Spark DataFrames:
Simple and Fast Analytics on Structured Data

Michael Armbrust

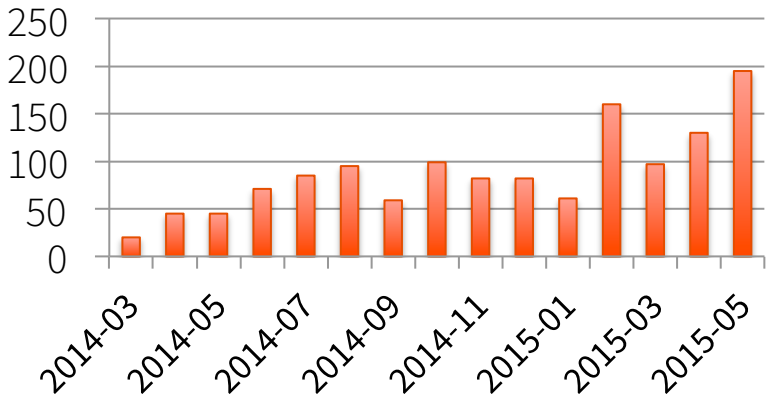
Spark Summit 2015 - June, 15th

About Me and Spark SQL

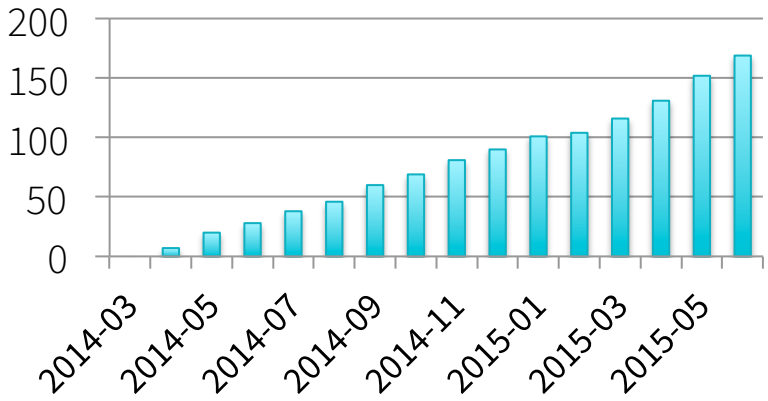
- Spark SQL
  - Part of the core distribution since Spark 1.0 (April 2014)

Graduated from Alpha in 1.3

Of Commits Per Month



# of Contributors



About Me and Spark SQL

- Spark SQL
  - Part of the core distribution since Spark 1.0 (April 2014)
  - Runs SQL / HiveQL queries, optionally alongside or replacing existing Hive deployments

Improved multi-version support in 1.4


```
SELECT COUNT(*)
FROM hiveTable
WHERE hive_udf(data)
```

About Me and Spark SQL

- Spark SQL
  - Part of the core distribution since Spark 1.0 (April 2014)
  - Runs SQL / HiveQL queries, optionally alongside or replacing existing Hive deployments
  - Connect existing BI tools to Spark through JDBC

About Me and Spark SQL

- Spark SQL
  - Part of the core distribution since Spark 1.0 (April 2014)
  - Runs SQL / HiveQL queries, optionally alongside or replacing existing Hive deployments
  - Connect existing BI tools to Spark through JDBC
  - Bindings in Python, Scala, Java, and R

About Me and Spark SQL

- Spark SQL
  - Part of the core distribution since Spark 1.0 (April 2014)
  - Runs SQL / HiveQL queries, optionally alongside or replacing existing Hive deployments
  - Connect existing BI tools to Spark through JDBC
  - Bindings in Python, Scala, Java, and R
- @michaelarmbrust
  - Lead developer of Spark SQL @databricks

The not-so-secret truth...

is about more than SQL.

Spark SQL: The whole story

Creating and Running Spark Programs Faster:

- Write less code
- Read less data
- Let the optimizer do the hard work

DataFrame

noun – [dey-tuh-freym]

1. A distributed collection of rows organized into named columns.
2. An abstraction for selecting, filtering, aggregating and plotting structured data (*cf. R, Pandas*).
3. Archaic: Previously SchemaRDD (*cf. Spark < 1.3*).

