Caching 101: on the JVM and beyond

DEVOXX United Kingdom

Aurélien Broszniowski
@AurBroszniowski
Louis Jacomet
@ljacomet
Software AG / Terracotta
ehcache.org

Agenda

- Caching theory
- Caching in Java
- Caching patterns
- Scaling

DEVOXX United Kingdom

Who are we?

- Aurélien Broszniowski
  - Works at Software AG / Terracotta since 2010 (startup days!)
  - Coding since the teenage days, will probably never stop
  - OSS developer, see for example the Rainfall performance testing framework
- Louis Jacomet
  - Principal Software Engineer at Software AG / Terracotta since 2013
  - Developper close to his forties who did not dodge all things management
  - Interests range from Concurrency to API design, with learning new things as the driving factor

DEVOXX United Kingdom

Who are you?

- Who knows nothing about caching?
- Who already uses caching in production?
- Who had caching related problems in production?
- Ehcache anyone?

DEVOXX United Kingdom

There are only two hard things in
Computer Science:
cache invalidation and naming things.

-- Phil Karlton

DEVOXX United Kingdom

DEVOXX United Kingdom

Caching theory

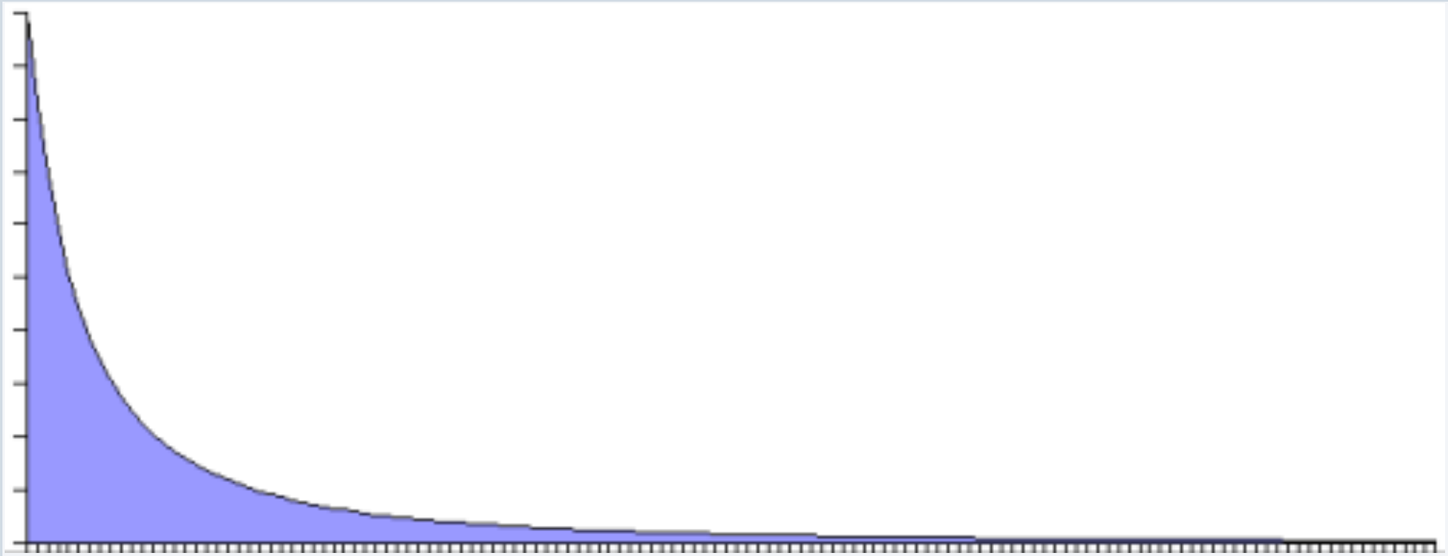
What is a cache?

- Data structure holding a ***temporary*** copy of some data
- Trade off between ***higher memory*** usage for ***reduced latency***
- Targets:
  - Data which is reused
  - Data which is expensive to compute or retrieve

Locality of reference

- Same values or related ones are frequently accessed
- One form of predictable behaviour
  - heavily used in CPUs for branch prediction
- Principle applied at different levels: CPUs, network and OS
- Ends up being a desirable feature in applications
  - Application closer to hardware behaviour

The long tail



- Coined by Chris Anderson
- Application of a Power Law probability distribution
  - Pareto Law or 80:20 rule

Terminology

- hit: when the cache returns a value
- miss: when the cache does not have a value
- cold: when the cache is empty
- warm-up: phase during which the cache fills up
- warm: when a cache is filled and helps the most

