Beyond Map-Reduce & Spark

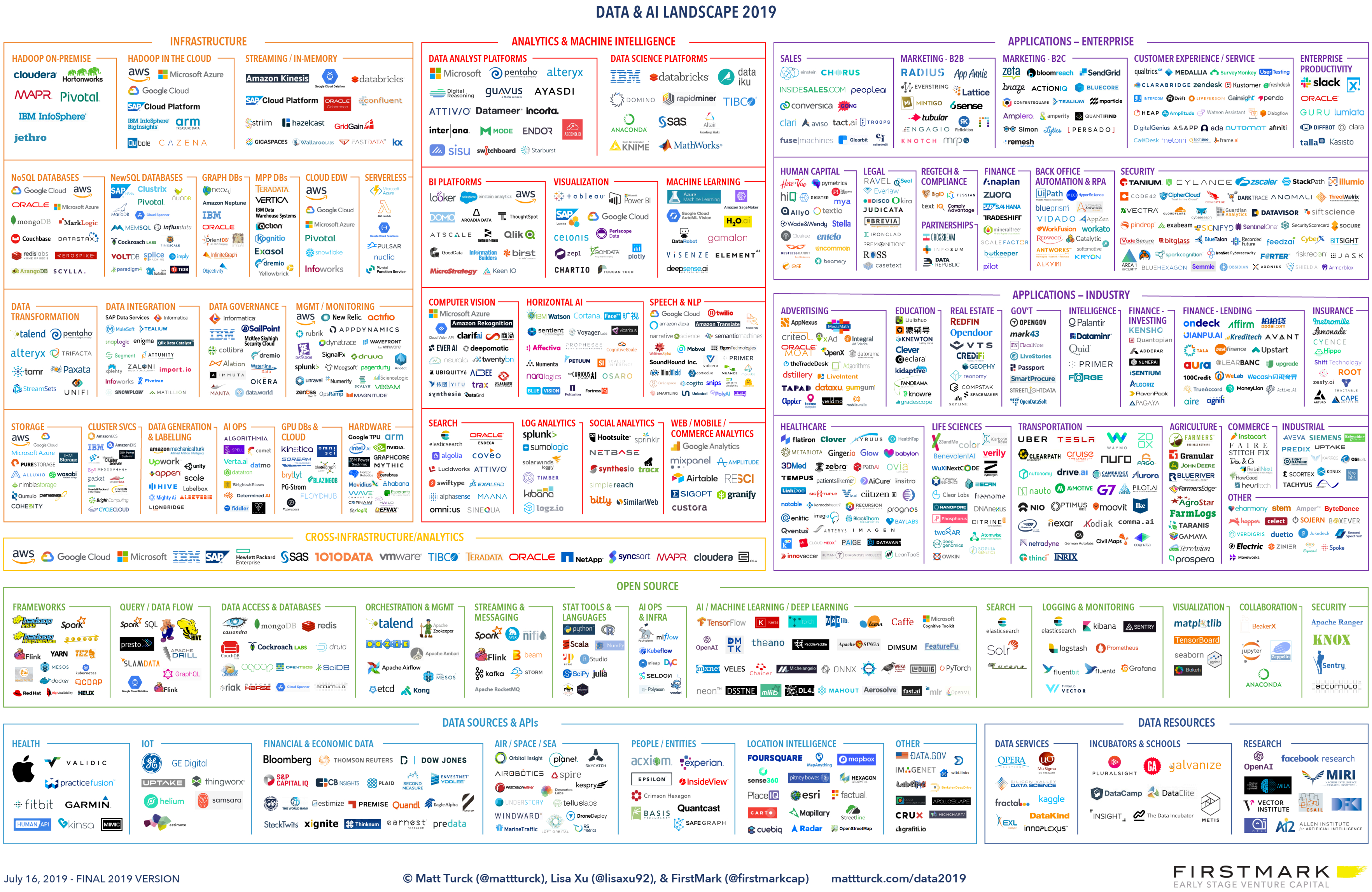
Riccardo Torlone

Università Roma Tre

ROMA TRE
UNIVERSITÀ DEGLI STUDI

Credits: Claudio Martella

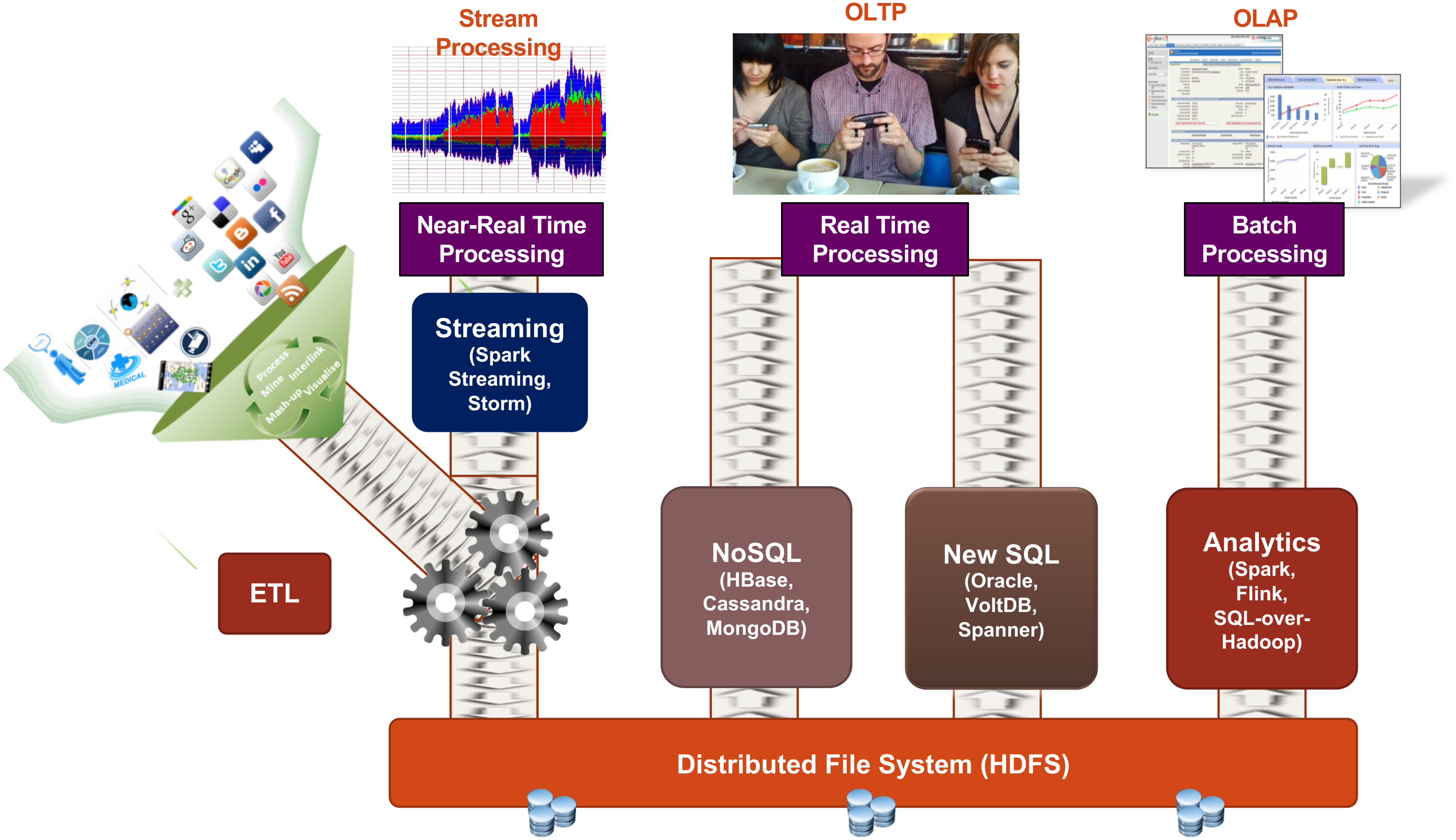
Tools for big data processing



Hundreds of solutions

- A possible classification:
  - Based on the features provided in the global architecture
  - Based on the approach to big data processing

A global view of Big Data processing



Stream Processing

OLTP

OLAP

Near-Real Time Processing

Real Time Processing

Batch Processing

Streaming (Spark Streaming, Storm)

ETL

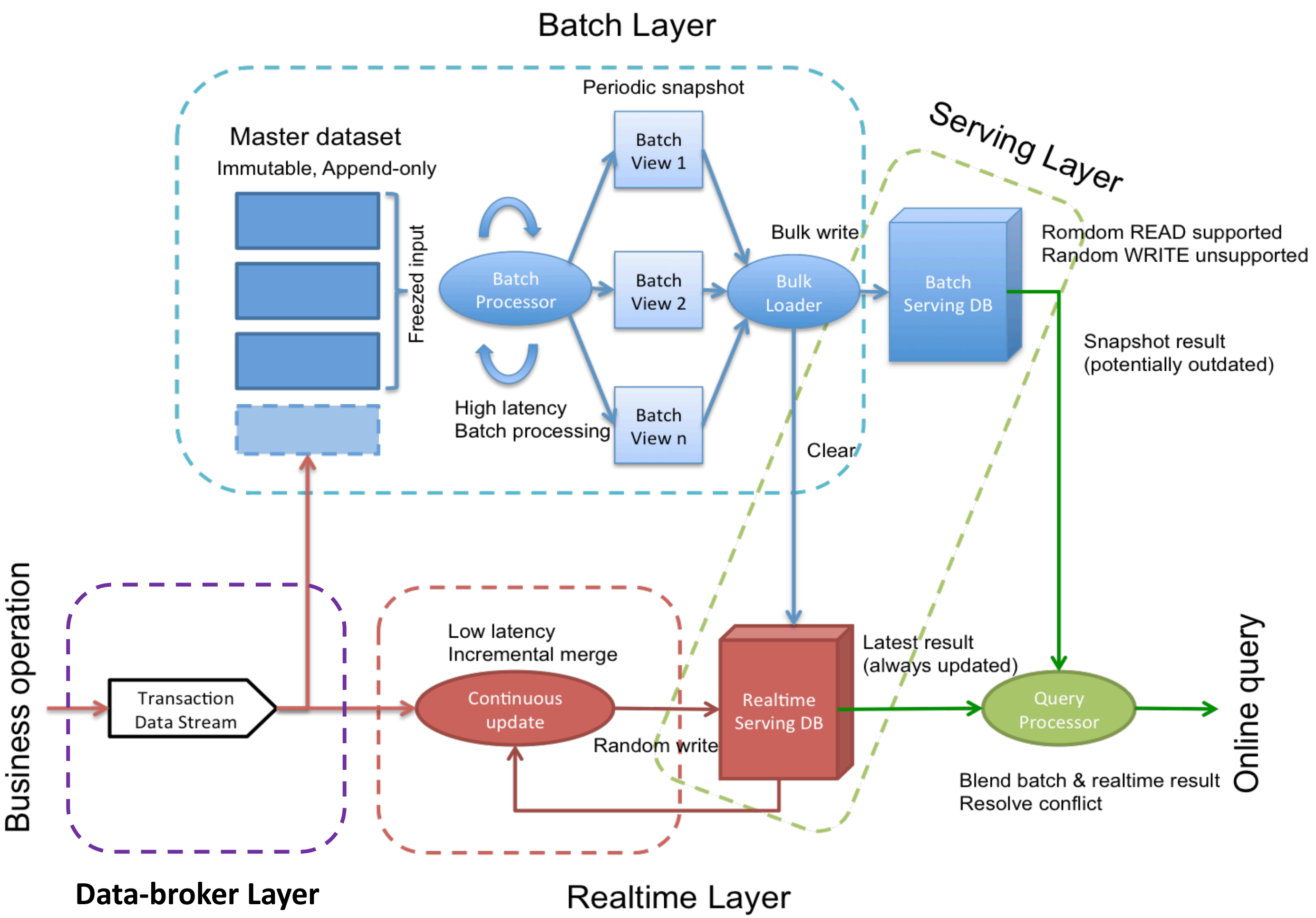
NoSQL (HBase, Cassandra, MongoDB)

New SQL (Oracle, VoltDB, Spanner)

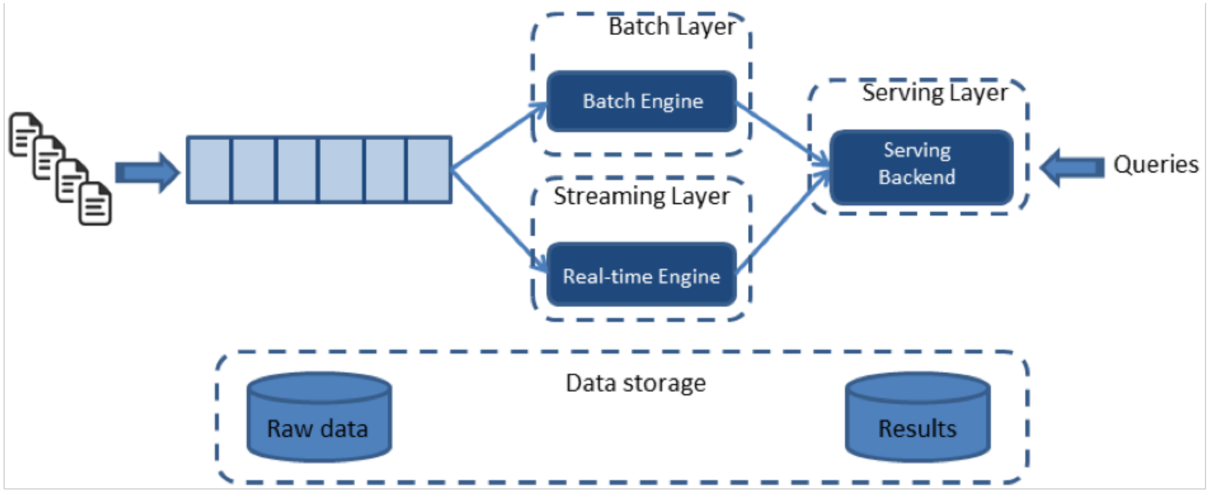
Analytics (Spark, Flink, SQL-over-Hadoop)

Distributed File System (HDFS)

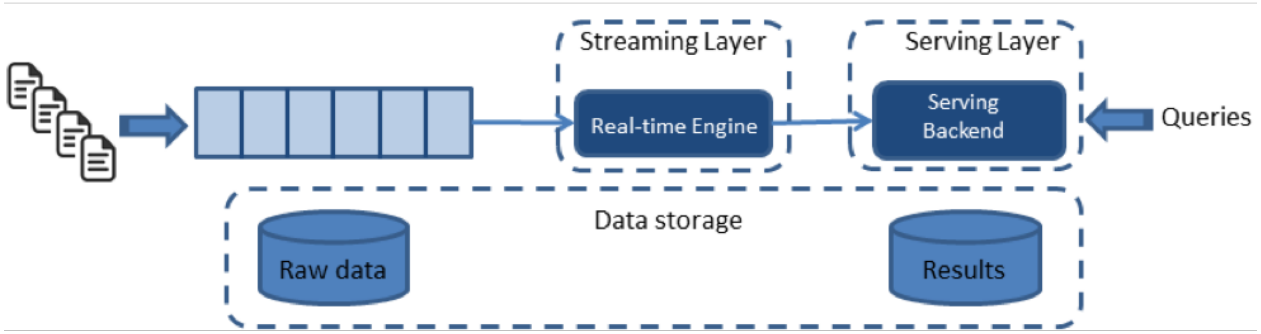
The lambda architecture for analytics



Lambda vs kappa architecture



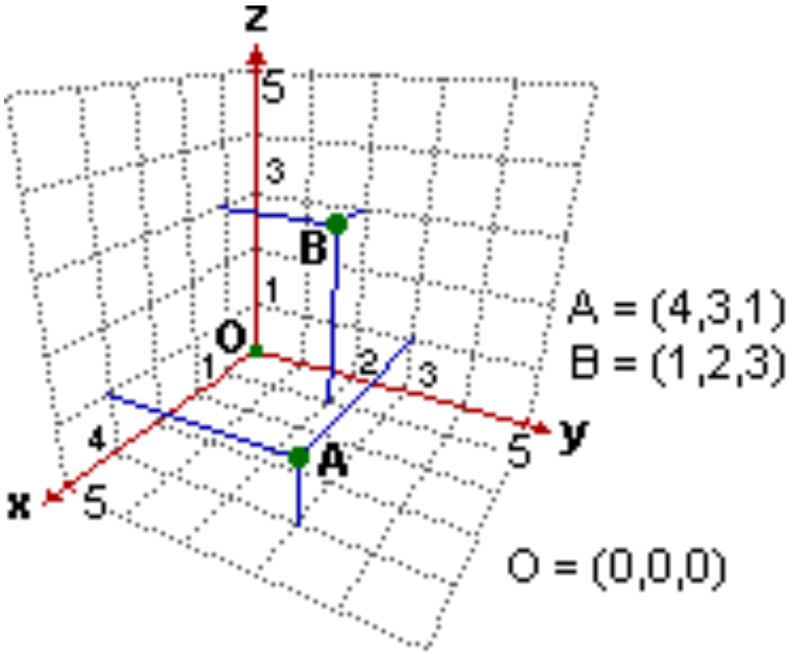
Architettura lambda



Architettura kappa

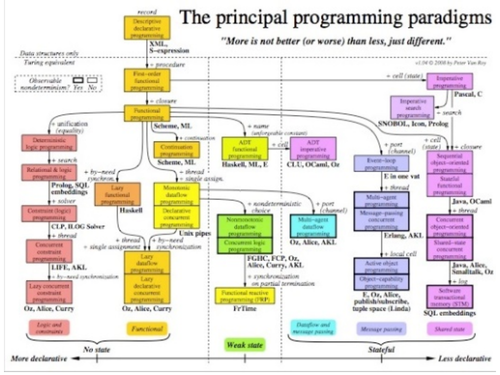
Orthogonal approaches to BD Processing

- Programming Model
  - DAG
  - Graph
  - BSP
  - SQL on Hadoop
  - NoSQL/NewSQL
- Efficiency
  - In-memory processing
  - Columnar storage
  - Multi-level execution trees
- Latency
  - Batch
  - Stream
  - OLTP



Alternative programming models

- DAG
  - Spark
  - Tez
  - Dremel
  - Storm
- BSP
  - MapReduce
  - Pregel
  - Giraph
  - Hama
- Graph
  - Giraph
  - GraphLab
  - GraphX
  - GDBMS


The principal programming paradigms
"More is not better (or worse) than less, just different."
More declarative
No state
Weak state
Stateful
Less declarative

