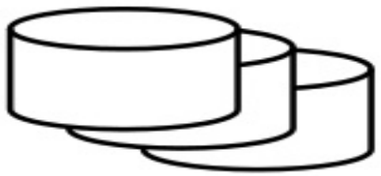
Lessons Learned

Optimizing NoSQL for Apache Spark

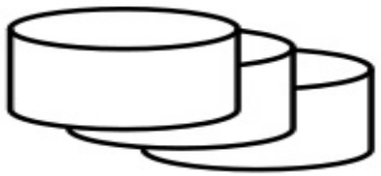
John Musser @johnmusser / Basho @basho

Spark Summit Europe, 2016

NoSQL



NoSQL



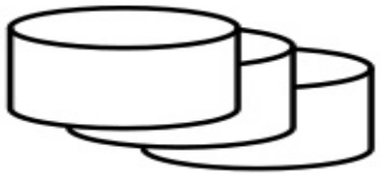
Key-Value

Document

Columnar

Graph

NoSQL



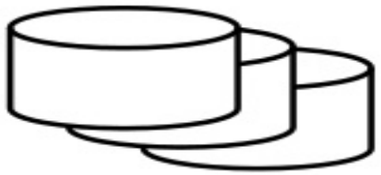
Key-Value

Document

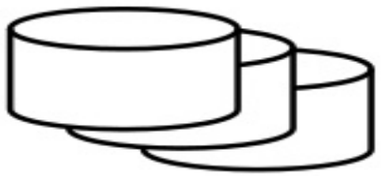
Columnar

Graph

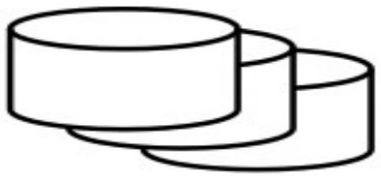
NoSQL + APACHE Spark™



NoSQL + APACHE Spark™ = ?

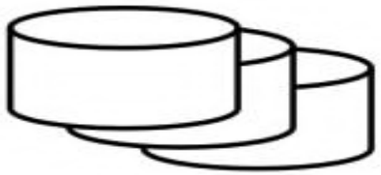


NoSQL + APACHE Spark™ =



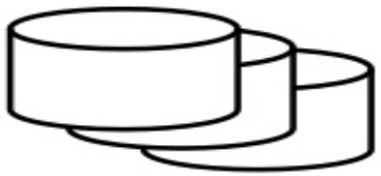
How do we turn this...

NoSQL + APACHE Spark™ =



into this?

NoSQL + APACHE Spark™ =



We built this.

We built this.
Here are our lessons...

Parallelize

Map smart

Optimize all the levels

Be flexible

Simplify

BACKGROUND

Riak

- Distributed, key-value NoSQL database
- Known for scalability, reliability, ops simplicity
- Launched 2009, used by 1/3 of Fortune 50
- Open source (Apache), on GitHub

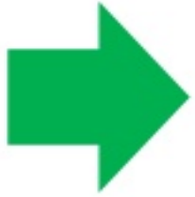
https://github.com/basho/riak/

- Enterprise Edition, see multi-cluster replication

We started to see in our customer base...

More demand around:
time series,
IoT,
metrics

More demand around:
time series,
IoT,
metrics



Riak TS

Spark-Riak
Connector

Riak KV

Key-Value data

User data
Session data
Profile data
Log data

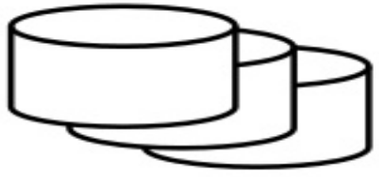
Riak TS

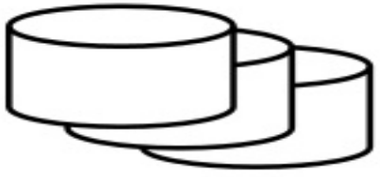
Released in 2016

Time Series data

IoT / Device data
Metrics data
Event data
Streaming data

Riak Core





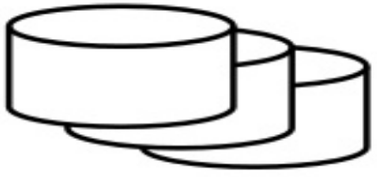
Riak TS

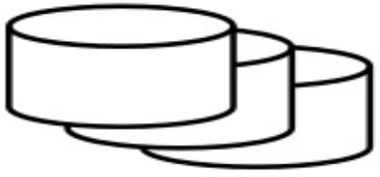
Released in 2016

Time Series data

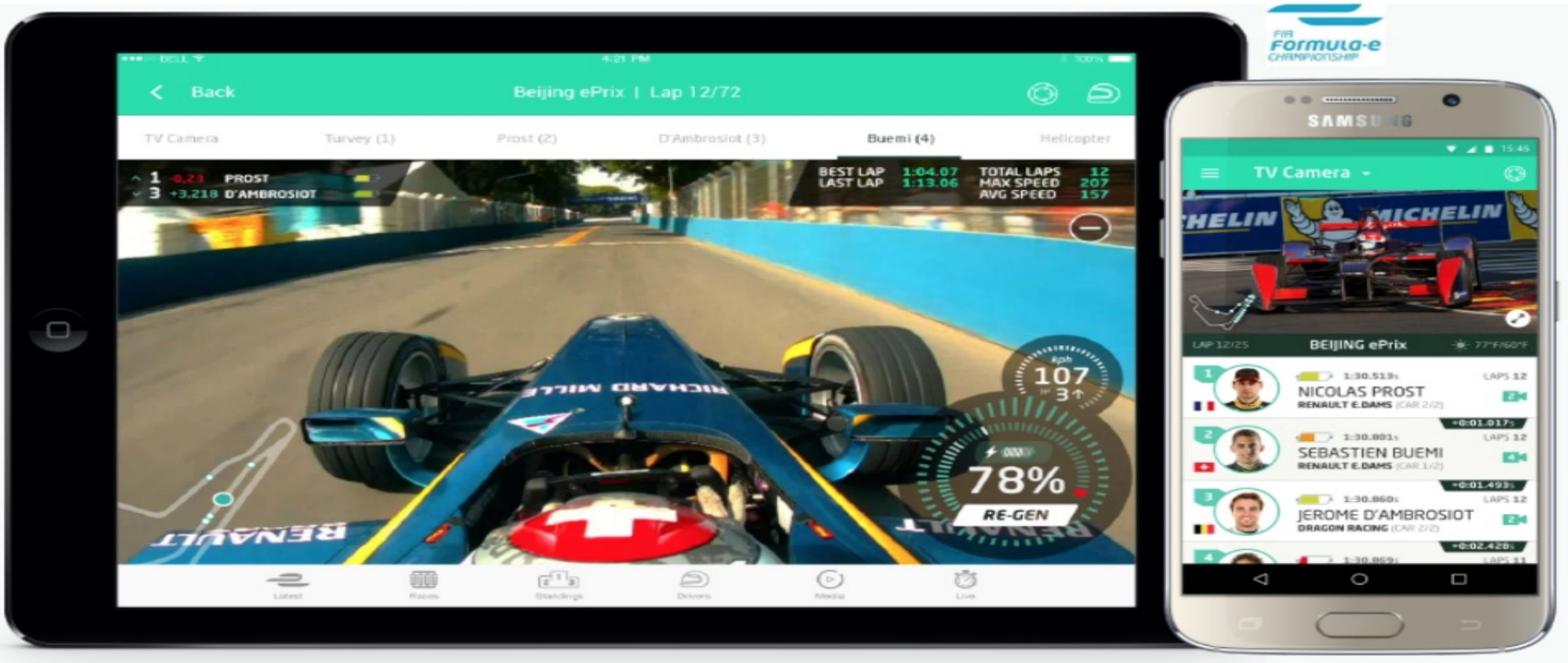
DDL for tables (with data types)
SQL subset (with filters and aggregations)
Fast bulk writes
Efficient reads via “time slice” queries

Riak Core






FIA
Formula·e
CHAMPIONSHIP
Intellicore
Intellicore Sports Data Platform
• 1GB telemetry per driver
• 400 packets/second
• 1.2M packets/race
• Platform setup for 40,000 TPS


Back
Beijing ePrix | Lap 12/72
TV Camera
Turvey (1)
Prost (2)
D'Ambrosiot (3)
Buemi (4)
Helicopter
1 -0,23 PROST
3 +3,218 D'AMBROSIOT
BEST LAP 1:04.07
LAST LAP 1:13.06
TOTAL LAPS 12
MAX SPEED 207
AVG SPEED 157
107
78%
RE-GEN
Latest
Races
Standings
Drivers
Media
Live
TV Camera
LAP 12/25
BEIJING ePrix
NICOLAS PROST
RENAULT E.DAMS
SEBASTIEN BUEMI
RENAULT E.DAMS
JEROME D'AMBROSIOT
DRAGON RACING

BACKGROUND

Spark-Riak Connector

- Version 1.0: published Sept. 2015
- Current version: 1.6, published Sept. 2016
- Scala / JVM based
- Support for Java, Scala, Python
- Supports Spark 1.6.x
- Open source (Apache), on GitHub

https://github.com/basho/spark-riak-connector/

READ

WRITE

STREAM

Enable SQL analytics over Riak

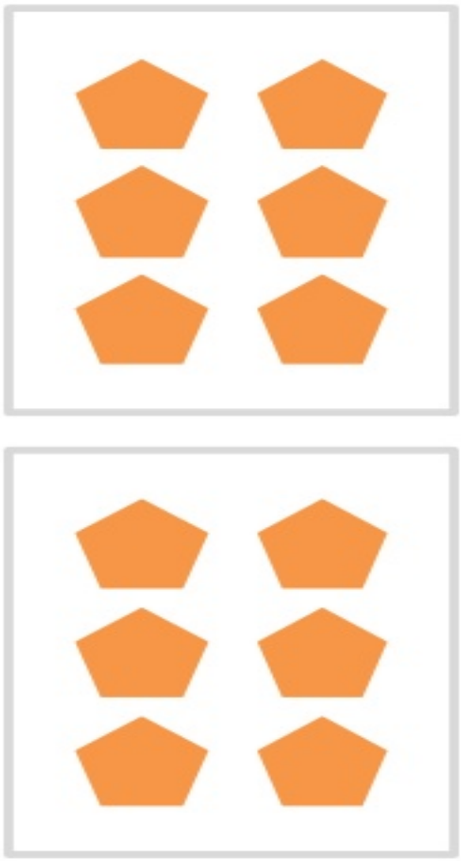
Use Riak to store results generated by Spark

Use Riak to store streaming data

(this in turn uses learnings from other connectors we've built...)

LESSON #1

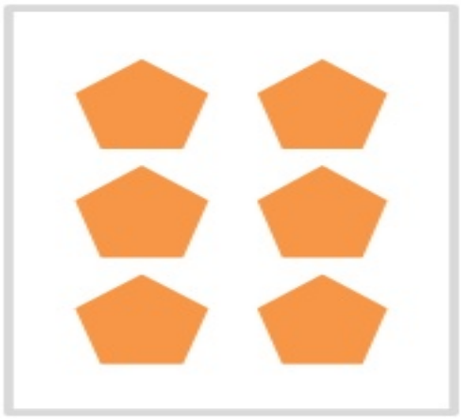
Parallelize whenever possible

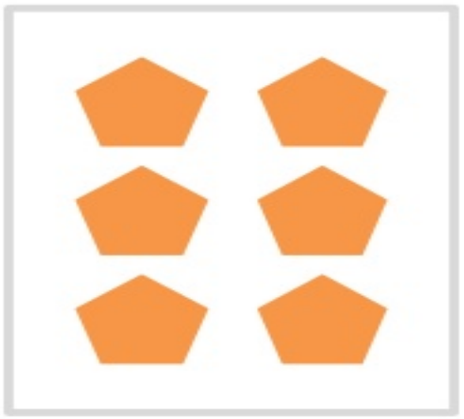


riak

APACHE Spark™

?





riak

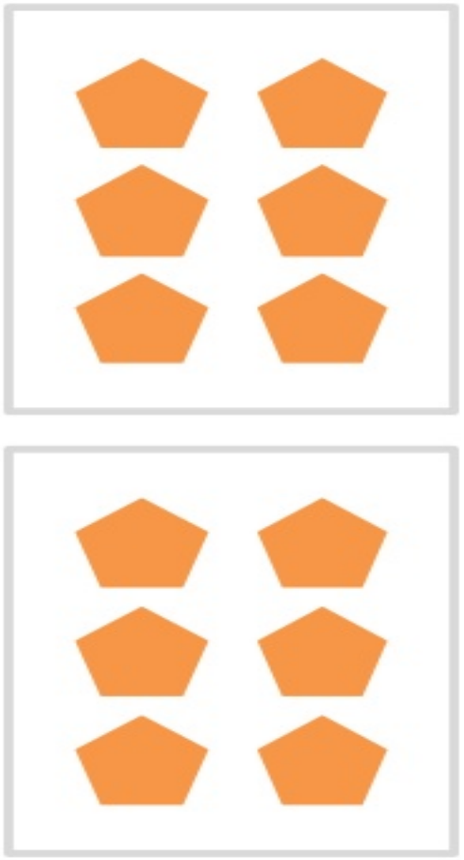
APACHE Spark™

How to move lots of data quickly and efficiently?

