spring I/O

BCN 19&20 MAY 2016

Caching in Spring

Michael Plöd - innoQ Germany
@bitboss

I will talk about

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

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

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

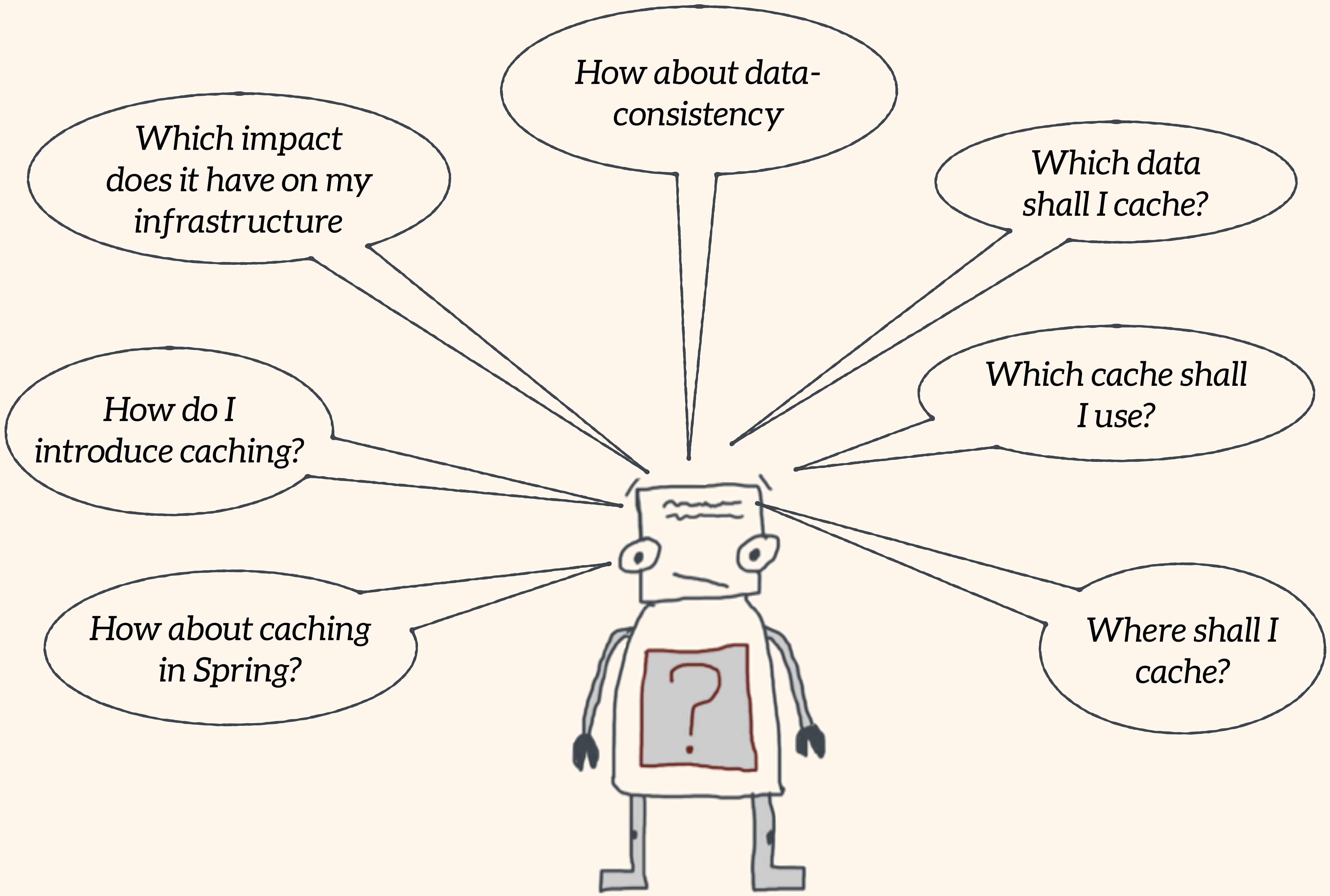
Types of Places for CACHES

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

Business-Applications

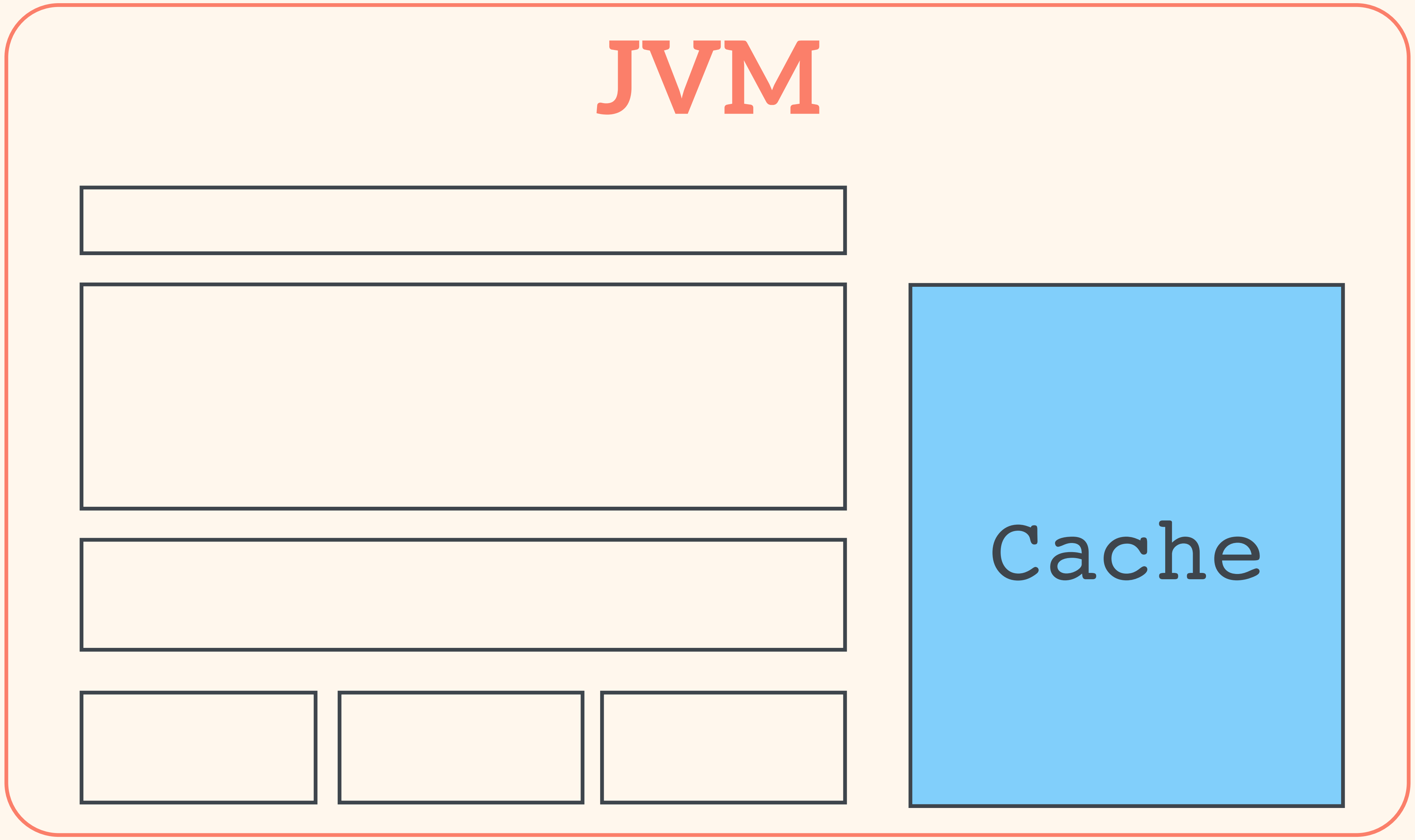
!=

Twitter / Facebook & co.

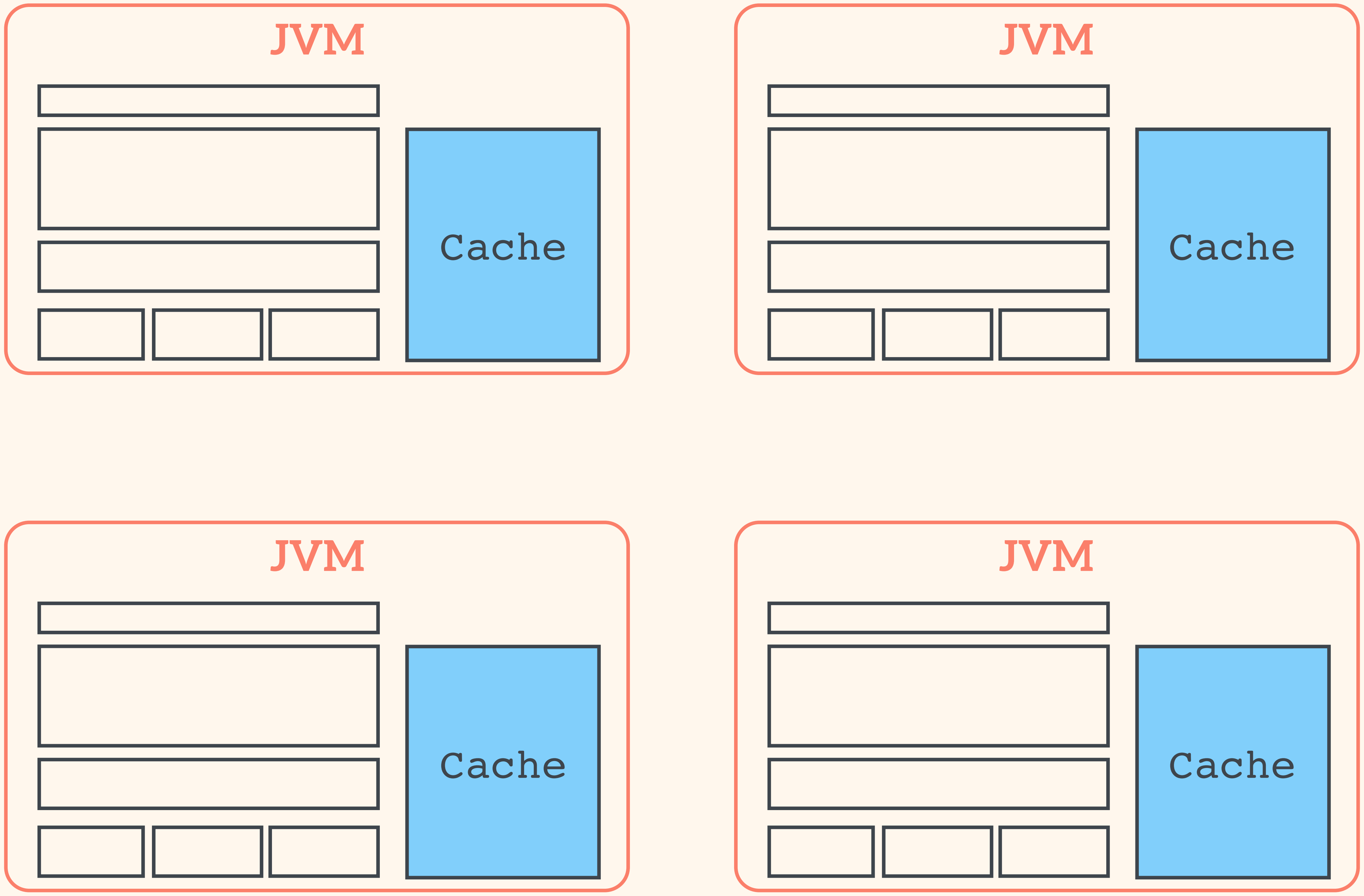

How about data-consistency
Which impact does it have on my infrastructure
Which data shall I cache?
Which cache shall I use?
How do I introduce caching?
How about caching in Spring?
Where shall I cache?

