



# Introduction to Infinispan

Tomáš Sýkora

JBoss Data Grid Quality Engineering  
Red Hat

Contact: [tsykora@redhat.com](mailto:tsykora@redhat.com)

IRC: #infinispan on freenode

November 21<sup>st</sup> 2014

**Don't work so hard... Beer? Parties? Nah, c'mon...**



Source: <http://bf4central.com/battlefield-4-wallpapers/>

3

JBoss 2014 @ CVUT | Tomáš Sýkora

DICE



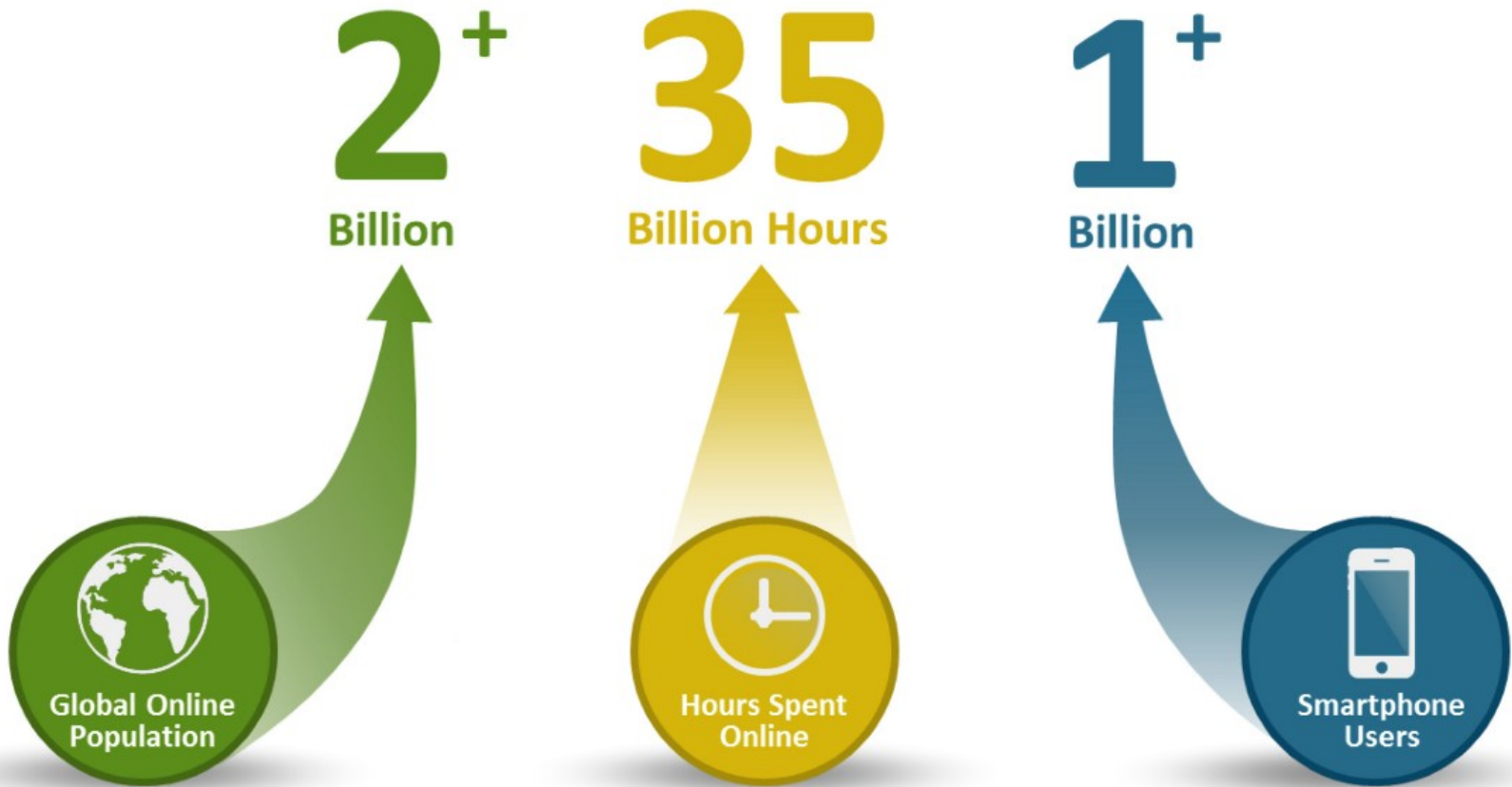
# Agenda

- NoSQL world
- What's Infinispan
- Why / When to use it
- How to plug it into your architecture
- Clustering modes
- Client / server access modes
- High level features
- Features in version 5.2, 6.0
- Brand new features in version 7.0





# Why NoSQL? (viral applications / services)



**Figure 1.** Big Users: With the growth in global internet use, the number of hours spent online, and the increase in smartphone users, it's not uncommon for apps to have millions of users per day.

From [www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf](http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf)



# Why NoSQL? (new types of data)

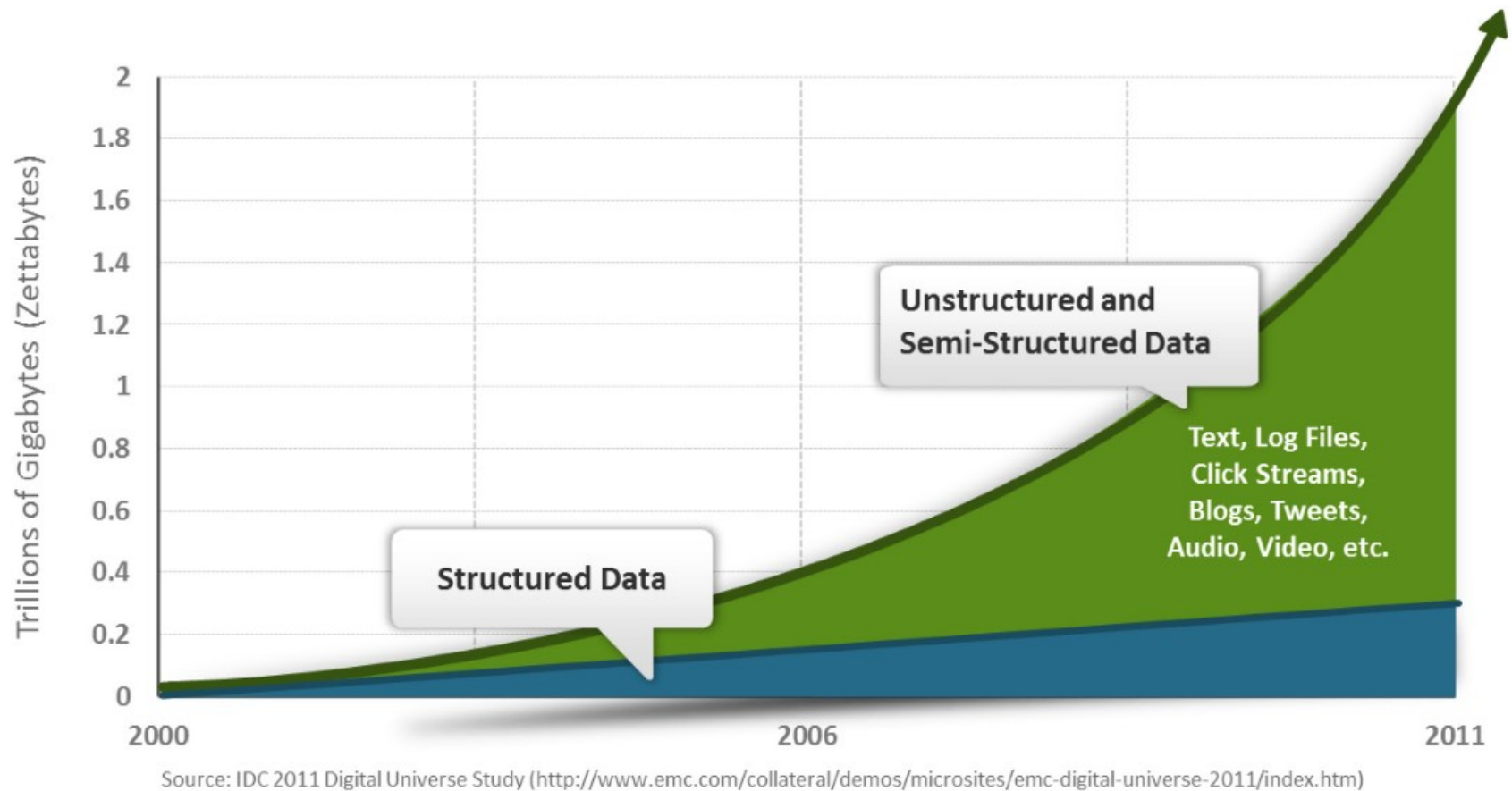


Figure 2. Big Data: The amount of data is growing rapidly, and the nature of data is changing as well. More than 80% of data generated today is unstructured or semi-structured.

From [www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf](http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf)



# Why NoSQL? (dealing with traffic peaks)

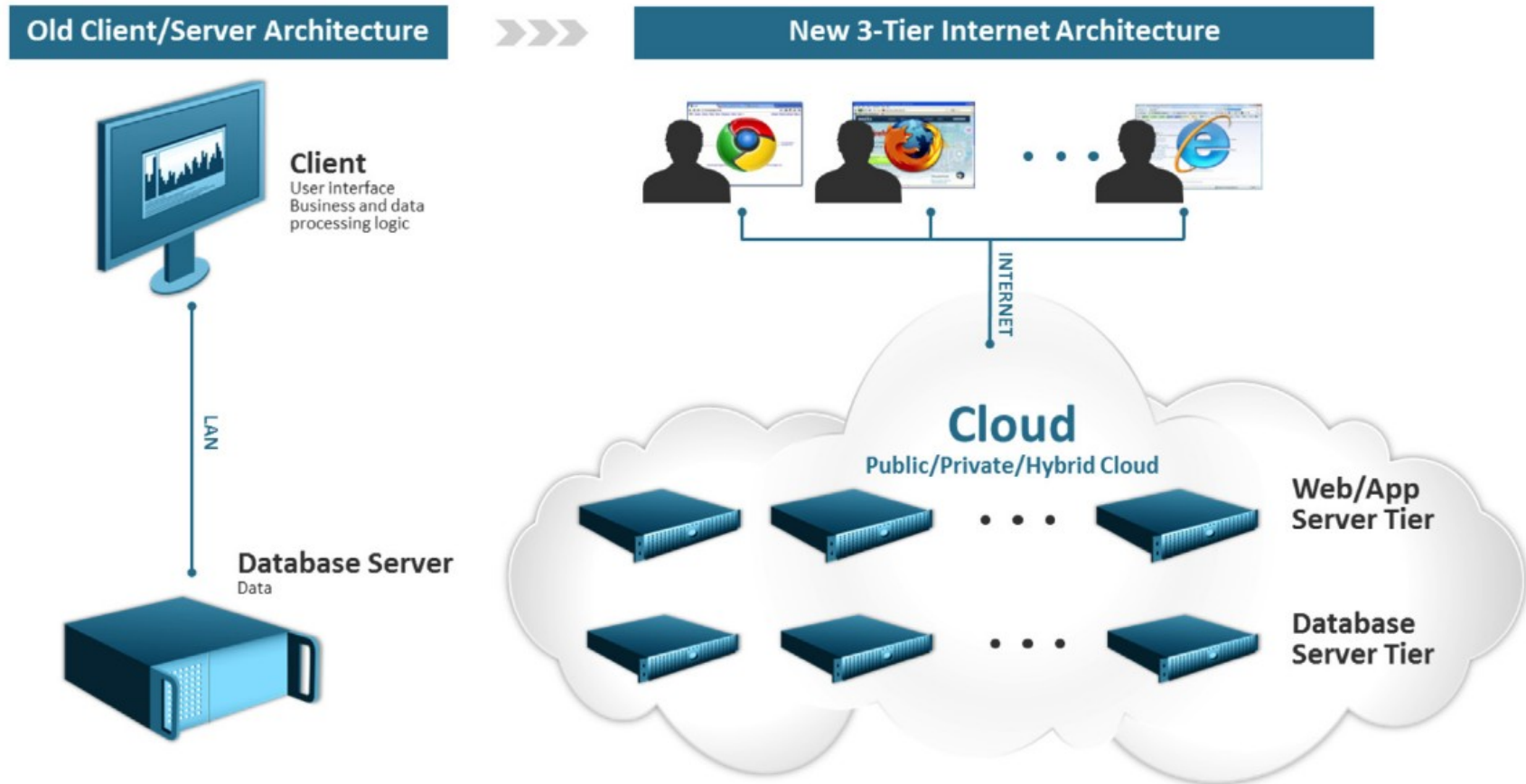


Figure 3. Applications today are increasingly developed using a three-tier internet architecture, requiring a horizontally scalable database tier that easily scales with the number of users and amount of data your application has.

From [www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf](http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf)



# Why NoSQL? (More flexibility?)

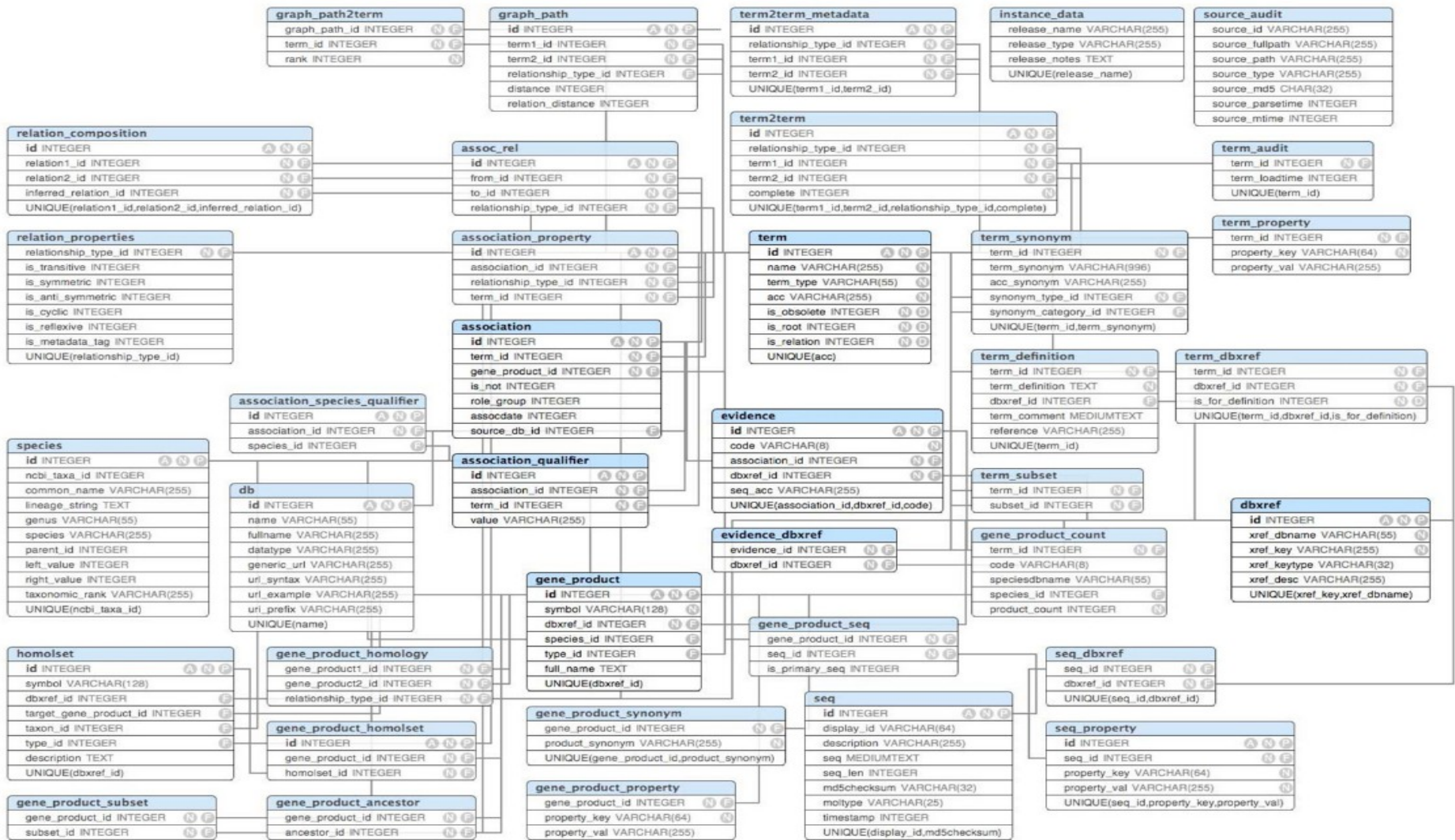


Figure 4: With relational databases, any operation requires the collection and processing of data from across tens or hundreds of interrelated tables, greatly hindering performance.