riak

APACHE Spark™

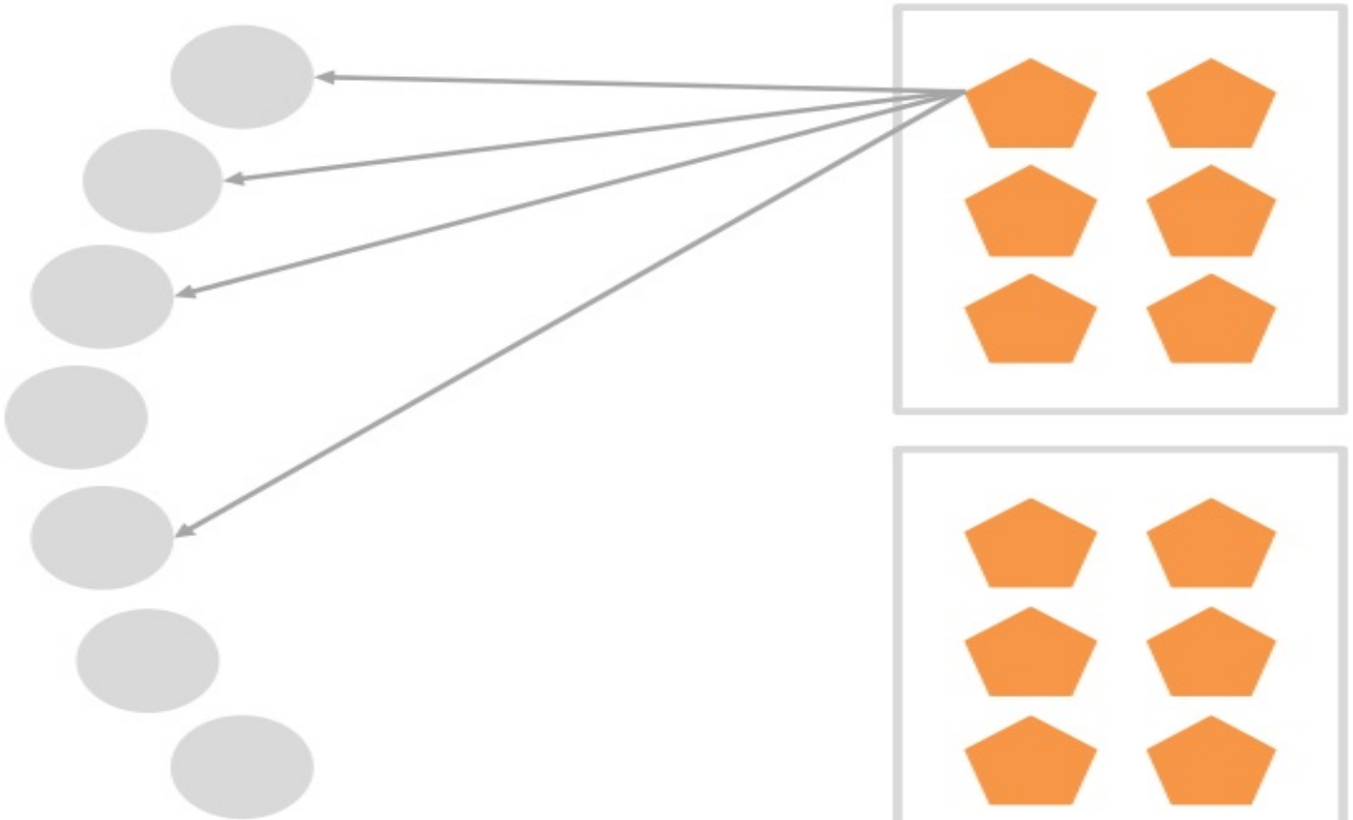
Using Direct Key-based GETs



riak

APACHE Spark™

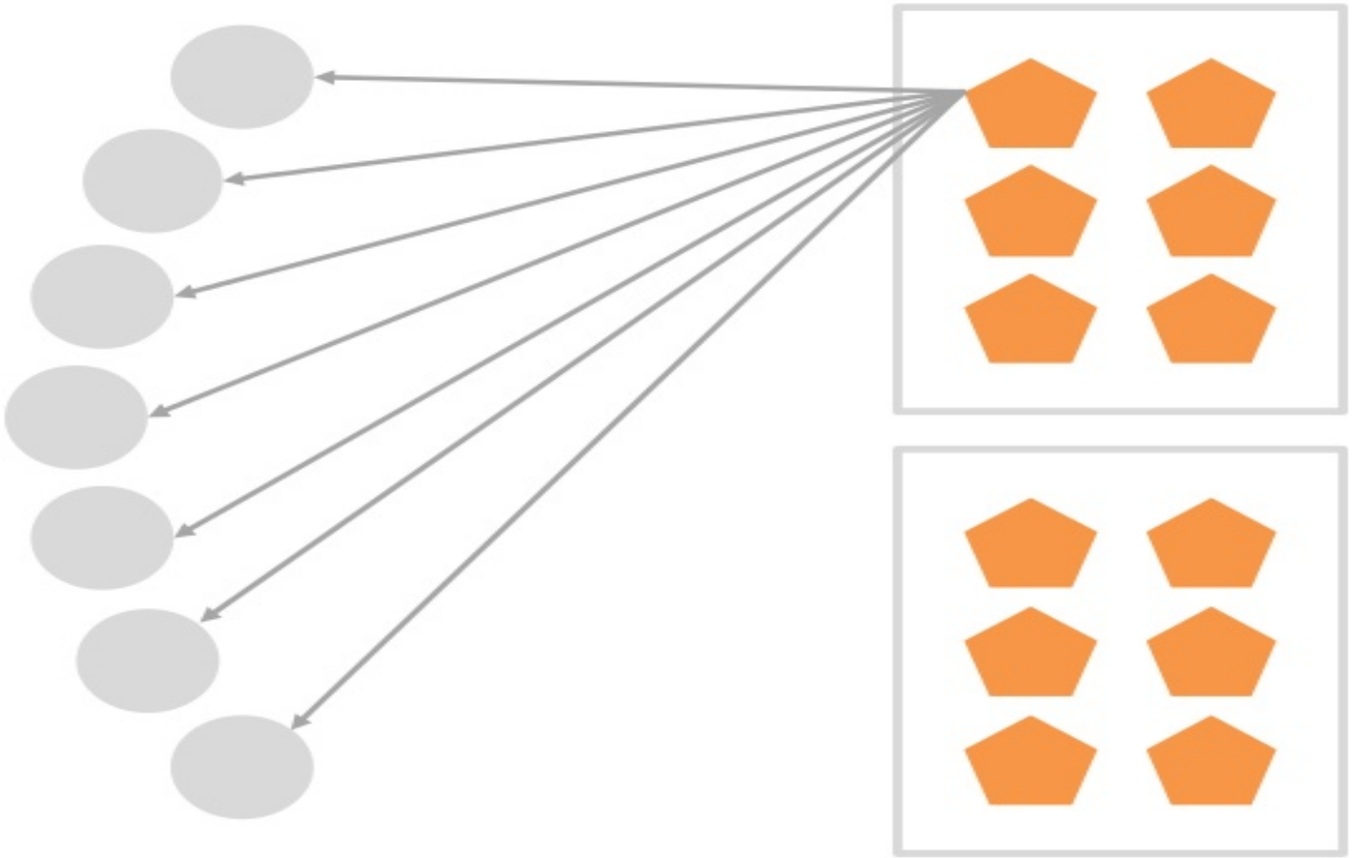
Using Direct Key-based GETs



riak

APACHE Spark™

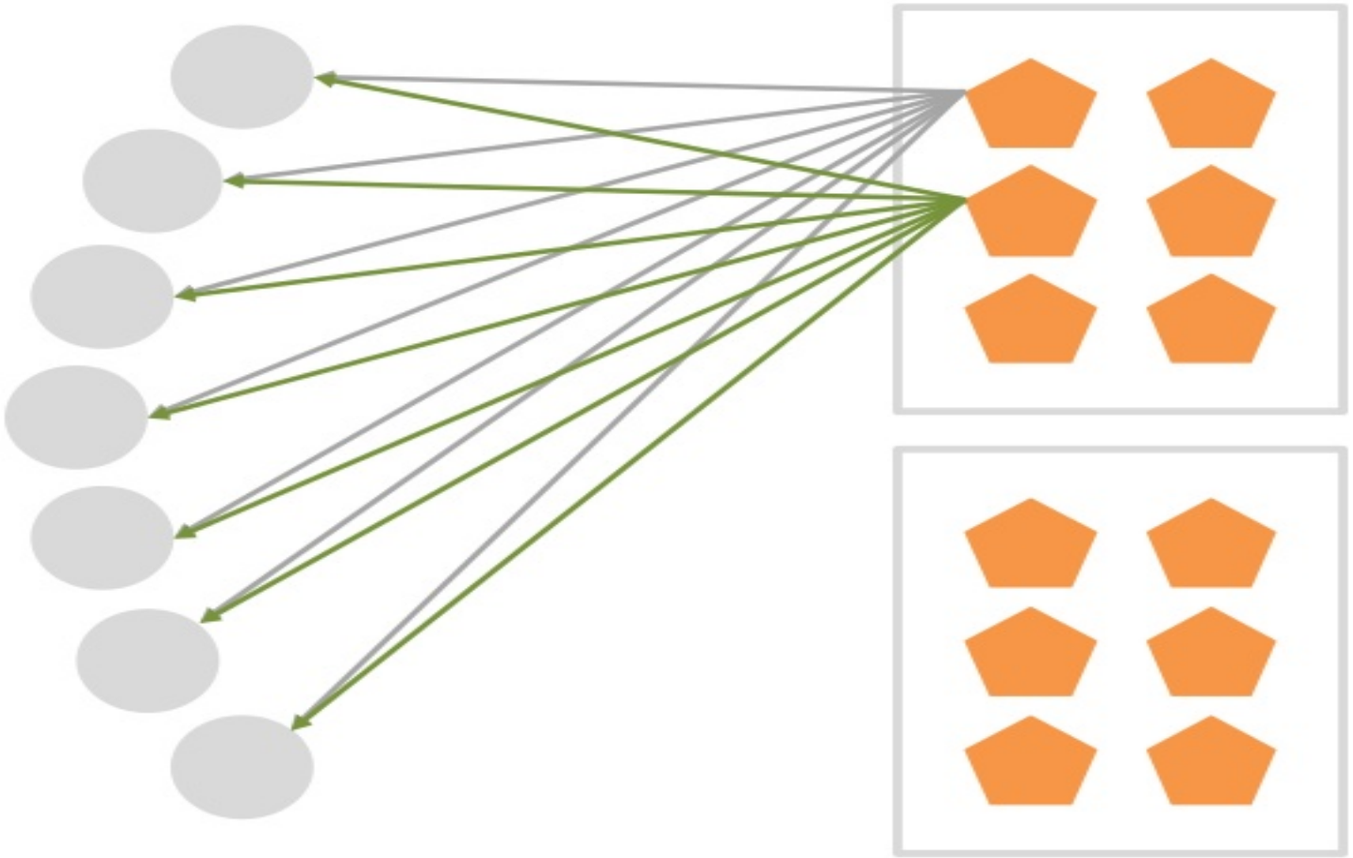
Using Direct Key-based GETs



riak

APACHE Spark™

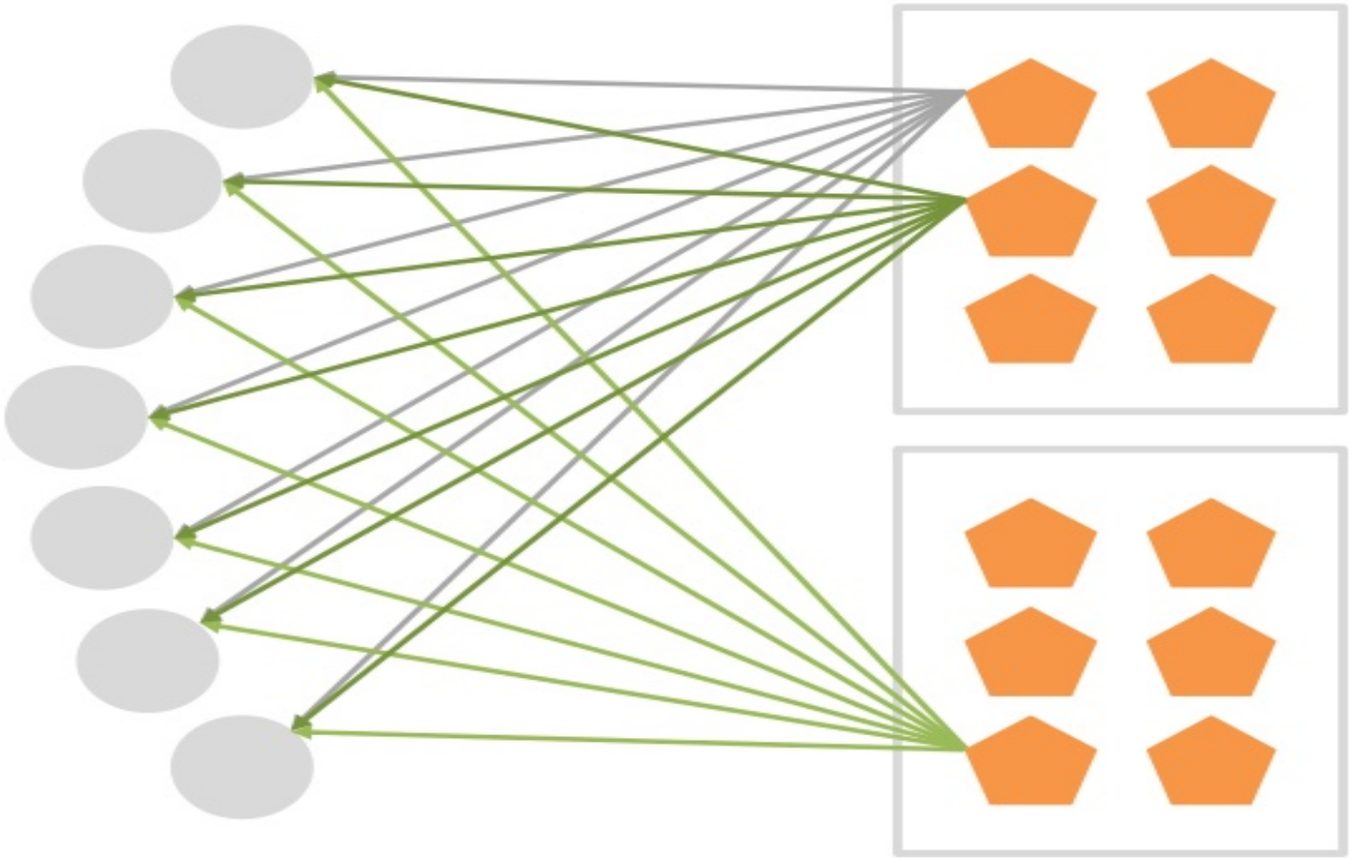
Using Direct Key-based GETs



riak

APACHE Spark™

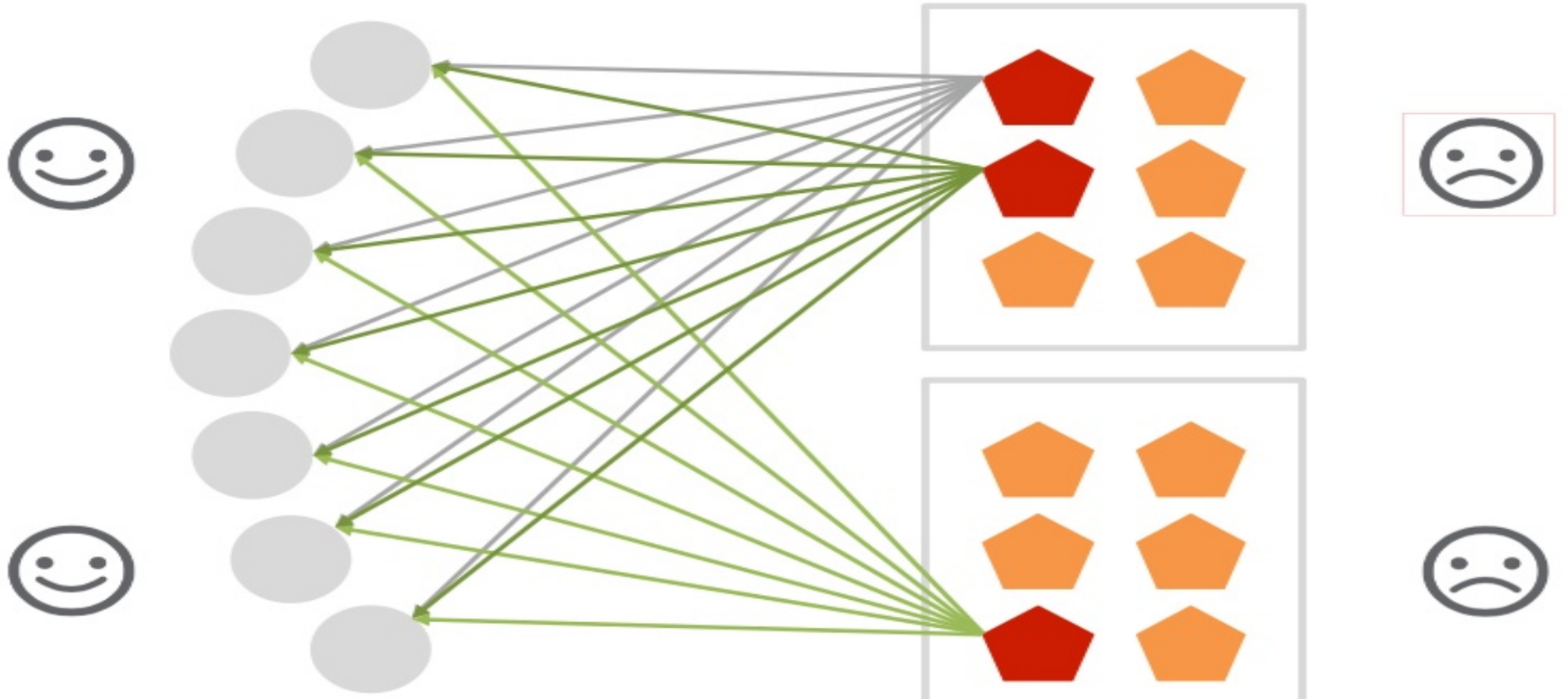
Using Direct Key-based GETs



riak

APACHE Spark™

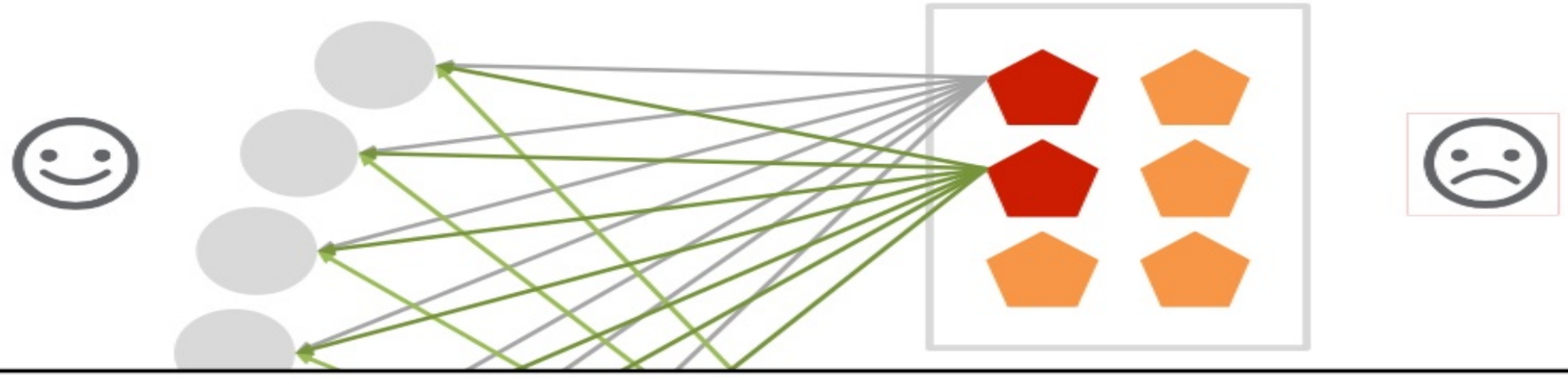
Using Direct Key-based GETs



riak

APACHE Spark™

Using Direct Key-based GETs



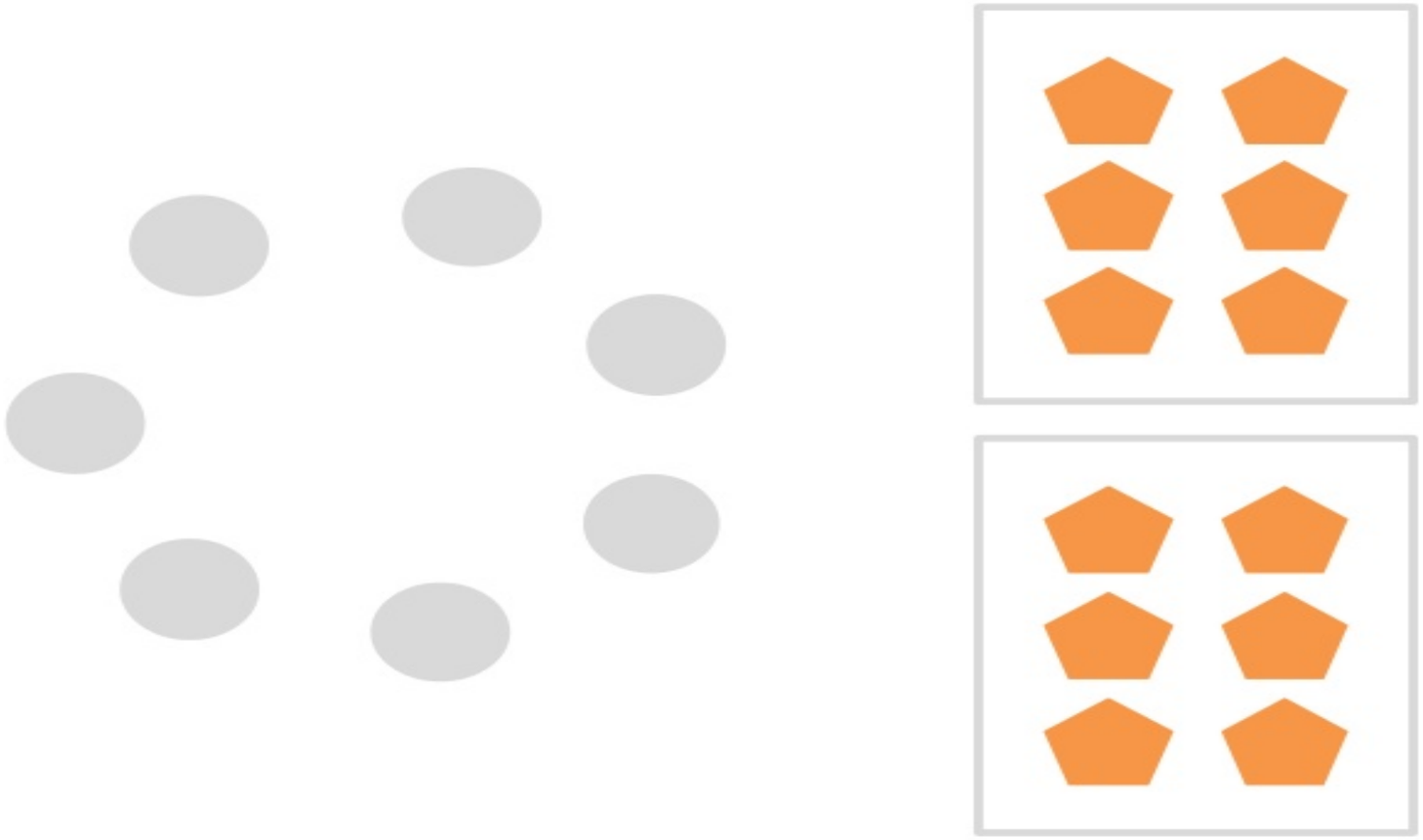
Lesson 1(a):

