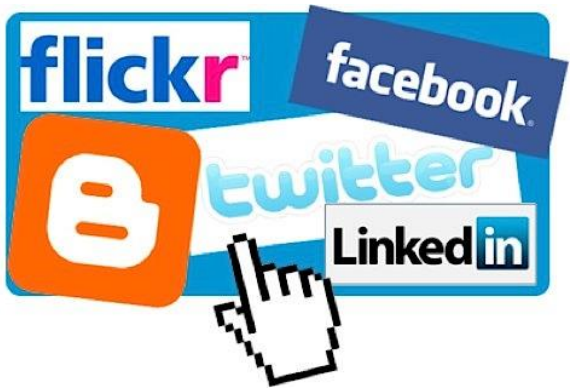
Programming for Data Science at Scale

Distributed Graph Processing

THE UNIVERSITY *of* EDINBURGH

Amir Shaikhha, Fall 2024

Graphs are everywhere



Social networks

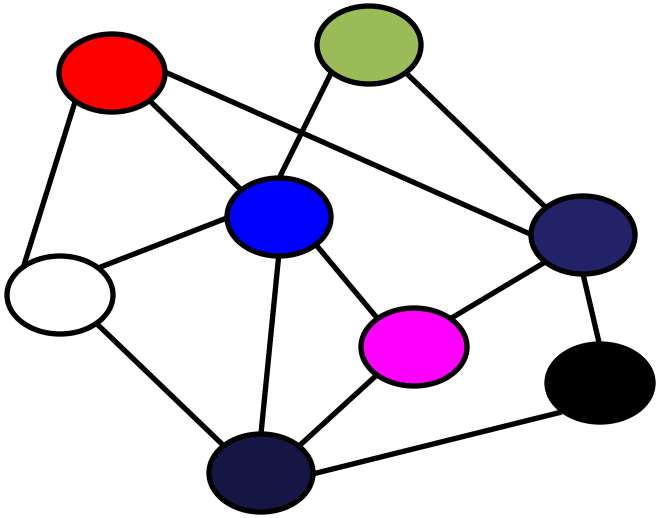
Web graph/search engine

E-commerce

Maps

Computational biology

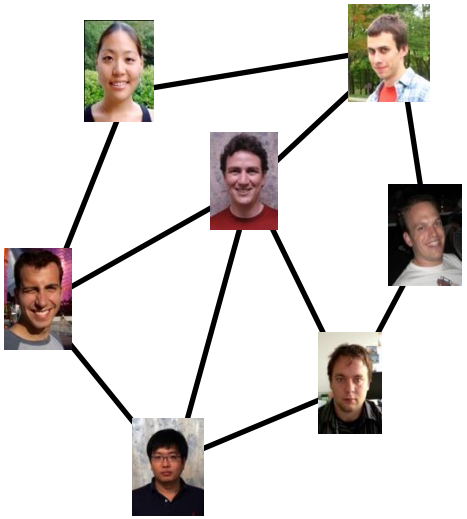
Example: PageRank



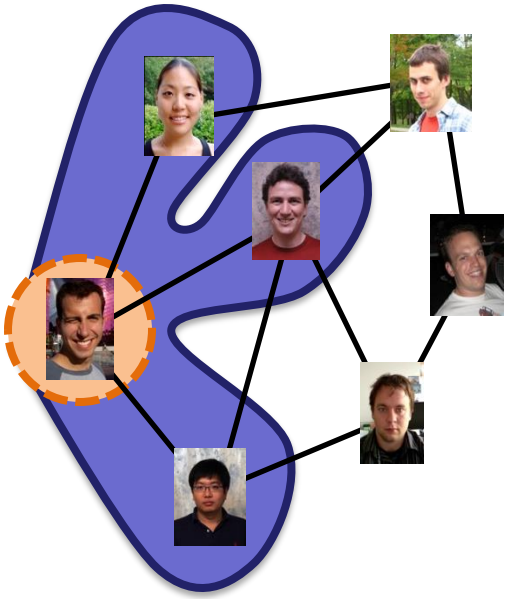
Web graph

