Introduction to Infinispan

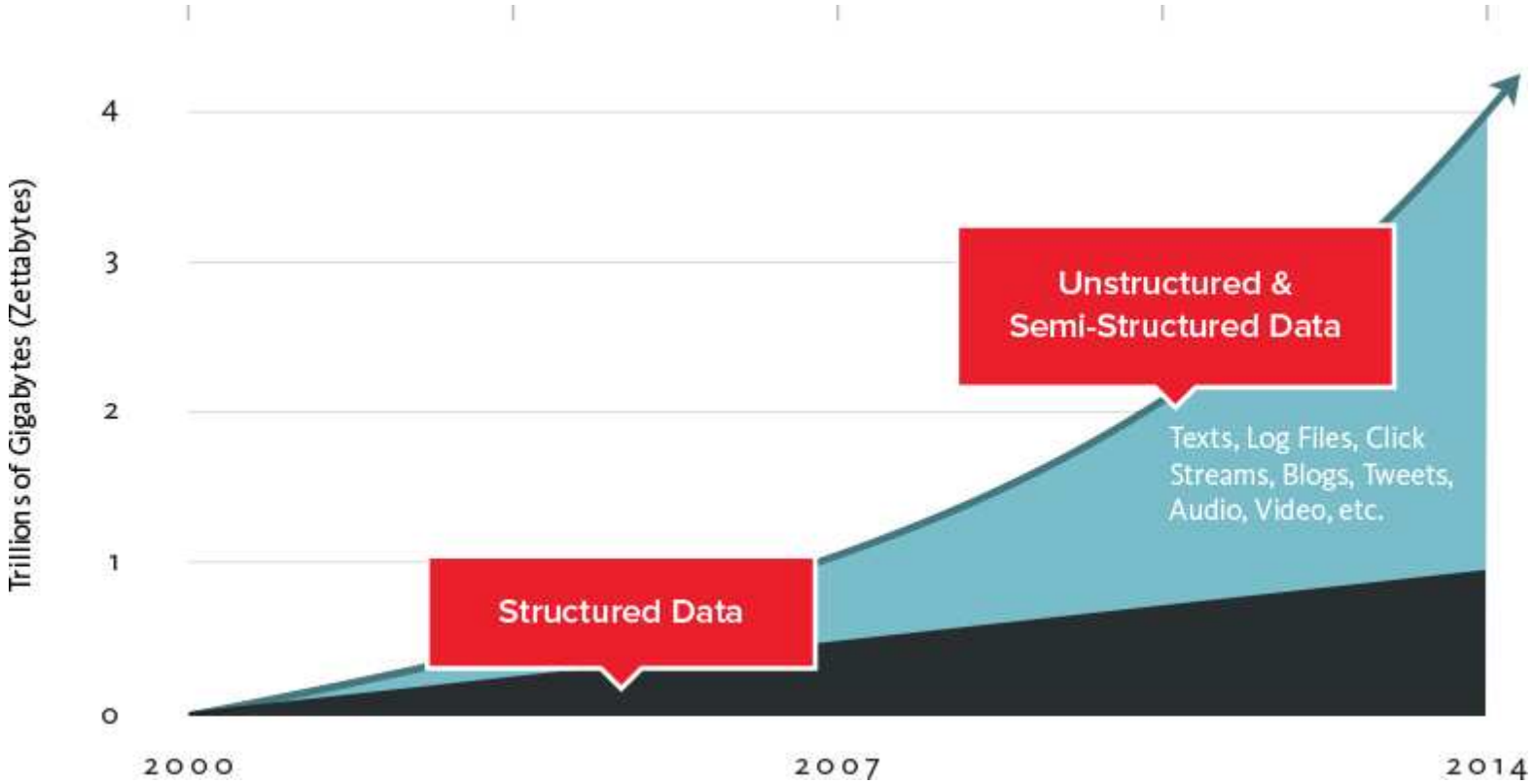
Vojtěch Juránek

JBoss - a division by Red Hat

18. 3. 2016, CTU FEL, Prague

Data today

Data today



Source: `http://www.couchbase.com/nosql-resources/what-is-no-sql`

How big are Big data?

How big are Big data?

Source: `https://twitter.com/DEVOPS_BORAT/status/288698056470315008`

How big are Big data?

Source: `https://twitter.com/DEVOPS_BORAT/status/288698056470315008`

- Data collection so large and complex it's impossible to process it on one computer
- You can scale up, but sooner or later you'll have to scale out

Big data characteristics

- **Volume:** unprecedented amount of data being stored
- **Velocity:** speed at which the data is generated
- **Variety:** the type and nature of the data - from structured data in traditional databases to unstructured text documents, email, video, audio etc.
- **Variability:** the amount of incoming data can highly vary
- **Veracity:** the quality of captured data can vary greatly as well

Big data characteristics

- **Volume:** unprecedented amount of data being stored
- **Velocity:** speed at which the data is generated
- **Variety:** the type and nature of the data - from structured data in traditional databases to unstructured text documents, email, video, audio etc.
- **Variability:** the amount of incoming data can highly vary
- **Veracity:** the quality of captured data can vary greatly as well

Big data characteristics

- **Volume:** unprecedented amount of data being stored
- **Velocity:** speed at which the data is generated
- **Variety:** the type and nature of the data - from structured data in traditional databases to unstructured text documents, email, video, audio etc.
- **Variability:** the amount of incoming data can highly vary
- **Veracity:** the quality of captured data can vary greatly as well

Big data characteristics

- **Volume:** unprecedented amount of data being stored
- **Velocity:** speed at which the data is generated
- **Variety:** the type and nature of the data - from structured data in traditional databases to unstructured text documents, email, video, audio etc.
- **Variability:** the amount of incoming data can highly vary
- **Veracity:** the quality of captured data can vary greatly as well

Big data characteristics

- **Volume:** unprecedented amount of data being stored
- **Velocity:** speed at which the data is generated
- **Variety:** the type and nature of the data - from structured data in traditional databases to unstructured text documents, email, video, audio etc.
- **Variability:** the amount of incoming data can highly vary
- **Veracity:** the quality of captured data can vary greatly as well

Big data characteristics

- **Volume:** unprecedented amount of data being stored
- **Velocity:** speed at which the data is generated
- **Variety:** the type and nature of the data - from structured data in traditional databases to unstructured text documents, email, video, audio etc.
- **Variability:** the amount of incoming data can highly vary
- **Veracity:** the quality of captured data can vary greatly as well

Big data - some of the challenges

- Analysis run on top of the huge amount of data
- Ability to store huge amount of unstructured data (often for performance reasons)
- But also ability to talk to RDBMS or query structured data is often needed as well
- Highly scalable solution (also because of cost effectiveness)
- Cloud architecture - everything is ephemeral
- Information privacy

NoSQL

NoSQL

Source: `https://twitter.com/devops_borat/status/141368065110708224`

NoSQL

- Nature of the data
- More flexible data mode
- Better scalablity
- Performance

Source: www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf

NoSQL

- Nature of the data
- More flexible data mode
- Better scalablity
- Performance

Source: www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf

NoSQL

- Nature of the data
- More flexible data mode
- Better scalablity
- Performance

Source: www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf

NoSQL

- Nature of the data
- More flexible data mode
- Better scalablity
- Performance

Source: `www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf`

What is a data grid?

