DISTRIBUTED DATA PROCESSING WITH INFINISPAN AND JAVA STREAMS

Galder Zamarreño Arrizabalaga

21st January 2016

Infinispan

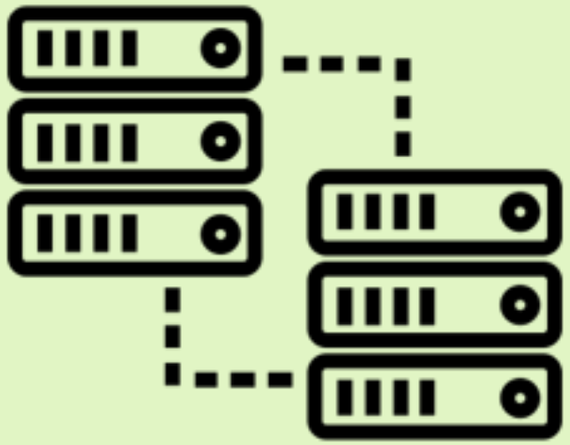
MOI

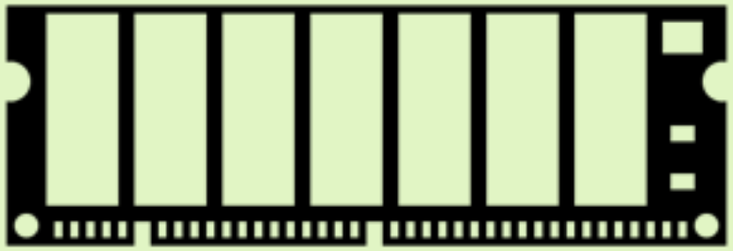
-  @  redhat.
- Infinispan co-founder
- JSR-107 
- Scala developer since 2009
- Functional programming 

@GALDERZ

WHAT IS INFINISPAN?

NOT JUST NOSQL



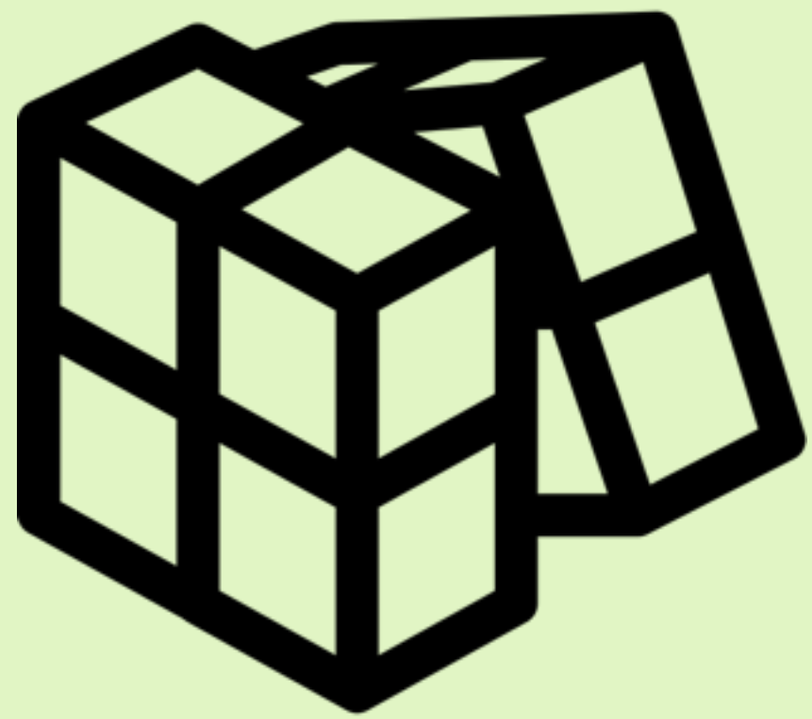


key/value data store
with optional schema,
available under the
ASL 2.0 license

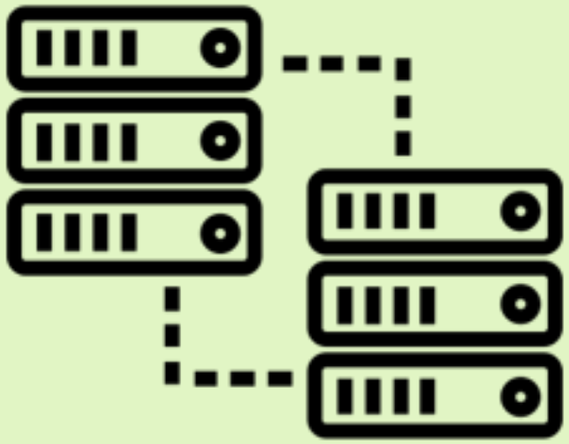
FROM LOCAL...

