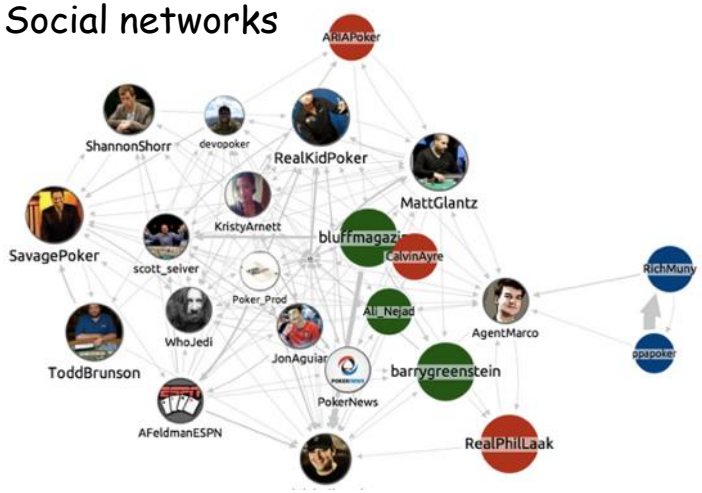
Alibaba Group
阿里巴巴集团

Managing, Analyzing, and Learning Heterogeneous Graph Data: Challenges and Opportunities

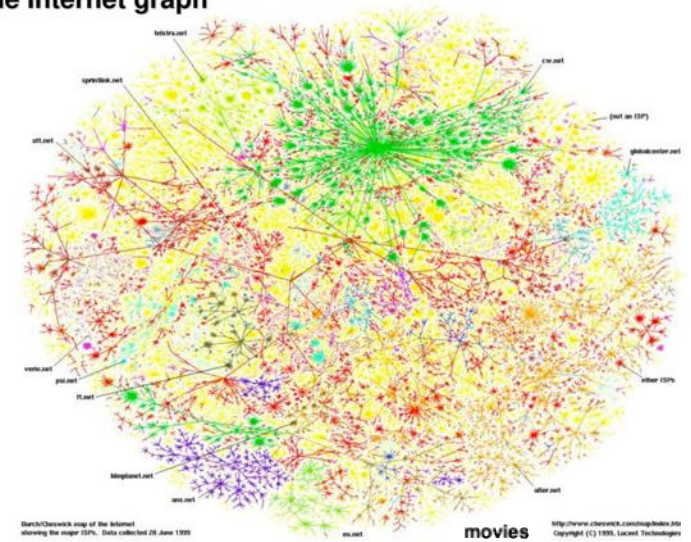
Jingren Zhou
Vice President
Alibaba Group

Graph Data is Prevalent

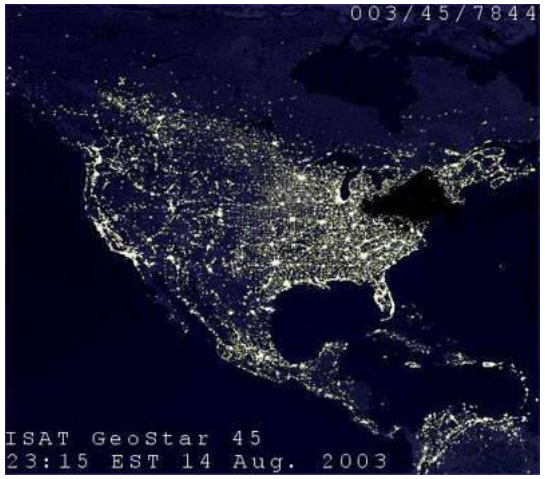
Social networks



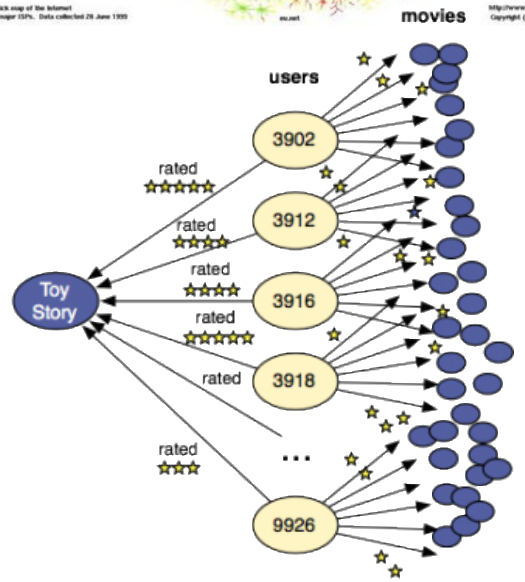
The Internet graph



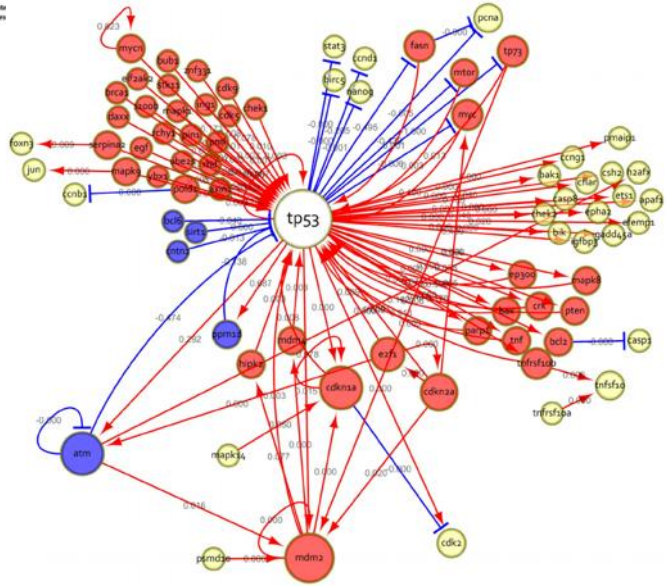
Traffic graph
(Visualization of global flights)



Electricity grid
(Blackout of August 14, 2003)



On-line behavior targeting



Gene analysis

Graph Processing: High Impact in Practice

Alibaba Group 阿里巴巴集团

Recommender System

Anti Fraud

Money-laundering detection

City Planning

Public Health

Social Marketing

Public Security

Knowledge Graph

Graph Processing

Infrastructure

Smart City

Data Security

Decision Support

FinTech

BioTech

WebTech

New business opportunities

Graph processing becomes extremely valuable in practice!

Graphs: Foundation of Future AI and Big Data?

Alibaba Group 阿里巴巴集团

Existing big data systems are no longer adequate

✕ Suboptimal performance

Ever more complex queries and AI model trainings could easily take hours or days even with hundreds of the state-of-the-art servers.

Novel graph systems

(1) are now 10-100x performant than 5 years ago, and still evolving fast.

(2) are proven to be able to speed up many AI and big data computations

✕ Hard to learn & use (high TCO)

Graph programming is hard. And building a real-life AI / big data application involves development across multiple languages and systems to gather, process and train data and models.

An easy to use system

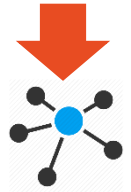
(1) Easy to design your own algorithms for parallel graph computation

(2) Automated / declarative parallelization (with little user involvement)

(3) End-to-end AI/big data solution in one box

✕ Big data and AI – a fragmented landscape

- **Data heterogeneity**: text, multimedia, graph, vector, SQL, streams
- **Fragmented tools**:
- **Lacks of a unified solution**



Graphs could provide a solution:

(1) graphs - an ideal way to unify heterogeneous data

(2) many AI and big data problems can be expressed as graphs

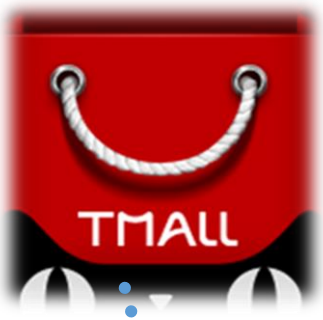
(3) more expressive than large vectors or SQL on tables

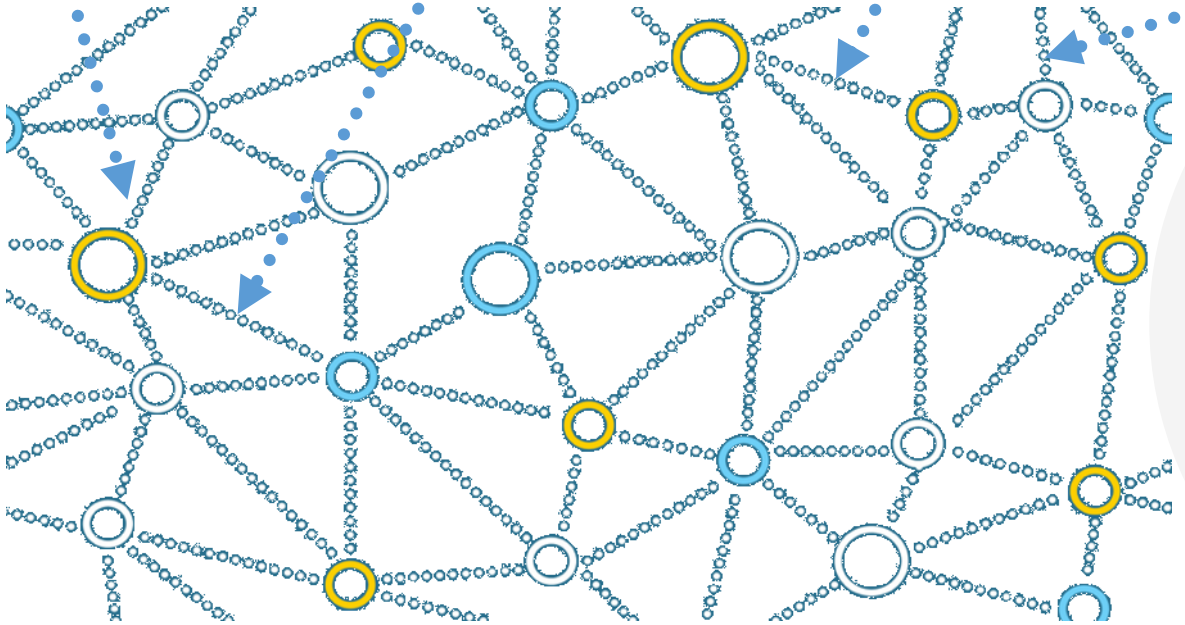
(4) industry starts to move towards this

Alibaba Ecosystem

New Capabilities Enabled by Massive Graph Analysis

Alibaba Group 阿里巴巴集团





- Billions of vertices
  - products, buyers, sellers, …
- Hundred billions of edges
  - clicks, orders, payments, …
- Real-time updates
  - e.g., 100K edge updates/s

Challenges for Graph Computing

Alibaba Group 阿里巴巴集团

- Heterogeneous data source
  - Social network, knowledge, transactions, etc.
- Large scale
  - Billions of edges and vertices
- Highly dynamic
  - Continuous updates at a tremendous scale
- Highly versatile
  - Graph traversal
  - Graph analytics
  - Graph learning
  - Graph mining

Common Architecture for Graph Computing

Real-time events
(On-line user behaviors, network events, etc.)

Requests

Updates

Other sources

Human knowledge

Business Operators

On-line reasoning

Real-time decisions

Data intelligence

Graph Engines

Off-line mining
(On a snapshot)

Data Scientists

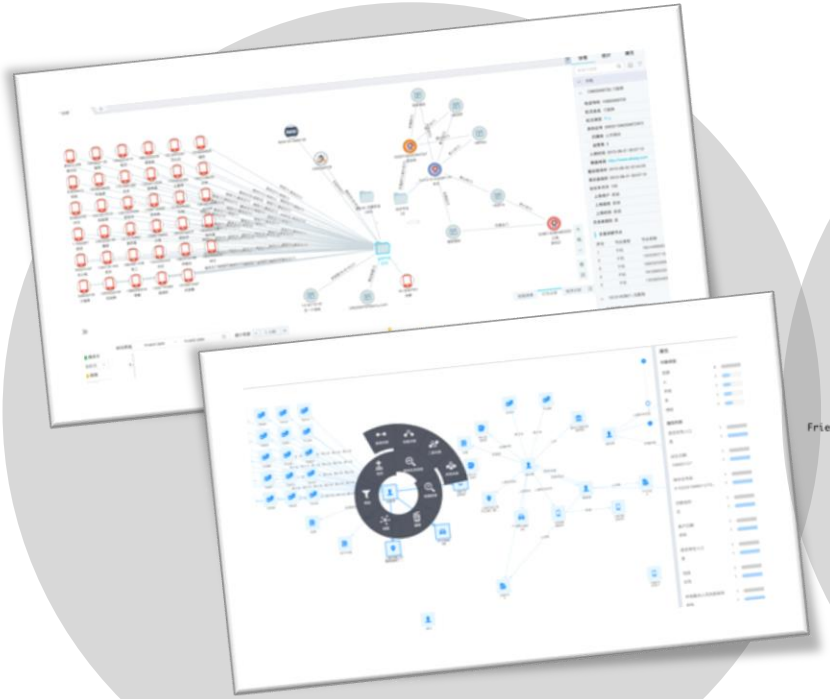
Agenda

Alibaba Group 阿里巴巴集团

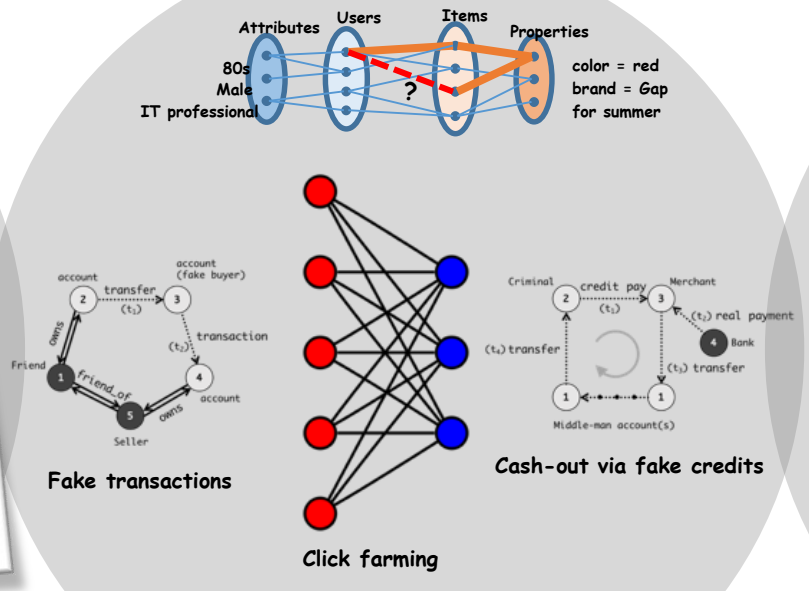
- Overview
- Challenges for graph computing
- Graph applications at Alibaba
  - Graph traversal
  - Graph pattern matching
  - Complex graph algorithms
  - Graph learning
- Conclusion

1：Distributed Graph Traversal

Graph Traversal - A Key Primitive for Interactive Exploration

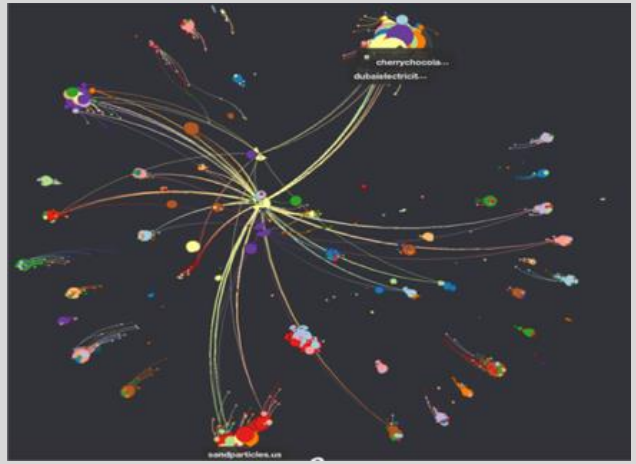


(1) Knowledge search & discovery



(2) Fraudulent activity detection in e-commerce

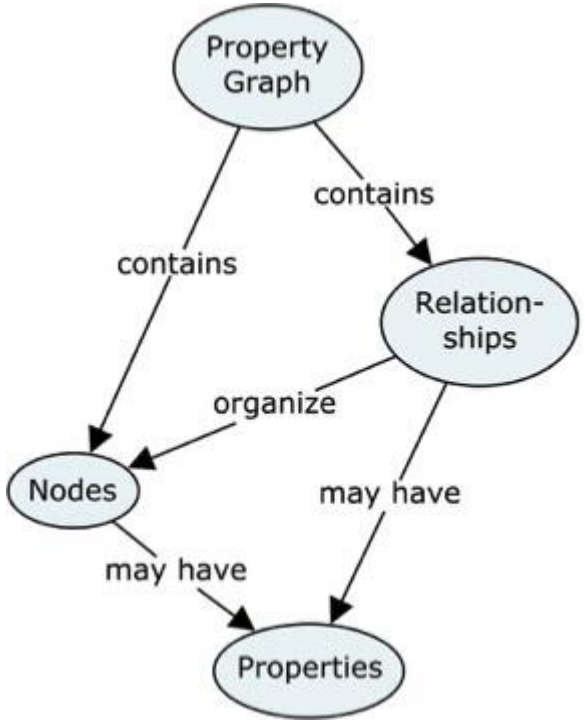
Find names of regex update[0-9]+.xyz.(com|net) and they are hosted by server 10.3.35.* and their nameserver is ns1.badnameserver.com.



