Parameter Servers

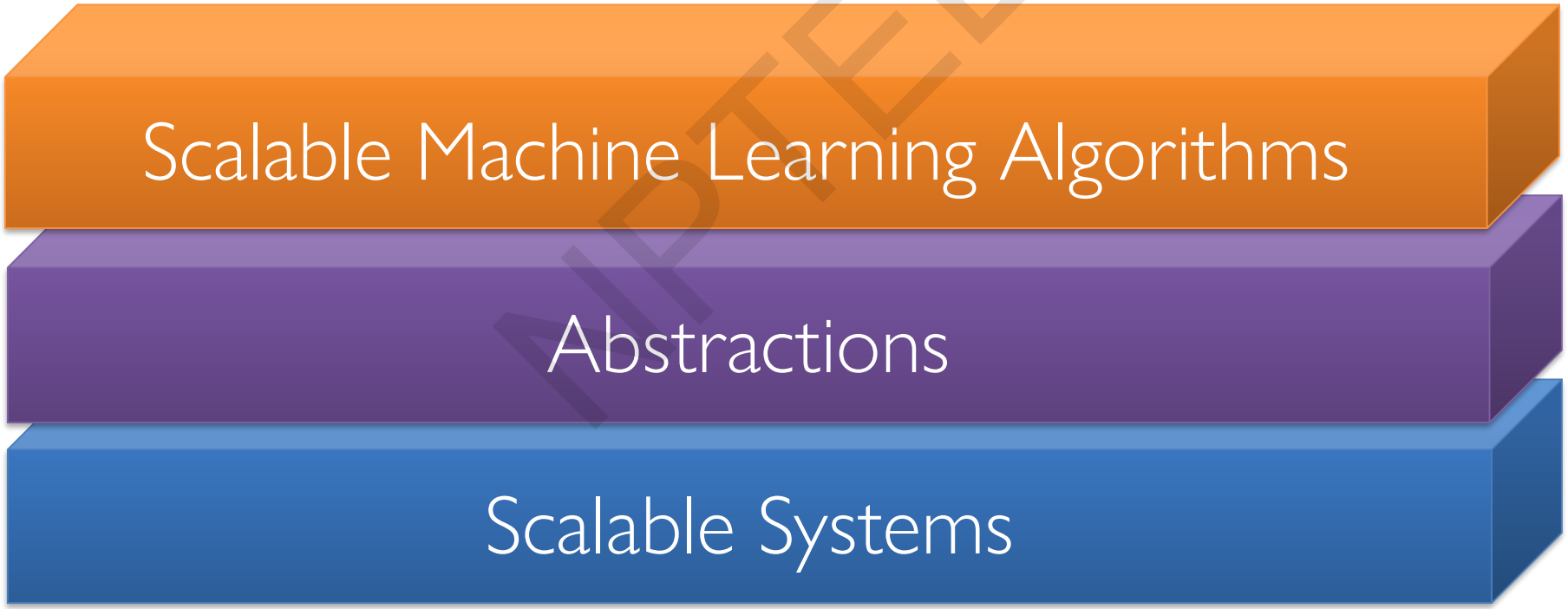
Dr. Rajiv Misra
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss the '**Parameters Servers**', also discuss its Stale Synchronous Parallel Model.

ML Systems



ML Systems Landscape

Dataflow Systems

Hadoop, Spark

Graph Systems

GraphLab, Tensorflow

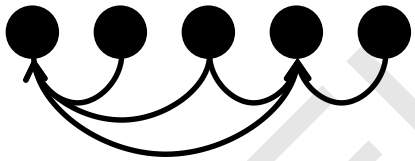
Shared Memory Systems

Bosen, DMTK, ParameterServer.org

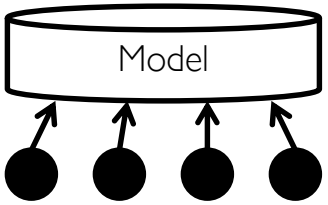
ML Systems Landscape

Dataflow Systems

Graph Systems



Shared Memory Systems



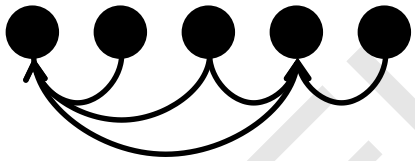
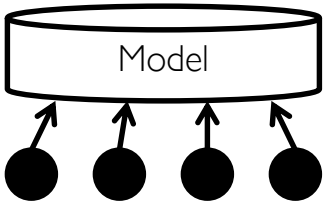
Algorithms

Hadoop, Spark

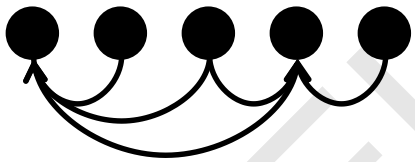
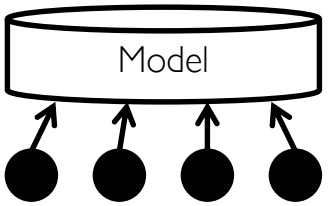
GraphLab, Tensorflow

Bosen, DMTK, ParameterServer.org

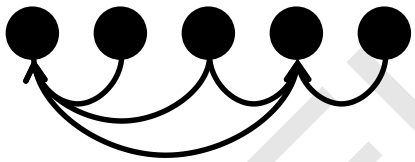
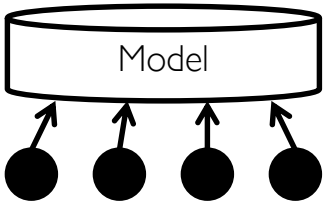
ML Systems Landscape

Dataflow Systems	Graph Systems	Shared Memory Systems
		
Naïve Bayes, Rocchio	Graph Algorithms, Graphical Models	SGD, Sampling *[NIPS'09, NIPS'13]*
Hadoop, Spark	GraphLab, Tensorflow	Bosen, DMTK, ParameterServer.org

ML Systems Landscape

Dataflow Systems	Graph Systems	Shared Memory Systems
		
Naïve Bayes, Rocchio	Graph Algorithms, Graphical Models	SGD, Sampling *[NIPS'09, NIPS'13]*
Abstractions		
Hadoop & Spark	GraphLab, Tensorflow	Bosen, DMTK, ParameterServer.org

ML Systems Landscape

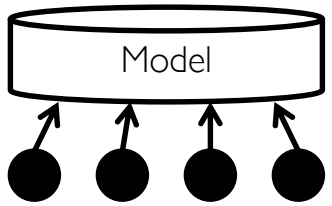
Dataflow Systems	Graph Systems	Shared Memory Systems
		
Naïve Bayes, Rocchio	Graph Algorithms, Graphical Models	SGD, Sampling *[NIPS'09, NIPS'13]*
PIG, GuineaPig, …	Vertex-Programs	Parameter Server [VLDB'10]
Hadoop & Spark	GraphLab, Tensorflow	Bosen, DMTK, ParameterServer.org

ML Systems Landscape

Dataflow Systems

Graph Systems

Shared Memory Systems



Simple case: Parameters of the ML system are stored in a **distributed** hash table that is accessible thru the **network**

[NIPS'09, NIPS'13]

Param Servers used in Google, Yahoo, ….
Academic work by Smola, Xing, …

Parameter Server
[VLDB'10]

Petuum closes $93 Million Series B round led by SoftBank with participation from previous investor Advantech Capital, becoming one of the highest funded early-stage Artificial Intelligence and Machine Learning startups

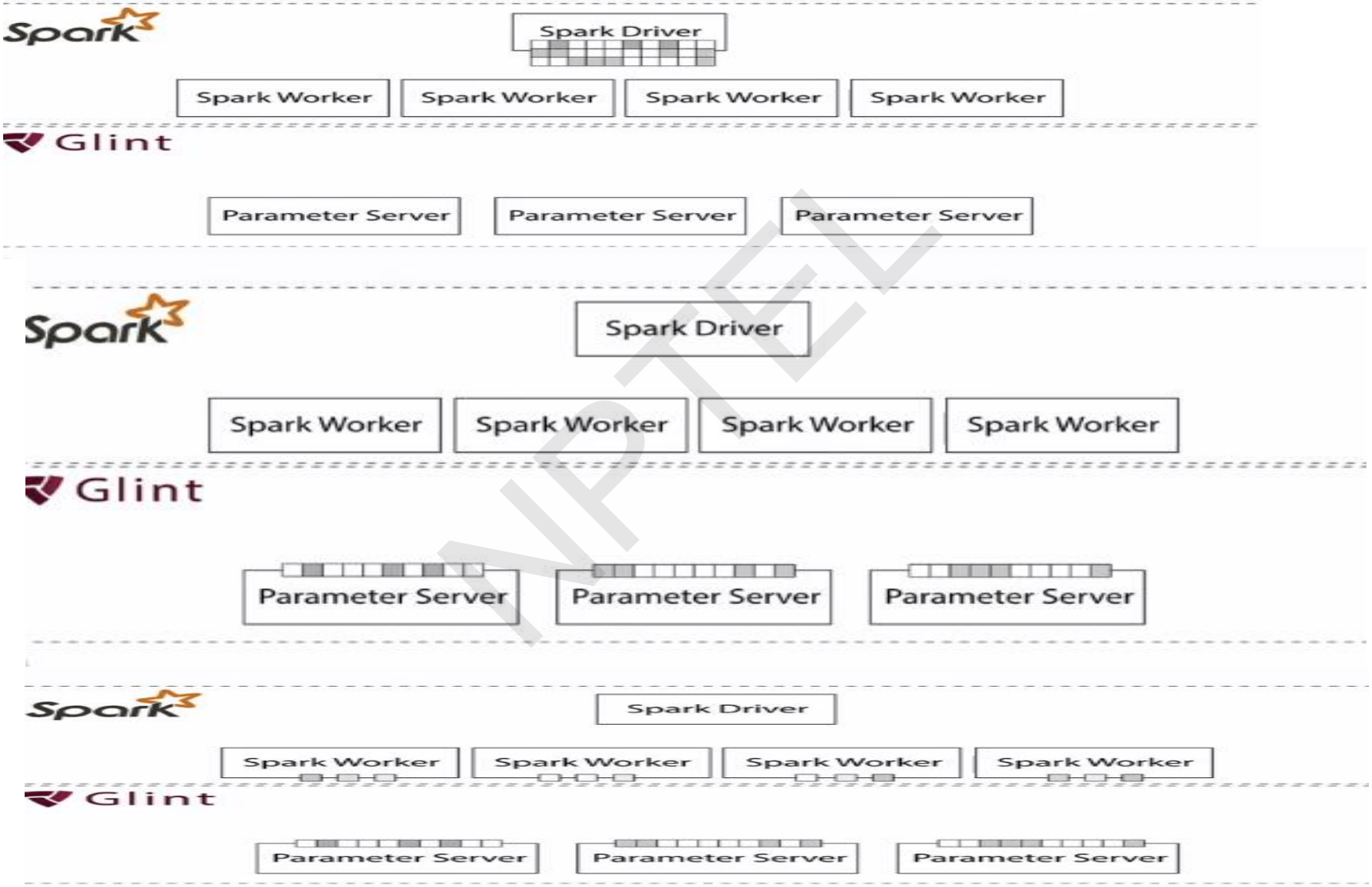
Parameter Server

- A machine learning framework
- Distributes a model over multiple machines
- Offers two operations:
  - **Pull**: query parts of the model
  - **Push**: update parts of the model
- Machine learning update equation

$$w_i \leftarrow w_i + \Delta$$

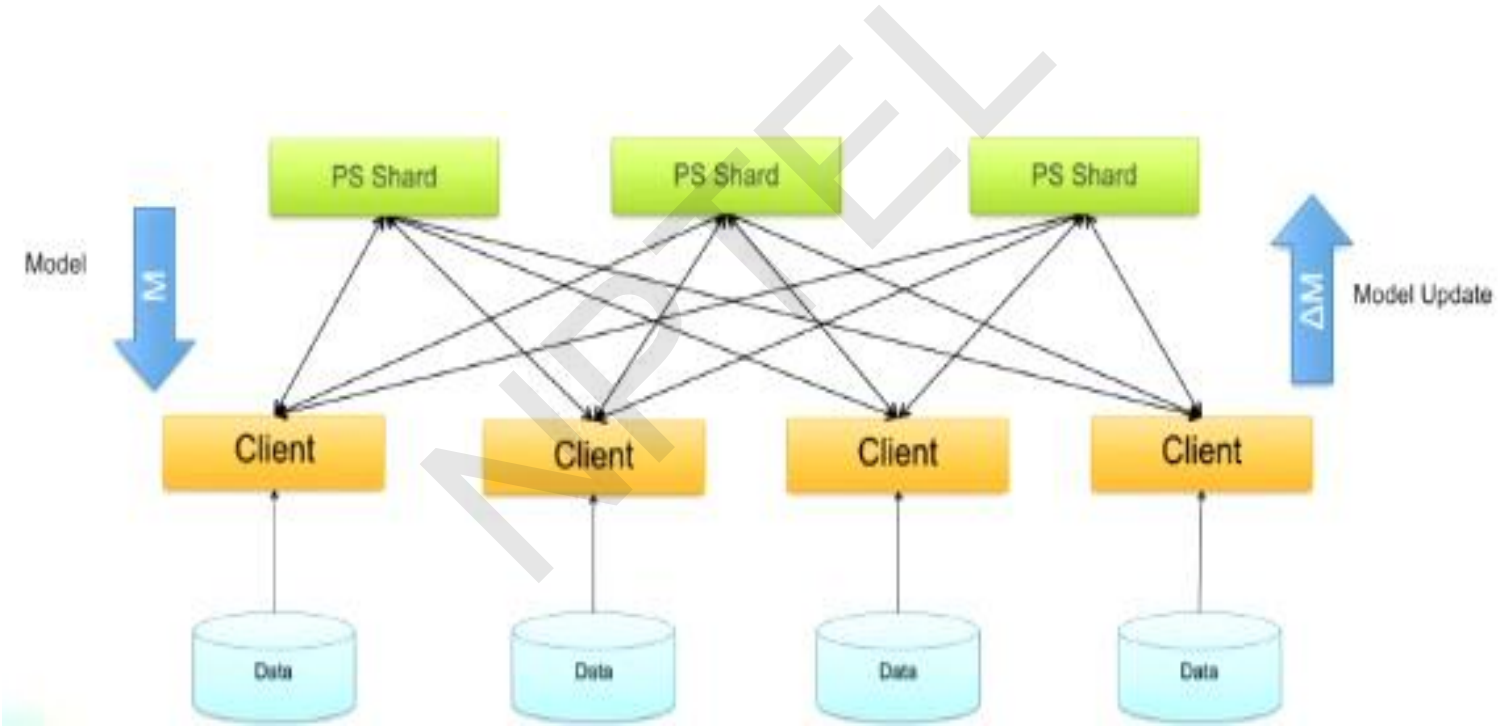
- (Stochastic) gradient descent
- Collapsed Gibbs sampling for topic modeling
- Aggregate **push** updates via addition (+)

Spark with Parameter Servers

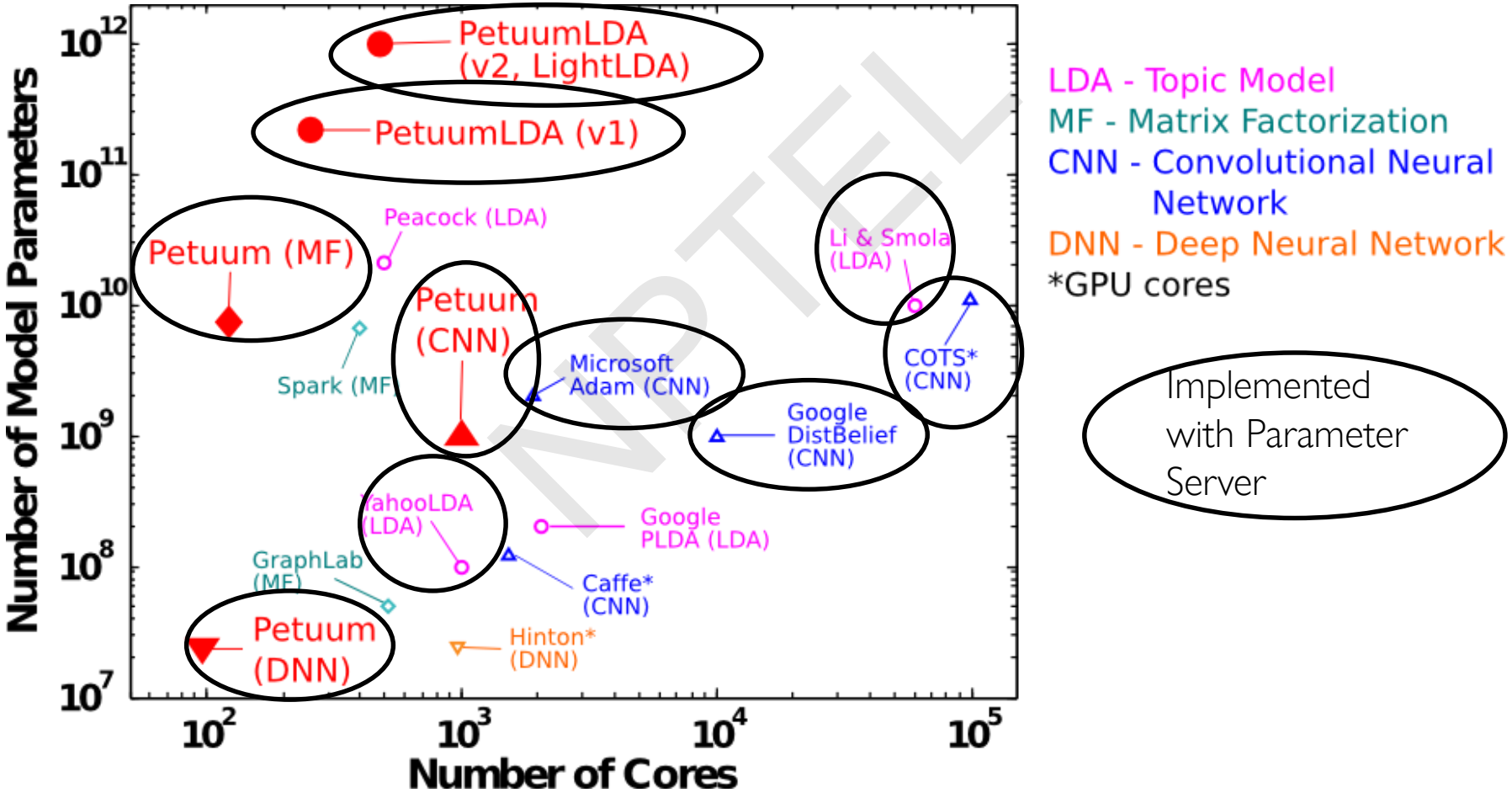


Parameter Server (PS)

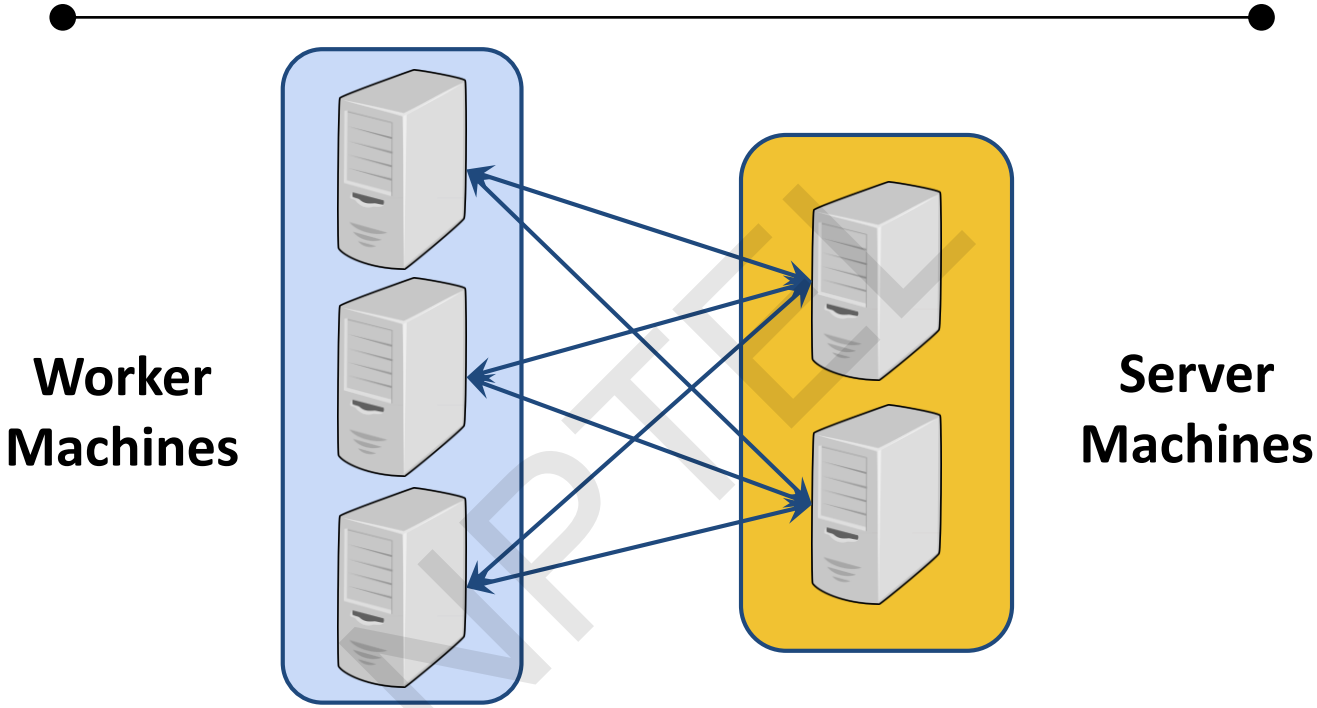
- Training state stored in PS shards, asynchronous updates



Parameter Servers Are Flexible



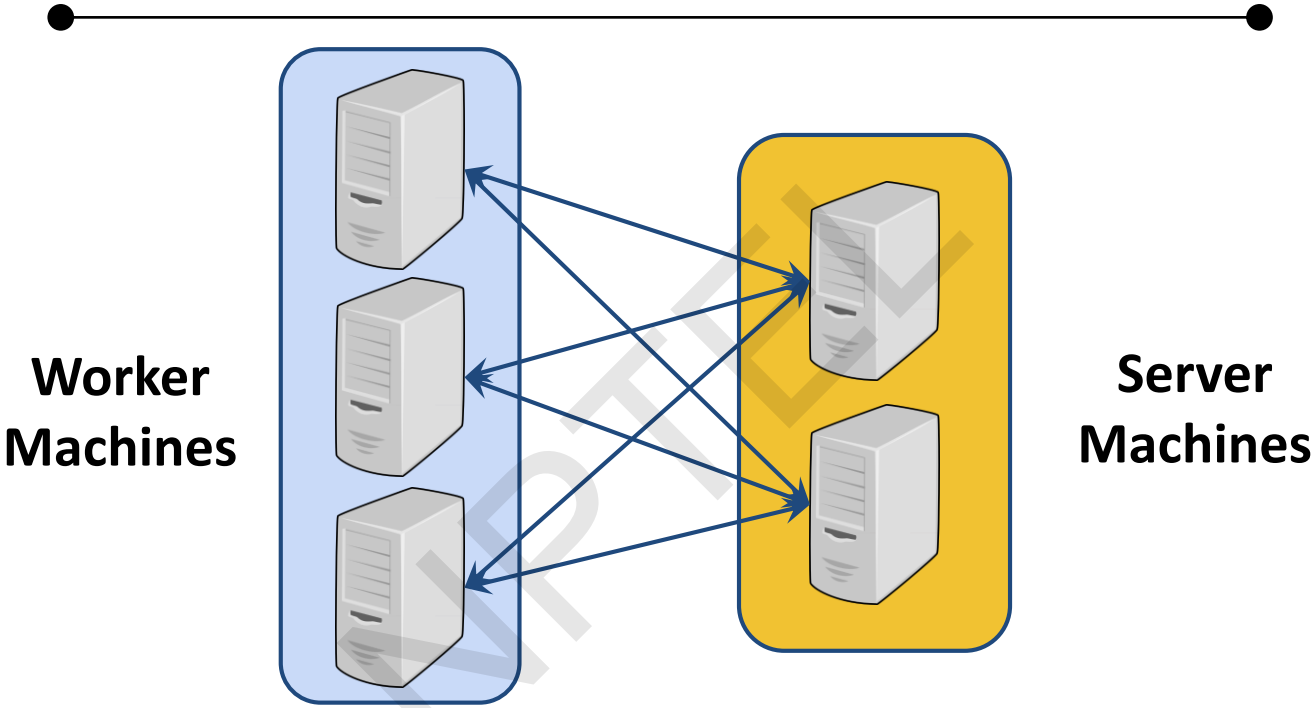
Parameter Server (PS)



- Model parameters are stored on PS machines and accessed via key-value interface (distributed shared memory)
- **Extensions**: multiple keys (for a matrix); multiple "channels" (for multiple sparse vectors, multiple clients for same servers, …)

[Smola et al 2010, Ho et al 2013, Li et al 2014]

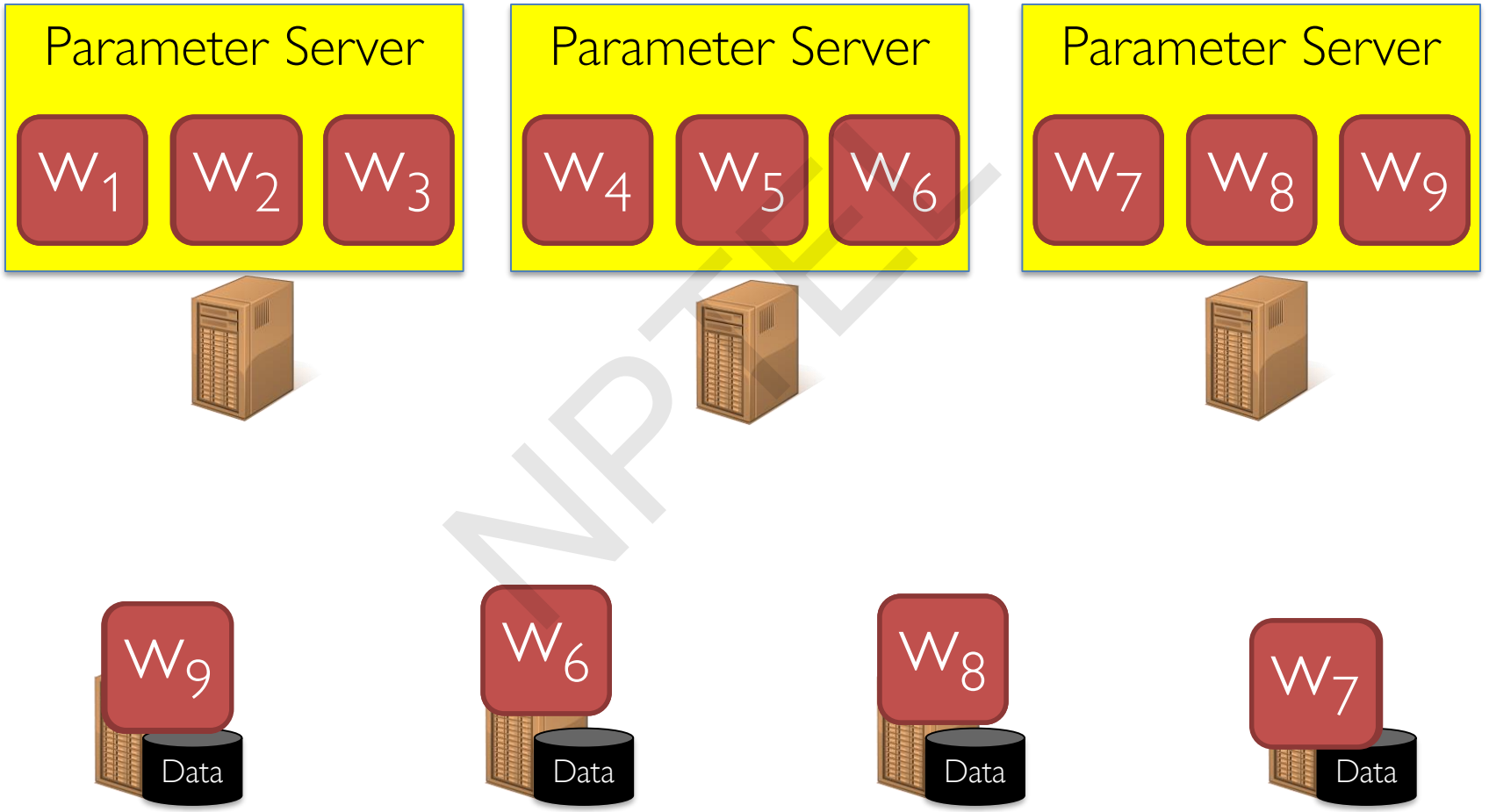
Parameter Server (PS)



- **Extensions:** push/pull interface to send/receive most recent copy of (subset of) parameters, blocking is **optional**
- **Extension:** can block until push/pulls with clock < $(t - \tau)$ complete

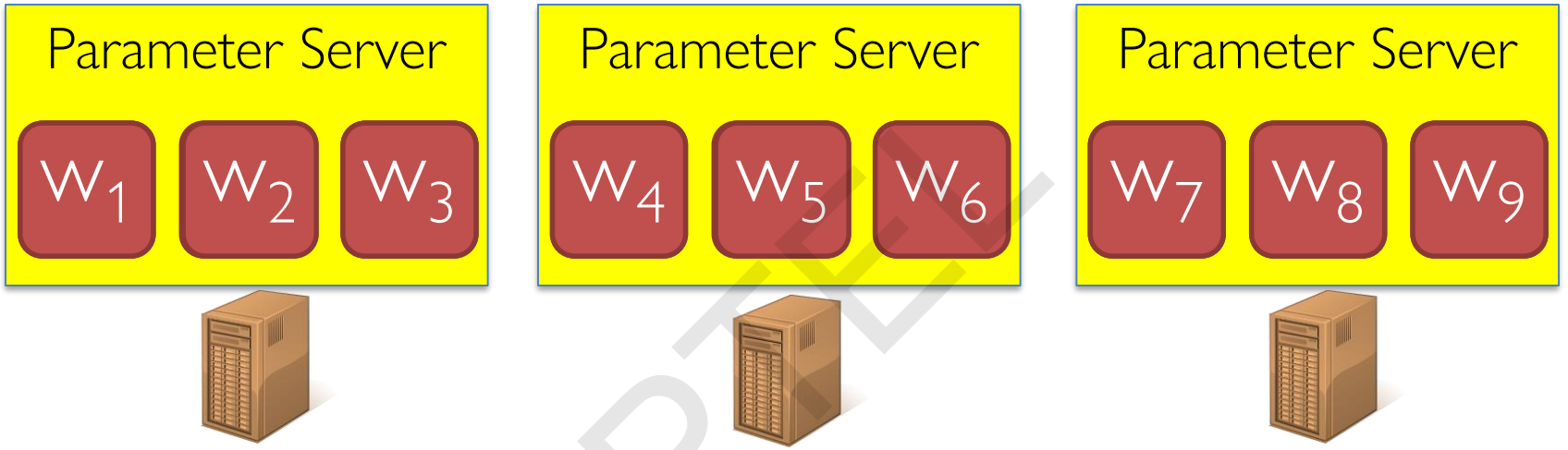
[Smola et al 2010, Ho et al 2013, Li et al 2014]

Data parallel learning with PS



Split Data Across Machines

Data parallel learning with PS



1. Different parts of the **model** on different servers.
2. Workers retrieve the part needed **as needed**

Split Data Across Machines

Abstraction used for

- Key-Value API for workers:

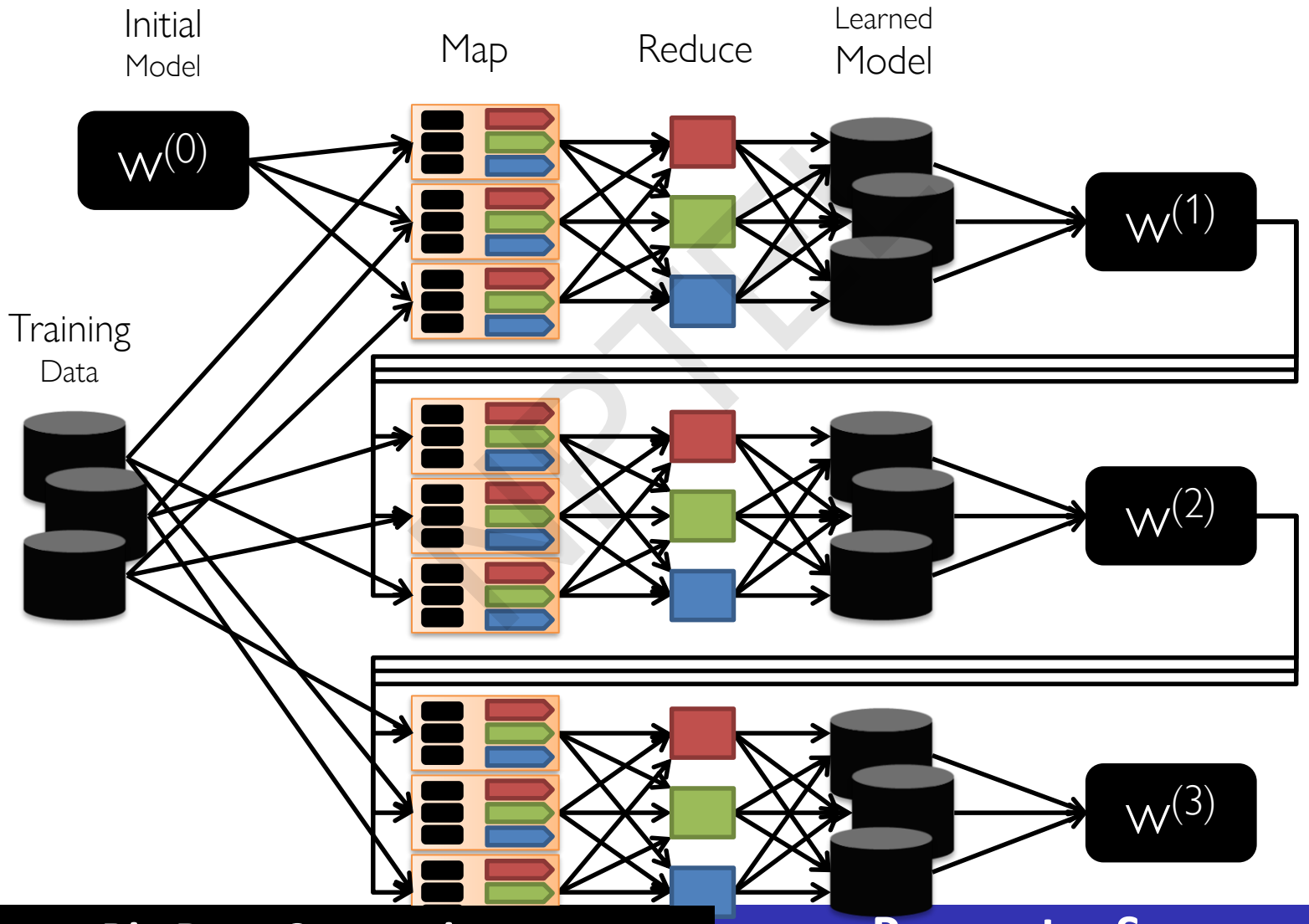
1. get(key) → value

$$\delta_i \leftarrow f(x_i, \text{Model})$$

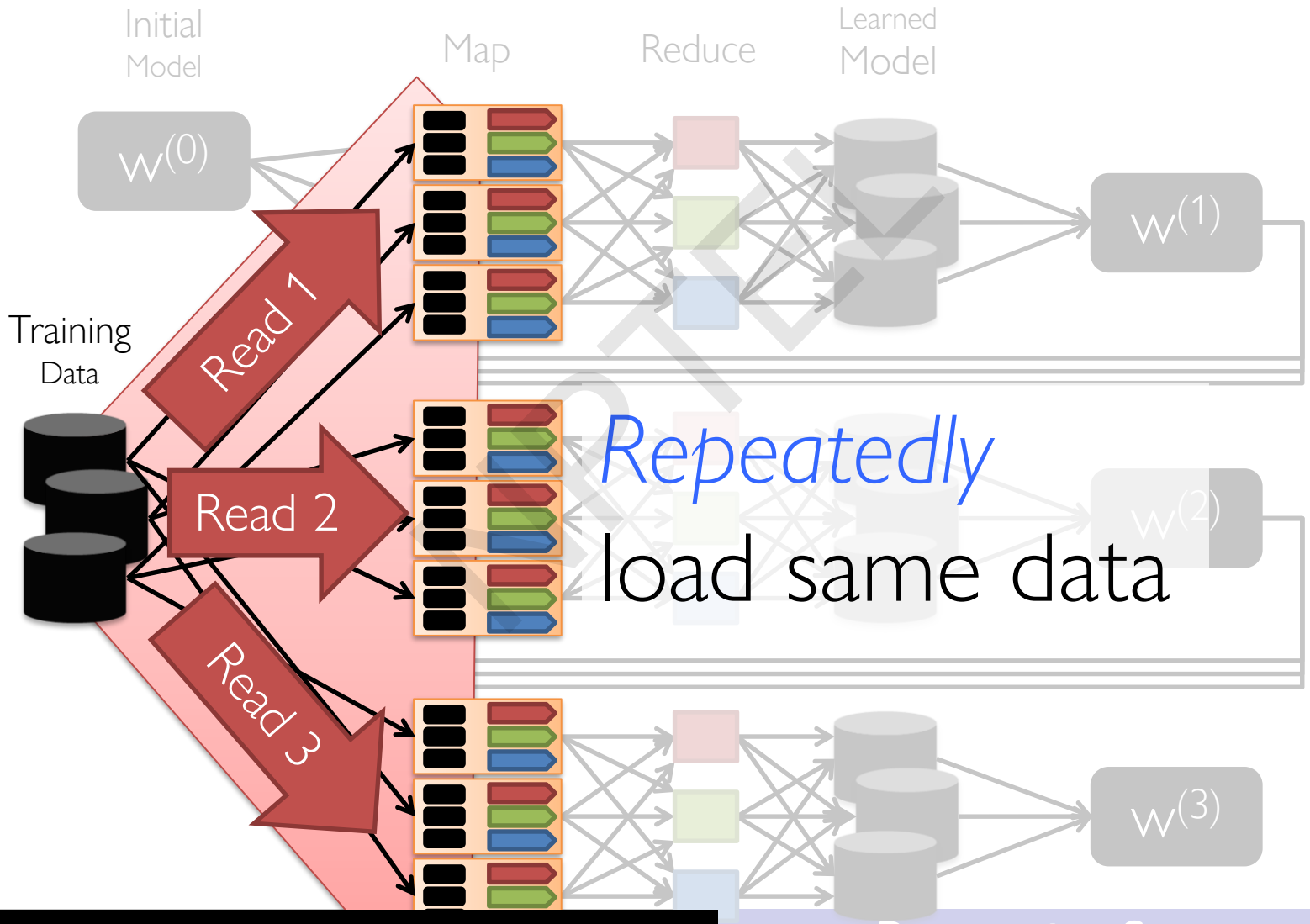
2. add(key, delta)

$$\text{Model} \leftarrow \text{Model} \oplus \delta_i$$

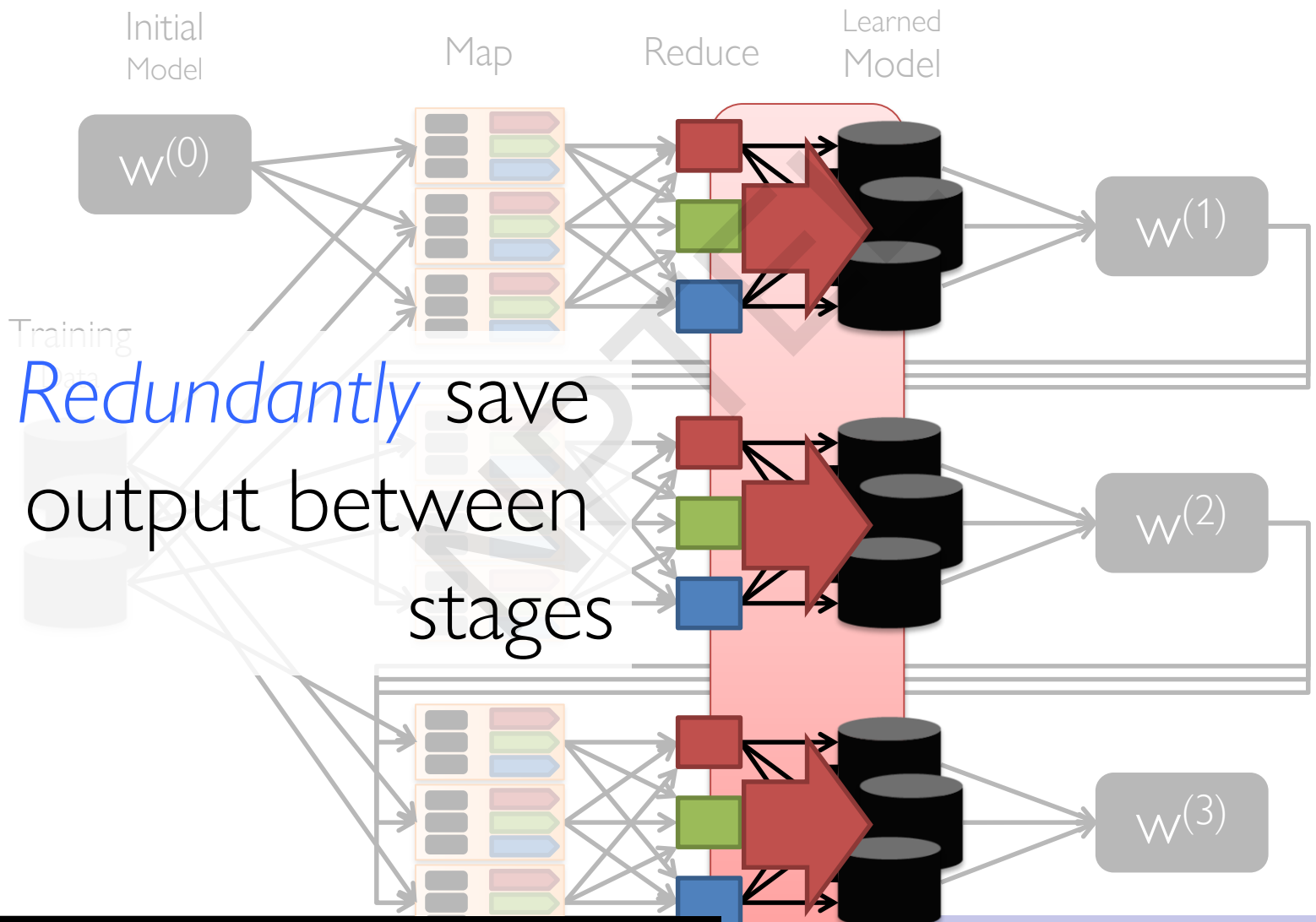
Iteration in Map-Reduce (IPM)



