O'REILLY®
oscon

Scalable graph analysis with Apache Giraph and Spark GraphX

oscon.com
#oscon

Roman Shaposhnik rvs@apache.org @rhatr
Director of Open Source, Pivotal Inc.

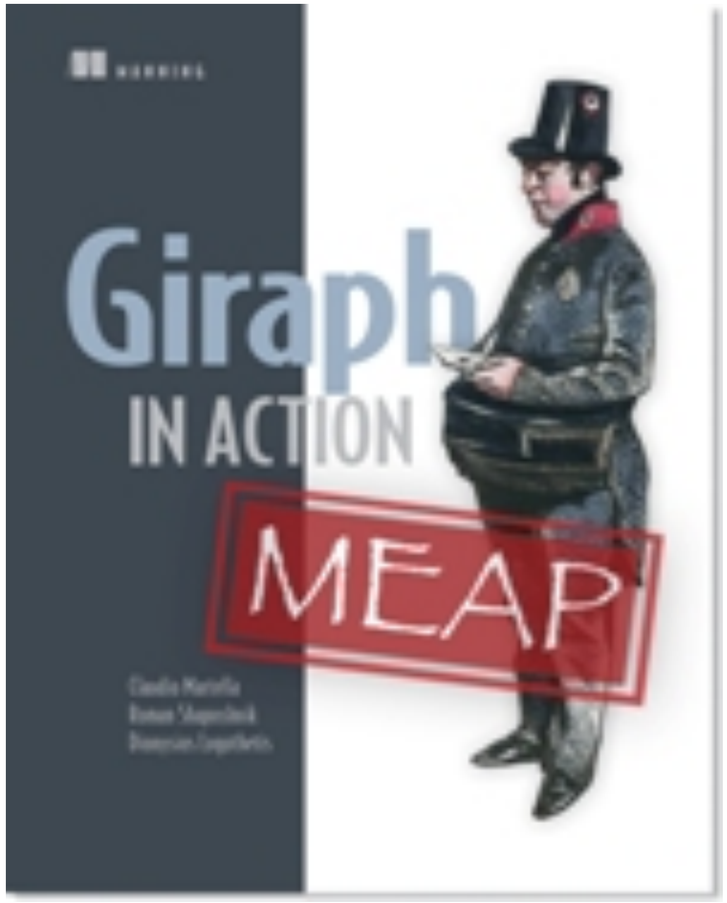
O'REILLY®
oscon

Introduction into scalable graph analysis with Apache Giraph and Spark GraphX

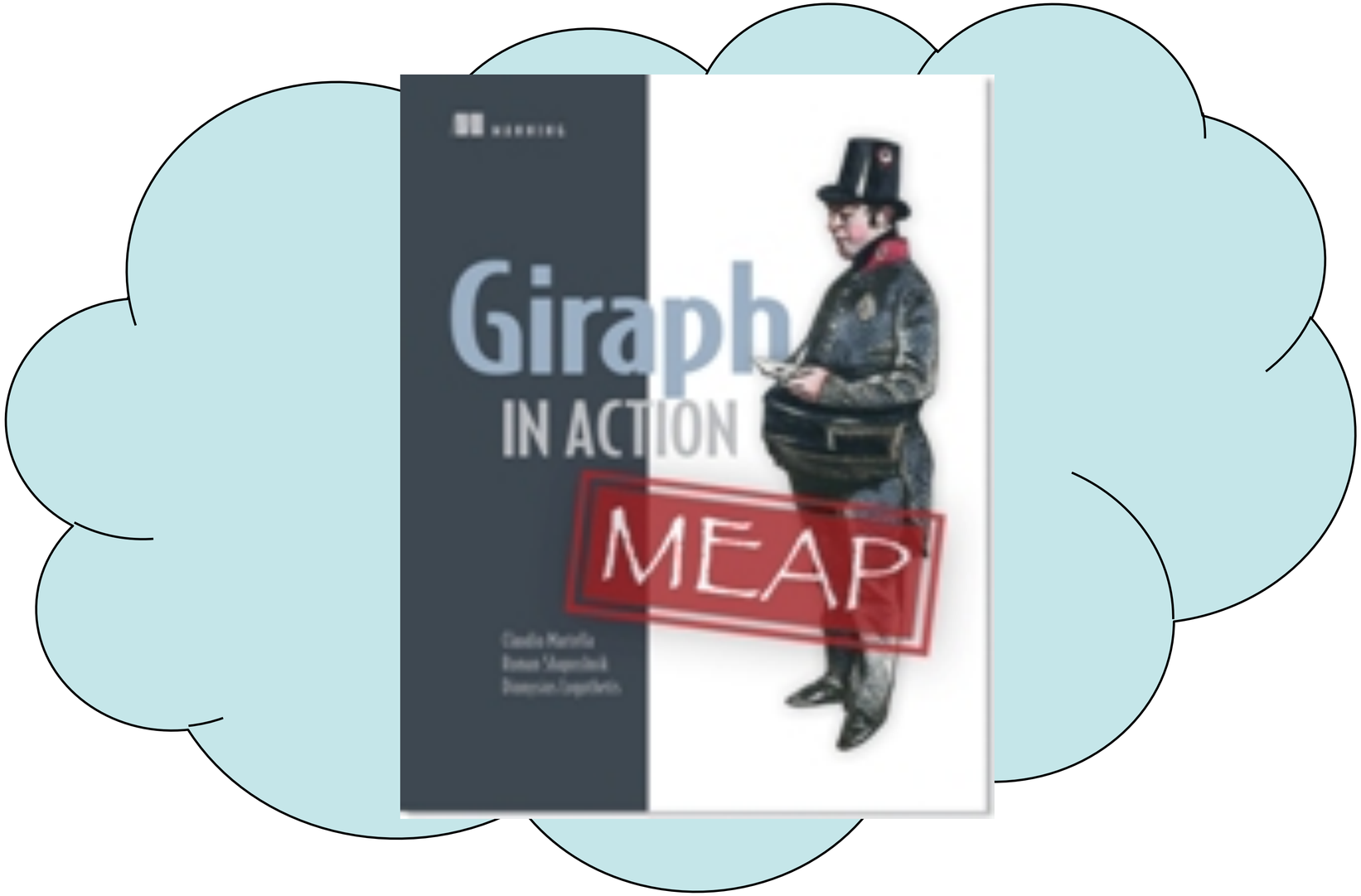
oscon.com
#oscon

Roman Shaposhnik rvs@apache.org @rhatr
Director of Open Source, Pivotal Inc.

Shameless plug #1



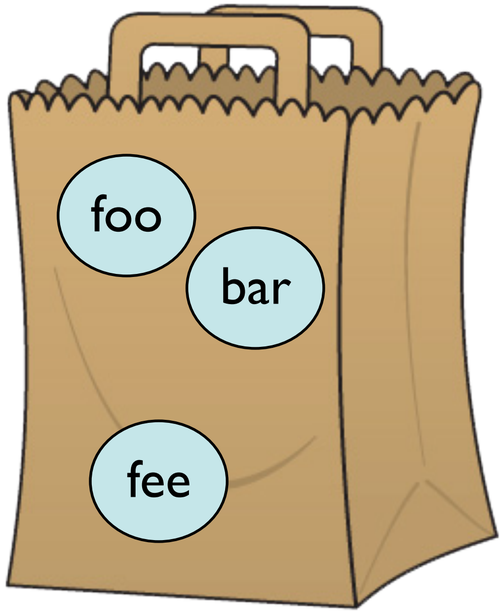
Shameless plug #1



Agenda:

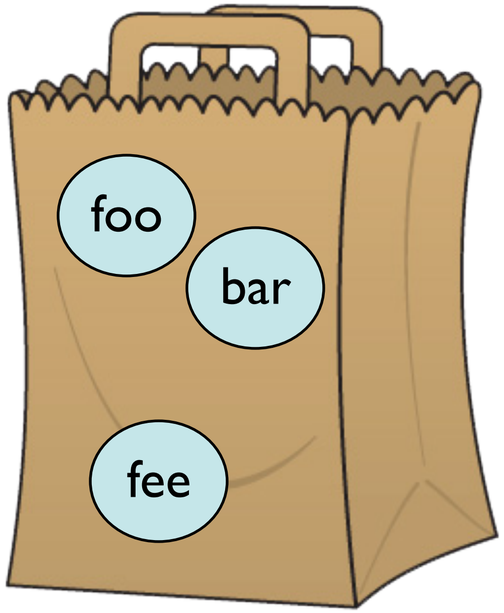
Lets define some terms

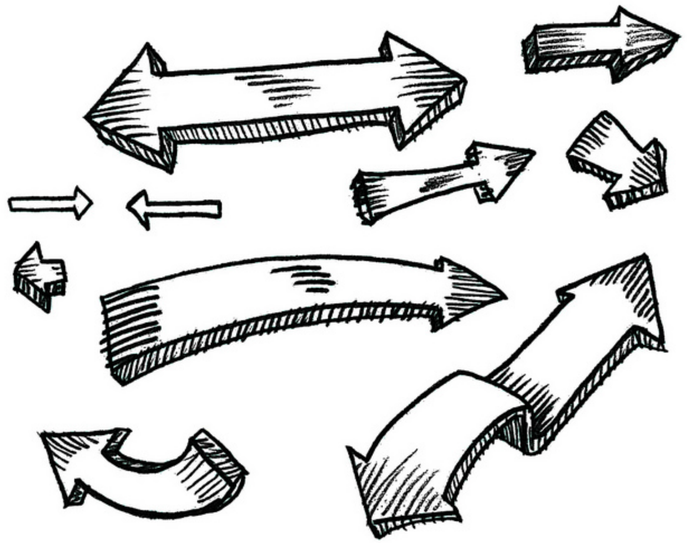
- Graph is a G = (V, E), where E ⊆ VxV
- Directed multigraphs with properties attached to each vertex and edge



Lets define some terms

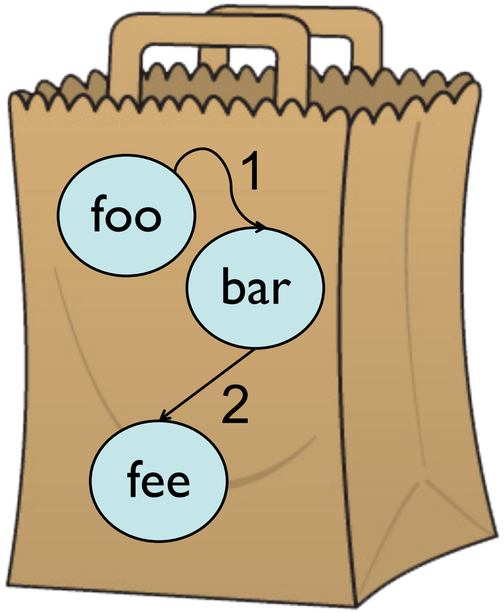
- Graph is a G = (V, E), where E ⊆ VxV
- Directed multigraphs with properties attached to each vertex and edge





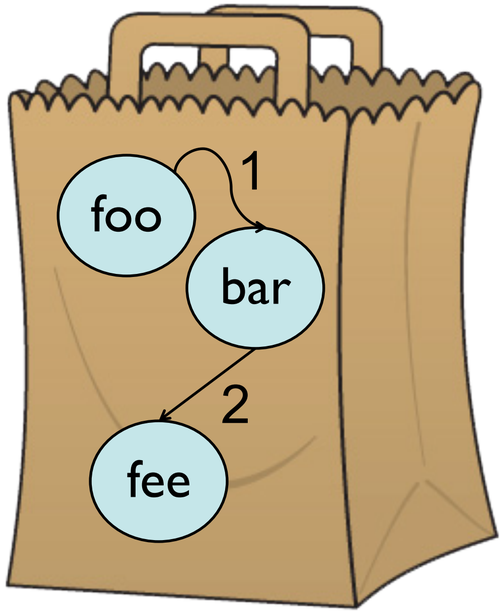
Lets define some terms

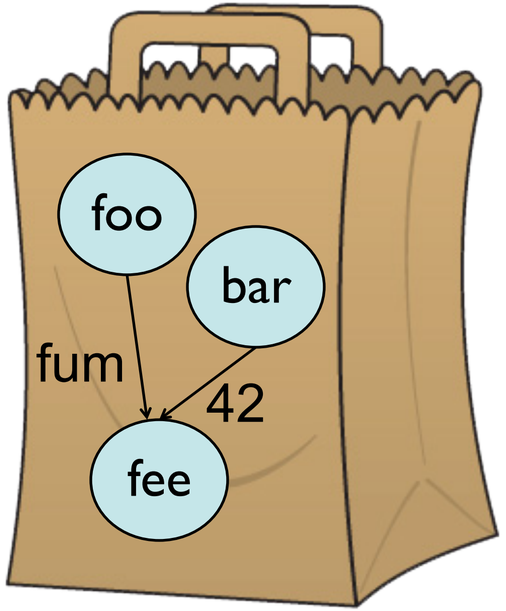
- Graph is a G = (V, E), where E $\subseteq$ VxV
- Directed multigraphs with properties attached to each vertex and edge



Lets define some terms

- Graph is a G = (V, E), where E ⊆ VxV
- Directed multigraphs with properties attached to each vertex and edge





What kind of graphs are we talking about?

- Page ranking on Facebook social graph (mid 2013)
  - 10^9 (billions) vertices
  - 10^12 (trillion) edges
  - 10^15 (petabtybe) cold storage data scale
  - 200 servers
  - …all in under 4 minutes!

O'REILLY®
oscon

“On day one Doug created HDFS and MapReduce”