Too many Gets make Spark unhappy

riak

APACHE Spark

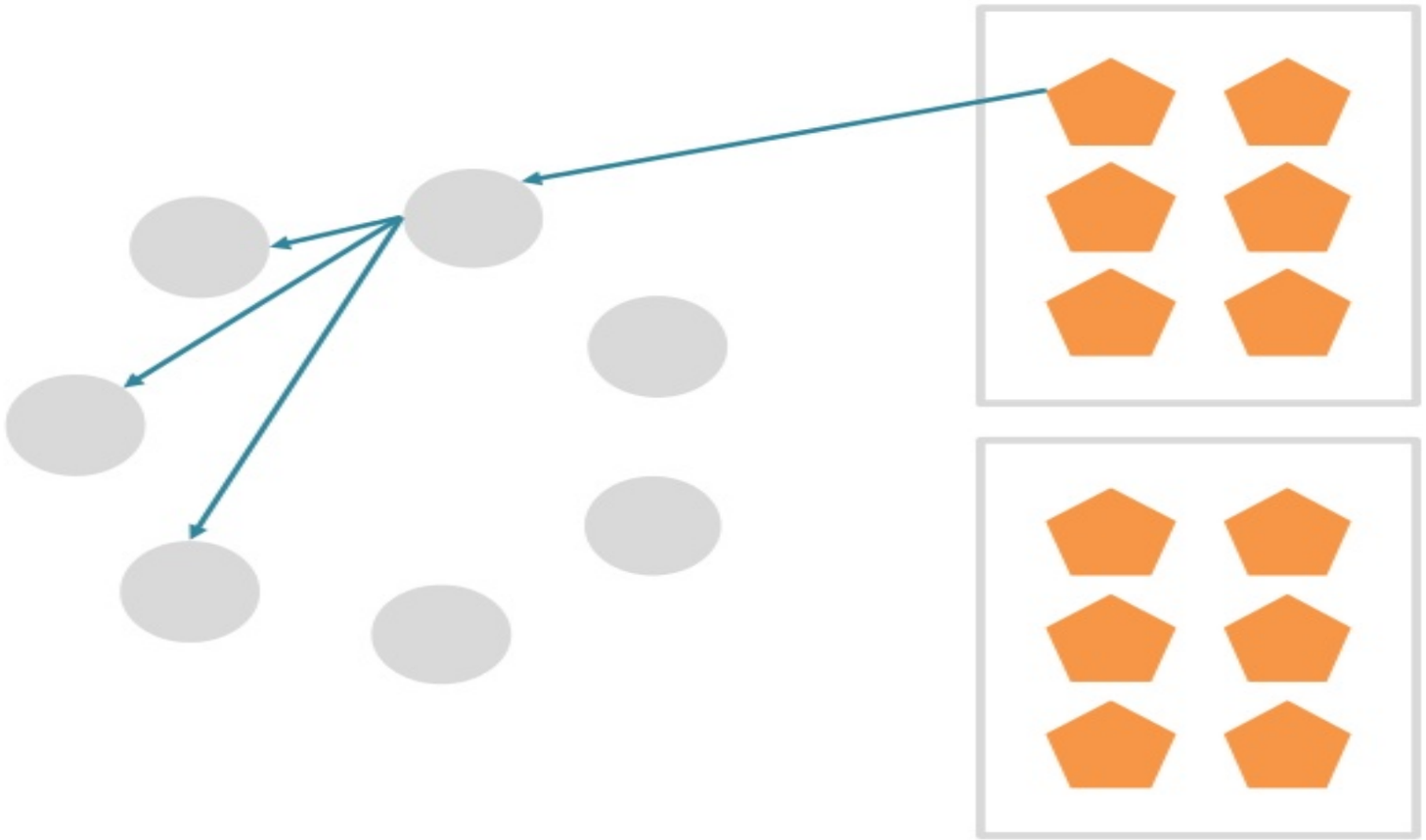
Using Secondary Index (2i)



riak

APACHE Spark™

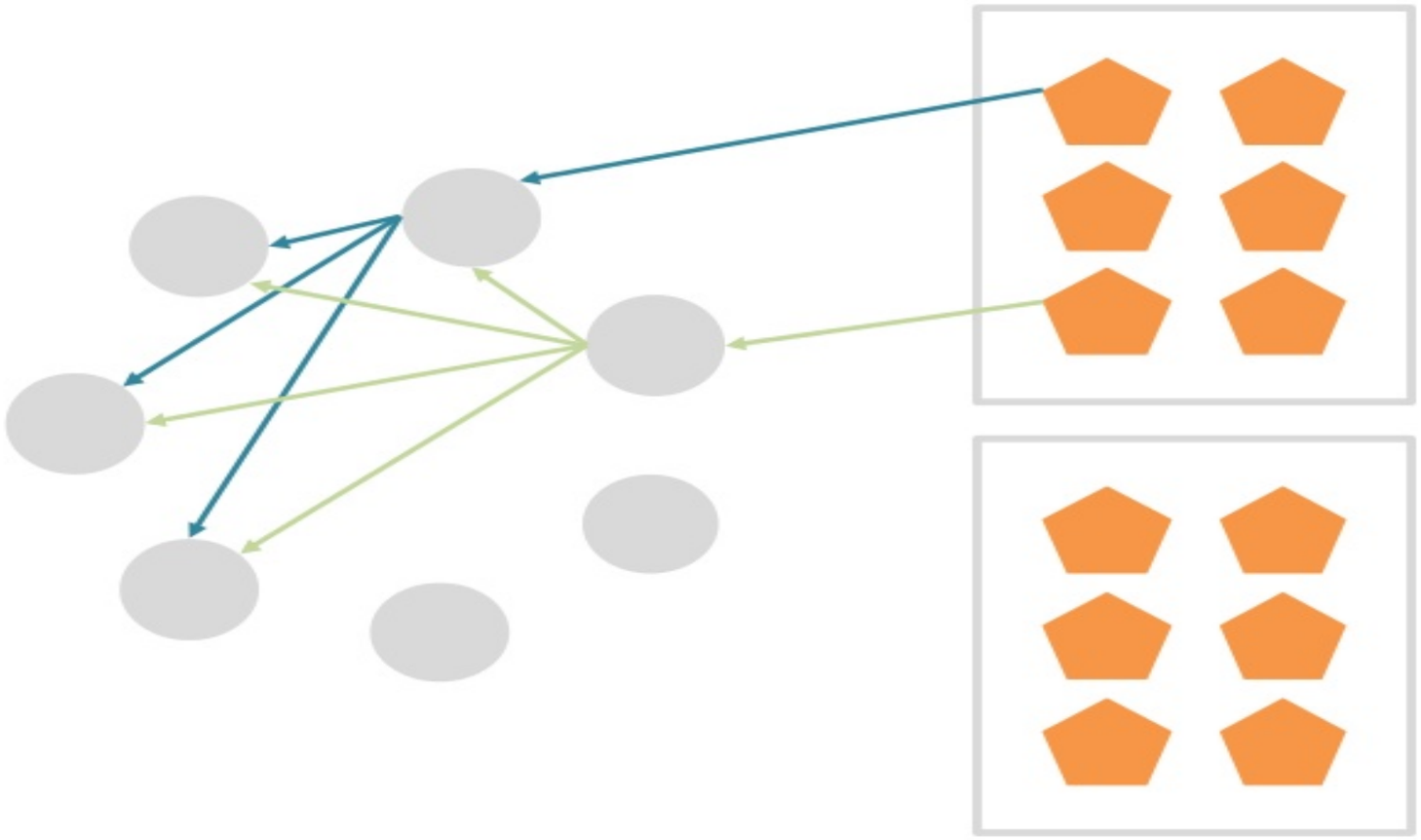
Using Secondary Index (2i)



riak

APACHE Spark™

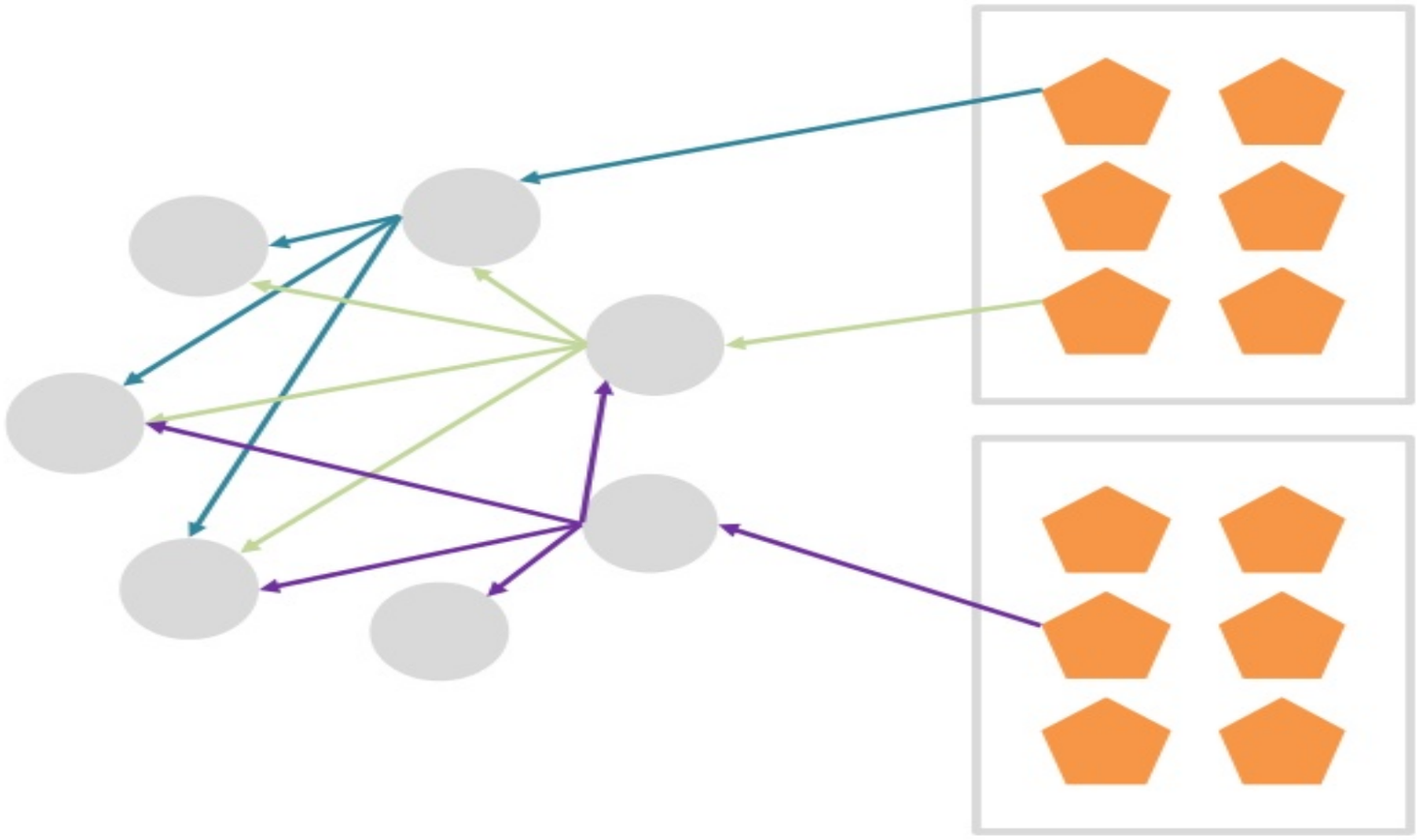
Using Secondary Index (2i)



riak

APACHE Spark™

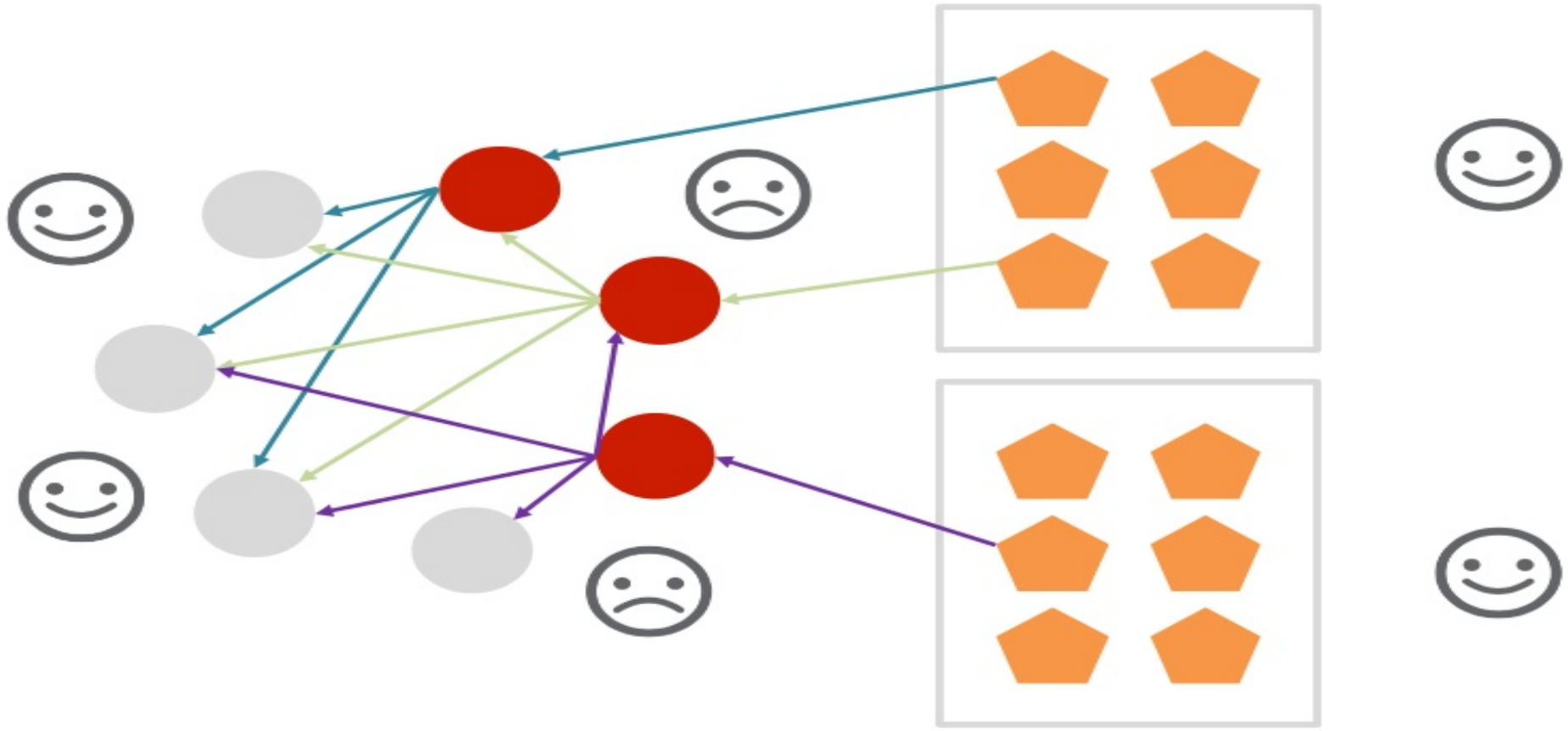
Using Secondary Index (2i)



riak

APACHE Spark™

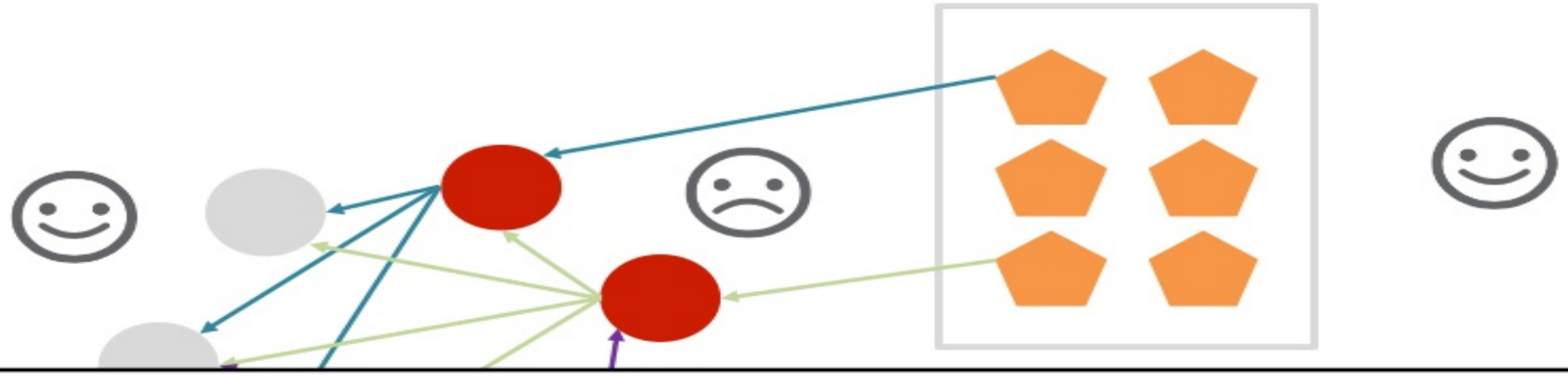
Using Secondary Index (2i)



riak

APACHE Spark

Using Secondary Index (2i)



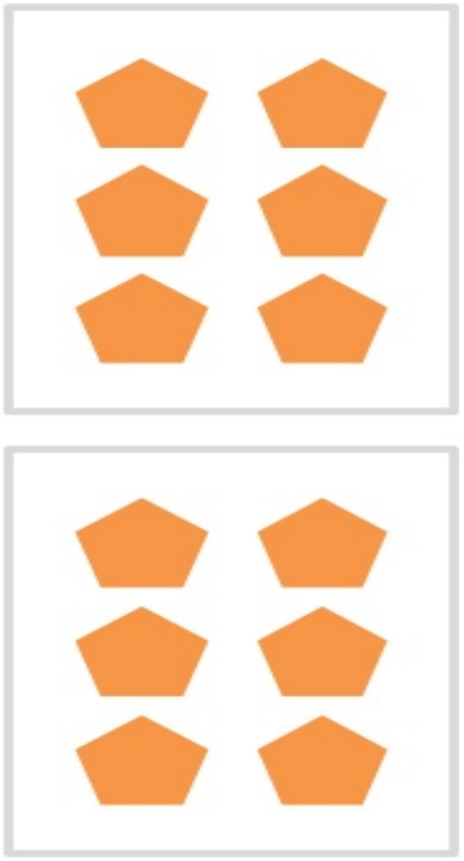
Lesson 1(b):

