

Introduction to Graph Databases

Alejandro Vaisman
avaisman@itba.edu.ar

Schedule

- **Lesson 1.** 31.3.4.2025. Course presentation. Introduction NoSQL and graph data models. Property graphs vs RDF graphs. Knowledge graphs. Activity 1: Representing graphs in the relational model, and querying graphs in SQL.
- **Lesson 2.** 1.4.2025. The Neo4j graph database. Graph query languages: . Cypher. Basic and advanced queries. Activity 2: Basic Cypher queries, path computation in Cypher.
- **Lesson 3.** 3.4.2025. Advanced Cypher queries (cont.). Graph Data Science. Collaborative and content-based filtering. Activity 3. Advanced Cypher queries
- **Lesson 4.** 4.4.2025. The RDF graph data model. The semantic web. The SPARQL query language. Comparison between Cypher and SPARQL semantics.

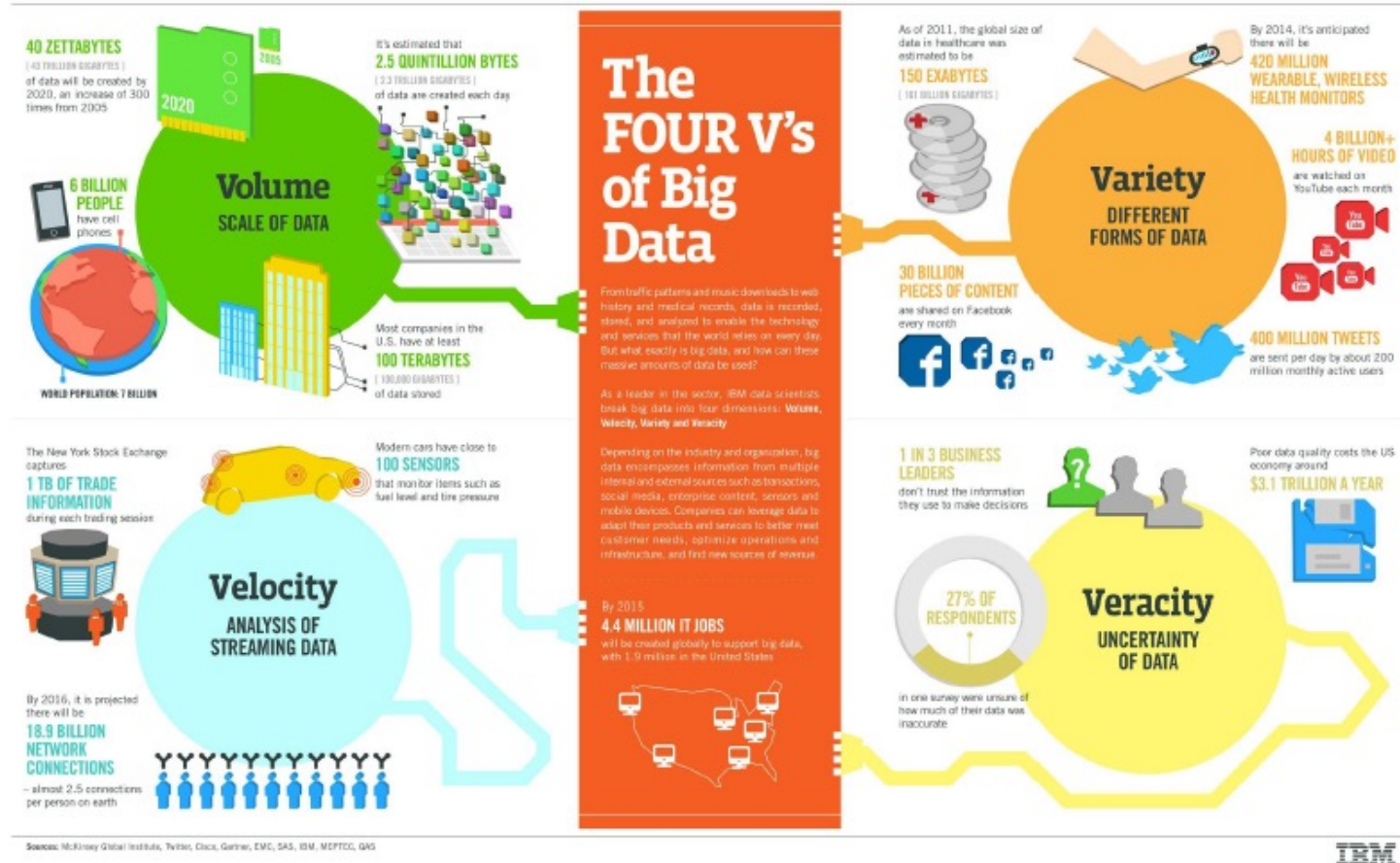
Lecture 1

NoSQL and Graph Databases

Typical BI scenario years ago ...

- **Early 90's**: Data Warehousing + Data Mining
- Big data: GB...TB!!
- Structured data
 - Mostly relational
 - Spreadsheets
 - (Some) Text
 - Web still in its infancy
- **Today**: Daily PB of data of different kinds
 - Geographic
 - Text
 - Video, image
 - Audio

Volume, Velocity, Variety, Veracity



Main characteristic of these data



A world of interrelated information

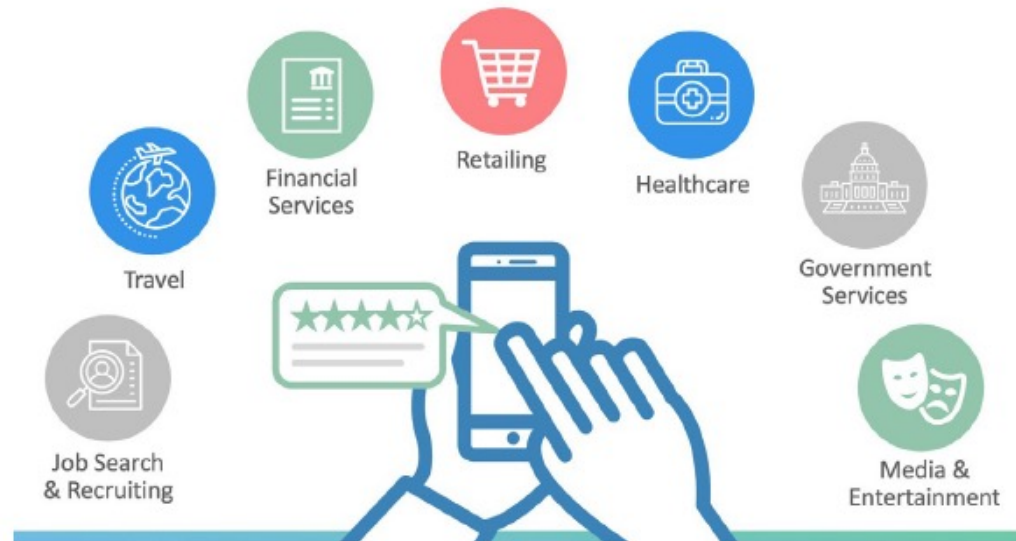
Fraud detection



<https://neo4j.com/whitepapers/top-ten-use-cases-graph-database-technology/>

A world of interrelated information

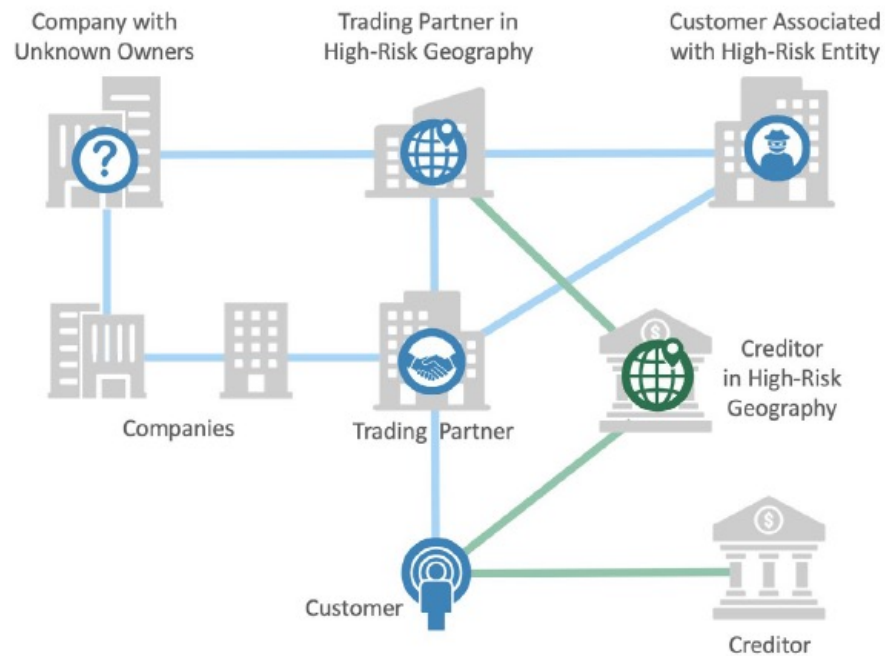
Real-time recommendation



<https://neo4j.com/whitepapers/top-ten-use-cases-graph-database-technology/>

A world of interrelated information

Anti-money laundering



<https://neo4j.com/whitepapers/top-ten-use-cases-graph-database-technology/>

How do we deal with this?

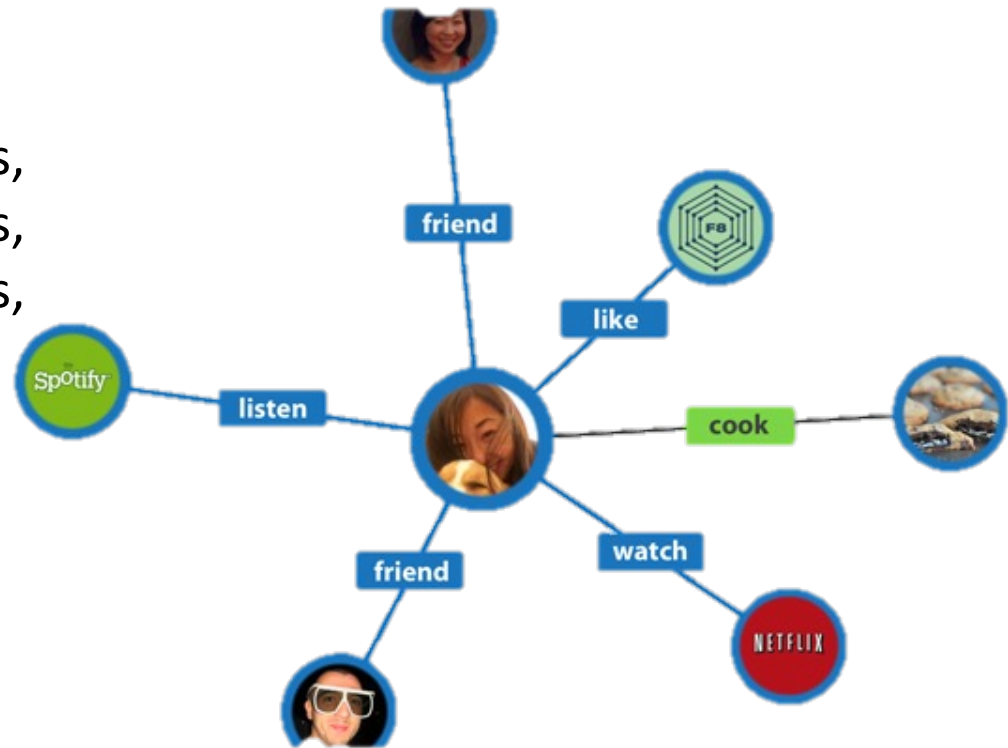
Is traditional DB technology enough?

We must address:

- Connectedness
- Unstructured data
- High Volumes
- Real-time

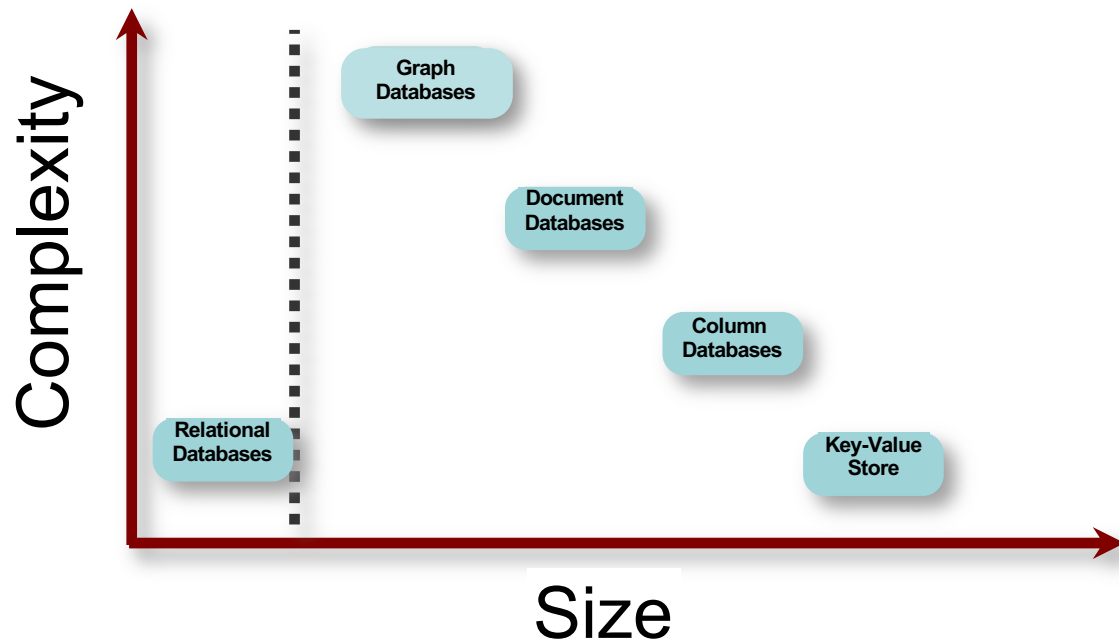
Modeling data connectedness

- A social network
- Persons, friendships, photos, locations, apps, pages, ads, interests, age range, etc.



