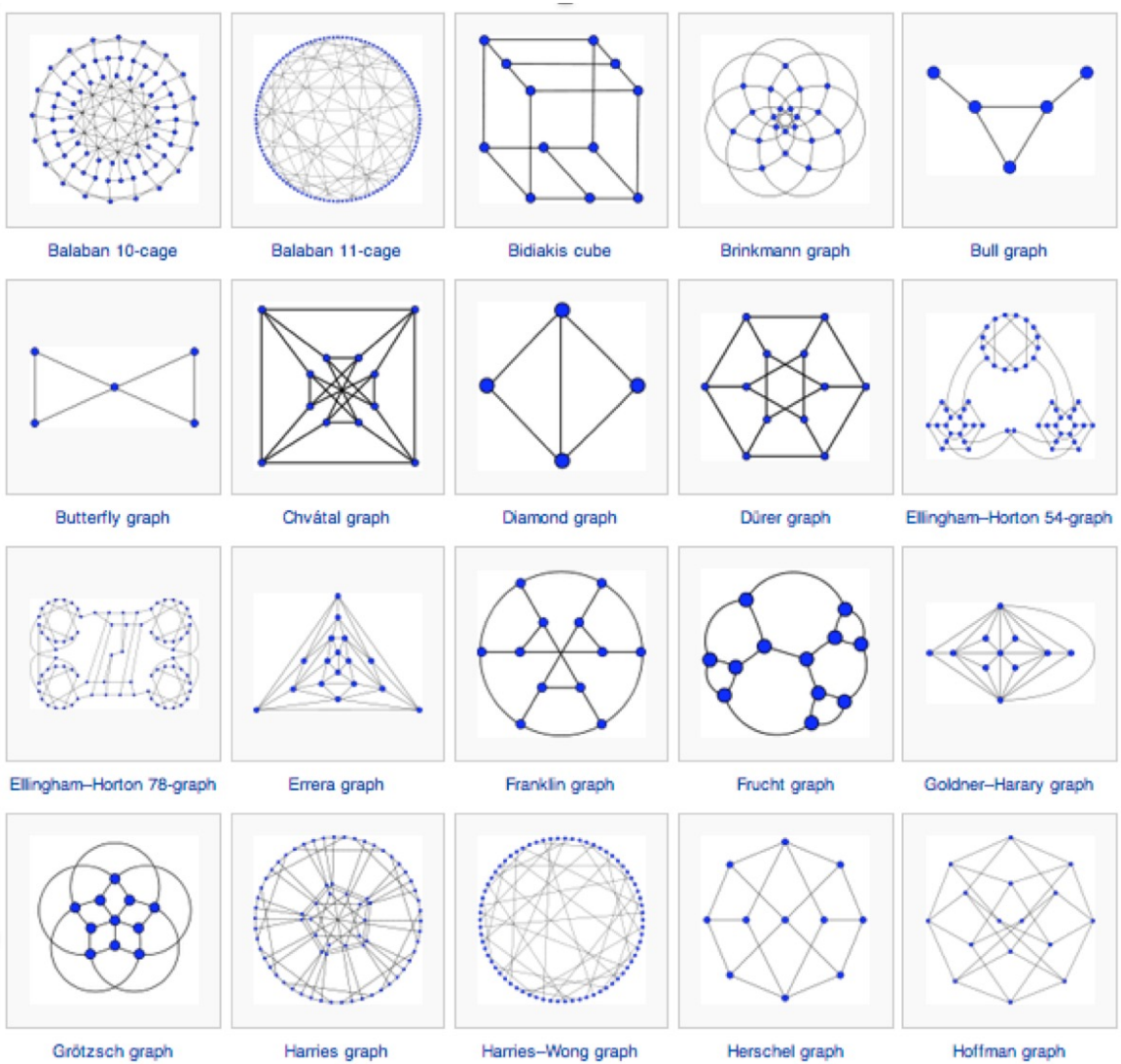
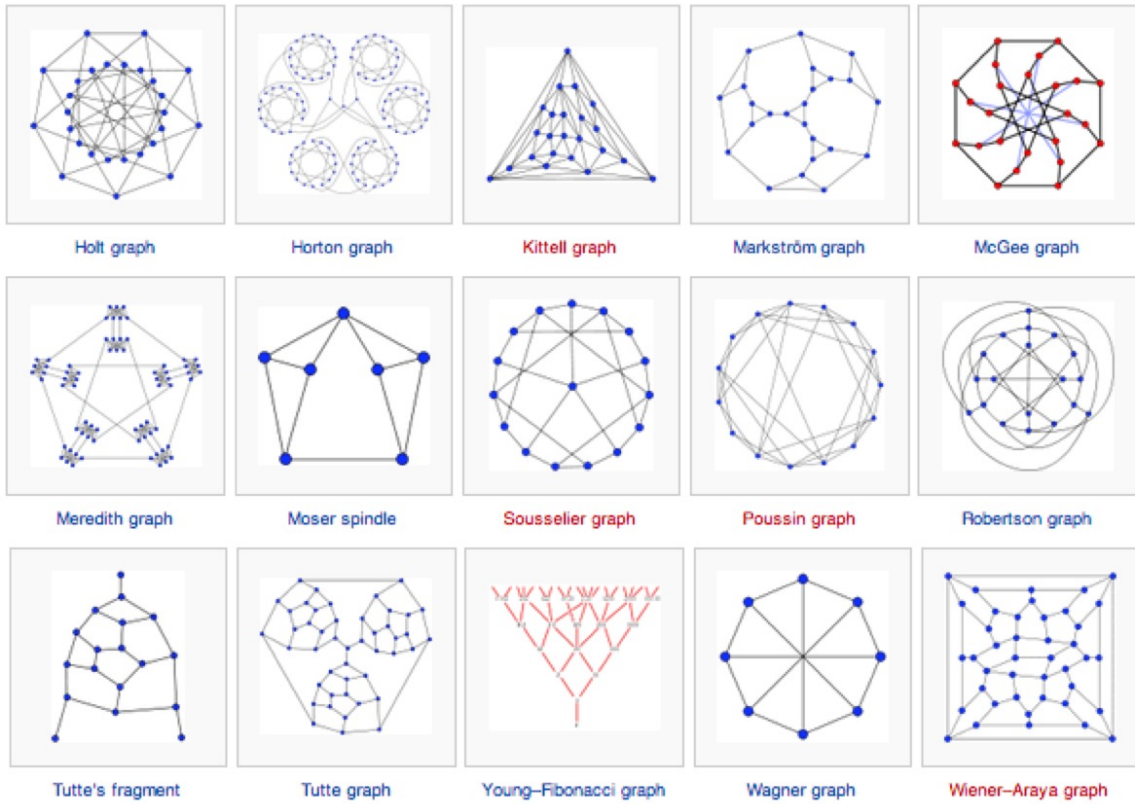
DS3000 / DS5110

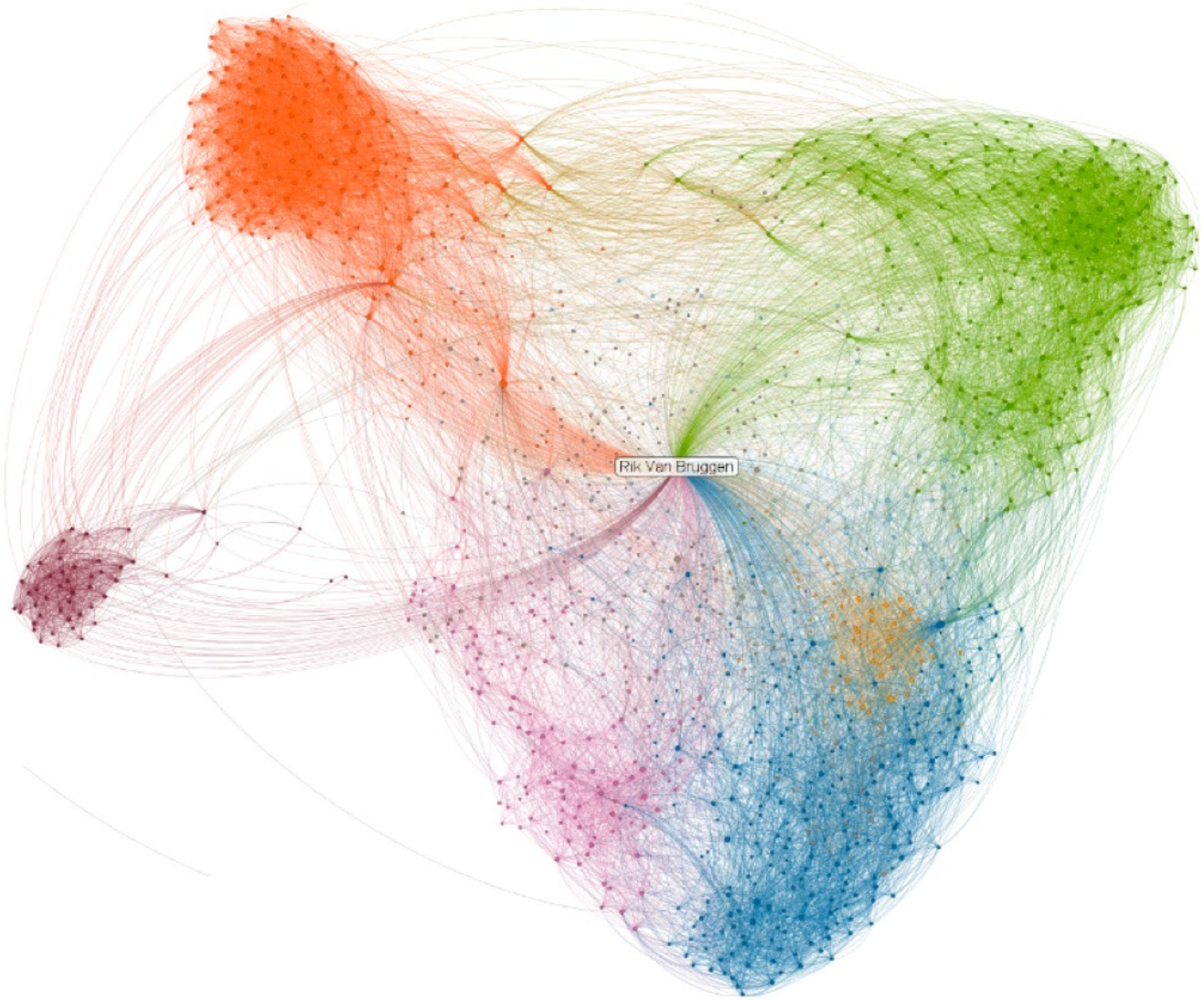
Graph Database Management

Graph Types

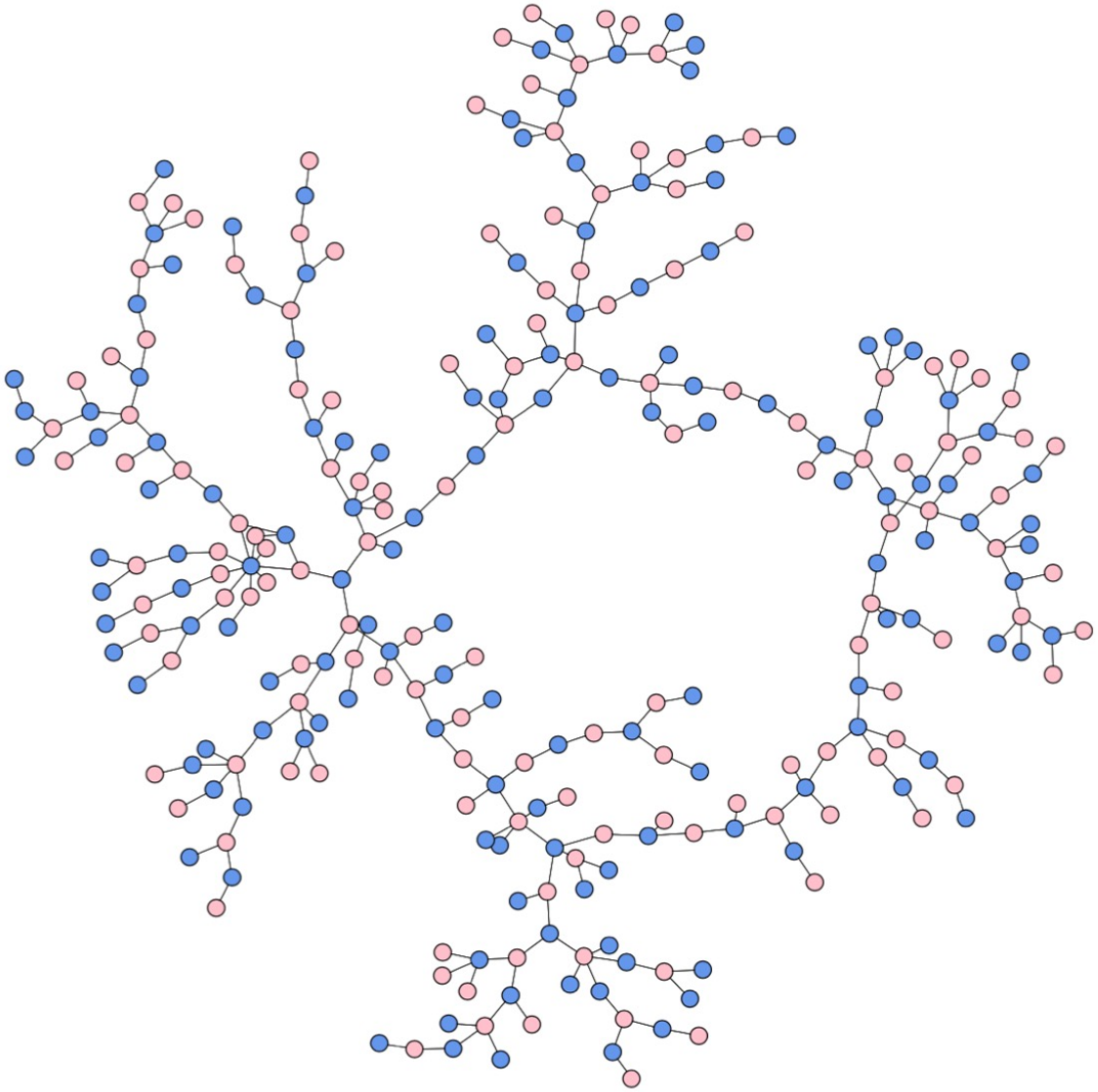

Balaban 10-cage
Balaban 11-cage
Bidiakis cube
Brinkmann graph
Bull graph
Butterfly graph
Chvátal graph
Diamond graph
Dürer graph
Ellingham–Horton 54-graph
Ellingham–Horton 78-graph
Errera graph
Franklin graph
Frucht graph
Goldner–Harary graph
Grötzsch graph
Harries graph
Harries–Wong graph
Herschel graph
Hoffman graph


Holt graph
Horton graph
Kittell graph
Markström graph
McGee graph
Meredith graph
Moser spindle
Sousselier graph
Poussin graph
Robertson graph
Tutte's fragment
Tutte graph
Young–Fibonacci graph
Wagner graph
Wiener–Araya graph

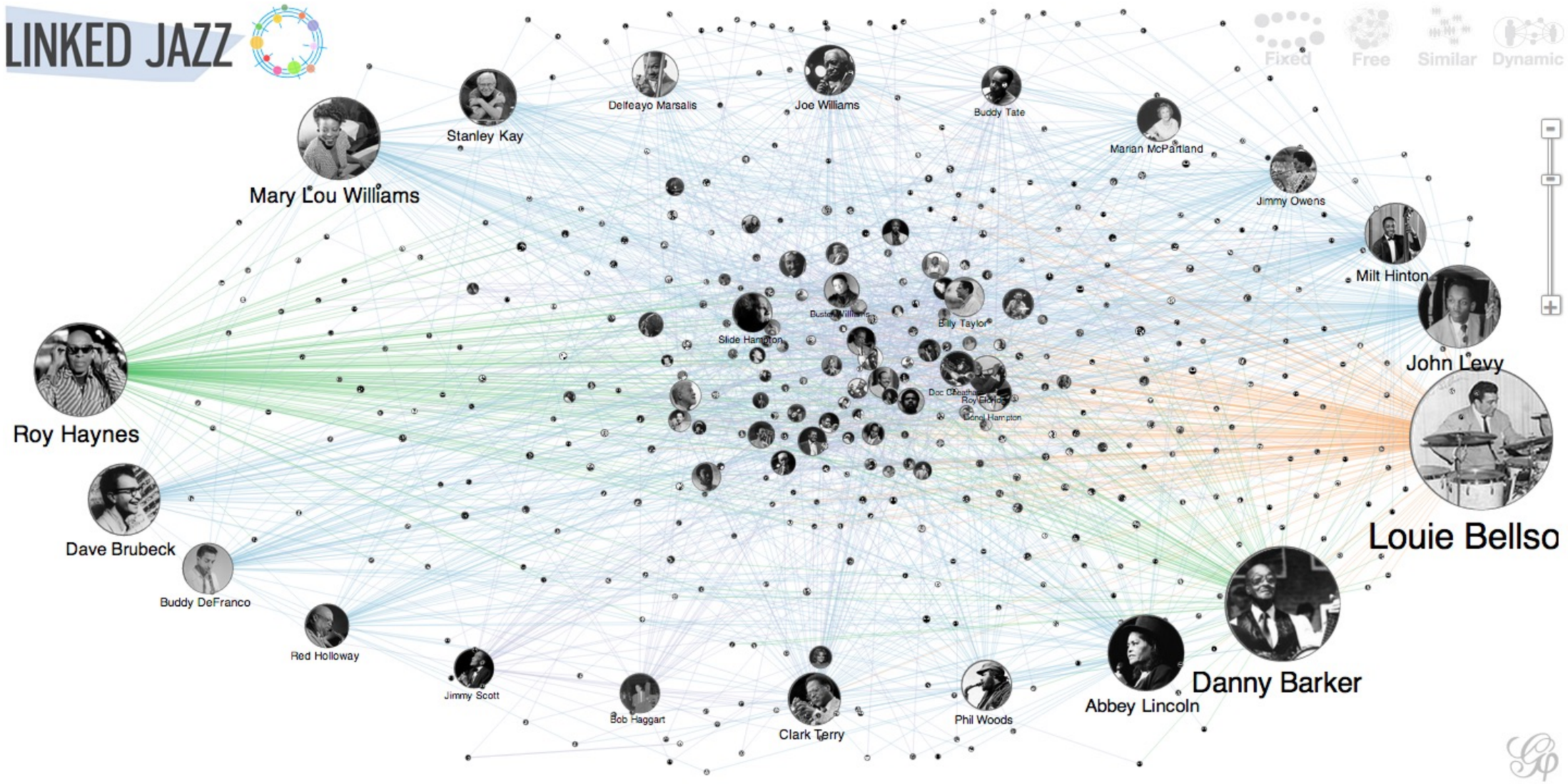
Graphs are Ubiquitous…in society



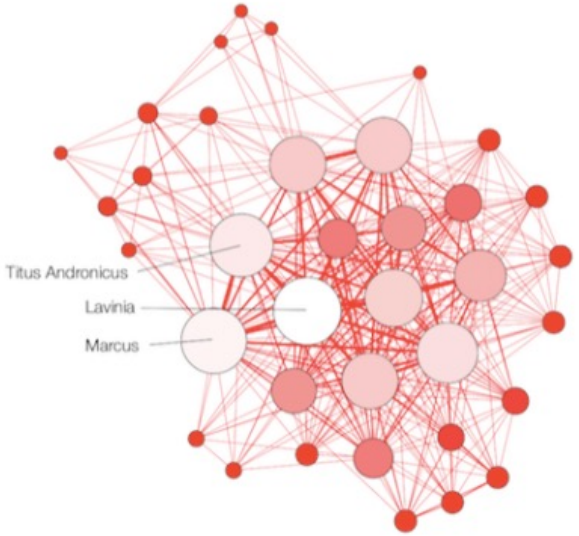
Dating Networks in High School



Graphs are Ubiquitous…in music



Graphs are Ubiquitous…in literature



TITUS ANDRONICUS

Number of characters 36 | 50% Network density

ROMEO AND JULIET

Number of characters 41 | 37% Network density

JULIUS CAESAR

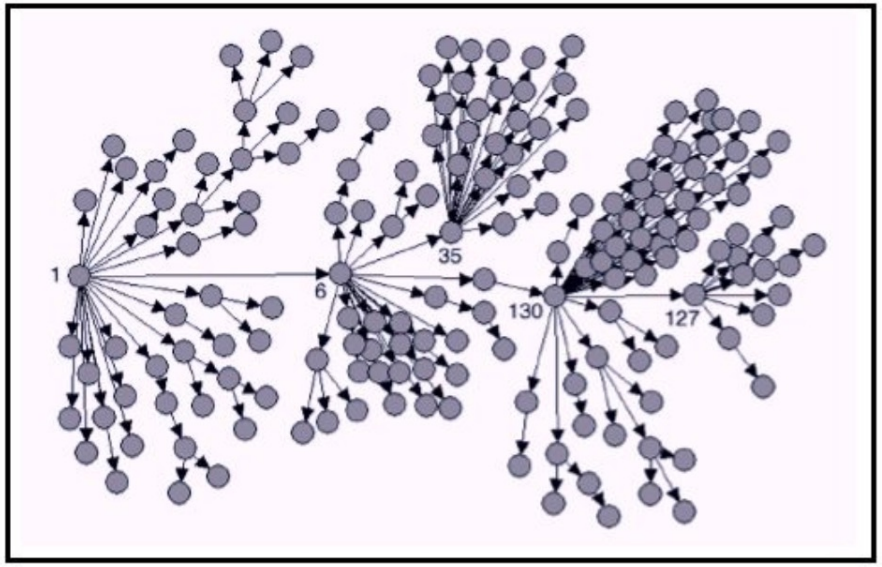
Number of characters 46 | 34% Network density