Possible usages

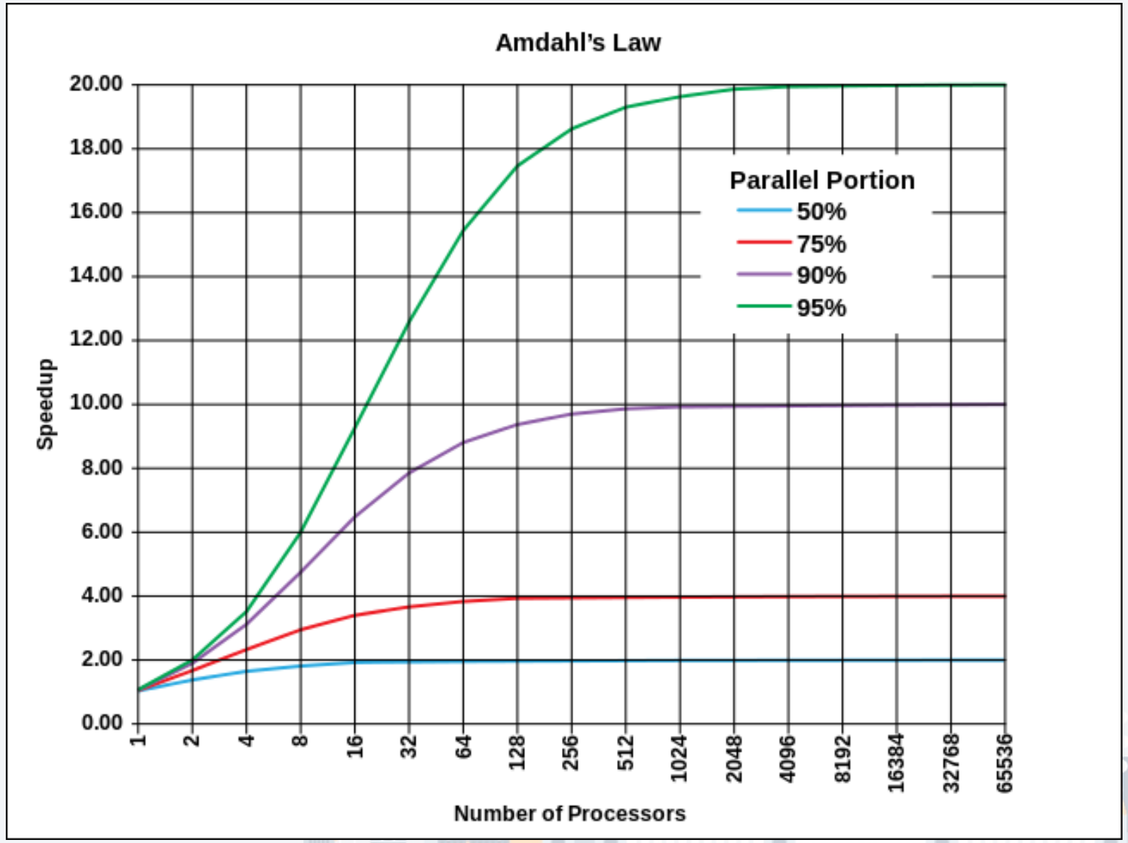
- CPU bound applications
  - Normal speedup through algorithm improvement or parallelisation
  - Cache can help by storing computation results
- I/O bound applications
  - Normal speedup through disk or network upgrades
  - Cache can help by storing data locally