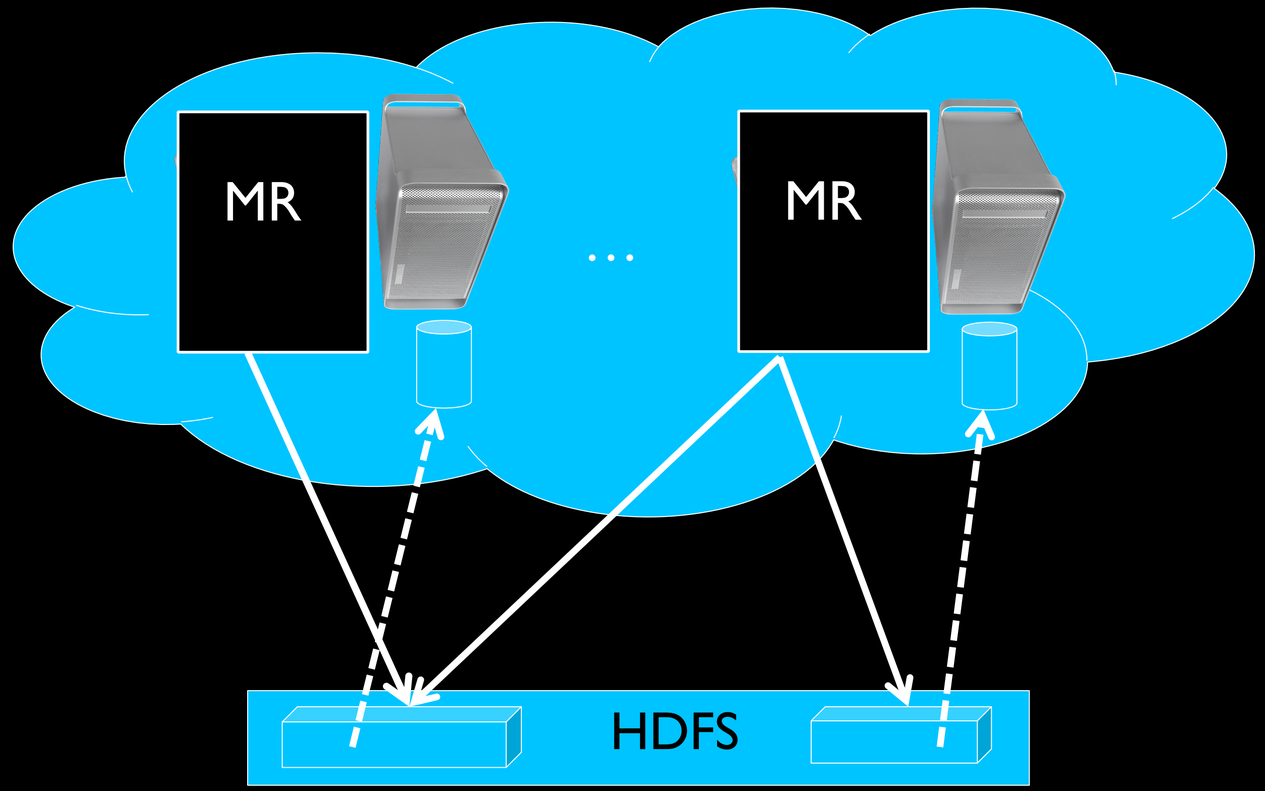
oscon.com
#oscon

Google papers that started it all

- GFS (file system)
  - distributed
  - replicated
  - non-POSIX

- MapReduce (computational framework)
  - distributed
  - batch-oriented (long jobs; final results)
  - data-gravity aware
  - designed for “embarrassingly parallel” algorithms

HDFS pools and abstracts direct-attached storage



A Unix analogy

- It is as though instead of:

```
$ grep foo bar.txt | tr “,” “ “ | sort -u
```

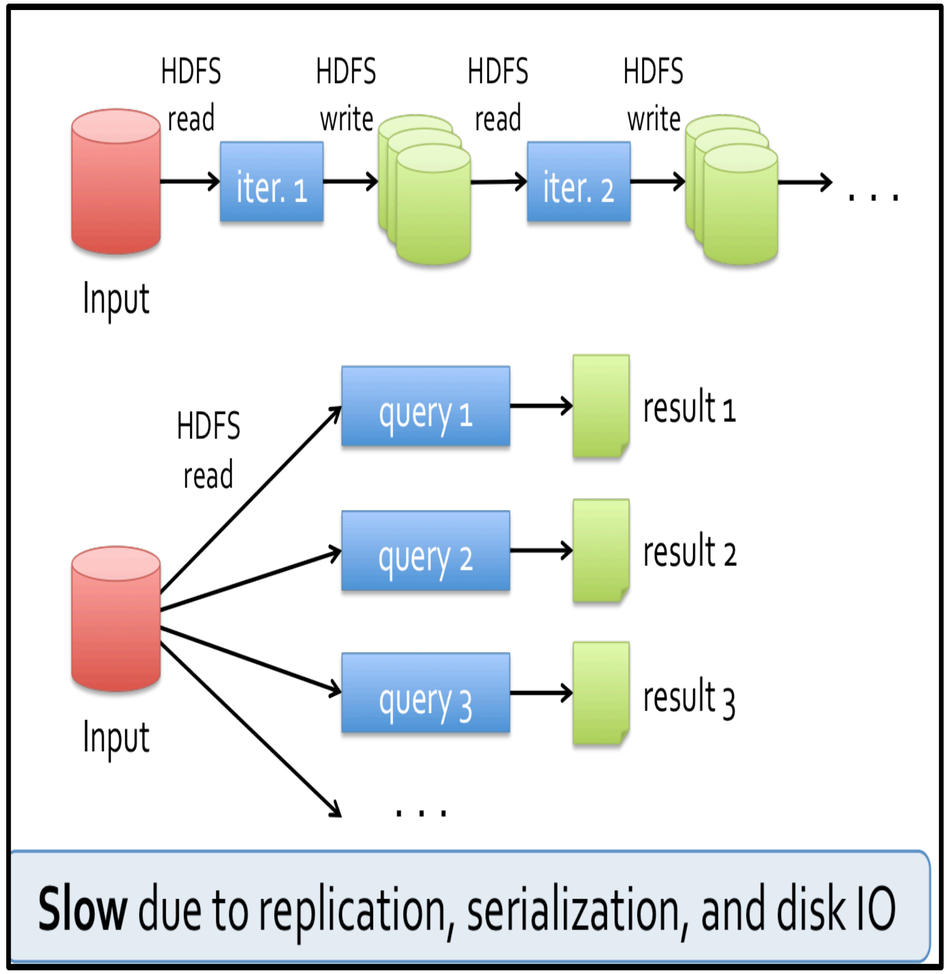
- We are doing:

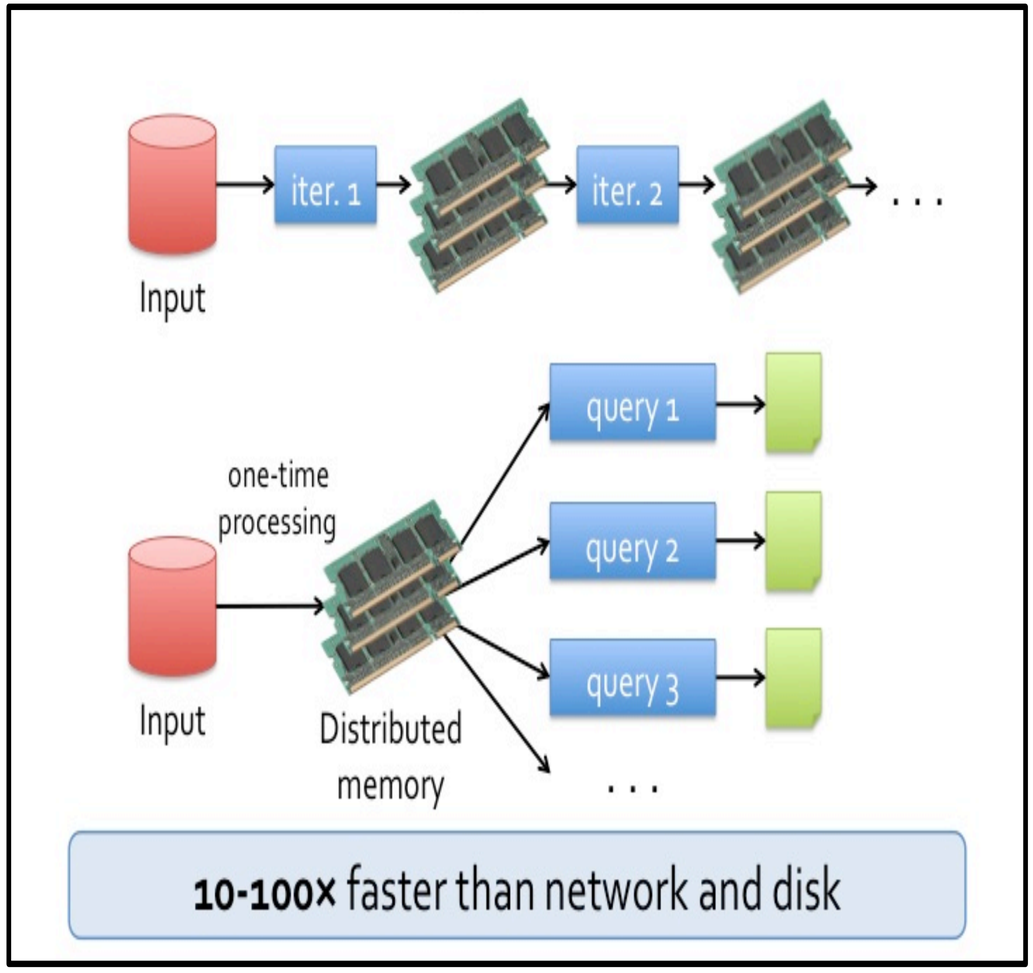
```
$ grep foo < bar.txt > /tmp/1.txt
$ tr “,” “ “  < /tmp/1.txt > /tmp/2.txt
$ sort –u < /tmp/2.txt
```

