E6893 Big Data Analytics Lecture 4:

Linked Big Data Analytics

Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science

September 27th, 2024

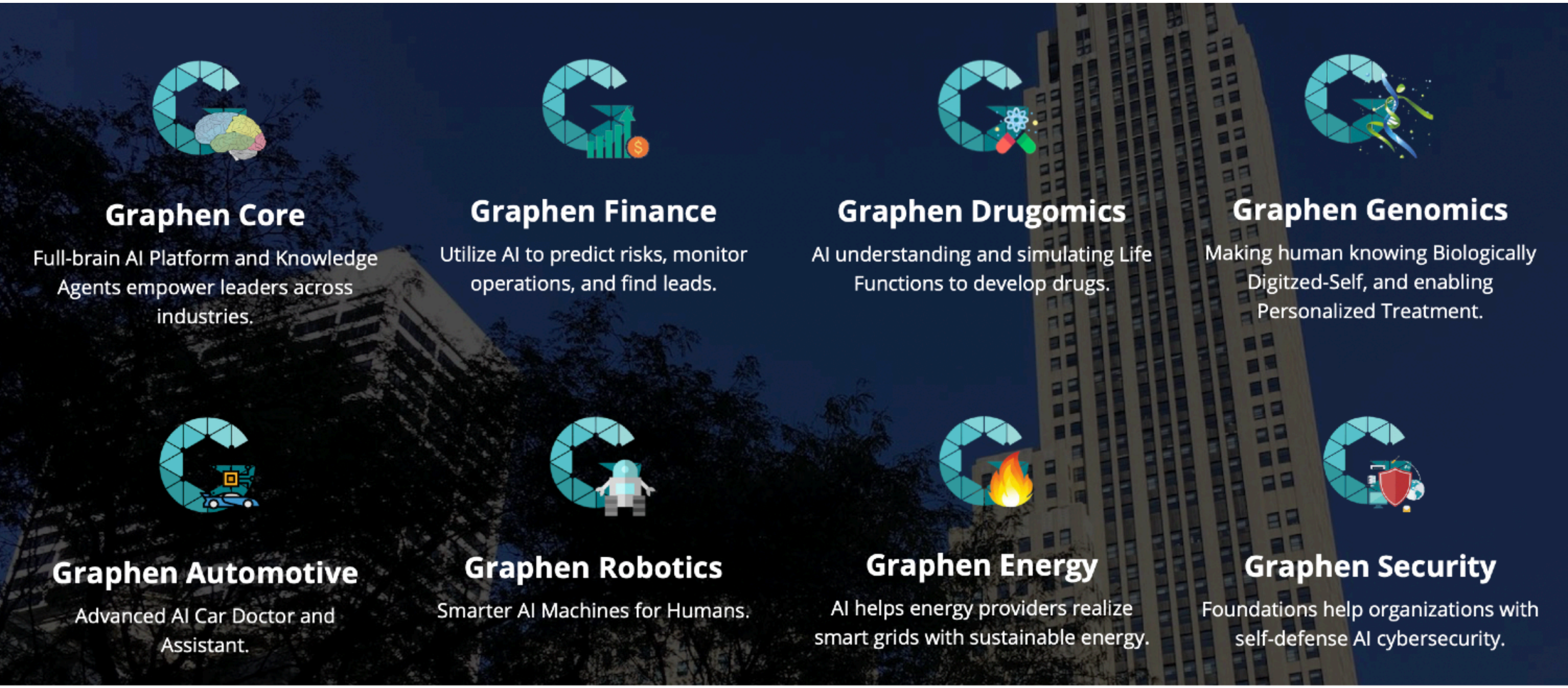
Start Your Final Project Planning

Start brainstorming your final project

- Start finding your teammates.

- Proposal (11/8/24) — preparing about 5-7 pages of slides (each item 1/5 of the proposal score):
  - Goal — novel? challenging?
  - Data — 3Vs? New dataset? Existing dataset?
  - Methods — planning of methodologies and algorithms? Feasible?
  - System — an overview of system. What will be implemented?
  - Schedule — what to achieve by what time, and by whom?

Example: Big Data Analytics & AI Application Areas



Area 1 'Cognitive Machine' Tasks List:

A1: Deep Video Understanding (Visual + Knowledge) — Face Recognition, Feeling Recognition, and Interaction

A2: Deep Video Understanding (Language + Knowledge) — Speech Recognition, Gesture Recognition, and Feeling Recognition

A3: Deep Video Understanding — Event and Story Understanding

A4: Humanized Conversation — Personality-Based Conversations

A5: Autonomous Robot Learning of Physical Environment

A6: Autonomous Task Learning via Mimicking

A7: Digital Human - Creation and Facial Expression

A8: Digital Human - Action

A9: Digital Human - Text-to-Audio, Lip Sync, and Audio-to-Text

A10: Human and Digital Human Interactions

A11: Feeling and Art Recognition

A12: Creative Writing & Story Telling

A13: Knowledge Learning & Construction

A14: Dreams — Simulating Brain functions while sleeping

A15: Self-Consciousness, Ethics, and Morality

Digital Human Examples

Graphen Ava

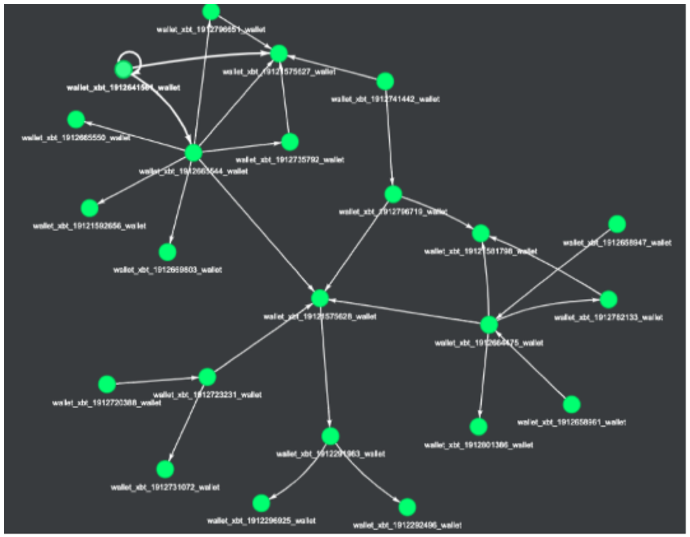
Meta

https://www.graphen.ai/products/Ava.html

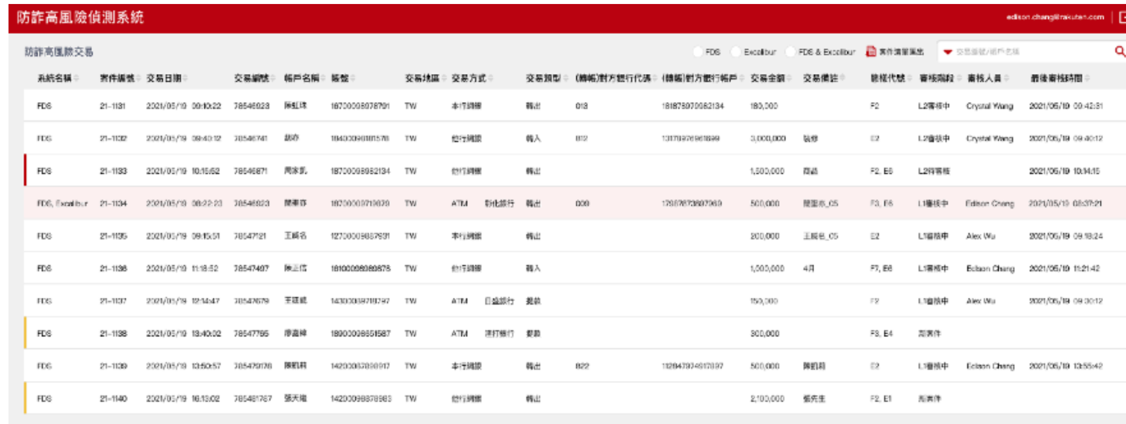
Area 2 ‘Finance Advisor’ Tasks List:

B1: Market Intelligence — Constructing Financial Knowledge Graphs
B2: Market Intelligence — Company Environmental, Societal, and Governance Performance
B3: Market Intelligence — Event Linkage and Impact Prediction
B4: Market Intelligence — Alpha Generation from Alternative Sources
B5: Advance KYC — Customer Profiling based on Personality, Needs, and Value
B6: Advanced KYC — Customer Behavior Prediction
B7: Investment Strategy — AI Trader (Foreign Exchange)
B8: Investment Strategy — AI Trader (Stock Markets)
B9: Investment Strategy — Automatic Dynamic Asset Allocation
B10: Customer Interaction — Customer Communication Strategies
B11: Customer Interaction — Insurance Product Sales & Marketing Strategy
B12: Automatic Story Telling for Marketing
B13: Automatic Market Competition Analysis
B14: Automatic Consumer Sales Leads Finding
B15: Human Capital Growth Recommendations

Real-Time Fraud Analysis Examples



Crypto Currencies



Online Banks

Credit Cards

Area 3 'Healthy Life' Tasks List:

C1: Precision Health — Gene and Protein Analysis of Network, Pathway, and Biomarkers

C2: Large-Scale System for Human Genome Analysis

C3: Secure Patient Data

C4: Medical Image Analysis

C5: Drugable Targets for Precision Medicine

C6: Virus Mutations and Function Prediction

C7: Microbe and Disease Knowledge Graph

C8: Disease Symptoms Knowledge Graphs

C9: Virtual Doctor

C10: Knowledge Graphs for Gene Interaction and Disease Similarity

C11: Biomedical Knowledge Construction and Extraction

C12: Generating Gene Therapy

C13: Molecular Drug Synthesis

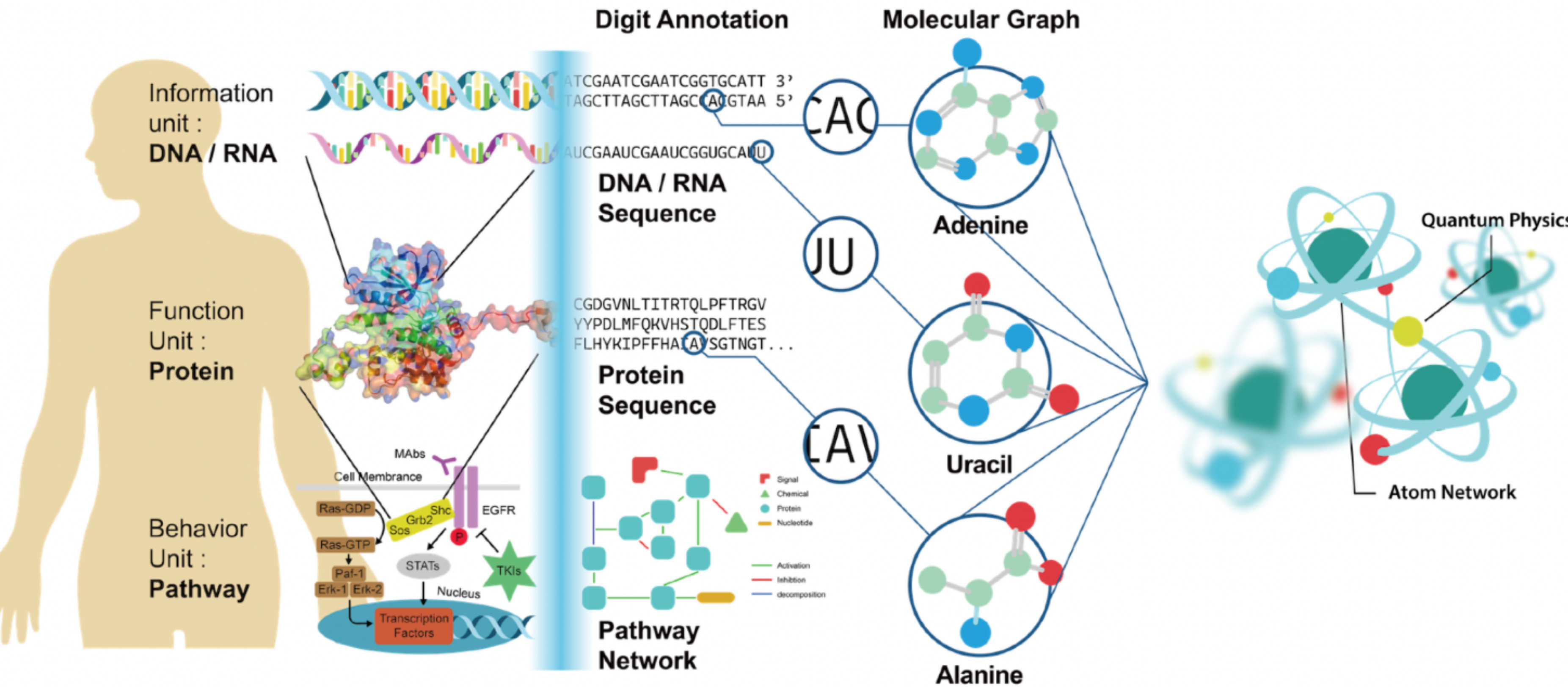
C14: Protein Interaction Predictor

C15: Aging Impacts

Digital Biology Examples

https://www.graphen.ai/products/atom.html

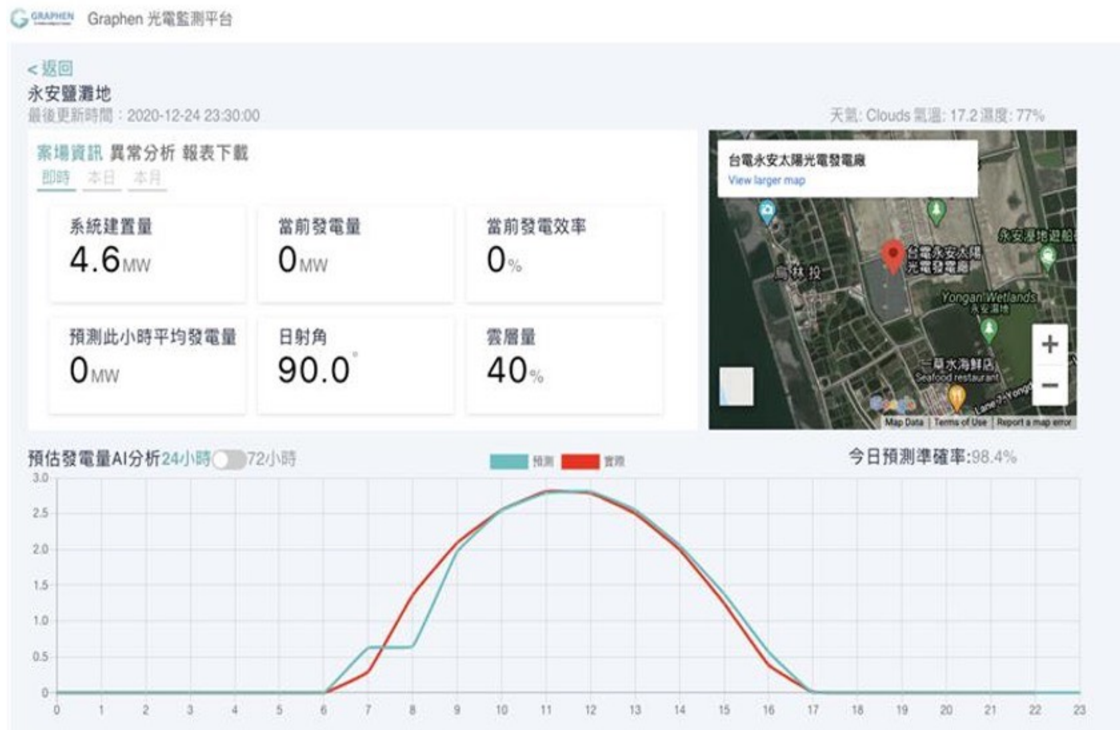
Biological Materials **Digitalize Bioinformation** **Atomotive Forces**



Area 4 ‘Green Earth’ Tasks List:

D1: Distributed Solar Power Load Forecasting and Predictive Maintenance
D2: Distributed Wind Power Load Forecasting and Predictive Maintenance
D3: Power Flow Optimization
D4: Smart Grid Pricing Strategy
D5: Cybersecurity of Smart Grid
D6: Stimulating Crop Growth
D7: Electronic Car Sensing and Predictive Maintenance
D8: Autonomous Driving
D9: Smart City of Connected Cars
D10: Social Policy Monitoring
D11: International Relationships and Policy Monitoring
D12: Mobile Cognition
D13: AI Chip Design
D14: Visual Exploration in Immersive Environment
D15: Computer Vision Enhanced Immersive Environment

Green Earth Examples

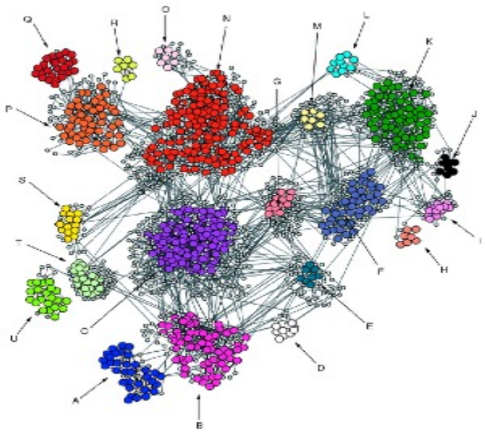


- **Renewable Energy Prediction**
- **Power System Anomaly Detection**
- **Predictive Maintenance**
- **Dispatching System**

https://www.youtube.com/watch?v=9PTlqCCMX-0 9/20/2022

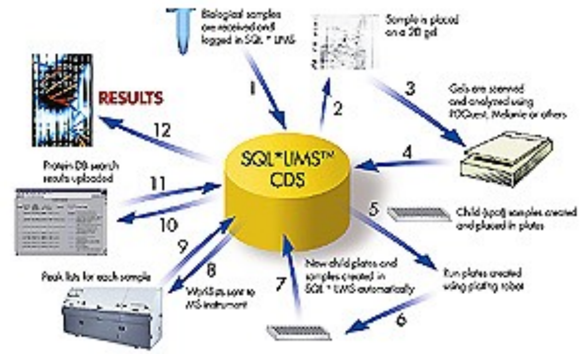
Link Big Data / Graph Analytics

Networks Everywhere

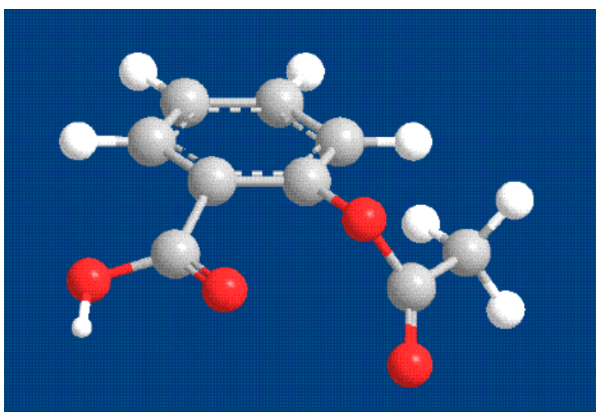


Magwene *et al.* *Genome Biology* 2004 **5**:R100

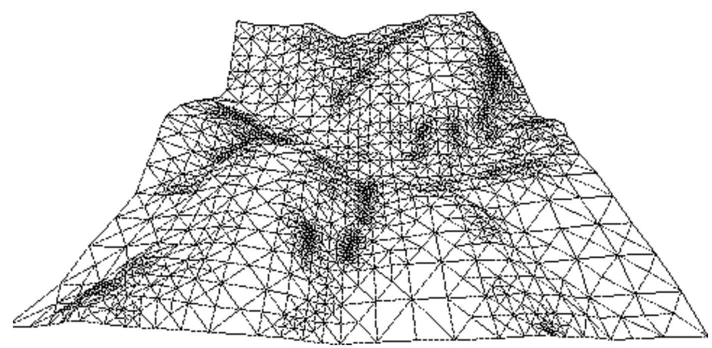
Gene Coexpression Network



Scientific Workflow



Chemical Compound



Mesh

First Reported Social Network Analysis

The New York Times.