Store data from slow systems

Store data hard to compute



...VIA TEMPORARY...

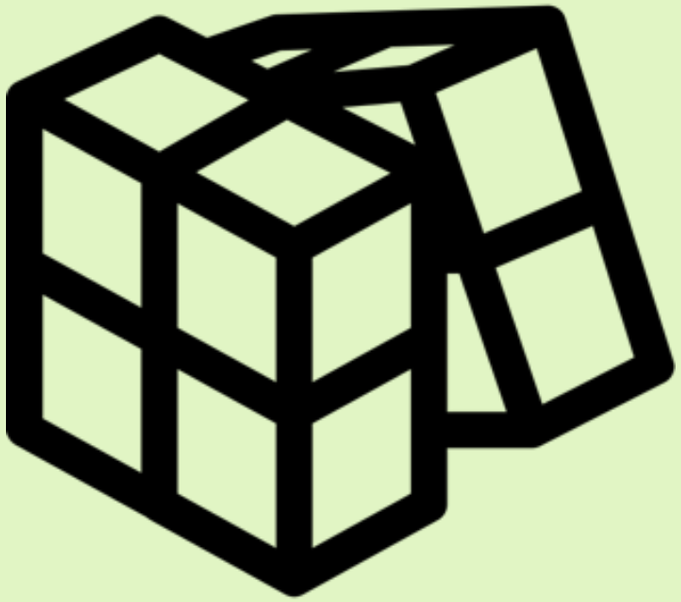
Store temporary data in 

for data that should survive

, e.g. 

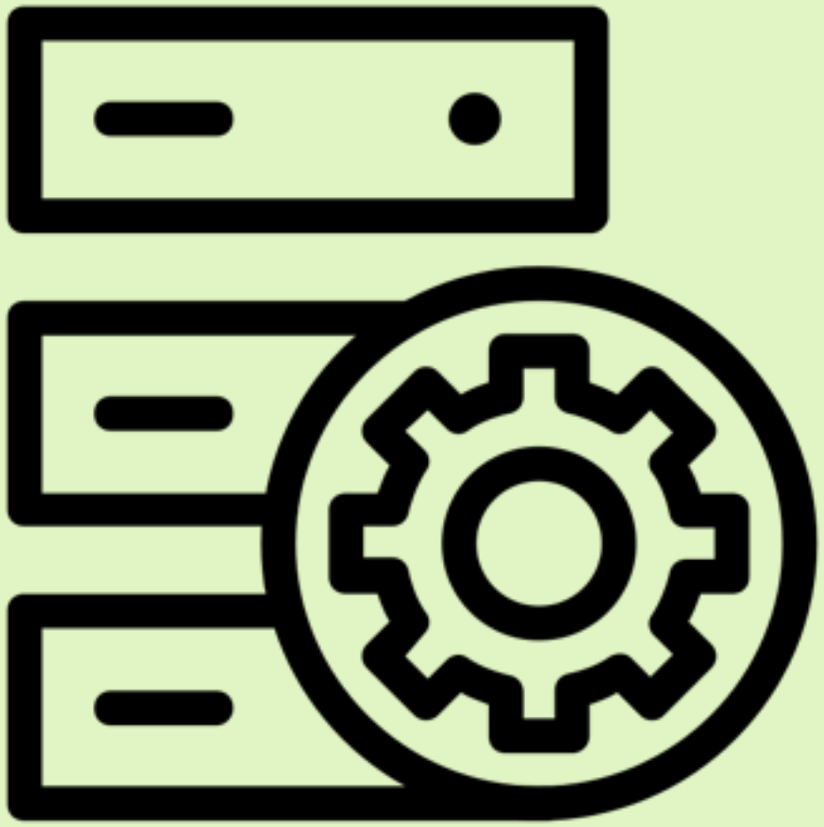
... TO DATA GRIDS

Used as primary store for :