What is a data grid?

- An in-memory distributed data store designed for fast access to large volumes of data and scalability.
- Commonly a complementary layer to the relational database and the application.

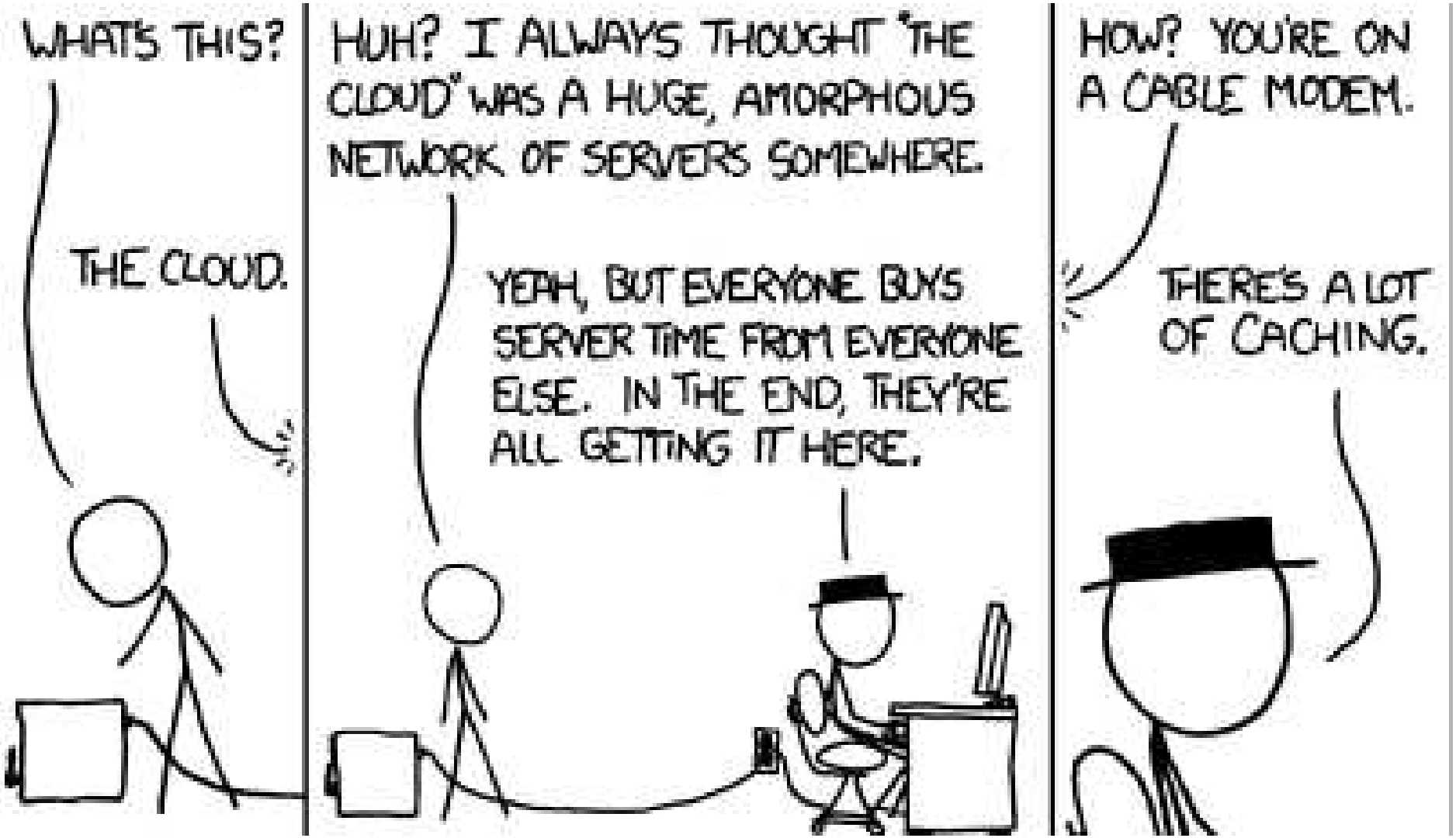
What is a data grid?

- An in-memory distributed data store designed for fast access to large volumes of data and scalability.
- Commonly a complementary layer to the relational database and the application.

Key data grid characteristics:

- In-memory, distributed caching
- Elastic and scalable
- Advanced querying
- Data replication
- Processing for streaming data
- Transaction capabilities

Why in-memory



Source: Part of xkcd #908

Why in-memory

- Lots of data is needed in real-time (BigData $\rightarrow$ FastData)
- Some tasks can be completed much faster when data are kept in memory
- Keeping data in memory during processing of whole application stack, not only during processing in one application in the stack
- With data replication you can keep your data only in memory (no need to store them in persistent storage)

Why in-memory

- Lots of data is needed in real-time (BigData $\rightarrow$ FastData)
- Some tasks can be completed much faster when data are kept in memory
- Keeping data in memory during processing of whole application stack, not only during processing in one application in the stack
- With data replication you can keep your data only in memory (no need to store them in persistent storage)

Why in-memory

- Lots of data is needed in real-time (BigData $\rightarrow$ FastData)
- Some tasks can be completed much faster when data are kept in memory
- Keeping data in memory during processing of whole application stack, not only during processing in one application in the stack
- With data replication you can keep your data only in memory (no need to store them in persistent storage)

Why in-memory

- Lots of data is needed in real-time (BigData $\rightarrow$ FastData)
- Some tasks can be completed much faster when data are kept in memory
- Keeping data in memory during processing of whole application stack, not only during processing in one application in the stack
- With data replication you can keep your data only in memory (no need to store them in persistent storage)

Scalability vs. elasticity

Sometimes used as synonyms, but usually

- **Scalability:** ability of the system to deal reasonable well with increasing load (data volume, traffic volume, complexity etc.), usually just by adding more resources.
- **Elasticity:** ability to fit the resources needed to cope with changed loads dynamically. Sometimes elasticity refers to fit the resources (add/remove resources) in an automated manner, when needed.

