Scalable Platforms for Graph Analytics and Collaborative Data Science

Amol Deshpande

Associate Professor

Department of Computer Science and UMIACS

University of Maryland at College Park

Joint work with many students and collaborators

These slides at: http://ter.ps/a37

Big Data

- Explosion of data, in pretty much every domain
  - Sensing devices and sensor networks (IoT) that can monitor everything from temperature to pollution to vital signs 24/7
  - Increasingly sophisticated smart phones
  - Internet, social networks making it very easy to publish data
  - Scientific experiments and simulations
  - Many aspects of life being turned into data ("dataification")
- "Big Data" (= extracting knowledge and insights from data) becoming fundamental
  - Science, business, politics -- largely driven by data and analytics
  - Many others (Education, Social Good) are slowly being

Four V's of Big Data

- Big data not just about "Volume"
  - Large scale of data certainly poses many problems
  - But most datasets are pretty small (10GB-500GB)…
- Variety and heterogeneity in both data and applications
  - Text, networks, time series, nested/hierarchical, multimedia, …
  - Increasingly complex and specialized analysis tasks
- Velocity
  - Data generated at very high rates and often needs to be processed in real time
- Veracity
  - What/who to trust? How to reason about data quality issues?
  - Easy to draw wrong statistical conclusions from large datasets
  - Issues becoming more important with increasing automation…

Focus of My Research Group at UMD

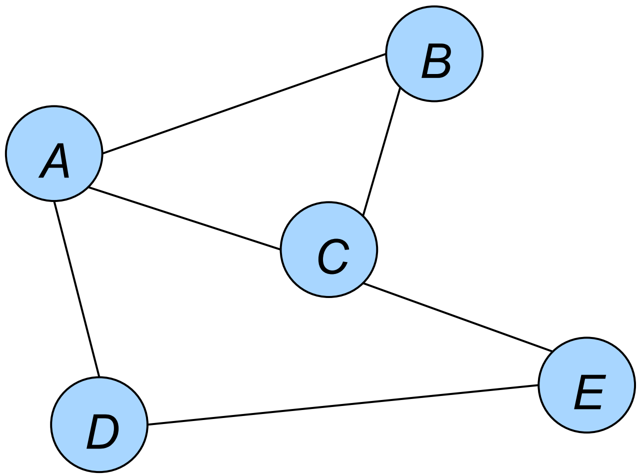
- Building data management systems to address challenges in managing and analyzing big data by..
  - Designing intuitive, formal, and declarative abstractions to empower users, and
  - Developing scalable platforms and algorithms to support those abstractions over large volumes of data
- Major research thrusts over the last 10 years
  - Uncertain and probabilistic data management
  - Graph data management
  - Data management in the cloud
  - Collaborative data analytics
  - Query processing and optimization

Outline

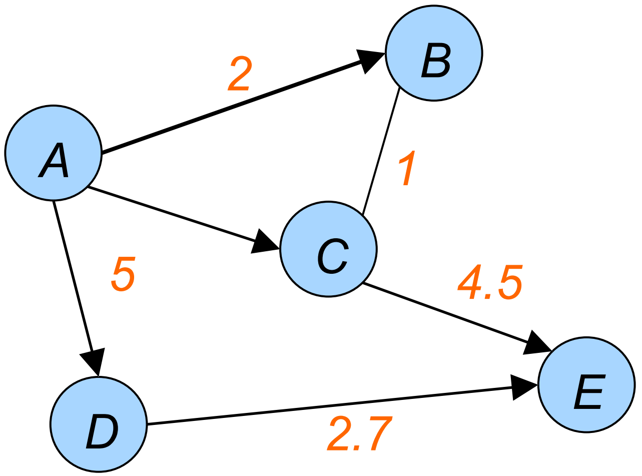
- Graph Data Management
  - A Framework for Distributed Graph Analytics
- DataHub: A platform for collaborative data science

Background: Graphs

- A *graph* captures a set of entities/objects, and interconnections between pairs of them
  - *Graphs* also often called *networks*
  - Entities/objects represented by *vertices or nodes*
  - Interconnections between pairs of vertices called *edges*
    - Also called *links, arcs, relationships*



An undirected, unweighted graph



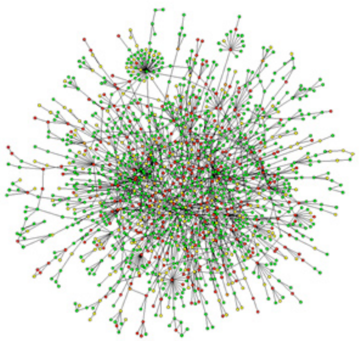
A directed, edge-weighted graph

Background: Graphs

- A *graph* captures a set of entities/objects, and interconnections between pairs of them
  - *Graphs* also often called *networks*
  - Entities/objects represented by *vertices or nodes*
  - Interconnections between pairs of vertices called *edges*
    - Also called *links, arcs, relationships*
- Graph theory, graph algorithms very well studied in Computer Science
  - Not as much work on managing large volumes of graph-structured data, or doing analytics over them

Graph Data

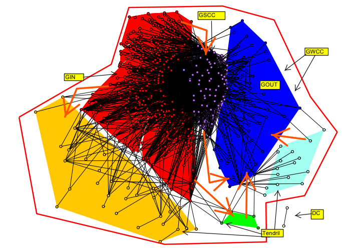
- Increasing interest in querying and reasoning about the *underlying graph (network) structure* in a variety of disciplines



A protein-protein interaction network

facebook twitter LinkedIn

Social networks



Federal funds networks

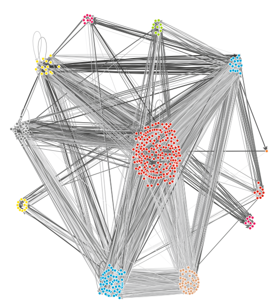
Knowledge Graph

World Wide Web

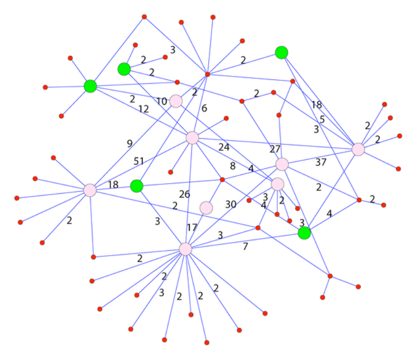
Citation networks

Communication networks

Disease transmission networks



Financial transaction networks



Stock Trading Networks

Motivation

- Underlying data hasn't necessarily changed that much
  - Aside from the data volumes and easier availability
- However, several new realizations:
  - Reasoning about graph structure provides useful and actionable insights *(network science/complex network analysis)*
  - Lose too much information/intuitions if graph structure ignored
  - Not easy to write many natural queries or tasks using traditional tools
    - Especially relational databases like Oracle
  - Hard to efficiently process inherently graph-structured queries or complex network analysis tasks using existing tools
    - A major concern with increasingly large graphs seen in practice

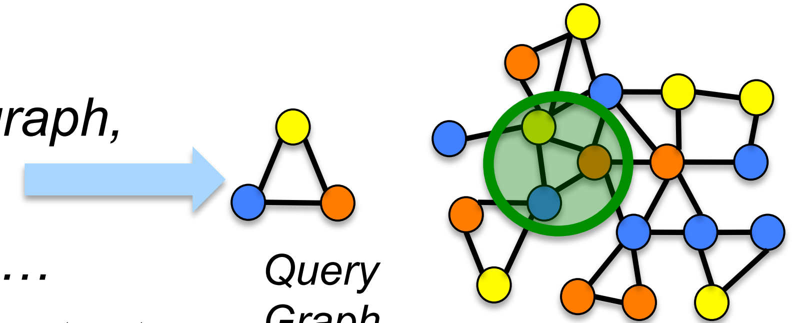
Wide Variety in Graph Queries/Analytics

<u>Different types of "queries"</u>

Subgraph pattern matching: Given a "query" graph, find where it occurs in a given "data" graph

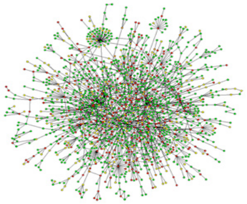
Reachability; Shortest path; Keyword search; …

Historical or Temporal queries: "Find most important nodes in a communication network in 2002?"



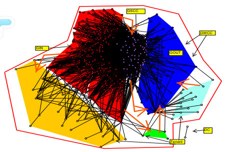
Query Graph

Data Graph



A protein-protein interaction network

Social networks



Federal funds networks

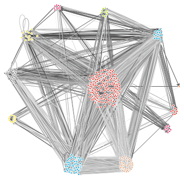
Knowledge Graph

World Wide Web

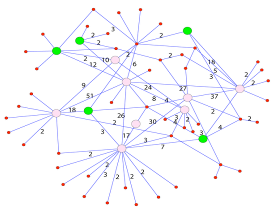
Citation networks

Communication networks

Disease transmission networks



Financial transaction networks



Stock Trading Networks

Wide Variety in Graph Queries/Analytics

Different types of "queries"

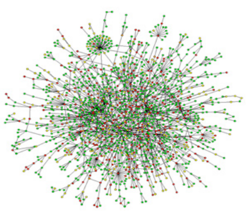
Subgraph pattern matching; Reachability; Shortest path; Keyword search; Historical or Temporal queries…

Continuous "queries" and Real-time analytics

Online prediction in response to new data

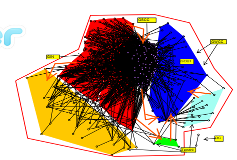
Monitoring: "Tell me when a topic is suddenly trending in my friend circle"

Anomaly/Event detection: "Alert me if the communication activity around a node changes drastically"



A protein-protein interaction network

Social networks

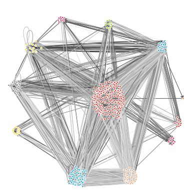


Federal funds networks

Knowledge Graph

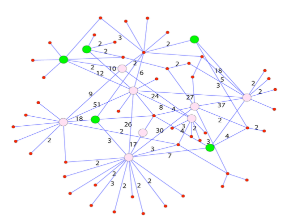
World Wide Web

Citation networks



Communication networks

Financial transaction networks



Disease transmission networks

Stock Trading Networks

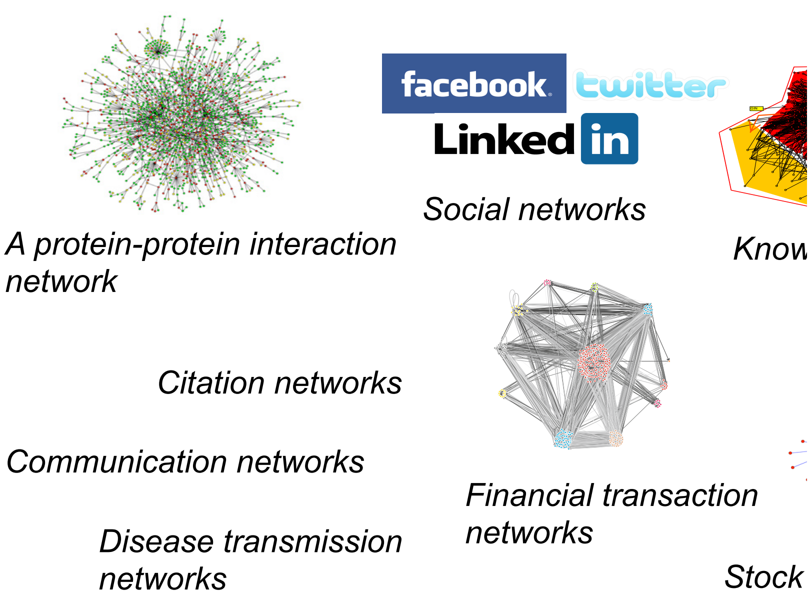
Wide Variety in Graph Queries/Analytics

Different types of "queries"

