VOXXED DAYS BERLIN

# Caching for Business Applications: Best Practices and Gotchas

**Michael Plöd**
**innoQ Deutschland GmbH**

Platinum Sponsor

servers.com

# *I will talk about*

Caching Types / Topologies
Best Practices for Caching in Enterprise Applications

# *I will <u>NOT</u> talk about*

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

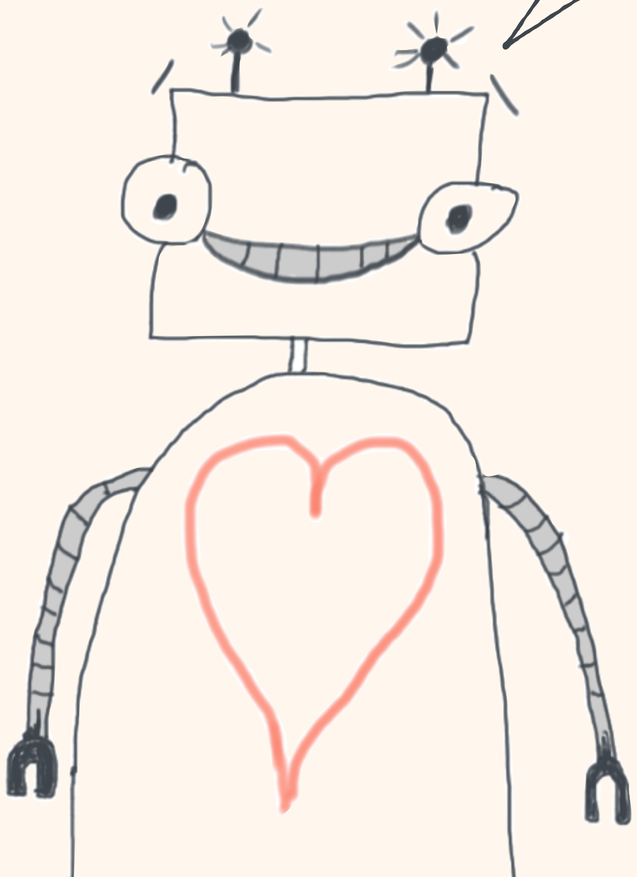
# Cache

## / kæʃ /

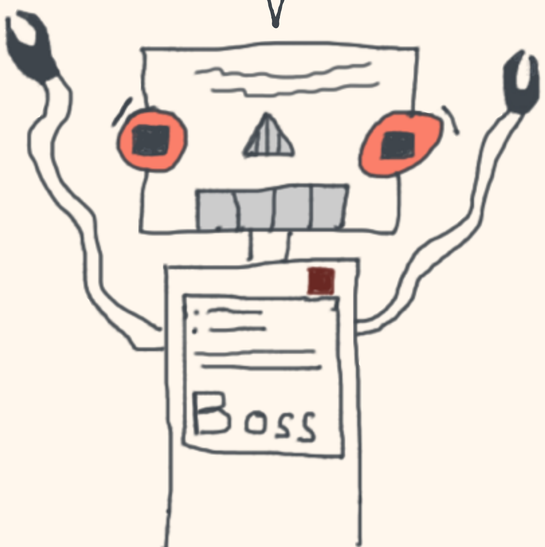
In computing, a cache is a component that **transparently stores data so that future requests for that data can be served faster**. The data that is stored within a cache might be values that have been computed earlier or duplicates of original values that are stored elsewhere. If requested data is contained 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

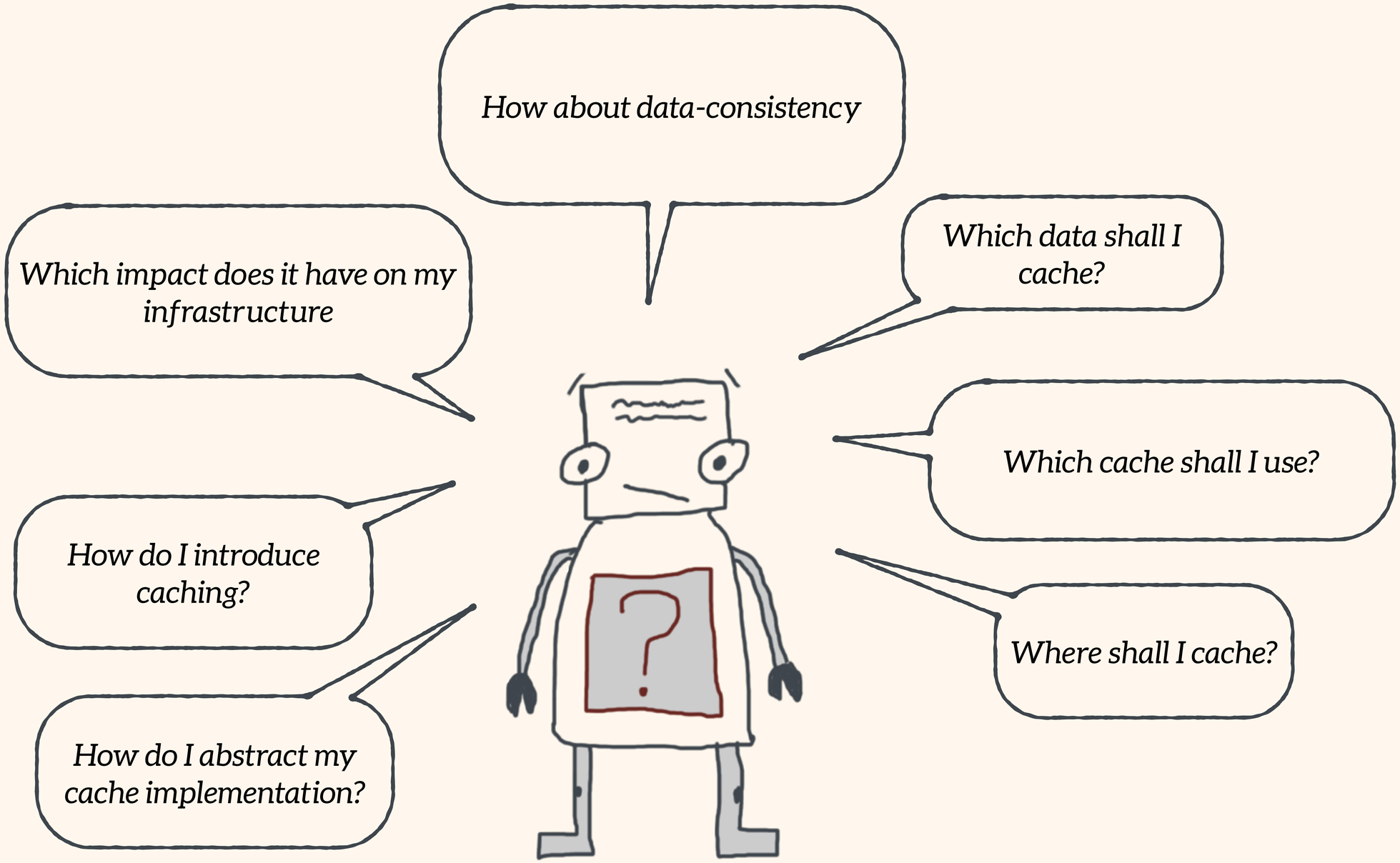
But with a well adjusted caching strategy you will make your application more scalable, faster and cheaper to operate.

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

# Types of Places for CACHES

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

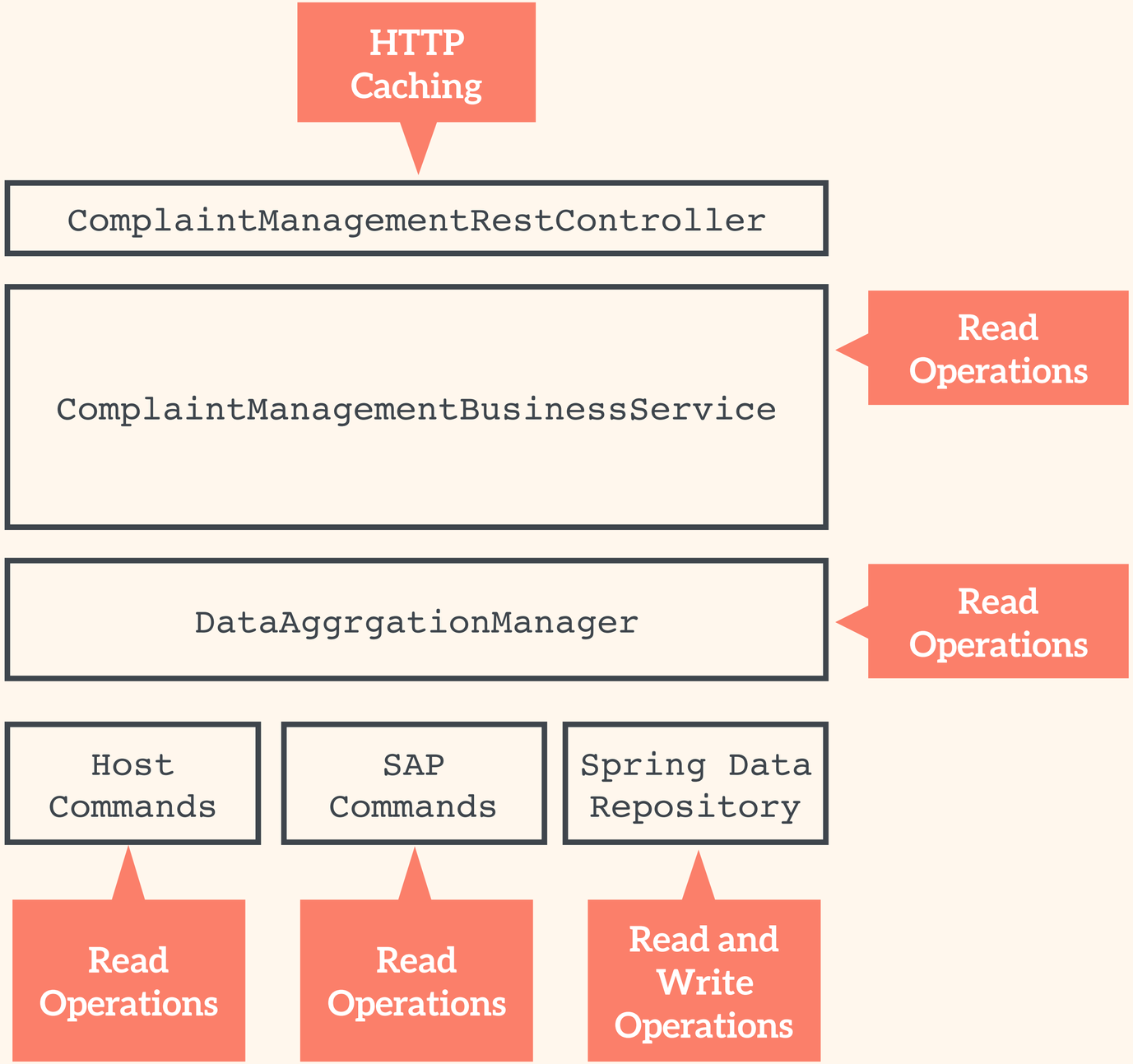
*We will focus on local and distributed caching at the application level with the Spring Framework*


