Neuro-Symbolic Knowledge Representation and Reasoning

Uli Sattler
Professor in Computer Science
University of Manchester

ESSAI 2024 Athens

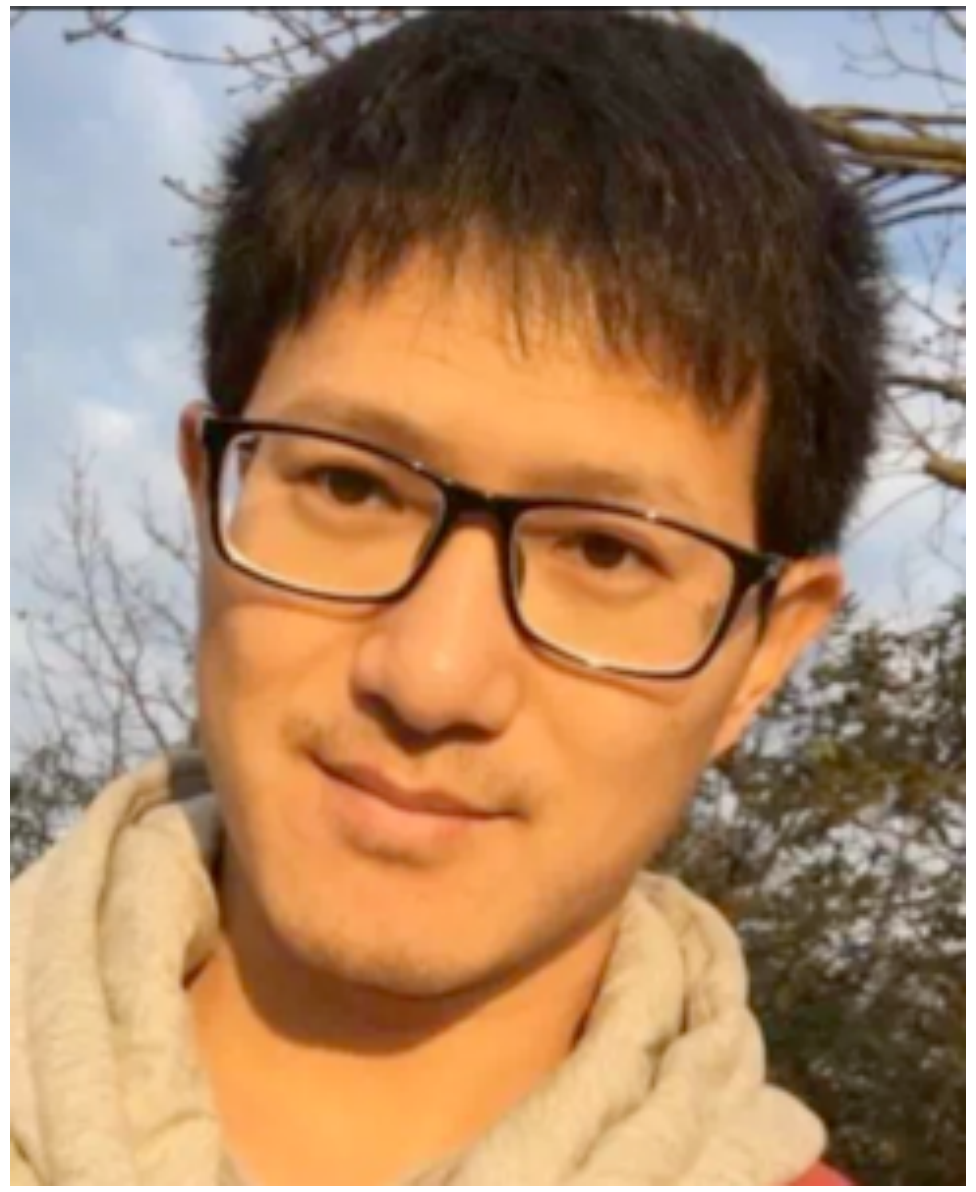
MANCHESTER 1824
The University of Manchester

Preamble

Uli Sattler
Professor in Computer Science
University of Manchester

ESSAI 2024 Athens

This course is



- introductory
- aimed at general computer scientist

- taught by
  - Jiaoyan Chen - days 3-5
  - Uli Sattler - days 1-2

- explores combination/integration/collaboration of
  - neural &
  - symbolic
    - approaches to knowledge representation, reasoning, ML, …

This course

- is rather intensive
  - 5 * 90mins
  - no coursework, example classes, exercises
- requires a lot of attention
  - please ask if something is unclear…
- is selective
  - we cannot cover all approaches/applications/views

Overview of this course

Day	Topic	Concepts	Technologies
1	Knowledge Graphs	parsing/serialisation, queries, schemas, validation & reasoning	RDF(S), SPARQL, SHACL,
2	Ontologies	Facts & background knowledge, entailments, reasoning & materialisation	OWL, OWL API, Owlready, Protégé
3	Knowledge Graph Embeddings	Classis Es, literal-aware Es, variants, evaluation	TransE, TransR
4	Ontology Embeddings	Geometric embeddings, literal-aware OEs, soundness & completeness	ELEm, BoxEL, Box^2EL, OWL2Vec*, HiT
5	Applications & Outlook	Preprocessing, materialisation, evaluation	DeepOnto, mOWL

Motivation

Uli Sattler
Professor in Computer Science
University of Manchester

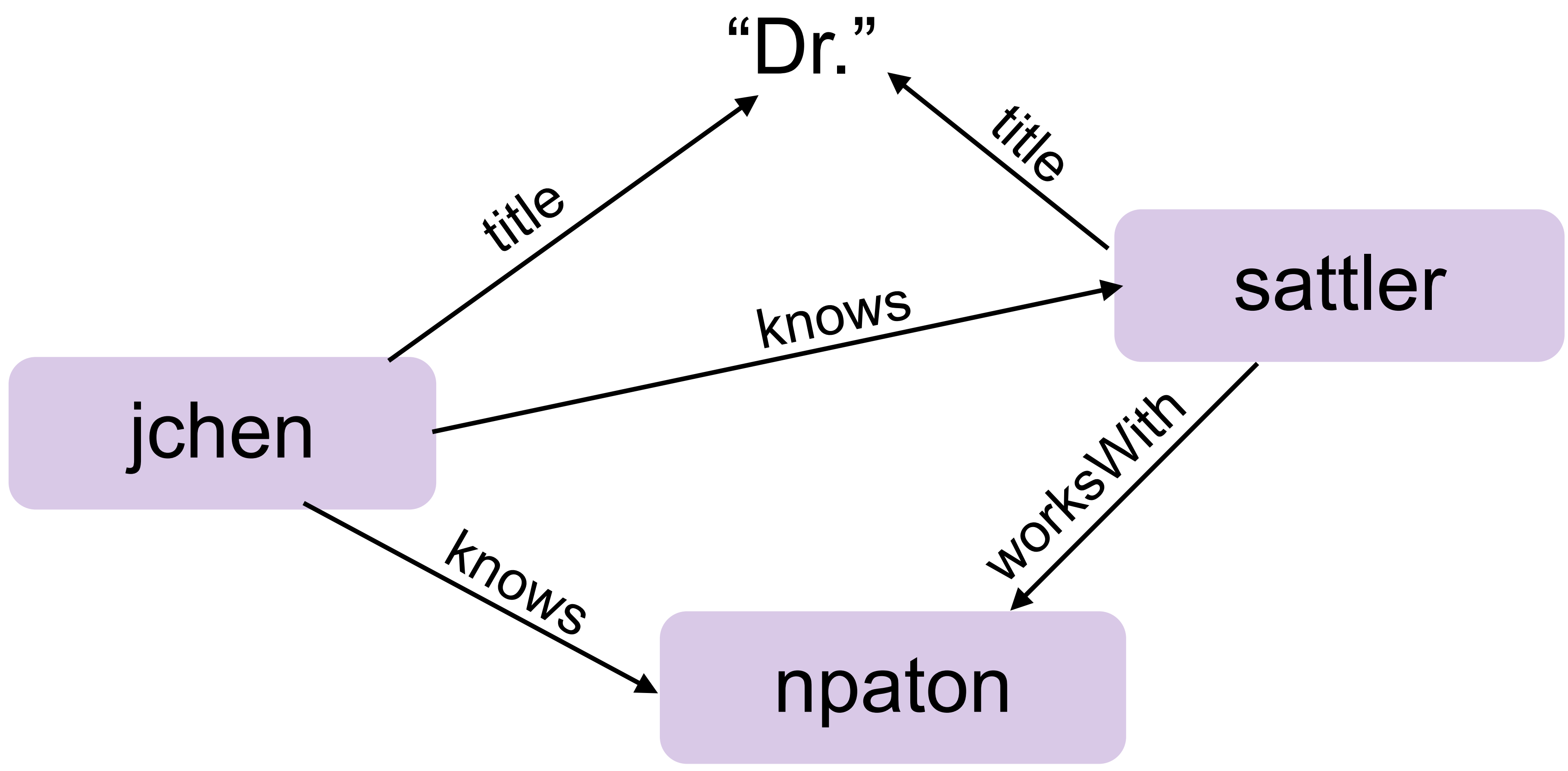
ESSAI 2024 Athens

How do we store/access data?

	name	company	address
2	Aleshia Tomkiewicz	Alan D Rosenburg Cpa Pc	14 Taylor St, St.
3	Evan Zigomalas	Cap Gemini America	5 Binney St, Abb
4	France Andrade	Elliott, John W Esq	8 Moor Place, Ea
5	Ulysses Mcwalters	Mcmahan, Ben L	505 Exeter Rd, H
6	Tyisha Veness	Champagne Room	5396 Forth Stre
7	Eric Rampy	Thompson, Michael C Esq	9472 Lind St, De
8	Marg Grasmick	Wrangle Hill Auto Auct & Slvg	7457 Cowl St #7
9	Laquita Hisaw	In Communications Inc	20 Gloucester P
10	Lura Manzella	Bizerba Usa Inc	929 Augustine S
11	Yuette Klapec	Max Video	45 Bradfield St #
12	Fernanda Writer	K & R Associates Inc	620 Northampto

- in **relational** database
  - ✓ manipulation via SQL
  - ✓ very fast access
  - ✓ highly optimised DBMSs
  - ✓ proven technology, well understood properties
  - ✓ with ACID guarantees
  - requires *schema,* certain *normal forms, joins*
  - tricky for
    - sharing/spontaneous usage
    - many many-to-many relations
    - navigational/path queries

How do we store/access data?

