One Trillion Edges

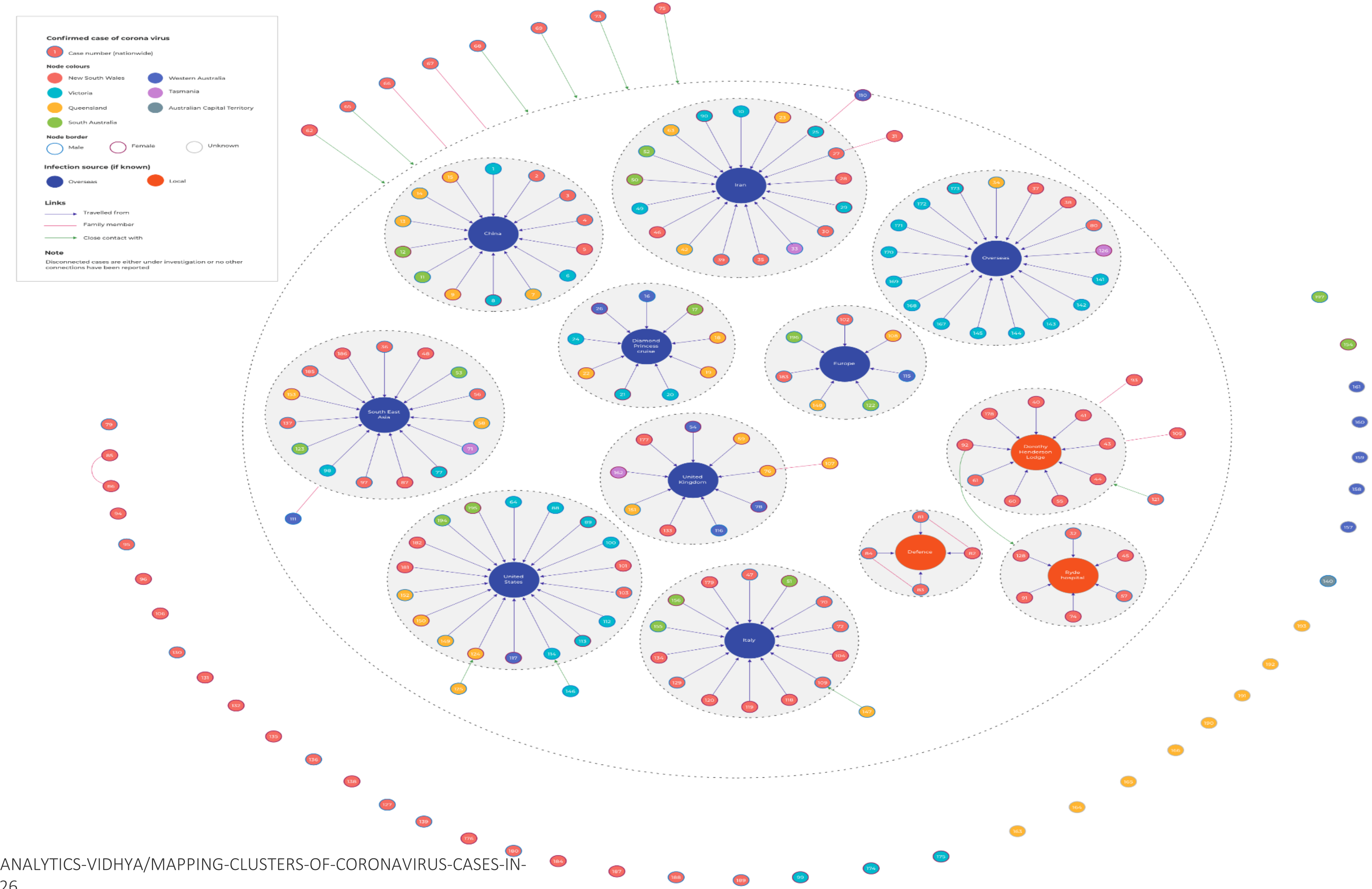
Graph Processing at Facebook Scale

Avery Ching, Sergey Edunov, Maja Kabiljo,

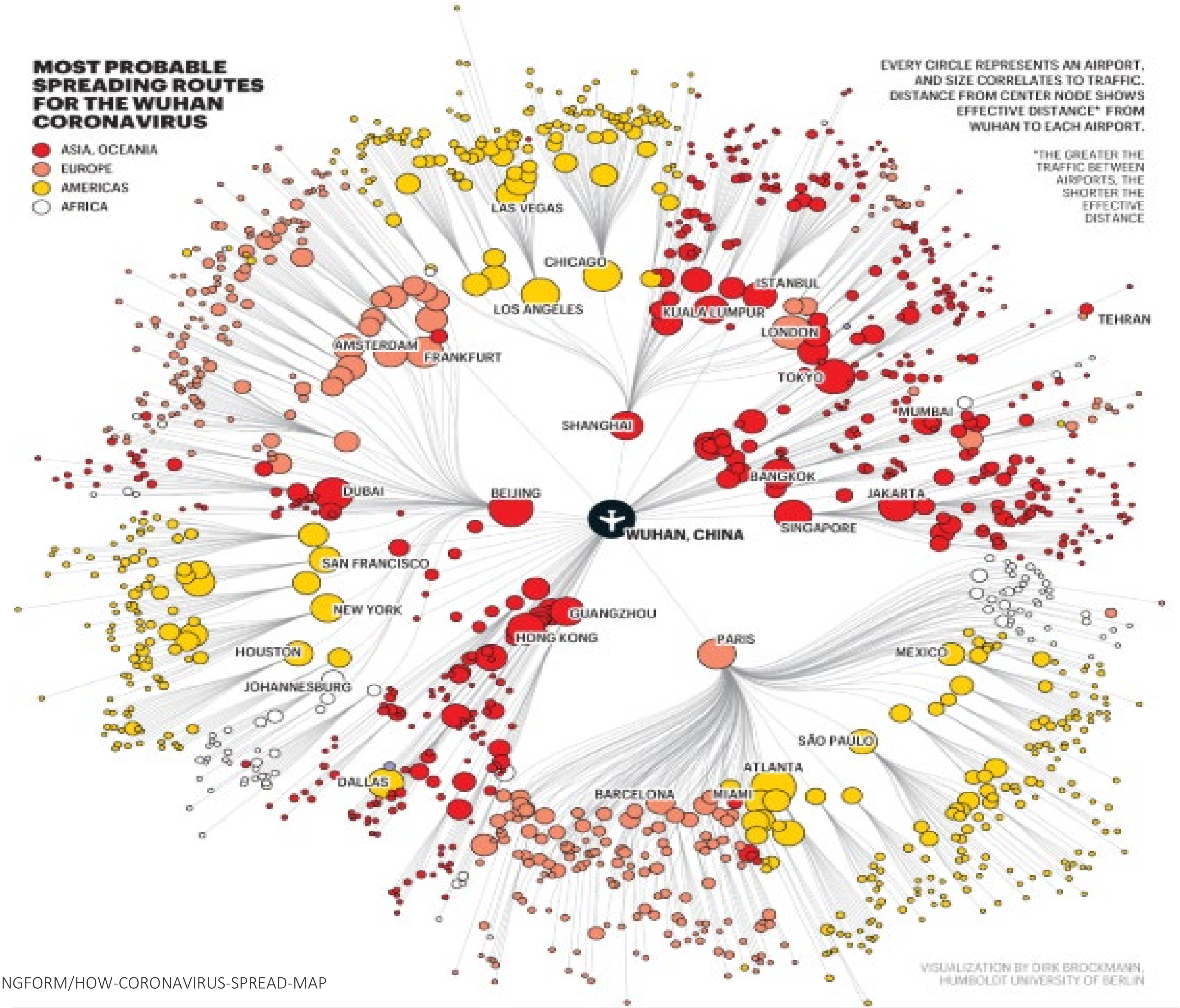
Dionysios Logothetis, Sambavi Muthukrishnan

