



Introduction to Infinispan

Tomáš Sýkora

Quality Assurance Associate, JBoss Data Grid
Red Hat

Contact: tsykora@redhat.com

IRC: #infinispan on freenode

PV243 Pokročilé Java technologie: JBoss, 2013 @ FIMU

April 8th 2013

Agenda

- What's Infinispan
- Why / When to use it
- High level features
- How to plug it into your architecture
- Clustering modes
- Client / server access modes
- Brand new features in version 5.2



What's Infinispan ?

- Open-source datagrid platform
- In-Memory, Schema-less, NoSQL key-value data store
- Distributed cache (offers massive heap)
- Scalable (goal: hundreds of nodes) + Elastic
- Highly available, resilient to node failures
- Concurrent
- Transactional
- Queryable

Red Hat Productized version: **JBoss Data Grid** (6.1 GA released March 2013)



For Java users: it's a Map

```
DefaultCacheManager cacheManager = new DefaultCacheManager("infinispan.xml");
```

```
Cache<String, Object> cache = cacheManager.getCache("namedCache");
```

```
cache.put("key", "value");
```

```
Object value = cache.get("key");
```

org.infinispan.Cache extends java.util.Map



Configuration in XML

```
<?xml version="1.0" encoding="UTF-8"?>  
<infinispan/>
```

Read more:

<https://docs.jboss.org/author/display/ISPN/Configuring+Cache+declaratively>



Programmatic Configuration

Configuration c =

```
new ConfigurationBuilder()  
    .clustering().cacheMode(CacheMode.REPL_SYNC)  
    .build();
```

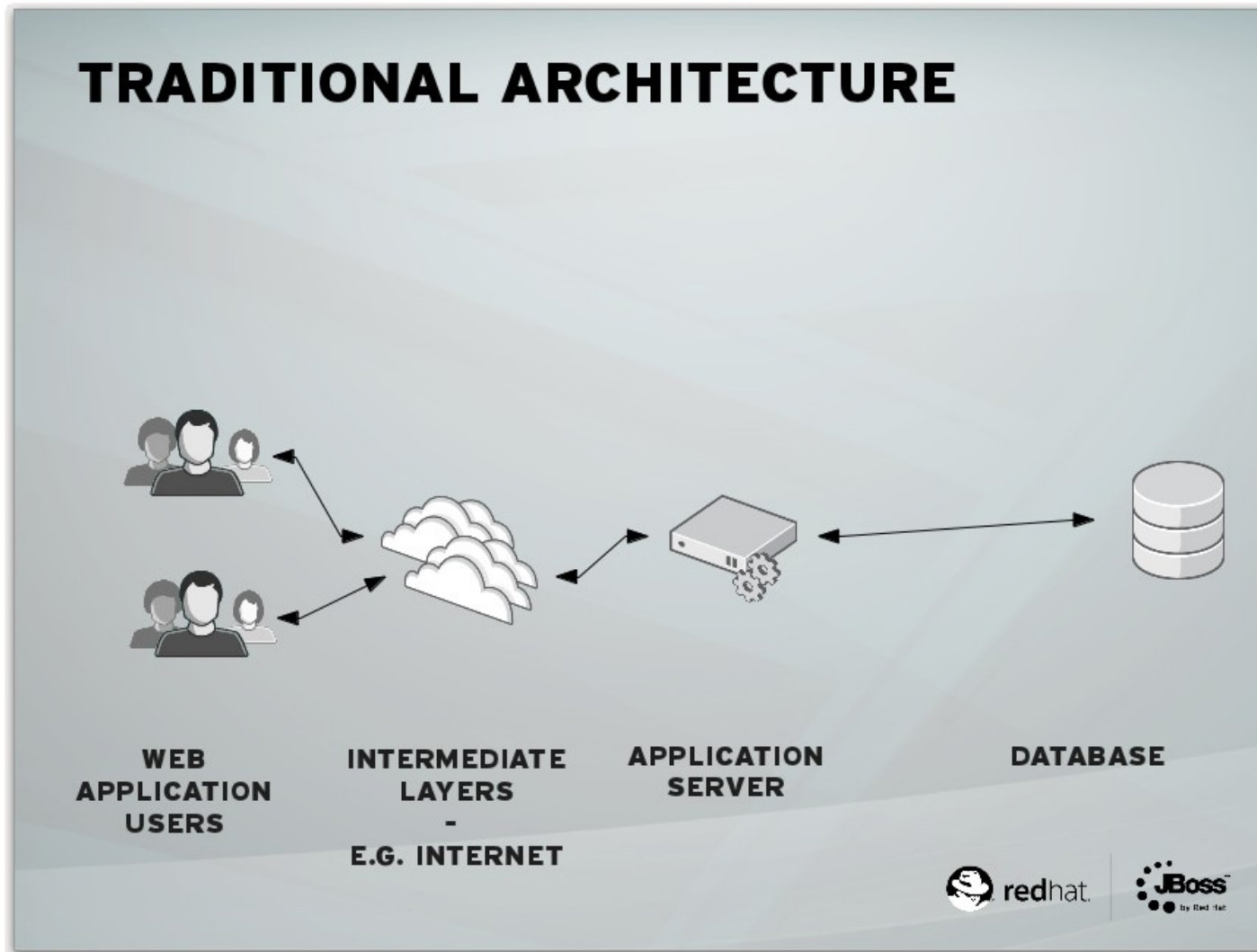
```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder()  
    .transport()  
    .clusterName("qa-cluster")  
    .addProperty("configurationFile", "jgroups-tcp.xml")  
    .machineId("qa-machine").rackId("qa-rack")  
    .build();
```

Read more:

<https://docs.jboss.org/author/display/ISPN/Configuring+cache+programmatically>



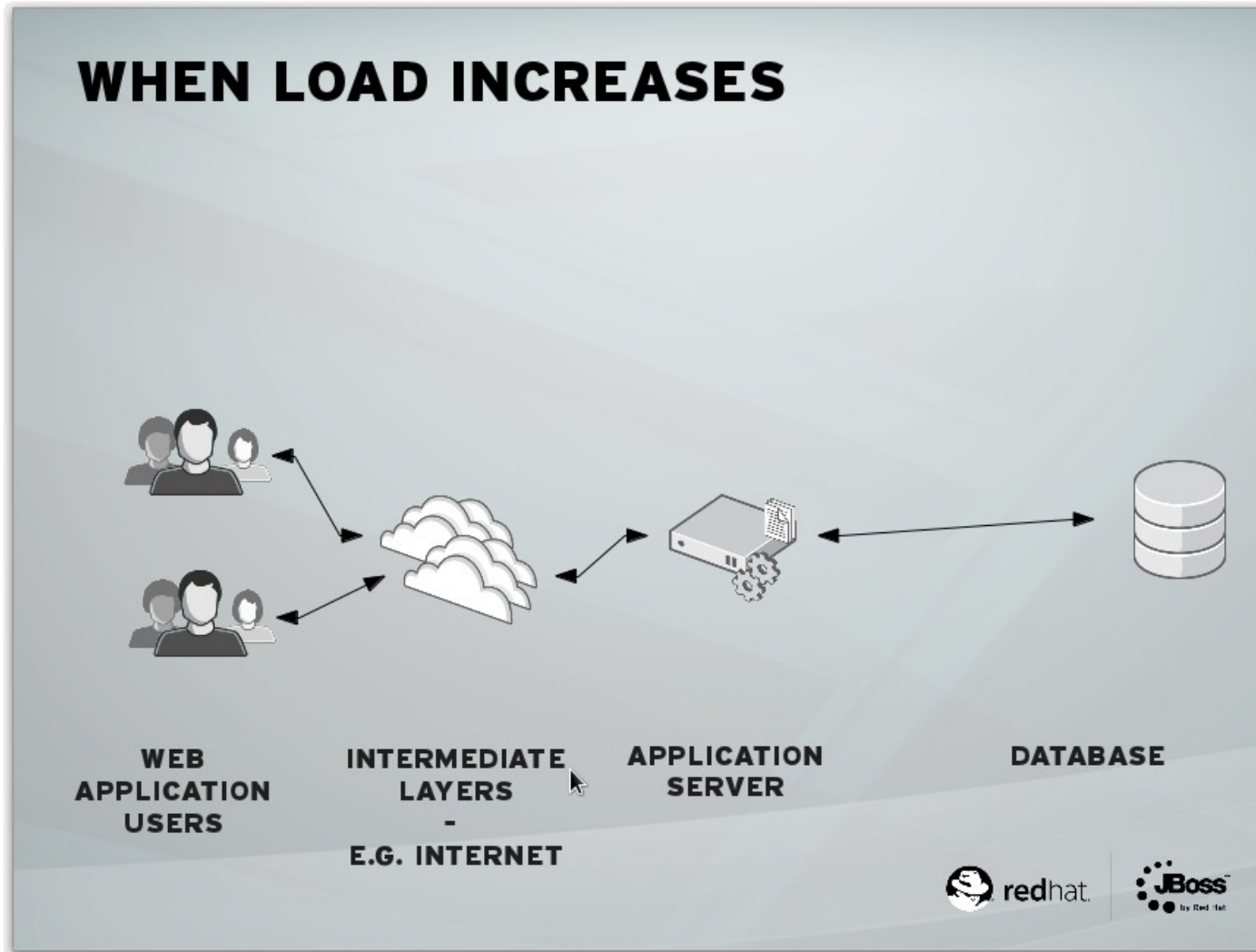
Why Datagrid ?



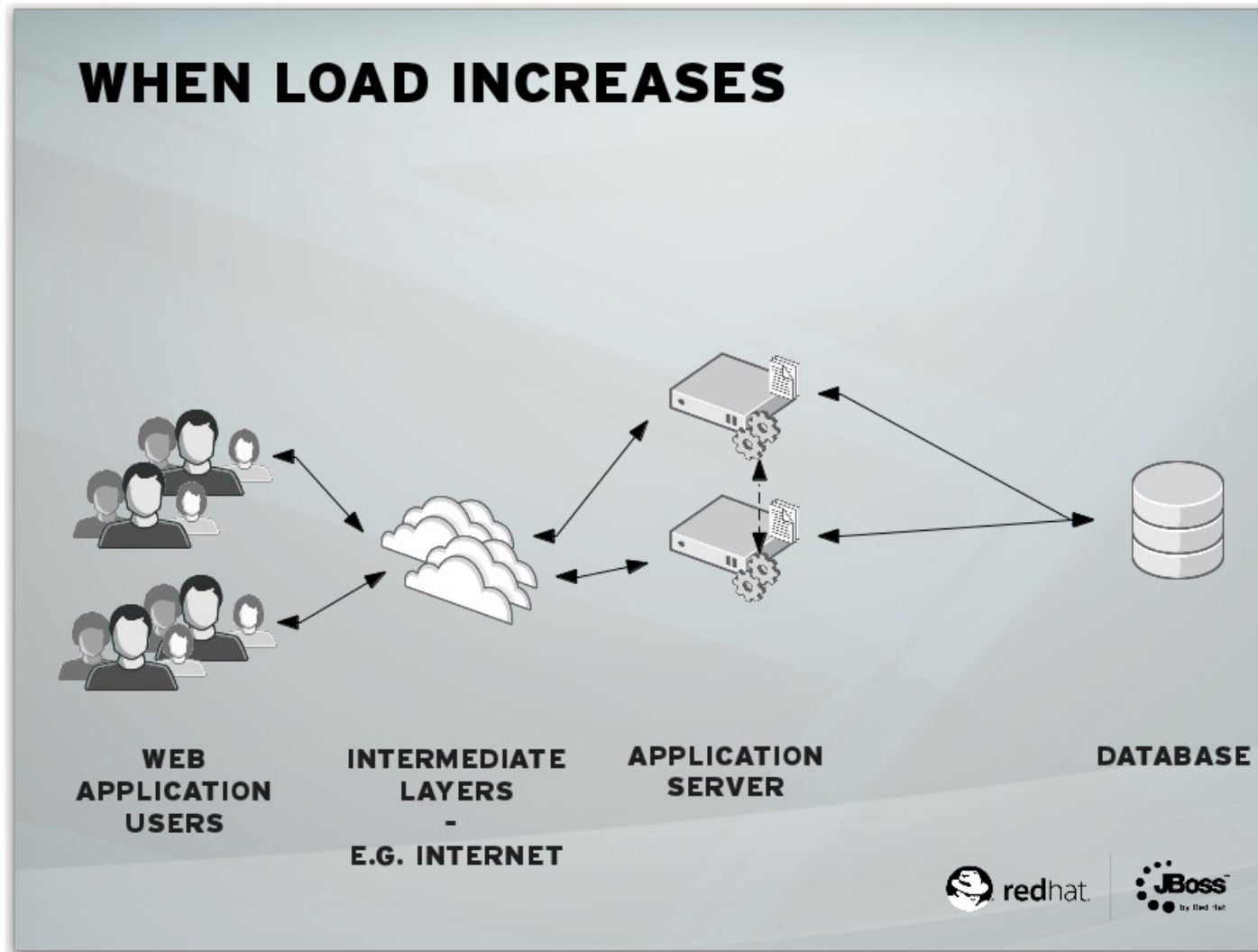
From <http://fhornain.wordpress.com/2012/04/21/jboss-data-grid-when-database-is-very-expensive/>



Why Datagrid ?



