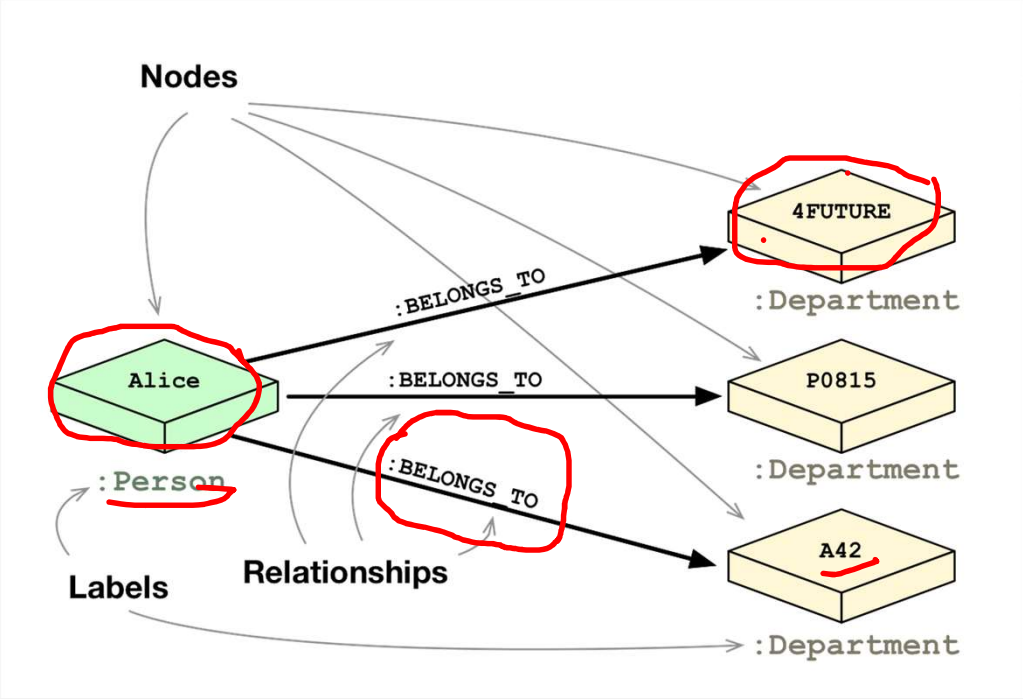
CLOUD COMPUTING APPLICATIONS

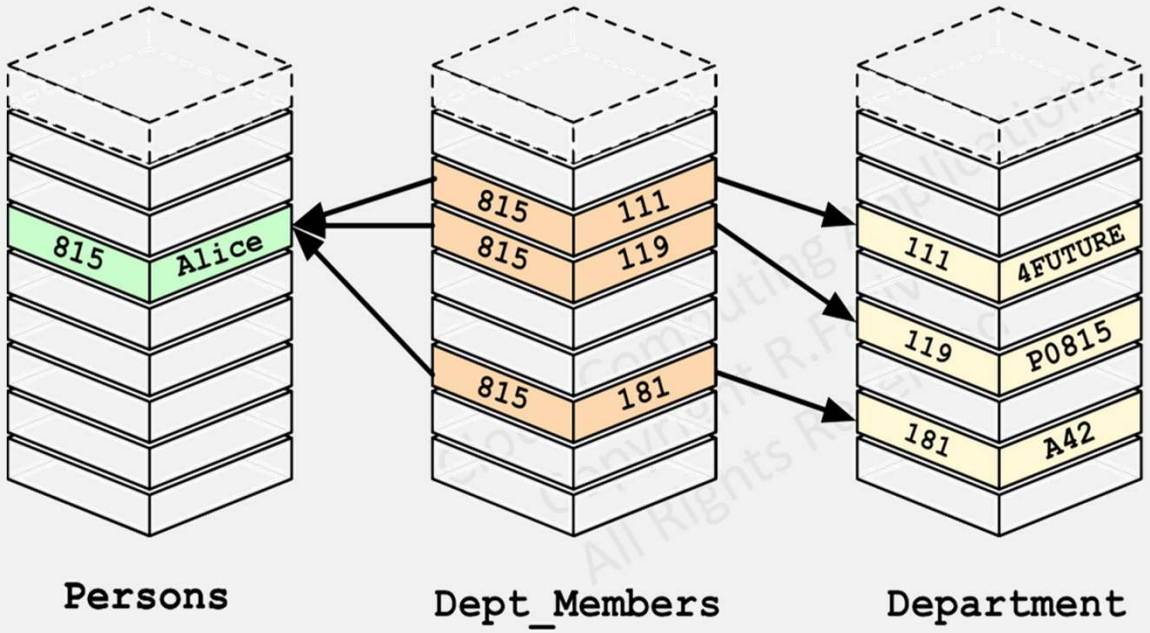
Graphs: Databases - Intro

Prof. Reza Farivar

Graph Model


Nodes
4FUTURE
:Department
:BELONGS_TO
Alice
:BELONGS_TO
P0815
:Department
:BELONGS_TO
:Person
A42
Labels
Relationships
:Department

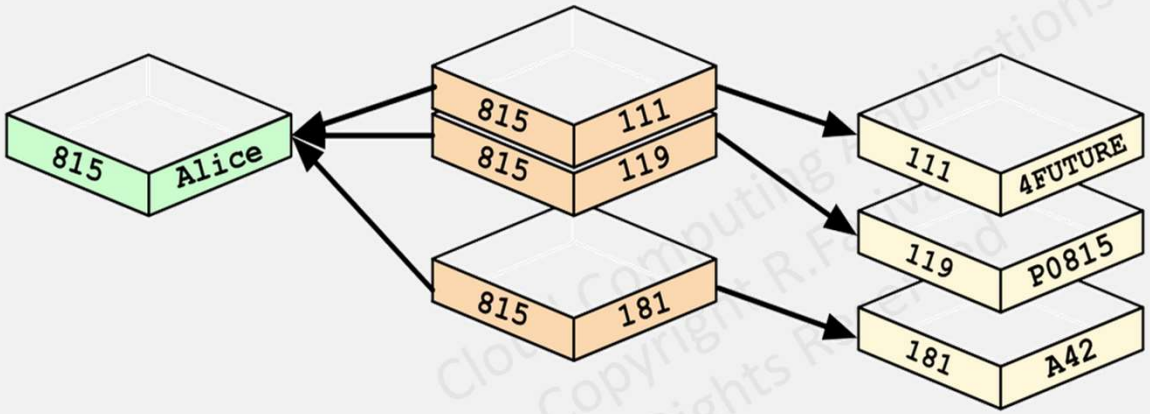
Relational Databases



Foreign Keys

Joins

Graph Databases

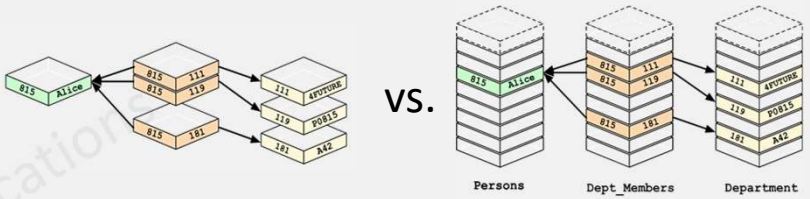


~~Foreign Keys~~

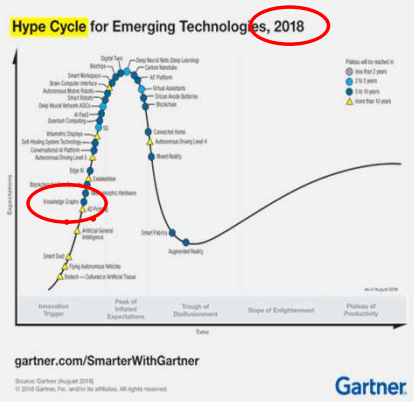
~~**Joins**~~

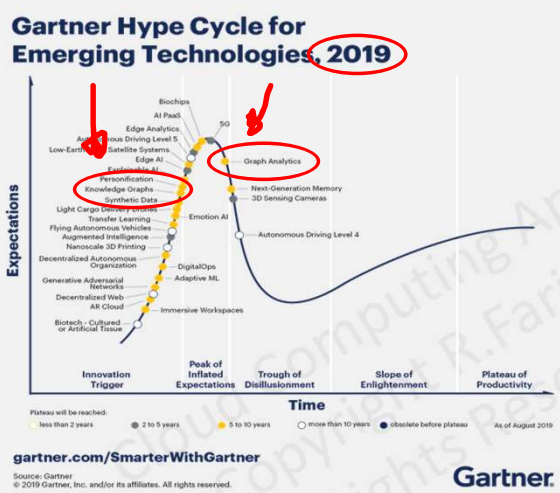
Graph and Relational Databases

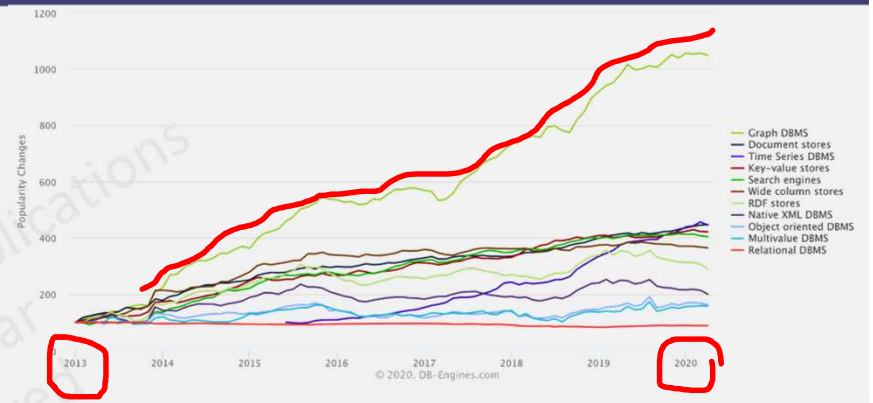
- Graph Database
  - Associative data sets
  - Structure of object-oriented applications
  - Do not require join operators
- Relational Database
  - Perform same operation on large numbers of data elements
  - Use relational model of data
  - Entity type has own table
    - Rows are instances of entity
    - Columns represent values attributed to that instance
  - Rows in one table can be related to rows in another table via unique key per row



Popularity Trend of Databases

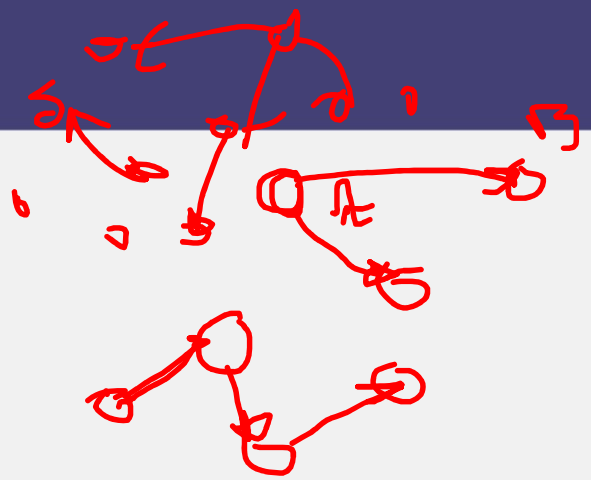






Graph Databases

- A database with an explicit graph structure
  - Each node knows its adjacent nodes
  - As the number of nodes increases, the cost of a local step (or hop) remains the same
  - Plus an Index for lookups
- Can be faster than relational, particularly for graph-type queries
  - Who is a friend of a friend?
- Scales well
- Does not require joins
- Less rigid schema permits easier evolution



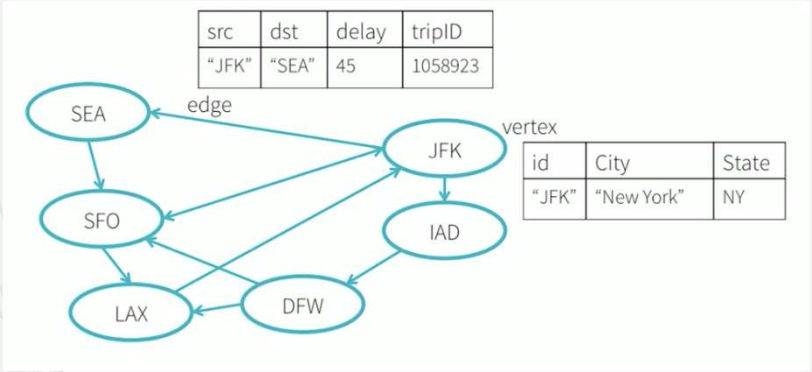
CLOUD COMPUTING APPLICATIONS

Graphs: Basics

Prof. Reza Farivar

Graphs: Overview

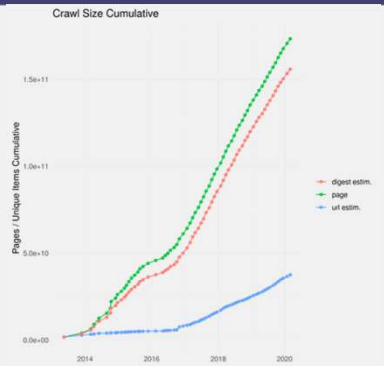
- Graph Data Structure
- Relations between vertices and edges
  - Properties: Properties are pertinent information that relate to nodes or edges
- Nodes represent entities (people, businesses, accounts…)
- Edges interconnect nodes to nodes or nodes to properties and they represent the relationship between the two
- Many domains are a natural fit



Scale of Real-World Graphs

Web Scale …

- ~2 billion vertices (websites), ~20 billion edges
  - ~10% active
  - Webpages estimated at ~60B
- 200 GB adjacency list

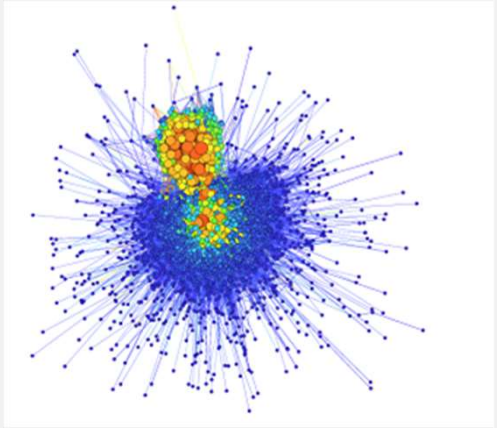


https://www.worldwidewebsize.com

Scale of Real-World Graphs

Social scale …

- 1 billion vertices, 100 billion edges
- 2.92 TB adjacency list

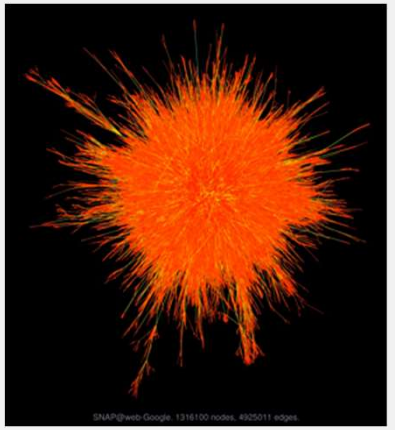


Twitter graph from Gephi data set (http://www.gephi.org)

Scale of Real-World Graphs

Web scale …

- 50 billion vertices, 1 trillion edges
- 29.5 TB adjacency list

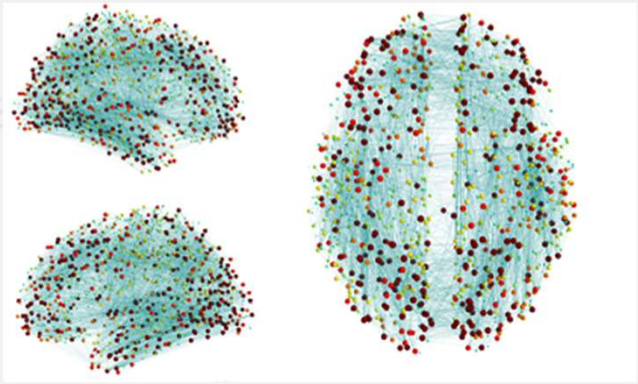


Web graph from the SNAP database (http://snap.stanford.edu/data)

Scale of Real-World Graphs

Brain scale …

- 100 billion vertices, 100 trillion edges
- 2.84 PB adjacency list



Human connectome. Gerhard et al., Frontiers in Neuroinformatics 5(3), 2011

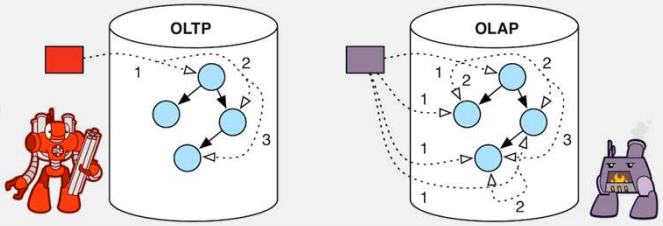
CLOUD COMPUTING APPLICATIONS

Graphs: Databases - Categories

Prof. Reza Farivar

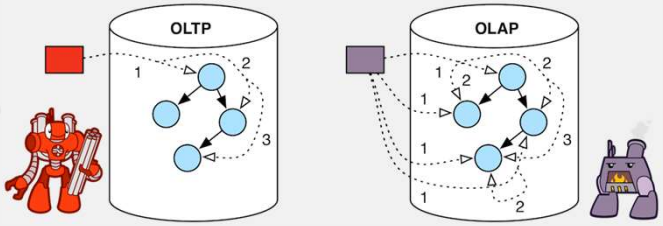
Graph Databases For Transaction Processing

- OLTP-based graph systems allow the user to query the graph in real-time
  - e.g. What soda does Sean buy?
- Typically, real-time performance only possible when a local traversal is enacted
  - A local traversal is one that starts at a particular vertex (or small set of vertices) and touches a small set of connected vertices (by any arbitrary path of arbitrary length).
  - OLTP queries interact with a limited set of data and respond on the order of milliseconds or seconds.
- Neo4J
  - Cypher language
  - graph pattern-match query language Cypher
- TinkerPop and Gremlin
- Amazon Neptune
- Azure CosmosDB



