Scaling Your Cache & Caching at Scale

TERRACOTTA

Alex Miller
@puredanger

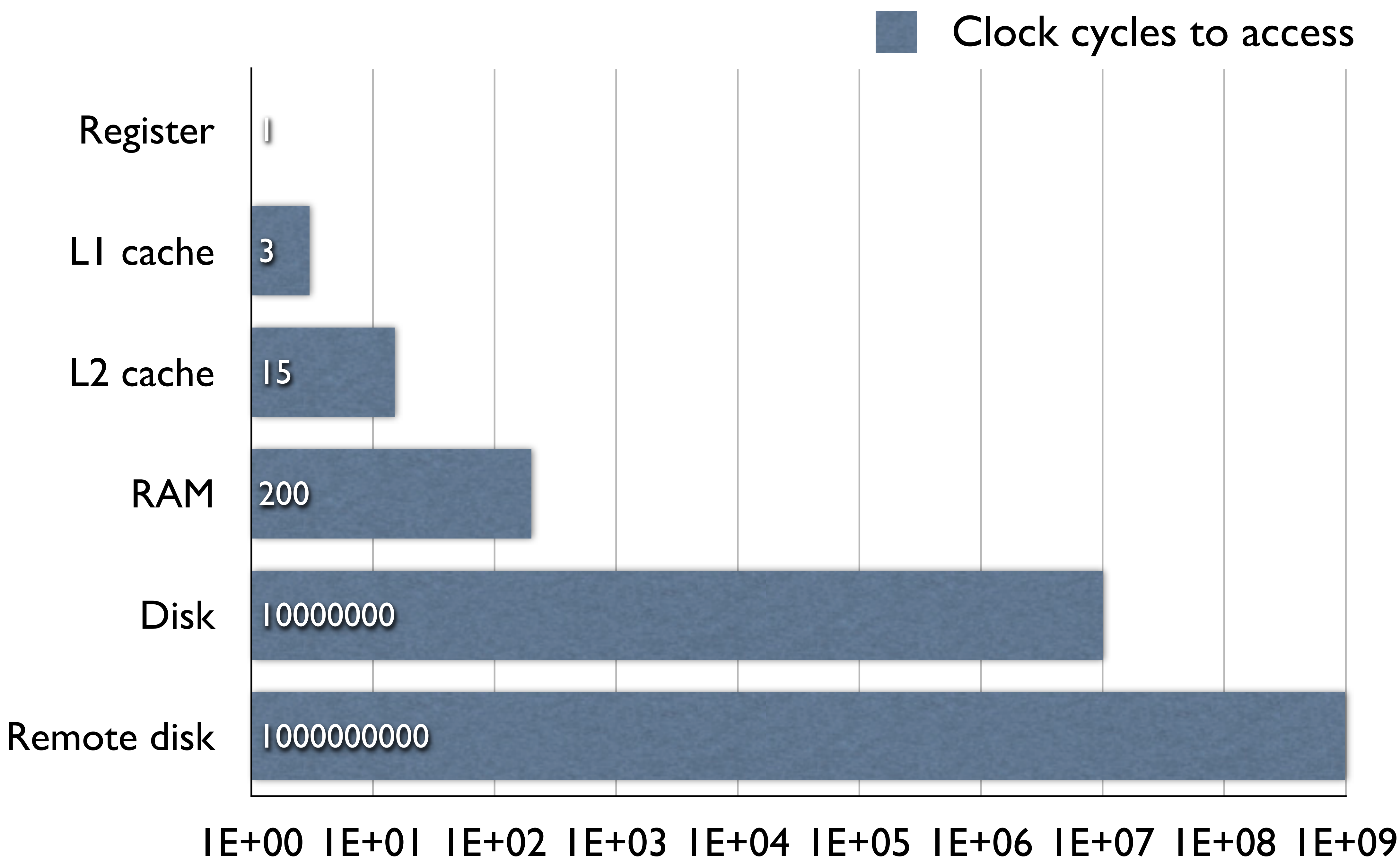
Mission

- Why does caching work?
- What's hard about caching?
- How do we make choices as we design a caching architecture?
- How do we test a cache for performance?

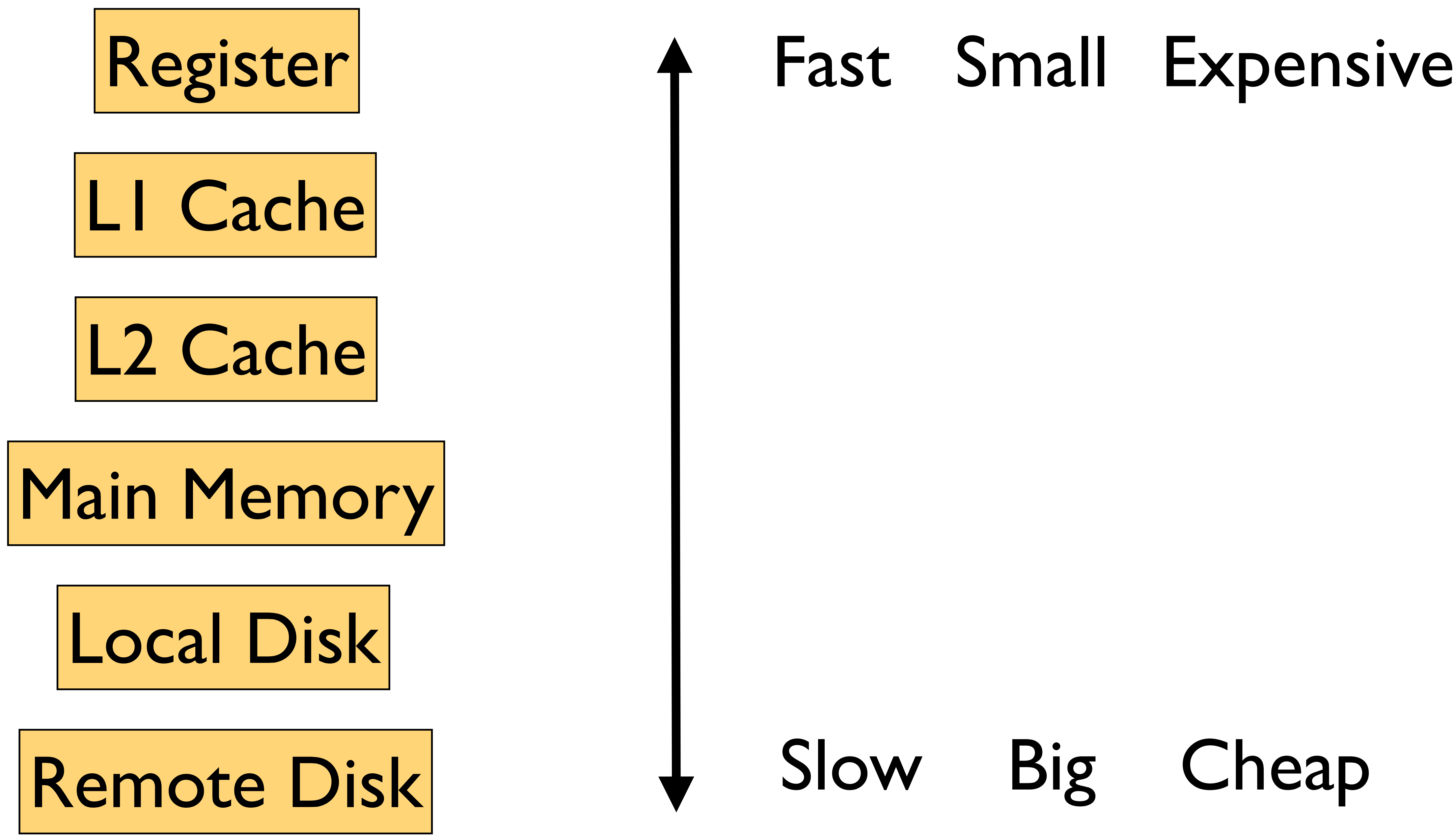
What is caching?