(3) Cyber threat intelligence

Gremlin

- Gremlin [1] is the de-facto standard query language for graph traversal with rich expressivity



Property graph data model

Example: Cycle detection
Find all cycles of length `k+1`, starting from the persons with first name "`Tom`".

filtering

loops

```
g.V().has('firstname','Tom').as('a')
 .repeat(out().simplePath()).times(k)
 .where(out().eq('a'))
 .path()
```

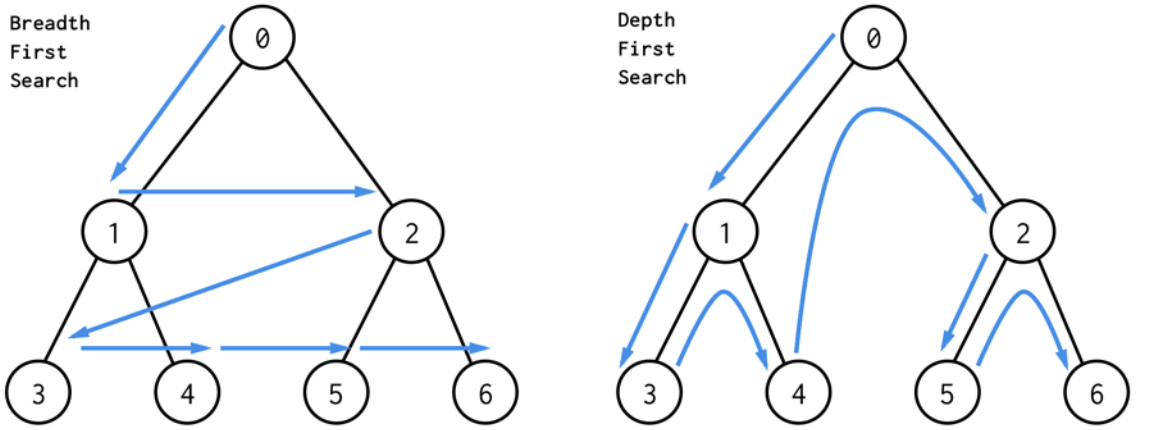
conditional

1. Marko A. Rodriguez. 2015. The Gremlin graph traversal machine and language. In Proceedings of the 15th Symposium on Database Programming Languages (DBPL), 2015

Graph Traversal Strategies

Alibaba Group 阿里巴巴集团

- General graph traversal strategies
  - Breadth-first Search (`BFS`)
  - Depth-first Search (`DFS`)

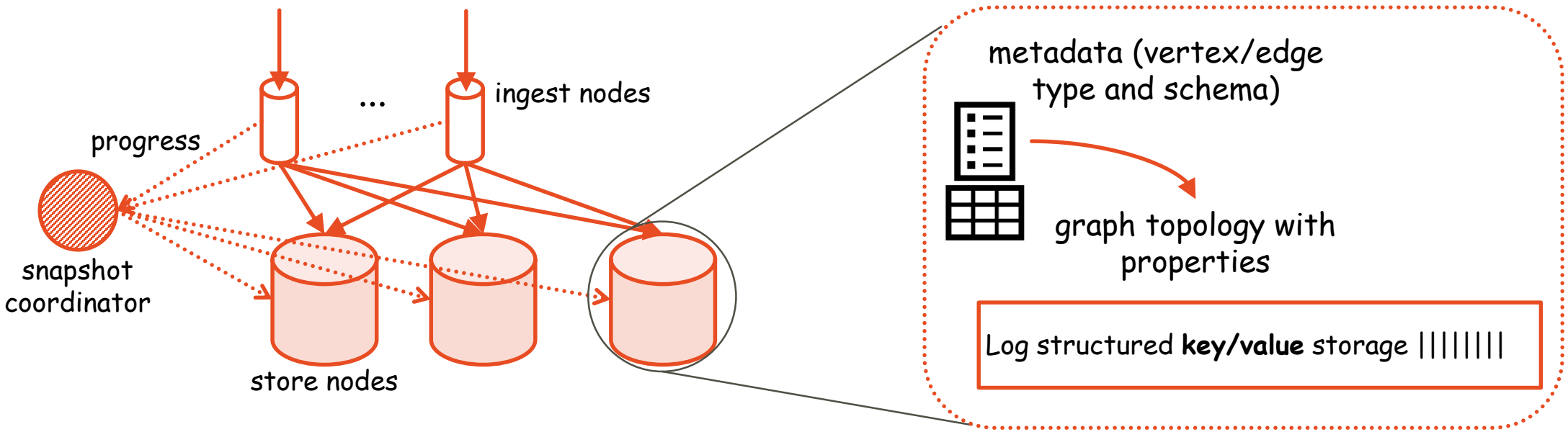


http://mishadoff.com/blog/dfs-on-binary-tree-array/

- Challenges
  - `BFS` could result in unbounded memory
    - All vertices at level `k-1` must be buffered before visiting a vertex at depth level `k` (i.e., the space complexity could be up to `O(n)`, where n is the number of vertices)
  - `DFS` is hard to parallelize
    - The backtracking in `DFS` is inherently sequential

We address the challenges using dataflow (with dynamic scheduling)

Distributed Graph Store

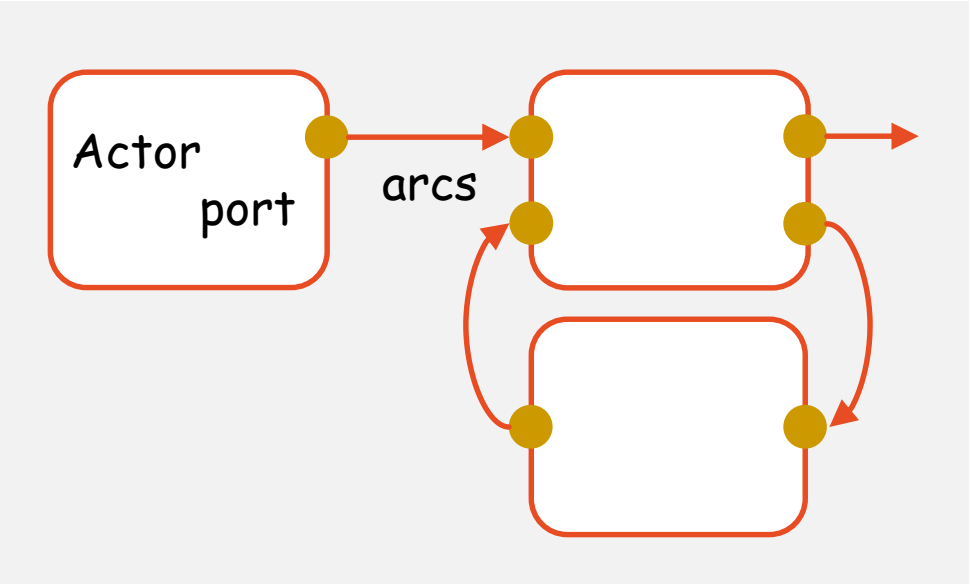


- Key design choices
  - Partitioning a large graph across distributed memory/SSDs
  - Real-time updates with **snapshot isolation** + fault tolerance
  - **Log structured** storage with efficient graph encoding

Dynamic Dataflow Model of Computation

- Dynamic dataflow [1]
  - Nodes: Actors with ports
  - Arcs: FIFO connections between the actors
    - No communication path exists between actors other than the data streams

- Execution semantics
  - An actor gets **fired** when data becomes available on one or more arcs
  - It can consume more than one tuples on one or more arcs per execution
  - A dataflow terminates if there are no executable actors



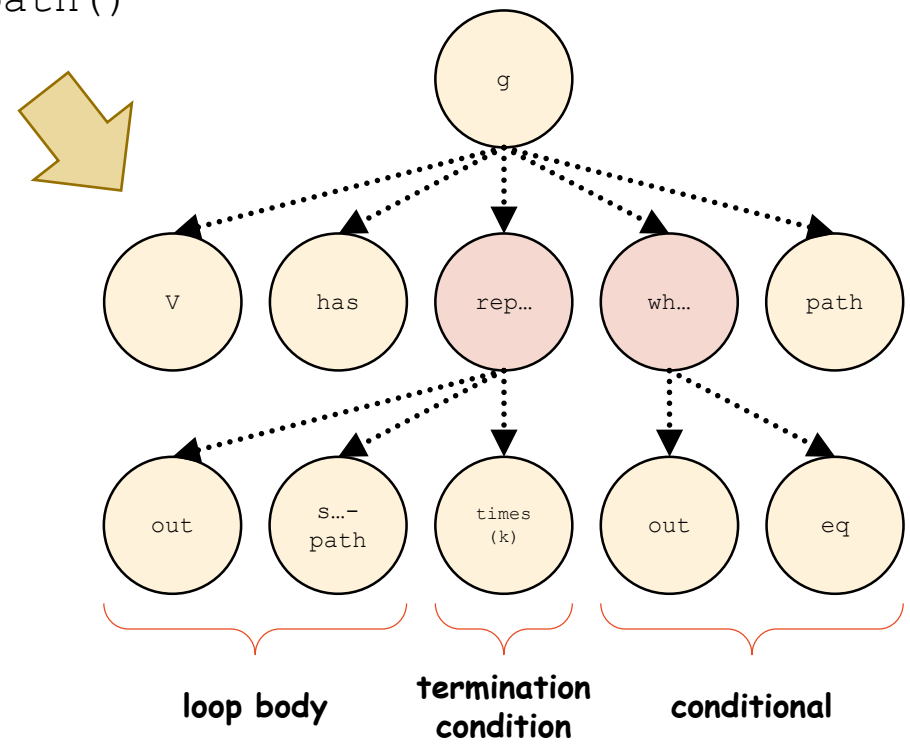
Actor-oriented design: dynamic dataflow

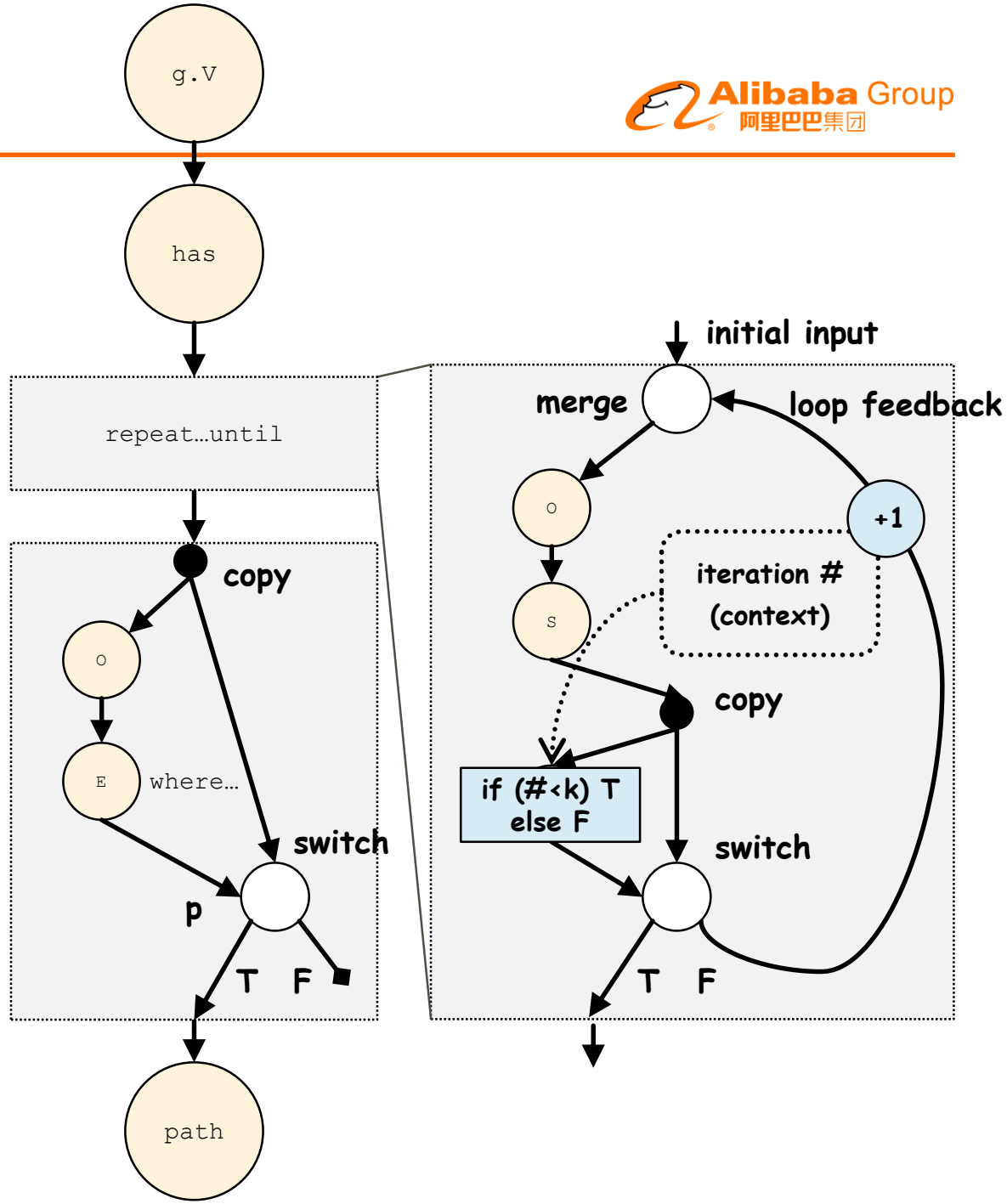
1. Arvind and R. S. Nikhil, Executing a program on the MIT tagged-token dataflow architecture, in IEEE Transactions on Computers, vol. 39, no. 3, pp. 300-318, March 1990

Dataflow Compilation

- A `Gremlin` query can be represented as an ordered tree of traversal steps, e.g.,

