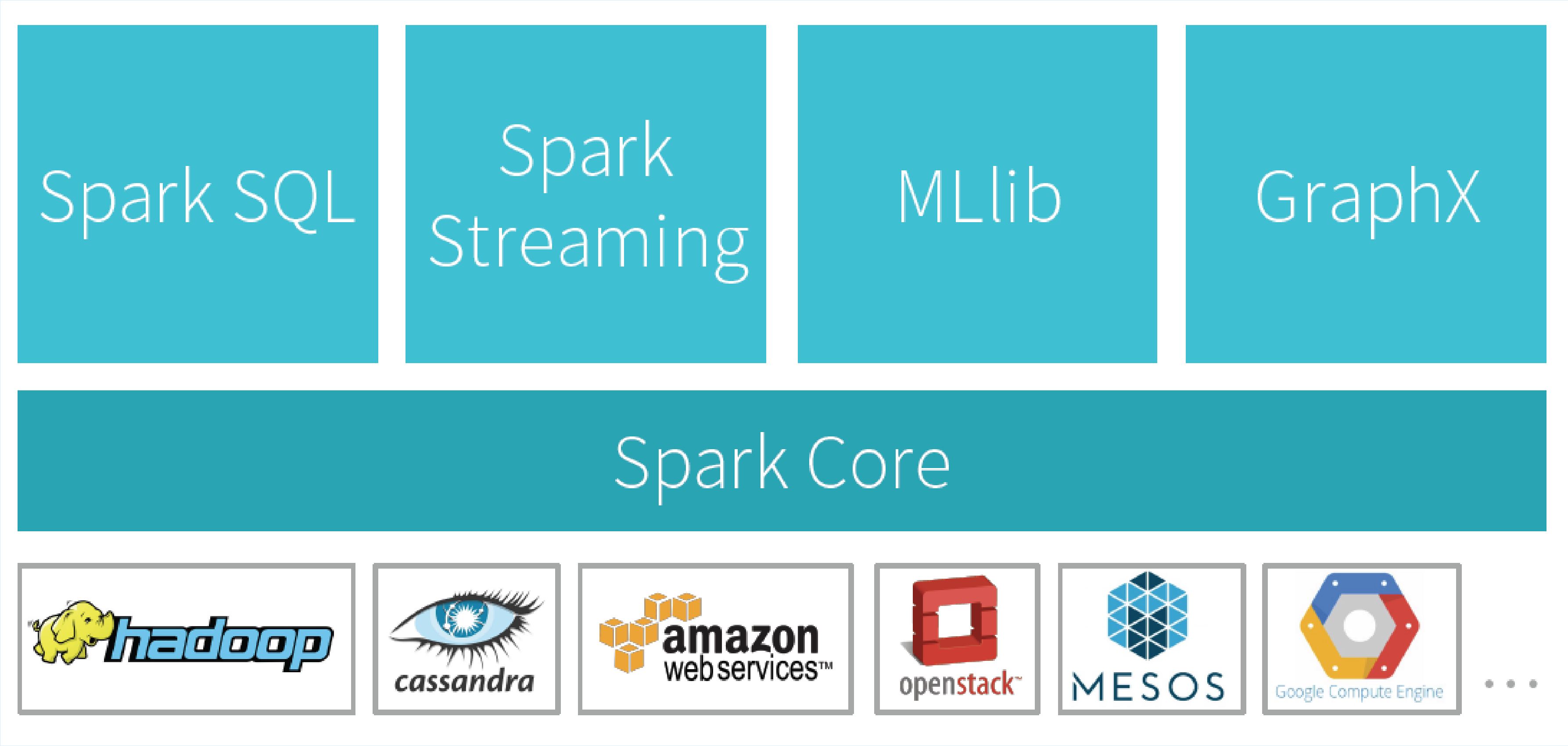
Graph Analytics Using GraphFrames and GraphX

Spark



Spark SQL

Spark Streaming

MLlib

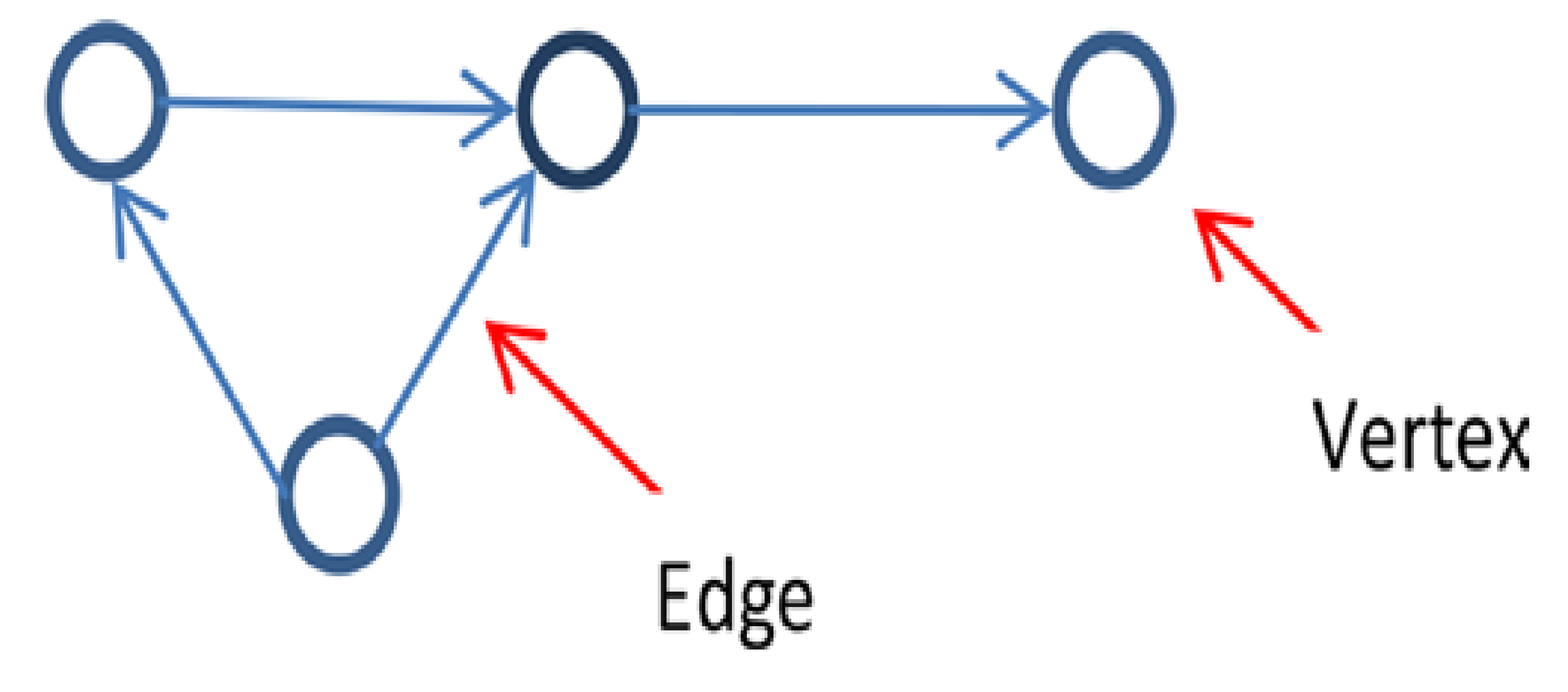
GraphX

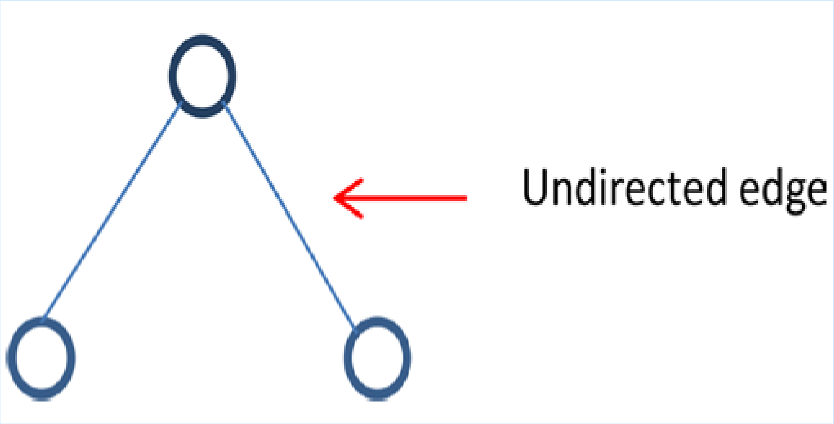
Spark Core

Graphs

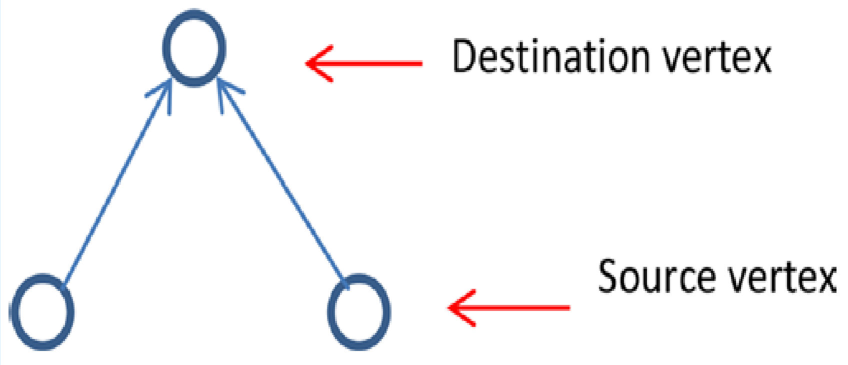
- Graphs “also called networks” are ubiquitous for web applications (and other use cases as well)
- Consists of nodes and edges
  - Node (or vertex) is an object
  - Edge is relation between any two objects
  - Could be directed or undirected
- Eg. Facebook network, LinkedIn, IOT – Graph of Graphs
- Why graphs?
  - Powerful and concise way to explore semantic relationships between entities
  - Allows us to see how communities evolve (topology analysis)