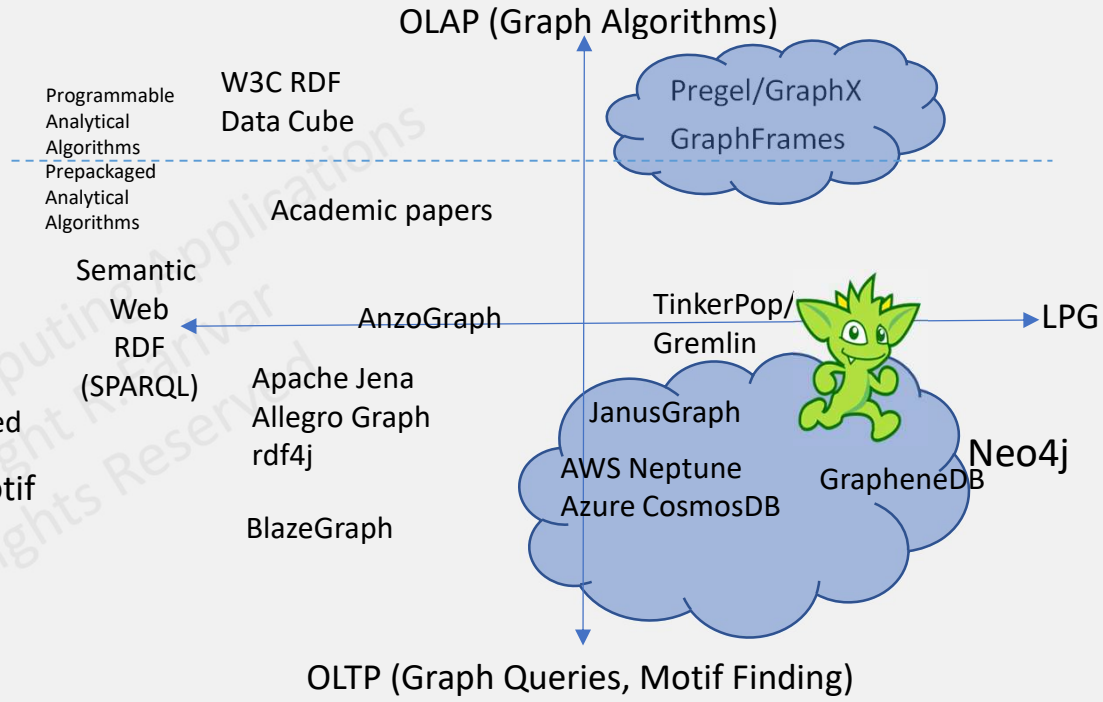
Graph Analytical Processing Systems

- With OLAP graph processing, the entire graph is processed
  - e.g. What is the average price for a soda paid by people like Sean?
  - every vertex and edge is analyzed
    - Possibly more than once for iterative, recursive algorithms
    - Standard or custom algorithms
  - Results are typically not real-time
    - Can take on the order of minutes or hours for massive graphs .
- Bulk Synchronous Parallel (BSP) programming model
  - Pregel -> Giraph -> Spark GraphX -> GraphFrames
  - Gremlin VertexProgram()



Four Graph Communities

- Four famous, yet somewhat disconnected, graph communities:
  - Semantic Web
    - RDF data model
    - SPARQL Query language
      - Declarative
  - Graph database (OLTP)
    - Labeled Property Graph data model
      - where data is organized as nodes, relationships, and properties (data stored on the nodes or relationships).
    - Pattern Matching Graph Traversal / Motif finding
      - Imperative: Gremlin
      - Declarative: Cypher
  - Big Data Graph Processing (OLAP)



CLOUD COMPUTING APPLICATIONS

Graphs: Databases - OLTP

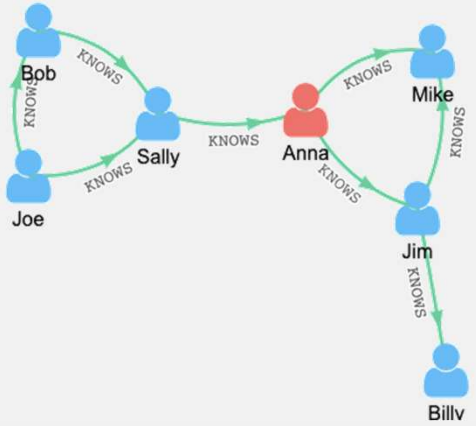
Prof. Reza Farivar

Neo4j

- Neo4j is an open-source, NoSQL, native graph database that provides an ACID-compliant transactional backend
- Cypher, a declarative query language similar to SQL, but optimized for graphs
- Morpheus: Cypher for Apache Spark

Cypher

openCypher



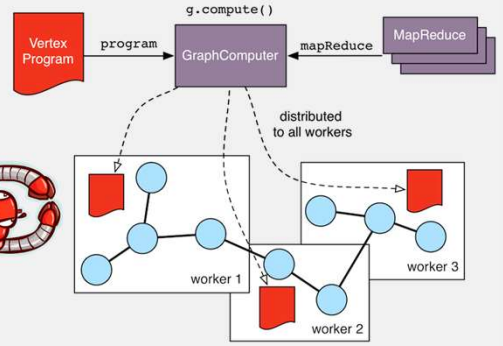
- Nodes (vertexes) are surrounded by parenthesis: () or {p}
- Labels start with : and group the node by roles or types
  - (p:Person:Mammal)
- Nodes can have properties
  - (p:Person {name: 'Jim'})
- Relationships (edges) are wrapped with square brackets and can have properties
  - --> or -[h:HIRED]->
- Direction of relationship is specified with < >

```
MATCH
        (person:Person)-[:KNOWS]-(friend:Person)-[:KNOWS]- (foaf:Person)
WHERE
        person.name = "Joe"
        AND NOT (person)-[:KNOWS]-(foaf)
RETURN
        foaf
```

- OpenCypher is widely used
  - Neo4J, SAP HANA, AnzoGraph, Cypher for Apache Spark (CAPS), Redisgraph, Cypher for Gremlin, …

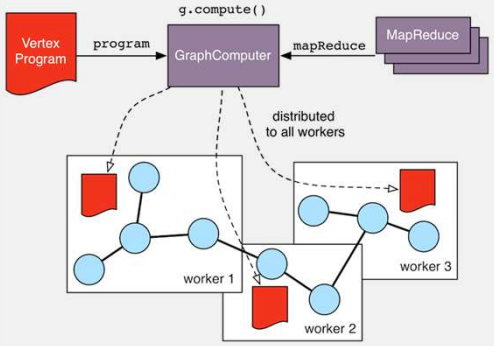
TinkerPop & Gremlin

- Supports both OLTP queries and OLAP algorithms
- TinkerPop 3 Components, altogether known as Gremlin
  - **Blueprints** → Gremlin Structure API
  - **Pipes** → GraphTraversal
  - **Frames** → Traversal
  - **Furnace** → GraphComputer and VertexProgram
  - **Rexster** → GremlinServer
- A *traversal* in Gremlin is a series of chained steps. It starts at a vertex (or edge). It walks the graph by following the outgoing edges of each vertex and then the outgoing edges of those vertices.



TinkerPop & Gremlin

- BSP type graph algorithms, similar to Pregel model, via VertexProgram
  - executed at each vertex in logically parallel manner until some termination condition is met (e.g. a number of iterations have occurred, no more data is changing in the graph, etc.)
  - The vertices are able to communicate with one another via messages
    - MessageScope.Local and MessageScope.Global
- At the core of TinkerPop3 is a Java8 API. The implementation of this core API is all that is required of a vendor wishing to provide a TinkerPop3-enabled graph engine.
- Gremlin is widely used
  - Amazon Neptune, Azure CosmosDB, DataStax, JanusGraph, OrientDB, …



GraphSON

- The Gremlin standard format for representing vertices, edges, and properties (single and multi-valued properties) using JSON
- Example: a vertex representation


```
{
  "id": "a7111ba7-0ea1-43c9-b6b2-efc5e3aea4c0",
  "label": "person",
  "type": "vertex",
  "outE": {
    "knows": [
      {
        "id": "3ee53a60-c561-4c5e-9a9f-9c7924bc9aef",
        "inV": "04779300-1c8e-489d-9493-50fd1325a658"
      },
      {
        "id": "21984248-ee9e-43a8-a7f6-30642bc14609",
        "inV": "a8e3e741-2ef7-4c01-b7c8-199f8e43e3bc"
      }
    ]
  },
  "properties": {
    "firstName": [
      {
        "value": "Thomas"
      }
    ],
    "lastName": [
      {
        "value": "Andersen"
      }
    ],
    "age": [
      {
        "value": 45
      }
    ]
  }
}
```

Gremlin graph traversal language (Imperative)

- **functional language**
- traversal operators are chained together to form path-like expressions
  - For example, "from Hercules, traverse to his father and then his father's father and return the grandfather's name."

```
gremlin> g.V().has('name', 'hercules').out('father').out('father').values('name')

==>Saturn
```

- Find all movies that John Doe appeared in
  - `gremlin> g.V().has('name','John Doe').out().values()`
- Find all relationships from a given actor/actress to John Doe (the six degrees)
  - `gremlin> g.V().has('name','John Doe').repeat(out().in().simplePath()) .until(has('name','Jane Doe')).path().by('name').by('title').limit(10)`

Gremlin Declarative vs. Imperative traversal

Declarative

Give this point in traversal a name

```
g.V().match(
  as("a").has("name","gremlin"),
  as("a").out("created").as("b"),
  as("b").in("created").as("c"),
  as("c").in("manages").as("d"),
    where("a",neq("c"))).
  select("d").
  groupCount().by("name")
```

Imperative

```
g.V().has("name","gremlin").as("a").
  out("created").in("created").
    where(neq("a")).
  in("manages").
  groupCount().by("name")
```

What are the names of the projects created by two friends?

- ...there exists some "a" who knows "b".
- ...there exists some "a" who created "c".
- ...there exists some "b" who created "c".
- ...there exists some "c" created by 2 people.
- Get the name of all matching "c" projects.

- first places a traverser at the vertex denoting Gremlin
- That traverser then splits itself across all of Gremlin's collaborators that are not Gremlin himself
- Next, the traversers walk to the managers of those collaborators to ultimately be grouped into a manager name count distribution.

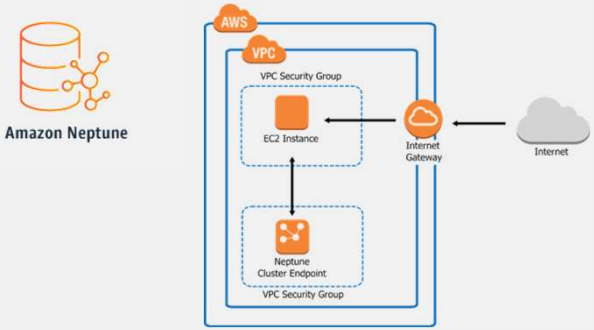
Cypher vs. Gremlin

- Gremlin has both imperative and declarative
  - At the core, Gremlin is a Groovy wrapper over underlying Java functions
  - In the imperative model, there is less room for abstraction, optimization and remoting, since you are effectively sending Groovy or Java code over the wire
  - You *can* write declarative statements
- Cypher is a declarative, non-turing complete language
  - Cypher is similar to SPARQL or SQL - a declarative, higher order description of WHAT you want, not HOW you want your results to be found
  - Well suited for remoting, standardization and optimization
- Interestingly, there is now a feature on Cypher to run on Gremlin

AWS Neptune

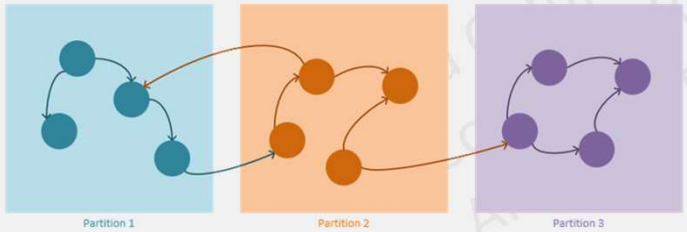
- OLTP
  - Property graph + TinkerPop Gremlin
  - RDF + SPARQL
  - each Neptune instance provides both a Gremlin Websocket Server and a SPARQL 1.1 Protocol REST endpoint
- Highly available, with read replicas, point-in-time recovery, continuous backup to Amazon S3, and replication across Availability Zones
  - replicates six copies of the data across three Availability Zones
  - instance failover typically takes less than 30 seconds.
- ACID Compliant
- HTTPS encrypted client connections and encryption at rest
- fully managed

- *Possibly based on Blazegraph project?* ☺

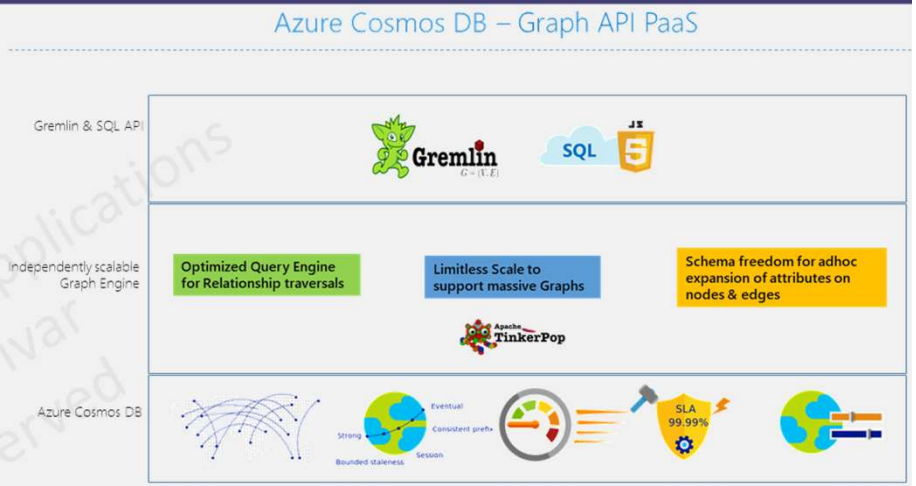


Azure CosmosDB

- Gremlin API for queries
- Supports horizontally scalable graph databases
  - Graph partitioning
    - Vertices require a partition key
    - Edges will be stored with their source vertex
    - Edges contain references to the vertices they point to
    - Graph queries need to specify a partition key
      - g.V('vertex_id').has('partitionKey', 'partitionKey_value')



- Returns the results in GraphSON format



CLOUD COMPUTING APPLICATIONS

Graphs: Semantic Web

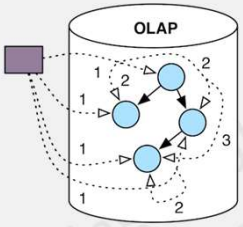
Prof. Reza Farivar

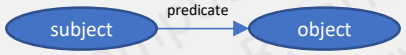
Semantic web

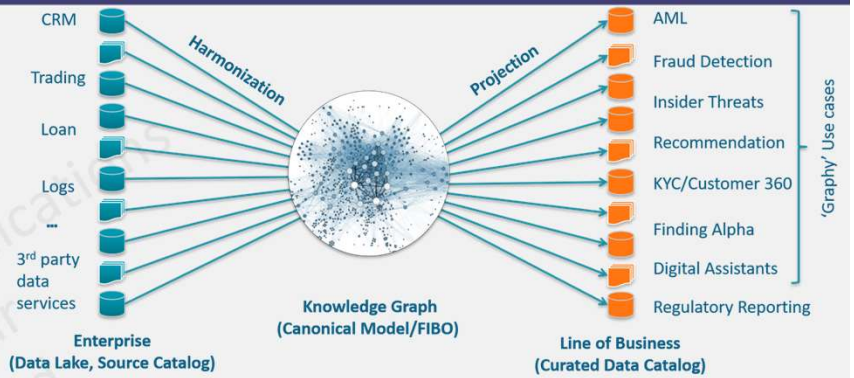
- Link explicit “data” on the world wide web in a machine readable format
  - Targeted semantic search
  - Automated agents
  - Fraud Detection
- Technologies
  - Data Model: RDF, RDF*
    - Collection of triples
    - RDF is the model, syntaxes vary: RDF/XML, Turtle, JSON-LD, etc.
      - **Terse RDF Triple Language** (**Turtle**)
        - Subject Predicate Object
  - Query language: SPARQL, GQL
  - Ontology language: OWL

<http://example.org/tea.owl> **rdf:type owl:Ontology** .

:Tea rdf:type owl:Class .

