redhat. | JBoss by Red Hat

Java. Cloud. Leadership.

<Add your title>

Name
Title
Red Hat, Inc.
Date

Agenda

- Introduction
  - What is Infinispan?
  - Principle use cases
  - Key features
- Hands-on demo
  - build an application using infinispan
- Extras
  - Querying the Grid
  - Database - OGM
  - Performance tuning - RadarGun
- Conclusion

Lab Setup

- Download the lab zip:

  http://bit.ly/infinispan-labs-checkpoint1

- Unzip the lab to your disk to a location of your choice
- If you are a git user, you can clone the repository:

```
git clone git://github.com/pmuir/infinispan-labs.git
```

- each stage of this lab has a checkpoint which is tagged, you can check out the code for each Checkpoint using:

```
git checkout Checkpointx
```

Lab Setup

- Follow along using

  http://bit.ly/infinispan-labs

- Download JBoss AS 7.0.2 from

  http://jboss.org/jbossas/downloads

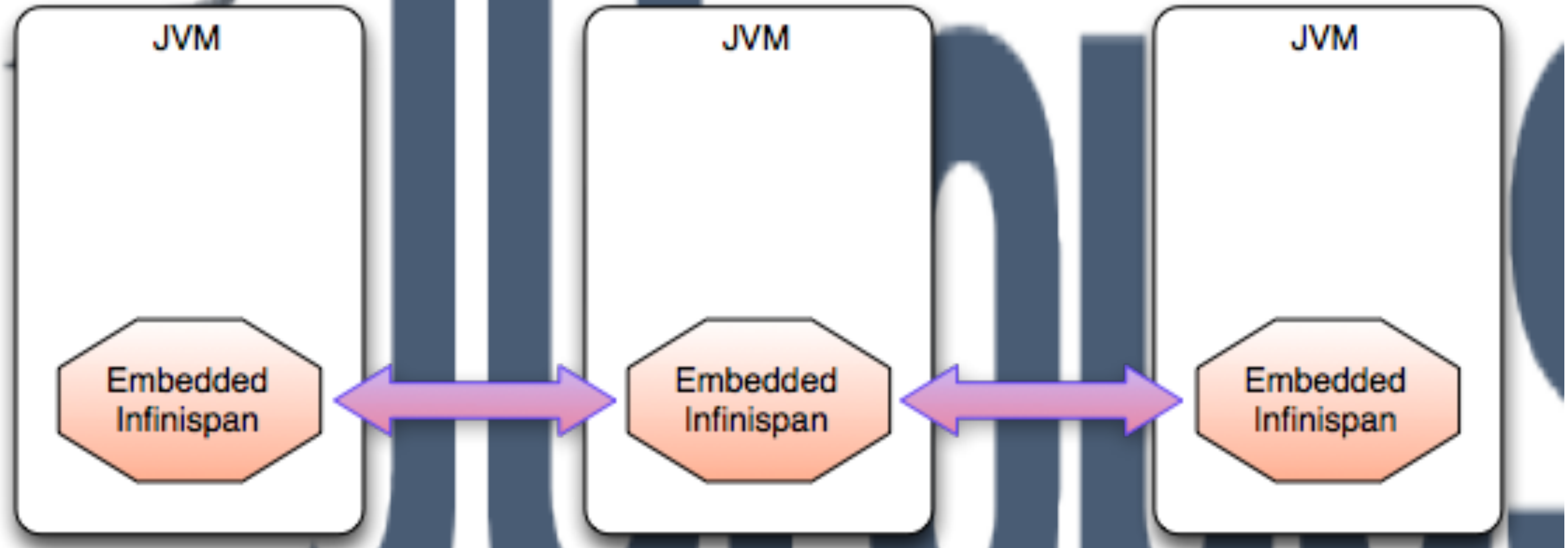
  - Unzip JBoss AS to your disk to a location of your choice

Introduction

So what is Infinispan?

- Distributed, in memory, data structure
- Highly available
- Elastic
- Open source

Distributed Data structure



High availability

- Memory is volatile
- Make redundant copies
  - Total replication (Replication Mode)
  - Partial replication (Distribution Mode)
- Topology changes
  - Node will crash!
  - Re-arrange state

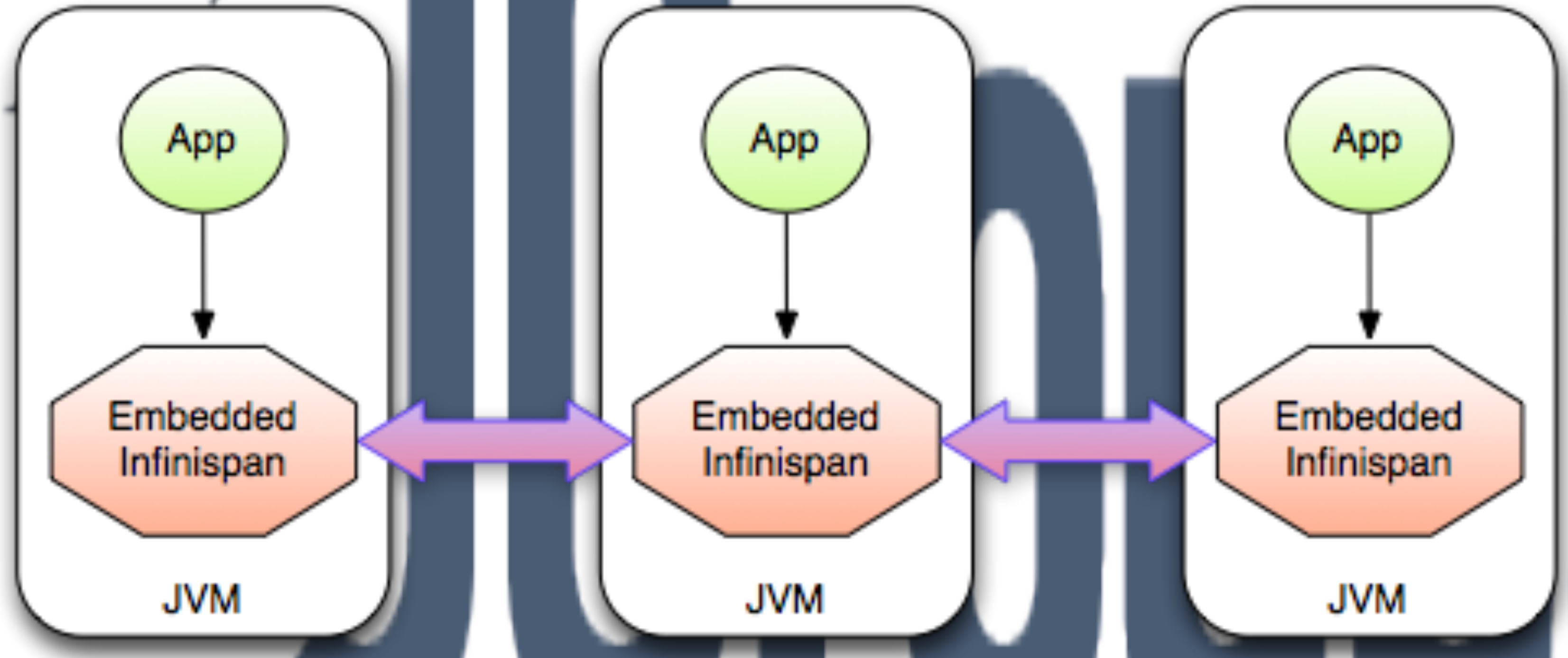
Elasticity

- Expect
  - Node additions
  - Node removals
- Topology changes
  - are totally consistent
  - do not "stop the world"

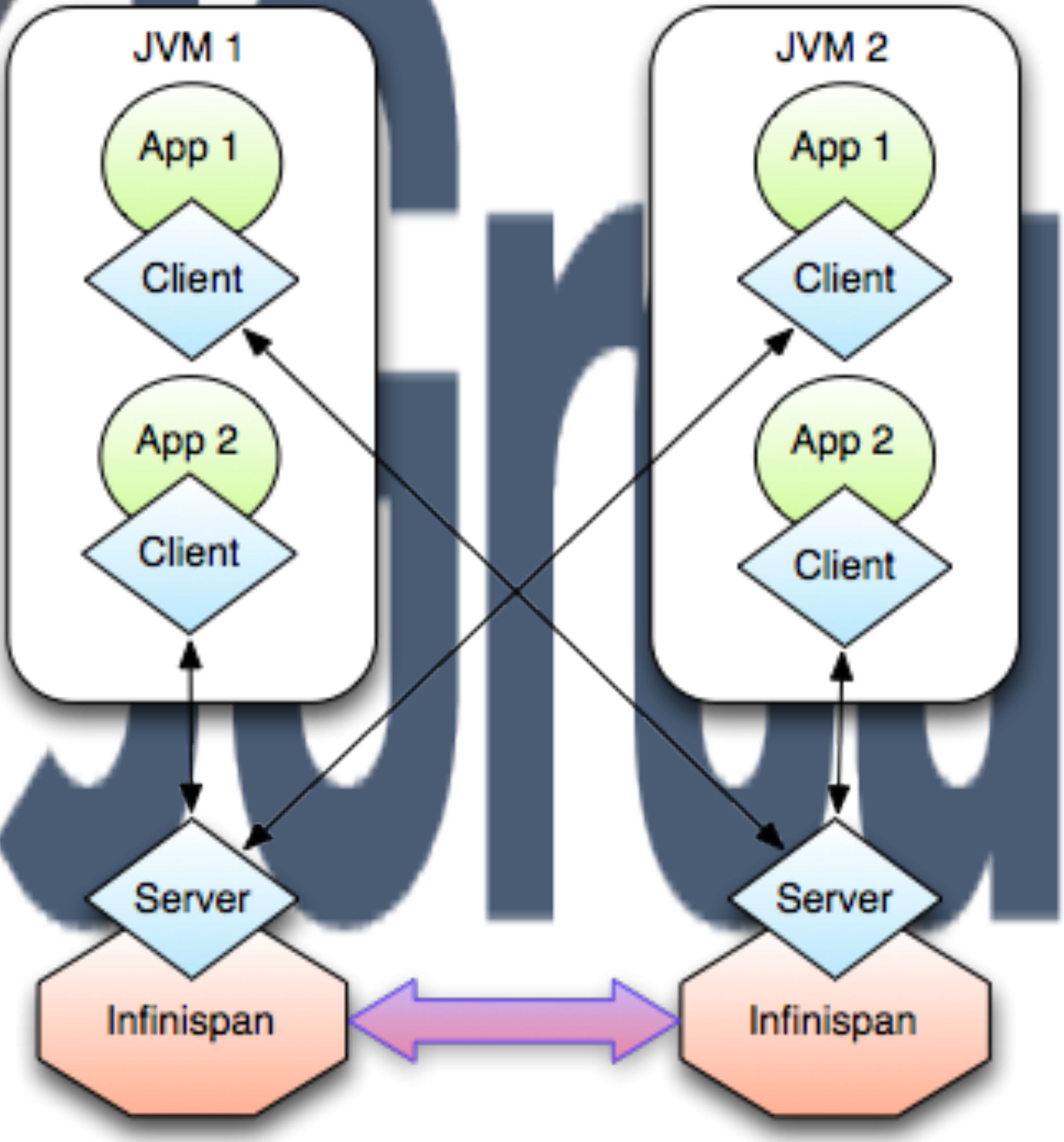
Access modes

- Embedded
  - client and node on same VM
  - fast!
- Client/server
  - different processes
  - multiple protocols
    - REST
    - Memcached
    - Hotrod