Terminology



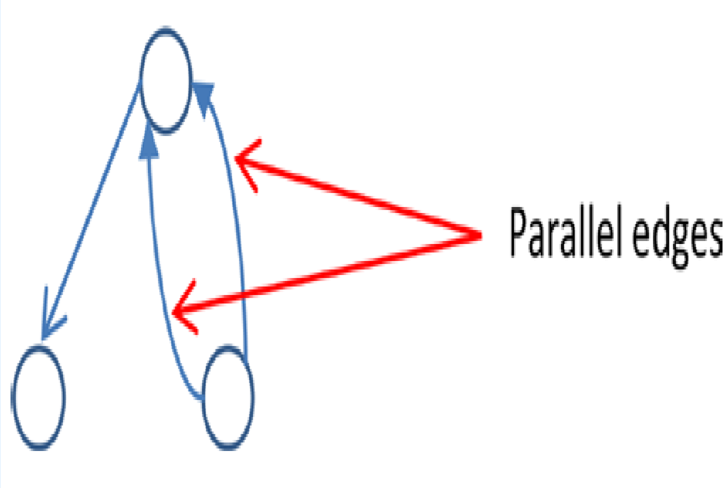


Undirected edge



Destination vertex

Source vertex



Parallel edges

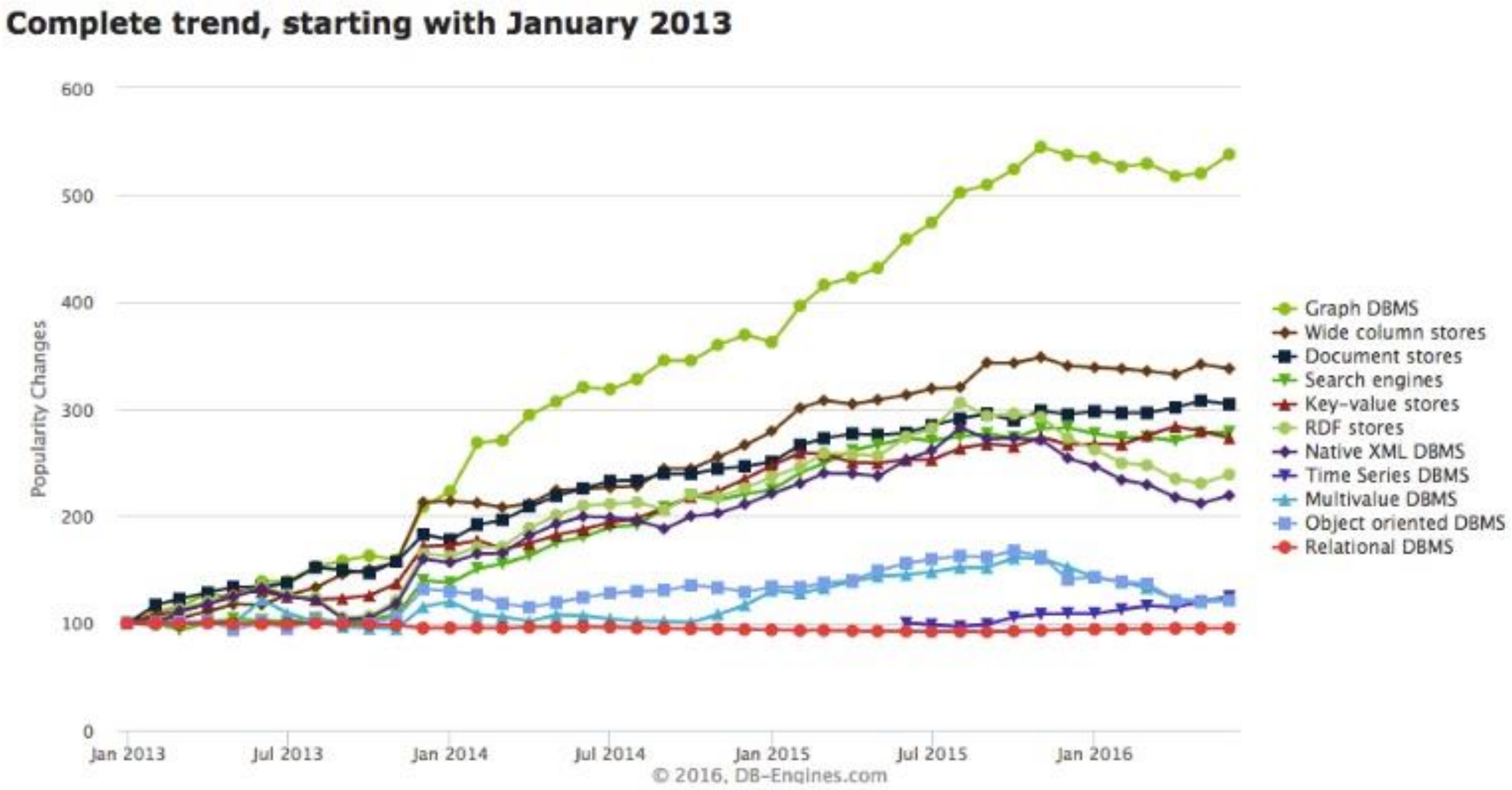
Graphs

- Hmm…seems a lot like RDBMS
- Both are highly structured
- Data generally stored as collection of rows
- But there is a huge difference
  - An edge in graph is a relationship that directly relates data items
  - It implies data is linked together directly
  - RDBMS on the other hand store links between data as tables.
  - Implies lots of joins to combine everything together
  - Makes retrieval of complex hierarchies a pain
  - Eg. Who are the most influential people in your circle?
- RDBMS not always efficient

Graphs

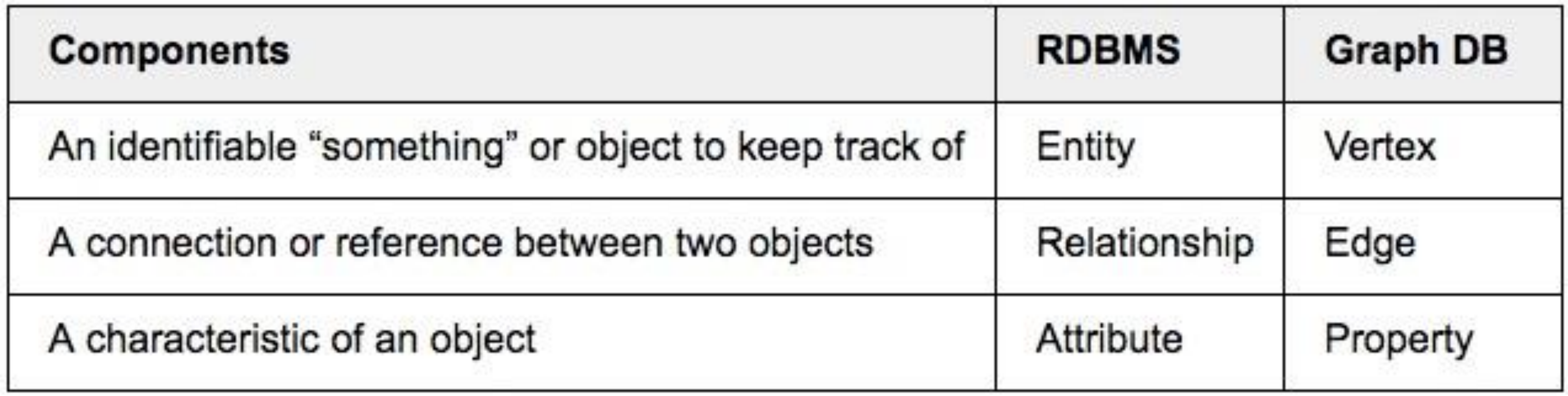
- Each node (entity or attribute) in the graph database model directly and physically contains a list of relationship-records that represent its relationships to other nodes.
- These relationship records are organized by type and direction and may hold additional attributes.
- Whenever you run the equivalent of a *JOIN* operation, the database just uses this list and has direct access to the connected nodes, eliminating the need for a expensive search / match computation.

