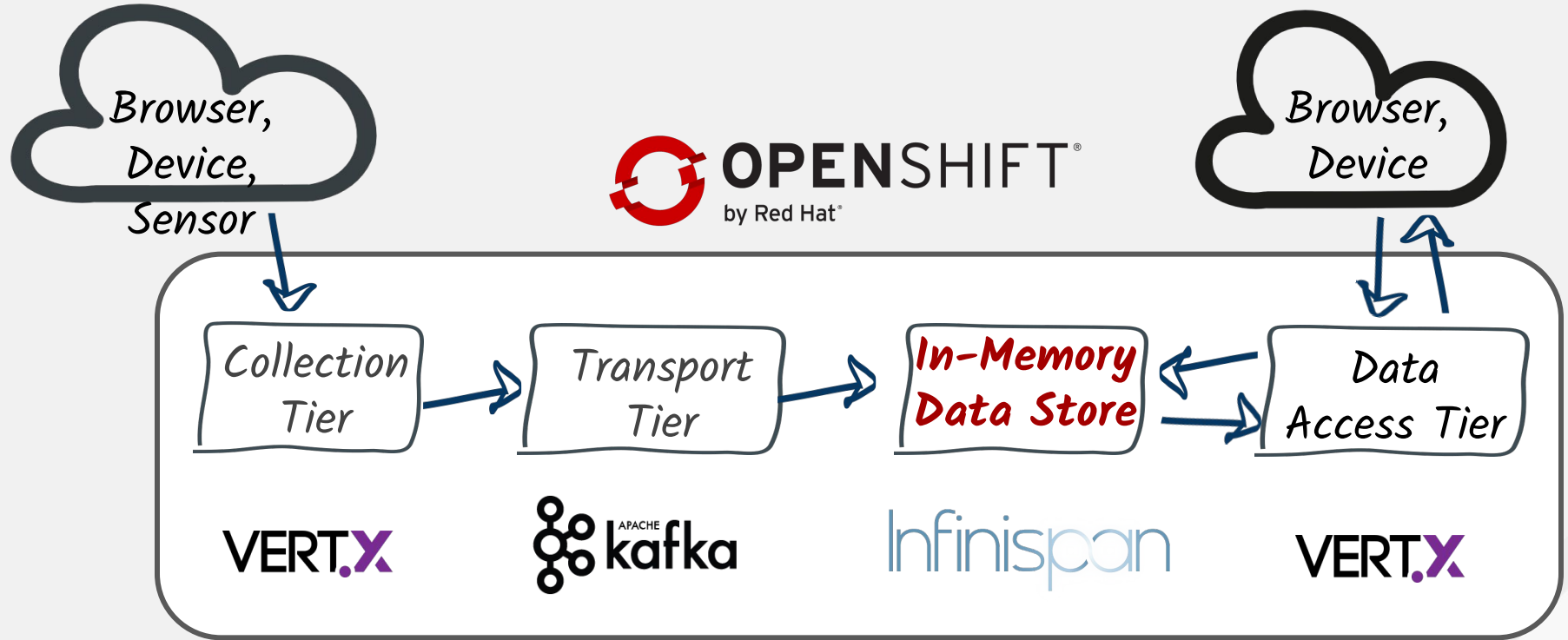




## In Memory Data Grid - Infinispan

# Streaming Data - Deep Dive



# *In-memory Data Store*

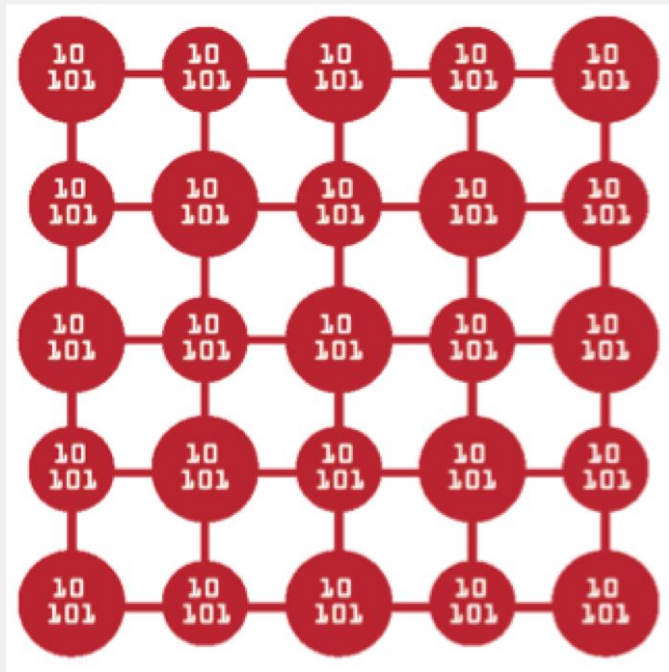