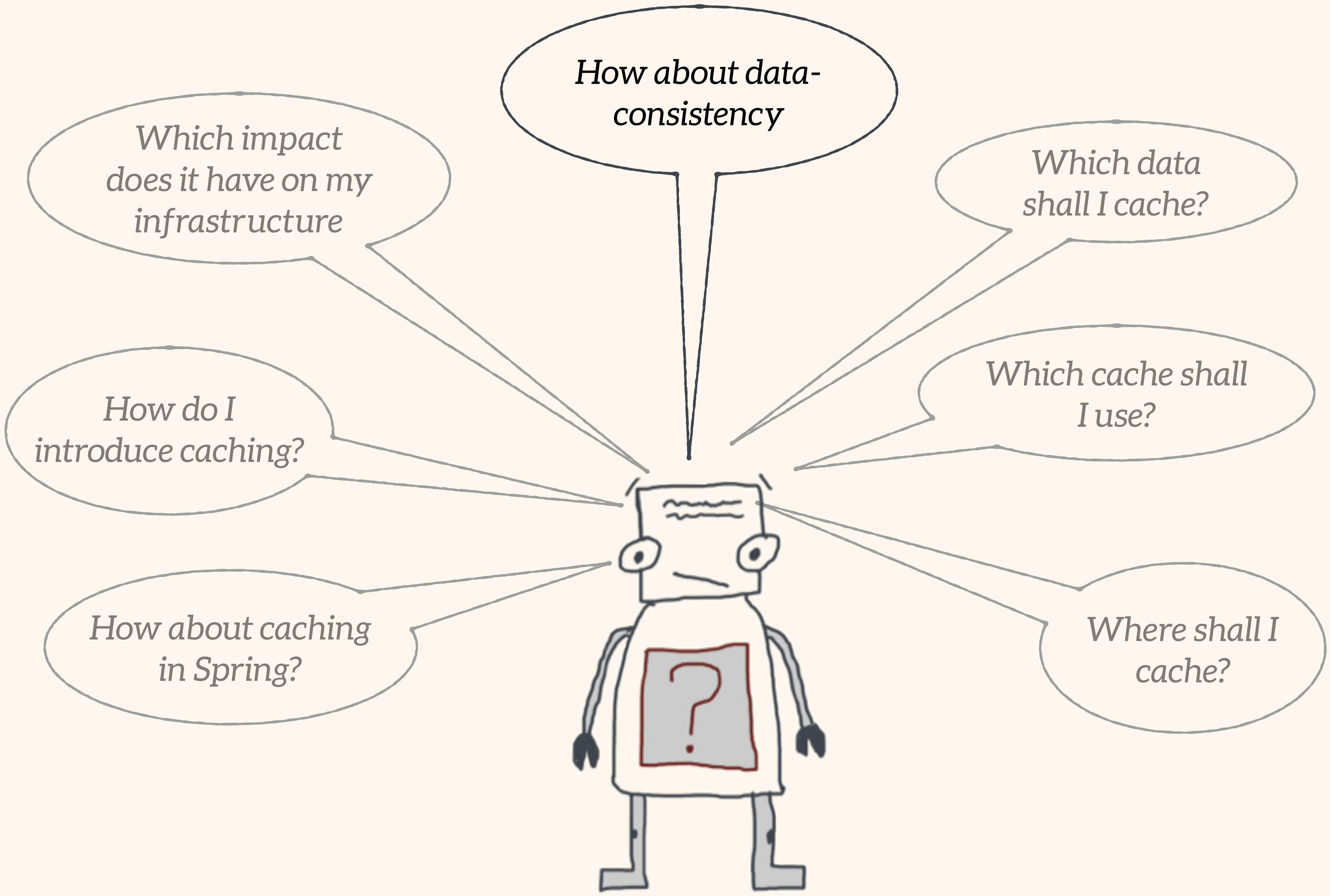
1 *Know your topology*

Local In-Memory

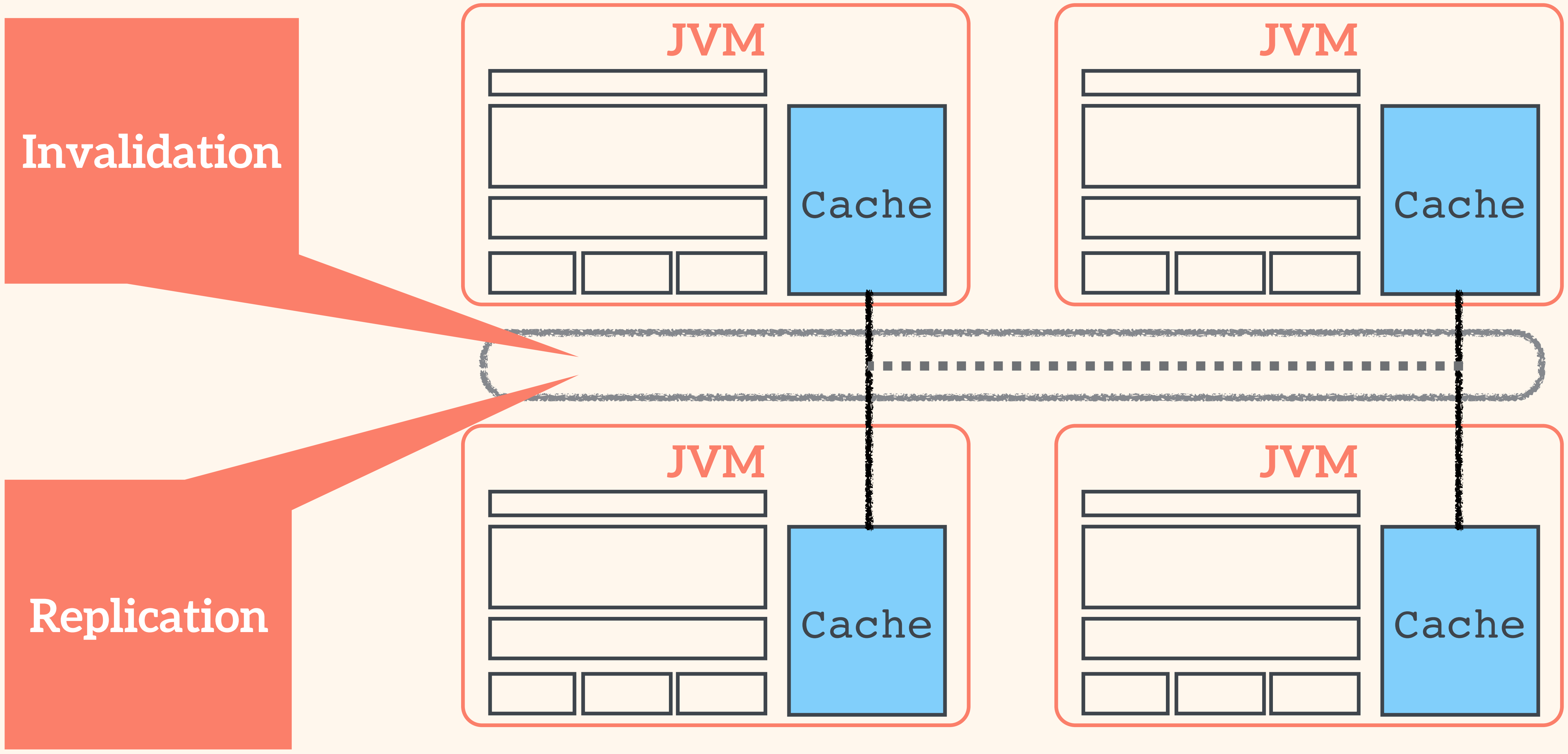


Clustered



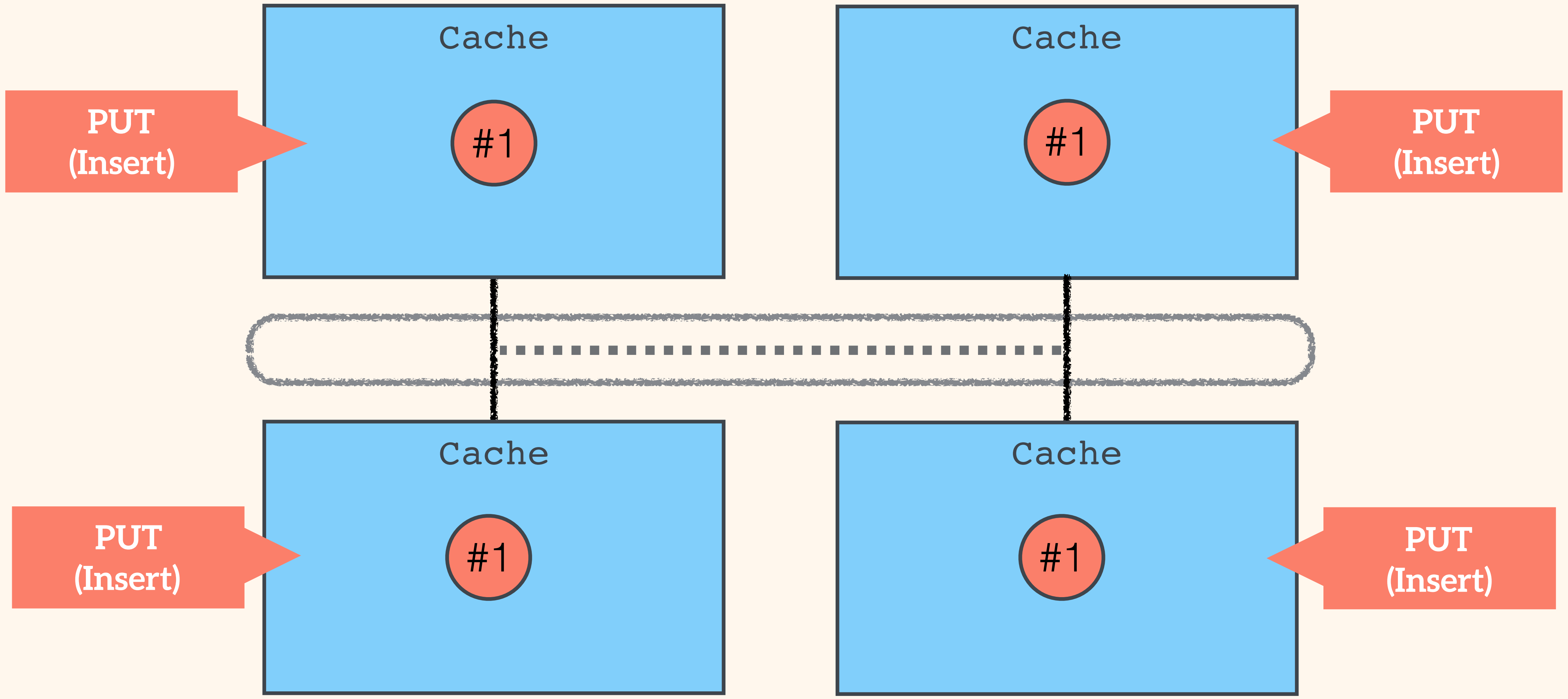

How about data-consistency
Which impact does it have on my infrastructure
Which data shall I cache?
How do I introduce caching?
Which cache shall I use?
How about caching in Spring?
Where shall I cache?
?

