Cloud Computing
CS 15-319

Pregel

Lecture 10, Feb 15, 2012

Majd F. Sakr, Suhail Rehman and

Mohammad Hammoud

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

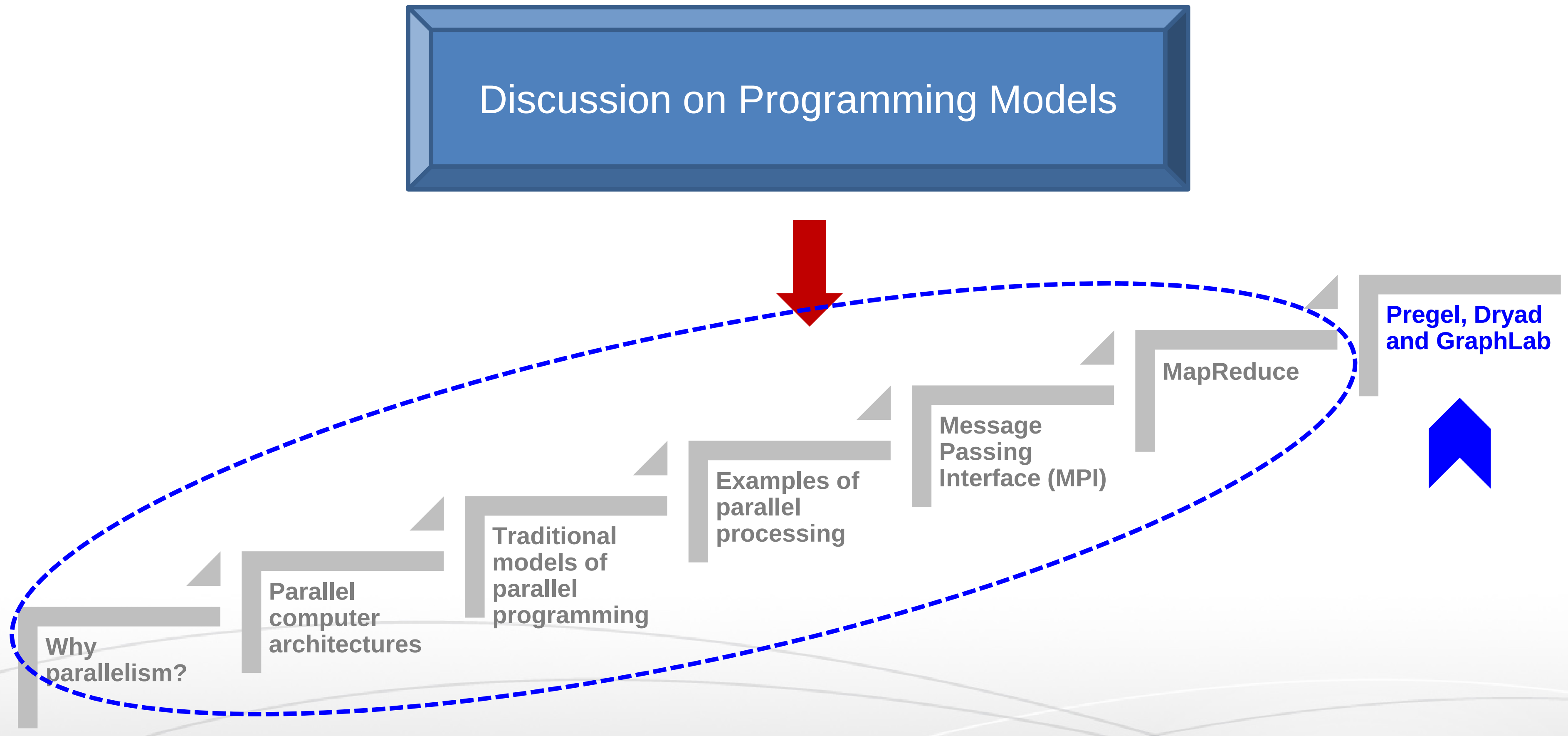
Today…

- Last session
  - Apache Mahout, Guest Lecture

- Today's session
  - Pregel

- Announcement:
  - Project Phases I-A and I-B are due today

Objectives



Discussion on Programming Models

Why parallelism?