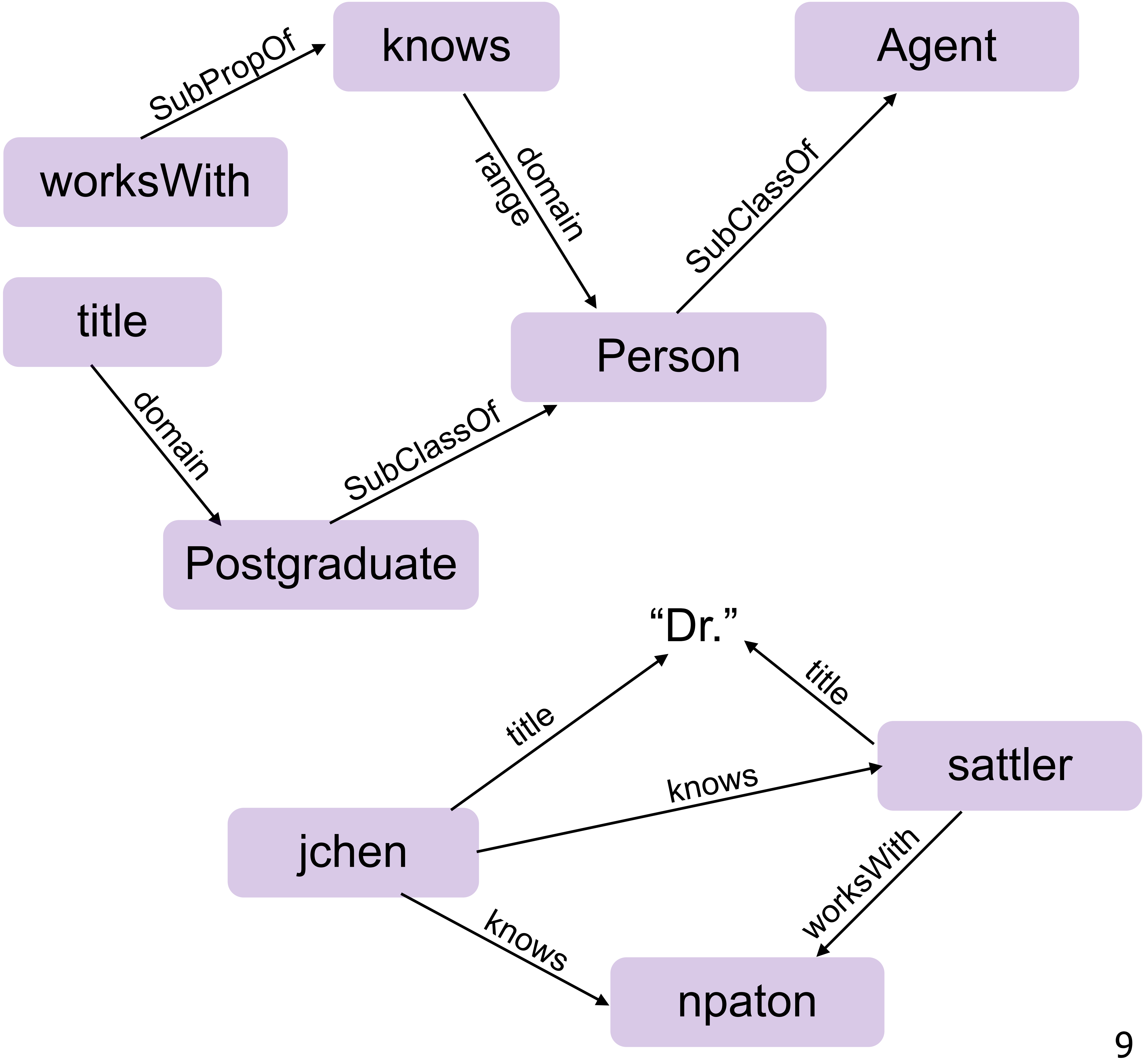


- as **graphs**/in **graph** database
  - for **objects** and their **relations**
  - manipulation:
    - programmatically
    - via query languages
  - relatively new, but powerful DBMSs available
    - often built 'on top' of RDMSs
  - doesn't require *schema or normal forms* or *joins*
  - suitable for
    - data with many many-to-many relations
    - navigational/path queries
    - sharing/integrating data

Data Graph or Knowledge Graph?

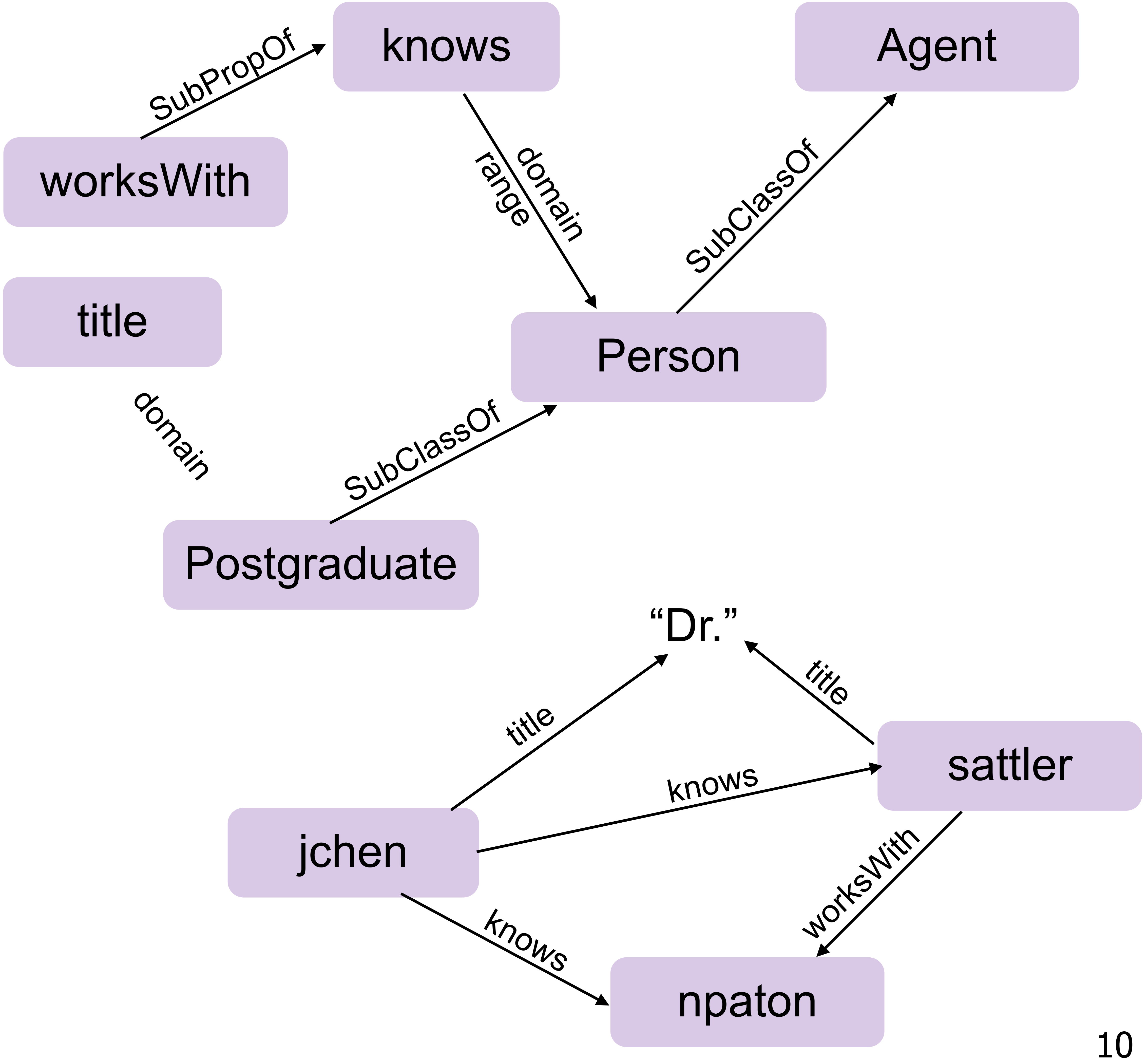
- objects + relations = knowledge?
  - context
  - meaning
  - understanding

- data: factual, about individuals
- knowledge: conceptual, about
  - concepts
  - their relations



Knowledge Representation & Reasoning

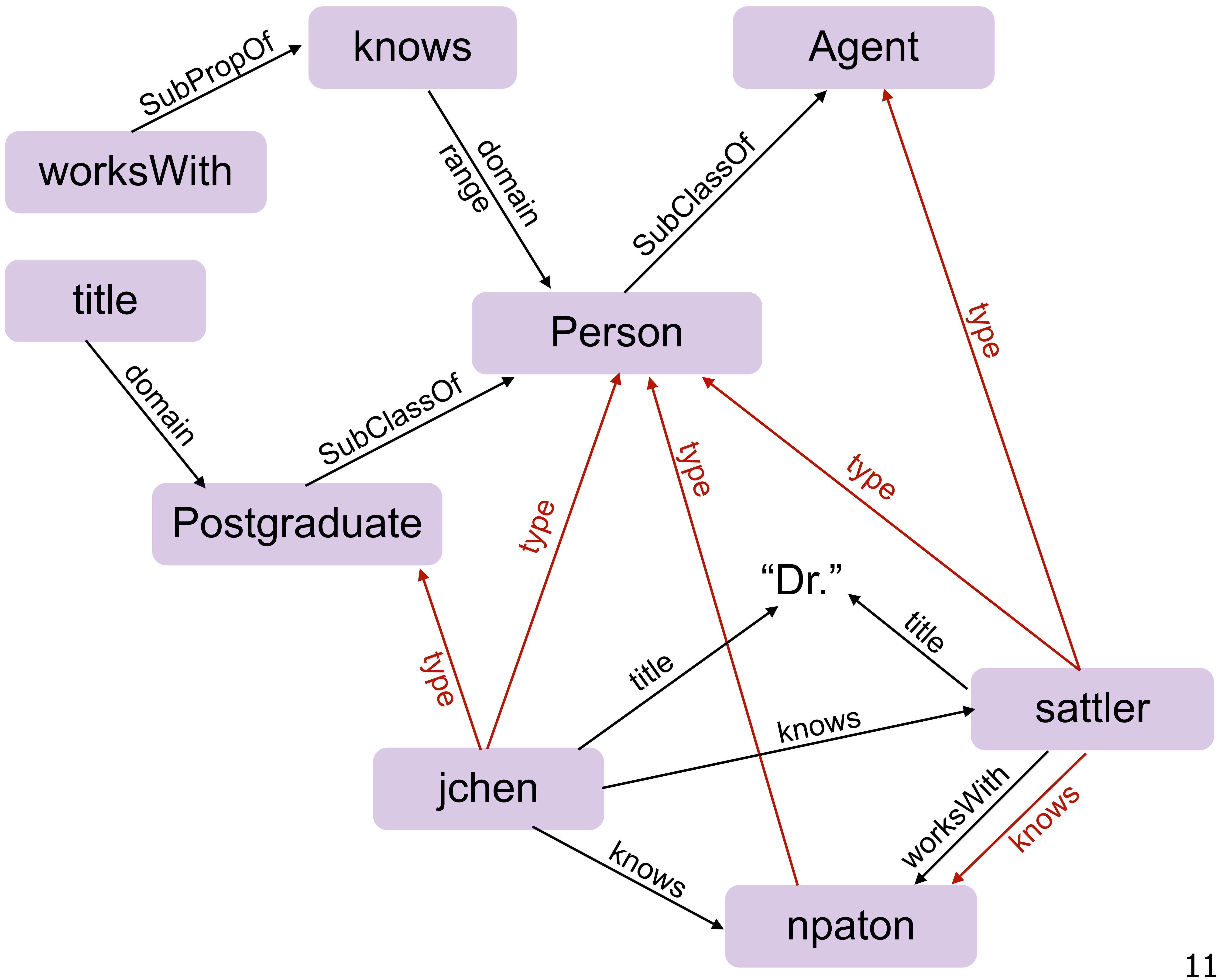
- store/access knowledge
  - factual, about individuals
  - conceptual
    - relevant concepts
    - their relations
      - hierachical/logical
      - domain dependent
- *reason* about it
  - draw conclusions from the explicitly stated knowledge



via well-understood *algorithms* implemented in powerful *reasoners*

Reasoning

- draw conclusions from the explicitly stated knowledge
- E.g.,
  - jchen is of type Person
  - sattler is of type Person
  - sattler knows npaton
  - npaton is of type Person
  - jchen is of type Postgraduate
  - sattler is of type Agent
  - …



