Data Models

- ✓ Relational
- ✓ Key-Value
- XML
- JSON
- RDF
- Graph

Extensible Markup Language

Unstructured : Text

Semi-structured : We can parse it without meaning.

Structured data : DB

- Goal: human-readable and machine-readable
- For semi-structured (without schema) and structured data (with schema)
- XML languages: RSS, XHTML, SVG, `.xlsx` files, ... uses XML.
- Unicode

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<university name="TUM" address="Arcisstraße 21">
  <faculties>
    <faculty>IN</faculty>
    <faculty>MA</faculty>
    <!-- and more -->
  </faculties>
  <president name="Wolfgang Herrmann" />
</university>
```

UTF-8

XML declaration

Element

Comment

Empty-element tag, with an attribute

Well-formedness

- XML documents have a single root, i.e. are one hierarchy
- Tags may not overlap (<a>)

Querying and Validation

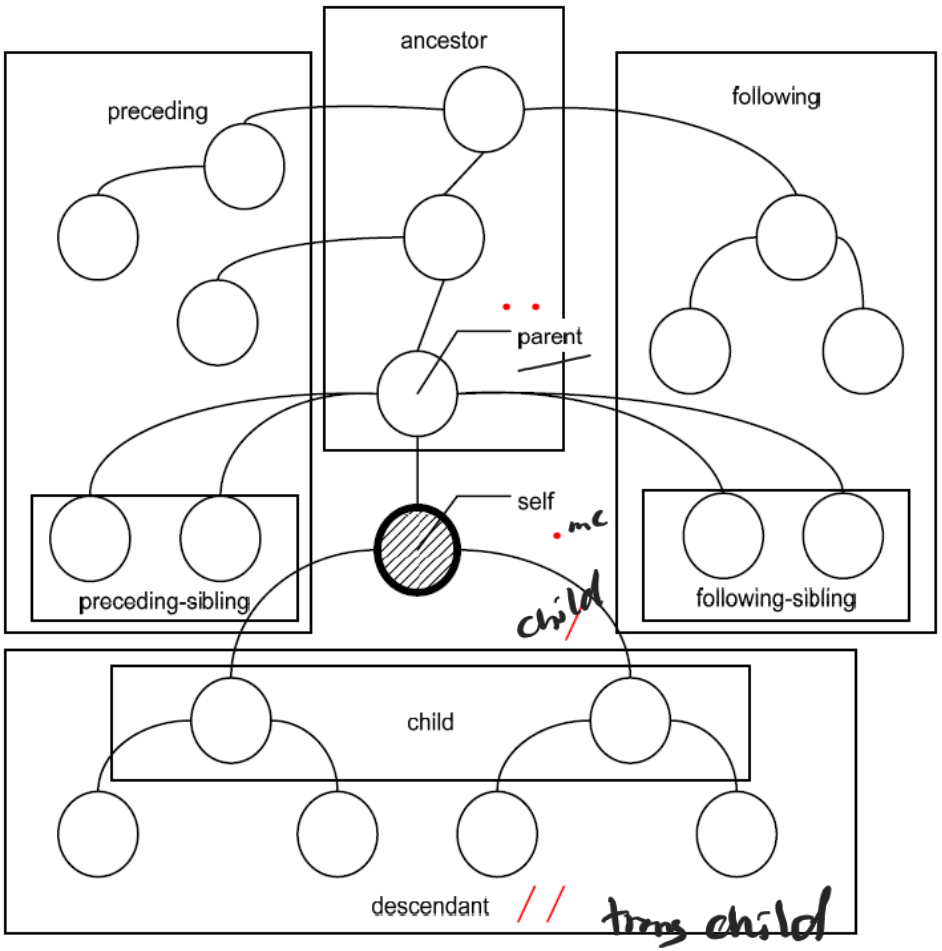
Querying

- Declarative querying
  - XML Path Language (XPath), for simple access
  - XML Query (XQuery), for CRUD operations, query language for XML databases
- Tree-traversal querying
  - Document Object Model (DOM)

Validation requires a definition of validness, i.e. a schema/grammar

- Document Type Definition (DTD), old and simple
- XML Schema (XSD), more powerful, but rarely used

XPath



`axis::node[predicate]/@attr`

`/university[@name="LMU"]`
→ empty set

`//faculty/text()`
→ IN, MA

`//president/@name`
→ `name="Wolfgang Herrmann"`

Activity

Load a dataset and prepare your environment:

- Install the command-line tool `xmllint`. We will use this syntax:
  `xmllint --xpath '<xpath expression>' <filename>`
- Download TUM's news feed from
  `https://www.tum.de/nc/en/about-tum/news/?type=100`.
  Familiarize yourself with the document's structure.

Use XPath to answer the following questions:

- List all titles. //title/text()
- List only the item titles (but not channel and image titles). //item/title/text()
- At what times does `letz@zv.tum.de` publish news?

Document Type Definition

- Associating an XML file with an existing DTD

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/
    xhtml1-transitional.dtd">
```

