GraphFrames

DataFrame-based graphs for Apache® Spark™

Joseph K. Bradley

4/14/2016

About the speaker: Joseph Bradley

Joseph Bradley is a Software Engineer and Apache Spark PMC member working on MLlib at Databricks. Previously, he was a postdoc at UC Berkeley after receiving his Ph.D. in Machine Learning from Carnegie Mellon U. in 2013. His research included probabilistic graphical models, parallel sparse regression, and aggregation mechanisms for peer grading in MOOCs.

About the moderator: Denny Lee

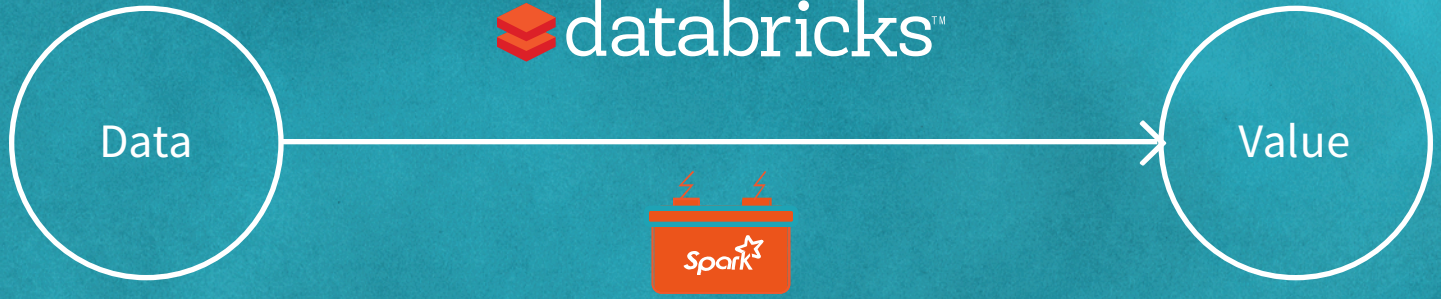
Denny Lee is a Technology Evangelist with Databricks; he is a hands-on data sciences engineer with more than 15 years of experience developing internet-scale infrastructure, data platforms, and distributed systems for both on-premises and cloud. Prior to joining Databricks, Denny worked as a Senior Director of Data Sciences Engineering at Concur and was part of the incubation team that built Hadoop on Windows and Azure (currently known as HDInsight).