NoSQL technologies

NoSQL databases and complexity



NoSQL Motivation

- RDBMS **too rigid** for Big Data scenarios
- Not the best to store **unstructured data**
- **One-size-fits-all approach** no longer valid in many scenarios
- RDBMS **hard to scale** for billions of rows
- Data structures used in RDBMS optimized for systems with **small amounts of memory**

NoSQL characteristics

NoSQL characteristics

- **Not** using the **relational model** for storing data

NoSQL characteristics

- **Not** using the **relational model** for storing data
- **Not** using **SQL** for retrieving data

NoSQL characteristics

- **Not** using the **relational model** for storing data
- **Not** using **SQL** for retrieving data
- **No schema**, allowing fields to be added to any record, without control

NoSQL characteristics

- **Not** using the **relational model** for storing data
- **Not** using **SQL** for retrieving data
- **No schema**, allowing fields to be added to any record, without control
- Ability to run on **clusters** of commodity hardware

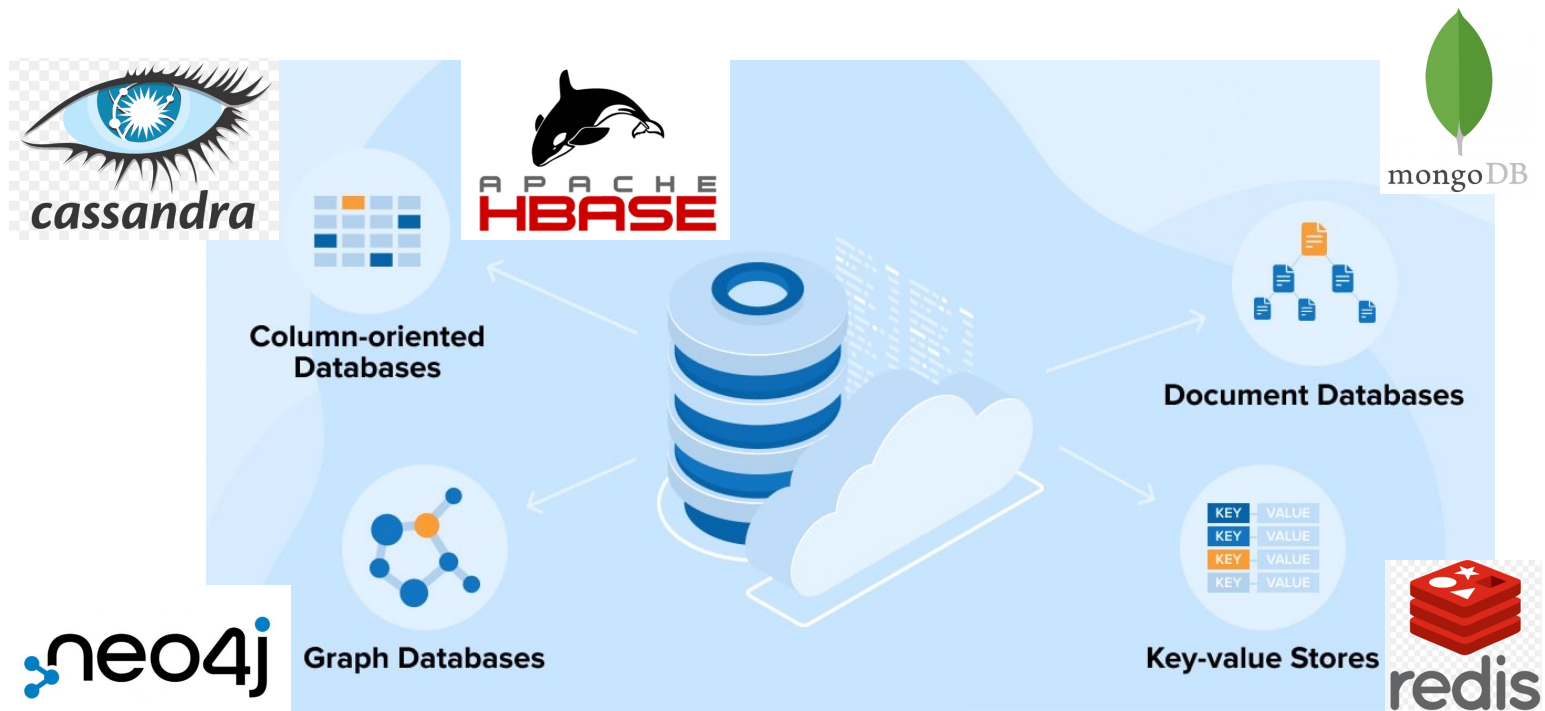
NoSQL characteristics

- **Not** using the **relational model** for storing data
- **Not** using **SQL** for retrieving data
- **No schema**, allowing fields to be added to any record, without control
- Ability to run on **clusters** of commodity hardware
- Ability to web-scale, with **horizontal scalability** in mind

NoSQL characteristics

- **Not** using the **relational model** for storing data
- **Not** using **SQL** for retrieving data
- **No schema**, allowing fields to be added to any record, without control
- Ability to run on **clusters** of commodity hardware
- Ability to web-scale, with **horizontal scalability** in mind
- Trade-off traditional consistency for other useful properties (e.g., **no ACID** support most of the time)

Types of NoSQL stores



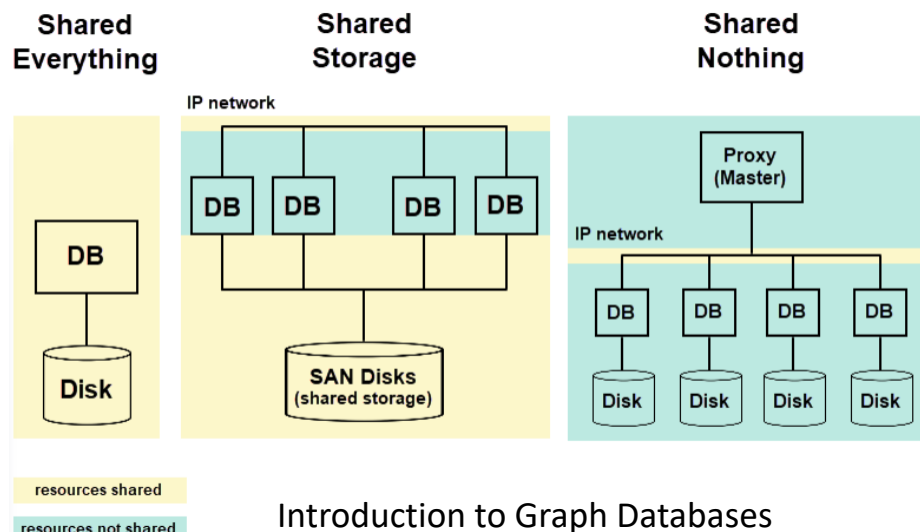
Distributed DB architectures

- Distributed database architecture: aimed at meeting availability, durability, performance, and scalability requirements
- Main mechanisms that DDS use to achieve this:
 - Replication, which places copies of data on different machines
 - Distribution, which places partitions of data on different machines

Distributed DB architectures

Architectures:

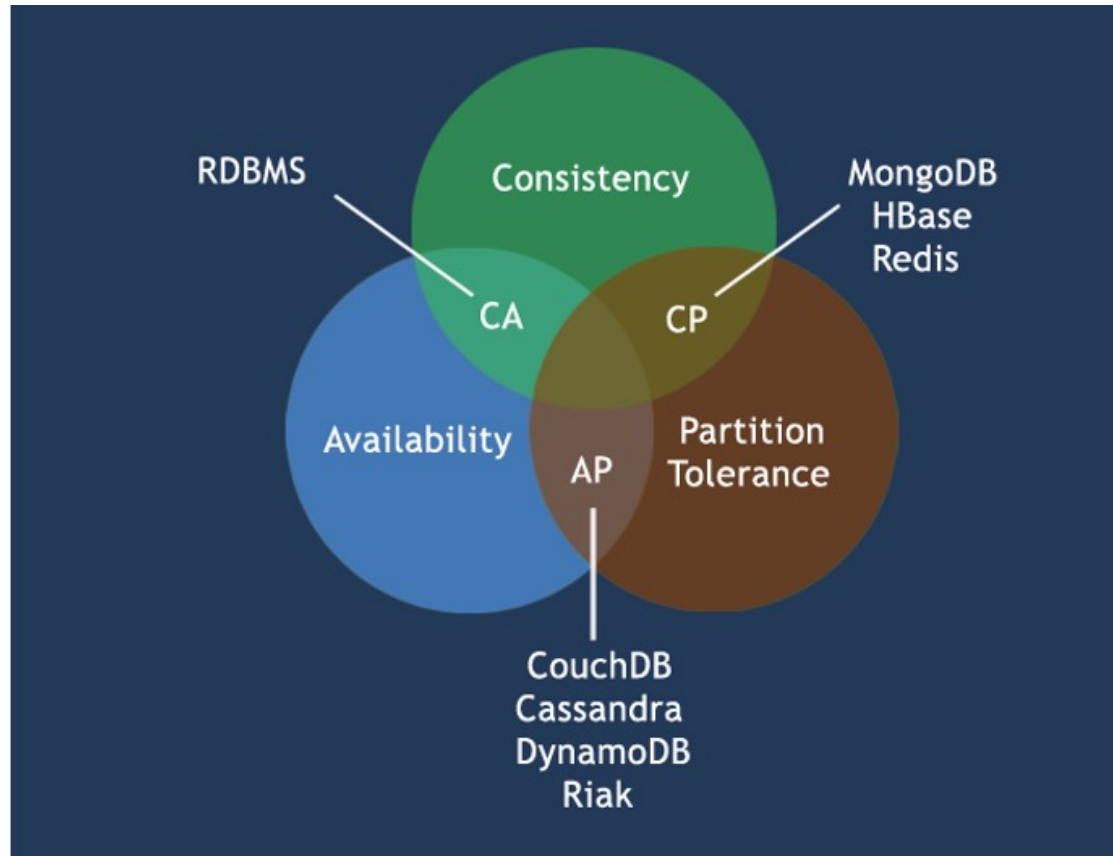
- Shared-memory: Main memory and disks shared by all processors.
- Shared-storage architecture: All nodes share the same storage devices but each node has its own private memory
- Shared-nothing architecture: Each processor in a has exclusive access to its main memory and disk and operates its own database and software



Recall the CAP theorem

- **Consistency:** Strong consistency of updates
- **Availability:** Guarantees that every request receives a response whether it succeeded or failed when retrieving data
- **Partition tolerance:** The system continues to operate despite arbitrary message lost or failure of part of the system
- **CAP Theorem:** A distributed data system cannot guarantee the three properties above, but only any combination of two of them.
- In other words, the CAP theorem states that in the presence of a network partition, one has to choose between consistency and availability.
- For example, a RDBMS can only guarantee **CA**

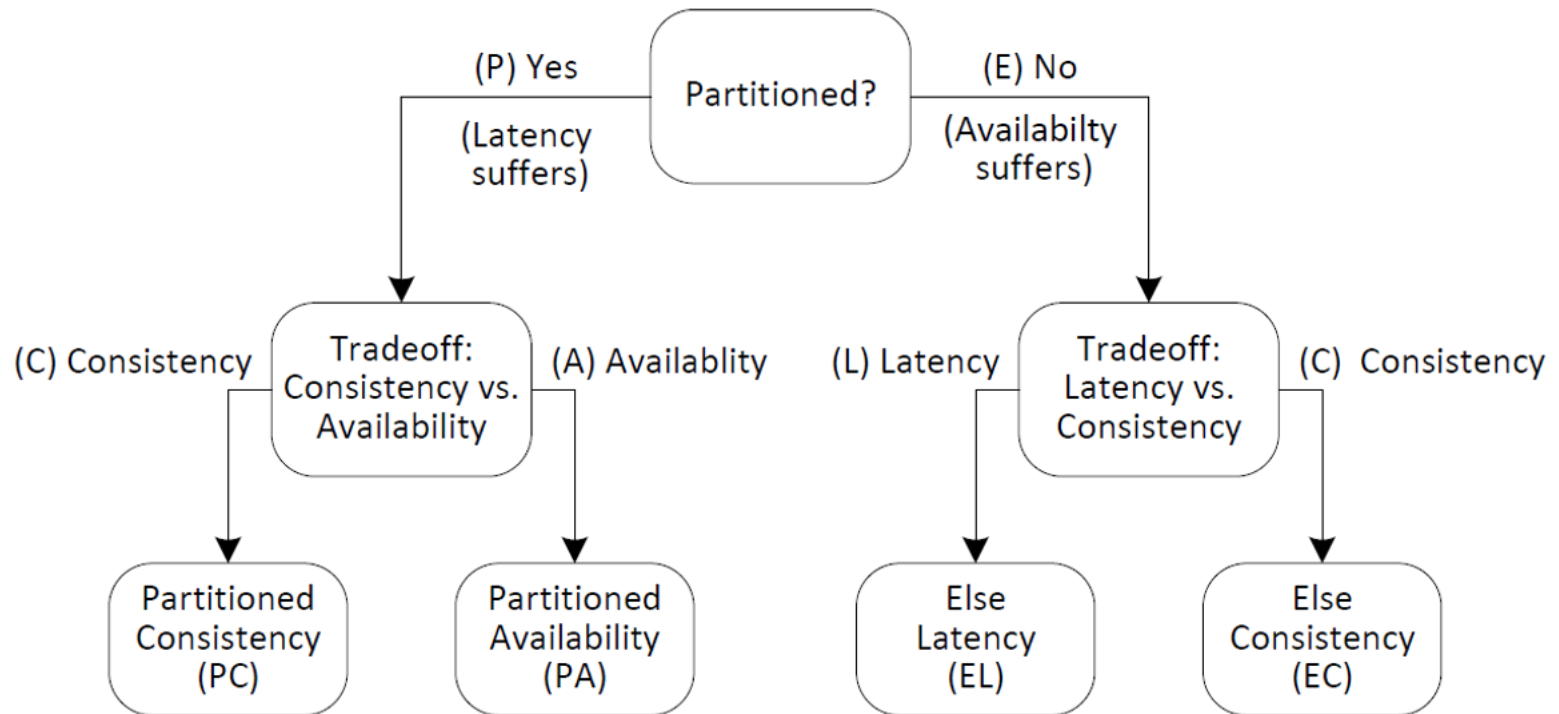
CAP theorem



PACELC theorem

- An extension to the CAP theorem
- The CAP theorem says nothing about what happens when there is **no network partition but only replication**
- The PACELC theorem addresses the choices of a distributed system when there is no partition, that is, high availability is achieved through **replication**
- In case of network partitioning (P) in a distributed computer system, we must choose between availability (A) and consistency (C) (as in the CAP theorem), but else (E), even when the system is running normally in the absence of partitions, we must choose between latency (L) and loss of consistency (C)