Infinispan

Infinispan

https://infinispan.org

https://github.com/infinispan
(Apache License, v2.0)

- In-memory data grid platform, written in Java
- Schema-less (optionally), No-SQL key-value data store
- Distributed cache - offers massive memory
- Elastic and scalable - can run on hundreds of nodes
- Highly available - no SPOF, resilient to node failures
- Multi-version concurrency control (MVCC)
- Transactional
- Queryable
- Processing for streaming data

Infinispan

https://infinispan.org

https://github.com/infinispan
(Apache License, v2.0)

- In-memory data grid platform, written in Java
- Schema-less (optionally), No-SQL key-value data store
- Distributed cache - offers massive memory
- Elastic and scalable - can run on hundreds of nodes
- Highly available - no SPOF, resilient to node failures
- Multi-version concurrency control (MVCC)
- Transactional
- Queryable
- Processing for streaming data

Infinispan

https://infinispan.org

https://github.com/infinispan
(Apache License, v2.0)

- In-memory data grid platform, written in Java
- Schema-less (optionally), No-SQL key-value data store
- Distributed cache - offers massive memory
- Elastic and scalable - can run on hundreds of nodes
- Highly available - no SPOF, resilient to node failures
- Multi-version concurrency control (MVCC)
- Transactional
- Queryable
- Processing for streaming data

Infinispan

https://infinispan.org

https://github.com/infinispan
(Apache License, v2.0)

- In-memory data grid platform, written in Java
- Schema-less (optionally), No-SQL key-value data store
- Distributed cache - offers massive memory
- Elastic and scalable - can run on hundreds of nodes
- Highly available - no SPOF, resilient to node failures
- Multi-version concurrency control (MVCC)
- Transactional
- Queryable
- Processing for streaming data

Infinispan

https://infinispan.org

https://github.com/infinispan
(Apache License, v2.0)

- In-memory data grid platform, written in Java
- Schema-less (optionally), No-SQL key-value data store
- Distributed cache - offers massive memory
- Elastic and scalable - can run on hundreds of nodes
- Highly available - no SPOF, resilient to node failures
- Multi-version concurrency control (MVCC)
- Transactional
- Queryable
- Processing for streaming data

Infinispan

https://infinispan.org

https://github.com/infinispan
(Apache License, v2.0)

- In-memory data grid platform, written in Java
- Schema-less (optionally), No-SQL key-value data store
- Distributed cache - offers massive memory
- Elastic and scalable - can run on hundreds of nodes
- Highly available - no SPOF, resilient to node failures
- Multi-version concurrency control (MVCC)
- Transactional
- Queryable
- Processing for streaming data

Infinispan

https://infinispan.org

https://github.com/infinispan
(Apache License, v2.0)

- In-memory data grid platform, written in Java
- Schema-less (optionally), No-SQL key-value data store
- Distributed cache - offers massive memory
- Elastic and scalable - can run on hundreds of nodes
- Highly available - no SPOF, resilient to node failures
- Multi-version concurrency control (MVCC)
- Transactional
- Queryable
- Processing for streaming data

Infinispan

https://infinispan.org

https://github.com/infinispan
(Apache License, v2.0)

- In-memory data grid platform, written in Java
- Schema-less (optionally), No-SQL key-value data store
- Distributed cache - offers massive memory
- Elastic and scalable - can run on hundreds of nodes
- Highly available - no SPOF, resilient to node failures
- Multi-version concurrency control (MVCC)
- Transactional
- Queryable
- Processing for streaming data

Infinispan

https://infinispan.org

https://github.com/infinispan
(Apache License, v2.0)

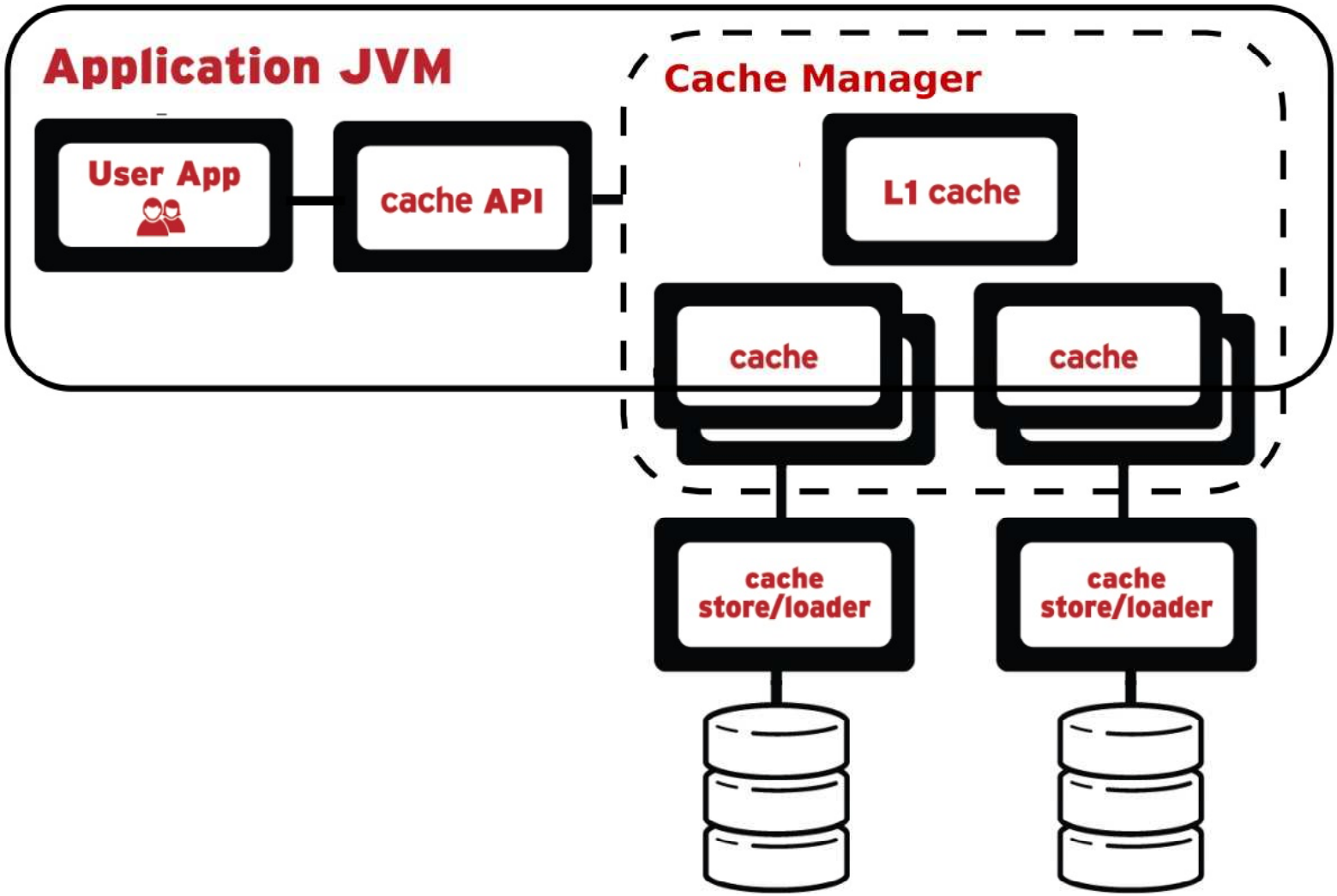
- In-memory data grid platform, written in Java
- Schema-less (optionally), No-SQL key-value data store
- Distributed cache - offers massive memory
- Elastic and scalable - can run on hundreds of nodes
- Highly available - no SPOF, resilient to node failures
- Multi-version concurrency control (MVCC)
- Transactional
- Queryable
- Processing for streaming data