Embedded access



Client/server access



Server endpo
- REST
- Memcached
- Hotrod

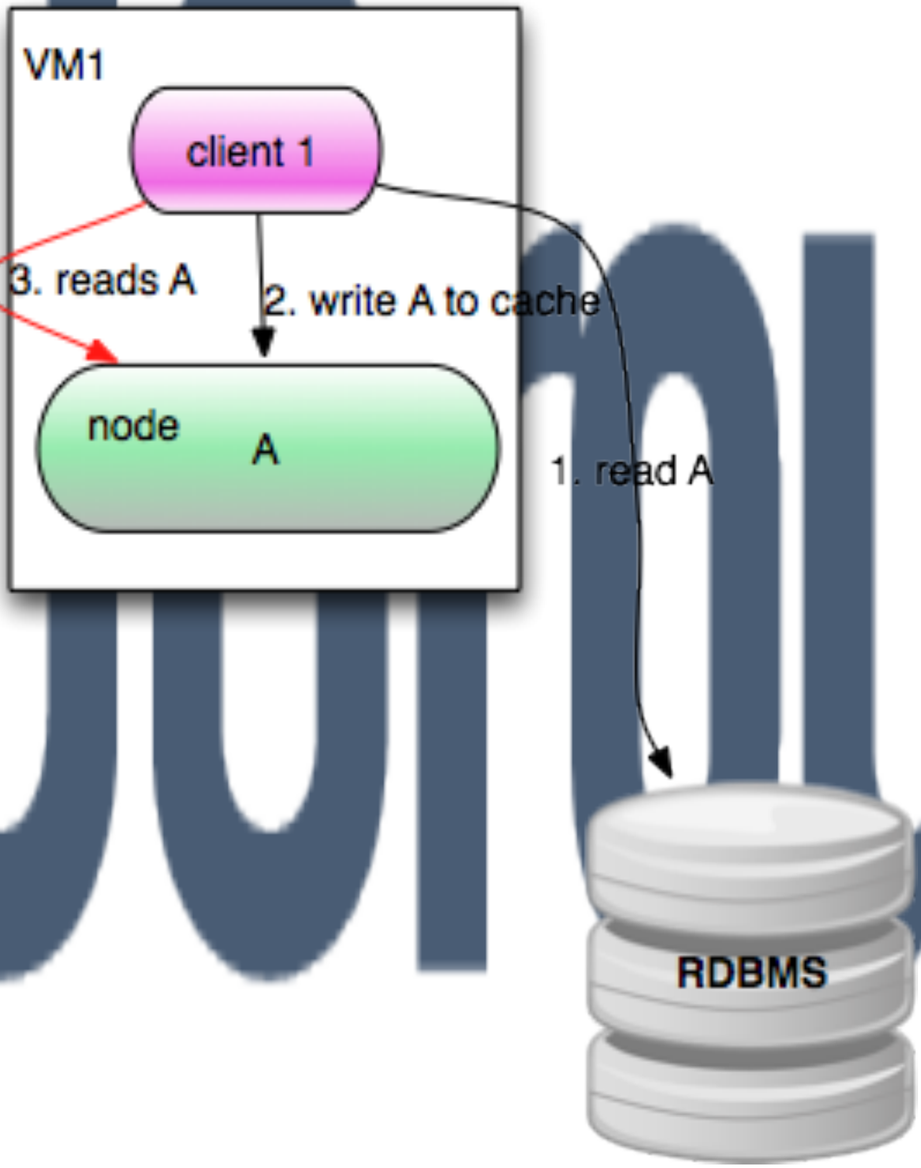
Main use cases

- Local cache
  - e.g. Hibernate 2nd level cache
- Cluster of caches
  - More caching capacity
  - Co-located clients
- Data Grid
  - dedicated cluster of servers
  - remote access

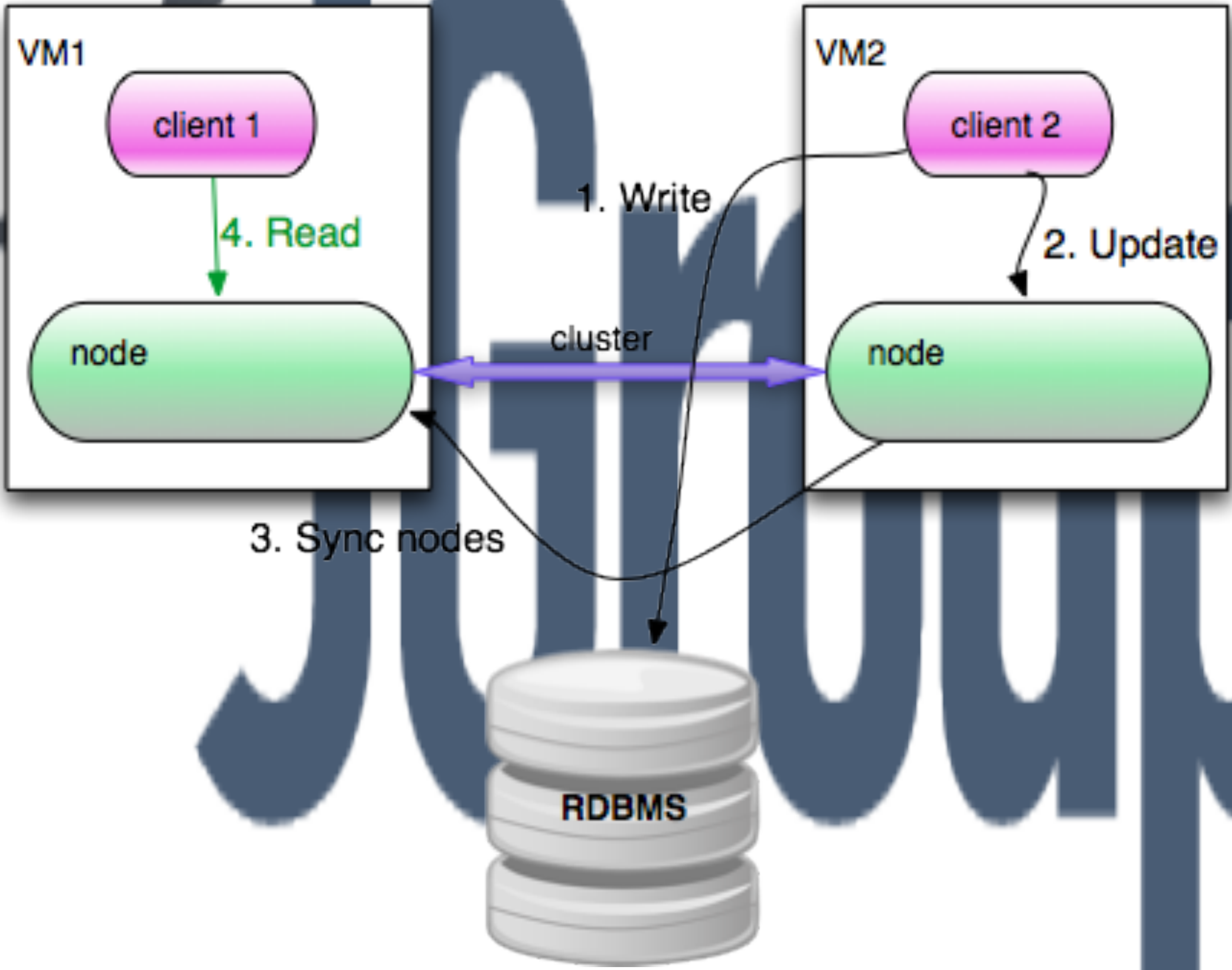
Good old caching...

- Local cache
  - java.util.Map
- And some more
  - eviction
  - expiry
  - write through/behind
  - passivation
  - preloading
  - notifications

