W UNIVERSITY of WASHINGTON

BIGDATA 210: Introduction to Data Engineering

Autumn 2018

Module 5: Programming Spark Part III

Jerry Kuch

jkuch@uw.edu

Week 5 Agenda

- VM/Docker/Sandbox Login Issues?
- Last Week: *Programming in Spark Part II*:
  - Partitioning and Shuffling
  - Persistence and Caching
  - Spark SQL, DataFrames, Datasets, Hive integration
  - We'll finish the very last bit off tonight…
- *Programming in Spark Part III*
  - – Shared Variables
  - – Spark UI
  - – Spark SQL Server
  - – Notebooks (Jupyter and Zeppelin)
  - – Summary

First off: Any Docker/VM Issues

- Any Issues?

STARTING AMBARI SERVICES...
(CHANTING:) BLOOD FOR BAAL! BLOOD FOR BAAL!

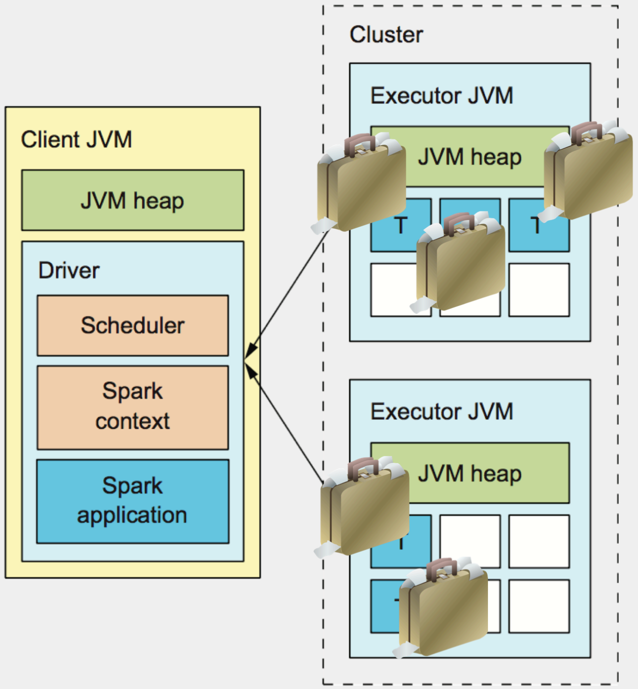
Spark Shared Variables

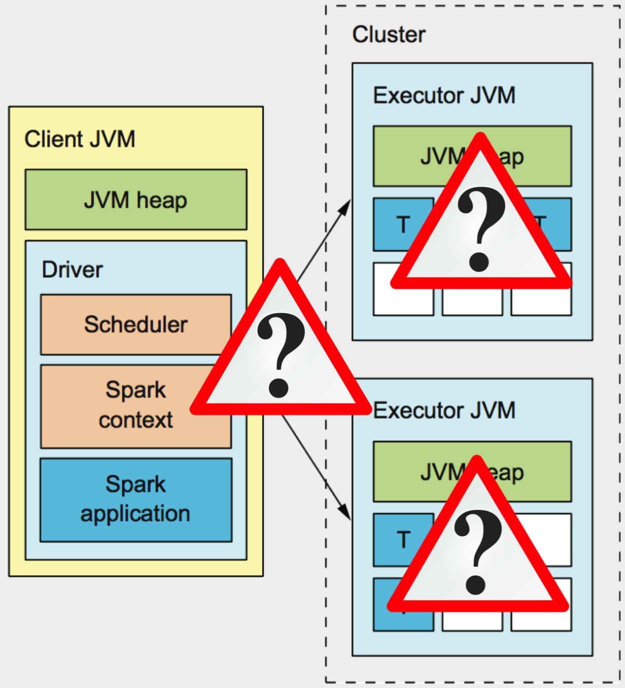
Broadcast Variables

and

Accumulators

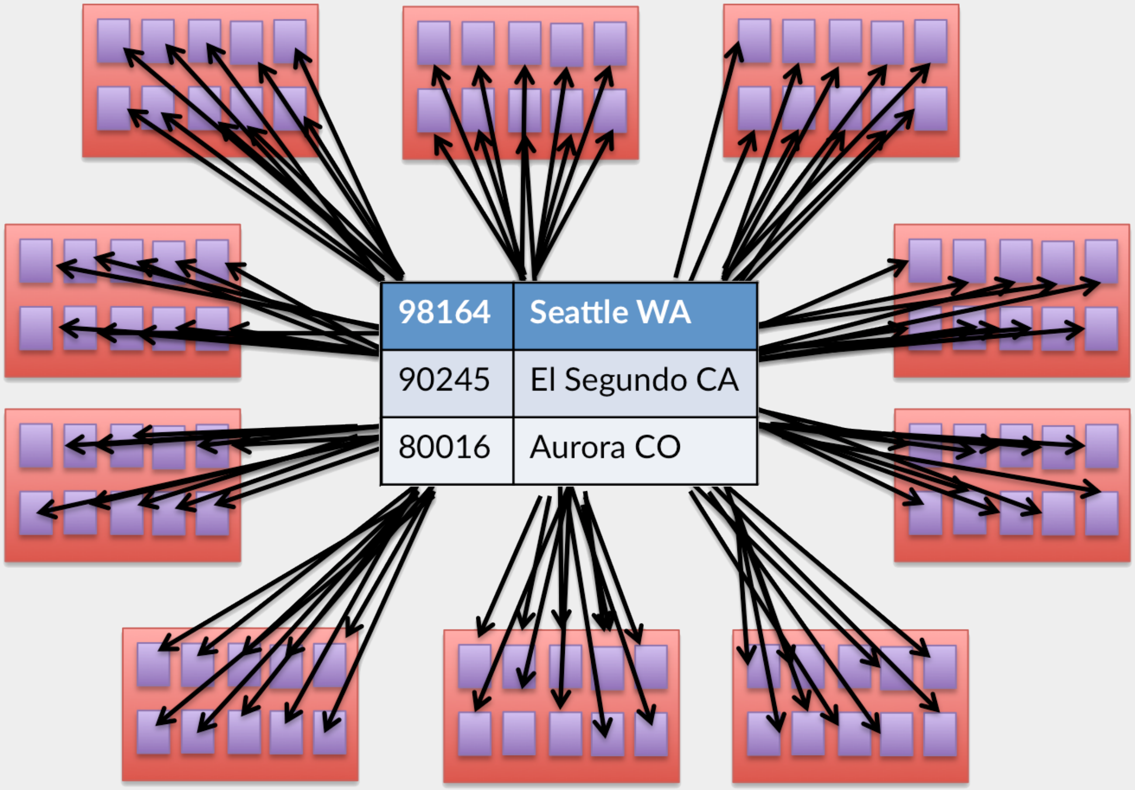
Spark Shared Variables





Broadcast Variables

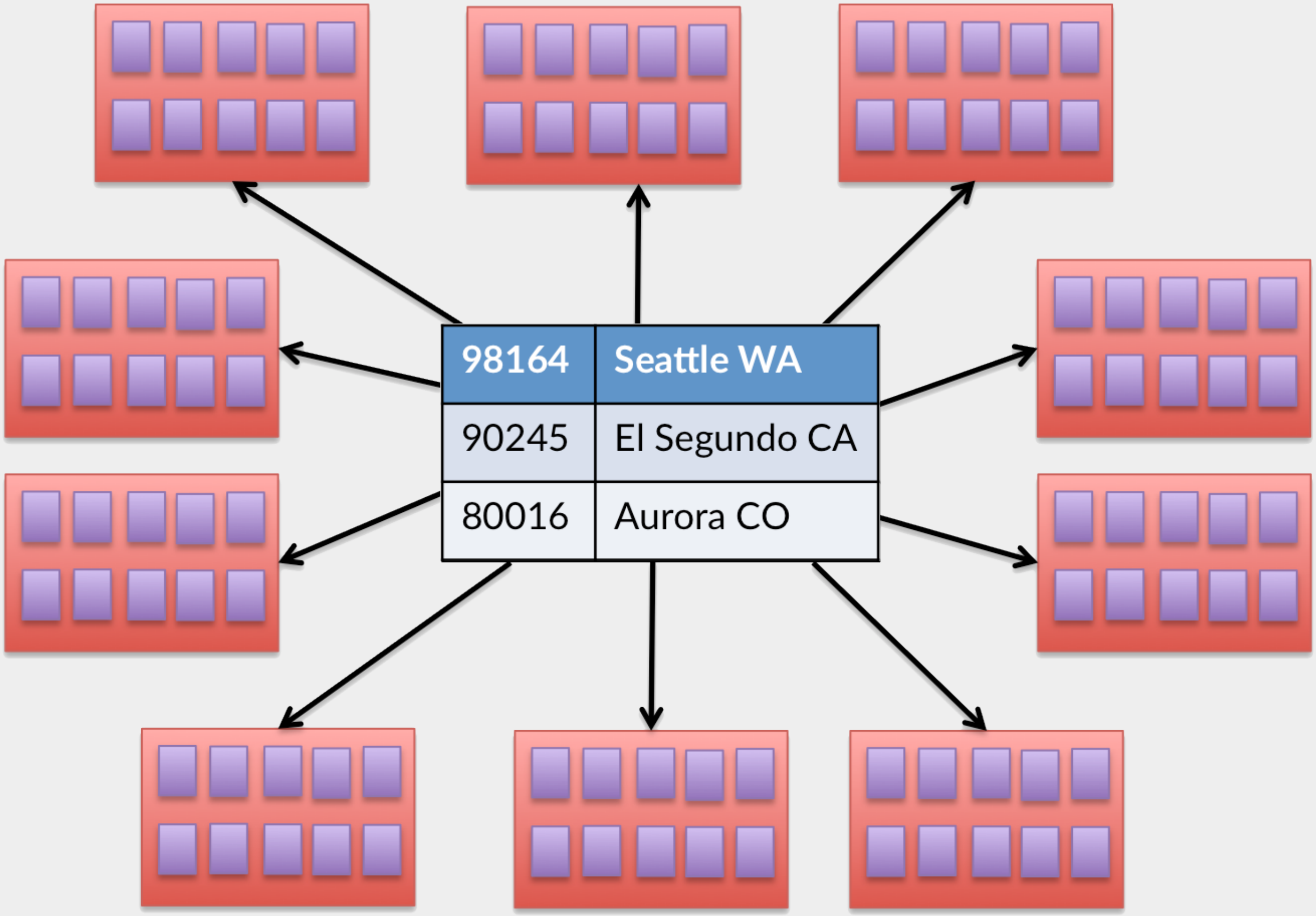
- Cache read-only variable on each node



98164	Seattle WA
90245	El Segundo CA
80016	Aurora CO

e.g. A Map stored as regular Scala Map sent to all tasks individually

Broadcast Variables



98164	Seattle WA
90245	El Segundo CA
80016	Aurora CO

Map stored as broadcast variable and sent only once to each node

Broadcast Variables

```
val broadcastMap = Map("98164" -> "Seattle WA", "90245" -> "El Segundo CA", "80016" -> "Aurora CO")
```

```
val broadcastVar = sc.broadcast(broadcastMap)
```