From [www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf](http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf)





# Why NoSQL? (Yes, more flexibility please)

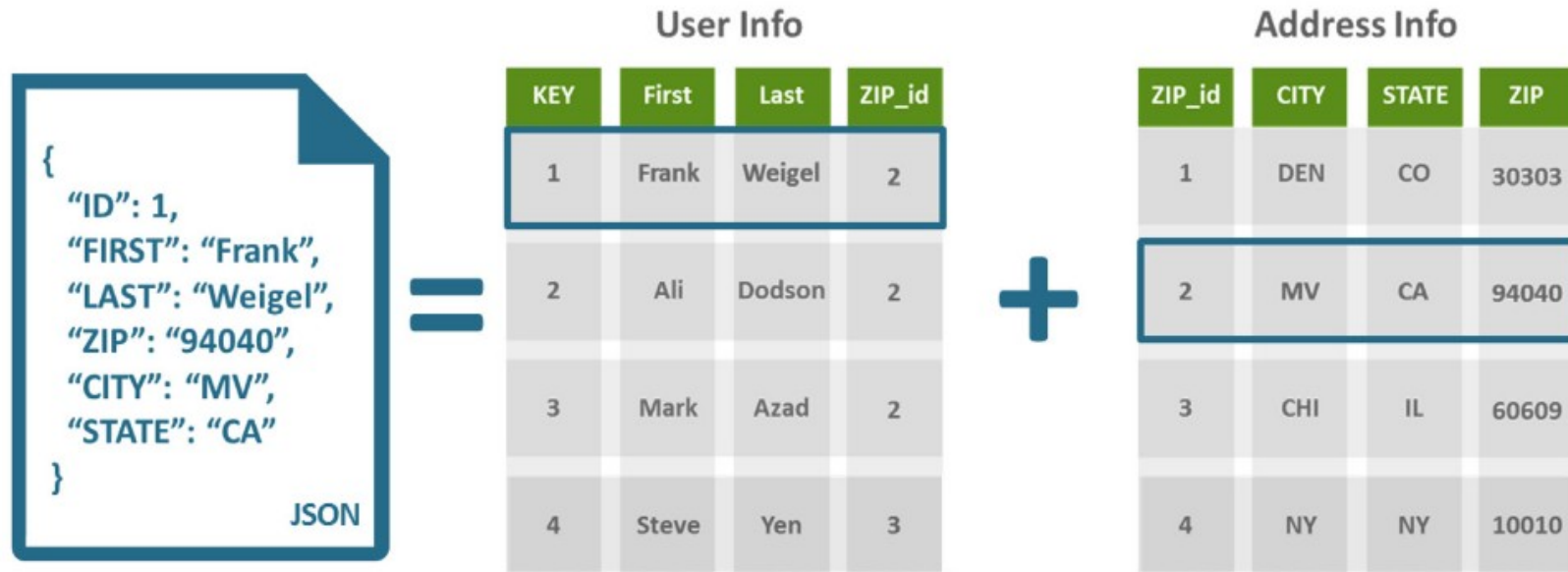


Figure 5: Unlike relational databases, which must store and retrieve data from scores of interrelated tables, document databases can store an entire object in a single JSON document, making it faster to retrieve.

## Developers: OOP vs RDBMS

From [www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf](http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf)





# Why developers are considering NoSQL databases?

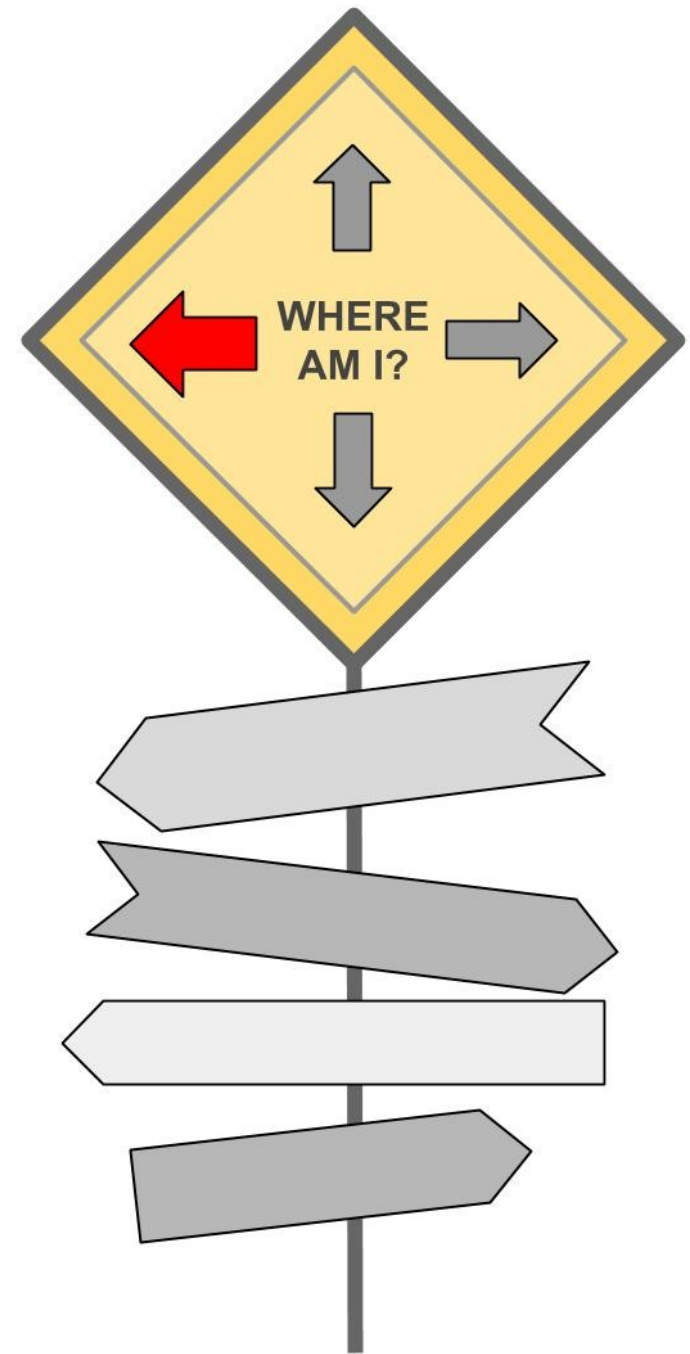
- Better application development productivity through a more flexible data model
- Greater ability to scale dynamically to support more users and data
- Improved performance to satisfy expectations of users wanting highly responsive applications

From [www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf](http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf)



# Find your path...

- NoSQL world
- **What's Infinispan**
- Why / When to use it
- Plug it into your architecture
- Infinispan clustering modes
- Client / server access modes
- High level features
- Features in version 5.2, 6.0
- Brand new features in version 7.0



# What's Infinispan?







Infinispan



# 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 (reads and writes at the same time)
- Transactional
- Queryable (also document store)

Red Hat productized version: **JBoss Data Grid**



# 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 (reads and writes at the same time)
- Transactional
- Queryable (also document store)

Red Hat productized version: **JBoss Data Grid**



# 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 (reads and writes at the same time)
- Transactional
- Queryable (also document store)

Red Hat productized version: **JBoss Data Grid**



# 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 (reads and writes at the same time)
- Transactional
- Queryable (also document store)

Red Hat productized version: **JBoss Data Grid**





# 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
- **Higly available, resilient to node failures**
- Concurrent (reads and writes at the same time)
- Transactional
- Queryable (also document store)

Red Hat productized version: **JBoss Data Grid**



# 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** (reads and writes at the same time)
- **Transactional**
- Queryable (also document store)

Red Hat productized version: **JBoss Data Grid**



# 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 (reads and writes at the same time)
- Transactional
- **Queryable** (also document store)

Red Hat productized version: **JBoss Data Grid**



# For Java users: it's a Map

**Yes, just a Map – everything else is for free!**

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

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

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

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





# Configuration in XML

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

Read more:

[http://infinispan.org/docs/7.0.x/user\\_guide/user\\_guide.html#\\_configuring\\_cache\\_declaratively](http://infinispan.org/docs/7.0.x/user_guide/user_guide.html#_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();
```

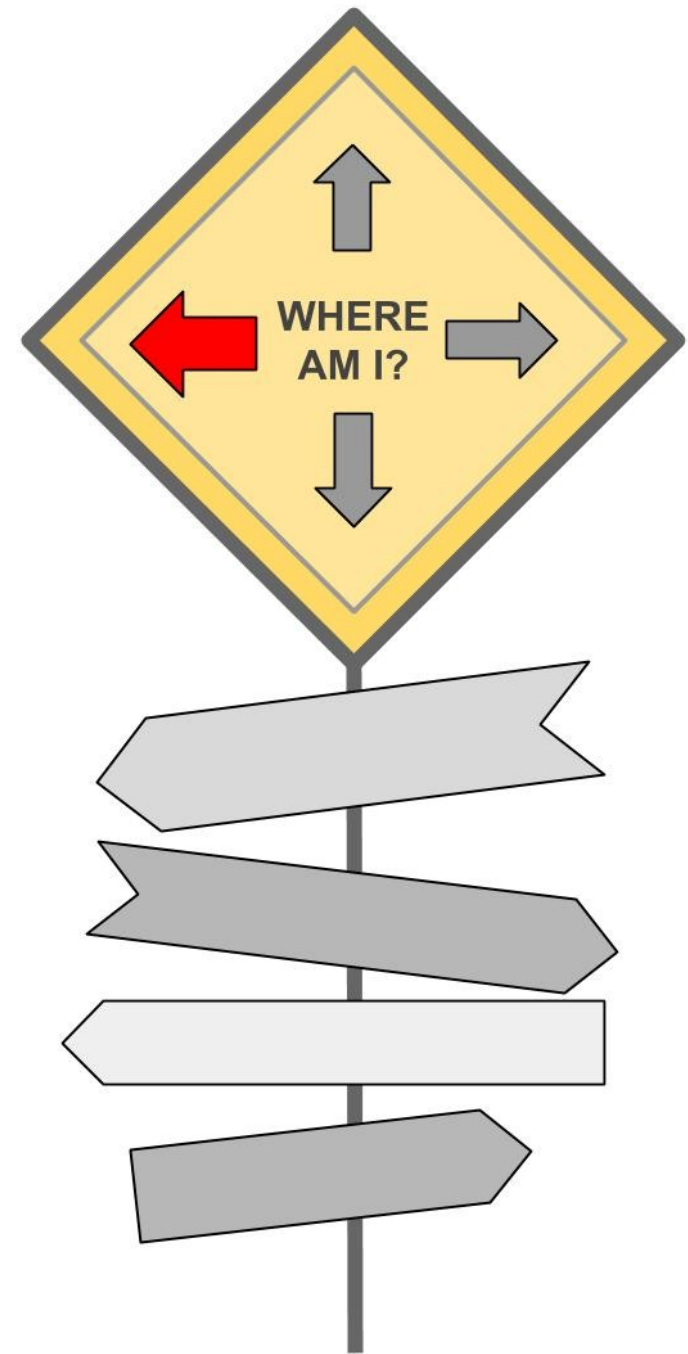
Read more:

[http://infinispan.org/docs/7.0.x/user\\_guide/user\\_guide.html#\\_configuring\\_cache\\_programmatically](http://infinispan.org/docs/7.0.x/user_guide/user_guide.html#_configuring_cache_programmatically)

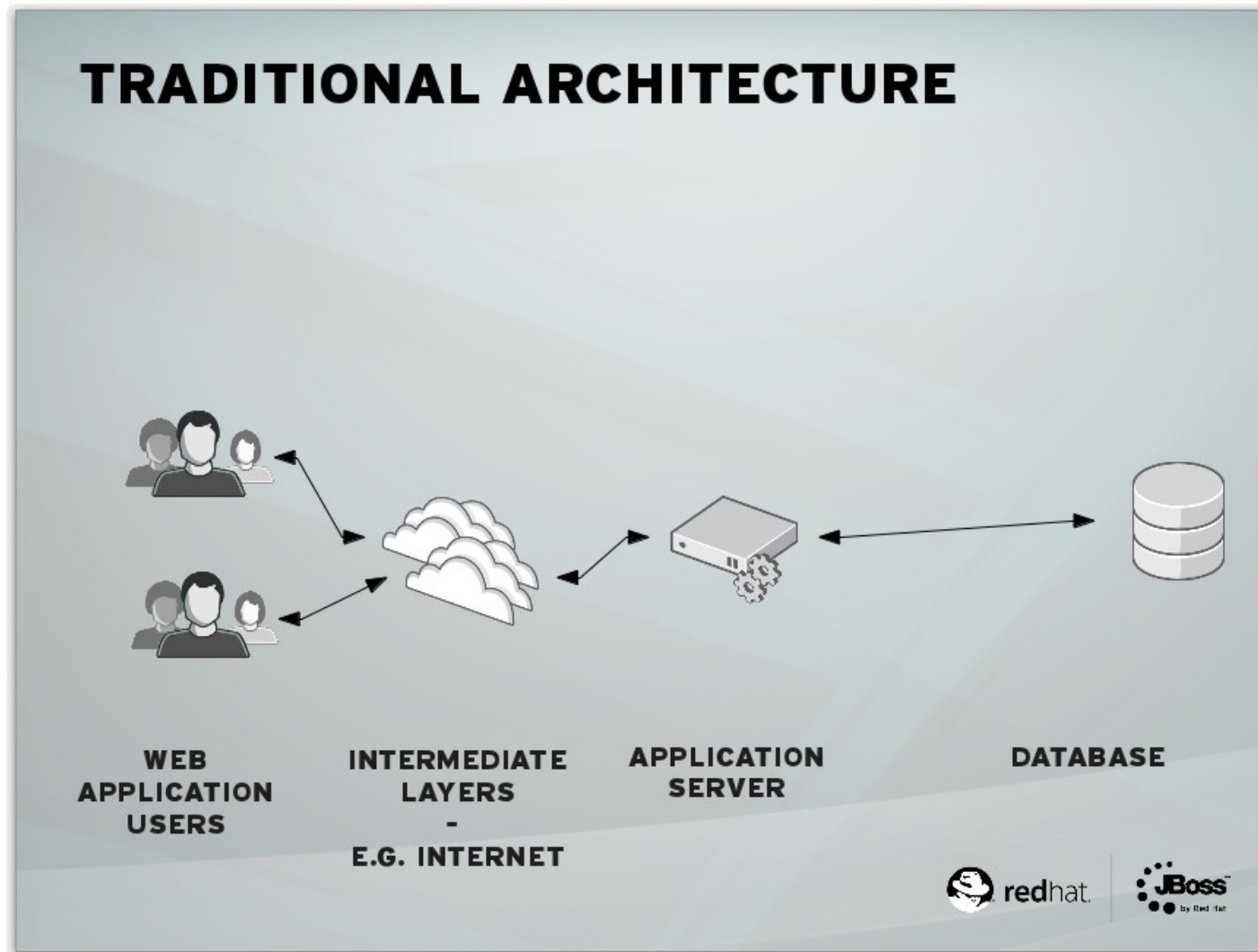


# Find your path...

- NoSQL world
- What's Infinispan
- **Why / When to use it**
- Plug it into your architecture
- Infinispan clustering modes
- Client / server access modes
- High level features
- Features in version 5.2, 6.0
- Brand new features in version 7.0



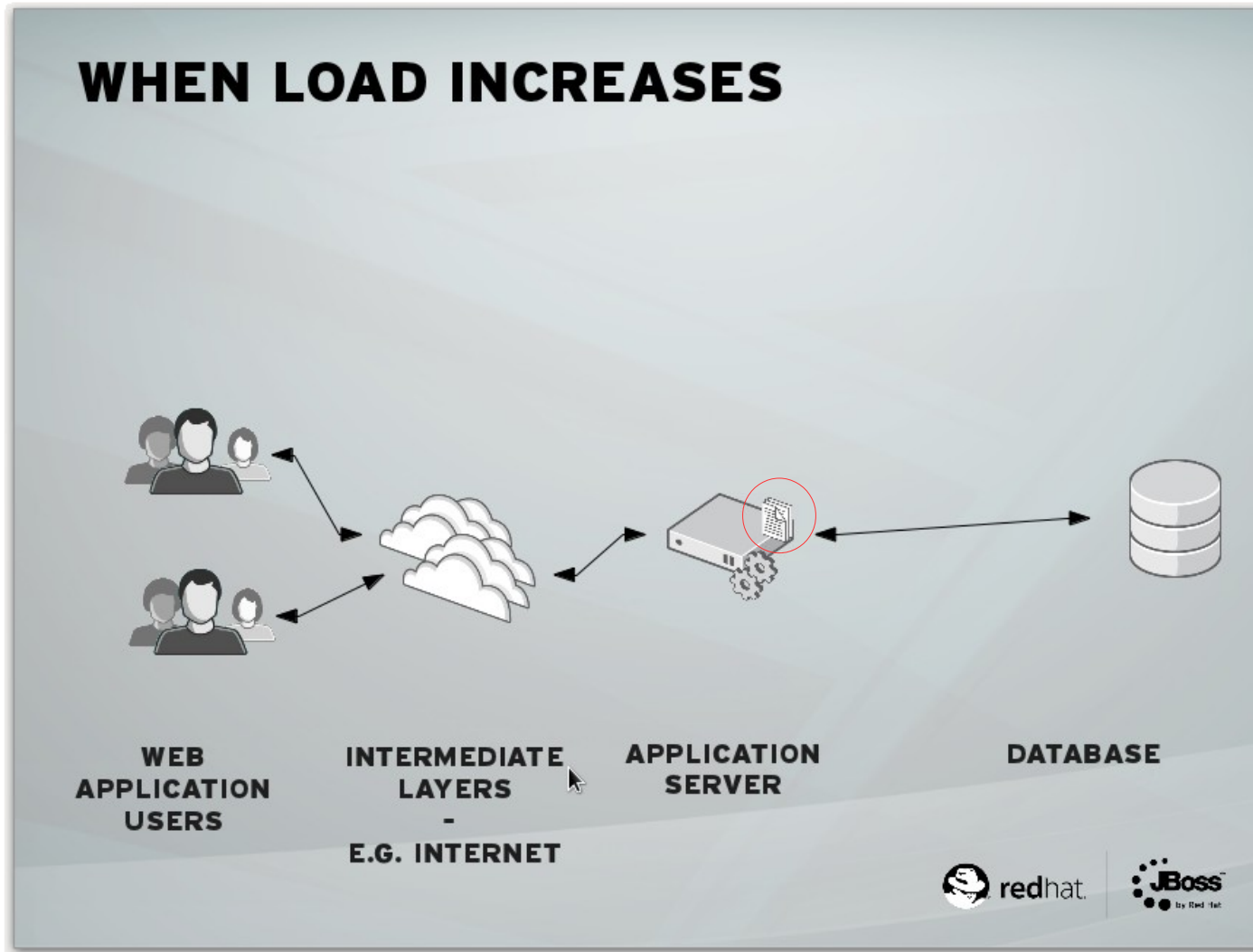
# Why Datagrid?



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



# Why Datagrid?

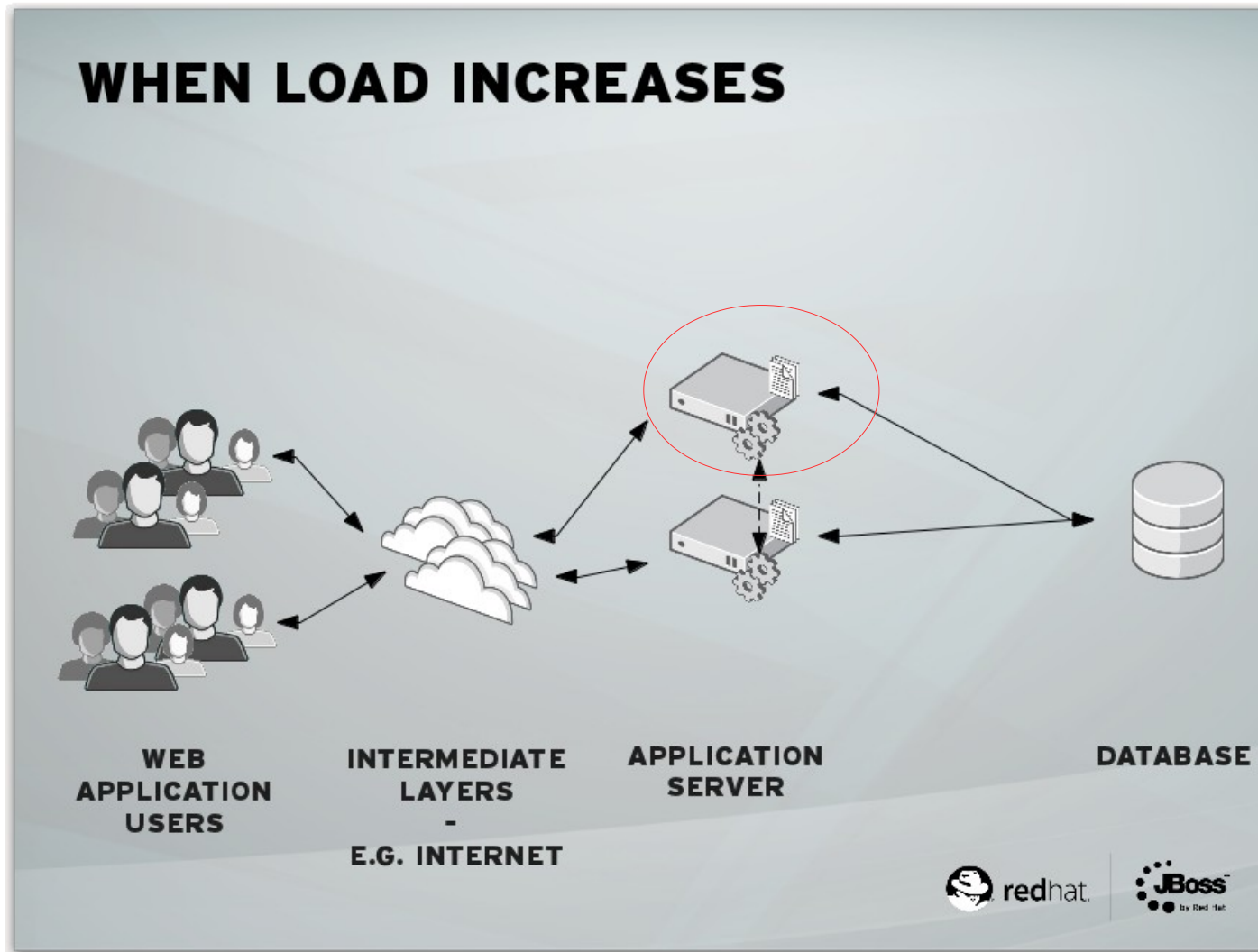


From <http://fhornain.wordpress.com/2012/04/21/jboss-data-grid-when-database-is-very-expensive/>





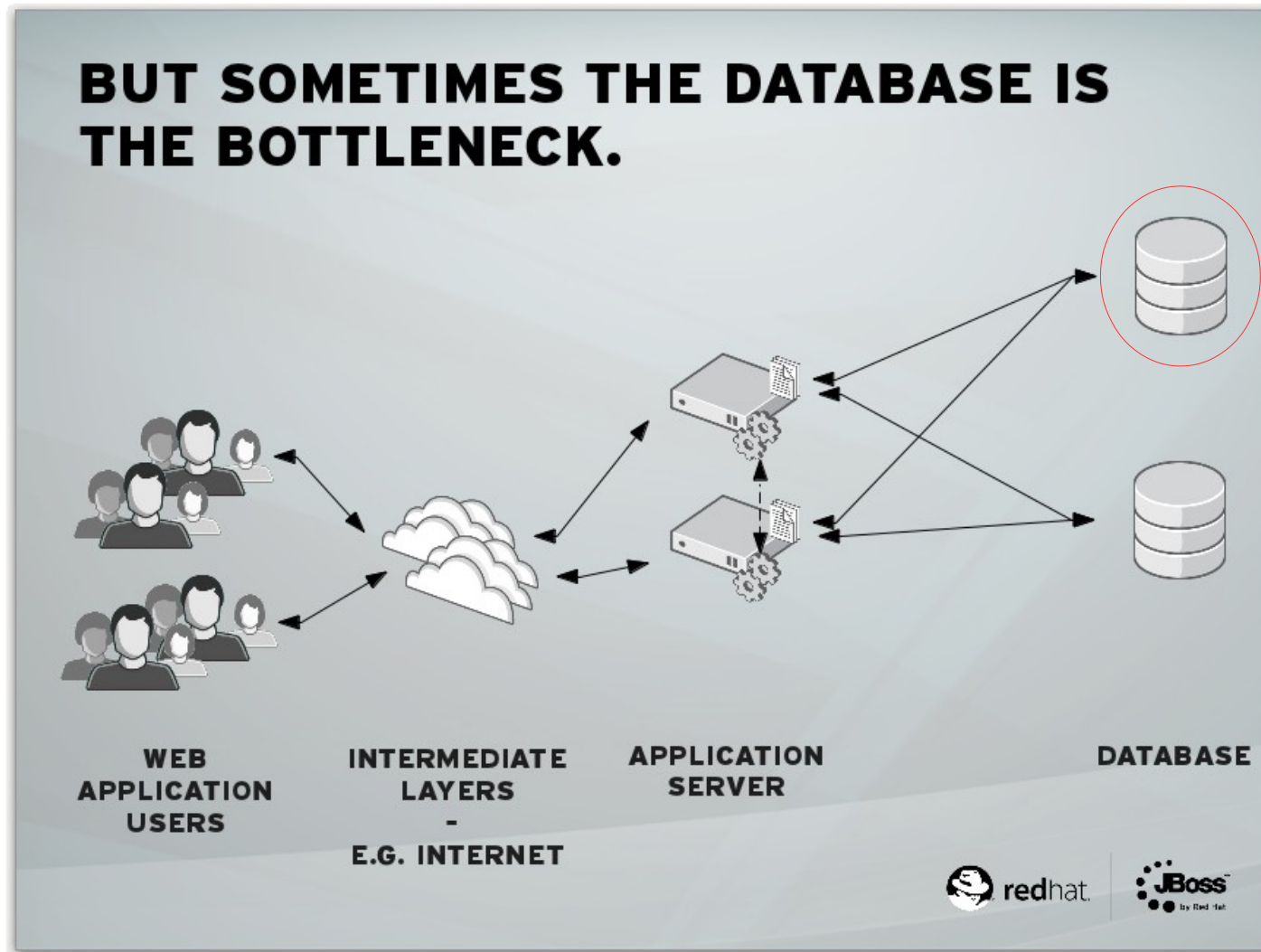
# Why Datagrid? (Your business is successful)



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



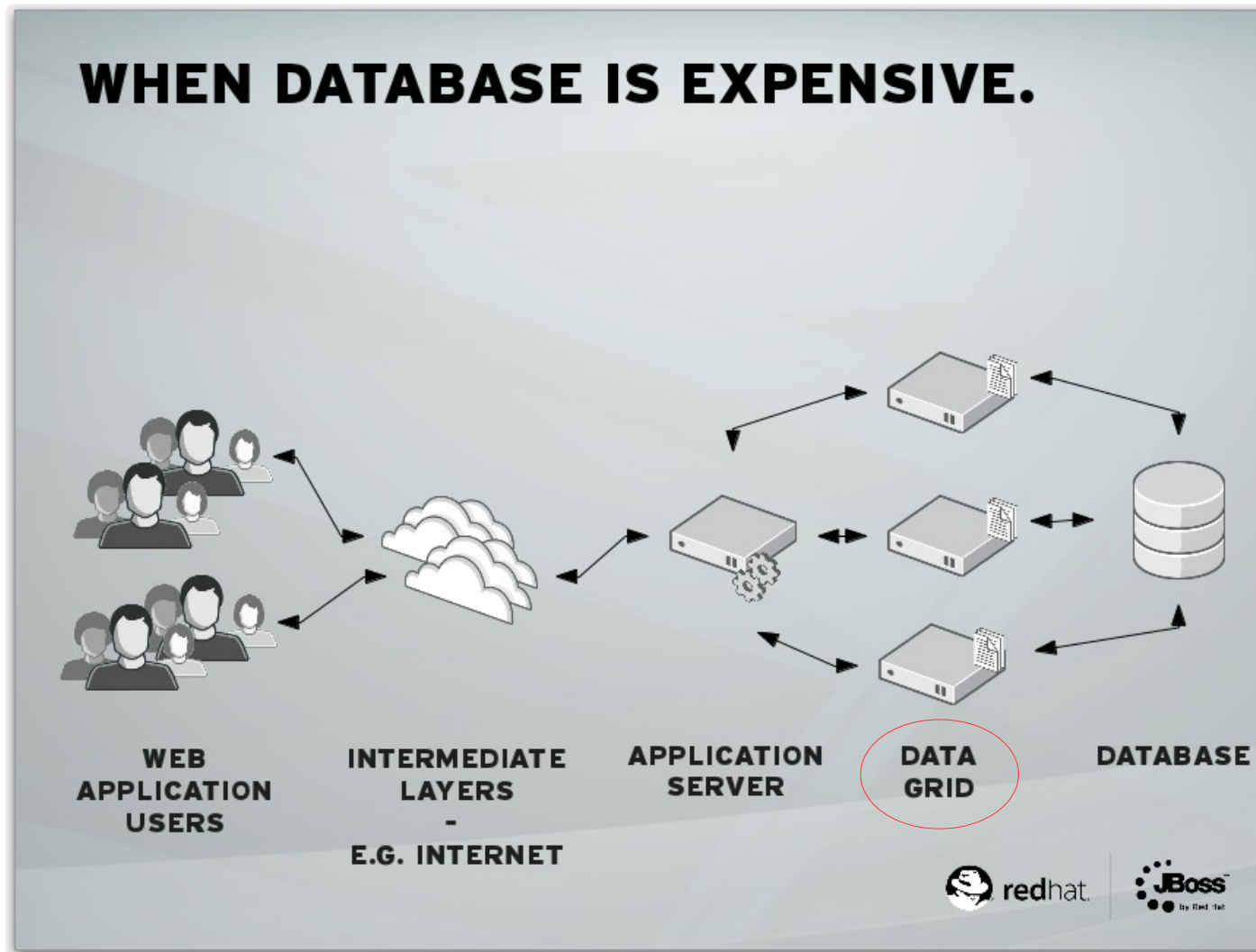
# Why Datagrid? (You are even more successful)



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



# Why Datagrid?

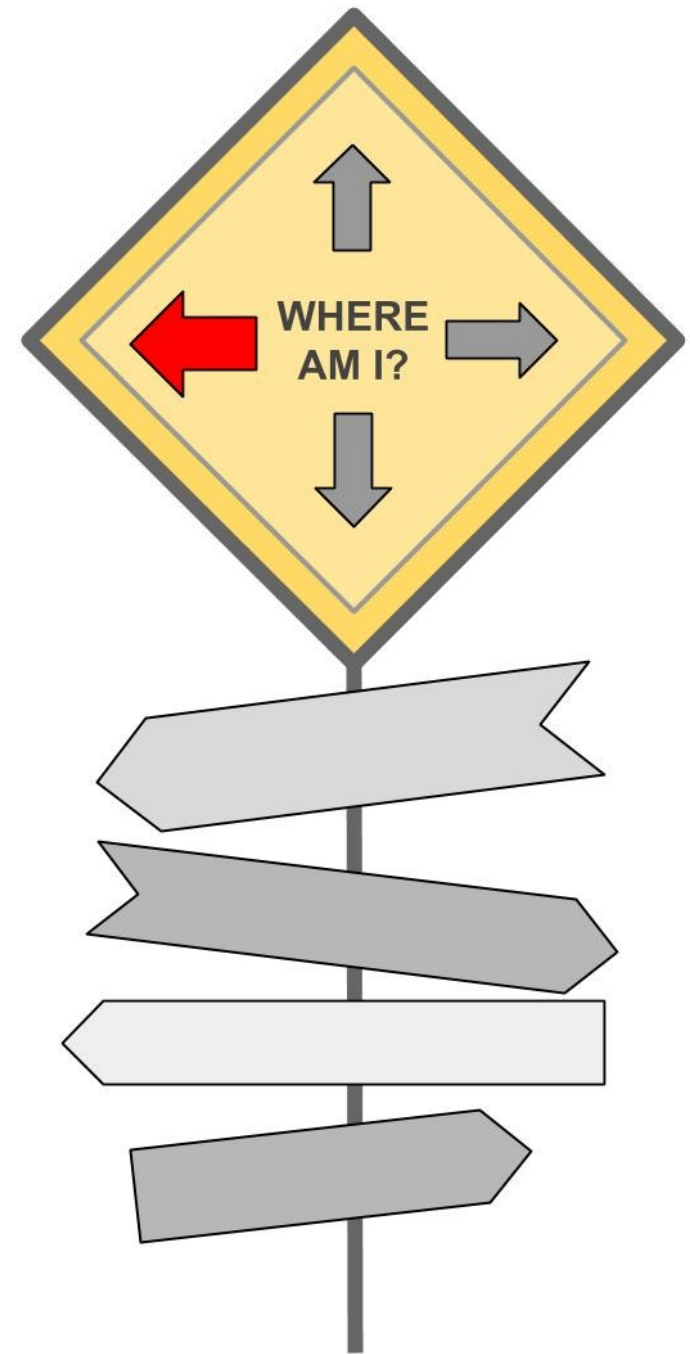


From <http://fhornain.wordpress.com/2012/04/21/jboss-data-grid-when-database-is-very-expensive/>



# Find your path...

- NoSQL world
- What's Infinispan
- Why / When to use it
- **Plug it into your architecture**
