Introduction to Graph Analytics

EMSE 6992 Introduction to Data Science

Benjamin S. Harvey

Reminders

- Assignment 3 Due today 11/29 (midnight)
  - Send me your portfolio links
- Assignment 4 Due 12/7
  - Show in class before your presentation
- We will have project and portfolio presentations on 12/6

Project Presentations

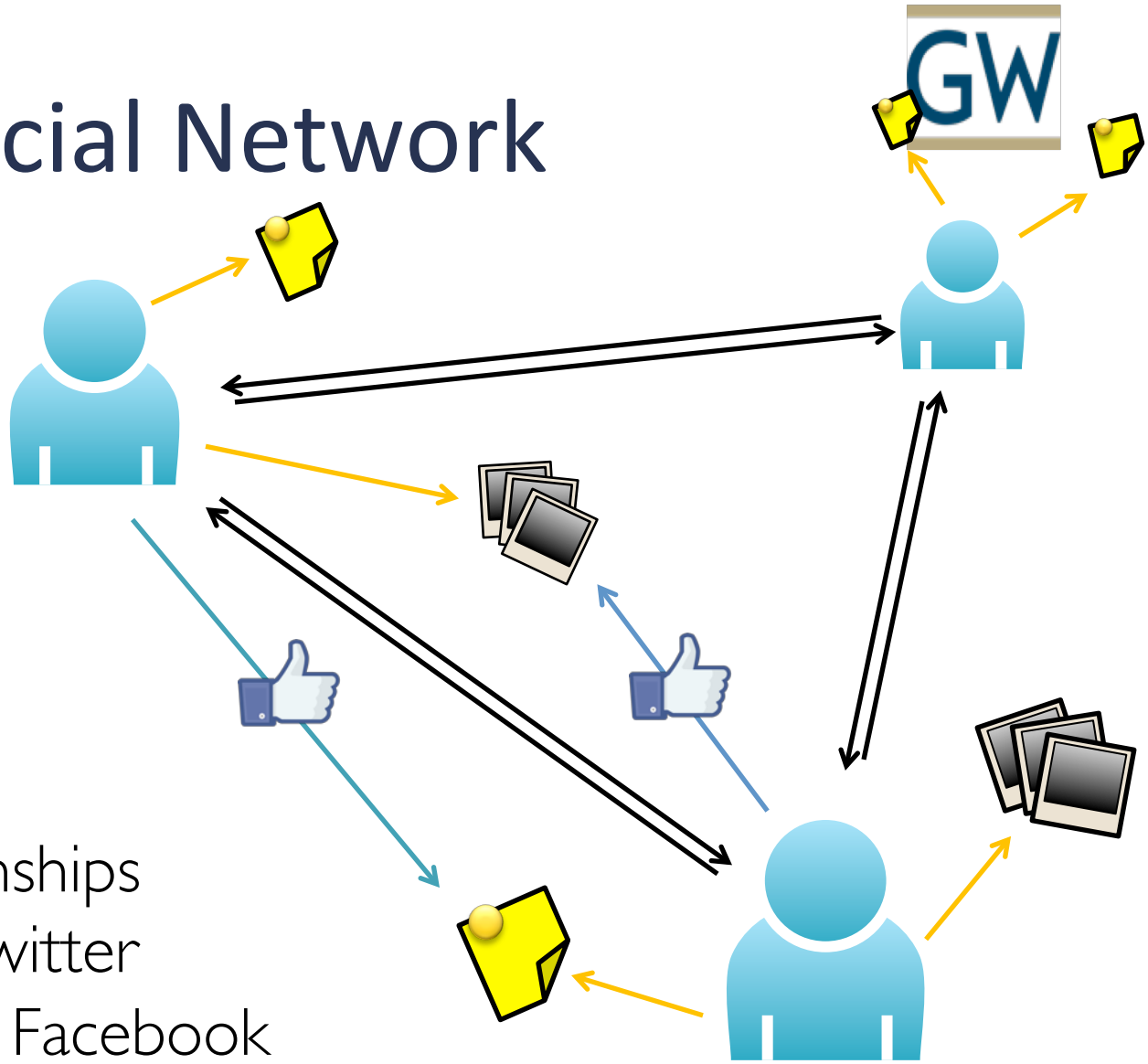
Last_Name	First_Name	ID	Time
Akinbule	Olatunji	akinbule_ola	6:00 - 6:12
Alshamrani	Amirah	amirah_shumrani	6:12 - 6:24
Han	Xiao	hanxiao	6:24 - 6:36
He	Yanjie	heyanjie	6:36 - 6:48
Pei	Lihua	lihuapei	6:48 - 7:00
Ren	Zhijie	zren88	7:00 - 7:12
Stumpf	Jonathan	jcstumpf	7:12 - 7:24
Tarar	Sadiqa	sadiqatarar	7:24 - 7:36
Tu	Ping	ptu16	7:36 - 7:48
Turano	Madison	madly9	7:48 - 8:00
Wang	Peiyu	oliviawpy	8:00 - 8:12
Warndorf	Maddie	mwarndorf65	8:12 - 8:24
Yu	Haotian	yuxx6789	8:24 - 8:36
Zhao	Lujin	lujin	8:36 - 8:48
Zhu	Ziqiu	zzhu158	8:48 - 9:00

Outline

1. Graph structured data
2. Common properties of graph data
3. Graph algorithms
4. Systems for large-scale graph computation
5. GraphX: Graph Computation in Spark
6. Summary of other graph frameworks

Graph structured data is everywhere …

Social Network



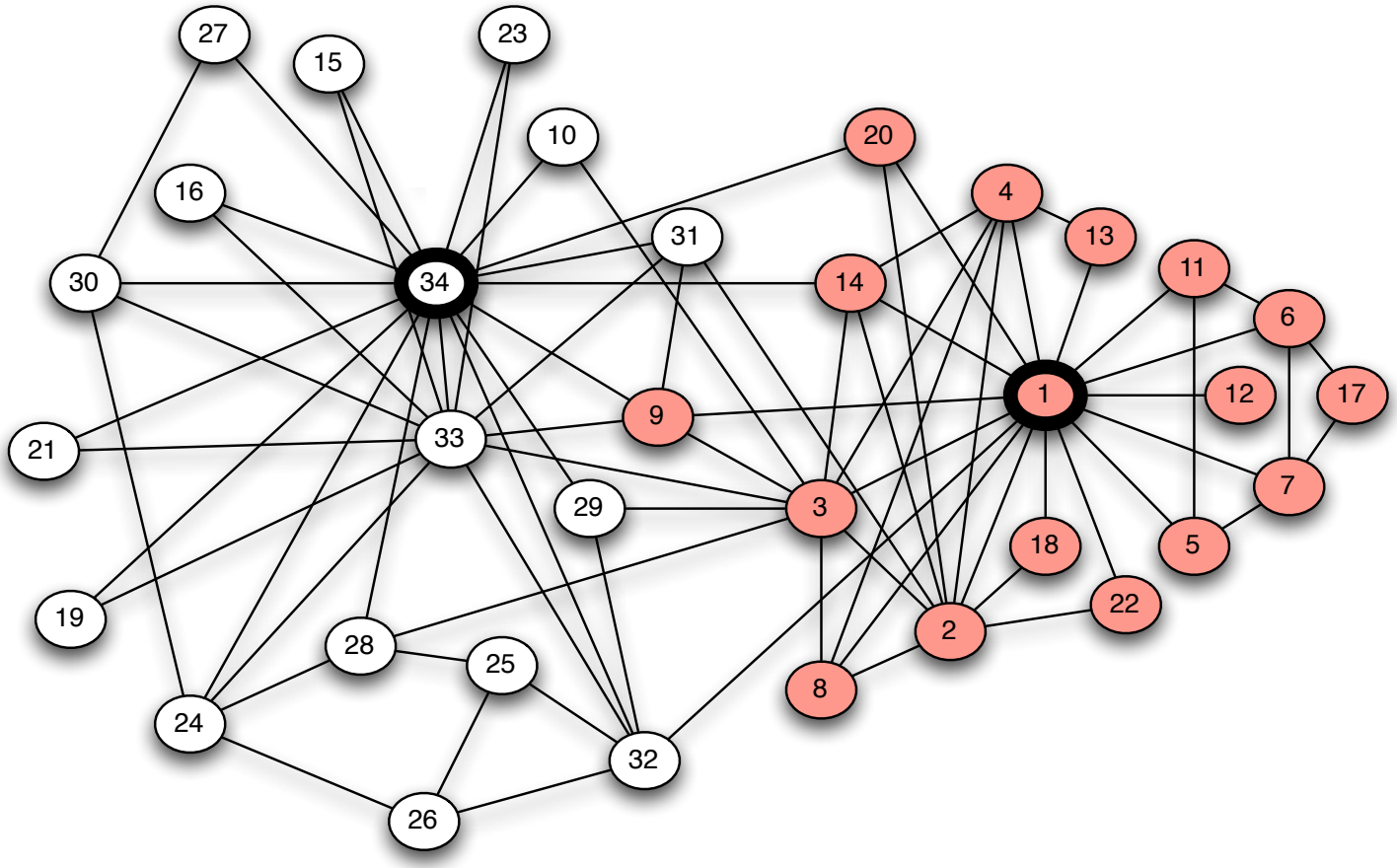
Vertices

- Users
- Posts / Images

Edges

- Social Relationships
  - *Directed:* Twitter
  - *Undirected:* Facebook
- Likes

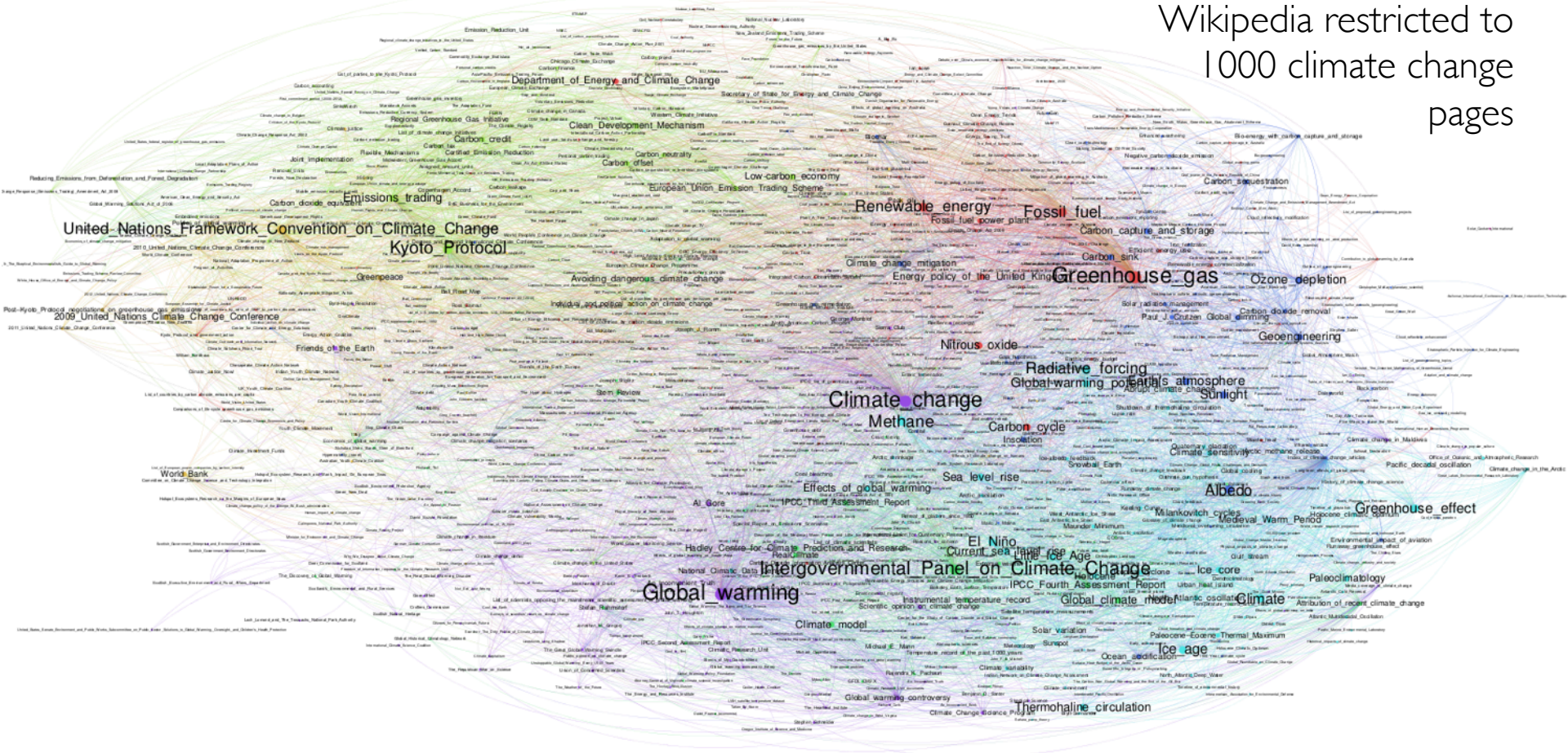
Actual Social Graph



Karate Club Network

Web Graphs

Wikipedia restricted to 1000 climate change pages



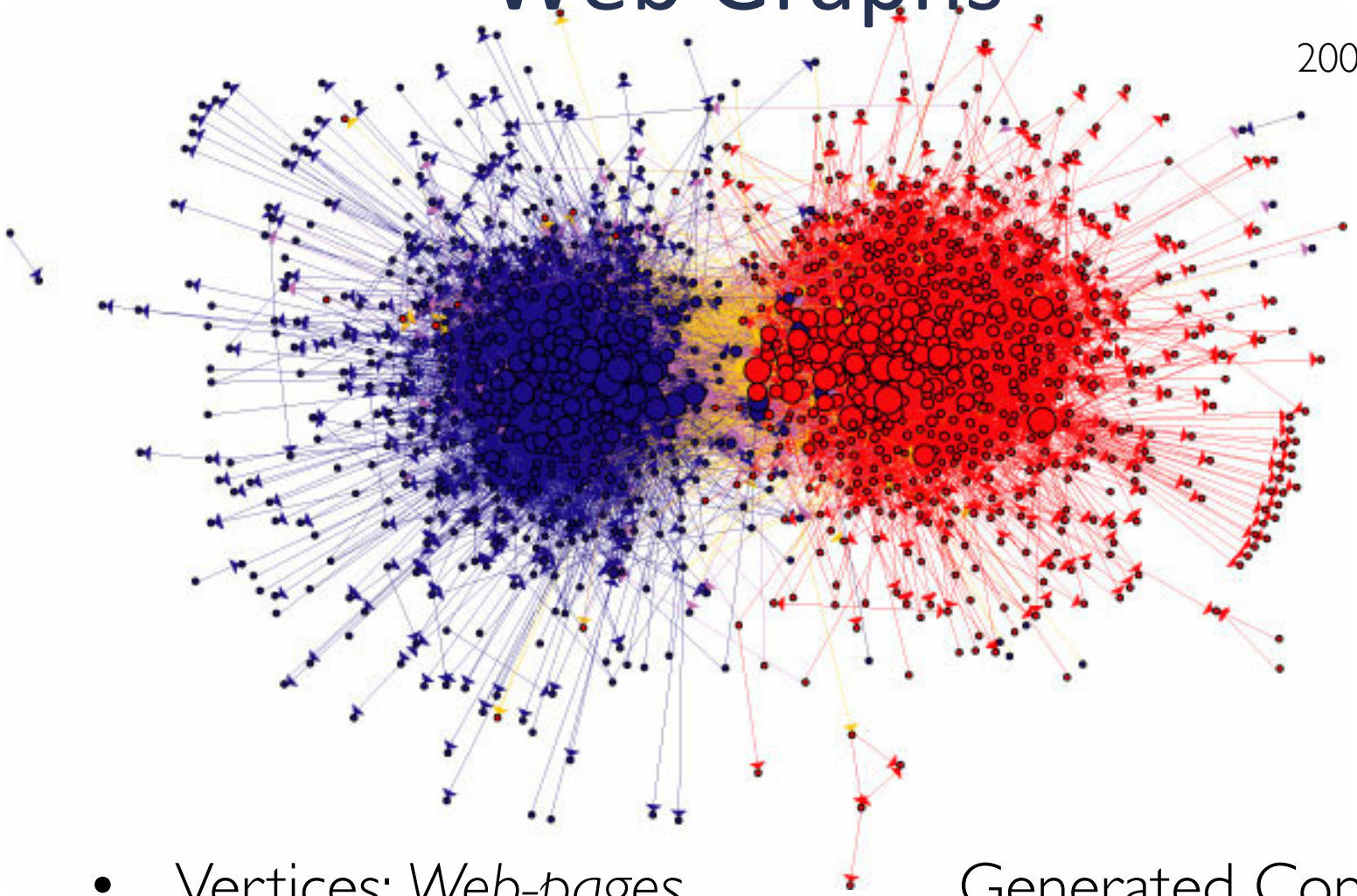
- Vertices: *Web-pages*
- Edges: *Links (Directed)*

Generated Content:

- Click-streams

Web Graphs

2004 Political Blogs



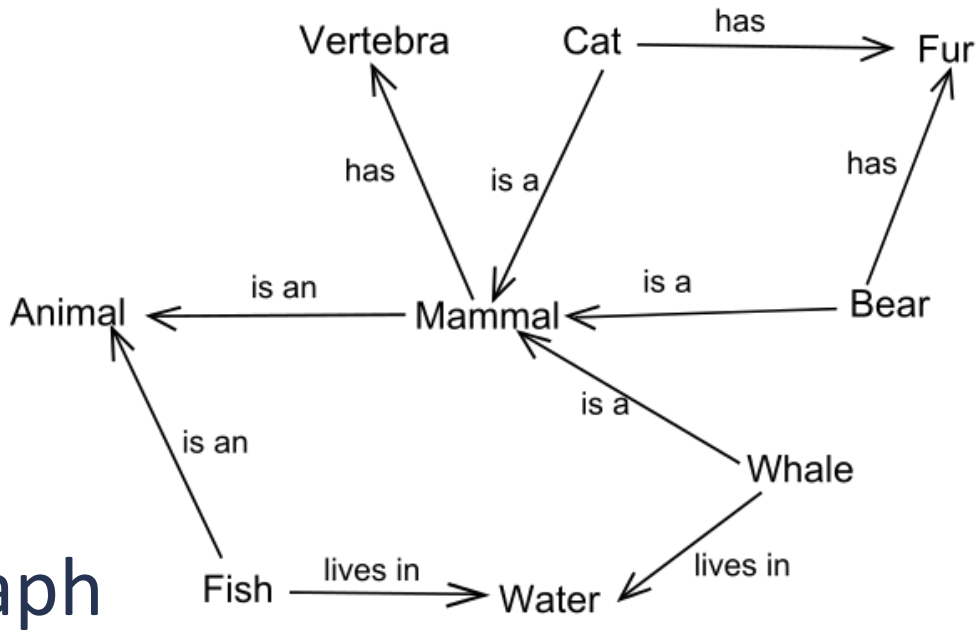
- Vertices: *Web-pages*
- Edges: *Links (Directed)*

Generated Content:

- Click-streams

Semantic Networks

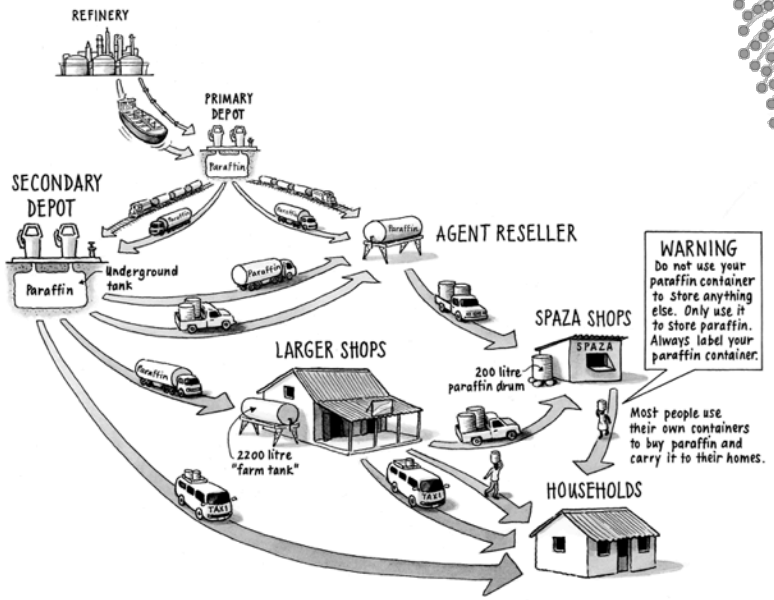
- Organize Knowledge
- Vertices: Subject, Object
- Edges: Predicates
- Example: Google Knowledge Graph
  - 570M Vertices
  - 18B Edges



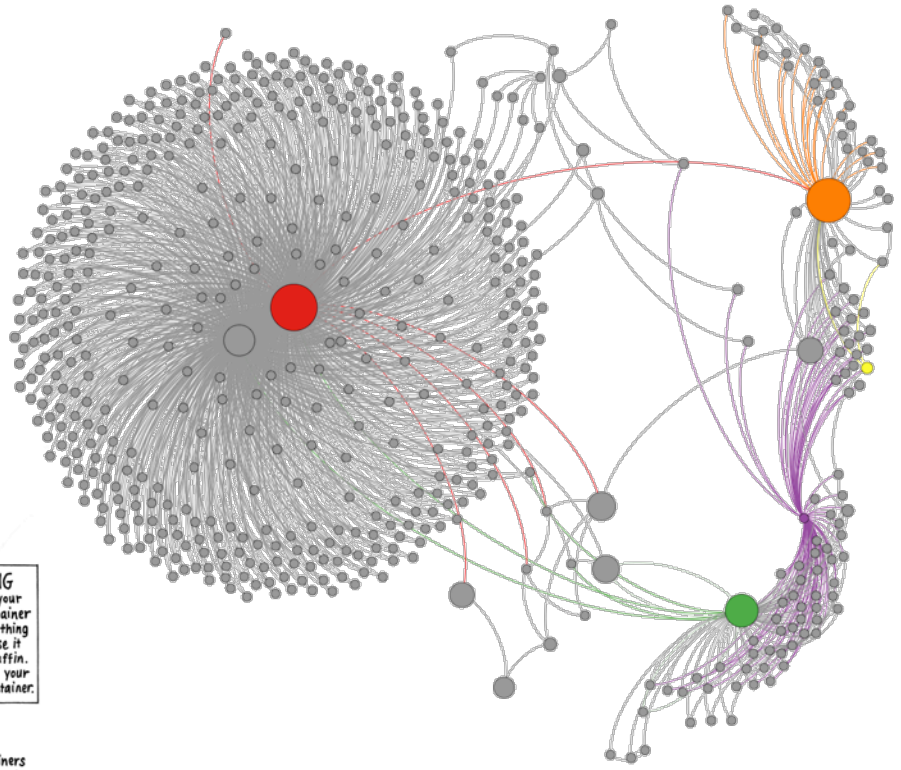
http://wiki.dbpedia.org

Transaction Networks

Supply Chain:
Vertices: Suppliers/Consumers
Edges: Exchange of Goods



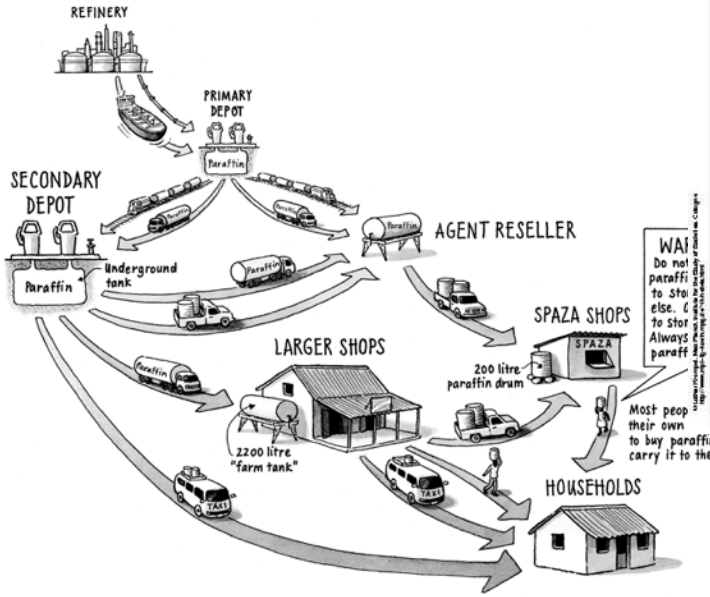
http://anonymity-in-bitcoin.blogspot.com/2011/07/bitcoin-is-not-anonymous.html