- SQL on Hadoop
  - Hive
  - Spark SQL
  - Drill
  - Impala
  - Presto
  - Spanner
  - Tajo
- NoSQL DBMS
  - Key-Value
  - Document store
  - Column family
- NewSQL DBMS
  - Google Spanner
  - VoltDB
  - ClusterixDB

Improving the performance

- In-memory processing
  - Spark
  - Flink
  - M3R
  - Terracotta/BigMemory
  - In-memory DBMS
    - Kognitio
    - Hana
    - VoltDB
    - Redis
    - …

- Columnar storage
  - Dremel
  - Impala
  - Parquet
  - Druid
- Multi-level execution trees
  - Tez
  - Dremel
  - Impala

Supporting low latency

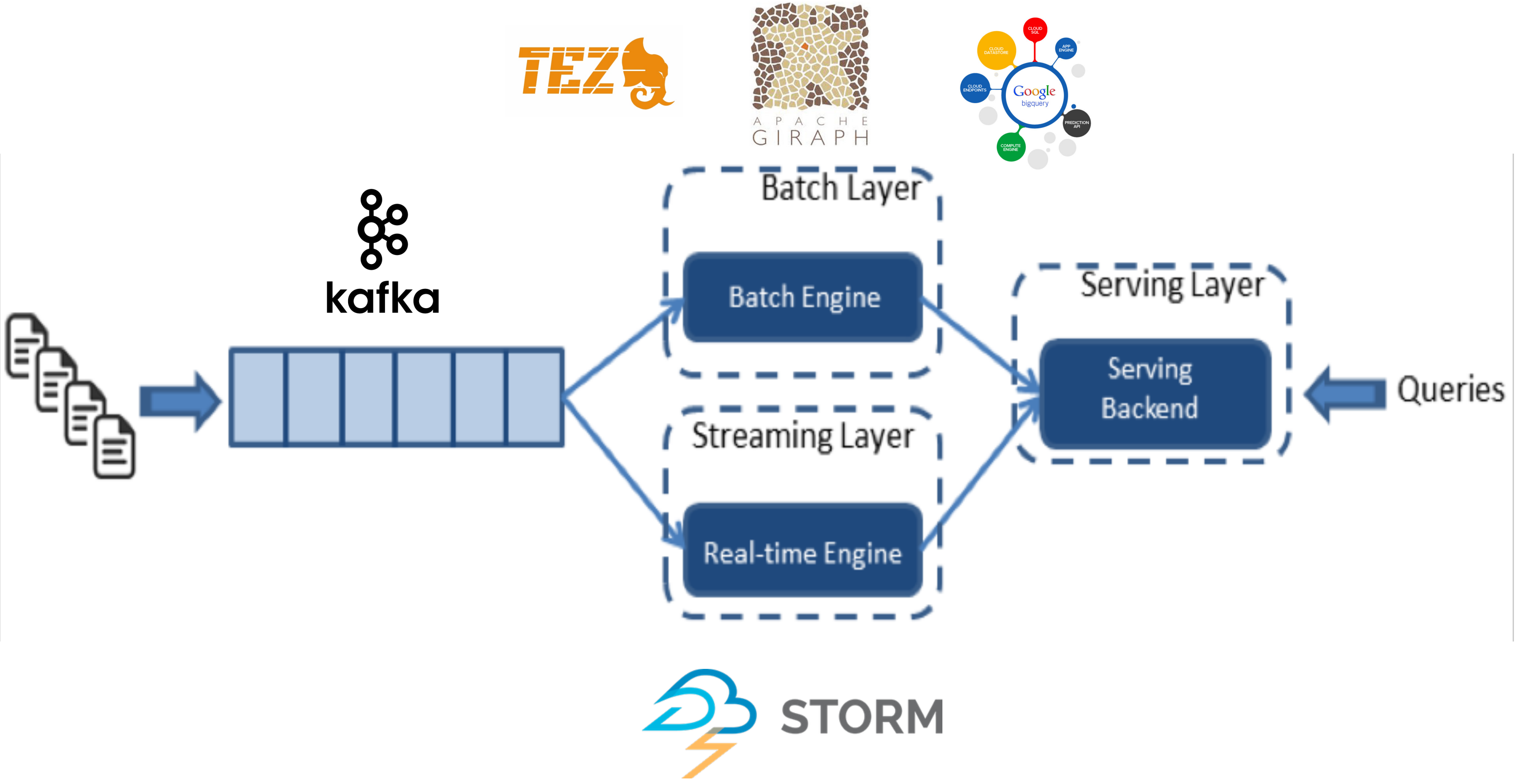
- Stream processing (near-real time)
  - Flink
  - Storm
  - Spark Streaming
  - S4
  - Samza
  - Dremel
  - Hyracks

- OLTP (real time)
  - NoSQL DBMSs
  - NewSQL DBMSs

What else?

- Data Ingestion (collecting, aggregating, and moving big data)
    - Kafka, Sqoop, Flume, …
- Scheduling and coordination (Hadoop workflow management and coordination)
    - Zookeeper, Oozie, Thrift, …
- System Deployment (Cluster management)
    - Ambari, Mesos, Helix, …
- Data cleaning
    - OpenRefine, DataCleaner , …
- Data visualization
    - Tableau, D3.js, Kibana, …
- …

An overview of some solutions for analytics



An overview of some solutions

- Kafka
  - Data Ingestion
  - collecting, aggregating, and moving big data
- Giraph
  - Graph data model
  - BSP processing model
- Storm
  - Stream processing
  - DAG processing model
- Tez
  - DAG processing model
  - SQL via Hive
- Dremel
  - Columnar storage
  - Multi-level execution trees
  - SQL via BigQuery

kafka

What is Kafka?

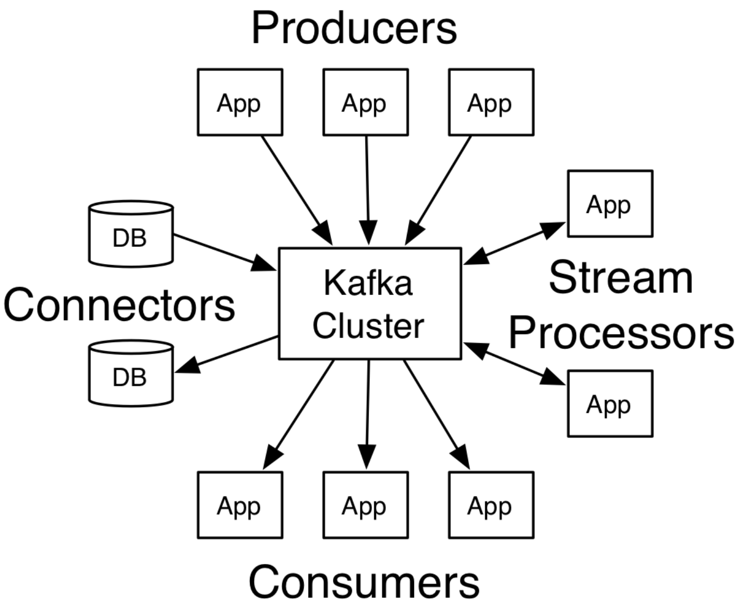
- Kafka is a distributed **publish-subscribe messaging system**
- It's designed to be
  - Fast
  - Scalable
  - Durable
- The whole job of Kafka is to provide an "**absorber**" between the flood of events and those who want to consume them in their own way

Capabilities and applications

- Kafka has three key capabilities:
  - **Publish** and **subscribe** streams of records.
  - **Store** streams of records in a fault-tolerant durable way.
  - **Process** streams of records as they occur.
- Kafka is generally used for two broad classes of applications:
  - Building real-time streaming data pipelines that reliably get data between systems or applications
  - Building real-time streaming applications that transform or react to the streams of data

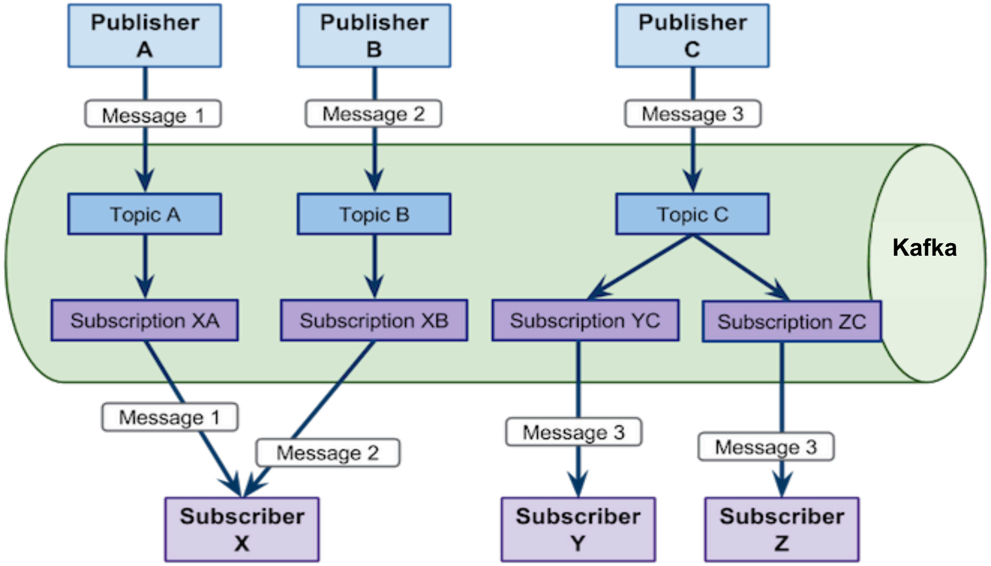
Actors in Kafka

- Kafka has four core APIs:
  - The **Producer** API allows an application to publish a stream of records to one or more Kafka topics.
  - The **Consumer** API allows an application to subscribe to one or more topics and process the stream of records produced to them.
  - The Streams API allows an application (**stream processor**) to consume an input stream from one or more topics and produce an output stream to one or more output topics, effectively transforming the input streams to output streams.
  - The **Connector** API allows the connection of Kafka topics to existing applications or data systems.



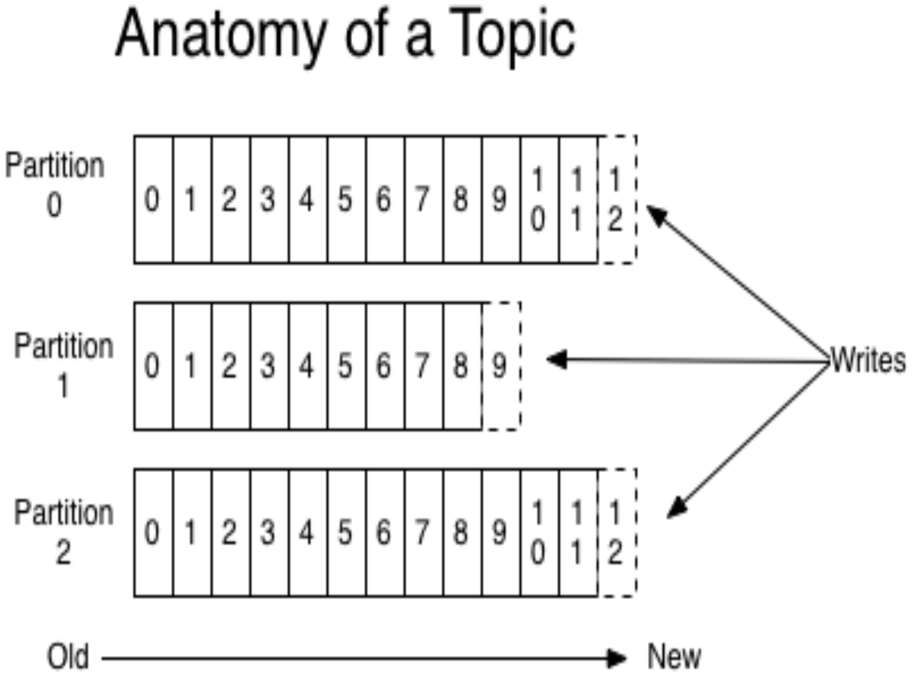
Publish/subscribe messaging system

- Kafka maintains feeds of messages in categories called **topics**
- **Producers** publish messages (records) to one or more topics
- **Consumers** subscribe to topics and process the feed of published messages
- A topic can have zero, one, or many consumers that subscribe to the data written to it.



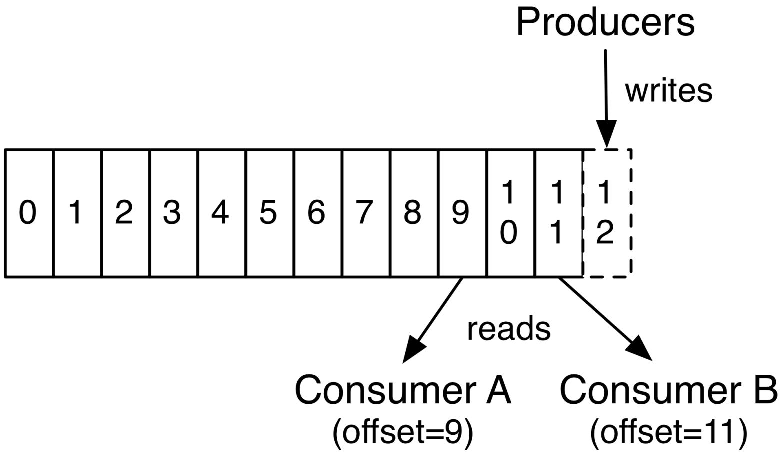
Anatomy of a topic

- For each topic, the Kafka cluster maintains a partitioned log
- Each partition is an ordered, immutable sequence of records that is continually appended
- The records in the partitions are each assigned a **sequential id** number called the **offset** that uniquely identifies each message within the partition.



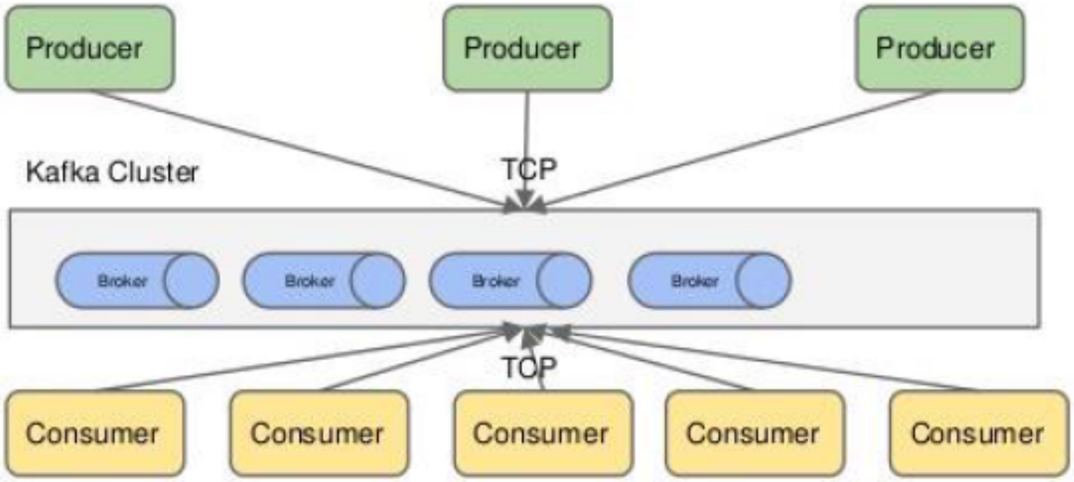
Retention

- The Kafka cluster **retains all published records**—whether or not they have been consumed—**for a configurable period of time**; after which it will be discarded to free up space.
- The **offset** of the records is **controlled by consumer**.
- Normally a consumer will advance its offset linearly as it reads records, but it can consume records in any order it likes.
- Kafka consumers can come and go without much impact on the cluster or on other consumers.

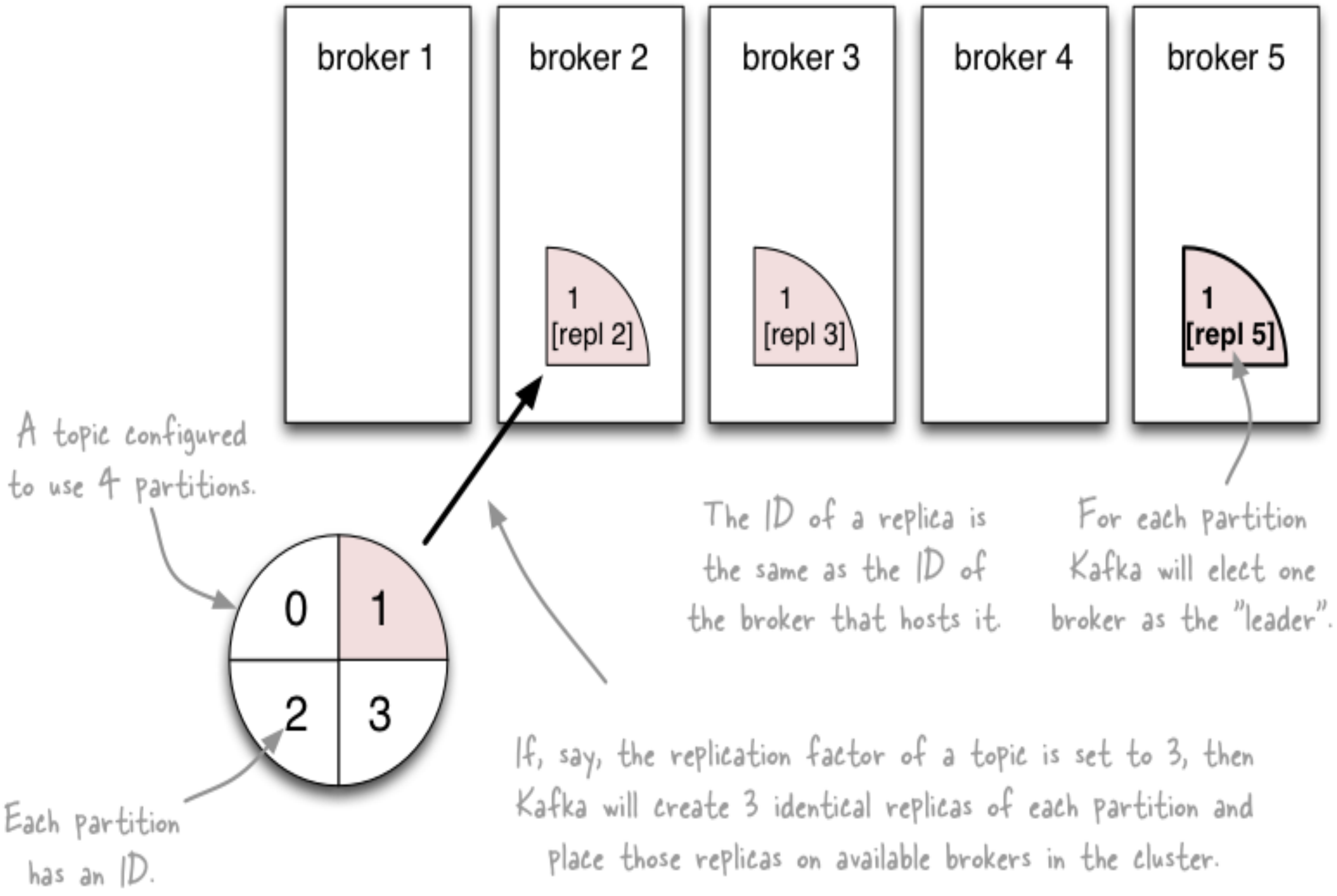


Kafka cluster

- Since Kafka is distributed in nature, Kafka is run as a cluster.
- A cluster is typically comprised **multiple servers**; each of which is called a **broker**.
- Communication between the clients and the servers takes place over TCP protocol



Distribution and partitions



A topic configured to use 4 partitions.