Cost of Iteration in Map-Reduce



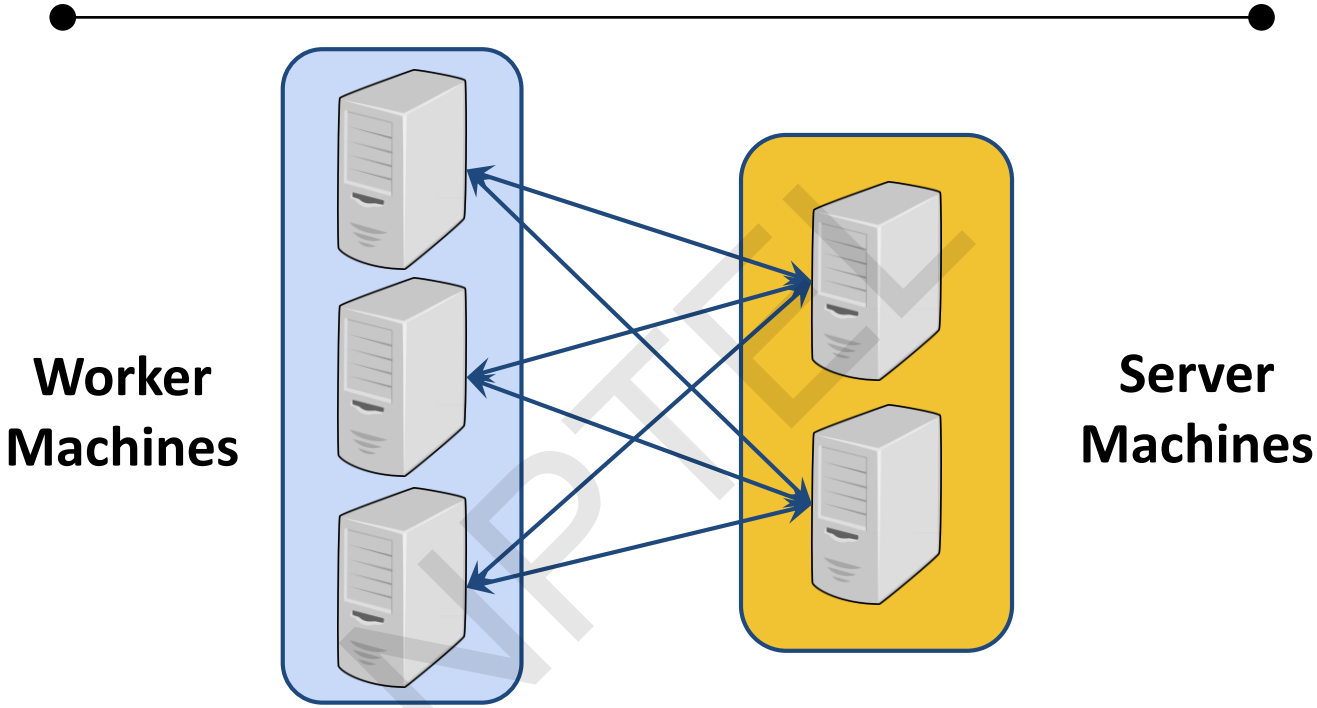
Cost of Iteration in Map-Reduce



Parameter Servers

Stale Synchronous Parallel Model

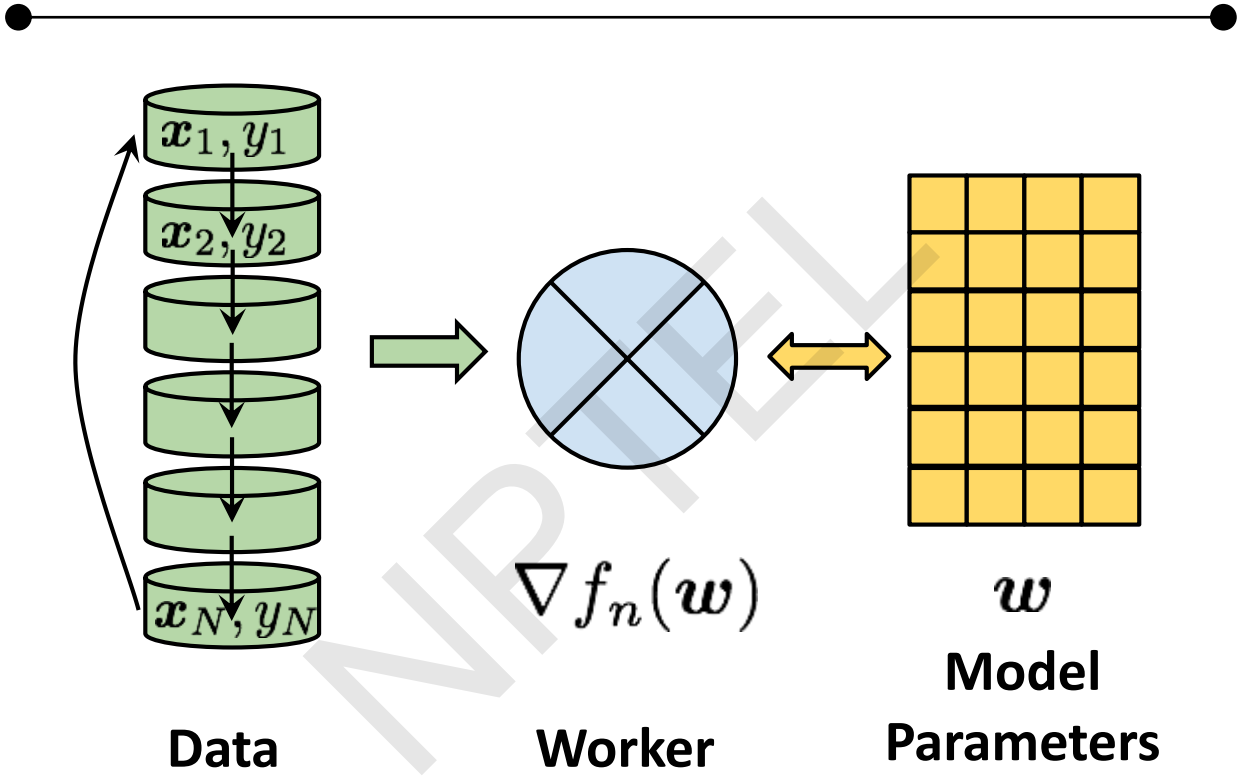
Parameter Server (PS)



- Model parameters are stored on PS machines and accessed via key-value interface (distributed shared memory)

[Smola et al 2010, Ho et al 2013, Li et al 2014]

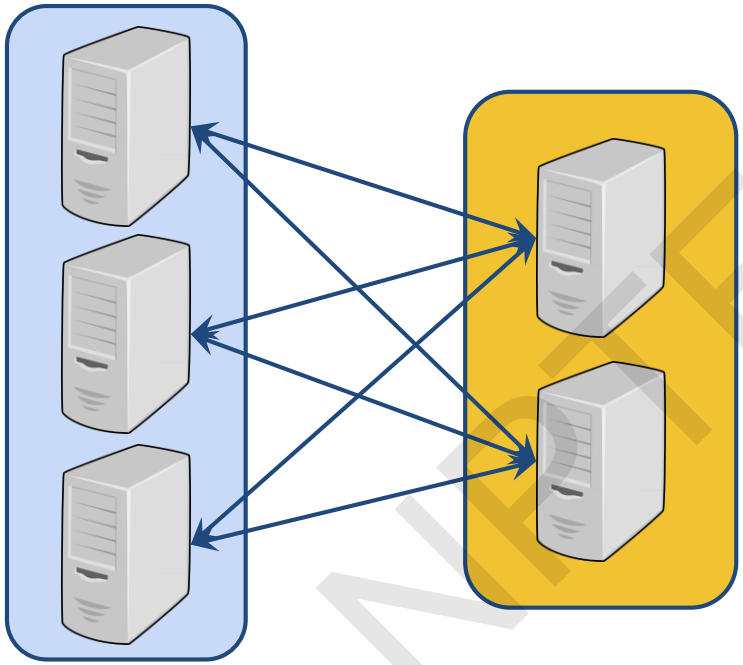
Iterative ML Algorithms



- Topic Model, matrix factorization, SVM, Deep Neural Network…

	Map-Reduce	vs.	Parameter Server
Data Model	Independent Records		Independent Data
Programming Abstraction	Map & Reduce		Key-Value Store (Distributed Shared Memory)
Execution Semantics	Bulk Synchronous Parallel (BSP)		?

The Problem: Networks Are Slow!



Worker Machines

Server Machines

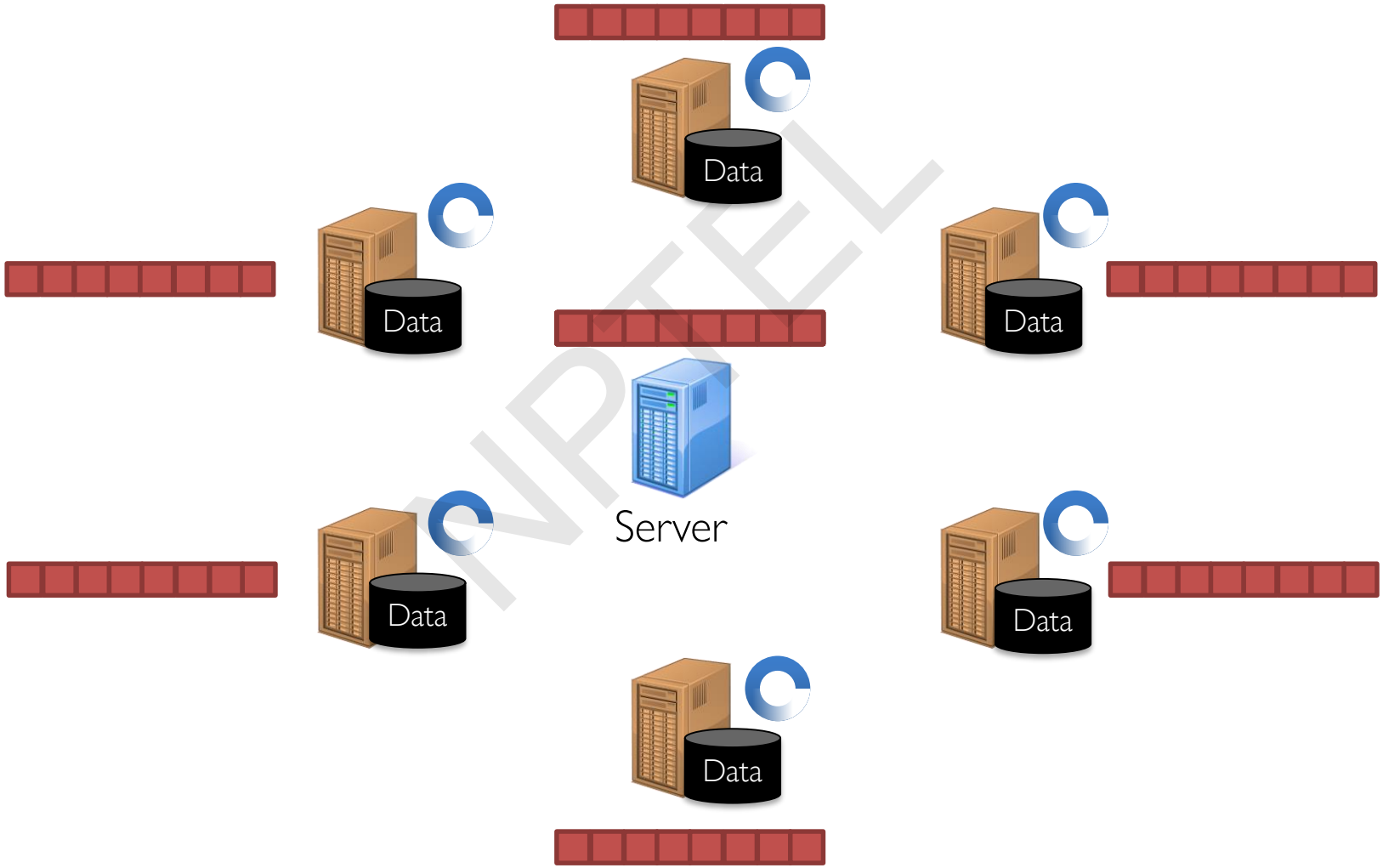
get(key)

add(key, delta)

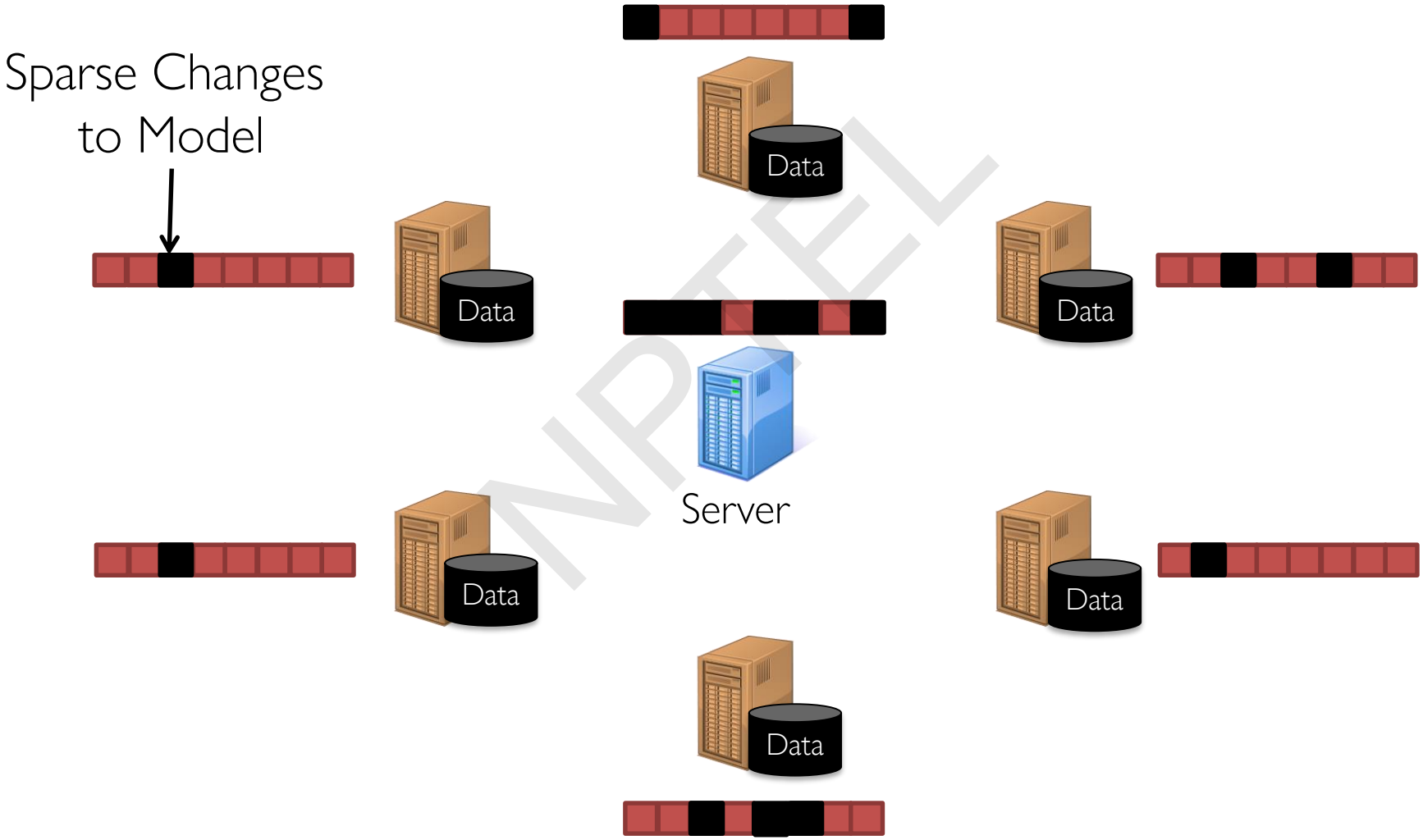
- Network is slow compared to local memory access
- We want to explore options for handling this….

[Smola et al 2010, Ho et al 2013, Li et al 2014]

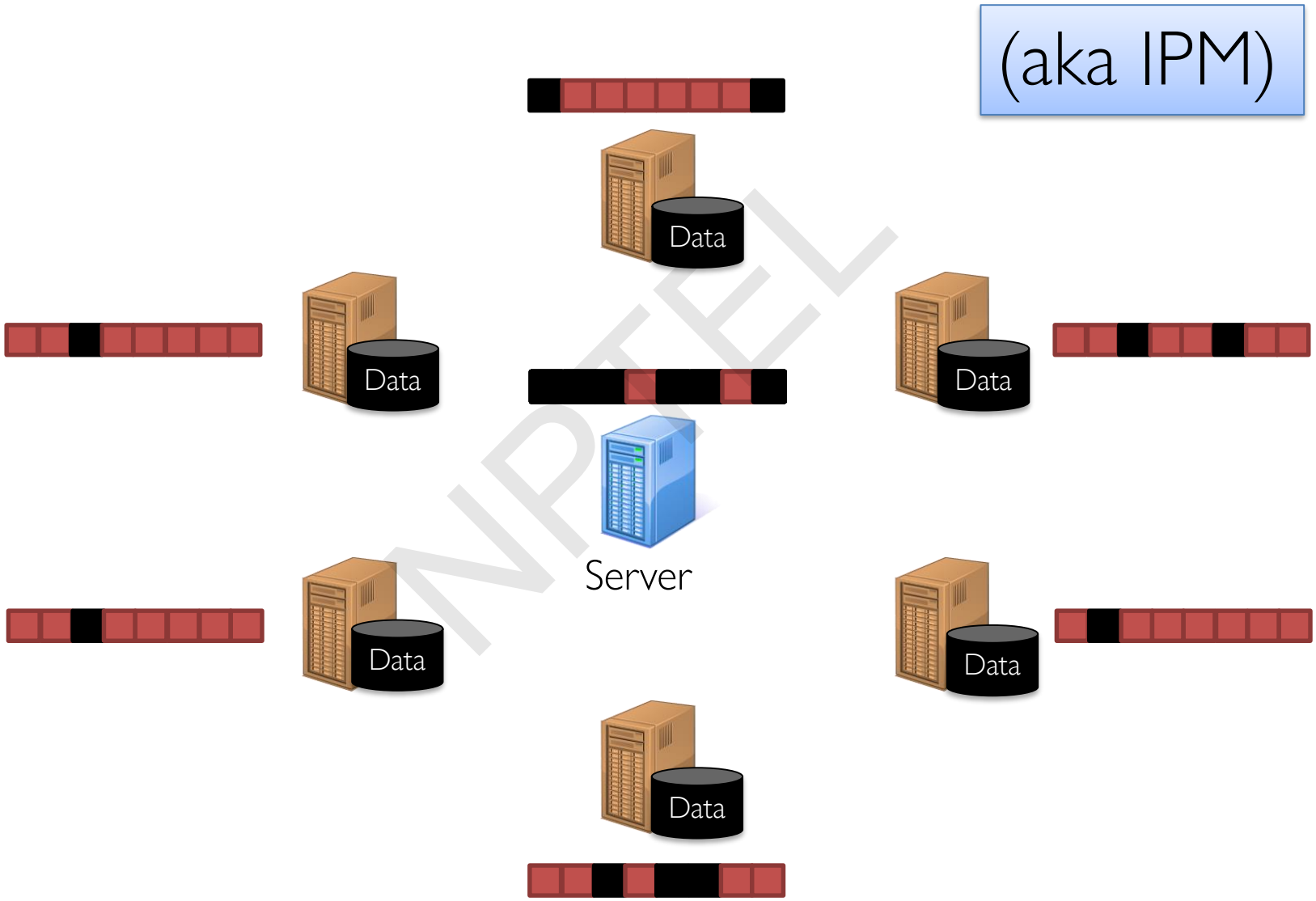
Solution 1: Cache Synchronization



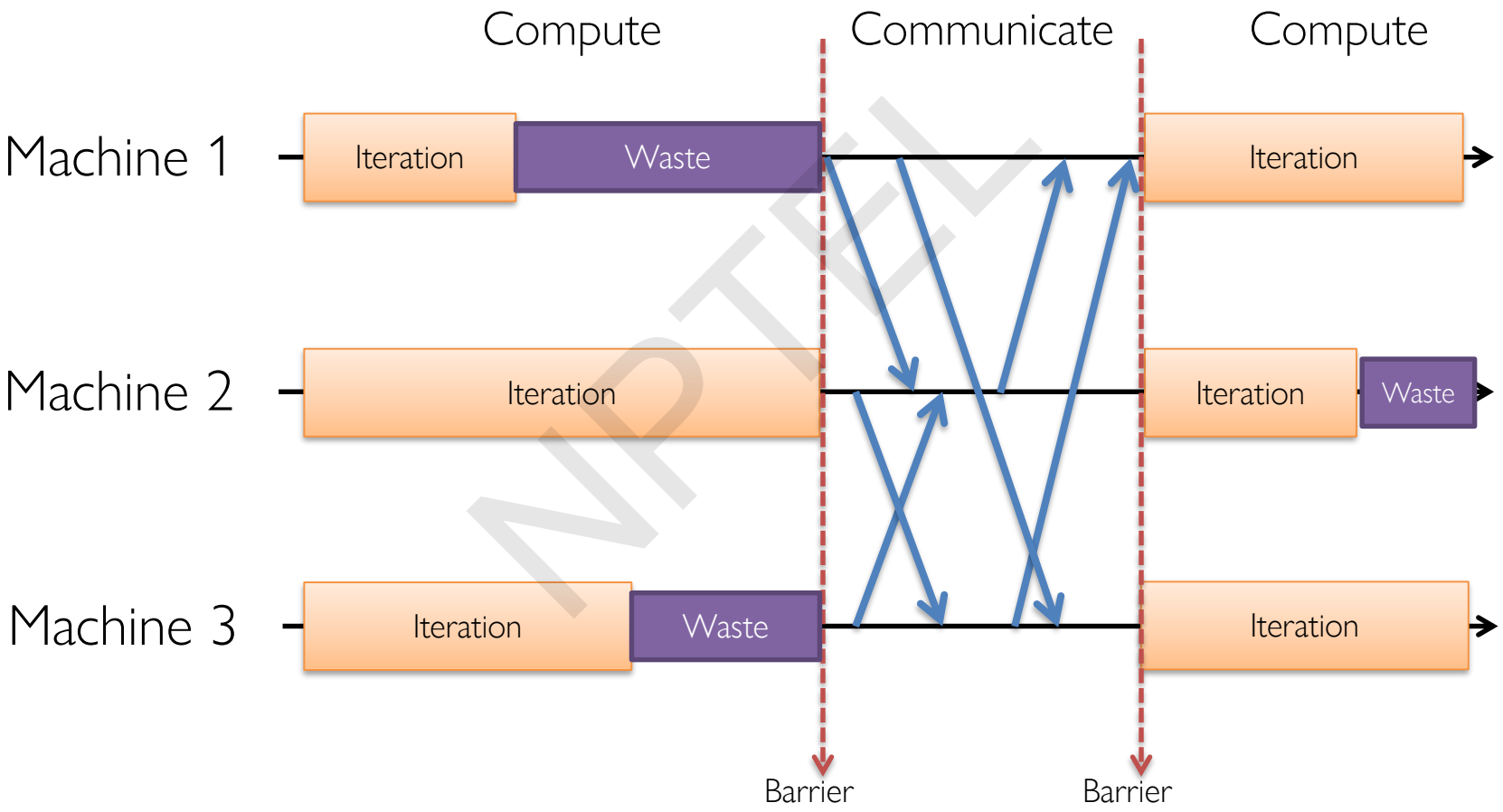
Parameter Cache Synchronization



Parameter Cache Synchronization

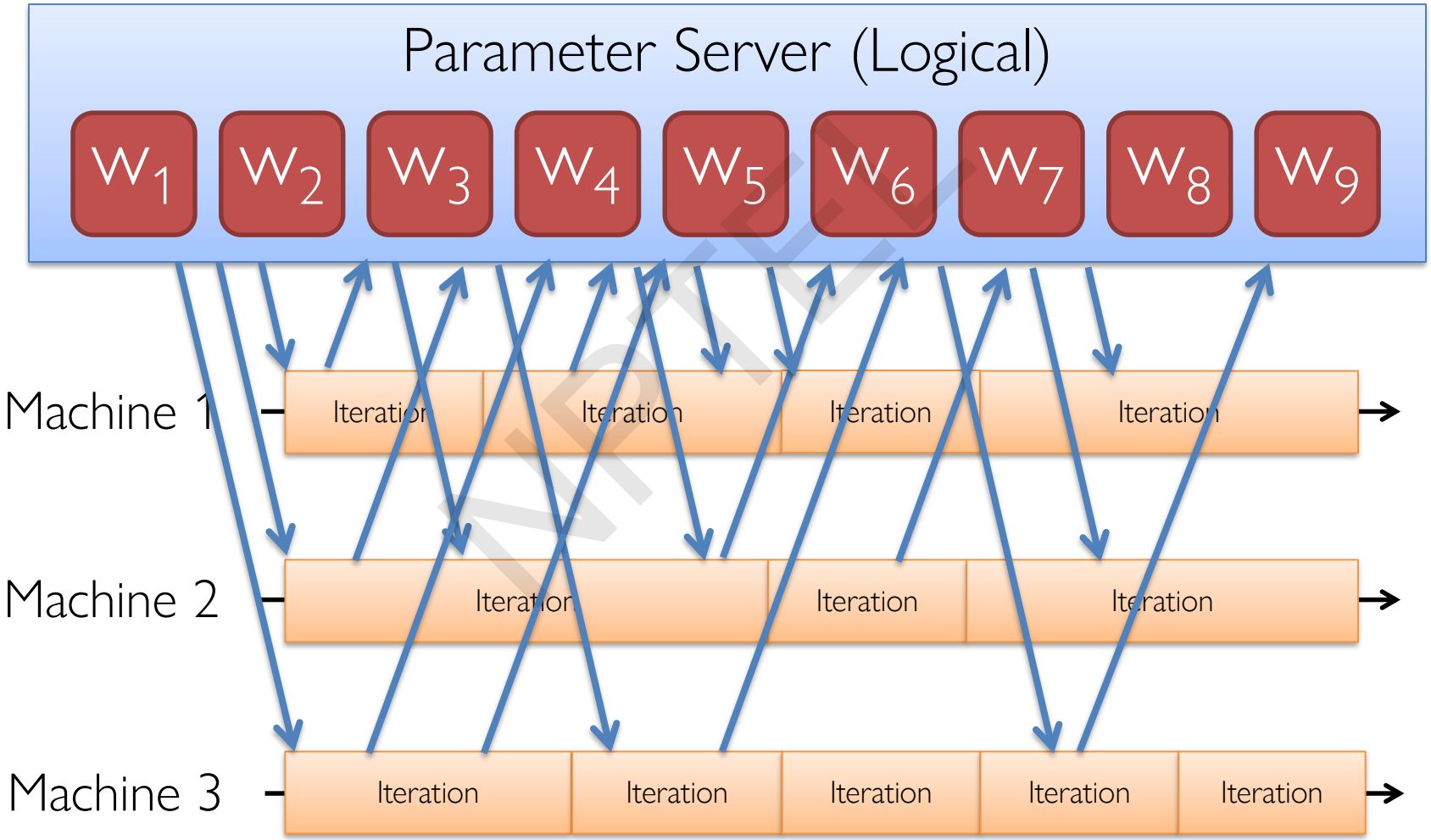


Solution 2: Asynchronous



Enable more frequent coordination on parameter values

Asynchronous Execution



[Smola et al 2010]

Asynchronous Execution

- Problem:
- Async lacks theoretical guarantee as distributed environment can have arbitrary delays from network & stragglers

- But….

RECAP

f is loss function, x is parameters

1. Take a gradient step: $x' = x_t - \eta_t g_t$
2. If you've restricted the parameters to a subspace X (e.g., must be positive, …) find the closest thing in X to x': $x_{t+1} = \text{argmin}_X \; dist(x - x')$
3. But…. you might be using a "stale" g (from τ steps ago)

Algorithm 1 Delayed Stochastic Gradient Descer

Input: Feasible space $X \subseteq \mathbb{R}^n$, annealing sched … elay $\tau \in \mathbb{N}$

Initialization: set $x_1 \ldots, x_\tau = 0$ and compute c… ng $g_t = \nabla f_t(x_t)$.

for $t = \tau + 1$ **to** $T + \tau$ **do**

  Obtain f_t and incur loss $f_t(x_t)$

  Compute $g_t := \nabla f_t(x_t)$

  Update $x_{t+1} = \text{argmin}_{x \in X} \|x - (x_t - \eta_t g_{t-\tau})\|$ (Gradient Step and Projection)

end for

	Map-Reduce	vs.	Parameter Server
Data Model	Independent Records		Independent Data
Programming Abstraction	Map & Reduce		Key-Value Store (Distributed Shared Memory)
Execution Semantics	Bulk Synchronous Parallel (BSP)		Bounded Asynchronous

Parameter Server

Stale synchronous parallel (SSP):

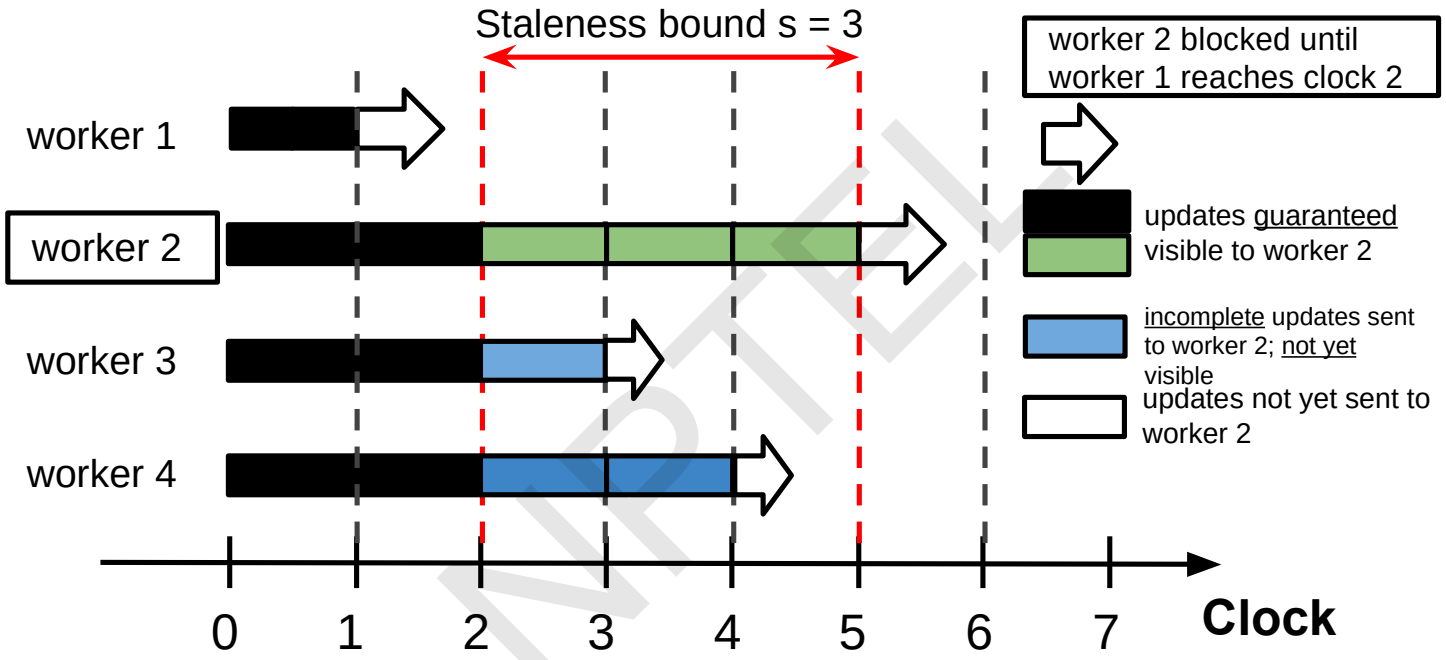
- Global clock time t
- Parameters workers "get" *can* be out of date
- but can't be older than t-τ

- τ controls "staleness"

- aka stale synchronous parallel (SSP)

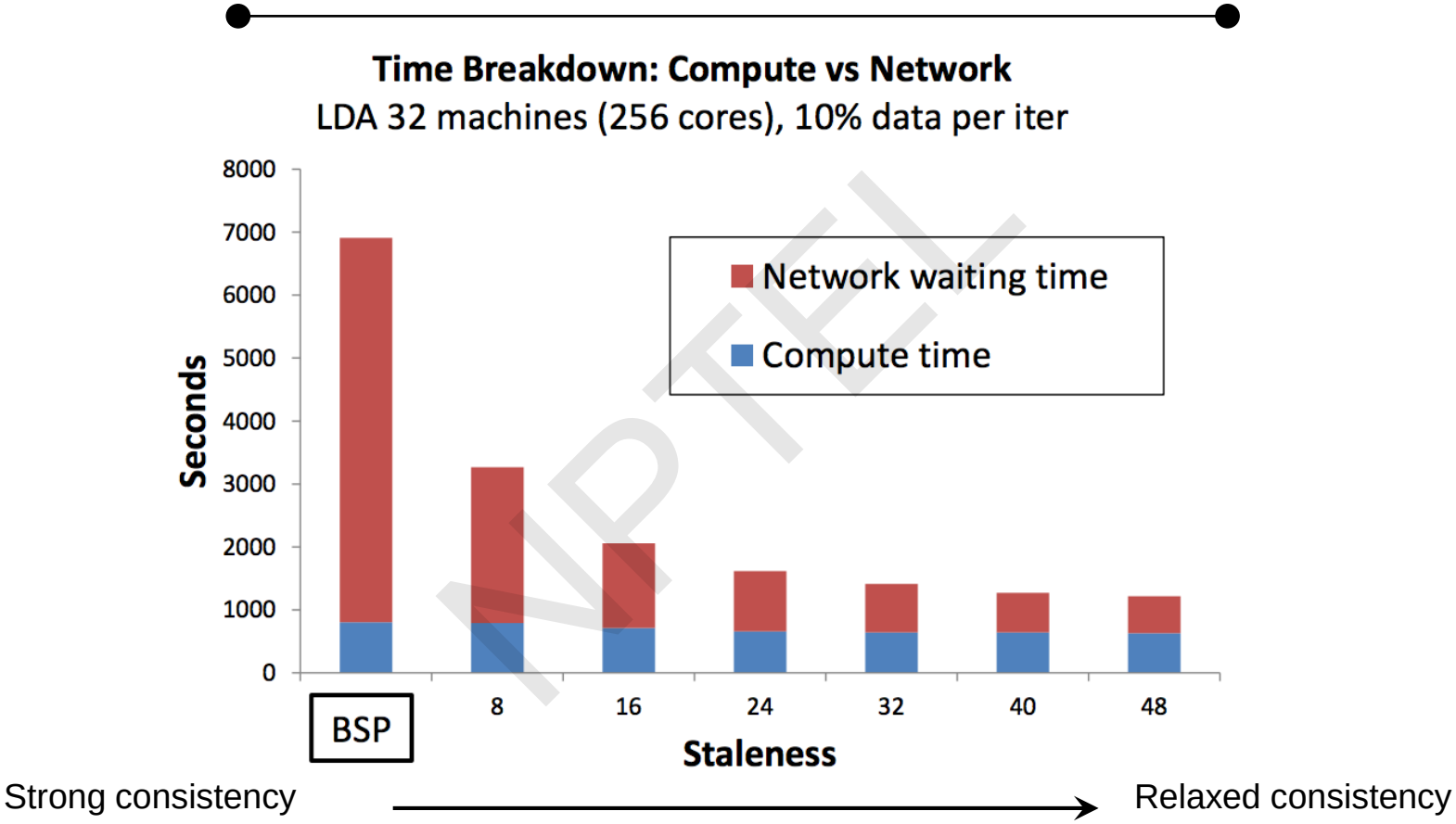
Bounded Asynchronous

Stale Synchronous Parallel (SSP)



- Interpolate between BSP and Async and subsumes both
- Allow workers to usually run at own pace
- Fastest/slowest threads not allowed to drift >s clocks apart
- Efficiently implemented: Cache parameters [Ho et al 2013]

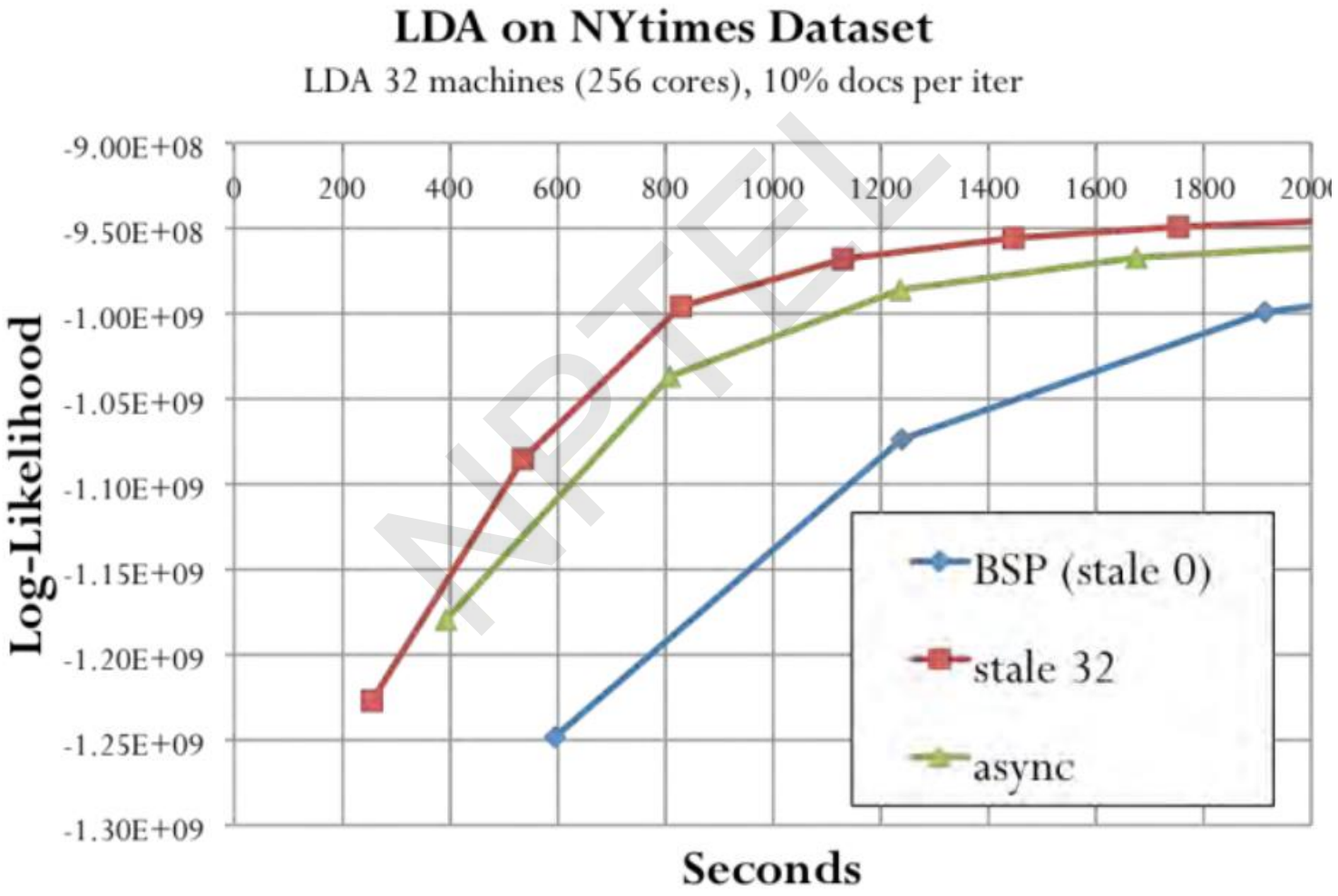
Consistency Matters



➢ Suitable delay (SSP) gives big speed-up

[Ho et al 2013]

Stale Synchronous Parallel (SSP)



[Ho et al 2013]

Conclusion

- In this lecture, we have discussed the parameters servers.