# WHAT IS INFINISPAN?

Distributed, in-memory  
key/value data store

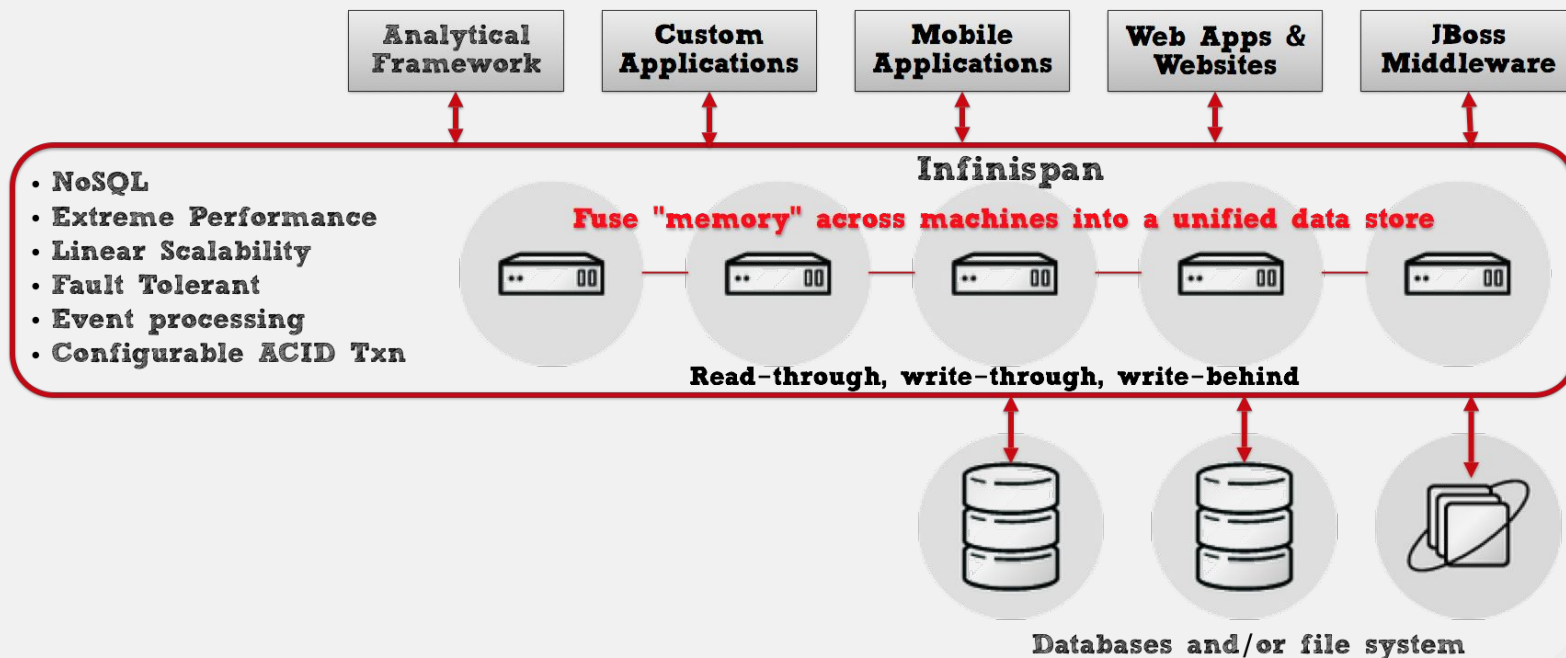
Uses peer-to-peer communication  
between nodes