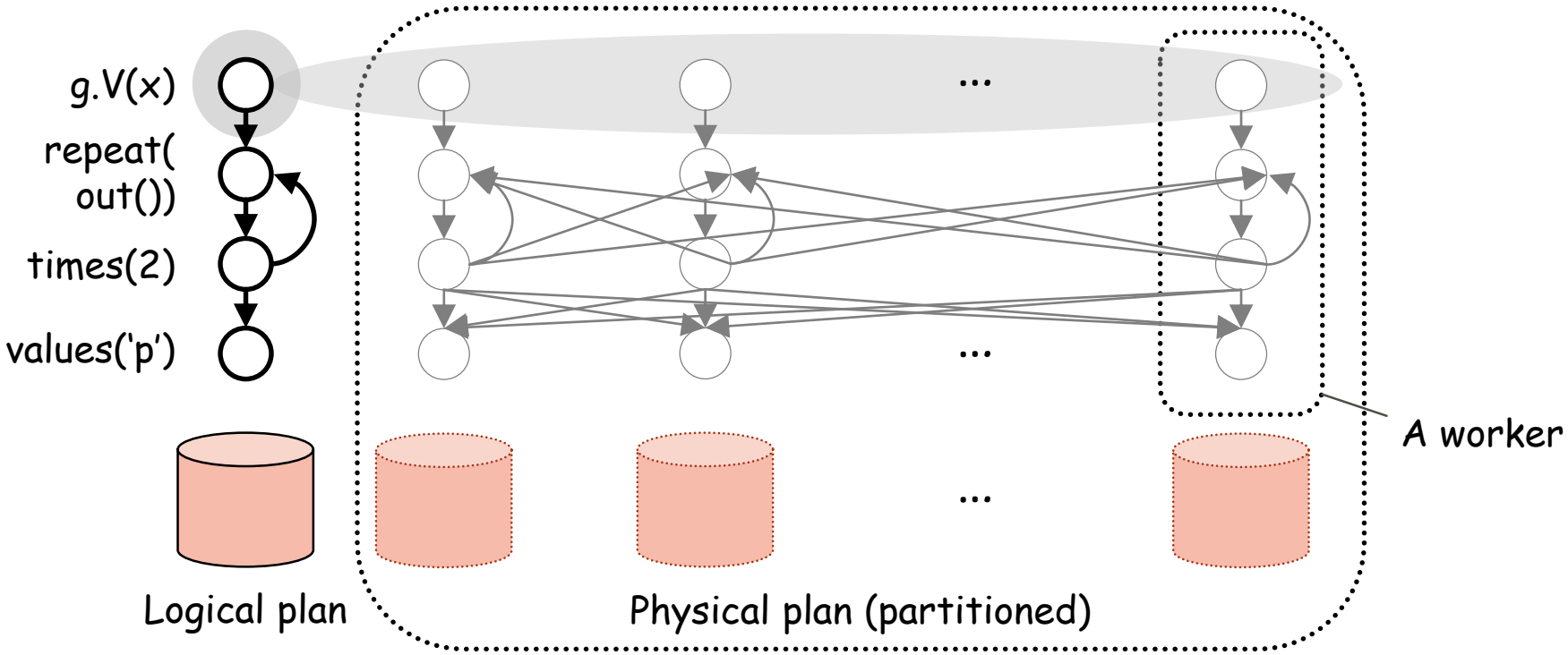
```
g.V().has('firstname','Tom').as('a')
 .repeat(out().simplePath()).times(k)
 .where(out().eq('a'))
 .path()
```





Parallel Execution

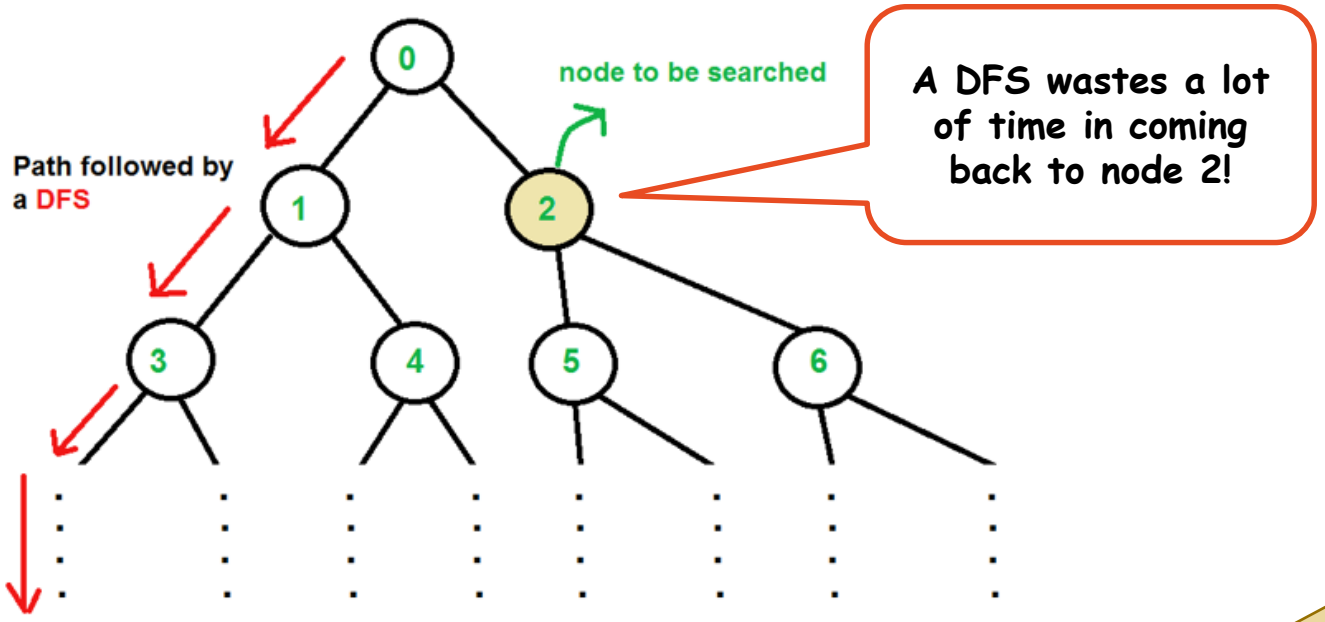
- Dataflow partitioning for parallel execution
  - Each graph partition has a query worker
  - Each worker runs a copy of the dataflow, processing vertices belonging to that partition in parallel
  - The workers exchange data among directly-connected actors if needed

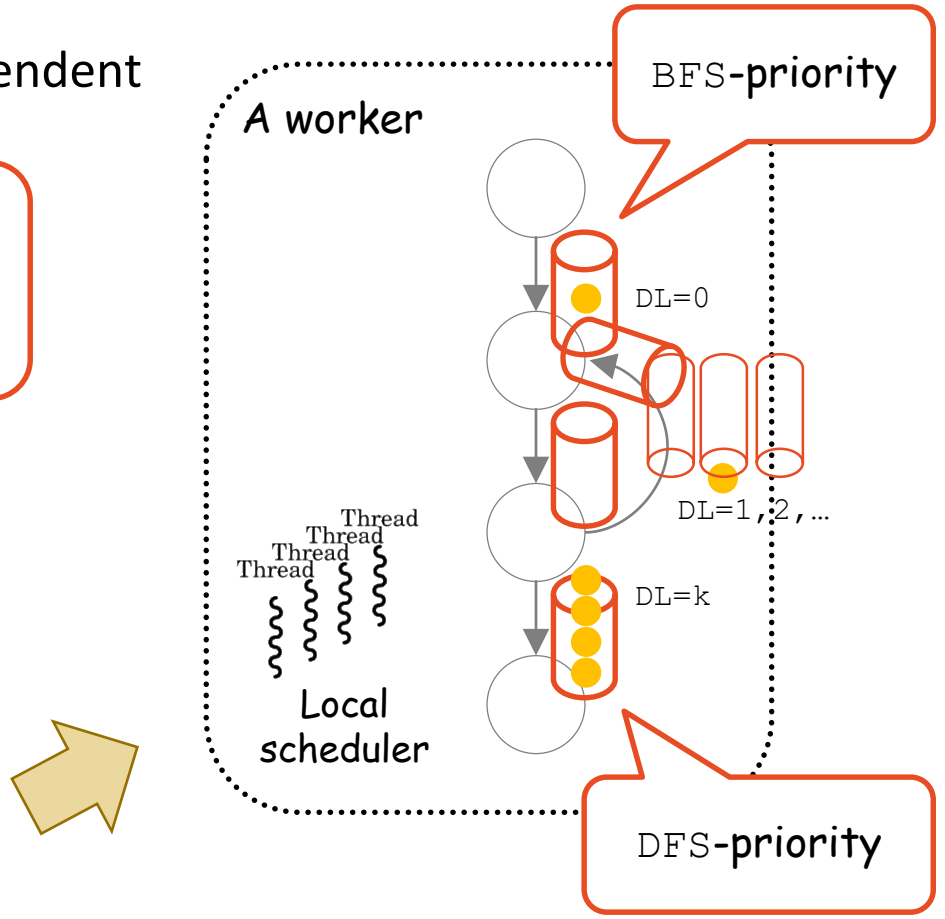


Optimizing Traversal Strategies

Alibaba Group 阿里巴巴集团

- Optimal graph traversal strategy is query- and/or data-dependent





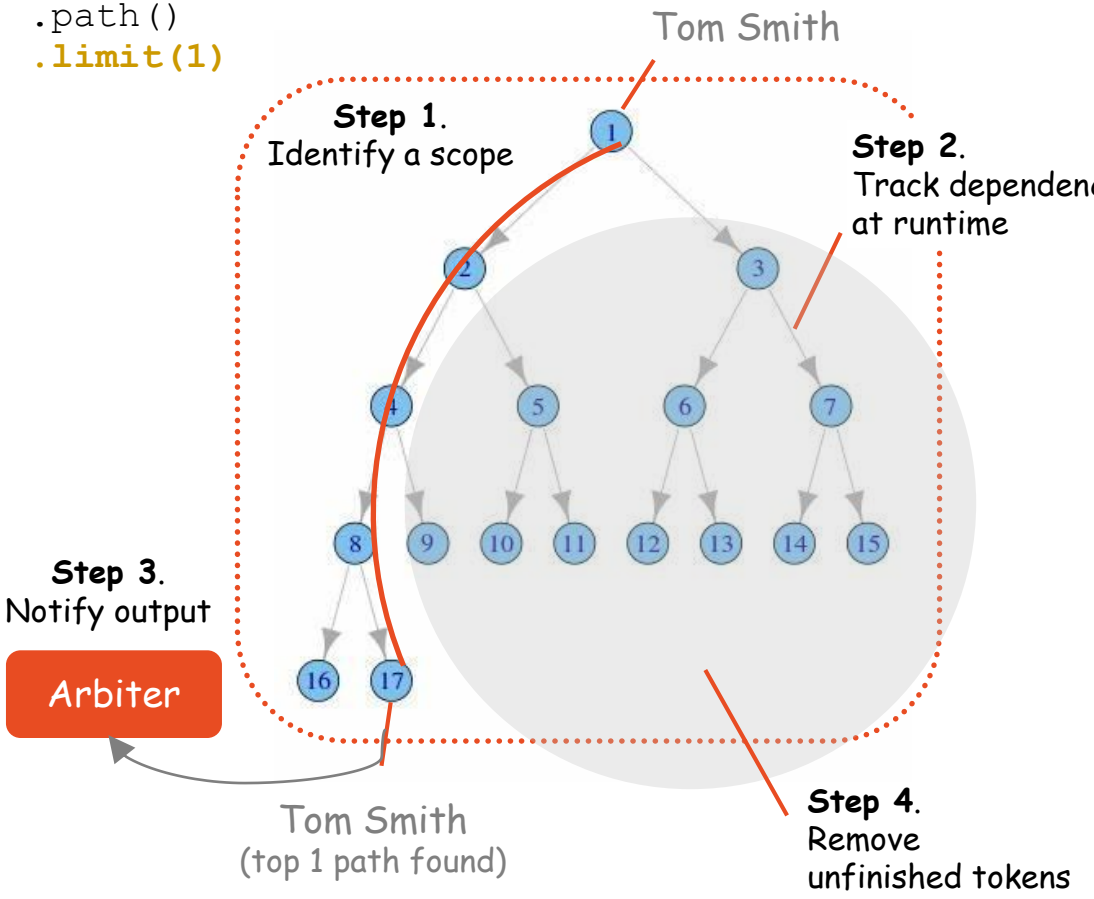
- We support different traversal strategies using scheduling
  - **Depth-first**: prioritize tuples with **large** depth levels
  - **Breadth-first**: prioritize tuples with **small** depth levels

All strategies can take advantage of additional parallelism via dynamic scheduling!
Remaining challenge: memory management!

Further Optimization: Early Out

- In many cases, we only need **top K** results from a traversal, e.g.,

```
g.V().has('firstname','Tom').as('a')
 .repeat(out().simplePath()).times(3)
 .where(out().eq('a'))
 .path()
 .limit(1)
```



Dynamic scheduling makes it possible!

To avoid wasted traversal, compiler/runtime

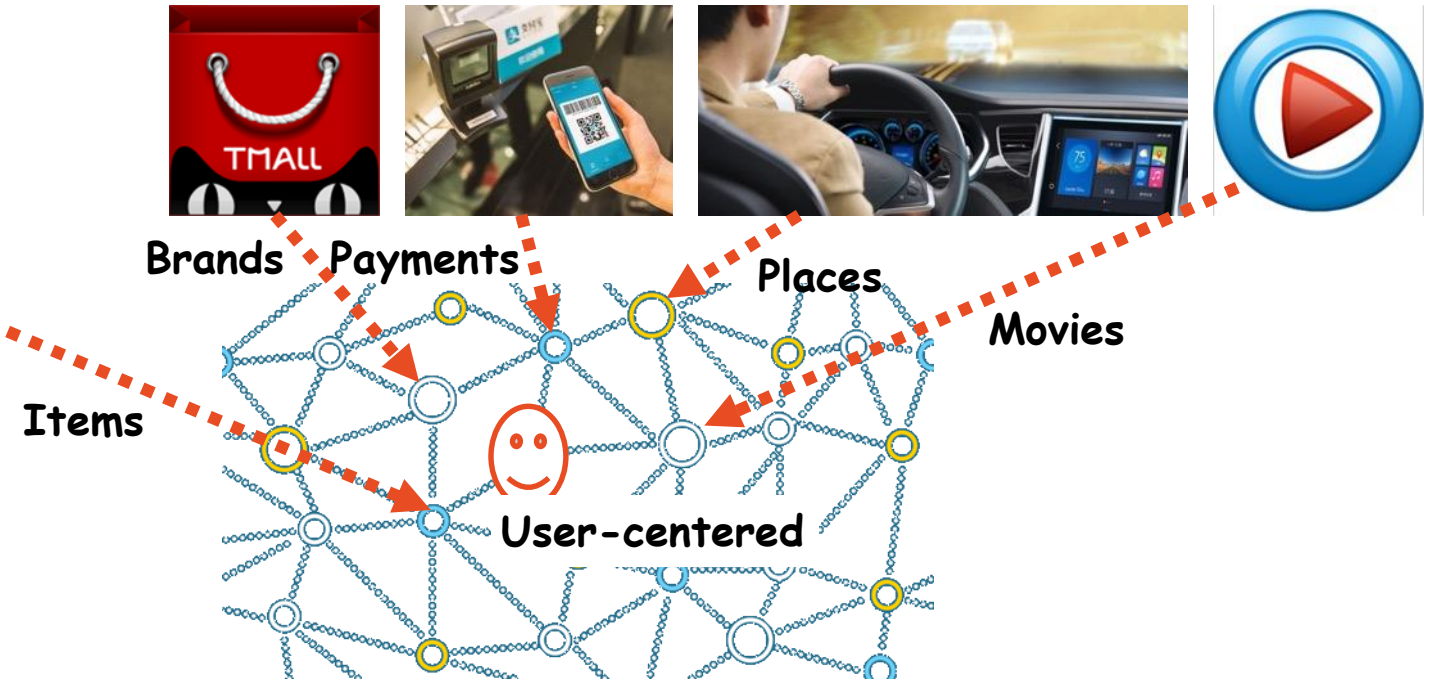
1) Identify early-stoppable scopes (i.e., subgraphs containing a `limit(k)` or conditional)
2) Track input/output dependencies
3) Notify an arbiter (= compiler-generated state machine) of any output from scopes
4) Upon early stop, the arbiter notifies scheduler to remove any dependent, unfinished tuples

(Scopes can be nested arbitrarily)

2: Optimizing Pattern Matching in Large Dynamic Graphs

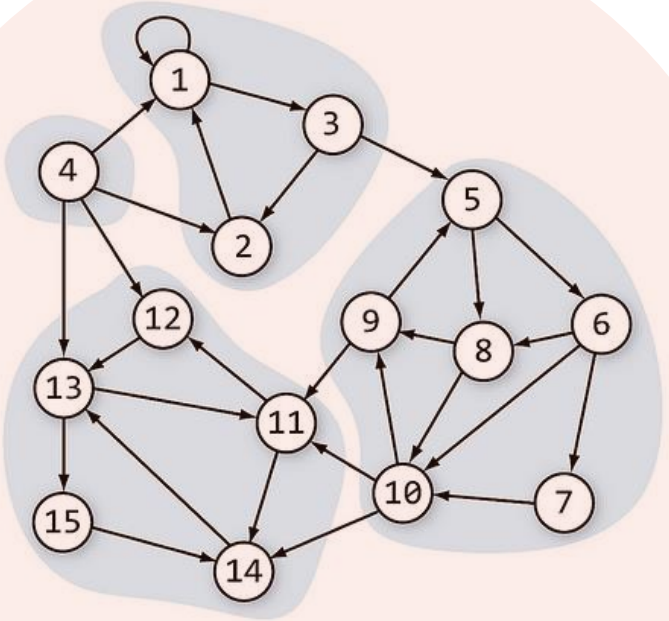
Motivation

A user behavior repository for buyers, sellers, products…, and their interactions (e.g., item clicks, orders…)