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

1

*Identify suitable layers for caching*

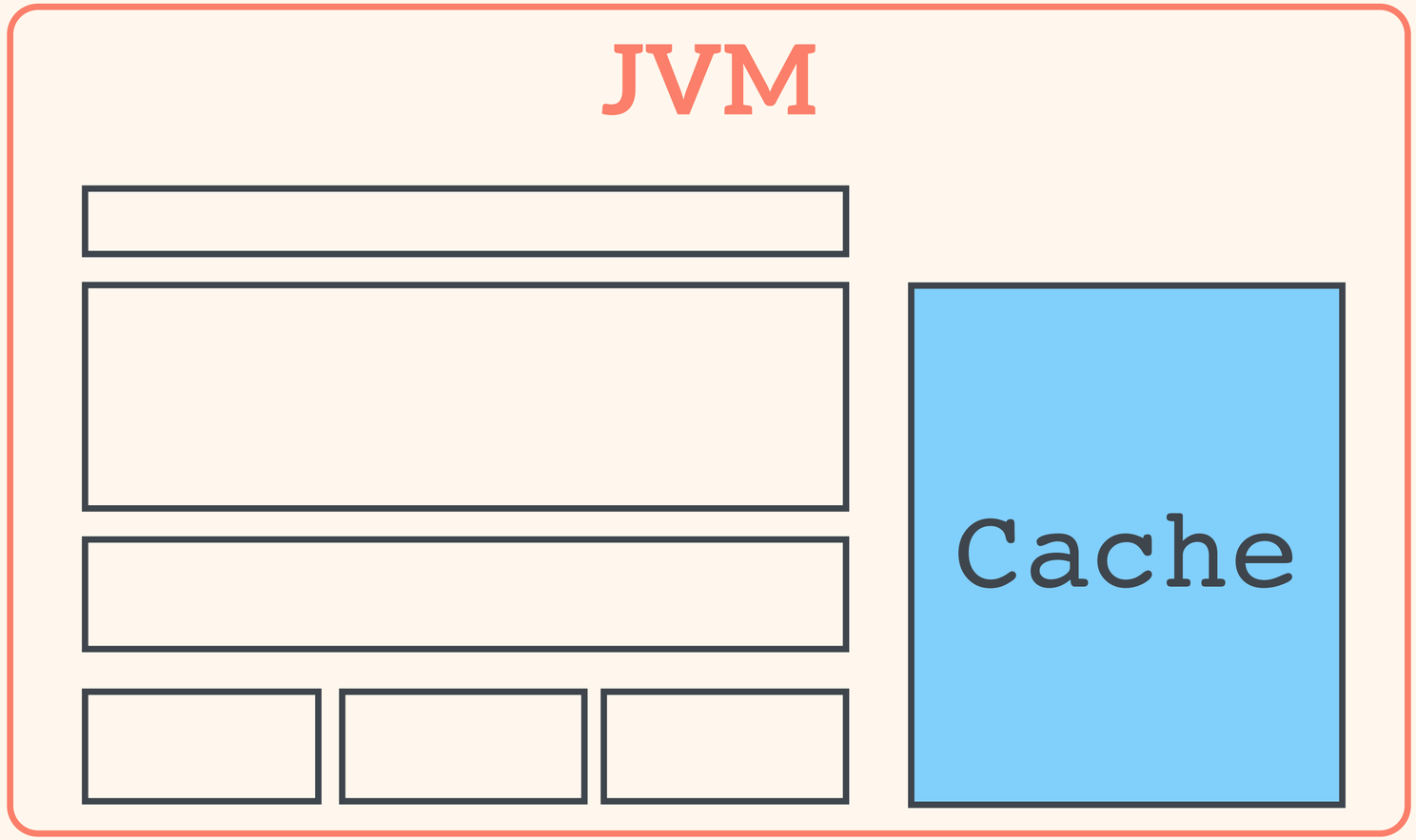
# Suitable Layers for Caching



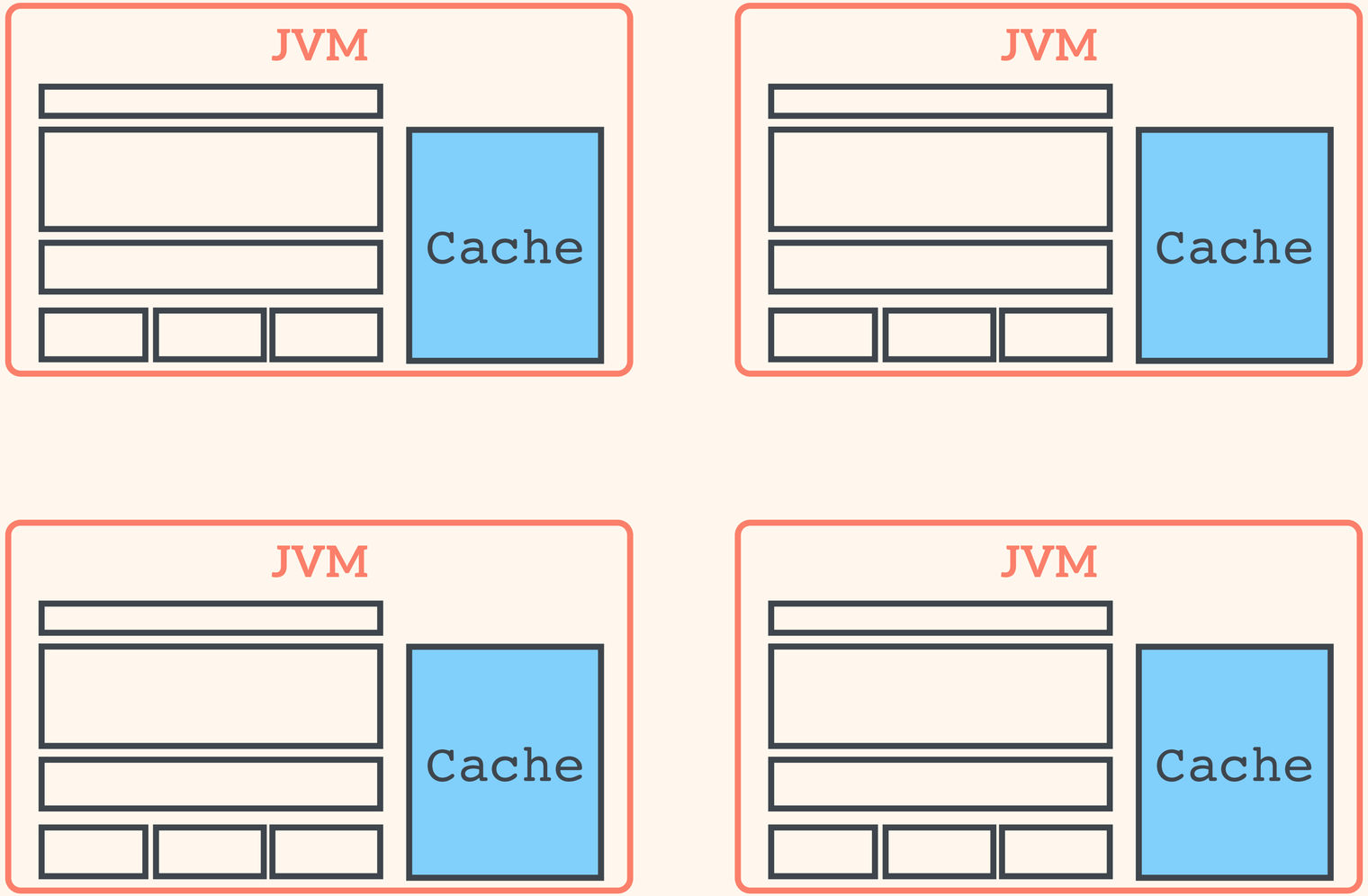
2

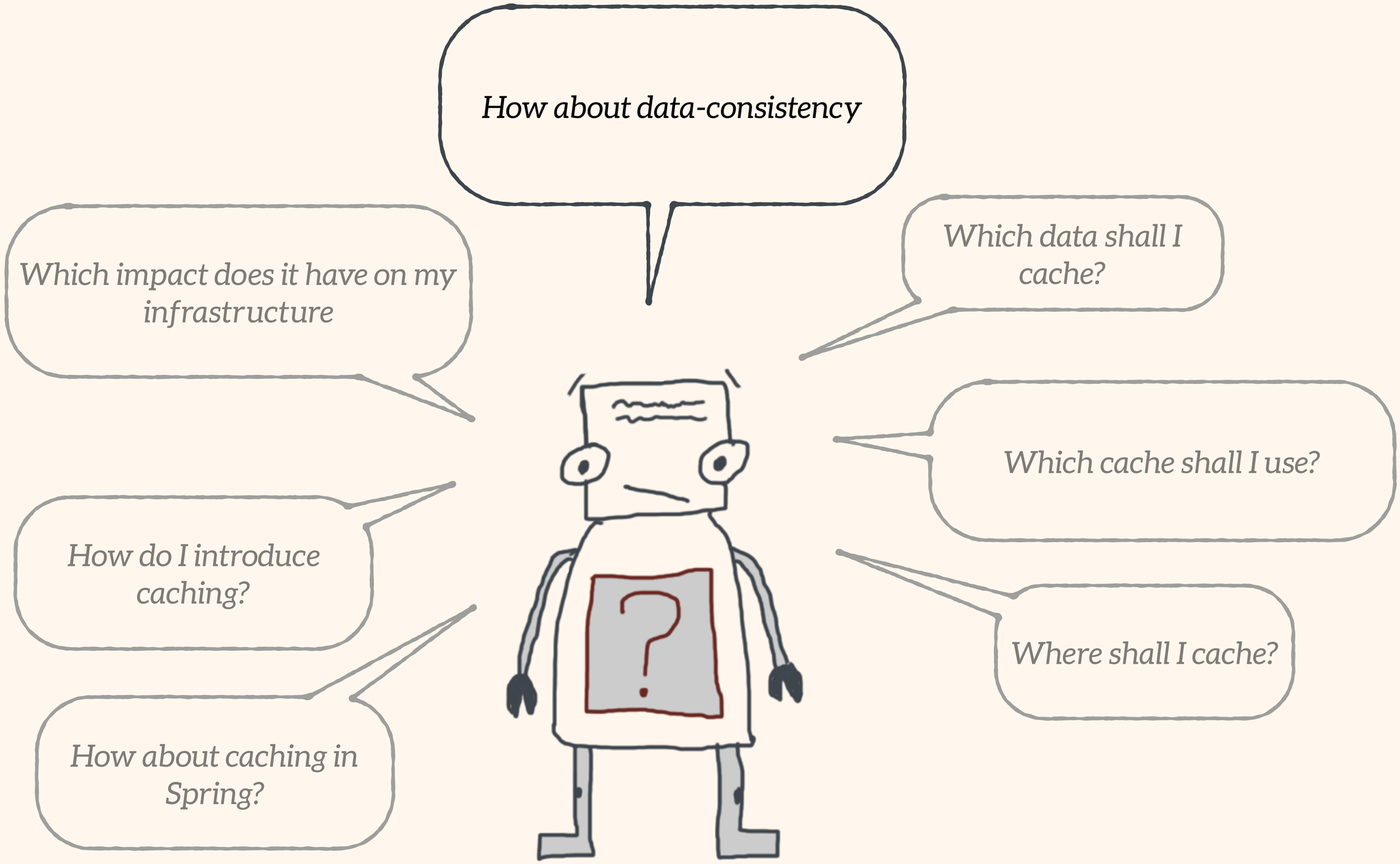
*Stay local as long as possible*

# *Lokal In-Memory*

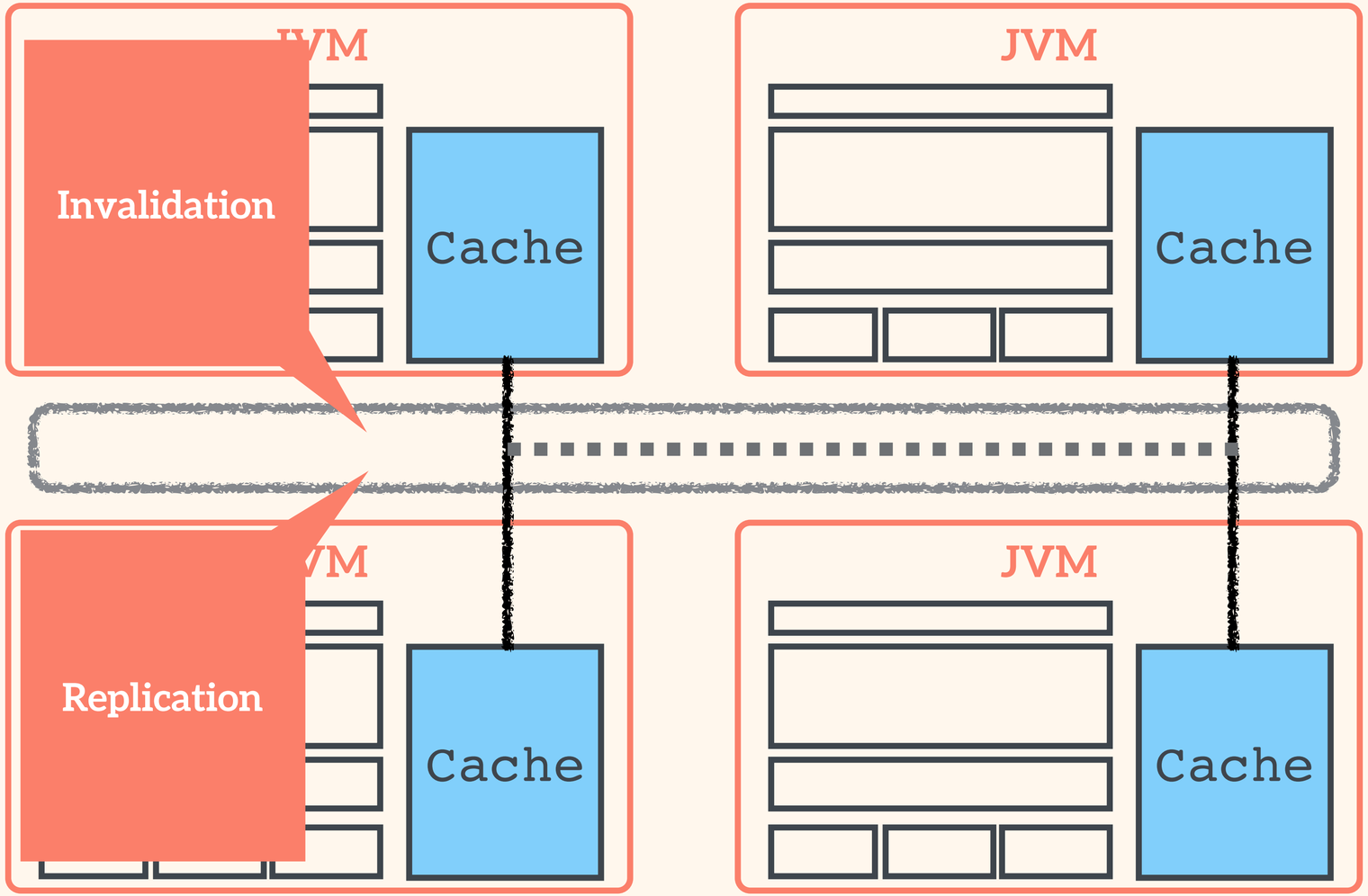