Infinispan cache

- Infinispan takes care about all that hard stuff.
- From user perspective Infinispan cache is **just a map!**

```
1 DefaultCacheManager cacheManager = new DefaultCacheManager("
      my_ispn_config.xml");
2 Cache<String, String> cache = cacheManager.getCache("myCache");
3 cache.put("key", "value");
4 String value = cache.get("key");
```

Infinispan (embedded) high level architecture



Infinispan embedded tutorial

Simple weather app using embedded Infinispan

- http://infinispan.org/tutorials/embedded/
- https://github.com/infinispan/infinispan-embedded-tutorial

```
1 git clone https://github.com/infinispan/infinispan-embedded-
    tutorial.git
2 cd infinispan-embedded-tutorial
3 git checkout -f step-2
4 sed -i 's/<!-- a/<a/;s/t -->/t>/' pom.xml #switch to local
    random weather service
5 mvn clean package
6 mvn exec:exec
```

Commercial break: JGroups

JGroups is a toolkit for reliable messaging written in Java.
It can be used to create clusters whose nodes can send messages to each other.

Main features:

- Cluster creation and deletion. Cluster nodes can be spread across LANs or WANs.
- Membership detection and notification about joined/left/crashed cluster nodes.
- Sending and receiving of node-to-cluster messages (point-to-multipoint).
- Sending and receiving of node-to-node messages (point-to-point).
- Detection and removal of crashed nodes.

More about JGroups in upcoming **WildFly clustering course**!

Commercial break: JGroups

JGroups is a toolkit for reliable messaging written in Java.
It can be used to create clusters whose nodes can send messages to each other.

Main features:

- Cluster creation and deletion. Cluster nodes can be spread across LANs or WANs.
- Membership detection and notification about joined/left/crashed cluster nodes.
- Sending and receiving of node-to-cluster messages (point-to-multipoint).
- Sending and receiving of node-to-node messages (point-to-point).
- Detection and removal of crashed nodes.

More about JGroups in upcoming **WildFly clustering course**!

Commercial break: JGroups

JGroups is a toolkit for reliable messaging written in Java.
It can be used to create clusters whose nodes can send messages to each other.

Main features:

- Cluster creation and deletion. Cluster nodes can be spread across LANs or WANs.
- Membership detection and notification about joined/left/crashed cluster nodes.
- Sending and receiving of node-to-cluster messages (point-to-multipoint).
- Sending and receiving of node-to-node messages (point-to-point).
- Detection and removal of crashed nodes.

More about JGroups in upcoming **WildFly clustering course**!

Commercial break: JGroups

JGroups is a toolkit for reliable messaging written in Java.
It can be used to create clusters whose nodes can send messages to each other.

Main features:

- Cluster creation and deletion. Cluster nodes can be spread across LANs or WANs.
- Membership detection and notification about joined/left/crashed cluster nodes.
- Sending and receiving of node-to-cluster messages (point-to-multipoint).
- Sending and receiving of node-to-node messages (point-to-point).
- Detection and removal of crashed nodes.

More about JGroups in upcoming **WildFly clustering course**!

Commercial break: JGroups

JGroups is a toolkit for reliable messaging written in Java.
It can be used to create clusters whose nodes can send messages to each other.

Main features:

- Cluster creation and deletion. Cluster nodes can be spread across LANs or WANs.
- Membership detection and notification about joined/left/crashed cluster nodes.
- Sending and receiving of node-to-cluster messages (point-to-multipoint).
- Sending and receiving of node-to-node messages (point-to-point).
- Detection and removal of crashed nodes.

More about JGroups in upcoming **WildFly clustering course**!

Commercial break: JGroups

JGroups is a toolkit for reliable messaging written in Java.
It can be used to create clusters whose nodes can send messages to each other.

Main features:

- Cluster creation and deletion. Cluster nodes can be spread across LANs or WANs.
- Membership detection and notification about joined/left/crashed cluster nodes.
- Sending and receiving of node-to-cluster messages (point-to-multipoint).
- Sending and receiving of node-to-node messages (point-to-point).
- Detection and removal of crashed nodes.

More about JGroups in upcoming **WildFly clustering course**!

Clustering modes

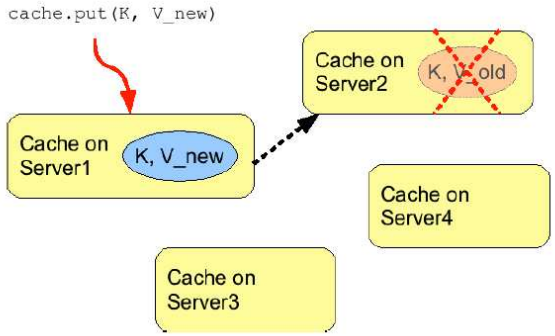
- Under the hood leverages JGroups project for clustering.
- Data is distributed and replicated in the background.
- Nodes can be added or removed smoothly.

Clustering modes

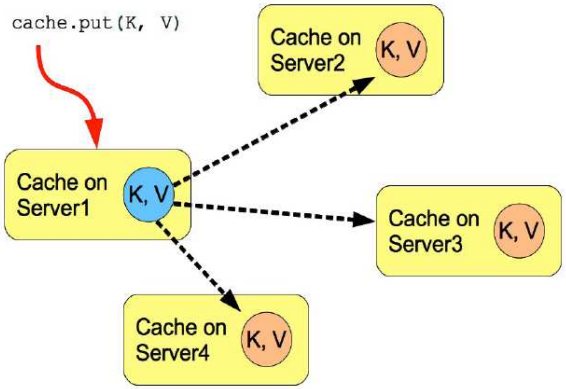
- Under the hood leverages JGroups project for clustering.
- Data is distributed and replicated in the background.
- Nodes can be added or removed smoothly.

