

Accessing and Querying Semantic Web Data

Ora Lassila, Ph.D

Principal Technologist, Nokia Location & Commerce
Elected Member, W3C Advisory Board

Invited talk @

New Trends in Soft Computing and Intelligent Systems

University of Granada, Spain

2012-02-17

Speaker background: Ora Lassila

- Employers (current and former):
 - “big data” analytics group @ Nokia’s “Location & Commerce” unit
 - Nokia Research Center, Nokia Venture Partners
 - MIT, CMU, Helsinki University of Technology
 - (own) startup, etc.
 - member of various advisory boards
- Education:
 - Ph.D in Computer Science from Helsinki University of Technology
- Some (dubious) achievements:
 - co-author of the original RDF standard (core Semantic Web specification)
 - co-author of the highest cited Semantic Web article
 - longest serving elected member of W3C Advisory Board
 - designer/developer of KR system on NASA’s “Deep Space 1” probe
 - grand prize winner of the 1989 Usenix Obfuscated C Code Contest

Outline of the lecture

1. Introduction to the Semantic Web, graphs and RDF
 - just to make sure we get the basics straight...
2. Different approaches to querying data
 - a look into SPARQL and some other query languages
3. Programming, (data) semantics and the Semantic Web
 - must explain why some of this stuff is actually difficult (and these difficulties are sometimes hard to spot)
4. Questions
 - maybe even answers...

What this lecture is **NOT**

- A comprehensive tutorial of RDF
 - however, I will cover some of the basics
- A comprehensive tutorial of SPARQL
 - again, I will cover some of the basics only

Introduction to Semantic Web, Graphs and RDF

Original Semantic Web vision

[Berners-Lee, Hendler & Lassila 2001]

- (Billed as) the next generation of the World Wide Web
- Make Web content amenable to automated processing
 - interpretation by machines
 - (most content out there is for human consumption)



- Implies the use of
 - artificial intelligence
 - agents

Original Semantic Web vision

[Berners-Lee, Hendler & Lassila 2001]

- (Billed as) the next generation of the World Wide Web
 - Make Web content amenable to automated processing
 - interpretation by machines
 - (most content out there is for human consumption)
- ↓
- Implies the use of
 - artificial intelligence
 - agents

- Next generation of personal computing
 - Computers working on behalf of users
 - (current usage: as tools)
 - more autonomy, handling unanticipated situations
- ↓
- Implies the use of
 - artificial intelligence
 - agents

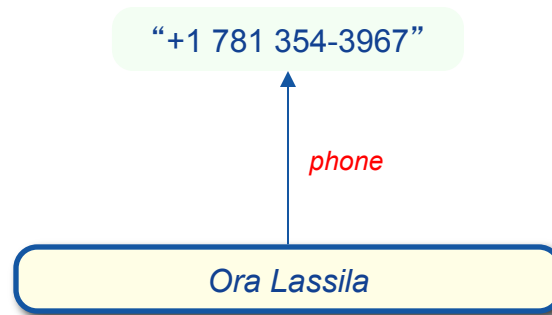
Why is it important to understand the SW vision?

- Semantic Web is ultimately about how we want to build information systems, and how we want information technology to serve people, with some key challenges:
 1. Access to data
 - how to represent the world, how to query
 - how to handle access control, ownership, etc. (= how to gain access to data in the first place)
 2. Ability to flexibly handle unanticipated situations
 3. Matching the richness of data with the expressiveness of user interaction and user experience
- The Semantic Web (vision) should not be considered in isolation, but as part of a broader vision that includes pervasive computing, context awareness, etc.
- Parts of the Semantic Web vision, as well as the constituent technologies, are important and useful for many things (new applications, new services, etc.)
 - e.g., “Linked Data”

Graphs... why graphs?

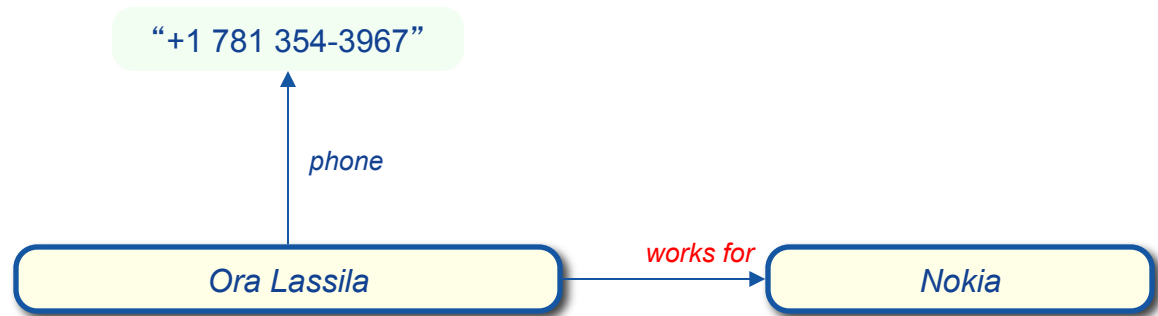
- The universality of graphs (as a data structure) underlies Semantic Web data
- It could be argued that (much of) computer science reduces to graph theory and algorithms on graphs. Really.
- This is not a class about graph theory, but some basic understanding of what graphs are **is important**

Graphs: a natural way to represent the world



- “Ora Lassila’ s **phone** is +1 781 354-3967”

Graphs: a natural way to represent the world



- “Ora Lassila’ s phone is +1 781 354-3967”
- “Ora Lassila **works for** Nokia”

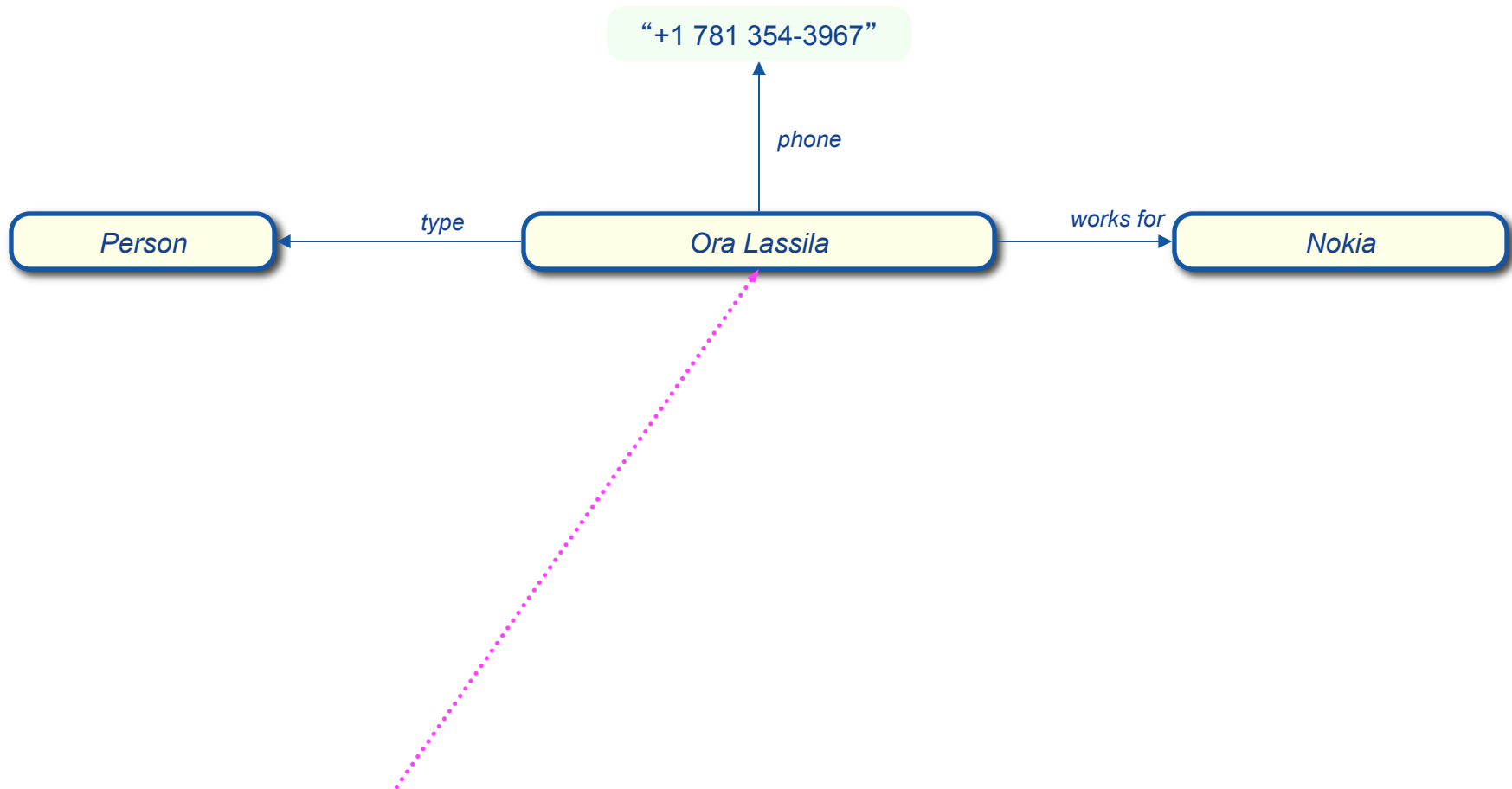
Graphs: a natural way to represent the world



- “Ora Lassila’ s phone is +1 781 354-3967”
- “Ora Lassila works for Nokia”
- “Ora Lassila **is a** Person”

Note: we are talking about graphs in the mathematical (graph-theoretical) sense, and not about visual graphs often used to illustrate statistics, etc.