We are Databricks, the company behind Apache Spark

Founded by the creators of Apache Spark in 2013

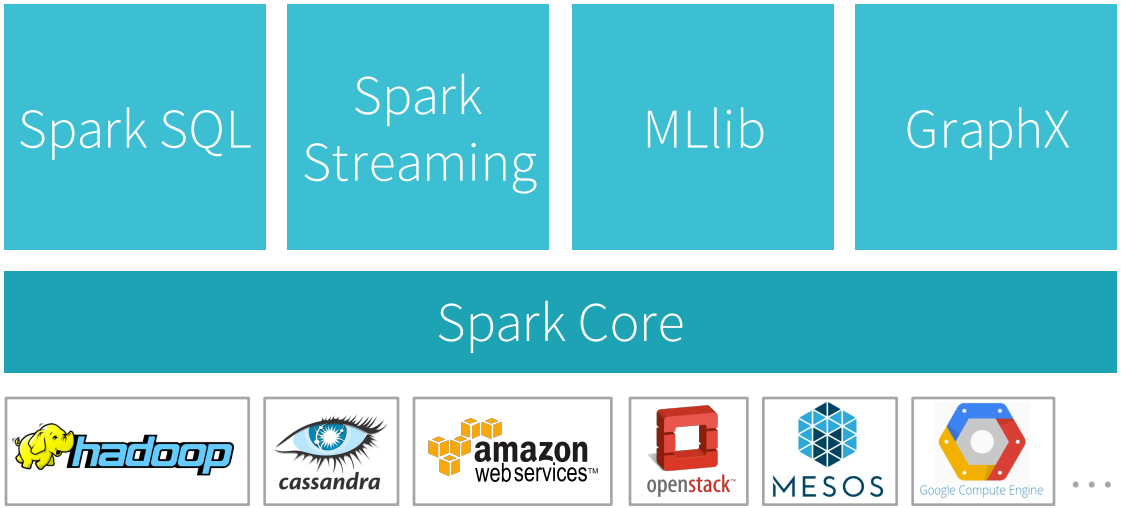
75%

Share of Spark code contributed by Databricks in 2014



Created **Databricks** on top of Spark to **make big data simple.**

Apache Spark Engine



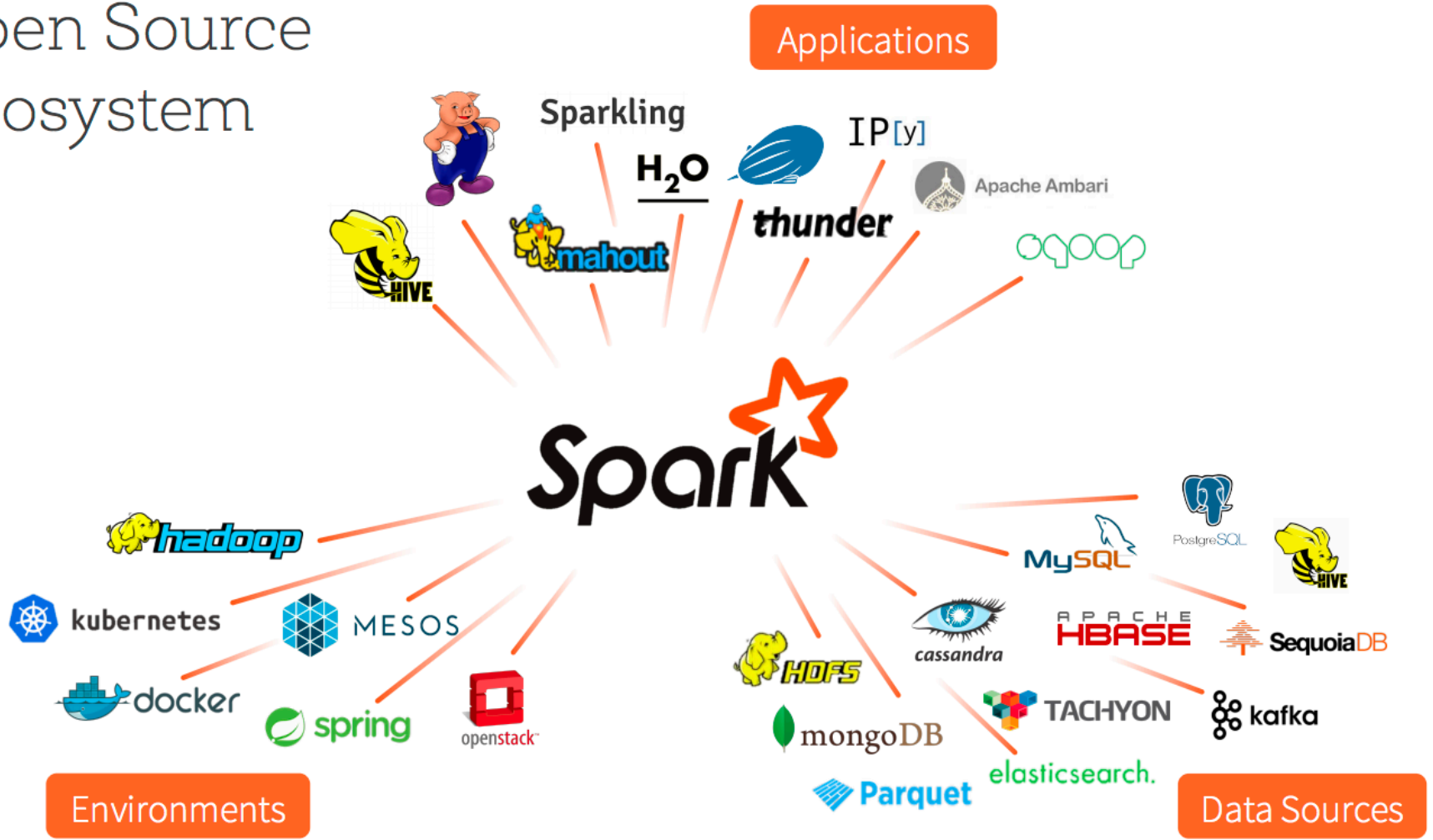
Scale out, fault tolerant

Python, Java, Scala, and R APIs

Standard libraries

Unified engine across diverse workloads & environments

Open Source Ecosystem



NOTABLE USERS THAT PRESENTED AT SPARK SUMMIT 2015 SAN FRANCISCO

Source: Slide 5 of Spark Community Update

Outline

GraphFrames overview

GraphFrames vs. GraphX and other libraries

Details for power users

Roadmap and resources

Outline

GraphFrames overview

GraphFrames vs. GraphX and other libraries

Details for power users

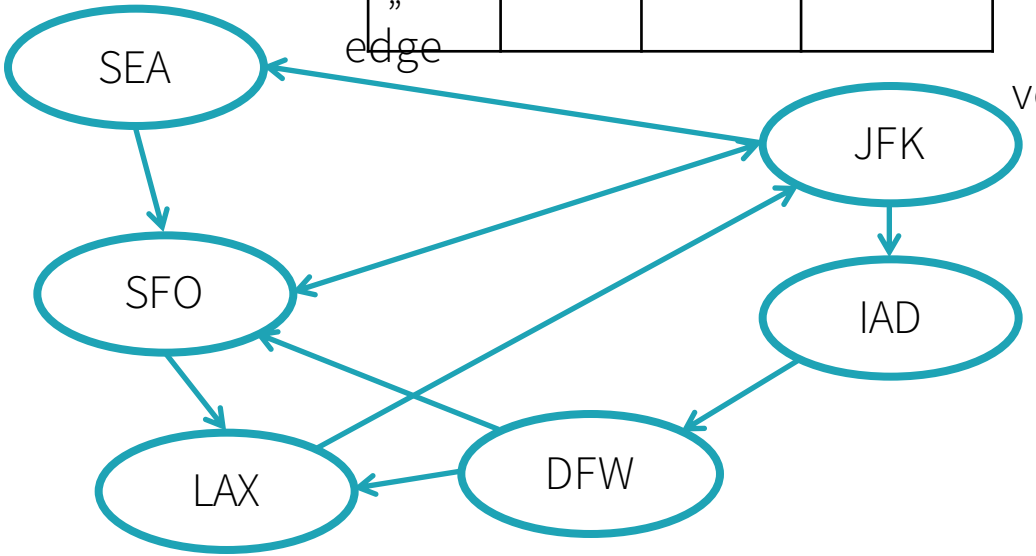
Roadmap and resources

Graphs

Example: airports & flights between them

edge

src	dst	delay	tripID
“JFK”	“SEA”	45	1058923



vertex

id	City	State
“JFK”	“New York”	NY

Apache Spark's GraphX library

Overview

- General-purpose graph processing library
- Optimized for fast distributed computing
- Library of algorithms: PageRank, Connected Components, etc.