Graph Parallel Algorithms

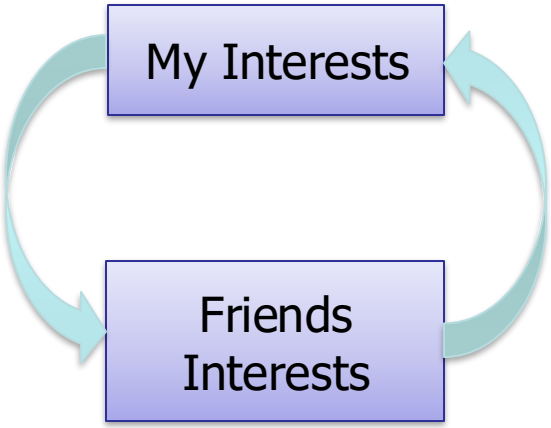
Dependency **Graph**



Local Updates



Iterative Computation



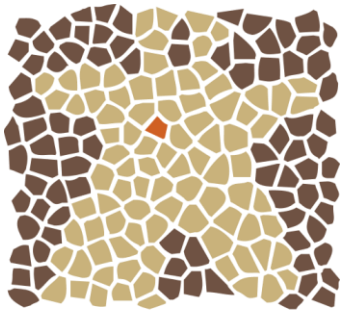
Many graph algorithms

- 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

Graph processing framework

Pregel

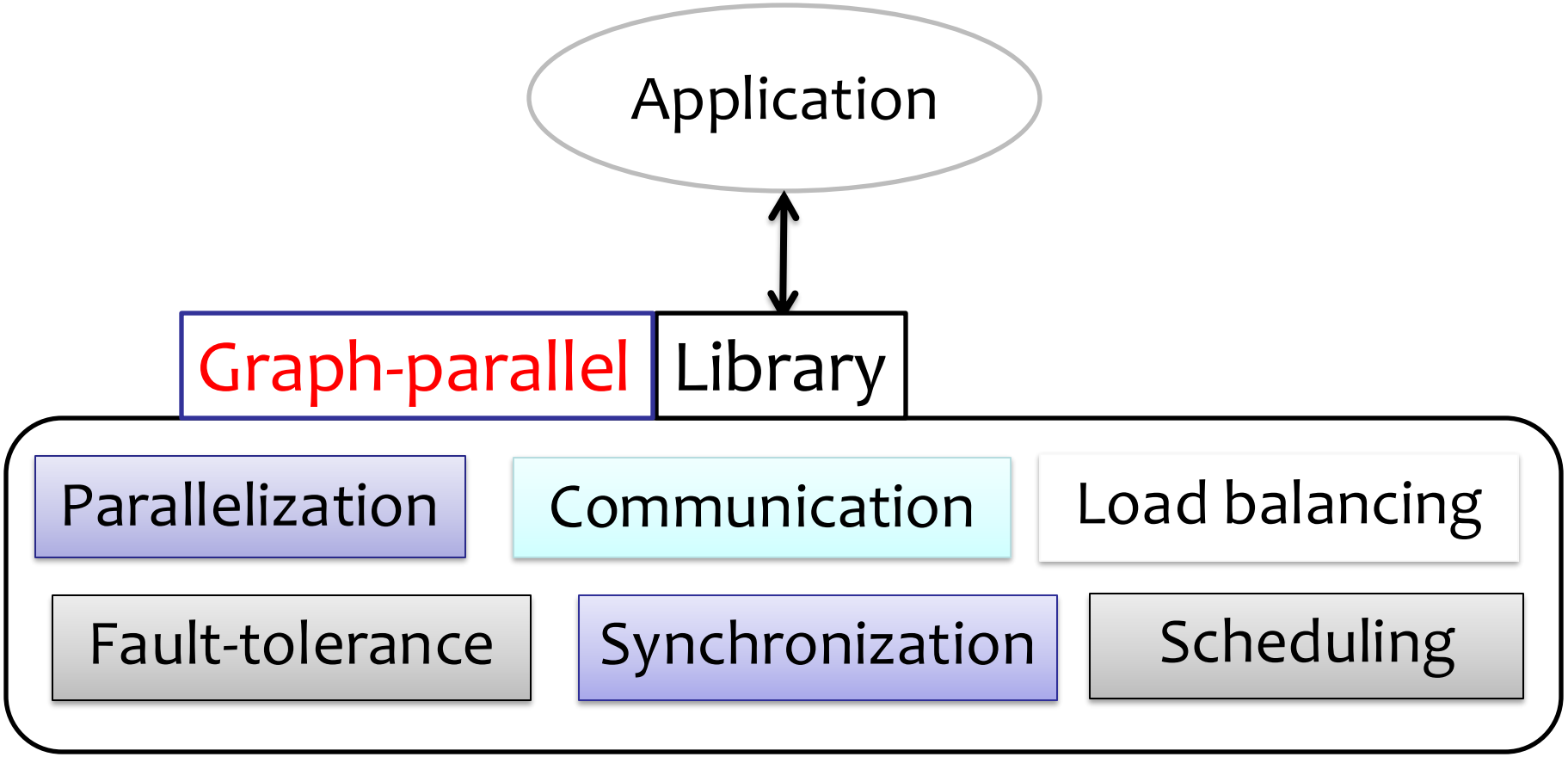
oogle



APACHE

GIRAPH

Graph processing frameworks



Why do we need a new framework?