Nodes and edges are named using URIs



- this is merely a human-readable name

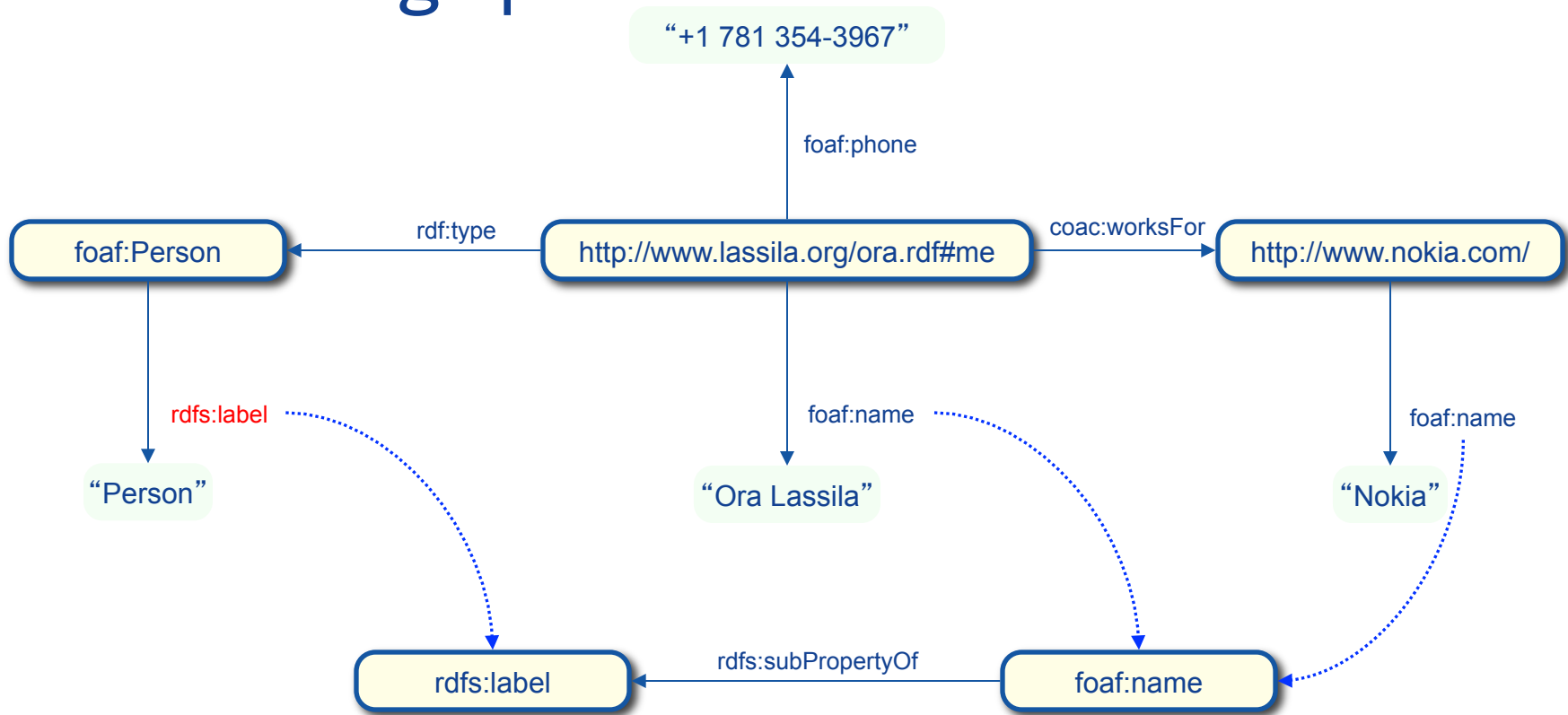
Nodes and edges are named using URIs



Note: (as an example) "rdfs:label" is the XML QName for the URI "http://www.w3.org/2000/01/rdf-schema#label"

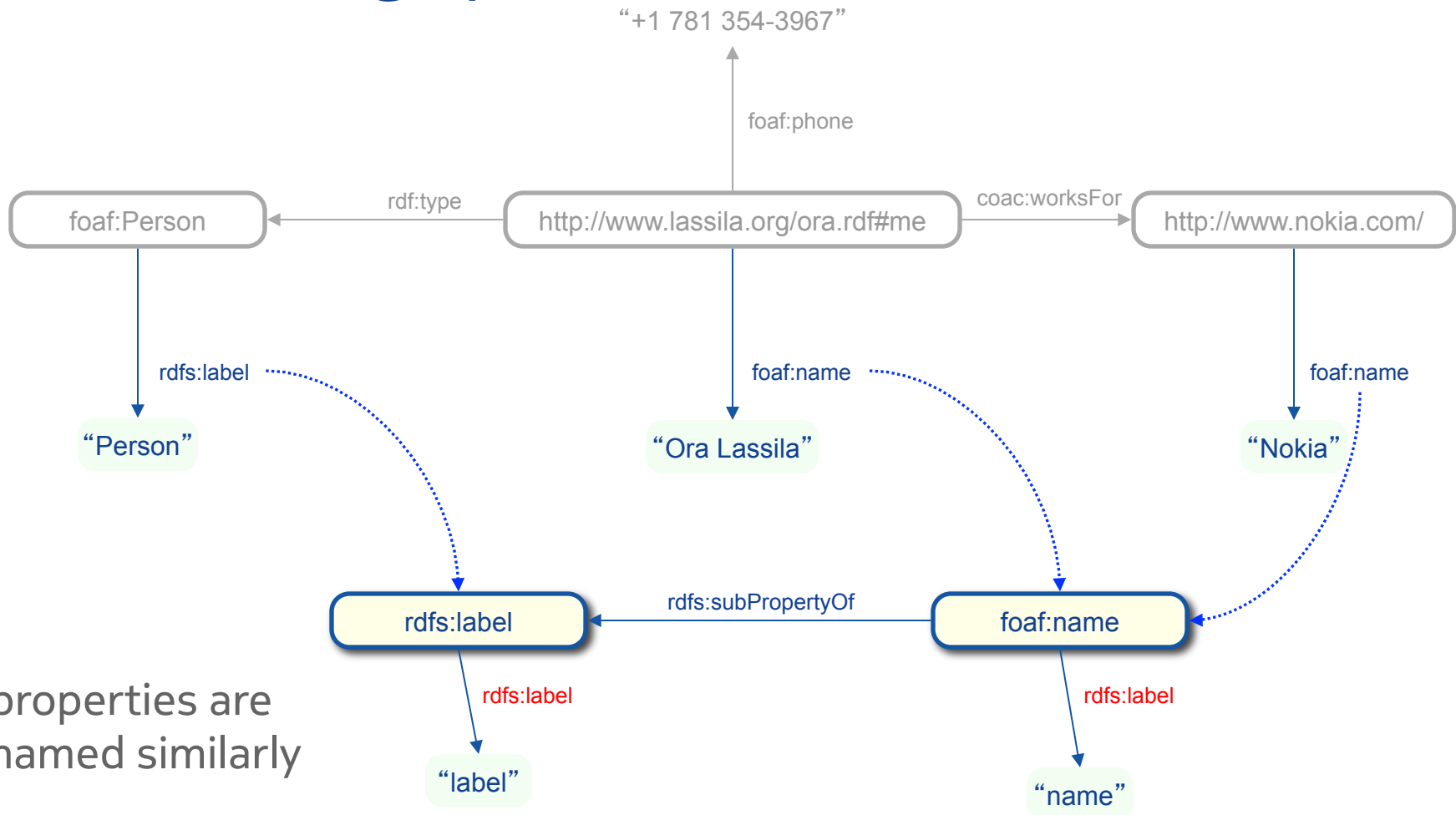
- this is merely a human-readable name

Properties (= edges) are represented by special nodes in the graph



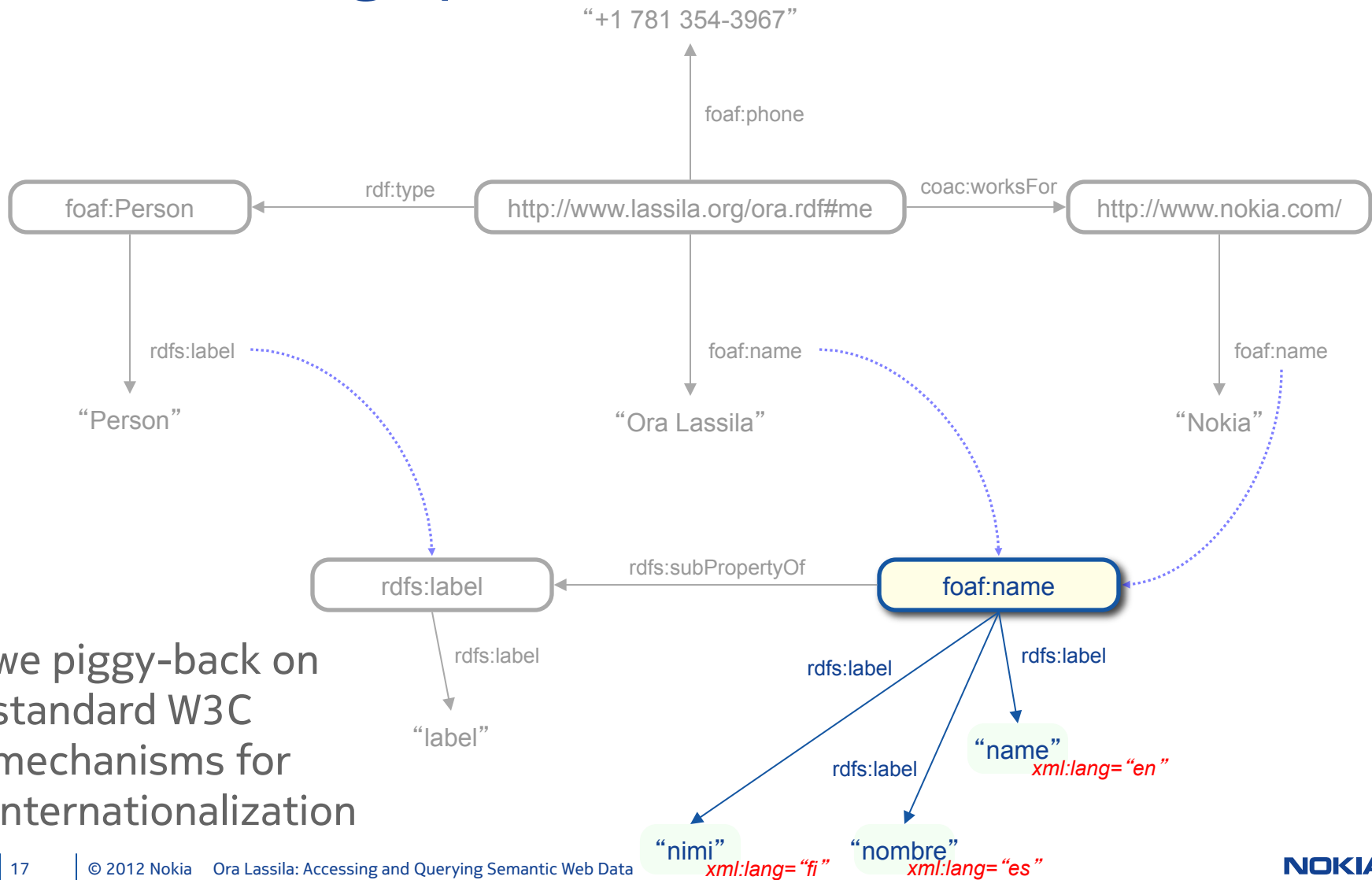
- human-readable form of a node is found by looking for **rdfs:label** properties
- a reasoner will find all sub-properties of rdfs:label as well

