

Lessons Learned

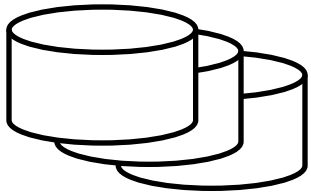
Optimizing NoSQL for Apache Spark

John Musser @johnmusser / Basho @basho

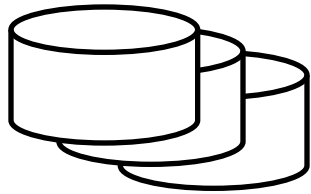
DataEngConf NYC, 2016



NoSQL



NoSQL



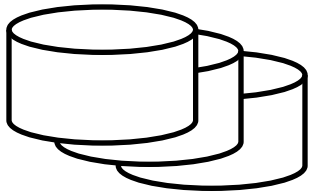
Key-Value

Document

Columnar

Graph

NoSQL




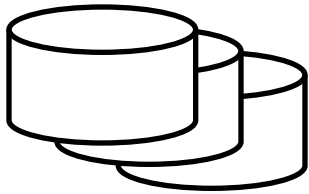
Key-Value

Document

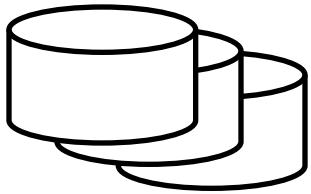
Columnar

Graph

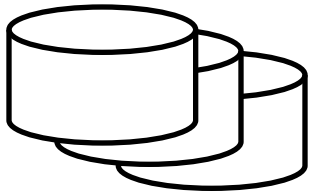
NoSQL +  APACHE
*Spark*TM



NoSQL +  = ?



NoSQL +  = 



How do we turn this...



into this?





We built this.



We built this.

Here are our lessons...

Parallelize

Map smart

Optimize all the levels

Be flexible

Simplify

BACKGROUND

Riak



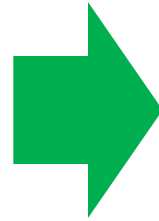
- Distributed, key-value NoSQL database
 - Known for scalability, reliability, ops simplicity
 - Launched 2009, used by 1/3 of Fortune 50
 - Open source (Apache), on GitHub
- <https://github.com/basho/riak/>
- Enterprise Edition, see multi-cluster replication

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

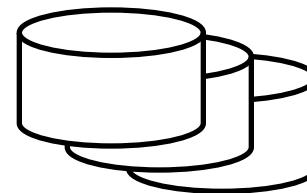
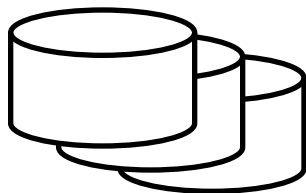
Time Series
data



IoT / Device data
Metrics data
Event data
Streaming data

Released
in 2016

Riak Core



Riak TS

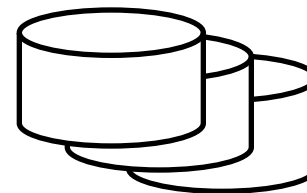
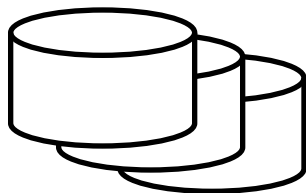
Released
in 2016

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

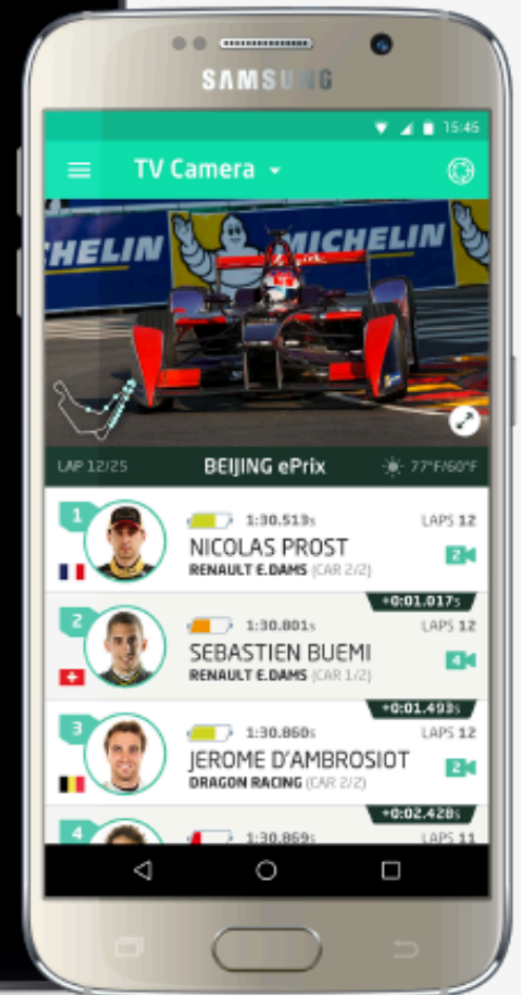
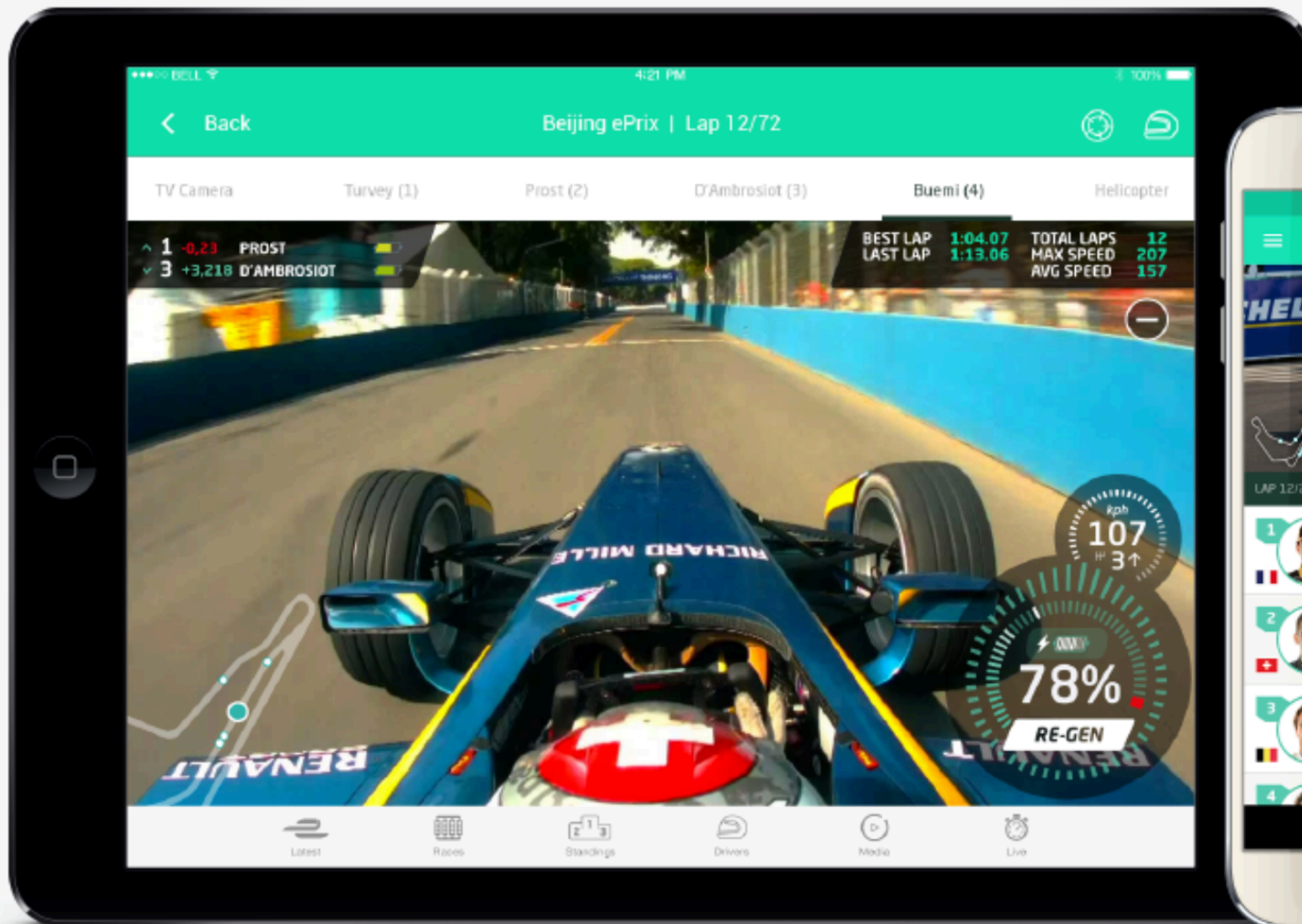




Intellicore Sports Data Platform

- 1GB telemetry per driver
- 400 packets/second
- 1.2M packets/race
- Platform setup for 40,000 TPS





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...)**



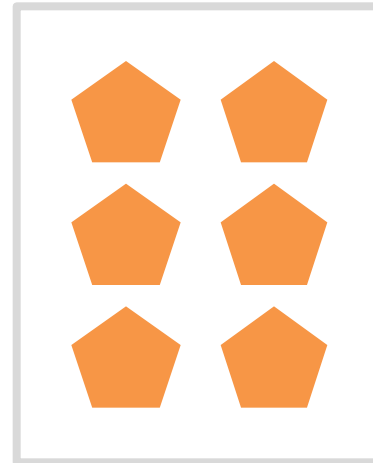
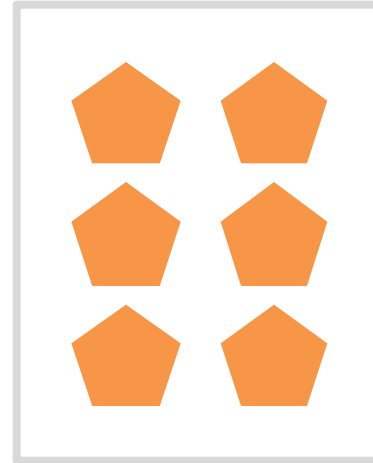
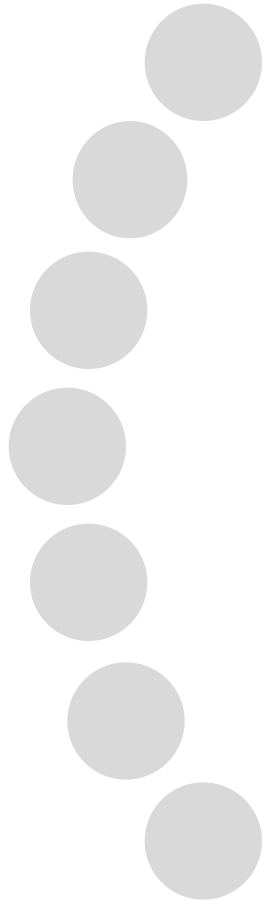
MESOS

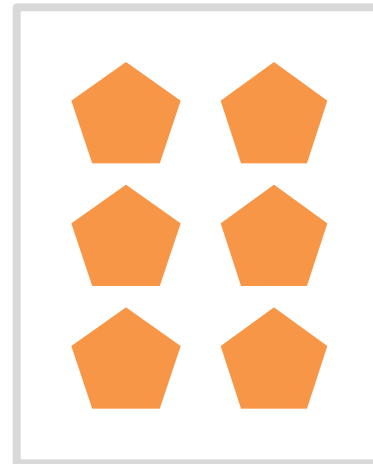
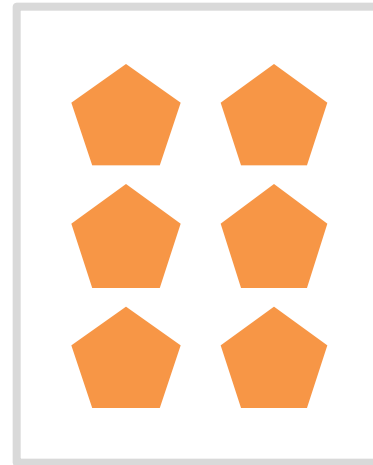
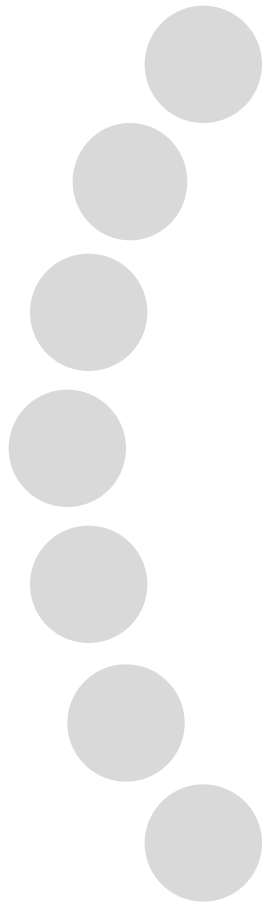


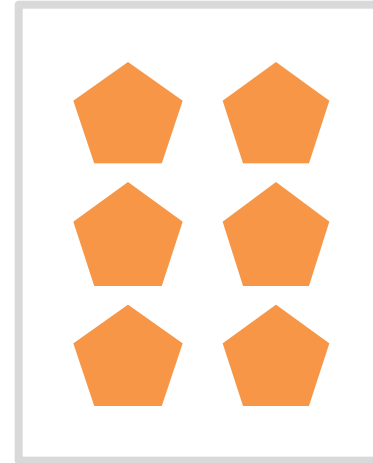
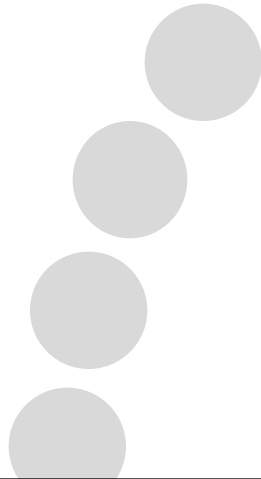
LESSON #1

Parallelize

whenever possible



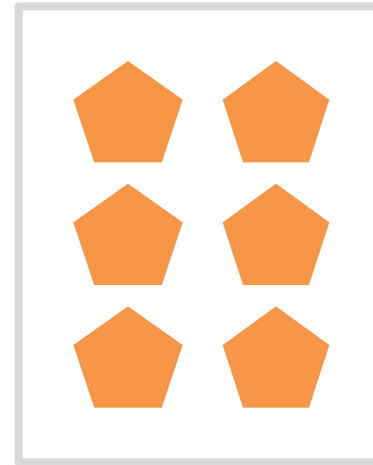
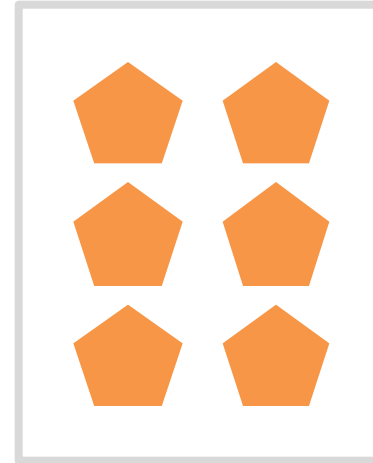
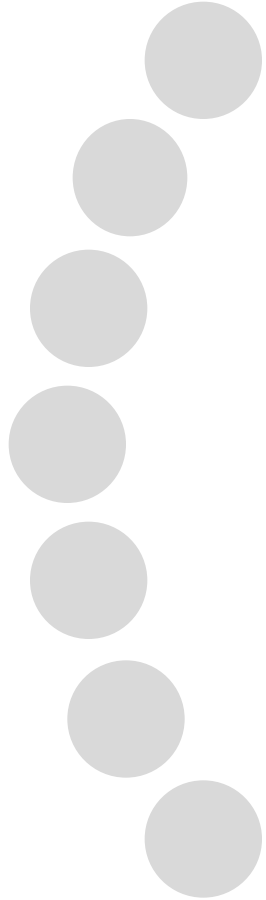