Amdhal's law

- Parallelisation

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

  - P: parallel proportion
  - N: processor count

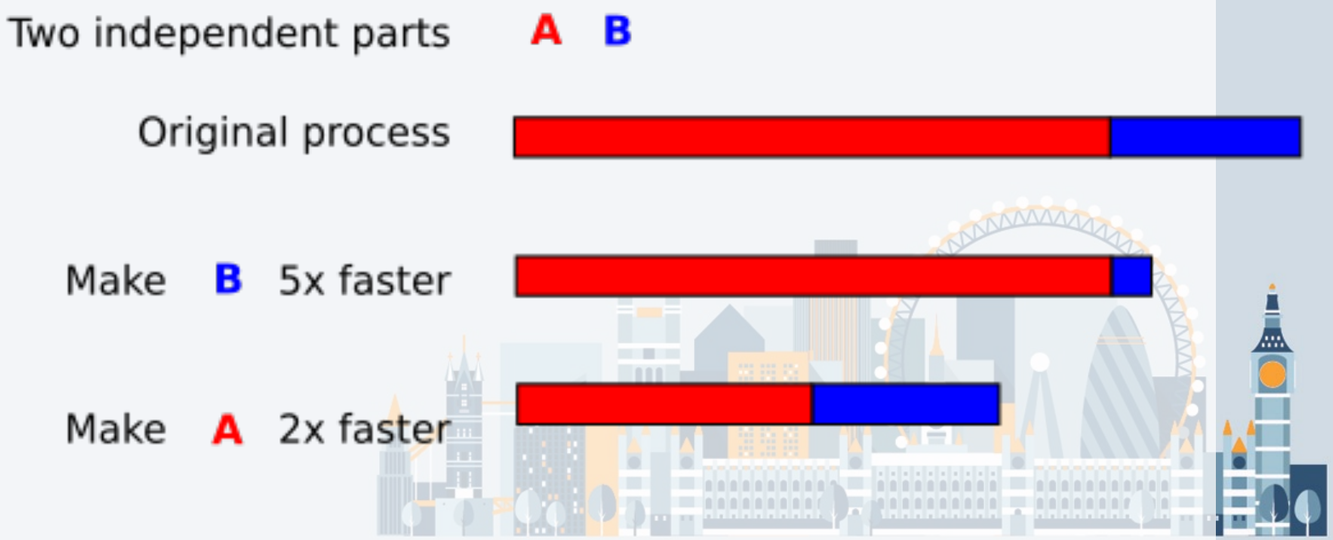


Amdhal's law

- Sequential

$$max\ speedup \leq \frac{p}{1 + f * (p - 1)}$$

- p: times a part was sped up
- f: fraction of time improved



Cache coherence

- P write A to X, P read X => returns A when no writes happened in between, must be valid for single processor too
- P2 writes A to X, P1 reads X => returns A if “enough time” has elapsed
- P1 writes A to X, P2 writes B to X => no one can ever see B then A at X

Desired cache features

- Capacity control / eviction
- Data freshness / expiry
- Data consistency / invalidation
- Fault tolerant

DEVOXX United Kingdom

Caching in Java

BlazingCache

Google Guava

And more …

DEVOXX United Kingdom

JSR-107 aka JCache aka javax.cache

- JSR submitted in 2001
  - Early draft in 2012 co-lead by Coherence and Ehcache representatives
- Specification finalised in March 2014
- Specifies API and semantics for **temporary**, **in-memory caching** of Java objects, including object creation, shared access, spooling, invalidation, and consistency across JVM's

javax.cache features

- `CacheManager` managing … `Cache`!
- Expiry
- Integration (`CacheLoader` and `CacheWriter`)
- Cache listeners
- Entry processors
- Annotations
- Statistics and MBeans

DEVOXX United Kingdom

javax.cache API

`CacheManager` repository, identified by `URI` and `ClassLoader`

```
CachingProvider provider = Caching.getCachingProvider();

CacheManager cacheManager = provider.getCacheManager();

Cache<Long, String> myCache = cacheManager.getCache("myCache", Long.class, String.class);
```

Named `Cache` repository, handles their lifecycle

`ConcurrentMap<K, V>` sibling, major interaction point for your logic

javax.cache integration

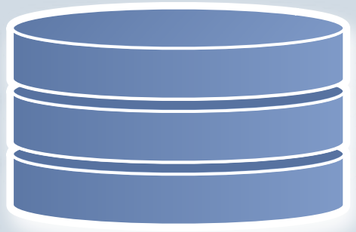
- Spring caching abstraction since version 4.1
- Hibernate 5.2 has JCache integration for 2nd level cache
  - Does not include transactional support
- More to come???

Caching guidelines

- Immutable keys
- Favour immutable values
  - store *by-value* or *by-ref* semantics
- Understand what you cache
  - Think hibernate entities with lazy attributes
- Always account for a *miss* due to *expiry* or *eviction*

DEVOXX United Kingdom

Caching patterns

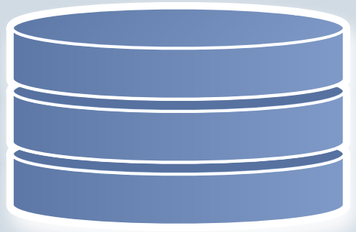


Database

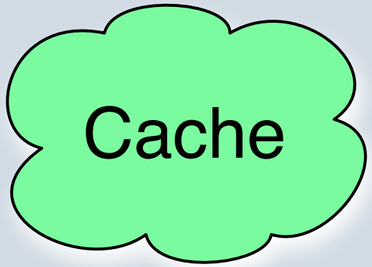
Cache

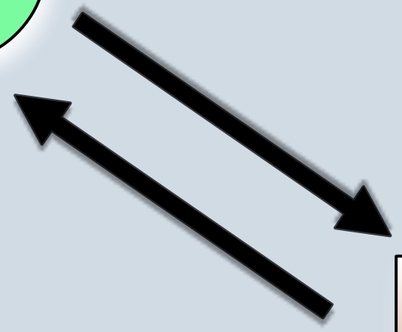
Application

Cache aside (miss)