- Local - no clustering

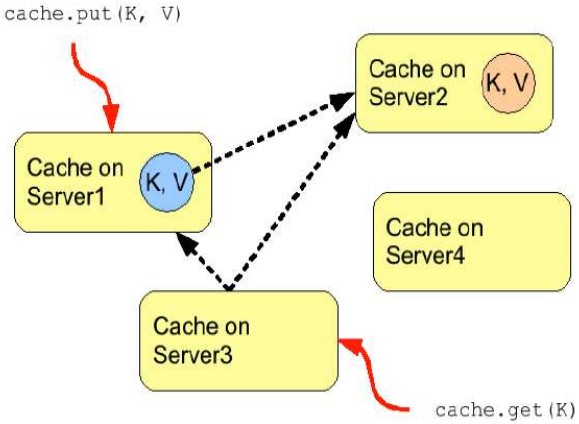
- Invalidation



- Replicated

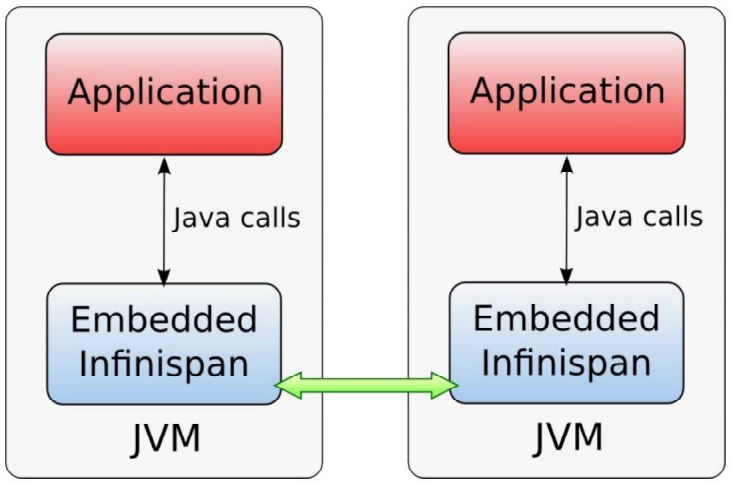


- Distributed

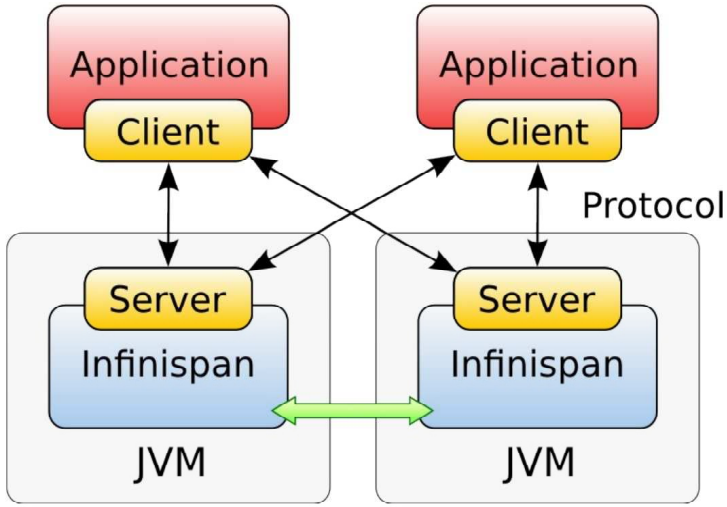


Infinispan modes

- Embedded (library, in-VM)

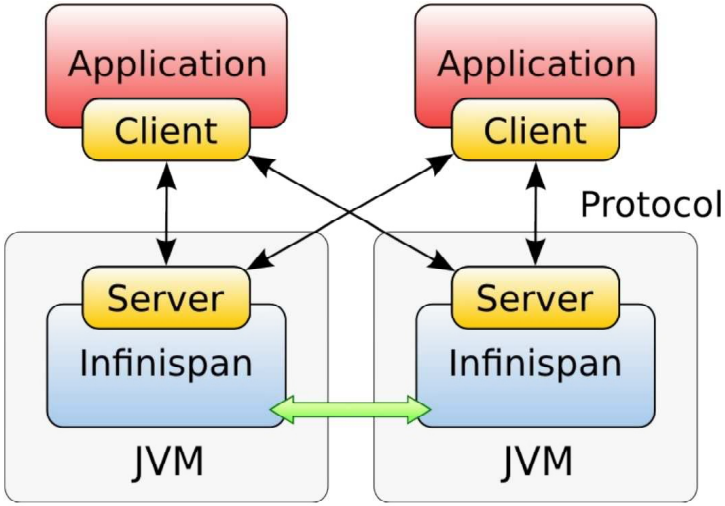


- Client-server (remote)



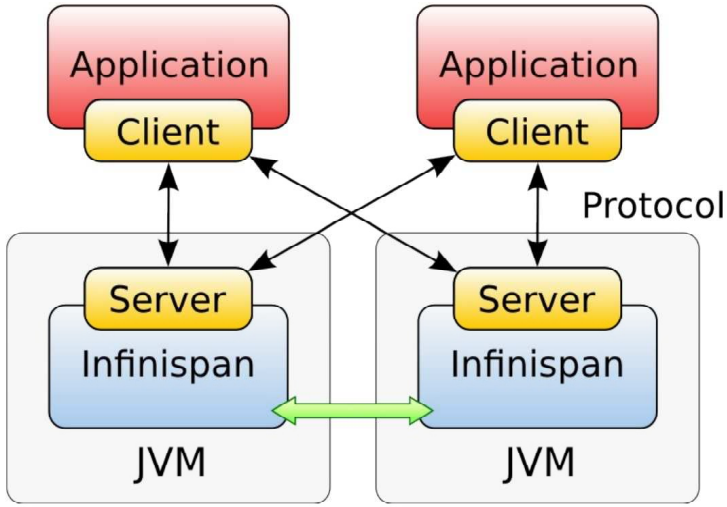
Remote protocols

- Hot Rod
  - hashing and topology aware
  - failover during topology changes
  - smart request routing
- Memcached
- REST



Remote protocols

- Hot Rod
  - hashing and topology aware
  - failover during topology changes
  - smart request routing
- Memcached
- REST



Protocol	Format	Client libs	Clustered	Smart routing	Load balancing / Failover
Hot Rod	binary	Java, C++, C#	yes	yes	dynamic
Memcached	text	many	yes	no	only predefined server list
REST	text	any HTTP client	yes	no	any HTTP load balancer

Hot Rod clients

Compatible with Java and non-Java platforms. Based on Protocol Buffers - Google's data interchange format.

Clients for

- Java
- C#
- C++
- Python
- Ruby

Python and Ruby clients have only basic functionality.

Commercial break: Protocol Buffers

Protocol Buffers (protobuf) are language-neutral, platform-neutral, extensible mechanism for serializing structured data developed by Google.