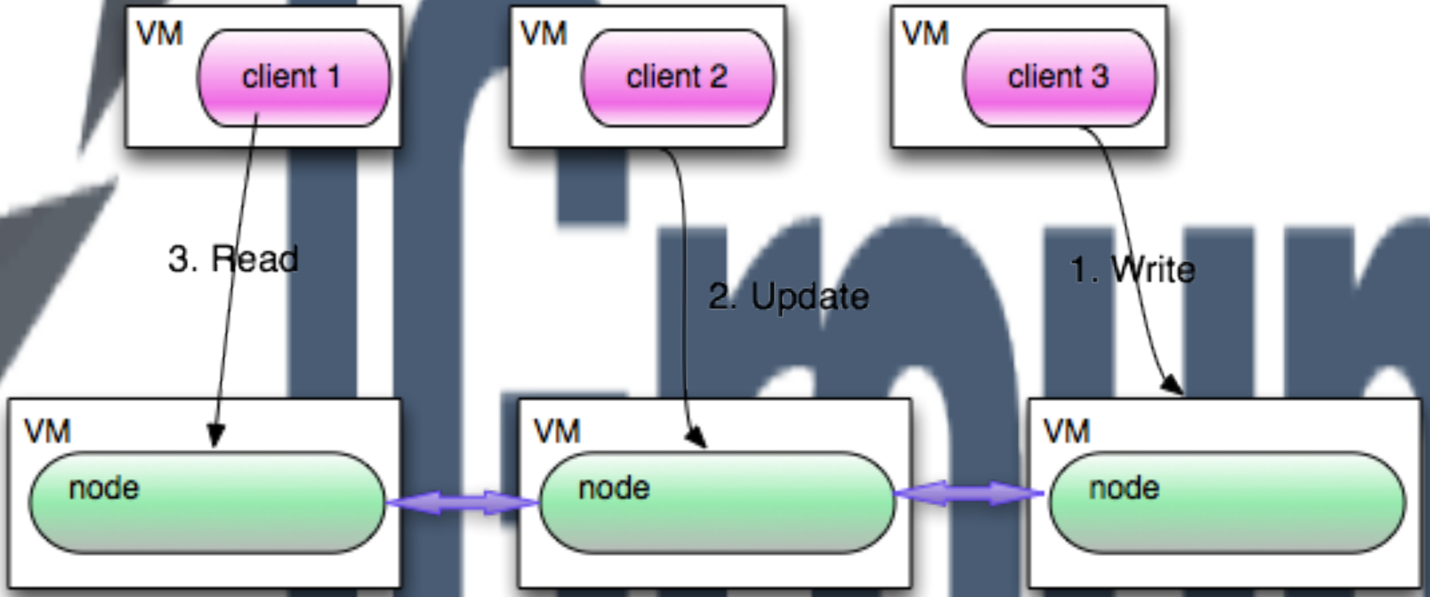
Use Case 1: Local Cache



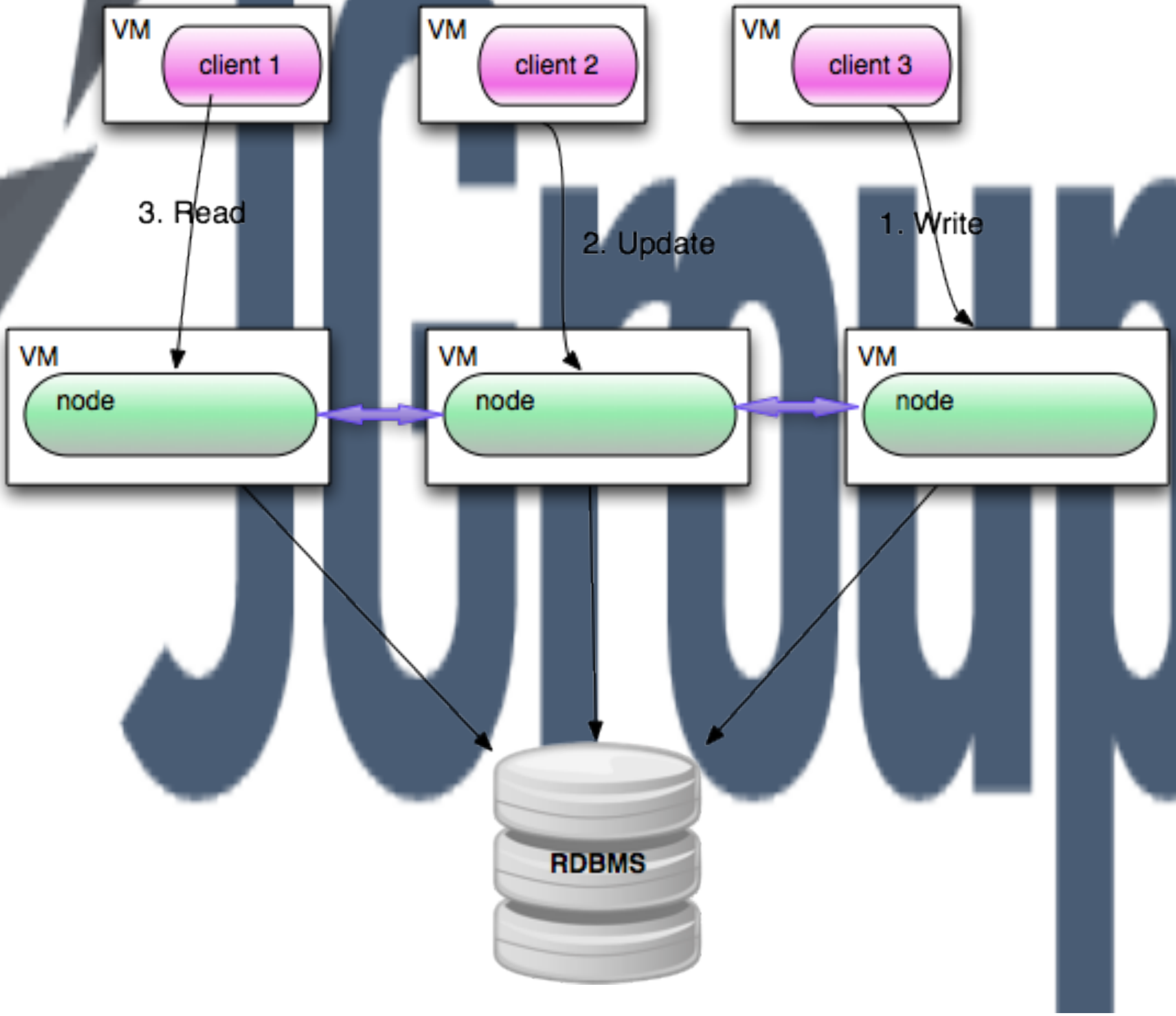
Use Case 2: Cluster of caches



Use Case 3: Data grid



Use Case 3: Data grid



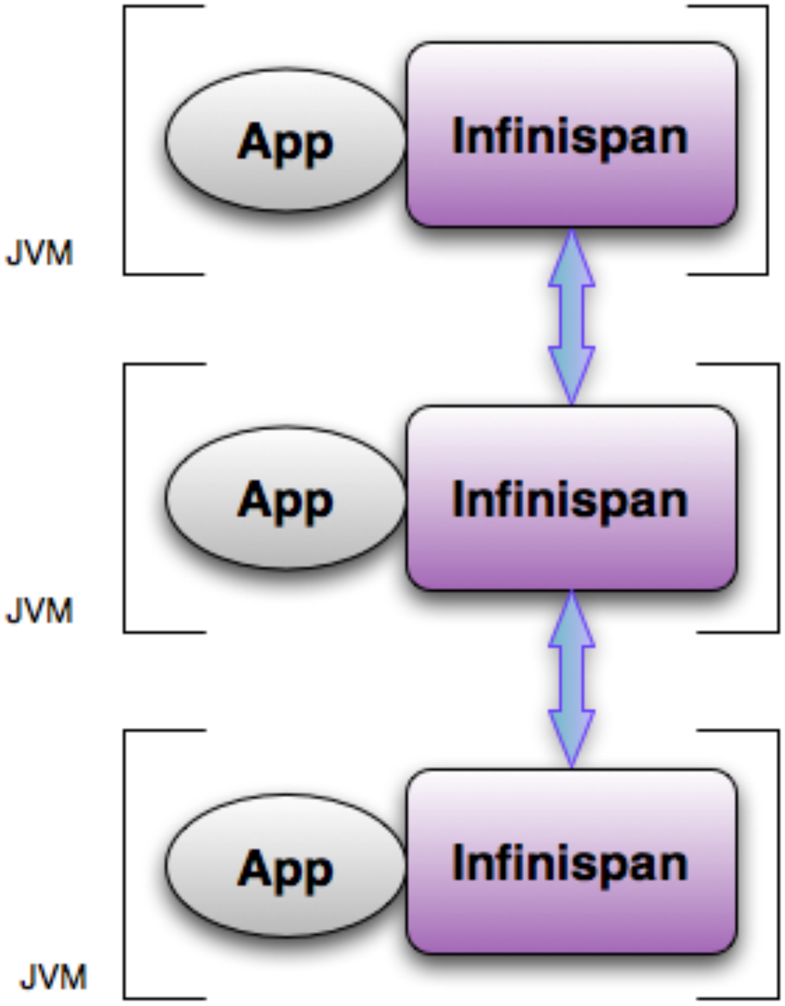
Key features

- Cloud oriented
- Transactions
- Querying
- Map/Reduce and Dist Executors
- Cache loaders
- Management
  - JMX
  - RHQ

Hands on Demo

Reliable Multipoint Communication

Why do you care?



Shall we try it out?

- In the lab project you'll find a test script for your network. Run it!
    - LAB_HOME/nic-test
- If all goes well, you'll get two windows in which you can draw up on your screen. Draw on one, see it in both.
- Easy to try: JGroups has no dependencies!

What is unreliable ?

- Messages get
  - dropped
    - too big (UDP has a size limit), no fragmentation
    - buffer overflow at the receiver, switch
      - NIC, IP network buffer
  - reordered
- We don't know who is in a cluster (IP multicast)
  - we don't know when a new node joins, leaves, or crashes
- Fast sender might overwhelm slower receiver(s)
  - flow control

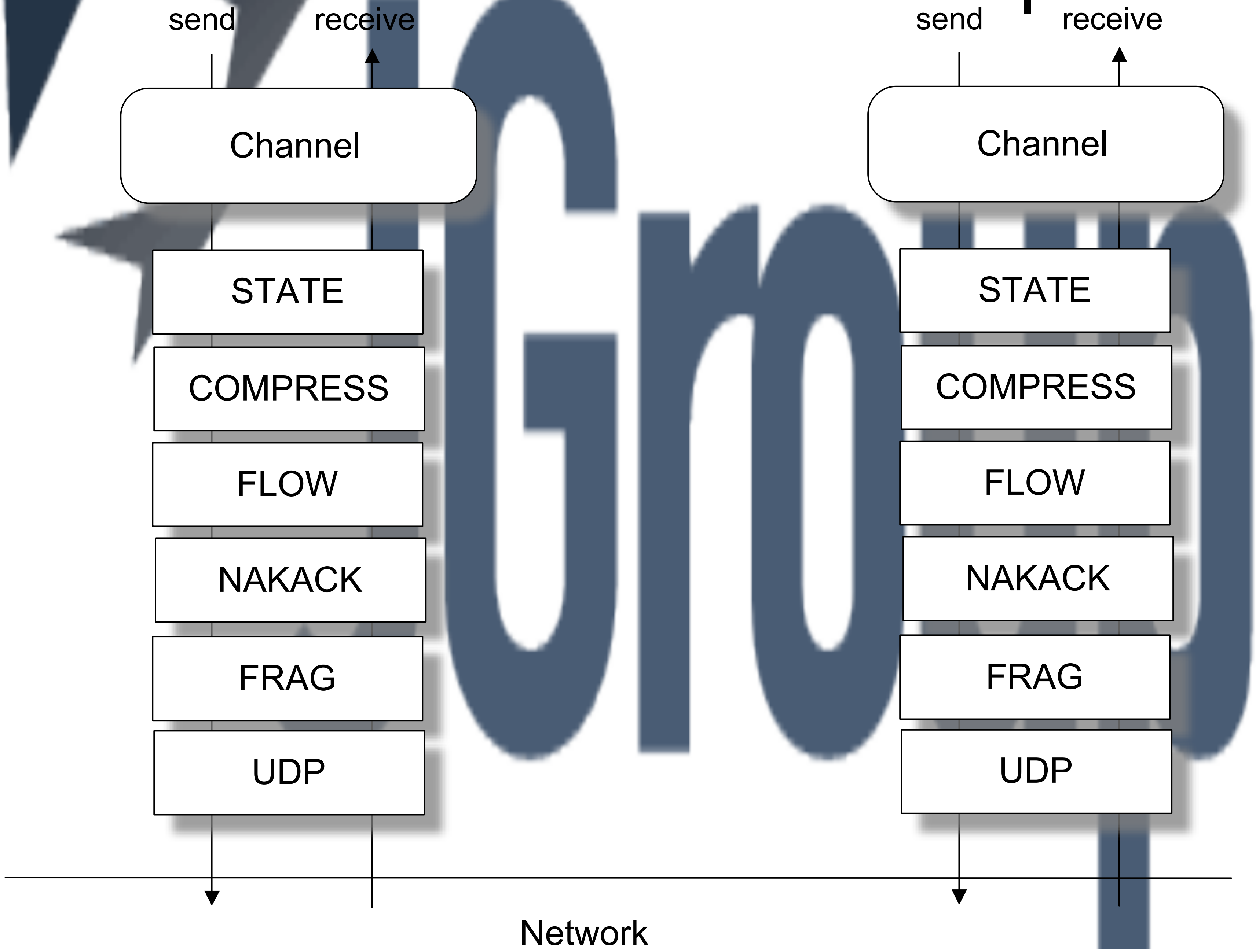
So what Is JGroups ?

- Library for reliable cluster communication
- Provides
  - Fragmentation
  - Message retransmission
  - Flow control
  - Ordering
  - Group membership, membership change notification
- LAN or WAN based
  - IP multicasting transport default for LAN
  - TCP transport default for WAN
  - Autodiscovery of cluster members

Overview

	reliable	unreliable
unicast	TCP / JGroups java.net.Socket java.net.ServerSocket org.jgroups.Channel)	UDP java.net.DatagramSocket
multicast	JGroups org.jgroups.Channel	IP Multicast java.net.MulticastSocket

Architecture of JGroups



Terminology

- Message
- Address
- View
- State transfer
- Group topology

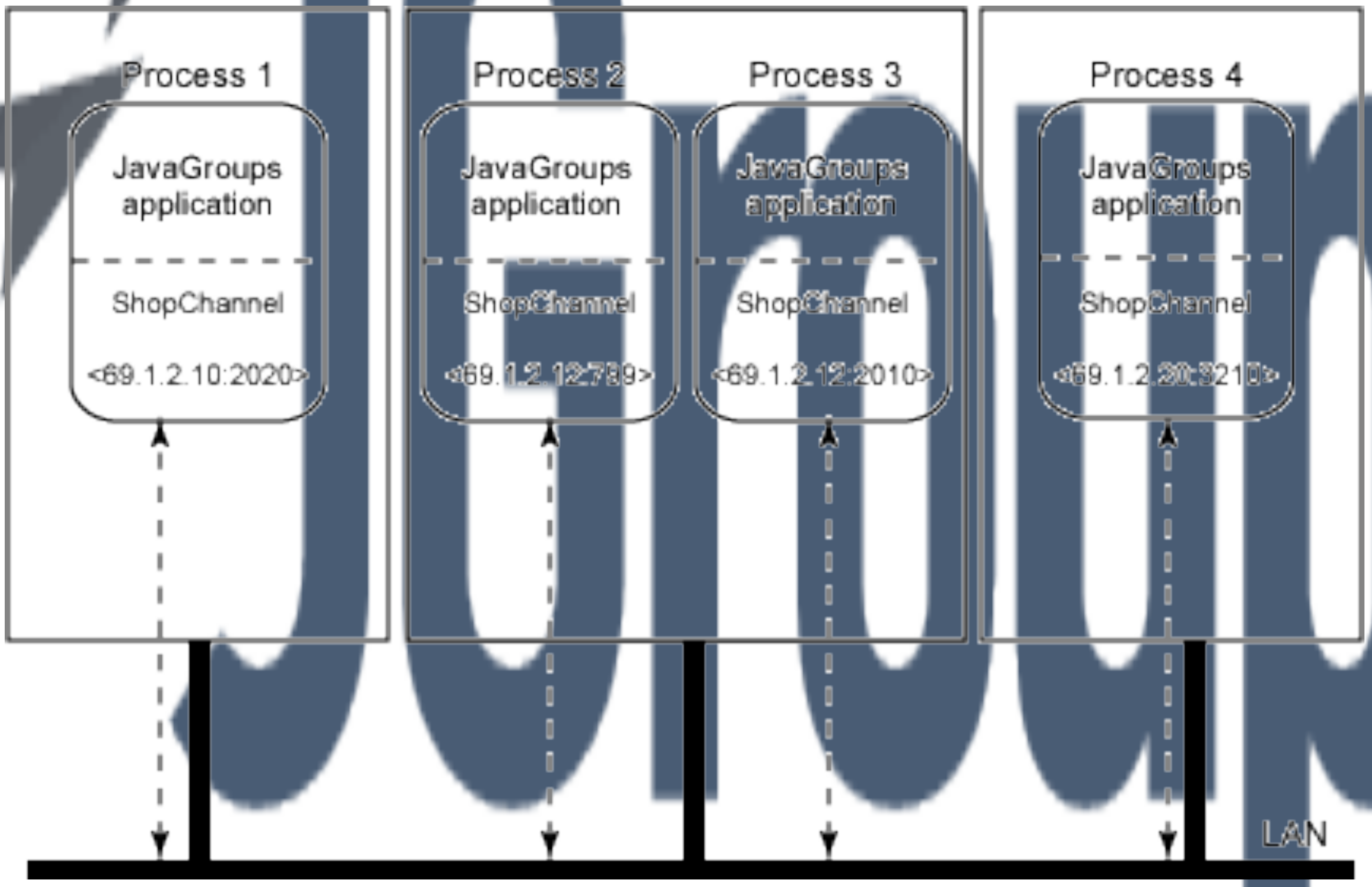
Address

- A cluster consists of a number of members
- Each member has an Address
- The address uniquely identifies the member
- Address is an abstract class
  - Implemented as a UUID
  - A UUID maps to a physical address
- An address can have a logical name
  - E.g. “A”
  - If not set, JGroups picks the name, e.g. “myhost-16524”