Copyright, 1933, by The New York Times Company.

Entered as Second-Class Matter, Postoffice, New York, N. Y. NEW YORK, THURSDAY, APRIL 3, 1933. TWO CENTS

EMOTIONS MAPPED BY NEW GEOGRAPHY

Charts Seek to Portray the Psychological Currents of Human Relationships.

FIRST STUDIES EXHIBITED

Colored Lines Show Likes and Dislikes of Individuals and of Groups.

MANY MISFITS REVEALED

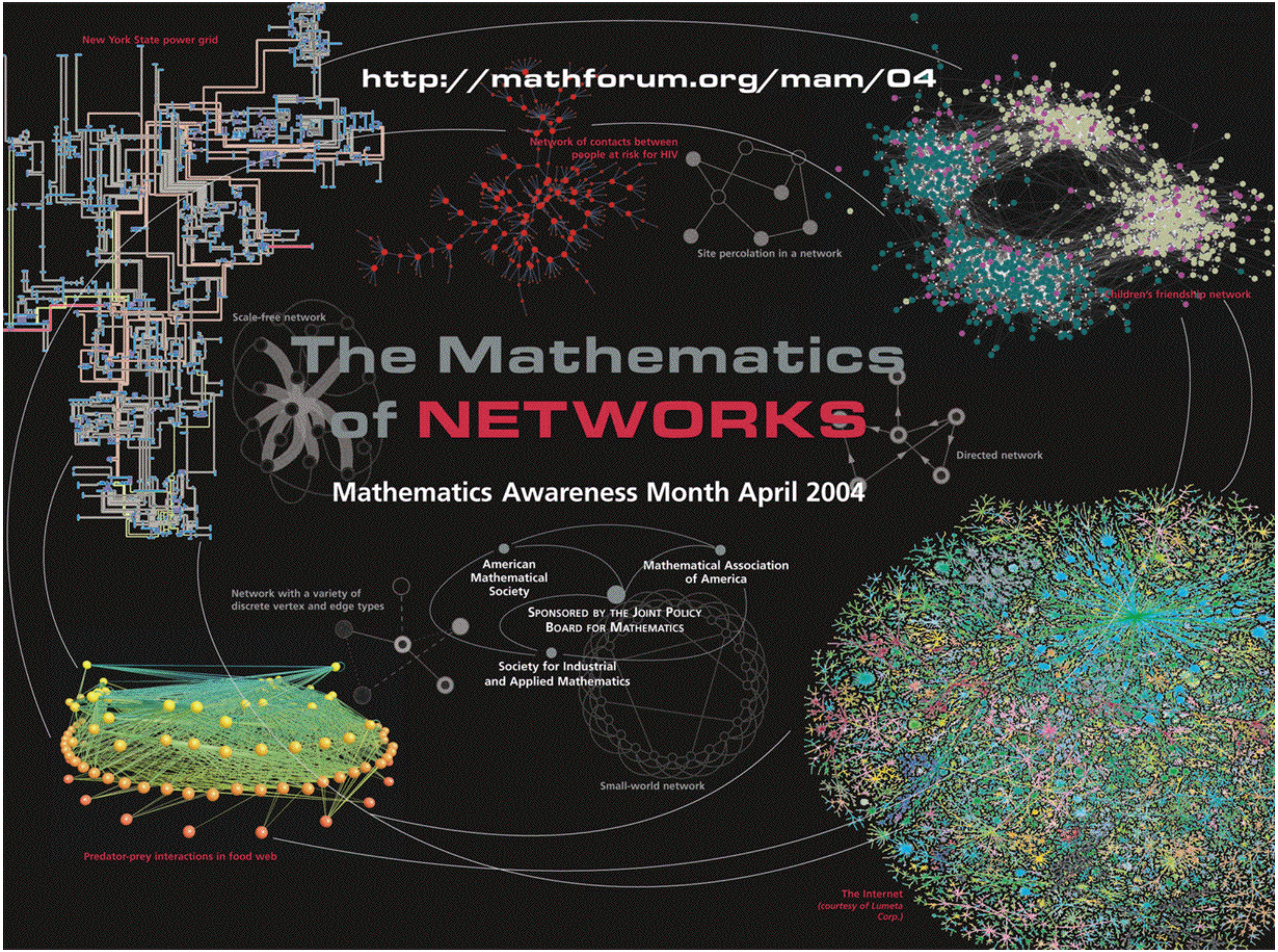
Dr. J. L. Moreno Calculates There Are 10 to 15 Million Isolated Individuals in Nation.

A new science, named psychological geography, which aims to chart the emotional currents, cross-currents and under-currents of human relationships in a community, was introduced here yesterday at the scientific exhibit of the Medical Society of the State of New York, which opens its 127th annual meeting here today at the Waldorf-Astoria.

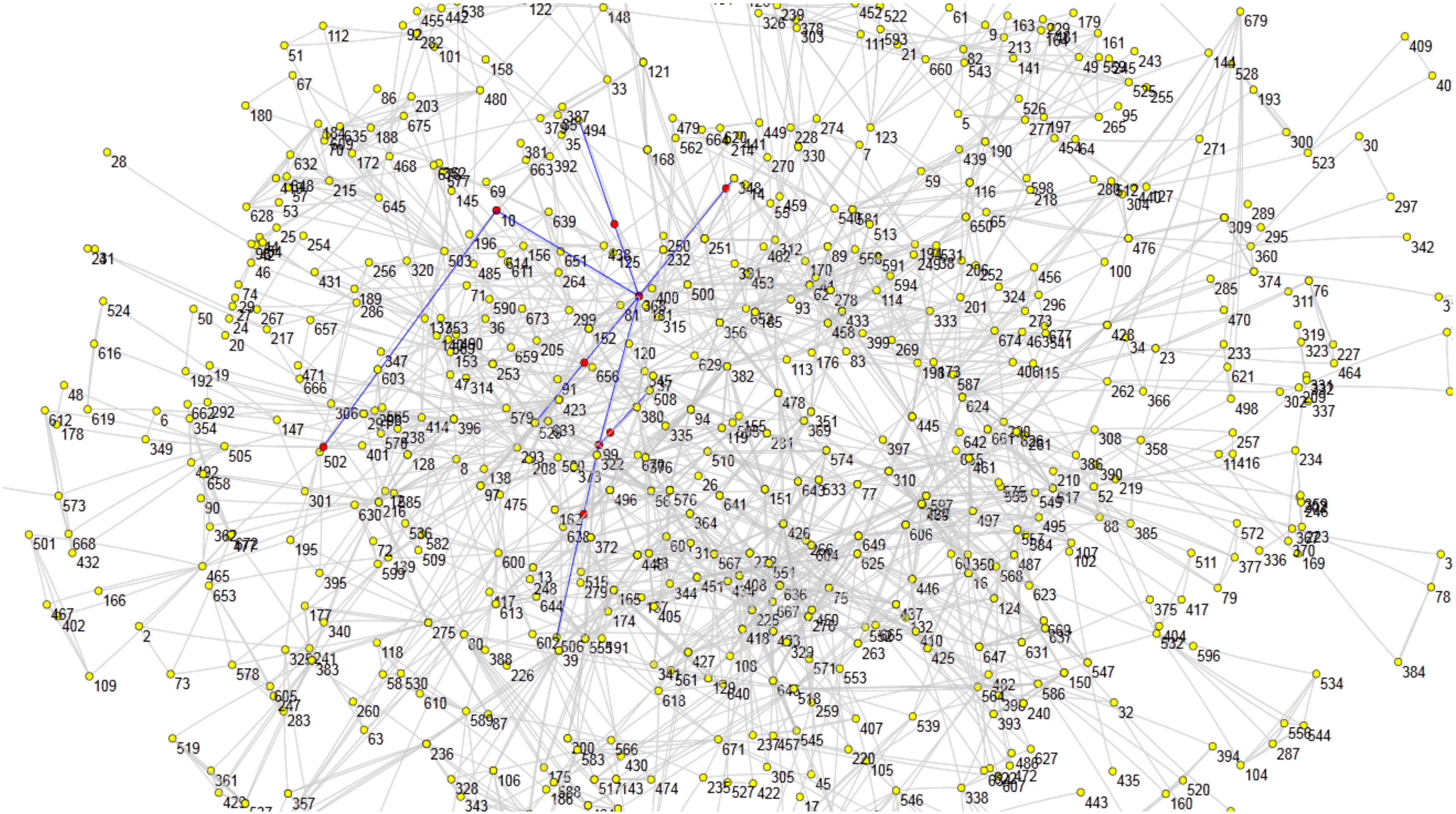
The first series of maps of the new human geography were shown by Dr. Jacob L. Moreno of New York, consulting psychiatrist of the National Committee of Prisons and Prison Labor and director of research, New York State Training School for Girls, Hudson, N. Y. The maps represent studies of the forces of attraction and repulsion of individuals within a group toward one another and toward the group, as well as the attitude of the group as a whole toward its individual members, and of one group toward another group.

Emotions are represented on these psychological maps by various colored lines. Red stands for liking, black for disliking. If individual A likes B a red line with an arrow points from A to B. If B reciprocates a similar red line points from him. If he dislikes A this is indicated by a black line with an arrow pointing toward A. If B is merely indifferent the feeling is shown by a blue line.

Group of 500 Girls Studied.


http://mathforum.org/mam/04
New York State power grid
Network of contacts between people at risk for HIV
Site percolation in a network
Children's friendship network
Scale-free network
The Mathematics of NETWORKS
Directed network
Mathematics Awareness Month April 2004
American Mathematical Society
Mathematical Association of America
Network with a variety of discrete vertex and edge types
Sponsored by the Joint Policy Board for Mathematics
Society for Industrial and Applied Mathematics
Small-world network
Predator-prey interactions in food web
The Internet (courtesy of Lumeta Corp.)

Columbia University

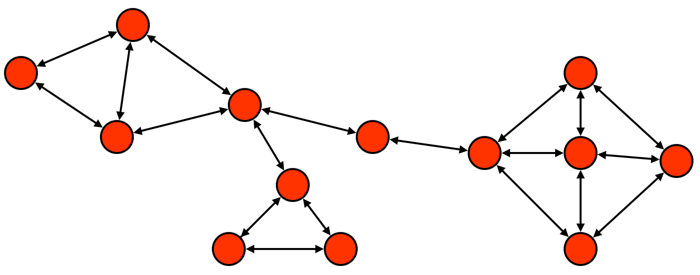


Network Science <=> Graph Technology

Graph Definitions and Concepts

- A graph:

$$G = (V, E)$$



- V = Vertices or Nodes
- E = Edges or Links

- The number of vertices: "Order"

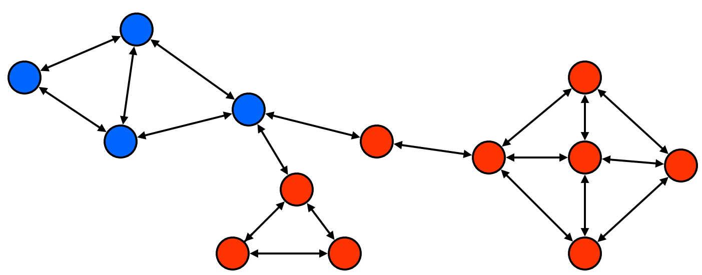
$$N_v = |V|$$

$$N_e = |E|$$

Subgraph

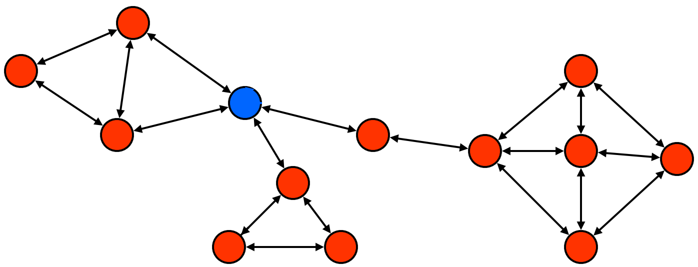
- A graph H is a subgraph of another graph G, if:

$$V_H \subseteq V_G \quad \text{and} \quad E_H \subseteq E_G$$

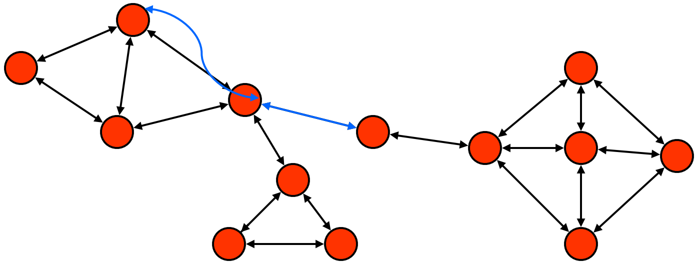


Multi-Graph vs. Simple Graph

- Loops:



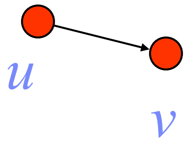
- Multi-Edges:



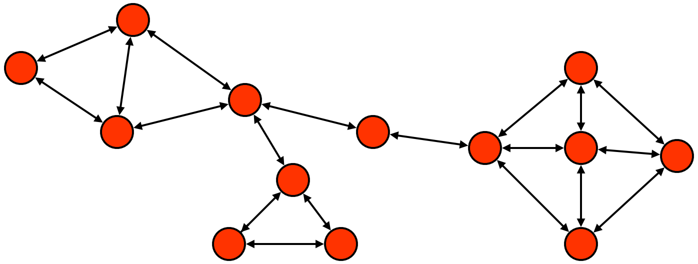
Directed Graph vs. Undirected Graph

- Directed Edges = Arcs:

$$\{u, v\}$$

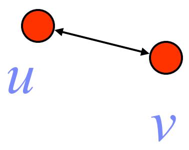


- Mutual arcs:

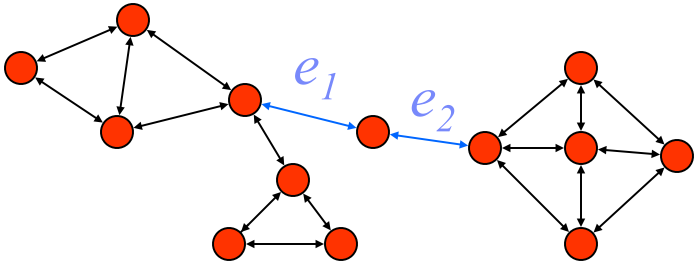


Adjacency

- u and v are adjacent if joined by an edge in E:

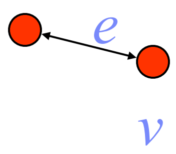


- Two edges are adjacent if joined by a common endpoint in V:

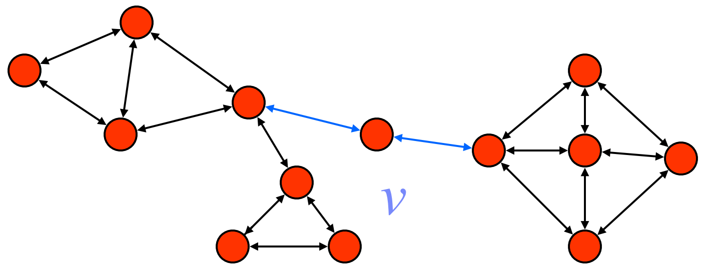


Incident and Degree

- A vertex $v \in V$ is incident on an edge $e \in E$ if v is an endpoint of e.



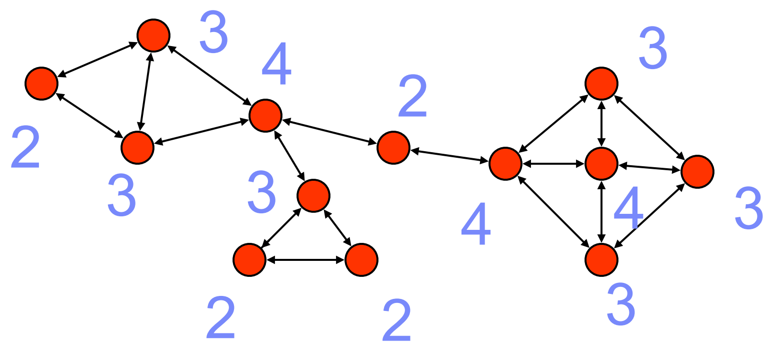
- The degree of a vertex v, say d_v, is defined as the number of edges incident on v.



$d_v=2$

Degree Sequence

- The degree sequence of a graph G is the sequence formed by arranging the vertex degrees d_v in non-decreasing order.

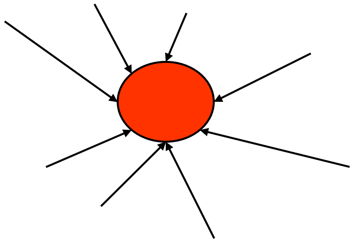


$$\{2,2,2,2,3,3,3,3,3,3,4,4,4\}$$

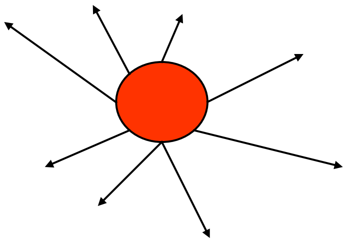
- The sum of the elements degree sequence equals to twice the number of edges in the graph (i.e. twice the size of the graph).