<u>*Presented By*</u>*: Shivani Tripathi*

Graphs are Common



HTTPS://MEDIUM.COM/ANALYTICS-VIDHYA/MAPPING-CLUSTERS-OF-CORONAVIRUS-CASES-IN-AUSTRALIA-76927EBF4526


MOST PROBABLE SPREADING ROUTES FOR THE WUHAN CORONAVIRUS
ASIA, OCEANIA
EUROPE
AMERICAS
AFRICA
EVERY CIRCLE REPRESENTS AN AIRPORT, AND SIZE CORRELATES TO TRAFFIC. DISTANCE FROM CENTER NODE SHOWS EFFECTIVE DISTANCE* FROM WUHAN TO EACH AIRPORT.
*THE GREATER THE TRAFFIC BETWEEN AIRPORTS, THE SHORTER THE EFFECTIVE DISTANCE
LAS VEGAS
CHICAGO
LOS ANGELES
ISTANBUL
KUALA LUMPUR
LONDON
TEHRAN
AMSTERDAM
FRANKFURT
TOKYO
SHANGHAI
MUMBAI
BANGKOK
DUBAI
BEIJING
JAKARTA
WUHAN, CHINA
SINGAPORE
SAN FRANCISCO
NEW YORK
GUANGZHOU
HONG KONG
PARIS
HOUSTON
MEXICO
JOHANNESBURG
SÃO PAULO
ATLANTA
DALLAS
BARCELONA
MIAMI
VISUALIZATION BY DIRK BROCKMANN, HUMBOLDT UNIVERSITY OF BERLIN

May 27, 2021

FORTUNE.COM/LONGFORM/HOW-CORONAVIRUS-SPREAD-MAP

Other Common Graphs

- Web & Social Networks
  - Web graph, Citation Networks, Twitter, Facebook
- Knowledge networks & relationships
  - Google's Knowledge Graph, NELL
- Cybersecurity
  - Telecom call logs, financial transactions, Malware
- Internet of Things
  - Transport, Power, Water networks
- Bioinformatics
  - Gene sequencing, Gene expression networks

Graphs are Huge!!!

- Real world web and social graphs are huge and continue to grow
  - Google estimated no. of web pages 30 trillion (2013)
  - Facebook has around 2 billion active users (2018)
  - Google has around 570 million users
  - Twitter claimed to have over 530 million users
- Relevant and personalized information for users relies strongly on iterative graph ranking algorithms (search results, social news, ads, etc)
  - In web graphs, page rank and its variants
  - In Social graphs, popularity rank, shared connection, shortest paths, etc.

Graph Algorithms

- Traversals: Paths & flows between different parts of the graph
  - Breadth First Search, Shortest path, Minimum Spanning Tree, Eulerian paths, MaxCut
- Clustering: Closeness between sets of vertices
  - Community detection & evolution, Connected components, K-means clustering, Max Independent Set
- Centrality: Relative importance of vertices
  - PageRank, Betweenness Centrality
- List is endless

Why Existing Solutions Fail?

- Shared memory algorithms don't scale!
- Graph algorithms are computationally expensive
- Do not fit naturally to Hadoop/MapReduce
  - Classical Map-Reduce Overheads (jobs startup/shutdown, reloading data from HDFS, shuffling)
  - Map-reduce programming model not good fit for graph algorithms
- Lot of work on parallel graph libraries for HPC
  - Storage & compute are (loosely) coupled, not fault tolerant
  - But everyone does not have a supercomputer
- Message Passing Interface
  - Not Fault-tolerant
  - Too generic

Pregel

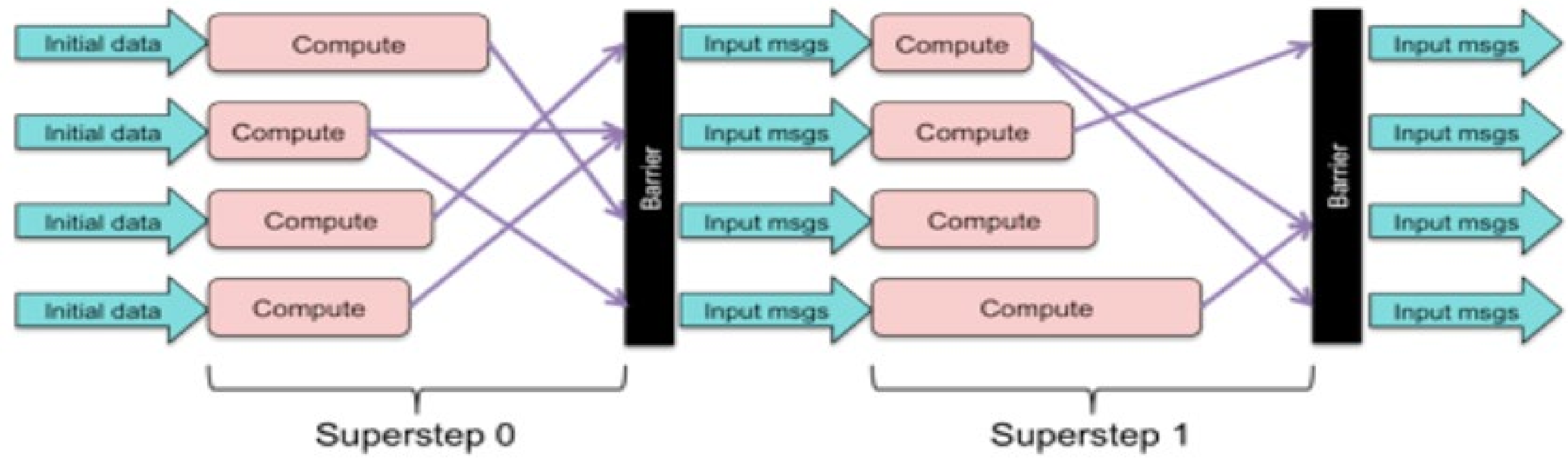
oogle

Overview of Pregel

- To overcome these challenges, google came up with Pregel
- Vertex-centric Model for writing Graph algorithms
  - Scalability
  - Expressibility in writing algorithms
  - Fault-tolerance
- The high level organization of Pregel programs is inspired by Valiant's Bulk Synchronous Parallel (BSP) model