Too many 2i queries make Riak unhappy

riak

APACHE Spark™

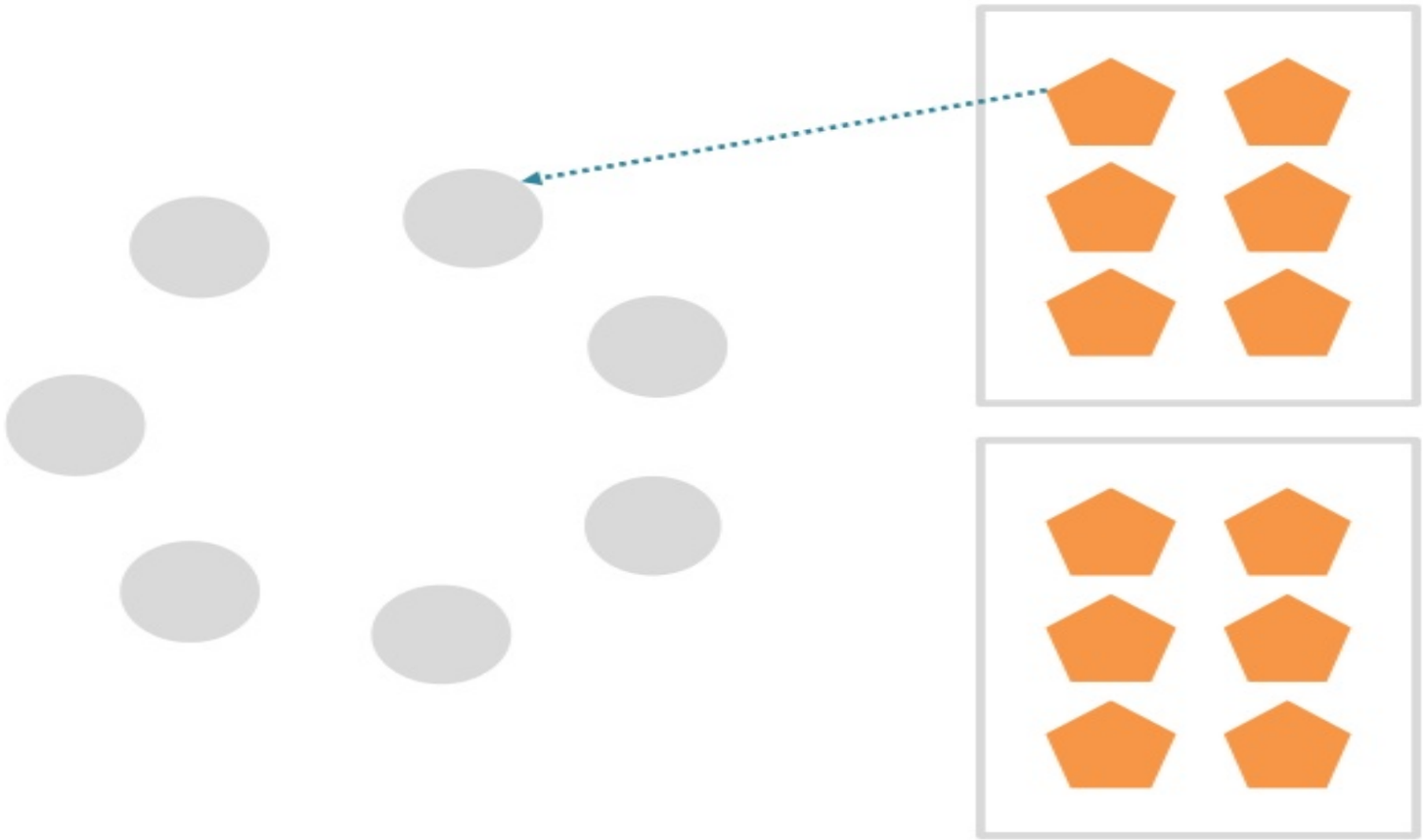
Coverage Plan + Parallel Extract



riak

APACHE Spark™

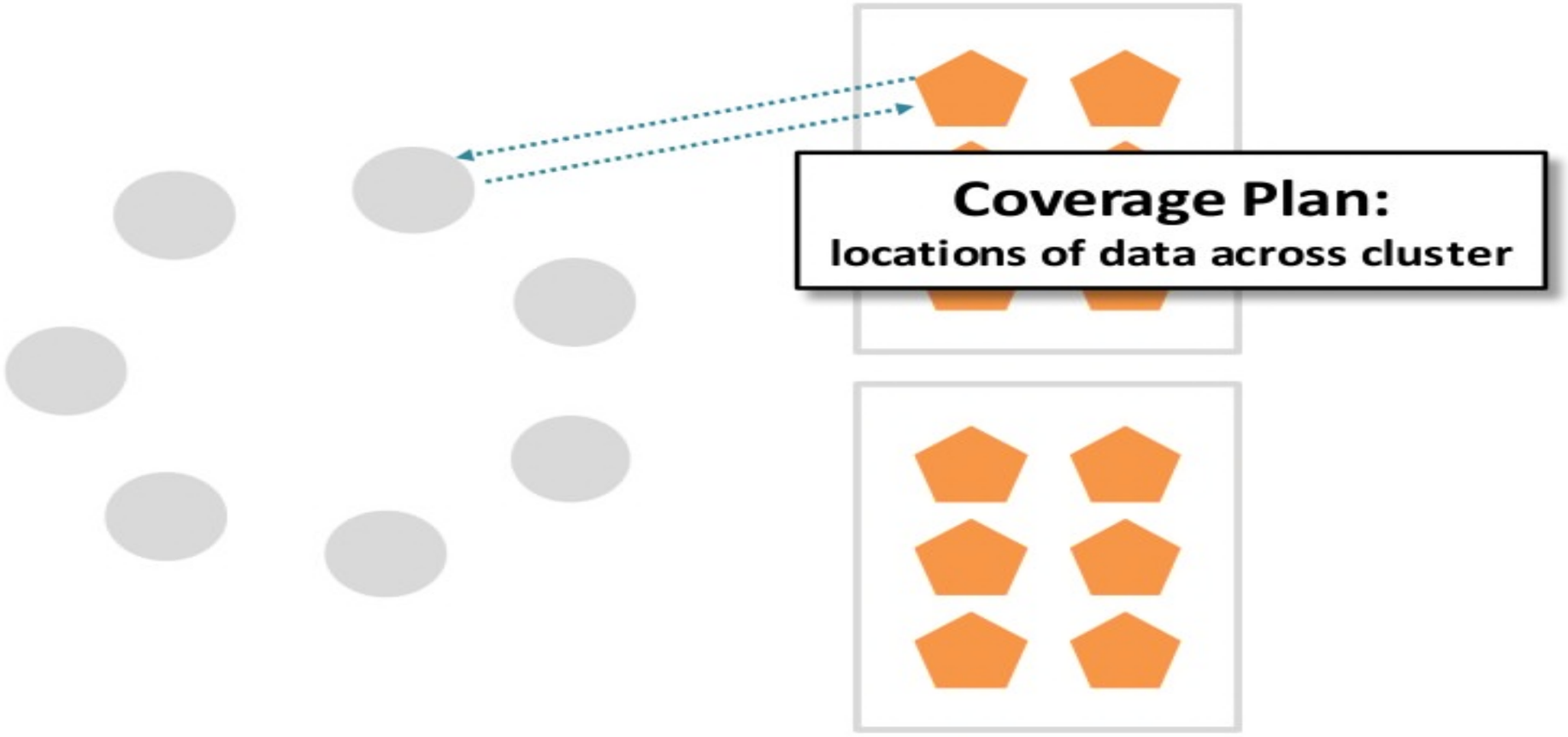
Coverage Plan + Parallel Extract



riak

APACHE Spark™

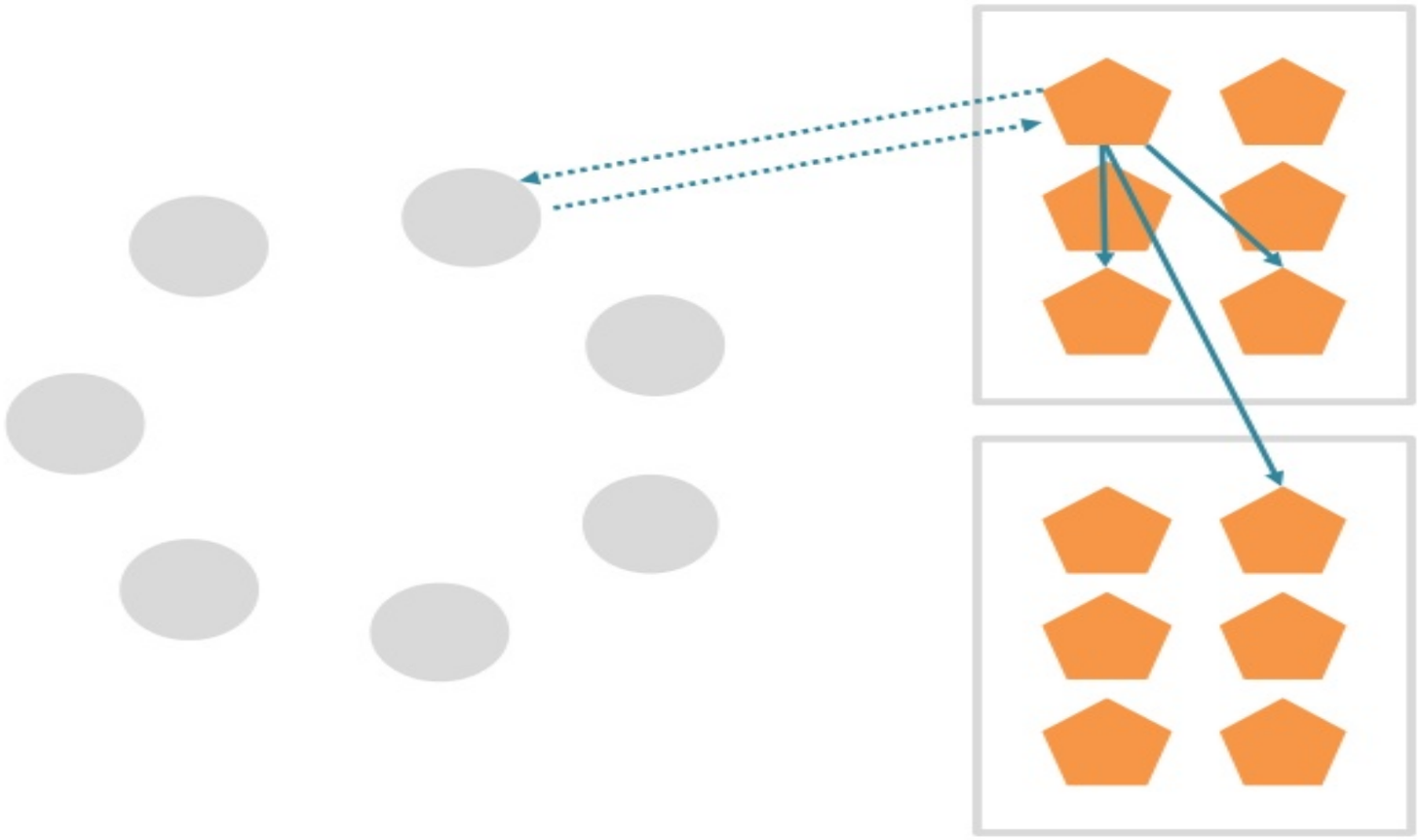
Coverage Plan + Parallel Extract



riak

APACHE Spark

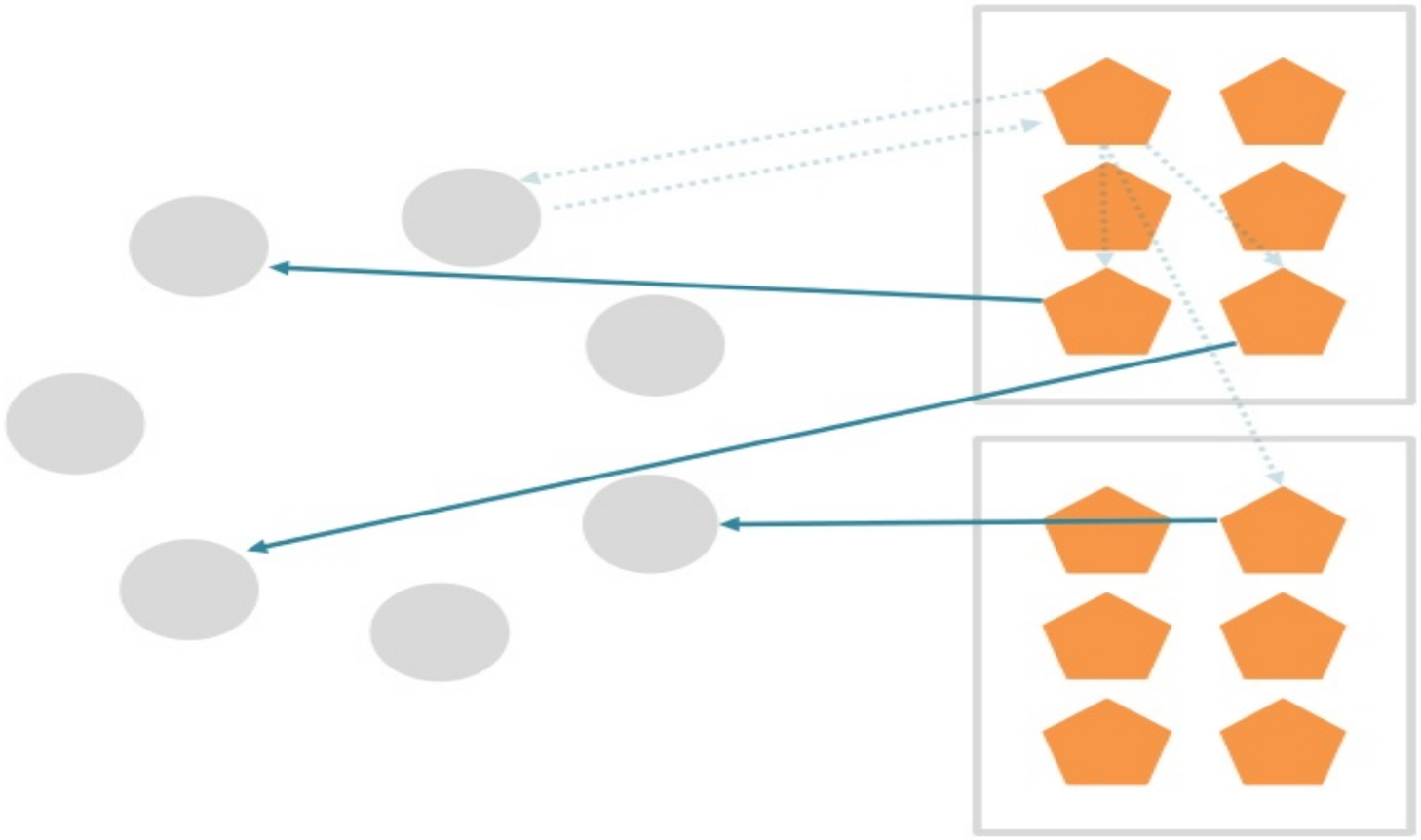
Coverage Plan + Parallel Extract



riak

APACHE Spark™

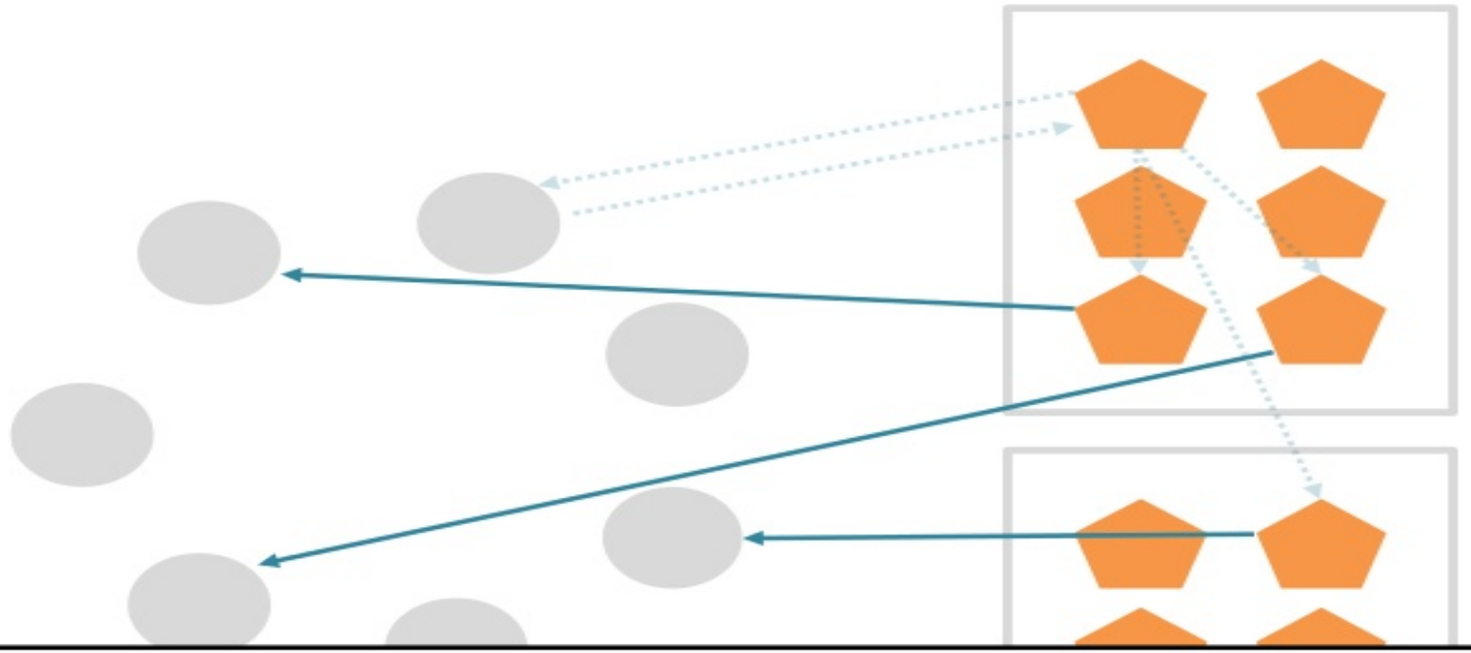
Coverage Plan + Parallel Extract



riak

APACHE Spark™

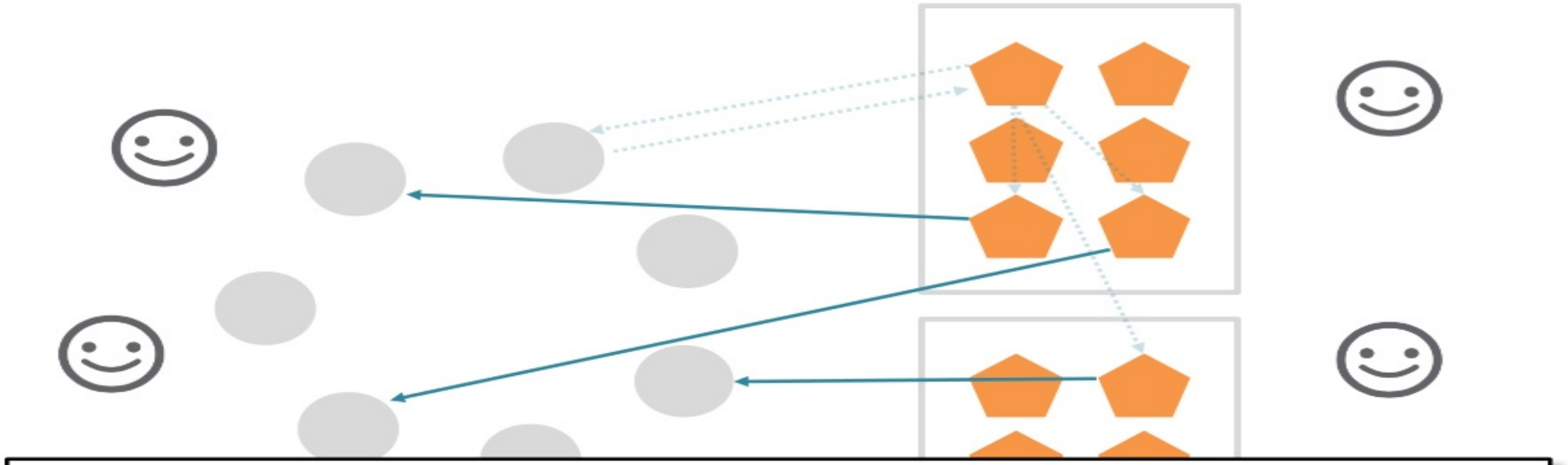
Coverage Plan + Parallel Extract



Parallelization

Coverage Plan + Parallel Extract

Coverage Plan + Parallel Extract



Parallelization

Coverage Plan + Parallel Extract
= everybody's happy

LESSON #2

Be smart about data mapping

Key-Value
& Time Series
Data

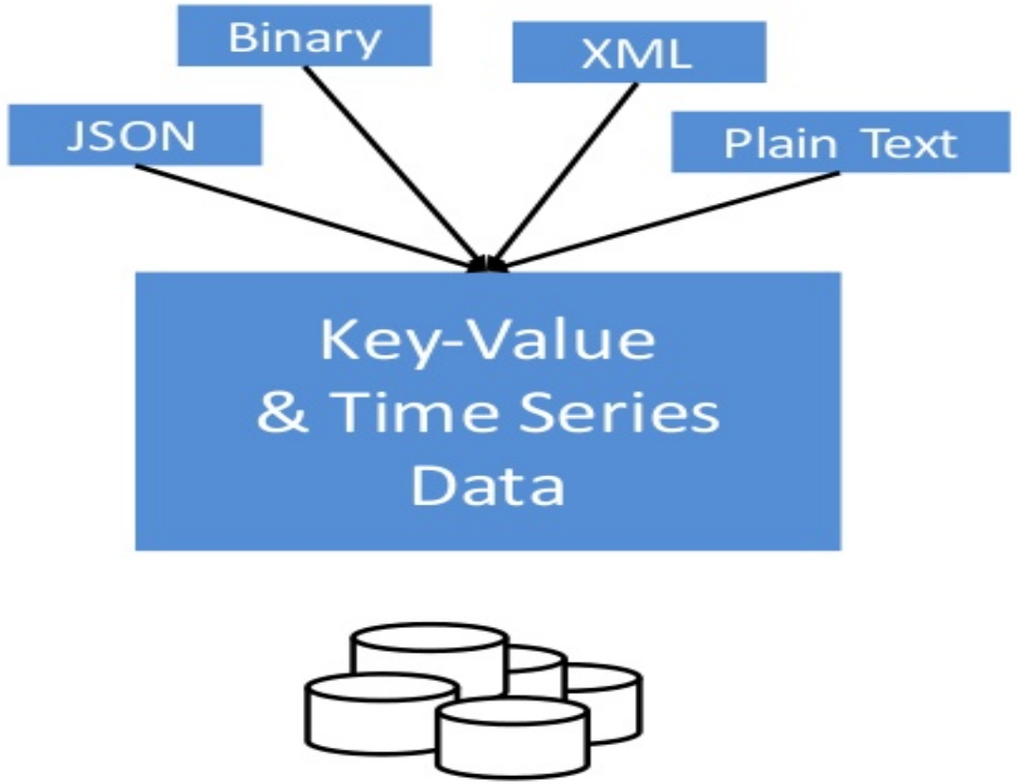
riak

RDDs

DataFrames

DataSets

APACHE Spark™



RDDs

DataFrames

DataSets

riak

APACHE Spark

Key-Value
& Time Series
Data

riak

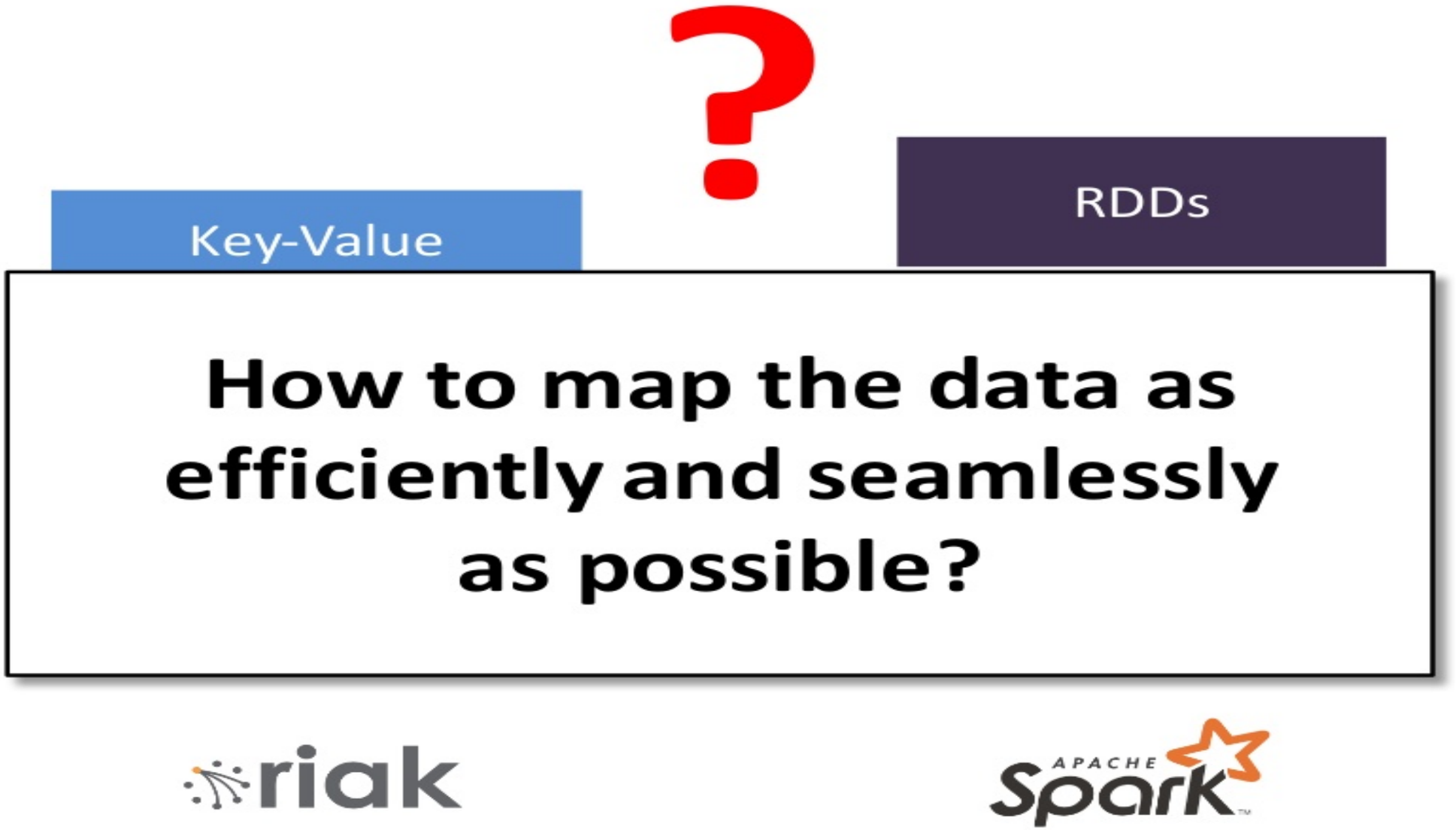
?