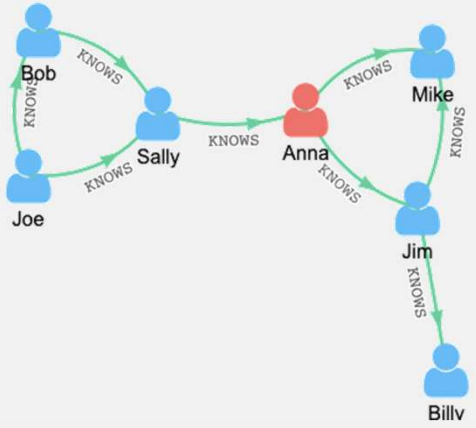






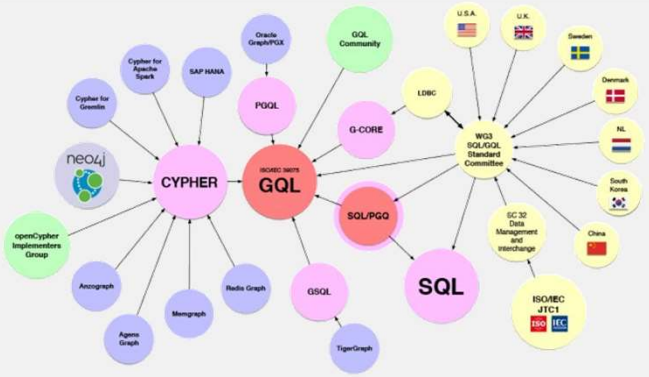
SPARQL

```
PREFIX foaf: http://xmlns.com/foaf/0.1/
SELECT   ?name
         ?email
WHERE
  {
       ?person      a            foaf:Person .
       ?person      foaf:name    ?name .
       ?person      foaf:mbox    ?email .
  }
```



Graph Query Languages

- RDF
  - SPARQL → Query
- Property Graph
  - Cypher → Query
    - Neo4j
    - Supported by SAP HANA
  - Gremlin → Traversal
    - Apache TinkerPop
      - DataStax
  - Apache Spark Graphframes .find() → Query
  - GQL: Voted as a new standard in 2019 → Query
    - Rooted in Cypher and Oracle's PGQL
- Competition is fierce!
  - There may be more than one winner



CLOUD COMPUTING APPLICATIONS

Graphs: OLAP - GraphFrames

Prof. Reza Farivar

Spark GraphFrames

- Spark 1.3 onwards move away from RDD and more towards DataFrames
- More user friendly, simple
  - GraphX uses lower-level RDD-based API (vs. DataFrames)
- Simplified API
  - Python interface
  - DataFrames
    - Can benefit from DataFrame optimizations
- Support motif finding for structural pattern searches
- Easier to do optimization under the hood

Comparison of Spark GraphX and GraphFrames

	GraphFrames	GraphX
Core APIs	Scala, Python (Java?)	Scala only
Programming Abstraction	DataFrames	RDDs
Use Cases	Algorithms, Queries, Motif Finding	Algorithms
VertexIds	Any type (in Catalyst)	Long
Vertex/edge attributes	Any number of DataFrame columns	Any type (VD,ED)
Return Types	GraphFrames/DataFrames	Graph [VD,ED] or RDD[Long, VD]

GraphX compatibility

- Easy to convert between GraphX and graphFrame

- GraphFrame → GraphX

```
val g: GraphFrame = ….
val gx: Graph [Row, Row] = g.toGraphX
```

- GraphX → GraphFrame

```
val g2: GraphFrame = GraphFrame.fromGraphX(gx)
```

Graph Construction

- To construct a graphFrame, you need two DataFrames
  - Vertices
    - 1 vertex per row
    - id: column with unique id
  - Edges
    - 1 edge per row
    - src, dst columns using ids from vertices.id
  - **val** g **= GraphFrame**(v, e)
- Any Spark method to save and load DataFrames can be used
  - vertices = sqlContext.read.parquet(…)
  - vertices.write.parquet(…)

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

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

Graph Algorithms

- Find important vertices
  - PageRank *
    - g.pageRank(resetProbability=0.15, tol=0.01)
    - g.pageRank(resetProbability=0.15, maxIter=10)
- Find paths between sets of vertices
  - Breadth-first search (BFS) ¤
    - paths = g.bfs("name = 'Esther'", "age < 32")
  - Shortest path *
    - g.shortestPaths(landmarks=["a", "d"])

* *Mostly wrappers around Spark GraphX algorithms*

¤ *Some algorithms implemented using DataFrames*

Graph Algorithms

- Find groups of vertices (components, communities)
  - Connected components *
  - Strongly connected components *
  - Label Propagation Algorithm (LPA) *
- Other
  - Triangle Counting ¤
  - SVDPlusPlus *
- Pregel

* *Mostly wrappers around Spark GraphX algorithms*

¤ *Some algorithms implemented using DataFrames*

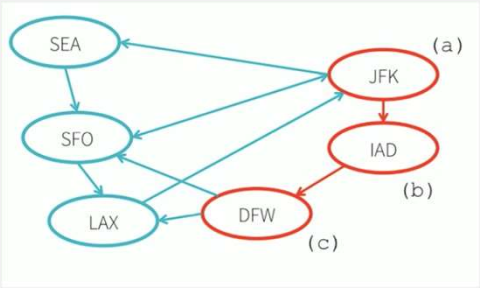
Simple Queries

- SQL queries on vertices & edges
- E.g. what trips are most likely to have significant delays?
  - Display(tripGraph.edges.groupBy("src", "dst").avg("delay").sort(desc("avg(delay)")))
- Graph Queries
  - Vertex degrees
    - Number of edges per vertex (incoming, outgoing, total)

Motif finding

- Search for structural patterns within a graph

```
motifs = g.find("(a)-[e1]->(b);
                 (b)-[e2]->(c);
                 !(c)-[]->(a)")
motifs.filter(“e1.delay > 20 and b.id = ‘SFO’”)
```



- Graphframes find() syntax

```
motifs = g.find("(a)-[e]->(b); (b)-[e2]->(a)")
motifs.filter("b.age > 30").show()
```

- filter

```
g1 = g.filterVertices("age > 30").filterEdges("relationship = 'friend’”)
```

Implementing Algorithms

- Method 1: DataFrame & GraphFrame operations
  - Motif finding
    - Series of DataFrame joins
  - Use other pre-packaged algorithms
- Method 2: Message passing
  - Send messages between vertices, and aggregate messages for each vertex
- Method 3: Pregel

GraphFrames AggregateMessages

- Send messages between vertices, and aggregate messages for each vertex
- aggregateMessages()
  - Similar to GraphX
  - Specify messages and aggregation using DataFrame expressions
- Joins: Join message aggregates with the original graph.
  - GraphFrames rely on DataFrame joins

```
from pyspark.sql.functions import sum as sqlsum
from graphframes.lib import AggregateMessages as AM

# For each user, sum the ages of the adjacent users.
msgToSrc = AM.dst["age"]
msgToDst = AM.src["age"]
agg = g.aggregateMessages( sqlsum(AM.msg).alias("summedAges"),
sendToSrc=msgToSrc, sendToDst=msgToDst)
agg.show()
```

Pregel in GraphFrames

- When a run starts, it expands the vertices DataFrame using column expressions defined by [[withVertexColumn]].
- Example code for PageRank

```
val edges = ...

val vertices = GraphFrame.fromEdges(edges).outDegrees.cache()

val numVertices = vertices.count()

val graph = GraphFrame(vertices, edges)

val alpha = 0.15

val ranks = graph.pregel.withVertexColumn(
                    "rank", lit(1.0 / numVertices),
                     coalesce(Pregel.msg, lit(0.0)) * (1.0 - alpha) + alpha / numVertices
 )

 .sendMsgToDst(Pregel.src("rank ") / Pregel.src("outDegree"))

 .aggMsgs(sum(Pregel.msg))

 .run()}
```

Creates a Column of literal value.

Returns the first column that is not null

CLOUD COMPUTING APPLICATIONS

Spark GraphX

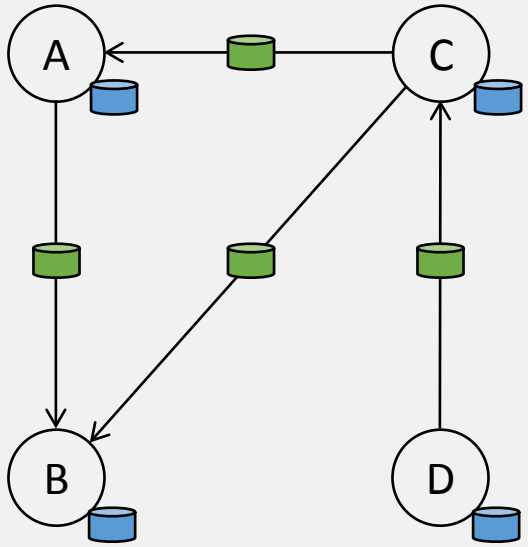
Roy Campbell & Reza Farivar

Many Graph-Parallel Algorithms

- Collaborative Filtering
  - Alternating Least Squares
  - Stochastic Gradient Descent
  - Tensor Factorization
- Structured Prediction
  - Loopy Belief Propagation
  - Max-Product Linear Programs
  - Gibbs Sampling
- Semi-supervised ML
  - Graph SSL
  - CoEM
- Community Detection
  - Triangle-Counting
  - K-core Decomposition
  - K-Truss
- Graph Analytics
  - PageRank
  - Personalized PageRank
  - Shortest Path
  - Graph Coloring
- Classification
  - Neural Networks

View a Graph as a Table

Property Graph



Vertex Property Table

Id	Property (V)
A	(P1., P2.)
B	(P3, P4.)
C	(P5., P2)
D	(P7., P4)

Edge Property Table

SrcId	DstId	Property (E)
A	B	P8
C	A	P9
D	C	P10
C	B	P11

Constructing the Graph

```
// Assume the SparkContext has already been constructed
val sc: SparkContext
// Create an RDD for the vertices
val users: RDD[(VertexId, (String, String))] =
  sc.parallelize(Array((1L, (”A", "student")), (2L, (”B", "P1")),
                       (3L, ("C", "P2")), (4L, ("D", "P3"))))
// Create an RDD for edges
val relationships: RDD[Edge[String]] =
  sc.parallelize(Array(Edge(1L, 2L, "P8"),    Edge(1L, 3L, "P2"),
                       Edge(2L, 4L, "P3"), Edge(3L, 4L, "P4")))
// Build the initial Graph
val graph = Graph(users, relationships)
```

Some Graph Operators

```
class Graph [ V, E ] {
  def Graph(vertices: Table[ (Id, V) ],
            edges: Table[ (Id, Id, E) ])
    // Table Views -----------------
    def vertices: Table[ (Id, V) ]
    def edges: Table[ (Id, Id, E) ]
    // Transformations -----------------------------
    def reverse: Graph[V, E]
    def subgraph(pV: (Id, V) => Boolean,
                 pE: Edge[V,E] => Boolean): Graph[V,E]
    def mapVertices(m: (Id, V) => T ): Graph[T,E]
    def mapEdges(m: Edge[V,E] => T ): Graph[V,T]
    // Joins ---------------------------------------
    def joinVertices(tbl: Table [(Id, Id, T)]): Graph[V, (E,
T)]
    // Computation ---------------------------------
    def connectedComponents(): Graph[V, E]
}
```

CLOUD COMPUTING APPLICATIONS

Graph Processing

Roy Campbell & Reza Farivar

Graph Processing

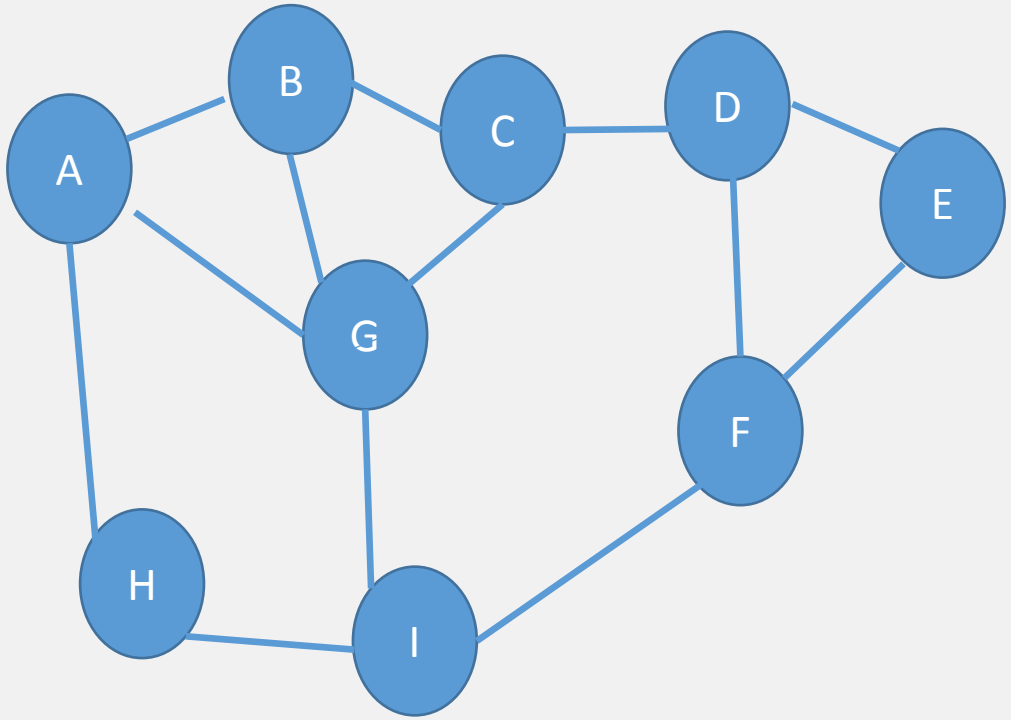
- A graph database is any storage system that provides index-free adjacency. Has pointers to adjacent elements…
- Nodes represent entities (people, businesses, accounts…)
- Properties are pertinent information that relate to nodes
- Edges interconnect nodes to nodes or nodes to properties and they represent the relationship between the two

Graph and Relational Databases

- Graph Database
  - Associative data sets
  - Structure of object-oriented applications
  - Do not require join operators
- Relational Database
  - Perform same operation on large numbers of data elements
  - Use relational model of data
  - Entity type has own table
    - Rows are instances of entity
    - Columns represent values attributed to that instance
  - Rows in one table can be related to rows in another table via unique key per row

Graph

Vertex	Property/Edge	Vertex
A		B, G, H
B		C, G, A
C		B, D, G
D		C, E, F
E		D, F
F		D, E, I
G		A, B, C, I
H		A, I
I		F, G, H



Or use a sparse matrix (table)

Relational Database

Attribute or
Column

Tuple or row

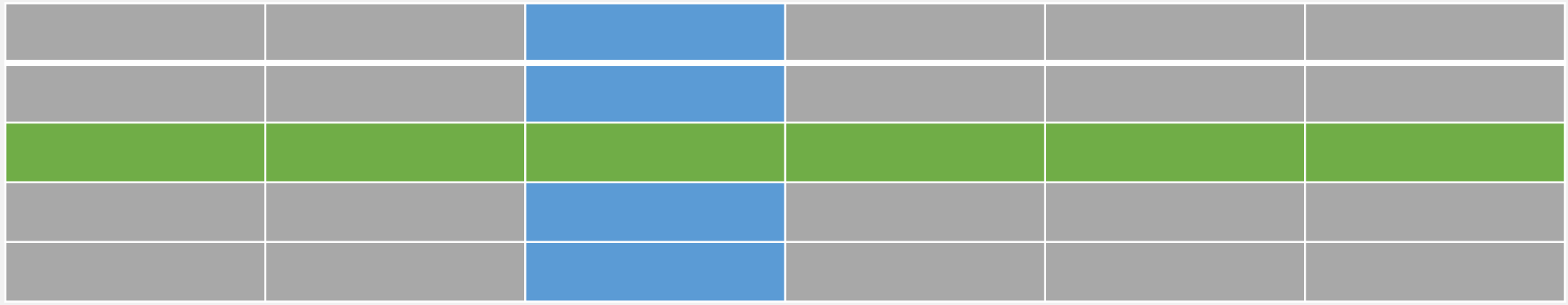
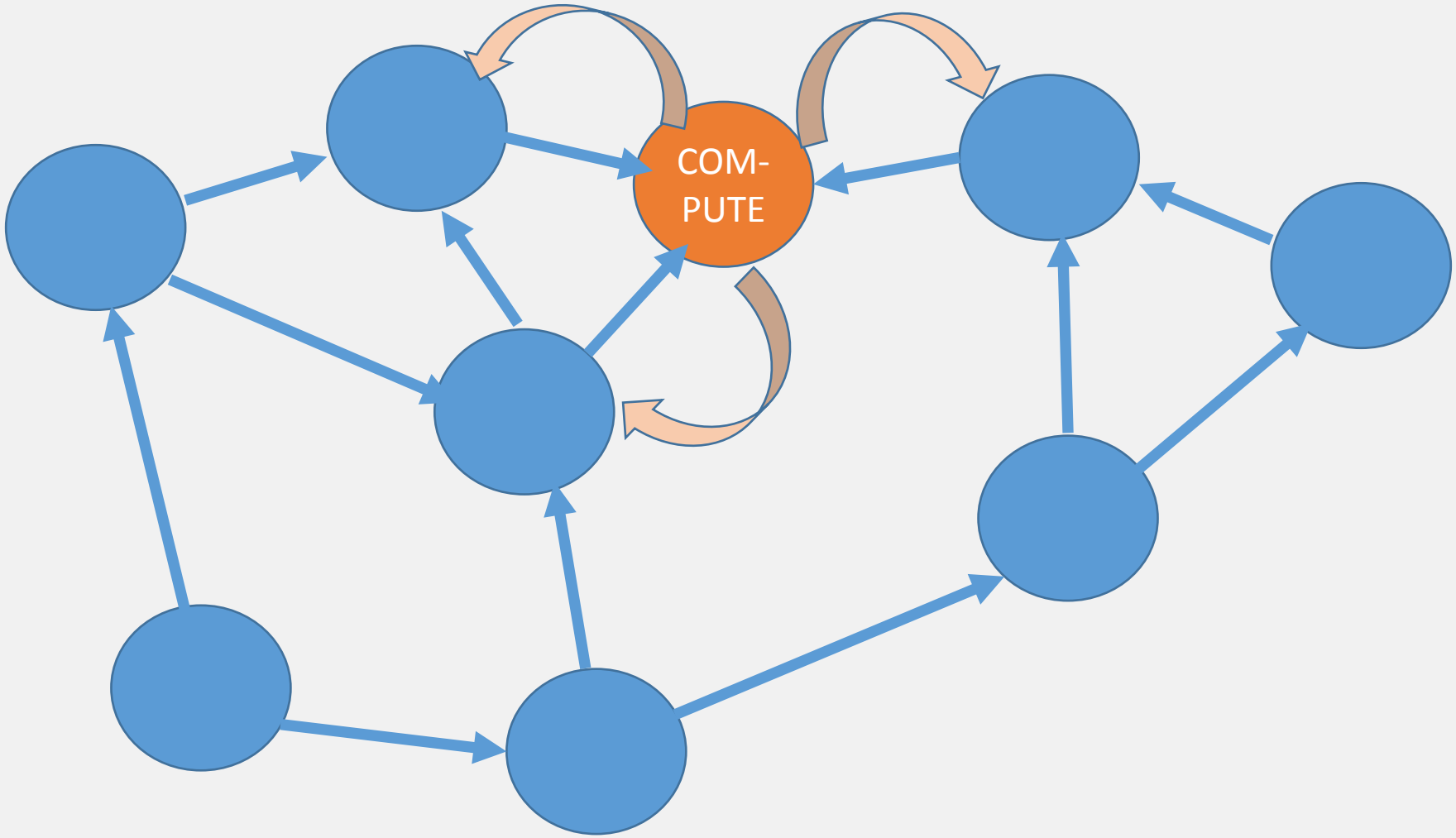


