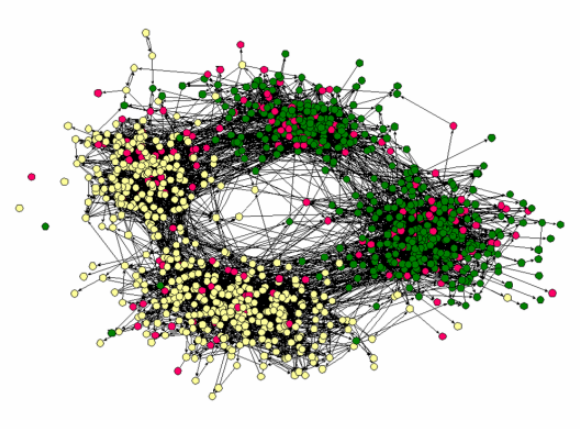
Graph Data Management Systems for New Application Domains: Social Networks & the Web of Data

Philippe Cudré-Mauroux
University of Fribourg
Switzerland

Sameh Elnikety
Microsoft Research
USA

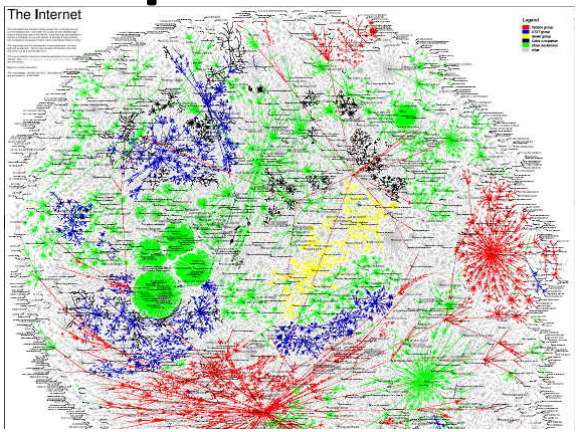
Tutorial at VLDB 2011

Welcome to Graphs



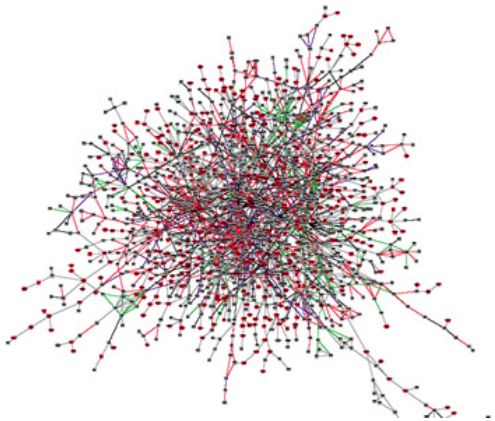
Friendship Network

[Moody'01]



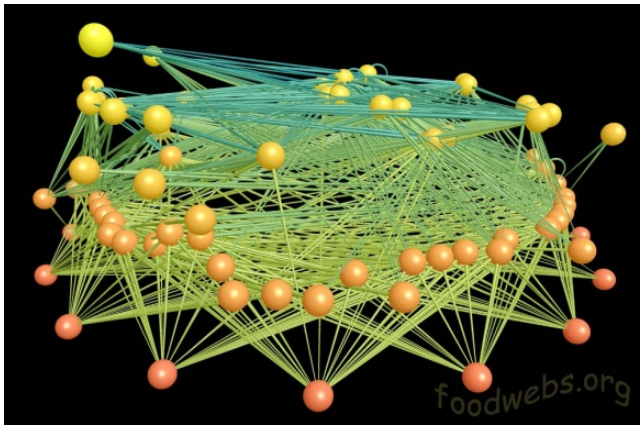
Internet Map

[lumeta.com]



Protein Interactions

[genomebiology.com]



Food Web

[foodwebs.org]

Graphs: Small and Large

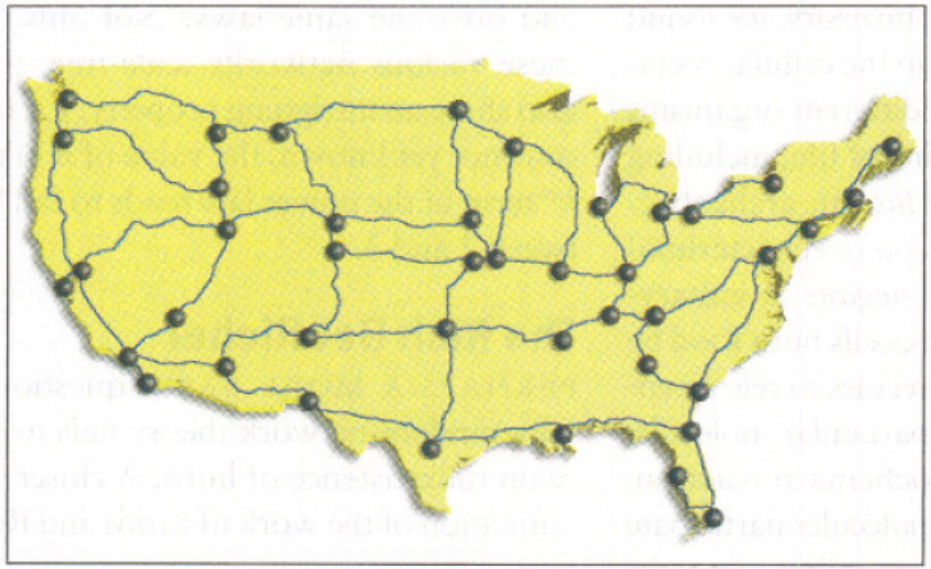
- Small graphs
  - Manage a collection of small graphs
  - Bioinformatics and cheminformatics
  - Well studied
- Large graphs
  - One large graph, aka “network”
  - Social network, and knowledge representation
  - Less studied

Classes of Large Graphs

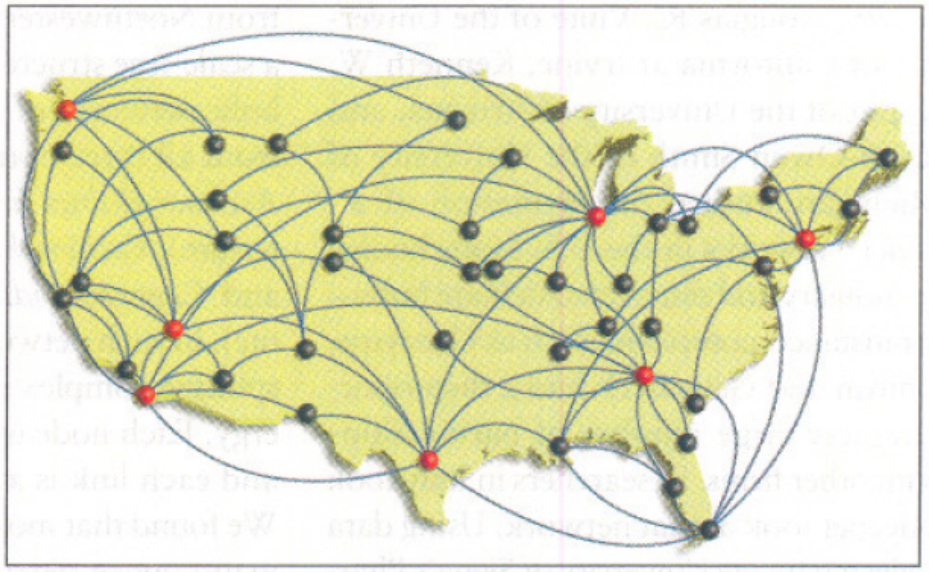
- Random graphs
  - Node degree is constrained
  - Less common
- Scale-free graphs
  - Distribution of node degree follows power law
  - Most large graphs are scale-free
  - Small world phenomena & hubs
  - Harder to partition

Classes of Large Graphs

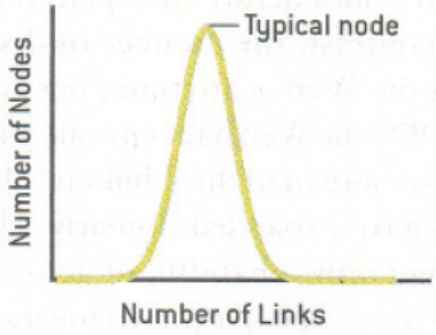
Random Network



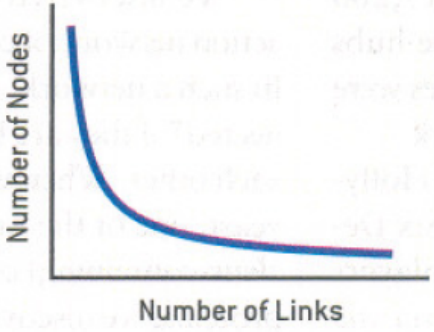
Scale-Free Network

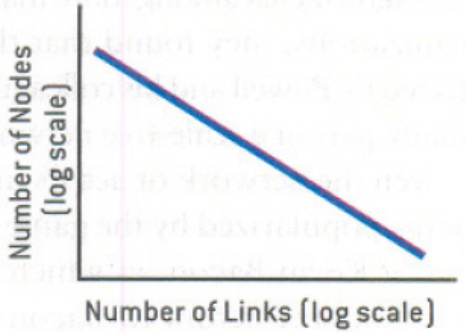


Bell Curve Distribution of Node Linkages

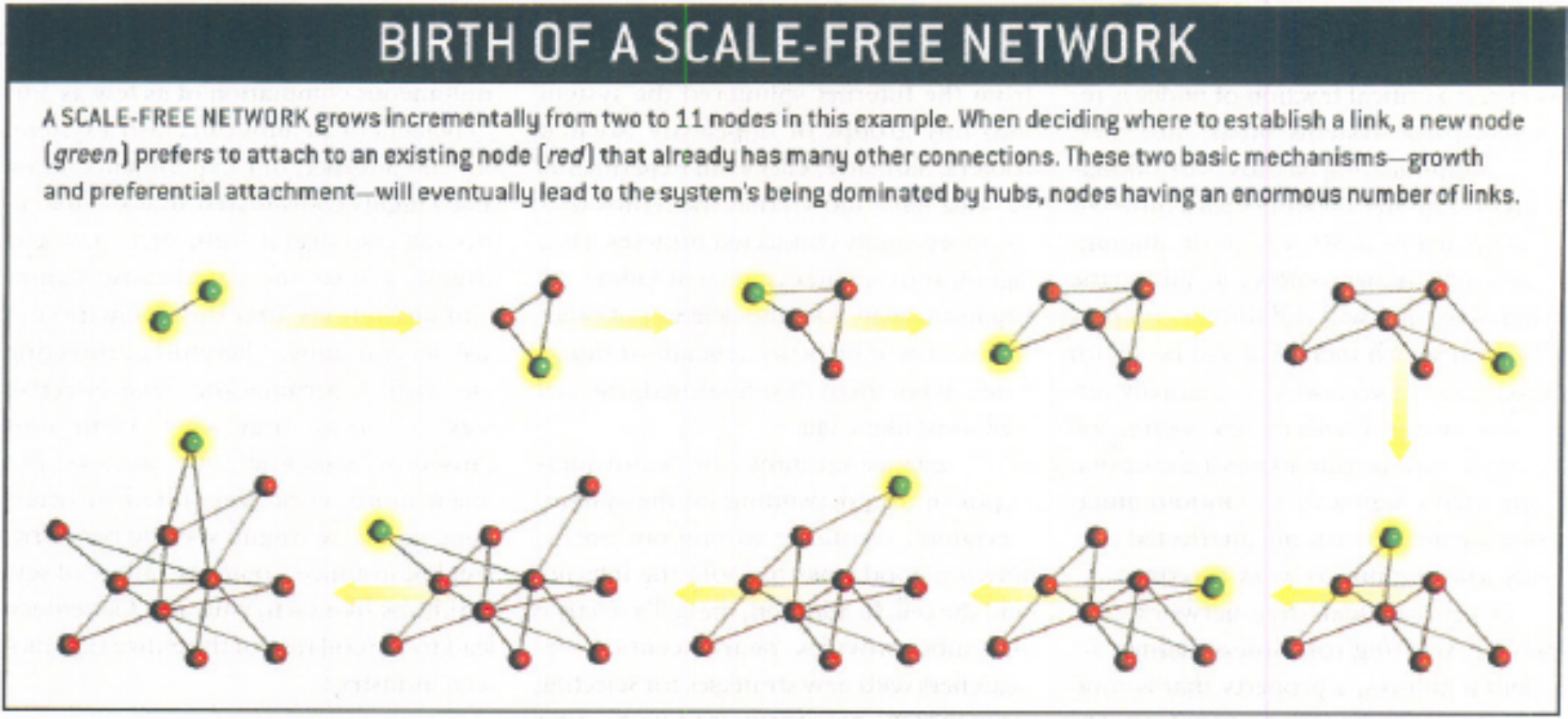


Power Law Distribution of Node Linkages





Organic Growth -> Scale Free



Examples of Organic Growth

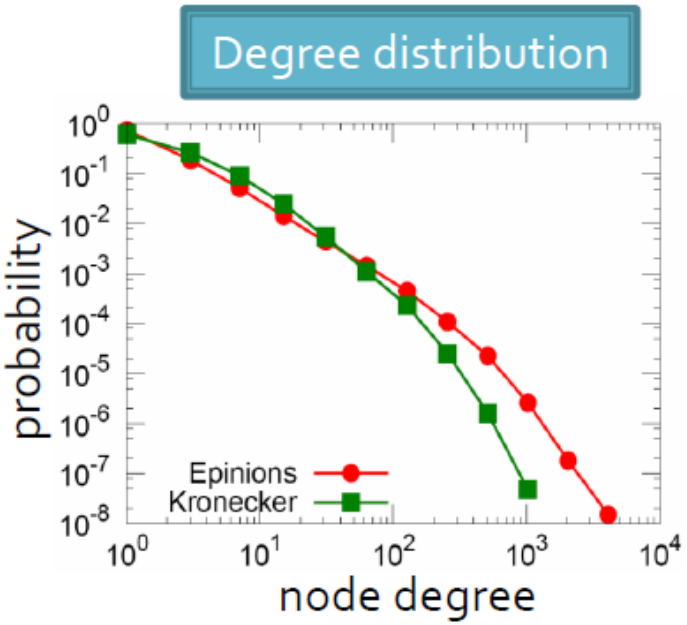
NETWORK	NODES	LINKS
Cellular metabolism	Molecules involved in burning food for energy	Participation in the same biochemical reaction
Hollywood	Actors	Appearance in the same movie
Internet	Routers	Optical and other physical connections
Protein regulatory network	Proteins that help to regulate a cell's activities	Interactions among proteins
Research collaborations	Scientists	Co-authorship of papers
Sexual relationships	People	Sexual contact