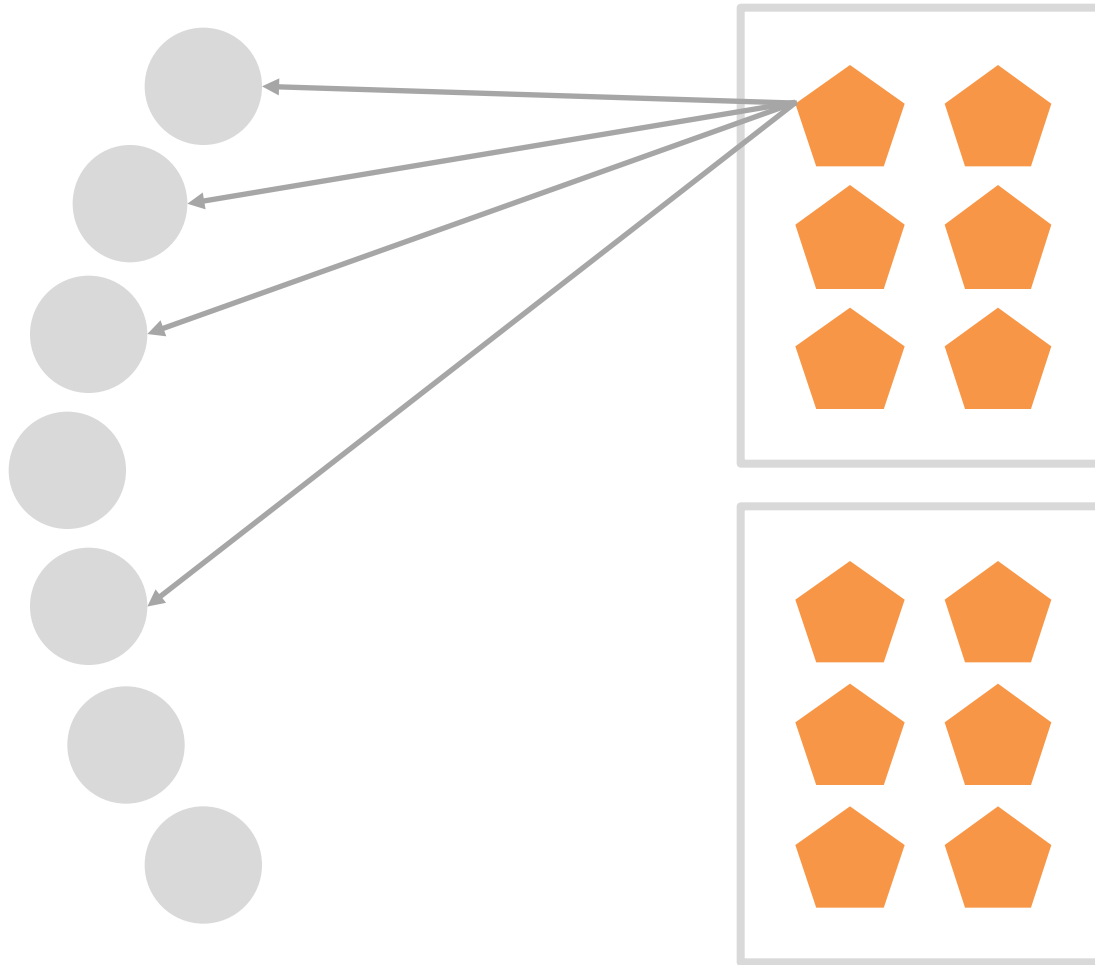
How to move lots of data quickly and efficiently?