Table is relation

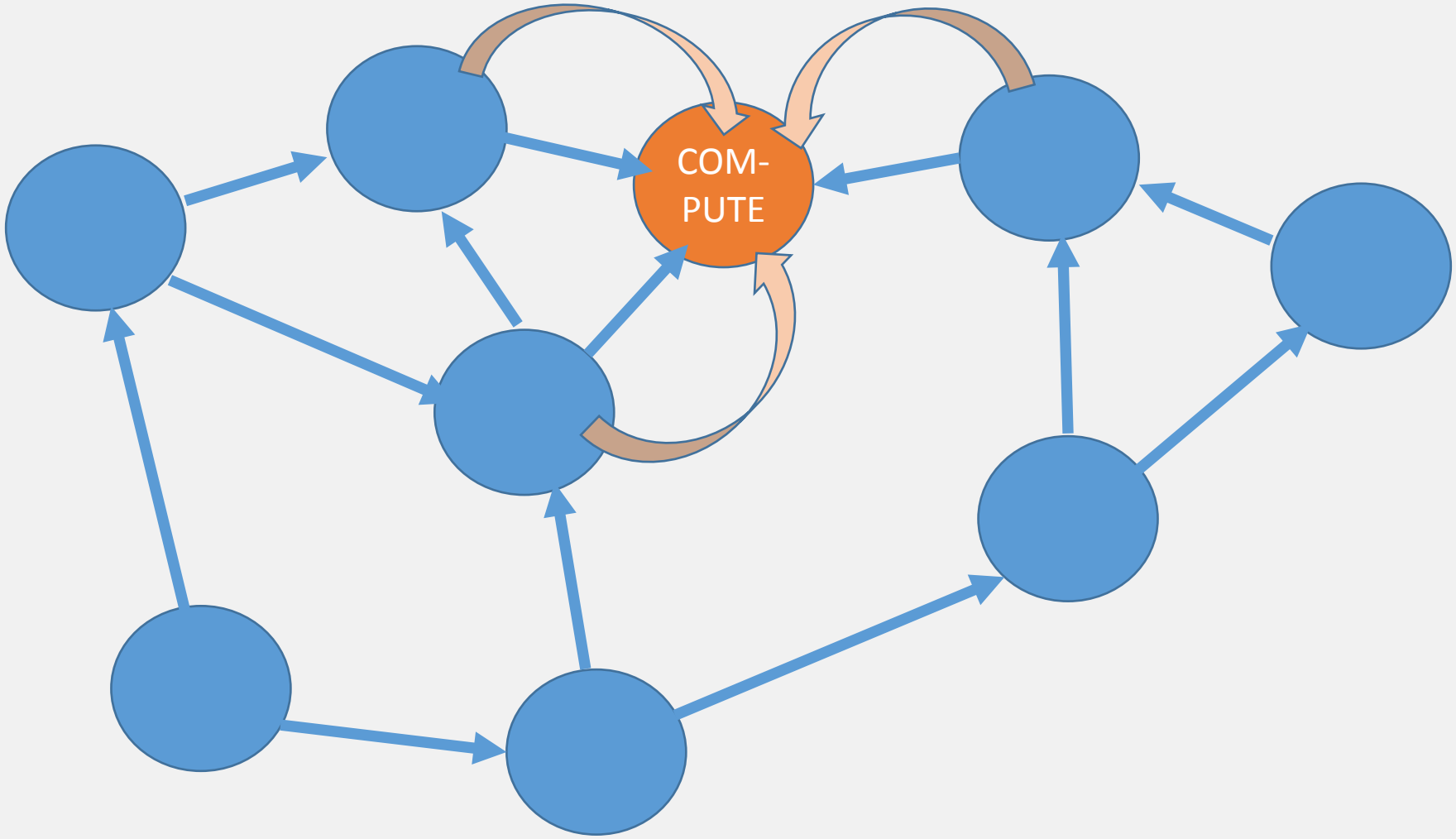
Graph Computing

- Think like a vertex
- Two basic operations:
  - Fusion: aggregate information from neighbors to a set of entities
  - Diffusion: propagate information from a vertex to neighbors

Diffusion



Fusion



Graph Problem Example

Return all sets of vertices (triad) with edges (A,B) (A,C) (C,B) from a directed graph

Graph Processing

Graph computations involve local data (small part of graph surrounding a vertex), and the connectivity between vertices is sparse. The data may not all fit into one node. This makes it difficult to fit always into the map/reduce model.

Large Graph Data	Graph Algorithms
Web	Page Rank
Transportation Routes	Shortest Path
Citation Relationships	Connected Components
Social Networks	Clustering Techniques

Scale of Graphs in Current MLDM Literature

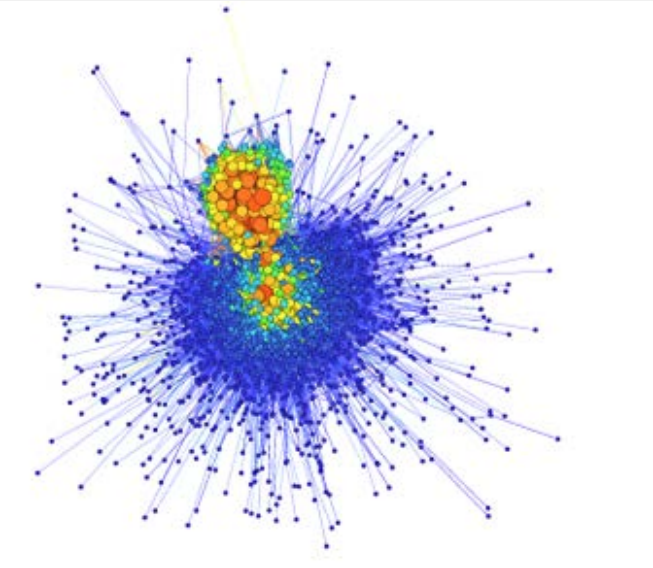
	n (vertices in millions)	m (edges in millions)	size
DBLP	0.3	1	10 MB
AS-Skitter	1.7	11	142 MB
LJ	4.8	69	337.2 MB
USRD	24	58	586.7 MB
BTC	165	773	5.3 GB
WebUK	106	1877	8.6 GB
Twitter	42	1470	24 GB
YahooWeb 2002	1413	6636	120 GB

Graph scale: on order of billions of edges, tens of gigabytes

Scale of Real-World Graphs

Social scale …

- 1 billion vertices, 100 billion edges
- 2.92 TB adjacency list

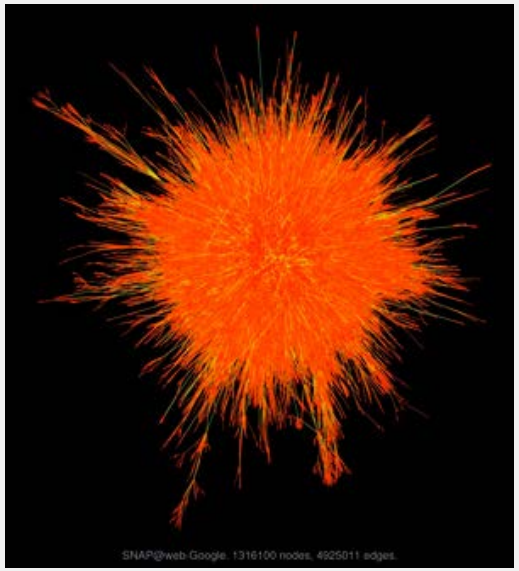


Twitter graph from Gephi data set (http://www.gephi.org)

Scale of Real-World Graphs

Web scale …

- 50 billion vertices, 1 trillion edges
- 29.5 TB adjacency list

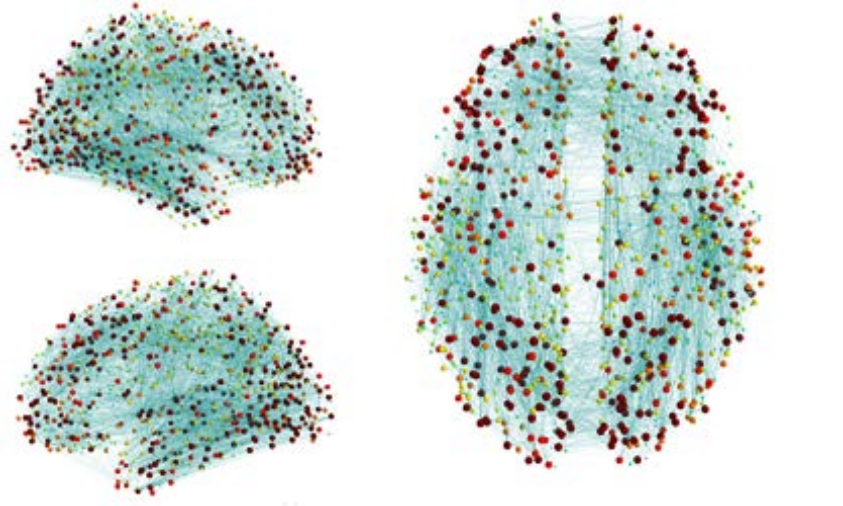


Web graph from the SNAP database (http://snap.stanford.edu/data)

Scale of Real-World Graphs

Brain scale …

- 100 billion vertices, 100 trillion edges
- 2.84 PB adjacency list



Human connectome. Gerhard et al., Frontiers in Neuroinformatics 5(3), 2011

CLOUD COMPUTING APPLICATIONS

Pregel - Part 1

Roy Campbell & Reza Farivar

Introduction

- Infrastructure for graph processing – expensive to design
- Can we use MapReduce?
  - Inefficient because the graph state must be stored at each stage of the graph algorithm, and each computational stage will produce much communication between stages
- Single computer and library approach – not scalable
- Use existing shared memory parallel graph algorithms – no fault-tolerance

Vertex-Oriented

- Based on BSP model
- Provides directed graph to Pregel
- Runs your computation at each vertex (processor)
- Repeats until every computation at each vertex votes to halt
- Pregel returns directed graph as a result

Primitives

- Vertices – first class
- Edges – not first class
- Both vertices can be created and destroyed

Pregel Organized via C++ API

- Supersteps S
- Application code subclasses Vertex, writes a Compute method
- Can get/set Vertex value
- Can get/set outgoing edges values
- Can send/receive messages
- Reads messages sent to V in superstep S-1. Sends messages to other vertices that will be received at superstep S+1; modifies state of V and its outgoing edges

C++ API

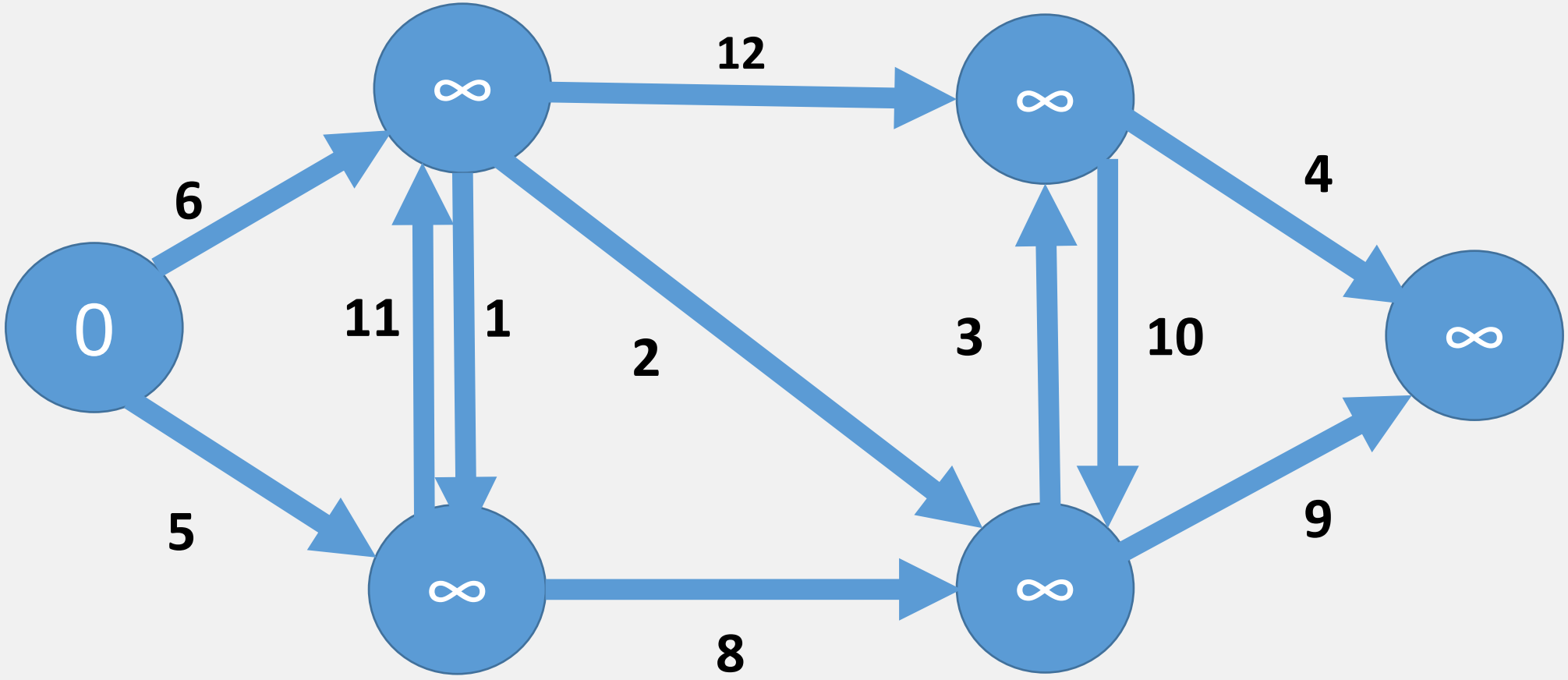
- Message passing
- No guaranteed message delivery order
- Messages delivered exactly once
- Can send a message to any node
- If destination doesn’t exist, user’s function is called