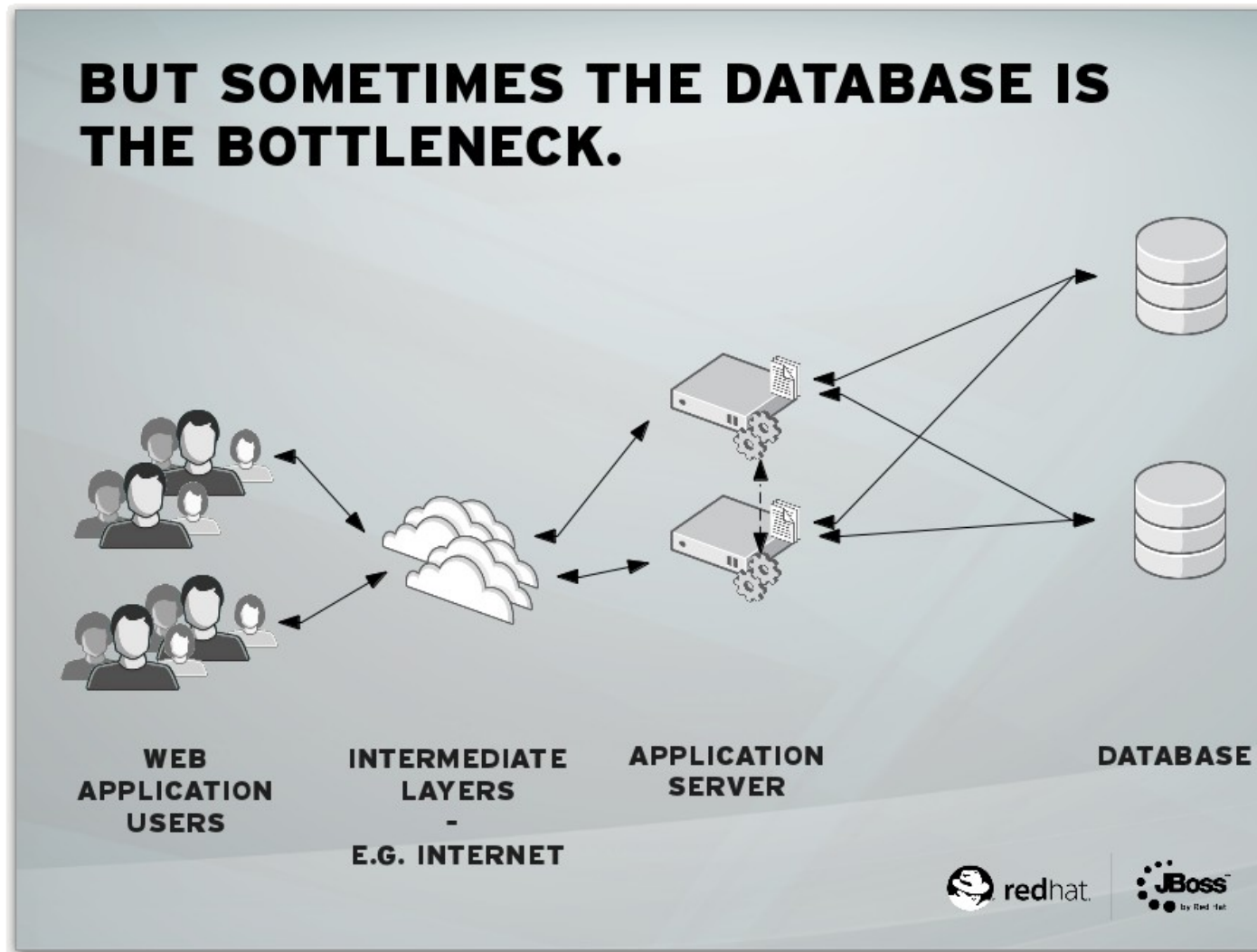
Why Datagrid ?



From <http://fhornain.wordpress.com/2012/04/21/jboss-data-grid-when-database-is-very-expensive/>



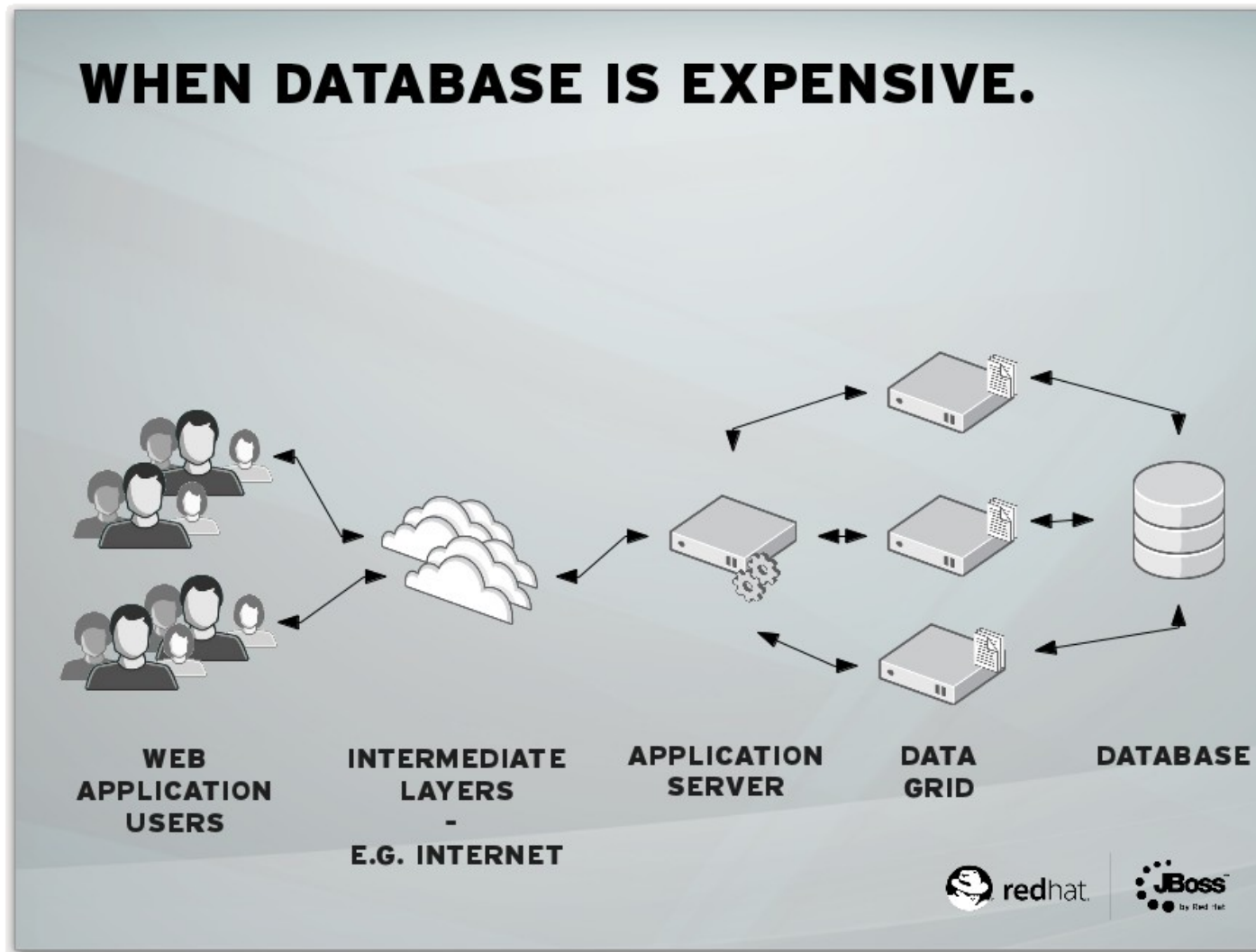
Why Datagrid ?



From <http://fhornain.wordpress.com/2012/04/21/jboss-data-grid-when-database-is-very-expensive/>



Why Datagrid ?



From <http://fhornain.wordpress.com/2012/04/21/jboss-data-grid-when-database-is-very-expensive/>



Features - Querying

// example values stored in the cache and indexed:

```
import org.hibernate.search.annotations.*;
```

//to be indexed the object needs both @Indexed and @ProvidedId annotations:

```
@Indexed @ProvidedId
```

```
public class Book {
```

```
    @Field String title;
```

```
    @Field String description;
```

```
    @Field @DateBridge(resolution=Resolution.YEAR) Date publicationYear;
```

```
    @IndexedEmbedded Set<Author> authors = new HashSet<Author>();
```

```
}
```

```
public class Author {
```

```
    @Field String name;
```

```
    @Field String surname;
```

```
    // hashCode() and equals() omitted
```

```
}
```

Read more: <https://docs.jboss.org/author/display/ISPN/Querying+Infinispan>



Features - Querying

// get search manager from cache

```
SearchManager searchManager = org.infinispan.query.Search.getSearchManager( cache );
```

```
QueryBuilder queryBuilder = searchManager.buildQueryBuilderForClass( Book.class ).get();
```

// the queryBuilder has a nice fluent API which guides you through all options

```
org.apache.lucene.search.Query luceneQuery = queryBuilder.phrase()  
    .onField( "description" )  
    .andField( "title" )  
    .sentence( "a book on highly scalable query engines" )  
    .createQuery();
```

// the query API itself accepts any Lucene Query, and on top of that

// you can restrict the result to selected class types:

```
CacheQuery query = searchManager.getQuery( luceneQuery, Book.class );
```

// and there are your results!

```
List<Book> objectList = query.list();
```

```
for ( Book book : objectList ) {  
    System.out.println( book.getTitle() );  
}
```

Read more: <https://docs.jboss.org/author/display/ISPN/Querying+Infinispan>



Features - Transactions

- Each cache is either
 - TRANSACTIONAL
 - or NON_TRANSACTIONAL (default)
- Transactional cache has two possible locking modes
 - OPTIMISTIC (default)
 - PESSIMISTIC
- Two isolation modes available
 - READ_COMMITTED (default)
 - REPEATABLE_READ

Read more: <https://docs.jboss.org/author/display/ISPN/Infinispan+transactions>



Features - Transactions

- JTA Transactions – to configure specify TransactionManagerLookup
 - In **JavaSE apps**: JBossStandaloneJTAManagerLookup – uses **JBoss Transactions**
 - In **JEE apps**: GenericTransactionManagerLookup – works with most popular containers
 - In **JBoss AS**: JBossTransactionManagerLookup

```
<transaction
  transactionManagerLookupClass=
    "org.infinispan.transaction.lookup.GenericTransactionManagerLookup"
  transactionMode="TRANSACTIONAL"
  lockingMode="OPTIMISTIC" />
```

Read more: <https://docs.jboss.org/author/display/ISPN/Infinispan+transactions>



Features - Transactions

```
Cache cache = cacheManager.getCache();
```

```
TransactionManager transactionManager =  
    cache.getAdvancedCache().getTransactionManager();
```

```
transactionManager.begin();  
cache.put(k1,v1);  
cache.remove(k2);  
transactionManager.commit();
```

Read more: <https://docs.jboss.org/author/display/ISPN/Infinispan+transactions>



Features - Transactions

- Explicit locking
- Deadlock detection
- Transaction recovery
- Distributed transactions

Read more: <https://docs.jboss.org/author/display/ISPN/Infinispan+transactions>



Features – Eviction

- Specify maximal number of entries to keep in cache
- Heap-load based eviction (being worked on)
- Eviction strategies
 - UNORDERED
 - FIFO
 - LRU – Least recently used
 - LIRS - Low Inter-reference Recency Set
S.Jiang and X.Zhang's 2002 paper: LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance

Read more: <https://docs.jboss.org/author/display/ISPN/Eviction>



Features – Expiration

- Specify maximal time entries are allowed
 - stay in cache (lifespan)
 - stay in cache untouched (maxIdle)
- Default expiration – specify in cache config
- Explicitly set lifespan or maxIdle with every PUT

```
cache.put("Grandma", "I'll stay only a minute", 1, TimeUnit.MINUTES);
```

```
cache.put("Tamagochi", "Watch me or I'll die", -1, TimeUnit.SECONDS,  
1, TimeUnit.SECONDS);
```

Read more: <https://docs.jboss.org/author/display/ISPN/Eviction>



Features – Cache stores

- Store data from memory to other kind of storage
 - File System
 - FileCacheStore – basic FS store implementation
 - BerkeleyDB JavaEdition
 - JBDM
 - Relational Database
 - JdbcBinaryCacheStore – hash of whatever
 - JdbcStringBasedCacheStore – String (needs mapping)
 - Other NoSQL stores
 - Cassandra
 - JClouds BlobStore
 - RemoteCacheStore – store to another Infinispan grid



Features – Cache stores

Passivation	Eviction	Behaviour
OFF	OFF	P = M (Write through) whenever an element is modified, added or removed, then that modification is persisted in the backend store
OFF	ON	P \supseteq M (Write through) P includes all entries while M may contain fewer entries (some of them might have been evicted)
ON	OFF	This is an invalid configuration and Infinispan logs a warning
ON	ON	P \cap M = \emptyset Writes to the persistent store via the cache store only occur as part of the eviction process. Data is deleted from the persistent store when read back into memory.