View

- List of members (Addresses)
- Is the same in all members:
  - A: {A,B,C}
  - B: {A,B,C}
  - C: {A,B,C}
  - (Same elements, same order)
- Updated when members join or leave

Group topology



Available protocols

- Transport
  - UDP (IP multicasting), TCP, TCP_NIO, Message batching
- Merging, failure detection (hangs, crashes)
- Reliable transmission and ordering
  - Using sequence numbers, dropped messages are retransmitted
- Distributed garbage collection
  - Consensus on received messages, older ones are purged

Available protocols

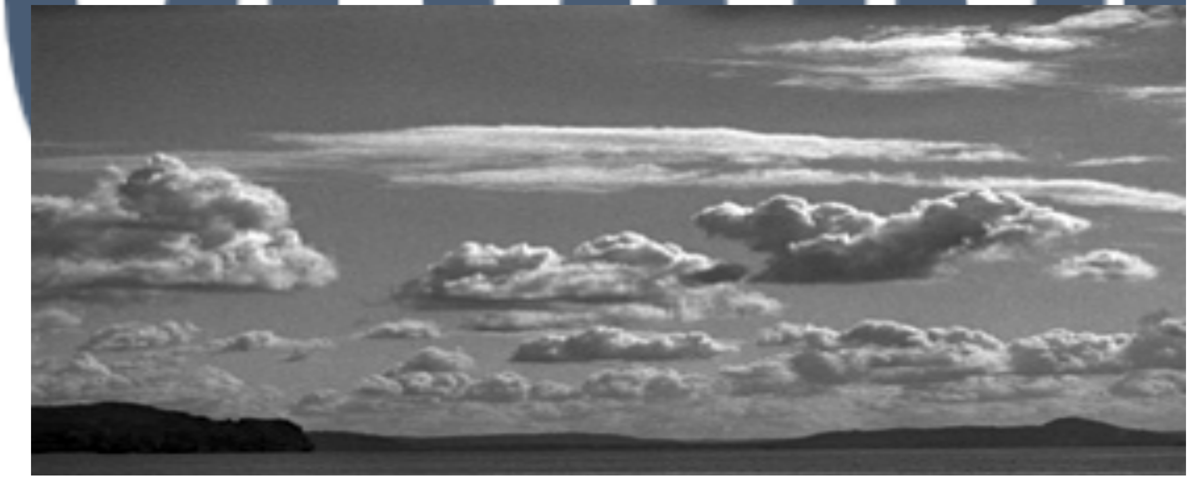
- Group membership
  - Installs new views across a cluster when members join, leave or crash
- Flow control
  - Fast sender is throttled down to the pace of the slowest receiver
- Fragmentation
  - Large packets are fragmented into smaller ones and unfragmented at the receiver side
- Compression, encryption, authentication

Available protocols

- State transfer
  - State transferred to a joining member without stopping the cluster
- Virtual Synchrony
  - All messages sent in view V1 are delivered in V1
  - Flushes unstable messages before a new view is installed
    - Makes sure all members have received all messages sent in V1 before installing V2
- Ordering: total, causal, FIFO

Discovery Protocols

- PING, MPING, BPING, ..
- TCP_PING
- JDBC_PING
- S3_PING
- CASSANDRA_PING

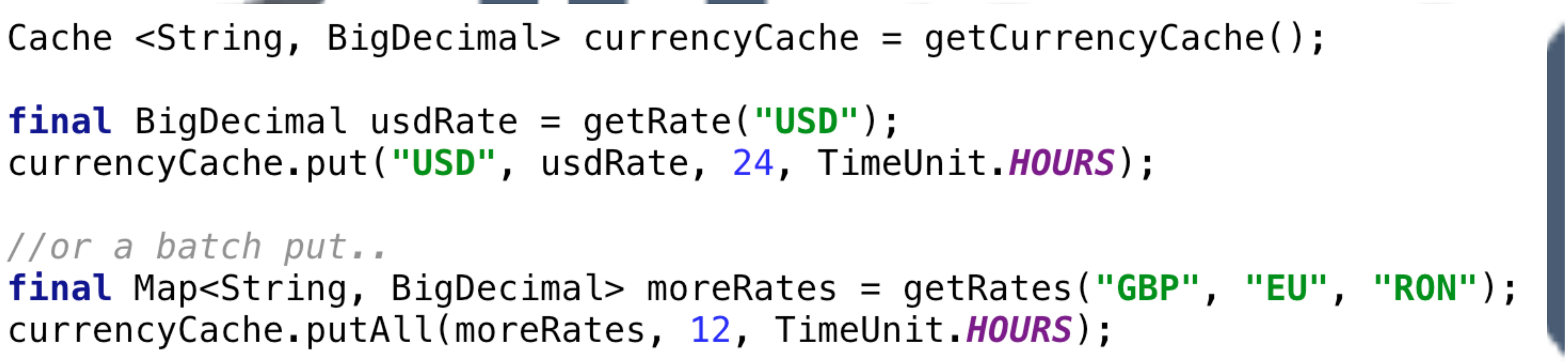


Eviction and expiration

Expiration

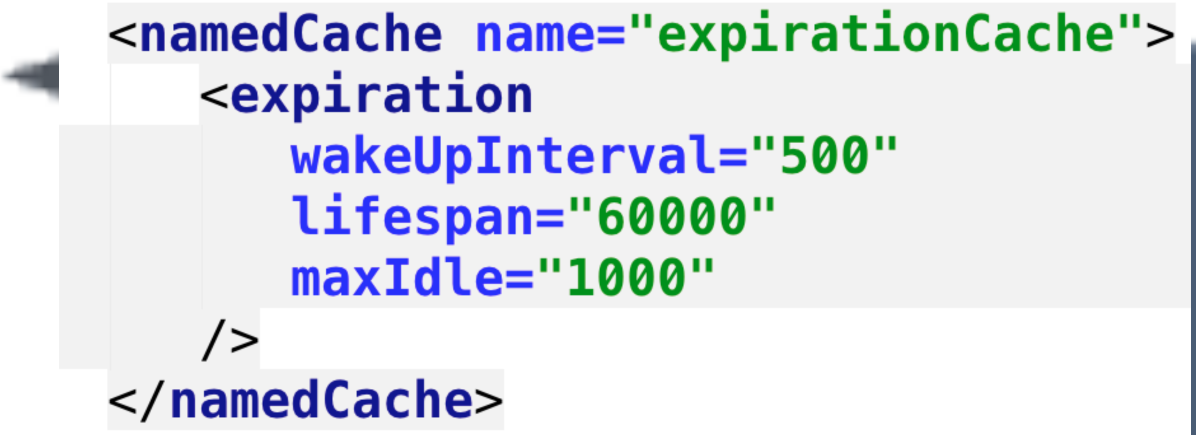
- Time based
  - lifespan
  - max idle
- Expired entries removed
  - from cache
  - from persistent store (if any)

API



```java
Cache <String, BigDecimal> currencyCache = getCurrencyCache();

final BigDecimal usdRate = getRate("USD");
currencyCache.put("USD", usdRate, 24, TimeUnit.HOURS);

//or a batch put..
final Map<String, BigDecimal> moreRates = getRates("GBP", "EU", "RON");
currencyCache.putAll(moreRates, 12, TimeUnit.HOURS);
```

Configuration



```
<namedCache name="expirationCache">
   <expiration
      wakeUpInterval="500"
      lifespan="60000"
      maxIdle="1000"
   />
</namedCache>
```

Eviction

- Memory is finite
  - something has to give!
- Evict based on data access
- Bounded caches

Eviction strategies

- None (default)
- Unordered
- FIFO
- LRU
- LIRS

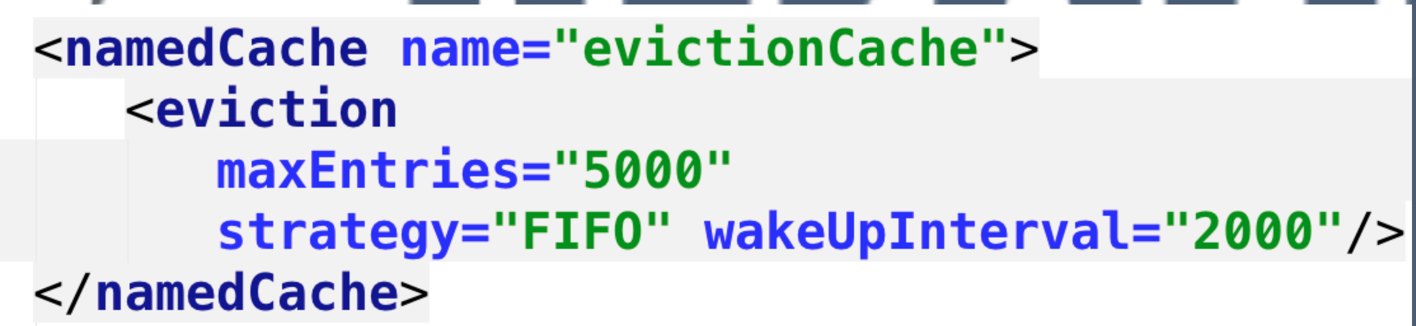
LIRS

- Low Inter-reference Recency Set replacement
- Hybrid
  - frequency of access
  - time of the last access

Passivation

- Evict to external store
    - file, database...
- Cheaper than remote access (?)
- Use the right eviction policy
    - keep relevant bits in memory

Configuration



```
<namedCache name="evictionCache">
   <eviction
      maxEntries="5000"
      strategy="FIFO" wakeUpInterval="2000"/>
</namedCache>
```

Tuning eviction

- What eviction policy should I use?
- Measure, don’t guess
  - Cache JMX stats
  - hits/misses ratio
- Memory issues?
  - Aggressive wakeup interval

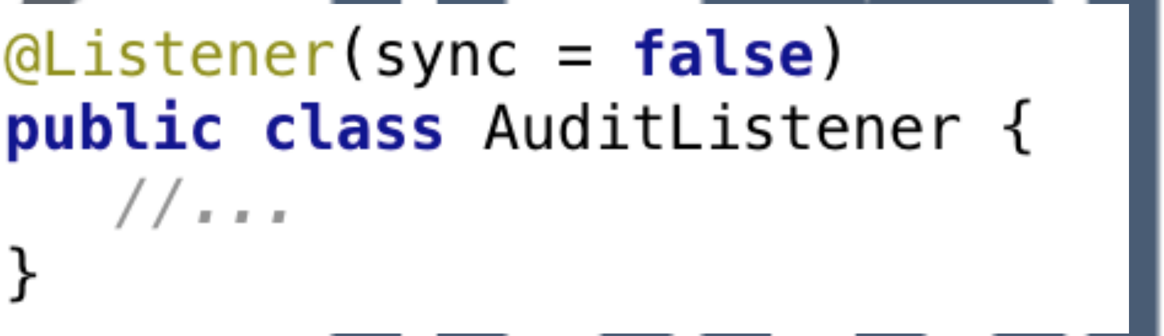
Listeners

Listener types

- Cache listeners
  - data: added, remove, changed, entry loaded
  - transaction: completed, registered
  - topology: changed, data rehashed
- Cache manager listeners
  - cache started/stopped, view changed/merge

Synchronicity

- listener executes in caller’s thread (default)
  - keep it short!
- Or as



```
@Listener(sync = false)
public class AuditListener {
   //...
}
```

- Listeners are local
- Can veto an operation
- Participate in transactions
- Do not work on RemoteCacheManager

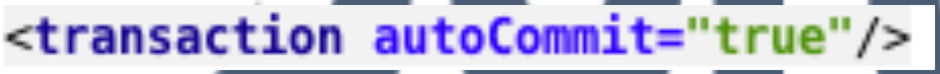
Transactions

Agenda

- Transactions
  - optimistic/pessimistic
  - JTA support
- XA (or not)
- Recovery
- Deadlock avoidance

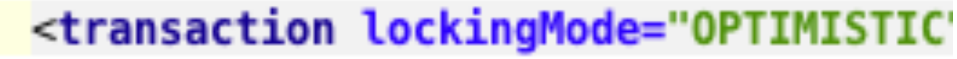
Cache types

- Non transactional
- Transactional
  - optimistic
  - pessimistic
  - TransactionManager required
- No mixed-access



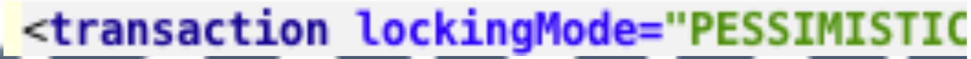
```
<transaction autoCommit="true"/>
```

Transactional caches



```
<transaction lockingMode="OPTIMISTIC
```

- Optimistic
  - no locks before prepare
  - small lock scope

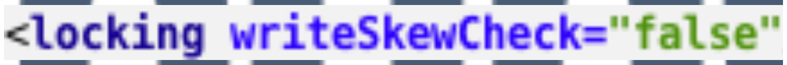


```
<transaction lockingMode="PESSIMISTIC
```

- Pessimistic
  - lock acquired on each write
  - writes block writes
  - reads do not block
- locks held longer

Pessimistic or Optimistic?