CLOUD COMPUTING APPLICATIONS

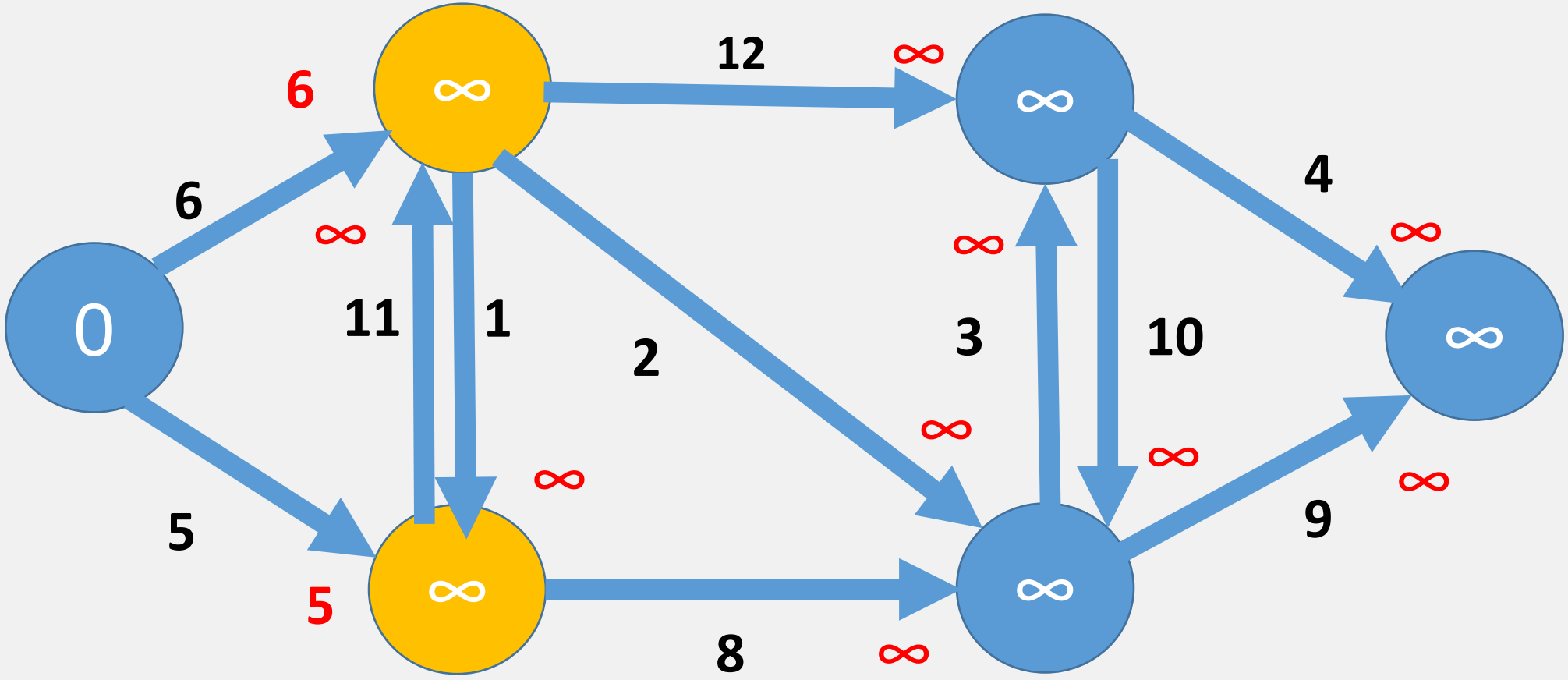
Pregel - Part 2

Roy Campbell & Reza Farivar

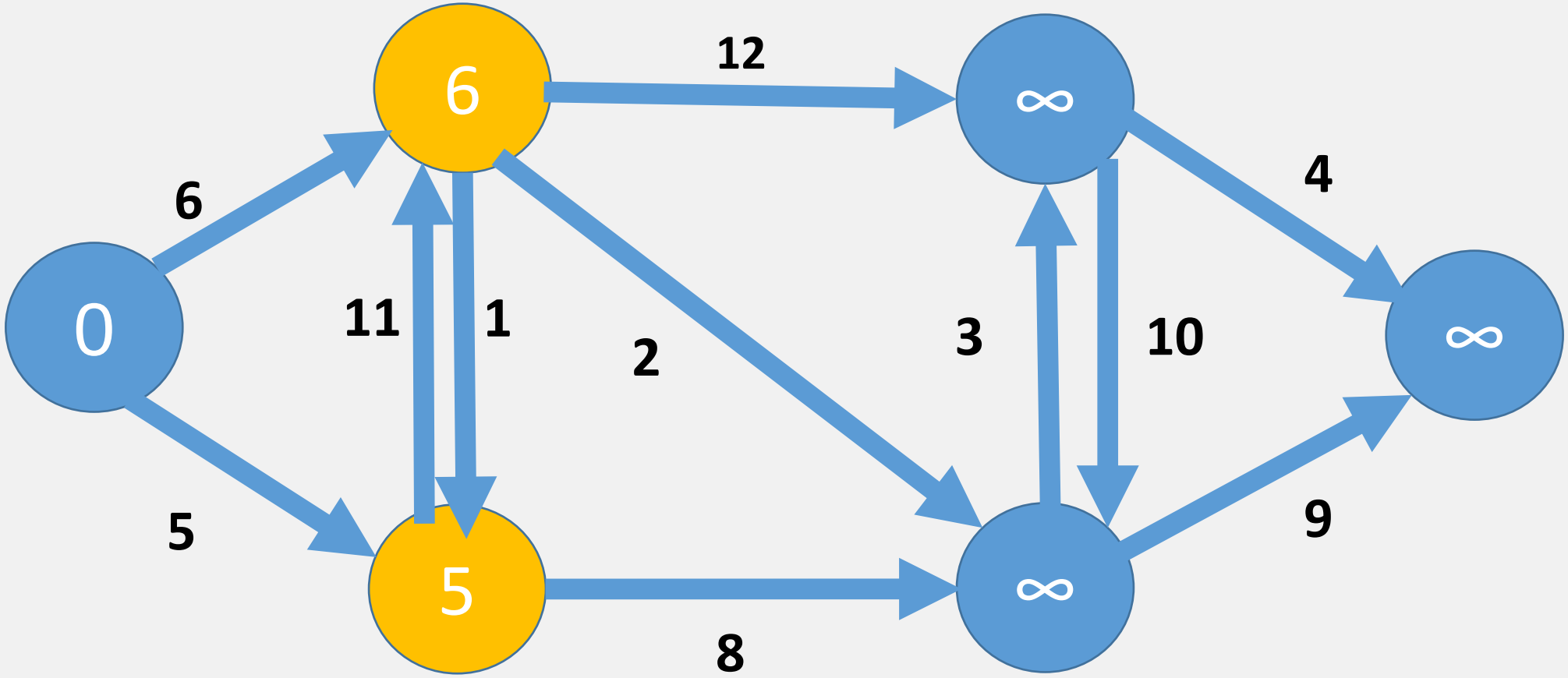
Parallel Breadth: First Search for Shortest Path



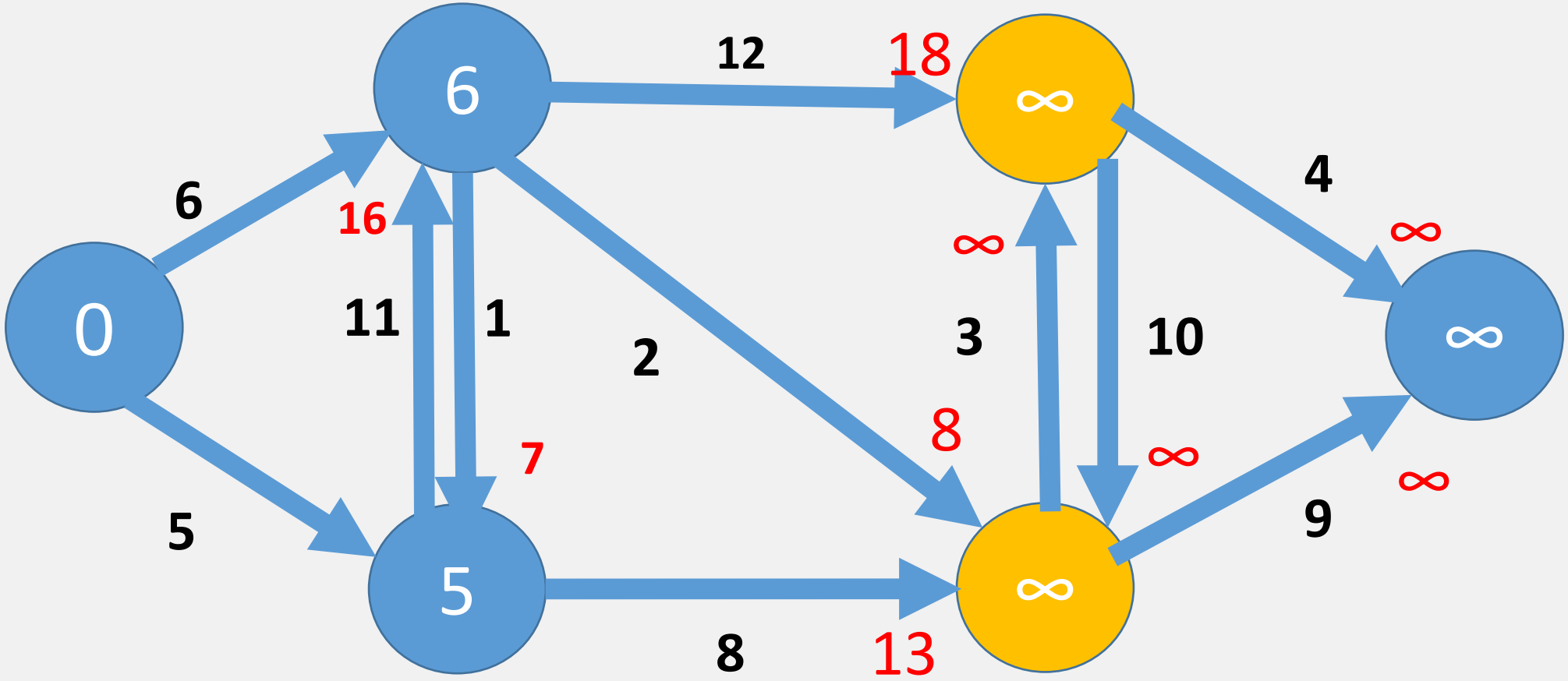
Parallel Breadth: First Search for Shortest Path



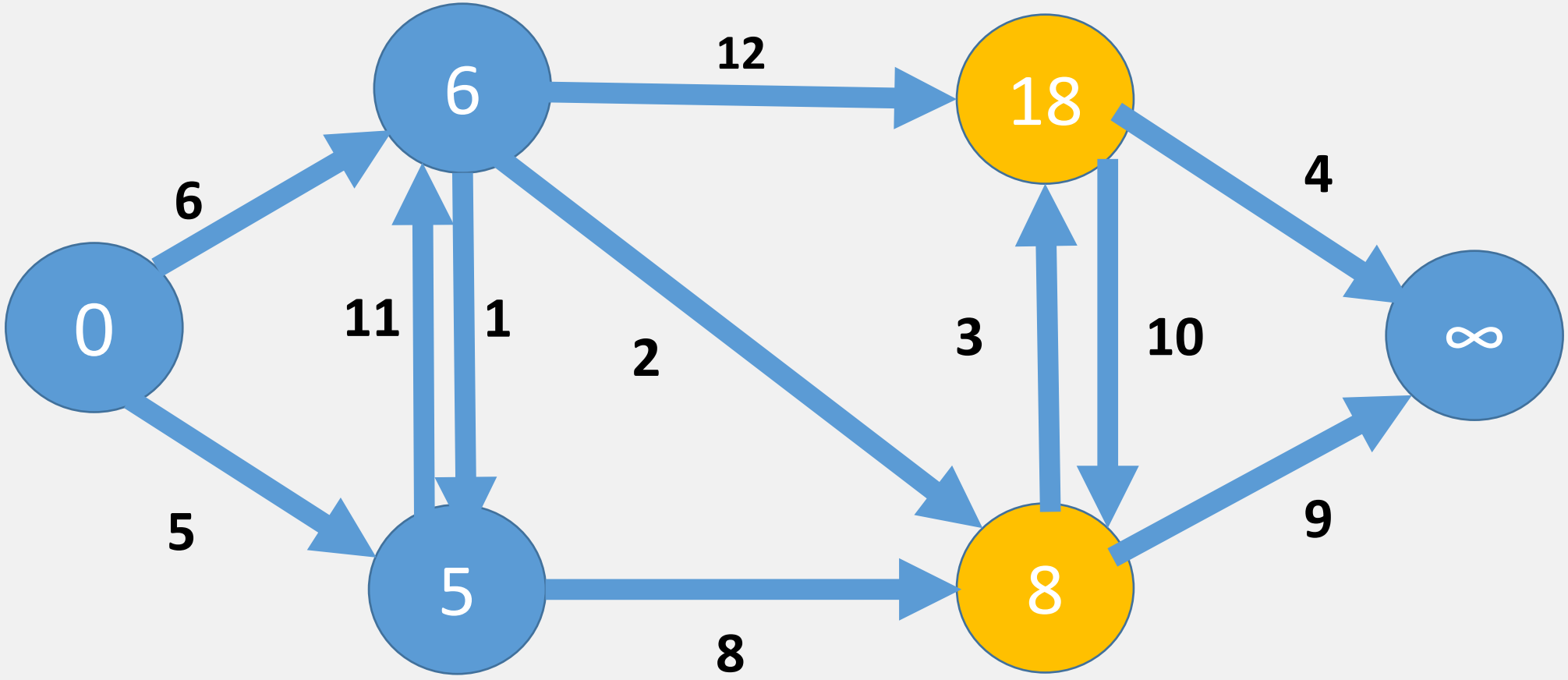
Parallel Breadth: First Search for Shortest Path



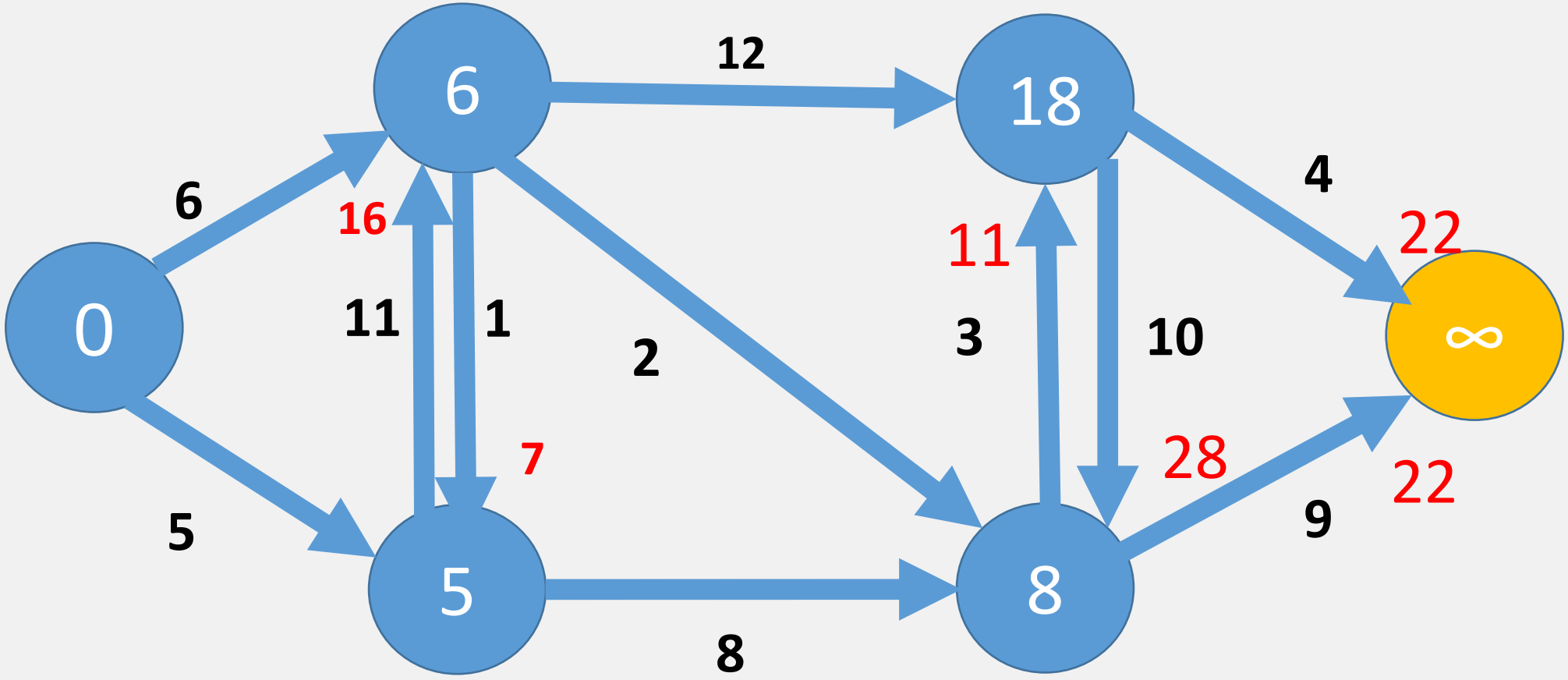
Parallel Breadth: First Search for Shortest Path



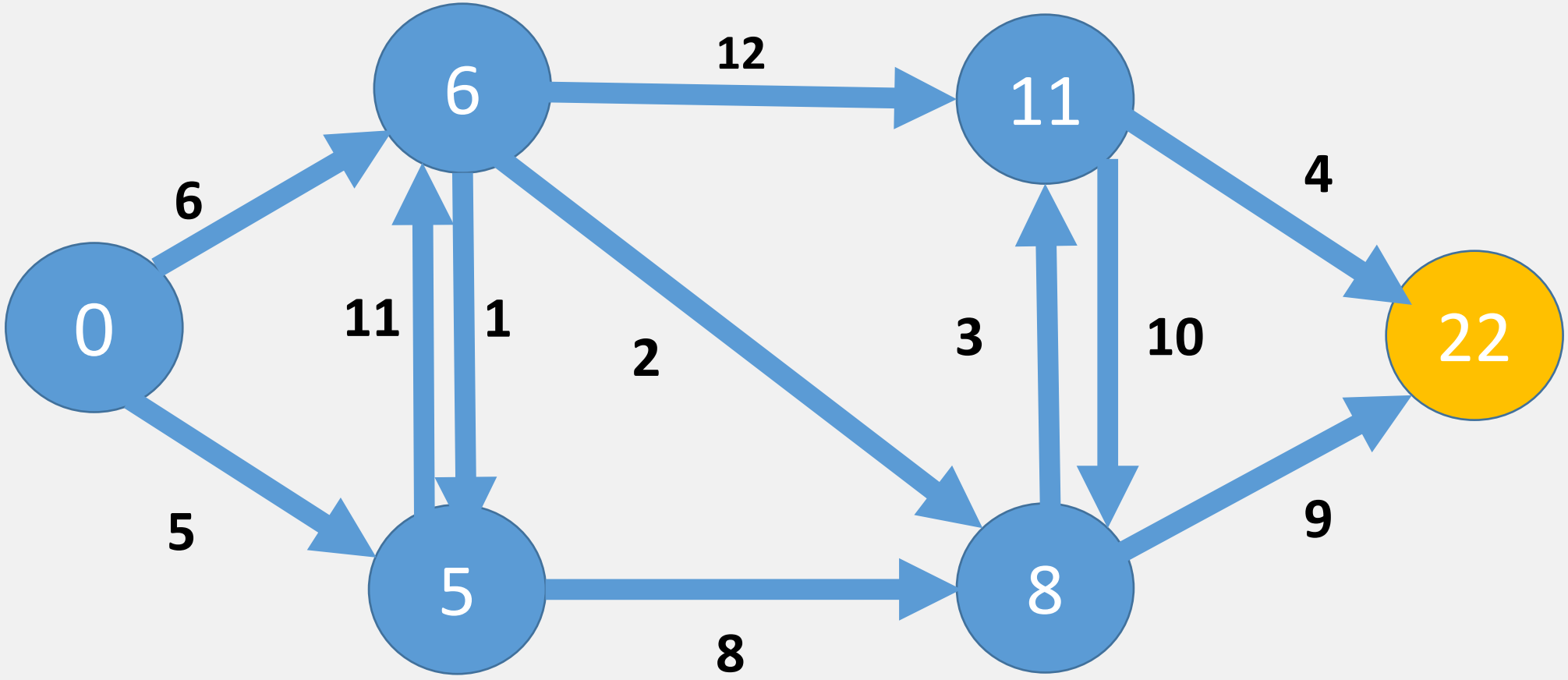
Parallel Breadth: First Search for Shortest Path