Write Less Code: Input & Output

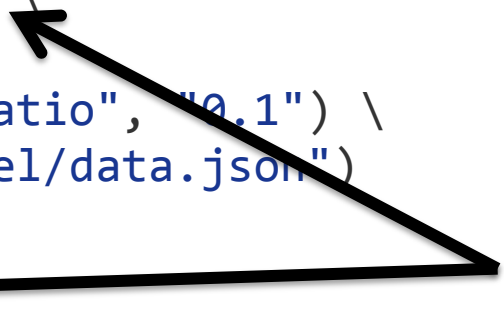
Unified interface to reading/writing data in a variety of formats:

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \
  .saveAsTable("fasterData")
```

Write Less Code: Input & Output

Unified interface to reading/writing data in a variety of formats:

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \
  .saveAsTable("fasterData")
```



`read` and `write` functions create new builders for doing I/O

Write Less Code: Input & Output

Unified interface to reading/writing data in a variety of formats:

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \
  .saveAsTable("fasterData")
```

Builder methods specify:

- Format
- Partitioning
- Handling of existing data

Write Less Code: Input & Output

Unified interface to reading/writing data in a variety of formats:

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \
  .saveAsTable("fasterData")
```

`load(…)`, `save(…)` or `saveAsTable(…)` finish the I/O specification

Write Less Code: Input & Output

Spark SQL's Data Source API can read and write DataFrames using a variety of formats.

Built-In

External

and more…

Find more sources at http://spark-packages.org/

ETL Using Custom Data Sources

```
sqlContext.read
  .format("com.databricks.spark.jira")
  .option("url", "https://issues.apache.org/jira/rest/api/latest/search")
  .option("user", "marmbrus")
  .option("password", "*******")
  .option("query", """
    |project = SPARK AND
    |component = SQL AND
    |(status = Open OR status = "In Progress" OR status = Reopened)""".stripMargin)
  .load()
  .repartition(1)
  .write
  .format("parquet")
  .saveAsTable("sparkSqlJira")
```

Write Less Code: High-Level Operations

Solve common problems concisely using DataFrame functions:

- Selecting columns and filtering
- Joining different data sources
- Aggregation (count, sum, average, etc)
- Plotting results with Pandas

Write Less Code: Compute an Average


```
private IntWritable one =
  new IntWritable(1)
private IntWritable output =
  new IntWritable()
proctected void map(
    LongWritable key,
    Text value,
    Context context) {
  String[] fields = value.split("\t")
  output.set(Integer.parseInt(fields[1]))
  context.write(one, output)
}

IntWritable one = new IntWritable(1)
DoubleWritable average = new DoubleWritable()

protected void reduce(
    IntWritable key,
    Iterable<IntWritable> values,
    Context context) {
  int sum = 0
  int count = 0
  for(IntWritable value : values) {
     sum += value.get()
     count++
    }
  average.set(sum / (double) count)
  context.Write(key, average)
}
```


```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [x.[1], 1])) \
   .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
   .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
   .collect()
```

Write Less Code: Compute an Average

Using RDDs

```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [int(x[1]), 1])) \
    .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
    .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
    .collect()
```

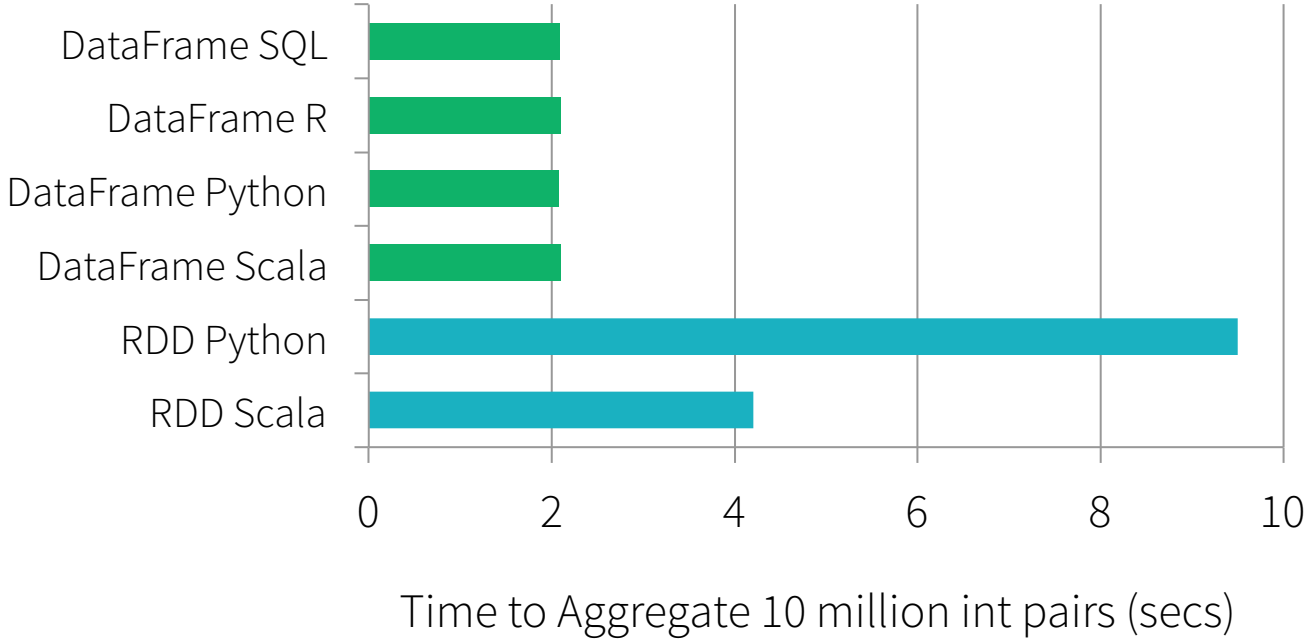
Using DataFrames

```
sqlCtx.table("people") \
    .groupBy("name") \
    .agg("name", avg("age")) \
    .collect()
```