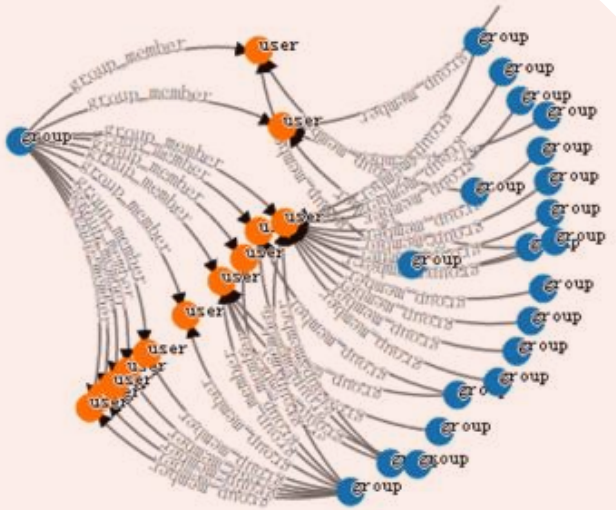


Leverages the patterns of the interconnected entities for fraud detection

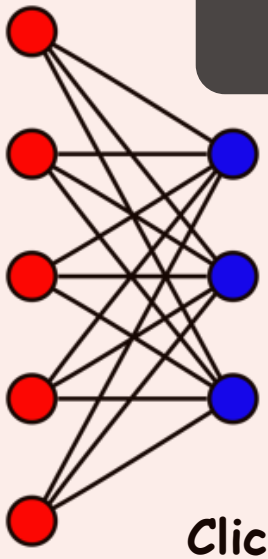
Complex Graph Pattern Matching



Community detection



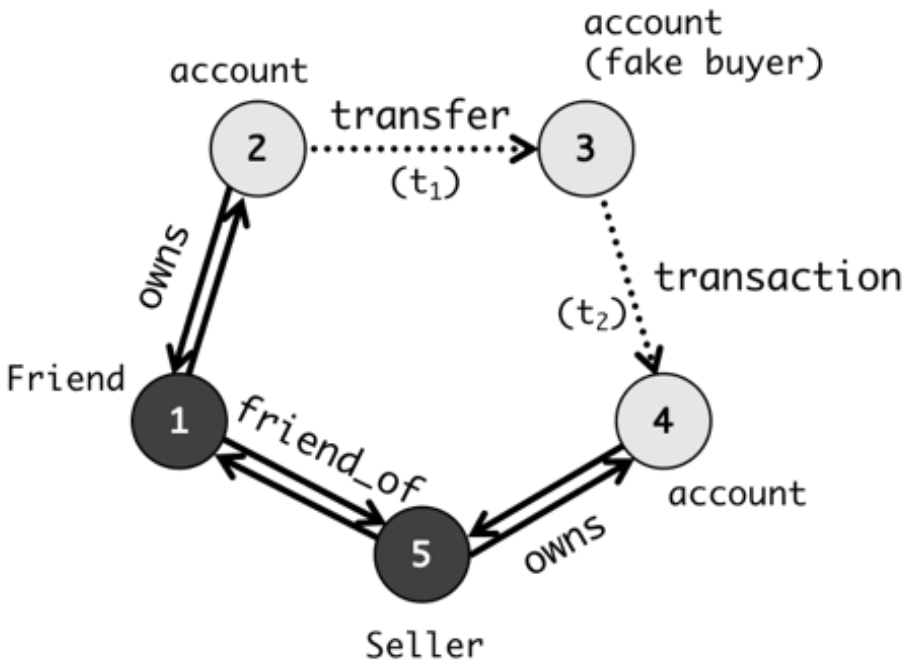
On-line gambling



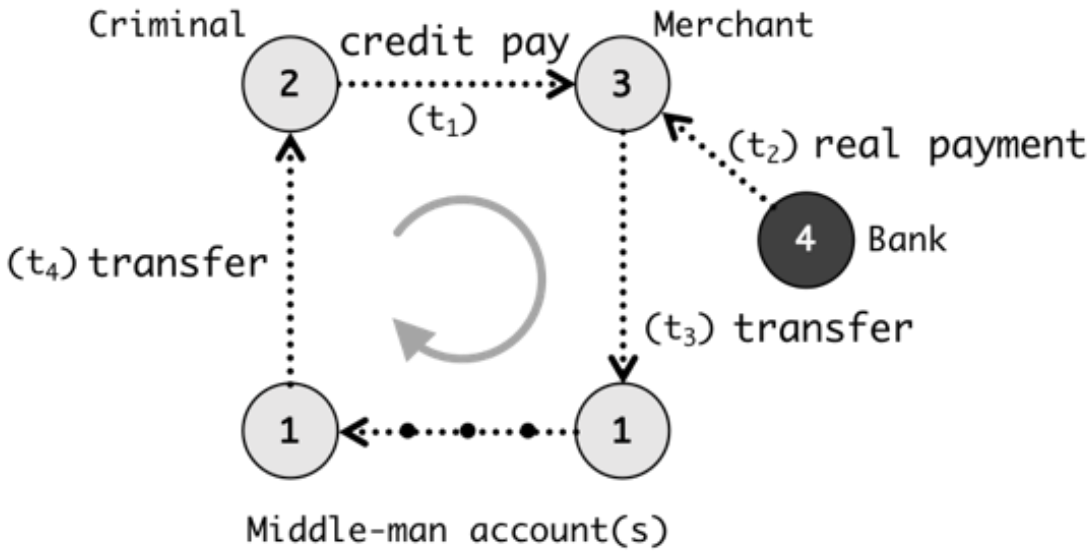
Click farming

Coupon fraud

Cycle Detection



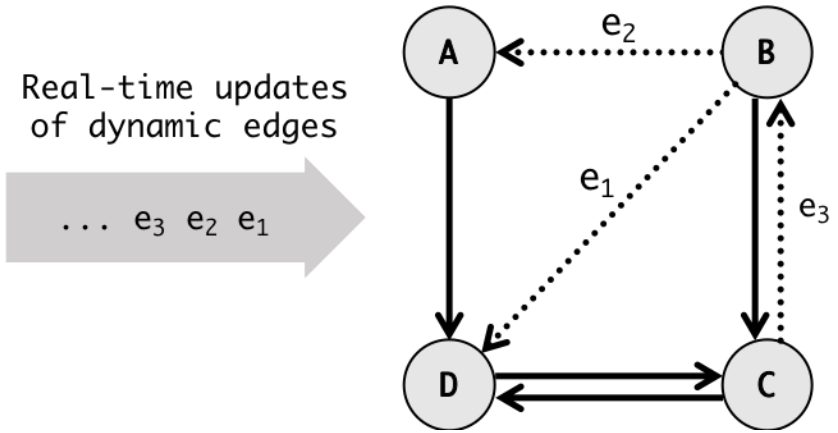
Fake transaction



Cash-out via fake credits

Xiafei Qiu, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, Jingren Zhou: Real-time Constrained Cycle Detection in Large Dynamic Graphs. PVLDB 2018

Problem Definition

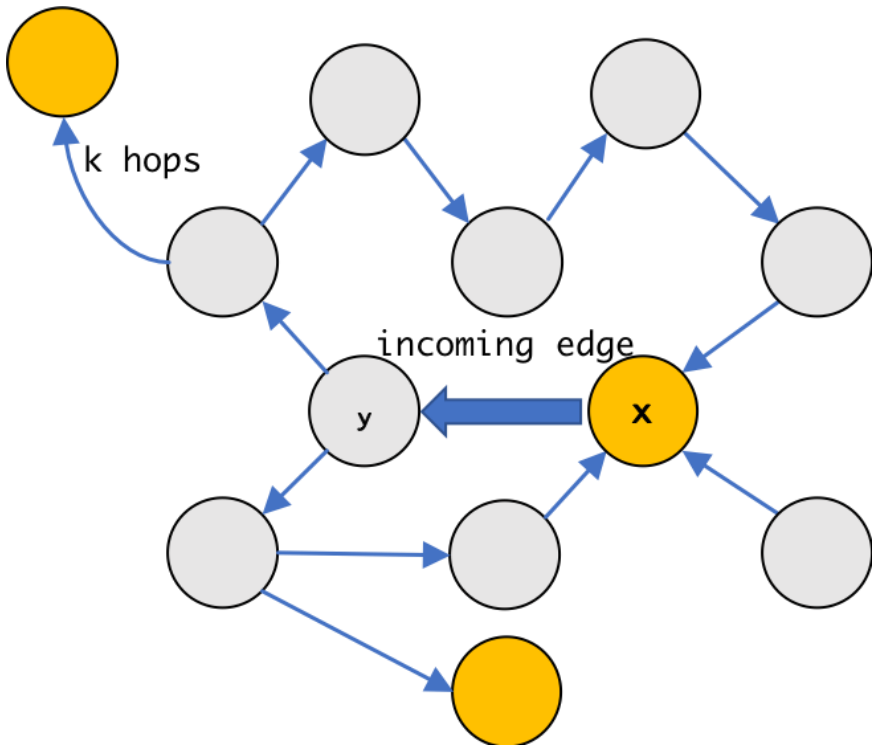


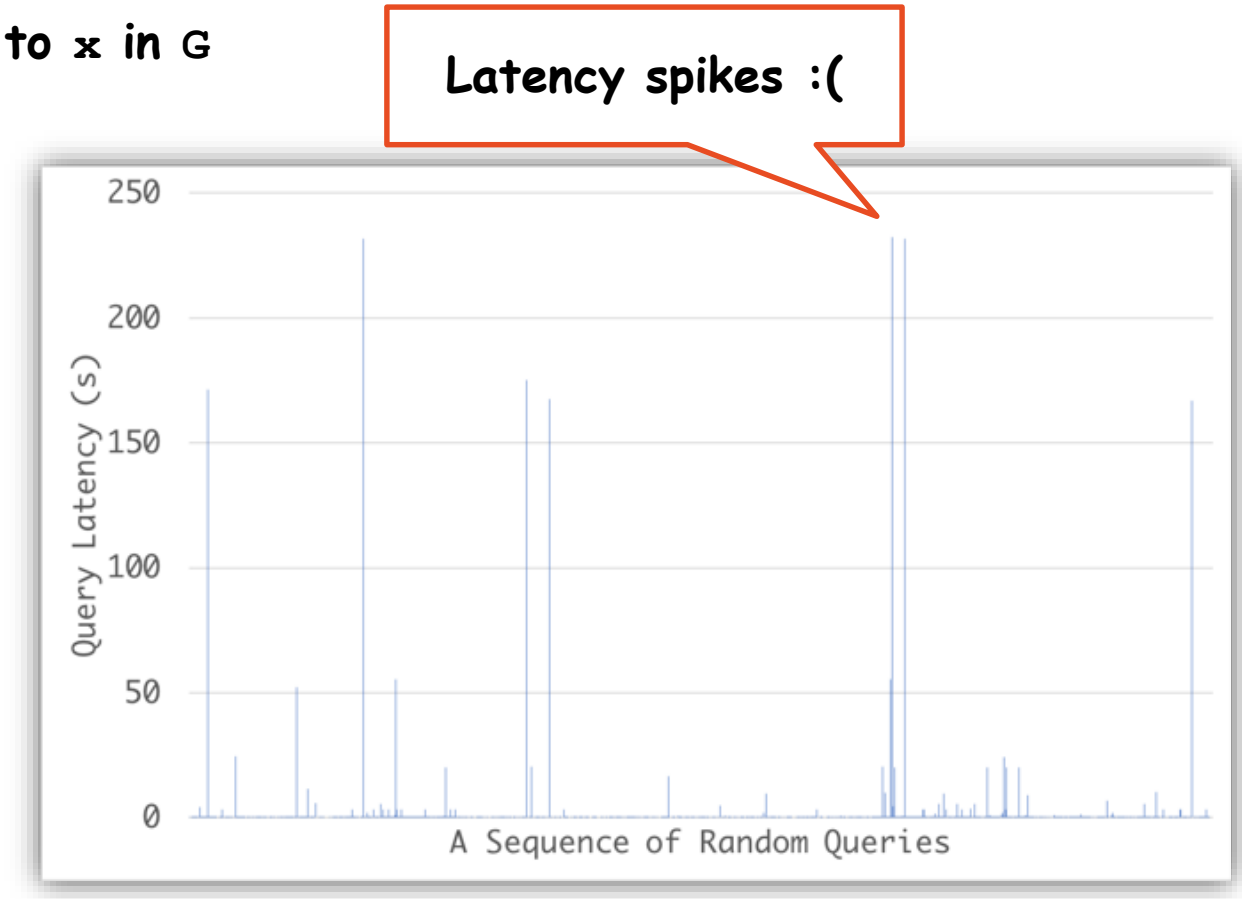
- A directed graph G
  - Size: 500 million vertices, 2 billion edges
  - Real-time updates: 20,000 edges/s at peak
    - Retention period: 2 days

- Continuous/incremental cycle detection
  - **Given G and one incoming edge x->y, find out if there is a circle within length k**

Baseline Solution

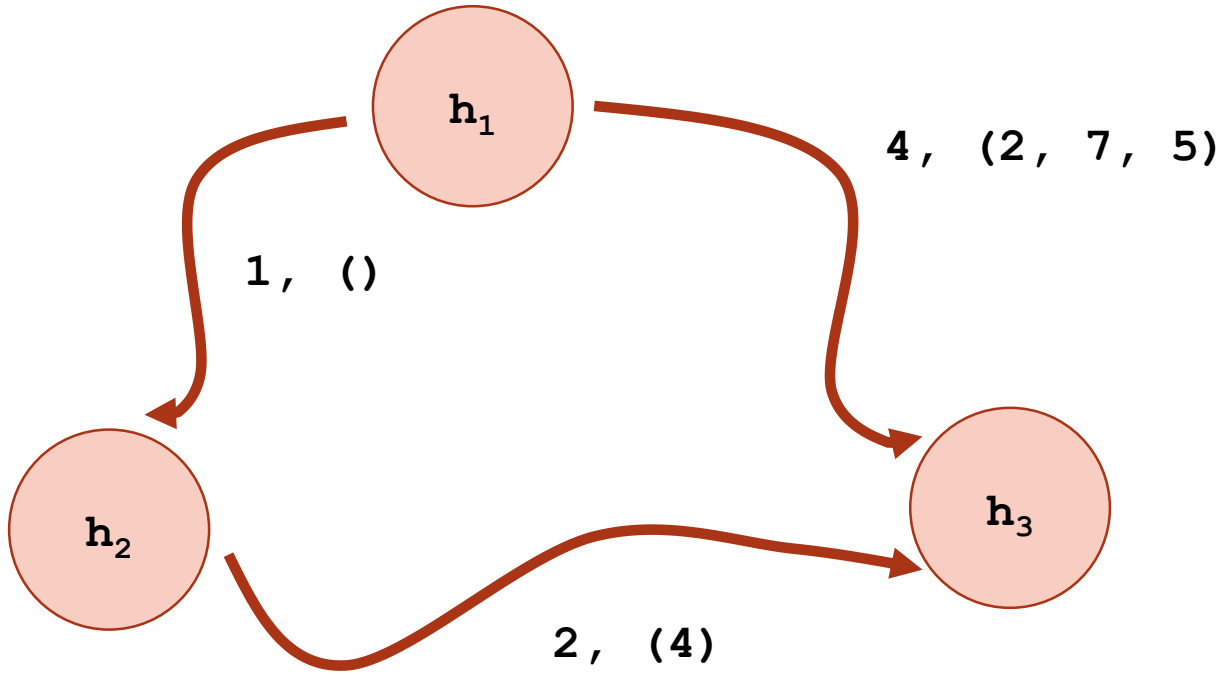
- FOREACH incoming edge x->y
  - **DO a depth-first search (DFS) from y to x in G**