From: www.martingrandjean.ch/network-visualization-shakespeare

Graphs are Ubiquitous…in medicine

Is there a small subset of SARS patients who account for a disproportionate share of transmission? **MMWR May 9, 2003 / Vol. 52 / No. 18 - I**

FIGURE 2. Probable cases of severe acute respiratory syndrome, by reported source of infection* — Singapore, February 25–April 30, 2003



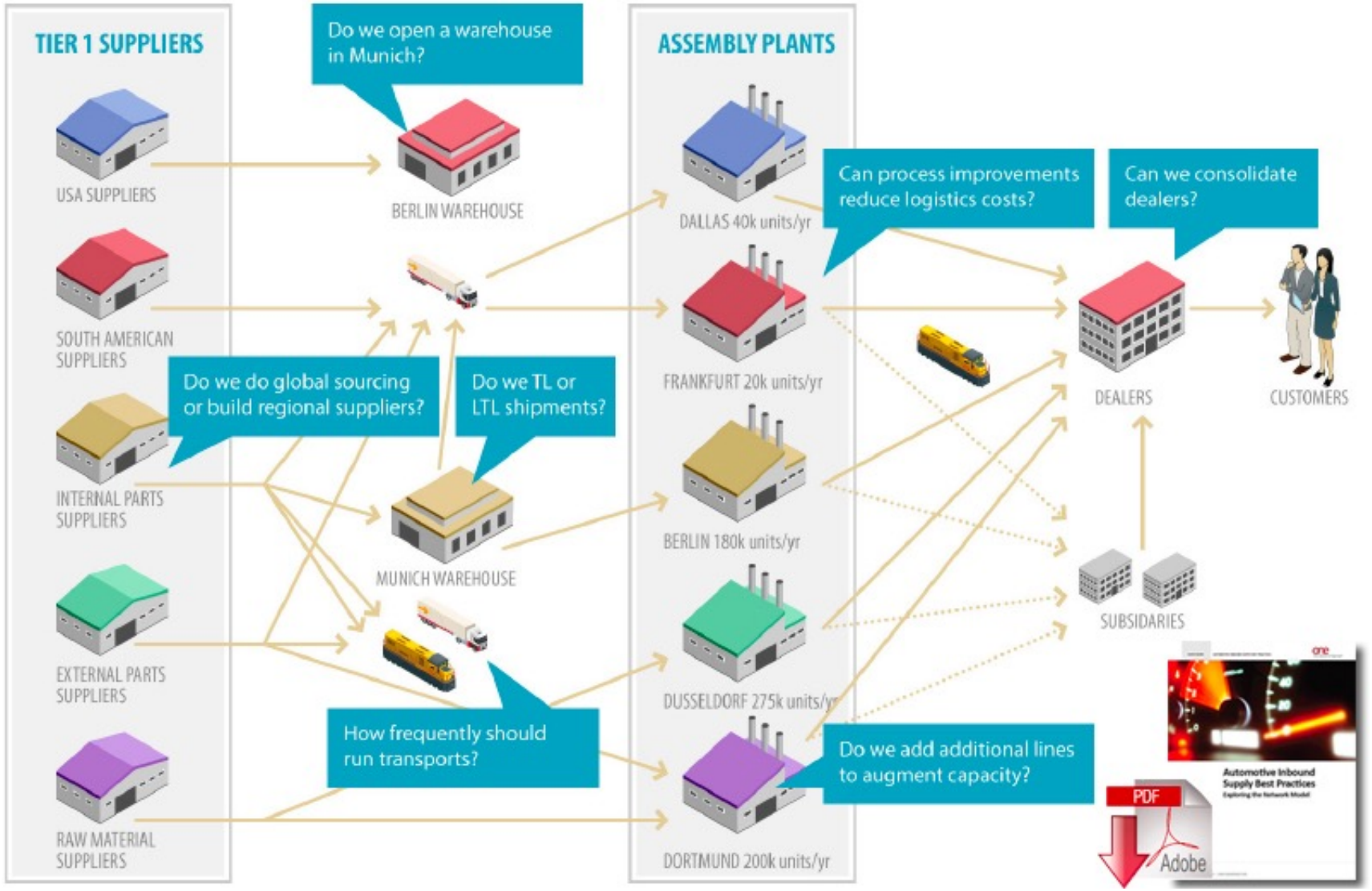
* Patient 1 represents Case 1; Patient 6, Case 2; Patient 35, Case 3; Patient 130, Case 4; and Patient 127, Case 5. Excludes 22 cases with either no or poorly defined direct contacts or who were cases translocated to Singapore and the seven contacts of one of these cases.

Reference: Bogatti SP. Netdraw 1.0 Network Visualization Software. Harvard, Massachusetts: Analytic Technologies, 2002.

Patients No: 1, 6, 35, 130&127 seemed to be **"hypertransmitters"**

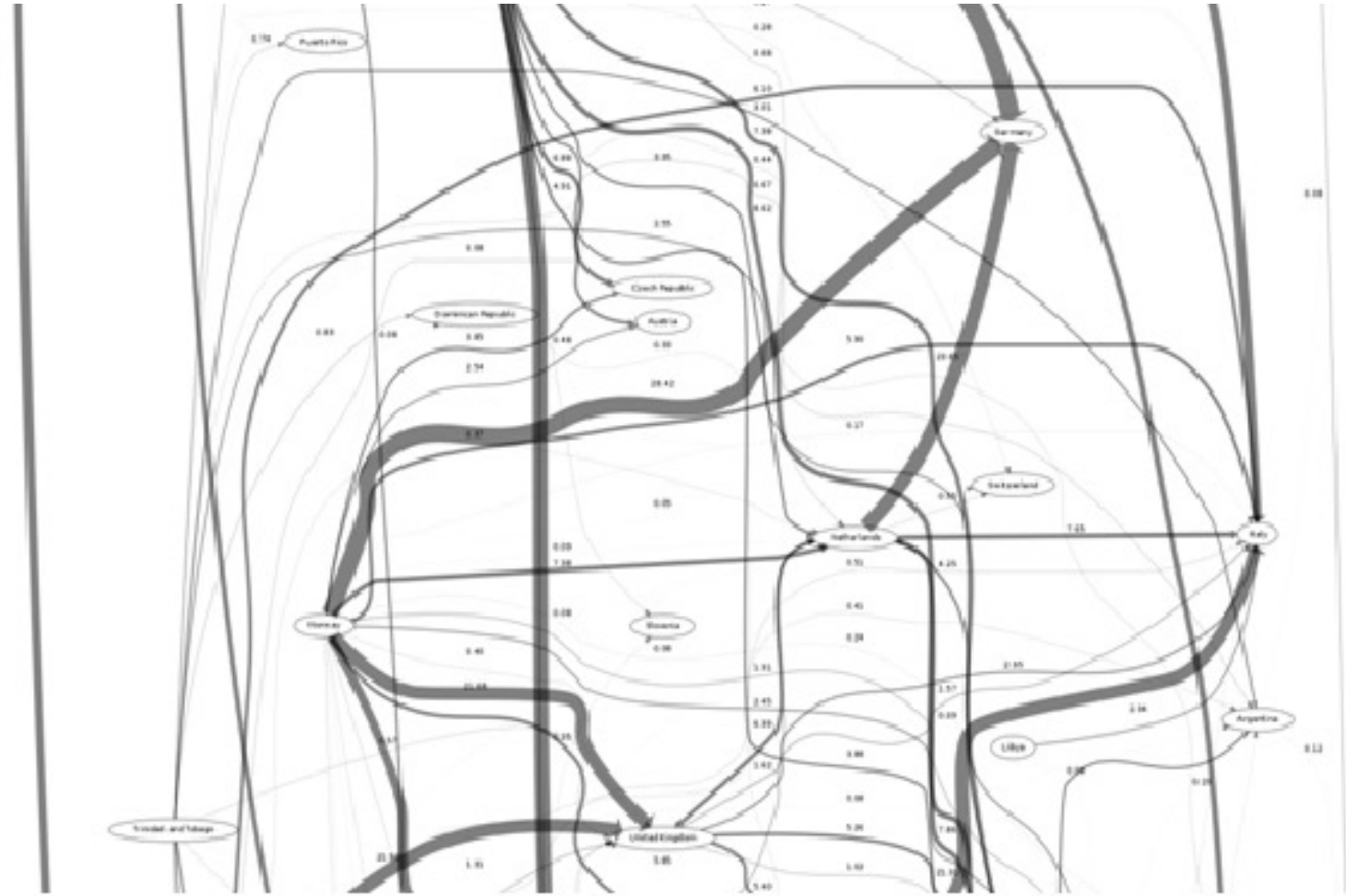
SOUTH CAROLINA AHEC

Graphs are Ubiquitous…in Supply Chain Optimization

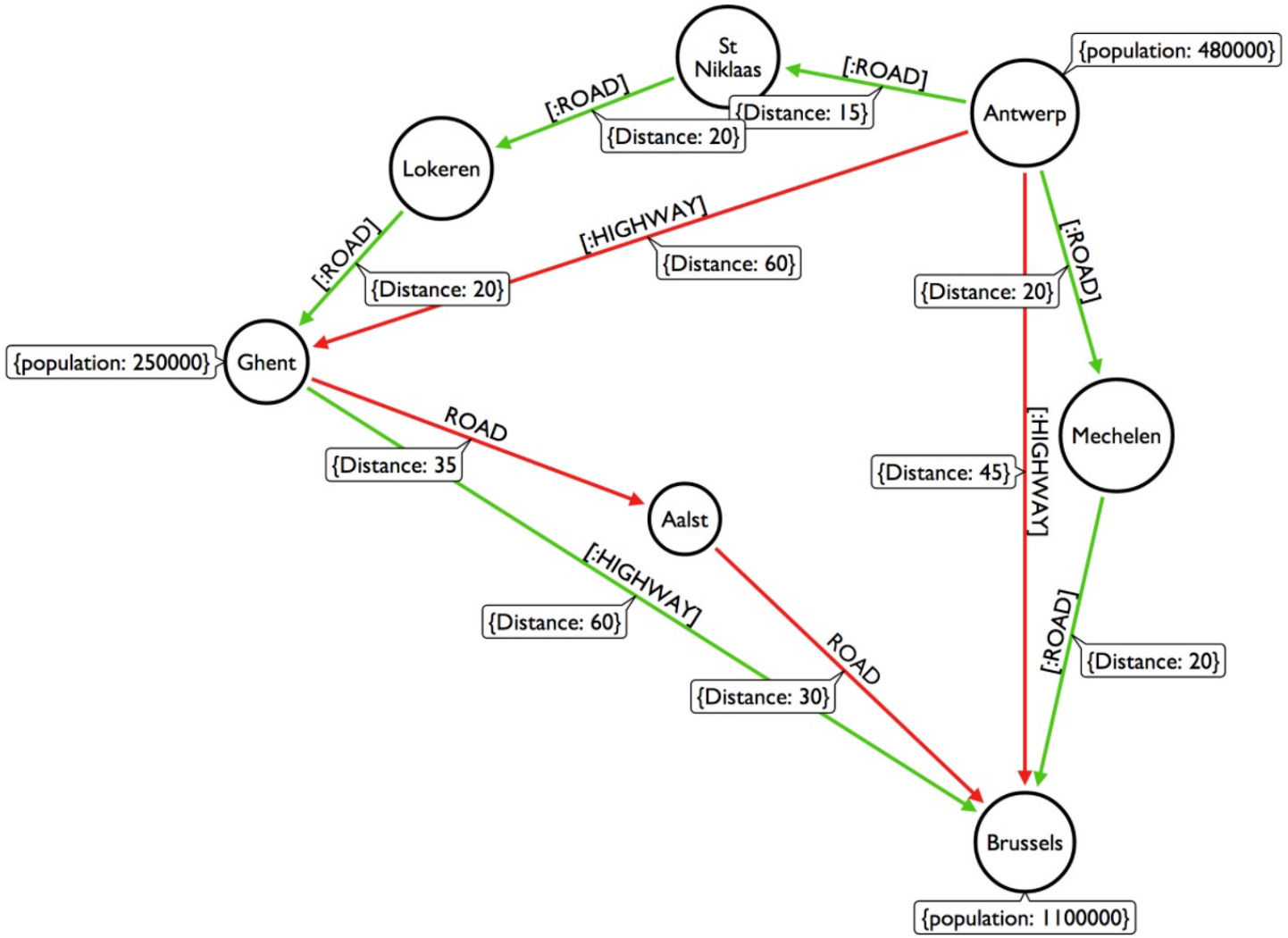


…where thinking about the flow of goods is essential.

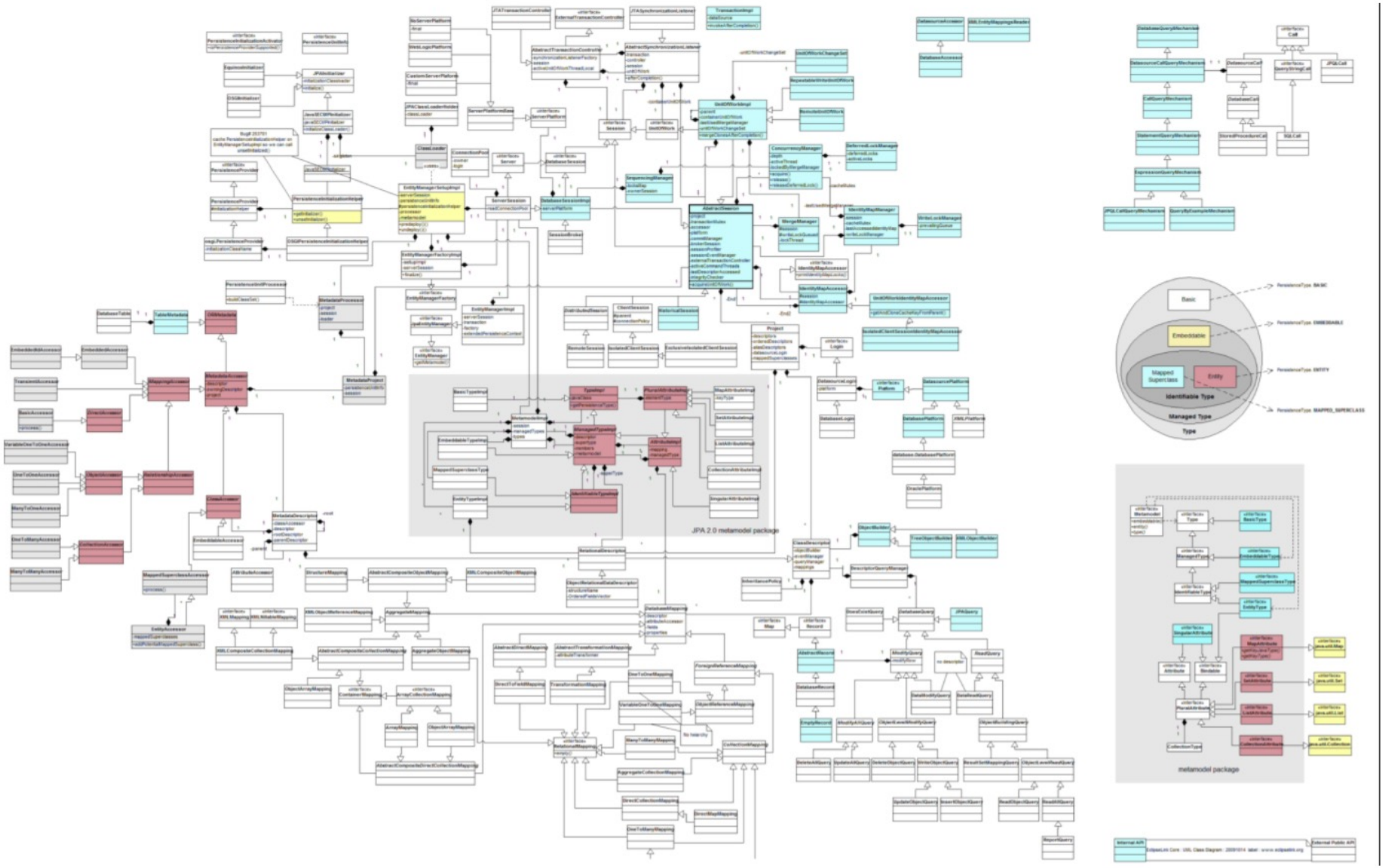
- Logistics
- Electrical grids
- Air travel
- Gas networks
- Etc.