Challenges

- No Java, Python APIs
- Lower-level RDD-based API (vs. DataFrames)
- Cannot use recent Spark optimizations: Catalyst query optimizer, Tungsten memory management

Enter GraphFrames

Goal: DataFrame-based graphs on Apache Spark

- Simplify interactive queries
- Support motif-finding for structural pattern search
- Benefit from DataFrame optimizations

Collaboration between Databricks, UC Berkeley & MIT

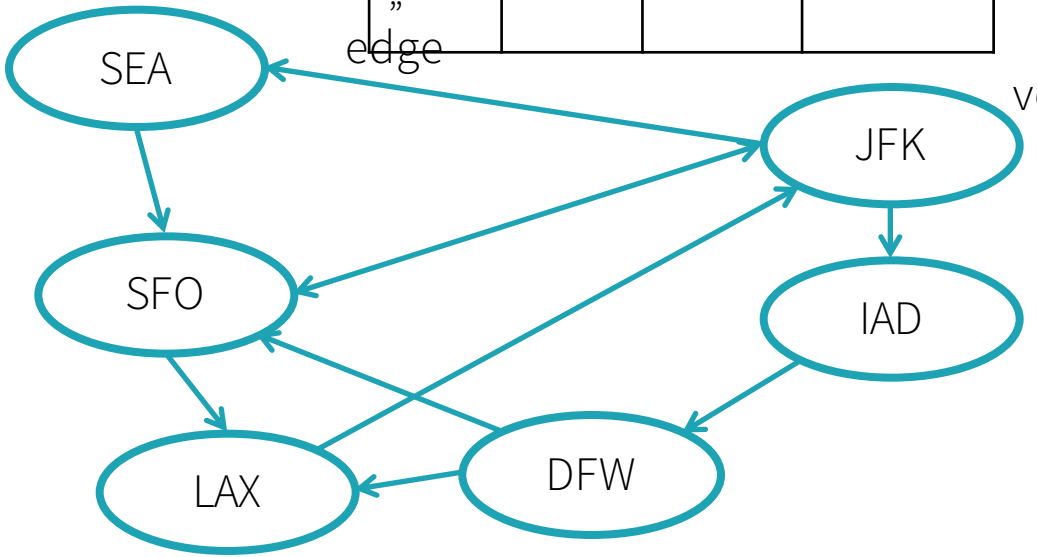
+ Now with community contributors!

Graphs

Example: airports & flights between them

edge

src	dst	delay	tripID
“JFK”	“SEA”	45	1058923



vertex

id	City	State
“JFK”	“New York”	NY

GraphFrames

"vertices" DataFrame

- 1 vertex per Row
- id: column with unique ID

id	City	State
"JFK"	"New York"	NY
"SEA"	"Seattle"	WA

"edges" DataFrame

- 1 edge per Row
- src, dst: columns using IDs from vertices.id

src	dst	delay	tripID
"JFK"	"SEA"	45	1058923
"DFW"	"SFO"	-7	4100224

Extra columns store vertex or edge data (a.k.a. attributes or properties).

Demo:
Building a GraphFrame

Load data as DataFrames

Extract the Airports and Departure Delays information from S3.

```
> # Load & prepare vertices DataFrame  # Set File Paths tripdelaysFilePath = "/dat ...
```

```
> # Prepare edges DataFrame  # Build `departureDelays_geo` DataFrame #  Obtain key ...
```

```
> display(airports)
```

id	City	State	Country
LRD	Laredo	TX	USA
INL	International Falls	MN	USA
SAF	Santa Fe	NM	USA
MSO	Missoula	MT	USA
GRR	Grand Rapids	MI	USA
SAT	San Antonio	TX	USA
MTJ	Montrose	CO	USA
SUN	Sun Valley	ID	USA
GSP	Greenville	SC	USA

Command took 0.66s

Build the Graph

```
> from graphframes import *

  # Note that we already cached our Vertices and Edges
  tripGraph = GraphFrame(airports, departureDelays_geo)
```

Command took 0.07s

```
> tripGraph.vertices
```

Out[6]: DataFrame[id: string, City: string, State: string, Country: string]

Command took 0.02s

```
> tripGraph.edges
```

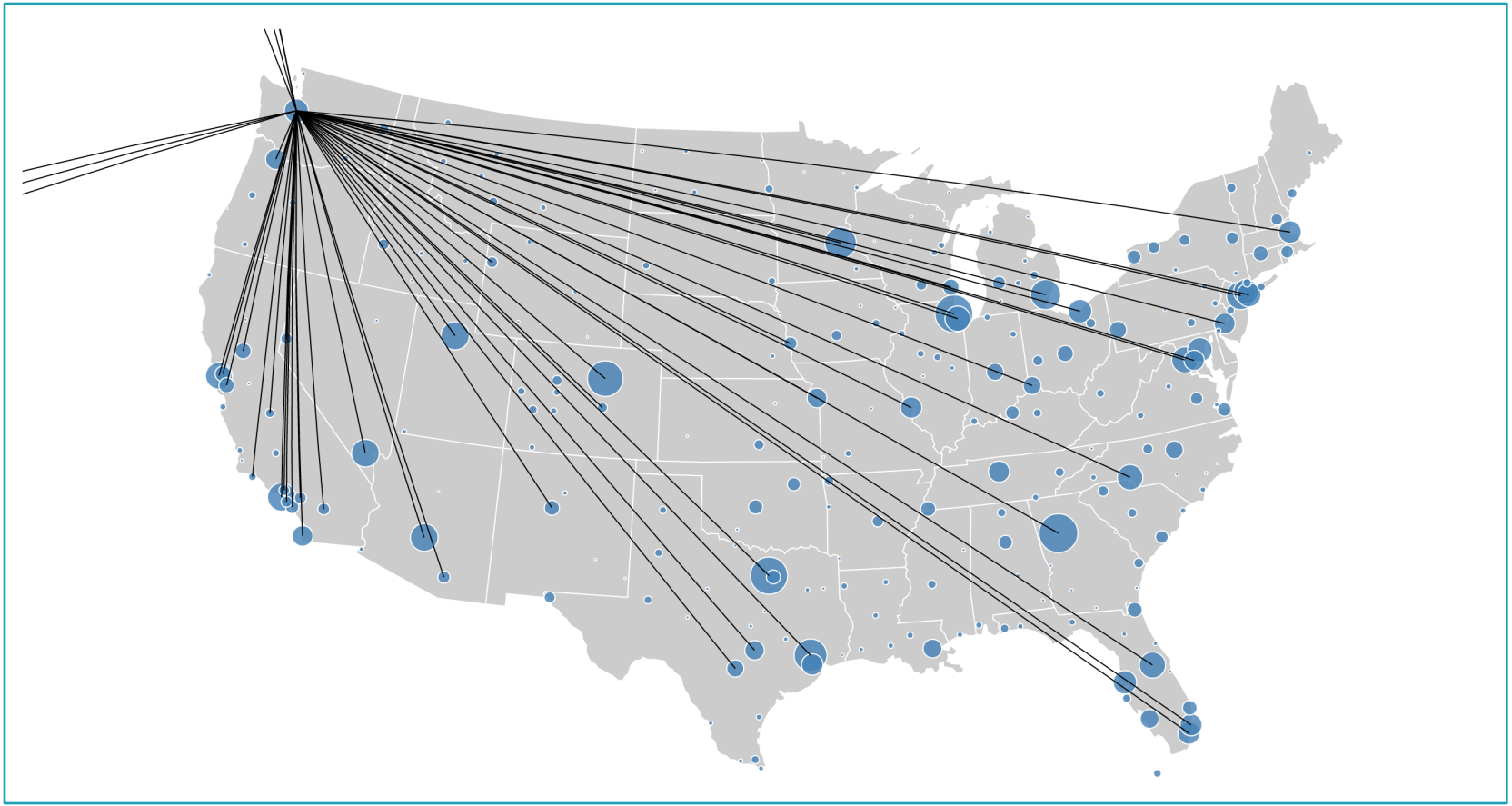
Out[7]: DataFrame[tripid: int, localdate: timestamp, delay: int, distance: int, src: string, dst: string, city_src: string, city_dst: string, state_src: string, state_dst: string]