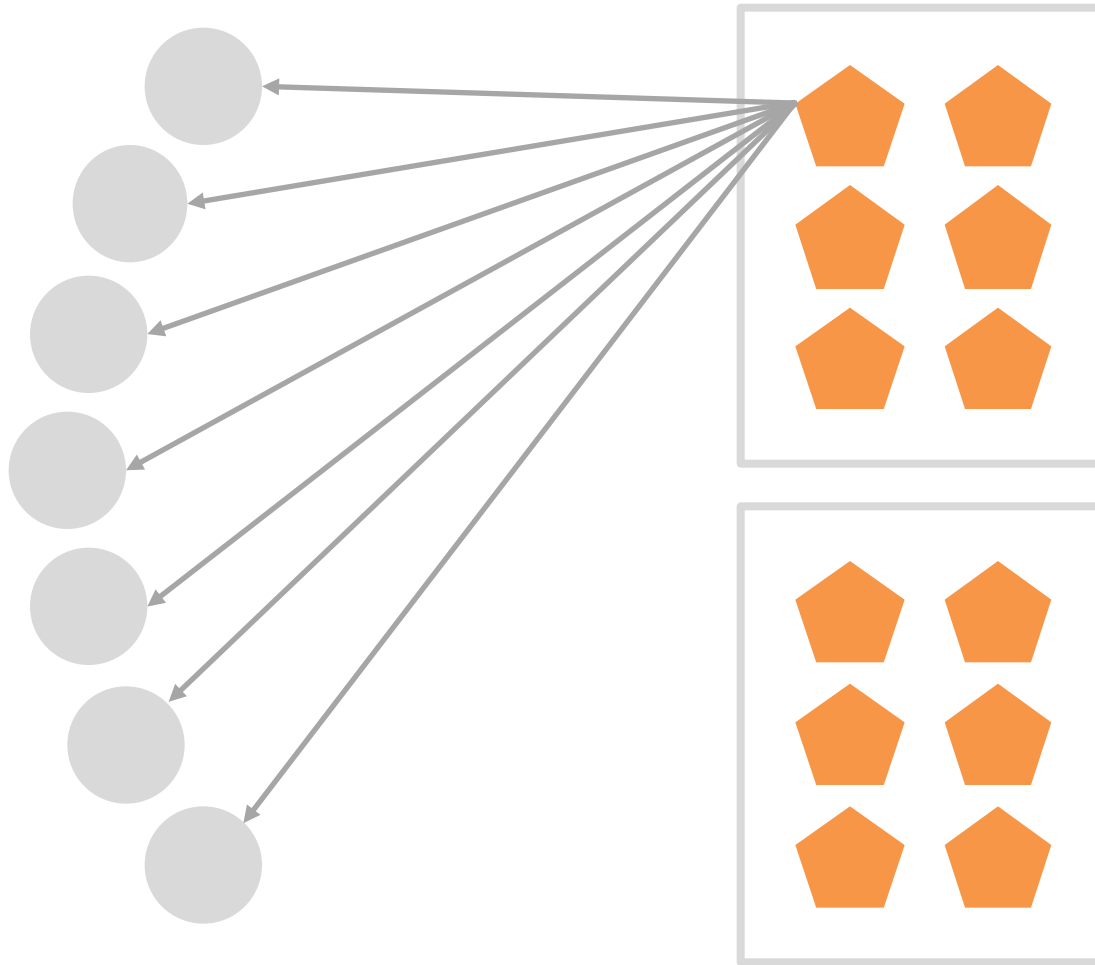
Using Direct Key-based GETs



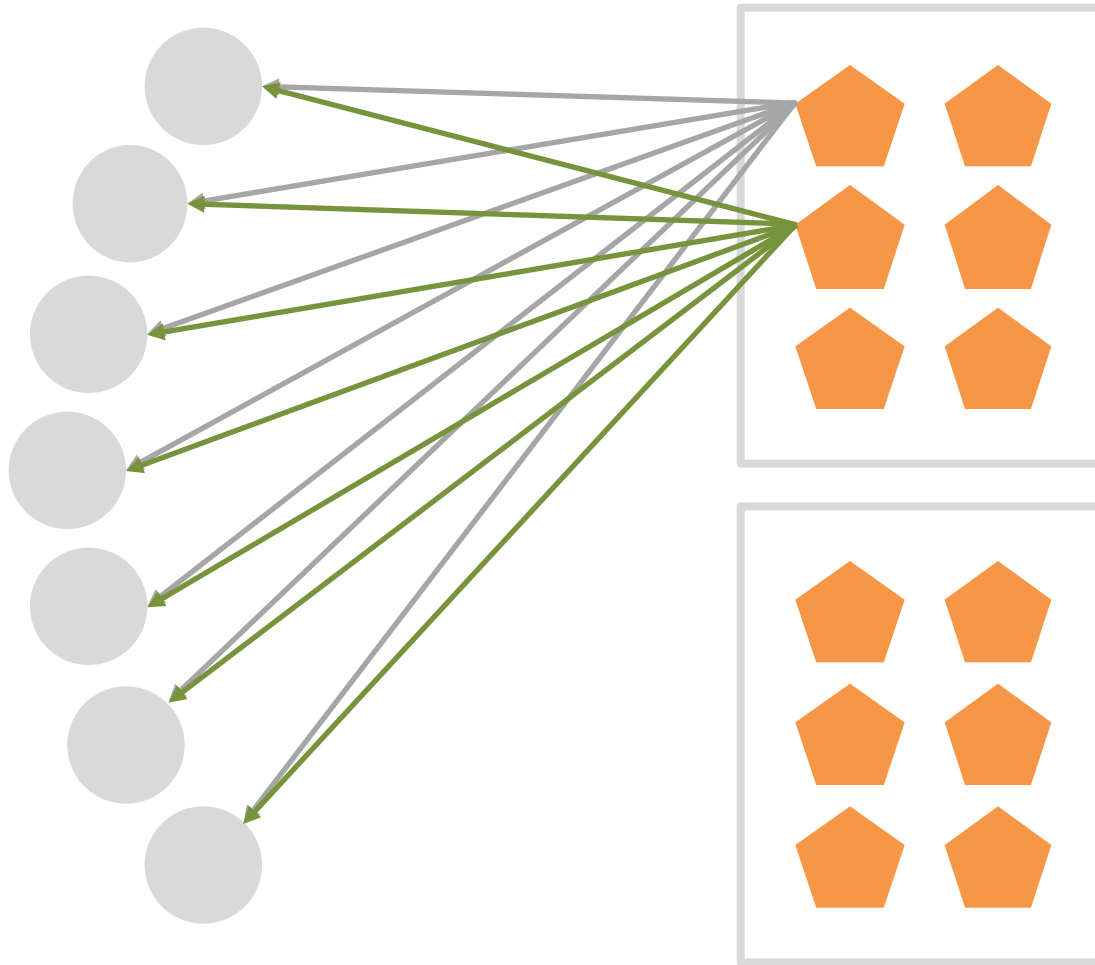
Using Direct Key-based GETs



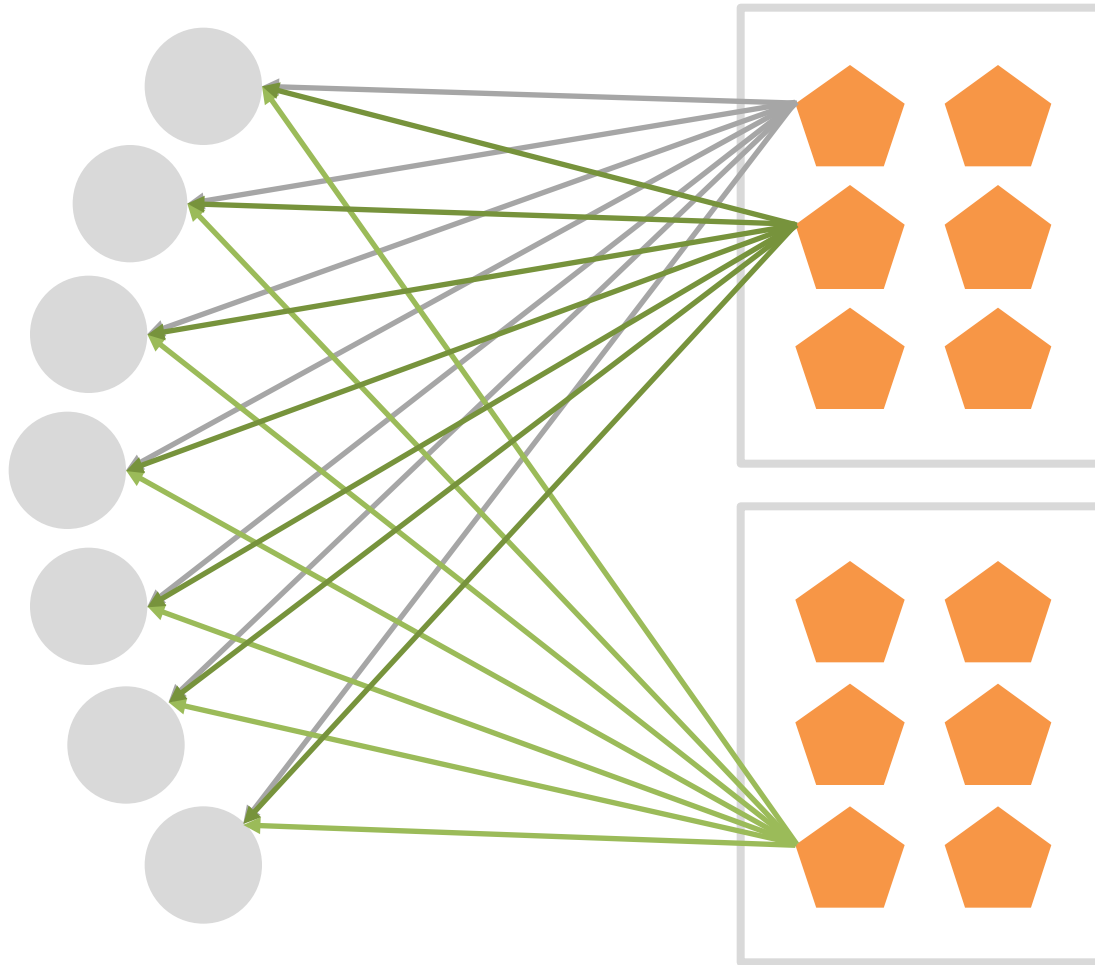
Using Direct Key-based GETs



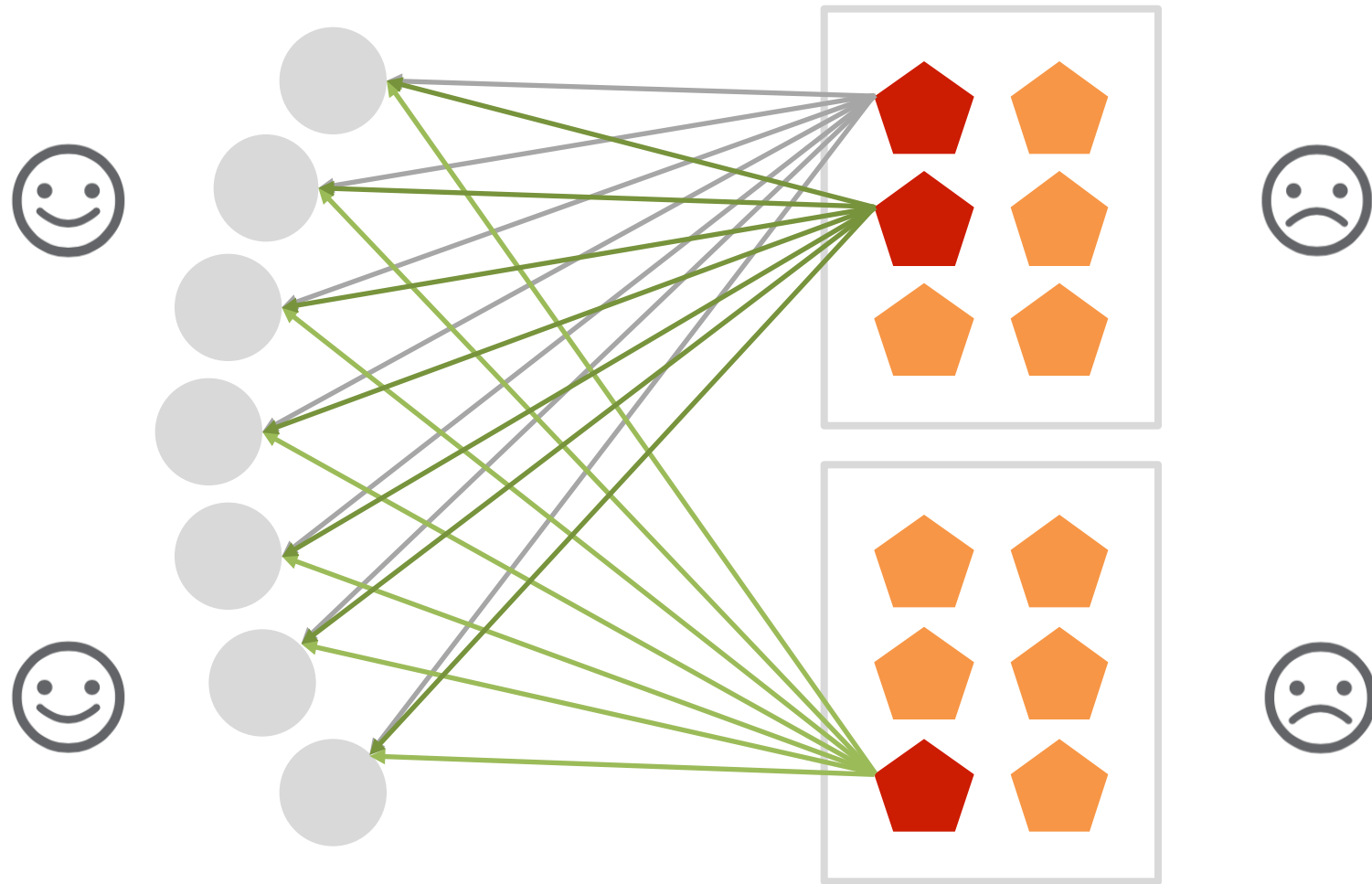
Using Direct Key-based GETs



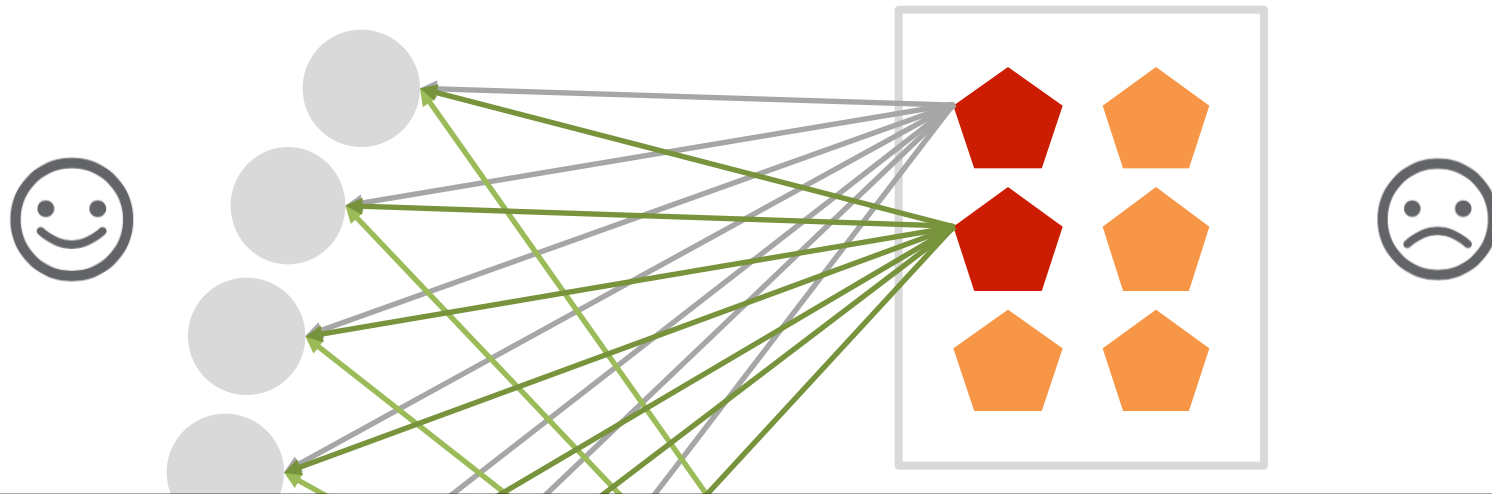
Using Direct Key-based GETs



Using Direct Key-based GETs



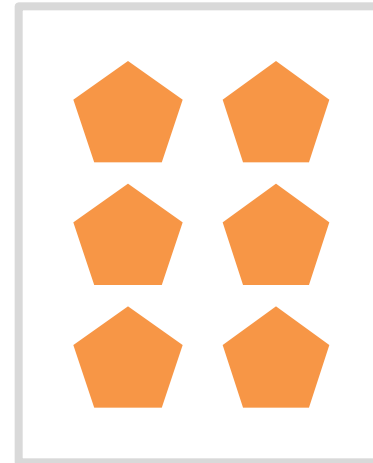
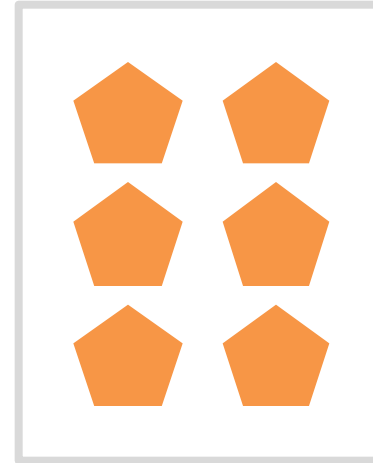
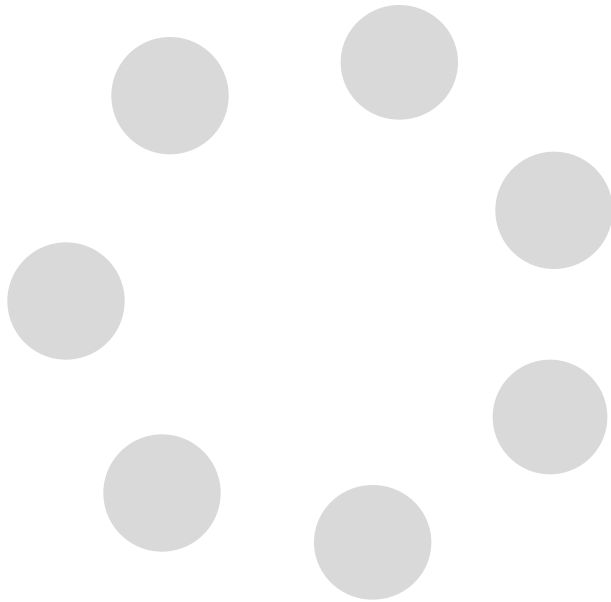
Using Direct Key-based GETs



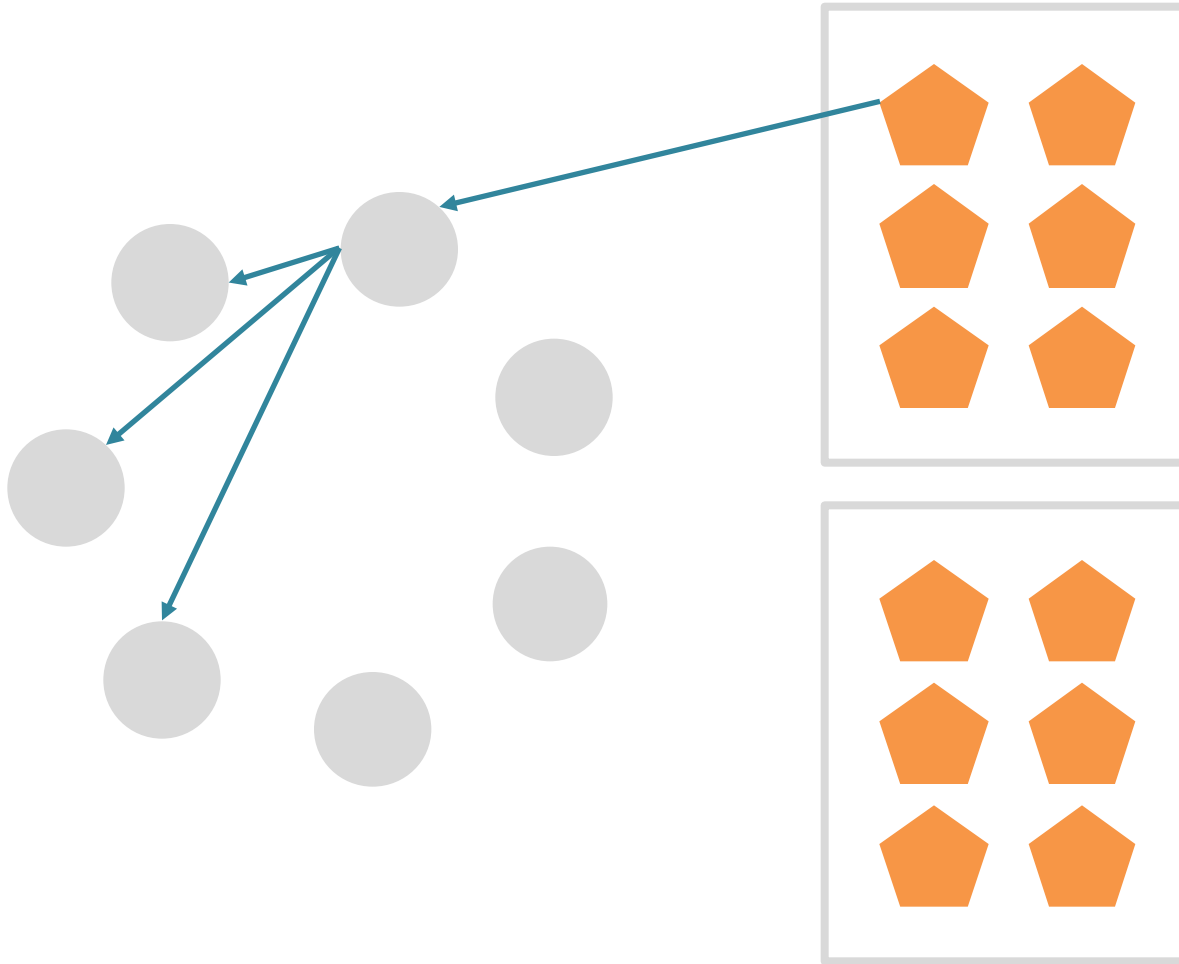
Lesson 1(a):
Too many Gets make Spark unhappy



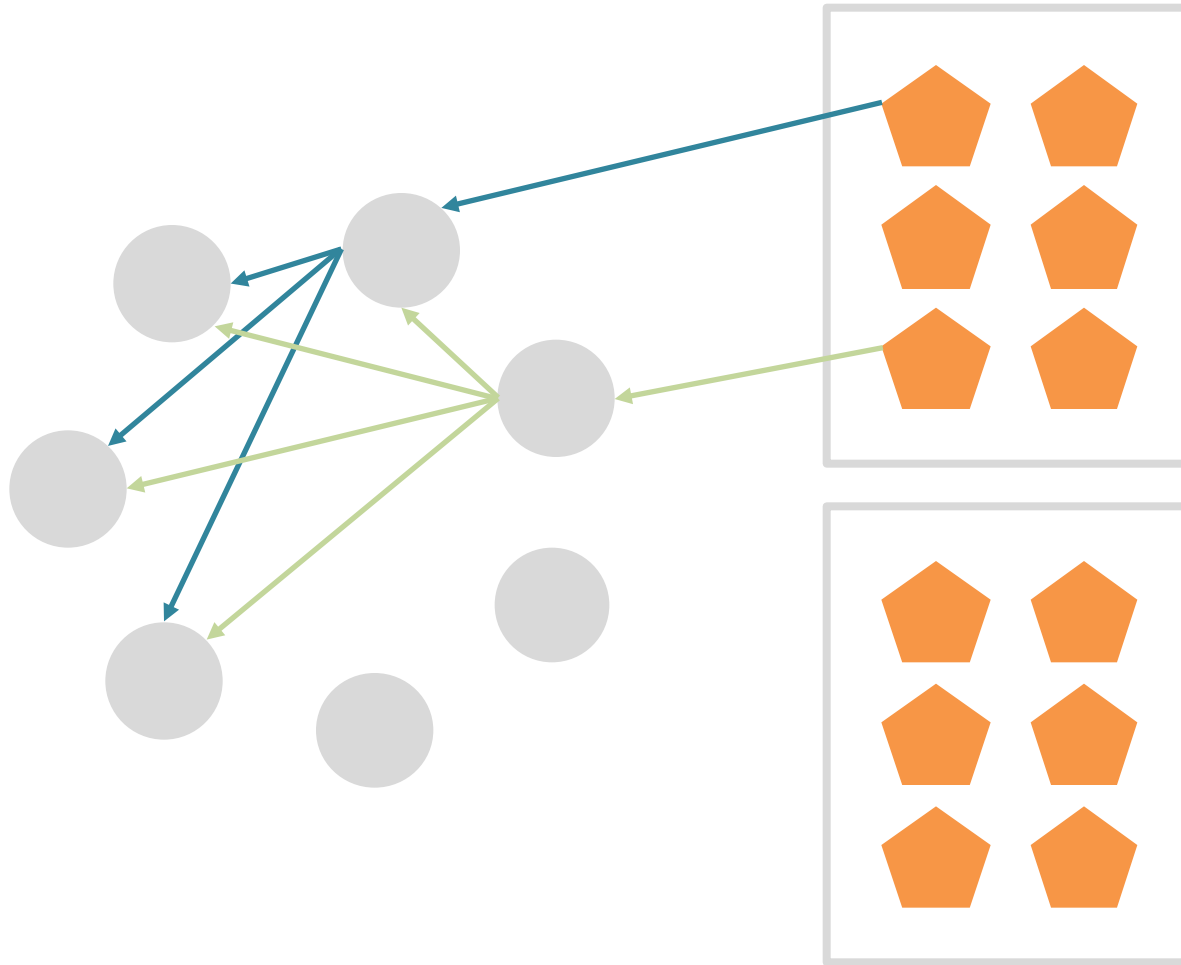
Using Secondary Index (2i)



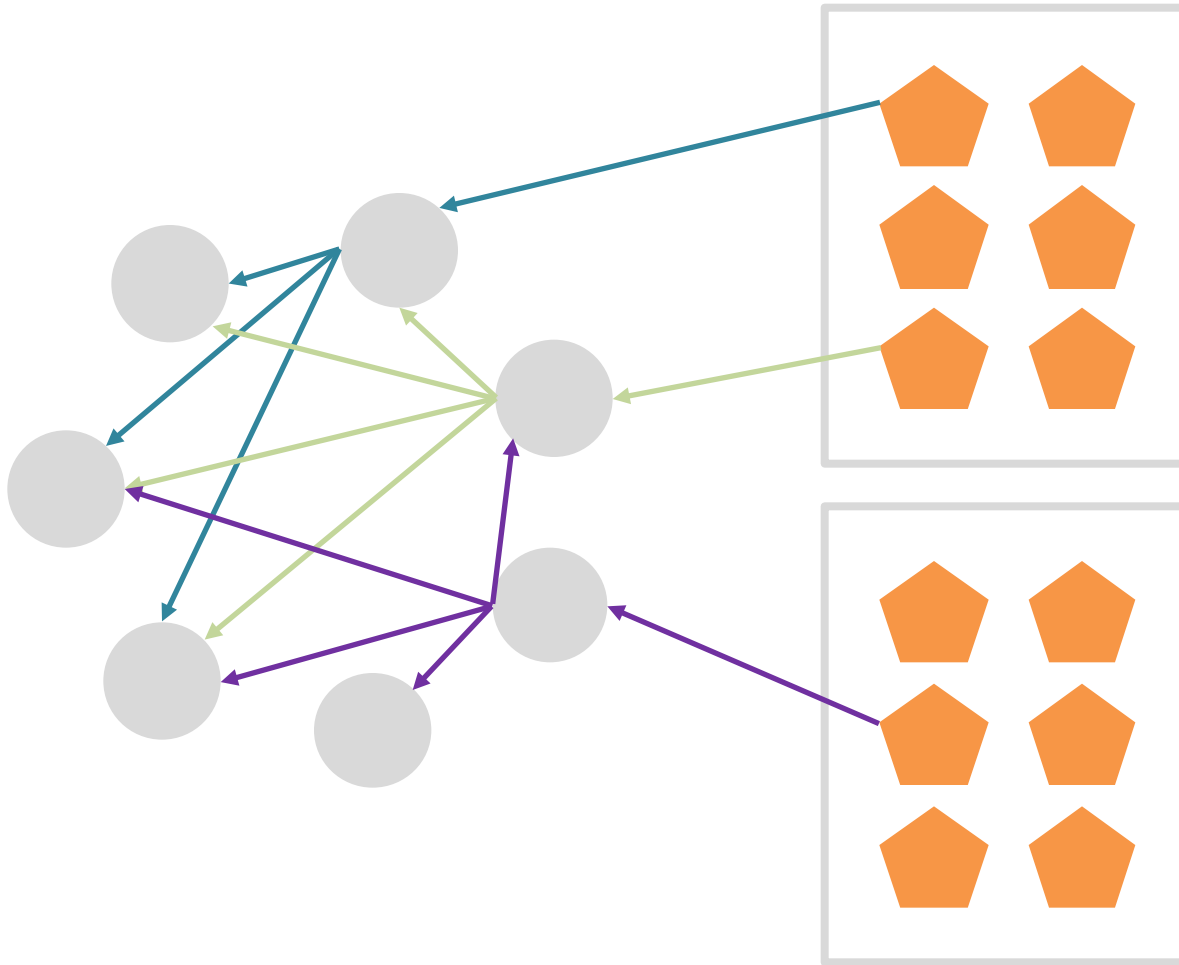
Using Secondary Index (2i)