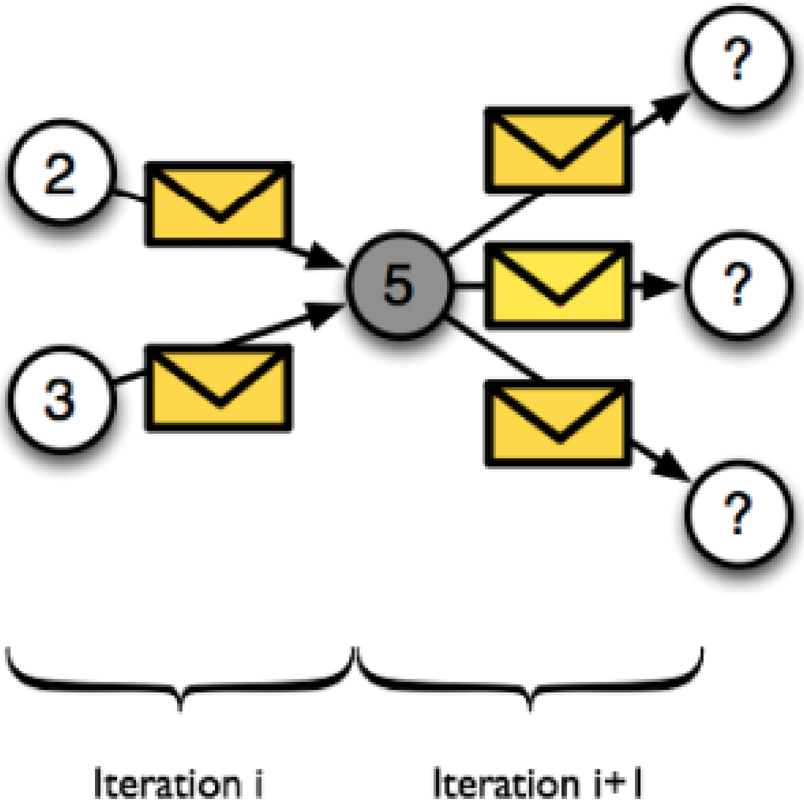
Bulk Synchronous Parallel (BSP)

- Computations consist of a sequence of iterations, called supersteps
  - Concurrent computation
  - Communication
  - Barrier synchronisation



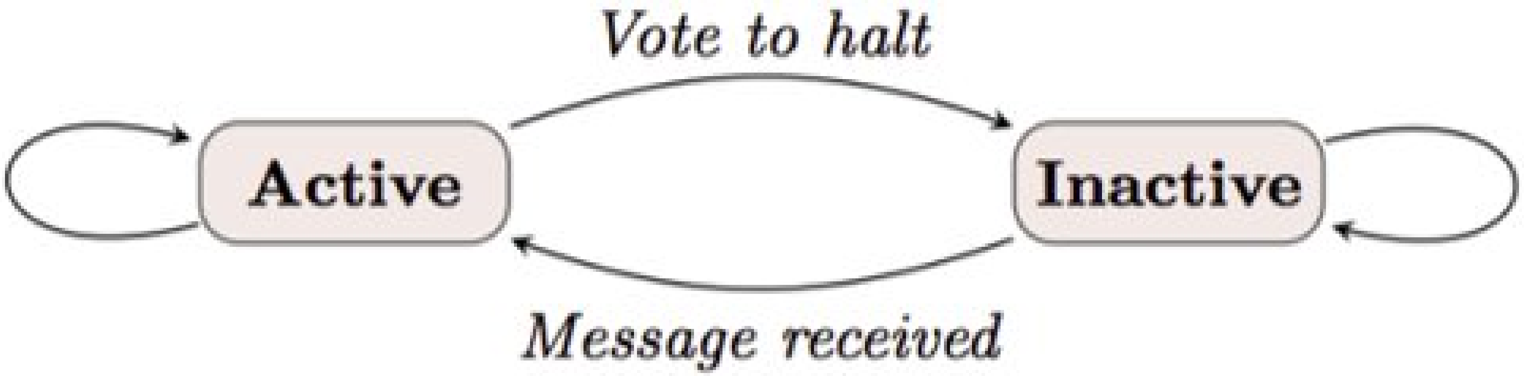
Vertex Centric Programming

- Think like a "Vertex"
  - Logic written from perspective on a single vertex
  - Executed on all vertices.
- Vertices know about
  - Their own value(s)
  - Their outgoing edges