- Defining a grammar/schema

```
<!ELEMENT university (faculties, president)>
<!ATTLIST president name CDATA #REQUIRED>
...
```

JSON

- Goal: human-readable and machine-readable
- Similar use-cases and approach as XML, seems to be replacing it
- Originally designed for JavaScript, now used independently of the programming language
- Key-value pairs (cf. NoSQL)
- Used within relational database to allow for arbitrary complex types
- Extensions/Derived languages e.g., for geographic data, linked data, as a binary data format, ...

Example

```
{
  "name":  "TUM",
  "address":  "Arcisstraße 21",
  "faculties":  [
    { "name":  "IN" },
    { "name":  "MA" },
    ...
  ],
  "president":  {
    "name":  "Wolfgang Herrmann"
  }
}
```

Key-value pair

Array

(Nested) Type

Schema

- Similar to XSD, and similarly rarely used
- Written in JSON

```
{
  "$schema": "http://json-schema.org/schema#",
  "title": "University",
  "type": "object",
  "required": ["name", "faculties", "president"],
  "properties": {
    "name": {
      "type": "string",
      "description": "Name of the university"
    },
    ...
  }
}
```

Activity

1. Access the JSON processor jq and the Bundesliga 2016/17 dataset:
   - Online: `jqplay.org`. Copy the dataset from `https://raw.githubusercontent.com/openfootball/football.json/master/2016-17/de.1.json`.
   - Command-line: Install `jq`.
     Syntax: `curl '<dataset url>' | jq '<filter>'`
2. Learn basic jq filter syntax from these examples:
   - Path navigation, array syntax: First round (1. Spieltag) matches:
     `.rounds[0].matches[]`
   - Pipes, object construction: Fixtures of the first round:
     `.rounds[0].matches[] | {A: .team1.code, B: .team2.code}`
   - Filters: Teams that won 2:1 in a home game[1]:
     `.rounds[].matches[] | select(.score1==2) | select(.score2==1).team1.name`
3. Use jq to find the dates of home games of FC Bayern München (FCB).

[1] The home team is listed first. Thus, its data is stored in `team1` and `score1`.

Resource Description Framework (RDF)

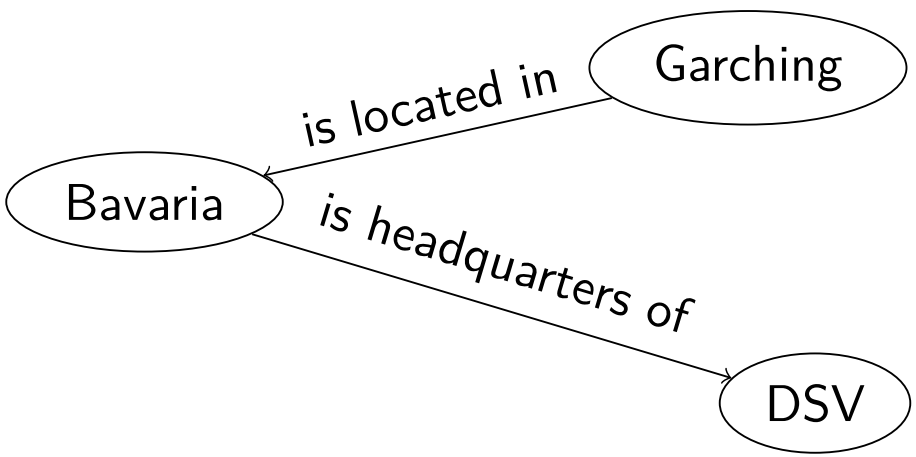
- W3C standards
- Intended use case: Meta-data model for resources
  - URIs as keys, for uniquely identifying those resources
  - Schema-less
- Nowadays mostly used for: Knowledge/Semantics data
  - New information can easily be added
  - URIs allow for arbitrary connection of RDF datasets

Triple: Smallest information unit in RDF

Garching (Subject) is located in (Predicate) Bavaria (Object).

