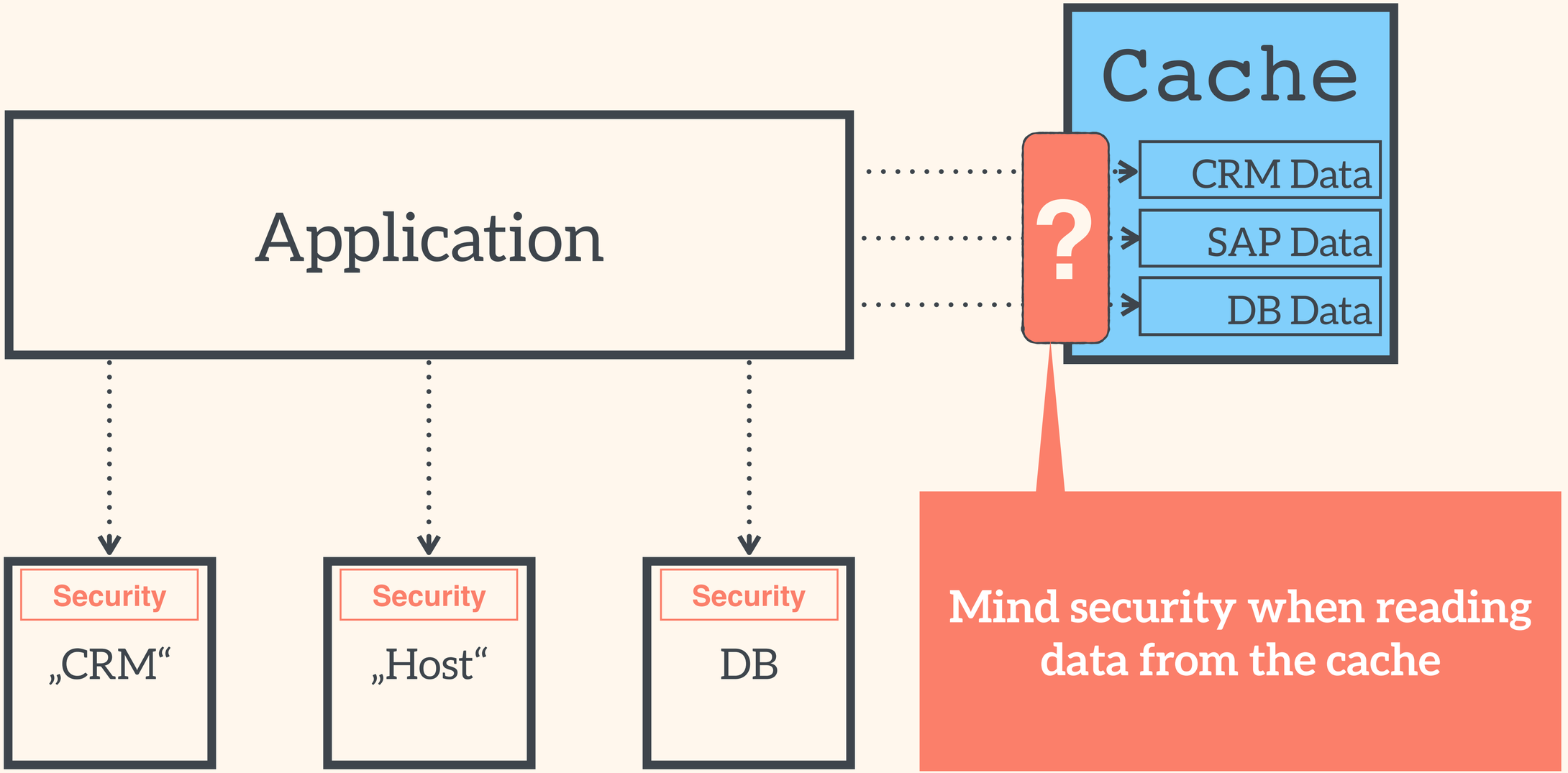
11

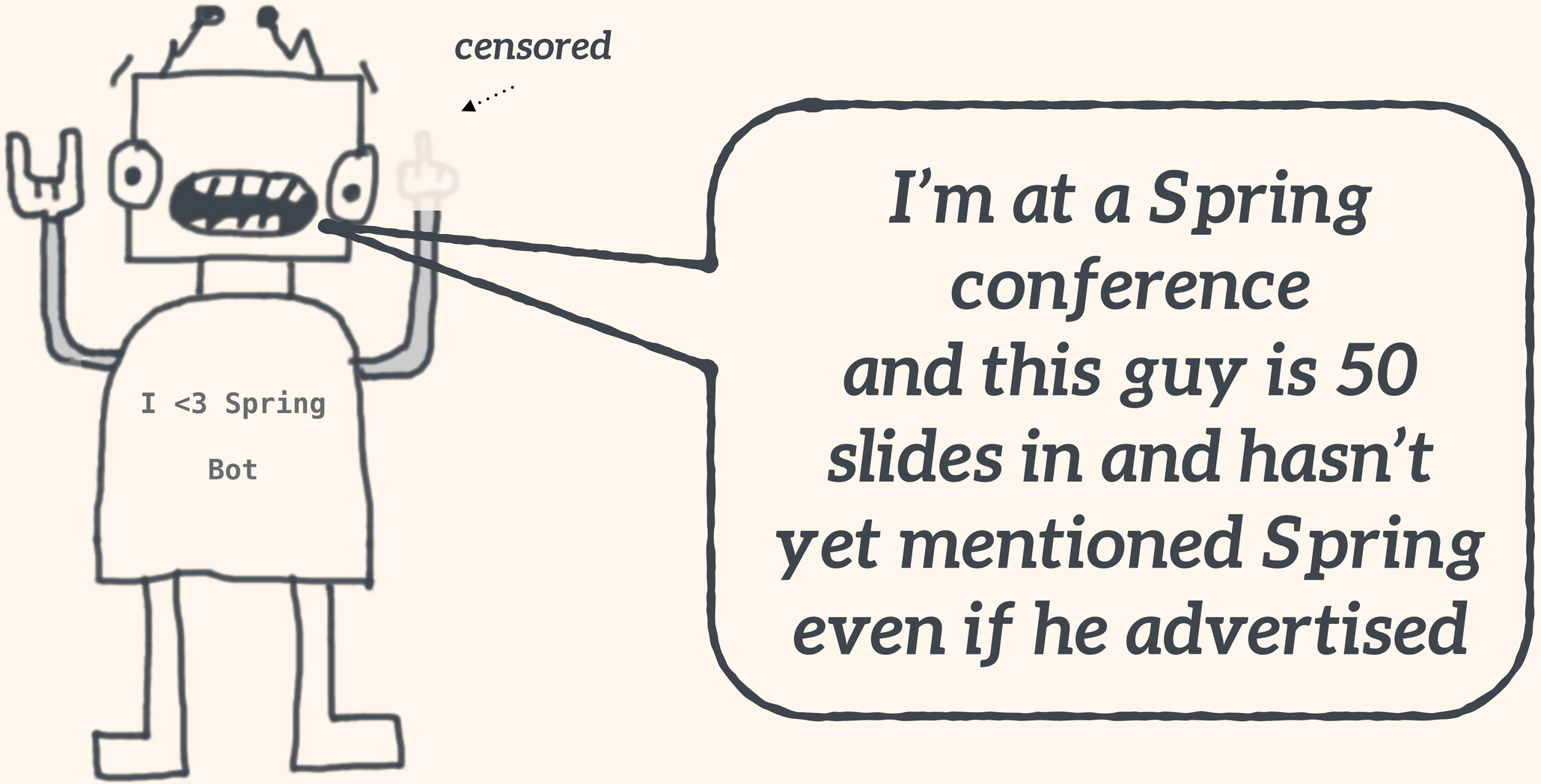
Use Off-Heap Storage for Cache instances with more than 4 GB Heap Size


No Garbage Collection
Off Heap
30 GB RAM
Cache Data
Very short Garbage Collections
JVM
2 GB HEAP
Cache Runtime

12

Mind the security gap


Application
Cache
CRM Data
SAP Data
DB Data
?
Security
„CRM“
Security
„Host“
Security
DB
Mind security when reading data from the cache


censored
I <3 Spring
Bot
I'm at a Spring conference and this guy is 50 slides in and hasn't yet mentioned Spring even if he advertised

13

Abstract your cache provider

Tying your code to a cache provider is bad practice

```
public Account retrieveAccount(String accountNumber)
{
  Cache cache = ehCacheMgr.getCache(„accounts“);
  Account account = null;
  Element element = cache.get(accountNumber);
  if(element == null) {
    //execute some business logic for retrieval
    //account = result of logic above
    cache.put(new Element(accountNumber, account));
  } else {
    account = (Account)element.getObjectValue();
  }
  return account;
}
```