P = set of keys kept in persisted storage

M = set of keys kept in memory

Passivation is a mode of storing entries in the cache store only when they are evicted from memory.



Features – Others

- Management via RHQ (<http://rhq-project.org>)
- JMX Statistics
- CDI, injection of Cache, RemoteCache
- Tree API (storing in hierarchical way, /persons/john)
- ... and more on next slides



How to plug it into your architecture ?

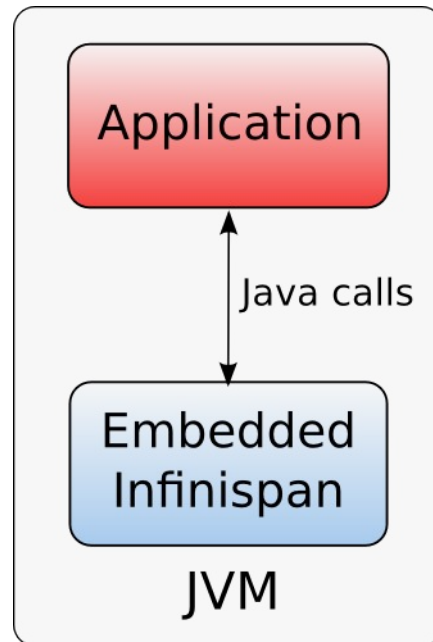


Modes of access / usage

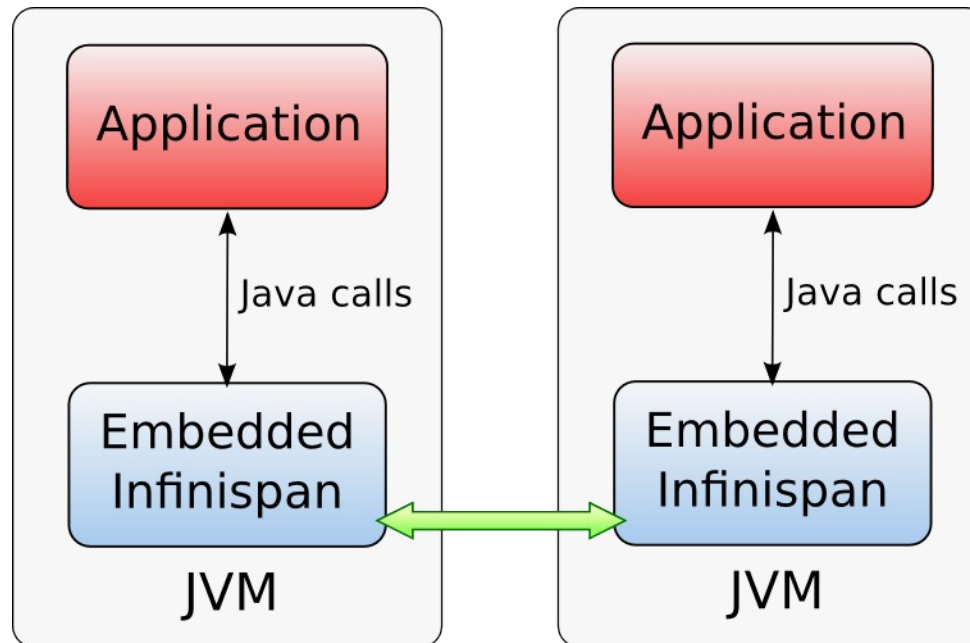
- Embedded (In-VM)
- Remote (Client/Server)
 - REST (HTTP)
 - Memcached
 - Hot Rod



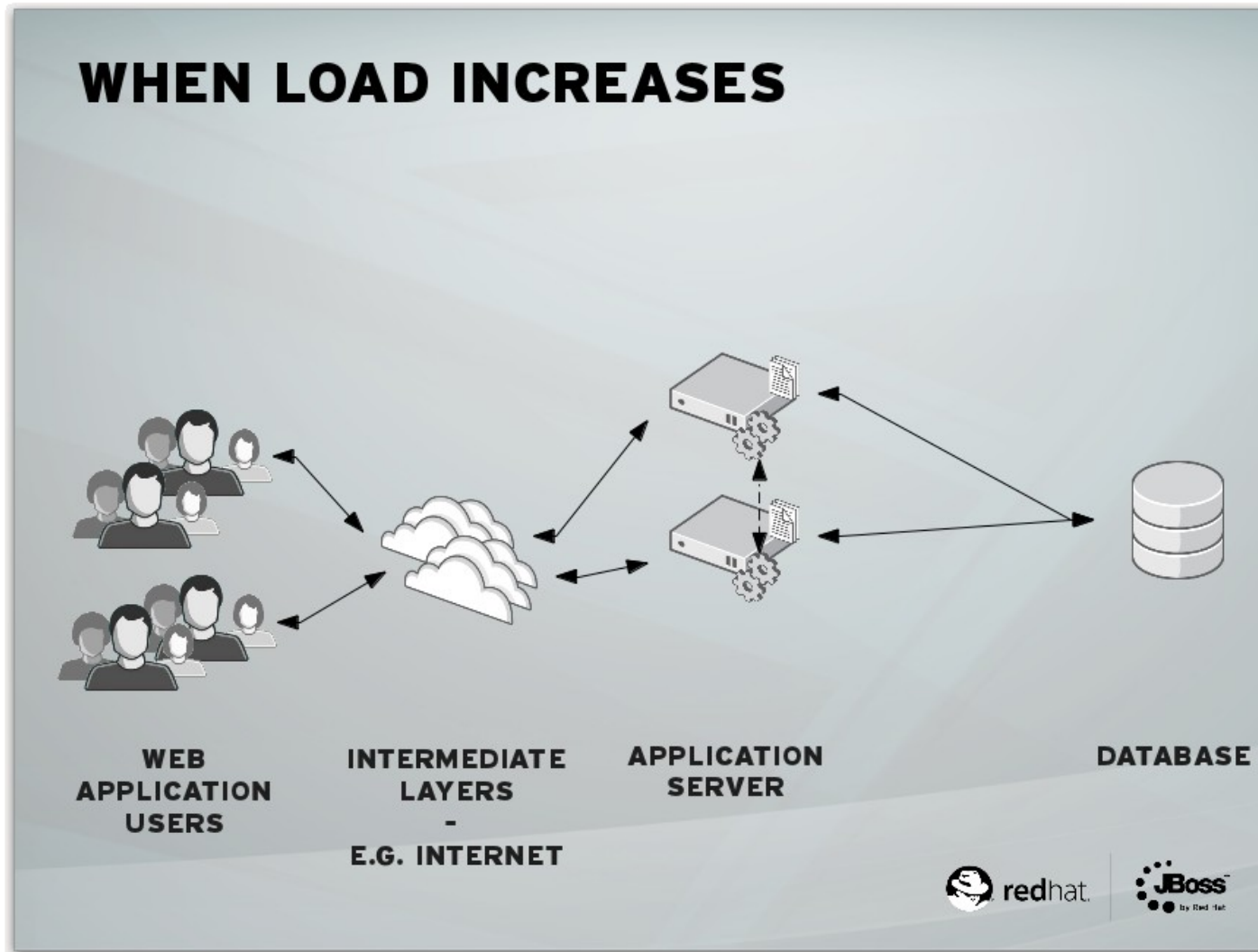
Embedded (In-VM) mode



Embedded (In-VM) mode - clustered



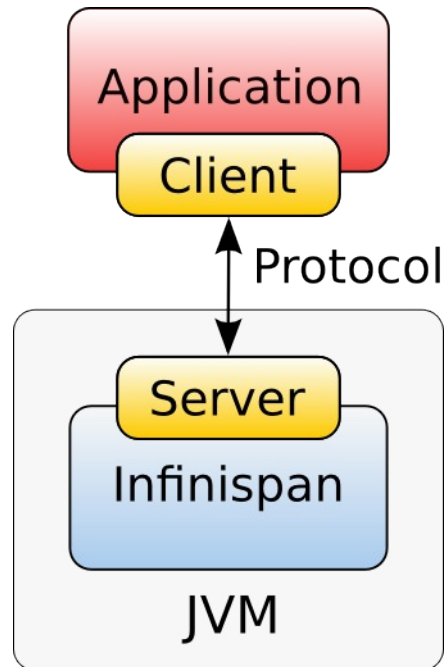
Embedded (In-VM) mode - clustered



From <http://fhornain.wordpress.com/2012/04/21/jboss-data-grid-when-database-is-very-expensive/>