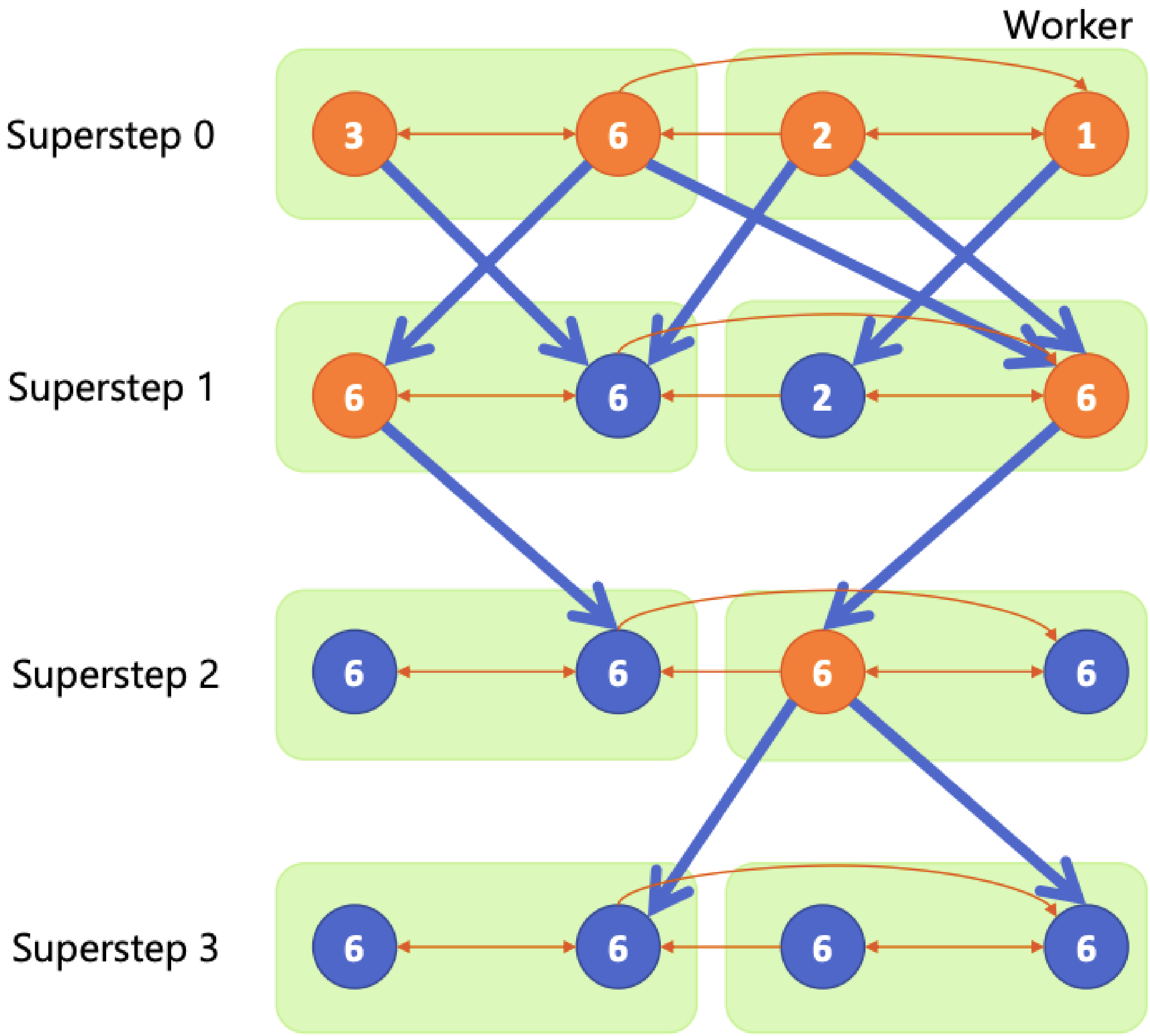


Vertex State Machine

- Algorithm termination is based on every vertex voting to halt.
- In superstep 0, every vertex is in the active state.
- A vertex deactivates itself by voting to halt.
- It can be reactivated by receiving an (external) message.



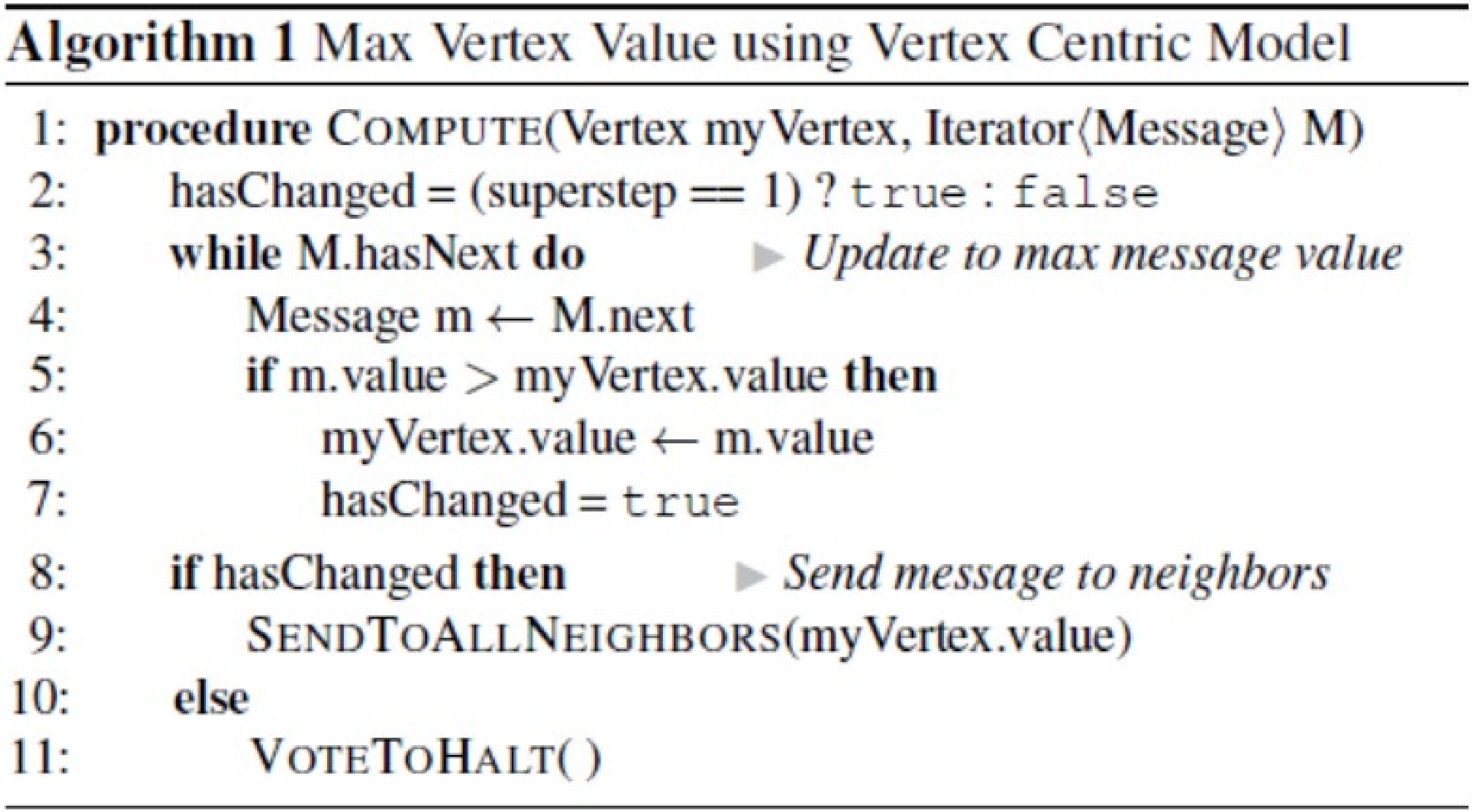
Example: Max Vertex Value



Blue Arrows are messages.

Blue vertices have voted to halt.

Max Vertex Value: Code



Algorithm 1 Max Vertex Value using Vertex Centric Model

```
procedure COMPUTE(Vertex myVertex, Iterator⟨Message⟩ M)
    hasChanged = (superstep == 1) ? true : false
    while M.hasNext do                    ▷ Update to max message value
        Message m ← M.next
        if m.value > myVertex.value then
            myVertex.value ← m.value
            hasChanged = true
    if hasChanged then                    ▷ Send message to neighbors
        SENDTOALLNEIGHBORS(myVertex.value)
    else
        VOTETOHALT( )
```

Advantages

- Makes distributed programming easy
  - No locks, semaphores, race conditions
  - Separates computing from communication phase
- Vertex-level parallelization
  - Bulk message passing for efficiency
- Stateful (in-memory)
  - Only messages & checkpoints hit disk

Why not Pregel?