Full API Docs

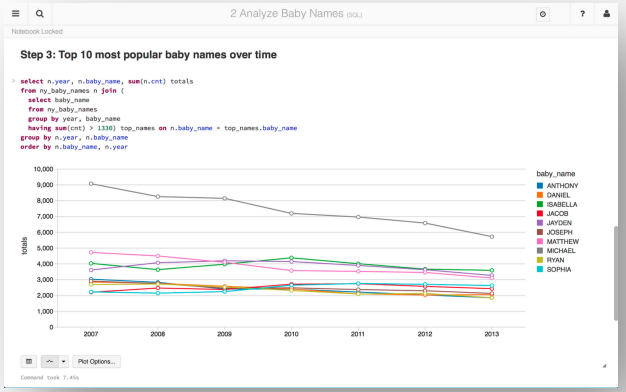
- Python
- Scala
- Java
- R

Not Just Less Code: Faster Implementations



Demo

Combine data from GitHub with data from JIRA



Running in databricks™

- Hosted Spark in the cloud
- Notebooks with integrated visualization
- Scheduled production jobs

https://accounts.cloud.databricks.com/

demo

```
> %run /home/michael/ss.2015.demo/spark.sql.lib ...
```

Start with a Hive Table, created with the code from earlier in the presentation

```
> %sql SELECT * FROM sparkSqlJira
```

		expand: editmeta,renderedFields,transitions,changelog,operations ▼ fields: ▶ aggregateprogress: {"percent":null,"progress":0,"total":0} aggregatetimeestimate: null aggregatetimeoriginalestimate: null aggregatetimespent: null assignee: null ▼ components: ▶ 0: {"description":"Spark SQL module","id":"12322623","name":"SQL","self":"https://issues.apache.org/jira created: 2015-06-14T08:32:33.000+0000 ▶ creator: {"active":true,"avatarUrls":{"16x16":"https://issues.apache.org/jira/secure/useravatar?size=xsmall& customfield_10010: null

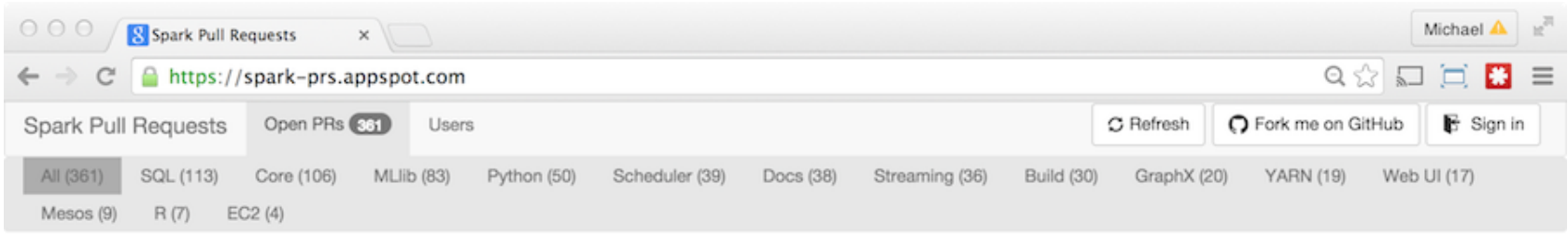
Showing the first 385 rows.

```
Command took 2.08s -- by dbadmin at 6/15/2015, 1:17:07 PM on michael (54 GB)
```

```
>
```