Database





Application

Cache aside (hit)

Demo

DEVOXX United Kingdom

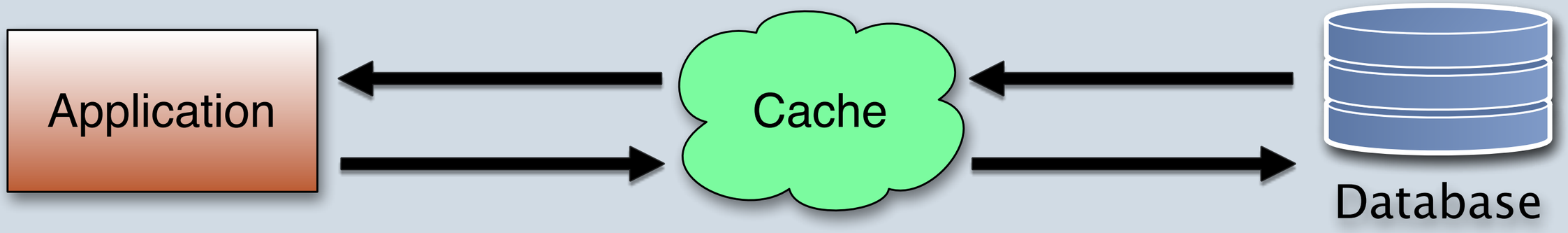
Hibernate 2nd level cache

- Stores ***dehydrated*** entities
- 4 strategies
  - ReadOnly
  - NonStrictReadWrite
  - ReadWrite
  - Transactional

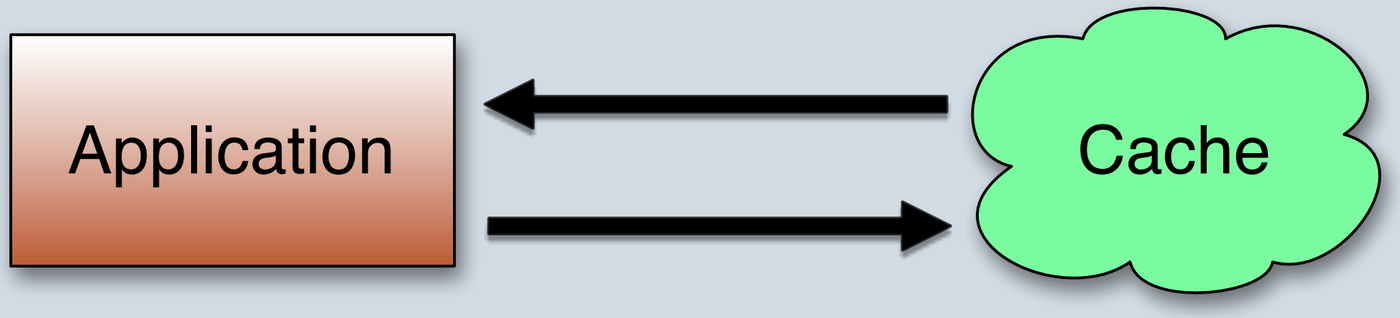
Summary

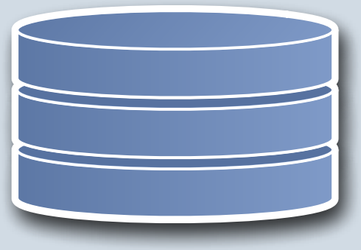
Cache aside conclusions

- Solution most seen out there
  - Spring, Play, Grails, …
  - Most often based on annotations
- Tricky to get the concurrency and / or atomicity right
  - Especially when rolling you own
- Suffers from thundering herd: multiple invocations when cache is cold



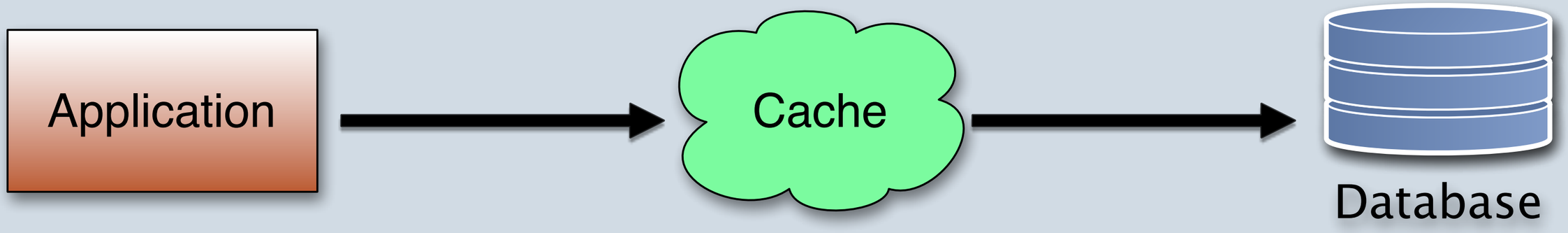
Cache through (miss)