Client / Server mode

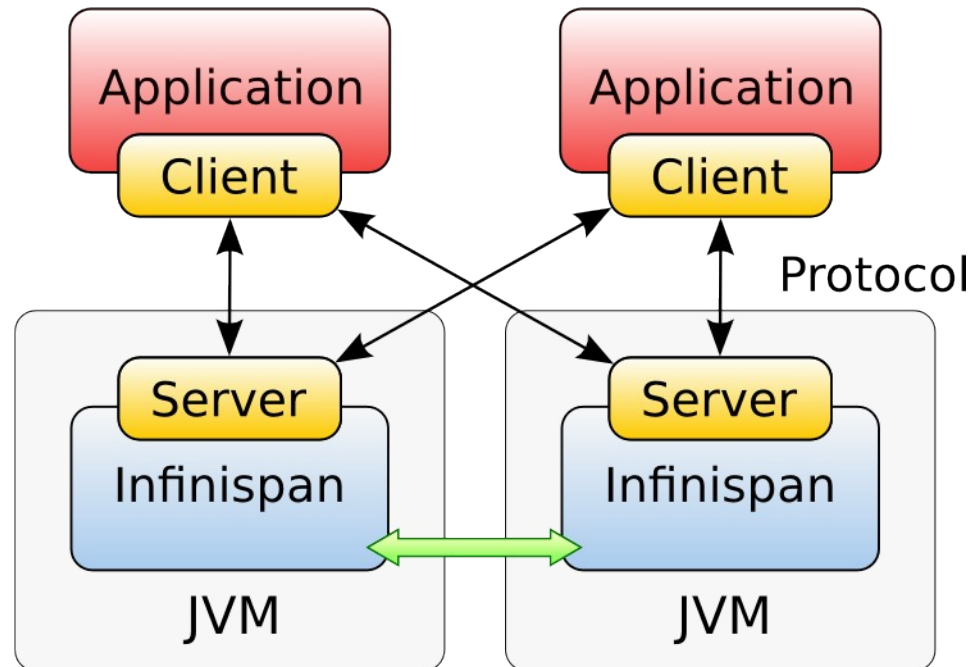


Protocols

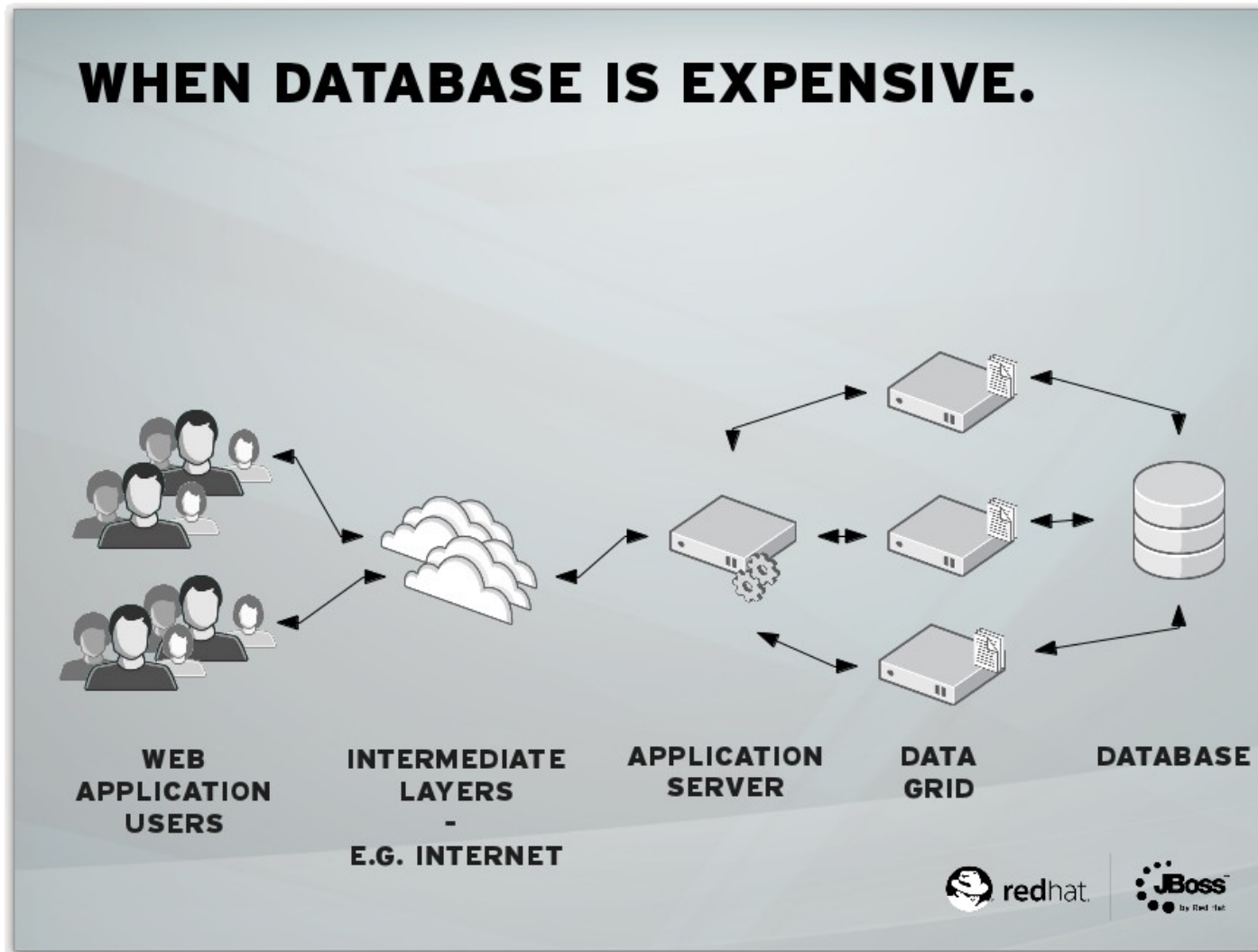
- REST
- Memcached
- Hot Rod



Client / Server mode - clustered



Client / Server mode - clustered

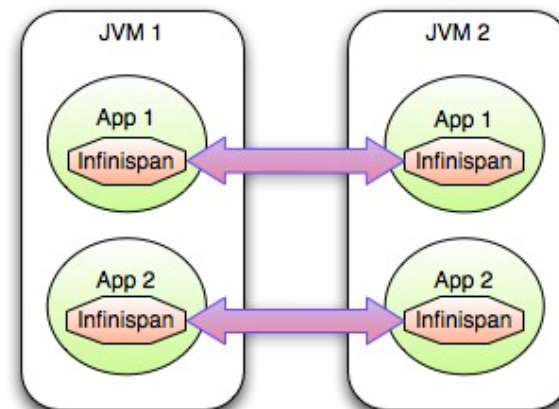
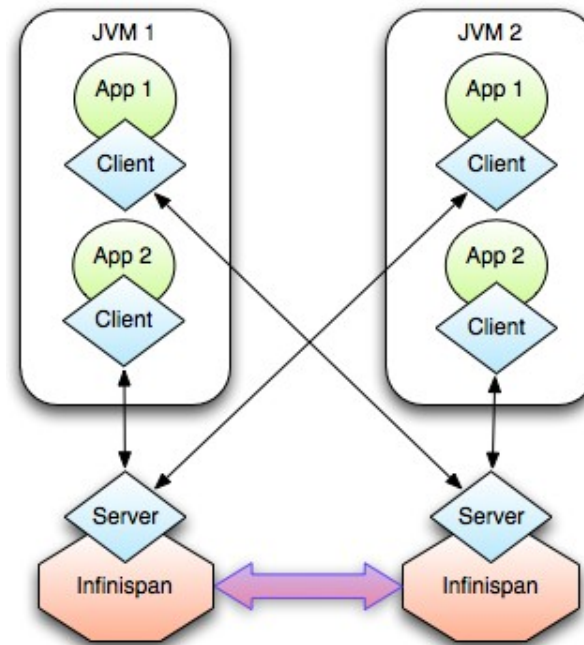


From <http://fhornain.wordpress.com/2012/04/21/jboss-data-grid-when-database-is-very-expensive/>

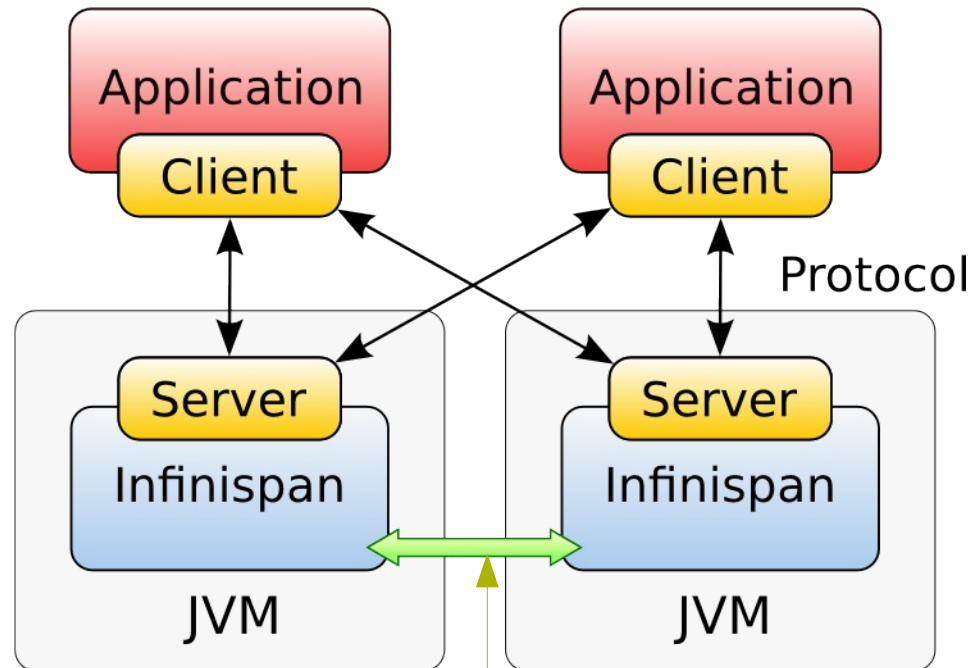


Client / Server mode - clustered

- Independent tier management
- Independently deploy new app version
- Security



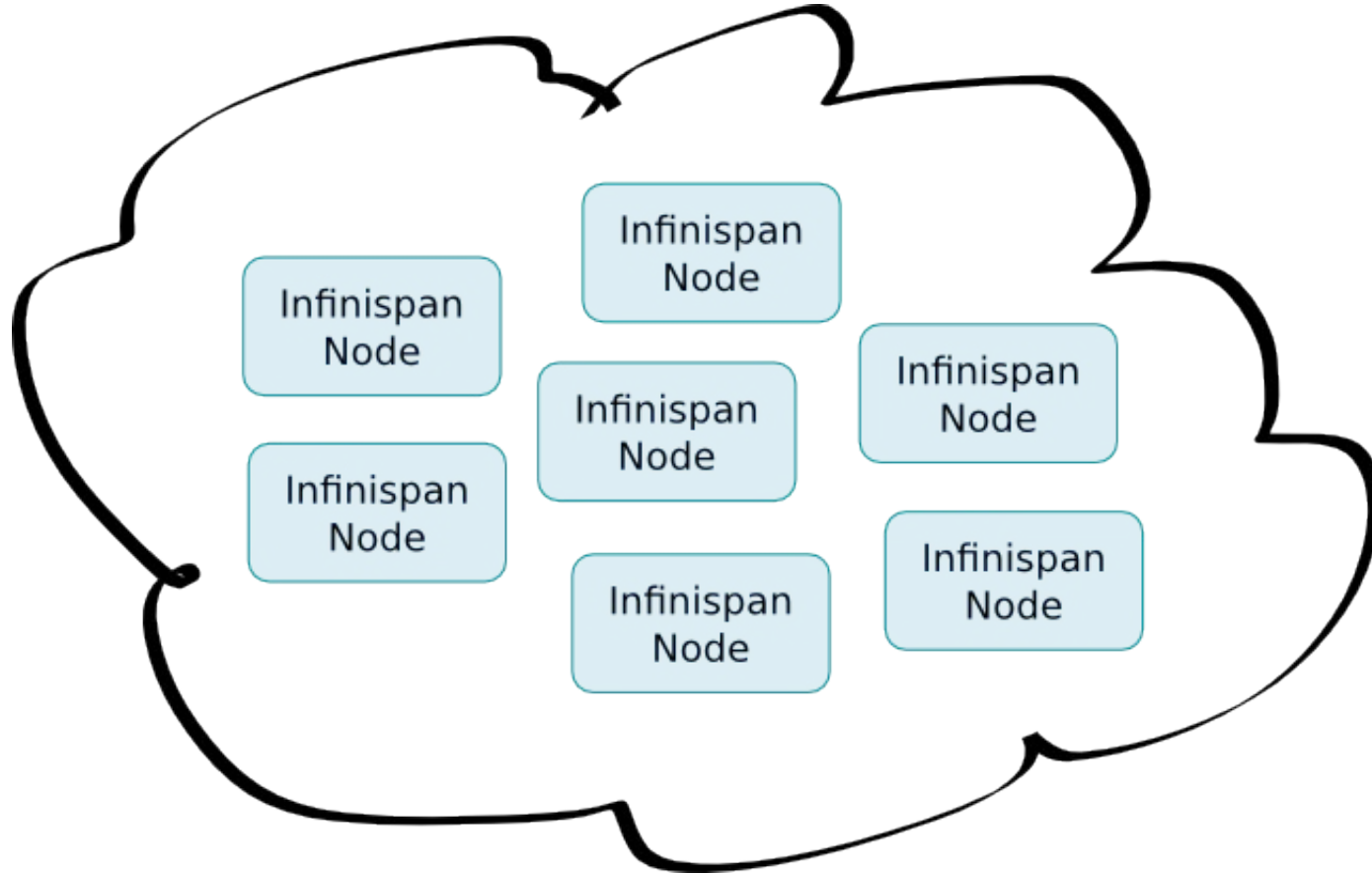
Client / Server mode - clustered



Big deal!



