Running Spark in Production in the Cloud is not easy

Nayur Khan

#SAISEnt12

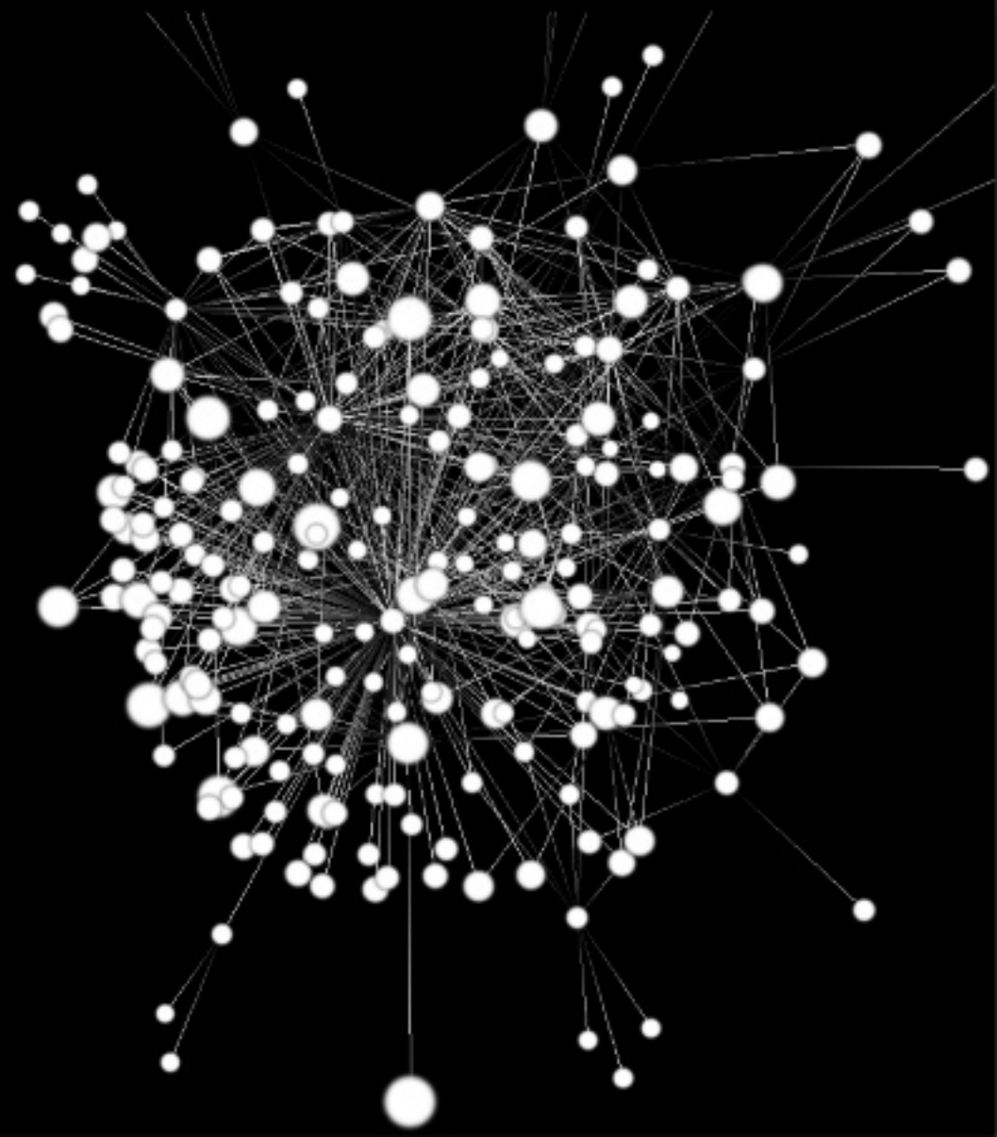
QUANTUMBLACK
A MCKINSEY COMPANY

Who am I?

Nayur Khan

Head of Platform Engineering

QUANTUMBLACK
A MCKINSEY COMPANY

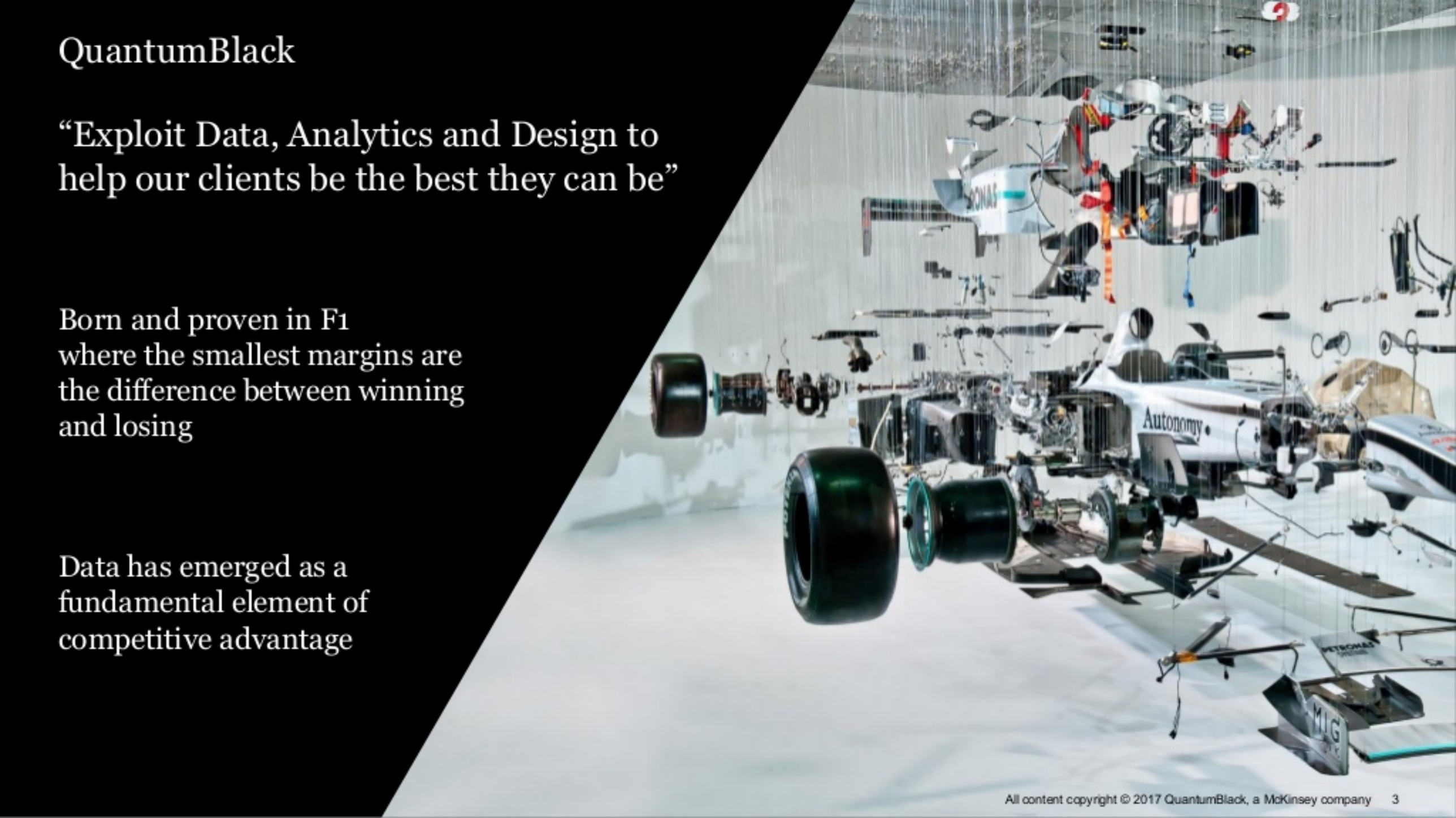


QuantumBlack

"Exploit Data, Analytics and Design to help our clients be the best they can be"

Born and proven in F1 where the smallest margins are the difference between winning and losing

Data has emerged as a fundamental element of competitive advantage



Not just Formula One...

Advanced Industries

Financial Services

Healthcare

Infra-structure

Telecoms

Natural Resources

Sport

Consumer

1 – Background

2 – Observations of using Cloud Object Storage

3 – Design To Operate

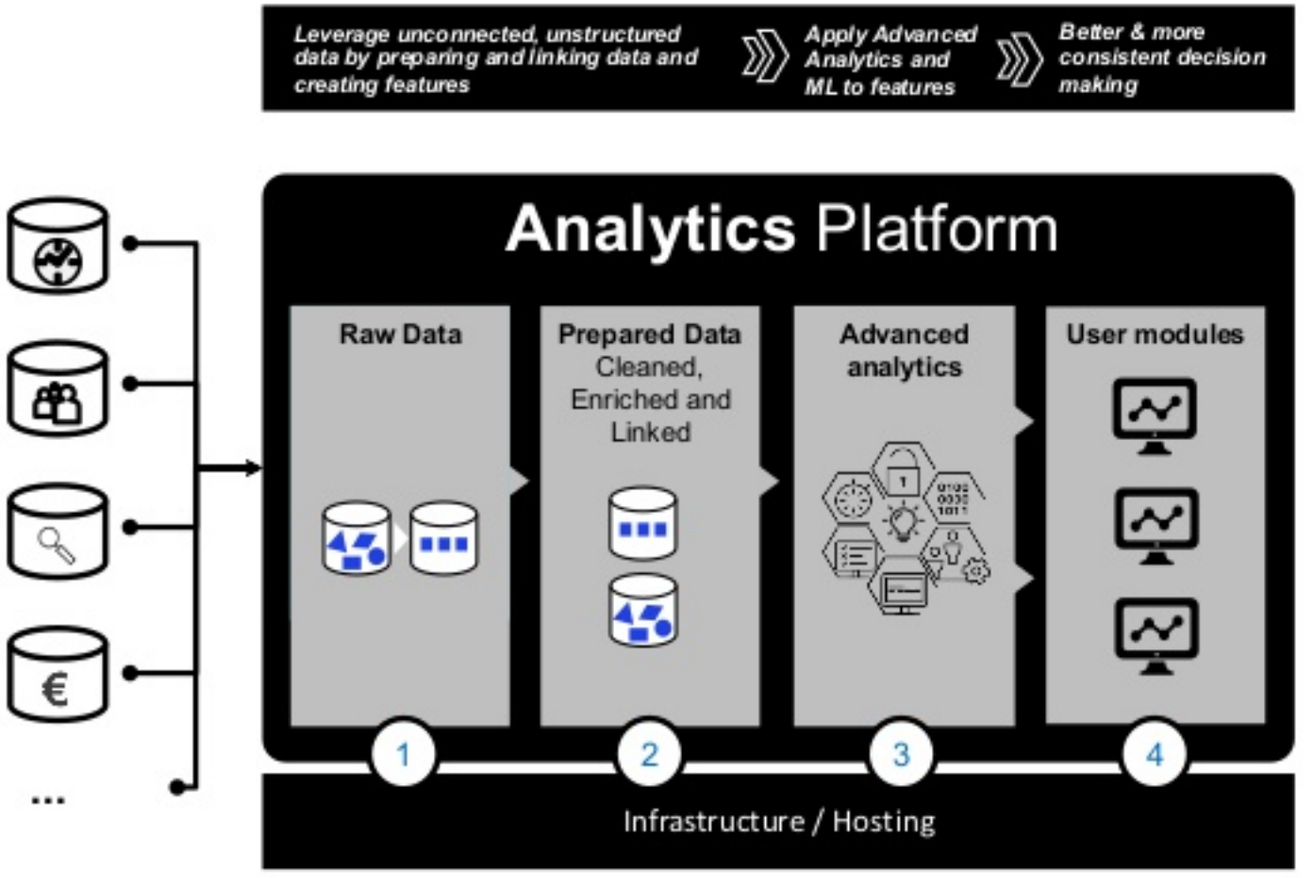
4 – Pipelines!

Background

1

Some of the Platforms we use

Typical Flow for production



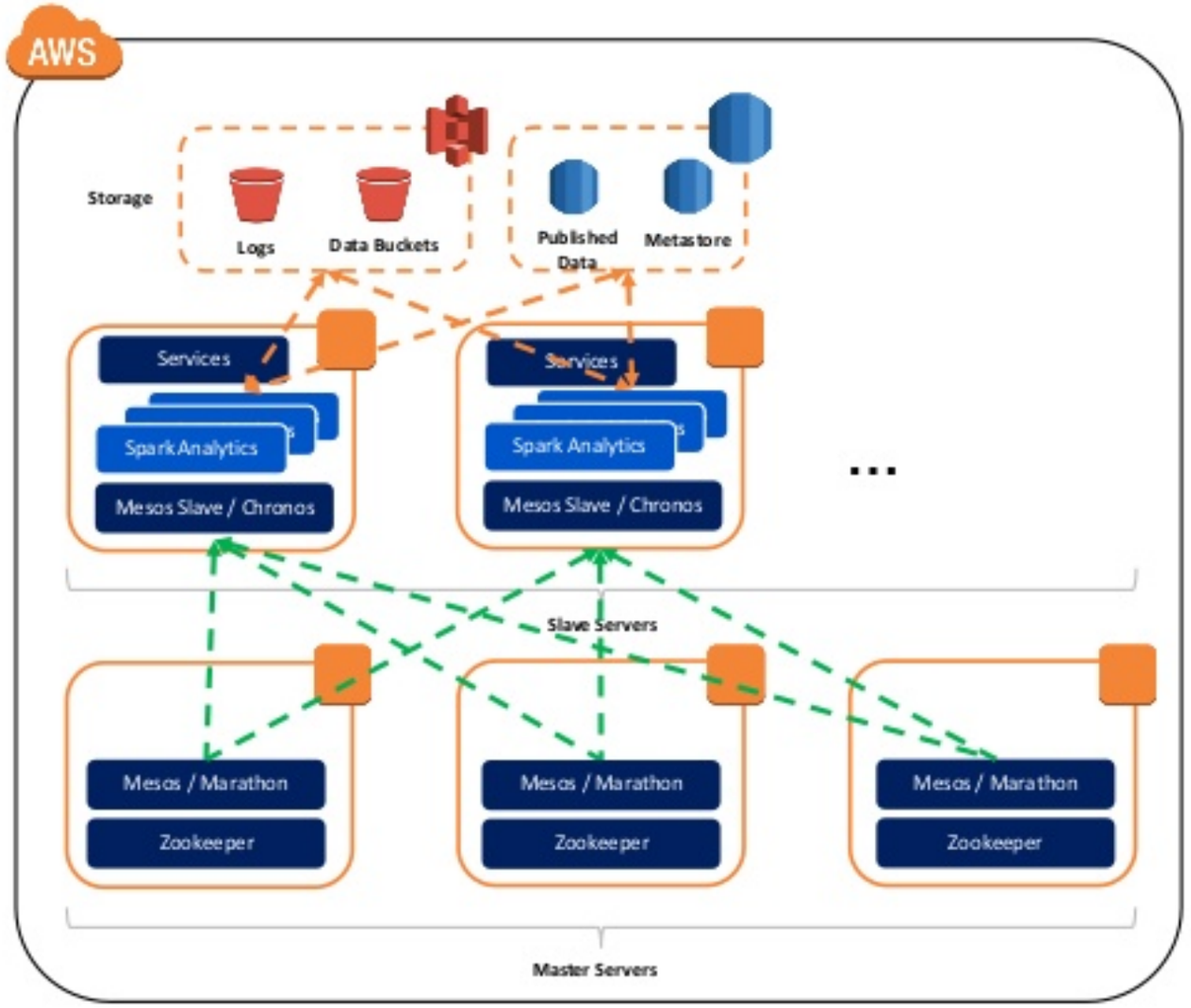
1. Data Ingested into a **Raw** storage area

2. Data is **Prepared**
   - Cleansed
   - Enriched
   - Linked
   - etc

3. **Models** are run

4. **User Tools** consume outputs
   - Rich custom built tools
   - Self service BI tools

Example of physical Architecture of Nerve Live in AWS



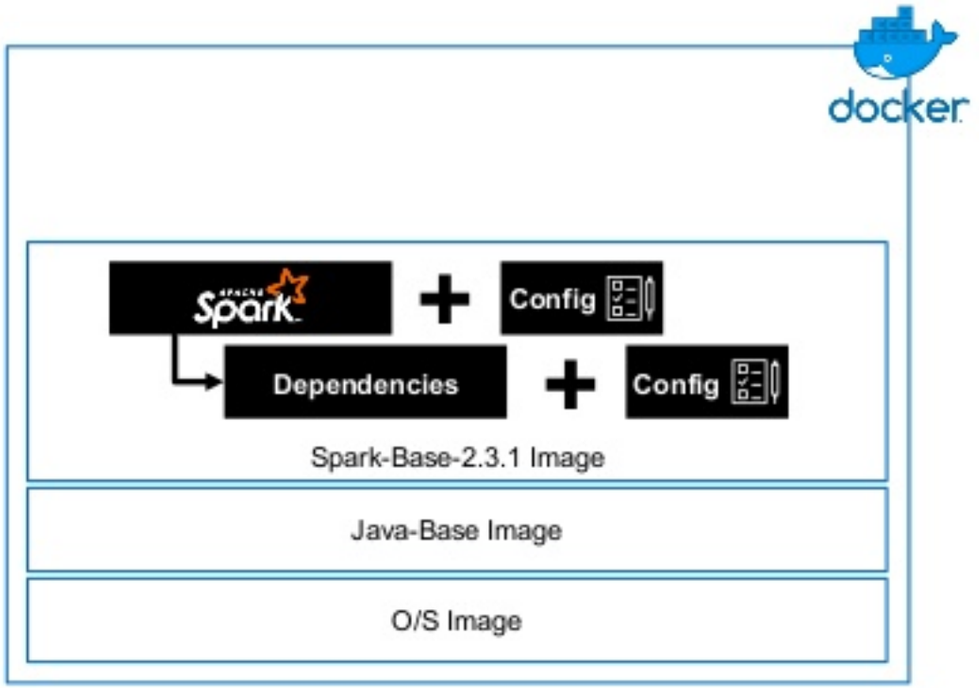
Nerve Live Analytics Cluster (AWS)