- Infinispan clustering modes
- Client / server access modes
- High level features
- Features in version 5.2, 6.0
- Brand new features in version 7.0



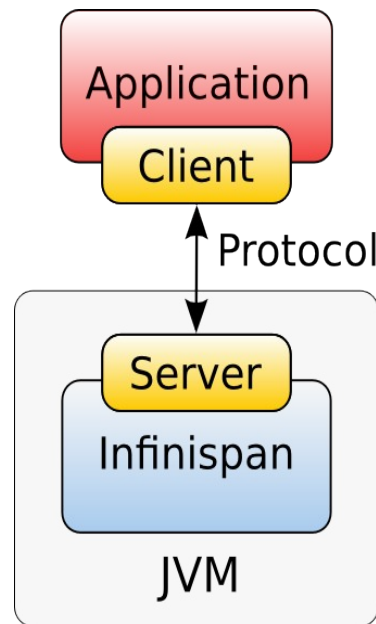
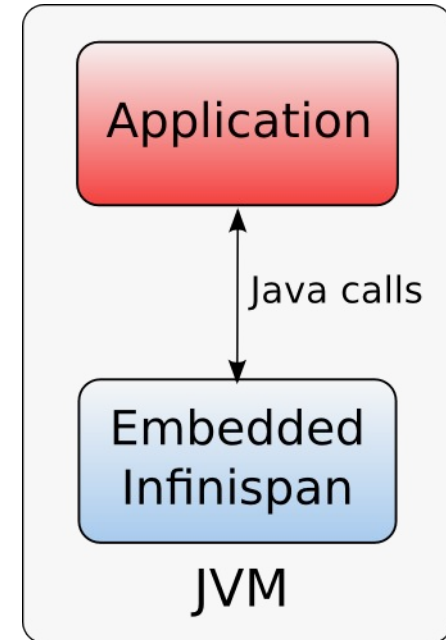
# How to plug it into your architecture ?



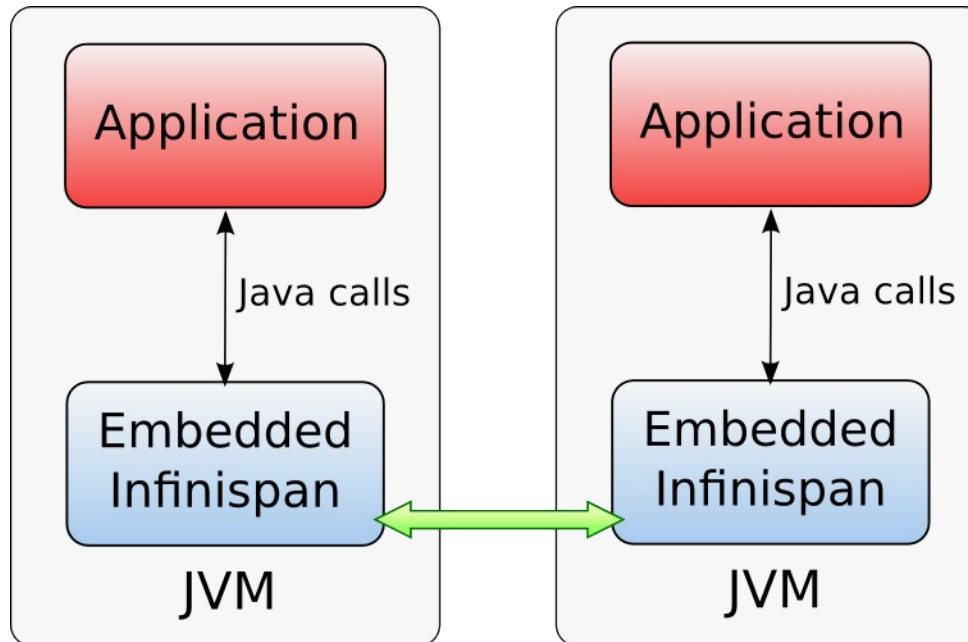


# Modes of access / usage

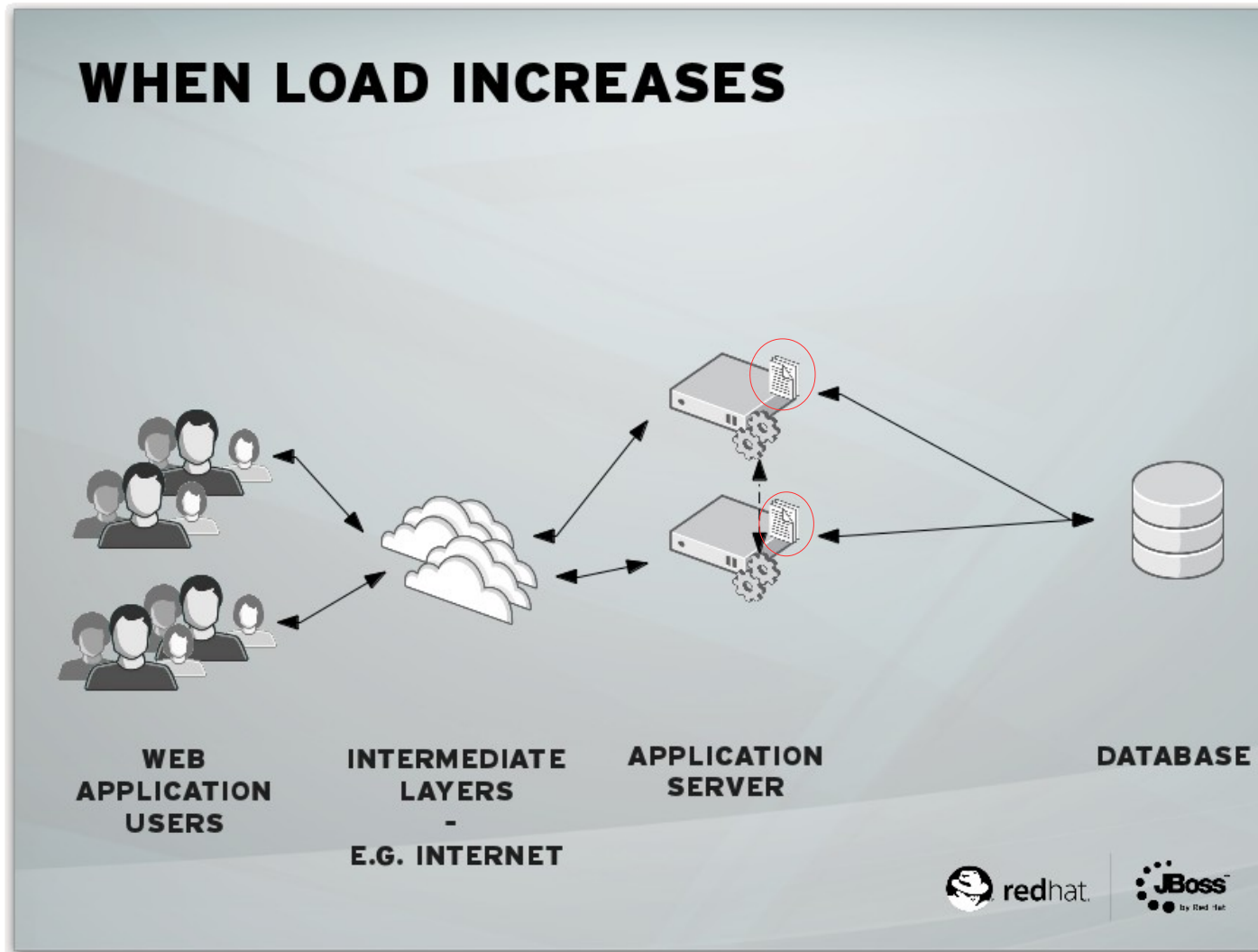
- Embedded (Library, In-VM mode)
- Remote (Client/Server mode)
  - REST (HTTP)
  - Hot Rod
  - Memcached
  - Websocket
  - OData



# Embedded (In-VM) mode - clustered



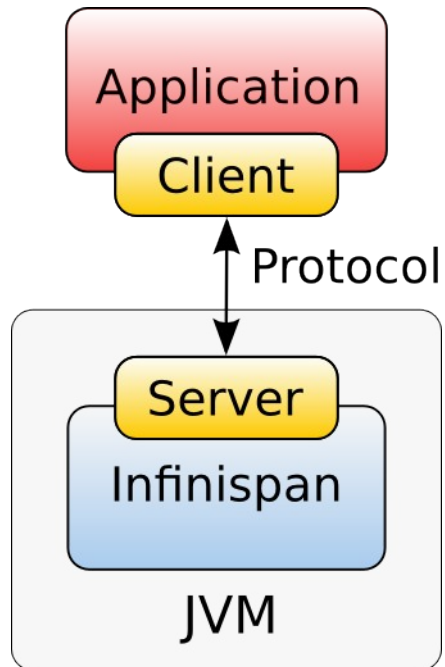
# Embedded (In-VM) mode - clustered



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



# Client / Server mode

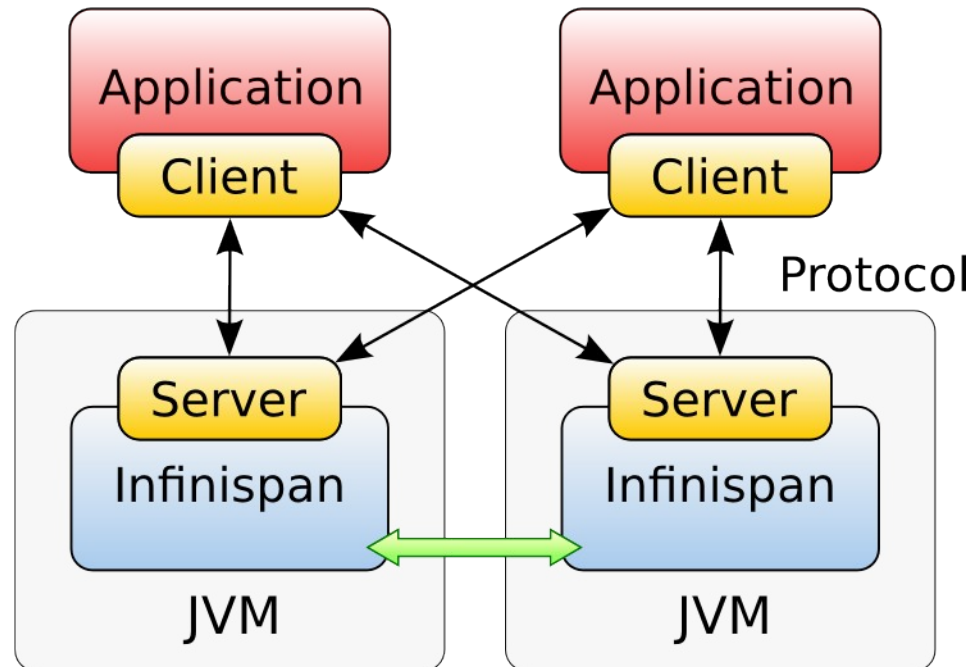


## Protocols

- REST
- Hot Rod
- Memcached
- Websocket
- OData

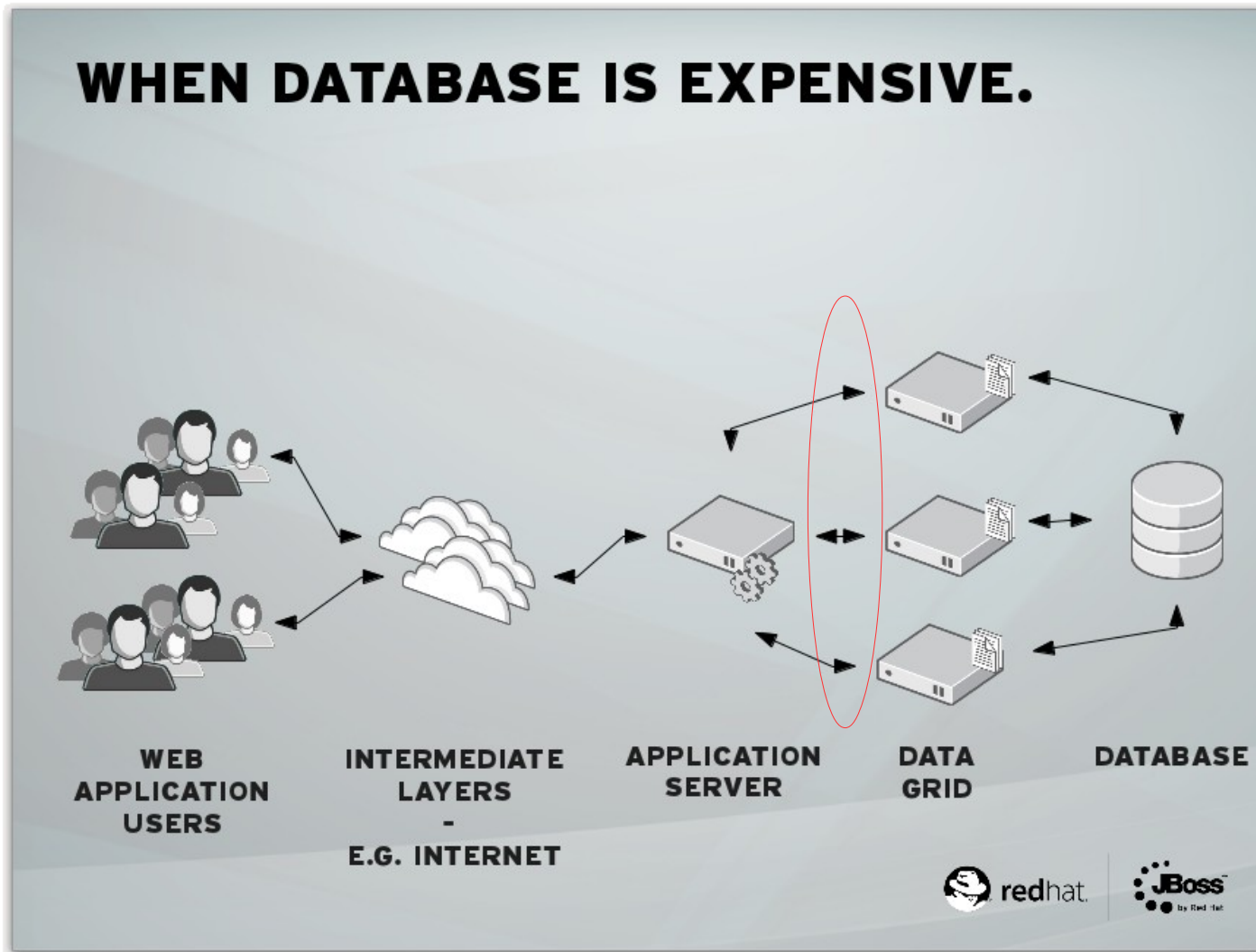


# Client / Server mode - clustered





# Client / Server mode - clustered

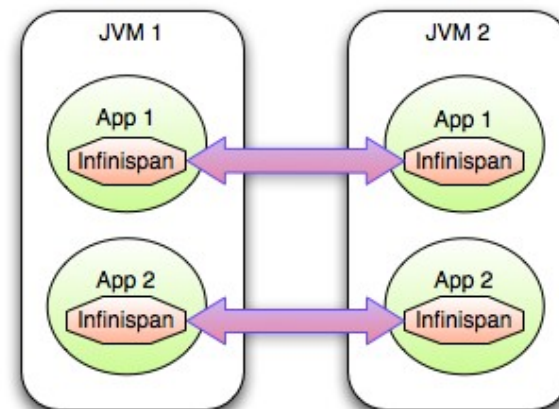
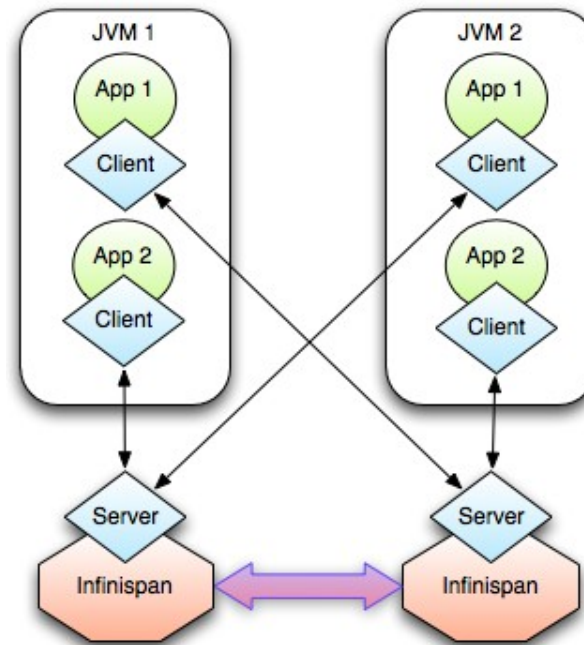


From <http://fhornain.wordpress.com/2012/04/21/jboss-data-grid-when-database-is-very-expensive/>

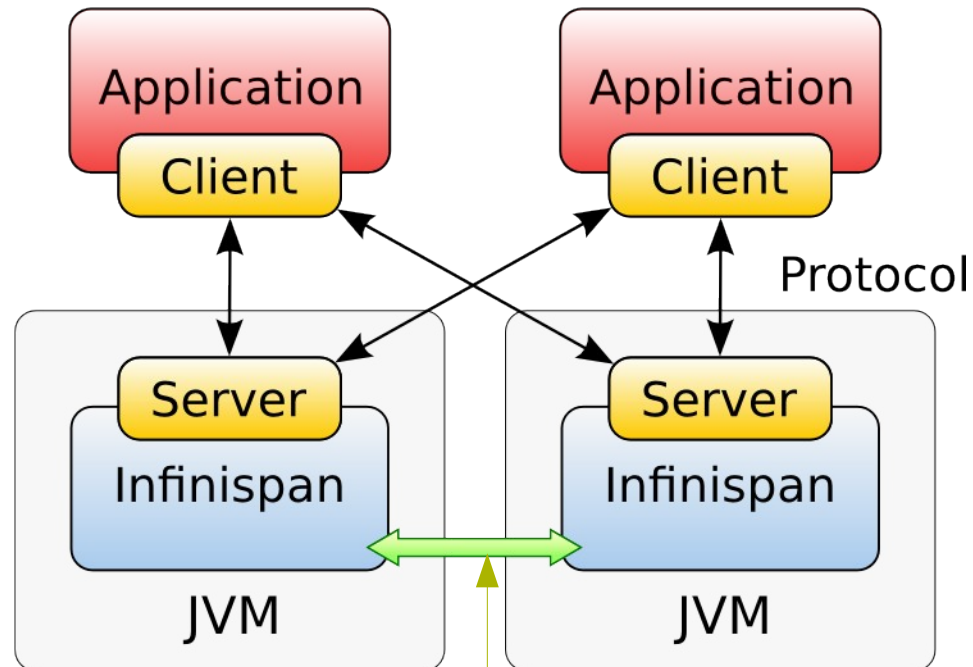


# Client / Server mode - clustered

- Independent tier management
- Independently deploy new app version



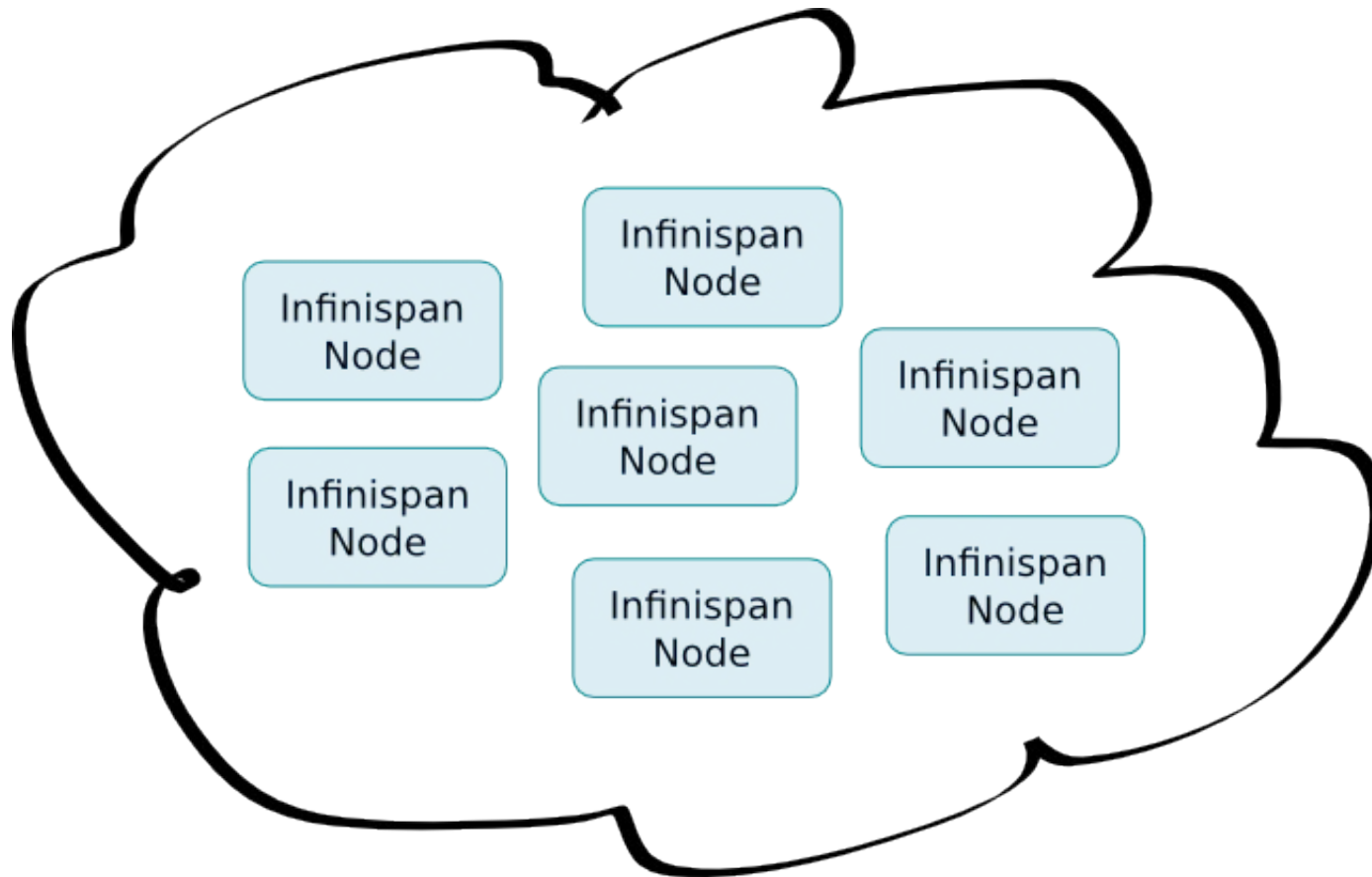
# Client / Server mode - clustered



**Big deal! JGroups**  
Toolkit for reliable messaging

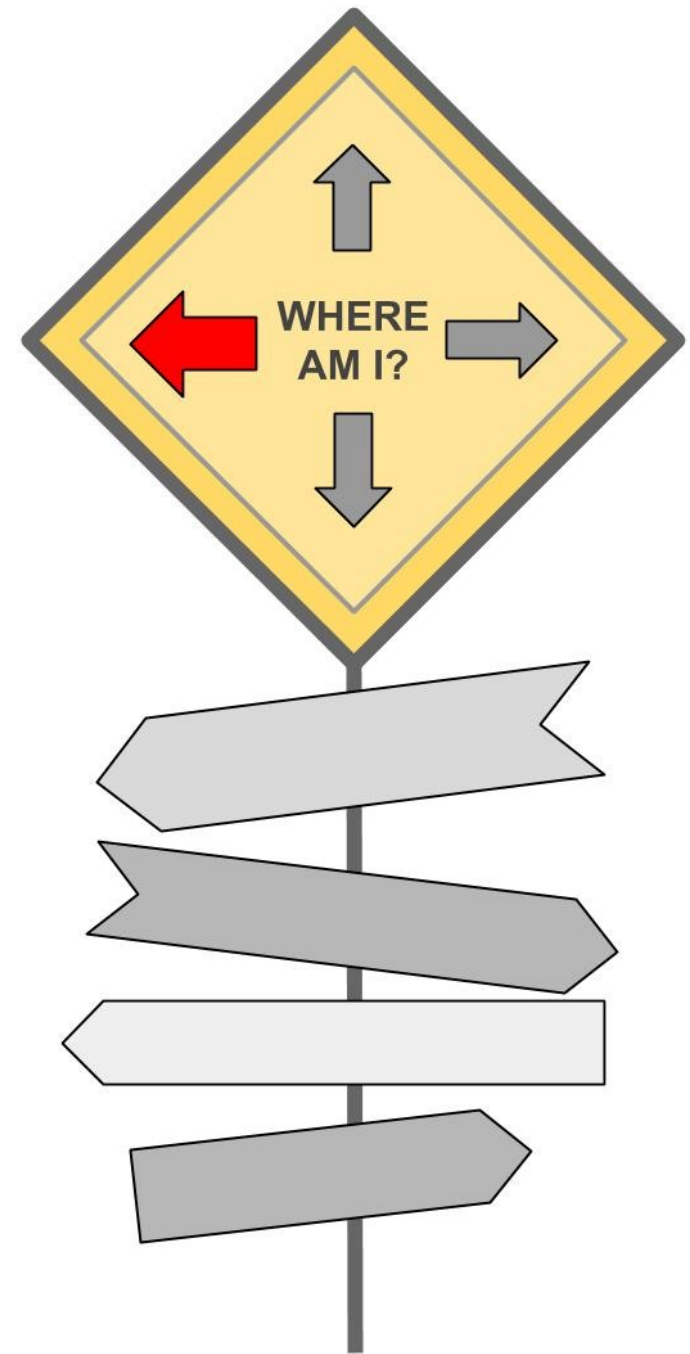


# What clustering / resilience / elasticity means



# Find your path...

- NoSQL world
- What's Infinispan
- Why / When to use it
- Plug it into your architecture
- **Infinispan clustering modes**