Now, some data from the Spark Pull Request Dashboard

https://spark-prs.appspot.com (https://spark-prs.appspot.com)



Number	JIRAs	Priority	Issue Type	Title	Author	Commenters	Changes	Merges	Jenkins	Updated
6818	8367	↑		Add a limit for 'spark.streaming.blockInterval' since a data loss bug.	SaintBacchus		+6 -1	✔	✔ Passed	2 minutes ago
5659	7067	↑		fix bug when use complex nested fields in ORDER BY	cloud-fan		+67 -64	✔	Asked to test	2 minutes ago
6804	8301	↑		Improve UTF8String substring/startsWith/endsWith/contains performance	tarekauel		+35 -22	✔	✖ Failed	32 minutes ago
6779	8234	↑		misc function: md5	qiansl127		+125 -0	✔	✖ Failed	34 minutes ago
6686	8056	↑		Design an easier way to construct schema for both Scala and Python	ilganeli		+179 -24	✔	→ Running	about an hour ago
6759	8307	↑		improve timestamp from parquet	davies		+163 -240	✔	✔ Passed	about an hour ago
6780	7289	↑		handle project -> limit -> sort efficiently	cloud-fan		+64 -41	✔	→ Running	about an hour ago

Data is exposed as JSON at: https://spark-prs.appspot.com/search-open-prs (https://spark-prs.appspot.com/search-open-prs)

```
> val rawPRs = sqlContext.read
      .format("com.databricks.spark.rest")
      .option("url", "https://spark-prs.appspot.com/search-open-prs")
      .load()
```

```
rawPRs: org.apache.spark.sql.DataFrame = [commenters: array<struct<data:struct<asked_to_close:boolea
n,avatar:string,body:string,date:array<string>,diff_hunk:string,said_lgtm:boolean,url:string>,usernam
e:string>>, components: array<string>, is_mergeable: boolean, jira_issuetype_icon_url: string, jira_i
ssuetype_name: string, jira_priority_icon_url: string, jira_priority_name: string, last_jenkins_comme
nt: struct<body:string,html_url:string,user:struct<login:string>>, last_jenkins_outcome: string, line
s_added: bigint, lines_changed: bigint, lines_deleted: bigint, number: bigint, parsed_title: struct<j
iras:array<bigint>,metadata:string,title:string>, state: string, updated_at: string, user: string]
```

```
Command took 1.95s -- by dbadmin at 6/15/2015, 1:18:46 PM on michael (54 GB)
```

```
> display(rawPRs)
```

(https://github.com/landscapeio/prospector) didn't flag this for me; maybe I need to turn up my linter's strictness.","date":["2015-06-13T18:11:3
\"[error] Could not find\", hadoop_version, \"in the list. Valid options\",\n+ print \"are 'hadoop1.0', 'hadoop2.0', 'hadoop2.2', and 'hadoop2.3'.\"
sys.exit(int(os.environ.get(\"CURRENT_BLOCK\", 255)))\n+\n+ return hadoop_profiles\n+\n+\n+def get_build_profiles(hadoop_version=\"hadoo
base_profiles=True,","said_lgtm":false,"url":"https://github.com/apache/spark/pull/5694#discussion_r32372531"},"username":"JoshRosen"},{
{"asked_to_close":false,"avatar":"https://avatars.githubusercontent.com/u/2230193?v=3","body":"@pwendell @JoshRosen thoughts on the in
minimal, additional set of test checks for MLlib, GraphX, and Streaming.","date":["2015-06-
10T22:19:00Z"],"diff_hunk":"","said_lgtm":true,"url":"https://github.com/apache/spark/pull/5694#issuecomment-110934573"},"username":"b
{"asked_to_close":false,"avatar":"https://avatars.githubusercontent.com/u/1606572?v=3","body":"btw, we are currently setting the PATH varia
as well as in the build config on jenkins. this makes for a sad shane... i'd really like us to only do this once, and if we do it in the build config it'll ta
tests.py. thoughts?","date":["2015-06-10T16:30:41Z"],"diff_hunk":"","said_lgtm":true,"url":"https://github.com/apache/spark/pull/5694#issuec
110823701"},"username":"shaneknapp"},{"data":{"asked_to_close":false,"avatar":"https://avatars.githubusercontent.com/u/320616?v=3","bo
figured we can do it incrementally.\n\nOn Thu, Jun 4, 2015 at 10:32 PM, Josh Rosen \nwrote:\n\n> Given that there's some messiness in reading
think it would be a net reduction of complexity to just include\n> run-tests-jenkins in this refactoring as well?\n>\n> —\n> Reply to this email dir

```
Command took 2.01s -- by dbadmin at 6/15/2015, 1:19:08 PM on michael (54 GB)
```