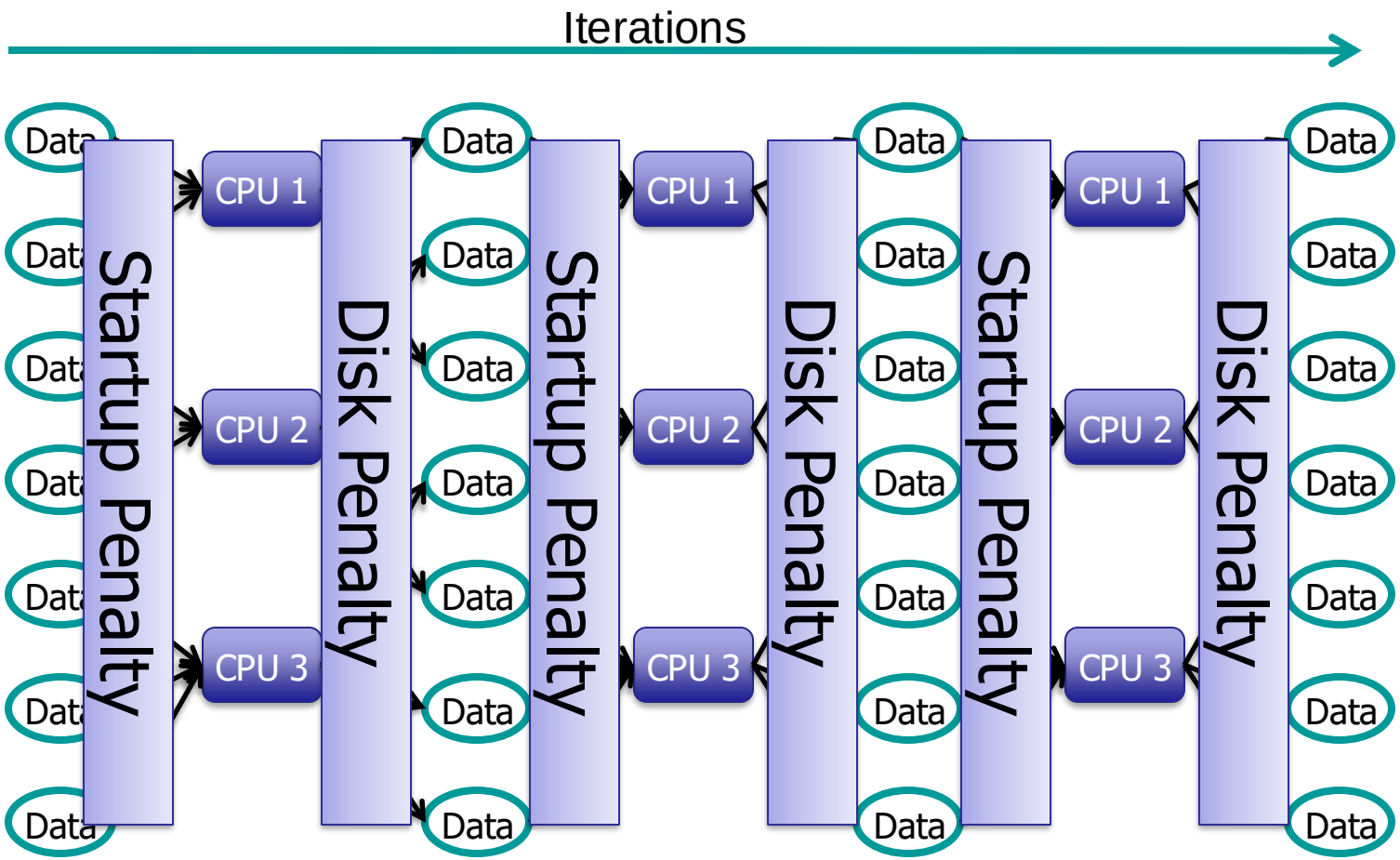
- Why don’t we just MapReduce?
- How would you implement Graph processing in MapReduce?

Data Dependencies are Difficult

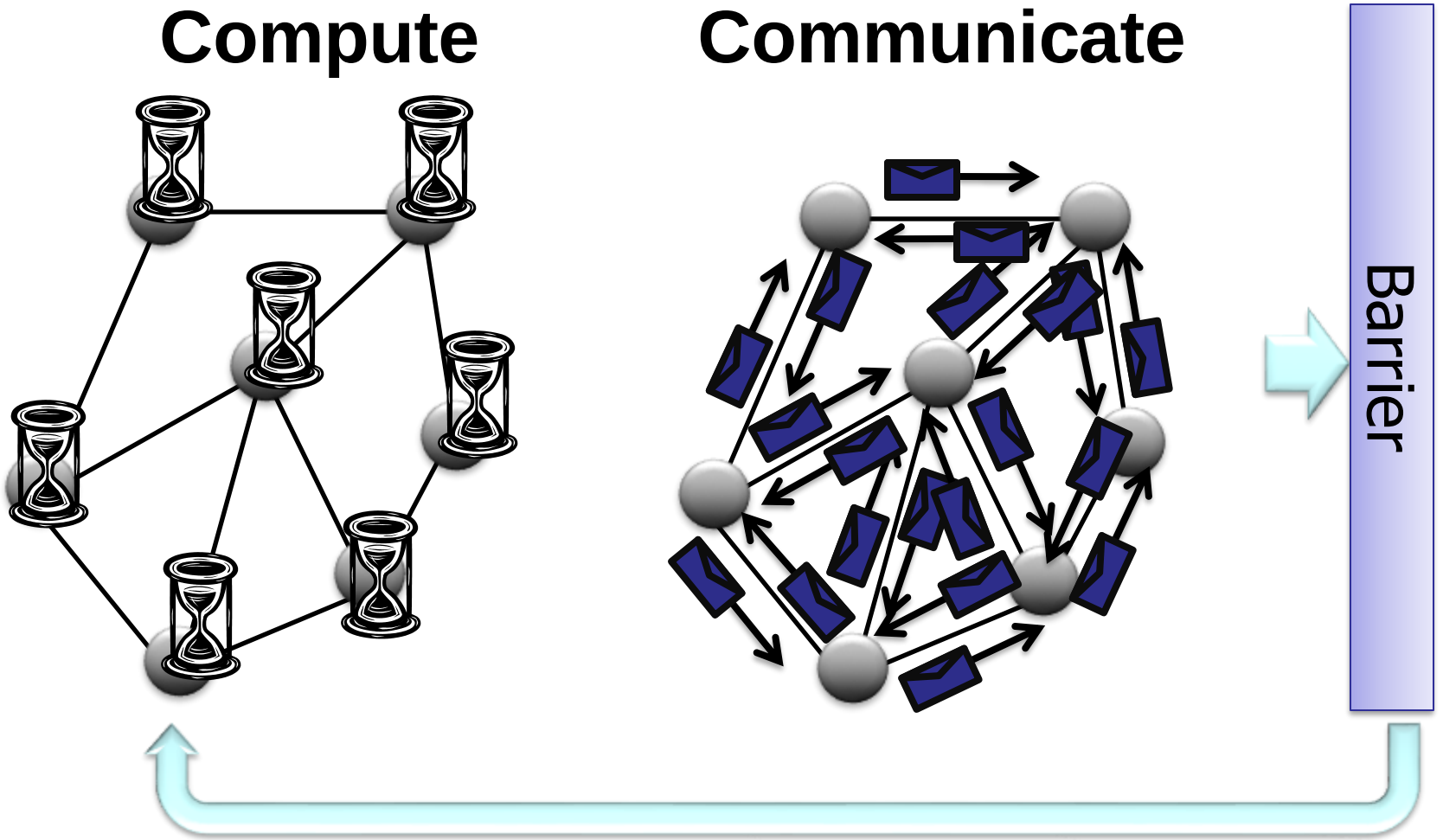
- Difficult to express dependent data in MR
  - Substantial data transformations
  - User managed graph structure
  - Costly data replication