Lots of data

Memory Hierarchy

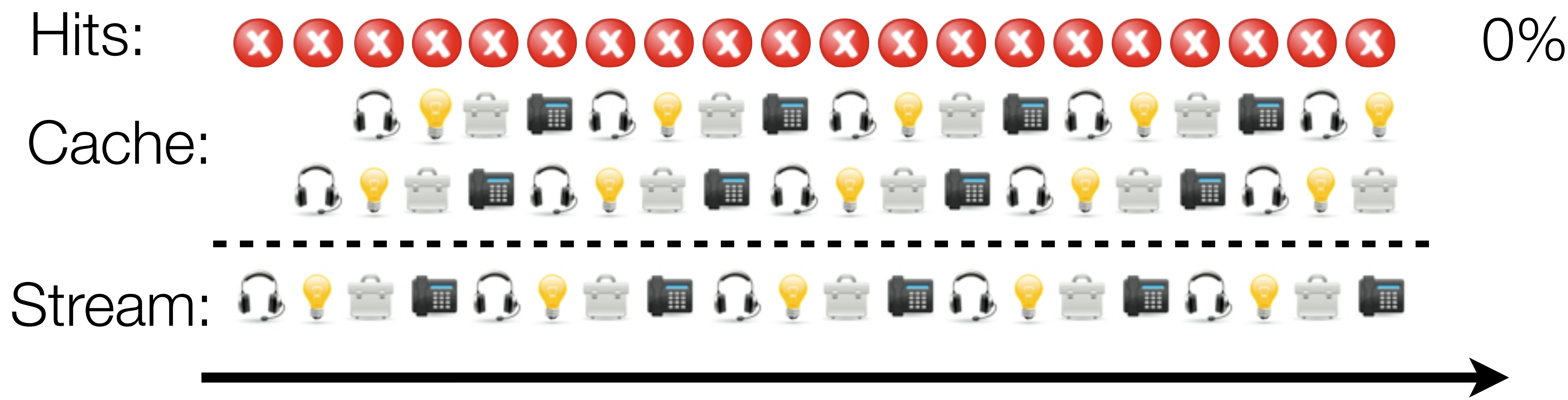


Facts of Life

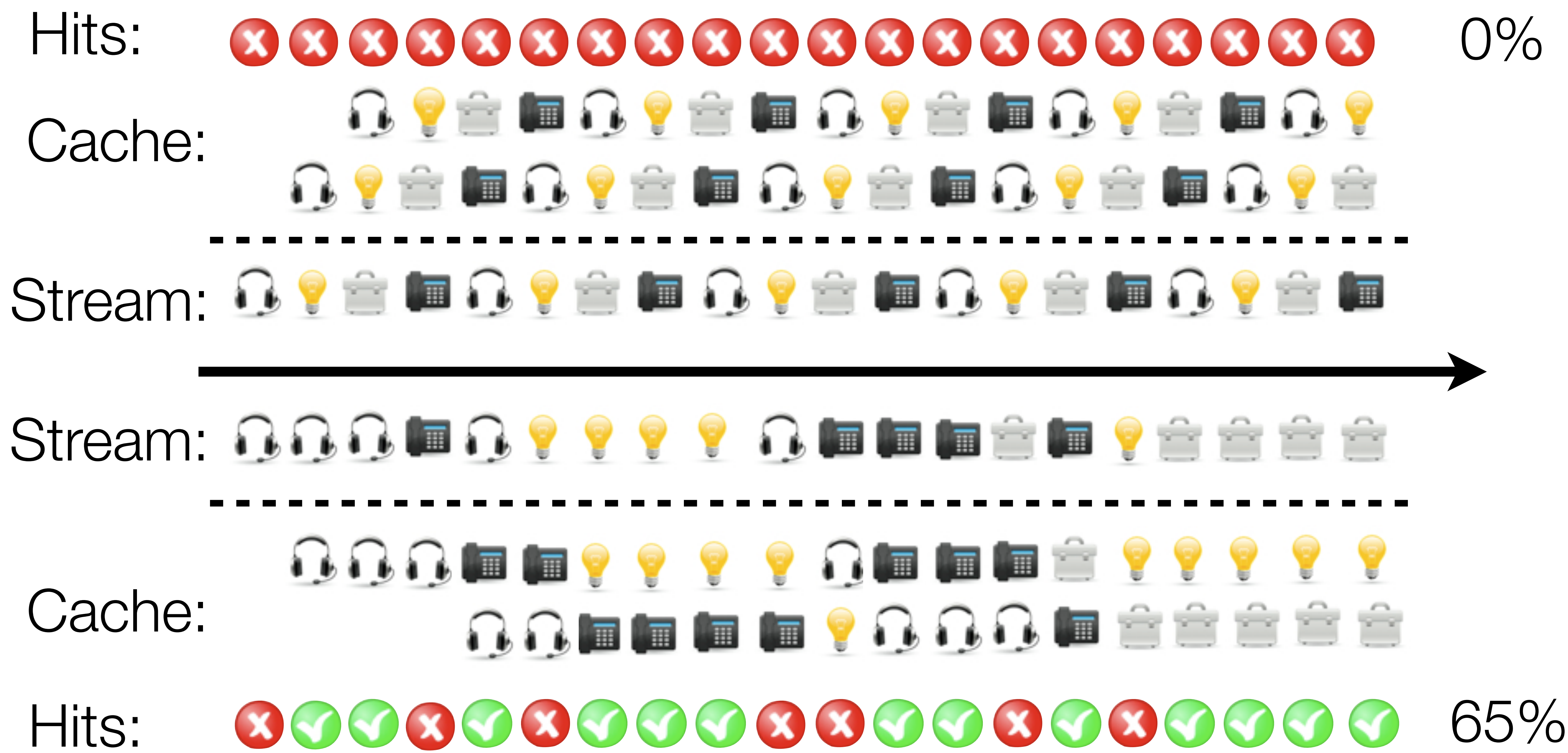


Caching to the rescue!