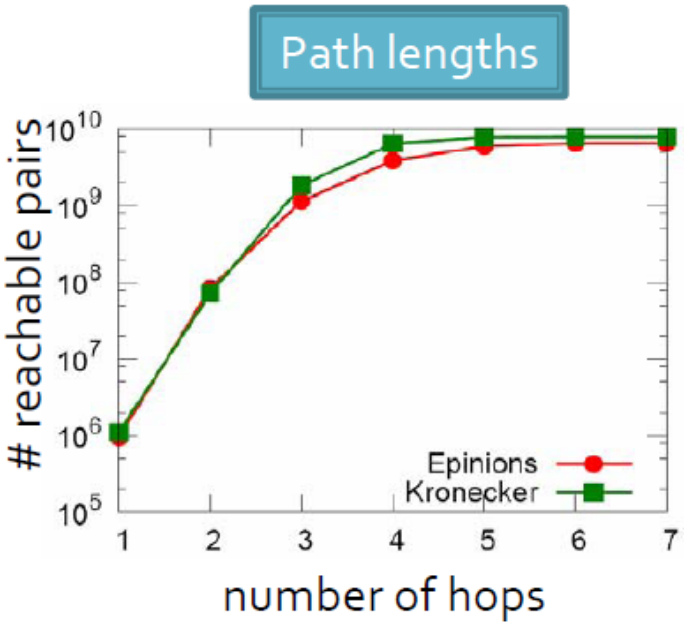
Generating a Large Graph

- Random graph
  - Fix the number of nodes (no growth)
  - Each edge connects two random nodes
- Scale-free graph
  - Copy model
    - Add new node
      - Take percentage of links from another node
  - Kronecker graphs

Kronecker Graph Example

- Epinions (N=76K, E=510K)
- Fitting time = 2 hours
- Real and Kronecker graphs are close





Let's Go Hyper!

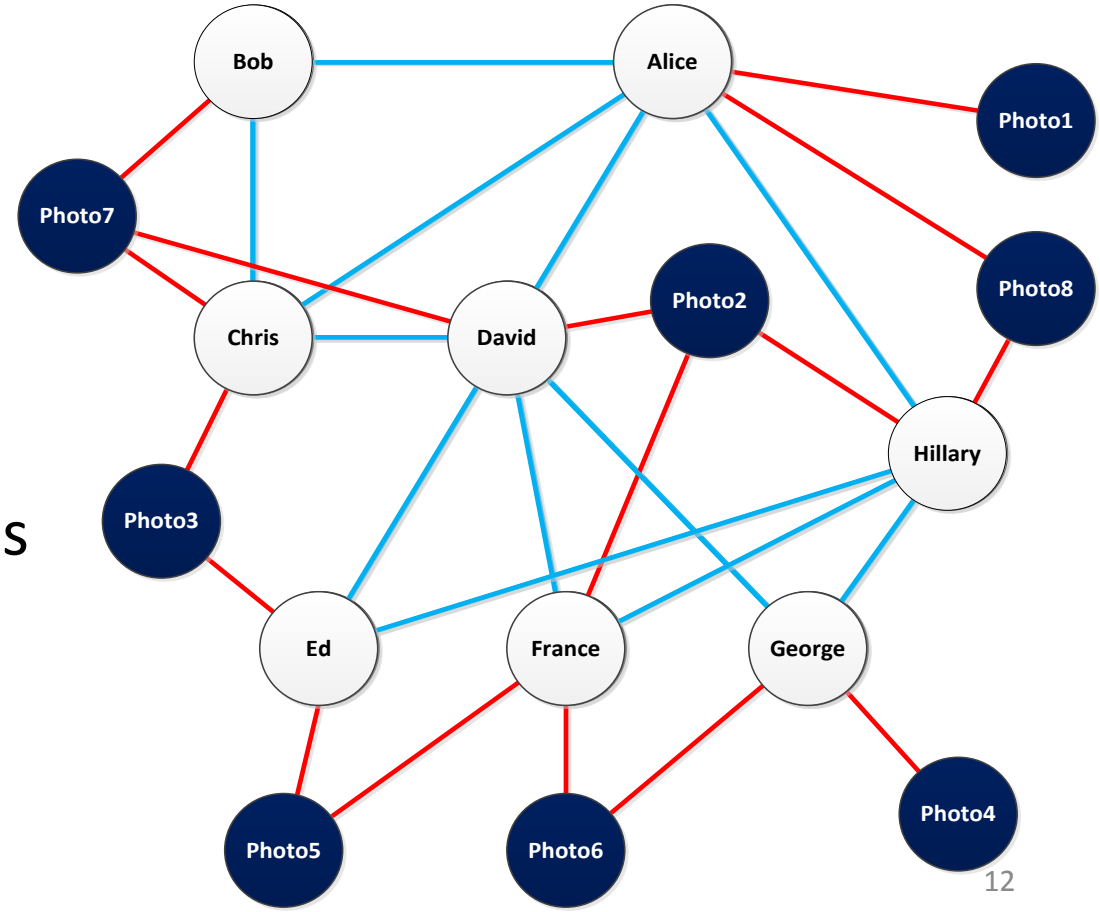
- Hyper-edge
  - A traditional edge is binary
  - A hyper edge relates **n** nodes
    - Order can be important
    - Child-of edge versus father, mother, child hyper-edge
- Hyper-node
  - A traditional node represents one entity
  - Hyper node represents a set of nodes
    - Person node versus family hyper-node

Roadmap

- Intro to Graphs
- **Social Networks**
  - Data Model
  - Queries
  - Processing
- Web of Data
  - Data Model
  - Queries
  - Processing
- Systems
- Current Research Directions

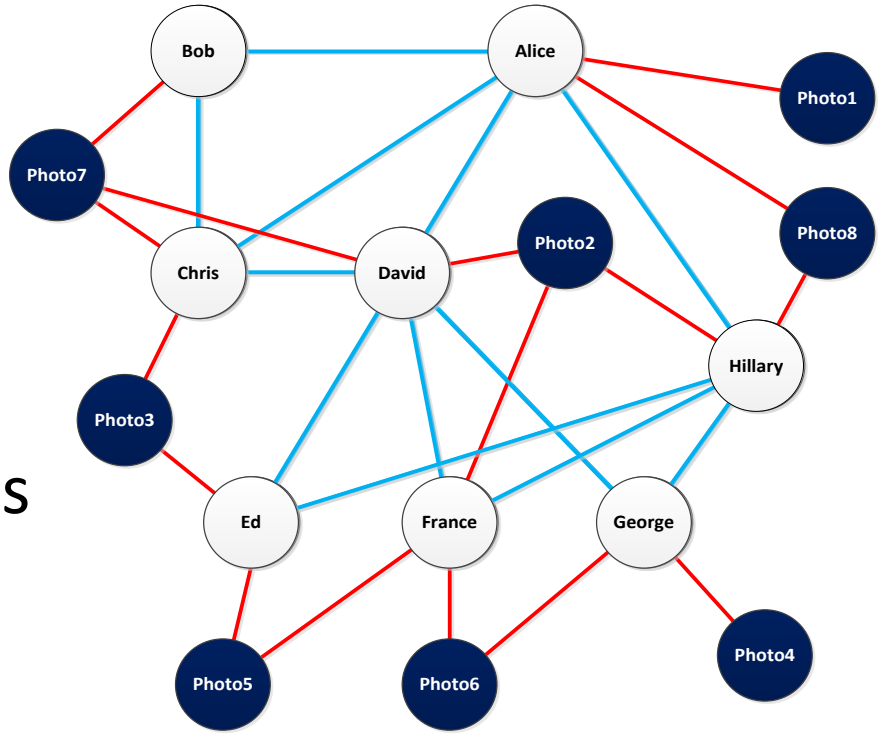
Social Networks

- Scale
  - LinkedIn
    - 70 million users
  - Facebook
    - 500 million users
    - 65 billion photos
- Queries
  - Alice's friends
  - Photos with friends
- Rich graph
  - Types, attributes



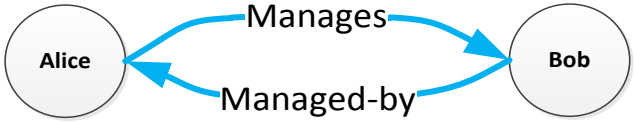
Social Networks: Data Model

- Node
  - ID, type, attributes
- Edge
  - Connects two nodes
  - Direction, type, attributes



App

System



Managing Graph Data

- Here we focus on online access
  - Rather than offline access
    - Network analytics and graph mining
- Queries
  - Read
- Updates
  - Data update: change node payload
  - Structural update: modify nodes and edges

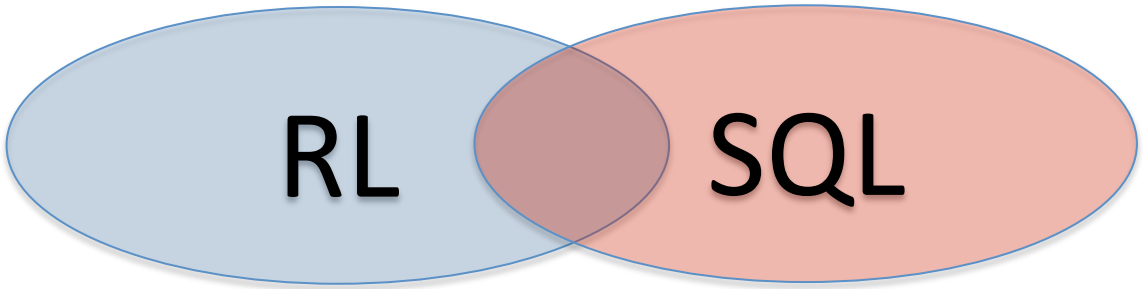
Updates: API

- add-node(node-id, payload)
- remove-node(node-id)
- update-node(node-id, payload)

- add-edge(s-node-id, d-node-id)
- remove-edge(s-node-id, d-node-id)

Graph Query Languages

- Several languages
- Trade-off
  - Expressiveness
  - Execution
- Regular language reachability
  - Used in Horton



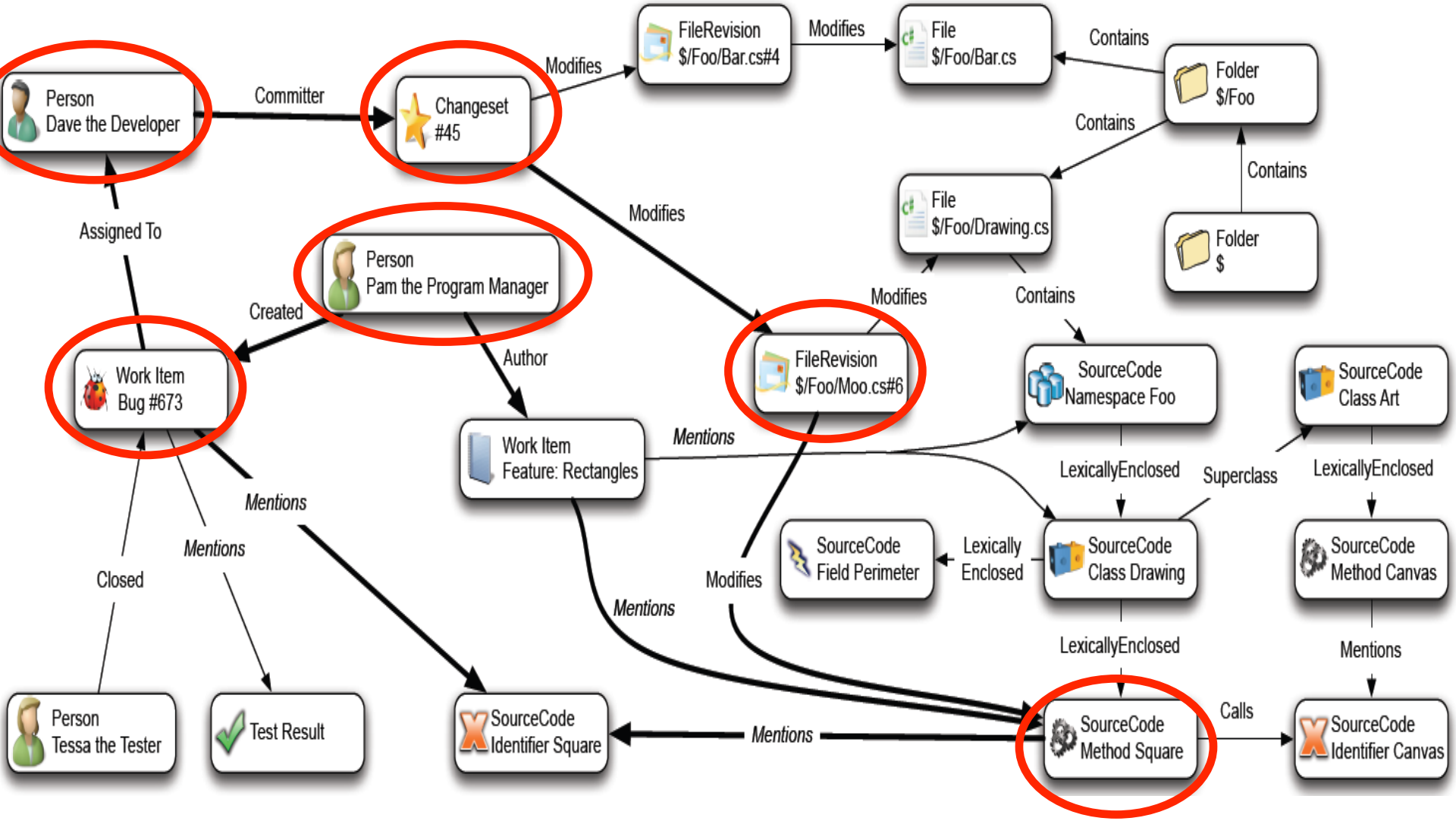
Regular Language

- Query is a regular expression
  - Sequence of node and edge predicates
- Example
  - Find Alice's photos
  - **Photo, tags, Alice**
  - Query =
    - Node: type=photo,
    - Edge: type=tags,
    - Node: type=person, name = Alice
  - Result: matching paths