Each partition has an ID.

The ID of a replica is the same as the ID of the broker that hosts it.

For each partition Kafka will elect one broker as the "leader".

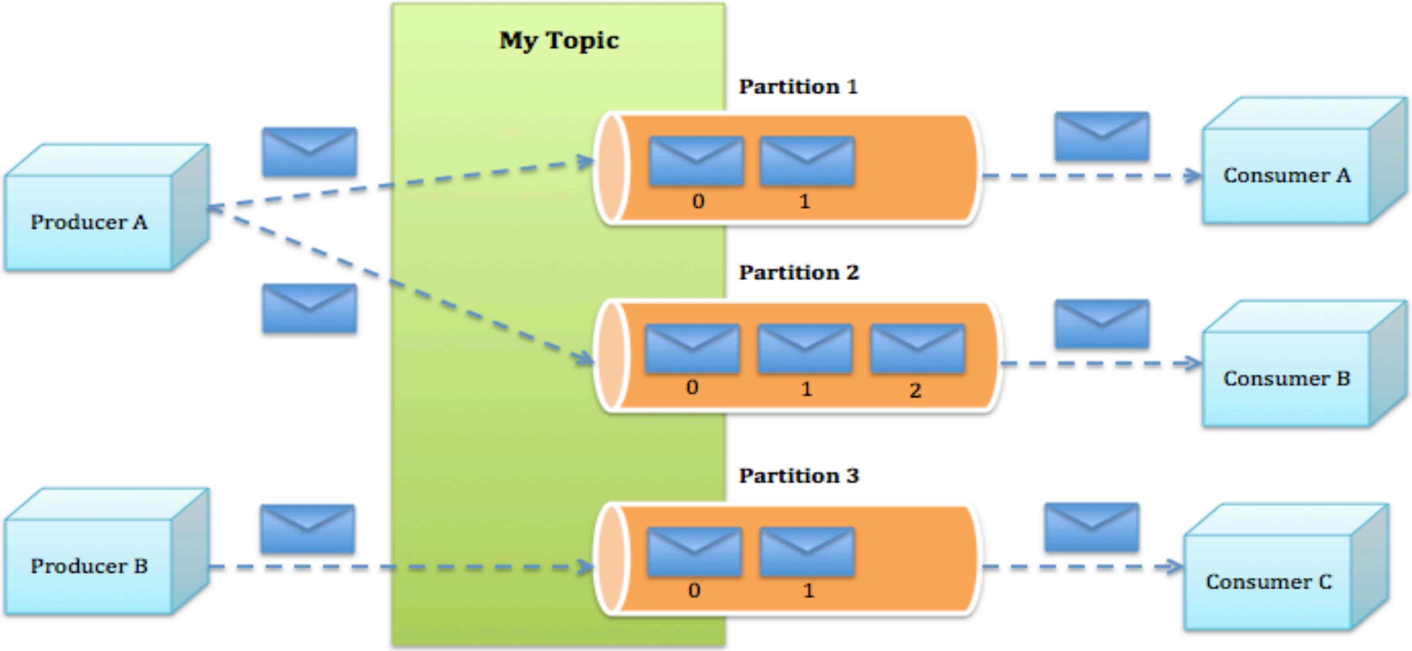
If, say, the replication factor of a topic is set to 3, then Kafka will create 3 identical replicas of each partition and place those replicas on available brokers in the cluster.

Distribution and fault tolerance

- Each partition has one server which acts as the "**leader**" and zero or more servers which act as "**followers**".
- **The leader handles all read and write requests** for the partition while the **followers passively replicate** the leader.
- If the leader fails, one of the followers will automatically become the new leader.
- **Each server acts as a leader for some partitions** and a follower for others so load is well balanced within the cluster.

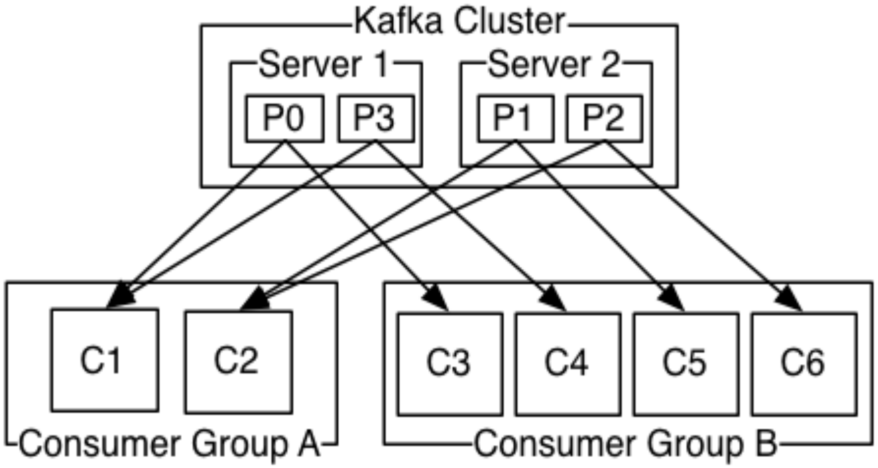
Producers

- Producers publish data to the topics by **assigning records to a partition** within the topic either in a **round-robin fashion** or according to some **semantic partition function** (say based on some key in the message).



Consumers

- Consumers can be grouped in **consumer groups**
- Each record published to a topic is delivered to one consumer within each consumer group.
- If **all the consumers** are in the **same consumer group**, then this works just like a traditional **queue** balancing load over the consumers.
- If **all the consumers** have **different consumer groups**, then this works like **publish-subscribe** and all messages are broadcast to all consumers.



Performance benchmark

- 500,000 messages published per second
- 22,000 messages consumed per second
- on a 2-node cluster
- with 6-disk RAID 10.

Key benefits

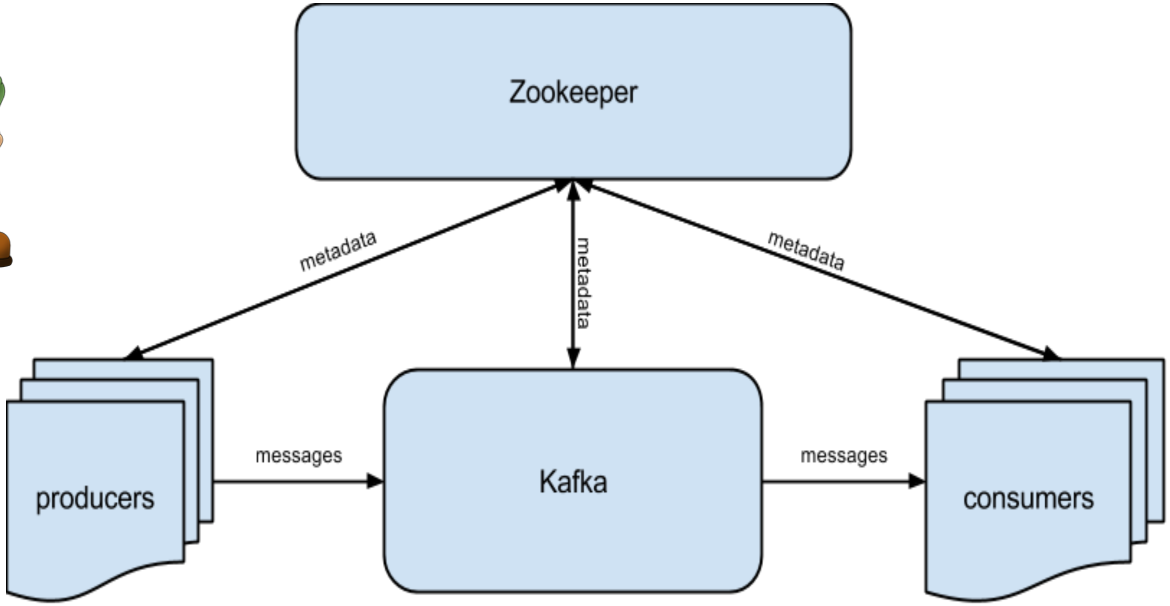
- **Horizontally scalable**
  - It’s a distributed system can be elastically and **transparently expanded** with **no downtime**
- **High throughput**
  - High throughput is provided for both publishing and subscribing even with many terabytes of stored messages
- **Reliable delivery**
  - Persists messages on disk, and provides intra-cluster replication
  - Supports large number of subscribers and automatically balances consumers in case of failure.

Uses of Kafka

- Kafka as a Messaging System
  - Messaging traditionally has two models: queuing and publish-subscribe. The consumer group concept in Kafka generalizes these two concepts.
- Kafka as a Storage System
  - Data written to Kafka is written to disk and replicated for fault-tolerance, decoupling the publishing phase from the consuming phase. This makes Kafka very good storage system.
- Kafka for Stream Processing
  - In Kafka a stream processor is anything that takes continual streams of data from input topics, performs some processing on this input, and produces continual streams of data to output topics.

Kafka uses ZooKeeper

- Kafka uses ZooKeeper, a centralized service used to maintain naming and configuration data in a distributed system and to provide flexible and robust synchronization.
- Zookeeper keeps track of status of the Kafka cluster nodes and keeps track of Kafka topics, partitions etc.



Usage

- Start the Kafka server:

```
➢ bin/kafka-server-start.sh config/server.properties
➢ bin/run-kafka.sh
```

- Create a topic named test:

```
➢ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 13 --topic test
➢ bin/create-topic.sh
```

- List topics:

```
> bin/kafka-topics.sh --list --zookeeper localhost:2181
test
```

- Publish data:

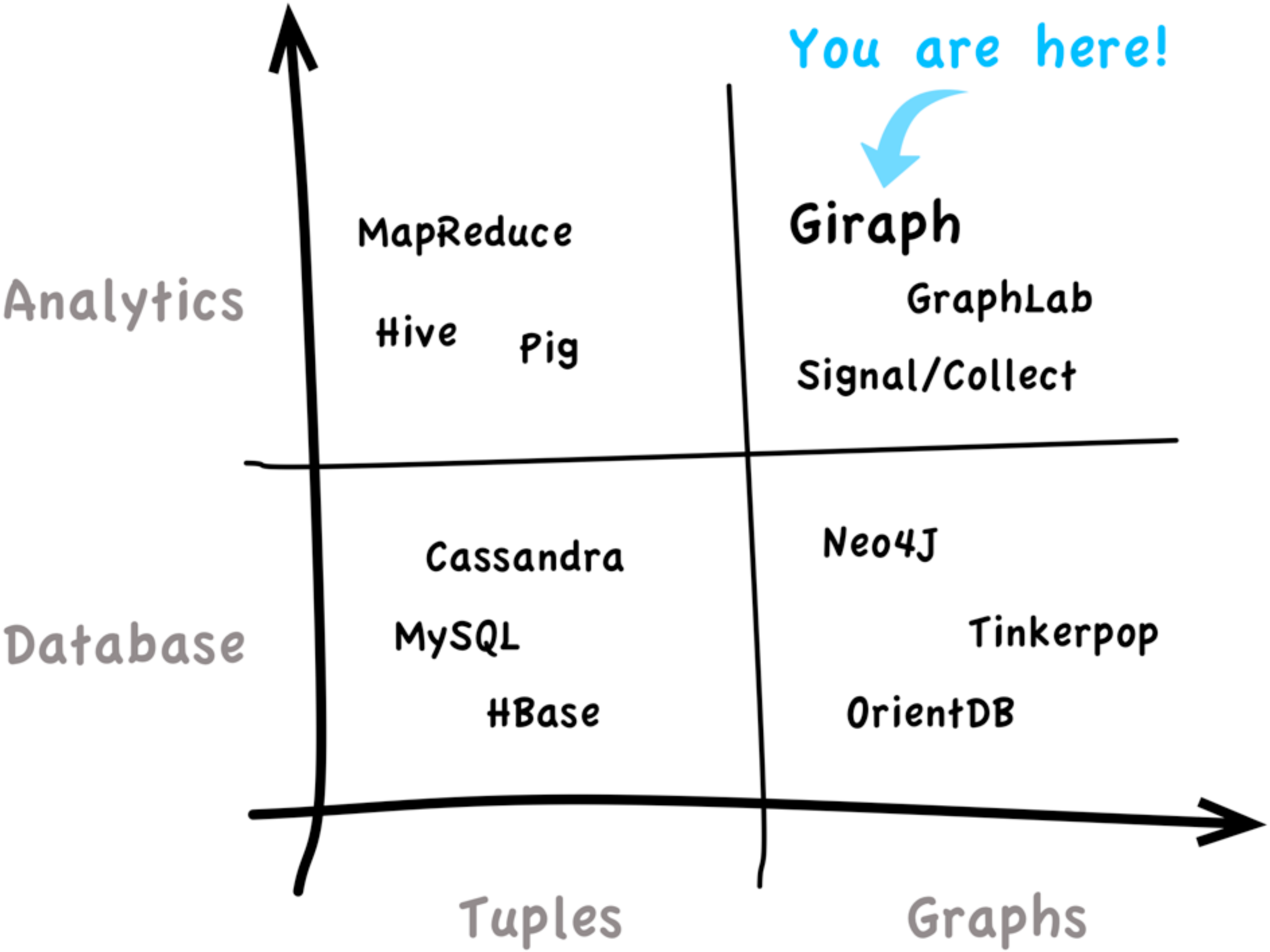
```
> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
This is a message
This is another message
```

- Consume data:

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning
This is a message
This is another message
```

- Kafka Connect is a tool included with Kafka that runs connectors, which implement the custom logic for interacting with an external system.

Giraph



You are here!

Analytics

MapReduce

Hive

Pig

Giraph

GraphLab

Signal/Collect

Database

Cassandra

MySQL

HBase

Neo4J

Tinkerpop

OrientDB

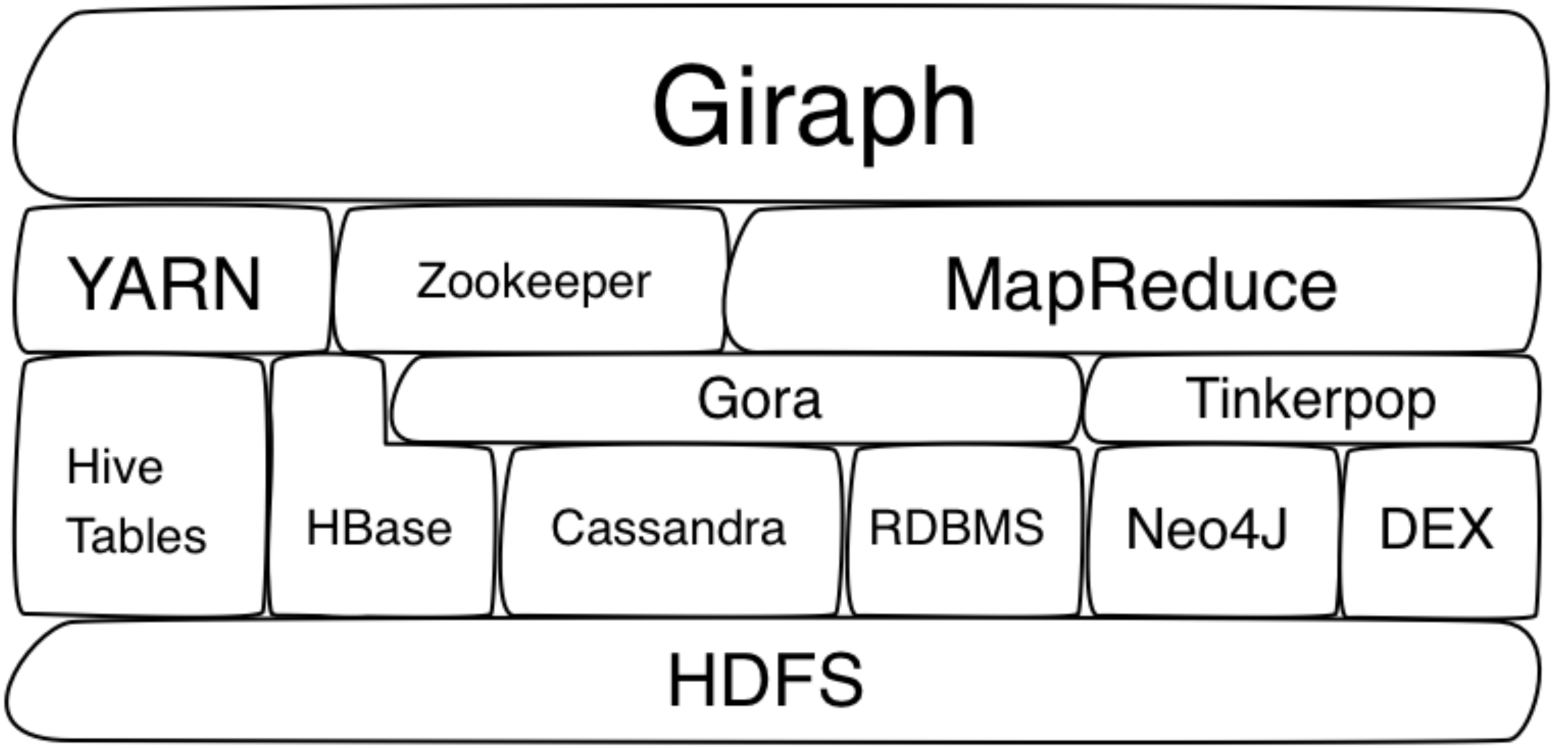
Tuples

Graphs

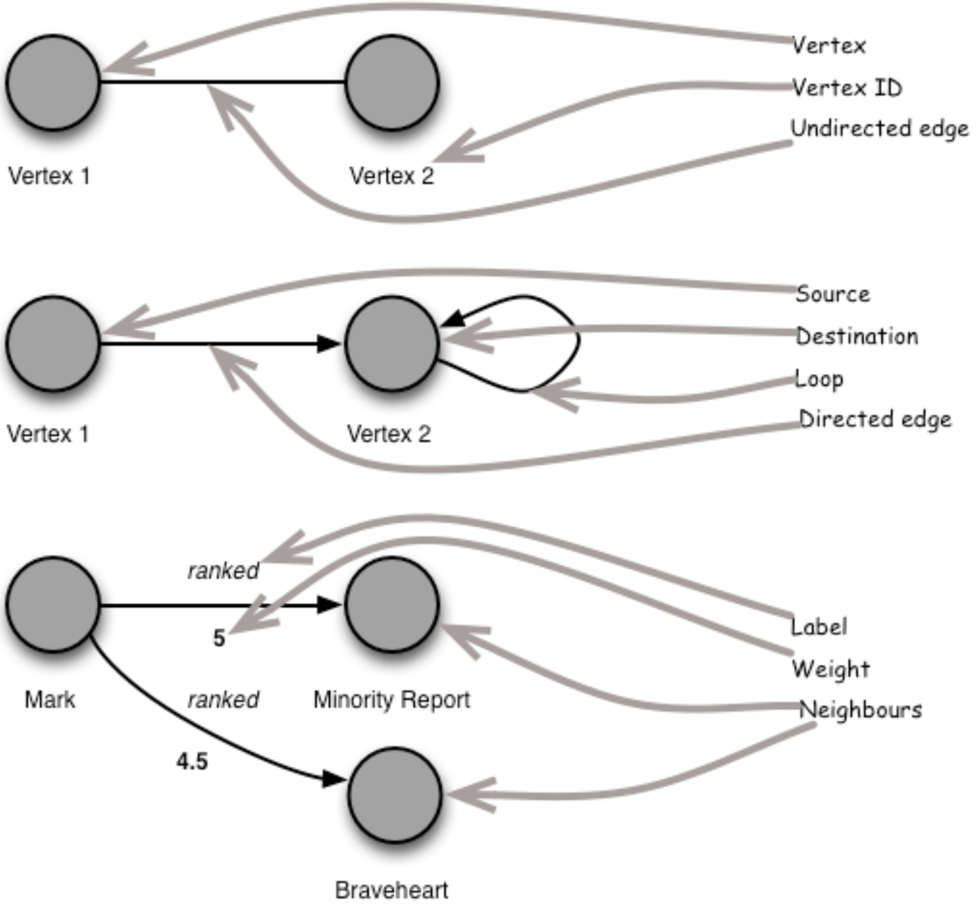
Timeline

- Inspired by Google Pregel (2010)
- Donated to ASF by Yahoo! in 2011
- Top-level project in 2012
- 1.0 release in January 2013
- 1.1 release in October 2014
- 1.2.0 release in October 2016

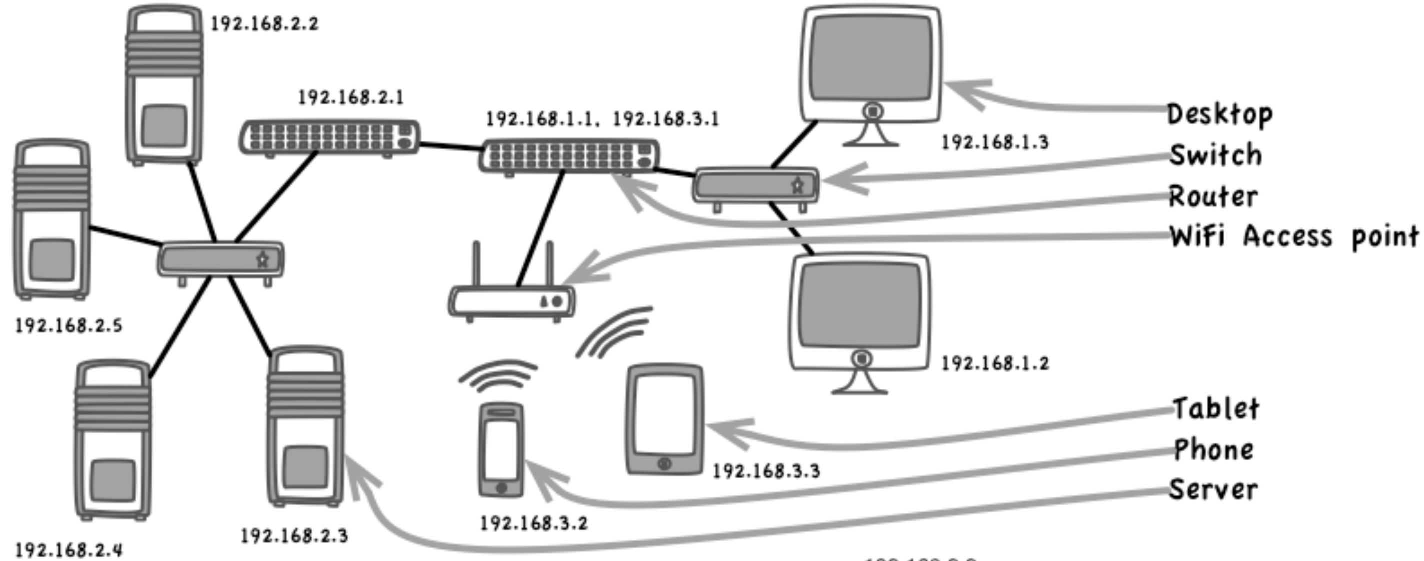
Plays well with Hadoop



Graphs are simple

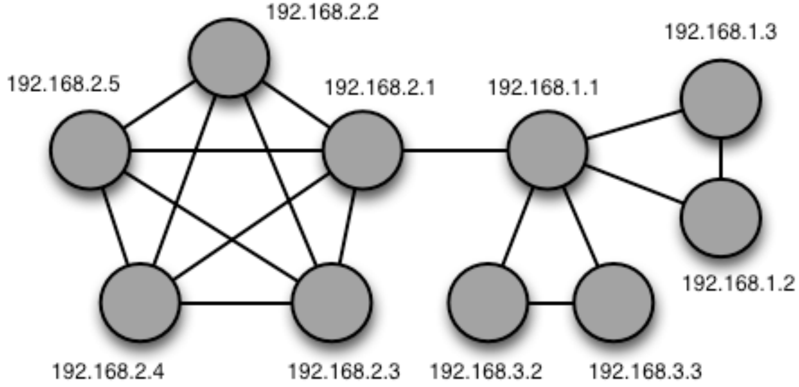


A computer network

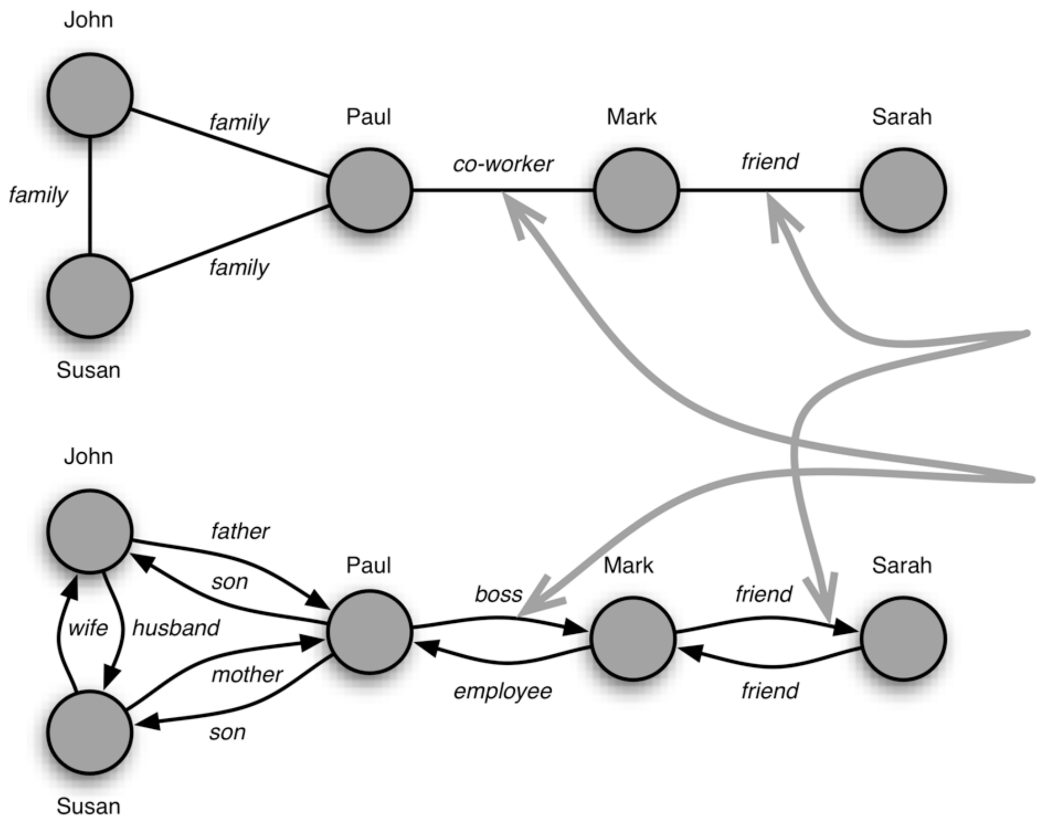


Note

1. There are three networks: servers, desktops and mobile.
2. They are connected through two routers/firewalls.
3. We ignored the switches and the access point in the graph.
4. Router 192.168.1.1 has two interfaces but we used one as vertex ID.



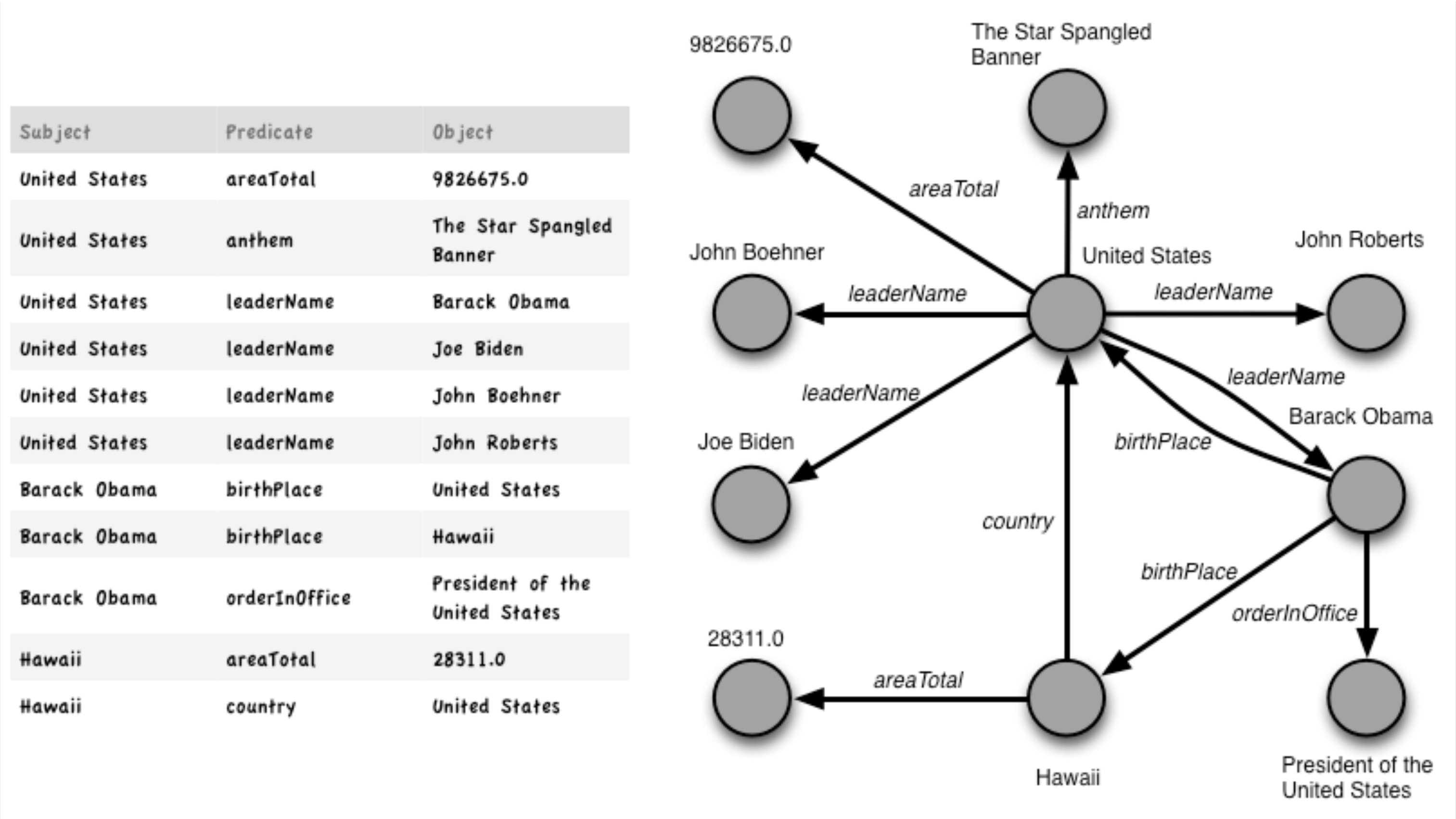
A social network



Note

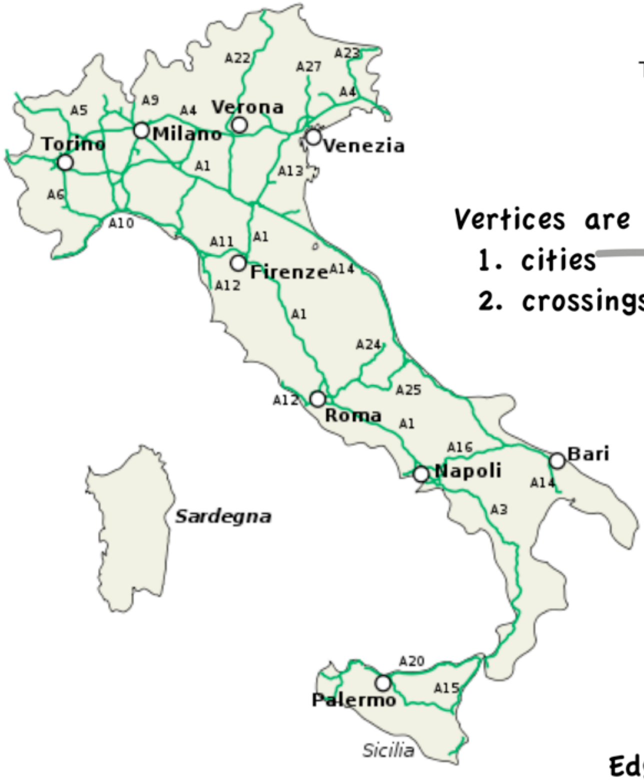
1. A symmetric relationship is substituted by two directed edges.