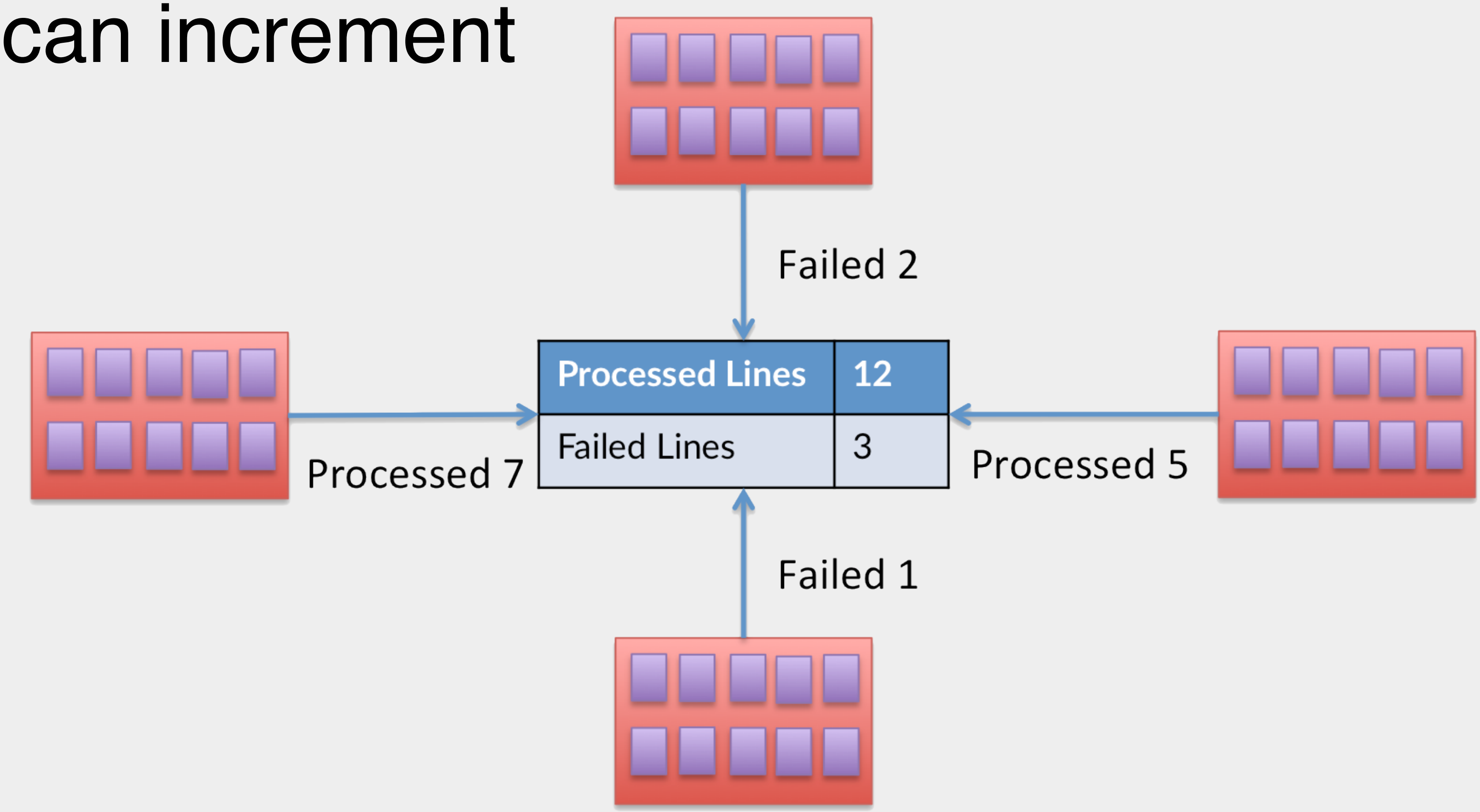
```
val data = sc.parallelize(Array("98164","90245","80016"))
val transformed = data.map(v => (broadcastVar.value(v)))
transformed.collect()
```

```
Array[String] = Array(Seattle WA, El Segundo CA, Aurora CO)
```

```
broadcastVar.unpersist
```

Accumulators

- Shared variable ***on the driver*** that all tasks can increment



Failed 2

Processed 7

Processed Lines	12
Failed Lines	3

Processed 5

Failed 1

Four executors writing to accumulator stored in the driver

Accumulators

```
val processed = sc.longAccumulator("Processed Lines")
val failed = sc.longAccumulator("Failed Lines")
```