KR&R and *symbolic* AI

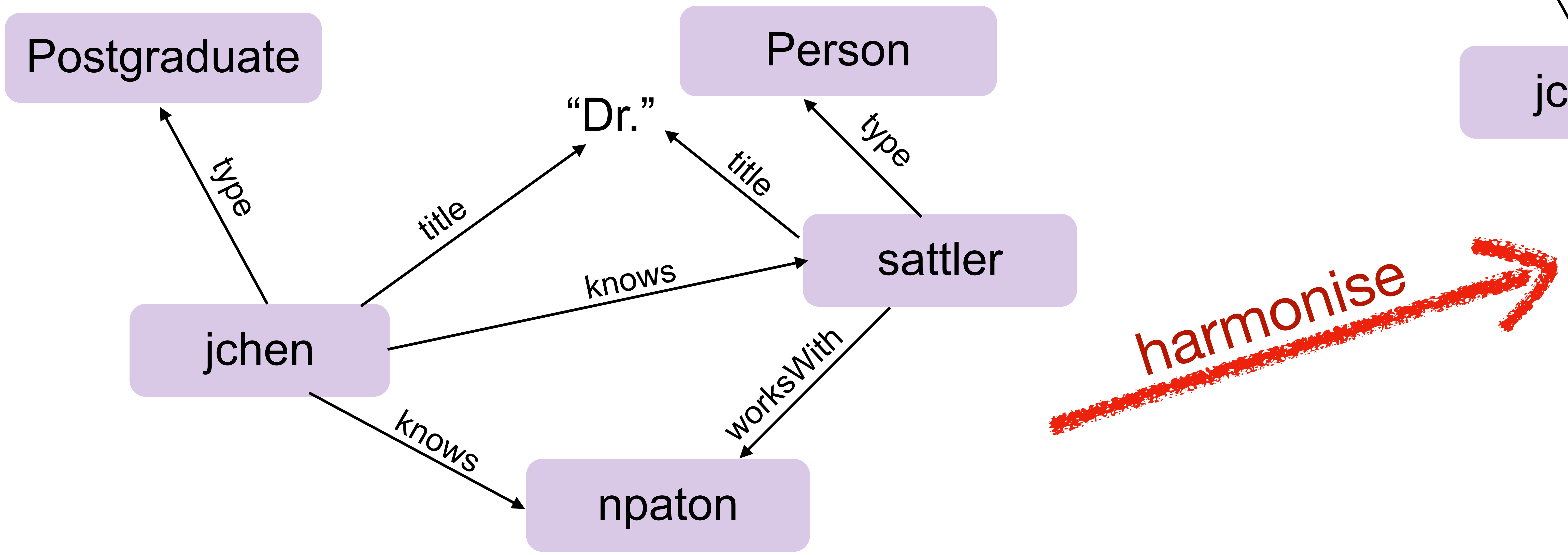
- in KR&R, we use *symbols* for
  - concepts
    - eg Person, Postgraduate, …
  - relations
    - eg title, knows
  - individuals
  - …and build *intelligent* systems to deal with these
- where does our knowledge (graph) come from?
  - domain experts, database, …
- what do we do with our knowledge?
  - use them to *harmonise KGs*

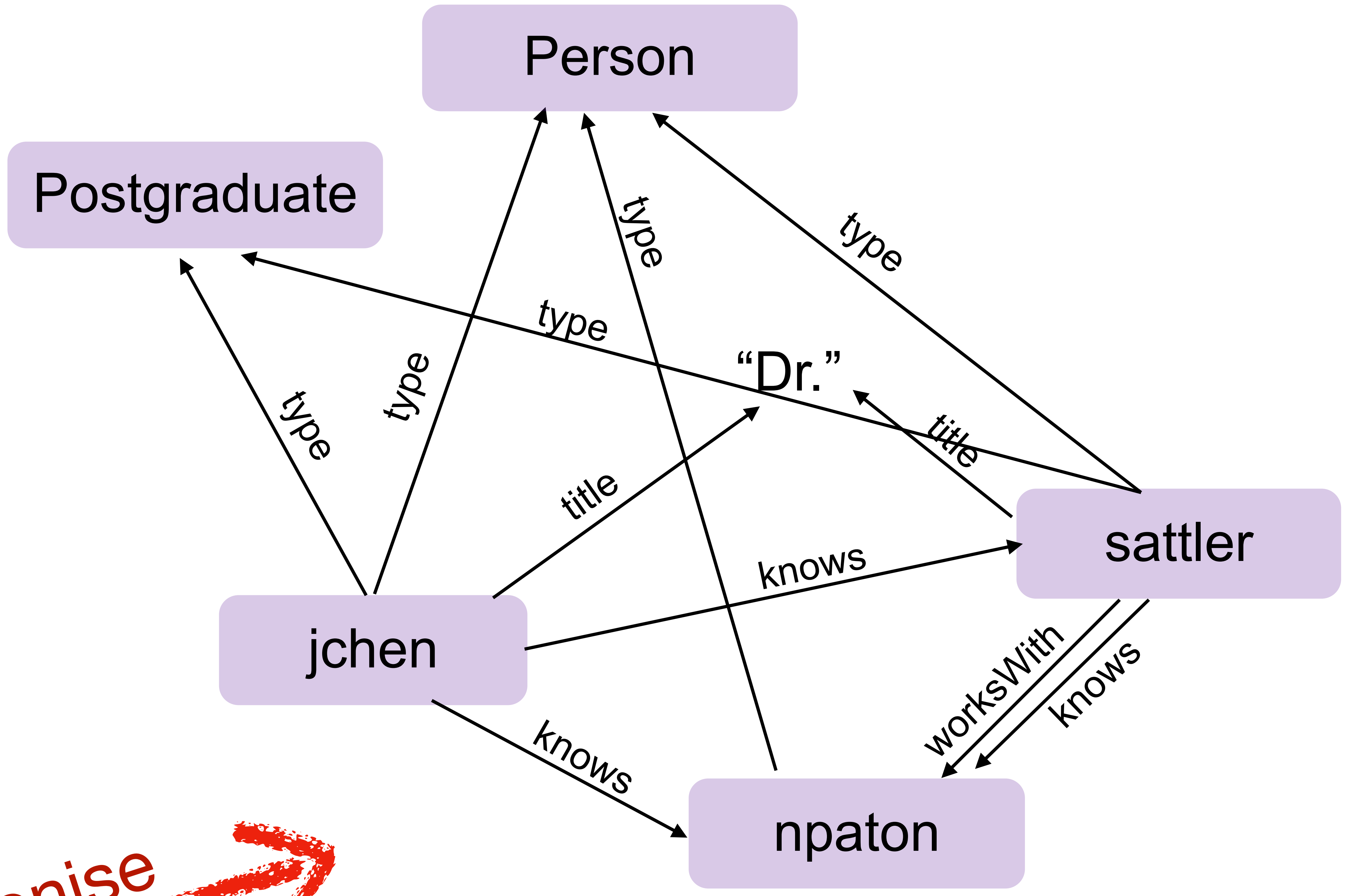
capable of reasoning

add inferred types & links

Harmonise KGs to…

- make explicit knowledge explicit
- reflect background knowledge
  - no need to guess/*predict links*
- increase *regularity* in KG
  - ➡ improve *machine learnability*





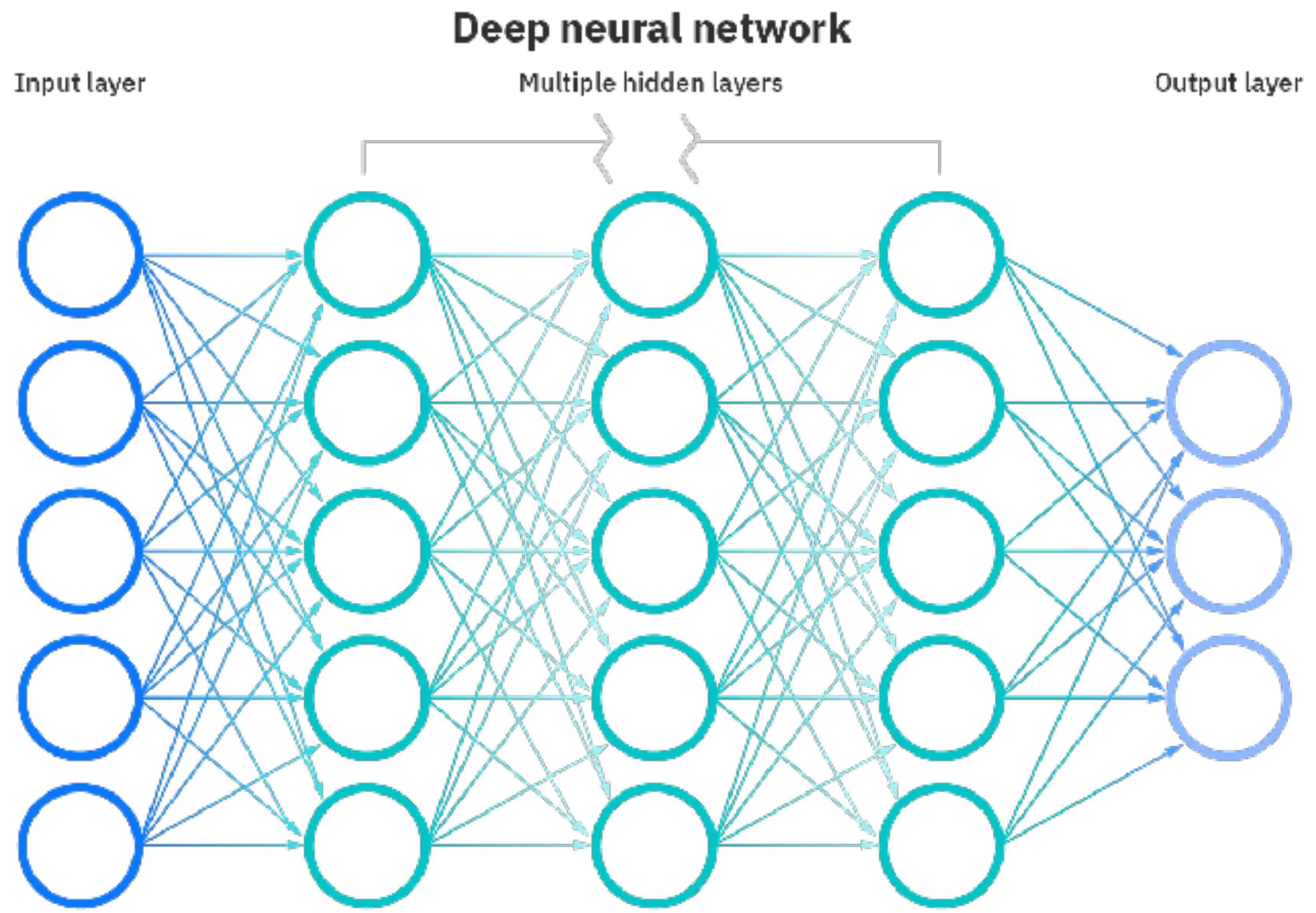
Sub-symbolic AI — Machine Learning

In sub-symbolic AI, we

- build/train *a model*
  - artificial/deep/graph neural networks
  - statistical methods
  - …