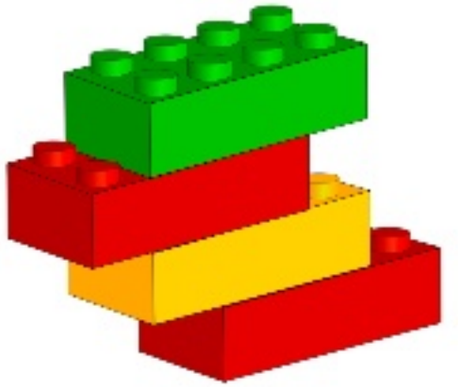
1. **Mesos** based **Spark** cluster
   - **Open Source** versions
2. Data stored mainly in **S3**
   - Raw
   - Prepared
   - Features
   - Models
3. Data **published** at end of Pipeline (**RDS**)
4. **Spark** Analytic workloads
   - Scheduled via **Marathon** or **Chronos**

Example of how we package Spark Jobs

1. Use Docker

2. Provide a Spark-Base-2.x.x Image to teams to package code into





Example of how we package Spark Jobs

1. Use Docker

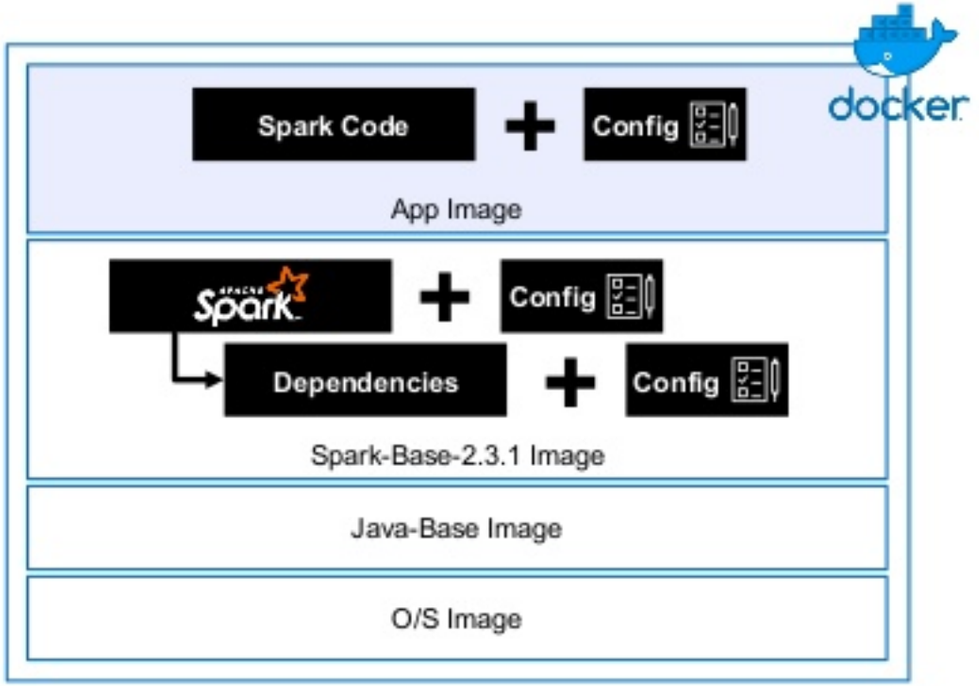
2. Provide a Spark-Base-2.x.x Image to teams to package code into

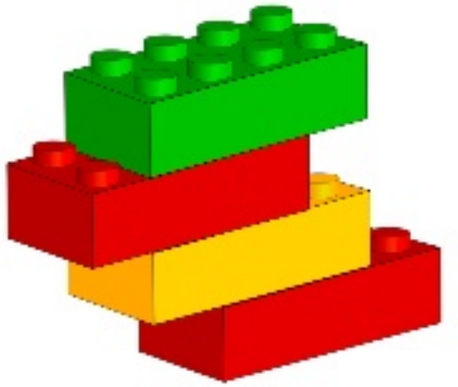
3. Teams package up their code into provided image

4. Can tell Spark to tell Mesos which Image to use for executors

   spark.mesos.executor.docker.image

5. Bonus - We can run different versions of Spark workloads in the cluster at same time!



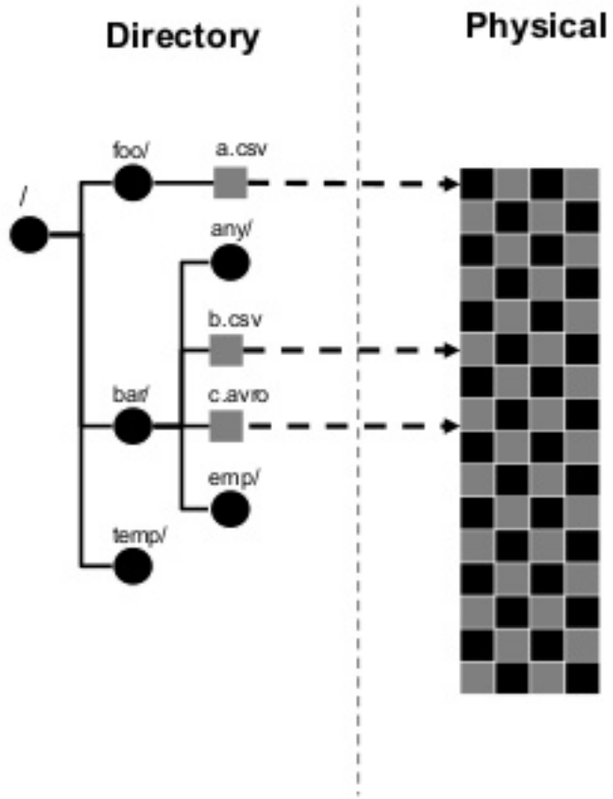


Observations of using Cloud Object Storage

2

Background – File Storage

File Storage



1. Organize (file) data using hierarchical structure (folder & files)

2. Allows
   - ✚ Traverse a path
   - ✚ Quick to “list” files in a directory
   - ✚ Quick to ”rename” files

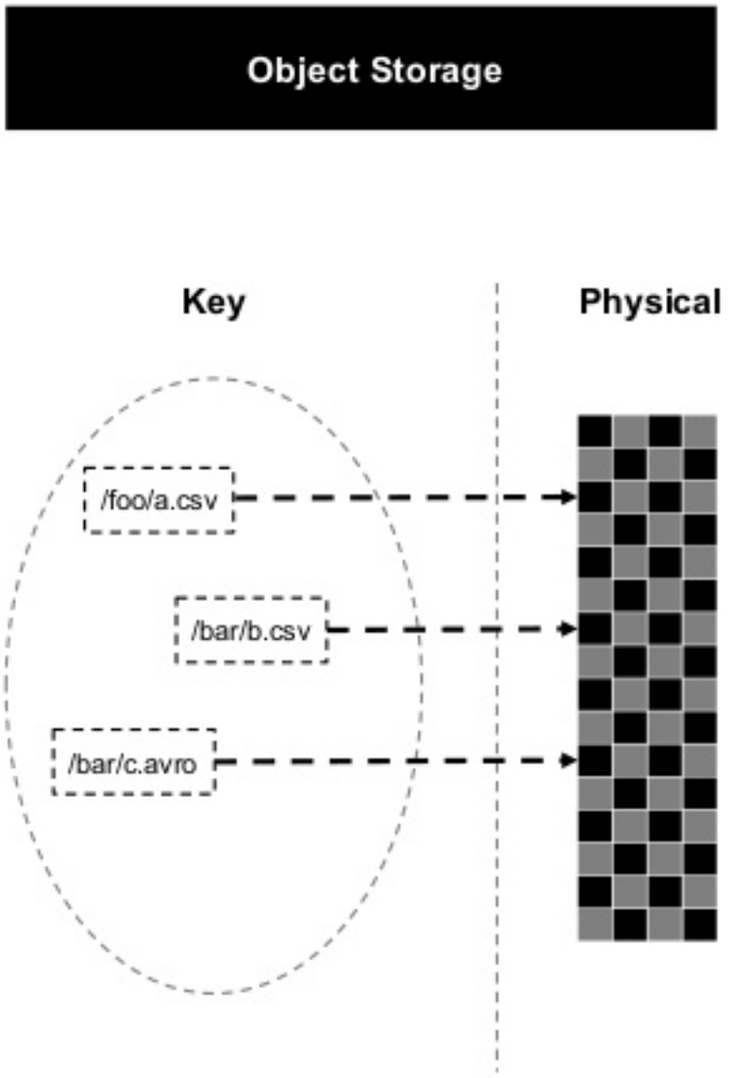
Background – Object Storage

1. No concept of organsiation

   Keys → Data

2. Unlike File Storage

   - Slow to “list” items in a “*directory*”
   - Slow to ”rename” files
   - Use REST calls



First Observation – The “Intent” to Read Data

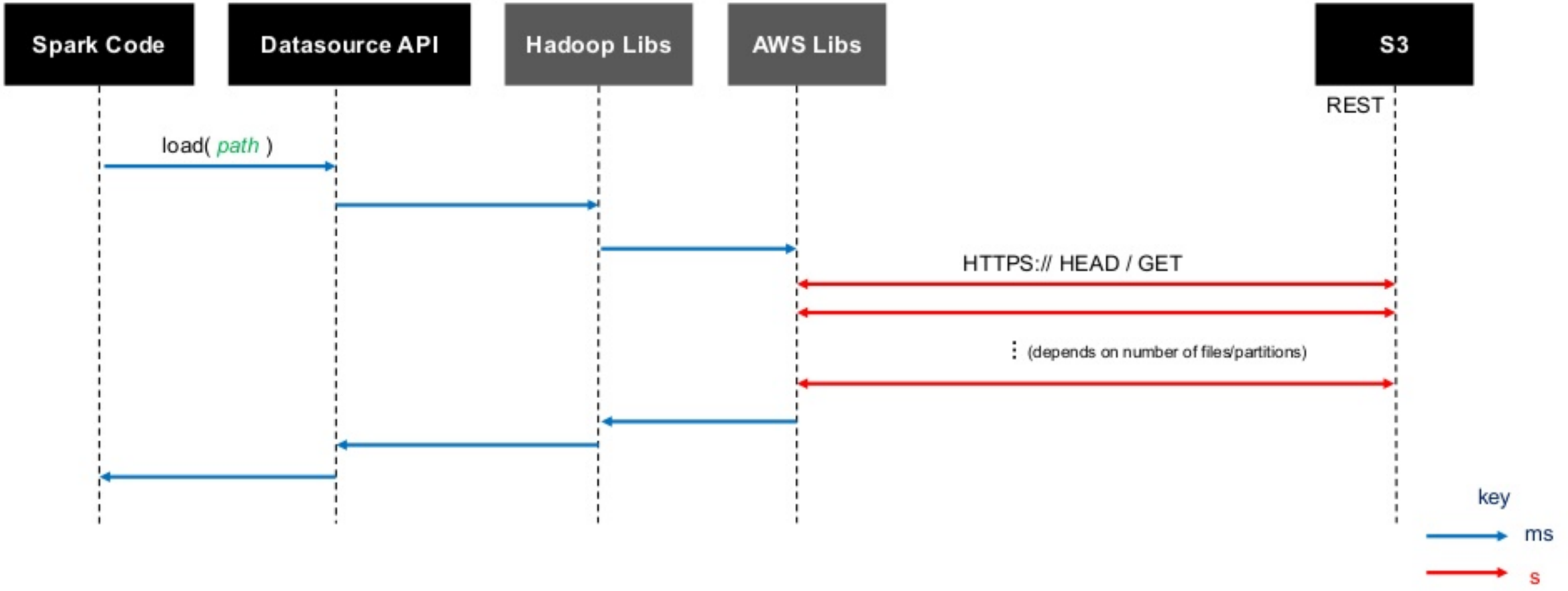
```
// Define a dataframe

val df = spark.read.format( “csv” )
           .schema( csvSchema ).load( “s3a://...” )

// Do some work
```

```
# Custom log4j properties
...

log4j.logger.com.amazonaws.request=DEBUG

...
```

First Observation – The "Intent" to Read Data

```
// Define a dataframe

val df = spark.read.format( "csv" )
           .schema( csvSchema ).load( "s3a://..." )

// Do some work
```

```
# Custom log4j properties
...

log4j.logger.com.amazonaws.request=DEBUG

...
```



First Observation – The “Intent” to Read Data – How many HTTPS calls?

With Schema

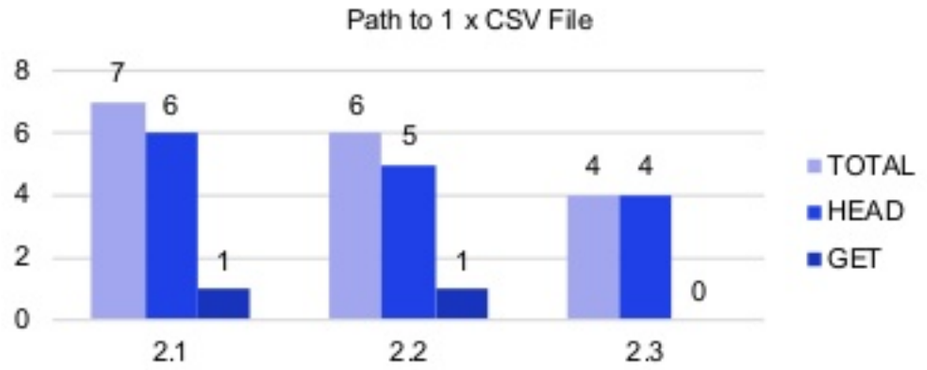
```
// Define a dataframe

val df = spark.read.format( “csv” )
            .schema( csvSchema ).load( “s3a://...” )

// Do some work
```

Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

First Observation – The “Intent” to Read Data – How many HTTPS calls?

With Schema

```
// Define a dataframe

val df = spark.read.format( “csv” )
            .schema( csvSchema ).load( “s3a://...” )

// Do some work
```

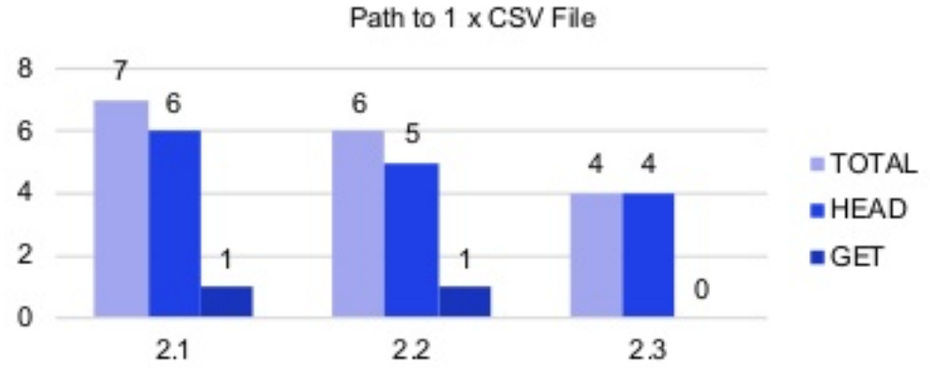


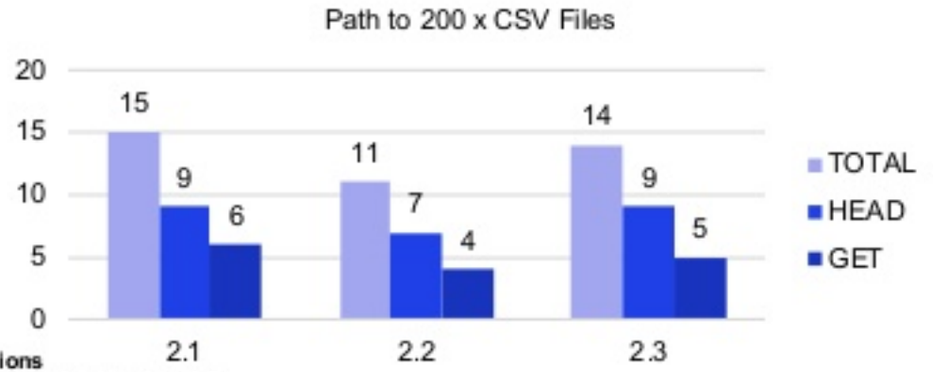
Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

First Observation – The "Intent" to Read Data – How many HTTPS calls?

With Schema

```
// Define a dataframe

val df = spark.read.format( "csv" )
           .schema( csvSchema ).load( "s3a://..." )

// Do some work
```



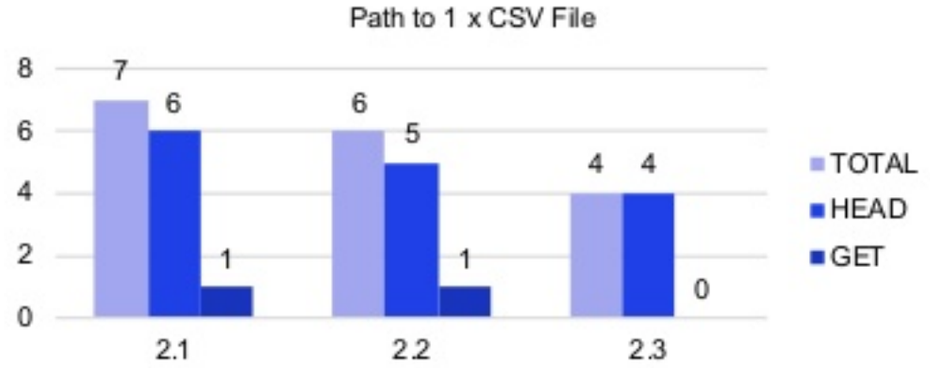


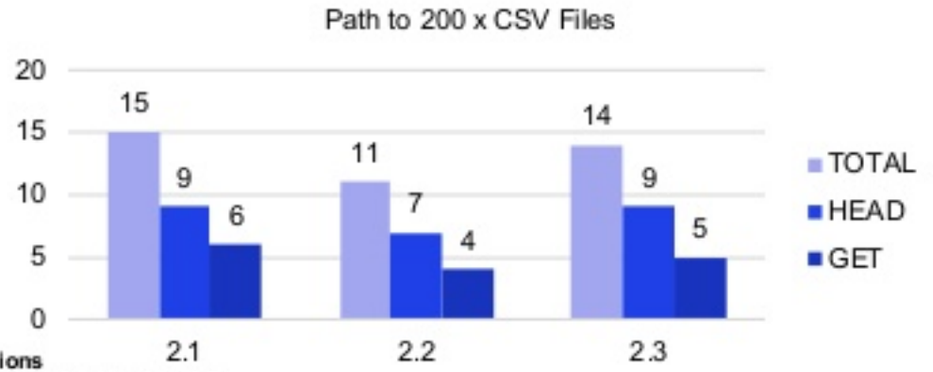
Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

First Observation – The "Intent" to Read Data – How many HTTPS calls?