# Clustered




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

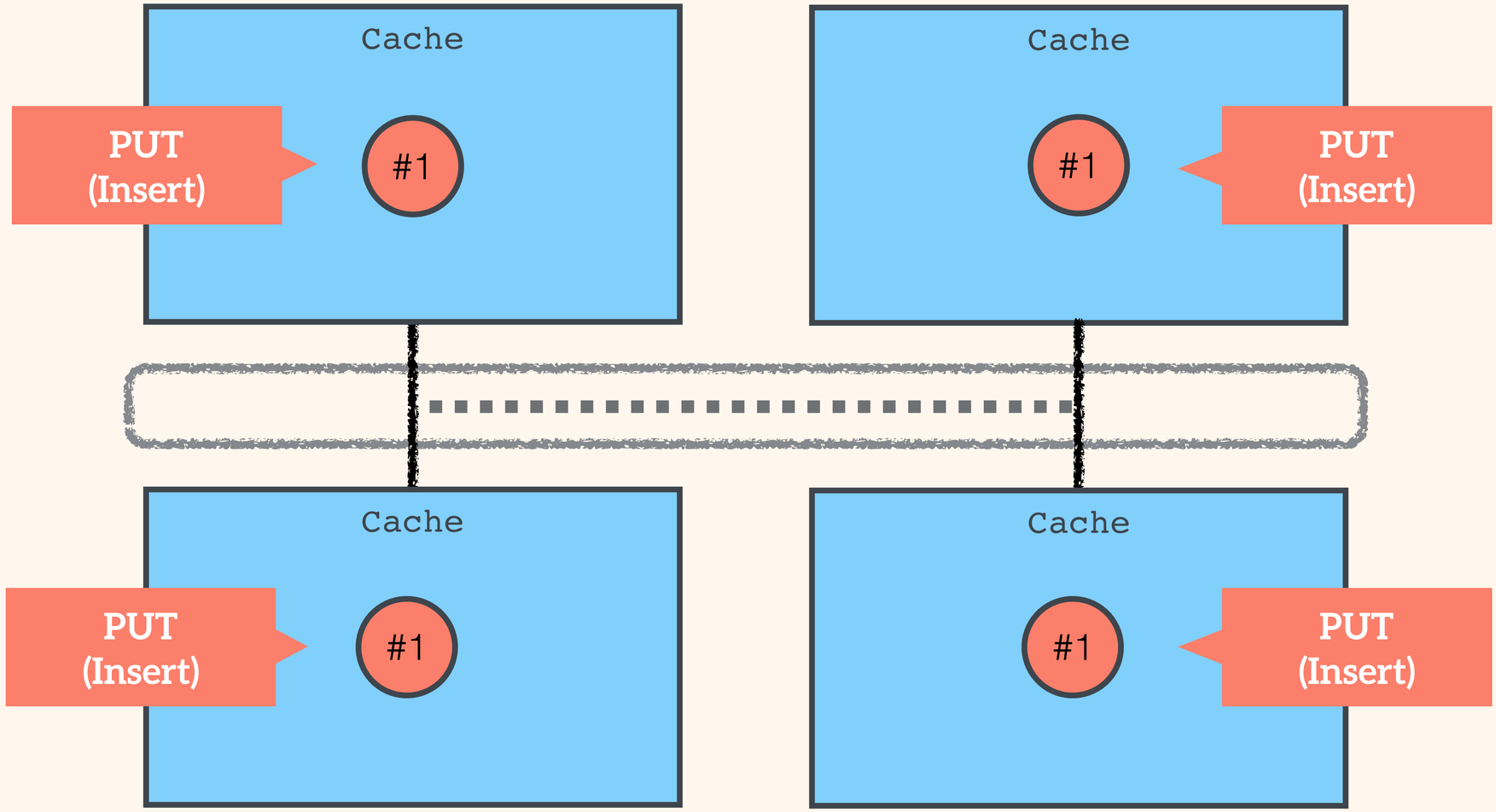
# Clustered - with sync



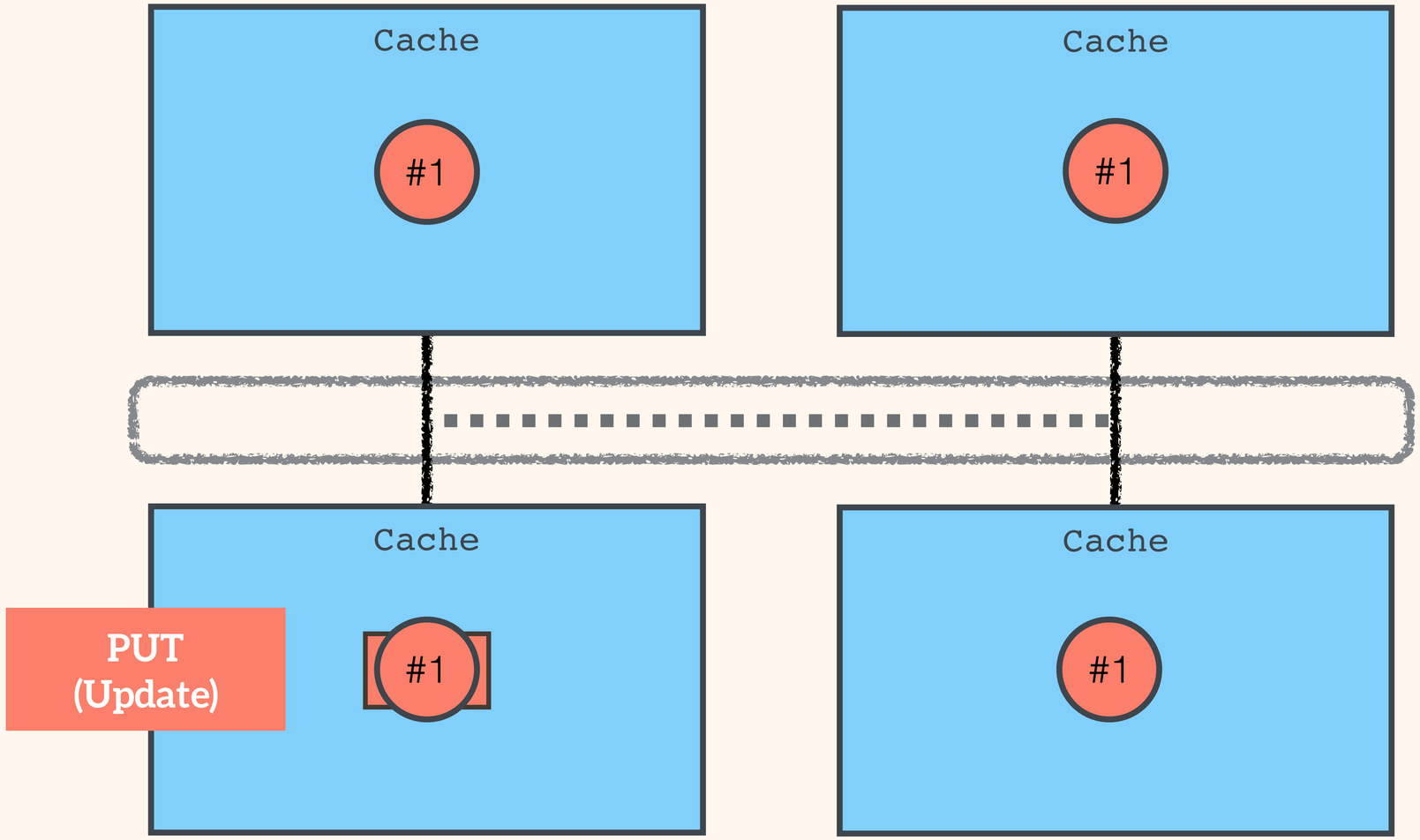
3

*Avoid real replication where possible*

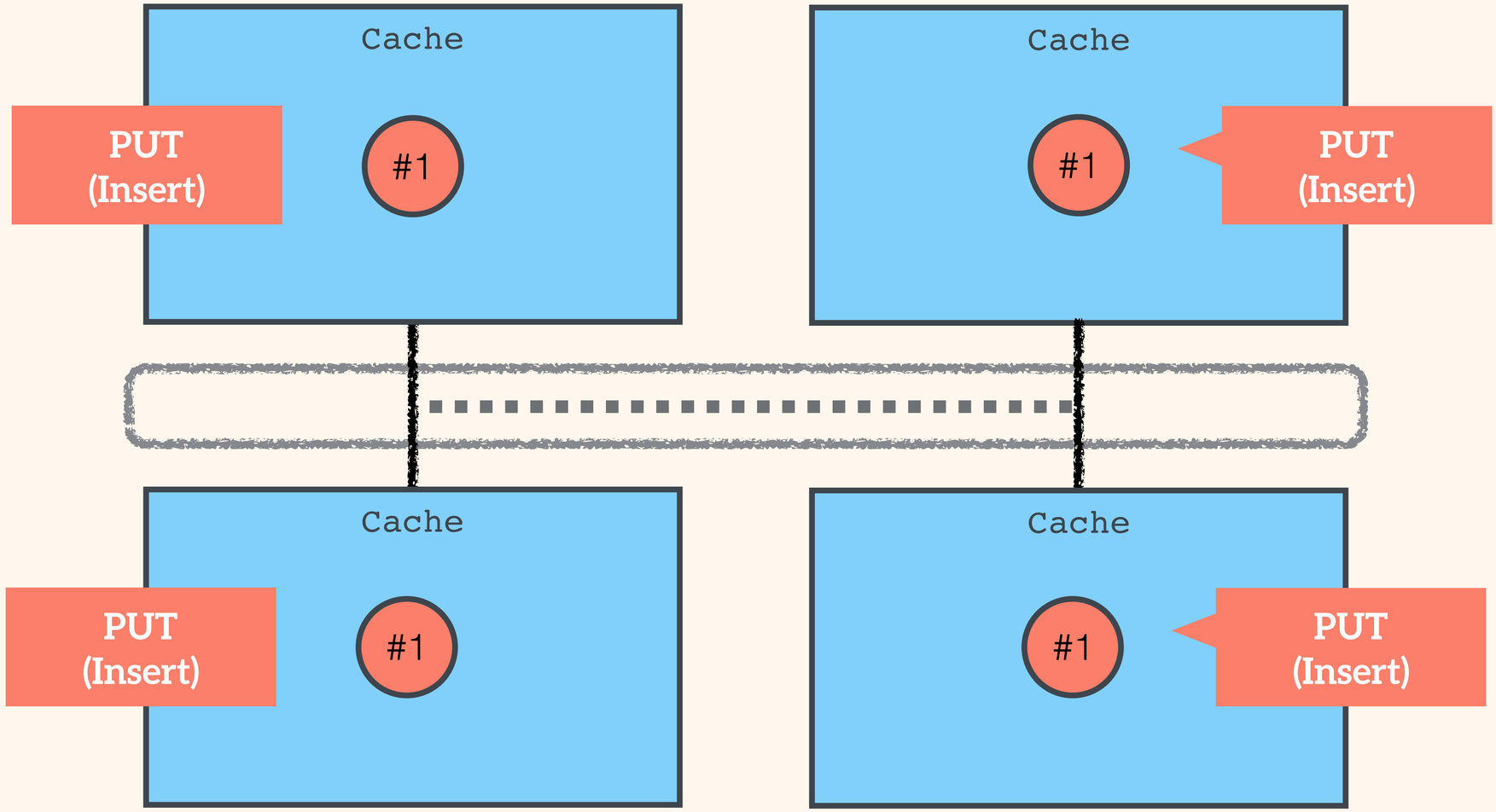
# Invalidation - Option 1



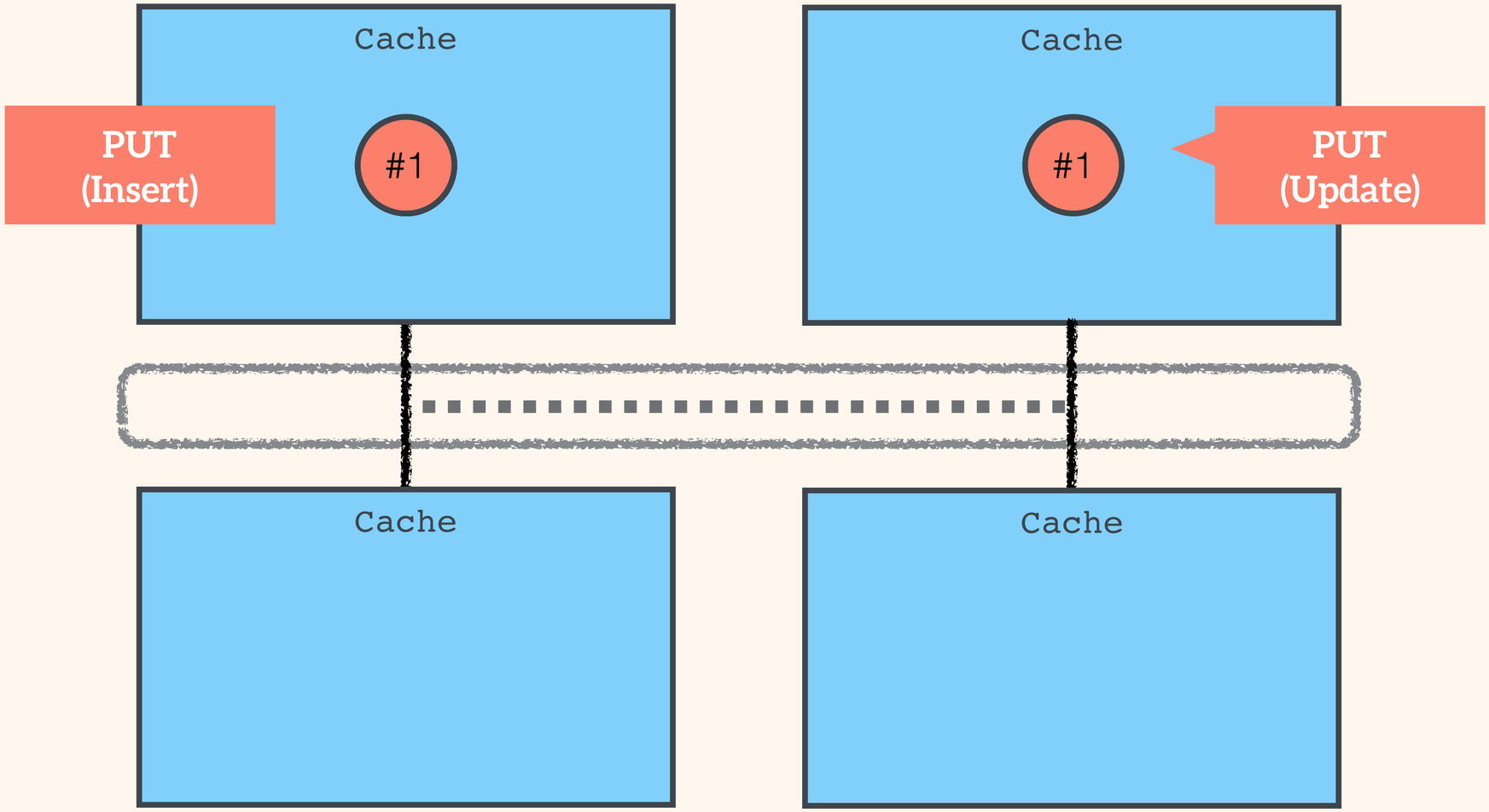