Clustered - with sync

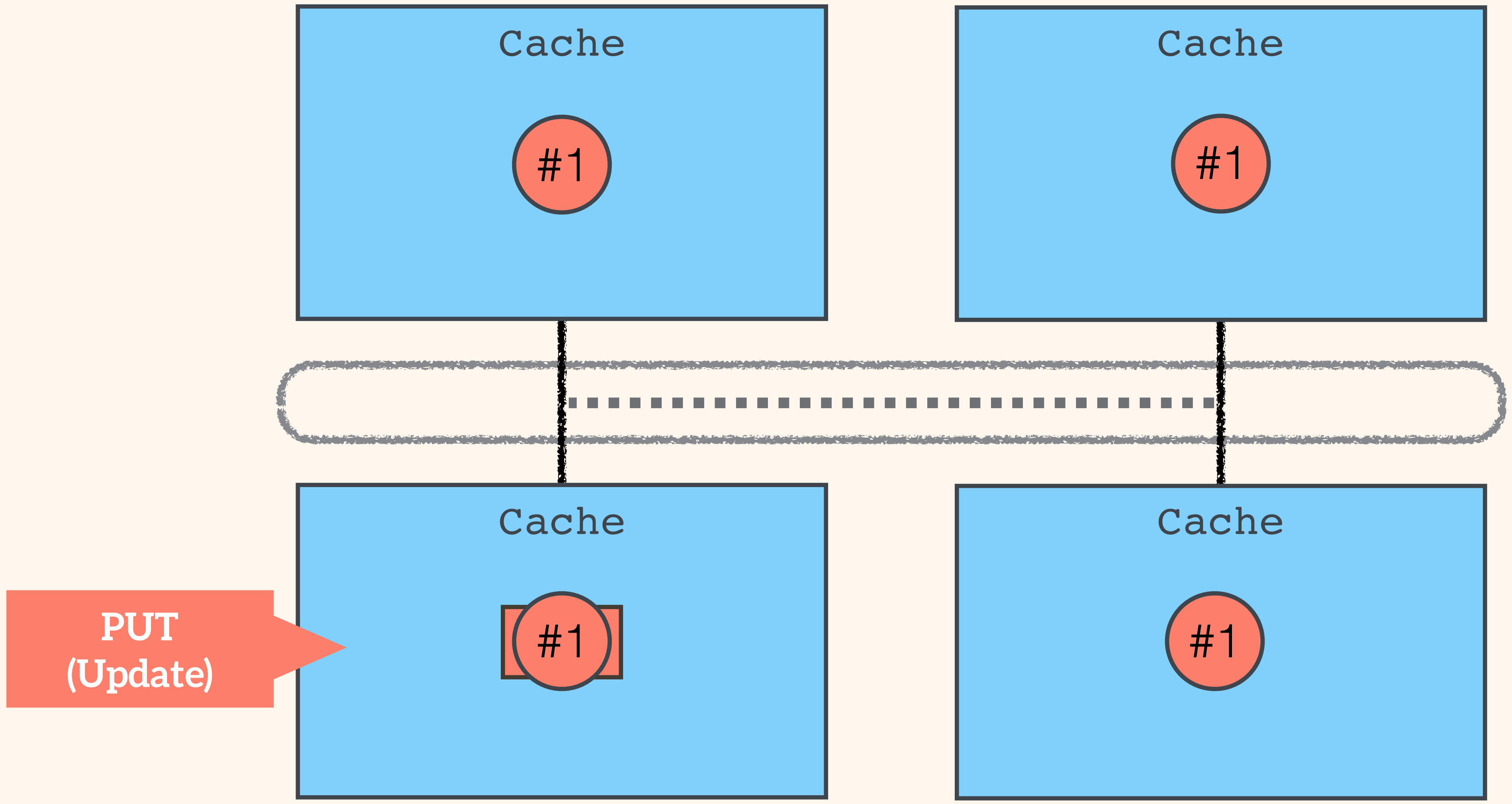


2 *Avoid real replication where possible*

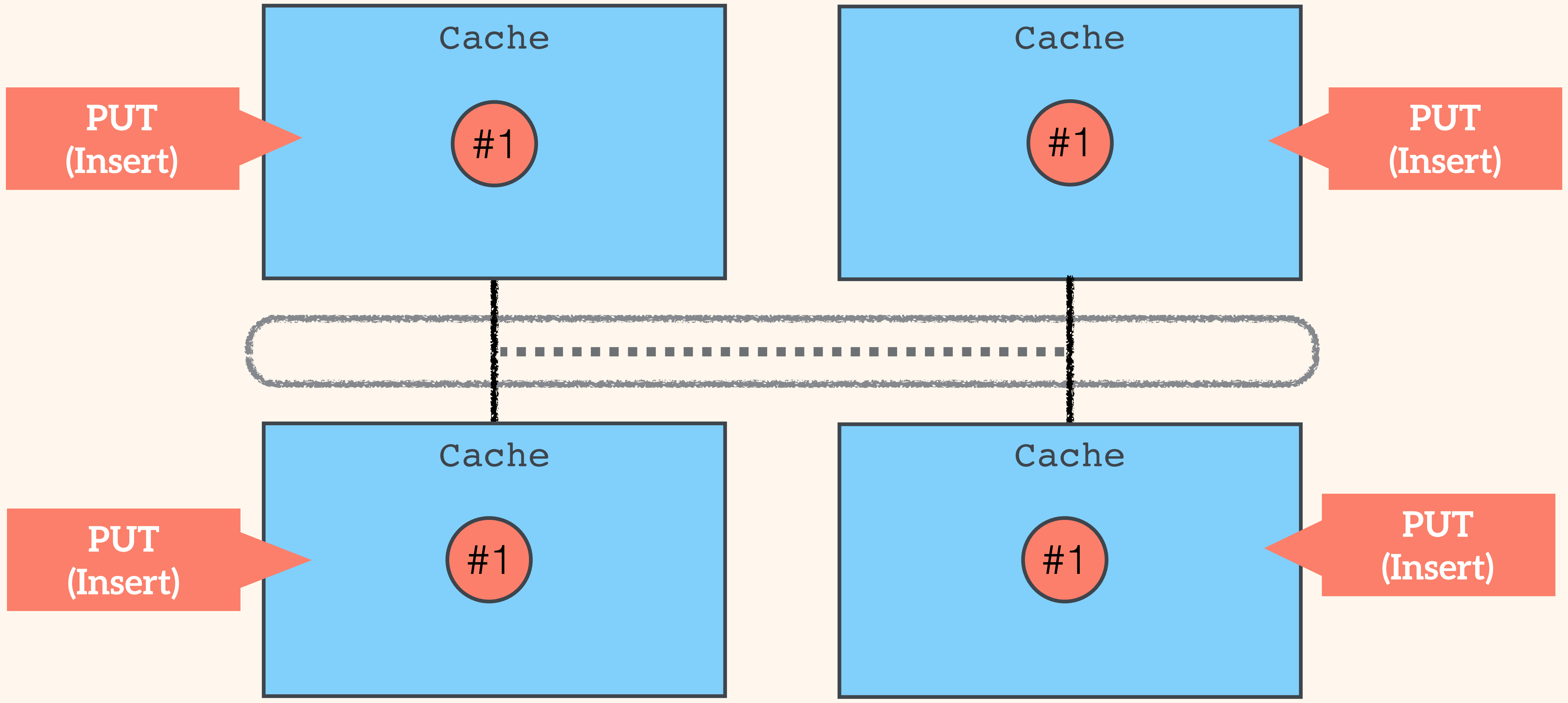
Invalidation - Option 1



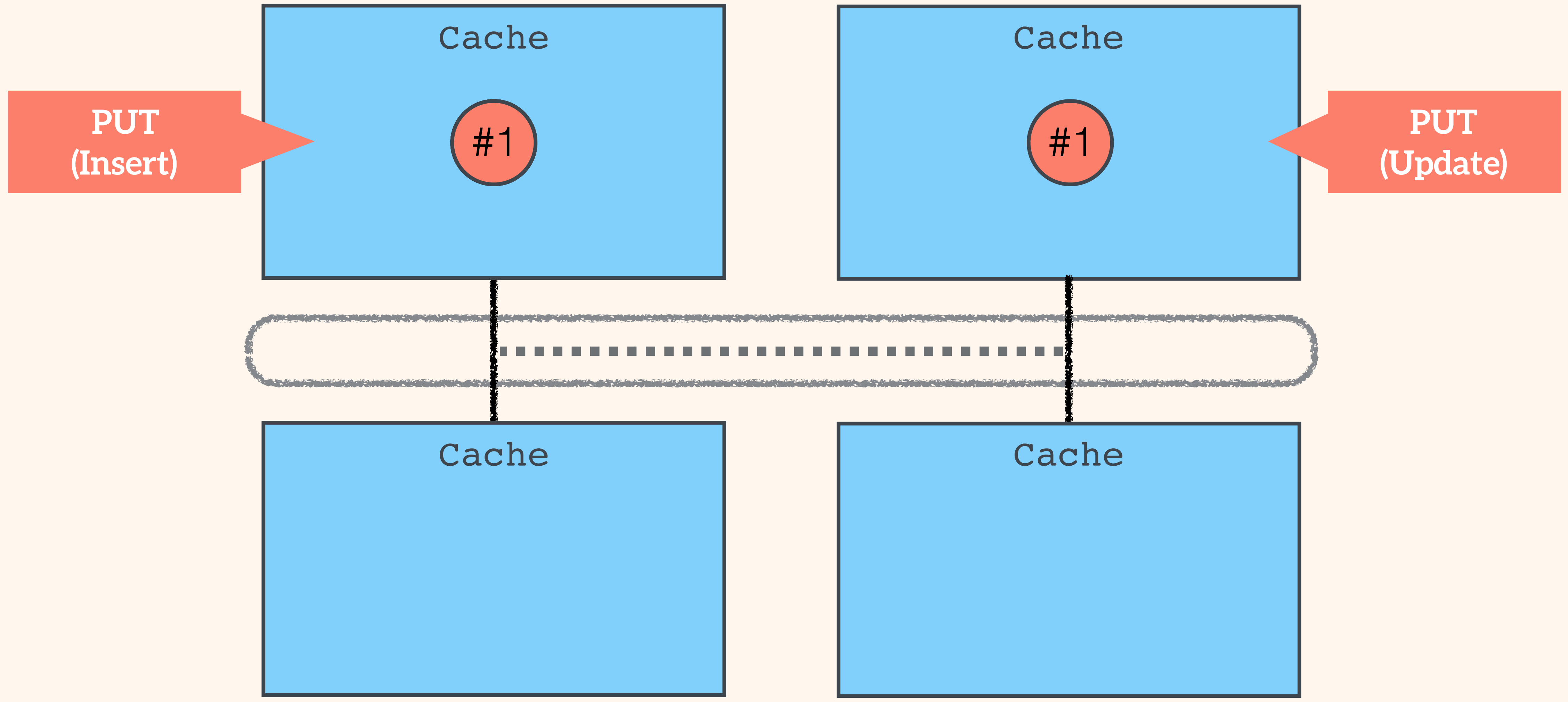
Invalidation - Option 1



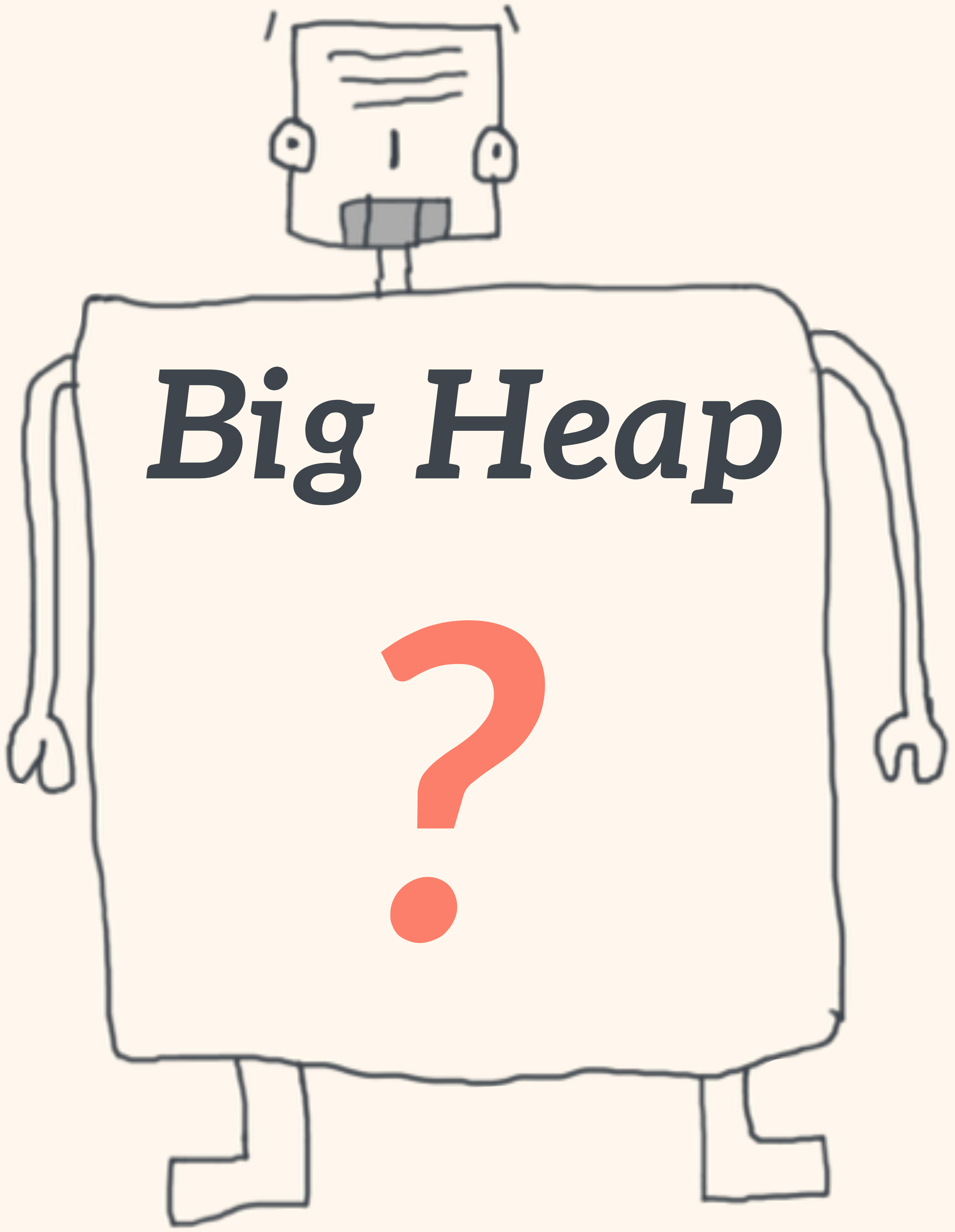
Invalidation - Option 2

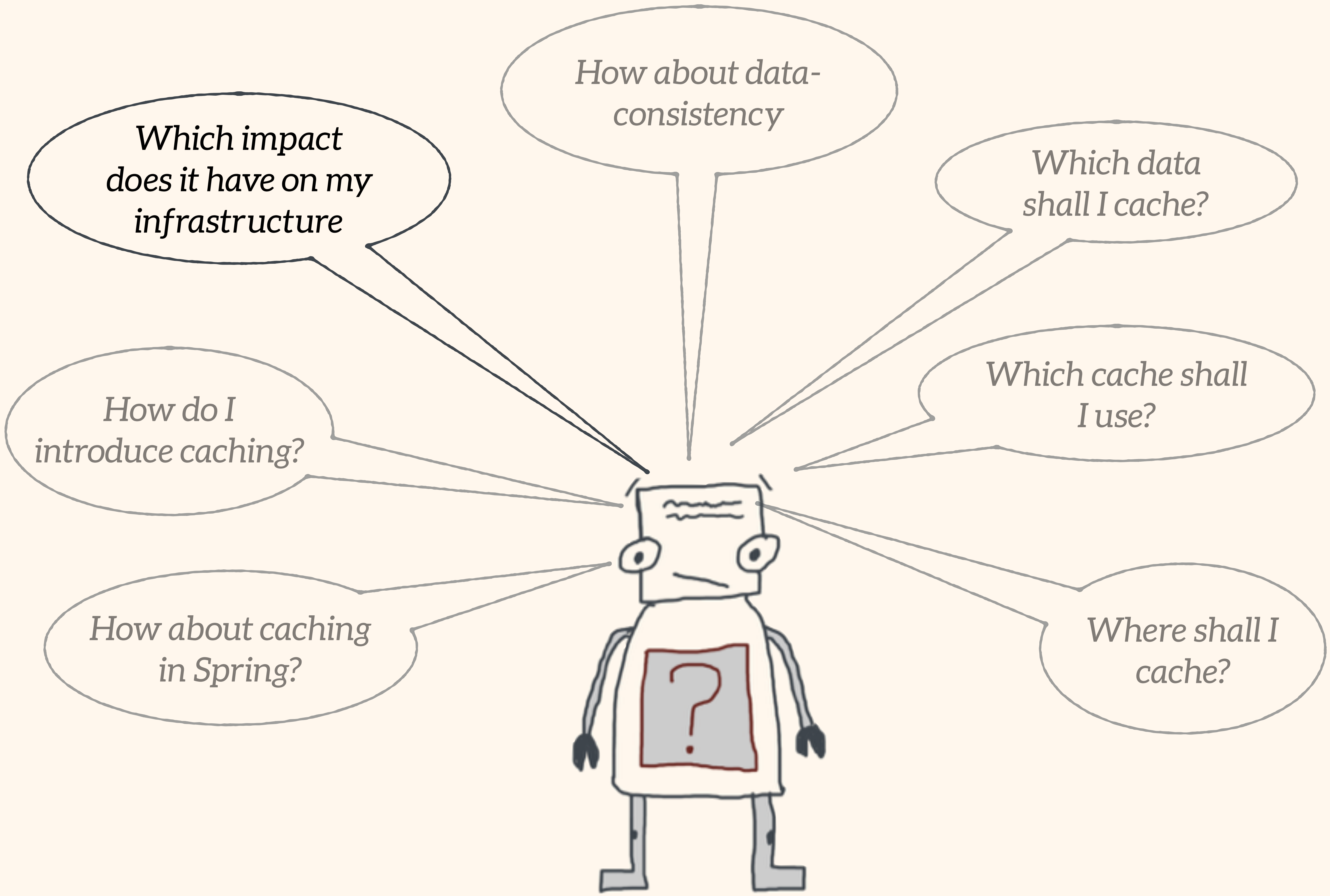


Replication