Slice and dice data with DataFrame operations and UDFs

```
> import org.apache.spark.sql.functions._

  val sparkPRs = rawPRs
    .select(
      // "Explode" nested array to create one row per item.
      explode($"components").as("component"),

      // Use a built-in function to construct the full 'SPARK-XXXX' key
      concat("SPARK-", $"parsed_title.jiras"(0)).as("pr_jira"),

      // Other required columns.
      $"parsed_title.title",
      $"jira_issuetype_icon_url",
      $"jira_priority_icon_url",
      $"number",
      $"commenters",
      $"user",
      $"last_jenkins_outcome",
      $"is_mergeable")
    .where($"component" === "SQL") // Select only SQL PRs
```

```
import org.apache.spark.sql.functions._
sparkPRs: org.apache.spark.sql.DataFrame = [component: string, pr_jira: string, title: string, jira_i
ssuetype_icon_url: string, jira_priority_icon_url: string, number: bigint, commenters: array<struct<d
ata:struct<asked_to_close:boolean,avatar:string,body:string,date:array<string>,diff_hunk:string,sai
d_lgtm:boolean,url:string>,username:string>>, user: string, last_jenkins_outcome: string, is_mergeabl
e: boolean]
```

```
Command took 0.26s -- by dbadmin at 6/15/2015, 1:19:39 PM on michael (54 GB)
```

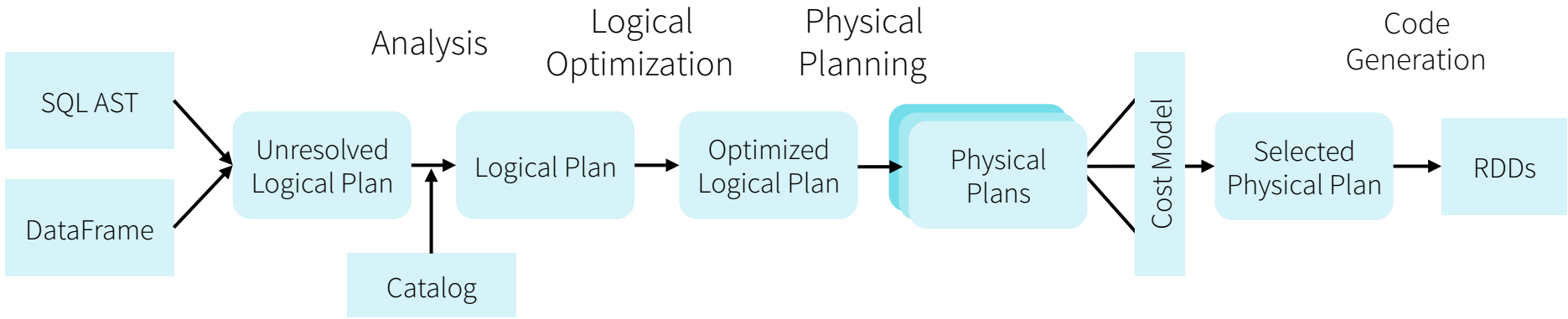
Now we can join the GitHub data with the JIRA data

```
> table("sparkSqlJira")
    .join(sparkPRs, $"key" === $"pr_jira")
    .jiraTable
```

SPARK-8301 (https://issues.apache.org/jira/browse/SPARK-8301)	✘ #6804 (https://github.com/apache/spark/pull/6804)	tarekauel	Improve UTF8String substring/startsWith/endsWith/c performance
SPARK-8056 (https://issues.apache.org/jira/browse/SPARK-8056)	✘ #6686 (https://github.com/apache/spark/pull/6686)	ilganeli	Design an easier way to construc schema for both Scala and Pytho
SPARK-8348 (https://issues.apache.org/jira/browse/SPARK-8348)	✓ #6824 (https://github.com/apache/spark/pull/6824)	yu-iskw	Add in operator to DataFrame Co
SPARK-8247 (https://issues.apache.org/jira/browse/SPARK-8247)	✘ #6762 (https://github.com/apache/spark/pull/6762)	chenghao-intel	string function: instr
SPARK-8363 (https://issues.apache.org/jira/browse/SPARK-8363)	✓ #6823 (https://github.com/apache/spark/pull/6823)	viirya	Move sqrt into math
SPARK-8234 (https://issues.apache.org/jira/browse/SPARK-8234)	✘ #6779 (https://github.com/apache/spark/pull/6779)	qiansl127	misc function: md5
SPARK-8213			

```
Command took 7.55s -- by dbadmin at 6/15/2015, 1:20:15 PM on michael (54 GB)
```