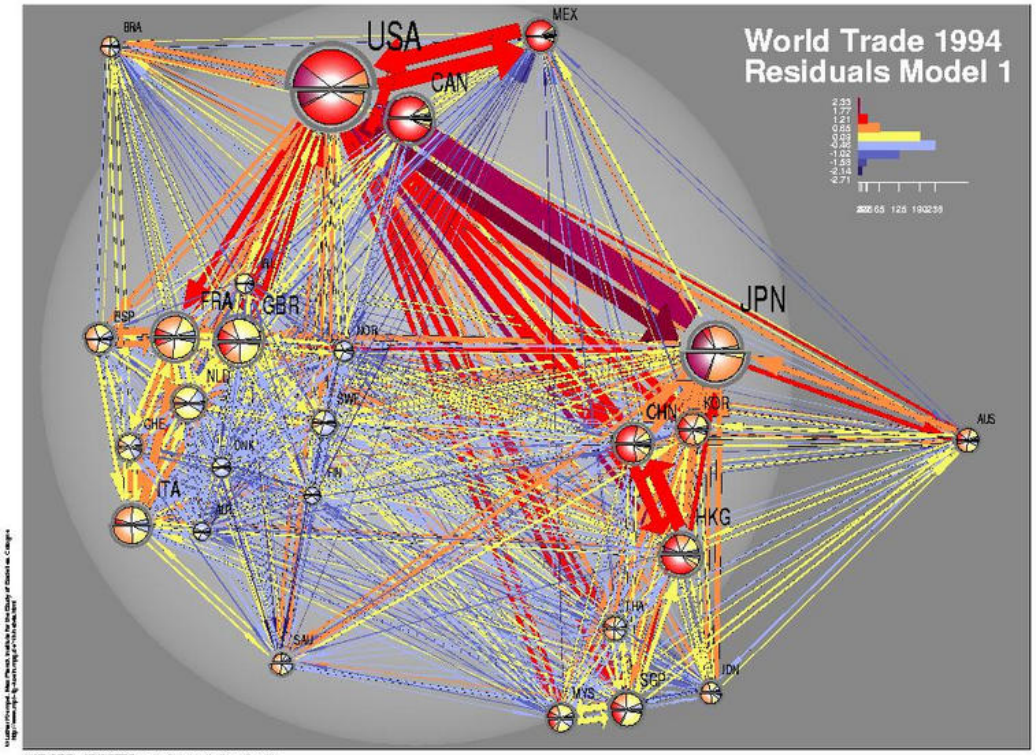


Transaction Networks (e.g., Bitcoin):
Vertices: Users
Edges: Exchange of Currency

Transaction Networks

Supply Chain:
Vertices: Suppliers/Consumers
Edges: Exchange of Goods





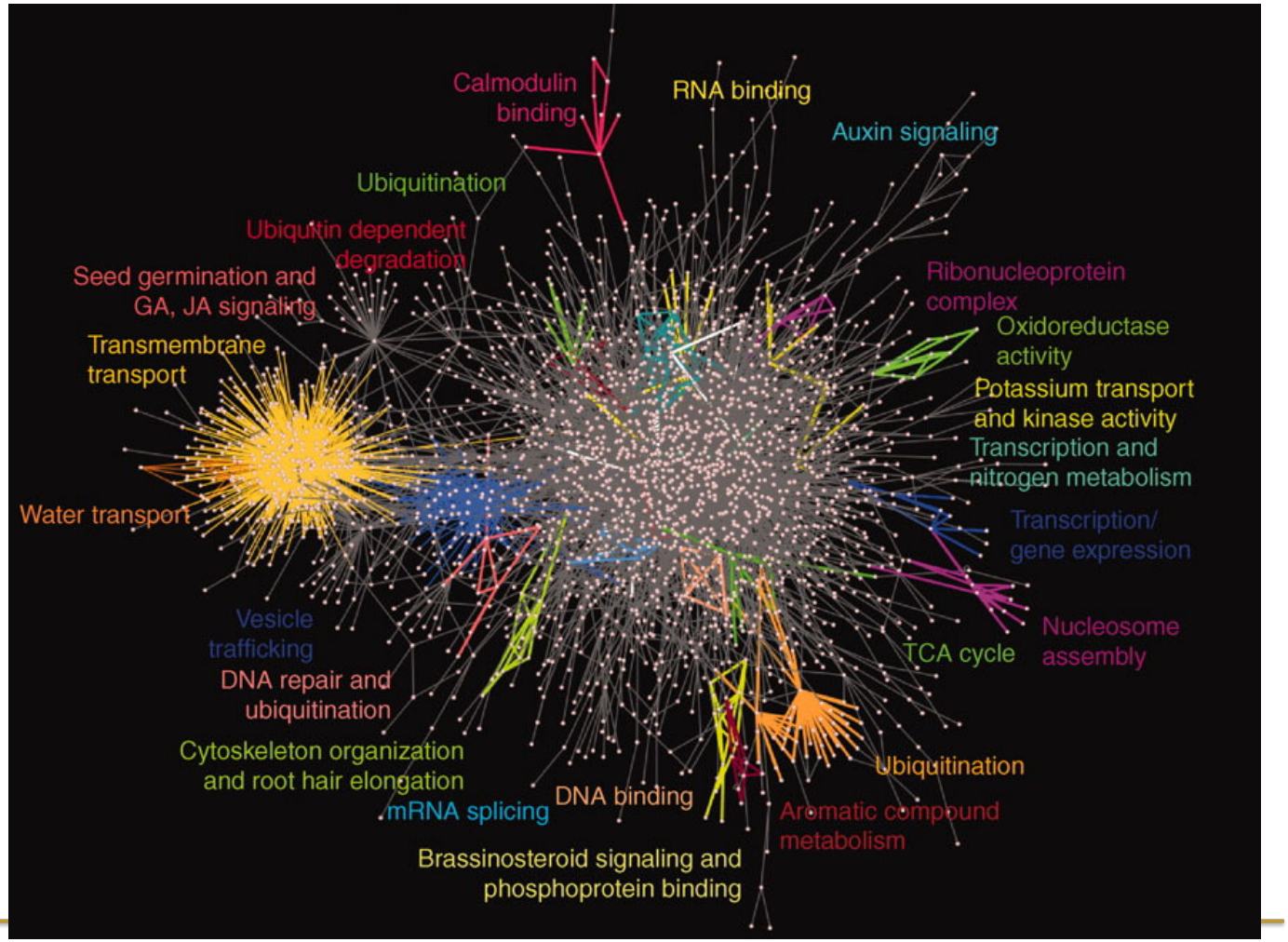
Transaction Networks (e.g., Bitcoin):
Vertices: Users
Edges: Exchange of Currency

Biological Networks

Protein-Protein Interaction Networks (Interactomes)

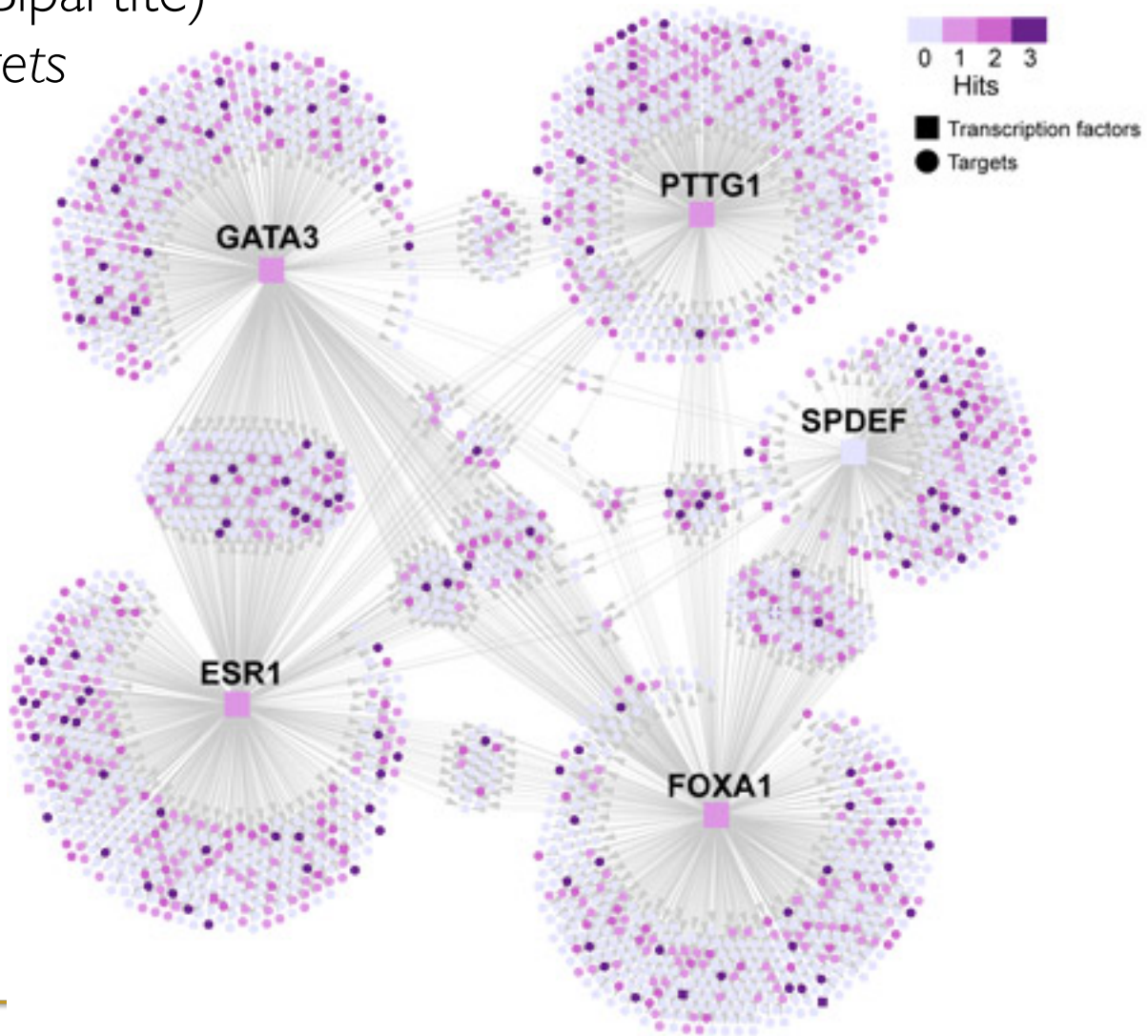
Vertices: *Proteins*

Edges: *Interactions*

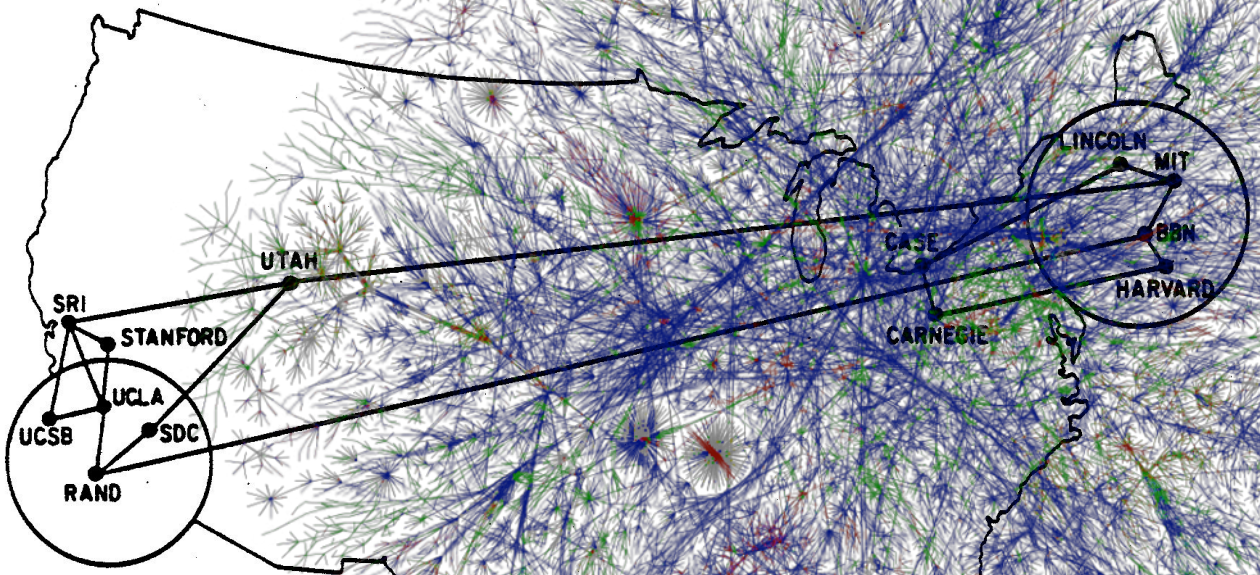


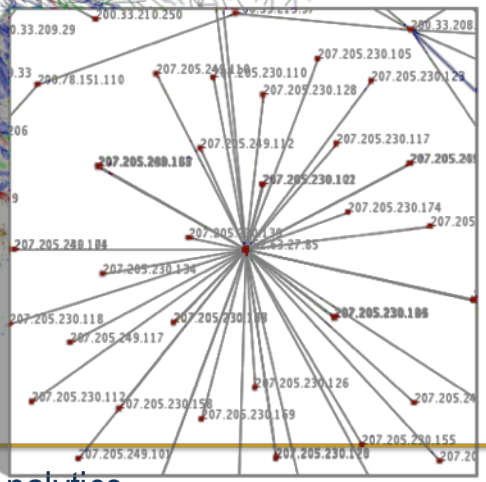
Biological Networks

Regulatory Networks (Bipartite)
Vertices: *Regulators, targets*
Edges: *Regulates target*



Communication Networks



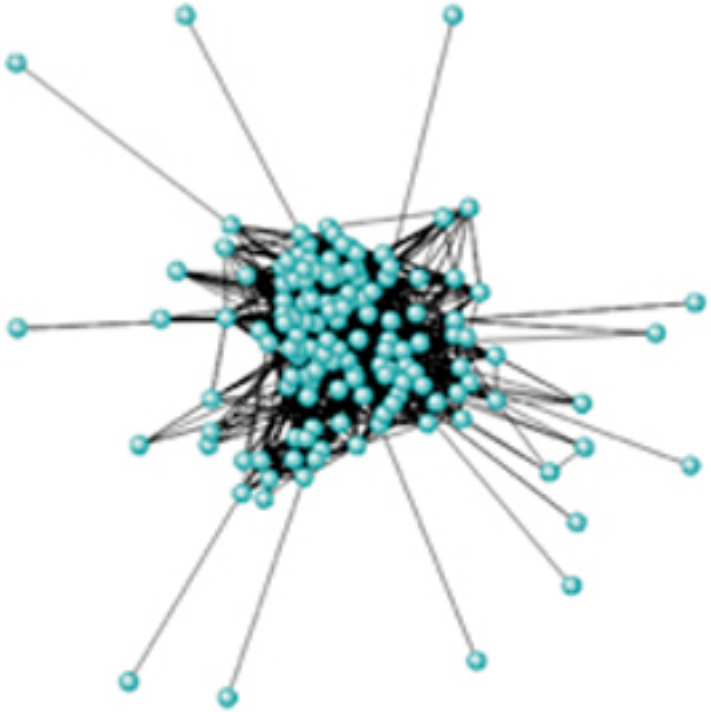


Vertices: *Devices, Routers*

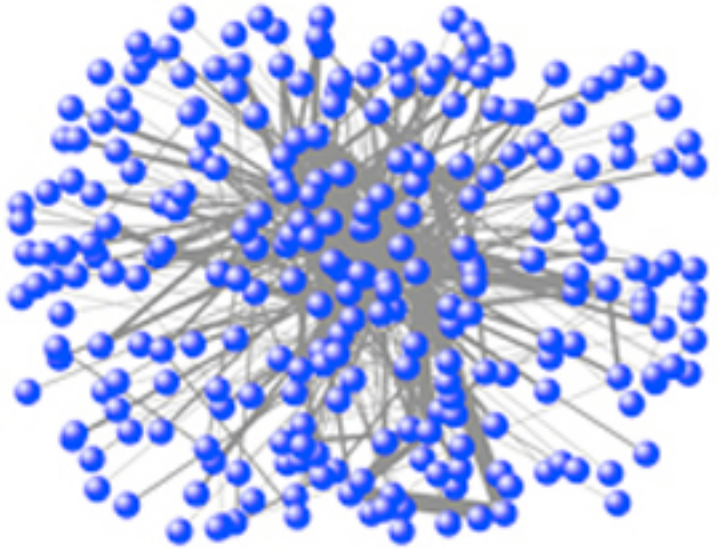
Directed Edges: *Network Flows*

Who Talks to Whom Graph

Enron Email Graphs



ILLICIT



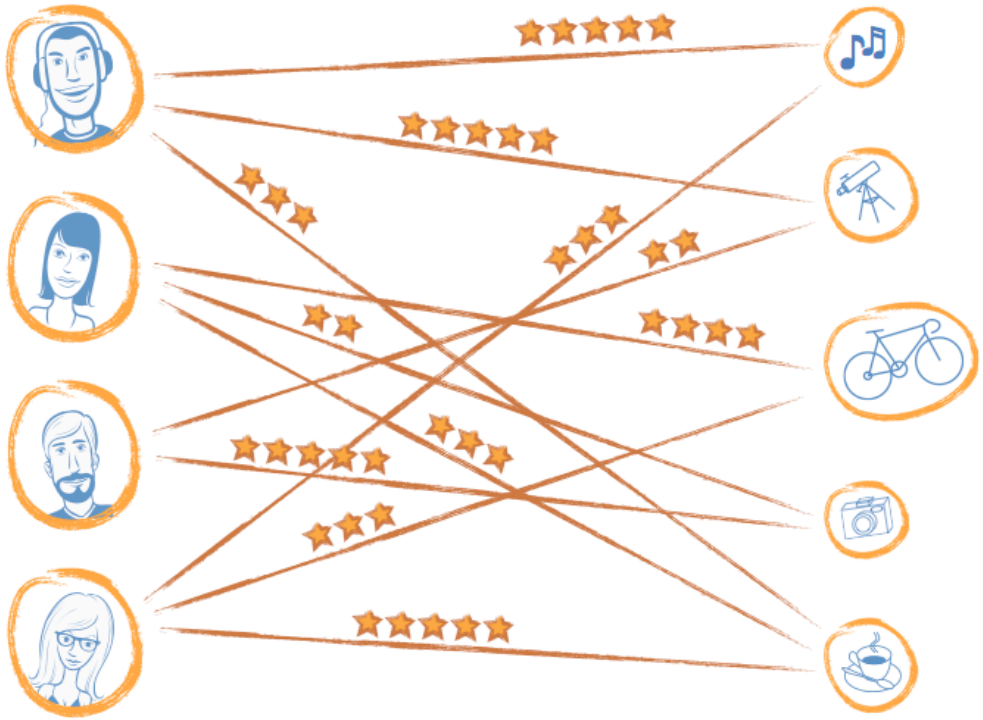
LEGAL

Vertices: *Users*

Directed Edges: *Email From → To*

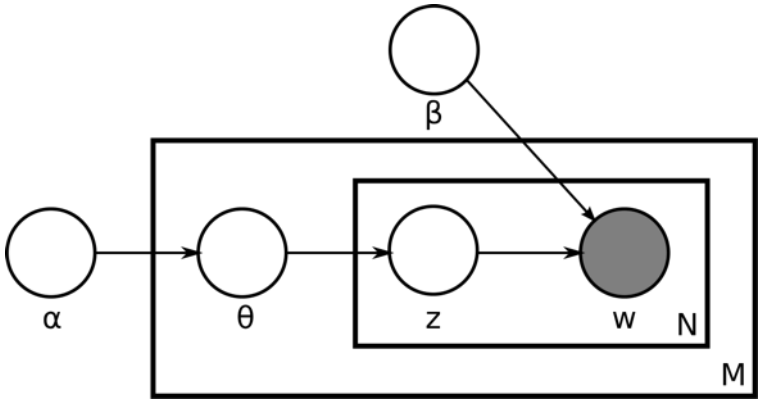
User - Item Graphs (Recommender Systems)