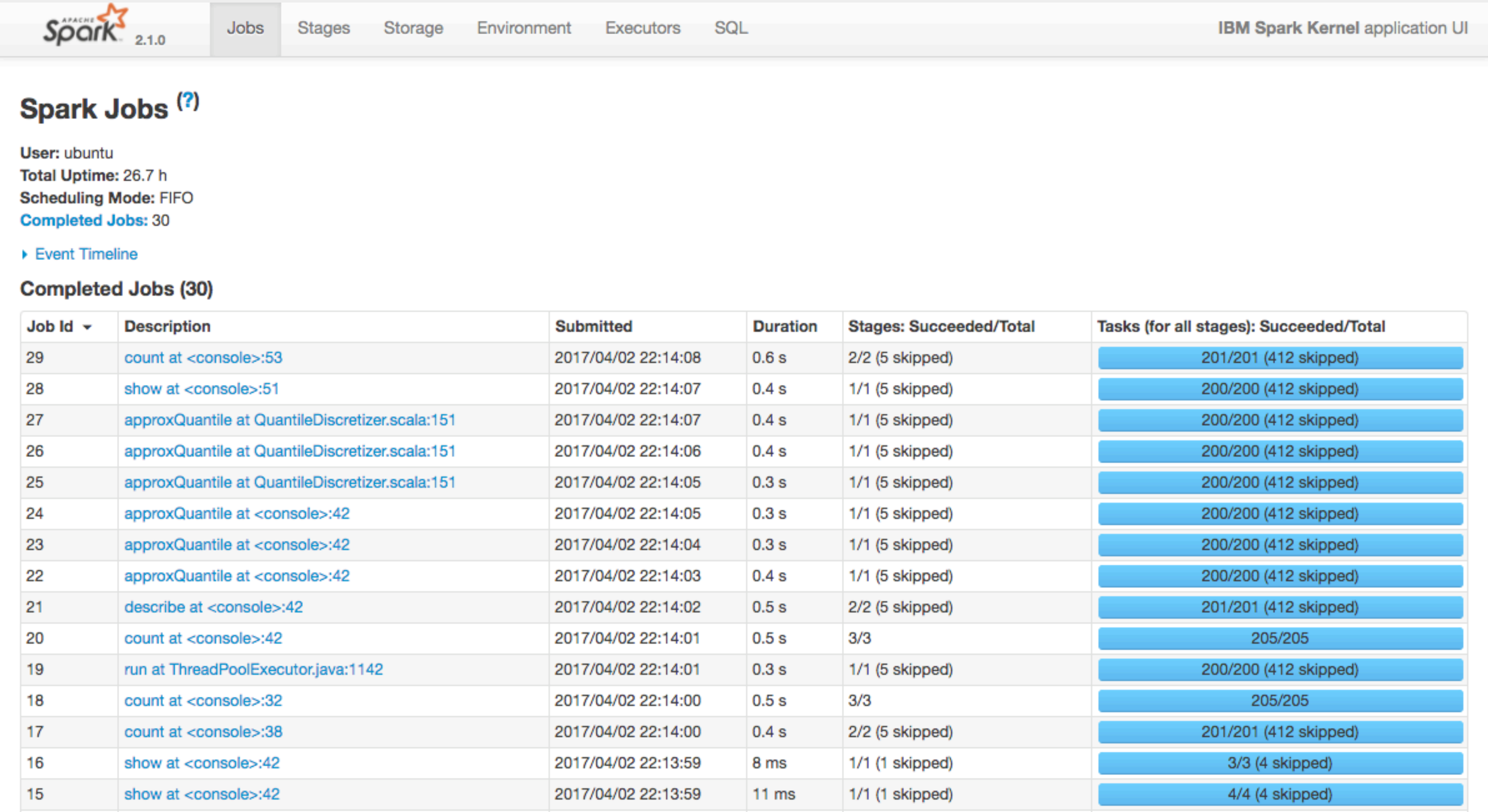
```
val rdd = sc.parallelize(1 to 12)
rdd.foreach(x => processed.add(1))
```

```
processed.value
```

```
Long = 12
```

Spark Web UI

Spark Web UI



Spark Jobs (?)

User: ubuntu
Total Uptime: 26.7 h
Scheduling Mode: FIFO
Completed Jobs: 30

▸ Event Timeline

Completed Jobs (30)