  to spot/learn regularities/patterns from data

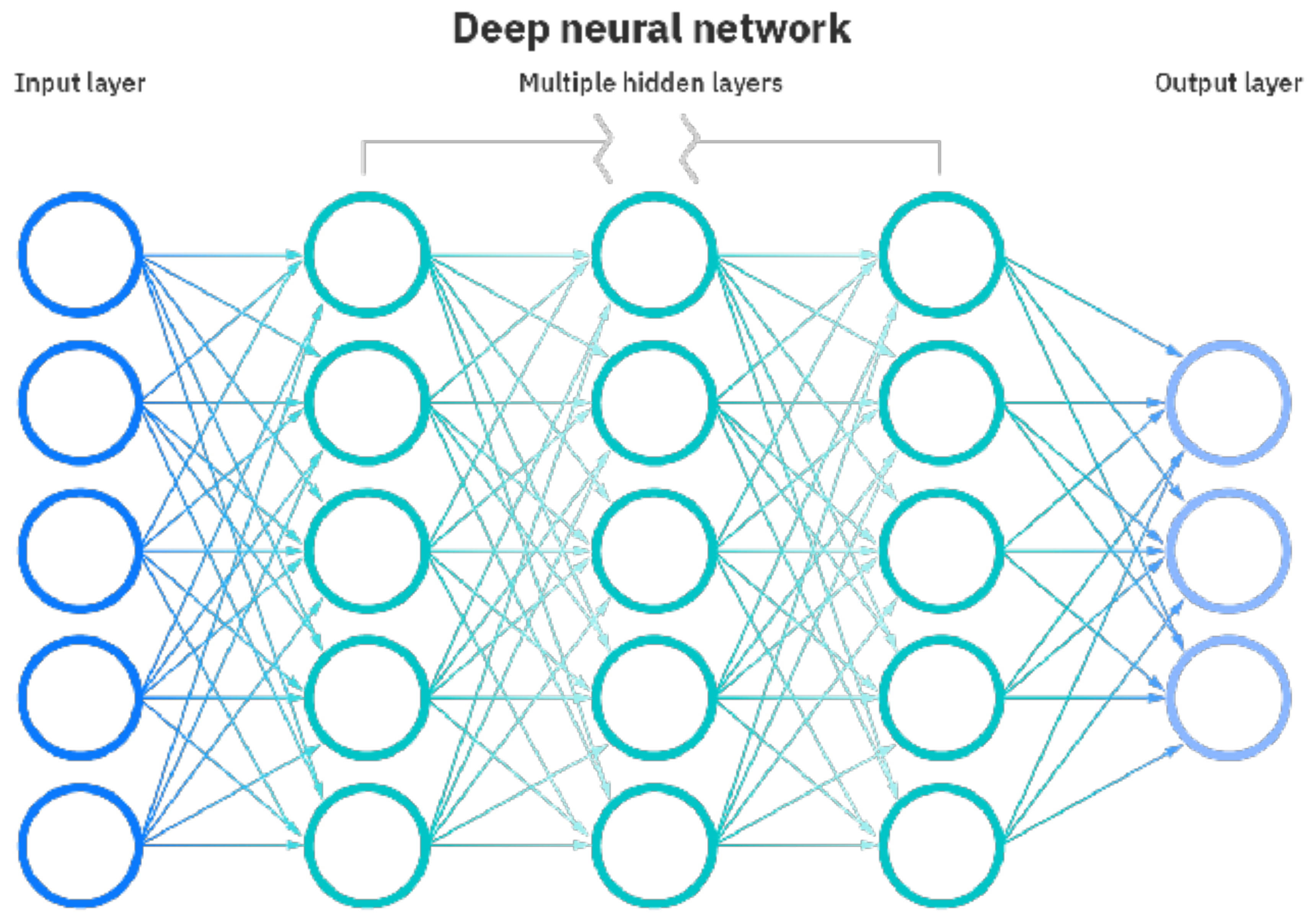
- don't need to
  - formulate explicit rules
  - identify the right terms/symbols
- get amazing results/performance



Sub-symbolic AI — Machine Learning

In sub-symbolic AI, we

- need huge amounts of data for training
  - costly
  - not always available
- bias in training data goes into model
- find it hard to analyse behaviour
  - other than testing it
  - independent on data



Neuro-symbolic KR&R

combines approaches

- neural/sub-symbolic and
- symbolic

- symbolic ⇒ sub-symbolic
  - inject *background knowledge* into ML models
  - informed embeddings

- sub-symbolic ⇒ symbolic
  - learn rules from data
  - learn facts
  - learn how to reason

Today:

- Knowledge Graphs
  - RDF
  - factual and conceptual knowledge
- Querying of and Reasoning with KGs
  - SPARQL
  - RDFS
  - SHACL
  - Materialisation of reasoning results

Day 1
Knowledge Graphs

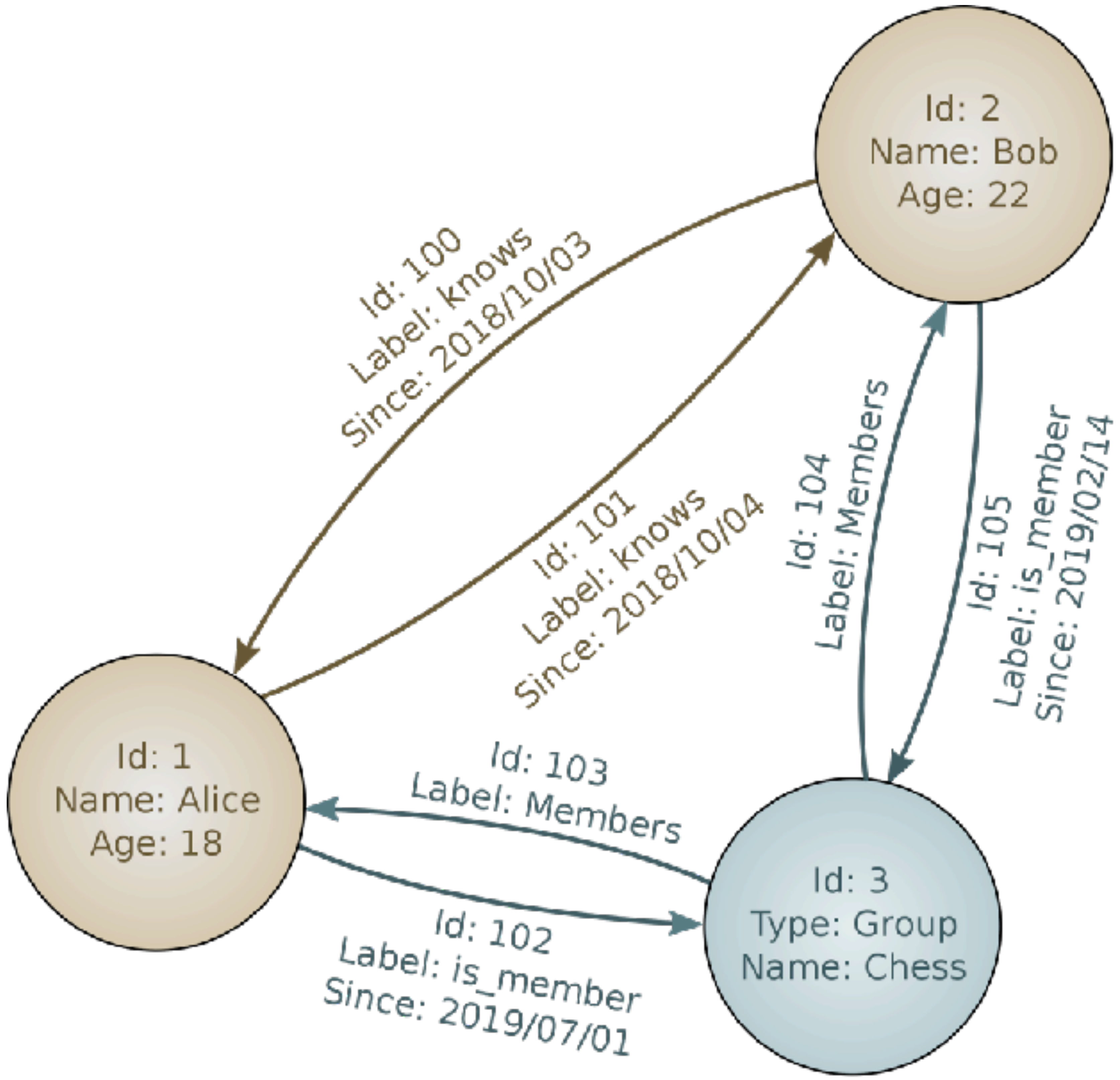
Uli Sattler
Professor in Computer Science
University of Manchester

ESSAI 2024 Athens

(Knowledge) Graphs

come in different shapes

- Google KG
  - both GDB and content
- Neo4J
- Amazon Neptune
- Arango
- RDF
  - a W3C standard for KGs
- …