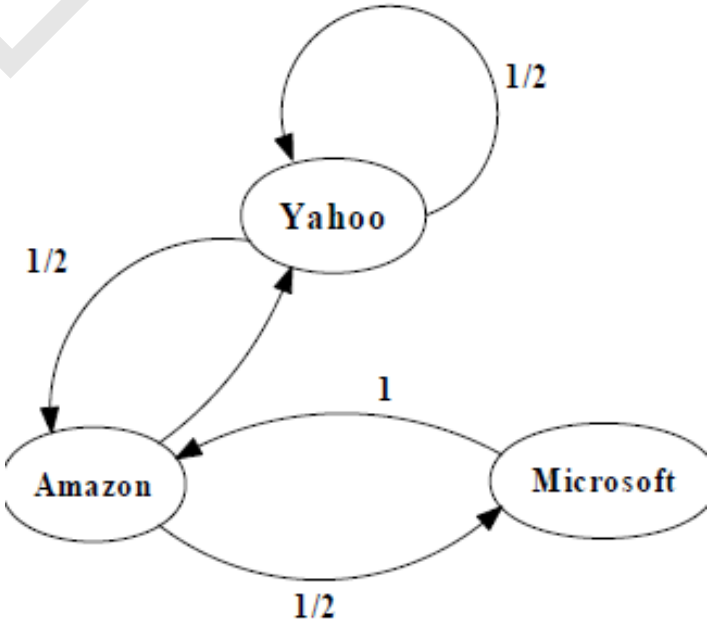
PageRank Algorithm in Big Data

Dr. Rajiv Misra
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

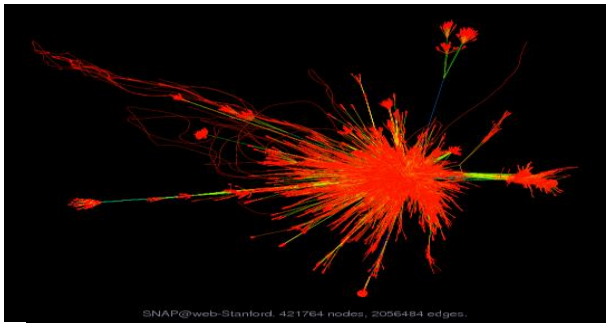
- In this lecture, we will discuss PageRank Algorithm in Big Data using different framework with different ways and scale.



Big Graphs

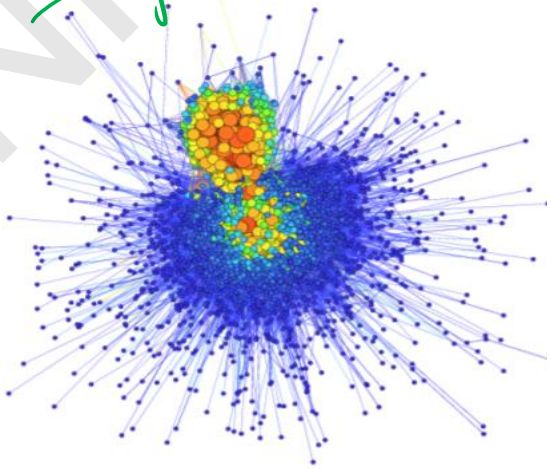
- Social scale graph
  - 1 billion vertices, 100 billion edges
- Web scale graph
  - 50 billion vertices, 1 trillion edges
- Brain scale graph
  - 100 billion vertices, 100 trillion edges

Social graph from Facebook
https://medium.com/@johnrobb/facebook-the-complete-social-graph-b58157ee6594

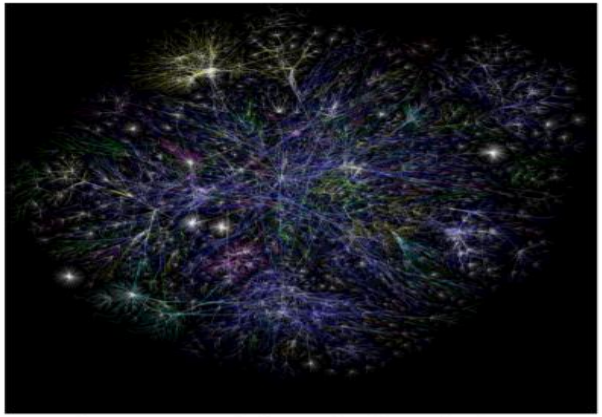


Web graph from the SNAP database
https://snap.stanford.edu/data/

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



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



Internet graph from the Opte Project
(http://www.opte.org/maps)

What is PageRank?

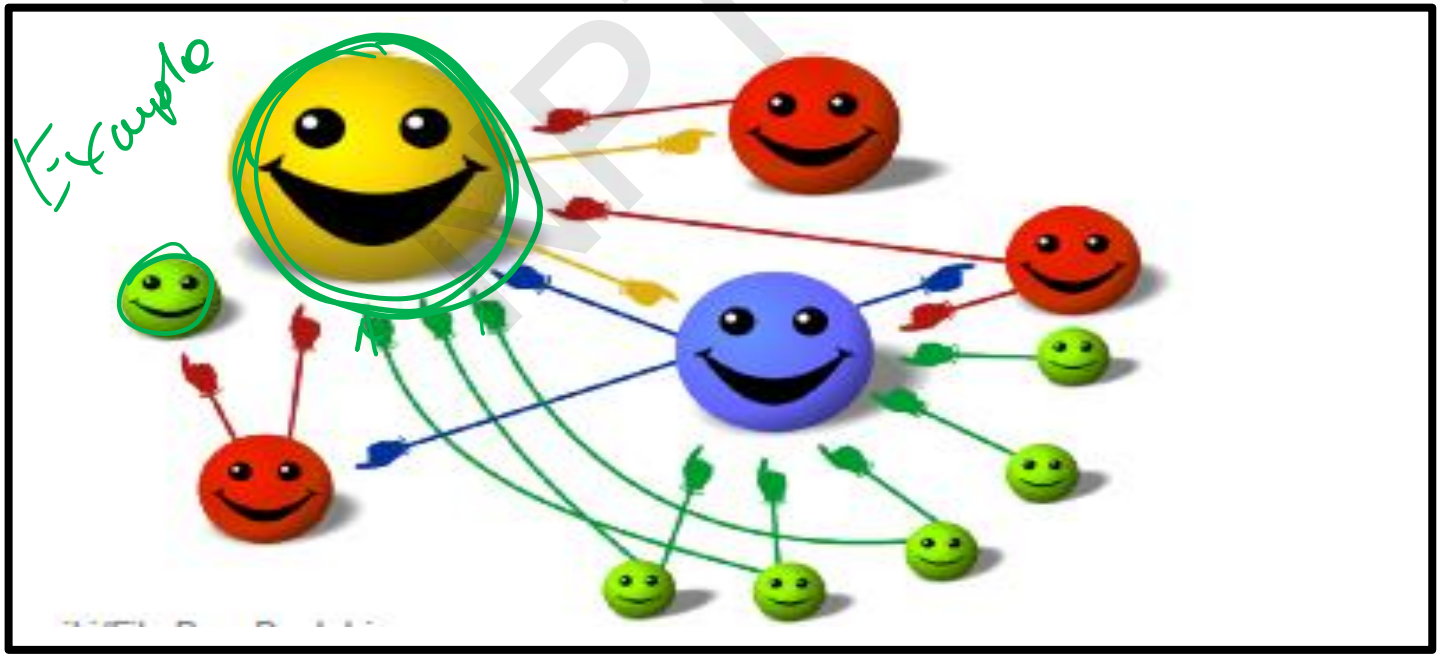
- Why is Page Importance Rating important?
  - New challenges for information retrieval on the World Wide Web.
  - Huge number of web pages: 150 million by1998
    1000 billion by 2008
  - Diversity of web pages: different topics, different quality, etc.

- What is PageRank?
  - A method for rating the importance of web pages objectively and mechanically using the link structure of the web.

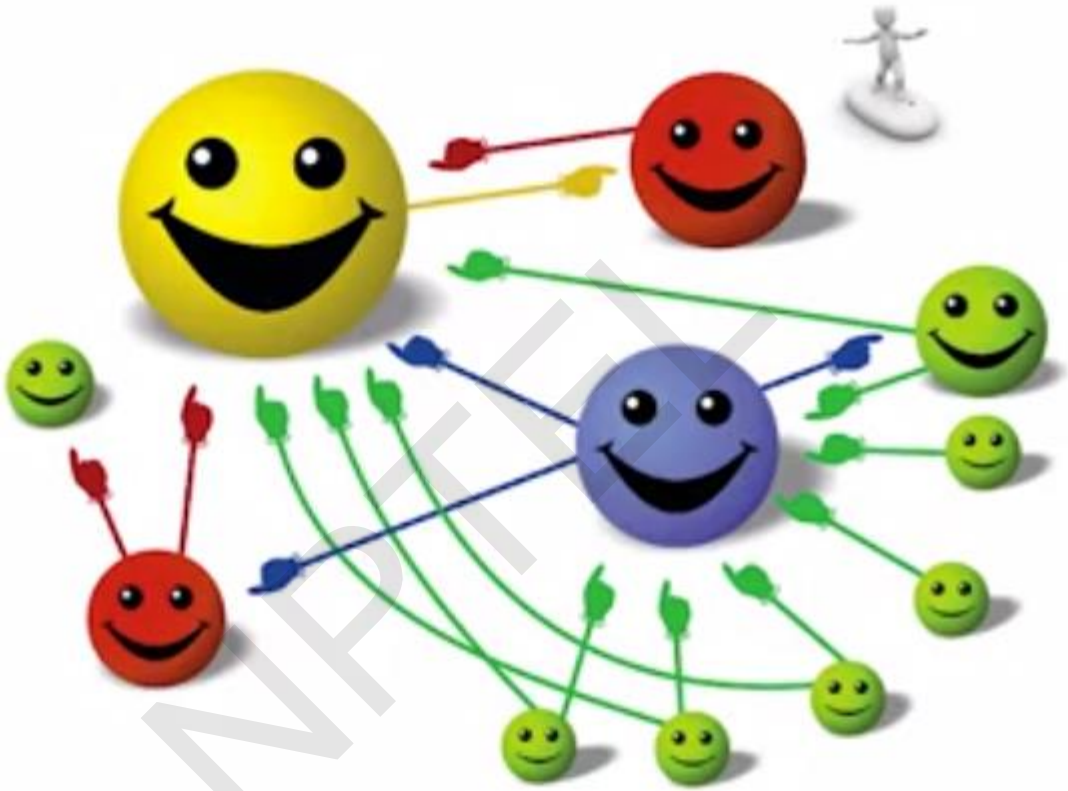
  - History:
    - PageRank was developed by Larry Page (hence the name *Page*-Rank) and Sergey Brin. of google
    - It is first as part of a research project about a new kind of search engine. That project started in 1995 and led to a functional prototype in 1998.

PageRank of E denotes by PR (E)

- Give pages ranks (scores) based on links to them
- Links from many pages ⇒ high rank
- Links from a high-rank page ⇒ high rank

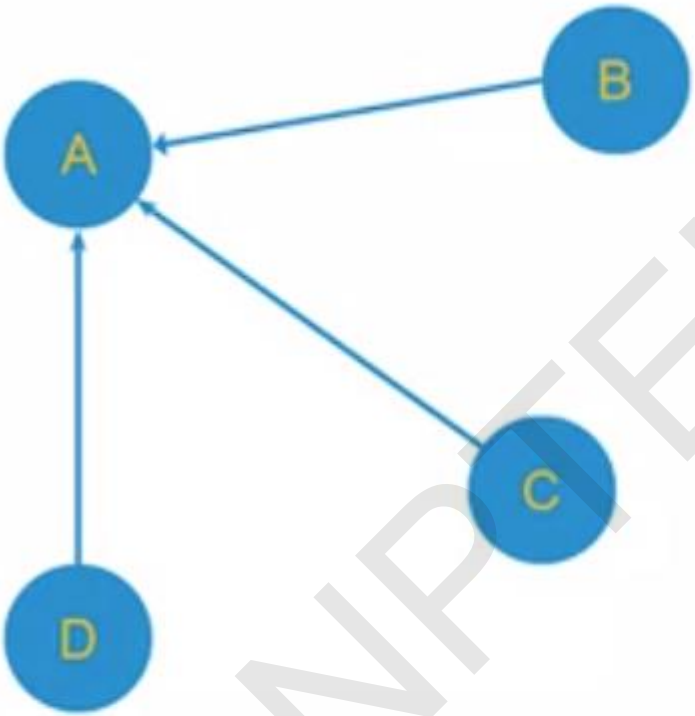


PageRank of E denotes by PR (E)

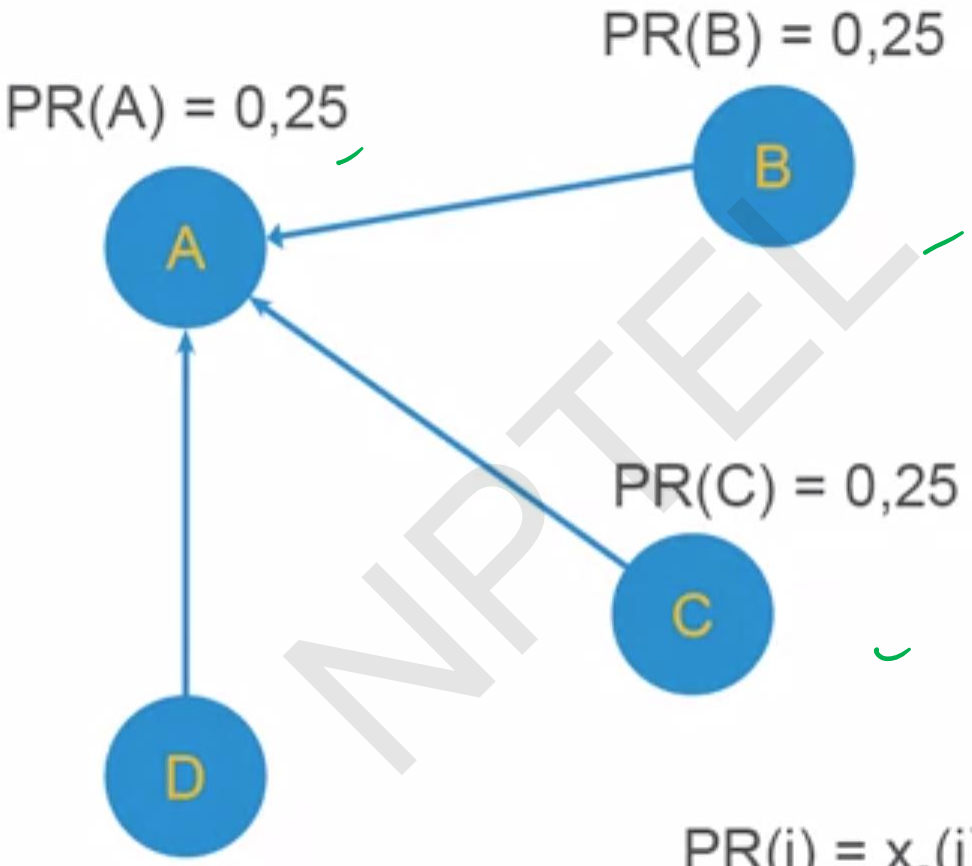


Stationary distribution $x^* = (x^*)^T P$

Example



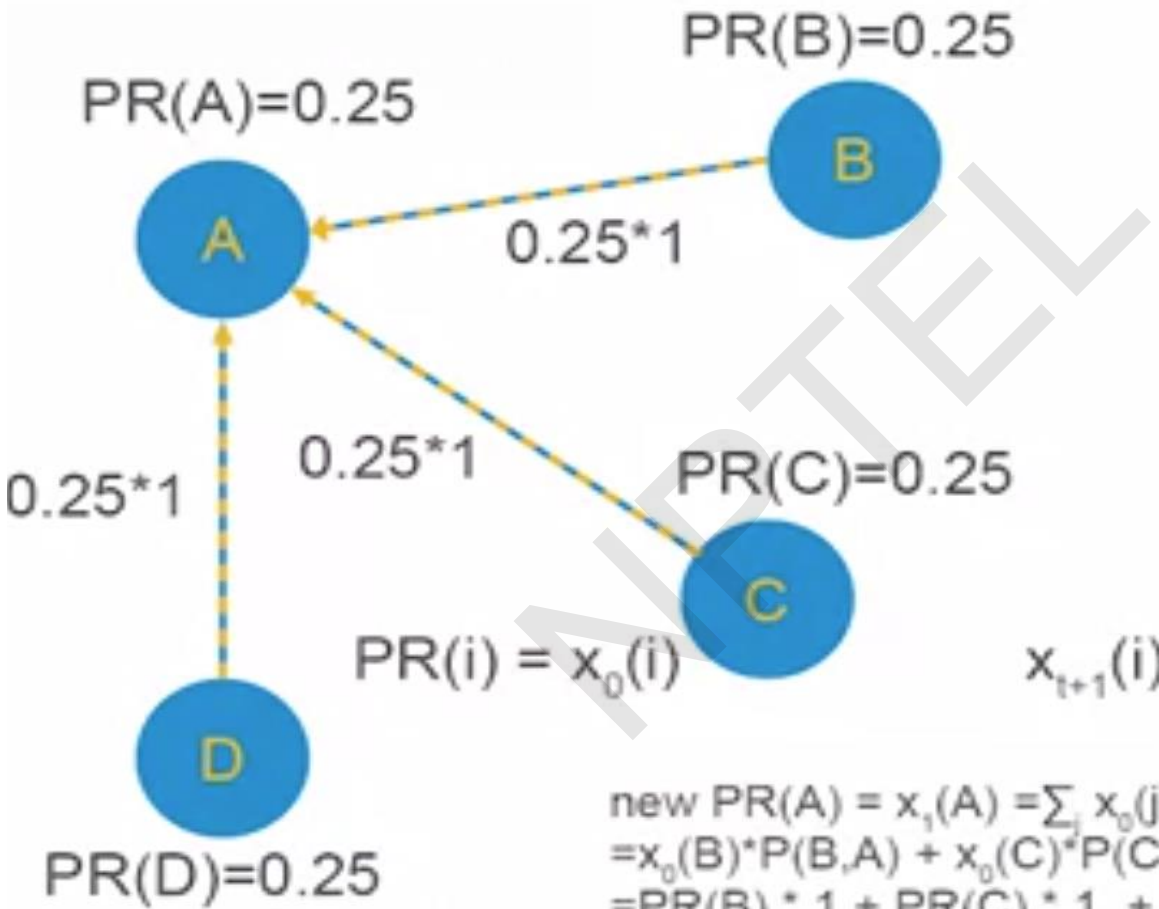
Example



PR(i) = $x_0(i)$ = probability that the surfer is at node i at time 0

initial page rank = $\frac{1}{4}$ for all nodes.

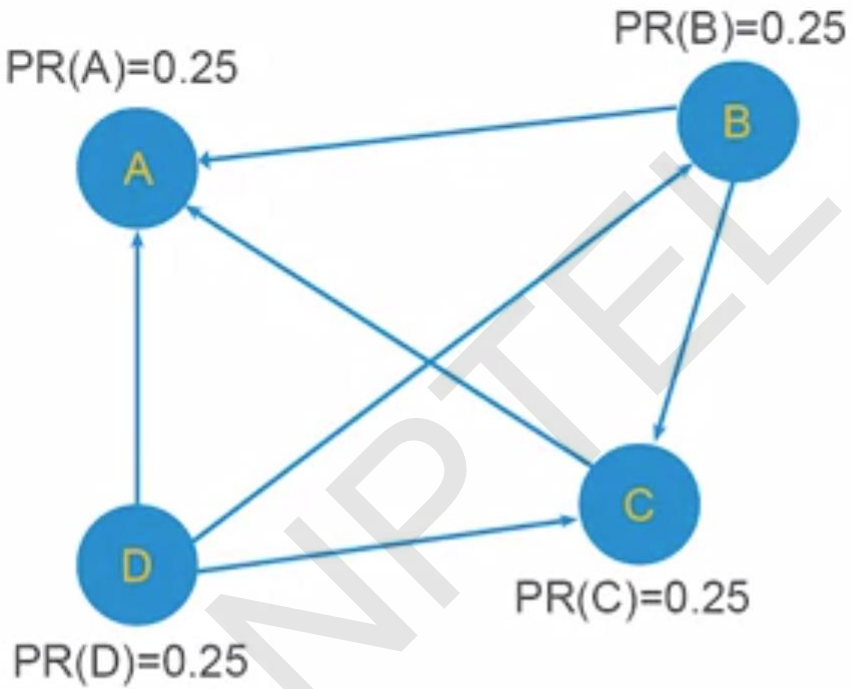
Example



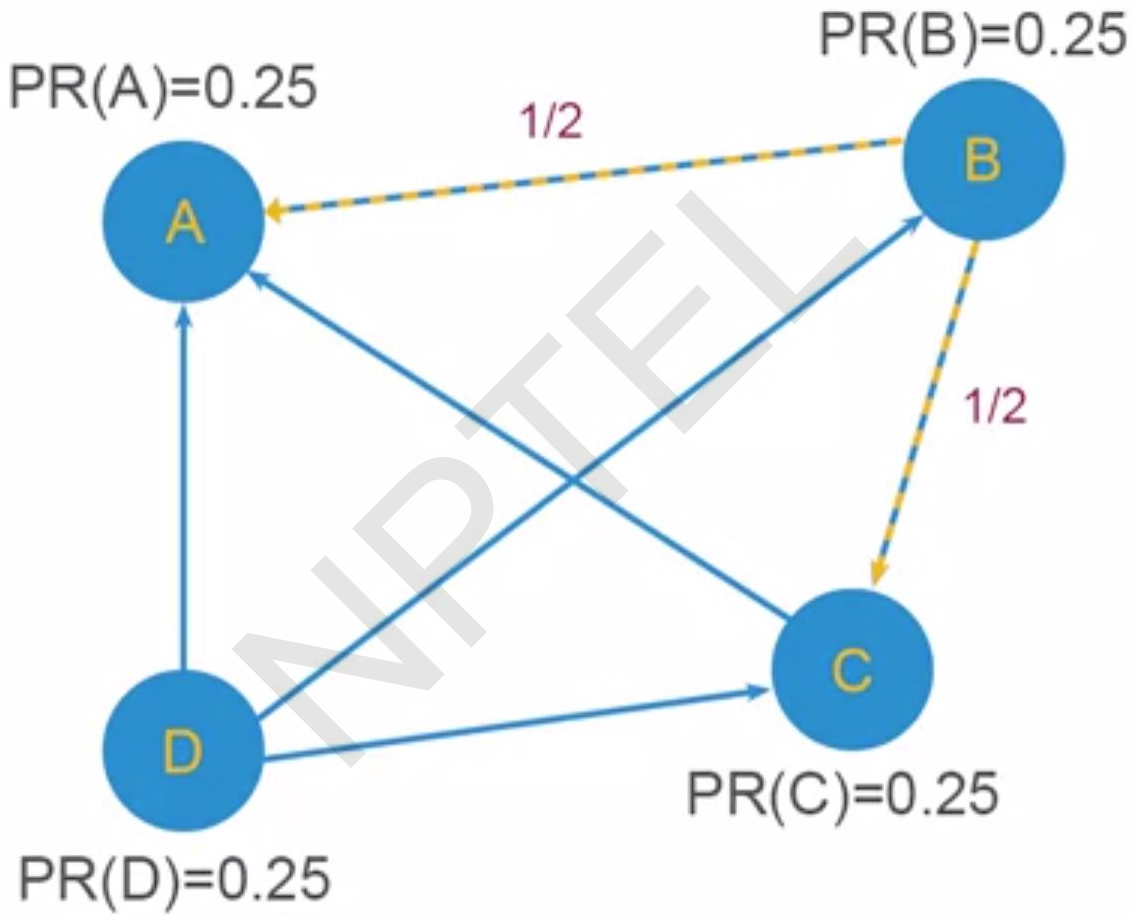
$x_{t+1}(i) = \sum_j x_t(j)*P(j,i)$

new PR(A) = $x_1(A) = \sum_j x_0(j)*P(j,A)$=
$= x_0(B)*P(B,A) + x_0(C)*P(C,A) + x_0(D)*P(D,A)$ =
$= PR(B) * 1 + PR(C) * 1 + PR(D) * 1 = 0.75$

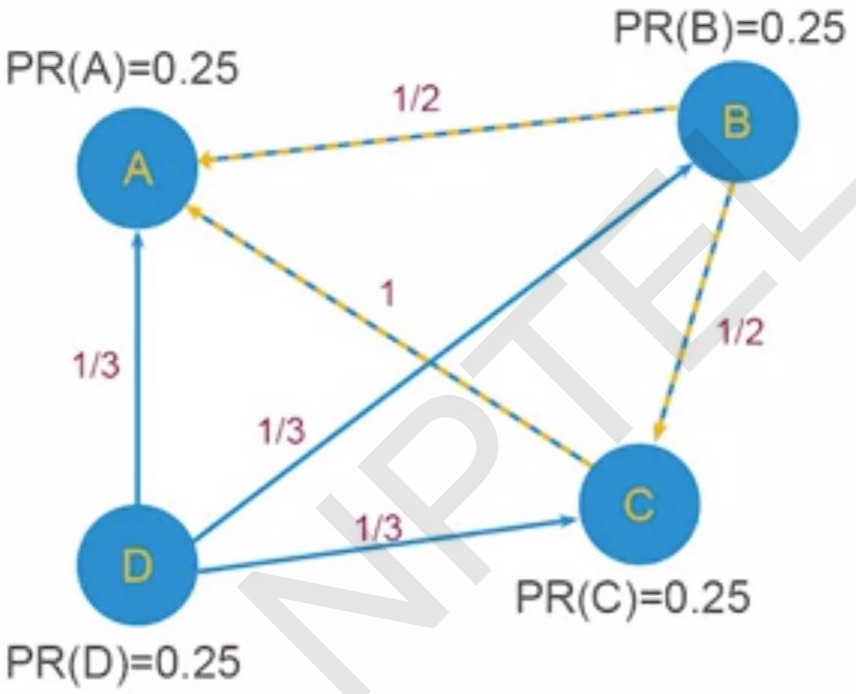
Example



Example



Example



new PR(A) = $x_1(A)$=

=$\sum_j x_0(j)*P(j,A)$ =

= $x_0(B)*P(B,A)$ +

+$x_0(C)*P(C,A)$ +

+$x_0(D)*P(D,A)$ =

= PR(B) * 1/2+

+ PR(C) * 1 +

+PR(D) * 1/3 =

=0.458

Page Rank

L(v) - the number of vertex v outbound links

Γ(v) - the set containing all pages linking to page v

$$PR(u) = \sum_{v \in \Gamma(u)} \frac{PR(v)}{L(v)}$$

Stationary Distribution

$x^* = (x^*)^T P$

Theorem

asserting that if a stochastic graph satisfies two conditions:

1. there is a path from every node to every node
2. the greatest common divider of all the cycle lengths id 1

then there is a unique stationary probability distribution

Page Rank

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots$$

damping factor

$$PR(A) = \frac{1-d}{N} + d\left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right)$$

Page Rank

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in \Gamma(p_i)} \frac{PR(p_j)}{L(p_j)}$$

where $p_1, p_2, \ldots, p_N$ are the pages under consideration

$\Gamma(p_i)$ is the set of pages that link p_i

$L(p_i)$ is the number of outbound links on page p_i

and N is the total number of pages

Page Rank

At t= 0, an initial probability distribution is assumed, usually $PR(p_i;0) = \frac{1}{N}$

At each time step $PR(p_i;t+1) = \frac{1-d}{N} + d \sum_{p_j \in \Gamma(p_i)} \frac{PR(p_j;t)}{L(p_j)}$

The computation ends:

1. After fixed number of iterations
2. When convergence is assumed

$$\left(\sum_{i=0}^{N} |PR(p_i, t+1) - PR(p_i, t)| \right) < \epsilon$$

$$PR(p_i; t+1) \xrightarrow[t \to \infty]{} x^*(p_i)$$

Summary

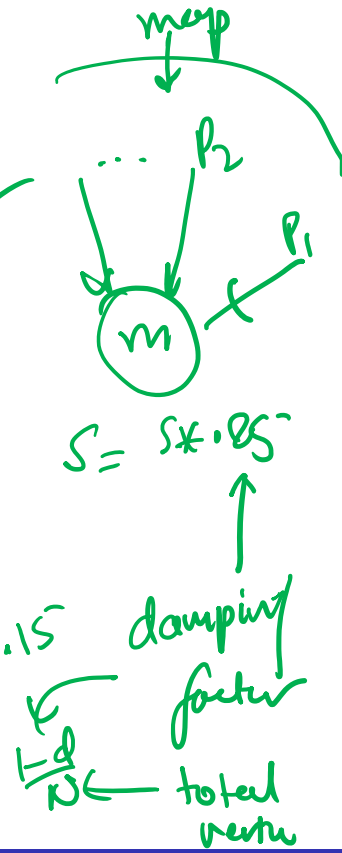
- PageRank (PR)- a first algorithm by Google Search to rank websites in their search engine results.

- We have shown how to calculate iteratively PageRank for every vertex in the given graph.

MapReduce for PageRank

```
class Mapper
    method Map(id n, vertex N)
    p ← N.PAGERANK/|N.ADJACENCYLIST|
    EMIT(id n, vertex N)
    for all nodeid m in N.ADJACENCYLIST do
        EMIT(id m, value p)
```

```
class Reducer
    method REDUCE(id m, [p1, p2, …])
        M ← null, s ← 0
        for all p in [p1, p2, …] do
            if ISVERTEX(p) then
                M ← p
            else
                s ← s + p
        M.PAGERANK ← s * 0.85 + 0.15 / TOTALVERTICES
        EMIT(id m, vertex M)
```



Problems

- The entire state of the graph is shuffled on every iteration

node object

- We only need to shuffle the new rank contributions, not the graph structure

Pregel, GraphLab, GIRAPH, GraphX

- Further, we have to control the iteration outside of MapReduce

Pregel

- Originally from Google
  - Open source implementations
  - Apache Giraph, Standard GPS, Jpregel, Hama

- Batch algorithms on large graphs

```
while any vertex is active or max iterations not reached:
    for each vertex:  <-- this loop is run in parallel
      process messages from neighbors from previous iteration
      send messages to neighbors
      set active flag appropriately
```

PageRank in Pregel

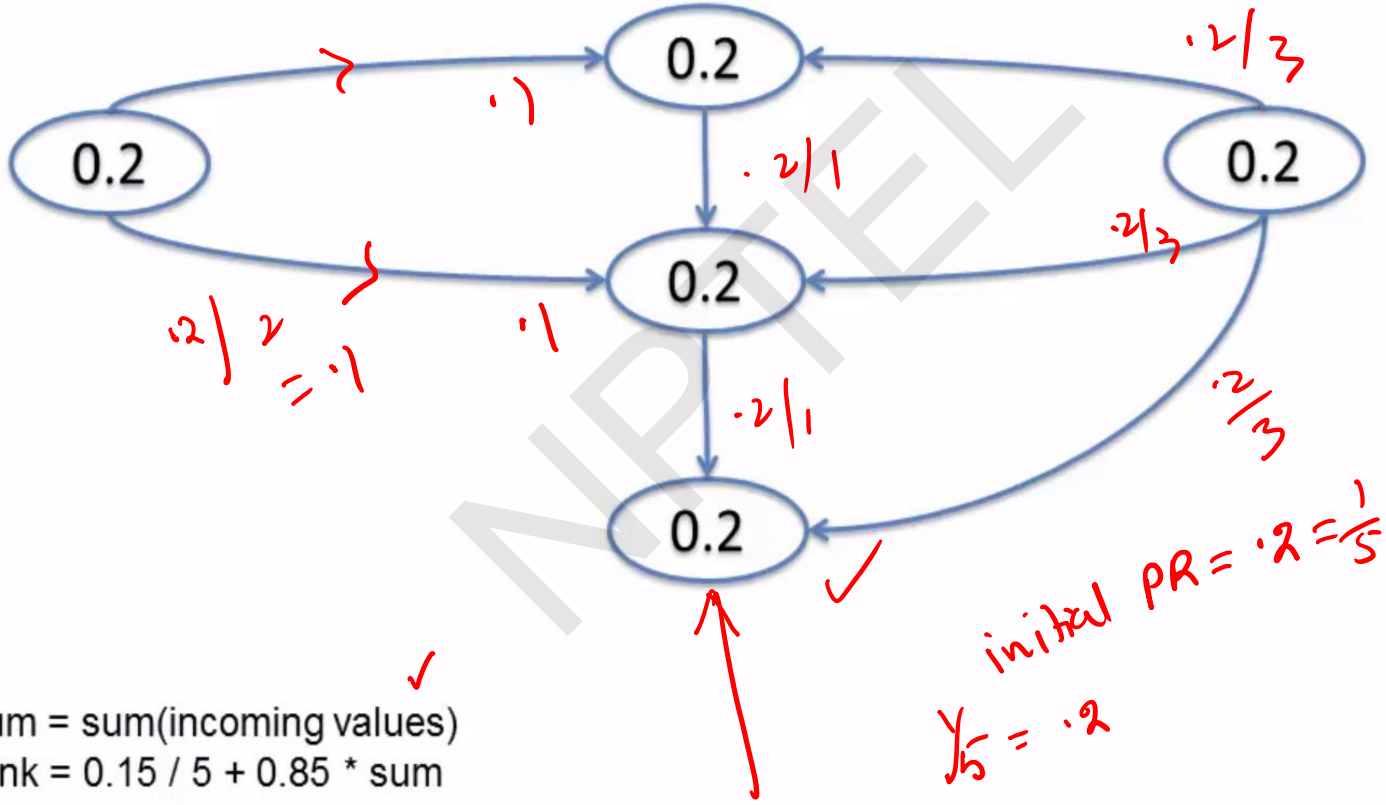
```
class PageRankVertex: public Vertex<double, void, double> {
    public:
    virtual void Compute(MessageIterator* msgs) {
        if (superstep() >= 1) {
            double sum = 0;
            for (; !msgs->Done(); msgs->Next())
                sum += msgs->Value();
            *MutableValue() = 0.15 / NumVertices() + 0.85 * sum;
        }
        if (superstep() < 30) {
            const int64 n = GetOutEdgeIterator().size();
            SendMessageToAllNeighbors(GetValue() / n);
        } else {
            VoteToHalt();
        }
    }
};
```

iteration

new page rank

Vertex program in parallel execution of all vertices in Graph.

Example



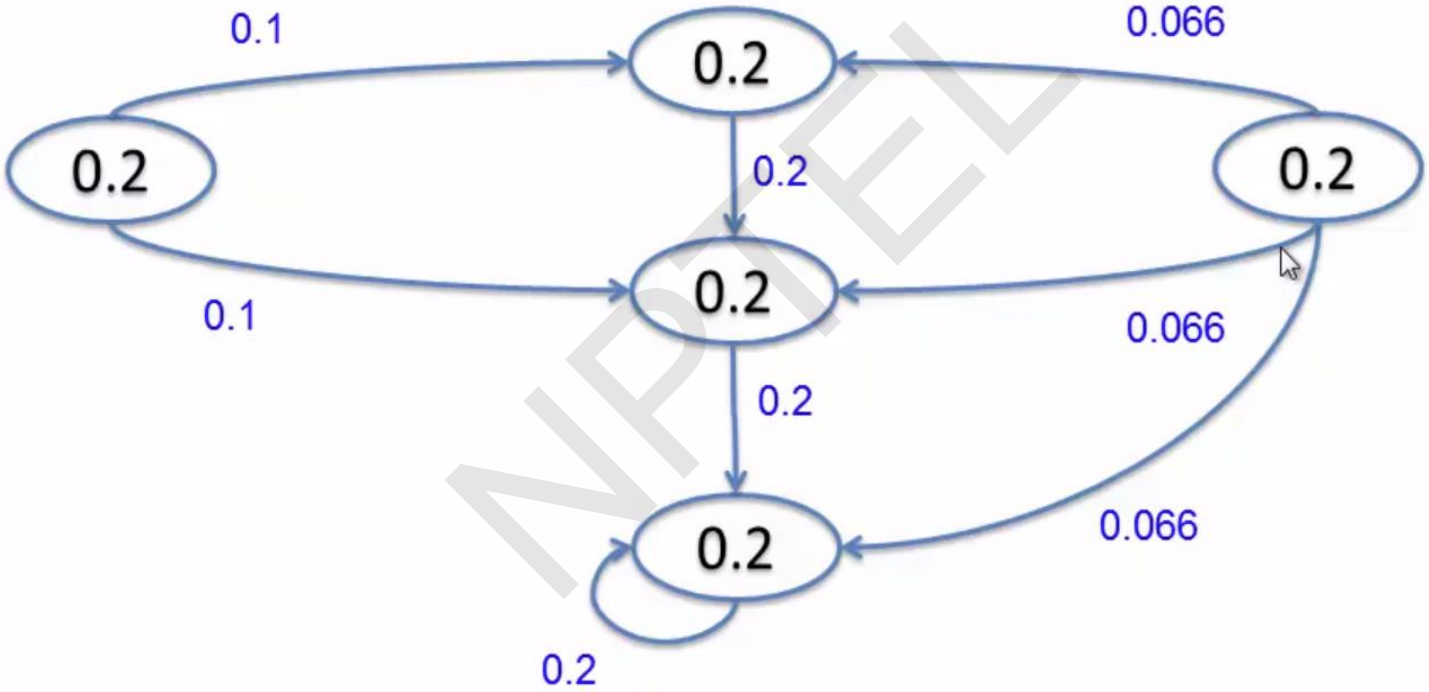
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum

initial PR = .2 = 1/5

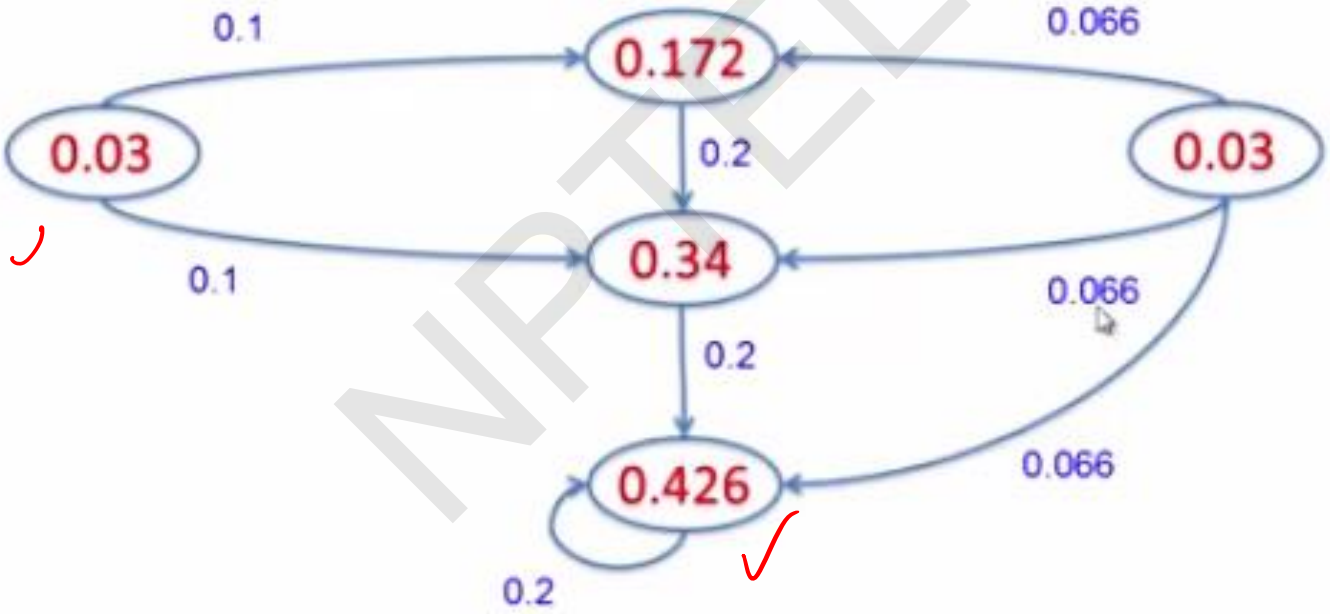
1/5 = .2

= .15 + .85 (.2/1 + .2/3)