Iterative Computation is Difficult

- System is not optimized for iteration:



Pregel: Bulk Synchronous Parallel



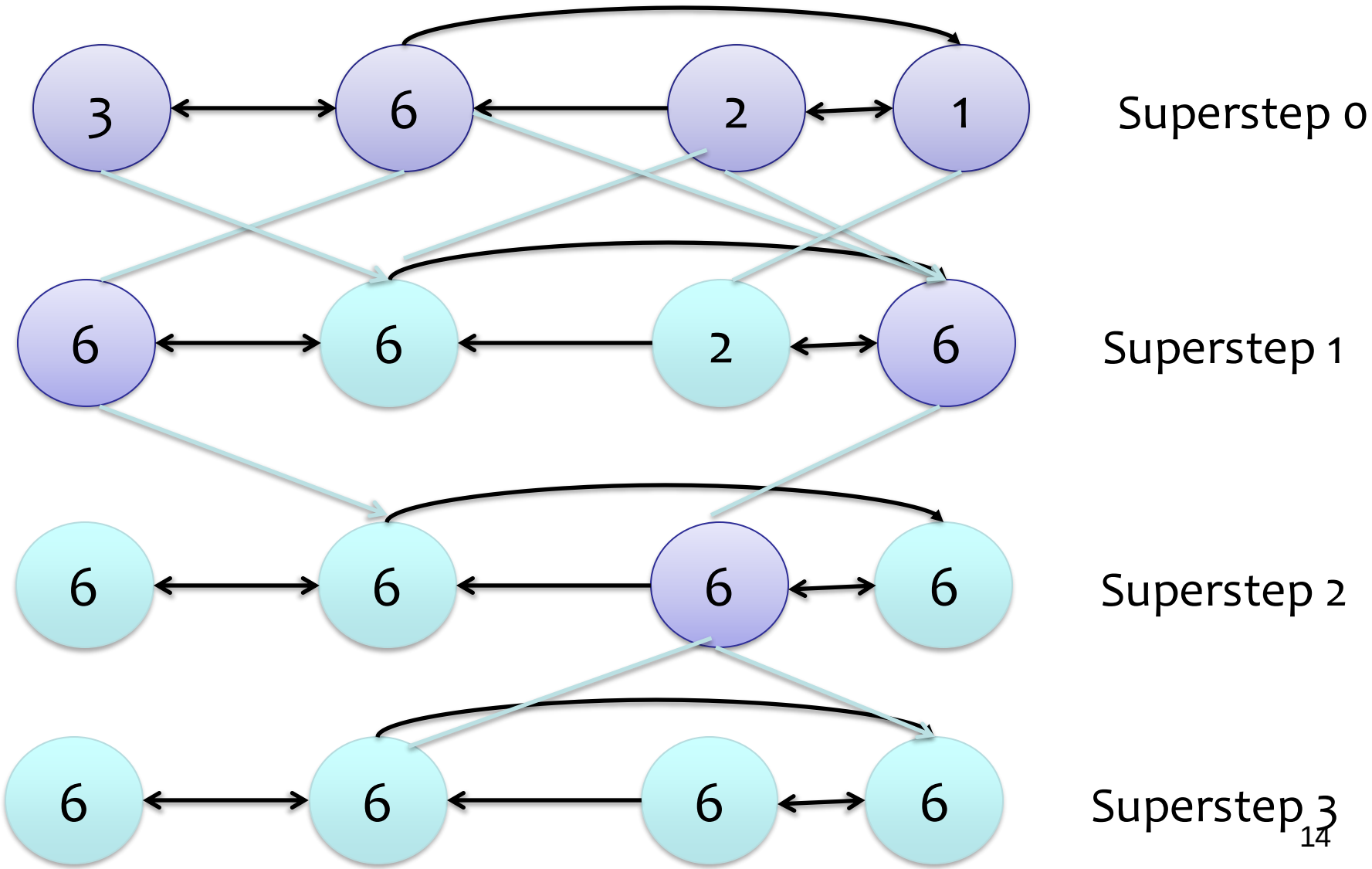
http://dl.acm.org/citation.cfm?id=1807184