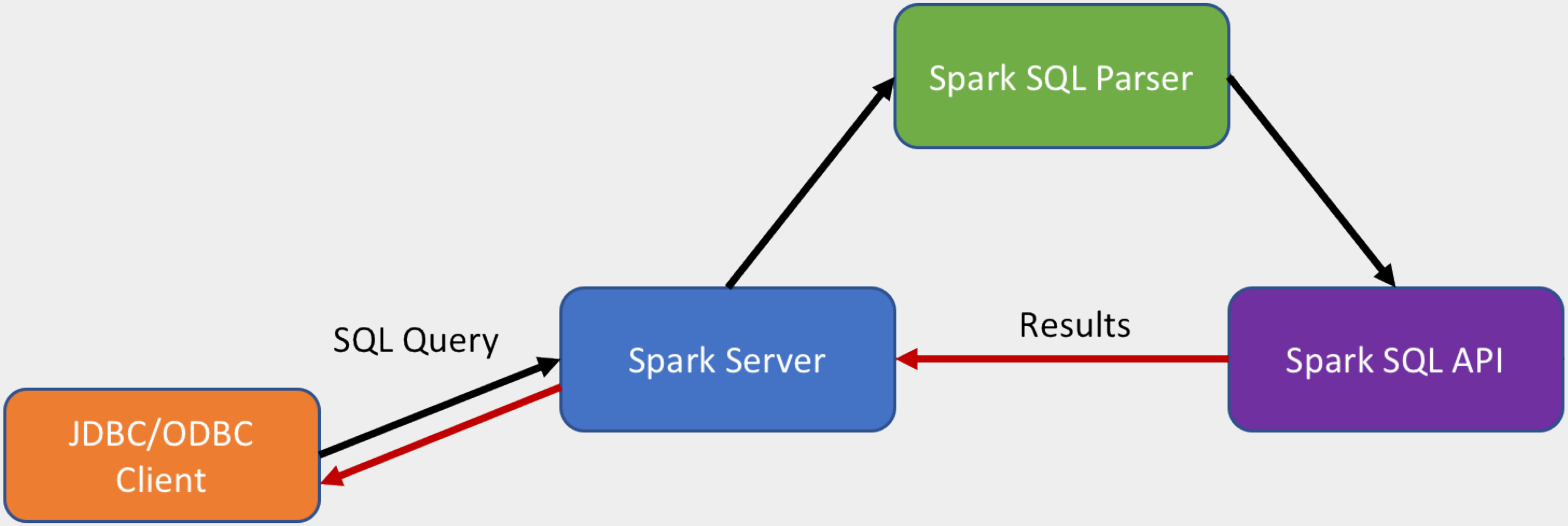
Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
29	count at <console>:53	2017/04/02 22:14:08	0.6 s	2/2 (5 skipped)	201/201 (412 skipped)
28	show at <console>:51	2017/04/02 22:14:07	0.4 s	1/1 (5 skipped)	200/200 (412 skipped)
27	approxQuantile at QuantileDiscretizer.scala:151	2017/04/02 22:14:07	0.4 s	1/1 (5 skipped)	200/200 (412 skipped)
26	approxQuantile at QuantileDiscretizer.scala:151	2017/04/02 22:14:06	0.4 s	1/1 (5 skipped)	200/200 (412 skipped)
25	approxQuantile at QuantileDiscretizer.scala:151	2017/04/02 22:14:05	0.3 s	1/1 (5 skipped)	200/200 (412 skipped)
24	approxQuantile at <console>:42	2017/04/02 22:14:05	0.3 s	1/1 (5 skipped)	200/200 (412 skipped)
23	approxQuantile at <console>:42	2017/04/02 22:14:04	0.3 s	1/1 (5 skipped)	200/200 (412 skipped)
22	approxQuantile at <console>:42	2017/04/02 22:14:03	0.4 s	1/1 (5 skipped)	200/200 (412 skipped)
21	describe at <console>:42	2017/04/02 22:14:02	0.5 s	2/2 (5 skipped)	201/201 (412 skipped)
20	count at <console>:42	2017/04/02 22:14:01	0.5 s	3/3	205/205
19	run at ThreadPoolExecutor.java:1142	2017/04/02 22:14:01	0.3 s	1/1 (5 skipped)	200/200 (412 skipped)
18	count at <console>:32	2017/04/02 22:14:00	0.5 s	3/3	205/205
17	count at <console>:38	2017/04/02 22:14:00	0.4 s	2/2 (5 skipped)	201/201 (412 skipped)
16	show at <console>:42	2017/04/02 22:13:59	8 ms	1/1 (1 skipped)	3/3 (4 skipped)
15	show at <console>:42	2017/04/02 22:13:59	11 ms	1/1 (1 skipped)	4/4 (4 skipped)

http://driver.ip:4040-?