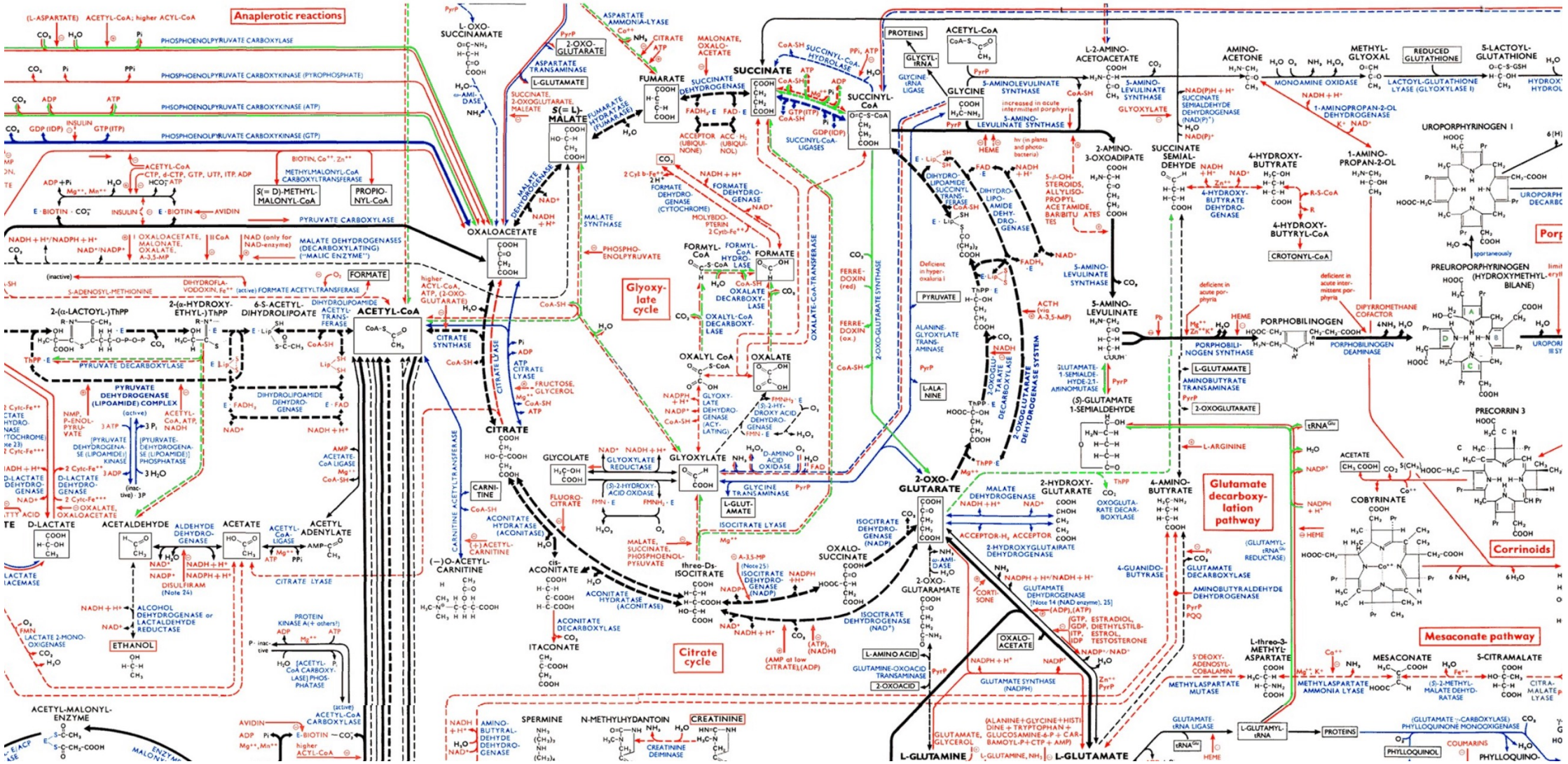
Graphs are Ubiquitous…in Navigation and Routing



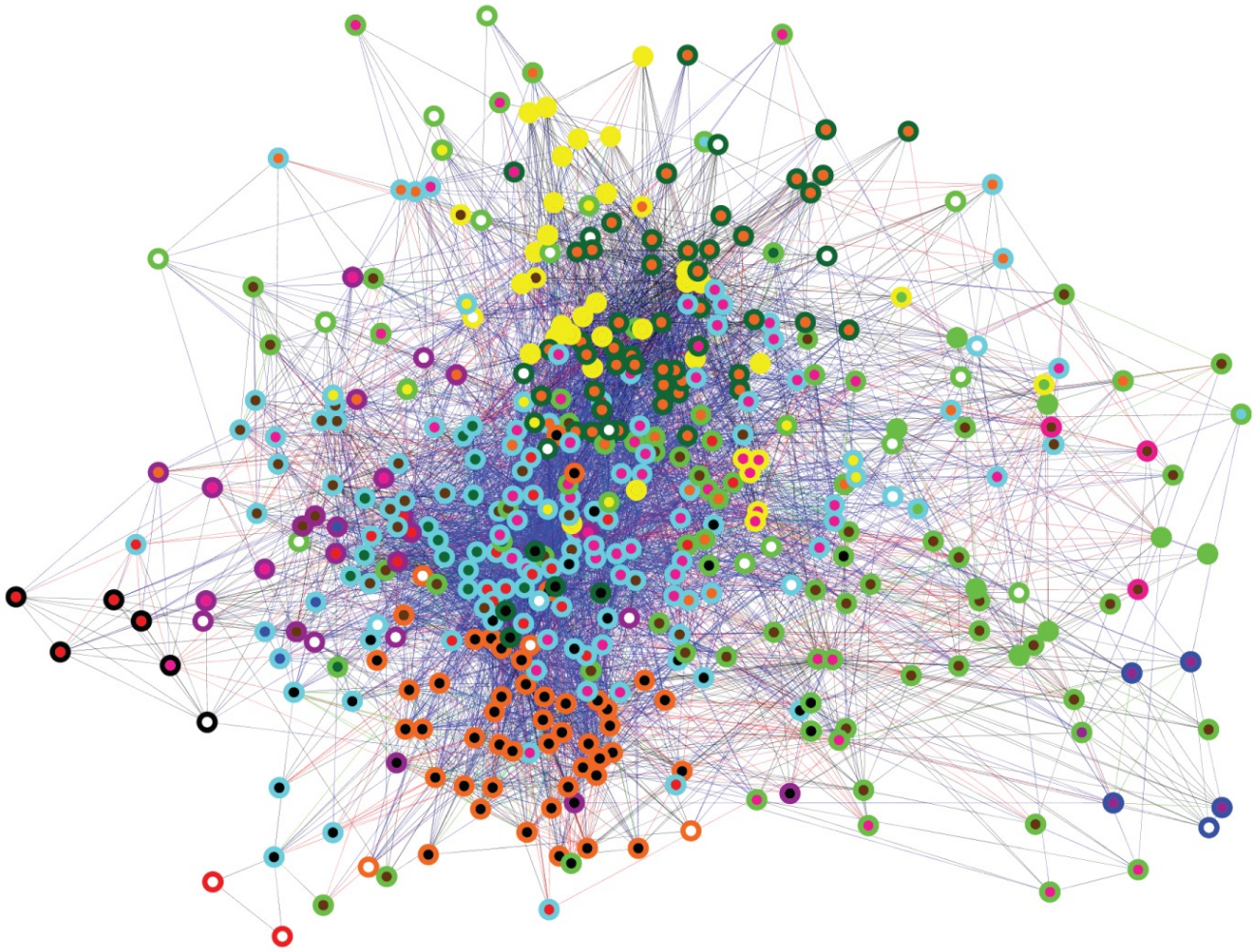
Graphs are Ubiquitous…in software design



Graphs are Ubiquitous…In Biology



Graphs are Ubiquitous…In Biology

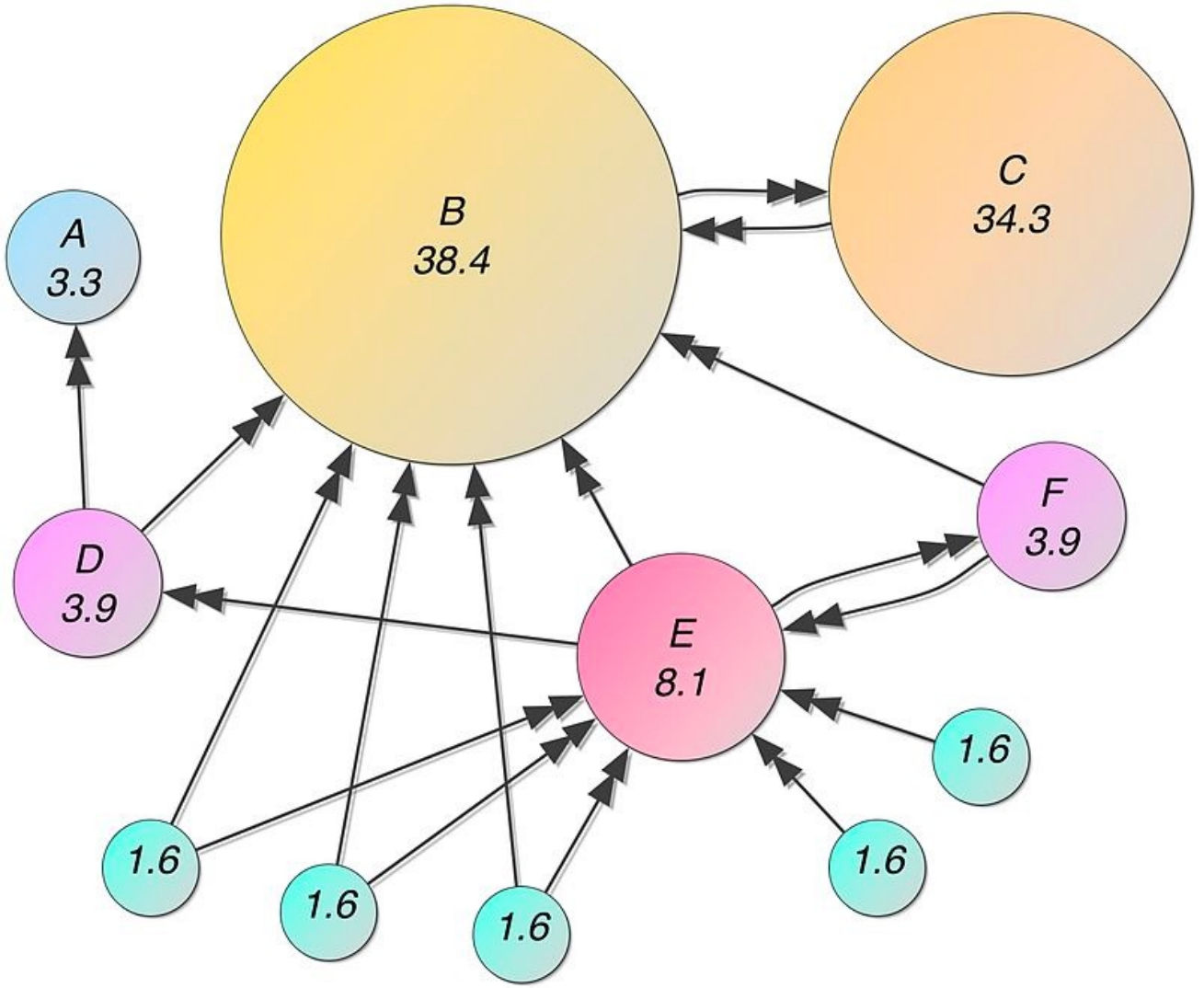


	Pure	Complete
Parasitic diseases	0.750	0.750
Nervous system neoplasms	1.000	0.833
Muscle and ligament disorders	0.556	1.000
Conduct disorder	0.667	1.000
Communication impairment	0.593	1.000
Sensation disorders	0.593	1.000
Malignant neoplasms	0.956	0.717
Neoplasms	0.733	0.155
Endocrine gland neoplasms	0.267	0.667
Dis. of hematopoietic system	0.667	0.375
Heart diseases	1.000	0.778
Disease of metabolism	0.455	0.128
"Brain diseases, metabolic"	0.182	1.000
Disease of metabolism	0.489	0.590
Endocrine pancreas disorders	0.149	1.000
Cardiovascular disease	0.652	0.366
Vascular diseases	0.478	0.611
Bacterial infections	0.600	0.429
Behavioral disease	0.500	0.129
Mental retardation	0.375	1.000
Psychotic disorders	0.125	0.111
Digestive system disorders	0.412	0.160
Endocrine system disease	0.353	0.150
Infectious diseases	0.875	0.412
Congenital abnormality	0.300	0.600
Head disease	0.275	0.917
Digestive system disorders	0.275	0.250

From: Davis and Chawla (2011) Exploring and Exploting Interactions from Multi-Relational Gene and Phenotypic Networks

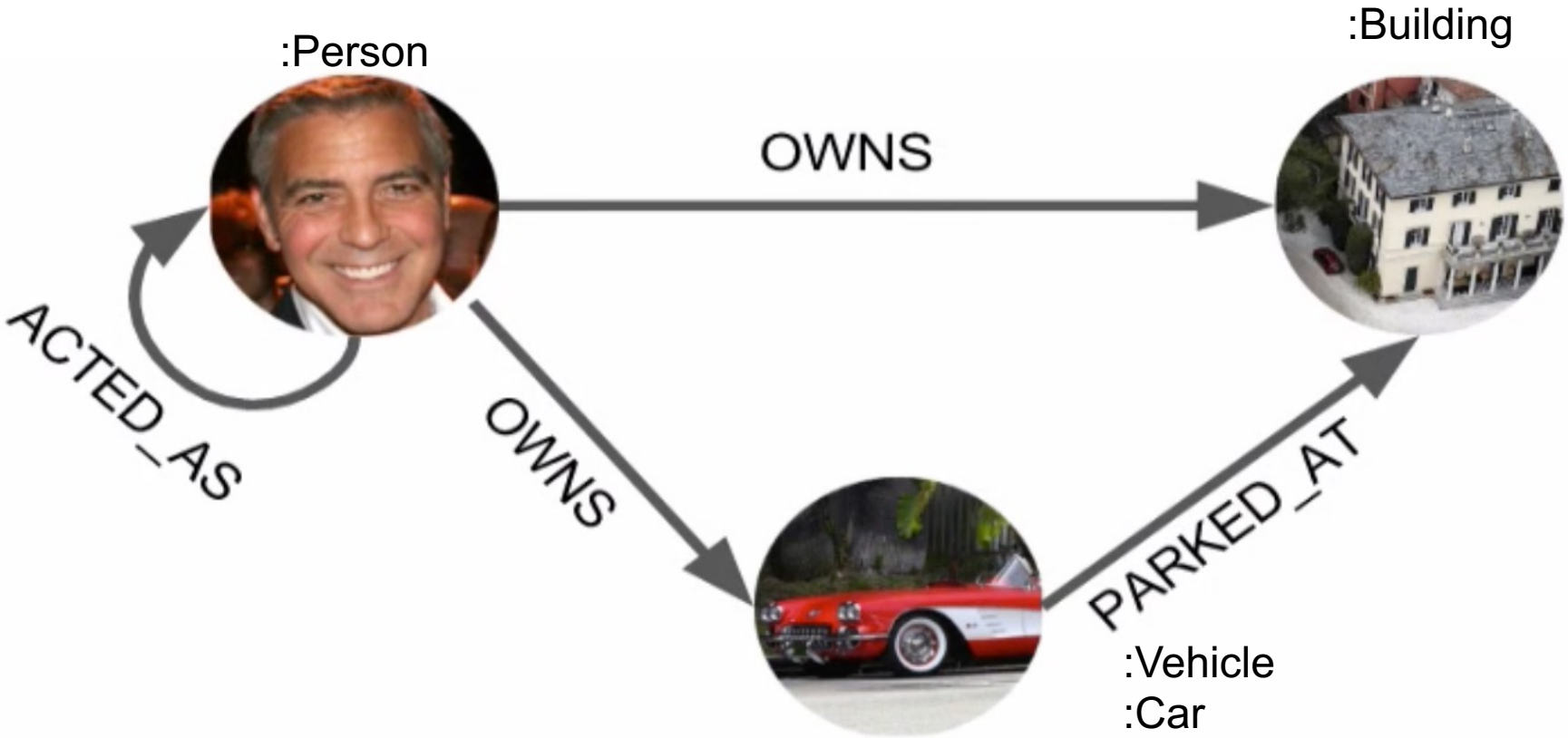
Page Rank

The more websites point to you the more important you are.



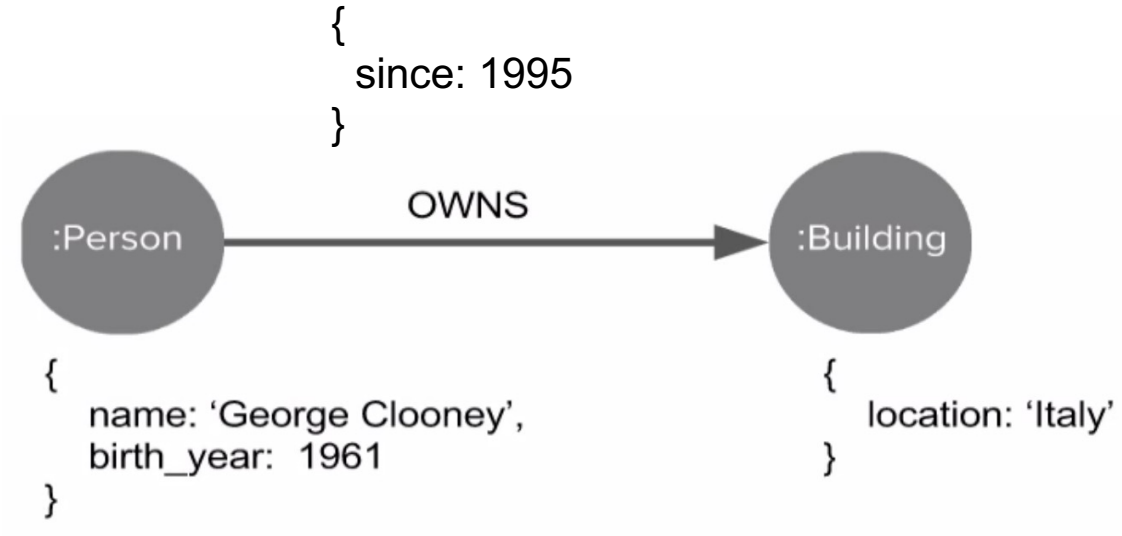
What is a Graph Database?