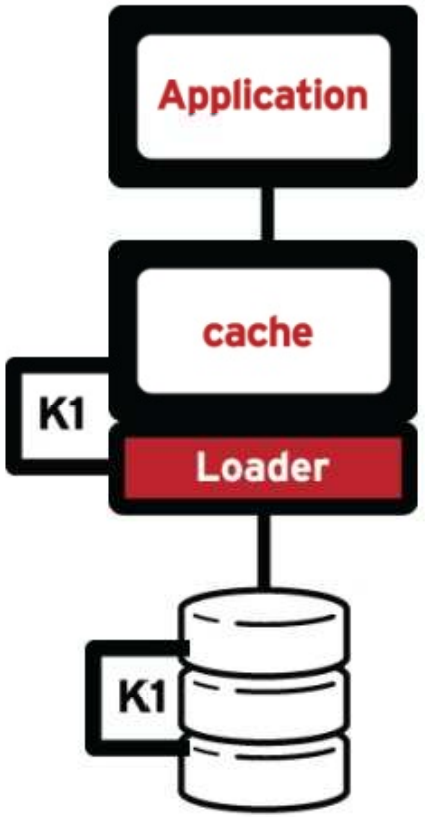
- Supports C++, C#, Go, Java, Python.
- You need to define data structure in protobuf file.
- In ISPN you can use also annotations in the your model.

Example of protobuf file:

```
1  message Address {
2      required string street = 1;
3      required string postCode = 2;
4  }
5
6   message Person {
7      optional int32 id = 1;
8      required string name = 2;
9      required string surname = 3;
10     optional Address address = 4;
11     optional string license = 5;
12     enum Gender {
13       MALE = 0;
14       FEMALE = 1;
15     }
16 }
```

Cache stores

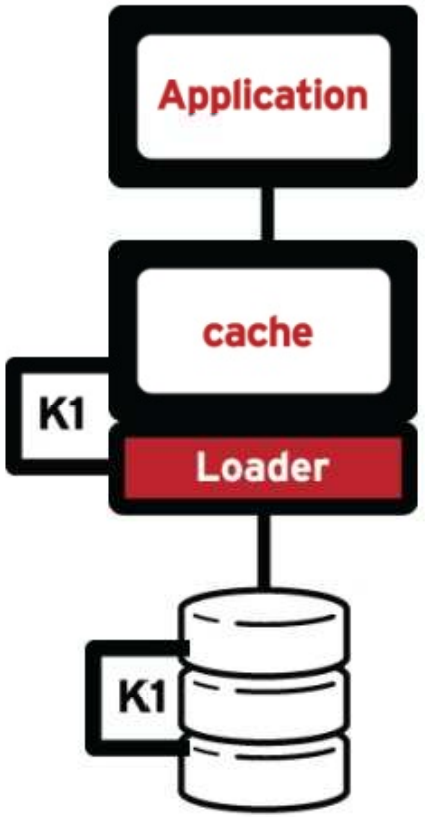
A way how to store cache content in some external (persistent) storage.



Cache stores

A way how to store cache content in some external (persistent) storage.
Two modes:

- Synchronous (write-through)
- Asynchronous (write-behind)



Cache stores

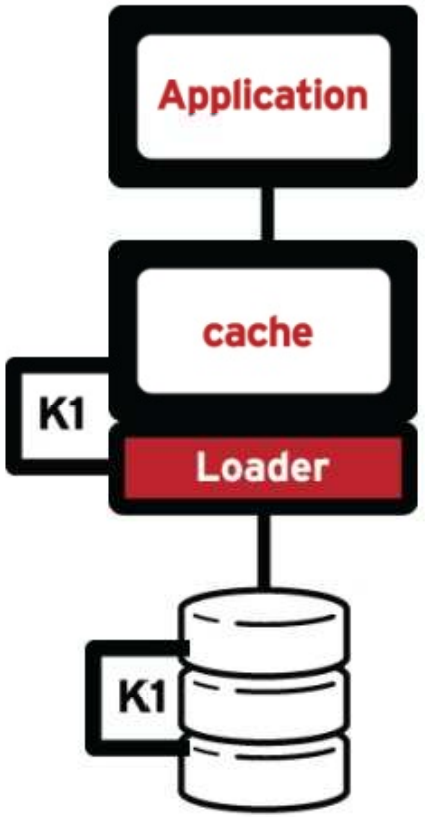
A way how to store cache content in some external (persistent) storage.
Two modes:

- Synchronous (write-through)
- Asynchronous (write-behind)

Cache stores:

- Single file store and soft-index file store
- JDBC and JPA cache stores
- LevelDB cache store
- Cloud cache store
- Remote store
- Cassandra store
- . . . and others

Also possible to define custom cache store.



Transactions, consistency, locking and isolation

- JTA-compliant transactions
- Deadlock detection and recovery (e.g. when ISPN fails during commit phase of the transaction)
- Data versioning
- Ensures consistency of data, consistency guarantee: lock for key `K` is always, acquired on the same node of the cluster (key **primary owner**), regardless of where the transaction originates

Transactions, consistency, locking and isolation

- JTA-compliant transactions
- Deadlock detection and recovery (e.g. when ISPN fails during commit phase of the transaction)
- Data versioning
- Ensures consistency of data, consistency guarantee: lock for key `K` is always, acquired on the same node of the cluster (key **primary owner**), regardless of where the transaction originates

Transactions, consistency, locking and isolation

- JTA-compliant transactions
- Deadlock detection and recovery (e.g. when ISPN fails during commit phase of the transaction)
- Data versioning
- Ensures consistency of data, consistency guarantee: lock for key `K` is always, acquired on the same node of the cluster (key **primary owner**), regardless of where the transaction originates

Transactions, consistency, locking and isolation

- JTA-compliant transactions
- Deadlock detection and recovery (e.g. when ISPN fails during commit phase of the transaction)
- Data versioning
- Ensures consistency of data, consistency guarantee: lock for key `K` is always, acquired on the same node of the cluster (key **primary owner**), regardless of where the transaction originates

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Transactions, consistency, locking and isolation (cont.)

- **Pessimistic** and **optimistic** locking available
  - Pessimistic locking: resource is locked all the time during the transaction (in ISPN when resource is changed, read is still possible).
  - Optimistic locking: state of the resource is saved at the beginning of the transaction (prepare phase) and other transactions ca access the resource. During commit phase of the resource is read again and if changed (write skew), transaction is rolled back.
- Isolation - how/when the changes made by one operation become visible to other. **Read committed** and **repeatable read** isolation levels.
  1. Thread1: tx.begin()
  2. Thread1: cache.get(k) returns v
  3. Thread2: tx.begin()
  4. Thread2: cache.get(k) returns v
  5. Thread2: cache.put(k, v2)
  6. Thread2: tx.commit()
  7. Thread1: cache.get(k)

  With `REPEATABLE_READ`, step 7 will still return `v`, while with `READ_COMMITTED` step 7 will return `v2`.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying

- Support for indexing and searching of objects stored in the cache.
- Search for data using data attributes instead of keys.
- Uses Hibernate Search and Apache Lucene to index and search objects.
- Queries can be constructed using ISPN fluent DSL API, Hibernate Search Query DSL or directly Lucene query API.
- Needs some data schema (protobuf file or annotations).
- Combine queries and aggregation functions (but doesn't support joins).
- Sort, filter, and paginate query results.
- Support for index or non-indexed queries.