Popularity of graph analytics



Graphs vs RDBMS

- Primary purpose of Graphs (DB and analytics) is to store and manage “highly connected and complex data”
- Efficient at handling high number of relationships between entities (data elements)



Components	RDBMS	Graph DB
An identifiable “something” or object to keep track of	Entity	Vertex
A connection or reference between two objects	Relationship	Edge
A characteristic of an object	Attribute	Property

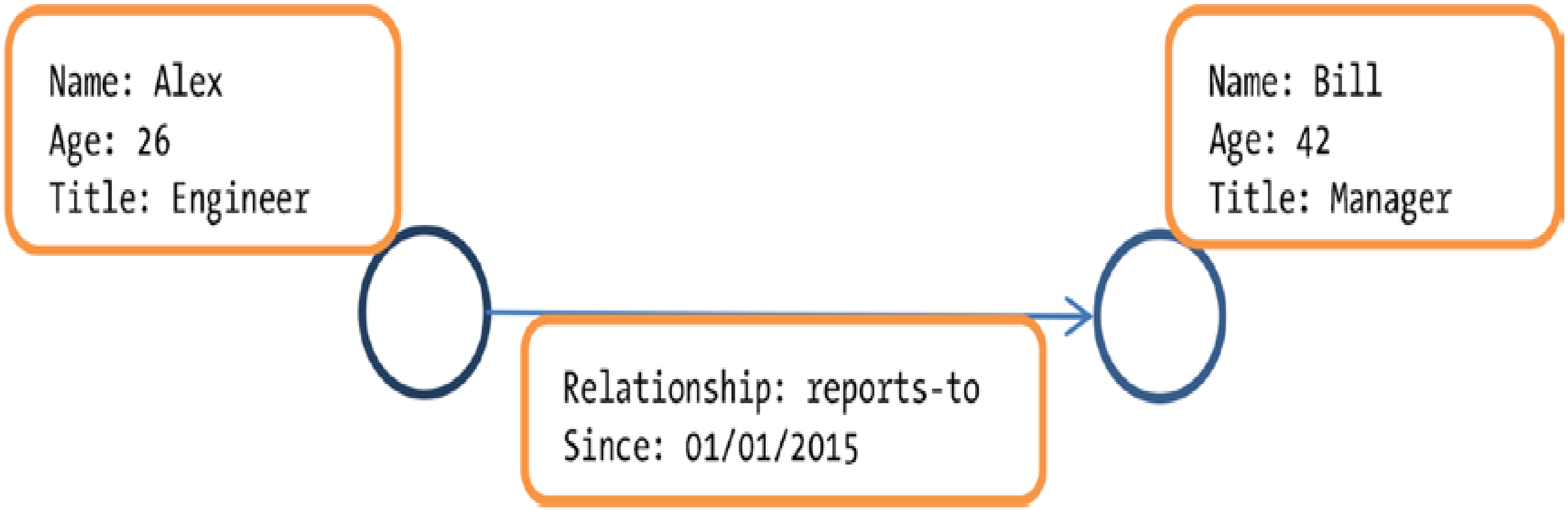
Graphs vs RDBMS

- Graphs can essentially be thought of as next generation relational databases
- Eliminated need of costly joins
    - Since relationships are now stored as edges
- As complexity of query increases, it becomes extremely costly

Example

- That movie about robots with the actor who was in that movie with that other actor that played the lead in Sivaji
- RDBMS approach:
  - Find actors in Sivaji
  - Find all movies they were in
  - Find all actors in all of those movies who were not the lead in Sivaji
  - Find all movies they were in
  - Filter the list to those descriptions containing robot
- Graph approach:
  - Walk from Sivaji to Thalaiva (Rajnikanth)
  - Links to movies he has been in
  - Links out of those movies to other actors
  - Follow back to list of movies and filter by robots
  - All of this is accomplished in one query without joins!!!

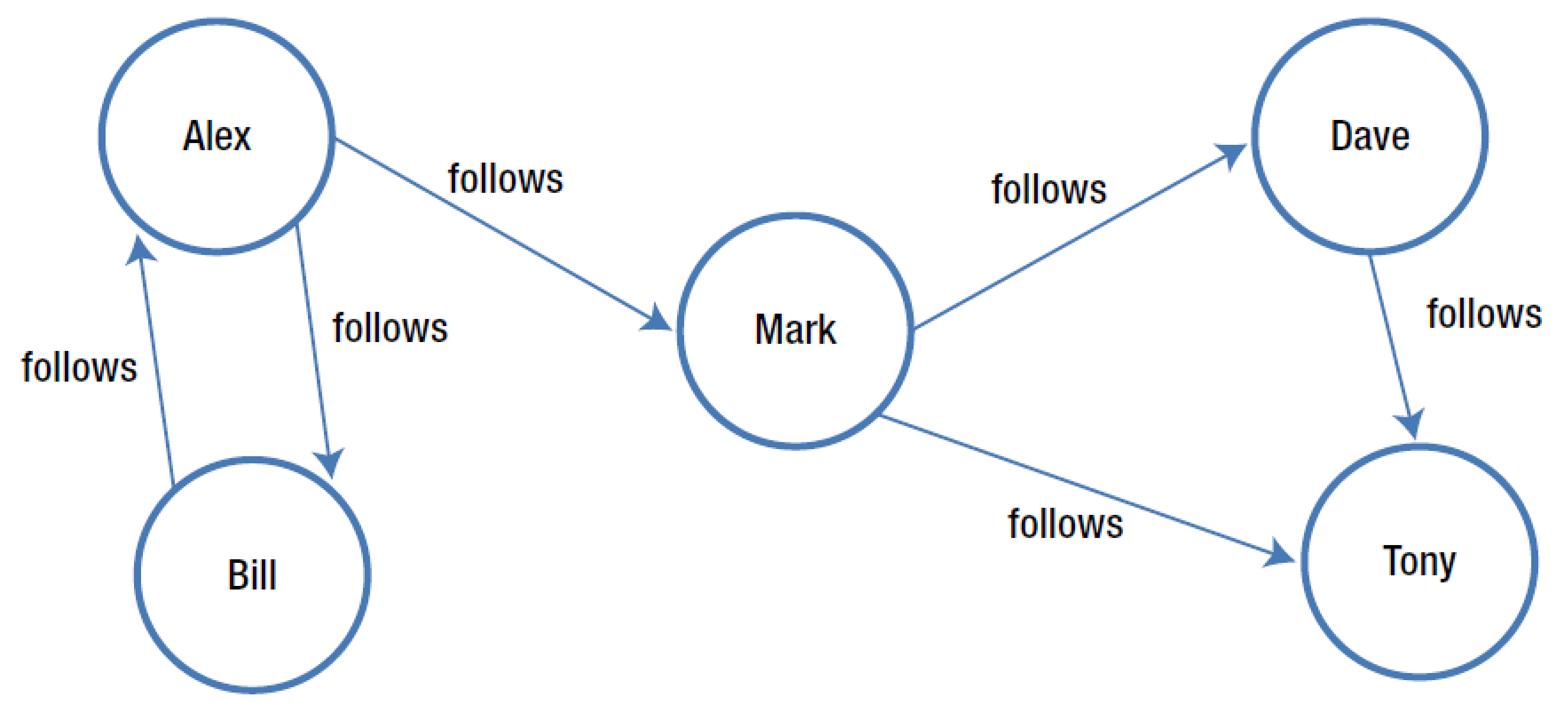
Property Graph



Name: Alex
Age: 26
Title: Engineer

Relationship: reports-to
Since: 01/01/2015

Name: Bill
Age: 42
Title: Manager


Alex
follows
Mark
follows
Dave
follows
follows
follows
Bill
follows
Tony

Another Example

- Imagine a social network similar to FB.
- A user can be friends with others, and can also like pages
- Let us structure that in tables