Query Language Operators

- Projection
  - Alice's photos
  - **SELECT photo FROM photo, tags, Alice**
- OR
  - **(Photo | video), tags, Alice**
- Kleene star
  - Alice's org chart
  - **Alice, (manages, person)***

Example: CodeBook - Graph



Example: CodeBook - Queries

1. **Person**, FileOwner>, **File**, FileOwner<, **Person**
2. **Person**, DiscussionOwner>, **Discussion**, DiscussionOwner<, **Person**
3. **Person**, WorkItemOwner>, **WorkItem**, WorkItemOwner< , **Person**
4. **Person**, Manages<, **Person**, Manages>, **Person**
5. **Person**, WorkItemOwner>, **WorkItem**, Mentions>, **File**, FileOwner<, **Person**
6. **Person**, FileOwner>, **File**, Mentions>, **WorkItem**, Mentions>, **File**, FileOwner<, **Person**

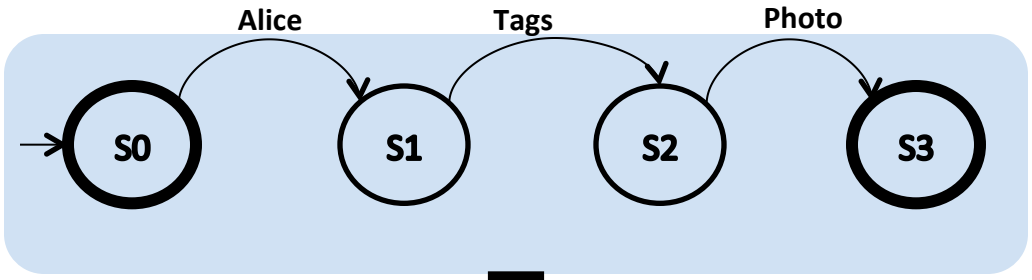
- Who are my colleagues?
- Who is calling my code?
- Who introduced a bug in my code?

Example Execution Engine

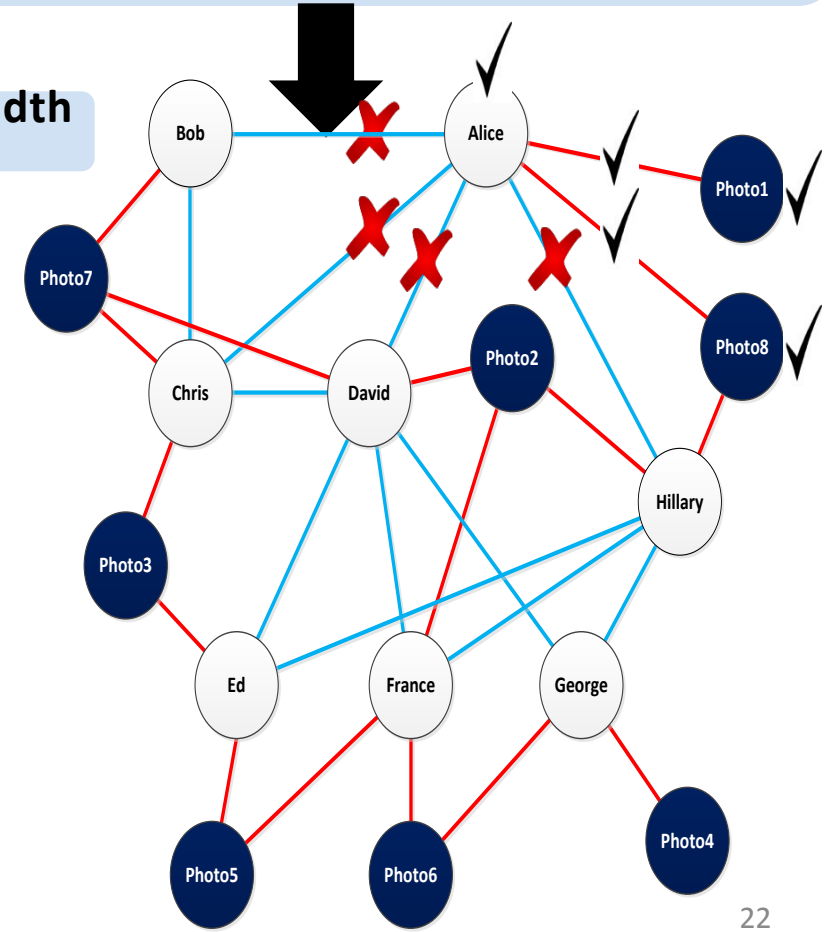
- Executing RL query
  1. Build a FSM
  2. Optimize FSM
  3. Execute FSM using distributed graph traversal

Centralized Query Execution

Alice, Tags, **Photo**



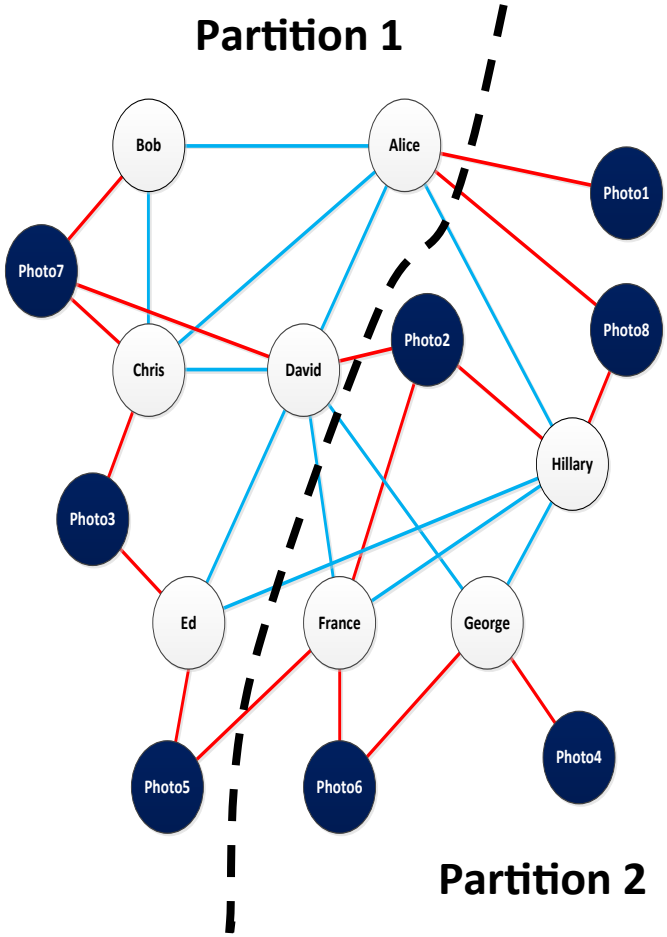
Traversal similar to Breadth First



Answer Paths:
Alice, Tags, **Photo1**
Alice, Tags, **Photo8**

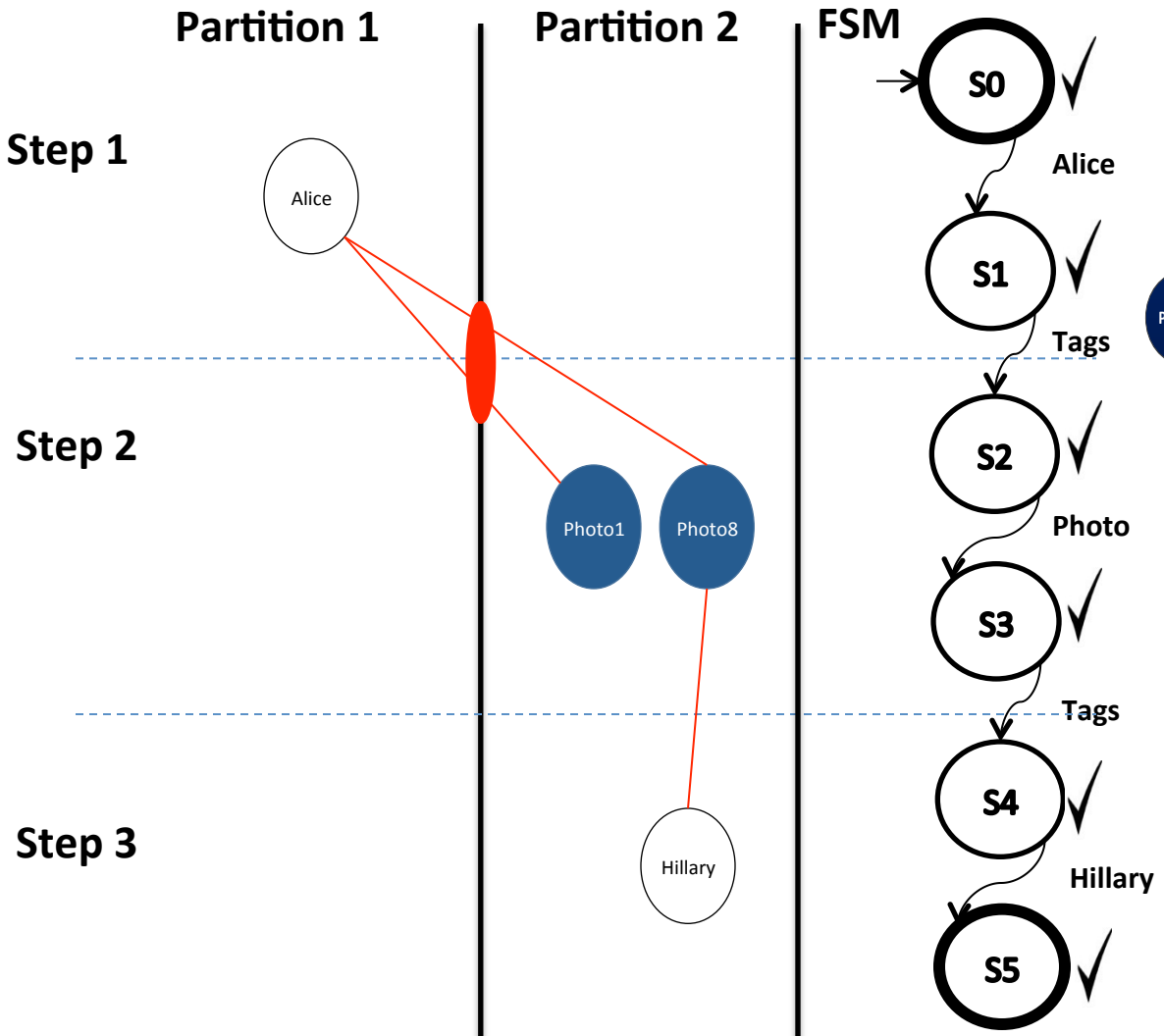
Distributed Query Execution

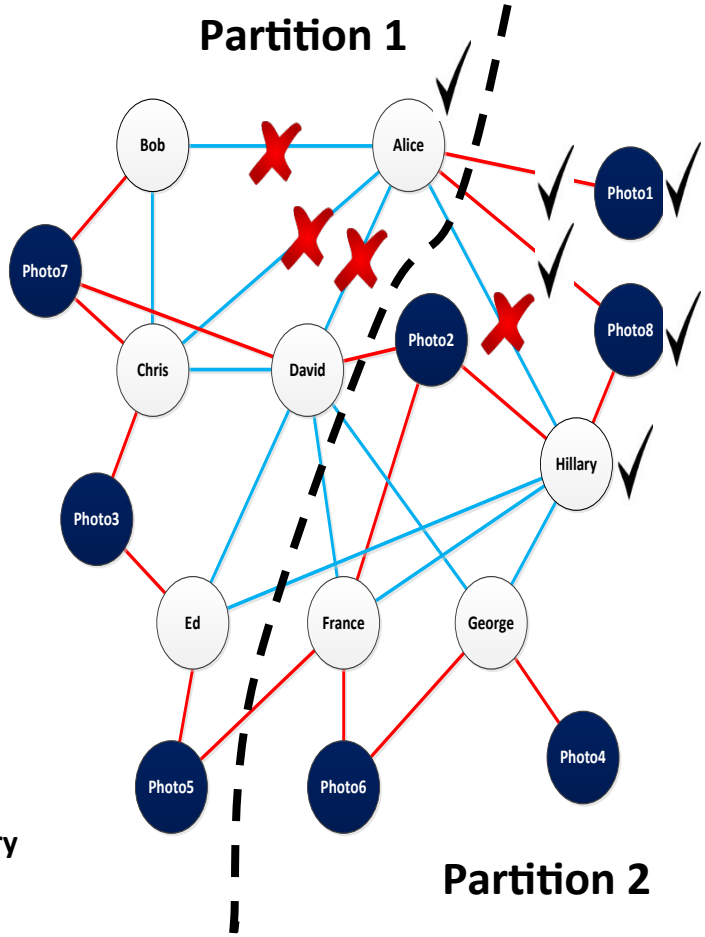
Alice, Tags, **Photo**, Tags, **Hillary**