Querying: example

```
1  @Indexed
2  public class Person {
3    @Field(store = Store.YES, analyze = Analyze.NO)
4    String name;
5
6    @Field(store = Store.YES, analyze = Analyze.NO, indexNullAs
         = Field.DEFAULT_NULL_TOKEN)
7    String surname;
8
9    public Person(String name, String surname) {
10     this.name = name;
11     this.surname = surname;
12   }
13 }
```

```java
public class InfinispanQuery {
    public static void main(String[] args) {
      ConfigurationBuilder b = new ConfigurationBuilder();
      b.indexing().index(Index.ALL).addProperty("default.
          directory_provider", "ram").addProperty("
          lucene_version", "LUCENE_CURRENT");
      DefaultCacheManager cm = new DefaultCacheManager(builder.
          build());
      Cache<String, Person> cache = cm.getCache();
      cache.put("person1", new Person("Will", "Shakespeare"));
      // Obtain a query factory for the cache
      QueryFactory<?> queryFactory = Search.getQueryFactory(
          cache);
      // Construct a query
      Query query = queryFactory.from(Person.class).having("
          name").eq("Will").toBuilder().build();
      // Execute the query
      List<Person> matches = query.list();
      matches.forEach(person -> System.out.printf("Match: %s",
          person));
      cacheManager.stop();
    }
}
```

Security

- Role based access control
- User authentication
- Node authentication and authorization
- Encryption of communication
- Audit logging
- Integration with LDAP and/or Kerberos server (includes Active Directory)

Security

- Role based access control
- User authentication
- Node authentication and authorization
- Encryption of communication
- Audit logging
- Integration with LDAP and/or Kerberos server (includes Active Directory)

Security

- Role based access control
- User authentication
- Node authentication and authorization
- Encryption of communication
- Audit logging
- Integration with LDAP and/or Kerberos server (includes Active Directory)

Security

- Role based access control
- User authentication
- Node authentication and authorization
- Encryption of communication
- Audit logging
- Integration with LDAP and/or Kerberos server (includes Active Directory)

Security

- Role based access control
- User authentication
- Node authentication and authorization
- Encryption of communication
- Audit logging
- Integration with LDAP and/or Kerberos server (includes Active Directory)

Security

- Role based access control
- User authentication
- Node authentication and authorization
- Encryption of communication
- Audit logging
- Integration with LDAP and/or Kerberos server (includes Active Directory)

Some other features - brief and selective list

- Data eviction and expiration
- Full JSR-107 support (Java Temporary Caching API)
- CDI support
- Remote events
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors

Some other features - brief and selective list

- Data eviction and expiration
- Full JSR-107 support (Java Temporary Caching API)
- CDI support
- Remote events
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors

Some other features - brief and selective list

- Data eviction and expiration
- Full JSR-107 support (Java Temporary Caching API)
- CDI support
- Remote events
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors

Some other features - brief and selective list

- Data eviction and expiration
- Full JSR-107 support (Java Temporary Caching API)
- CDI support
- Remote events
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors

Some other features - brief and selective list

- Data eviction and expiration
- Full JSR-107 support (Java Temporary Caching API)
- CDI support
- Remote events
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors

Some other features - brief and selective list

- Data eviction and expiration
- Full JSR-107 support (Java Temporary Caching API)
- CDI support
- Remote events
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors

Some other features - brief and selective list

- Data eviction and expiration
- Full JSR-107 support (Java Temporary Caching API)
- CDI support
- Remote events
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors

Some other features - brief and selective list

- Data eviction and expiration
- Full JSR-107 support (Java Temporary Caching API)
- CDI support
- Remote events
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors

Some other features - brief and selective list

- Data eviction and expiration
- Full JSR-107 support (Java Temporary Caching API)
- CDI support
- Remote events
- Client near cache
- Rolling upgrades
- Cross data center replication (also Hot Rod clients support failover to another data center)
- Command line interface
- Distributed executors

Recently implemented features: Infinispan 8

- Functional API
- Distributed streams
- Continuous querying, grouping and aggregation
- New management console
- Integration with Apache Spark and Hadoop
- . . . and more

Recently implemented features: Infinispan 8

- Functional API
- Distributed streams
- Continuous querying, grouping and aggregation
- New management console
- Integration with Apache Spark and Hadoop
- . . . and more

Recently implemented features: Infinispan 8

- Functional API
- Distributed streams
- Continuous querying, grouping and aggregation
- New management console
- Integration with Apache Spark and Hadoop
- . . . and more

Recently implemented features: Infinispan 8

- Functional API
- Distributed streams
- Continuous querying, grouping and aggregation
- New management console
- Integration with Apache Spark and Hadoop
- . . . and more

Recently implemented features: Infinispan 8

- Functional API
- Distributed streams
- Continuous querying, grouping and aggregation
- New management console
- Integration with Apache Spark and Hadoop
- . . . and more

Integration with other frameworks

- Hibernate
  - 2-nd level cache
- Lucene directory
  - In-memory Lucene index
- Apache Camel
  - Infinispan component for Camel
- Hadoop
  - In-memory data source for Hadoop cluster
- Apache Spark
  - Data source for Spark map-reduce jobs

Integration with other frameworks

- Hibernate
  - 2-nd level cache
- Lucene directory
  - In-memory Lucene index
- Apache Camel
  - Infinispan component for Camel
- Hadoop
  - In-memory data source for Hadoop cluster
- Apache Spark
  - Data source for Spark map-reduce jobs

Integration with other frameworks

- Hibernate
  - 2-nd level cache
- Lucene directory
  - In-memory Lucene index
- Apache Camel
  - Infinispan component for Camel
- Hadoop
  - In-memory data source for Hadoop cluster
- Apache Spark
  - Data source for Spark map-reduce jobs

Integration with other frameworks

- Hibernate
  - 2-nd level cache
- Lucene directory
  - In-memory Lucene index
- Apache Camel
  - Infinispan component for Camel
- Hadoop
  - In-memory data source for Hadoop cluster
- Apache Spark
  - Data source for Spark map-reduce jobs

Integration with other frameworks

- Hibernate
  - 2-nd level cache
- Lucene directory
  - In-memory Lucene index
- Apache Camel
  - Infinispan component for Camel
- Hadoop
  - In-memory data source for Hadoop cluster
- Apache Spark
  - Data source for Spark map-reduce jobs

Integration with other frameworks

- Hibernate
  - 2-nd level cache
- Lucene directory
  - In-memory Lucene index
- Apache Camel
  - Infinispan component for Camel
- Hadoop
  - In-memory data source for Hadoop cluster
- Apache Spark
  - Data source for Spark map-reduce jobs

Summary

- Amount and structure of the data has changed rapidly during past couple of years.
- Cloud applications and Big/Fast data require new approaches and tools, data grids are important building blocks of such solutions.
- Infinispan is mature and feature rich data grid solution, which integrates well with other frameworks and can be used as backbone for new generation of enterprise applications.