Properties (= edges) are represented by special nodes in the graph



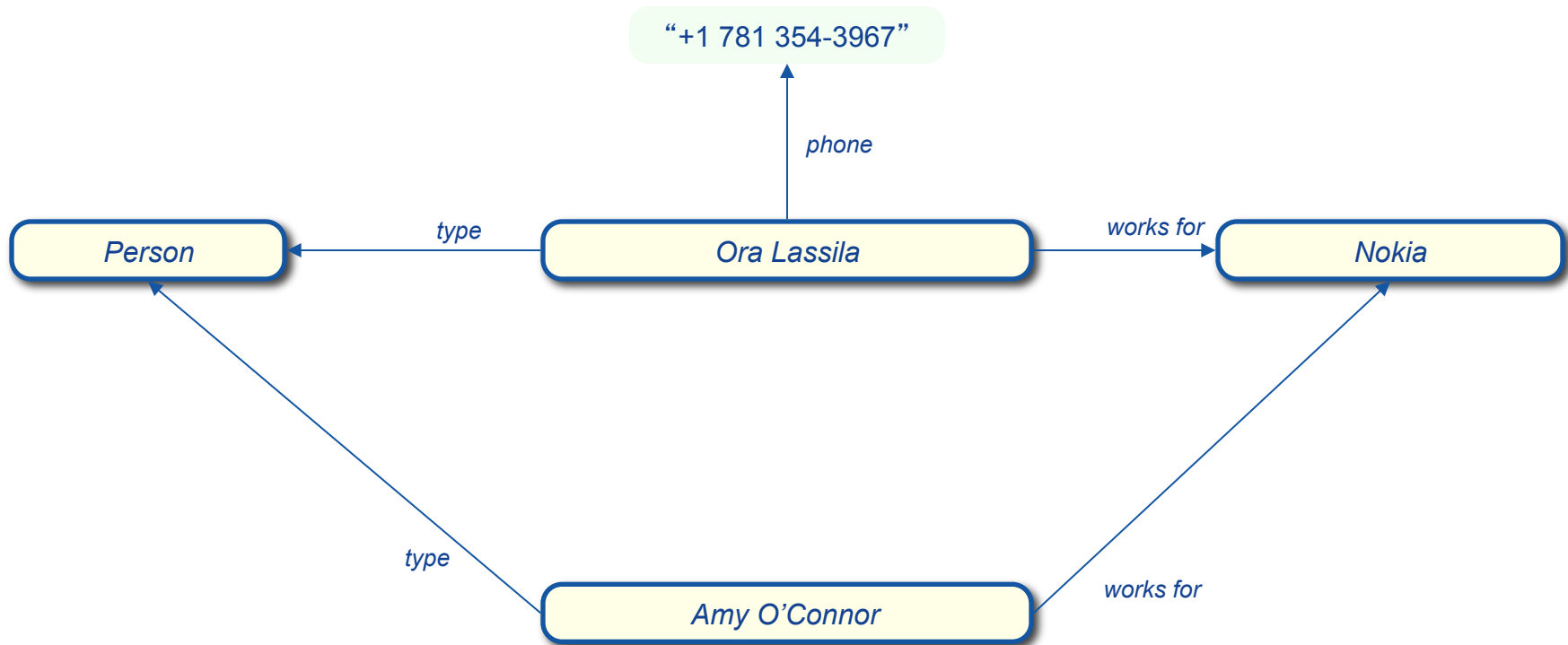
- properties are named similarly
- schema and instance-level data use same representation

Properties (= edges) are represented by special nodes in the graph



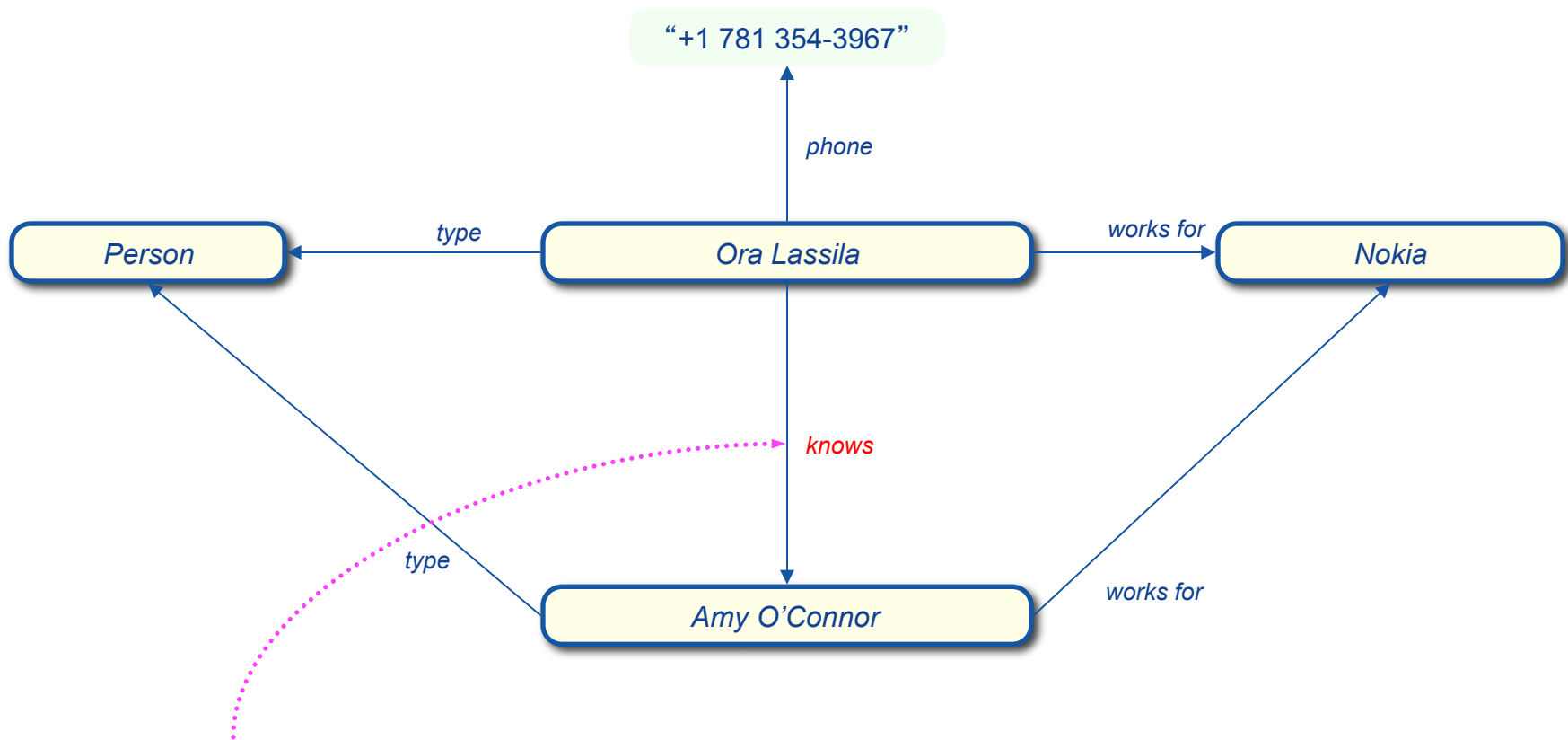
- we piggy-back on standard W3C mechanisms for internationalization

Let's add more people



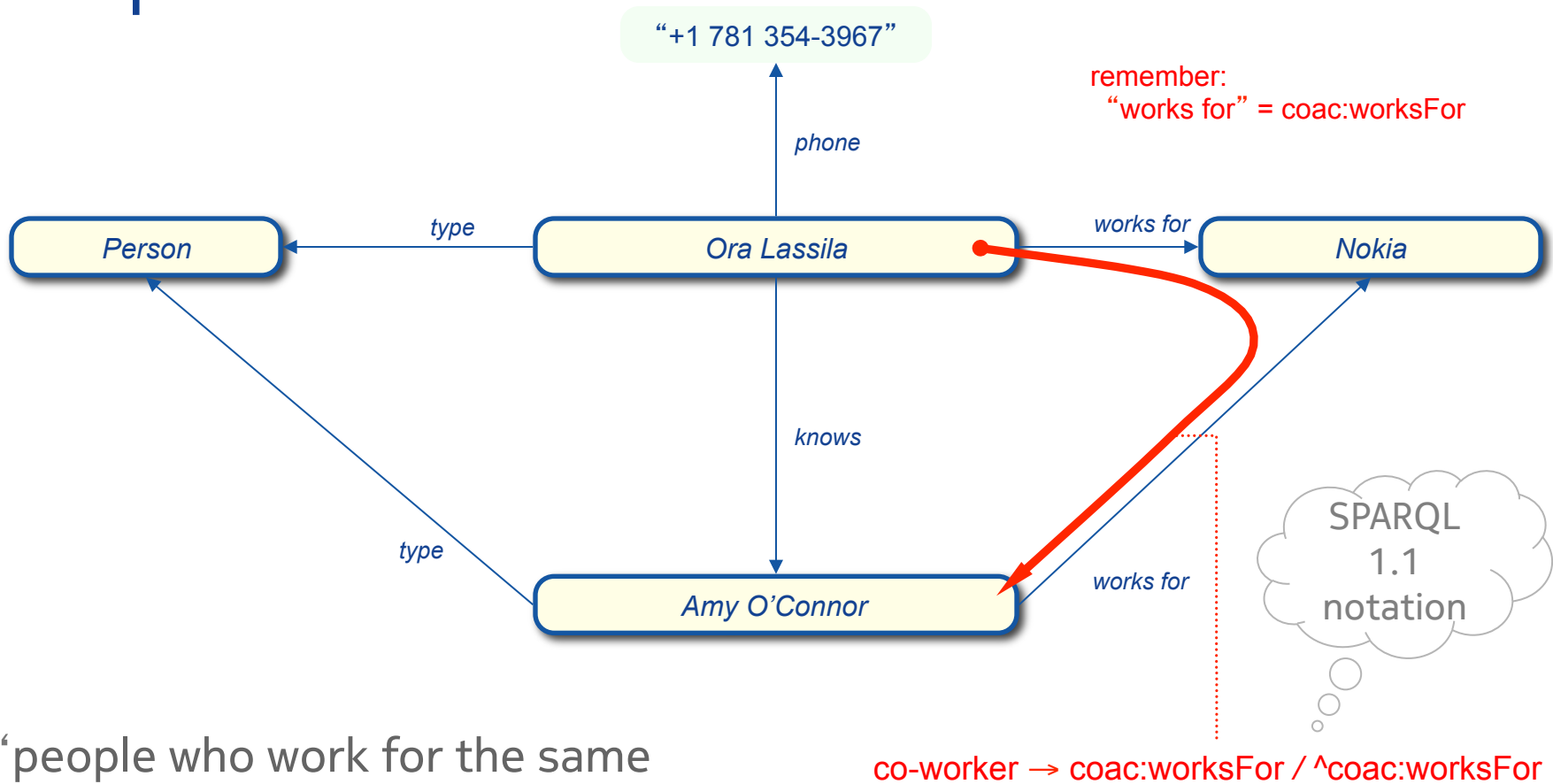
- “Amy O’Connor works for Nokia”