In-degrees and out-degrees

- For Directed graphs:



In-degree = 8



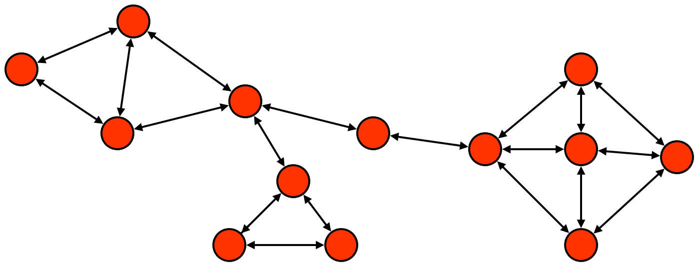
Out-degree = 8

Walk

- A walk on a graph G, from v_0 to v_l, is an alternating sequence:

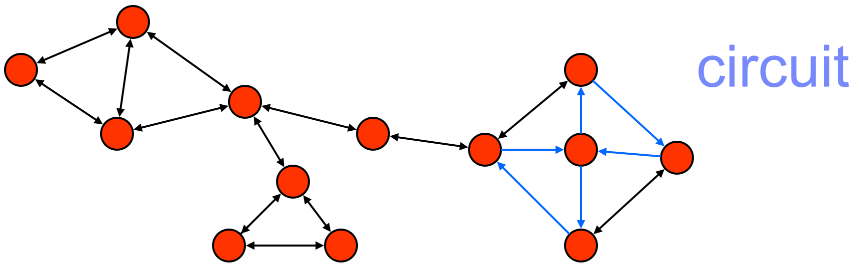
$$\{v_0, e_1, v_1, e_2, \ldots, v_{l-1}, e_l, v_l\}$$

- The length of this walk is l.
- A walk may be:
  - Trail --- no repeated edges
  - Path --- trails without repeated vertices.

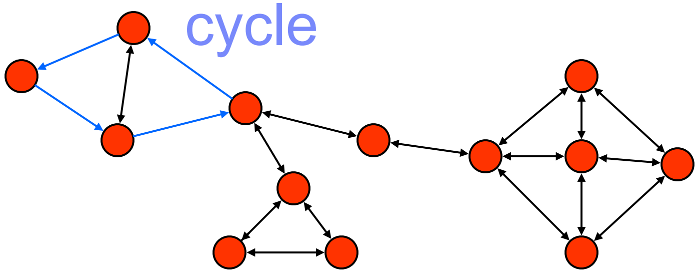


Circuit, Cycle, and Acyclic

- Circuit: A trail for which the beginning and ending vertices are the same.



  - Cycle: a walk of length at least three, the beginning node = ending node, all other nodes are distinct



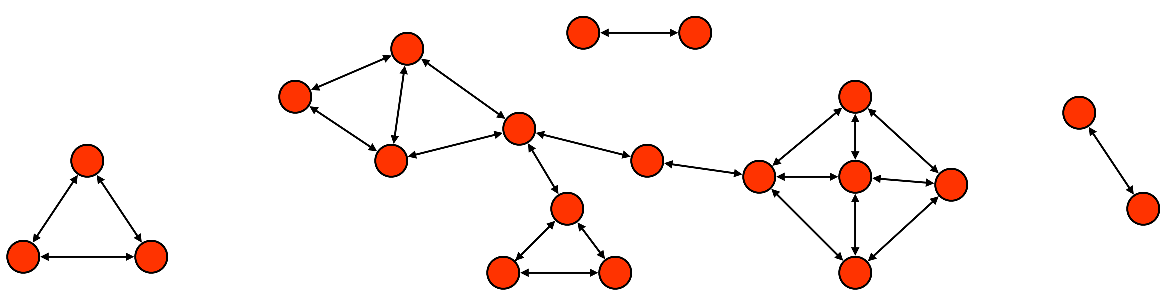
- Acycle: graph contains no cycle

Reachable, Connected, Component

- Reachable: A vertex v in a graph G is said to be reachable from another vertex u if there exists a walk from u to v.

- Connected: A graph is said to be connected if every vertex is reachable from every other.

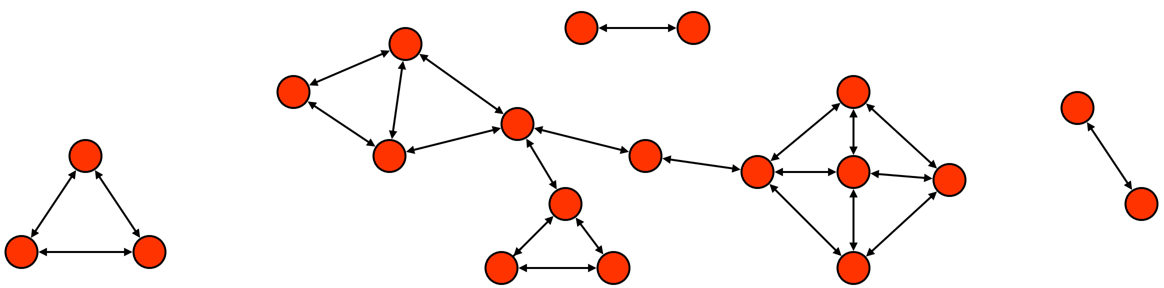
- Component: A component of a graph is a maximally connected subgraph.



Connection in a digraph

- Weakly connected: If its underlying graph is connected after stripping away the direction.

- Strongly connected: every vertex is reachable from every other vertex by a directed walk.

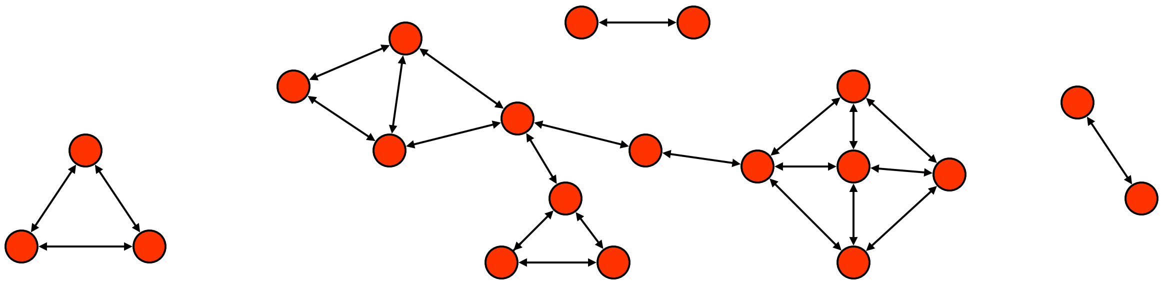


Distance

- Distance of two vertices: The length of the shortest path between the vertices.

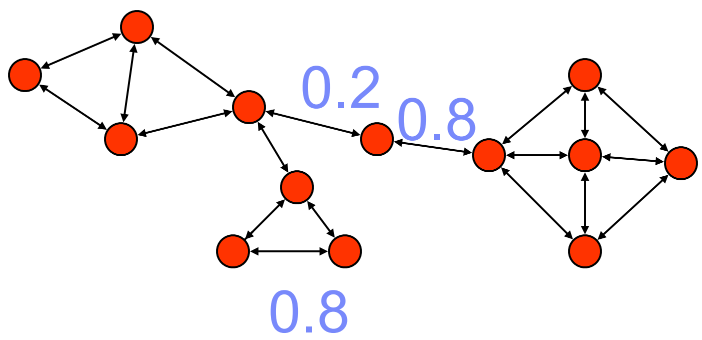
- Geodesic: another name for shortest path.

- Diameter: the value of the longest distance in a graph



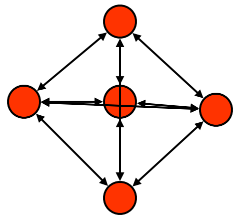
Decorated Graph

- Weighted Edges

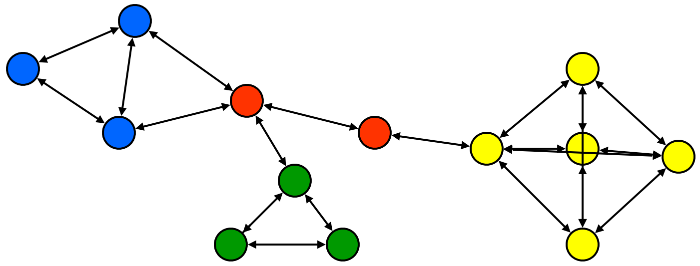


Families of Graphs

- Complete Graph: every vertex is linked to every other vertex.

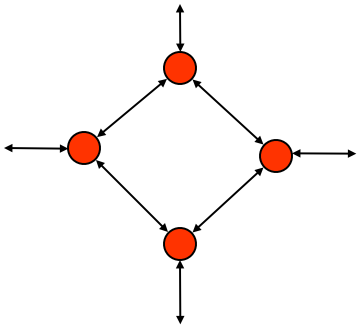


- Clique: a complete subgraph.



Regular Graph

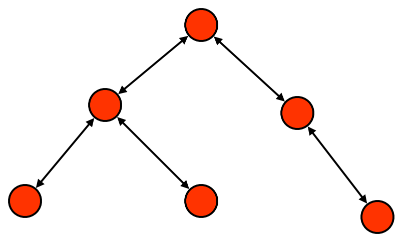
- Regular Graph: a graph in which every vertex has the same degree.



a 3-regular graph

Tree and Forest

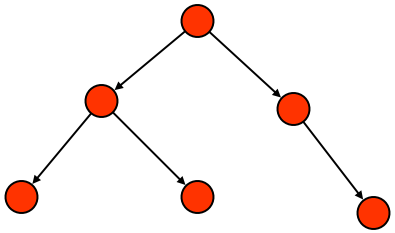
- Tree: a connected graph with no cycle.



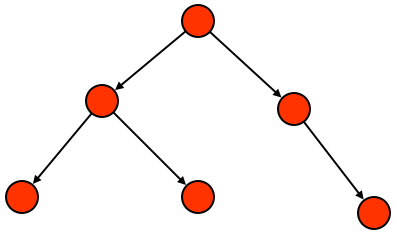
- Forest: a disjoint union of trees is called a forest.

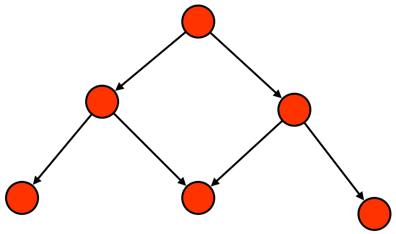
Labels in a directed tree

- Root
- Ancestor
- Descendant
- Parent
- Children
- Leaf: a vertex without children



Rooted Tree vs Directed Acyclic Graph (DAG)

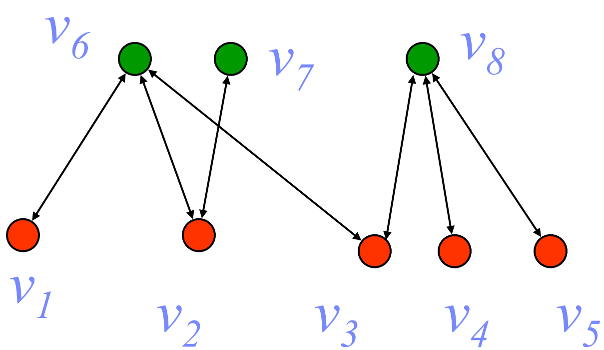




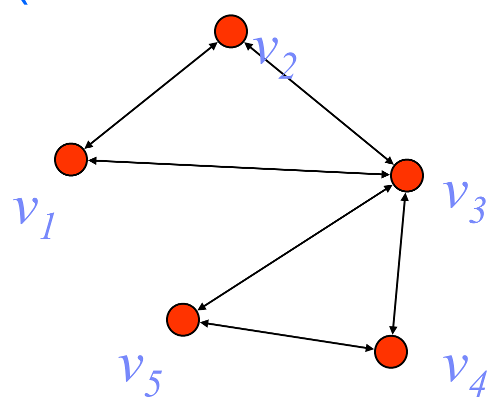
- DAG: Directed Acycle Graph. Underlining undirected graph has cycle.

Bipartite Graph

- Bipartite Graph: Vertices are partitioned into two sets. Edges link only between these two sets.



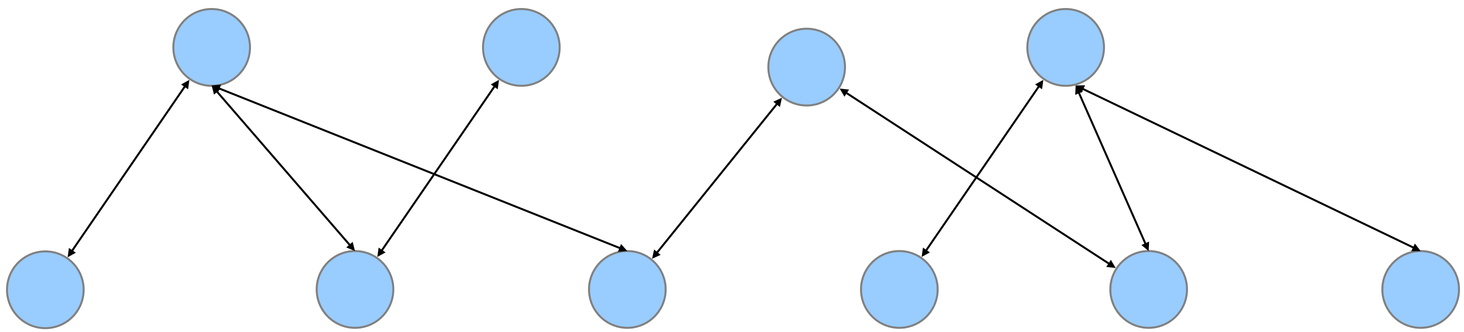
- Induced Graph (Collaborative Filtering):



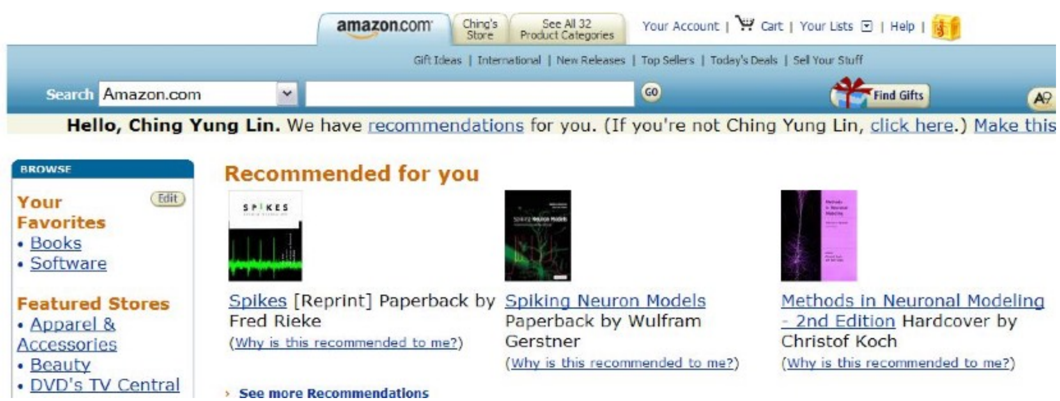
Recommendation Technique – Collaborative Filtering

item

user



= Recommendation

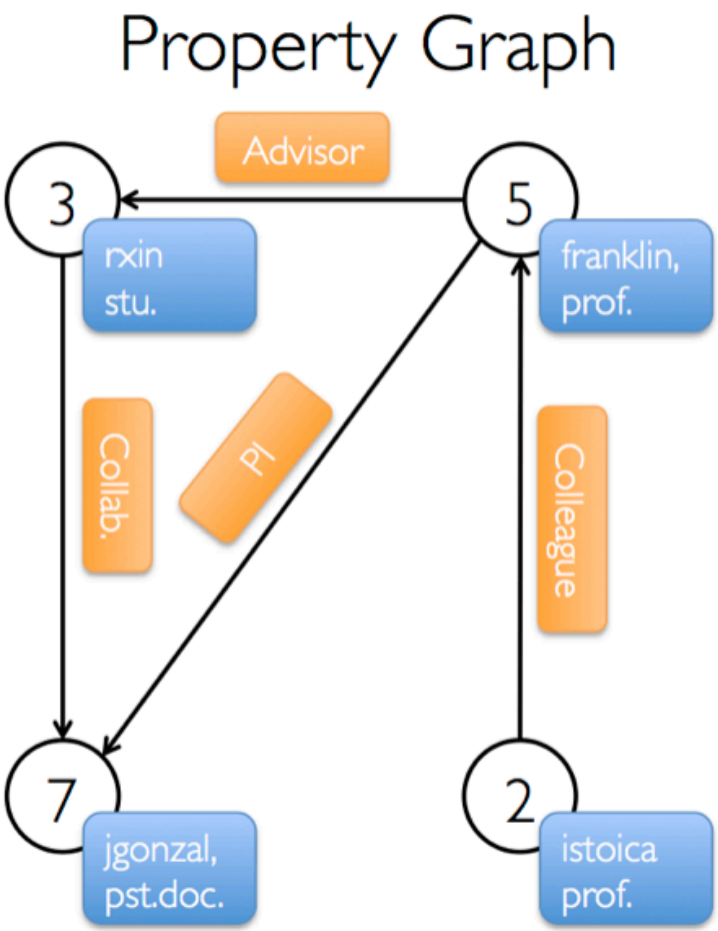


Graphs and Matrix Algebra

- The fundamental connectivity of a graph G may be captured in an $N_v \times N_v$ binary symmetric matrix $\boldsymbol{A}$ with entries:

$$A_{ij} = \begin{cases} 1, & if \ \{i, j\} \in E \\ 0, & otherwise \end{cases}$$

$\boldsymbol{A}$ is called the Adjacency Matrix of G



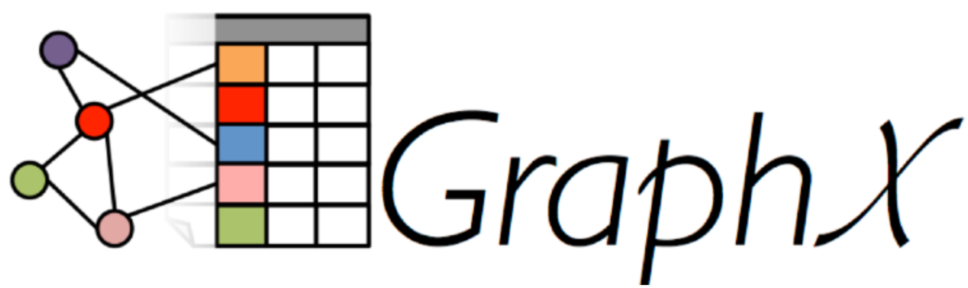
Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

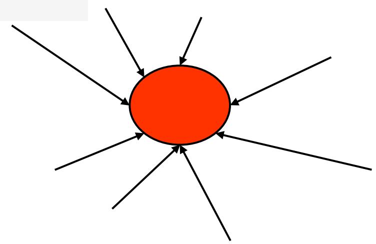
SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

- The Property Graph
  - Example Property Graph
- Graph Operators
  - Summary List of Operators
  - Property Operators
  - Structural Operators
  - Join Operators
  - Neighborhood Aggregation
    - Aggregate Messages (aggregateMessages)
    - Map Reduce Triplets Transition Guide (Legacy)
    - Computing Degree Information
    - Collecting Neighbors
  - Caching and Uncaching
- Pregel API
- Graph Builders
- Vertex and Edge RDDs
  - VertexRDDs
  - EdgeRDDs
- Optimized Representation
- Graph Algorithms
  - PageRank
  - Connected Components
  - Triangle Counting