PACELC theorem



PACELC and databases

DDBS	P+A	P+C	E+L	E+C
Aerospike ^[8]	✓	paid only	optional	✓
Bigtable/HBase		✓		✓
Cassandra	✓		✓ ^[a]	
Cosmos DB		✓	✓ ^[b]	
Couchbase		✓	✓	✓
Dynamo	✓		✓ ^[a]	
DynamoDB		✓	✓	✓
FaunaDB ^[10]		✓	✓	✓
Hazelcast IMDG ^{[6][7]}	✓	✓	✓	✓
Megastore		✓		✓
MongoDB	✓			✓
MySQL Cluster		✓		✓
PNUTS		✓	✓	
PostgreSQL		✓		✓
Riak	✓		✓ ^[a]	
SpiceDB ^[11]		✓	✓	✓
VoltDB/H-Store		✓		✓

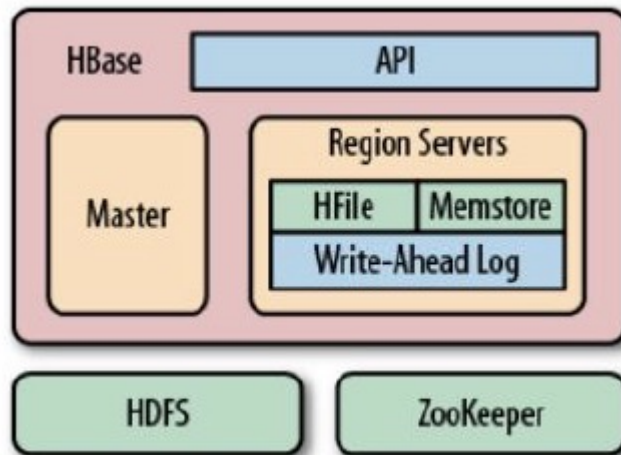
Column Stores: Apache HBASE

- Open source, distributed, scalable, consistent, low latency, random access, non-relational, column-oriented DB built over Apache Hadoop
- Based on **Google's Big Table**
- Provides NoSQL DB capabilities over Hadoop
- Supported by many companies, e.g., Cloudera
- Provides consistency & scalability
- Write-ahead log



Apache HBASE

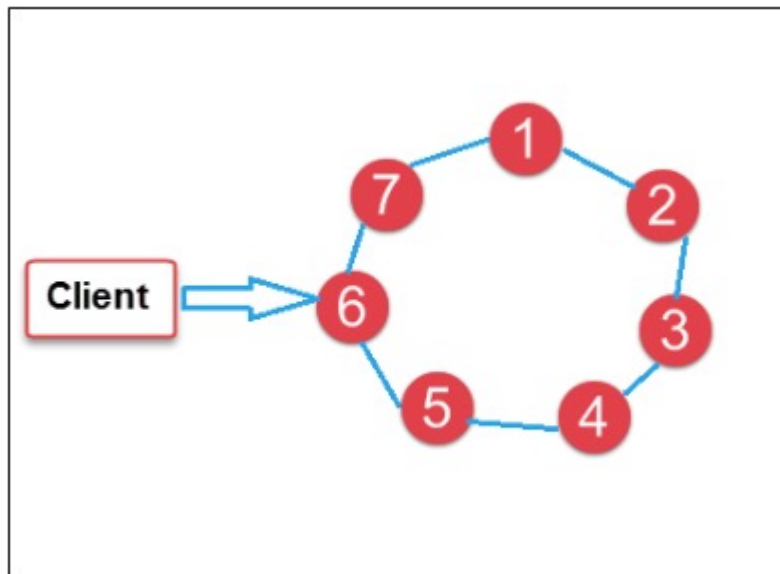
- Open source, distributed, scalable, consistent, low latency, random access, non-relational, column-oriented DB, built over Hadoop
- Based on Google's Big Table
- Provides NoSQL DB capabilities over Hadoop
- Provides consistency & scalability (CP on the CAP Theorem)



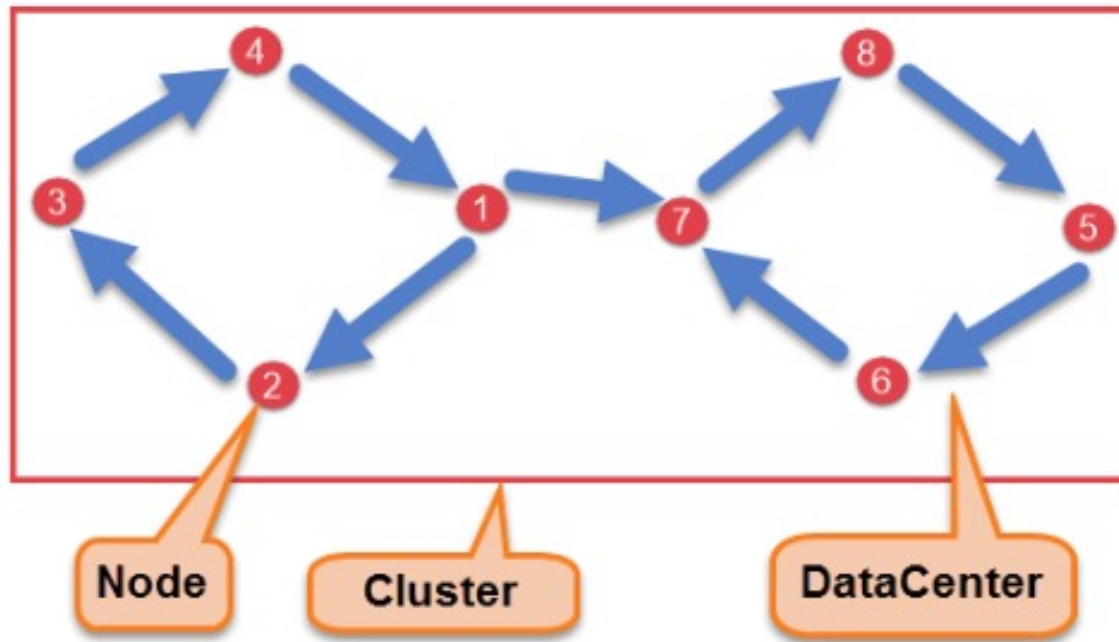
- **Master Nodes**
 - Job tracker
 - Hbase Master
 - Located in the HDFS's **NameNode** of the cluster
- **Slave Nodes**
 - Task tracker
 - Hbase RegionServer
 - Located in the HDFS's **DataNodes**
- **Zookeeper**

Apache Cassandra

- Decentralized (every node in the cluster has the same role)
- Linear scalability of R/W increasing the cluster size
- Provides consistency & scalability (CP on the CAP Theorem)



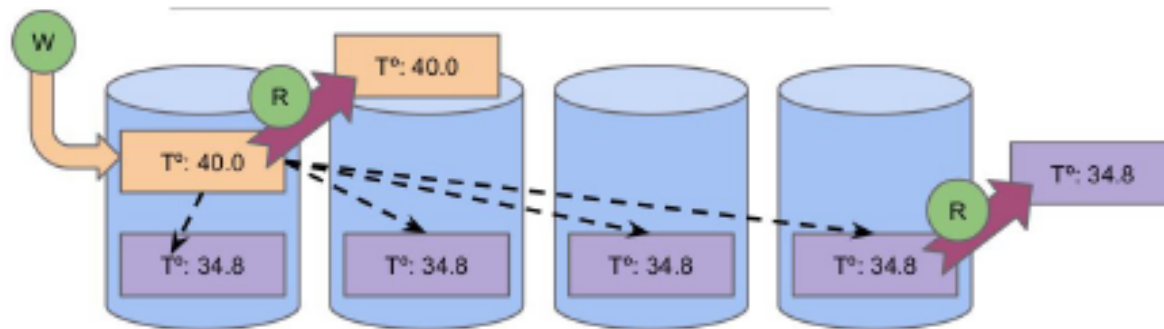
Apache Cassandra



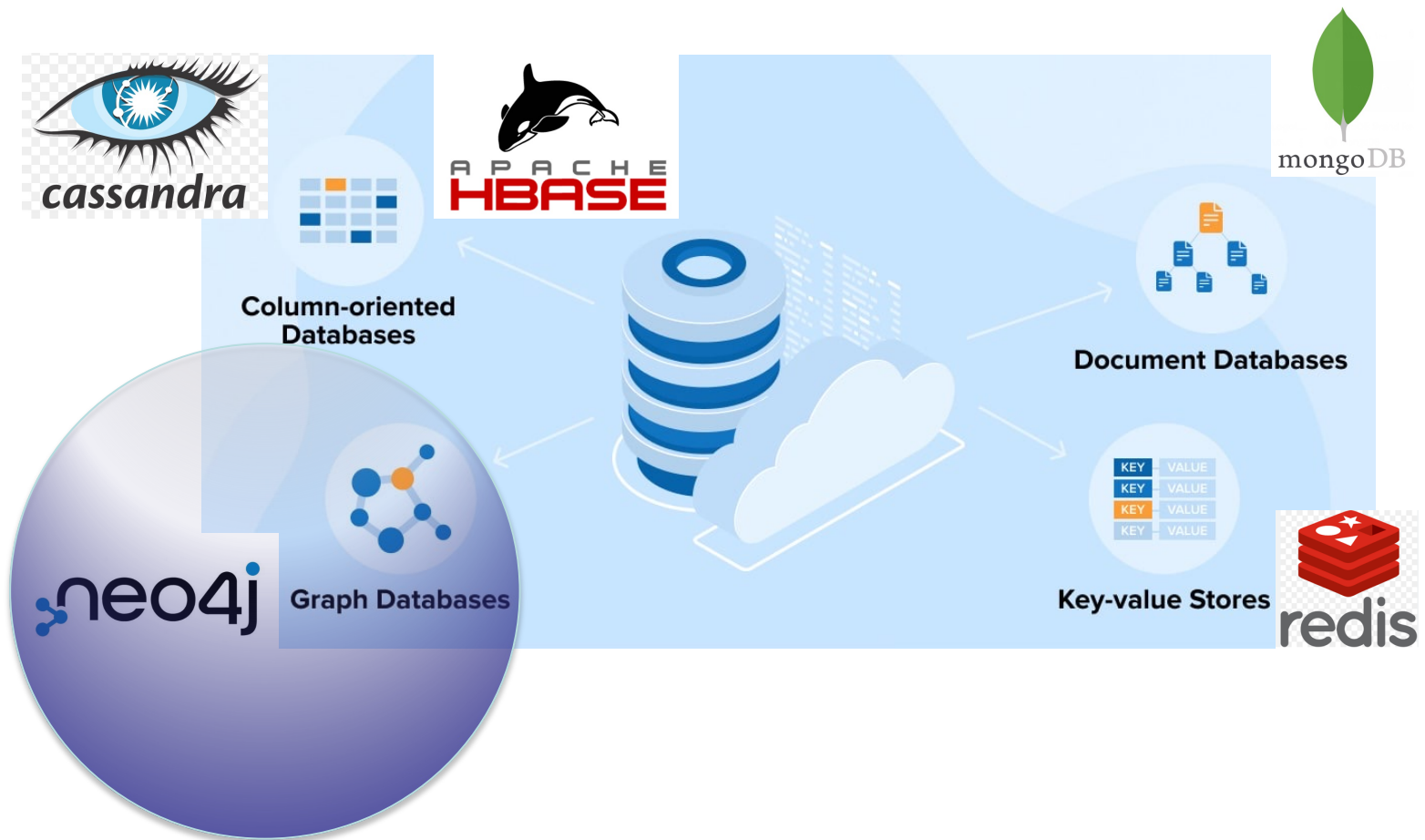
- **Node** The place where data are stored. Basic component of Cassandra.
- **Rack** A collection of nodes
- **Data Center** A collection of racks.
- **Cluster** A collection of data centers.
- **Mem-Table** After writing in Commit log, data are written in Mem-table (temporarily).
- **SSTable** When Mem-table reaches a certain threshold, data is flushed to an SSTable disk file.

Eventual consistency

- **Assumption:** in the absence of new writes, **eventually** consistency will be achieved, and all replicas that are responsible for a data item will agree on the same version and return the last updated value.
- With fewer replicas, R/W operations complete more quickly, lowering latency.

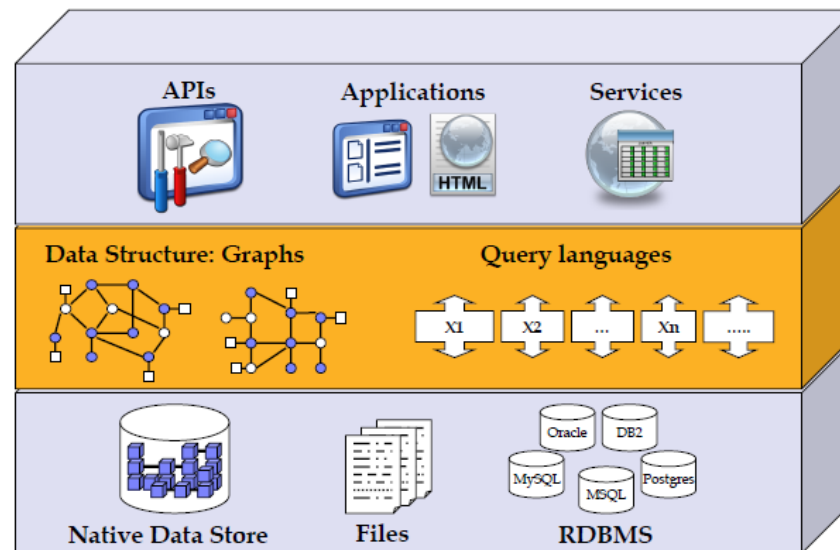


In the remainder....



