Fast and Reliable Apache Spark SQL Releases

DataWorks Summit Barcelona

March 21st, 2019

databricks

About us

BOGDAN GHIT

Databricks, Software Engineer

- Spark performance

IBM T.J. Watson Research Center

- Research intern on big data
- Bid advisor for cloud spot markets

Delft University of Technology, PhD in Computer Science

- Resource management in datacenters
- Performance of Spark, Hadoop

NICOLAS POGGI

Databricks, Performance Engineer

- Spark benchmarking

Barcelona Supercomputing - Microsoft Research Centre

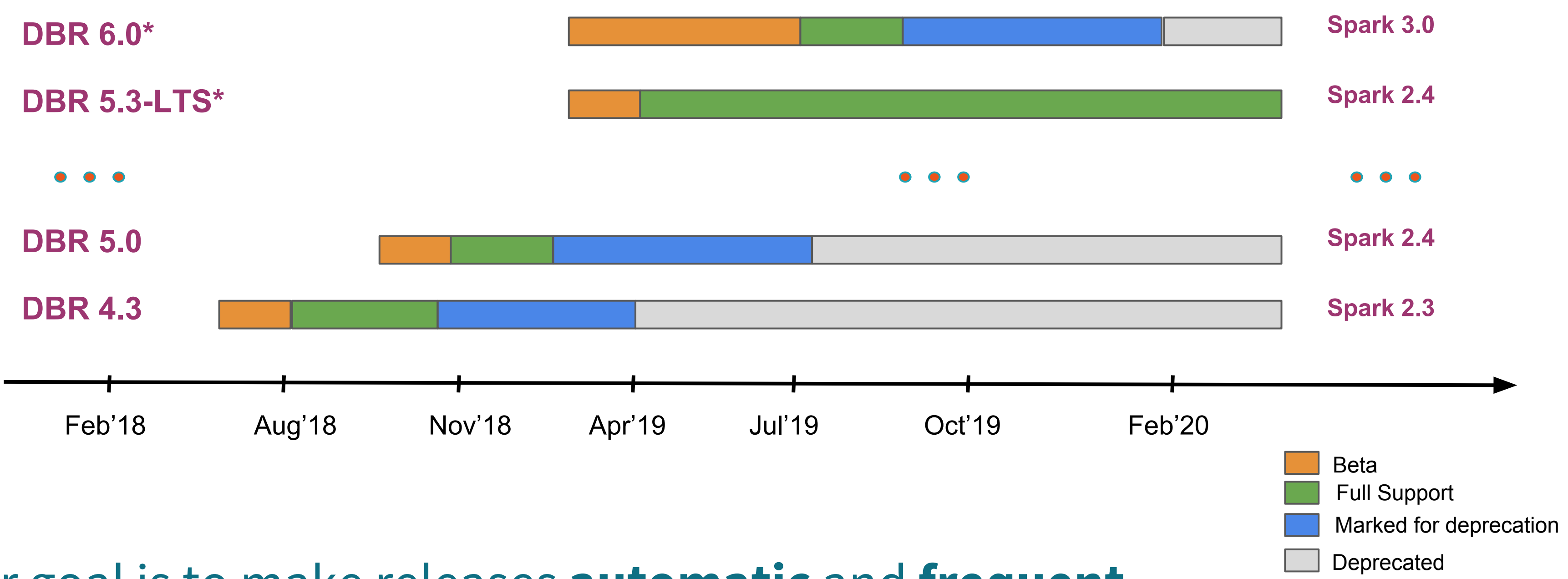
- Lead researcher ALOJA project
- New architectures for Big Data

BarcelonaTech (UPC), PhD in Computer Architecture

- Autonomic resource manager for the cloud
- Web customer modeling

Databricks ecosystem

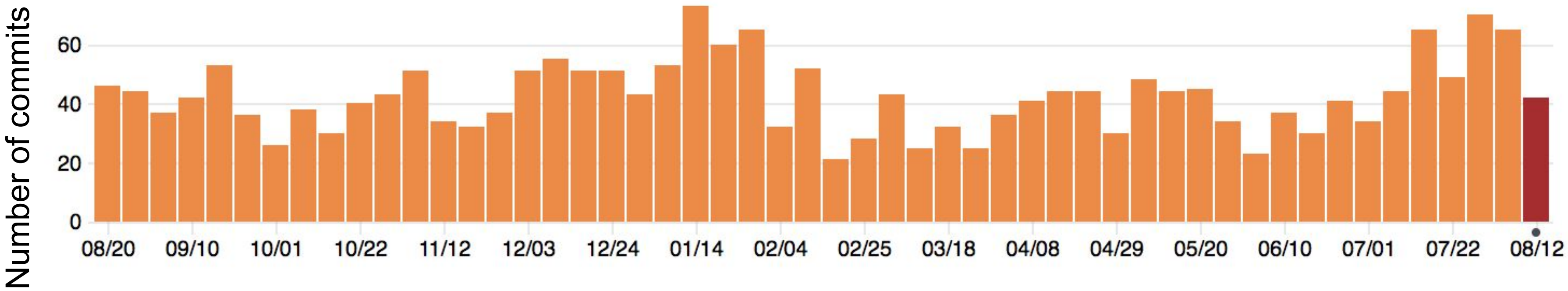
Databricks runtime (DBR) releases



Our goal is to make releases **automatic** and **frequent**

* dates and LTS-tag new releases are subject to change

Apache Spark contributions

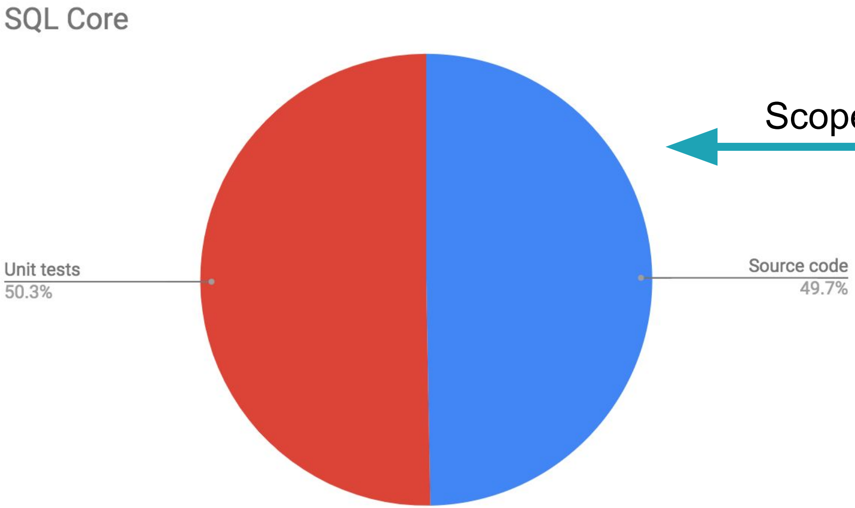


Hundreds of commits monthly to the Apache Spark project

At this pace of development, **mistakes** are bound to happen

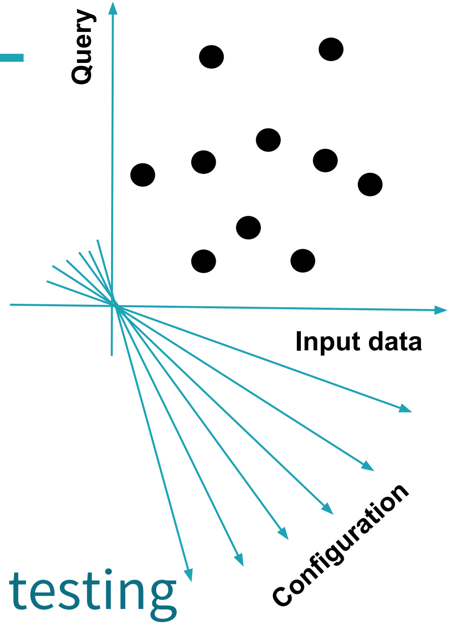
databricks

Where do these contributions go?