Bipartite Graphs
Vertices: *Users and Items*
Edges: *Ratings*

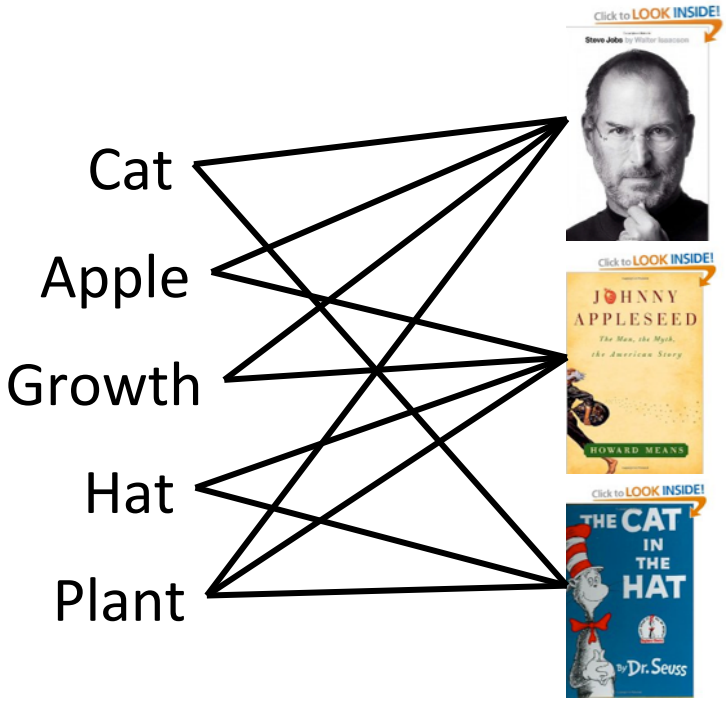


Graphical Models

Vertices: *Random Variables, Factors*
Edges: Statistical Dependencies



LDA



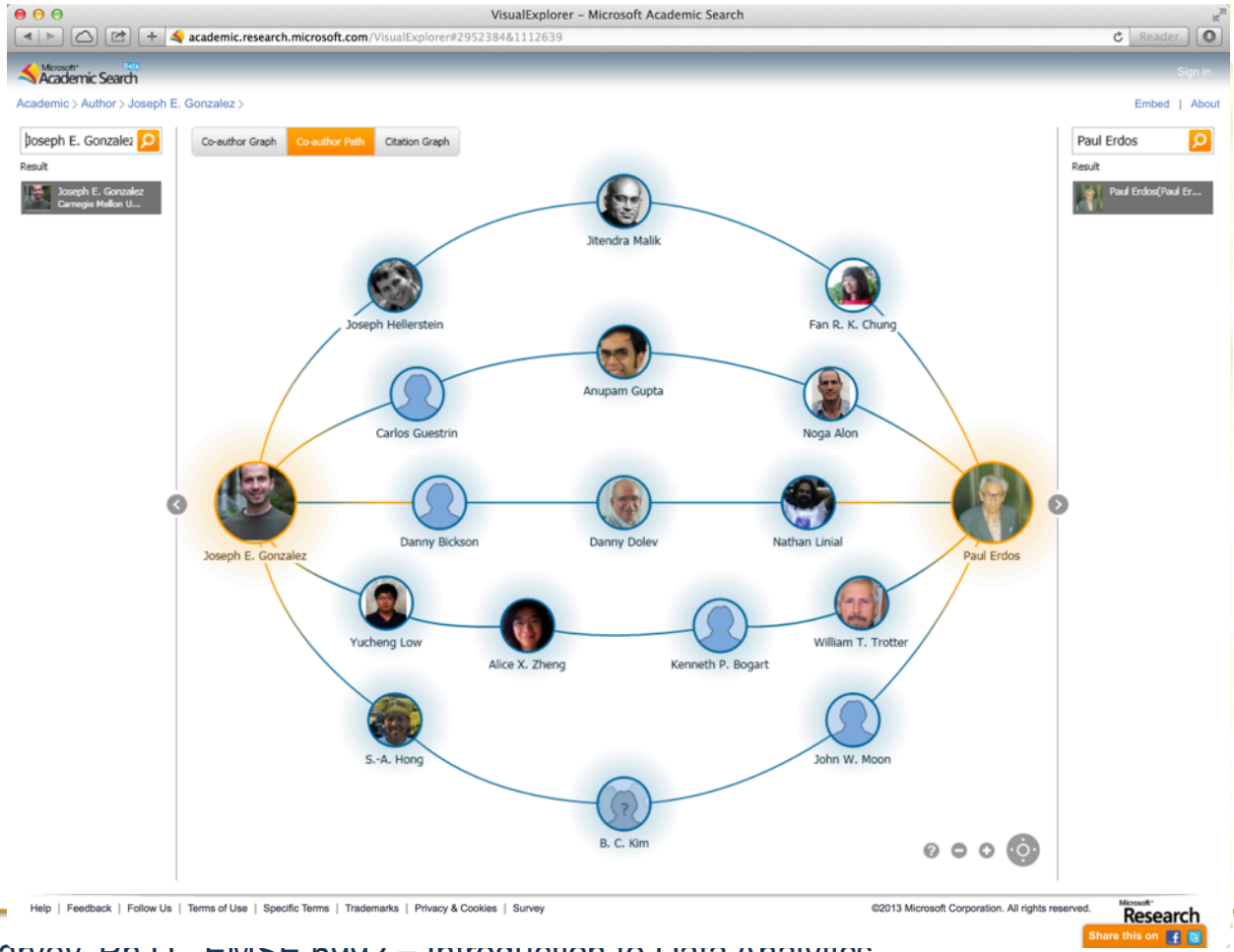
Co-Authorship Network

Vertices: *Authors*

Edges: *Co-authorship*

http://academic.research.microsoft.com/VisualExplorer#2952384&1112639

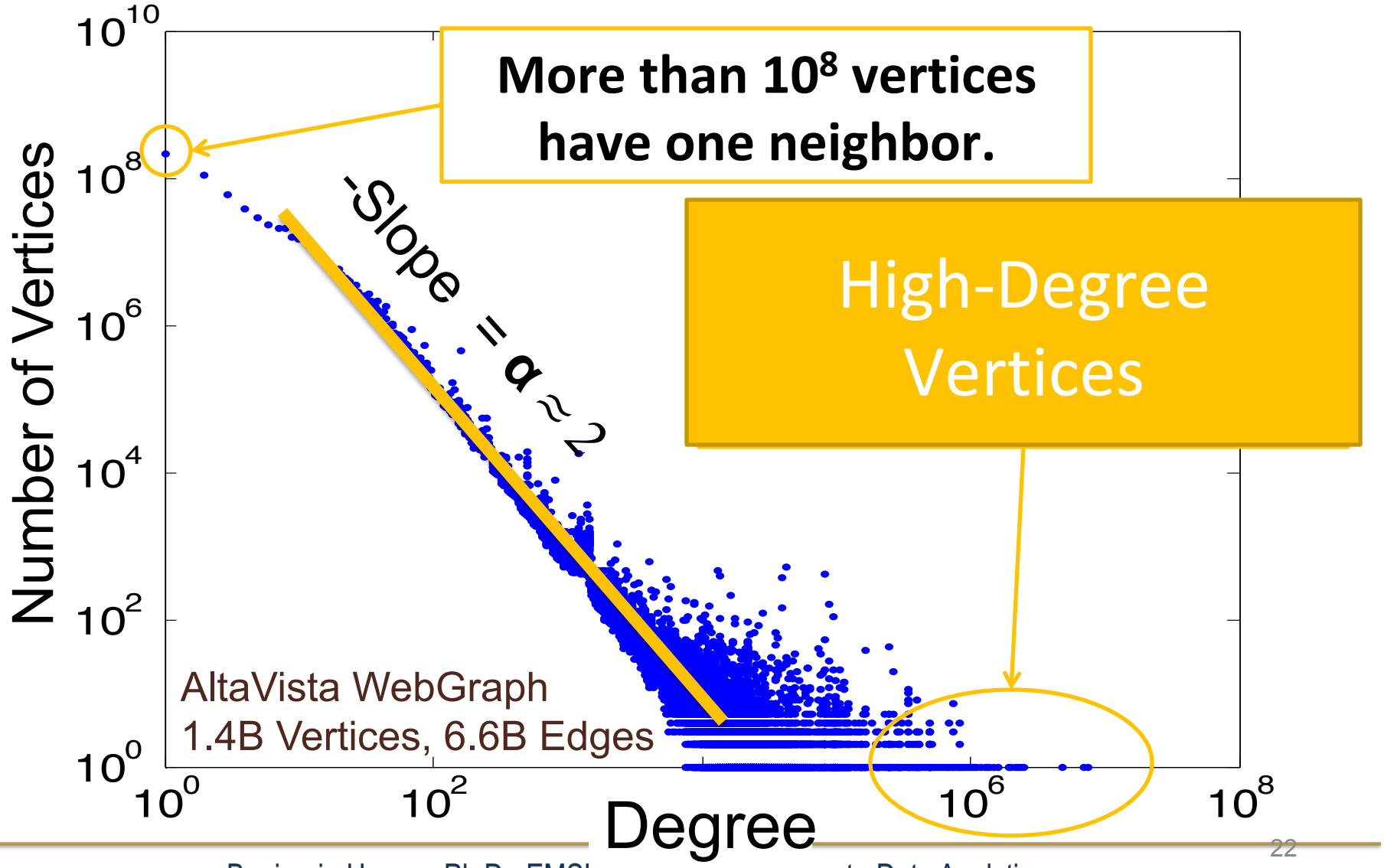
Example: Erdos Number



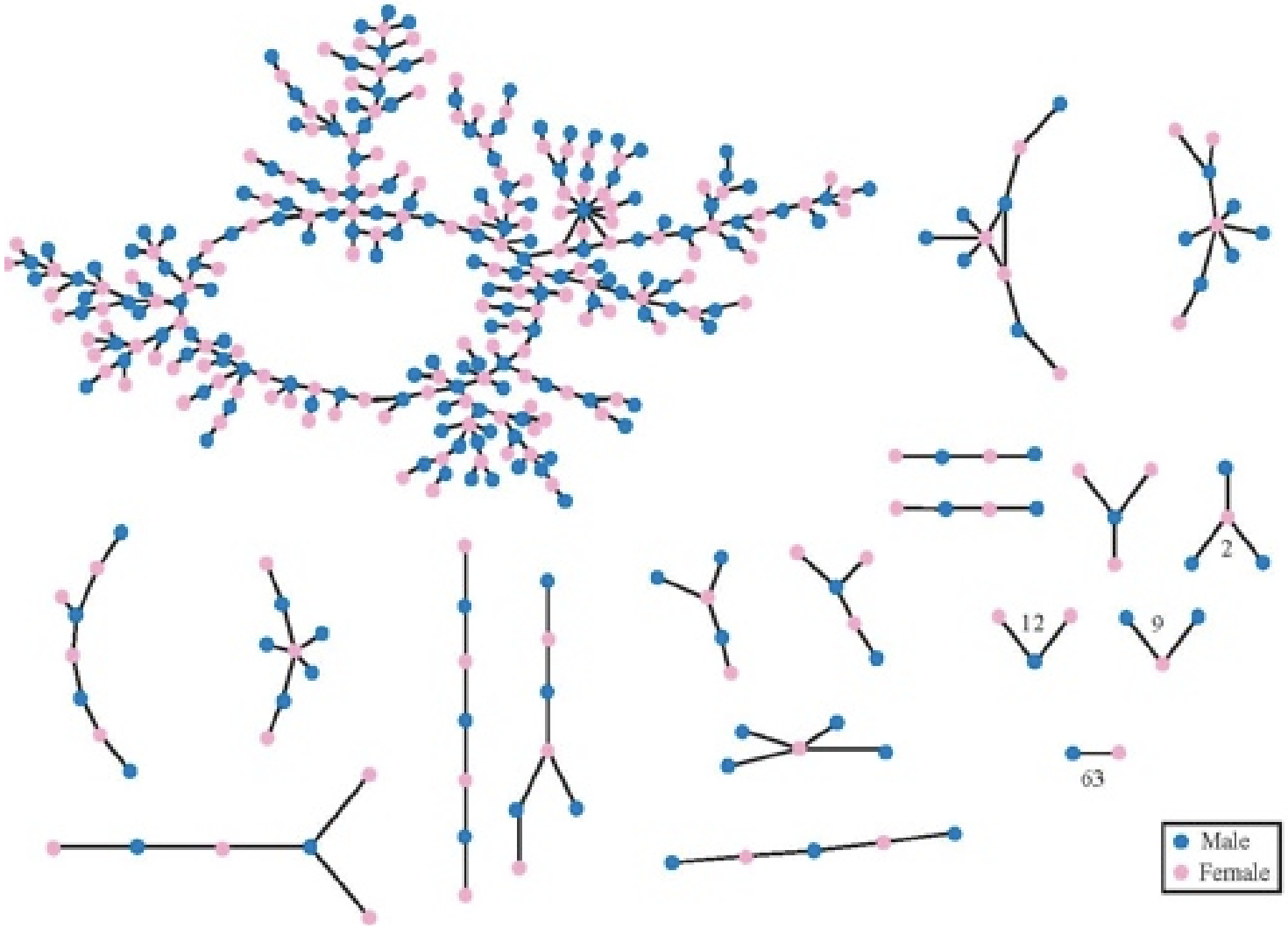
Others?

Common properties of graphs derived from natural phenomena

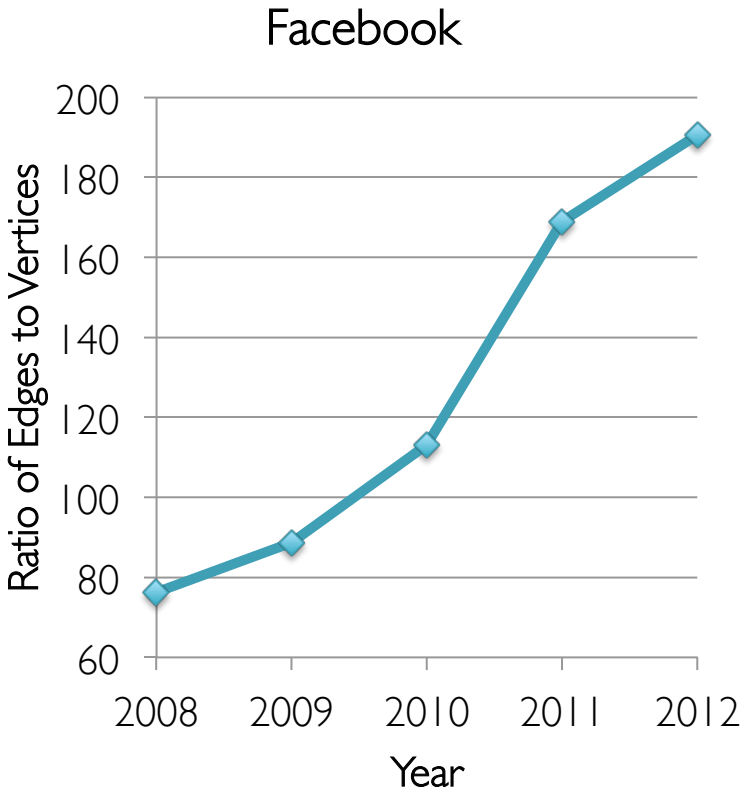
Power-Law Degree Distribution

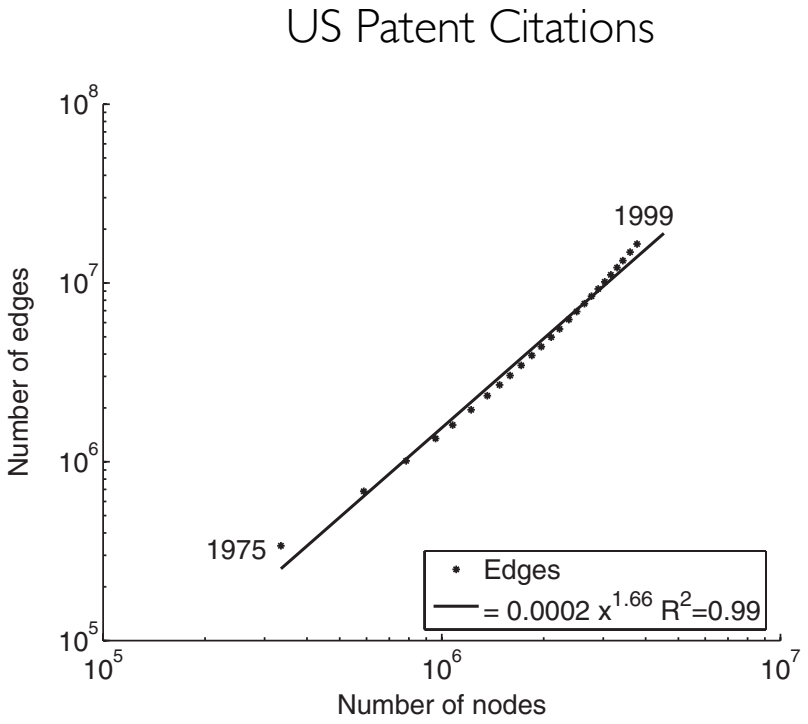


Giant Connected Component



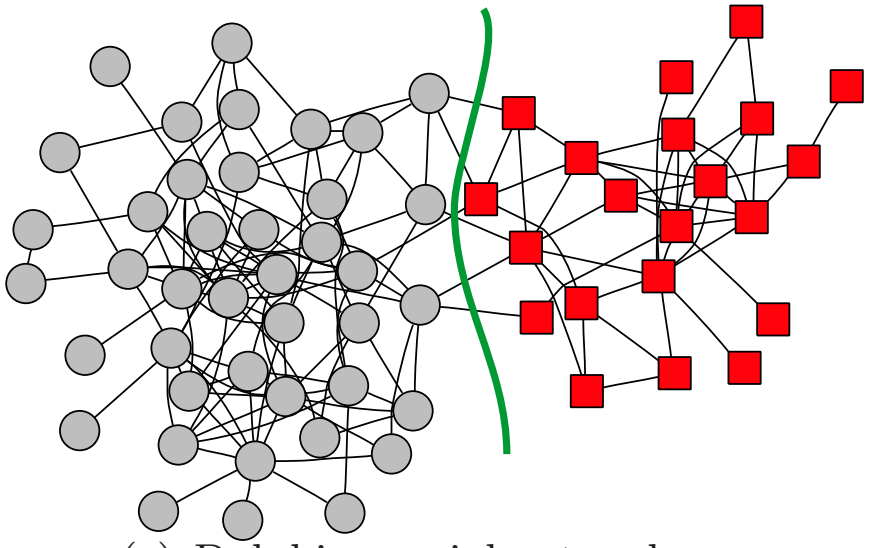
Densification



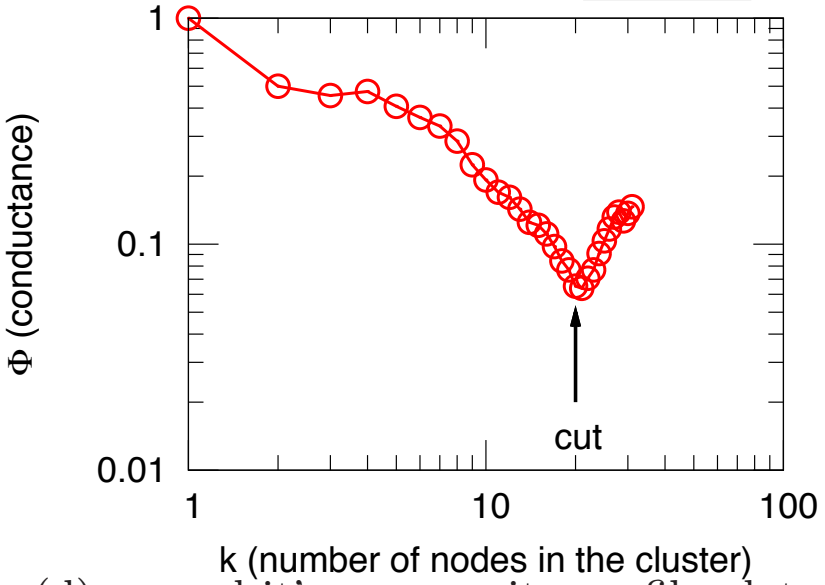


Average distance between nodes *reduces* over time.

Community Structure

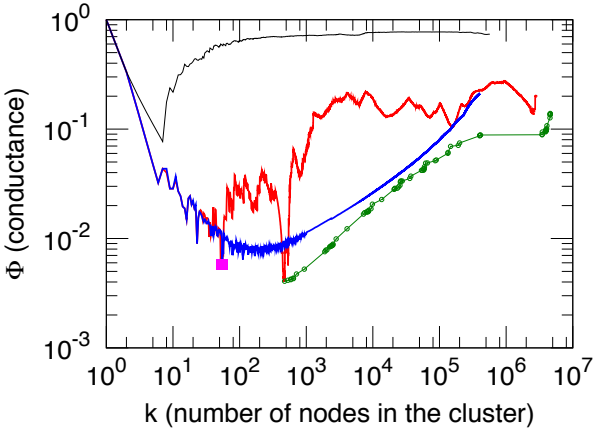


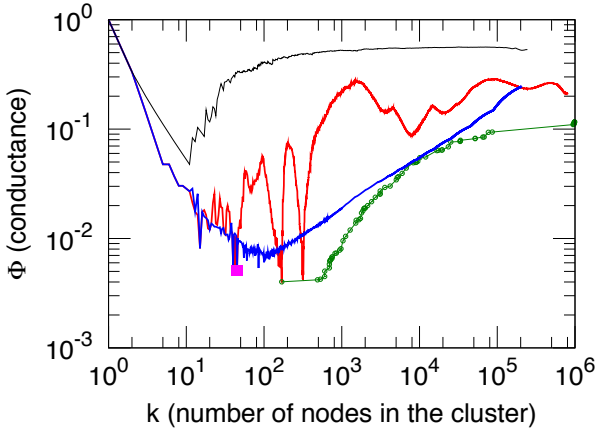
(c) Dolphins social network . . .



(d) . . . and it's community profile plot

Linked-In





Messenger

Graph Algorithms

"Think Globally, Act Locally"

PageRank (Centrality Measures)

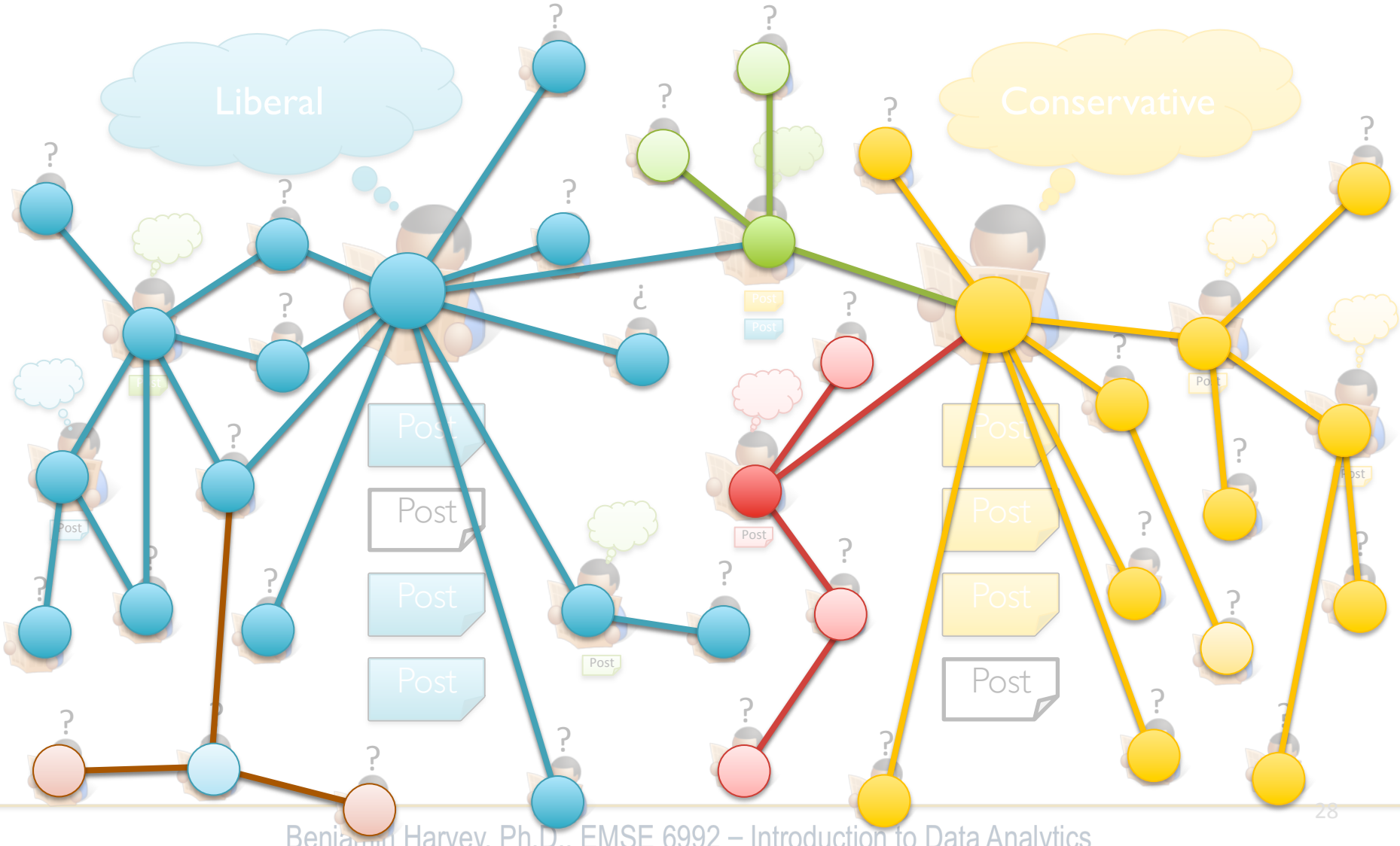
- Recursive Relationship:

$$R[i] = \alpha + (1-\alpha) \sum_{(j,i) \in E} \frac{1}{L[j]} R[j]$$

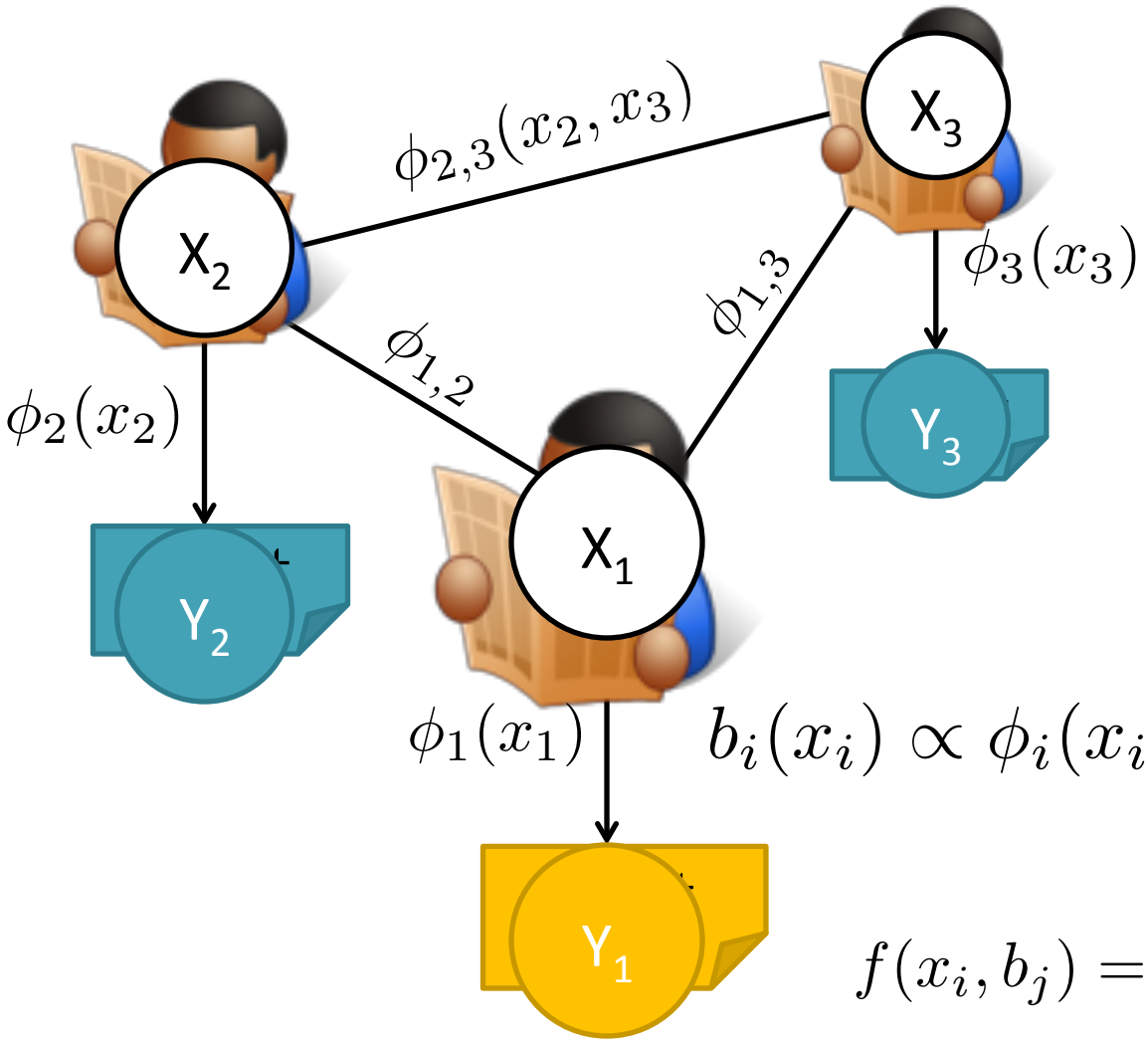
- Where:
  - α is the random reset probability (typically 0.15)
  - $L[j]$ is the number of links on page j

$$R[5] = \alpha + (1-\alpha)\left(\frac{1}{3}R[1] + \frac{1}{1}R[4]\right)$$

Predicting Behavior



Mean Field Algorithm



Sum over Neighbors

$$b_i(x_i) \propto \phi_i(x_i) \exp\left(\sum_{j \in N_i} f(x_i, b_j)\right)$$

$$f(x_i, b_j) = \sum_{x_j} b_j(x_j) \log \phi_{i,j}(x_i, x_j)$$

Label Propagation (Structured Prediction)

- Social Arithmetic:

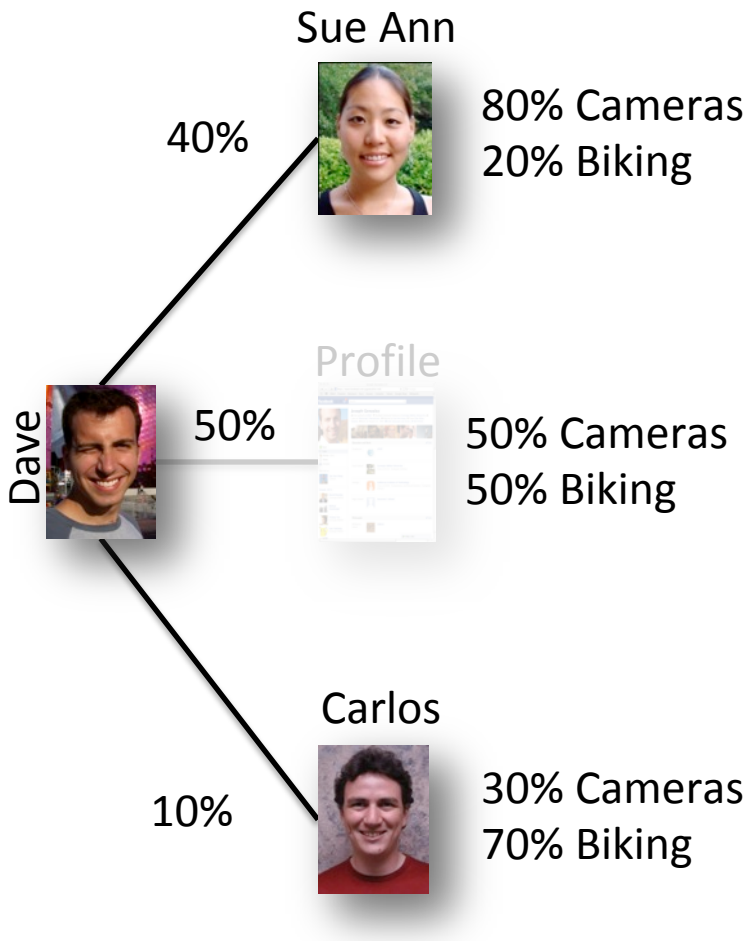
  50% What I list on my profile
  40% Sue Ann Likes
  \+ 10% Carlos Like

  I Like: 60% Cameras, 40% Biking

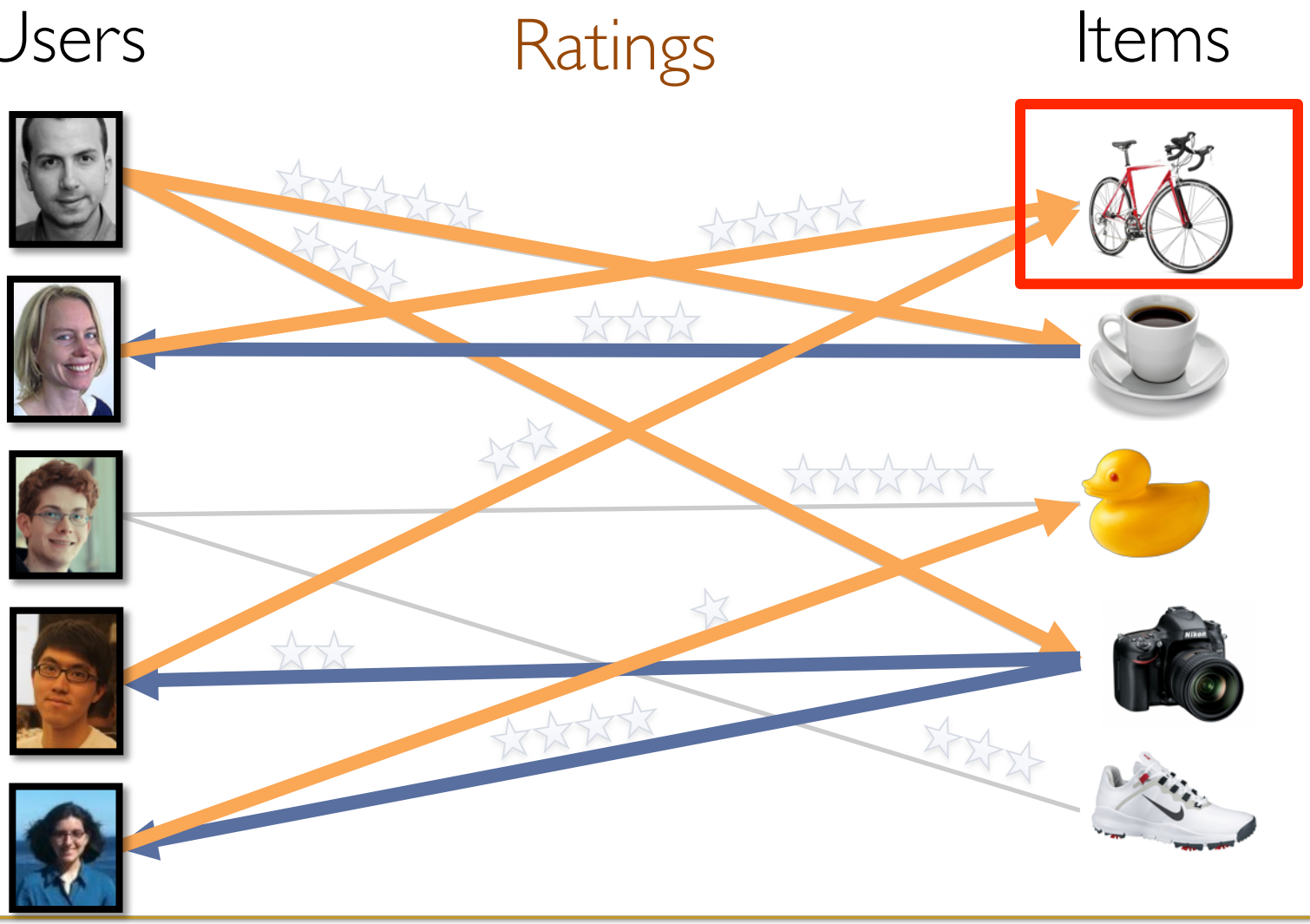
- Recurrence Algorithm:

$$Likes[i] = \sum_{j \in Friends[i]} W_{ij} \times Likes[j]$$

– iterate until convergence

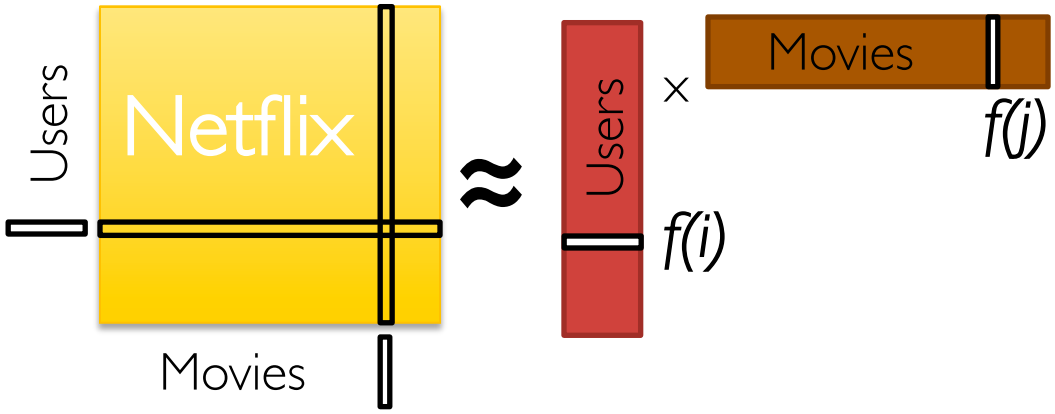


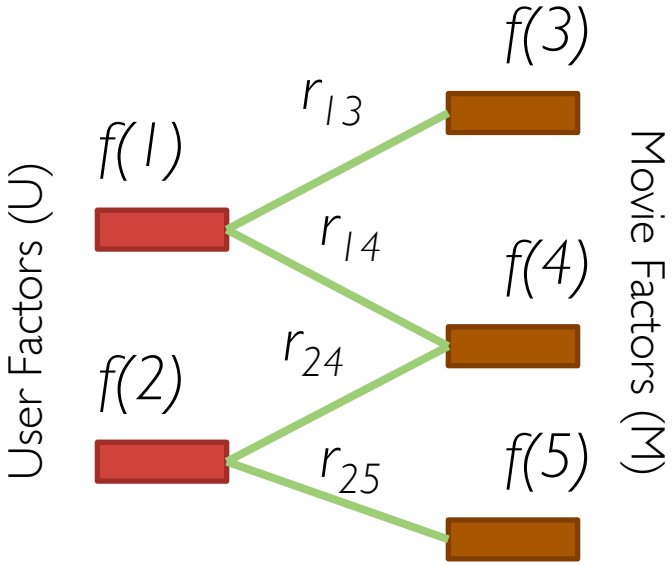
Recommending Products



Recommending Products

Low-Rank Matrix Factorization:



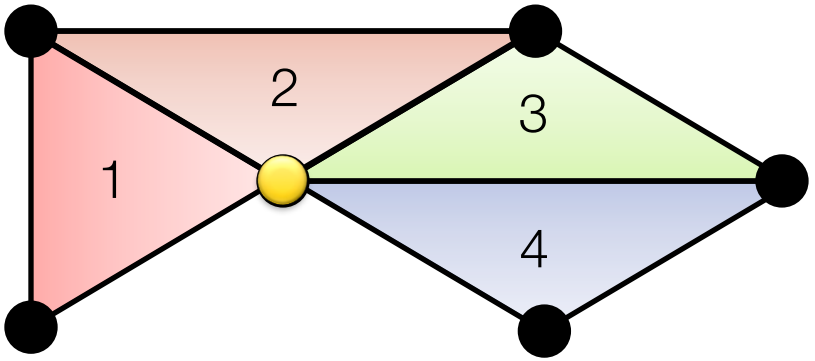


Iterate:

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

Finding Communities

- Count triangles passing through each vertex:
-

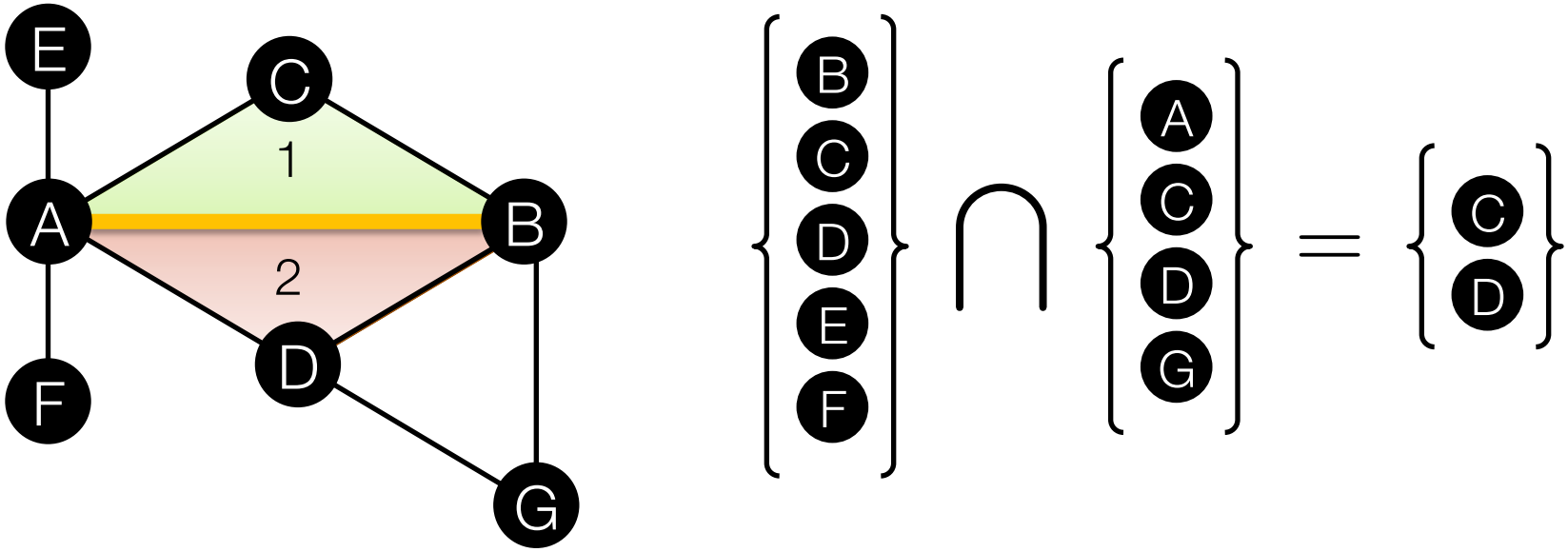


- Measure "cohesiveness" of local community

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

Counting Triangles

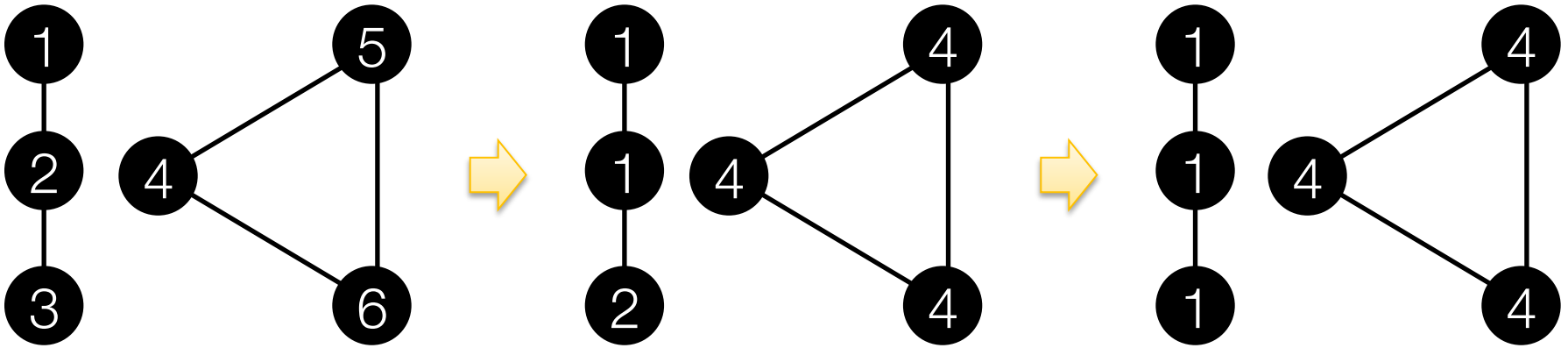
- Count triangles passing through each vertex by counting triangles on each edge:



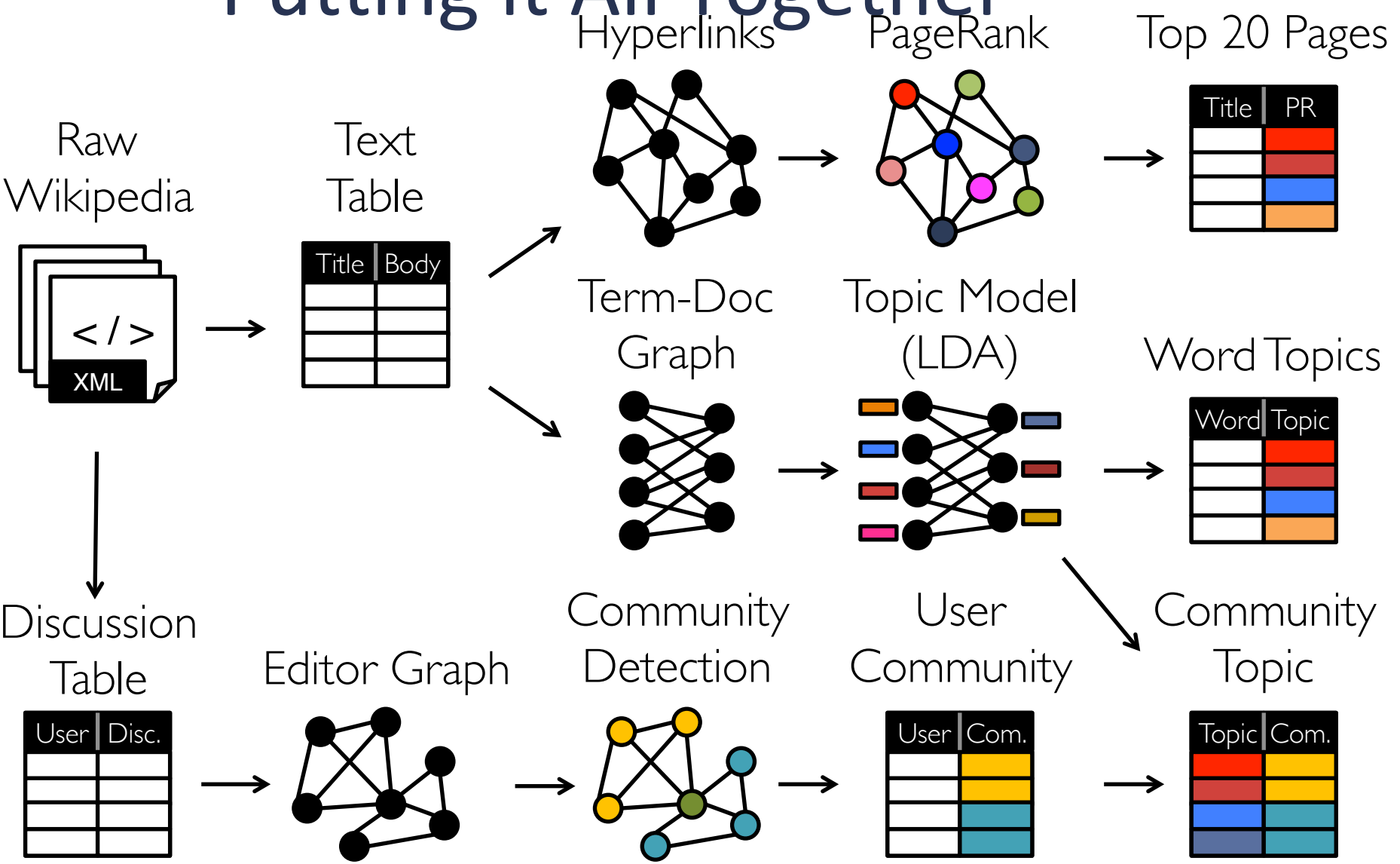
Connected Components

- Every vertex starts out with a unique component id (typically it's vertex id):