- Client / server access modes
- High level features
- Features in version 5.2, 6.0
- Brand new features in version 7.0



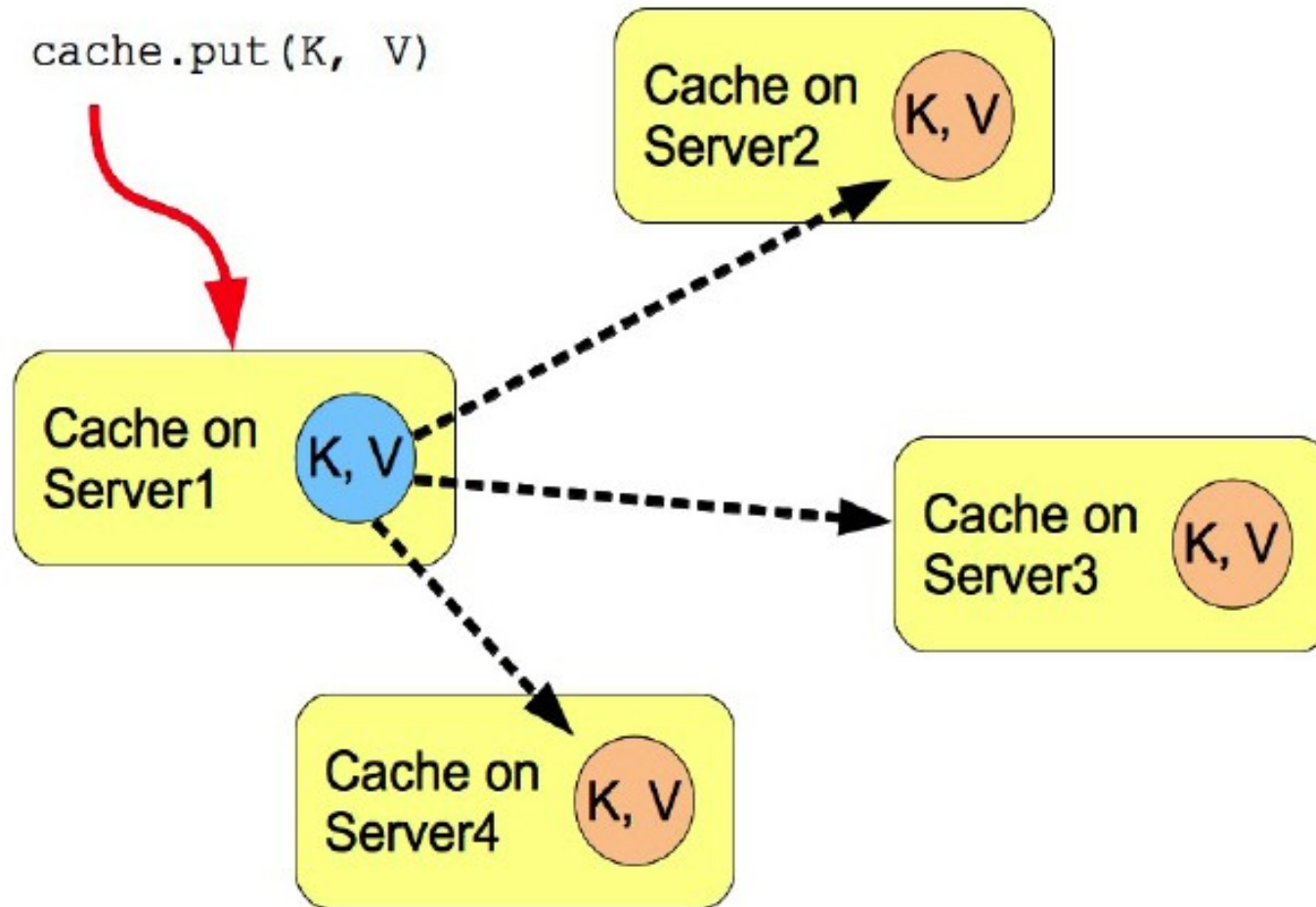


# Clustering modes

- **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



# Replication mode



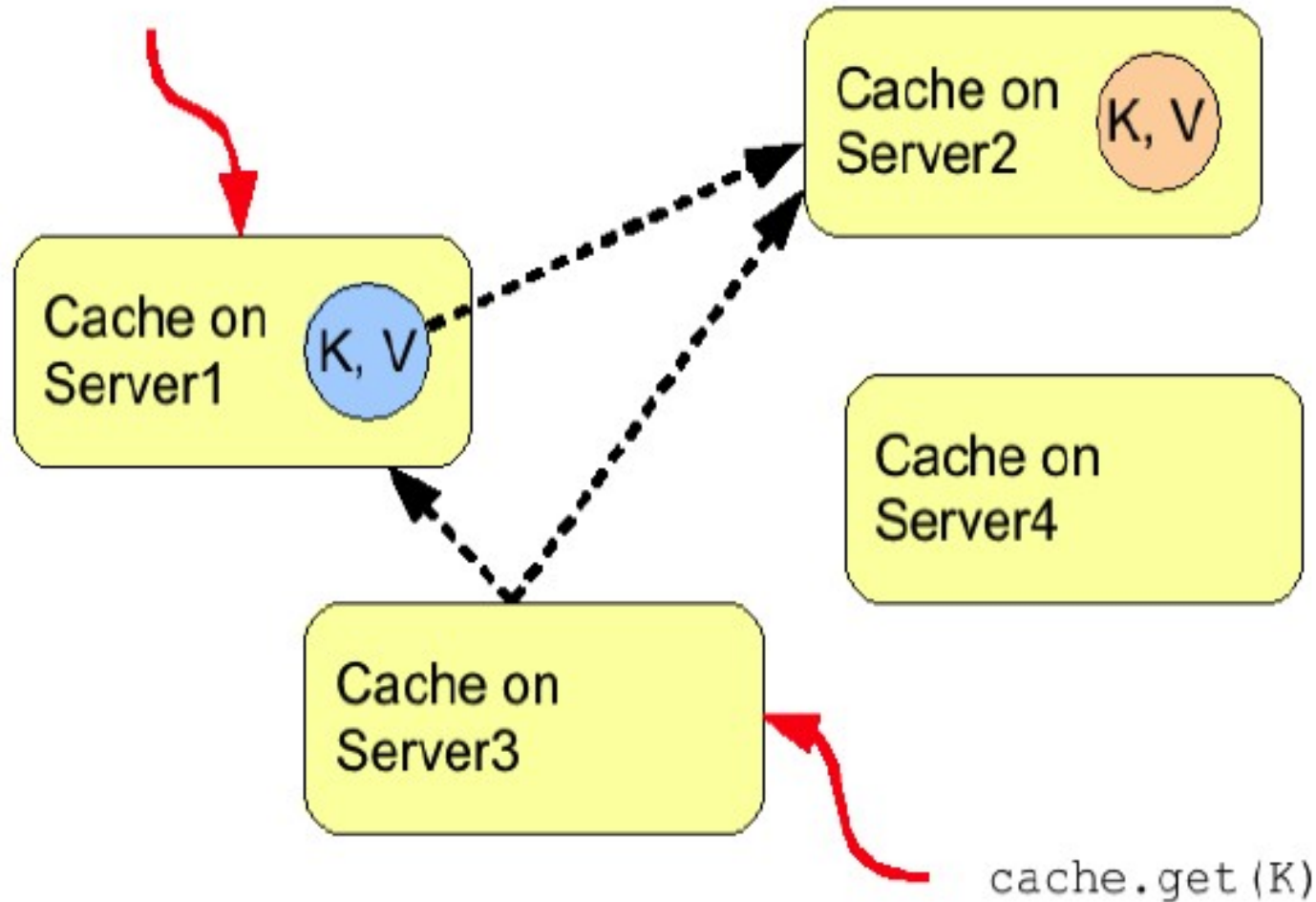
# Replication mode

- Advantages
  - N node cluster tolerates **N-1 failures**
  - **Read friendly** – no 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



# 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)



# Sync vs Async mode

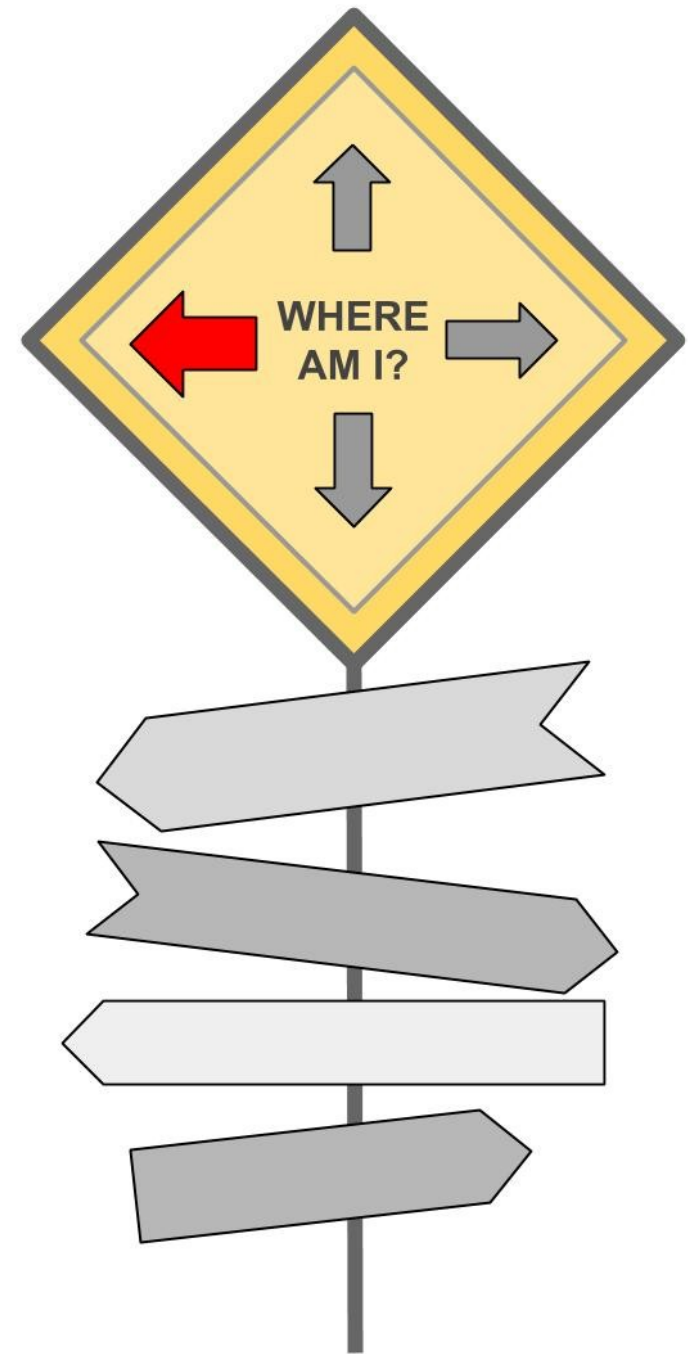
- Sync
  - All operations get confirmation that the other relevant cluster nodes reached the desired state
- Async
  - All operations block only until they perform local changes, we don't wait for JGroups responses.
  - Better throughput but no guarantees on data integrity in cluster.





# Find your path...

- NoSQL world
- What's Infinispan
- Why / When to use it
- Plug it into your architecture
- Infinispan clustering modes
- **Client / server access modes**
- High level features
- Features in version 5.2, 6.0
- Brand new features in version 7.0



# REST Server

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

e.g.

`http://localhost:8080/rest/default/abcd`

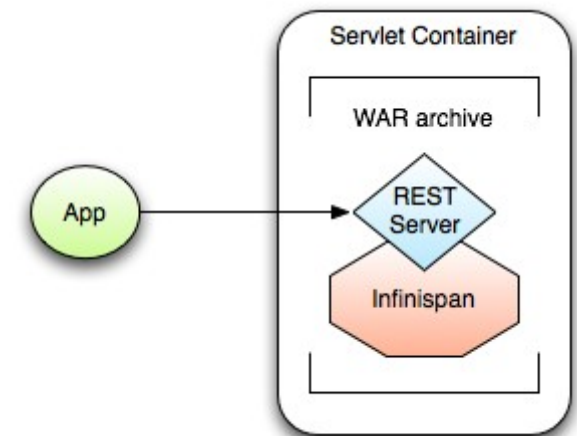
HTTP Methods supported:

HEAD, GET, PUT, POST, DELETE

Standard headers supported:

Content-Type  
Last-Modified

**Former: war deploy to AS**



**Now we have standalone Infinispan-server which is starting endpoints for clients.**



# 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/rest/my_cache/my_key
```