Command took 0.07s

```
> # Airport and trip counts
  print "Airports: %d" % tripGraph.vertices.count()
  print "Trips: %d" %  tripGraph.edges.count()
```

```
Airports: 279
Trips: 1361141
```

Command took 0.78s



Queries

Simple queries

Motif finding

Graph algorithms

Simple queries

SQL queries on vertices & edges

E.g., what trips are most likely to have significant delays?

Graph queries

- Vertex degrees
  - # edges per vertex (incoming, outgoing, total)
- Triplets
  - Join vertices and edges to get (src, edge, dst)

What trips are most likely to have significant delays?

```
> display(tripGraph.edges
          .groupBy("src", "dst")
          .avg("delay")
          .sort(desc("avg(delay)")))
```

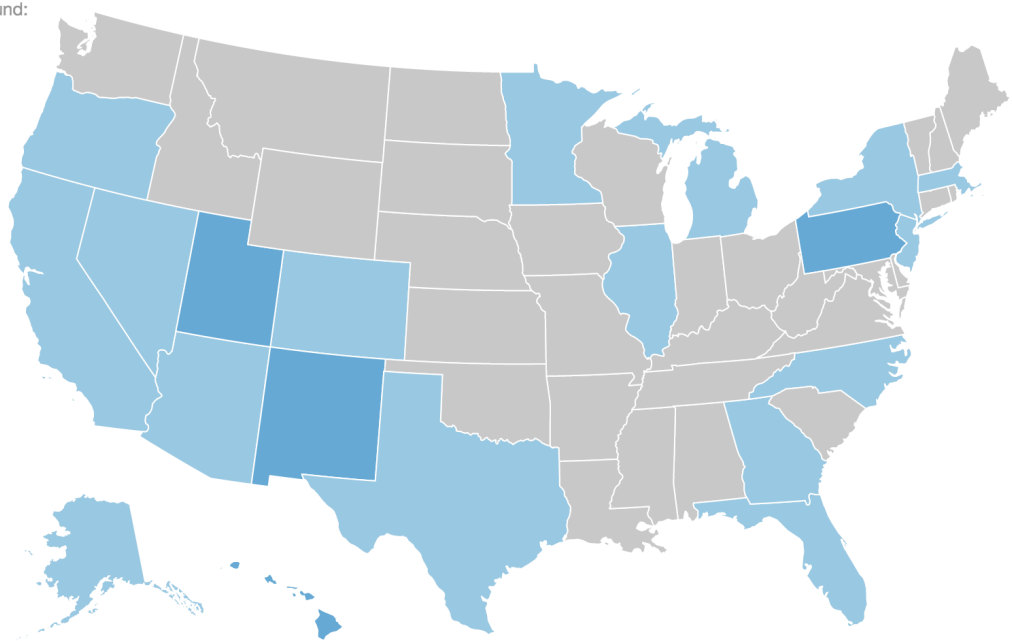
src	dst	avg(delay)
JFK	JAC	322
JAC	JFK	307
SYR	BTV	257
CRW	DTW	131
IND	EVV	129
LGA	DSM	125
MSP	AUS	114.66666666666667
DEN	GSO	98.5
CVG	LEX	97

Showing the first 1000 rows.

Command took 0.88s

```
> # States with the longest cumulative delays (with individual delays > 100 minutes) (origin: Seattle)
  srcSeattleByState = tripGraph.edges.filter("src = 'SEA' and delay > 100")
  display(srcSeattleByState)
```