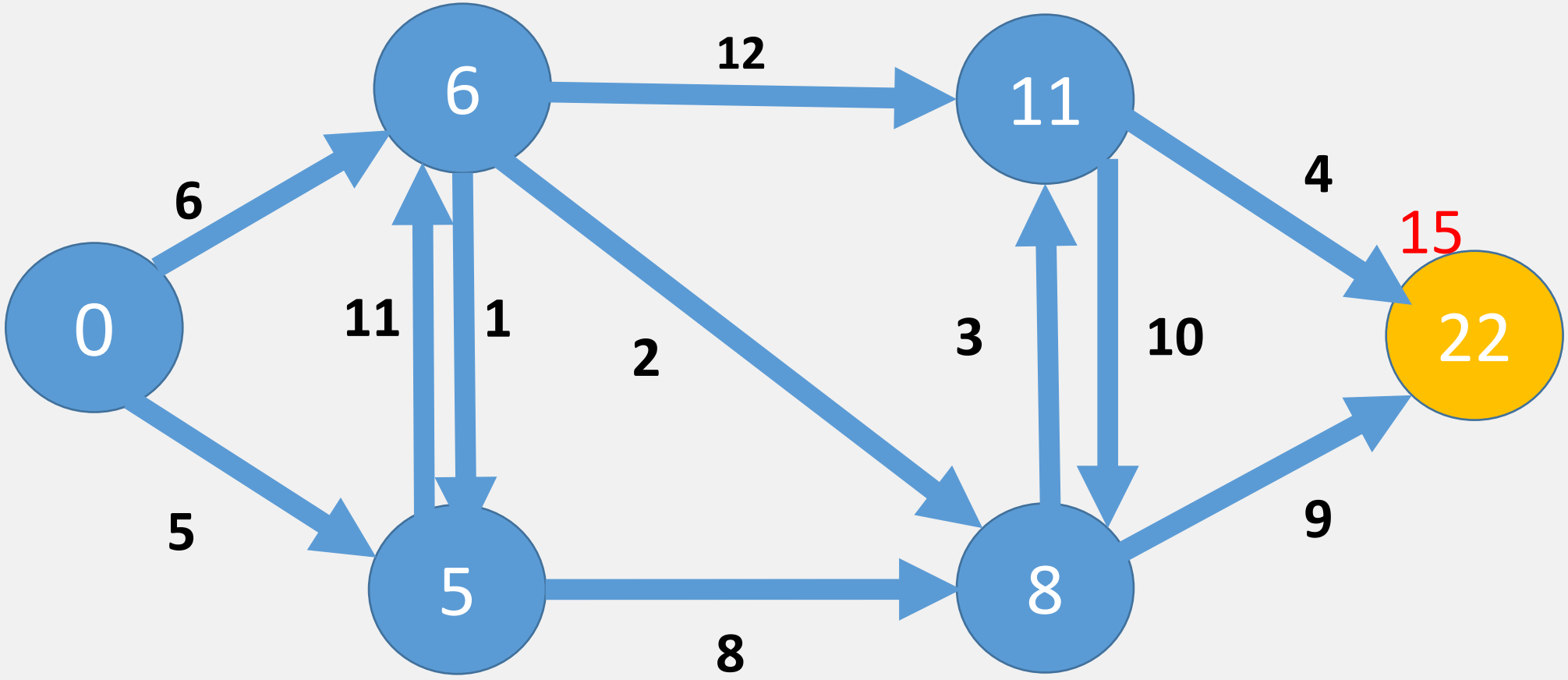
Parallel Breadth: First Search for Shortest Path



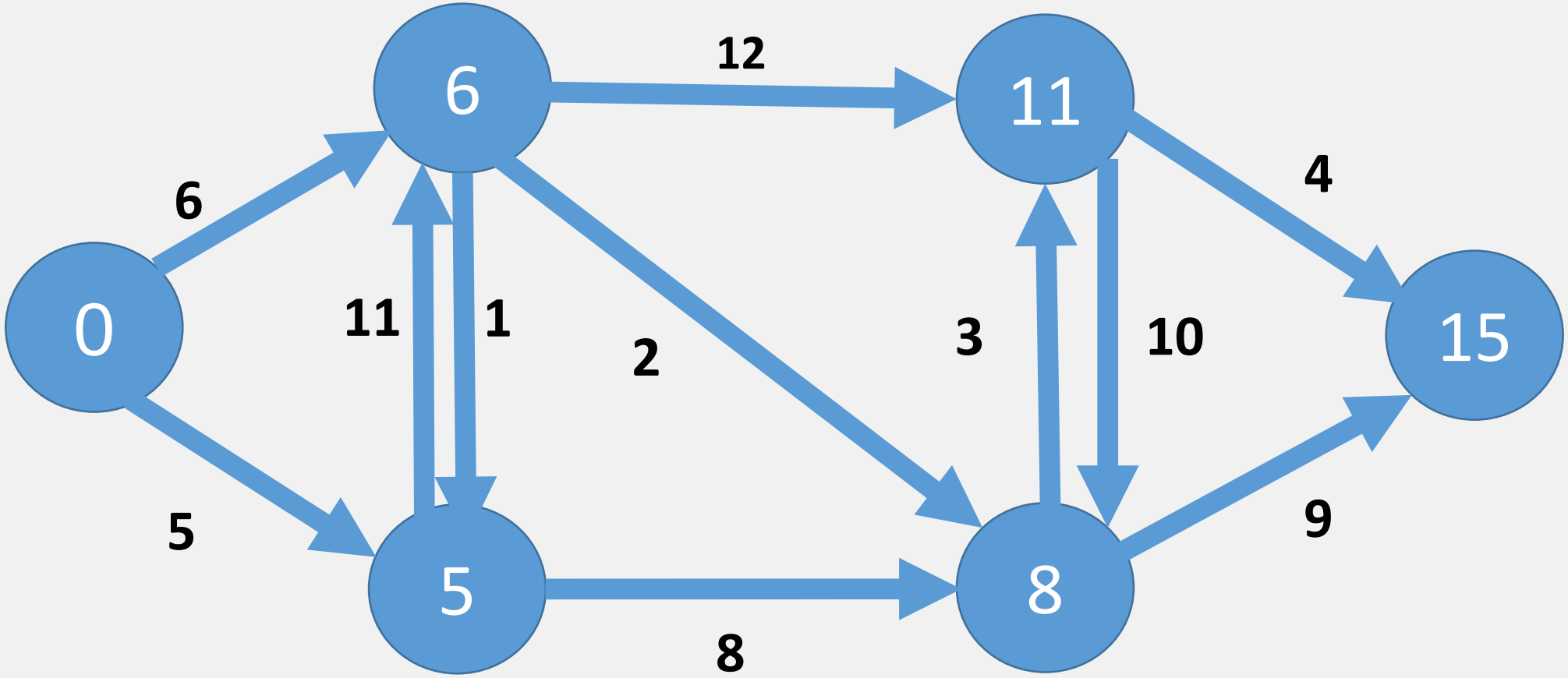
Parallel Breadth: First Search for Shortest Path



Parallel Breadth: First Search for Shortest Path



Parallel Breadth: First Search for Shortest Path



Writing a Pregel Program: C++ API

- Subclassing the predefined Vertex class

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

Override

In

Out

Example: Vertex Class for SSSP

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

CLOUD COMPUTING APPLICATIONS

Pregel - Part 3

Roy Campbell & Reza Farivar

System Architecture

- Pregel system uses the master/worker model
  - Master
    - Maintains worker
    - Recovers faults of workers
    - Provides Web-UI monitoring tool of job progress
  - Worker
    - Processes its task
    - Communicates with the other workers
- Persistent data is stored as files on a distributed storage system (such as GFS, HDFS, or BigTable)
- Temporary data is stored on local disk

Execution of a Pregel Program

1. Many copies of the program begin executing on a cluster of machines
2. The master assigns a partition of the input to each worker
   - Each worker loads the vertices and marks them as active
3. The master instructs each worker to perform a superstep
   - Each worker loops through its active vertices and computes for each vertex
   - Messages are sent asynchronously, but are delivered before the end of the superstep
   - This step is repeated as long as any vertices are active, or any messages are in transit
4. After the computation halts, the master may instruct each worker to save its portion of the graph

Fault Tolerance

- Checkpointing
  - The master periodically instructs the workers to save the state of their partitions to persistent storage
    - e.g., Vertex values, edge values, incoming messages
- Failure detection
  - Using regular “ping” messages
- Recovery
  - The master reassigns graph partitions to the currently available workers
  - The workers all reload their partition state from most recent available checkpoint

CLOUD COMPUTING APPLICATIONS

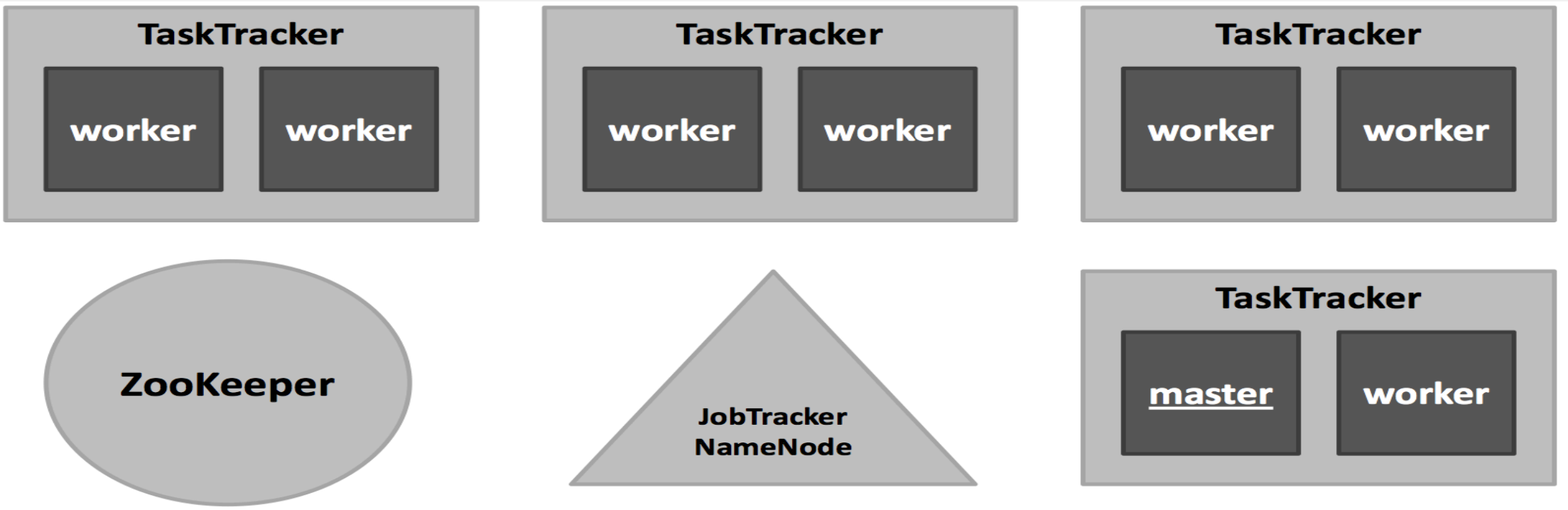
Giraph Introduction

Roy Campbell & Reza Farivar

Apache Giraph

- Open source implementation based on Pregel
- Giraph is currently in Apache incubator
- Modifications are continuous

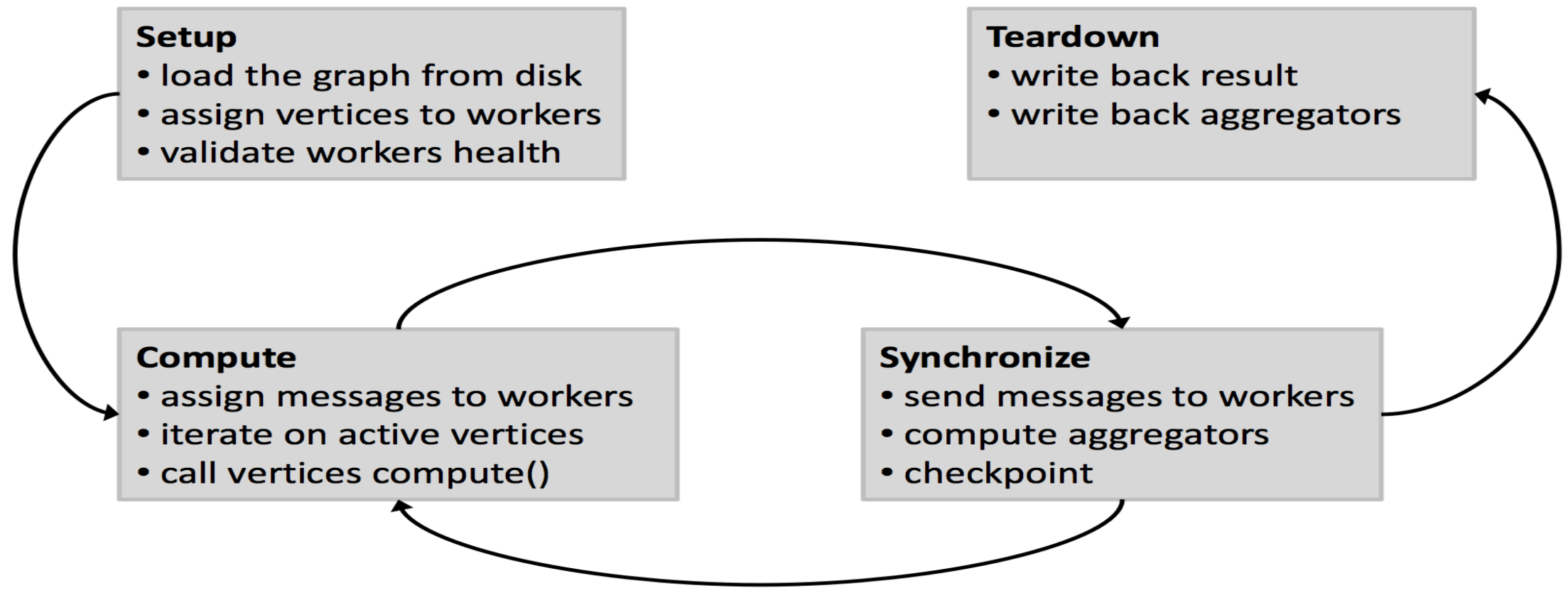
Giraph Framework



Task Assignment

- **ZooKeeper**: responsible for **computation state**
  - Partition/worker mapping
  - Global state: #superstep
  - Checkpoint paths, aggregator values, statistics
- **Master**: responsible for **coordination**
  - Assigns partitions to workers
  - Coordinates synchronization
  - Requests checkpoints
  - Aggregates aggregator values
  - Collects health statuses
- **Worker**: responsible for **vertices**
  - Invokes active vertices compute() function
  - Sends, receives, and assigns messages
  - Computes local aggregation values

Anatomy of an Execution



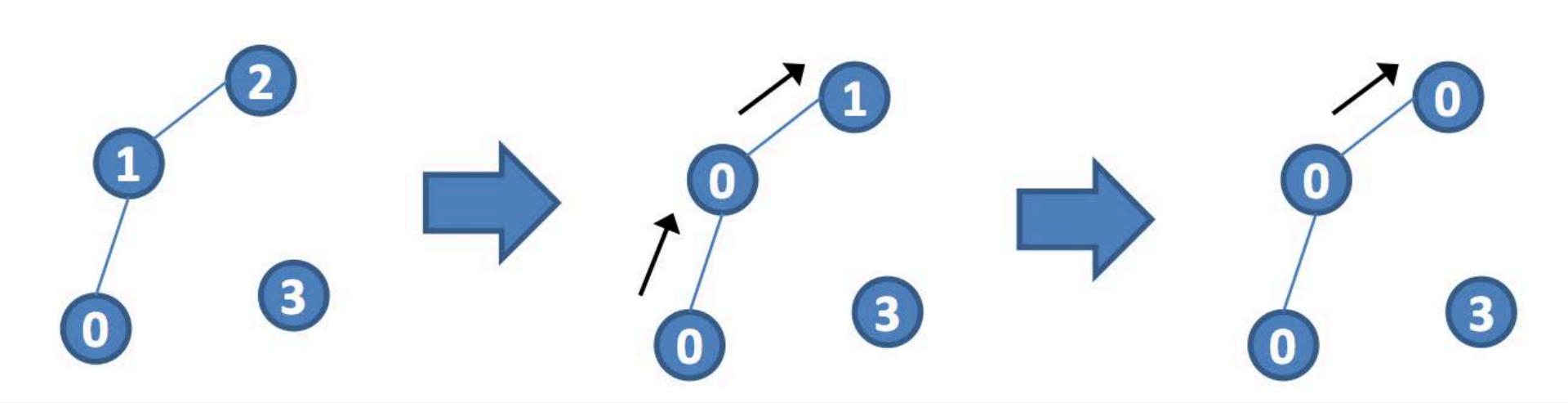
CLOUD COMPUTING APPLICATIONS

Giraph Example

Roy Campbell & Reza Farivar

Giraph Ex: Connected Components of an Undirected Graph

- *Algorithm*: propagate smallest vertex label to neighbors until convergence



- In the end, all vertices of a component will have the same label

Create a Custom Vertex

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.giraph.examples;

import org.apache.giraph.graph.IntIntNullIntVertex;
import org.apache.hadoop.io.IntWritable;

import java.io.IOException;
```

Create a Custom Vertex

```
/**
 * Implementation of the HCC algorithm that identifies connected components and
 * assigns each vertex its "component identifier" (the smallest vertex id
 * in the component)
 *
 * The idea behind the algorithm is very simple: propagate the smallest
 * vertex id along the edges to all vertices of a connected component. The
 * number of supersteps necessary is equal to the length of the maximum
 * diameter of all components + 1
 *
 * The original Hadoop-based variant of this algorithm was proposed by Kang,
 * Charalampos, Tsourakakis and Faloutsos in
 * "PEGASUS: Mining Peta-Scale Graphs", 2010
 *
 * http://www.cs.cmu.edu/~ukang/papers/PegasusKAIS.pdf
 */
@Algorithm(
    name = "Connected components",
    description = "Finds connected components of the graph"
)
```

Create a Custom Vertex

```java
public class ConnectedComponentsVertex extends IntIntNullIntVertex {
  /**
   * Propagates the smallest vertex id to all neighbors. Will always choose to
   * halt and only reactivate if a smaller id has been sent to it.
   *
   * @param messages Iterator of messages from the previous superstep.
   * @throws IOException
   */
  @Override
  public void compute(Iterable<IntWritable> messages) throws IOException {
    int currentComponent = getValue().get();

    // First superstep is special, because we can simply look at the neighbors
    if (getSuperstep() == 0) {
      for (IntWritable neighbor : getNeighbors()) {
        if (neighbor.get() < currentComponent) {
          currentComponent = neighbor.get();
        }
      }
      // Only need to send value if it is not the own id
      if (currentComponent != getValue().get()) {
        setValue(new IntWritable(currentComponent));
        for (IntWritable neighbor : getNeighbors()) {
          if (neighbor.get() > currentComponent) {
            sendMessage(new IntWritable(neighbor.get()), getValue());
          }
        }
      }

      voteToHalt();
      return;
    }

    boolean changed = false;
    // did we get a smaller id ?
    for (IntWritable message : messages) {
      int candidateComponent = message.get();
      if (candidateComponent < currentComponent) {
        currentComponent = candidateComponent;
        changed = true;
      }
    }