Social relations are easy...



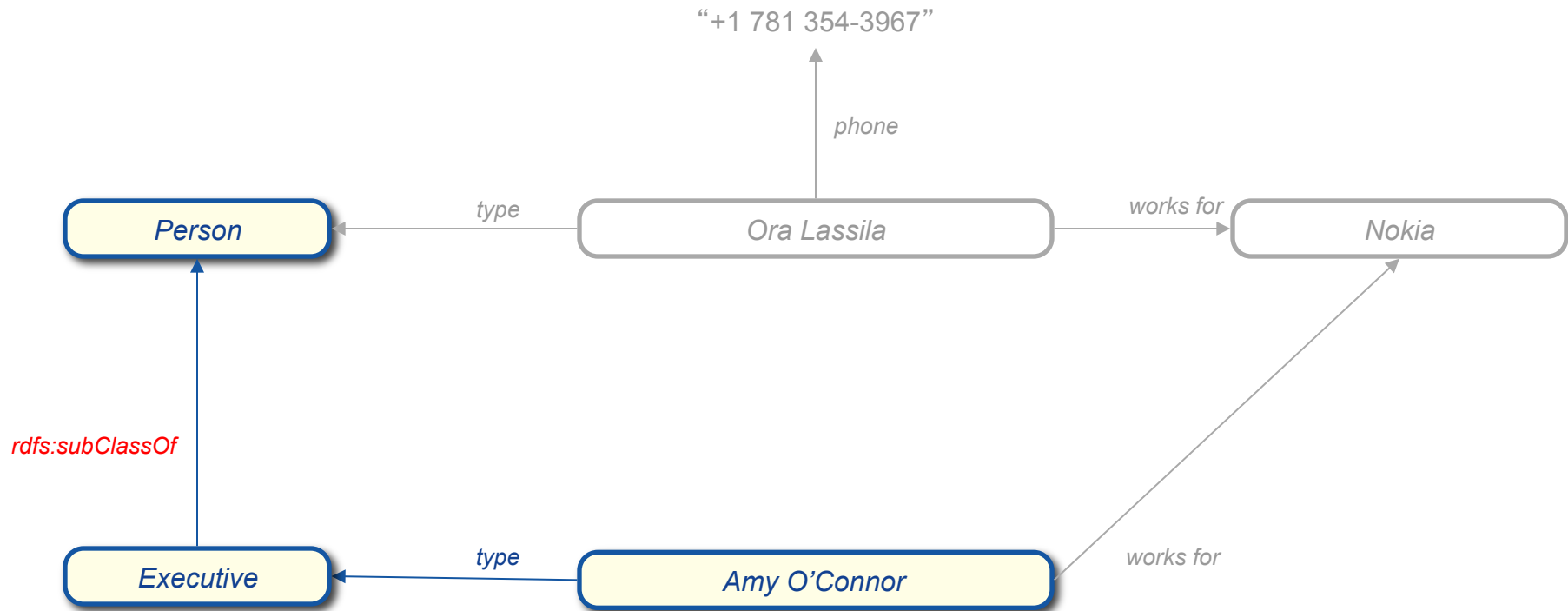
- “Ora **knows** Amy”

Path patterns can be used to represent more complex relations



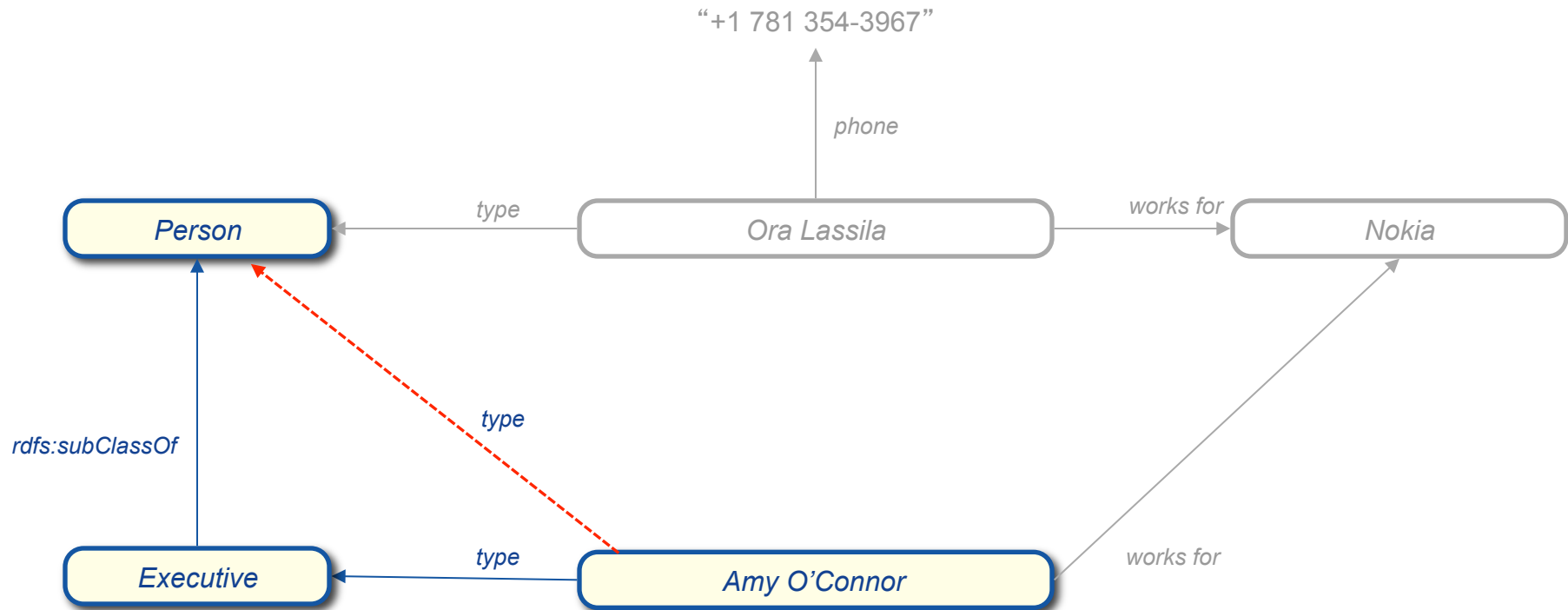
- “people who work for the same organization are co-workers”
- “Ora’s co-workers” = nodes reachable through the path “co-worker”

Polymorphism



- “Executive” is a **subclass** of “Person”

Polymorphism and reasoning



- “Executive” is a subclass of “Person”
- a reasoner **will conclude** that Amy O’Connor is also a Person

Different Approaches to Querying (Semantic Web) Data

Graph storage

- graphs can be broken into “triples”



http://www.lassila.org/ora.rdf#me	foaf:phone	“+1 781 354-3967”
http://www.lassila.org/ora.rdf#me	coac:worksFor	http://www.nokia.com/
http://www.lassila.org/ora.rdf#me	foaf:name	“Ora Lassila”
http://www.lassila.org/ora.rdf#me	rdf:type	foaf:Person
http://www.nokia.com/	foaf:name	“Nokia”
foaf:Person	rdfs:label	“Person”

Graph storage

- graphs can be broken into “triples”



<code>http://www.lassila.org/ora.rdf#me</code>	<code>foaf:phone</code>	<code>"+1 781 354-3967"</code>
<code>http://www.lassila.org/ora.rdf#me</code>	<code>coac:worksFor</code>	<code>http://www.nokia.com/</code>
<code>http://www.lassila.org/ora.rdf#me</code>	<code>foaf:name</code>	<code>"Ora Lassila"</code>
<code>http://www.lassila.org/ora.rdf#me</code>	<code>rdf:type</code>	<code>foaf:Person</code>
<code>http://www.nokia.com/</code>	<code>foaf:name</code>	<code>"Nokia"</code>
<code>foaf:Person</code>	<code>rdfs:label</code>	<code>"Person"</code>

Graph storage

- arbitrary metadata could be associated with each triple (creator, timestamp, source, etc.)
- most triple stores are actually “quad stores”



http://www.lassila.org/ora.rdf#me	foaf:phone	"+1 781 354-3967"	
http://www.lassila.org/ora.rdf#me	coac:worksFor	http://www.nokia.com/	metadata...
http://www.lassila.org/ora.rdf#me	foaf:name	"Ora Lassila"	
http://www.lassila.org/ora.rdf#me	rdf:type	foaf:Person	
http://www.nokia.com/	foaf:name	"Nokia"	
foaf:Person	rdfs:label	"Person"	