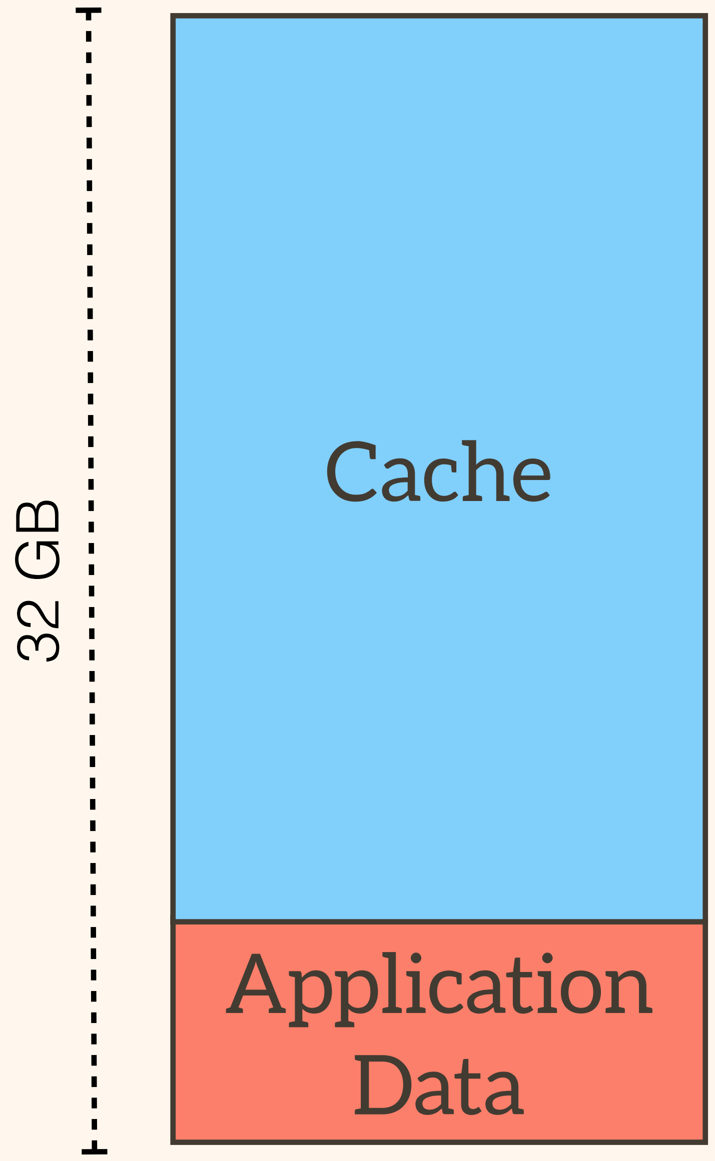


As of now every cache could potentially hold every data which consumes heap memory

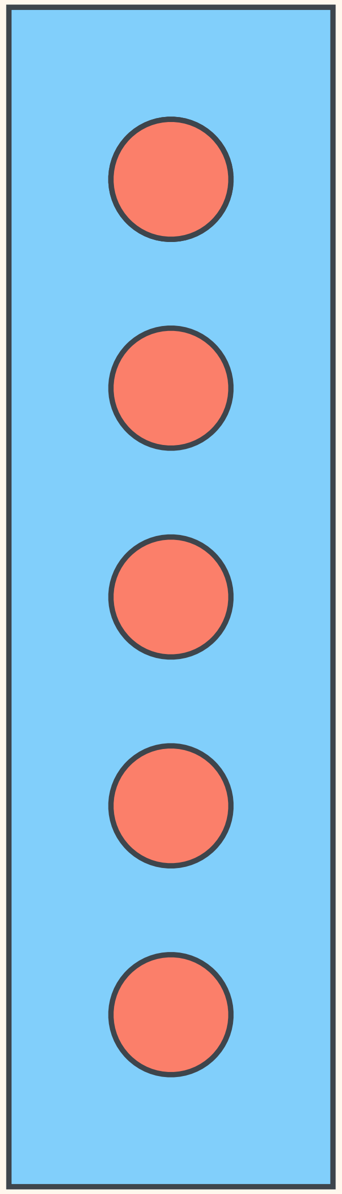

Big Heap
?


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

3 *Avoid big heaps just for caching*



Big heap leads to long major GCs



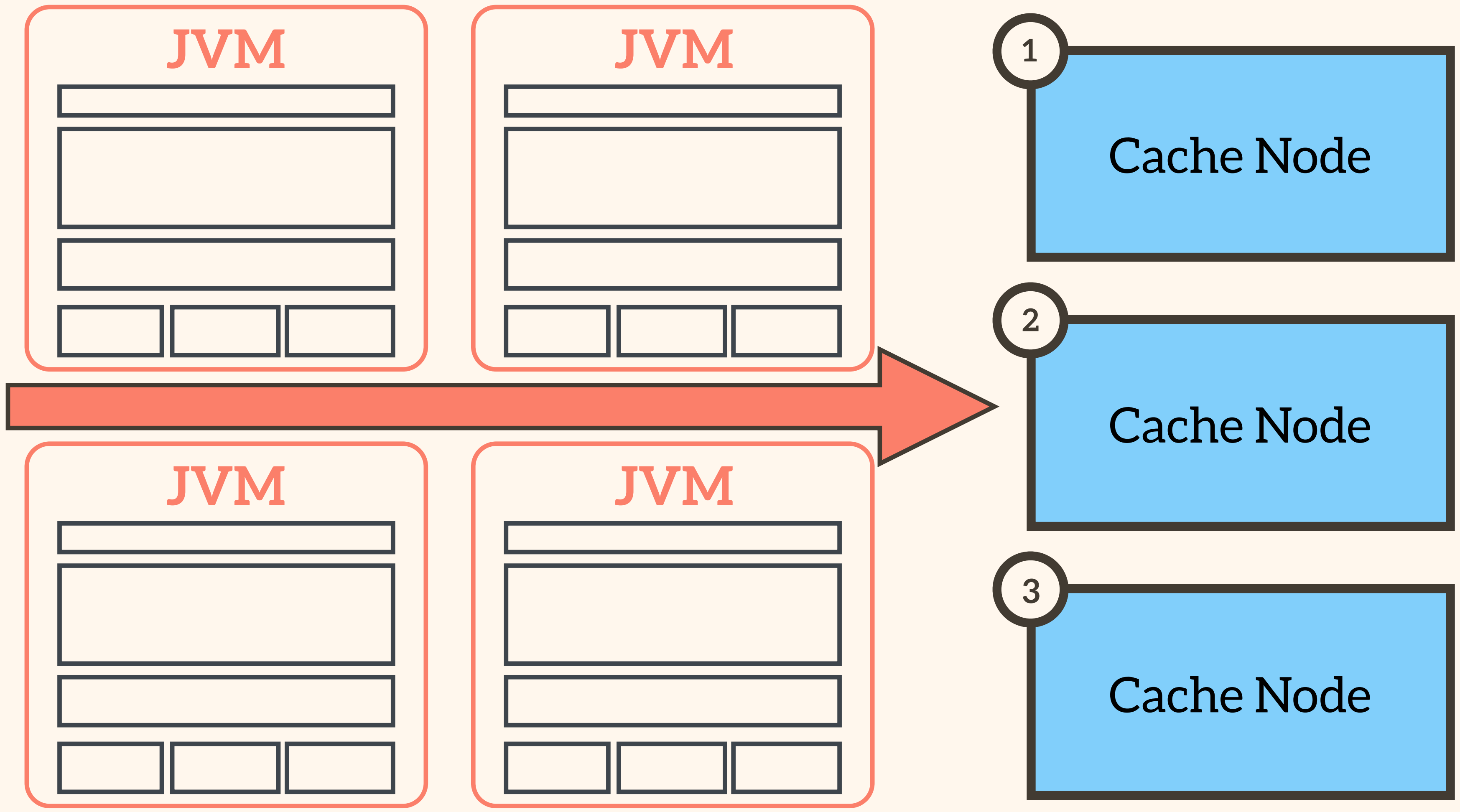
Small caches are a bad idea!