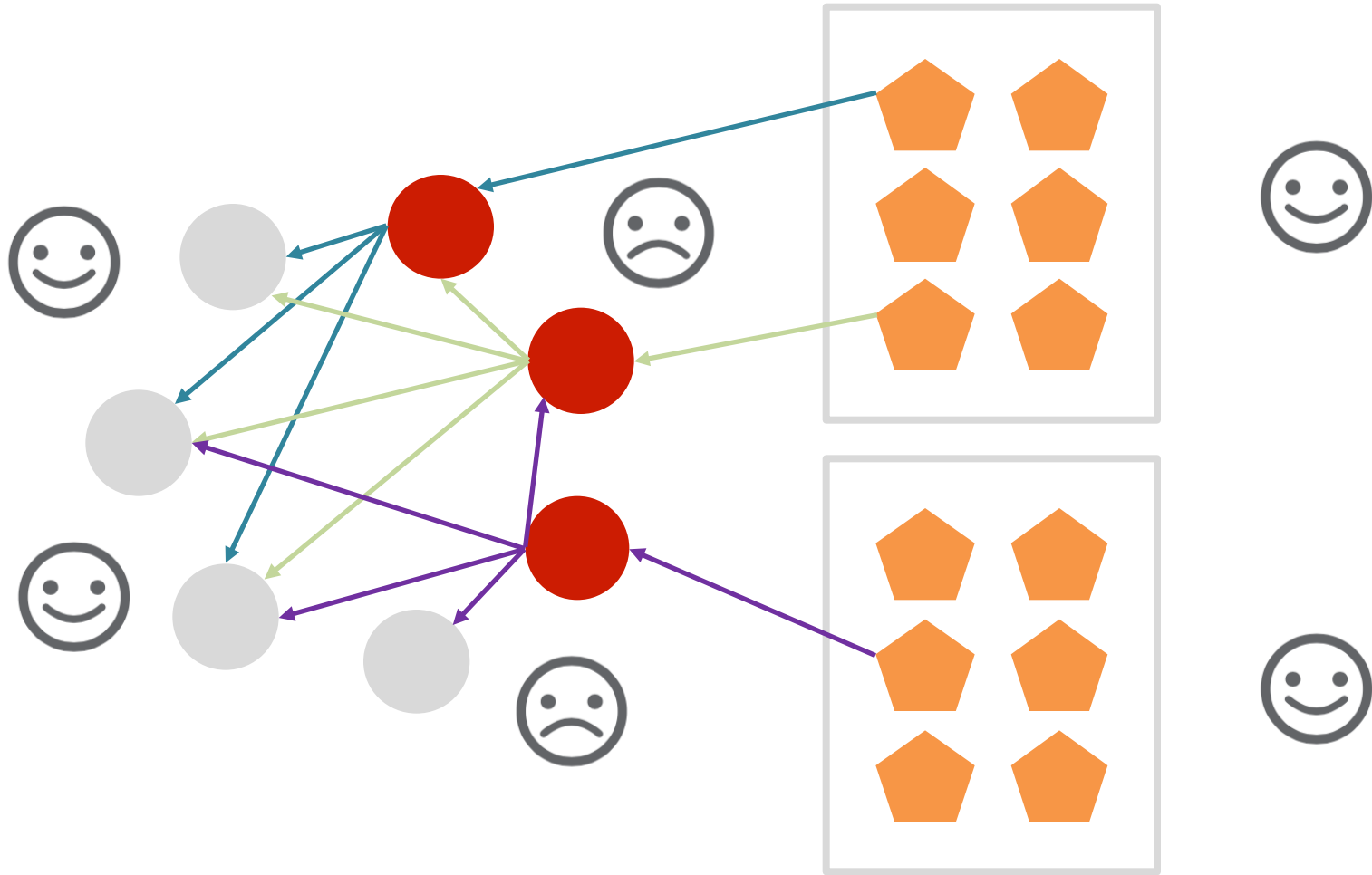
Using Secondary Index (2i)



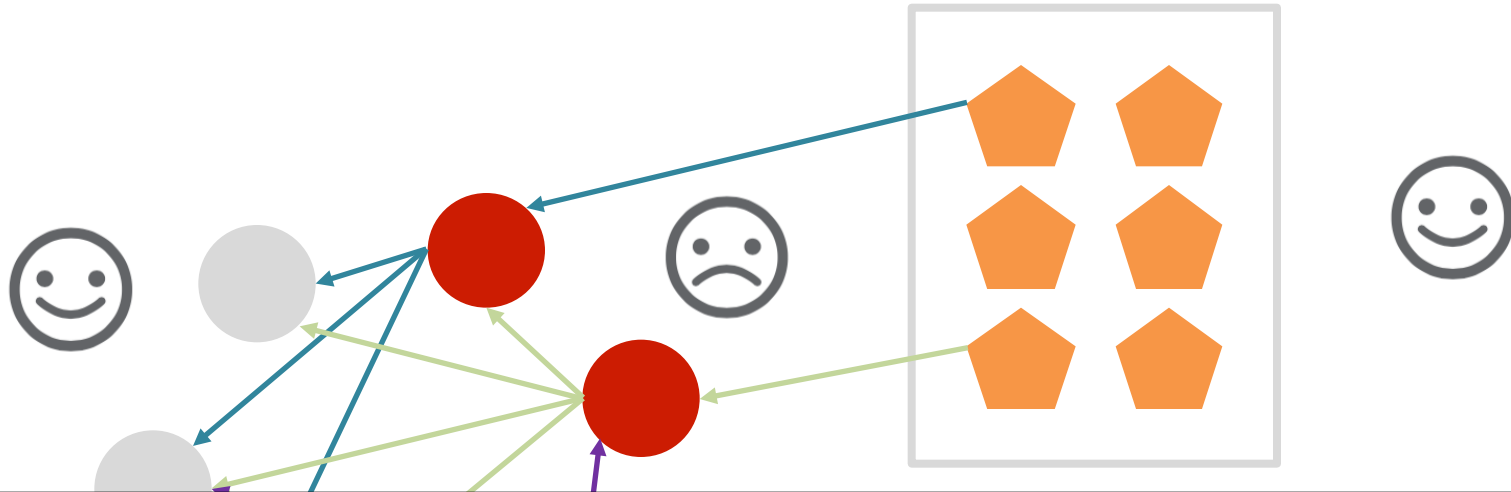
Using Secondary Index (2i)



Using Secondary Index (2i)



Using Secondary Index (2i)

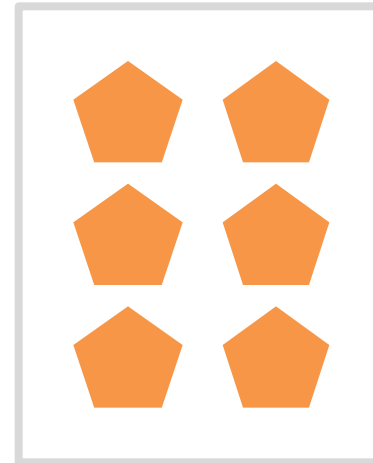
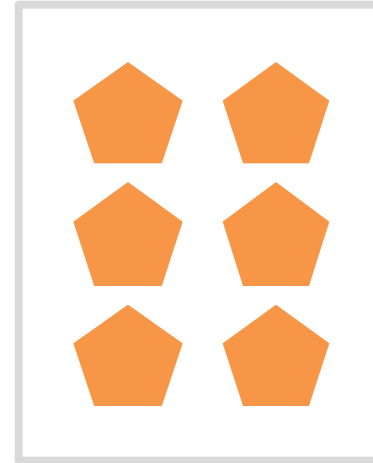
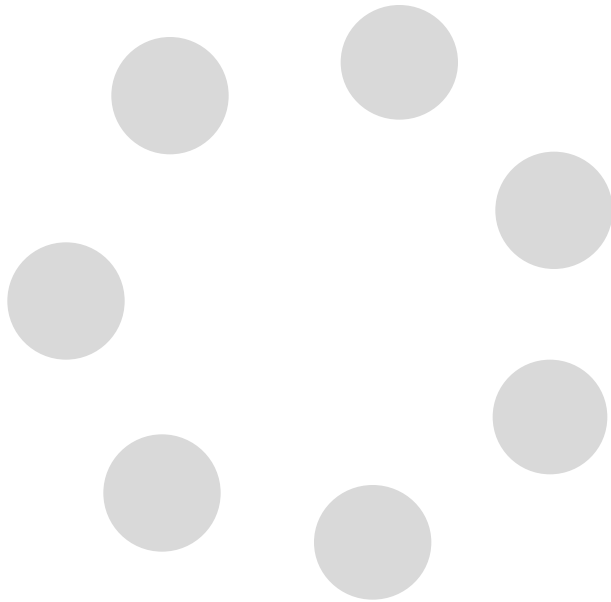


Lesson 1(b):

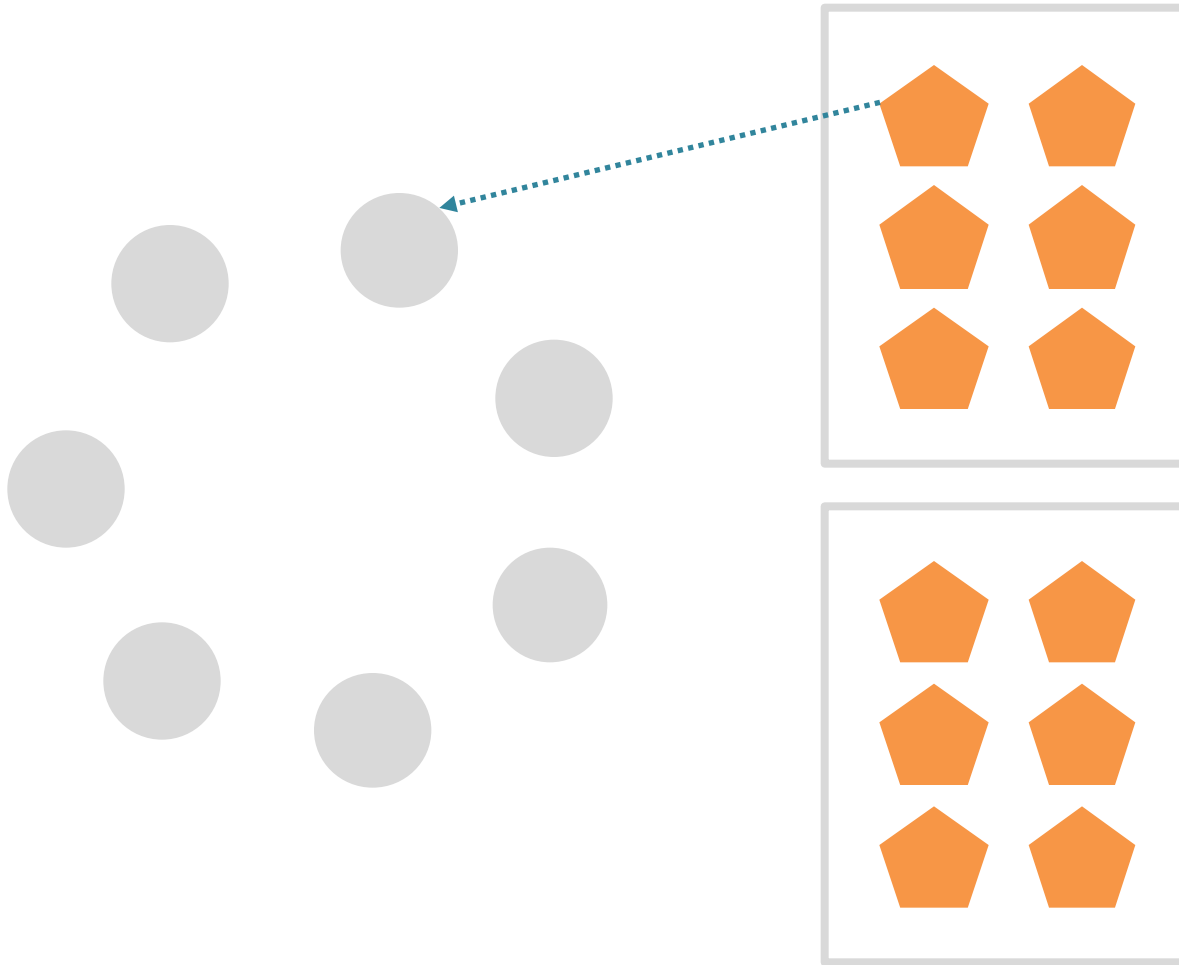
**Too many 2i queries make
Riak unhappy**