ACCESS MODE

Application and data live in same JVM

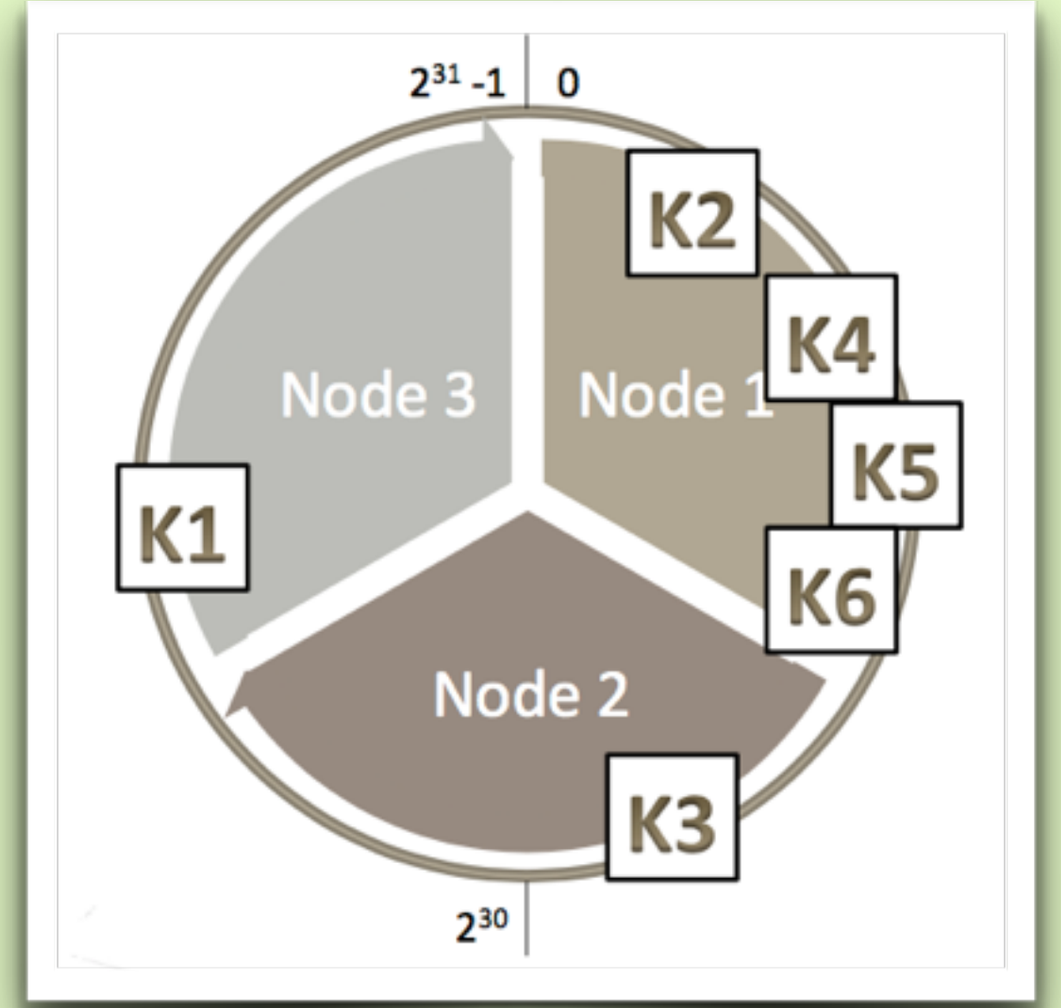


ACCESS MODE

Application and data separated by network

CLUSTERING

- Distribution mode
- N copies of data in cluster
- Data location defined by Consistent Hash

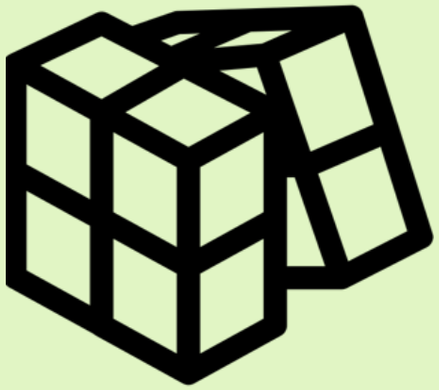


STORE & RETRIEVE

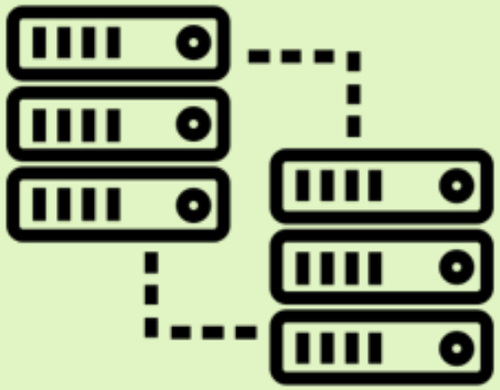
- ConcurrentMap
- JSR-107 Cache
- CDI
- SpringCache
- Experimental Functional API