What clustering / resilience / elasticity means

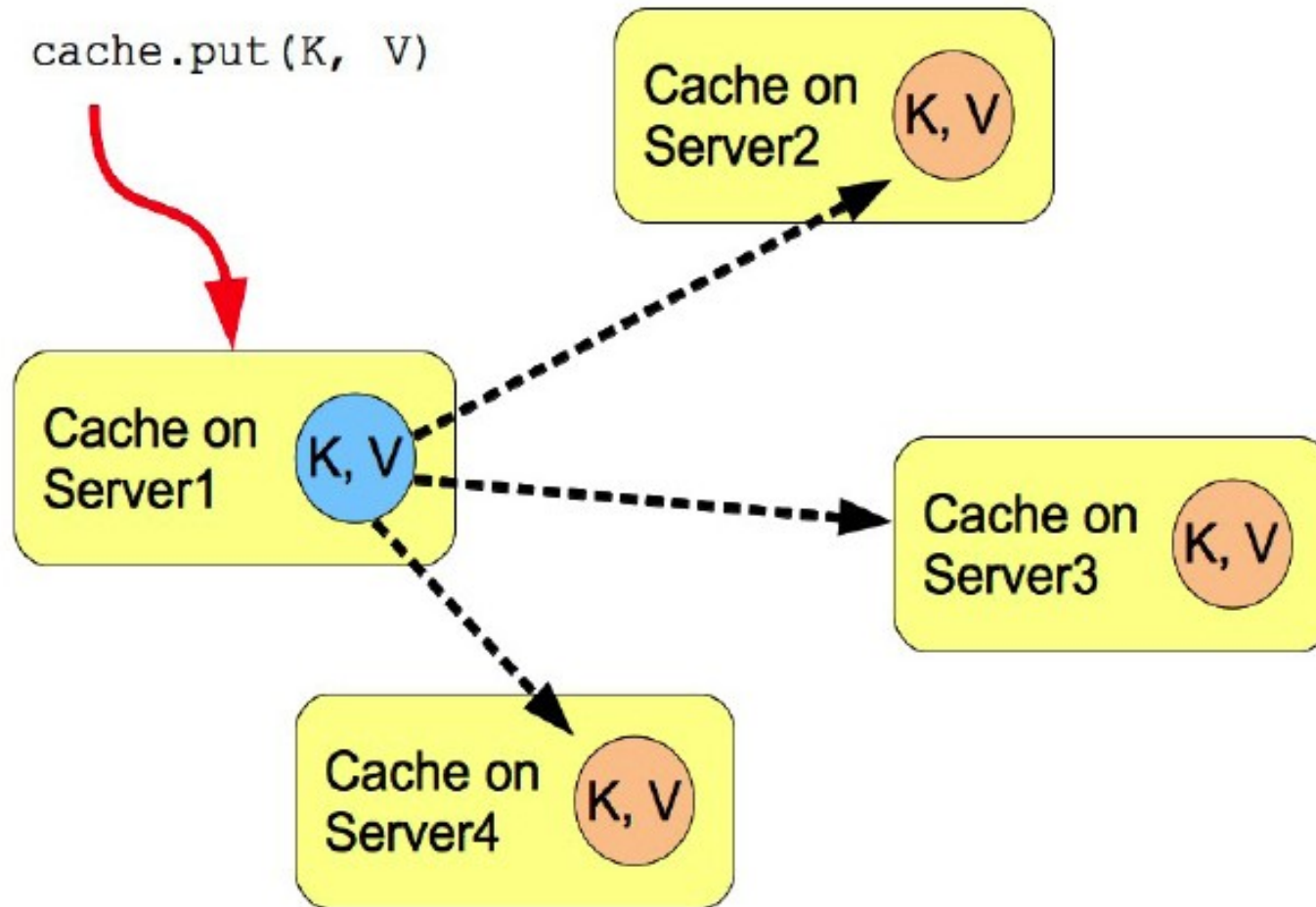


Clustering modes

- **Local** - no clustering
 - unaware of other instances on network
- **Replication** – each node contains all the entries
- **Distribution** – each entry is on x nodes
 - $1 \leq x \leq \text{Number of nodes}$
- **Invalidation** – for use with shared cache store
 - explained later



Replication mode

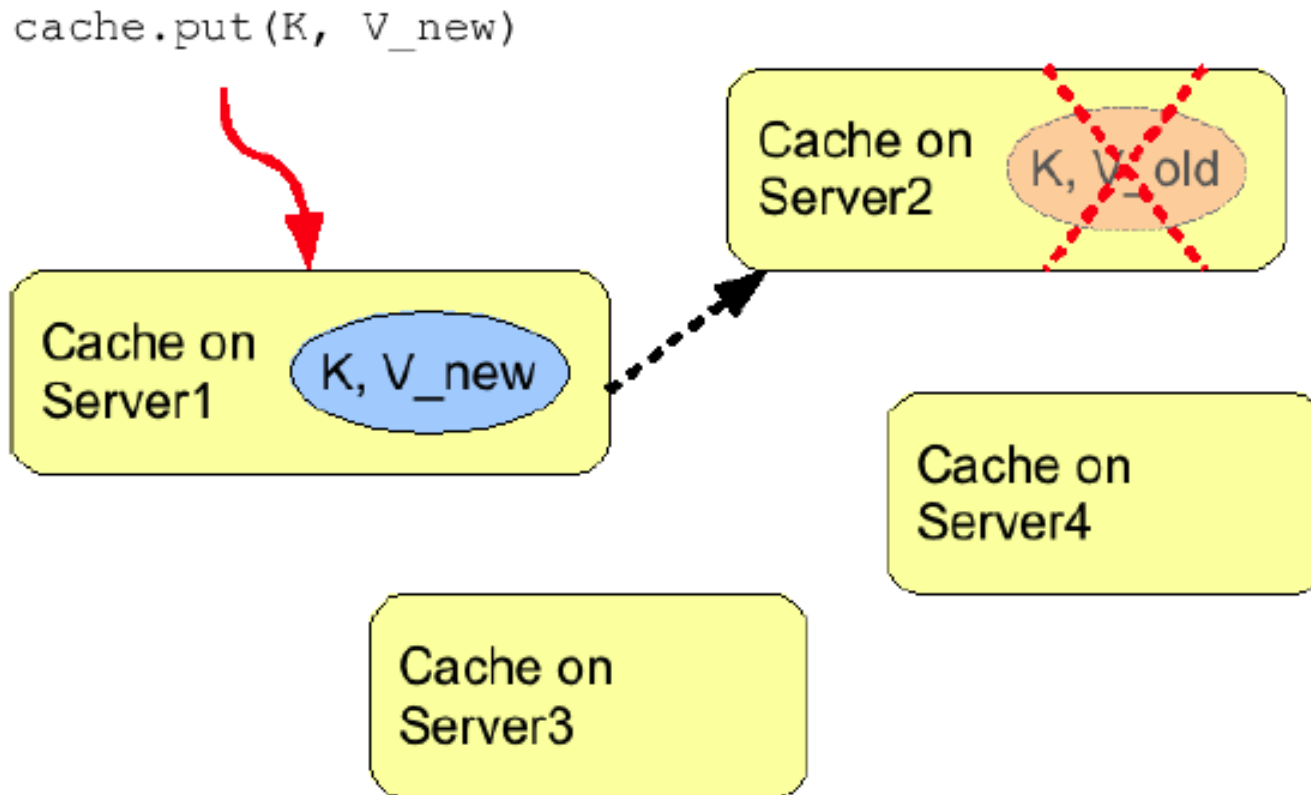


Replication mode

- Advantages
 - N node cluster tolerates N-1 failures
 - Read friendly – we don't need to fetch data from owner node
 - Instant scale-in, no state transfer on leave
- Disadvantages
 - Write unfriendly, put broadcast to every node
 - Doesn't scale well
 - When node joins all state has to be transferred to new node
 - Heap size stays the same when we add nodes



Invalidation mode



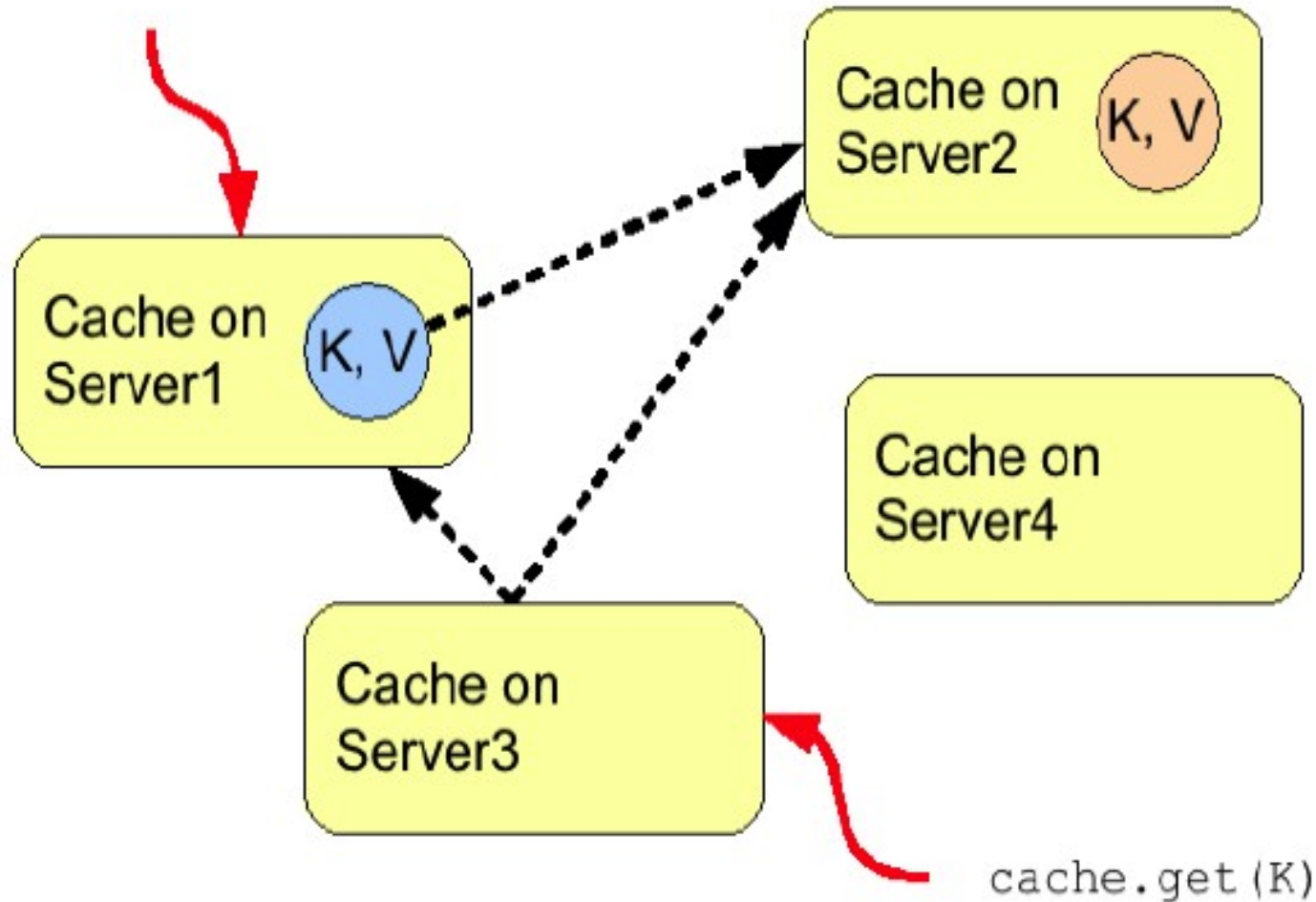
Invalidation mode

- Suitable for RDBMS off-loading, used with shared cache store
- Entry exists in node's local cache => it's valid and can be returned to requestor
- Entry doesn't exist in node's local cache => it's retrieved from the persistent store
- If a node modifies/removes entry it's invalidated in other nodes
- Low internode message traffic, PUT sends only invalidation messages and they are small.