Hot Point-based Index

- A set of hot points $\{h_i \mid d(h_i) \geq t\}$
  - **All the paths with length no larger than k among them**

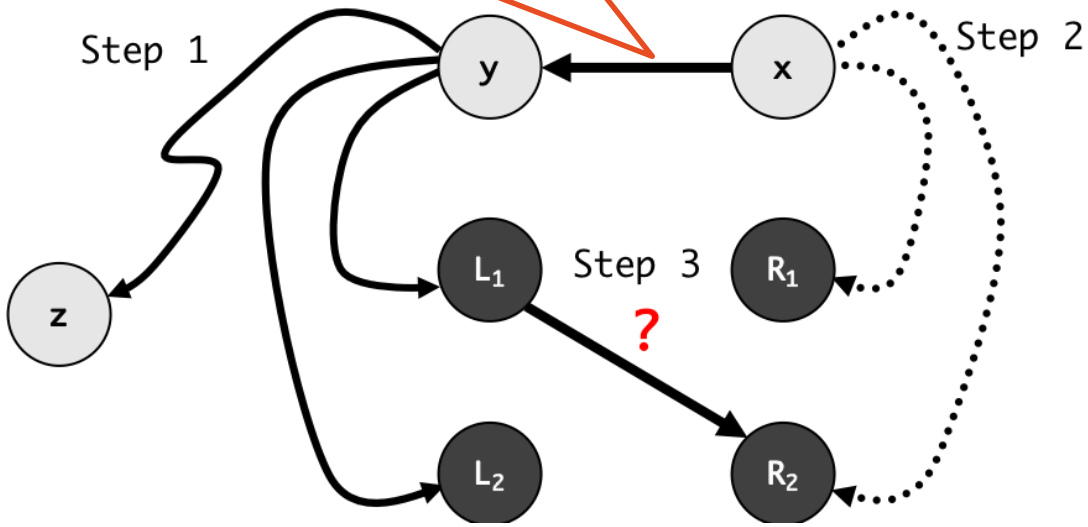


Search Algorithm

- **Step 1**: Start from `y` BFS, stop if
  - (i) reach `x` ,
  - (ii) already length `k` or
  - (iii) meet hot point $\{L_i\}$

Index update: a by-product of the search algorithm

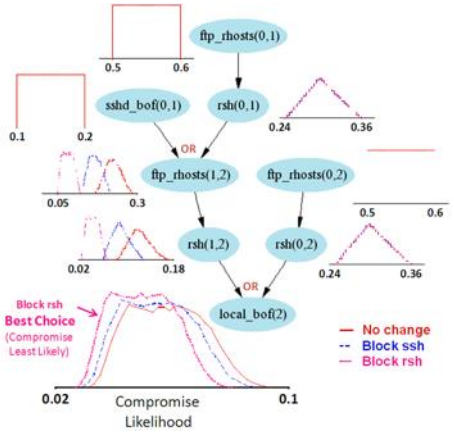
- **Step 2**: Start from `x` BFS on inversed graph `G'` , stop if
  - (i) already length `k-1` or
  - (ii) meet hot points `{Ri}`



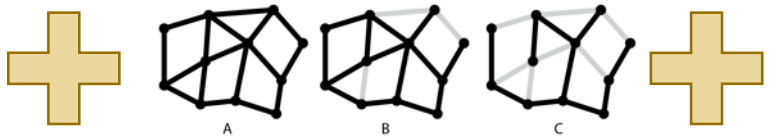
- **Step 3**: Identify the remaining circles involving more than one hot points by utilizing `HP-Index`

Further Optimizations

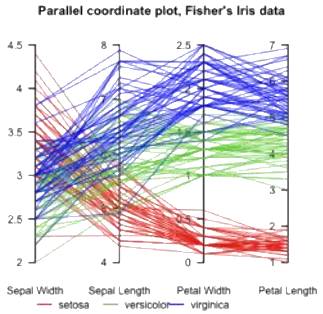
- Extend the approach to a class of (fuzzy) complex pattern matching
  - Automatically analyze and generate query-specific indices
  - Optimizations are transparent to users



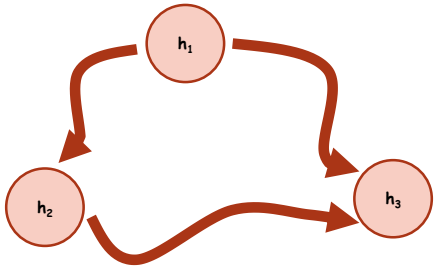
Graph data measures



(Gremlin) query patterns



Profiling/runtime statistics



Auto-generated & self-adaptive graph indices

3: Parallelizing Graph Computations

Graph Analytics

Alibaba Group 阿里巴巴集团

Analytics on graphs has been studied for decades

- ✓ **General analytics:** PageRank, shortest path, maximum flow, minimum spanning tree…
- ✓ **Community detection:** maximum clique/bi-clique, connected components (Shiloach-Vishkin's Algorithm), triangle counting, label propagation algorithm. …
- ✓ **Graph mining:** structure mining, graph pattern discovery,…
- ✓ **Graph keyword search**
- ✓ **Structure prediction**
- ✓ **…**

Many might work on small graphs, but on big graphs, it will be a different story

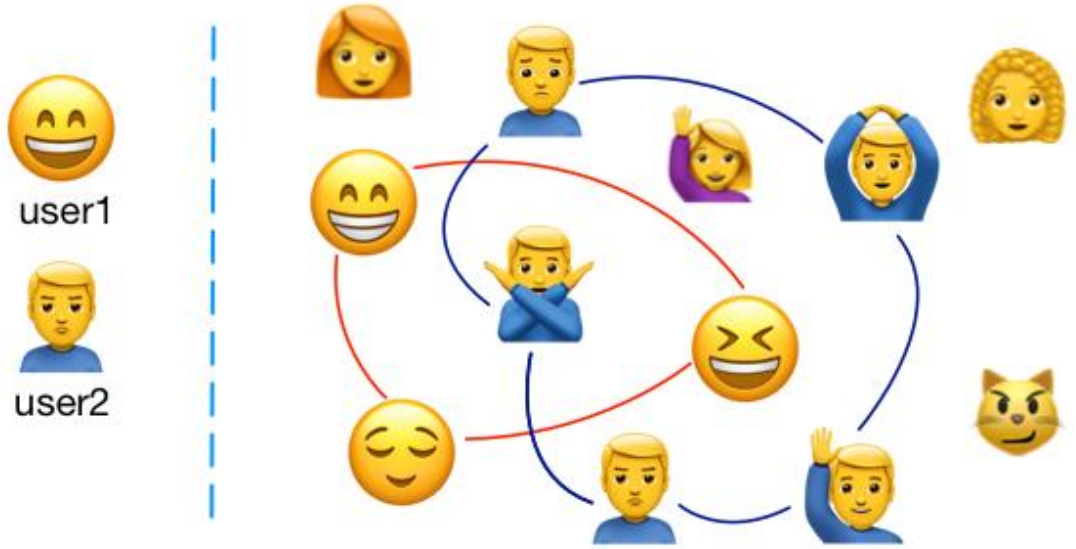
- ✓ A declarative and standardized programming paradigm is not yet in place
- ✓ Graph computations are essentially iterative and recursive (high cost)
- ✓ On large graphs (billions of nodes, trillions of edges), parallel computation is a must

Goal: How to parallelize existing graph algorithms?

Graph Analytics in Action: Entity Resolution

Alibaba Group 阿里巴巴集团

- **Goal:** identify and link/group different manifestations of the same real world object.
  - ✓ Data from **a large number** of online and offline services, ranging from *online marketplace*, *travel agency* and *video streaming* to *local supermarkets*, *food delivery* and *cinema ticketing*
  - ✓ **Question**: how many unique users, items, business-clients are there across those services?
  - ✓ A common problem faced by many internet companies

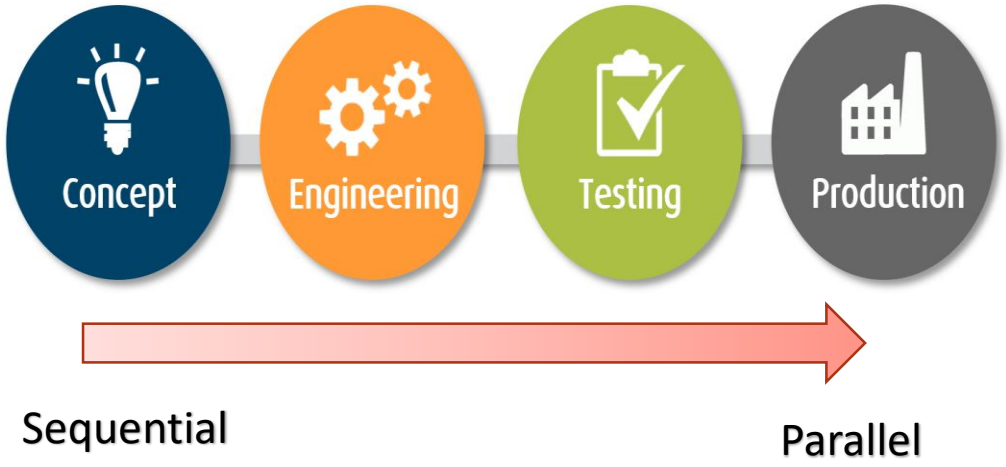


- **Challenges**:
  - Iterative and intensive computation
  - Large scale and heterogeneous data, dynamic changes (PBs of data with billions of records across hundreds data sources with TBs of update daily)
  - Multi-domain connections/relations between possible entities
  - Noises and errors, incomplete/missing data

From Sequential to Parallel

A wide range of graph algorithms are developed to tackle the problem:

- **Vertices** – possible entities extracted from 100+ heterogeneous data sources
- **Edges** – possible connections between those entities
- **Scale** – 10+ billion vertices, 100+ billion edges, and keep growing
- Various sophisticated graph algorithms were designed on this graph for *link prediction* (for completing missing links), *shortest path* (for computing weighted distance), *fuzzy transitive closures* (for merging equivalent entities), …
- By the end of the **ER** process, equivalent/duplicated entities are merged together

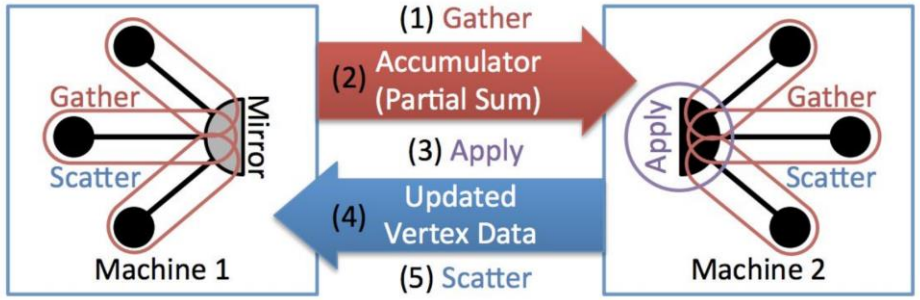


- New ideas are easier to try and test in **sequential** environment
- Many algorithms might already exist a **sequential** version

Nontrivial to be parallelized…

Vertex-Centric Model Systems

- ***"Think like a vertex"*** paradigm
- **Simple APIs**, with which some algorithms can be neatly written (e.g., CC, PageRank)
- **Industry systems:** Pregel/Giraph, GraphX, …



We used to work on an in-house vertex-centric graph system to parallelize the entity resolution. **However, ever-growing challenges emerged over the years**.

- **Hard to program**. Sometimes **impossible** to recast sequential algorithms without a **complete re-design**. Especially challenging for data scientists who are not trained with parallel programming.
- **Constant struggles over ad-hoc trade-offs.** For many algorithms, to parallelize them efficiently in vertex-centric model often means **losing precision or quality**. Worse still, users often have **little** control over the process.
- **Time-consuming and huge cost.** It often took **several** senior engineers **months of effort** to bring an algorithm online.
- **Subpar performance.** A single parallel program can take **7+ hours** over a graph with **90B edges** just to get a rough result with **degraded quality**. End-to-end **ER** process can easily **take days to finish** even with thousands of processors. Worse still, some expensive computations are not viable due to the rapid burst of messages.

https://www.cl.cam.ac.uk/~ey204/teaching/ACS/R244_2018_2019/presentation/S3/PREGEL_Vikash.pdf

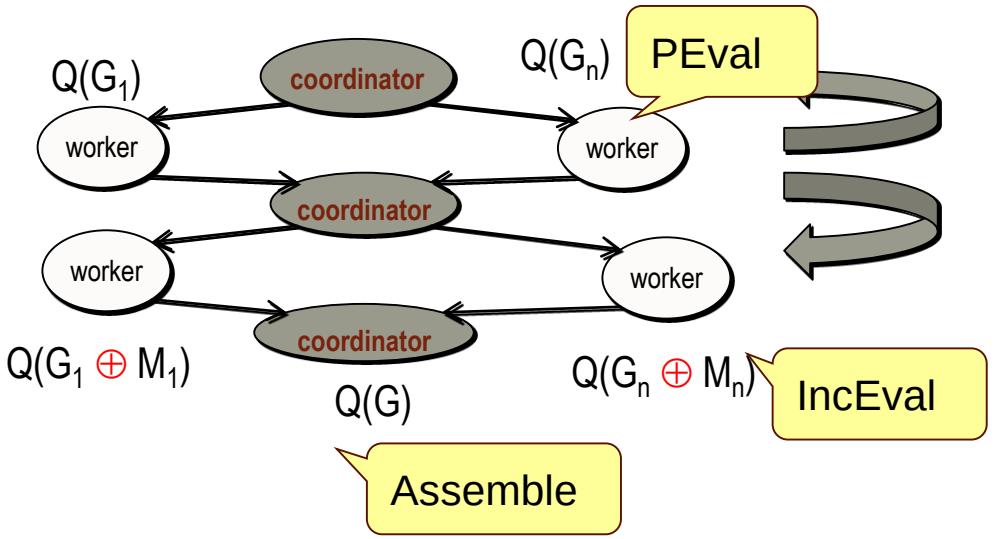
Alternative to Vertex-Centric? - PIE Model

- PIE - a graph parallel computation paradigm originally presented in the paper

  Wenfei Fan, Wenyuan Yu, Jingbo Xu, Jingren Zhou, Xiaojian Luo, Qiang Yin, Ping Lu, Yang Cao, Ruiqi Xu: Parallelizing Sequential Graph Computations. ACM Trans. Database Syst. 43(4)

- To compute Q(G), users to provide 3 functions:
  - ✓ **PEval**: a (sequential) algorithm for Q, for partial evaluation
  - ✓ **IncEval**: a (sequential) incremental algorithm for Q
  - ✓ **Assemble**: a (sequential) algorithm (often just taking a union of partial results)

Data partitioned parallelism: Fragmented graph G = $(G_1, \ldots, G_n)$, distributed to workers



evaluate Q on smaller G_i

① Partial evaluation PEval: evaluate $Q(G_i)$ *in parallel*

② Repeat incremental IncEval: compute $Q(G_i \oplus M_i)$ *in parallel*, by treating messages M_i between different workers as "updates"