Example

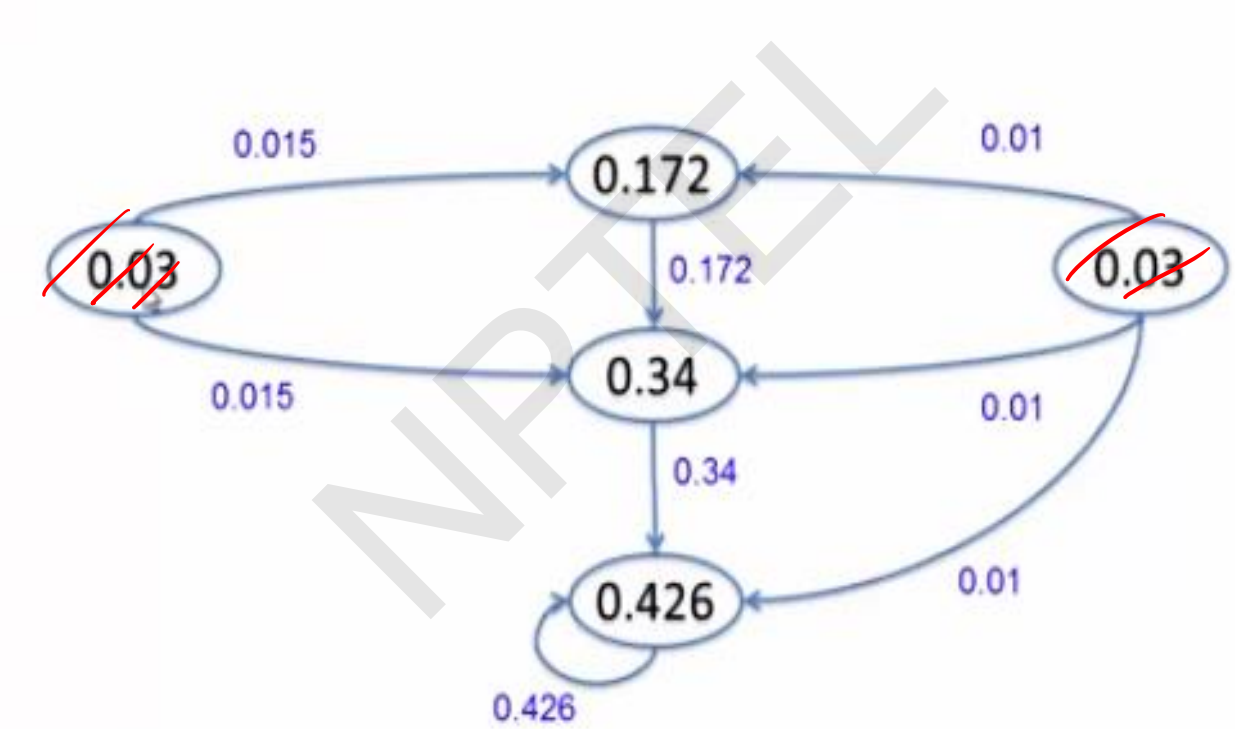


Example



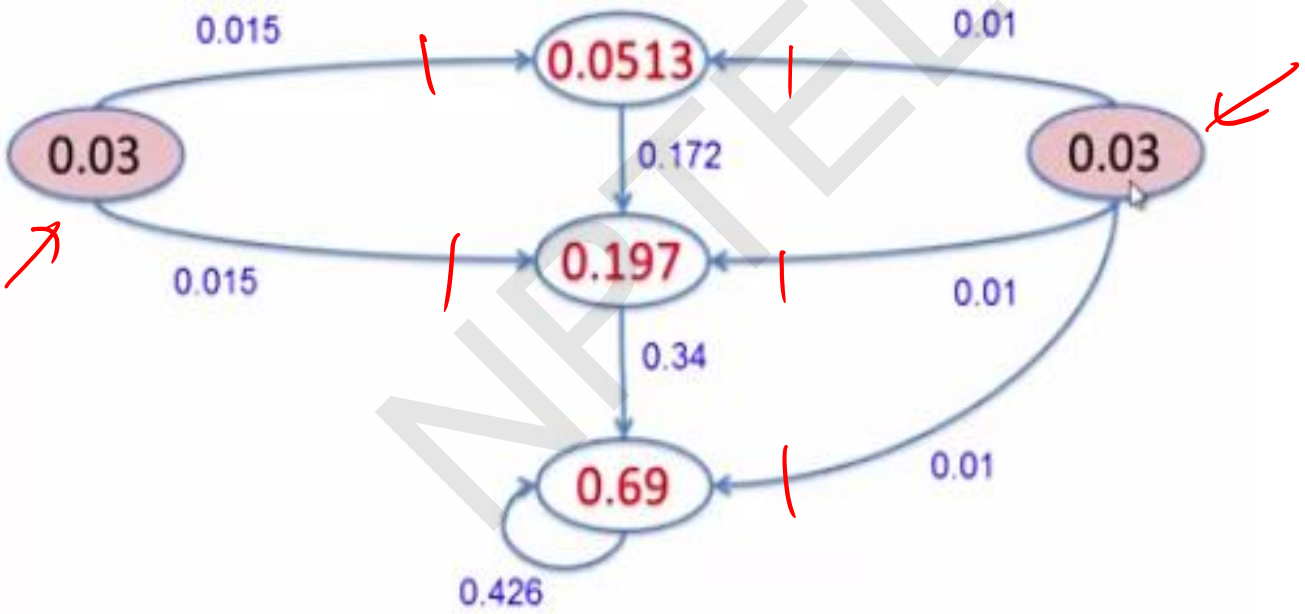
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum

Example



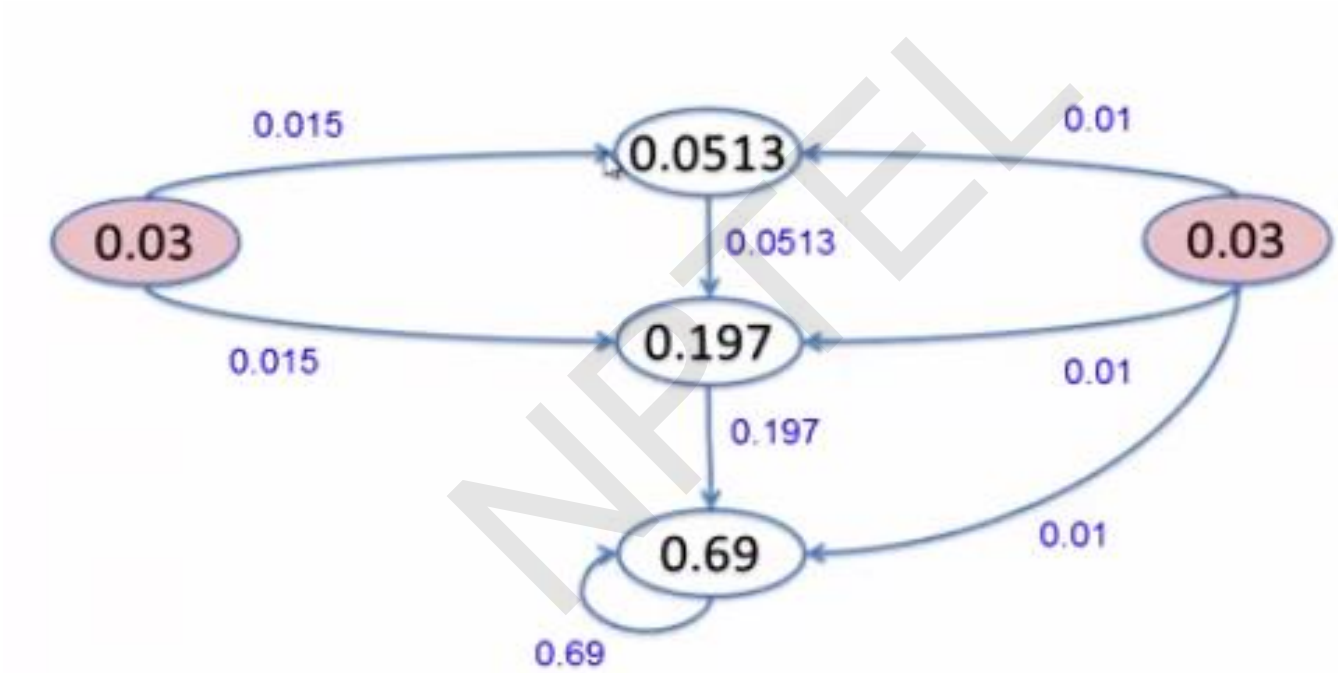
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum

Example



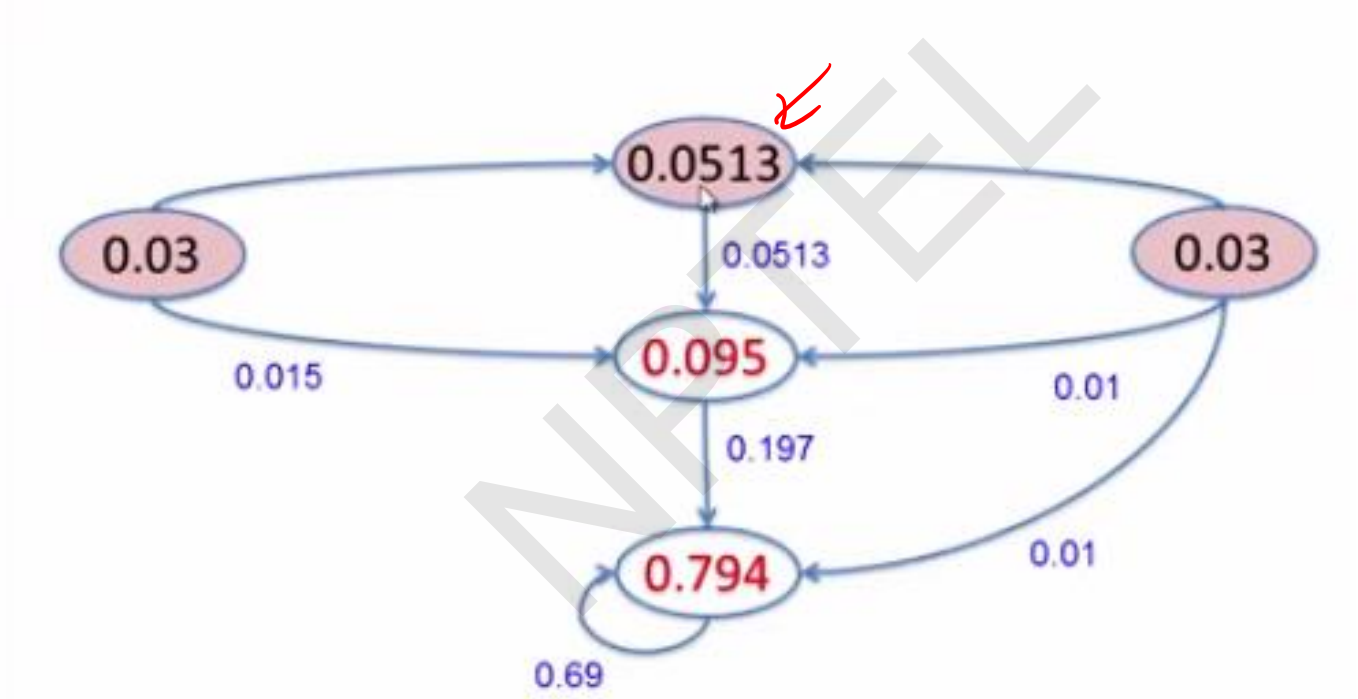
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum

Example



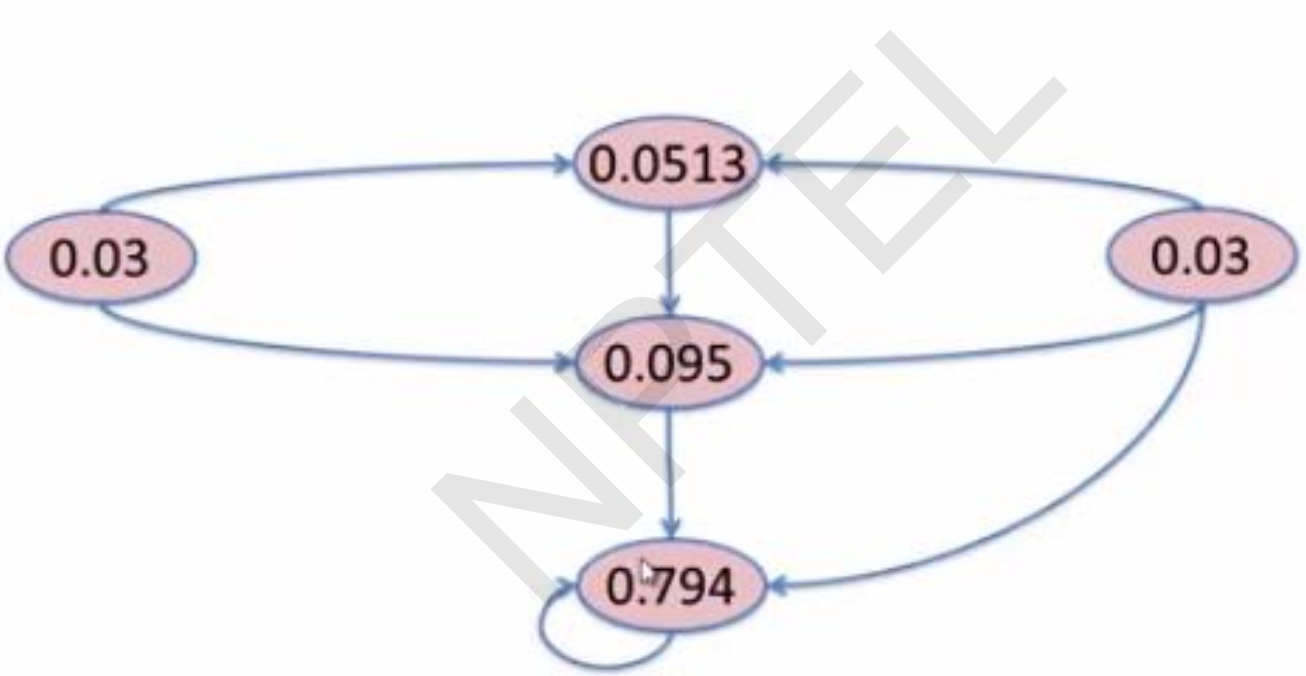
sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum

Example



sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum

Example



sum = sum(incoming values)
rank = 0.15 / 5 + 0.85 * sum

Conclusion

- Graph-structured data are increasingly common in data science contexts due to their ubiquity in modeling the communication between entities: people (social networks), computers (Internet communication), cities and countries (transportation networks), or corporations (financial transactions).

- In this lecture, we have discussed PageRank algorithms for extracting information from graph data using MapReduce and Pregel.

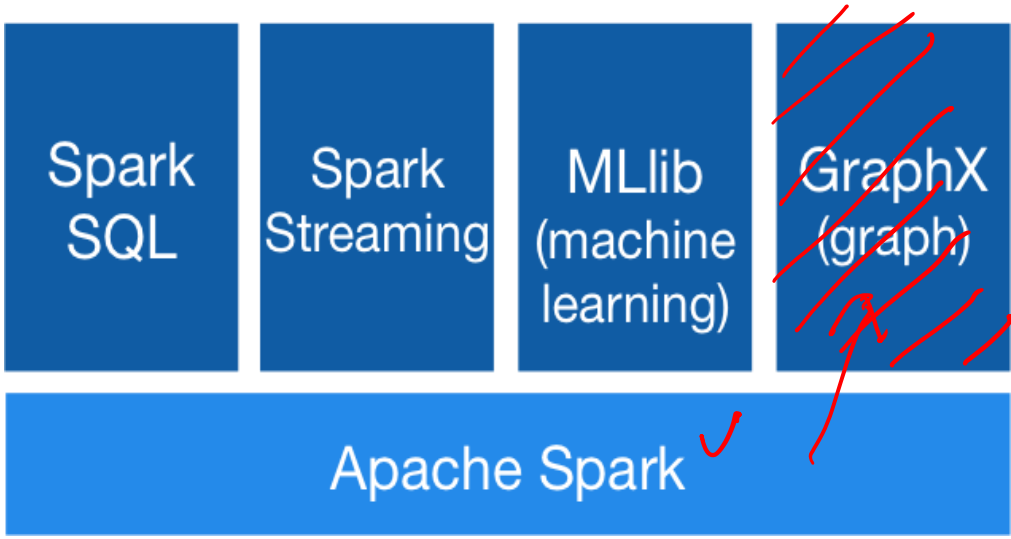
Spark GraphX & Graph Analytics

Dr. Rajiv Misra
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

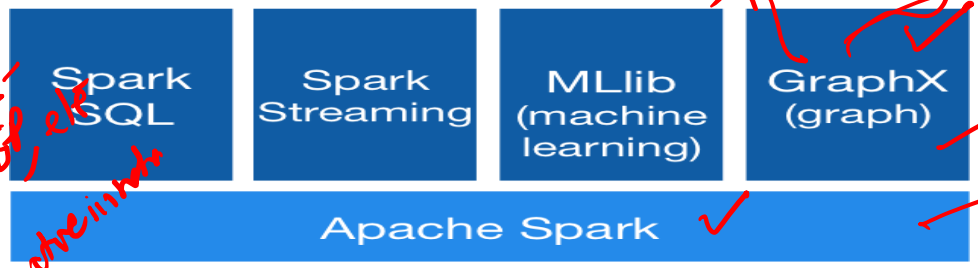
Content of this Lecture:

- In this lecture, we will discuss GraphX: a distributed graph computation framework that unifies graph-parallel and data parallel computation for Big Data Analytics and also discuss as a case study of Graph Analytics with GraphX.

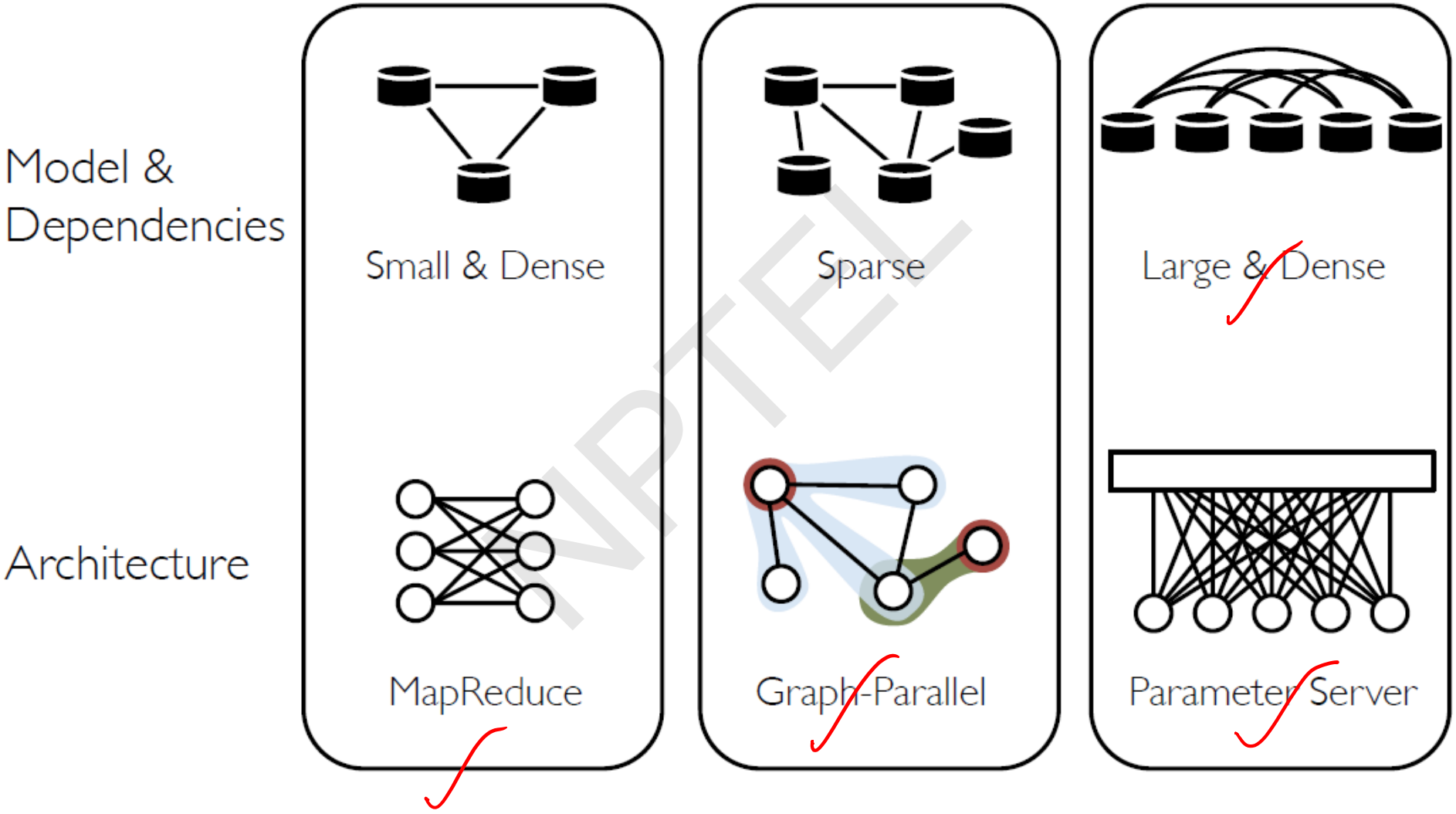


Introduction

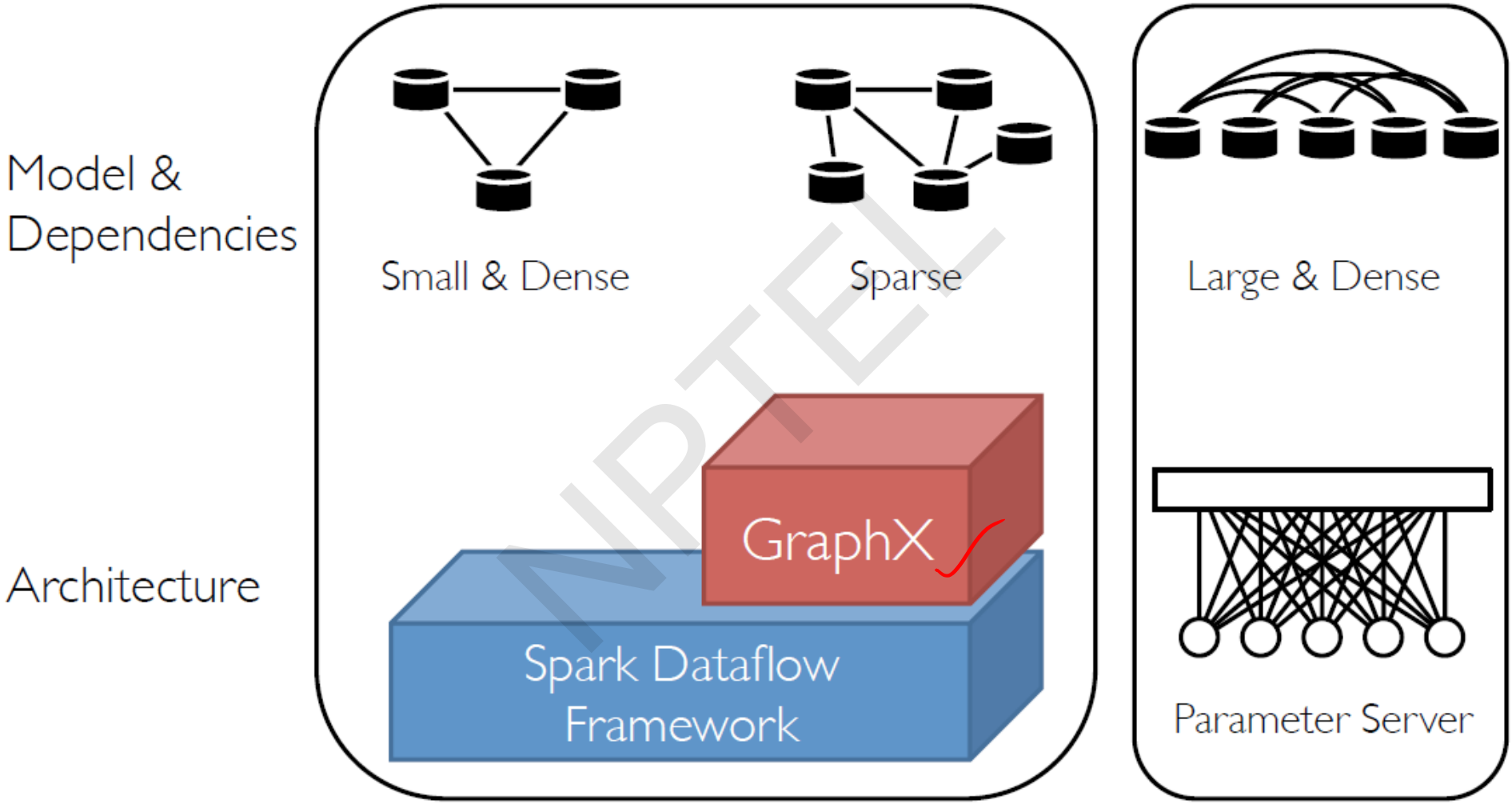
- A piece of technology build on top of spark core.
- Graphs like our social network of graphs in the computer science/network sense.
- However, Graphs are only useful for specific things.
  - It can measure things like “connectedness”, degree distribution, average path length, triangle counts-high level measures of a graph.
  - It can count triangles in the graph, and apply the PageRank algorithm to it.
  - It can also join graphs together and transform graphs quickly
  - It supports the Pregel API(google) for traversing a graph.
    - Introduces VertexRDD and EdgeRDD, and the Edge data type .



Graphs in Machine Learning Landscape



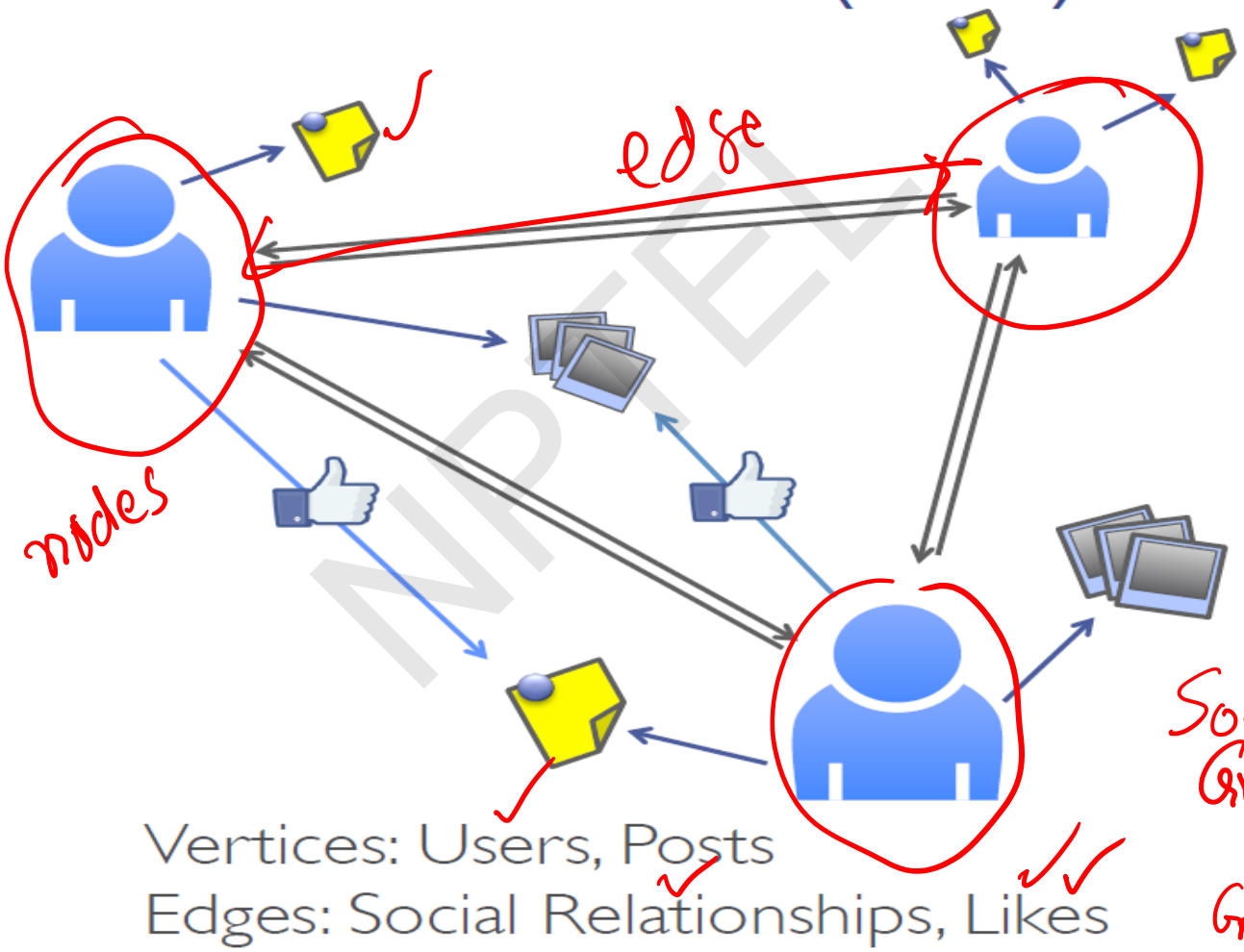
Machine Learning Landscape



Graphs

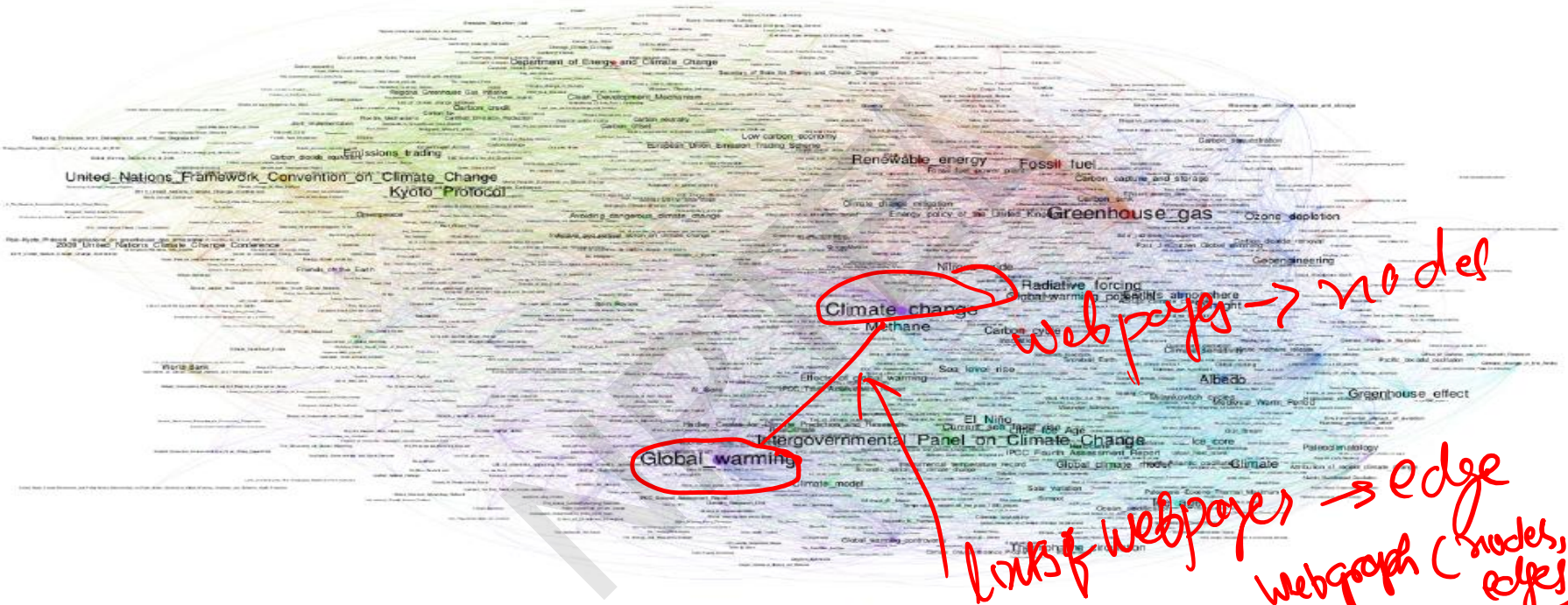
Graph-structured data is everywhere

Social Networks (1/5)



Graph-structured data is everywhere

Web Graphs (2/5) ✓



Wikipedia restricted to 1000 climate change pages

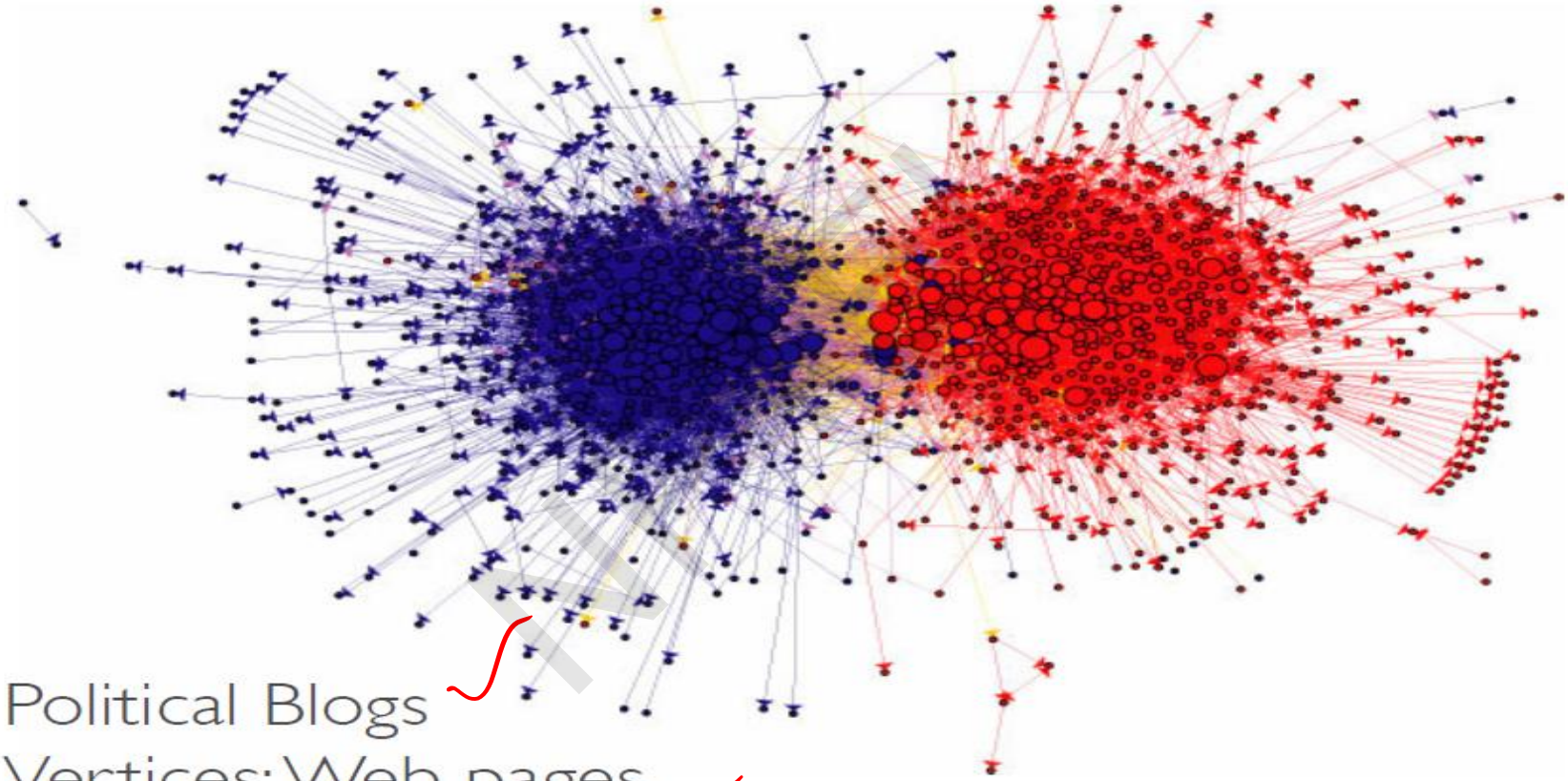
Vertices: Web pages ✓

Edges: Links ✓

http://www.emapsproject.com/blog/archives/1572

Graph-structured data is everywhere

Web Graphs (2/5)

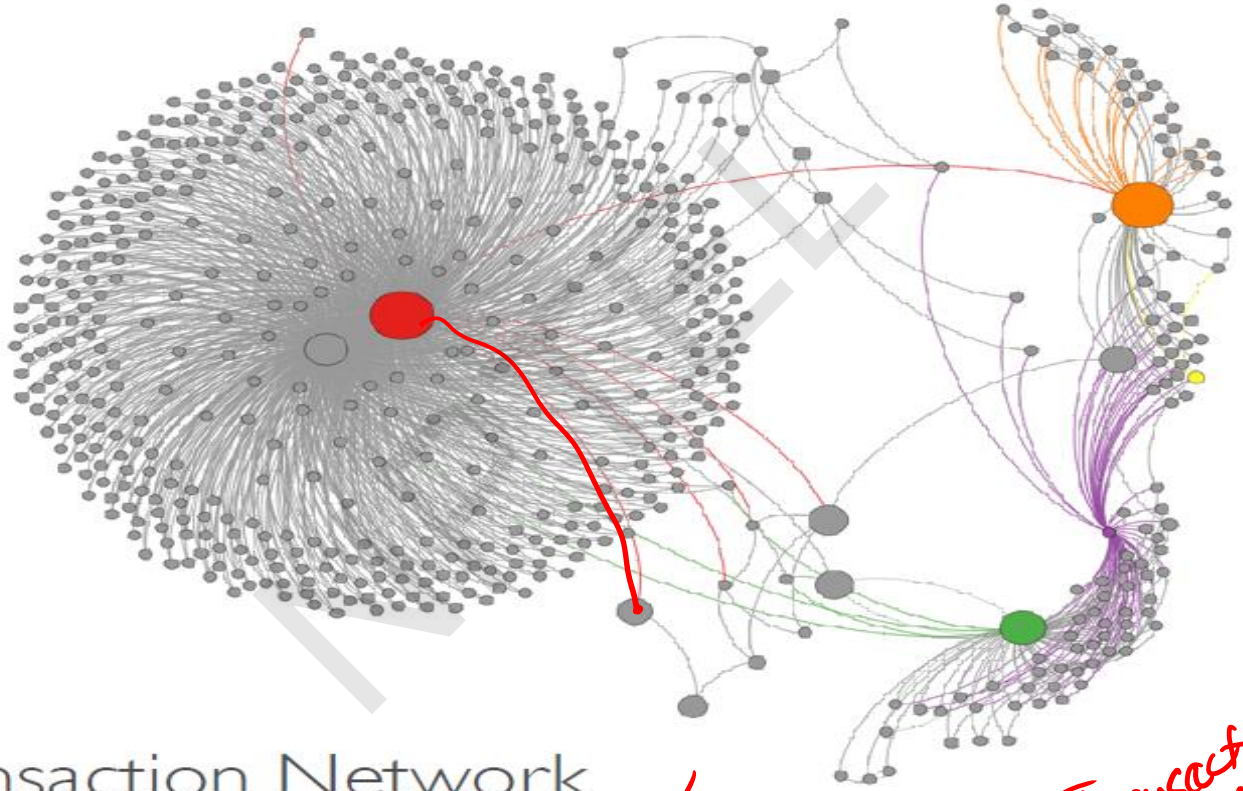


Political Blogs
Vertices: Web pages
Edges: Links

"The political blogosphere and the 2004 U.S. election: divided they blog."
Adamic and Glance, 2005.