Database

Cache through (hit)



DEVOXX United Kingdom

Cache through (write)

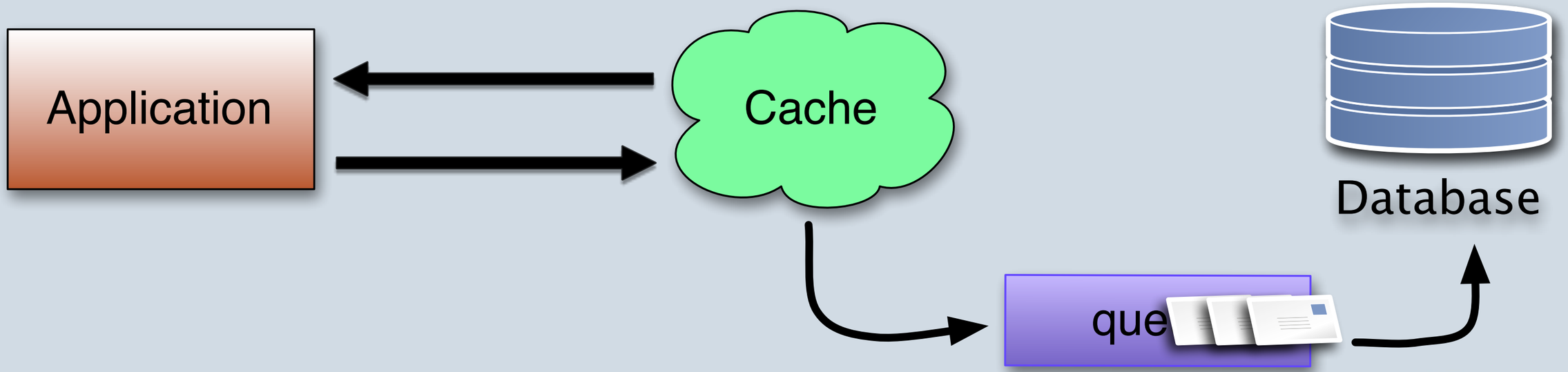
Demo

DEVOXX United Kingdom

Summary

Cache through conclusions

- Requires different abstraction / modelling
  - Viewing the system of record through the cache may not be easy
- Provides better guarantees and consistency as invalidation is no longer required
  - Outside of external system of record modifications
- Cost of writing is paid by application thread putting in the cache