Parallel computer architectures

Traditional models of parallel programming

Examples of parallel processing

Message Passing Interface (MPI)

MapReduce

Pregel, Dryad and GraphLab

Last 3 Sessions

Pregel

Pregel

- In this part, the following concepts of Pregel will be described:
  - Motivation for Pregel
  - The Pregel Computation Model
  - The Pregel API
  - Execution of a Pregel Program
  - Fault Tolerance in Pregel

Pregel

- In this part, the following concepts of Pregel will be described:
  - Motivation for Pregel
  - The Pregel Computation Model
  - The Pregel API
  - Execution of a Pregel Program
  - Fault Tolerance in Pregel

Motivation for Pregel

- How to implement algorithms to process large graphs?
  - Create a custom distributed infrastructure for each new algorithm
  - Rely on existing distributed computing platforms such as MapReduce
  - Use a single-computer graph algorithm library like BGL, LEDA, NetworkX etc.
  - Use a parallel graph processing system like Parallel BGL or CGMGraph

Motivation for Pregel

- How to implement algorithms to process large graphs?
  - Create a custom distributed infrastructure for each new algorithm

    Difficult!
  - Rely on existing distributed computing platforms such as MapReduce

    Inefficient and Cumbersome!
  - Use a single-computer graph algorithm library like BGL, LEDA, NetworkX etc.

    Too large to fit on single machine!
  - Use a parallel graph processing system like Parallel BGL or CGMGraph

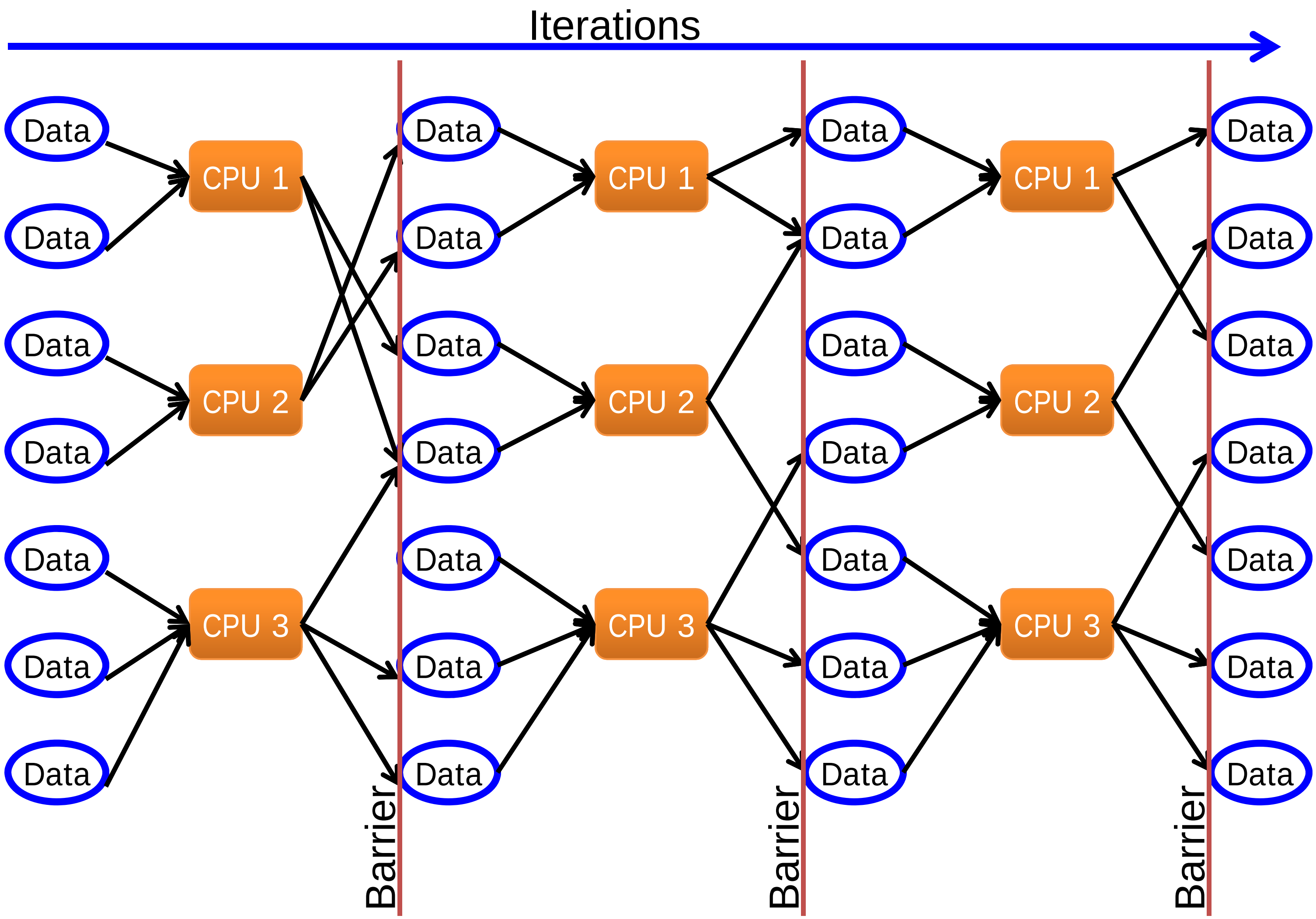
    Not suited for Large Scale Distributed Systems!