RDDs

DataFrames

DataSets

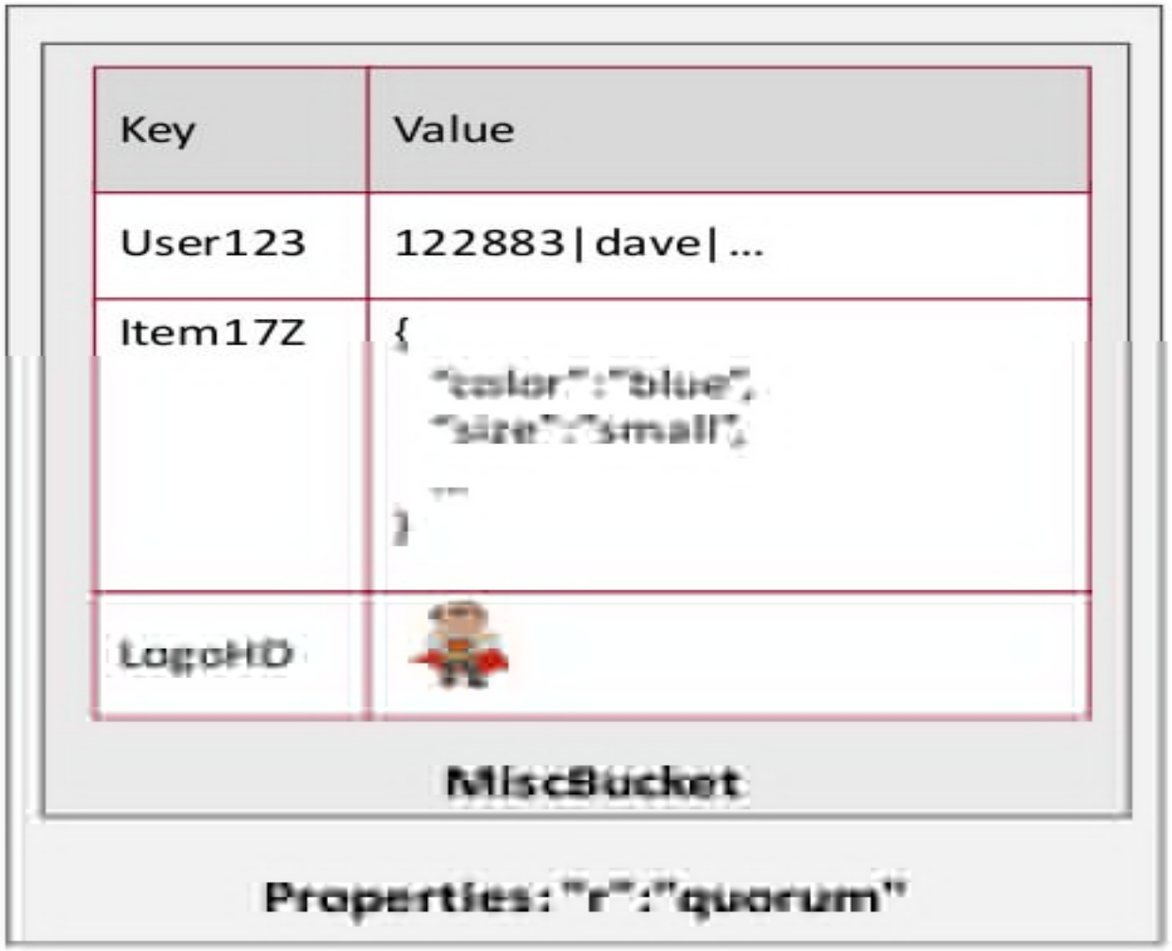
APACHE Spark™


?
Key-Value
RDDs
How to map the data as efficiently and seamlessly as possible?
riak
APACHE
Spark

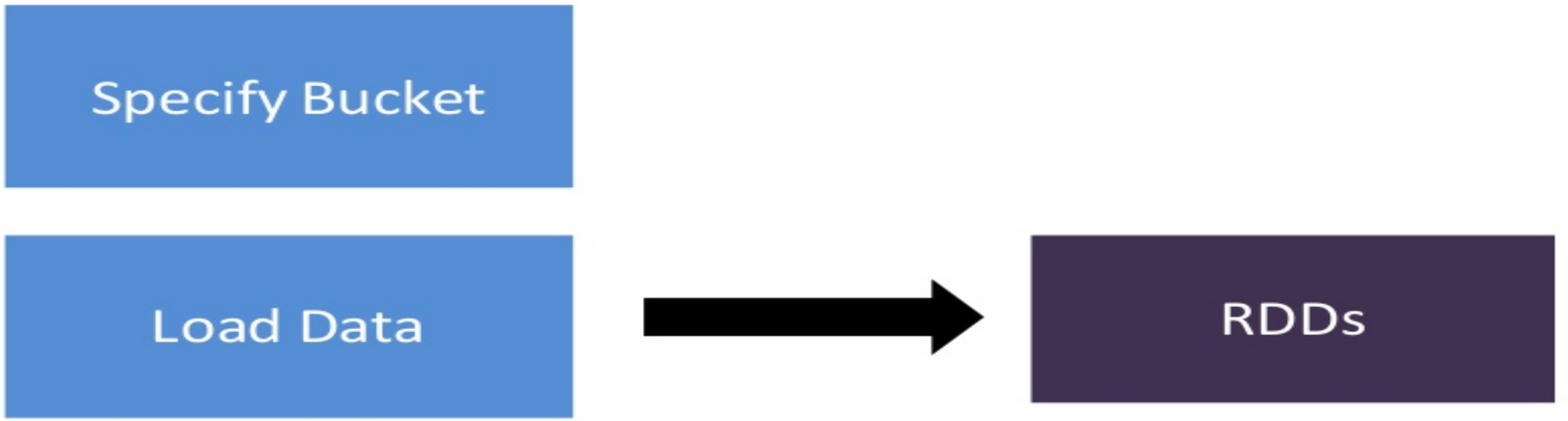
BACKGROUND

How Riak Stores Data

- Keys
- Values
- Buckets
- Bucket Types



Key-Value Data



Code: Key/Value Query

Specify Bucket

```
val kv_bucket = new Namespace(”MiscBucket")
```

Load Data

```
val riakRdd =
    sc.riakBucket[String](kv_bucket).queryAll()
```

Code: Key/Value Query

Query by Keys

```
val rdd =
    sc.riakBucket[String](kv_bucket_name)
    .queryBucketKeys("Alice", "Bob", "Charlie")
```

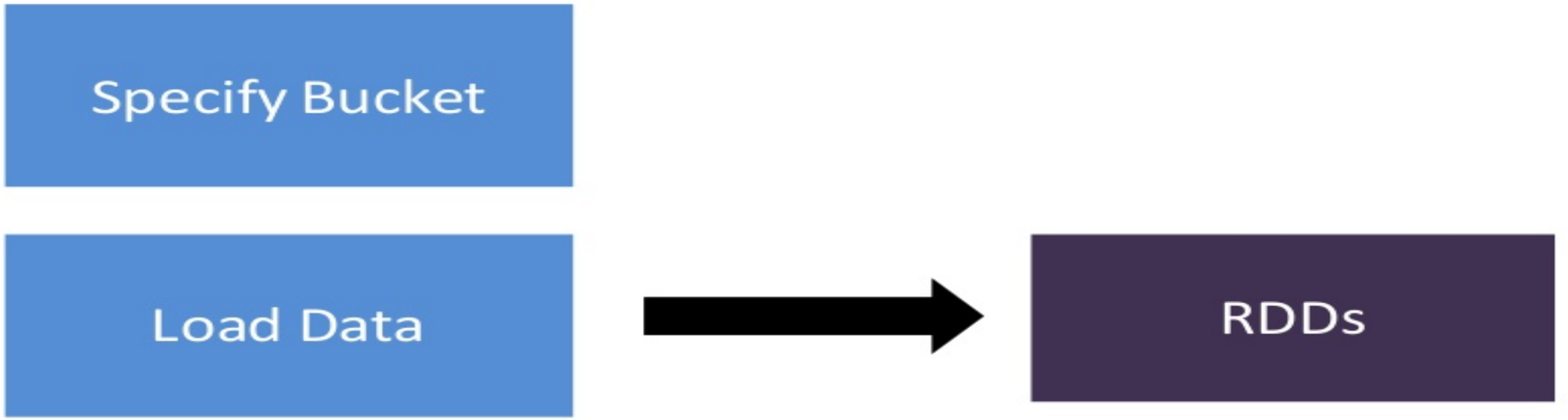
Query by 2i Range

```
val rdd =
    sc.riakBucket[String](kv_bucket_name)
    .query2iRange("myIndex", 1L, 5000L)
```

Query by 2i Strings

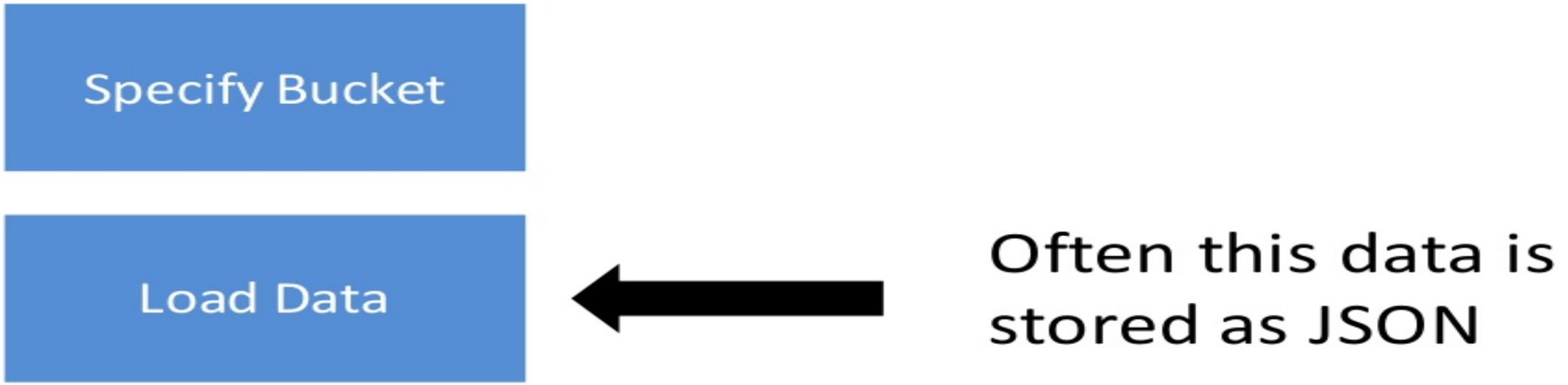
```
val rdd =
    sc.riakBucket[String](kv_bucket_name)
    .query2iKeys("dailyDataIndx", ”Jan", ”Feb”)
```

Key-Value Data

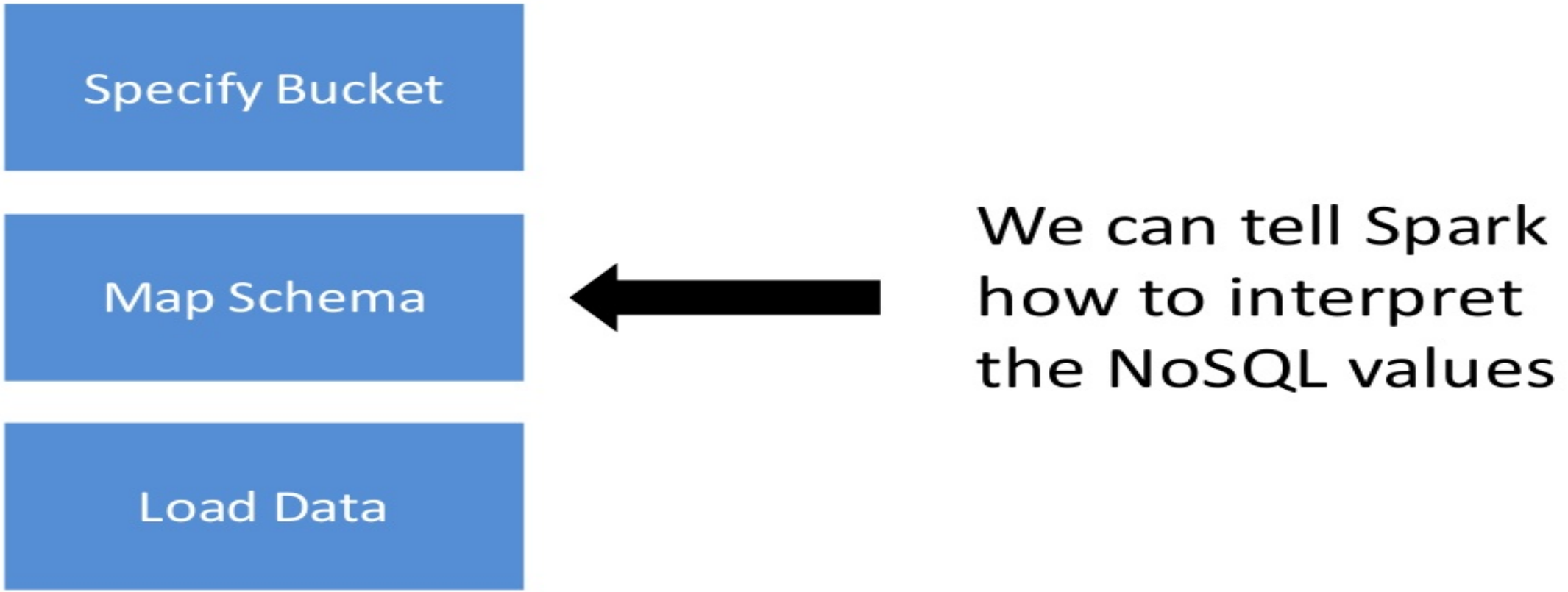


Which is “fine”, but, the data is still a bit opaque…

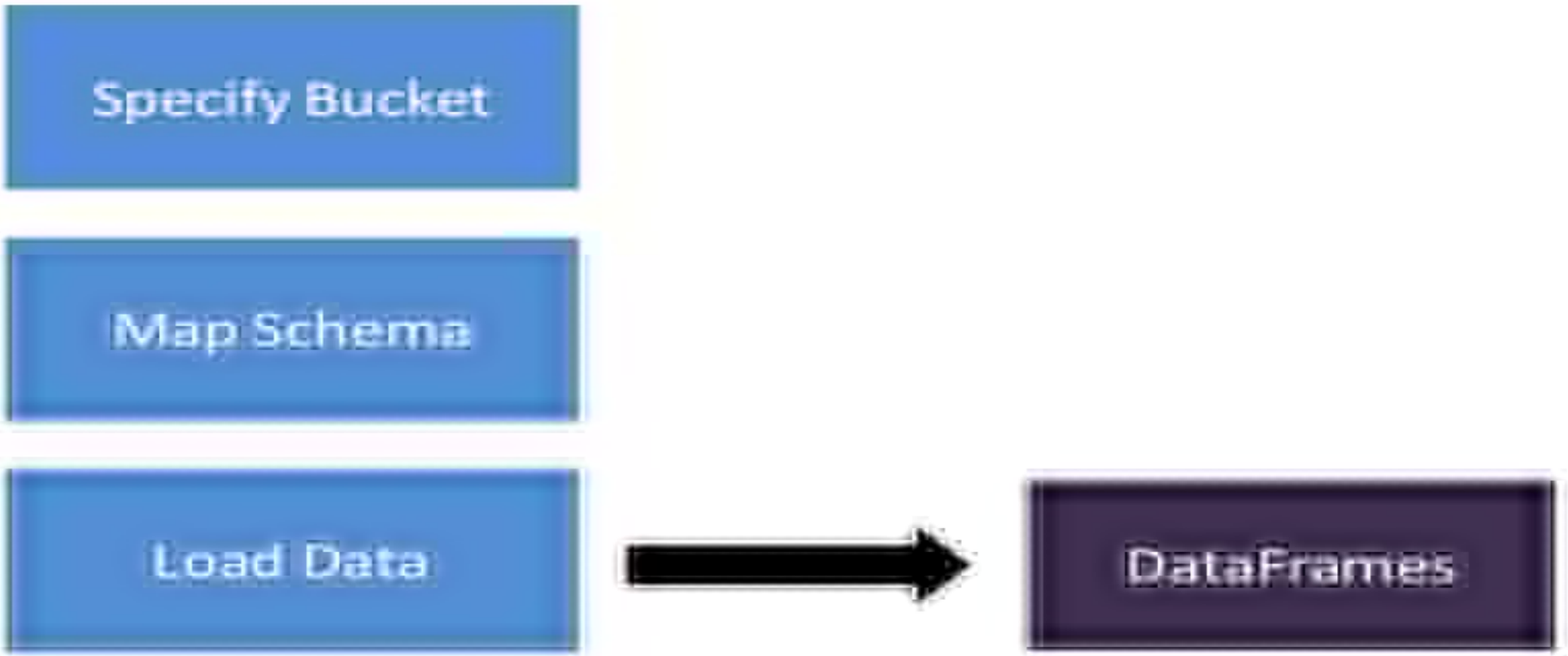
Key-Value Data



Key-Value Data



Key-Value Data



Key-Value Data

Specify Bucket

Map Schema

Now we have full-fledged DataFrames

Code

Specify Bucket

```
val kv_bucket = new Namespace(”MiscBucket")
```

Map Schema

```
case class UserData(
        user_id: String, name: String, age: Int)
```