- Optimistic
  - low contention
  - high contention -> many rollbacks
  - disable version check



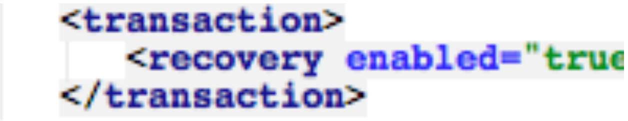
```
<locking writeSkewCheck="false"
```

- Pessimistic
  - high key contention
  - rollbacks are less desirable
- more costly/more guarantees

JTA integration

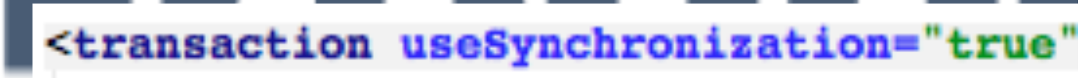
- JTA transactions
    - known API
- Multiple options
    - full xa (XAResource)
    - less strict (Synchronization)

XA or not?



```
<transaction>
   <recovery enabled="true
</transaction>
```

- XA
  - proper distributed transactions
  - recovery enabled
    - or not



```
<transaction useSynchronization="true"
```

- Synchronization
  - cache backed by a data store
  - Transaction more efficient
- 1PC optimisation
- TransactionManager not writing logs
- Hibernate 2LC

Recovery

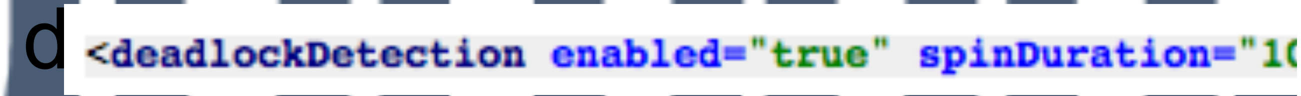
- When is needed?
  - prepare successful, commit fails
  - inconsistent state!
- How to handle it
  - TransactionManager informs SysAdmin
  - JMX tooling exposed to
  - force commit
  - force rollback

Deadlocks

- Deadlock
  - Tx1: a -> b
  - Tx2: b -> a
  - “right” timing
- Bad for system throughput
  - threads blocked until (one) tx timeouts
  - lockAcquisitionTimeout defaults to 10 seconds!
  - a,b are locked during this time -> potentially more deadlocks

What’s to be done?

- Order key
  - e.g. lexicographically
  - Tx1: a -> b
  - Tx2: a -> b
  - not always possible
- Use deadlock d



```
<deadlockDetection enabled="true" spinDuration="1
```

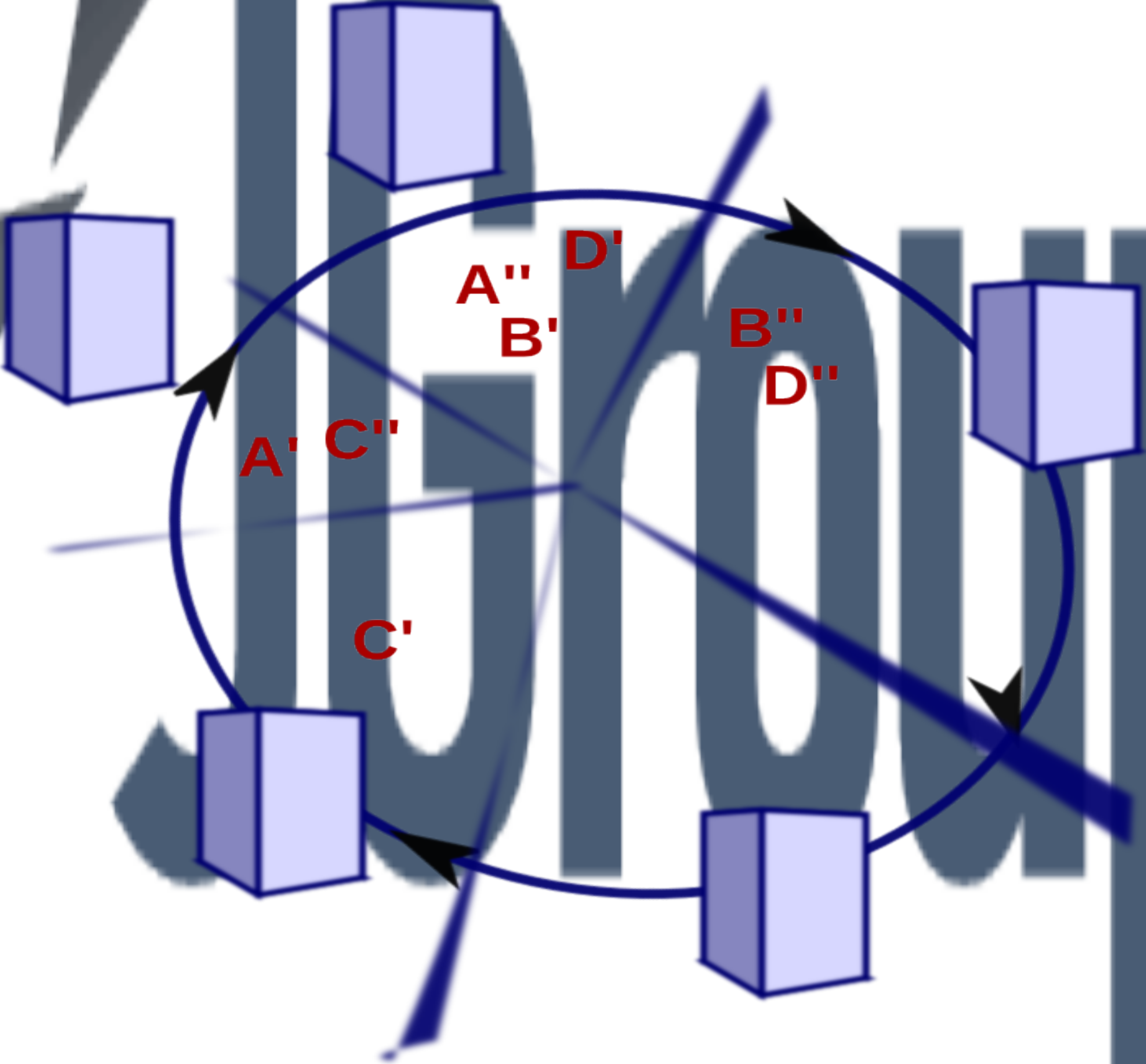
  - fail fast
  - one tx succeeds

New deadlock avoidance techniques (5.1)

- Single lock owner
  - avoid same key-deadlocks
- Optimistic only
  - Incremental locking
    - acquire locks on the same node sequence
- Lock reordering
  - based on consistent hash