Pregel

- Pregel is a framework developed by Google. It provides:
  - High scalability
  - Fault-tolerance
  - Flexibility in expressing arbitrary graph algorithms

- Pregel is inspired by Valiant's Bulk Synchronous Parallel (BSP) model

Bulk Synchronous Parallel Model



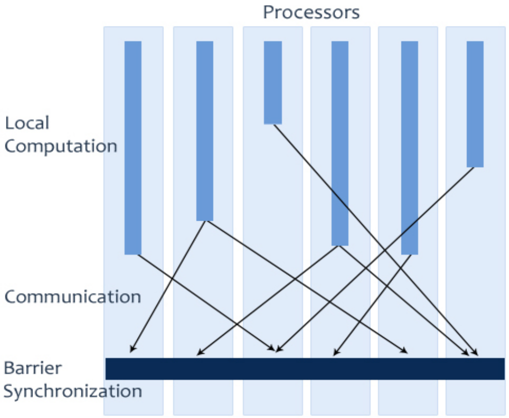
Pregel

- In this part, the following concepts of Pregel will be described:
  - Motivation for Pregel
  - The Pregel Computation Model
  - The Pregel API
  - Execution of a Pregel Program
  - Fault Tolerance in Pregel

Entities and Supersteps

- The computation is described in terms of vertices, edges and a sequence of iterations called *supersteps*

- You give Pregel a directed graph consisting of vertices and edges
  - Each vertex is associated with a modifiable user-defined value
  - Each edge is associated with a source vertex, value and a destination vertex