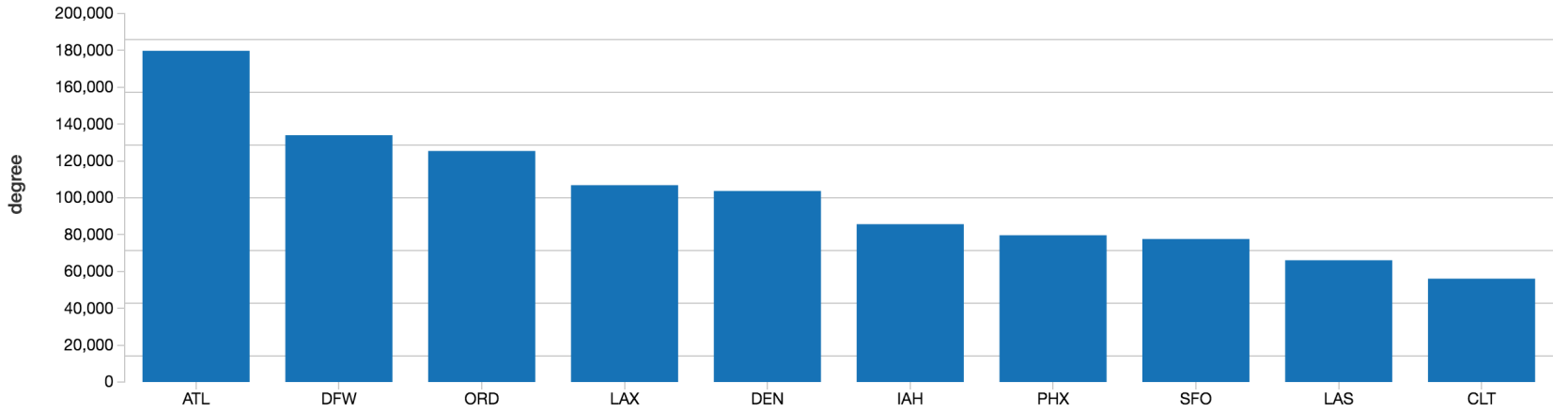
Following states were not found:



Plot Options...

Command took 0.41s

```
> display(tripGraph.degrees
          .sort(desc("degree"))
          .limit(10))
```

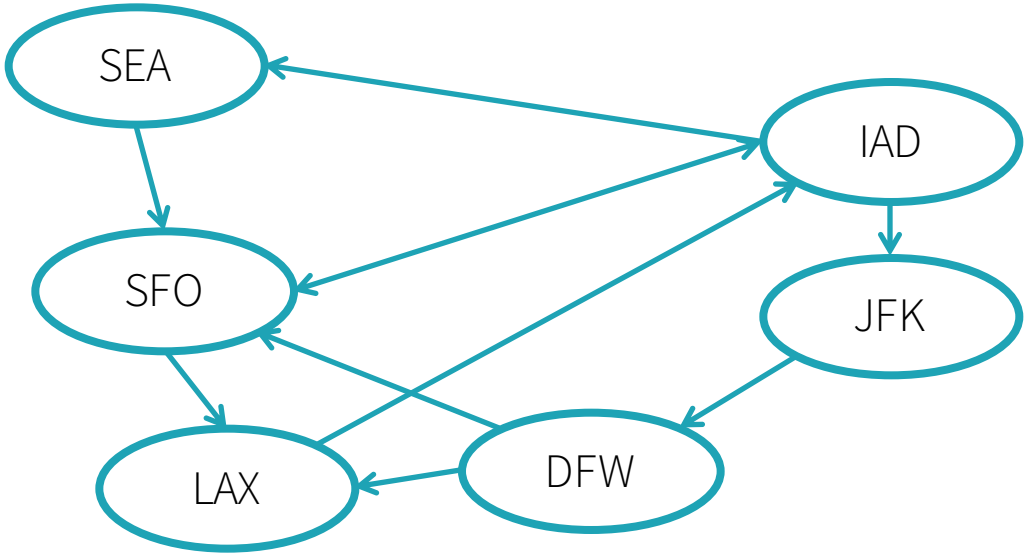


Command took 1.43s

Motif finding

Search for structural patterns within a graph.

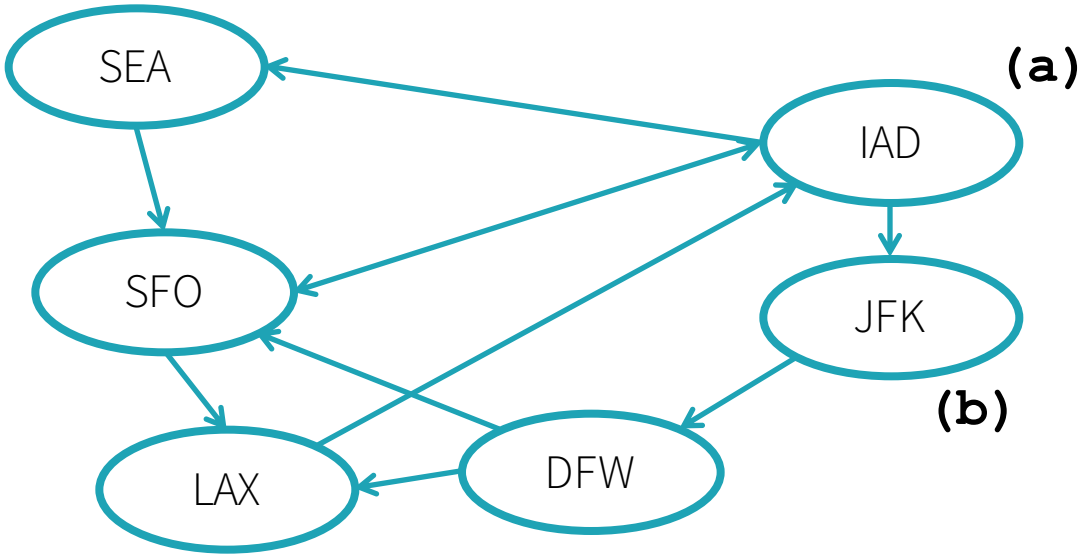
```
val paths: DataFrame =
  g.find("(a)-[e1]->(b);
          (b)-[e2]->(c);
         !(c)-[]->(a)")
```



Motif finding

Search for structural patterns within a graph.