Subgraph pattern matching; Reachability; Shortest path; Keyword search; Historical or Temporal queries…

Continuous "queries" and Real-time analytics

Online prediction; Monitoring; Anomaly/Event detection



Batch analysis tasks

Centrality analysis: Find the most central nodes in a network

Community detection: Partition vertices into groups with dense interactions

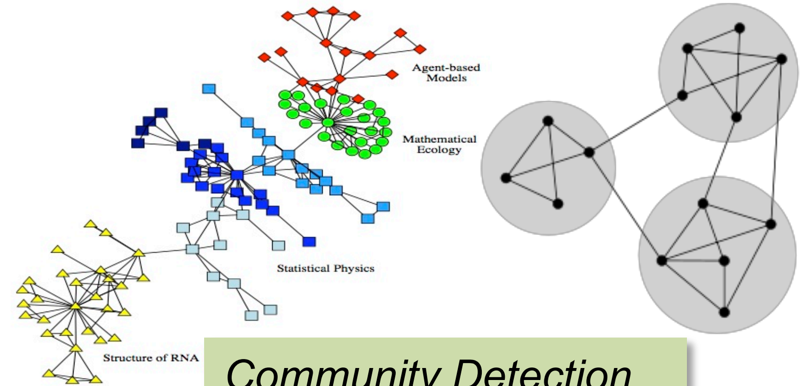
Network evolution: Build models for network formation and evolution

Network measurements: Measure statistical properties

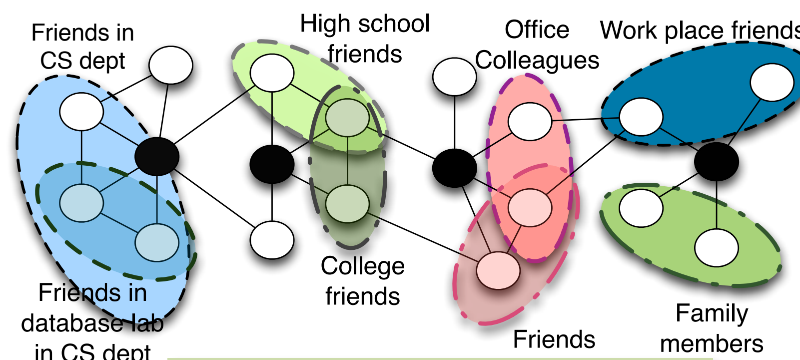
Graph cleaning/inference: Remove noise in the observed network data

Examples of Graph Analysis Tasks

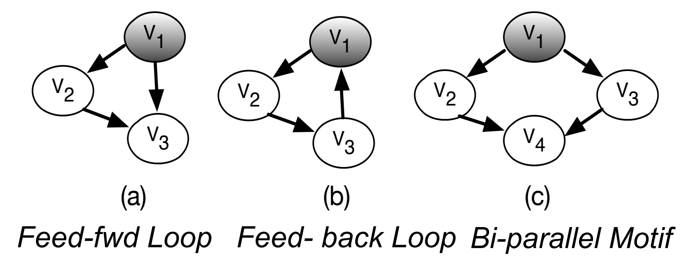
- Community Detection: partitioning the vertices into (potentially overlapping) groups based on the interconnections between them
  - Provide insights into how networks function; identify functional modules; improve performance of Web services…
- Analyzing "ego-networks"
  - Properties of neighborhoods around a large number of nodes
- Building models of evolution
  - Measuring properties of networks
  - Constructing evolution models that can explain those



Community Detection



Identify Social circles



Counting network motifs

Wide Variety in Graph Queries/Analytics

<u>Different types of "queries"</u>

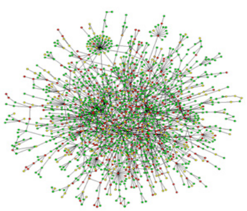
Subgraph pattern matching; Reachability; Shortest path; Keyword search; Historical or Temporal queries…

<u>Continuous "queries" and Real-time analytics</u>

Online prediction; Monitoring; Anomaly/Event detection

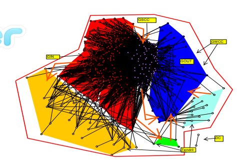
<u>Batch analysis tasks</u>

Centrality analysis; Community detection; Network evolution; Network measurements; Graph cleaning/inference



A protein-protein interaction network

Social networks



Federal funds networks

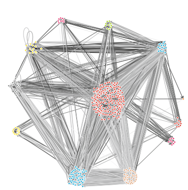
Knowledge Graph

World Wide Web

<u>Machine learning tasks</u>

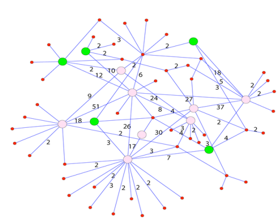
Many algorithms can be seen as message passing in specially constructed graphs

Citation networks



Communication networks

Financial transaction networks



Disease transmission networks

Stock Trading Networks

Graph Data Management: State of the Art

- Much prior and ongoing work – most of it outside, or on top of, general-purpose data management systems
  - Specialized indexes or algorithms for specific types of queries
  - Stand-alone prototypes for specific analysis tasks
- Emergence of specialized graph databases in recent years
  - Neo4j, Titan, OrientDB, DEX, AllegroGraph, …
  - Rudimentary declarative interfaces/query languages
- Several “vertex-centric” frameworks in recent years
  - Pregel, Giraph, GraphLab, GRACE, GraphX, …
  - Only work well for a very limited set of tasks
- Little work on continuous/real-time query processing, or on supporting evolutionary or temporal analytics

What we are doing

- Goal: A graph data management system with unified declarative abstractions for graph queries and analytics
- Work so far
  - Declarative graph cleaning [GDM'11, SIGMOD Demo'13]
  - NScale: a distributed analysis framework [VLDB Demo'14, VLDBJ'15]
  - Real-time continuous queries [SIGMOD'12, ESNAM'14, SIGMOD'14]
    - Techniques for continuous query processing over large dynamic graphs
    - Expressive query language for specifying anomaly detection queries
  - Historical graph data management [ICDE'13, SIGMOD Demo'13,arXiv'15]
    - A distributed indexing structure for retrieving historical snapshots
    - Temporal/evolutionary analytics framework, built on top of Apache Spark
  - Subgraph pattern matching and counting [ICDE'12, ICDE'14]
  - GraphGen: graph analytics over relational data [VLDB Demo'15]

Outline

- Graph Data Management
  - A Framework for Distributed Graph Analytics
- DataHub: A platform for collaborative data science

Scaling Graph Analysis Tasks

- Graph analytics/network science tasks too varied
- Hard to build general platforms like Hadoop/Dryad/Spark
  - What is a good programming abstraction to provide?
    - Needs to cover a large fraction of use cases, and be easy to use
    - MapReduce works very well for other analysis tasks, but not a good fit for graph analytics
  - No clear winner yet, so little progress on systems
    - Especially on distributed or parallel systems
  - Application developers largely doing their own thing

"Vertex-centric" Frameworks

- Introduced by Google in a system called "Pregel"
  - Inspired by BSP (Bulk Synchronous Protocol)
- Adopted by many other systems
  - GraphLab, Apache Giraph, GraphX, Xstream, …
  - Most of the research, especially in databases, focuses on it
- "Think like a vertex" paradigm
  - User provides a single **compute()** function that operates on a vertex
  - Executed in parallel on all vertices in an iterative fashion
  - Exchange information at the end of each iteration through message passing

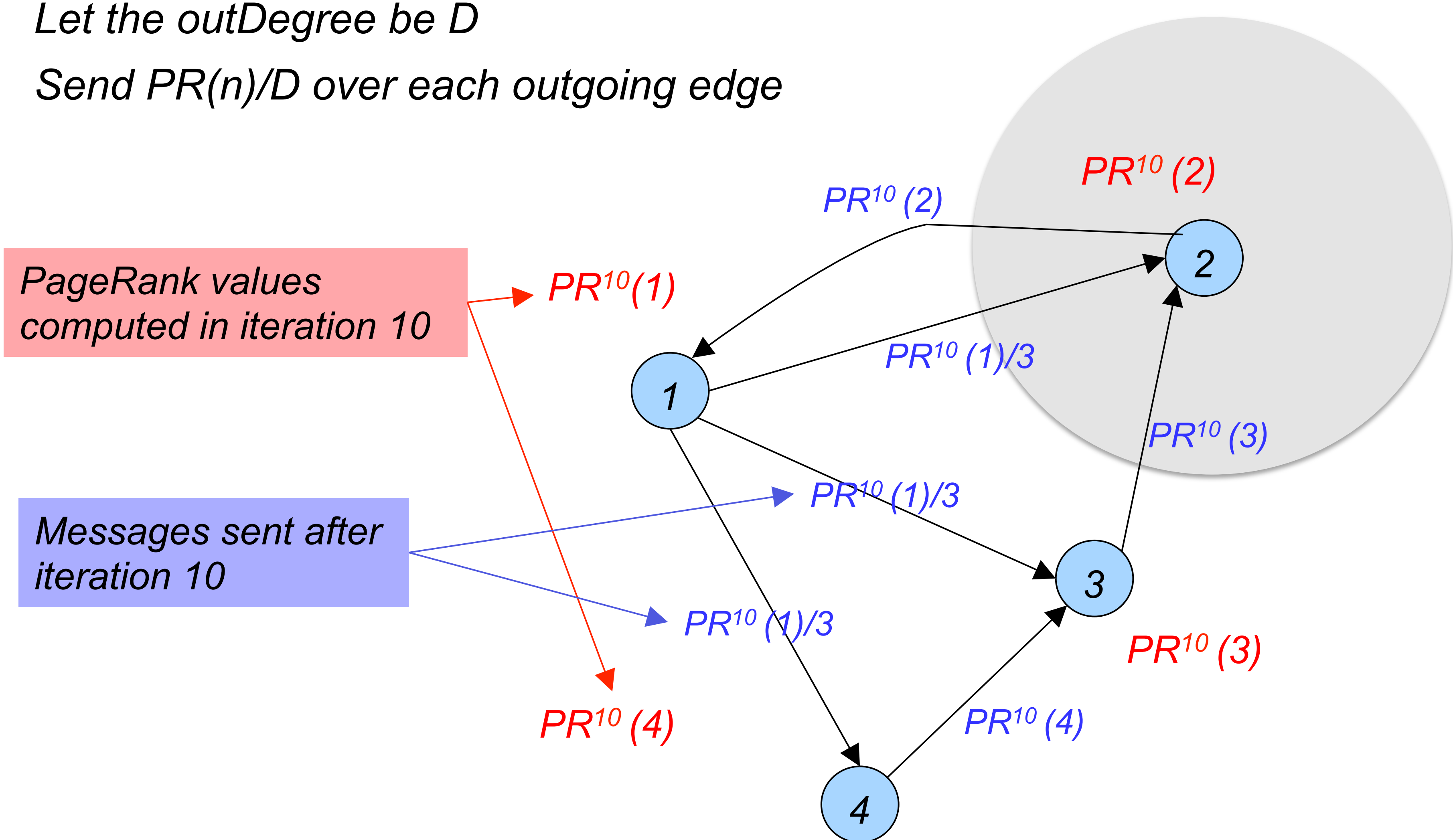
Example: PageRank

Compute() at Node n:

PR(n) = sum up all the incoming weights

Let the outDegree be D

Send PR(n)/D over each outgoing edge



Programming Frameworks

- Vertex-centric framework
  - Works well for some applications
    - Pagerank, Connected Components, …
    - Some machine learning algorithms can be mapped to it
  - However, the framework is very restrictive
    - Most analysis tasks or algorithms cannot be written easily
    - Simple tasks like counting neighborhood properties infeasible
    - Fundamentally: Not easy to decompose analysis tasks into vertex-level, independent local computations
- Alternatives?
  - Galois, Ligra, GreenMarl: Not sufficiently high-level
  - Some others (e.g., Socialite) restrictive for different reasons

Example: Local Clustering Coefficient

Compute() at Node n:

Need to count the no. of edges between neighbors

But does not have access to that information

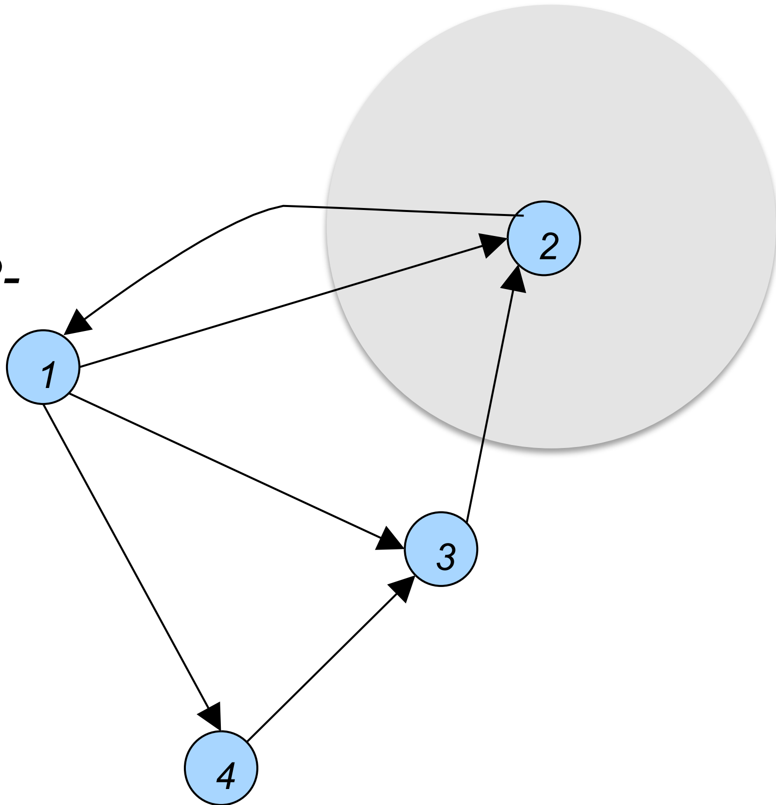
Option 1: Each node transmits its list of neighbors to its neighbors

Huge memory consumption

Option 2: Allow access to neighbors' state

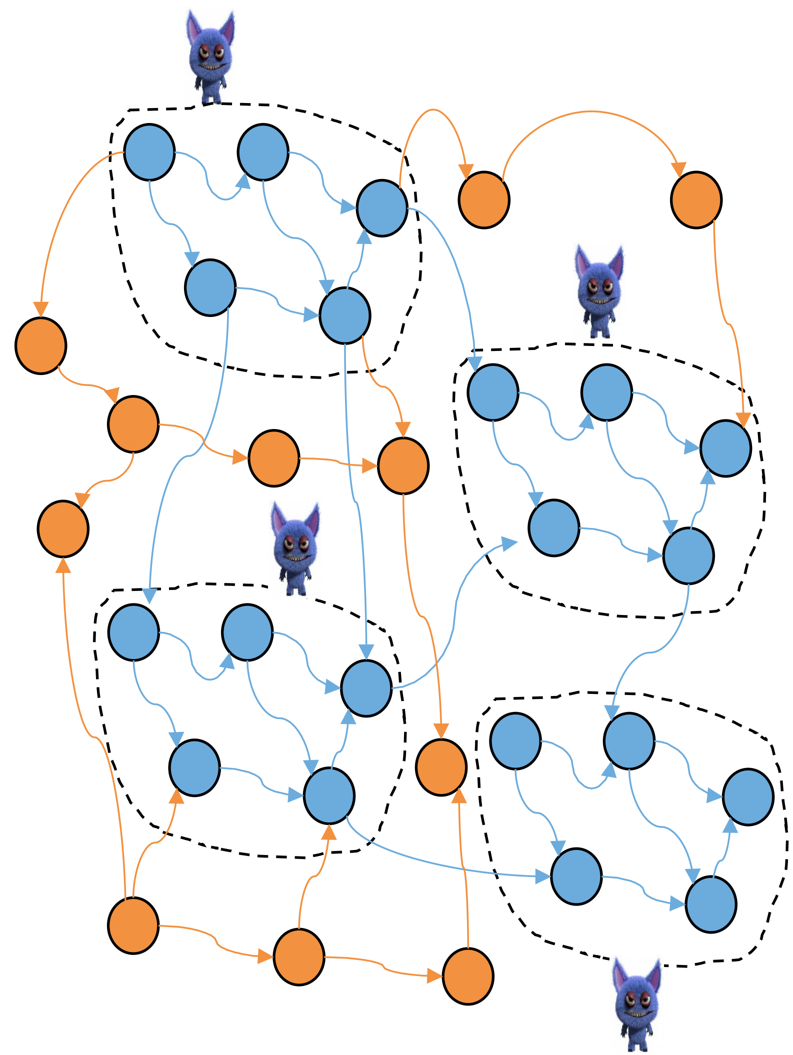
Neighbors may not be local

What about computations that require 2-hop information?



NScale Programming Framework

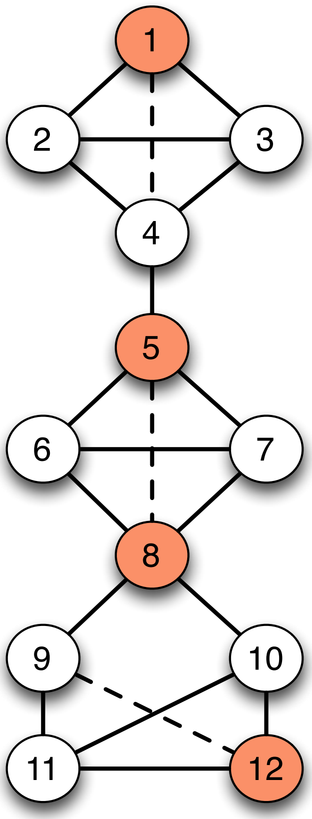
- An end-to-end distributed graph programming framework
- Users/application programs specify:
  - Neighborhoods or subgraphs of interest
  - A kernel computation to operate upon those subgraphs
- Framework:
  - Extracts the relevant subgraphs from underlying data and loads in memory
  - Execution engine: Executes user computation on materialized subgraphs
  - Communication: Shared state/ message passing



NScale: LCC Computation Walkthrough

NScale programming model

Underlying graph data on HDFS



Subgraph extraction query:

Compute (LCC) on
Extract ({Node.color=orange}
{k=1}
{Node.color=white}
{Edge.type=solid}
)

Query-vertex predicate

Neighborhood Size

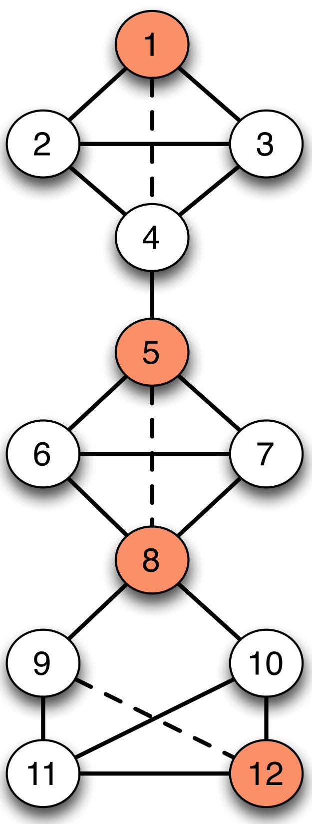
Neighborhood vertex predicate

Neighborhood edge predicate

NScale: LCC Computation Walkthrough

NScale programming model

Underlying graph data on HDFS



Specifying Computation: BluePrints API

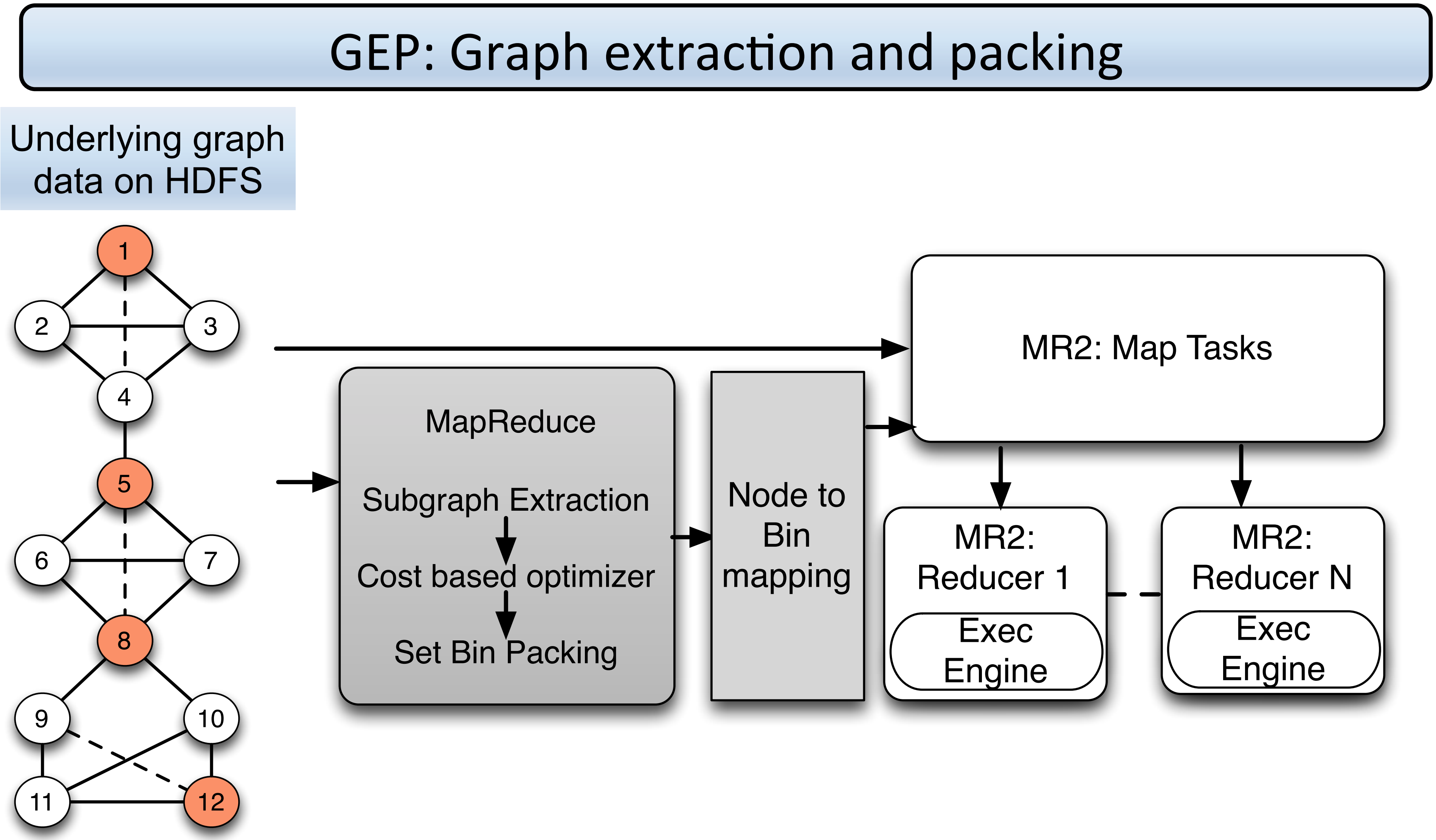
```
ArrayList<RVertex> n_arr = new ArrayList<RVertex>();
for(Edge e: this.getQueryVertex().getOutEdges)
     n_arr.add(e.getVertex(Direction.IN));

int possibleLinks = n_arr.size()* (n_arr.size()-1)/2;

// compute #actual edges among the neighbors
for(int i=0; i < n_arr.size()-1; i++)
   for(int j=i+1; j < n_arr.size(); j++)
      if(edgeExists(n_arr.get(i), n_arr.get(j)))
         numEdges++;
```

Program cannot be executed as is in vertex-centric programming frameworks.