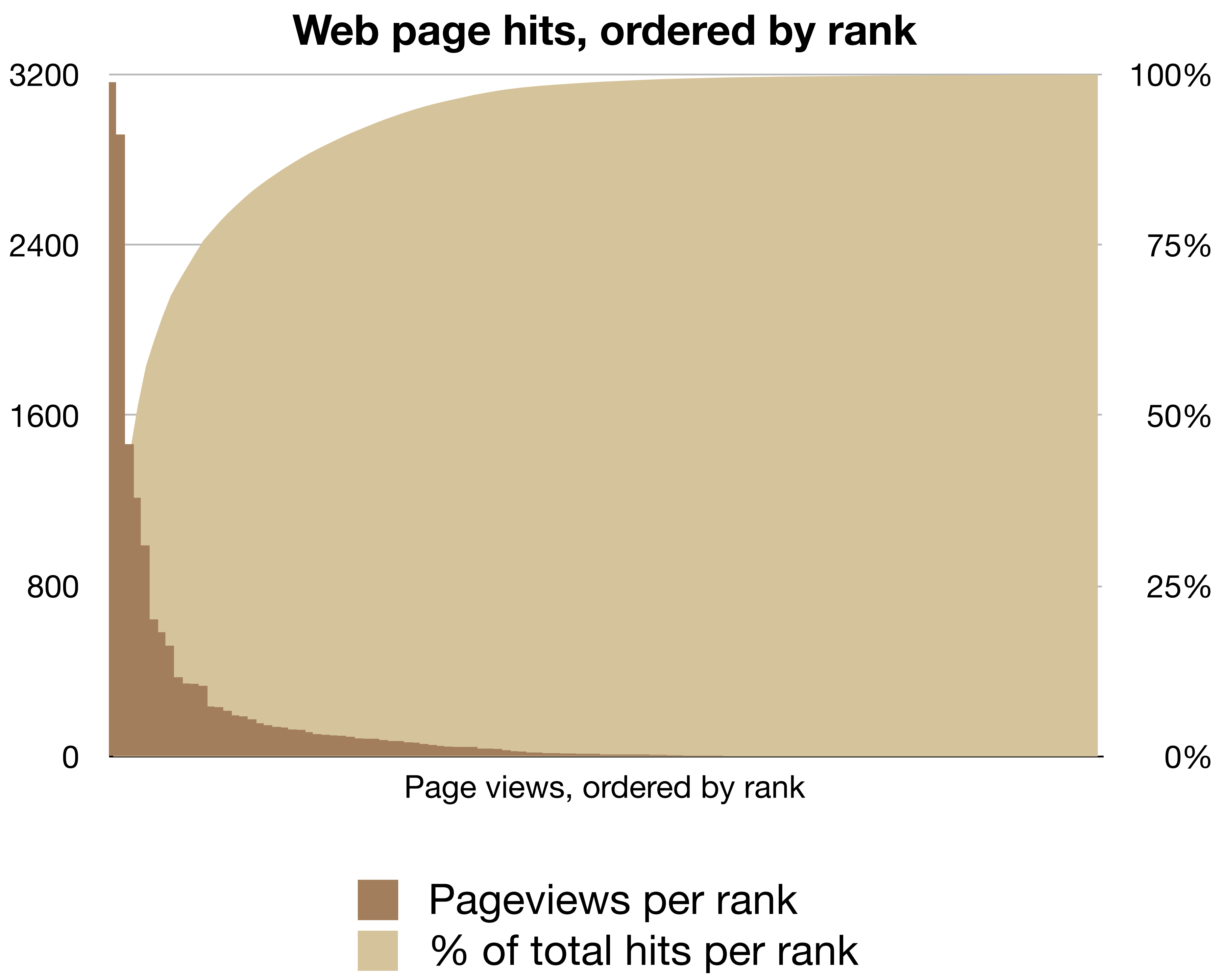
Temporal Locality



Temporal Locality



Non-uniform distribution



Temporal locality

+

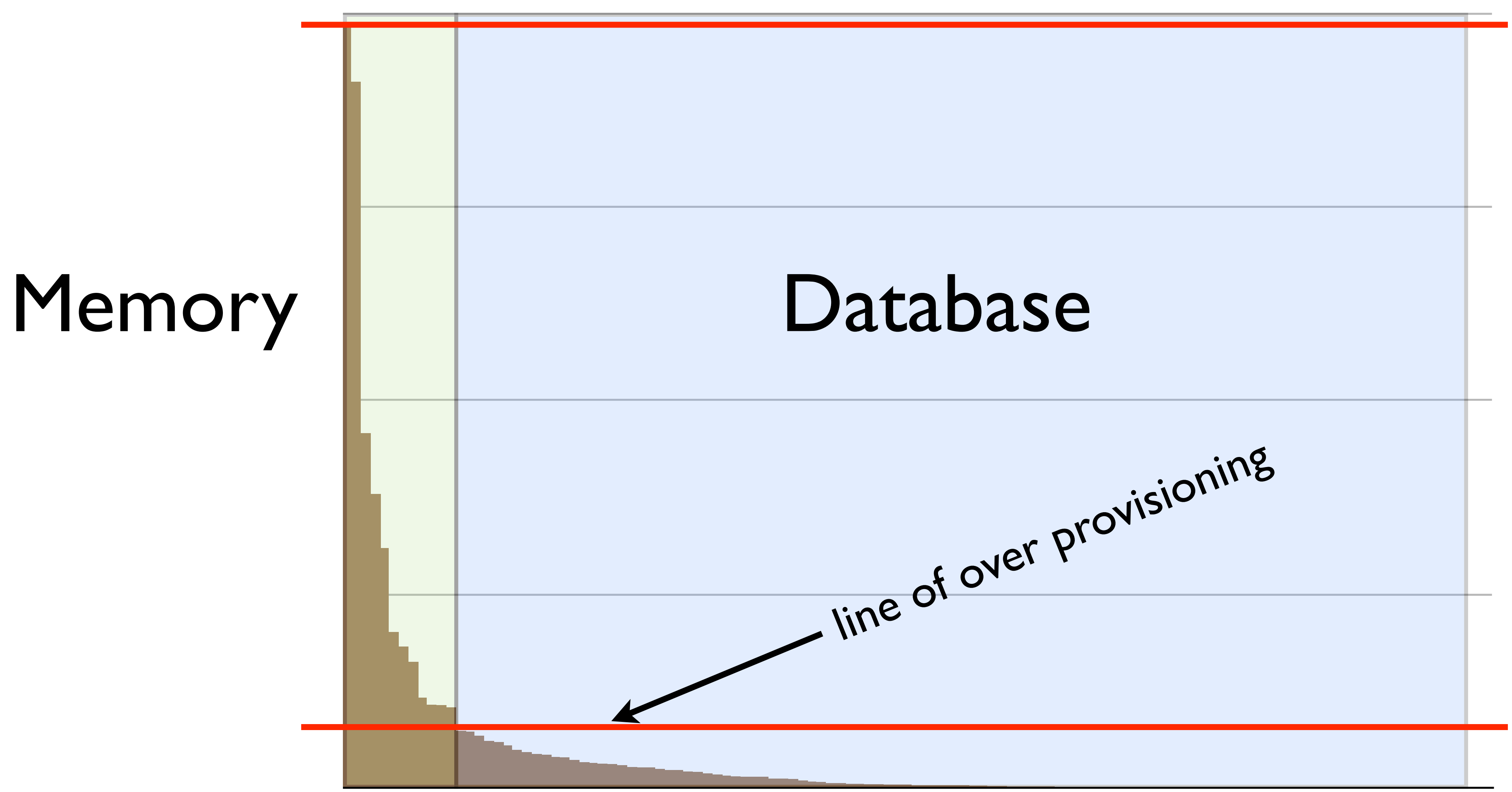
Non-uniform distribution

17000 pageviews
assume avg load = **250 ms**

cache **17 pages** / 80% of views
cached page load = 10 ms
new avg load = **58 ms**

trade memory for latency reduction

The hidden benefit: reduces database load



A brief aside...

