goto; conference aarhus

Getting to know the Grid

Syed M Shaaf
Red Hat

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

Quick introduction

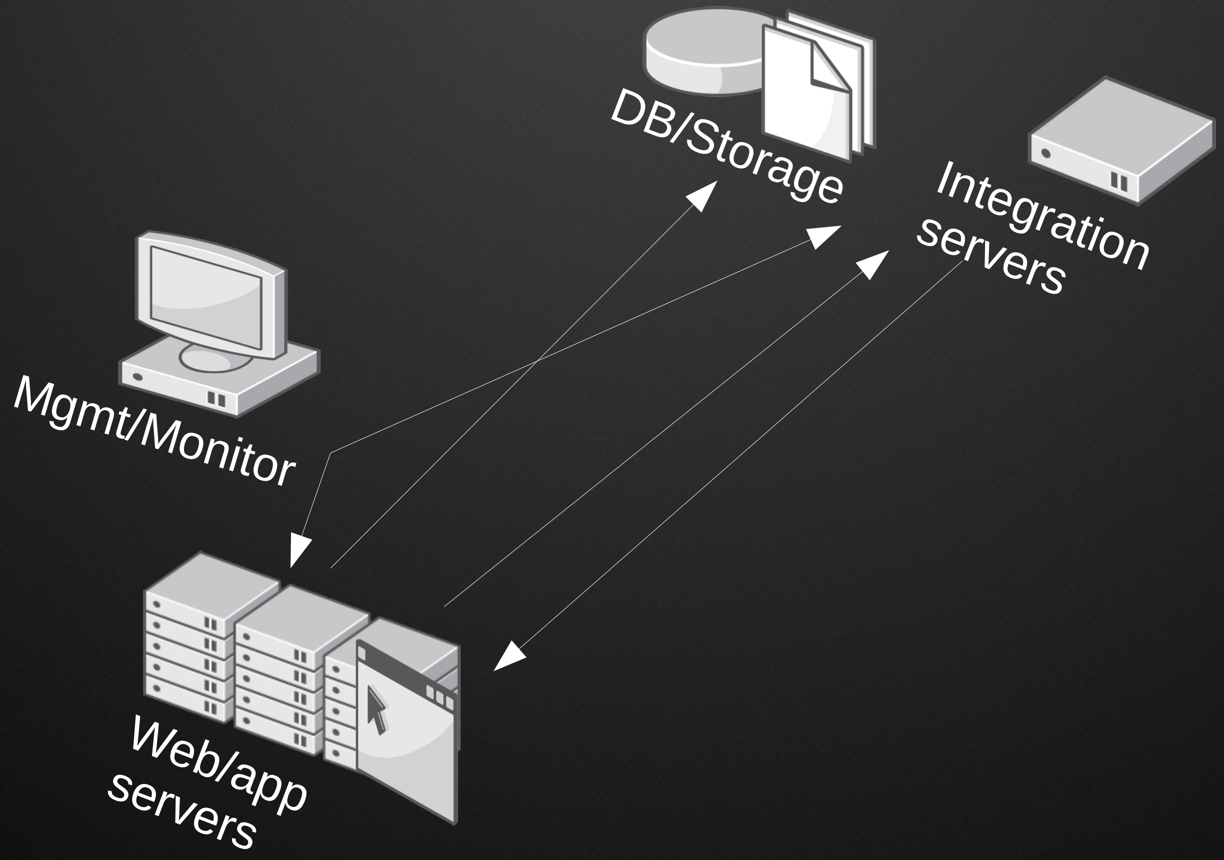
Solutions Architect at Redhat Nordics

Red Hat JBoss middleware

@sshaaf @RedHatNordics

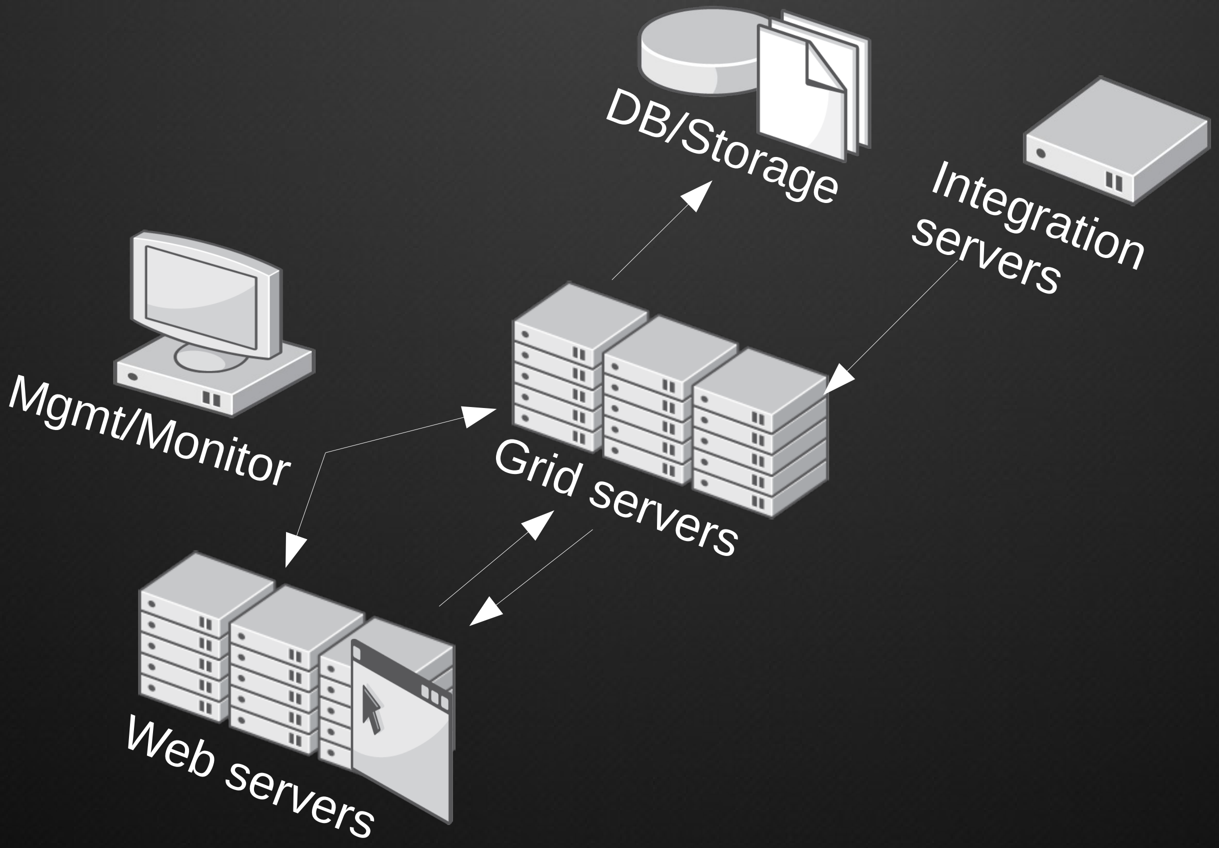
http://www.redhat.com

One Scenario



Another Scenario

Data Replication and Cache



What is?

- Schema-less key/value store
- Compatible with applications written in any language, using any framework
- Easy access through APIs

- Consistent hash-based distribution
  - Self-healing
  - No single point of failure
- Durability (persistence)
- Memory management (eviction, expiration)
- XA transactions

JBoss Data Grid and JSR

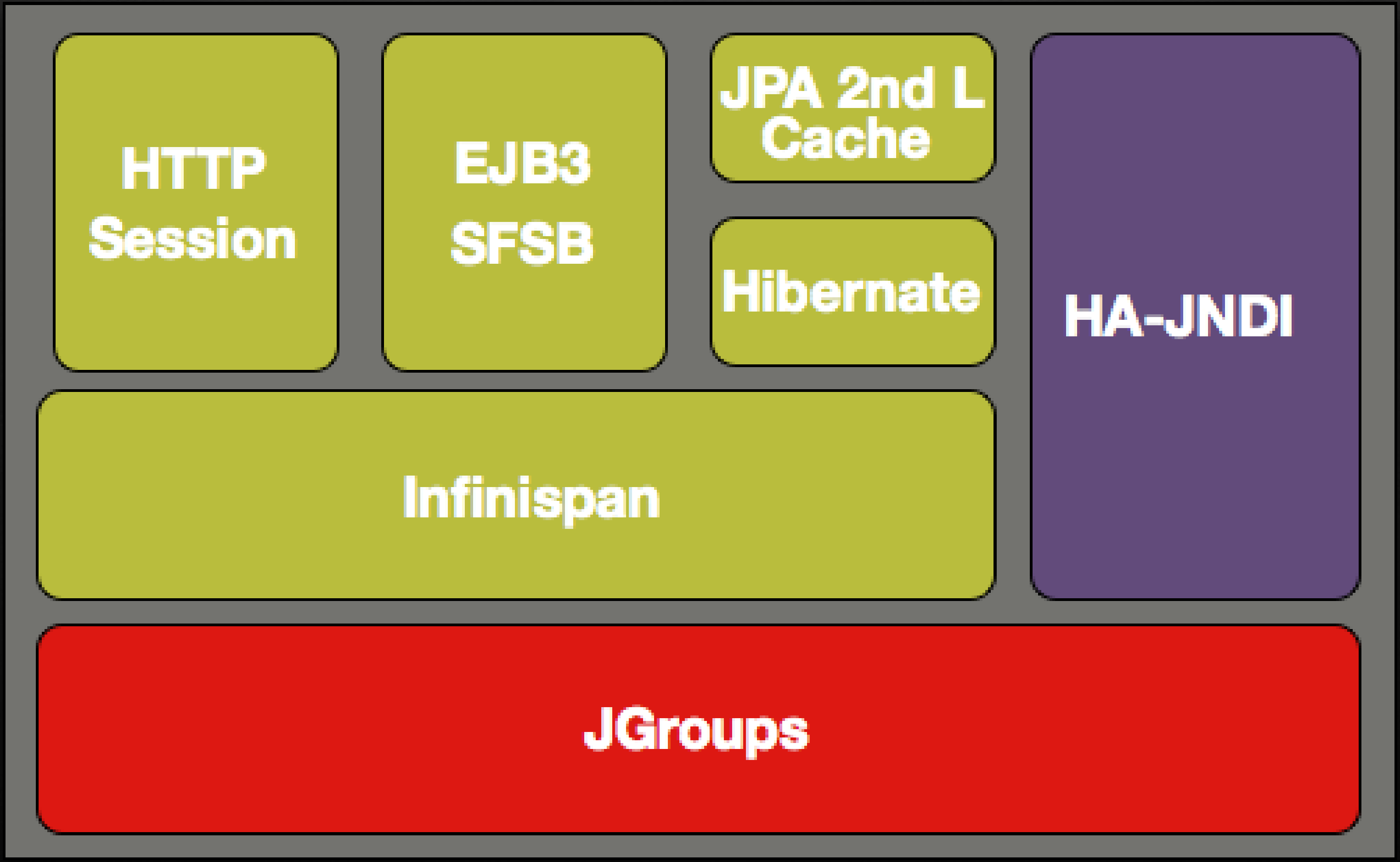
- JSR-107: Temporary caching API
- JSR-347: Data grids
  - Development led by Red Hat
- JSR-346: CDI1.1
  - Programming model for data grids
- JSR-317: JPA2
  - Data grids form caching API for database via JPA2

And then its a matter of scaling..