Bavaria (Subject) is headquarters of (Predicate) the German Ski Association (Object).

Graph representation:



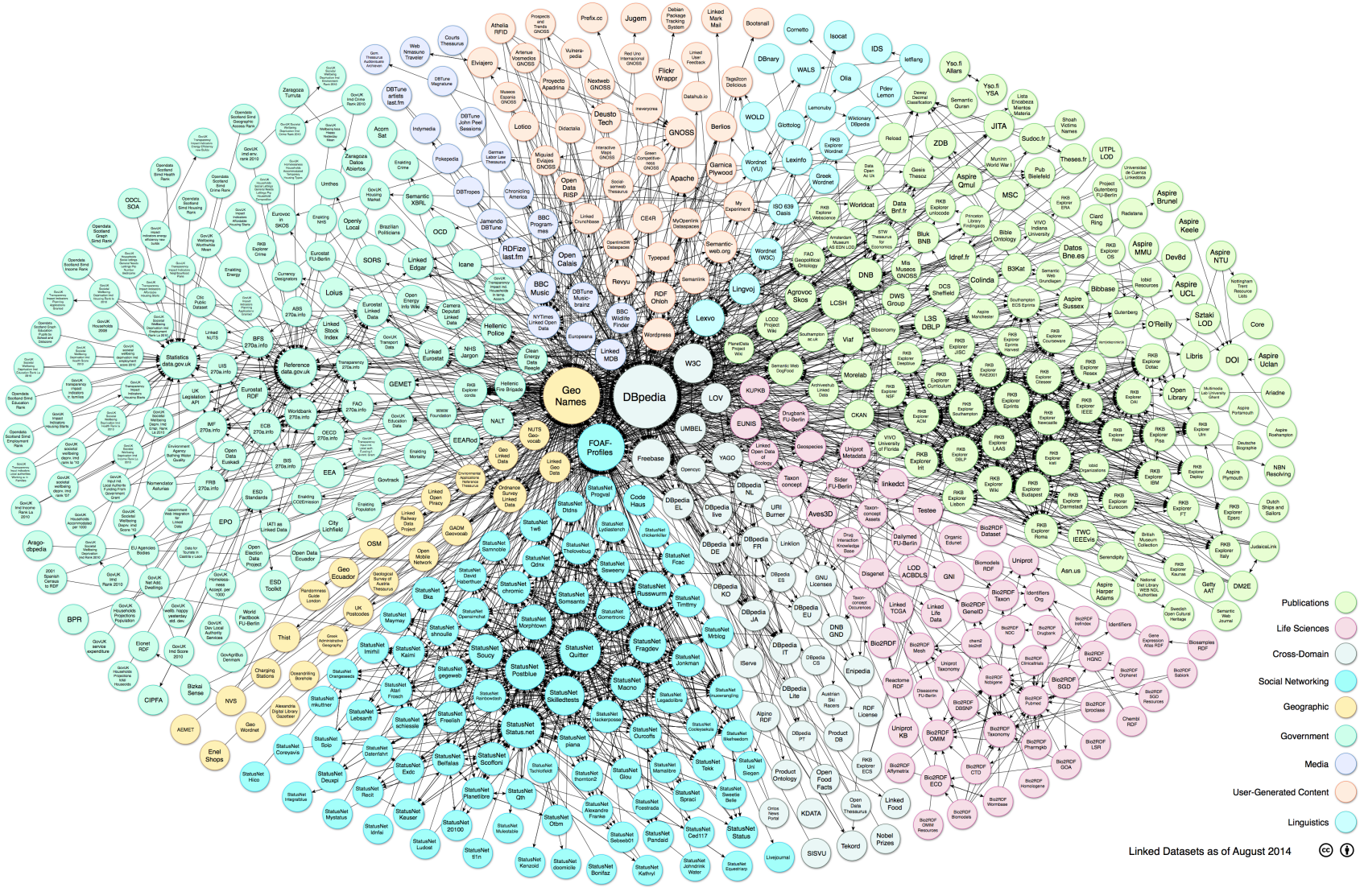
SPO triplestore representation:

S	P	O
Garching	is located in	Bavaria
Bavaria	is headquarters of	German Ski Association

Graphs: Linking information snippets together

- Multi-graph data with labeled edges
- Arbitrary RDF datasets can be connected using
  - Common data (i.e. common nodes/URIs),
    e.g. `dbr:Garching`
  - Common model (i.e. common node types and edges),
    e.g. `gr:acceptedPaymentMethods` for eCommerce data
- Ontologies: philosophical effort to categorize all existing entities into a hierarchy
  - e.g.: a `dbo:Settlement` is described by an official name, an average temperature, a phone prefix, ...

Linked Open Data Cloud[2]



[2]Linking Open Data cloud diagram 2014, by Max Schmachtenberg, Christian Bizer, Anja Jentzsch and Richard Cyganiak. `http://lod-cloud.net/`

SPARQL

Names of persons with black hair and green eyes:

```
SELECT ?n
WHERE {
  ?p rdf:type dbo:Person .
  ?p dbo:hairColor "Black" .
  ?p dbo:eyeColor "Green" .
  ?p dbp:name ?n .
}
```

Serialization / File formats

- N-Triples and N-Quads:
  `subject predicate object^^datatype (graph tag) .`, e.g.
  `<http://en.wikipedia.org/wiki/Helium> <http://example.org/elements/atomicNumber> "2"^^<http://www.w3.org/2001/XMLSchema#integer> .`
- Notation3 (N3): avoid repetition by defining URI prefixes and storing multiple P-O pairs without repeating S:

```
@prefix dbo:http://dbpedia.org/ontology/
subject predicate object ;
        predicate object .
```

- RDF/XML
- Microformats and RDFa: annotate (X)HTML documents with RDF snippets:
  `<div typeof="dbo:Person">...</div>`

Datatypes

```
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
```

Built-in datatypes

- xsd:integer, xsd:decimal, xsd:double, ...
- xsd:boolean
- xsd:string
- xsd:dateTime
- xsd:base64Binary
- ...

User-defined Datatypes:

```
<xsd:simpleType name="adultAge">
  <xsd:restriction base="integer">
    <xsd:minInclusive value="18">
  </xsd:restriction>
</xsd:simpleType>
```

Web Ontology Language (OWL)

- RDF language, schema vocabulary, can thus be defined in any of the file formats, rarely used
- Definition of classes and properties
- Reasoning and inference over relationships
  - OWL Full is undecidable, OWL DL is a decidable-yet-expressive subset
- Basic concepts
  - Classes: `owl:Class`
  - Objects that are literals: `owl:DatatypeProperty`, `rdf:XMLLiteral`
  - Objects that are nodes: `owl:ObjectProperty`
- Class hierarchies: `rdfs:subClassOf`, `owl:disjoint`, ...
- Relationships: `owl:inverseOf`, `owl:minCardinality`, ...

RDF vs. relational modeling

- Easier to iteratively change ("evolve") data and schema
- Easier to connect to other data sources and models
- Harder to validate data against its schema
- Harder to efficiently query/store

SPARQL Protocol and RDF Query Language (SPARQL)

- "SQL for triples"
- Pattern matching over triples, variable binding between patterns and to the output
- Returns matching triples
- Only basic functionality, but many extensions exist (insert/update/delete, subqueries, aggregates, . . .)