- Requires its own computing infrastructure
- Not available unless you work at Google
- Master is Single Point of Failure

Apache Giraph

Open-Source Implementation of Pregel

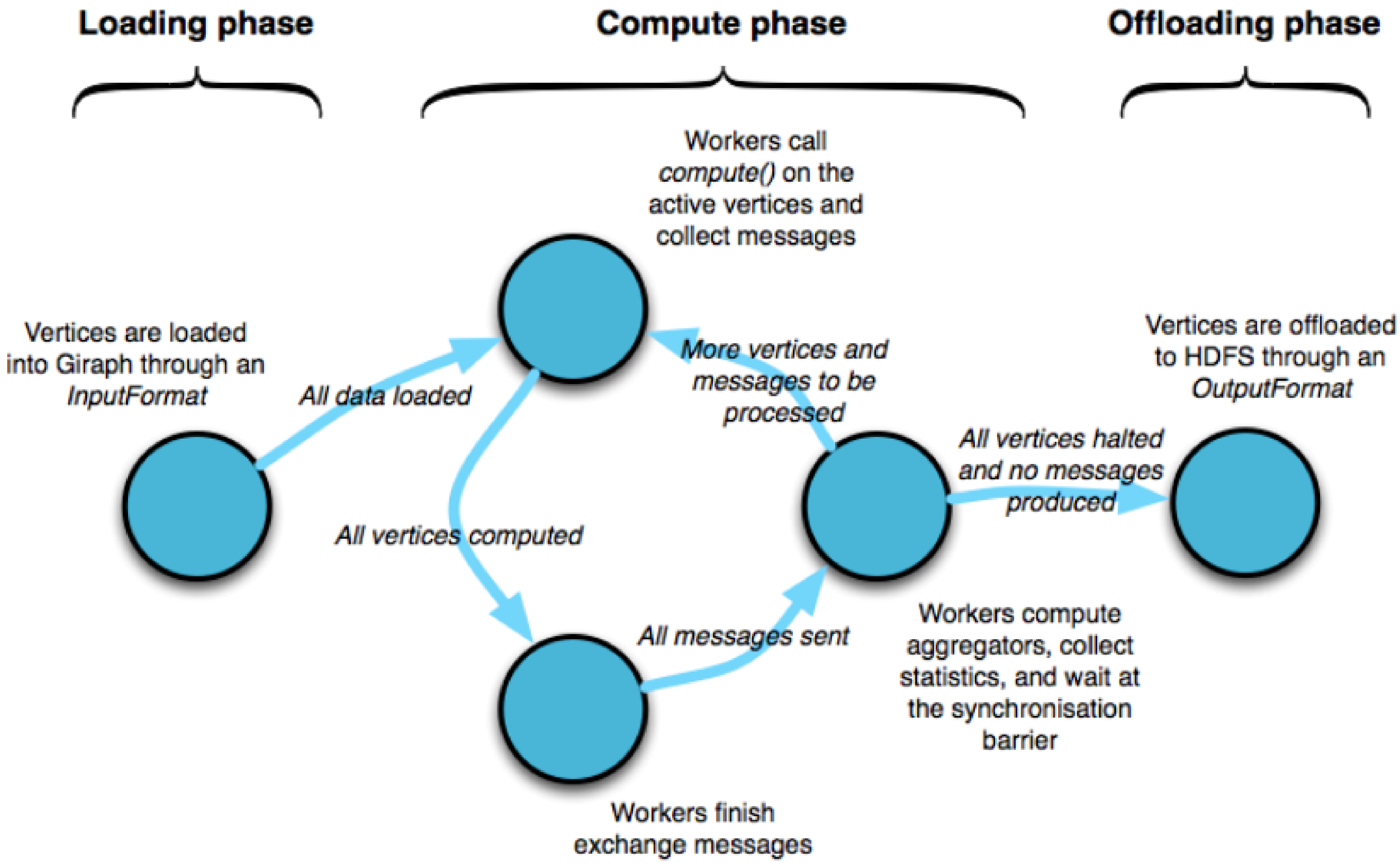
Giraph

- Leverage Hadoop Installations around the world for iterative graph processing
  - Big Data today is processed on Hadoop with map-reduce computing model
  - Map-reduce with Hadoop is widely deployed
- Bulk Synchronous Processing (BSP) Computing Model
  - Input data loaded once during the application, all messaging in memory
- Fault-tolerant/Dynamic Graph Processing Infrastructure
  - Automatically adjust to available resources on Hadoop grid
  - No single point of failure except Hadoop namenode and Jobtracker
  - Relies on Zookeeper as a fault-tolerant coordination service
- Vertex Centric API to do graph processing in a BSP computing model
  - Inspired by Pregel
- Open Source

Giraph API

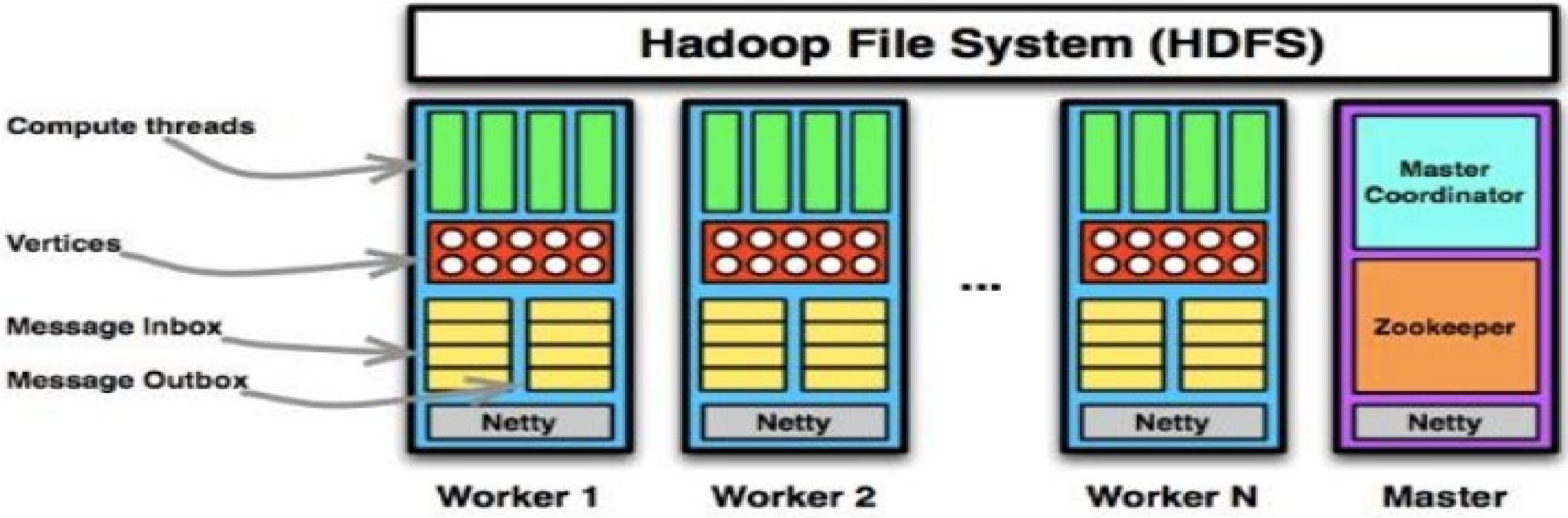
- void compute(Iterator msgs)
  - getSuperstep()
  - getVertexValue()
  - edges = iterator() //list of edges
  - sendMsg(edge, value)
  - sendMsgToAllEdges(value)
  - VoteToHalt()
- Messages Passing
  - Message ordering not guaranteed
  - Can send messages to any node
  - Message is delivered exactly once