Graph-structured data is everywhere

Transaction Networks (3/5)

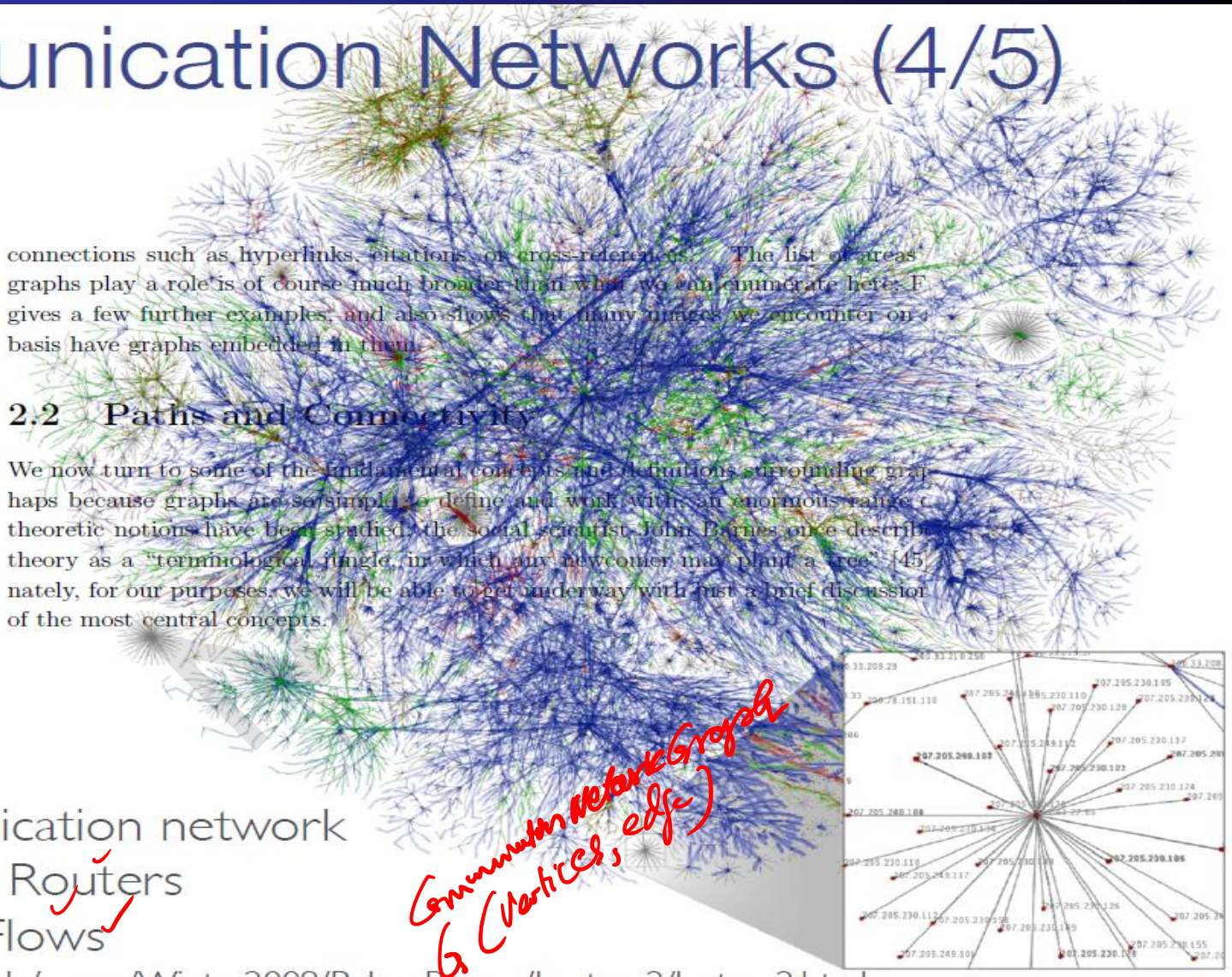


Bitcoin Transaction Network
Vertices: People and Organizations
Edges: Exchange of Currency
http://anonymity-in-bitcoin.blogspot.com/2011/07/bitcoin-is-not-anonymous.html

Bitcoin Transaction network
Graph:
Vertex - People/org
edge - exchange of currency

Graph-structured data is everywhere

Communication Networks (4/5)

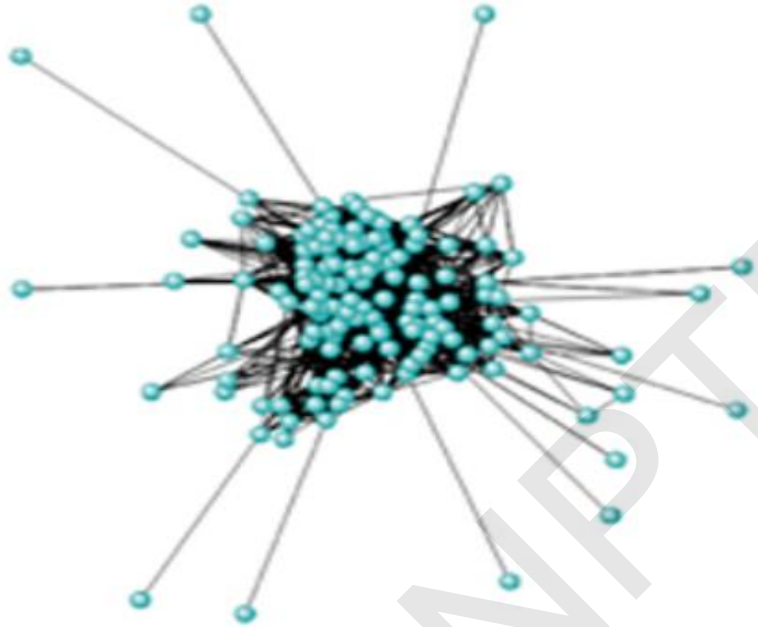


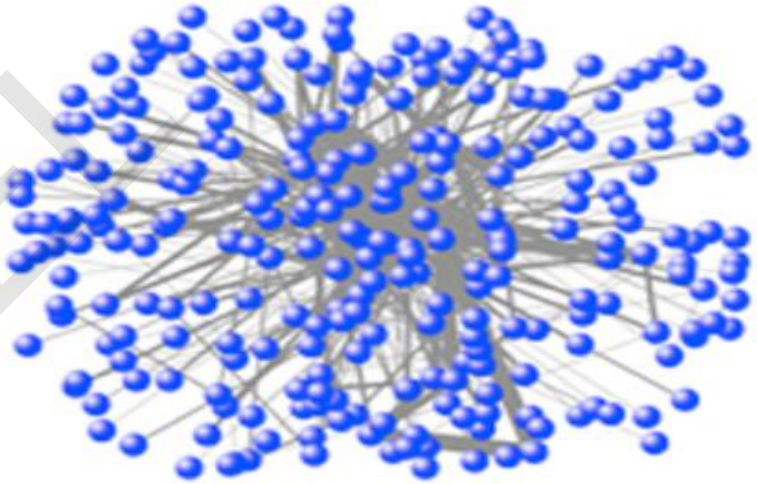
Internet communication network
Vertices: Devices, Routers
Edges: Network Flows

http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture2/lecture2.html

Graph-structured data is everywhere

Communication Networks (4/5)





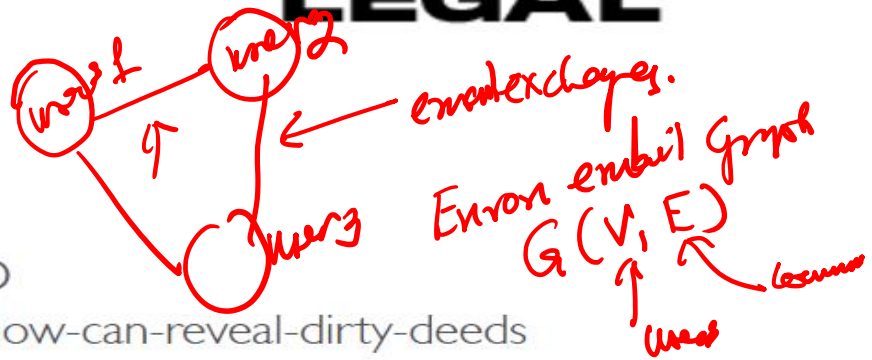
ILLICIT **LEGAL**

Enron Email Graph

Vertices: Users

Directed Edges: Email From → To

https://www.sciencenews.org/article/information-flow-can-reveal-dirty-deeds



Graph-structured data is everywhere

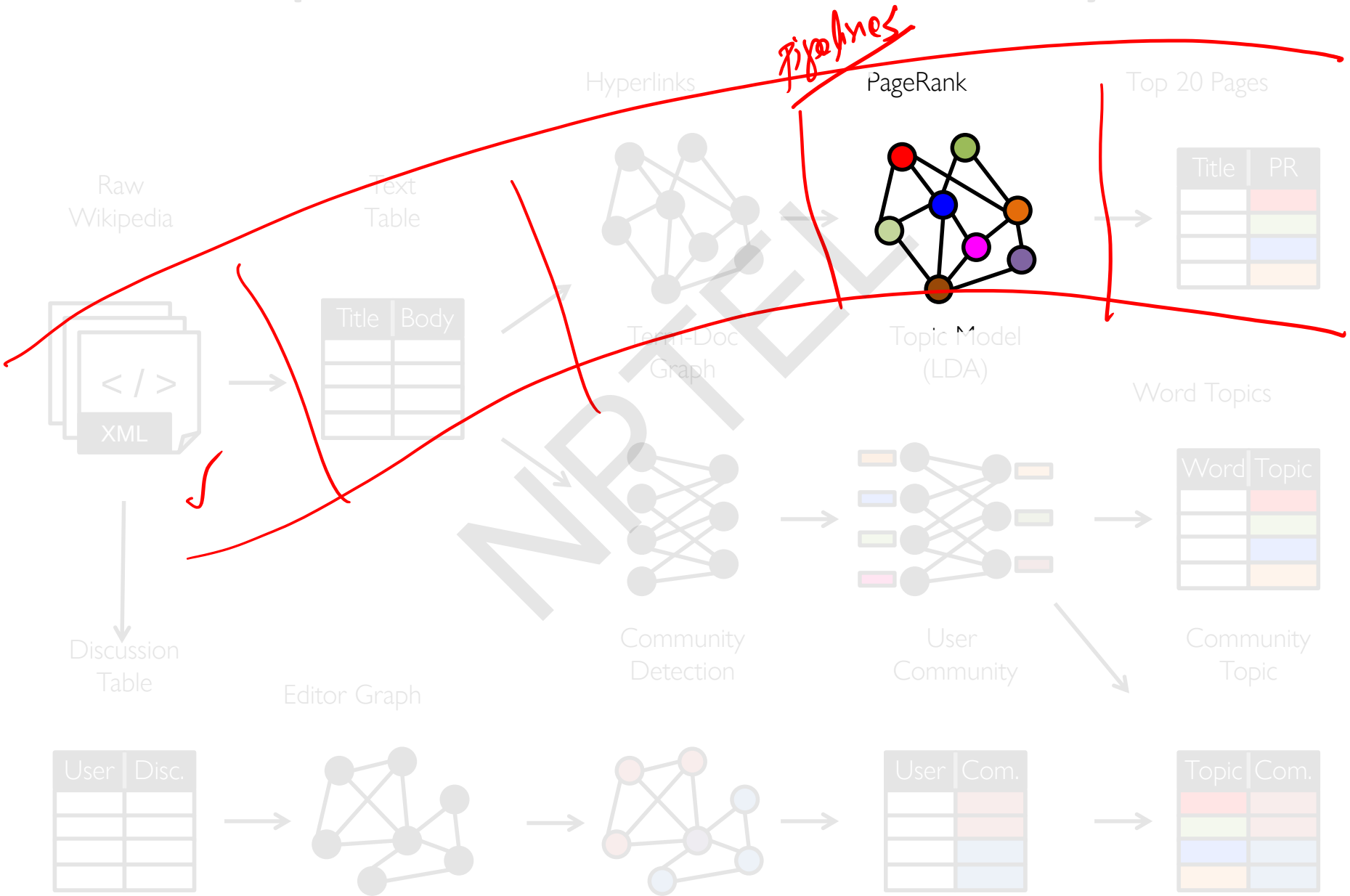
User-Item Graphs (5/5)

Vertices: *Users and Items*

Edges: *Ratings*

Graph Algorithms

Graphs are Central to Analytics



PageRank: Identifying Leaders

$$R[i] = 0.15 + \sum_{j \in \mathrm{Nbrs}(i)} w_{ji} R[j]$$

Rank of user *i*

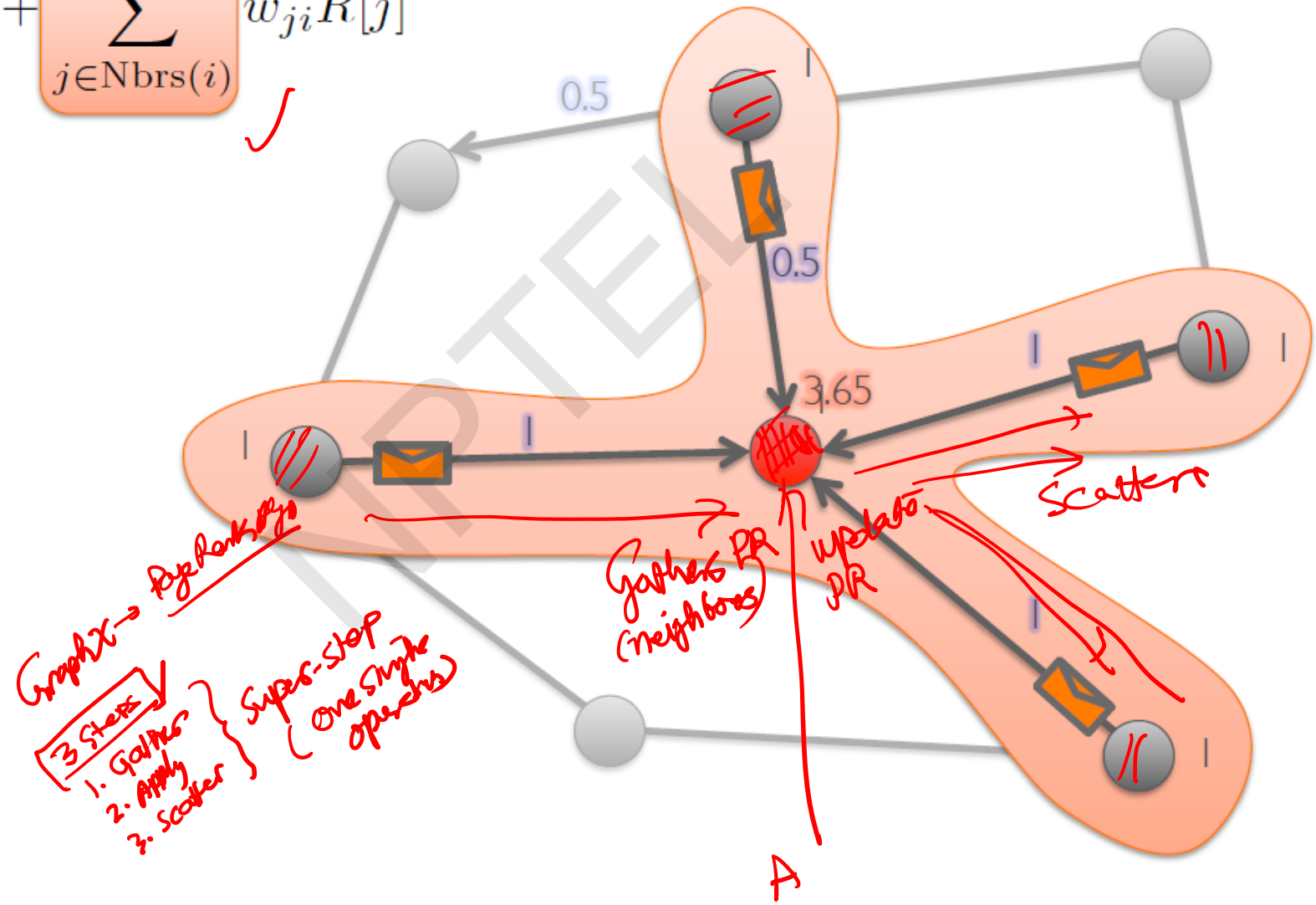
Weighted sum of neighbors' ranks

Update ranks in parallel

Iterate until convergence

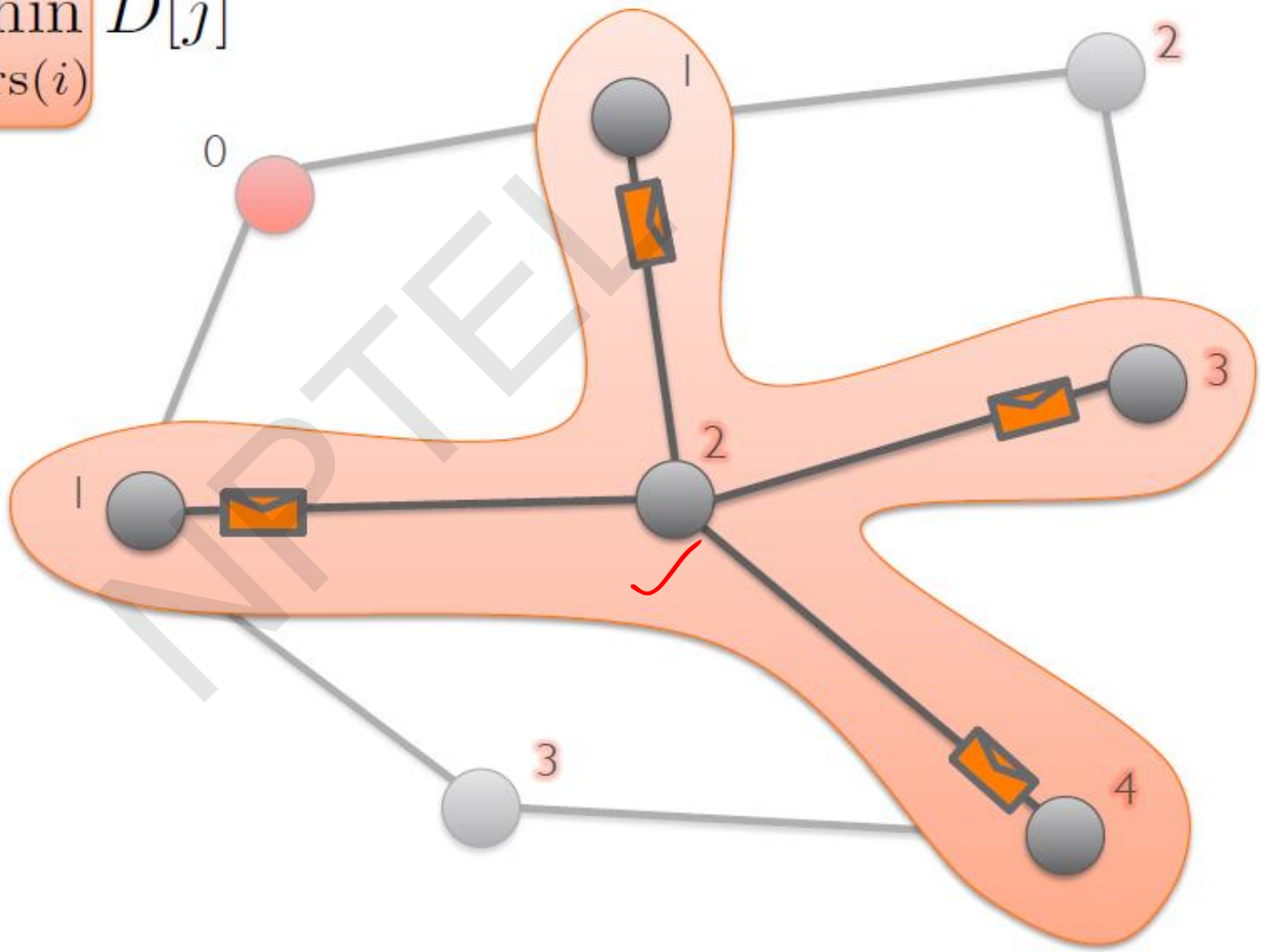
PageRank

$$R[i] = 0.15 + \sum_{j \in \mathrm{Nbrs}(i)} w_{ji} R[j]$$



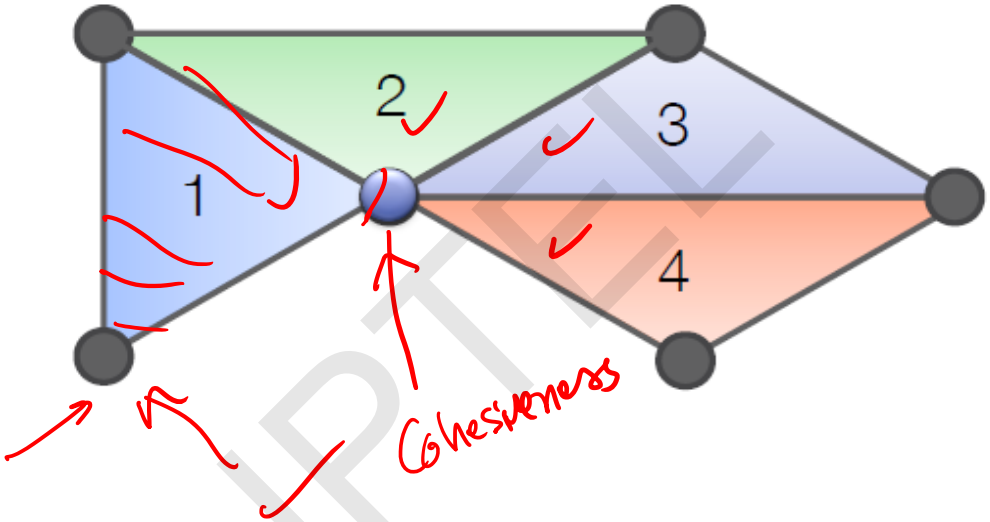
Single-Source Shortest Path

$$D[i] = 1 + \underset{j \in \text{Nbrs}(i)}{\arg\min} D[j]$$



Finding Communities

Count triangles passing through each vertex:

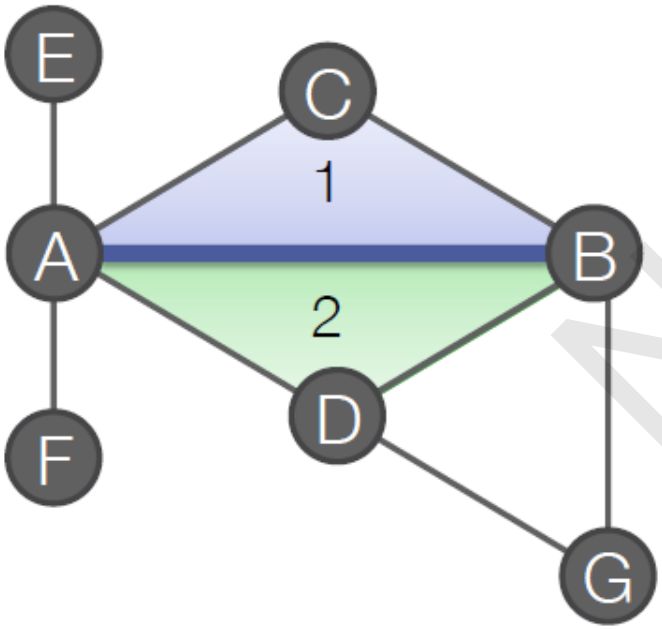


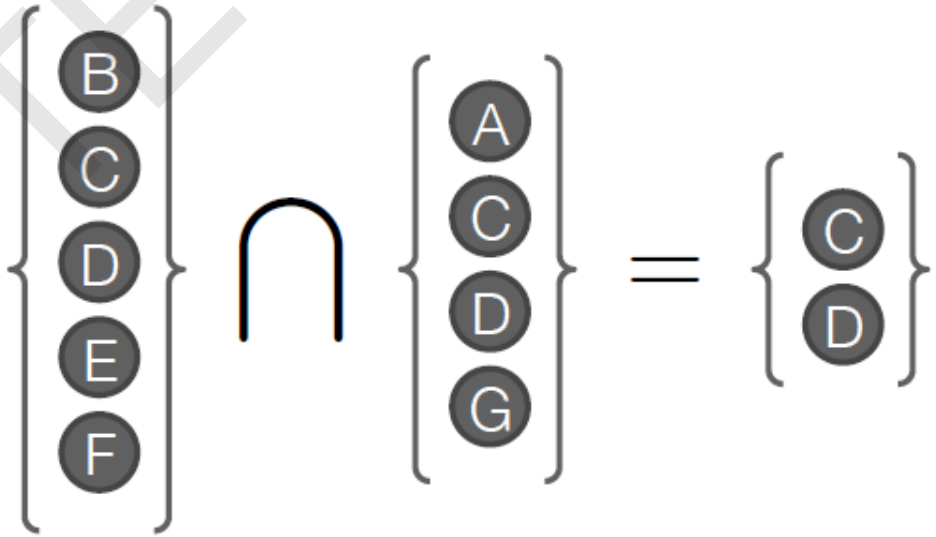
Measure "cohesiveness" of local community

$$\text{ClusterCoeff}[i] = \frac{2 * \#\text{Triangles}[i]}{\text{Deg}[i] * (\text{Deg}[i] - 1)}$$

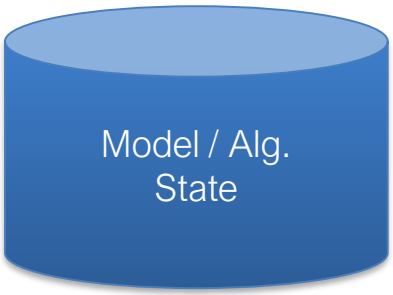
Counting Triangles

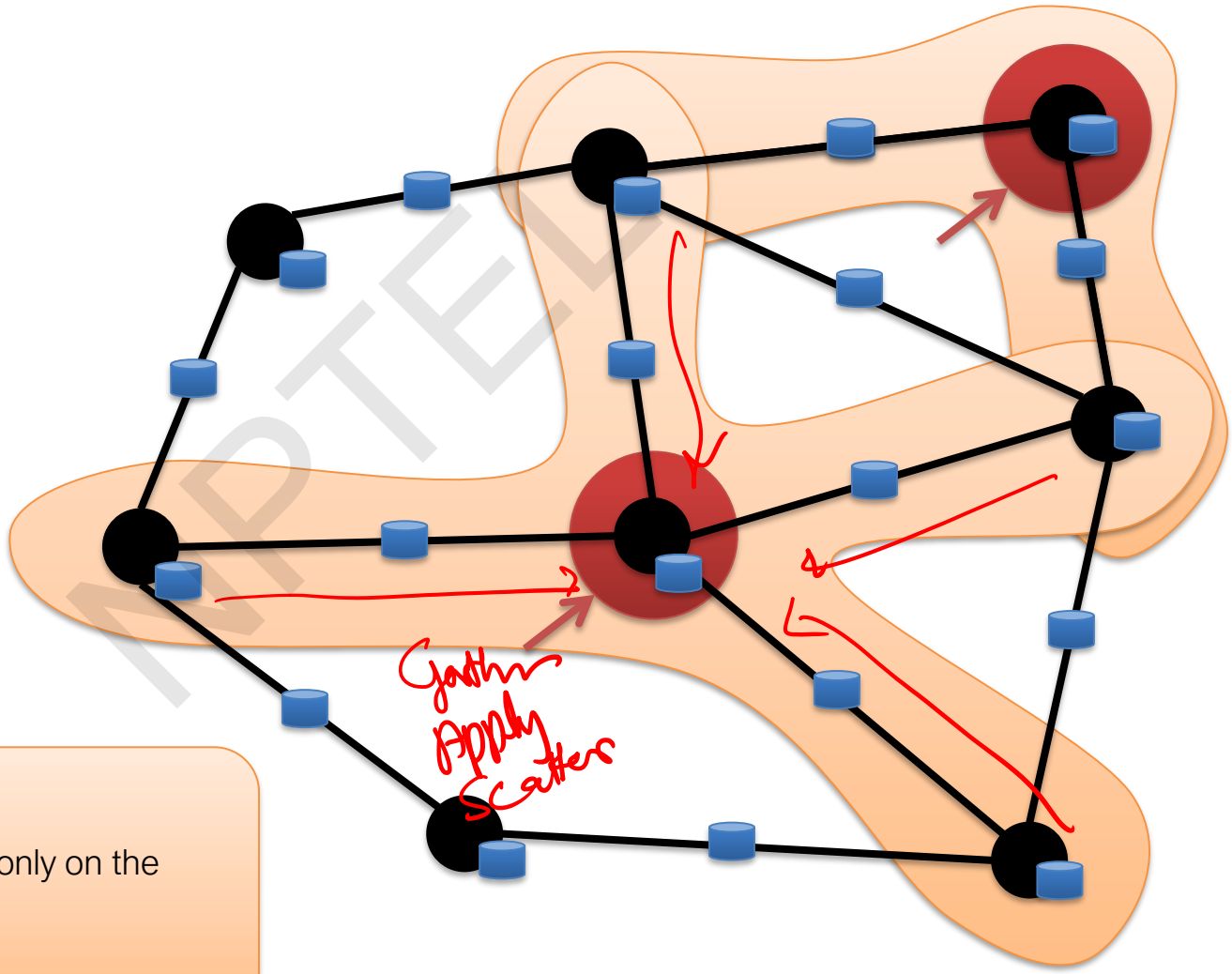
Count triangles passing through each vertex by counting triangles on each edge:





The Graph-Parallel Pattern

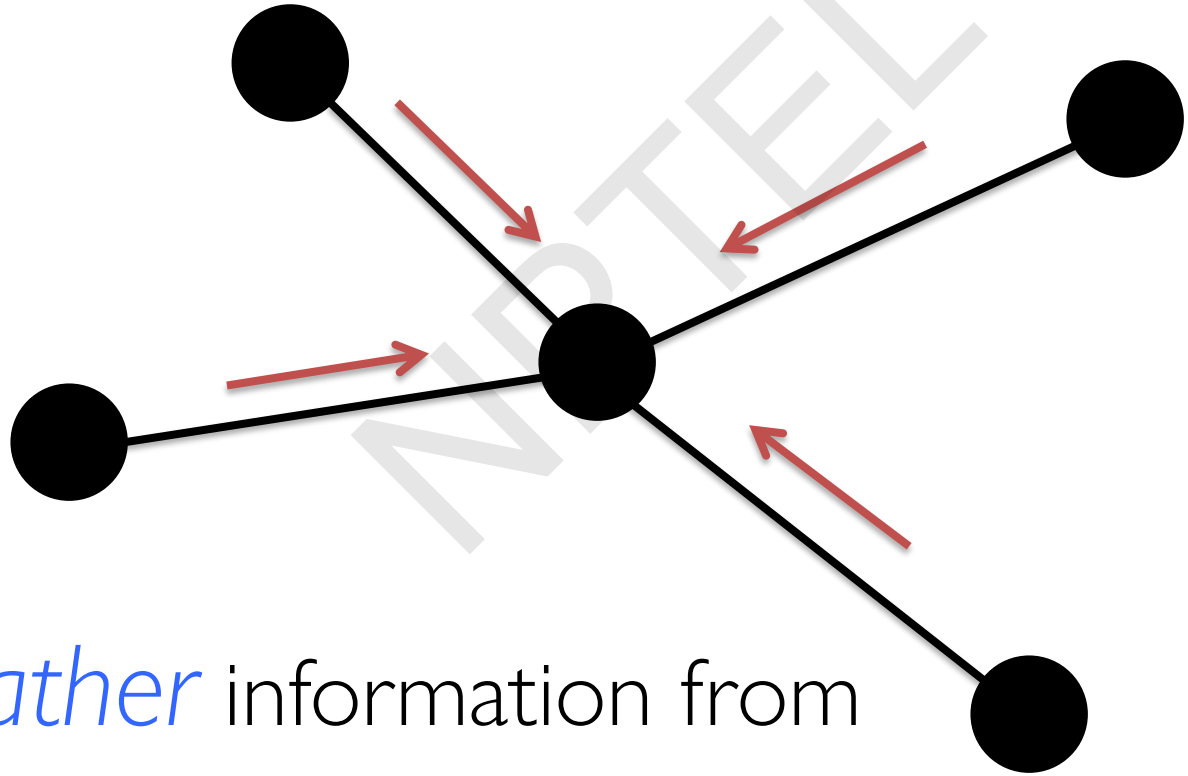




Computation depends only on the **neighbors**

Graph-Parallel Pattern

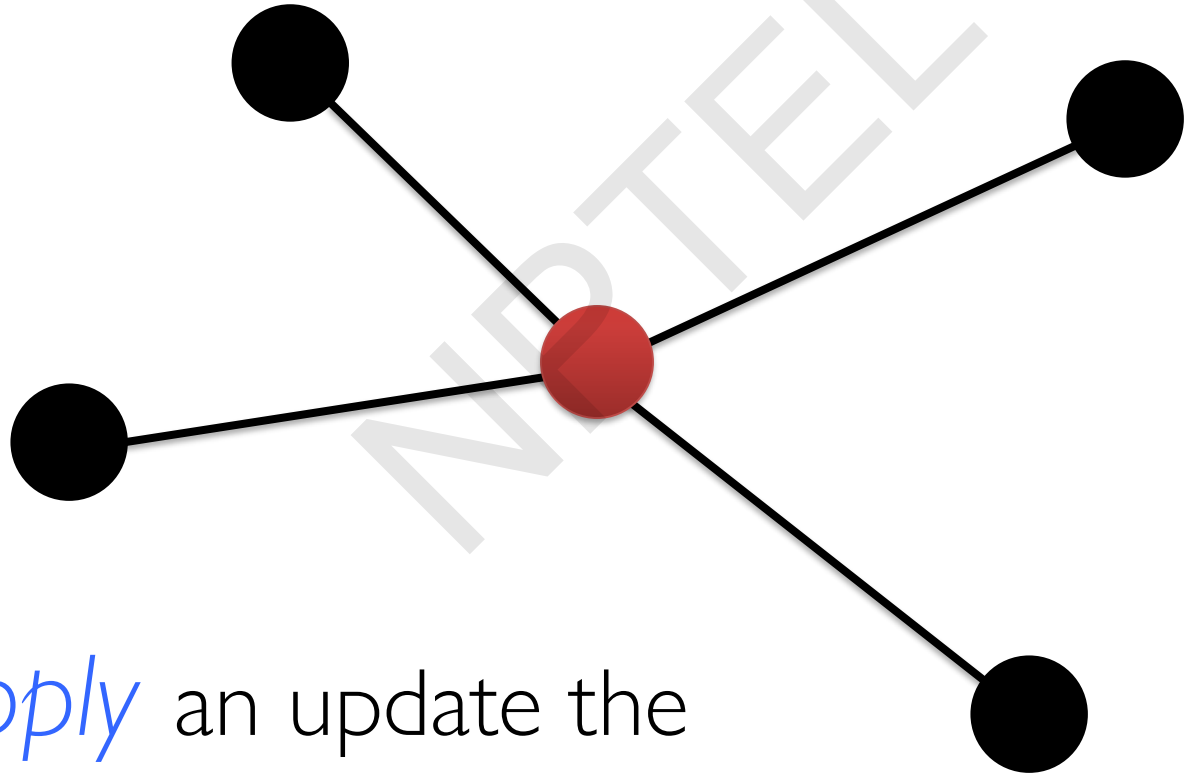
- Gonzalez et al. [OSDI'12]



Gather information from neighboring vertices

Graph-Parallel Pattern

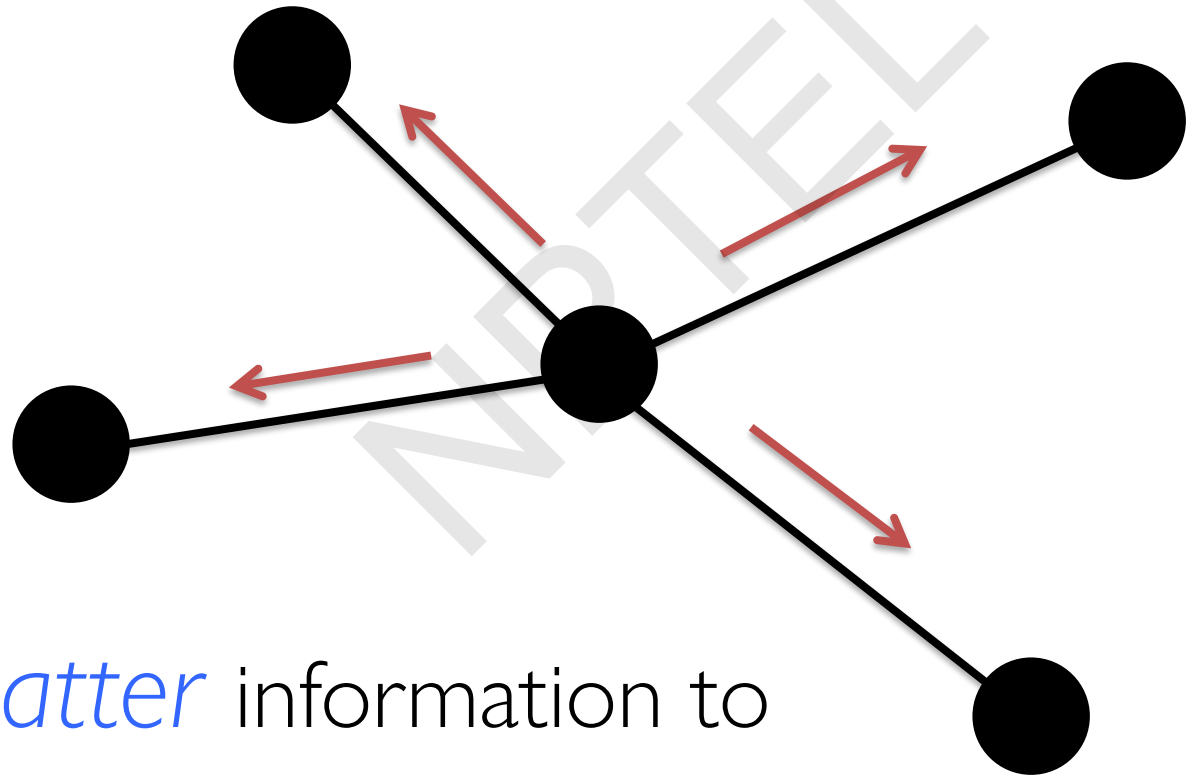
- Gonzalez et al. [OSDI'12]



Apply an update the vertex property

Graph-Parallel Pattern

- Gonzalez et al. [OSDI'12]



Scatter information to neighboring vertices

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

Graph-Parallel Systems

Expose specialized APIs to simplify graph programming.

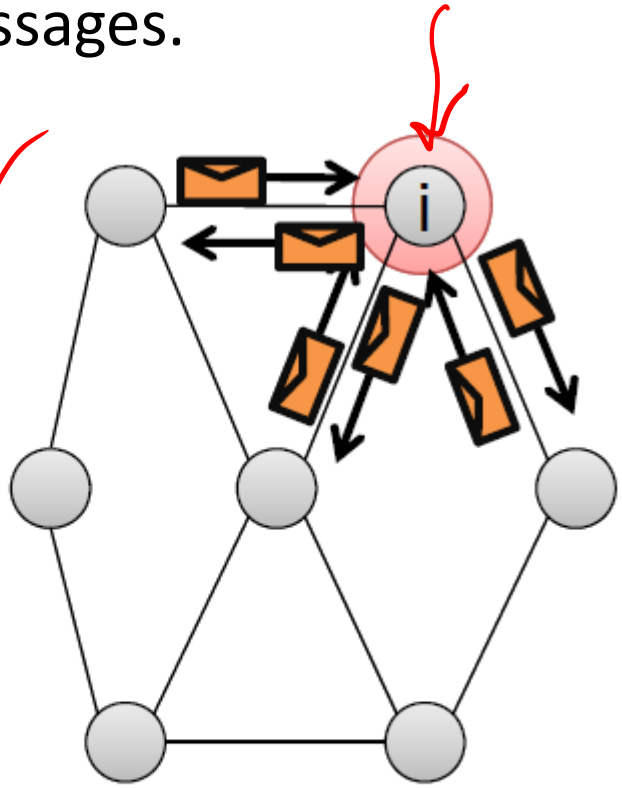
The Pregel (Push) Abstraction

- "Think like a Vertex." - Pregel [SIGMOD'10]
- Vertex-Programs interact by sending messages.

```
Pregel_PageRank(i, messages) :
  // Receive all the messages
  total = 0
  foreach( msg in messages) :
    total = total + msg

  // Update the rank of this vertex
  R[i] = 0.15 + total

  // Send new messages to neighbors
  foreach(j in out_neighbors[i]) :
    Send  msg(R[i]) to vertex j
```

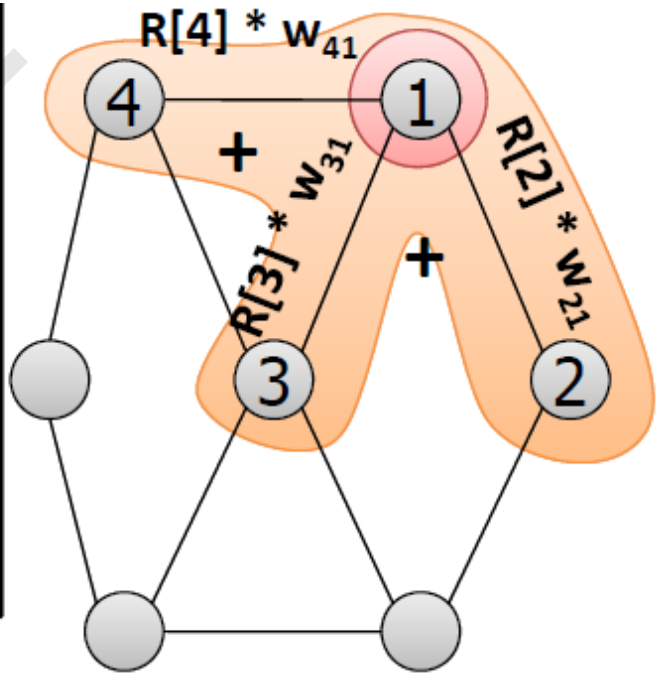


The GraphLab (Pull) Abstraction

- Vertex Programs directly access adjacent vertices and edges

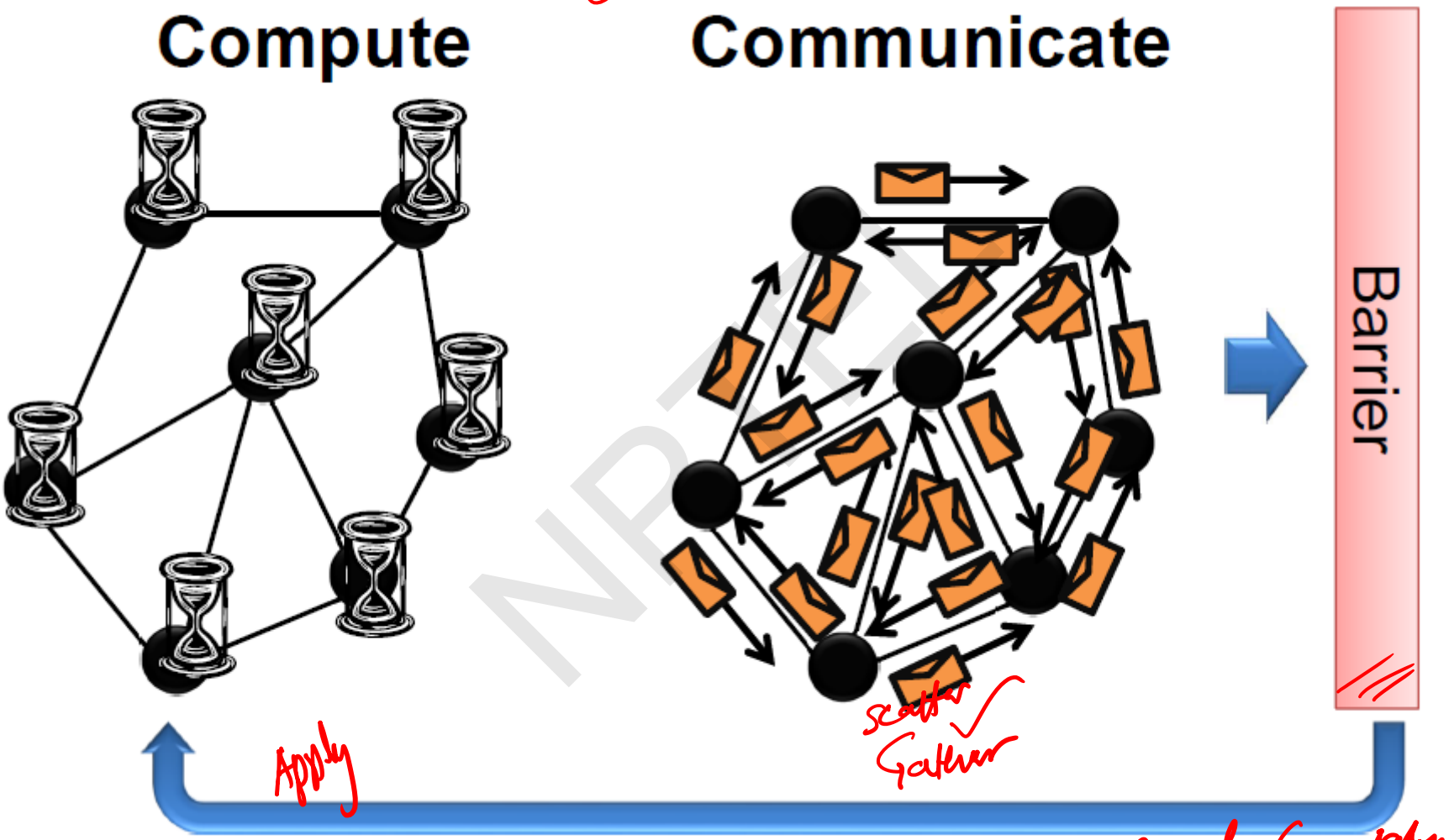
```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in neighbors(i)):
    total = total + R[j] * w_ji

  // Update the PageRank
  R[i] = 0.15 + total
```



- Data movement is managed by the system and not the user.