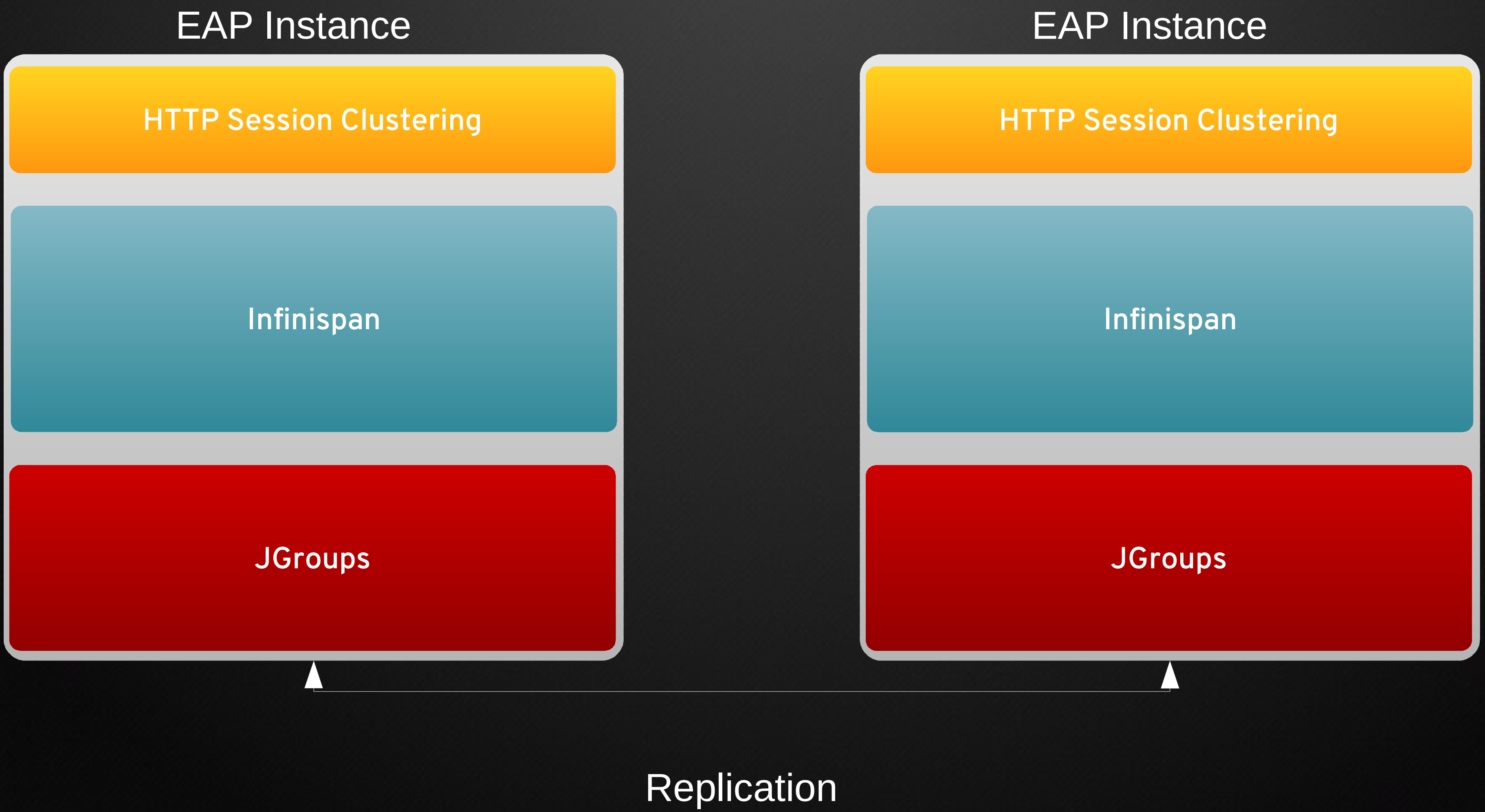
Clustering subsystems

- **JGROUPS** - toolkit for the underlying communication between nodes . Configured with 2 stacks for communication **UDP** (default) and **TCP** (if the environment is not multicasting)
- **INFINISPAN** - data caching and object replication and comes with 3 preconfigured caches:
  - **cluster** - Replication of objects in a HA cluster
  - **web** - Session replication
  - **sfsb** - Replication of stateful session bean
  - **hibernate** - 2nd level entity caching for JPA/Hibernate
- **MODCLUSTER**- software LB spreads requests among two or more nodes

Clustering architecture



Cluster architecture



mode=replication

All the data is stored on all cluster nodes

Writes are sent to all nodes

- Every node updates its local cache

Reads are always local

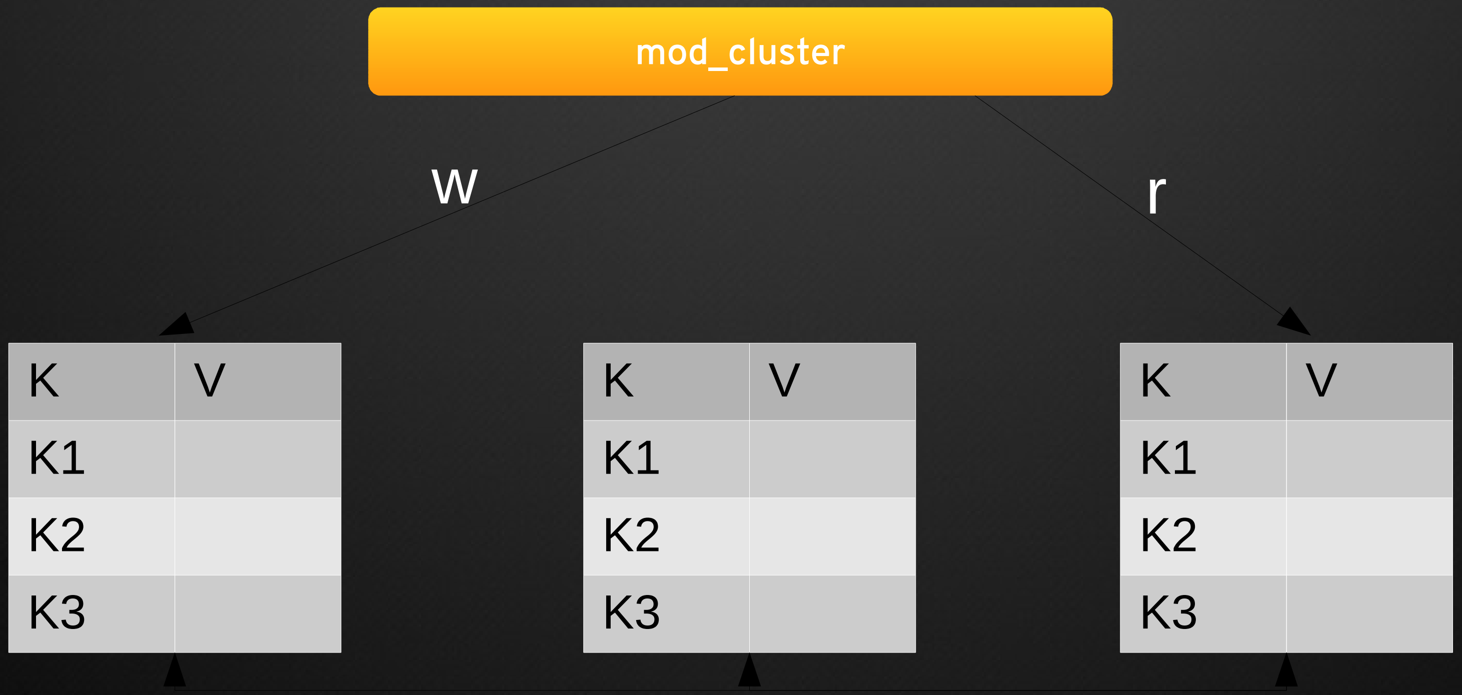
New nodes acquire the initial state from the oldest node

Clients can access any node for reading or writing

Scalability is limited by cluster size and data size

10 nodes with 100MB state each: every node needs 1GB

mode=replication; action=rw



mod_cluster

w

r

K	V
K1	
K2	
K3	

K	V
K1	
K2	
K3	

K	V
K1	
K2	
K3	

Replication

Mode=distribution

Data is only stored on N cluster nodes (say N=2)

A consistent hash on a key “id” determines the 2 servers for “id”

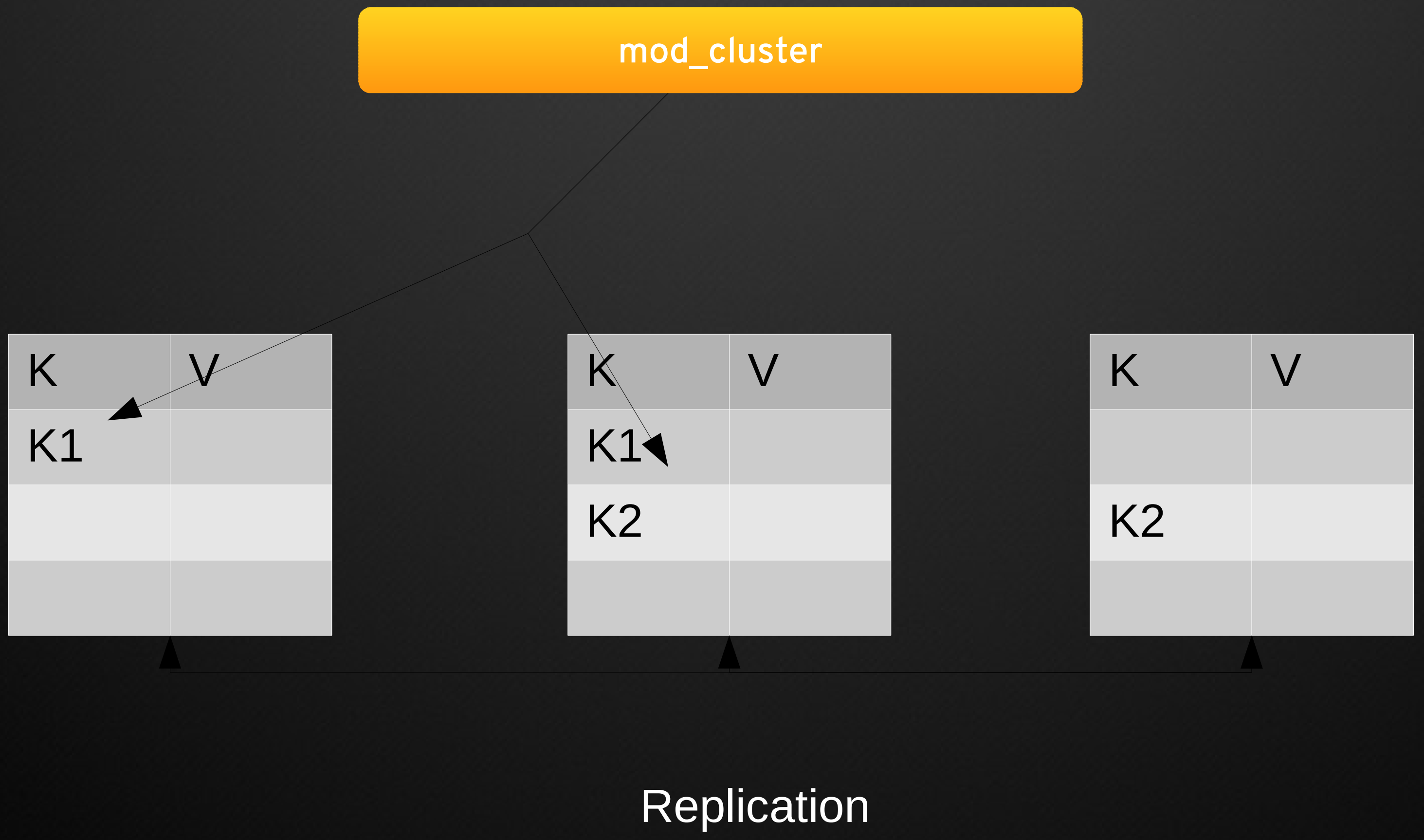
- Example: cluster is {A,B,C,D,E,F}
- Hash(“id”) = 8; 8 MOD 6 = 2
- --> Primary owner = B, backup owner = C