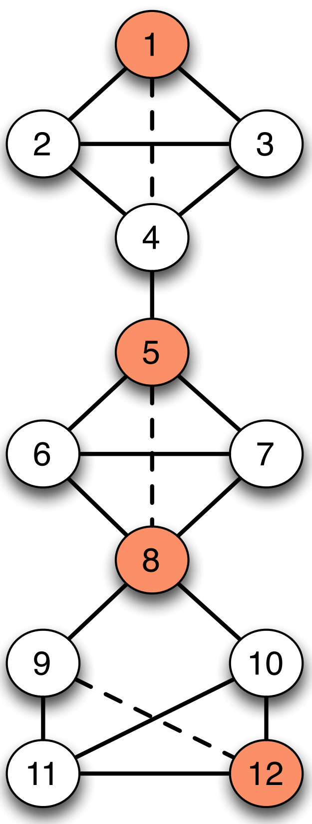
NScale: LCC Computation Walkthrough



NScale: LCC Computation Walkthrough

GEP: Graph extraction and packing

Underlying graph data on HDFS



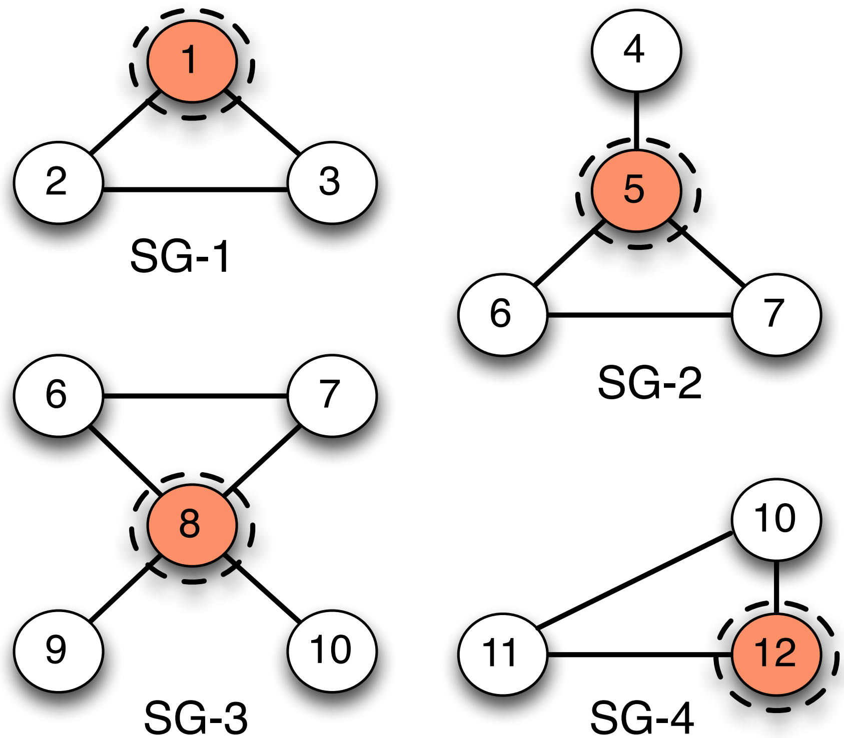
Graph Extraction and Loading

MapReduce (Apache Yarn)

↓

Subgraph extraction

Extracted Subgraphs



NScale: LCC Computation Walkthrough

GEP: Graph extraction and packing

Goal:

- Group graphs with high similarity
- Minimizes memory consumption

Techniques explored

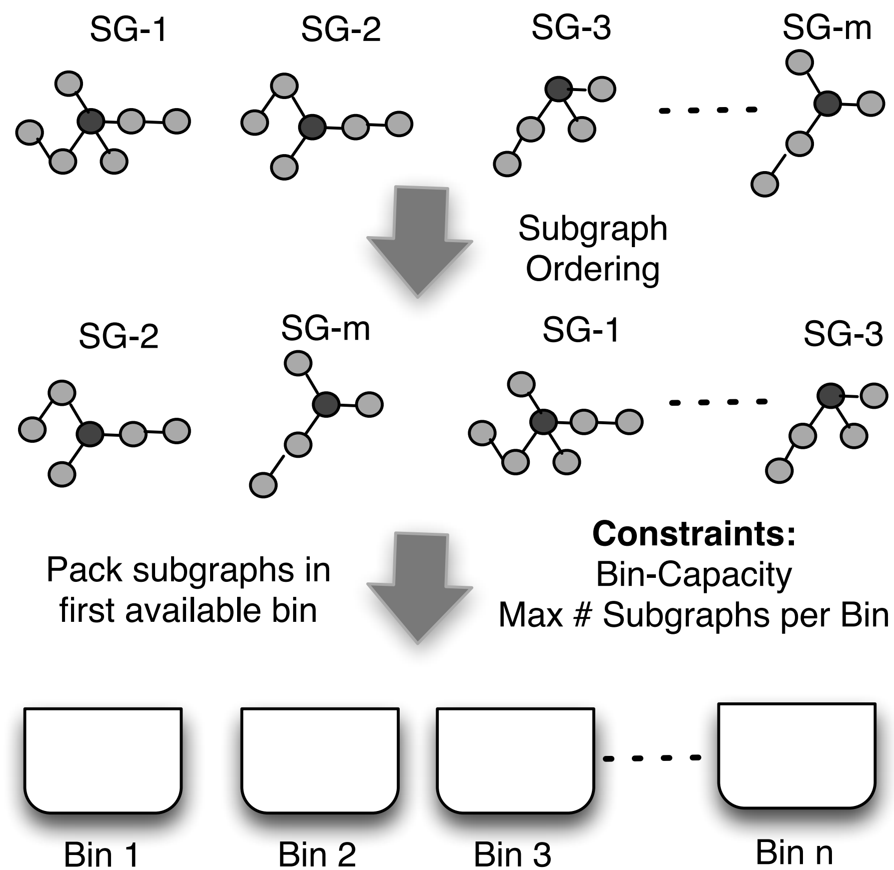
- Set bin packing, graph partitioning, clustering

Shingle based set bin packing

- **Min-hash signatures** based sorting
- Grouping based on **Jaccard** similarity

Bin Packing

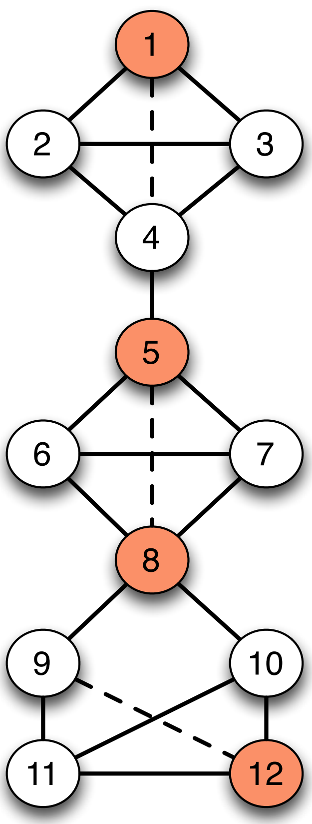
- Set union operation
- Bin Capacity: **Elastic resource allocation**
- Max # Subgraphs: **Handles Skew**



NScale: LCC Computation Walkthrough

GEP: Graph extraction and packing

Underlying graph data on HDFS



Graph Extraction and Loading

MapReduce (Apache Yarn)