Distributed Query Execution

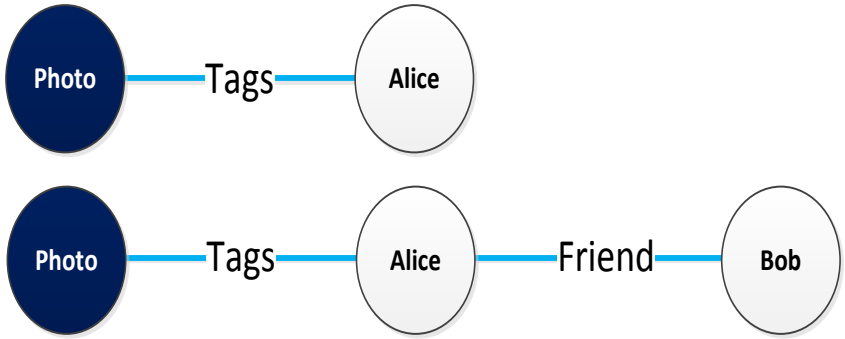
Alice, Tags, **Photo**, Tags, **Hillary**



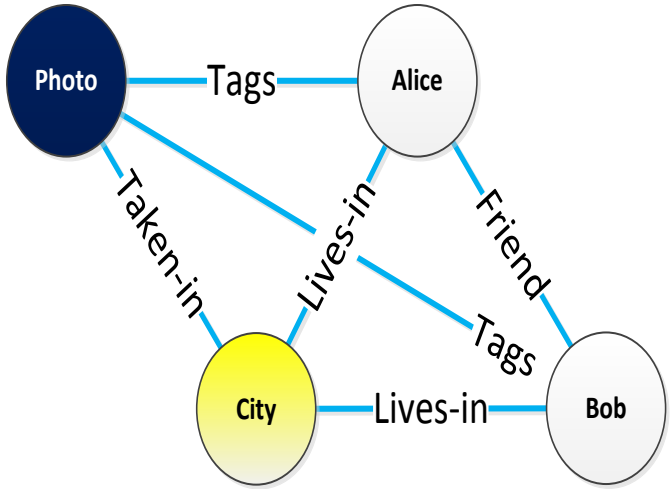


Sub-graph Matching

– **From:** path
  – Sequence of predicates



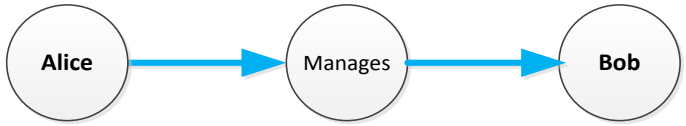
– **To:** sub-graph
  – Graph pattern



– Sub-graph isomorphism

Mappings Are Not Rigid

- Example
  - Edge are entities
    - Types, attributes, ids
  - Supporting edge entities
  - Supporting hyper-nodes and hyper-edges

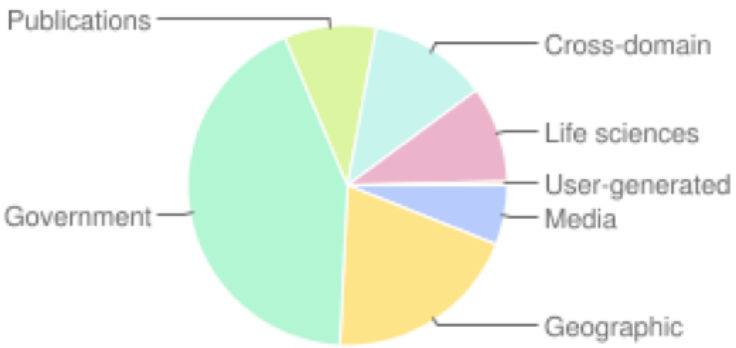


Roadmap

- Intro to Graphs
- Social Networks
  - Data Model
  - Queries
  - Processing
- **Web of Data [WoD]**
  - Data Model
  - Queries
  - Processing
- Systems
- Current Research Directions

Example: Linked open Data[LoD]

- Scale
  - Hundreds of data sets
  - 30B+ tuples
- Queries
  - SPARQL
- Domains



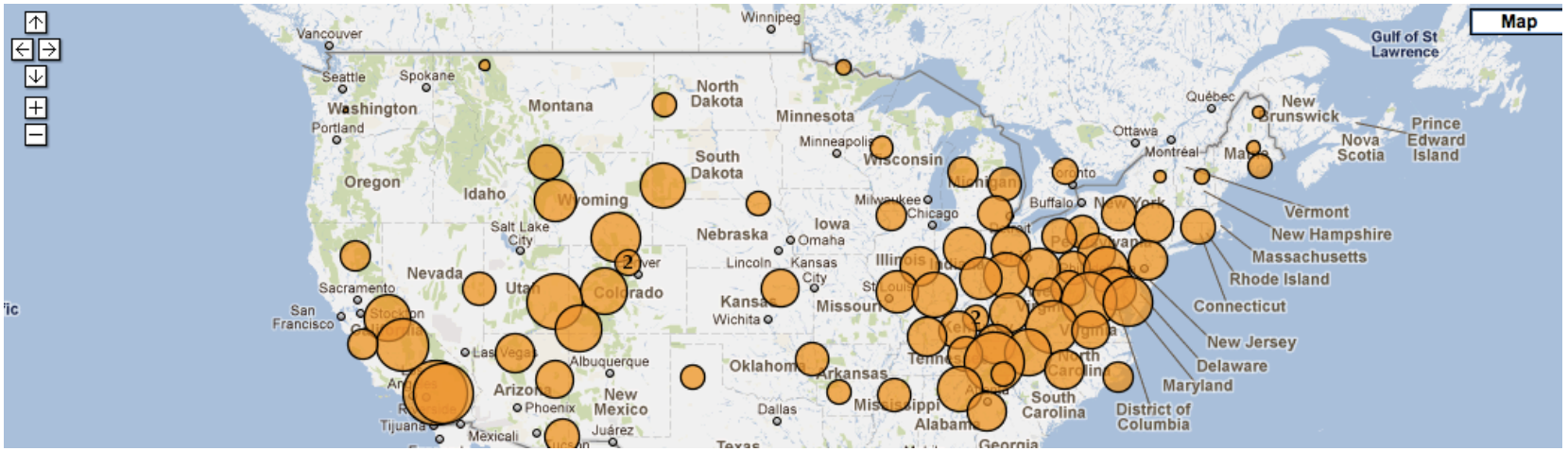
http://www4.wiwiss.fu-berlin.de/lodcloud/state/

Linked Open Data Principles

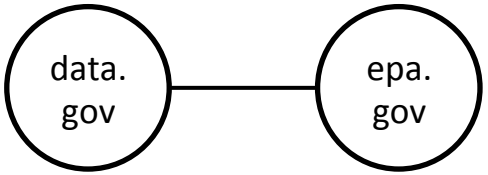
- Four basic principles [Berners-Lee06]
  - Use ***URIs*** to identify things.
  - Use HTTP URIs to ***dereference*** URIs
  - Provide structured data about URI in ***RDF***
  - Include ***links*** to related URIs

LoD Application Example

- DATA.GOV EMPOWERING PEOPLE ozone level visualization



- 2 data sets
  - clean air status [data.gov]
  - Castnet site information [epa.gov]
- 2 SPARQL queries



Web of Data: Data Model (1)

- Structured data
  - Resource Description Framework (RDF) [Manola04]
- **Triples**!

1:subject, 2:predicate, 3:object

ex.: philippe, made, idmesh_paper:

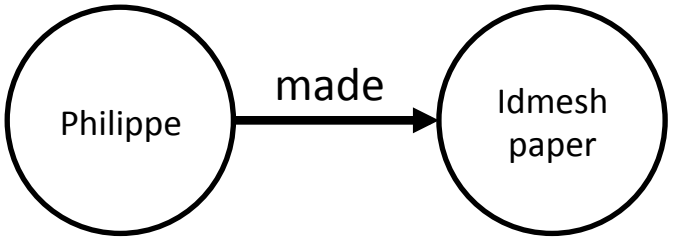
1: http://data.semanticweb.org/person/philippe-cudre-mauroux

2: http://xmlns.com/foaf/0.1/made

3: http://data.semanticweb.org/conference/www/2009/paper/60

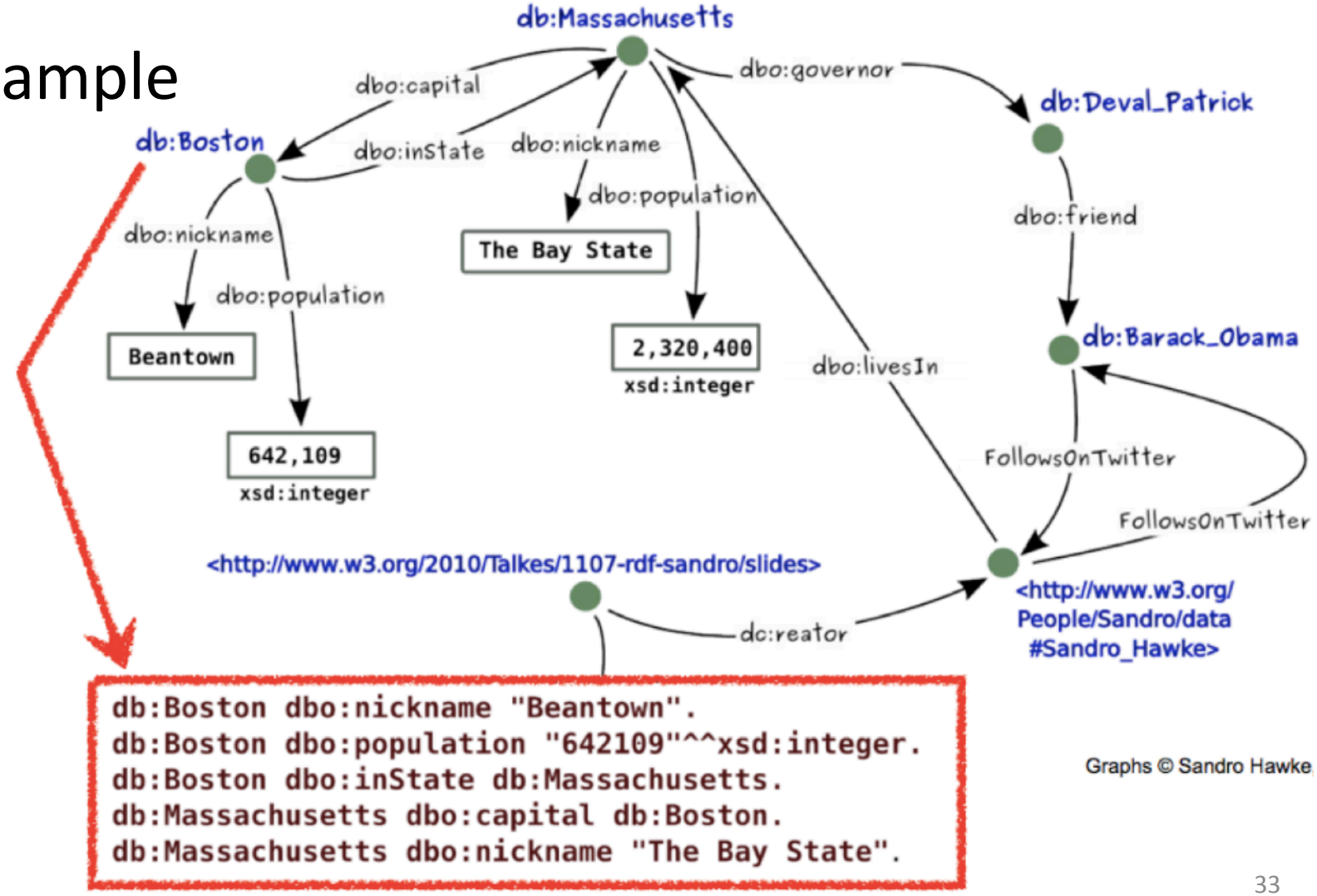
Web of Data: Data Model (2)

- Naturally forms (distributed) **graphs**
- **Nodes**
  - URIs [subjects]
  - URIs / literals [objects]
- **Edges**
  - URIs [predicates]
  - Directed



Web of Data: Data Model (3)