Try switching from EHCache to Hazelcast

```
public Account retrieveAccount(String accountNumber)
{
  Cache cache = ehCacheMgr.getCache(„accounts“);
  Account account = null;
  Element element = cache.get(accountNumber);
  if(element == null) {
    //execute some business logic for retrieval
    //account = result of logic above
    cache.put(new Element(accountNumber, account));
  } else {
    account = (Account)element.getObjectValue();
  }
  return account;
}
```

You will have to adjust these lines of code to the Hazelcast API

You can't switch cache providers between environments

```
public Account retrieveAccount(String accountNumber)
{
  Cache cache = ehCacheMgr.getCache(„accounts“);
  Account account = null;
  Element element = cache.get(accountNumber);
  if(element == null) {
    //execute some business logic for retrieval
    //account = result of logic above
    cache.put(new Element(accountNumber, account));
  } else {
    account = (Account)element.getObjectValue();
  }
  return account;
}
```

EHCache is tightly coupled to your code

You mess up your business logic with infrastructure

```
public Account retrieveAccount(String accountNumber)
{
  Cache cache = ehCacheMgr.getCache(„accounts“);
  Account account = null;
  Element element = cache.get(accountNumber);
  if(element == null) {
    //execute some business logic for retrieval
    //account = result of logic above
    cache.put(new Element(accountNumber, account));
  } else {
    account = (Account)element.getObjectValue();
  }
  return account;
}
```

This is all caching related code without any business relevance

Introducing Spring's cache abstraction

```
<cache:annotation-driven cache-manager="ehCacheManager"/>

<!-- EH Cache local -->
<bean id="ehCacheManager"
  class="org.springframework.cache.ehcache.EhCacheCacheManager"
        p:cacheManager-ref="ehcache"/>

<bean id="ehcache"
  class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"
  p:configLocation="/ehcache.xml"/>
```

```
@Cacheable("Customers")
public Customer getCustomer(String customerNumber) {
  …
}
```

Spring's Caching Annotations

Annotation	Description
@Cacheable	Demarcates cachable methods, can read and write to the cache(s)
@CacheEvict	Demarcates methods that perform cache eviction, that is methods that act as triggers for removing data from the cache.
@CachePut	Updates the cache with the annotated method's return value. Will always execute the method.
@Caching	Allows multiple nested @Cacheable, @CacheEvict and @CachePut annotations to be used on the same method
@CacheConfig	Class-level annotation that allows to share the cache names, the custom KeyGenerator, the custom CacheManager and finally the custom CacheResolver. Does not enable caching.

Default Key Generation Strategy

Annotation	*Key*

`@Cacheable("Customers")`
`public Customer getCustomer(String customerNumber) {`
`  …`
`}`
	customerNumber

`@Cacheable("CustomerList")`
`public List<Customer> listCustomers(int start, int count) {`
`  …`
`}`
	SimpleKey containing start and count

`@Cacheable("MonthlyReport")`
`public Report getMonthlyReport() {`
`  …`
`}`
	SimpleKey.EMPTY

You need a custom default KeyGenerator?

```
public class MyOwnKeyGenerator implements KeyGenerator {
    @Override
    public Object generate(Object target, Method method, Object... params) {
        if (params.length == 0) {
            return new SimpleKey("EMPTY");
        }
        if (params.length == 1) {
            Object param = params[0];
            if (param != null && !param.getClass().isArray()) {
                return param;
            }
        }
        return new SimpleKey(params);
    }
}
```

```
<cache:annotation-driven cache-manager="hazelcastCacheManager"
                         keyGenerator="myOwnKeyGenerator" />
```