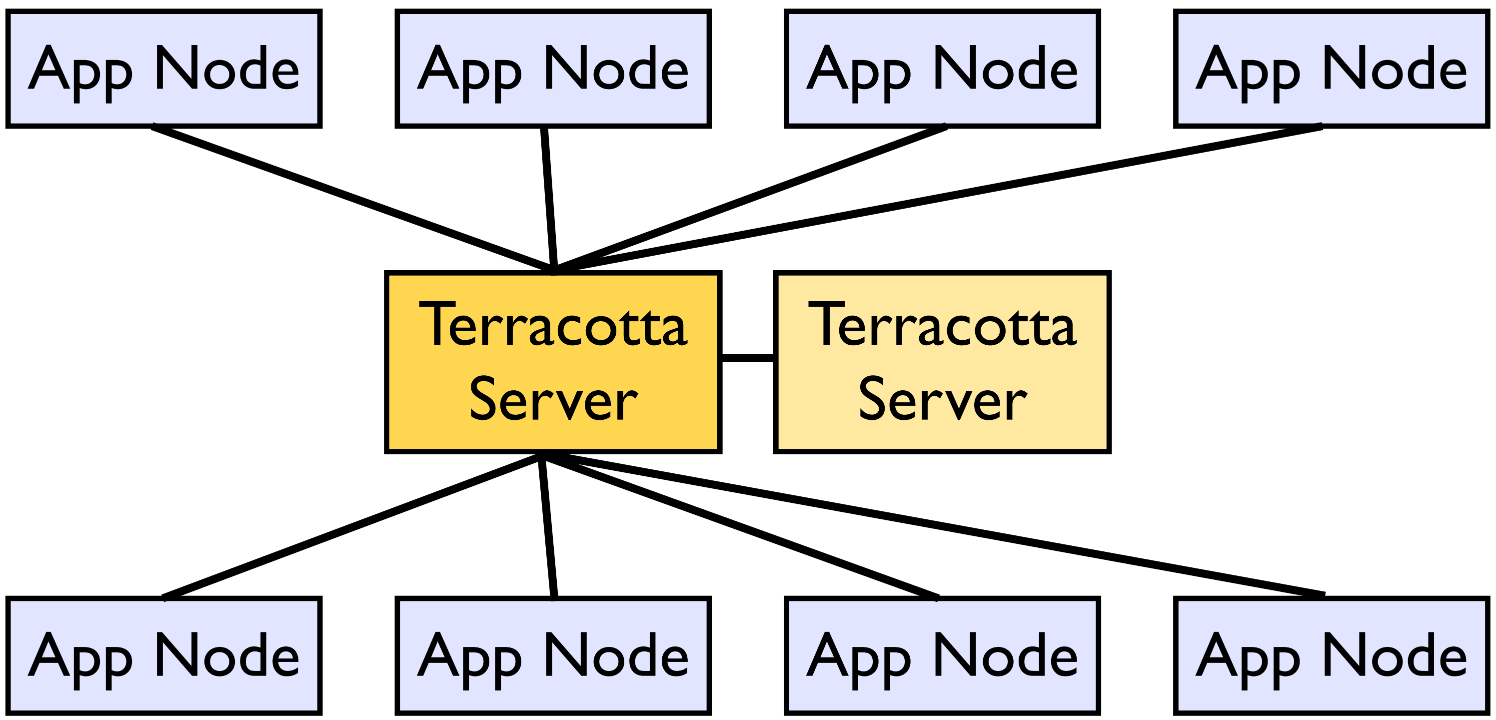
- What is Ehcache?
- What is Terracotta?

Ehcache Example


```
CacheManager manager = new CacheManager();
Ehcache cache = manager.getEhcache("employees");
cache.put(new Element(employee.getId(), employee));
Element element = cache.get(employee.getId());
```

```
    <cache  name="employees"
            maxElementsInMemory="1000"
            memoryStoreEvictionPolicy="LRU"
            eternal="false"
            timeToIdleSeconds="600"
            timeToLiveSeconds="3600"
            overflowToDisk="false"/>
```

Terracotta


App Node
App Node
App Node
App Node
Terracotta Server
Terracotta Server
App Node
App Node
App Node
App Node


But things are not
always so simple...

Pain of Large Data Sets

- How do I choose which elements stay in memory and which go to disk?
- How do I choose which elements to evict when I have too many?
- How do I balance cache size against other memory uses?

Eviction

When cache memory is full, what do I do?

- Delete - Evict elements
- Overflow to disk - Move to slower, bigger storage
- Delete local - But keep remote data

Eviction in Ehcache

Evict with “Least Recently Used” policy:

```
    <cache  name="employees"
            maxElementsInMemory="1000"
            memoryStoreEvictionPolicy="LRU"
            eternal="false"
            timeToIdleSeconds="600"
            timeToLiveSeconds="3600"
            overflowToDisk="false"/>
```

Spill to Disk in Ehcache

Spill to disk:

```
<diskStore path="java.io.tmpdir"/>

<cache  name="employees"
        maxElementsInMemory="1000"
        memoryStoreEvictionPolicy="LRU"
        eternal="false"
        timeToIdleSeconds="600"
        timeToLiveSeconds="3600"

        overflowToDisk="true"
        maxElementsOnDisk="1000000"
        diskExpiryThreadIntervalSeconds="120"
        diskSpoolBufferSizeMB="30" />
```