With Schema

```
// Define a dataframe

val df = spark.read.format( "csv" )
           .schema( csvSchema ).load( "s3a://..." )

// Do some work
```





Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

vs.

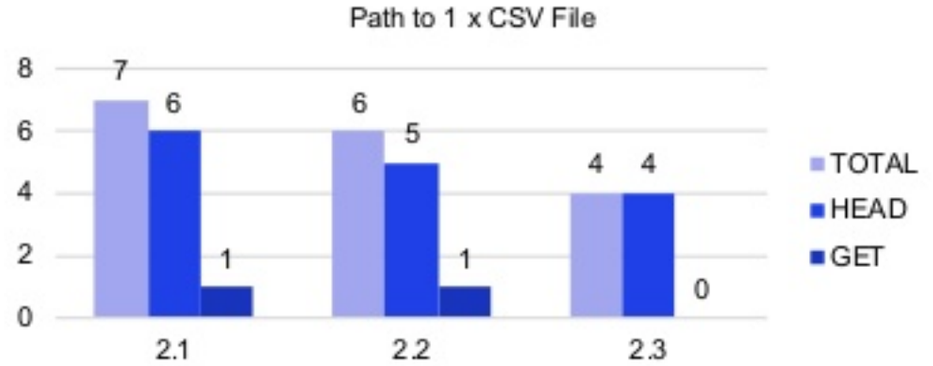
Infer Schema

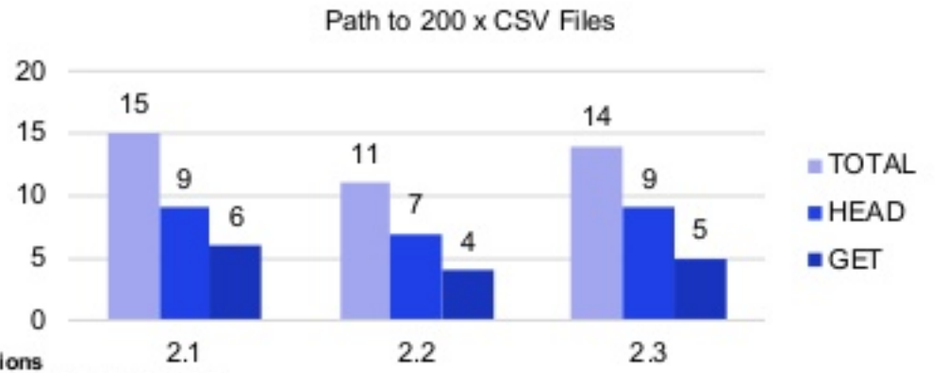
```
// Define a dataframe

val df = spark.read.format( "csv" )
           .option( "inferSchema", true )
           .load( "s3a://..." )

// Do some work
```

First Observation – The "Intent" to Read Data – How many HTTPS calls?

With Schema

```
// Define a dataframe

val df = spark.read.format( "csv" )
           .schema( csvSchema ).load( "s3a://..." )

// Do some work
```



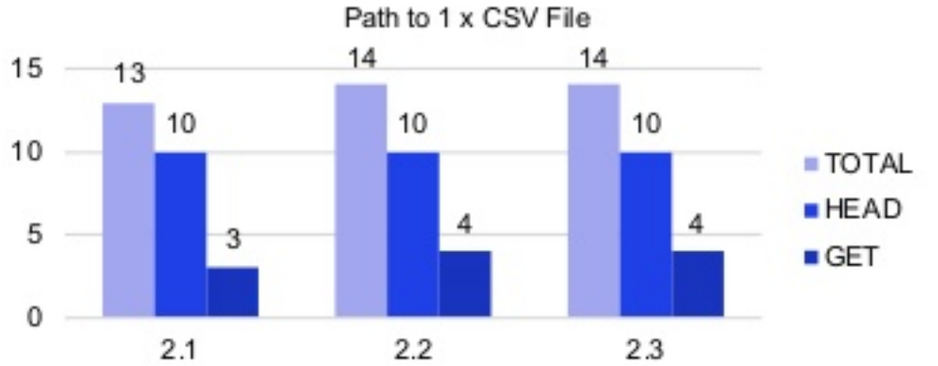


Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

vs.

Infer Schema

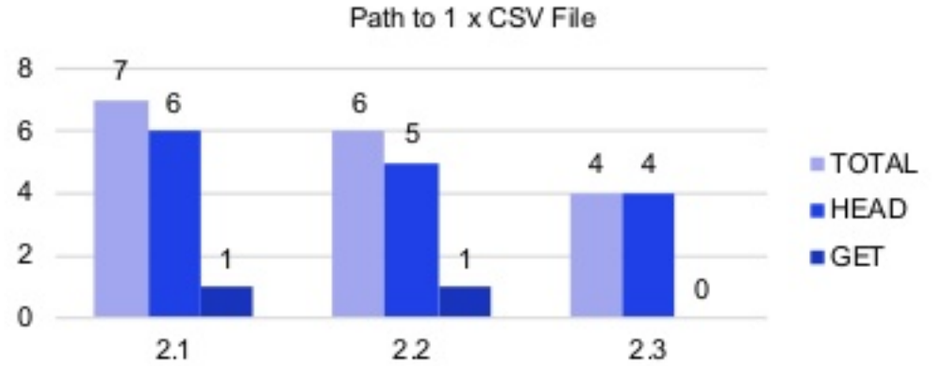
```
// Define a dataframe

val df = spark.read.format( "csv" )
           .option( "inferSchema", true )
           .load( "s3a://..." )

// Do some work
```

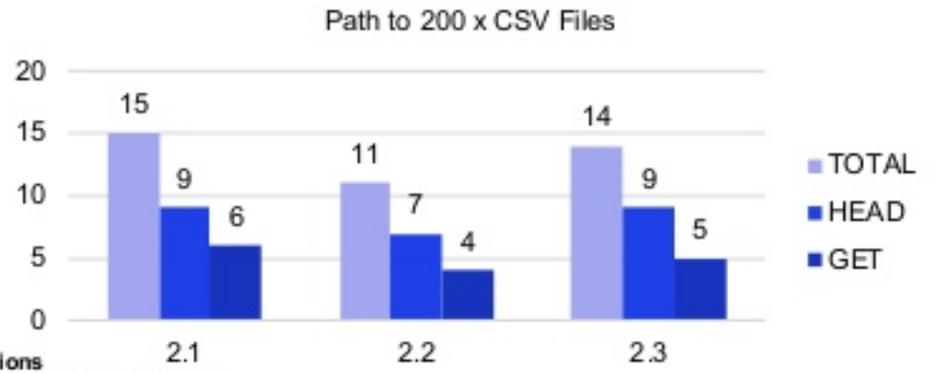


First Observation – The "Intent" to Read Data – How many HTTPS calls?

With Schema

```
// Define a dataframe

val df = spark.read.format( "csv" )
            .schema( csvSchema ).load( "s3a://..." )

// Do some work
```



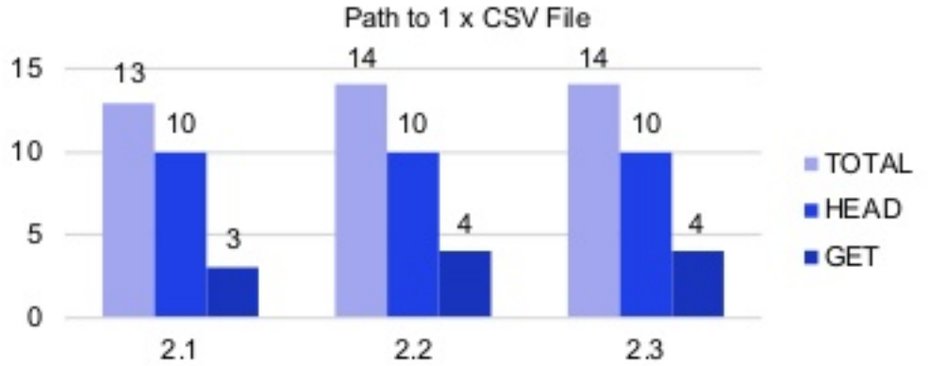


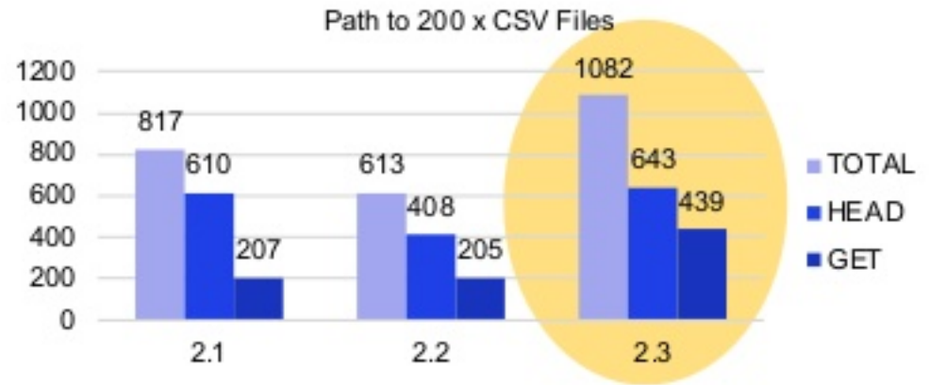
Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

vs.

Infer Schema

```
// Define a dataframe

val df = spark.read.format( "csv" )
            .option( "inferSchema", true )
            .load( "s3a://..." )

// Do some work
```





Second Observation – Writing Data – why does it take so long?

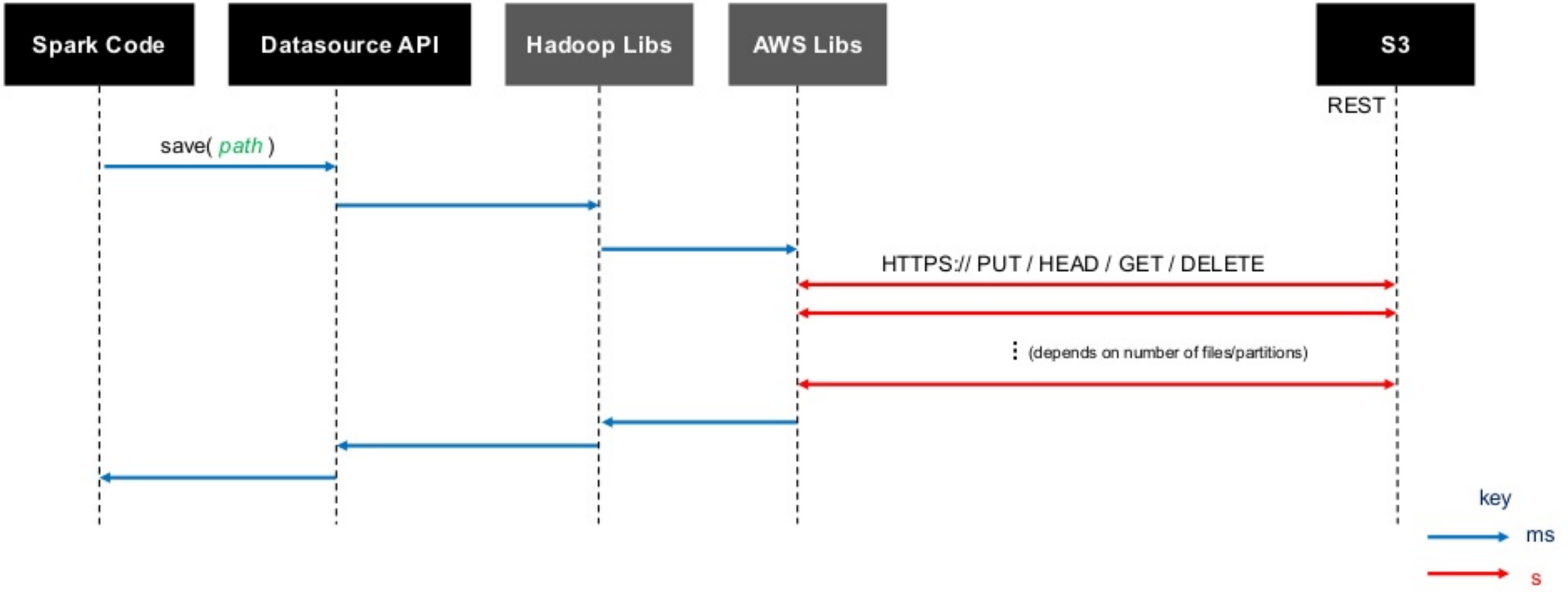
```
// Define a dataframe

df.write.format( “parquet” ).save( “s3a://...” )

// Do some work
```

```
# Custom log4j properties
...

log4j.logger.com.amazonaws.request=DEBUG

...
```

Second Observation – Writing Data – why does it take so long?

```
// Define a dataframe
df.write.format( "parquet" ).save( "s3a://..." )
// Do some work
```

```
# Custom log4j properties
...
log4j.logger.com.amazonaws.request=DEBUG
...
```



Second Observation – Writing Data – How many HTTPS calls?

Default

```
// Define a dataframe
df.write.format( “parquet” ).save( “s3a://...” )
// Do some work
```

Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

Second Observation – Writing Data – How many HTTPS calls?

Default

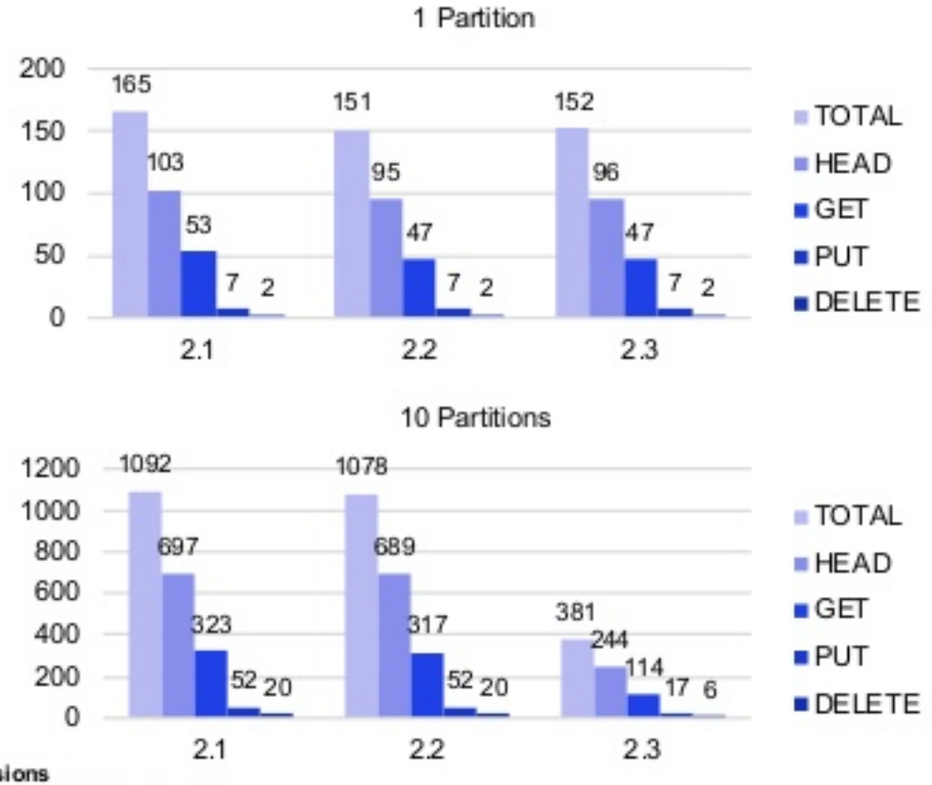
```
// Define a dataframe
df.write.format( “parquet” ).save( “s3a://...” )
// Do some work
```

Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

Second Observation – Writing Data – How many HTTPS calls?

Default

```
// Define a dataframe
df.write.format( “parquet” ).save( “s3a://...” )
// Do some work
```

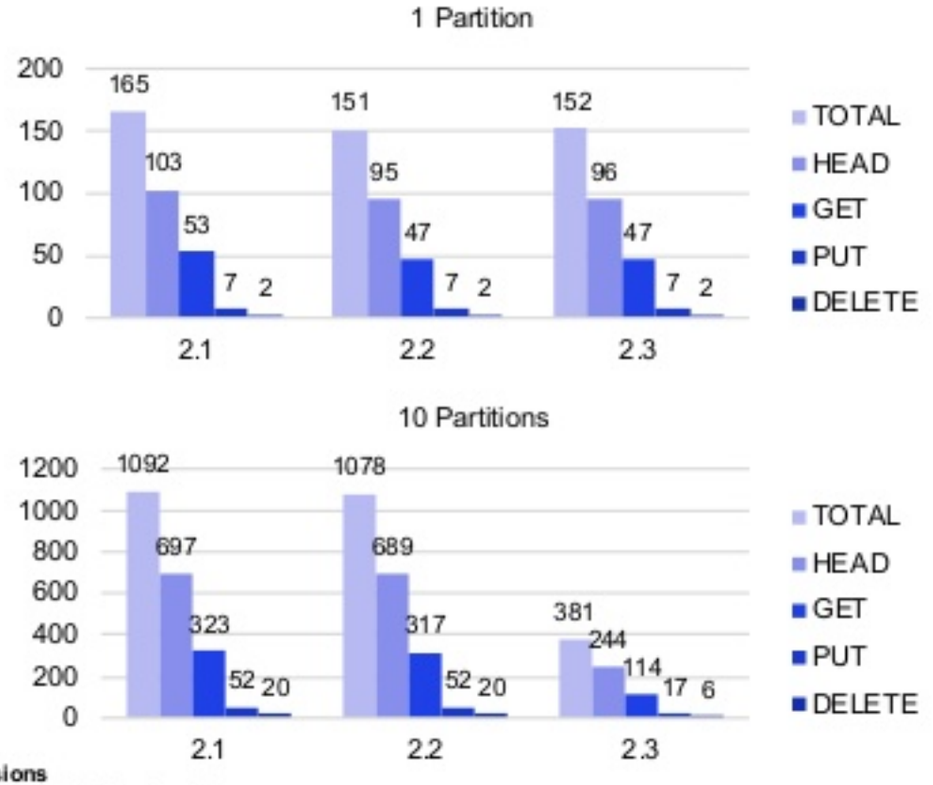


Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

Second Observation – Writing Data – How many HTTPS calls?