2. A relationship does not have to be substituted by two edges, but e.g. by a more specific one.

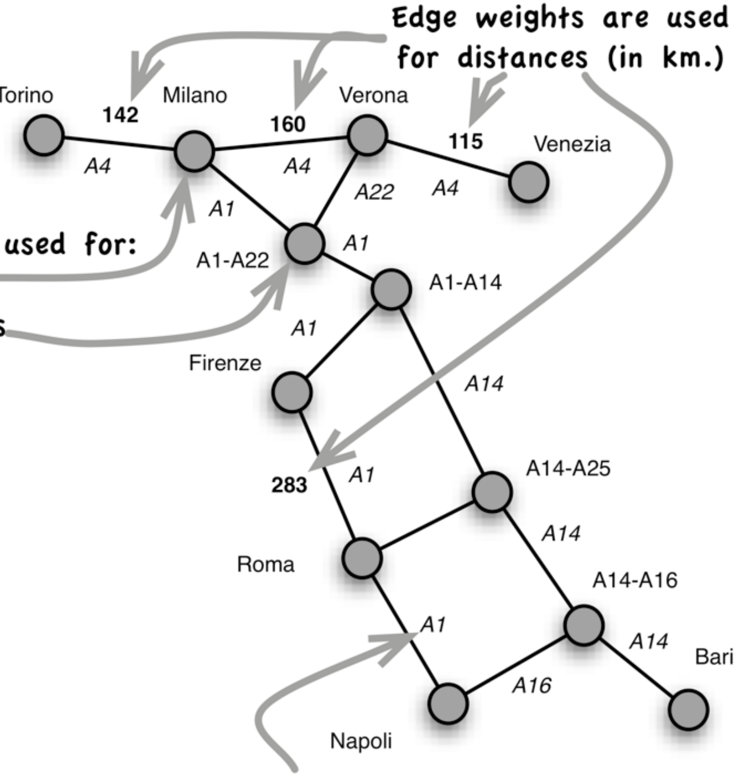
A semantic network



Subject	Predicate	Object
United States	areaTotal	9826675.0
United States	anthem	The Star Spangled Banner
United States	leaderName	Barack Obama
United States	leaderName	Joe Biden
United States	leaderName	John Boehner
United States	leaderName	John Roberts
Barack Obama	birthPlace	United States
Barack Obama	birthPlace	Hawaii
Barack Obama	orderInOffice	President of the United States
Hawaii	areaTotal	28311.0
Hawaii	country	United States

A map





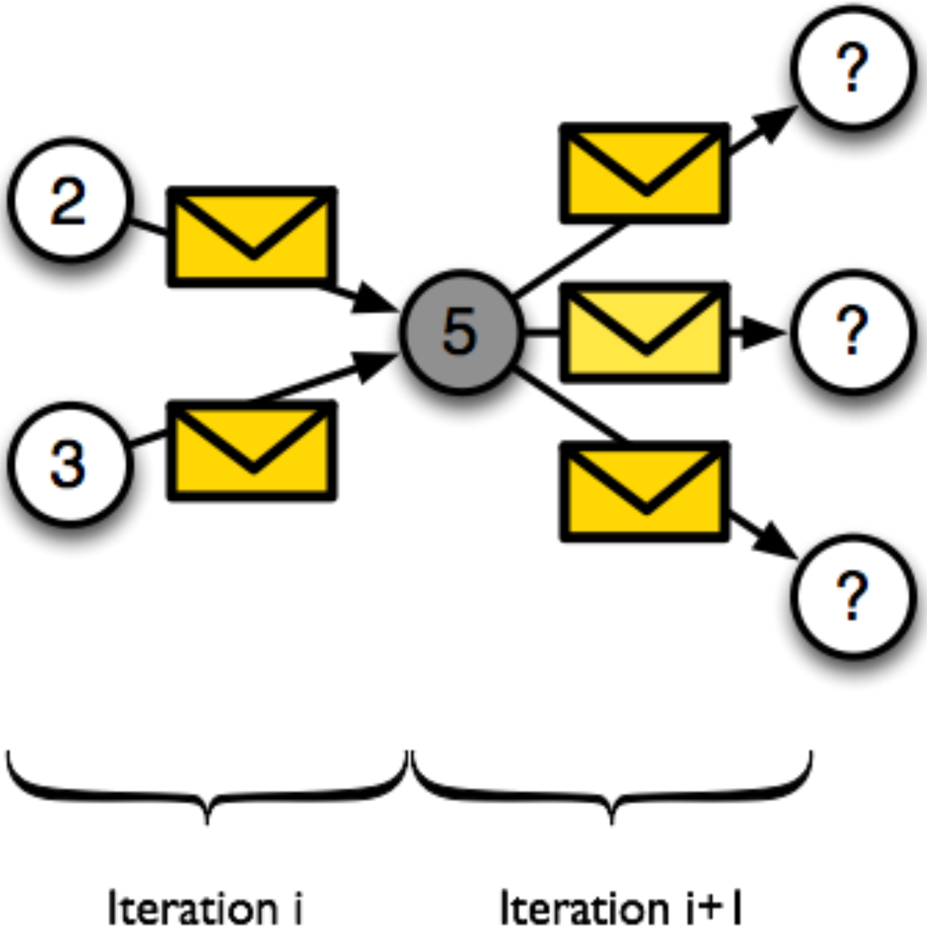
Edge weights are used for distances (in km.)

Vertices are used for:

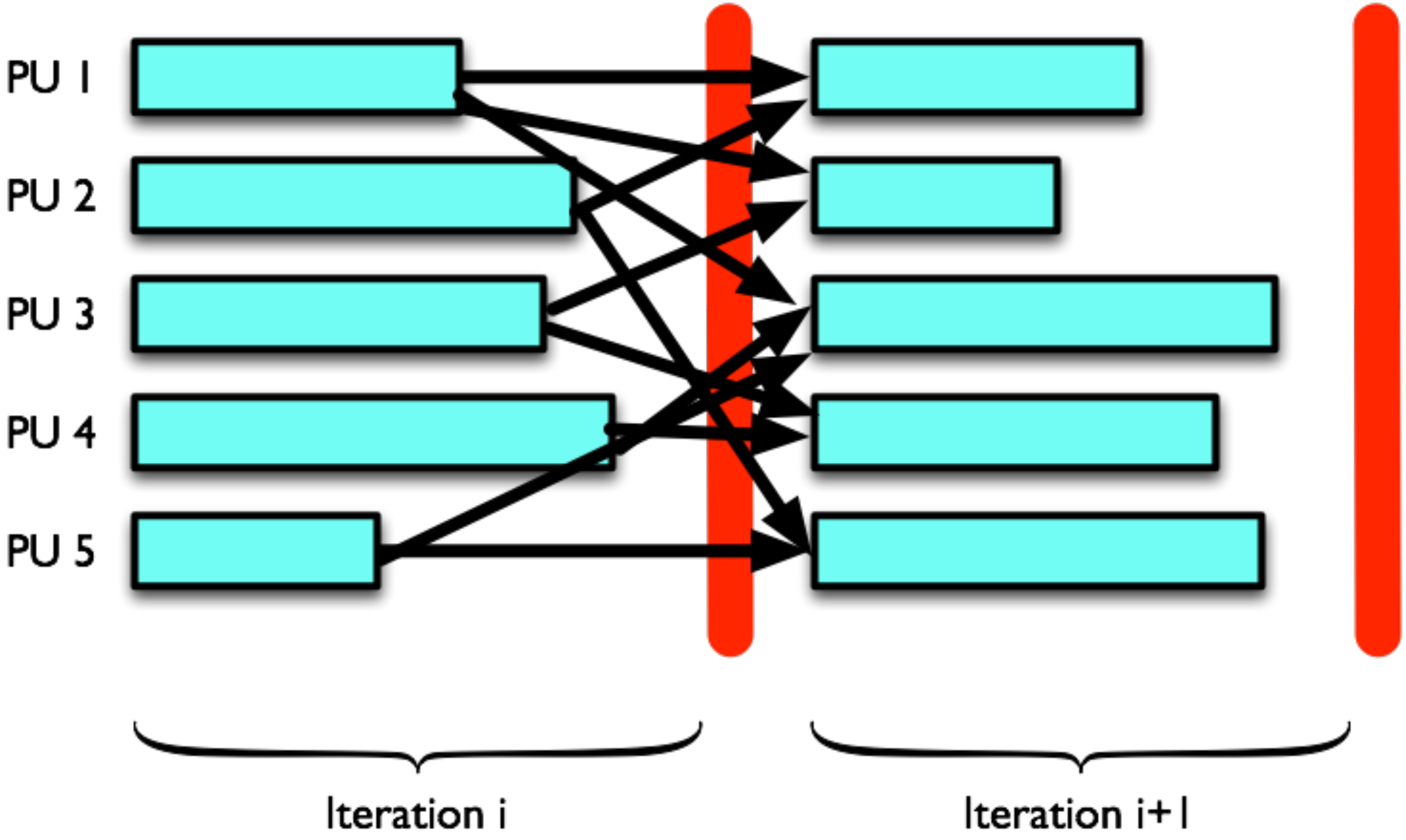
1. cities
2. crossings

Edge labels are used for the highway names

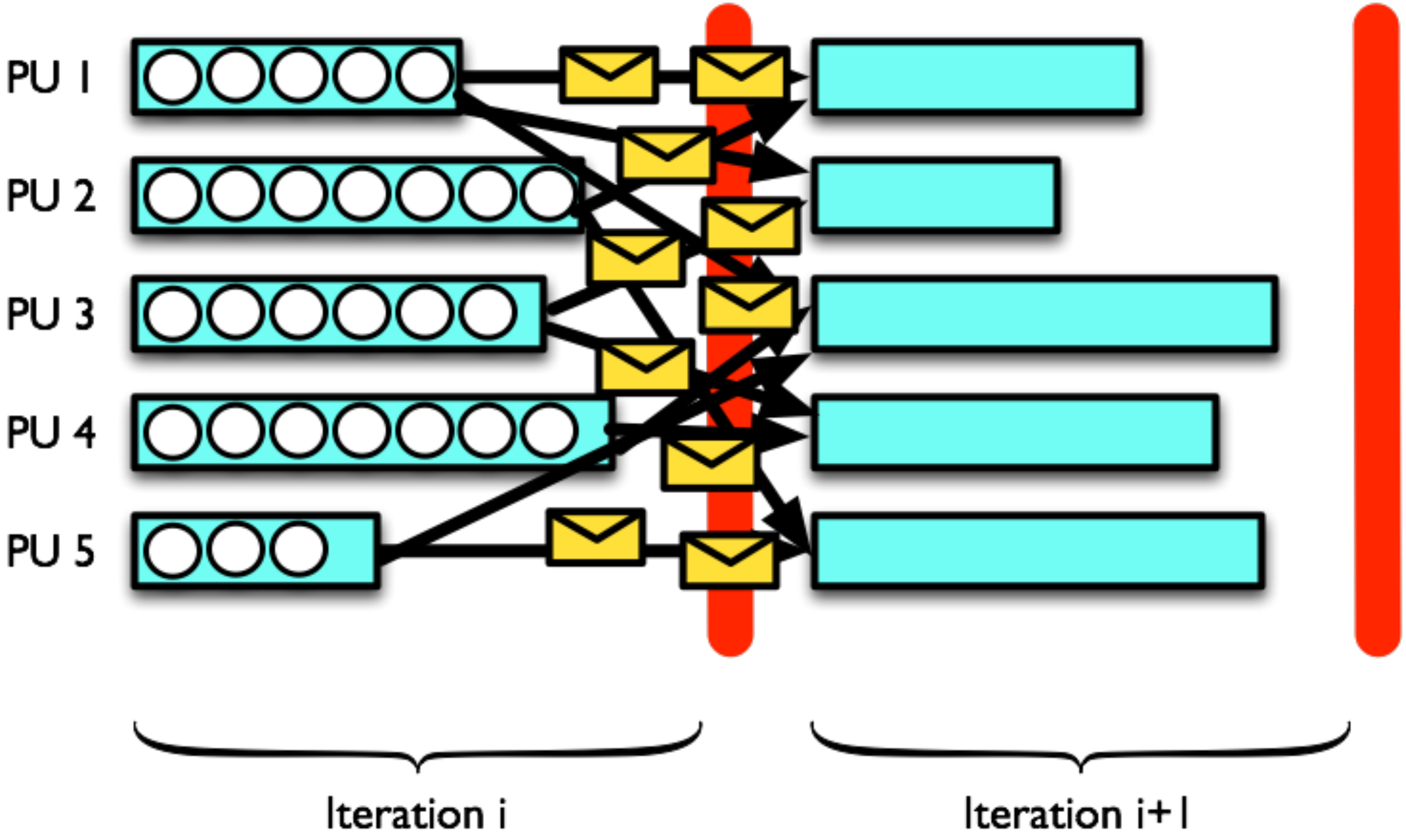
Vertex-centric API



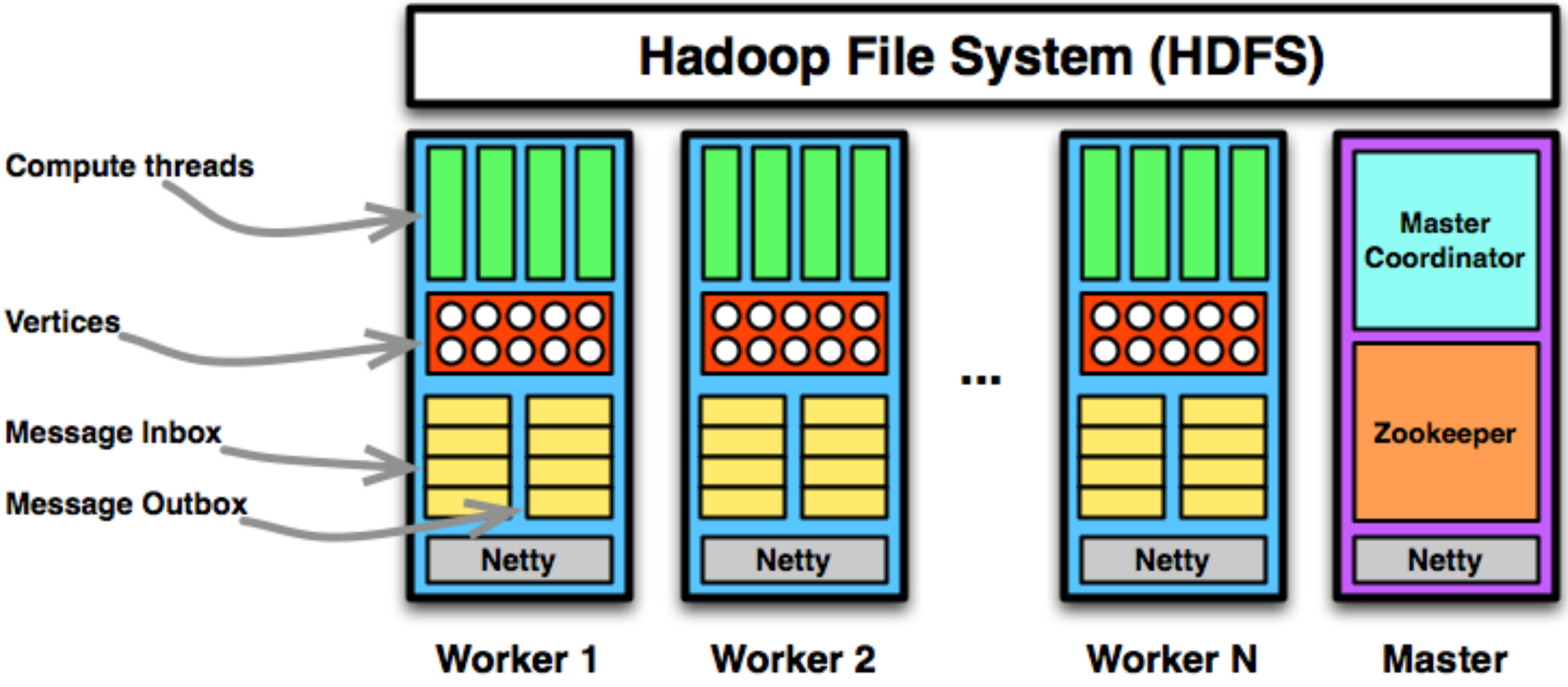
BSP machine



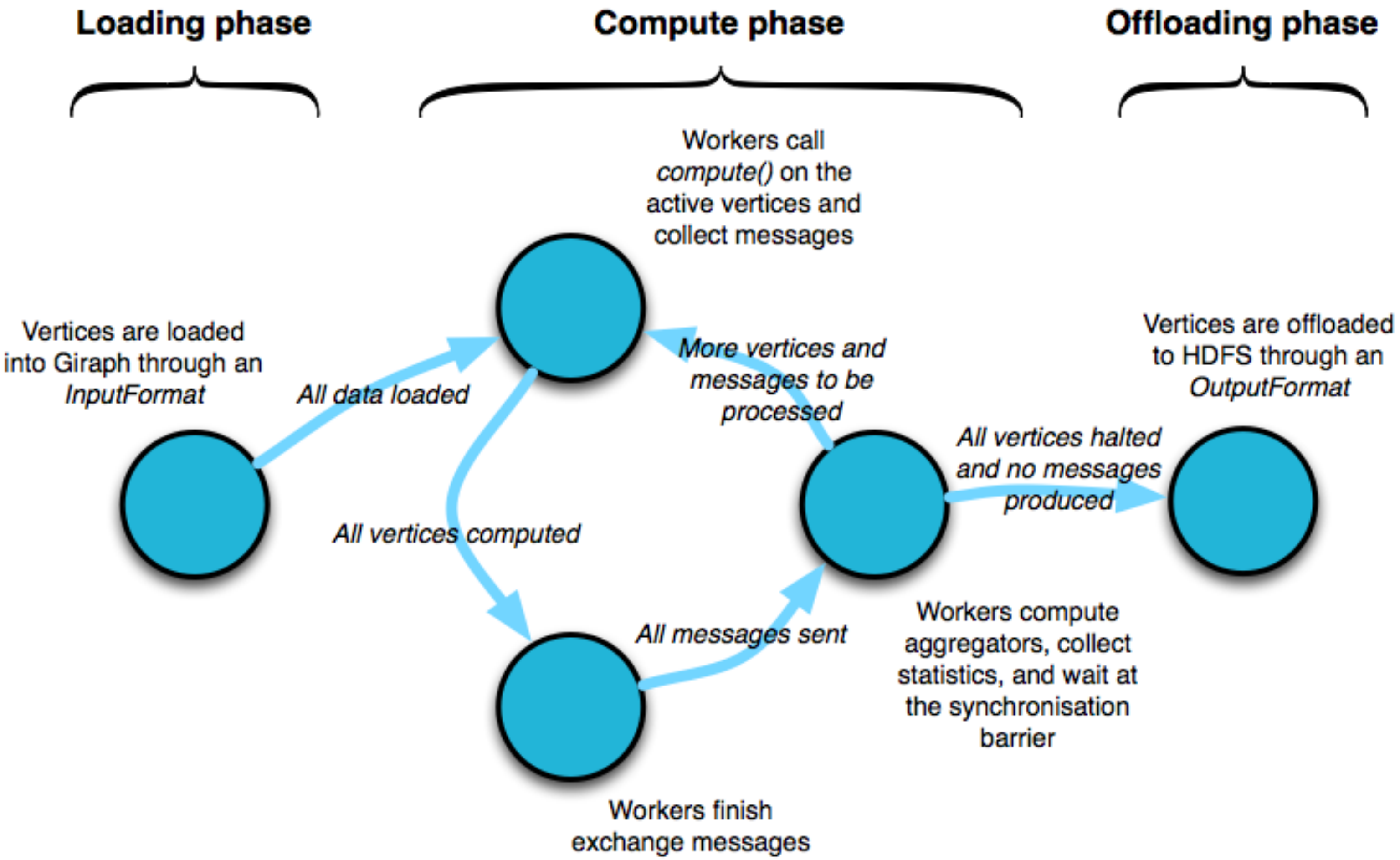
BSP & Giraph



Architecture



Giraph job lifetime



Loading phase

Compute phase

Offloading phase

Vertices are loaded into Giraph through an *InputFormat*

All data loaded

Workers call *compute()* on the active vertices and collect messages

All vertices computed

Workers finish exchange messages

All messages sent