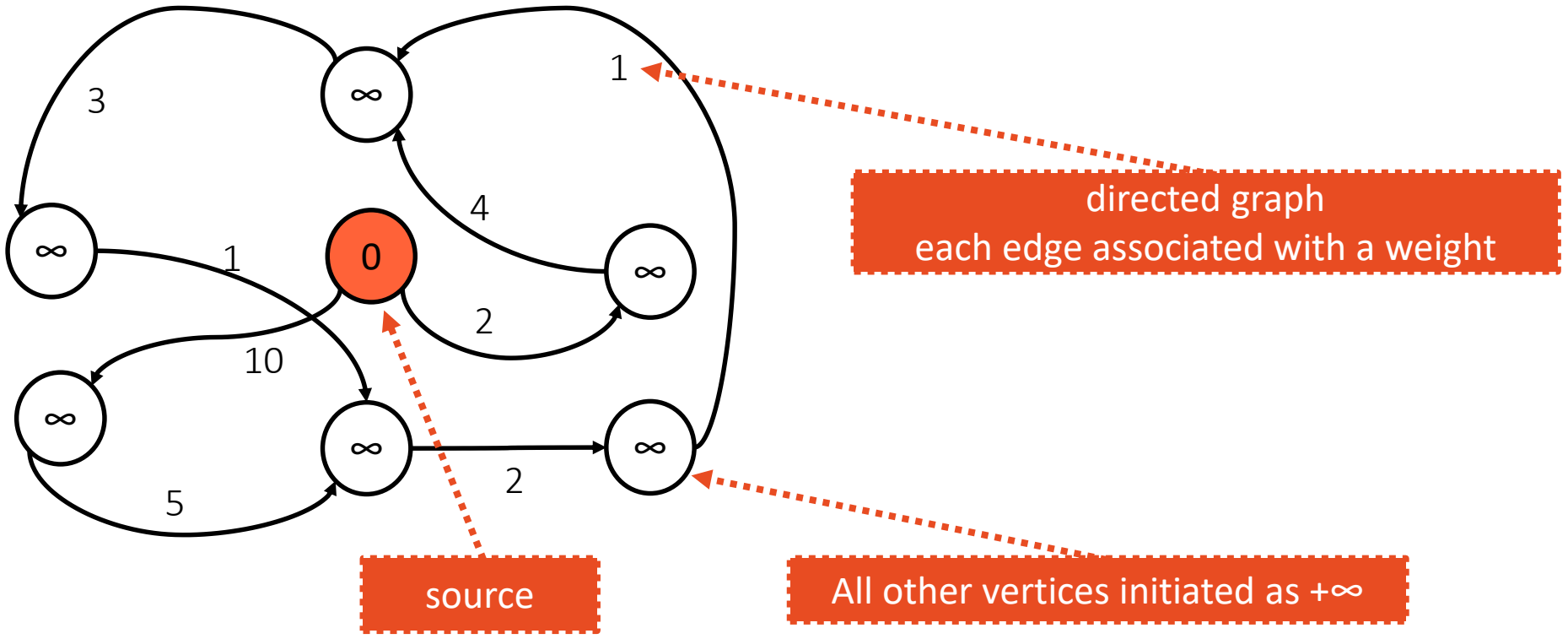
Messages M_i: aggregated partial results pertaining to border vertices

③ Assemble partial results when it reaches a "fixpoint"

Entity Resolution - Single Source Shortest Path

Key observation: if the weighted distance between two vertices is small, chances are they could be of the same entity

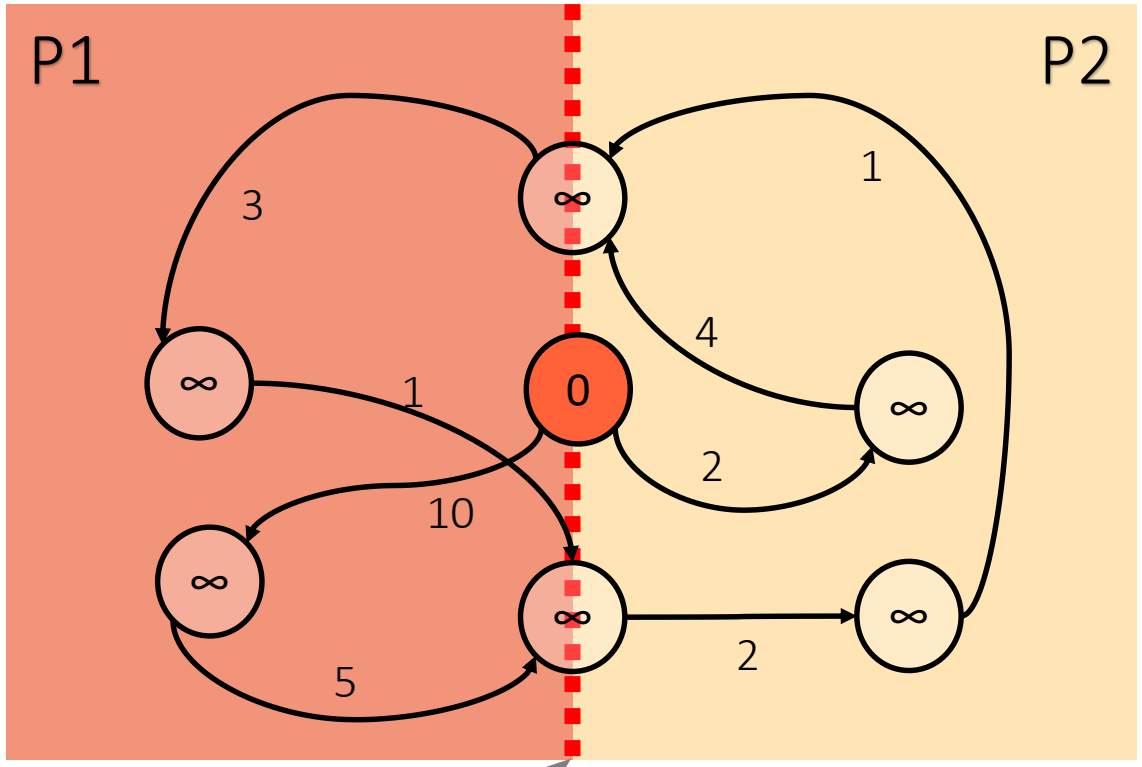
- SSSP – to compute the shortest paths between a given source and other vertices



Entity Resolution - Single Source Shortest Path

Key observation: if the weighted distance between two vertices is small, chances are they could be of the same entity

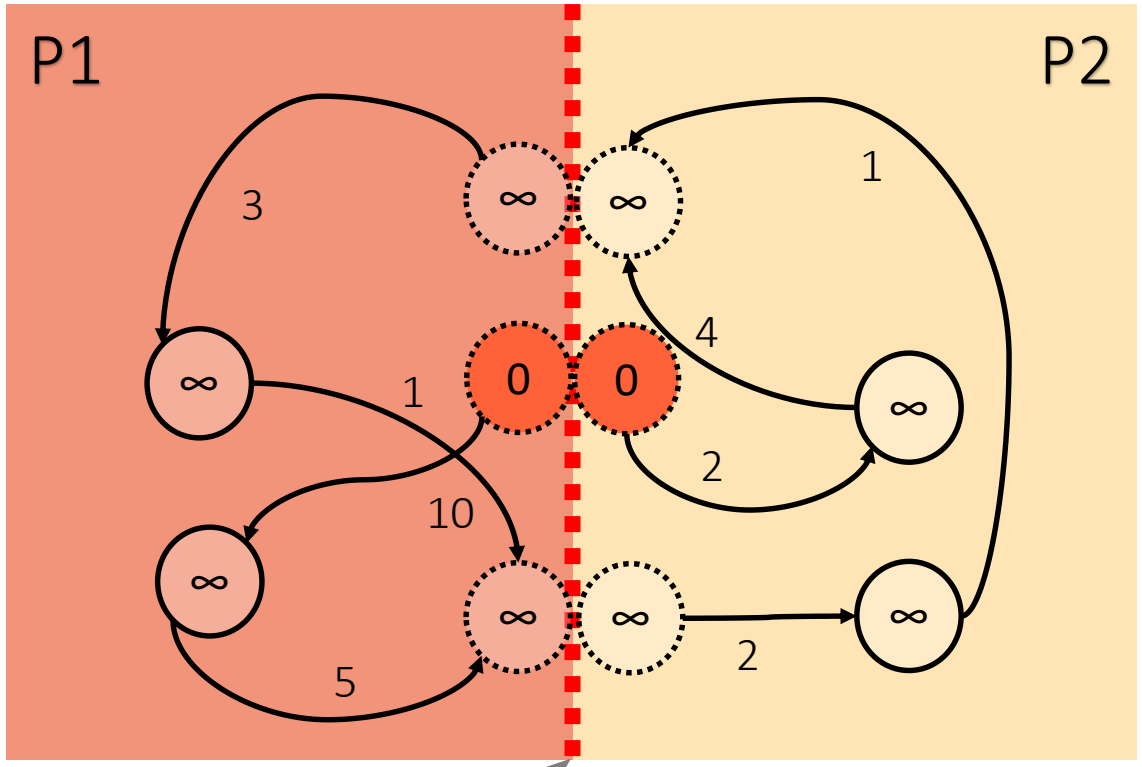
- SSSP – to compute the shortest paths between a given source and other vertices



Entity Resolution - Single Source Shortest Path

Key observation: if the weighted distance between two vertices are small, chances are they could be of the same entity

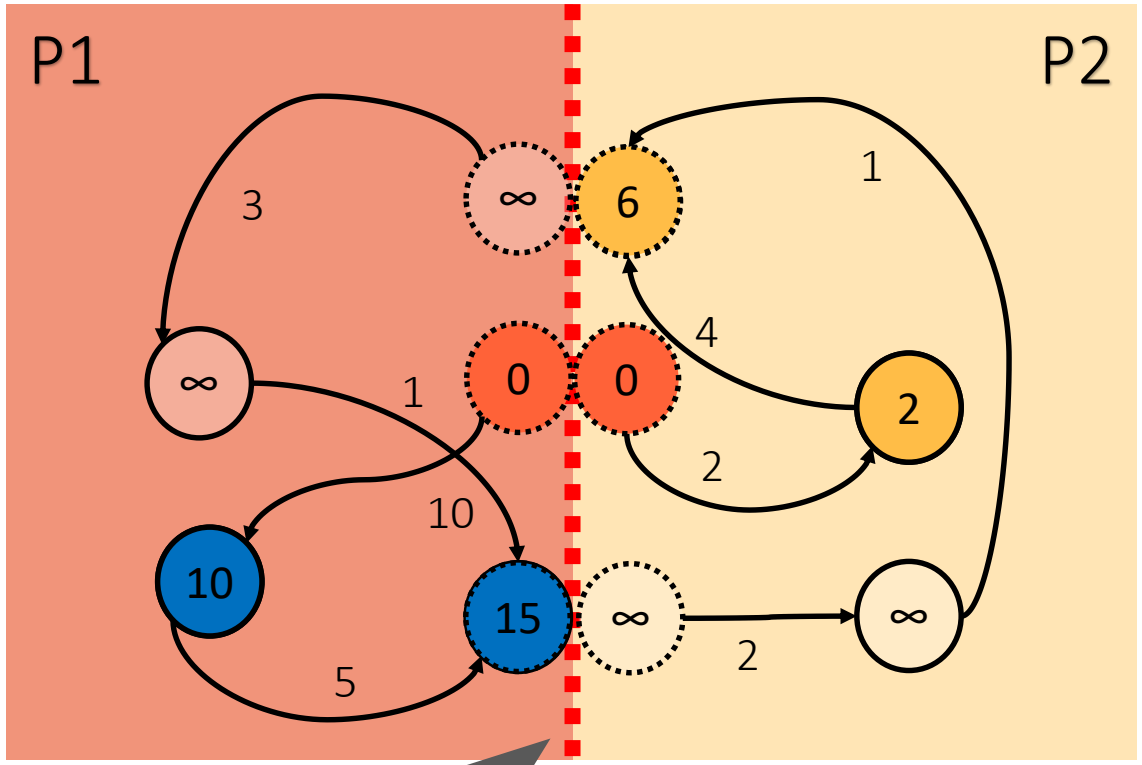
- SSSP – to compute the shortest paths between a given source and other vertices



Entity Resolution - Single Source Shortest Path

Key observation: if the weighted distance between two vertices is small, chances are they could be of the same entity

- SSSP – to compute the shortest paths between a given source and other vertices



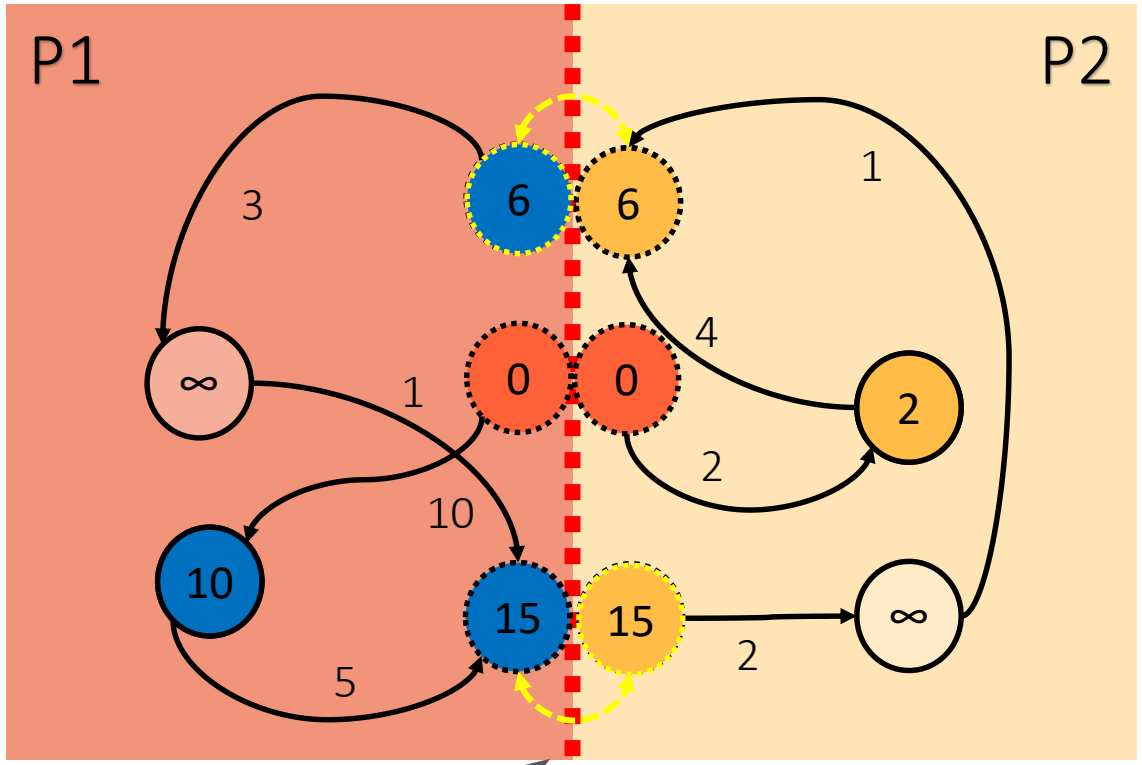
- **PEval** – Dijkstra's algorithm

PEval:
Execute Dijkstra's alg on each fragment to compute partial (local) result

Entity Resolution - Single Source Shortest Path

Key observation: if the weighted distance between two vertices is small, chances are they could be of the same entity

- SSSP – to compute the shortest paths between a given source and other vertices



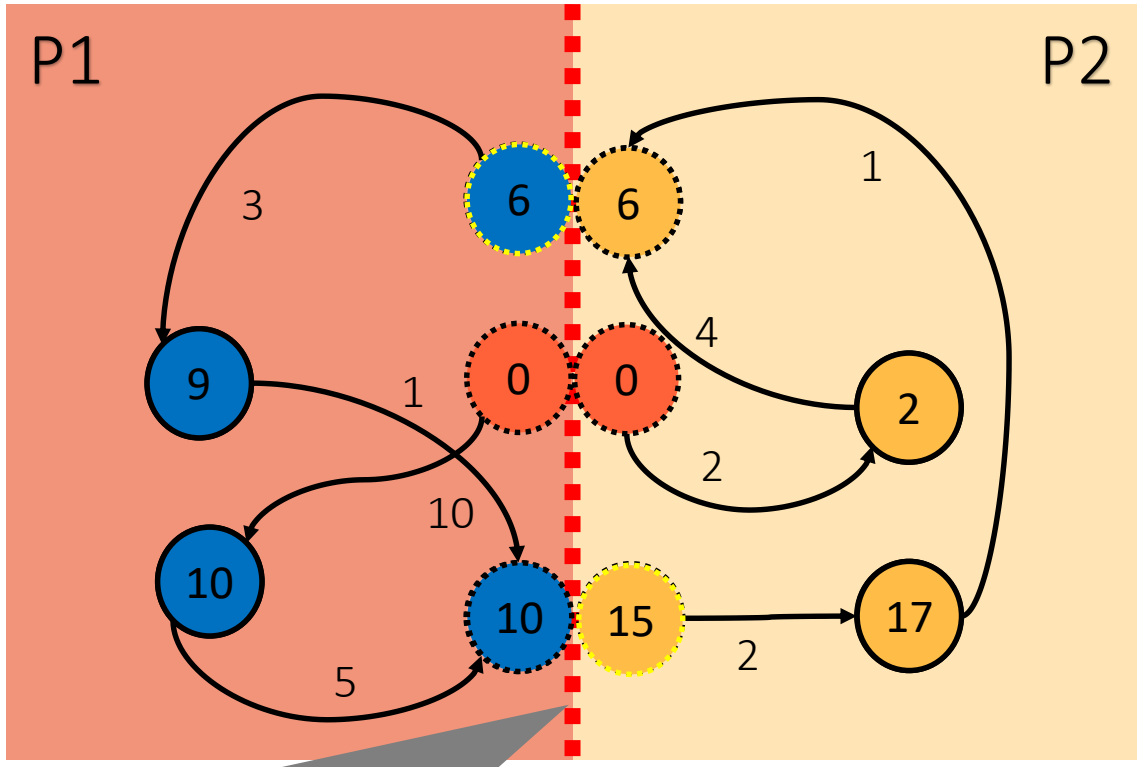
Use “min” to aggregate partial results

- **PEval** – Dijkstra’s algorithm

Entity Resolution - Single Source Shortest Path

Key observation: if the weighted distance between two vertices is small, chances are they could be of the same entity