SPARQL: Prefixes and ~~rdf:type~~ a

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX dbp:  <http://dbpedia.org/property/>
SELECT ?n
WHERE {
  ?p ~~rdf:type~~ a dbo:Person .
  ?p dbo:hairColor "Black" .
  ?p dbo:eyeColor "Green" .
  ?p dbp:name ?n .
}
```

SPARQL: Blank nodes

- An anonymous resource
- In RDF data: three types of subjects/objects: URI, literal, blank node
- In SPARQL queries: Variables with local scope, thus cannot appear in `SELECT`
- Syntax `[]` when only used once:
  ```
  SELECT ?n WHERE { [] dpb:name ?n . }
  ```
- Syntax `_:xyz` when used multiple times:
  ```
  SELECT ?p ?n WHERE { ?p dbp:predecessor _:s .
    _:s dpb:name ?n . }
  ```

SPARQL: Advanced SELECT queries

- `SELECT DISTINCT ...WHERE ...`
- `SELECT ...WHERE ...ORDER BY ?x DESC(?y)`
- `SELECT ...WHERE ...ORDER BY ...LIMIT 5 OFFSET 10`
- `FILTER`: Within `WHERE`, for filters that are more complex than simple pattern matching:
  Regular expression: `FILTER regex(?title, "^TUM")`
  Arithmetic expression `FILTER (?price <30.5)`
- Property Paths (graph pattern matching): `?p foaf:knows+/foaf:name ?n .`

SPARQL: Other query types

- Retrieve all information about the matched resources (i.e. not only the matching triples but all triples that describe the resources):
  `DESCRIBE ?p WHERE { ?p dbo:eyeColor "Green" . }`
- Containment check:
  `ASK { ?x dbp:name "John Doe" . }`
- Return a graph instead of a list of triples:
  `CONSTRUCT ...WHERE ...`

Activity

You can access the DBpedia dataset[3] at `https://dbpedia.org/sparql`.

- World record: Find the `dbo:Film` with the longest `dbo:runtime`.
- Data cleansing: Find all `dbo:Film` with negative `dbo:runtime`.
- Alien actors: Find the `dbp:name` and `dbo:thumbnail` of persons `dbo:starring` in `dbo:Film` that have "UFO" in their title (`dbp:name`)

[3] Semantic information from Wikipedia, extracted e.g. from infoboxes

Triplestores

- Database systems for RDF data
- Either specialized new systems or extensions to existing (relational or document) DBMSs