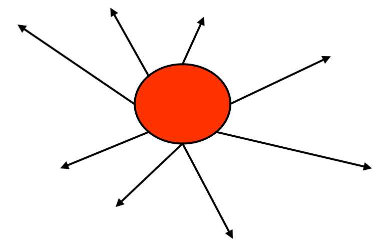


GraphX Graph Operations

```
// Information about the Graph ======
val numEdges: Long
val numVertices: Long
val inDegrees: VertexRDD[Int]
val outDegrees: VertexRDD[Int]
val degrees: VertexRDD[Int]
```

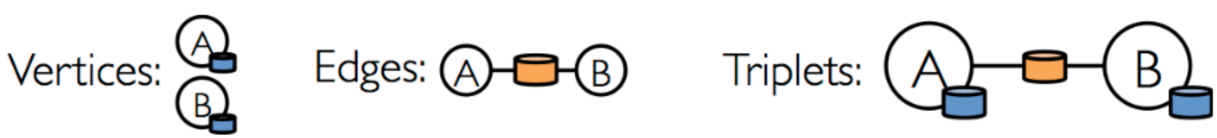


In-degree = 8



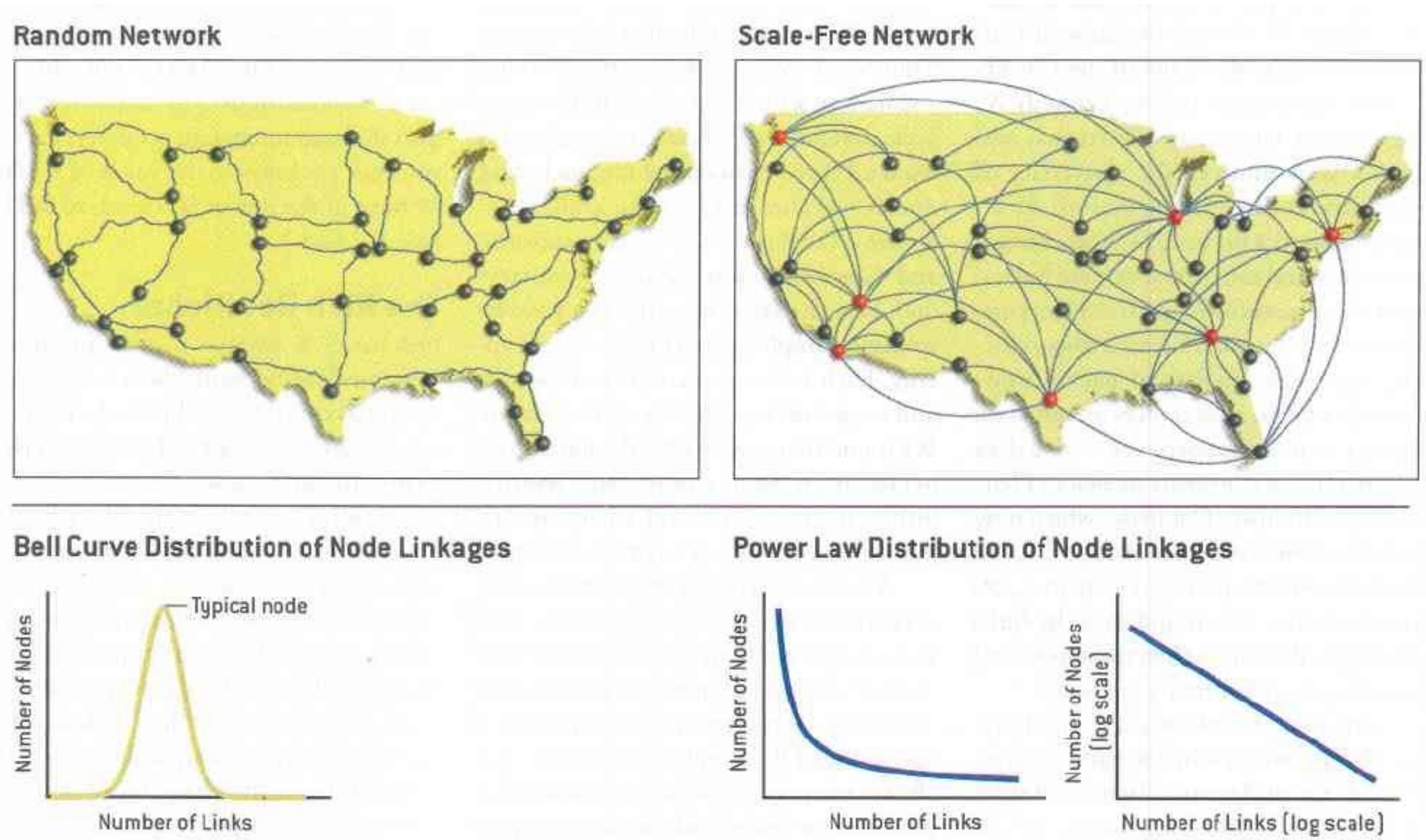
Out-degree = 8

```
// Views of the graph as collections ===
val vertices: VertexRDD[VD]
val edges: EdgeRDD[ED]
val triplets: RDD[EdgeTriplet[VD, ED]]
```



Degree Distribution Example: Power-Law Network

A. Barbasi and E. Bonabeau, “Scale-free Networks”, Scientific American 288: p.50-59, 2003.



$$p_k = e^{-m} \cdot \frac{m^k}{k!}$$

$$p_k = C \cdot k^{-\tau} e^{-k/\kappa}$$

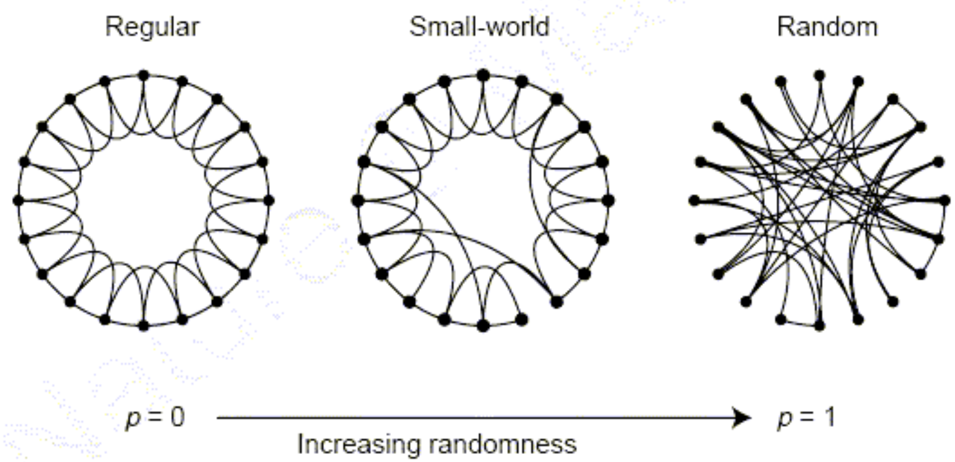
Newman, Strogatz and Watts, 2001

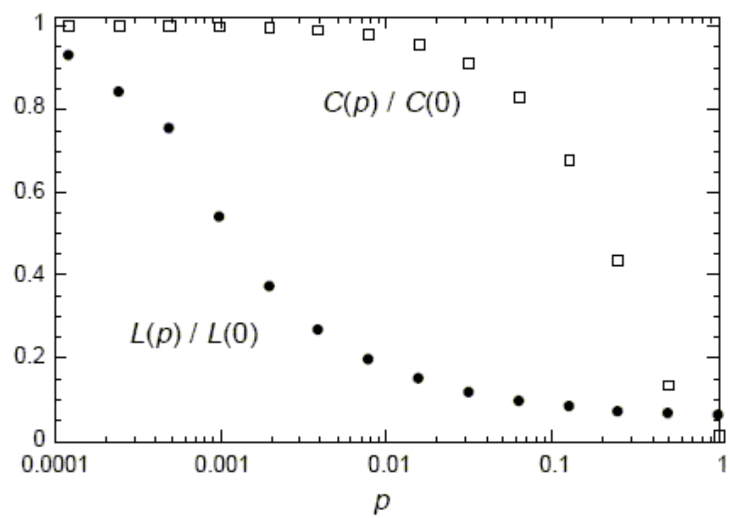
Another example of complex network: Small-World Network

Six Degree Separation:

adding long range link, a regular graph can be transformed into a small-world network, in which the average number of degrees between two nodes become small.

from Watts and Strogatz, 1998

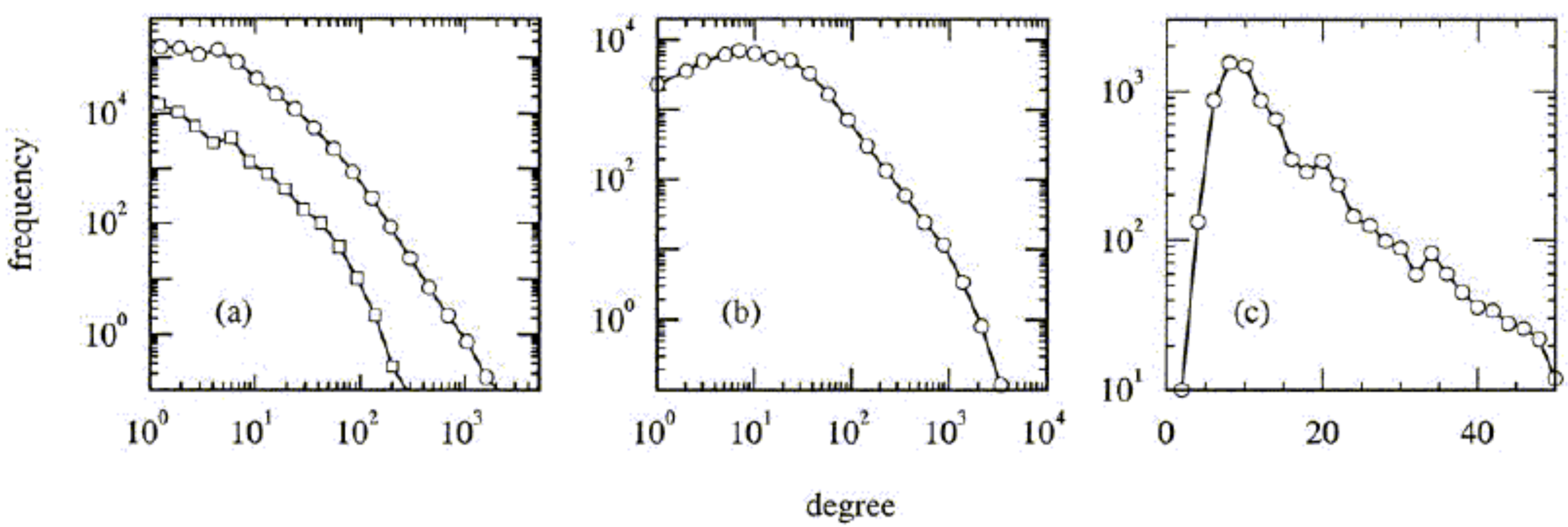




C: Clustering Coefficient, L: path length,
(C(0), L(0)): (C, L) as in a regular graph
(C(p), L(p)): (C,L) in a Small-world graph
with randomness p.

Some examples of Degree Distribution

(a) scientist collaboration: biologists (circle) physicists (square), (b) collaboration of move actors, (d) network of directors of Fortune 1000 companies



Basic graph algorithms in GraphX

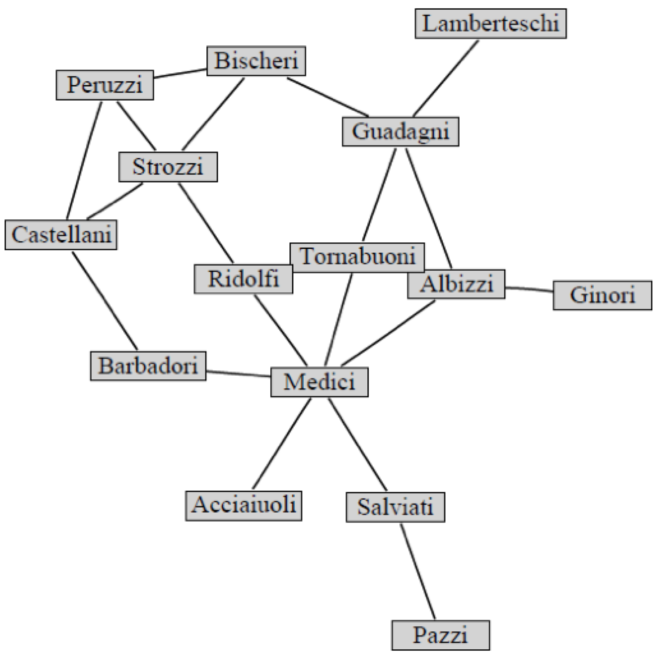
```
// Basic graph algorithms ========================================
def pageRank(tol: Double, resetProb: Double = 0.15): Graph[Double, Double]
def connectedComponents(): Graph[VertexId, ED]
def triangleCount(): Graph[Int, ED]
def stronglyConnectedComponents(numIter: Int): Graph[VertexId, ED]
```

Centrality

"There is certainly no unanimity on exactly what centrality is or its conceptual foundations, and there is little agreement on the procedure of its measurement." – Freeman 1979.

Degree (centrality)
Closeness (centrality)
Betweeness (centrality)
Eigenvector (centrality)

Network Analysis Example Centrality Ranking in Large Networks



[15th Century Florentine Family]

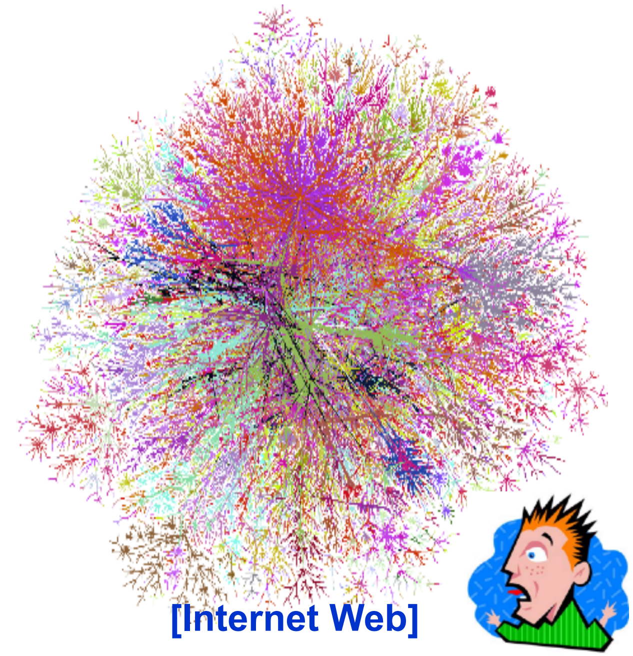
"Who are the most important actors?"

Three centralities

Degree: # of neighbor
Closeness: avg. shortest path length
Betweenness: # of times a node sits between shortest path

Application

Measuring the financial company value
Network attack monitoring



[Internet Web]

Small network	Complexity	Large network
|V| = 15, |E| = 19		|V| = Billions, |E| = Billions
Degree : Easy	O(|E|)	Degree : Easy
Closeness : Easy	O(|V|3)	Closeness : Hard
Betweenness : Easy	O(|V|2log|V|)	Betweenness : Hard

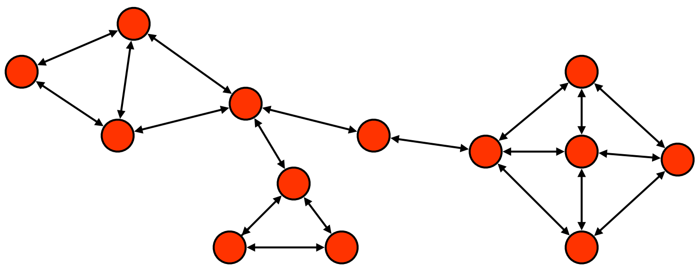
For 2 Billon Edges,
- standard closeness: 30,000 years

Closeness

Closeness: A vertex is 'close' to the other vertices

$$c_{CI}(v) = \frac{1}{\sum_{u \in V} dist(v,u)}$$

where dist(v,u) is the geodesic distance between vertices v and u.



Betweenness

Betweenness measures are aimed at summarizing the extent to which a vertex is located 'between' other pairs of vertices.

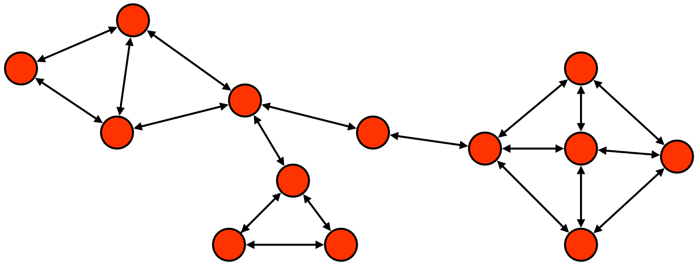
Freeman's definition:

$$c_B(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma(s,t \mid v)}{\sigma(s,t)}$$

Calculation of all betweenness centralities requires

calculating the lengths of shortest paths among all pairs of vertices

Computing the summation in the above definition for each vertex

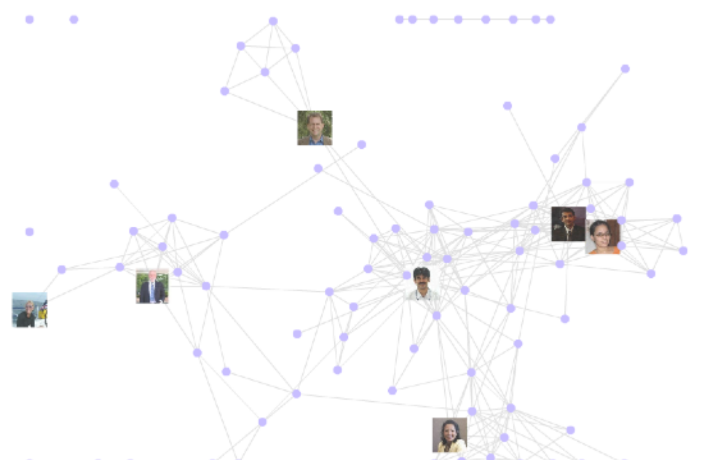


Betweenness ==> Bridges

Example: Healthcare experts in the world

Connections between different divisions

Example: Healthcare experts in the U.S.



Key social bridges

Eigenvector Centrality

Try to capture the 'status', 'prestige', or 'rank'.
More central the neighbors of a vertex are, the more central the vertex itself is.