COMPUTE

Extended Java 8 Stream API to



data stored in



JAVA 8 STREAM

```
List<Integer> numbers = Arrays.asList(
  4, 74, 20, 97, 118, 50, 97, 34, 48);
numbers.stream()
  .filter(i -> i > 70)
  // ^ Returns Stream<Integer>
  .map(n -> new String(Character.toChars(n)))
  // ^ Returns Stream<String>
  .reduce("", String::concat);
```

Returns "Java"

LAZYNESS

Does nothing

```
IntStream iterStream = IntStream.iterate(0, i -> i + 1);
```

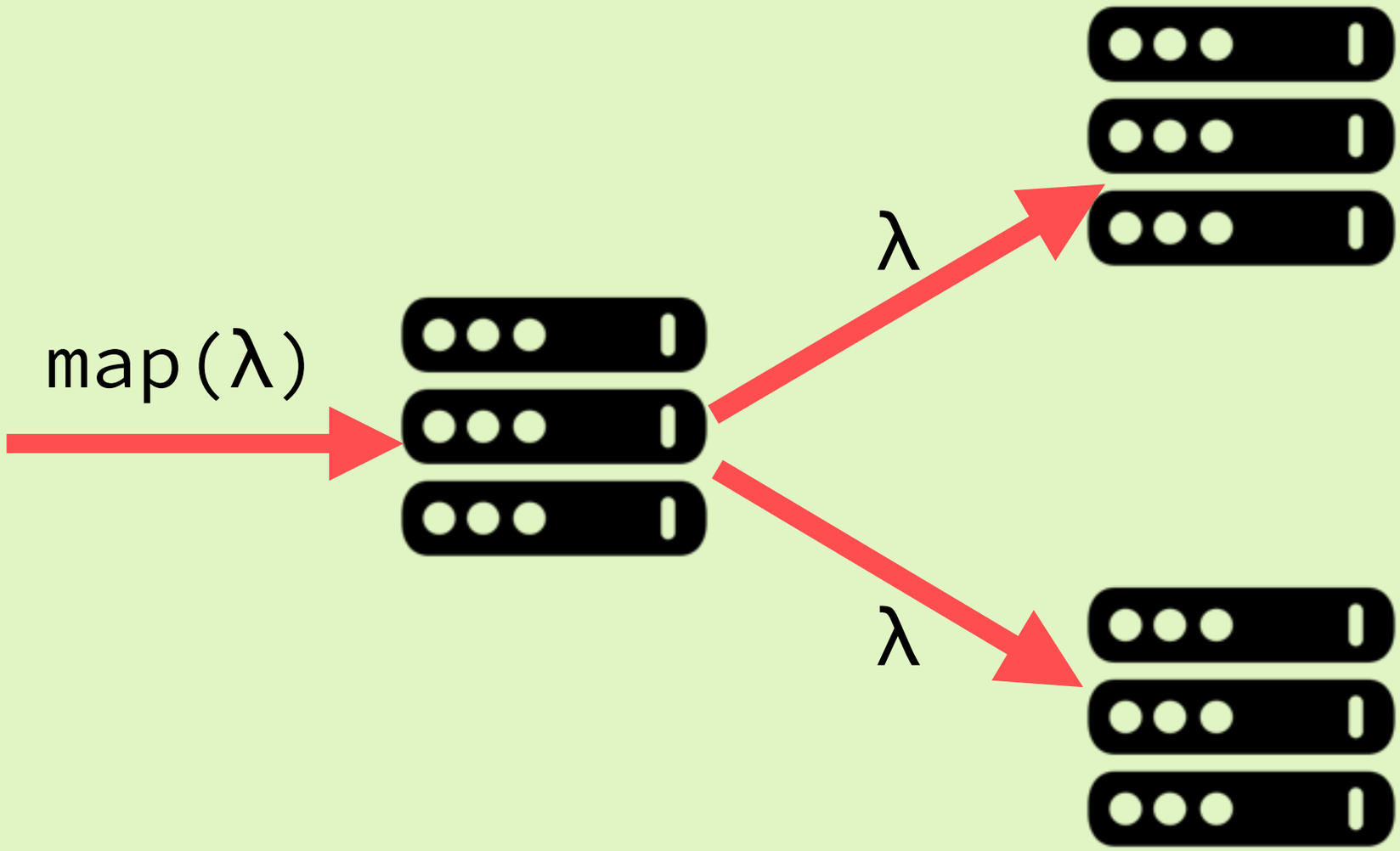
Prints 1 to 10

```
IntStream.iterate(0, i -> i + 1)
   .limit(10) // Returns IntStream
   .forEach(System.out::println); // Returns void
```