Crash of B, new view is {A,C,D,E,F}

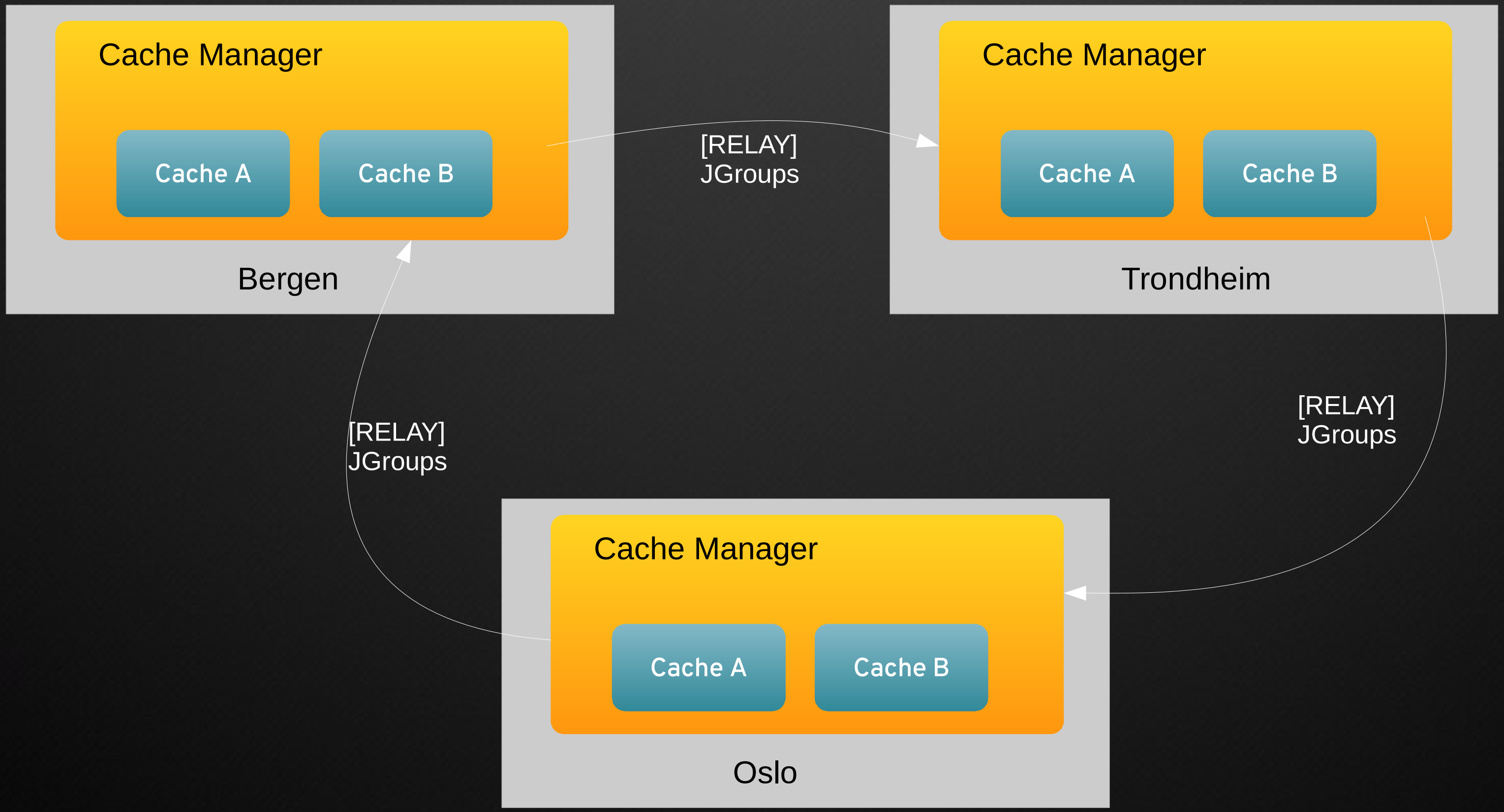
- --> Primary owner = D, backup owner = E
- --> C needs to transfer “id” to D and E and remove it locally

Knowing the key, we always find the right server(s)

mode=distribution; action=w



Cross Site replication



Data access is important?

Client and server

Multiple access protocols

Protocol	Format	Client type	Smart?	Load balance and failover
REST	text	any	no	external
Memcached	text	any	no	pre-defined
HotRod	binary	Java, C#, Python	yes	auto/dynamic

Advanced functionality

Eviction, expiration, and passivation

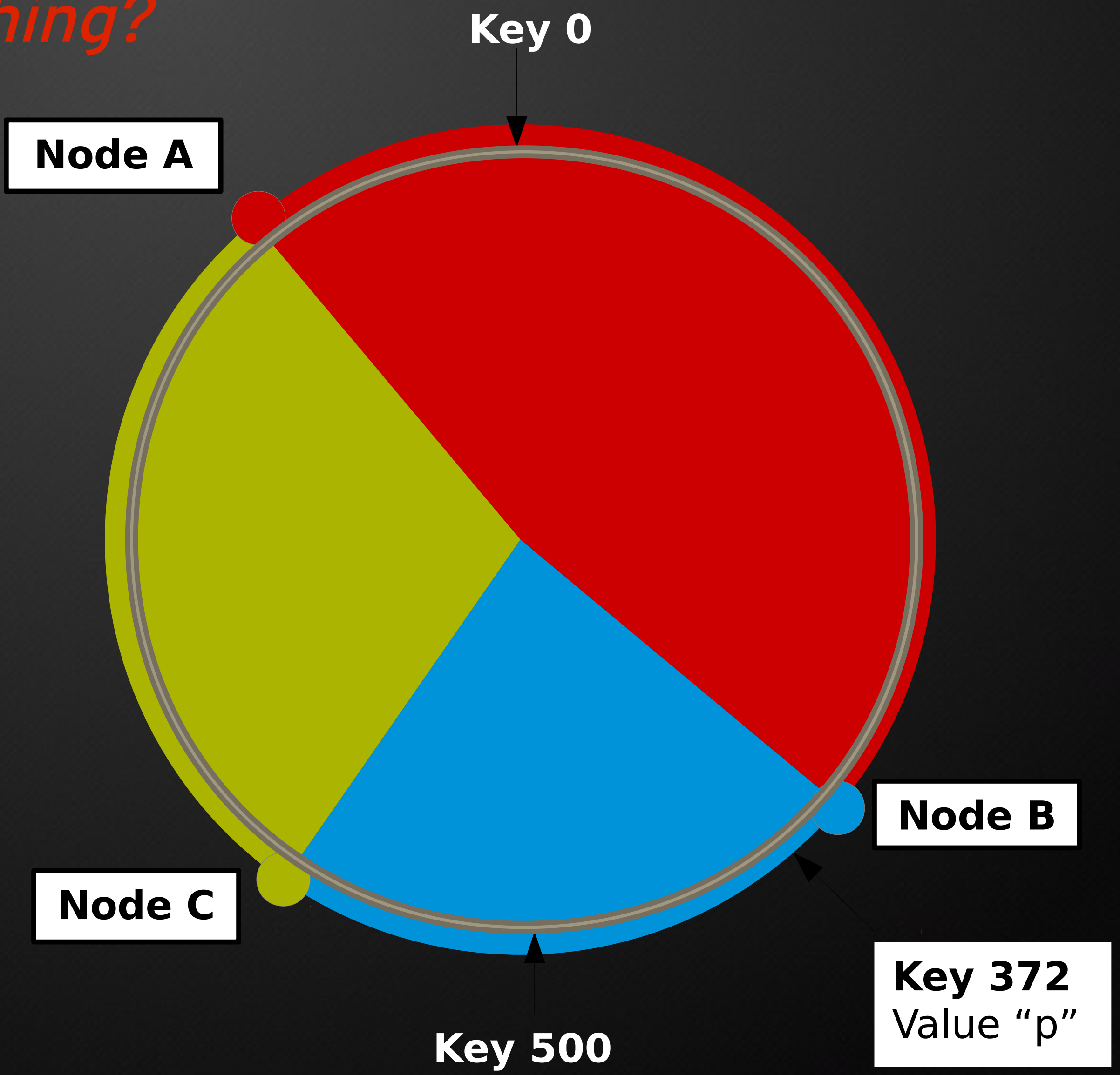
- Expiration – defined per entry or cache
- Eviction – FIFO, LRU, unordered, LIRS, none
- Passivation

Step	Action	Keys in memory	Keys on disk
1	Insert K1	K1	n/a
2	Insert K2	K1, K2	n/a
3	Eviction thread - K1	K2	K1
4	Read K1	K1, K2	n/a
5	Eviction thread K2	K1	K2
6	Remove K2	K1	n/a

Advanced functionality

Why use consistent hashing?

- Cost-effective, speed benefits
- Deterministic location of keys
- Sufficient copies for fault tolerance and durability but without an overabundance of copies



Advanced functionality

Consistent hashing

Hash ring

- Cost-effective, speed benefits
- Deterministic location of keys
- Sufficient copies for fault tolerance and durability without an overabundance of copies

Node A

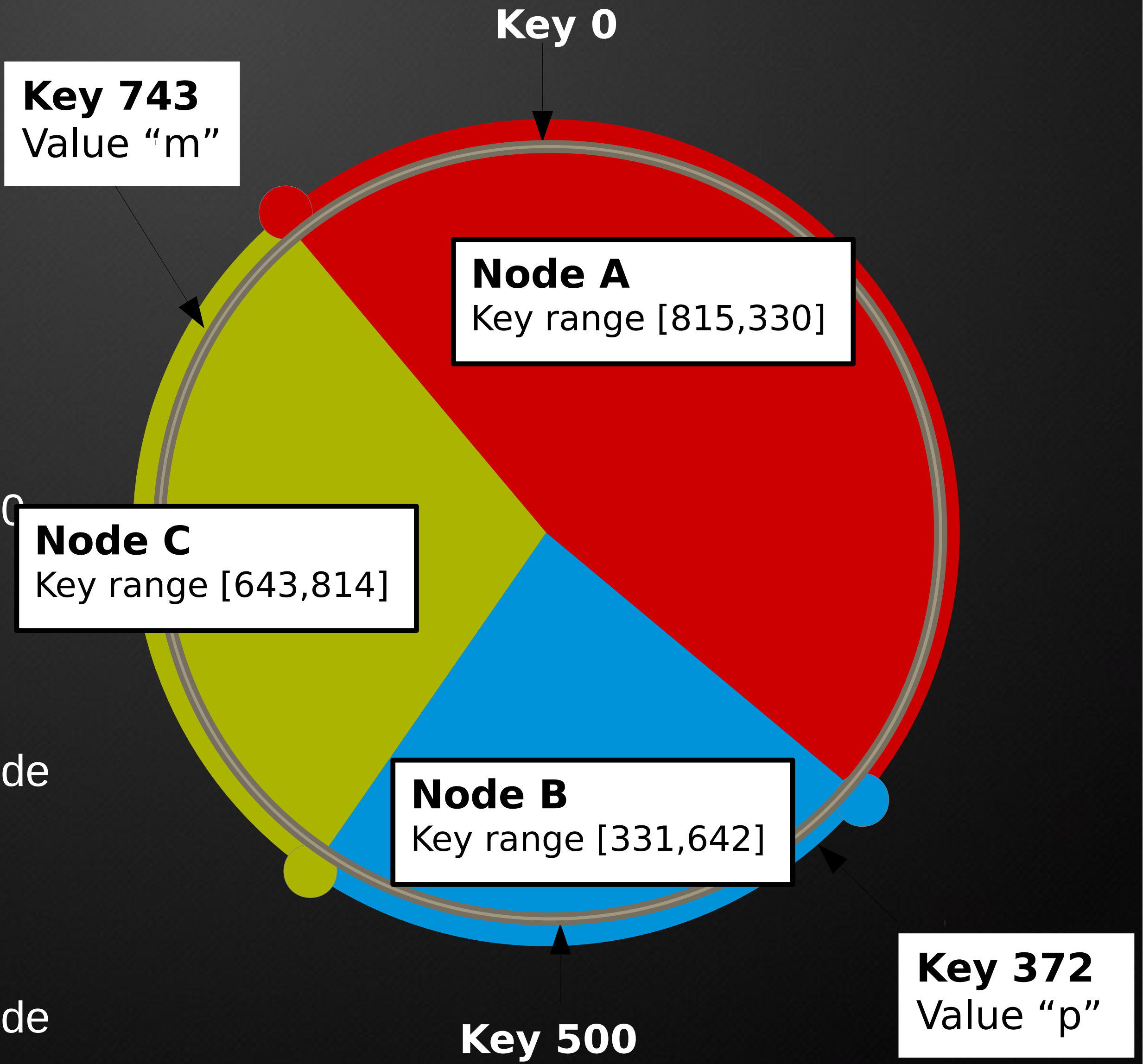
- Stores values of keys 815-1000-330
- Wraps around