    // propagate new component id to the neighbors
    if (changed) {
      setValue(new IntWritable(currentComponent));
      sendMessageToAllEdges(getValue());
    }
    voteToHalt();
  }
}
```

CLOUD COMPUTING APPLICATIONS

Cloud Machine Learning: Introduction

Prof. Reza Farivar

Machine Learning

- **Include machine learning and data mining tools**
  - Analyze/mine/summarize large data sets
  - Extract knowledge from past data
  - Predict trends in future data

Data Mining & Machine Learning

- Subset of Artificial Intelligence (AI)
- Lots of related fields and applications
  - Information retrieval
  - Stats
  - Biology
  - Linear algebra
  - Marketing and sales

Tools & Algorithms

- Collaborative filtering
- Clustering techniques
- Classification algorithms
- Association rules
- Frequent pattern mining
- Statistical libraries (Regression, SVM, …)
- Others…

Common Use Cases

- Recommend friends/dates/products
- Classify content into predefined groups
- Find similar content
- Find associations/patterns in actions/behaviors
- Identify key topics/summarize text
  - Documents and Corpora
- Detect anomalies/fraud
- Ranking search results
- Others?

In Our Context…

--Efficient in analyzing/mining data
--Does not scale

--Efficient in managing big data
--Does not analyze or mine the data

How to integrate these two worlds together

Machine Learning and AI in the Cloud

- Many cloud providers offer PaaS and SaaS products targeting machine learning and AI services
- PaaS for Data Science
  - Structured Data
  - Rows and columns
- PaaS / SaaS for AI
  - Vision / images
  - Voice / Audio
  - Natural language

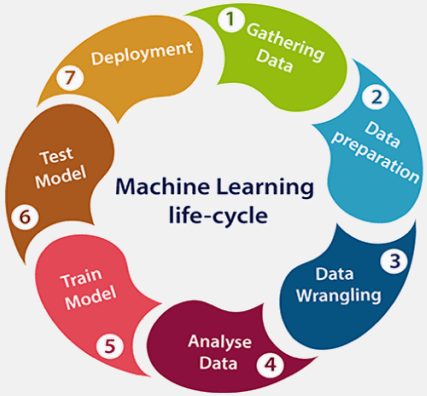
CLOUD COMPUTING APPLICATIONS

Cloud Machine Learning: Workflow

Prof. Reza Farivar

Machine Learning Workflow

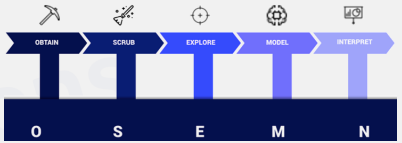
- AI/ML Life Cycle Workflow
  - 7-step model
- The OSEMN Data Science model
  - Obtain
  - Scrub
  - Explore
  - Model
  - Interpret

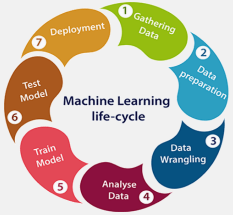


- *A Taxonomy of Data Science* by *Hillary Mason and Chris Wiggins*
- Most providers offer PaaS solutions for the workflow steps

OSEMN: Obtain

- Data Sources on the cloud
  - Amazon: AES Open Data Registry
  - Azure: Open Datasets
  - Google: Cloud Public Datasets

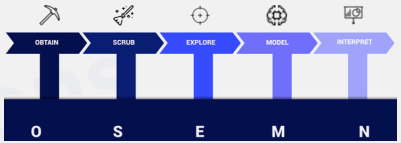


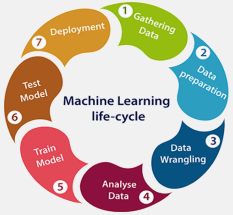


- Command line
- APIs (REST, etc.)
- Jupyter notebooks
- Spreadsheets

OSEMN: Obtain

- Structured Data: Rows and Columns
- Tools:
  - Cloud Storage
  - Cloud Databases
    - SQL
    - NoSQL
      - e.g. MongoDB
  - Big Data
    - Parquet
    - HDFS
    - HDF
    - Pig, Hive



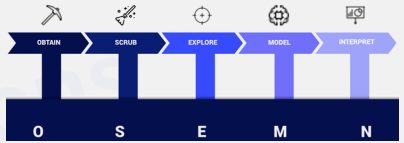


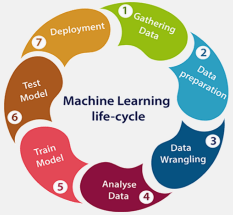
OSEMN: Scrub Data

- Data Preparation and Data Wrangling (2 and 3)
- Clean and filter data
- Consolidate multiple files
- Extracting and replacing values
- Split, merge and extract columns

- Jupyter notebooks

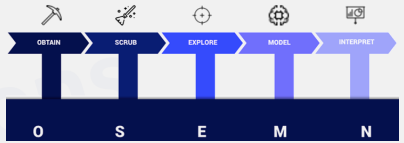
- Python, R for data that can fit in one machine
- Spark, MapReduce for Big Data

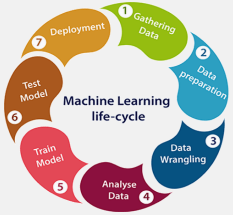




OSEMN: Explore Data

- Inspect data, data wrangling, analyze data (3 and 4)
- Descriptive statistics
- Test significant variables
  - Correlation
- Feature selection
- Data visualization



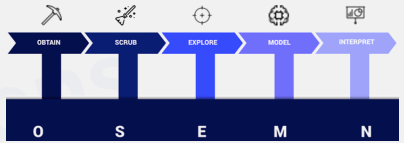


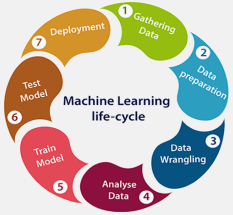
- Jupyter notebooks

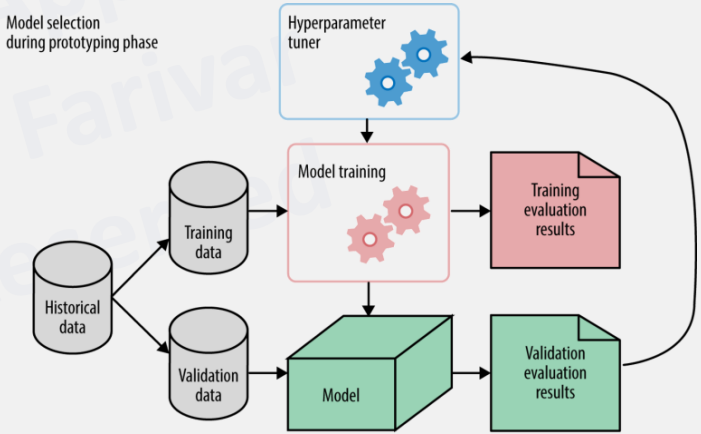
- If data is small → Python, R
  - Numpy
  - Matplotlib
  - Pandas
  - Scipy
- For Big Data
  - Spark
  - EMR

OSEMN: Model Data (1)

- *"Where the magic happens"*
  - Train and test model (5 and 6)
- Feature Engineering
  - Dimensionality reduction
- Model training
  - Regression
  - Classification
  - Clustering
  - Frequent Pattern Mining
  - Decision Trees, Random Forests
  - XGBoost
  - Deep Learning

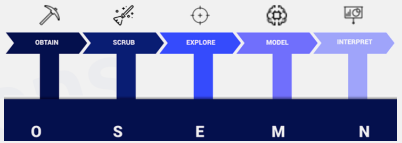


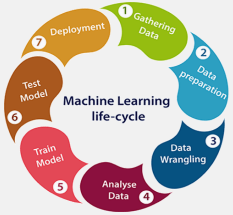




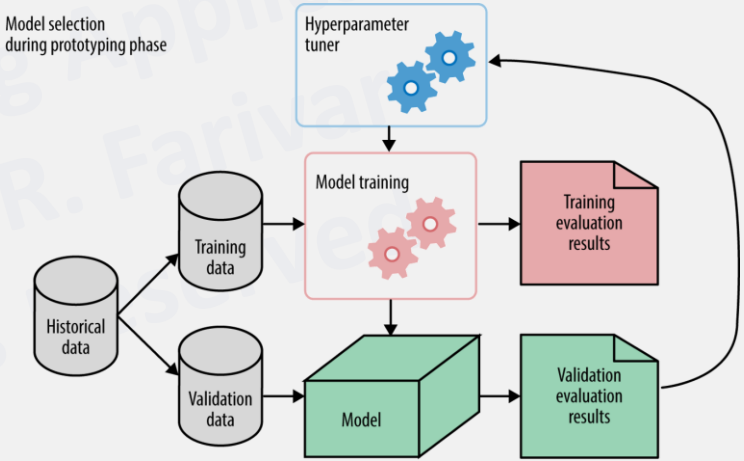
OSEMN: Model Data (2)

- Evaluation
  - Precision
  - Recall
  - F1 Scores
  - Regression
    - MAE (Mean Average Error)
    - RMSE (Root Mean Square Error)



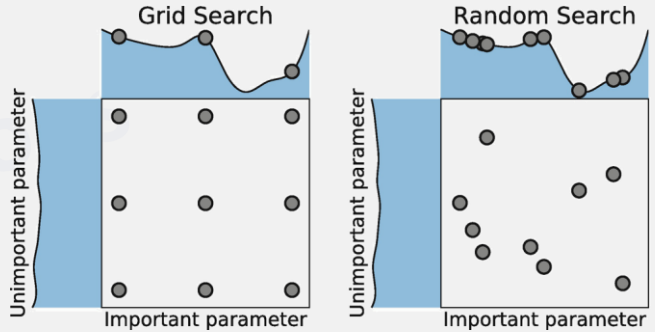


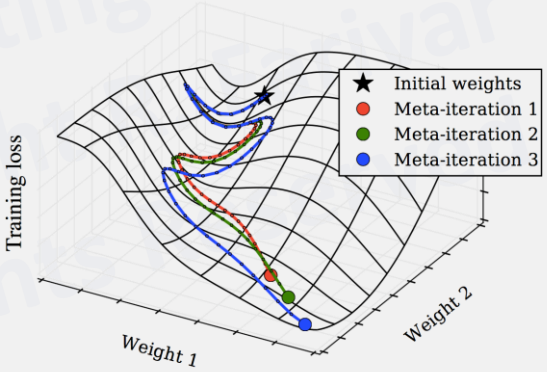
- Small Data: Python, R
  - Scikit Learn
  - H2O
- Big Data
  - Spark Mllib
  - Mahout
  - Google Cloud Dataproc
    - Managed Apache Spark and Hadoop clusters



Hyper Parameter Optimization and AutoML

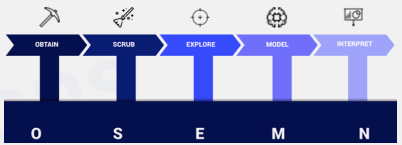
- Hyperparameters: parameters about the training of the model
  - Number of iterations
  - Topology and Size of a neural network
  - Learning rate
- Very time consuming to do manually
- The search space can be huge
- AutoML strategies
  - Grid search
  - Random search
  - Gradient descent
- AutoML vs. Hyperparameter optimization
- Hot competition
  - Azure ML
  - Google AutoML
  - AWS Sagemaker autopilot
  - H2O driverless AI
  - DataRobot

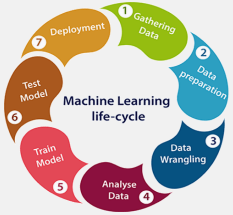




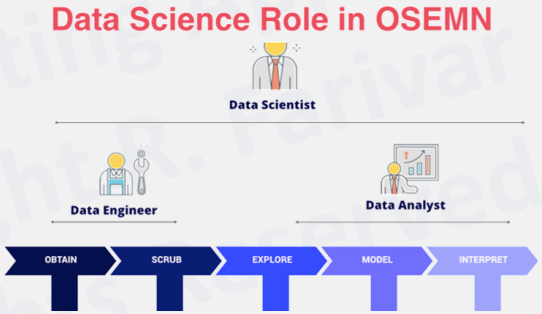
OSEMN: Interpreting Data

- Presentation of the model to a non-technical layman
- Visualizations



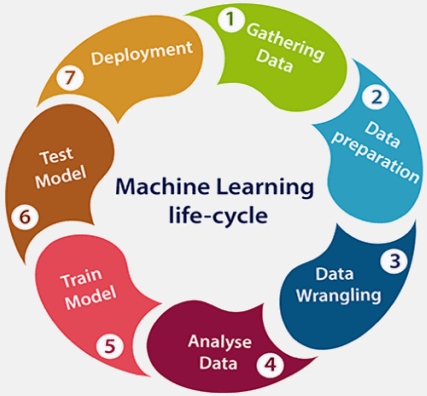


- Matplotlib
- Tableau
- D3.js
- Seaborn



Model Deployment

- Model Artifacts
  - Program
  - Parameters
- Keeping the model up to date
  - Data drift detection
  - Model drift detection
  - Version management
- Example: Google AI Platform Prediction



CLOUD COMPUTING APPLICATIONS

Cloud Machine Learning: Cloud Providers

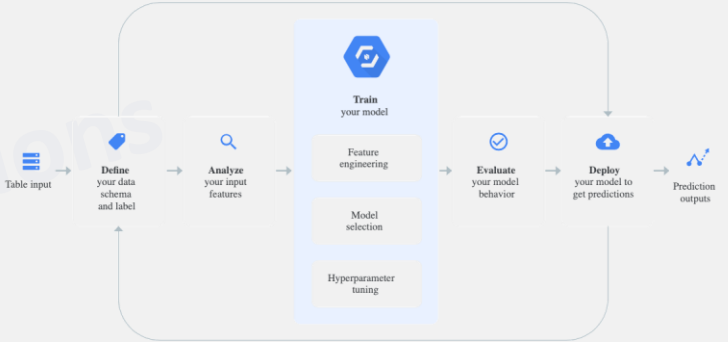
Prof. Reza Farivar

Cloud Based Machine Learning

- ML frameworks on virtual machines
  - Pre-built images
  - Tensorflow
  - MXNet
  - Mahout
- Cloud Managed platforms

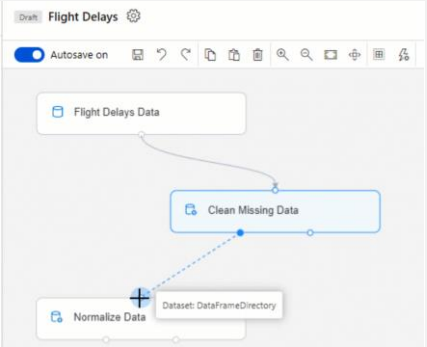
Google Cloud AI Platform

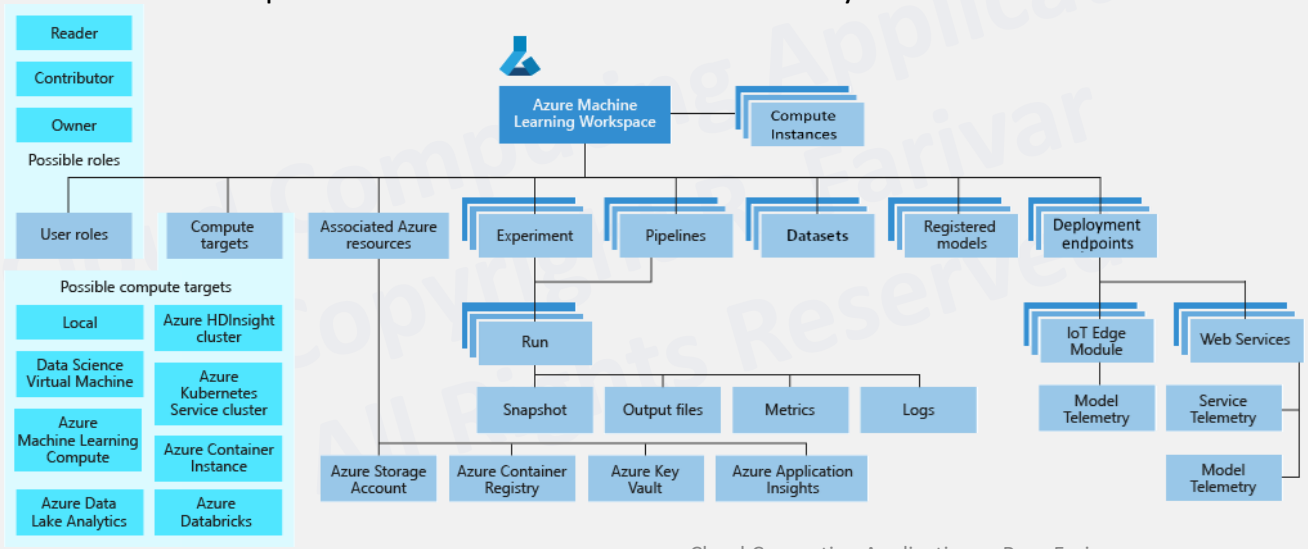
- AI Platform Notebooks
  - Managed notebooks
- AI Platform Training
  - Training with hyperparameter optimizations
- Continuous Evaluation
  - Model optimization
- AI Platform Predictions
  - Server model hosting deployment
- Kubeflow
  - Deployment of machine learning workflows on Kubernetes
- AutoML Tables



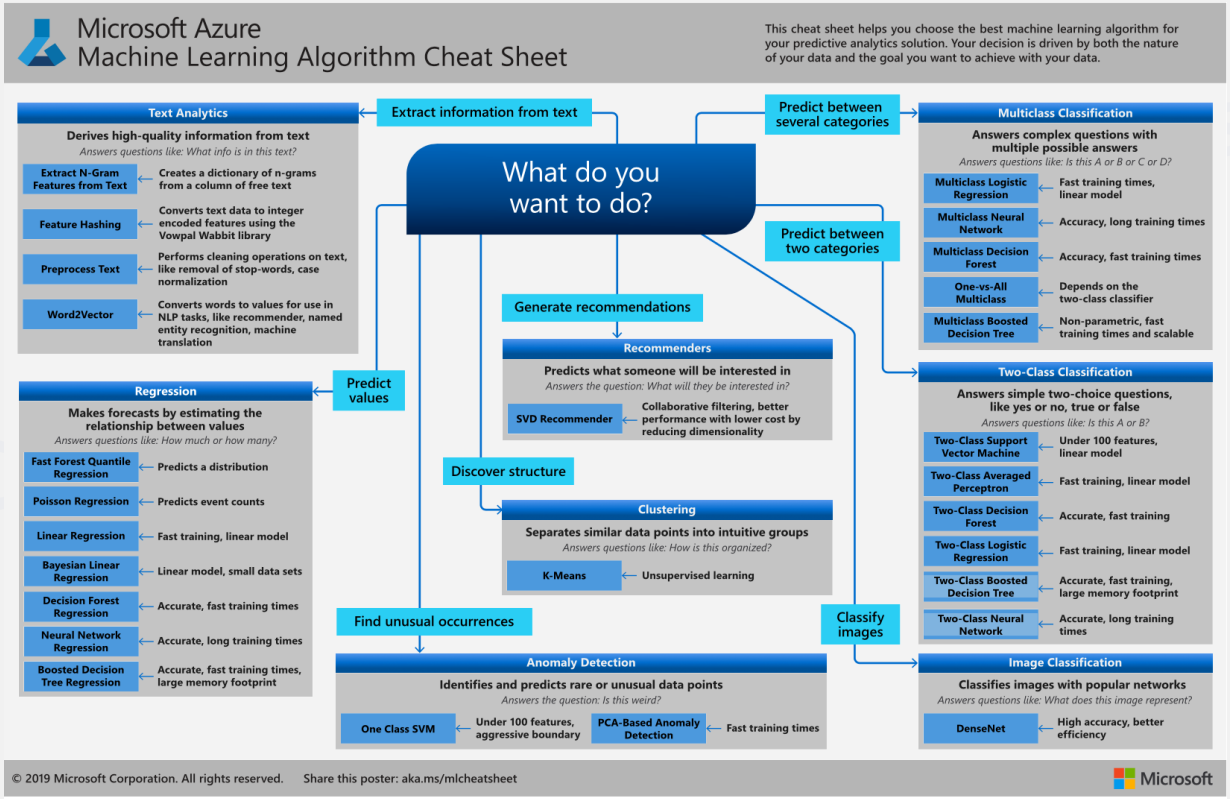
MS Azure Machine Learning

- Managed Platform
- Visual workflow design for no-code ML tasks: designer
- Managed Jupyter notebooks
  - The platform can launch the backend VM for you





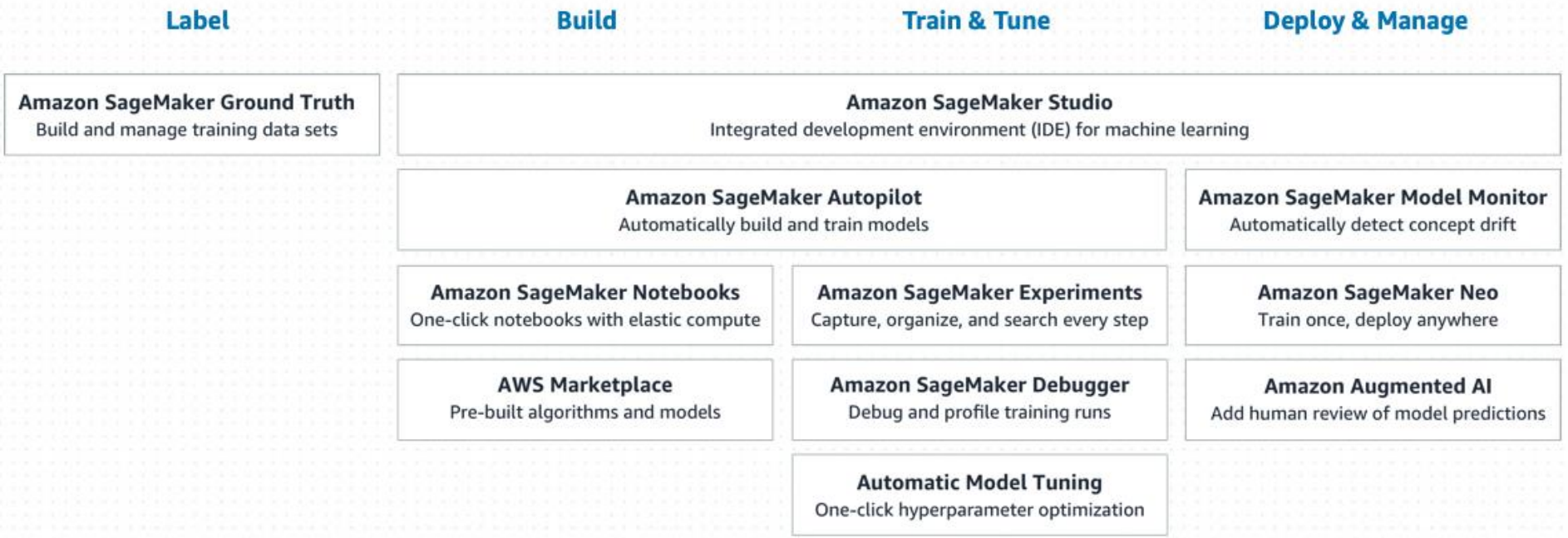
Azure Machine Learning Algorithm Cheat Sheet



Big Data Deployment on the Cloud

- Azure Databricks
  - Managed Spark Deployment
  - Spark MLlib
    - RDD based API
      - Older, will be deprecated
    - DataFrame based API
      - Also called Spark ML
- Azure HDInsight
  - Hadoop → Mahout
  - Spark
  - ML Services
    - Python and R-based analytics

AWS SageMaker



Cloud Machine Learning Artifacts Case Study

- Sagemaker training algorithms are packaged as Docker images
- Training
  - Any training algorithm can be utilizes as long as properly packaged
  - Sagemaker training job:
    - Launches a ML compute instance
    - Runs the train docker image
      - Creates the docker container in the ML compute instance
    - Injects the training data from an S3 location into the container
    - Uses the training code and training dataset to train the model
    - Saves the resulting model artifacts and other output in the output S3 bucket
- Model deployment
  - Creates the model resource
    - S3 path where the model artifacts are stored
    - Docker registry path for the image that contains the inference code
  - Creates an HTTPS endpoint
  - ML compute instances to deploy the model resource
- Model Usage
  - The client application sends requests to the Sagemaker HTTPS endpoint to obtain inferences from a deployed model

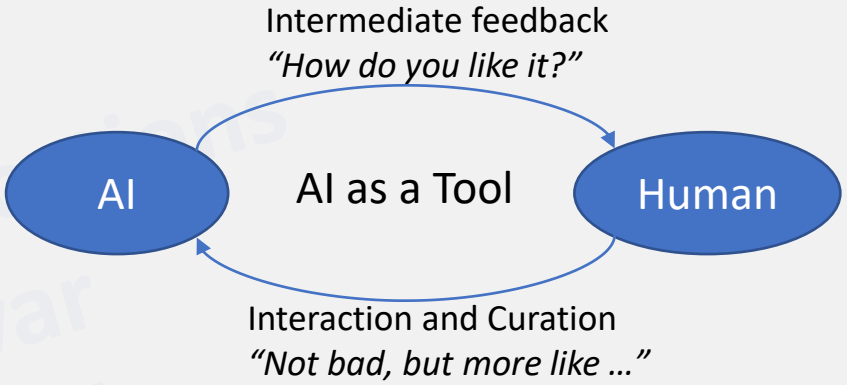
CLOUD COMPUTING APPLICATIONS

Cloud Machine Learning: Human in the loop AI

Prof. Reza Farivar

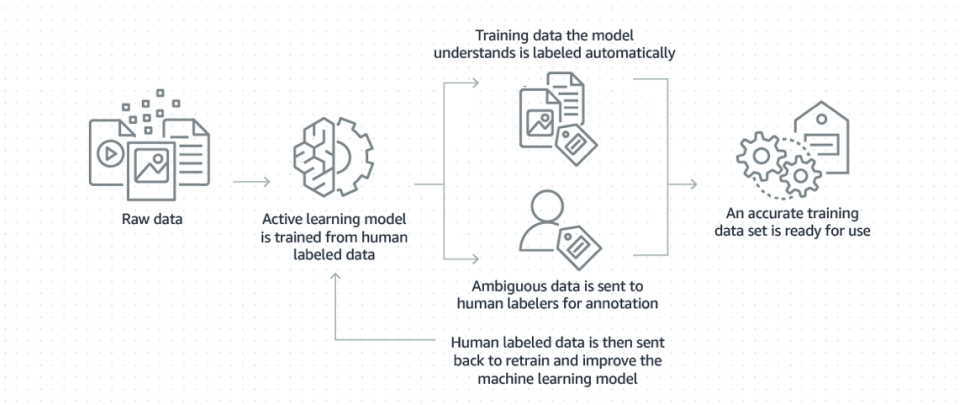
Human in the loop

- Instead of removing human involvement, use selective inclusion of human participation
- Harness the efficiency of computers, while utilizing human intelligence
- reframe an automation problem as a Human-Computer Interaction (HCI) design problem
- Benefits:
  - Gain in transparency
  - Human judgement
  - No need to build the perfect AI system



Human in the loop ML

- Amazon SageMaker Ground Truth
- Build accurate datasets
  - Your own data labelers
  - Cloud-provided: Amazon Mechanical Turk



Human in the loop ML

- Amazon Sagemaker Augmented AI
- Human review of ML predictions
  - Your own reviewers
  - Cloud-provided: Amazon Mechanical Turk

CLOUD COMPUTING APPLICATIONS

Cloud Machine Learning: Unstructured Data

Prof. Reza Farivar

Cloud Machine Learning for Unstructured Data

- Vision
- Voice
- Language

Vision-based ML services

- AWS
  - Rekognition
  - Textractor
- Azure
  - Form Recognizer
  - Computer Vision
  - Face
  - Ink Recognizer
  - Video Indexer
  - Bing Visual Search
  - Bing Image Search
  - Bing Video Search
  - Kinect SDK
- Google
  - Vision AI
  - Video AI
- IBM
  - Watson Visual Recognition

Voice-based services

- Amazon
  - Polly
  - Transcribe
  - Translate
  - Lex
- Google
  - Cloud Speech-to-Text API
  - Cloud Text-to-Speech API
- Microsoft Azure
  - Speech to Text
  - Text to Speech
  - Speech Translation
- IBM
  - Watson Speech to Text
  - Watson Text to Speech

Natural Language Services

- Amazon Azure
  - Comprehend
  - Polly
  - Lex
- Microsoft
  - Language Understanding
  - QnA Maker
  - Translator Text
  - Immersive Reader
- Google
  - Natural Language
  - Translation
- IBM
  - Watson Natural Language Classifier
  - Watson Language Translator
  - Watson Knowledge Studio

CLOUD COMPUTING APPLICATIONS

Spark MLlib

Roy Campbell & Reza Farivar

Spark MLlib

- Spark's machine learning (ML) library
  - Ease of Use
  - Scalable

Collection of ML Libraries: Classification and Regression

- linear models (SVMs, logistic regression, linear regression)
- naive Bayes
- decision trees
- ensembles of trees (Random Forests and Gradient-Boosted Trees)

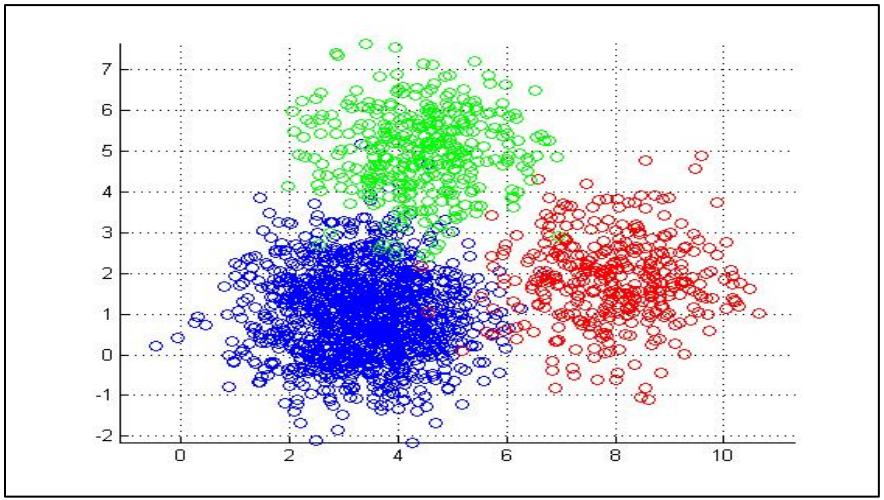
Collection of ML Libraries: Clustering

- k-means
- Gaussian mixture
- power iteration clustering (PIC)
- latent Dirichlet allocation (LDA)
- streaming k-means

Collection of ML Libraries: Dimensionality Reduction

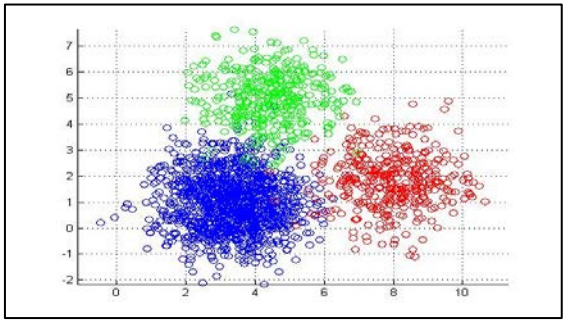
- Singular Value Decomposition (SVD)
- Principal Component Analysis (PCA)

Example: K-Means Clustering



K-Means in MapReduce

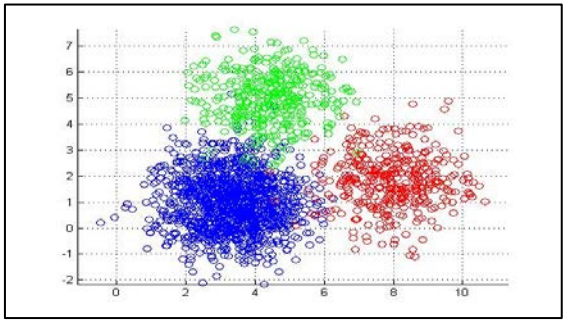
- **Input**
  - Dataset (set of points in 2D) --Large
  - Initial centroids (K points) --Small



- **Map Side**
  - Each map reads the K-centroids + one block from dataset
  - Assign each point to the closest centroid
  - Output <centroid, point>

K-Means in MapReduce (Cont'd)

- **Reduce Side**
  - Gets all points for a given centroid
  - Re-compute a new centroid for this cluster
  - Output: <new centroid>



- **Iteration Control**
  - Compare the old and new set of K-centroids
    - If similar ➔ Stop
    - Else
      - If max iterations has reached ➔ Stop
      - Else ➔ Start another Map-Reduce Iteration

K-Means Clustering in Spark