Another Example

User			
ID	Name	Gender	Age
1	John	M	28
2	Mary	F	26
3	Francis	F	31

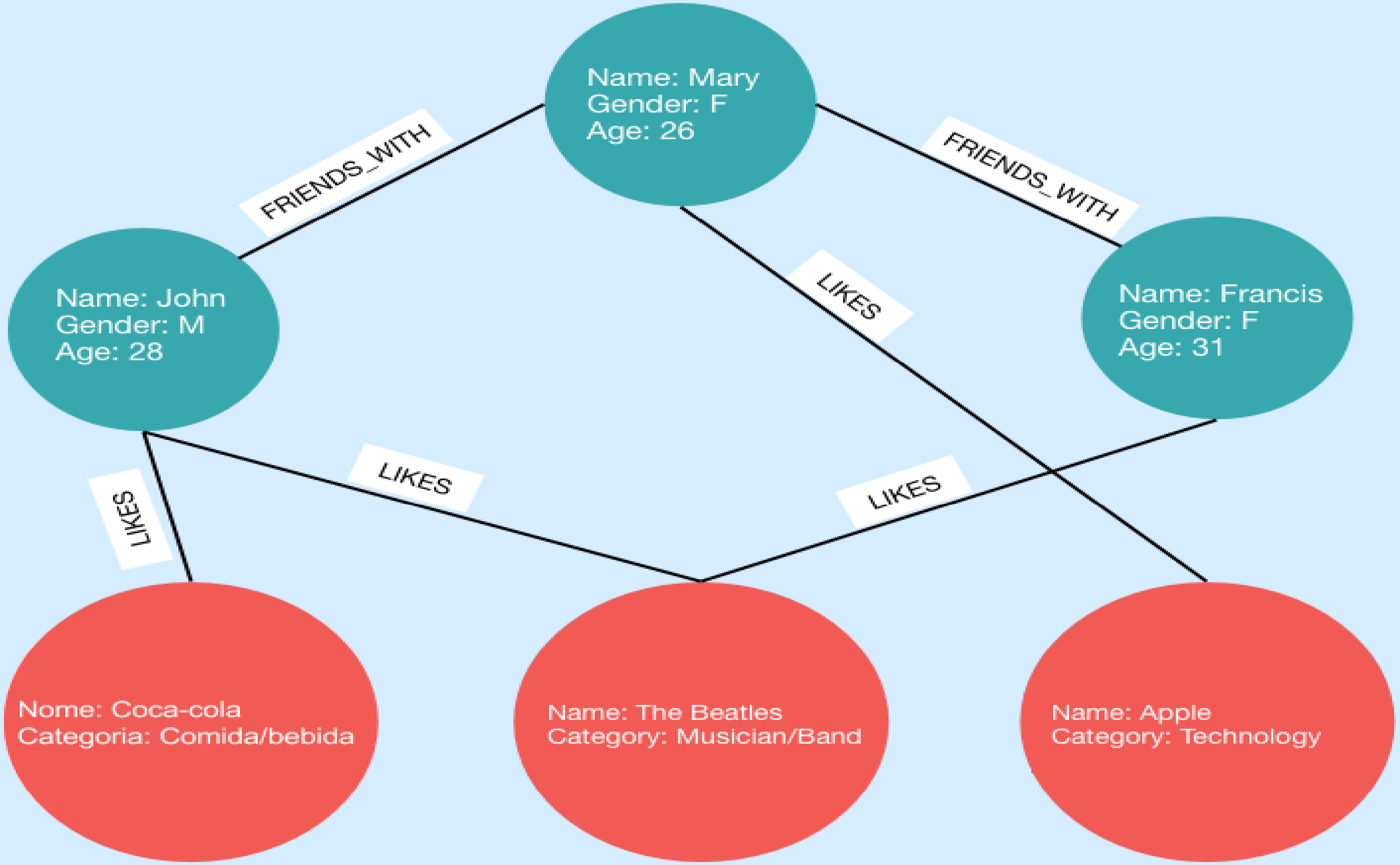
Friends_with	
ID_1	ID_2
1	2
2	3

Page		
ID	Name	Category
1	Coca-cola	Food/Beverage
2	The Beatles	Musician/Band
3	Apple	Technology

Likes	
ID_User	ID_Page
1	2
3	2
1	1
2	3

Another Example

- We can represent same info in graphs



Introduction to GraphX

- Earliest API to do graph processing in Spark
- General purpose graph processing library
- Collection of functions (also known as operators)
  - Fundamental graph operators
- Advanced Operators
  - Page rank
  - Triangle count
- Integrated platform for complete graph analytics
- Optimized for fast distributed computing

API

- Data types for representing graph oriented data
- Operators for graph analytics
- Data abstraction
  - Vertex RDD
  - Edge
  - EdgeRDD
  - EdgeTriplet
  - Graph

Data Types

- Vertex RDD
  - Distributed collection of vertices
  - One entry for each vertex
  - Vertex represented by a key-value pair
- Edge
  - Abstracts a directed edge
  - Source vertex id, destination vertex id, edge attributes
- EdgeRDD
  - Distributed collection of edges
- Graph
  - Abstraction for property graphs
  - Immutable, distributed, fault tolerant

Issues with GraphX

- No Java, Python API
  - Stuck with Scala
- RDD based API
- Doesn’t leverage more advanced Spark techniques
  - Query optimizer
  - Memory management

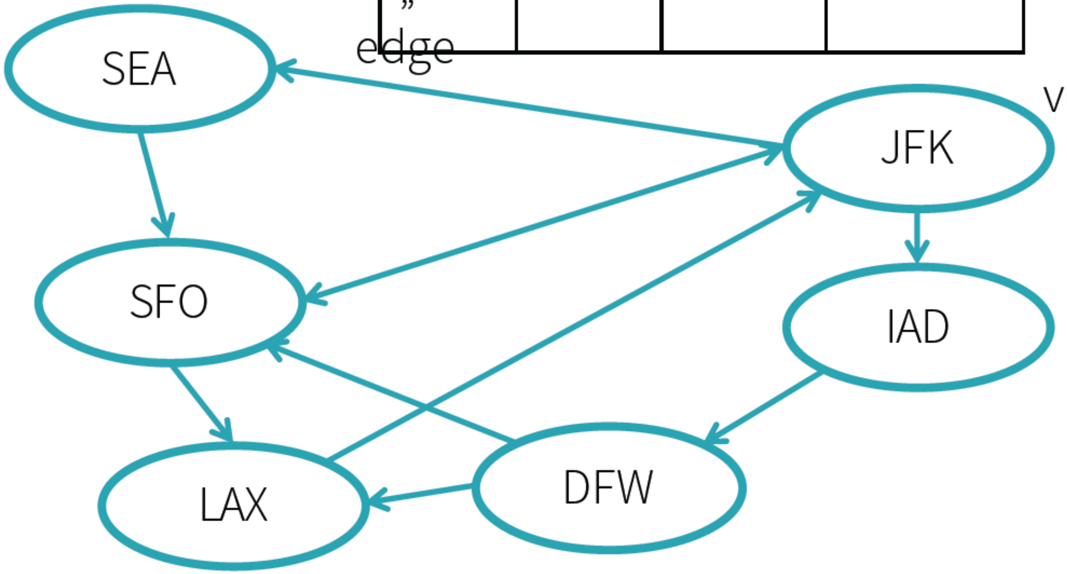
Introducing GraphFrames

- DataFrame based graphs
  - Simplify querying
  - Motif based finding (great for structural patterns)
  - Query optimizations (as now we are leveraging DataFrames)
- Remember, graphs are highly structured data