Default

```
// Define a dataframe

df.write.format( “parquet” ).save( “s3a://...” )

// Do some work
```



Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

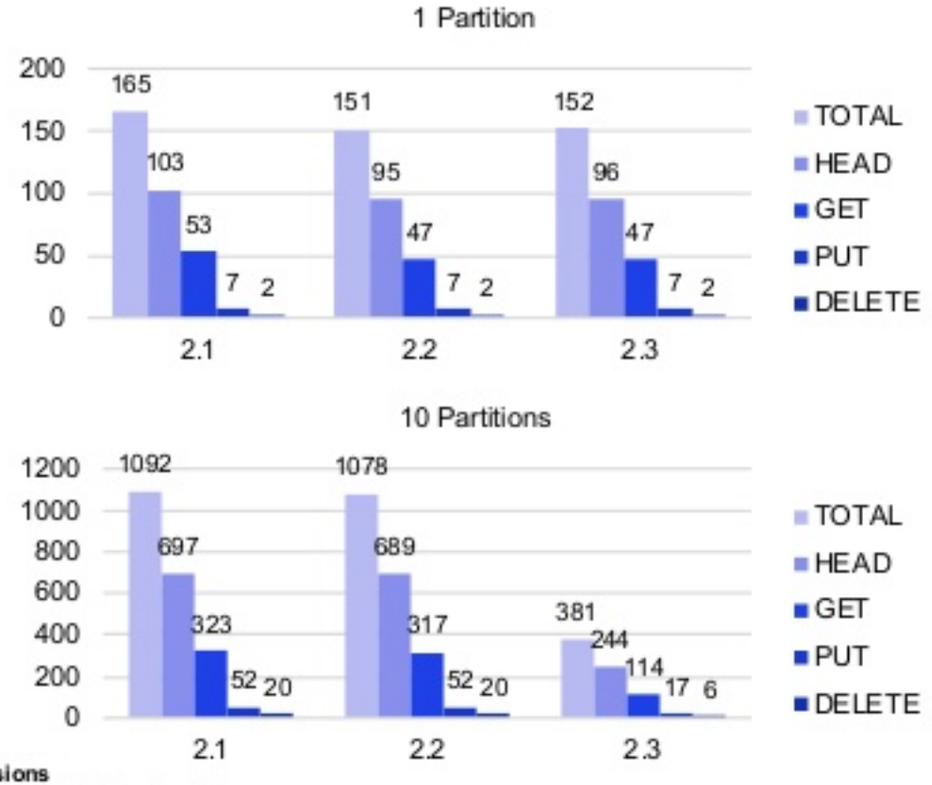
Go Faster

```
spark.conf.set(
“spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version”, 2)

df.write.format( “parquet” ).save( “s3a://...” )
```

Second Observation – Writing Data – How many HTTPS calls?

Default

```
// Define a dataframe
df.write.format( “parquet” ).save( “s3a://...” )
// Do some work
```



Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

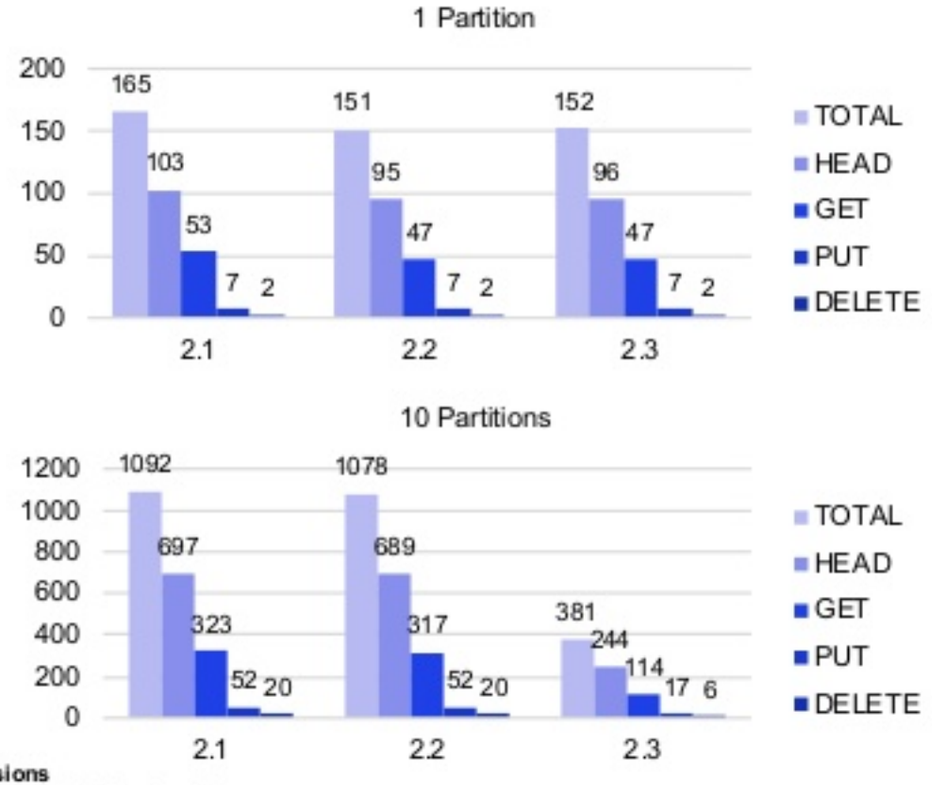
Go Faster

```
spark.conf.set(
“spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version”, 2)
df.write.format( “parquet” ).save( “s3a://...” )
```

Second Observation – Writing Data – How many HTTPS calls?

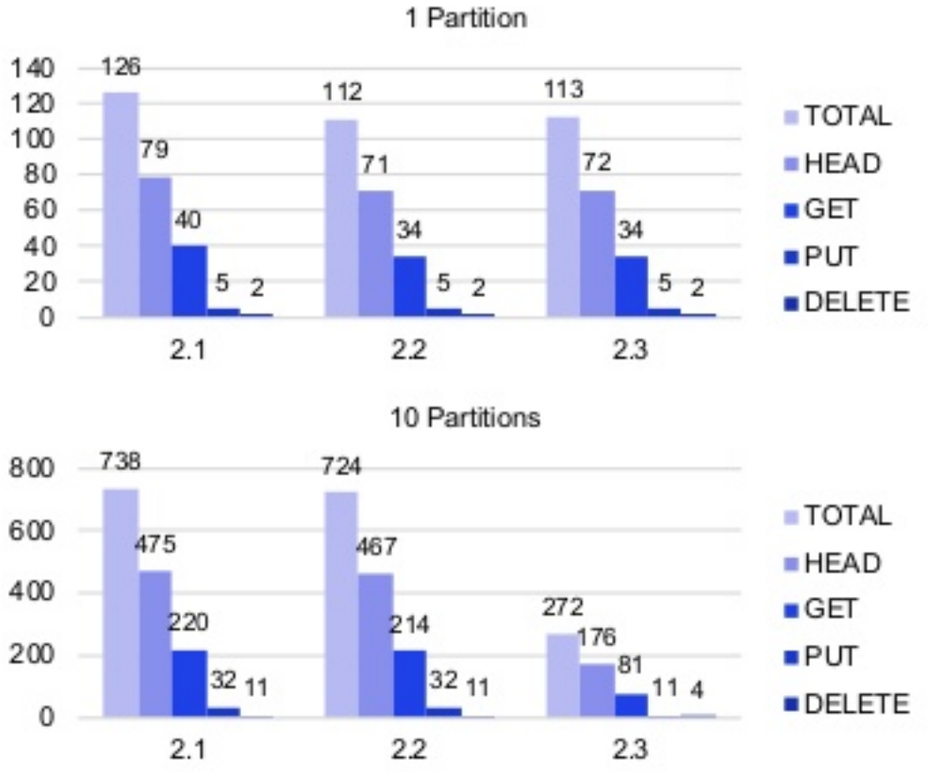
Default

```
// Define a dataframe
df.write.format( “parquet” ).save( “s3a://...” )
// Do some work
```



Go Faster

```
spark.conf.set(
“spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version”, 2)
df.write.format( “parquet” ).save( “s3a://...” )
```



Spark Versions
Spark 2.1 - spark-2.1.0-hadoop2.7
Spark 2.2 - spark-2.2.1-hadoop2.7
Spark 2.3 - spark-2.3.1-hadoop2.7

Conclusion

1. Cloud object storage has tradeoffs
2. Gives incredible storage capacity and scalability
3. However, do not expect same performance/ or characteristics as a file system
4. There are ***lots*** of configuration settings that can affect your performance with reading/writing to cloud object storage

```
spark.hadoop.fs.s3a.fast.upload=true
spark.hadoop.fs.s3a.connection.maximum=xxx
spark.hadoop.fs.s3a.multipart.size=xxx
spark.hadoop.fs.s3a.attempts.maximum=xxx
spark.hadoop.fs.s3a.multipart.threshold=xxx
spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version=2
etc
...
```

Note: Some are Hadoop version specific!

Design to Operate

3

Design to Operate – Choices

1. Our teams have faced **choices** for Loading / Saving data
   - What **URI scheme** to use?
     - i.e. S3://… S3n://… S3a://…, JDBC:// etc
   - What **file format**?
     - i.e. Parquet, Avro, CSV, etc
   - What **compression** to use?
     - i.e. Snappy, LZO, Gzip, etc
   - What additional options….
     - i.e. inferSchema, mergeSchema, pushDownPredicate…etc
   - Even what path to load/save data

Design to Operate – Choices

1. Our teams have faced **choices** for Loading / Saving data
   - What **URI scheme** to use?
     - i.e. S3://… S3n://… S3a://…, JDBC:// etc
   - What **file format**?
     - i.e. Parquet, Avro, CSV, etc
   - What **compression** to use?
     - i.e. Snappy, LZO, Gzip, etc
   - What additional options….
     - i.e. inferSchema, mergeSchema, pushDownPredicate…etc
   - Even what path to load/save data

```
val path = "s3n://my-bucket/csv-data/people"

val peopleDF =
        spark.read.format( "csv" )
        .option( "sep", ";" )
        .option( "inferSchema", "true" )
        .option( "header", "true" )
        .load( path )

...
```

2. Choices end up being "**embedded**" into code

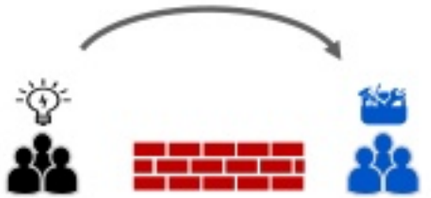
Design to Operate – Challenges

1. Different teams working in same cluster
   - Working on different use cases
   - Likely that Spark Jobs are running with **different settings** in same cluster

Design to Operate – Challenges

1. Different teams working in same cluster
   - Working on different use cases
   - Likely that Spark Jobs are running with **different settings** in same cluster

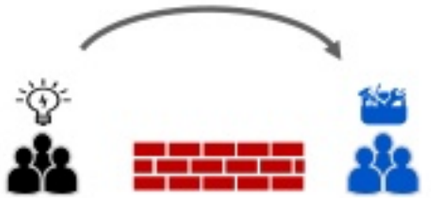
2. Those that build may not be the ones that end up running/operating Spark Jobs
   - Support Ops
   - Or Client IT teams



Design to Operate – Challenges

1. Different teams working in same cluster
   - Working on different use cases
   - Likely that Spark Jobs are running with **different settings** in same cluster

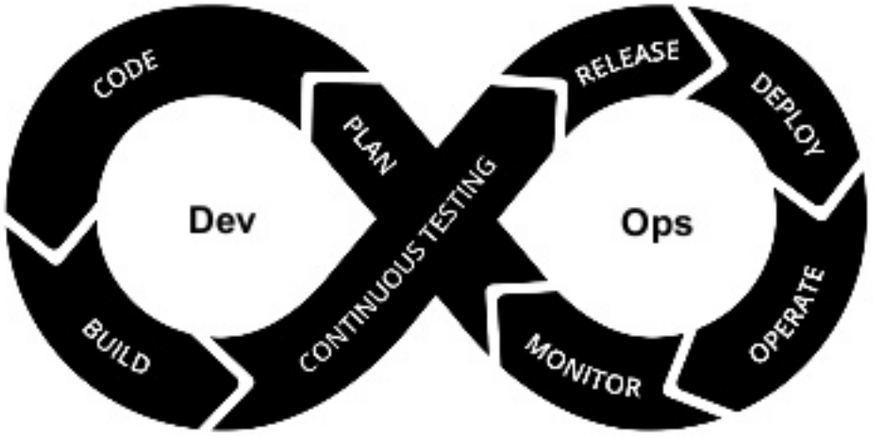
2. Those that build may not be the ones that end up running/operating Spark Jobs
   - Support Ops
   - Or Client IT teams



3. Difficult for those operating to understand
   - What configuration settings
   - What does it depend on? i.e. Data it needs
   - What depends on it? i.e. Data it writes
   - How can I change things quickly?

Design to Operate – Our Approach

1. Borrow ideas from Devops world
   - **"Strict separation of config from code"**

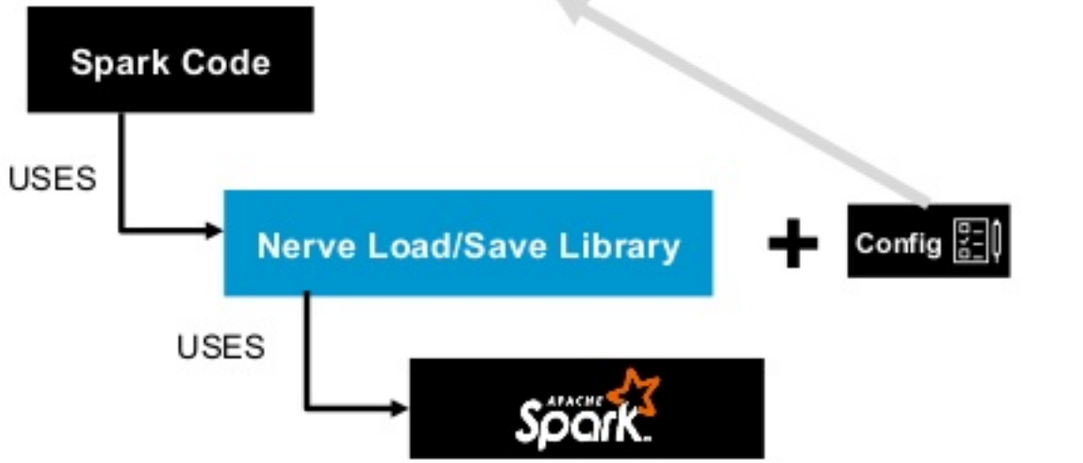


Design to Operate – Our Approach

1. Borrow ideas from Devops world
   - **"Strict separation of config from code"**

2. Custom **library** and **config** for loading and saving

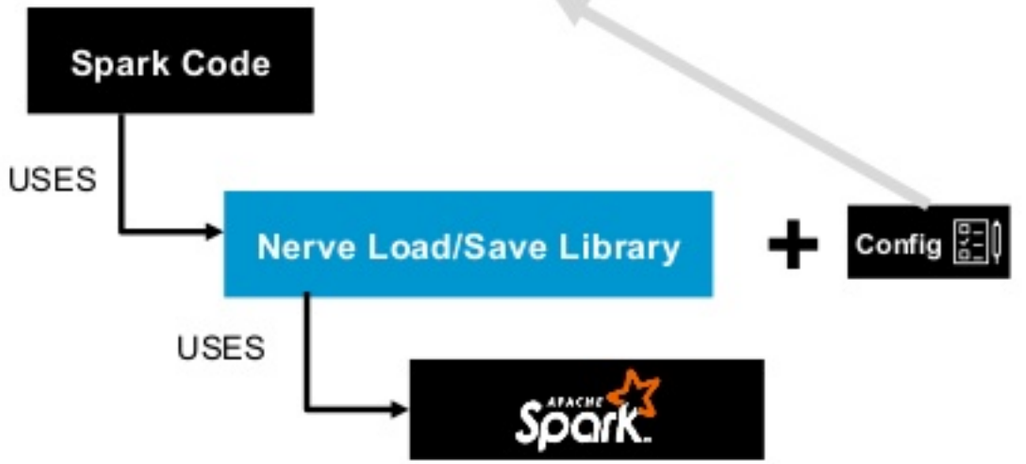
```
"system" : "bank",
"table": "customers",
"type": "input",
"description": "A Table with banking customer details.",
"format": {
  "type": "csv",
  "compression": "snappy",
  "partitions": "20"
  "mode": "overwrite"
}
"location": {
  "type": "s3",
  "properties": { ... }
}
"schema": [
  {
    "name": "id",
    "type": "int",
    "doc": "User Id"
  },
  ...
```



Design to Operate – Our Approach

1. Borrow ideas from Devops world
   - **"Strict separation of config from code"**
2. Custom **library** and **config** for loading and saving
3. For Loading / Saving data
   - What URI scheme to use?
     - i.e. S3://… S3n://… S3a://…, JDBC:// etc
   - What format?
     - i.e. Parquet, Avro, CSV, etc
   - What compression to use?
     - i.e. Snappy, LZO, Gzip, etc
   - What additional options….
     - i.e. inferSchema, mergeSchema, overwrite or append….
   - Even what path to load/save data

```
"system" : "bank",
"table": "customers",
"type": "input",
"description": "A Table with banking customer details.",
"format": {
  "type": "csv",
  "compression": "snappy",
  "partitions": "20"
  "mode": "overwrite"
}
"location": {
  "type": "s3",
  "properties": { ... }
}
"schema": [
  {
    "name": "id",
    "type": "int",
    "doc": "User Id"
  },
  ...
```