Value “m”

- Stored in Key 743
- Based on key value, located on Node C

Value “p”

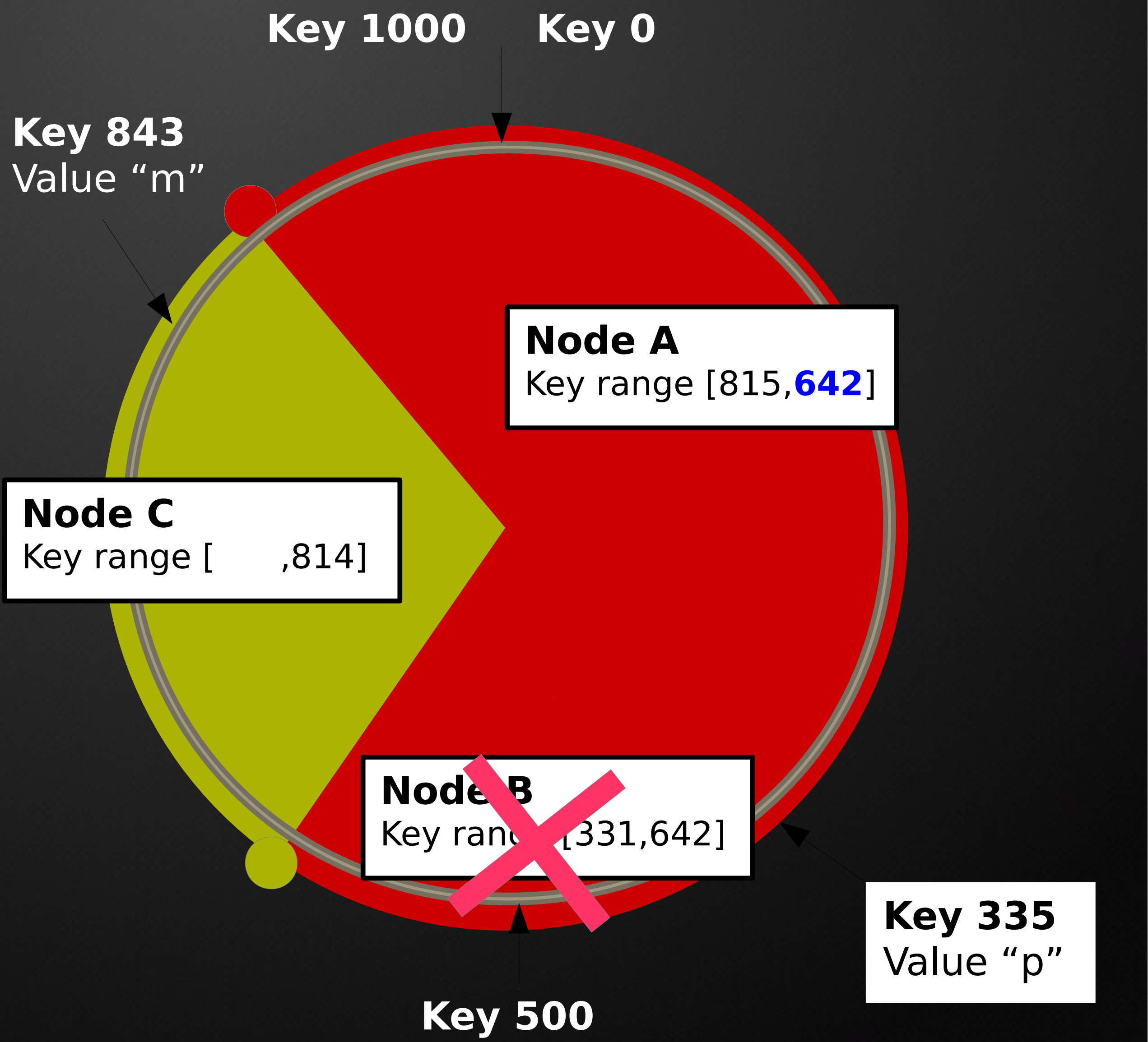
- Stored in Key 372
- Based on key value, located on Node B



Advanced functionality

Consistent hashing

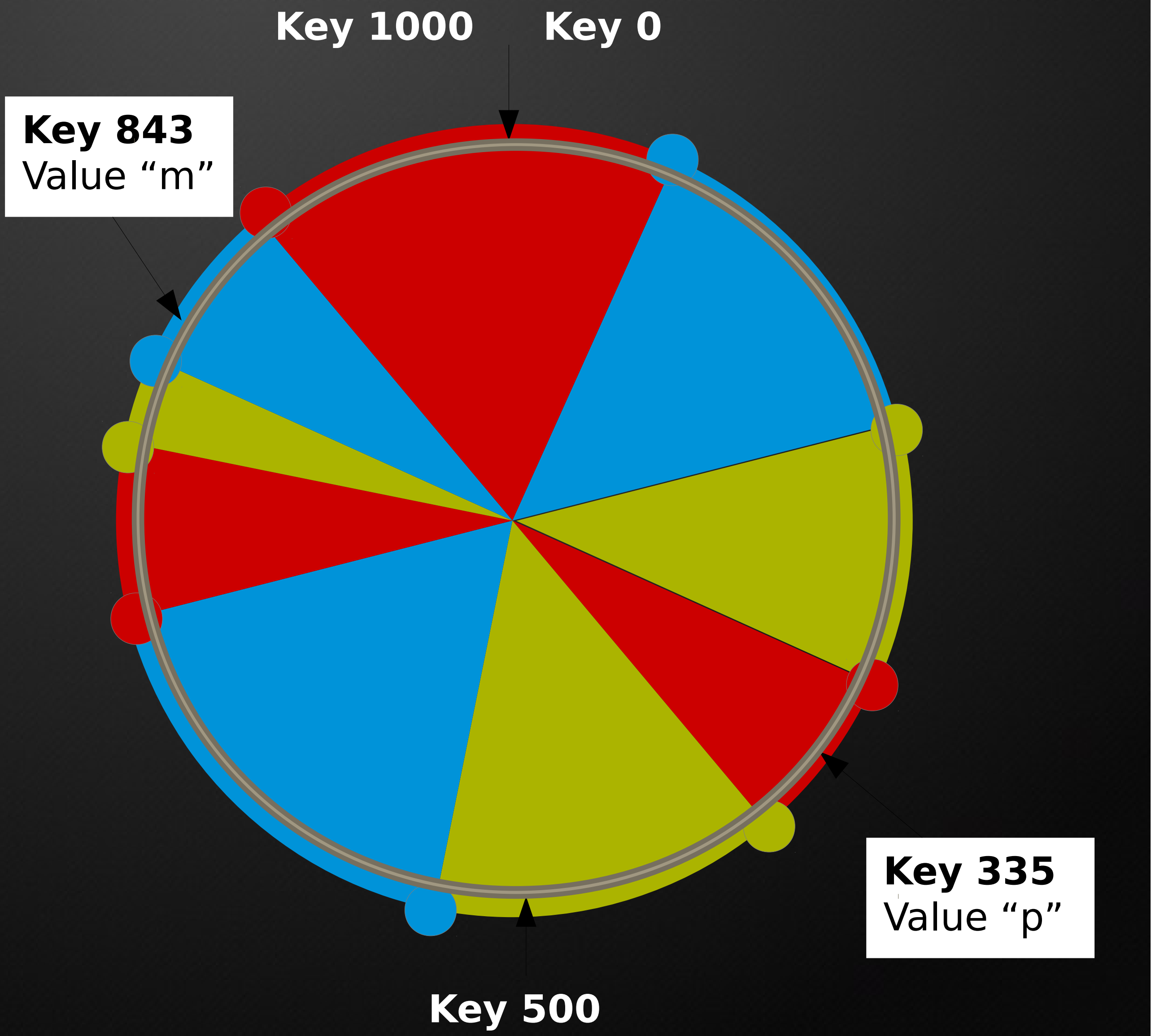
- Event: Node B goes offline
- Node A
  - Now stores keys 815-642
- Node C - unchanged
- Value “m” - unchanged
- Value “p”
  - Stored in key 335
  - Now located on Node A



Advanced functionality

Consistent hashing – Virtual nodes

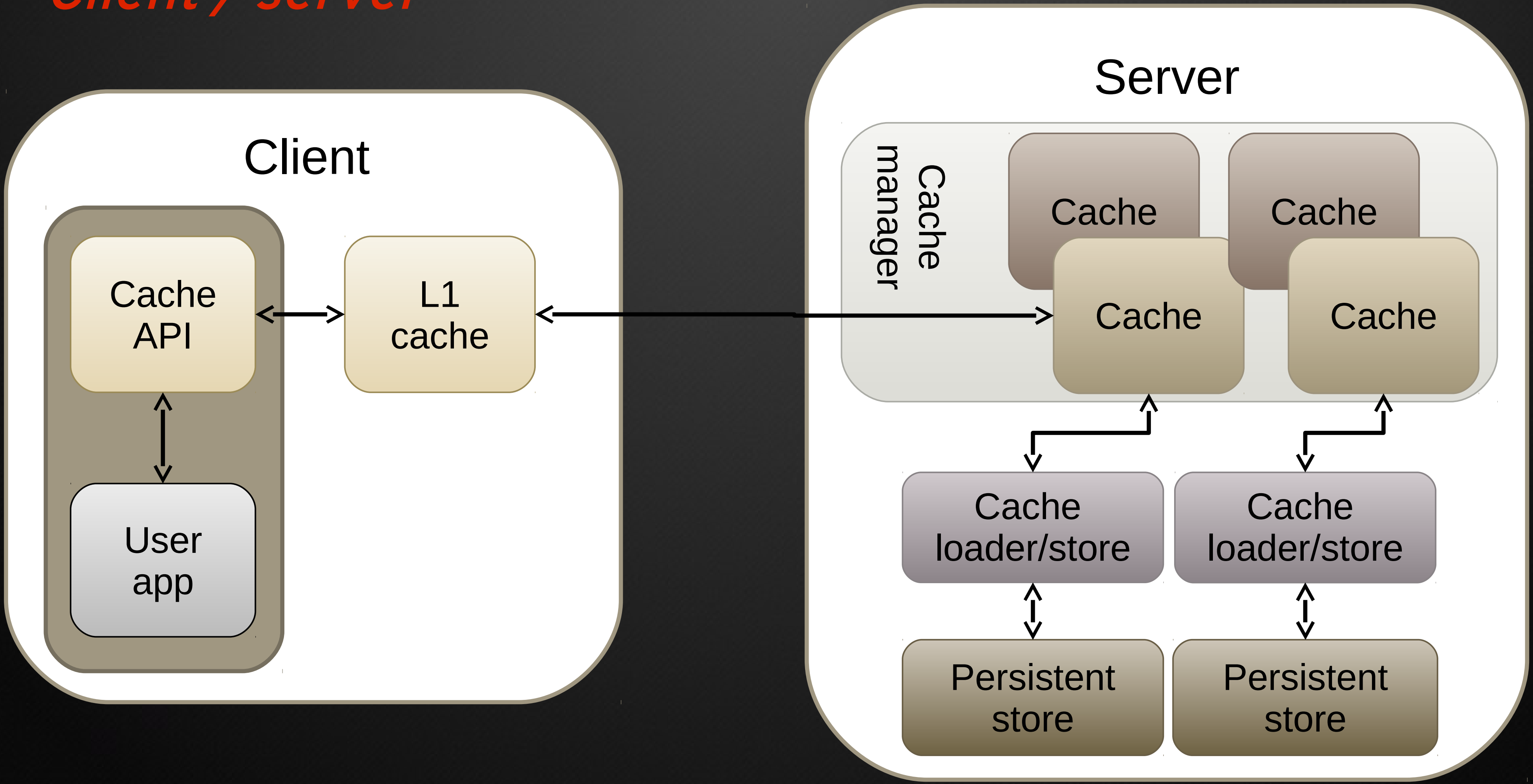
- Addresses irregularities in node distribution
- Location of entry determined algorithmically
- Allocates multiple blocks throughout the hash space when a node joins or leaves grid