Example

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

edge



vertex

id	City	State
“JFK”	“New York”	NY

GraphFrame

“vertices” DataFrame

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

id	City	State
“JFK”	“New York”	NY
“SEA”	“Seattle”	WA

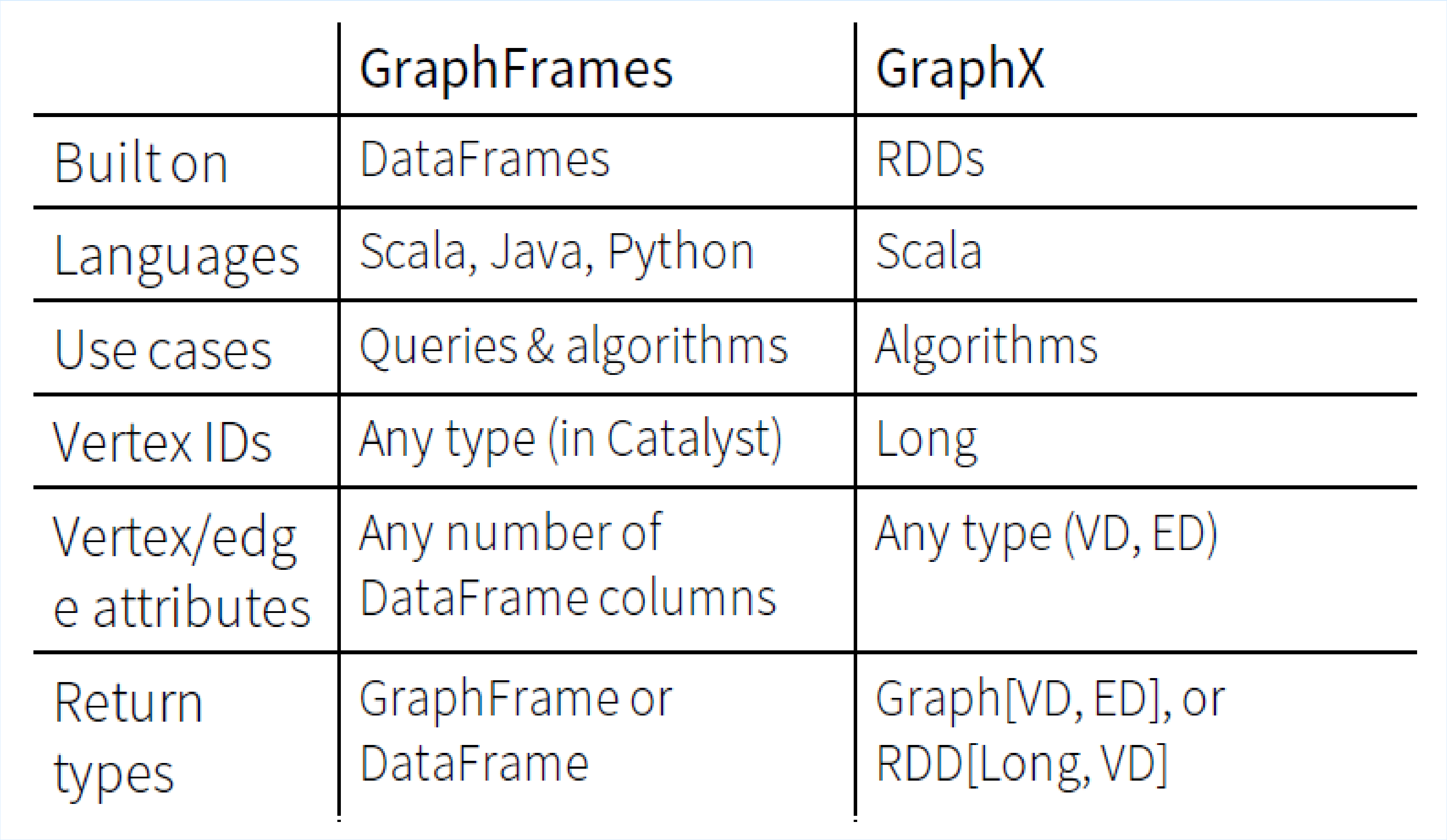
“edges” DataFrame

- 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).

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]

Graph Analytics Pipeline

- Read raw data
- Preprocess
- Extract vertices and edges and create property graph
- Slice
- Graph Algorithms
- Analyze results

Workflow using GraphFrames

- Load data as DataFrames (from a CSV file or any other type of file)
  - One DataFrame for vertices
  - One DataFrame for edges
- Import GraphFrame package
- Create GraphFrame (Graph essentially) from above two dataframes
  - Graph1 = GraphFrame(vertexDF, edgeDF)
- Voila, start querying

Querying

- Three types
  - Simple Queries
  - Motif
  - Graph Algorithms

Example

- Let us understand this with help of an example
- A handy one would be flight data
- CSV file
  - Each row contains info about a particular flight
  - Eg. Source, destination, date, departure, arrival, late?, carrier etc
- We can start asking questions here

Simple Queries

- SQL queries on vertices & edges
  - E.g., what trips are most likely to have significant delays?
  - SQL query?
- Graph queries
  - Vertex degrees
  - Edges per vertex (incoming, outgoing, total)
  - SQL query?
- Triplets
  - Join vertices and edges to get (src, edge, dst)

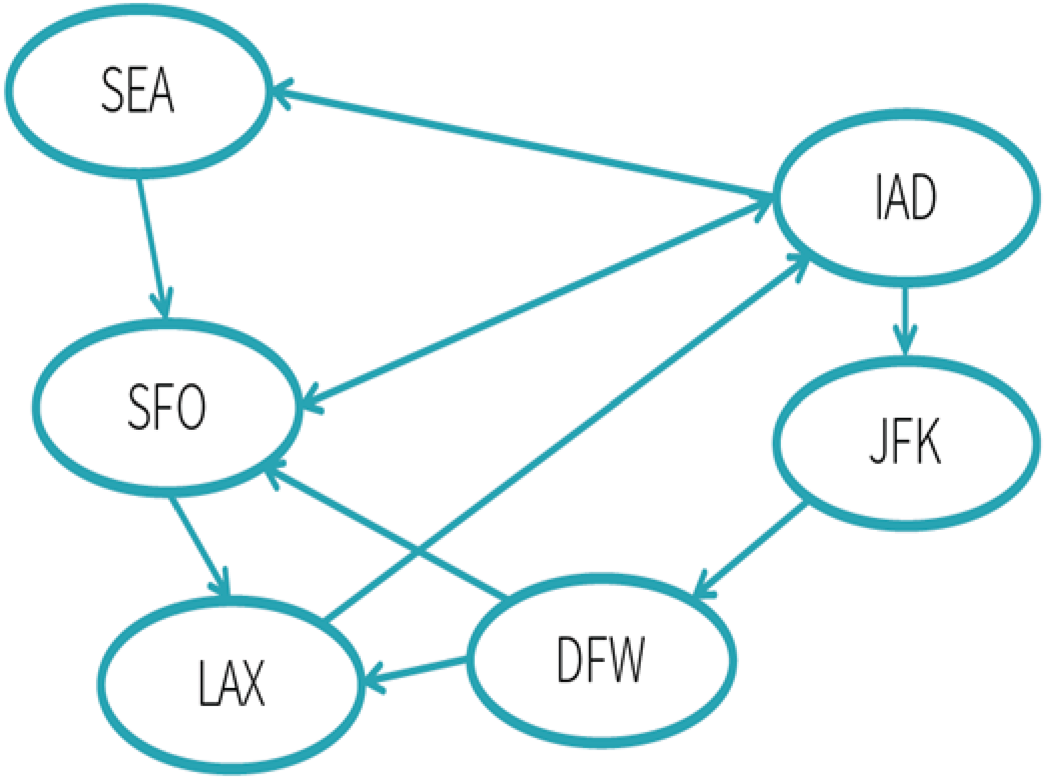
Let’s dig deeper

- How about finding all one hop flights from a carrier from Delhi to Mumbai, but there should be no direct connection between Mumbai to Delhi?
  - SQL query?
- Let us assume flights to London are very delayed. They are all one hop flights with the hop being Mumbai. Heathrow suspects that Mumbai is the culprit. How can we figure out if Mumbai is to be blamed?

Motif

Search for structural patterns within a graph.

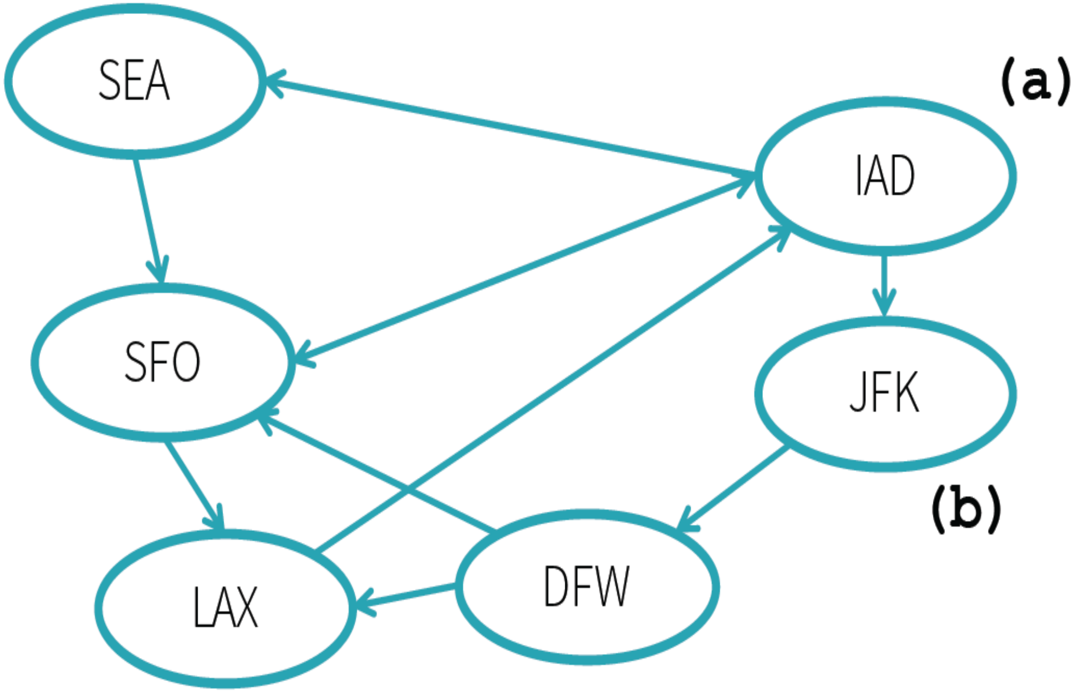
```
  paths                =
g.find("(a)-[e1]->(b);
          (b)-[e2]->(c);
         !(c)-[]->(a)")
```



Motif

Search for structural patterns within a graph.