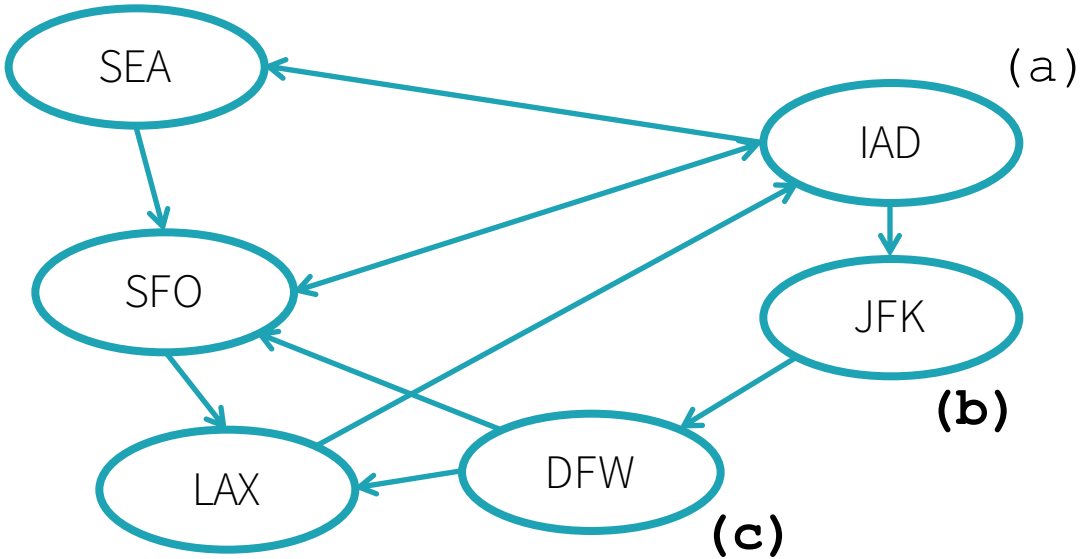
```
val paths: DataFrame =
  g.find("(a)-[e1]->(b);
          (b)-[e2]->(c);
         !(c)-[]->(a)")
```



Motif finding

Search for structural patterns within a graph.

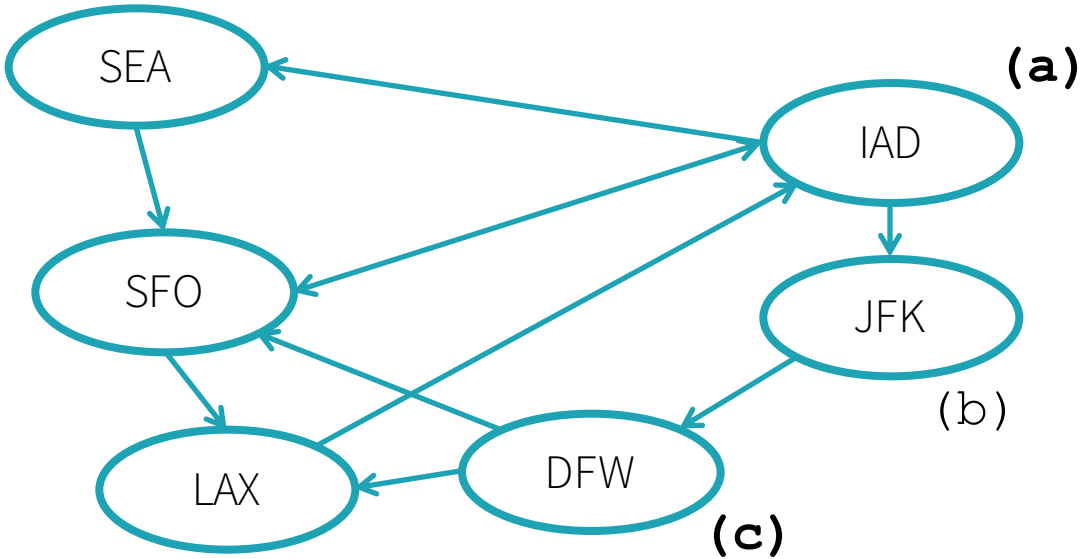
```
val paths: DataFrame =
  g.find("(a)-[e1]->(b);
          (b)-[e2]->(c);
         !(c)-[]->(a)")
```



Motif finding

Search for structural patterns within a graph.

```
val paths: DataFrame =
  g.find("(a)-[e1]->(b);
          (b)-[e2]->(c);
         !(c)-[]->(a)")
```



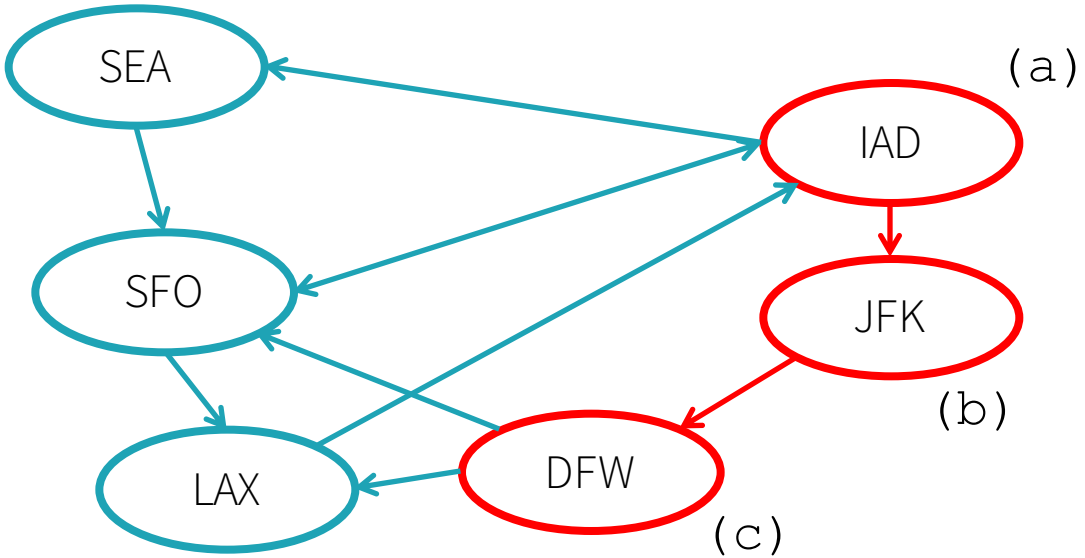
Motif finding

Search for structural patterns within a graph.

```
val paths: DataFrame =
  g.find("(a)-[e1]->(b);
          (b)-[e2]->(c);
         !(c)-[]->(a)")
```