Giraph Job Lifetime


Loading phase
Compute phase
Offloading phase
Workers call *compute()* on the active vertices and collect messages
Vertices are loaded into Giraph through an *InputFormat*
All data loaded
More vertices and messages to be processed
Vertices are offloaded to HDFS through an *OutputFormat*
All vertices halted and no messages produced
All vertices computed
All messages sent
Workers compute aggregators, collect statistics, and wait at the synchronisation barrier
Workers finish exchange messages

Giraph Architecture

- Master (responsible for coordination)
  - Assigns partitions to workers, Synchronization
- Workers (responsible for vertices)
  - Operates on set of vertices called partitions
  - Invokes active vertices, sends/receive and assign messages
- Zookeeper (responsible for computation state)
  - Keeps track of the computation state



Fault Tolerance

- Checkpointing
  - The master periodically (alternate supersteps) instructs the workers to save the state of their partitions to HDFS.
  - 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.
- No single point of failure from BSP threads
- Hadoop single point of failure still exists

Additional Features

Combiners

- Sending a message to another vertex that exists on a different machine has some overhead.
- User specifies a way to reduce many messages into one value (ala Reduce in MR).
  - by overriding the Combine() method.
  - Must be commutative and associative.
- Runs on both the client side and server side
  - Client side saves memory and message traffic
  - Server side saves memory
- Exceedingly useful in certain contexts (e.g., 4x speedup on shortest-path computation).

Aggregators

- A mechanism for global communication, monitoring, and data.
  - Each vertex can produce a value in a superstep S for the Aggregator to use.
  - The Aggregated value is available to all the vertices in superstep S+1.
- Commutative and associate operations that are performed globally.
- Aggregators can be used for statistics and for global communication.
  - E.g., Sum applied to out-edge count of each vertex.
    - *generates the total number of edges in the graph and communicate it to all the vertices.*

Application

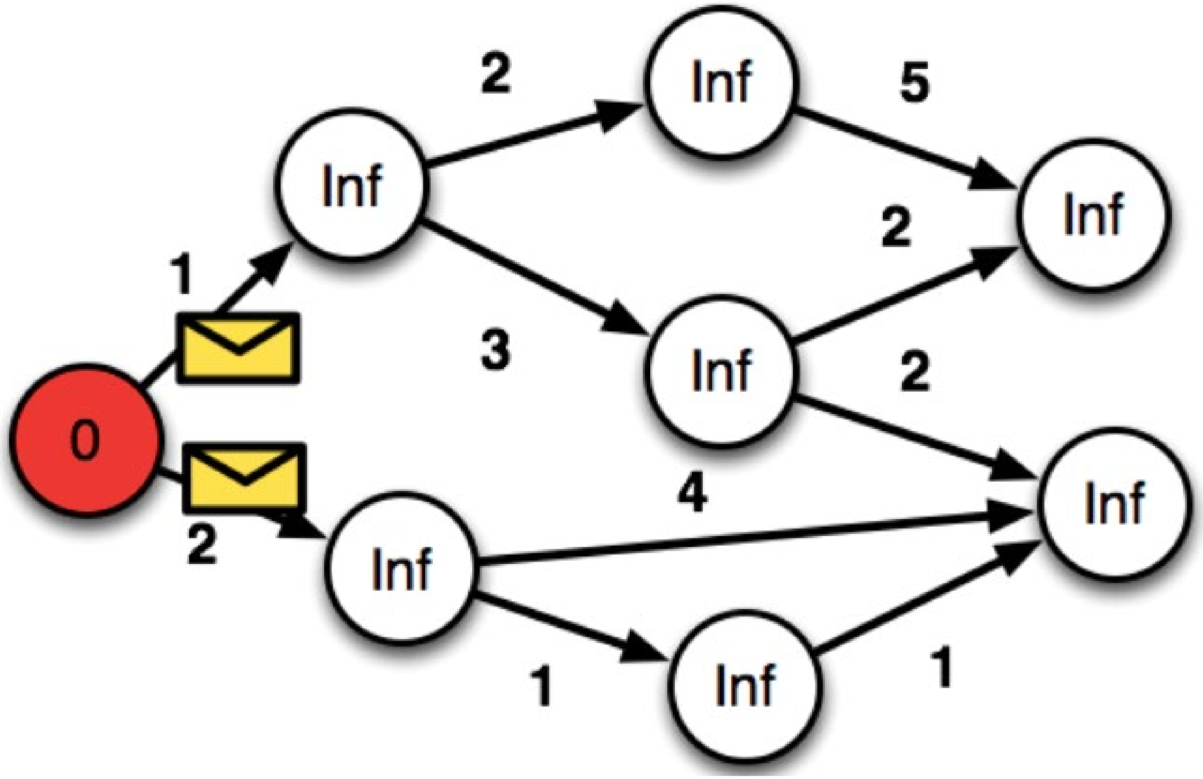
Shortest Path

```
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();
  }
};
```

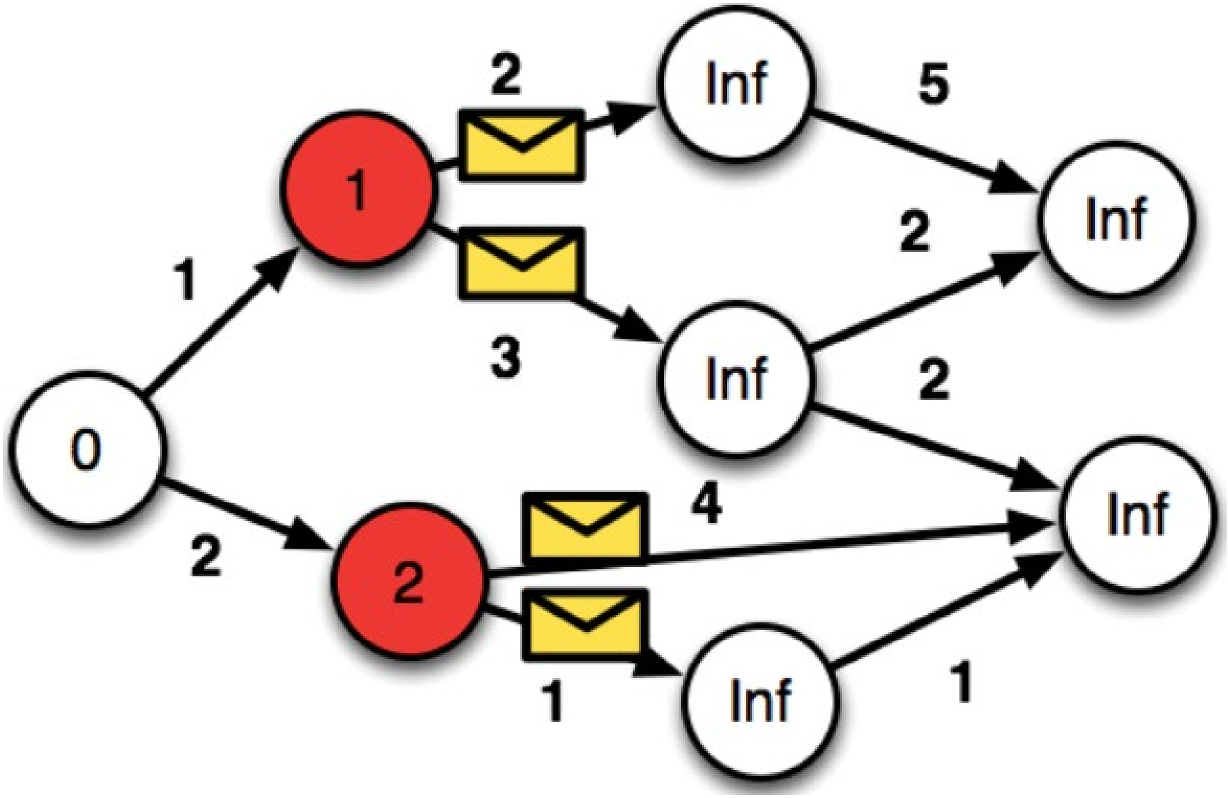
In the 0th superstep, only source vertex will update its value