Write behind

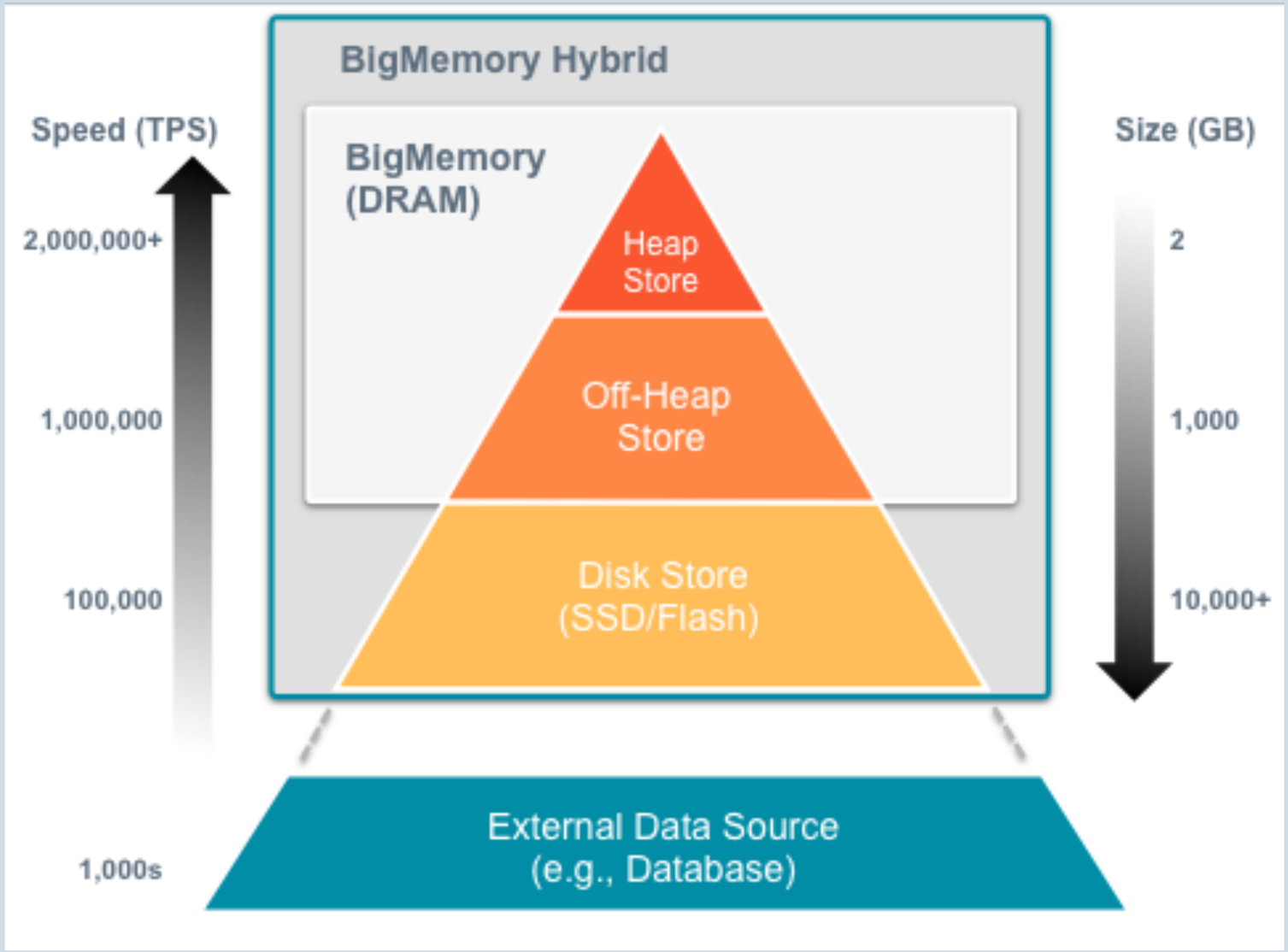
Demo

DEVOXX United Kingdom

Summary

Write behind conclusions

- Impacts your domain modelling
- Scales out your writes
  - Can benefit from batching and coalescing
- Durability function of the queue persistence
- Idempotent operations are important in a distributed setup where failures do happen



DEVOXX United Kingdom

Scaling

Moving off heap

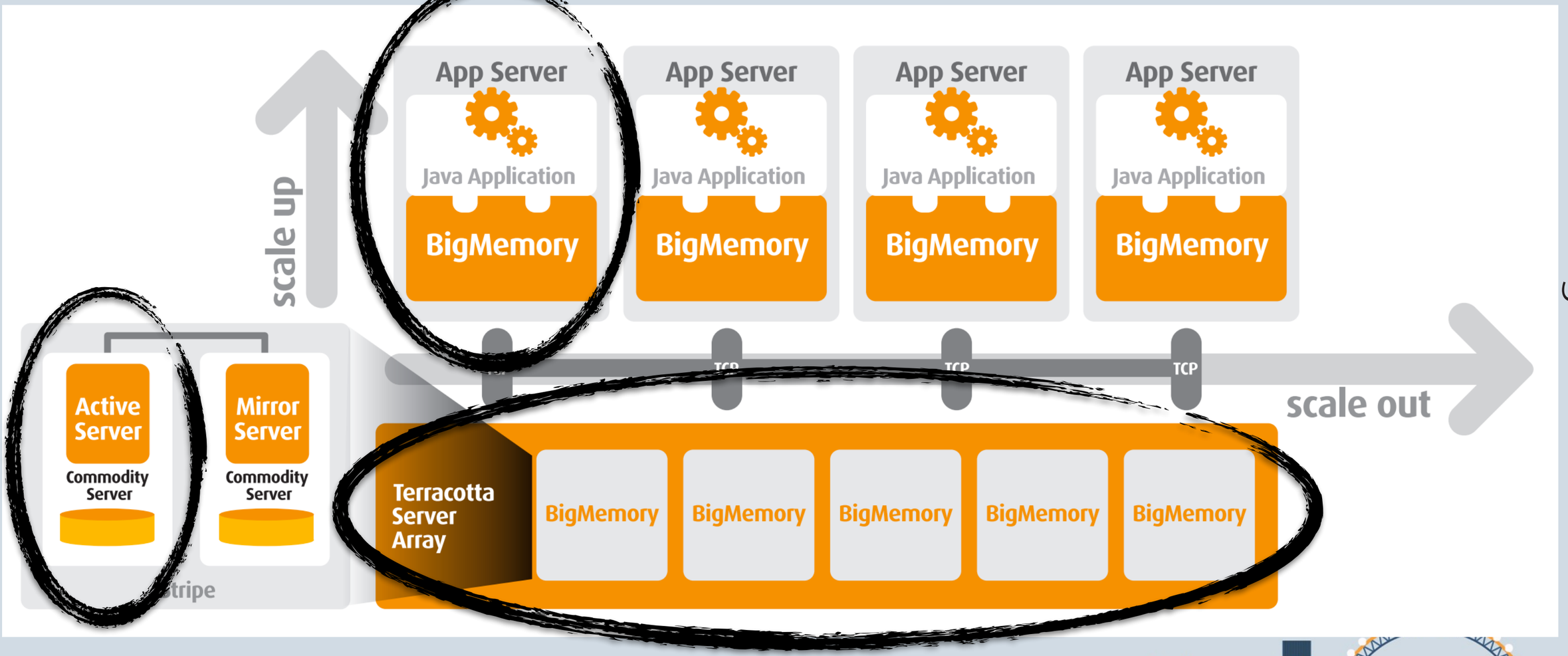
- JVMs have issues with large heaps due to garbage collection
- off heap is a nice alternative for data having a well known lifecycle
  - Perfect match for cache data
- Performance hit due to required serialization in place of object reference