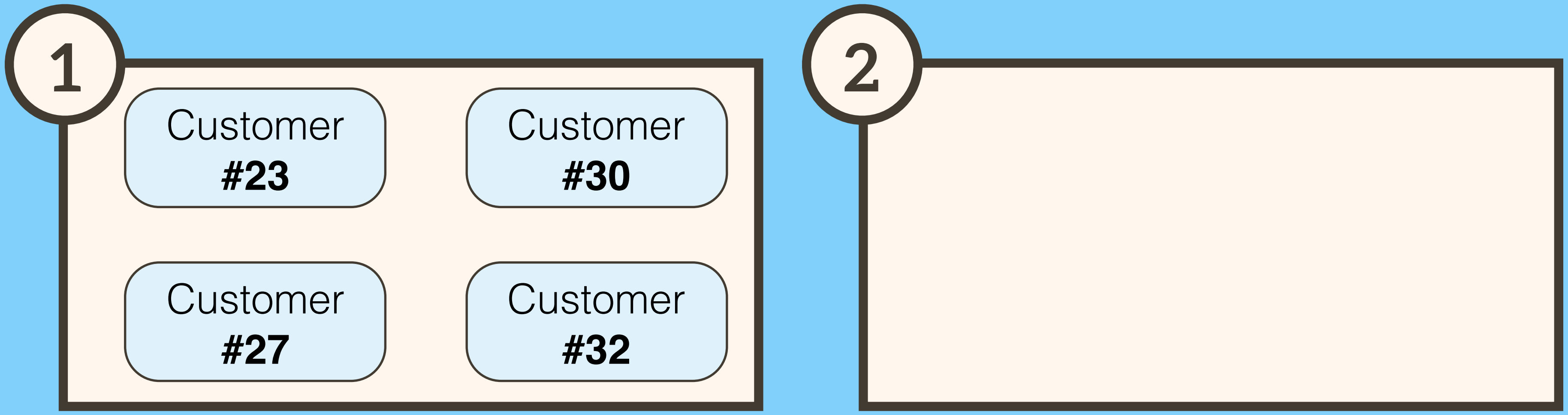
Many evictions,
fewer hits,
no „hot data".

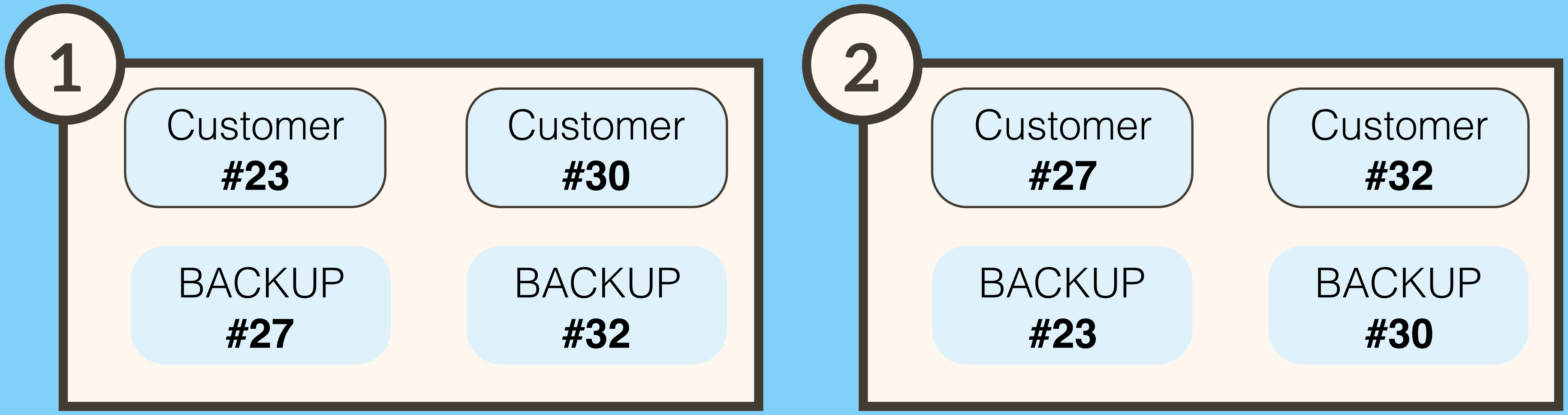
This is especially
critical for
replicating caches.