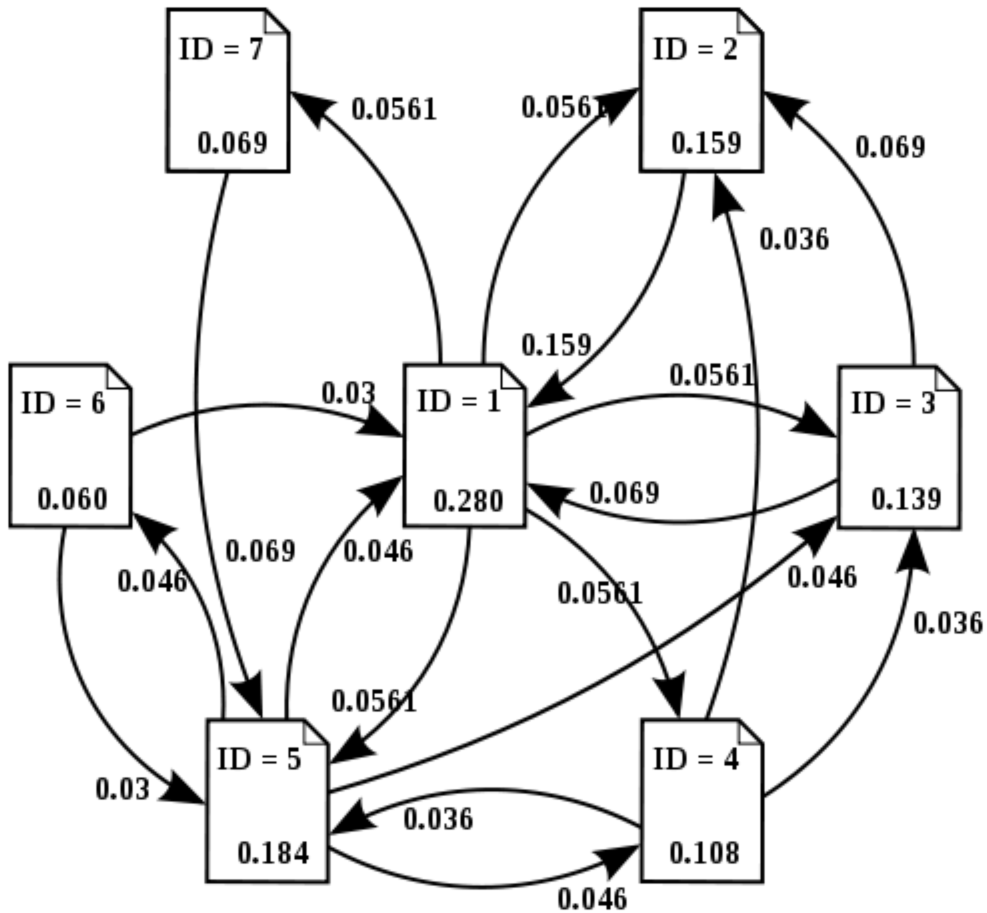
$$c_{Ei}(v) = \alpha \sum_{\{u,v\} \in E} c_{Ei}(u)$$

The vector $\mathbf{c}_{Ei} = (c_{Ei}(1), \ldots, c_{Ei}(N_v))^T$ is the solution of the

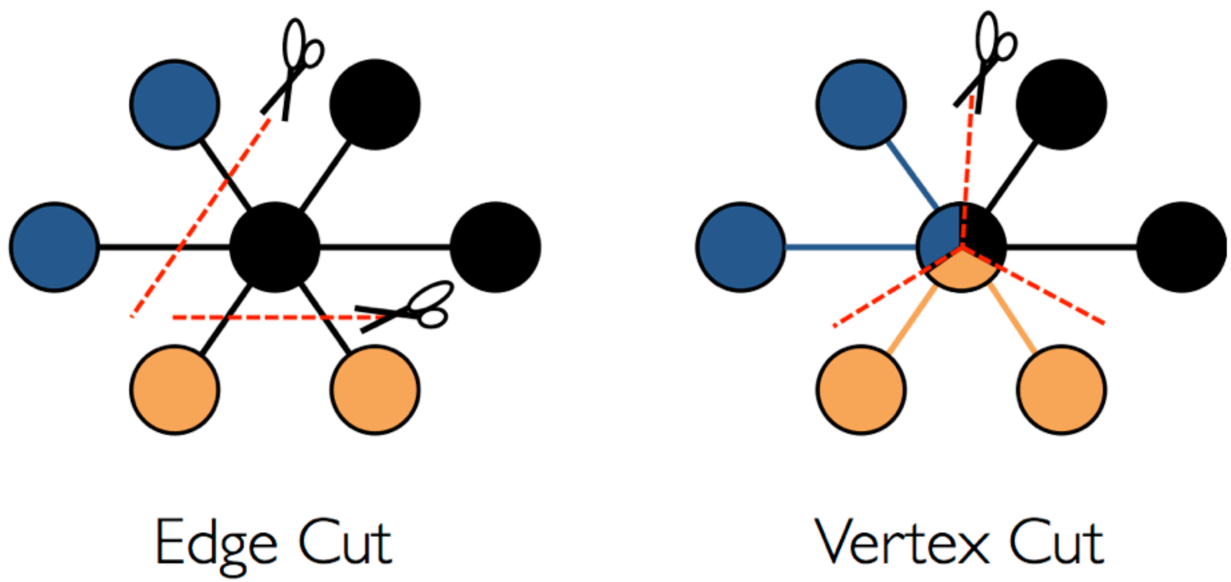
eigenvalue problem:

$$\mathbf{A} \cdot \mathbf{c}_{Ei} = \alpha^{-1} \mathbf{c}_{Ei}$$

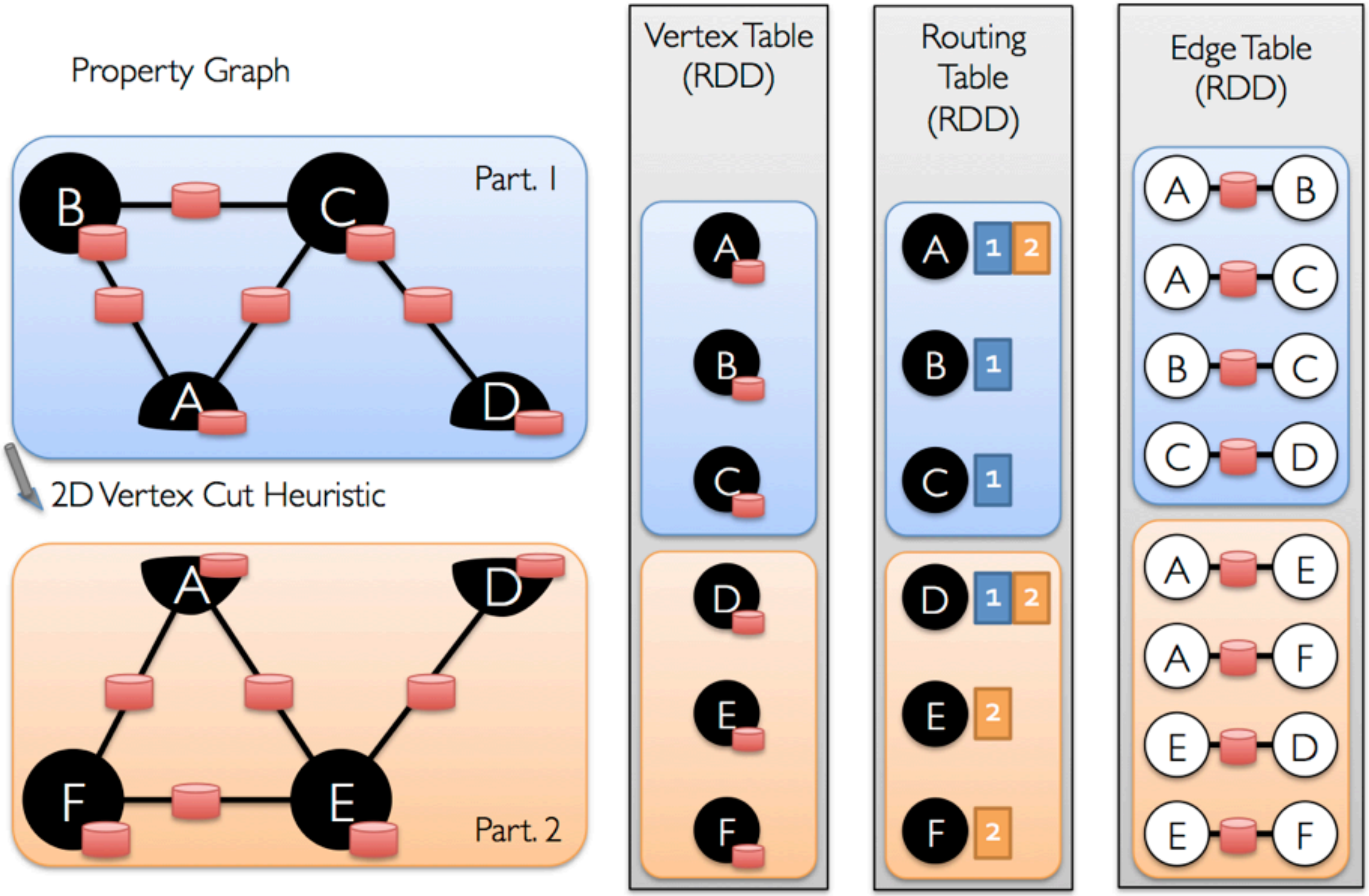
PageRank Algorithm (Simplified)



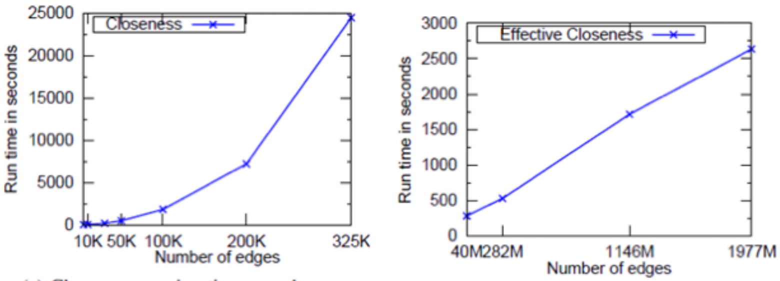
Graph Partitioning



Distributed Graph Computation in GraphX



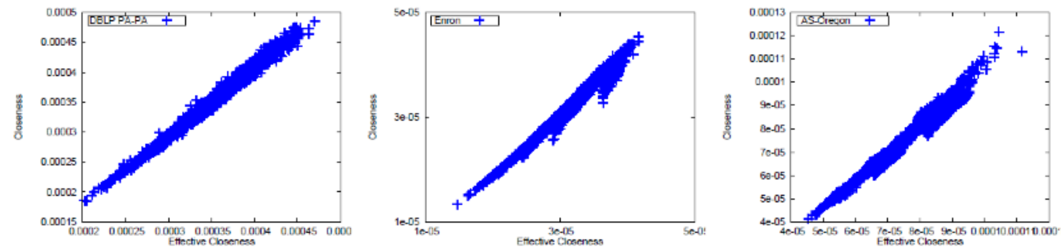
- Example -- we proposed two new centralities (`effective closeness' and `LineRank'), and efficient large scale algorithms for billion-scale graphs.



(a) Closeness: running time vs. edges (c) Effective Closeness: running time vs. edges

Scalability Results

(Near-linear scalability)

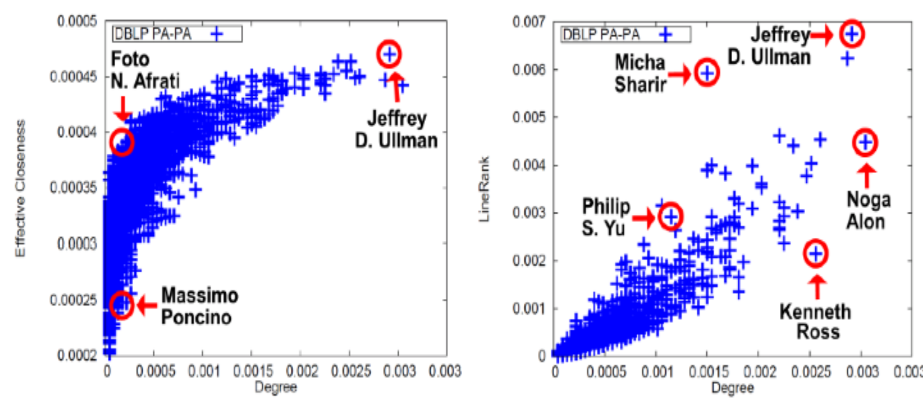


(a) DBLP Authors (b) Enron (c) AS-Oregon

Effective Closeness vs. Closeness

(Near-linear correlation (≥97.8%))

> ***For 2 Billon Edges,***
> ***- standard closeness: 30,000 years***
> ***- effective closeness: ~ 1 day !***
> ***1,000,000 times faster!***



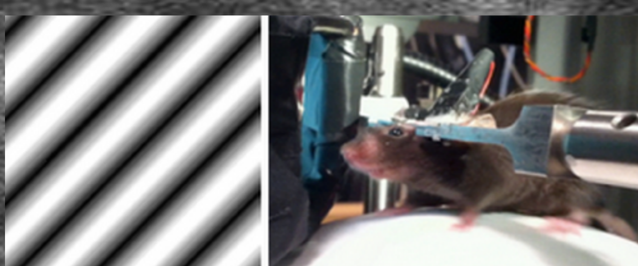
(a) DBLP Authors: Effective Closeness vs. Degree (b) DBLP Authors: LineRank vs. Degree

Analysis of Real-World Graph

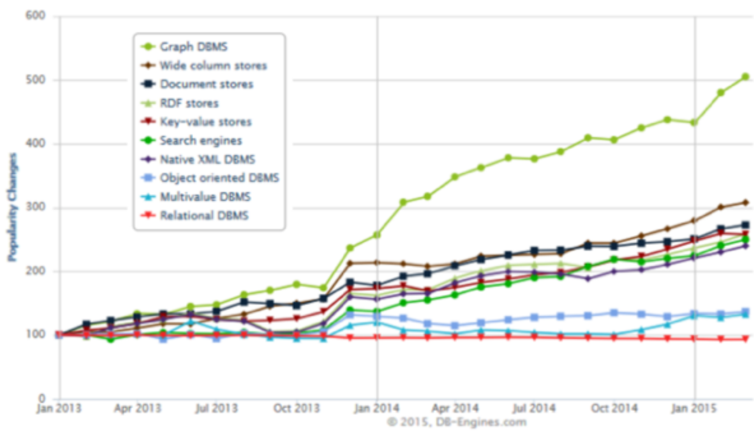
Kang, Tong, Sun, Lin, and Faloutsos, "GBase: A Scalable and general graph management system", KDD 2011

Graph Database — RDF and SPARQL

Network / Graph is the way we remember, we associate, and we understand.



Example: Graph Technology for Financial Service Sectors



- Graph Database is much more efficient than traditional relational database

- **How does FINRA analyze ~50B events per day TODAY?** - *Build a graph of market order events from multiple sources [ref]*

- **How did journalists uncover the Swiss Leak scandal in 2014 and also Panama Papers in 2016?** -- *Using graph database to uncover information thousands of accounts in more than 20 countries with links through millions of files [ref]*

RDF and SPARQL

WHAT DO RDF AND SPARQL BRING TO BIG DATA PROJECTS?

Bob DuCharme
TopQuadrant, Charlottesville, Virginia

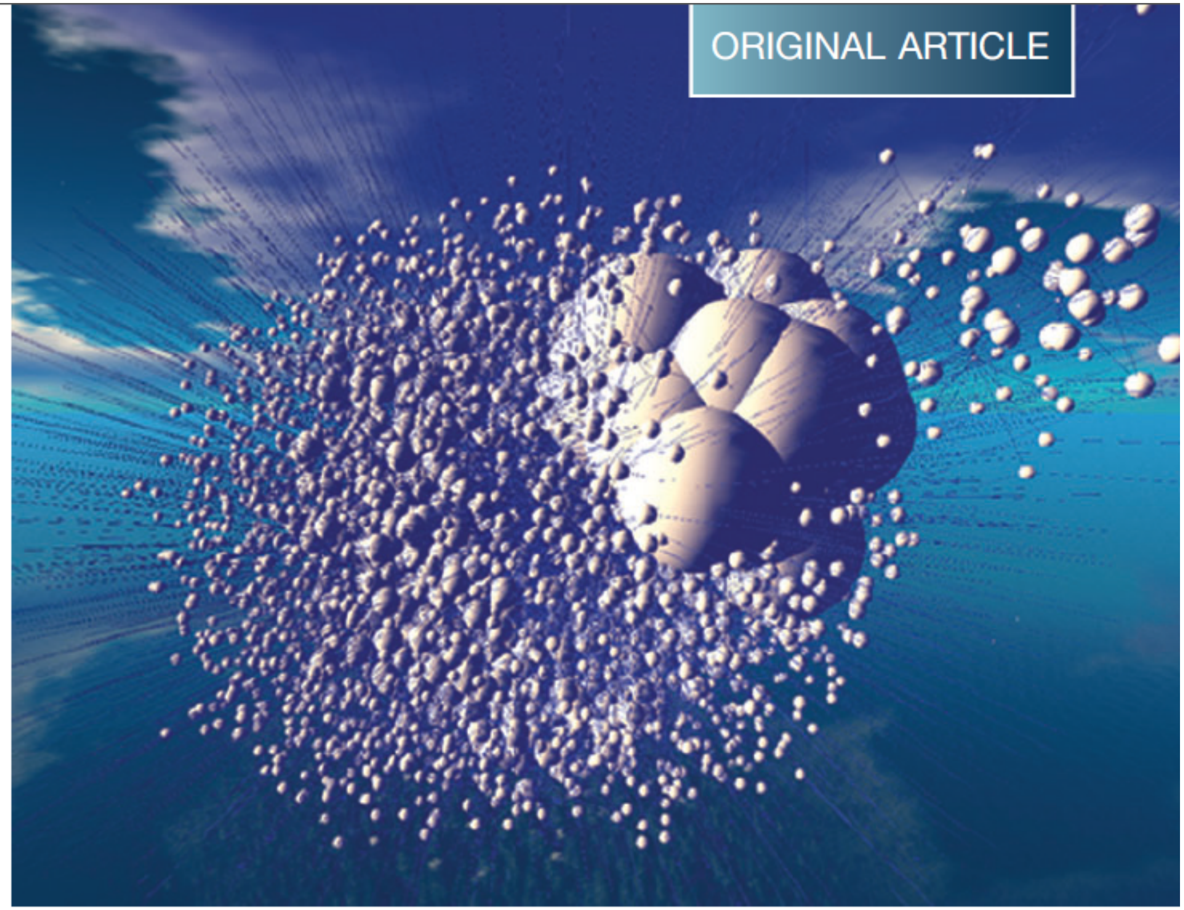


Photo Credit, Erich Bremer: http://www.ebremer.com/nexus/2011-05-15

Resource Description Format (RDF)

- A W3C standard since 1999
- Triples
- Example: A company has nine of part p1234 in stock, then a simplified triple representing this might be {p1234 inStock 9}.
- Instance Identifier, Property Name, Property Value.
- In a proper RDF version of this triple, the representation will be more formal. They require uniform resource identifiers (URIs).

```
@prefix fbd:<http://foobarco.net/data/>.
@prefix fbv:<http://foobarco.net/vocab/>.

fbd:p1234 fbv:inStock "9".
fbd:p1234 fbv:supplier "Joe's Part Company".
```

An example complete description

```
@prefix fbd:<http://foobarco.net/data/> .
@prefix fbv:<http://foobarco.net/vocab/> .
fbd:p1234 fbv:inStock "9".
fbd:p1234 fbv:name "Blue reverse flange".
fbd:p1234 fbv:supplier fbd:s9483.
fbd:s9483 fbv:name "Joe's Part Company".
fbd:s9483 fbv:homePage "http://www.joespartco.com".
fbd:s9483 fbv:contactName "Gina Smith".
fbd:s9483 fbv:contactEmail "gina.smith@joespartco.com".
```

Advantages of RDF

- Virtually any RDF software can parse the lines shown above as self-contained, working data file.
  - You can declare properties if you want.
  - The RDF Schema standard lets you declare classes and relationships between properties and classes.
  - The flexibility that the lack of dependence on schemas is the first key to RDF's value.