Runs forever :(

```
IntStream.iterate(0, i -> i + 1)
   .forEach(System.out::println);
```

DISTRIBUTED STREAMS



SHIPPING LAMBDAS

- Cast lambda to Serializable

```
numbers.stream()
  .filter((Serializable & Predicate<Integer>) i -> i > 70)
```

514 BYTES :(

SHIPPING LAMBDAS

- Use @SerializeFunctionWith

```
Predicate<Integer> pserwith = new PredicateSerializeWith();

@SerializeFunctionWith(PredicateSerializeWithExt.class)
private static final class PredicateSerializeWith
      implements Predicate<Integer> {
   public boolean test(Integer i) { return i > 70; }
}

public static final class PredicateSerializeWithExt
      implements Externalizer<Object> {
   public void writeObject(ObjectOutput oo, Object o) {}
   public Object readObject(ObjectInput input) {
      return new PredicateSerializeWith();
   }
}
```

272 BYTES :|

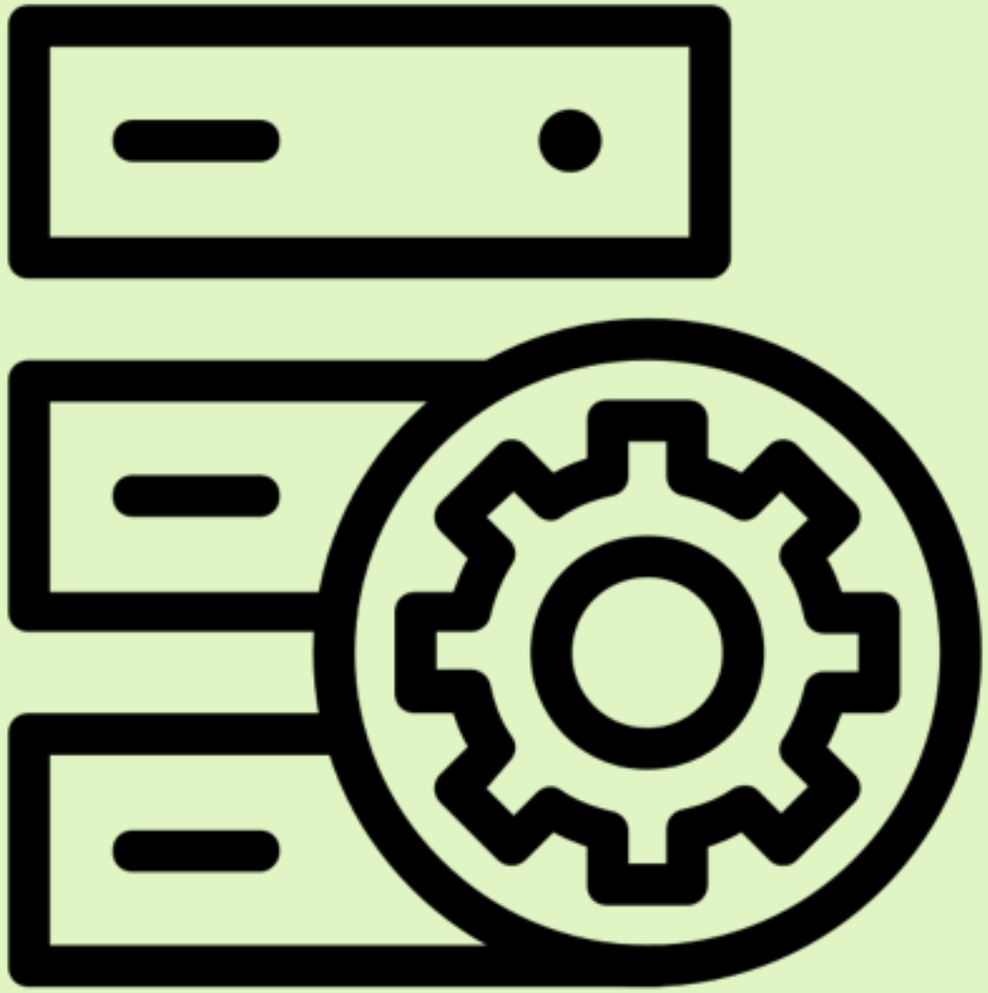
SHIPPING LAMBDAS

- Pre-register externalizer

```
static class PredicateInteger implements Predicate<Integer> {
   public boolean test(Integer i) { return i > 70; }
}

public static final class PredicateIntegerExt implements AdvancedExternalizer<Object> {
   public void writeObject(ObjectOutput oo, Object o) {}
   public Object readObject(ObjectInput input) {
      return new PredicateInteger();
   }

   public Set<Class<?>> getTypeClasses() {
      return Util.<Class<? extends Object>>asSet(PredicateInteger.class);
   }

   public Integer getId() { return 1234; }
}

GlobalConfigurationBuilder global = ...
global.serialization().addAdvancedExternalizer(new PredicateIntegerExt());
EmbeddedCacheManager cm = new DefaultCacheManager(global.build());
```

7 BYTES :)

DEMO



TOPOLOGY CHANGES

- Streams processed without data loss when topology changes
- Retries might happen…
- Strive for idempotent lambdas
- Idempotent forEach tricky…

SPECIAL INTERMEDIATE OPERATIONS

- distinct → origin + remote
- limit → origin + remote
- skip/peek → origin only
- sorted → origin only

MEMORY ++

REMOTE STREAMS

REMOTE SCRIPTS

word-count.js

```
// mode=local,language=javascript
var Function = Java.type("java.util.function.Function")
var Collectors = Java.type("java.util.stream.Collectors")
var Arrays = Java.type("org.infinispan.scripting.utils.JSArrays")