Conceptual architecture

JBoss Data Grid conceptual architecture

Client / server



Conceptual architecture

Cache API and L1 cache

User application

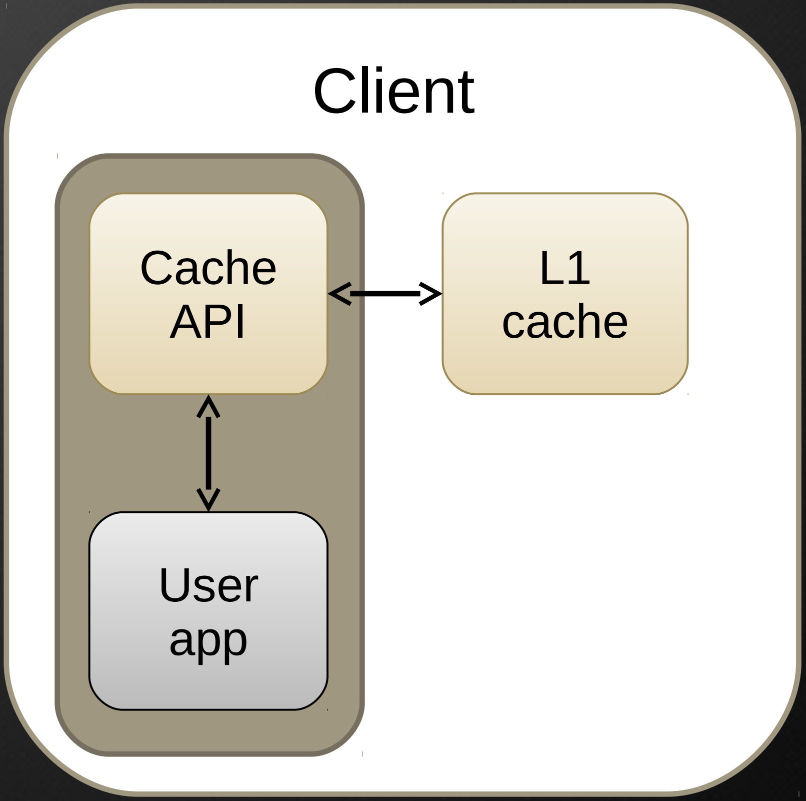
- End-user interface (i.e. web application, Java server application)

Cache API

Uses memcached, Hot Rod, or REST APIs

L1 near cache

- Stores remote cache entries after they are initially accessed
- For fast retrieval and to prevent unnecessary remote fetch operations



Conceptual architecture

Cache and cache manager

Cache manager

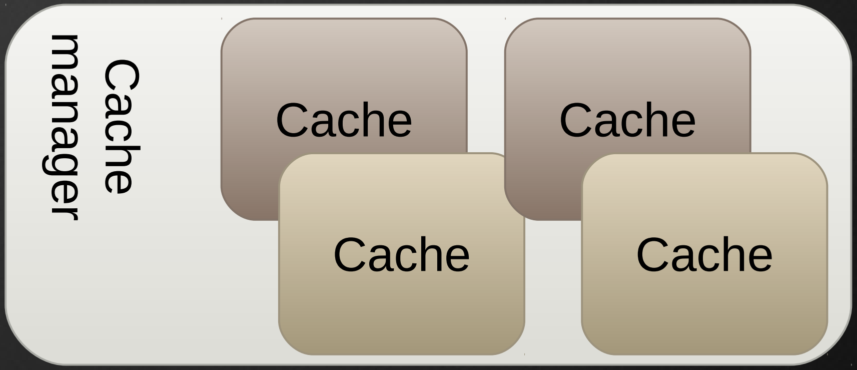
- Primary mechanism to retrieve a cache instance

Cache

- Houses cache instances

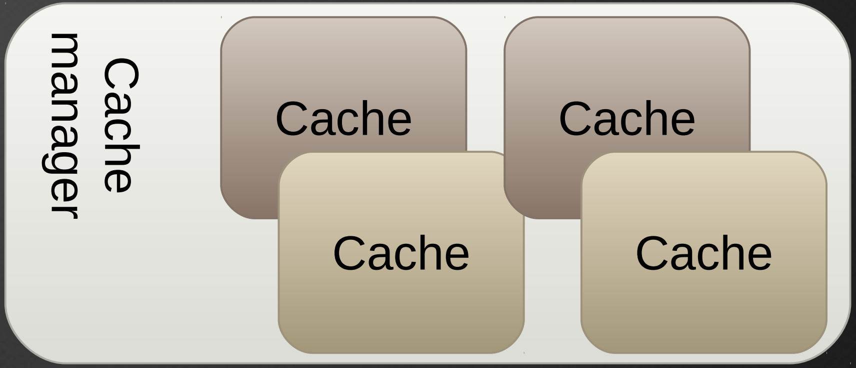
Flexible setup

- One cache manager per process
- Multiple caches per cache manager
- One interface per cache



Conceptual architecture

Cache and cache manager



Cache configuration

- Locking policy
- Transactions
- Eviction policy
- Expiration policy
- Persistence mechanism
- Backups
- L1 cache policy

Cache manager configuration

- Name / Alias / JNDI
- Start-up policy
- Transport policies
- Caches

Conceptual architecture

Cache store, cache loader, and persistent store

Cache loader

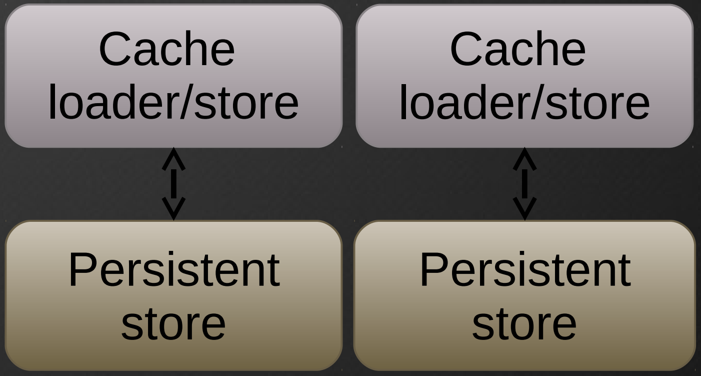
- Ready-only interface – locate and retrieve data

Cache store

- Cache loader with write capabilities

Persistent store

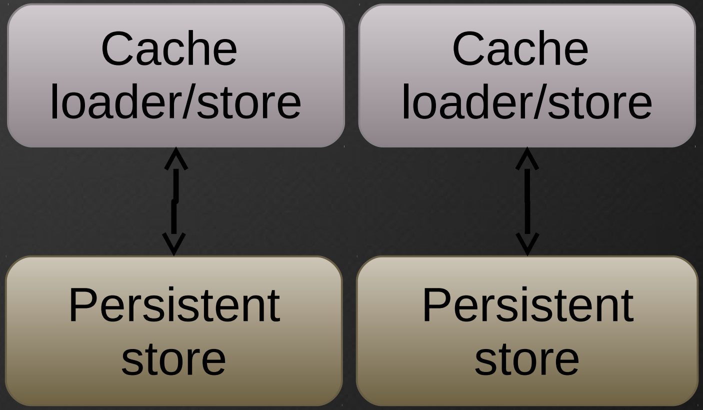
- Permanent store for cache instances and entries (i.e. relational database)



Conceptual architecture

The cache store

- Write-behind or write-through behavior
- A cache has one or more cache stores
- Cache stores can be chained
- Can be loaded or purged on start
- Open and supported API for custom stores
- File, JDBC, remote



JBoss Data Grid: Use cases

Use case - Local cache

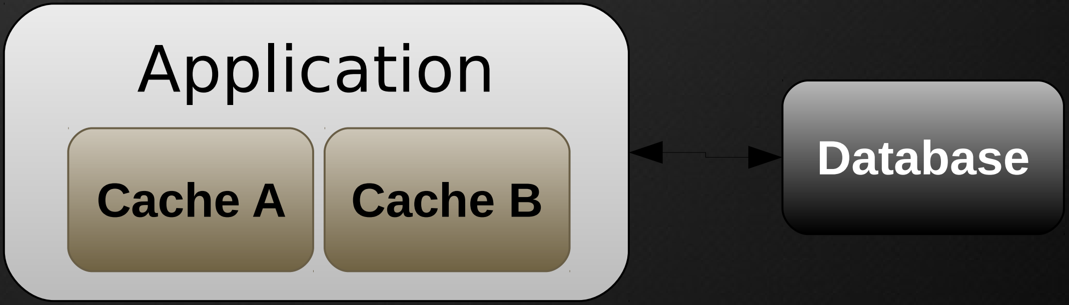
Boost application performance

A more sophisticated HashMap

- Memory management
  - Persistence
  - Eviction, expiration
  - Eliminate OOM
- Warm-start, preload
- Transaction capable (JTA)
- Monitor-able (JMX)
- Events and notifications
- Plugs into many frameworks to boost performance

Ideal for:

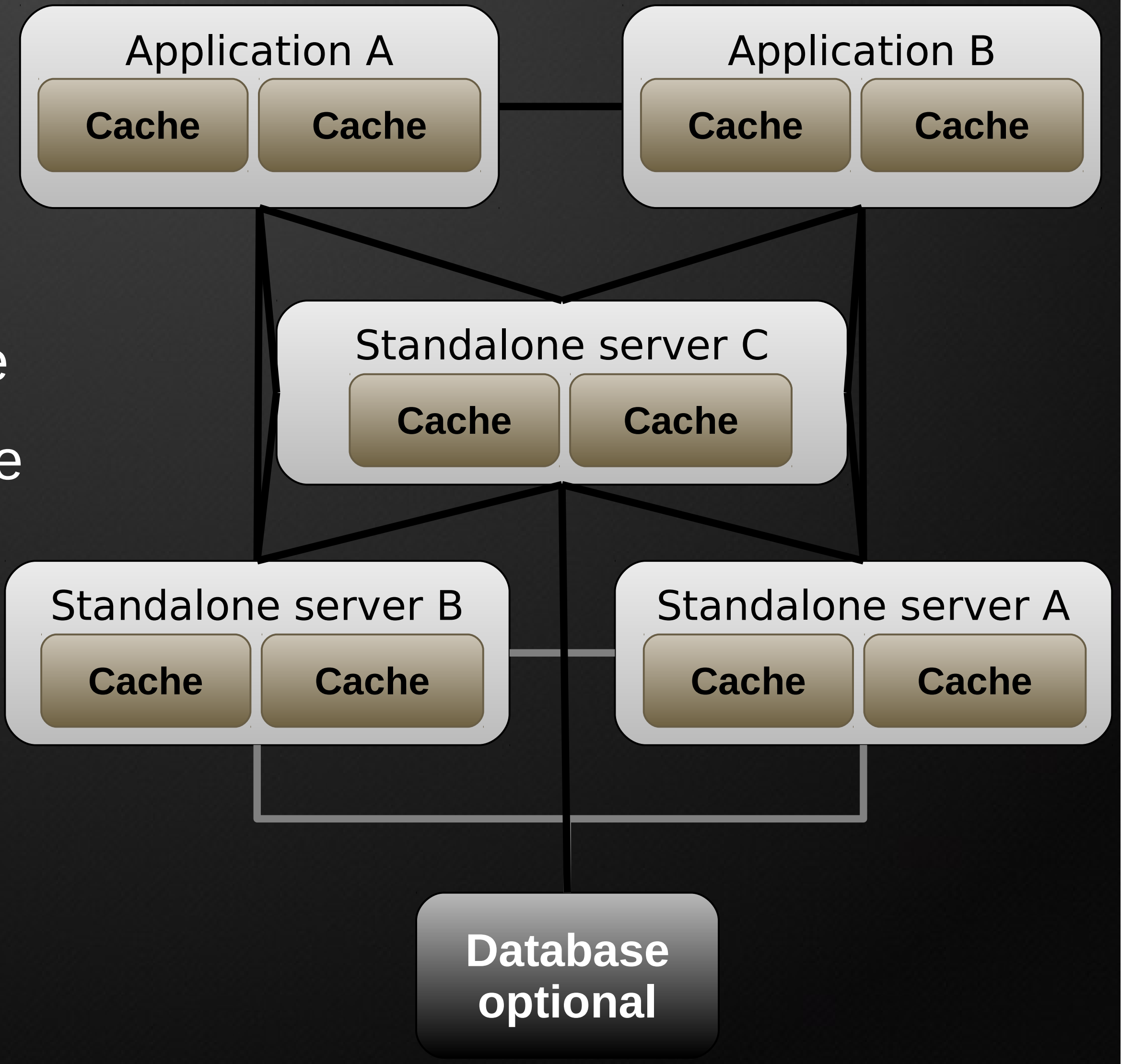
- *Single processes*
- *Data unique to a process*
- *Unshared data*



Use case – Data grid

Achieve massive elastic big data scale

- Distributed, horizontally scalable, unlimited storage
- Move processing to data with map and reduce
- Low-latency, fast performance
- Eliminate single point of failure
- Built on Red Hat-led JSR-347 (data grids) standards
- Multiple access protocols
- Compatible with applications written in any language, any framework



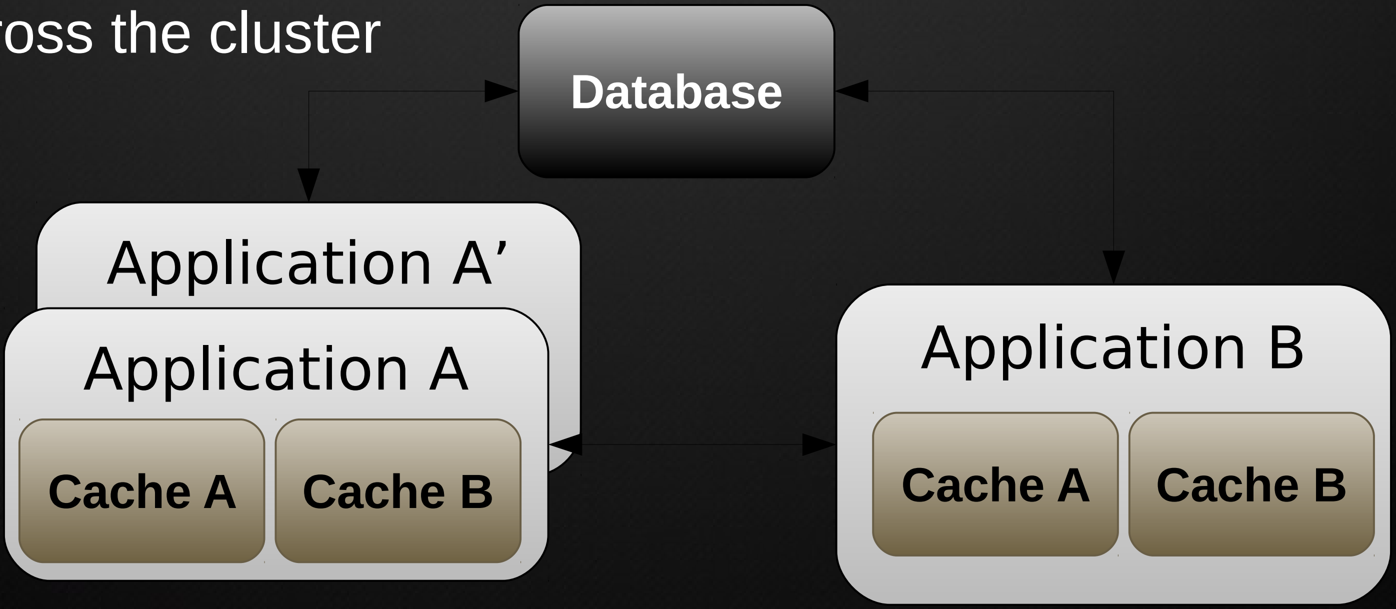
Use case - Replicated cache

Ultimate failover protection

- Instant reads, linear performance scalability
- Network overhead scales linearly
- Limited to a single JVM heap size
- Replicate the same key/value, updates across the cluster

Ideal for:

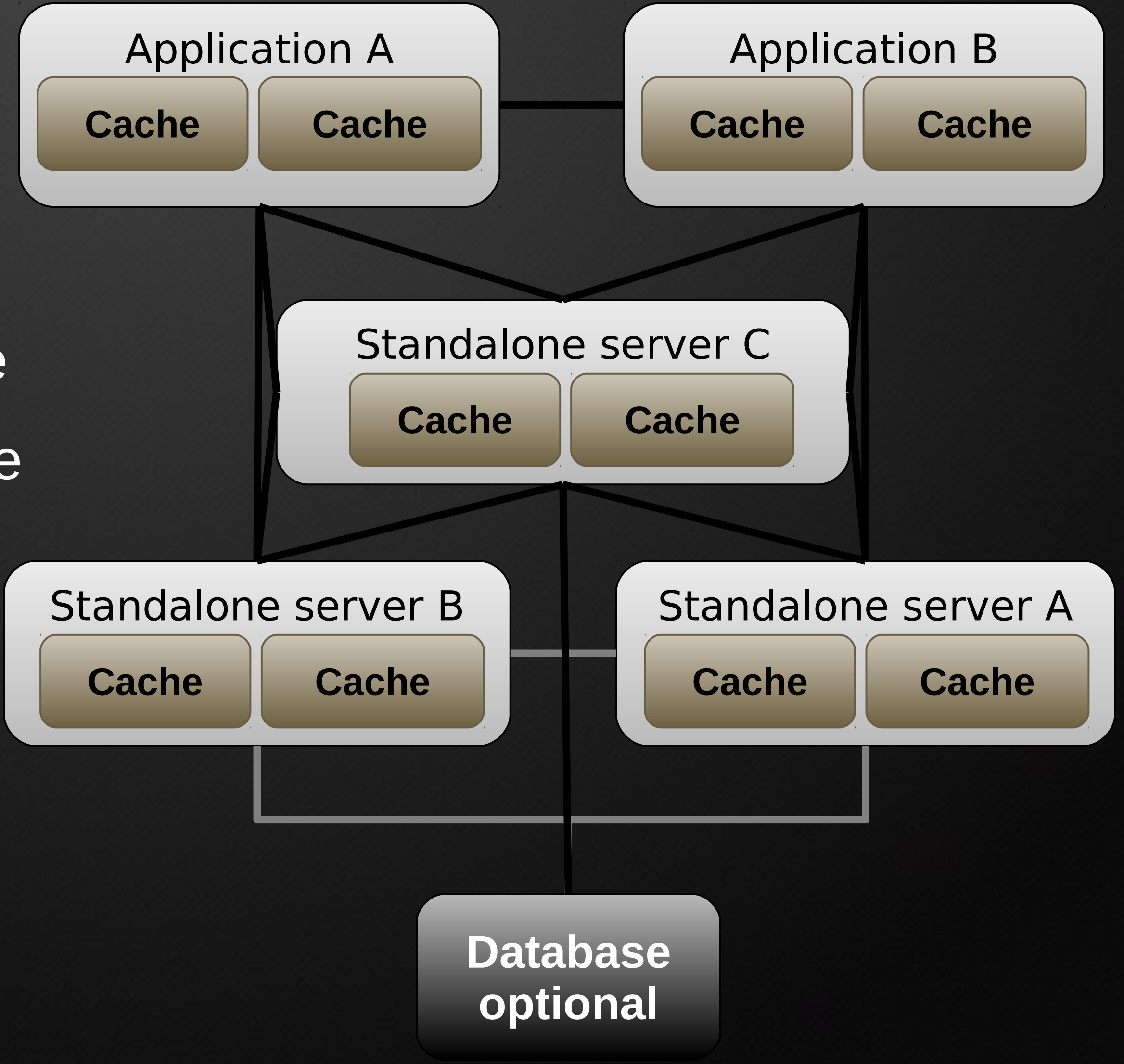
- *Small, fixed datasets*
- *Scenarios requiring extremely high fault tolerance*



Use case – Data grid

Achieve massive elastic big data scale

- Distributed, horizontally scalable, unlimited storage
- Move processing to data with map and reduce
- Low-latency, fast performance
- Eliminate single point of failure
- Built on Red Hat-led JSR-347 (data grids) standards
- Multiple access protocols
- Compatible with applications written in any language, any framework