Graph Database Models

A Possible Architecture



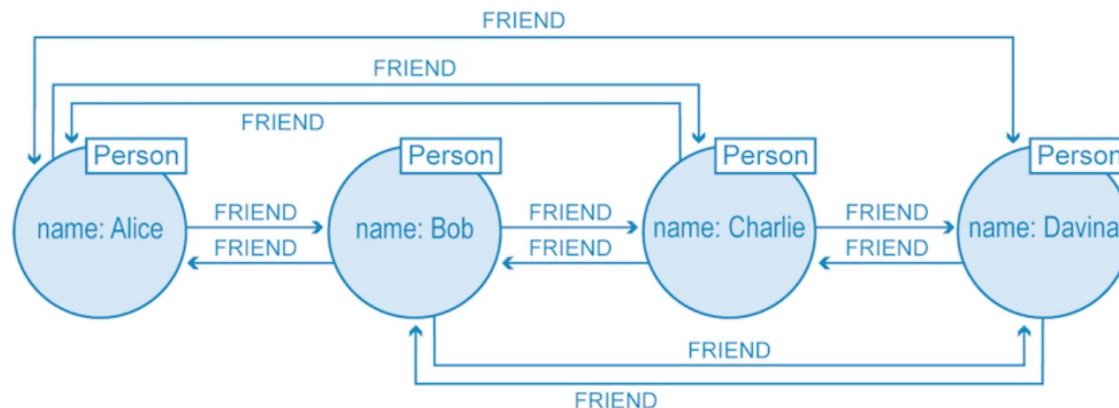
Graph database models*

- **Database model:** three components: a set of data structure types, a set of operators or inference rules, and a set of integrity rules (Codd, 1980)
- *Data and/or the schema represented by graphs, hypergraphs, hypernodes*
Node -> entity, edge -> relationship between entities, property -> feature
- *Data manipulation expressed by graph transformations, or operations on graph features: paths, neighborhoods, subgraphs, patterns, connectivity, graph statistics (e.g., diameter, centrality, etc.)*
- *Integrity constraints enforce data consistency, schema-instance consistency, identity & referential integrity, functional dependencies. E.g.: labels w/ unique names, constraints on nodes, domain and range of properties*

* C. Gutiérrez, R. Angles. A Survey on Graph Database Models ACM Computing Surveys, 2008

Native vs. Non-native graph databases

- **Native** graph storage have underlying storage designed specifically for the storage and management of graphs. They are designed to maximize the speed of traversals during arbitrary graph algorithms (adjacency lists, double linked lists, etc.)
- In a native graph database, a node record just points to structured lists of relationships, labels, and properties



<https://neo4j.com/blog/cypher-and-gql/native-vs-non-native-graph-technology/>

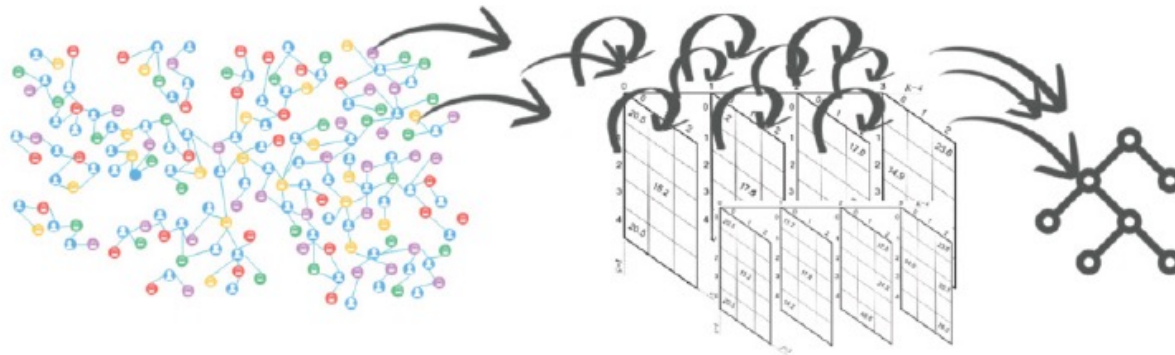
Native vs. Non-native graph databases

- **Non-native** graph storage uses a relational database, a columnar database, or some other general-purpose data store rather than being specifically engineered for the requirements of graphs. They would store nodes and relationships in their native data paradigm (e.g., as tables or documents).
- Non-native graph databases are not optimized for storing graphs
- Non-native graph databases use many types of indexes to compute the joins to link entities together
- Reversing the direction of a traversal is more expensive
 - Give me the friends of A and E: easy
 - Give me the persons who have D as friend: + expensive (inverse index)

Person	Friend
A	B
A	C
A	D
E	F
E	G
E	D

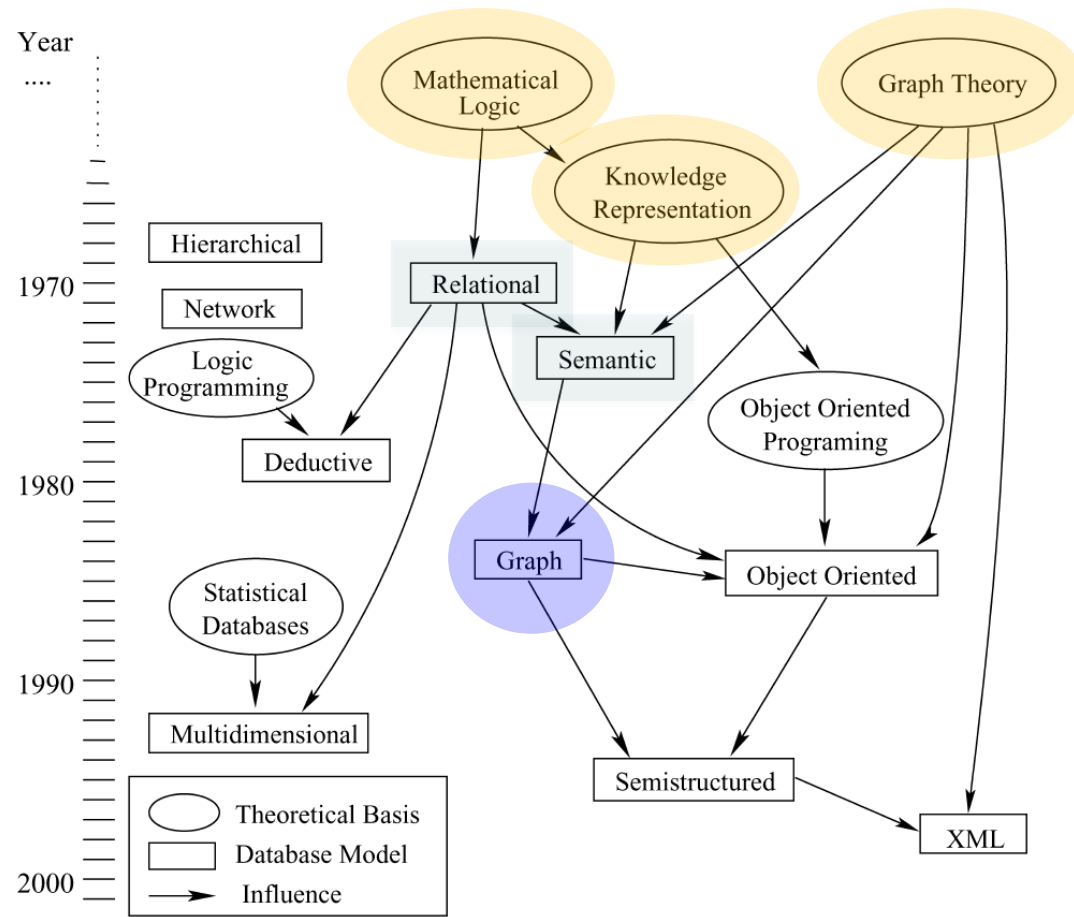
Exploiting connectedness with graphs: Native vs. Non-native GDB

- **Querying** graph databases with high-level query languages (Cypher, GraphQL, Gremlin)
- **Graph data science:** a new emerging field for predictive analytics and machine learning over graph data
 - Graph analysis helps in representing contexts, which are crucial in predictive analysis
 - Knowledge graphs, used for knowledge representation
 - **Native graph db's avoid going from graphs (real world) to tables to, e.g., decision trees**

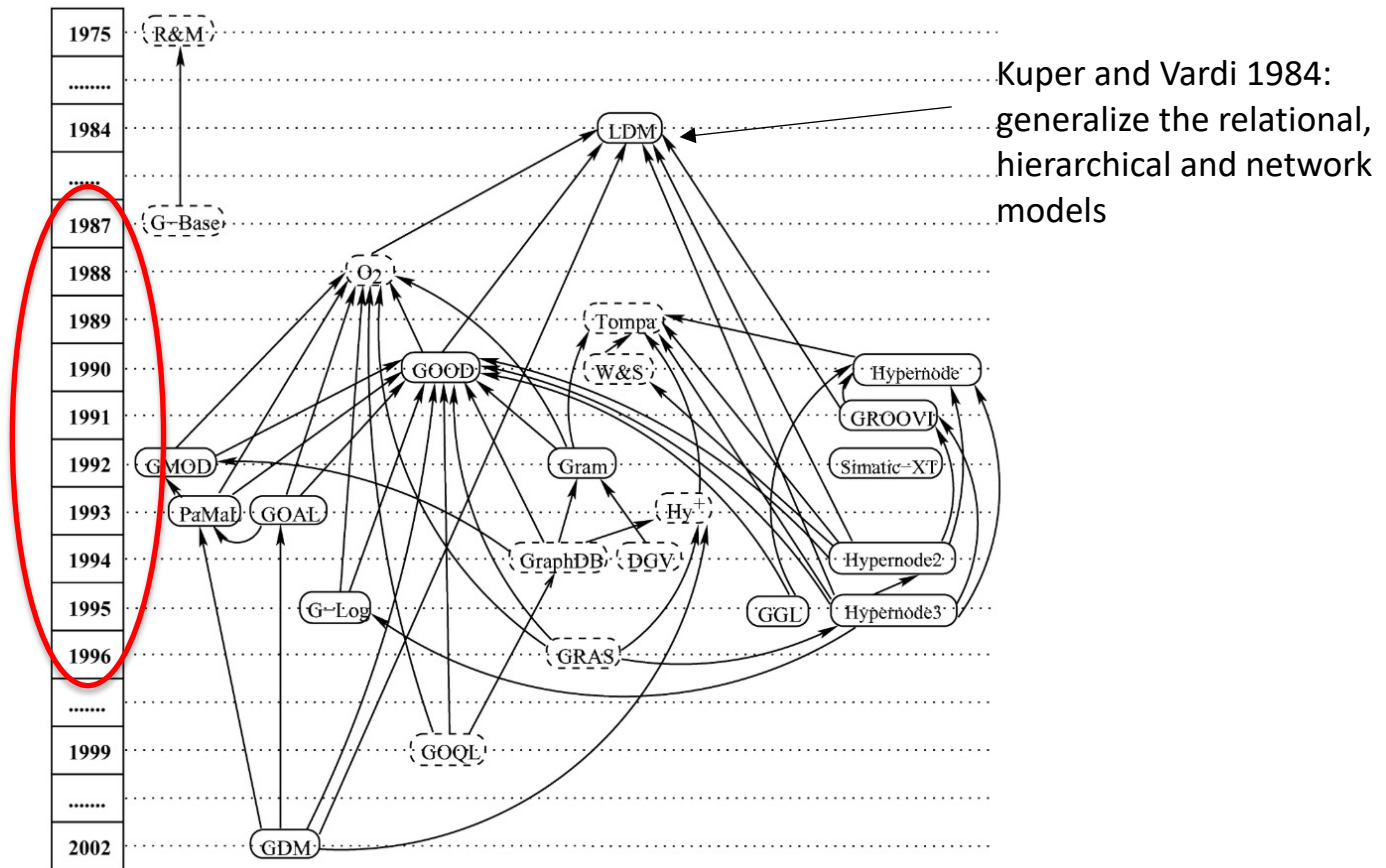


- **Graph visualization** for ad hoc data discovery and exploration



A history of database models (A. Mendelzon)