- Example

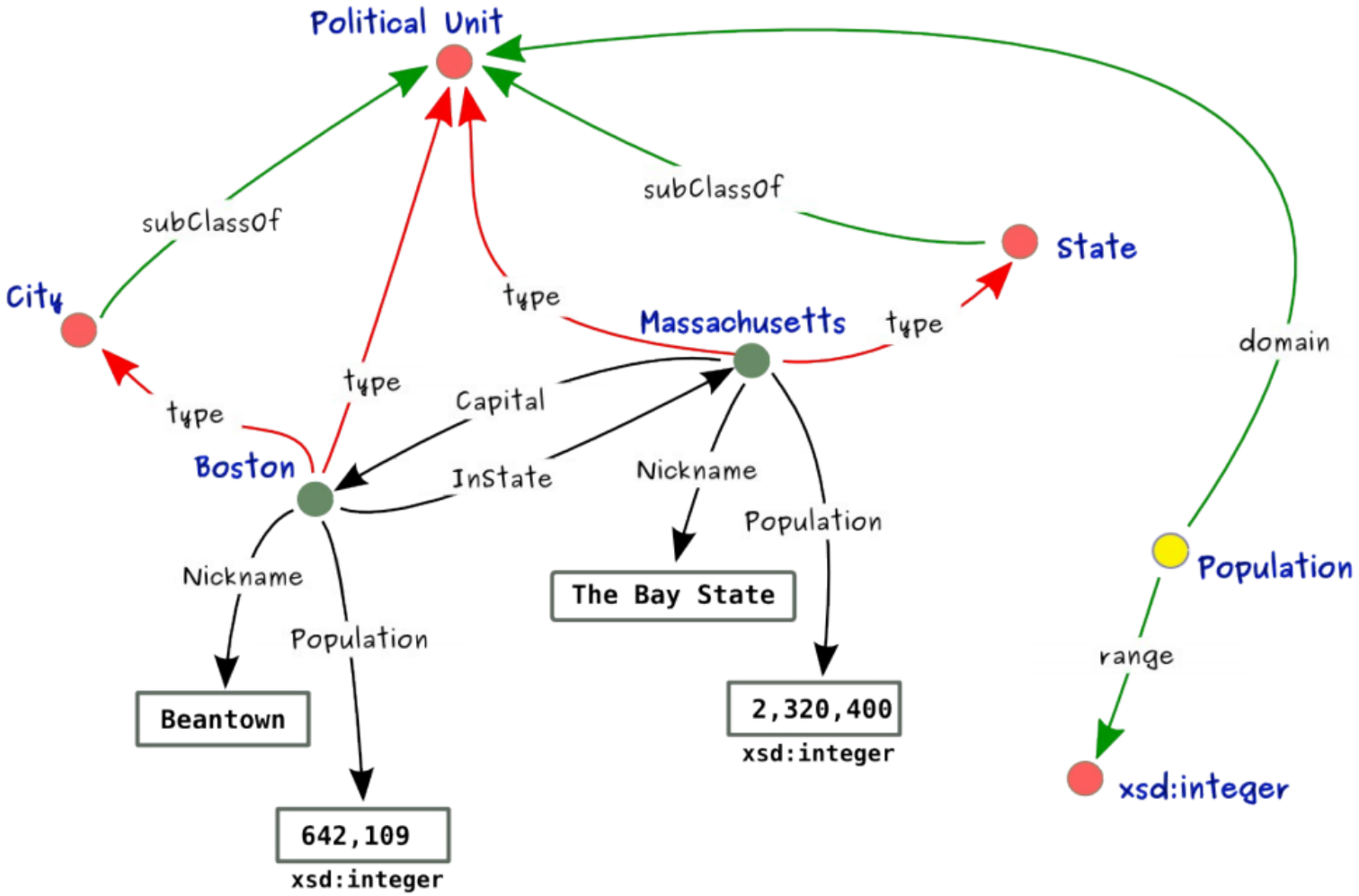


RDF Schemas (RDFS) [Brickley04]

- Classes, inheritance
  - *Class, Property, SubClass, SubProperty*
- Constraints on structure
  - Constraints on subjects (*Domain*)
  - Constraints on objects (*Range*)
- Collections
  - *List*, *Bag*
- Reification

Schemas can be **reused**, **mixed**

RDFS Example



Ontologies (OWL) [W3COWL09]

- Very expressive schemas (ontologies)
- Based on Description Logics
  - Exists in different flavors
- Example: OWL 2 EL axioms:

  class inclusion (SubClassOf)
  class equivalence (EquivalentClasses)
  class disjointness (DisjointClasses)
  object property inclusion (SubObjectPropertyOf) with or without property chains, and data property inclusion (SubDataPropertyOf)
  property equivalence (EquivalentObjectProperties and EquivalentDataProperties),
  transitive object properties (TransitiveObjectProperty)
  reflexive object properties (ReflexiveObjectProperty)
  domain restrictions (ObjectPropertyDomain and DataPropertyDomain)
  range restrictions (ObjectPropertyRange and DataPropertyRange)
  assertions (SameIndividual, DifferentIndividuals, ClassAssertion, ObjectPropertyAssertion, DataPropertyAssertion, NegativeObjectPropertyAssertion, andNegativeDataPropertyAssertion)
  functional data properties (FunctionalDataProperty)
  keys (HasKey)

- **Inference**! ex.: TransitiveObjectProperty(hasAncestor)
  $hasAncestor(x, y) \land hasAncestor(y, z) \rightarrow hasAncestor(x, z)$

RDF Storage (1)

- XML/JSON **Serialization**
  - Exchange format
    - Not meant for humans (ugly)
    - Not meant for DBMSs (verbose)
  - Example:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:db="http://dbpedia.org/resource/">
<rdf:Description rdf:about="http://dbpedia.org/resource/Massachusetts">
      <db:Governor>
            <rdf:Description rdf:about="http://dbpedia.org/resource/Deval_Patrick" />
      </db:Governor>
      <db:Nickname>Bay State</db:Nickname>
            <db:Capital>
                  <rdf:Description rdf:about="http://dbpedia.org/resource/Boston">
                        <db:Nickname>Beantown</db:Nickname>
                  </rdf:Description>
            </db:Capital>
      </rdf:Description>
</rdf:RDF>
```

RDF Storage (2)

- RDFa
  - Embedding RDF information in **HTML** pages
  - Supported by Google, Yahoo, etc
  - Example:

```
<body>
  <div about="http://dbpedia.org/resource/Massachusetts">The Massachusetts governor is
    <span rel="db:Governor">
         <span about="http://dbpedia.org/resource/Deval_Patrick">Deval Patrick
         </span>,
    </span>
    the nickname is "<span property="db:Nickname">Bay State</span>",
    and the capital
    <span rel="db:Capital">
         <span about="http://dbpedia.org/resource/Boston">
          has the nickname "<span property="db:Nickname">Beantown</span>".
         </span>
    </span>
  </div>
</body>
```

RDF Storage (3)

- Various internal formats for DBMSs
  - Giant **triple table** (*triple stores*)
    - |subject|predicate|object|
  - **Property tables**
    - |subject|property1|property2|property3|…|
  - **Sub-graphs**

WoD: Workloads

- Bulk inserts
- **Read-mostly**
  - Node/triple look-ups
  - Distributed entity retrieval queries
  - Sub-graph queries
  - Path queries
  - Inference queries
- Mostly using SPARQL query language [Prud'hommeaux08]
  - Alternatives exist, e.g., “Thread: A Path-Based Query Language” [McDonald11]

SPARQL (1/2)

- Declarative query language for SW data
- SPJ combinations of ***triple patterns***
  - E.g., "Retrieve all students who live in Seattle and take a graduate course"
  - Select ?s Where {
    ?s is_a Student
    ?s lives_in Seattle
    ?s takes ?c
    ?c is_a GraduateCourse }

SPARQL Query Execution

- Typically start from bound variables and performs **self-joins** on giant triple table
  - Select ?s Where {
    ?s is_a Student
    ?s lives_in Seattle
    ?s takes ?c
    ?c is_a GraduateCourse }
  - $\pi_s \, \sigma_{p=\text{“is\_a”} \wedge o=\text{“Student”}}$
    $\bowtie \pi_s \, \sigma_{p=\text{“lives\_in”} \wedge o=\text{“Seattle”}}$
    $\bowtie \pi_s \, (\sigma_{p=\text{“takes”}} \; {}_o\bowtie_s \; \sigma_{p=\text{“is\_a”} \wedge o=\text{“GraduateCourse”}})$

SPARQL (2/2)

- Beyond conjunctions of triple patterns
  - Named graphs
  - Disjunctions
    - UNION
    - OPTIONAL (semi-structured data model)
  - Predicate filters
    - FILTER (?price < 30)
  - Duplicate handling (*bag semantics*)
    - DISTINCT, REDUCED
  - Wildcards
  - *Negation as failure*

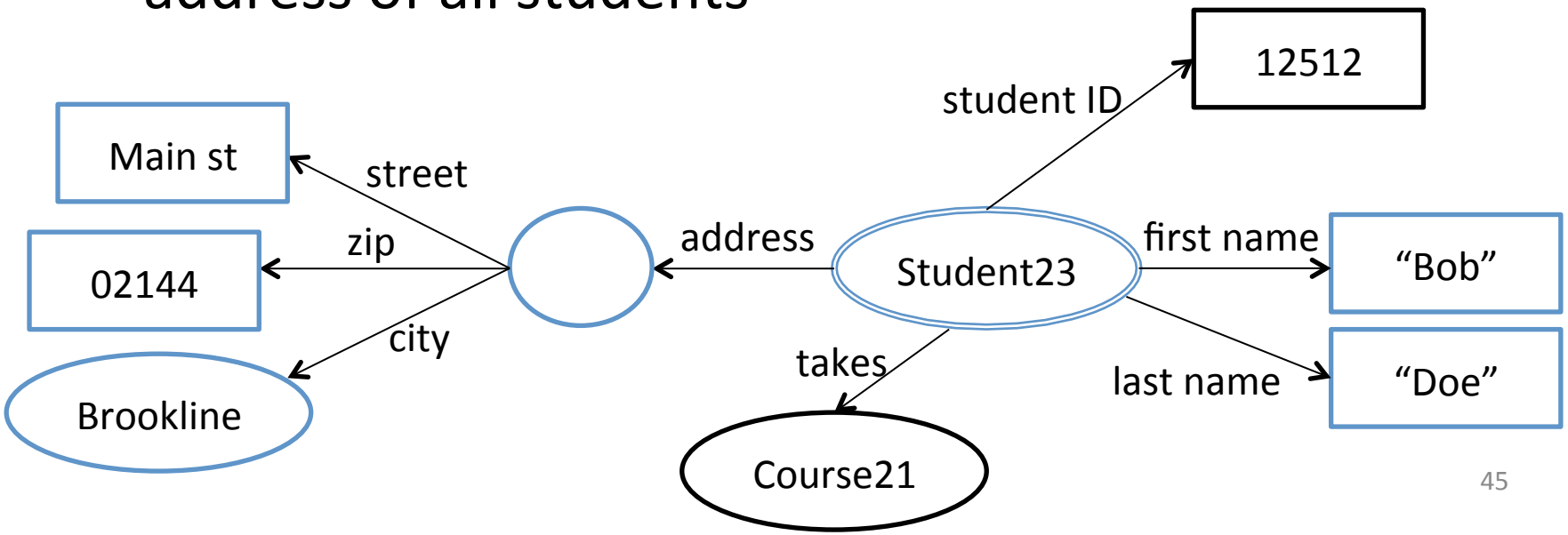
```
WHERE { ?x foaf:givenName  ?name .
        OPTIONAL { ?x dc:date ?date } .
        FILTER (!bound(?date)) }
```

SPARQL 1.1 [Harris11]

- Candidate recommendation
- Adds a whole new set of beasts
  - Aggregates
  - Subqueries
  - Filters
    - EXISTS, NOT EXISTS
  - **Property paths** (? + * ^ / |)
  - **Inference queries**
    - Entailment regimes[Glimm11]

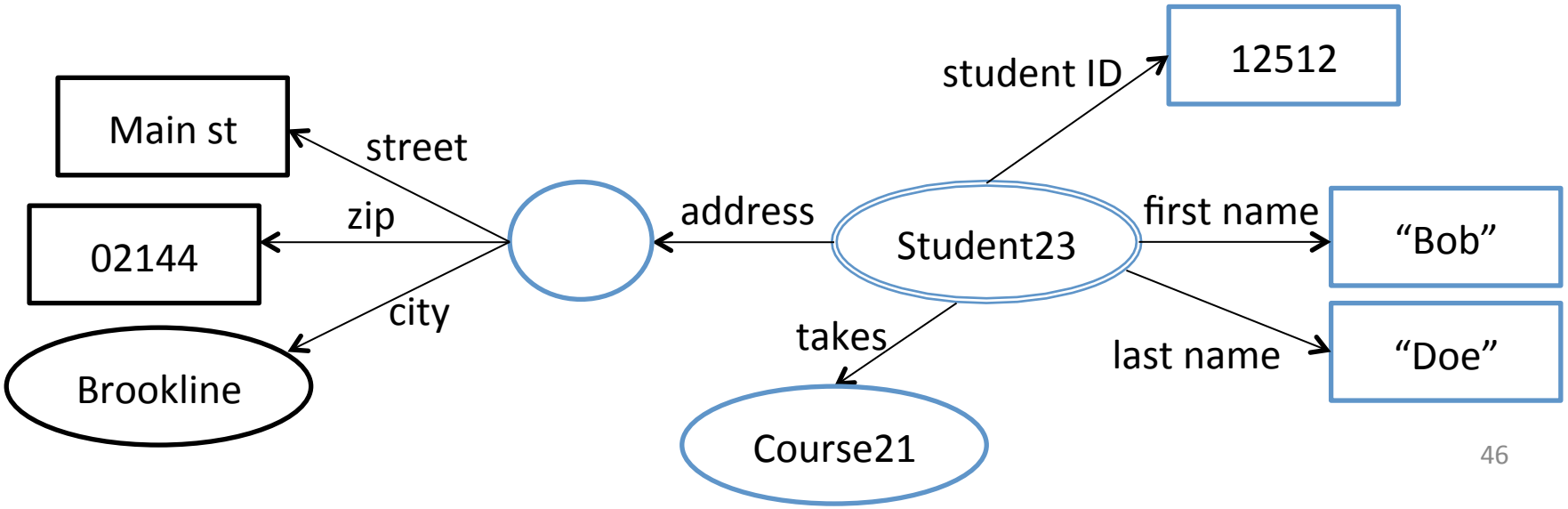
Sub-graph Queries (1)

- ***Molecule*** queries
  - Star-shape sub-queries
  - Combining properties of a given entity
  - E.g., "Retrieve the first name, last name and full address of all students"



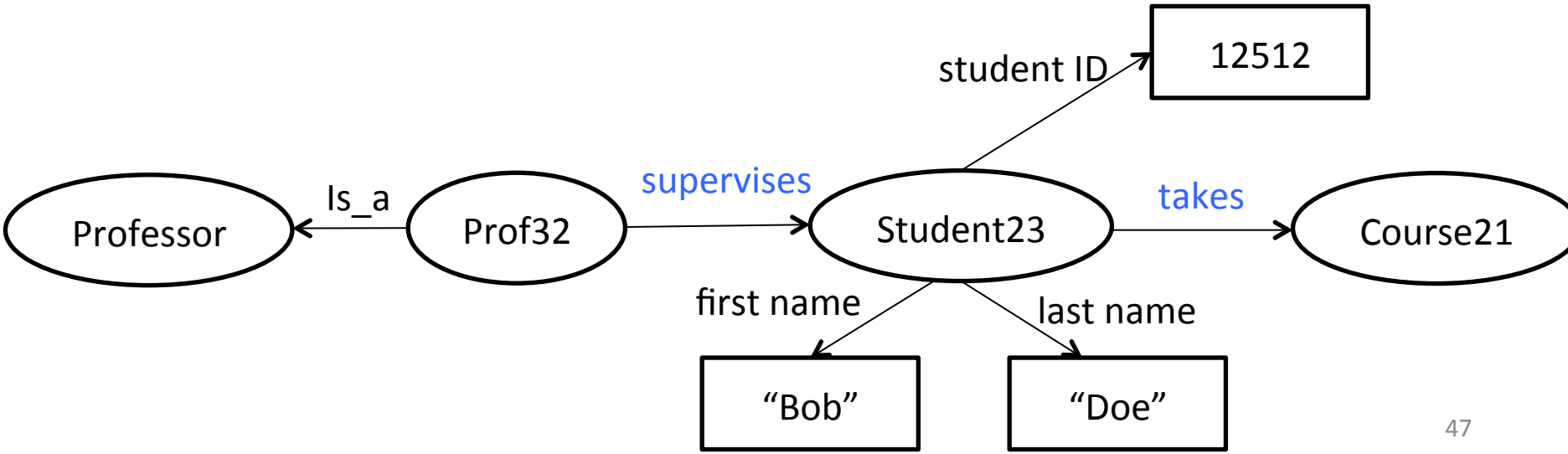
Sub-graph Queries (2)

- ***Scope*** queries
  - Retrieve all triples within a certain scope from a given root node (typically for visualization purposes)
  - E.g., scope 1 from "Student23"



Path Queries (1)

- ***Property path*** queries
  - Queries on series of predicates
- E.g., "find all professors who supervise students following courses"



Path Queries (2)

- Various sub-flavors
  - Frequent path queries
    - For optimization or visualization purposes
    - E.g., “Find the most frequent paths of length 2”
  - Regular expressions for properties (SPARQL 1.1)
    - ? + * ^ / |
    - E.g., “find **reachable** friends through 2 different paths”
      - SELECT * WHERE {
        :John (foaf:friendOf|urn:friend)+ ?friend. }

Inference Queries

- Additional data can be inferred using various sets of logical rules
- Specify which ones to use by entailment regimes [Glimm11]
  - **RDF Schema** has 14 entailment rules
    - E.g., (p,rdfs:domain,x) && (u, p, y)
      => (u rdf:type x)
  - **OWL 2 RL** has 70+ entailment rules.
    - E.g., (p,rdf:type,owl:FunctionalProperty) && (x, p, y1) && (x,p,y2)
      => (y1, owl:sameAs, y2)

RDF/OWL Benchmarks

- Large choice of benchmarks, e.g., focusing on
  - Large knowledge bases (LUBM) [Guo05]
  - Library search and visualization (Barton) [Abadi07]
  - Linked Open Data (BSBM) [Bizer11]
  - RDF Analytics (BowlognaBench) [Demartini11]

Roadmap

- Intro to Graphs
- Social Networks
  - Data Model
  - Queries
  - Processing
- Web of Data [WoD]
  - Data Model
  - Queries
  - Processing
- **Systems**
- Current Research Directions

WoD Systems

- Many interesting approaches/systems
- Only very small sample here

RDF-3X [Neumann08]

- Max Planck Institut für Informatik
  - Thomas Neumann & Gerhard Weikum
- Open-Source
- Triple-table storage
- No turning knobs
  - Workload-independent physical design
- Reduced instruction set
  - Merge-joins over sorted lists

RDF-3X: Storage and Indexing

- Dictionary encoding of all literals
- Exhaustive-indexing approach
  - Clustered B+-trees on all six SPO permutations (see also Hexastore [Weiss08])
  - Also on six binary and three unary projections
    - Indexing count aggregates
  - Support for versioning by two additional fields for each triple
    - Created and deleted timestamps

RDF-3X: Compression

- Triples stored at the leaves of the tree
  - Value1, Value2, Value3
  - Neighboring triples are often very similar
    - Value1 and Value2 the same
- Leaf pages use byte-wise compression
  - Store deltas for each value

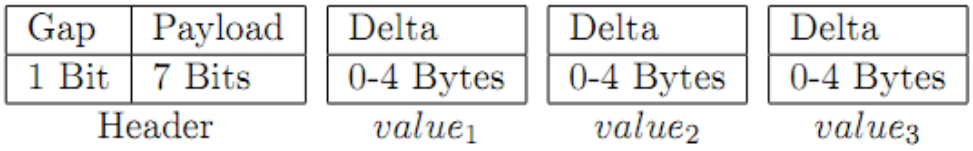


Figure 3.1: Structure of a compressed triple

- Triple often encoded in a single byte

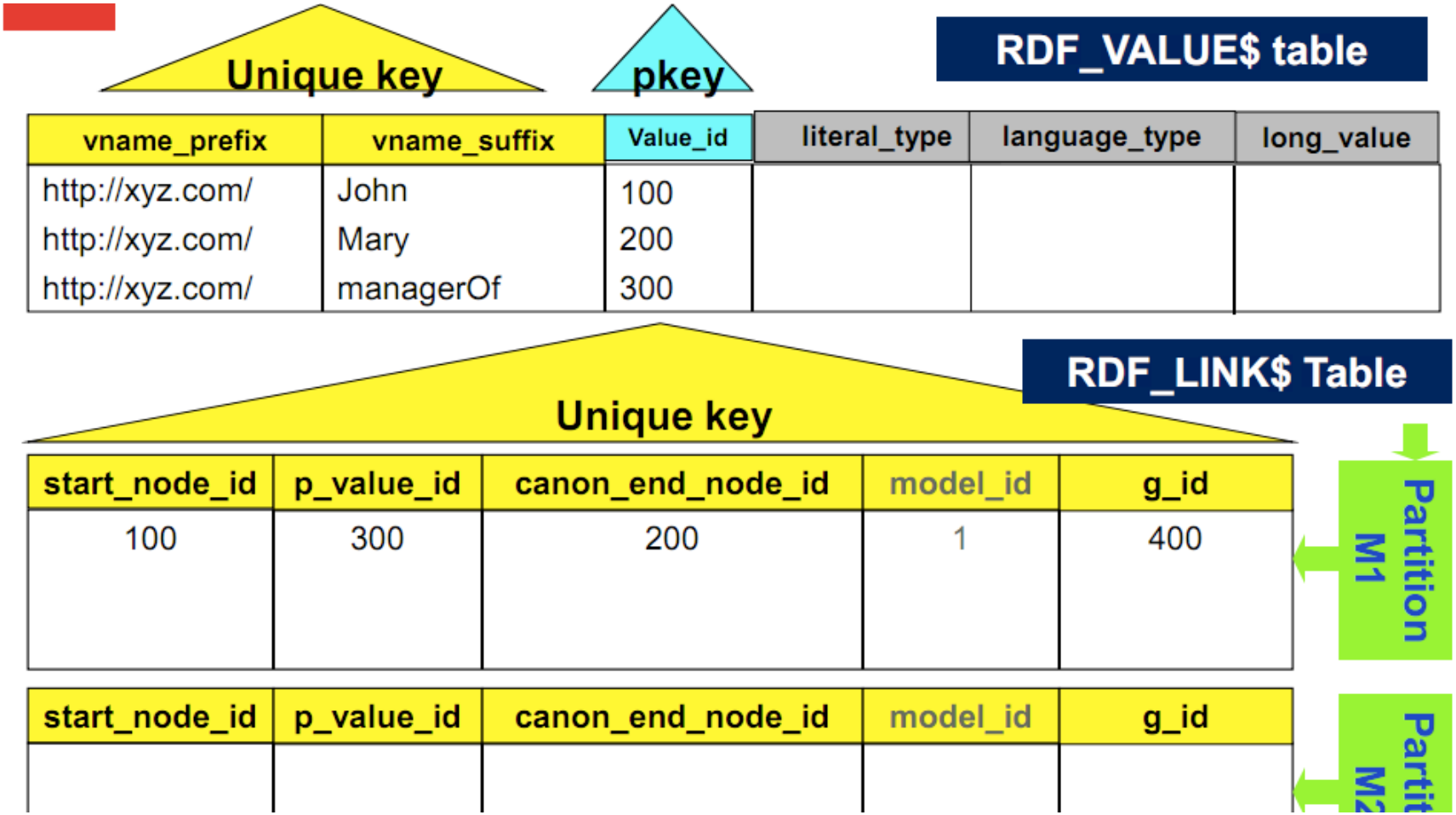
RDF-3X: Query Optimization

- Triple pattern
  - Single range scan
- Multiple triple patterns
  - Order-preserving merge-joins
  - Join ordering
    - Dynamic programming; tradeoff between
      - Use literals in triple patterns as index prefix
      - Produce interesting orders for subsequent merge-joins
    - Plan pruning based on estimated execution costs
    - Costs based on selectivity estimates
      - Histograms
      - Join-path cardinalities

Oracle Semantic Web Technologies

- Part of Oracle Database 11g
  - http://www.oracle.com/technetwork/database/options/semantic-tech/index.html
- RDF data stored in two main tables
  - **Nodes, edges**
- Optional **B-tree** indexing
  - add_sem_index(column_list)
- Mixing SQL and SPARQL
  - SEM_MATCH
- Efficient **inference**

Oracle: Triple Storage



Oracle: Inference

- Inference done using forward chaining
  - Triples inferred and stored ahead of query time
- Various profiles supported
  - RDFS, OWL 2 RL, SKOS, subset of OWL 2 EL
- Large scale owl:sameAs handling
  - Compact materialization of owl:sameAs closure
- User-defined SWRL-like rules
- Incremental, parallel reasoning

OWLIM [Bishop11]

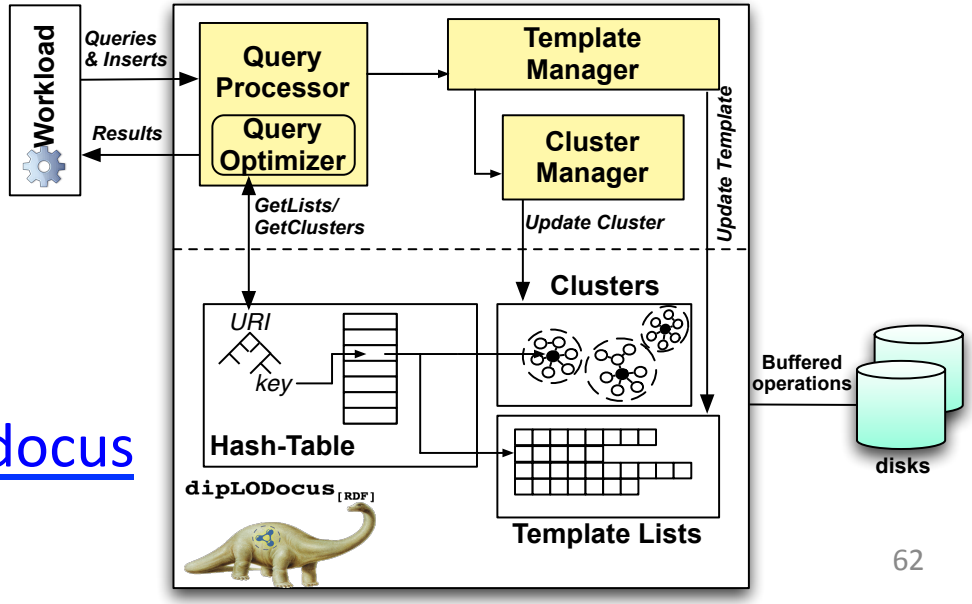
- Commercial, native RDF/OWL DBMS
  - Comes in different flavors
    - Main-memory
    - Disk-based
      - Persistency through N-triple files
  - Scalable forward-chaining inference
  - Several interesting search/ranking features

OWLIM: Searching & Ranking

- **Full-text search** support
  - Arbitrary string operations in SPARQL
- RDF **ranking**
  - Relevance of entities based on their interconnectedness
- RDF "priming"
  - Contextualized query processing based on starting nodes
- **Publish/subscribe** mechanisms

dipLODocus[RDF] [Wylot11]

- Blazing-fast, hybrid storage system for RDF
  - Aggressive compression (lexicographical tree)
  - Pre-computed joins (declarative molecule storage)
  - Efficient support for aggregate/analytic operations on literals



http://diuf.unifr.ch/xi/diplodocus

Graph Systems

- Relational: SQL
- Triple store: SPARQL
- Custom graph server: API

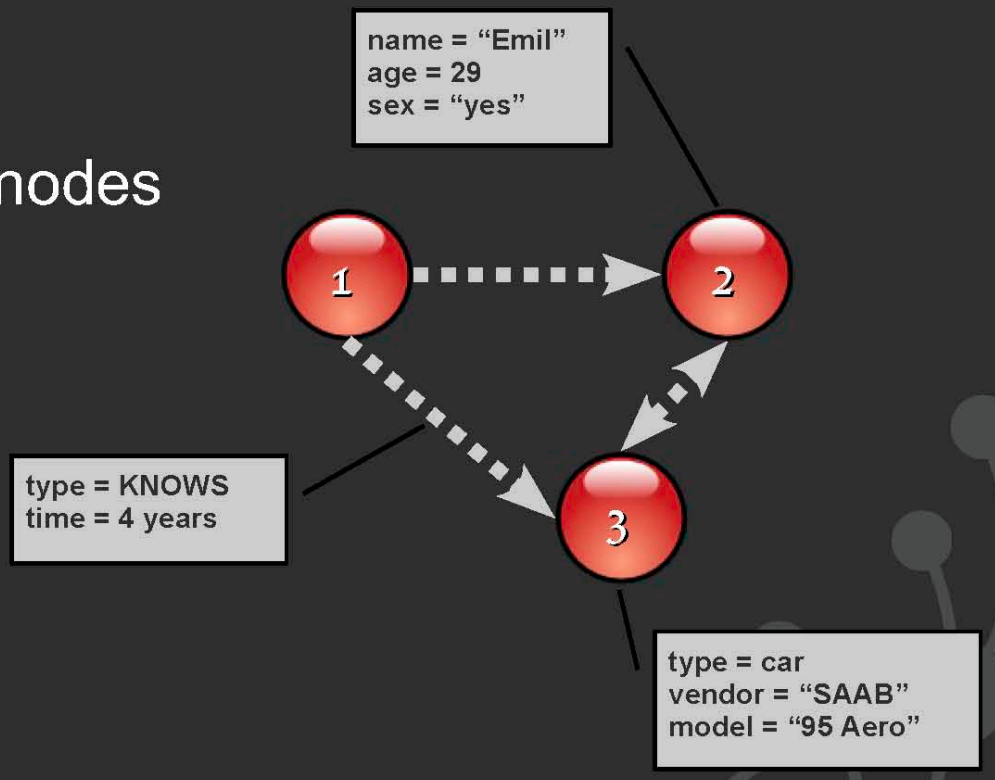
Graph Servers

- Neo4j
- InfiniteGraph
- Google Pregel
- Microsoft Horton & Trinity
- DEX
- …

The Neo4j model: Property Graph

Core abstractions:

- Nodes
- Relationships between nodes
- Properties on both



Building a node space (core API)