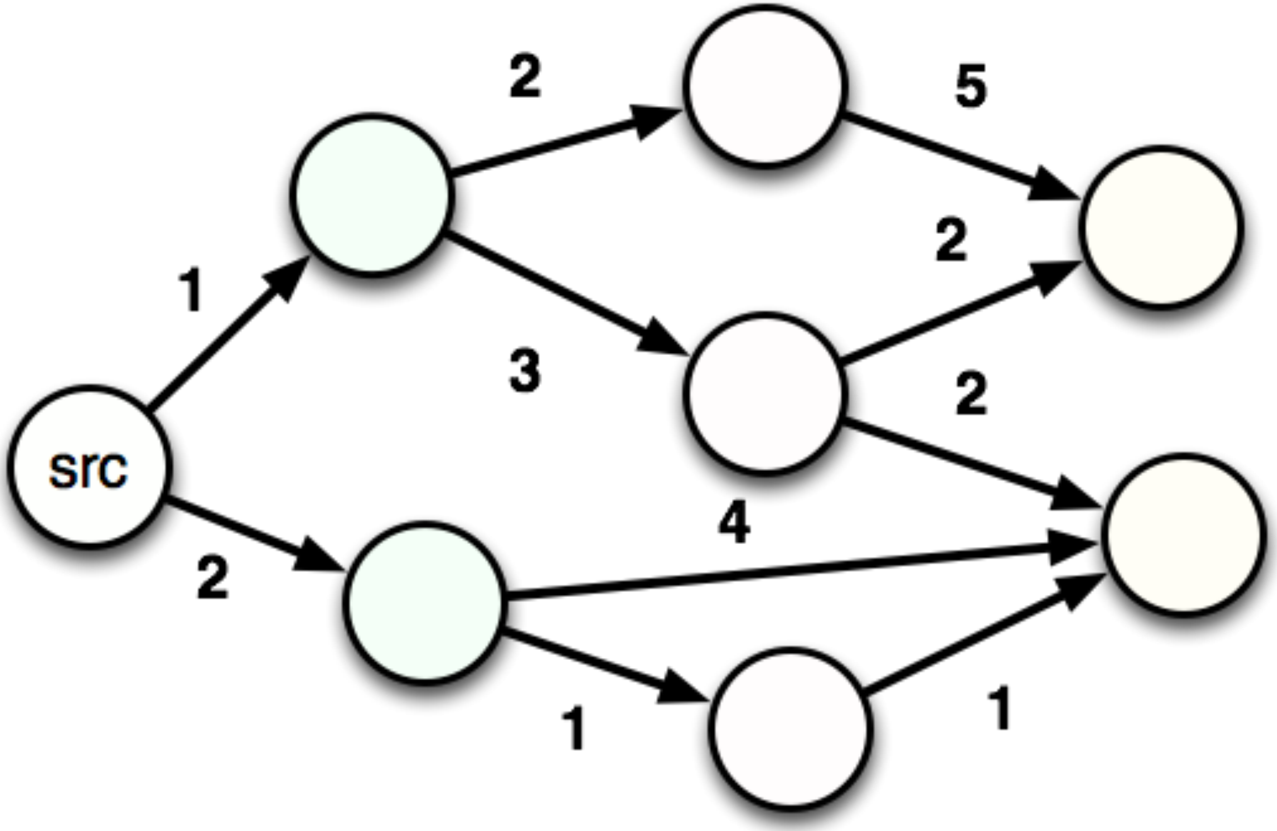
Workers compute aggregators, collect statistics, and wait at the synchronisation barrier

More vertices and messages to be processed

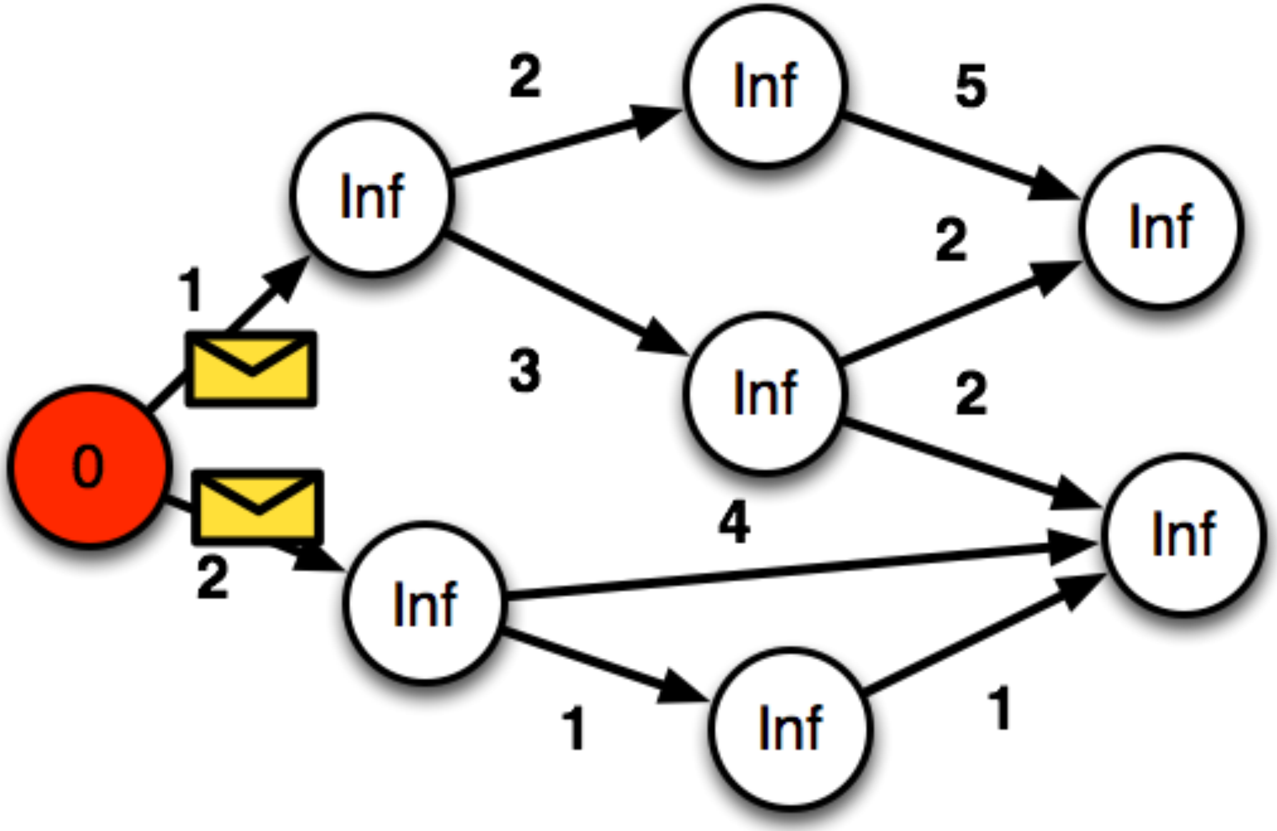
All vertices halted and no messages produced

Vertices are offloaded to HDFS through an *OutputFormat*

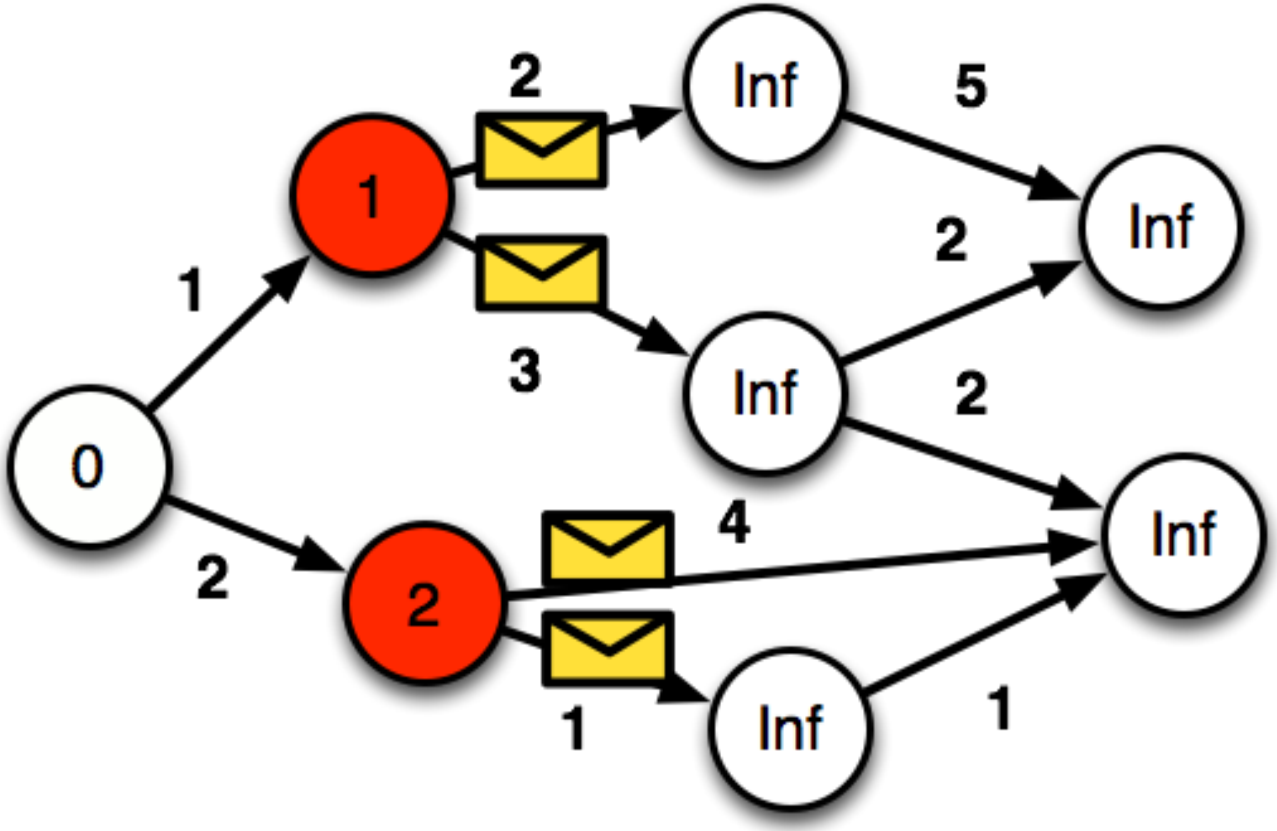
Shortest Paths



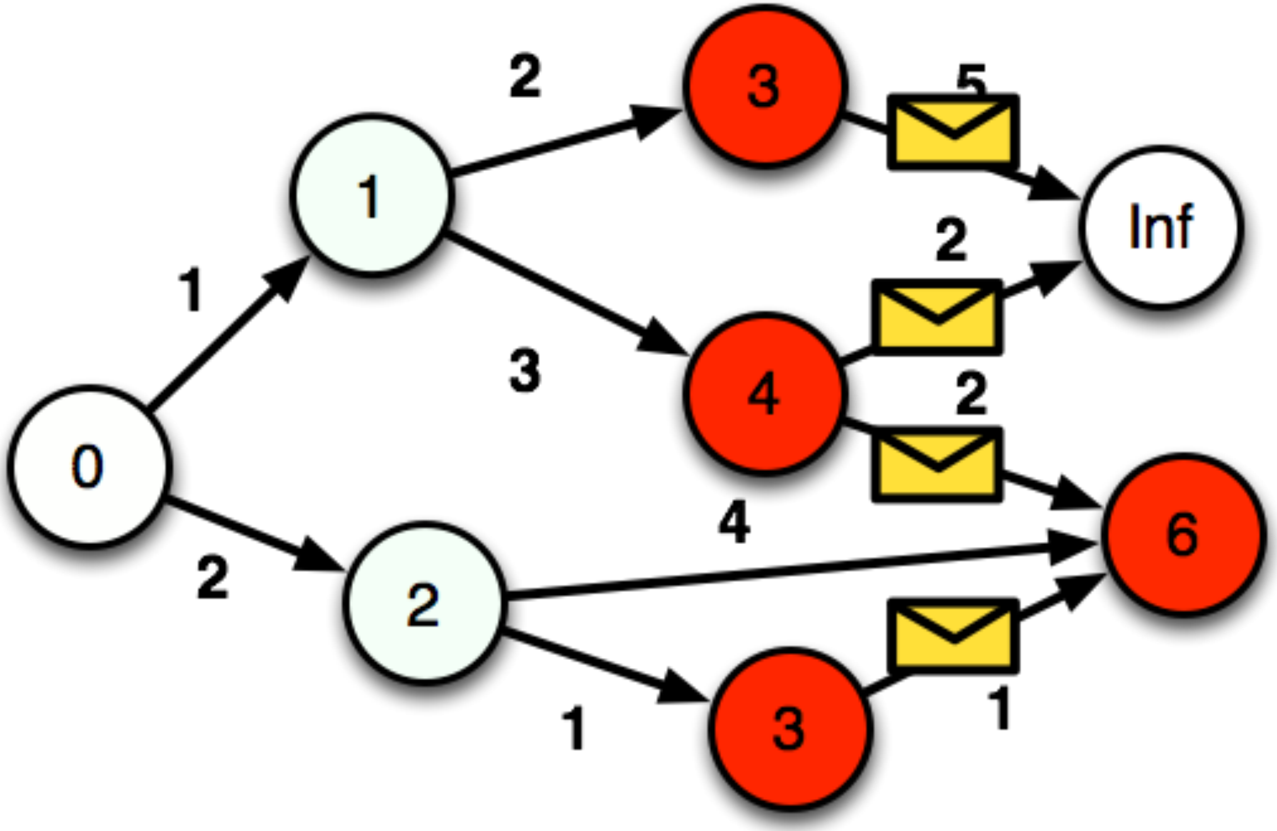
Shortest Paths



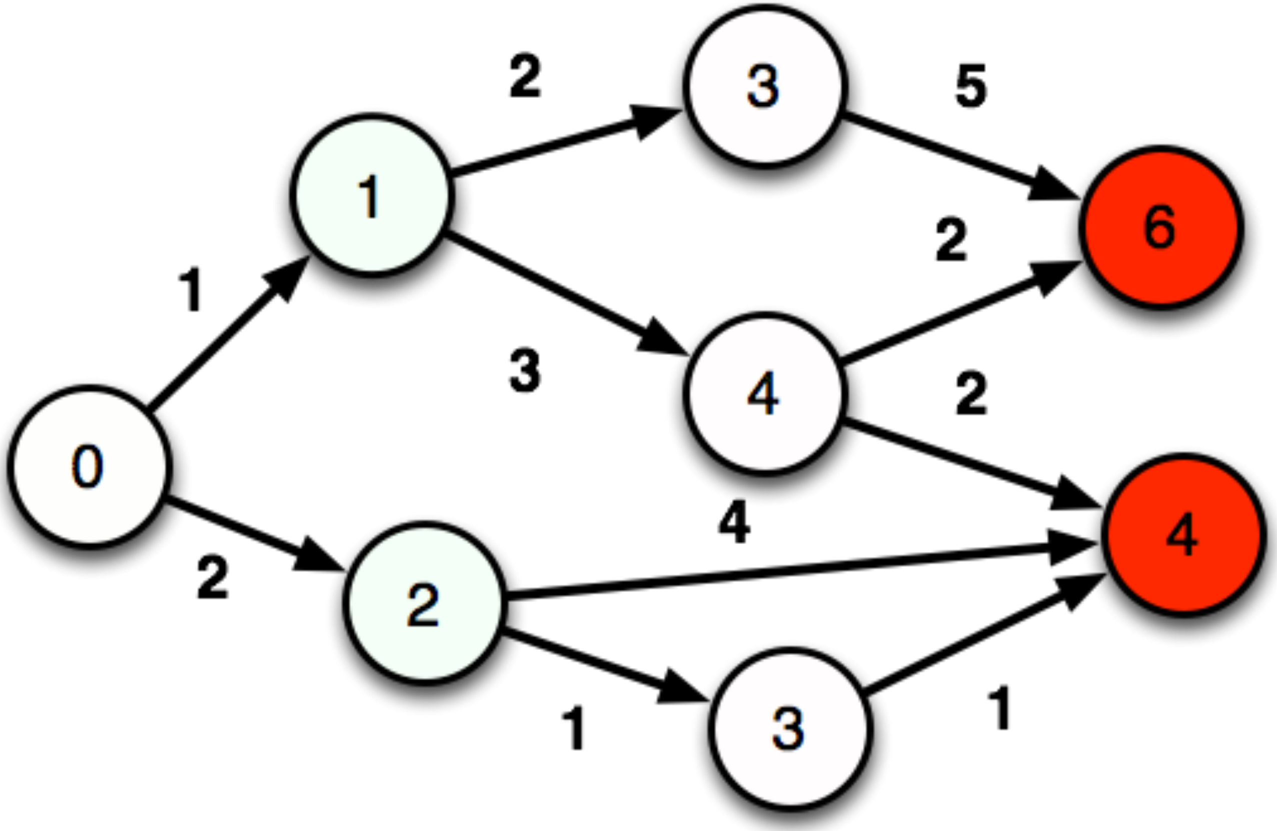
Shortest Paths



Shortest Paths



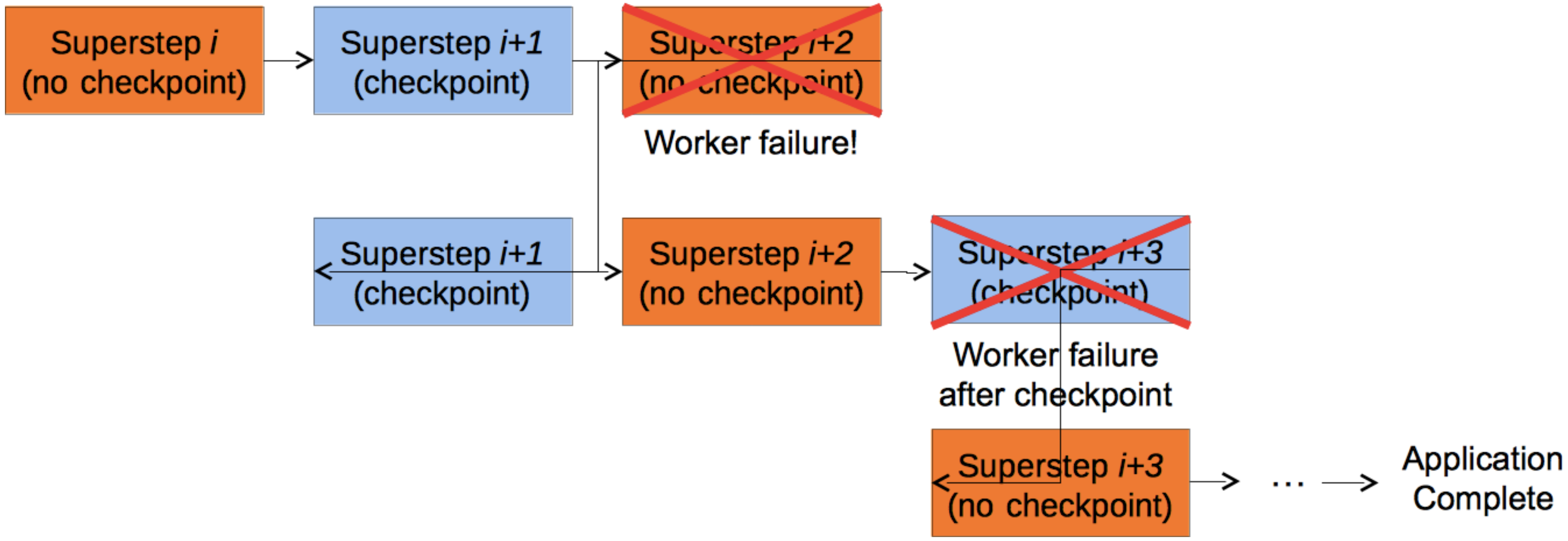
Shortest Paths



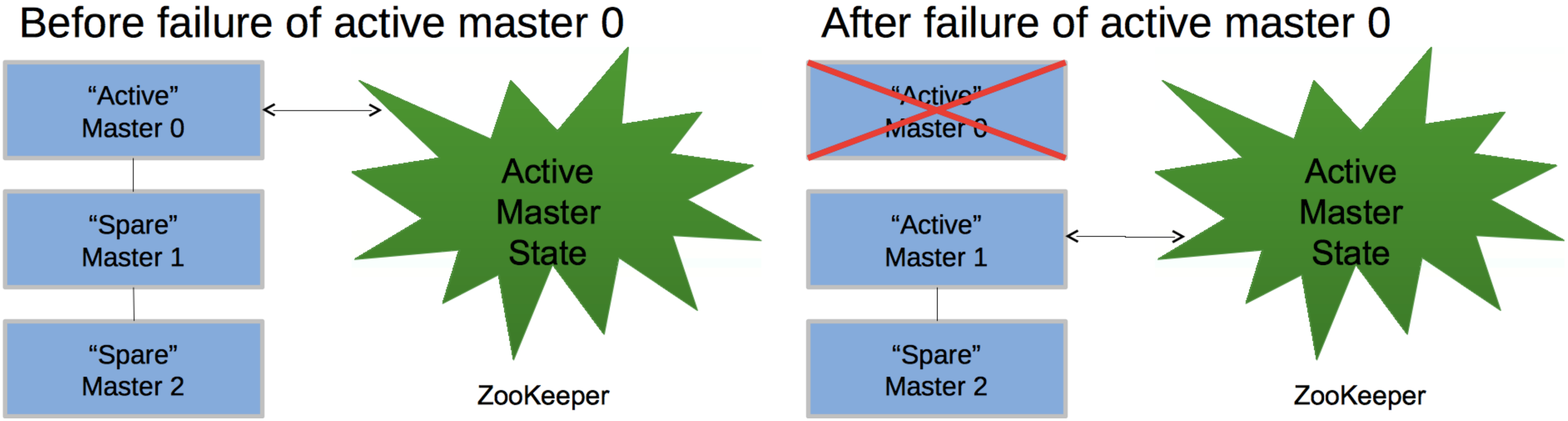
Properties

- Stateful (in-memory)
- Only intermediate values (messages) sent
- Hits the disk at input, output, checkpoint
- Combiners (minimizes messages)
- Aggregators (global aggregations)

Checkpointing



Failure management



Before failure of active master 0

"Active" Master 0

"Spare" Master 1

"Spare" Master 2

Active Master State

ZooKeeper

After failure of active master 0

"Active" Master 0

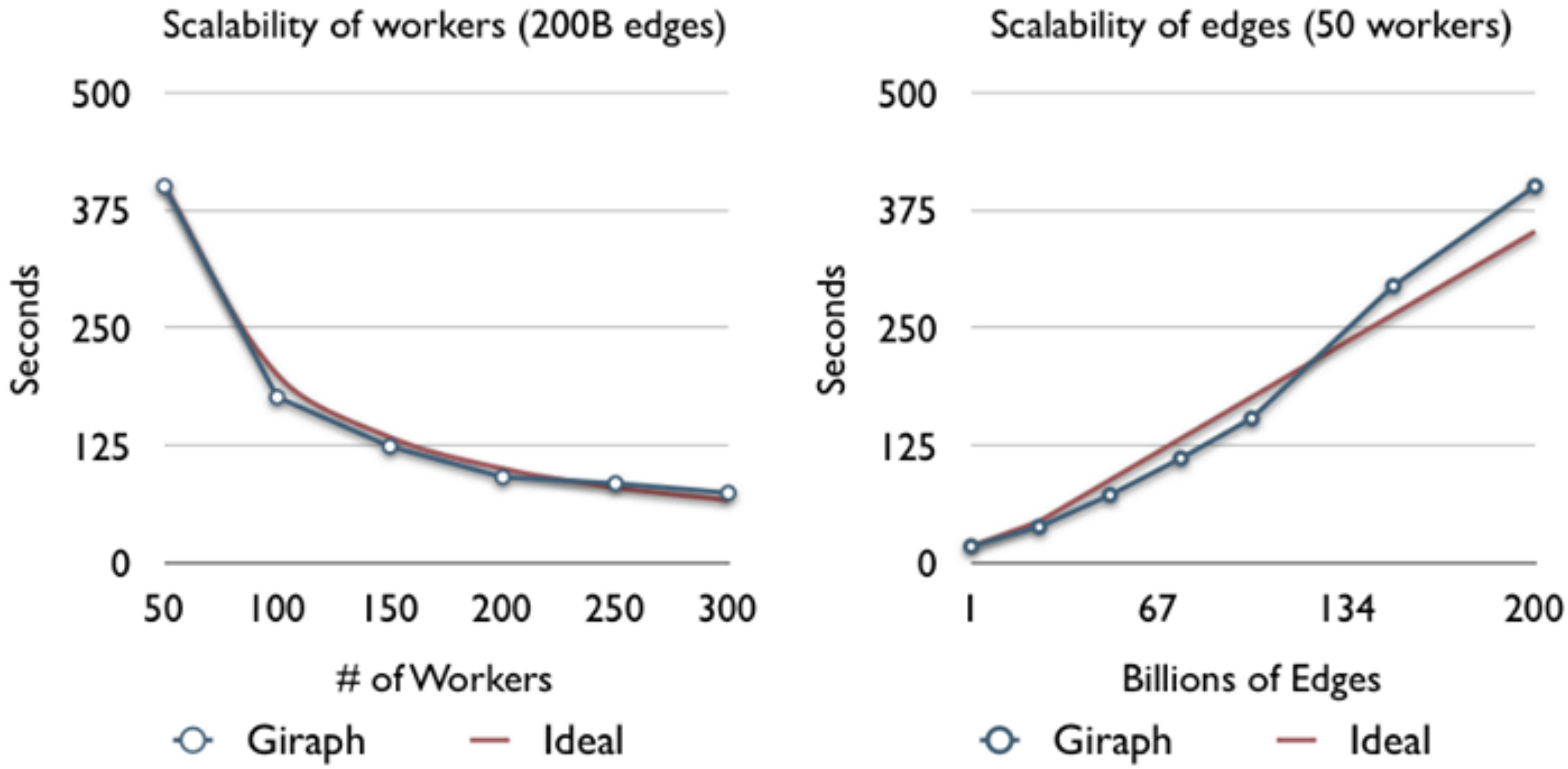
"Active" Master 1

"Spare" Master 2

Active Master State

ZooKeeper

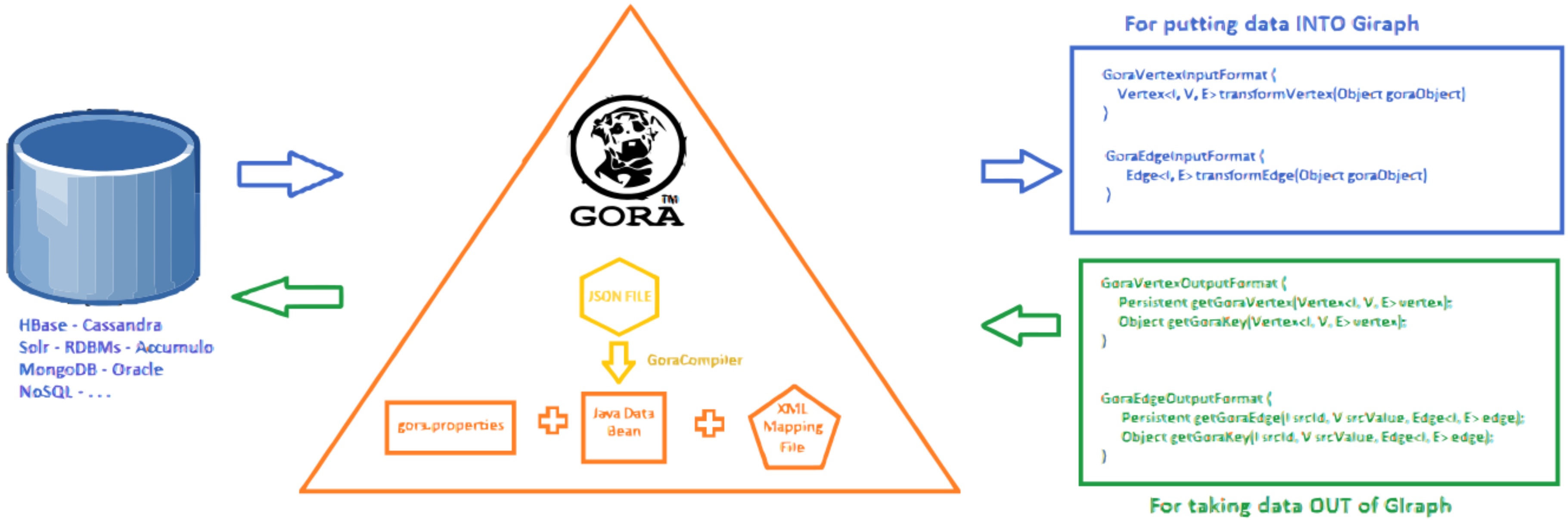
Giraph scales



Giraph is fast

- 100x over MR
- Jobs run within minutes
- Given you have resources

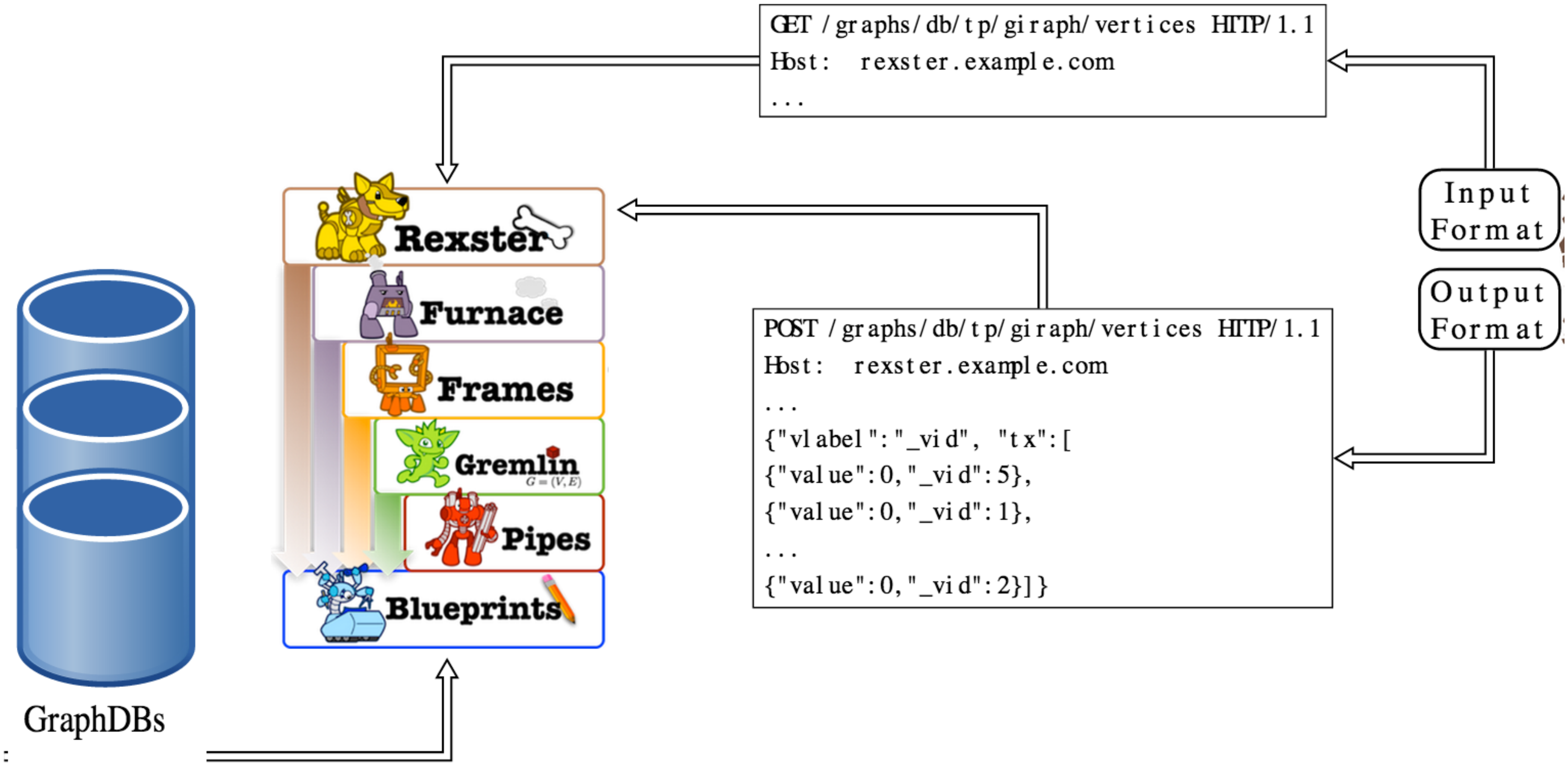
Many stores with Gora