Clustering

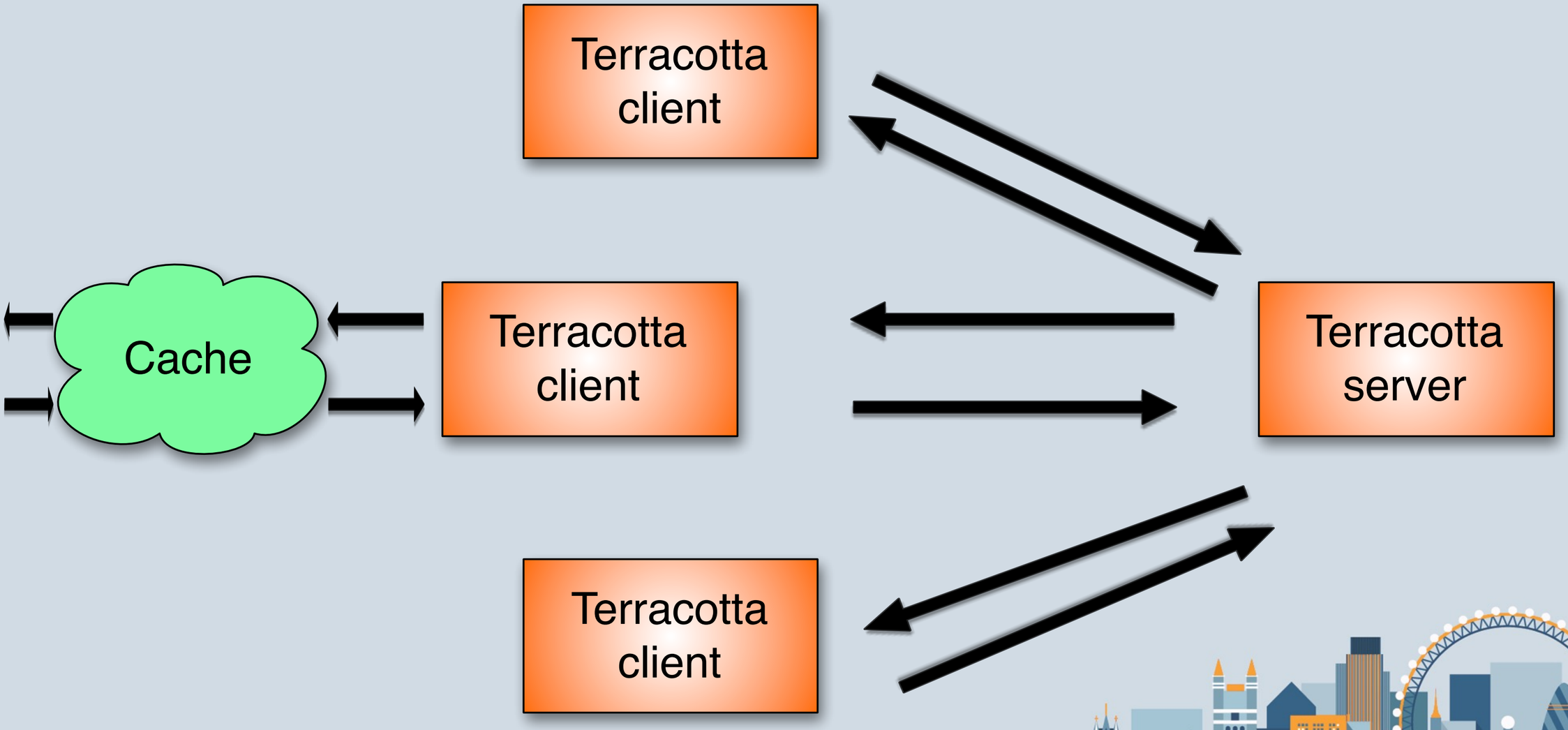
- Cache shared between multiple machines
- Different topologies
  - peer to peer
  - client - server
    - with or without client caching
- Consistency becomes a much harder problem