Modes of Operation

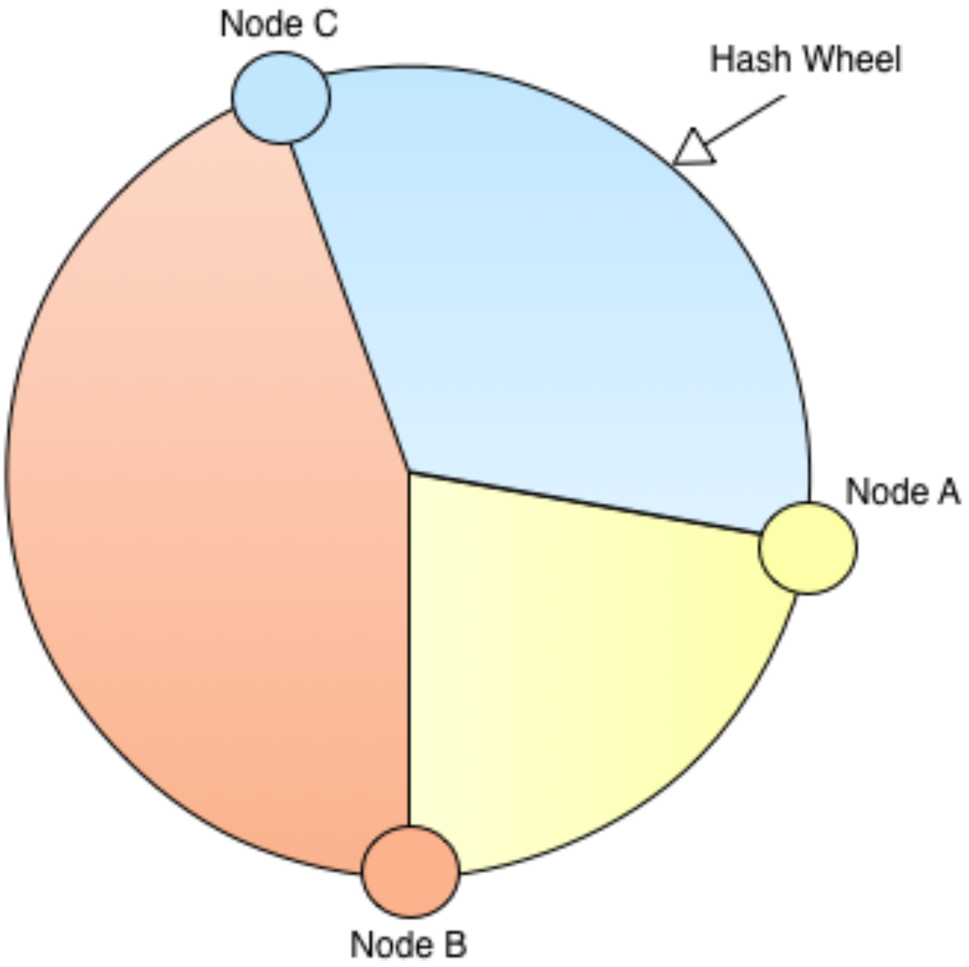
Consistent Hashing: DIST



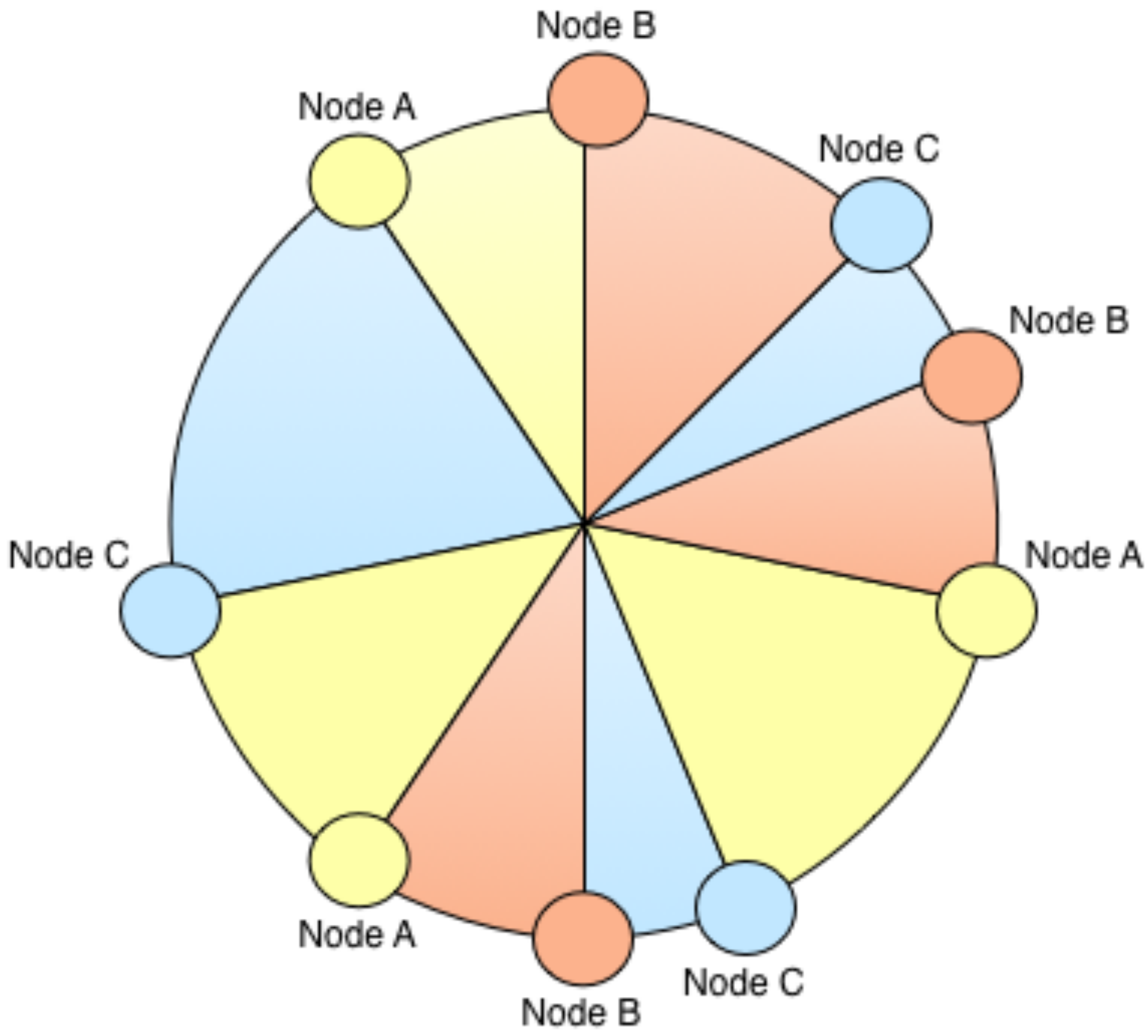
Clustering: Cache modes

- DIST
  - Sync/Async
- REPL
  - Sync/Async
- LOCAL
  - Doesn't have async
- INV
  - Sync/Async

DIST again

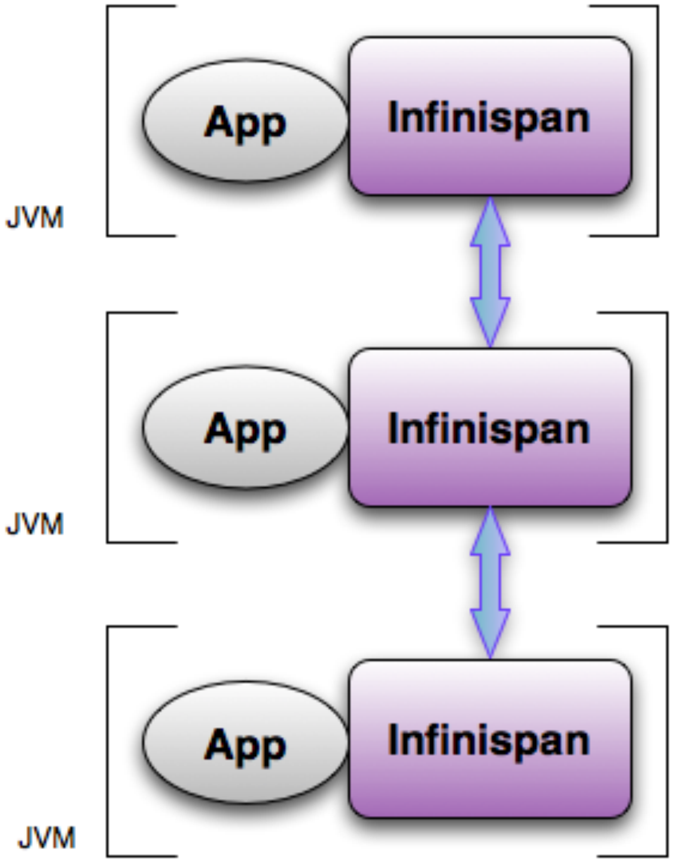


DIST + VNodes



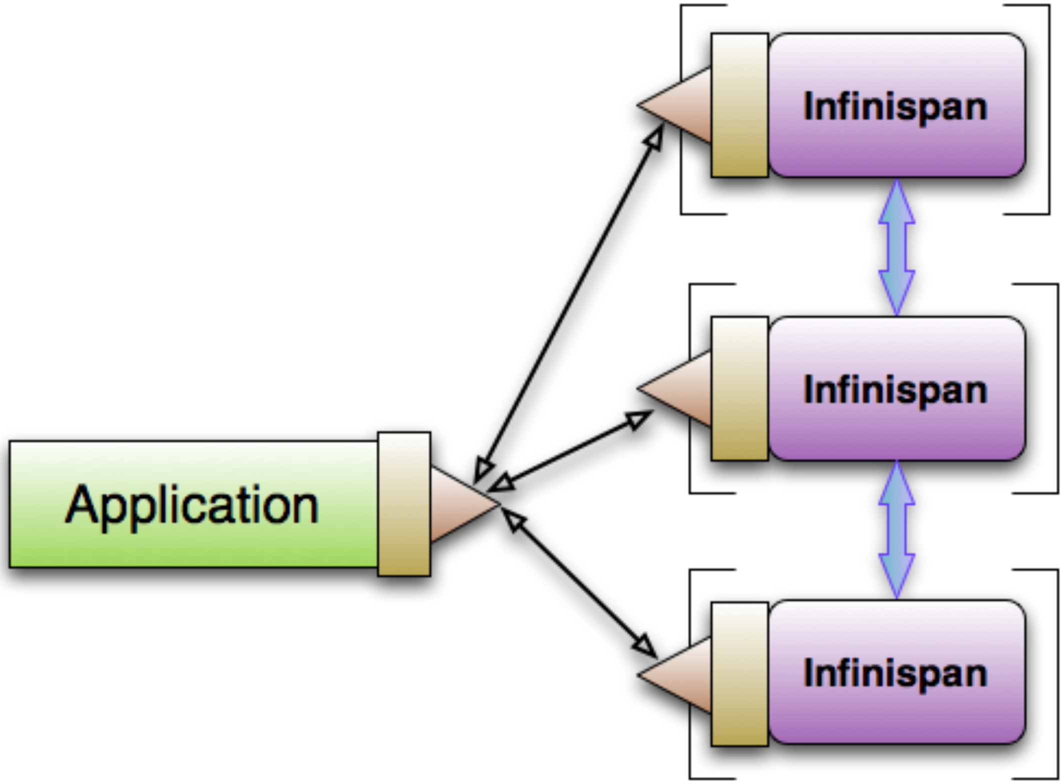
Client Server

Peer to peer



Client/Server Architecture

Supported Protocols
REST
Memcached
Hot Rod



Hotrod?!

- Wire protocol for client server communications
- Open
- Language independent
- Built-in failover and load balancing
- Smart routing
- xa support - to come

Server Endpoint Comparison

	Protocol	*Client Libraries*	*Clustered?*	*Smart Routing*	*Load Balancing/Failover*
REST	***Text***	N/A	*Yes*	***No***	Any HTTP load balancer
Memcached	***Text***	Plenty	*Yes*	***No***	Only with predefined server list
Hot Rod	***Binary***	Java, Python	*Yes*	*Yes*	Dynamic

Client/Server - when?

- Client not affected by server topology changes
- Multiple apps share the same grid
- Tier management
  - incompatible JVM tuning
  - security
- Non-JVM clients

Cache Stores

Why use cache stores?

- Durability
- More caching capacity
- Warm caches
  - preload

Features

- Chaining
  - more than one per cache
- Passivation
  - with eviction
- Async
  - write behind
- Shared

Types of cache stores

- File system
  - FileCacheStore
  - BdbjeCacheStore
- JDBC
- Cloud cache store (JCouds)

More cache stores

- RemoteCacheStore
  - use Hotrod
- Cassandra
- ClusterCacheStore
  - alternative to state transfer
- Custom!

Extras

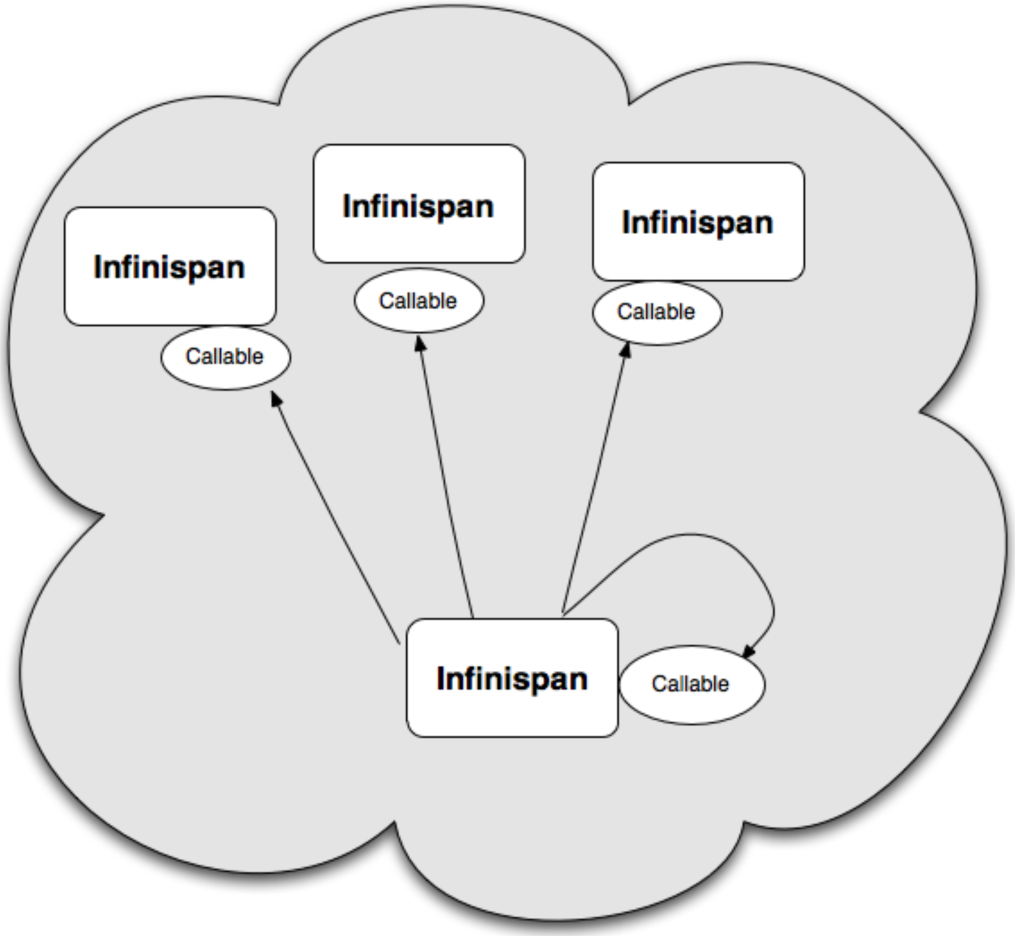
Map Reduce
& Distributed Executors

Distributed Executors

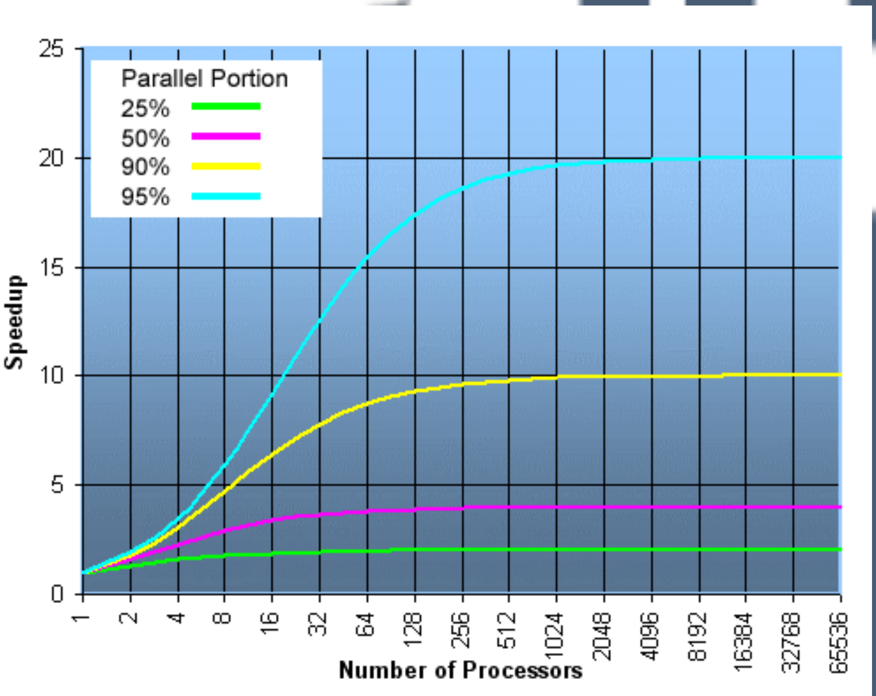
```
public interface DistributedExecutorService extends ExecutorServ

   <T, K> Future<T> submit(Callable<T> task, K... input);

   <T> List<Future<T>> submitEverywhere(Callable<T> task);

   <T, K > List<Future<T>> submitEverywhere(Callable<T> task, K... input);
}
```

```
public interface DistributedCallable<K, V, T> extends Callable<T

   void setEnvironment(Cache<K, V> cache, Set<K> inputKeys);
}
```

However,behind the scenes..