SpEL in Caching Annotations

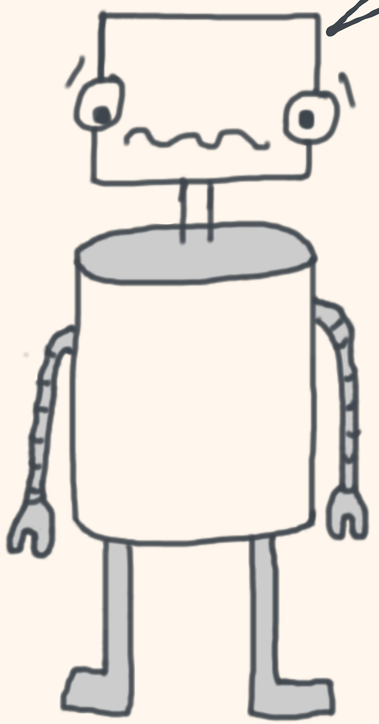
Annotation	*Effect*
`@Cacheable("concerts", key="#location.id")` `public List<Concert> findConcerts(Location location)`	Key: id of location

`@Cacheable("concerts",`
`key="T(someType).hash(#location)")`
`public List<Concert> findConcerts(Location location)`
	Key: hashCode of location

`@Cacheable("concerts",`
`condition="#location.city == 'Dallas'",`
`unless="#location.outOfBusiness")`
`public List<Concert> findConcerts(Location location)`
	Conditional Caching if Location is in Dallas in operating

`@CachePut("locations", key="#result.id")` `public Location saveLocation(Location location)`	Key: generated id of result

I have multiple Caches and Cache Managers!



```
@Cacheable("concerts", cacheManager="hazelCastCacheManager")
public List<Concert> findConcerts(Location location)
```

Manual Assignment

```
@Cacheable("bands", cacheManager="gemfireCacheManager"))
public List<Band> listBand(int start, int count)
```

Manual Assignment

```
@Cacheable("bands", cacheResolver="myOwnCacheResolver"))
public List<Band> listBand(int start, int count)
```

Programmatic resolution through an implementation of the CacheResolver Interface

Working with CacheResolvers

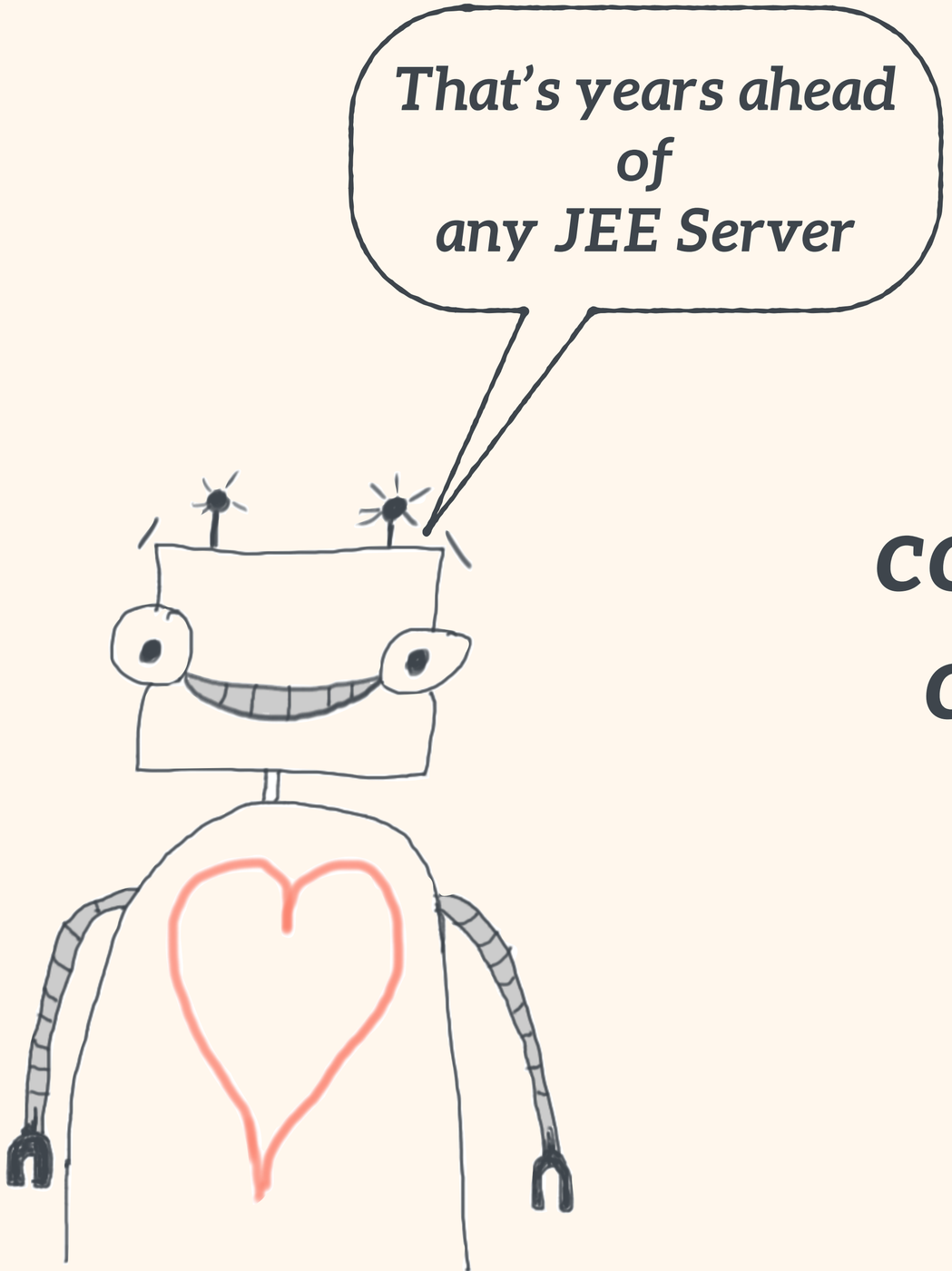
```
@Cacheable("bands", cacheResolver="myOwnCacheResolver"))
public List<Band> listBand(int start, int count)
```

```
public class MyOwnCacheResolver extends AbstractCacheResolver {
  @Autowired
  public MyOwnCacheResolver(CacheManager cacheManager) {
    super(cacheManager);
  }

  protected Collection<String> getCacheNames(CacheOperationInvocationContext<?> context) {
    return getCacheNames(context.getTarget().getClass());
  }

  private getCacheNames(Class<?> businessServiceClass) {
    ...
  }
}
```