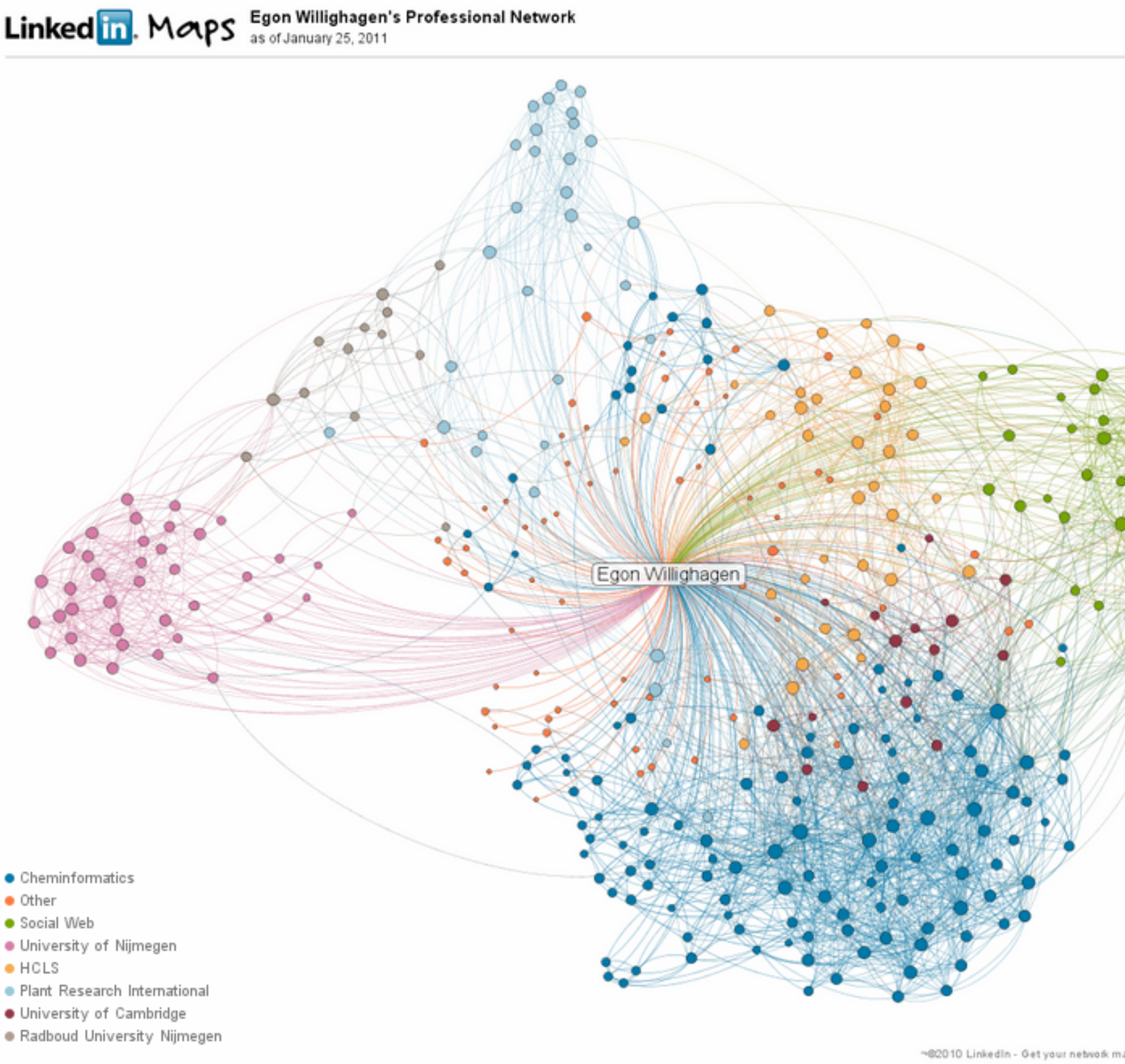


- nodes
- edges - between nodes
- labels - on edges and nodes

(Knowledge) Graphs

are everywhere

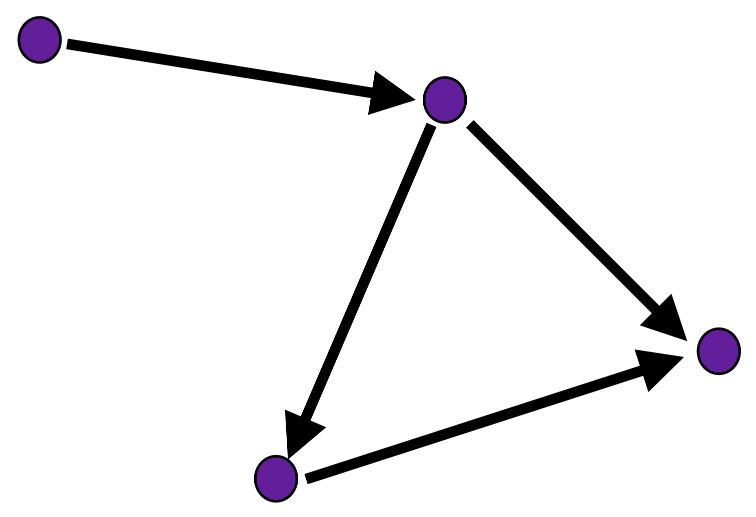
- “knows” on people
  - social networks
- “isRelatedTo” in genealogy
- “interactsWith” on proteins
  - bio-chemistry
- “educatedAt” etc in Wikidata
- …



Graph Basics

- A graph G = (V,E) is a pair with
  - V a set of vertices (also called) nodes, and
  - $E \subseteq V \times V$ a set of edges

- Variants:
  - (in)finite graphs: V is a (in)finite set
  - (un)directed graphs: E (is) is not a symmetric relation
    - i.e., if G is undirected, then $(x,y) \in E$ implies $(y,x) \in E$.
  - node/edge labelled graphs: a label set S, labelling function(s)
    - $L: V \rightarrow S$ (node labels)
    - $L: E \rightarrow S$ (edge labels)



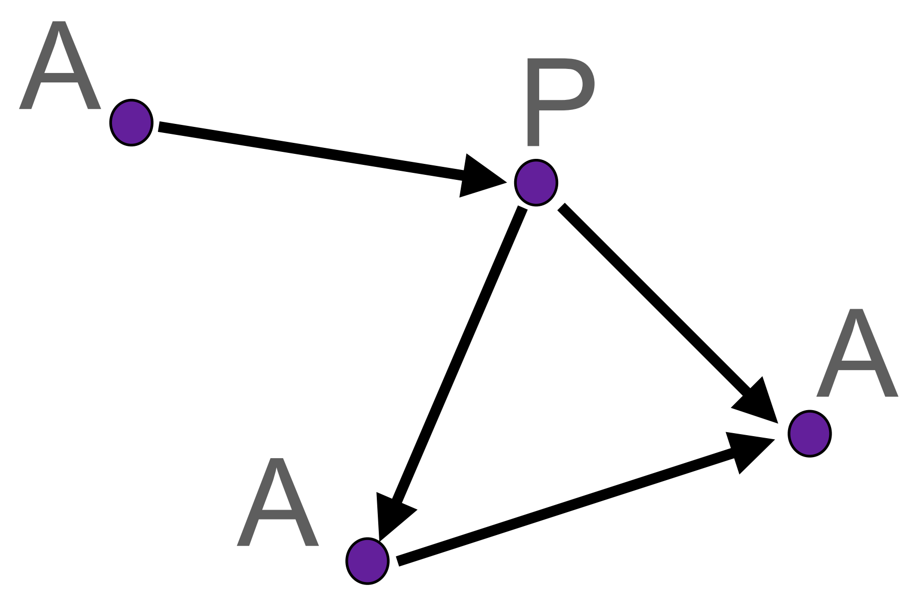
Example:

G = ({a,b,c,d},
 {(a,b), (b,c), (b,d), (c,d)})

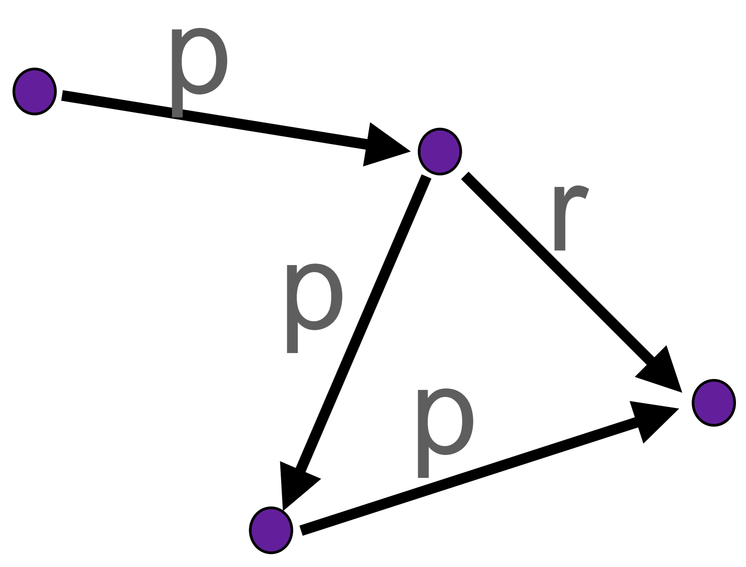
- where are a,….,d in this graph's picture?

Graph Basics (2)

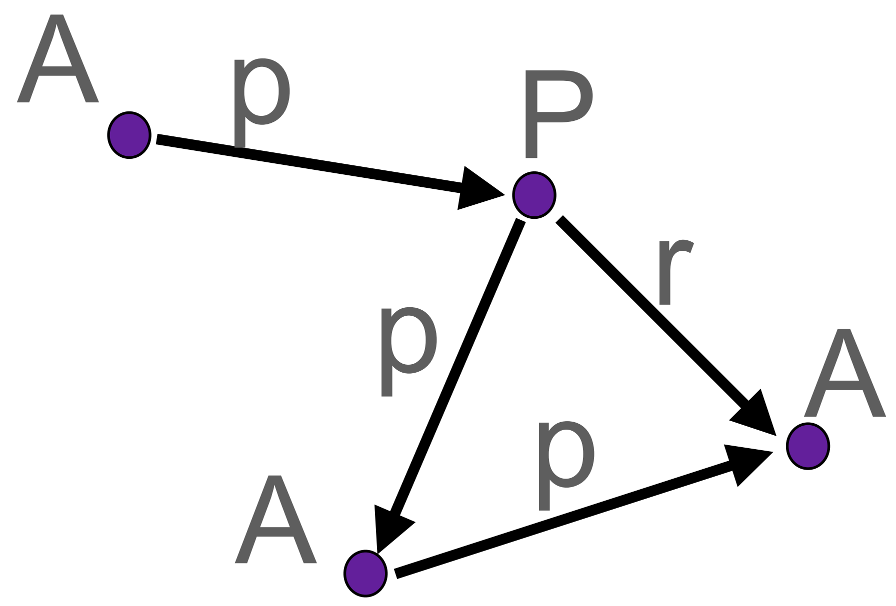
- Example: node-labelled graph
  - L: V $\rightarrow$ {A,P}