- A database optimized for storing *nodes* and *relationships.*
- Nodes have one or more categorical labels and represent things
- Relationships have a single type and are a connected arc between two nodes
- Nodes and relationships can have properties



Graph properties

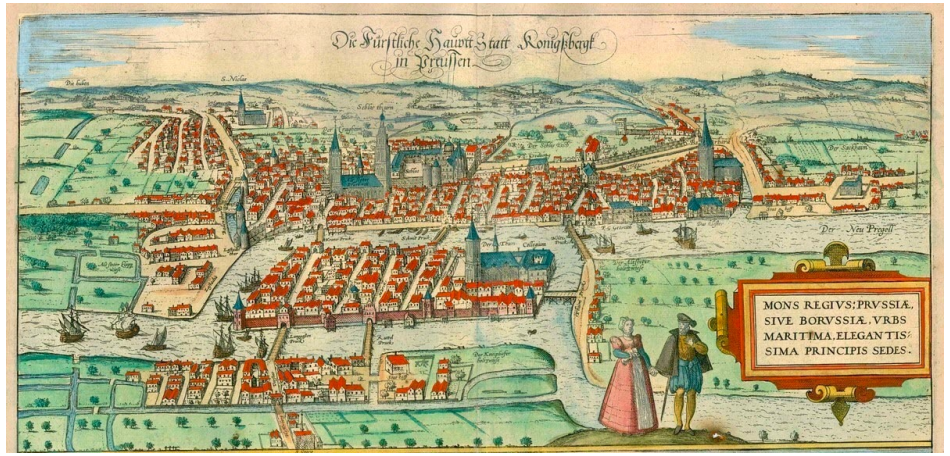
- Properties are key value pairs that are assigned to a node or a relationship
- Nodes have one or more categorical labels and represent things
- Relationships have a single type and are a connected arc between two nodes

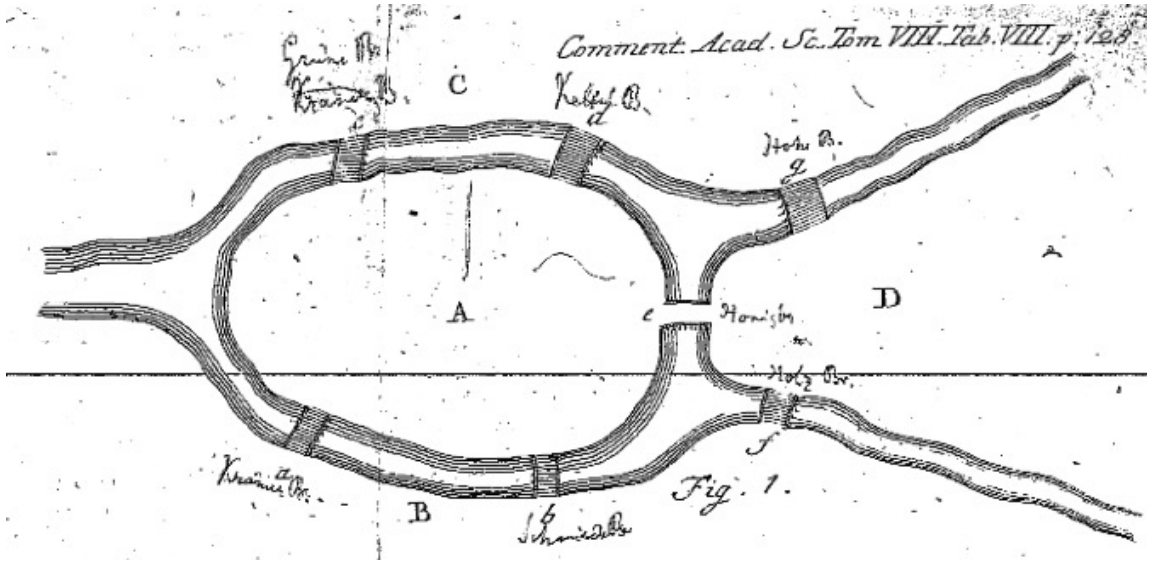


Value Types

Type	Example
Boolean	true, false
Text	"AKA string"
Numbers	123, 56.70
Lists	['must', 'be', 'same', 'type']

Great names in Graph Theory

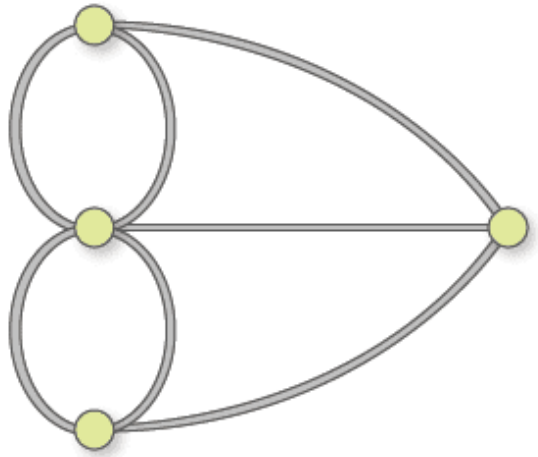




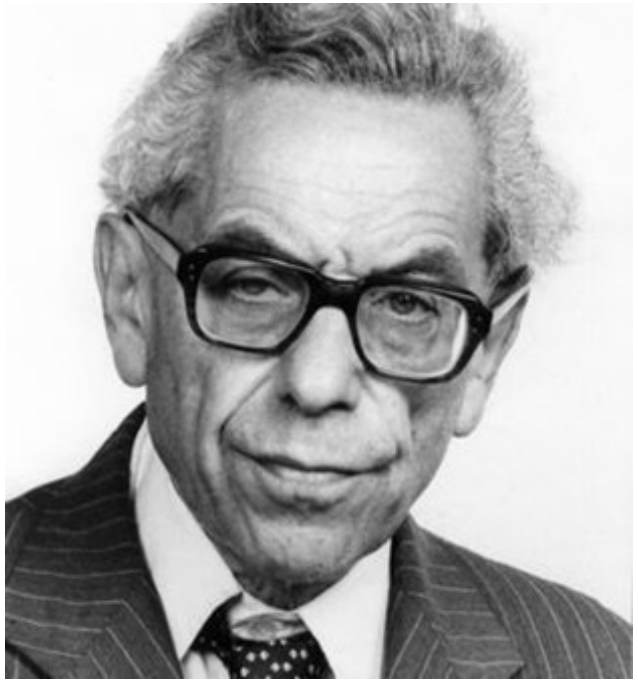
Leonhard Euler
1707 - 1783

An **Euler path** is a path that uses every edge of a graph exactly once.

A graph **G** has an **Euler path** if and only if there are exactly two vertices with odd degree.

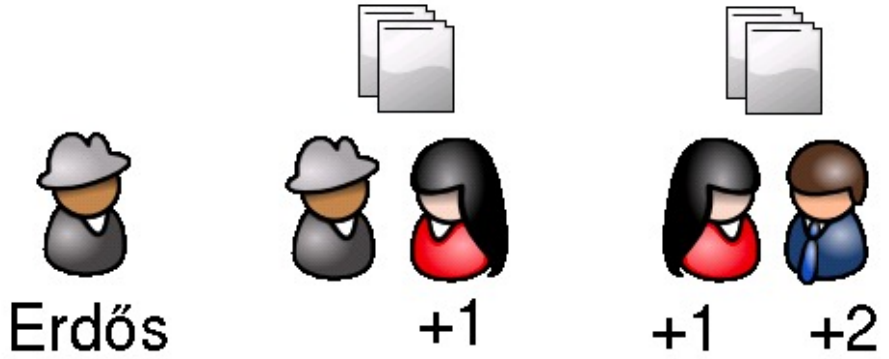


Great names in Graph Theory



Paul Erdős
1913 - 1996

Erdős Numbers



Albert Einstein: 2
Richard Feynman: 3
Stephen Hawking: 4

JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 13, Number 10, 2006

Pp. 1659–1672

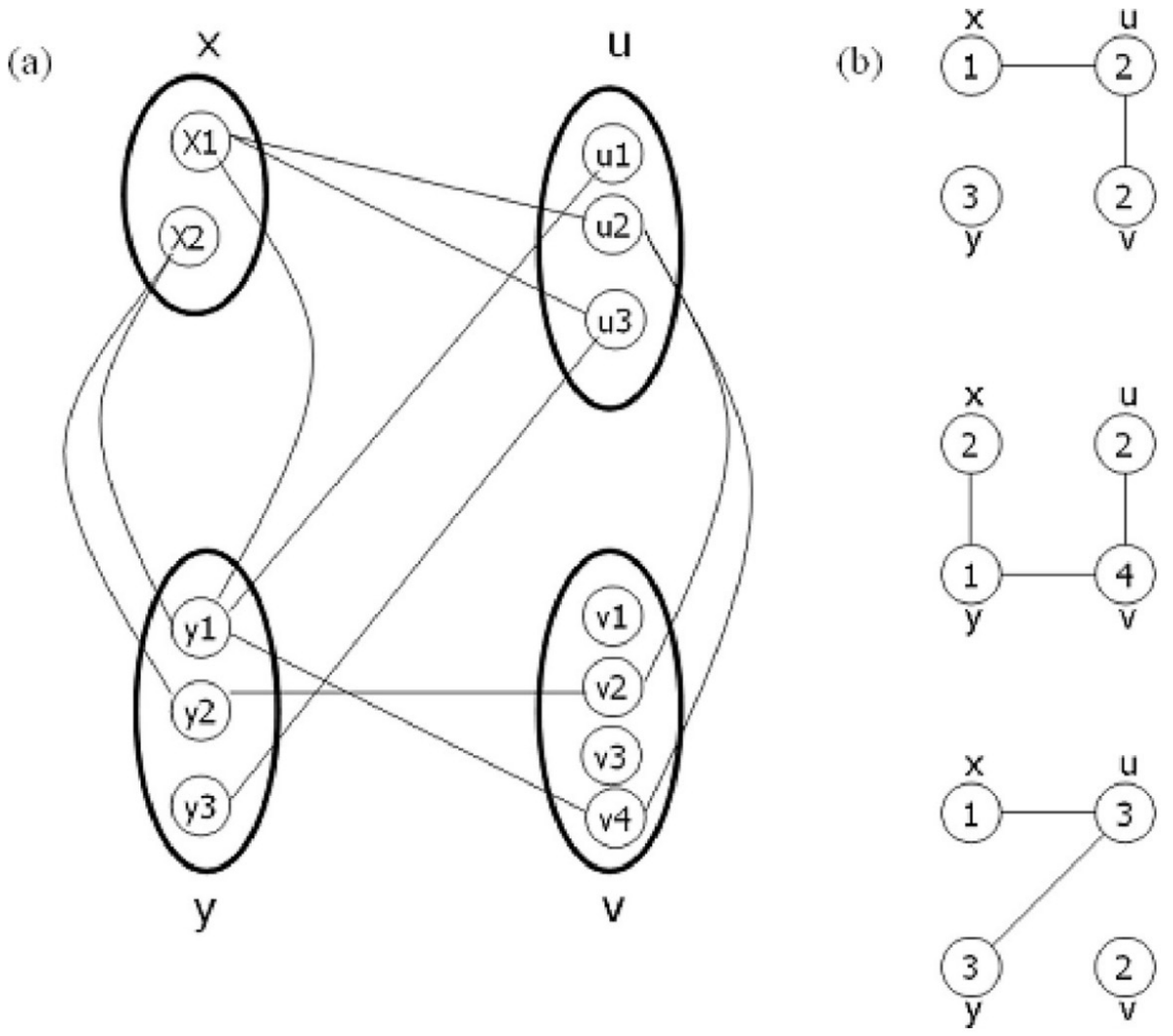
Multi-Node Graphs: A Framework for Multiplexed Biological Assays

NOGA ALON,[1,2] VERA ASODI,[2] CHARLES CANTOR,[3] SIMON KASIF,[3,4] and JOHN RACHLIN[5]

ABSTRACT

Multiplex polymerase chain reaction (PCR) is an extension of the standard PCR protocol in which primers for multiple DNA loci are pooled together within a single reaction tube, enabling simultaneous sequence amplification, thus reducing costs and saving time. Potential cost saving and throughput improvements directly depend on the level of multiplexing achieved. Designing reliable and highly multiplexed assays is challenging because primers that are pooled together in a single reaction tube may cross-hybridize, though this can be addressed either by modifying the choice of primers for one or more amplicons, or by altering the way in which DNA loci are partitioned into separate reaction tubes. In this paper, we introduce a new graph formalism called a *multi-node graph*, and describe its application to the analysis of multiplex PCR scalability. We show, using random multi-node graphs that the scalability of multiplex PCR is constrained by a phase transition, suggesting fundamental limits on efforts to improve the cost-effectiveness and throughput of standard multiplex PCR assays. In particular, we show that when the multiplexing level of the reaction tubes is roughly $\Theta(\log(sn))$ (where s is the number of primer pair candidates per locus and n is the number of loci to be amplified), then with very high probability we can 'cover' all loci with a valid assignment to one of the tubes in the assay. However, when the multiplexing level of the tube exceeds these bounds, there is no possible cover and moreover the size of the cover drops dramatically. Simulations using a simple greedy algorithm on real DNA data also confirm the presence of this phase transition. Our theoretical results suggest, however, that the resulting phase transition is a fundamental characteristic of the problem, implying intrinsic limits on the development of future assay design algorithms.

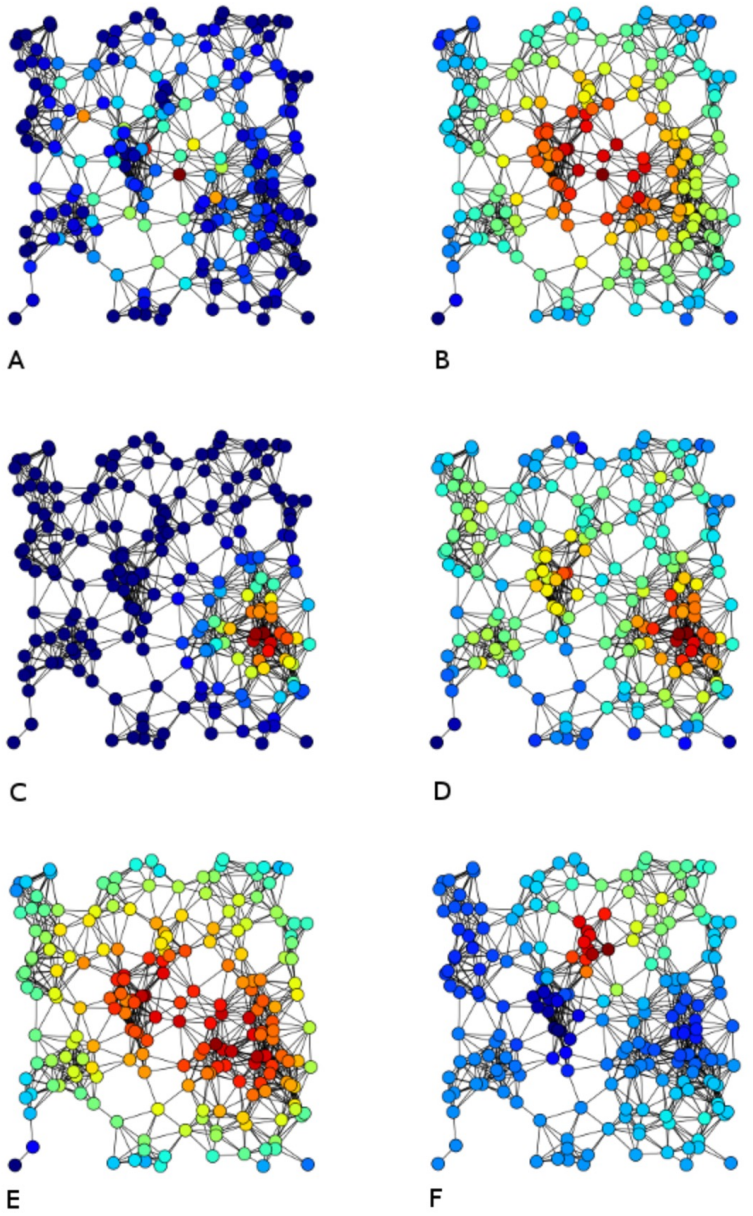
Key words: algorithms, combinatorics, computational molecular biology.



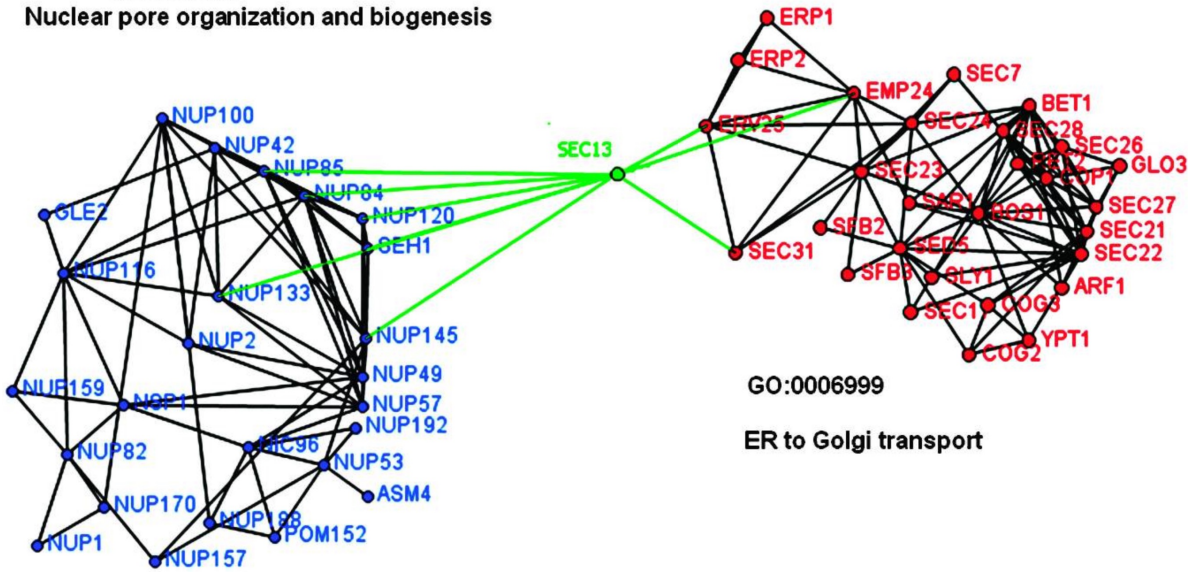
Centrality

In graph theory and network analysis, indicators of **centrality** identify the most important vertices within a graph. Applications include identifying the most influential person(s) in a social network, key infrastructure nodes in the Internet or urban networks, and super-spreaders of disease.

-- Wikipedia



Genes can play different roles: “Interactively promiscuous”



Molecular Systems Biology (2006) doi:10.1038/msb4100103

www.molecularsystemsbiology.com
Article number: 66

molecular systems biology

Biological context networks: a mosaic view of the interactome

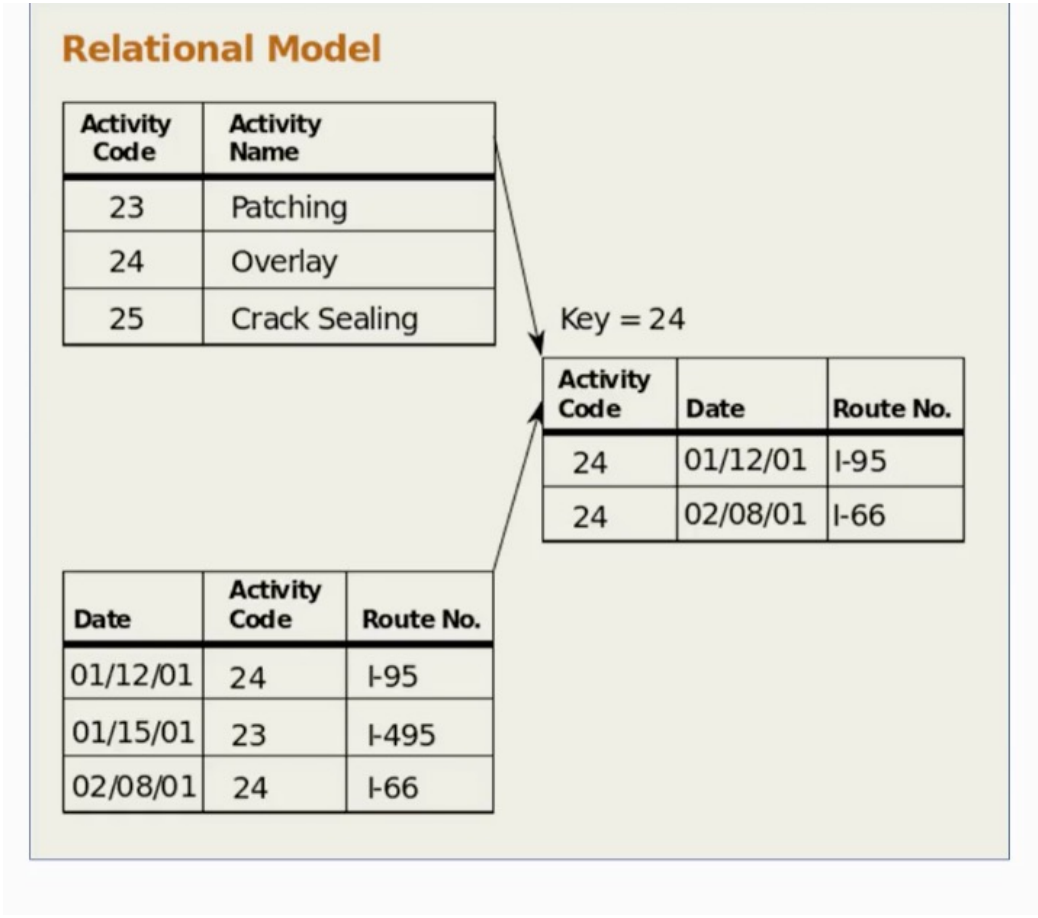
John Rachlin[1,2,*], Dikla Dotan Cohen[2], Charles Cantor[3,4,5] and Simon Kasif[2,3,6]