No master/slaves  
No single bottleneck or point of failure

Designed for commodity hardware



# WHAT IS INFINISPAN?



# WHAT IS INFINISPAN USED FOR?

Distributed Cache

INSERT  
ICON  
HERE

Data Analysis

INSERT  
ICON  
HERE

Temporary Data Store

INSERT  
ICON  
HERE

Event-Driven Computation

INSERT  
ICON  
HERE

# DISTRIBUTED CACHE

Boost performance of application

Store frequently accessed data from slow systems, e.g. DB or web services

Store data that's hard to compute (e.g. scientific apps)

Example: Hibernate second-level cache

# TEMPORARY DATA STORE

Flexible way to store data without constraints of fixed data model

Temporary data can failover so it survives if machines go down

Can be persisted for recovery/backup/archiving

Data updates can participate in transactions

Example: HTTP session in-memory store for application servers



# DATA ANALYSIS

Click to add subtitle

Analysed in-memory stored data

Java Streams API extended to run in distributed environment and analyse data

More complex data analytics via Spark/Hadoop APIs combined with Infinispan backend

# EVENT-DRIVEN COMPUTATION

Supports event-driven computing via listeners

Trigger execution of application logic as data changes in Infinispan

Bringing data closer to processing can help reduce event response latencies

Real time computation can be very useful for fraud detection or risk analysis use cases

# USE CASE EXAMPLES

## WEB E-COMMERCE

Http session

Shopping carts

## DATABASE LEGACY OFFLOAD

Product catalog

Caching

## TELCO

Cellular billings

Call routing,  
session info

SMS content /  
notification

# USE CASE EXAMPLES

## TRAVEL

Aggregated flight  
pricing

Availability of  
flights

## FINANCIAL

Per-user portfolio  
data

Risk analysis

Aggregated  
ticker stream

## DEFENSE

Sensor network  
data process

Threat detection

# ROLE OF INFINISPAN

Click to add subtitle

## *Mixed*

- Temporary data store for analysis
- Event-driven computation for notifying interested parties

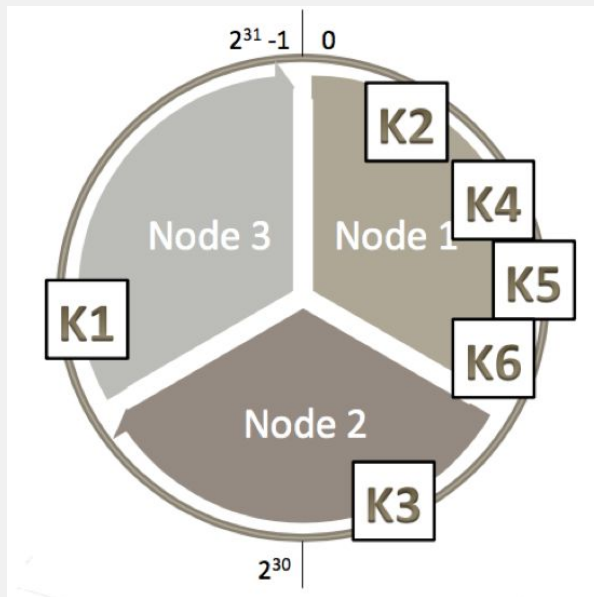
# HOW DOES INFINISPAN DISTRIBUTE DATA?