Distribution mode

`cache.put (K, V)`

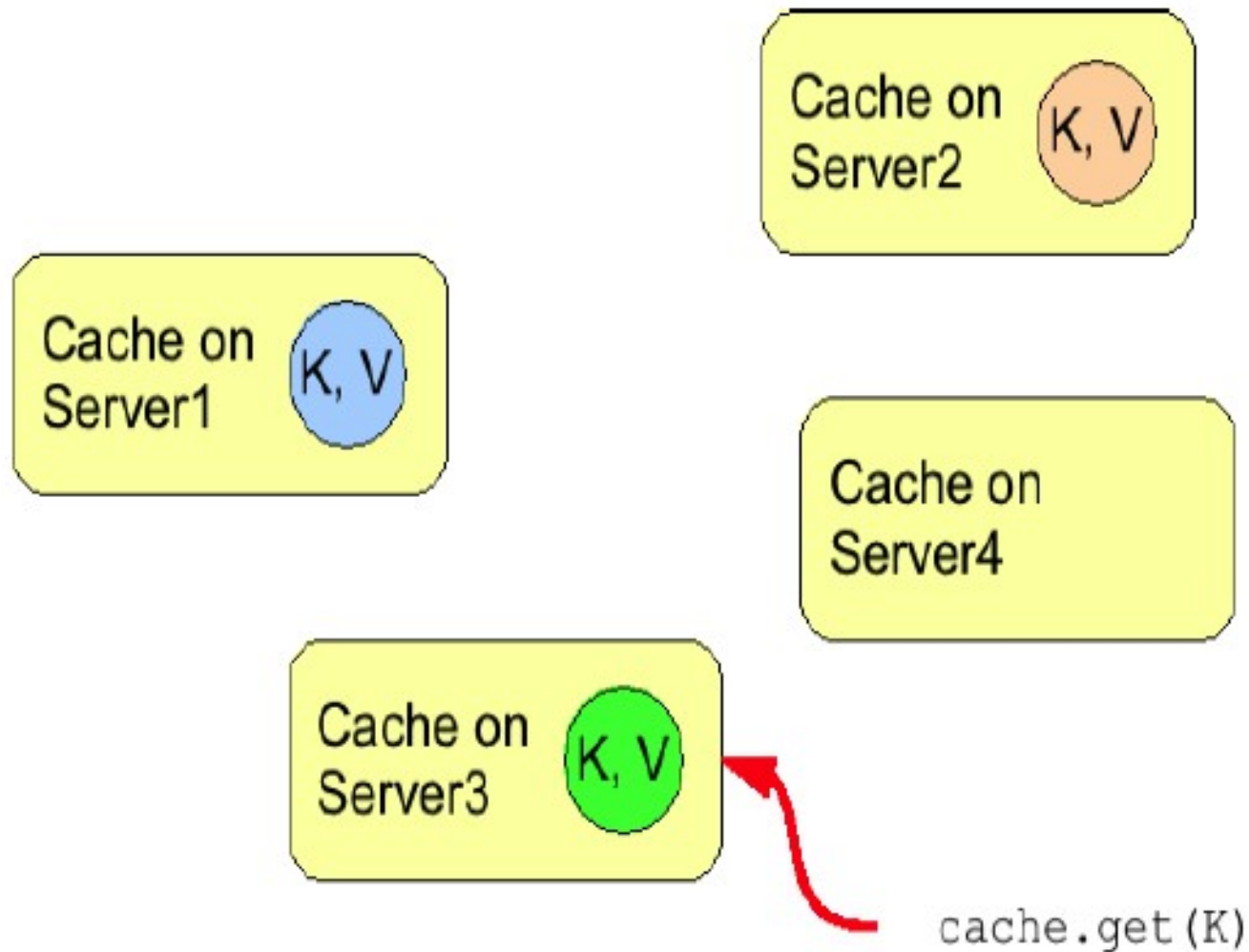


Distribution mode

- Advantages
 - Scalability – number of replication RPCs independent of cluster size – depends only on numOwners
 - set numOwners to compromise between failure tolerance and performance
 - Virtual heap size = $\text{numNodes} * \text{heapSize} / \text{numOwners}$
- Disadvantages
 - Not every node is an owner of the key, GET may require network hops
 - Hash function is not perfect (in 5.1+ virtual nodes improved this greatly)
 - Node join/leave => State transfer (rehash)



Distribution mode – L1 Cache



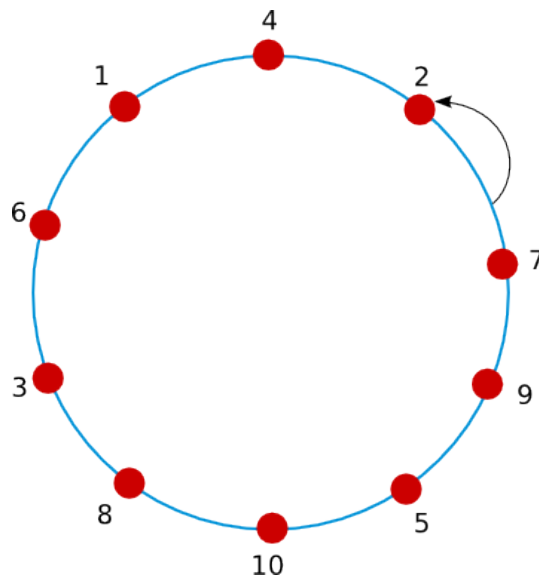
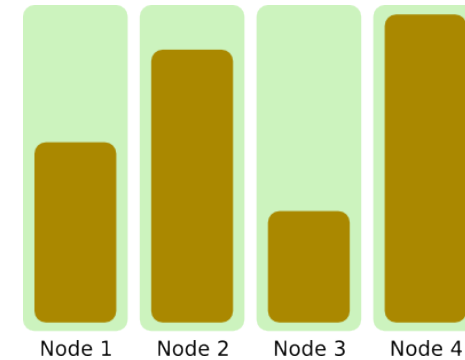
Distribution mode – L1 Cache

- Advantages
 - subsequent GETs don't fetch remote data
- Disadvantages
 - L1 cache needs to be invalidated – number of invalidation messages can be $>$ numOwners (anyone can have a cached copy)
 - L1 cache takes up more memory

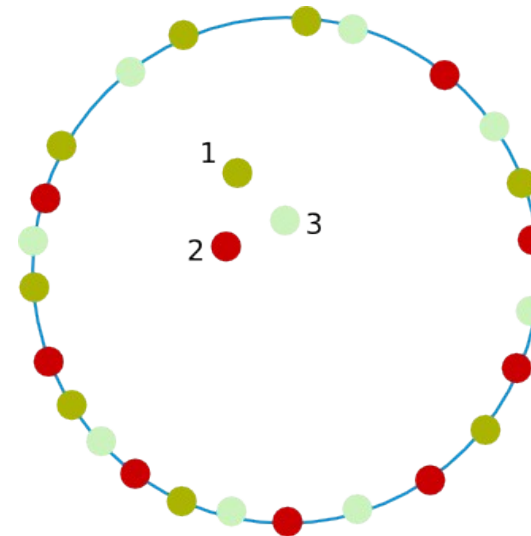


Why good Consistent Hash function matters

- Even distribution of entries – balanced load
- Less expected rehash on node leave / join



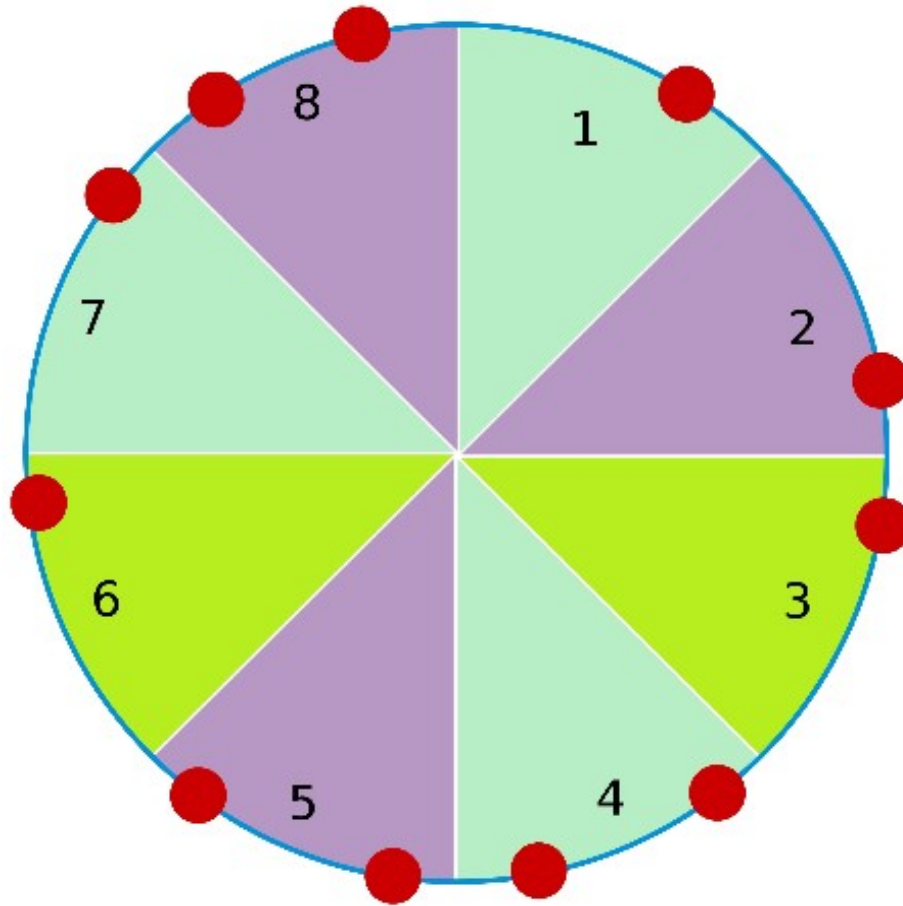
Hash wheel



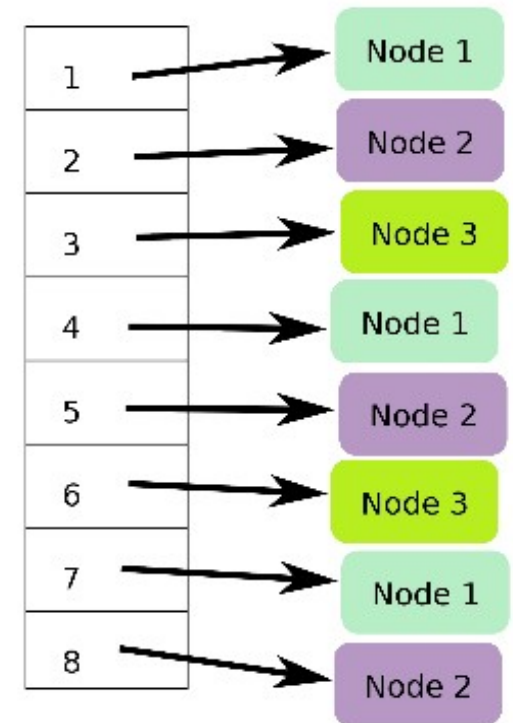
Virtual nodes



Consistent Hash function in Infinispan 5.2.x



Hash wheel



Routing table



Sync vs Async mode

- Sync
 - All operations get confirmation that the other relevant cluster nodes reached the desired state
- Async
 - All operations block only until they perform local changes, we don't wait for JGroups responses.
 - Better throughput but no guarantees on data integrity in cluster.



REST Server

`http://<hostname>[:<port>]/infinispan-server-rest/rest/<cache_name>/<key>`

e.g.

`http://localhost:8080/infinispan-server-rest/rest/___defaultcache/abcd`

HTTP Methods supported:

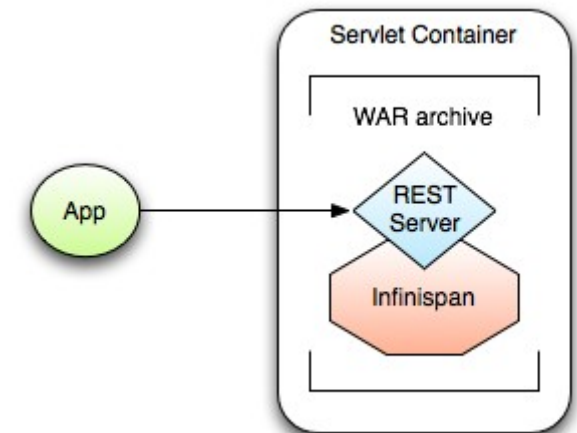
HEAD, GET, PUT, POST, DELETE

Standard headers supported:

Content-Type

ETag

Last-Modified



REST Server access via Python

```
#  
# Sample python code using the standard http lib only  
#  
import httplib  
  
#putting data in  
conn = httplib.HTTPConnection("localhost:8080")  
data = "SOME DATA HERE !" #could be string, or a file...  
conn.request("POST", "/infinispan/rest/Bucket/0", data, {"Content-Type": "text/plain"})  
response = conn.getresponse()  
print response.status  
  
#getting data out  
import httplib  
conn = httplib.HTTPConnection("localhost:8080")  
conn.request("GET", "/infinispan/rest/Bucket/0")  
response = conn.getresponse()  
print response.status  
print response.read()
```



REST Server access via Ruby

```
#
# Shows how to interact with Infinispan REST api from ruby.
# No special libraries, just standard net/http
#
# Author: Michael Neale
#
require 'net/http'

http = Net::HTTP.new('localhost', 8080)

#Create new entry
http.post('/infinispan/rest/MyData/MyKey', 'DATA HERE', {"Content-Type" => "text/plain"})

#get it back
puts http.get('/infinispan/rest/MyData/MyKey').body

#use PUT to overwrite
http.put('/infinispan/rest/MyData/MyKey', 'MORE DATA', {"Content-Type" => "text/plain"})

#and remove...
http.delete('/infinispan/rest/MyData/MyKey')

#Create binary data like this... just the same...
http.put('/infinispan/rest/MyImages/Image.png', File.read('/Users/michaelneale/logo.png'), {"Content-Type" => "image/png"})

#and if you want to do json...
require 'rubygems'
require 'json'

#now for fun, lets do some JSON !
data = {:name => "michael", :age => 42 }
http.put('/infinispan/rest/Users/data/0', data.to_json, {"Content-Type" => "application/json"})
```



REST Server access via command line (curl)

PUT

```
curl -X PUT -d "aaa" http://localhost:8080/infinispan-server-rest/rest/___defaultcache/aaa
```

GET

```
curl -X GET http://localhost:8080/infinispan-server-rest/rest/___defaultcache/aaa
```

DELETE

```
curl -X DELETE http://localhost:8080/infinispan-server-rest/rest/___defaultcache/aaa
```