↓

Subgraph extraction

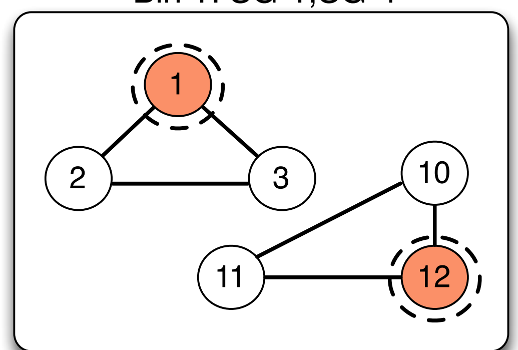
Cost Based Optimizer

↓

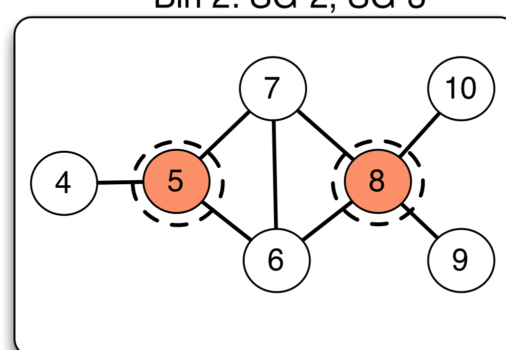
Data Rep & Placement

Sample bin packing using Shingles

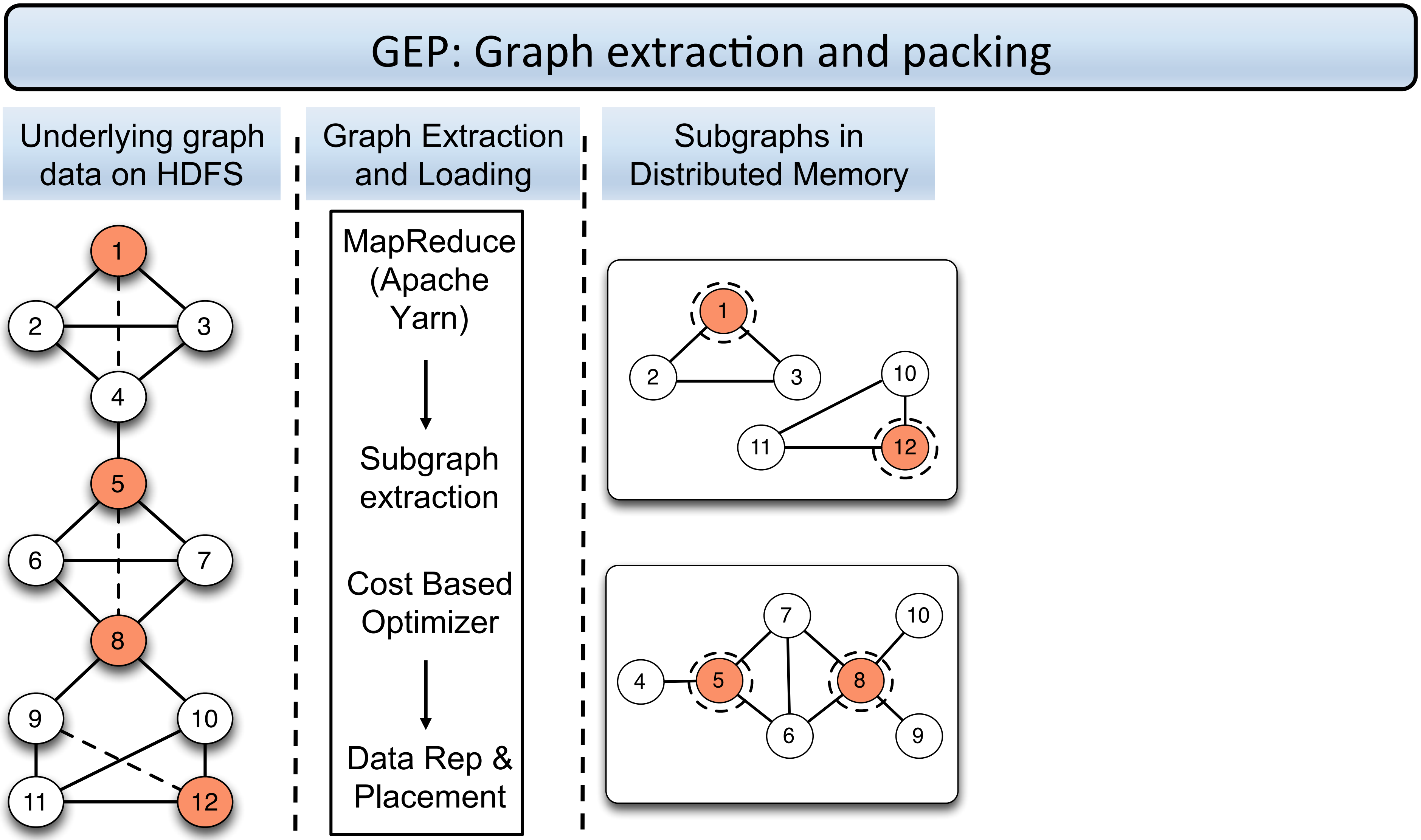
Bin 1: SG-1,SG-4



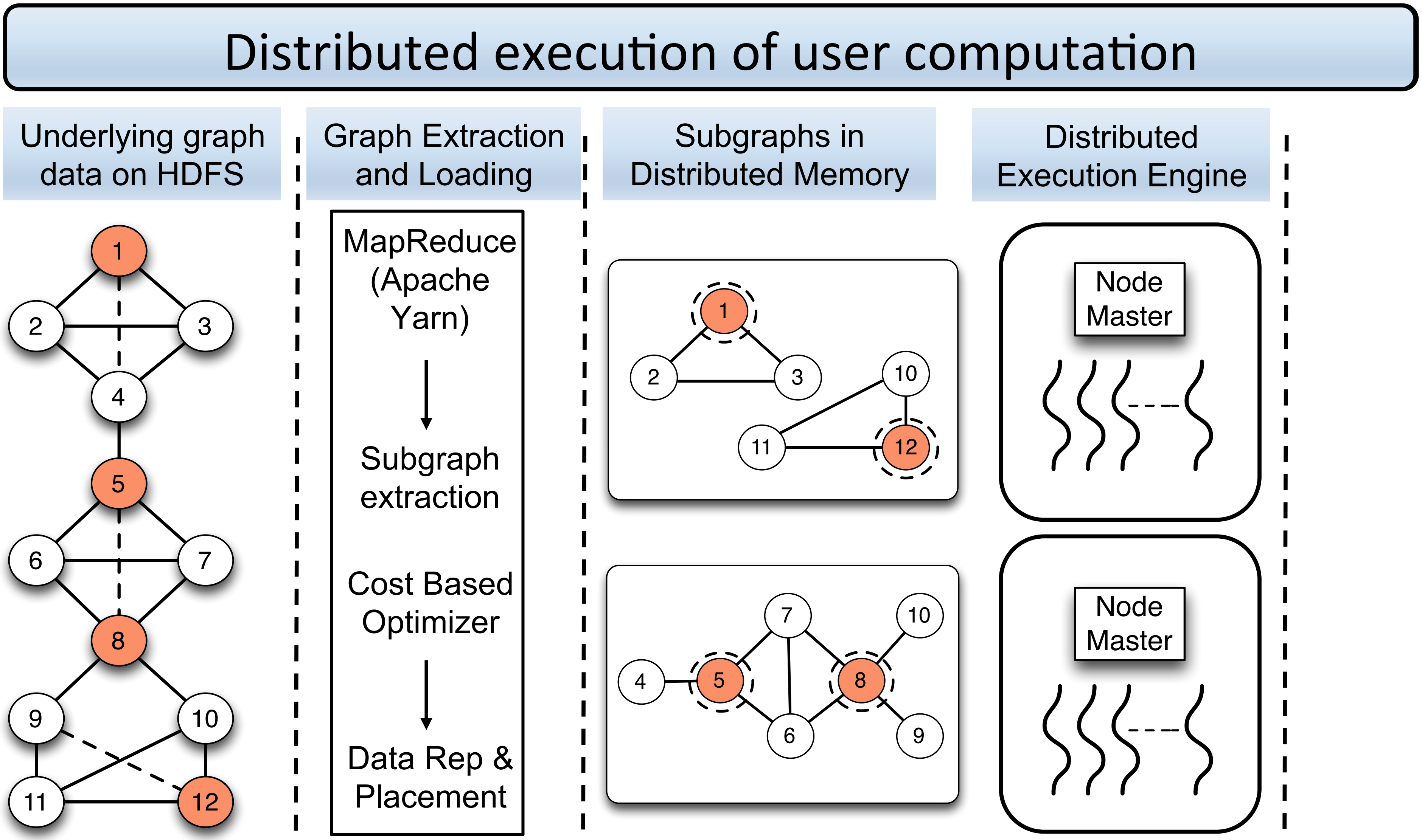
Bin 2: SG-2, SG-3



NScale: LCC Computation Walkthrough



NScale: LCC Computation Walkthrough



NScale: Summary

- Users write programs at the abstraction of a graph
  - More intuitive for graph analytics
  - Captures mechanics of common graph analysis/cleaning tasks
- Generalization: Flexibility in subgraph definition
  - Subgraph = vertex and associated edges: vertex-centric programs
  - Subgraph = an entire graph: global programs
- Scalability
  - Only relevant portions of the graph data loaded into memory
    - User can specify subgraphs of interest, and select nodes or edges based on properties
  - Carefully partition (pack) nodes across machines so that:
    - Every subgraph is entirely in memory on a machine, while using very few machines

Experimental Evaluation

- **Datasets**
  - Web graphs
  - Communication/interaction graphs
  - Social networks
- **Graph applications**
  - Local Clustering Coefficient
  - Motif counting
  - Identifying weak ties
  - Triangle Counting
  - Personalized Page Rank

- **Baselines**
  - Apache Giraph
  - GraphLab
  - GraphX
- **Evaluation Metrics**
  - Computational Effort
  - Execution Time
  - Cluster Memory
- **Cluster Setup**
  - 16 Node Cluster
  - Apache YARN (MRv2)
  - Each Node:
    - 2 x 4-core Intel Xeon
    - 24GB RAM, 3 x 2 TB disks

Experimental Evaluation

	Local Clustering Coefficient							
Dataset	NScale		Giraph		GraphLab		GraphX	
	CE (Node-Secs)	Cluster Mem (GB)	CE (Node-Secs)	Cluster Mem (GB)	CE (Node-Secs)	Cluster Mem (GB)	CE (Node-Secs)	Cluster Mem (GB)
EU Email	377	9.00	1150	26.17	365	20.10	225	4.95
NotreDame	620	19.07	1564	30.14	550	21.40	340	9.75
Google Web	658	25.82	2024	35.35	600	33.50	1485	21.92
WikiTalk	726	24.16	DNC	OOM	1125	37.22	1860	32.00
LiveJournal	1800	50.00	DNC	OOM	5500	128.62	4515	84.00
Orkut	2000	62.00	DNC	OOM	DNC	OOM	20175	125.00

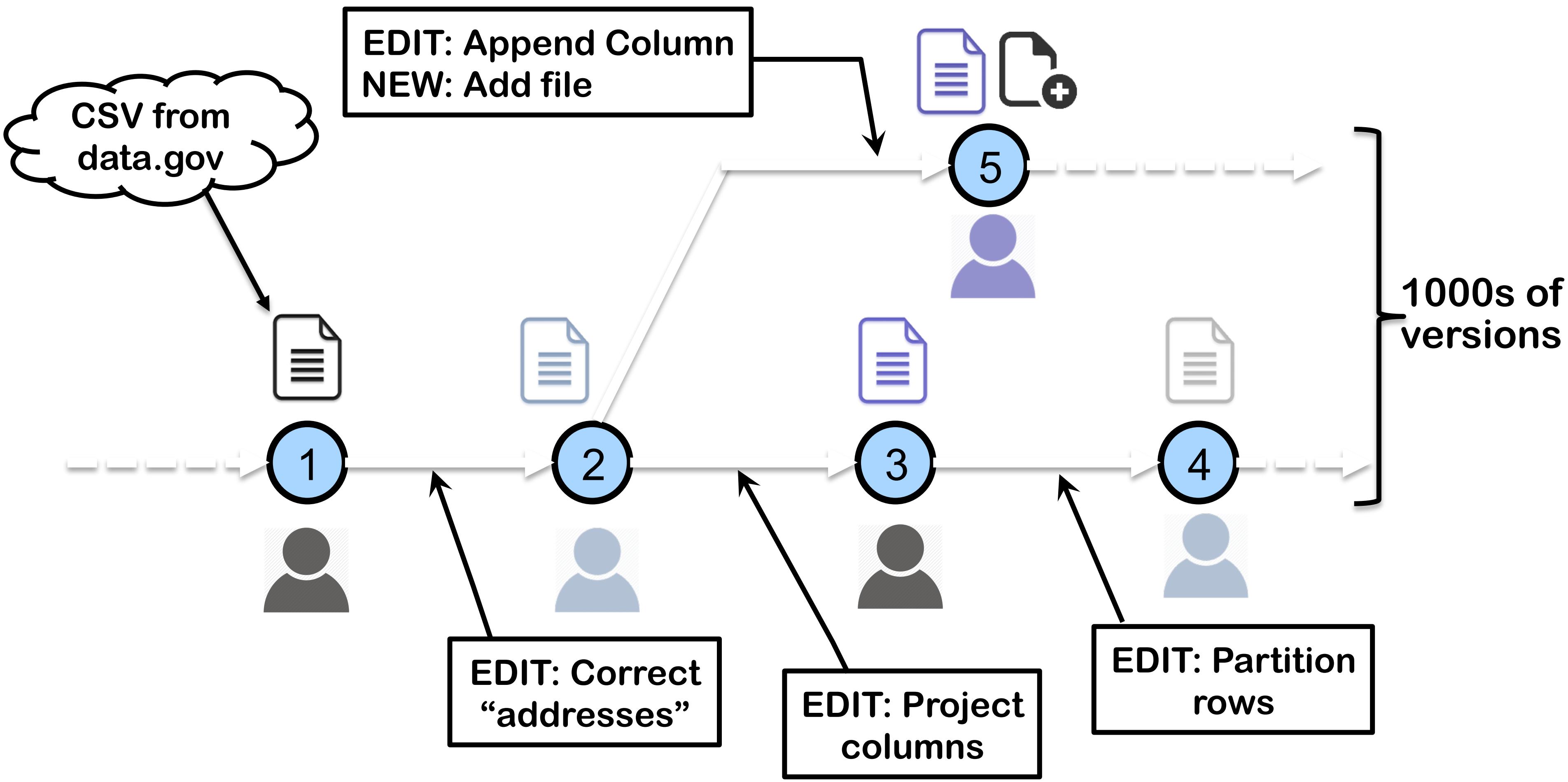
		Personalized Page Rank on 2-Hop Neighborhood							
Dataset		NScale		Giraph		GraphLab		GraphX	
	#Source Vertices	CE (Node-Secs)	Cluster Mem (GB)	CE (Node-Secs)	Cluster Mem (GB)	CE (Node-Secs)	Cluster Mem (GB)	CE (Node-Secs)	Cluster Mem (GB)
EU Email	3200	52	3.35	782	17.10	710	28.87	9975	85.50
NotreDame	3500	119	9.56	1058	31.76	870	70.54	50595	95.00
Google Web	4150	464	21.52	10482	64.16	1080	108.28	DNC	-
WikiTalk	12000	3343	79.43	DNC	OOM	DNC	OOM	DNC	-
LiveJournal	20000	4286	84.94	DNC	OOM	DNC	OOM	DNC	-
Orkut	20000	4691	93.07	DNC	OOM	DNC	OOM	DNC	-