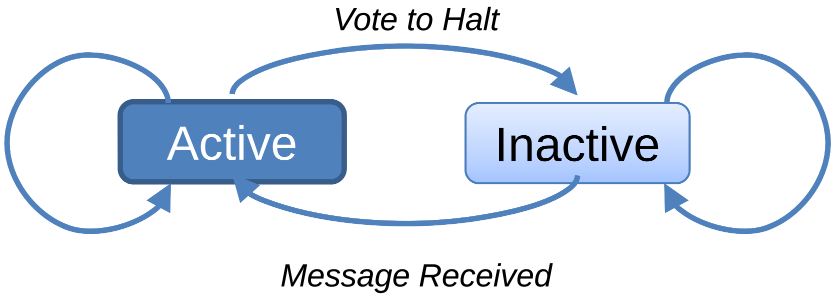


Vertical Structure of a Superstep

- During a superstep:
  - A user-defined function ***F*** is executed at each vertex ***V***
  - ***F*** can read messages sent to ***V*** in superstep ***S – 1*** and send messages to other vertices that will be received at superstep ***S + 1***
  - ***F*** can modify the state of ***V*** and its outgoing edges
  - ***F*** can change the topology of the graph

Algorithm Termination

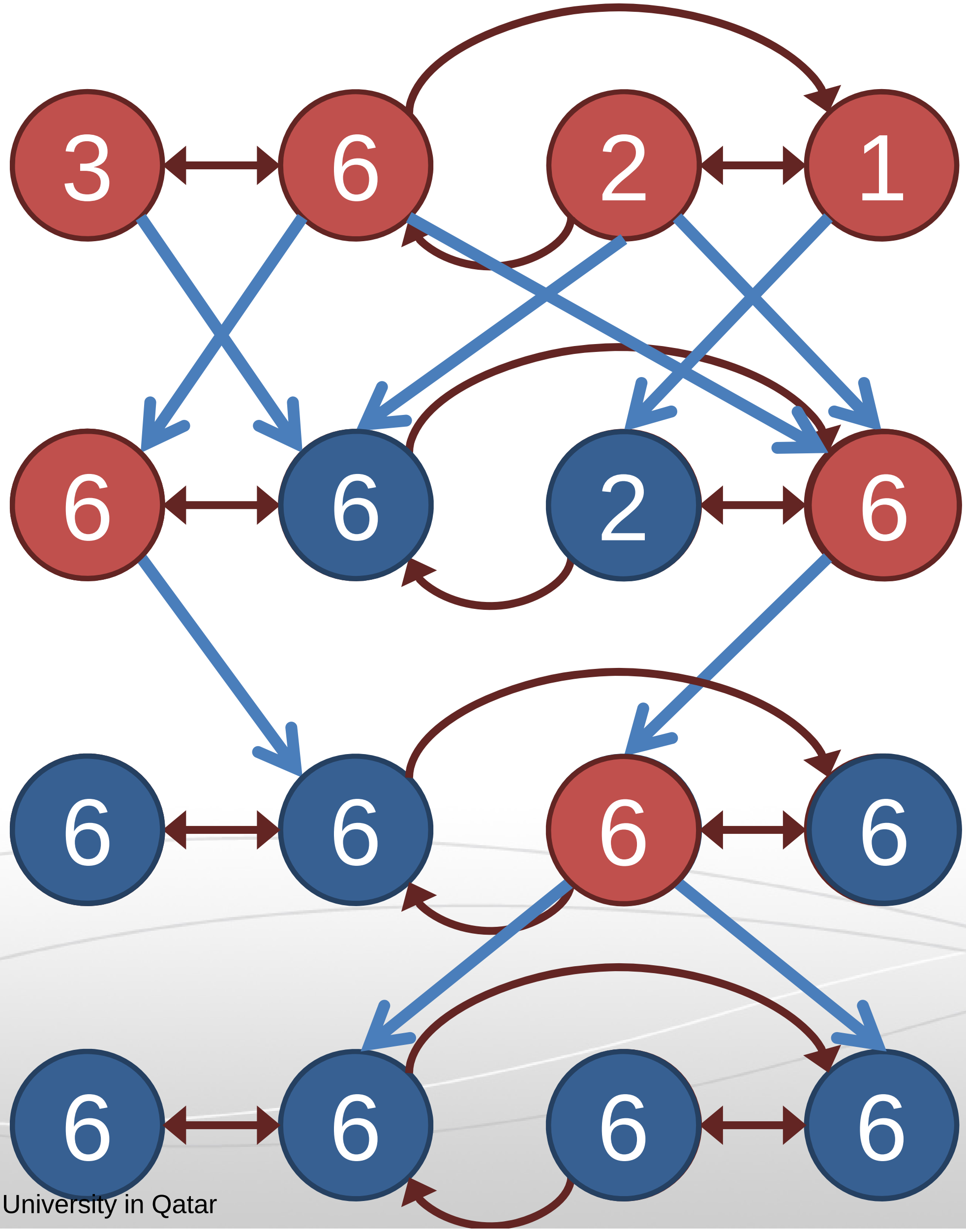
- Algorithm termination is based on every vertex voting to halt
  - In superstep 0, every vertex is active
  - All active vertices participate in the computation of any given superstep
  - A vertex deactivates itself by voting to halt and enters an inactive state
  - A vertex can return to active state if it receives an external message