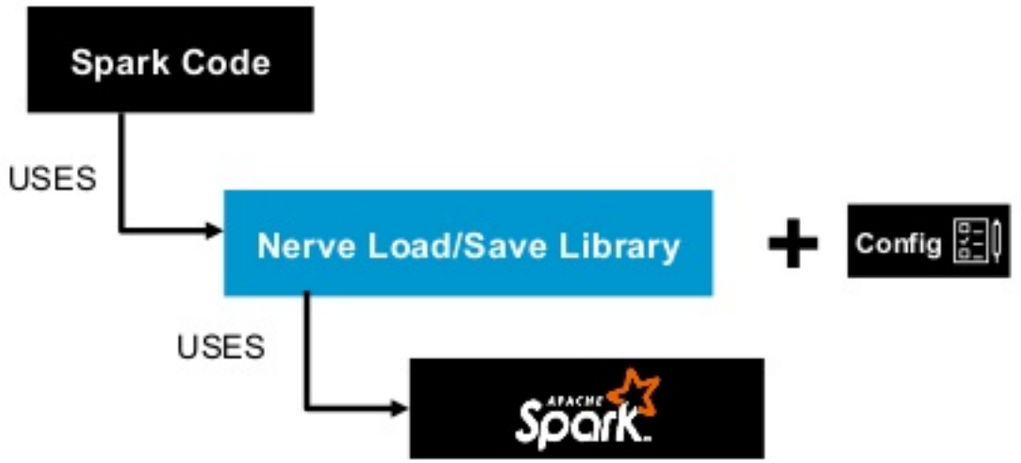
Design to Operate – Our Approach

1. Engineers use this library in their code

```
// Load a dataframe
val df = nerve.load( “system”, “table” )

// Save a dataframe
nerve.save( df, “system”, “table” )
```

2. Allows Engineers to focus on business logic and not worry about concerns such as:
   - Data formats
   - Compressions
   - Schemas
   - Partitions
   - Weird optimisation settings

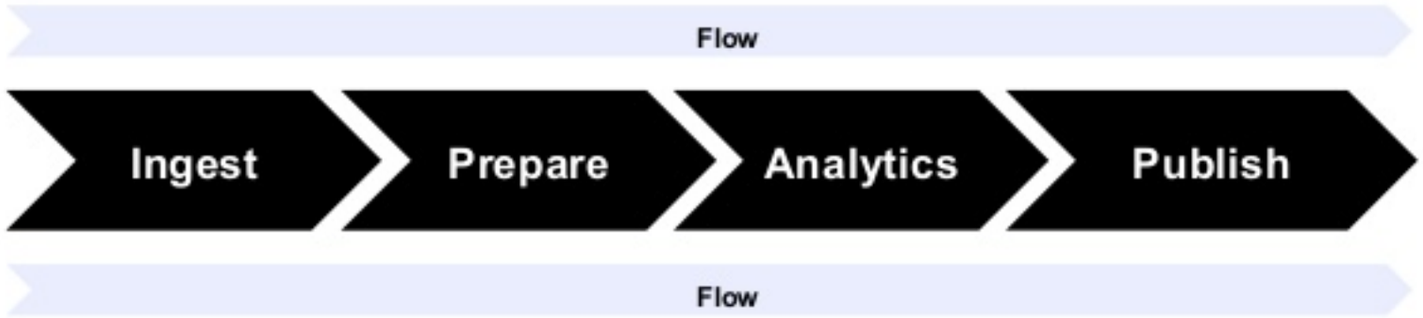
3. Allows teams that focus on operating/running to focus on
   - Consistency
   - Runtime optimisations



Pipelines!

4

Pipelines



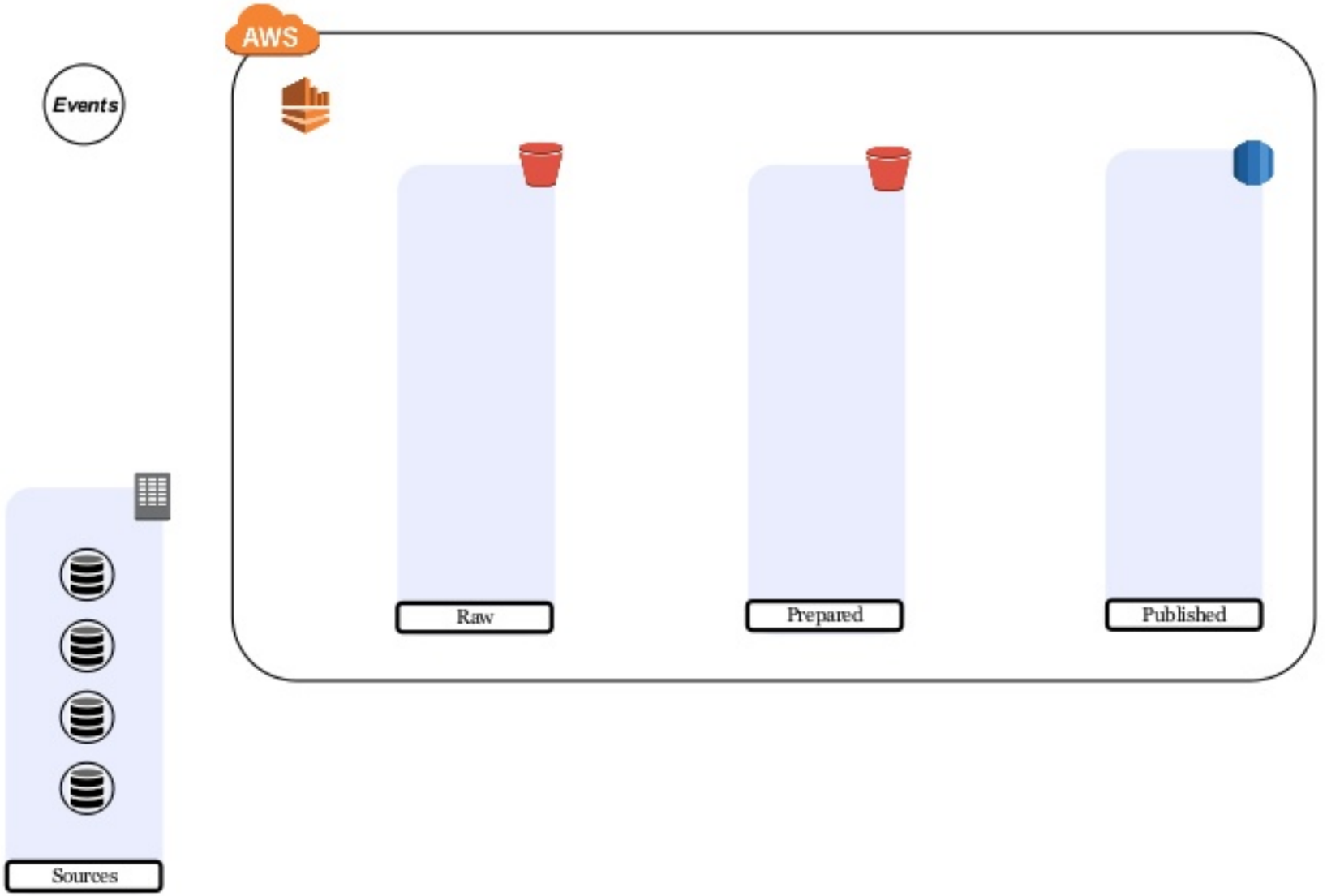
1. Pipelines mean different things to different people

2. Ultimately, its about the ***Flow of data***

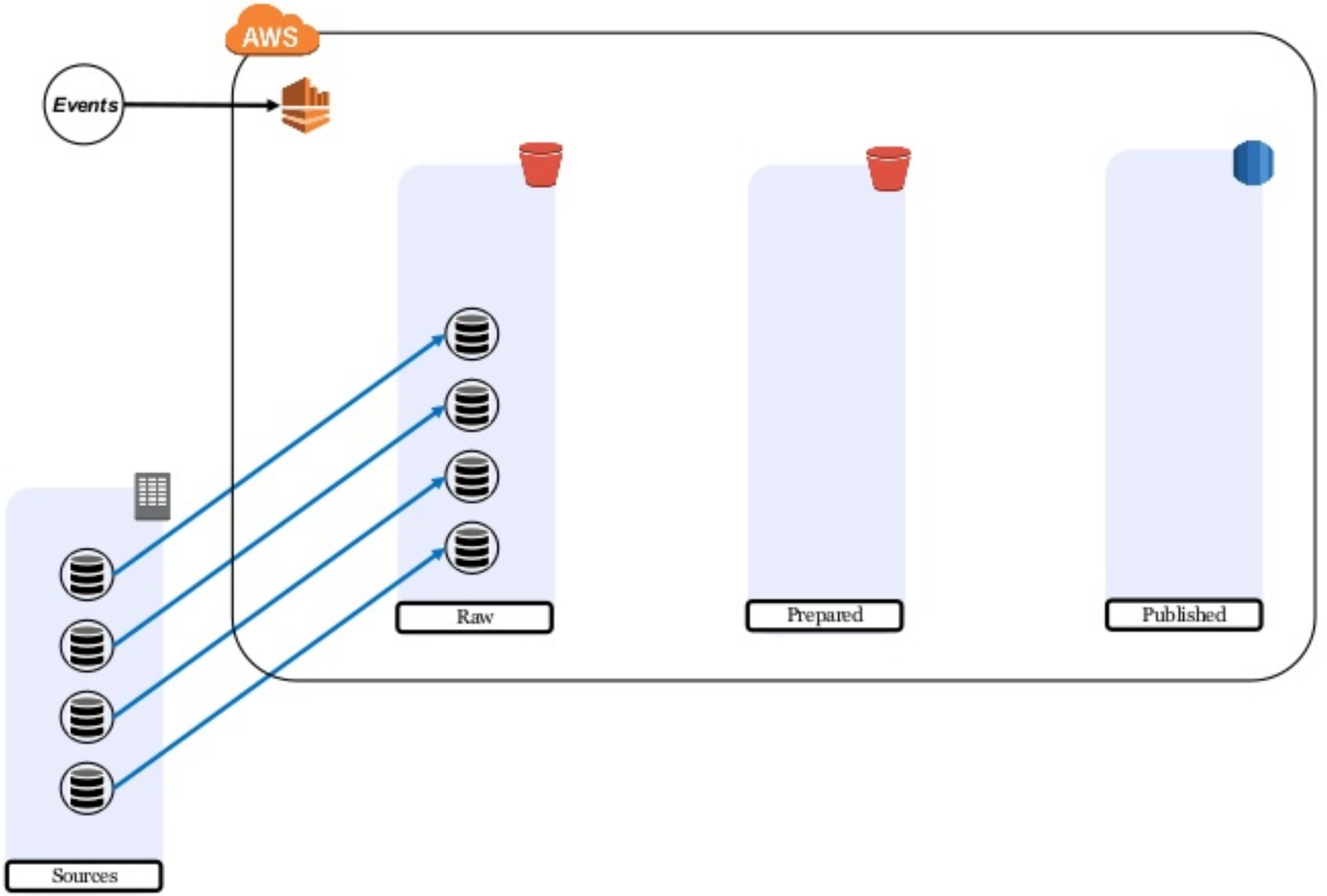
3. We need to
   - Monitor
   - Understand
   - React