- Example: edge-labelled graph
  - L: E $\rightarrow$ {p,r,s}



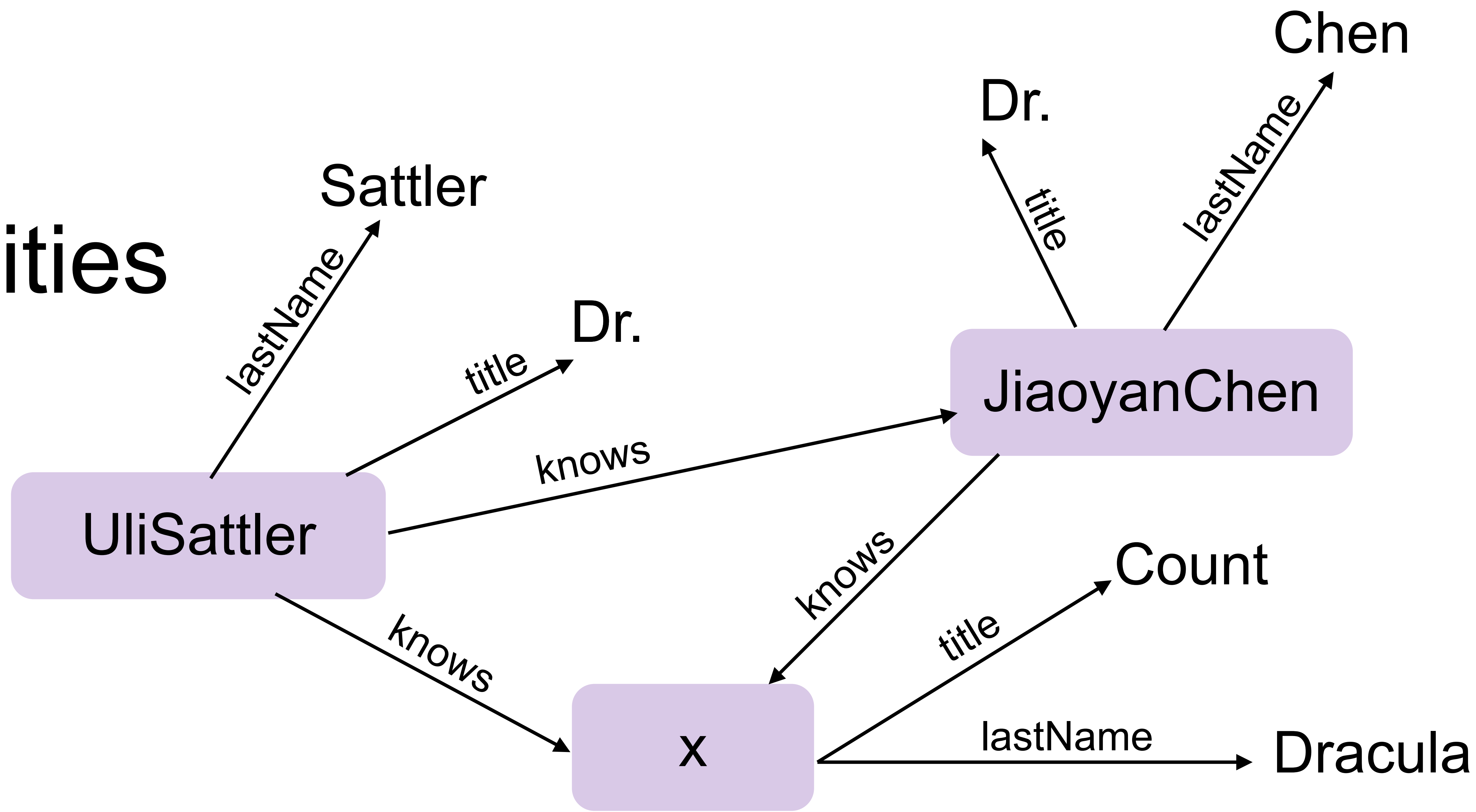
- Example: node-and-edge-labelled graph
  - L: V $\rightarrow$ {A,P}
  - L: E $\rightarrow$ {p,r,s}



Knowledge Graph

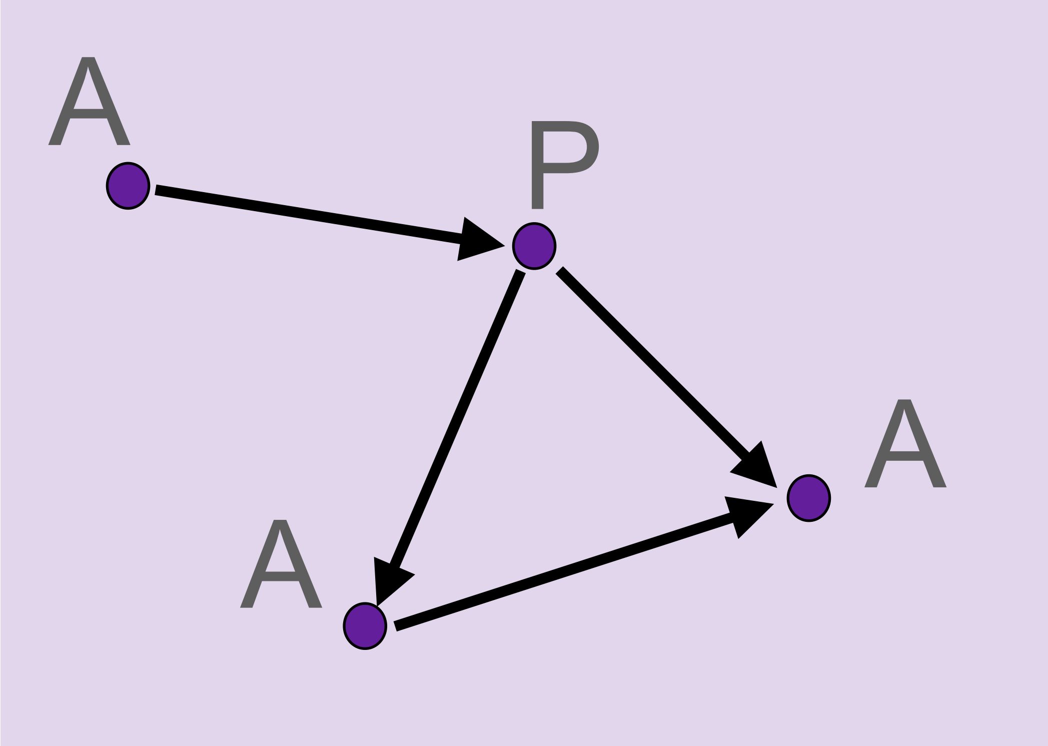
- Nodes ~ entities
  - possibly with *attributes* for *features* of nodes
- Edges ~ relations between entities
  - edge labels to describe kind

- Great for
  - many-to-many relations
  - cyclic relations
  - path queries

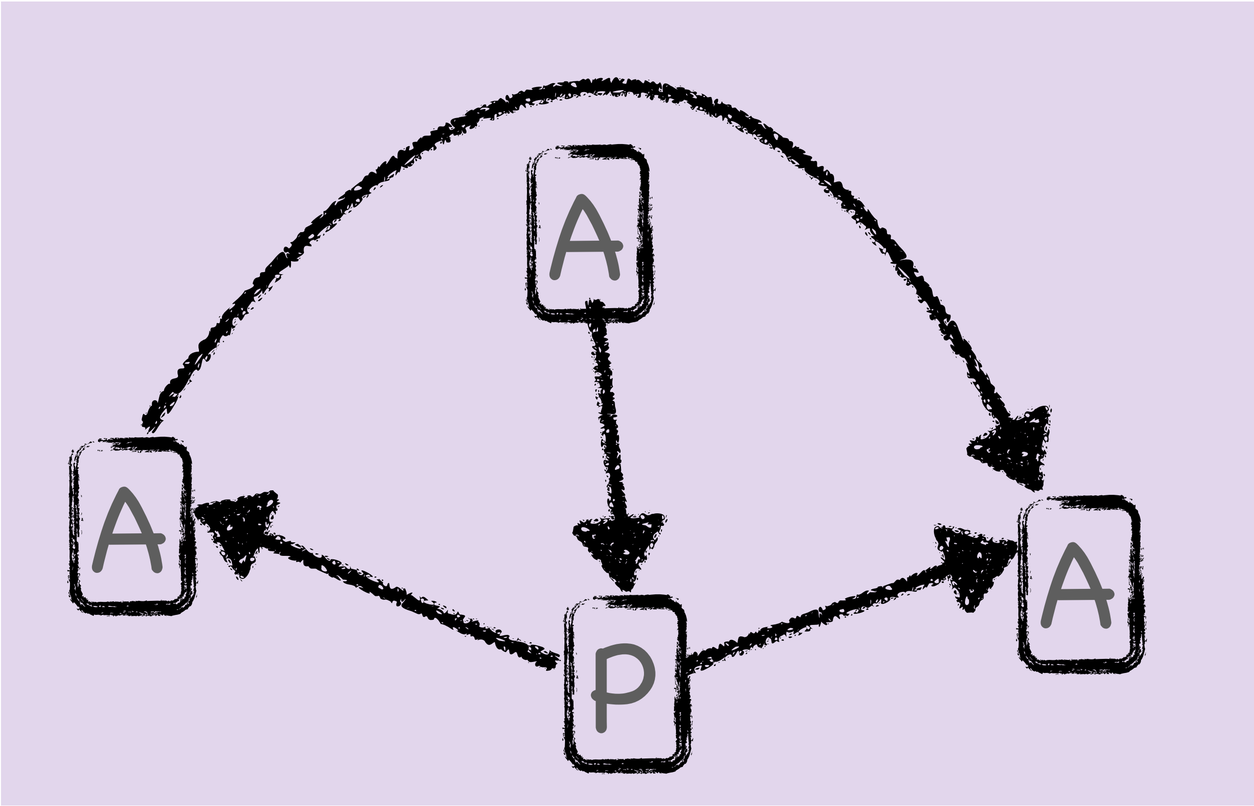


Graph Basics: **External** Representation

- Pictures are a bad external representations for graphs



=



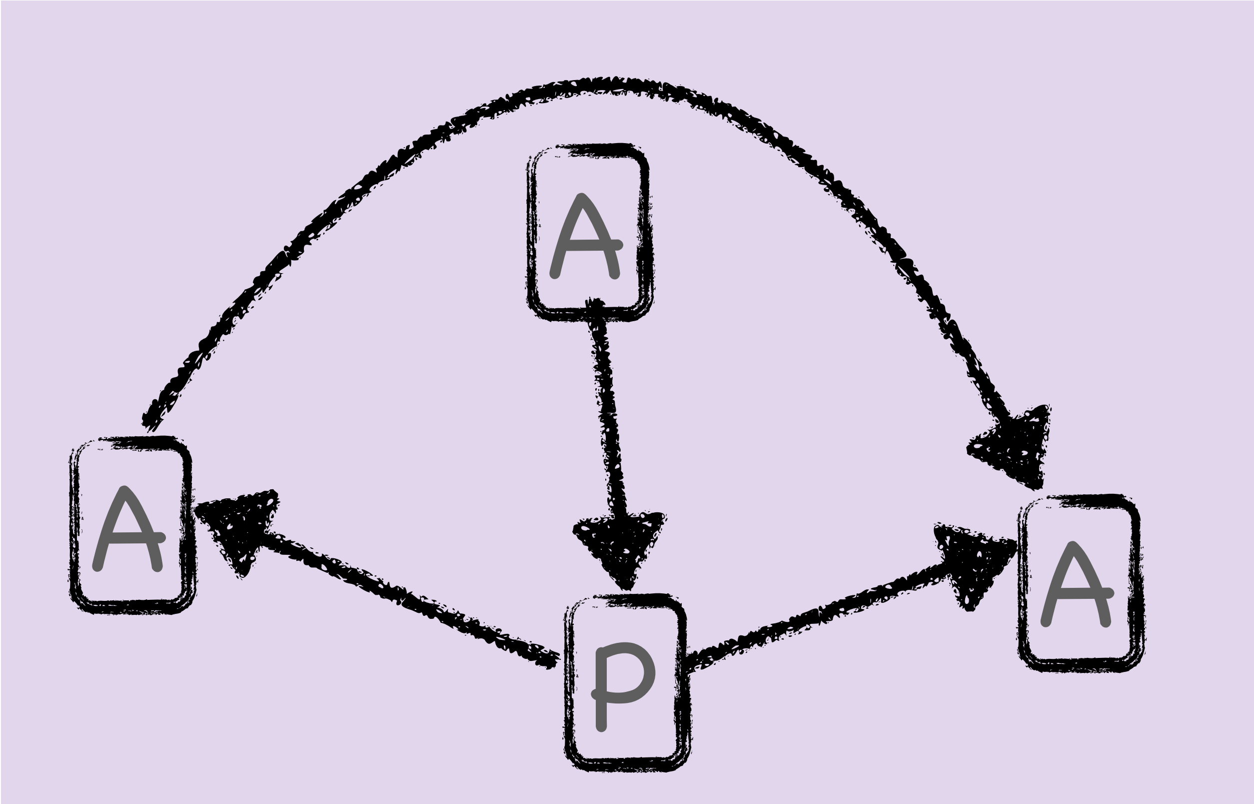
= =

G = ({a,b,c,d},
{(a,b), (b,c), (b,d), (b,c)},
L: V ⟶ {A,P}
L: a ↦ A, b ↦ P, c ↦ A, d ↦A)

= …

Graph Basics: External Representation

- Pictures are a bad **external representations** for graphs
  - capture loads of irrelevant information
    - colour
    - location, geometry,
    - shapes, strokes, …
  - what if labels are more complex/structured?
  - how do we *parse* a picture into an **internal representation**?
    - what is a *good* internal representation?