- Split trips into several lines that won't affect their collective meaning, which makes sharding of data collections easy.
  - Multiple datasets can be combined into a usable whole with simple concatenation.

- For the inventory dataset's property name URIs, sharing of vocabulary makes easy to aggregate.

SPARQL – Query Langauge for RDF

The following SPQRL query asks for all property names and values associated with the fbd:s9483 resource:

```
PREFIX fbd:<http://foobarco.net/data/>

SELECT ?property ?value
WHERE {fbd:s9483 ?property ?value.}
```

The heart of any SPARQL query is the WHERE clause, which specifies the triples to pull out of the dataset. Various options for the rest of the query tell the SPARQL processor what to do with those triples, such as sorting, creating, or deleting triples. The above query's WHERE clause has a single triple pattern, which resembles a triple but may have variables substituted for any or all of the triple's three parts. The triple pattern above says that we're interested in triples that have fbd:s9483 as the subject and—because variables function as wildcards—anything at all in the triple's second and third parts.

The SPAQRL Query Result from the previous example

```
-----------------------------------------------------------------------------------------
|property                                     |value                                     |
=========================================================================================
|<http://foobarco.net/vocab/contactEmail>     |"gina.smith@joespartco.com"               |
|<http://foobarco.net/vocab/contactName>      |"Gina Smith"                              |
|<http://foobarco.net/vocab/homePage>         |"http://www.joespartco.com"               |
|<http://foobarco.net/vocab/name>             |"Joe's Part Company"                      |
-----------------------------------------------------------------------------------------
```

Another SPARQL Example

What is this query for?

```
PREFIX fbd: <http://foobarco.net/data/>
PREFIX fbv: <http://foobarco.net/vocab/>

SELECT ?flangeContactEmail
WHERE
{
  ?part fbv:name "Blue reverse flange".
  ?part fbv:supplier ?supplier.
  ?supplier fbv:contactEmail ?flangeContactEmail.
}
```

Data

```
@prefix fbd: <http://foobarco.net/data/>.
@prefix fbv: <http://foobarco.net/vocab/>.
fbd:p1234 fbv:inStock "9".
fbd:p1234 fbv:name "Blue reverse flange".
fbd:p1234 fbv:supplier fbd:s9483.
fbd:s9483 fbv:name "Joe's Part Company".
fbd:s9483 fbv:homePage "http://www.joespartco.com".
fbd:s9483 fbv:contactName "Gina Smith".
fbd:s9483 fbv:contactEmail "gina.smith@joespartco.com".
```

Apache Jena | Home | Download | Learn | Ask | Get involved

Apache Jena

A free and open source Java framework for building Semantic Web and Linked Data applications.

Get started now! | Download

RDF

RDF API

Interact with the core API to create and read Resource Description Framework (RDF) graphs. Serialise your triples using popular formats such as RDF/XML or Turtle.

ARQ (SPARQL)

Query your RDF data using ARQ, a SPARQL 1.1 compliant engine. ARQ supports remote federated

Triple store

TDB

Persist your data using TDB, a native high performance triple store. TDB supports the full range of Jena APIs.

Fuseki

Expose your triples as a SPARQL end-point accessible over HTTP. Fuseki provides REST-style interaction with your RDF data.

OWL

Ontology API

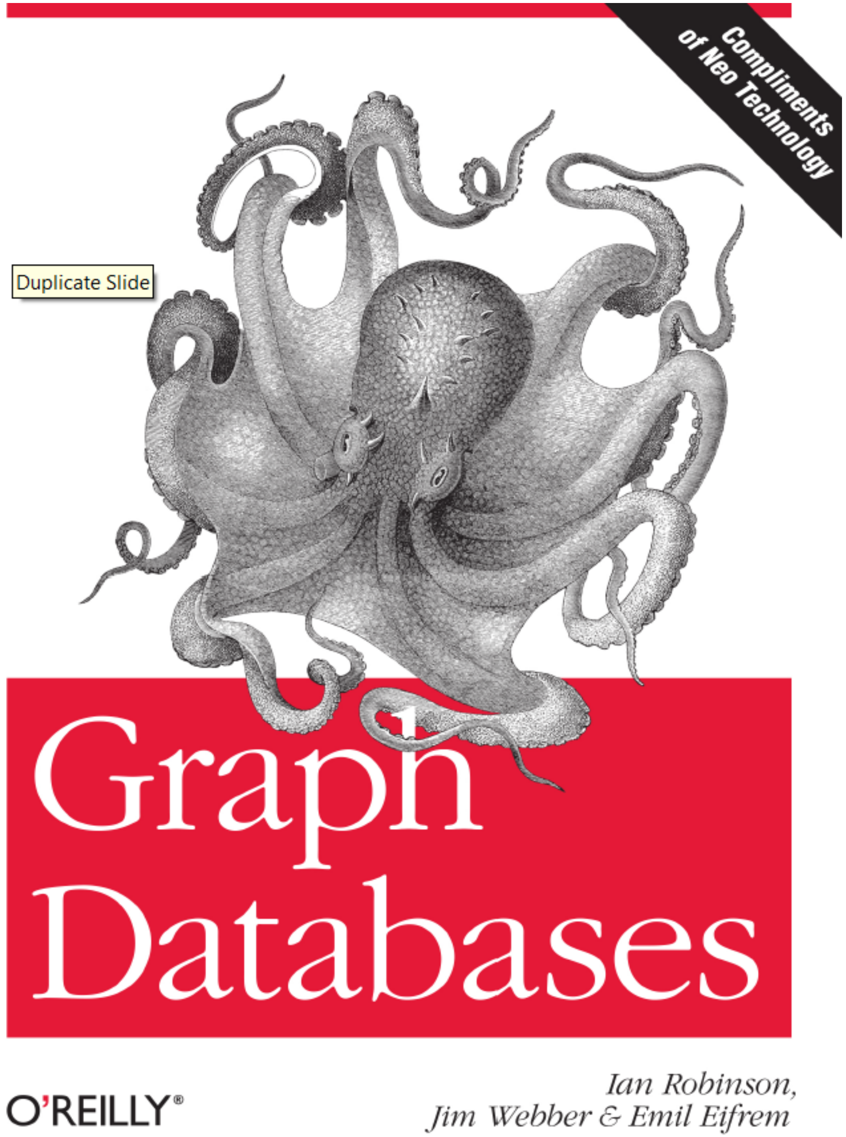
Work with models, RDFS and the Web Ontology Language (OWL) to add extra semantics to your RDF data.

Inference API

Reason over your data to expand and check the content of your triple store. Configure your own inference rules or

Graph Database — Property Graphs

Reference


Compliments of Neo Technology
Duplicate Slide
Graph Databases
O'REILLY®
Ian Robinson,
Jim Webber & Emil Eifrem

A usual example

User

UserID	User	Address	Phone	Email	Alternate
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
...	...	...	...	...	...
99	Zach	99 South St.		zach@example.org	

Order

OrderID	UserID
1234	1
5678	1
...	...
5588	99

LineItem

OrderID	ProductID	Quantity
1234	765	2
1234	987	1
...	...	...
5588	765	1

Product

ProductID	Description	Handling
321	strawberry ice cream	freezer
765	potatoes	
...	...	
987	dried spaghetti	

Figure 2-1. Semantic relationships are hidden in a relational database

Query Example – I

Person

ID	Person
1	Alice
2	Bob
...	...
99	Zach

PersonFriend

PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

Figure 2-2. Modeling friends and friends-of-friends in a relational database

Asking "who are Bob's friends?" is easy, as shown in Example 2-1.

Example 2-1. Bob's friends

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.FriendID = p1.ID
JOIN Person p2
  ON PersonFriend.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```

Query Examples – II & III

Example 2-2. Who is friends with Bob?

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.PersonID = p1.ID
JOIN Person p2
  ON PersonFriend.FriendID = p2.ID
WHERE p2.Person = 'Bob'
```

Example 2-3. Alice's friends-of-friends

```
SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
  ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
  ON pf2.PersonID = pf1.FriendID
JOIN Person p2
  ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

Computational intensive

Graph Database Example

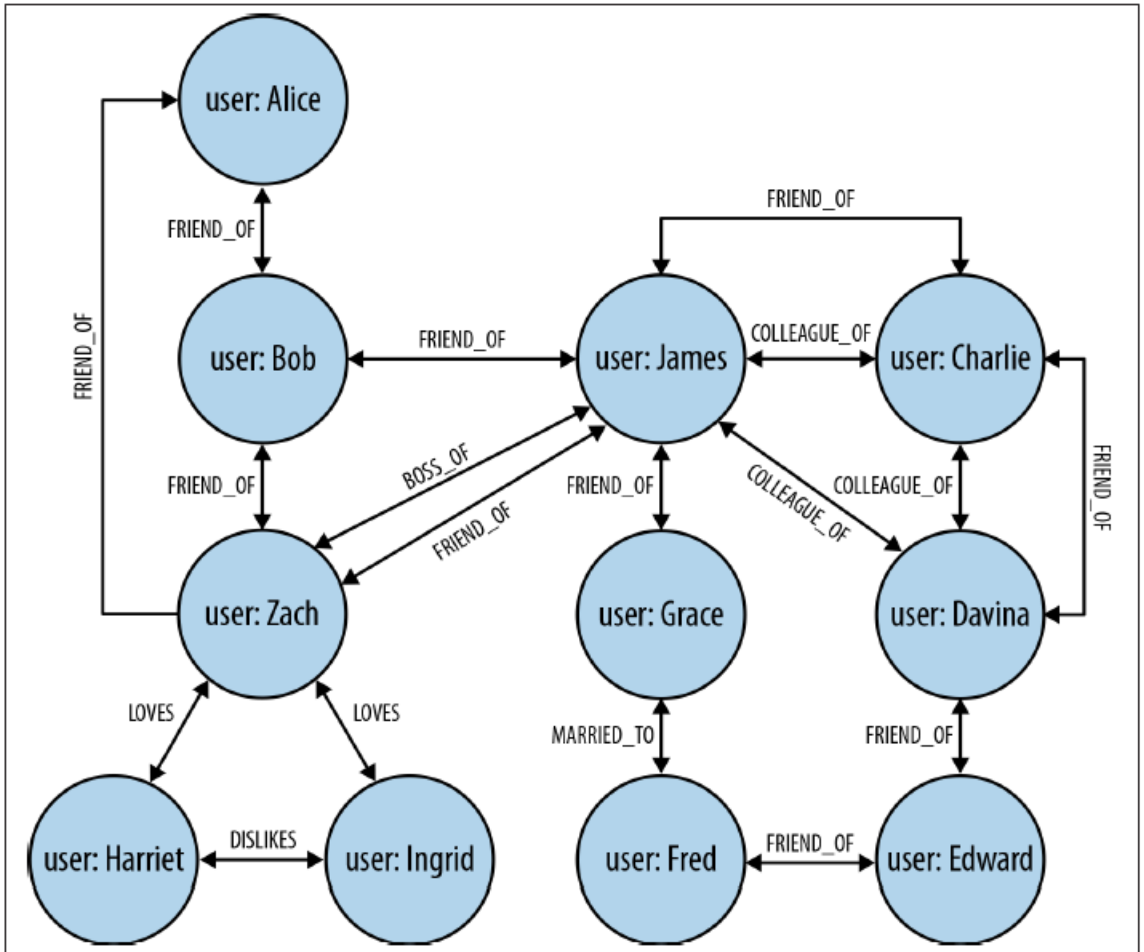


Figure 2-5. Easily modeling friends, colleagues, workers, and (unrequited) lovers in a graph

Execution Time in the example of finding extended friends (by Neo4j)

Partner and Vukotic's experiment seeks to find friends-of-friends in a social network, to a maximum depth of five. Given any two persons chosen at random, is there a path that connects them that is at most five relationships long? For a social network containing 1,000,000 people, each with approximately 50 friends, the results strongly suggest that graph databases are the best choice for connected data, as we see in Table 2-1.

Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

Depth	RDBMS execution time (s)	Neo4j execution time (s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Modeling Order History as a Graph

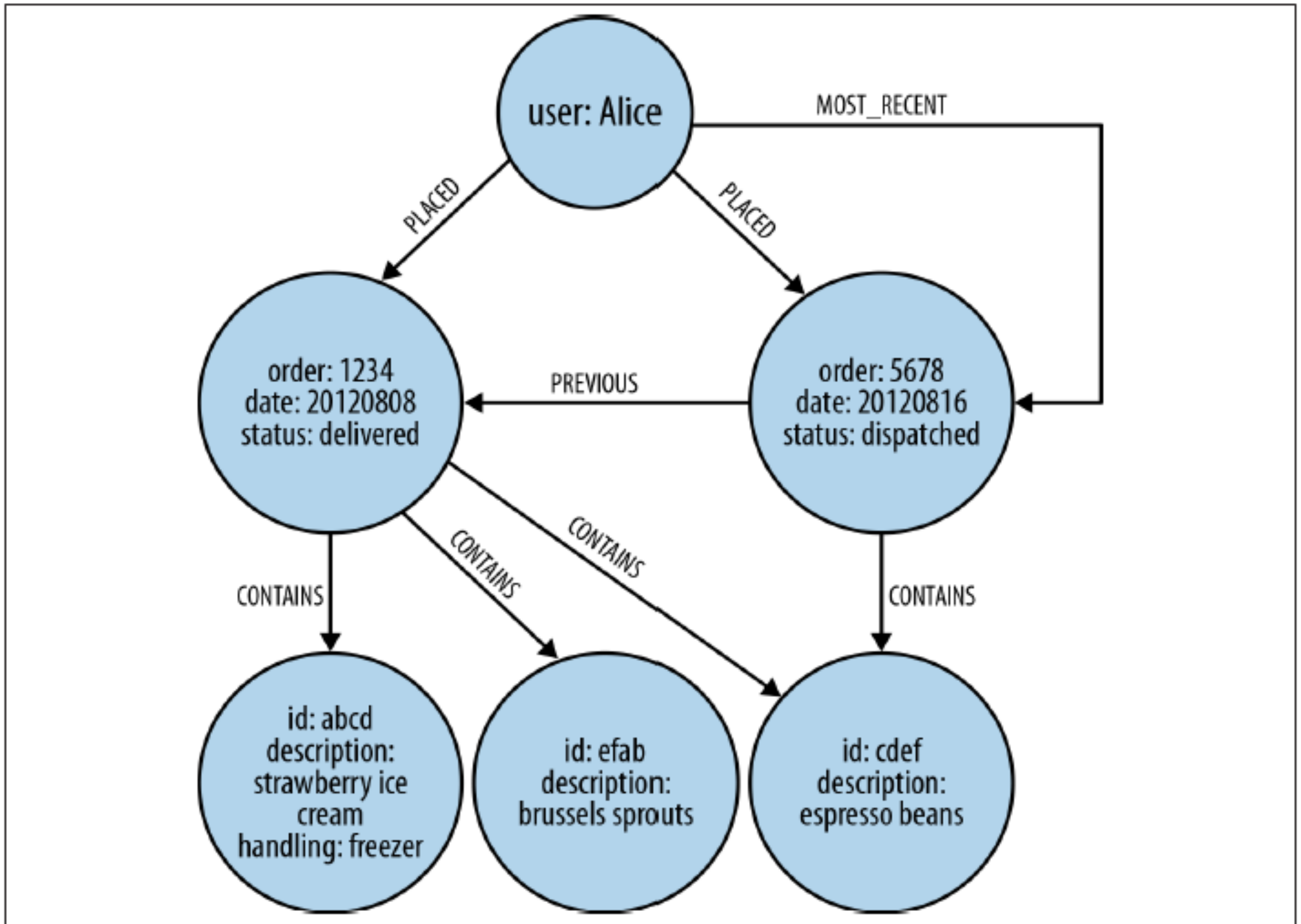


Figure 2-6. Modeling a user's order history in a graph

A query language on Property Graph – Cypher

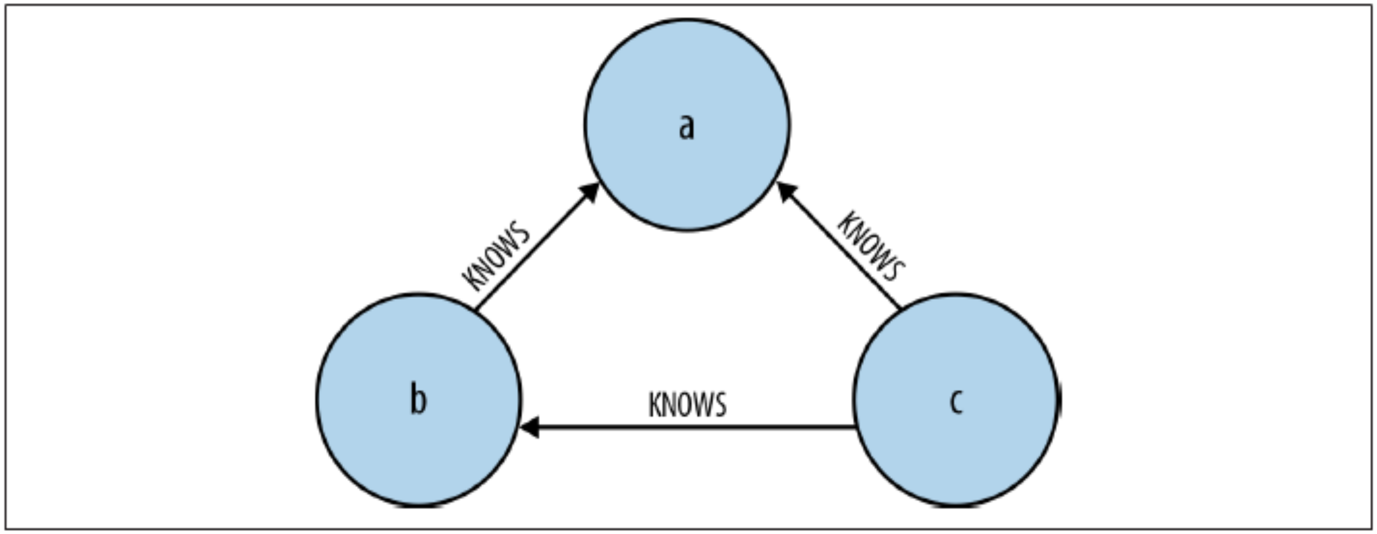


Figure 3-1. A simple graph pattern, expressed using a diagram

This pattern describes three mutual friends. Here's the equivalent ASCII art representation in Cypher:

```
(a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
```

Like most query languages, Cypher is composed of clauses. The simplest queries consist of a `START` clause followed by a `MATCH` and a `RETURN` clause (we'll describe the other clauses you can use in a Cypher query later in this chapter). Here's an example of a Cypher query that uses these three clauses to find the mutual friends of user named *`Michael`*:

```
START a=node:user(name='Michael')
MATCH (a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
RETURN b, c
```


Other Cypher Clauses

`WHERE`

Provides criteria for filtering pattern matching results.