Vertex centric API

Programming API

```
Class Vertex{
        //Main methods
        Compute(MessageIterator *msgs);
        SendMsgTo(dest, msg);
        VoteToHalt();

        //Auxiliary methods
        GetValue();
        MutableValue();
        GetOutEdgeIterator();
        SuperStep();
}
```

Example: maximum value



Example: PageRank

$$PageRank\ of\ site = \sum \frac{PageRank\ of\ inbound\ link}{Number\ of\ links\ on\ that\ page}$$

OR

$$PR(u) = (1-d) + d \times \sum \frac{PR(v)}{N(v)}$$

Rank of webpage *i*

Weighted sum of neighbors' ranks

Iterate until it converges

Example: PageRank

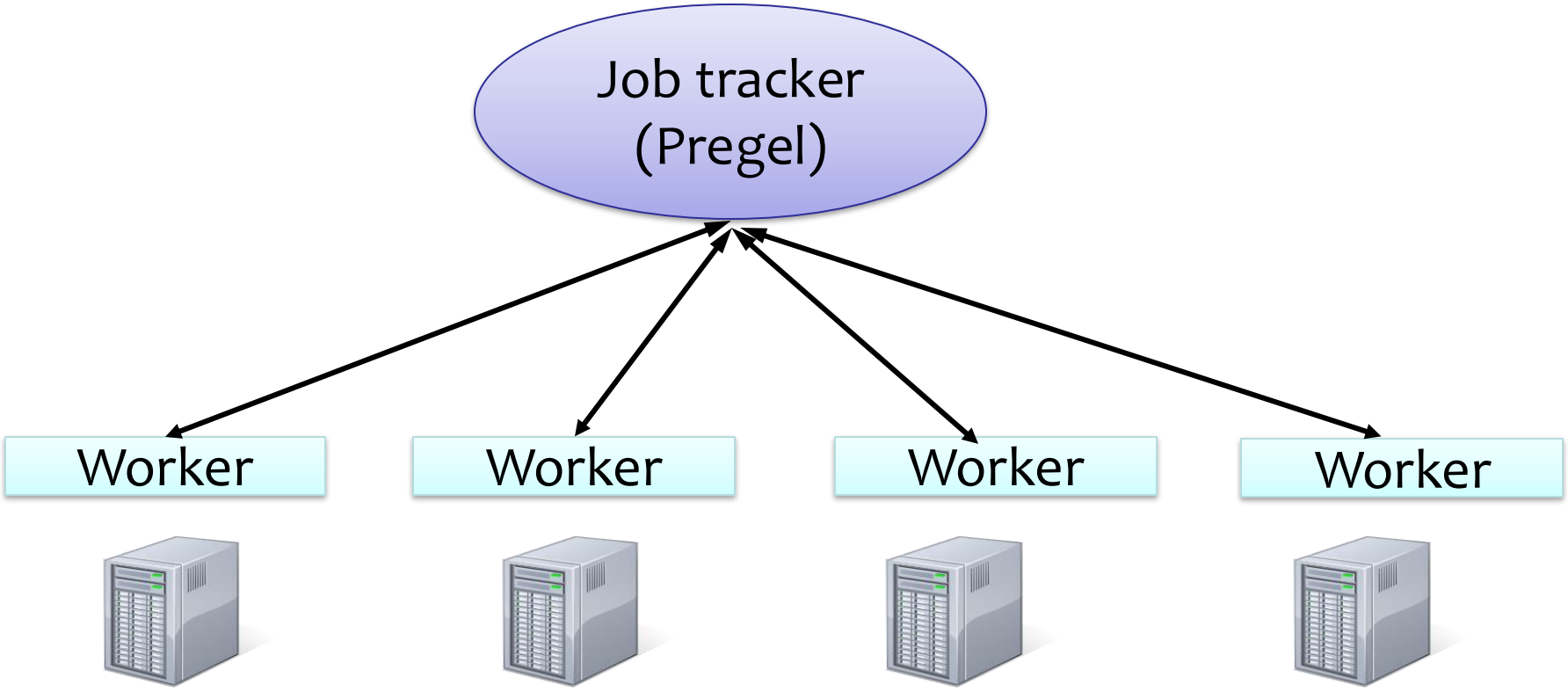
```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```

Additional features

- Combiners
- Aggregators