4 Use a distributed cache for big amounts of data

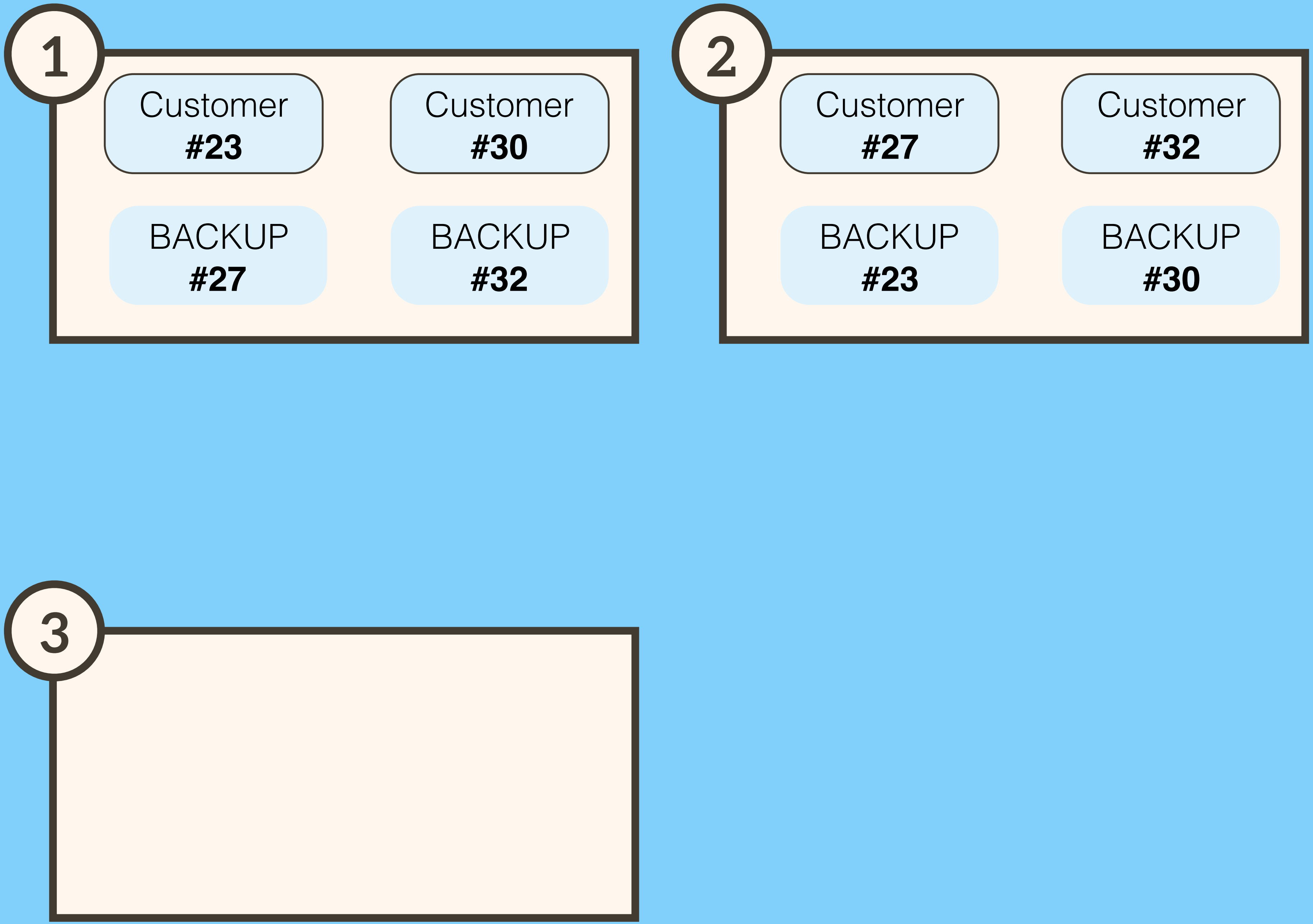
Distributed Caches

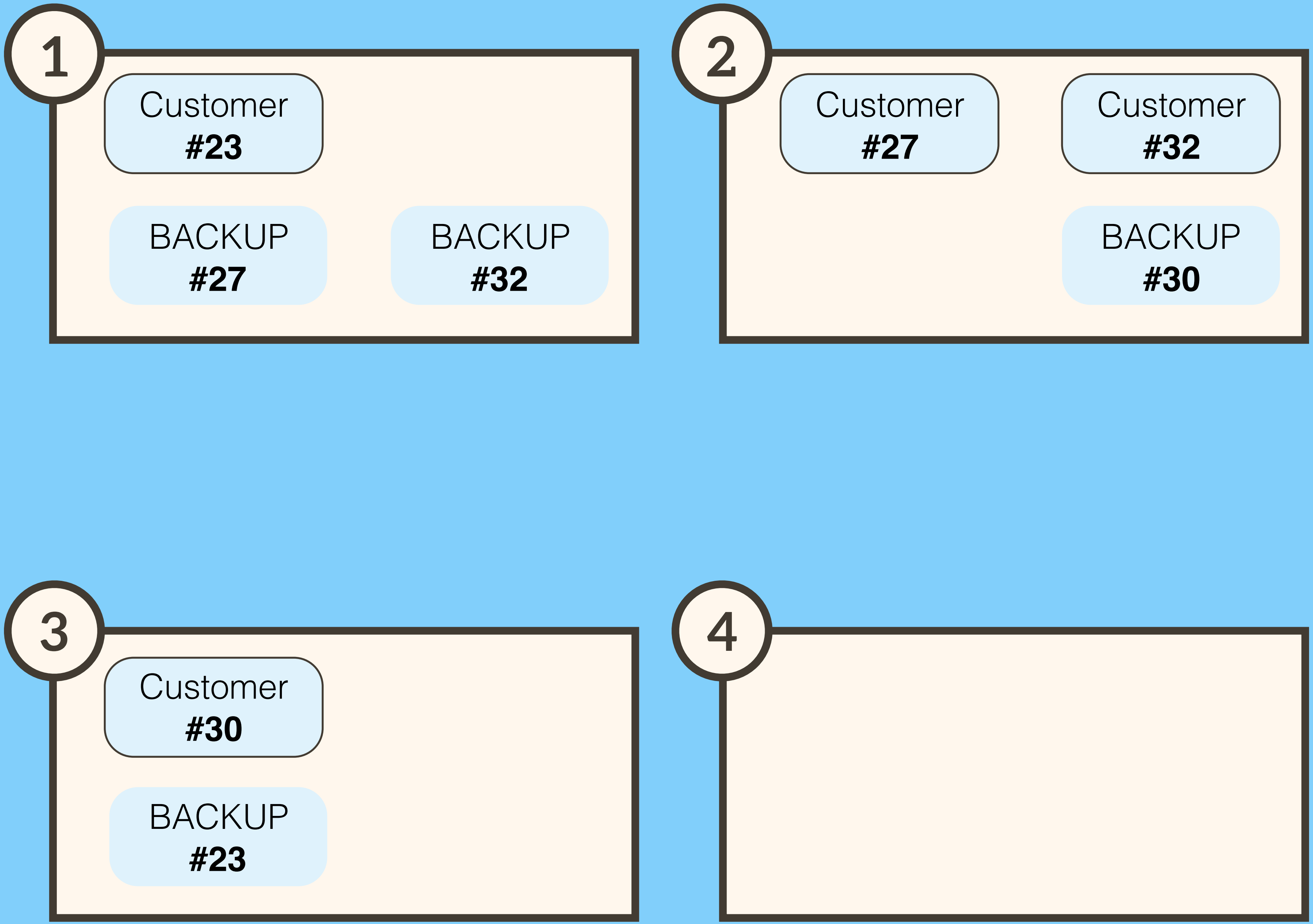


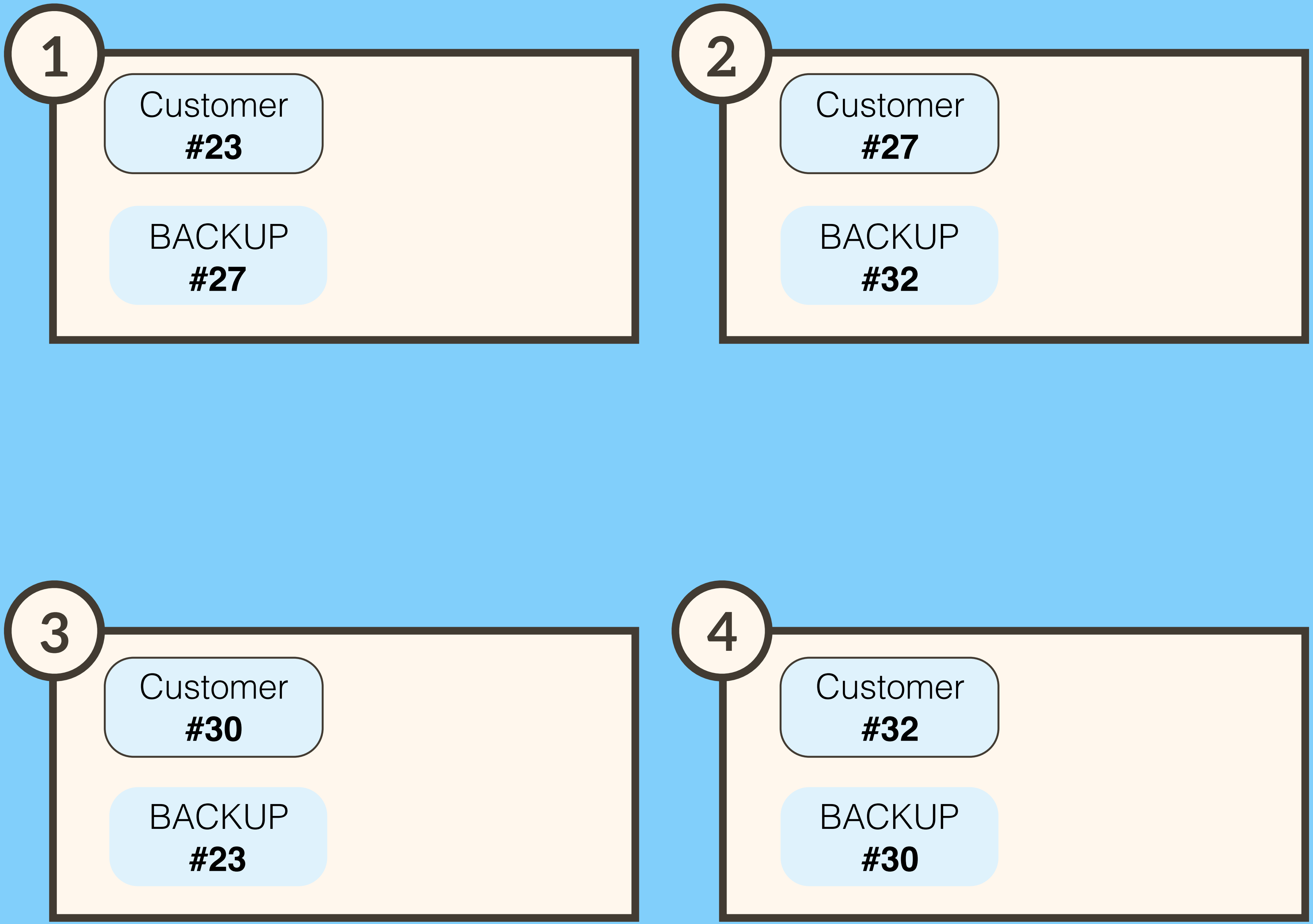

1
Customer #23
Customer #30
Customer #27
Customer #32
2



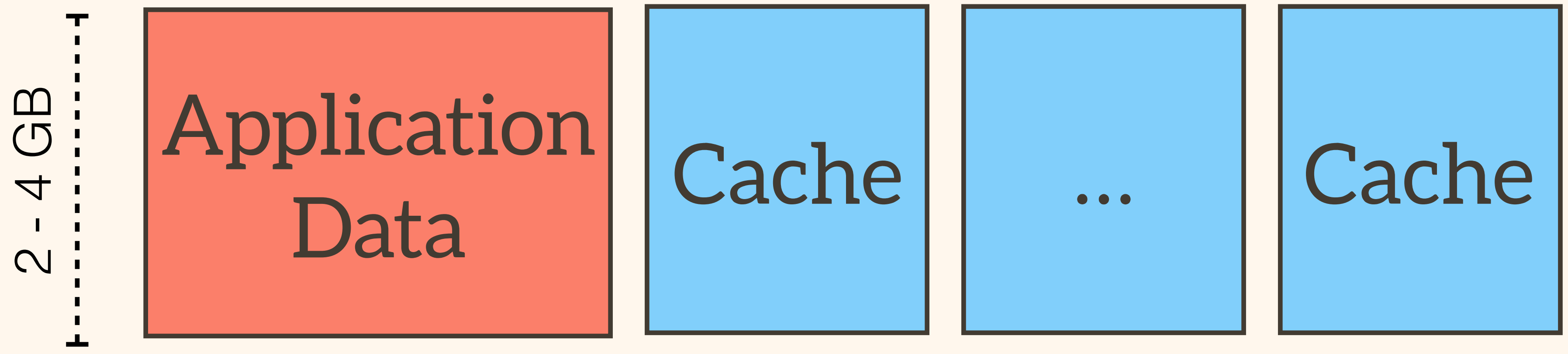
Data is being distributed and backed up

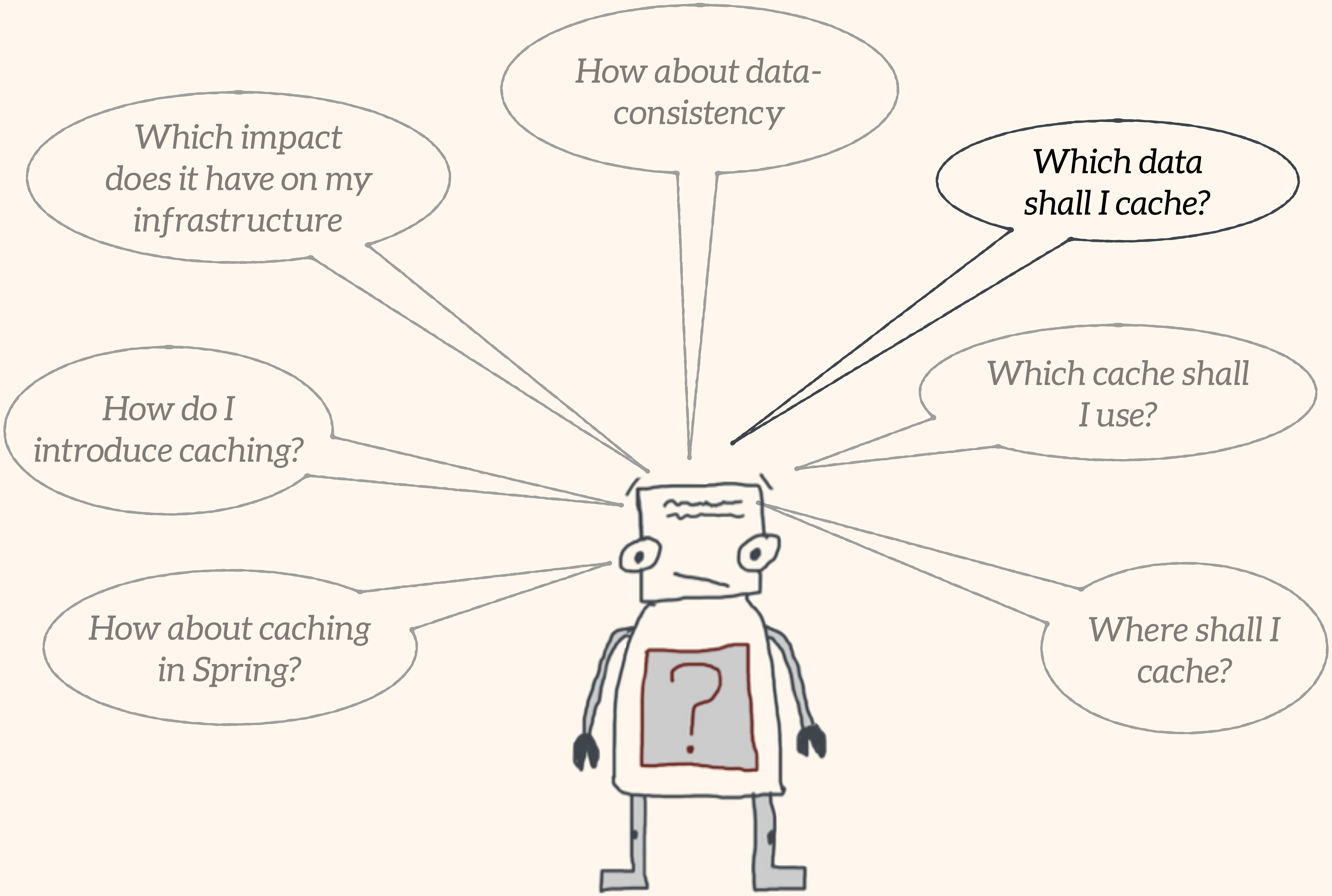

1
Customer #23
Customer #30
BACKUP #27
BACKUP #32
2
Customer #27
Customer #32
BACKUP #23
BACKUP #30
3


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

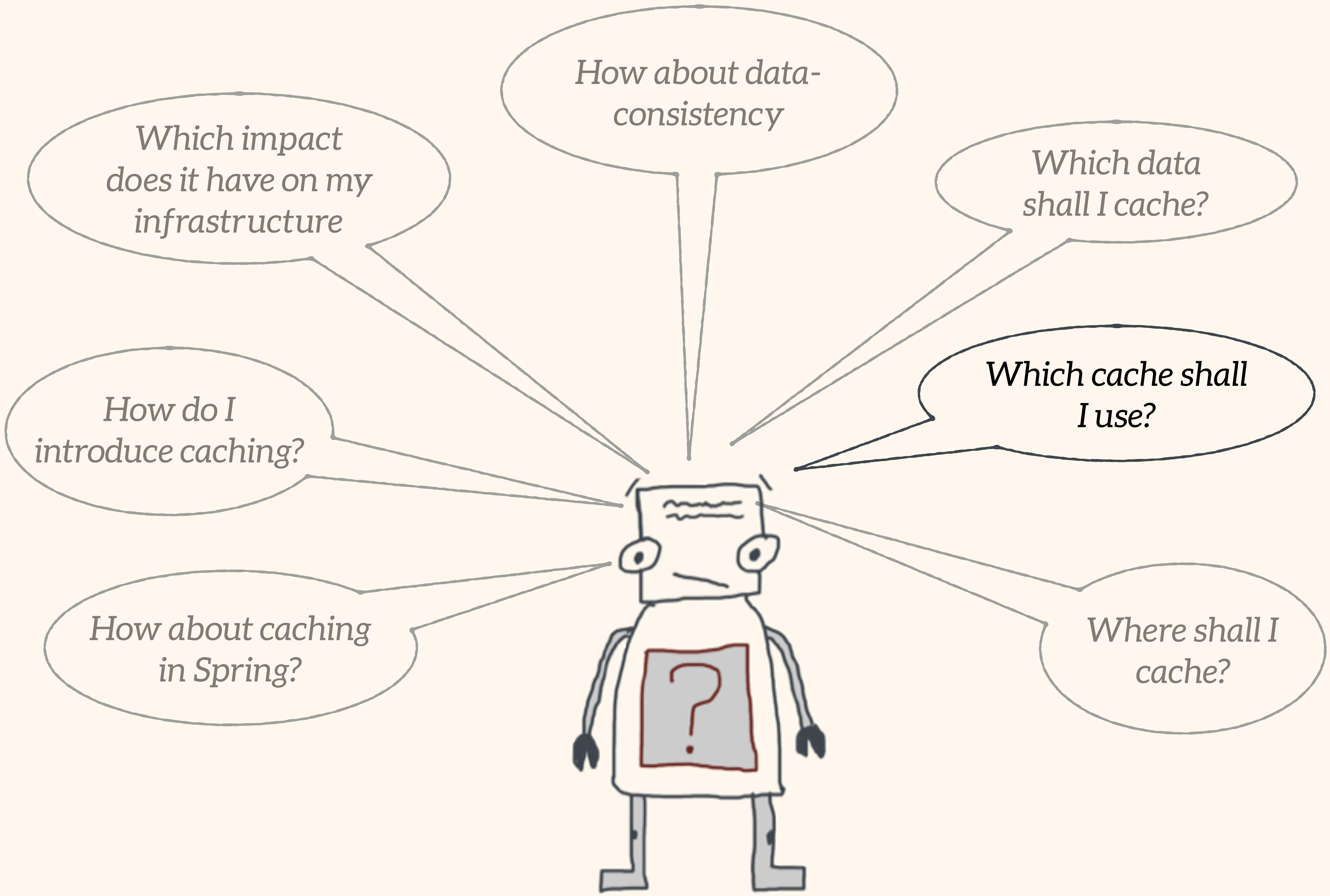



Which impact does it have on my infrastructure
How about data-consistency
Which data shall I cache?
How do I introduce caching?
Which cache shall I use?
How about caching in Spring?
Where shall I cache?