# Invalidation - Option 1



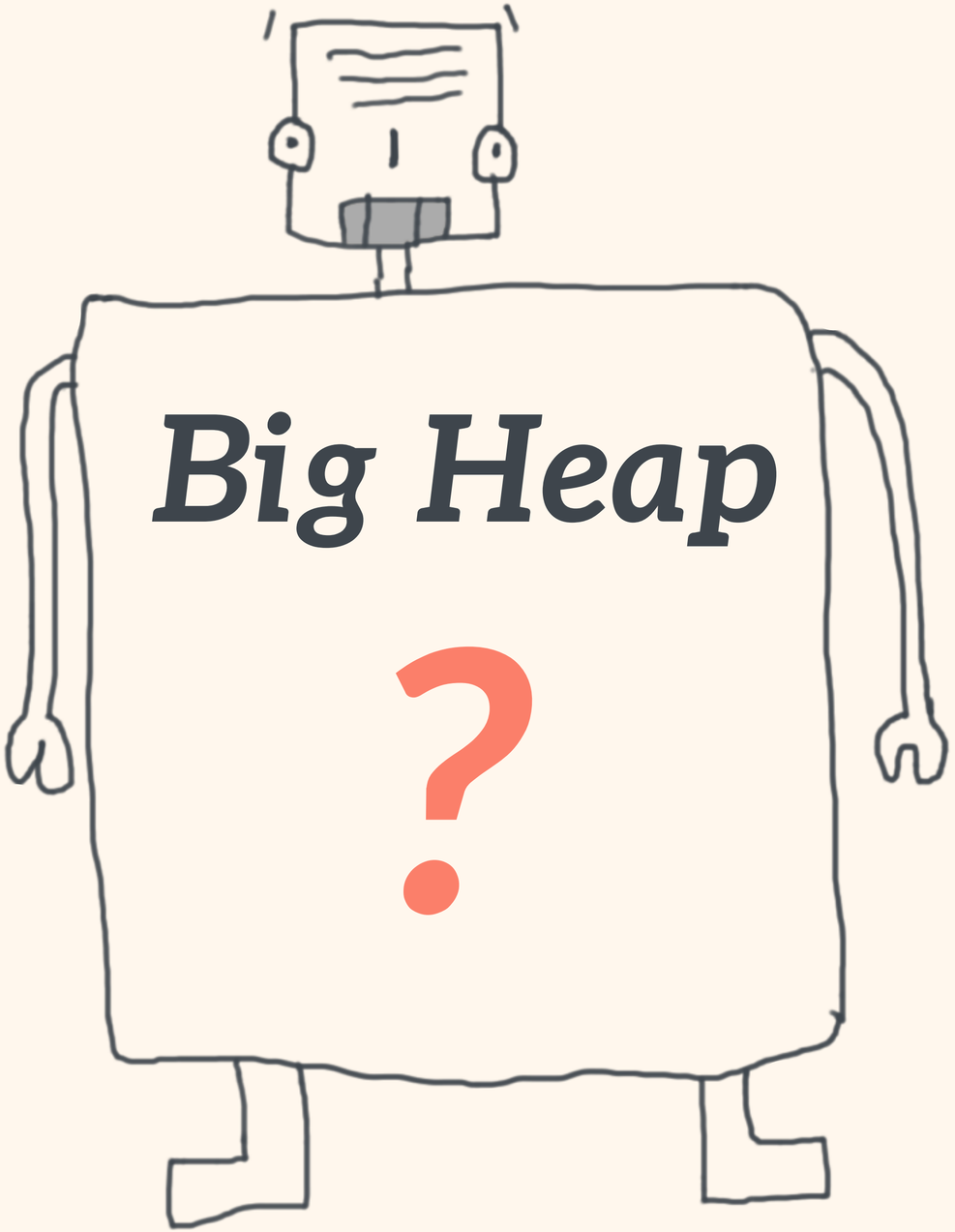
# Invalidation - Option 2

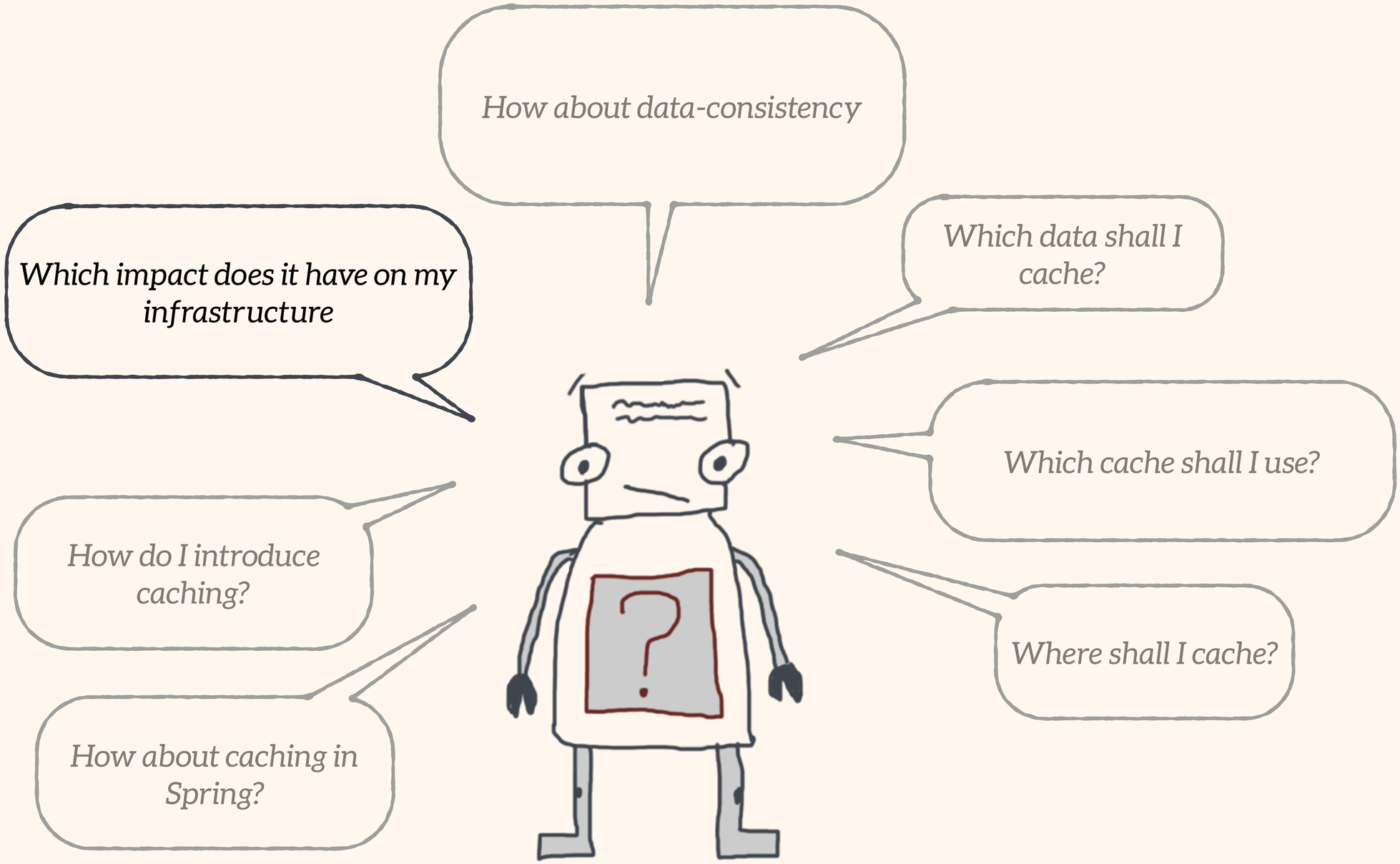


# Replication



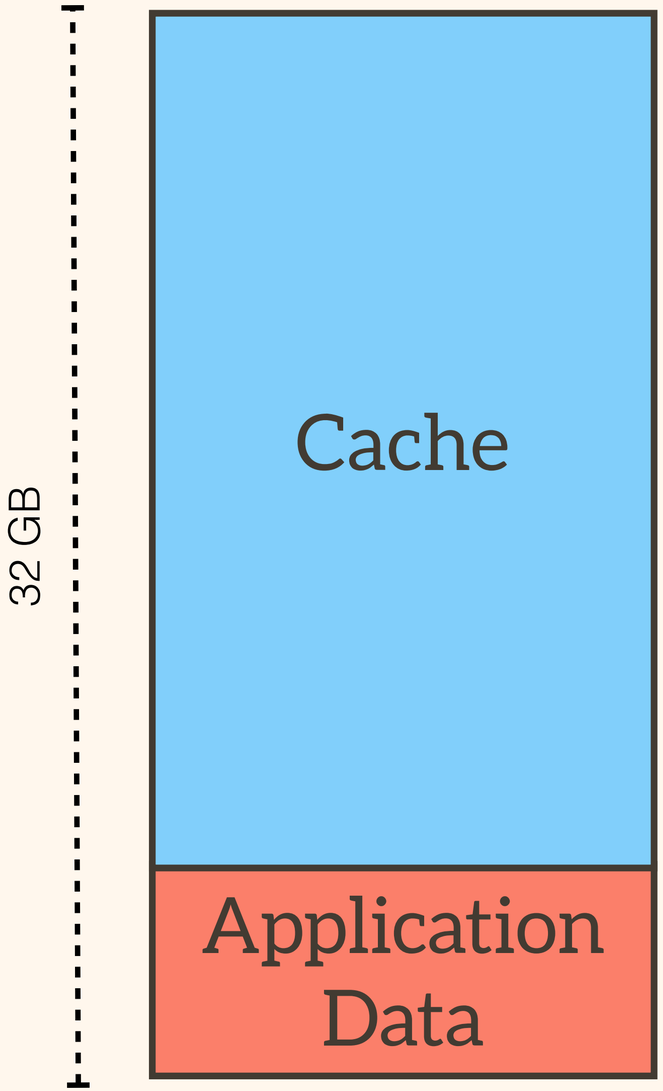
*As of now every cache could potentially hold every data which consumes heap memory*


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?