- SSSP – to compute the shortest paths between a given source and other vertices



IncEval:
Treat updated partial result as updates, and compute the updated partial (local) results incrementally

- **PEval** – Dijkstra's algorithm
- **IncEval** – A modified Dijkstra's algorithm that handles updated partial results

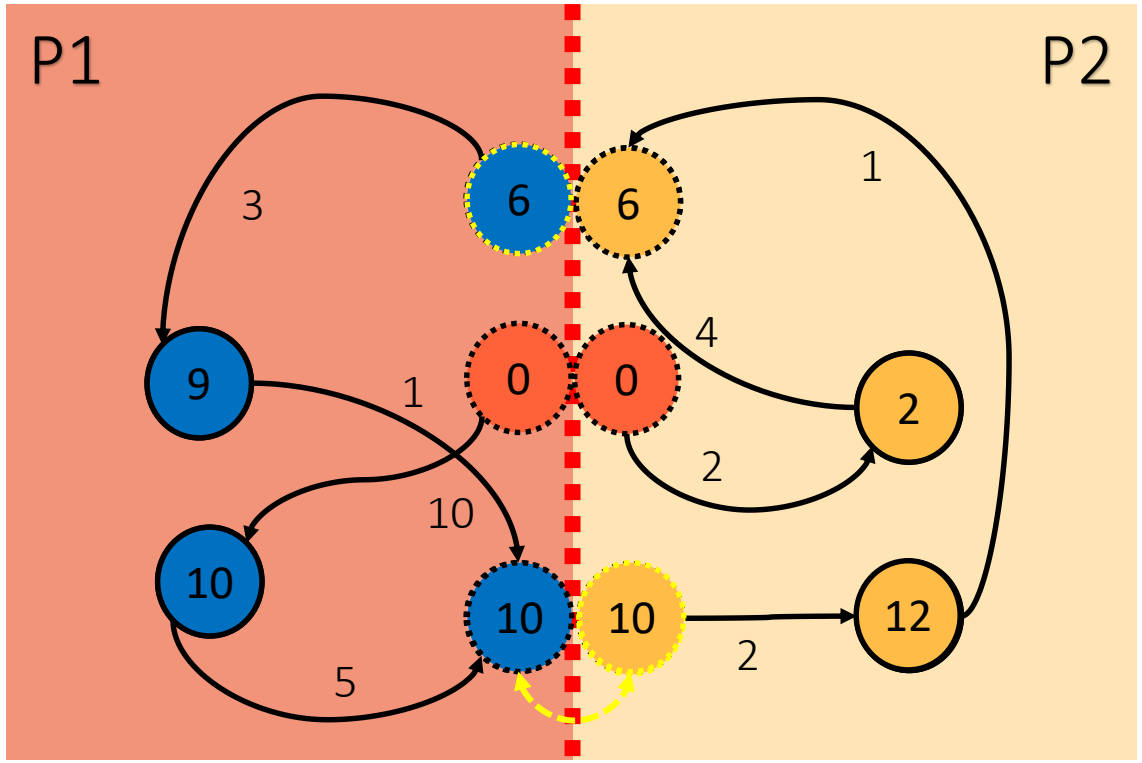
Bounded

- G. Ramalingam and T. Reps. On the computational complexity of dynamic graph problems. TCS, 158(1-2), 1996.

Entity Resolution - Single Source Shortest Path

Key observation: if the weighted distance between two vertices is small, chances are they could be of the same entity

- SSSP – to compute the shortest paths between a given source and other vertices



- **PEval** – Dijkstra's algorithm
- **IncEval** – A modified Dijkstra's algorithm that handles updated partial results
- **Assemble** – return the partial results

- **Existing efficient algorithms** for all the 3 UDFs.
- **No explicit message passing** from users
- Optimization techniques of Dijkstra's algorithms such as "*priority queue*" are **automatically inherited**, which are not possible with vertex-centric systems

The process repeats until all reaches fixpoint

Guaranteed to terminate, since "min" is monotonic

Further Optimizations

Alibaba Group 阿里巴巴集团

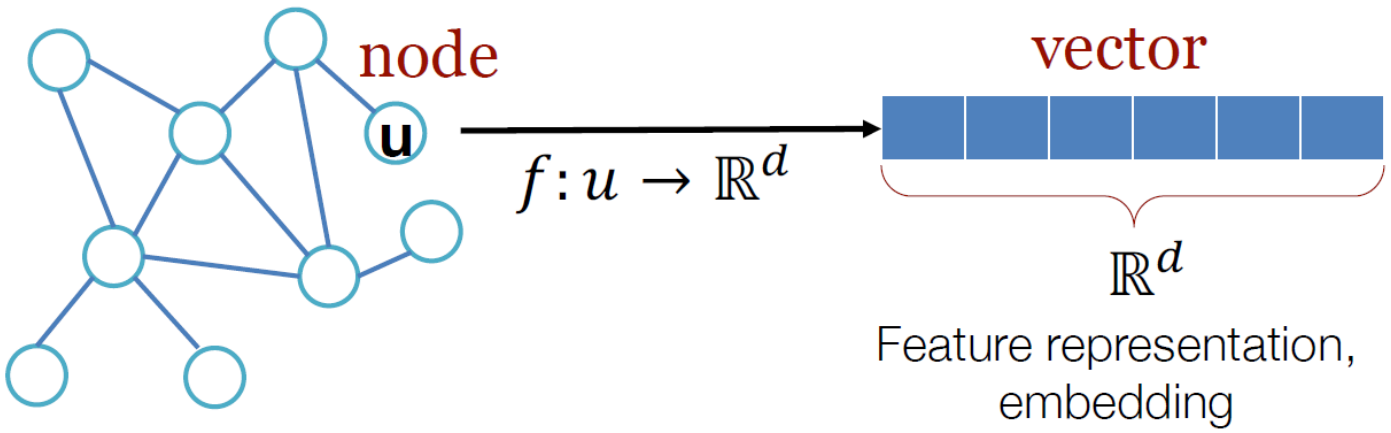
- ✓ Dynamic scaling for parallel graph computations*
  - Elasticity could be relatively easily achieved by vertex-centric systems due to the smaller scheduling unit (vertex).
  - For PIE models, dynamically repartitioning the graph is often needed.

*Preliminary study to be published in VLDB2019: Wenfei Fan, Chunming Hu, Muyang Liu, Ping Lu, Qiang Yin, Jingren Zhou: “Dynamic Scaling for Parallel Graph Computations ”

- ✓ Incrementalization of graph algorithms
  - It is not easy to develop an efficient incremental algorithms on graphs (IncEval for PIE)
  - We have been developing new techniques and tools to help users with little experience to design efficient incremental algorithms.
- ✓ Interoperability with ML systems, graph storage, graph streaming processing, …

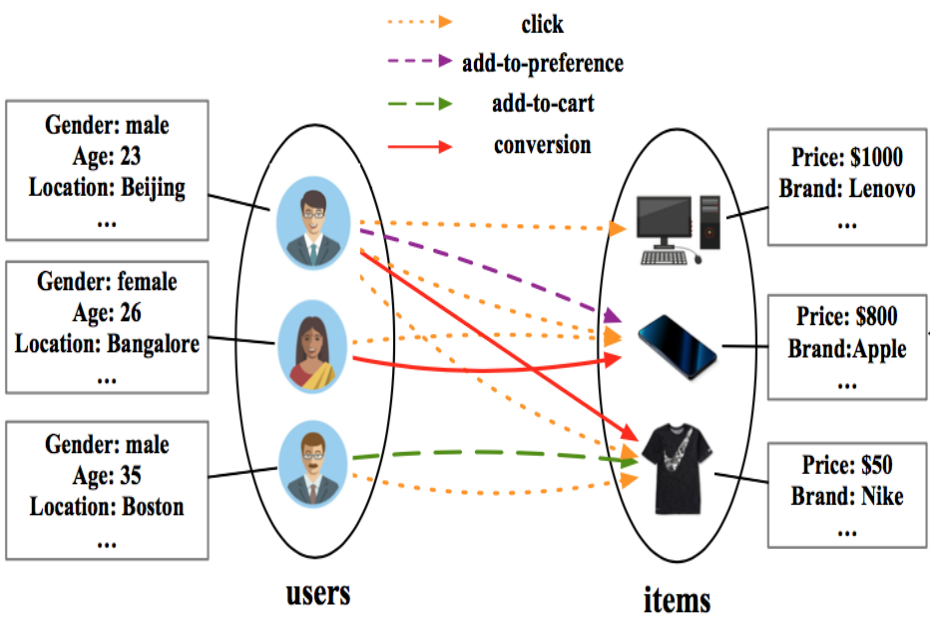
4: Graph Neural Network

Graph Embedding



- Graph embedding: low-dimensional vector representation
  - Distill high-dimensional node information into a dense vector embedding
- Together with machine learning, graph embedding proves valuable for many applications
  - Search, node classification, clustering, link predictions, etc.

A Graph Embedding Recommendation System



Sampling:
Representative and Negative Samples

Multiplex:
Node and Edge Heterogeneities

Mixture Modes:
Users with Different Latent Interest Categories

Hierarchical:
Users' community structures and items' categories

Extremely Large Scale Attributed Heterogeneous Graphs with Billions of Nodes and Trillions of Edges

Unified Graph Embeddings

Practical Challenges and Our Current Focuses

Cognitive Intelligence Platform

Alibaba Group 阿里巴巴集团

Large Scale Graph Neural Network (GNN) Oriented

阿里妈妈 Alimama.com

淘宝网 Taobao.com

天猫 TMALL.COM

Alibaba Economy Big Data Platform

飞猪

优酷 youku.com

高德地图 amap.com

菜鸟 CAINIAO

Weibo

Material, Entertainment and Health

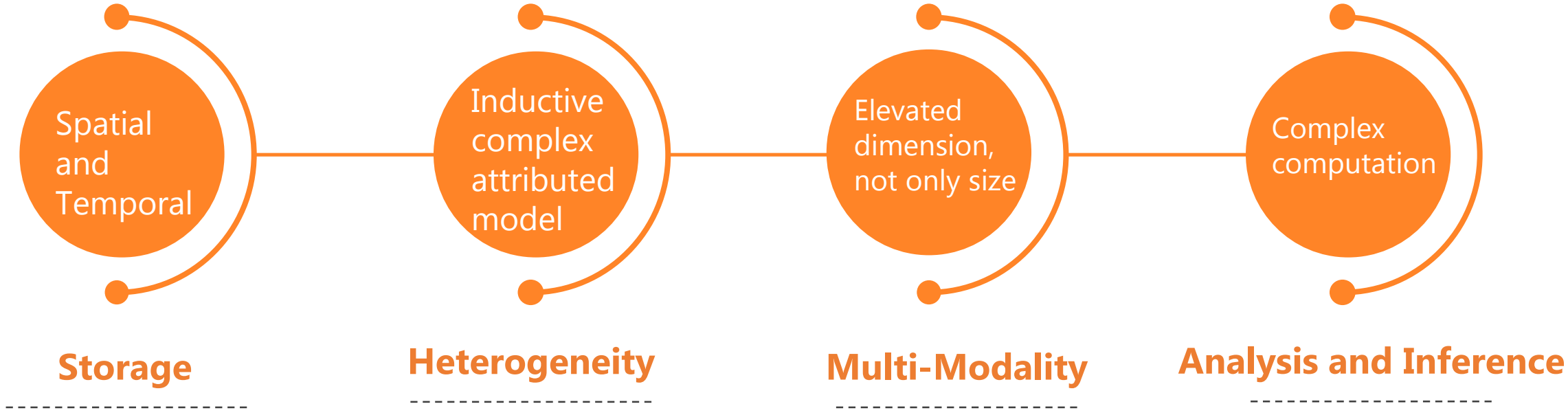
Lead Consumers Lives

Text, image and video influence the mind of consumers

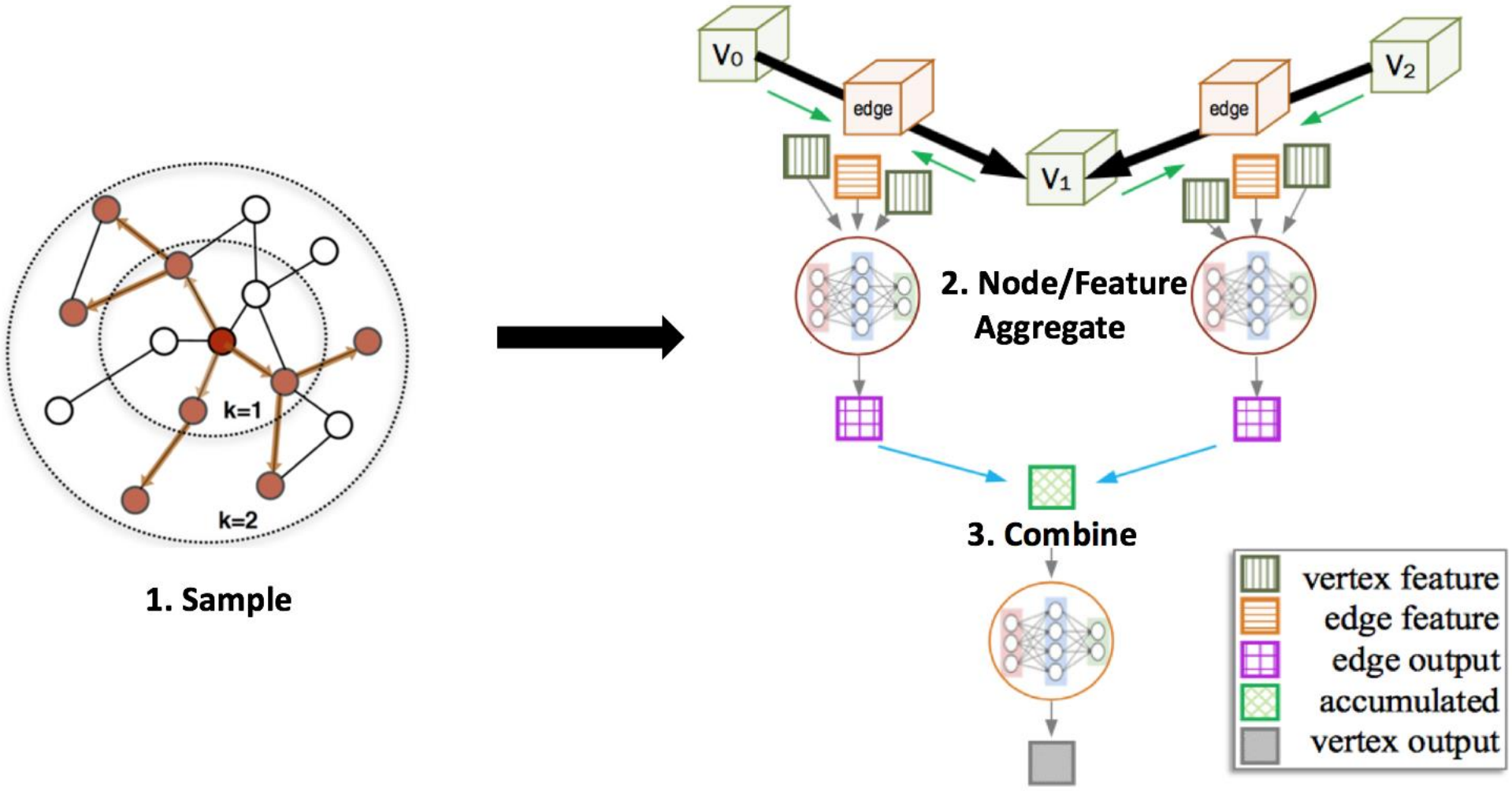
GNN Build Personalized Brain

Capable of inductive reasoning, explainable, better know consumers than themselves