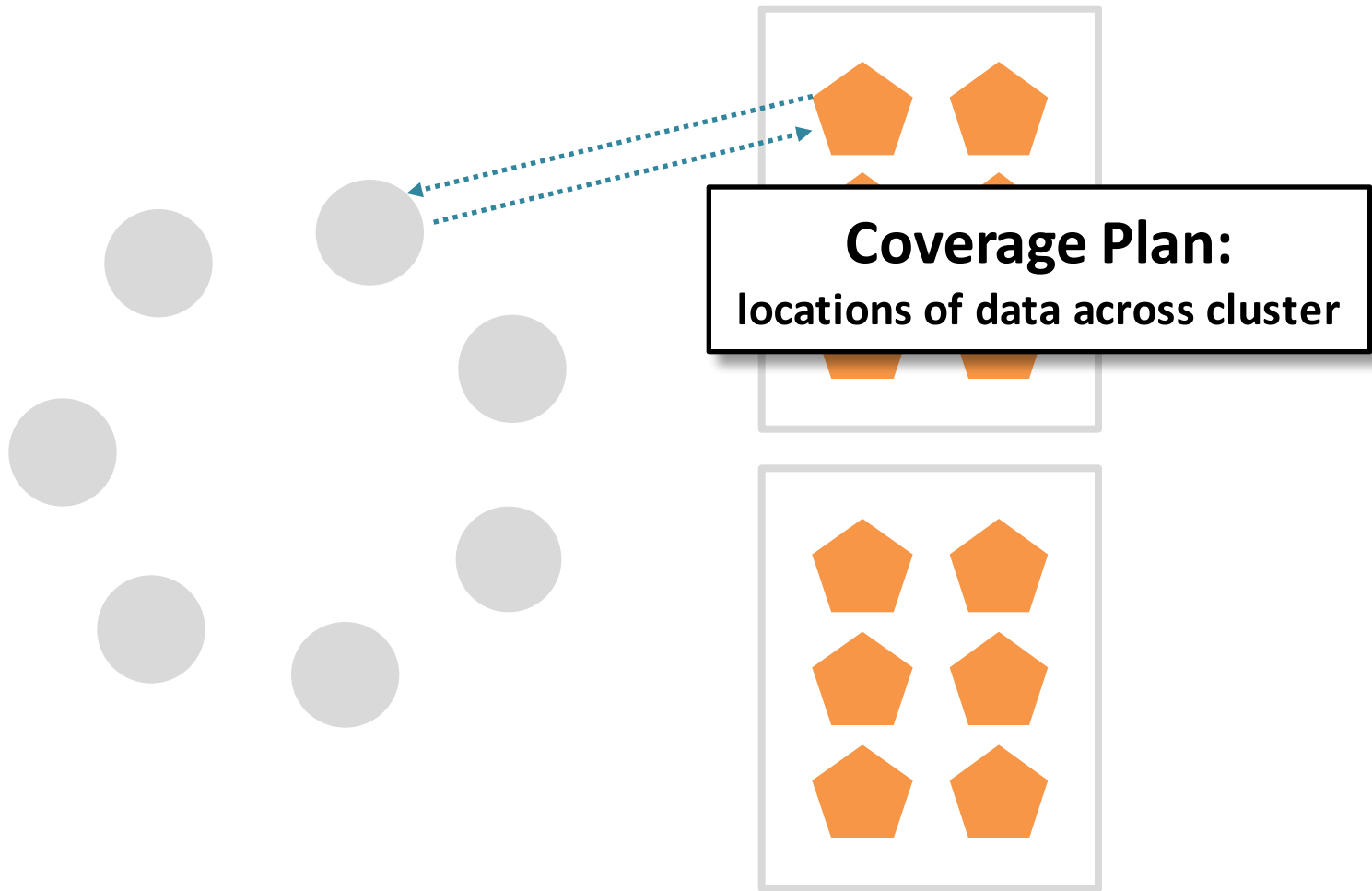
Coverage Plan + Parallel Extract



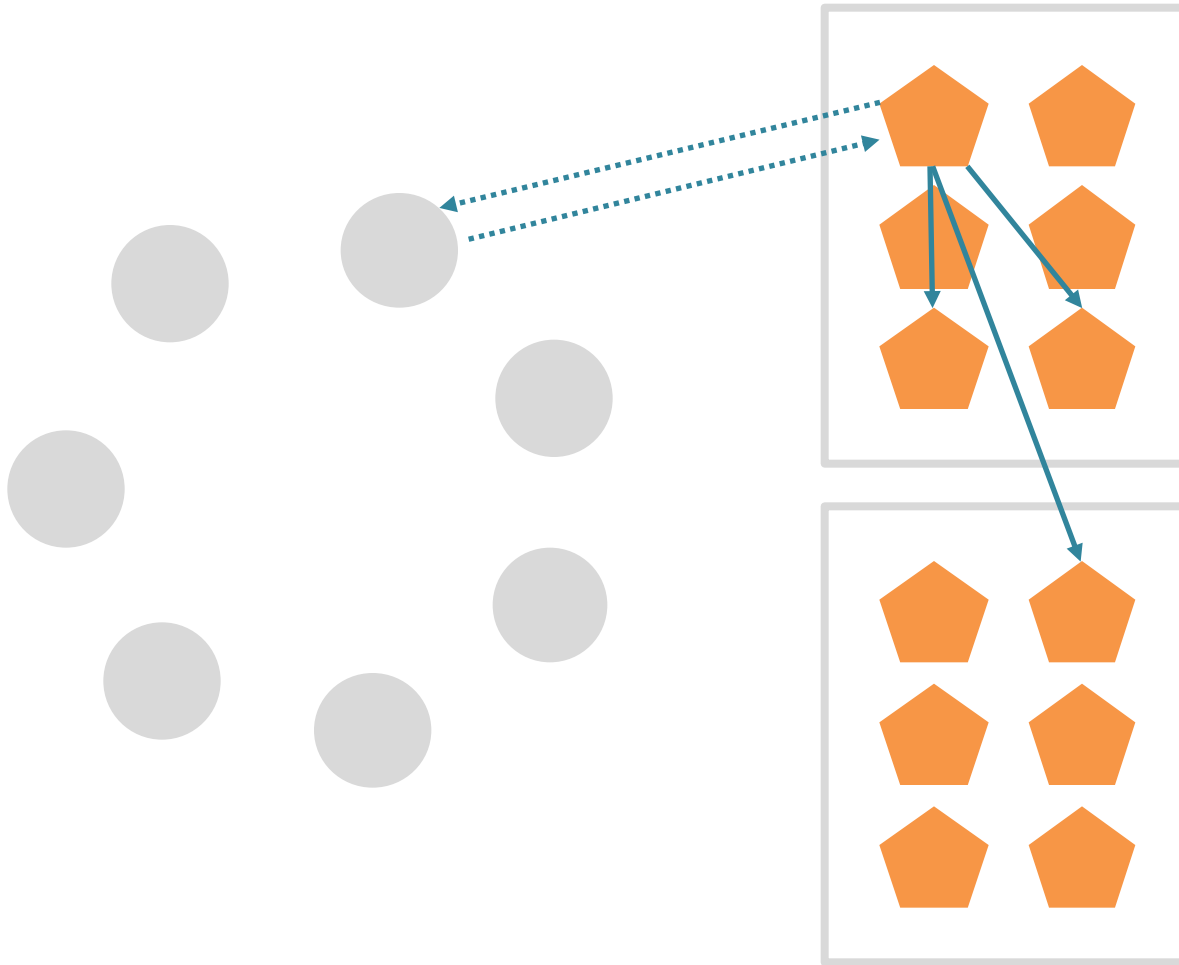
Coverage Plan + Parallel Extract



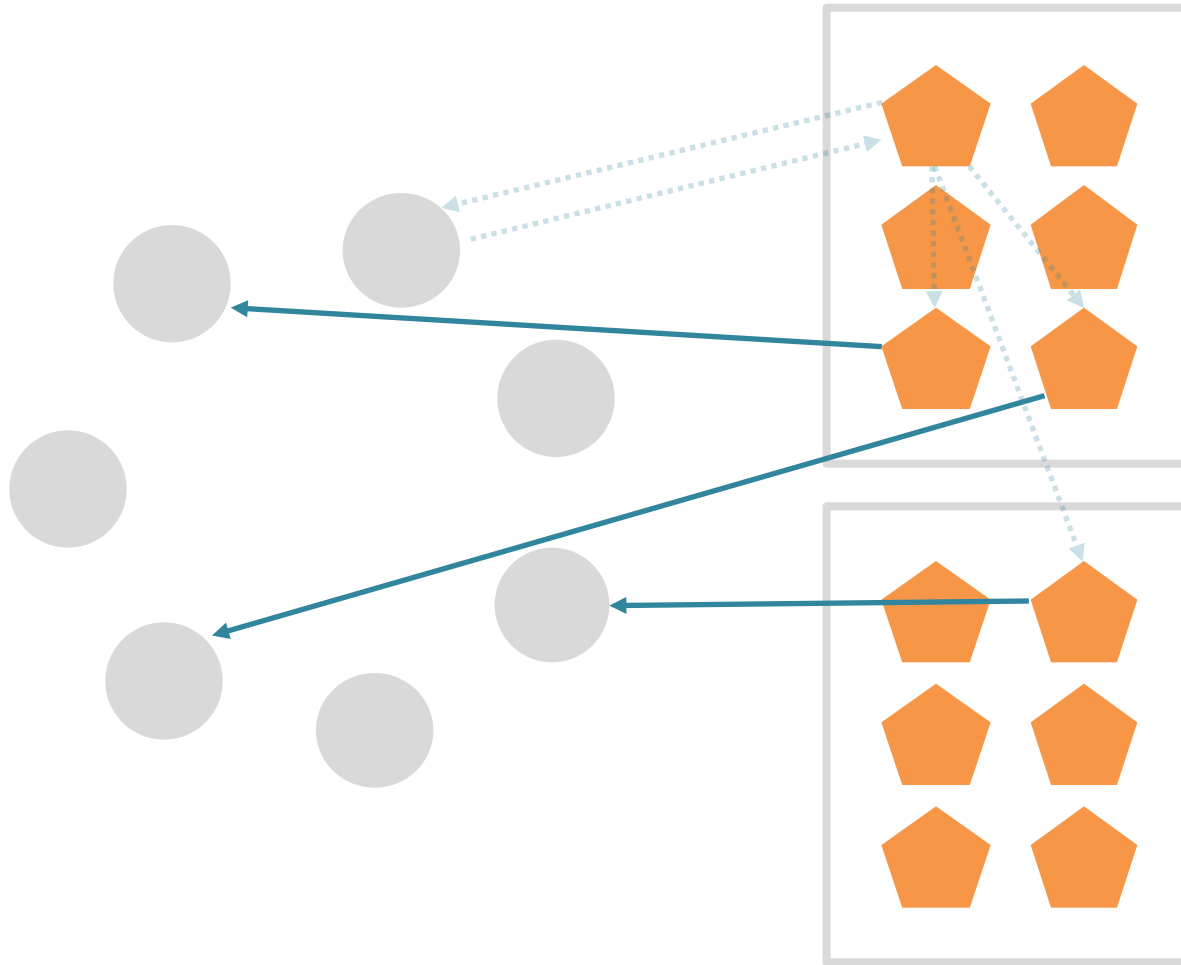
Coverage Plan + Parallel Extract



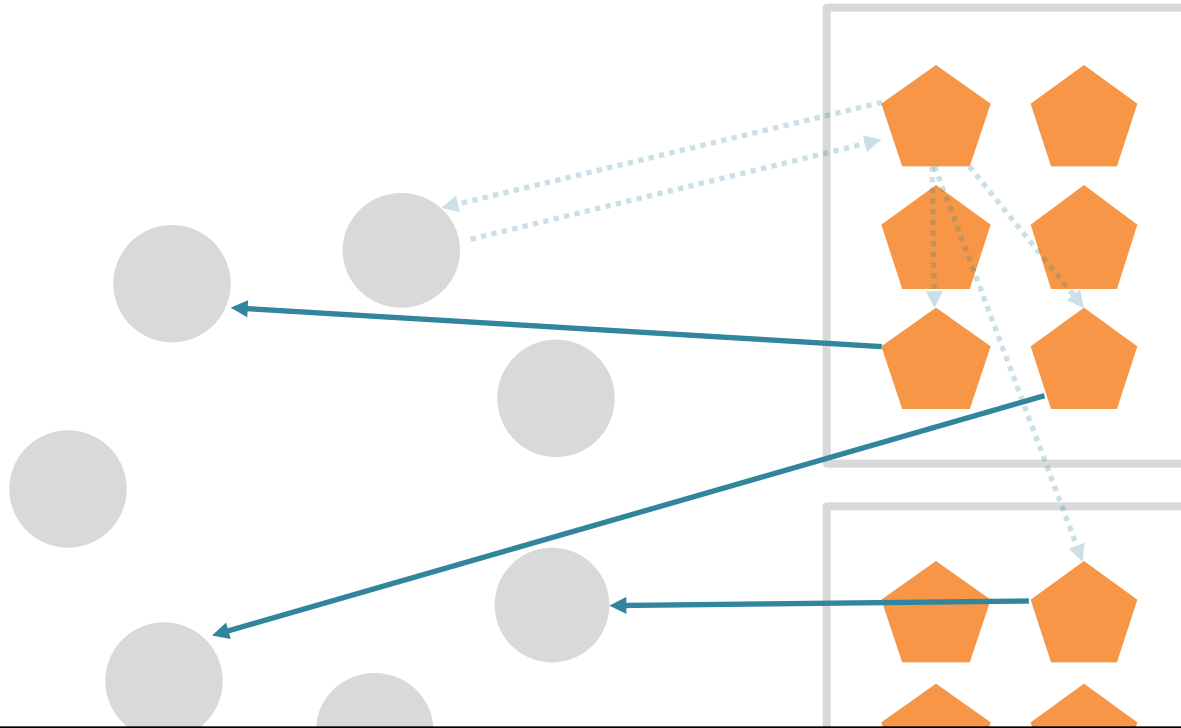
Coverage Plan + Parallel Extract



Coverage Plan + Parallel Extract



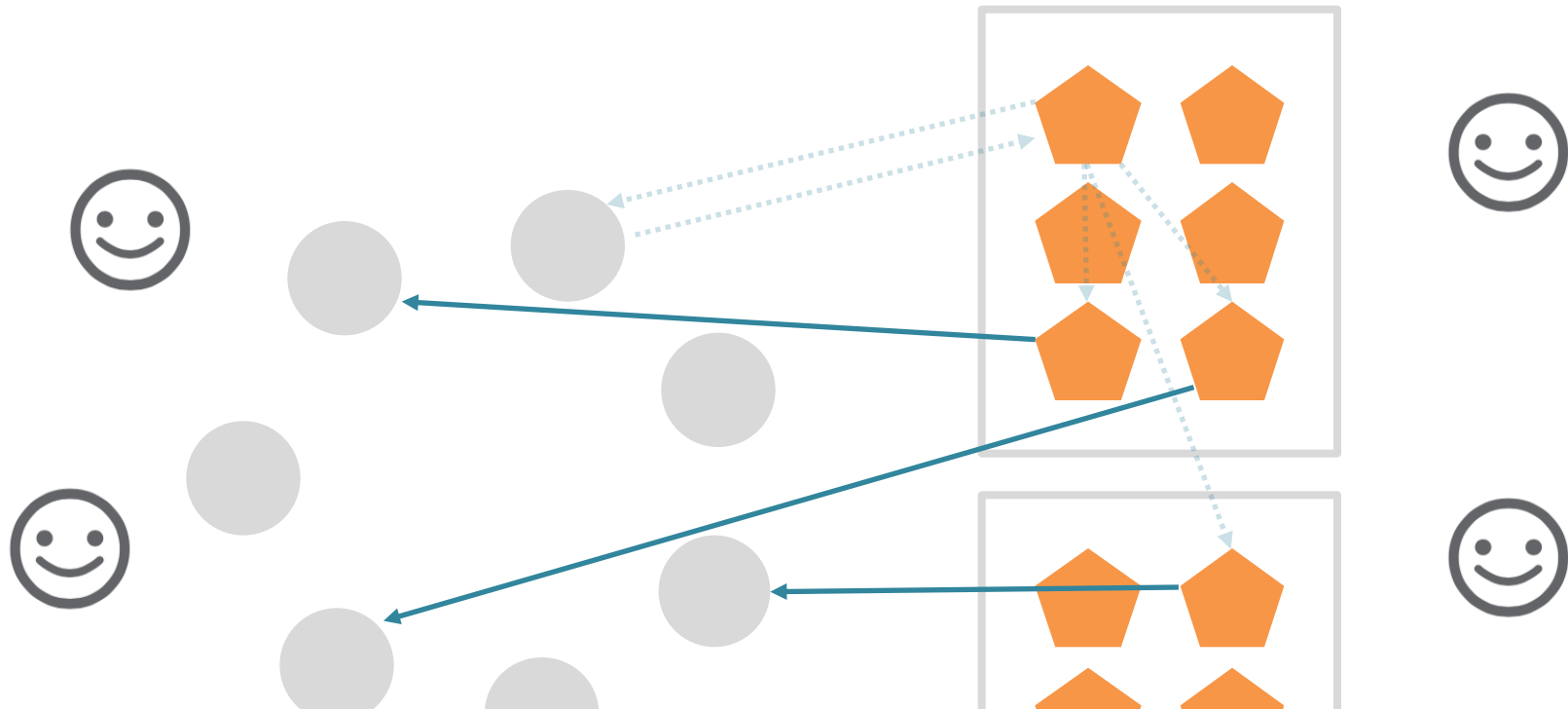
Coverage Plan + Parallel Extract



Parallelization

Coverage Plan + Parallel Extract

Coverage Plan + Parallel Extract



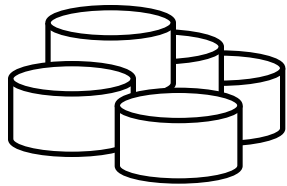
Parallelization

**Coverage Plan + Parallel Extract
= everybody's happy**

LESSON #2

**Be smart
about
data mapping**

Key-Value
& Time Series
Data

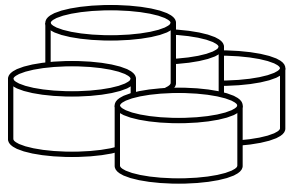
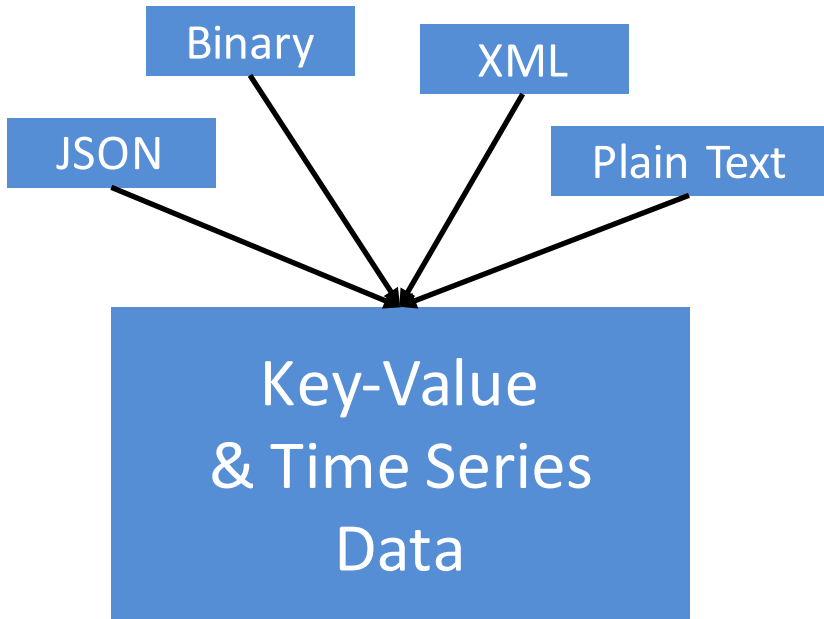