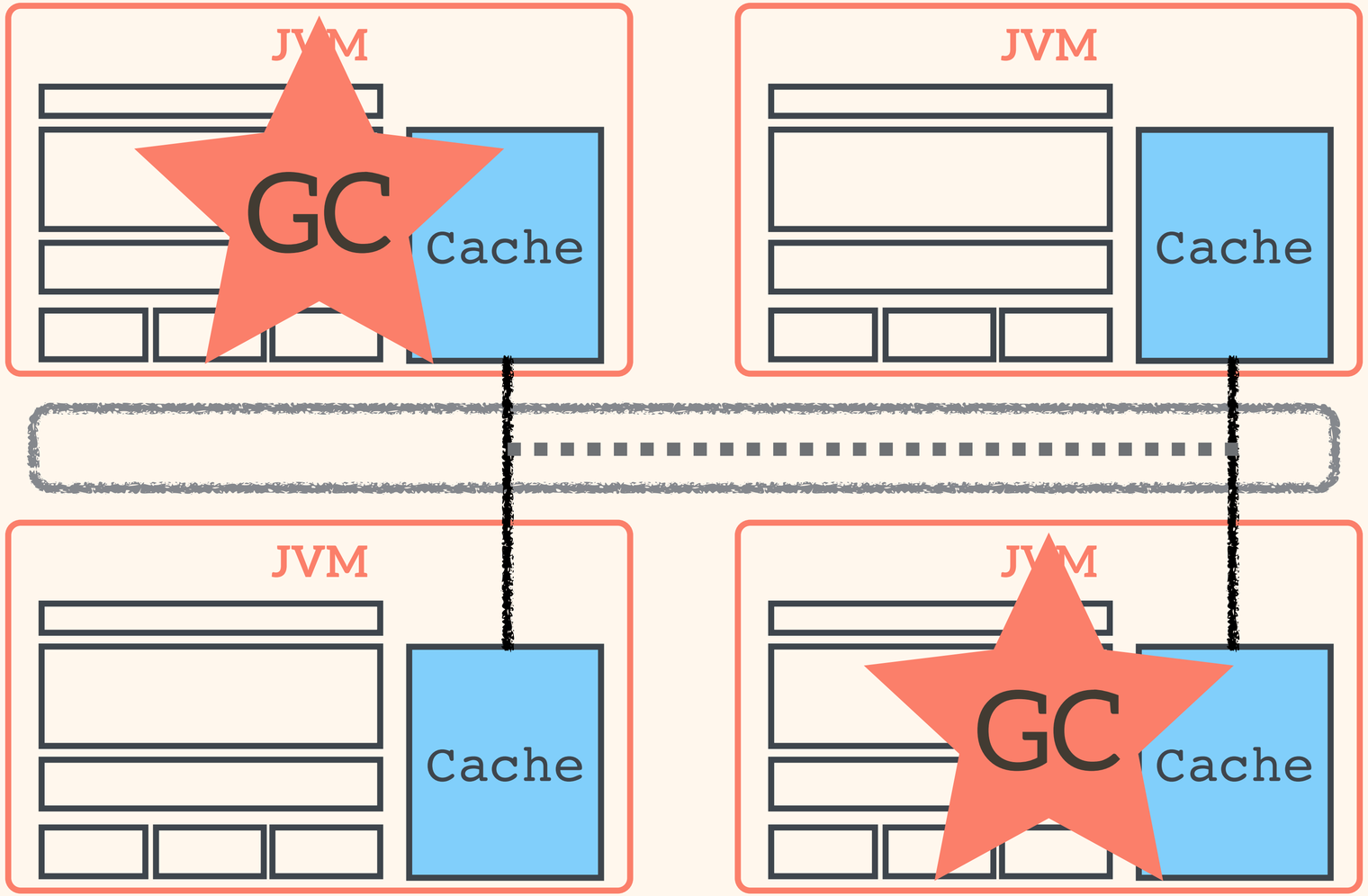
4

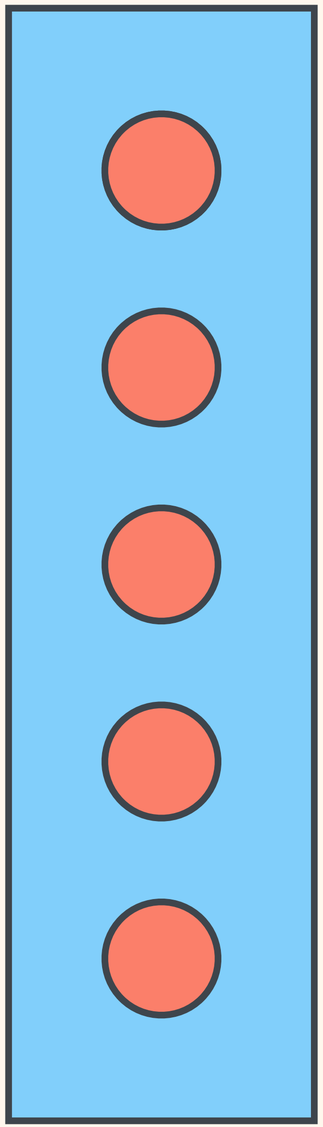
*Avoid big heaps just for caching*




*Big heap leads to long major GCs*

# *Long GCs can destabilize your cluster*





# *Small caches are a bad idea!*

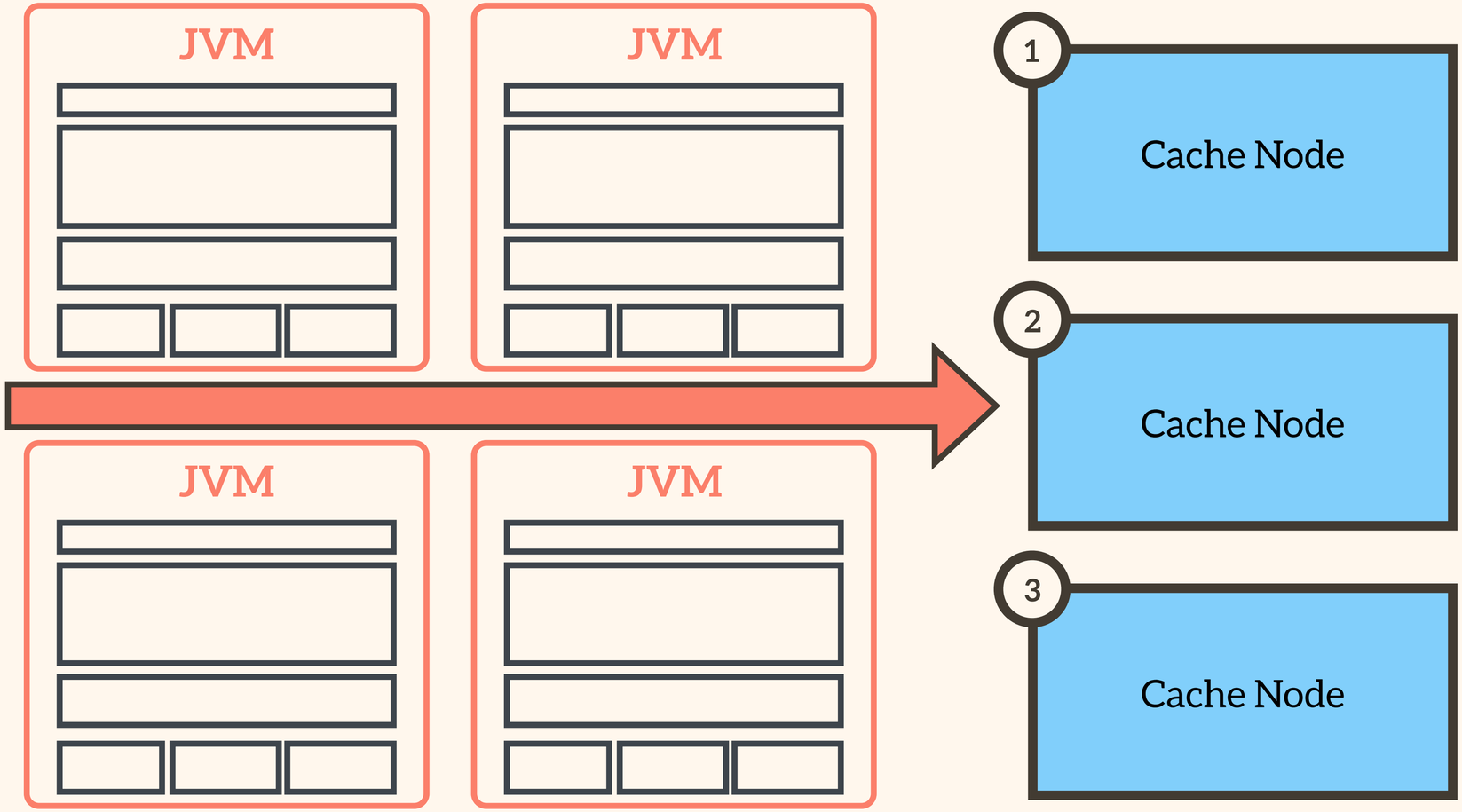
Many evictions, fewer hits, no „hot data“.

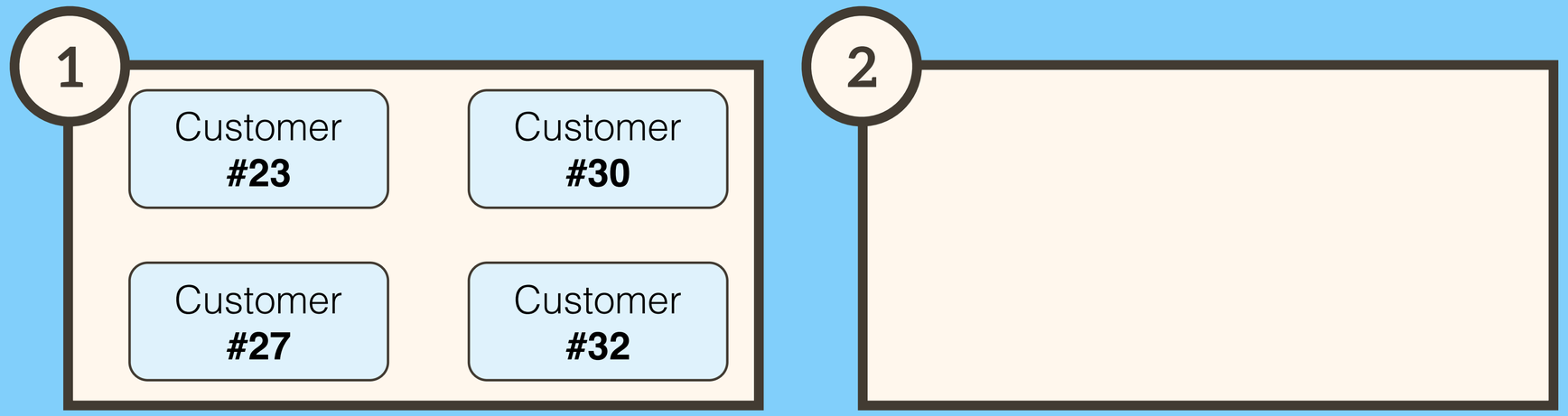
This is especially critical for replicating caches.