Iterative Bulk Synchronous Execution

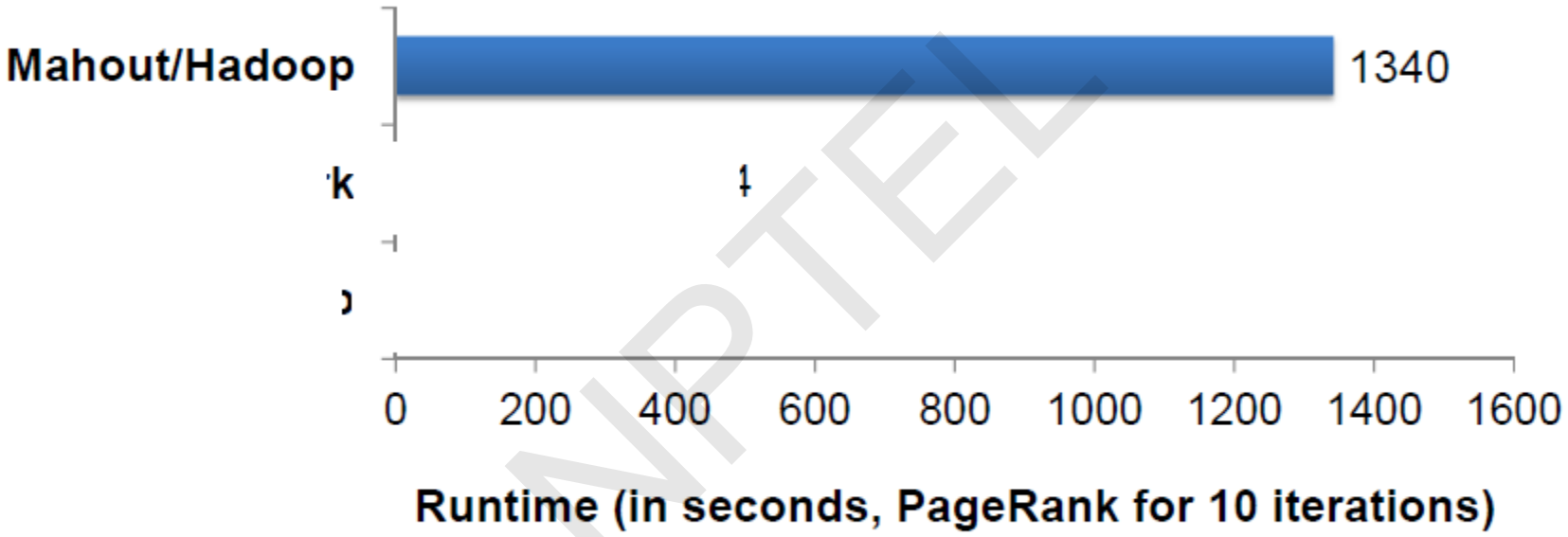


Graph-Parallel Systems

Expose specialized APIs to simplify graph programming.

Exploit graph structure to achieve orders-of-magnitude performance gains over more general data-parallel systems.

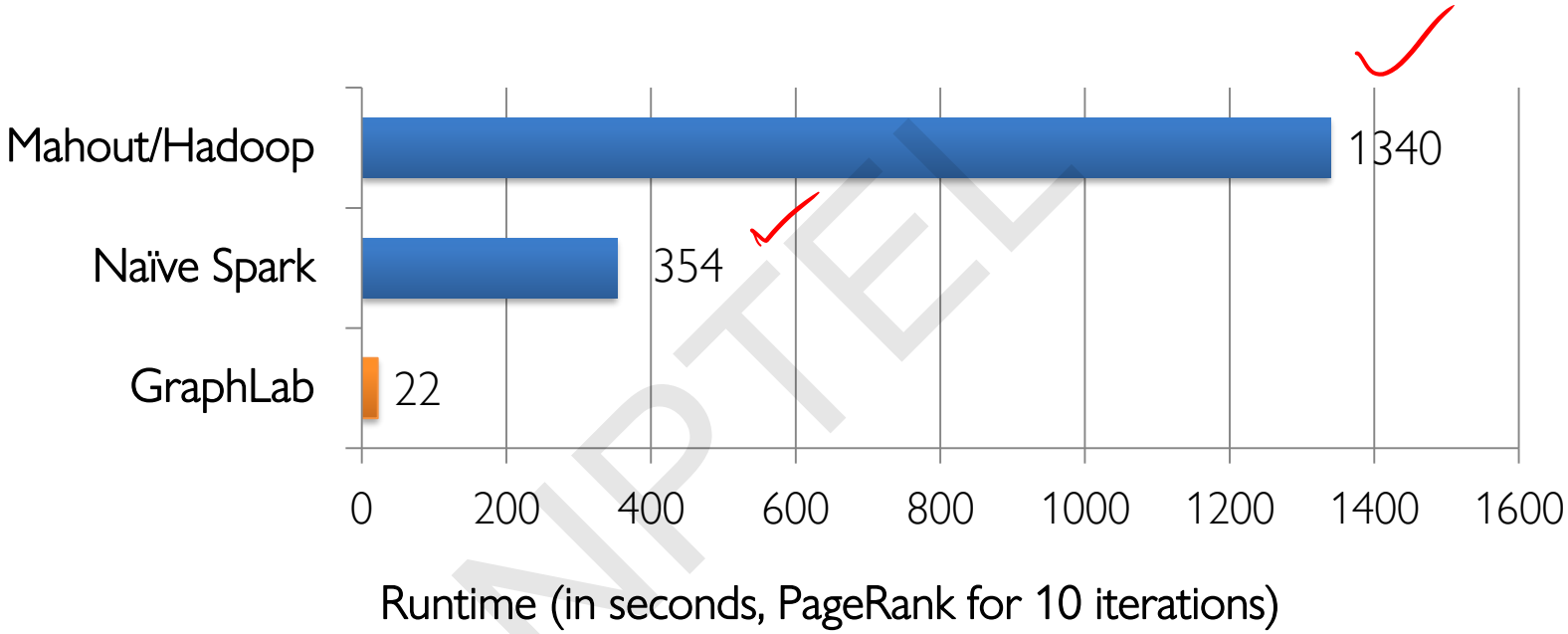
PageRank on the Live-Journal Graph



Spark is 4x faster than Hadoop

GraphLab is 16x faster than Spark

PageRank on the Live-Journal Graph



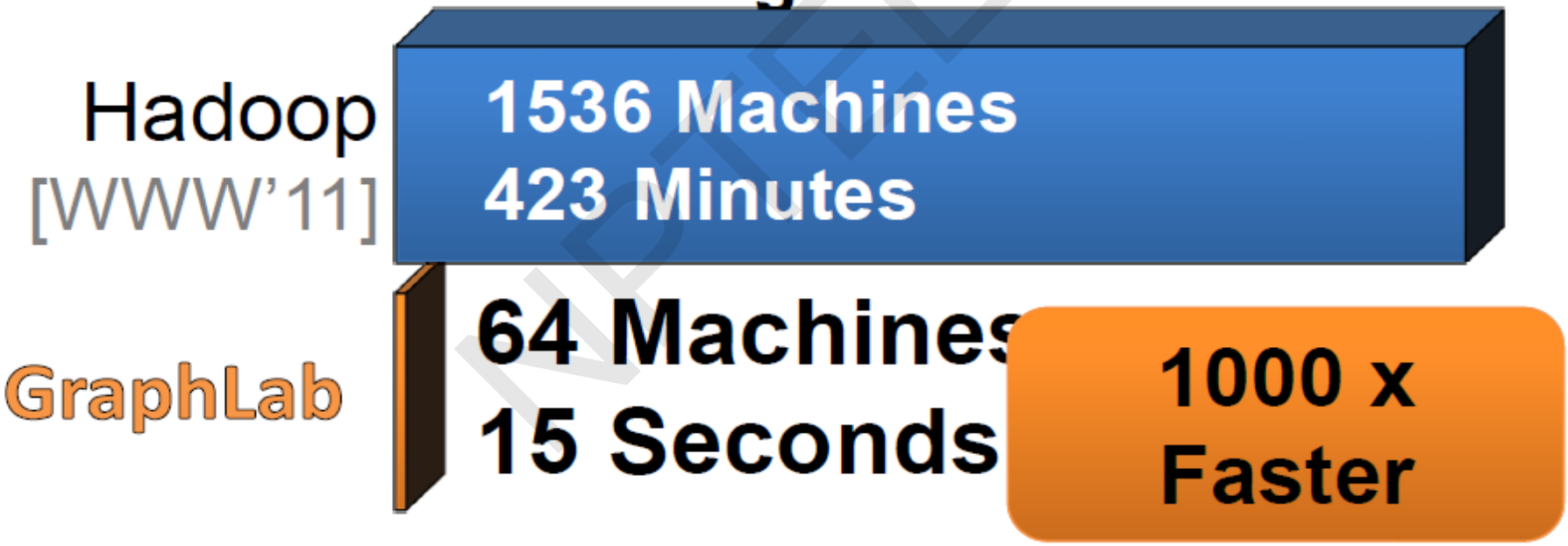
GraphLab is *60x faster* than Hadoop

GraphLab is *16x faster* than Spark

Triangle Counting on Twitter

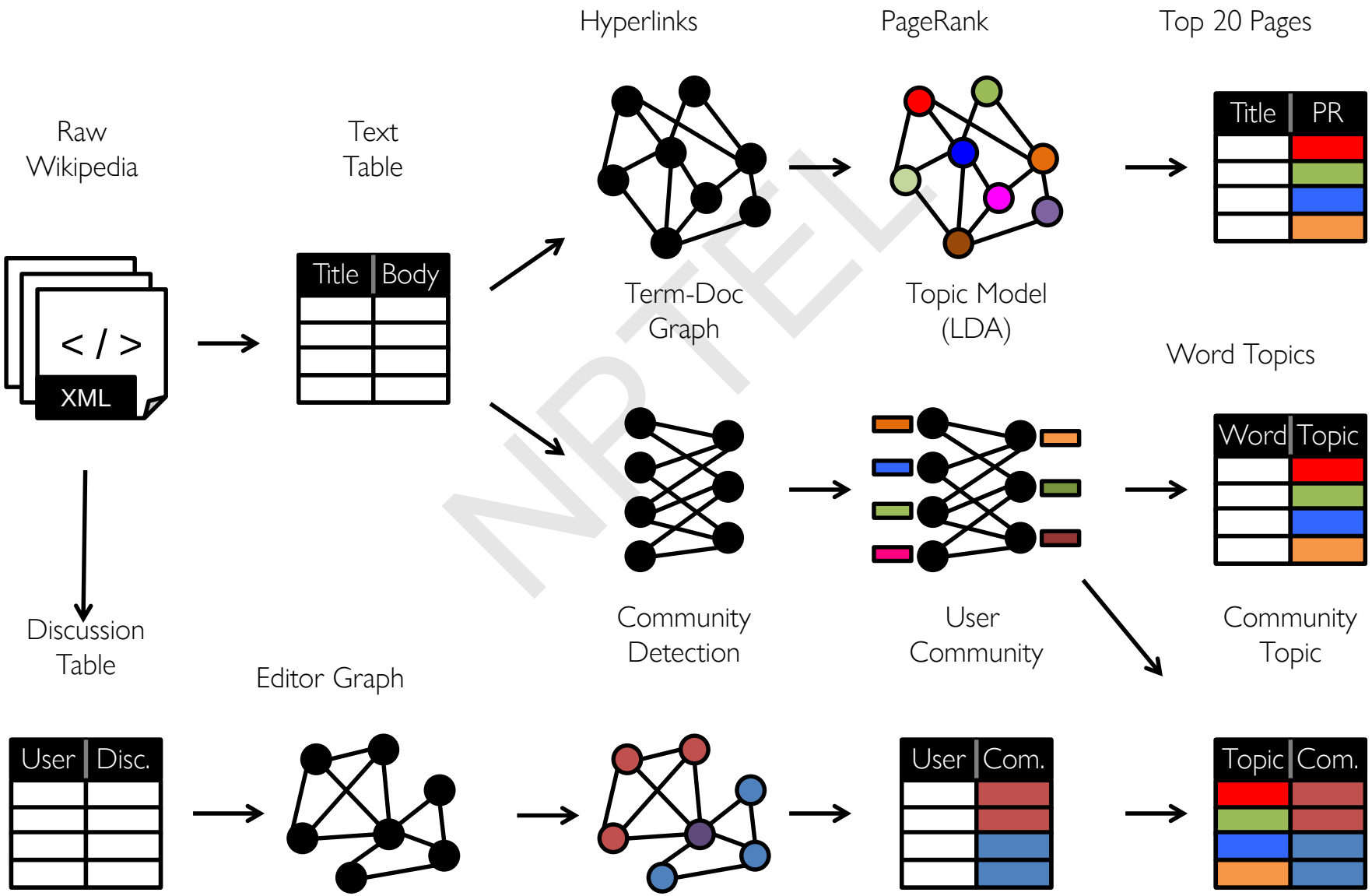
40M Users, 1.4 Billion Links

Counted: 34.8 Billion Triangles



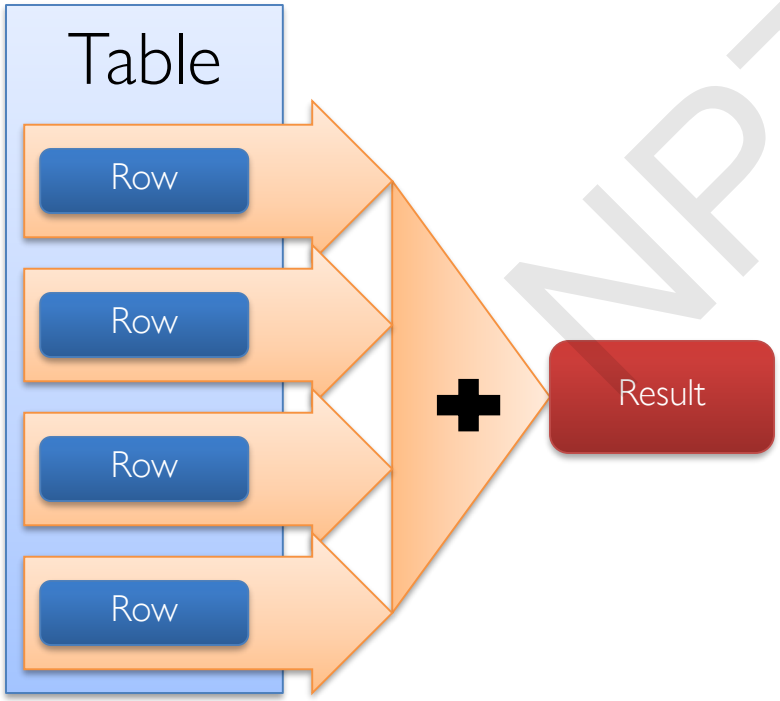
S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," WWW'11

Graphs are Central to Analytics

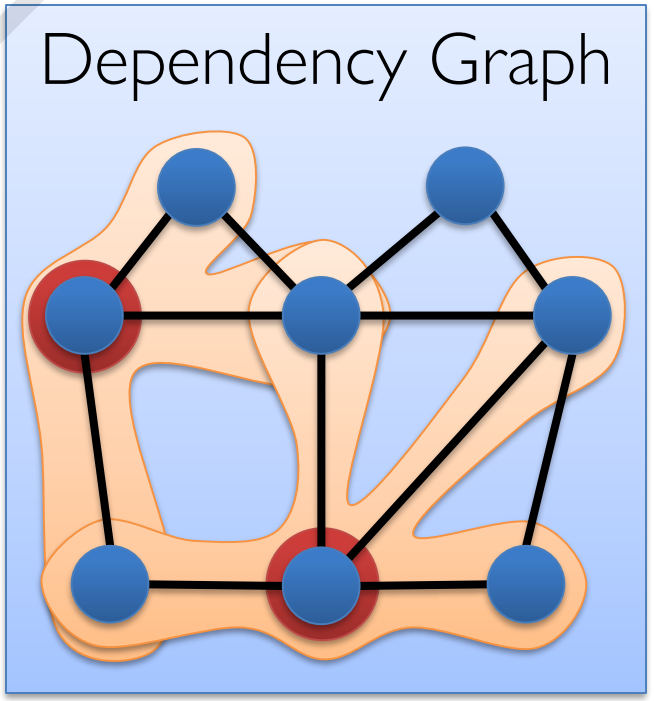


Separate Systems to Support Each View

Table View



Graph View



Having separate systems

for each view is

difficult to use and

inefficient

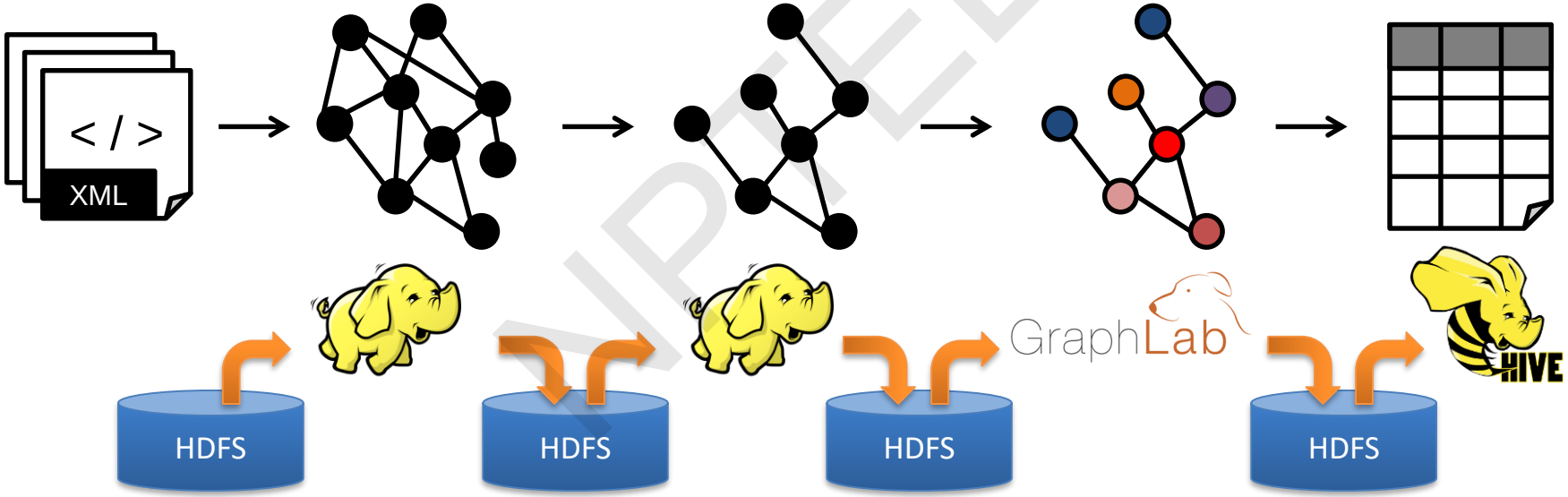
Difficult to Program and Use

Users must ***Learn***, ***Deploy***, and ***Manage*** multiple systems

Leads to brittle and often complex interfaces

Inefficient

Extensive data movement and duplication across the network and file system

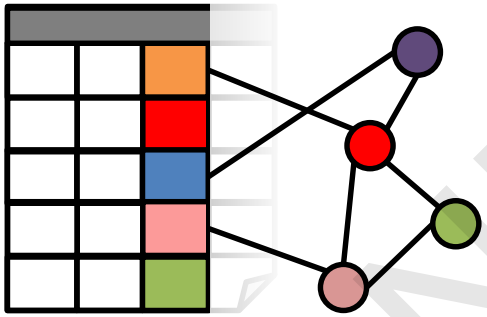


Limited reuse internal data-structures across stages

Solution: The GraphX Unified Approach

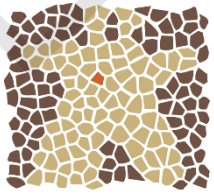
New API

Blurs the distinction between Tables and Graphs



New System

Combines Data-Parallel Graph-Parallel Systems



Enabling users to easily and efficiently express the entire graph analytics pipeline

Tables and Graphs are composable views of the same physical data

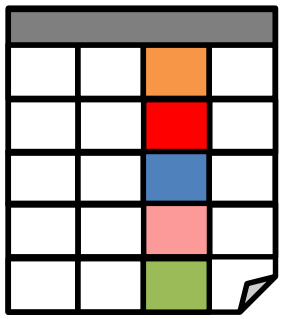
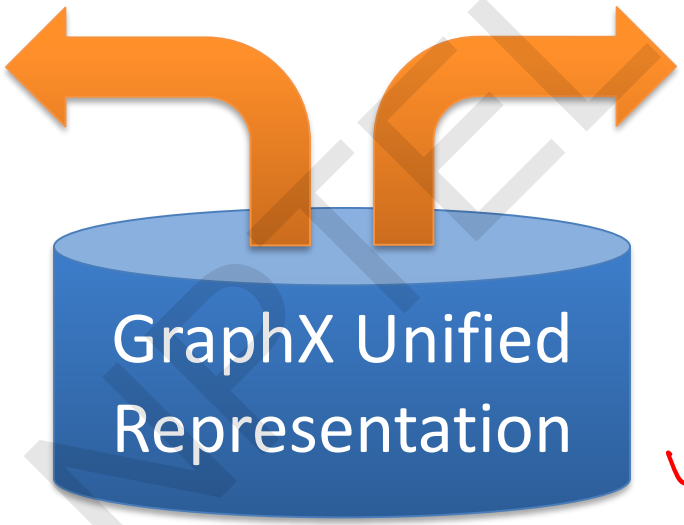
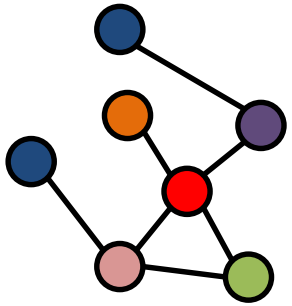


Table View





Graph View

data parallel

graph parallel

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

Graphs → Relational Algebra

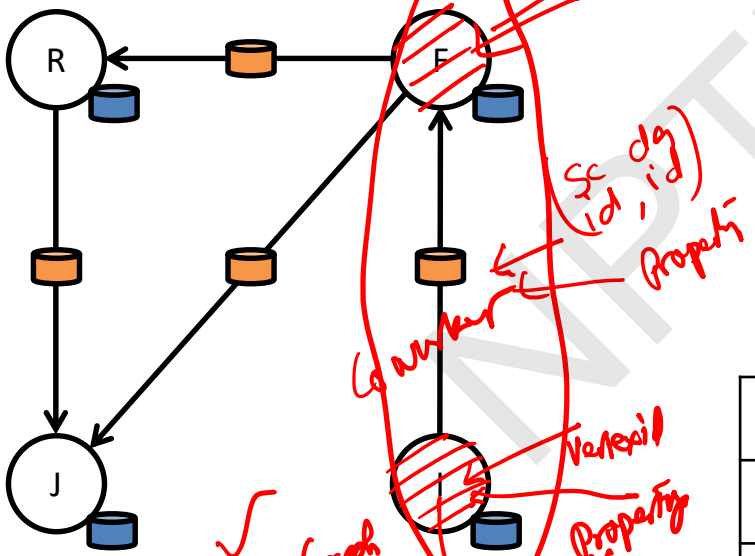
1. Encode graphs as distributed tables
2. Express graph computation in relational algebra
3. Recast graph systems optimizations as:
    1. Distributed join optimization
    2. Incremental materialized maintenance

Integrate Graph and Table data processing systems.

Achieve performance parity with specialized systems.

View a Property Graph as a Table

Property Graph



Vertex Property Table

Id	Property (V)
Rxin	(Stu., Berk.)
Jegonzal	(PstDoc, Berk.)
Franklin	(Prof., Berk)
Istoica	(Prof., Berk)

Edge Property Table

SrcId	DstId	Property (E)
rxin	jegonzal	Friend
franklin	rxin	Advisor
istoica	franklin	Coworker
franklin	jegonzal	PI

Property Graphs as Distributed Tables

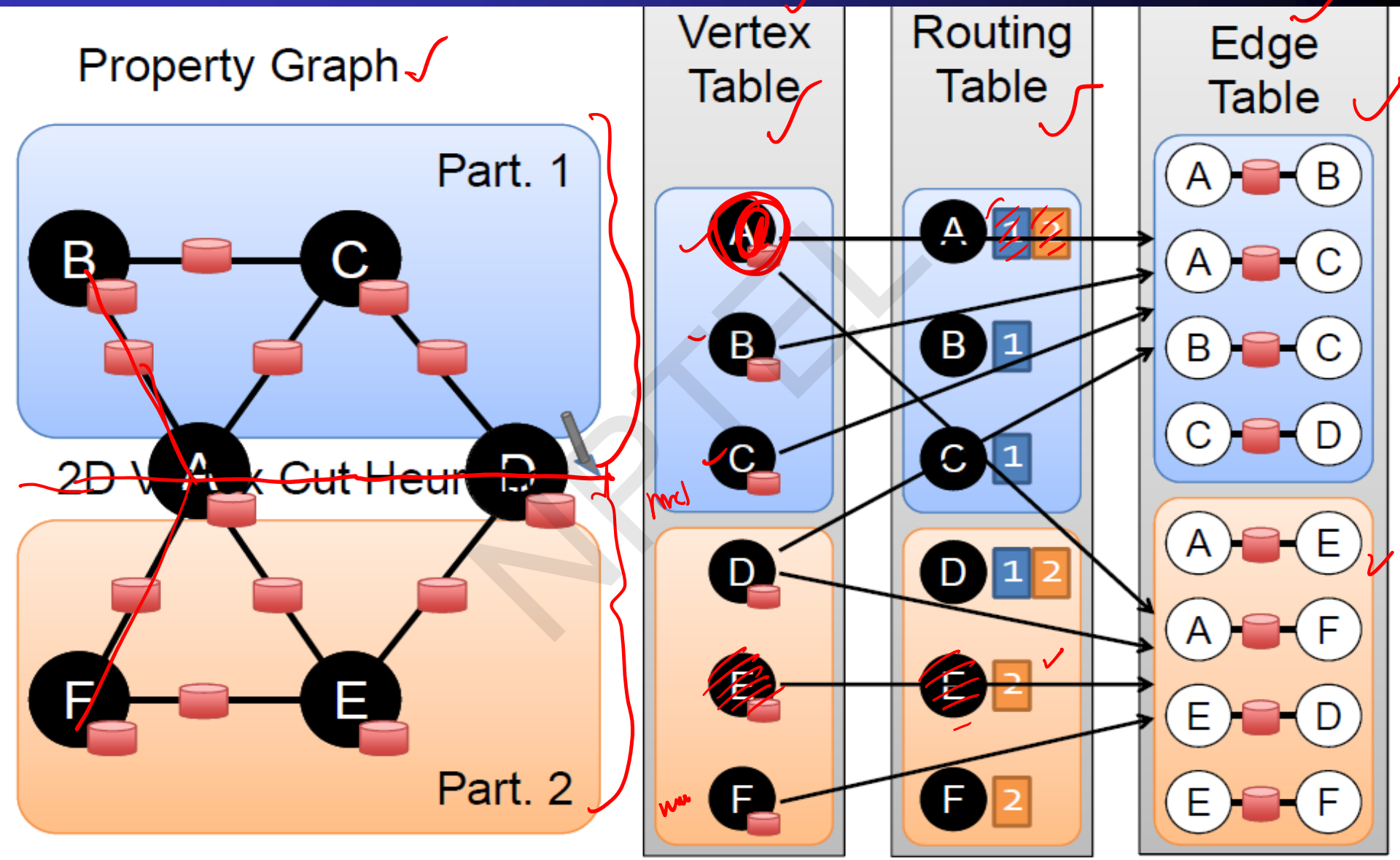


Table Operators

- Table (RDD) operators are inherited from Spark:

map	reduce	sample
filter	count	take
groupBy	fold	first
sort	reduceByKey	partitionBy
union	groupByKey	mapWith
join	cogroup	pipe
leftOuterJoin	cross	save
rightOuterJoin	zip	…

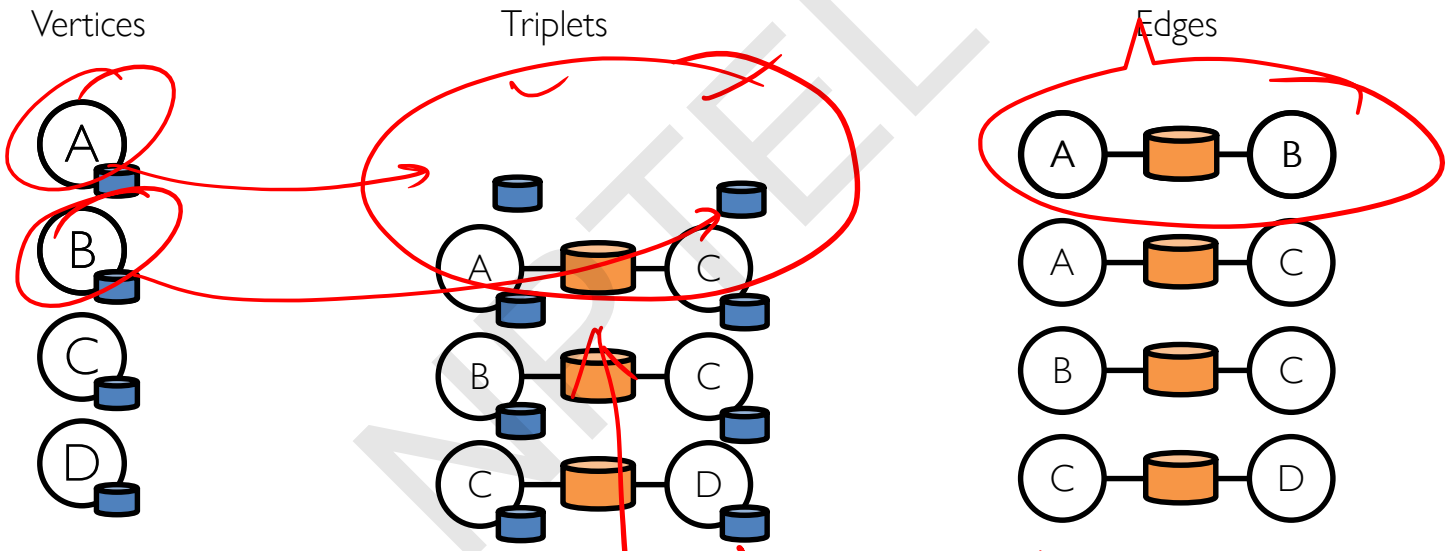
Graph Operators

```
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) ]
  def triplets: Table [ ((Id, V), (Id, V), 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]
}
```

(Vertex property with Edge property)

Triplets Join Vertices and Edges

- The ***triplets*** operator joins vertices and edges:

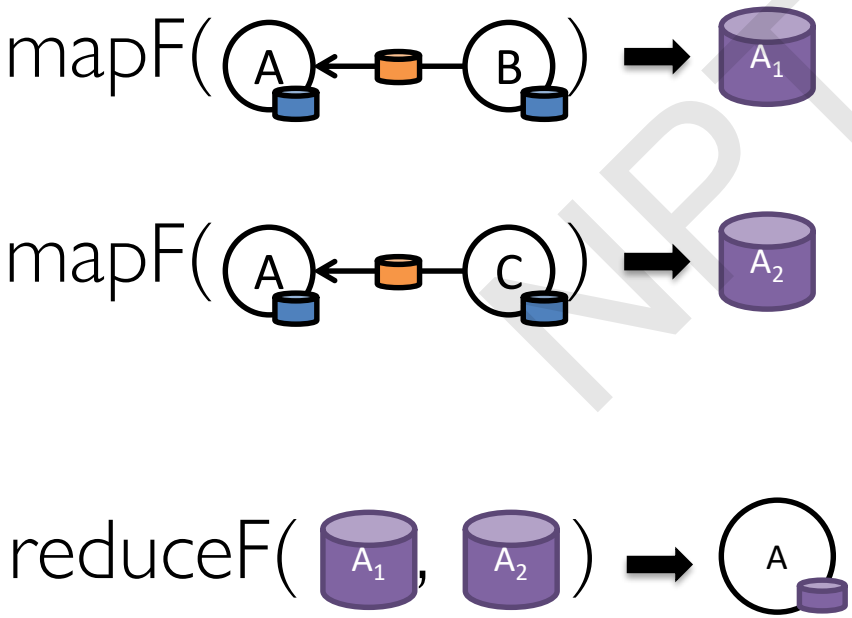


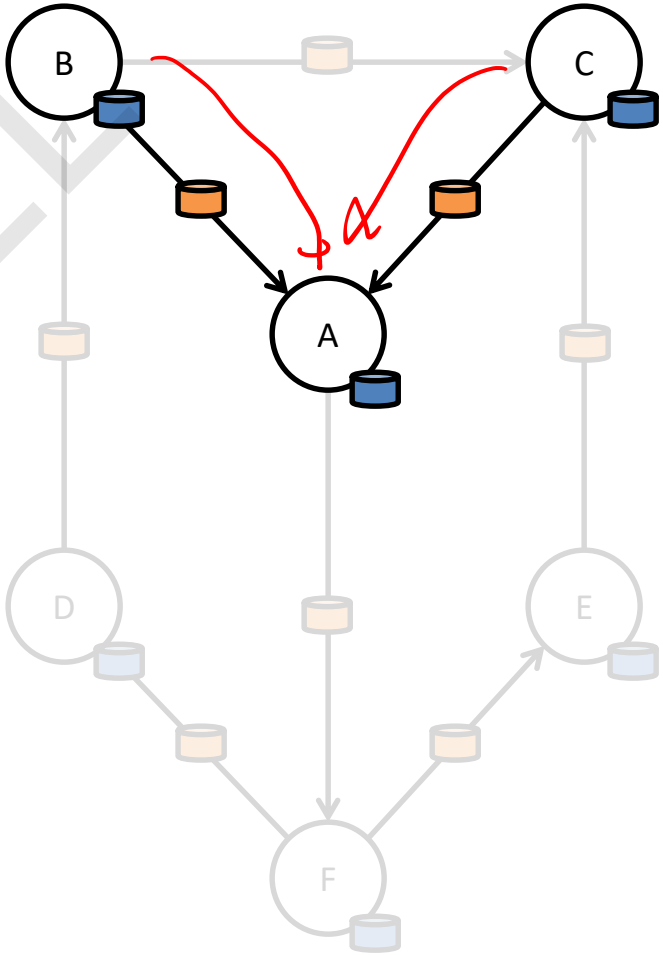
The *mrTriplets* operator sums adjacent triplets.

SELECT t.dstId, *reduceUDF*(*mapUDF*(t)) **AS** sum
FROM triplets **AS** t **GROUPBY** t.dstId

Map Reduce Triplets

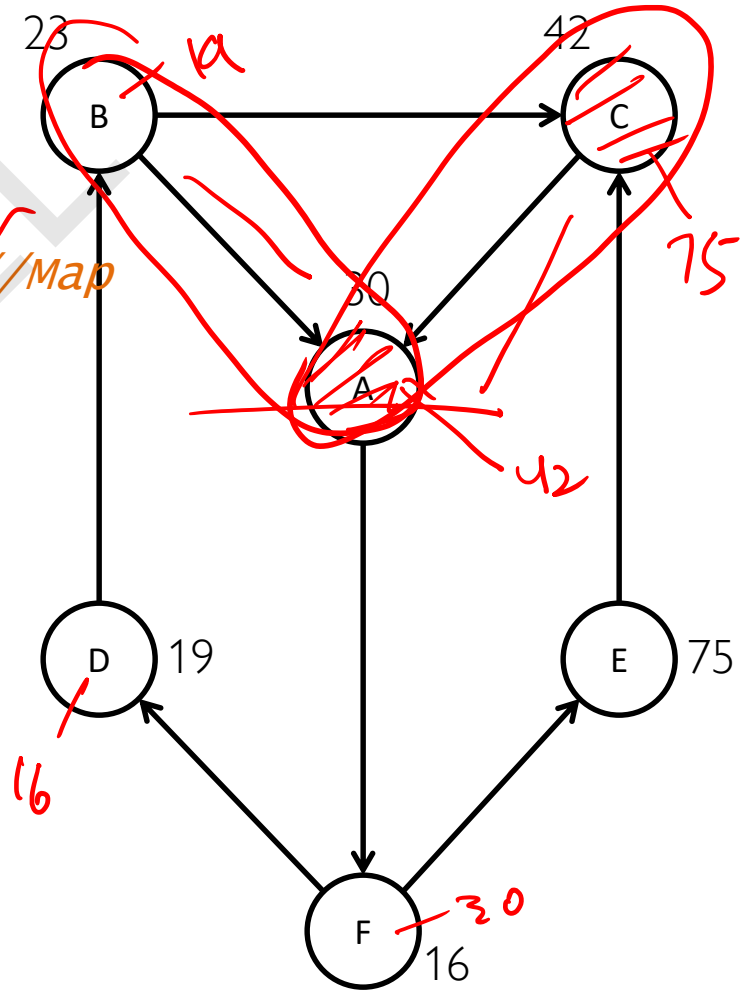
- *Map-Reduce* for each vertex





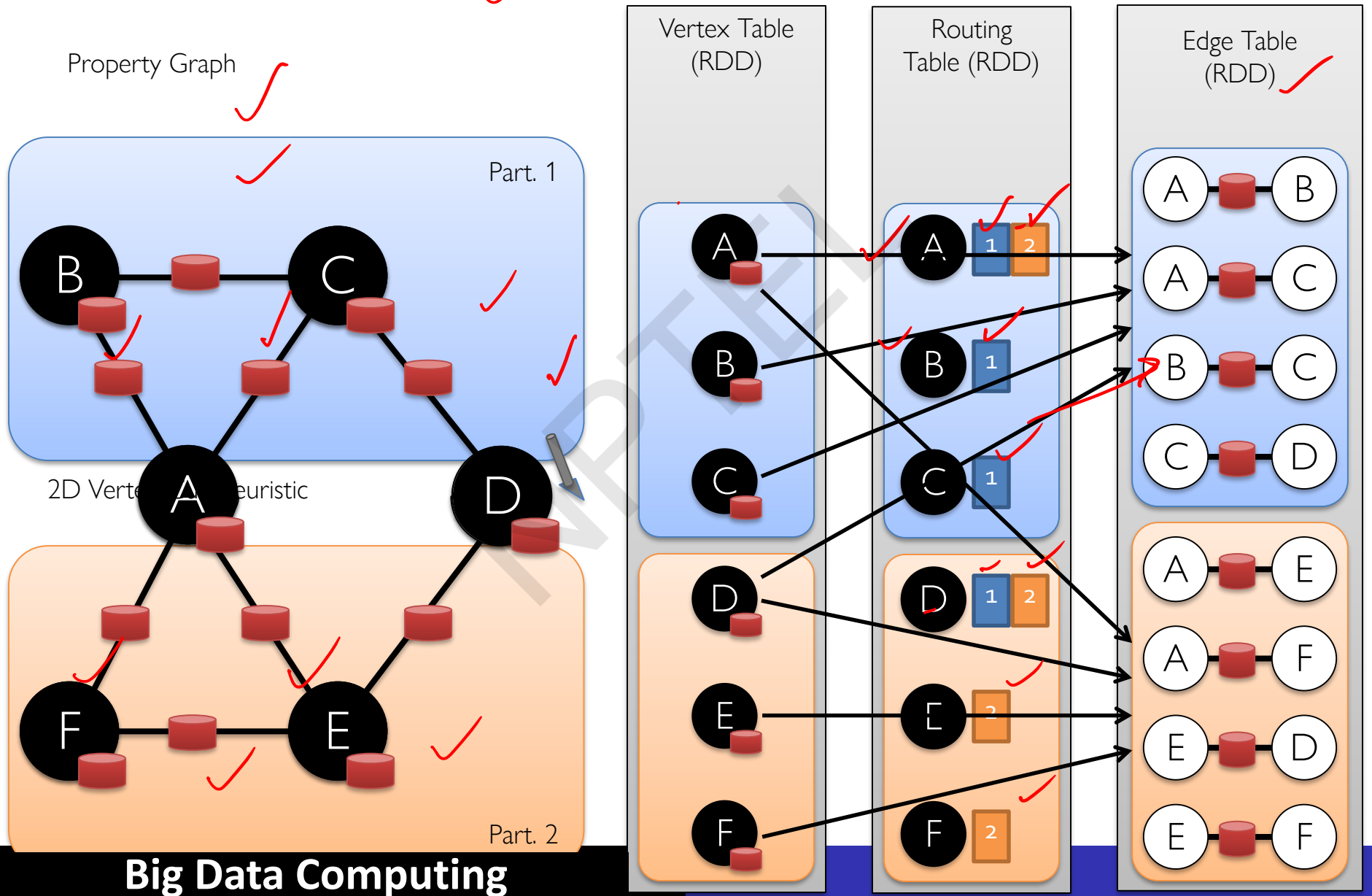
Example: Oldest Follower

- *What is the age of the oldest follower for each user?*
- ```
  val oldestFollowerAge = graph
    .mrTriplets(
      e=> (e.dst.id, e.src.age), //Map
      (a,b)=> max(a, b) //Reduce
    )
    .vertices
  ```