O'REILLY®
oscon

Enter Apache Spark

RAM is the new disk, Disk is the new tape

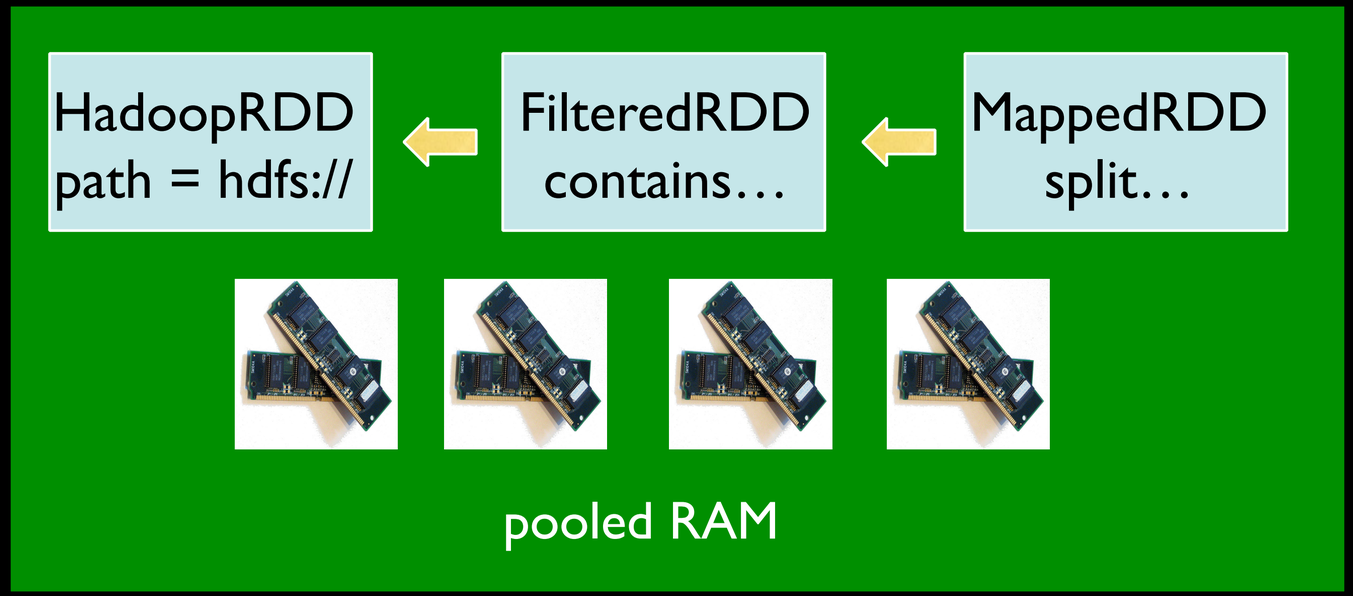




Source: UC Berkeley Spark project (just the image)

RDDs instead of HDFS files, RAM instead of Disk

```
warnings = textFile(…).filter(_.contains(“warning”))
                      .map(_.split(‘ ’)(1))
```



RDDs: resilient, distributed, datasets

- Distributed on a cluster in RAM
- Immutable (mostly)
- Can be evicted, snapshotted, etc.
- Manipulated via parallel operators (map, etc.)
- Automatically rebuilt on failure
- A parallel ecosystem
- A solution to iterative and multi-stage apps

O'REILLY®
oscon

What's so special about Graphs and big data?

oscon.com
#oscon

Graph relationships

- Entities in your data: tuples
  - customer data
  - product data
  - interaction data

- Connection between entities: graphs
  - social network or my customers
  - clustering of customers vs. products

A word about Graph databases

- Plenty available
  - Neo4J, Titan, etc.
- Benefits
  - Query language
  - Tightly integrate systems with few moving parts
  - High performance on known data sets
- Shortcomings
  - Not easy to scale horizontally
  - Don’t integrate with HDFS
  - Combine storage and computational layers
  - A sea of APIs

What's the key API?

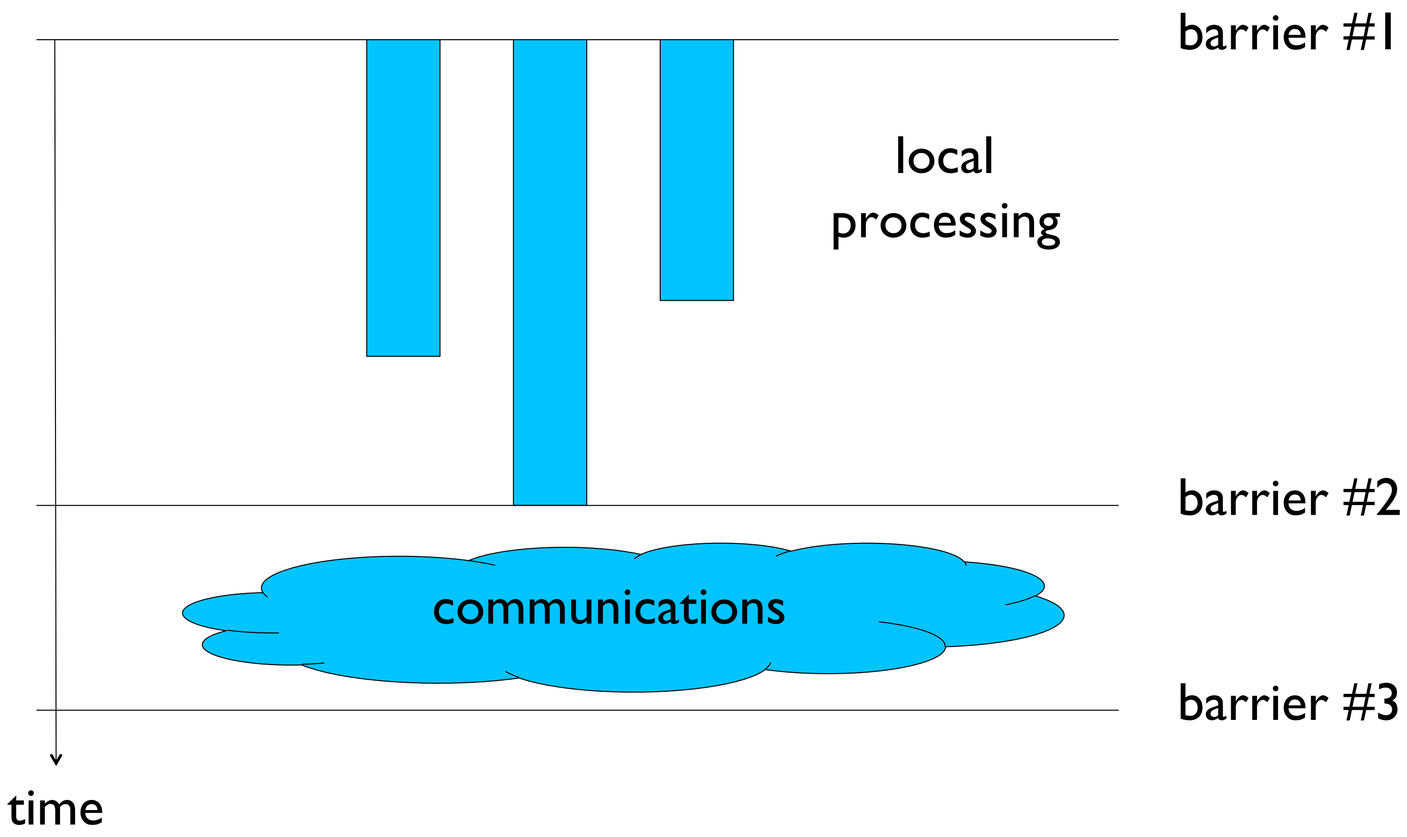
- Directed multi-graph with labels attached to vertices and edges
- Defining vertices and edges dynamically
- Selecting sub-graphs
- Mutating the topology of the graph
- Partitioning the graph
- Computing model that is
  - iterative
  - scalable (shared nothing)
  - resilient
  - easy to manage at scale