```
GraphDatabaseService graphDb = ... // Get factory


// Create Thomas 'Neo' Anderson
Node mrAnderson = graphDb.createNode();
mrAnderson.setProperty( "name", "Thomas Anderson" );
mrAnderson.setProperty( "age", 29 );

// Create Morpheus
Node morpheus = graphDb.createNode();
morpheus.setProperty( "name", "Morpheus" );
morpheus.setProperty( "rank", "Captain" );
morpheus.setProperty( "occupation", "Total bad ass" );

// Create a relationship representing that they know each other
mrAnderson.createRelationshipTo( morpheus, RelTypes.KNOWS );
// ...create Trinity, Cypher, Agent Smith, Architect similarly
```

Building a node space

```
GraphDatabaseService graphDb = ... // Get factory
Transaction tx = graphdb.beginTx();

// Create Thomas 'Neo' Anderson
Node mrAnderson = graphDb.createNode();
mrAnderson.setProperty( "name", "Thomas Anderson" );
mrAnderson.setProperty( "age", 29 );

// Create Morpheus
Node morpheus = graphDb.createNode();
morpheus.setProperty( "name", "Morpheus" );
morpheus.setProperty( "rank", "Captain" );
morpheus.setProperty( "occupation", "Total bad ass" );

// Create a relationship representing that they know each other
mrAnderson.createRelationshipTo( morpheus, RelTypes.KNOWS );
// ...create Trinity, Cypher, Agent Smith, Architect similarly
tx.commit();
```

Code (2): Traversing a node space

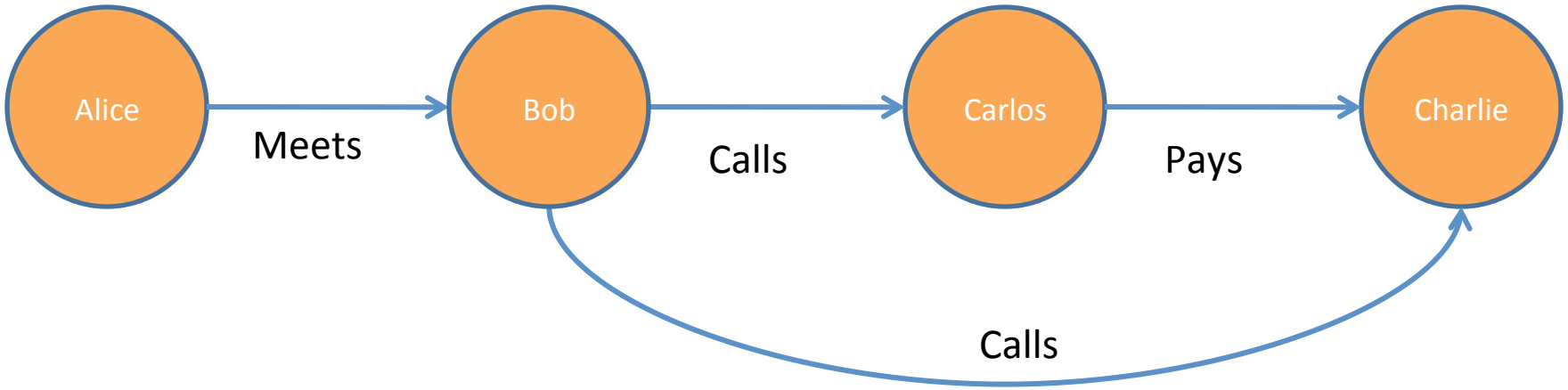
```
// Instantiate a traverser that returns Mr Anderson's friends
Traverser friendsTraverser = mrAnderson.traverse(
    Traverser.Order.BREADTH_FIRST,
    StopEvaluator.END_OF_GRAPH,
    ReturnableEvaluator.ALL_BUT_START_NODE,
    RelTypes.KNOWS,
    Direction.OUTGOING );

// Traverse the node space and print out the result
System.out.println( "Mr Anderson's friends:" );
for ( Node friend : friendsTraverser )
{
    System.out.printf( "At depth %d => %s%n",
        friendsTraverser.currentPosition().getDepth(),
        friend.getProperty( "name" ) );
}
```

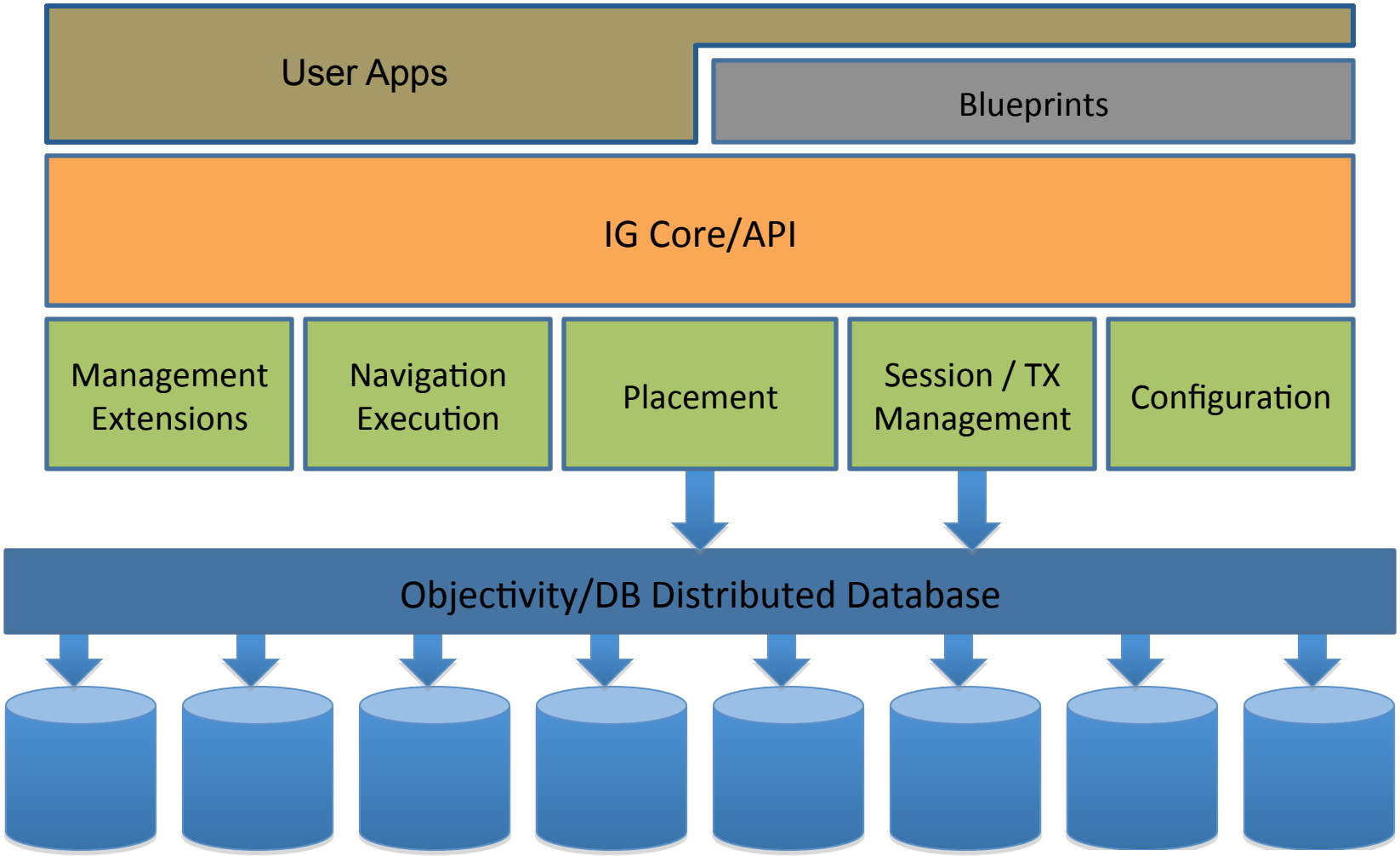
Infinite Graph