Vertex State Machine

- Program terminates when all vertices are simultaneously inactive and there are no messages in transit

Finding the Max Value in a Graph



Blue Arrows are messages

Blue vertices have voted to halt

Pregel

- In this part, the following concepts of Pregel will be described:
  - Motivation for Pregel
  - The Pregel Computation Model
  - The Pregel API
  - Execution of a Pregel Program
  - Fault Tolerance in Pregel

The Pregel API in C++

- A Pregel program is written by subclassing the 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();
};
```

To define the types for vertices, edges and messages

Override the compute function to define the computation at each superstep

To get the value of the current vertex

To modify the value of the vertex

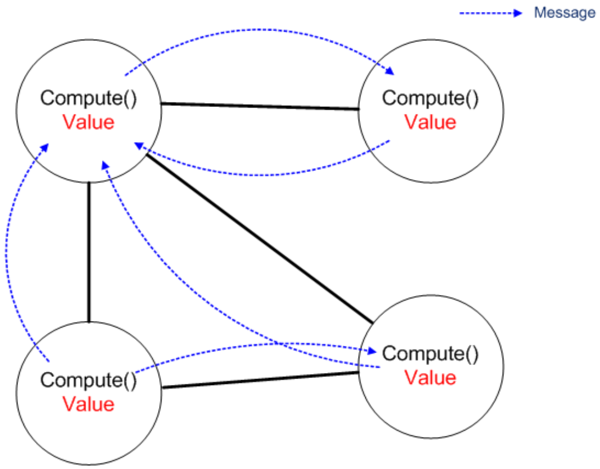
To pass messages to other vertices

Pregel Code for Finding the Max Value