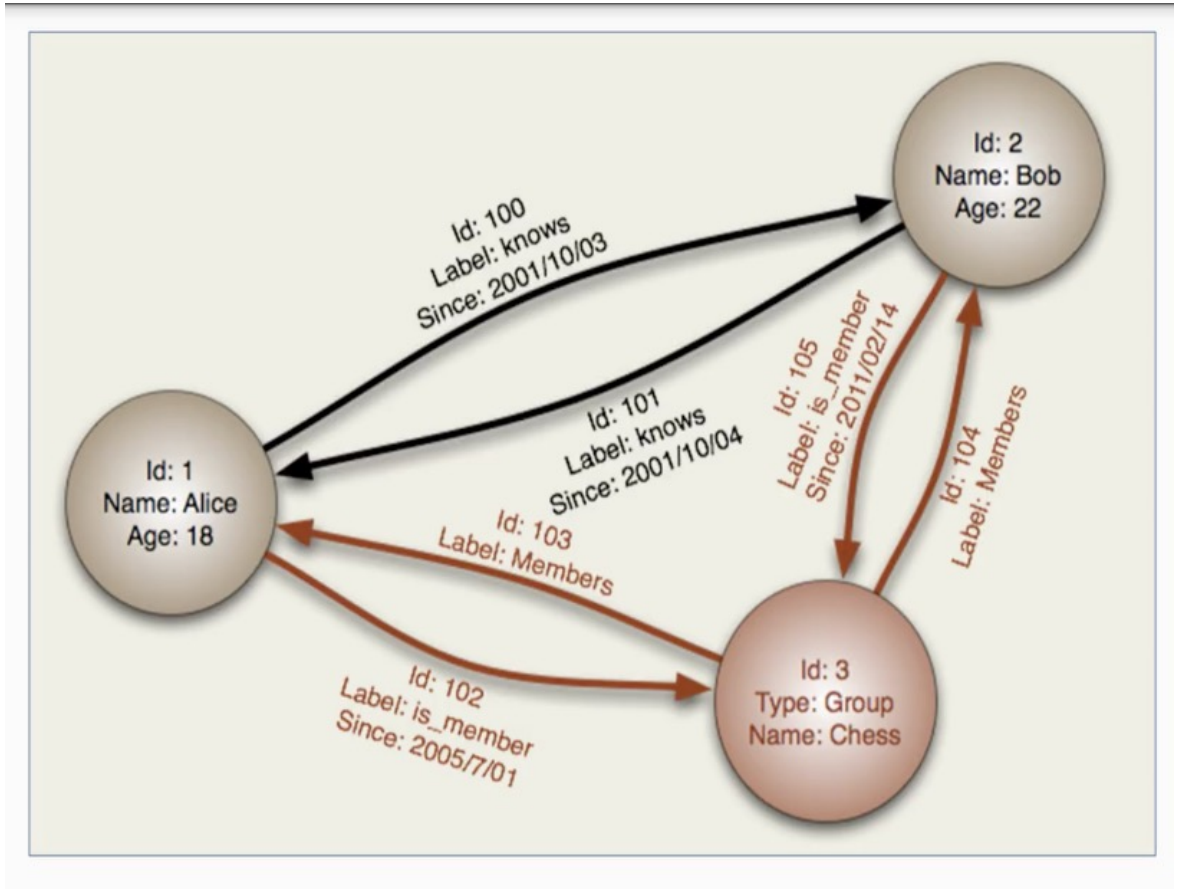
[1] Department of Computer Science, Boston University, Boston, MA, USA, [2] Center for Advanced Genomic Technologies, Boston University, Boston, MA, USA, [3] Department of Biomedical Engineering, Boston University, Boston, MA, USA, [4] Center for Advanced Biotechnology, Boston University, Boston, MA, USA, [5] SEQUENOM Inc., San Diego, CA, USA and [6] Children's Hospital Boston, Boston, MA, USA
* Corresponding author. Department of Computer Science, Boston University, 111 Cummington Ave, Boston, MA 02215, USA. Tel.: +1 617 921 9669; Fax: +1 617 353 4814; E-mail: rachlin@bu.edu

Received 30.1.06; accepted 22.9.06

Network models are a fundamental tool for the visualization and analysis of molecular interactions occurring in biological systems. While broadly illuminating the molecular machinery of the cell, graphical representations of protein interaction networks mask complex patterns of interaction that depend on temporal, spatial, or condition-specific contexts. In this paper, we introduce a novel graph construct called a biological context network that explicitly captures these changing patterns of interaction from one biological context to another. We consider known gene ontology biological process and cellular component annotations as a proxy for context, and show that aggregating small process-specific protein interaction sub-networks leads to the emergence of observed scale-free properties. The biological context model also provides the basis for characterizing proteins in terms of several context-specific measures, including 'interactive promiscuity,' which identifies proteins whose interacting partners vary from one context to another. We show that such context-sensitive measures are significantly better predictors of knockout lethality than node degree, reaching better than 70% accuracy among the top scoring proteins.
Molecular Systems Biology 28 November 2006; doi:10.1038/msb4100103
Subject Categories: metabolic and regulatory networks
Keywords: bioinformatics; biological context; network models; PPI networks; scale-free networks

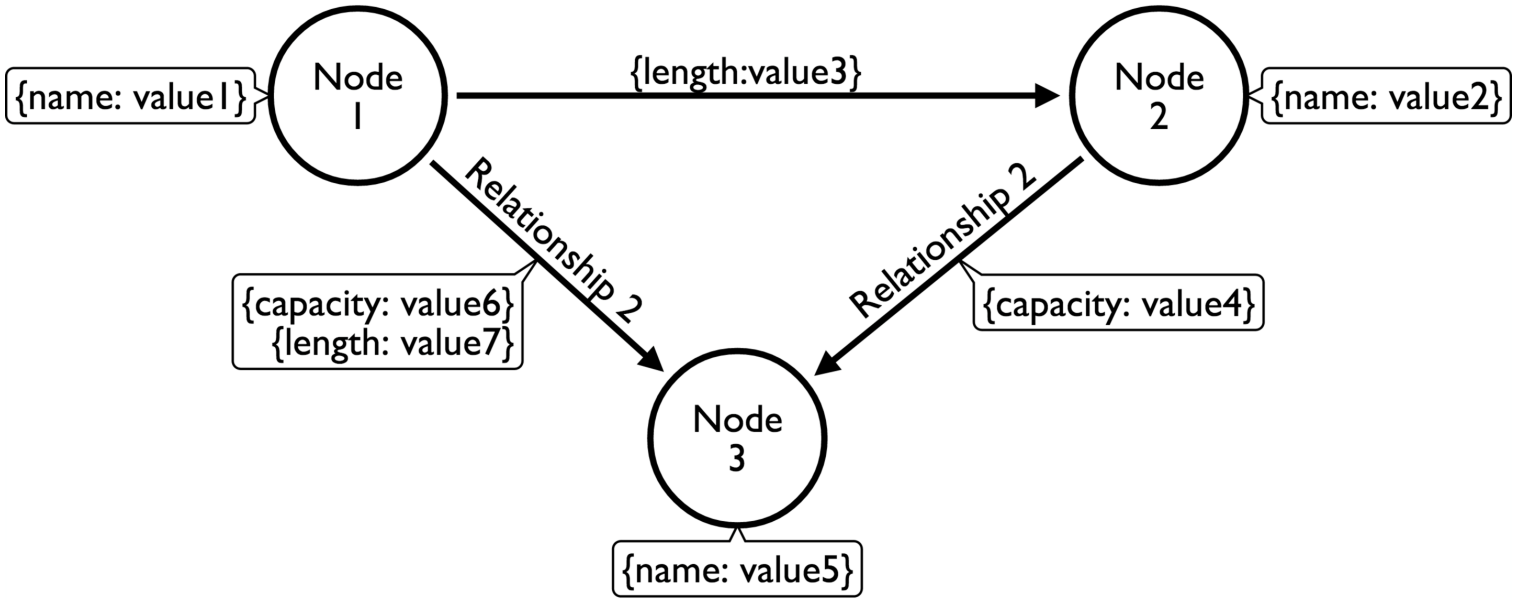
Relational vs Property Graph





Property graphs

- No fixed schema
- Both nodes and relationships can have properties
- Directed graphs
- Multiple relationships between nodes
- Nodes can have 0,1,many **labels**
- Relationships have one and only one **type**



Index-Free Adjacency

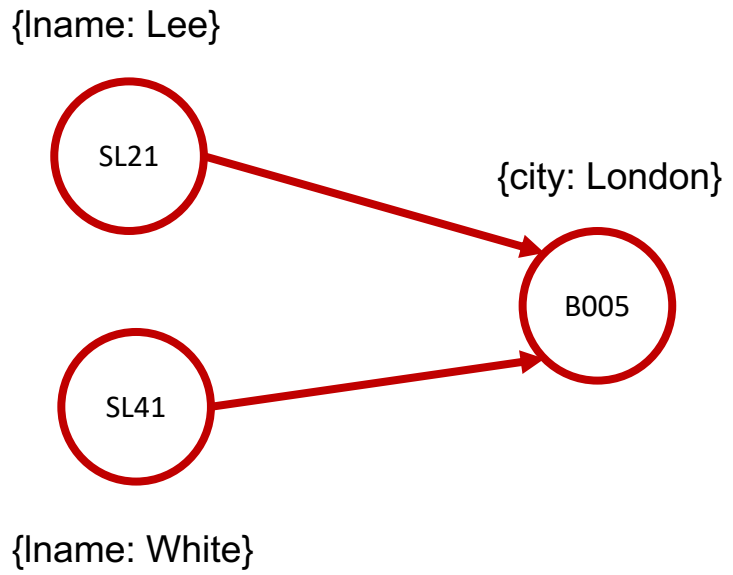
In a graph database it doesn't require a search through an index to find adjacent nodes.

Branch

branchNo	street	city	postCode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

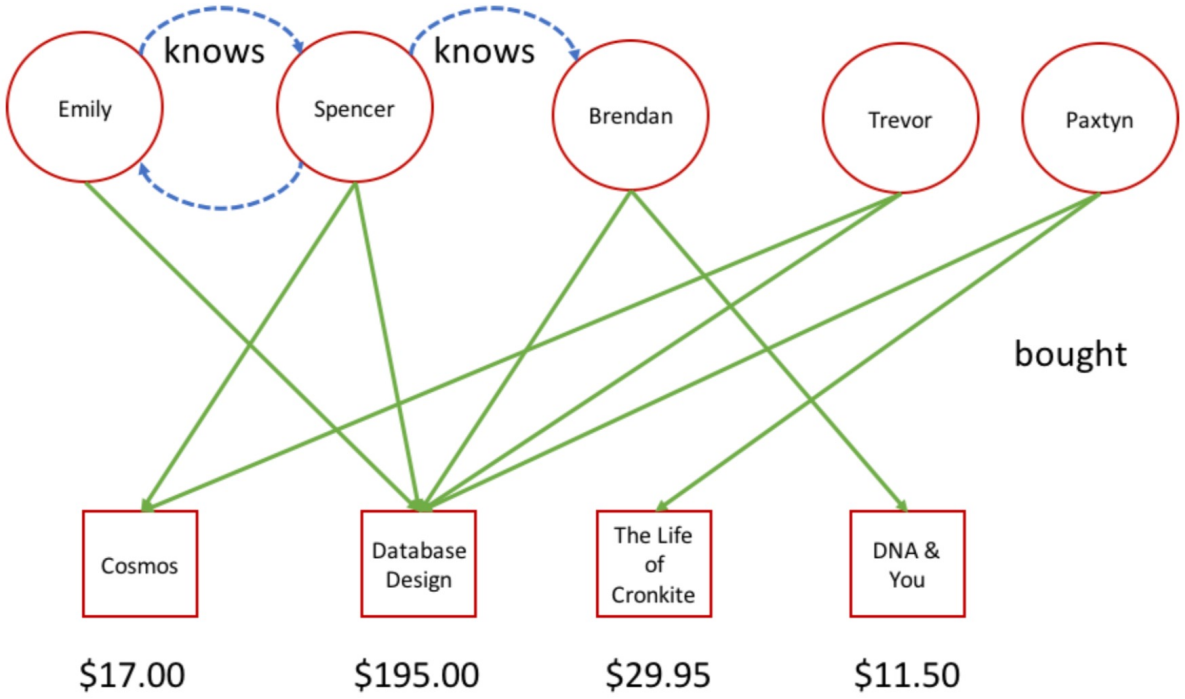
Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005



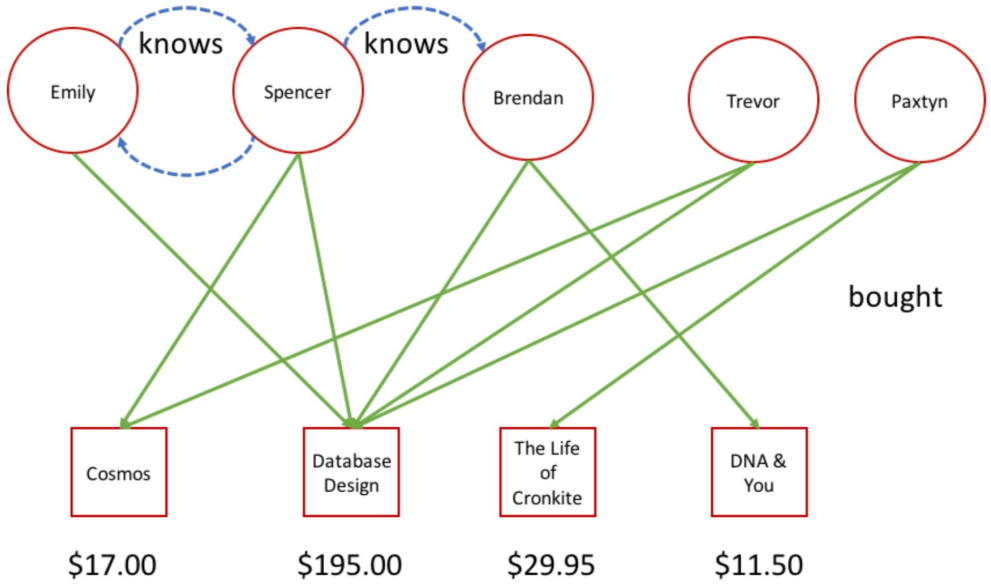
Instead we follow or *chase* the pointers!

Representing Graphs in a Relational DB




```
create table node (
 node_id int primary key,
 type varchar(20)
 );

 create table edge (
 edge_id int primary key,
  in_node int,
  out_node int,
  type varchar(20)
  );

create table node_props (
  node_id int,
  propkey varchar(20),
  string_value varchar(100),
  num_value double
  );
```

Representing Graphs in a Relational DB




```
insert into node values
(1,'Person'),
(2,'Person'),
(3,'Person'),
(4,'Person'),
(5,'Person'),
(6,'Book'),
(7,'Book'),
(8,'Book'),
(9,'Book');
```

```
insert into edge values
(1, 1, 7, 'bought'),
(2, 2, 6, 'bought'),
(3, 2, 7, 'bought'),
(4, 3, 7, 'bought'),
(5, 3, 9, 'bought'),
(6, 4, 6, 'bought'),
(7, 4, 7, 'bought'),
(8, 5, 7, 'bought'),
(9, 5, 8, 'bought'),
(10, 1,2,'knows'),
(11, 2, 1, 'knows'),
(12, 2, 3, 'knows');
```

```
insert into node_props values
(1, 'name', 'Emily', null),
(2, 'name', 'Spencer', null),
(3, 'name', 'Brendan', null),
(4, 'name', 'Trevor', null),
(5, 'name', 'Paxton', null),
(6, 'title', 'Cosmos', null),
(6, 'price', null, 17.00),
(7, 'title', 'Database Design', null),
(7, 'price', null, 195.00),
(8, 'title', 'The Life of Cronkite', null),
(8, 'price', null, 29.95),
(9, 'title', 'DNA and you', null),
(9, 'price', null, 11.50);
```

Representing Graphs in a Relational DB

Who does Spencer know?

```
select npin.string_value who_in, e.type, npout.string_value who_out
from edge e
    join node nin on (e.in_node = nin.node_id)
    join node_props npin on (nin.node_id = npin.node_id)
    join node nout on (e.out_node = nout.node_id)
    join node_props npout on (nout.node_id = npout.node_id)
where e.type = 'knows'
and npin.propkey = 'name'
and npin.string_value = 'Spencer';
```

What books were bought by people who Spencer knows?

```
select  npout.string_value title, count(*) purchased
from edge e
    join node nin on (e.in_node = nin.node_id)
    join node_props npin on (nin.node_id = npin.node_id)
    join node nout on (e.out_node = nout.node_id)
    join node_props npout on (nout.node_id = npout.node_id)
where e.type = 'bought'
and npin.propkey = 'name'
and npin.string_value in (
    select  npout.string_value
    from edge e
        join node nin on (e.in_node = nin.node_id)
        join node_props npin on (nin.node_id = npin.node_id)
        join node nout on (e.out_node = nout.node_id)
        join node_props npout on (nout.node_id = npout.node_id)
    where e.type = 'knows'
    and npin.propkey = 'name'
    and npin.string_value = 'Spencer'
)
and npout.propkey = 'title'
group by title
order by count(*) desc;
```

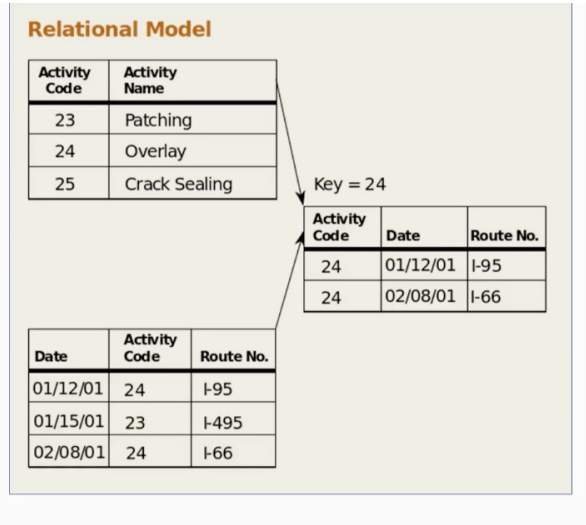
How are Graph DBs different from Relation DBs

Relational:

Highly structured with rows and tables

patterns of join: 1:1, 1:*, *:*

Recursion is difficult

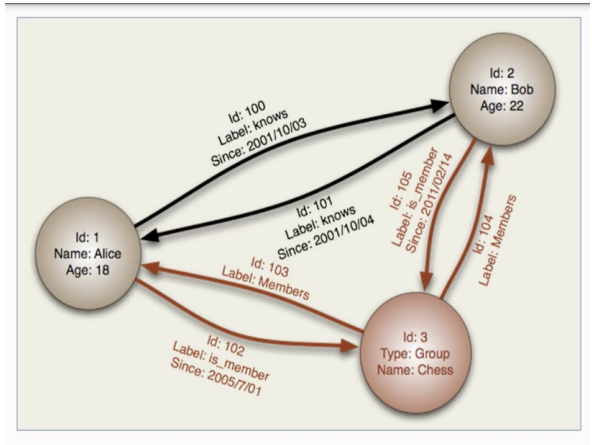


Graph:

Data is not just hierarchical

Flexible relationships and data structures

No pre-defined data models



neo4j

PRODUCTS SOLUTIONS CUSTOMERS PARTNERS RESOURCES DEVELOPERS DOWNLOAD NEO4J

INTRODUCING

Neo4j 4.0

The Next-Generation Graph Database Built for Unlimited Scale and Development Agility

Learn More

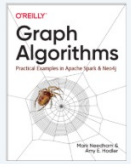
```
MERGE (n:DBMS {name: 'Neo4j'})
SET n += {scalability: 'Unlimited',
          security: 'Granular',
          architecture: 'Reactive'}
```

IBM Microsoft die Bayerische CenturyLink AstraZeneca

What is Neo4j?