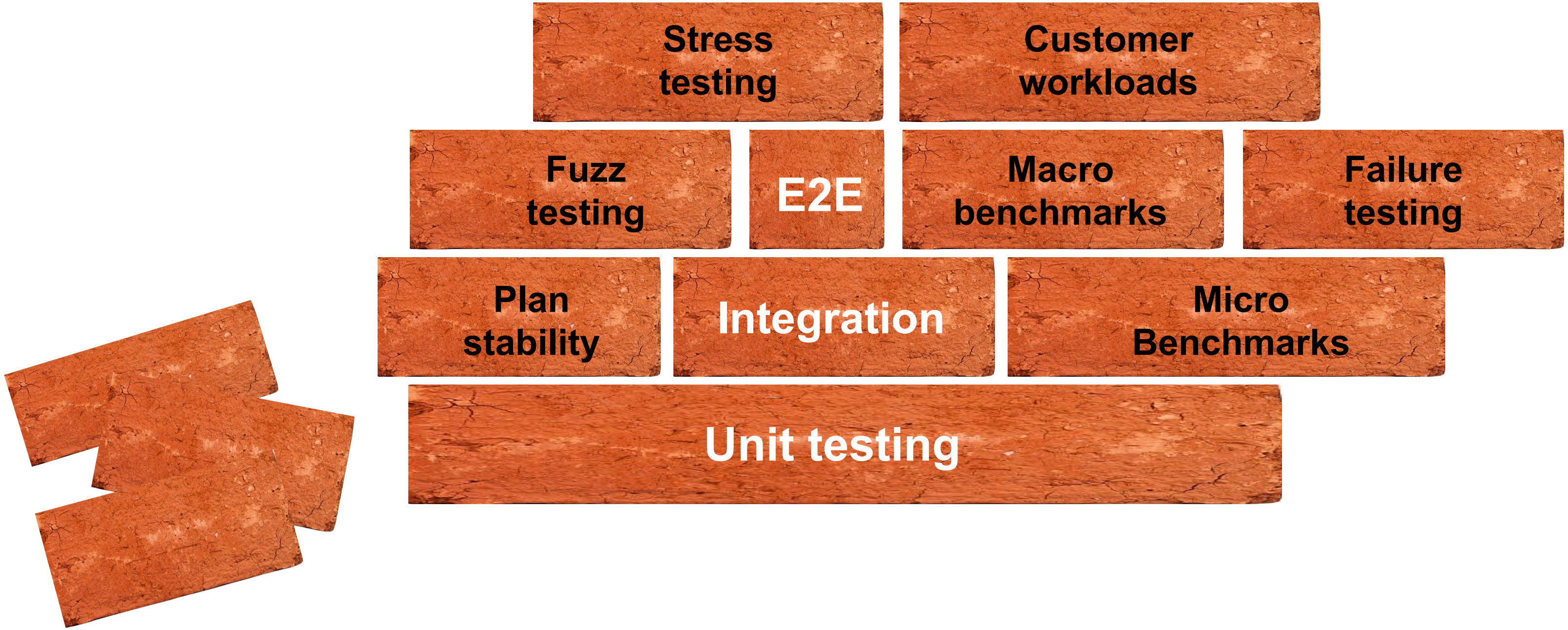
Scope of the testing

Over 200 built-in functions



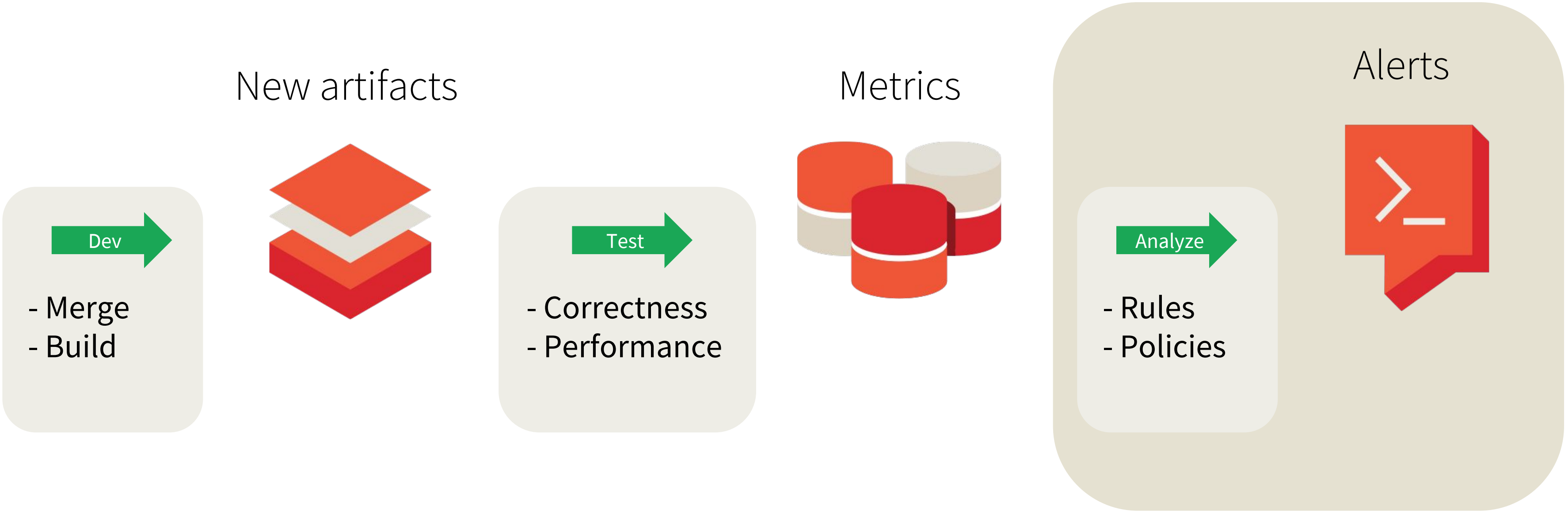
Developers put a significant **engineering effort** in testing

Yet another brick in the wall

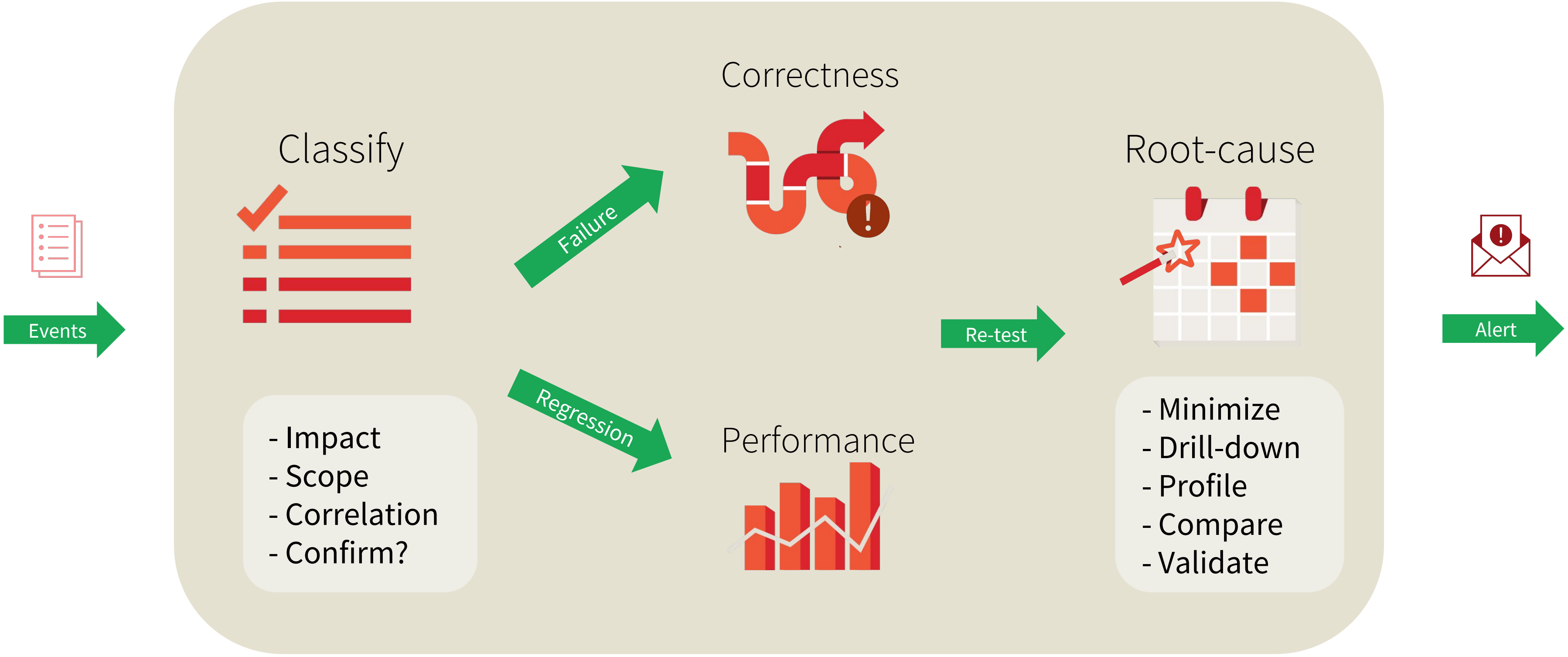


Unit testing ***is not enough*** to guarantee correctness and performance

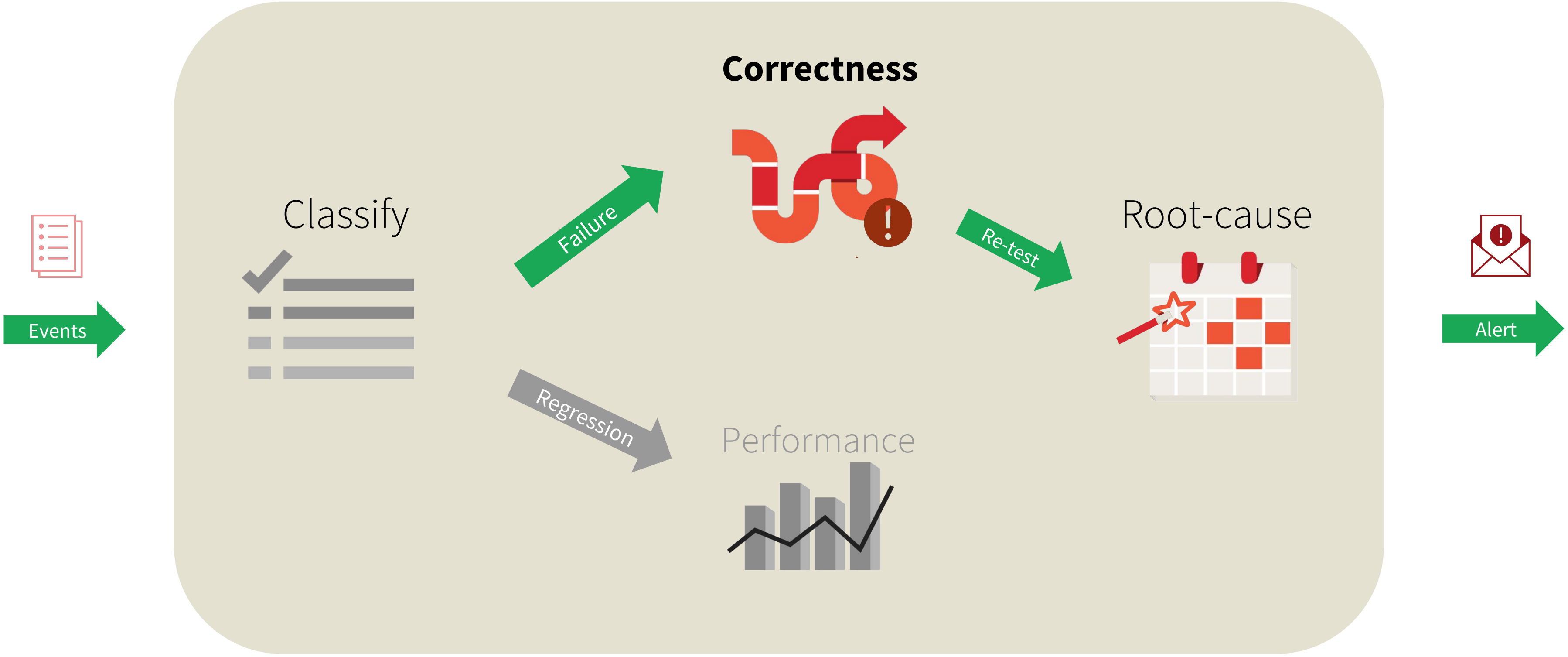
Continuous integration pipeline



Classification and alerting

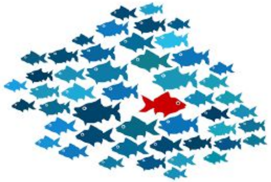


Correctness



How SQLite is tested

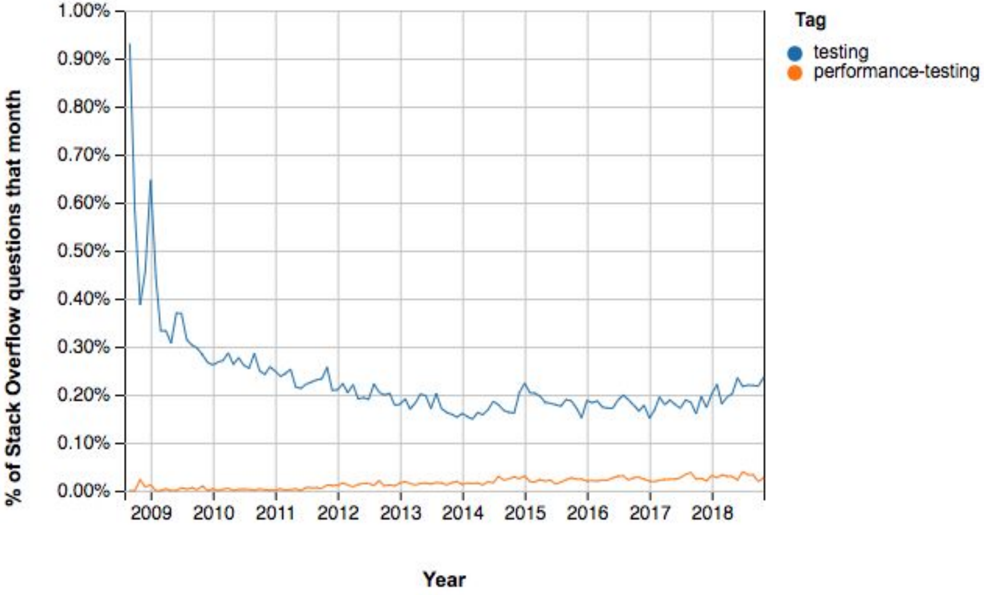
Anomaly testing



- Out-of-memory testing
- IO-error testing
- Crash testing
- Compound failure tests