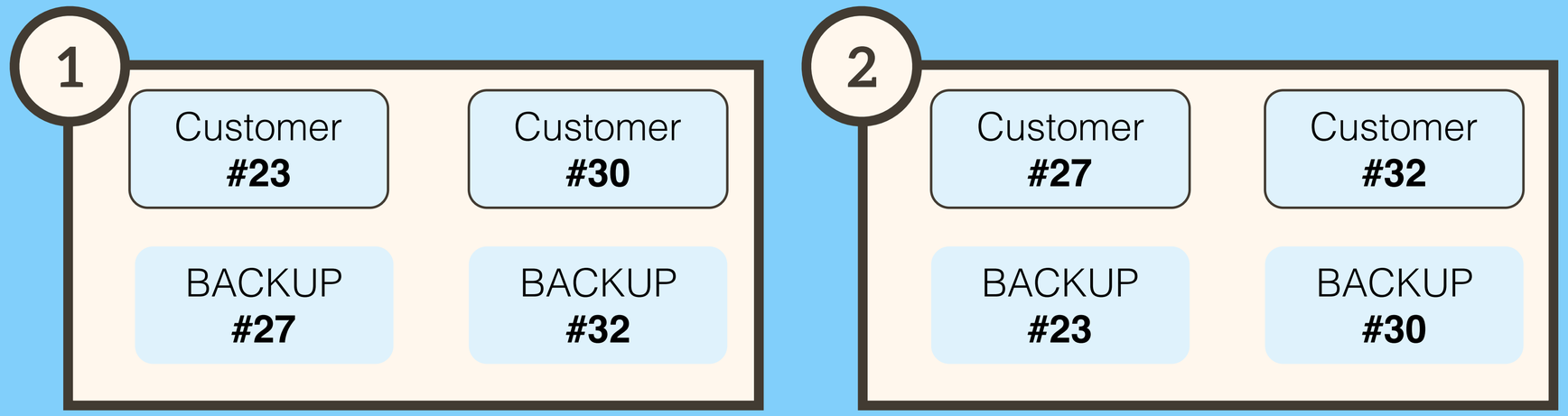
5

*Use a distributed cache for big amounts of data*

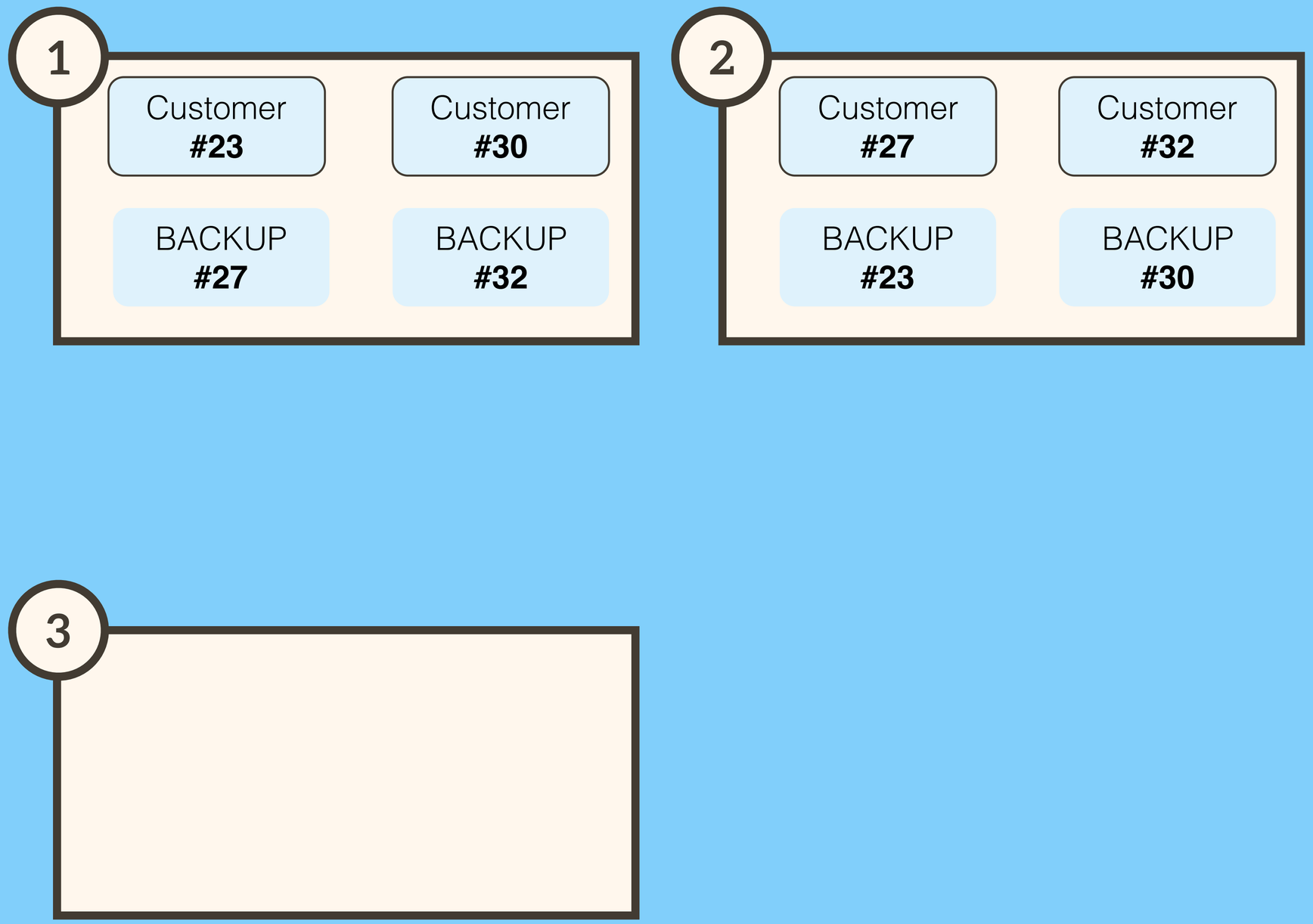
# Distributed Caches

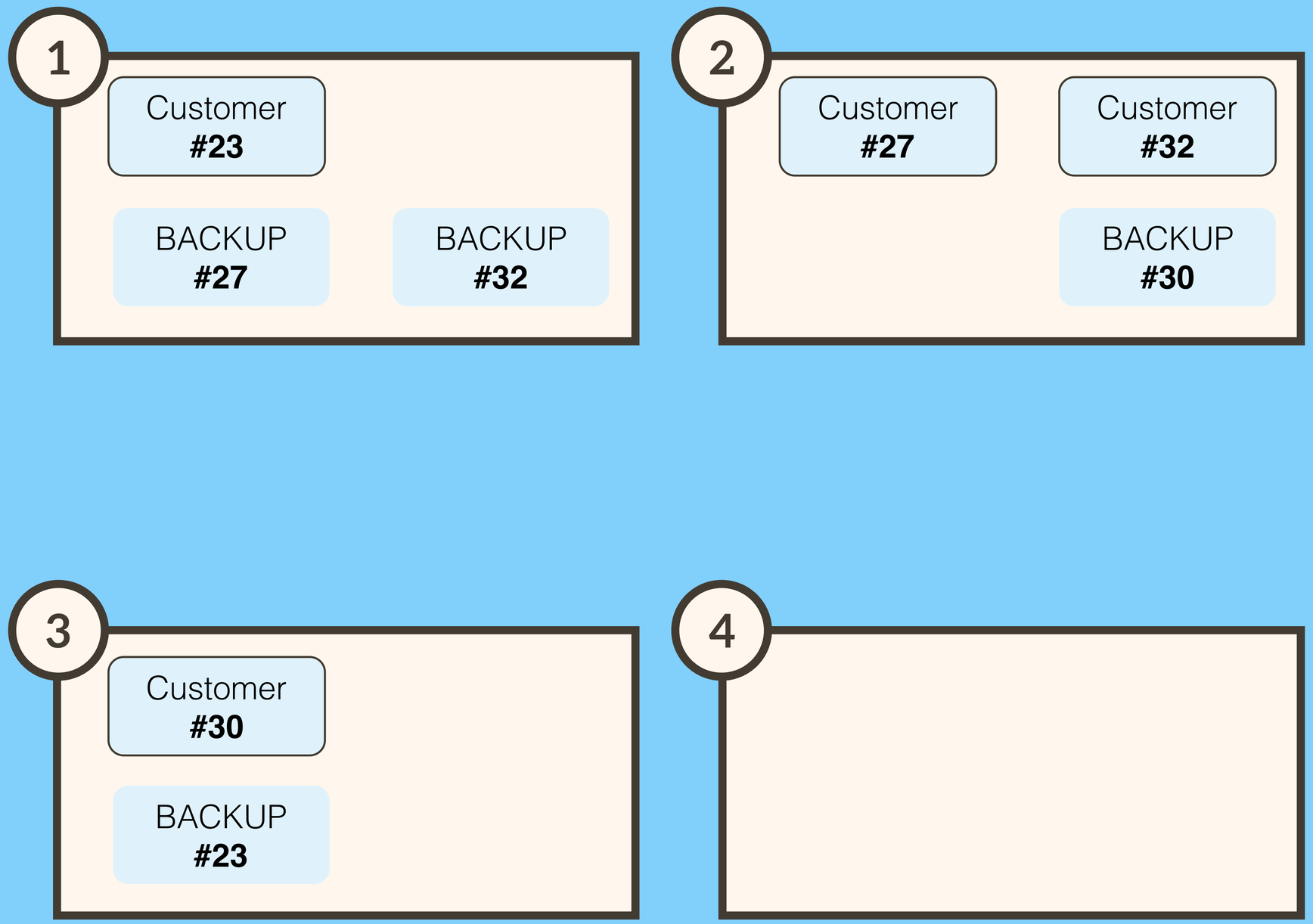



1
Customer
#23
Customer
#30
Customer
#27
Customer
#32
2




# *Data is being distributed and backed up*


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


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

1

Customer
**#23**

BACKUP
**#27**

2

Customer
**#27**

BACKUP
**#32**

3

Customer
**#30**

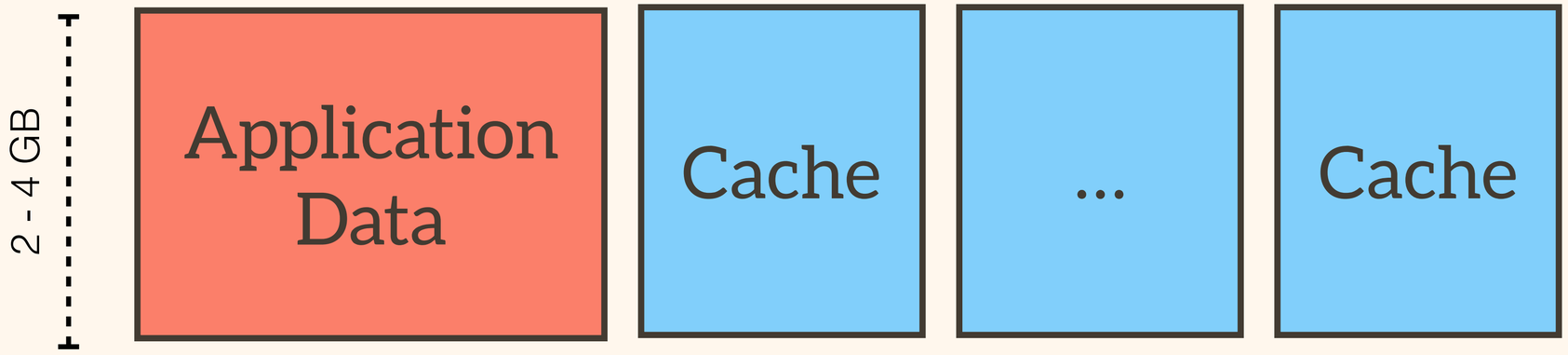
BACKUP
**#23**

4

Customer
**#32**

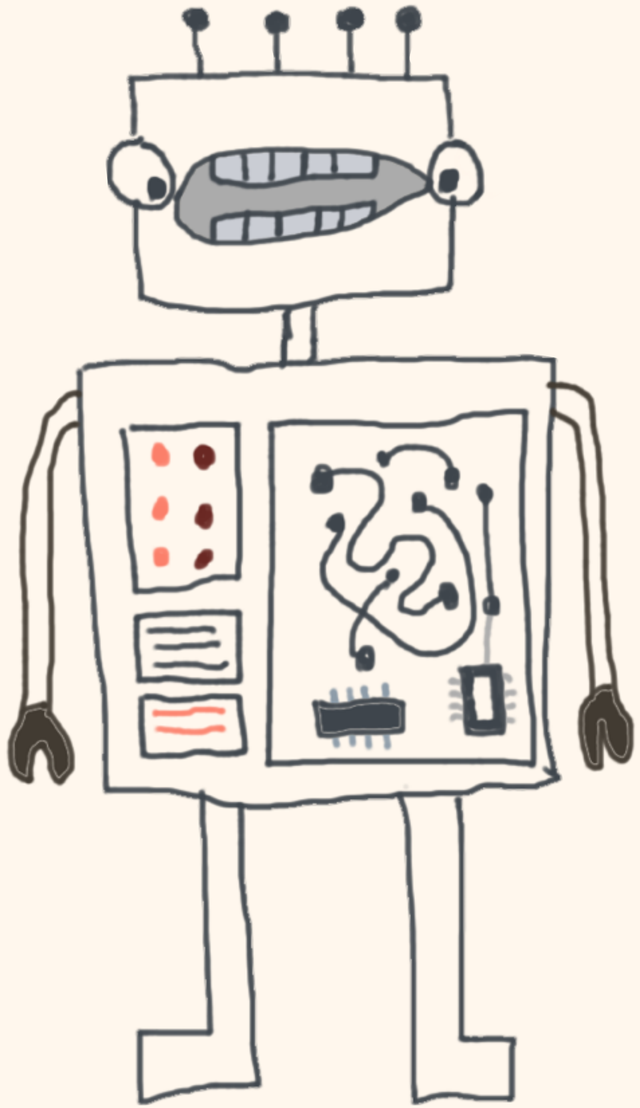
BACKUP
**#30**

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

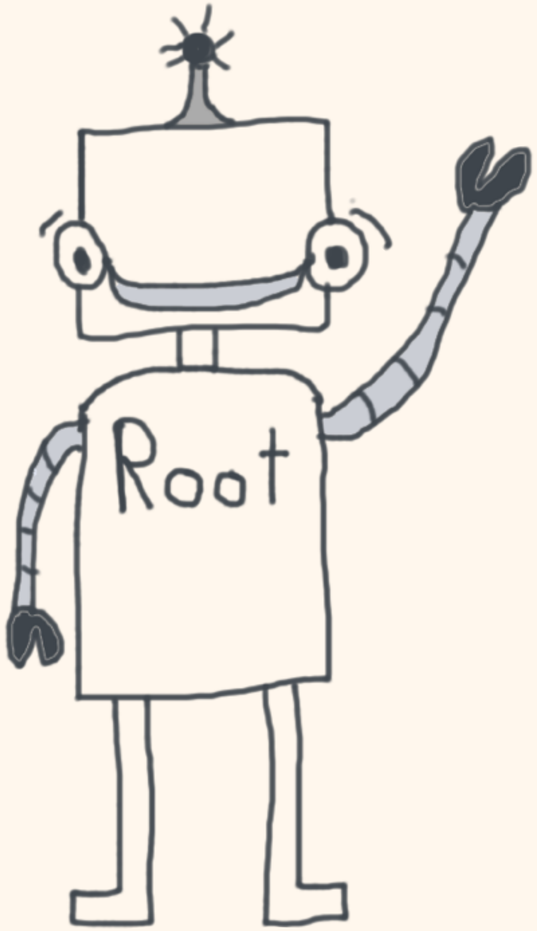


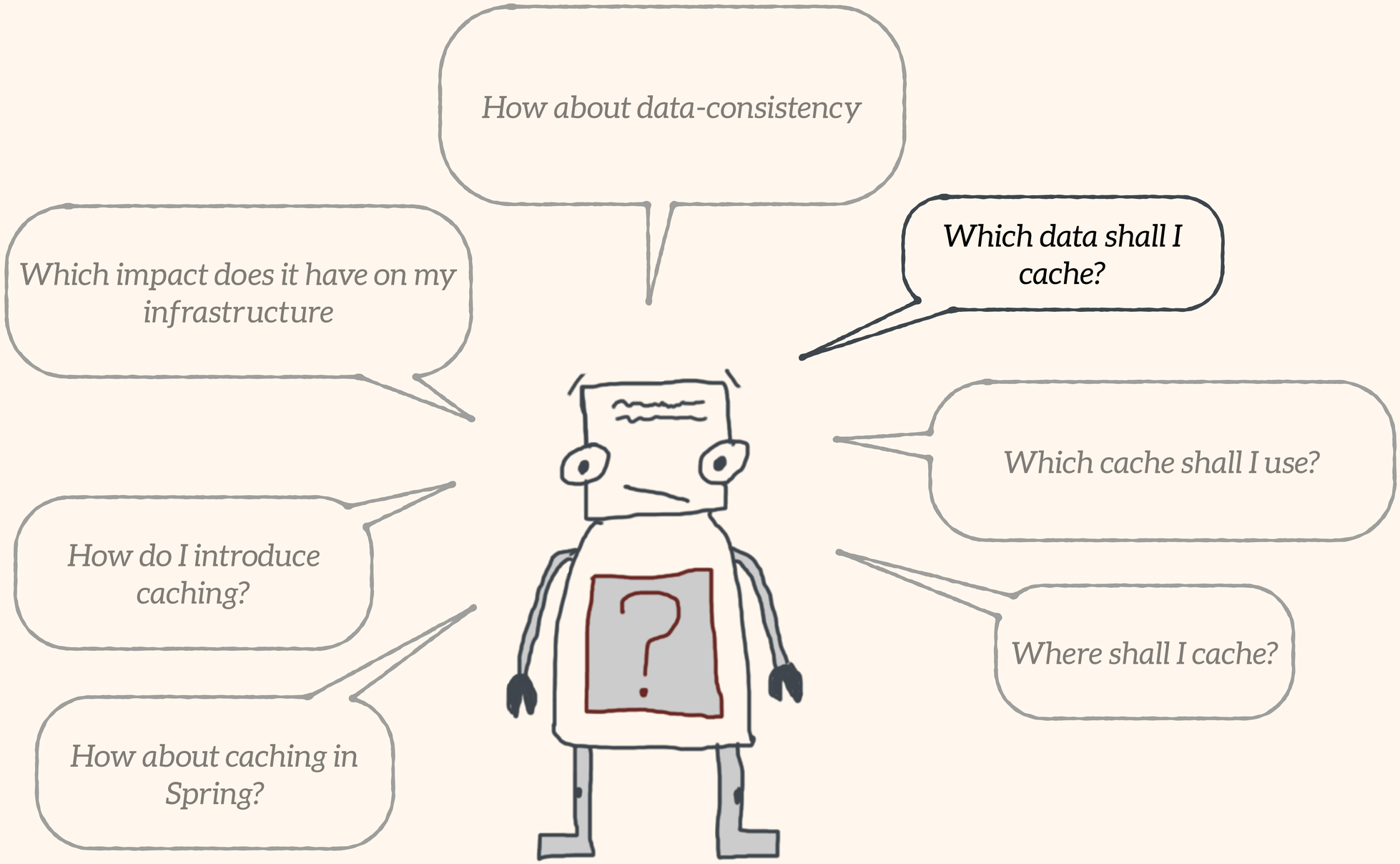
6