`CREATE` *and* `CREATE UNIQUE`

Create nodes and relationships.

`DELETE`

Removes nodes, relationships, and properties.

`SET`

Sets property values.

`FOREACH`

Performs an updating action for each element in a list.

`UNION`

Merges results from two or more queries (introduced in Neo4j 2.0).

`WITH`

Chains subsequent query parts and forward results from one to the next. Similar to piping commands in Unix.

Property Graph Example – Shakespeare

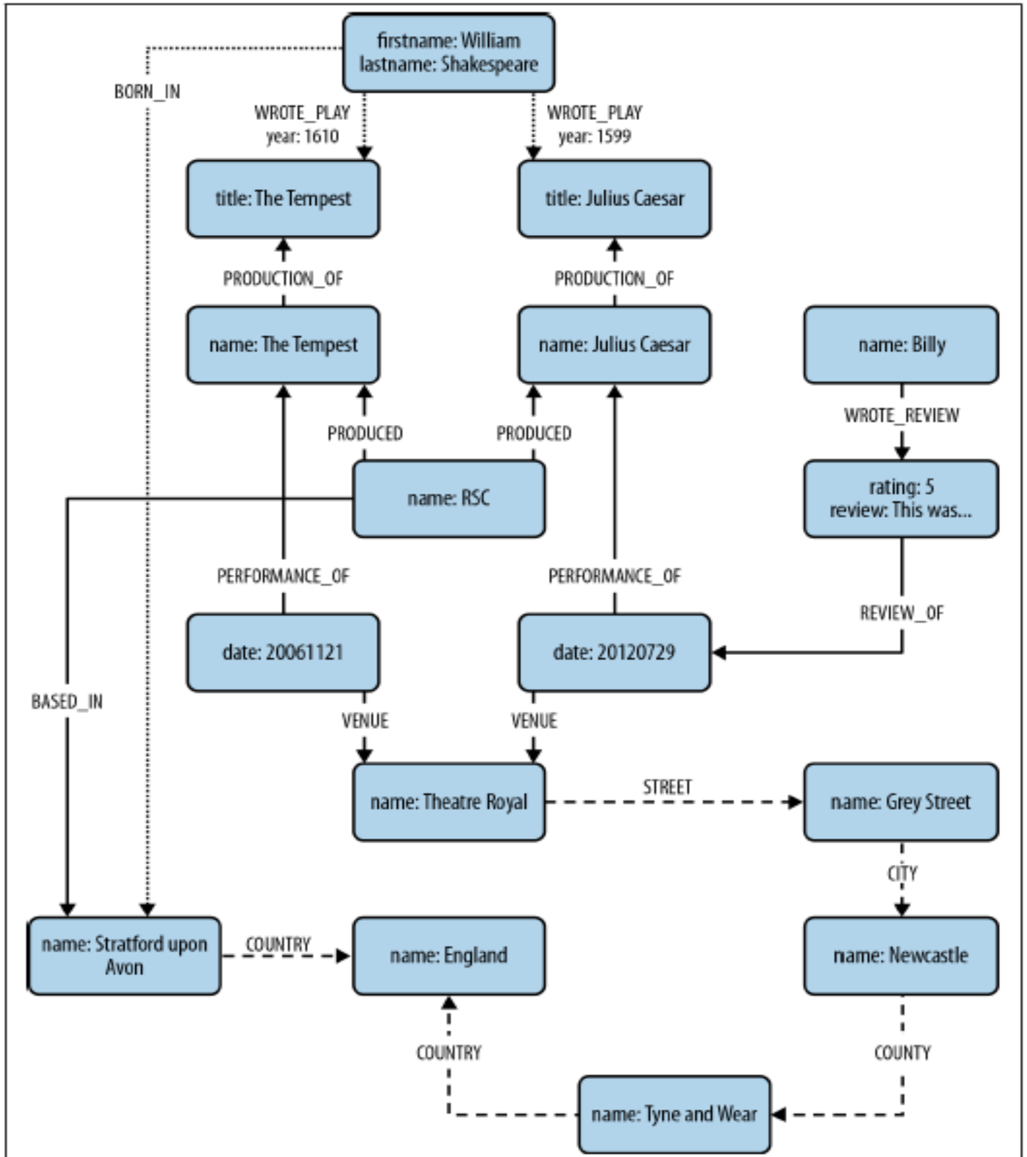


Figure 3-6. Three domains in one graph

Creating the Shakespeare Graph

```
CREATE (shakespeare { firstname: 'William', lastname: 'Shakespeare' }),
       (juliusCaesar { title: 'Julius Caesar' }),
       (shakespeare)-[:WROTE_PLAY { year: 1599 }]->(juliusCaesar),
       (theTempest { title: 'The Tempest' }),
       (shakespeare)-[:WROTE_PLAY { year: 1610}]->(theTempest),
       (rsc { name: 'RSC' }),
       (production1 { name: 'Julius Caesar' }),
       (rsc)-[:PRODUCED]->(production1),
       (production1)-[:PRODUCTION_OF]->(juliusCaesar),
       (performance1 { date: 20120729 }),
       (performance1)-[:PERFORMANCE_OF]->(production1),
       (production2 { name: 'The Tempest' }),
       (rsc)-[:PRODUCED]->(production2),
       (production2)-[:PRODUCTION_OF]->(theTempest),
       (performance2 { date: 20061121 }),
       (performance2)-[:PERFORMANCE_OF]->(production2),
       (performance3 { date: 20120730 }),
       (performance3)-[:PERFORMANCE_OF]->(production1),
       (billy { name: 'Billy' }),
       (review { rating: 5, review: 'This was awesome!' }),
       (billy)-[:WROTE_REVIEW]->(review),
       (review)-[:RATED]->(performance1),
       (theatreRoyal { name: 'Theatre Royal' }),
       (performance1)-[:VENUE]->(theatreRoyal),
       (performance2)-[:VENUE]->(theatreRoyal),
       (performance3)-[:VENUE]->(theatreRoyal),
       (greyStreet { name: 'Grey Street' }),
       (theatreRoyal)-[:STREET]->(greyStreet),
       (newcastle { name: 'Newcastle' }),
       (greyStreet)-[:CITY]->(newcastle),
       (tyneAndWear { name: 'Tyne and Wear' }),
       (newcastle)-[:COUNTY]->(tyneAndWear),
       (england { name: 'England' }),
       (tyneAndWear)-[:COUNTRY]->(england),
       (stratford { name: 'Stratford upon Avon' }),
       (stratford)-[:COUNTRY]->(england),
       (rsc)-[:BASED_IN]->(stratford),
       (shakespeare)-[:BORN_IN]->stratford
```

Query on the Shakespeare Graph

```
START theater=node:venue(name='Theatre Royal'),
      newcastle=node:city(name='Newcastle'),
      bard=node:author(lastname='Shakespeare')
MATCH (newcastle)<-[:STREET|CITY*1..2]-(theater)
      <-[:VENUE]-()-[:PERFORMANCE_OF]->()-[:PRODUCTION_OF]->
      (play)<-[w:WROTE_PLAY]-(bard)
WHERE w.year > 1608
RETURN DISTINCT play.title AS play
```

Adding this `WHERE` clause means that for each successful match, the Cypher execution engine checks that the `WROTE_PLAY` relationship between the Shakespeare node and the matched play has a `year` property with a value greater than 1608. Matches with a `WROTE_PLAY` relationship whose `year` value is greater than 1608 will pass the test; these plays will then be included in the results. Matches that fail the test will not be included in the results. By adding this clause, we ensure that only plays from Shakespeare's late period are returned:

```
+---------------+
| play          |
+---------------+
| "The Tempest" |
+---------------+
1 row
```

Another Query on the Shakespeare Graph

```
START theater=node:venue(name='Theatre Royal'),
      newcastle=node:city(name='Newcastle'),
      bard=node:author(lastname='Shakespeare')
MATCH (newcastle)<-[:STREET|CITY*1..2]-(theater)
      <-[:VENUE]-()-[p:PERFORMANCE_OF]->()-[:PRODUCTION_OF]->
      (play)<-[:WROTE_PLAY]-(bard)
RETURN   play.title AS play, count(p) AS performance_count
ORDER BY performance_count DESC
```

The `RETURN` clause here counts the number of `PERFORMANCE_OF` relationships using the identifier `p` (which is bound to the `PERFORMANCE_OF` relationships in the `MATCH` clause) and aliases the result as `performance_count`. It then orders the results based on `performance_count`, with the most frequently performed play listed first:

```
+-----------------------------------+
| play            | performance_count |
+-----------------------------------+
| "Julius Caesar" | 2                 |
| "The Tempest"   | 1                 |
+-----------------------------------+
2 rows
```

Building Application Example – Collaborative Filtering

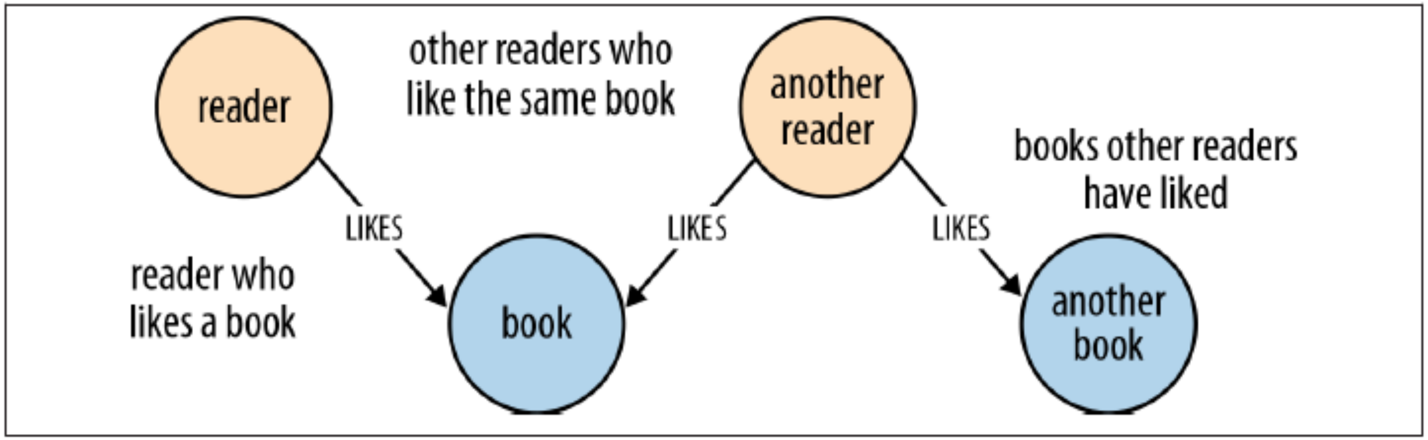


Figure 4-1. Data model for the book reviews user story

Because this data model directly encodes the question presented by the user story, it lends itself to being queried in a way that similarly reflects the structure of the question we want to ask of the data:

```
START reader=node:users(name={readerName})
      book=node:books(isbn={bookISBN})
MATCH reader-[:LIKES]->book<-[:LIKES]-other_readers-[:LIKES]->books
RETURN books.title
```


Chaining on the Query

```
START bard=node:author(lastname='Shakespeare')
MATCH (bard)-[w:WROTE_PLAY]->(play)
WITH play
ORDER BY w.year DESC
RETURN collect(play.title) AS plays
```

Executing this query against our sample graph produces the following result:

```
+-------------------------------+
| plays                         |
+-------------------------------+
| ["The Tempest","Julius Caesar"] |
+-------------------------------+
1 row
```

Example – Email Interaction Graph

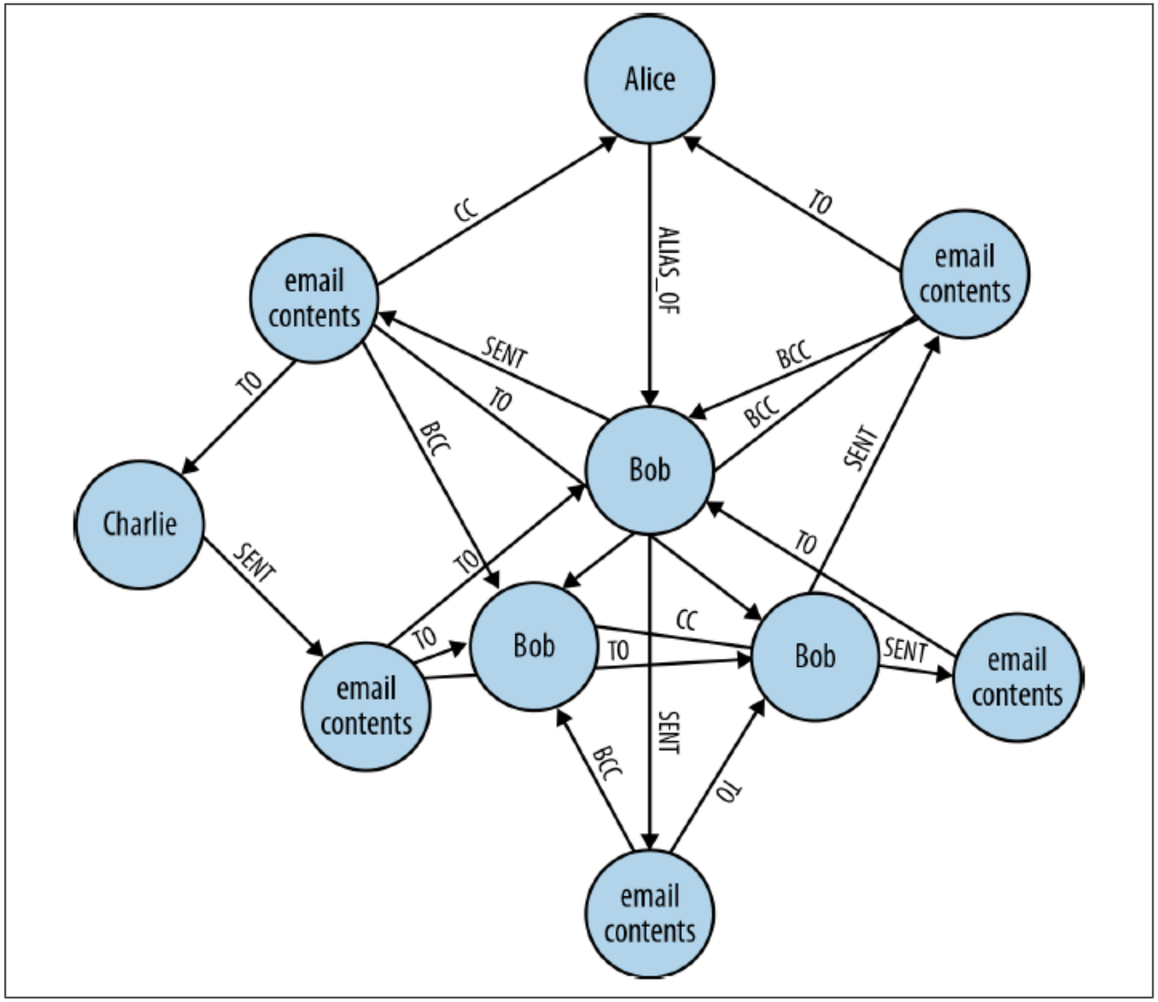


Figure 3-10. A graph of email interactions

```
START bob=node:user(username='Bob')
MATCH (bob)-[:SENT]->(email)-[:CC]->(alias),
      (alias)-[:ALIAS_OF]->(bob)
RETURN email
```

What's this query for?

How to make graph database fast?