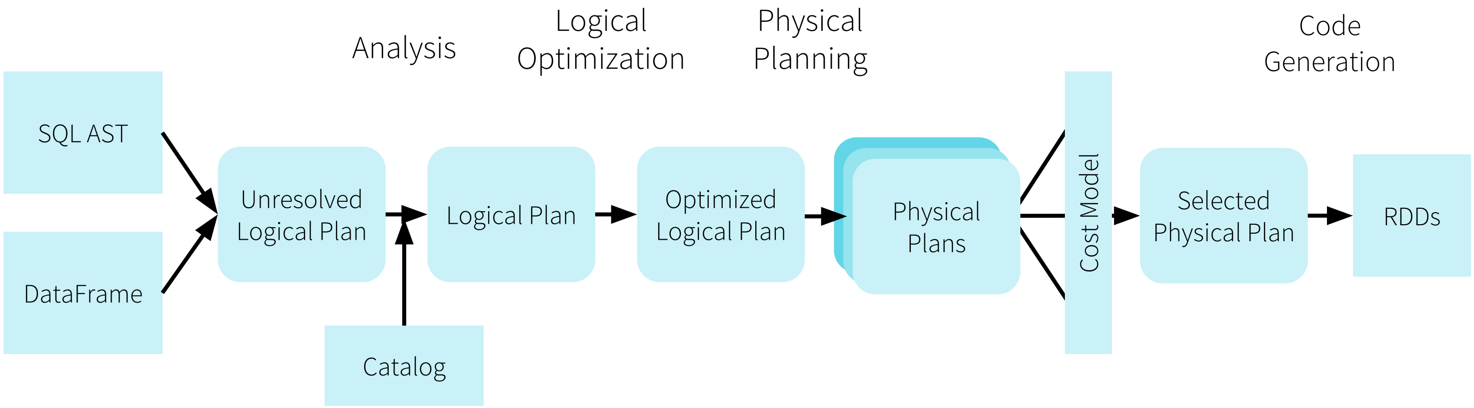
Fuzz testing

- SQL Fuzz
- Malformed database files
- Boundary value tests



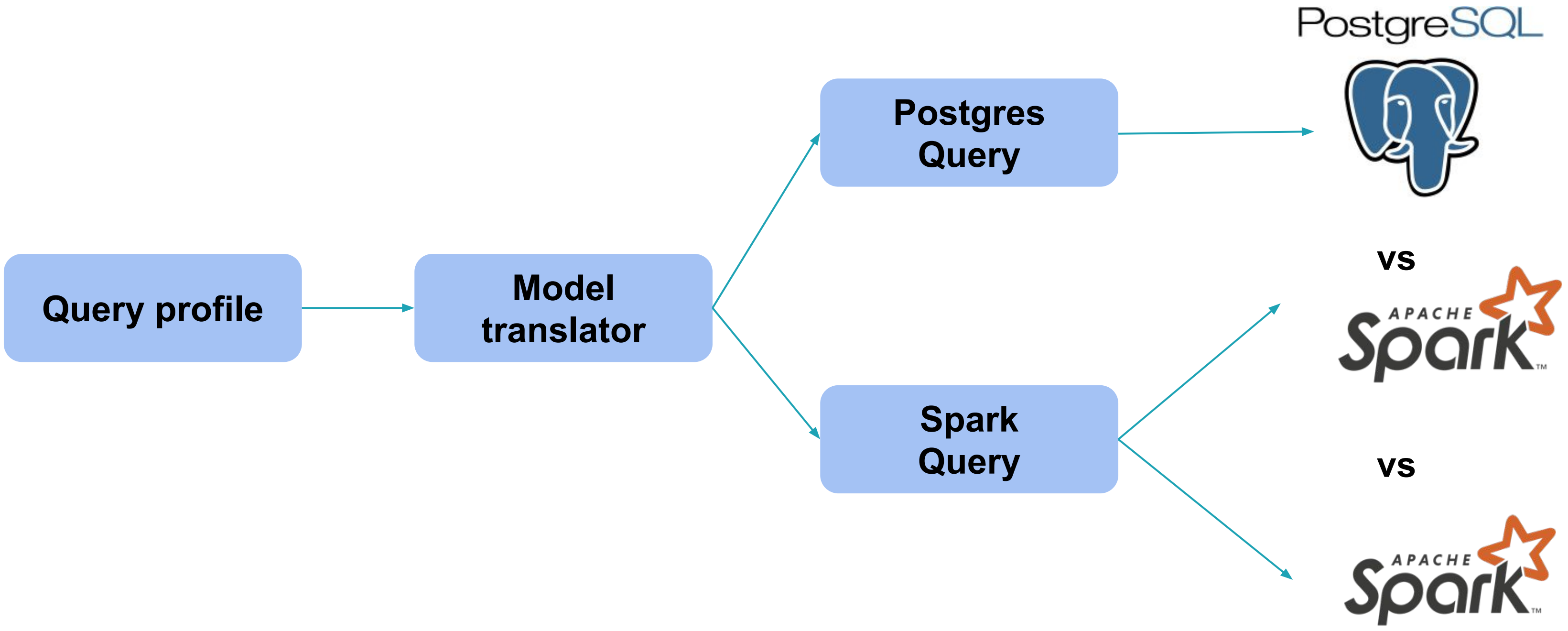
How SQLite Is Tested: https://www.sqlite.org/testing.html

Spark SQL behind the scenes

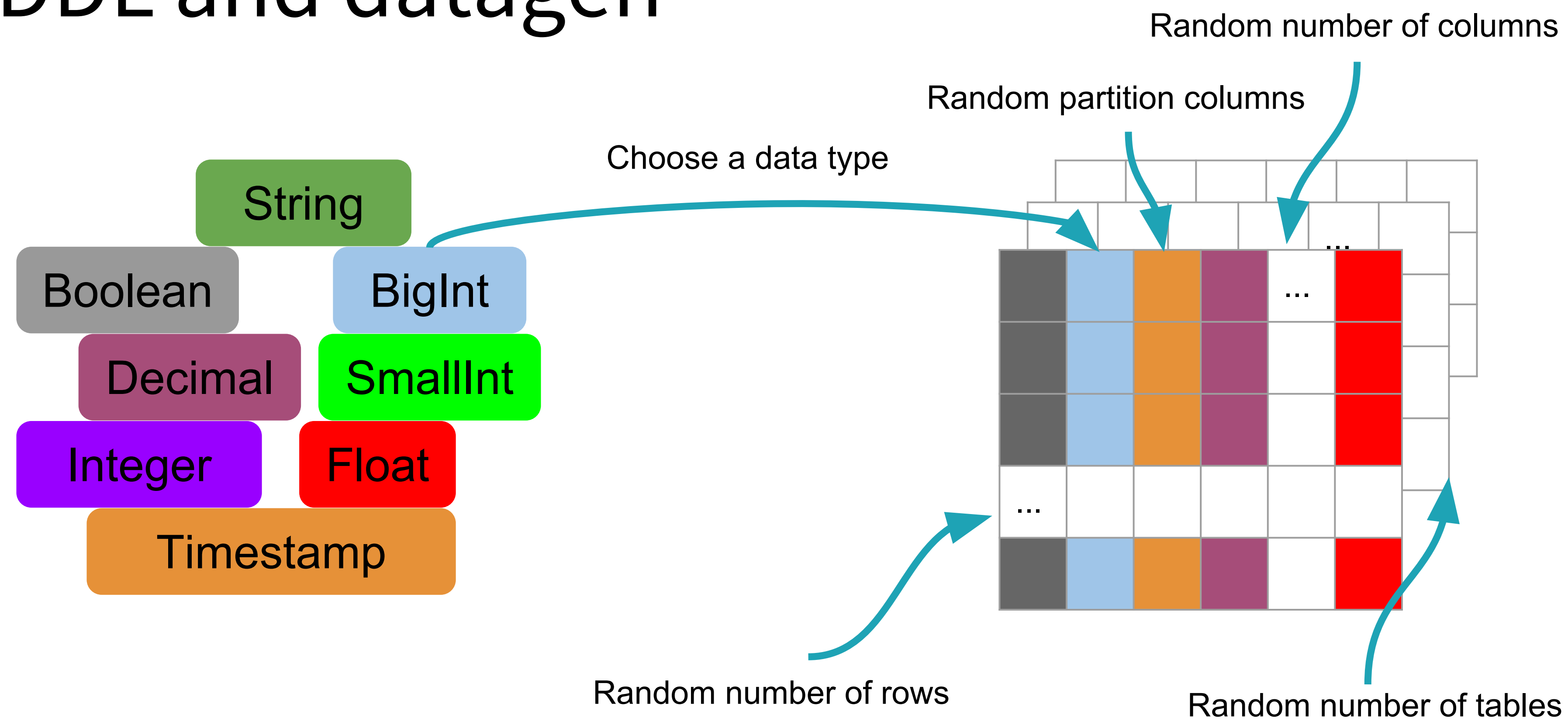


SQL operators can be represented as trees
Phases of transformation prepare the trees for execution
Rules can be applied once or to fix-point