Spark SQL

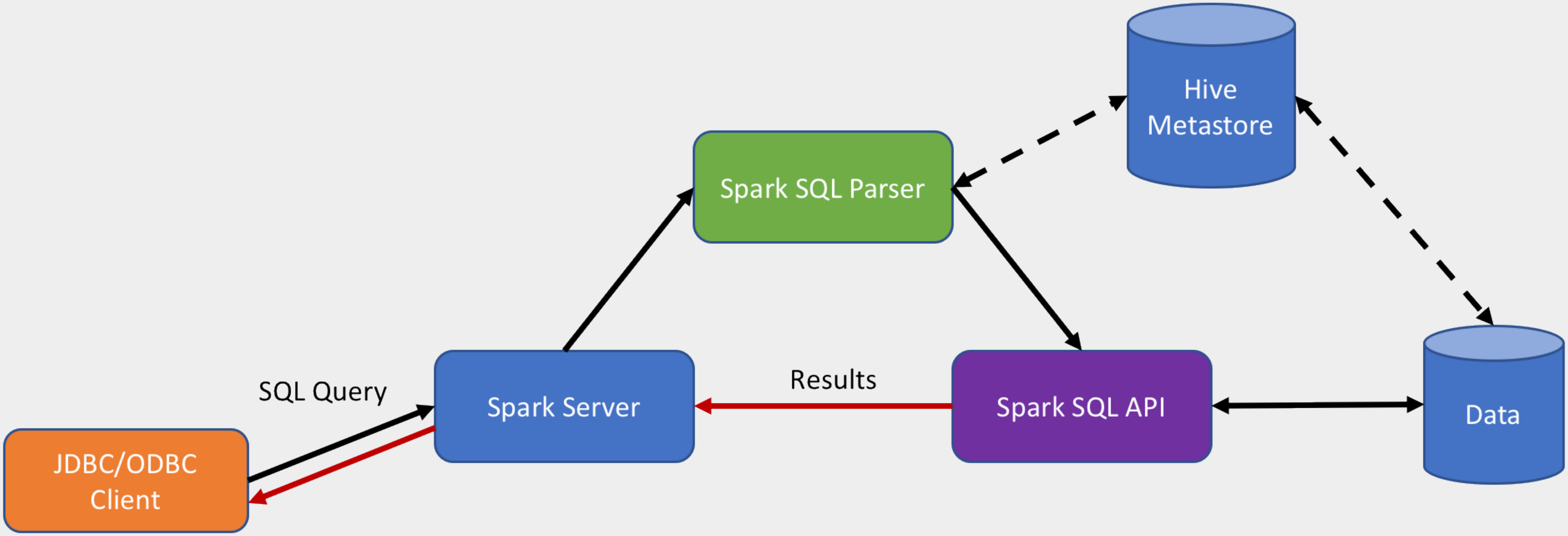
Thrift Server, Hive
and Command Line

Spark SQL Thrift Server



© Jason Kolter

Spark SQL Thrift Server: With the Hive megastore…



Spark SQL Command Line

- Run ‘spark-sql’ in the Sandbox
    - Connects to Hive metastore / Thrift Server
    - Starts up Web UI similar to spark-shell on indicated port ~4040 or so
- Then do SQL queries!
- Observe what happens in the UI

Notebooks

Jupyter and Zeppelin

Notebooks: Jupyter and Zeppelin

- *Notebooks* are another front end to various runtime environments
  - Interact with user via web browser
  - Backend environments provided by *kernels* that expose various runtimes (e.g. Scala, Spark, Python)
- Our universe contains two major players:
  - Jupyter
  - Zeppelin

Jupyter Notebook

- Python based notebook platform
- Formerly "IPython"
- Supports lots of "*kernels*" besides Spark
  - Spark Scala support can be buggy
  - Needs special installation
- To use PySpark, see week 5 assignment for setup and usage

Zeppelin Notebook

- Part of “Hadoop ecosystem” (Apache)
- Already set up in your sandbox
- Accessible through Ambari UI
- Lots of “big data” tools
  - Flink, Cassandra, Beam

Notebooks: Pros and Cons

- **Pros:**
  - Convenient
  - Easy scratch pad or workspace
  - “Self documenting”
- **Cons:**
  - Statefulness can yield confusion, especially w.r.t. intermediate results
  - Editing code is clunky and primitive (compared to good IDEs)
  - Some kernels are buggy