Terracotta Clustering

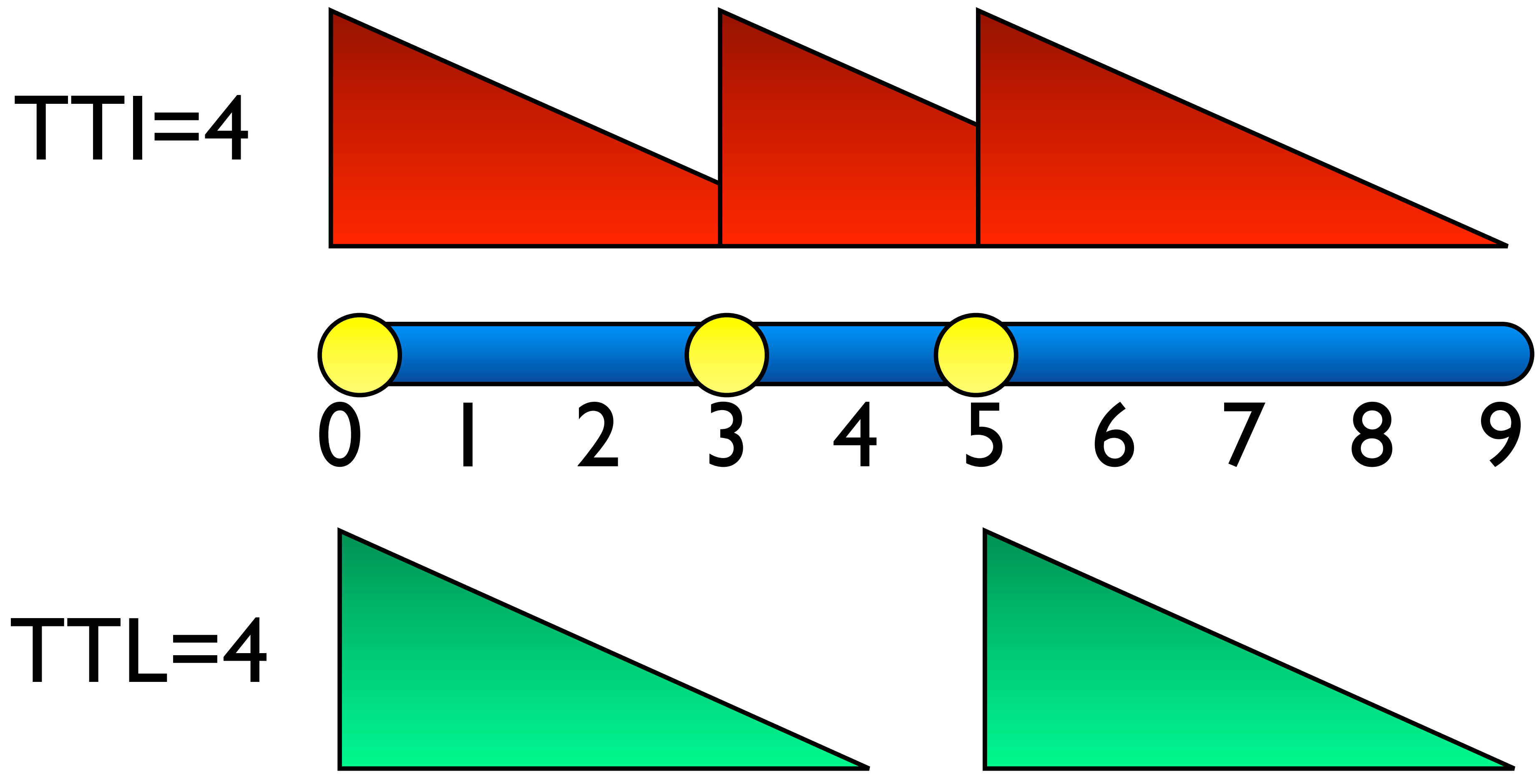
Terracotta configuration:

```
<terracottaConfig url="server1:9510,server2:9510"/>

<cache  name="employees"
        maxElementsInMemory="1000"
        memoryStoreEvictionPolicy="LRU"
        eternal="false"
        timeToIdleSeconds="600"
        timeToLiveSeconds="3600"
        overflowToDisk="false">

    <terracotta/>
</cache>
```

Pain of Stale Data

- How tolerant am I of seeing values changed on the underlying data source?
- How tolerant am I of seeing values changed by another node?

Expiration



TTI and TTL in Ehcache

```
<cache  name="employees"
        maxElementsInMemory="1000"
        memoryStoreEvictionPolicy="LRU"
        eternal="false"
        timeToIdleSeconds="600"
        timeToLiveSeconds="3600"
        overflowToDisk="false"/>
```

Replication in Ehcache

```
<cacheManagerPeerProviderFactory
    class="net.sf.ehcache.distribution.
           RMICacheManagerPeerProviderFactory"
    properties="hostName=fully_qualified_hostname_or_ip,
                peerDiscovery=automatic,
                multicastGroupAddress=230.0.0.1,
                multicastGroupPort=4446, timeToLive=32"/>

<cache name="employees" ...>
    <cacheEventListenerFactory
class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
         properties="replicateAsynchronously=true,
         replicatePuts=true,
         replicatePutsViaCopy=false,
         replicateUpdates=true,
         replicateUpdatesViaCopy=true,
         replicateRemovals=true
         asynchronousReplicationIntervalMillis=1000"/>
</cache>
```

Terracotta Clustering

Still use TTI and TTL to manage stale data between cache and data source

Coherent by default but can relax with coherentReads=”false”

Pain of Loading

- How do I pre-load the cache on startup?
- How do I avoid re-loading the data on every node?

Persistent Disk Store

```
<diskStore path="java.io.tmpdir"/>

<cache  name="employees"
        maxElementsInMemory="1000"
        memoryStoreEvictionPolicy="LRU"
        eternal="false"
        timeToIdleSeconds="600"
        timeToLiveSeconds="3600"
        overflowToDisk="true"
        maxElementsOnDisk="1000000"
        diskExpiryThreadIntervalSeconds="120"
        diskSpoolBufferSizeMB="30"

        diskPersistent="true" />
```

Bootstrap Cache Loader

Bootstrap a new cache node from a peer:

```
<bootstrapCacheLoaderFactory
      class="net.sf.ehcache.distribution.
             RMIBootstrapCacheLoaderFactory"
      properties="bootstrapAsynchronously=true,
                  maximumChunkSizeBytes=5000000"
      propertySeparator=",” />
```

On startup, create background thread to pull the existing cache data from another peer.

Terracotta Persistence

Nothing needed beyond setting up
Terracotta clustering.

Terracotta will automatically bootstrap:
- the cache key set on startup
- cache values on demand