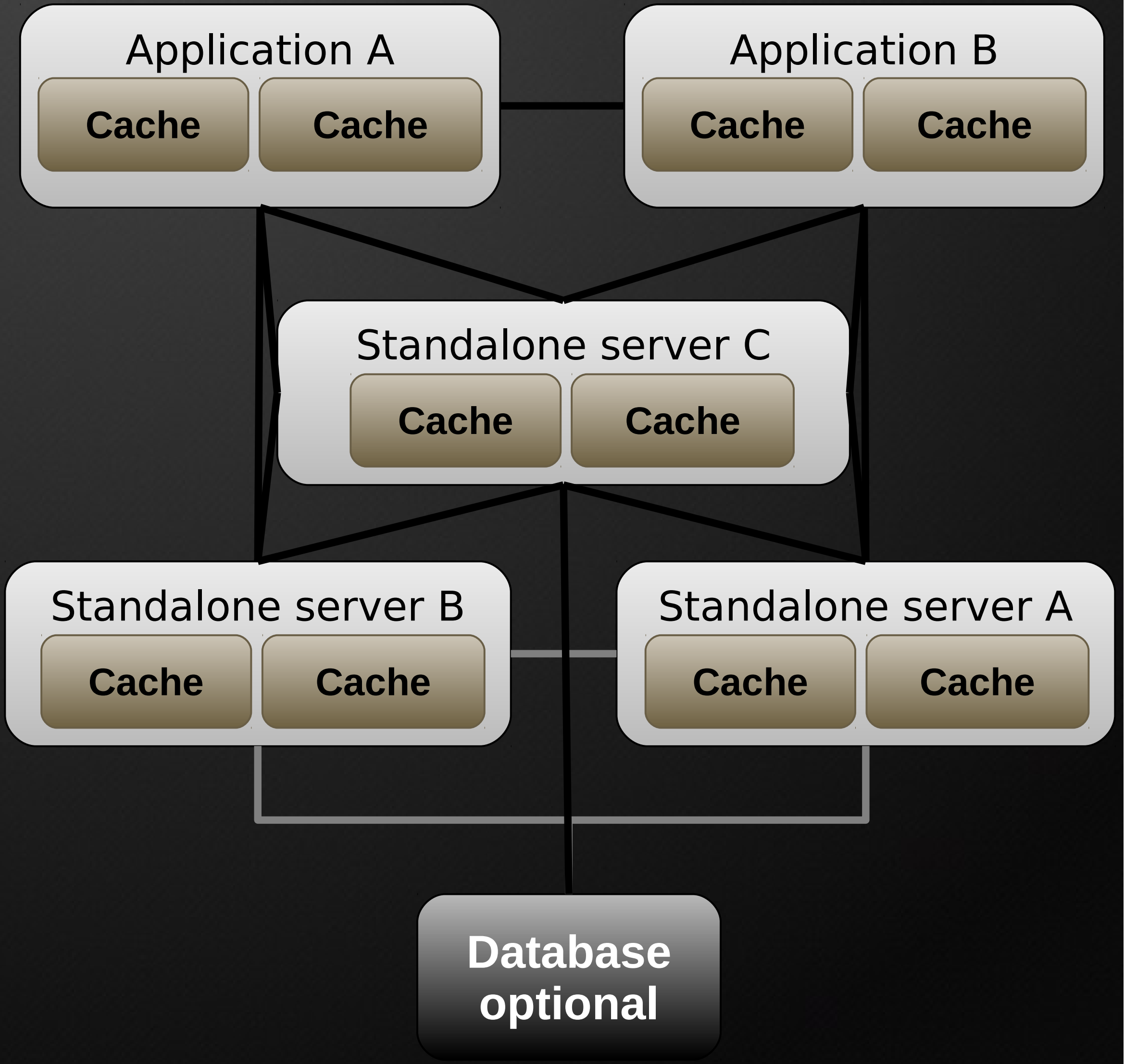


Use case – Data grid

Achieve massive elastic big data scale

Ideal for:

- *Massive distributed datasets like those from global, decentralized locations*
- *Elastic datasets that experience large fluctuations, periodicity, or unpredictability*
- *Transferring transaction loads away from local cache and traditional databases*

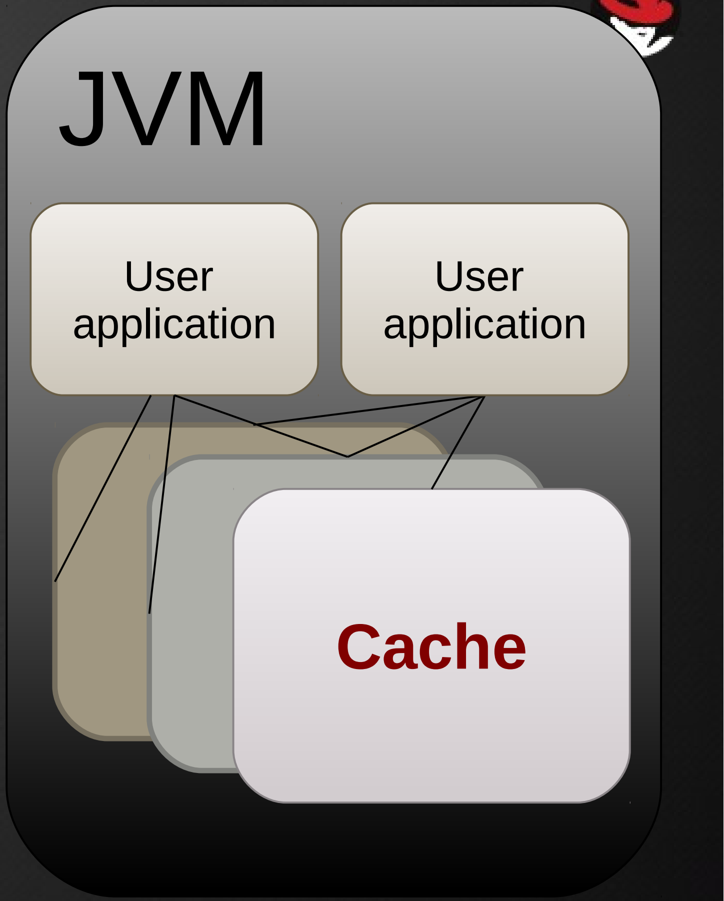


JBoss Data Grid: Deployment and use patterns

Deployment

Library mode

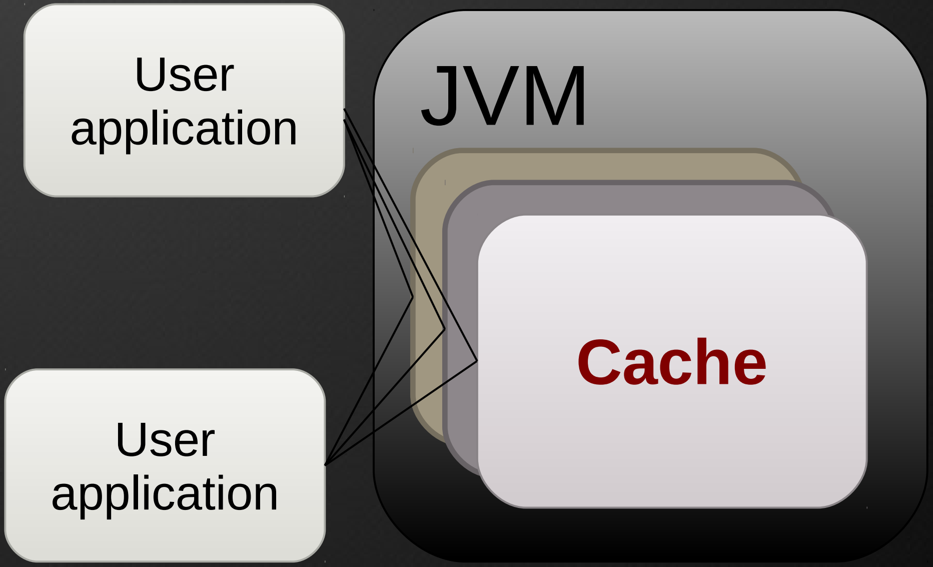
- “Bring your own” container
- Within one JVM:
  - Multiple caches
  - One node / cache
  - Multiple caches / application
- ‘Cache hit’ is in memory
- Memory management
- Transactions, monitoring, events, and notifications



Deployment

Client / Server stand-alone mode

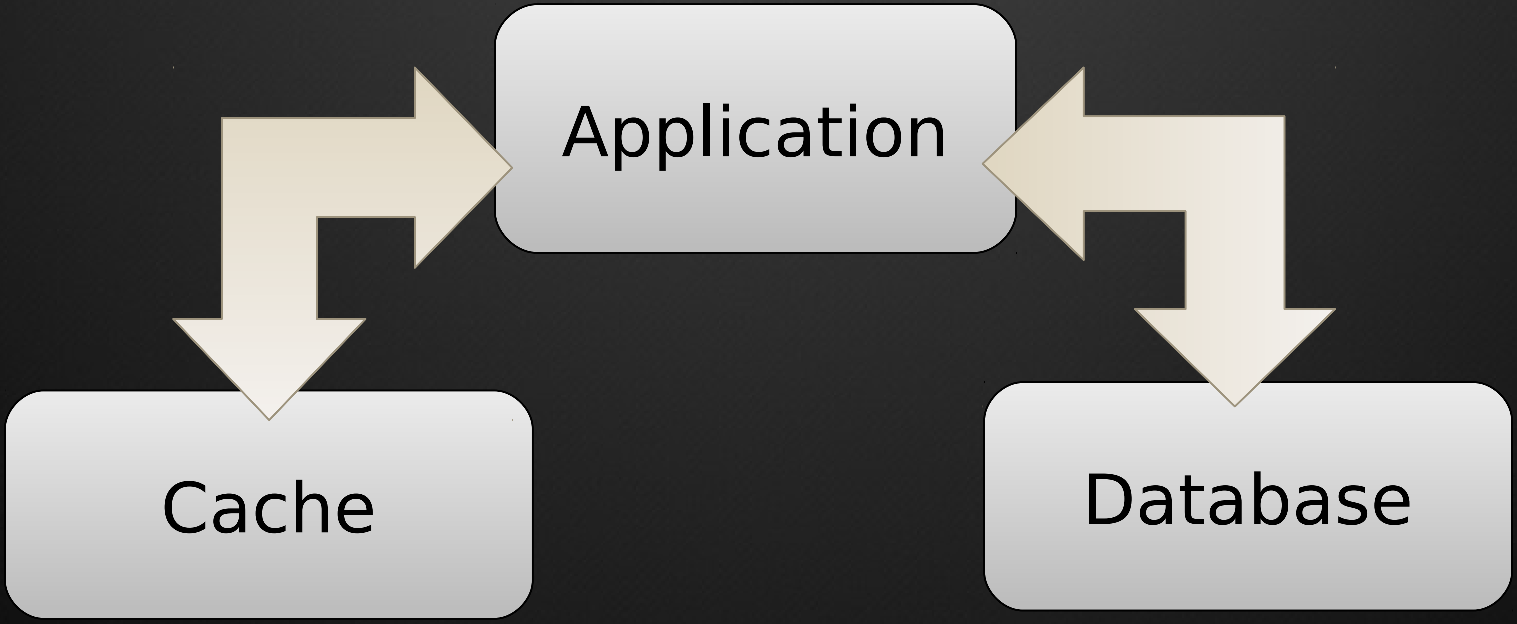
- “Remote” clients
- Within one service JVM
  - Multiple caches
  - One node / cache
  - Multiple caches / application
- Cache hit, not in local memory
- Compatibility - language agnostic
- Separate app and storage life cycles