```
Vertex alice = myGraph.addVertex(new Person("Alice"));
Vertex bob = myGraph.addVertex(new Person("Bob"));
Vertex carlos = myGraph.addVertex(new Person("Carlos"));
Vertex charlie = myGraph.addVertex(new Person("Charlie"));

alice.addEdge(new Meeting("Denver", "5-27-10"), bob);
bob.addEdge(new Call(timestamp), carlos);
carlos.addEdge(new Payment(100000.00), charlie);
bob.addEdge(new Call(timestamp), charlie);
```



Infinite Graph Architecture



InfiniteGraph

Active Research Topics

- Transactions
- Partitioning
- Indexing
- Parallel execution

Partitioning A Large Graph

- Motivation
  - Graph too big for one machine
- Solutions
  - Hash partition
  - METIS
  - Local approaches
  - Hierarchical approaches

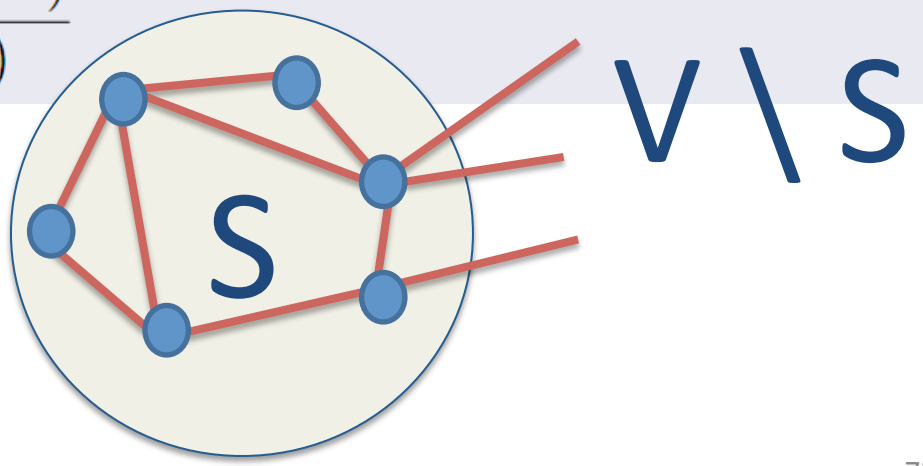
Conductance

- Graph G(V,E), search for subset S of V
- V = S union V\S
- Find S with small conductance

$$\textbf{volume}: \quad \mathrm{vol}(S) = \text{sum of the degrees of vertices in } S.$$

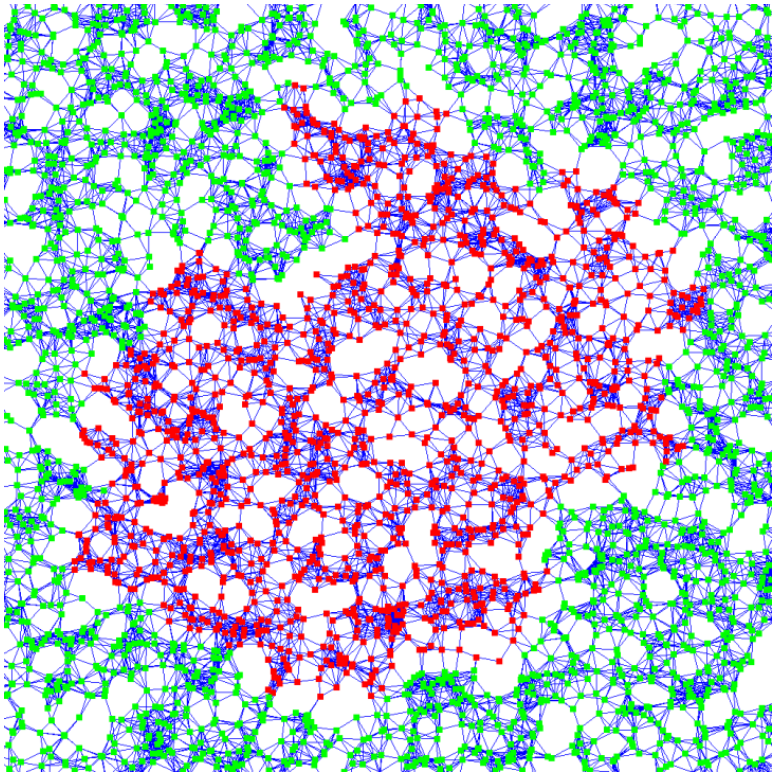
$$\textbf{cutsize}: \quad e(S, V \setminus S) = \text{number of edges between } S \text{ and } V \setminus S.$$

$$\textbf{sparsity}: \quad \phi(S) = \frac{e(S, V \setminus S)}{\mathrm{vol}(S)}$$



Evolving set partitioning algorithm

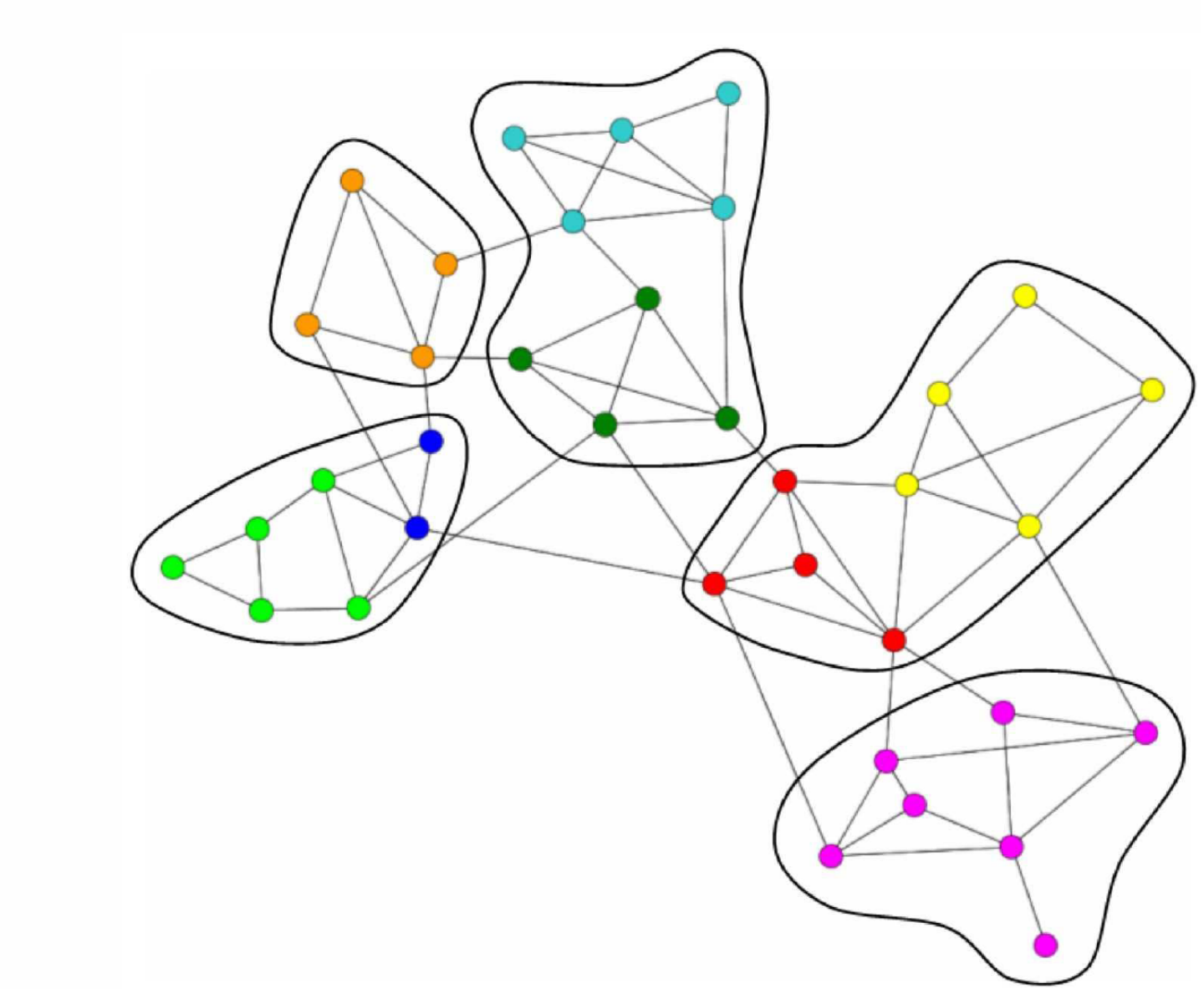
- Randomized algorithm
- Approximation guarantee
- **Local partitioning**



Hierarchical Partitioning

- Hierarchical communities
- Modularity metric
  - Each node joins the neighbor that maximizes modularity

Hierarchical Partitioning



Transactions

- Motivation
  - Correctness
    - Simplifies applications
- Workload characteristics
  - Dominated by reads
  - Small updates

Centralized Transactions

- Solutions
  - Locking schemes
  - Optimistic concurrency control
    - Multi-versioning: snapshot isolation
    - Distributed transactions

Distributed Transactions

- Motivation
  - Graph too large for one machine
  - Graph is partitioned
  - Replication becomes requirement
    - Availability
    - Scalability

WoD: Current Research Directions

- Read-Write Linked Data
- Large-Scale Inference/Query Processing
- Publication of Linked Data
- Entity Matching

(1) Read-Write Linked Data

- How to handle updates / transactions?
- **Protocols**
  - HTTP PUT to overwrite file [Berners-Lee10]
  - SPARQL update [Gearon11]
- Updates & transaction at the **back-end**
  - See for instance x-RDF-3x
- Also interesting problems relating to lineage
  - Dublin Core, W3C Provenance Group, OPM…

(2) Large-Scale Inference/Query Processing

- Large-scale **inference**
  - Old topic, hard
  - Inference onlarge A-Boxes (instances)
  - Distributed inference on heterogeneous, conflicting data sets
- Distributed **query processing**
  - Partitioning/caching triples
  - Optimizing queries across *N* SPARQL end-points

(3) Publication of Linked Data

- From websites/text
  - Entity extraction, NLP
- From relational databases
  - Rel2rdf
- Knowledge elicitation
  - Crowdsourcing

Entity Matching

- The great thing about **unique identifiers** is that there are **so many** to choose from
  - ID jungle!
  - *Hundreds* of identifier for one referent
  - ➠ Matching URIs at LoD scale

WoD: To Go Further

- 1-day tutorial on the Semantic Web and WoD
  - http://people.csail.mit.edu/pcm/SemWebTutorial.html
- International Semantic Web Conf. 2011
  - http://iswc2011.semanticweb.org/
- List of large triple stores
  - http://www.w3.org/wiki/LargeTripleStores
- Some benchmarks & results for triple stores
  - http://www.w3.org/wiki/RdfStoreBenchmarking

References (1)

[Manola04] Frank Manola and Eric Miller (Eds): RDF Primer. W3C Recommendation (2004). http://www.w3.org/TR/rdf-primer/

[Brickley04] Dan Brickley and R.V. Guha (Eds.): RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation (2004). http://www.w3.org/TR/rdf-schema/

[W3COWL09] W3C OWL Working Group: OWL 2 Web Ontology Language Document Overview. W3C Recommendation (2009). http://www.w3.org/TR/owl2-overview/

[Berners-Lee06] Tim Berners-Lee: Design Issues: Linked Data. http://www.w3.org/DesignIssues/LinkedData

[McDonal11] Glenn McDonald: Thread: A Path-Based Query Language for Graph Databases. SemTech 2011. http://semtech2011.semanticweb.com/uploads/handouts/SemTech2011_Thread_4144_3497.pdf

[Prud'hommeaux08] Eric Prud'hommeaux and Andy Seaborne: SPARQL Query Language for RDF. W3C Recommendation (2008). http://www.w3.org/TR/rdf-sparql-query/

[Glim11] Birte Glimm and Chimezie Ogbuji (Eds): SPARQL 1.1 Entailment Regimes. W3C Working Draft (2011). http://www.w3.org/TR/sparql11-entailment/

[Harris11] Steve Harris and Andy Seaborne: SPARQL 1.1 Query Language. W3C Working Draft (2011). http://www.w3.org/TR/sparql11-query/

[Berners-Lee10] Tim Berners-Lee: Design Issues: Read-Write Linked Data. http://www.w3.org/DesignIssues/ReadWriteLinkedData.html

References (2)

[Ogbuji11] Chimezie Ogbuji. SPARQL 1.1 Graph Store HTTP Protocol. W3C Working Draft (2011). http://www.w3.org/TR/sparql11-http-rdf-update/

[Gearon11] Paul Gearon, Alexandre Passant, and Axel Polleres: SPARQL 1.1 Update. W3C Working Draft (2011). http://www.w3.org/TR/2011/WD-sparql11-update-20110512/

[Abadi07] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach: Using The Barton Libraries Dataset As An RDF benchmark. MIT-CSAIL-TR-2007-036 (2007).

[Guo05] Yuanbo Guo, Zhengxiang Pan and Jeff Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics 3(2), 2005.

[Bizer11] Chris Bizer et al.: The Berlin SPARQL Benchmark (BSBM). http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/

[Demartini11] Gianluca Demartini, Iliya Enchev, Joël Gapany, and Philippe Cudré-Mauroux: BowlognaBench—Benchmarking RDF Analytics. SIMPDA 2011.

[Neumann08] Thomas Neumann and Gerhard Weikum: RDF-3X: a RISC-style engine for RDF. PVLDB 1(1), 2008.

[Weiss08] Cathrin Weiss and Panagiotis Karras and Abraham Bernstein:
Hexastore: sextuple indexing for semantic web data management. PVLDB 1(1), 2008.

[Bishop11] Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov: OWLIM: A family of scalable semantic repositories. Semantic Web Journal 2(1), 2011.

[Wylot11] Marcin Wylot, Jige Pont, Mariusz Wisniewski, and Philippe Cudre-Mauroux: dipLODocus[RDF]--Short and Long-Tail RDF Analytics for Massive Webs of Data. ISWC 2011.

Acknowledgements

- **Orleans team (MSR):**


Alan Geller


Jim Larus


Sergey Bykov


CJ Williams


Jorgen Thelin


Suyash Sinha


Gabriel Kliot


Ravi Pandya


Yuxiong He

- **Academic collaborators:**
  - Jiaqing Du (EPFL), Mohamed Fathallah (MSR), Sherif Sakr (NICTA), Mohamed Sarwat (UMN), Willy Zwaenepoel (EPFL)
- **The whole eXascale Infolab team @ U. Fribourg**
  - **http://diuf.unifr.ch/xi/**

Conclusions

- New application domains require new systems
  - New platforms for Social networks & the WoD
- In the future we expect increasing convergence between social networks and the Web of data
  - Data Models: SNs models are richer and more diverse
    - But as we pointed out they can be mapped onto RDF
  - Queries: Nothing standard for SNs, SPARQL++ (i.e., with reachability) might well take over
  - Graph Systems are already transactional for SNs
    - Systems are increasingly focusing on transactions for WoD