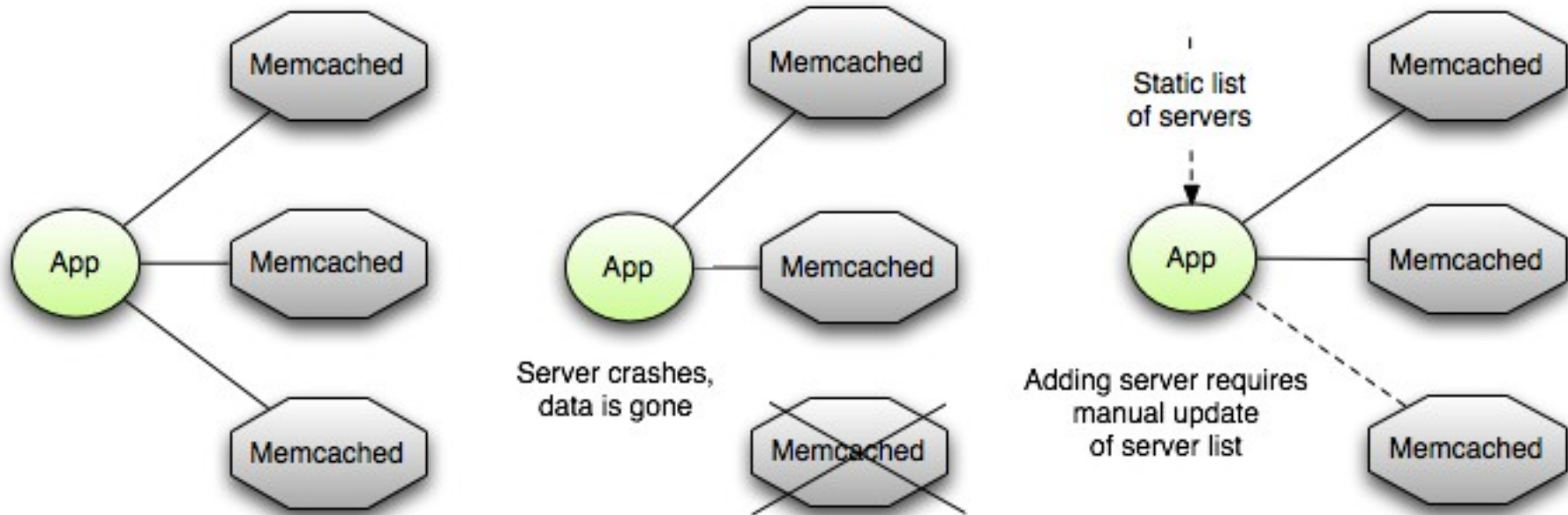


Memcached

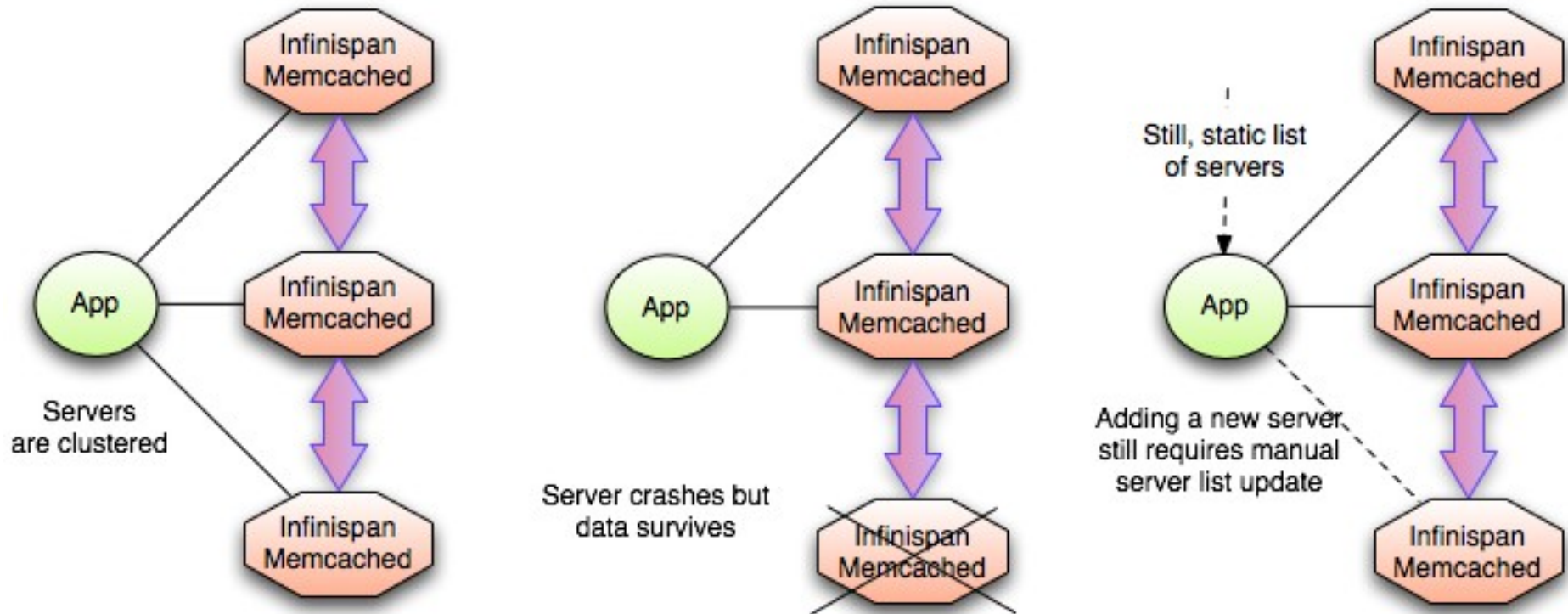
- Open protocol for popular memcached server:
<http://memcached.org/>
- Python
 - Python-memcached client library
- Java
 - Spymemcached client
- There is Binary and Text protocol version
- Infinispan supports text protocol only



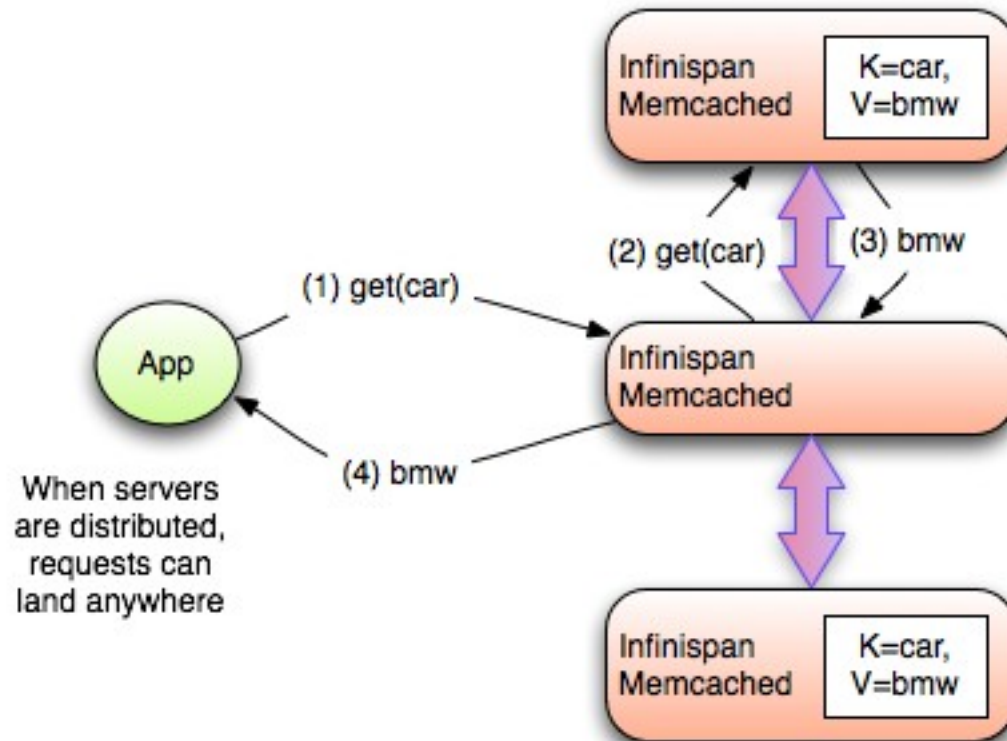
Memcached server (original version)



Memcached server (Infinispan implementation)