```
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors

// Load and parse the data
val data = sc.textFile("data/mllib/kmeans_data.txt")
val parsedData = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble))).cache()

// Cluster the data into two classes using KMeans
val numClusters = 2
val numIterations = 20
val clusters = KMeans.train(parsedData, numClusters, numIterations)

// Evaluate clustering by computing Within Set Sum of Squared Errors
val WSSSE = clusters.computeCost(parsedData)
println("Within Set Sum of Squared Errors = " + WSSSE)

// Save and load model
clusters.save(sc, "myModelPath")
val sameModel = KMeansModel.load(sc, "myModelPath")
```

CLOUD COMPUTING APPLICATIONS

Spark Naïve Bayes

Roy Campbell & Reza Farivar

Definition

Naive Bayes is a simple multiclass classification algorithm with the assumption of independence between every pair of features.

NaiveBayes implements multinomial naive Bayes.

Input is an RDD of LabeledPoint and an optionally smoothing parameter lambda

Output is a NaiveBayesModel

```
from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint

def parseLine(line):
    parts = line.split(',')
    label = float(parts[0])
    features = Vectors.dense([float(x) for x in parts[1].split(' ')])
    return LabeledPoint(label, features)

data = sc.textFile('data/mllib/sample_naive_bayes_data.txt').map(parseLine)

# Split data aproximately into training (60%) and test (40%)
training, test = data.randomSplit([0.6, 0.4], seed = 0)

# Train a naive Bayes model.
model = NaiveBayes.train(training, 1.0)

# Make prediction and test accuracy.
predictionAndLabel = test.map(lambda p : (model.predict(p.features), p.label))
accuracy = 1.0 * predictionAndLabel.filter(lambda (x, v): x == v).count() / test.count()

# Save and load model
model.save(sc, "myModelPath")
sameModel = NaiveBayesModel.load(sc, "myModelPath")
```

CLOUD COMPUTING APPLICATIONS

Frequent Pattern Mining

Roy Campbell & Reza Farivar

Spark mllib and fpm

spark.mllib provides a parallel implementation of FP-growth, a popular algorithm to mining frequent itemsets.

FP-growth algorithm: Given a dataset of transactions,

1. Calculate item frequencies and identify frequent items,
2. Use a suffix tree (FP-tree) structure to encode transactions,
3. Extract frequent itemsets from the FP-tree.

Frequent Pattern Mining: FP-growth

FPgrowth takes a RDD of transactions, where each transaction is an Array of items of a generic type.

Calling FPGrowth.run with transactions returns
an FPGrowthModel that stores the frequent item sets with their frequencies.

```
from pyspark.mllib.fpm import FPGrowth

data = sc.textFile("data/mllib/sample_fpgrowth.txt")
transactions = data.map(lambda line: line.strip().split(' '))
model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)
result = model.freqItemsets().collect()
for fi in result:
    print(fi)
```