Usage patterns

Side cache

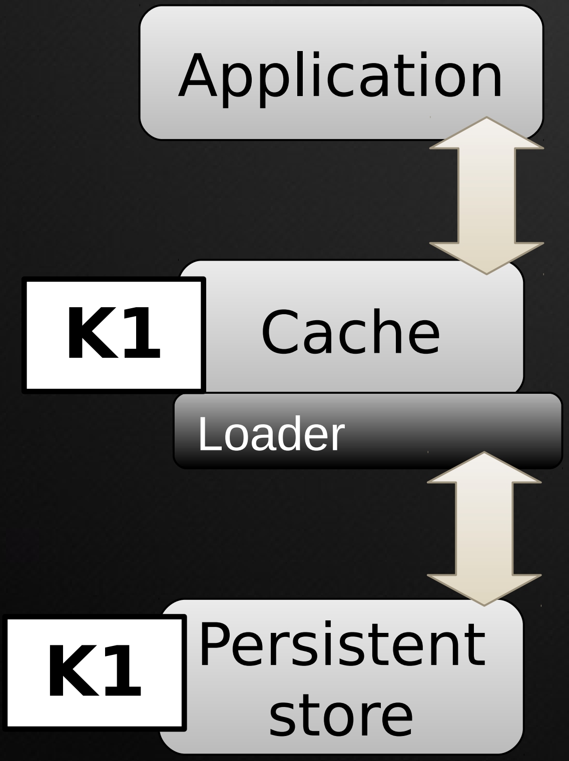
- Application manages cache



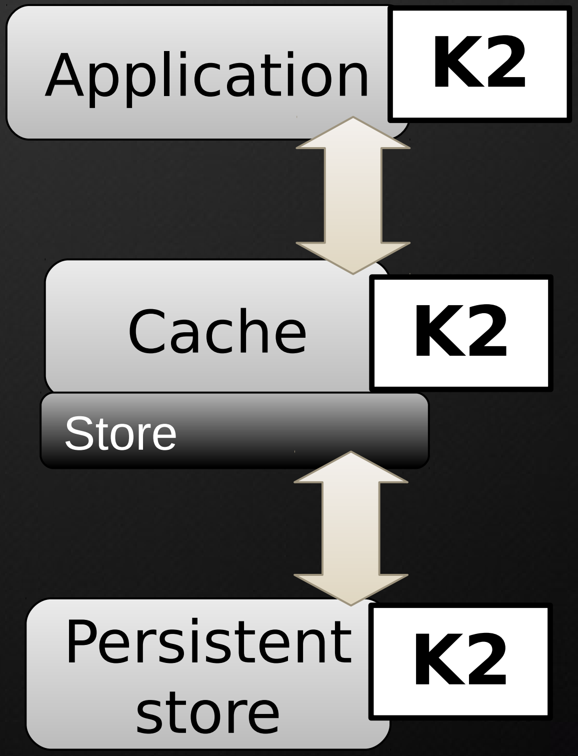
Usage patterns

Inline cache - Application speaks only to cache

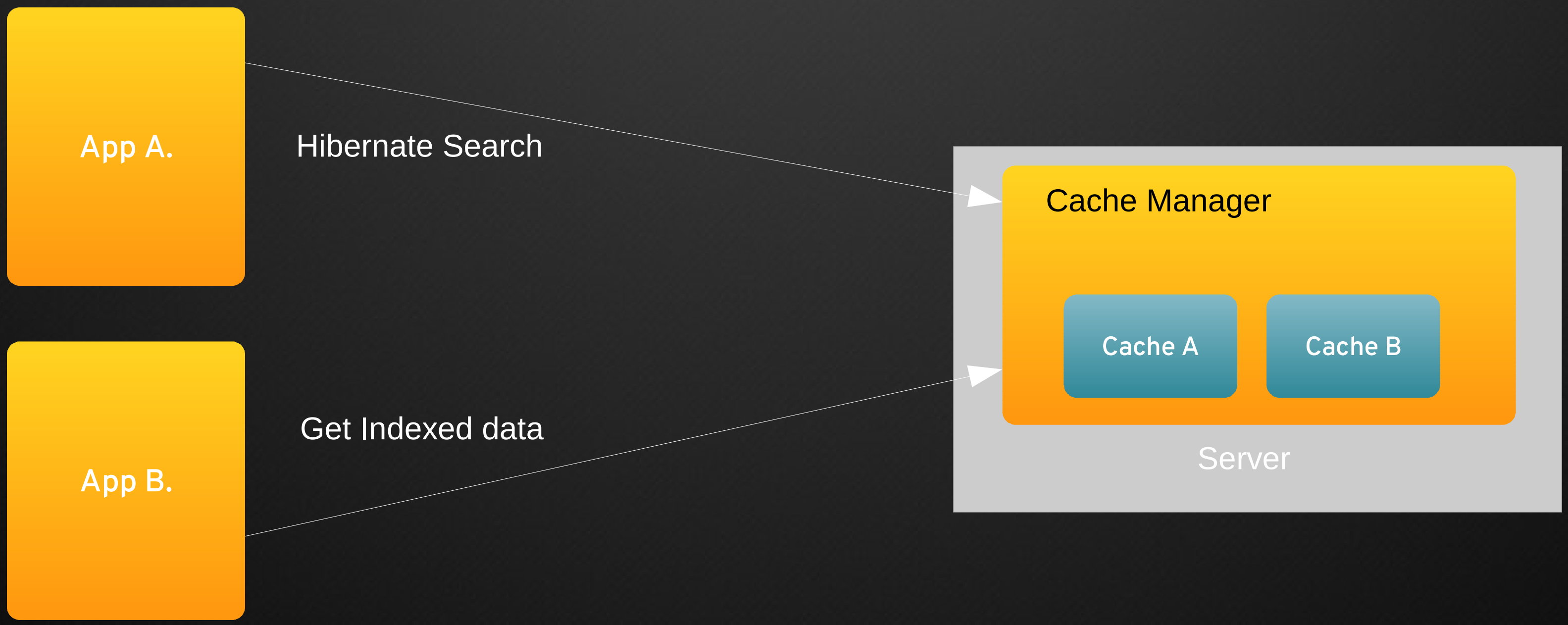
1) App requests data (K1)
2) Cache loader retrieves from persistent store (K1)



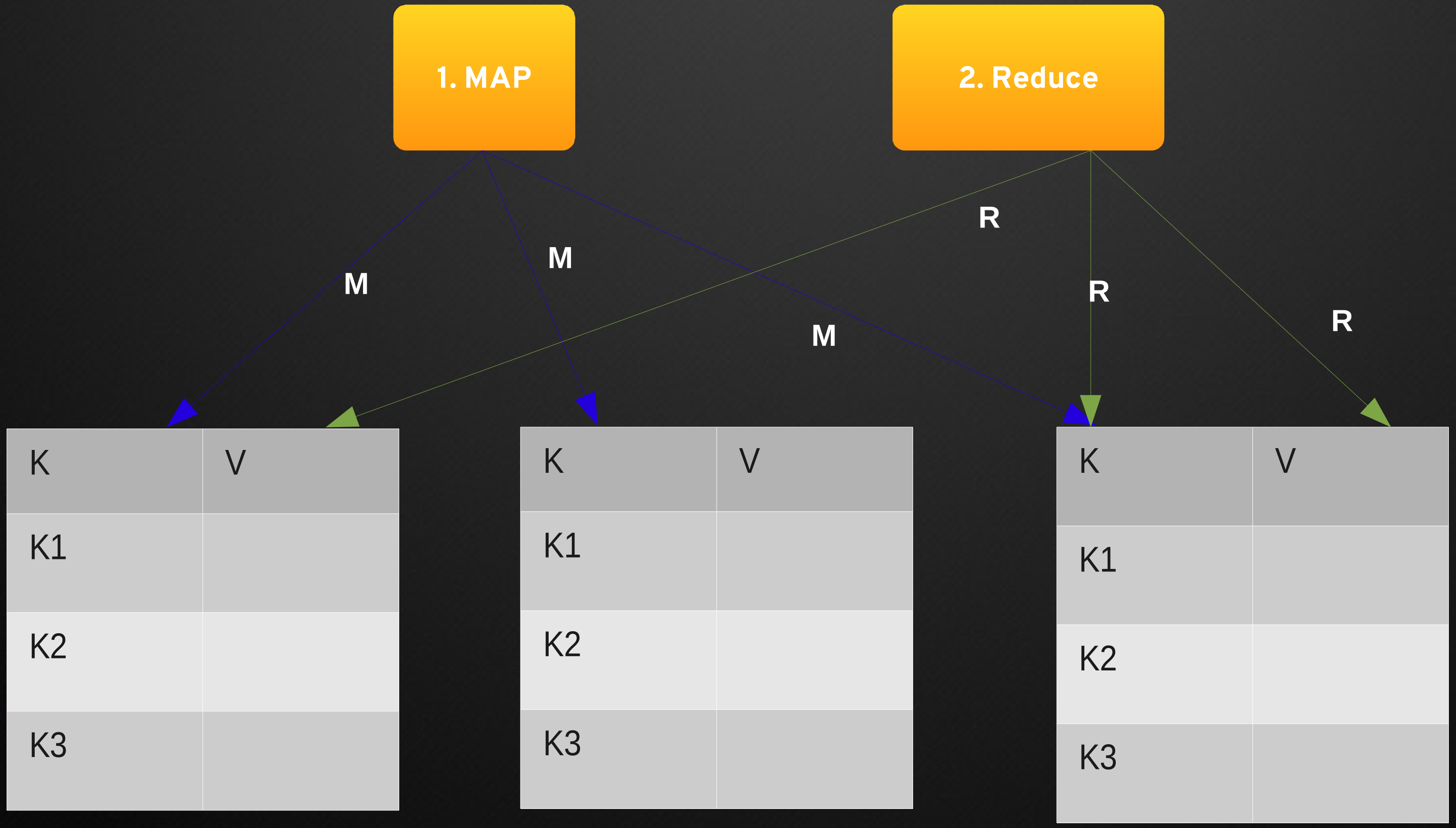
1) App writes data (K2)
2) Cache writes to persistent store (K2)



Searching/Indexing



Map/Reduce



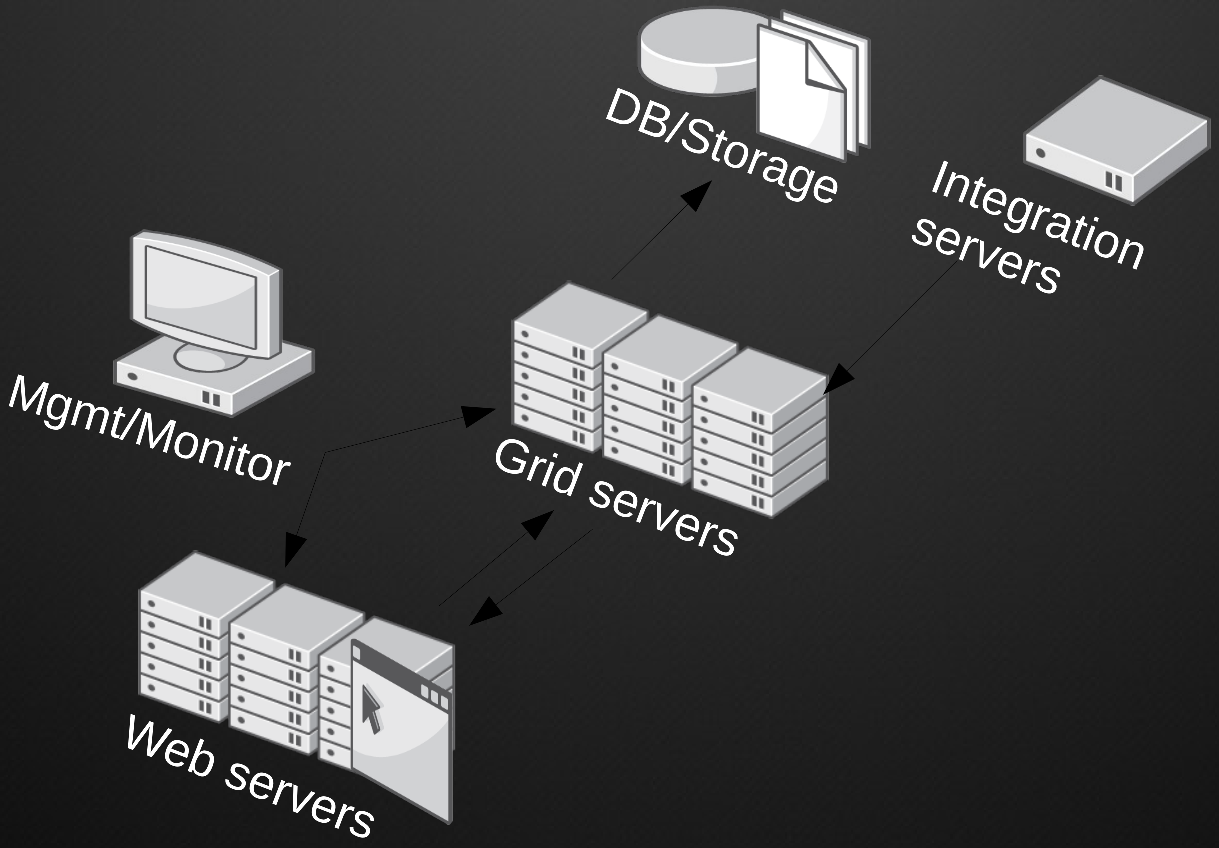
K	V
K1	
K2	
K3	

K	V
K1	
K2	
K3	

K	V
K1	
K2	
K3	

One Scenario

Data Replication and Cache



References

- Http://www.redhat.com
- Http://access.redhat.com
- Http://www.openshift.com
- Http://www.jboss.org/infinispan
- Http://www.jboss.org/jgroups