$$CC[i] = \arg \min_{\{i,j\} \in E} CC[j]$$



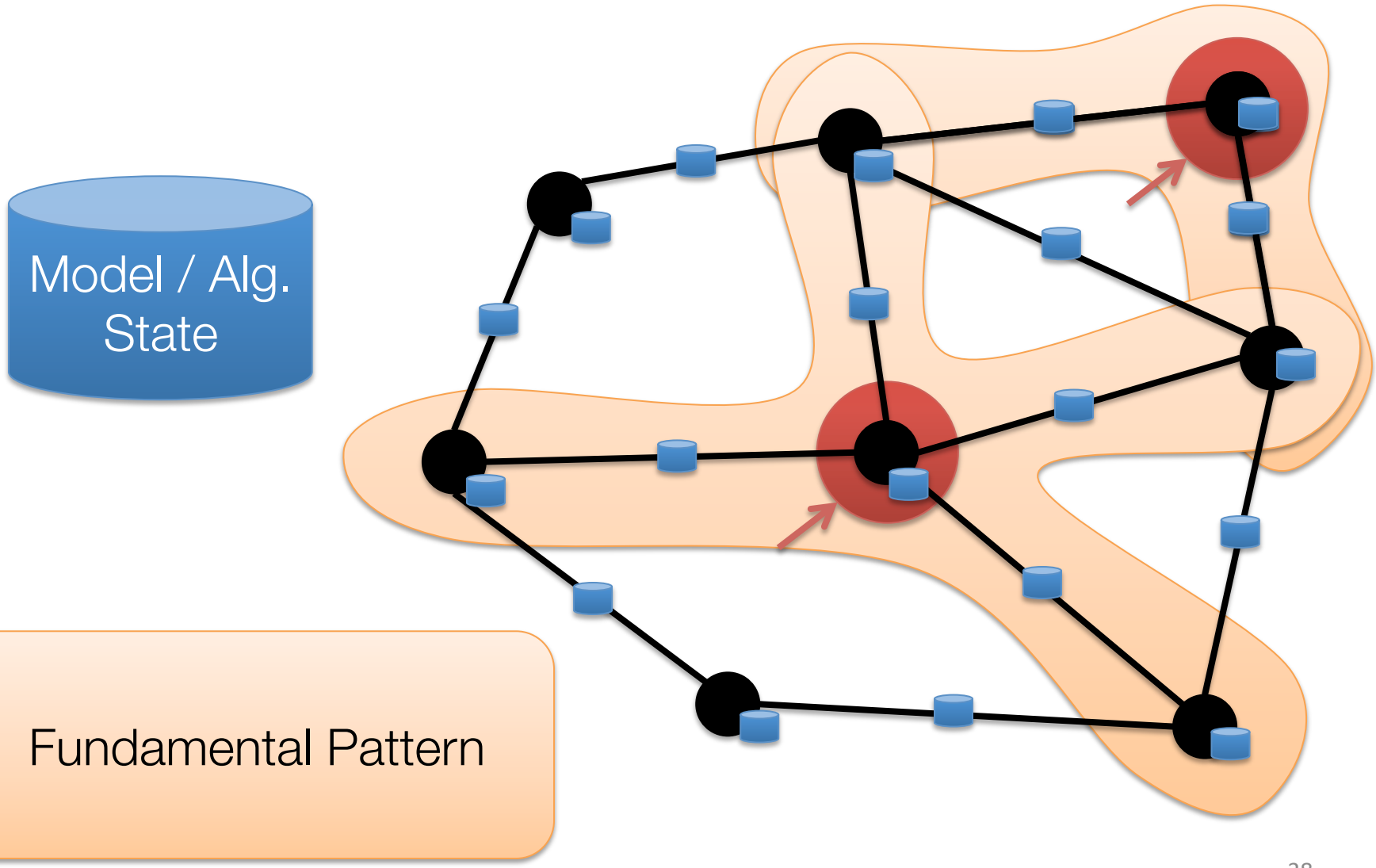
Putting it All Together



Many Other 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

The Graph-Parallel Pattern



Graph-Parallel Systems

Pregel

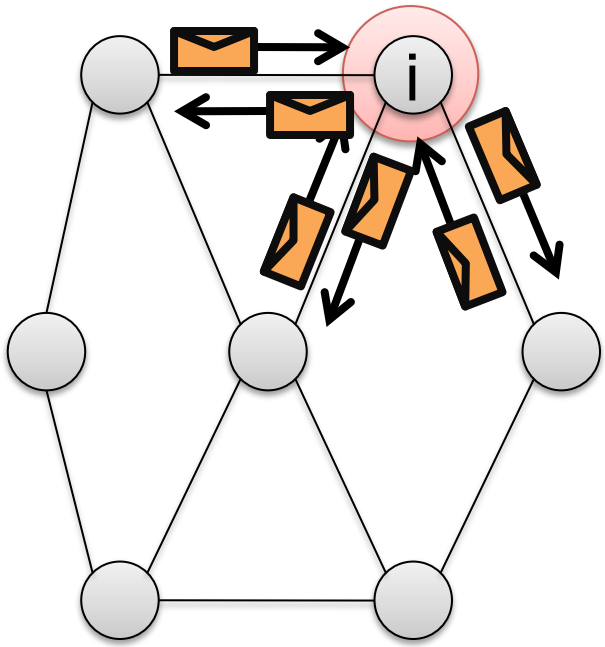
GraphLab

Expose specialized APIs to simplify graph programming.

The Vertex Program Abstraction

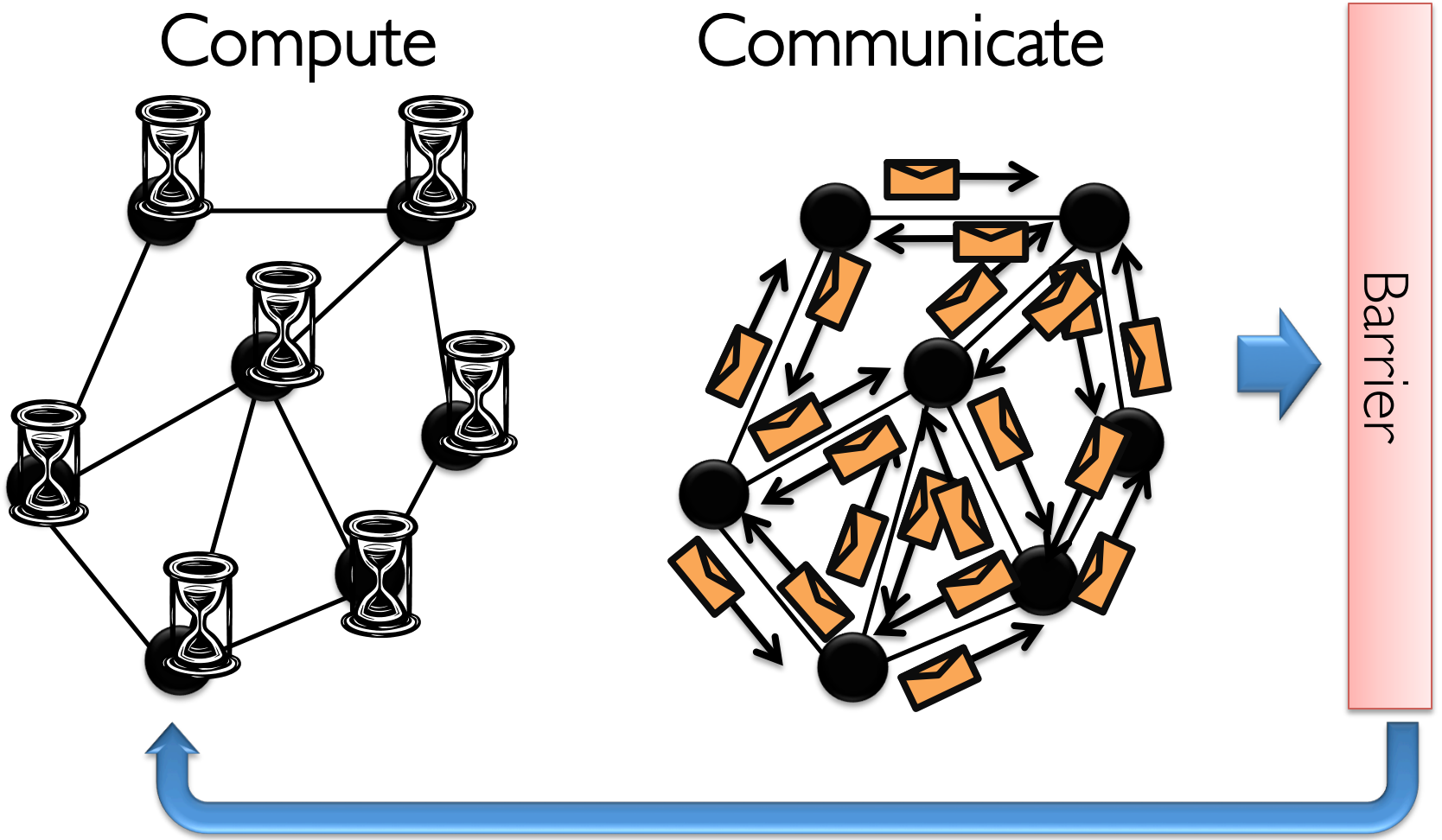
Vertex-Programs interact by sending **messages**.

```
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
```



Malewicz et al. [PODC'09, SIGMOD'10]

Iterative Bulk Synchronous Execution



Graph-Parallel Systems

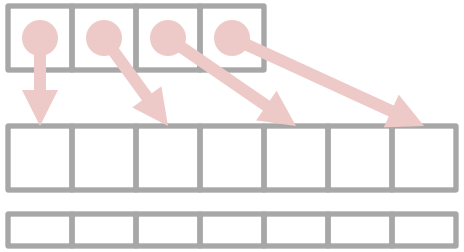
Pregel

GraphLab

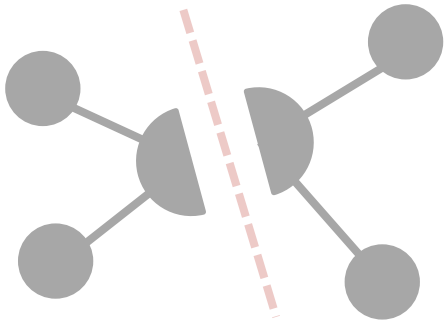
Exploit graph structure to achieve orders-of-magnitude performance gains over more general data-parallel systems.

Graph System Optimizations

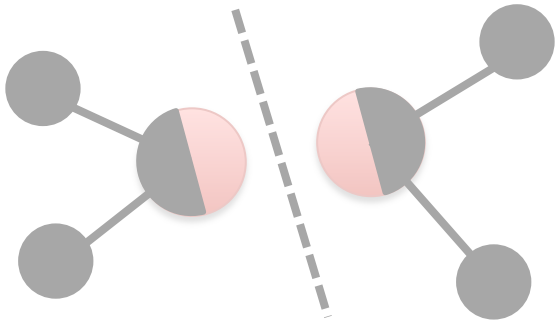
Specialized Data-Structures



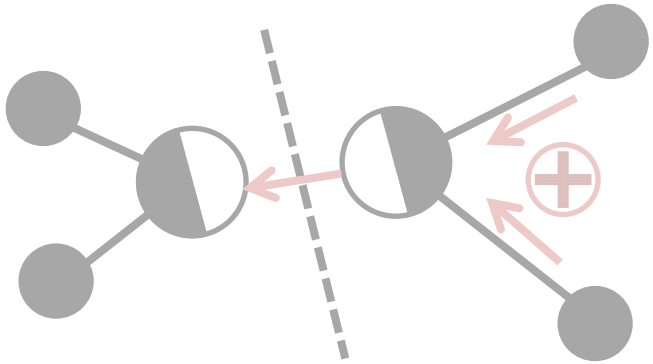
Vertex-Cuts Partitioning



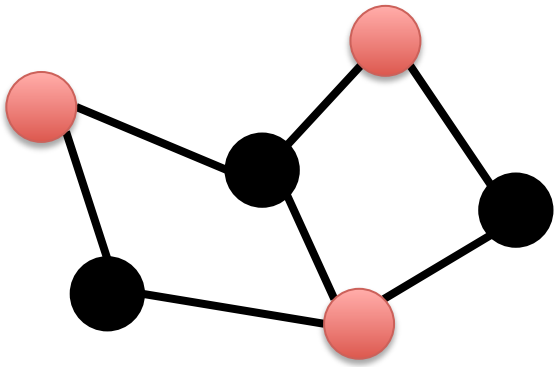
Remote Caching / Mirroring



Message Combiners



Active Set Tracking

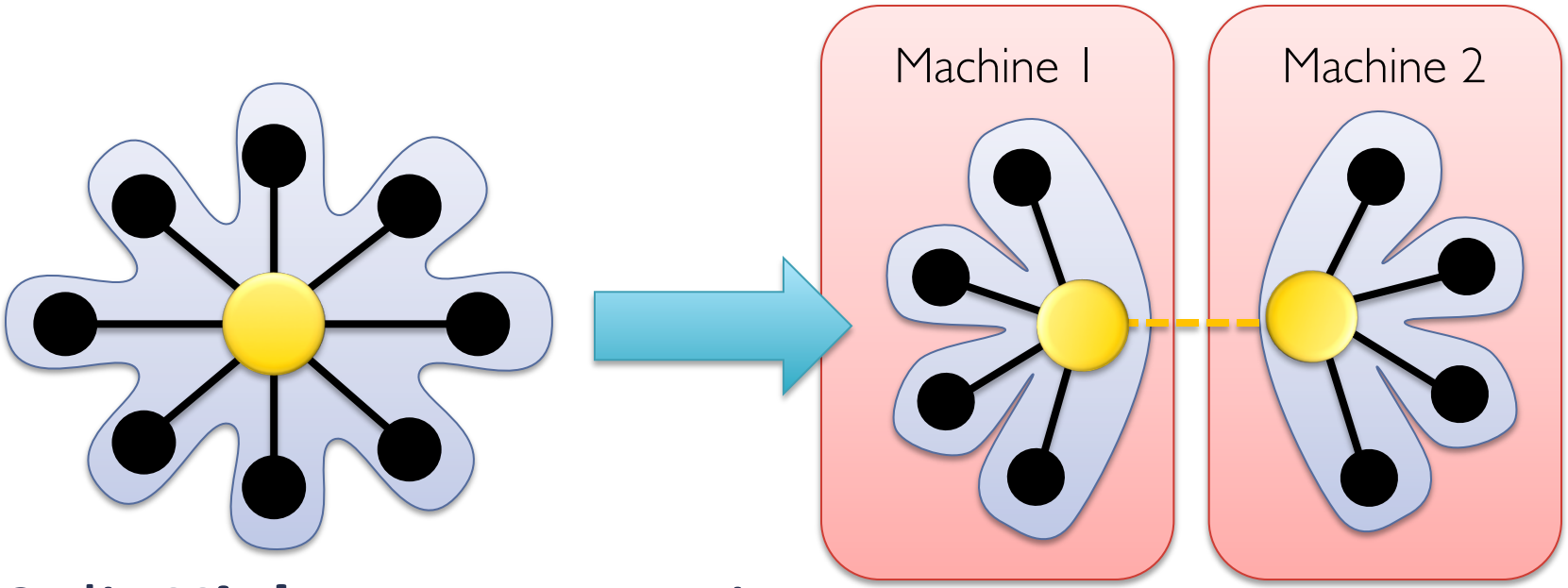


GraphLab

(PowerGraph, OSDI'12)

Program This

Run on This



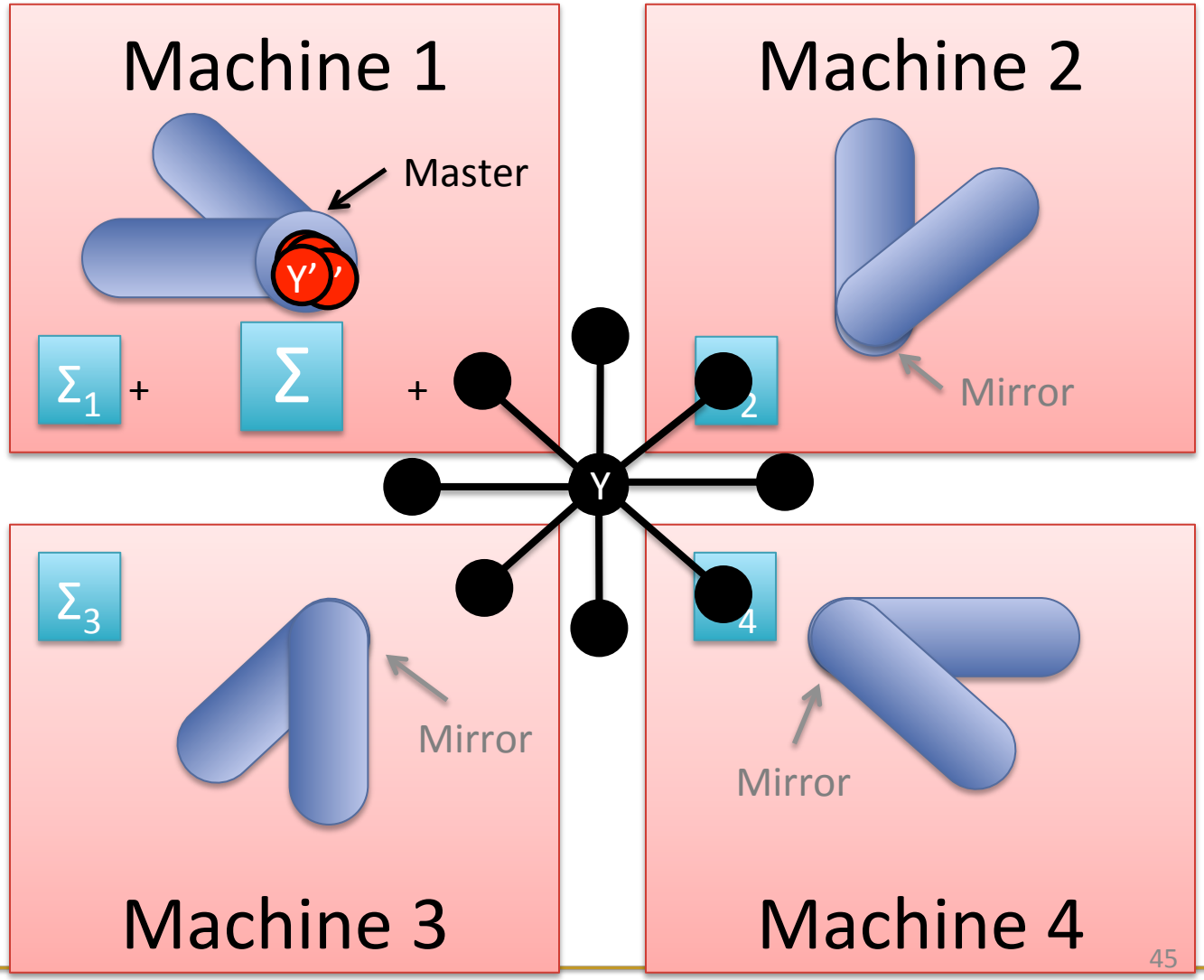
- Split **High-Degree** vertices
- **New Abstraction** → ***Equivalence*** *on Split Vertices*

GAS Decomposition

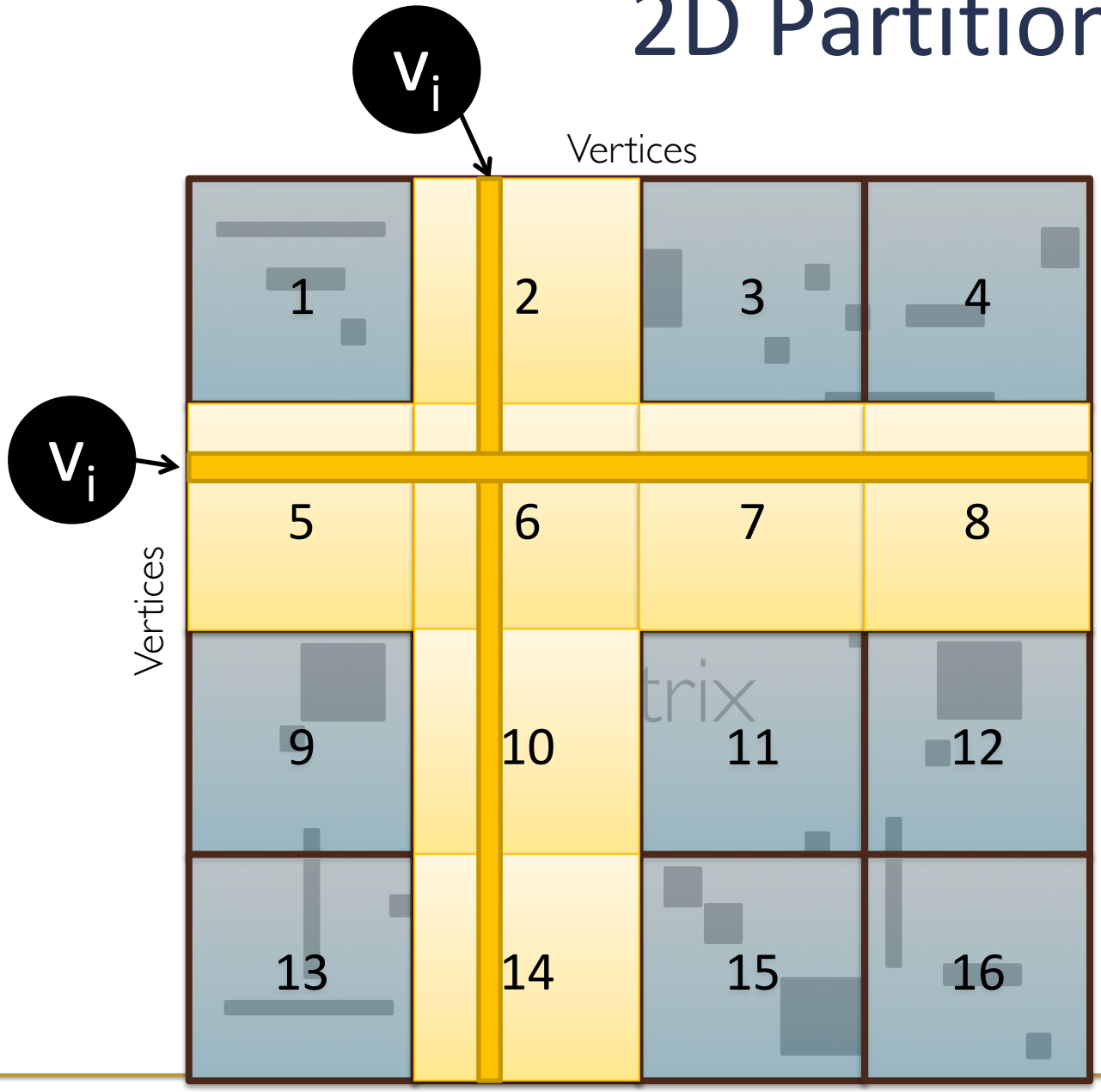
Gather

Apply

Scatter



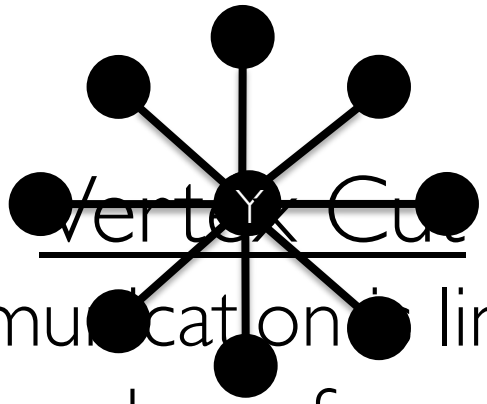
2D Partitioning



16 Machines

v_i only has neighbors on 7 machines

Minimizing Communication in PowerGraph



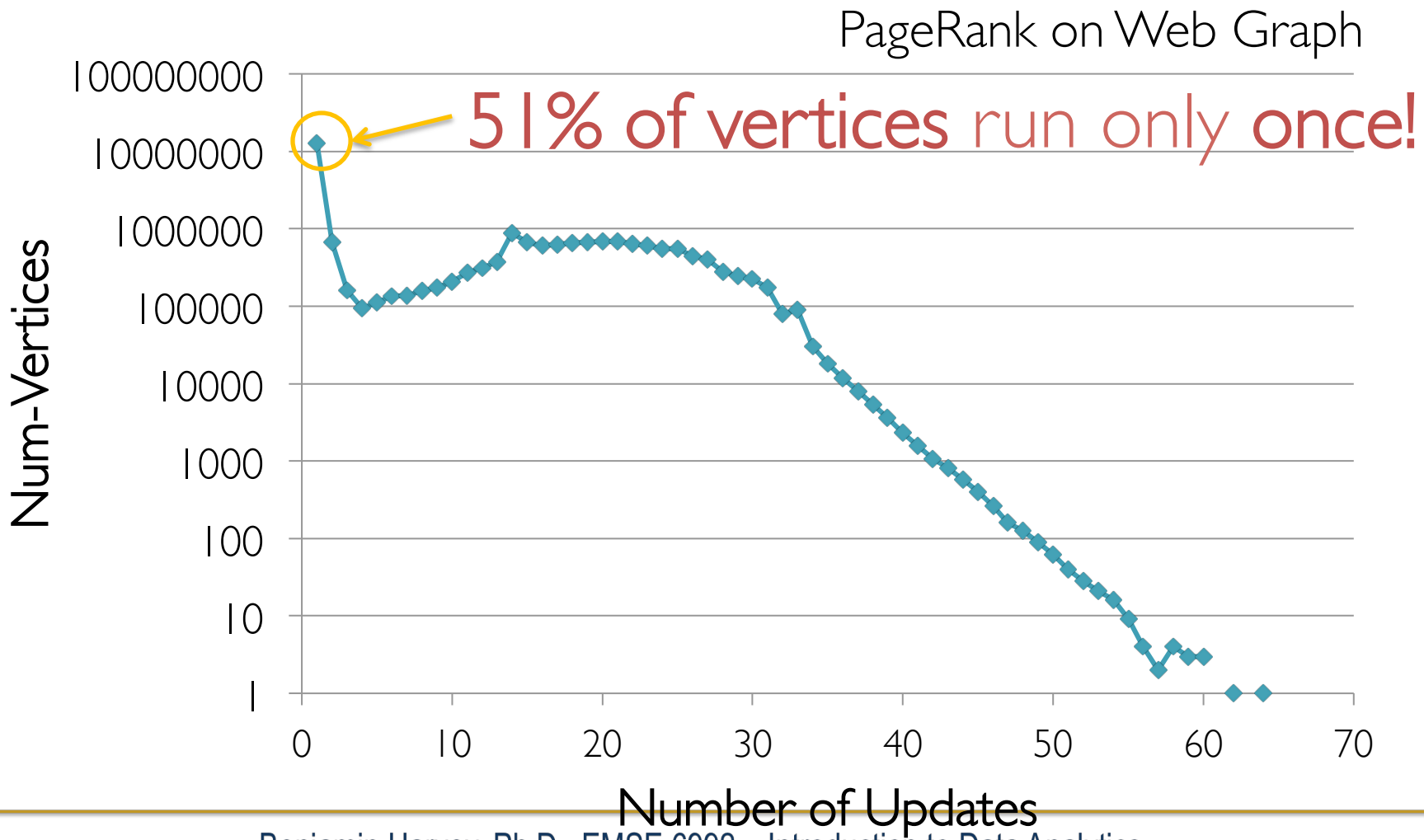
Vertex Cut

Communication is linear in the number of machines each vertex spans.