O'REILLY®
oscon

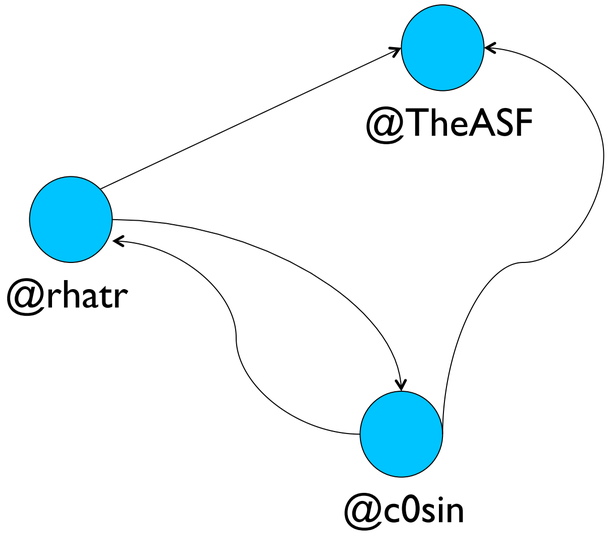
Bulk Synchronous Parallel
BSP compute model

oscon.com
#oscon

BSP in a nutshell



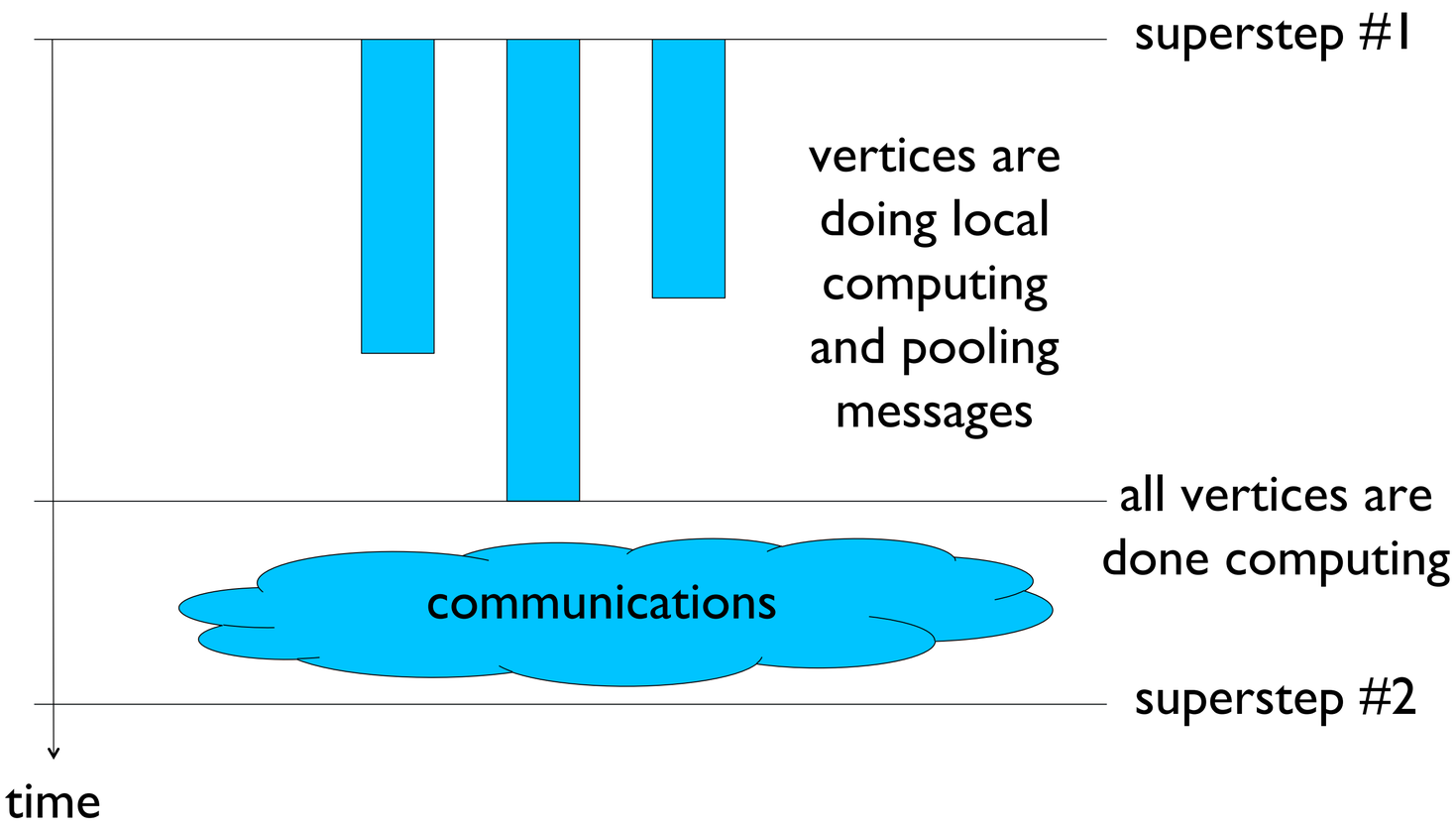
Vertex-centric BSP application



"Think like a vertex"

- I know my local state
- I know my neighbors
- I can send messages to vertices
- I can declare that I am done
- I can mutate graph topology