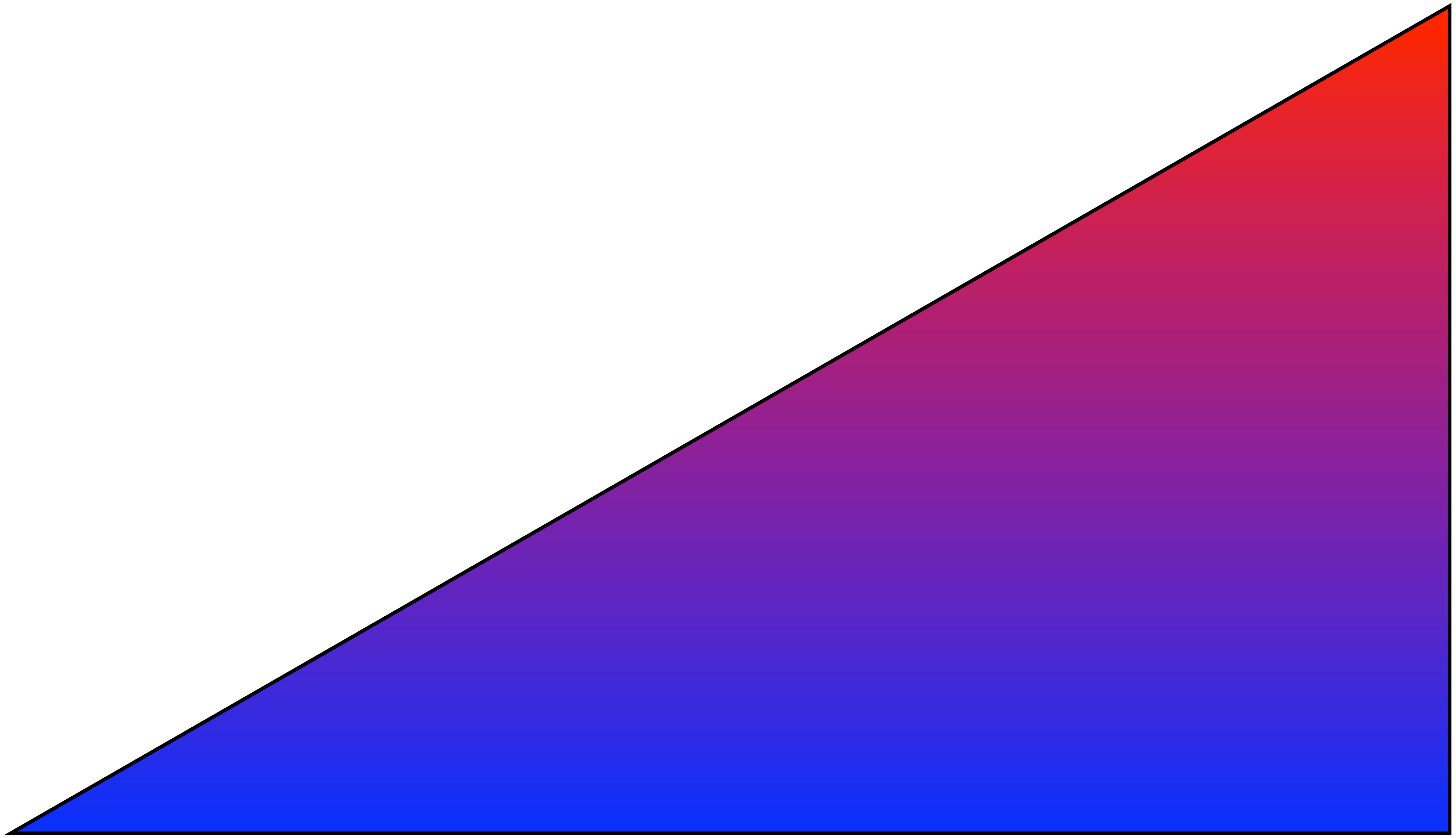
Pain of Duplication

- How do I get failover capability while avoiding excessive duplication of data?

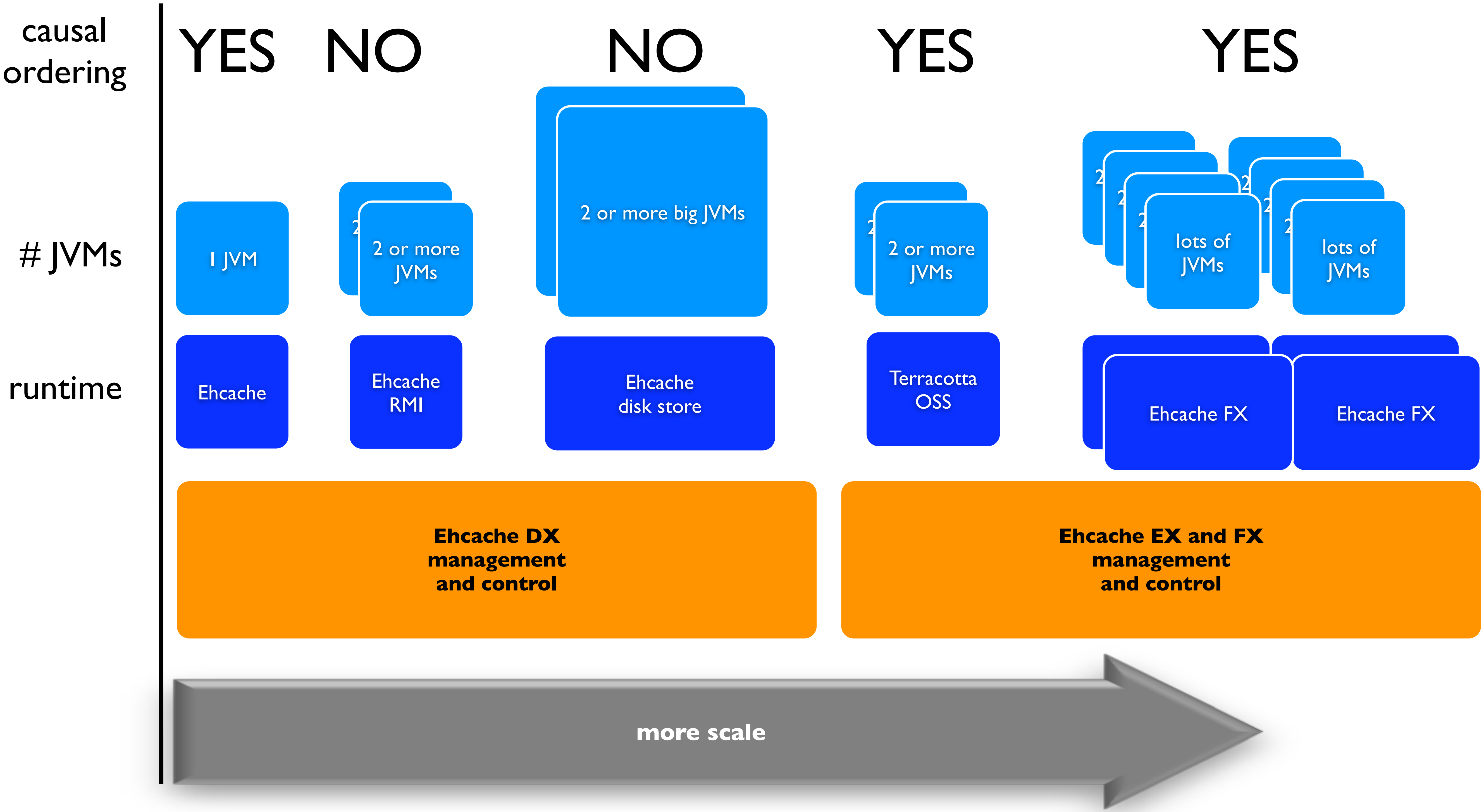
Partitioning + Terracotta Virtual Memory

- Each node (mostly) holds data it has seen
- Use load balancer to get app-level partitioning
- Use fine-grained locking to get concurrency
- Use memory flush/fault to handle memory overflow and availability
- Use causal ordering to guarantee coherency