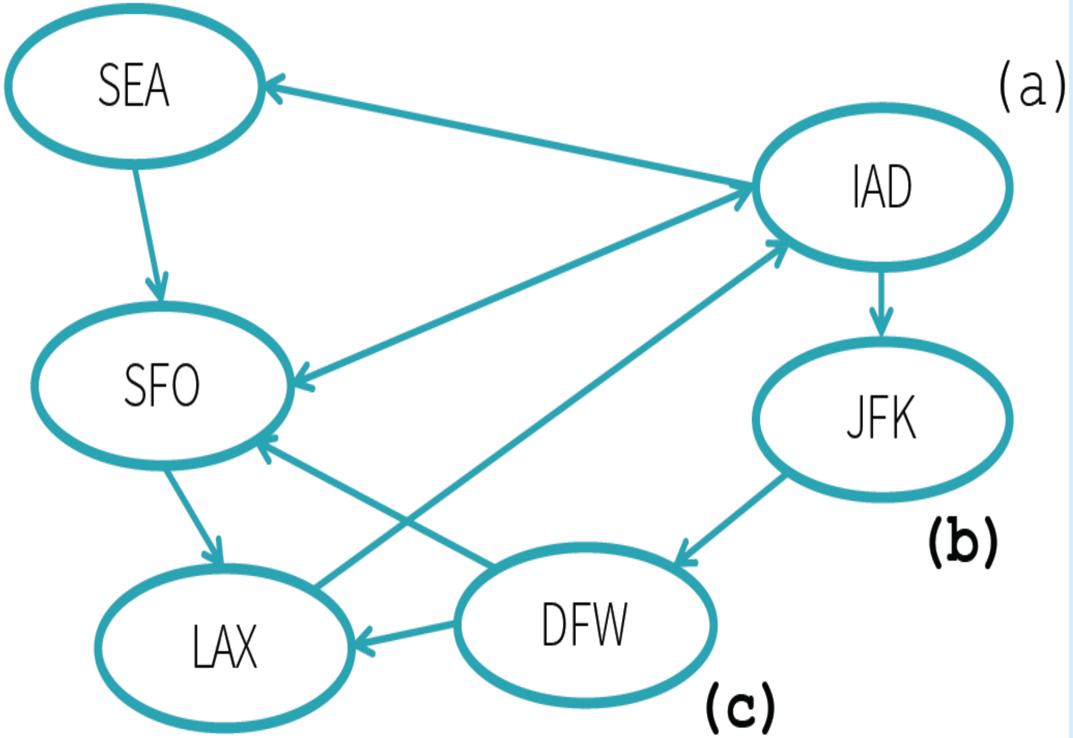
```
val paths: DataFrame =
  g.find("(a)-[e1]->(b);
          (b)-[e2]->(c);
          !(c)-[]->(a)")
```



Motif

Search for structural patterns within a graph.

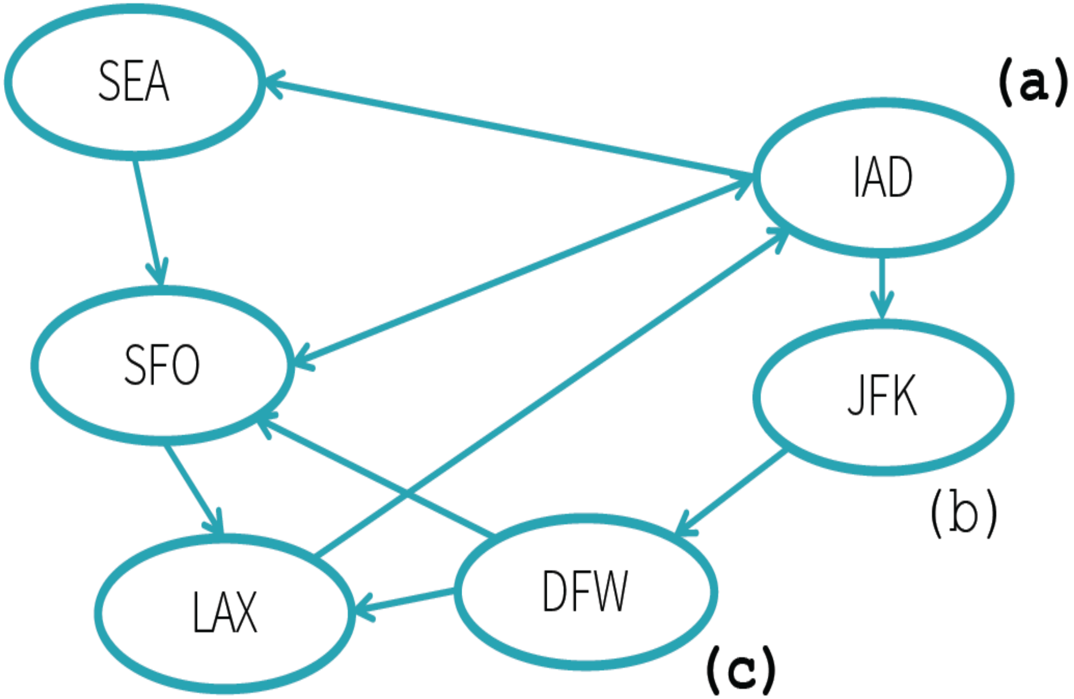
```
val paths: DataFrame =
  g.find("(a)-[e1]->(b);
            (b)-[e2]->(c);
          !(c)-[]->(a)")
```



Motif

Search for structural patterns within a graph.

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



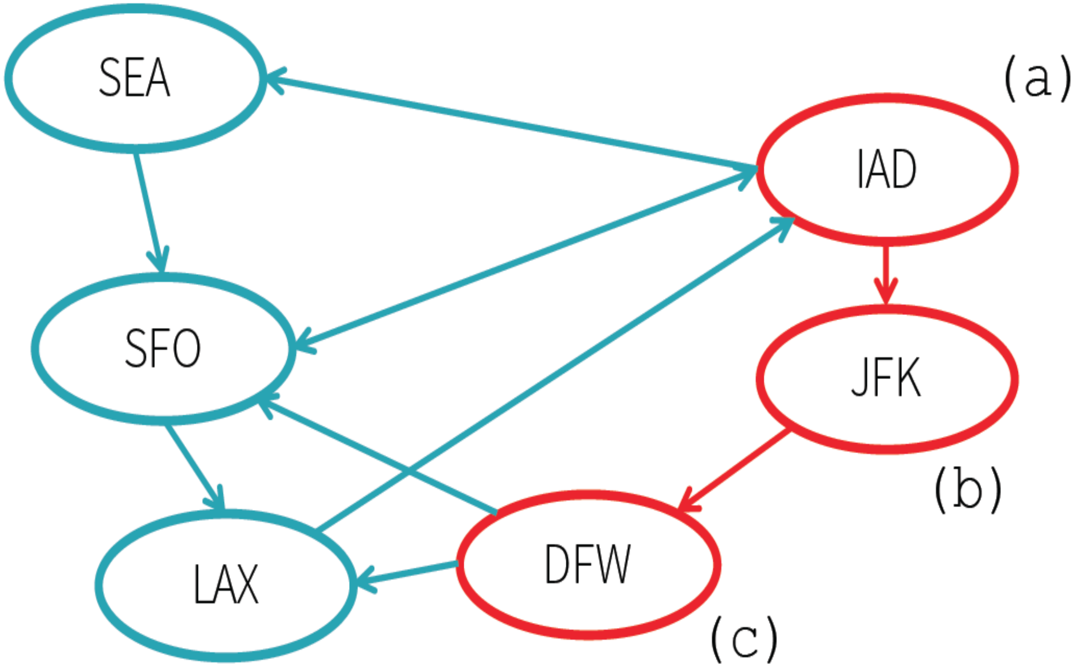
Motif

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.