## *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.***



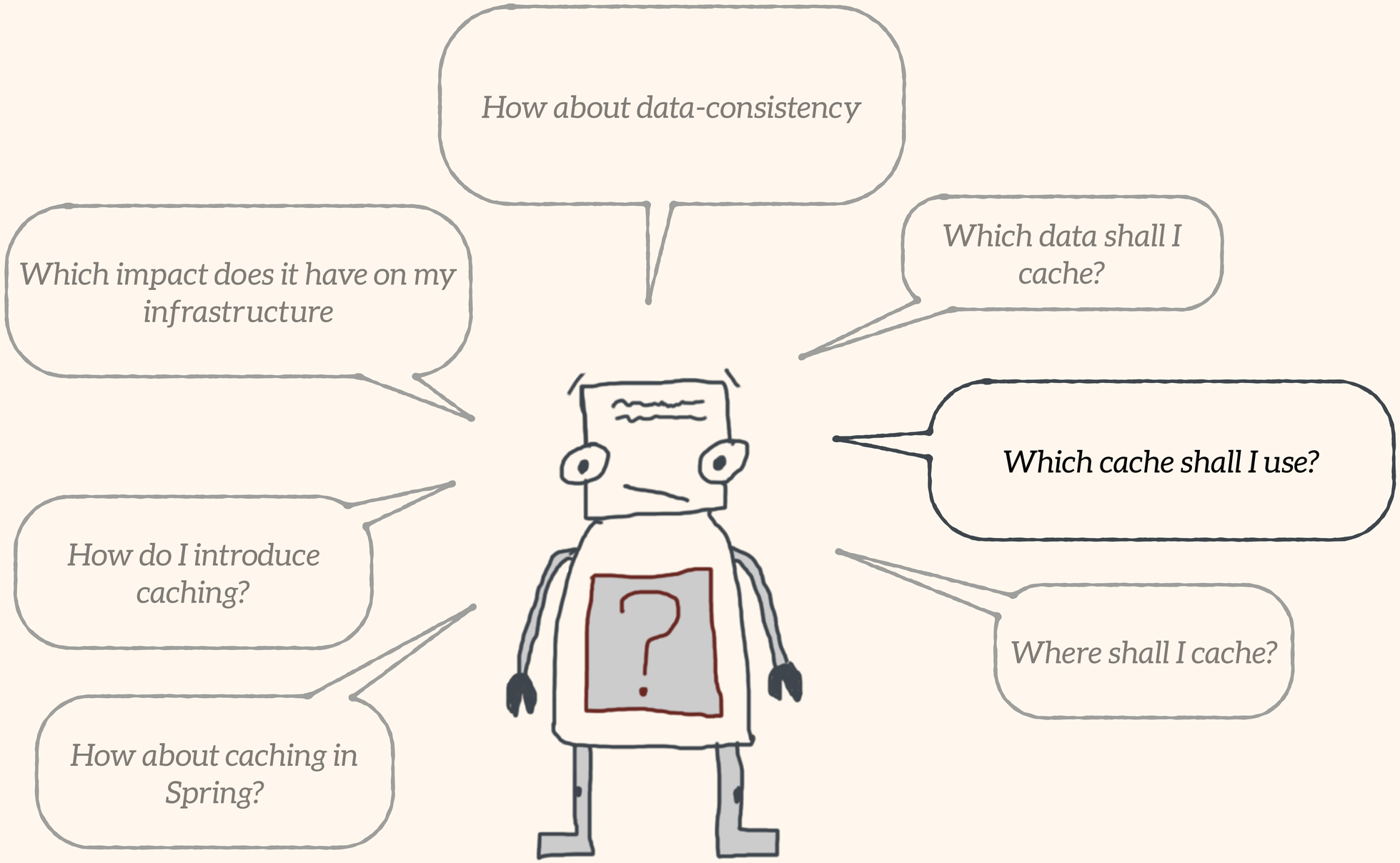

How about data-consistency
Which data shall I cache?
Which impact does it have on my infrastructure
Which cache shall I use?
How do I introduce caching?
Where shall I cache?
How about caching in Spring?

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?

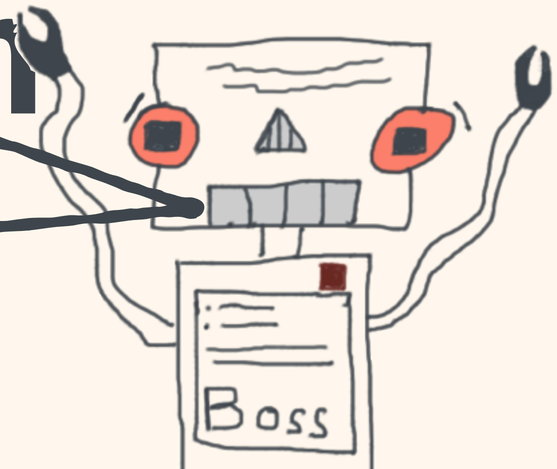
8

*Only use existing cache implementations*

# NEVER

write your own cache implementation

EVER

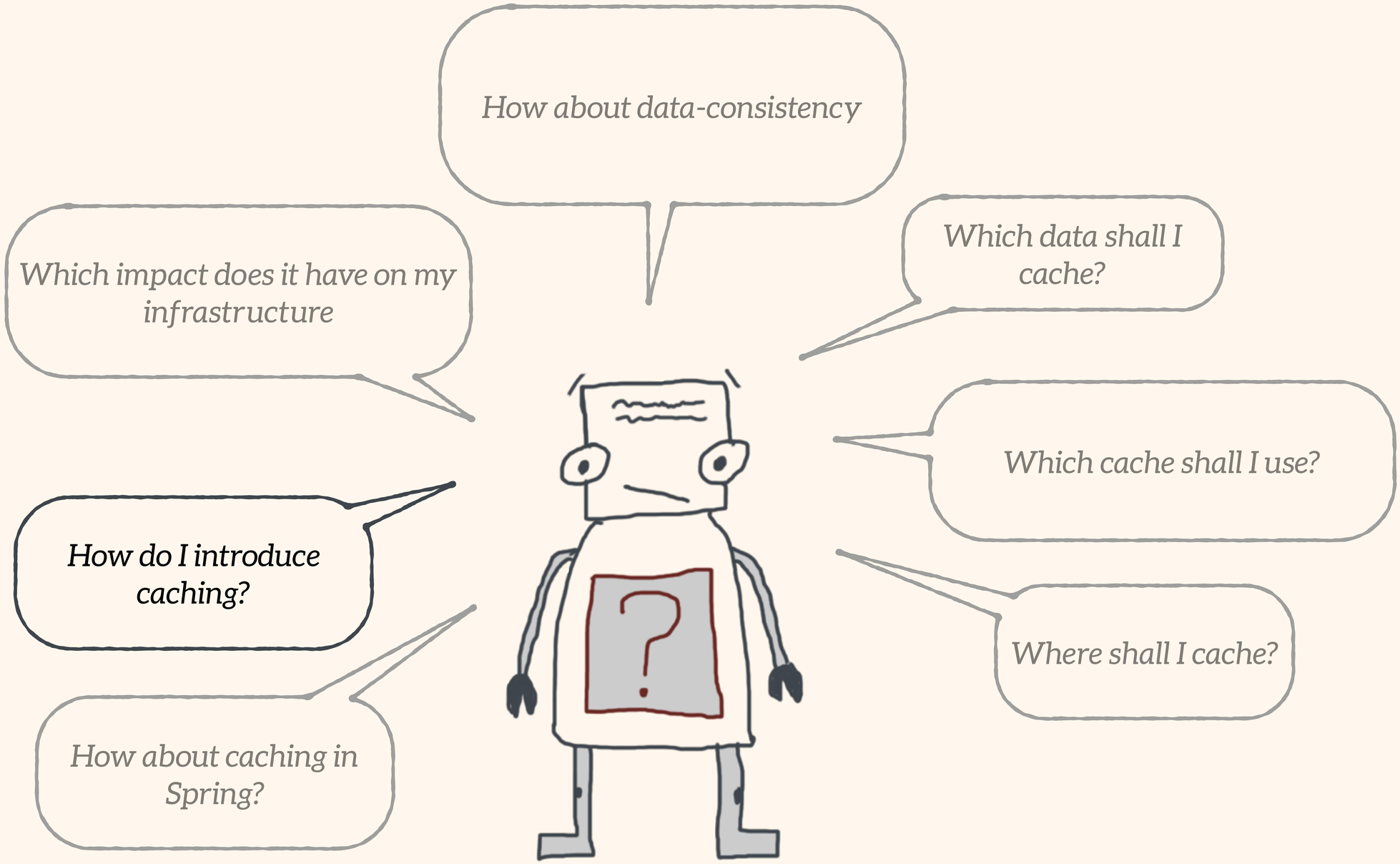


Infinispan, EHCache, Hazelcast, Couchbase, Memcache, OSCache, SwarmCache, Xtreme Cache, Apache DirectMemory

# CACHE

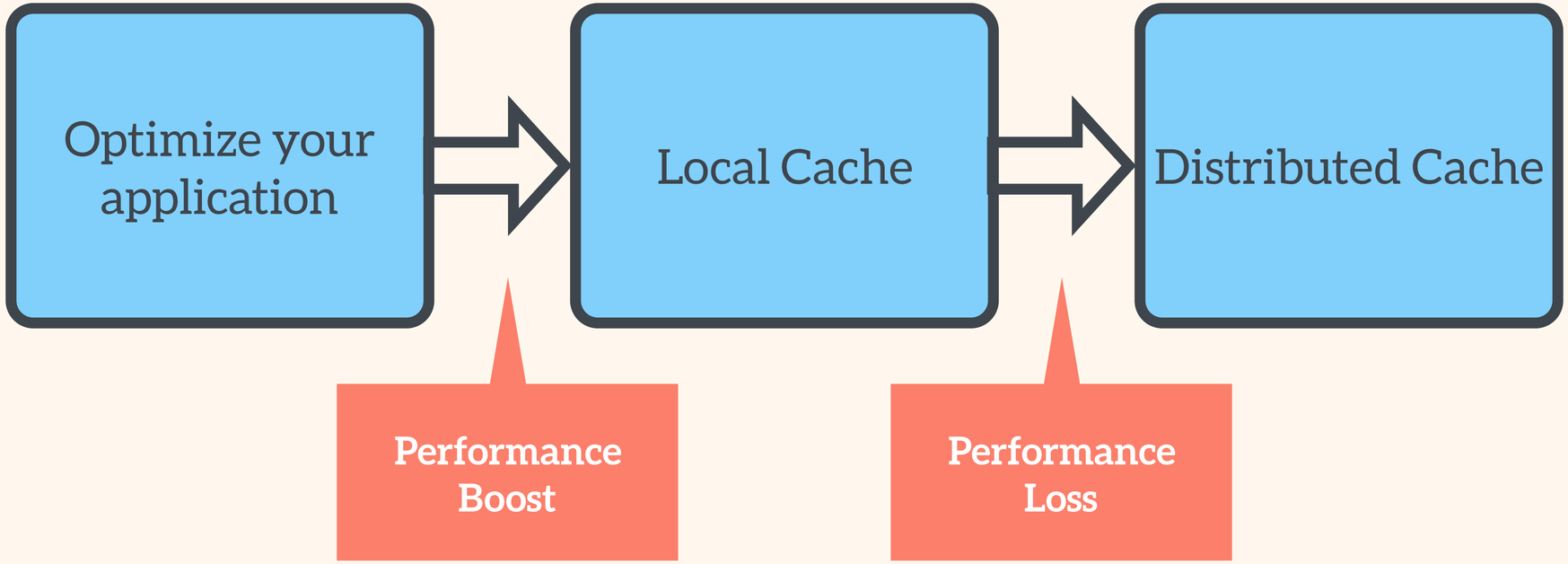
## Implementations

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?
Which cache shall I use?
How do I introduce caching?
Where shall I cache?
How about caching in Spring?