Some triplestores

- OpenLink Virtuoso (combines SQL and RDF, open-source edition)
- AllegroGraph (ACID, available on EC2)
- Apache Jena TDB (open-source, part of a Java framework)
- Ontotext GraphDB ("semantic repository", with text mining features)
- Blazegraph (GPU-accelerated, scale-out, open-source edition)
- Oracle Spatial and Graph (Oracle Database extension)

What is a graph?

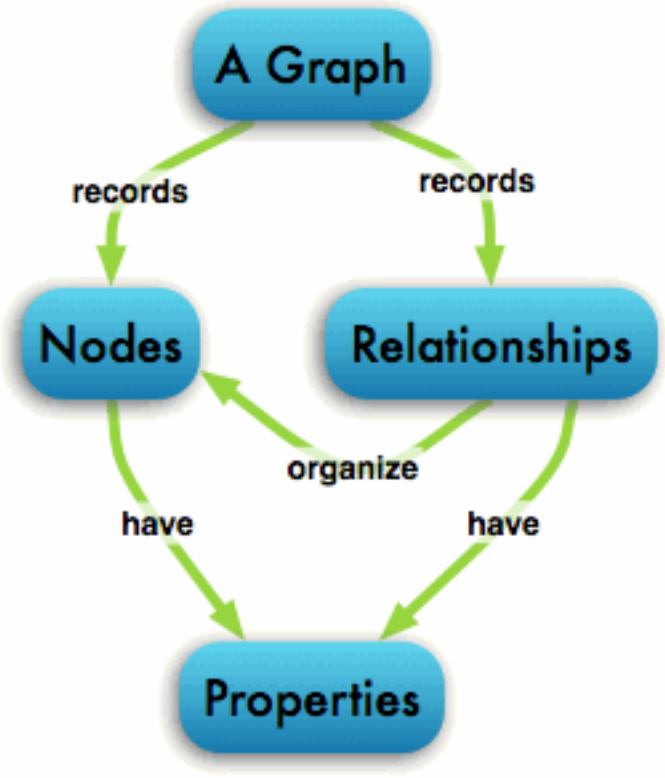
$$G = (V, E)$$

- A set of vertices (v, nodes) and edges (e)
- Many variants: directed?, multi-graph?, connected?, cyclic?, hyper-graph?, ...
- Add "properties" (key-value pairs) to vertices and/or edges to make them useful for data engineering

RDF vs. Property graphs

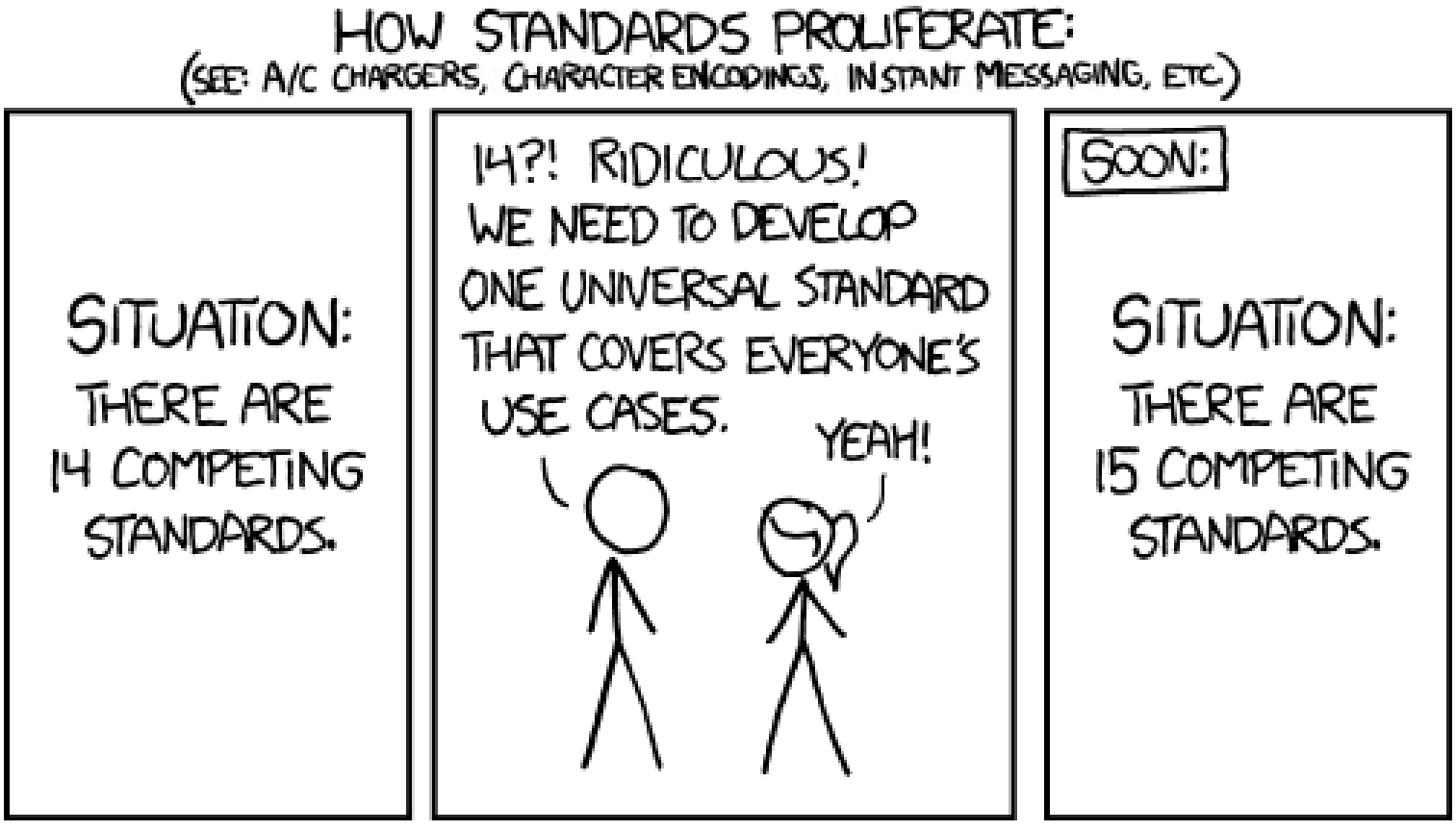
Very similar concepts

- RDF is a special, restricted type of graphs
- Graph databases can store more variants, e.g. undirected graphs, hypergraphs
- RDF has more standards as it is focused on creating linked, shared data
- RDF engines can infer new knowledge through semantic reasoning
- Graph engines are well-suited for graph traversals (e.g. shortest path algorithms)



A property graph of the property graph data model

Graph Query Languages



SPARQL – SQL – Cypher – Gremlin – PGQL/GMQL – GraphQL – Nepal – GRAPHiQL – Green-Marl – . . .

Activity: Neo4j and Cypher

Neo4j: A Java-based open-source graph database

Cypher: The declarative graph query language of Neo4j

- Download community edition from `neo4j.com`, extract archive
- Start the database: `./bin/neo4j console`
- Open the web interface: `http://localhost:7474/`
- Login (neo4j/neo4j), Change the password, login again
- Start tutorial: Type `:play movie graph` into the web console
- Follow tutorial, try to understand Cypher syntax
- Want to learn more about Cypher? Type `:play cypher`

Query Types

- Pattern matching, e.g., structural, approximate, variable length
- Reachability, e.g., shortest path, transitive closure
- Aggregation
- Graph properties, e.g., degree of a vertex, diameter of the graph