Local state, global messaging

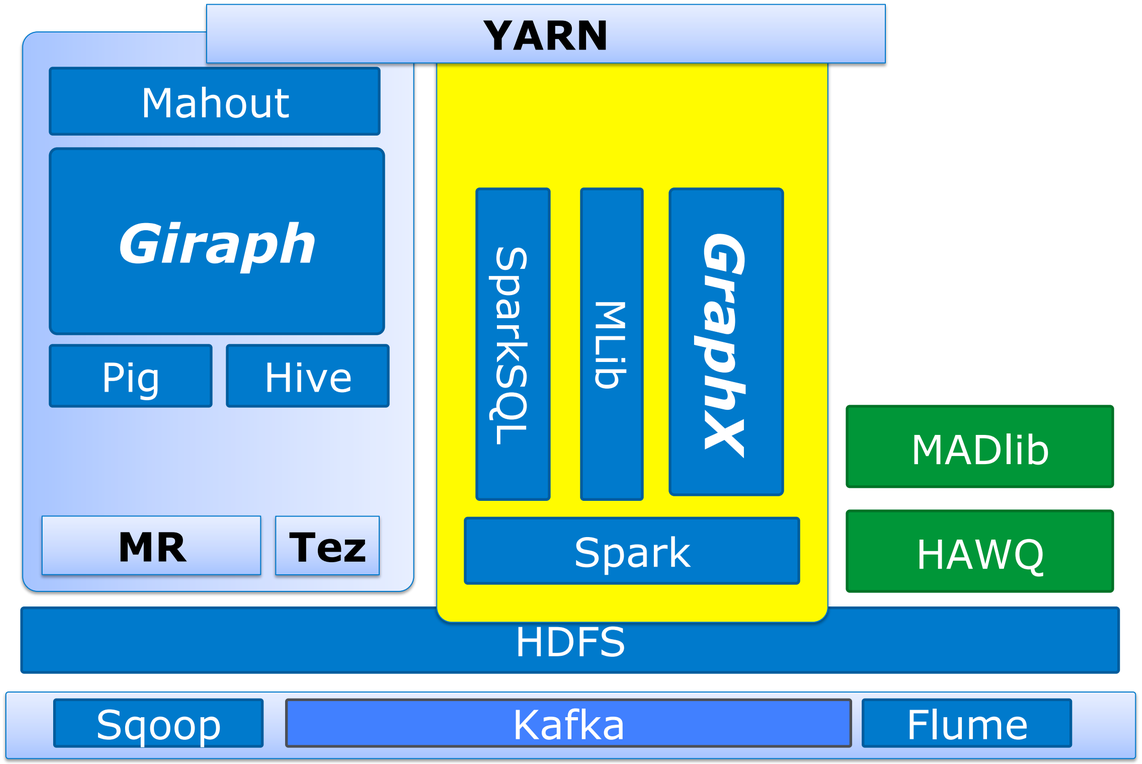


O'REILLY®
oscon

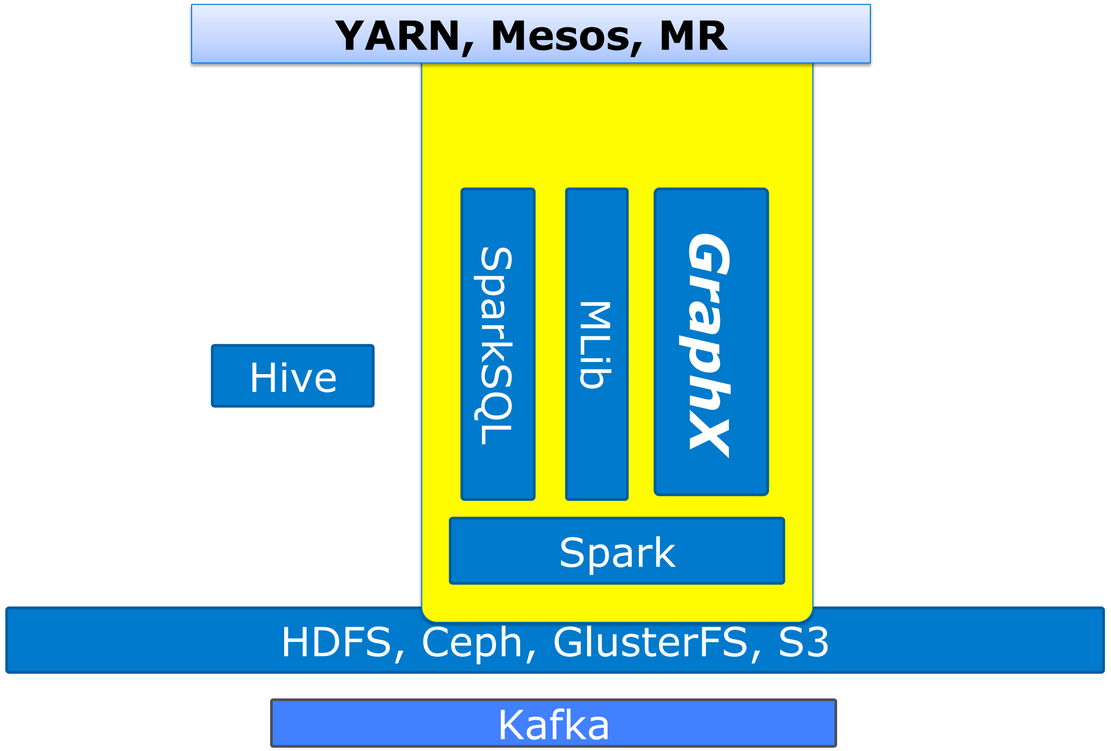
Lets put it all together

oscon.com
#oscon

Hadoop ecosystem view



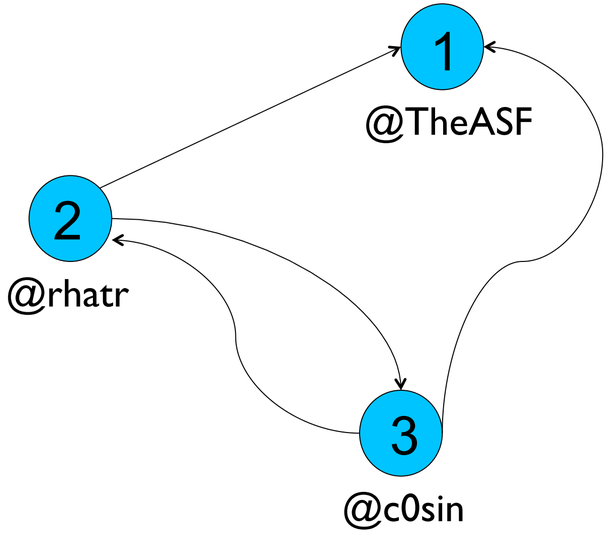
Spark view


YARN, Mesos, MR
SparkSQL
MLib
GraphX
Hive
Spark
HDFS, Ceph, GlusterFS, S3
Kafka

O'REILLY®
oscon

Enough boxology!
Lets look at some code

Our toy for the rest of this talk



Adjacency lists stored on HDFS

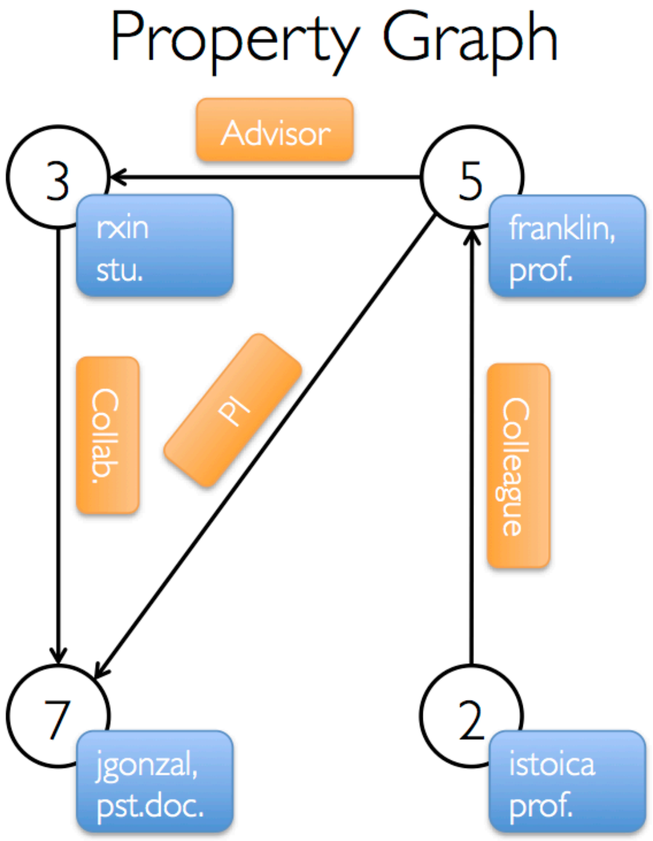
```
$ hadoop fs –cat /tmp/graph/1.txt
 1
 2 1 3
 3 1 2
```

Graph modeling in GraphX

- The property graph is parameterized over the vertex (VD) and edge (ED) types

```
class Graph[VD, ED] {
    val vertices: VertexRDD[VD]
    val edges: EdgeRDD[ED]
}
```

- Graph[(String, String), String]



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Hello world in GraphX

```
$ spark*/bin/spark-shell

scala> val inputFile = sc.textFile(“hdfs:///tmp/graph/1.txt”)

scala> val edges = inputFile.flatMap(s => {                    // “2 1 3”
             val l = s.split("\t");                            // [ “2”, “1”, “3” ]
             l.drop(1).map(x => (l.head.toLong, x.toLong))   // [ (2, 1), (2, 3) ]
           })

scala> val graph = Graph.fromEdgeTuples(edges, "")    // Graph[String, Int]

scala> val result = graph.collectNeighborIds(EdgeDirection.Out).map(x =>
           println("Hello world from the: " + x._1 + " : " + x._2.mkString(" ")) )

scala> result.collect() // don’t try this @home
```

Hello world from the: 1 :

Hello world from the: 2 : 1 3

Hello world from the: 3 : 1 2

Graph modeling in Giraph

```
BasicComputation<I extends WritableComparable,     // VertexID   -- vertex ref
                 V extends Writable,               // VertexData -- a vertex datum
                 E extends Writable,               // EdgeData   -- an edge label
                 M extends Writable>               // MessageData-- message payload
```