4. It looks simple, but in fact its ***not***

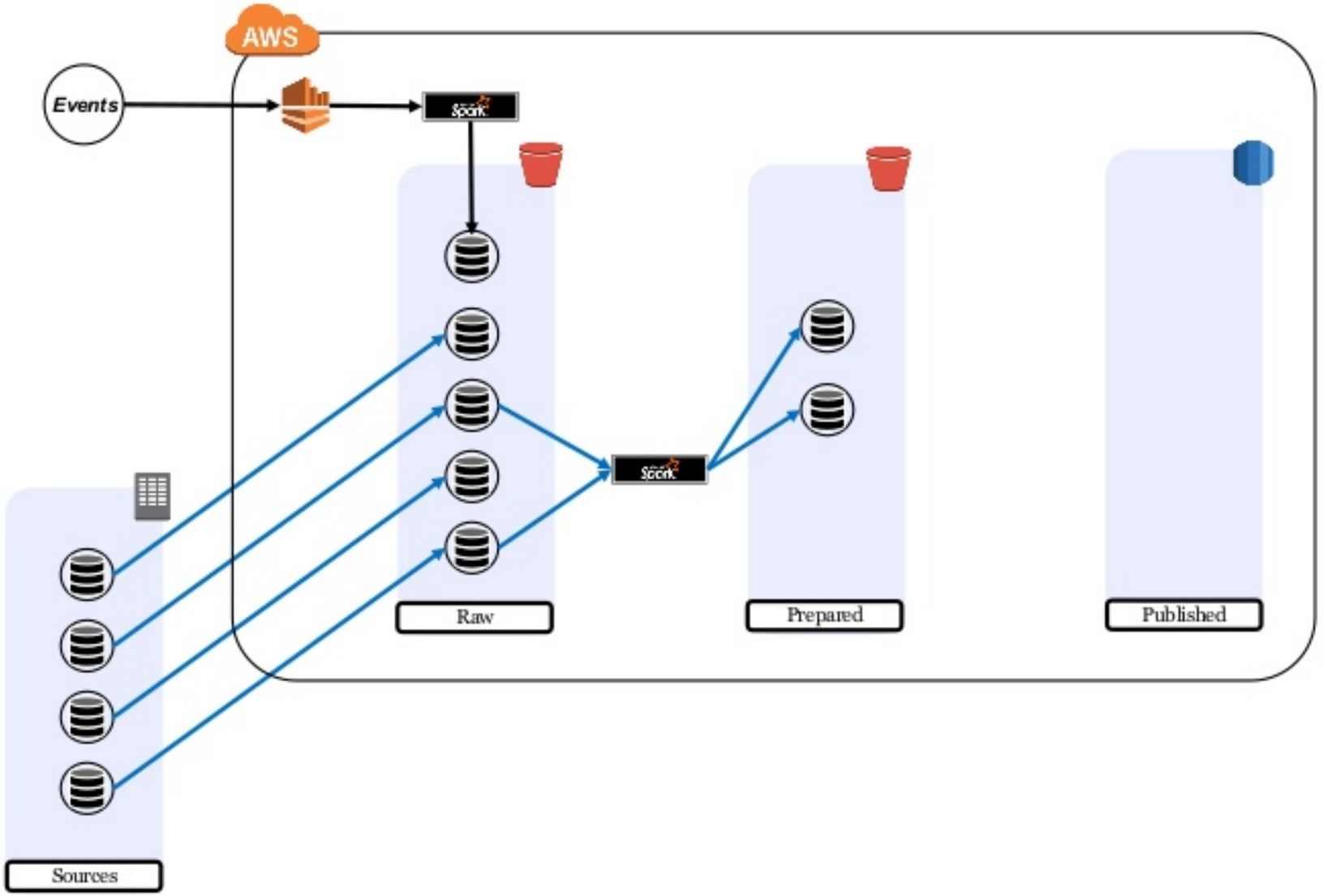
Pipelines – Example



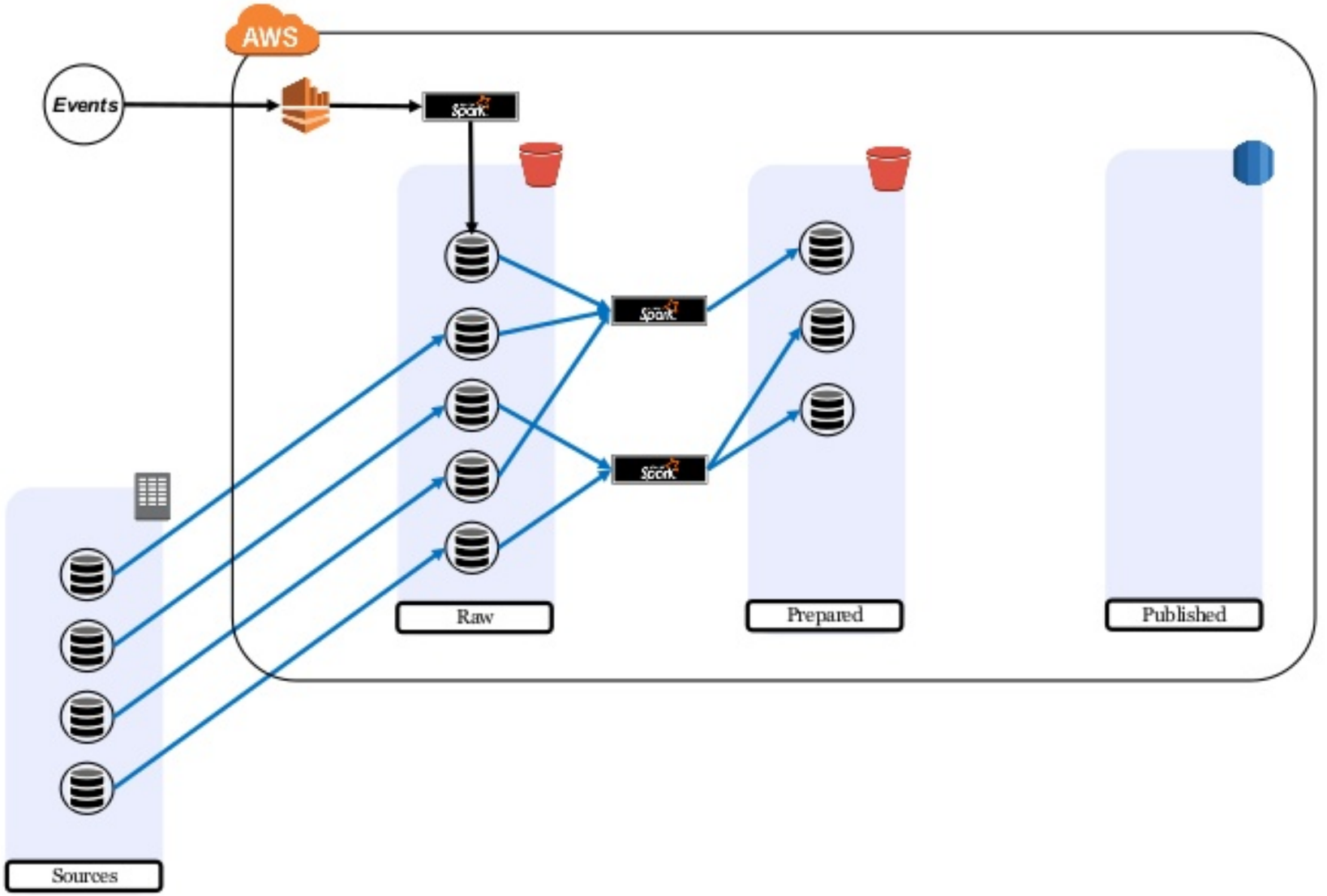
Pipelines – Example



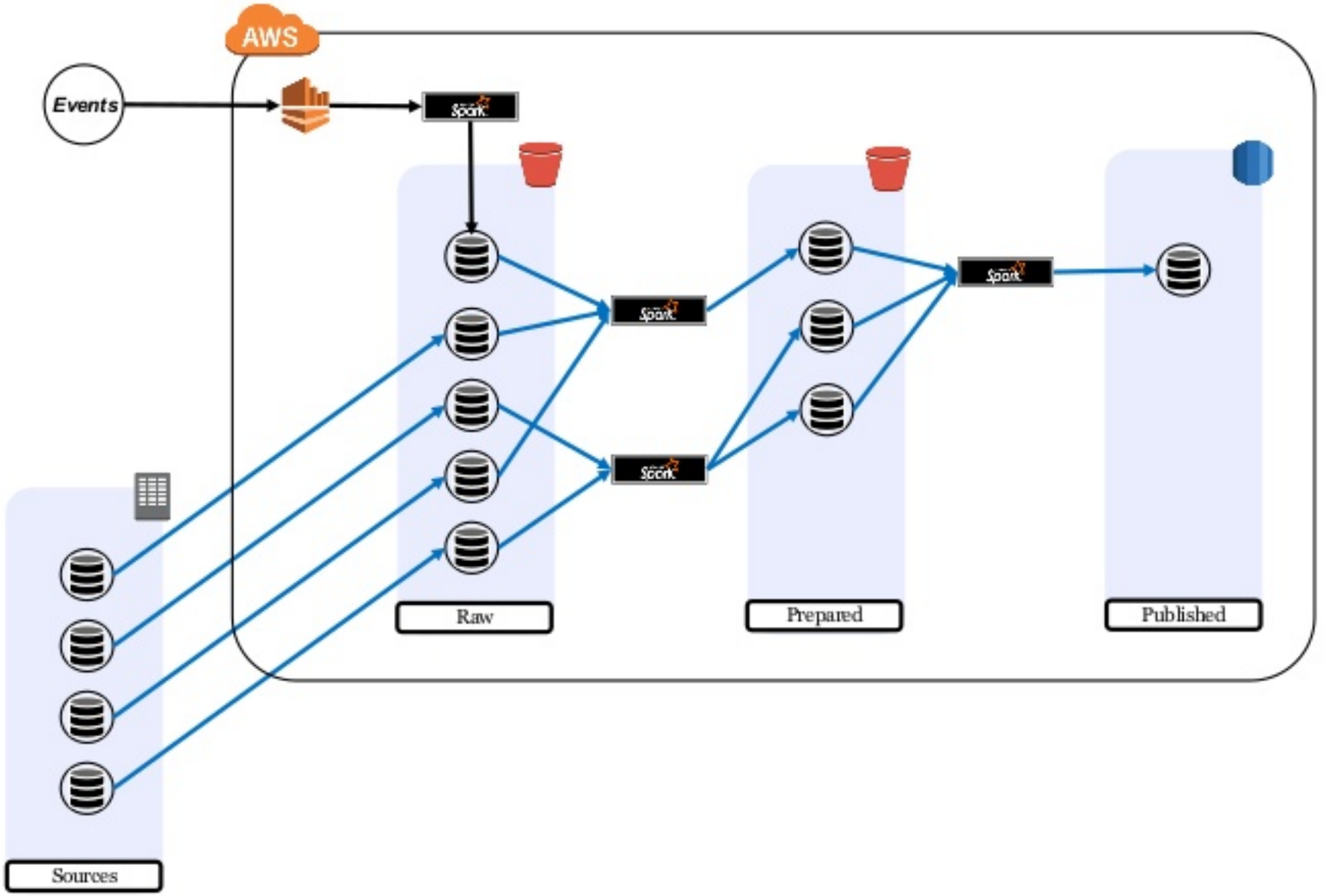
Pipelines – Example



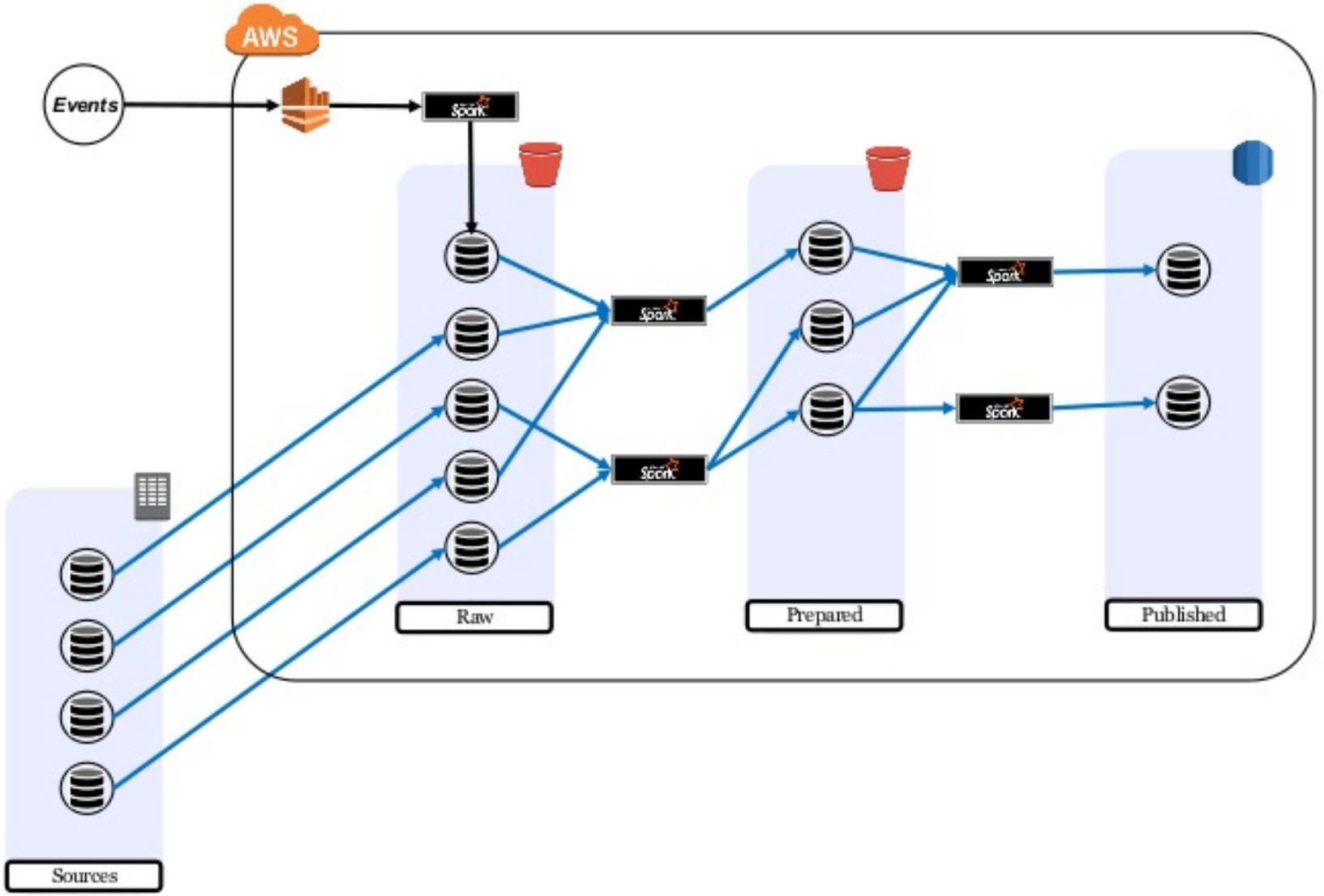
Pipelines – Example



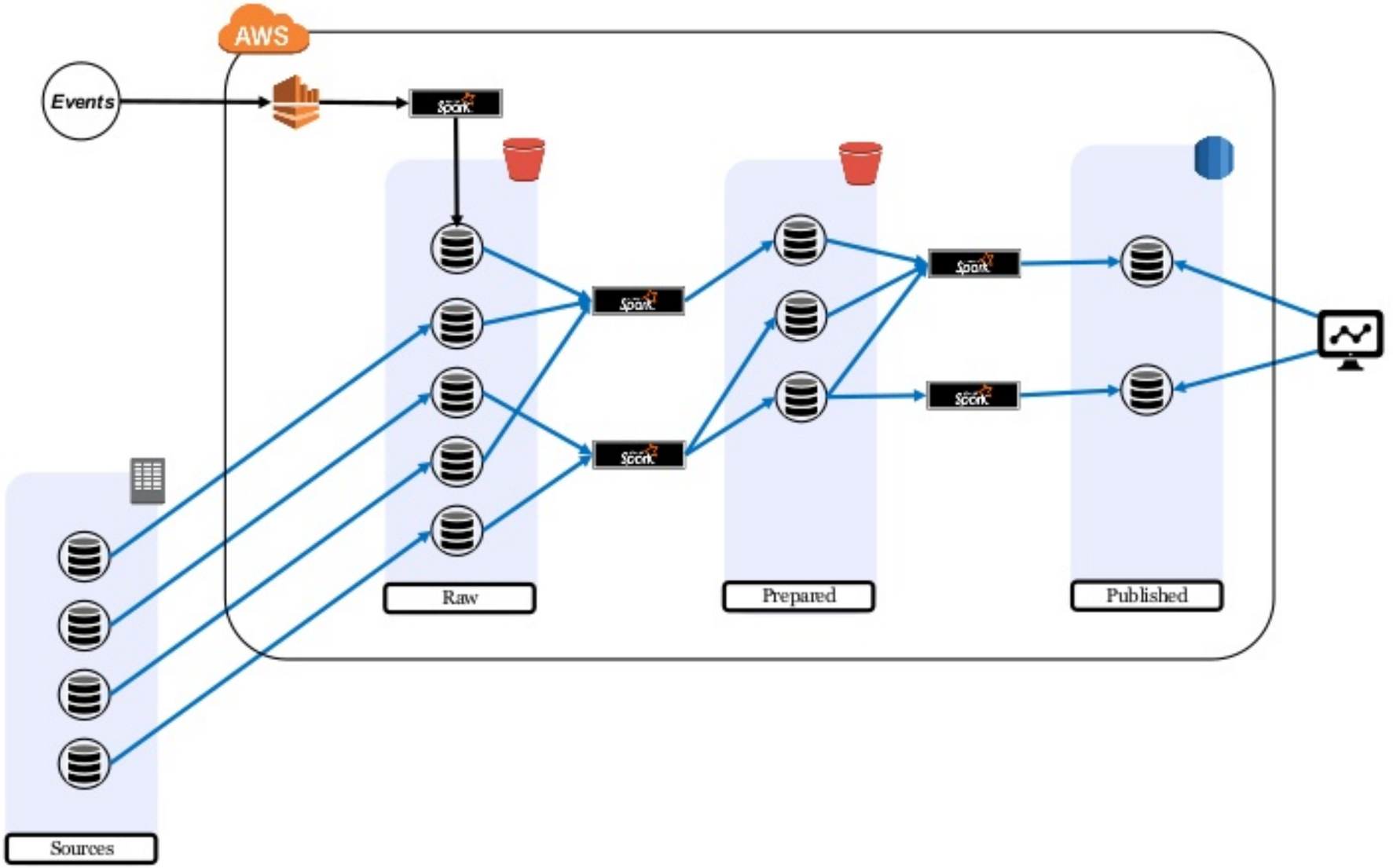
Pipelines – Example



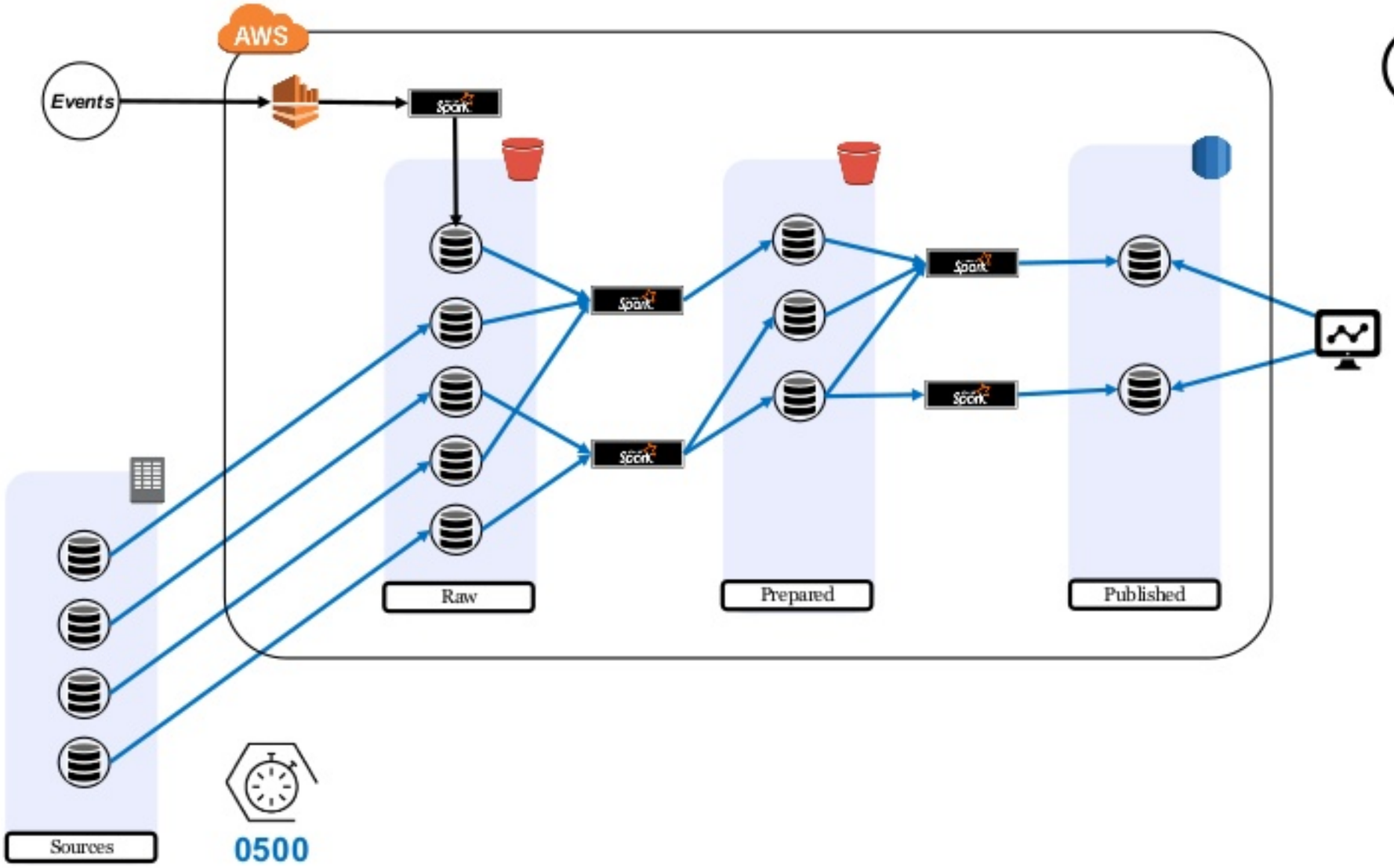
Pipelines – Example



Pipelines – Example

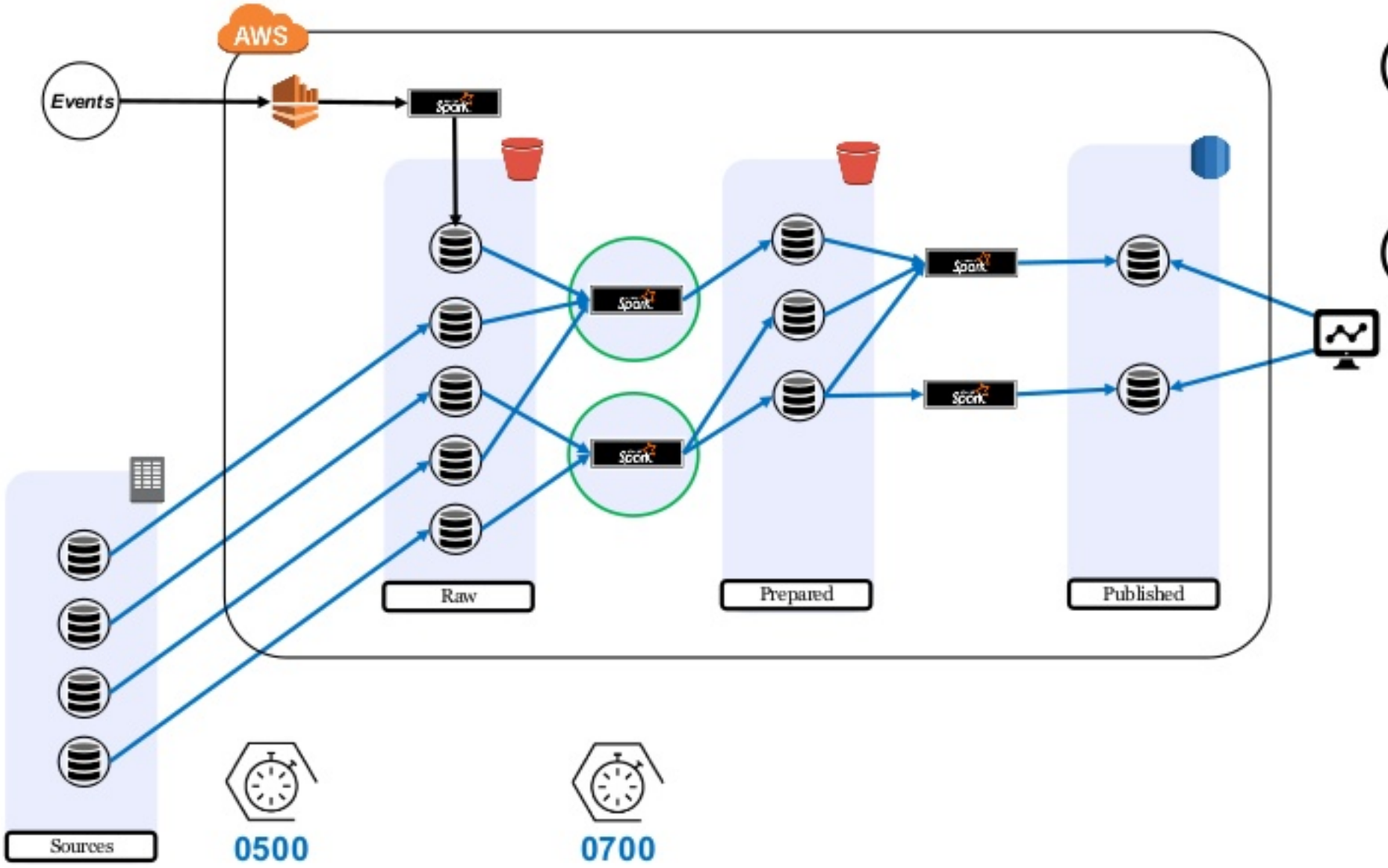


Pipelines – Example – Agreed Schedule



1 Agree a schedule with client when to send data

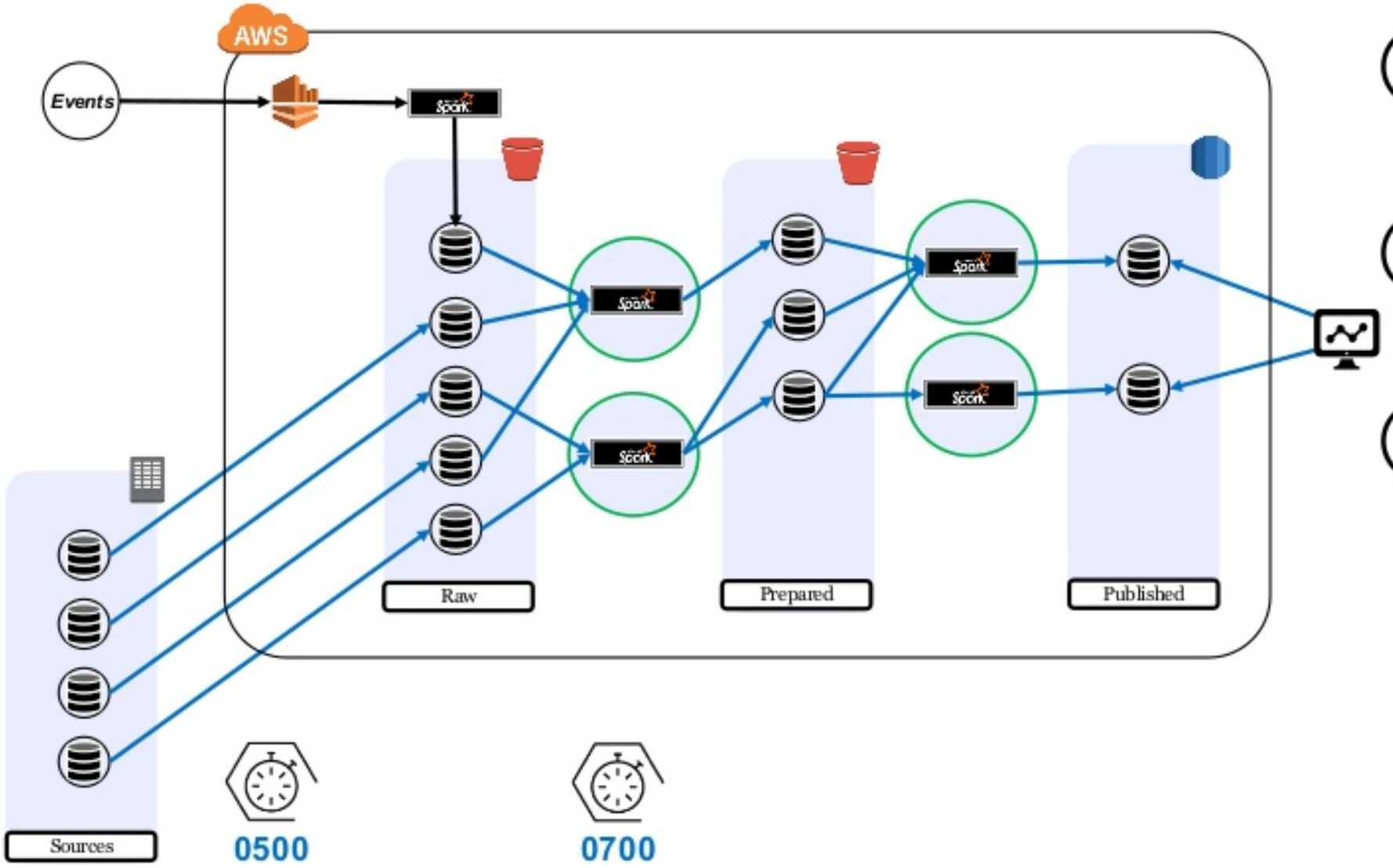
Pipelines – Example – Agreed Schedule



1. Agree a schedule with client when to send data