Day 1: RDF
a graph-shaped data model

Uli Sattler
Professor in Computer Science
University of Manchester

ESSAI 2024 Athens

MANCHESTER 1824
The University of Manchester

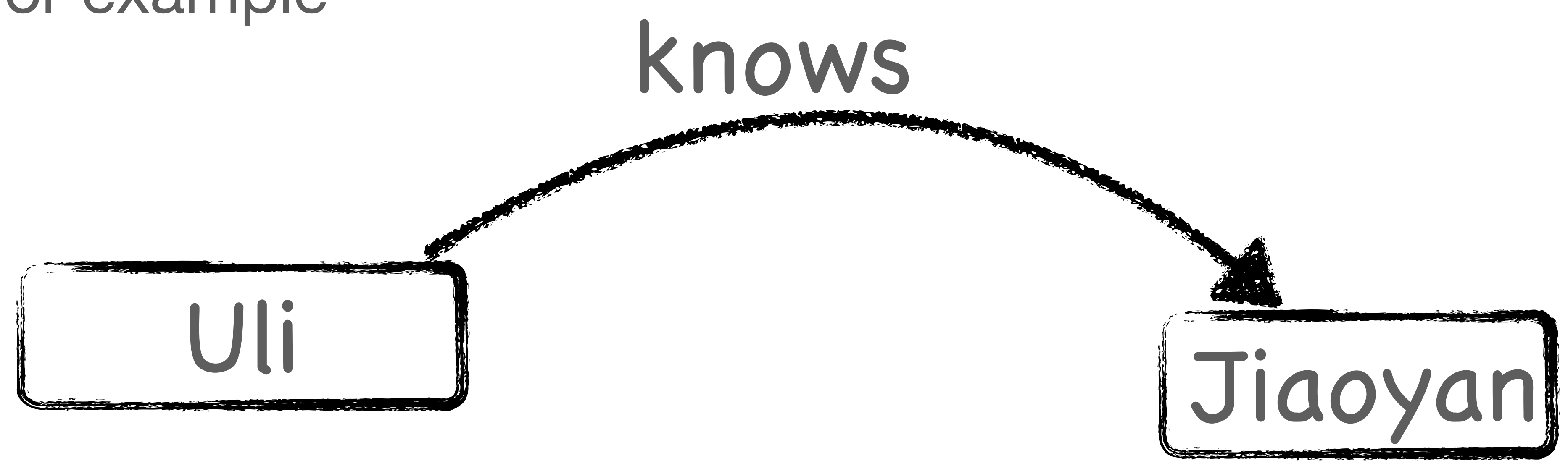
A Graph Formalism: RDF…why?

- RDF is an
  - *independent* data model
  - standardised by W3C
  - supported by various of the about GBDMSs
- other graph formalisms/data models are available, eg
  - Neo4J
  - GraphDB
  - MongoDB

A Graph Formalism: RDF

- **R**esource **D**escription **F**ramework
- a **graph-based** data structure formalism
- a W3C standard for the representation of **graphs**
- comes with various syntaxes for External Representation
- is based on **triples** *(subject, predicate, object)*

For example

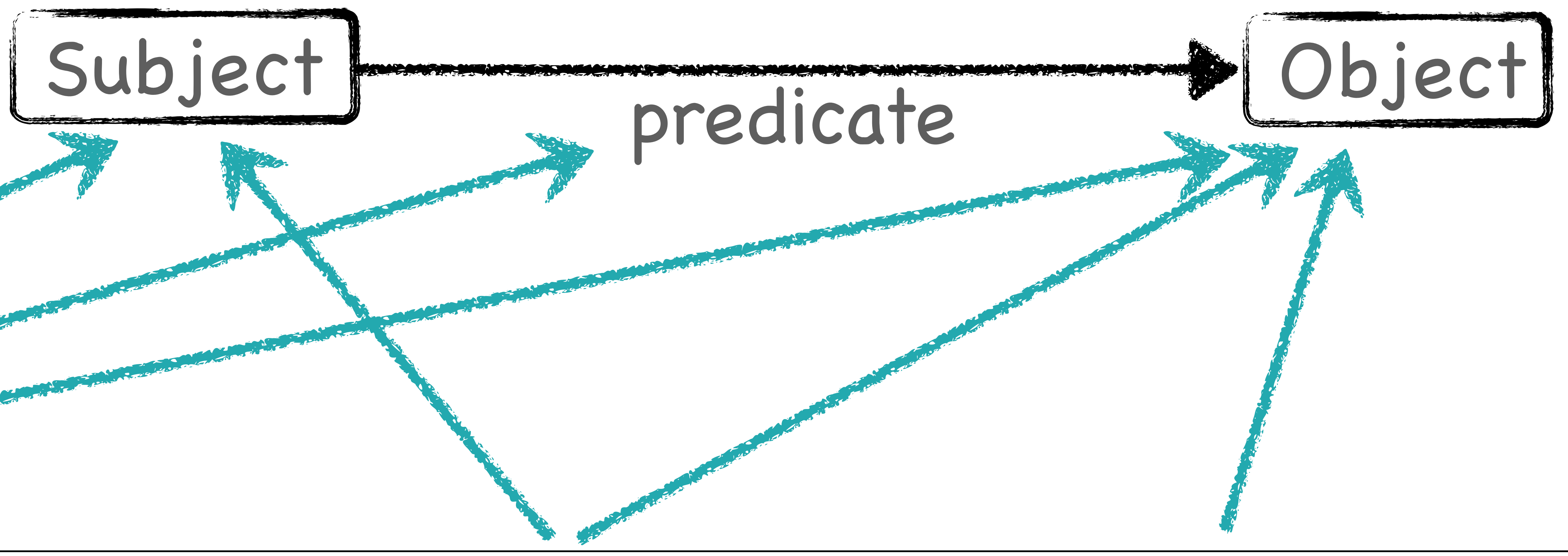


RDF: basics

U: URIs (for *resources*), incl. rdf:type
B: Blank nodes
L: Literals

- an RDF **graph G** is a **set** of **triples** $\{(s_i, p_i, o_i) \mid 1 \leq i \leq n\}$

- where each
  - $s_i \in U \cup B$
  - $p_i \in U$
  - $o_i \in U \cup B \cup L$



U: URIs, incl. rdf:type, **B**: Blank nodes, **L**: Literals

RDF: basics

U: URIs (for *resources*), incl. rdf:type
B: Blank nodes
L: Literals

- an RDF **graph G** is a **set** of **triples** $\{(s_i, p_i, o_i) \mid 1 \leq i \leq n\}$
- where each
  - $s_i \in U \cup B$
  - $p_i \in U$
  - $o_i \in U \cup B \cup L$

a graph ???

{(ex:jchen, foaf:knows, ex:sattler),
(ex:jchen, rdf:type, foaf:Person),
(ex:jchen, rdf:type, foaf:Agent),
(ex:sattler, foaf:title, “Dr.”),
(ex:sattler, foaf:lastName, “Sattler”),
(ex:jchen, foaf:title, “Dr.”),
(ex:sattler, foaf:knows, ex:npaton),
(ex:jchen, foaf:knows, ex:npaton) }

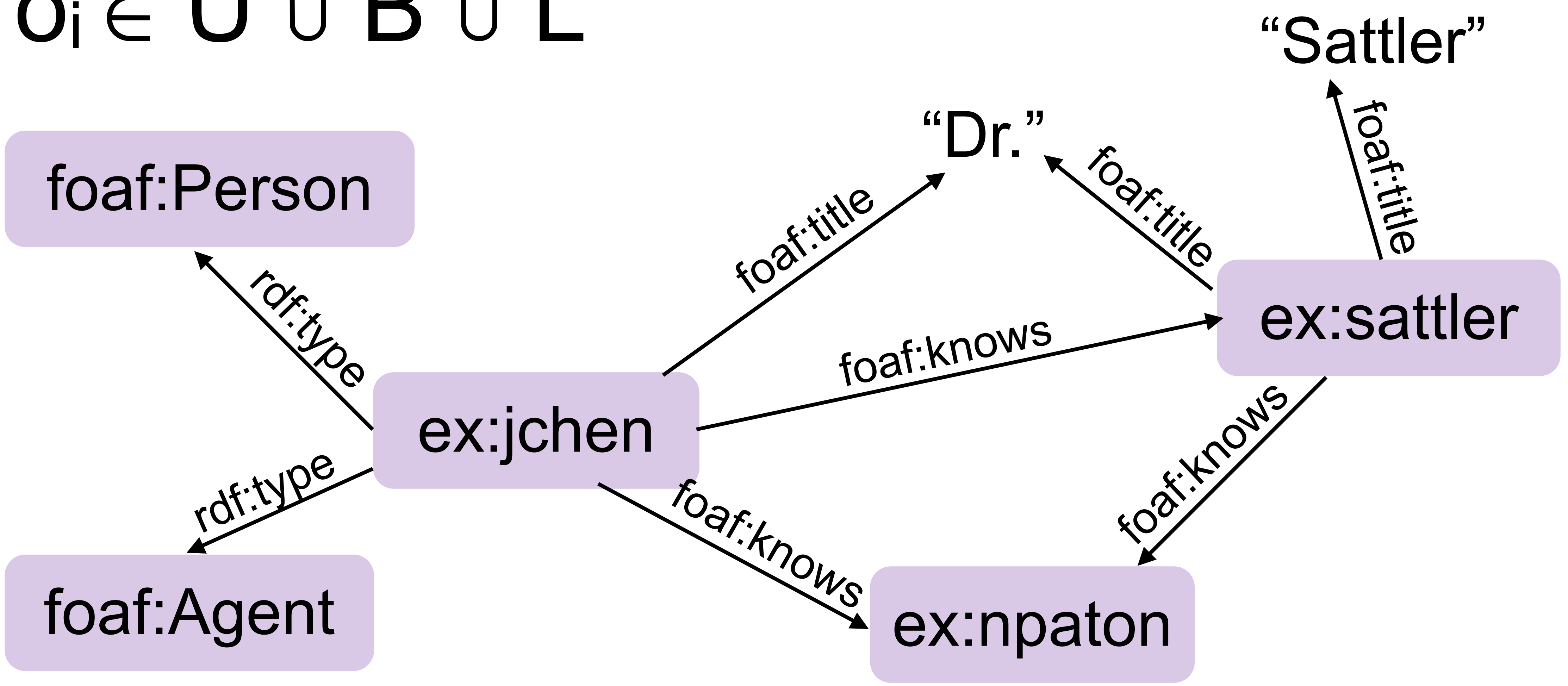
abbreviate: ex: for http://www.cs.man.ac.uk/
foaf: for http://xmlns.com/foaf/0.1/