9

# *Introduce Caching in three steps*


Optimize your application
Local Cache
Distributed Cache
Performance Boost
Performance Loss

10

# *Optimize Serialization*

# *Example: Hazelcast*

## *putting and getting 10.000 objects locally*

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

# *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

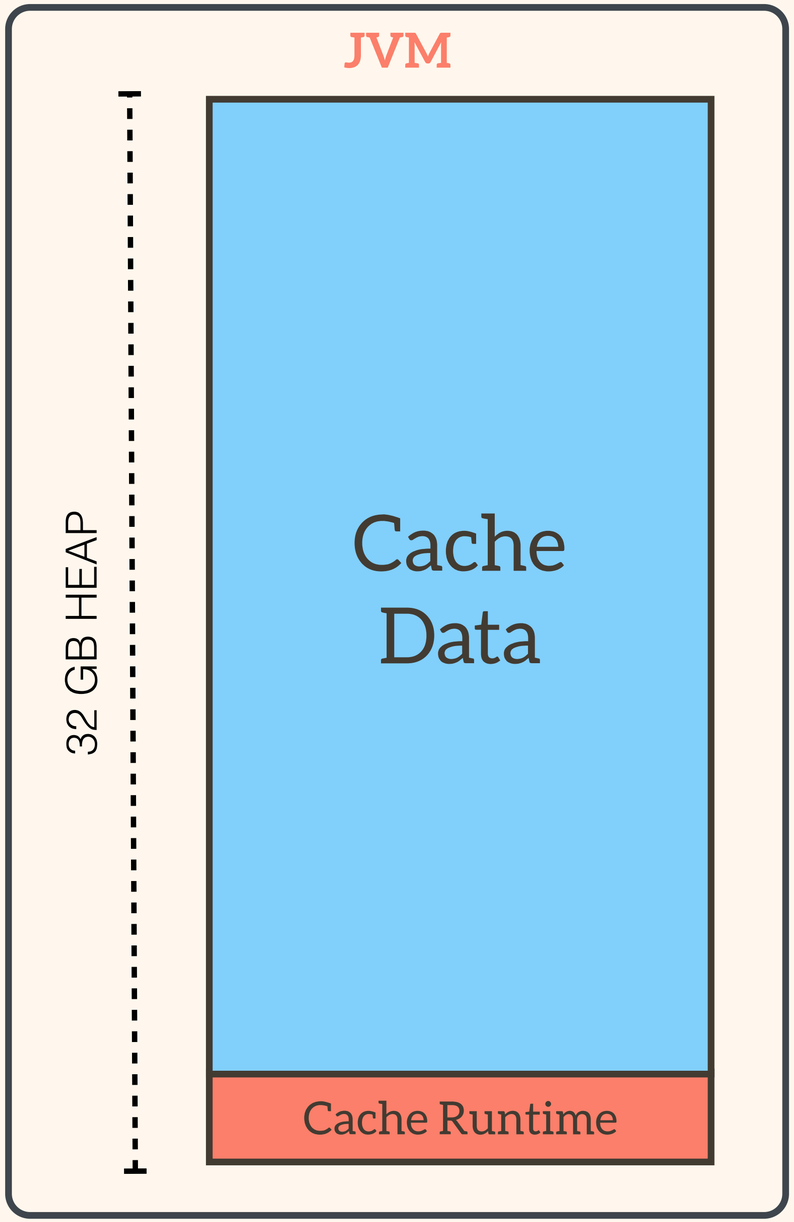
# *SERIALIZATION*

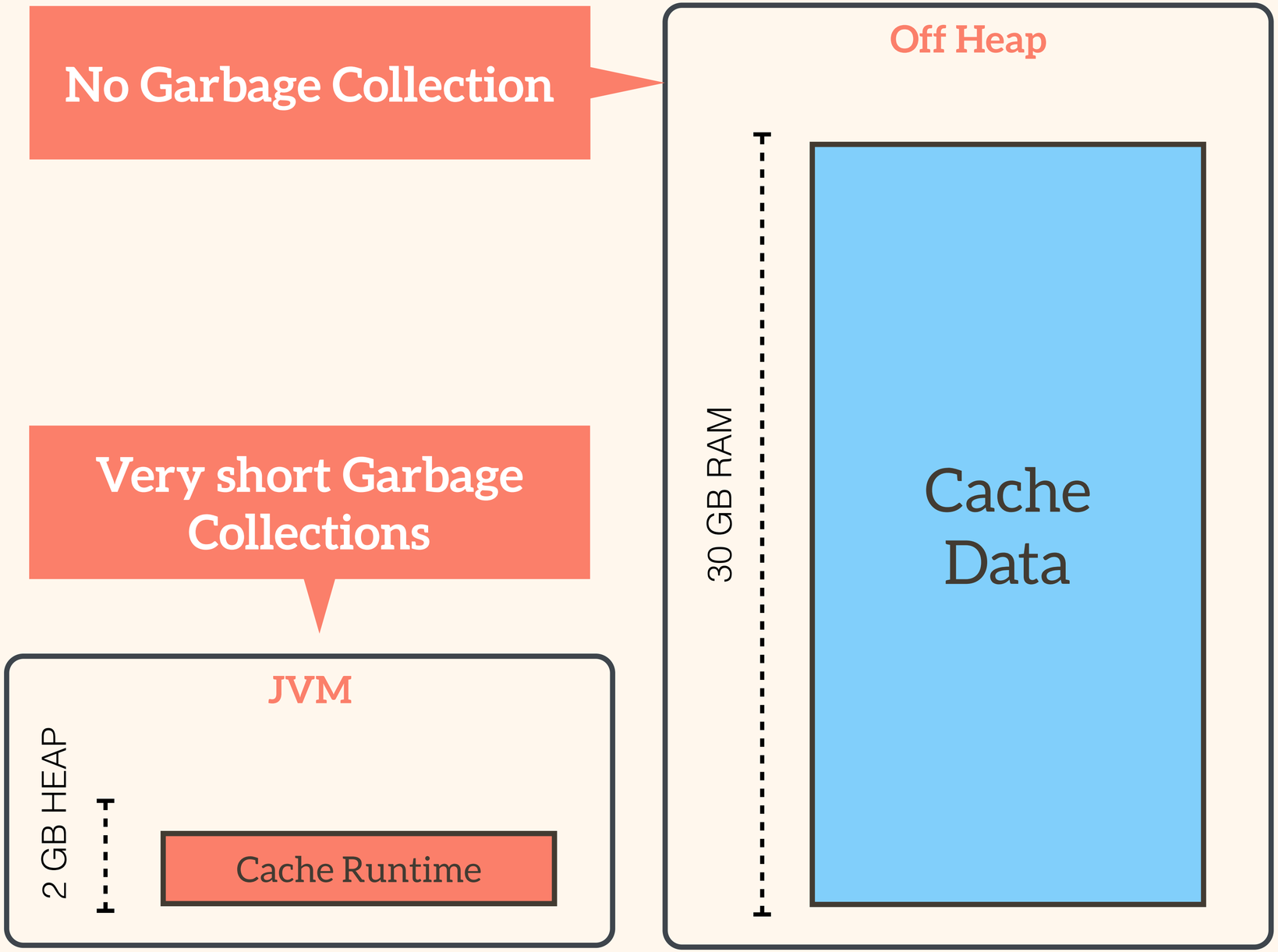
# SUCKS

*for Caching if alternatives are present*

11

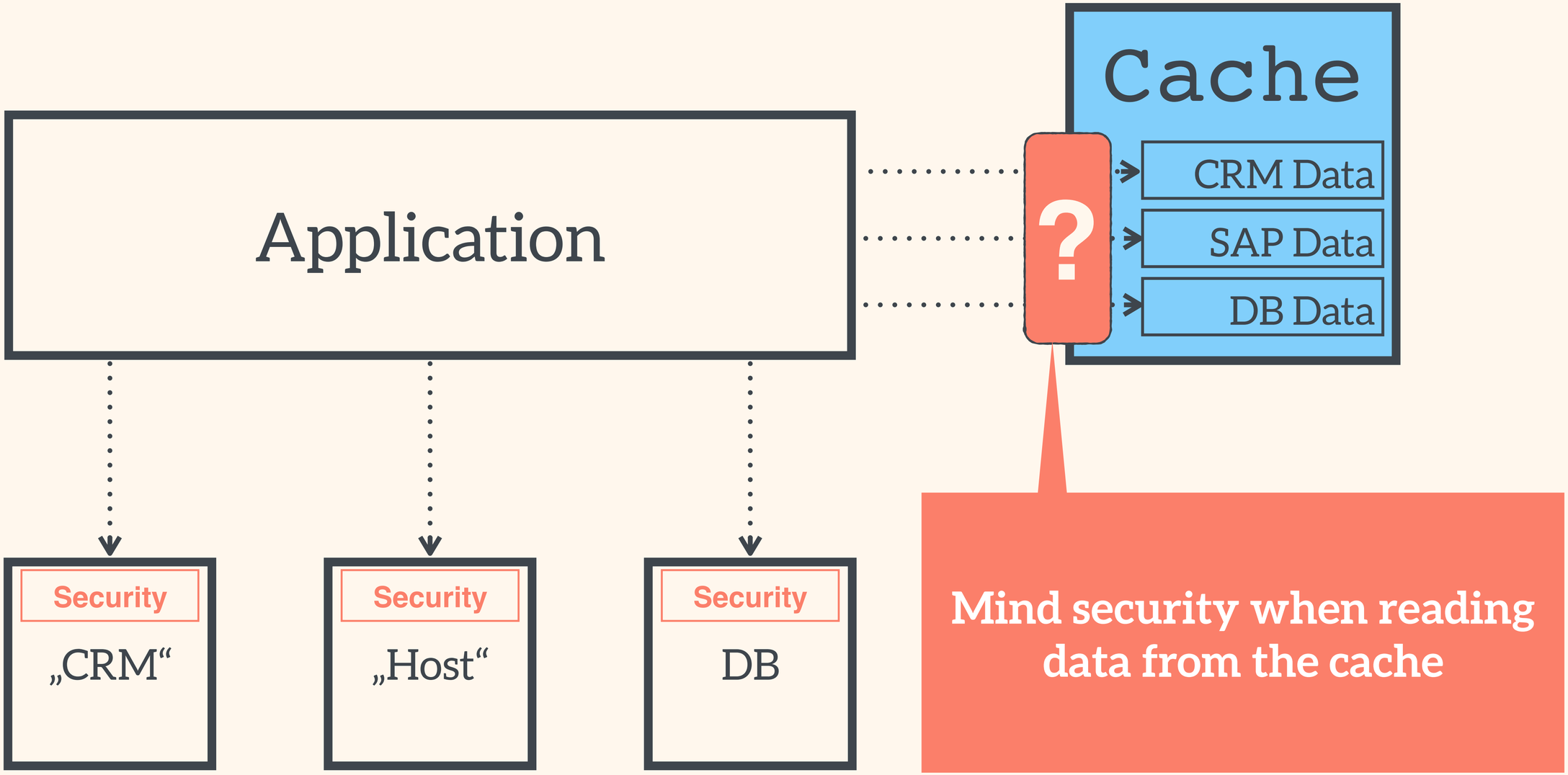
# *Use Off-Heap Storage for Cache instances with more than 4 GB Heap Size*


JVM
32 GB HEAP
Cache
Data
Cache Runtime


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

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 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 |

VOXXED DAYS BERLIN

# Thank You!

**Michael Plöd**
**innoQ Deutschland GmbH**

**@bitboss**
**https://slideshare.net/mploed**

Platinum Sponsor

servers.com