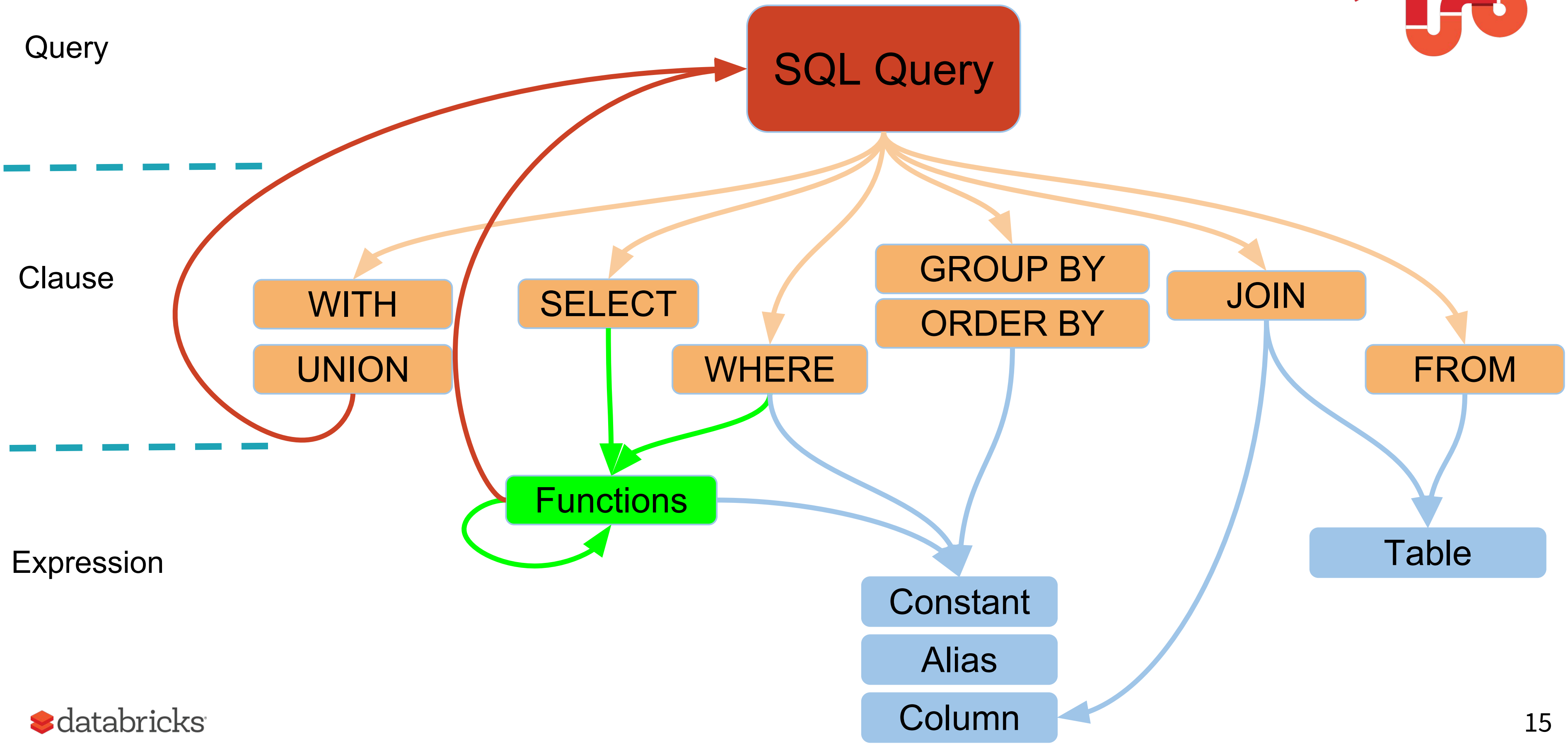
Random query generation



DDL and datagen



Recursive query model



Probabilistic query profile

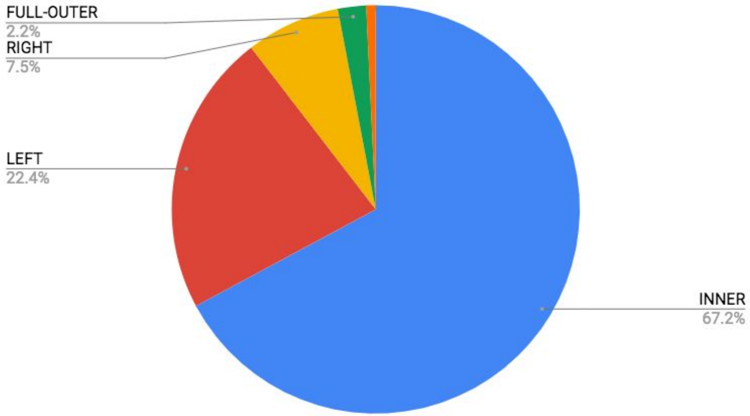
Independent weights

- Optional query clauses

UNION 10% | GROUP BY 10% | WHERE 50% | ORDER BY 10%

Inter-dependent weights

- Join types
- Select functions



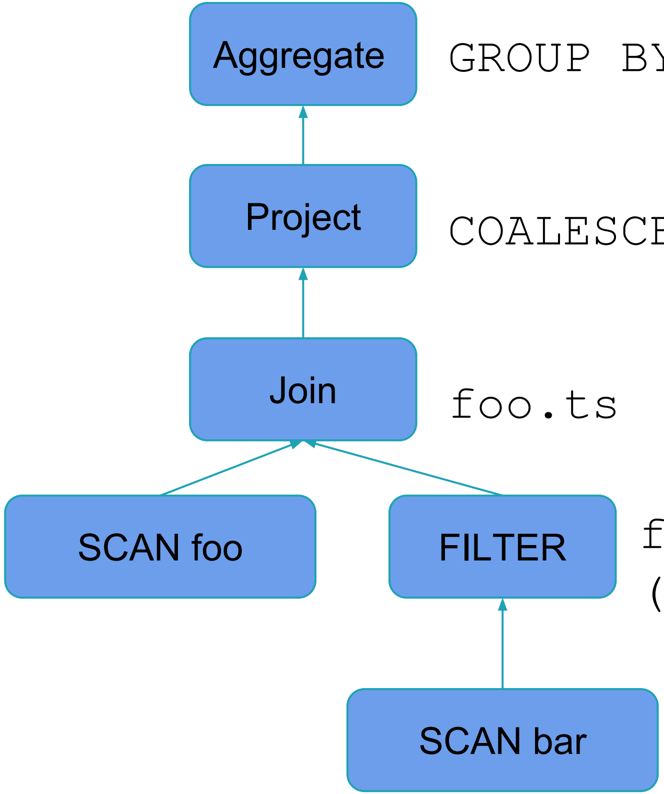
Coalesce flattening (1/4)

```
SELECT COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3) AS int_col,
       IF(NULL, VARIANCE(COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)),
       COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)) AS int_col_1,
       STDDEV(t2.double_col_2) AS float_col,
       COALESCE(MIN((t1.smallint_col_3) - (COALESCE(t2.smallint_col_3, t1.smallint_col_3,
       t2.smallint_col_3))), COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3),
       COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)) AS int_col_2
FROM table_4 t1
INNER JOIN table_4 t2 ON (t2.timestamp_col_7) = (t1.timestamp_col_7)
WHERE (t1.smallint_col_3) IN (CAST('0.04' AS DECIMAL(10,10)), t1.smallint_col_3)
GROUP BY COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)
```

Small dataset with 2 tables of 5x5 size
Within 10 randomly generated queries

Error: Operation is in ERROR_STATE

Coalesce flattening (2/3)



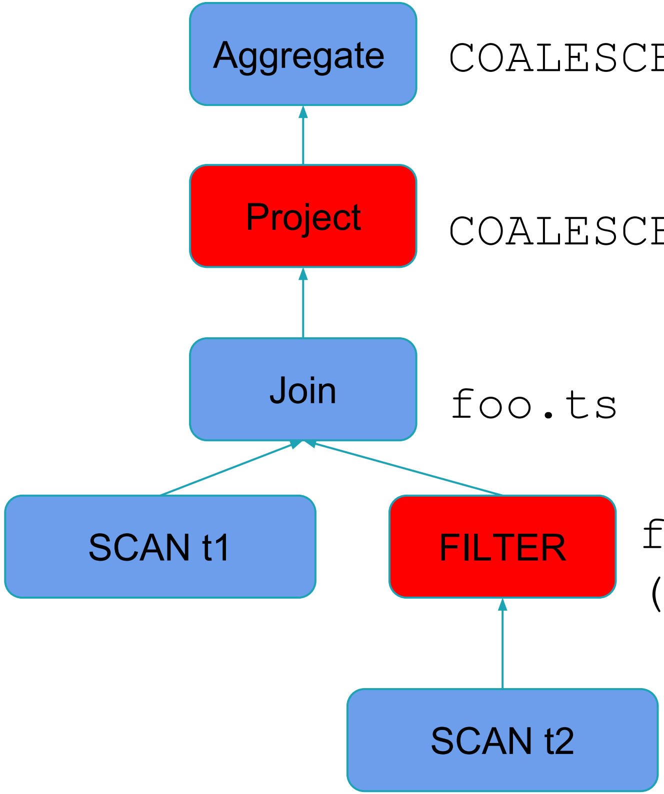

```
GROUP BY COALESCE(foo.id, foo.val)

COALESCE(COALESCE(foo.id, foo.val), 88)

foo.ts  = bar.ts

foo.id IN
(CAST('0.04' AS DECIMAL(10, 10)), foo.id)
```

Coalesce flattening (3/4)



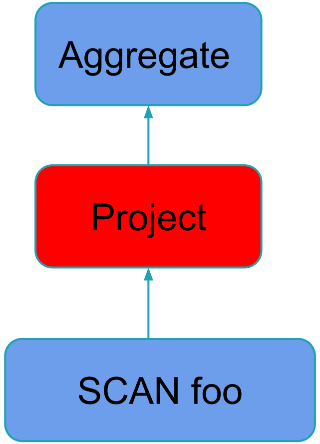

```
COALESCE(foo.id, foo.val)

COALESCE(COALESCE(foo.id, foo.val), 88)

foo.ts  = bar.ts

foo.id IN
(CAST('0.04' AS DECIMAL(10, 10)), foo.id)
```

Coalesce flattening (4/4)



Minimized query:

```
SELECT
   COALESCE(COALESCE(foo.id, foo.val), 88)
FROM foo
GROUP BY
   COALESCE(foo.id, foo.val)
```

Analyzing the error

- The optimizer flattens the nested coalesce calls
- The SELECT clause doesn't contain the GROUP BY expression
- Possibly a problem with any GROUP BY expression that can be optimized

Lead function (1/3)

```
SELECT (t1.decimal0803_col_3) / (t1.decimal0803_col_3) AS decimal_col,
        CAST(696 AS STRING) AS char_col, t1.decimal0803_col_3,
       (COALESCE(CAST('0.02' AS DECIMAL(10,10)),
                 CAST('0.47' AS DECIMAL(10,10)),
                 CAST('-0.53' AS DECIMAL(10,10)))) +
       (LEAD(-65, 4) OVER (ORDER BY (t1.decimal0803_col_3) / (t1.decimal0803_col_3),
                 CAST(696 AS STRING))) AS decimal_col_1,
                 CAST(-349 AS STRING) AS char_col_1

FROM table_16 t1
WHERE (943) > (889)
```