The “Just Enough Scala for Spark” Notebook

- A Jupyter notebook
- Available in a Docker container a few weeks back…

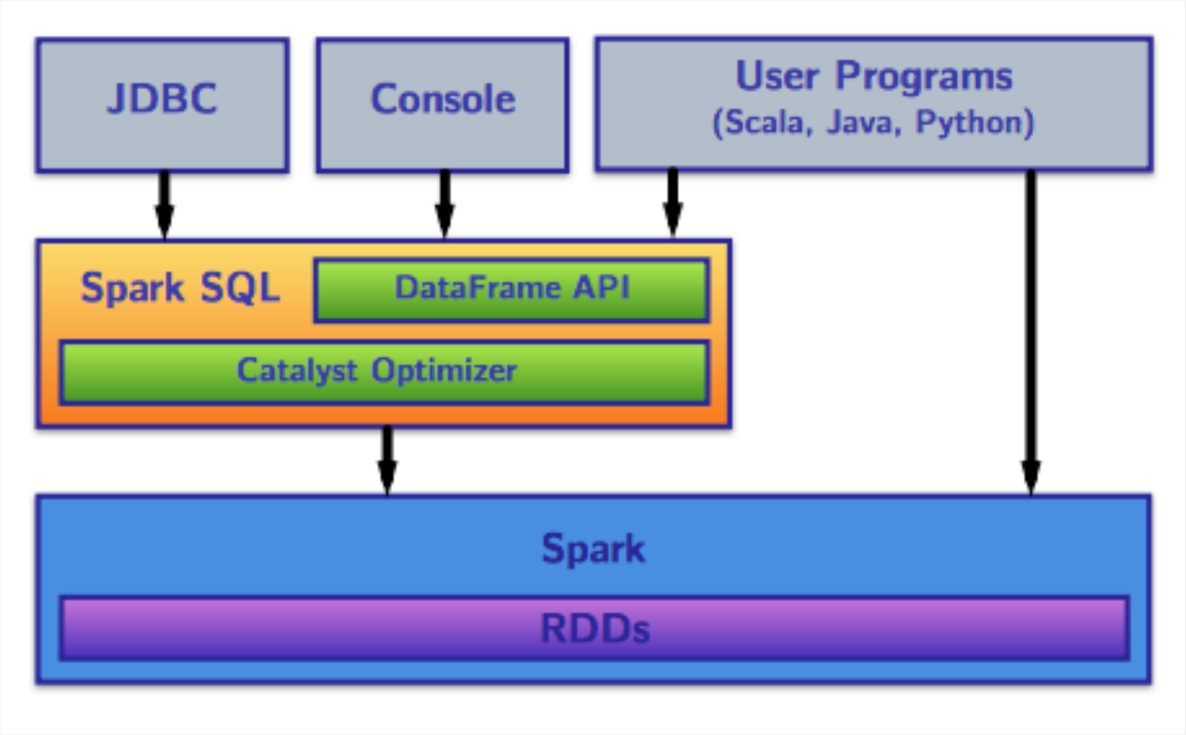
Module 5: Summary

Programming in Spark, Part III Summary (this week and last)

- Spark SQL provides three main things:
    - SQL literal syntax
        - Allows us to mix SQL queries with Scala/Spark
        - Possibility of database style optimizations
    - DataFrames
        - Spark SQL's core abstraction
        - Basically an RDD full of records with a known schema—schema required, unlike with RDDs
        - ***Untyped***, i.e. Scala compiler doesn't check types in the schema! You can hit runtime exceptions!
        - Register a DataFrame as a temporary SQL view to query
    - DataSets
        - Provides Spark SQL options plus ***type safety***
        - In reality: **type** DataFrame = Dataset[**Row**]
        - Are ***typed*** distributed collections of data
        - Require structured/semi-structured data, like DataFrames, but can mix and match DataFrame and RDD APIs
        - DataSets kind of lie "in the middle" between DataFrames and RDDs
            - Can use relational DF operations on them
            - Datasets add more typed operations; and you can use map, flatMap, filter

Programming in Spark, Part III Summary (this week and last)

- Broadcast Variables and Accumulators
- Notebooks: Jupyter and Zeppelin
- Spark SQL's pieces:



Programming in Spark, Part III Summary (this week and last)

- See the references in this week's resources, especially https://spark.apache.org/docs/latest/sql-programming-guide.html
- *Spark in Action*, chapter 5 has good coverage of Spark SQL
- Spark SQL in its modern form, DataFrames and Datasets are fairly new