5 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


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

6

Only use existing cache implementations

NEVER

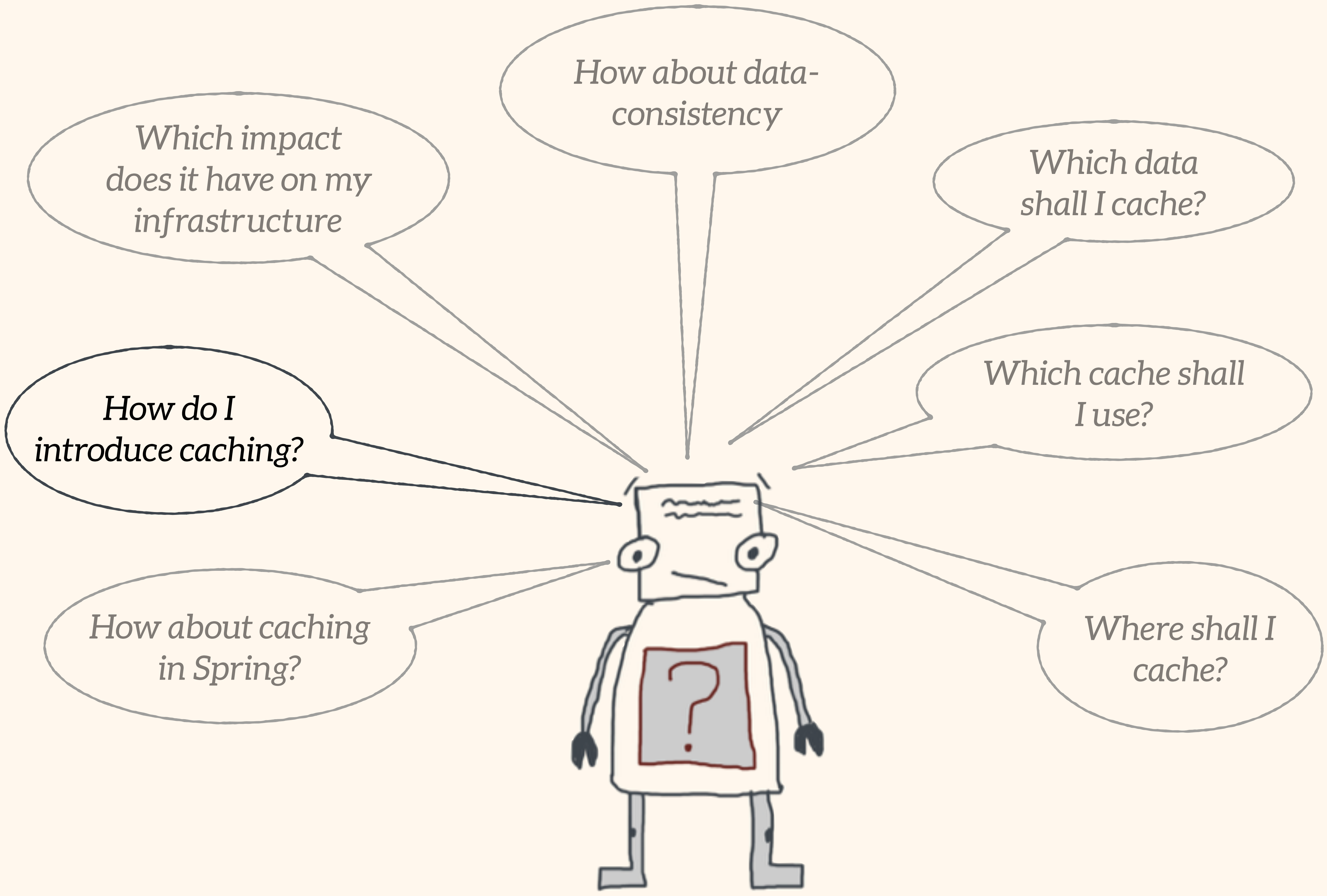
write your own cache implementation

EVER

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

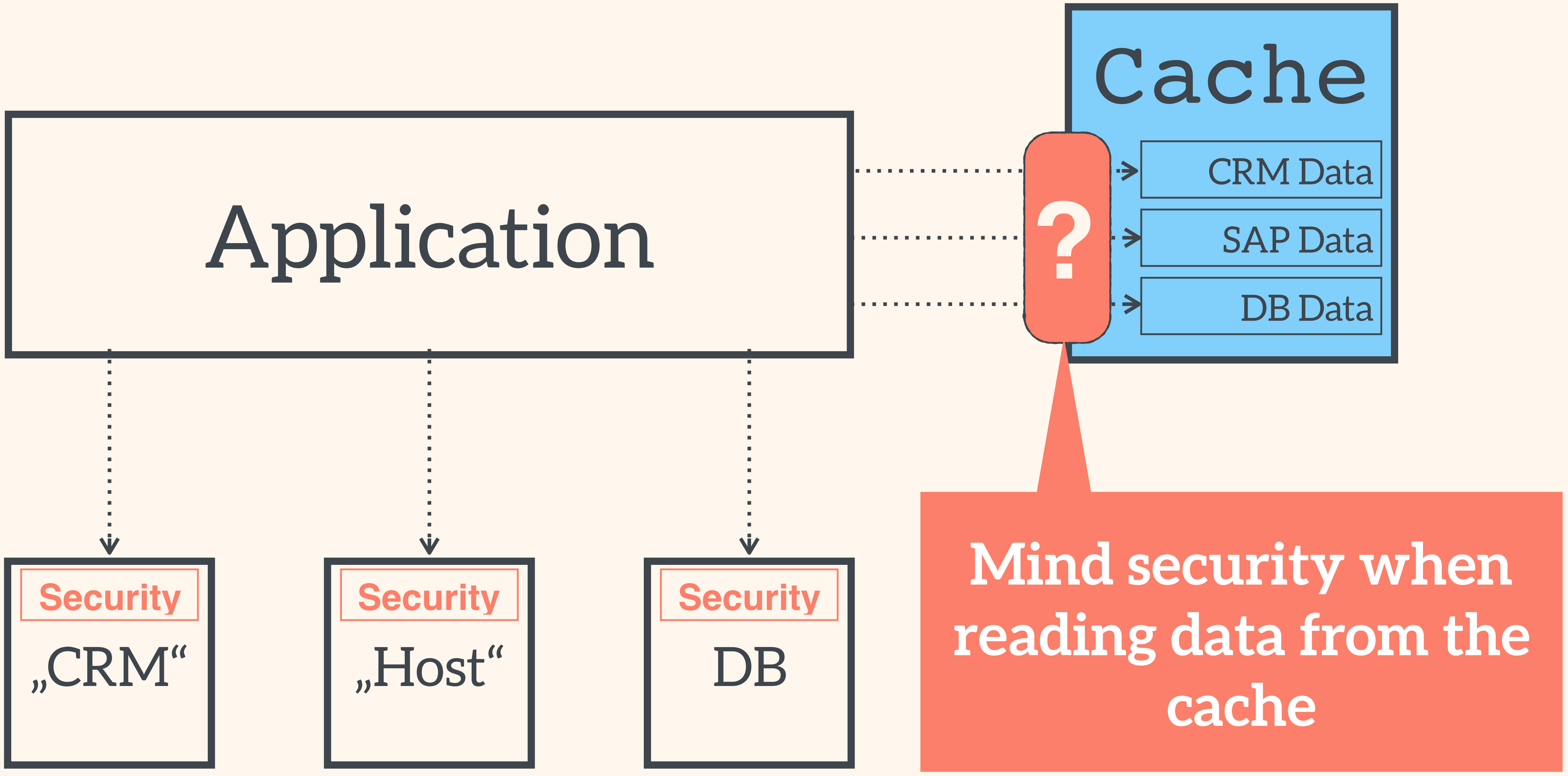
CACHE Implementations

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

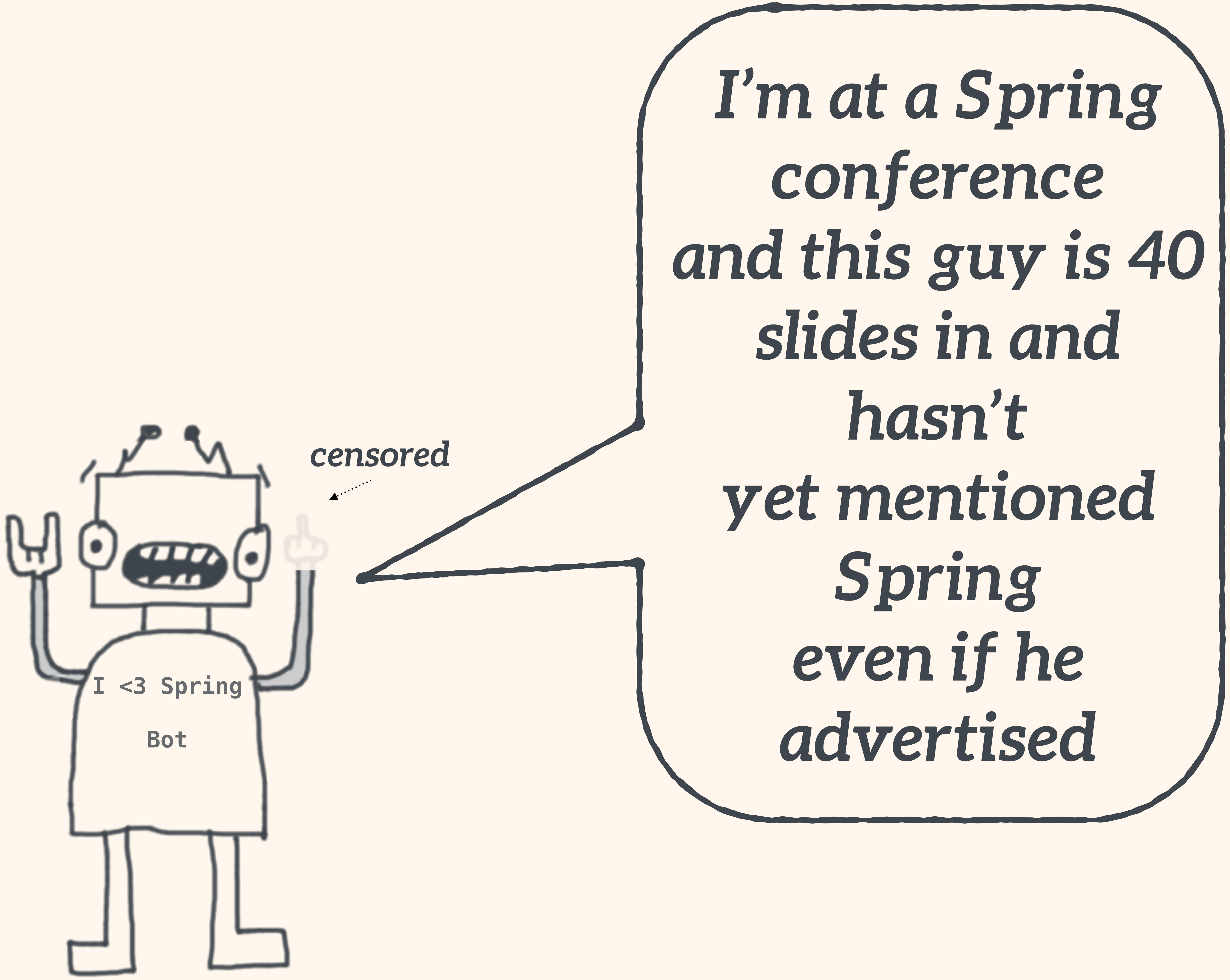

Which impact does it have on my infrastructure
How about data-consistency
Which data shall I cache?
How do I introduce caching?
Which cache shall I use?
How about caching in Spring?
Where shall I cache?
?