RDDs

DataFrames

DataSets





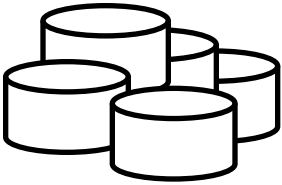
RDDs

DataFrames

DataSets



Key-Value
& Time Series
Data



RDDs

DataFrames

DataSets





Key-Value

RDDs

**How to map the data as
efficiently and seamlessly
as possible?**



BACKGROUND


How Riak Stores Data

Keys

Values

Buckets

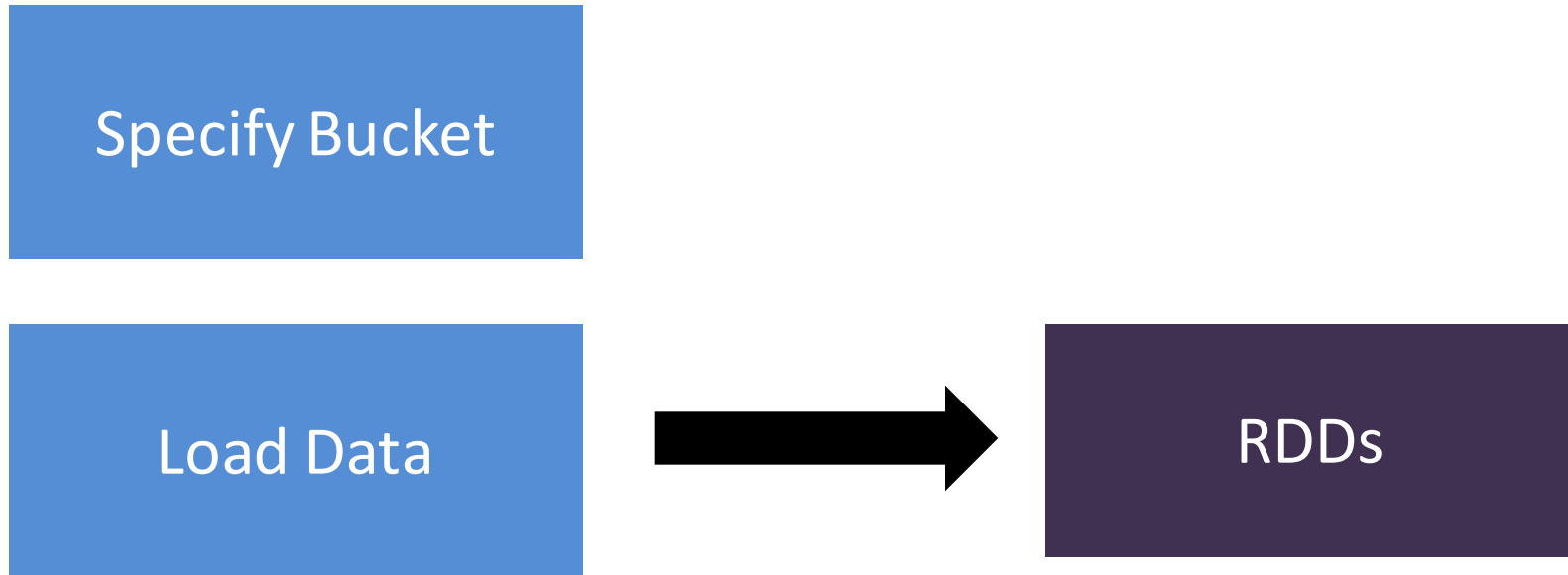
Bucket Types

Key	Value
User123	122883 dave ...
Item17Z	{ "color": "blue", "size": "small", ... }
LogoHD	

MiscBucket

Properties: "r": "quorum"

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")
```

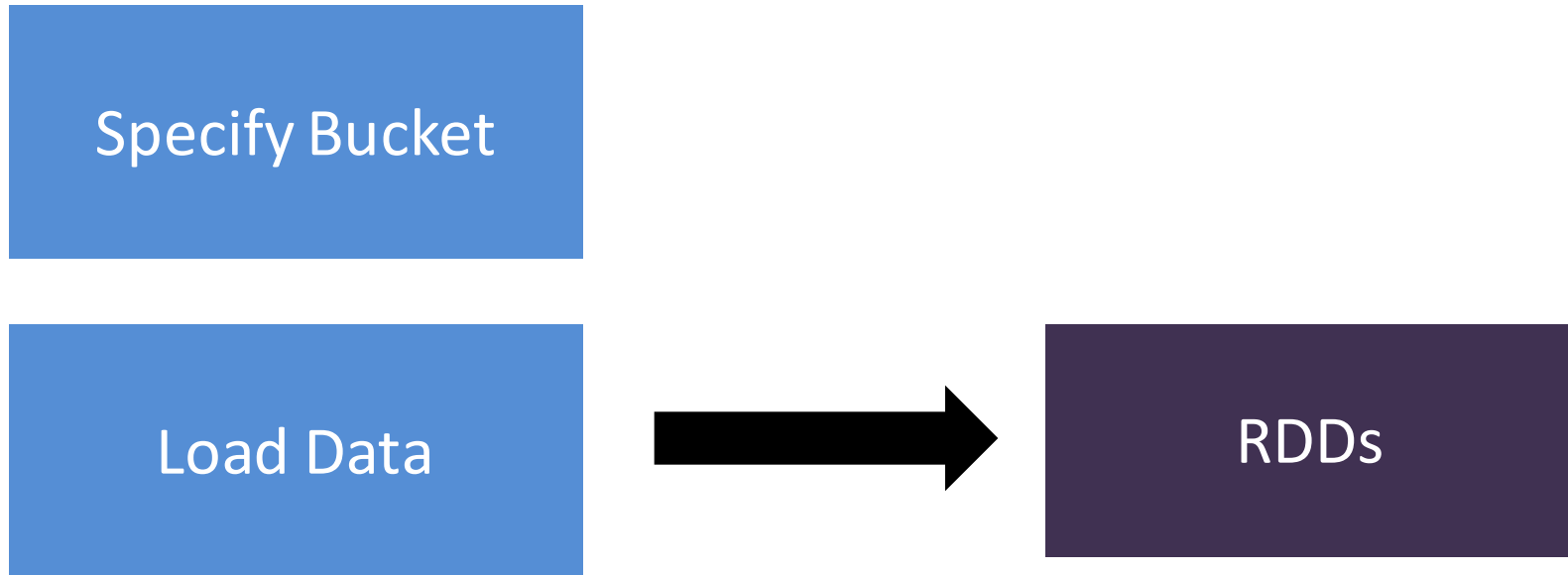
Query by 2i Range

```
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

Specify Bucket

Load Data



Often this data is stored as JSON

Key-Value Data

Specify Bucket

Map Schema

Load Data



We can tell Spark how to interpret the NoSQL values

Key-Value Data

Specify Bucket

Map Schema

Load Data



DataFrames

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

Specify Bucket

Map Schema

Load Data

Time Series Data

Specify Bucket

Map Schema

Load Data



But time series
data already has
a schema defined

Time Series Data

Specify Bucket

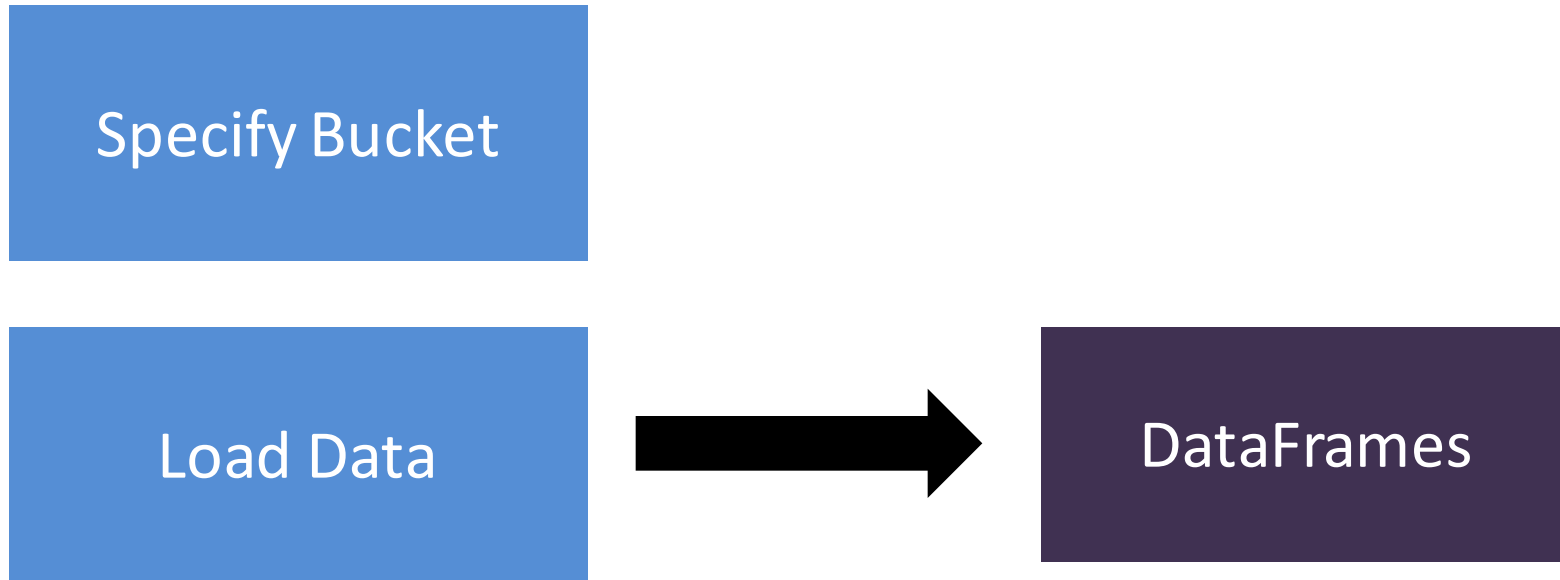
Map Schema

Load Data



So let's use
automatic schema
discovery instead

Time Series Data



Time Series Code

Specify Table

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

Load Data

```
df = sqlContext.read.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

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

Load Data

```
df = sqlContext.read.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)")
```

Use Data

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

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

Time Series Code

Specify Table

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

Uses the Spark
Data Source API



Load Data

```
df = sqlContext.read.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)")
```

Use Data

```
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

HTTP

Protocol Buffers

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

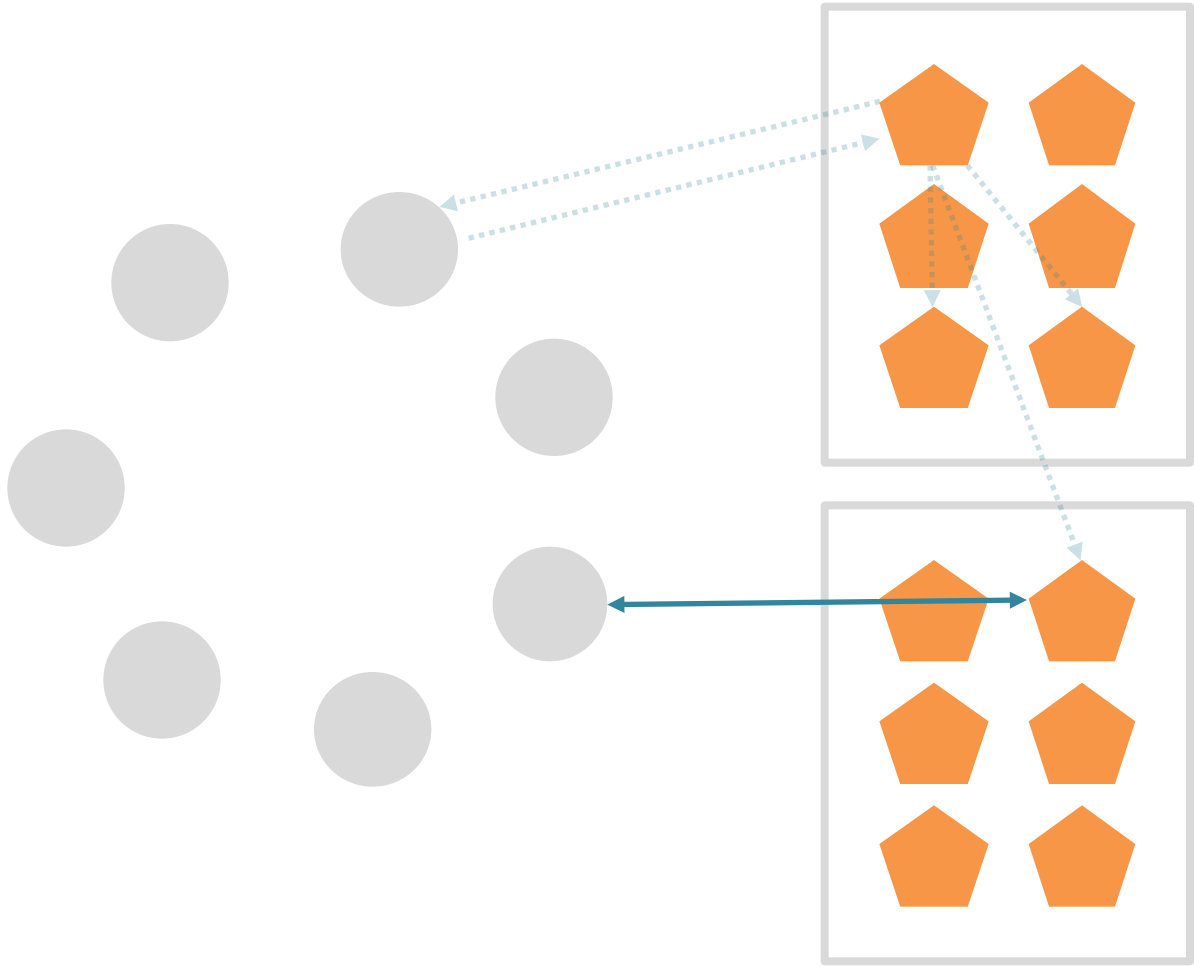
Note: In Riak, you typically don't have to know the details, the client SDKs take care of it for you

How much faster?

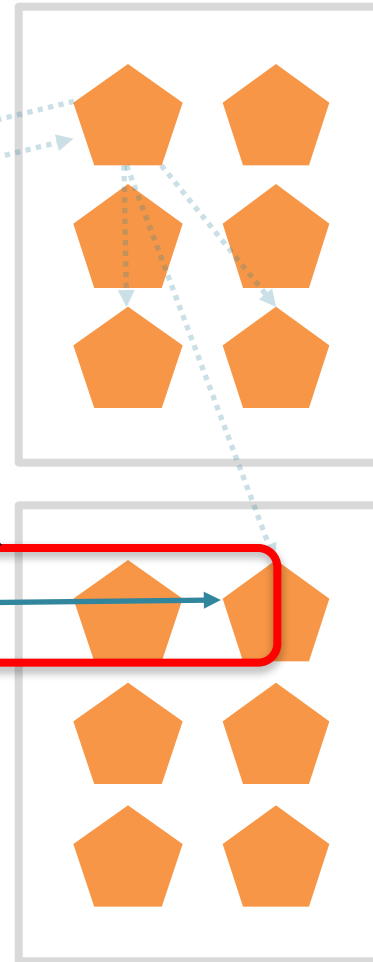
HTTP

Protocol Buffers

150-300%
faster



This interaction defaults to using Protocol Buffers to optimize performance





HTTP

Protocol Buffers

What if we can make this faster?