Outline

- Graph Data Management
  - A Framework for Distributed Graph Analytics
- DataHub: A platform for collaborative data science

Collaborative Data Science

- Widespread use of "data science" in many many domains



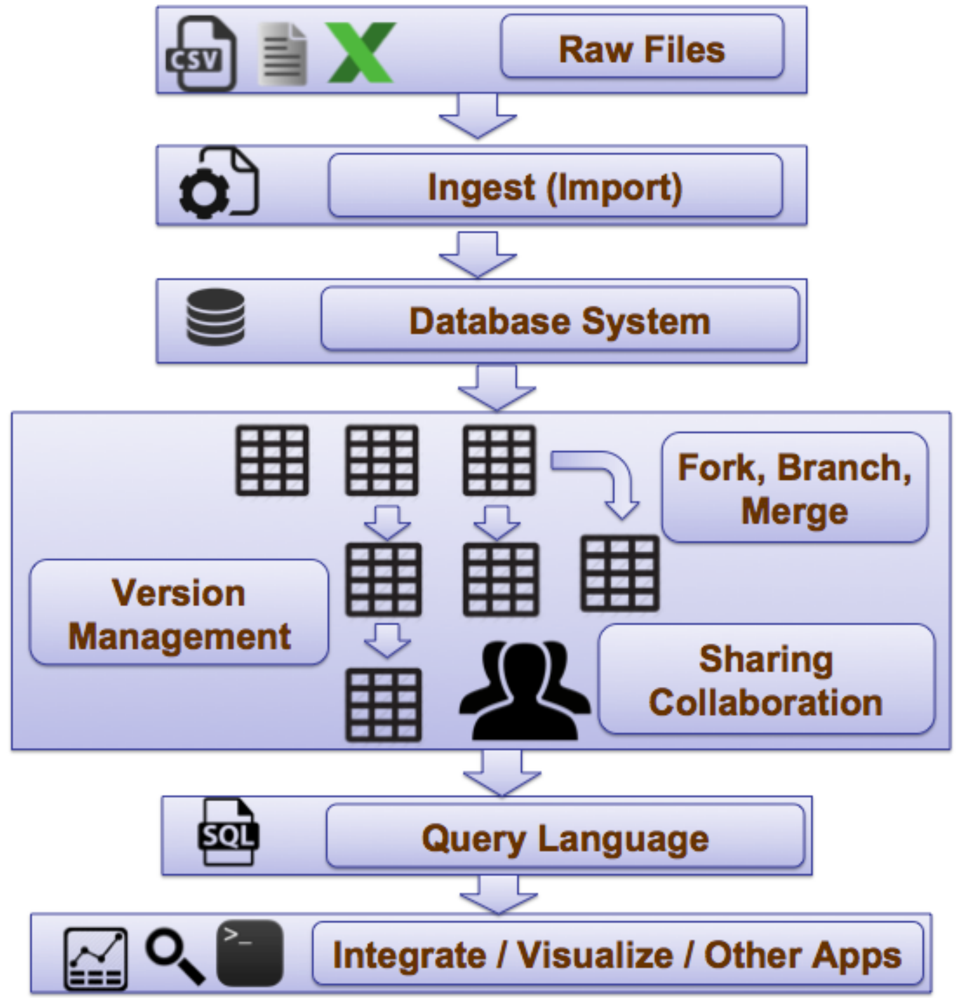
A typical data analysis workflow

Collaborative Data Science

- Widespread use of "data science" in many many domains
- Increasingly the "pain point" is managing the *process,* especially during collaborative analysis
  - Many private copies of the datasets ➔ Massive redundancy
  - No easy way to keep track of dependencies between datasets
  - Manual intervention needed for resolving conflicts
  - No efficient organization or management of datasets
  - No way to analyze/compare/query versions of a dataset
- Ad hoc data management systems (e.g., Dropbox) used
  - Much of the data is unstructured so typically can't use DBs
  - The process of data science itself is quite ad hoc and exploratory
  - Scientists/researchers/analysts are pretty much on their own

DataHub: A Collaborative Data Science Platform

The one-stop solution for collaborative data science and dataset version management



http://data-hub.org

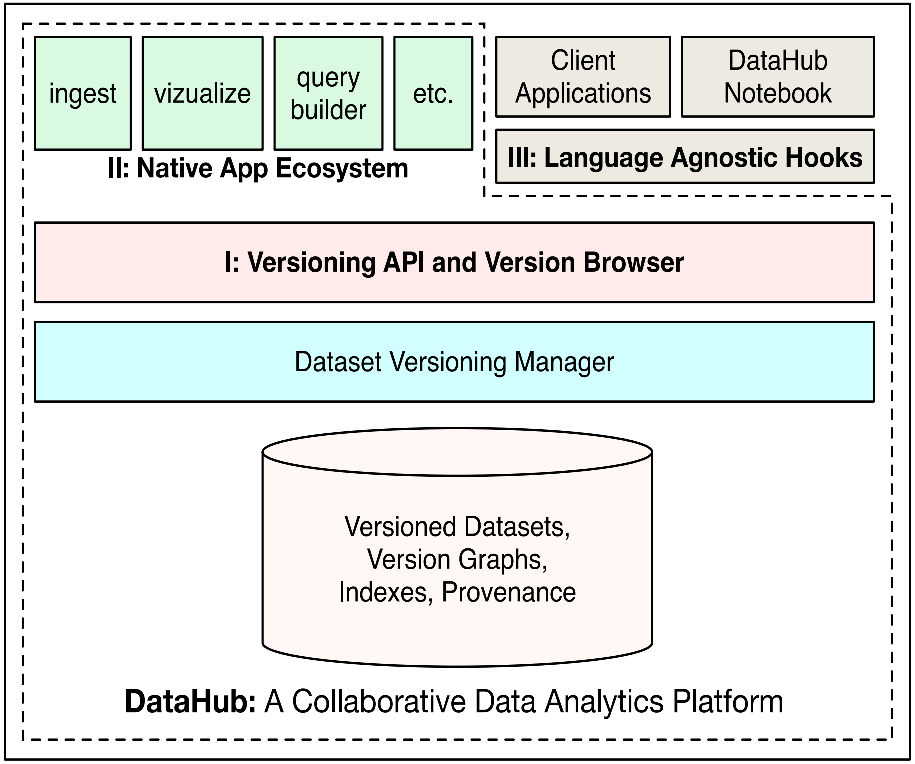
Work being done in collaboration with Sam Madden (MIT) and Aditya Parameswaran (UIUC)

DataHub: A Collaborative Data Science Platform

- a dataset management system – import, search, query, analyze a large number of (public) datasets

- a dataset version control system – branch, update, merge, transform large structured or unstructured datasets

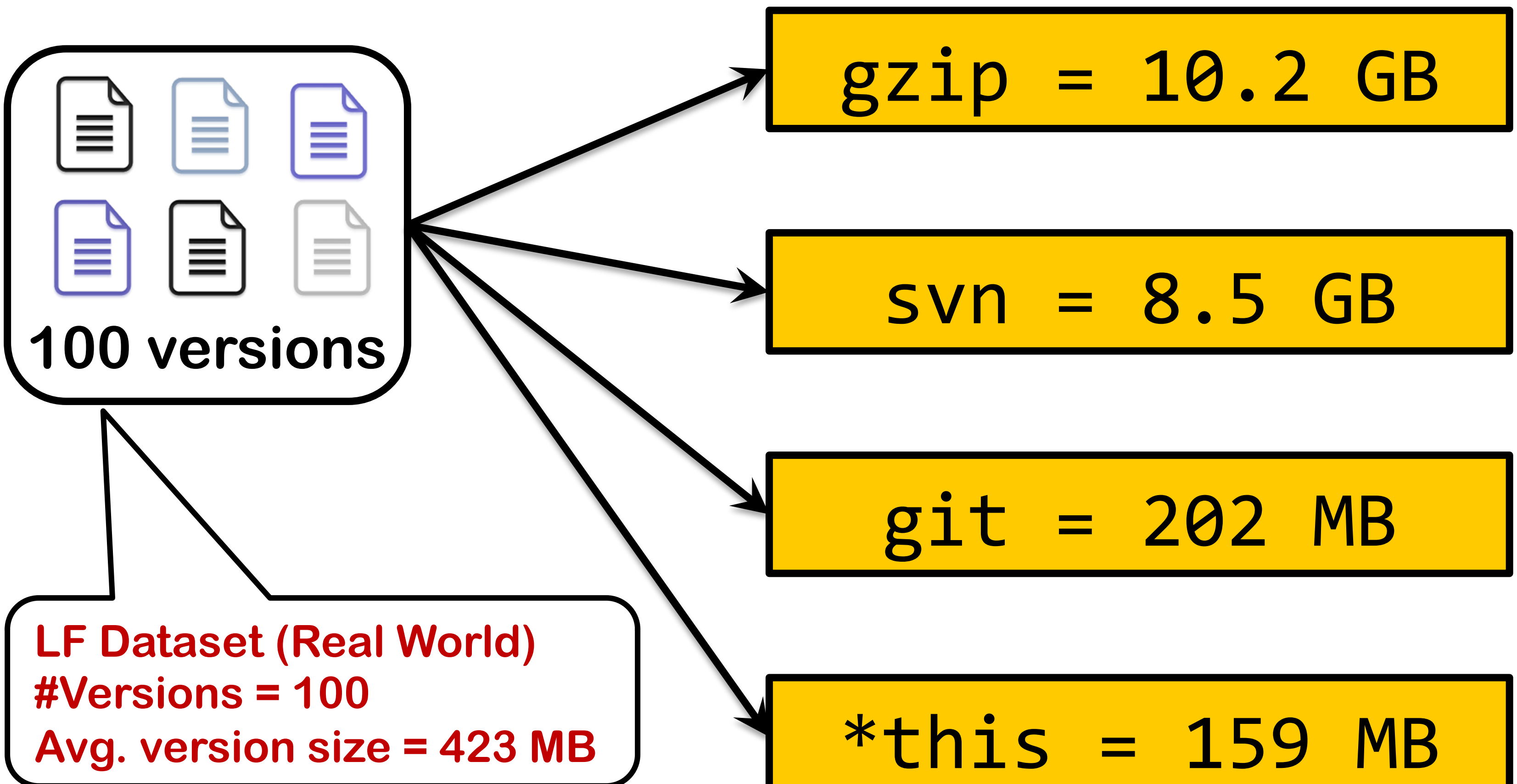
- an app ecosystem and hooks for external applications (Matlab, R, iPython Notebook, etc)