Learn the basics in less than 2 minutes

Watch Video



Free O'Reilly Ebook

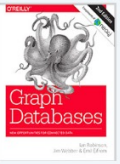
Graph Algorithms: Examples in Spark and Neo4j

Sample code and tips for over 20 practical graph algorithms. Find vulnerabilities, detect communities and improve machine learning.

No Download Required

Get Started Quickly with the Neo4j Sandbox

Start using Neo4j within seconds, with built-in guides and sample datasets for popular use cases. No Neo4j experience necessary.



Free O'Reilly Ebook

Graph Databases for Connected Data

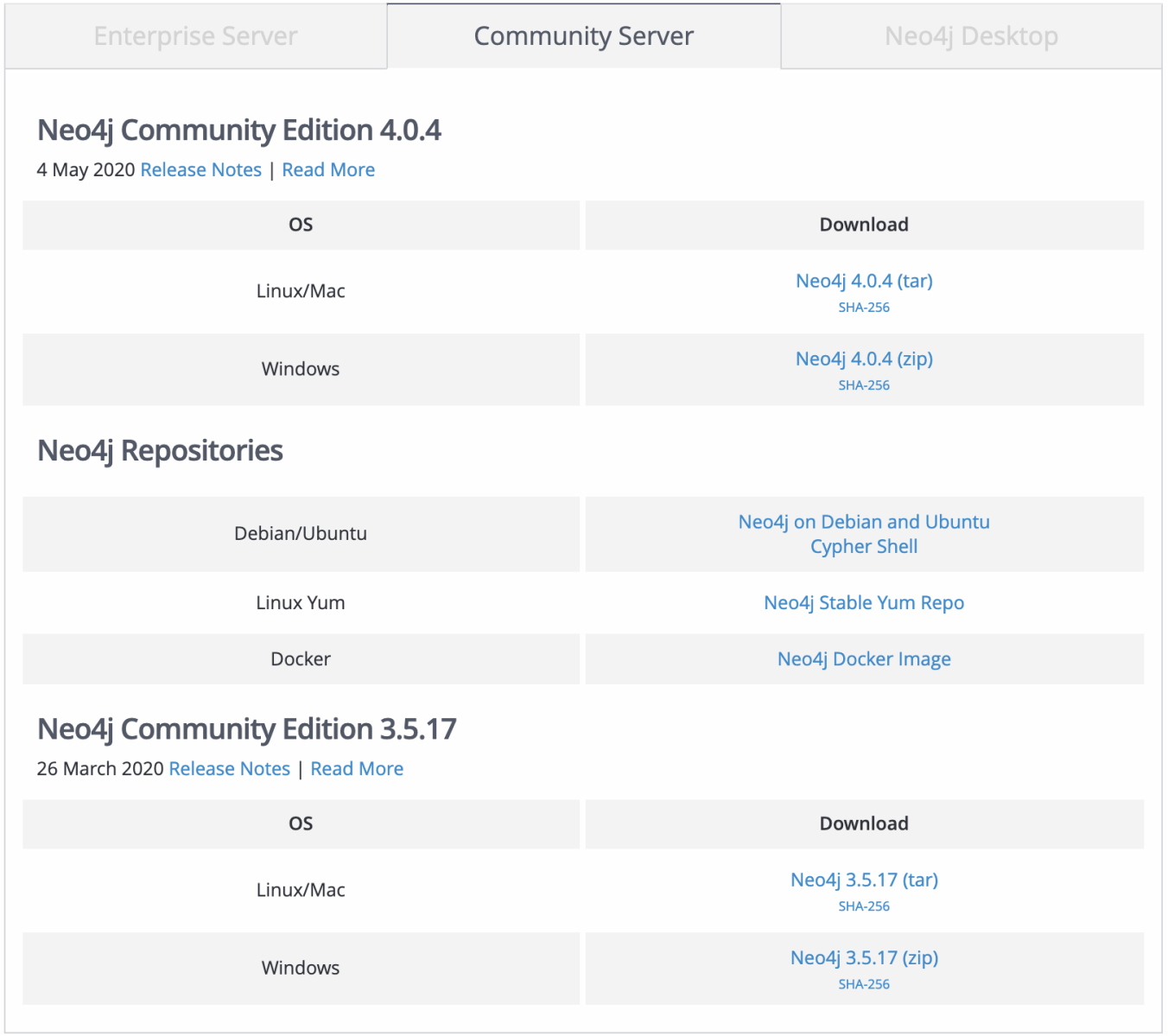
Get started with O'Reilly's *Graph Databases* and discover how graph databases can help you manage and query highly connected data.

Installing Neo4J: https://neo4j.com/download-center/#community

Download the server

Version 4.0.4 requires Java 11
Version 3.5.17 requires Java 8

Either will work for this class.



Enterprise Server | Community Server | Neo4j Desktop

Neo4j Community Edition 4.0.4

4 May 2020 Release Notes | Read More

OS	Download
Linux/Mac	Neo4j 4.0.4 (tar) SHA-256
Windows	Neo4j 4.0.4 (zip) SHA-256

Neo4j Repositories

Debian/Ubuntu	Neo4j on Debian and Ubuntu Cypher Shell
Linux Yum	Neo4j Stable Yum Repo
Docker	Neo4j Docker Image

Neo4j Community Edition 3.5.17

26 March 2020 Release Notes | Read More

OS	Download
Linux/Mac	Neo4j 3.5.17 (tar) SHA-256
Windows	Neo4j 3.5.17 (zip) SHA-256

Installing the Server on a Mac

I installed the Neo4J 3.5.17 server in my $HOME/apps folder:

```
[539 uranus:~ ] cd apps
[540 uranus:apps ] cp ~/Downloads/neo4j-community-3.5.17-unix.tar.gz .
[541 uranus:apps ] tar xzf neo4j-community-3.5.17-unix.tar.gz
[542 uranus:apps ] ls
```

```
neo4j-community-3.3.4
neo4j-community-3.5.17
neo4j-community-3.5.17-unix.tar.gz
neo4j-community-3.5.3
neo4j-community-4.0.3
```

Installing (Mac)

I edited my **.bash_profile** as follows:

```
export APPS=/Users/rachlin/apps
export NEO4J_HOME=$APPS/neo4j-community-3.5.17
export PATH=$NEO4J_HOME/bin:$PATH
```

Users of Mac OSX Catalina:
Apple now uses zsh instead of bash as the default shell. Instead of editing .bashrc or .bash_profile, you would edit .zshrc or .zprofile.

Starting the server: $ neo4j start or $ neo4j console

```
(base) [543 uranus:apps ] neo4j start
Active database: graph.db
Directories in use:
  home:         /Users/rachlin/apps/neo4j-community-3.5.17
  config:       /Users/rachlin/apps/neo4j-community-3.5.17/conf
  logs:         /Users/rachlin/apps/neo4j-community-3.5.17/logs
  plugins:      /Users/rachlin/apps/neo4j-community-3.5.17/plugins
  import:       /Users/rachlin/apps/neo4j-community-3.5.17/import
  data:         /Users/rachlin/apps/neo4j-community-3.5.17/data
  certificates: /Users/rachlin/apps/neo4j-community-3.5.17/certificates
  run:          /Users/rachlin/apps/neo4j-community-3.5.17/run
Starting Neo4j.
Started neo4j (pid 64350). It is available at http://localhost:7474/
There may be a short delay until the server is ready.
See /Users/rachlin/apps/neo4j-community-3.5.17/logs/neo4j.log for current status
```

Windows Installation

Neo4j Community Edition 4.0.4

4 May 2020 Release Notes | Read More

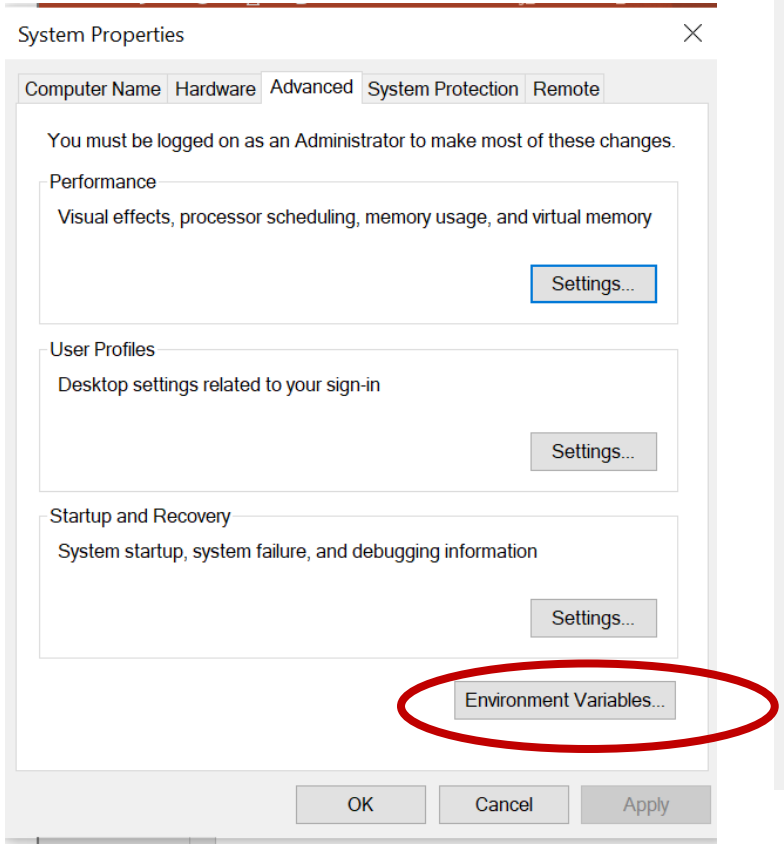
OS	Download
Linux/Mac	Neo4j 4.0.4 (tar) SHA-256
Windows	Neo4j 4.0.4 (zip) SHA-256

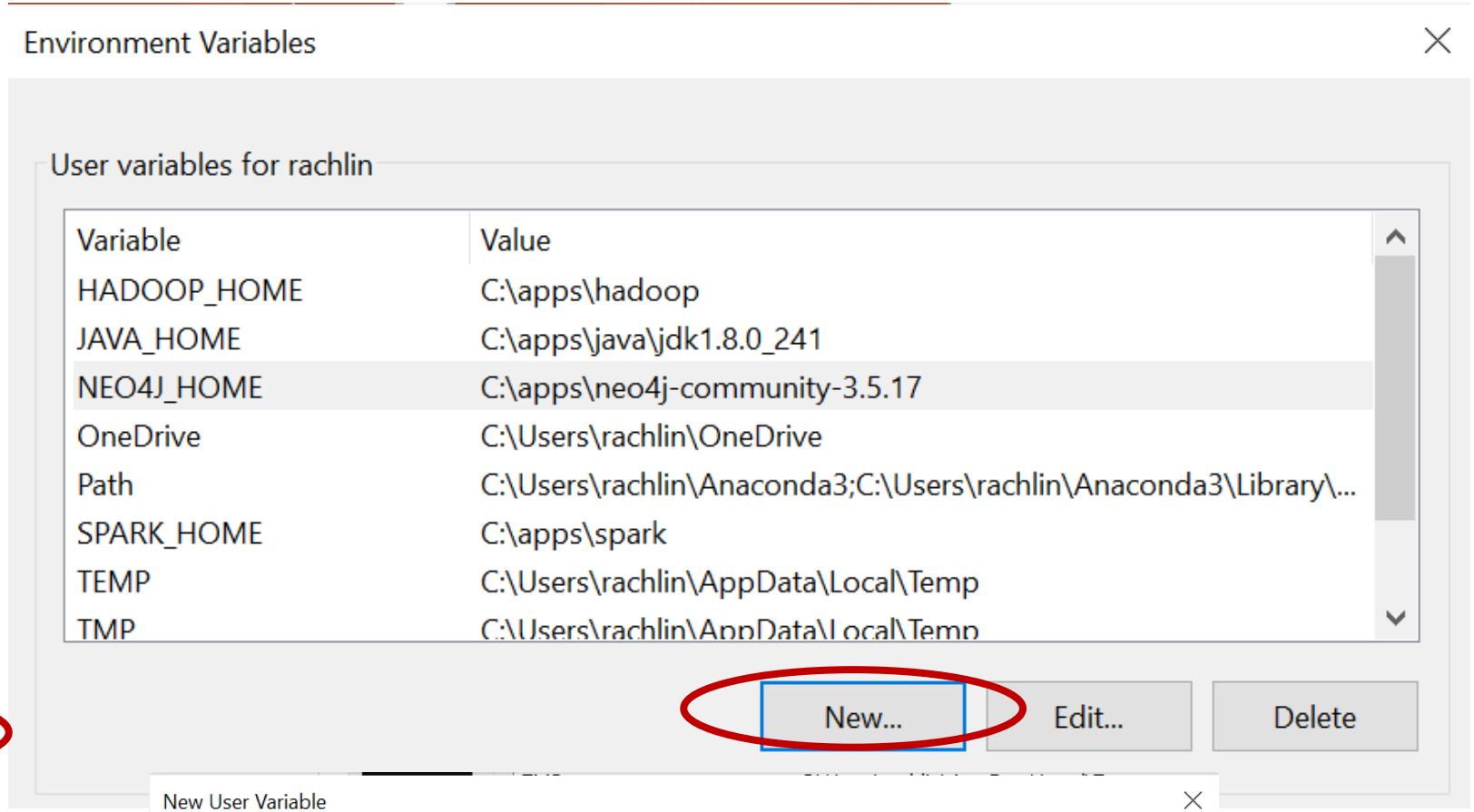
Extract zip file to a directory of your choice

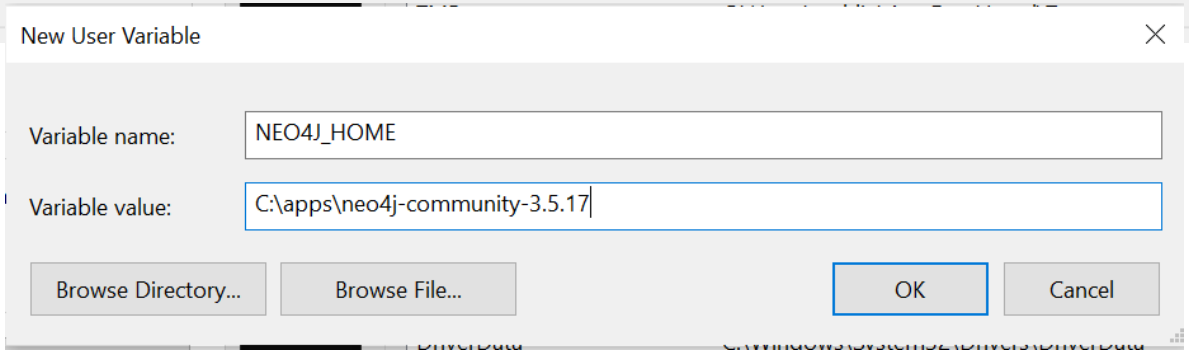
I like to use “C:\apps” for this sort of thing.

```
C:\apps>dir
 Volume in drive C has no label.
 Volume Serial Number is 5AAB-E91D

 Directory of C:\apps

05/06/2020  04:20 PM    <DIR>          .
05/06/2020  04:20 PM    <DIR>          ..
04/06/2020  08:46 PM    <DIR>          hadoop
04/06/2020  08:30 PM    <DIR>          java
05/06/2020  04:14 PM    <DIR>          neo4j-community-3.5.17
04/06/2020  08:40 PM    <DIR>          spark
04/06/2020  09:21 PM    <DIR>          zeppelin
               0 File(s)              0 bytes
               7 Dir(s)  59,934,076,928 bytes free

C:\apps>
```

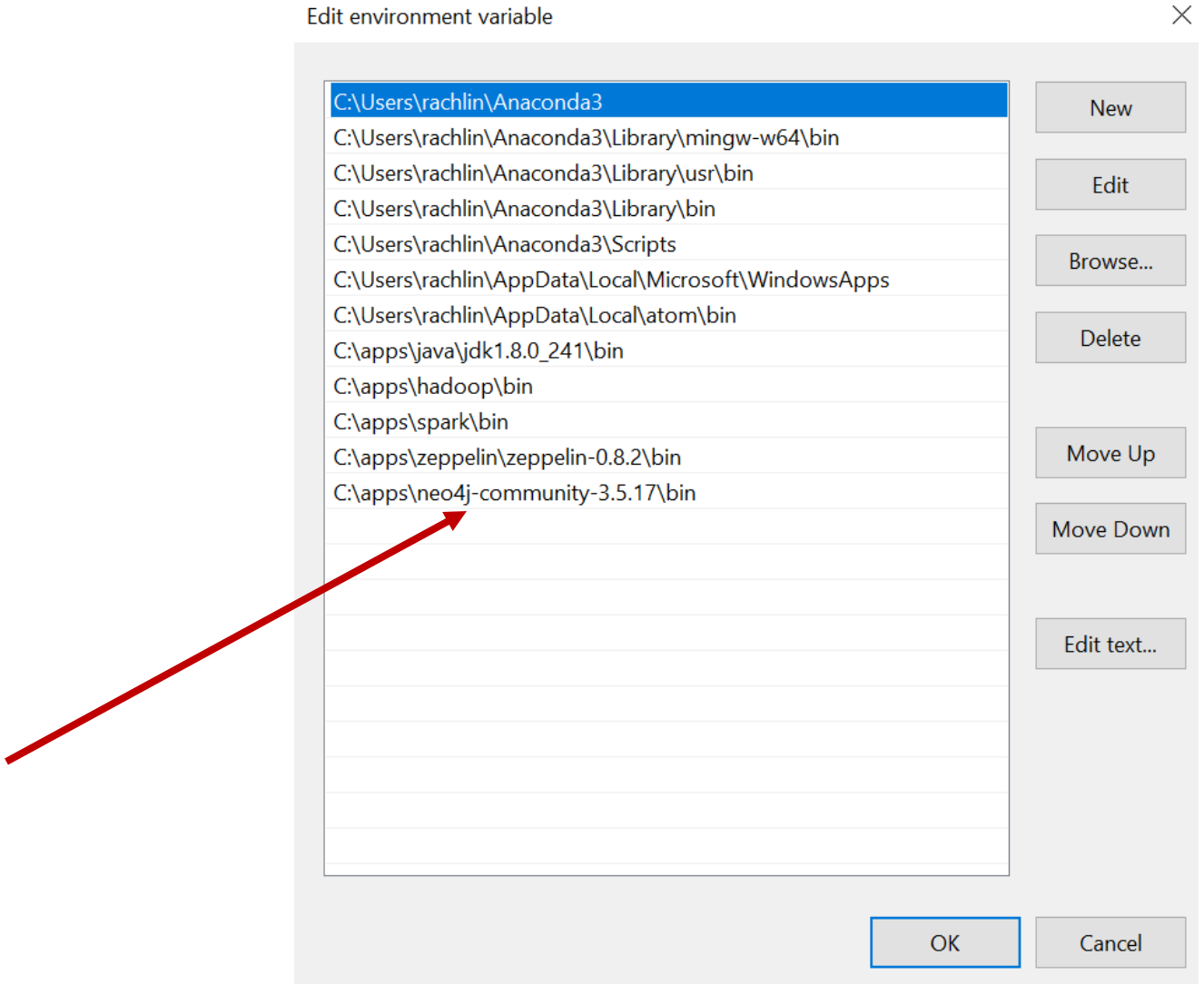
Add a windows environmental variable for NEO4J_HOME





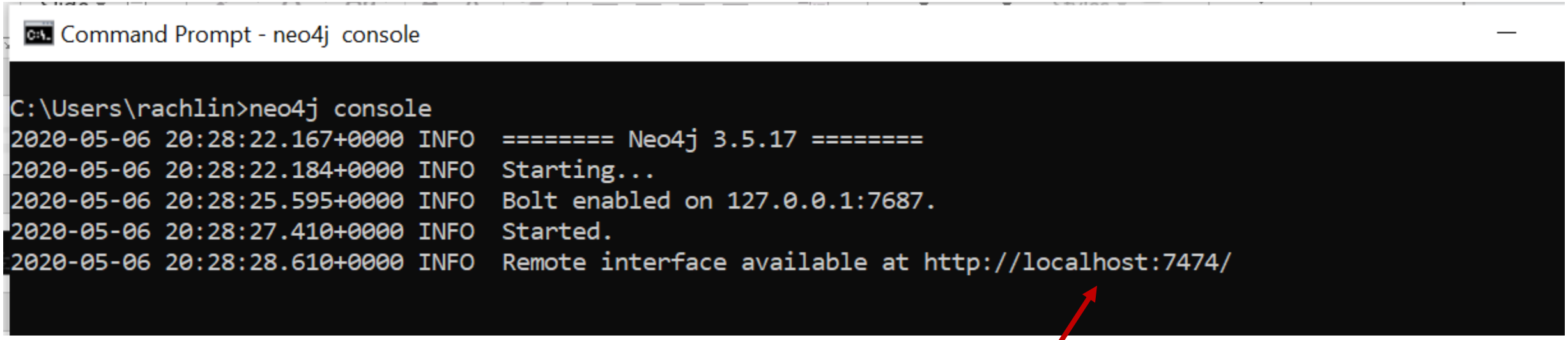


Also edit your path and add the *bin* directory



After saving and restarting the terminal….