Probabilistically Bounded Staleness

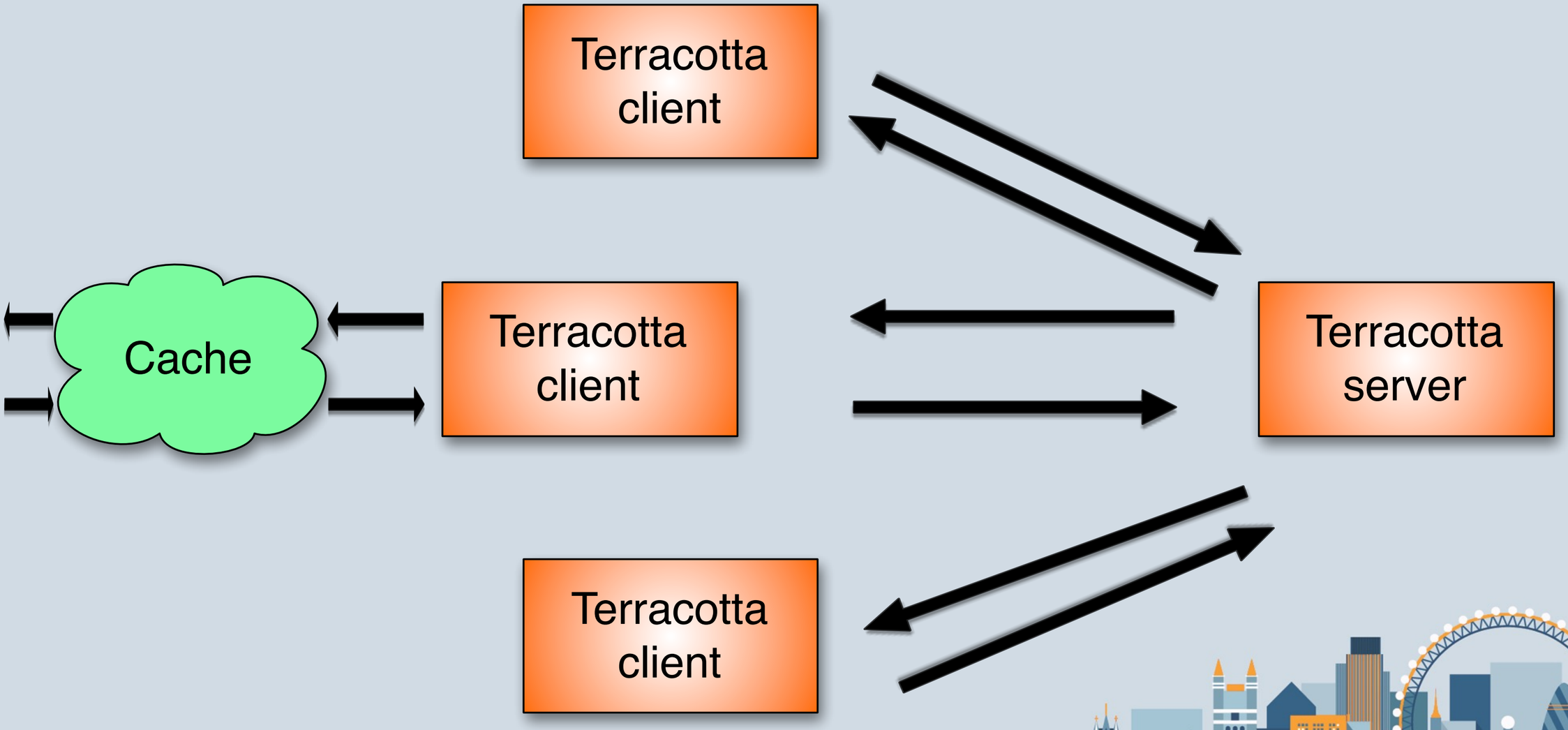
- How **long** for a write to be **readable** by **all**?
- How many **old versions** are still around?
- Depending on
  - network delay
  - node processing time
  - .. delayed replication (i.e. batching)



Terracotta clustering



Terracotta consistency: strong



Terracotta consistency: eventual

Summary

Conclusion

- You know it all now!
  - Understand your needs and what caching could bring
- Consider it early on
  - Patterns and topologies can have an impact

DEVOXX United Kingdom

DEVOXX United Kingdom

Q&A