DataHub Architecture

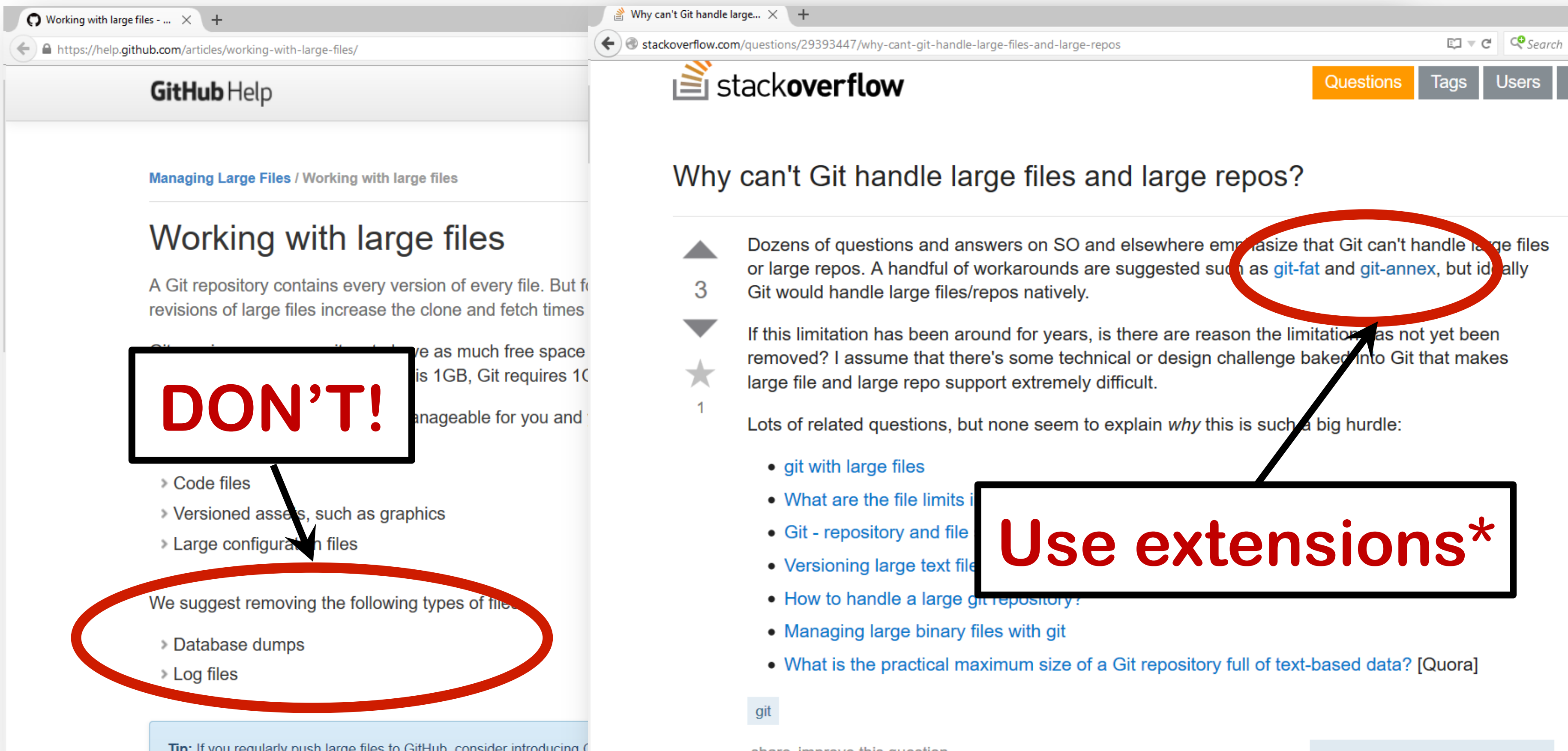
Can we use Version Control Systems (e.g., Git)?

- ✗ No, because they typically use fairly simple algorithms and are optimized to work for code-like data



Can we use Version Control Systems (e.g., Git)?

- ✗ No, because they typically use **fairly simple algorithms** and are optimized to work for code-like data
- ✗ Git ends up using **large amounts of RAM** for large files



Can we use Version Control Systems (e.g., Git)?

- **No, because they typically use fairly simple algorithms and are optimized to work for code-like data**
- **Git ends up using large amounts of RAM for large files**
- **Querying and retrieval functionalities are primitive, and revolve around single version and metadata retrieval**
- **No way to specify queries like:**
  - *identify all datasets derived of dataset A that satisfy property P*
  - *identify all predecessor versions of version A that differ from it by a large number of records*
  - *rank a set of versions according to a scoring function*
  - *find the version where the result of an aggregate query is above a threshold*
  - *find parent records of all records in version A that satisfy certain property*

Can we use Version Control Systems (e.g., Git)?

- ✗ **No, because they typically use fairly simple algorithms and are optimized to work for code-like data**
- ✗ **Git ends up using large amounts of RAM for large files**

> VQuel: A Unified Query Language for querying versioning and derivation information [USENIX TAPP'15]
>
> Example: What changes did Alice make after January 01, 2015?
>
> ```
> range of V is Version
> retrieve V.all
> where V.author.name = "Alice" and
>       V.creation_ts >= "01/01/2015"
> ```

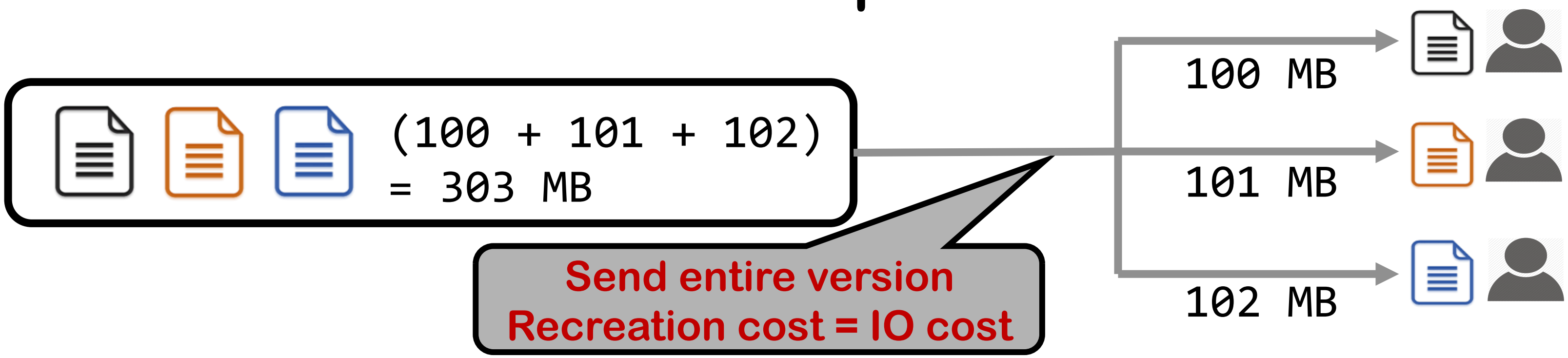
- *find the version where the result of an aggregate query is above a threshold*
- *find parent records of all records in version A that satisfy certain property*

Outline

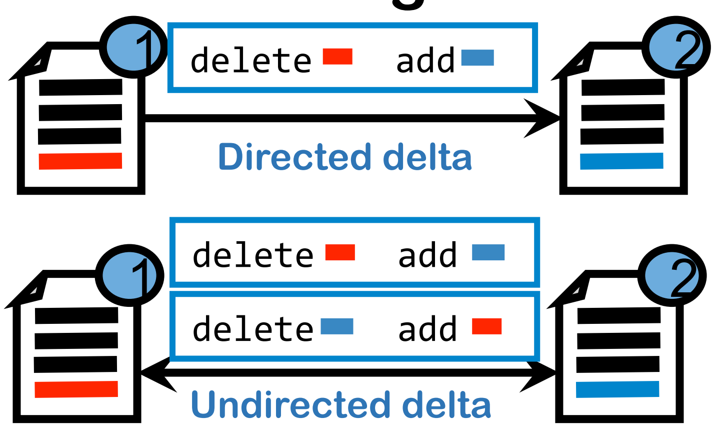
- Graph Data Management
  - A Framework for Distributed Graph Analytics
- DataHub: A platform for collaborative data science
  - Recreation/Storage Tradeoff in Version Management [VLDB'15]

Storage cost is the space required to store a set of versions

Recreation cost is the time* required to access a version



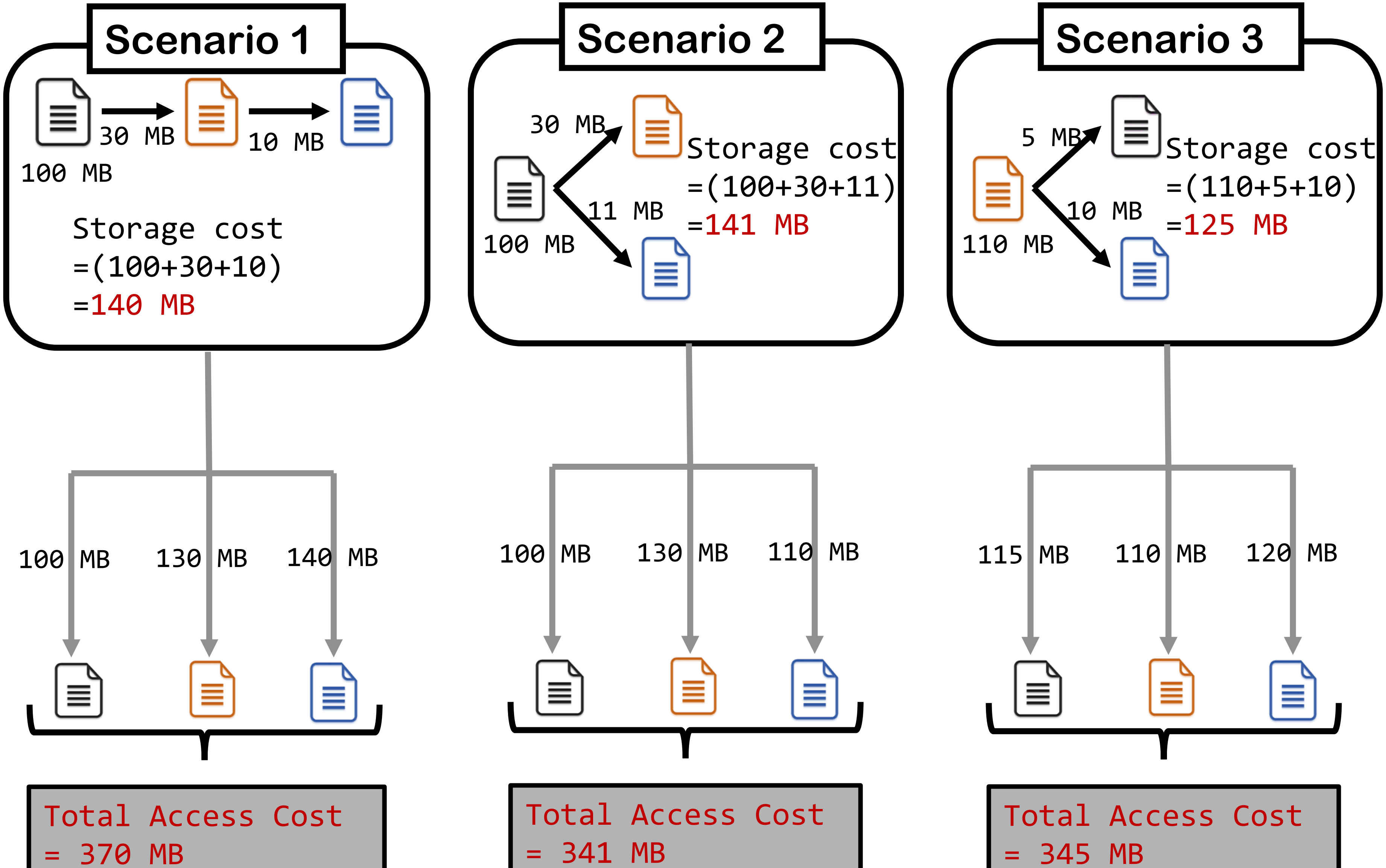
A **delta** between versions is a file which allows constructing one version given the other



Example: Unix diff, xdelta, XOR, etc.