Consistent-Hash based data ownership

N copies kept for each key/value pair

Algorithm applied to key to decide owner

Automatic rebalancing



# ACCESSING INFINISPAN - EMBEDDED

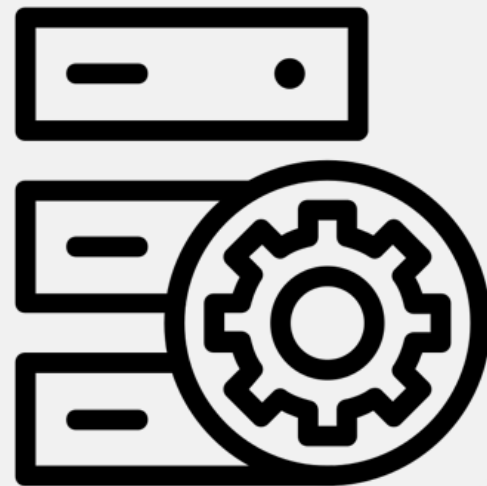
Application and data live in same JVM

Infinispan used as embedded library

Distribute data running your app in multiple nodes

Low latency but harder to tune

**Example:** Wildfly application server



# ACCESSING INFINISPAN

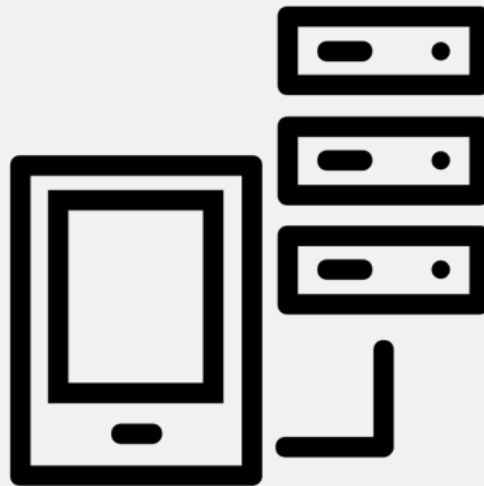
## Remote

Application and data separated by network

Data accessible via:

- Binary protocol (Hot Rod)
- HTTP REST protocol

App and data can be scaled and tuned independently





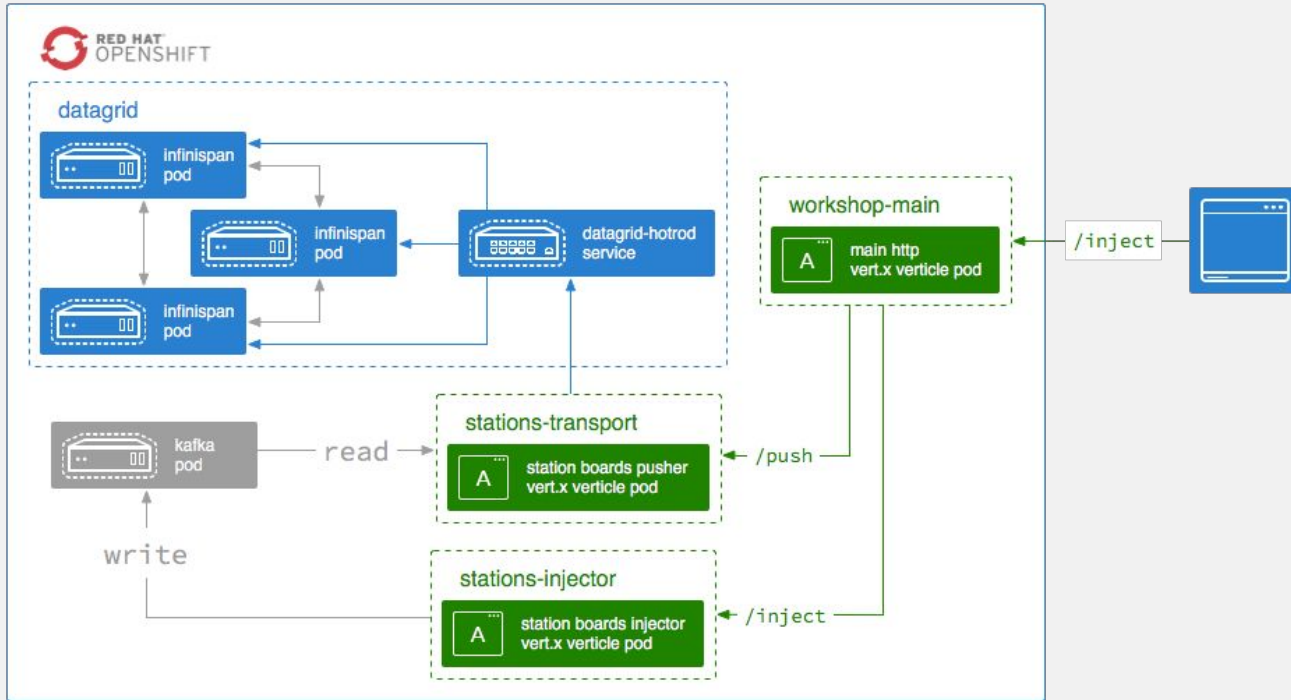
# STORING DATA REMOTELY

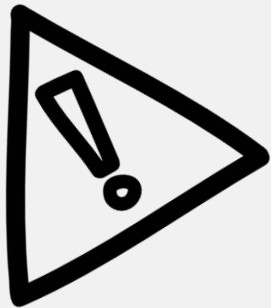
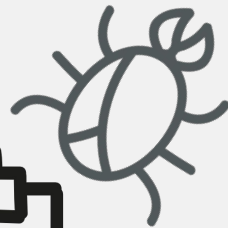
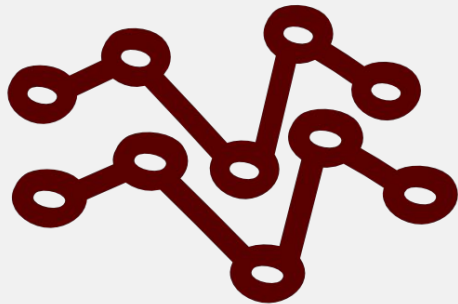
```
RemoteCacheManager cacheContainer = new  
RemoteCacheManager();
```

```
RemoteCache<String, String> cache =  
cacheContainer.getCache();
```

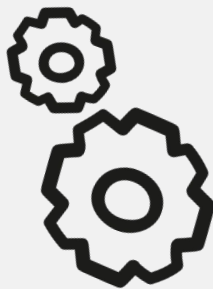
```
CompletableFuture<String> completableFuture =  
cache.putAsync("car", "volvo");
```

# DIAGRAM





*DEMO TIME!*



# *From Kafta to Infinispan*

# RETRIEVING DATA REMOTELY

```
RemoteCache<String, String> cache =  
    cacheContainer.getCache();
```

**cache**

```
.putAsync("car", "volvo")  
.thenCompose(prev -> cache.getAsync("car"))  
.thenAccept(v -> {  
    assert v.equals("volvo");  
});
```

# QUERYING DATA REMOTELY

Binary data cannot be queried as is

Binary data needs to be unmarshalled into POJOs to understand structure

Infinispan uses **protobuf** as language independent POJO structure

Once in POJO format, queries can be executed...