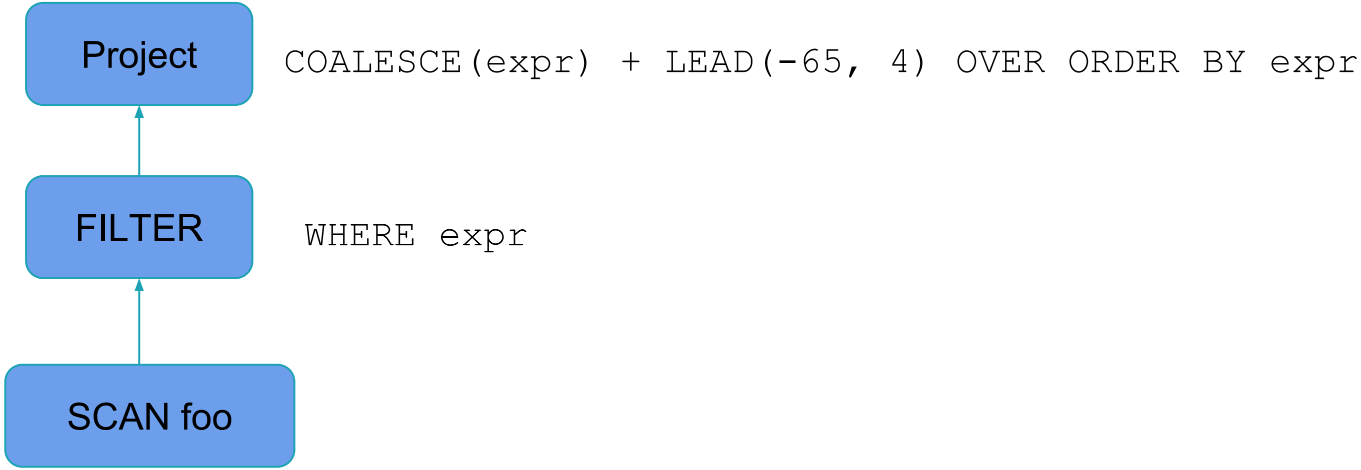
Error: Column 4 in row 10 does not match:

[1.0, 696, -871.81, <<-64.98>>, -349] SPARK row

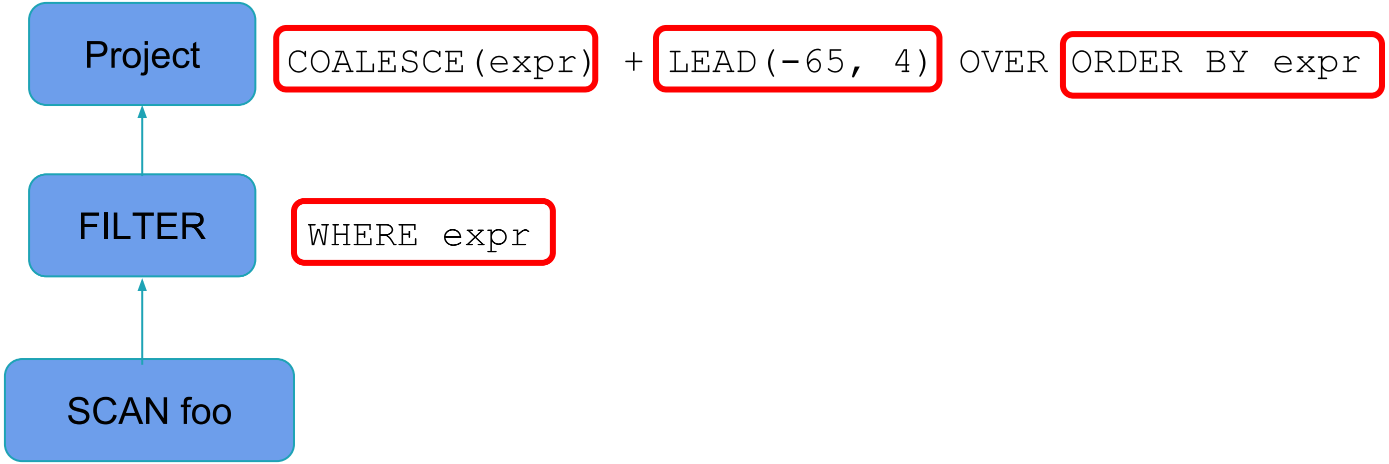
[1.0, 696, -871.81, <<None>>, -349] POSTGRESQL row

databricks

Lead function (2/3)



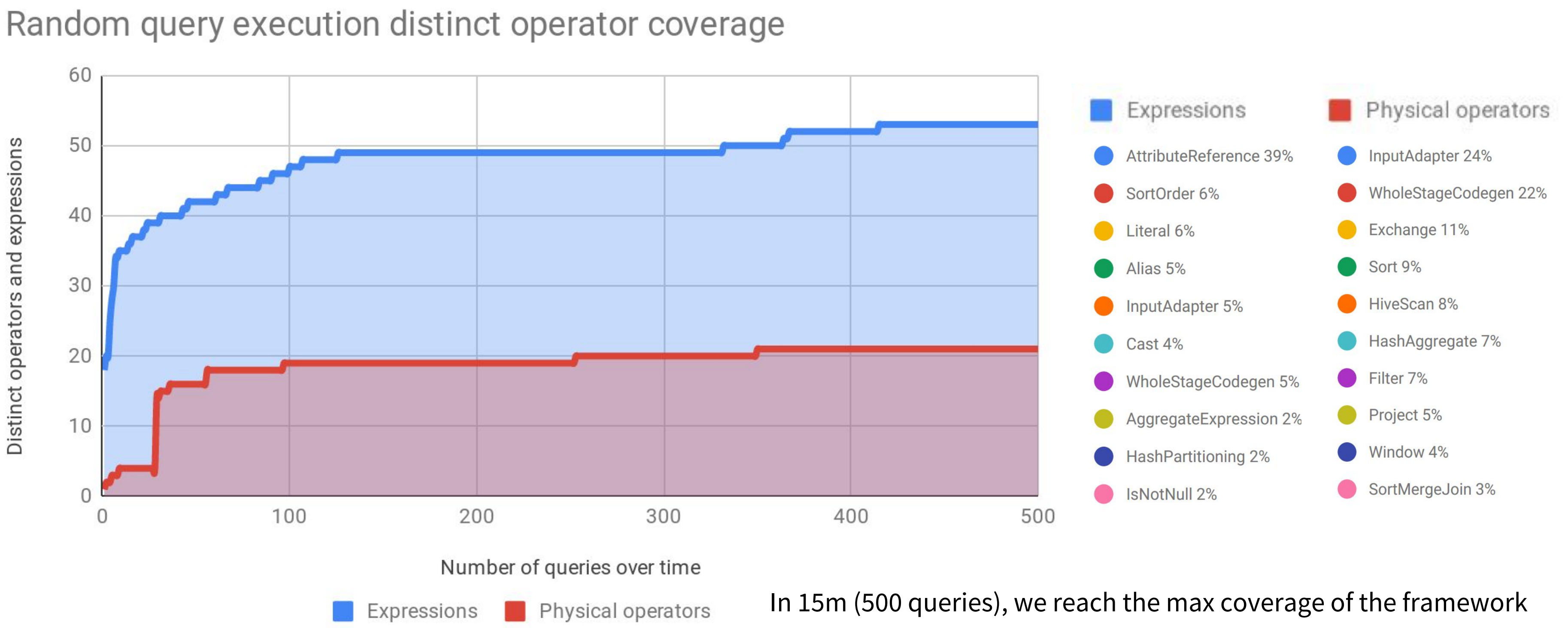
Lead function (3/3)



Analyzing the error

- Using constant input values breaks the behaviour of the LEAD function
- SPARK-16633: https://github.com/apache/spark/pull/14284