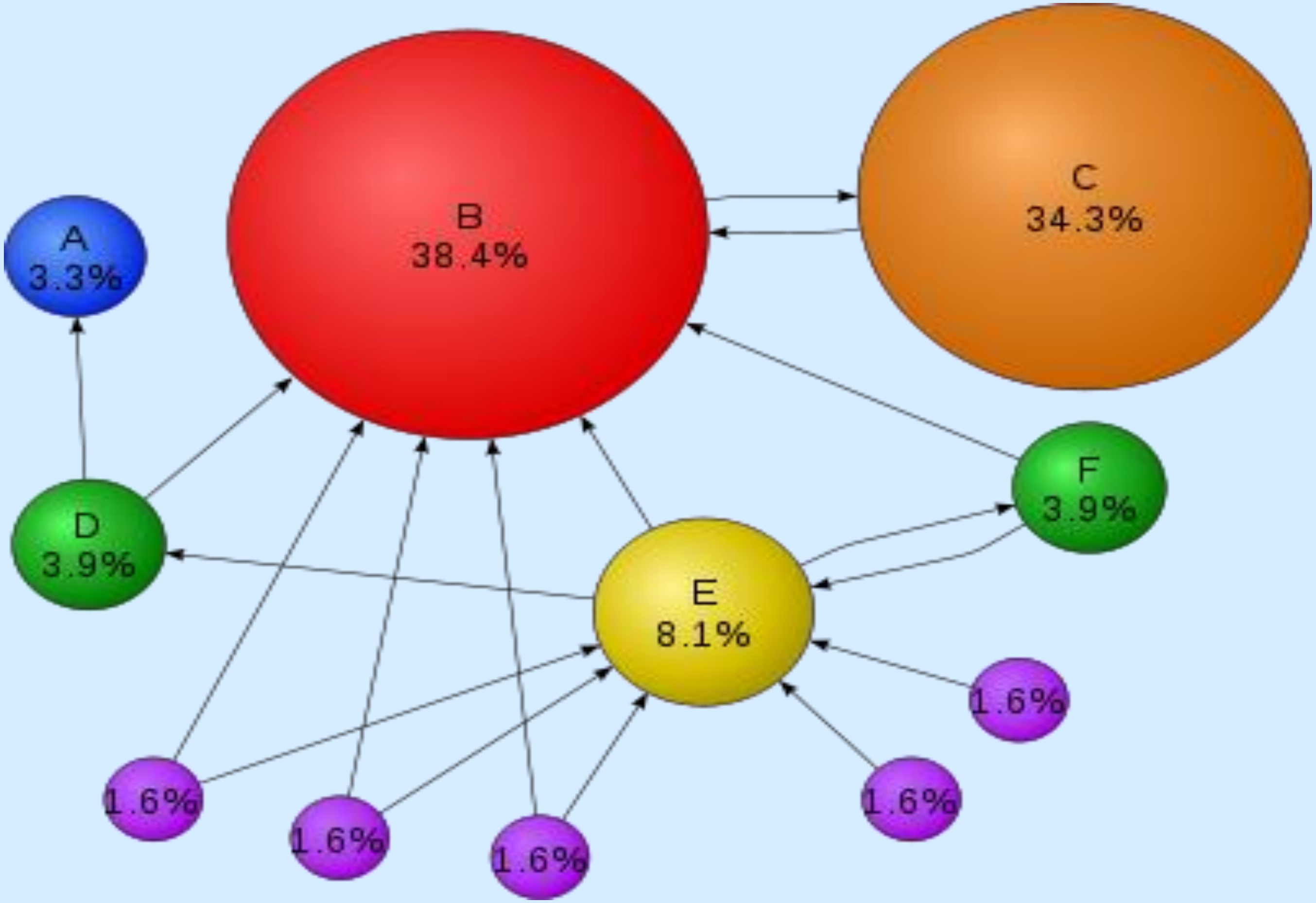
```
paths.filter(“e1.delay > 20”)
```



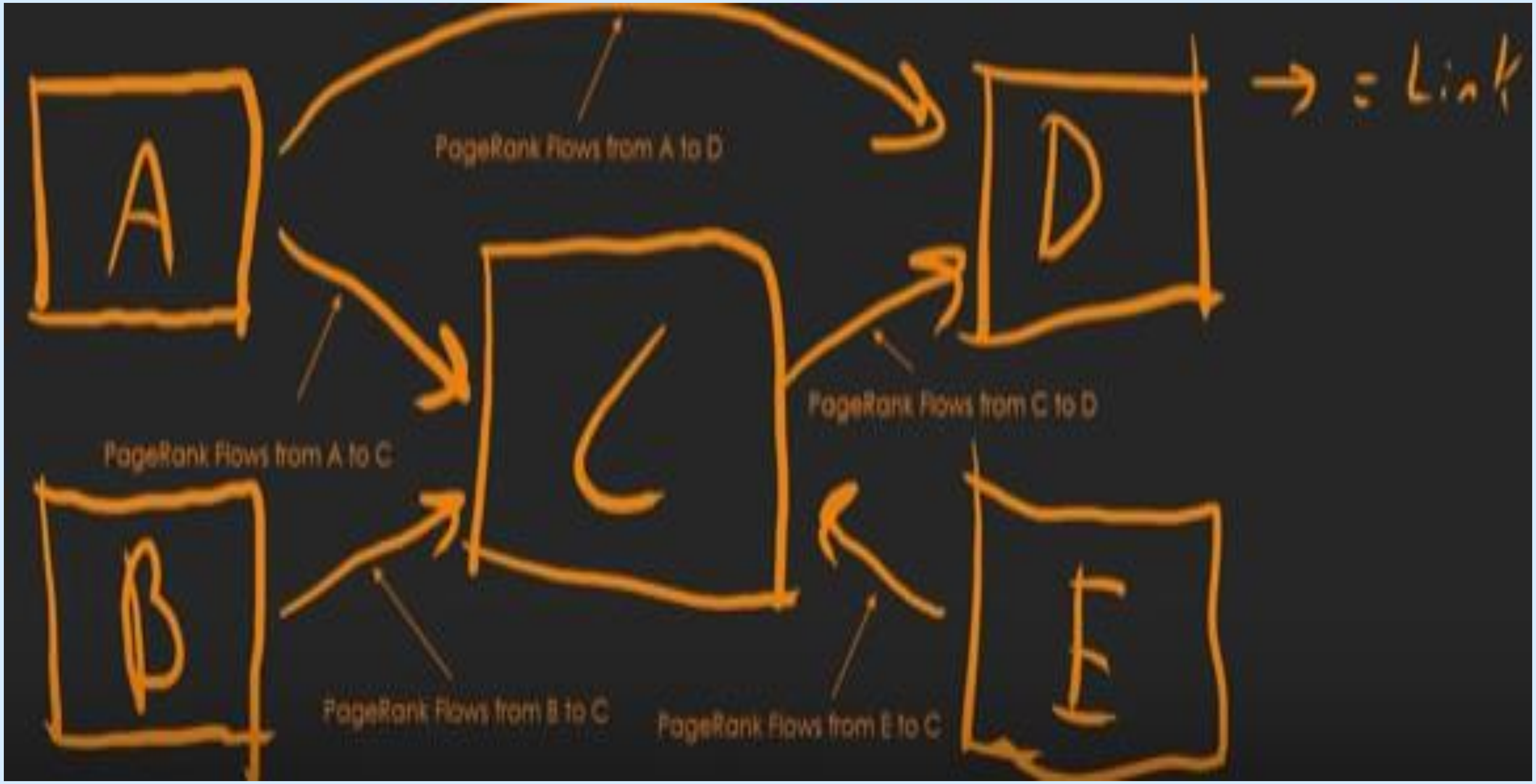
Graph Algorithms

- PageRank
  - Find important vertices (Influential connections or hubs)
- BFS, Shortest paths
  - Find paths between sets of vertices
- Label propagation
  - Communities
  - Groups of vertices
- Many others
- Algos mostly wrapper to GraphX
  - Some are implemented using DataFrames