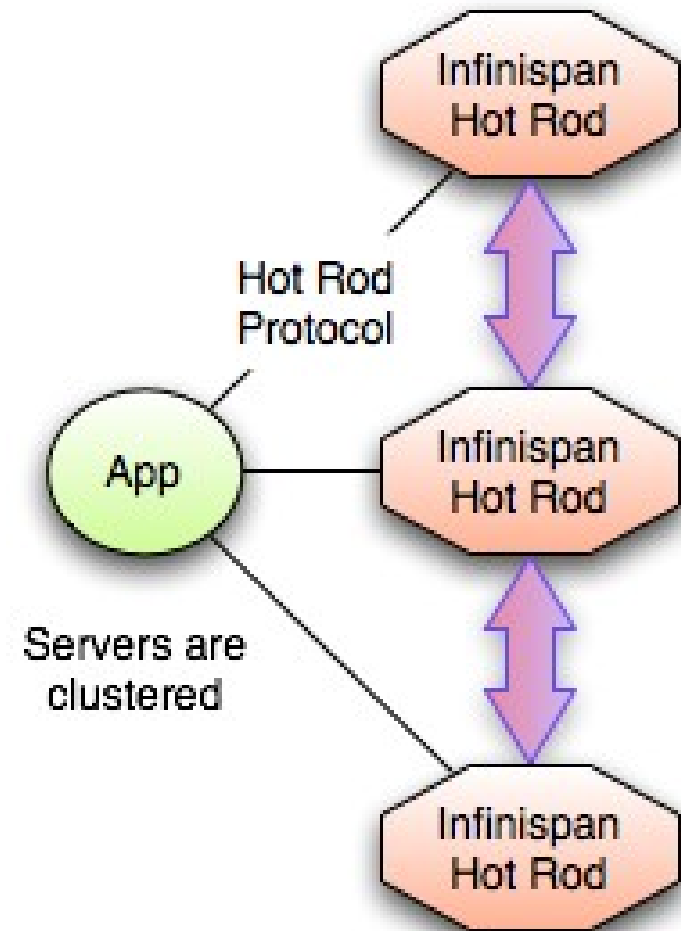


Routing not so smart

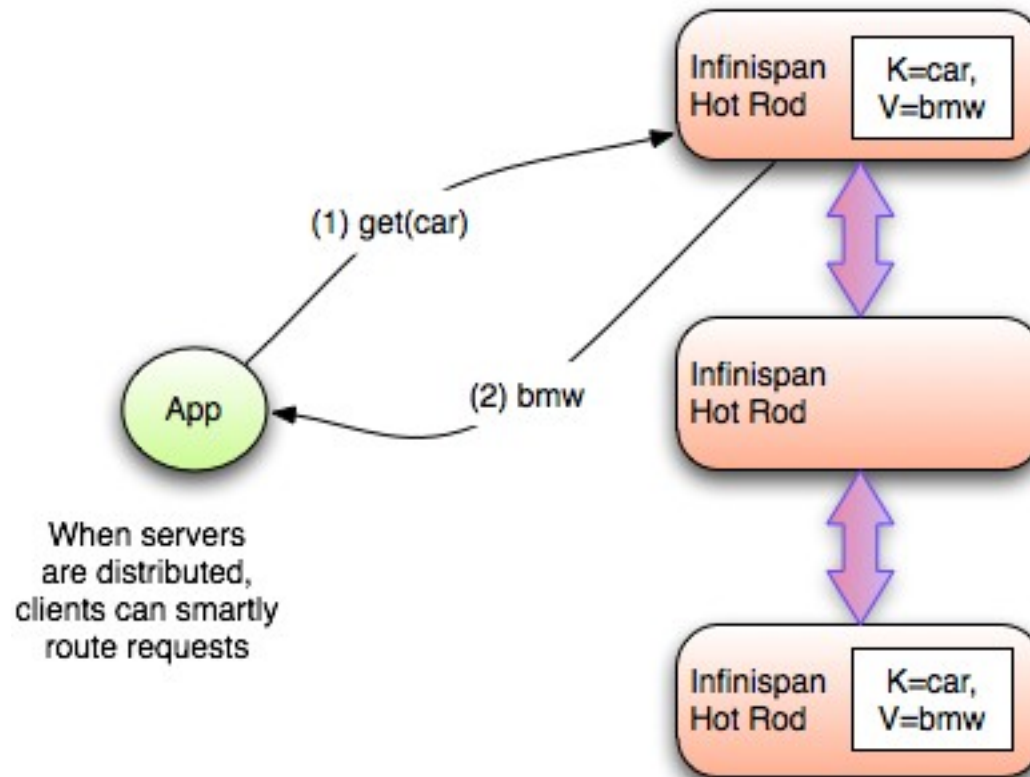


Hot Rod

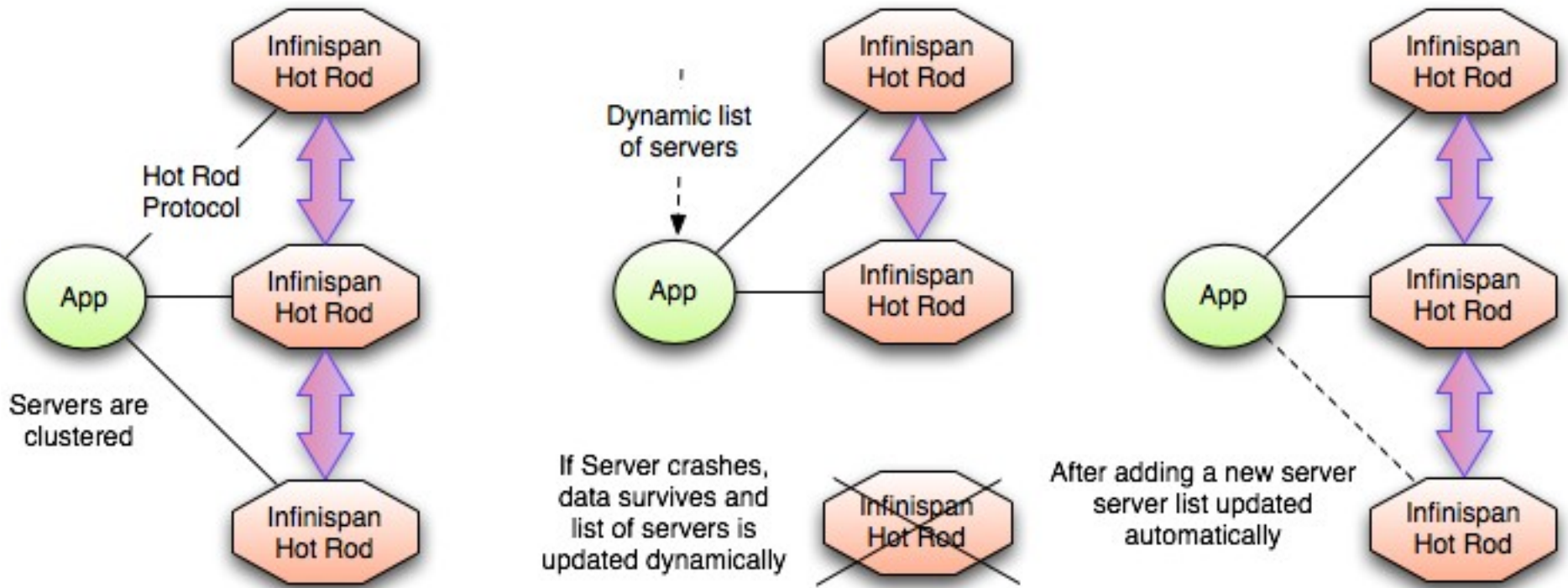
- Infinispan's own binary wire protocol
- Open and language independent
- Built-in dynamic failover and load balancing
- Smart routing



Smart routing with Hot Rod



Dynamic routing with Hot Rod



For Java users: it's a Map (again)

```
// DefaultCacheManager cacheManager = new DefaultCacheManager("infinispan.xml");  
  
RemoteCacheManager cacheManager = new RemoteCacheManager("localhost:11222");  
cacheManager.start();  
  
Cache<String, Object> cache = cacheManager.getCache("namedCache");  
  
cache.put("key", "value");  
  
Object value = cache.get("key");
```



Clients - comparison

	Protocol	Client libraries	Clustered ?	Smart routing	Load balancing / Failover
REST	Text	standard HTTP clients	Yes	No	Any HTTP load balancer
Memcached	Text	Plenty	Yes	No	Only with predefined server list
Hot Rod	Binary	Java, python, C++ on the way	Yes	Yes	Dynamic



Brand new features in version 5.2

- Non-blocking state transfer
- Cross site replication
- Rolling upgrades (HotRod only)
- Command line interface (ispn-cli.sh)

More info:

<https://community.jboss.org/wiki/RollingUpgradesForRemoteClientsUsingHotRod>

<https://community.jboss.org/wiki/DesignForCrossSiteReplication>

<https://community.jboss.org/wiki/Non-BlockingStateTransferV2>

<https://docs.jboss.org/author/display/ISPN/Command-Line+Interface+%28CLI%29>

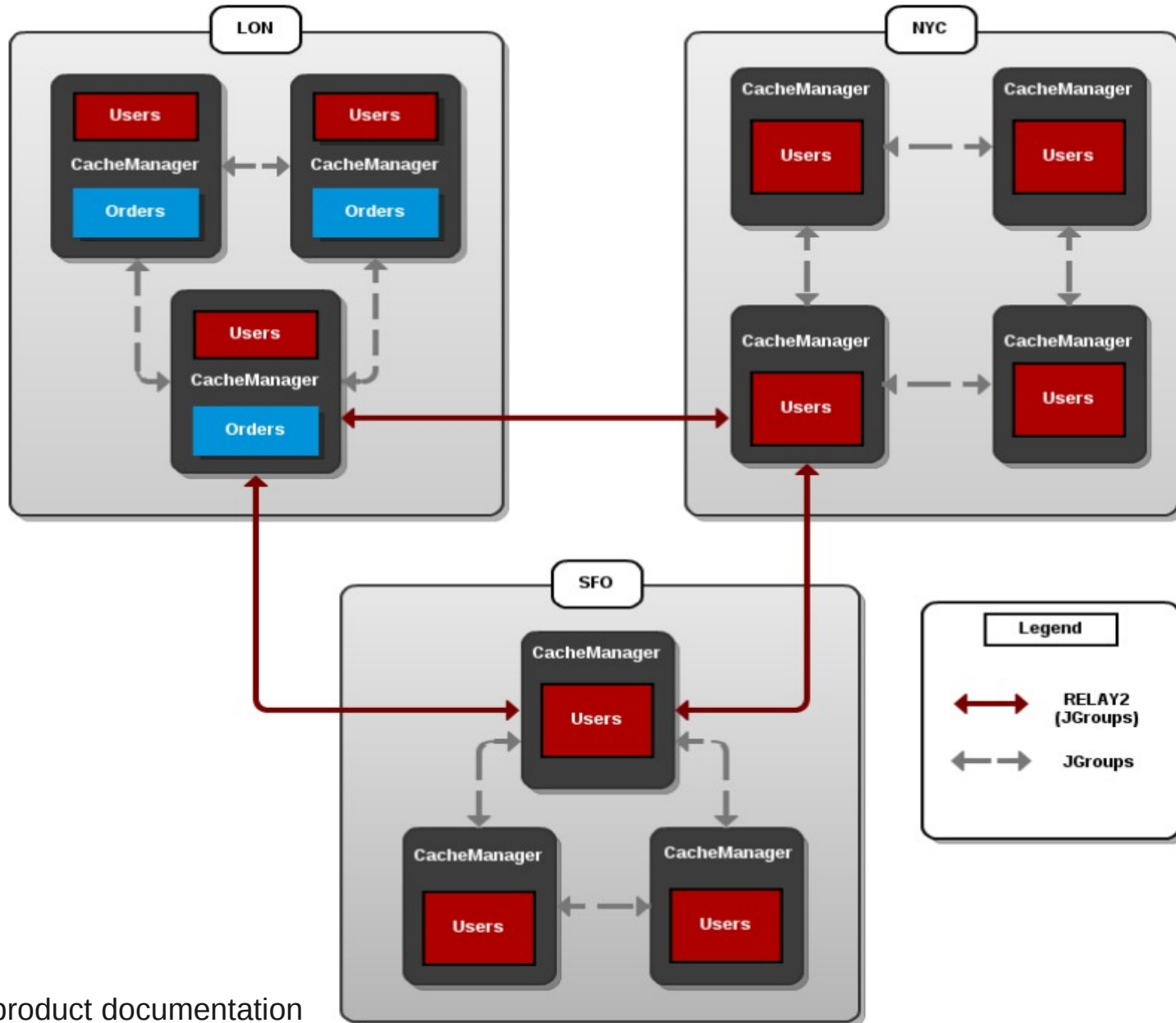


•Non-blocking state transfer

- State transfer
 - Adjusting internal cache's state during view change
 - View change: any join, leave or merge of nodes
- Non-blocking
 - Allow writes during state transfer
 - Minimize interval where cluster can't respond
 - Minimize interval where member stops responding



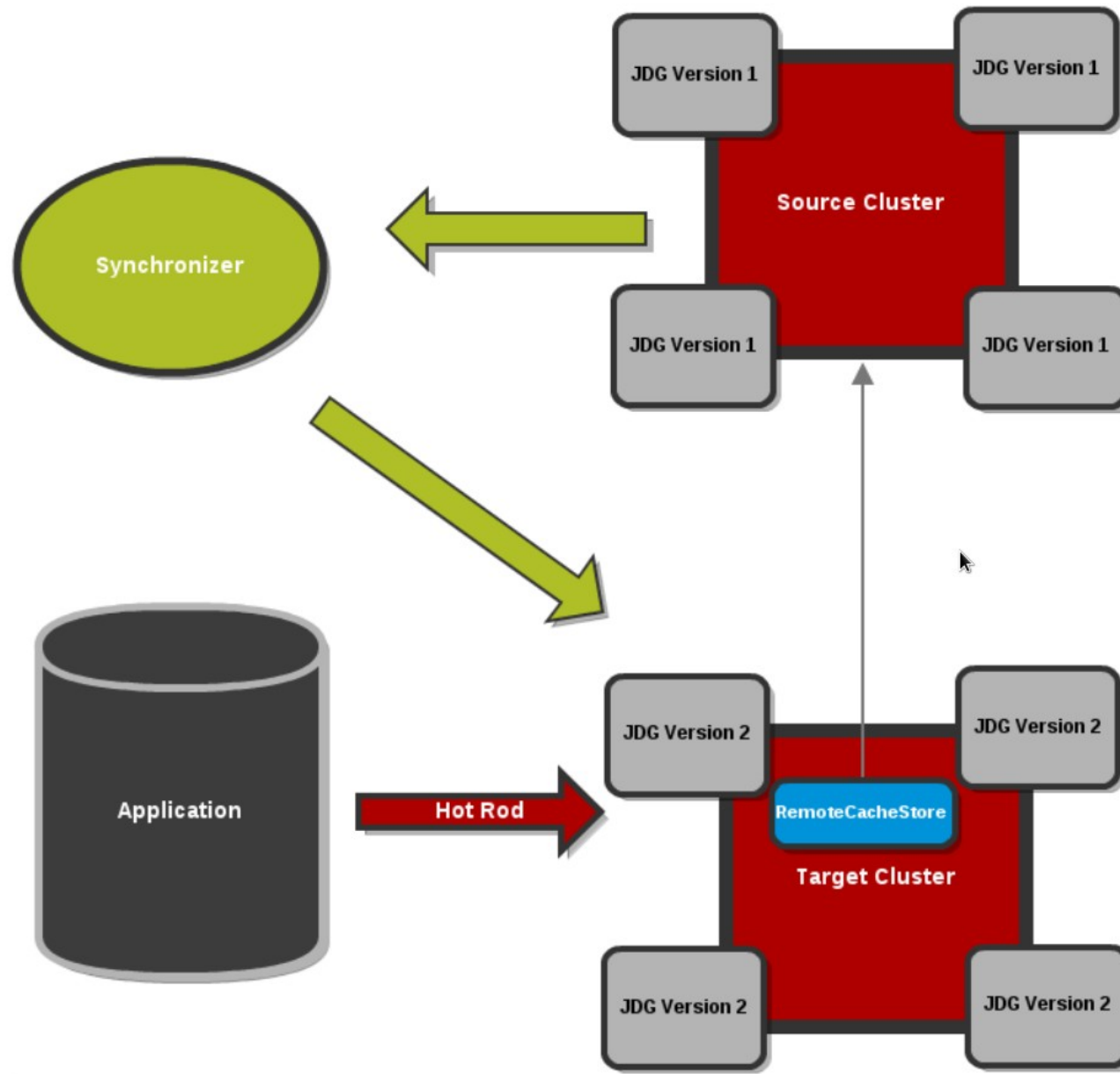
Cross site replication



Source: JDG product documentation



Roll



Source: JDG product documentation



Quick review

- What's Infinispan
- Why / When to use it
- High level features
- How to plug it into your architecture
- Clustering modes
- Client / server access modes
- Brand new features in version 5.2



Questions?



Thank you!