7

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


I'm at a Spring conference and this guy is 40 slides in and hasn't yet mentioned Spring even if he advertised
censored
I <3 Spring
Bot

8

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

```
@Configuration
@EnableCaching
public class CacheConfiguration implements CachingConfigurer {
  ...
}
```

```
<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

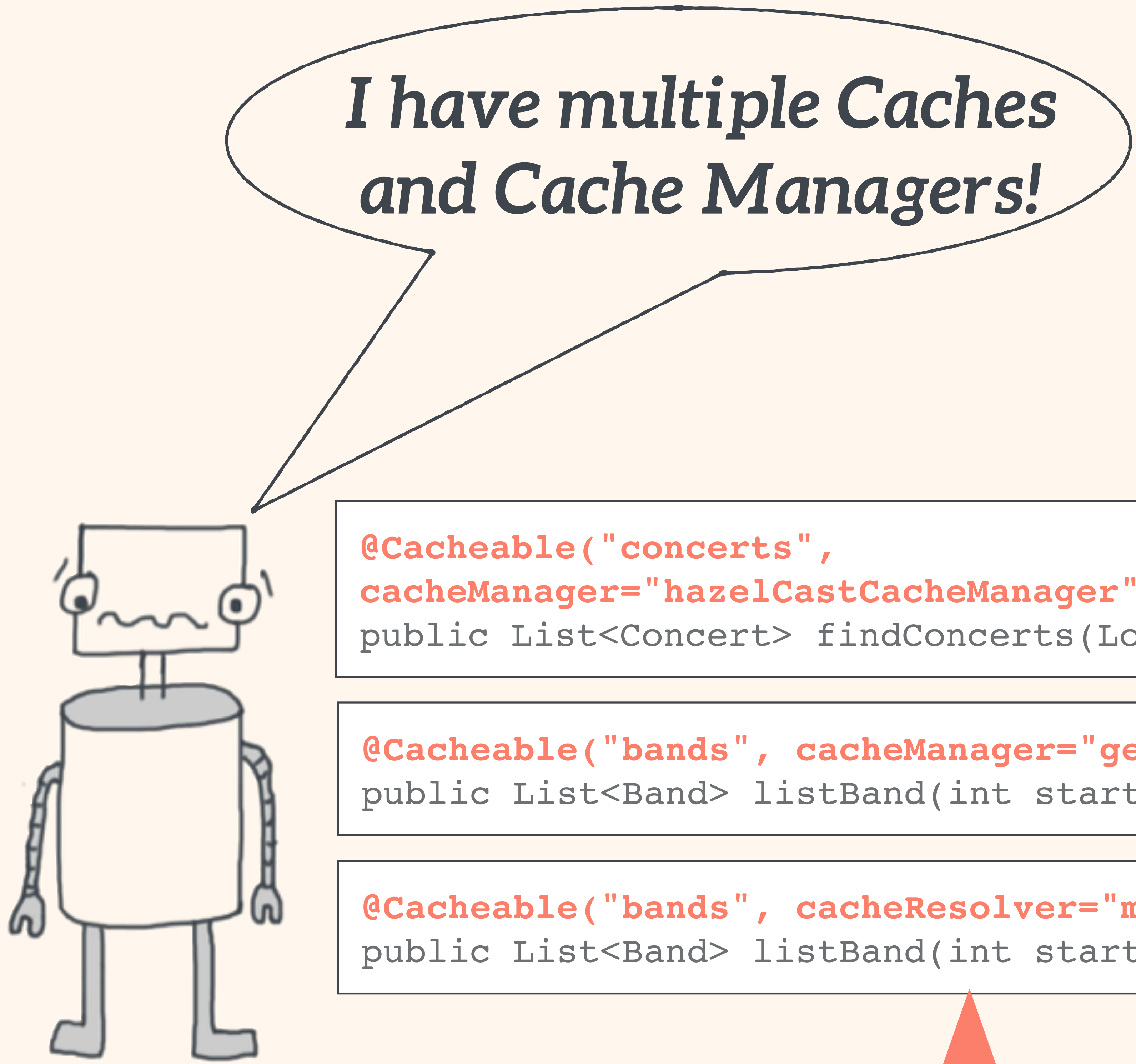
Annotation	Effect

`@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 and operating

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



```
@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

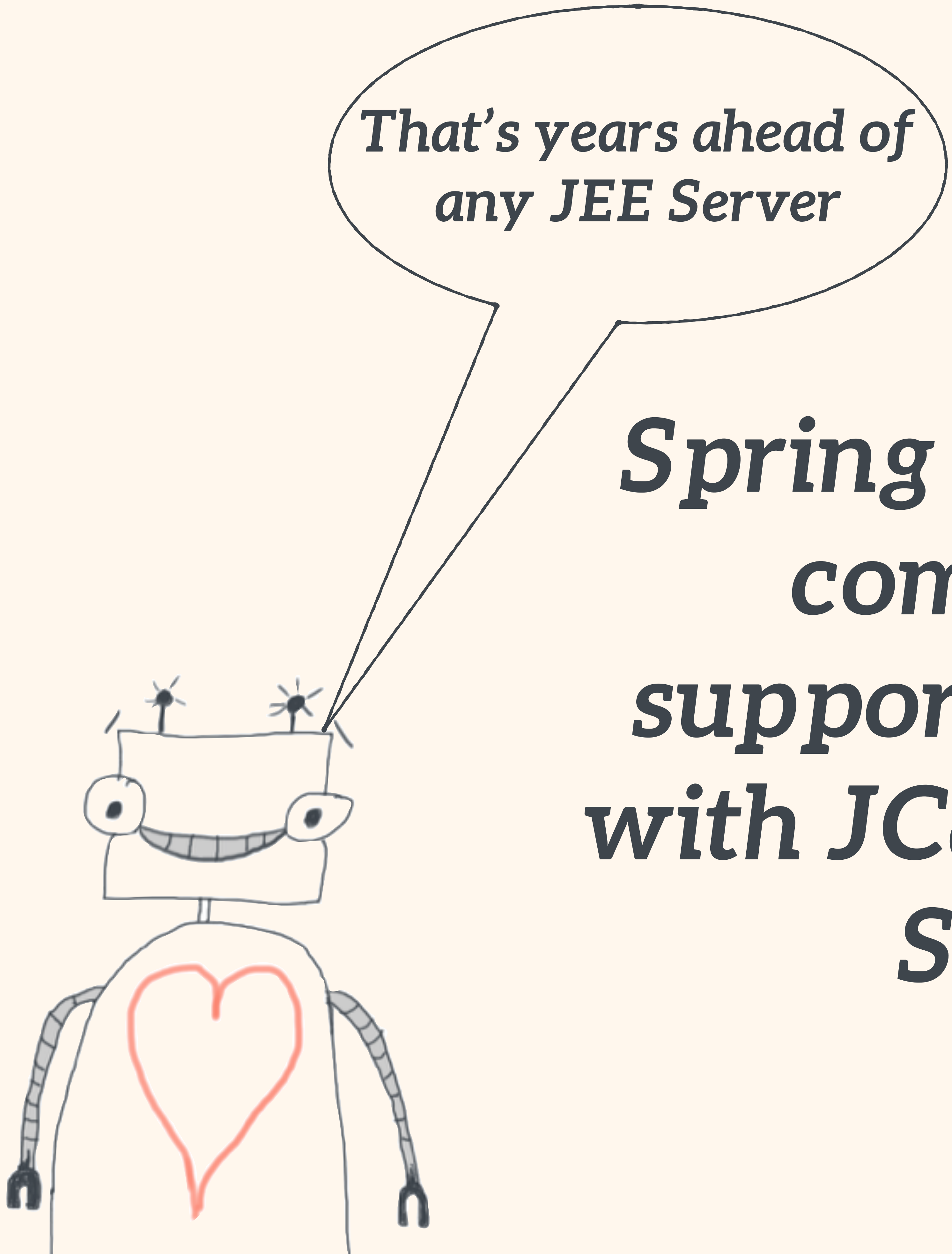
```
@Cacheable("bands", cacheResolver="myOwnCacheResolver"))
public List<Band> listBand(int start, int count)
```

```
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)
```

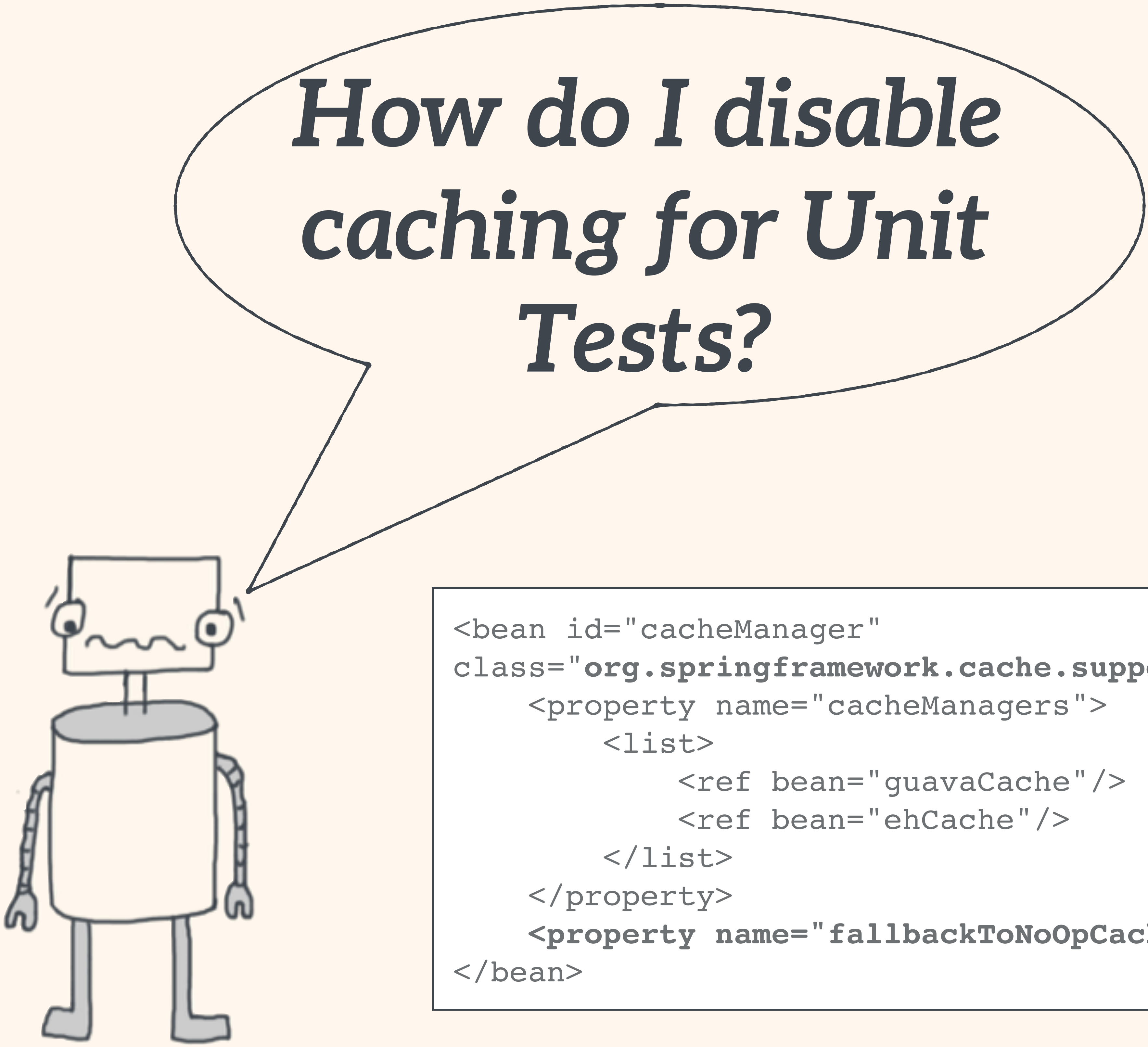


Spring 4.x is the first commerically supported container with JCache (JSR-107) Support!

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



```
<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
@bitboss

THANKS

Michael Plöd

@bitboss

https://slideshare.net/mploed