Summary

- Amount and structure of the data has changed rapidly during past couple of years.
- Cloud applications and Big/Fast data require new approaches and tools, data grids are important building blocks of such solutions.
- Infinispan is mature and feature rich data grid solution, which integrates well with other frameworks and can be used as backbone for new generation of enterprise applications.

Summary

- Amount and structure of the data has changed rapidly during past couple of years.
- Cloud applications and Big/Fast data require new approaches and tools, data grids are important building blocks of such solutions.
- Infinispan is mature and feature rich data grid solution, which integrates well with other frameworks and can be used as backbone for new generation of enterprise applications.

Summary

- Amount and structure of the data has changed rapidly during past couple of years.
- Cloud applications and Big/Fast data require new approaches and tools, data grids are important building blocks of such solutions.
- Infinispan is mature and feature rich data grid solution, which integrates well with other frameworks and can be used as backbone for new generation of enterprise applications.

Materials from this course

- This presentation: https://github.com/vjuranek/presentations/tree/master/CTU_Prague2016
- ISPN embedded tutorial (The Weather App): http://infinispan.org/tutorials/embedded
- GitHub repo: https://github.com/infinispan/infinispan-embedded-tutorial
- ISPN simple tutorials: https://github.com/infinispan/infinispan-simple-tutorials
- ISPN qickstarts (simple applications) at the bottom of the page: http://infinispan.org/tutorials
- Some more ISPN snippets: https://github.com/vjuranek/infinispan-snippets

Infinispan downloads:

- Main ISPN download page: http://infinispan.org/download/
- If you want to play with ISPN in Docker: https://hub.docker.com/r/jboss/infinispan-server/

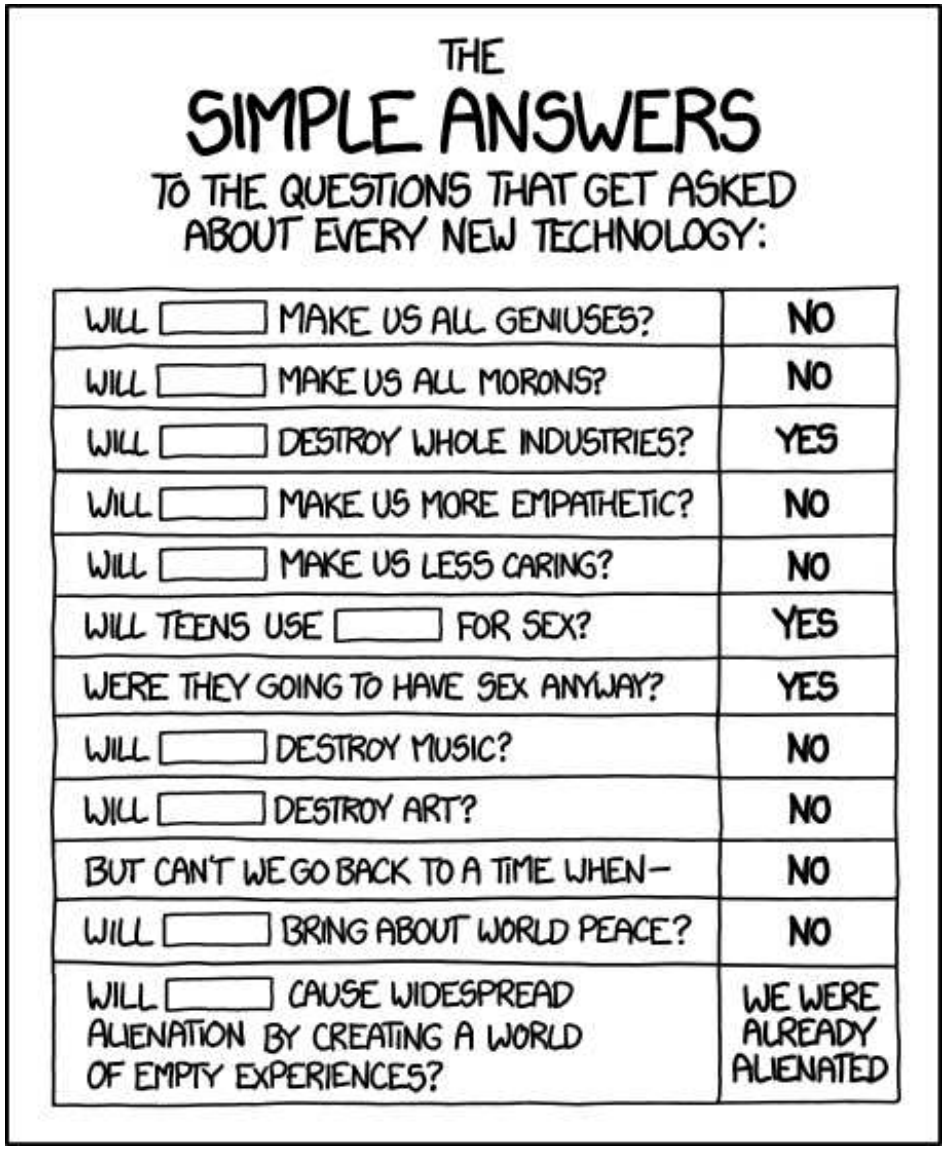
Further study materials

- Infinispan documentation
- JSR 107: JCACHE - Java Temporary Caching API
- M. Surtani, F. Marchioni, Infinispan Data Grid Platform, Packt Publishing, 2012
- W. dos Santos, Infinispan Data Grid Platform Definitive Guide, Packt Publishing, 2015
- M. Kleppmann, Designing Data-Intensive Applications, O'Reilly Media, Inc., 2016

- B. Burke, A. Rubinger, Enterprise JavaBeans 3.1, 6th Edition, O'Reilly Media, Inc., 2010

- Coursera: Cloud Computing Concepts
- Coursera: Cloud Computing Concepts: Part 2
- Coursera: Cloud Computing Applications

Question?



Source: `https://xkcd.com/1289/`

http://infinispan.org/

Thank you for your attention!