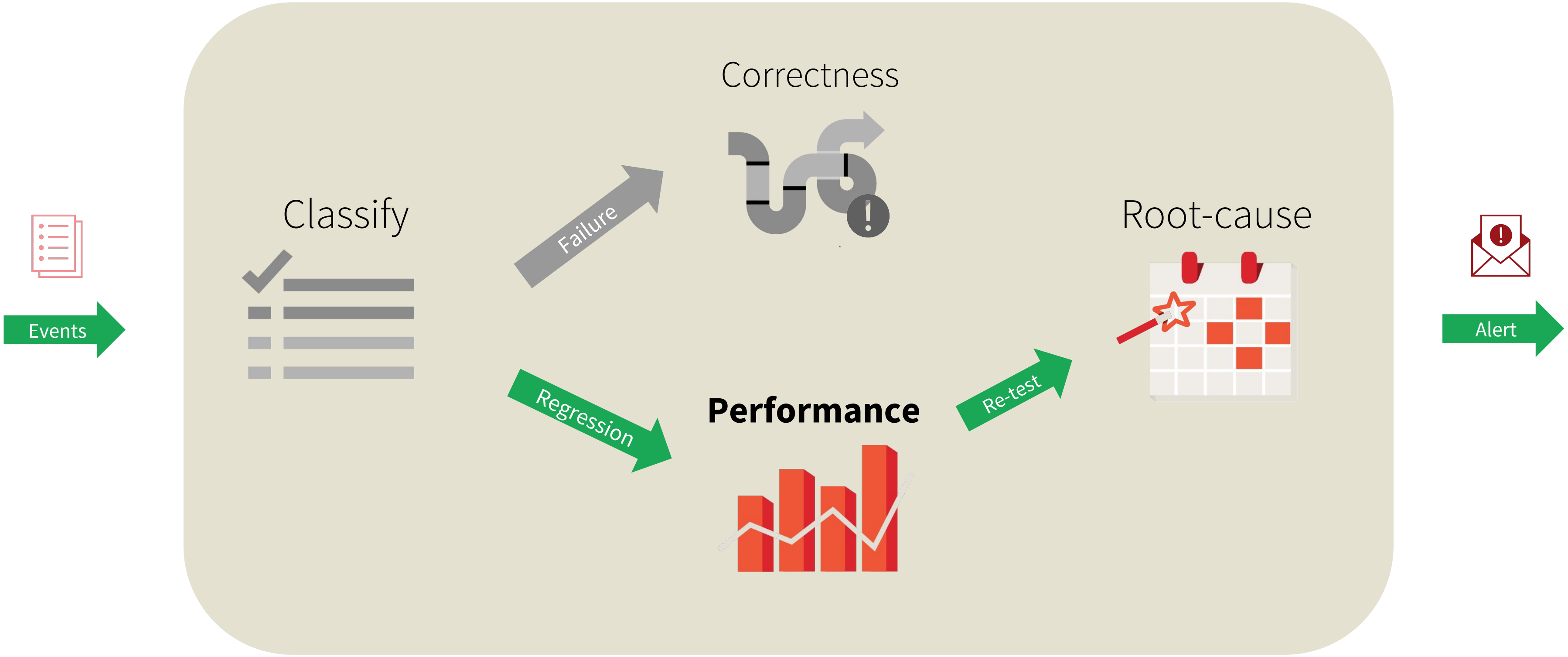
Query operator coverage analysis



In 15m (500 queries), we reach the max coverage of the framework

databricks

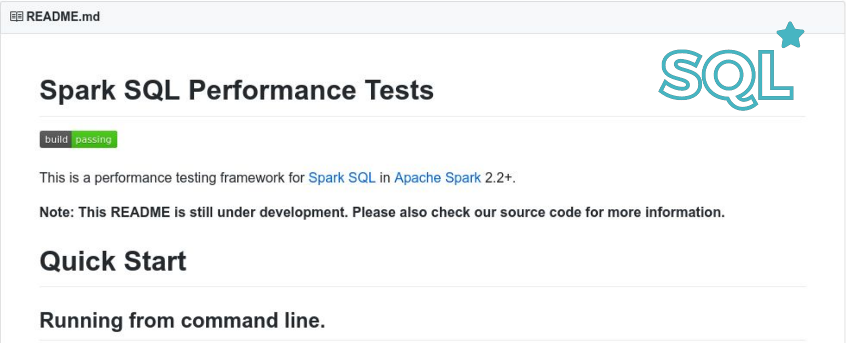
Performance



Benchmarking tools

- We use spark-sql-perf public library for TPC workloads
  - Provides datagen and import scripts
  - local, cluster, S3
  - Dashboards for analyzing results

- The Spark micro benchmarks
- And the async-profiler
  - to produce flamegraphs


README.md

Spark SQL Performance Tests

build passing

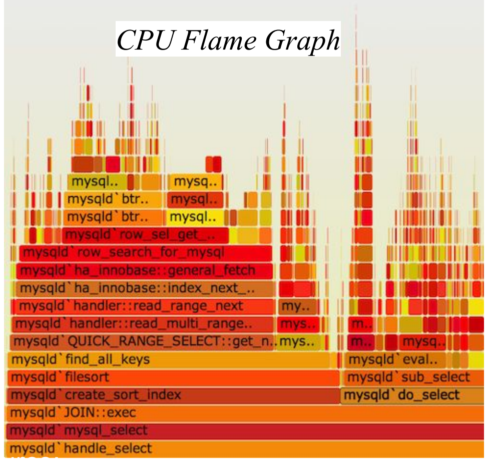
This is a performance testing framework for Spark SQL in Apache Spark 2.2+.

Note: This README is still under development. Please also check our source code for more information.

Quick Start

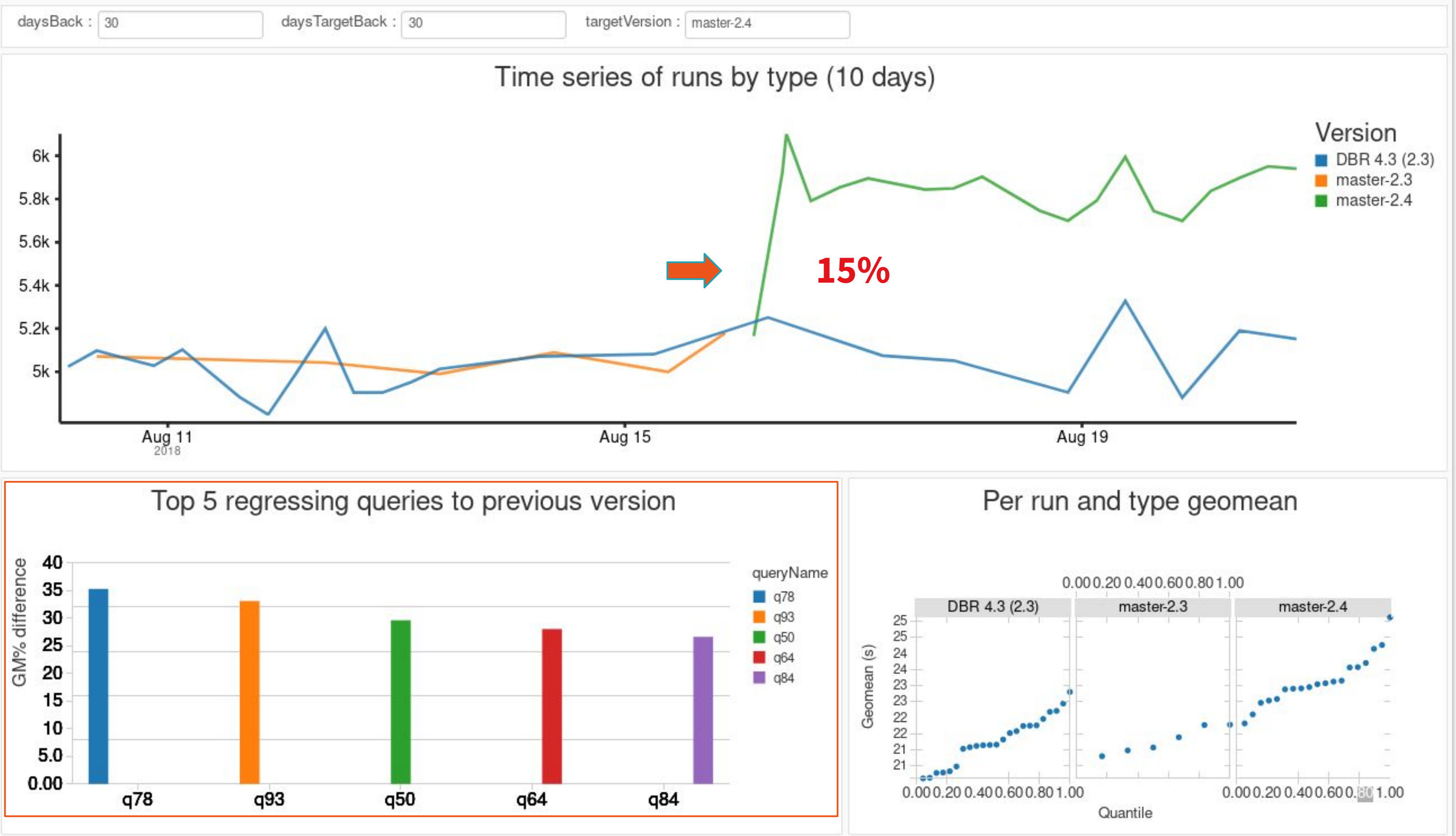
Running from command line.

https://github.com/databricks/spark-sql-perf



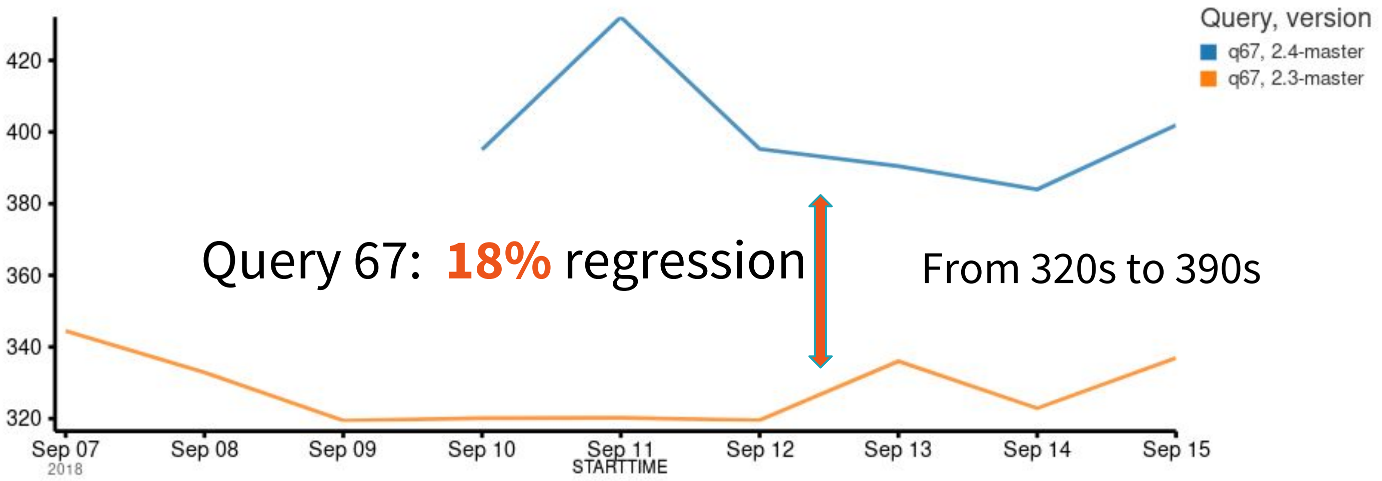
http://www.brendangregg.com/flamegraphs.html