The Golden age of GDB

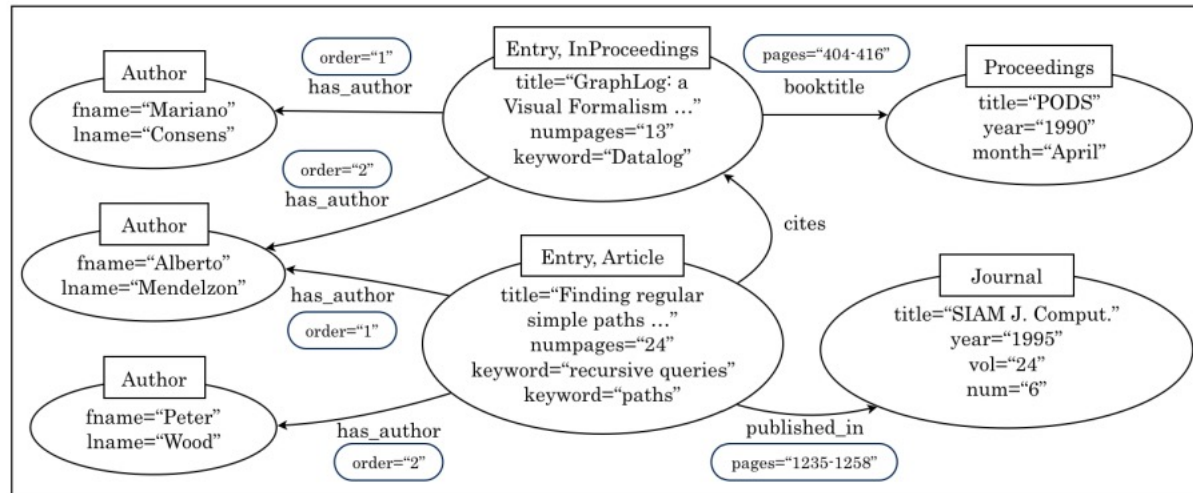


Two graph data models

Linked Data Knowledge Graph	RDF Knowledge Graph <ul style="list-style-type: none">• Data Federation• Knowledge Representation• Metadata Management		<ul style="list-style-type: none">▪ Life Sciences▪ Health Care▪ Publishing▪ Finance
Graph Analytics	Property Graph <ul style="list-style-type: none">• Path Analytics• Social Network Analysis• Entity Analytics		<ul style="list-style-type: none">▪ Financial▪ Retail, Marketing▪ Social Media▪ Smart Manufacturing
Use Case	Graph Model	Industry Domain	

- Perry, M. Introduction to RDF Graph for Oracle Database 19c. Architecture and Overview (2019)

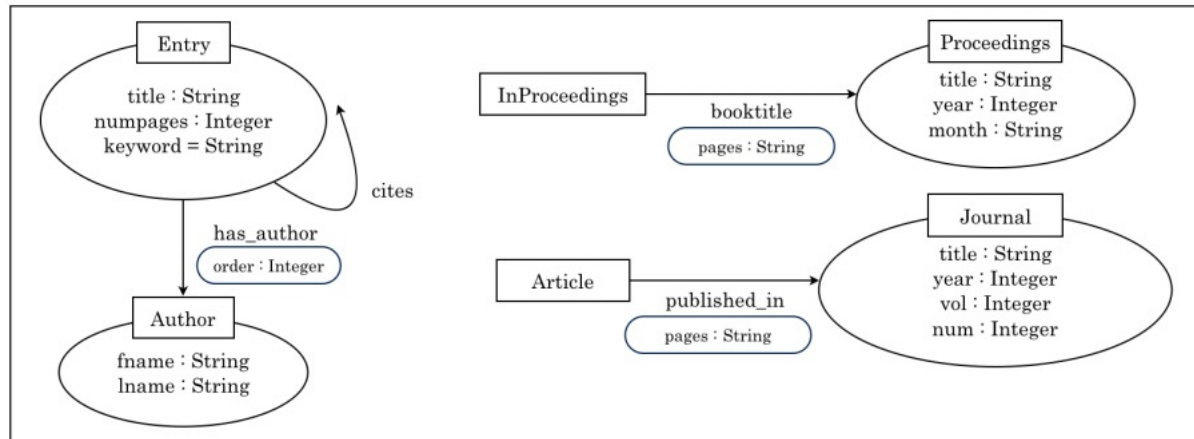
The property graph data model*



- First formal definition (Angles, 2018)

1. N is a finite set of nodes (also called vertices);
2. E is a finite set of edges such that E has no elements in common with N ;
3. $\rho : E \rightarrow (N \times N)$ is a total function that associates each edge in E with a pair of nodes in N (i.e., ρ is the usual incidence function in graph theory);
4. $\lambda : (N \cup E) \rightarrow \text{SET}^+(L)$ is a partial function that associates a node/edge with a set of labels from L (i.e., λ is a labeling function for nodes and edges);
5. $\sigma : (N \cup E) \times P \rightarrow \text{SET}^+(V)$ is a partial function that associates nodes/edges with properties, and for each property it assigns a set of values from V .

The property graph data model*



- Schema

1. $T_N \subset \mathbf{L}$ is a finite set of labels representing node types;
2. $T_E \subset \mathbf{L}$ is a finite set of labels representing edge types, satisfying that T_E and T_N are disjoint;
3. $\beta : (T_N \cup T_E) \times \mathbf{P} \rightarrow \mathbf{T}$ is a partial function that defines the properties for node and edge types, and the datatypes of the corresponding values;
4. $\delta : (T_N, T_N) \rightarrow \text{SET}^+(T_E)$ is a partial function that defines the edge types allowed between a given pair of node types.

Querying graphs example: Social Network

“path” performance

- All old experiment:
 - ~1k persons
 - Average 50 friends per person
 - `pathExists(a,b)` limited to depth 4
 - Caches warm to eliminate disk IO

	# persons	query time
Relational database	1000	2000ms

Social Network “path” performance

- AI old experiment:
 - ~1k persons
 - Average 50 friends per person
 - `pathExists(a,b)` limited to depth 4
 - Caches warm to eliminate disk IO

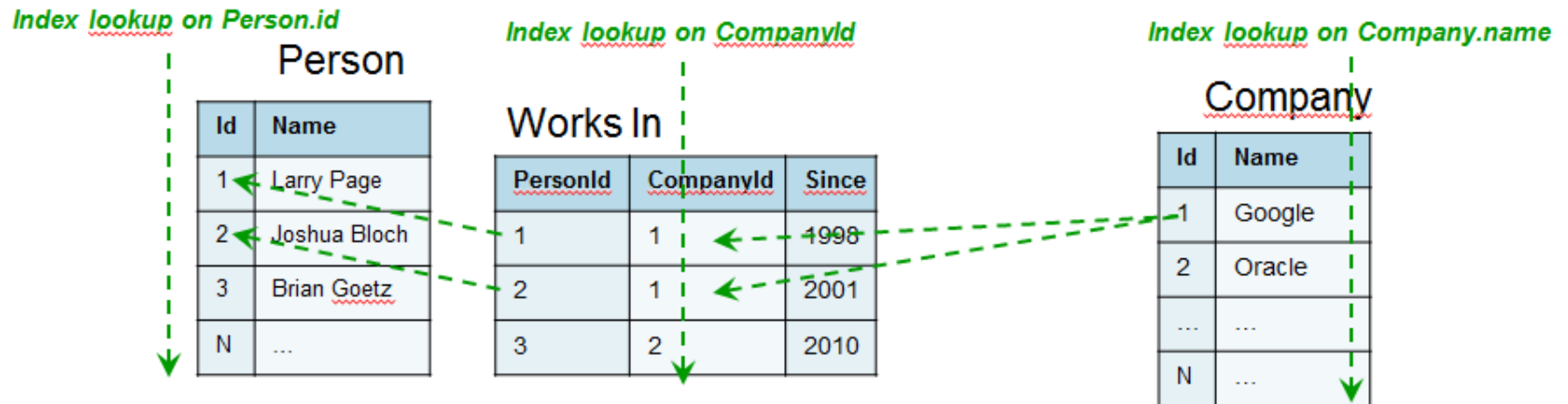
	# persons	query time
Relational database	1000	2000ms
Neo4j	1000	2ms

Social Network “path” performance

- AI old experiment:
 - ~1k persons
 - Average 50 friends per person
 - `pathExists(a, b)` limited to depth 4
 - Caches warm to eliminate disk IO

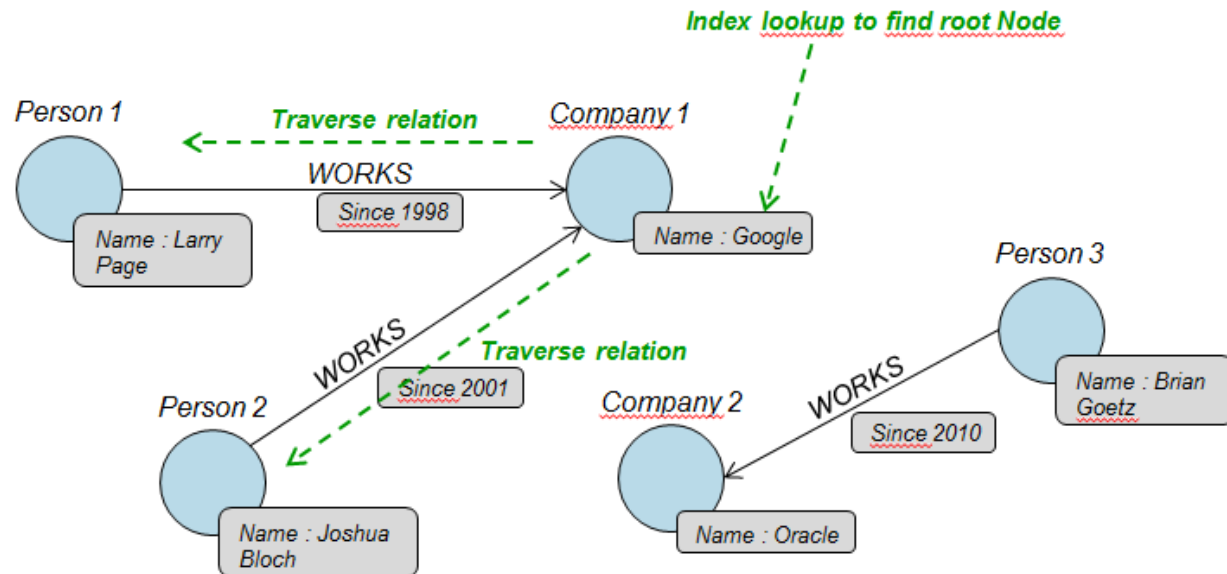
	# persons	query time
Relational database	1000	2000ms
Neo4j	1000	2ms
Neo4j	1000000	2ms

A typical SQL query



*Select Person.Name
from Person, Company, WorksIn
where Company.name='Google'
and WorksIn.CompanyId = Company.Id
and WorksIn.PersonId = Person.Id*

Same query on graphs



The deepest the navigation, the largest the difference with RDBs

Traversing data in a RDBMS

- Based on joining and selecting data

User

User	Address	Phone	Email	AltEmail
User1	Address 1	ph1	em1	emalt1
User2	Address 2	ph2	em2	emalt2
...

```
SELECT *  
FROM User u, UserOrder uo, OrderItem oi, Item I  
WHERE u.user = uo.user AND uo.orderId = oi.orderId  
      AND i.linItemId = oi.linItemId AND  
      u.user = "User1"
```

User = 5.000.000
UserOrder = 100.000.000
OrderItem = 1.000.000.000
Items = 35.000

Can you estimate the query cost?

User	OrderId
User1	3644
User1	3645
...	..

UserOrder

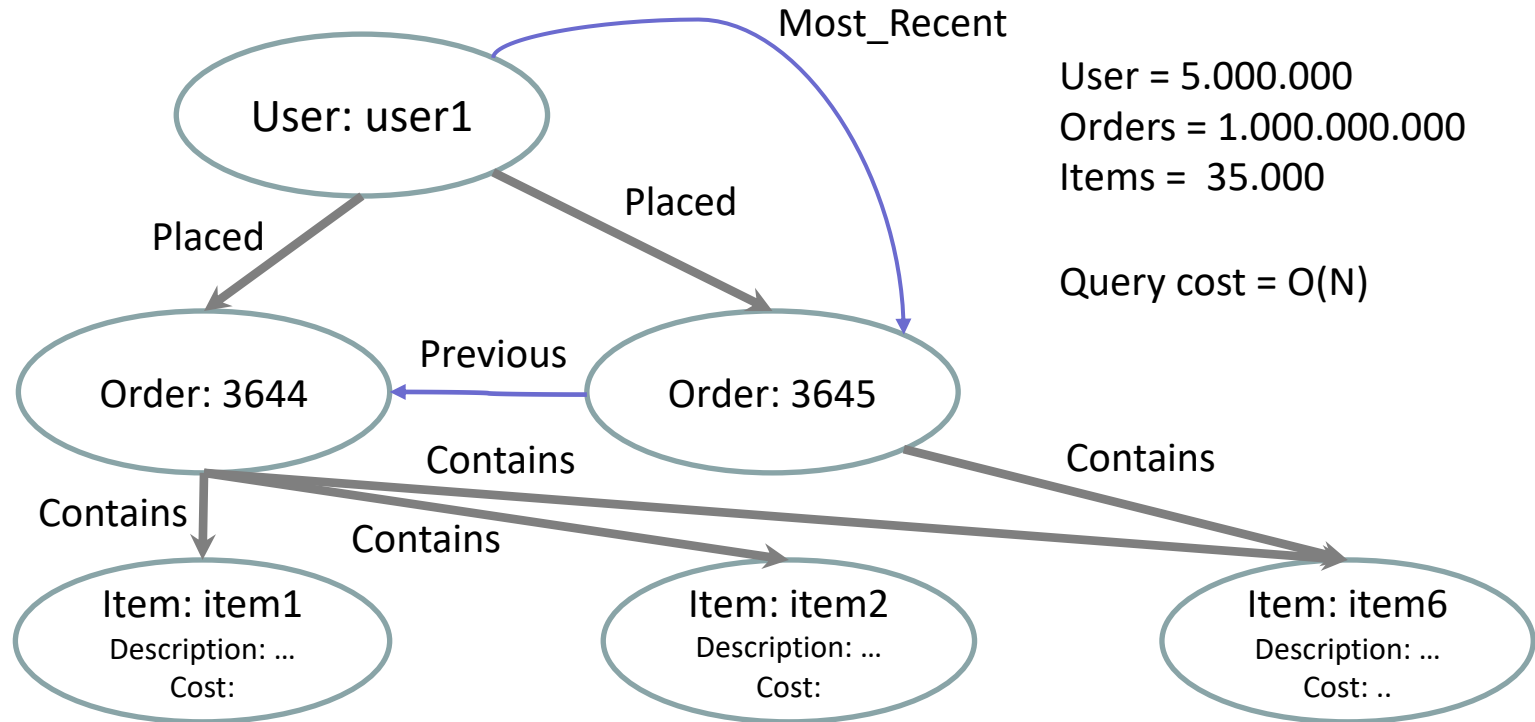
OrderId	LinItemId
3644	item1
3644	item2
3644	item6
3645	item6
...	..