## GET

```
curl -X GET http://localhost:8080/rest/my_cache/my_key
```

## DELETE

```
curl -X DELETE http://localhost:8080/rest/my_cache/my_key
```



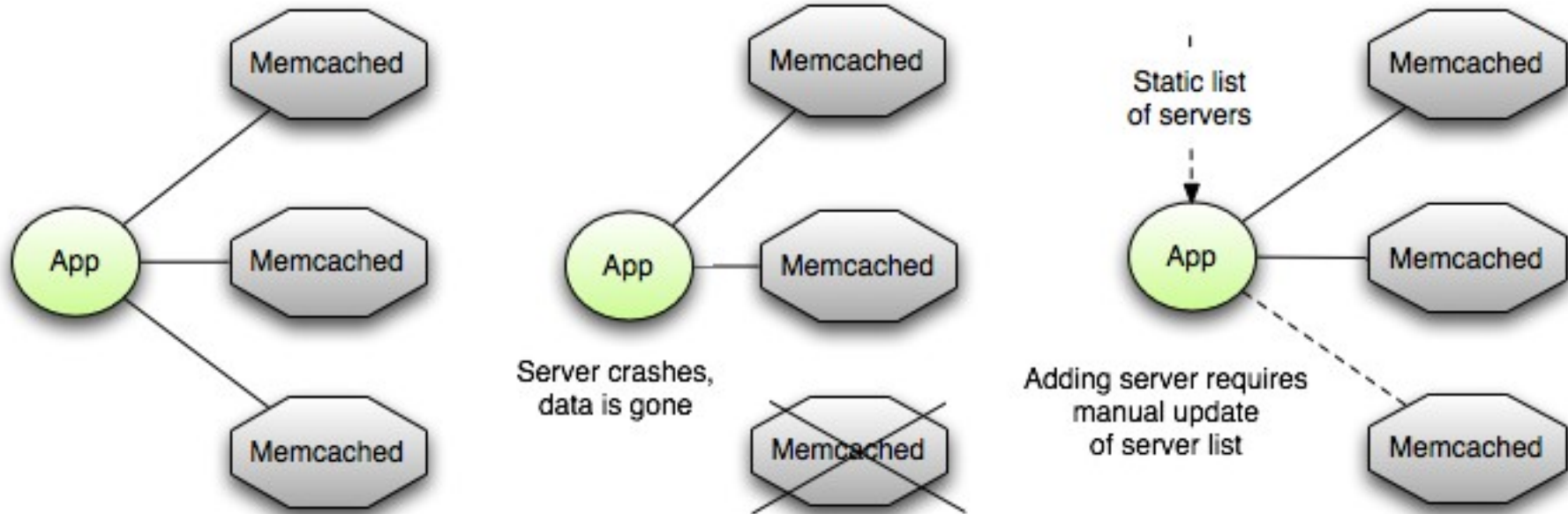
# Memcached

- 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

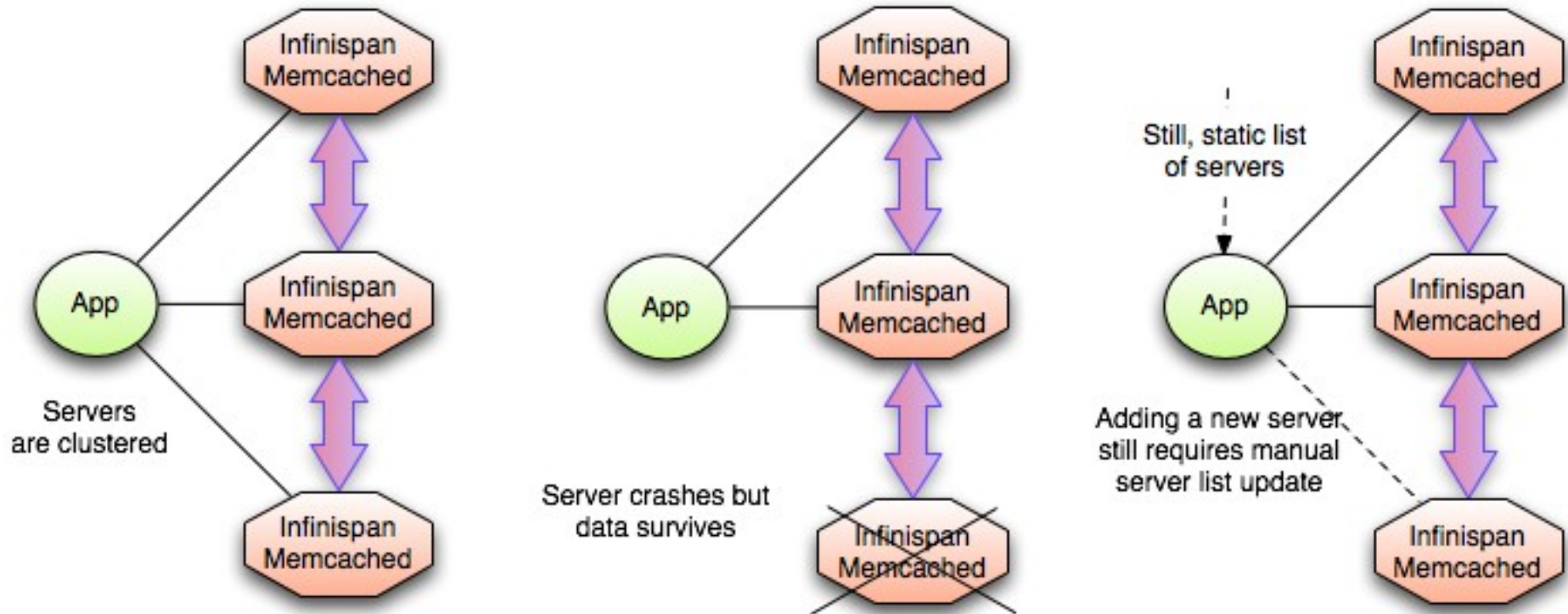




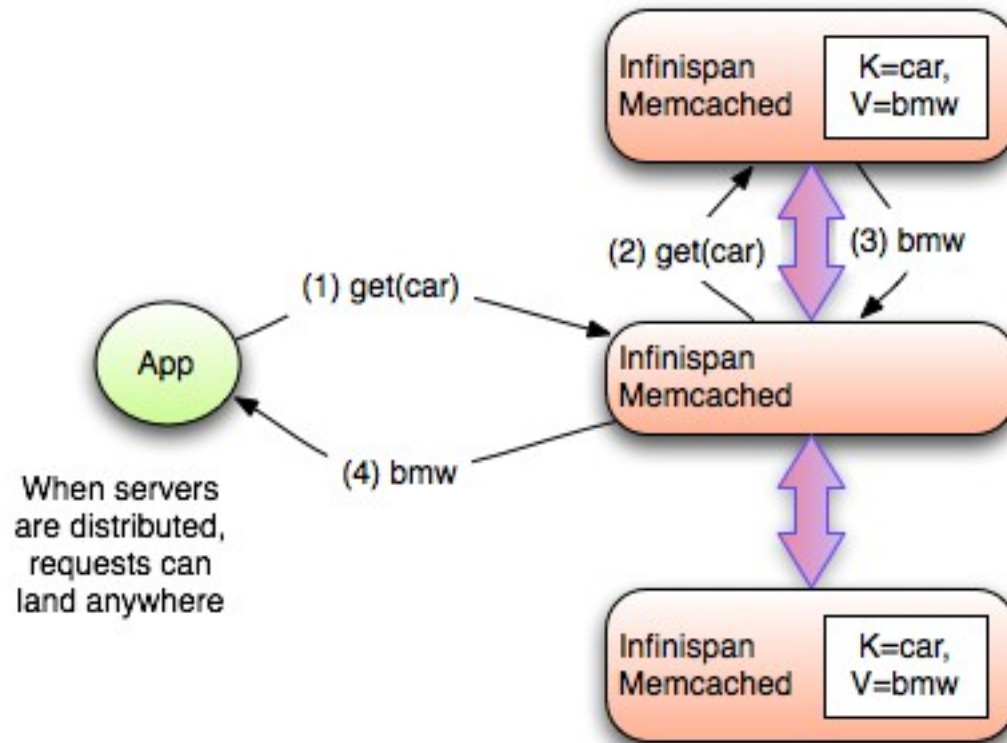
# Memcached server (original version)