DBR 5.0-beta (v2.4) performance ~~tracking~~ --- journey



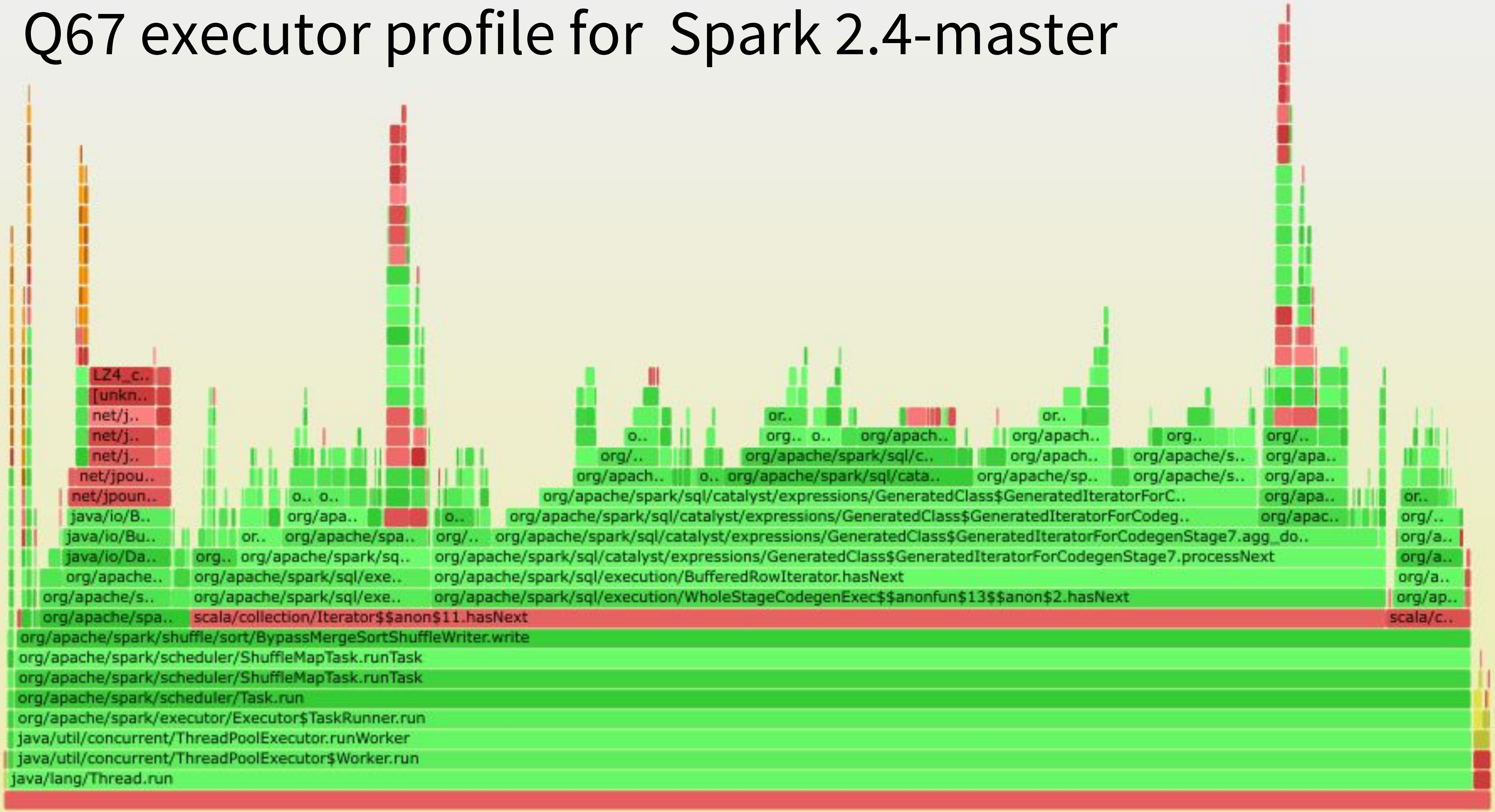
Per query drill-down: q67

First, **scope** and **validate**

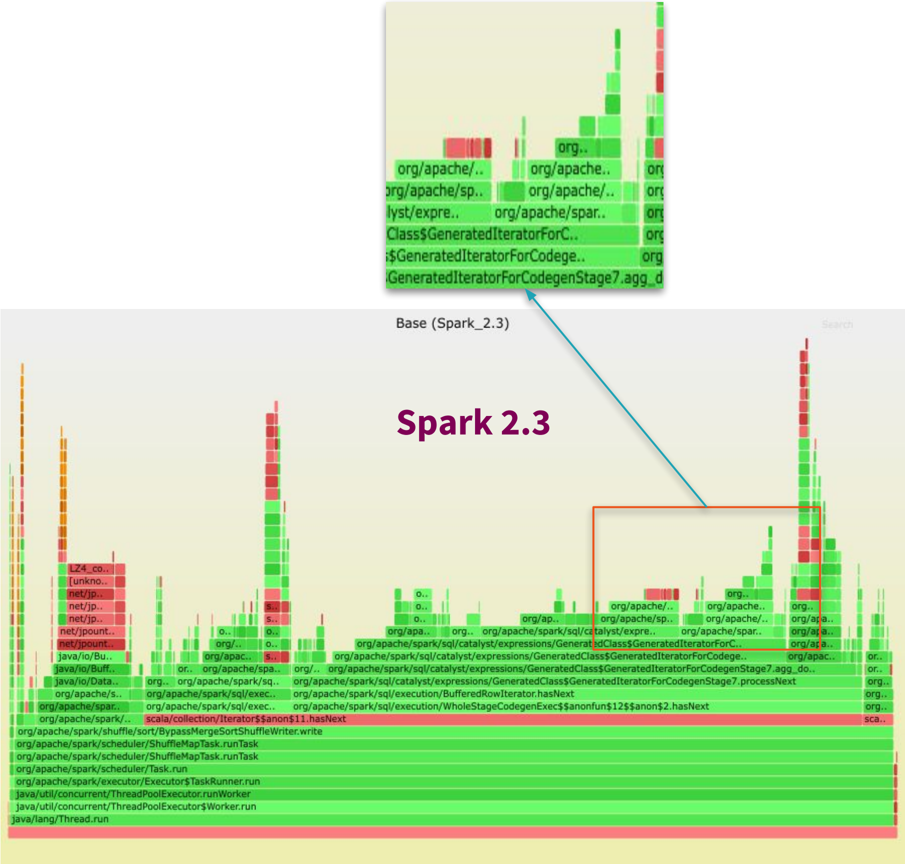


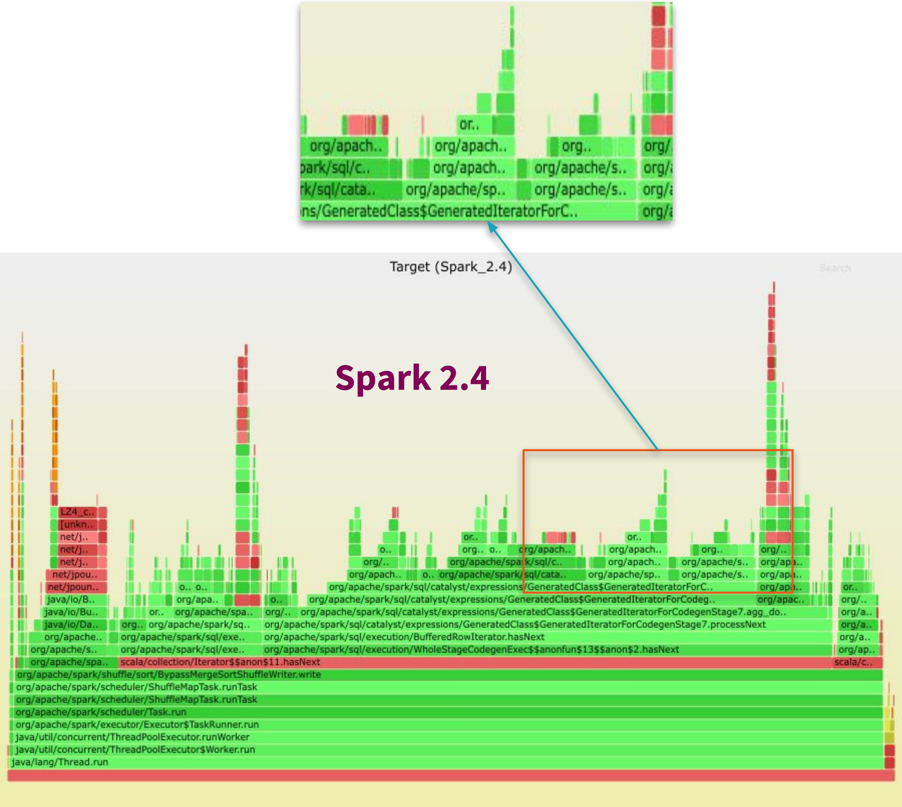
- in 2.4-master (dev) compared
- to 2.3 in DBR 4.3 (prod)

Q67 executor profile for Spark 2.4-master



Side-by-side 2.3 vs 2.4: find the differences



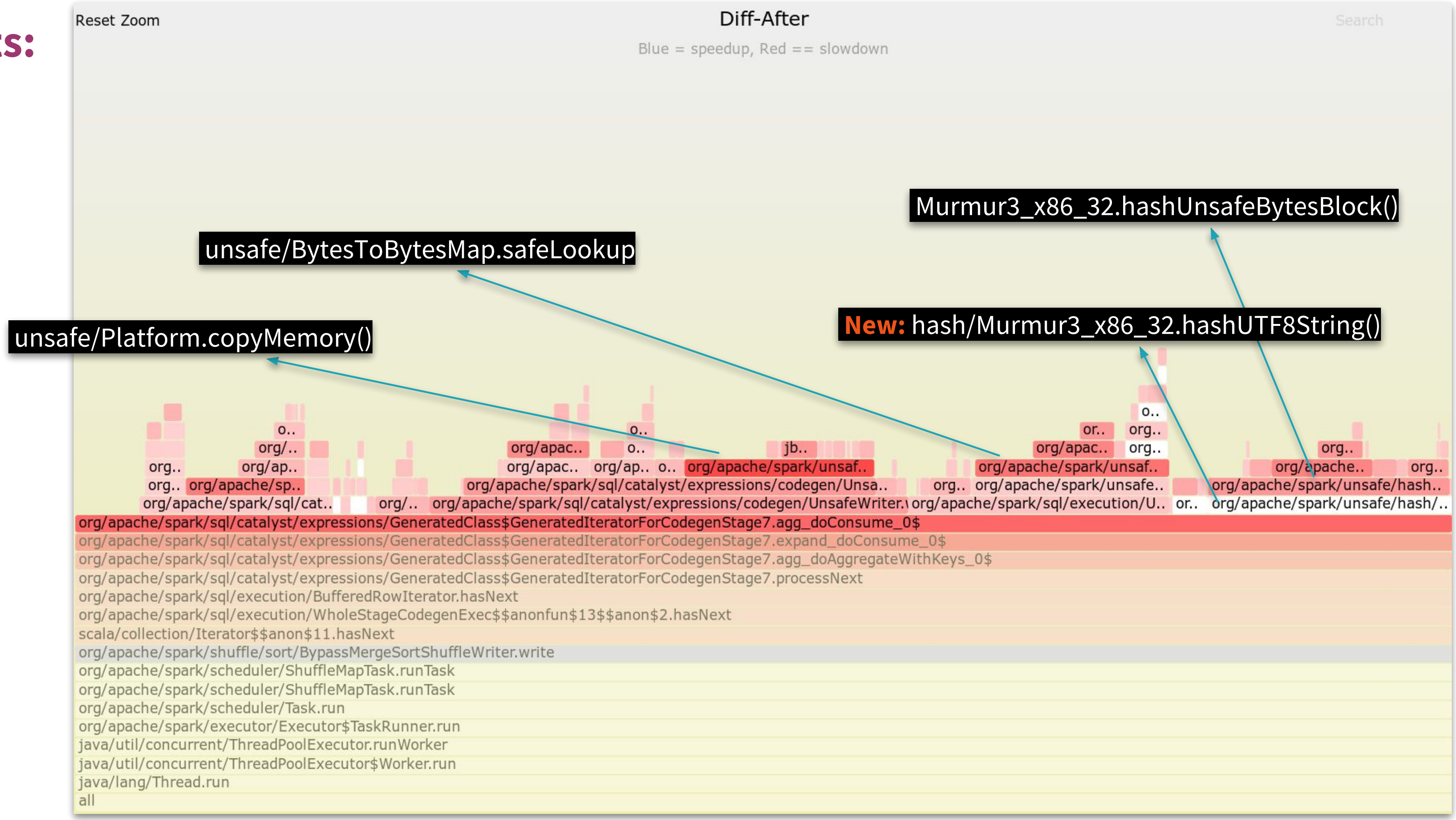


Framegraph diff zoom

Red slower **White** new

Look for hints:

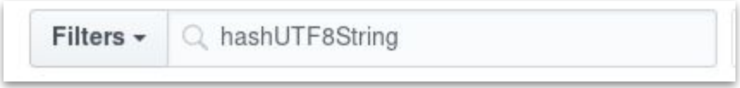
- Mem mgmt
- Hashing
- unsafe



databricks

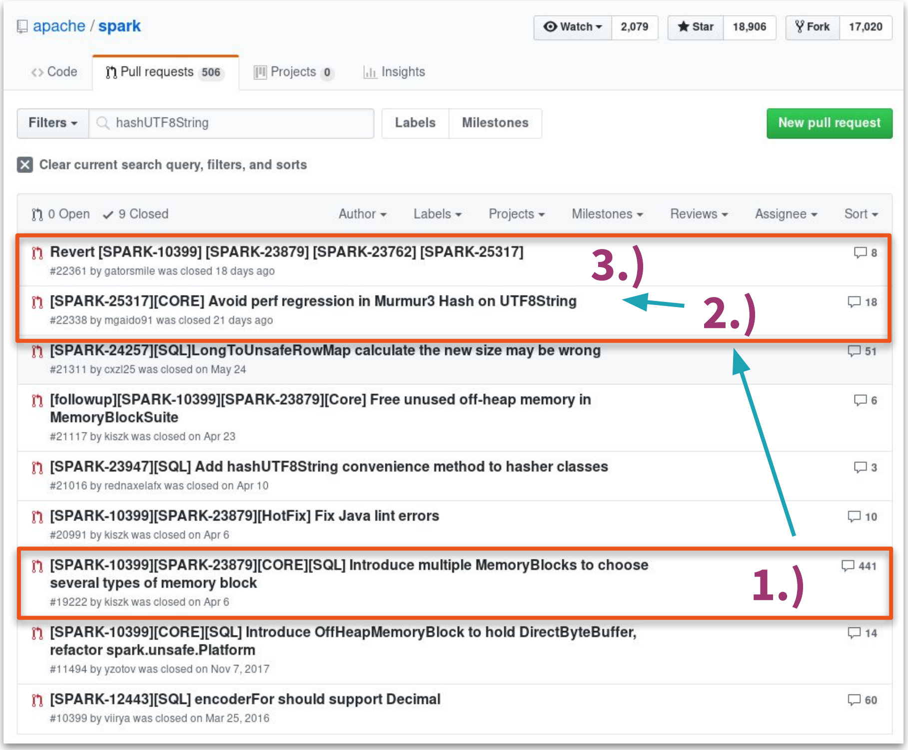
Root-causing

GIT BISECT



Microbenchmark for UTF8String

```scala
test("hashing") {
    import org.apache.spark.unsafe.hash.Murmur3_x86_32
    import org.apache.spark.unsafe.types.UTF8String
    val hasher = new Murmur3_x86_32(0)
    val str = UTF8String.fromString("b" * 10001)
    val numIter = 100000
    val start = System.nanoTime
    for(i <- 0 until numIter) {
      Murmur3_x86_32.hashUTF8String(str, 0)
```

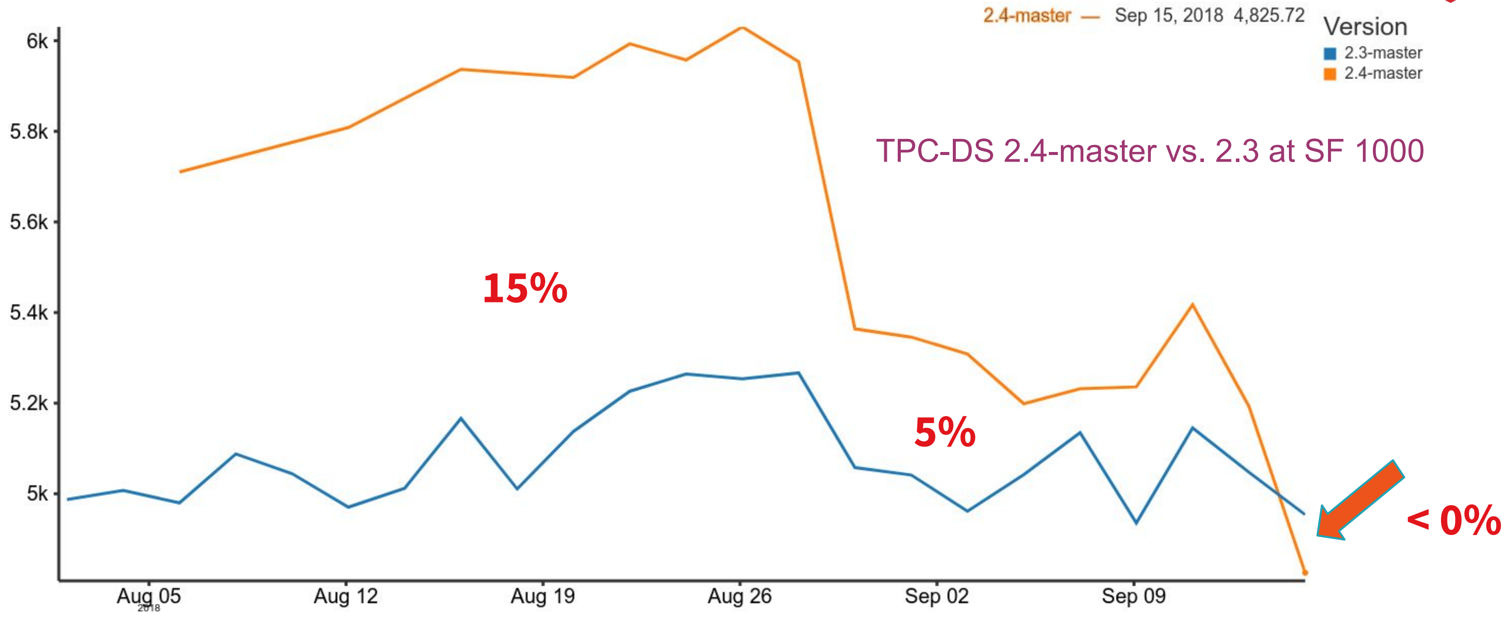


Results:

- Spark 2.3: hashUnsafeBytes() -> **40µs**
- Spark 2.4 *hashUnsafeBytesBlock()* -> **140µs**

- also slower *UTF8String.getBytes()*

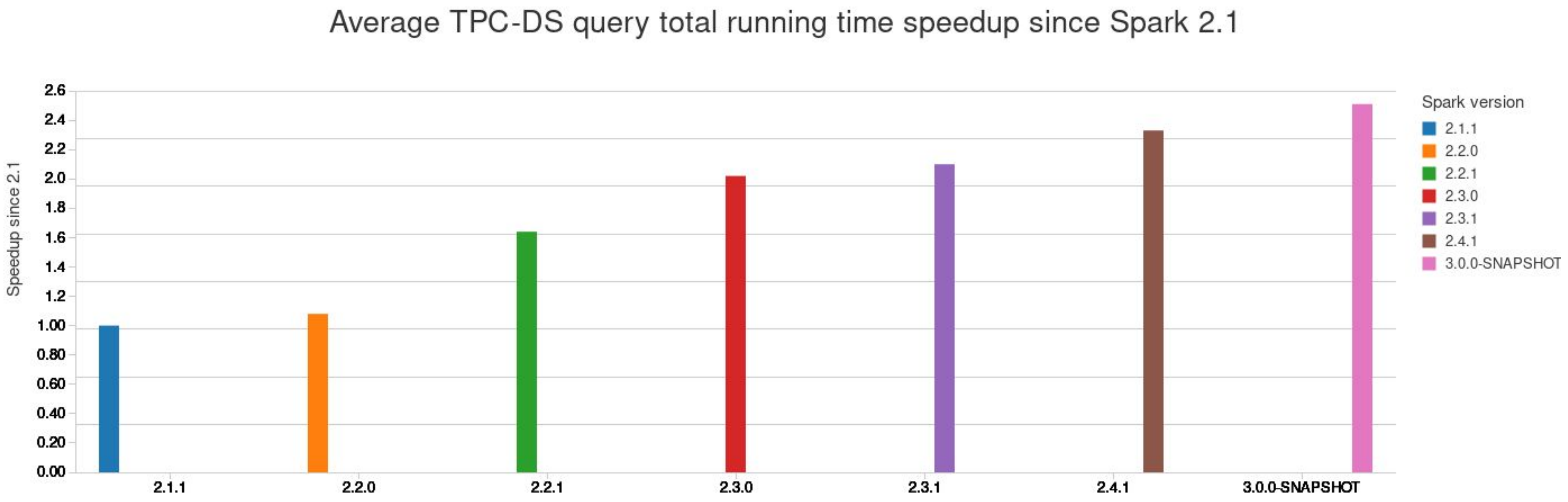
It is a journey to get a release out



DBR and Spark testing and performance are a **continuous effort**

- **Over a month** effort to bring performance to improving

… a journey that pays off quickly



Query times have improved over **2X**

in the Spark 2.x branch measured in the Databricks platform

databricks

Note: both 2.4.1 and 3.0.0 are not released yet

Conclusion

Spark in production is *not just the framework*

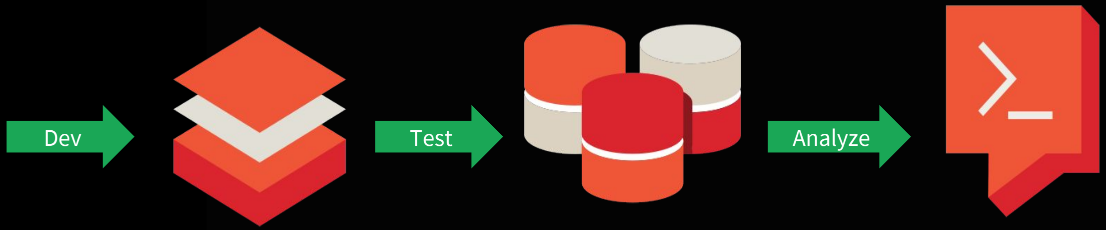
Unit and integration testing are not enough

We need Spark specific tools to automate the process

to ensure both correctness and performance

Fast and Reliable Apache Spark SQL Releases

Thanks!



Feedback: {Nico.Poggi, Bogdan.Ghit}@databricks.com

March 2019