Load Data

```
val riakRdd =
    sc.riakBucket[UserData](kv_bucket).queryAll()

val df = riakRdd.toDF()
```

Key-Value Data

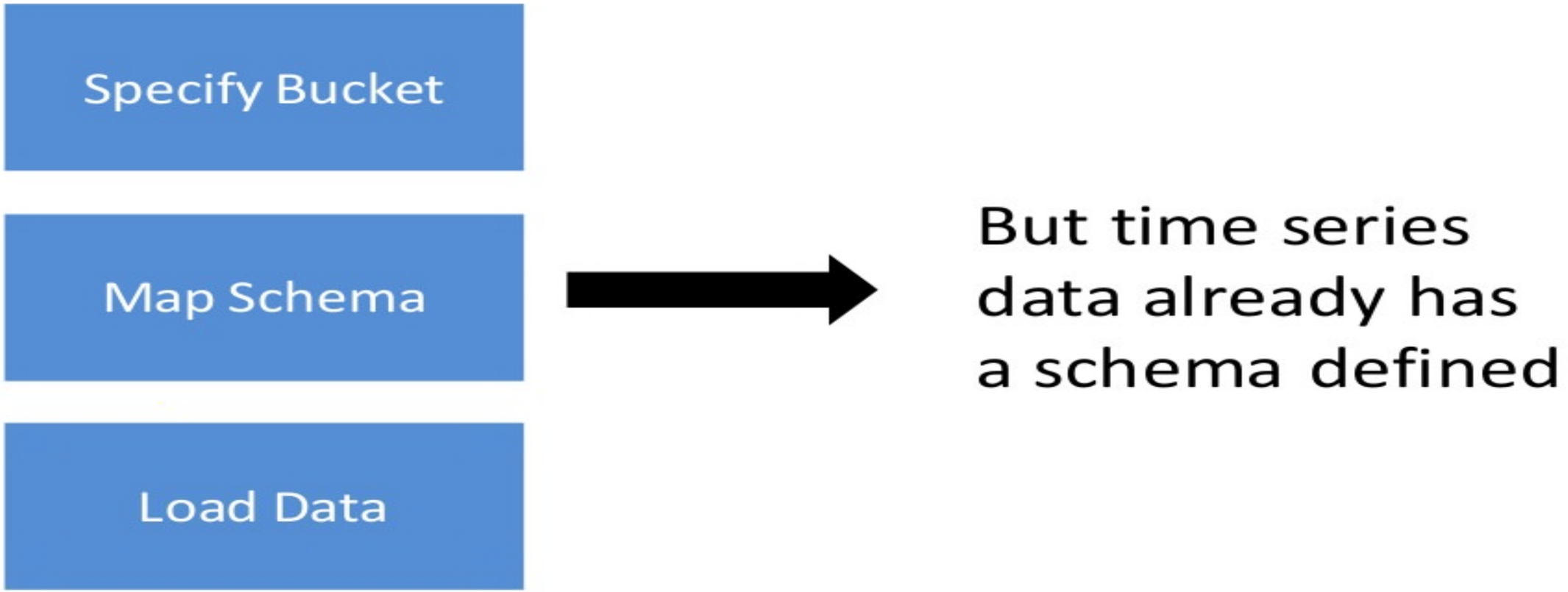
Specify Bucket

Map Schema

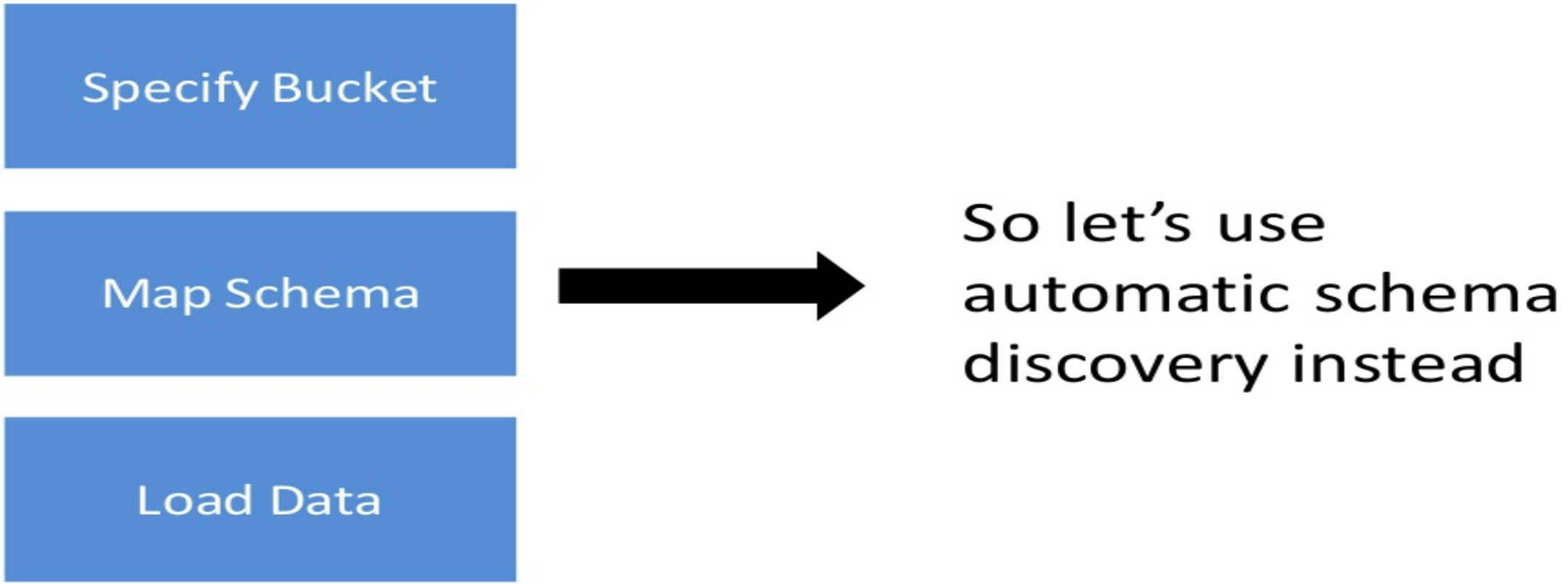
Load Data

Time Series Data

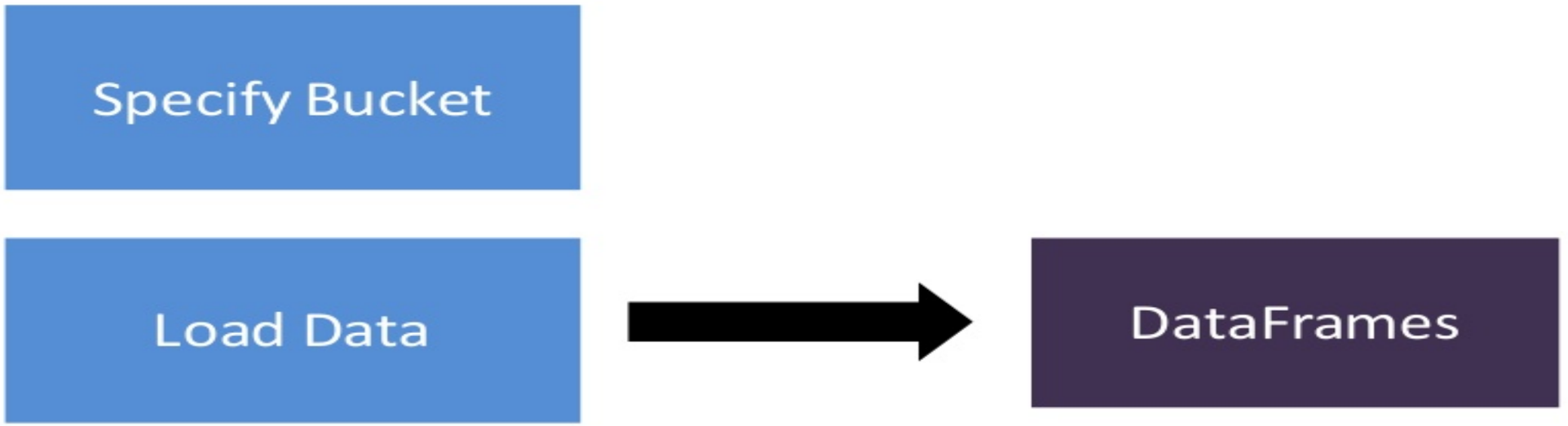
Time Series Data



Time Series Data



Time Series Data



Time Series Code

Specify Table

Load Data

```
val ts_table_name = "test-table"

df = sqlContext.re.option(
        "spark.riak.connection.hosts",
        "riak_host_ip:10017")
    .format("org.apache.spark.sql.riak")
    .load(ts_table_name)
    .select("time", "col1", "col2")
    .filter(s"time >= CAST($from AS TIMESTAMP)")
```

Time Series Code

Specify Table

Load Data

Use Data

```
val ts_table_name = "test-table"
```

```
df = sqlContext.re.option(
        "spark.riak.connection.hosts",
        "riak_host_ip:10017")
    .format("org.apache.spark.sql.riak")
    .load(ts_table_name)
    .select("time", "col1", "col2")
    .filter(s"time >= CAST($from AS TIMESTAMP)")
```

```
df.where(df("age") >= 50).select("id", "name")
```

```
df.groupBy("age").count
```

Time Series Code

Specify Table

Load Data

Use Data

Uses the Spark Data Source API

```
val ts_table_name = "test-

df = sqlContext.re.option(
        "spark.riak.connection.hosts",
        "riak_host_ip:10017")
    .format("org.apache.spark.sql.riak")
    .load(ts_table_name)
    .select("time", "col1", "col2")
    .filter(s"time >= CAST($from AS TIMESTAMP)")

df.where(df("age") >= 50).select("id", "name")

df.groupBy("age").count
```

LESSON #3

Optimize all the levels

LESSON #3

Optimize all the layers

2 primary interfaces to Riak

HTTP

Protocol Buffers

2 primary interfaces to Riak

Flexibility

Performance

BACKGROUND

Protocol Buffers

- Data serialization and interchange
- Developed by Google
- IDL + RPC
- Messages serialized to binary wire format
- Library support for 20+ languages

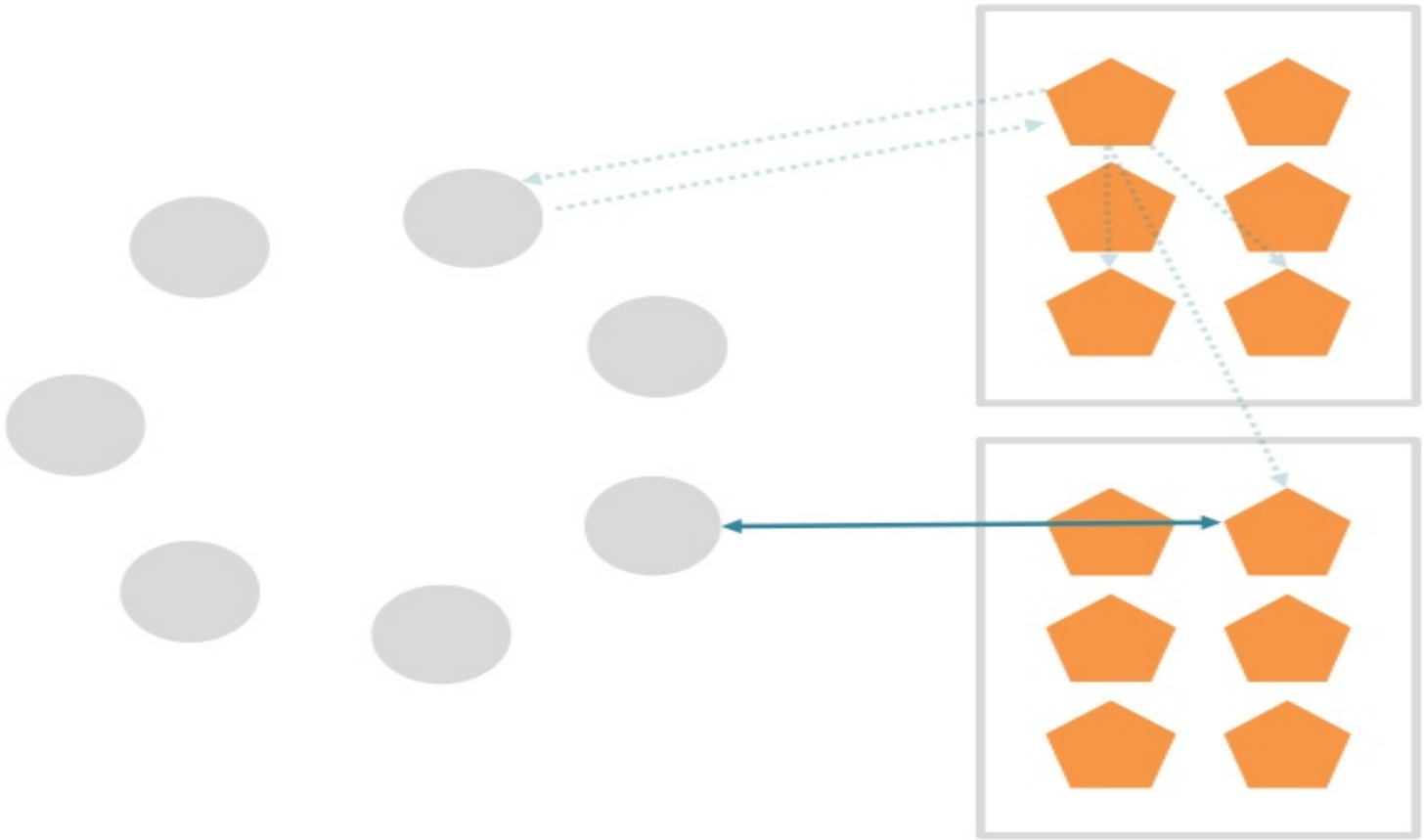
BACKGROUND

Protocol Buffers

Note: In Riak, you typically don't have to know the details, the client SDKs take care of it for you

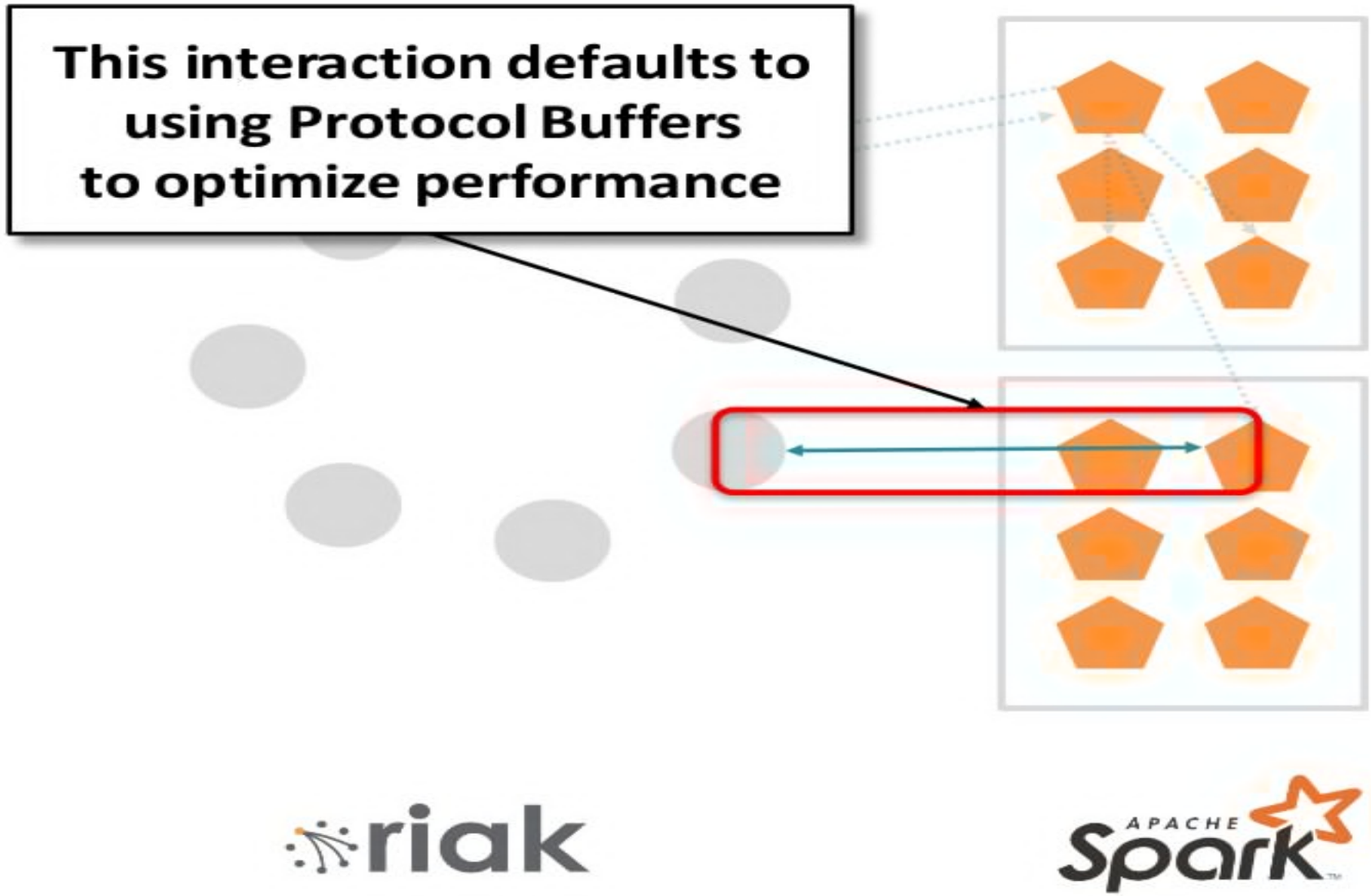
How much faster?