Plan Optimization & Execution



DataFrames and SQL share the same optimization/execution pipeline

Seamlessly Integrated

Intermix DataFrame operations with
custom Python, Java, R, or Scala code

```
zipToCity = udf(lambda zipCode: <custom logic here>)

def add_demographics(events):
    u = sqlCtx.table("users")
    events \
      .join(u, events.user_id == u.user_id) \
      .withColumn("city", zipToCity(df.zip))
```

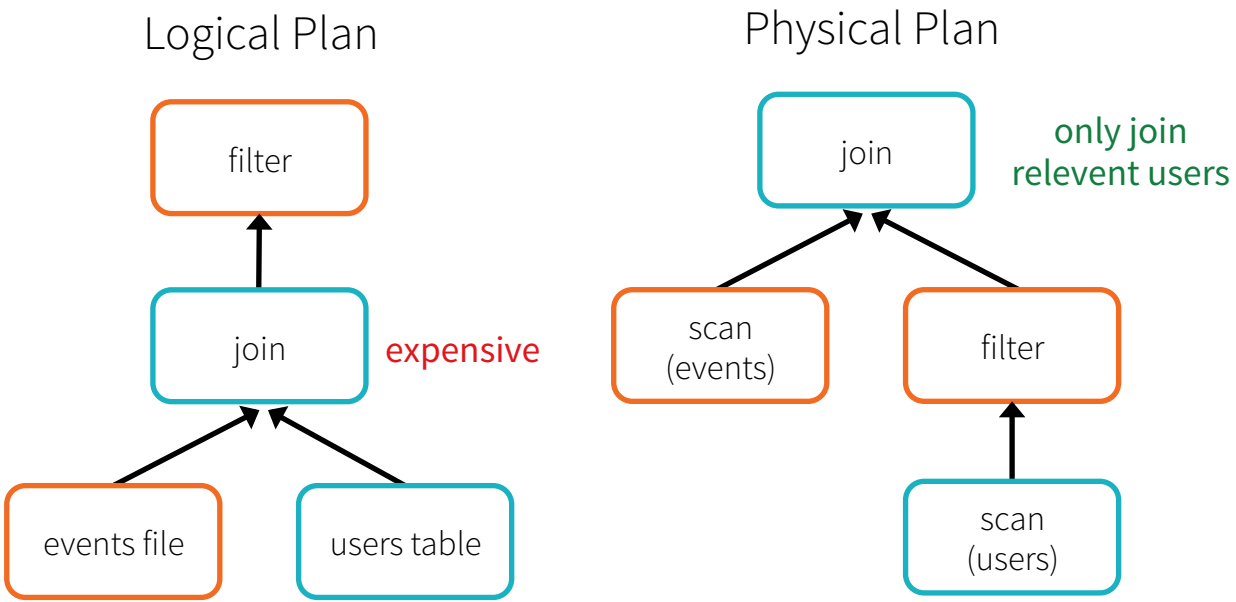
Augments any DataFrame that contains `user_id`

Optimize Entire Pipelines

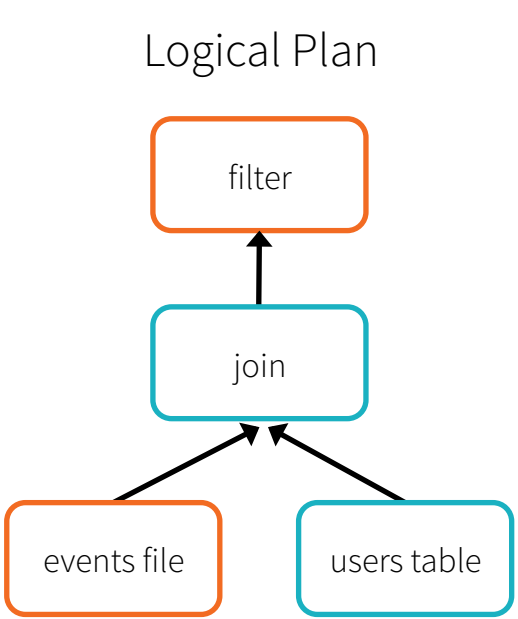
Optimization happens as late as possible, therefore Spark SQL can optimize *even across functions.*