A delta has its own **storage** cost and **recreation** cost, which, in general, are **independent** of each other

Storage-Recreation Tradeoff



Storage-Recreation Tradeoff

Given

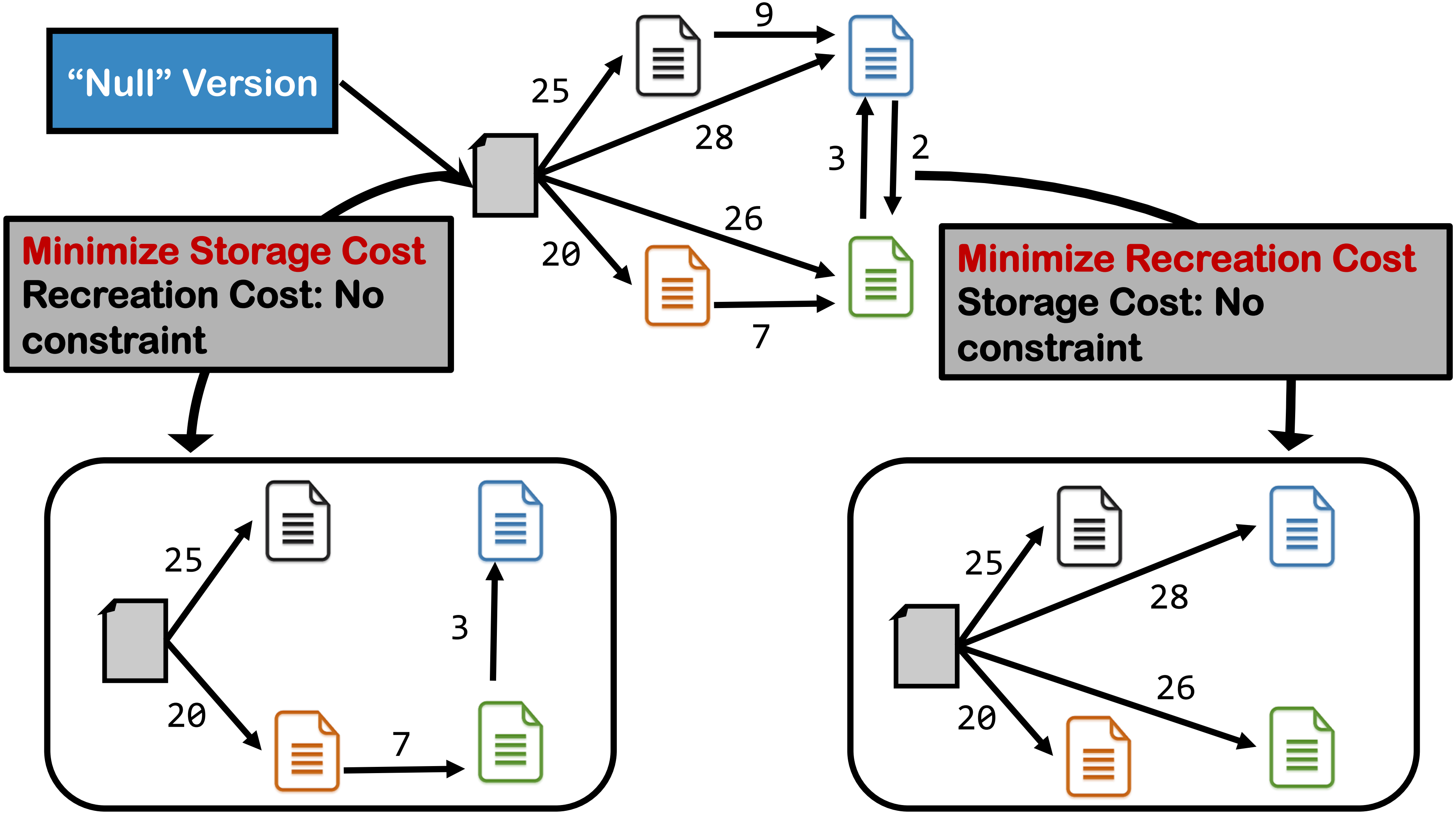
1) a set of versions
2) partial information about deltas between versions

Find a Storage Solution that:

- minimizes total recreation cost given a storage budget, or
- minimizes max recreation cost given a storage budget

	Storage Cost	Recreation Cost	Undirected Case, $\Delta = \Phi$	Directed Case, $\Delta = \Phi$	Directed Case, $\Delta \neq \Phi$
P1	min C	$R_i < \infty, \forall i$	PTime, Minimum Cost Arborescence (MCA)		
P2	$C < \infty$	$\min \{\max \{R_i \mid 1 \le i \le n\}\}$	PTime, Shortest Path Tree (SPT)		
P3	$C \le \beta$	$\min \{ \sum_{i=1}^{n} R_i \}$	NP-hard, LAST* Alg	NP-hard, LMG Algorithm	
P4	$C \le \beta$	$\min \{\max \{R_i \mid 1 \le i \le n\}\}$		NP-hard, MP Algorithm	
P5	min C	$\sum_{i=1}^{n} R_i \le \theta$	NP-hard, LAST* Alg	NP-hard, LMG Algorithm	
P6	min C	$\max \{R_i \mid 1 \le i \le n\} \le \theta$		NP-hard, MP Algorithm	

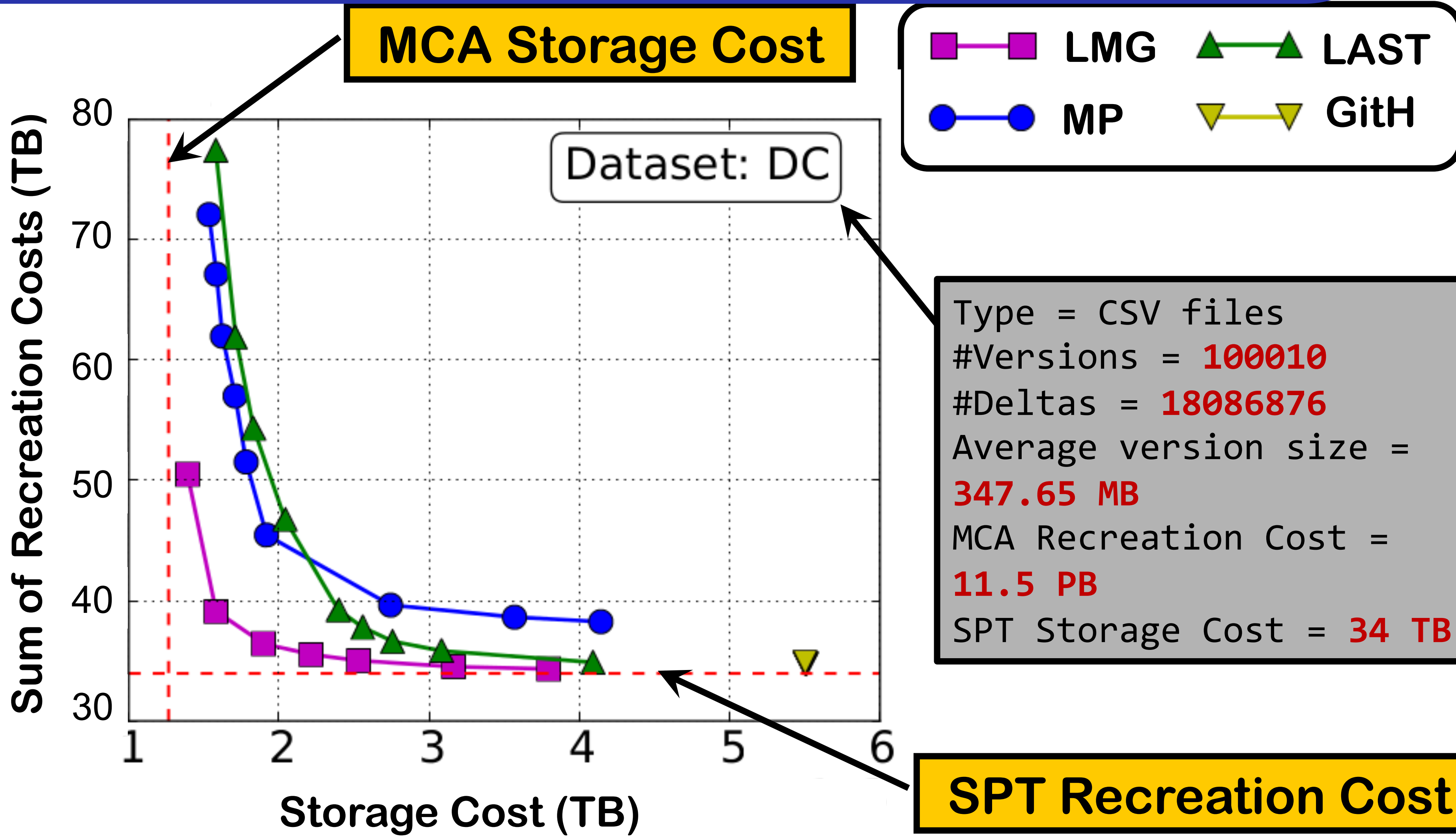
Baselines



Minimum Cost Arborescence (MCA)
Edmonds' algorithm
Time complexity = O(E + V logV)

Shortest Path Tree (SPT)
Dijkstra's algorithm
Time complexity = O(E logV)

Comparing Different Solutions



Type = CSV files
#Versions = **100010**
#Deltas = **18086876**
Average version size = **347.65 MB**
MCA Recreation Cost = **11.5 PB**
SPT Storage Cost = **34 TB**

Storage budget of 1.1X the MCA reduces total recreation cost by 1000X

The Road Ahead

Extensions

- Include user defined functions – e.g., custom "diff" functions for two versions
- Additional graph traversal operators

Engagement with users to refine the constructs

Implementation Challenges

Data is stored in a compressed fashion, to exploit overlaps between versions → *Need new query execution and optimization strategies*

Version graph can become very large in a "dynamic update" environment → *Need scalable methods to handle the version graph*

Thanks !!

More at: http://www.cs.umd.edu/~amol

Questions ?