cache
   .entrySet().stream()
   .map(function(e) e.getValue())
   .map(function(v) v.toLowerCase())
   .map(function(v) v.split(/[\W]+/))
   .flatMap(function(f) Arrays.stream(f))
   .collect(Collectors.groupingBy(Function.identity(),
      Collectors.counting()));
```

```
RemoteCache<Integer, String> remoteCache = ...
Map<String, Long> results = remoteCache.execute("word-count.js",
    Collections.emptyMap());
```

"The Streams API will internally decompose your query to leverage the multiple cores on your computer."

Raoul-Gabriel Urma

"Infinispan Distributed Streams API will internally decompose your query to leverage the computing power of multiple machines"

Galder Zamarreño

SPARK/HADOOP INTEGRATION

- Suits Spark/Hadoop users wanting different backend
- Need to process data real-time, e.g. sliding windows
- Remote access only

QUERY API

- Find data looking at values
  - e.g. full text search
- Based on Lucene and Hibernate Search
- Lucene Query and Query DSL
- Remote Querying with Protobuf

CONTINUOUS QUERY

- Continuous query matching
- Incoming & outgoing matches
- Improved efficiency

WHICH API?

- Start → Java Stream API
- Spark/Hadoop require management/configuration
- Query API helps with deep understanding of values

SUMMARY

- Infinispan...
  - is a distributed K/V store
  - expands Java Streams to run in multi-node environments
  - offers more options for processing data: Query API, Spark/Hadoop...etc

CREDITS

engineer by Wilson Joseph from the Noun Project

panel by gira Park from the Noun Project

Approve by Aha-Soft from the Noun Project

Database sharing by YuguDesign from the Noun Project

ram by Andrea Rizzato from the Noun Project

Database Search by Nimal Raj from the Noun Project

Cloud Analytics by Kevin Augustine LO from the Noun Project

Broken Computer by Dan Hetteix from the Noun Project

data search by Gregor Črešnar from the Noun Project

Server by Creative Stall from the Noun Project

Network by Creative Stall from the Noun Project

transformation by Felipe Perucho from the Noun Project

analytics by Roman Kovbasyuk from the Noun Project

Server by Designify.me from the Noun Project

MERCI

http://infinispan.org

http://blog.infinispan.org

@infinispan