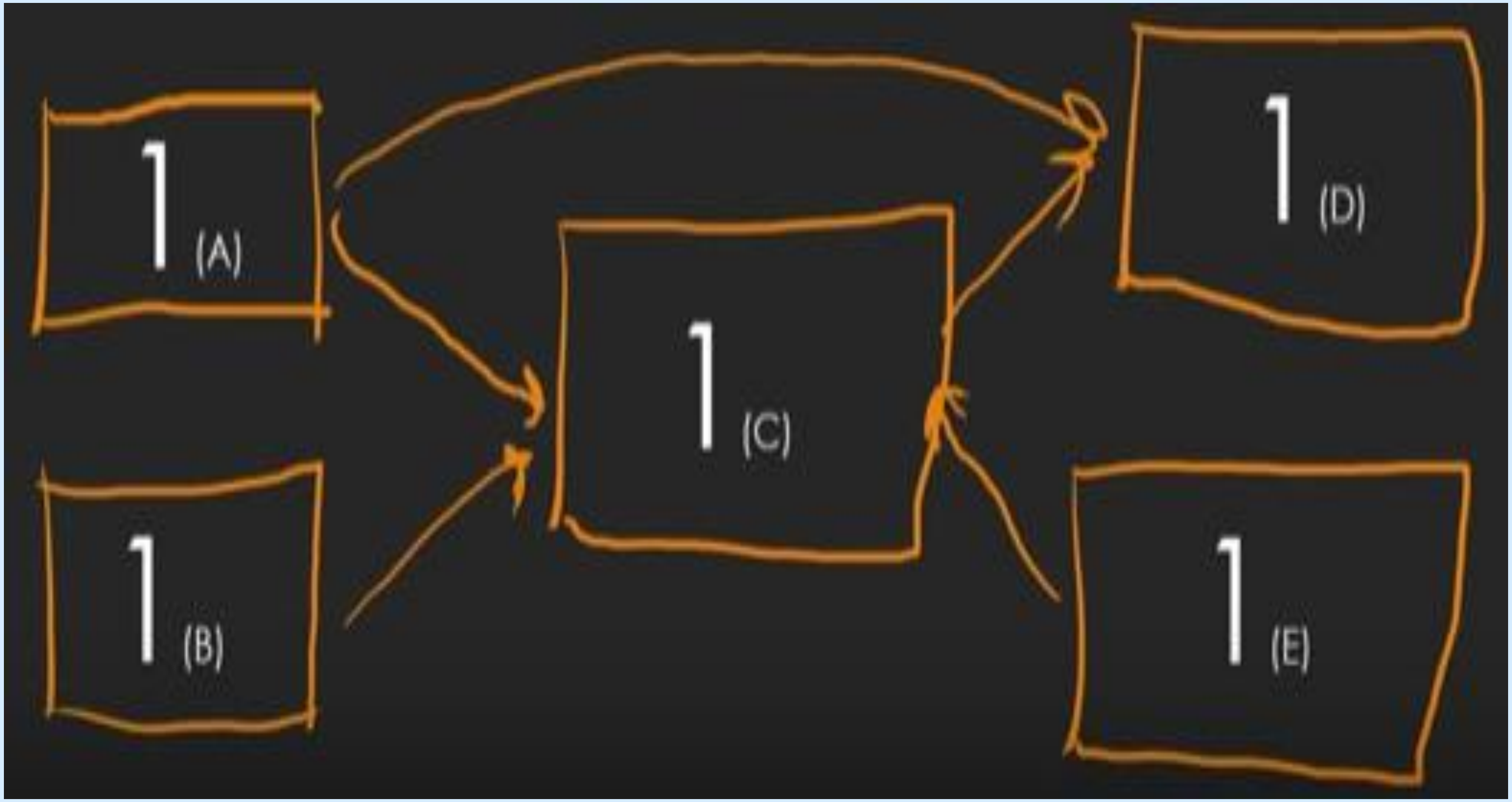
PageRank



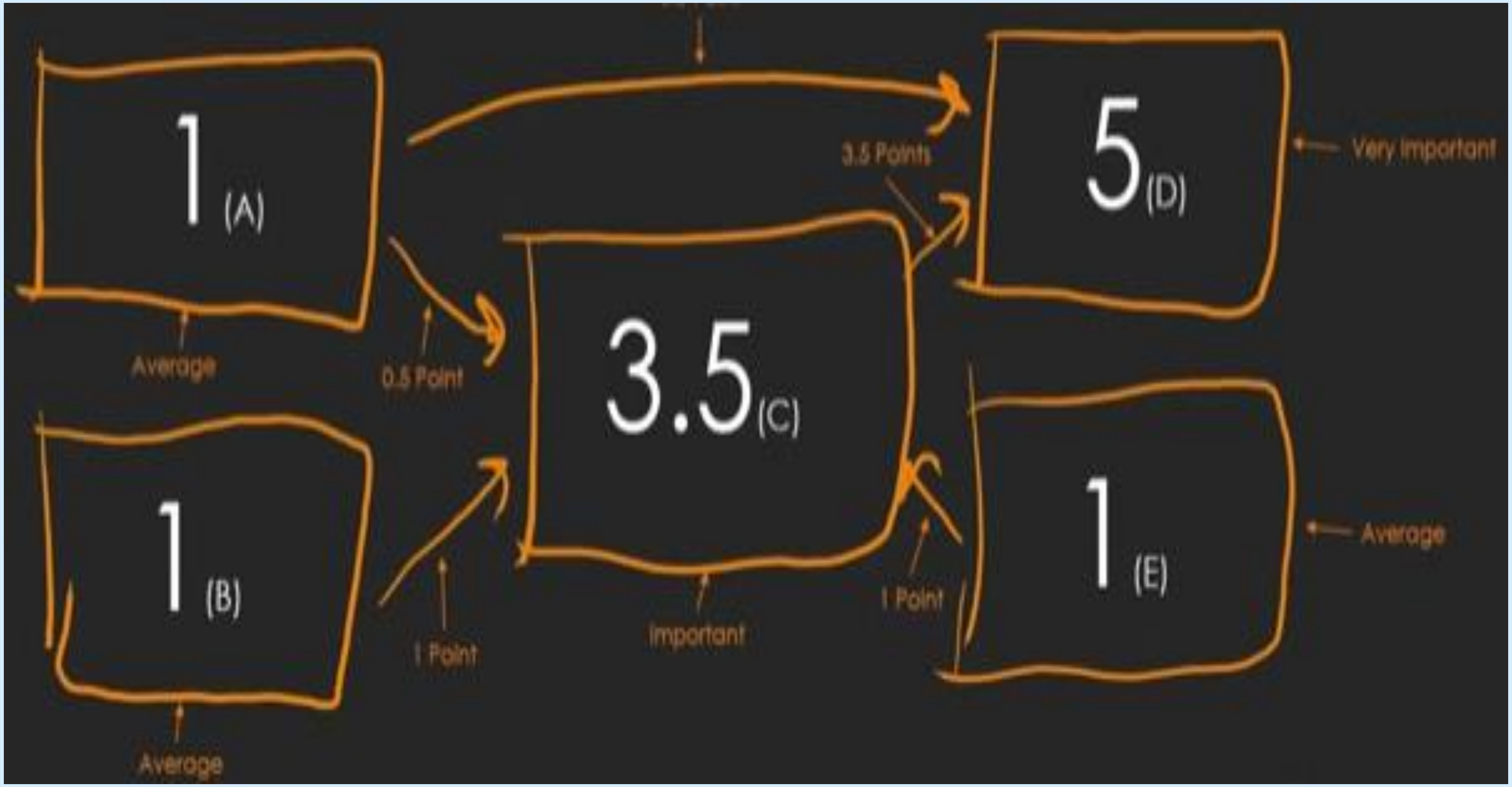
PageRank


A
B
C
D
E
→ = Link
PageRank Flows from A to D
PageRank Flows from A to C
PageRank Flows from C to D
PageRank Flows from B to C
PageRank Flows from E to C

PageRank



PageRank



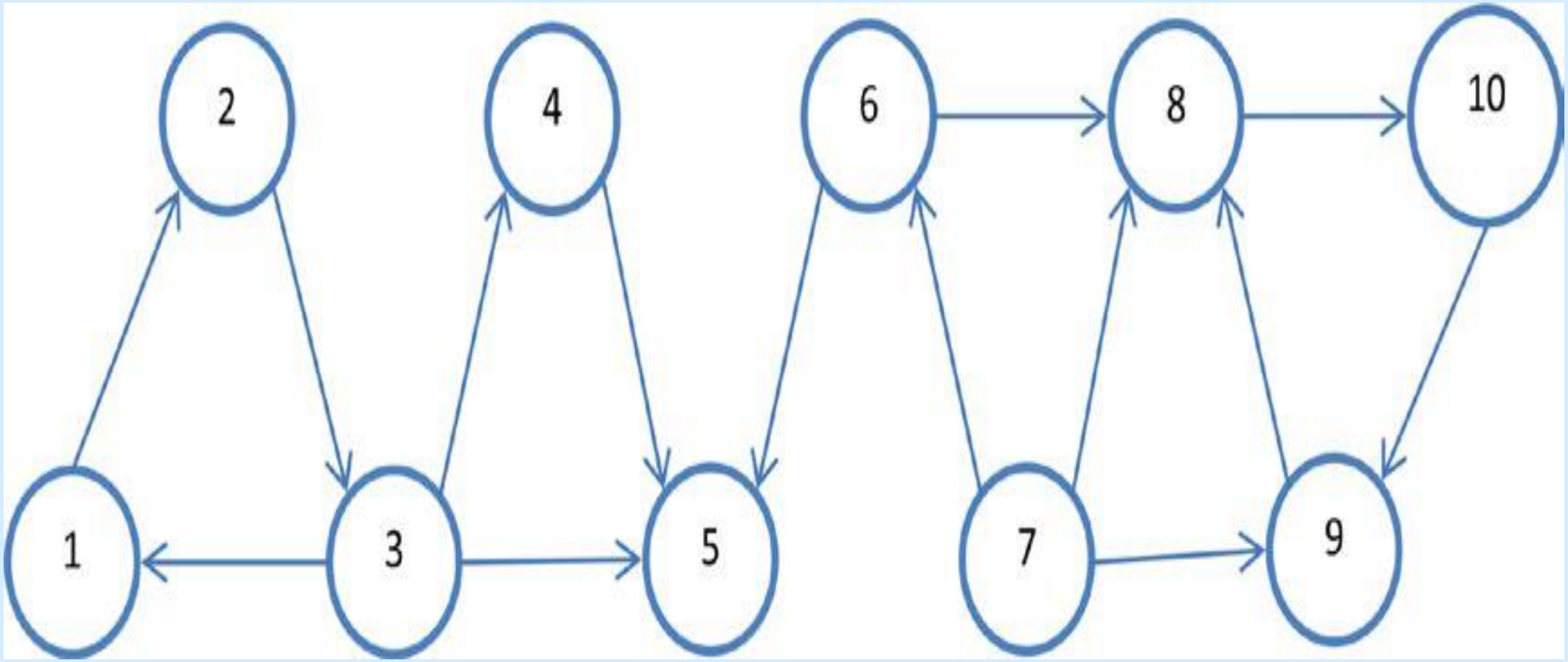
1 (A)
Average
1 (B)
Average
0.5 Point
1 Point
3.5 (C)
Important
3.5 Points
5 (D)
Very Important
1 Point
1 (E)
Average

Save and Load

- Save & load the DataFrames.
- vertices = sqlContext.read.parquet(...)
- edges = sqlContext.read.parquet(...)
- g = GraphFrame(vertices, edges)
- g.vertices.write.parquet(...)
- g.edges.write.parquet(...)

Creating a Graph

- Let us imitate social network graph
- Vertex represents user
- Directed edge represents follow relationship


2
4
6
8
10
1
3
5
7
9

Hands On

- Let us now move on to hands on
- We will first imitate a social network
- After that, we will move to an airline case study