Do not forget Gene Amdahl



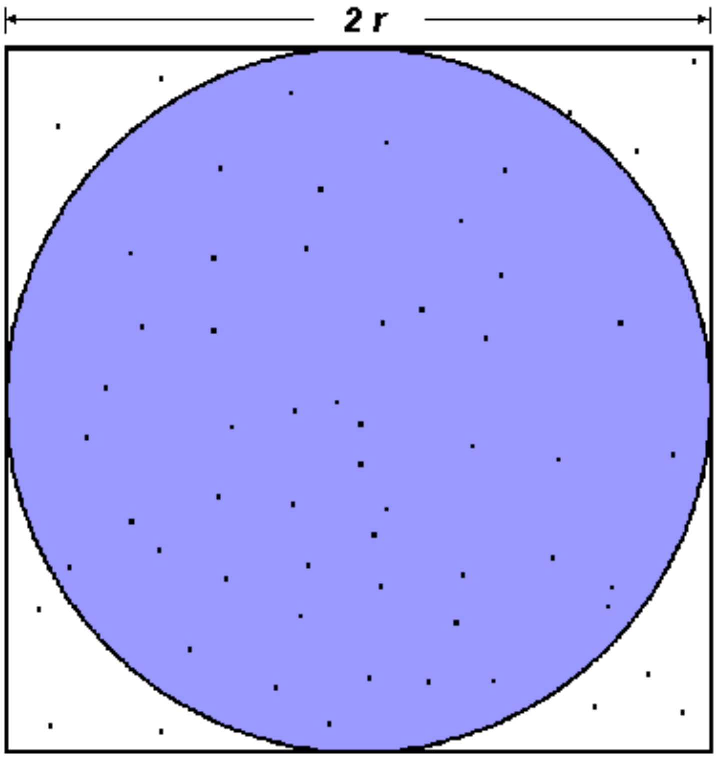
Speedup = 1/(p/n)+(1-p)

However, problems that increase the percenta of parallel time with their size are more scalab than problems with fixed percentage of paralle time

p = parallel fraction
n = number of proc

Source:
https://computing.llnl.gov/tutorials/parallel_comp/

π approximation



$$A_S = (2r)^2 = 4r^2$$

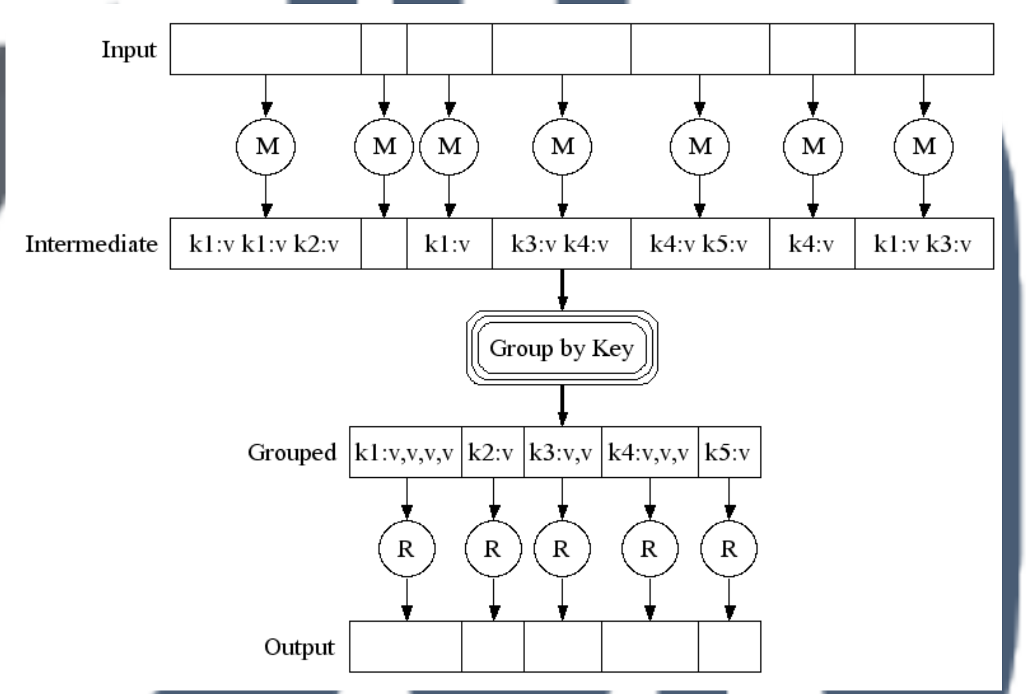
$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

Infinispan MapReduce

- We already have a data grid!
- Leverages Infinispan’s DIST mode
- Cache data is input for MapReduce tasks
- Task components: Mapper, Reducer, Collator
- MapReduceTask cohering them together

MapReduce model



Source:
http://labs.google.com/papers/mapreduce.html

Mapper, Reducer, Collator

```
public interface Mapper<KIn, VIn, KOut, VOut> extends Serializable

    void map(KIn key, VIn value, Collector<KOut, VOut> collector);
}


public interface Reducer<KOut, VOut> extends Serializable {

    VOut reduce(KOut reducedKey, Iterator<VOut> iter);
}

public interface Collator<KOut, VOut, R> {

    R collate(Map<KOut, VOut> reducedResults);
}
```

Querying

To query a Grid

- What's in C7 ?

```
Object p =
    cache.get(“c7”);
```

- Where is the white King?

Infinispan and Queries

- How to query the grid
  - Key access
  - Statistics
  - Map/Reduce
  - Indexing of stored objects
- Integrate with existing search engines
  - Scale
  - Highly available

Indexing of stored objects

- Maven module: infinispan-query
- Configuration: indexing=true
  - Will trigger on annotated objects
- Integrates hibernate-search-engine
- Based on Apache Lucene

Enable indexing

```
Configuration c = new Configuration()
      .fluent()
      .indexing()
      .addProperty(
  "hibernate.search.option", "value")
      .build();
CacheManager manager = new DefaultCacheManager(c);
```

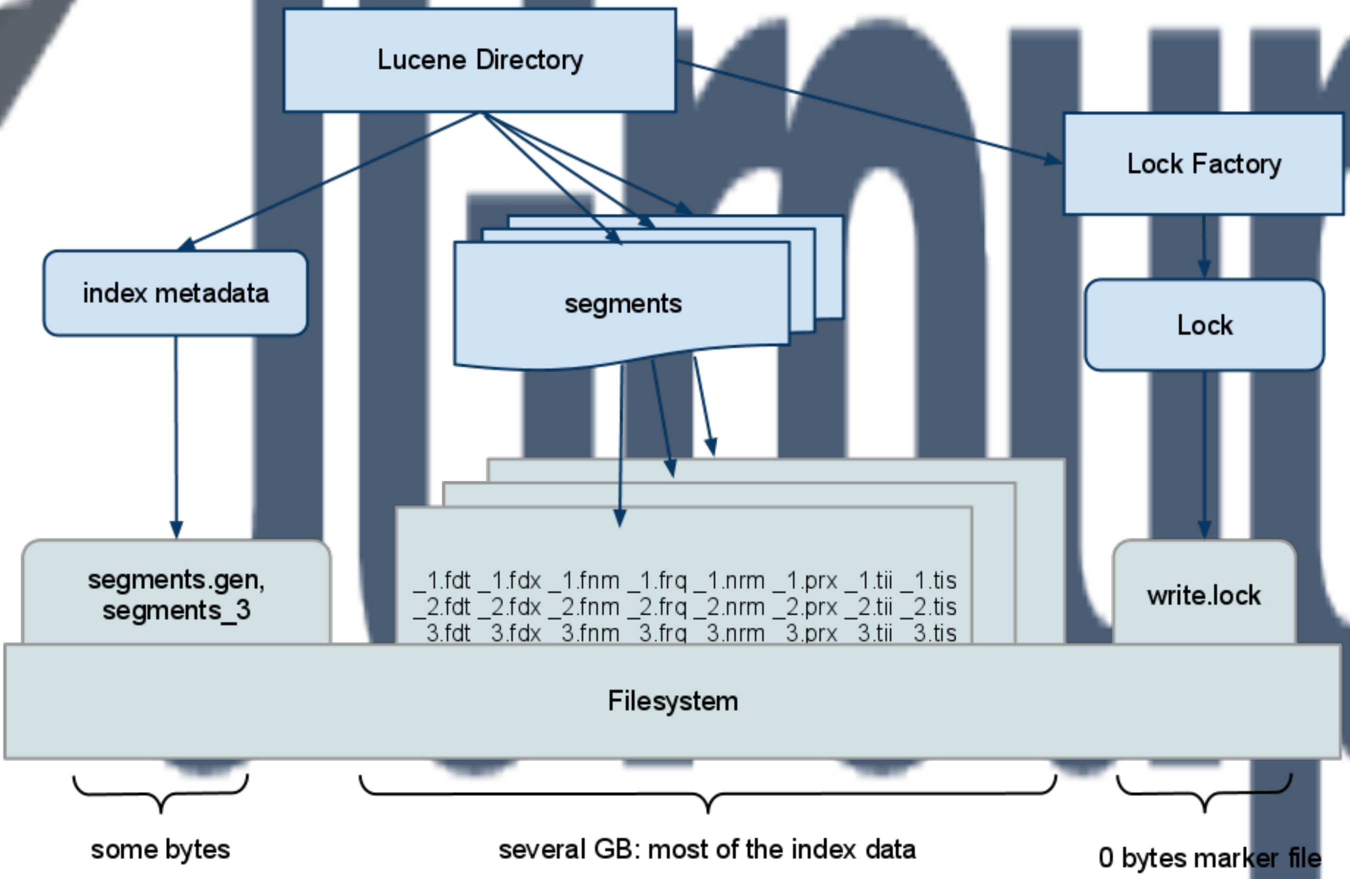
Annotate your objects

- ***@ProvidedId @Indexed***
- public class ***Book*** implements Serializable {
- ***@Field*** String ***title***;
- ***@Field*** String ***author***;
- ***@Field*** String ***editor***;
- ...
- }