150-300% faster



riak

APACHE Spark™



riak

APACHE Spark

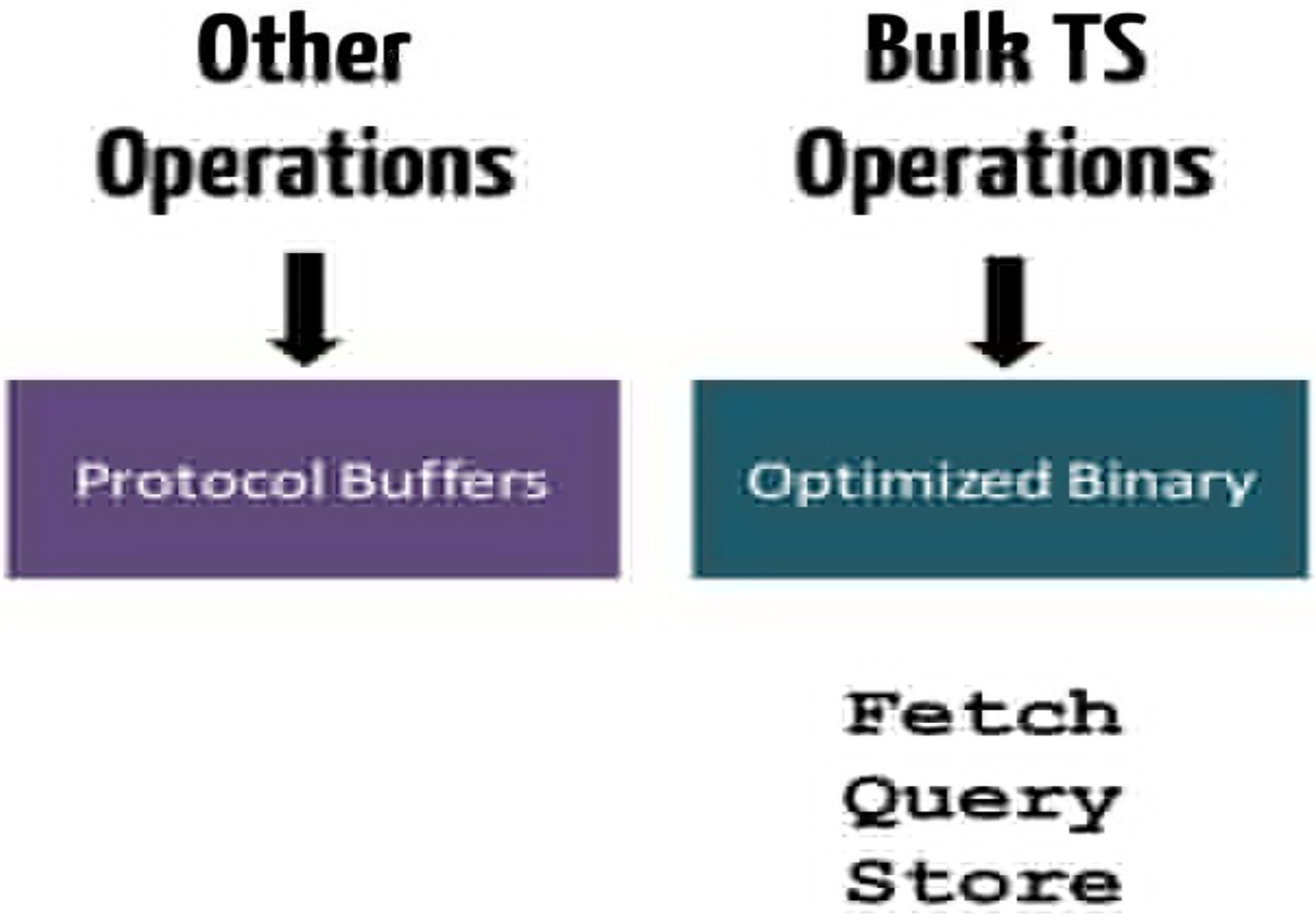
HTTP

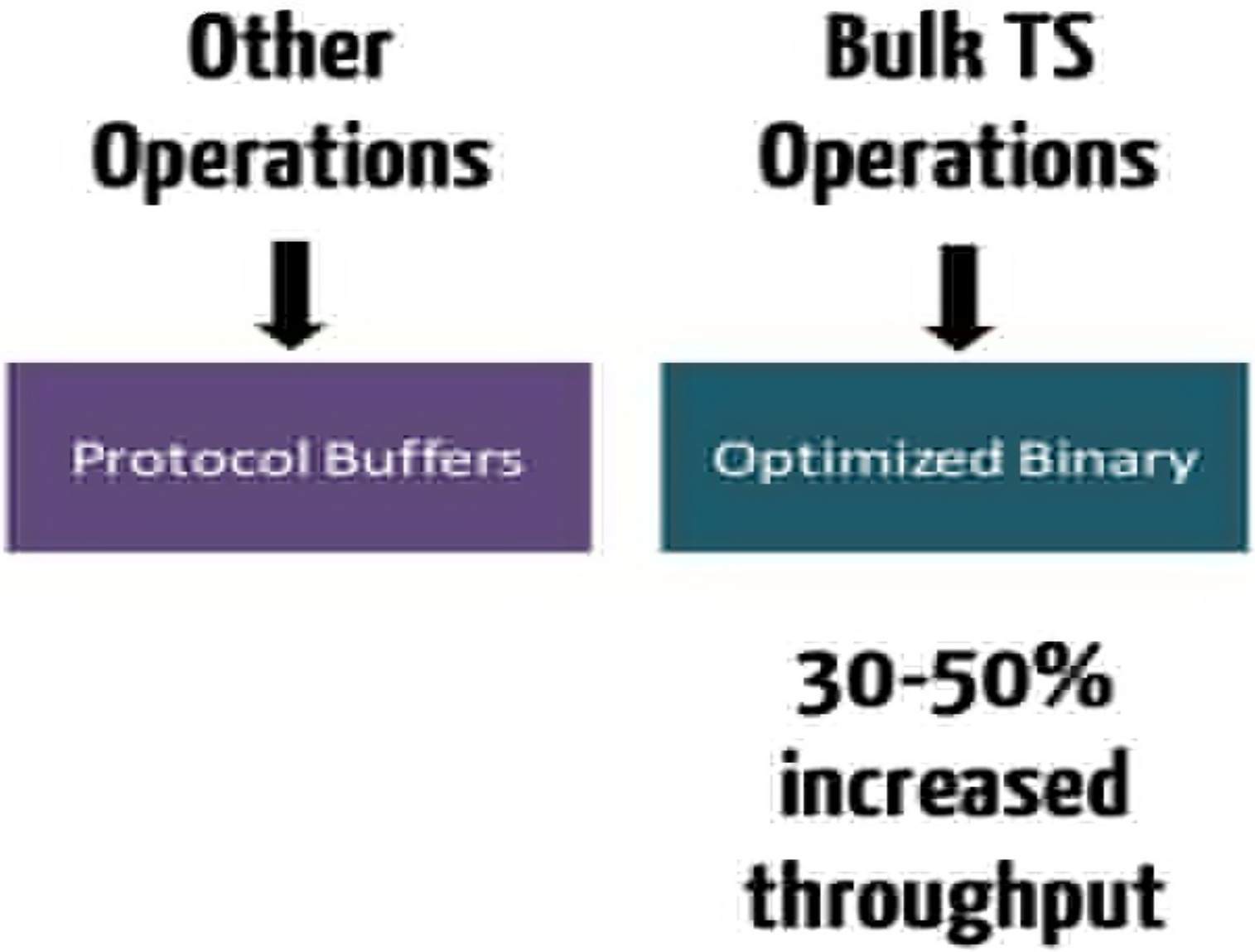
Protocol Buffers

What if we can make this faster?


HTTP
Protocol Buffers
Optimized Binary

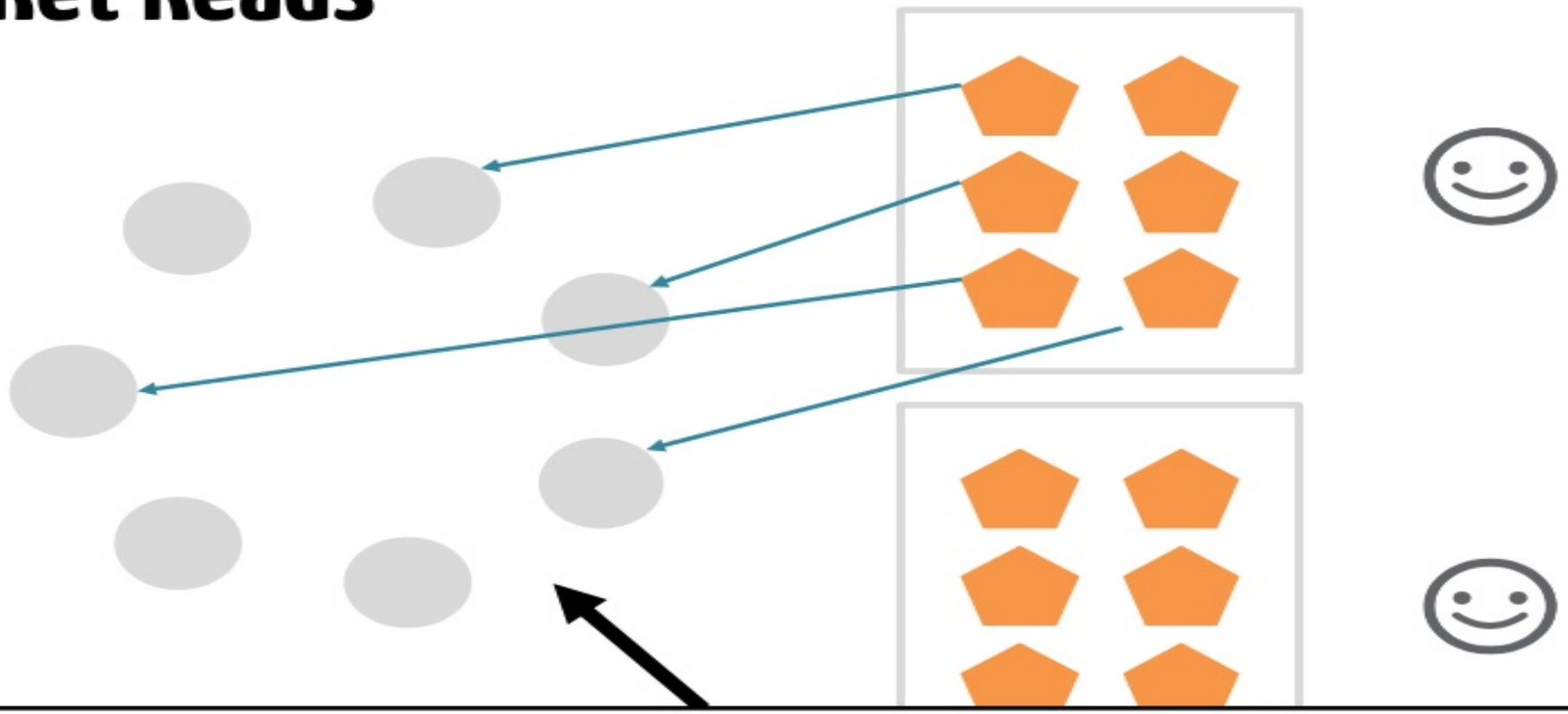
Spark-Riak Connector dynamically selects based on query type


Other Operations
Bulk TS Operations
Protocol Buffers
Optimized Binary
Fetch
Query
Store


Other Operations
Bulk TS Operations
Protocol Buffers
Optimized Binary
30-50% increased throughput

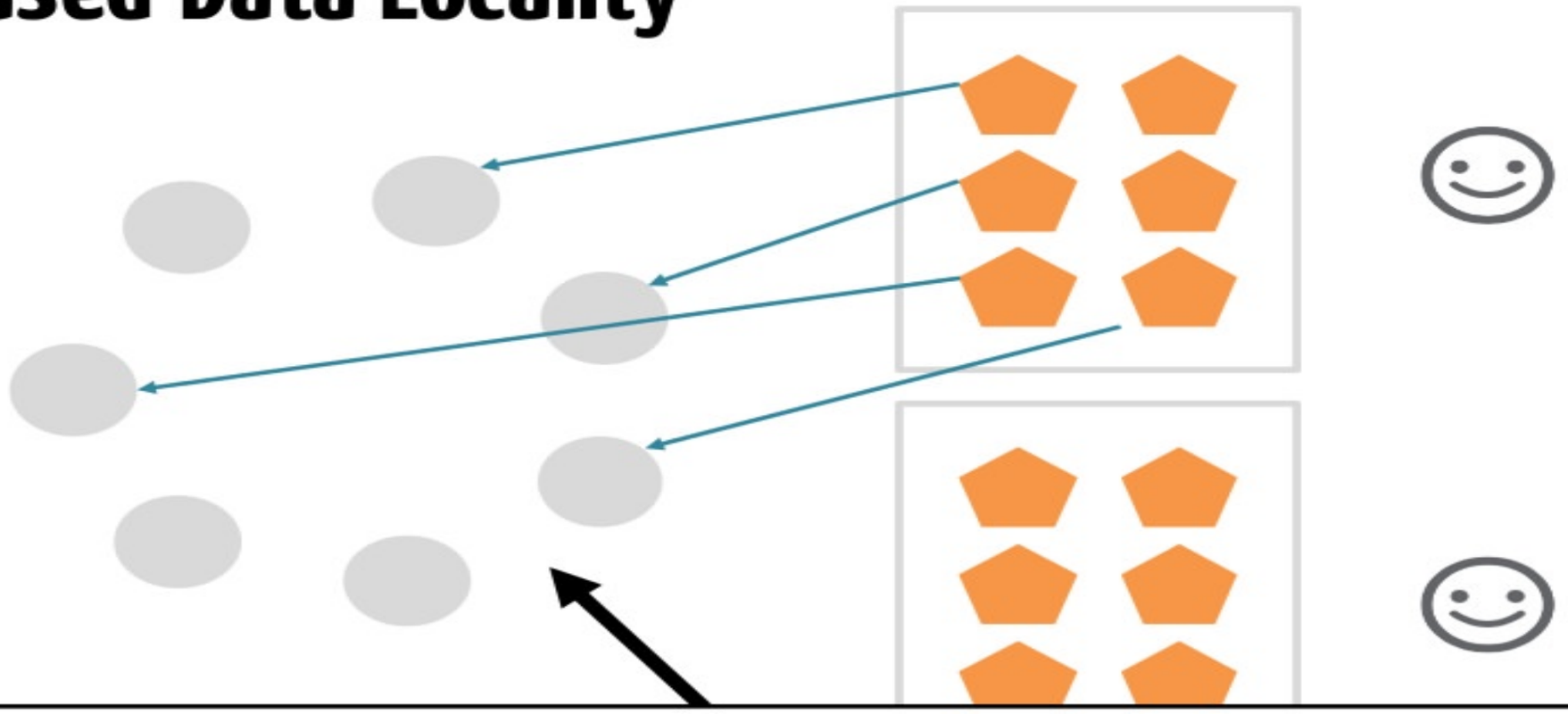
2 use case-specific optimizations

Full Bucket Reads



Riak KV supports these as optimization:
Give me all the data in this bucket,
and I'll work with it over here in Spark

Time-based Data Locality



Riak TS uses a time based ‘quanta’ to intelligently partition data across the cluster based on user-specified time

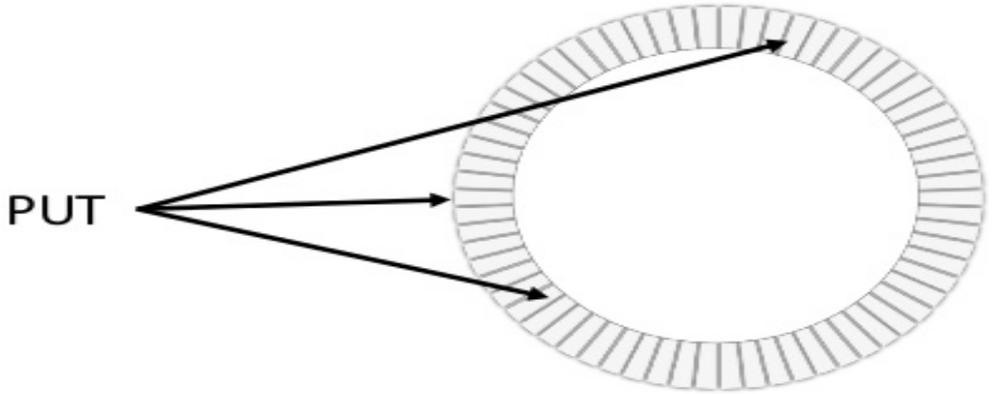
Location, location, location

Key Nature system provides

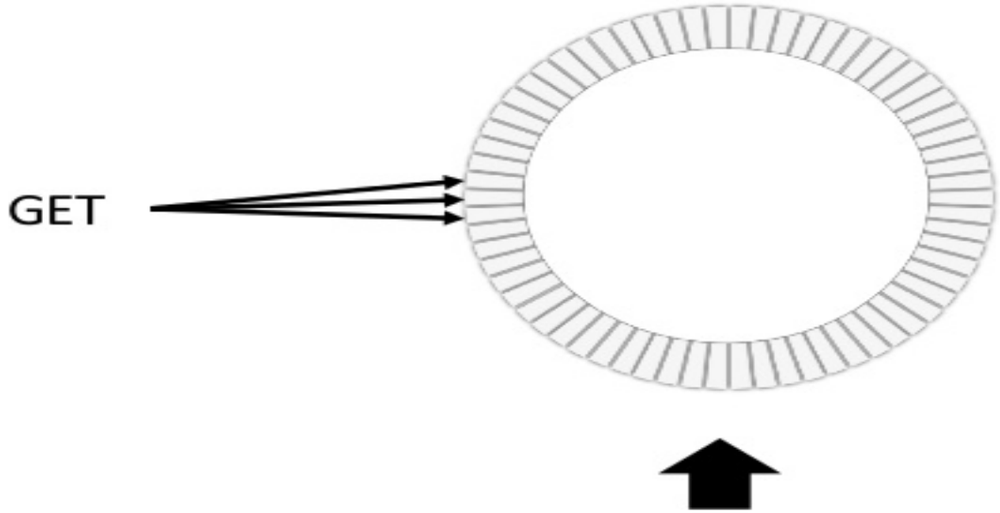
Three Gorges ecosystem provides

Location, location, location

Key/Value cluster vnodes



Time Series cluster vnodes



Local grouping based on time quanta
Write to same vnode
Query direct to data

Riak Time Series SQL

Define table

```
CREATE TABLE WEATHER (
   region          VARCHAR    NOT NULL,
   city            VARCHAR    NOT NULL,
   time            TIMESTAMP  NOT NULL,
   [illegible]     [illegible]

   [illegible]
      [illegible]
      [illegible]
   )
)
```

Riak Time Series SQL

Define table

```
CREATE TABLE WEATHER (
   region       VARCHAR   NOT
   city         VARCHAR   NOT
   time         TIMESTAMP NOT NULL,
   temperature  DOUBLE,

   PRIMARY KEY(
      (region, state, QUANTUM(time, 2, 'h')),
      region, state, time
   )
)
```

The quantum is the tunable key to performance

Riak Time Series SQL

Define table

```
CREATE TABLE WEATHER (
   region       VARCHAR   NOT NULL,
   city         VARCHAR   NOT NULL,
   time         TIMESTAMP NOT NULL,
   temperature  DOUBLE,

   PRIMARY KEY(
      (region, state, QUANTUM(time, 2, 'h')),
      region, state, time
   )
)
```

Query

```
SELECT * FROM WEATHER
WHERE city = 'Brussels'
      time >= '2016-01-01' AND
      time <= '2016-02-01 00:00:00'
```

LESSON #4

LESSON #4

Be polyglot

Support multiple languages

Python

Setup

```
import pyspark_riak

conf = pyspark.SparkConf().setAppName("My Spark Riak App")
conf.set("spark.riak.connection.host", "127.0.0.1:8087")

sc = pyspark.SparkContext(conf)
pyspark_riak.riak_context(sc)
```

Python

Setup

```
import pyspark_riak

conf = pyspark.SparkConf().setAppName("My Spark Riak App")
conf.set("spark.riak.connection.host", "127.0.0.1:8087")

sc = pyspark.SparkContext(conf)
pyspark_riak.riak_context(sc)
```

Write

```
my_data = [{'key0':{'data': 0}}, {'key1':{'data': 1}}]
kv_write_rdd = sc.parallelize(my_data)
kv_write_rdd.saveToRiak(`kv_sample_bucket’)
```

Python

Setup

```
import pyspark_riak

conf = pyspark.SparkConf().setAppName("My Spark Riak App")
conf.set("spark.riak.connection.host", "127.0.0.1:8087")

sc = pyspark.SparkContext(conf)
pyspark_riak.riak_context(sc)
```

Write

```
my_data = [{'key0':{'data': 0}}, {'key1':{'data': 1}}]
kv_write_rdd = sc.parallelize(my_data)
kv_write_rdd.saveToRiak(`kv_sample_bucket’)
```

Read

```
kv_read_rdd = sc.riakBucket(`kv_sample_bucket’).queryAll()
print(kv_read_rdd.collect())
```

Spark Streaming

Spark Streaming

Spark-Riak Streaming

Setup

```
import com.basho.riak.spark.streaming._

val ssc = new StreamingContext(sparkConf, Seconds(1))
```

Stream

```
val lines = ssc.socketTextStream(serverIP, serverPort)

val errs = lines.filter(lines => lines contains "ERROR")
```