Total communication upper bound:

$$O\left(\#\text{vertices}\sqrt{\#\text{machines}}\right)$$

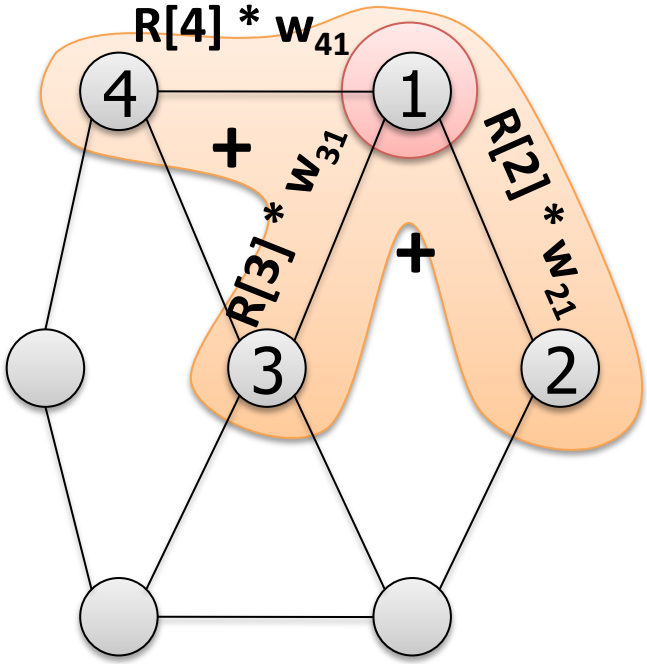
Shrinking Working Sets



The GraphLab (Pull) Abstraction

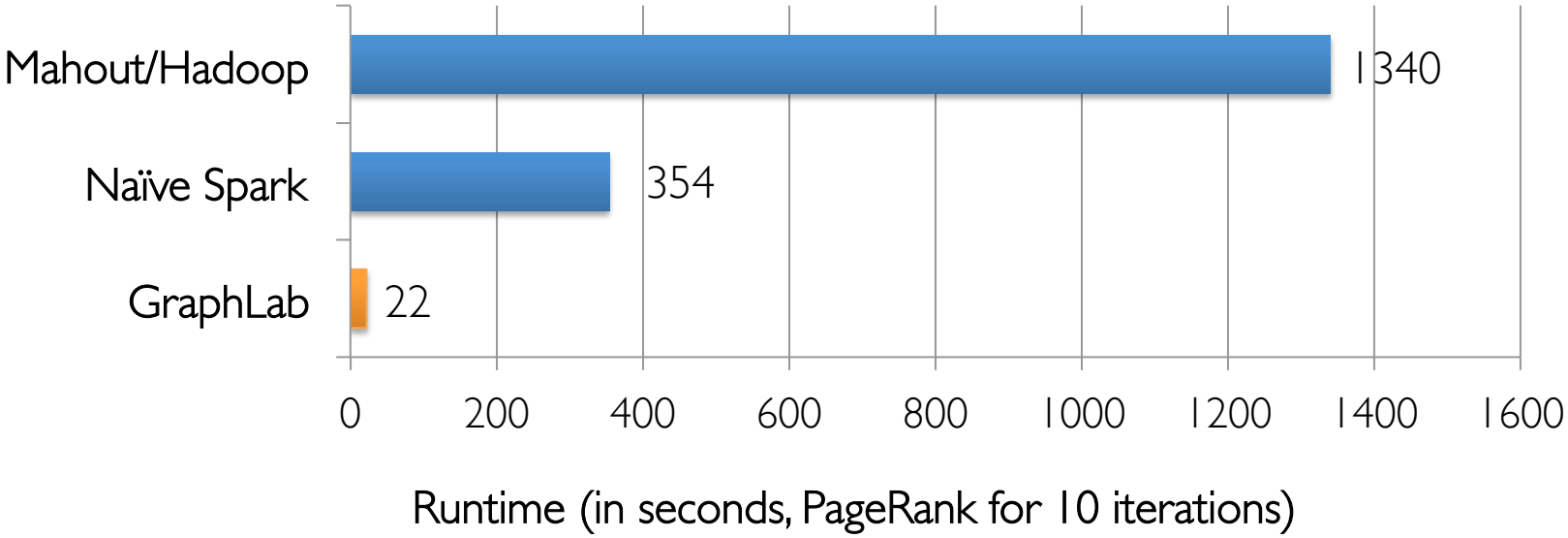
Vertex Programs directly **access** adjacent vertices and edges

```
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

  // Trigger neighbors to run again
  if R[i] not converged then
   signal nbrsOf(i) to be recomputed
```



Trigger computation *only* when necessary.

PageRank on the Live-Journal Graph



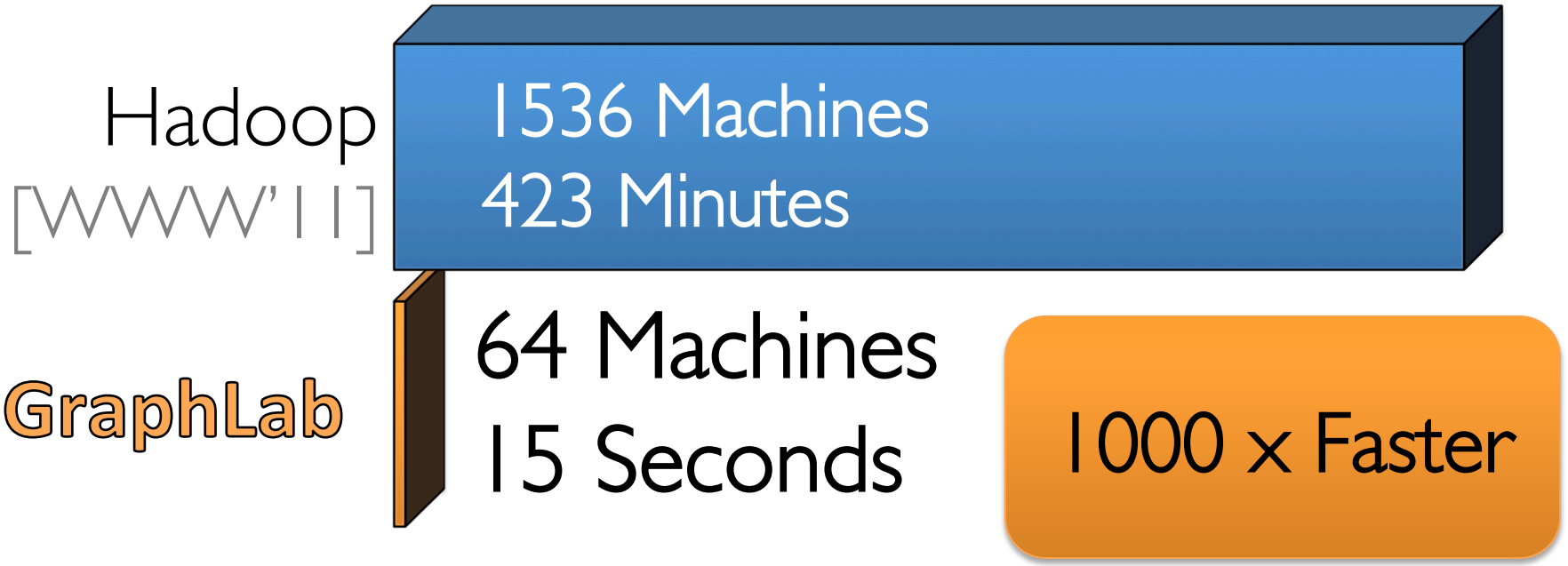
GraphLab is *60x faster* than Hadoop

GraphLab is *16x faster* than Spark

Triangle Counting on Twitter

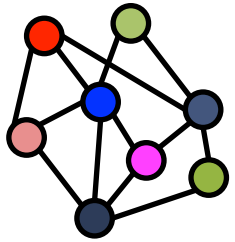
40M Users, 1.4 Billion Links

Counted: 34.8 Billion Triangles



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

PageRank



Tables

Raw Wikipedia

Text Table

Title	Body

Hyperlinks

PageRank

Top 20 Pages

Title	PR

Term-Doc Graph

Topic Model (LDA)

Word Topics

Word	Topic

Discussion Table

User	Disc.

Editor Graph

Community Detection

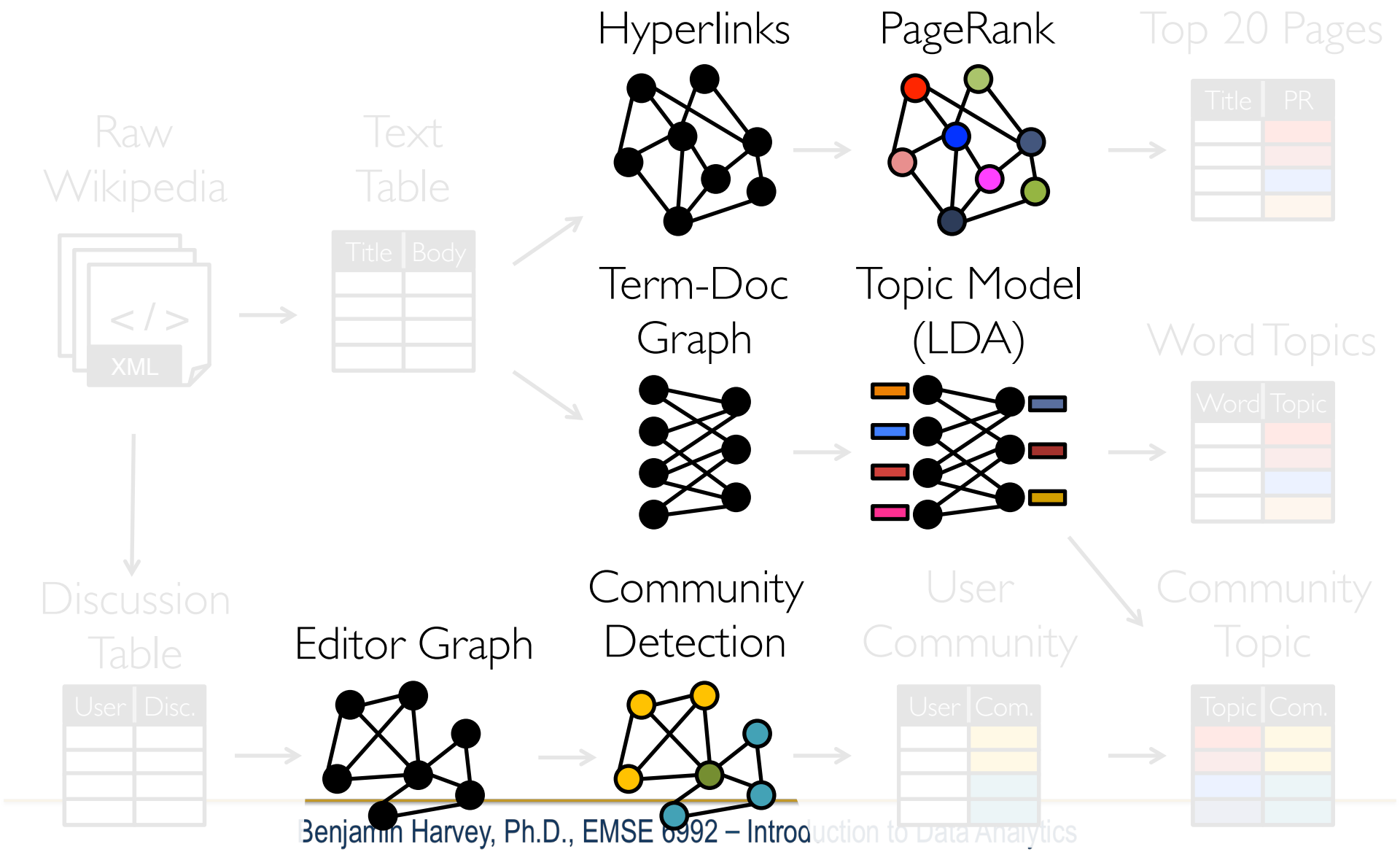
User Community

User	Com.

Community Topic

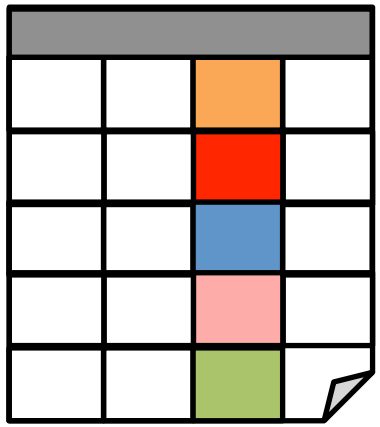
Topic	Com.

Graphs

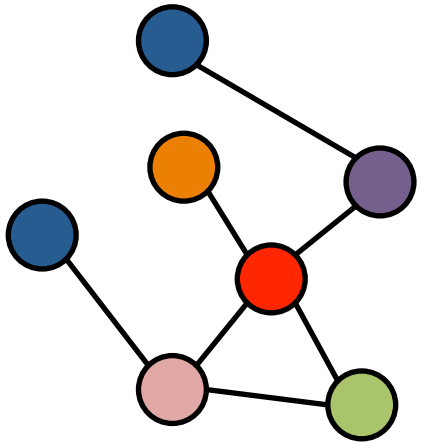

Hyperlinks
PageRank
Top 20 Pages
Raw Wikipedia
Text Table
Title
Body
PR
XML
Term-Doc Graph
Topic Model (LDA)
Word Topics
Word
Topic
Discussion Table
Community Detection
User Community
Community Topic
Editor Graph
User
Disc.
User
Com.
Topic
Com.