V is sort of like VD
E is sort of like ED

Hello world in Giraph

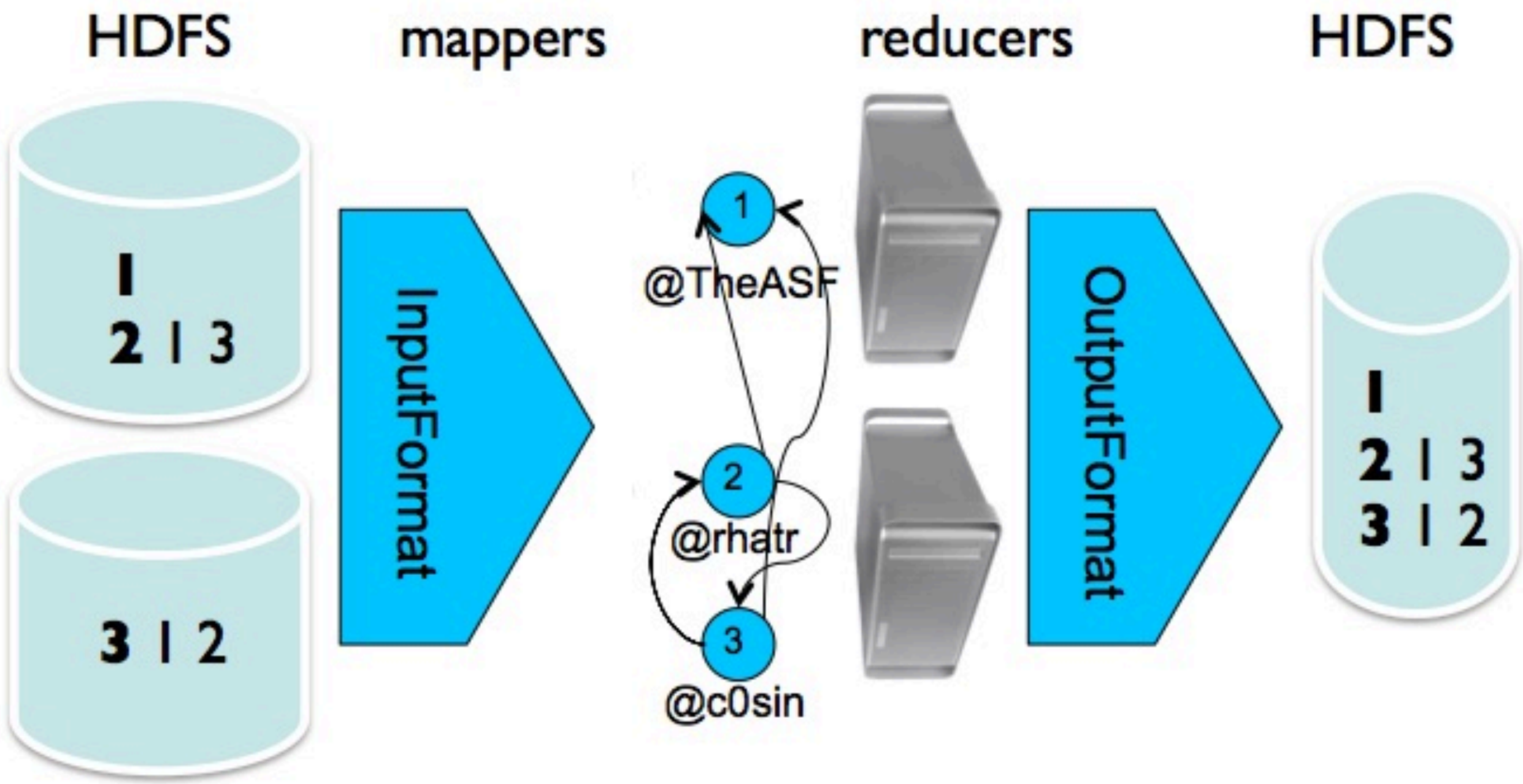
```
public class GiraphHelloWorld extends
    BasicComputation<IntWritable, IntWritable, NullWritable, NullWritable> {
    public void compute(Vertex<IntWritable, IntWritable, NullWritable> vertex,
                                Iterable<NullWritable> messages) {
      System.out.print("Hello world from the: " + vertex.getId() + " : ");
      for (Edge<IntWritable, NullWritable> e : vertex.getEdges()) {
        System.out.print(" " + e.getTargetVertexId());
      }
      System.out.println("");
      vertex.voteToHalt();
    }
}
```

How to run it

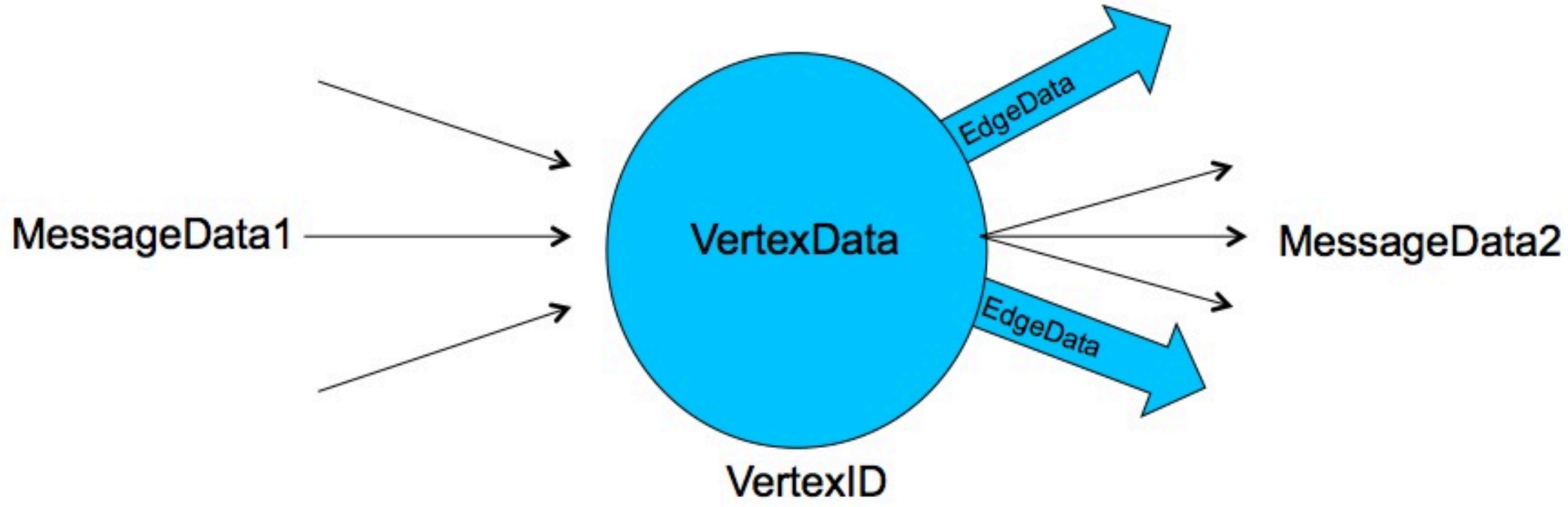
```
$ giraph target/*.jar giraph.GiraphHelloWorld \
   -vip /tmp/graph/ \
   -vif org.apache.giraph.io.formats.IntIntNullTextInputFormat  \
   -w 1 \
   -ca giraph.SplitMasterWorker=false,giraph.logLevel=error
```

```
Hello world from the: 1 :
Hello world from the: 2 : 1 3
Hello world from the: 3 : 1 2
```