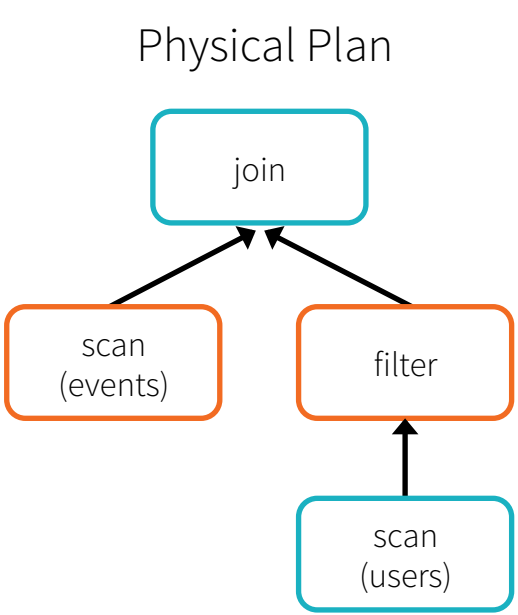
```
events = add_demographics(sqlCtx.load("/data/events", "json"))

training_data = events \
  .where(events.city == "San Francisco") \
  .select(events.timestamp) \
  .collect()
```

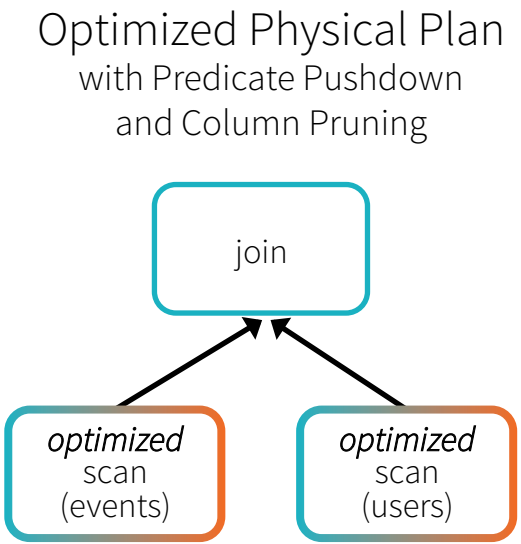
```python
def add_demographics(events):
    u = sqlCtx.table("users")                    # Load Hive table
    events \
      .join(u, events.user_id == u.user_id) \    # Join on user_id
      .withColumn("city", zipToCity(df.zip))     # Run udf to add city column

events = add_demographics(sqlCtx.load("/data/events", "json"))
training_data = events.where(events.city == "San Francisco").select(events.timestamp).collect()
```



```
def add_demographics(events):
    u = sqlCtx.table("users")                    # Load partitioned Hive table
    events \
      .join(u, events.user_id == u.user_id) \    # Join on user_id
      .withColumn("city", zipToCity(df.zip))     # Run udf to add city column

events = add_demographics(sqlCtx.load("/data/events", "parquet"))
training_data = events.where(events.city == "San Francisco").select(events.timestamp).collect()
```