- Topology mutations
  - Partial ordering
    - Removal first (Edge → vertex removal)
    - Addition (vertex → edge addition)
  - Handlers
    - User-defined functions to resolve conflicts
- Input/output
  - File, GFS, BigTable, etc.

Implementation

Master

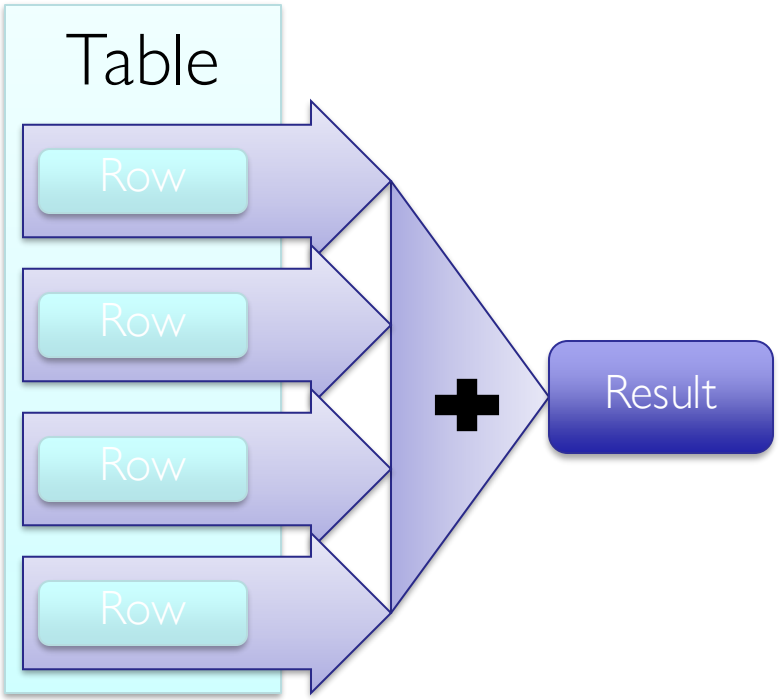


Fault tolerance

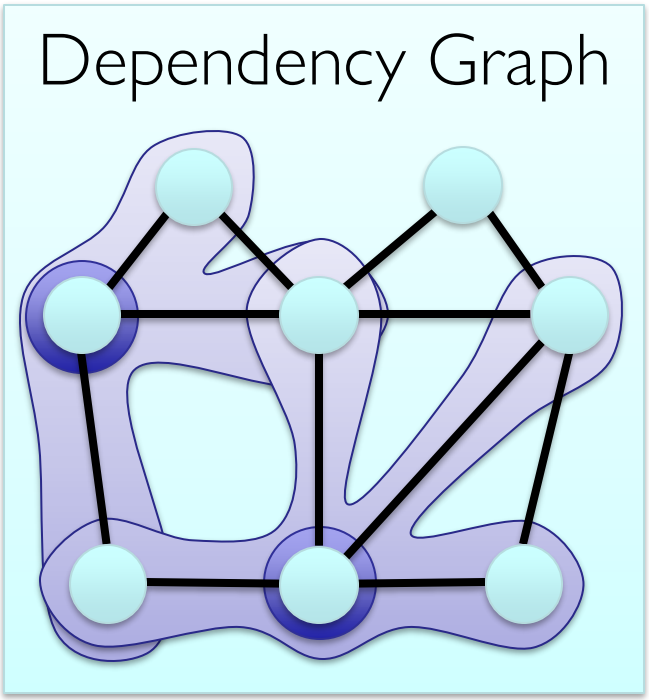
- Achieved through checkpointing
- At the beginning of a super-step, master instructs the workers to take a check-point
- When a worker fails --- the master re-assigns the partition to a new worker, and restarts from the latest checkpoint

Distributed Collections Views

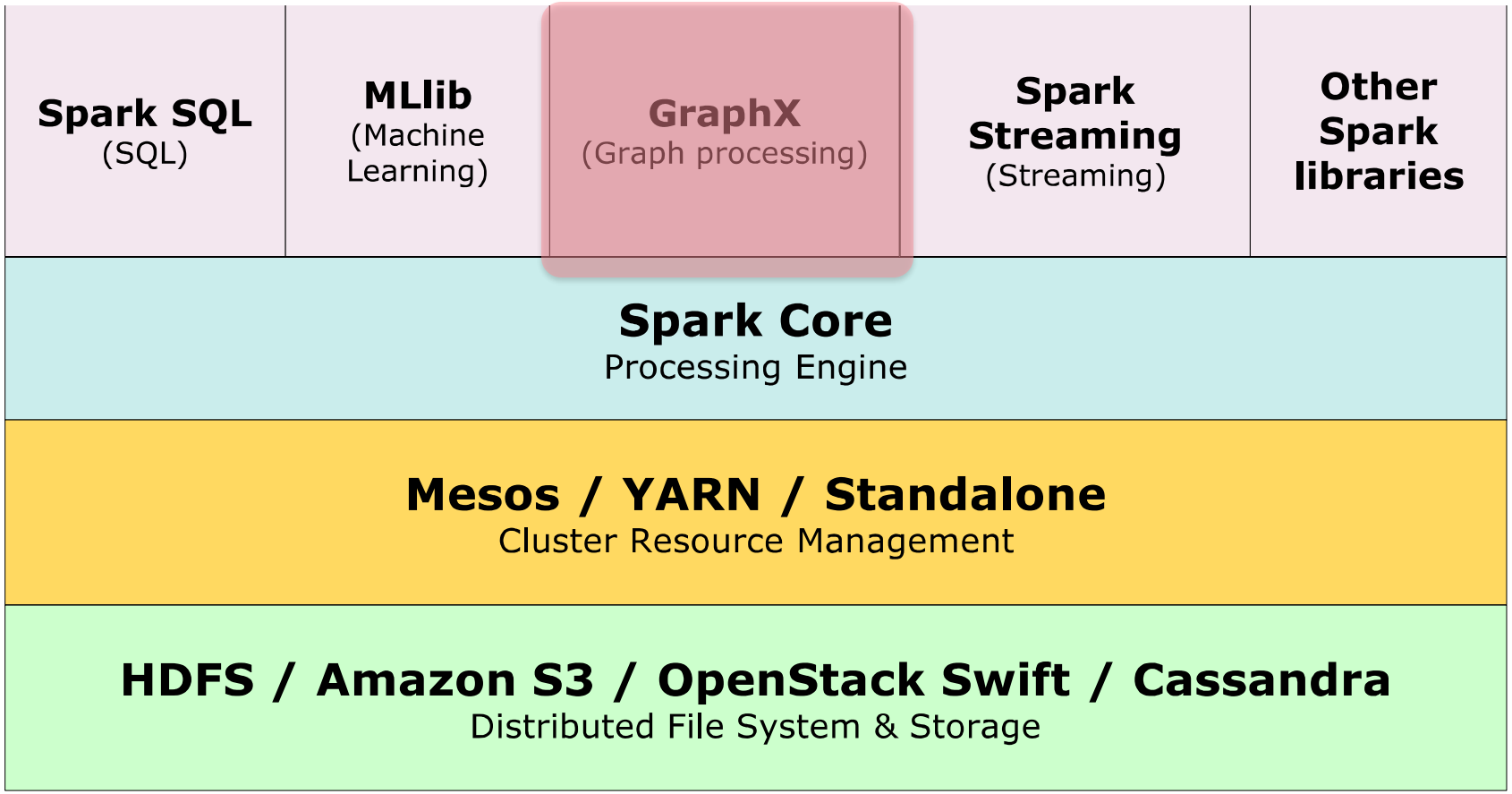
Table View



Graph View

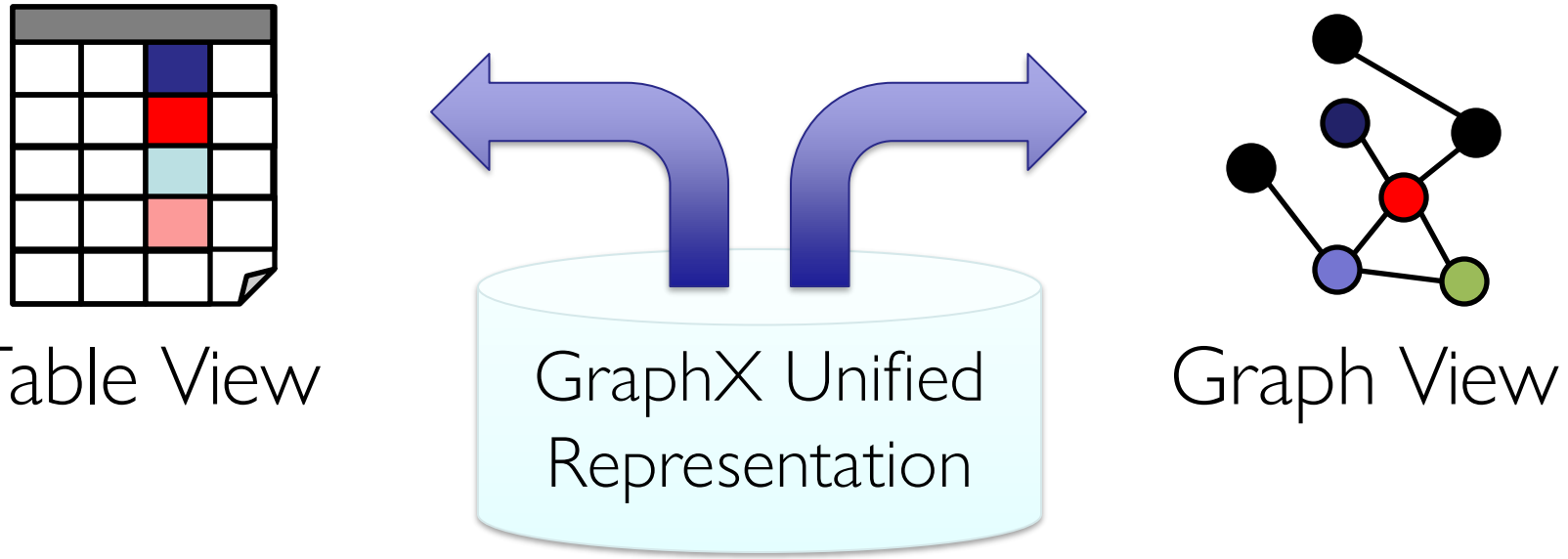


Spark Software Stack



GraphX