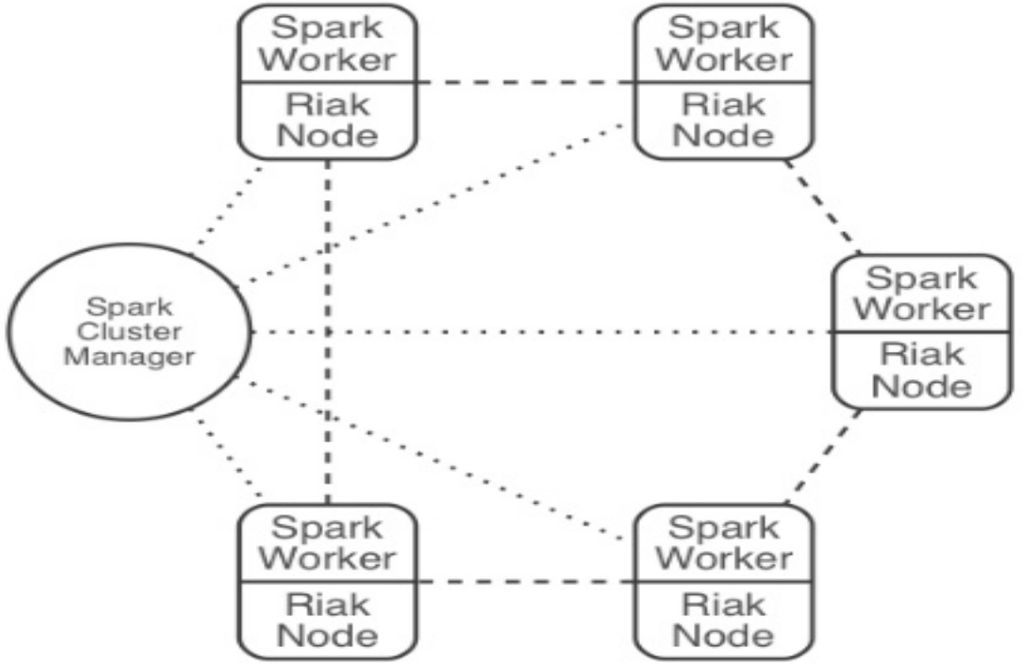
Save

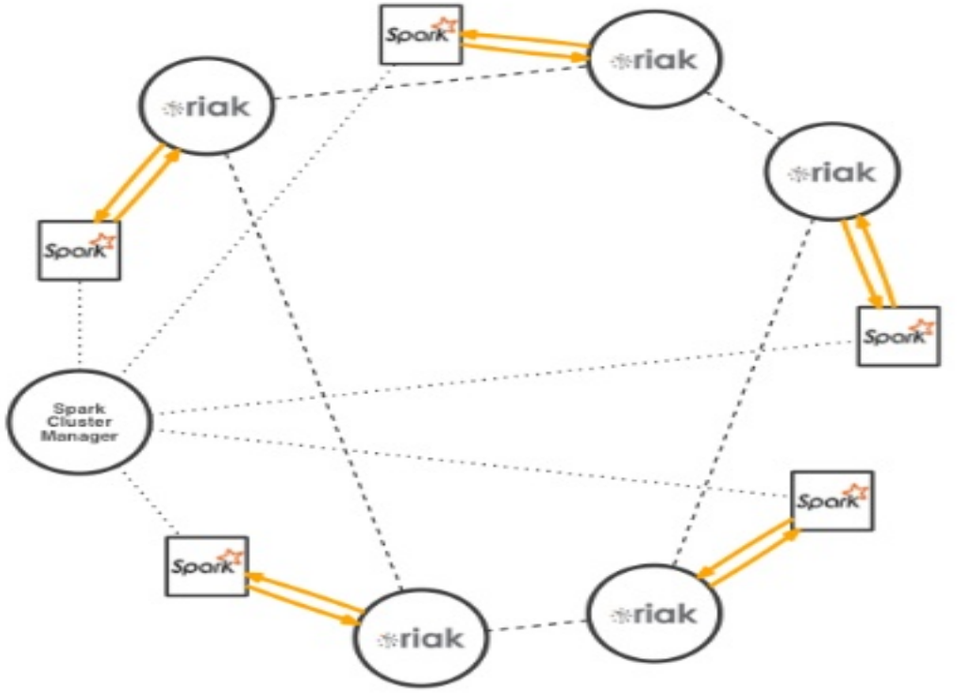
```
errs.saveToRiak("test-bucket-4store")
```

Deployment

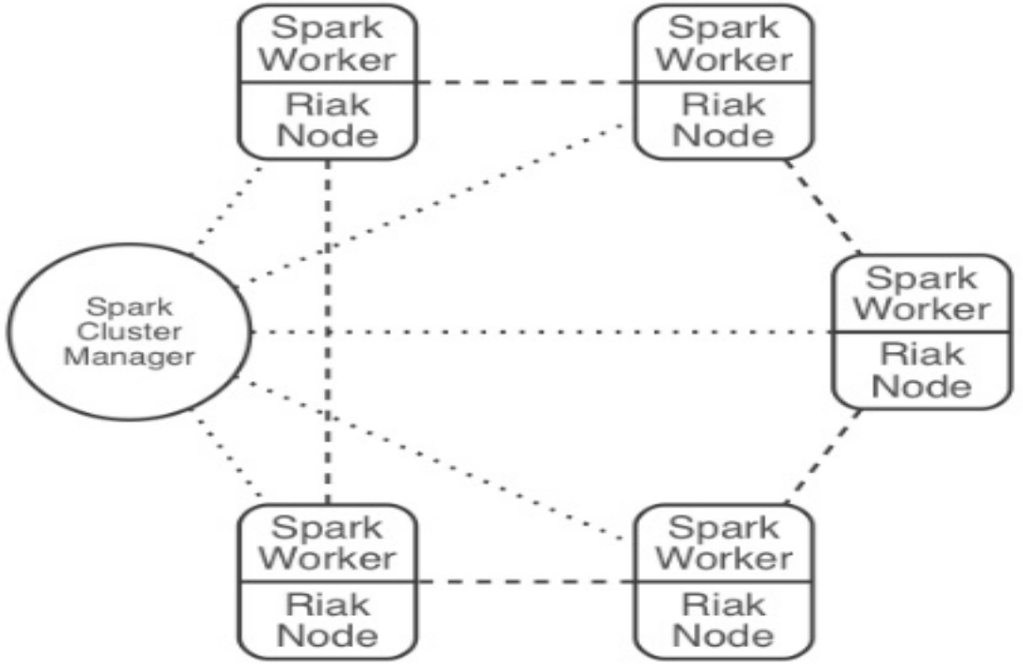
On premise
Cloud
Hybrid
Geo-distributed

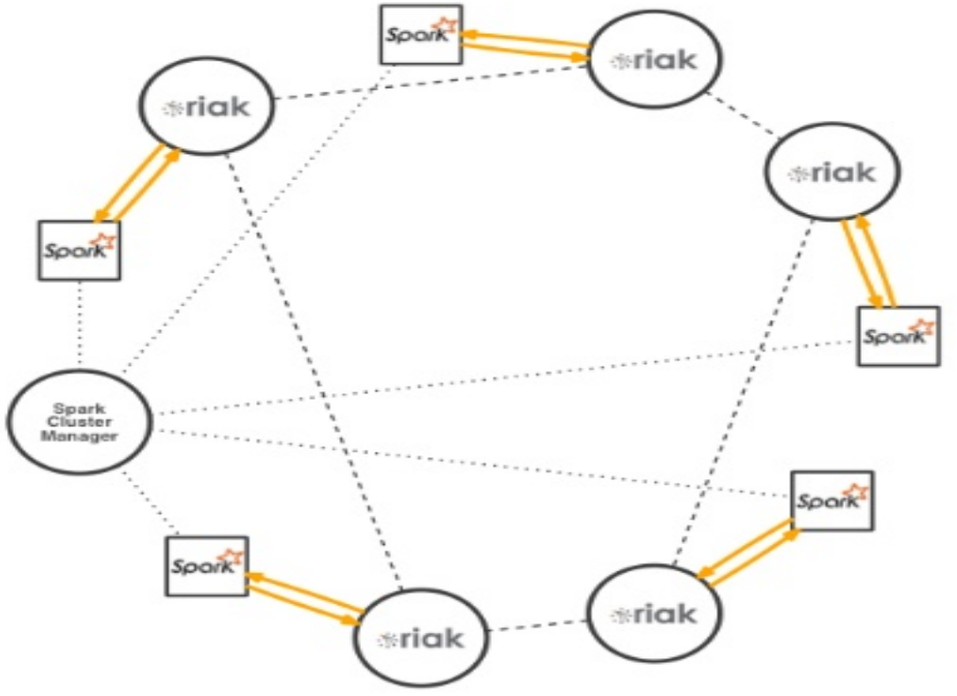
Deployment


Spark Worker
Riak Node
Spark Worker
Riak Node
Spark Cluster Manager
Spark Worker
Riak Node
Spark Worker
Riak Node
Spark Worker
Riak Node


Spark
riak
riak
Spark
riak
Spark
Spark Cluster Manager
Spark
Spark
riak
riak

Deployment


Spark Worker
Riak Node
Spark Worker
Riak Node
Spark Cluster Manager
Spark Worker
Riak Node
Spark Worker
Riak Node
Spark Worker
Riak Node


Spark
riak
riak
Spark
riak
Spark
Spark Cluster Manager
Spark
Spark
riak
riak

Deployment

Apache
MESOS™

https://github.com/basho-labs/riak-mesos

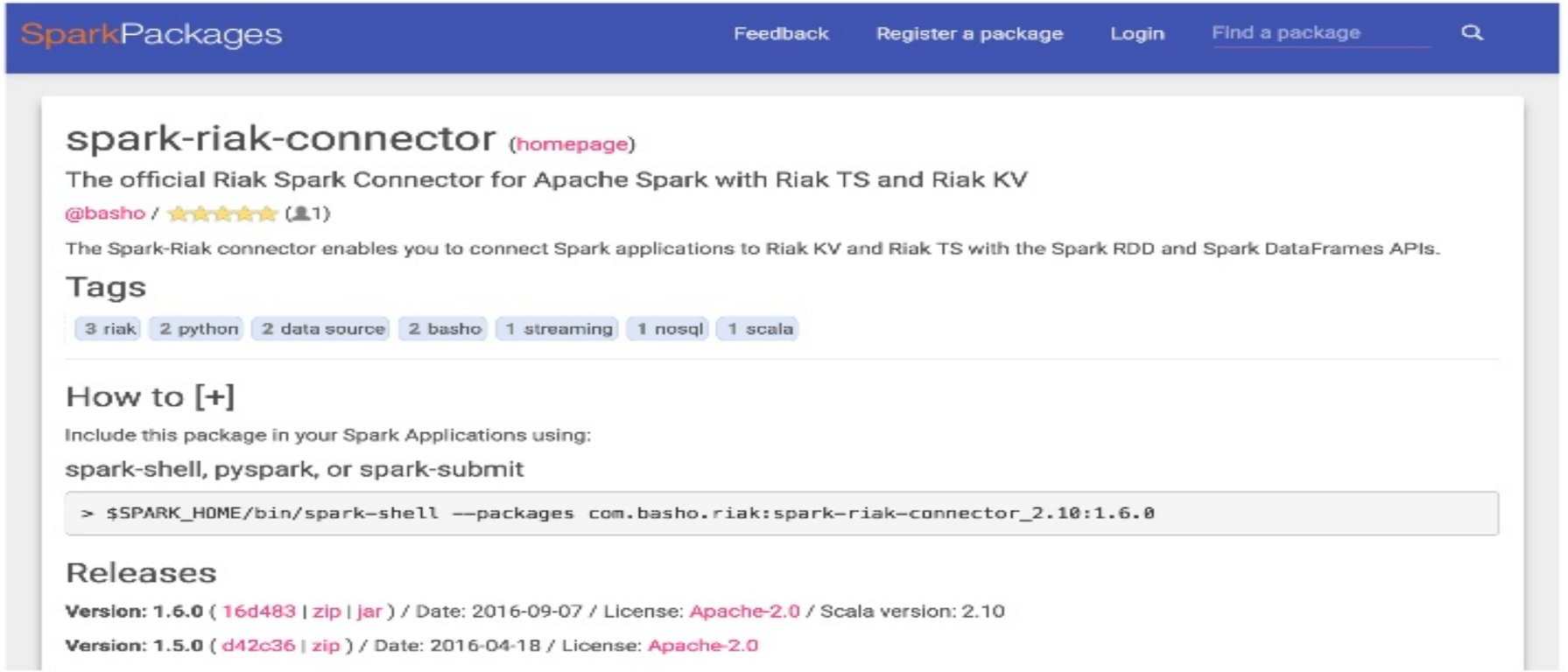
LESSON #5

Simplify

LESSON #5

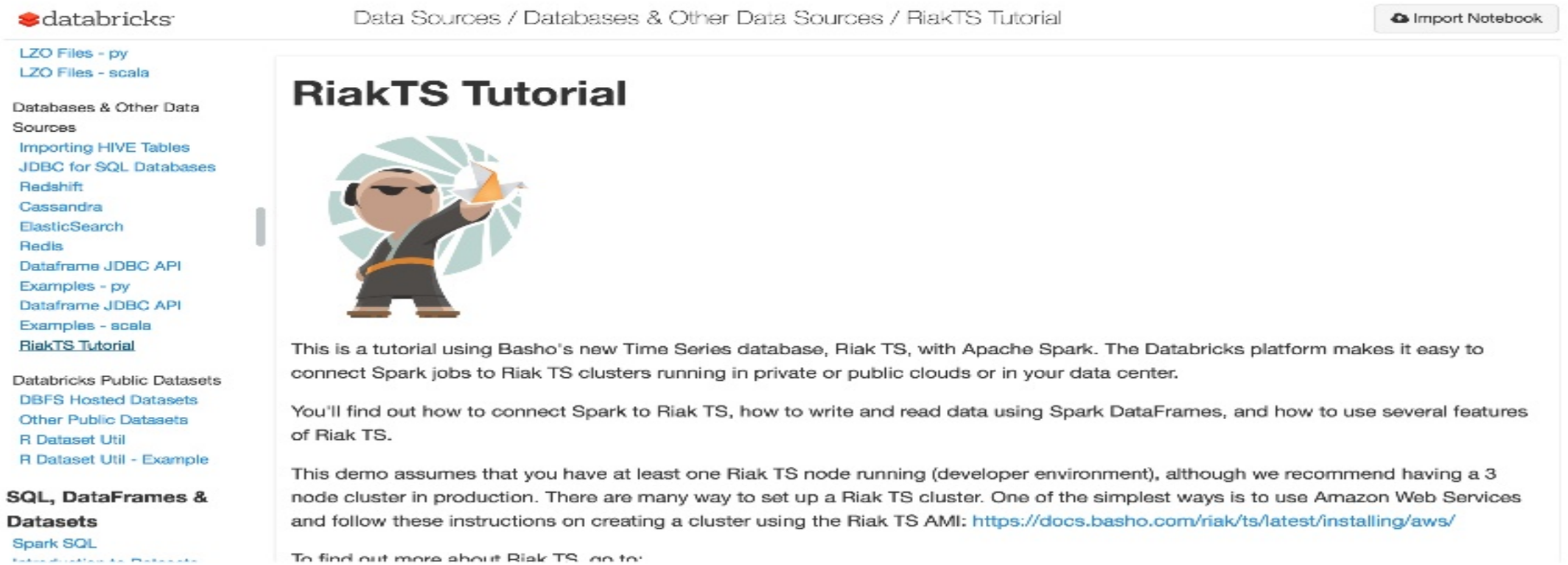
How to reduce friction?

Connector hosted at Spark-Packages



https://spark-packages.org/package/basho/spark-riak-connector

Tutorial notebook on Databricks.com

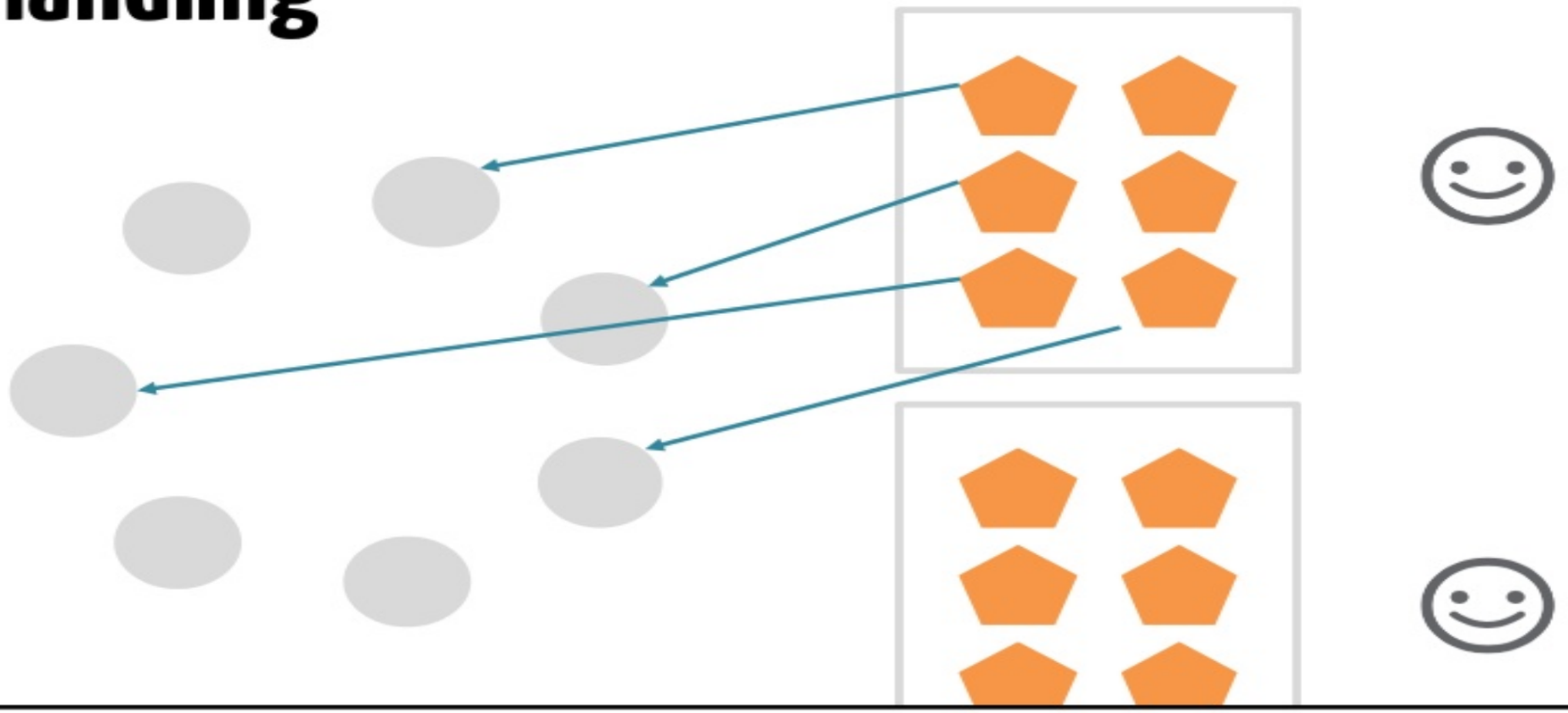


https://docs.cloud.databricks.com/docs/latest/databricks_guide/index.html

BONUS LESSON

Don’t die

Failure Handling



If a Riak node dies during data retrieval, Spark connector will request an Alternative Coverage Plan

Next steps for Riak-Spark?

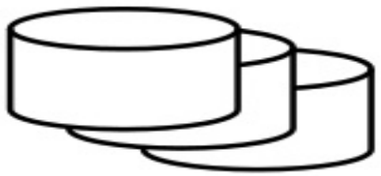
Spark 2.0

DataSets

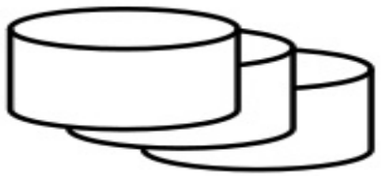
Structured Streaming

So, back to the question...

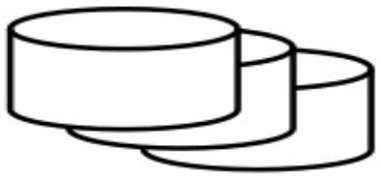
NoSQL + APACHE Spark™ = ?



NoSQL + APACHE Spark™ =



NoSQL + APACHE Spark™ =



LESSONS

Parallelize
Map smart
Optimize all the levels
Be flexible
Simplify

Thank You

@johnmusser

@basho

Photo Credits

Race car: Spacesuit Media
Intellicore application screenshots: Intellicore, http://www.intellicore.tv/