Separate Systems

Tables

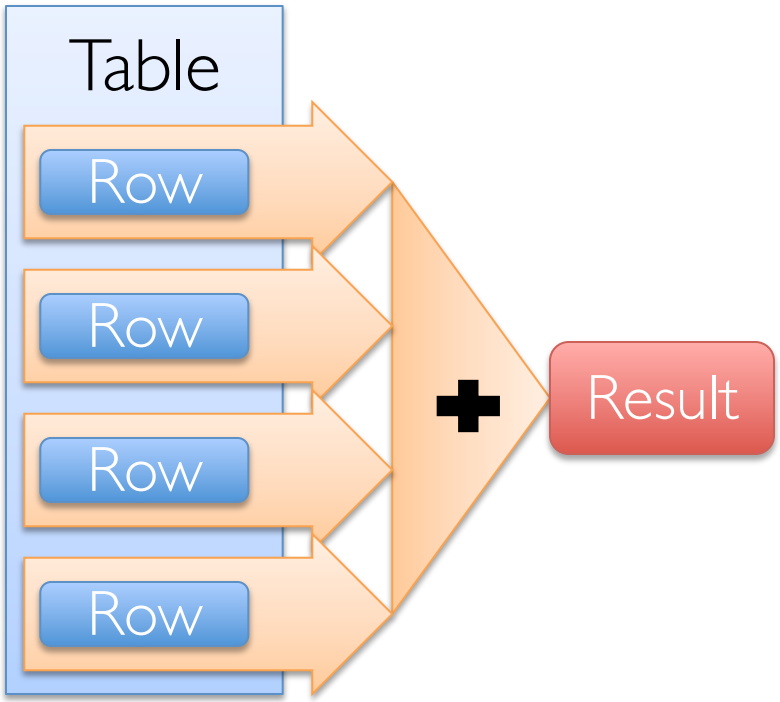


Graphs

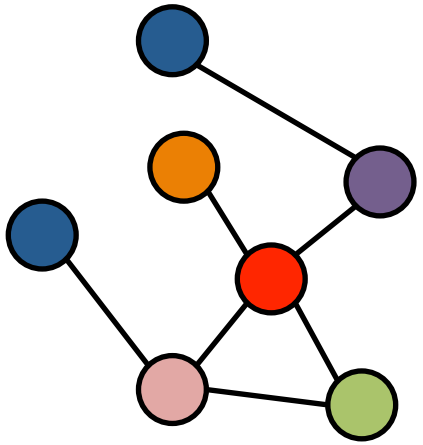


Separate Systems

Dataflow Systems

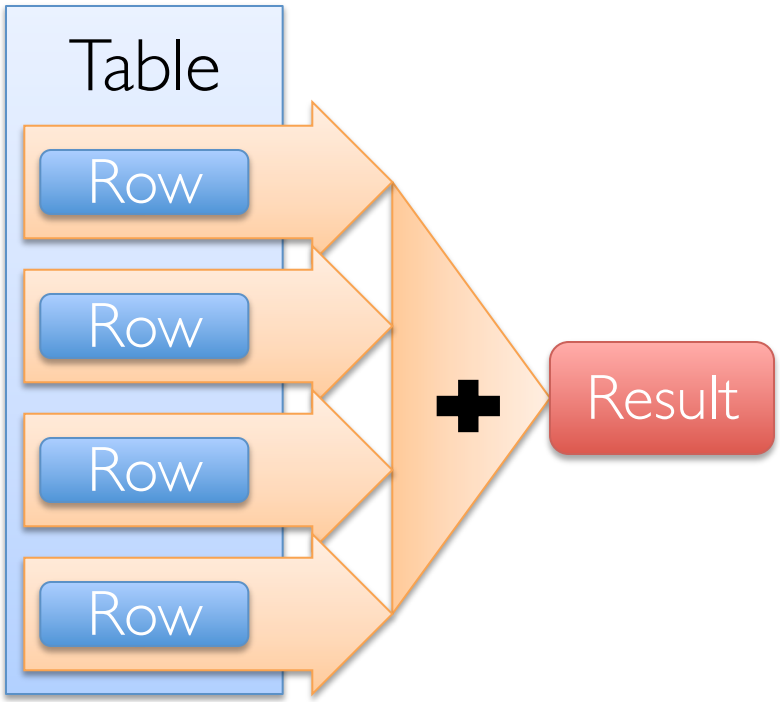


Graphs

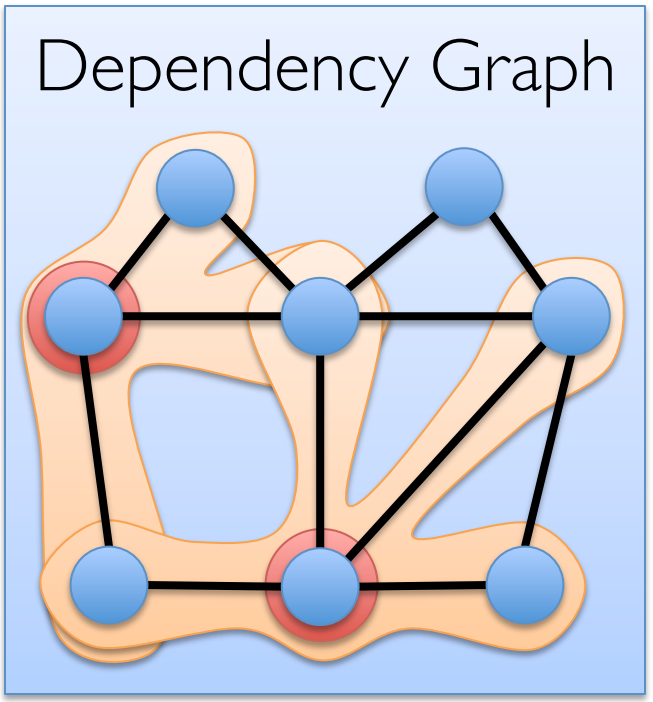


Separate Systems

Dataflow Systems



Graph Systems



Separate systems
for each view can be
difficult to use and inefficient

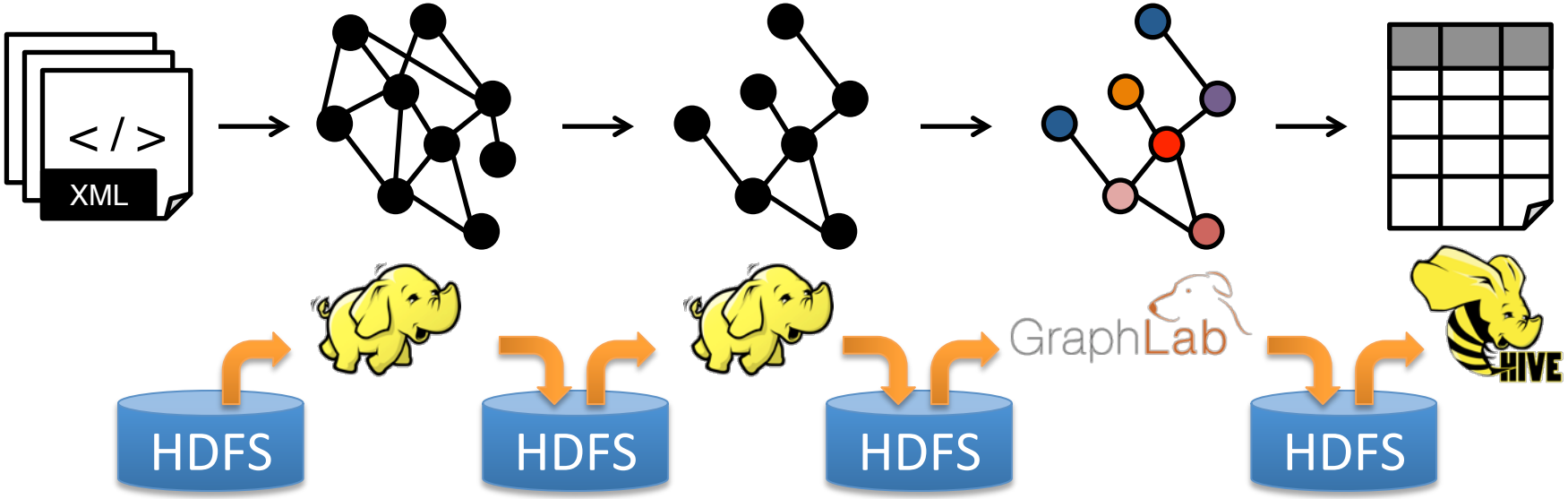
Difficult to Program and Use

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

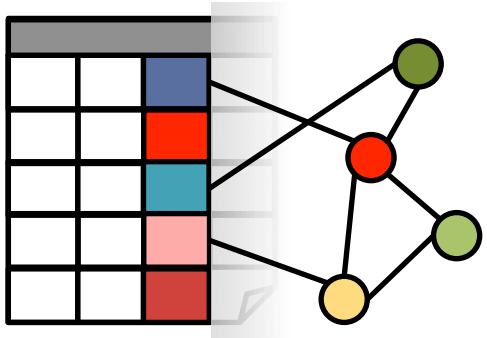


Limited reuse internal data-structures across stages

The GraphX Unified Approach

New API

Blurs the distinction between Tables and Graphs



New System

Combines Data-Parallel Graph-Parallel Systems

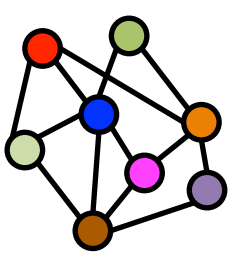
APACHE GIRAPH

Spark

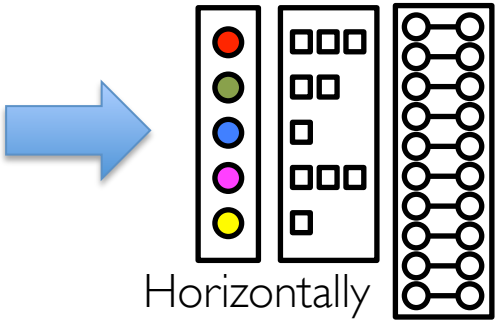
GraphLab

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

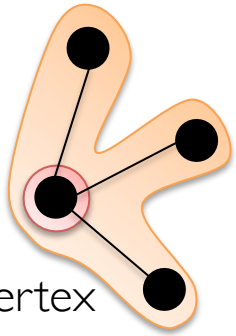
Representation



Distributed Graphs

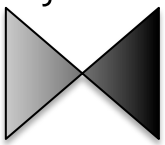


Horizontally Partitioned Tables



Vertex Programs

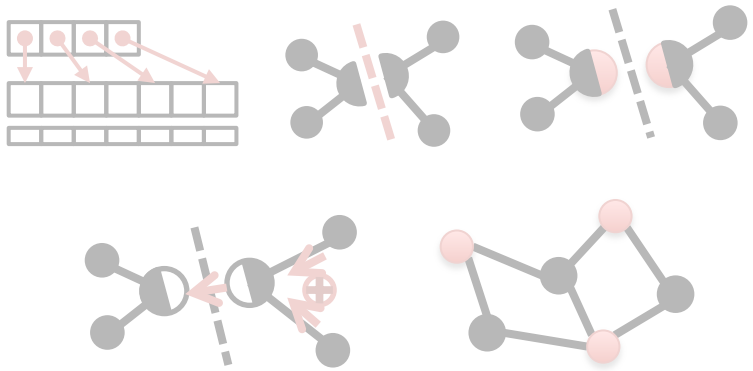
Join



Dataflow Operators

Optimizations

Advances in Graph Processing Systems

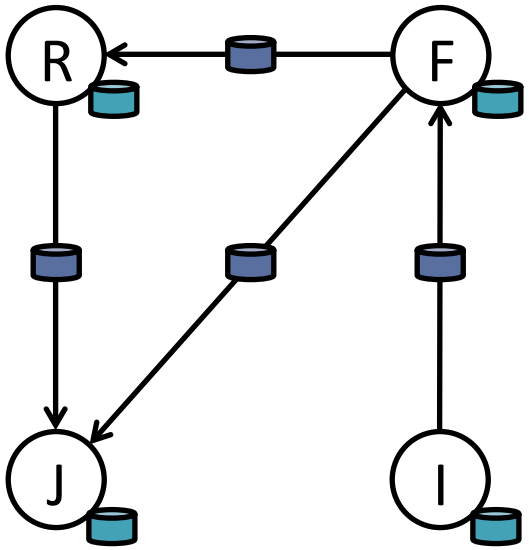


Distributed Join Optimization

Materialized View Maintenance

View a 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

Spark 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 (Scala)

```
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]
}
```

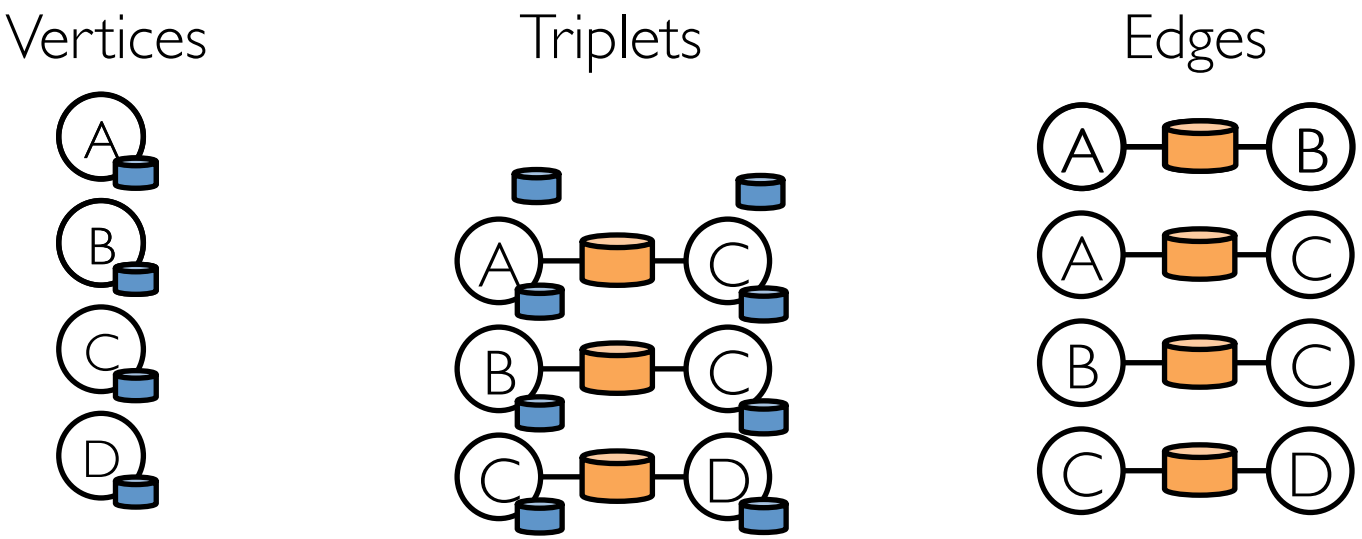
Graph Operators (Scala)

```
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]
}
```

capture the *Gather-Scatter pattern* from specialized graph-processing systems

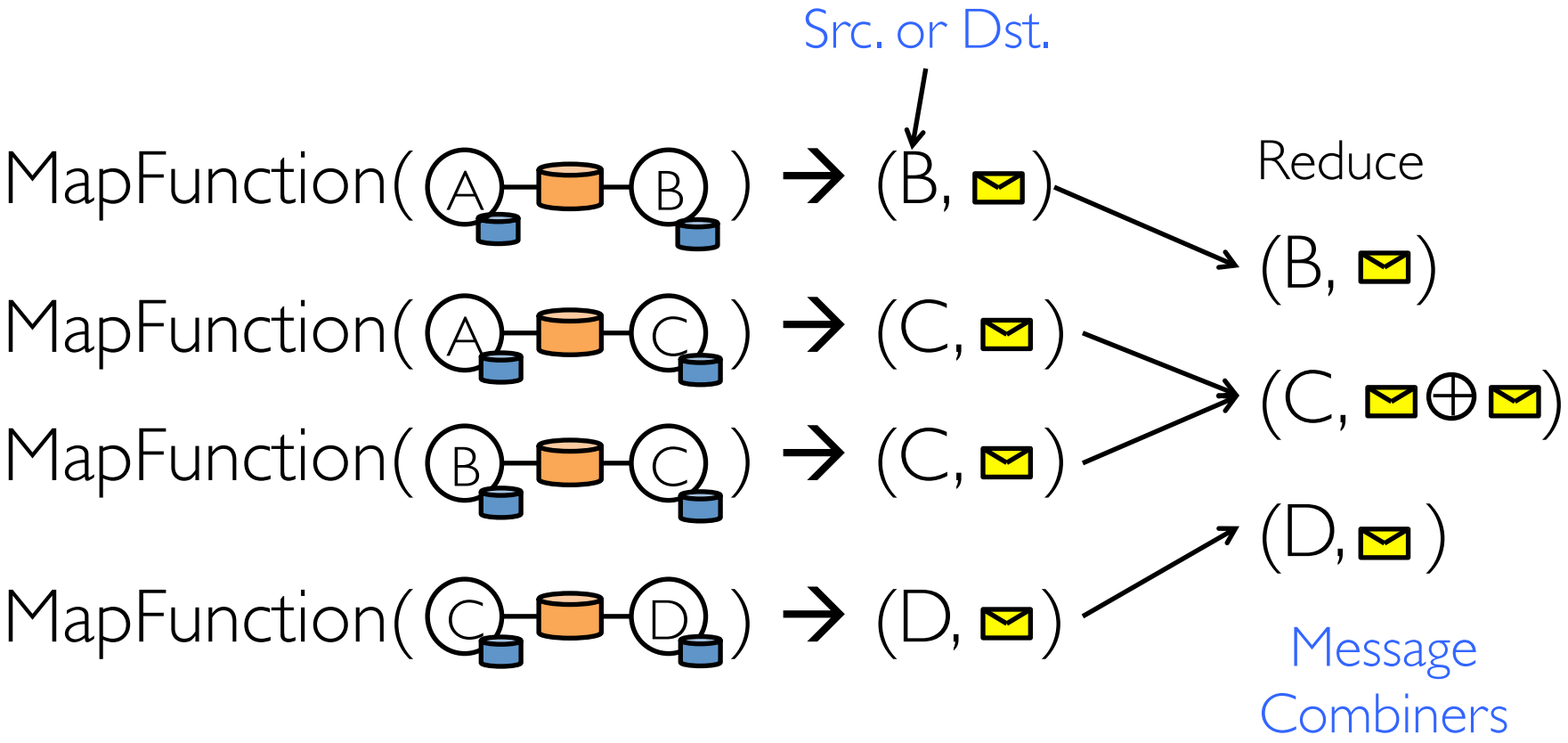
Triplets Join Vertices and Edges

The *triplets* operator joins vertices and edges:



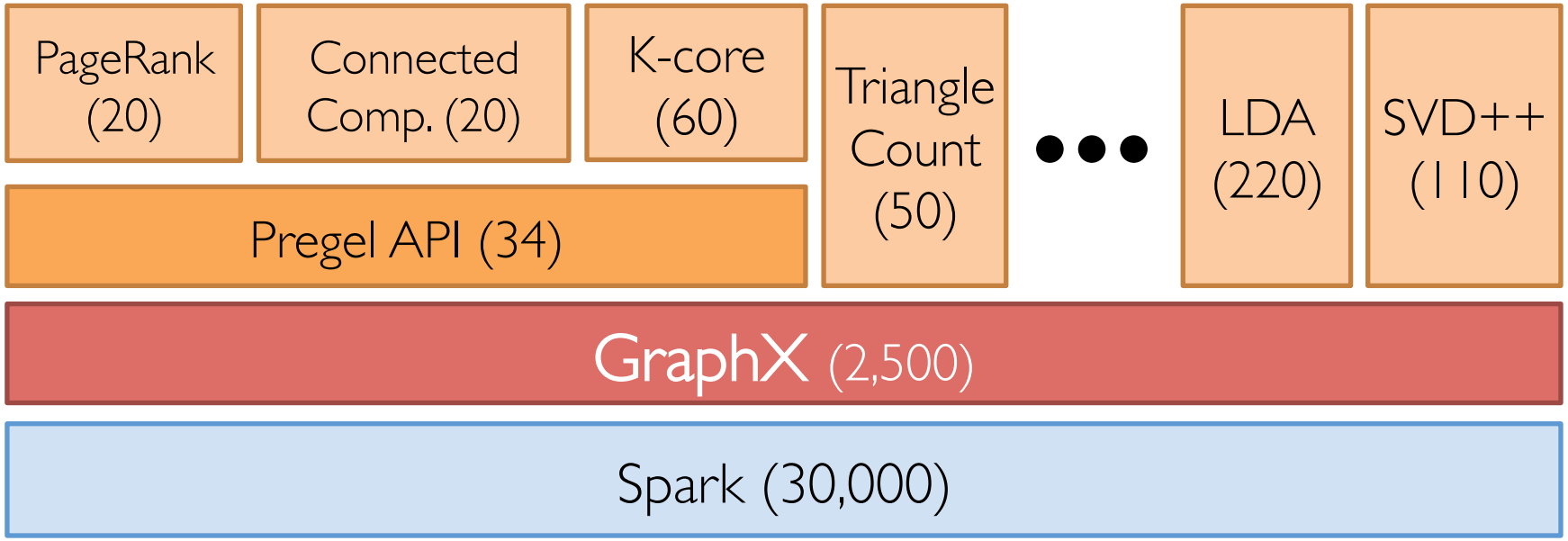
Map-Reduce Triplets

Map-Reduce triplets collects information about the neighborhood of each vertex:



Using these basic GraphX operators we implemented Pregel and GraphLab in under 50 lines of code!

The GraphX Stack (Lines of Code)



Some algorithms are more naturally expressed using the GraphX primitive operators

We express *enhanced* Pregel and GraphLab abstractions using the GraphX operators in less than 50 lines of code!

Enhanced Pregel in GraphX

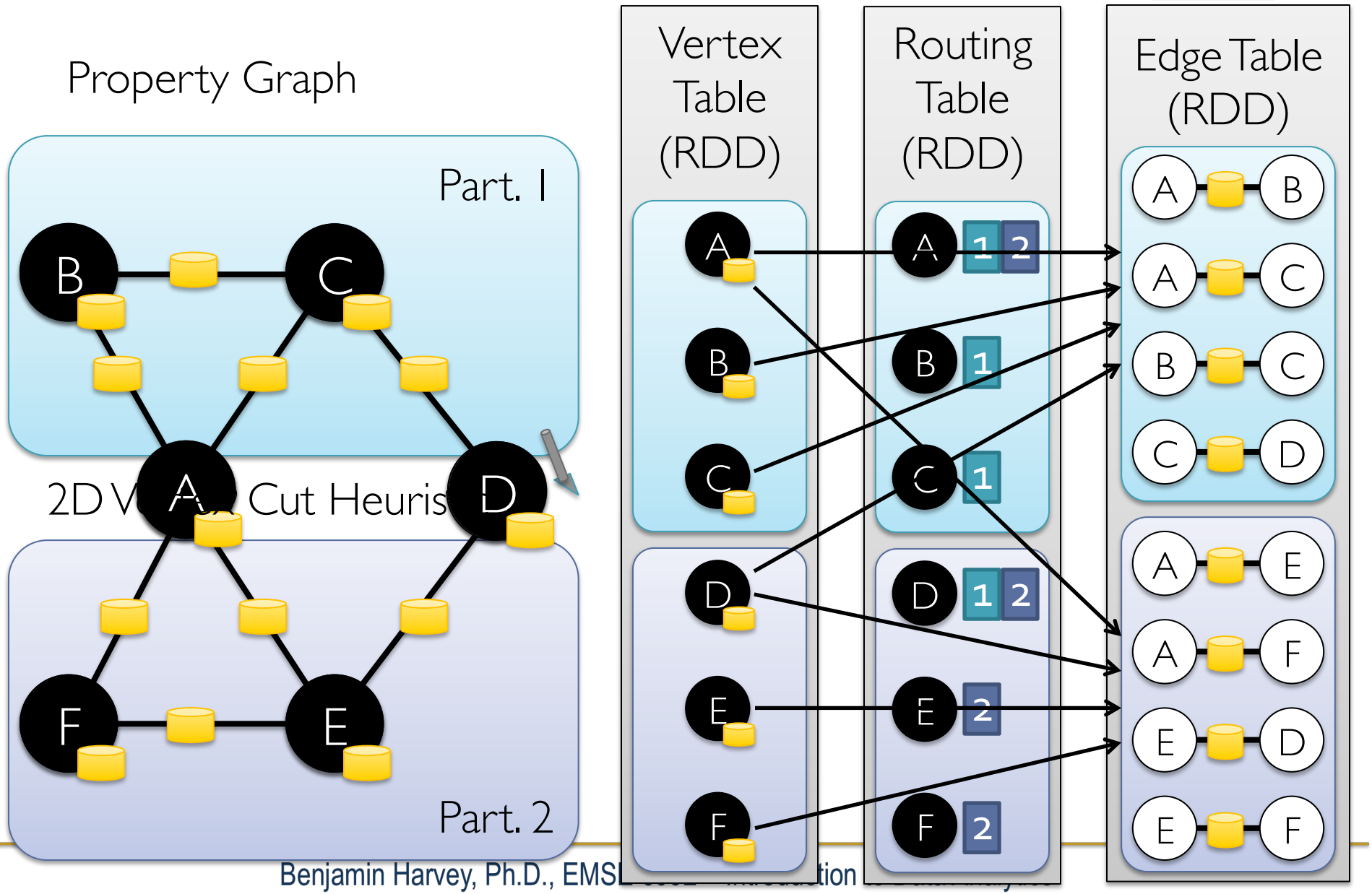
```
pregelPR(i, messageSum):
   // Receive all the messages
   total = 0
   foreach( msg in messageList) :
     total = total + msg

   // Update the rank of this vertex
   R[i] = 0.15 + total
combineMsg(a, b):
   // Compute sum of two messages
   return a + b
sendMsg(i→j, R[i], R[j], E[i,j]):
   // Compute single message
   return msg(R[i]/E[i,j])
```

Require Message Combiners

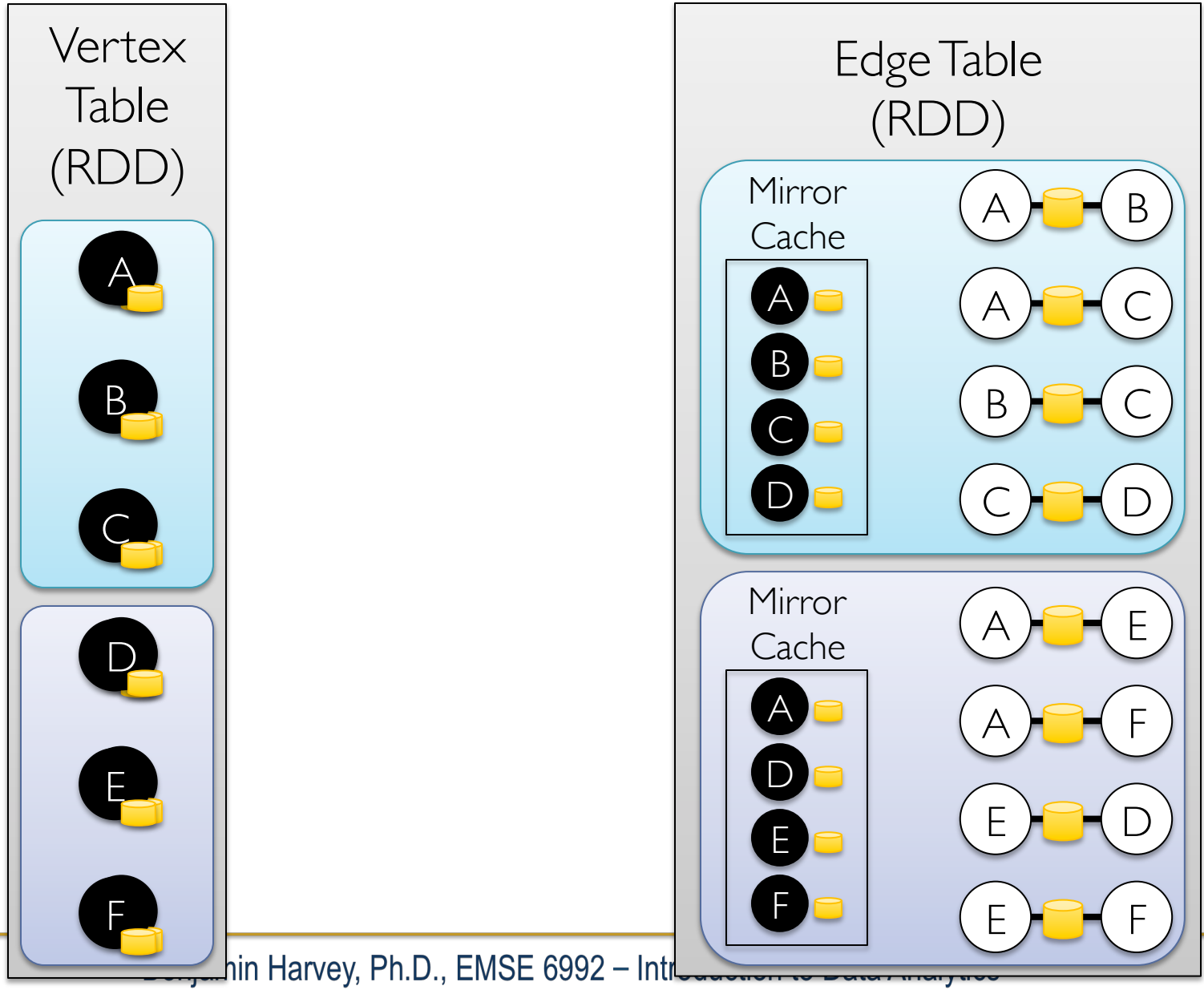
Remove Message Computation from the Vertex Program