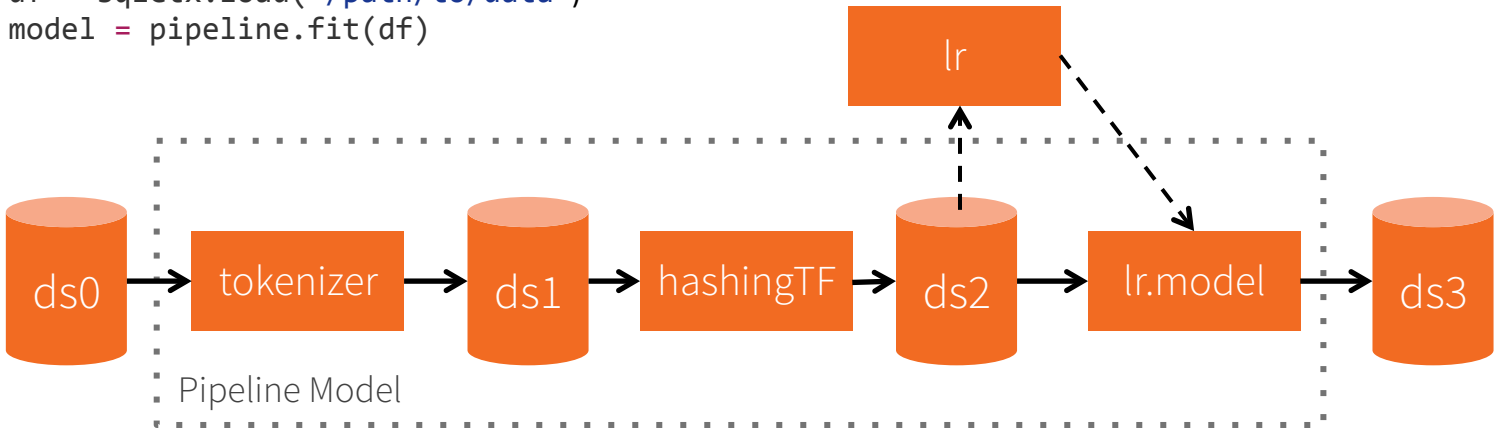






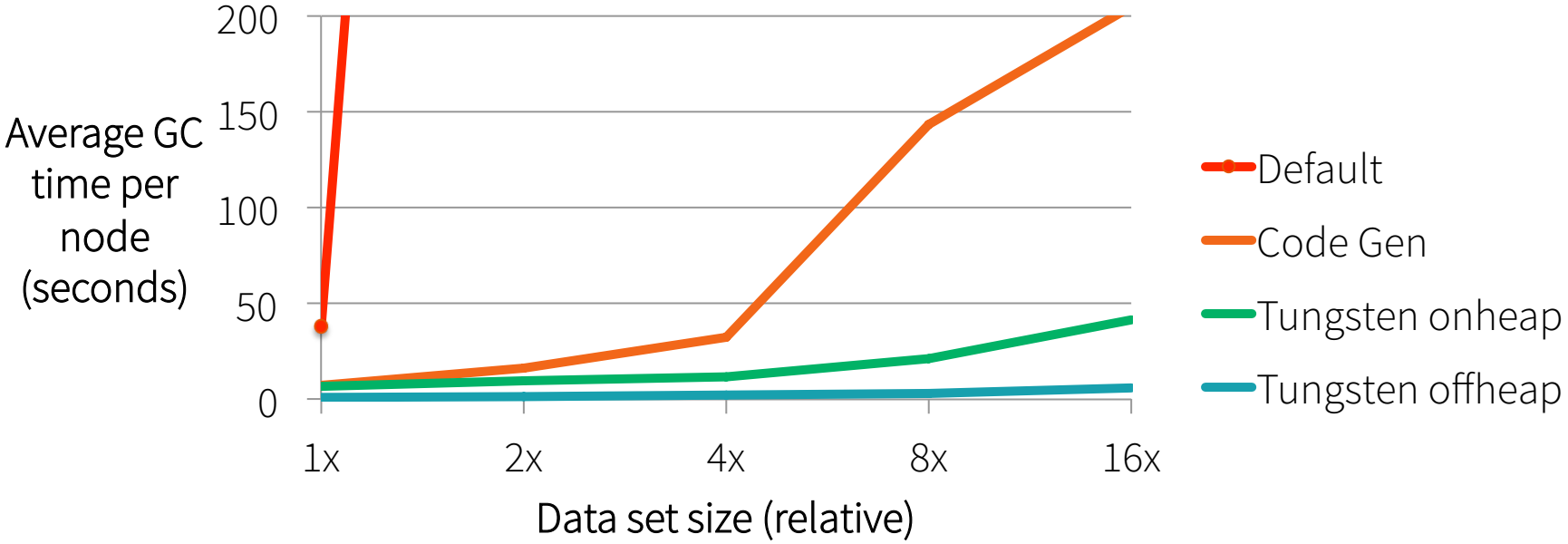
Machine Learning Pipelines

```
tokenizer = Tokenizer(inputCol="text", outputCol="words”)
hashingTF = HashingTF(inputCol="words", outputCol="features”)
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

df = sqlCtx.load("/path/to/data")
model = pipeline.fit(df)
```



Find out more during Joseph’s Talk: 3pm Today

Project Tungsten: Initial Results



Find out more during Josh's Talk: 5pm Tomorrow

Questions?

Spark SQL Office Hours Today

- Michael Armbrust 1:45-2:30
- Yin Huai 3:40-4:15

Spark SQL Office Hours Tomorrow

- Reynold 1:45-2:30