```
GoraVertexInputFormat {
   Vertex<I, V, E> transformVertex(Object goraObject)
}

GoraEdgeInputFormat {
    Edge<I, E> transformEdge(Object goraObject)
}
```

```
GoraVertexOutputFormat {
   Persistent getGoraVertex(Vertex<I, V, E> vertex);
   Object getGoraKey(Vertex<I, V, E> vertex);
}

GoraEdgeOutputFormat {
    Persistent getGoraEdge(I srcId, V srcValue, Edge<I, E> edge);
    Object getGoraKey(I srcId, V srcValue, Edge<I, E> edge);
}
```

And graph databases



```
GET /graphs/db/tp/giraph/vertices HTTP/1.1
Host:  rexster.example.com
...
```

```
POST /graphs/db/tp/giraph/vertices HTTP/1.1
Host:  rexster.example.com
...
{"vlabel":"_vid", "tx":[
{"value":0,"_vid":5},
{"value":0,"_vid":1},
...
{"value":0,"_vid":2}]}
```

STORM

Storm?

- Storm is distributed processing of big data streams
- “Distributed and fault-tolerant real-time computation”
- http://storm.incubator.apache.org/
- Originated at BackType/Twitter, open sourced in late 2011
- Implemented in Clojure, some Java
- 12 core committers, plus ~ 70 contributors
- Current version: 2.1.0 (Oct 2019)
- Competitors: Flink, Streaming Spark, Samza, Apex, ..

WordCount example

```
(1.1.1.1, “foo.com”)
(2.2.2.2, “bar.net”)
(3.3.3.3, “foo.com”)
(4.4.4.4, “foo.com”)
(5.5.5.5, “bar.net”)
```

DNS queries

↓ ?

```
( (“foo.com”, 3)
  (“bar.net”, 2) )
```

Top queried domains

```
( (1.1.1.1, “foo.com”)
  (2.2.2.2, “bar.net”)
  (3.3.3.3, “foo.com”)
  (4.4.4.4, “foo.com”)
  (5.5.5.5, “bar.net”) )
```

DNS queries

⬇

```
(“foo.com”, “bar.net”, “foo.com”,
 “foo.com”, “bar.net”)
```

f

⬇

```
{“bar.net” -> 2, “foo.com” -> 3}
```

g

⬇

```
( (“foo.com”, 3)
  (“bar.net”, 2) )
```

h

Clojure

- Is a dialect of Lisp that targets the JVM (and JavaScript)
  - clojure-1.5.1.jar
- "Dynamic, compiled programming language"
  - Predominantly functional programming
- Many interesting characteristics and value propositions for software development, notably for concurrent applications
- Storm’s core is implemented in Clojure

Previous WordCount example in Clojure

$$\overbrace{\texttt{(sort-by val >}}^{h}\ \overbrace{\texttt{(frequencies}}^{g}\ \overbrace{\texttt{(map second}}^{f}\ \texttt{queries)))}$$

Alternative, left-to-right syntax with **->>**:

```
(->> queries (map second) frequencies (sort-by val >))
```

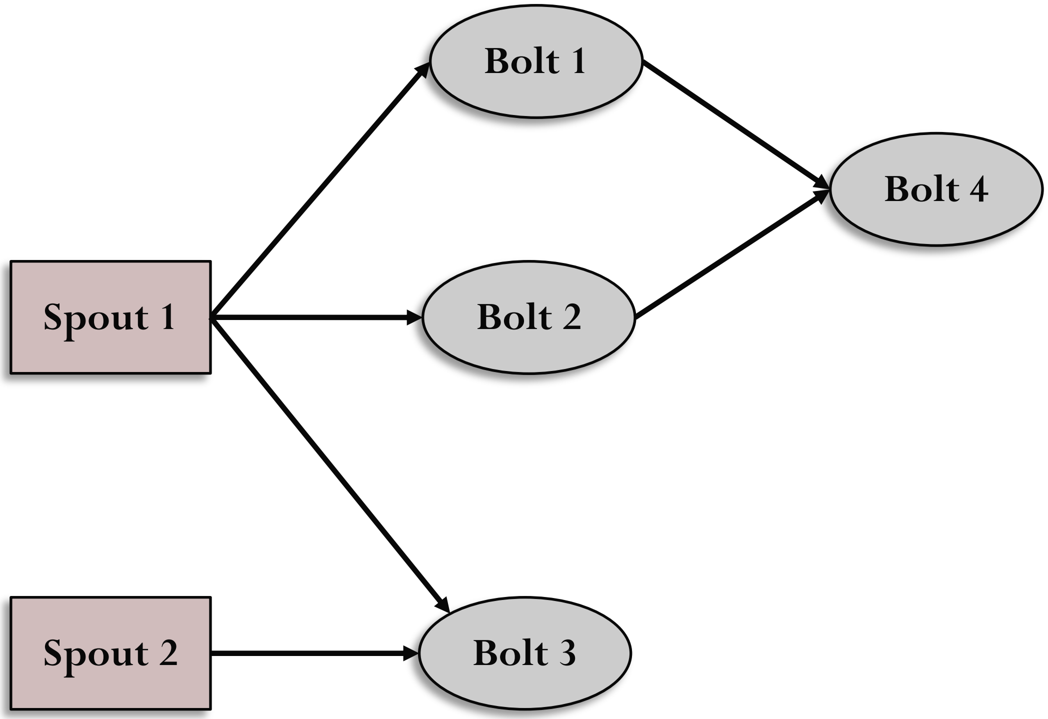
Clojure REPL