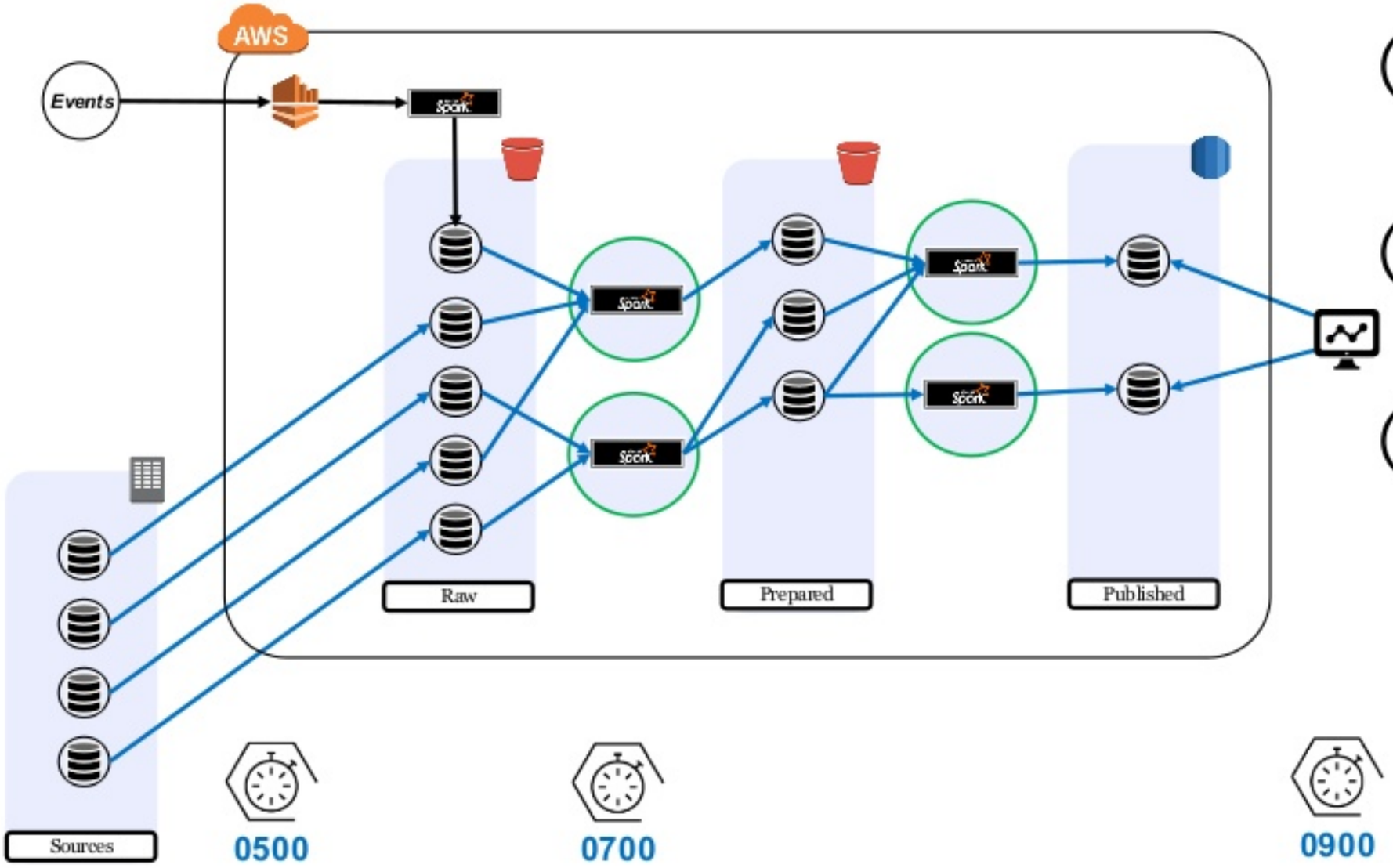
2. Configure schedule to kick off Spark Jobs

Pipelines – Example – Agreed Schedule



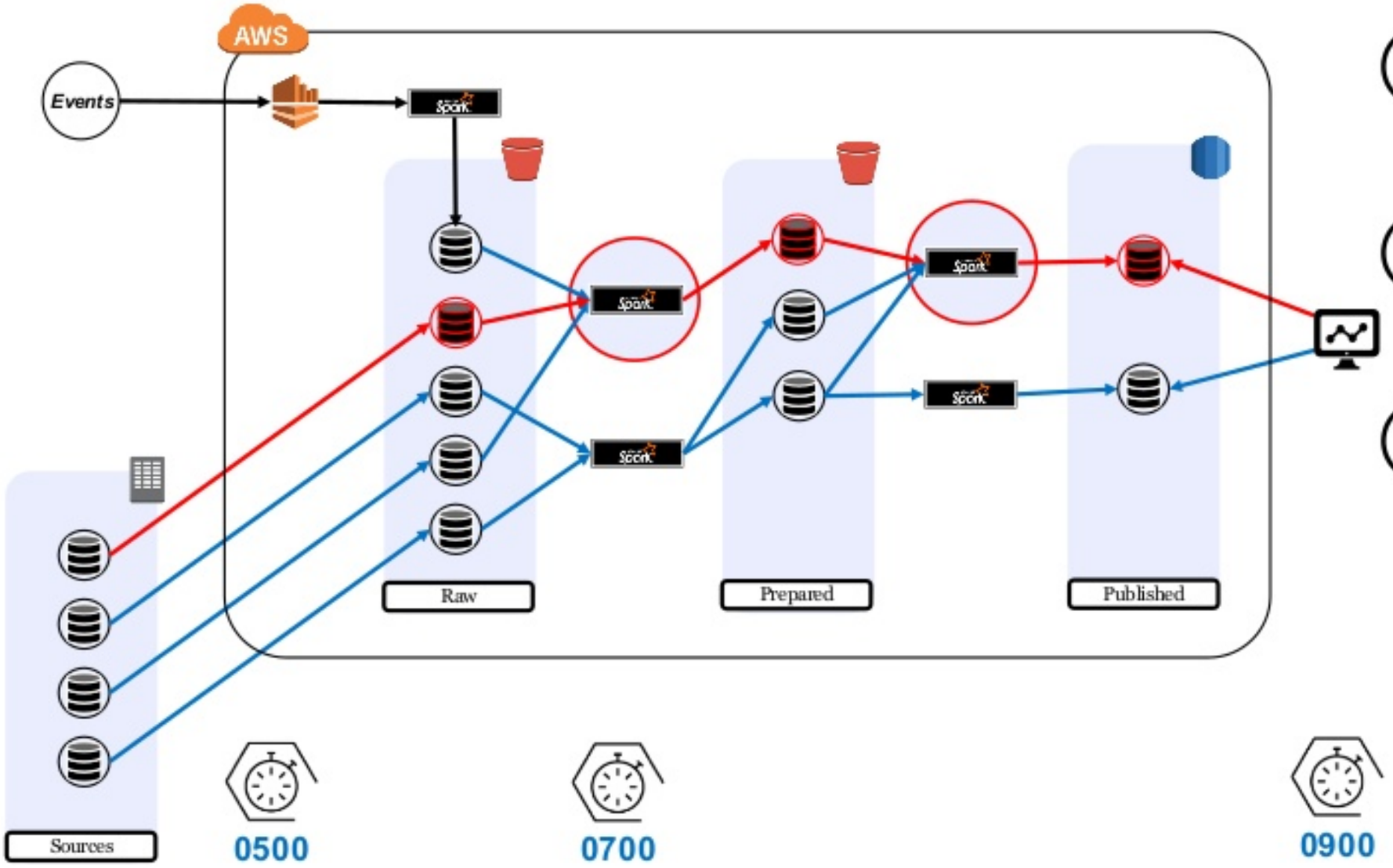
1. Agree a schedule with client when to send data
2. Configure schedule to kick off Spark Jobs
3. Subsequent Spark Jobs kick off when previous have completed

Pipelines – Example – Agreed Schedule



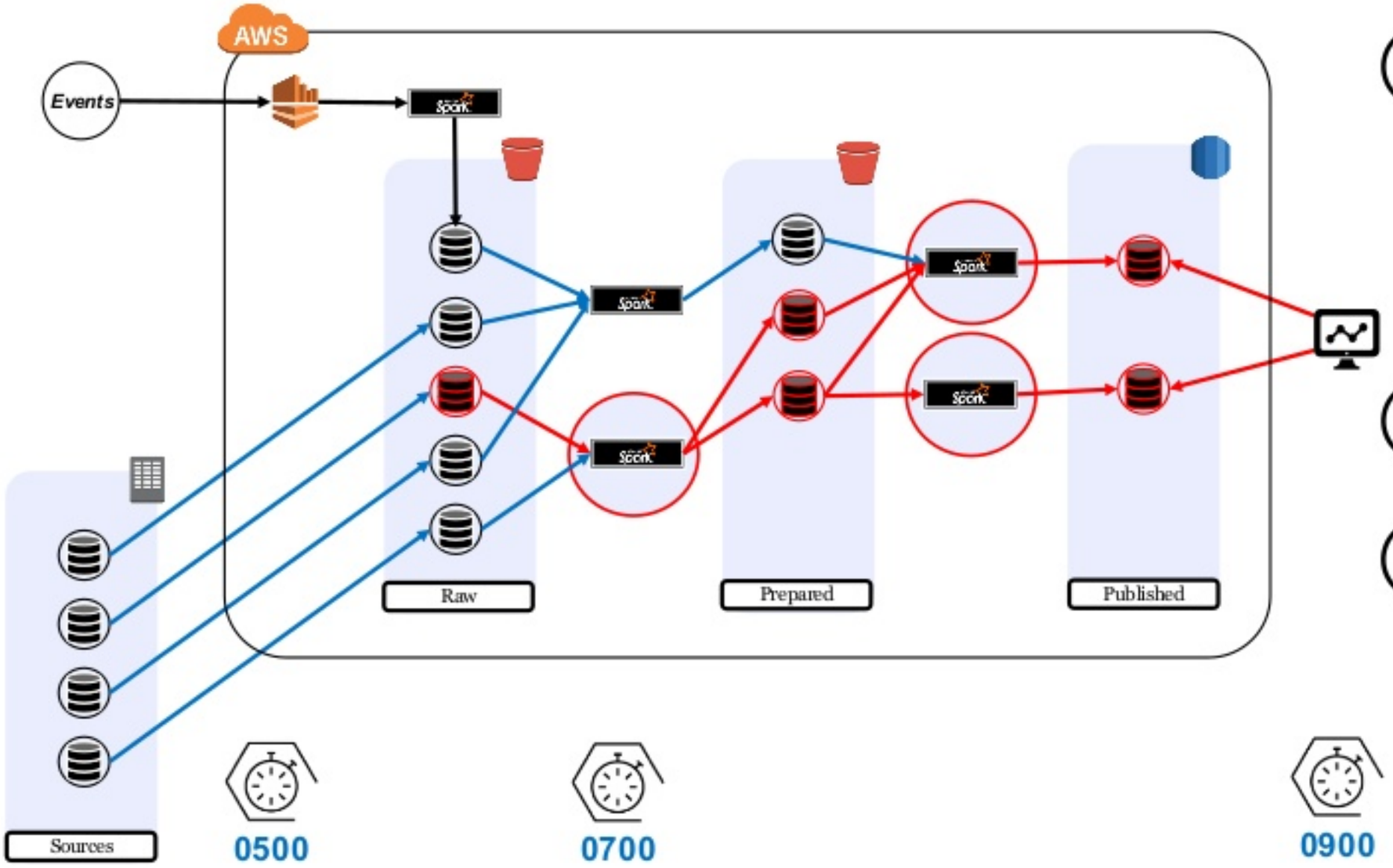
1. Agree a schedule with client when to send data
2. Configure schedule to kick off Spark Jobs
3. Subsequent Spark Jobs kick off when previous have completed

Pipelines – Example – Challenge 1 – Untimely Data



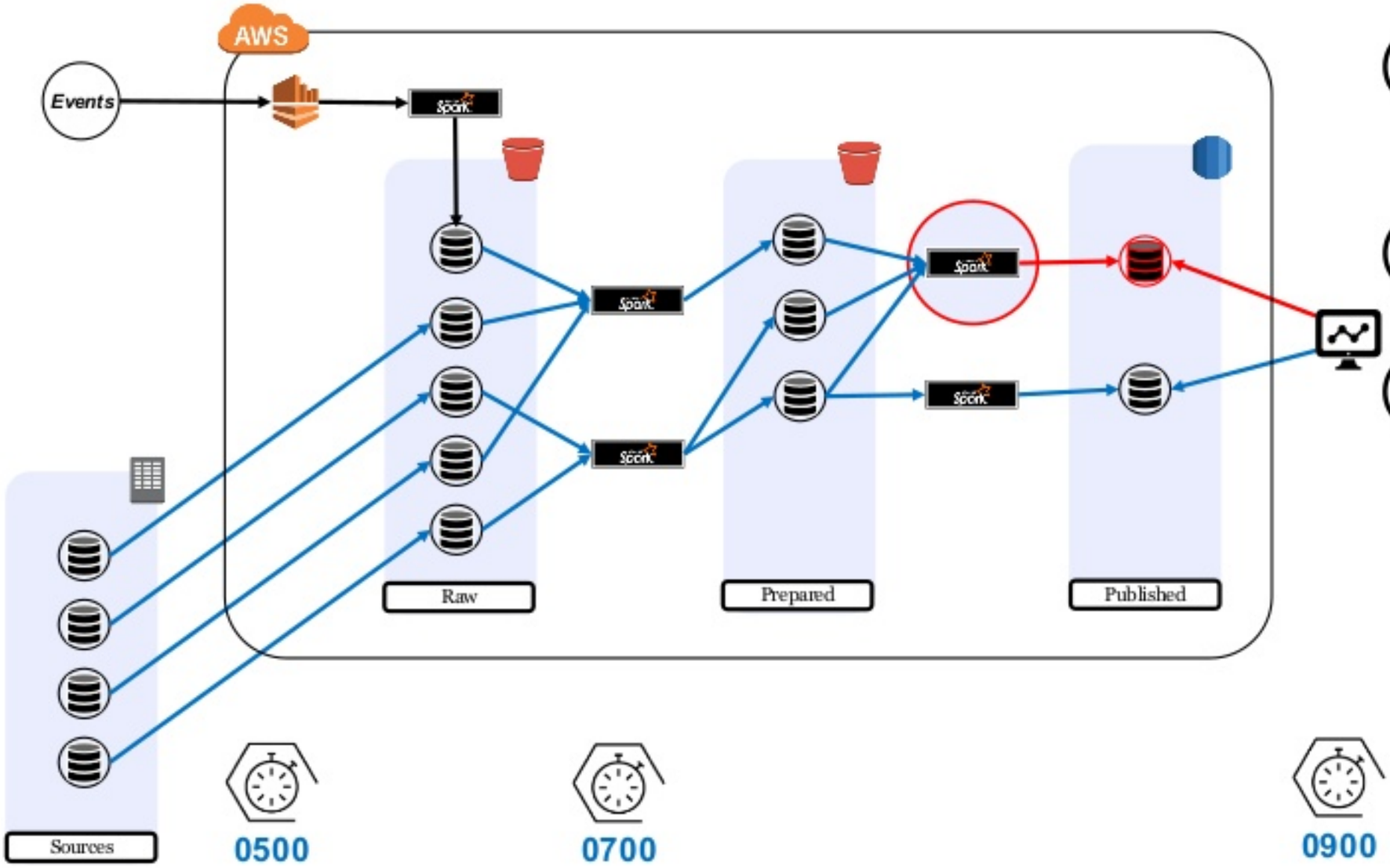
1. Data failed to be sent up on time
2. Impact to rest of the Pipeline
3. Impact to end user!