OrderItem

ItemId	Description	UCost
item1	dcs1	42
Item2	dsc2	45
Item3	dsc3	74
item4	dsc4	43
...

Item

Traversing data in a GDB

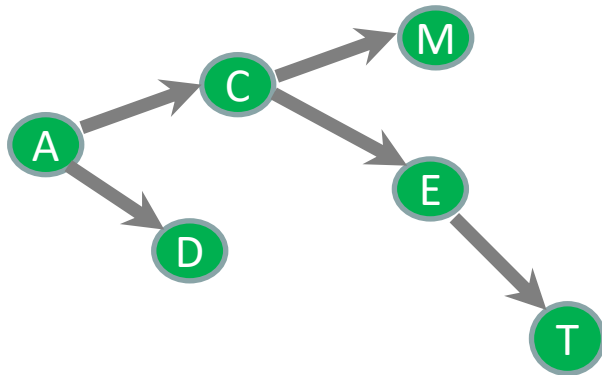


Recursive queries

We want to compute the closure of a relation “ReportsDirectlyTo”, that is, to whom someone “ReportsTo”, either directly or indirectly

These queries are normally more expensive in the Relational Model, since they imply MULTIPLE JOINS. Joins are expressed at the schema level rather than at the instance level.

How would we represent this in a Graph Data Model?



ReportsDirectlyTo	
Boss	Employee
A	C
A	D
C	E
C	M
E	T

ReportsTo		
Boss	Employee	
A	C	Iteration 1
A	D	
C	E	
C	M	
A	E	Iteration 2
A	M	
C	T	
A	T	Iteration 3

ReportsDirectlyTo	
Boss	Employee
A	C
A	D
C	E
C	M
E	T

We want to compute the closure of the relation “ReportsDirectlyTo”, that is, to whom someone reports (“ReportsTo”), either directly or indirectly. SQL supports these kinds of recursive queries. Recursively joining ReportsTo and ReportsDirectlyTo on RT.Employee=RDT.Boss

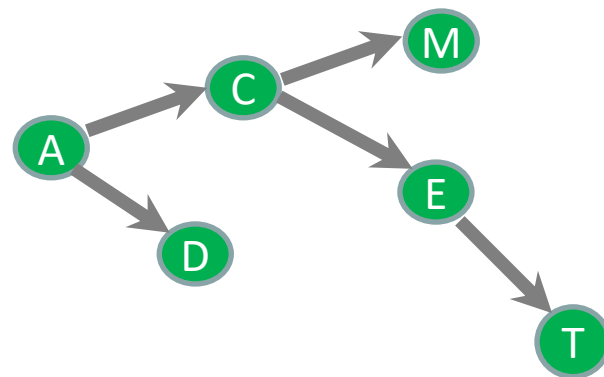
```
WITH recursive ReportsTo(Boss, Employee) AS
  (SELECT Boss, Employee
   FROM   ReportsDirectlyTo
   UNION
   SELECT ReportsTo.Boss, ReportsDirectlyTo.Employee
   FROM   ReportsTo, ReportsDirectlyTo
   WHERE  ReportsTo.Employee = ReportsDirectlyTo.Boss )
SELECT * FROM ReportsTo
```

These queries are normally more expensive in the Relational Model, since they imply MULTIPLE JOINS.

Joins are expressed at the schema level rather than at the instance level.

How would we represent this in a Graph Data Model?

ReportsDirectlyTo	
Boss	Employee
A	C
A	D
C	E
C	M
E	T



Paths in a graph are expressed at the instance level (there is no schema). Just check if there is an outgoing edge.

Recursive queries

Ejemplo

```
CREATE TABLE dependedirectamentede(jefe char(1), empleado char(1))
```

```
insert into dependedirectamentede Values ('A','C');
```

```
insert into dependedirectamentede Values ('A','D');
```

```
insert into dependedirectamentede Values ('C','E');
```

```
insert into dependedirectamentede Values ('C','M');
```

```
insert into dependedirectamentede Values ('E','T');
```

```
select * from dependedirectamentede
```

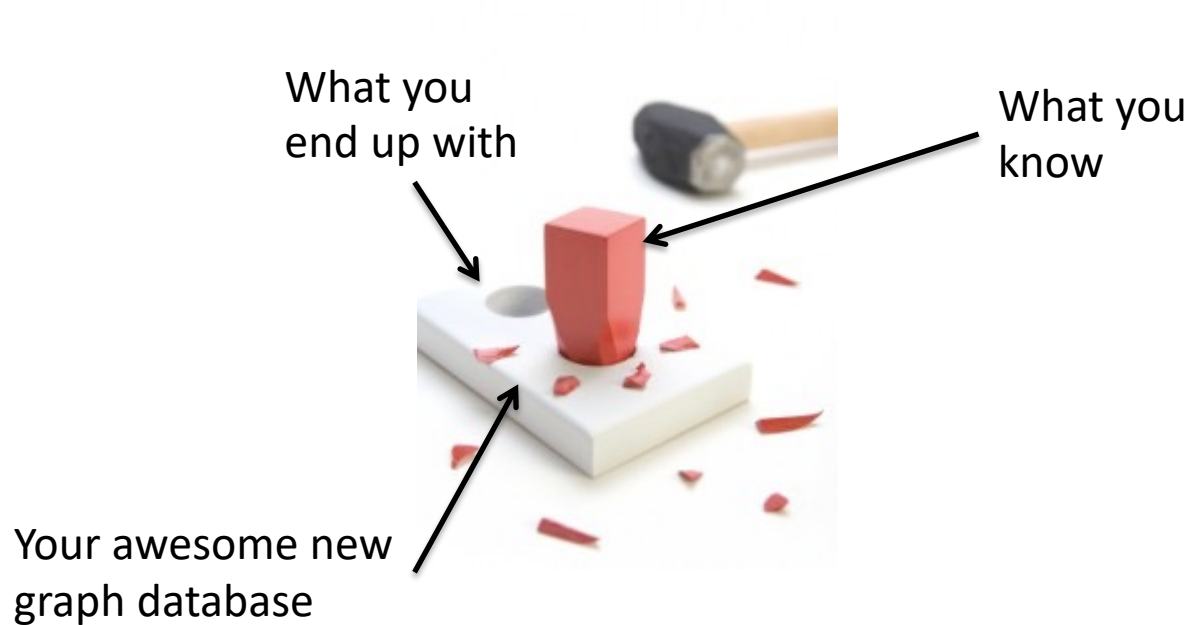
DependeDirectamenteDe	
Jefe	Empleado
A	C
A	D
C	E
C	M
E	T

DependeDe		
Jefe	Empleado	
A	C	Iteración 1
A	D	
C	E	
C	M	
E	T	
A	E	Iteración 2
A	M	
C	T	
A	T	Iteración 3

De esta relación que representa “DependeDirectamenteDe” quiero calcular la clausura, o sea quien “DependeDe” en forma directa o indirecta. SQL tiene previsto este tipo de consulta recursivas.

```
WITH recursive DependeDe(Jefe, Empleado) AS
    (SELECT Jefe, Empleado
     FROM DependeDirectamenteDe
    UNION
     SELECT calculado.jefe, origen.empleado
     FROM DependeDe as calculado, DependeDirectamenteDe as origen
     WHERE calculado.empleado = origen.jefe )
SELECT * FROM DependeDe
```

Forcing the relational model in a graph

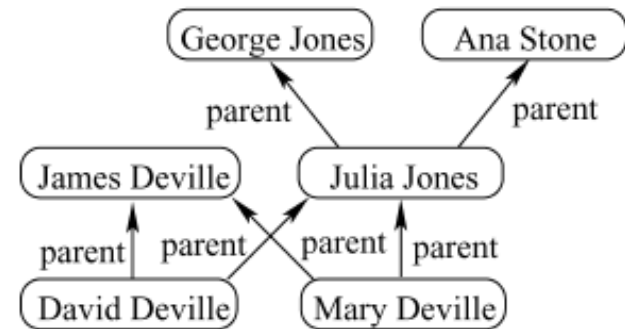


Think in terms of nodes and edges!

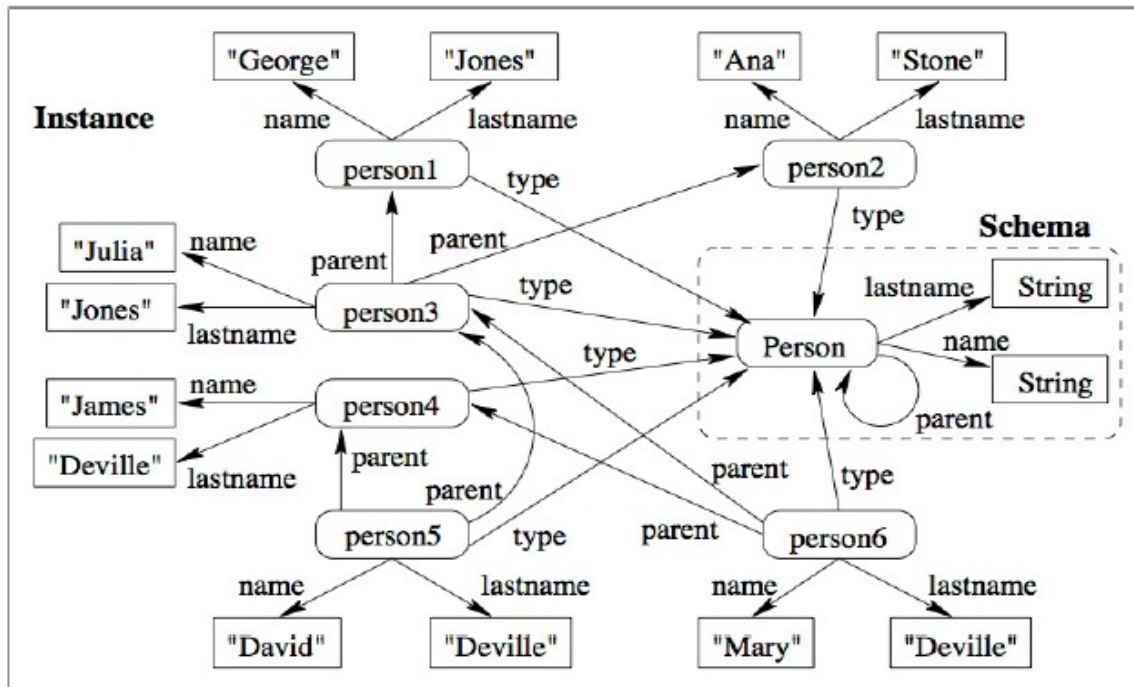
The RDF Data Model

Example: genealogy and data diagram

NAME	LASTNAME	PERSON	PARENT
George	Jones	Julia	George
Ana	Stone	Julia	Ana
Julia	Jones	David	James
James	Deville	David	Julia
David	Deville	Mary	James
Mary	Deville	Mary	Julia

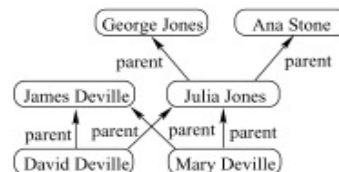


Genealogy – RDF model



- Not oriented explicitly to model connectivity.
- Originally devised to represent metadata.
- Represents resources and relations between resources.
- No assumption of the application domain.
- An RDF graph is a collection of (subject, predicate, object) triples

NAME	LASTNAME	PERSON	PARENT
George	Jones	Julia	George
Ana	Stone	Julia	Ana
Julia	Jones	David	James
James	Deville	David	Julia
David	Deville	Mary	James
Mary	Deville	Mary	Julia



RDF Model

- **RDF** allows to express facts: Ana is the mother of Julia
- But we'd like to be able to express more generic knowledge, e.g.,
 - If somebody has a daughter, then that person is a parent
 - All textbooks are books
- This kind of knowledge is often called *schema* knowledge or *terminological* knowledge.
- **RDF Schema** allows us to do some schema knowledge modeling, OWL gives even more expressivity

RDF classes & instances

- Classes stand for sets of things. In RDF: Sets of URIs.
- `book:uri` (a textbook) is a member of the class `ex:Textbook`

```
book:uri    rdf:type    ex:Textbook .
```

- An URI can belong to several classes (uses the **predefined** predicate type)

```
book:uri    rdf:type    ex:Textbook .  
book:uri    rdf:type    ex:WorthReading .
```

- Classes can be arranged in hierarchies: each textbook is a book

```
ex:Textbook  rdfs:subClassOf  ex:Book .
```

- `Type`, `subClassOf`, `subPropertyOf`, etc., have a **predefined semantics**. This is a key difference with the property graph data model

RDF pre-defined classes

- Every URI denoting a class is a member of `rdfs:Class`

```
ex:Textbook    rdf:type    rdfs:Class .
```

- This also makes `rdfs:Class` a member of `rdfs:Class`

```
rdfs:Class    rdf:type    rdfs:Class .
```

- `rdfs:Resource` (class of all URIs)
- `rdf:Property` (class of all properties)
- `rdf:XMLLiteral`
- `rdfs:Literal` (each datatype is a subclass)
- `rdf:Bag`, `rdf:Alt`, `rdf:Seq`, `rdfs:Container` , `rdf:List`
`rdfs:ContainerMembershipProperty`
- `rdfs:Datatype` (contains all datatypes – a class of classes)
- `rdf:Statement`

RDF implicit knowledge (Inference)

If an RDFS document contains

```
u    rdf:type    ex:Textbook .
```

and

```
ex:Textbook    rdfs:subClassOf    ex:Book .
```

then

```
u    rdf:type    ex:Book .
```

is *implicitly* also the case: it's a *logical consequence*. (We can also say it is *deduced* (deduction) or *inferred* (inference). We do not have to state this explicitly. Which statements are logical consequences is governed by the formal semantics.

RDF implicit knowledge (cont.)