- Analytical, e.g., PageRank, clustering coefficient

Graph queries—unlike SQL queries—support multiple return types

- Node ; multiple nodes
- Table
- Path (usually represented as $[v, e, v, e, \ldots]$) ; multiple paths
- Subgraph (i.e., a set (V, E))

Not all languages cover all query and return types.

Distributed graph systems: Pregel/Giraph

- Pregel: Google's iterative graph processing system
- Apache Giraph: open-source counterpart
- Gremlin: functional graph traversal language
- For billions of vertices and trillions of edges
- For thousands of commodity computers

"Think like a vertex": The vertex-centric scatter-gather programming model

- Per iteration, a vertex can ...
  - receive messages sent in the previous iteration
  - modify its state
  - modify the state of its outgoing edges
  - send messages to other vertices
- Implicit synchronization after each iteration

Distributed graph systems: Pregel/Giraph

Why Pregel/Giraph? There already is MapReduce/Hadoop!

– Yes, but you cannot have operations on a vertex-level in MapReduce. Hence, you cannot "think like a vertex".

– In addition, the MapReduce model requires transferring all state information to other compute nodes, whereas the only inter-node communication in the Pregel model are messages. Thus, Pregel keeps the communication overhead low while allowing vertices to keep state.

Distributed graph systems: Turi GraphLab/SGraph

- Turi: Apple's machine learning framework
- SGraph: Turi's open-source graph engine
- Runs on one or many nodes without code modification
- Seamless transformation between tabular and graph representation
- Disk-based, HDFS integration
- Spark integration
- Many machine learning libraries available

I think, therefore I am.

"Think like an edge"
- RDF

"Think like a vertex"
- Distributed graph engines (Giraph, Turi)

"Think like a graph"
- (A new approach for distributed graph engines. Looks at the whole graph structure to compute a better partitioning.)

I think, therefore I am