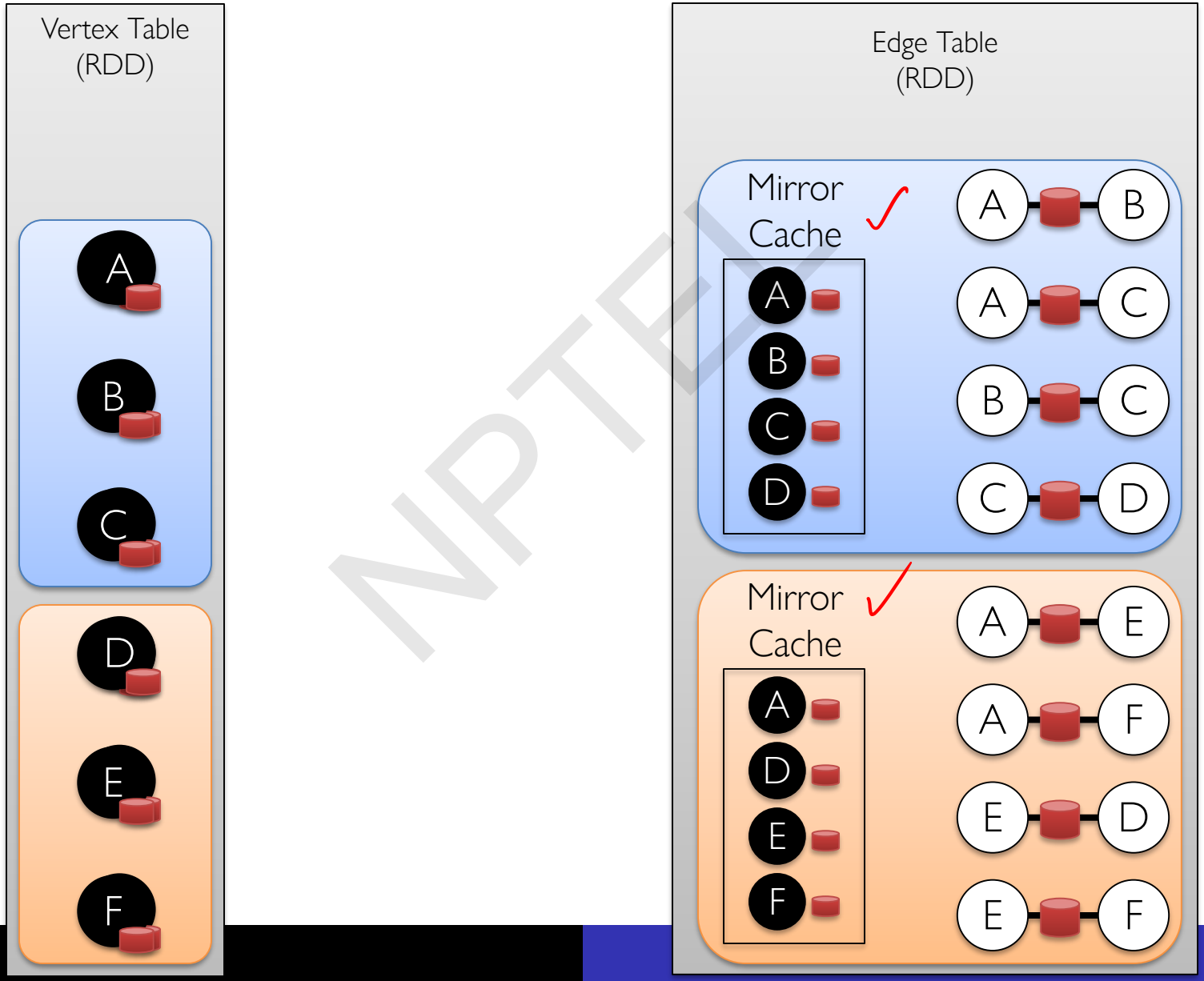

  ```

GraphX System Design

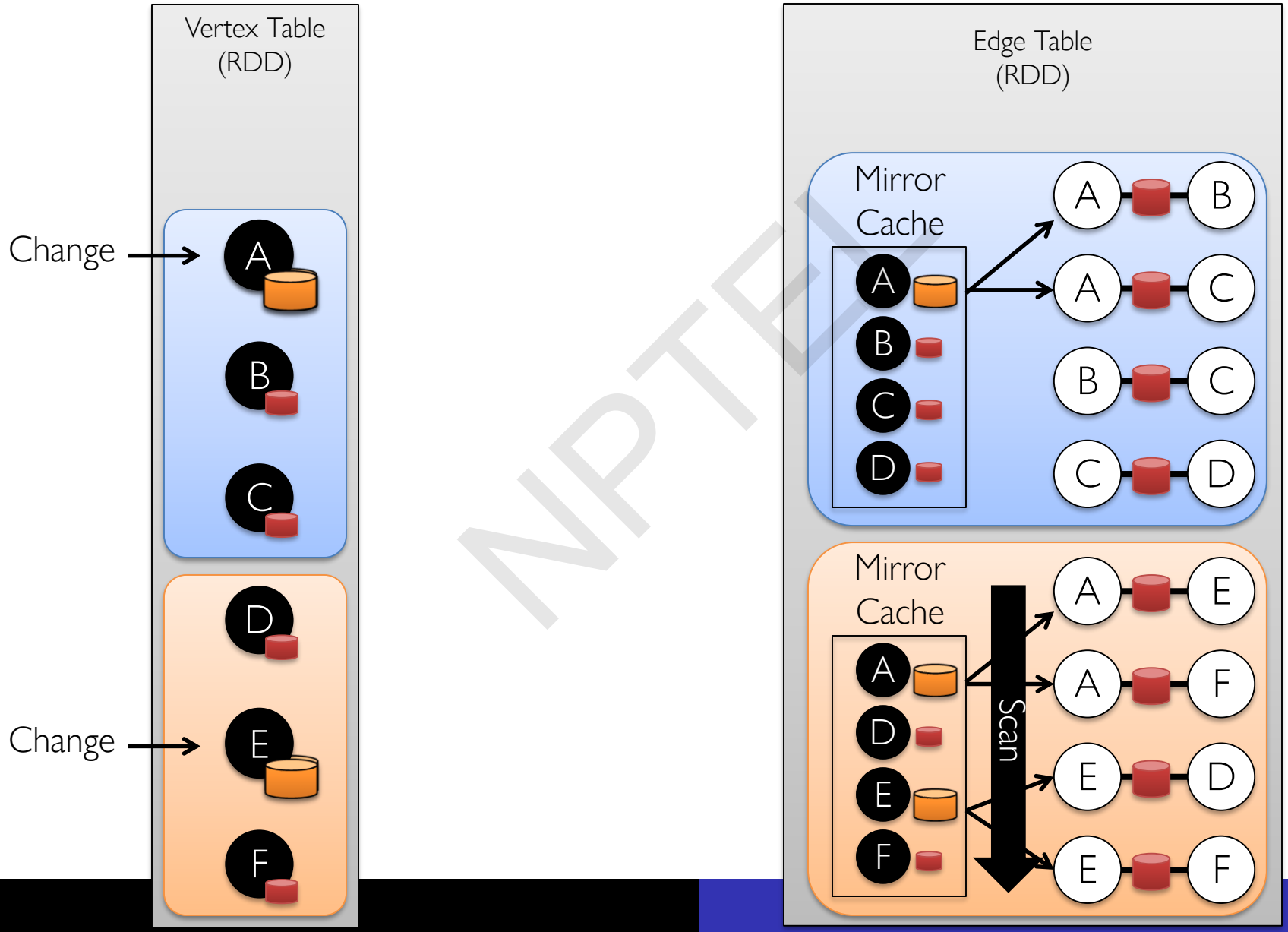
Distributed Graphs as Tables (RDDs)



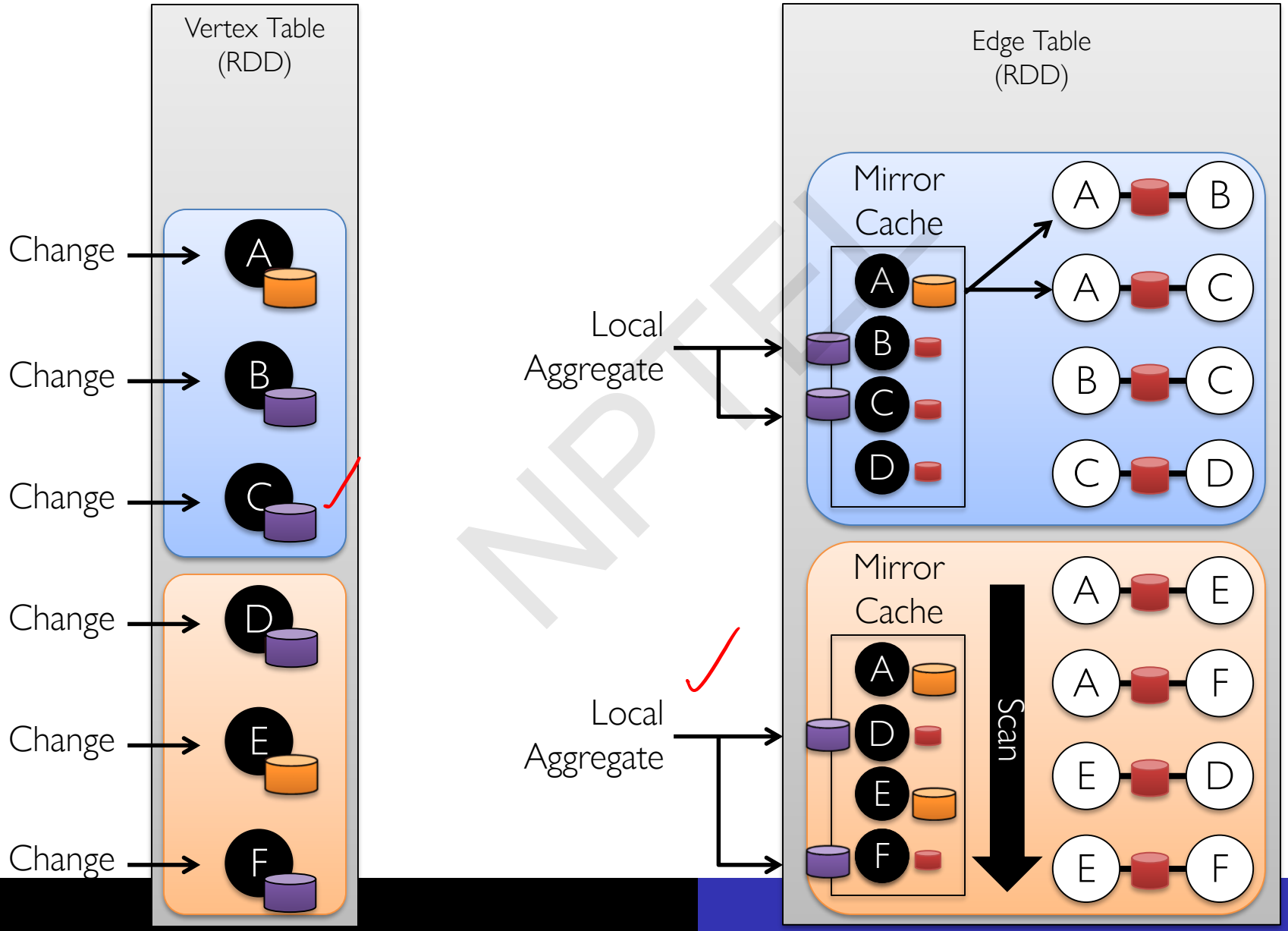
Caching for Iterative mrTriplets



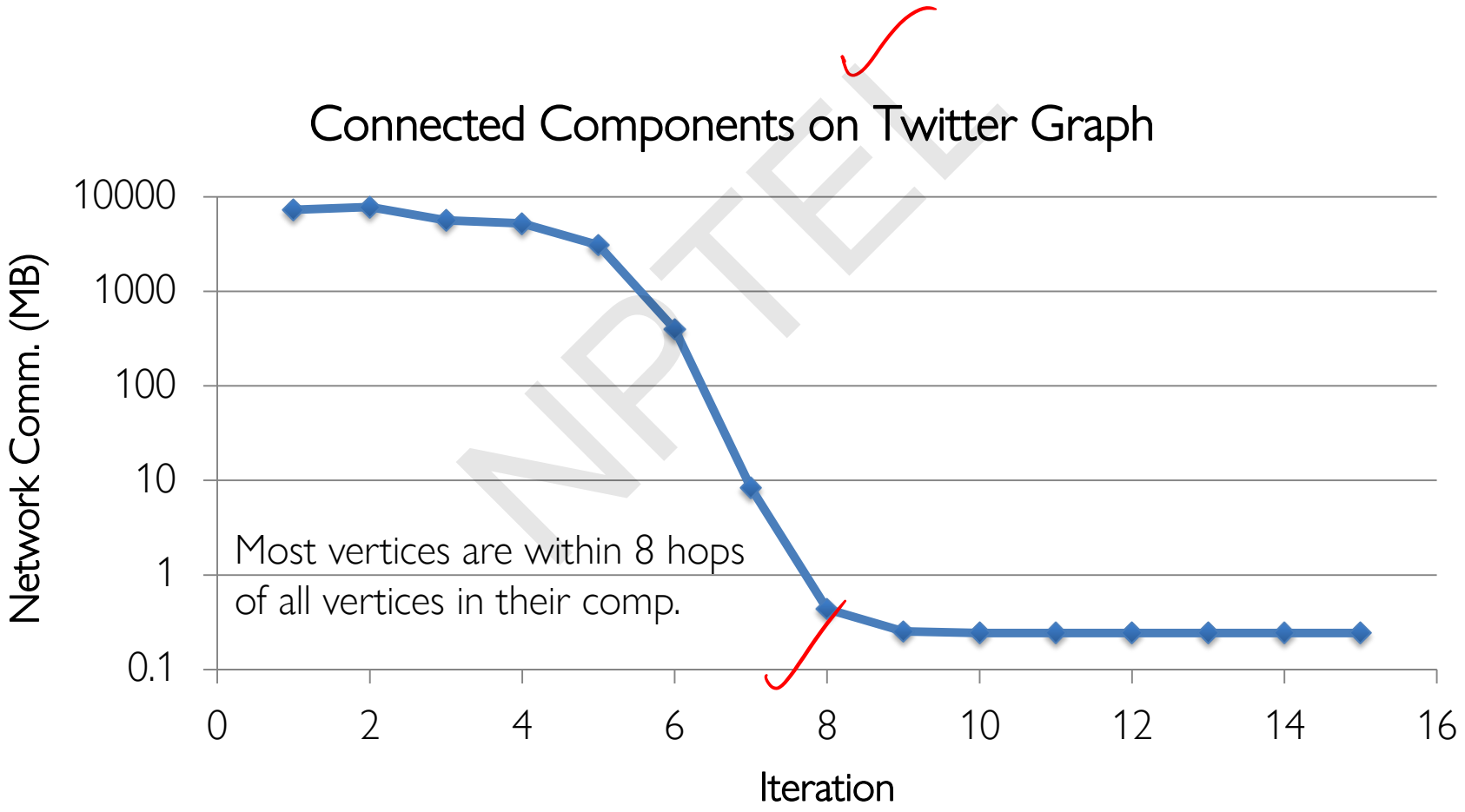
Incremental Updates for Iterative mrTriplets



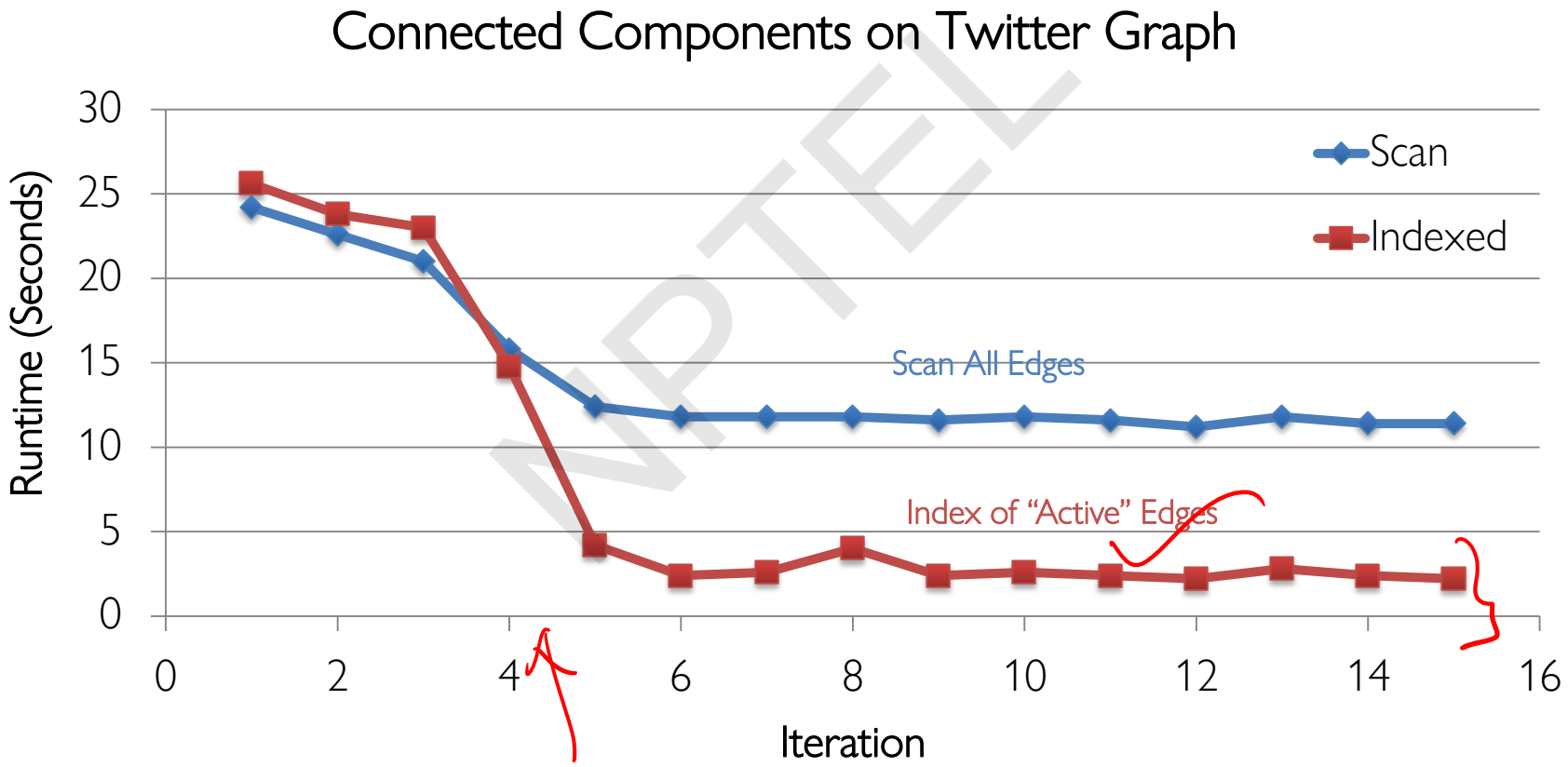
Aggregation for Iterative mrTriplets



Reduction in Communication Due to Cached Updates



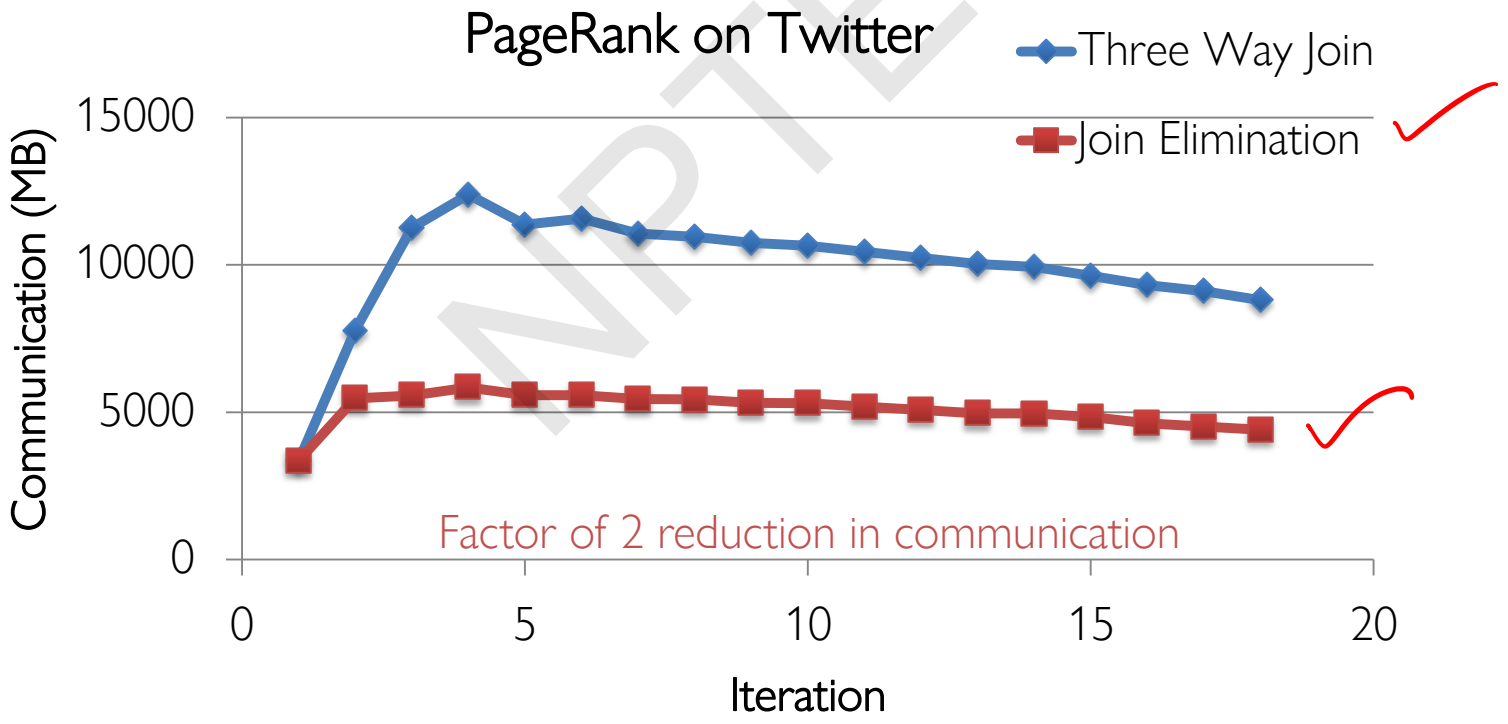
Benefit of Indexing *Active* Edges



Join Elimination

Identify and bypass joins for unused triplets fields

- *Example:* PageRank only accesses source attribute

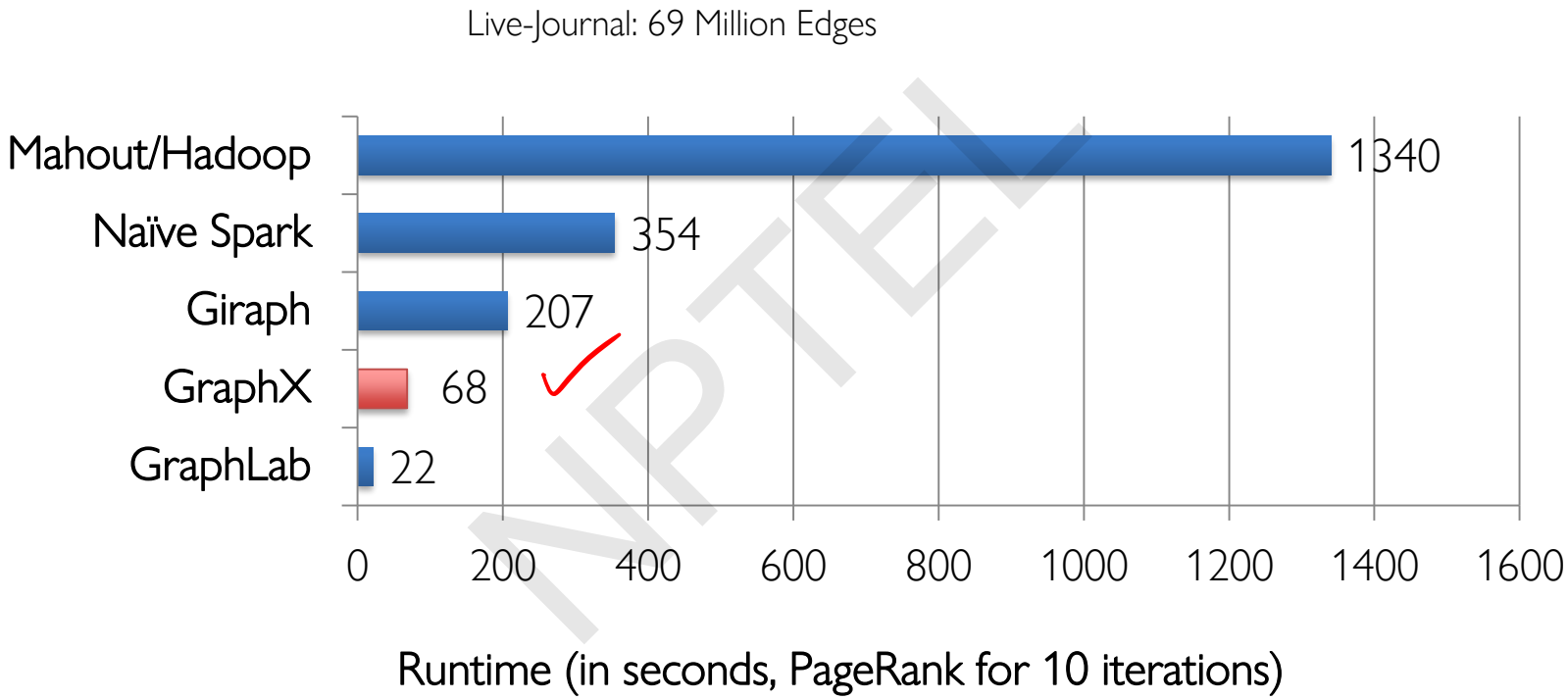


Additional Query Optimizations

- Indexing and Bitmaps:
  - To accelerate joins across graphs
  - To efficiently construct sub-graphs
- Substantial Index and Data Reuse:
  - Reuse routing tables across graphs and sub-graphs
  - Reuse edge adjacency information and indices

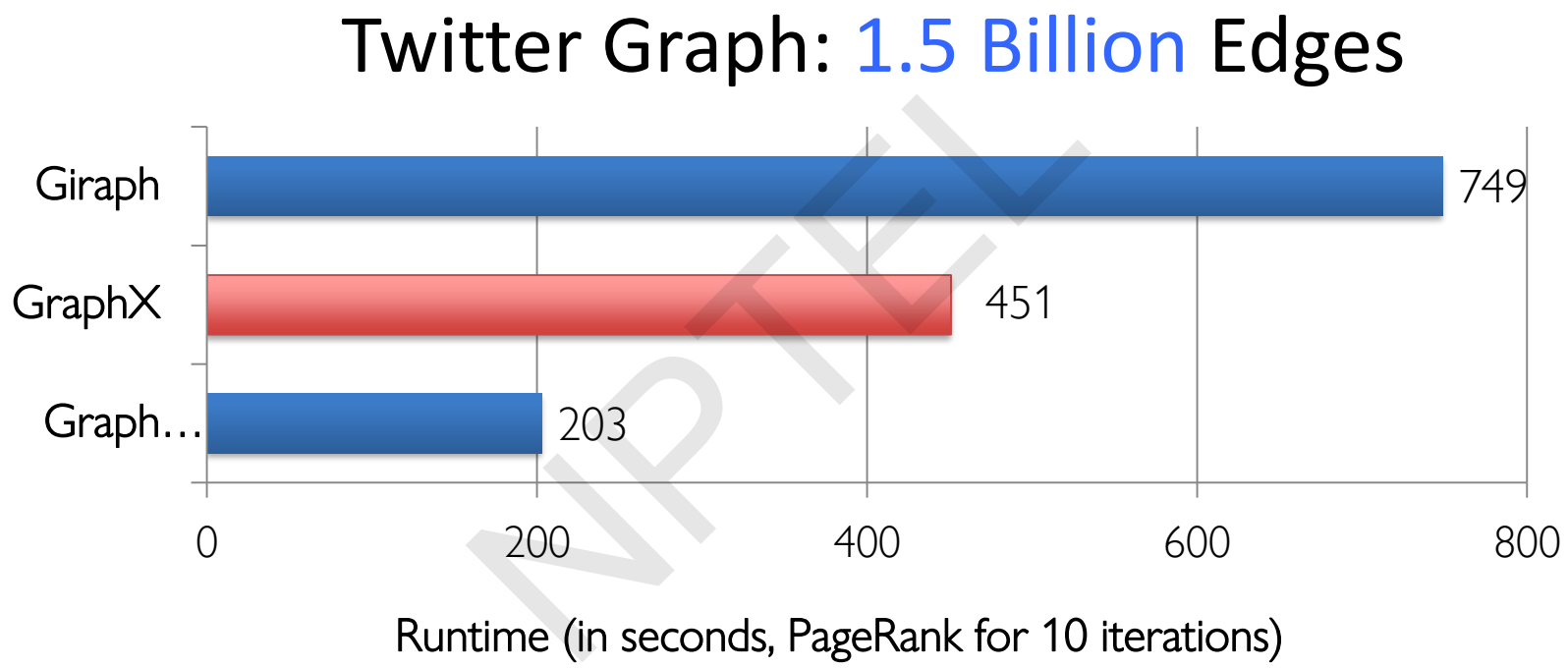
GraphX

Performance Comparisons



GraphX is roughly 3x slower than GraphLab

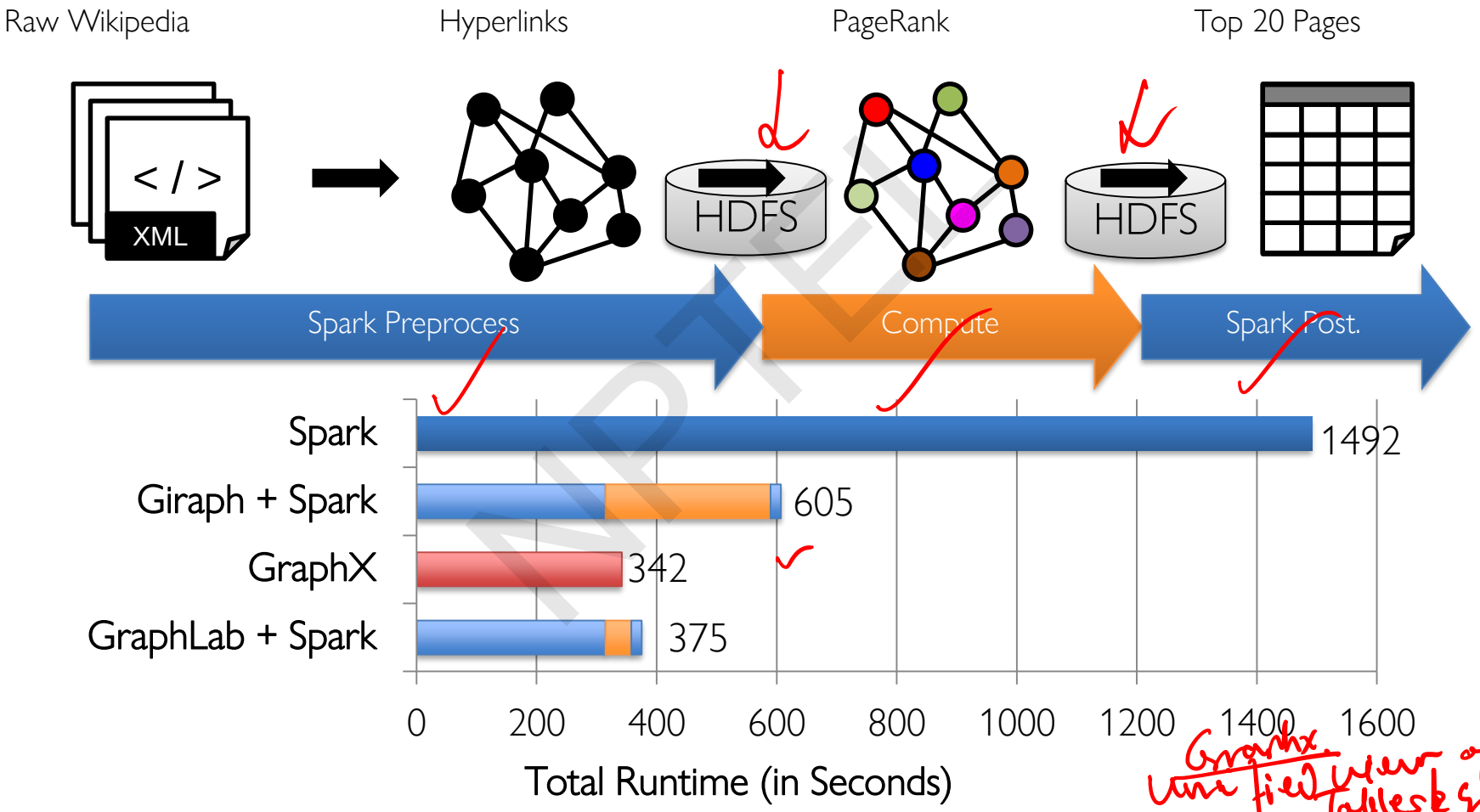
GraphX scales to larger graphs



GraphX is roughly 2x slower than GraphLab

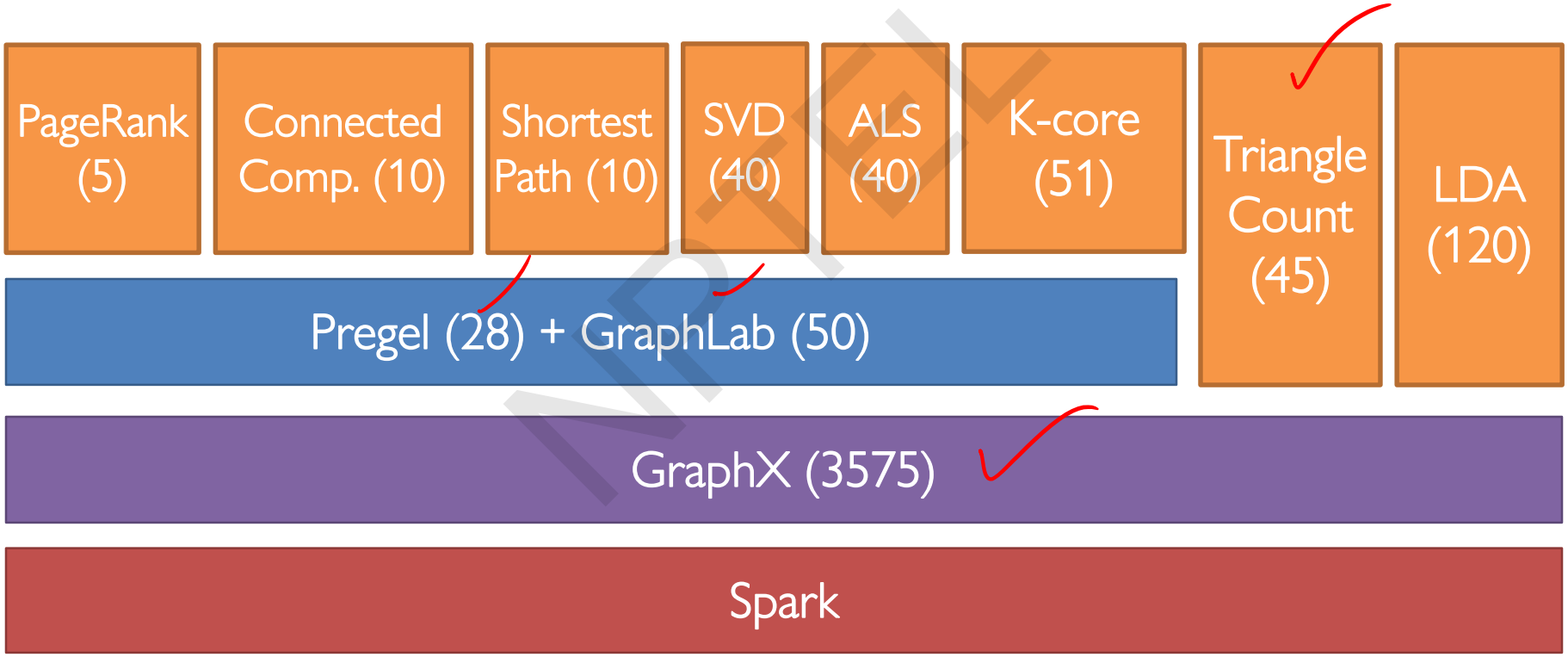
- » Scala + Java overhead: Lambdas, GC time, …
- » No shared memory parallelism: 2x increase in comm.

A Small Pipeline in GraphX



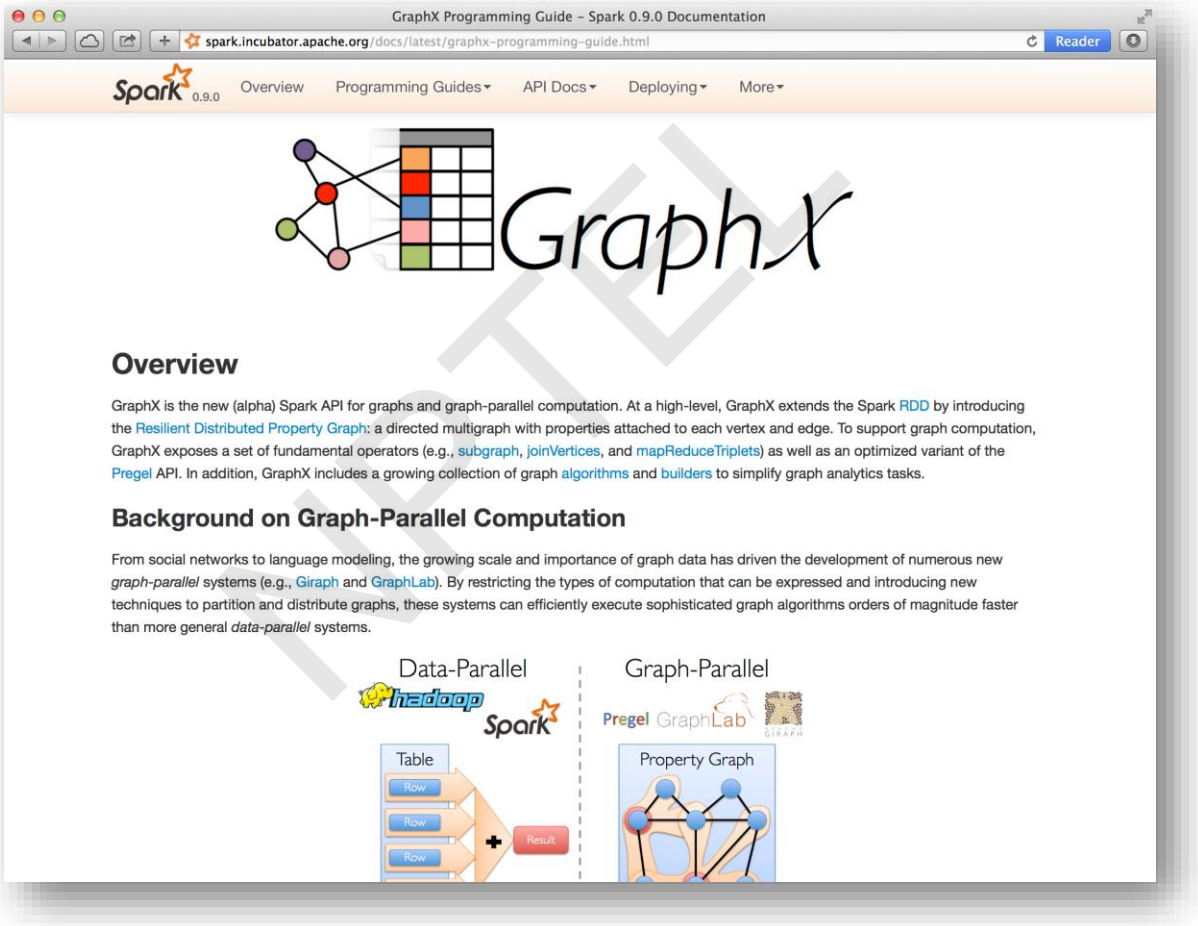
Timed end-to-end GraphX is *faster* than GraphLab

The GraphX Stack (Lines of Code)



Status

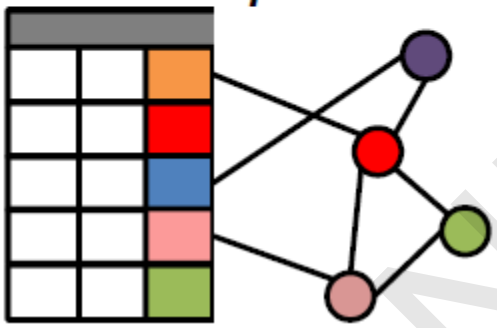
- Alpha release as part of Spark 0.9



GraphX: Unified Analytics

New API

Blurs the distinction between Tables and Graphs



New System

Combines Data-Parallel Graph-Parallel Systems

- Enabling users to easily and efficiently express the entire graph analytics pipeline

A Case for Algebra in Graphs

- A standard algebra is essential for graph systems:
- e.g.: SQL → proliferation of relational system

- By embedding graphs in *relational algebra:*

- Integration with tables and preprocessing
- Leverage advances in relational systems
- Graph opt. recast to relational systems

Observations

- Domain specific views: *Tables* and *Graphs*
  - tables and graphs are first-class composable objects
  - specialized operators which exploit view semantics

- Single system that efficiently spans the pipeline
  - minimize data movement and duplication
  - eliminates need to learn and manage multiple systems

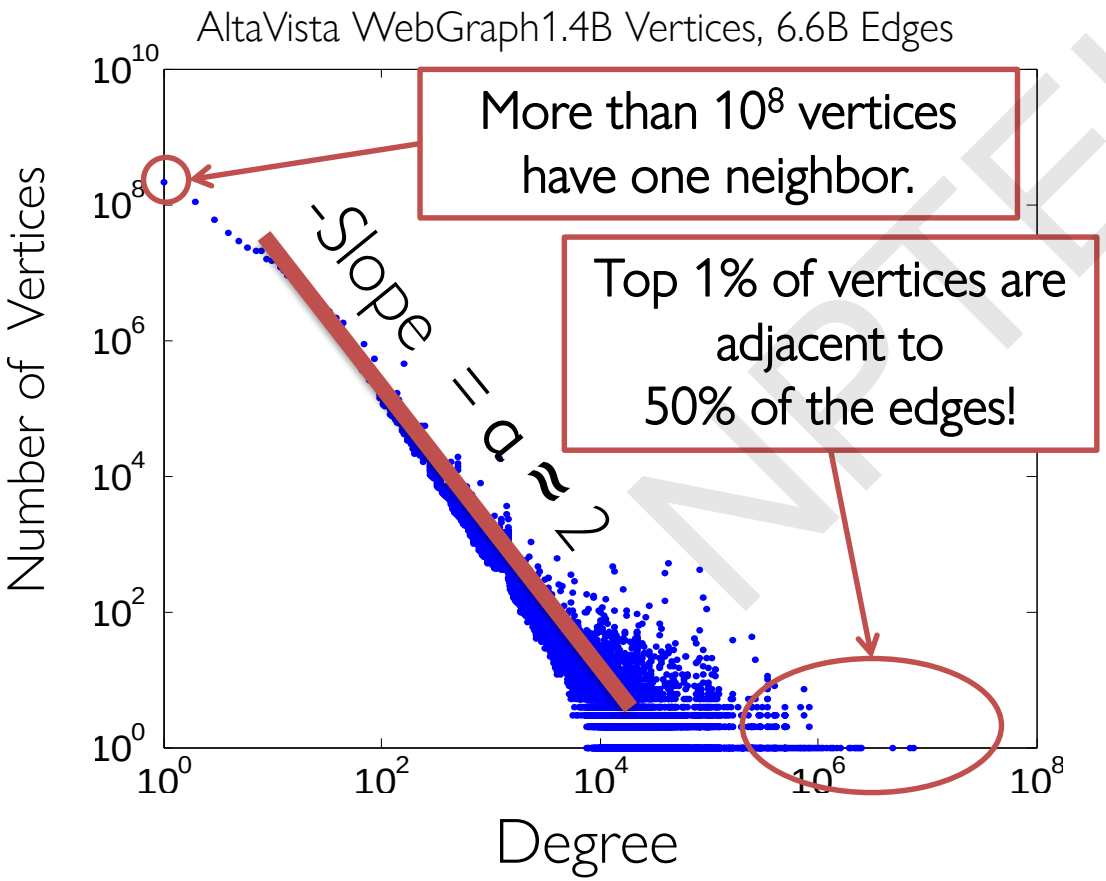
- Graphs through the lens of database systems
  - Graph-Parallel Pattern → Triplet joins in relational alg.
  - Graph Systems → Distributed join optimizations

Active Research

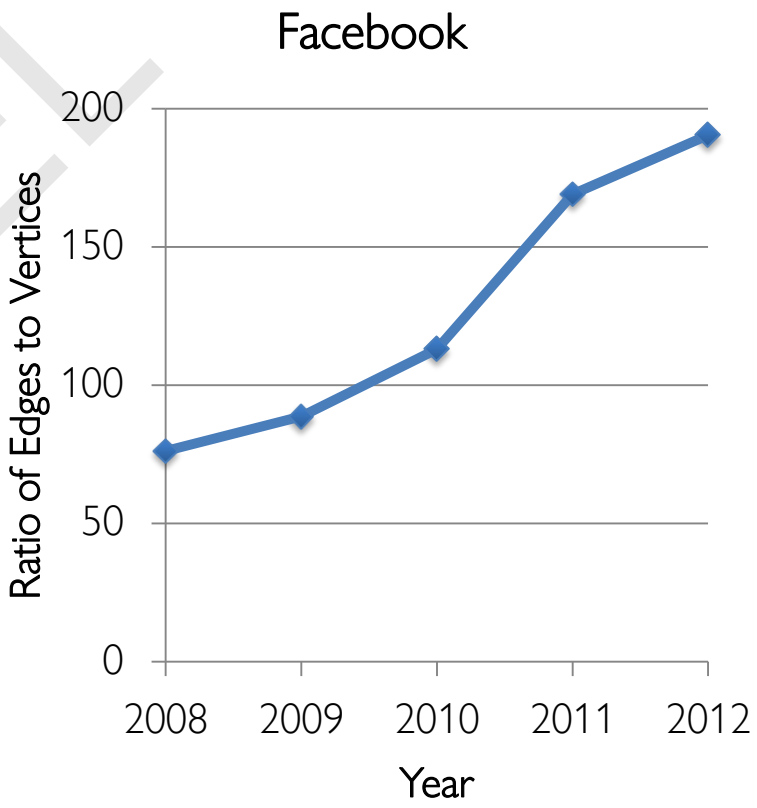
- Static Data → Dynamic Data
  - Apply GraphX unified approach to time evolving data
  - Model and analyze relationships over time
- Serving Graph Structured Data
  - Allow external systems to interact with GraphX
  - Unify distributed graph databases with relational database technology

Graph Property 1 Real-World Graphs

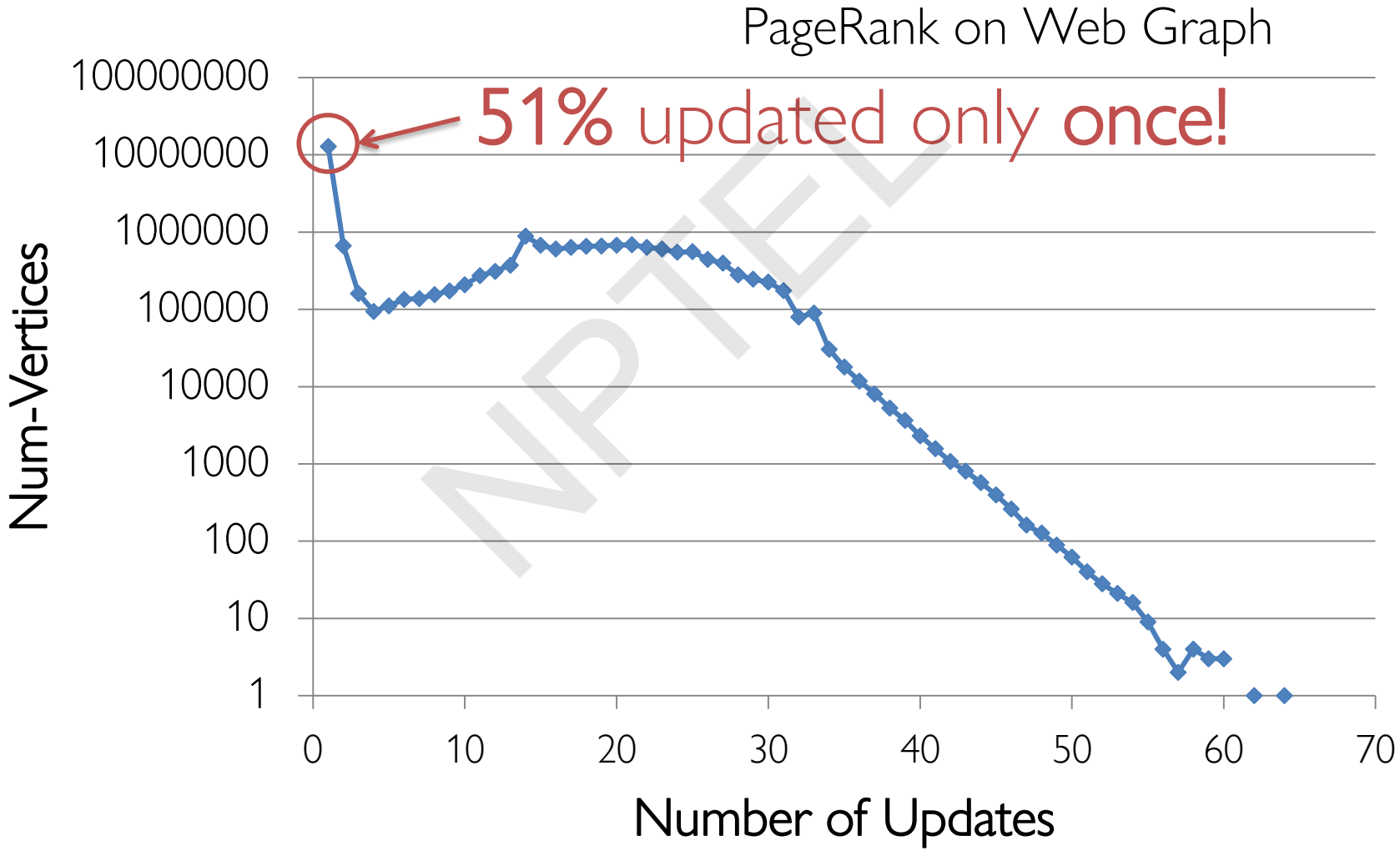
Power-Law Degree Distribution



Edges >> Vertices



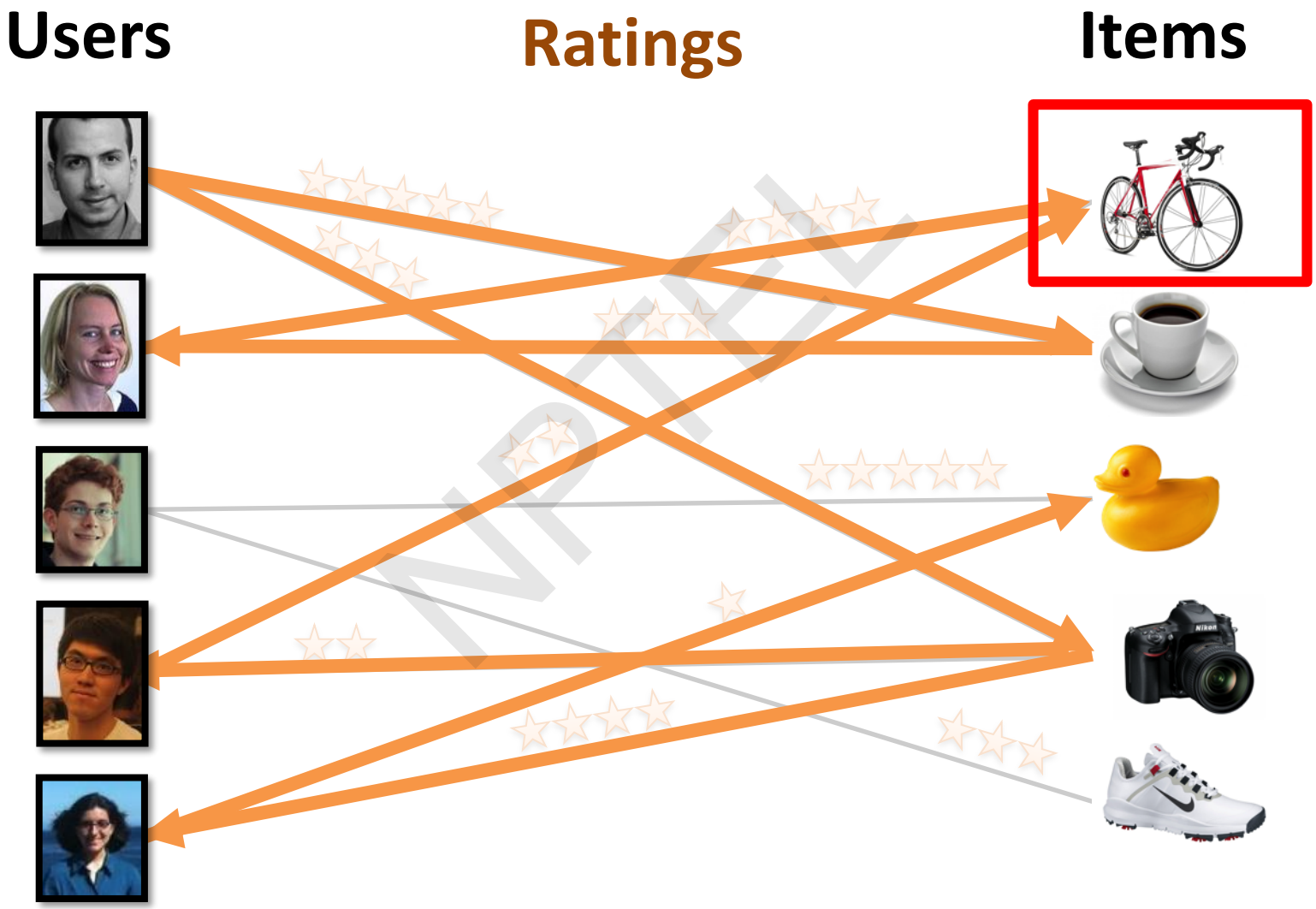
Graph Property 2 Active Vertices



Graphs are Essential to Data Mining and Machine Learning

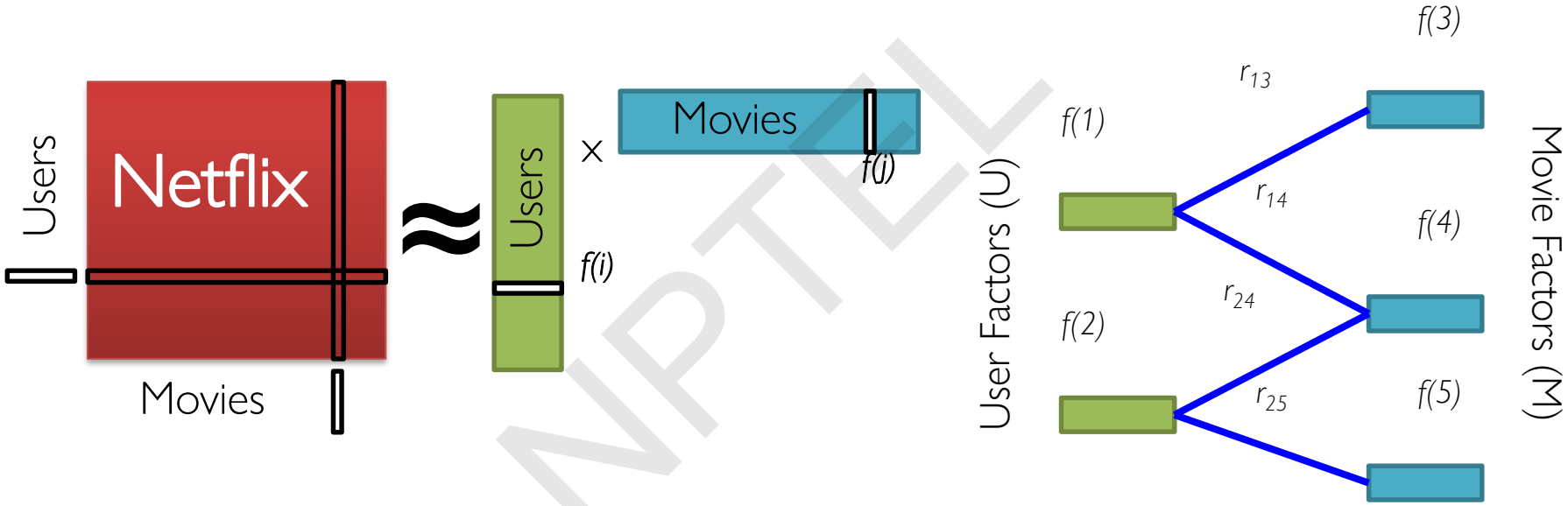
- Identify influential people and information
- Find communities
- Understand people's shared interests
- Model complex data dependencies

Recommending Products



Recommending Products

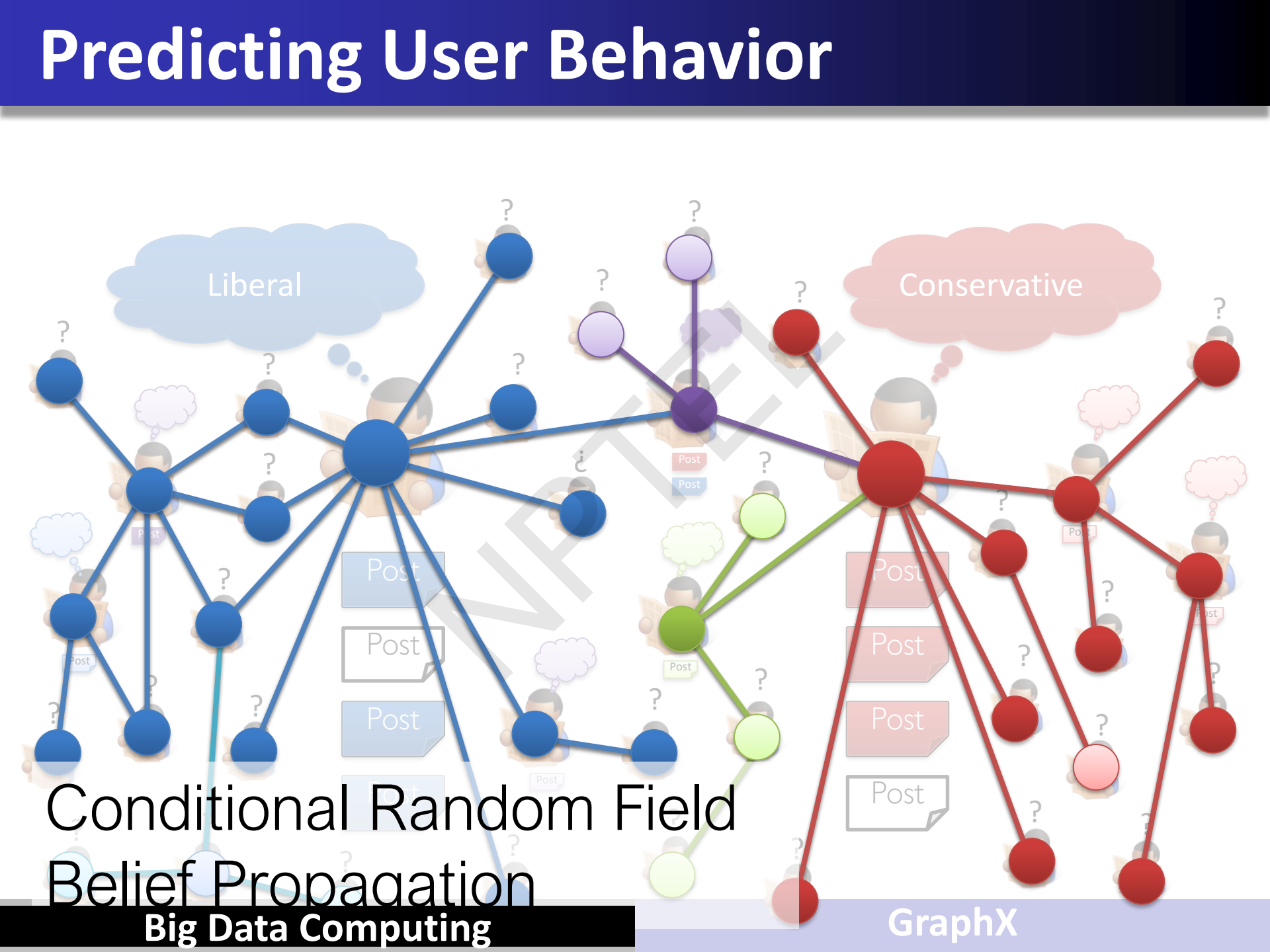
Low-Rank Matrix Factorization:



Iterate:

$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda ||w||_2^2$$

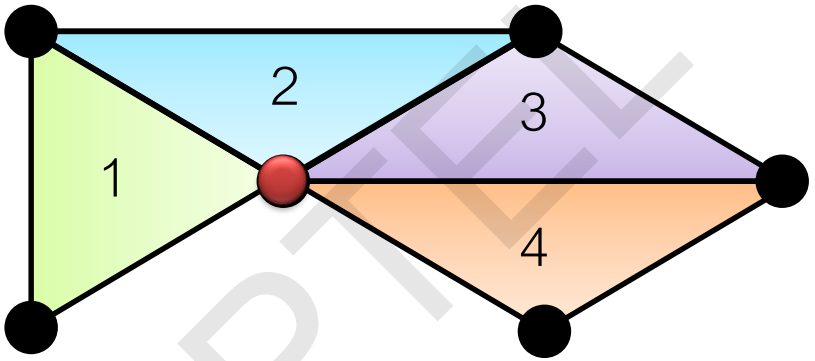
Predicting User Behavior



Conditional Random Field Belief Propagation

Finding Communities

- Count triangles passing through each vertex:

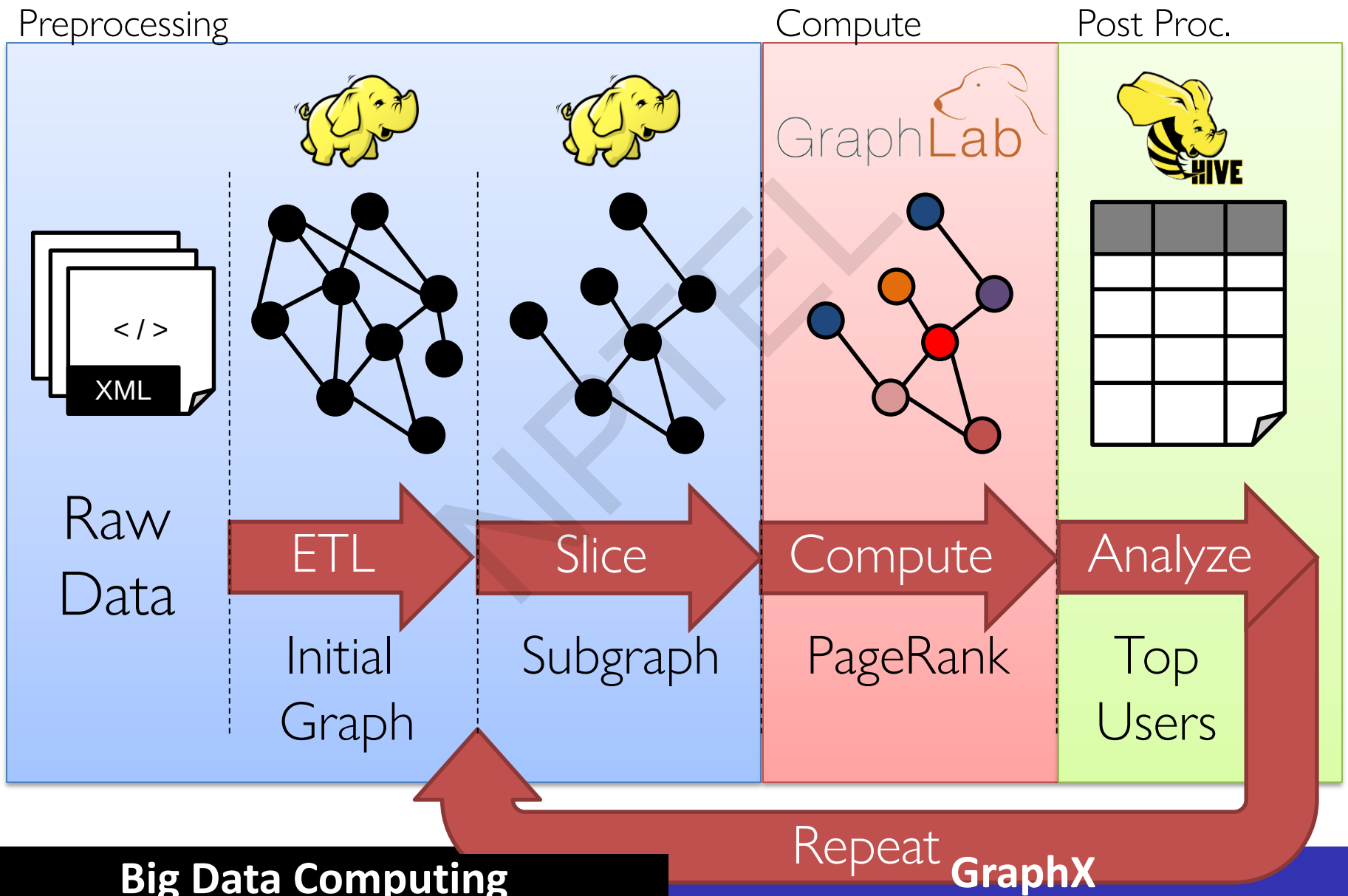


- Measures "cohesiveness" of local community

Fewer Triangles
Weaker Community

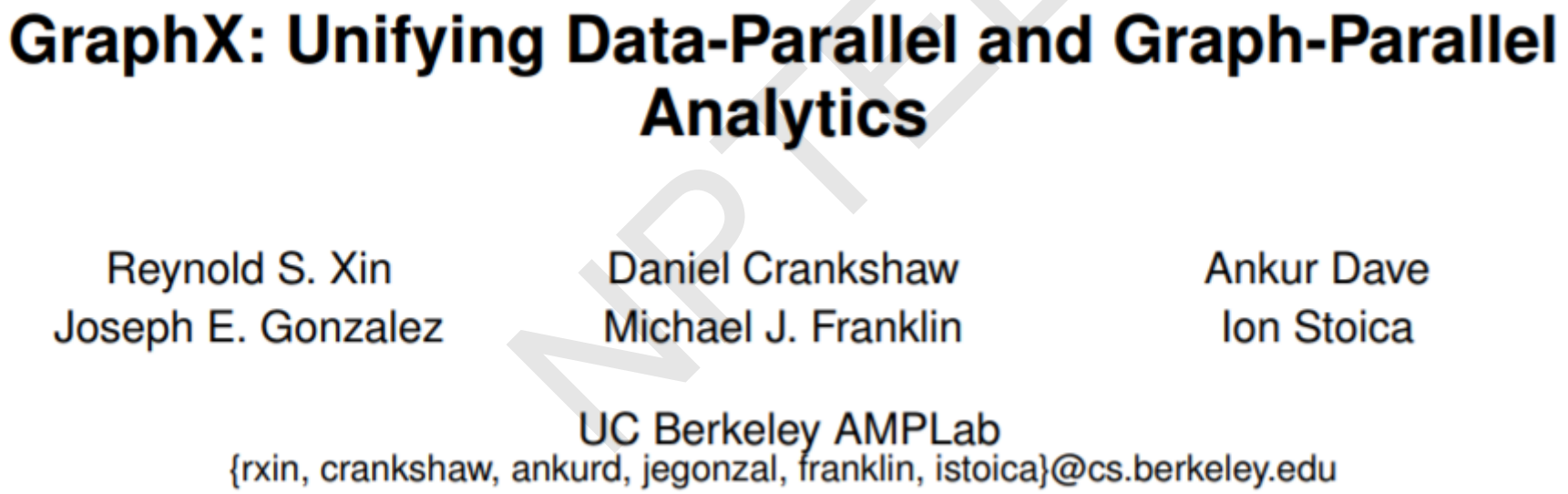
More Triangles
Stronger Community

Example Graph Analytics Pipeline



References

- Xin, R., Crankshaw, D., Dave, A., Gonzalez, J., Franklin, M.J., & Stoica, I. (2014). GraphX: Unifying Data-Parallel and Graph-Parallel Analytics. *CoRR, abs/1402.2394*.



GraphX: Unifying Data-Parallel and Graph-Parallel Analytics

Reynold S. Xin, Daniel Crankshaw, Ankur Dave
Joseph E. Gonzalez, Michael J. Franklin, Ion Stoica

UC Berkeley AMPLab
{rxin, crankshaw, ankurd, jegonzal, franklin, istoica}@cs.berkeley.edu

- http://spark.apache.org/graphx

Conclusion

- The growing scale and importance of graph data has driven the development of specialized graph computation engines capable of inferring complex recursive properties of graph-structured data.

- In this lecture we have discussed GraphX, a distributed graph processing framework that unifies graph-parallel and data-parallel computation in a single system and is capable of succinctly expressing and efficiently executing the entire graph analytics pipeline.

Case Study: Flight Data Analysis using Spark GraphX

Problem Statement

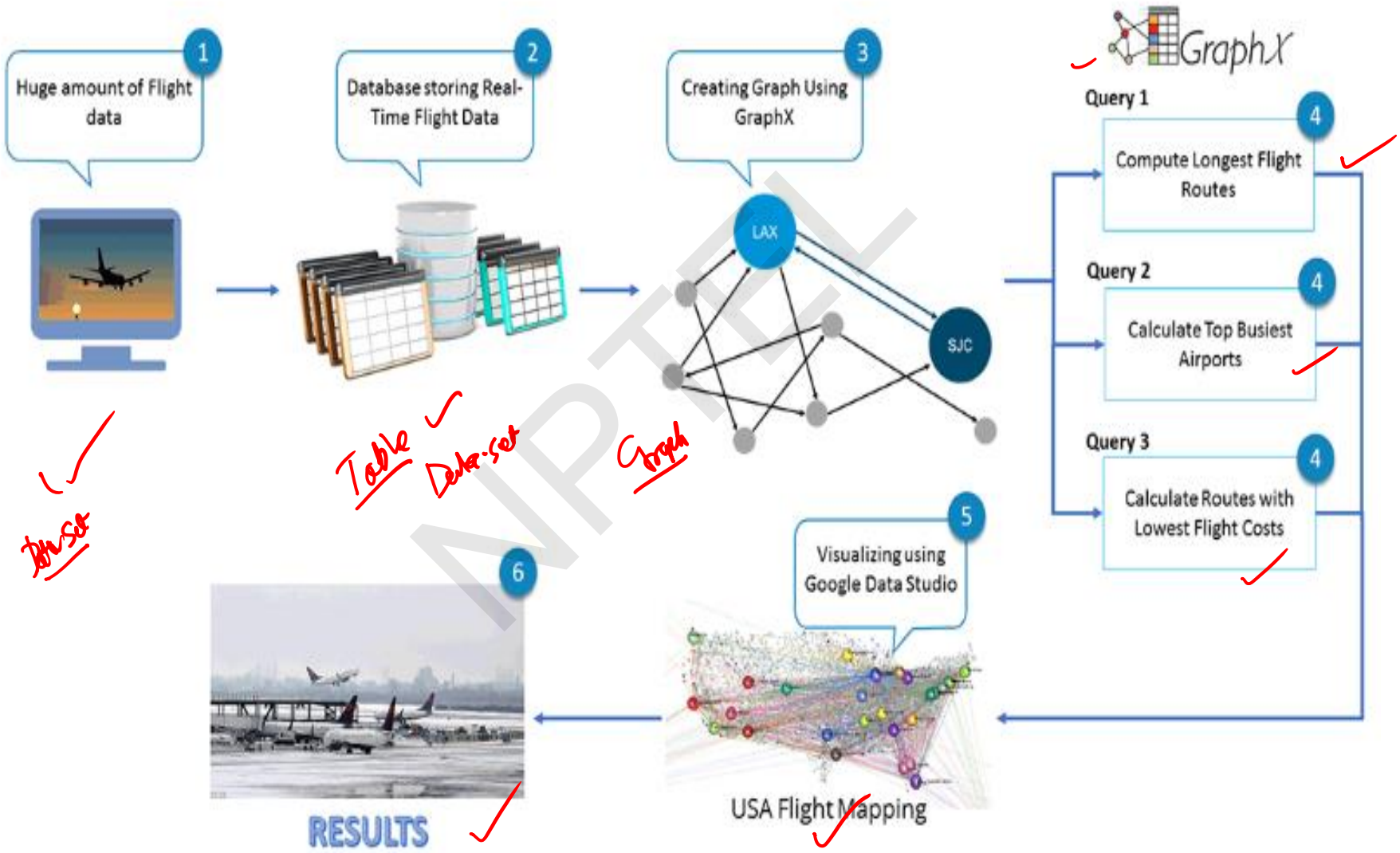
- To analyze Real-Time Flight data using Spark GraphX, provide near real-time computation results and visualize the results.

Flight Data Analysis using Spark GraphX

Dataset Description:

The data has been collected from U.S. Department of Transportation's (DOT) Bureau of Transportation Statistics site which tracks the on-time performance of domestic flights operated by large air carriers. The Summary information on the number of on-time, delayed, canceled, and diverted flights is published in DOT's monthly Air Travel Consumer Report and in this dataset of January 2014 flight delays and cancellations.

Big Data Pipeline for Flight Data Analysis using Spark GraphX



UseCases

I. Monitoring Air traffic at airports
II. Monitoring of flight delays
III. Analysis overall routes and airports
IV. Analysis of routes and airports per airline

Objectives

- Compute the total number of airports
- Compute the total number of flight routes
- Compute and sort the longest flight routes
- Display the airports with the highest incoming flights
- Display the airports with the highest outgoing flights
- List the most important airports according to PageRank
- List the routes with the lowest flight costs
- Display the airport codes with the lowest flight costs
- List the routes with airport codes which have the lowest flight costs

Features

- 17 Attributes

Attribute Name	Attribute Description
dOfM	Day of Month
dOfW	Day of Week
carrier	Unique Airline Carrier Code
tailNum	Tail Number
fNum	Flight Number Reporting Airline
origin_id	Origin Airport Id
origin	Origin Airport
dest_id	Destination Airport Id
dest	Destination Airport
crsdepttime	CRS Departure Time (local time: hhmm)
deptime	Actual Departure Time

Features

Attribute Name	Attribute Description
depdelaymins	Difference in minutes between scheduled and actual departure time. Early departures set to 0.
crsarrtime	CRS Arrival Time (local time: hhmm)
arrtime	Actual Arrival Time (local time: hhmm)
arrdelaymins	Difference in minutes between scheduled and actual arrival time. Early arrivals set to 0.
crselapsedtime	CRS Elapsed Time of Flight, in Minutes
dist	Distance between airports (miles)

Sample Dataset

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	dOfM	dOfW	carrier	tailNum	flNum	origin_id	origin	dest_id	dest	crsdeptime	deptime	depdelaymins	crsarrtime	arrtime	arrdelaymins	crselapsedtime	dist
2	1	3	AA	N338AA	1	12478	JFK	12892	LAX	900	914	14	1225	1238	13	385	2475
3	2	4	AA	N338AA	1	12478	JFK	12892	LAX	900	857	0	1225	1226	1	385	2475
4	4	6	AA	N327AA	1	12478	JFK	12892	LAX	900	1005	65	1225	1324	59	385	2475
5	5	7	AA	N323AA	1	12478	JFK	12892	LAX	900	1050	110	1225	1415	110	385	2475
6	6	1	AA	N319AA	1	12478	JFK	12892	LAX	900	917	17	1225	1217	0	385	2475
7	7	2	AA	N328AA	1	12478	JFK	12892	LAX	900	910	10	1225	1212	0	385	2475
8	8	3	AA	N323AA	1	12478	JFK	12892	LAX	900	923	23	1225	1215	0	385	2475
9	9	4	AA	N339AA	1	12478	JFK	12892	LAX	900	859	0	1225	1204	0	385	2475
10	10	5	AA	N319AA	1	12478	JFK	12892	LAX	900	929	29	1225	1245	20	385	2475

Spark Implementation