As another example, from

```
ex:Textbook    rdfs:subClassOf    ex:Book .
```

and

```
ex:Book        rdfs:subClassOf    ex:PrintMedia .
```

then

```
ex:Textbook    rdfs:subClassOf    ex:PrintMedia .
```

Therefore, `rdfs:subClassOf` is a *transitive relation*.

RDF implicit knowledge (cont.)

Property hierarchies

From

```
ex:isHappilyMarriedTo  rdf:subPropertyOf  ex:isMarriedTo.
```

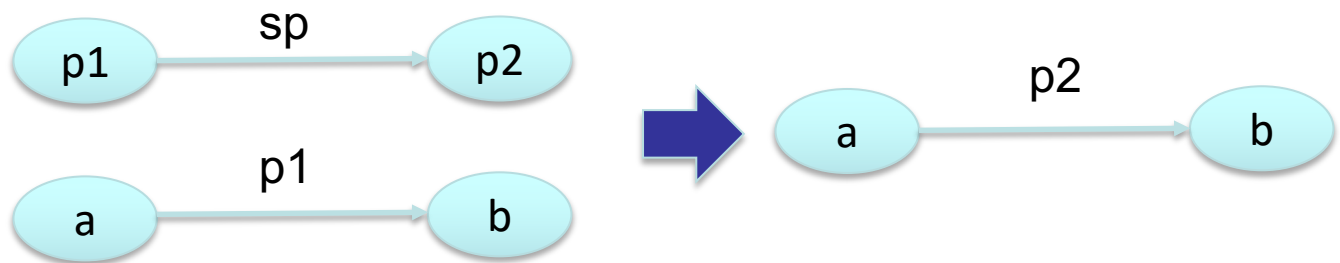
and

```
ex:markus  ex:isHappilyMarriedTo  ex:anja .
```

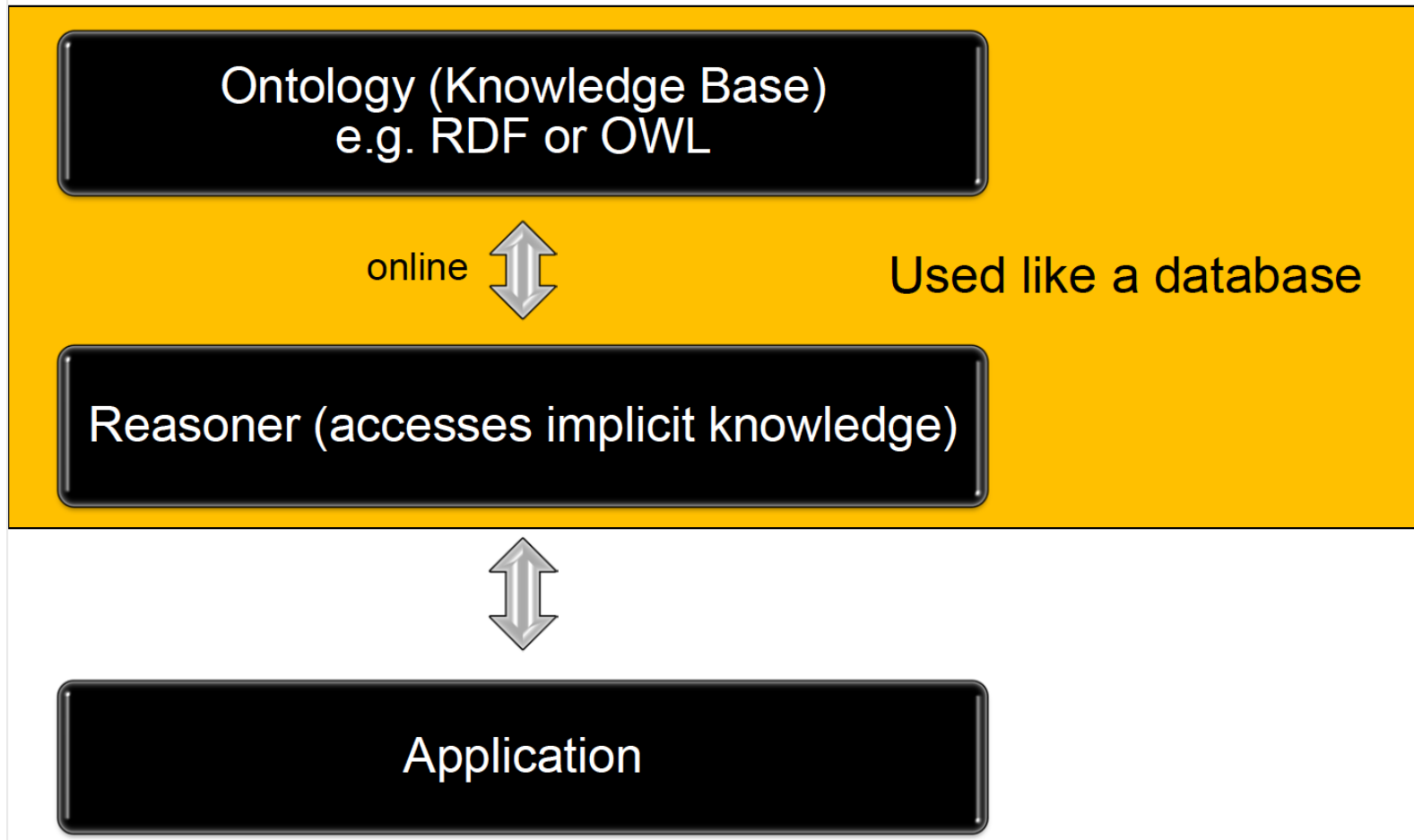
then

```
ex:markus  ex:isMarriedTo  ex:anja .
```

Thus, if p_1 is subproperty of p_2 , and a p_1 b , then a p_2 b



Using implicit knowledge



Knowledge graphs*

- Many (sometimes conflicting) definitions, technical and general
- Knowledge graph: a graph of data (or data graph) intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities
- The graph of data conforms to a graph-based data model (e.g., a directed edge-labelled graph, a property graph, etc.)
- Knowledge: something that is known, which may be accumulated from external sources, or extracted from the KG itself
- Two types of KG in practice: open knowledge graphs (e.g., Dbpedia) and enterprise knowledge graphs

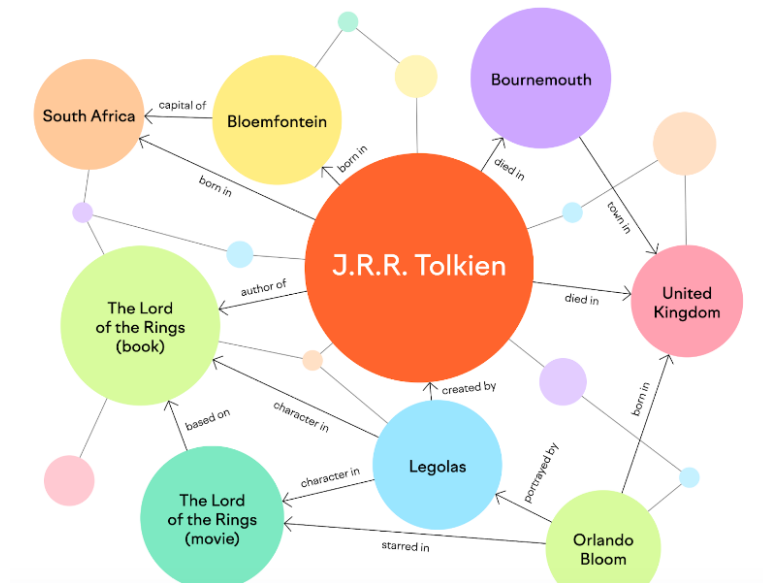
* Hogan et al. Knowledge Graphs, ArXiv:2003.02320v6, 2021.

Knowledge graphs

- Basically, a semantic network that stores data
 - Google, Wikidata, DPPedia, etc.
- Google KG Introduced in 2012, with the slogan “things, not strings”
- Aim: understand the meaning behind real-world entities and their relationships to one another
- Opposite to just show results based on the strings of words (keyword search)
- Google’s KG contains two basic types of data
 - Entities: Real-world topics like J. R. R. Tolkien, “The Lord of the Rings,” UK, etc.
 - Relationships: How entities connect to each other, e.g., J. R. R. Tolkien is the author of the book named “The Lord of the Rings.” Tolkien was born in South Africa in 1892.
- A large part of the information on the Google Knowledge Graph comes from Wikipedia and Wikidata (Wikipedia's KG)

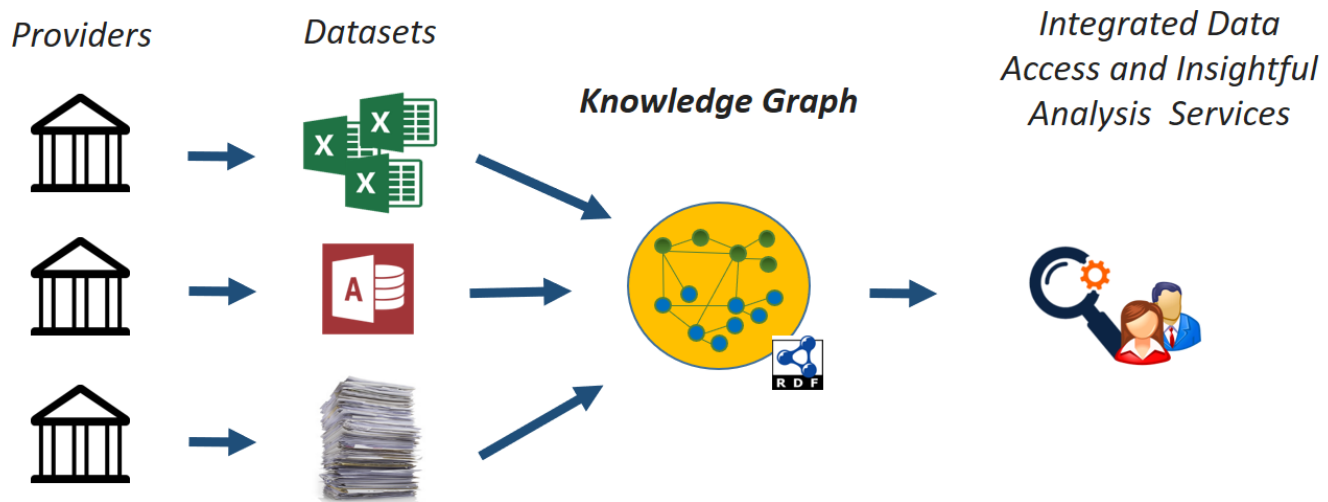
Knowledge graphs

- Companies want to be found by Google's KG
- Use structured information
 - Schema markup (with JSON-LD, microdata, and RDFa)
 - RDF graphs
 - Link the data to other websites



From <https://www.semrush.com/>

RDF knowledge graphs



Papadaki et al. A Brief Survey of Methods for Analytics over RDF Knowledge Graphs, 2023

RDF knowledge graphs: real-world examples

- Datasets represented in RDF
 - Wikipedia, Wikibooks, Geonames, MusicBrainz, WordNet, DBLP bibliography
- Governments:
 - USA, UK, Japan, Austria, Belgium, France, Germany, ...
- Active community

– http://en.wikipedia.org/wiki/Open_Data

– <http://www.w3.org/LOD>

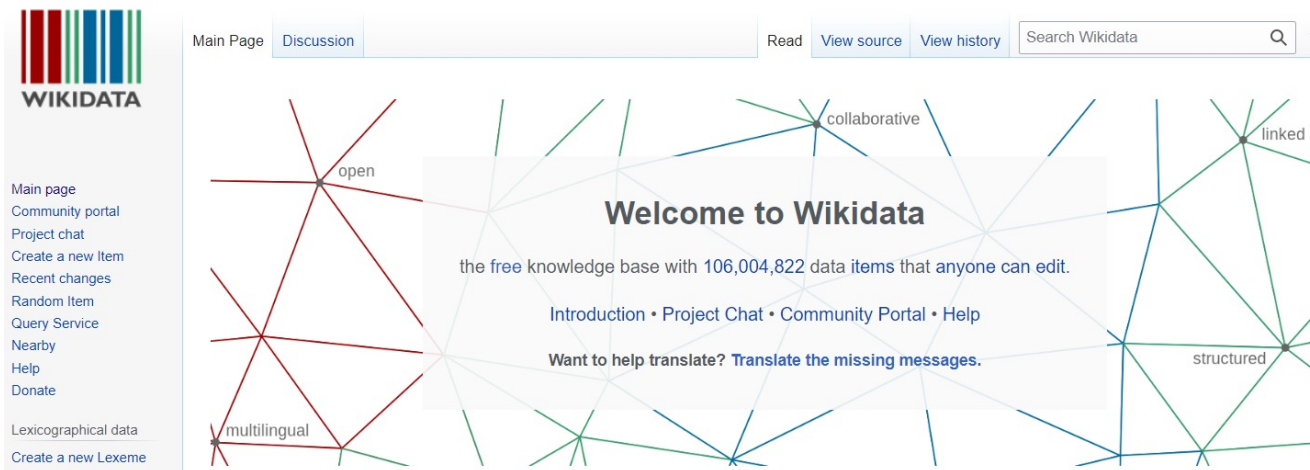
RDF knowledge graphs

- Freebase (2007 – 2016)
- Based on **graphs**:
 - nodes, links, types, properties, namespaces
- Google KG based in part on Freebase
 - Knowledge graph
 - Words become concepts
 - Semantic questions
 - Semantic associations
 - Browsing knowledge
 - Knowledge engine



RDF knowledge graphs

- **Wikidata** (<https://www.wikidata.org>)
- A free and open KB that can be read and edited by both humans and machines
- A central storage for the **structured data** of other projects like Wikipedia, Wikivoyage, Wiktionary, Wikisource, etc.
- Most data encoded via an item (rdf:subject), a property (rdf:predicate) and a value for that property (rdf:object), that is, RDF data