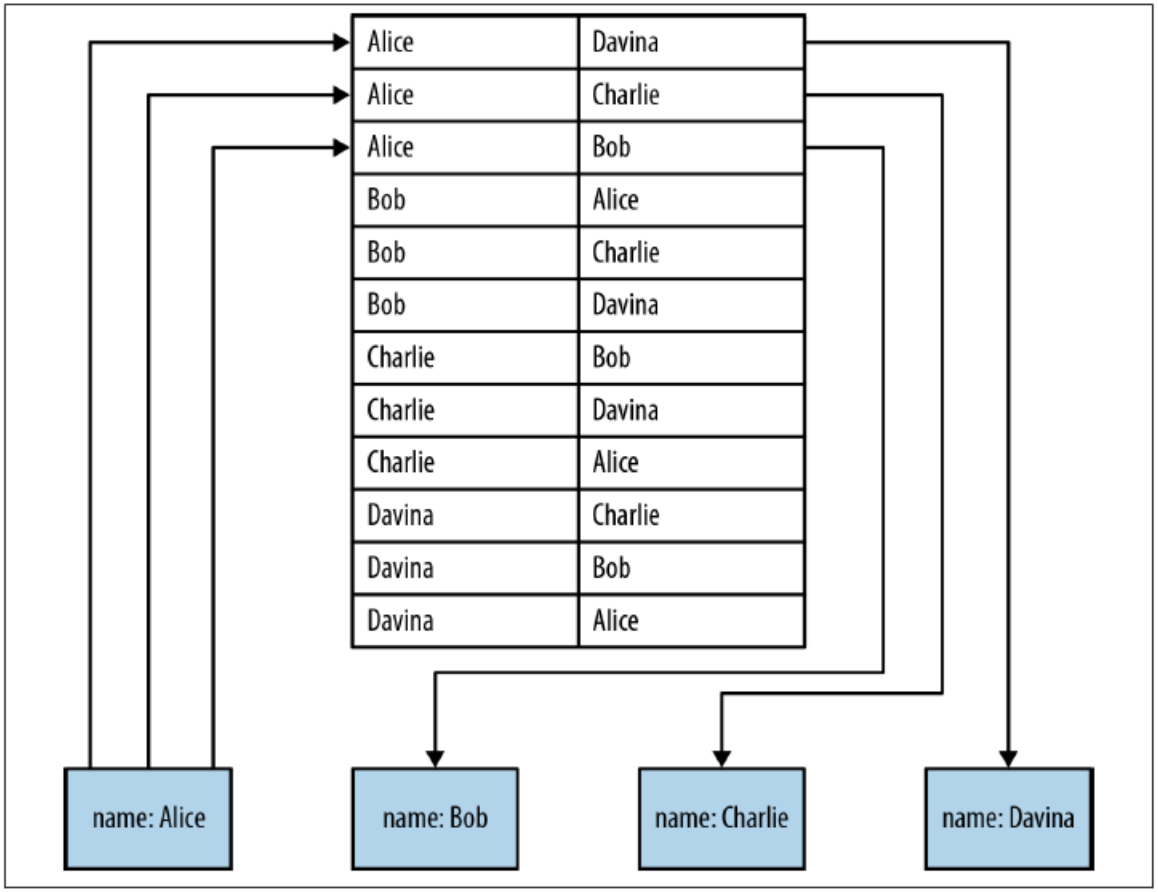
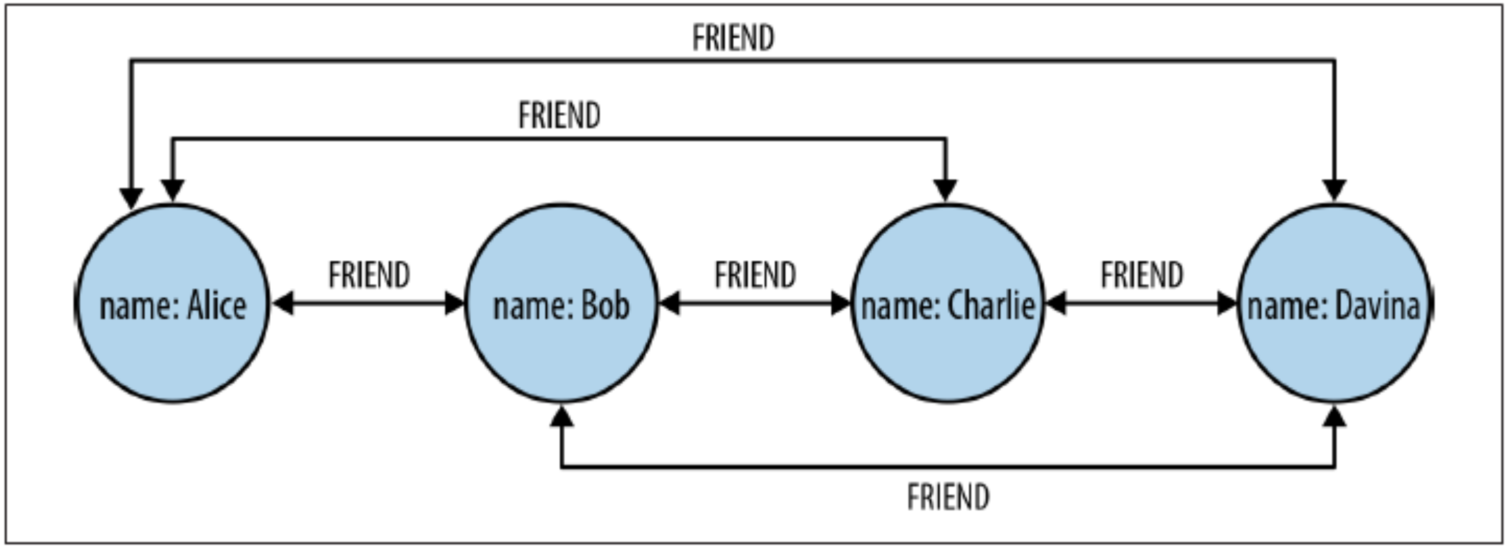


Figure 6-1. Nonnative graph processing engines use indexing to traverse between nodes

Use Relationships, not indexes, for fast traversal



Storage Structure Example

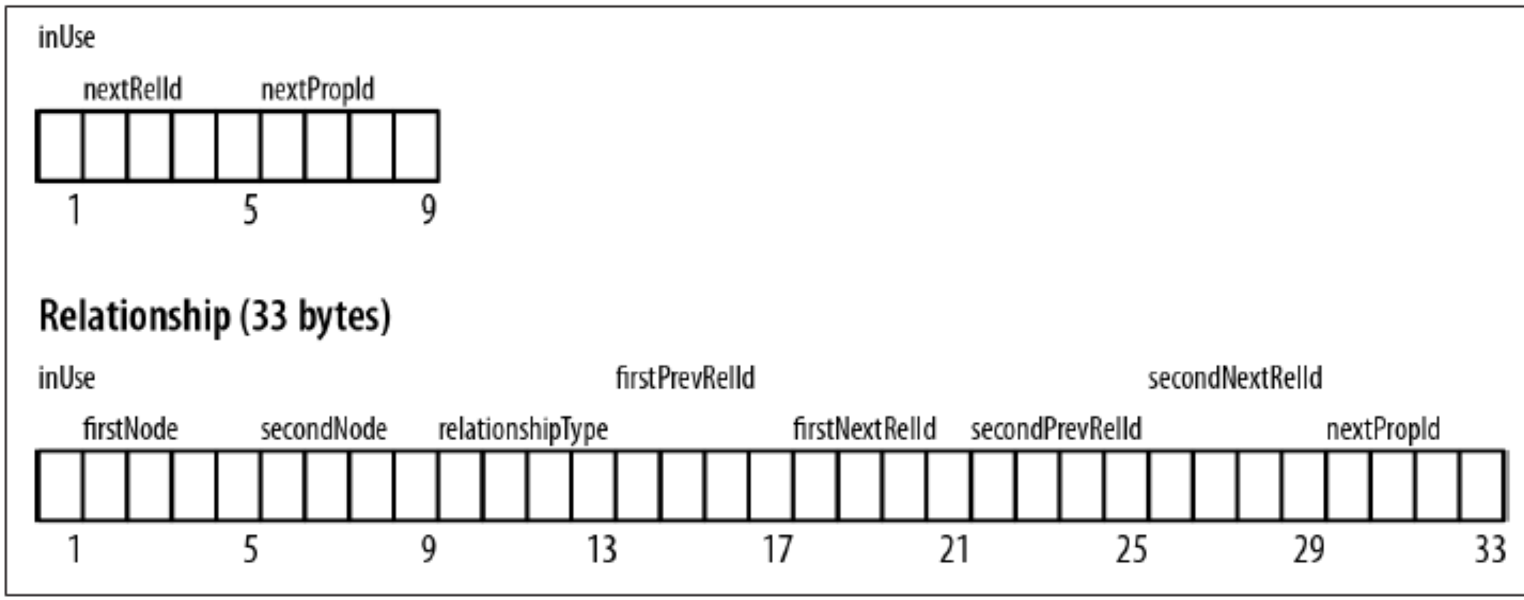


Figure 6-4. Neo4j node and relationship store file record structure

An experiment

Dataset: 12.2 million edges, 2.2 million vertices

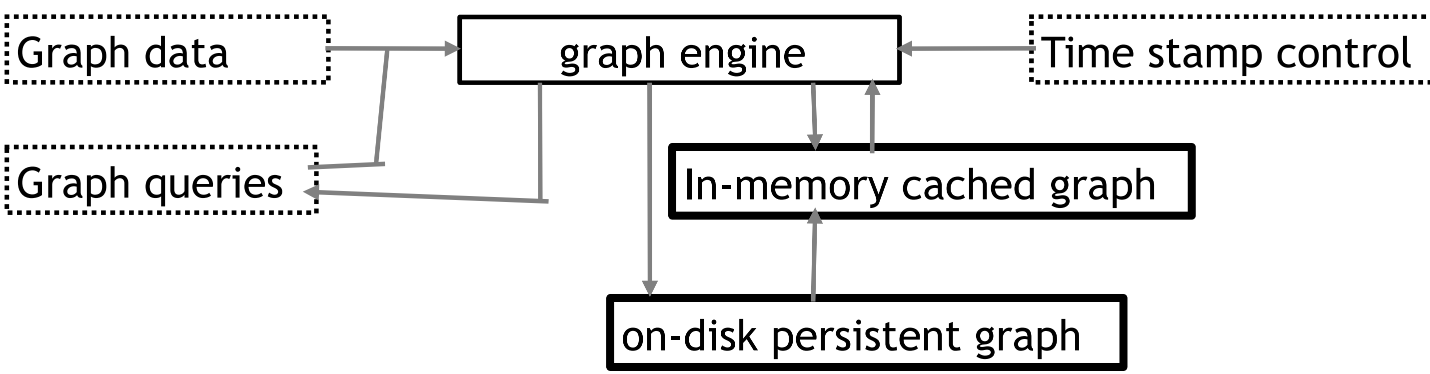
Goal: Find paths in a property graph. One of the vertex property is call TYPE. In this scenario, the user provides either a particular vertex, or a set of particular vertices of the same TYPE (say, "DRUG"). In addition, the user also provides another TYPE (say, "TARGET"). Then, we need find all the paths from the starting vertex to a vertex of TYPE “TARGET”. Therefore, we need to 1) find the paths using graph traversal; 2) keep trace of the paths, so that we can list them after the traversal. Even for the shortest paths, it can be multiple between two nodes, such as: drug->assay->target , drug->MOA->target

		Avg time (100 tests)	
	First test (cold-start)	Requested depth 5 traversal	Requested full depth traversal
NativeStore C++	39 sec	3.0 sec	4.2 sec
NativeStore JNI	57 sec	4.0 sec	6.2 sec
Neo4j (Blueprints 2.4)	105 sec	5.9 sec	8.3 sec
Titan (Berkeley DB)	3861 sec	641 sec	794 sec
Titan (HBase)	3046 sec	1597 sec	2682 sec

First full test - full depth 23. All data pulled from disk. Nothing initially cached.
Modes - All tests in default modes of each graph implementation. Titan can only be run in transactional mode. Other implementations do not default to transactional mode.

Native Store Overview

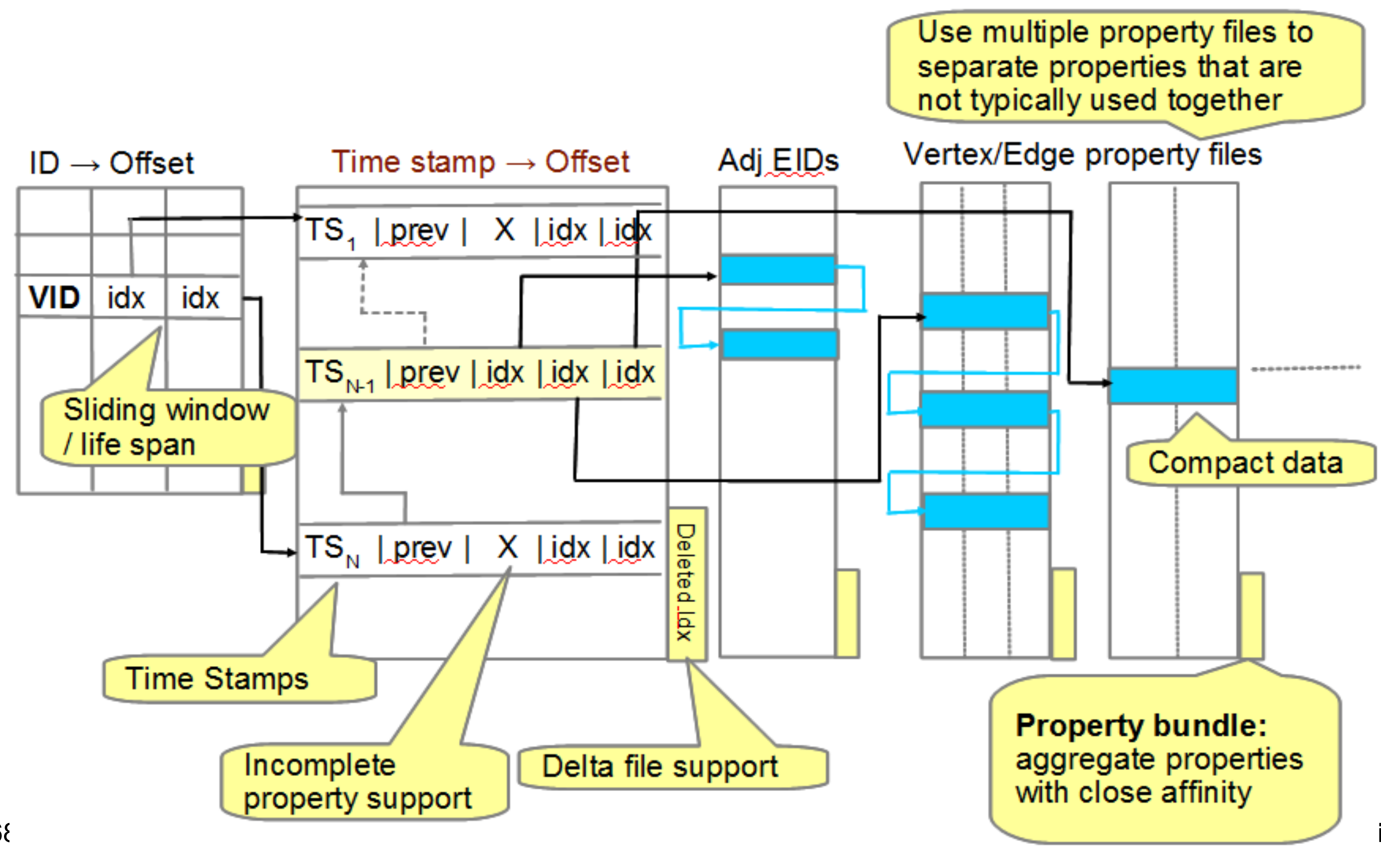
- **Native store represents graphs in-memory and on-disk**
  – Organizing graph data for representing a graph that stores both graph structure and vertex properties and edge properties
  – Caching graph data in memory in either batch-mode or on-demand from the on-disk streaming graph data
  – Accepting graph updates and modifying graph structure and/or property data accordingly and incorporating time stamps
    - Add edge, remove vertex, update property, etc.
  – Persisting graph updates along with the time stamps from in-memory graph to on-disk graph
  – Performing graph queries by loading graph structure and/or property data
    - Find neighbors of a vertex, retrieve property of an edge, traverse a graph, etc.



On-Disk Graph Organization

- **Native store organizes graph data for representing a graph with both structure and the vertex properties and edge properties using multiple files in Linux file system**
  - Creating a list called ID → Offset where each element translates a vertex (edge) ID into two offsets, pointing to the earliest and latest data of the vertex/edge, respectively
  - Creating a list called Time_stamp → Offset where each element has a time stamp, an offset to the previous time stamp of the vertex/edge, and a set of indices to the adjacent edge list and properties
  - Create a list of chained block list to store adjacent list and properties

On-disk persistent graph:

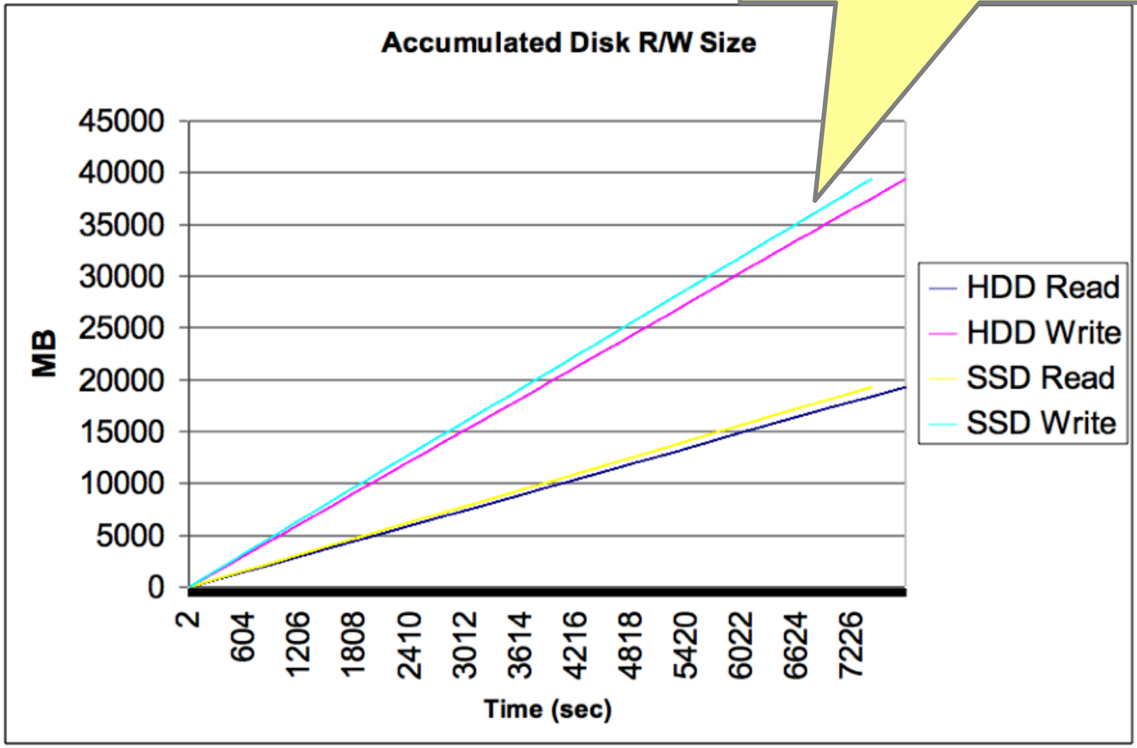


Impact from Storage Hardware

- Convert csv file (adams.csv 20G) to datastore
  - Similar performance: 7432 sec versus 7806 sec
  - CPU intensive
    - Average CPU util.: 97.4 versus 97.2
  - I/O pattern
    - Maximum read rate: 5.0 vs. 5.3
    - Maximum write rate: 97.7 vs. 85.3

Ratio HDD/SDD	TYPE1	TYPE2	TYPE3	TYPE4
	13.79	6.36	19.93	2.44

SSD offers consistently higher performance for both read and write



Queries

- Type 1: find the most recent URL and PCID of a user
- Type 2: find all the URLs and PCIDs
- Type 3: find all the most recent properties
- Type 4: find all the historic properti

Impact from Storage Hardware — 2

- Dataset: Knowledge Repository
  - 138614 Nodes, 1617898 Edges
- OS buffer is flushed before test
- Processing 320 queries in parallel
- In memory graph cache size: 4GB (default value)