```
user> queries
(("1.1.1.1" "foo.com") ("2.2.2.2" "bar.net")
 ("3.3.3.3" "foo.com") ("4.4.4.4" "foo.com")
 ("5.5.5.5" "bar.net"))

user> (map second queries)
("foo.com" "bar.net" "foo.com" "foo.com" "bar.net")

user> (frequencies (map second queries))
{"bar.net" 2, "foo.com" 3}

user> (sort-by val > (frequencies (map second queries)))
(["foo.com" 3] ["bar.net" 2])
```

DAG processing model

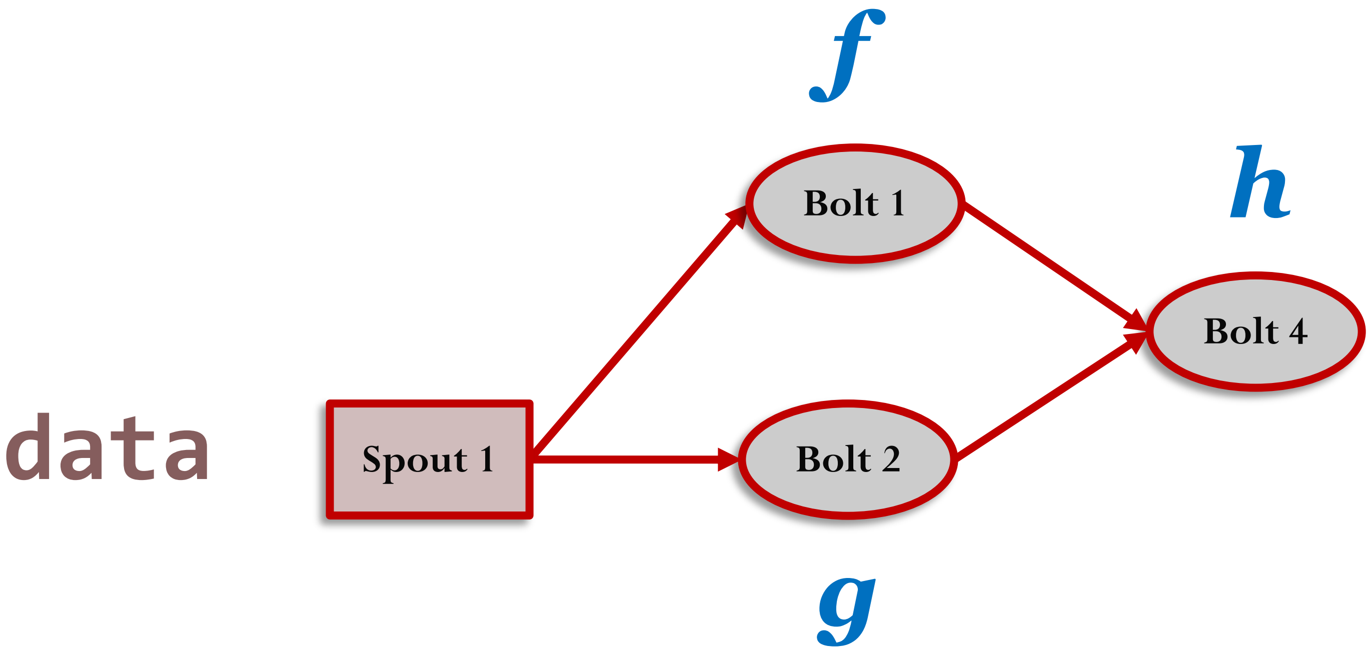
- A topology in Storm wires data and functions via a DAG
- Executes on many machines like a MR job in Hadoop

functions



data

Relationship between DAG and FP



f

h

Bolt 1

Bolt 4

data

Spout 1

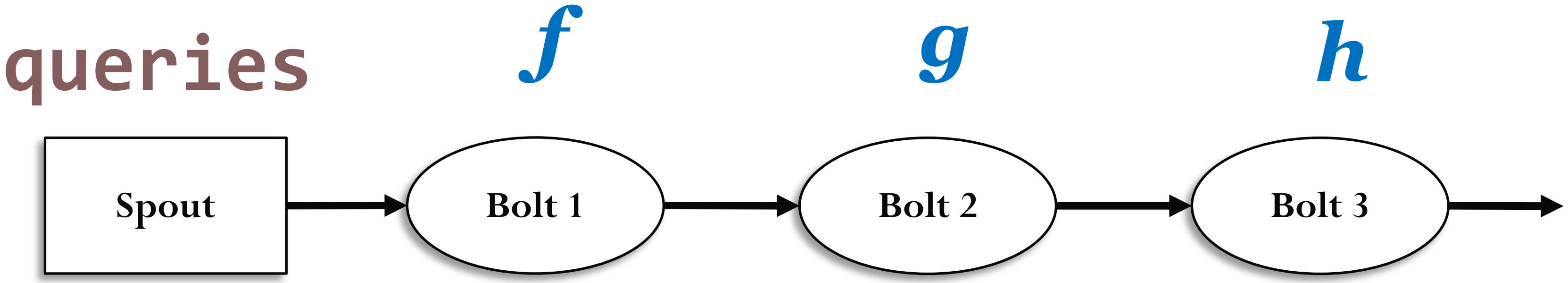
Bolt 2

g

h(*f*(data), *g*(data))

Previous WordCount example in Storm

```
(->> queries  (map second) frequencies  (sort-by val >) )
```



Data model

- Tuple = datum containing 1+ fields

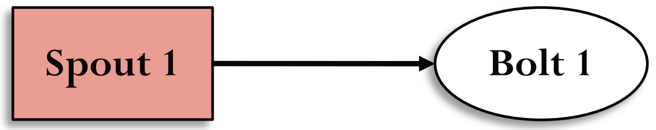
```
(1.1.1.1, “foo.com”)
```

- Values can be of any type
- Stream = unbounded sequence of tuples

```
          ...
(1.1.1.1, “foo.com”)
(2.2.2.2, “bar.net”)
(3.3.3.3, “foo.com”)
          ...
```

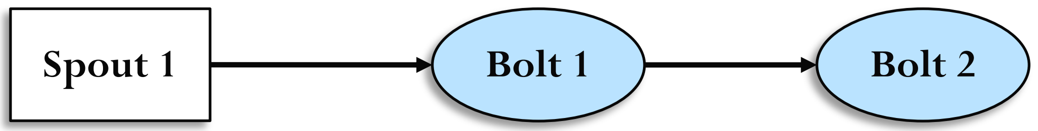
Spouts and bolts

- **Spout**: source of data streams



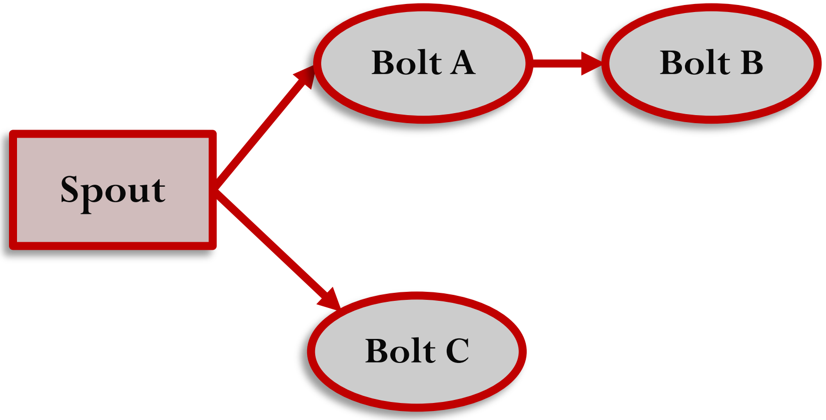
- Unreliable (fire-and-forget) or reliable (replay failed tuples).

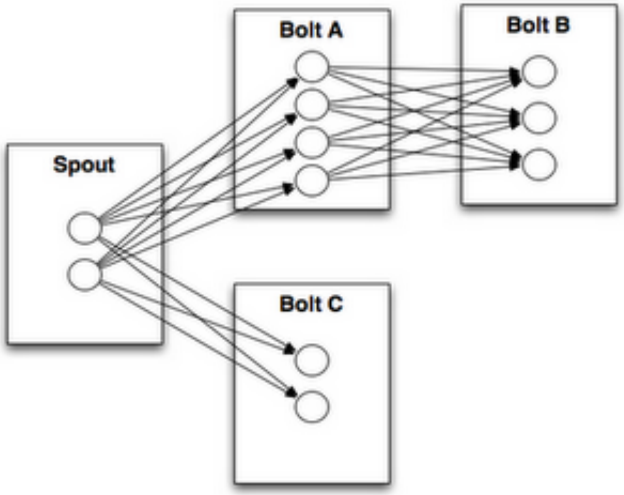
- **Bolt**: consumes 1+ streams and potentially produces new streams



- Can do anything from running functions, filter tuples, joins, talk to DB, etc.
- Complex stream transformations often require multiple steps and thus multiple bolts.

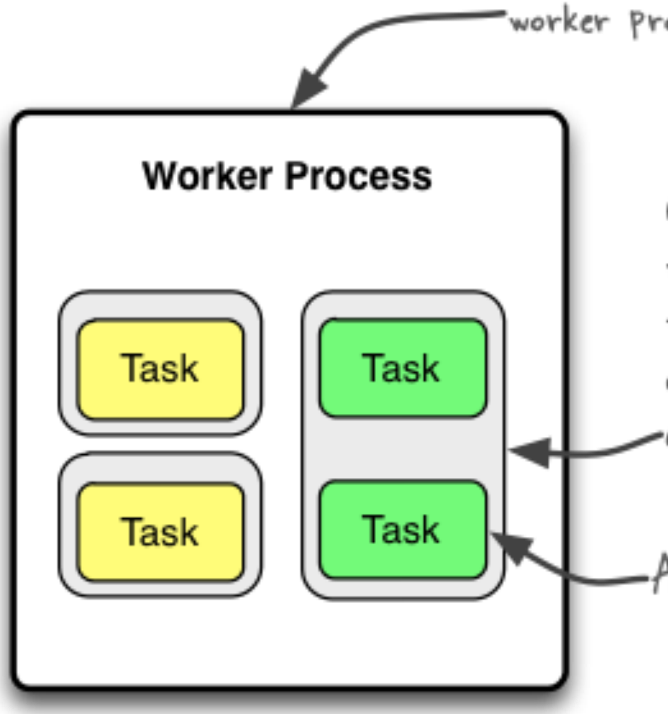
Stream groupings control the data flow





- Shuffle grouping: random; distribute load evenly to downstream bolts
- Fields grouping: GROUP BY field(s)
- All grouping: replicates stream across all the bolt's tasks;
- Global grouping: stream goes to a single one of the bolt's tasks;
- Direct grouping: data producer decides which task of the consumer will receive the data
- Custom groupings are possible, too.

Run time

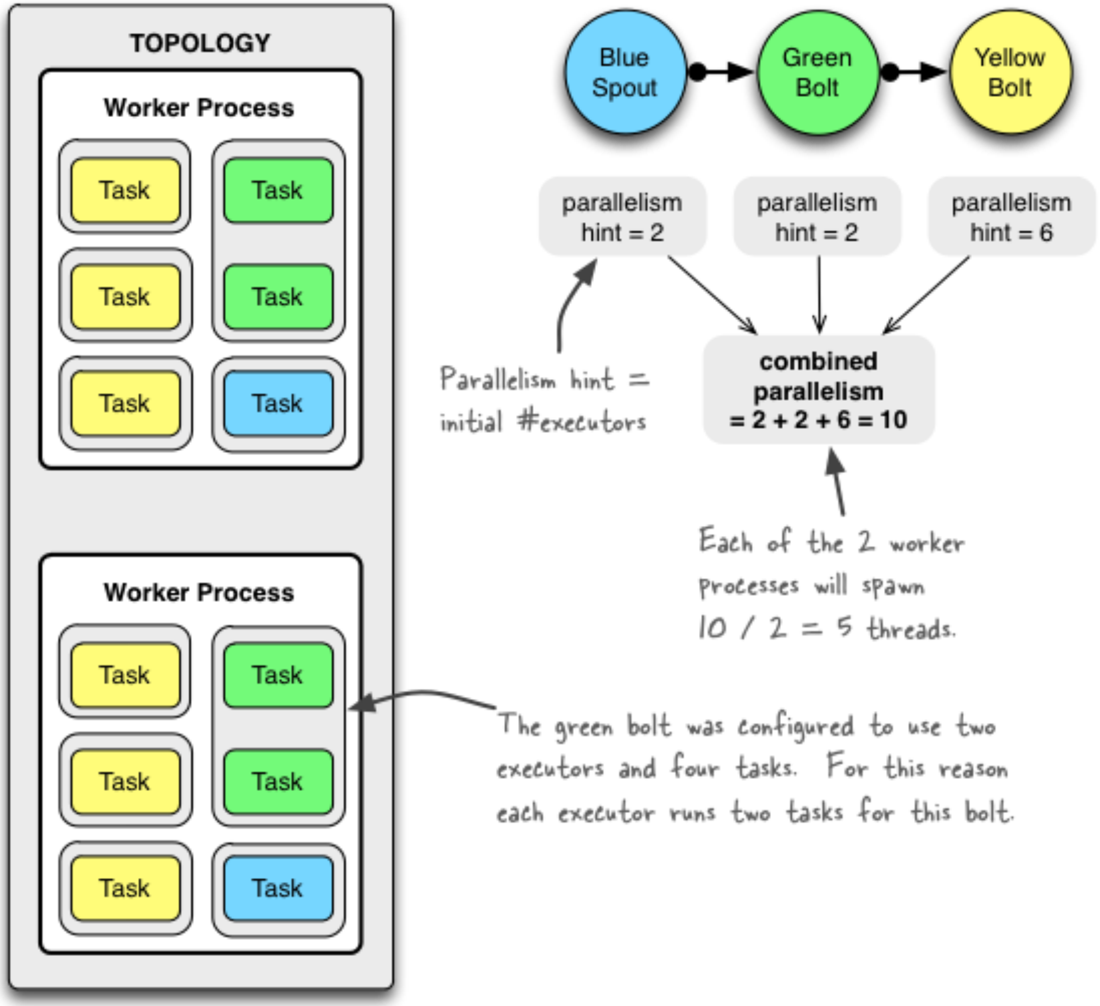


A machine in a Storm cluster may run one or more worker processes for one or more topologies. Each worker process runs executors for a specific topology.

One or more executors may run within a single worker process, with each executor being a thread spawned by the worker process. Each executor runs one or more tasks of the same component (spout or bolt).

A task performs the actual data processing.

Example of a running topology

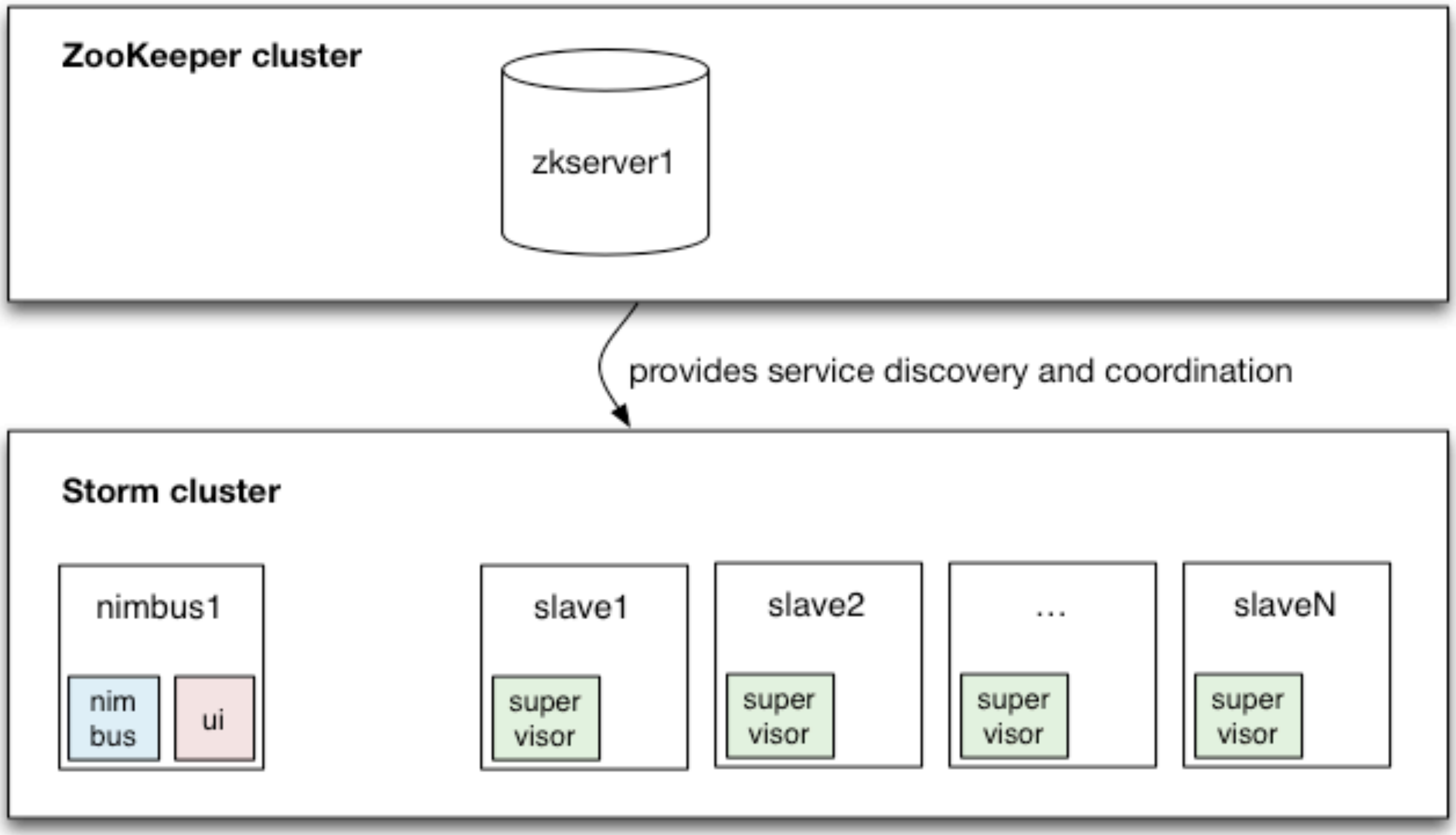


Parallelism hint = initial #executors

Each of the 2 worker processes will spawn 10 / 2 = 5 threads.

The green bolt was configured to use two executors and four tasks. For this reason each executor runs two tasks for this bolt.

Storm architecture



Commercial solutions: the big boys

- Oracle
  - Big Data Discovery
  - GoldenGate for Big Data,
  - Big Data SQL
  - NoSQL Database
- IBM
  - BLU Acceleration
  - PureData System for Hadoop
  - InfoSphere BigInsights
  - InfoSphere Streams
- Microsoft
  - Windows Azure HDInsight
- Amazon
  - Amazon Web Services
- Google
  - Dremel
  - BigQuery
- SAS
  - In-Memory Statistics
  - Visual Analytics
- SAP
  - Hana
  - SAP IQ
  - SAP ESP
- VMWare
  - vSphere
- Cisco
  - Connected Analytics
  - Big Data Warehouse Expansion
  - Prime Analytics

Many new players