# Memcached server (Infinispan implementation)

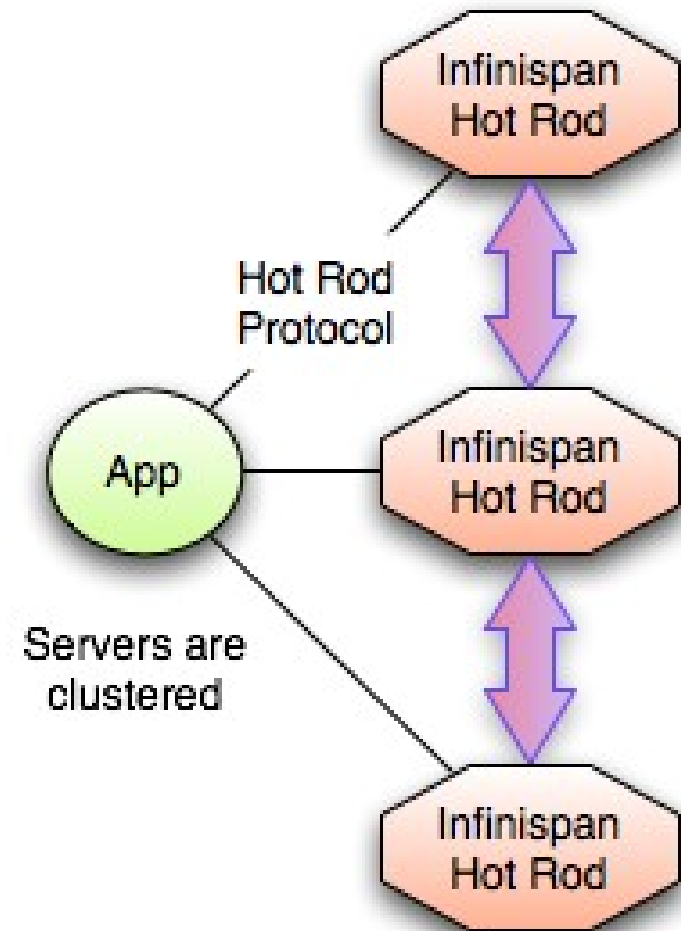


# Routing not so smart

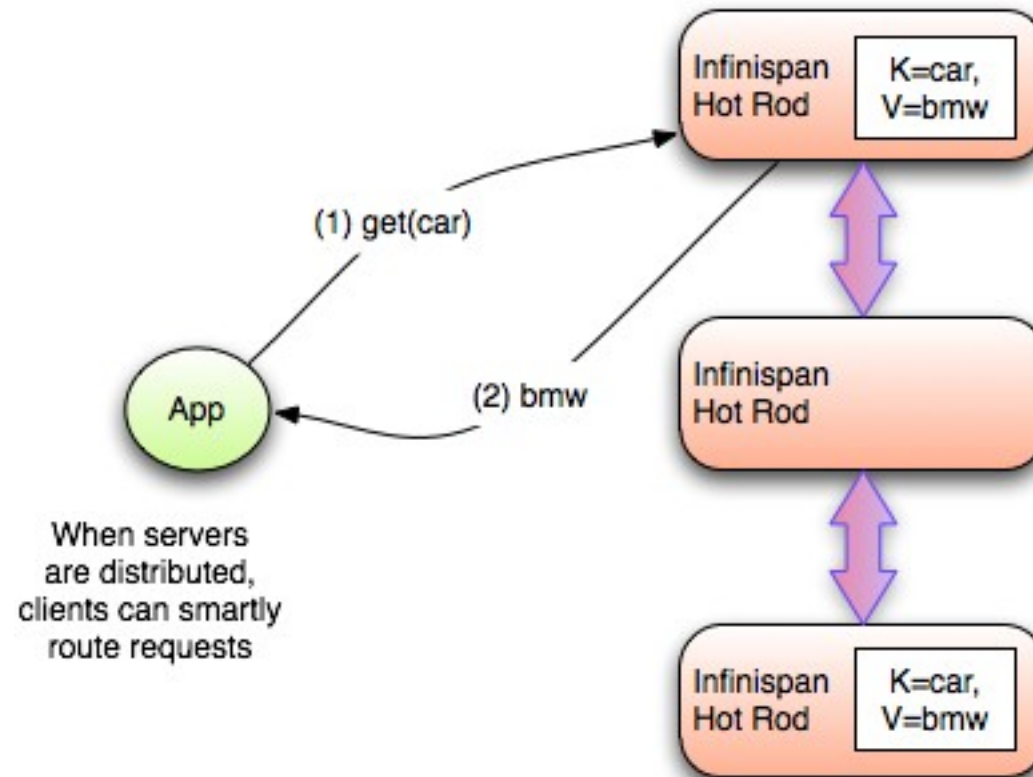


# Hot Rod

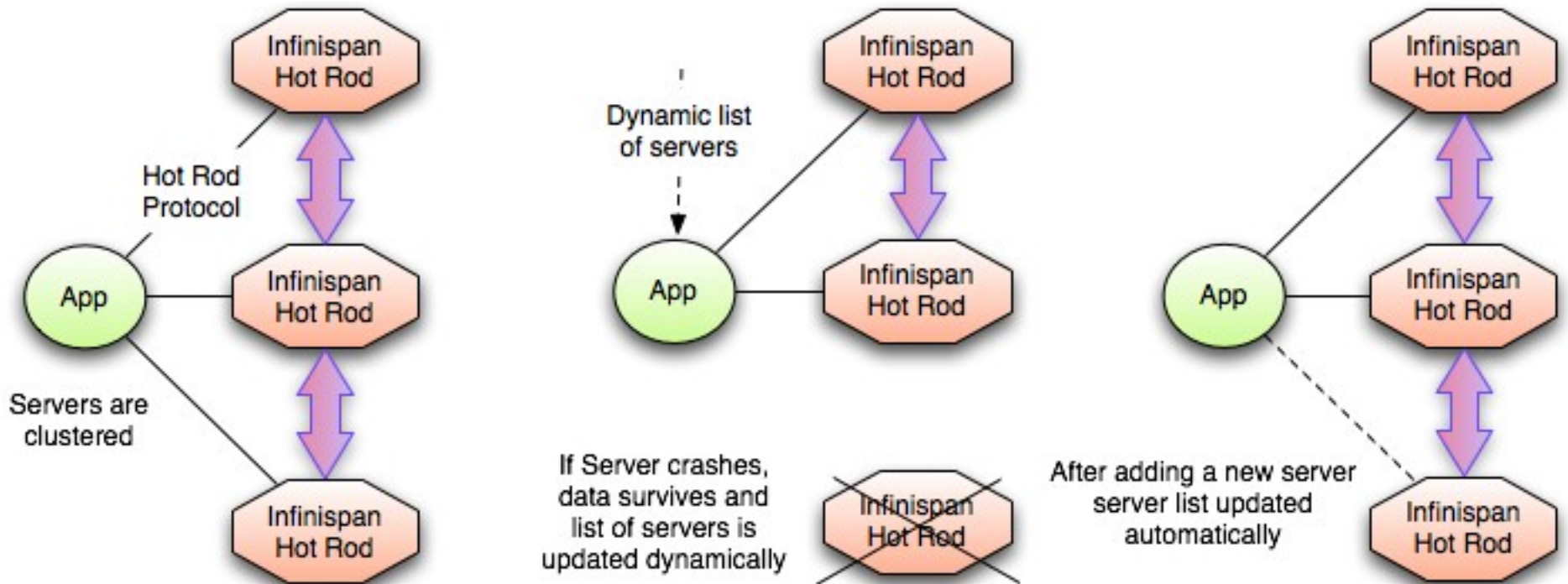
- Infinispan's own binary wire protocol
- Open and language independent
- Built-in dynamic failover and load balancing
- Smart routing
- Fast



# Smart routing with Hot Rod



# Dynamic routing with Hot Rod





# 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

+ **OData:** <http://tsykora-tech.blogspot.cz/2014/02/introducing-infinispan-odata-server.html>



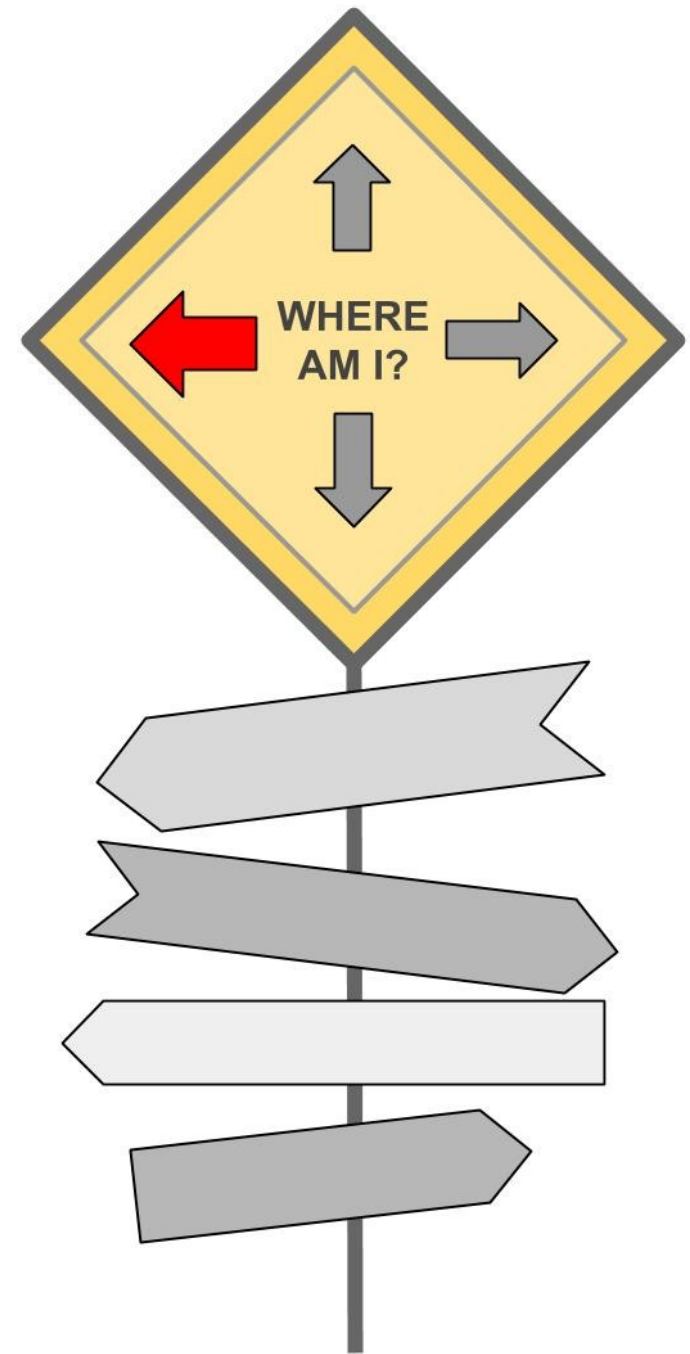
# For Java users: it's a Map (again)

```
// 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");
```



# Find your path...

- NoSQL world
- What's Infinispan
- Why / When to use it
- Plug it into your architecture
- Infinispan clustering modes
- Client / server access modes
- **High level features**
- Features in version 5.2, 6.0
- Brand new features in version 7.0



# Closer look on special features

- Eviction
- Expiration
- Cache stores
- Queries



# 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: [http://infinispan.org/docs/7.0.x/user\\_guide/user\\_guide.html#eviction\\_anchor](http://infinispan.org/docs/7.0.x/user_guide/user_guide.html#eviction_anchor)



# 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: [http://infinispan.org/docs/7.0.x/user\\_guide/user\\_guide.html#\\_expiration](http://infinispan.org/docs/7.0.x/user_guide/user_guide.html#_expiration)





# Features – Cache stores

- Store data from memory to other kind of storage
  - File System
    - SingleFileStore – basic FS store implementation
    - SoftIndexFileStore
  - Relational Database
    - JdbcBinaryCacheStore – hash of whatever
    - JdbcStringBasedCacheStore – String (needs mapping)
  - Other NoSQL stores
    - [Cassandra](#)
    - RemoteCacheStore – store to another Infinispan grid



# Features – Cache stores

Passivation	Eviction	Behaviour
OFF	OFF	<b>P = M (Write through)</b> whenever an element is modified, added or removed, then that modification is persisted in the backend store
OFF	ON	<b>P <math>\supseteq</math> M (Write behind)</b> 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	<b>P <math>\cap</math> M = <math>\emptyset</math></b> 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 - Querying

## Map / Key-Value approach:

Key1 --> Value1

Key2 --> Value2 (dictionary like)

**Querying mechanism allows you to query  
Infinispan over the values!**

? “data where name='infinispan' and ...” --> **Value1, Value2**

(Apache Lucene, Hibernate search)



# Features - Querying

// example values stored in the cache and indexed:

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

//to be indexed the object needs @Indexed annotation:

```
@Indexed
```

```
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: [http://infinispan.org/docs/7.0.x/user\\_guide/user\\_guide.html#\\_querying\\_infinispan](http://infinispan.org/docs/7.0.x/user_guide/user_guide.html#_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: [http://infinispan.org/docs/7.0.x/user\\_guide/user\\_guide.html#\\_querying\\_infinispan](http://infinispan.org/docs/7.0.x/user_guide/user_guide.html#_querying_infinispan)



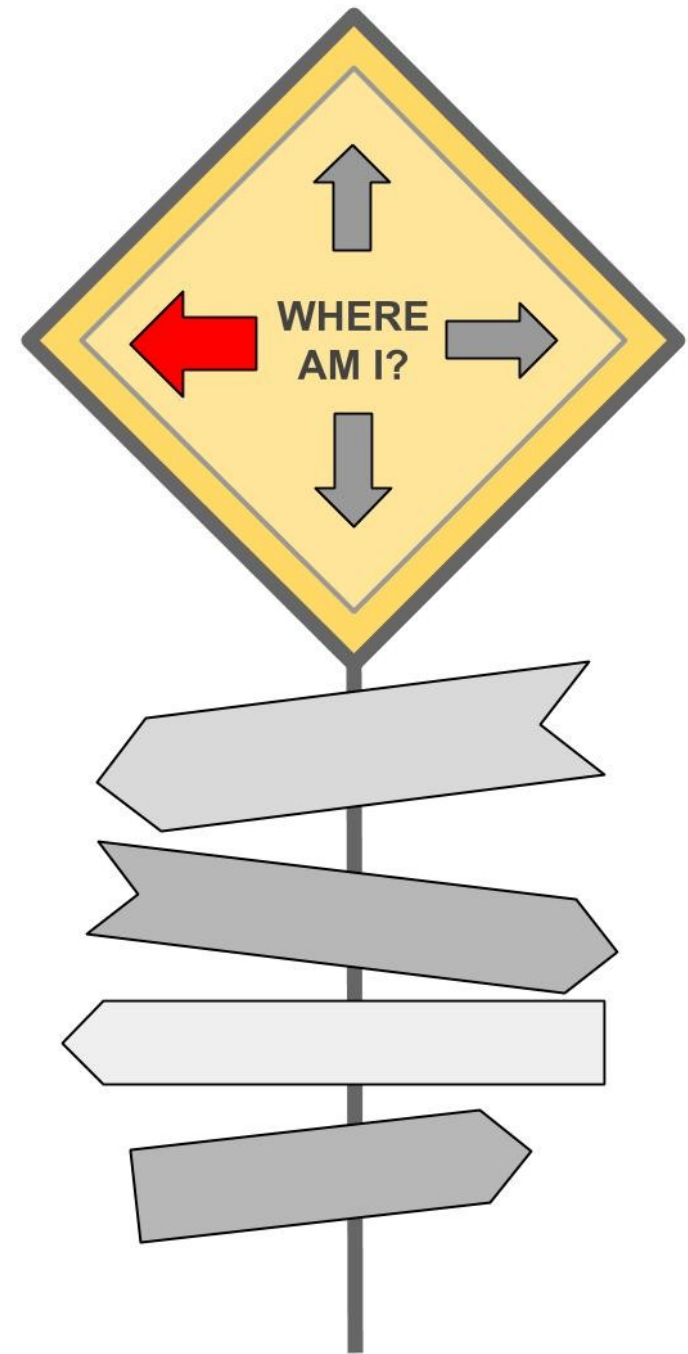
# Features – Others

- Management via RHQ (<http://rhq-project.org>)
- JMX Statistics (writes, reads, hits, misses)
- CDI, injection of Cache, RemoteCache
- Tree API (storing in hierarchical way, /persons/john)
- ... and more on next slides (v5.2, v6.0 & v7.0)



# Find your path...

- NoSQL world
- What's Infinispan
- Why / When to use it
- Plug it into your architecture
- Infinispan clustering modes
- Client / server access modes
- High level features
- **Features in version 5.2, 6.0**
- Brand new features in version 7.0





# Interesting features in version 5.2

- Command line interface (ispn-cli.sh)

See next slides for:

- Non-blocking state transfer
- Cross site replication
- Rolling upgrades (HotRod only)

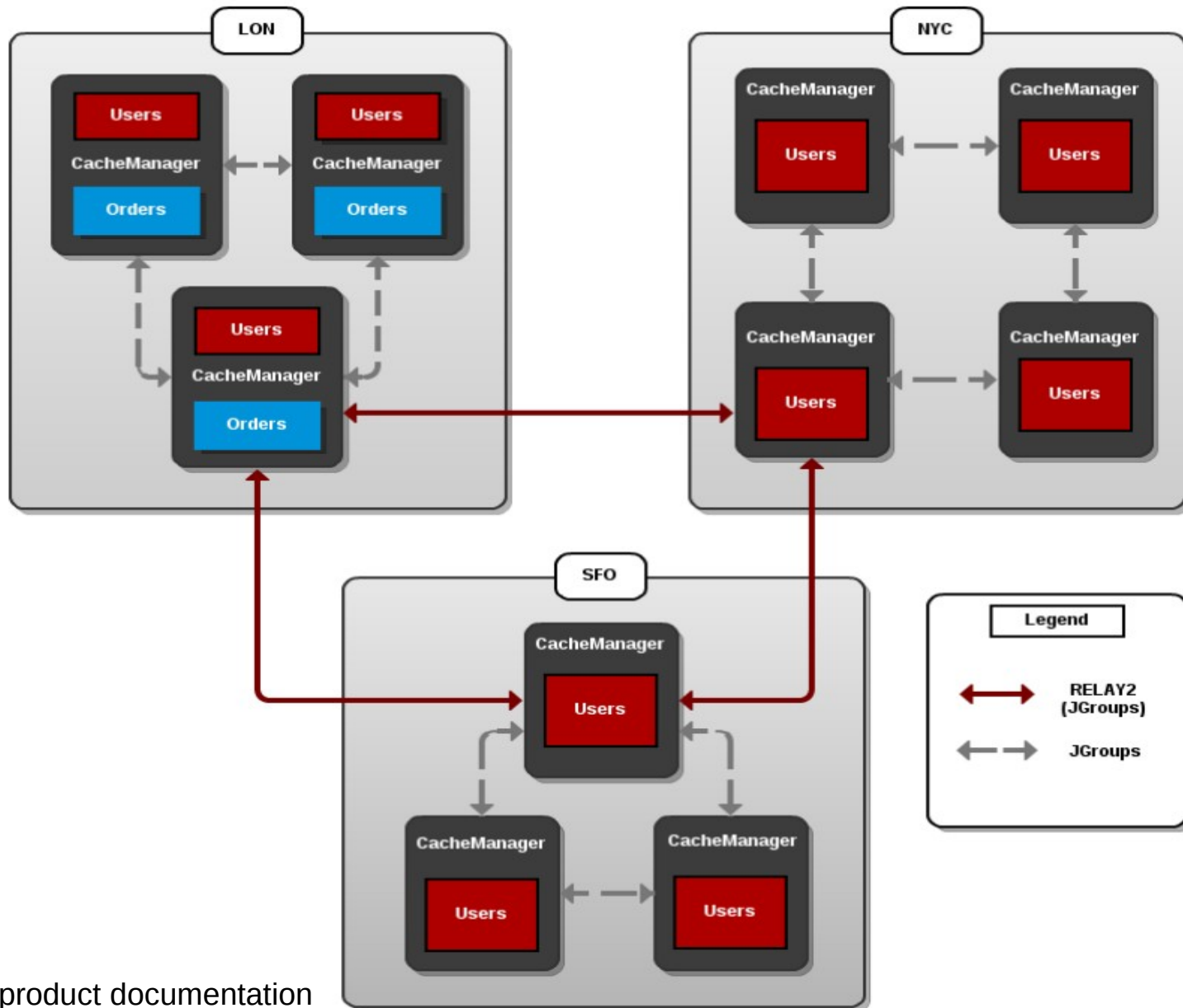


# Non-blocking state transfer

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



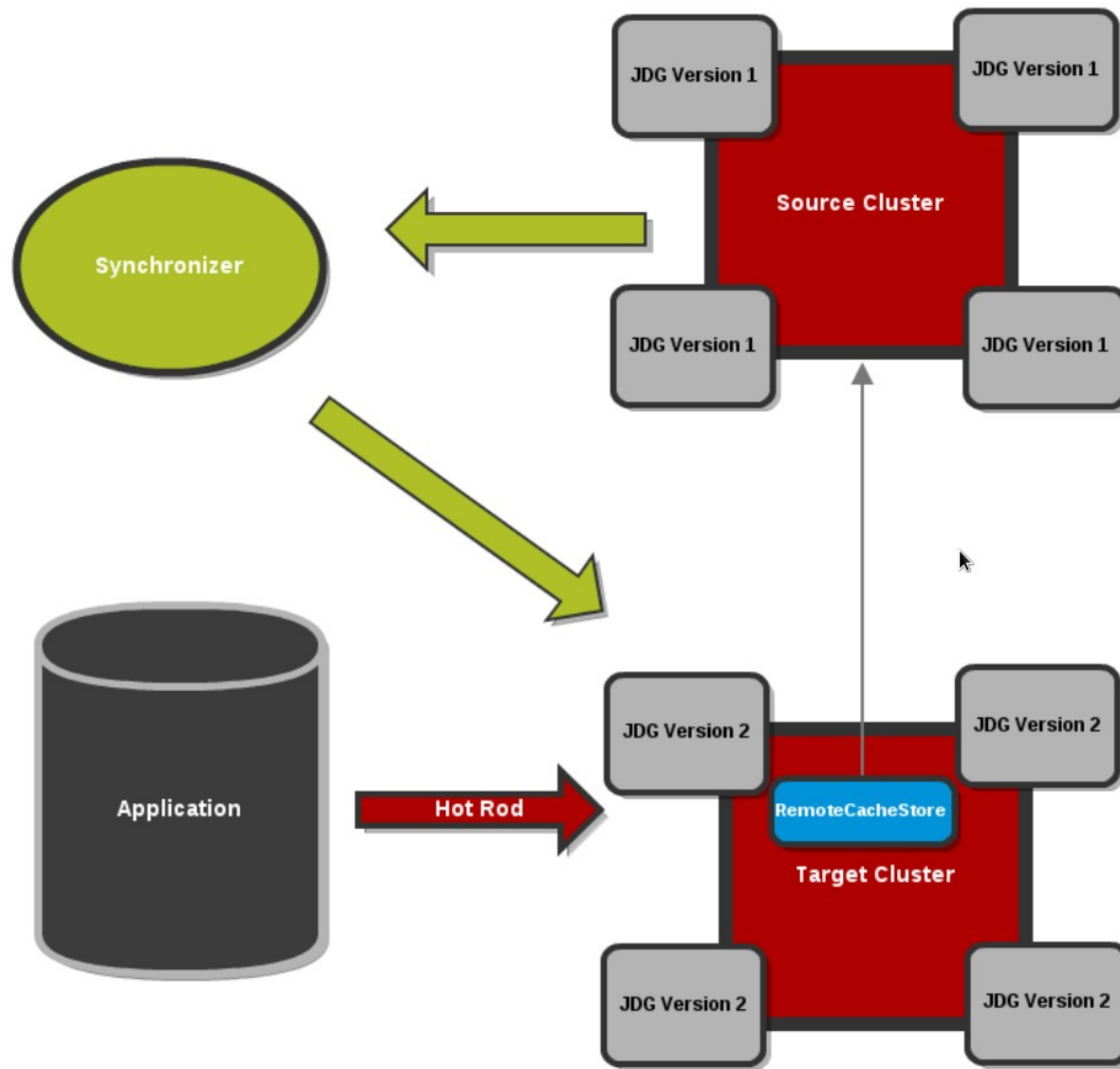
# Cross site replication



Source: JDG product documentation



# Rolling upgrades



Source: JDG product documentation



# New features in version 6.0

- Rolling upgrades for REST clusters
- C++ HotRod client
  - C++ apps can read and write data to remote cluster
- New high performance file cache store
  - in-memory indexes of keys for persisted entities
- Cross data center replication scalability
  - Multiple site-masters, replication between clusters



# New features in version 6.0

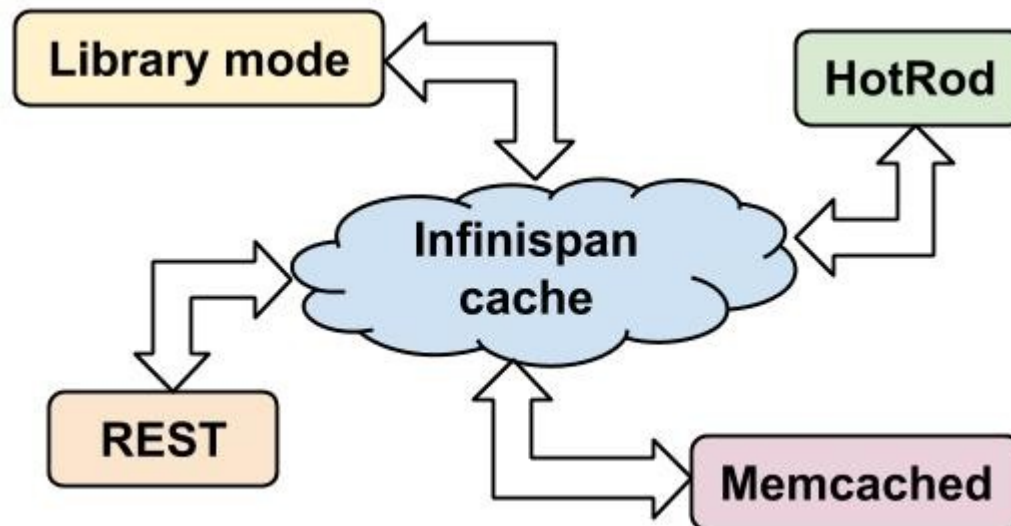
See next slides for:

- Compatibility mode
- Remote queries via Java HotRod client



# Compatibility mode

- Interoperability between Embedded and Remote Server Endpoints
- **Put** using REST, **get** using HotRod? No problem!
- Disabled by default (conversion has its cost)





# Remote queries via HotRod

- It **was** possible to index and search **only** Java entities...
- **Now**, we have support for:
  - Remote and language neutral querying
- New query language (DSL – Domain Specific Language)
- Structure of entities is known to both server and client
  - Google's Protocol Buffers encoding format
  - Entities declared by using .proto files

Read more:

<https://developers.google.com/protocol-buffers/docs/overview>

[http://infinispan.org/docs/7.0.x/user\\_guide/user\\_guide.html#\\_querying\\_via\\_the\\_java\\_hot\\_rod\\_client](http://infinispan.org/docs/7.0.x/user_guide/user_guide.html#_querying_via_the_java_hot_rod_client)



# Remote queries via HotRod (data structure definition)

```
library.proto
```

```
package book_sample;
```

```
message Book {  
    required string title = 1;  
    required string description = 2;  
    // no native Date type available in Protobuf  
    required int32 publicationYear = 3;  
    repeated Author authors = 4;  
}
```

```
message Author {  
    required string name = 1;  
    required string surname = 2;  
}
```



```

import org.infinispan.client.hotrod.*;
import org.infinispan.query.dsl.*;
...

RemoteCacheManager remoteCacheManager = ...;
RemoteCache<Integer, Book> remoteCache = remoteCacheManager.getCache();

Book book1 = new Book();
book1.setTitle("Hibernate in Action");
remoteCache.put(1, book1);

Book book2 = new Book();
book2.setTitle("Infinispan Data Grid Platform");
remoteCache.put(2, book2);

QueryFactory qf = Search.getQueryFactory(remoteCache);
Query query = qf.from(Book.class)
    .having("title").like("%Infinispan%").toBuilder()
    .build();

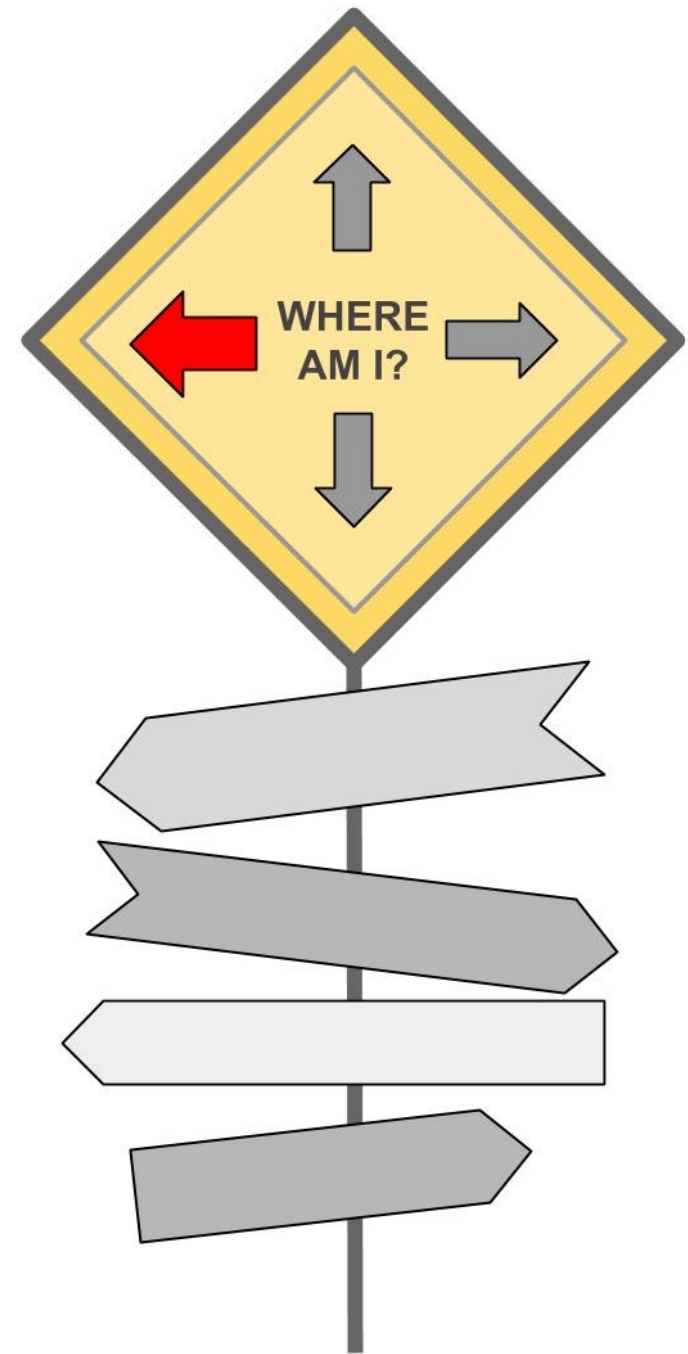
List<Book> list = query.list(); // Voila! We have our book back from the cache!

```



# Find your path...

- NoSQL world
- What's Infinispan
- Why / When to use it
- Plug it into your architecture
- Infinispan clustering modes
- Client / server access modes
- High level features
- Features in version 5.2, 6.0
- **Brand new features in version 7.0**



# New features in version 7.0

- X-Site clustering
  - Former: when node joins, only new updates was replicated
  - It is possible to do state-transfer of existing data (in cache)
- Partitioning handling
  - Countermeasures against cluster split-brain and handling
- Security
  - Role based access
- Uber JARs (comfortable usability)



# Uber JARs

infinispan-commons.jar

infinispan-cachestore-jdbc.jar

infinispan-core.jar

**infinispan-embedded.jar**

infinispan-cdi.jar

infinispan-cachestore-jpa.jar

infinispan-cachestore-leveldb.jar

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-embedded</artifactId>
  <version>${infinispan.version}</version>
</dependency>
```



# Uber JARs

```
<dependency>  
  <groupId>org.infinispan</groupId>  
  <artifactId>infinispan-embedded</artifactId>  
  <version>${infinispan.version}</version>  
</dependency>
```

```
<dependency>  
  <groupId>org.infinispan</groupId>  
  <artifactId>infinispan-embedded-query</artifactId>  
  <version>${infinispan.version}</version>  
</dependency>
```

```
<dependency>  
  <groupId>org.infinispan</groupId>  
  <artifactId>infinispan-remote</artifactId>  
  <version>${infinispan.version}</version>  
</dependency>
```





# Quick review... the end is coming...

- NoSQL world
- What's Infinispan
- Why / When to use it
- How to plug it into your architecture
- Clustering modes
- Client-server access modes / protocols
- High level features
- Features in version 5.2, 6.0
- Brand new features in version 7.0



# Questions?



# Thank you!