RDF knowledge graphs

- **Yago (<https://yago-knowledge.org/>)**
 - Created at Max Plank institute, 2008
 - A KB with knowledge about the real world
 - Contains entities (movies, people, cities, countries, etc.) and relations between these entities (who played in which movie, etc.)
 - More than 50 million entities and 90 million facts
 - Organizes its entities into classes: Elvis Presley belongs to the class of people, Paris belongs to the class of cities, and so on
 - Classes arranged in a taxonomy: The class of cities is a subclass of the class of populated places, this class is a subclass of geographical locations, etc.
 - Defines which relations can hold between which entities: birthPlace, e.g., is a relation that can hold between a person and a place
 - The definition of these relations and the taxonomy is called the **ontology**
 - **Endpoint at <https://yago-knowledge.org/sparql>**

Knowledge graphs: DBPedia

- A project dealing with structured data
- Wikipedia, DBpedia and Wikidata fulfill very different tasks
- DBpedia **extracts structured data from the infoboxes** in Wikipedia, and publishes them in RDF and a few other formats. It also provides a number of services with these data: DBpedia mobile, SPARQL endpoint, mappings to external ontologies, etc.
- Wikidata provides a database of structured data that everyone can edit. Instead of extracting structured data from infoboxes, it allows infoboxes to be created from structured data.
- Both projects publish RDF data about entities. The source of the data is very different: whereas DBpedia **extracts the data** from the infoboxes, Wikidata **collects data entered** through its interfaces

Querying the WWW (at DBpedia)

- Endpoint <http://dbpedia.org/sparql>

Query: Find the Music artists born in Argentina before 1970.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
PREFIX dbo: <http://dbpedia.org/ontology/>
```

```
SELECT DISTINCT ?person ?name ?birth
```

```
WHERE {
```

```
    ?person a dbo:MusicalArtist .
```

```
    ?person dbo:birthPlace dbr:Argentina .
```

```
    ?person dbo:birthDate ?birth .
```

```
    ?person foaf:name ?name .
```

```
    ?person rdfs:comment ?description .
```

```
    FILTER (LANG(?description) = 'en') .
```

```
    FILTER (LANG(?name) = 'en') .
```

```
    FILTER (?birth < '1970-01-01'^^xsd:date)
```

```
} ORDER BY ASC(?birth)
```

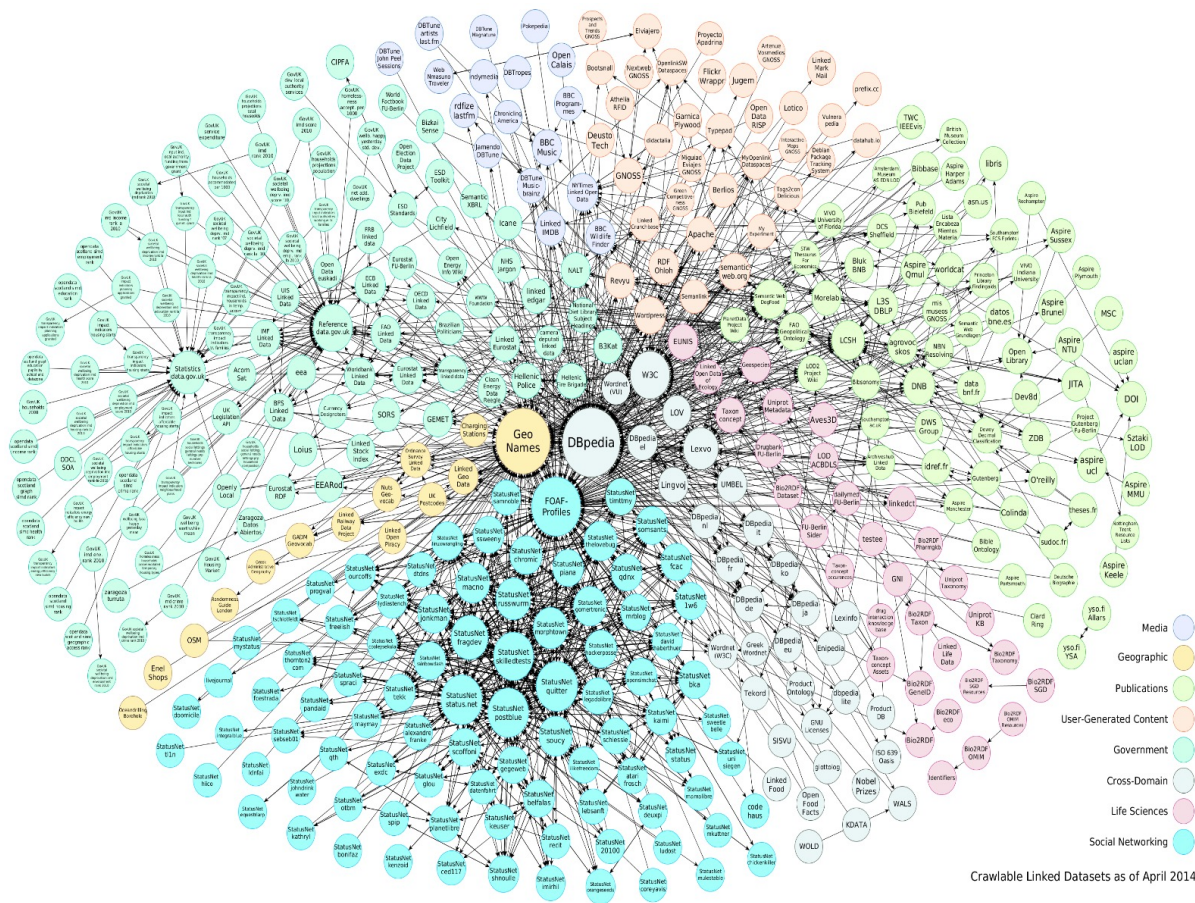
Querying the WWW (at Linkedgeodata.org)

- Endpoint: <http://linkedgeodata.org/sparql/>

Query: *Find historic churches within 100m from Leipzig Central Station*

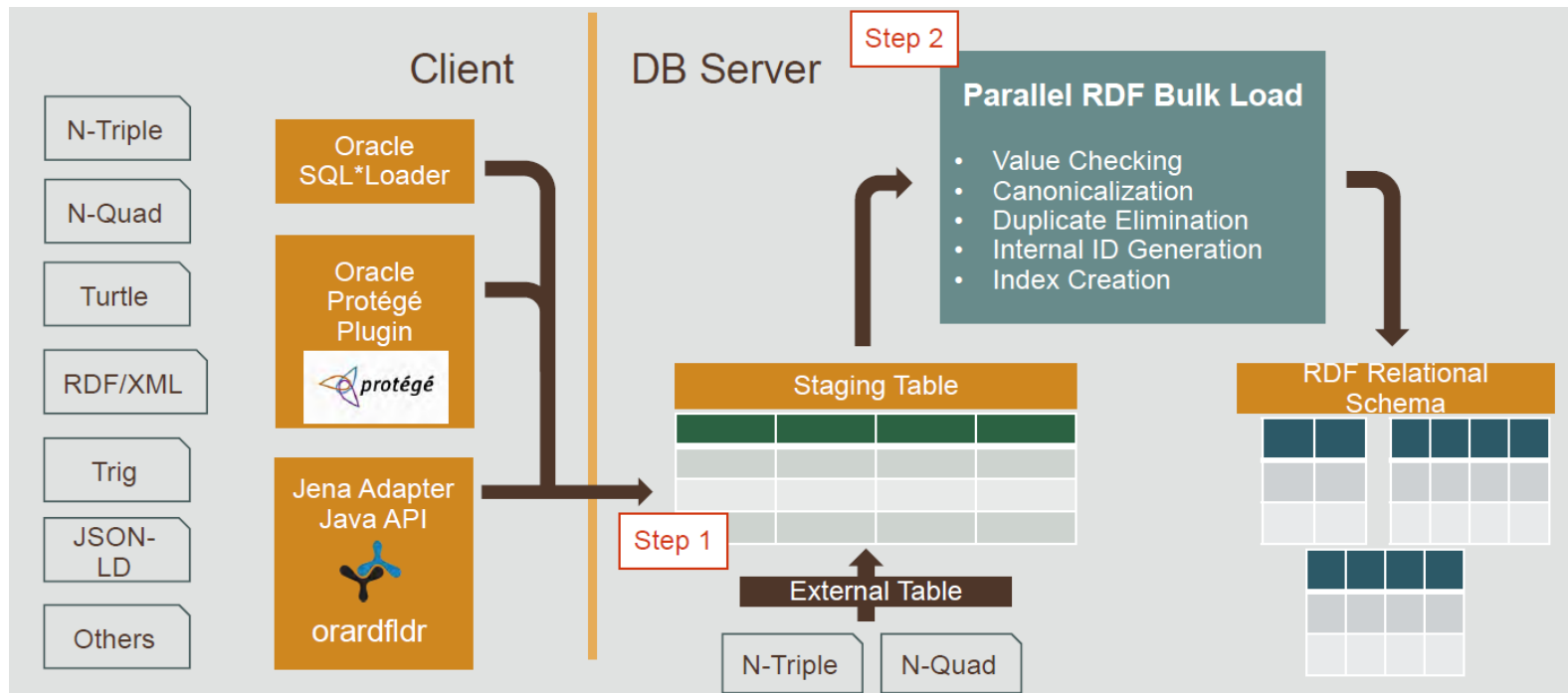
```
PREFIX lgdr:<http://linkedgeodata.org/triplify/>
PREFIX geom:<http://geovocab.org/geometry#>
PREFIX lgdo:<http://linkedgeodata.org/ontology/>
PREFIX ogc:<http://www.opengis.net/ont/geosparql#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
SELECT ?l ?xg {
    ?s owl:sameAs <http://dbpedia.org/resource/Leipzig_Hauptbahnhof>;
    geom:geometry[ogc:asWKT ?sg].
    ?x a lgdo:HistoricChurch; rdfs:label ?l ;
    geom:geometry[ogc:asWKT ?xg].
FILTER (bif:st_intersects(?sg,?xg,0.1)) . }
```

Knowledge Graphs: Linked Data Cloud



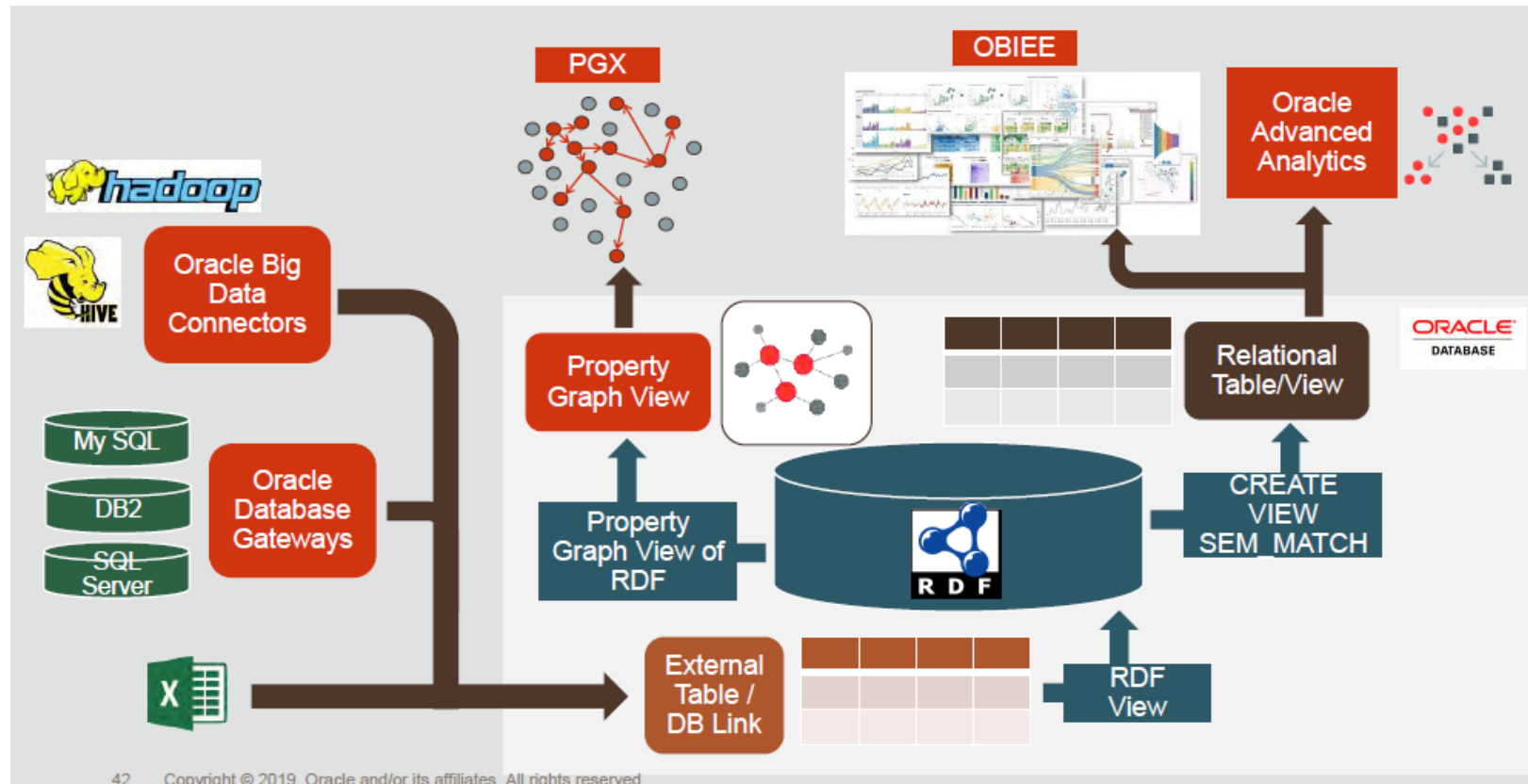
An RDBMS-based example

- Read RDF graphs into a relational database and embed SPARQL query in SQL (Oracle example below)



- Perry, M. Introduction to RDF Graph for Oracle Database 19c. Architecture and Overview (2019)

Using both models for analytics (example)



Next, we get into GRAPH DB's