Graph storage via existing database technologies

- “schema-agnostic” approach (shown below) in a SQL database
 - flexible and simple to manage, but querying often requires complex self-joins or nested SELECTs
- “schema-aware” approach in a SQL database
 - recognizing that some properties (rdf:type, rdfs:subClassOf, etc.) are special, and creating special tables for them
 - Nokia N9’s triple store uses an approach where most properties have their own columns in a relational table
- vertically partitioned databases and key-value stores can also be used

http://www.lassila.org/ora.rdf#me	foaf:phone	+1 781 354-3967
http://www.lassila.org/ora.rdf#me	coac:worksFor	http://www.nokia.com/
http://www.lassila.org/ora.rdf#me	foaf:name	Ora Lassila
http://www.lassila.org/ora.rdf#me	rdf:type	foaf:Person
http://www.nokia.com/	foaf:name	Nokia
foaf:Person	rdfs:label	Person

Types of queries

- “S-P-O”** • matching triples, with wildcards allowed for any **S**, **P**, and **O**
- Relational** • multiple conjunctive **S-P-O** patterns
 - wildcards and variables allowed for any **S**, **P**, and **O**
- Path** • regular expressions over **Ps**
 - possibly with constraints over **S** and **O** values
- Logic, rules** • query resembles a logic program or rule execution

S	P	O
http://www.lassila.org/ora.rdf#me	foaf:phone	+1 781 354-3967
http://www.lassila.org/ora.rdf#me	coac:worksFor	http://www.nokia.com/
http://www.lassila.org/ora.rdf#me	foaf:name	Ora Lassila
http://www.lassila.org/ora.rdf#me	rdf:type	foaf:Person
http://www.nokia.com/	foaf:name	Nokia
foaf:Person	rdfs:label	Person

Types of queries – examples of query languages

“S-P-O”

- all triple store **APIs**

Relational

- **SPARQL** (and many of its predecessors that led to its specification: RDQL, SeRQL, etc.)

Path

- path languages (Lorel, Cypher, Versa, **WilburQL**, etc.)
- **SPARQL 1.1**

Logic, rules

- OWL-QL, Triple, N3, etc.

S

P

O

http://www.lassila.org/ora.rdf#me	foaf:phone	+1 781 354-3967
http://www.lassila.org/ora.rdf#me	coac:worksFor	http://www.nokia.com/
http://www.lassila.org/ora.rdf#me	foaf:name	Ora Lassila
http://www.lassila.org/ora.rdf#me	rdf:type	foaf:Person
http://www.nokia.com/	foaf:name	Nokia
foaf:Person	rdfs:label	Person

Data modification

- Most software developers expect support for atomic CRUD operations (“create, read, update, delete”) for their data
- RDF presents an additional problem when updating or deleting data – what is the “scope” of the operation (e.g., from the standpoint of atomicity)?
- Acting on individual triples may be too fine grained if larger parts of an RDF graph are thought of as objects
- Many attempts to define parts of an RDF graph:
 - Concise Bounded Descriptions
 - RDF Molecules
 - Named Graphs
 - etc.

Queries: S-P-O query APIs

- Sesame (Java): interface “**Graph**” has a method “**match**”
- Jena (Java): interface “**Model**” has a method “**query**” which takes a “**Selector**” instance
 - S-P-O queries are provided via the “**SimpleSelector**” class
- Redland (C): function “**librdf_model_find_statements**” takes a partially filled “**librdf_statement**” (pass NULL for wildcards)
- Piglet (C++ and Python): class “**DB**” has a method “**query**” with parameters *s*, *p*, *o* (pass 0 to indicate a wildcard)
- etc.

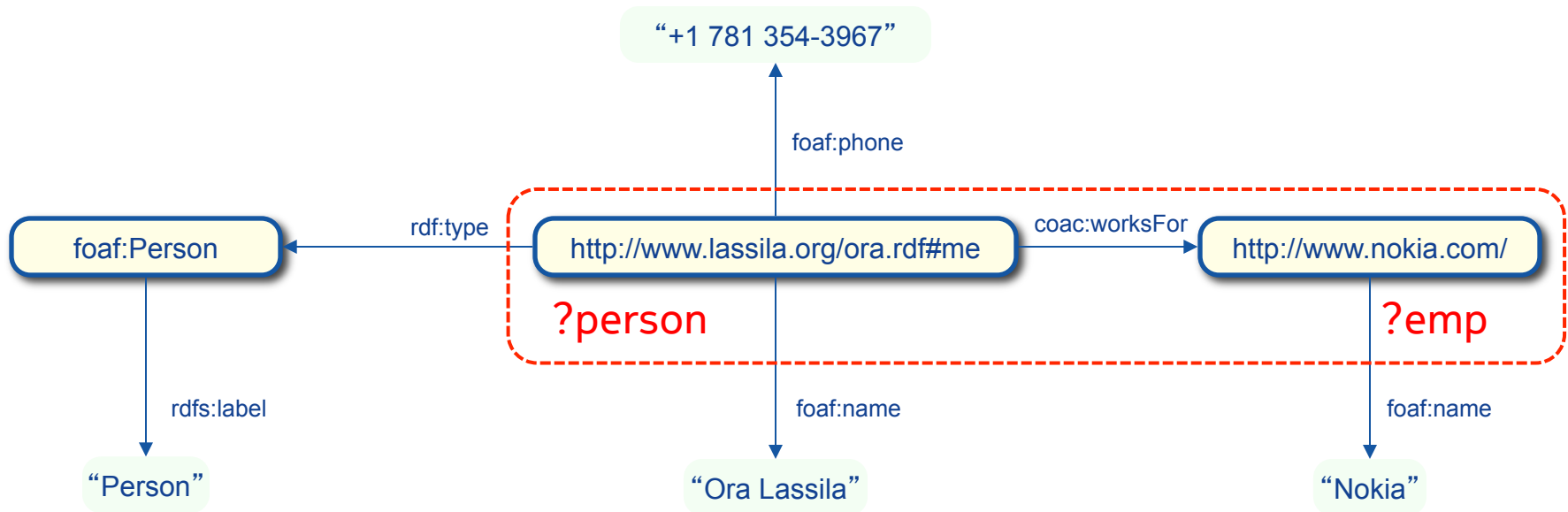
Relational queries: SPARQL

- SPARQL is the W3C standard on querying RDF
- SPARQL version 1.0 only offers “relational” queries:
<http://www.w3.org/TR/rdf-sparql-query/>
 - no transitive closure
 - no possibility to modify data
 - “insensitive” to inference (but you could query against deductive closure)
- SPARQL version 1.1 (draft: <http://www.w3.org/TR/sparql11-query/>) adds a number of things, including
 - data modification (“SPARQL update”)
 - federated queries
 - path queries (including transitive closure)

Understanding SPARQL

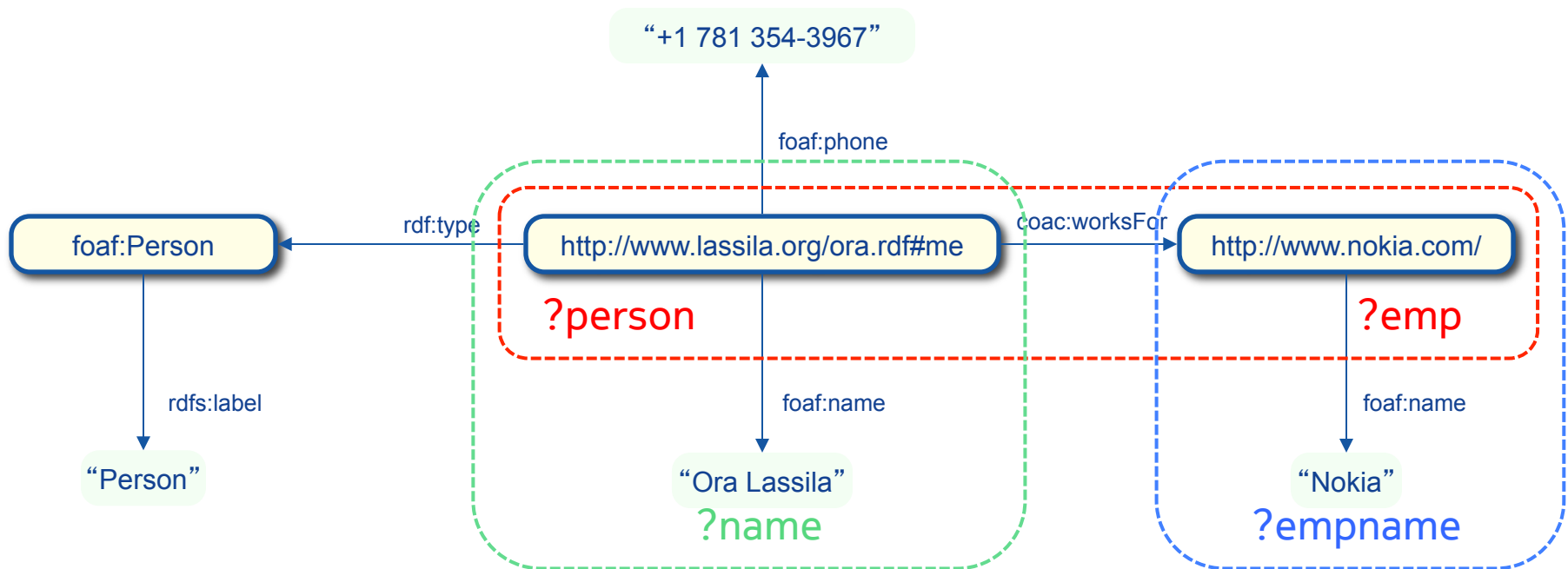
- Structure of a SPARQL query:
 1. prefix declarations → *how we refer to things*
 2. dataset definitions → *sources of data*
 3. result clause → *what does the query return*
 4. query pattern → *conditions that the results have to satisfy*
 5. modifiers → *how to modify the results*
- There are variables in #4 that the query execution binds, these are then used by #5 and #3
- Conditions in #4 are
 - triple patterns (S-P-O)
 - filter conditions
- My favorite SPARQL tutorial is this:
 - <http://www.cambridgesemantics.com/2008/09/sparql-by-example/>

Formulating SPARQL queries (1)



```
SELECT ?person, ?emp
WHERE {
  ?person coac:worksFor ?emp .
}
```

Formulating SPARQL queries (2)



```
SELECT ?name, ?empname
WHERE {
  ?person coac:worksFor ?emp .
  ?person foaf:name ?name .
  ?emp foaf:name ?empname .
}
```