```
vertex_scatter(vertex v)
  send updates over outgoing edges of v

vertex_gather(vertex v)
  apply updates from inbound edges of v

while not done
  for all vertices v that need to scatter updates
    vertex_scatter(v)
  for all vertices v that have updates
    vertex_gather(v)
```

Vertex-centric scatter-gather
(cf. Pregel/Giraph)

```
edge_scatter(edge e)
  send update over e

update_gather(update u)
  apply update u to u.destination

while not done
  for all edges e
    edge_scatter(e)
  for all updates u
    update_gather(u)
```

Edge-centric scatter-gather

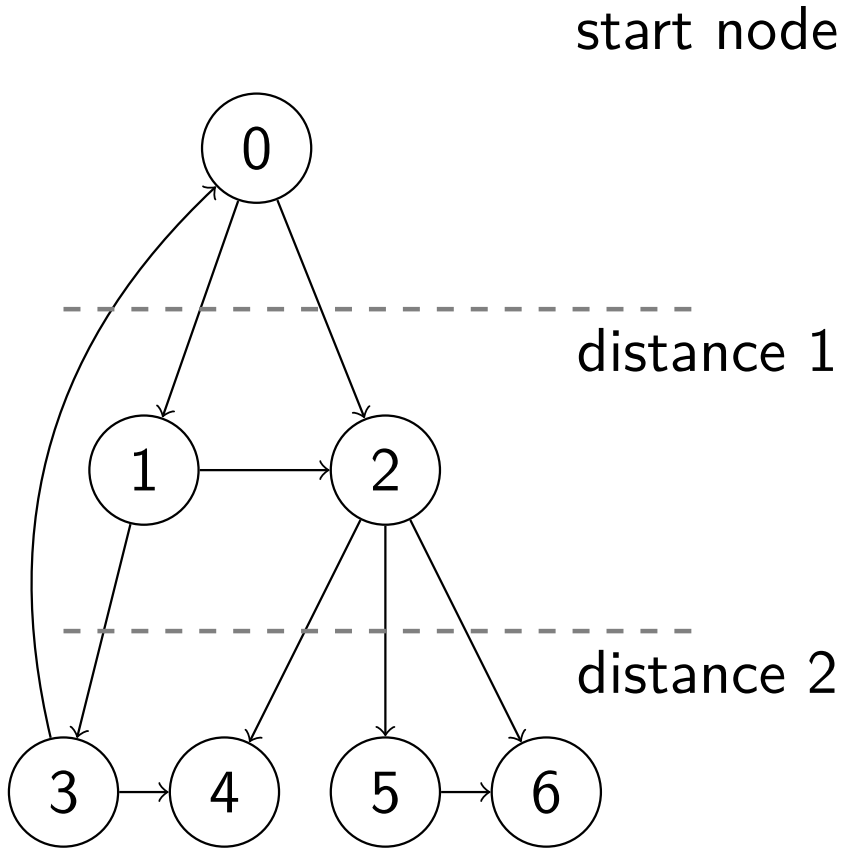
(Roy, A., Mihailovic, I., & Zwaenepoel, W. (2013). X-Stream: Edge-centric Graph Processing using Streaming Partitions. Proceedings of SIGOPS (pp. 472-488). ACM.)

Comparison of distributed graph systems

- Both Giraph and GraphLab use the vertex-centric scatter-gather model
- Both are built to scale out and handle huge datasets
- Similar overall performance and memory usage
- Explicit vs. implicit data transfer
  - Giraph uses explicit messaging, whereas GraphLab implicitly transfers information between vertices by making changes visible to adjacent vertices
  - Thus, vertices in Giraph can only access data in the message, whereas vertices in GraphLab can access data of adjacent vertices directly
- Giraph connects better to the Java world (e.g. Hadoop)
- GraphLab supports many machine learning tasks

Graph algorithms: Breadth-first search (BFS)

- Graph traversal algorithm
- From a start node: explore all reachable nodes ordered by their distance from the start node
- $O(|V| + |E|)$
- Building block for many graph algorithms, e.g., graph serialization, checking for bipartiteness, shortest path



Graph algorithms: Breadth-first search (BFS)

BFS in a declarative language (Cypher):

```
MATCH (<start node>)
MATCH p = (n)-[*1..]->(m)
RETURN last(nodes(p)) ORDER BY length(p) ASC
```

BFS in imperative pseudo-code:

```
create queue
enqueue start node, mark as visited
while queue is not empty do
    take first element from queue
    print it
    foreach unvisited adjacent nodes do
        enqueue, mark as visited
    end
end
```

Graph algorithms: Dijkstra's algorithm

- Single-source-single-sink shortest path algorithm

```
distance[]←Inf
distance[start node]←0
current←start node
while end node is not visited do
    foreach unvisited adjacent neighbor n do
        distance[n]←min(distance[n], distance[current] + edge weight)
    end
    mark current node as visited
    current←smallest distance of unvisited nodes
end
```

- $O(|V|^2)$. Maintain a Fibonacci heap to efficiently find the next `current` and reduce the complexity to $O(|V|log|V| + |E|)$
- Edge weights must be non-negative! Otherwise: Bellman-Ford