GraphX System Design

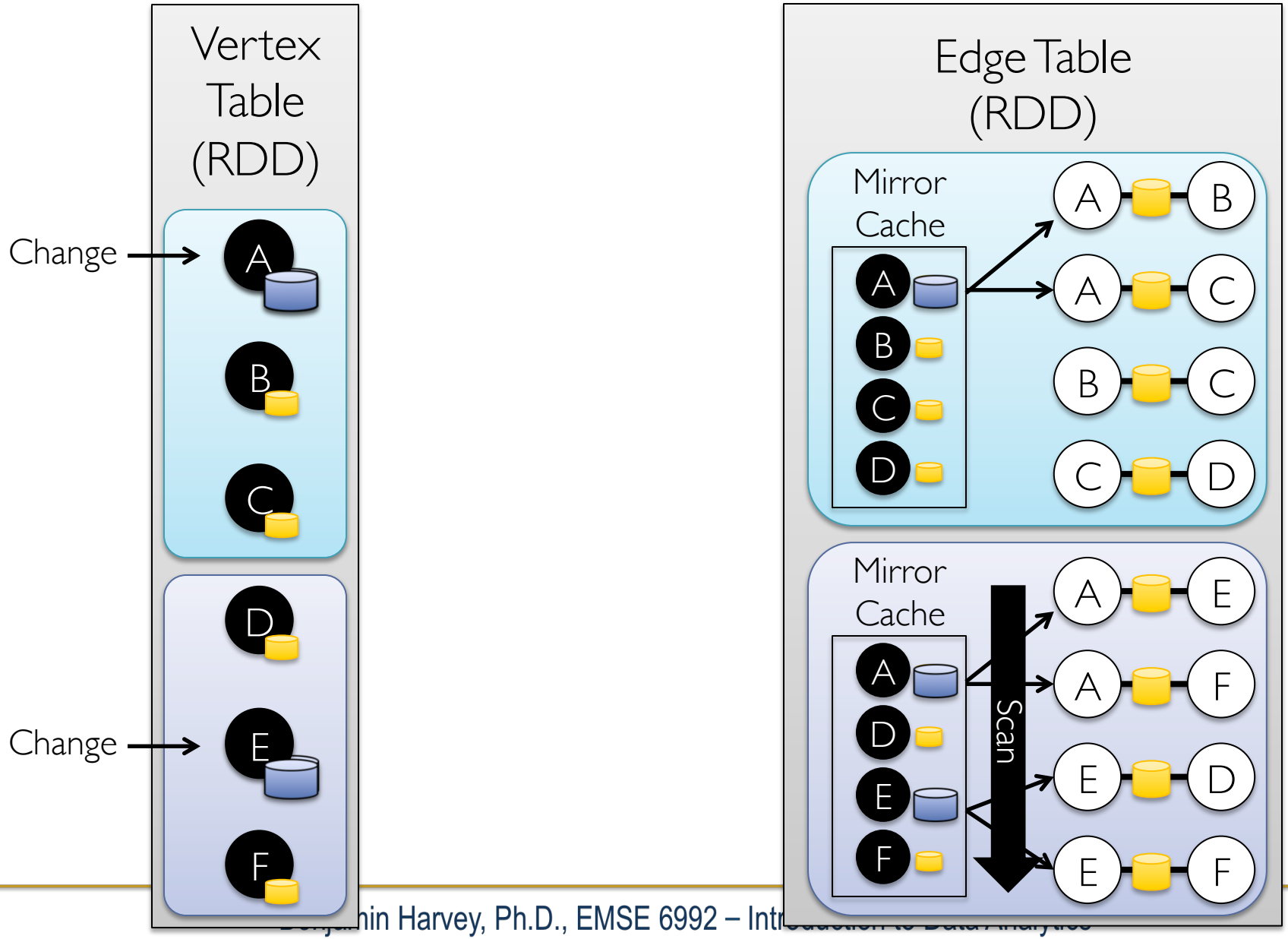
Distributed Graphs as Tables (RDDs)



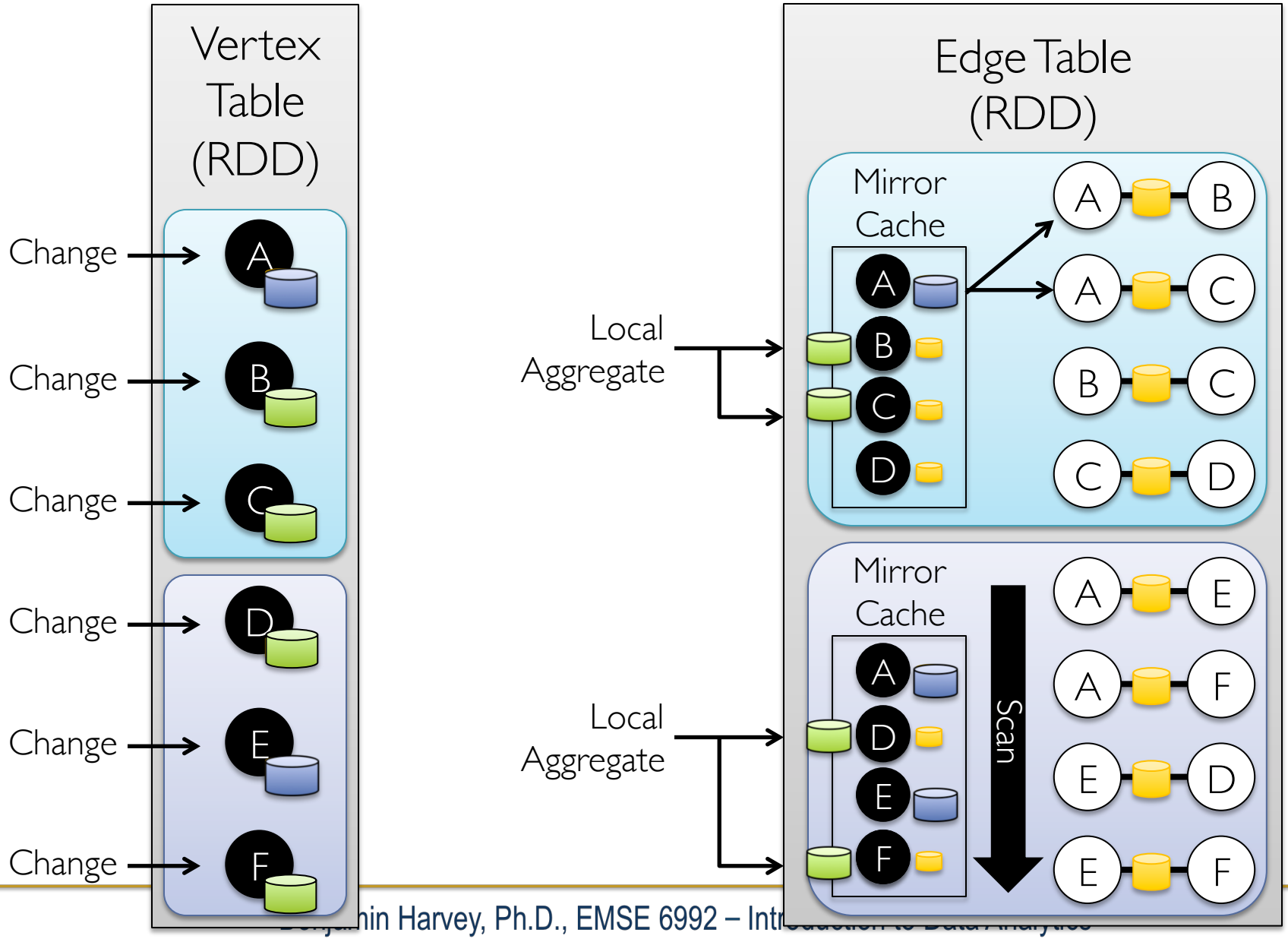
Caching for Iterative mrTriplets



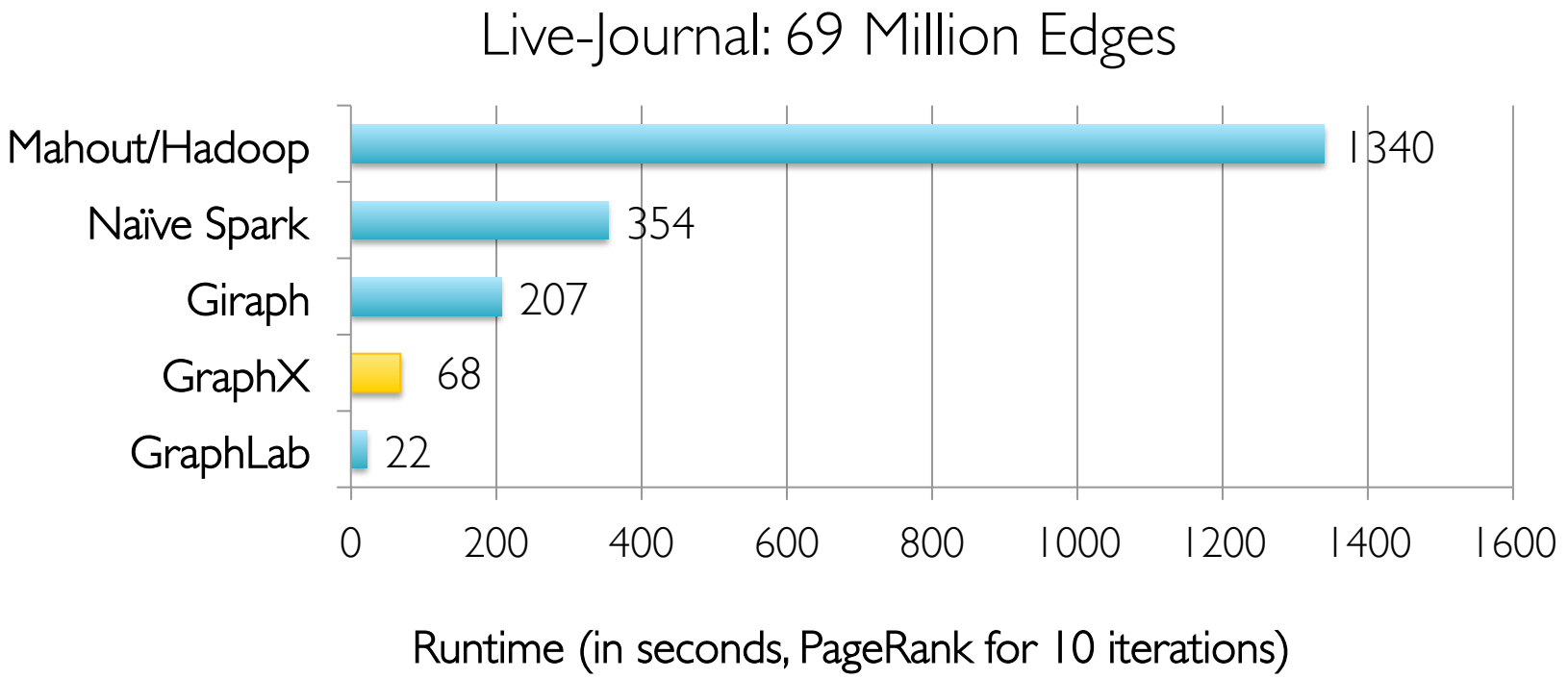
Incremental Updates for Iterative mrTriplets



Aggregation for Iterative mrTriplets

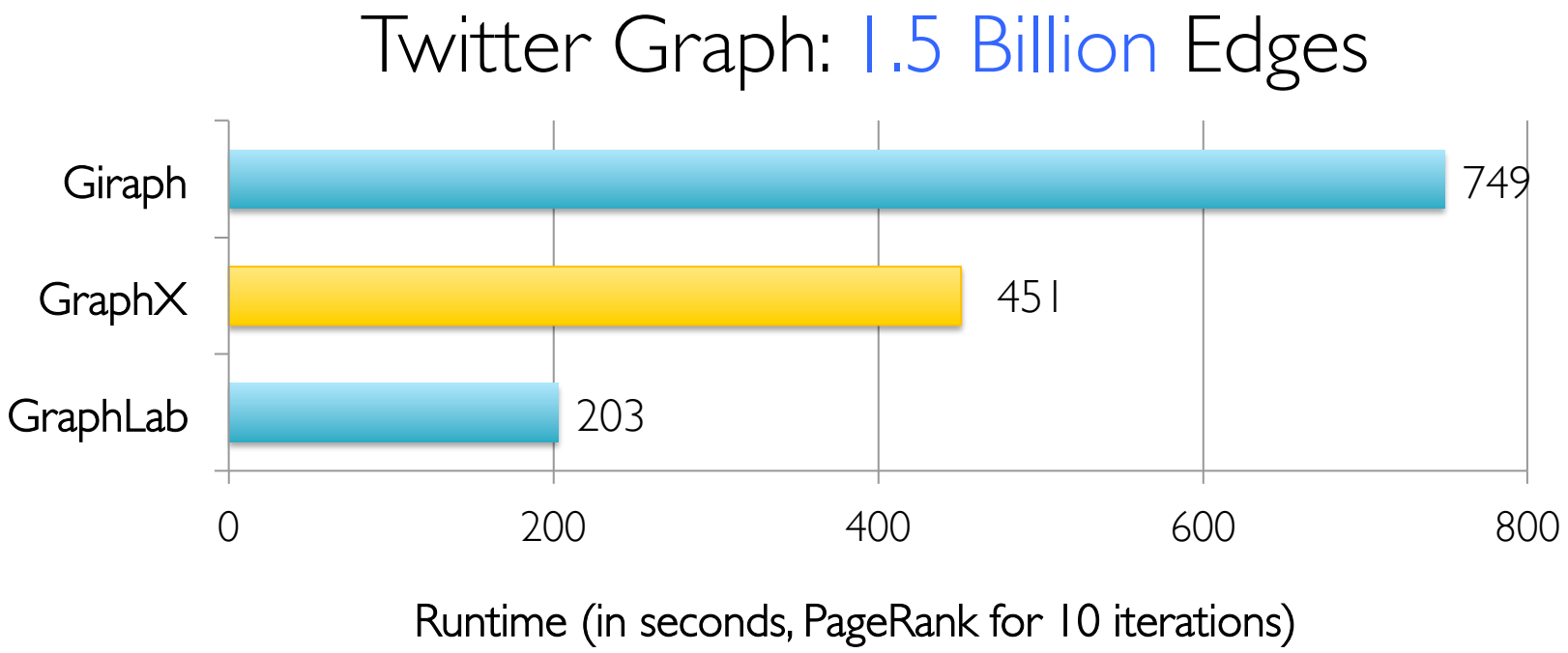


Performance Comparisons



GraphX is roughly 3x slower than GraphLab

GraphX scales to larger graphs



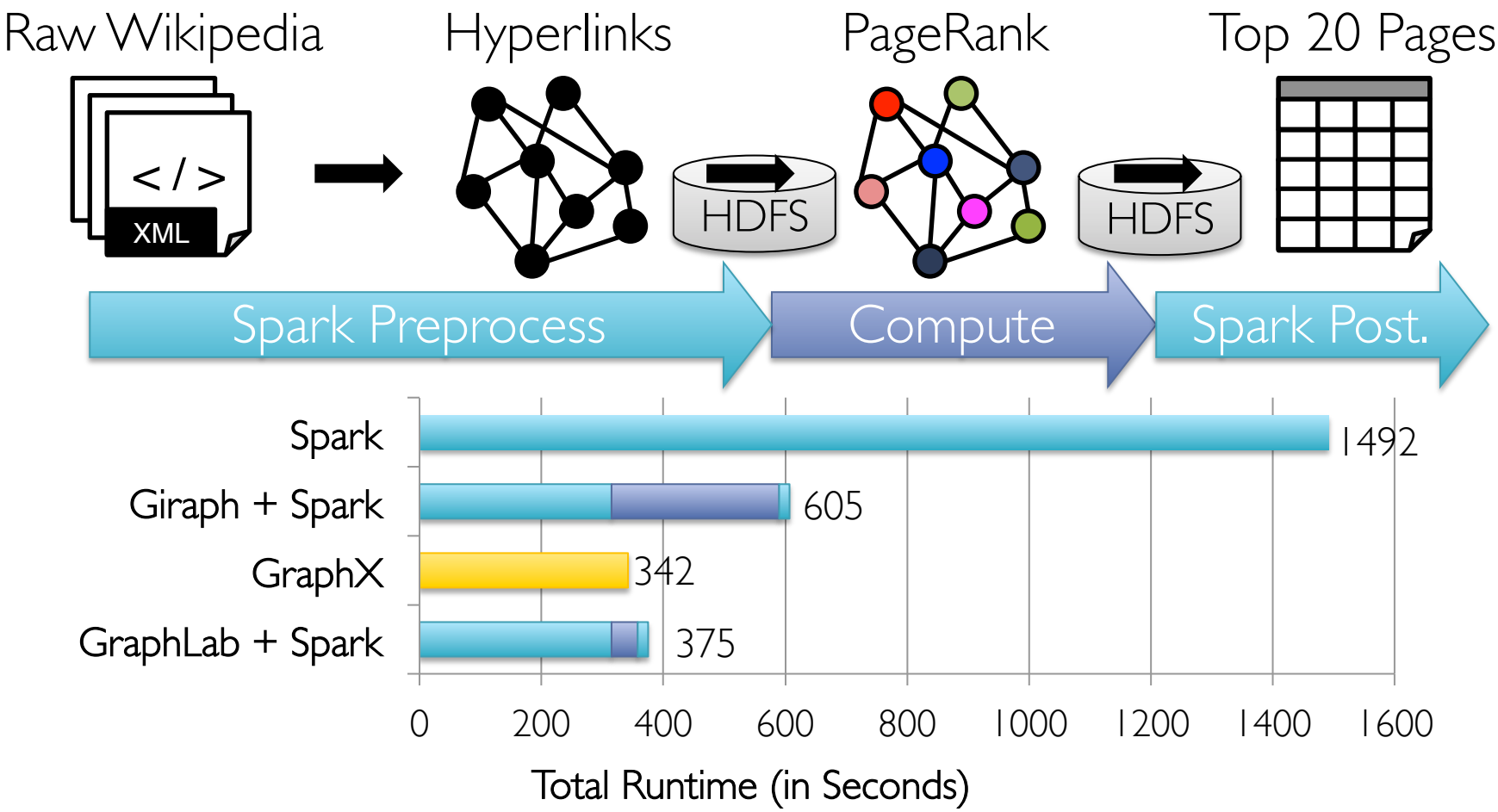
GraphX is roughly 2x slower than GraphLab

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

PageRank is just one stage….

What about a pipeline?

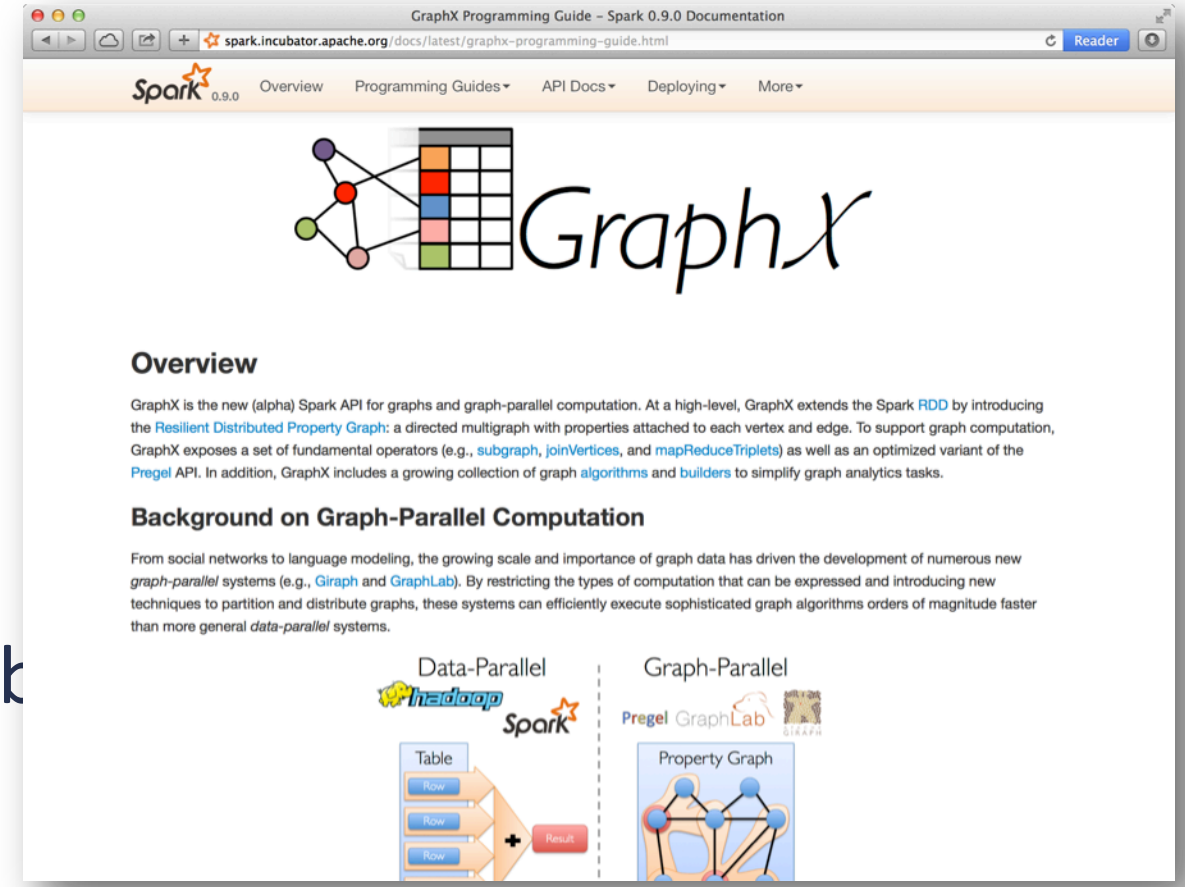
A Small Pipeline in GraphX



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

Open Source Project

- Alpha release since Spark 0.9



- Contrib
-

Graph Processing Systems

- Apache Giraph: java Pregel implementation
- GraphLab.org: C++ GraphLab implementation
- NetworkX: python API for small gaphs
- GraphLab Create: commercial GraphLab python framework for large graphs and ML
- Gephi: graph visualization framework

Graph Database Technologies

- Property graph data-model for storing and retrieving graph structured data.
- Neo4j: popular commercial graph database
- Titan: open-source distributed graph database

About Scala

- High-level language for the Java VM
  - Object-oriented + functional programming
- Statically typed
  - Comparable in speed to Java
  - But often no need to write types due to type inference
- Interoperates with Java
  - Can use any Java class, inherit from it, etc; can also call Scala code from Java

Quick Tour

- Declaring variables:
- `var x: Int = 7`
  `var x = 7  // type inferred`
- `val y = "hi"  // read-only`

- Java equivalent:
- `int x = 7;`
- `final String y = "hi";`

Functions:

```
def square(x: Int): Int = x*x

def min(a:Int, b:Int): Int = {
  if (a < b) a else b
}

def announce(text: String) {
  println(text)
}
```

Java equivalent:

```
int square(int x) {
  return x*x;
}

void announce(String text) {
  System.out.println(text);
}
```

Quick Tour

- Generic types:
- `var arr = new Array[Int](8)`

```
var lst = List(1, 2, 3)
// type of lst is List[Int]
```

- Java equivalent:
- `int[] arr = new int[8];`
-

```
List<Integer> lst =
  new ArrayList<Integer>();
lst.add(...)
```

Indexing:

```
arr(5) = 7

println(lst(5))
```

Java equivalent:

```
arr[5] = 7;

System.out.println(lst.get(5));
```

Quick Tour

- Processing collections with functional programming:

Function expression (closure)

```
val list = List(1, 2, 3)
list.foreach(x => println(x))  // prints 1, 2, 3
list.foreach(println)          // same
list.map(x => x + 2)  // => List(3, 4, 5)
list.map(_ + 2)       // same, with placeholder notation
list.filter(x => x % 2 == 1)  // => List(1, 3)
list.filter(_ % 2 == 1)       // => List(1, 3)
list.reduce((x, y) => x + y)  // => 6
list.reduce(_ + _)            // => 6
```

All of these leave the list unchanged (List is immutable)

Scala Closure Syntax

- `(x: Int) => x + 2   // full version`
- `x => x + 2   // type inferred`
- `_ + 2   // when each argument is used exactly once`
- `x => {   // when body is a block of code`

```
  val numberToAdd = 2
  x + numberToAdd
}
```

- `// If closure is too long, can always pass a function`

```
def addTwo(x: Int): Int = x + 2

list.map(addTwo)
```

Scala allows defining a "local function" inside another function

Other Collection Methods

- Scala collections provide many other functional methods; for example, Google for "Scala Seq"

Method on Seq[T]	Explanation
`map(f: T => U): Seq[U]`	Pass each element through f
`flatMap(f: T => Seq[U]): Seq[U]`	One-to-many map
`filter(f: T => Boolean): Seq[T]`	Keep elements passing f
`exists(f: T => Boolean): Boolean`	True if one element passes
`forall(f: T => Boolean): Boolean`	True if all elements pass
`reduce(f: (T, T) => T): T`	Merge elements using f
`groupBy(f: T => K): Map[K,List[T]]`	Group elements by f(element)
`sortBy(f: T => K): Seq[T]`	Sort elements by f(element)
...	