Some additional observations about SPARQL

- SPARQL is not just a query language
 - SPARQL protocol defines how clients talk to servers (“SPARQL endpoints”)
 - Graph Store protocol defines how to manage graph content through REST-style interface
 - different formats are available for returned query results
- LOD paradigm implies that data be published through open SPARQL endpoints
 - however, the cost of running such endpoints (due to the computational cost of SPARQL queries) may be higher than expected
 - (there is a broader issue of “business models” for Semantic Web...)
- Although now outdated, [Haase et al 2004] is an interesting comparison of various RDF query languages
 - [Lassila 2007] adds SPARQL and WilburQL using the same tests

Path queries

- WilburQL [Lassila 2007]
 - query language of the Wilbur Semantic Web toolkit; also implemented in the Piglet triple store
 - based on the path language of the BEEF frame-based KR system [Hynynen & Lassila 1989, Lassila 1992] which, in turn, was inspired by the path languages of CRL & SRL [Fox 1985]
- various other graph query languages (sometimes as part of graph databases)
 - also: XPath (for querying XML via paths through DOM trees)
- SPARQL version 1.1 adds “property paths”

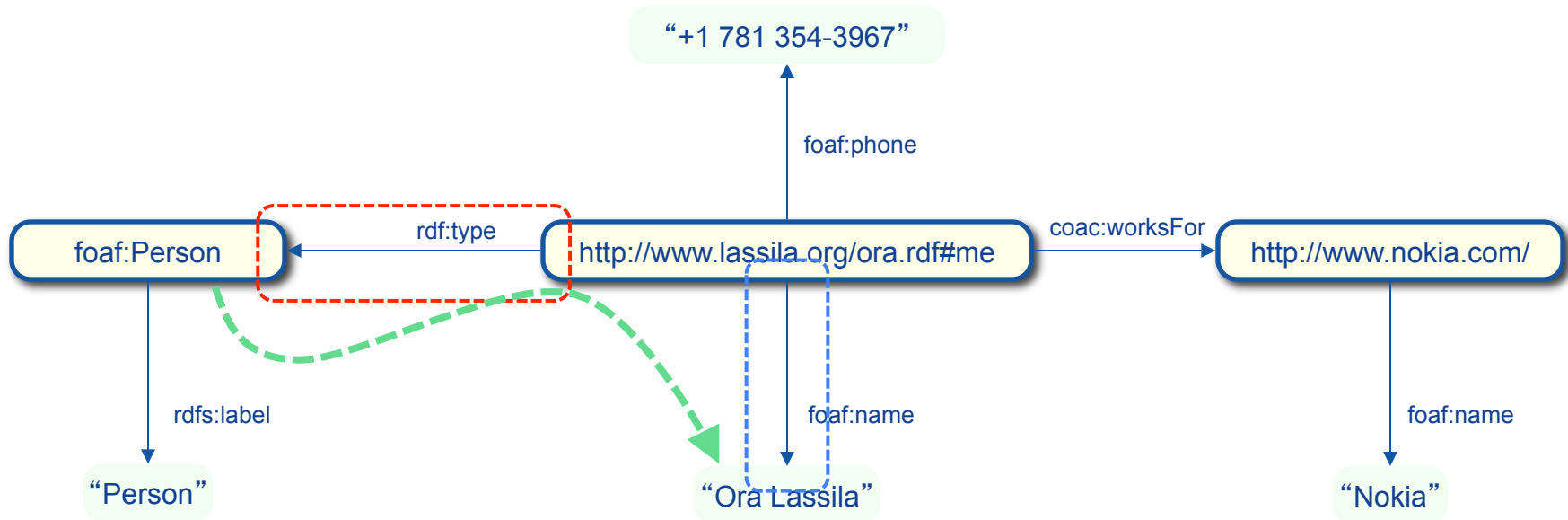
Path queries and SPARQL

- Lack of ability to express transitive closure makes relational query languages ill-suited for graph manipulation
 - there is well known literature about this already from the 1970s
- Ability to “map” SPARQL queries to SQL queries (of an underlying relational storage solution) could be seen as motivation to stay with the relational model
 - not as straightforward as one would think
 - solutions now exist, however, to do this (e.g., D2RQ)
- Early on in the SPARQL development process, I suggested that path queries should be considered as well
 - this was rejected at the time, but later the need for property paths has become clear and version 1.1 adds these
 - so far, not much implementation experience (as far as I understand), initial implementations reportedly slow

WilburQL explained

- Query execution constitutes collecting all nodes in the graph reachable via a given path, starting from a “root” node
 - alternatively, one can merely test for “reachability”
- Paths are regular expressions over RDF properties (graph edge labels)
- Paths are constructed using the normal set of regex operators:
 - **seq**: sequence – concatenation of sub-paths
 - **or**: disjunction – alternative sub-paths
 - **inv**: inverse – reverse traversal of a sub-path
 - **rep**: repetition – transitive closure of a sub-path
- Implemented by transforming regular expressions into *deterministic finite-state automata* (FSA) that are used to “walk” the graph
 - maximal “non-transitive” sub-paths are transformed into (potentially complex) SQL queries against an S-P-O table; these sub-path queries are driven by an FSA

Formulating path queries (1)



Names of all known persons (using `foaf:Person` as the “root”)

WilburQL: `seq(inv(rdf:type), foaf:name)`

SPARQL 1.1: `^rdf:type / foaf:name`

Formulating path queries (2)



Names of employers of all known persons (using foaf:Person as the “root”)

WilburQL: seq(inv(rdf:type), coac:worksFor, foaf:name)

SPARQL 1.1: ^rdf:type / coac:worksFor / foaf:name

Some uses of path queries with RDF

- Transitive closure is needed when exact length of path is not known (e.g., due to repetitive or recursive graph patterns)
 - concept hierarchies, organizational hierarchies, etc.
- Path discovery (i.e., are two nodes linked in some way)
 - interesting implementation is described in [Atre 2011]
- Interactive access to RDF data by capturing browsing paths as path queries [Lassila 2006b]
 - a different take on “query by example”
- Reasoning via rewritten path queries [Lassila 2007]
 - expresses entailments as “alternate paths” through the graph
→ deductive closure is a “virtual view” over the original graph

Programming, (Data) Semantics and the Semantic Web

What is a “metamodel”...? (informal discussion)

- In the context of object systems, a metamodel allows us to describe what “all objects look like”
 - independent of domain modeling considerations (i.e., knowledge of the metamodel allows you to examine the structure of data without interpreting a domain schema)
 - sometimes referred to as the “object model”
- In the context of RDF, we could say that
 - graphs consist of nodes (“resources” & “literals”) and edges (“properties”)
 - resources and edges are named (labeled) using URIs
 - multiple edges with the same label are permitted for any node
 - (note that RDF permits an interpretation of data either as objects or as graphs)
- If you compare the “RDF metamodel” with, say, the “JSON metamodel” you will notice that they are different – more on this later...

What establishes (data) semantics?

1. Some (procedural) software that “hard-wires” how to process certain kind of data
 2. Relationship of data to (declarative) definitions of data types (e.g., object-oriented classes); relationships between definitions (e.g., class-subclass relations)
 3. Relationship of data to some other data
 - e.g., reasoning, based on some ontology
- Sometimes, we do not need to understand the semantics of data *per se*, we only need knowledge of the metamodel
 - a Web browser knows how to render HTML without having to care about higher-level semantics
 - similarly, RDF can be “browsed” without interpreting the semantics of associated schemata or ontologies [e.g., Lassila 2006a]

Different object systems have different semantics

- XML (trees) vs. RDF (graphs) – poor match
 - consequently, XML tools are largely useless when it comes to RDF
 - exception: using XSLT to transform legacy XML into RDF (in this case, the XSLT script “embodies” the necessary semantics)
 - generally, the choice of XML as the standard serialization syntax for RDF was ill-advised (it was motivated by politics, not by rational design)
- JSON (e.g., JSON-LD) vs. RDF – poor match
 - JSON-LD cannot be parsed (correctly) into RDF without knowledge of the domain schema (thus violates a core principle of RDF)
 - many subtle differences in how JSON behaves vs. how RDF behaves – note that in the case of JSON-LD, we have “layered” additional semantics on top of JSON
- **Warning:** Most people simply do not understand the “semantic mismatch” between object systems (one you use, the other one used to implement)
 - (this is the case even after you attempt to explain it to them...)

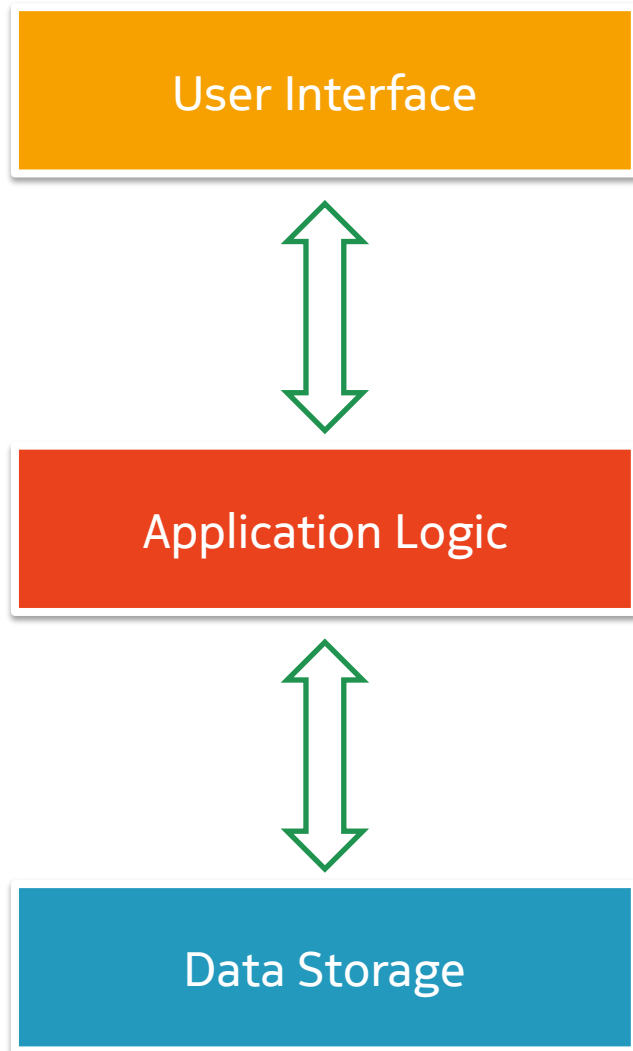
Programming languages vs. KR systems

- One can **implement** RDF data using, say, Java objects, but the resulting objects (RDF resources, etc.) do not behave like Java objects
 - e.g., Java type system is not the same as the RDF type system
- Multiple ways to (attempt to) reconcile semantics:
 1. pick KR system, design your own programming language with matching semantics – doable, and has been attempted, but **highly impractical**
 2. pick programming language, design your own KR system with matching semantics – was doable earlier, but now we have **KR standards** (OWL, RDF)
 3. “tweak” programming language semantics to match KR system – almost doable in some really expressive languages like CLOS, but in practice **not 100% viable**
 4. accept that #1, #2 and #3 do not work, implement data access through a low-level API or something like an “ORM” (Object-Relational Mapper) – and **accept the difference in semantics**