HTTP

Protocol Buffers

Optimized Binary

Spark-Riak Connector dynamically selects based on query type



HTTP

Protocol Buffers

Optimized Binary

**Other
Operations**



Protocol Buffers

**Bulk TS
Operations**



Optimized Binary

**Fetch
Query
Store**

**Other
Operations**



Protocol Buffers

**Bulk TS
Operations**

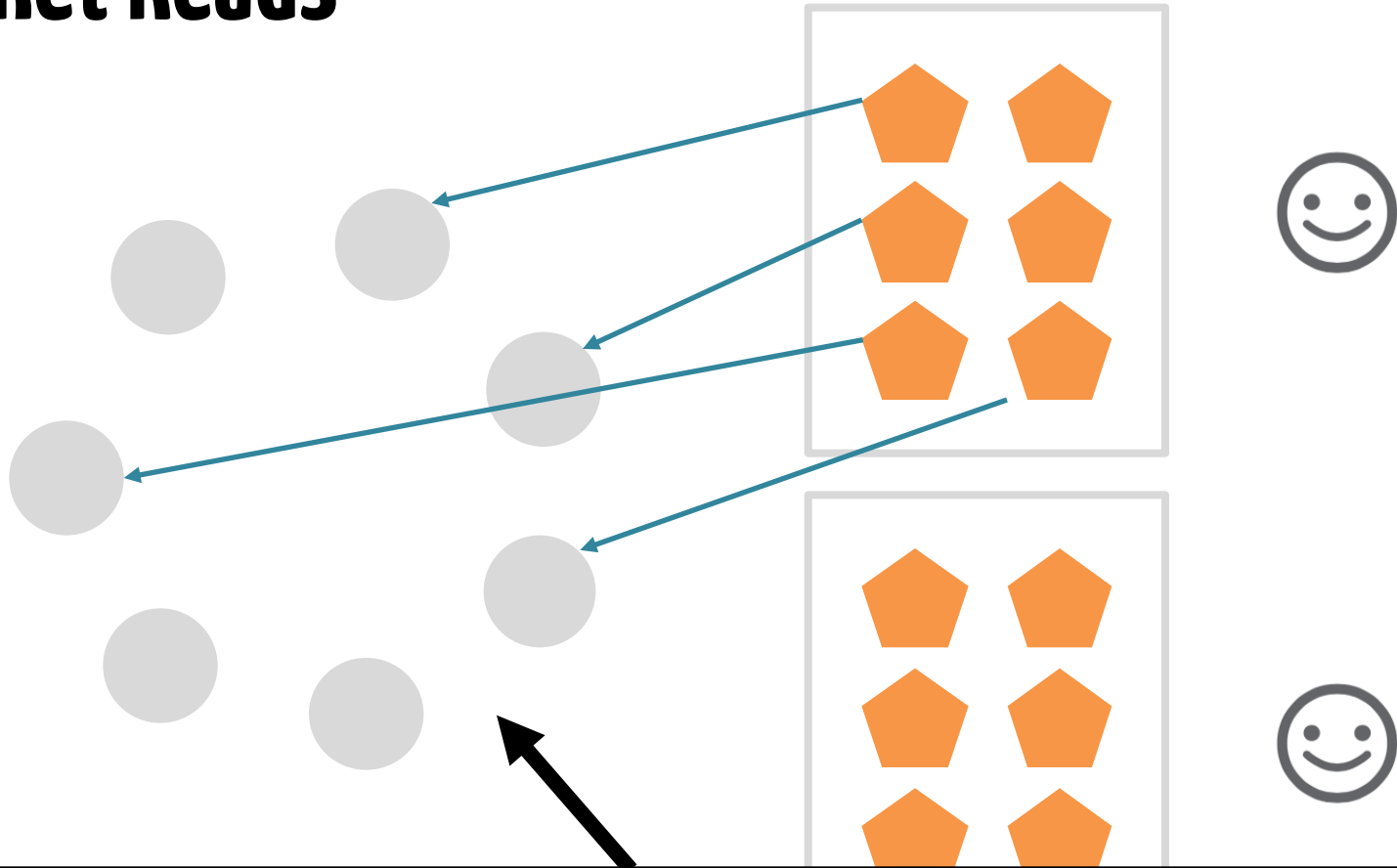


Optimized Binary

**30-50%
increased
throughput**

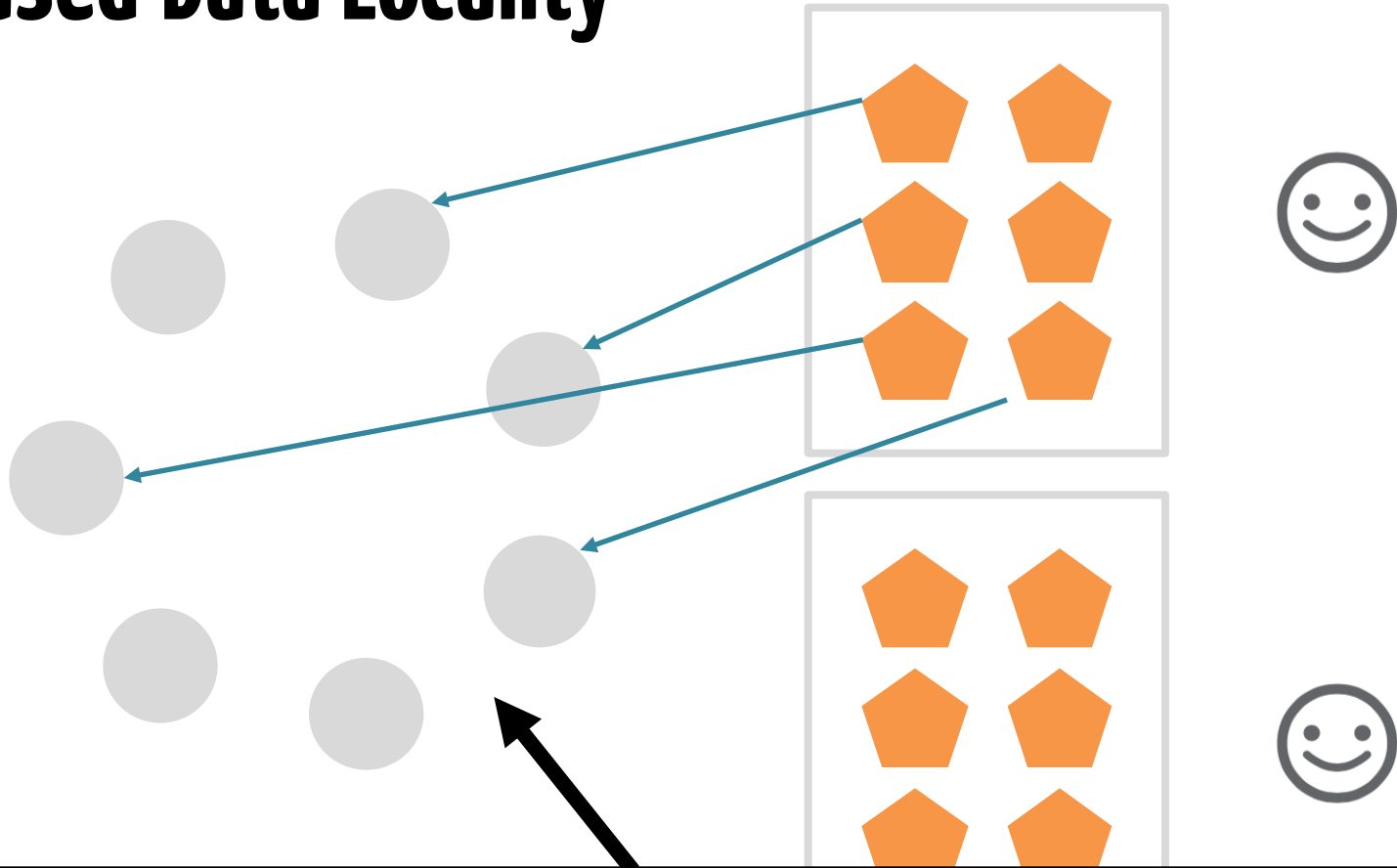
**2 use case-specific
optimizations**

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**

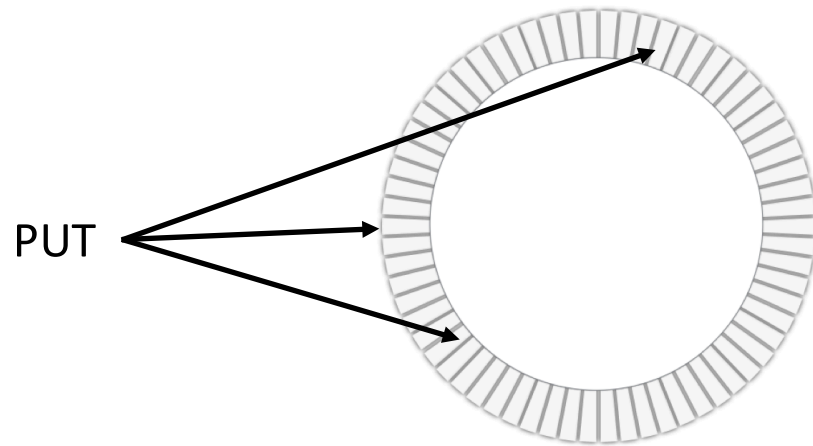
Time-based Data Locality



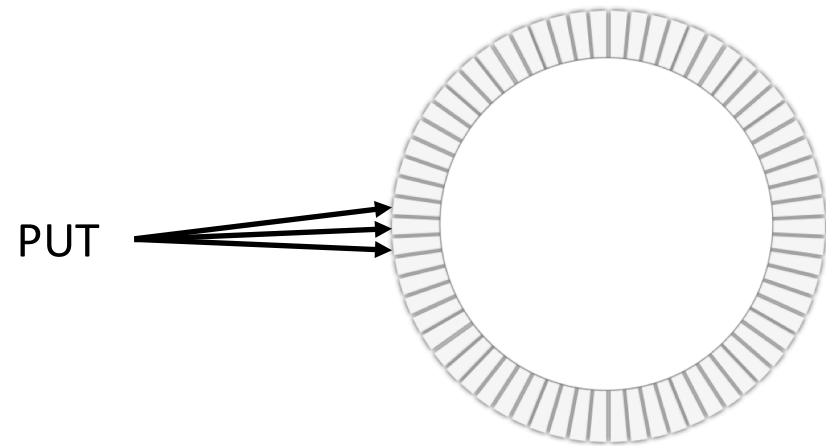
Riak TS uses a time based 'quanta' to intelligently partition data across the cluster based on user-specified time

Location, location, location

Key/Value cluster vnodes



Time Series cluster vnodes

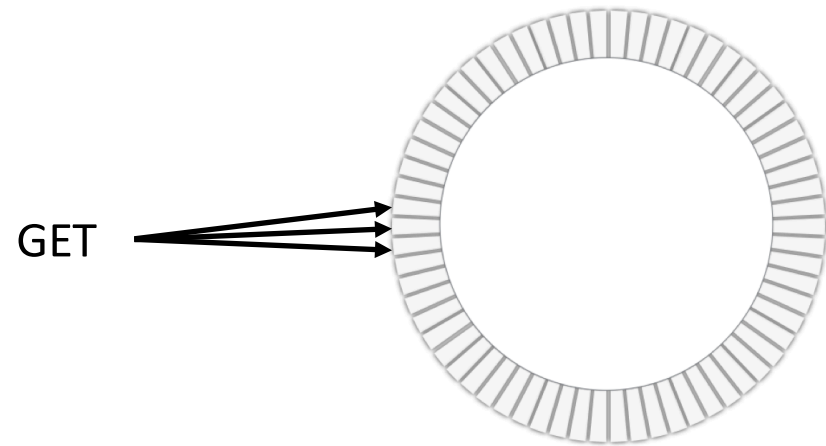
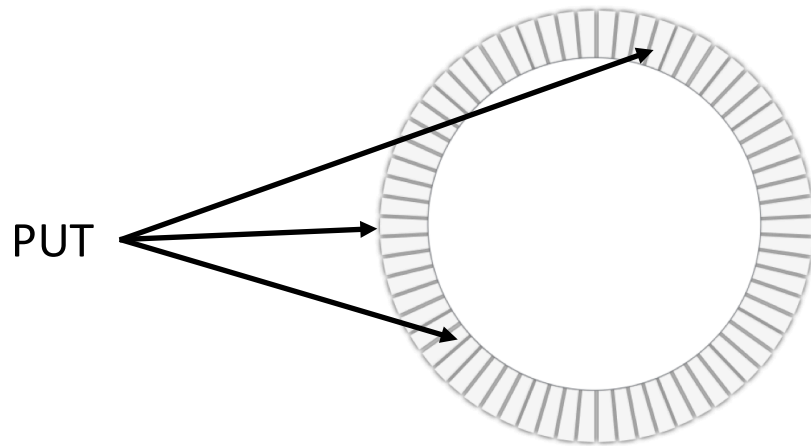