You can use your own custom Annotations

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
@Cacheable("concerts", key="id")
public @interface DefaultConcertCacheable {
}
```

```
@DefaultConcertCacheable
public Concert getConcert(Long id)
```



Spring 4.x is the first commerically supported container with JCache (JSR-107) Support!

Spring vs JCache Annotations

Spring	JCache	Description
`@Cacheable`	`@CacheResult`	Similar, but @CacheResult can cache Exceptions and force method execution
`@CacheEvict`	`@CacheRemove`	Similar, but @CacheRemove supports eviction in the case of Exceptions
`@CacheEvict (removeAll=true)`	`@CacheRemoveAll`	Same rules as for @CacheEvict vs @CacheRemove
`@CachePut`	`@CachePut`	Different semantic: cache content must be annotated with @CacheValue. JCache brings Exception caching and caching before or after method execution
`@CacheConfig`	`@CachePut`	Identical

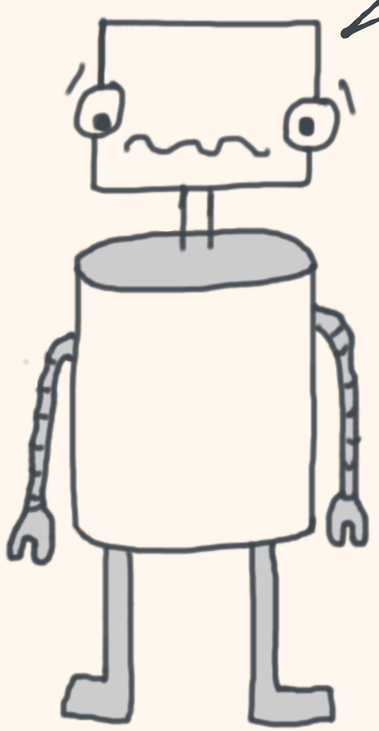
Except for the dependencies JCache API and spring-context-support no further steps need to be taken to enable JCache Annotations in Spring Applications

Your Cache Provider has no integration for Spring or JCache

Code Walkthrough

CacheManager & Cache API

How do I disable caching for Unit Tests?



```
<bean id="cacheManager"
class="org.springframework.cache.support.CompositeCacheManager">
    <property name="cacheManagers">
        <list>
            <ref bean="guavaCache"/>
            <ref bean="ehCache"/>
        </list>
    </property>
    <property name="fallbackToNoOpCache" value="true"/>
</bean>
```

Thank you!

Michael Plöd - Freelance Consultant

https://github.com/mploed

@bitboss

http://slideshare.net/mploed