Object systems vs. storage systems

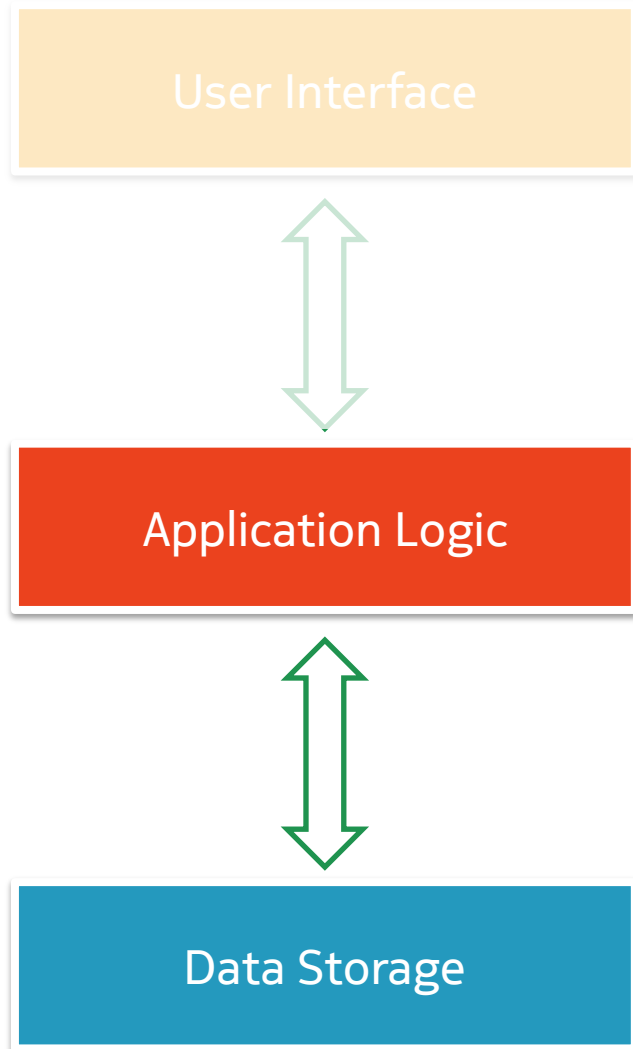
- Same problem, again
 - e.g., Java objects vs. SQL database
- Typical solutions are also similar
 - e.g., object-relational mappers
 - RDF presents an additional problem, namely the “object/graph dichotomy” (i.e., do you treat the data as a graph or as objects)
- Bottom line: work through some API, keep the differences in semantics in mind

Typical application architecture



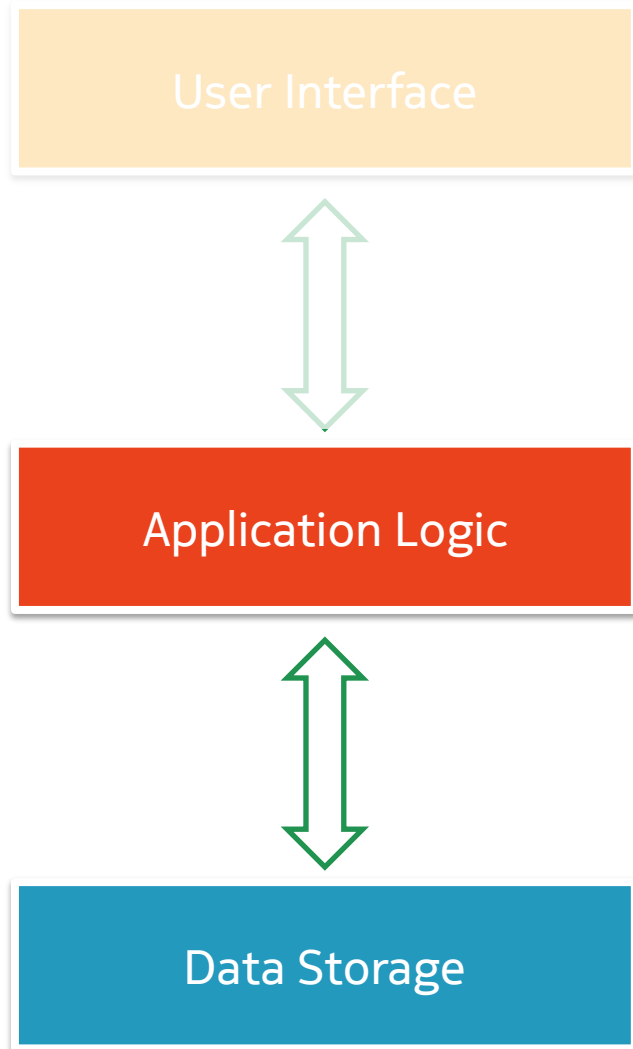
- Conceptually, the models in the three different parts are the same, but actual implementations differ
- Things get difficult when object system semantics differ
 - e.g., **Javascript** vs. **C++** vs. **RDBMS**
- Mappings in-between may not be fully formal, or not formalized at all
 - they “live” inside developers’ heads
- Much effort is spent on managing the mappings; this is distracting and error-prone

Typical application architecture



- In this lecture, I want to focus on the issues between Application Logic and Data Storage
- We assume that we want to store Semantic Web data (RDF graphs)
- Central questions:
 - how does the “RDF object system” map onto the object system your programming language offers?
 - what is the role of the API of the storage layer?
 - what query language to use, what do queries look like?

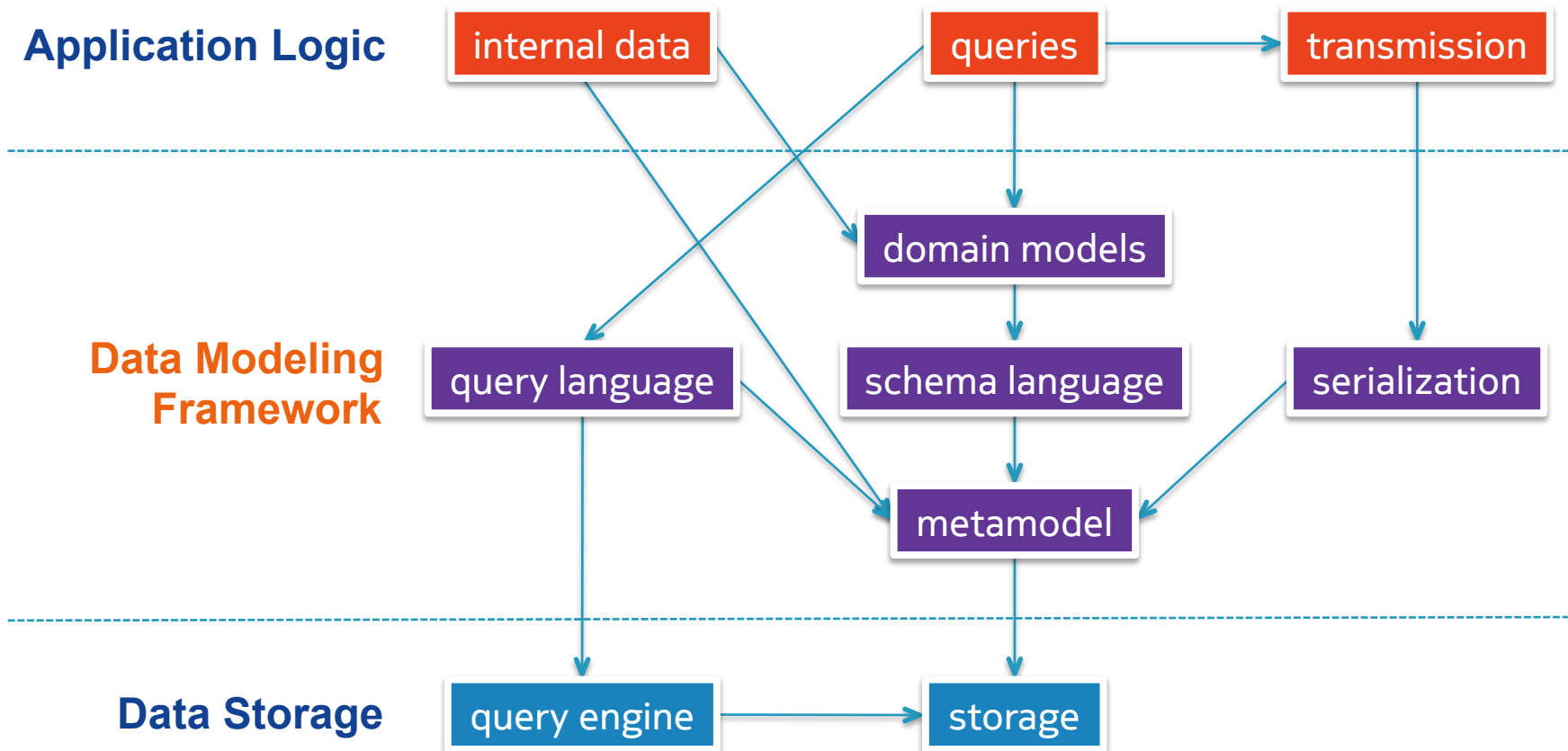
Typical application architecture



- Actually, the situation is much more complex than I may have indicated... Sorry.