Local grouping based on time quanta
Write to same vnode
Query direct to data

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

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 = 'New York City'  
      time >= '2016-01-01' AND  
      time <= '2016-02-01 00:00:00'
```

LESSON #4

Be

flexible

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

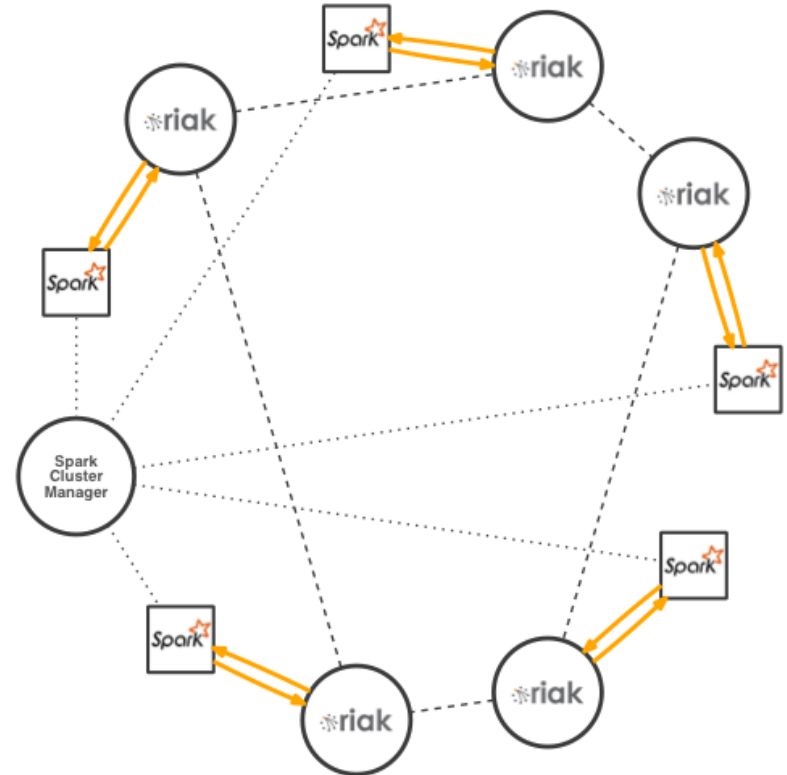
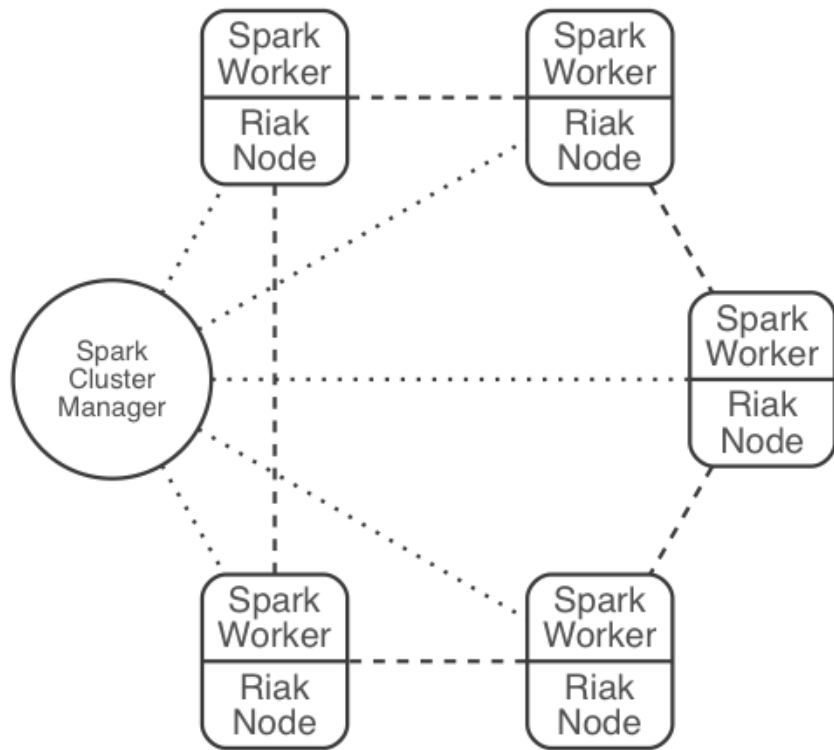
On premise

Cloud

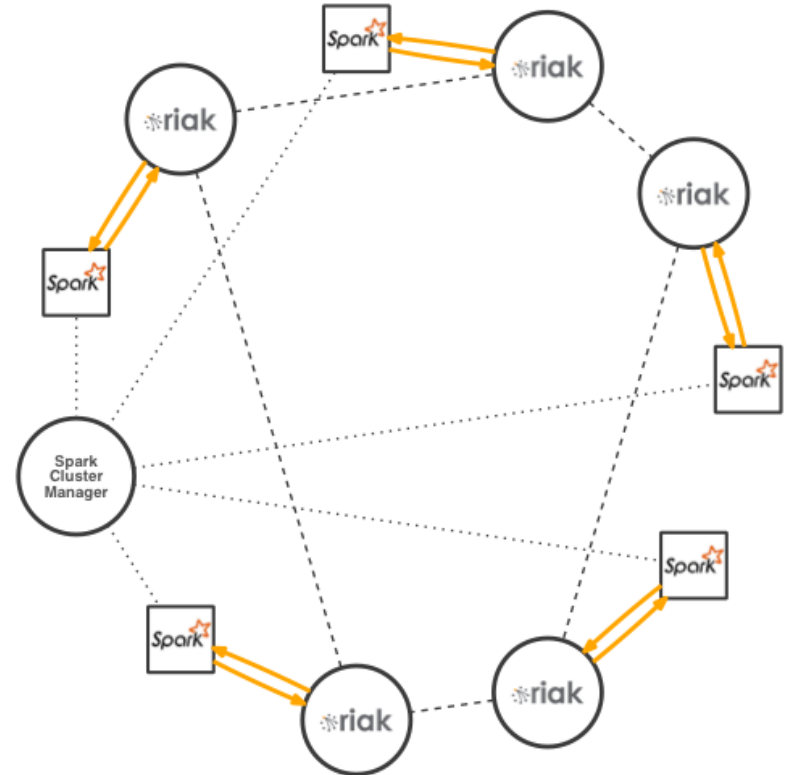
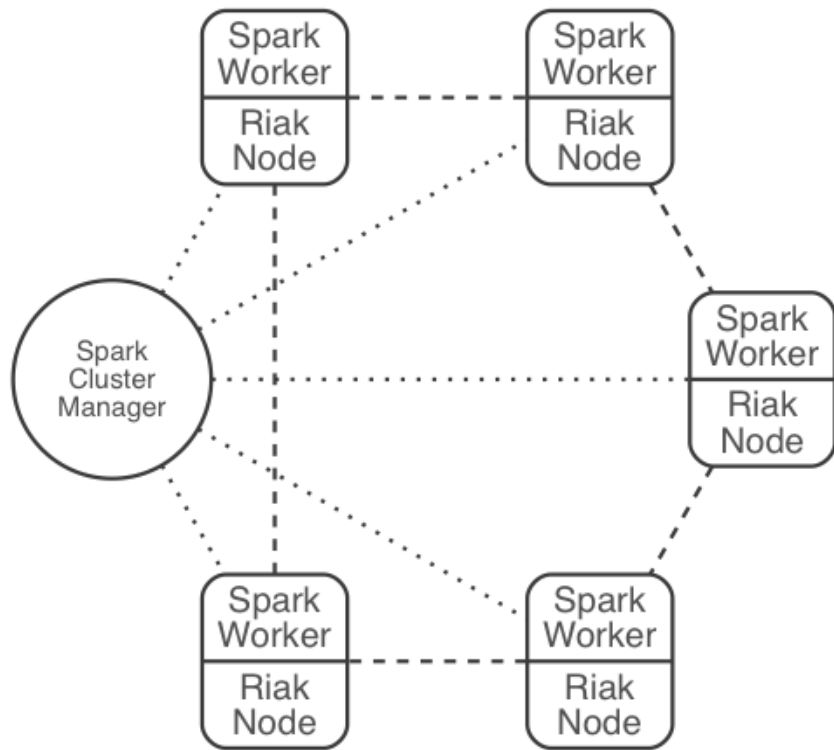
Hybrid

Geo-distributed

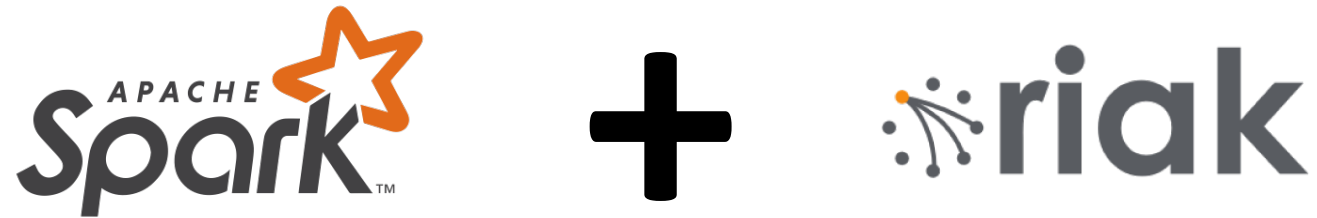
Deployment



Deployment



Deployment



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

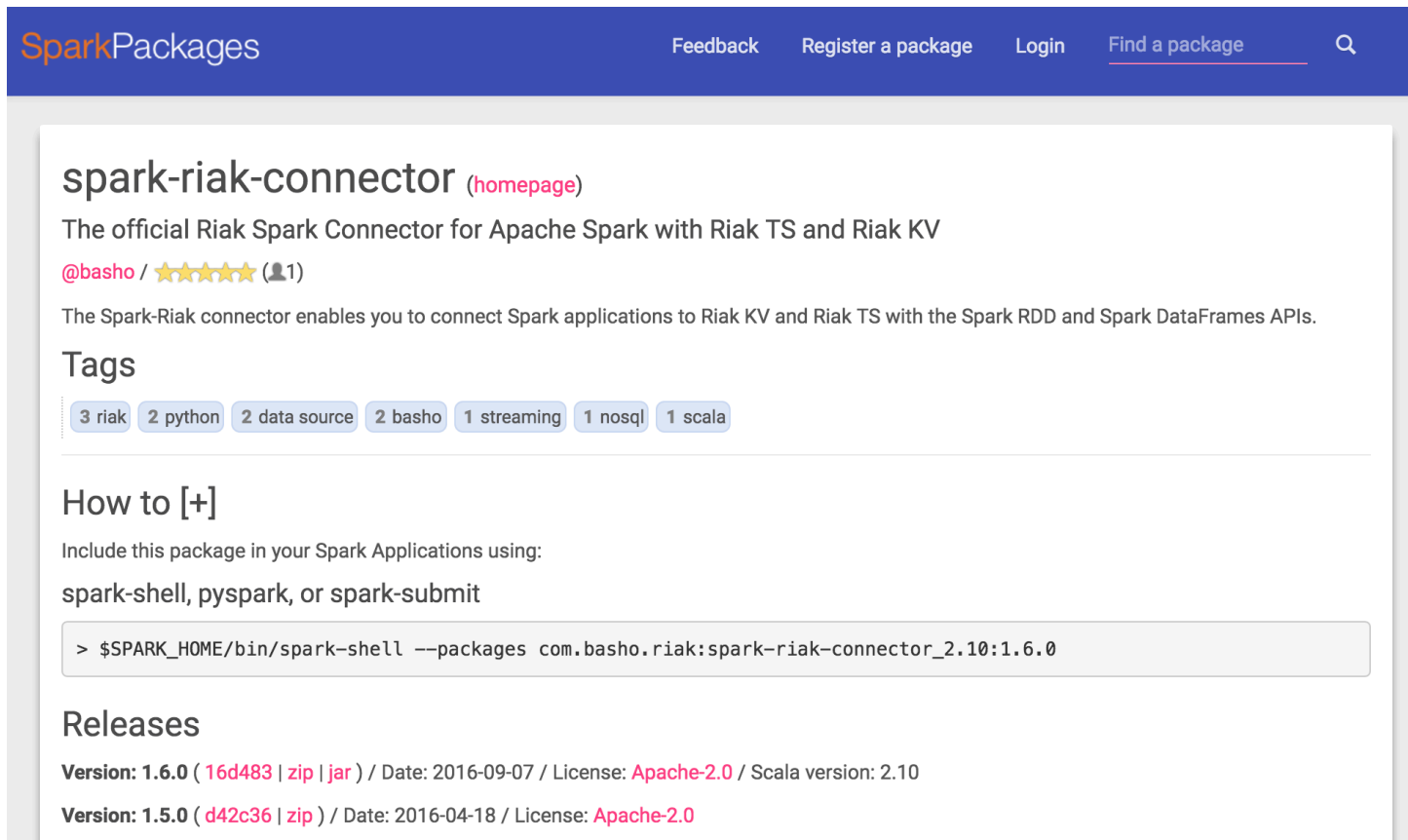
LESSON #5

Simplify

LESSON #5

**How to reduce
friction?**

Connector hosted at Spark-Packages



The screenshot shows the Spark-Packages website interface. At the top, there is a blue navigation bar with the Spark-Packages logo on the left and links for Feedback, Register a package, Login, Find a package, and a search icon on the right. The main content area features the package name 'spark-riak-connector' with a link to its homepage. Below the name is a description: 'The official Riak Spark Connector for Apache Spark with Riak TS and Riak KV'. It also shows the maintainer '@basho' with a 5-star rating and 1 user. A paragraph describes the connector's purpose: 'The Spark-Riak connector enables you to connect Spark applications to Riak KV and Riak TS with the Spark RDD and Spark DataFrames APIs.' A 'Tags' section lists categories like 'riak', 'python', 'data source', 'basho', 'streaming', 'nosql', and 'scala' with their respective counts. A 'How to [+]' section provides instructions on including the package in Spark applications, listing 'spark-shell', 'pyspark', and 'spark-submit'. A code block shows the command: '> \$SPARK_HOME/bin/spark-shell --packages com.basho.riak:spark-riak-connector_2.10:1.6.0'. Finally, a 'Releases' section lists two versions: 1.6.0 (dated 2016-09-07) and 1.5.0 (dated 2016-04-18), both licensed under Apache-2.0.

SparkPackages

Feedback Register a package Login Find a package

spark-riak-connector (homepage)

The official Riak Spark Connector for Apache Spark with Riak TS and Riak KV

@basho / ★★★★★ (1)

The Spark-Riak connector enables you to connect Spark applications to Riak KV and Riak TS with the Spark RDD and Spark DataFrames APIs.

Tags

3 riak 2 python 2 data source 2 basho 1 streaming 1 nosql 1 scala

How to [+]

Include this package in your Spark Applications using:

spark-shell, pyspark, or spark-submit

```
> $SPARK_HOME/bin/spark-shell --packages com.basho.riak:spark-riak-connector_2.10:1.6.0
```

Releases

Version: 1.6.0 (16d483 | zip | jar) / Date: 2016-09-07 / License: Apache-2.0 / Scala version: 2.10

Version: 1.5.0 (d42c36 | zip) / Date: 2016-04-18 / License: Apache-2.0

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

Tutorial notebook on Databricks.com



[LZO Files - py](#)

[LZO Files - scala](#)

Databases & Other Data

Sources

[Importing HIVE Tables](#)

[JDBC for SQL Databases](#)

[Redshift](#)

[Cassandra](#)

[ElasticSearch](#)

[Redis](#)

[Dataframe JDBC API](#)

[Examples - py](#)

[Dataframe JDBC API](#)

[Examples - scala](#)

[RiakTS Tutorial](#)

Databricks Public Datasets

[DBFS Hosted Datasets](#)

[Other Public Datasets](#)

[R Dataset Util](#)

[R Dataset Util - Example](#)

SQL, DataFrames & Datasets

[Spark SQL](#)

[Introduction to Datasets](#)

RiakTS Tutorial



This is a tutorial using Basho's new Time Series database, Riak TS, with Apache Spark. The Databricks platform makes it easy to connect Spark jobs to Riak TS clusters running in private or public clouds or in your data center.

You'll find out how to connect Spark to Riak TS, how to write and read data using Spark DataFrames, and how to use several features of Riak TS.

This demo assumes that you have at least one Riak TS node running (developer environment), although we recommend having a 3 node cluster in production. There are many way to set up a Riak TS cluster. One of the simplest ways is to use Amazon Web Services and follow these instructions on creating a cluster using the Riak TS AMI: <https://docs.basho.com/riak/ts/latest/installing/aws/>

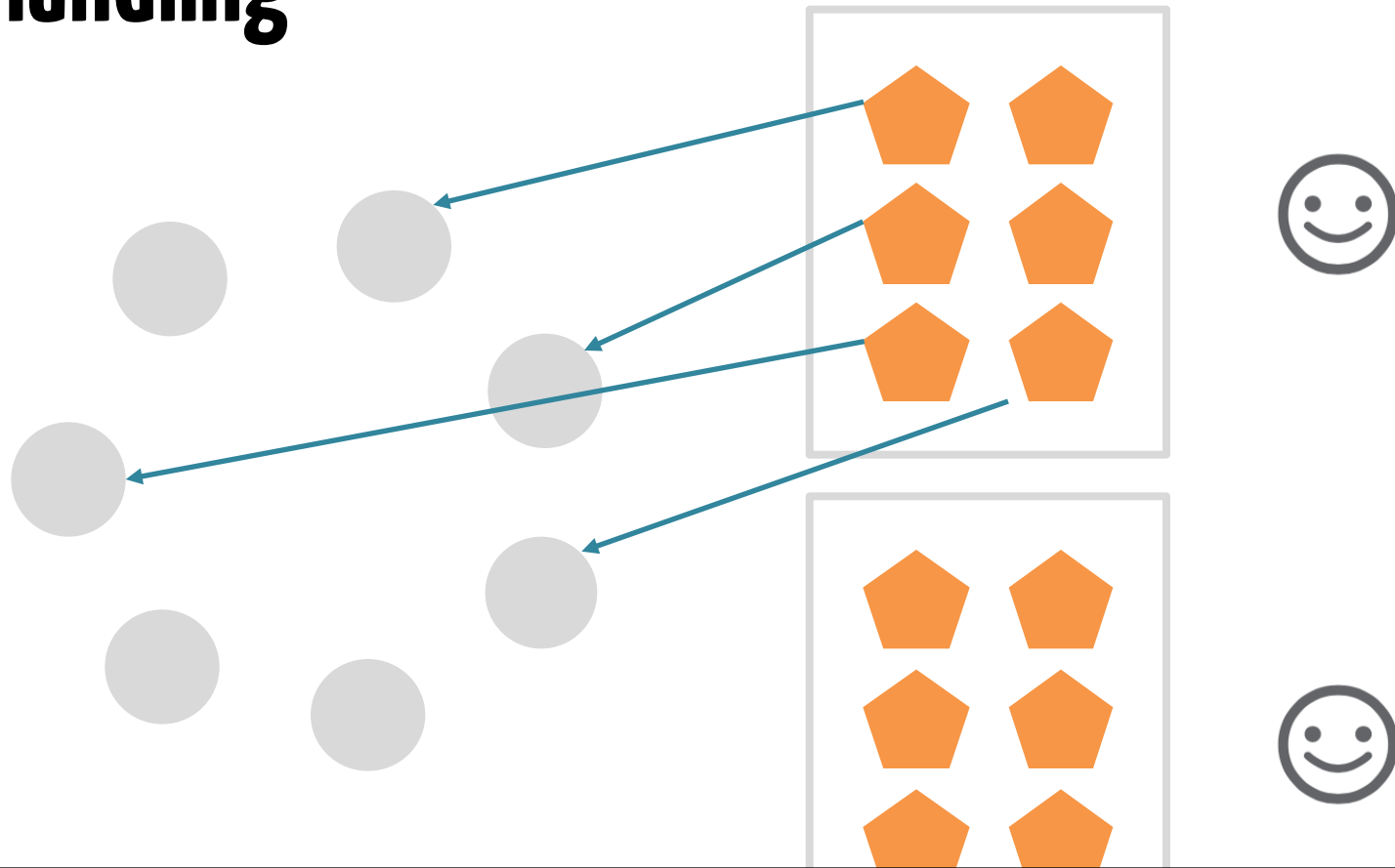
To find out more about Riak TS, go to:

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?

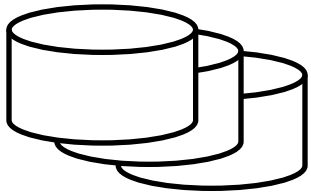
Spark 2.0

DataSets

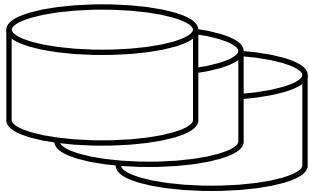
Structured Streaming

So, back to the question...

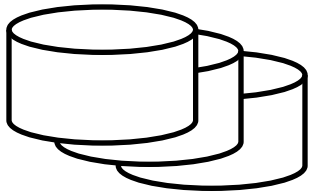
NoSQL +  = ?



NoSQL +  = 



NoSQL +  = 



LESSONS

Parallelize

Map smart

Optimize all the levels

Be flexible

Simplify

Thank You



We're hiring!

@johnmusser

@basho



Photo Credits

Race car: Spacesuit Media

Intellicore application screenshots: Intellicore, <http://www.intellicore.tv/>