Then filter using vertex & edge data.

```
paths.filter("e1.delay > 20")
```



City / Flight Relationships through Motif Finding

What delays might we blame on SFO?

```
> motifs = tripGraph.find("(a)-[e1]->(b); (b)-[e2]->(c)")\
    .filter("(b.id = 'SFO') and (e1.delay > 500 or e2.delay > 500) and e1.tripid < e2.tripid")
  display(motifs)
```

e1	a	b	e2
{"tripid":1011126,"localdate":"2014-01-01T11:26:00.000+0000","delay":-4,"distance":1421,"src":"IAH","dst":"SFO","city_src":"Houston","city_dst":"San Francisco","state_src":"TX","state_dst":"CA"}	{"id":"IAH","City":"Houston","State":"TX","Country":"USA"}	{"id":"SFO","City":"San Francisco","State":"CA","Country":"USA"}	{" 13 Fra
{"tripid":1011126,"localdate":"2014-01-01T11:26:00.000+0000","delay":-4,"distance":1421,"src":"IAH","dst":"SFO","city_src":"Houston","city_dst":"San Francisco","state_src":"TX","state_dst":"CA"}	{"id":"IAH","City":"Houston","State":"TX","Country":"USA"}	{"id":"SFO","City":"San Francisco","State":"CA","Country":"USA"}	{" 03 Fra
{"tripid":1011126,"localdate":"2014-01-01T11:26:00.000+0000","delay":-4,"distance":1421,"src":"IAH","dst":"SFO","city_src":"Houston","city_dst":"San Francisco","state_src":"TX","state_dst":"CA"}	{"id":"IAH","City":"Houston","State":"TX","Country":"USA"}	{"id":"SFO","City":"San Francisco","State":"CA","Country":"USA"}	{" 09 Fra
{"tripid":1011126,"localdate":"2014-01-01T11:26:00.000+0000","delay":-4,"distance":1421,"src":"IAH","dst":"SFO","city_src":"Houston","city_dst":"San Francisco","state_src":"TX","state_dst":"CA"}	{"id":"IAH","City":"Houston","State":"TX","Country":"USA"}	{"id":"SFO","City":"San Francisco","State":"CA","Country":"USA"}	{" 19 Fra

Showing the first 1000 rows.

Command took 5.98s

Disclaimer: *In reality, we would need a more careful analysis to draw conclusions about SFO!*

City / Flight Relationships through Motif Finding

What delays might we blame on SFO?

```
> motifs = tripGraph.find("(a)-[e1]->(b); (b)-[e2]->(c)")\
    .filter("(b.id = 'SFO') and (e1.delay > 500 or e2.delay > 500) and e1.tripid < e2.tripid")
  display(motifs)
```

e1	a	b	e2
▾ object tripid: 1011126 localdate: 2014-01-01T11:26:00.000+0000 delay: -4 distance: 1421 src: IAH dst: SFO city_src: Houston city_dst: San Francisco state_src: TX state_dst: CA	▸ {"id":"IAH","City":"Houston","State":"TX","Country":"USA"}	▸ {"id":"SFO","City":"San Francisco","State":"CA","Country":"USA"}	▸ {" 137 Fra
▸ {"tripid":1011126,"localdate":"2014-01-	▸ {"id":"IAH","City":"Houston","State":"TX","Country":"USA"}	▸ {"id":"SFO","City":"San	▸ {"

Showing the first 1000 rows.

Command took 5.98s -- by joseph@databricks.com at 4/11/2016, 11:29:21 AM on unknown cluster (0 GB)

Disclaimer: *In reality, we would need a more careful analysis to draw conclusions about SFO!*

Graph algorithms

Find important vertices
- PageRank

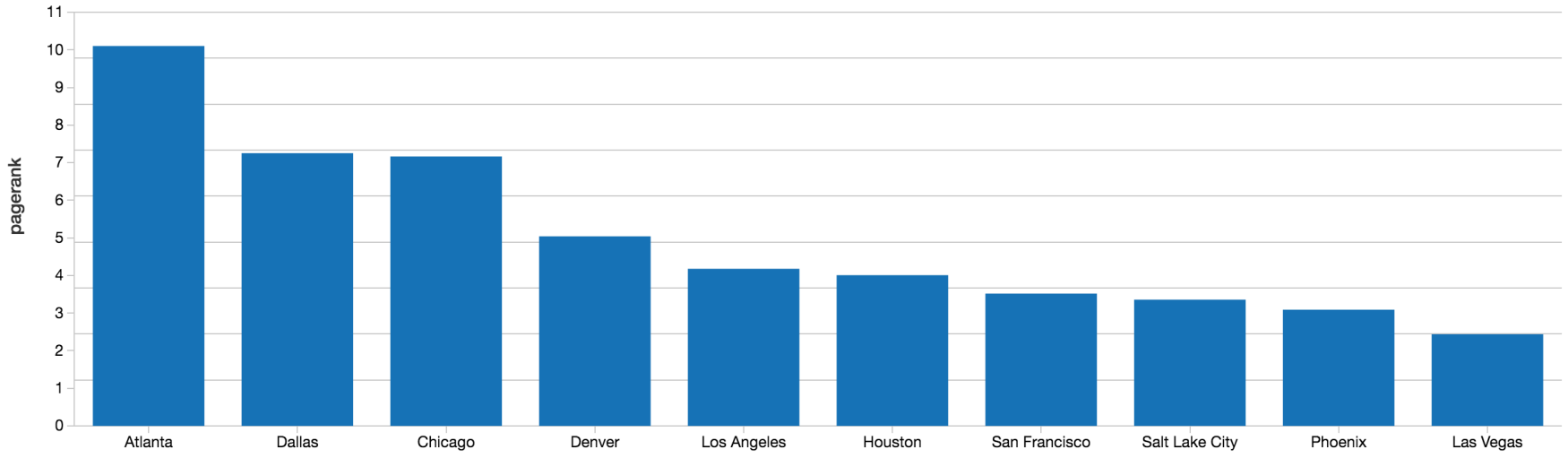
Find paths between sets of vertices
- Breadth-first search (BFS)
- Shortest paths