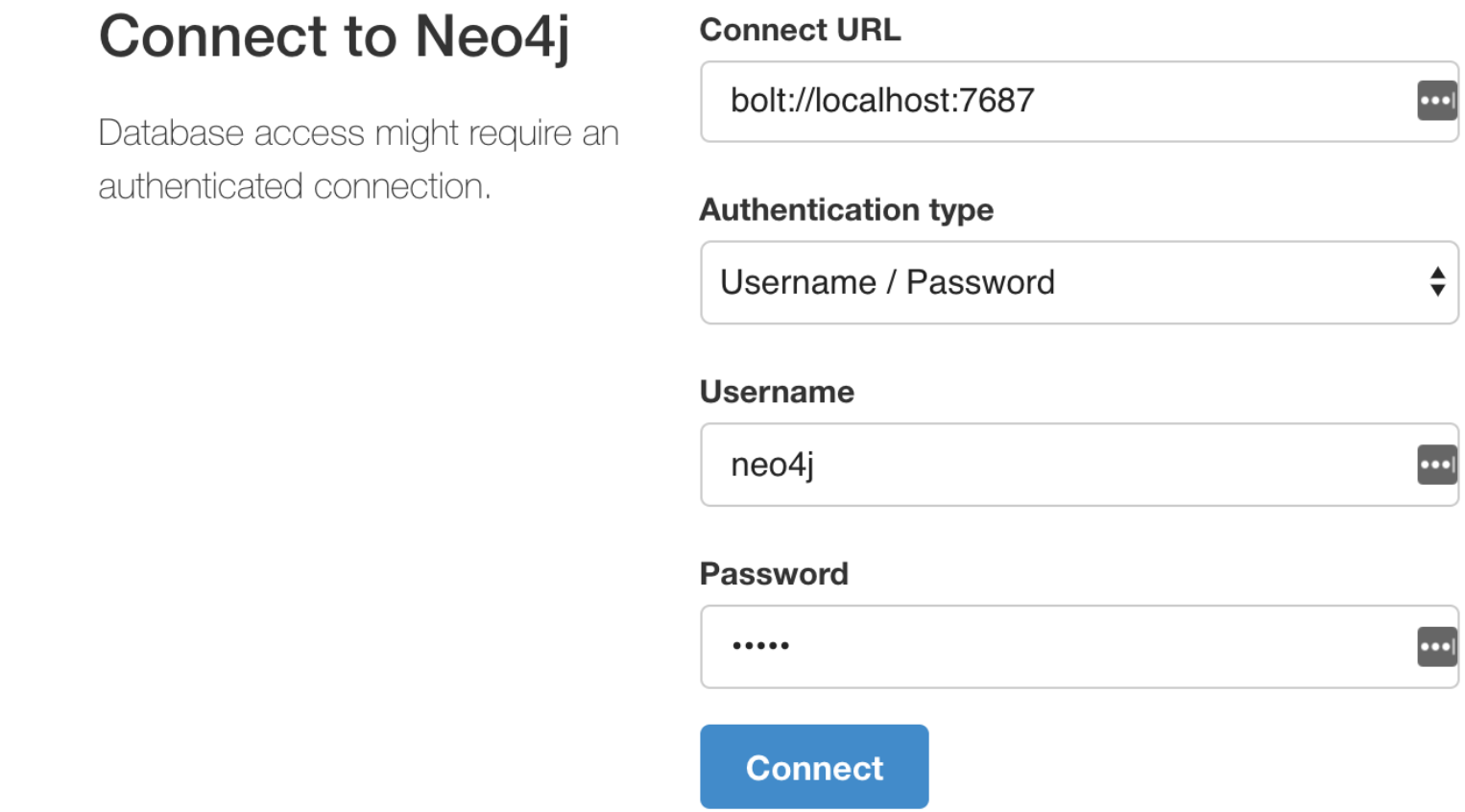
You can start the NEO4J server with the command: *neo4j console*



Open browser to use Neo4J

Connecting to server: localhost:7474

```
$ :server connect
```



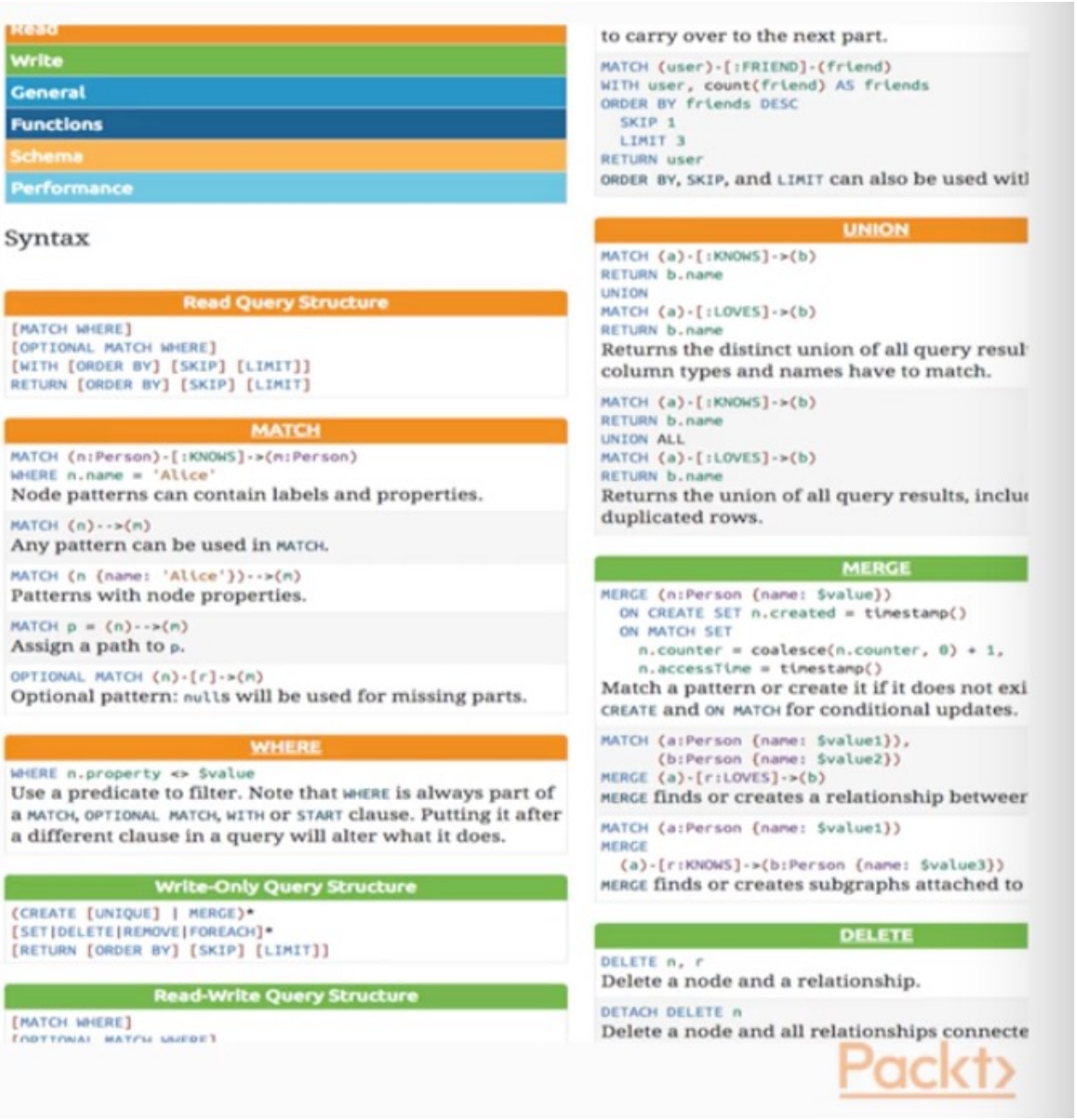
Default: **neo4j**

Default: **neo4j**

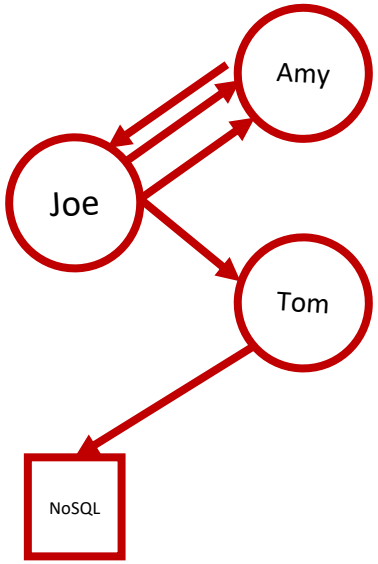
(It will ask you to change)

Helpful Resources

- Helpful links
  - www.neo4j.com
  - neo4j.com/docs/cypher-refcard/current/
  - https://github.com/neo4j
  - https://github.com/issues
- SLACK!
  - https://neo4j.com/blog/public-neo4j-users-slack-group/



Native vs. Non-native graph storage



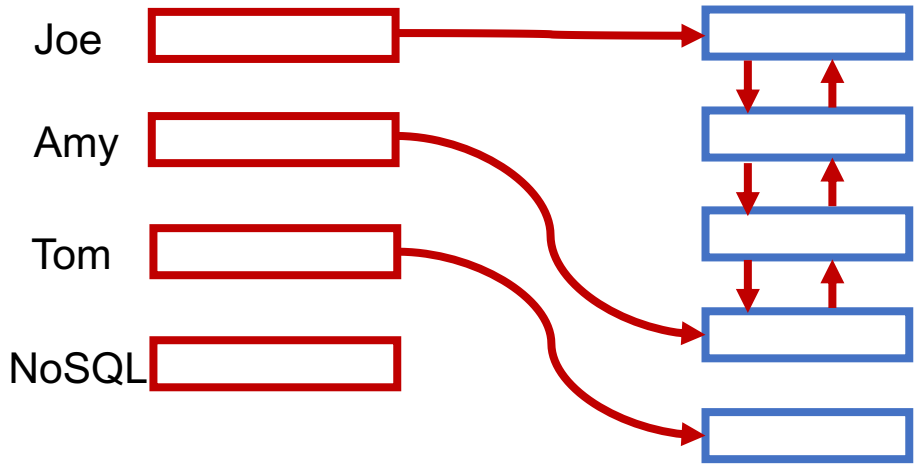
Non-Native

Nodes
1 Person
2 Person
3 Person
4 Book

Relationships
1 1 2 Knows
2 1 2 MarriedTo
3 2 1 MarriedTo
4 1 3 Knows
5 3 4 Bought

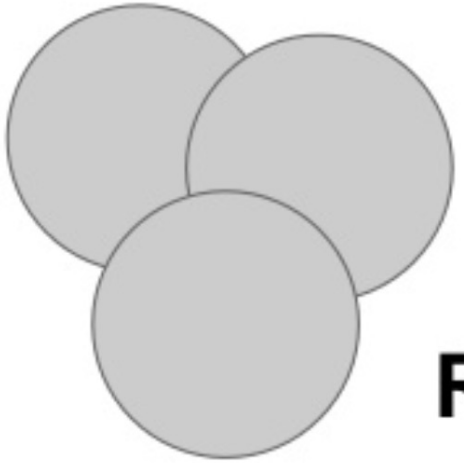
Properties
1 1 Name Joe
2 2 Name Amy
3 3 Name Tom
4 4 Title NoSQL
5 4 Price 29.95

Native



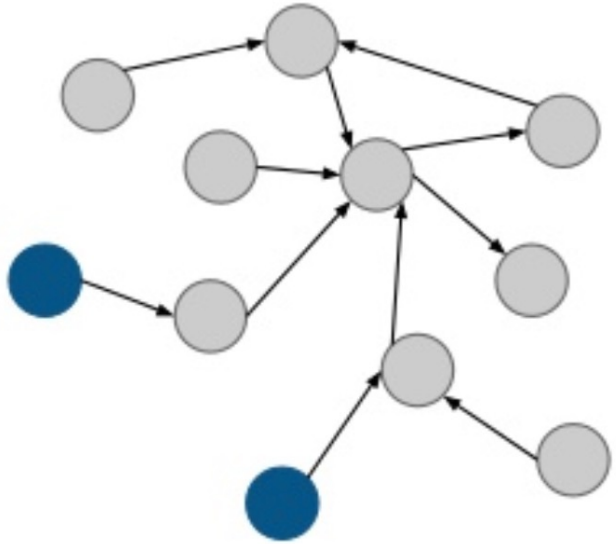
Native vs. Non-native graph processing

Indexes are **only** used to find the starting point for queries.



Relational

Use index scans to look up rows in tables and join them with rows from other tables



Graph

Use indexes to find the starting points for a query.

- Index-free adjacency (no global index)
- Pointer chasing to search graph

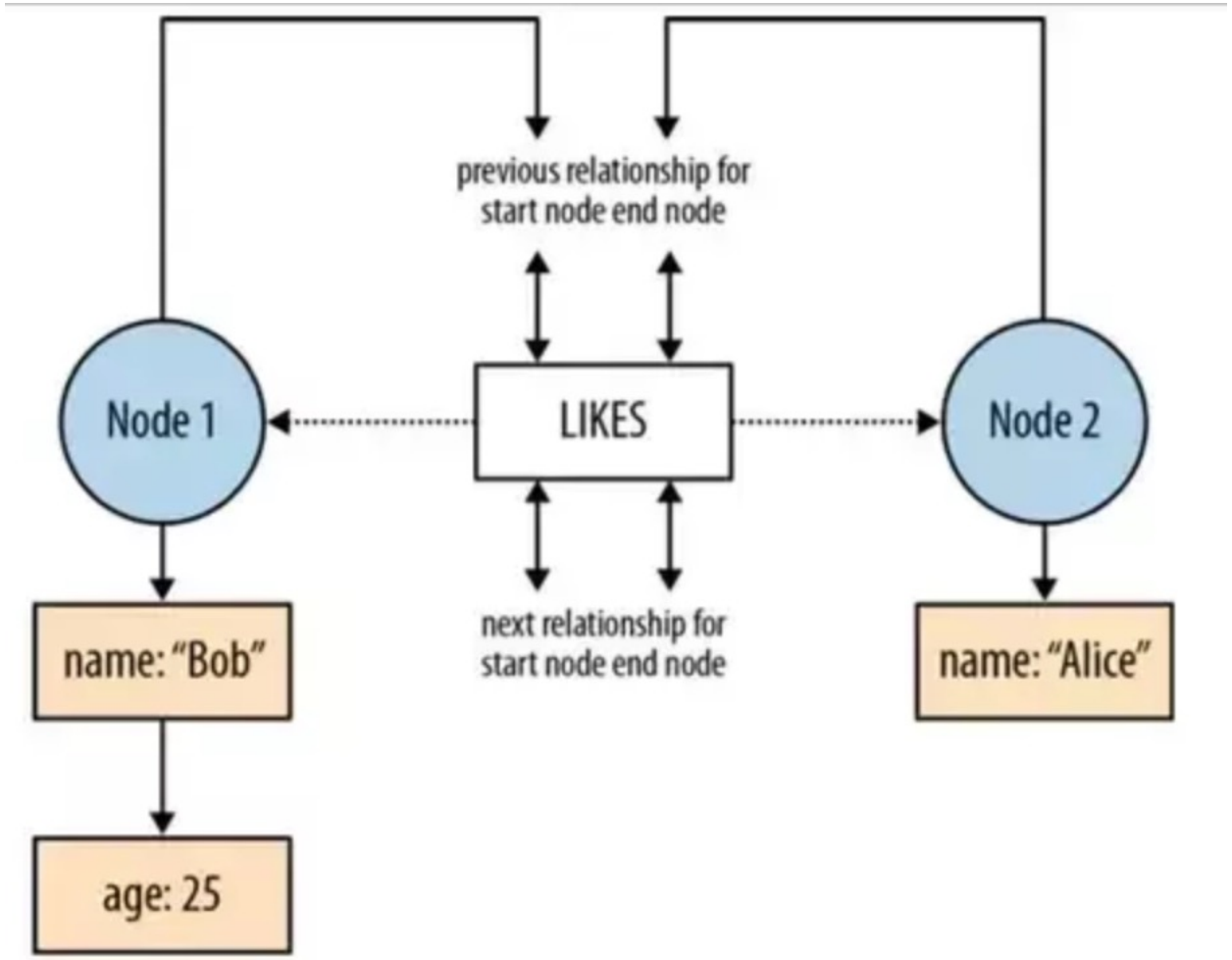
Neo4J Storage Engine

Linked-lists of fixed-size records on disk.