Pipelines – Example – Challenge 2 – Bad Data



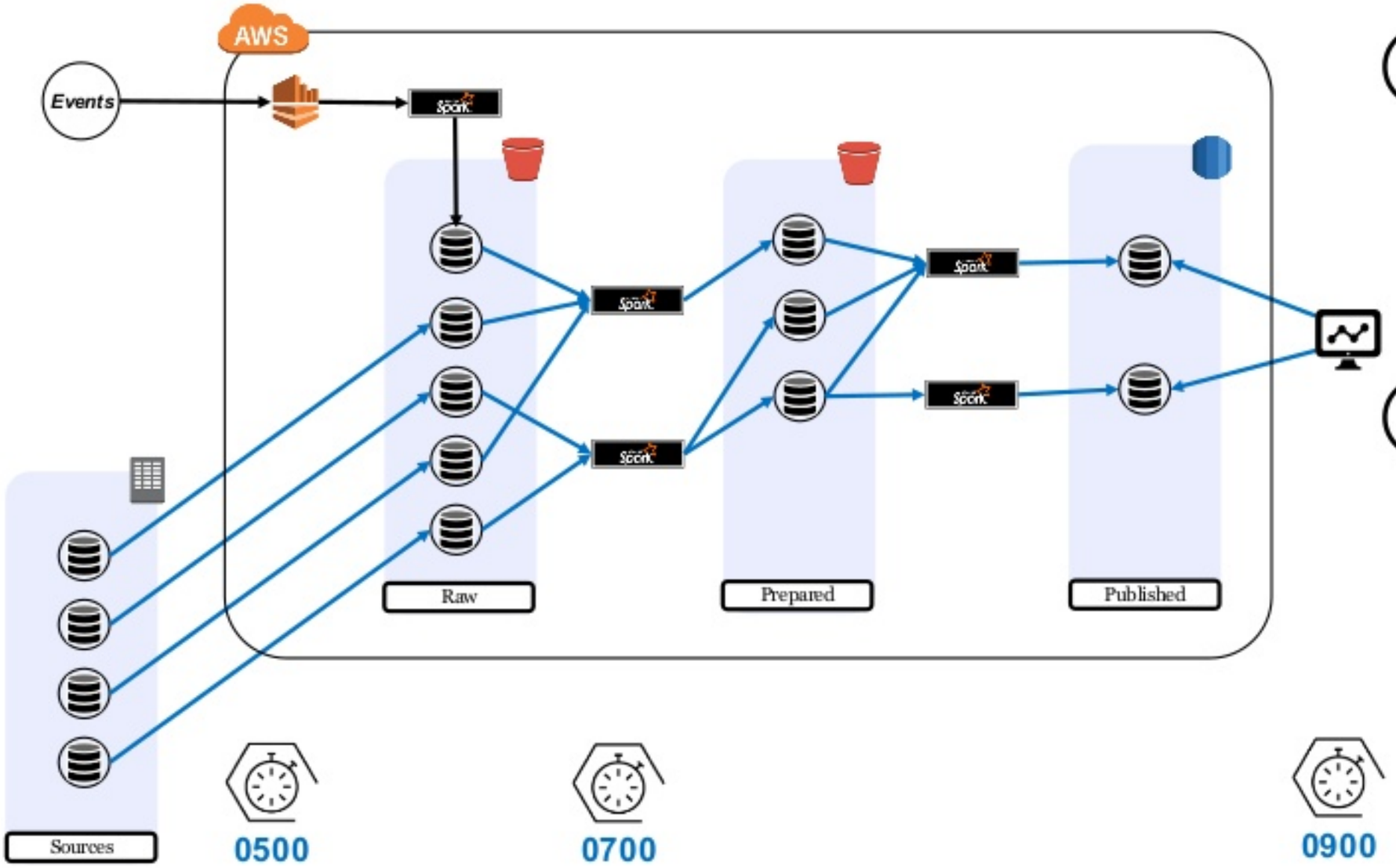
1. Bad/Unexpected Data
   - Schema change
   - Change in Business rules
   - Data Quality Issue
2. Impact to Pipeline
3. Impact to end user!

Pipelines – Example – Challenge 3 – Jobs Taking Longer



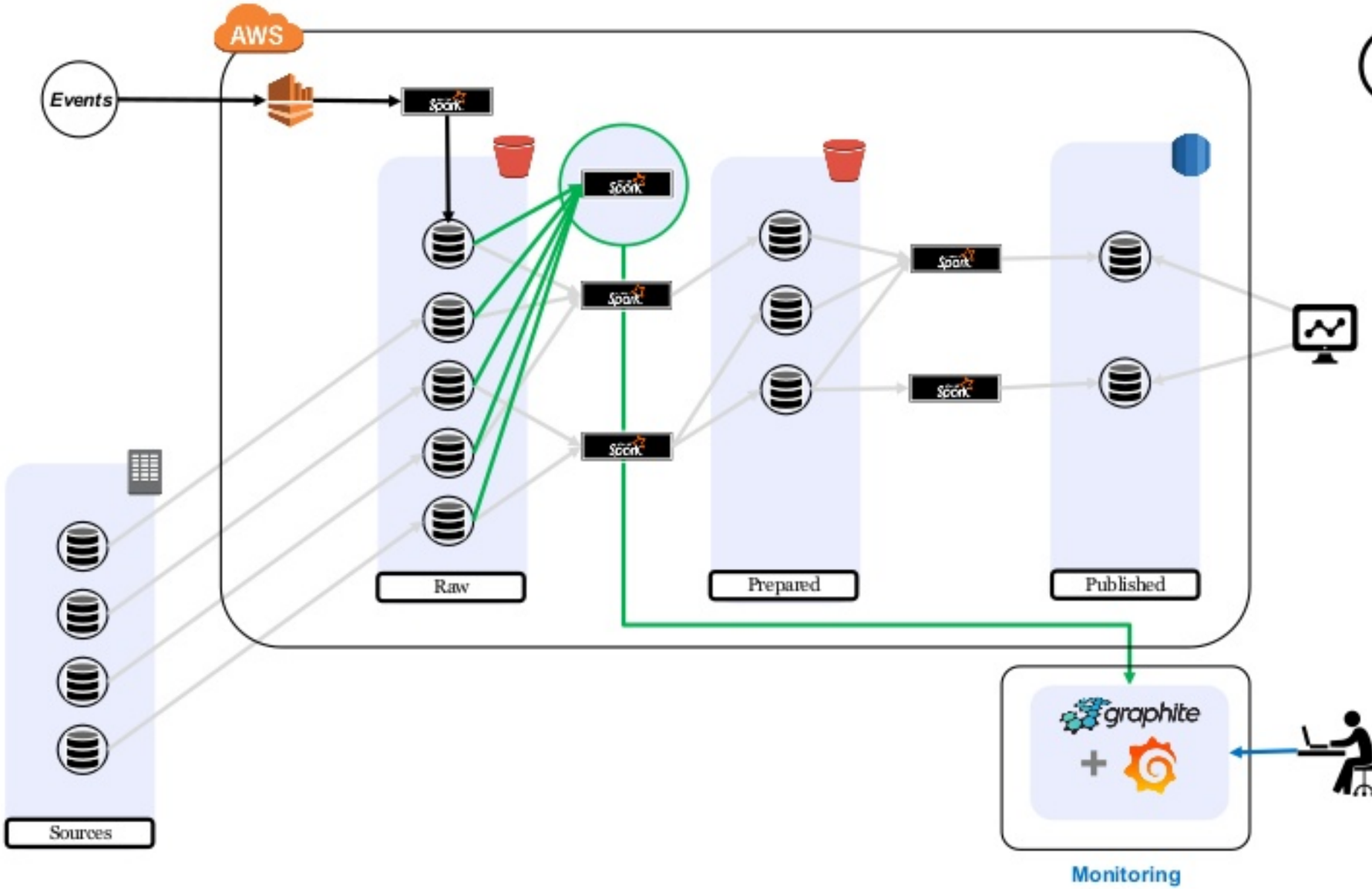
1. Spark Job's taking longer and longer
2. Impact to Pipeline
3. Impact to end user!

Pipelines – Example - Challenges



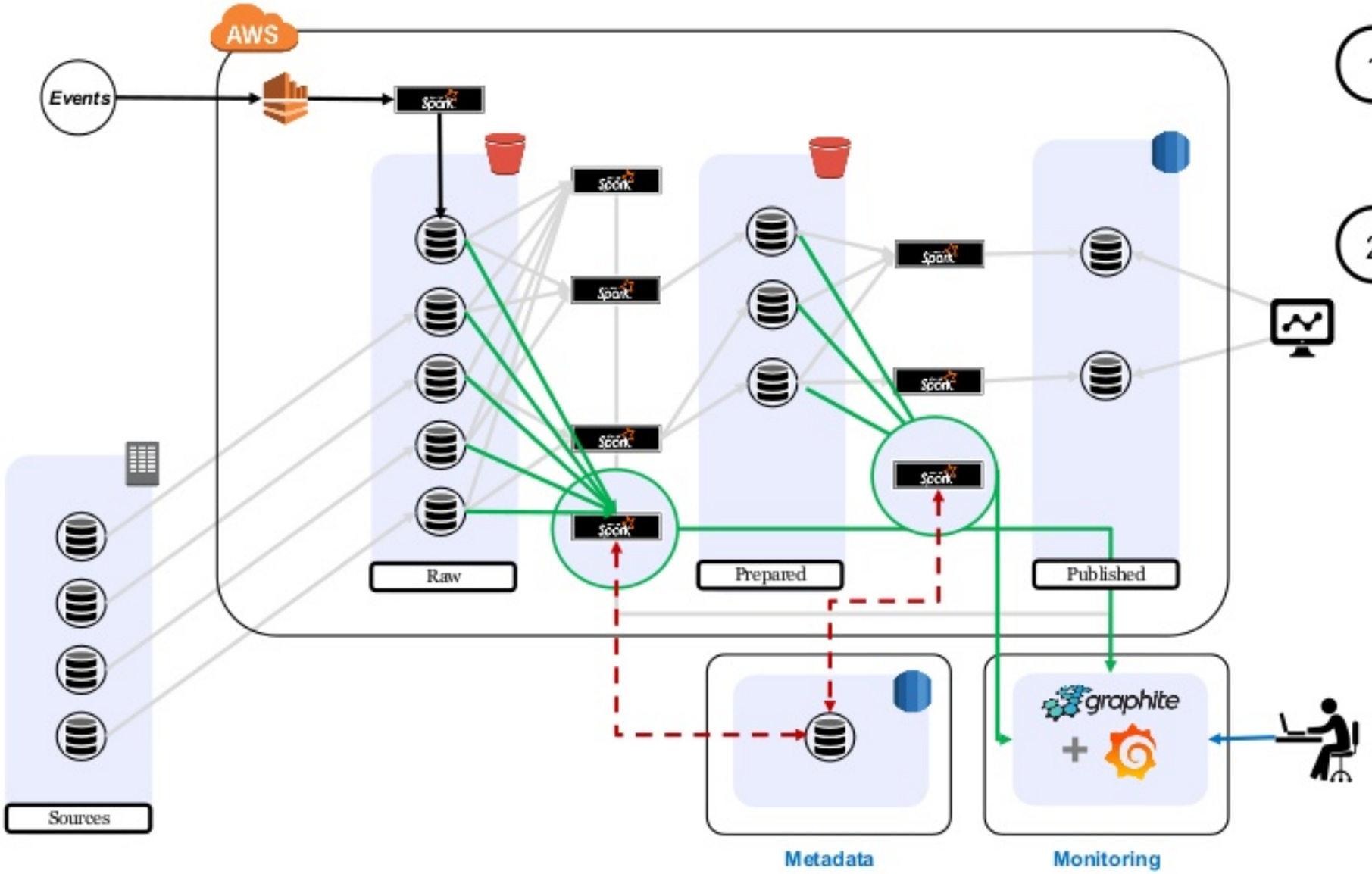
1. We need to
   - Monitor
   - Understand
   - React
2. It looks simple, but in fact its ***not***

Pipelines – Example – Our Approach



1 Monitor all Raw sources for timely data

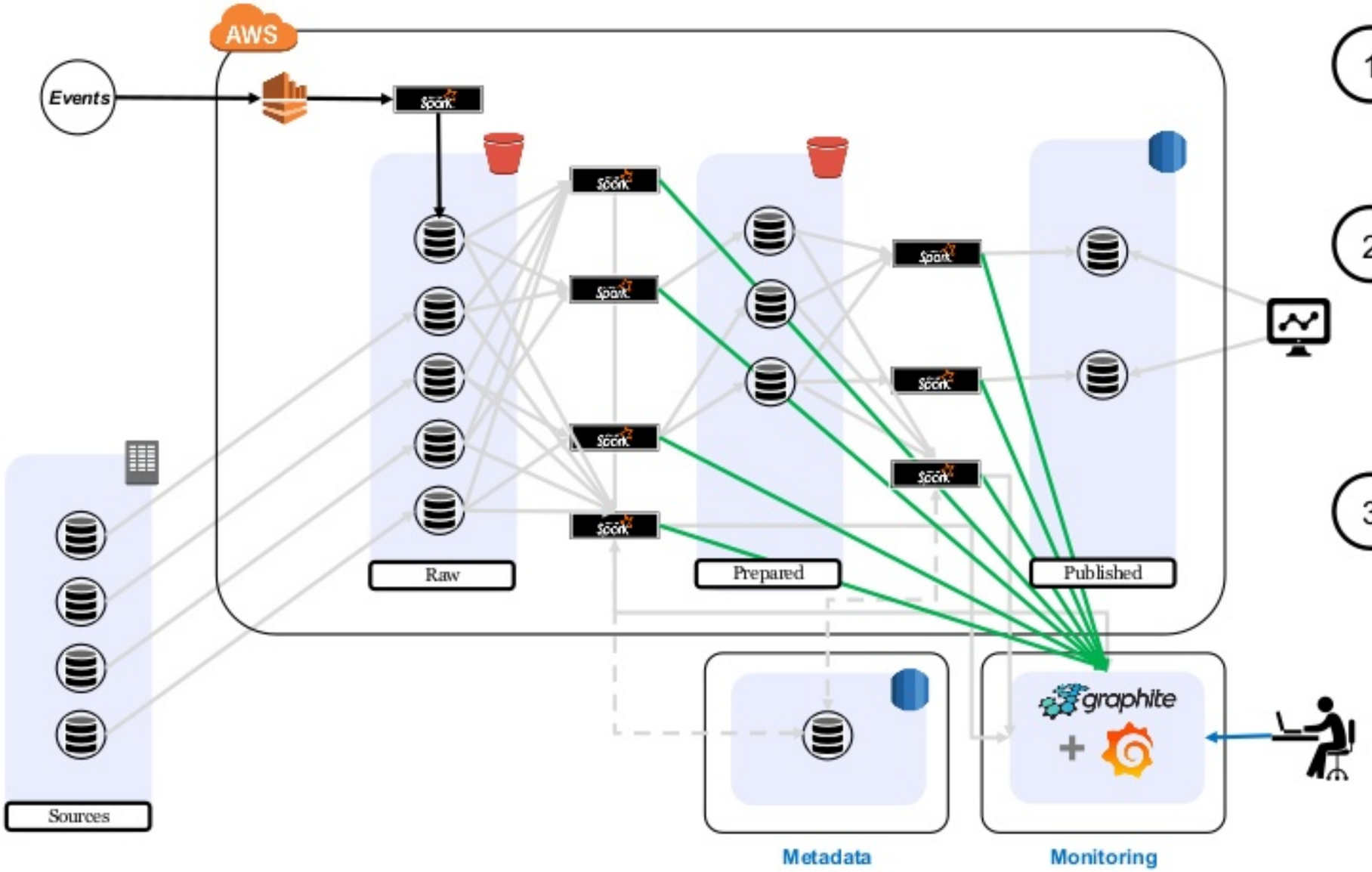
Pipelines – Example – Our Approach



1. Monitor all Raw sources for timely data

2. Monitor quality of data
   - Config driven
   - Agreed business rules

Pipelines – Example – Our Approach



1. Monitor all Raw sources for timely data

2. Monitor quality of data
   - Config driven
   - Agreed business rules

3. Monitor Spark Metrics
   - At least how long
   - Additional Metrics incl:
     - Intent to load data ☺
     - Saving data ☺

Recap

1 – Observations of using Cloud Object Storage

2 – Design To Operate

3 – Pipelines!

Recap

1 – Observations of using Cloud Object Storage

2 – Design To Operate

3 – Pipelines!

Lets Talk!