Find groups of vertices (components, communities)
- Connected components
- Strongly connected components
- Label Propagation Algorithm (LPA)

Other
- Triangle counting
- SVDPlusPlus

Determining airport importance using PageRank

```
> ranks = tripGraph.pageRank(maxIter=5)
  display(ranks.vertices
          .sort(ranks.vertices.pagerank.desc())
          .limit(10))
```



Command took 23.56s

Algorithm implementations

Mostly wrappers for GraphX

- PageRank
- Shortest paths
- Connected components
- Strongly connected components
- Label Propagation Algorithm (LPA)
- SVDPlusPlus

Some algorithms implemented using DataFrames

- Breadth-first search
- Triangle counting

Saving & loading graphs

Save & load the DataFrames.

```
vertices = sqlContext.read.parquet(...)
edges = sqlContext.read.parquet(...)
g = GraphFrame(vertices, edges)
g.vertices.write.parquet(...)
g.edges.write.parquet(...)
```

In the future...

- SQL data sources for graph formats

APIs: Scala, Java, Python

API available from all 3 languages

→ First time GraphX functionality has been available to Java & Python users

2 missing items (WIP)

- Java-friendliness is currently in alpha.
- Python does not have aggregateMessages (for implementing your own graph algorithms).

Outline

GraphFrames overview

GraphFrames vs. GraphX and other libraries

Details for power users

Roadmap and resources

2 types of graph libraries

Graph algorithms

Standard & custom algorithms
Optimized for batch processing

Graph queries

Motif finding
Point queries & updates

GraphFrames: Both algorithms & queries (but not point updates)

GraphFrames vs. GraphX

	GraphFrames	GraphX
Built on	DataFrames	RDDs
Languages	Scala, Java, Python	Scala
Use cases	Queries & algorithms	Algorithms
Vertex IDs	Any type (in Catalyst)	Long
Vertex/edge attributes	Any number of DataFrame columns	Any type (VD, ED)
Return types	GraphFrame or DataFrame	Graph[VD, ED], or RDD[Long, VD]

GraphX compatibility

Simple conversions between GraphFrames & GraphX.

```
val g: GraphFrame = ...

// Convert GraphFrame → GraphX
val gx: Graph[Row, Row] = g.toGraphX

// Convert GraphX → GraphFrame
val g2: GraphFrame = GraphFrame.fromGraphX(gx)
```

Vertex & edge attributes are Rows in order to handle non-Long IDs

Wrapping existing GraphX code: See Belief Propagation example:
https://github.com/graphframes/graphframes/blob/master/src/main/scala/org/graphframes/examples/BeliefPropagation.scala

Outline

GraphFrames overview

GraphFrames vs. GraphX and other libraries

Details for power users

Roadmap and resources

Scalability

Current status

- DataFrame-based parts benefit from DataFrame scalability + performance optimizations (Catalyst, Tungsten).
- GraphX wrappers are as fast as GraphX (+ conversion overhead).

WIP

- GraphX has optimizations which are not yet ported to GraphFrames.
- See next slide…

WIP optimizations

Join elimination

- GraphFrame algorithms require lots of joins.
- Not all joins are necessary

Solution:

- Vertex IDs serve as unique keys.
- Tracking keys allows Catalyst to eliminate some joins.

Materialized views

- Data locality for common use cases
- Message-passing algorithms often need “triplet view” (src, edge, dst)

Solution:

- Materialize specific views
- Analogous to GraphX’s “replicated vertex view”

For more info & benchmark results, see Ankur Dave’s SSE 2016 talk.

https://spark-summit.org/east-2016/events/graphframes-graph-queries-in-spark-sql/

Implementing new algorithms

Method 1: DataFrame & GraphFrame operations

Motif finding

- Series of DataFrame joins

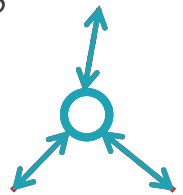
Triangle count

- DataFrame ops + motif finding

BFS

- DataFrame joins & filters

Method 2: Message passing



`aggregateMessages`

- Same primitive as GraphX
- Specify messages & aggregation using DataFrame expressions

Belief propagation example code

Outline

GraphFrames overview

GraphFrames vs. GraphX and other libraries

Details for power users

Roadmap and resources

Current status

Published

- Open source (Apache 2.0) on Github https://github.com/graphframes/graphframes
- Spark package http://spark-packages.org/package/graphframes/graphframes

Compatible

- Spark 1.4, 1.5, 1.6
- Databricks Community Edition

Documented

- http://graphframes.github.io/

Roadmap

- Merge WIP speed optimizations
- Java API tests & examples
- Migrate more algorithms to DataFrame-based implementations for greater scalability
- Get community feedback!

Contribute

- Tracking issues on Github
- Thanks to those who have already sent pull requests!

Resources for learning more

User guide + API docs http://graphframes.github.io/

- Quick-start
- Overview & examples for all algorithms
- Also available as executable notebooks:
  - Scala: http://go.databricks.com/hubfs/notebooks/3-GraphFrames-User-Guide-scala.html
  - Python: http://go.databricks.com/hubfs/notebooks/3-GraphFrames-User-Guide-python.html

Blog posts

- Intro: https://databricks.com/blog/2016/03/03/introducing-graphframes.html
- Flight delay analysis: https://databricks.com/blog/2016/03/16/on-time-flight-performance-with-spark-graphframes.html


COMING TO SF
THIS JUNE.
#sparksummit
Spark summit
SPARK SUMMIT 2016
JUNE 6-8, 2016 • SAN FRANCISCO
REGISTER TODAY
ORGANIZED BY
databricks

Thank you!

Thanks to
- Denny Lee & Bill Chambers (demo)
- Tim Hunter, Xiangrui Meng, Ankur Dave & others (GraphFrames development)