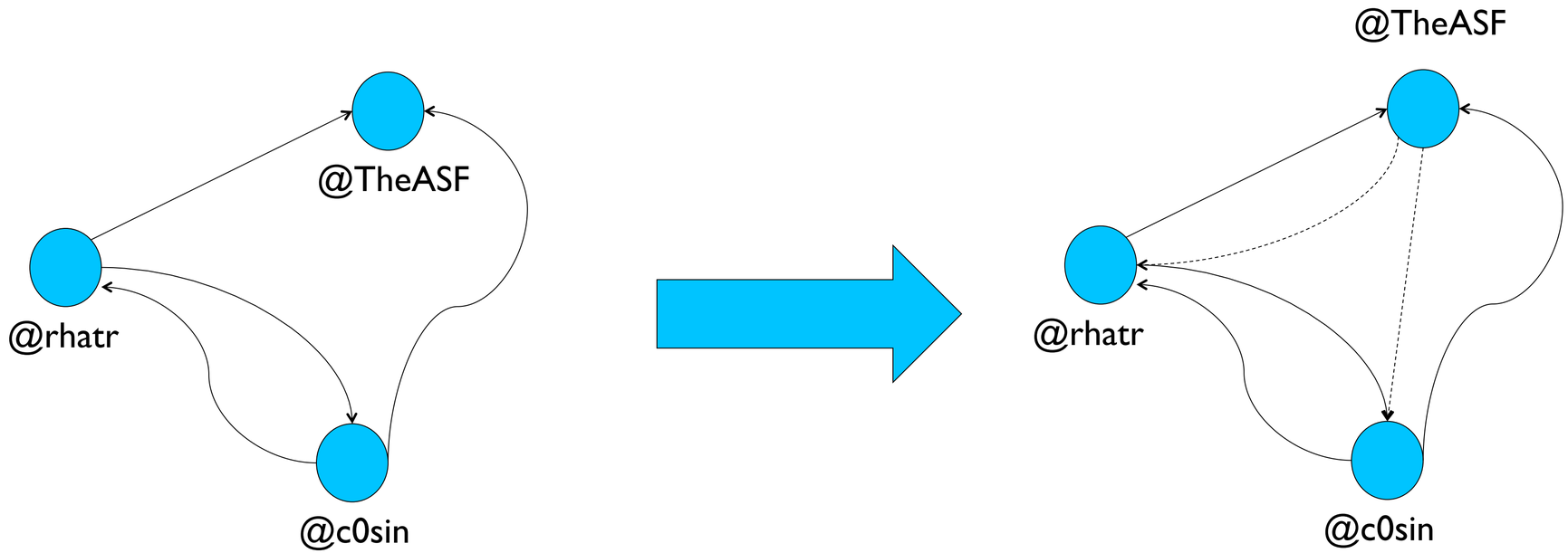
Anatomy of Giraph run



BSP assumes an exclusively vertex view



Turning Twitter into Facebook



Hello world in Giraph

```
public void compute(Vertex<Text, DoubleWritable, DoubleWritable> vertex, Iterable<Text> ms ){
        if (getSuperstep() == 0) {
           sendMessageToAllEdges(vertex, vertex.getId());
        } else {
            for (Text m : ms) {
               if (vertex.getEdgeValue(m) == null) {
                 vertex.addEdge(EdgeFactory.create(m, SYNTHETIC_EDGE));
               }
           }
        }
        vertex.voteToHalt();
}
```

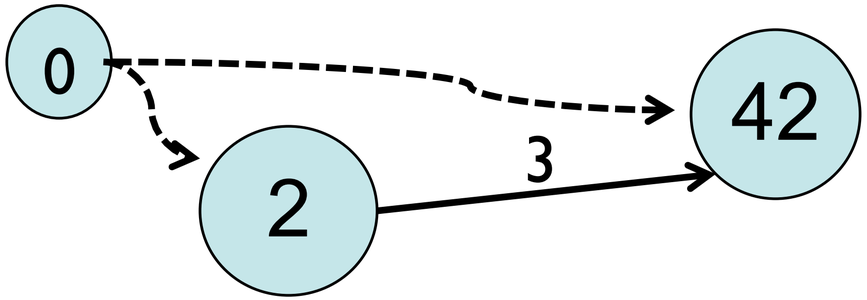
O'REILLY®
oscon

BSP in GraphX

oscon.com
#oscon

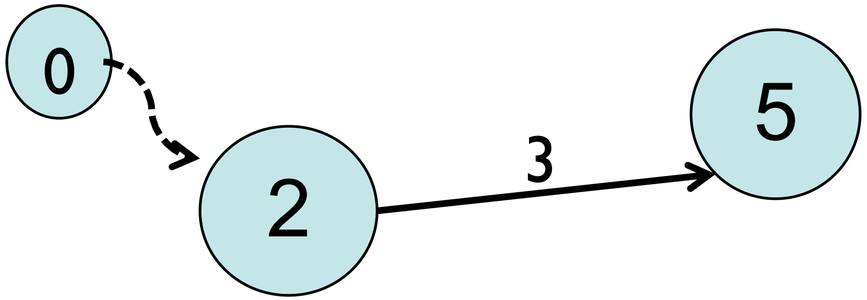
Single source shortest path

```
scala> val sssp = graph.pregel(Double.PositiveInfinity)   // Initial message
((id, dist, newDist) => math.min(dist, newDist),          // Vertex Program
  triplet => {                                            // Send Message
    if (triplet.srcAttr + triplet.attr < triplet.dstAttr) {
      Iterator((triplet.dstId, triplet.srcAttr + triplet.attr))
    } else {
      Iterator.empty
    }
  },
  (a,b) => math.min(a,b))                                 // Merge Messages
scala> println(sssp.vertices.collect.mkString("\n"))
```

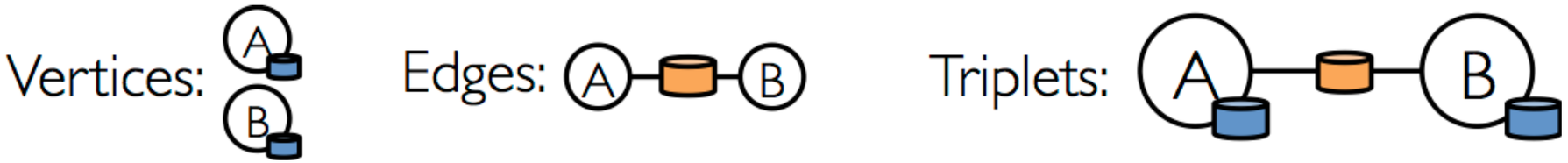


Single source shortest path

```
scala> val sssp = graph.pregel(Double.PositiveInfinity)   // Initial message
((id, dist, newDist) => math.min(dist, newDist),          // Vertex Program
  triplet => {                                            // Send Message
    if (triplet.srcAttr + triplet.attr < triplet.dstAttr) {
      Iterator((triplet.dstId, triplet.srcAttr + triplet.attr))
    } else {
      Iterator.empty
    }
  },
  (a,b) => math.min(a,b))                                 // Merge Messages
scala> println(sssp.vertices.collect.mkString("\n"))
```



Operational views of the graph



Masking instead of mutation

- def subgraph(

    epred: EdgeTriplet[VD,ED] => Boolean = (x => true),

    vpred: (VertexID, VD) => Boolean = ((v, d) => true))

  : Graph[VD, ED]

- def mask[VD2, ED2](other: Graph[VD2, ED2]): Graph[VD, ED]

Built-in algorithms

- def pageRank(tol: Double, resetProb: Double = 0.15):
  Graph[Double, Double]
- def connectedComponents(): Graph[VertexID, ED]
- def triangleCount(): Graph[Int, ED]
- def stronglyConnectedComponents(numIter: Int): Graph[VertexID, ED]

Final thoughts

Giraph

- An unconstrained BSP framework
- Specialized fully mutable, dynamically balanced in-memory graph representation
- Very procedural, vertex-centric programming model
- Genuine part of Hadoop ecosystem
- Definitely a 1.0

GraphX

- An RDD framework
- Graphs are “views” on RDDs and thus immutable
- Functional-like, “declarative” programming model
- Genuine part of Spark ecosystem
- Technically still an alpha

O'REILLY®
oscon

Q&A

Thanks!

oscon.com
#oscon