# QUERY BUILDER API

```
QueryFactory queryFactory =  
    Search.getQueryFactory(cache);
```

```
Query query = queryFactory.from(Stop.class)  
    .having("delayMin").gt(0L)  
    .build();
```

# ICKLE QUERY LANGUAGE

```
QueryFactory queryFactory =  
    Search.getQueryFactory(cache);
```

```
Query query = queryFactory.create(  
    "select p.name from model.Person p where name =  
    :personName");
```

```
query.setParameter("personName", "katia");
```



# CONTINUOUS QUERY

Query associated with a live-updating listener

Listener invoked for each entry that matches query

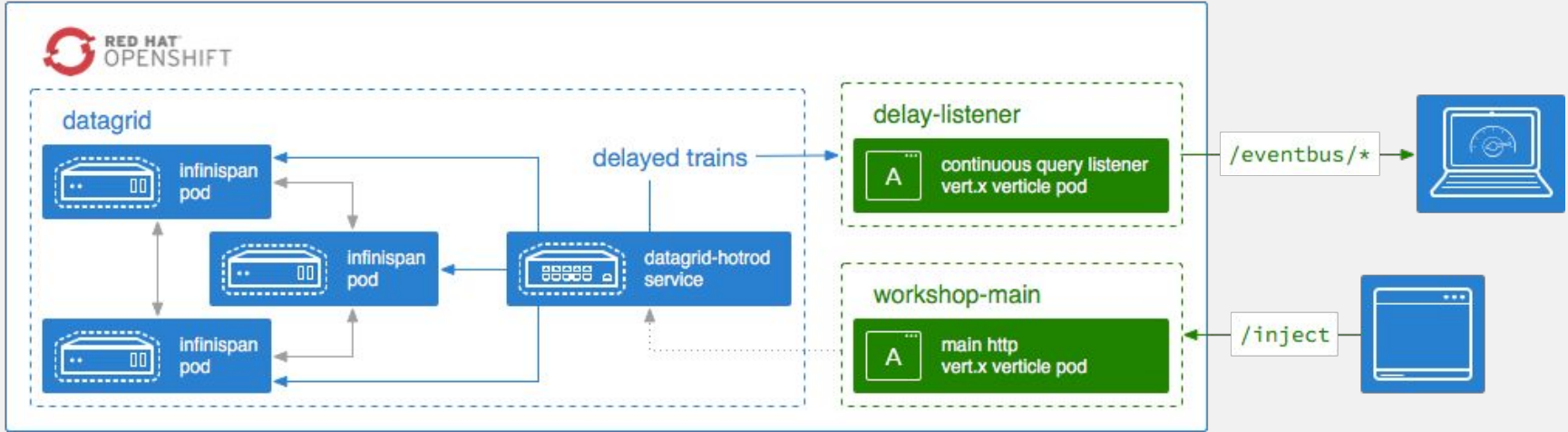
As data gets added that matches query, listener is invoked

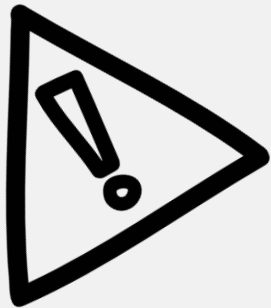
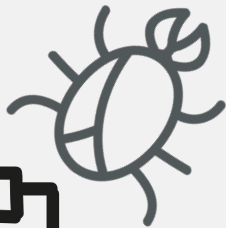
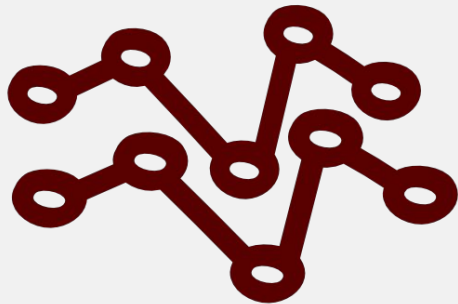
If data that was part of result set is deleted, the listener can also get invoked

# CONTINUOUS QUERY

```
public interface ContinuousQueryListener<K, V> {  
    default void resultJoining(K key, V value) {}  
    default void resultUpdated(K key, V value) {}  
    default void resultLeaving(K key) {}  
}
```

# DIAGRAM





*DEMO TIME!*