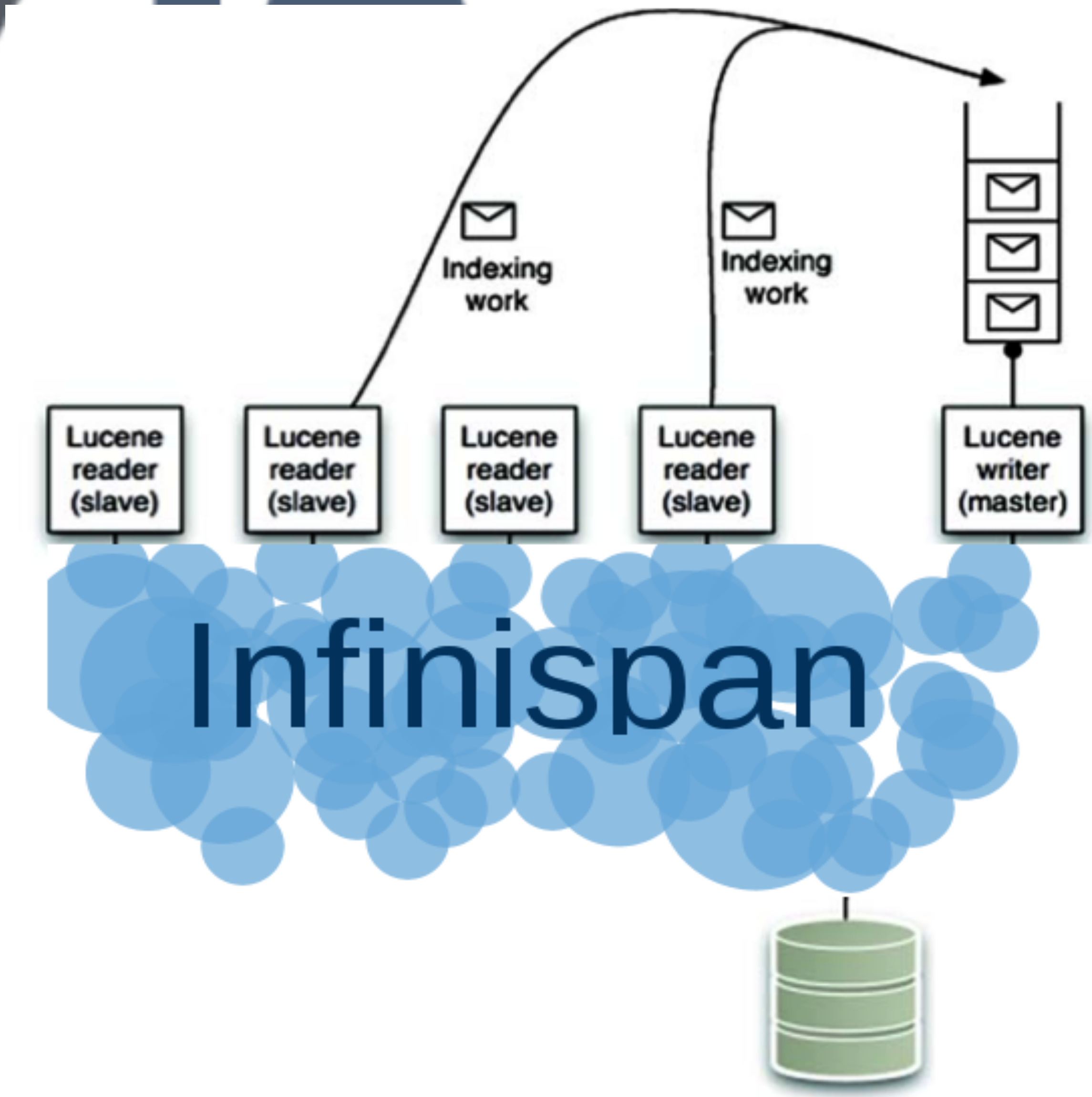
Search them!

```
SearchManager sm = Search.getSearchManager(cache);

Query query = sm.buildQueryBuilderForClass(Book.class)
    .get()
        .phrase()
            .onField("title")
            .sentence("in action")
    .createQuery();

List<Object> list = sm.getQuery(query).list();
```

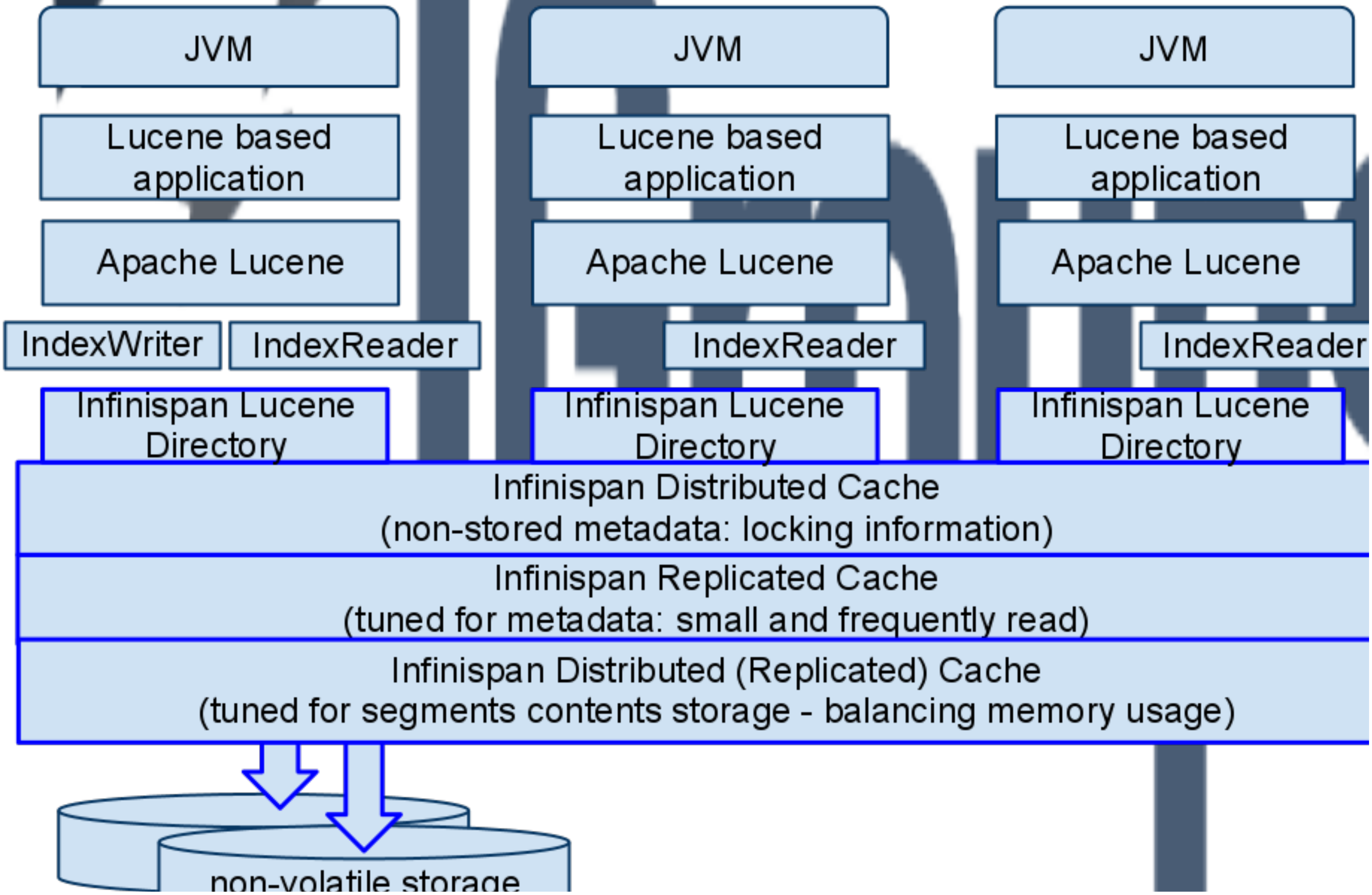
Lucene API, storing in Infinispan


Lucene Directory
Lock Factory
index metadata
segments
Lock
segments.gen,
segments_3
_1.fdt _1.fdx _1.fnm _1.frq _1.nrm _1.prx _1.tii _1.tis
_2.fdt _2.fdx _2.fnm _2.frq _2.nrm _2.prx _2.tii _2.tis
3.fdt 3.fdx 3.fnm 3.frq 3.nrm 3.prx 3.tii 3.tis
write.lock
Filesystem
some bytes
several GB: most of the index data
0 bytes marker file

Limited write concurrency



Example of multi-cache app



JVM

Lucene based application

Apache Lucene

IndexWriter

IndexReader

Infinispan Lucene Directory

JVM

Lucene based application

Apache Lucene

IndexReader

Infinispan Lucene Directory

JVM

Lucene based application

Apache Lucene

IndexReader

Infinispan Lucene Directory

Infinispan Distributed Cache
(non-stored metadata: locking information)

Infinispan Replicated Cache
(tuned for metadata: small and frequently read)

Infinispan Distributed (Replicated) Cache
(tuned for segments contents storage - balancing memory usage)

non-volatile storage

HIBERNATE OG
JGroups

HIBERNATE OGM

- OGM: Object/Grid Mapper
- Implements JPA for NoSQL engines
  - Infinispan as first supported “engine”
  - More coming
- Simplified migration across different NoSQL, SQL databases
  - With transactions, or whatever is possible.
  - Fast? Contribute tests and use cases!

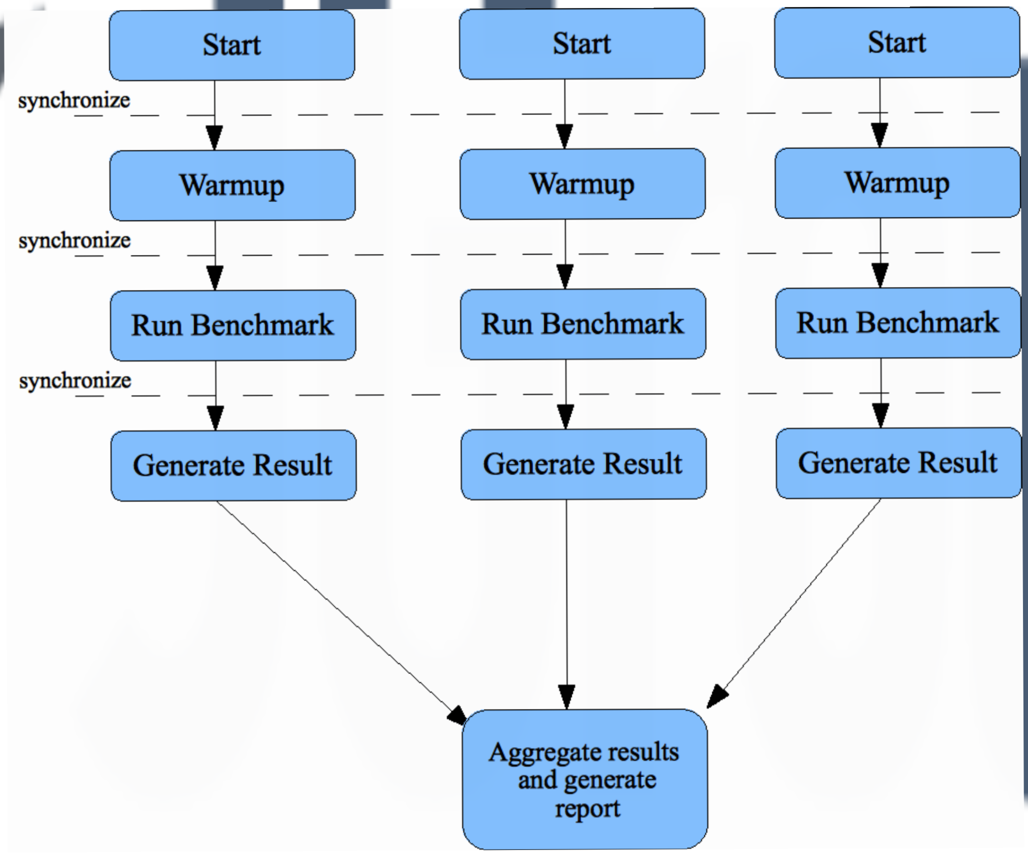
- JPA on NoSQL: an approach with Hibernate OGM
  - Devoxx 2011
    - November 17th (conf Day 4) - 14:00 – 15:00
    - Emmanuel Bernard

Radargun

What is Radargun?

- Benchmarking tool
  - in memory data grids
- Pluggable
  - products
  - data access patterns

Basic Idea


Start
Start
Start
synchronize
Warmup
Warmup
Warmup
synchronize
Run Benchmark
Run Benchmark
Run Benchmark
synchronize
Generate Result
Generate Result
Generate Result
Aggregate results and generate report

Status

- 1.0 Released
  - Web session replication
  - Transaction benchmarks
  - run on 100+ nodes
- 1.1 on the way
  - TPC-C plugin for tx benchmarking
  - consistent hash efficiency

Conclusion

Use Cases

- Local Cache
- Distributed Cache
- Data Grid

Access Modes

- Embedded
- Remote
    - Hot Rod
    - REST
    - Memcache

Control

- Eviction
- Expiration
- Management

Transaction & Locking

- XA
- Local

Persistence

- Cache Stores

Q&A