Properties: Link-list of key-value pairs

Nodes: Point to start of a relationship list, property list

Relationships: A doubly-linked list of relationships associated with both the start node and the end node



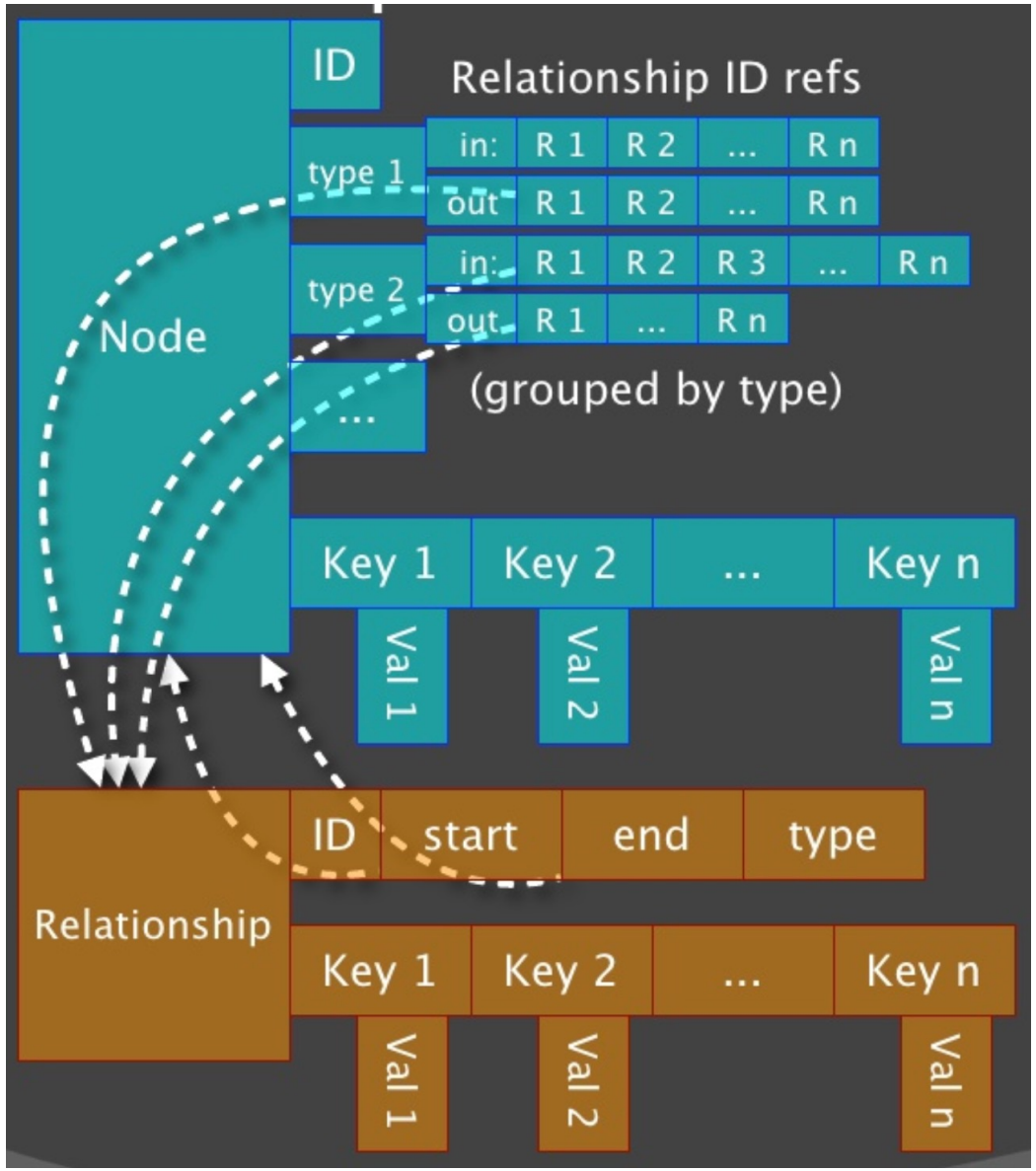
Neo4J Caching

Low-level (File System) cache:

- Stores are partitioned into regions
- Cache holds subset of regions
- Least-Frequently Used swapping of entire regions

High-level (Object) cache:

Optimized for traversal, particularly along a particular relationship type

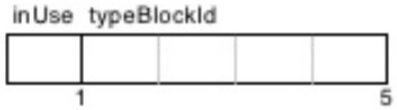


Neo4J Record Stores

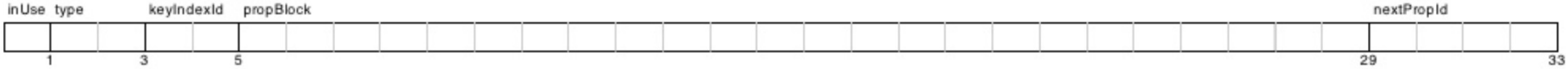
Node (9 bytes)

Relationship (33 bytes)

Relationship Type (5 bytes)



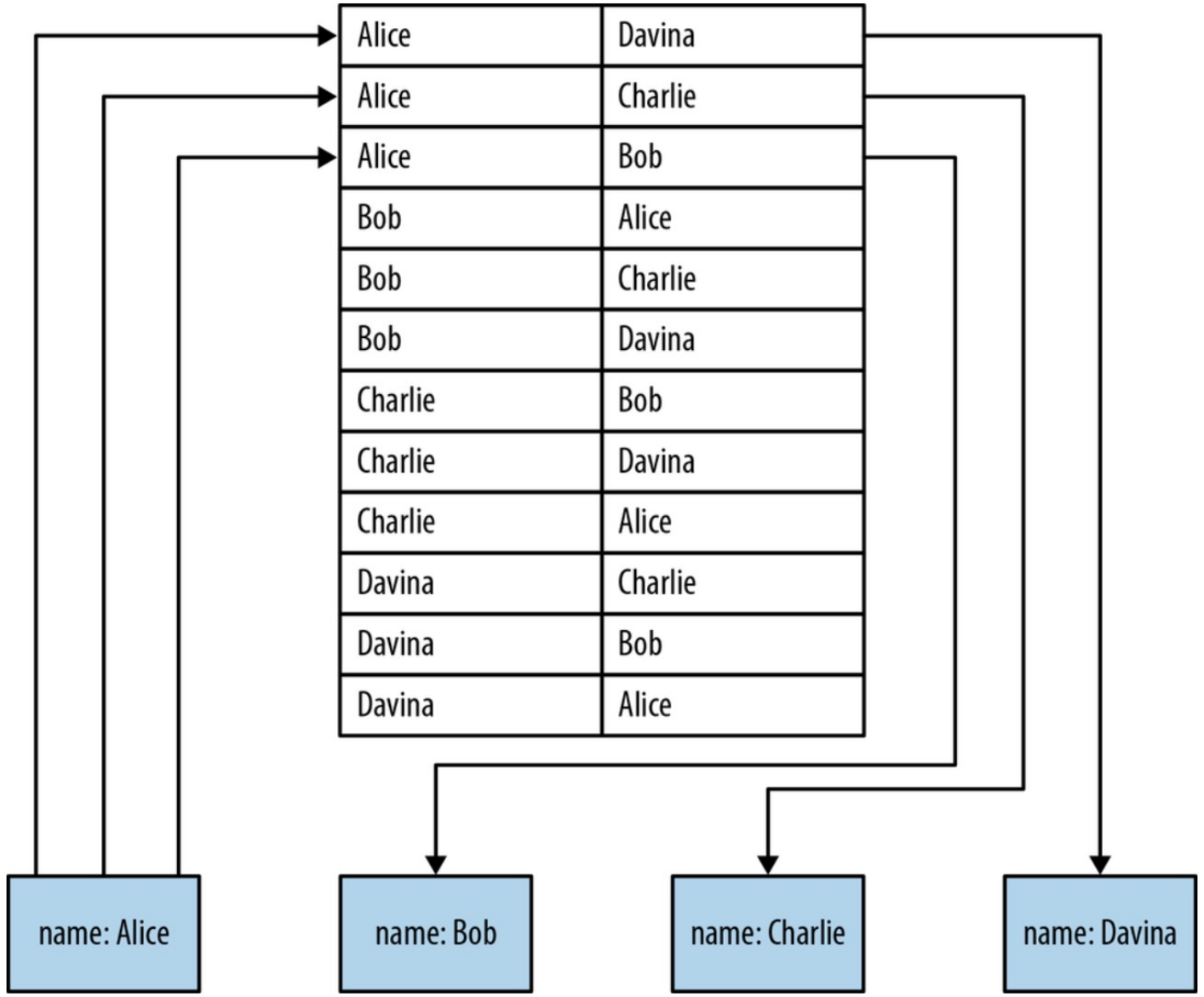
Property (33 bytes)



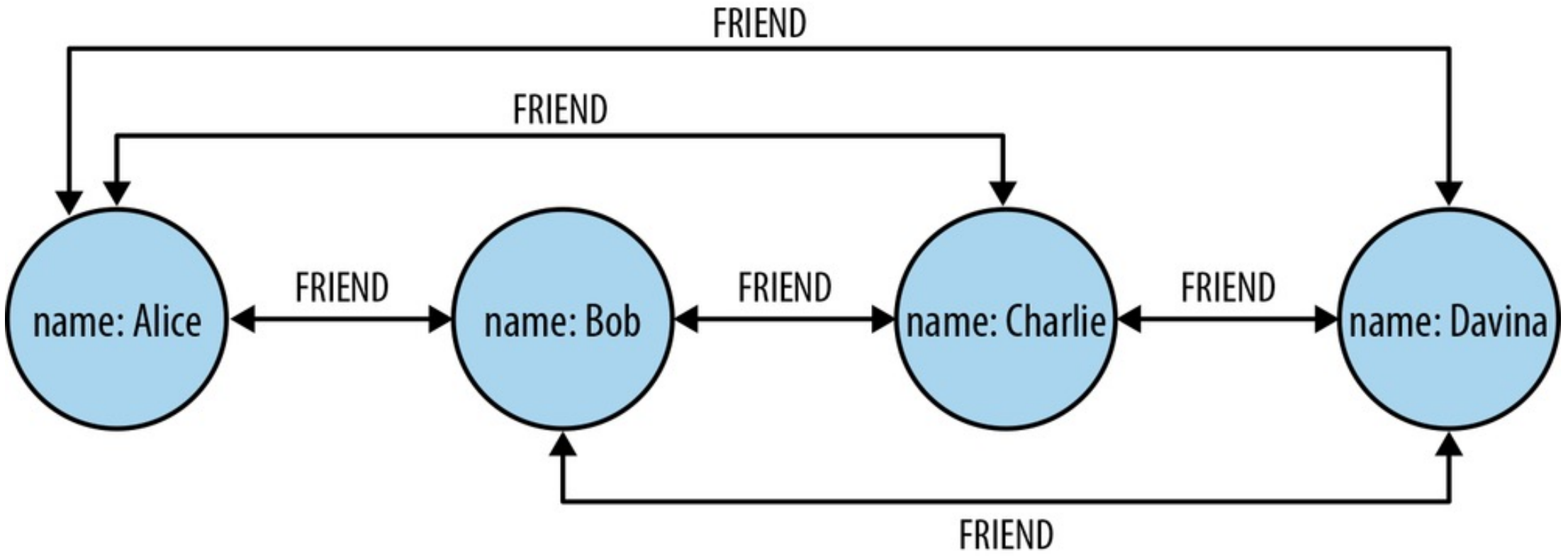
Non-native graph processing

Finding who Alice likes requires an index search **O**(log n)

The reverse search requires a second index or a scan across all records **O**(n)



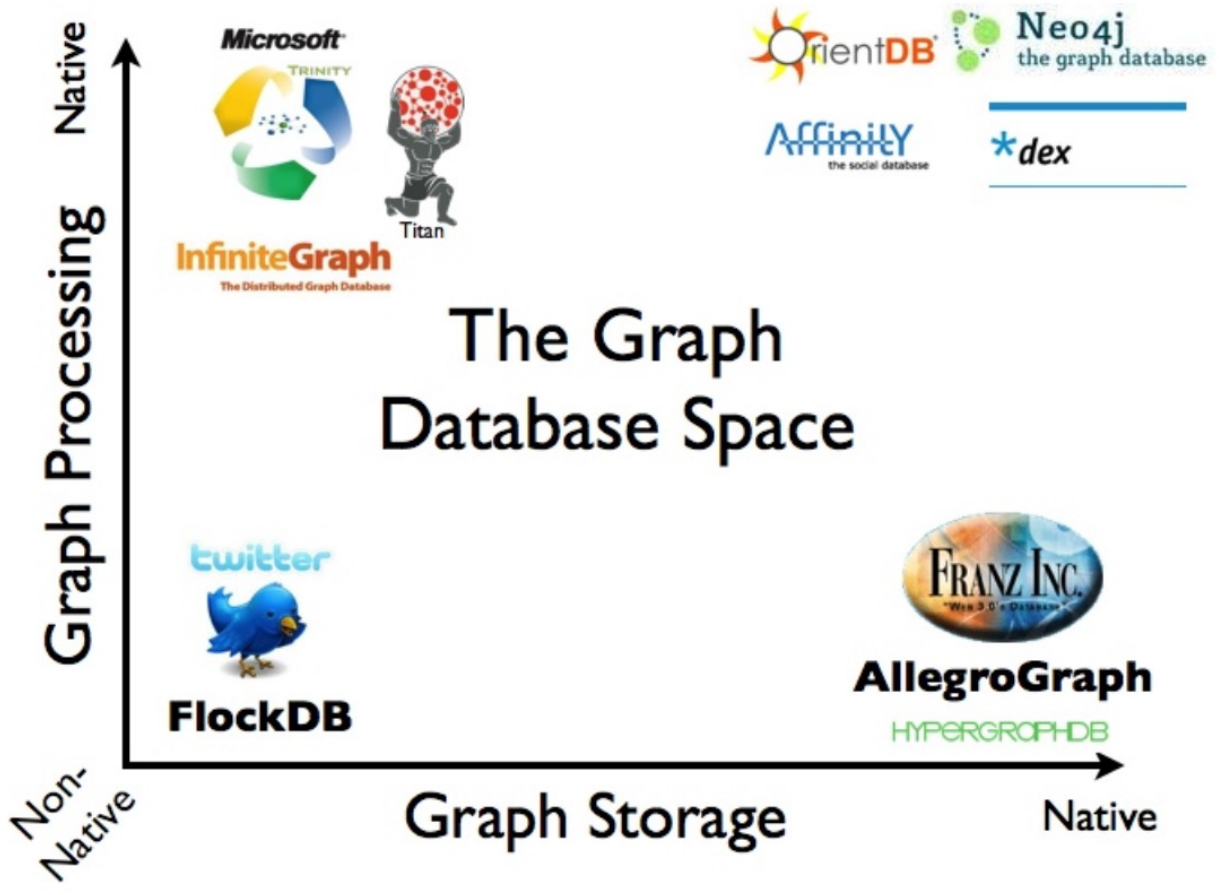
Native Graph Processing



We can find who Alice likes and who likes Alice in **O**(m) time.
(m is the number of Alice's friends, and typically m<<n)

Storage	Processing	Advantages	Disadvantages	Performance (Graph as n nodes)
Non-native (e.g., RDB)	**Non-native** (Global indexes, JOINS, and other SQL-driven queries)	Mature technology stack	Complex queries Some graph queries (like path-finding) are impractical or impossible.	Traverse m edges **O**(m log n) Fetch node properties **O**(log n)
Non-native	**Native / mixed** nodes and relationships mapped to in-memory graph model. Properties kept in an RDB.	Leverage mature technology for durability. Graph algorithms can now be implemented	Memory limits Sync between RDB and in-memory model (limited to static data, non-transactional)	Traverse m edges **O**(m) Fetch node properties **O**(log n)
Native	**Native** Data is stored on disk in a way that reflects the graph's topology. (Neighboring nodes are stored close together.) *Index-free adjacency / pointer-chasing.*	Speed. Scalability. Query run-time depends on how much of the graph is traversed, not the total size of the graph.	Graph traversal algorithms may be quite slow on large graphs. Requires specialization in NoSQL.	Traverse m edges **O**(m) Fetch node properties **O**(1)

Native vs. Non-Native storage and processing



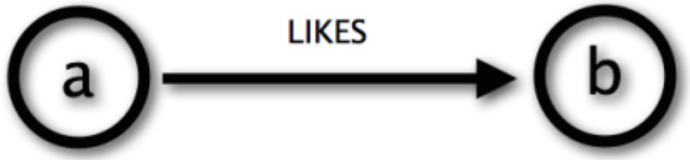
Native:
Index-free Adjacency
Object records maintain pointers to neighbors
Processing time determined by amount of graph traversed.

Non-native:
Indexes are global.
Processing time impacted by total graph size.

Cypher

- Cypher is a declarative, SQL-inspired language for describing patterns in graphs visually using an ascii-art syntax.

- It allows us to state what we want to select, insert, update or delete from our graph data without requiring us to describe exactly how to do it.



Cypher

```
(a) -[:LIKES]-> (b)
```

Cypher: CREATE

```
CREATE (n {name: $value})
```

Create a node with the given properties.

```
CREATE (n $map)
```

Create a node with the given properties.

```
UNWIND $listOfMaps AS properties
CREATE (n) SET n = properties
```

Create nodes with the given properties.

```
CREATE (n)-[r:KNOWS]->(m)
```

Create a relationship with the given type and direction; bind a variable to it.

```
CREATE (n)-[:LOVES {since: $value}]->(m)
```

Create a relationship with the given type, direction, and properties.

Cypher: LOAD CSV

```
LOAD CSV FROM
'https://neo4j.com/docs/cypher-refcard/3.5/csv/artists.csv' AS line
CREATE (:Artist {name: line[1], year: toInteger(line[2])})
```

Load data from a CSV file and create nodes.

```
LOAD CSV WITH HEADERS FROM
'https://neo4j.com/docs/cypher-refcard/3.5/csv/artists-with-headers.csv' AS line
CREATE (:Artist {name: line.Name, year: toInteger(line.Year)})
```

Load CSV data which has headers.

```
USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM
'https://neo4j.com/docs/cypher-refcard/3.5/csv/artists-with-headers.csv' AS line
CREATE (:Artist {name: line.Name, year: toInteger(line.Year)})
```

Commit the current transaction after every 500 rows when importing large amounts of data.

Cypher DELETE

```
DELETE n, r
```

Delete a node and a relationship.

```
DETACH DELETE n
```

Delete a node and all relationships connected to it.

```
MATCH (n)
DETACH DELETE n
```

Delete all nodes and relationships from the database.

Cypher: Matching

```
MATCH (n:Person)-[:KNOWS]->(m:Person)
WHERE n.name = 'Alice'
```

Node patterns can contain labels and properties.

```
MATCH (n)-->(m)
```

Any pattern can be used in `MATCH`.

```
MATCH (n {name: 'Alice'})-->(m)
```

Patterns with node properties.

```
MATCH p = (n)-->(m)
```

Assign a path to `p`.

```
OPTIONAL MATCH (n)-[r]->(m)
```

Optional pattern: `nulls` will be used for missing parts.

Cypher Patterns

```
(n:Person)
```
Node with `Person` label.

```
(n:Person:Swedish)
```
Node with both `Person` and `Swedish` labels.

```
(n:Person {name: $value})
```
Node with the declared properties.

```
()-[r {name: $value}]-()
```
Matches relationships with the declared properties.

```
(n)-->(m)
```
Relationship from `n` to `m`.

```
(n)--(m)
```
Relationship in any direction between `n` and `m`.

Cypher Patterns

`(n:Person)-->(m)`

Node `n` labeled `Person` with relationship to `m`.

`(m)<-[:KNOWS]-(n)`

Relationship of type `KNOWS` from `n` to `m`.

`(n)-[:KNOWS|:LOVES]->(m)`

Relationship of type `KNOWS` or of type `LOVES` from `n` to `m`.

`(n)-[r]->(m)`

Bind the relationship to variable `r`.

`(n)-[*1..5]->(m)`

Variable length path of between 1 and 5 relationships from `n` to `m`.

`(n)-[*]->(m)`

Variable length path of any number of relationships from `n` to `m`. (See Performance section.)

Cypher Patterns

```
(n)-[:KNOWS]->(m {property: $value})
```
A relationship of type `KNOWS` from a node `n` to a node `m` with the declared property.

```
shortestPath((n1:Person)-[*..6]-(n2:Person))
```
Find a single shortest path.

```
allShortestPaths((n1:Person)-[*..6]->(n2:Person))
```
Find all shortest paths.

```
size((n)-->()-->())
```
Count the paths matching the pattern.