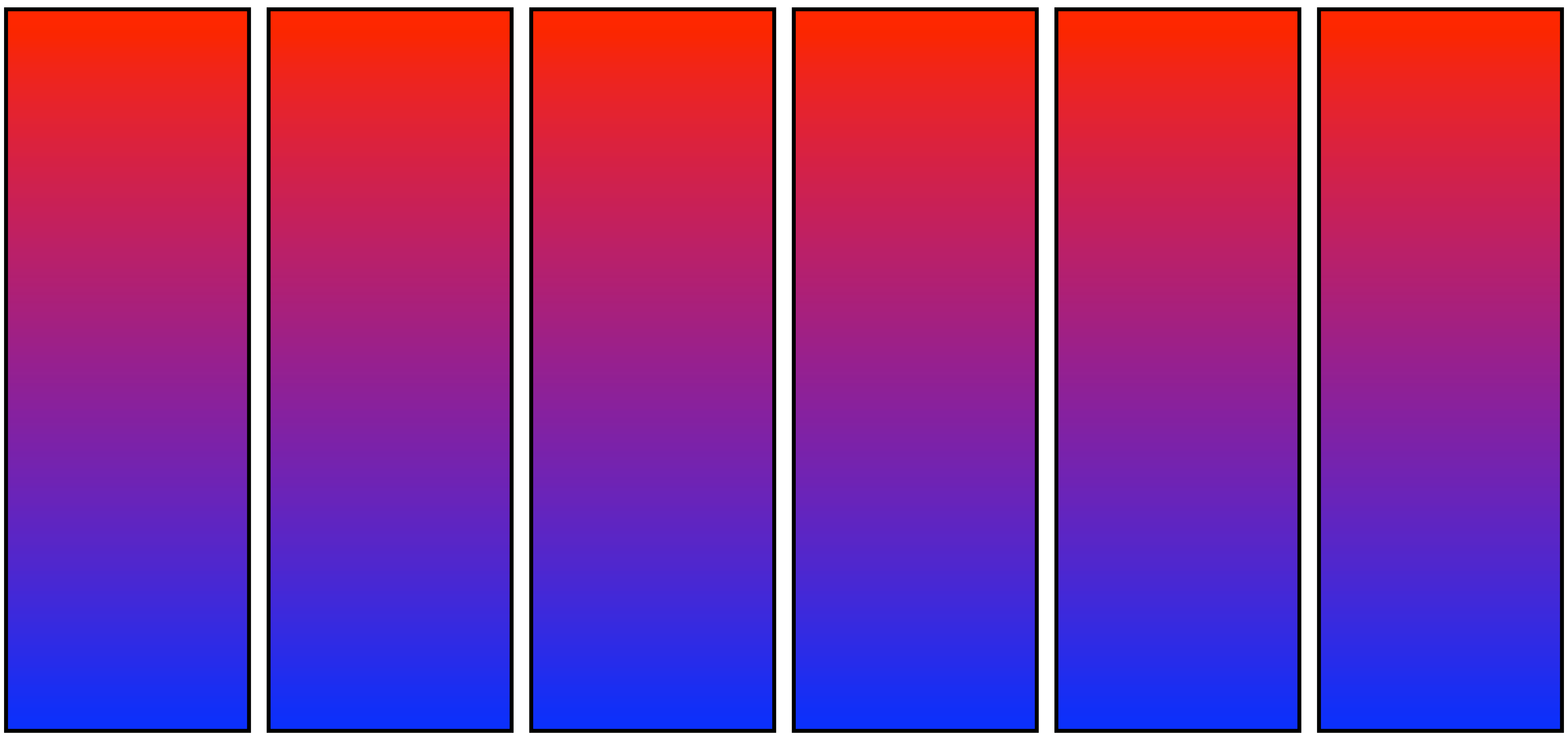
Scaling Your Cache



Scalability Continuum



Caching at Scale



Know Your Use Case

- Is your data partitioned (sessions) or not (reference data)?
- Do you have a hot set or uniform access distribution?
- Do you have a very large data set?
- Do you have a high write rate (50%)?
- How much data consistency do you need?

Types of caches

Name	Communication	Advantage
Broadcast invalidation	multicast	low latency
Replicated	multicast	offloads db
Datagrid	point-to-point	scalable
Distributed	2-tier point-to-point	all of the above

Common Data Patterns

I/O pattern	Locality	Hot set	Rate of change
Catalog/ customer	low	low	low
Inventory	high	high	high
Conversations	high	high	low

Catalogs/customers

- warm all the data into cache
- High TTL

Inventory

- fine-grained locking
- write-behind to DB

Conversations

- sticky load balancer
- disconnect conversations from DB

Build a Test

- As realistic as possible
- Use real data (or good fake data)
- Verify test does what you think
- Ideal test run is 15-20 minutes

Cache Warming

Cache Warming

- Explicitly record cache warming or loading as a testing phase
- Possibly multiple warming phases

Lots o’ Knobs

Things to Change

- Cache size
- Read / write / other mix
- Key distribution
- Hot set
- Key / value size and structure
- # of nodes

Lots o' Gauges

Things to Measure

- Application throughput (TPS)
- Application

Benchmark and Tune

- Create a baseline
- Run and modify parameters
  - Test, observe, hypothesize, verify
- Keep a run log