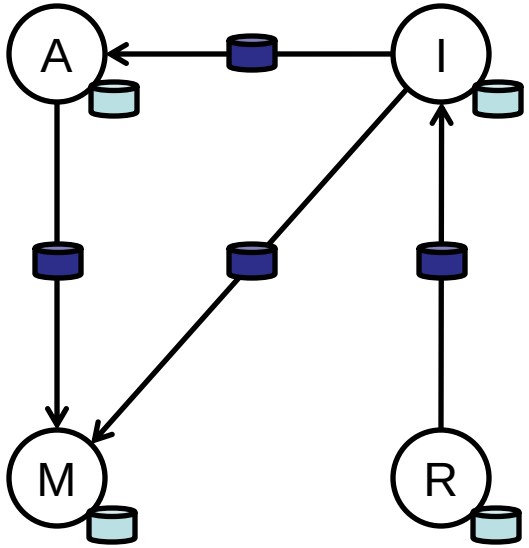
- Tables and Graphs are composable views of the same physical data



- Each view has its own operators that exploit the semantics of the view to achieve efficient execution

View a Graph as a Table

Property Graph



Vertex Property Table

Id	Property (V)
A	(P1, P4)
M	(P2,P4)
I	(P3, P4)
R	(P3, P4)

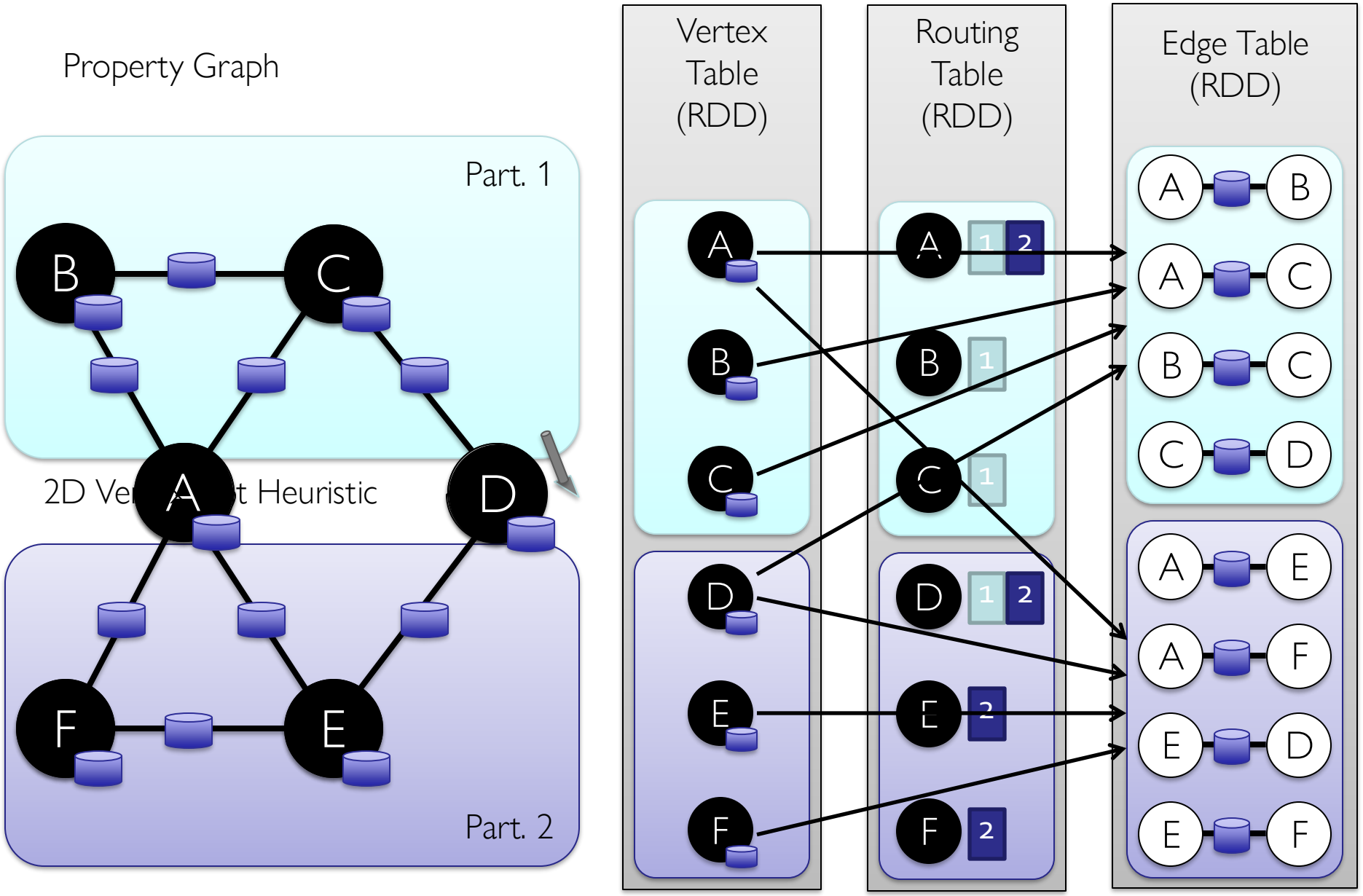
Edge Property Table

SrcId	DstId	Property (E)
A	M	P5
I	M	P6
I	A	P7
R	I	P8

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 mapV(m: (Id, V) => T ): Graph[T,E]
    def mapE(m: Edge[V,E] => T ): Graph[V,T]
    // Joins ----------------------------------------
    def joinV(tbl: Table [(Id, T)]): Graph[(V, T), E ]
    def joinE(tbl: Table [(Id, Id, T)]): Graph[V, (E, T)]
    // Computation ----------------------------------
    def mrTriplets(mapF: (Edge[V,E]) => List[(Id, T)],
                   reduceF: (T, T) => T):Graph[T, E]
}
```

Distributed Graphs as RDDs



Summary

- Graph processing with Pregel/Giraph
  - Bulk Synchronous Programming (BSP) model
- Graph processing on Spark with GraphX
- Resources:
  - Giraph: http://giraph.apache.org/
  - GraphX: https://spark.apache.org/graphx/
  - GraphLab: http://graphlab.org/
  - Okapi: http://grafos.ml/

Resources

- Compulsory reading:
  - Pregel [SIGMOD'10] https://kowshik.github.io/JPregel/pregel_paper.pdf
- Recommended reading
  - GraphX [OSDI'14]
    - Graph processing framework built on top of Spark
  - GraphLab [OSDI'12]
    - Edge-centric graph processing framework

QUESTIONS?