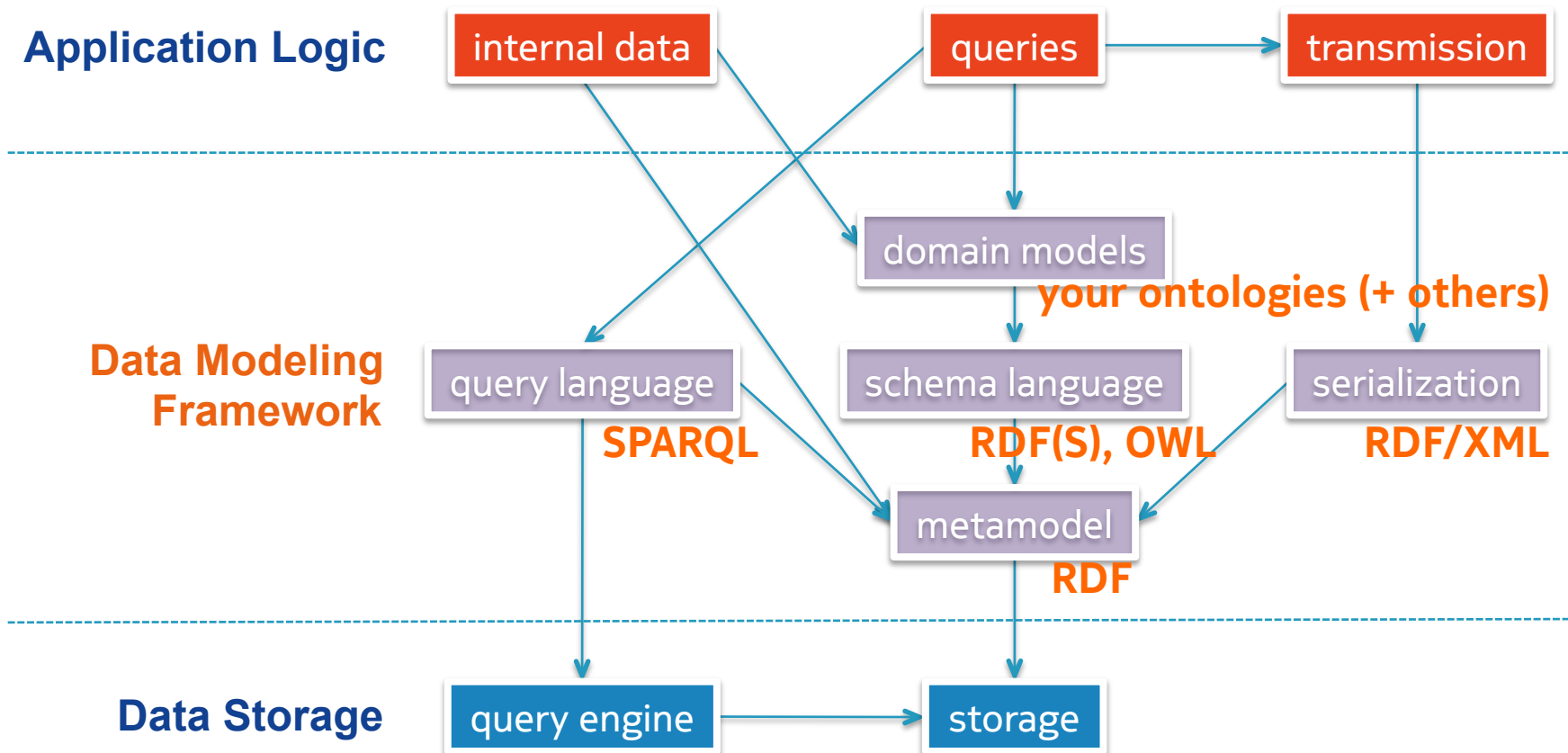
Application ↔ data

- There are dependencies between technologies in an application architecture



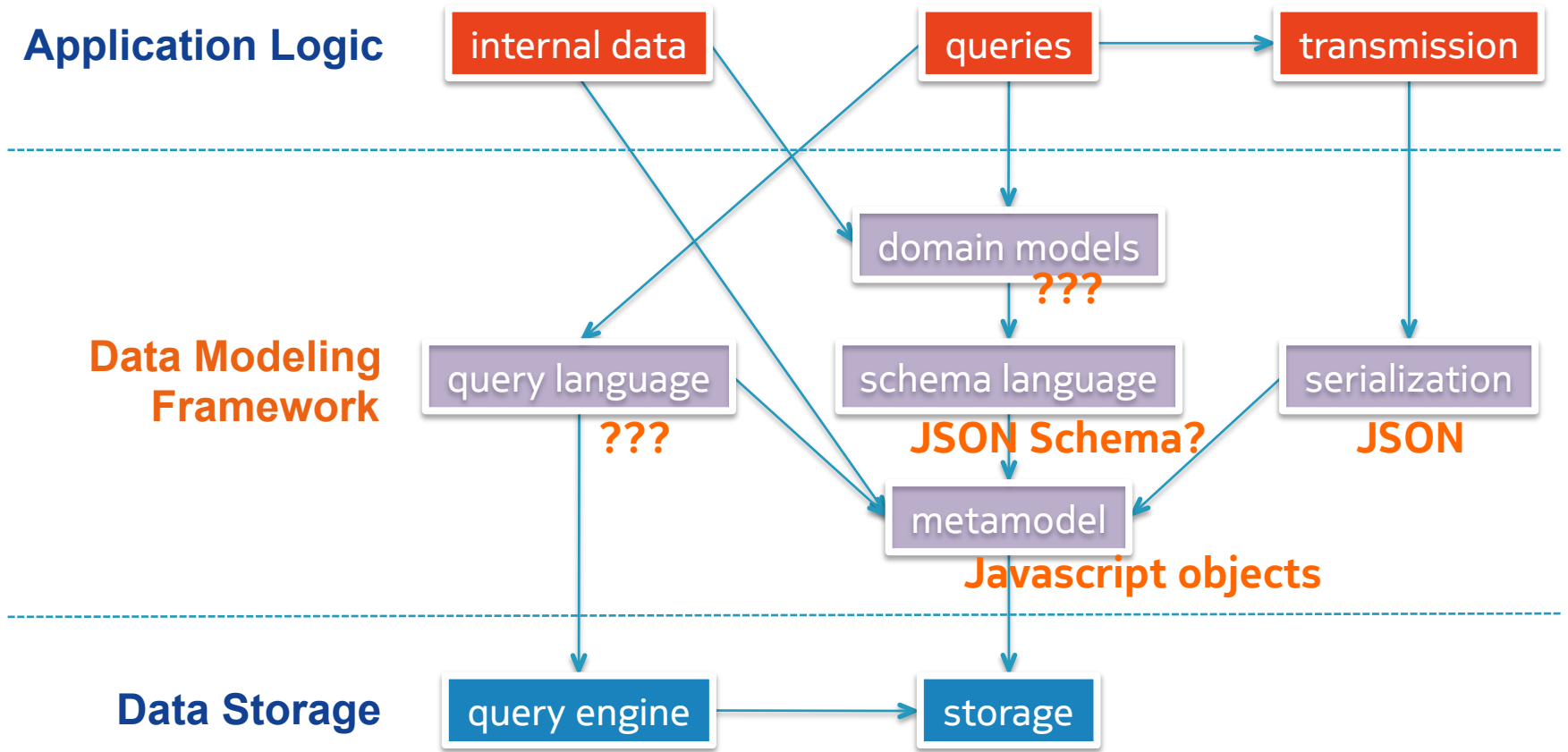
Application ↔ data

- You must make a consistent technology selection



Application ↔ data

- You must make a consistent technology selection – this may not always be easy



Semantic Web programming challenges

- Lots of data, complex/rich data models
- Tools not fully mature yet (compared to “mainstream ICT”)
- Application architectures and programming approaches may be different
 - how we program vs. “unanticipated situations”
 - expressiveness of data vs. expressiveness of UI
- Confusion arising from different object system semantics
 - earlier: RDF vs. XML
 - today: RDF vs. JSON
- W3C is considering on starting work on “Linked Data Platform”
 - guidance on how Semantic Web and Linked Data technologies should be applied when developing applications
 - standards already exist, but people do not know how to apply them
 - (some pieces still missing; e.g., access control)

Conclusions

- RDF data is a graph. Graphs are easy to understand.
- RDF can be stored in many different ways. Querying may not be so easy. SPARQL is the standard.
- Real problems start when you map semantics of one system to another. And we have to do it all the time.

(All this is meant as a cautionary tale and as “food for thought”. Be careful out there.)

Some literature

- [Atre 2011] M. Atre. ***Bit-by-Bit: Indexing and Querying RDF data using Compressed Bit-Vectors***. Ph.D dissertation, Rensselaer Polytechnic Institute, August 2011
- [Berners-Lee, Hendler & Lassila 2001] T. Berners-Lee, J. Hendler, and O. Lassila. **The Semantic Web**. *Scientific American*, 284(5):34–43, May 2001
- [Fox 1985] M. S. Fox. **Knowledge Representation for Decision Support**. In Methlie and Sprague, eds., *Knowledge Representation for Decision Support Systems*. Elsevier, 1985
- [Haase et al 2004] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. **A Comparison of RDF Query Languages**. In McIlraith et al, eds, *The Semantic Web – ISWC 2004, 3rd International Semantic Web Conference, LNCS 3298*, 502–517. Springer-Verlag, 2004
- [Hynynen & Lassila 1989] J. Hynynen and O. Lassila. **On the Use of Object-Oriented Paradigm in a Distributed Problem Solver**. In Jaakkola and Linnainmaa, eds., *Proc. 2nd Scandinavian Conference on Artificial Intelligence (SCAI'89)*, 657–669, Tampere, Finland, 1989
- [Lassila 1992] O. Lassila. **The Design and Implementation of a Frame System**. Master's thesis, Faculty of Technical Physics, Helsinki University of Technology, 1992
- [Lassila 2006a] O. Lassila. **Browsing the Semantic Web**. In *17th International Conference on Database and Expert Systems Applications (DEXA'06)*, 365–369, Krakow, Poland, 2006. IEEE Computer Society
- [Lassila 2006b] O. Lassila. **Generating Rewrite Rules by Browsing RDF Data**. In *Proc. 2nd International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2006)*. IEEE Computer Society, 2006
- [Lassila 2007] O. Lassila. **Programming Semantic Web Applications: A Synthesis of Knowledge Representation and Semi-Structured Data**. PhD thesis, Helsinki University of Technology, November 2007

Questions & Answers

How to contact me later

- Email: ora.lassila@nokia.com
- Twitter: [@gotsemantics](https://twitter.com/gotsemantics)
- Web: <http://www.lassila.org/>