Challenges for GNN Platform



GNN Framework Overview

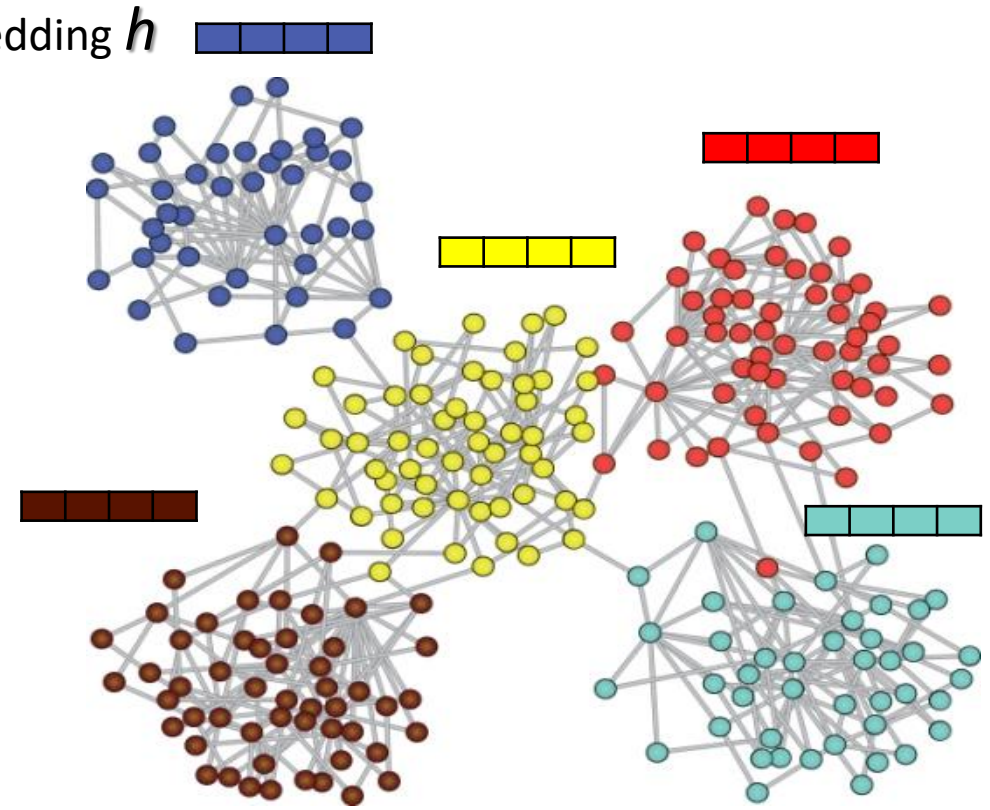


System Optimizations

Algorithm 1: GNN Framework

Input: network $\mathcal{G}$, embedding dimension $d \in \mathbb{N}$, a vertex feature $\mathbf{x}_v$ for each vertex $v \in \mathcal{V}$ and the maximum hops of neighbors $k_{max} \in \mathbb{N}$.
Output: embedding result $\mathbf{h}_v$ of each vertex $v \in \mathcal{V}$

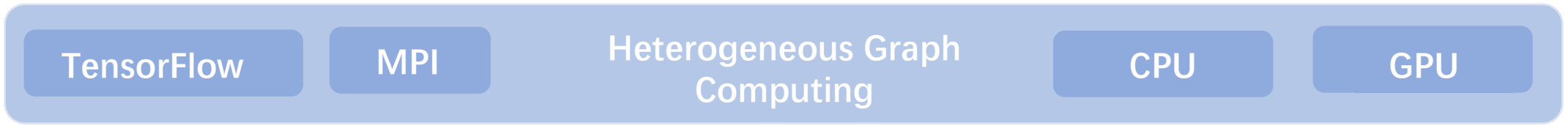
$\mathbf{h}_v^{(0)} \leftarrow \mathbf{x}_v$
for $k \leftarrow 1$ *to* k_{max} **do**
  for *each vertex* $v \in \mathcal{V}$ **do**
    $S_v \leftarrow \text{SAMPLE}(Nb(v))$
    $\mathbf{h}'_v \leftarrow \text{AGGREGATE}(\mathbf{h}_u^{(k-1)}, \forall u \in S)$
    $\mathbf{h}_v^{(k)} \leftarrow \text{COMBINE}(\mathbf{h}_v^{(k-1)}, \mathbf{h}'_v)$
  normalize all embedding vectors $\mathbf{h}_v^{(k)}$ for all $v \in \mathcal{V}$
$\mathbf{h}_v \leftarrow \mathbf{h}_v^{(k_{max})}$ for all $v \in \mathcal{V}$ return $\mathbf{h}_v$ as the embedding result for all $v \in \mathcal{V}$



- Graph partitioning and clustering
- Co-locate embedding variables with graph partitions
- Reuse embedding if possible

GNN Platform Architecture

1. Rich algorithm warehouse with improved accuracy measures (5%-90%) for practical challenges;
2. Performs an order of magnitude faster in terms of graph building, e.g., 492.90 million vertices, 6.82 billion edges and rich attributes, 5 minutes vs hours by other state-of-the-arts;
3. 40%-50% faster with the novel caching strategy and demonstrates around 12 times speed up with the improved runtime.



Algorithm Warehouse

Alibaba Group 阿里巴巴集团

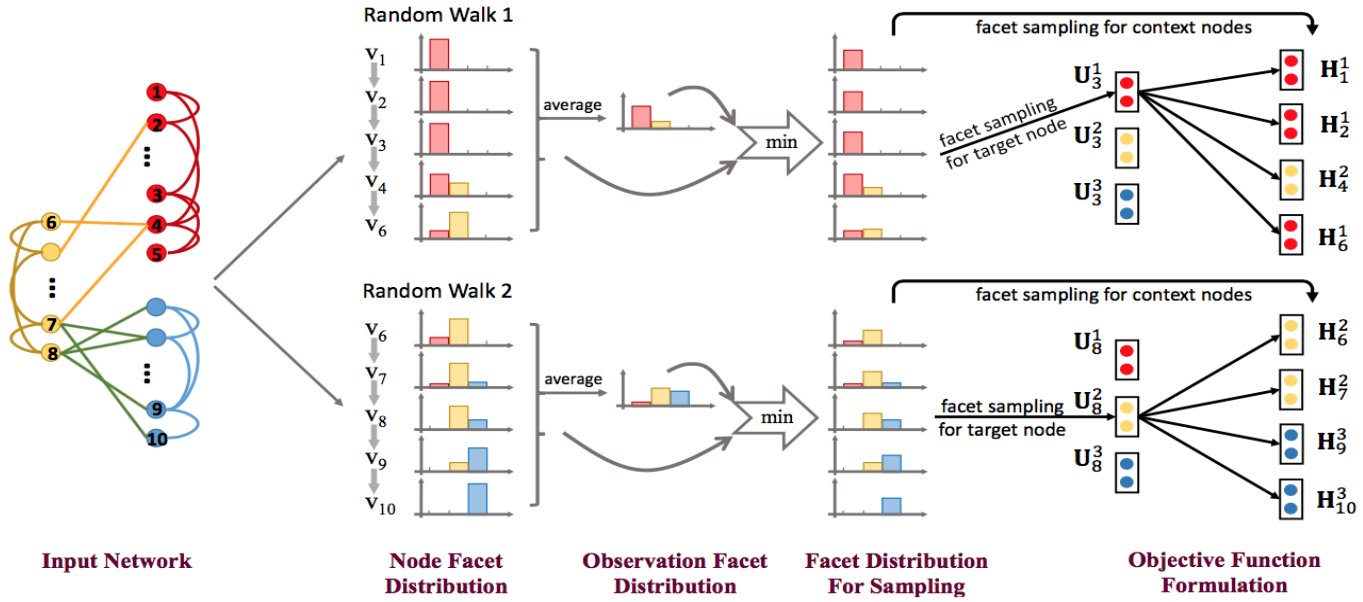
Table 1: The property of different methods.

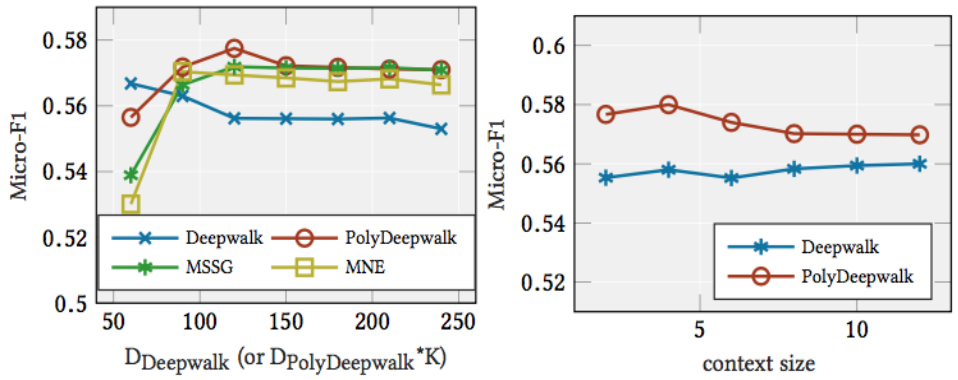
Category	Method	Heterogeneous		Attributed	Dynamic	Large-Scale
		Node	Edge			
Classic Graph Embedding	DeepWalk	✗	✗	✗	✗	✗
	Node2Vec	✗	✗	✗	✗	✗
	LINE	✗	✗	✗	✗	✗
	NetMF	✗	✗	✗	✗	✗
	TADW	✗	✗	✓	✗	✗
	LANE	✗	✗	✓	✗	✗
	ASNE	✗	✗	✓	✗	✗
	DANE	✗	✗	✓	✗	✗
	ANRL	✗	✗	✓	✗	✗
	PTE	✗	✓	✗	✗	✗
	Methpath2Vec	✗	✓	✗	✗	✗
	HERec	✗	✓	✗	✗	✗
	HNE	✗	✗	✗	✗	✗
	PMNE	✗	✓	✓	✗	✗
	MVE	✗	✓	✓	✗	✗
	MNE	✗	✓	✓	✗	✗
	Mvn2Vec	✗	✓	✓	✗	✗
GNN	Structural2Vec	✗	✗	✓	✗	✗
	GCN	✗	✗	✓	✗	✗
	FastGCN	✗	✗	✓	✗	✗
	AS-GCN	✗	✗	✓	✗	✗
	GraphSAGE	✗	✗	✓	✗	✗
	HEP	✓	✓	✓	✗	✗
	AHEP	✓	✓	✓	✗	✓
	GATNE	✓	✓	✓	✗	✓
	Mixture GNN	✓	✓	✓	✗	✗
	Hierarchical GNN	✓	✓	✓	✗	✗
	Bayesian GNN	✗	✓	✓	✗	✗
	Evolving GNN	✗	✓	✓	✓	✗

- Training data selection:
  - ✓ Representative and negative sampling
  - ✓ Borrow the ideas of importance sampling
  - ✓ Extend from node-wise to batch-wise
- Multiplex:
  - ✓ Rich information between different edges types but the various influential factors between different edge types is hard to capture
  - ✓ The relationship of different edge types is considered through attention based framework
  - ✓ More than 20% improvement over state-of-the-arts (e.g., DeepWalk)

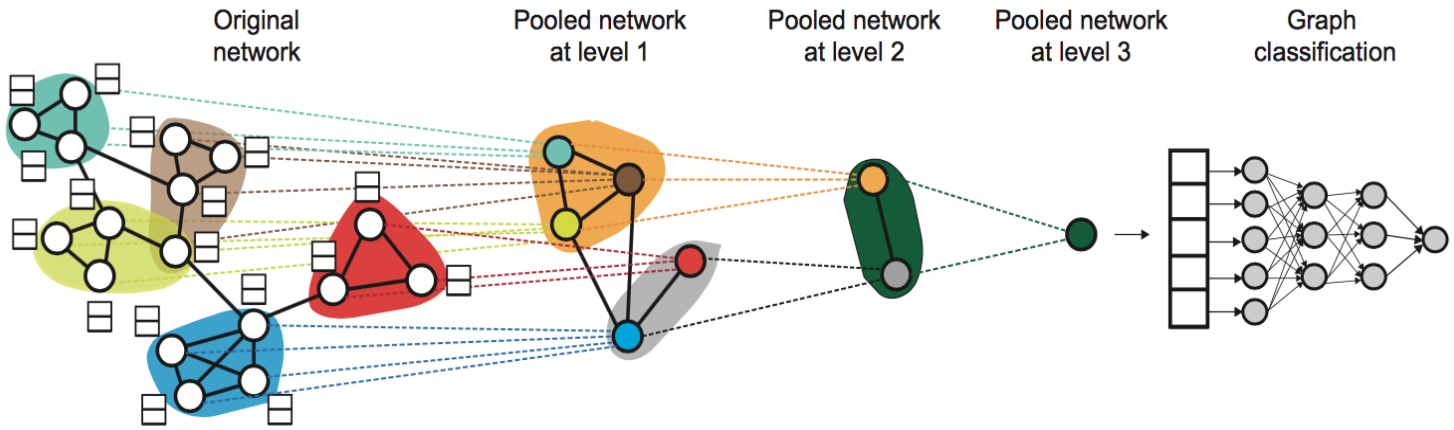
Mixture GNN

- Most web applications contain various scenarios in which recommendation happens
- A large portion of scenarios in a system are actually long-tailed, without enough user feedback
- Polysemous skip-gram model for both homogeneous and heterogeneous models





Hierarchical GNN



Method	Accuracy	F1	AUC (ROC)	AUC (PR)
GraphSAGE	88.3%	47.4%	83.9%	46.1%
Bi-GNN	88.0%	44.6%	81.5%	41.3%
DiffPool-S	87.6%	38.2%	77.1%	33.6%
O-GNN	87.4%	25.8%	64.2%	19.2%
Bi-HGNN	**90.5%**	**59.1%**	**90.2%**	**63.2%**

- Current GNN methods are inherently flat and do not learn hierarchical representations
- Hierarchical GNN uses a hierarchical representations of graphs
- Yields an average improvement of 5–10% accuracy on graph classification benchmarks, compared to all existing pooling approaches

Agenda

Alibaba Group 阿里巴巴集团

- Overview
- Challenges for graph computing
- Graph applications
  - Graph traversal
  - Graph pattern matching
  - Complex graph algorithms
  - Graph learning
- Conclusion

Graph Computing is the Next Big Thing!

Alibaba Group 阿里巴巴集团

- Graph data and its complex analysis are crucial for many applications
  - Support a wide range of business applications
  - Scenarios are highly versatile
- Managing and analyzing graph data at scale
  - Requires distributed graph store with high frequent updates
  - Efficient parallel graph processing is a must
- GNN combines machine learning with graph analytics
  - Gains increasing popularity in various domains, including social network, knowledge graph, recommender system, and even life science.
  - Requires sophisticated system optimization to improve performance and scalability

Data Analytics and Intelligence Lab

Jingren Zhou
Vice President & Lab Head

Boling Ding
Senior Staff Engineer

Hongxia Yang
Senior Staff Engineer

Zhengping Qian
Senior Staff Engineer

Wenyuan Yu
Senior Staff Engineer

Kai Zeng
Staff Engineer

Collaborators: Prof. Wenfei Fan, Prof. Xuemin Lin and their groups

Alibaba Group
阿里巴巴集团

Thank You!