```
1  //Importing the necessary classes
2  import org.apache.spark._
3  ...
4  import java.io.File
5
6  object airport {
7
8   def main(args: Array[String]){
9
10 //Creating a Case Class Flight
11 case class Flight(dofM:String, dofW:String, ... ,dist:Int)
12
13 //Defining a Parse String function to parse input into Flight class
14 def parseFlight(str: String): Flight = {
15 val line = str.split(",")
16 Flight(line(0), line(1), ... , line(16).toInt)
17 }
18 val conf = new SparkConf().setAppName("airport").setMaster("local[2]")
19 val sc = new SparkContext(conf)
```

Spark Implementation

```
20  //Load the data into a RDD
21
22  val textRDD = sc.textFile ("/home/iitp/spark-2.2.0-bin-hadoop-2.6/flights/airportdataset.csv")
23
24  //Parse the RDD of CSV lines into an RDD of flight classes
25  val flightsRDD = Map ParseFlight to Text RDD
26
27  //Create airports RDD with ID and Name
28  val airports = Map Flight OriginID and Origin
29  airports.take(1)
30
31  //Defining a default vertex called nowhere and mapping Airport ID for printlns
32  val nowhere = "nowhere"
33  val airportMap = Use Map Function .collect.toList.toMap
34
35  //Create routes RDD with sourceID, destinationID and distance
36  val routes = flightsRDD. Use Map Function .distinct
37  routes.take(2)
38
39  //Create edges RDD with sourceID, destinationID and distance
40  val edges = routes.map{( Map OriginID and DestinationID ) => Edge(org_id.toLong,
41  edges.take(1)
42
43  //Define the graph and display some vertices and edges
44  val graph = Graph( Airports, Edges and Nowhere )
45  graph.vertices.take(2)
46  graph.edges.take(2)
```

Graph Operations

```
//Query 1 - Find the total number of airports
val numairports = Vertices Number

//Query 2 - Calculate the total number of routes?
val numroutes = Number Of Edges

//Query 3 - Calculate those routes with distances more than 1000 miles
graph.edges.filter { Get the edge distance )=> distance > 1000}.take(3)
```

Subgraph
- Tombstone

Graph Operations

- How many airports are there?
- How many unique flights from airport A to airport B are there?

```
graph.numVertices // 305
graph.numEdges // 5366
```

- What are the top 10 flights from airport to airport?

```
graph.triplets.sortBy(_.attr, ascending=false).map(triplet =>
    "There were " + triplet.attr.toString + " flights from " + triplet.srcAttr + " to "
 + triplet.dstAttr + ".").take(10)
```

```
res60: Array[String] = Array(There were 13788 flights from SFO to LAX., There were 1339
0 flights from LAX to SFO., There were 12383 flights from OGG to HNL., There were 12035
 flights from LGA to BOS., There were 12029 flights from BOS to LGA., There were 12014
 flights from HNL to OGG., There were 11773 flights from LAX to LAS., There were 11729
 flights from LAS to LAX., There were 11257 flights from LAX to SAN., There were 11224
 flights from SAN to LAX.)
```

Graph Operations

- What are the lowest 10 flights from airport to airport?

```
graph.triplets.sortBy(_.attr).map(triplet =>
    "There were " + triplet.attr.toString + " flights from " + triplet.srcAttr + " to " + tr
iplet.dstAttr + ".").take(10)
```

```
res62: Array[String] = Array(There were 1 flights from RNO to PIH., There were 1 flights fro
m PHL to ICT., There were 1 flights from FSD to PIA., There were 1 flights from RIC to JAX.,
 There were 1 flights from MOD to BFL., There were 1 flights from ASE to MSN., There were 1
 flights from JFK to HPN., There were 1 flights from MCO to LIT., There were 1 flights from
 ROA to BWI., There were 1 flights from OMA to ABQ.)
```

Graph Operations

what airport has the most in degrees or unique flights into it?

```
graph.inDegrees.join(airportVertices).sortBy(_._2._1, ascending=false).take(1)
```

```
Array[(org.apache.spark.graphx.VertexId, (Int, String))] = Array((2042033420,(173,ATL)))
```

And out of it?

```
graph.outDegrees.join(airportVertices).sortBy(_._2._1, ascending=false).take(1)
```

```
Array[(org.apache.spark.graphx.VertexId, (Int, String))] = Array((2042033420,(173,ATL)))
```

Graph Operations

What are our most important airports ?

```
val ranksAndAirports = ranks.join(airportVertices).sortBy(_._2._1, ascending=false).map(_._
2._2)
ranksAndAirports.take(10)
```

```
res81: Array[String] = Array(ATL, DFW, ORD, MSP, SLC, DEN, DTW, IAH, CVG, LAX)
```

Graph Operations

Output the routes where the distance between airports exceeds 1000 miles

```
graph.edges.filter {
case (Edge(org_id, dest_id, distance)) => distance > 1000
    }.take(5).foreach(println)
```


Graph Operations

Output the airport with maximum incoming flight

```
// Define a reduce operation to compute the highest degree vertex
def max: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
if (a._2 > b._2) a else b
}
// Compute the max degrees
val maxInDegree: (VertexId, Int) = graph.inDegrees.reduce(max)
```

Graph Operations

Output the airports with maximum incoming flights

```
val maxIncoming=graph.inDegrees.collect.sortWith (_._2 >
        _._2).map(x => (airportMap(x._1),
        x._2)).take(10).foreach(println)
```

Graph Operations

Output the longest routes

```
graph.triplets.sortBy(_.attr, ascending = false).map(triplet =>
"There were " + triplet.attr.toString + " flights from " + triplet.srcAttr
    + " to " + triplet.dstAttr + ".").take(20).foreach(println)
```

Graph Operations

Output the cheapest airfare routes

```
val gg = graph.mapEdges(e => 50.toDouble + e.attr.toDouble / 20)
//Call pregel on graph
val sssp = initialGraph.pregel(Double.PositiveInfinity)(
//vertex program
(id, distCost, newDistCost) => math.min(distCost, newDistCost),triplet => {
//send message
if(triplet.srcAttr + triplet.attr < triplet.dstAttr)
{
Iterator((triplet.dstId, triplet.srcAttr + triplet.attr))
}
else
{
Iterator.empty
}
},
//Merge Messages
(a,b) => math.min(a,b)
)
//print routes with lowest flight cost
print("routes with lowest flight cost")
println(sssp.edges.take(10).mkString("\n"))
```

Graph Operations (PageRank)

Output the most influential airports using PageRank

```
val rank = graph.pageRank(0.001).vertices
val temp = rank.join(airports)
temp.take(10).foreach(println)
```

Output the most influential airports from most influential to latest

```
val temp2 = temp.sortBy(_._2._1,false)
```

Conclusion

- The growing scale and importance of graph data has driven the development of specialized graph computation engines capable of inferring complex recursive properties of graph-structured data.

- In this lecture we have discussed GraphX, a distributed graph processing framework that unifies graph-parallel and data-parallel computation in a single system and is capable of succinctly expressing and efficiently executing the entire graph analytics pipeline.