May 27, 2021

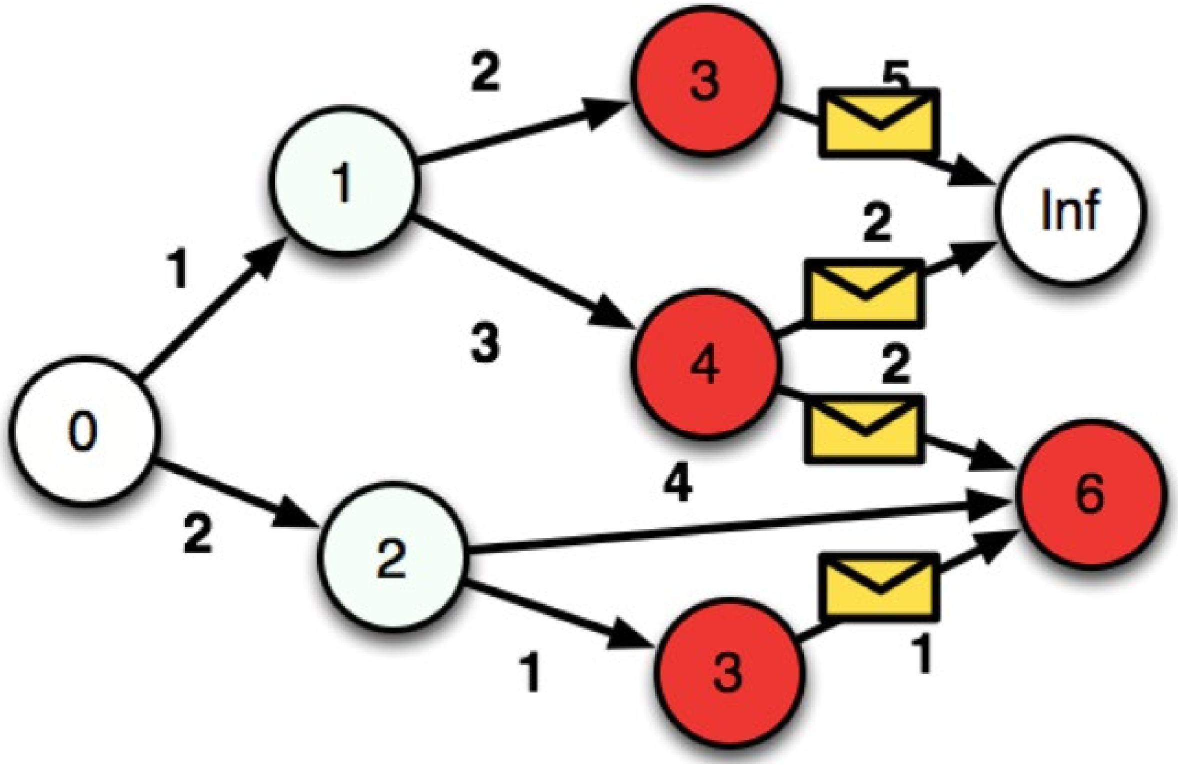
Shortest Path



Shortest Path

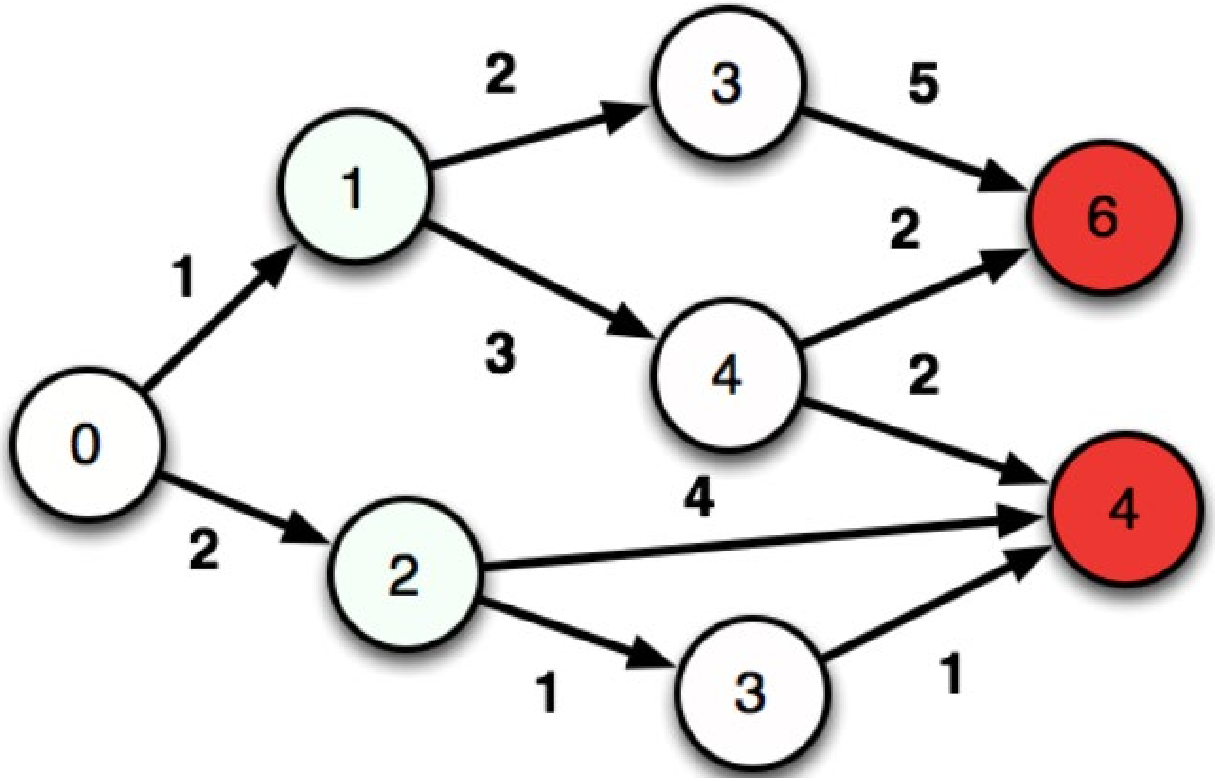


Shortest Path



May 27, 2021

Shortest Path

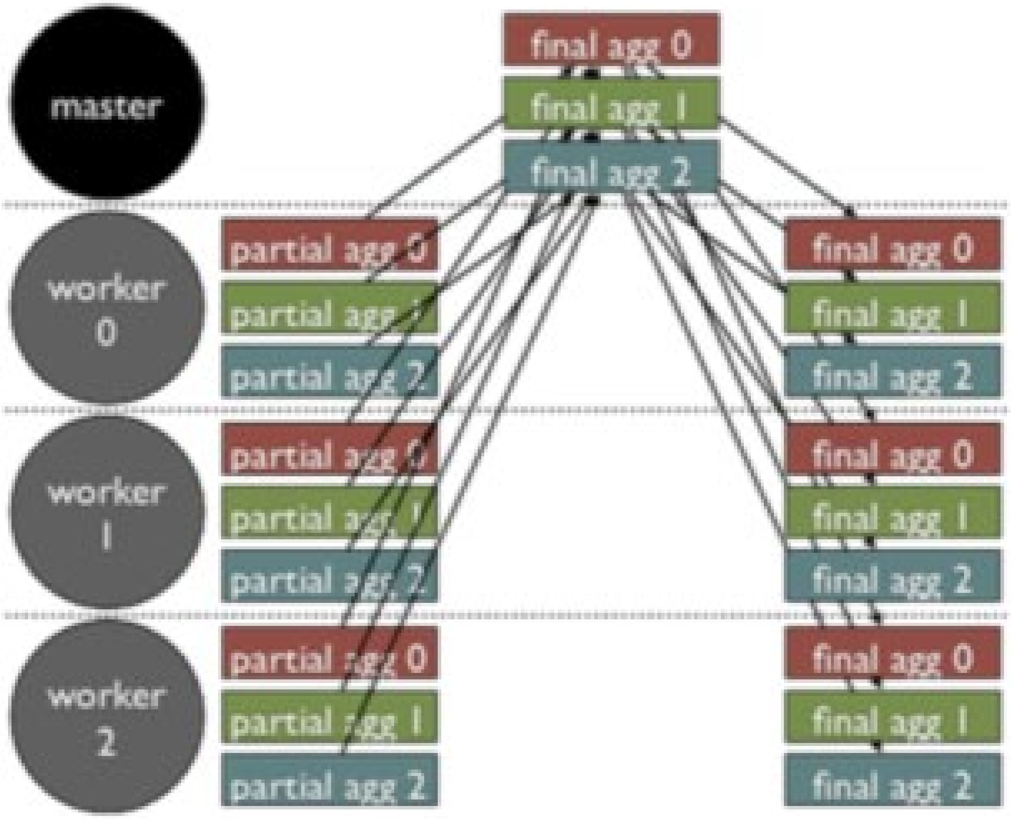


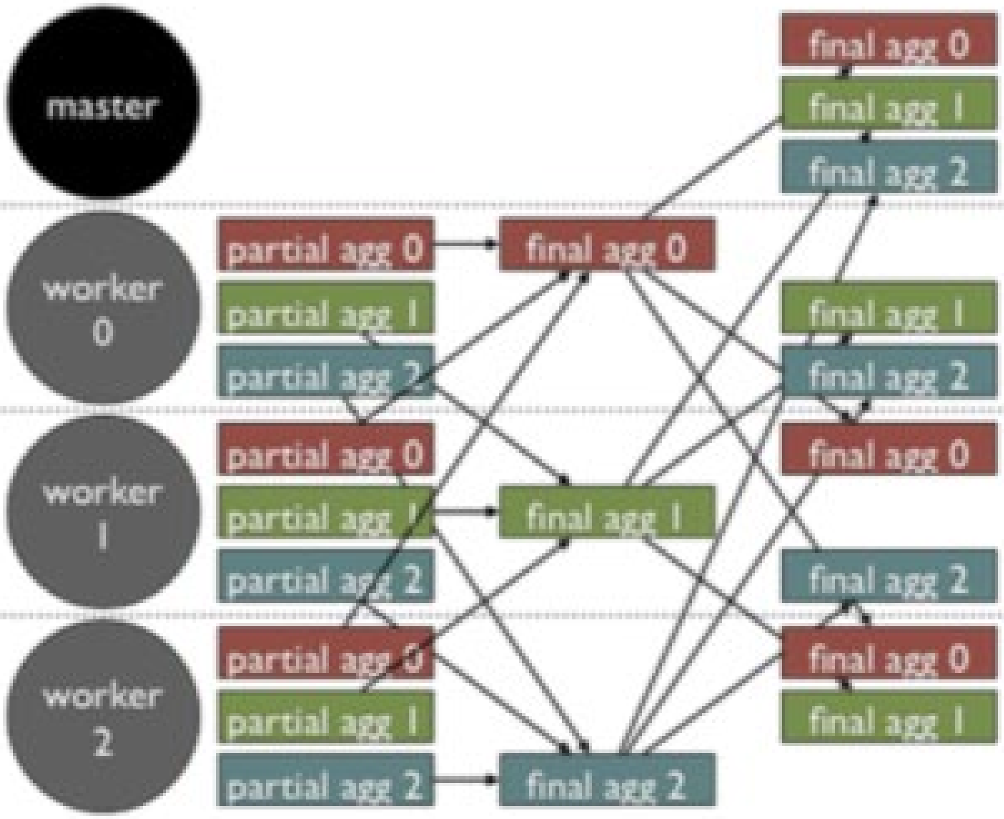
Ready to Process Trillion Edges

Platform Improvement

- Efficient Memory Management (MM)
  - Vertex and Edge data stored using serialized byte array
  - Better MM -> Less GC
- Support for Multi-Threading
  - Maximized resource utilization
  - Linear speed-up for CPU bound applications like K-Means Clustering
- Flexible IO Format
  - Reduces Pre-processing
  - Allows Vertex and Edge data to be loaded from different sources
- Sharded Aggregator
  - Aggregator responsibilities are balanced across workers
  - Different Aggregators can be assigned to different workers.

Sharded Aggregator





Compute Model Extensions

- Master Compute
  - Allows centralized execution of computation
- Worker Phases
  - Special methods which by-pass Pregel Model, but add ease of usability
  - Applicability is application specific
- Computation Composability
  - Decouples Vertex and Computation
  - Existing Computation implementation can be decoupled for multiple applications
- Superstep Splitting
  - Master runs same “Message Heavy” Superstep for fixed number of iterations
  - For an iteration, vertex computation invoked if vertex passes hash function
  - Example : Friends-of-Friends Computation

Take Away

- Giraph is a graph processing infrastructure that runs on existing Hadoop infrastructure
- Open source- available on GitHub
- Scales to “Trillion” edge graph

THANK
YOU