```
Class MaxFindVertex
        : public Vertex<double, void, double> {
  public:
        virtual void Compute(MessageIterator* msgs) {
                int currMax = GetValue();
                SendMessageToAllNeighbors(currMax);
                for ( ; !msgs->Done(); msgs->Next()) {
                        if (msgs->Value() > currMax)
                                currMax = msgs->Value();
                }
                if (currMax > GetValue())
                        *MutableValue() = currMax;
                else VoteToHalt();
        }
};
```

Message Passing, Combiners, and Aggregators

- Messages can be passed from any vertex to any other vertex in the Graph



  - Any number of messages may be passed
  - Message order is not guaranteed
  - Messages will not be duplicated

- Combiners can be used to reduce the number of messages passed between supersteps

- Aggregators are available for reduction operations such as sum, min, max etc.

Topology Mutations, Input and Output

- The graph structure can be modified during any superstep
  - Vertices and edges can be added or deleted
  - Conflicts are handled using partial ordering of operations
  - User-defined handlers are also available to manage conflicts

- Flexible input and output formats
  - Text File
  - Relational Database
  - Bigtable Entries

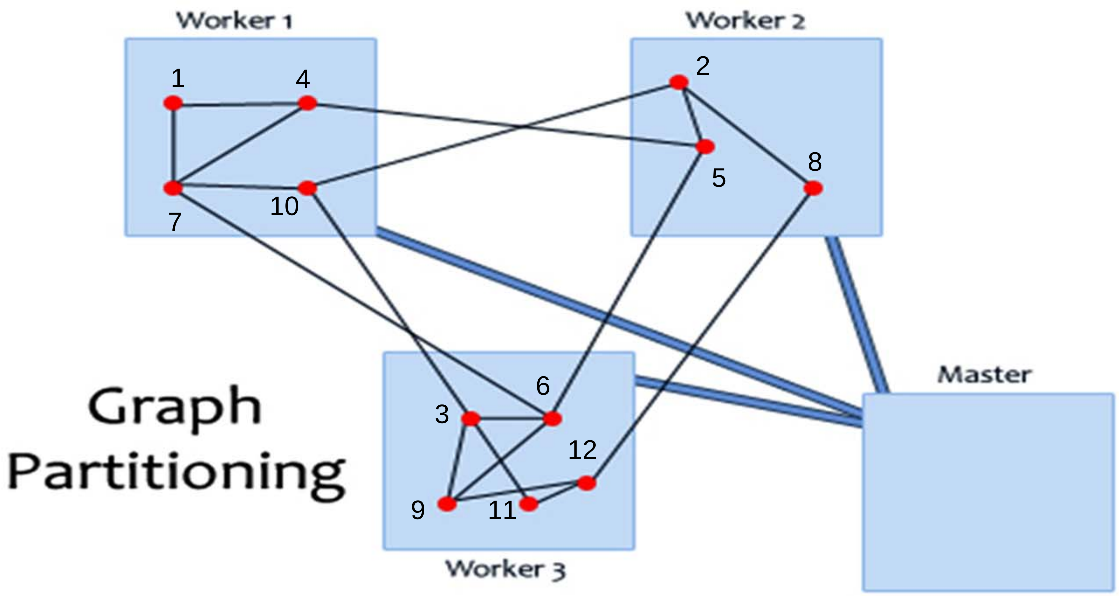
- Interpretation of input is a “pre-processing” step separate from graph computation
  - Custom formats can be created by sub-classing the Reader and Writer classes

Pregel

- In this part, the following concepts of Pregel will be described:
    - Motivation for Pregel
    - The Pregel Computation Model
    - The Pregel API
    - Execution of a Pregel Program
    - Fault Tolerance in Pregel

Graph Partitioning

- The input graph is divided into partitions consisting of vertices and outgoing edges
  - Default partitioning function is ***hash(ID) mod N***, where ***N*** is the # of partitions
  - It can be customized



Execution of a Pregel Program

- Steps of Program Execution:

1. Copies of the program are distributed across all workers

   1.1 One copy is designated as a master

2. Master partitions the graph and assigns workers their respective partition(s) along with portions of the input
3. Master coordinates the execution of supersteps and delivers messages among vertices
4. Master calculates the number of inactive vertices after each superstep and signals workers to terminate if all vertices are inactive and no messages are in transit
5. Each worker may be instructed to save its portion of the graph

Pregel

- In this part, the following concepts of Pregel will be described:
  - Motivation for Pregel
  - The Pregel Computation Model
  - The Pregel API
  - Execution of a Pregel Program
  - Fault Tolerance in Pregel

Fault Tolerance in Pregel

- Fault tolerance is achieved through checkpointing
  - At the start of every superstep the master may instruct the workers to save the state of their partitions in a stable storage
- Master uses *ping* messages to detect worker failures
- If a worker fails, the master reassigns corresponding vertices and input to another available worker and restarts the superstep
  - The available worker reloads the partition state of the failed worker from the most recent available checkpoint

Next Class

Dryad and GraphLab