Bottleneck Analysis

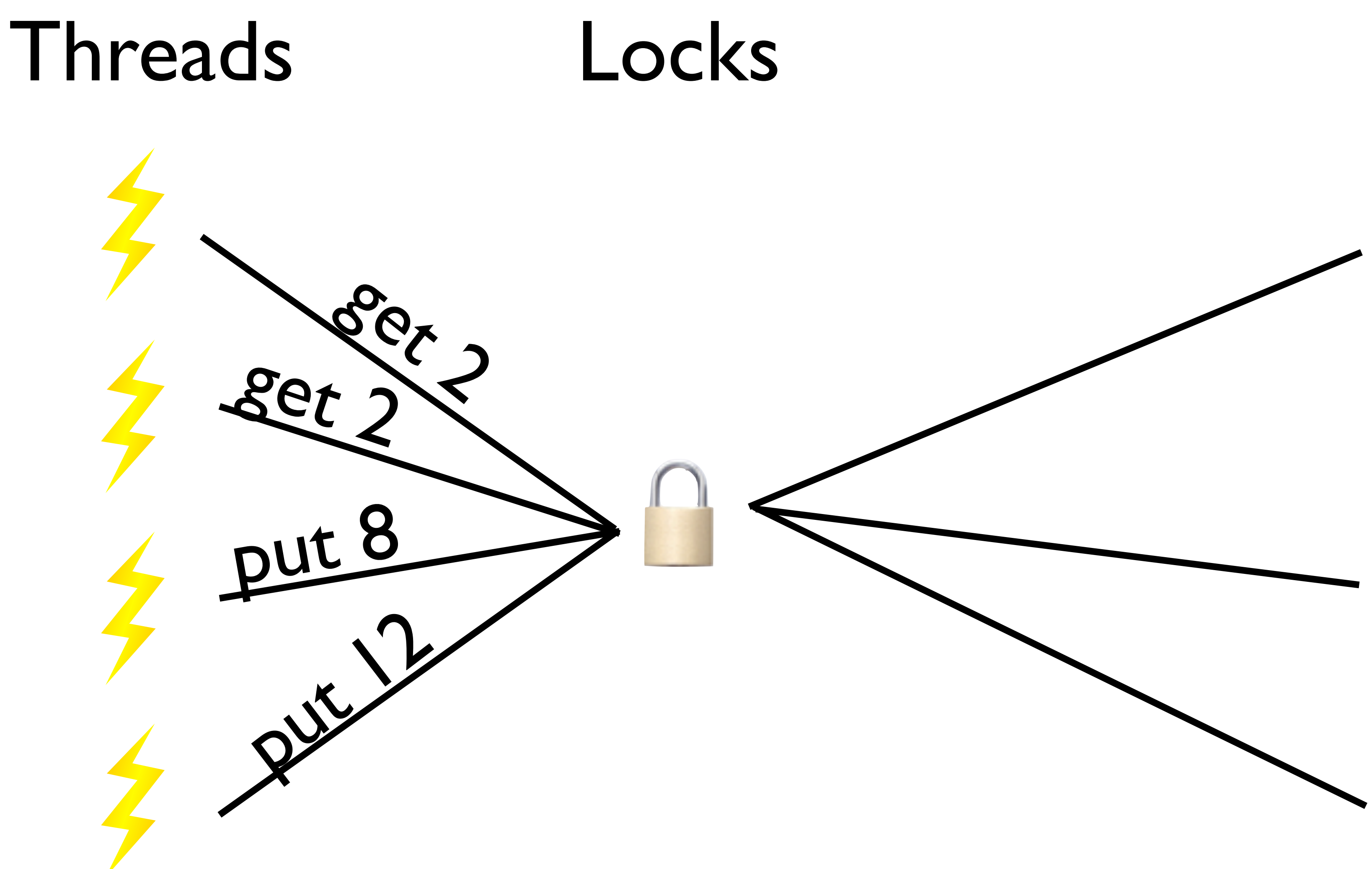
Pushing It

- If CPUs are not all busy...
  - Can you push more load?
  - Waiting for I/O or resources
- If CPUs are all busy...
  - Latency analysis

I/O Waiting

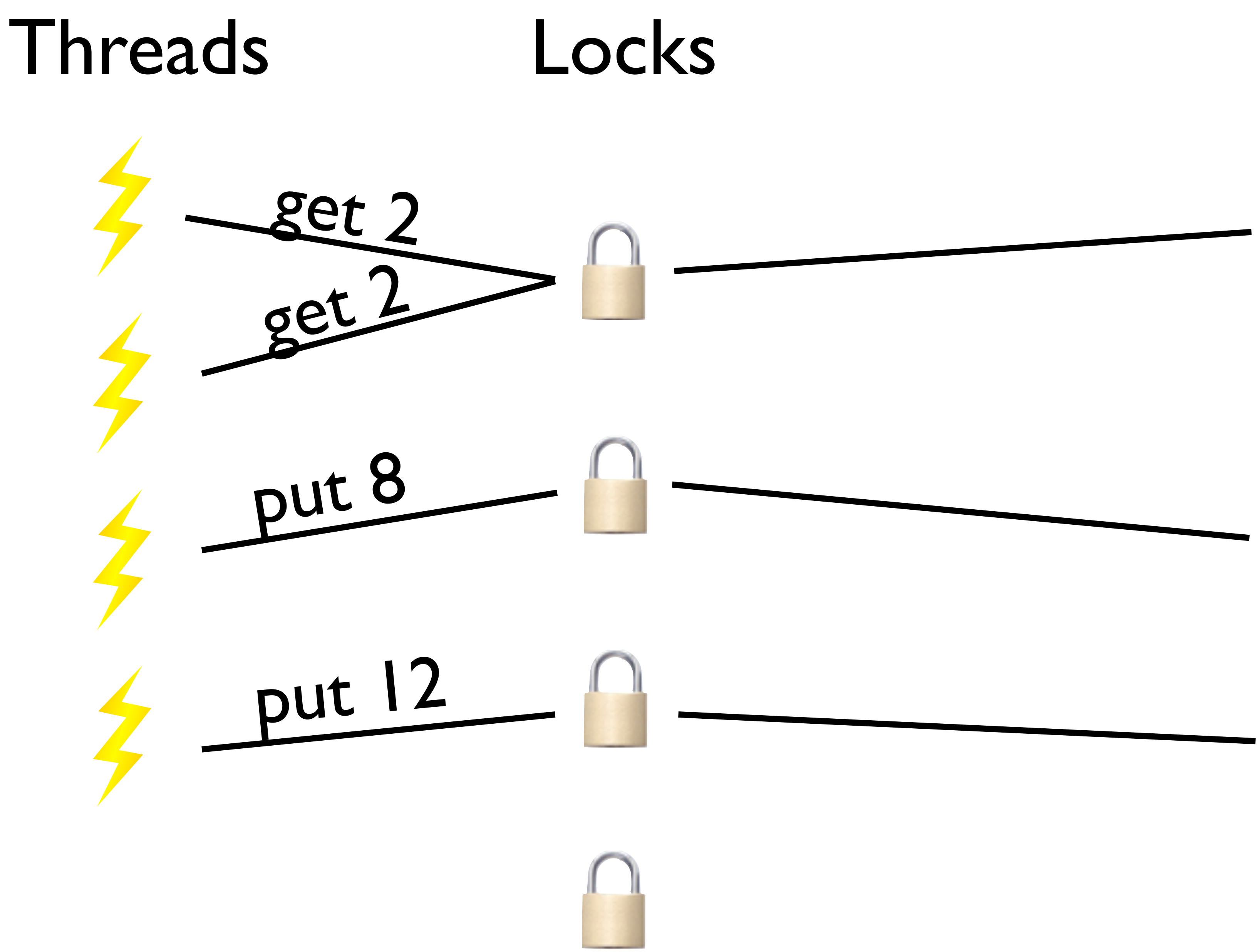
- Database
  - Connection pooling
  - Database tuning
  - Lazy connections
- Remote services

Locking and Concurrency



Key	Value
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	

Locking and Concurrency



Key	Value
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	

Objects and GC

- Unnecessary object churn
- Tune GC
  - Concurrent vs parallel collectors
  - Max heap
  - ...and so much more
- Watch your GC pauses!!!

Cache Efficiency

- Watch hit rates and latencies
  - Cache hit - should be fast
    - Unless concurrency issue
  - Cache miss
    - Miss local vs
    - Miss disk / cluster

Cache Sizing

- Expiration and eviction tuning
  - TTI - manage moving hot set
  - TTL - manage max staleness
  - Max in memory - keep hot set resident
  - Max on disk / cluster - manage total disk / clustered cache

Cache Coherency

- No replication (fastest)
- RMI replication (loose coupling)
- Terracotta replication (causal ordering) - way faster than strict ordering

Latency Analysis

- Profilers
- Custom timers
- Tier timings
- Tracer bullets

mumble-mumble*

It’s time to add it to Terracotta.

* lawyers won’t let me say more

Thanks!

- Twitter - @puredanger
- Blog - http://tech.puredanger.com
- Terracotta - http://terracotta.org
- Ehcache - http://ehcache.org