RDF: basics

U: URIs (for *resources*), incl. rdf:type
B: Blank nodes
L: Literals

- an RDF **graph G** is a **set** of **triples** $\{(s_i, p_i, o_i) \mid 1 \leq i \leq n\}$
- where each
  - $s_i \in U \cup B$
  - $p_i \in U$
  - $o_i \in U \cup B \cup L$

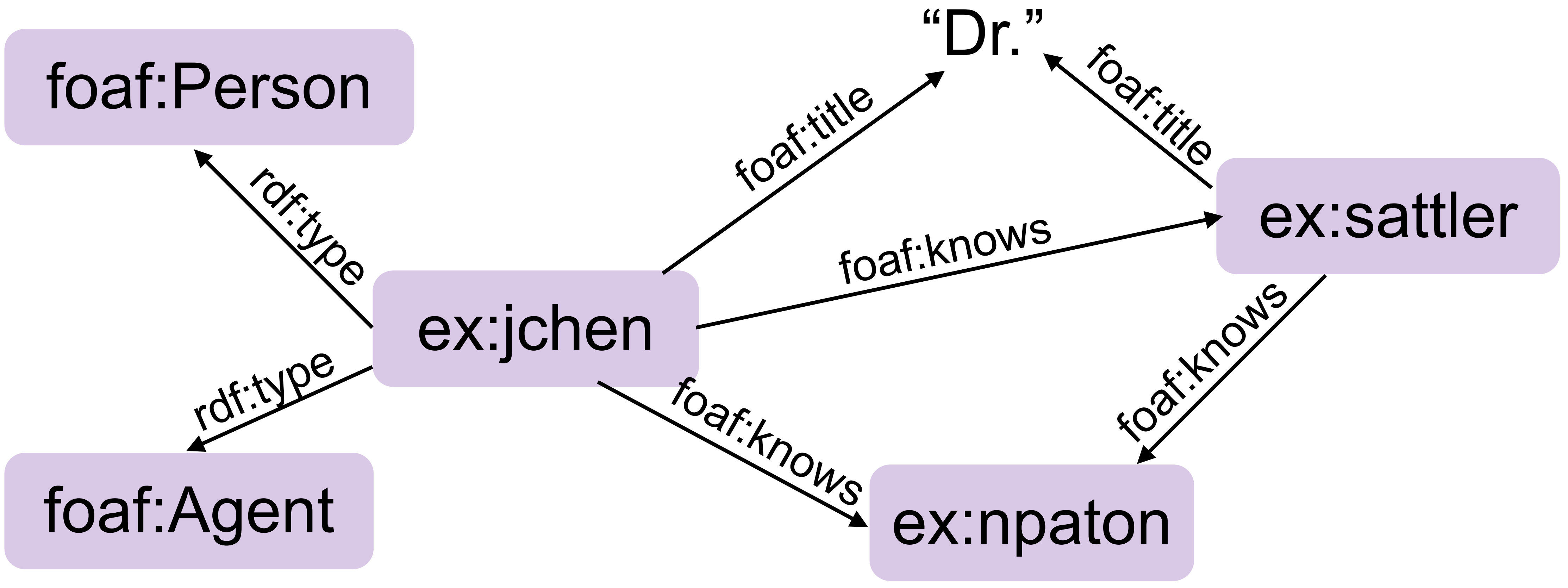
a graph !!!



{(ex:jchen, foaf:knows, ex:sattler),
(ex:jchen, rdf:type, foaf:Person),
(ex:jchen, rdf:type, foaf:Agent),
(ex:sattler, foaf:title, “Dr.”),
(ex:sattler, foaf:lastName, “Sattler”),
(ex:jchen, foaf:title, “Dr.”),
(ex:sattler, foaf:knows, ex:npaton),
(ex:jchen, foaf:knows, ex:npaton) }

abbreviate: ex: for http://www.cs.man.ac.uk/
foaf: for http://xmlns.com/foaf/0.1/

RDF syntaxes

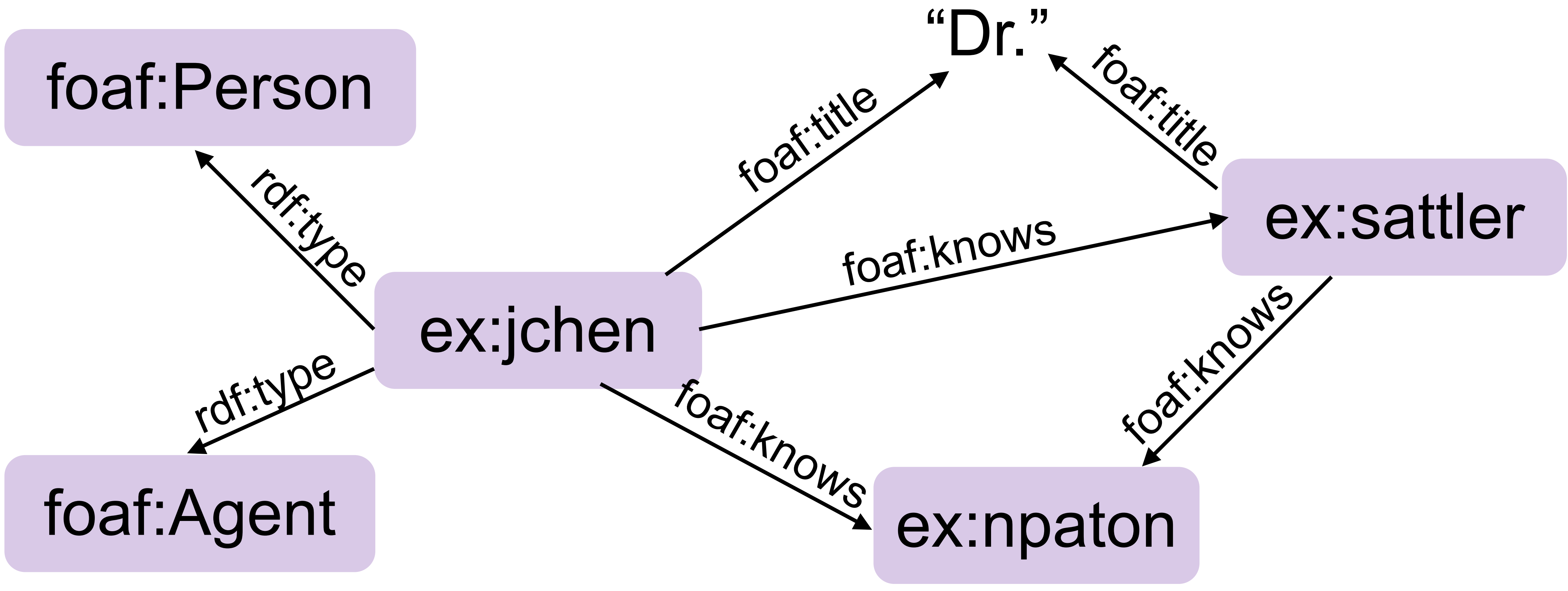


- "serialisation formats"
  - for ExtRep of RDF graphs
    - graphs are IntReps!
- there are several:
  - **Turtle**
  - N-Triples
  - JSON-LD
  - N3
  - RDF/XML
  - …

~~{(ex:jchen, foaf:knows, ex:sattler),
(ex:jchen, rdf:type, foaf:Person),
(ex:jchen, rdf:type, foaf:Agent),
(ex:sattler, foaf:title, "Dr."),
(ex:sattler, foaf:lastName, "Sattler"),
(ex:jchen, foaf:title, "Dr."),
(ex:sattler, foaf:knows, ex:npaton),
(ex:jchen, foaf:knows, ex:npaton) }~~

RDF syntaxes

- “serialisation formats”
  - for ExtRep of RDF graphs
    - graphs are IntReps!
- there are several:
  - **Turtle**
  - N-Triples
  - JSON-LD
  - N3
  - RDF/XML
  - …

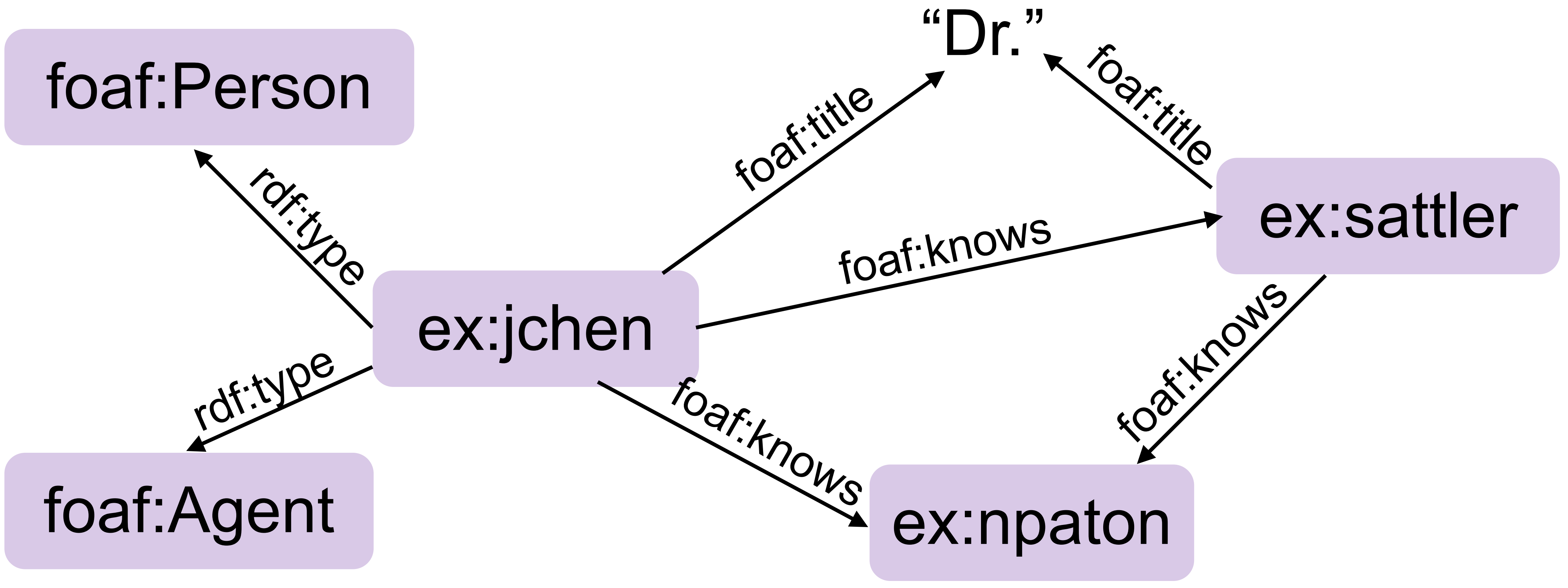


7 triples in **Turtle**:

```
@prefix rdf:   <http://www.w3.org/
1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex:   <http://www.cs.man.ac.uk/> .

ex:sattler
  foaf:title  "Dr." ;
  foaf:lastName  “Sattler" ;
  foaf:knows ex:jchen ;
  rdf:type foaf:Person ;
ex:jchen
 foaf:title  "Dr." ;
 foaf:knows ex:npaton ;
 foaf:knows ex:sattler ;
.
```

{(ex:jchen, foaf:knows, ex:sattler),
(ex:jchen, rdf:type, foaf:Person),
(ex:jchen, rdf:type, foaf:Agent),
(ex:sattler, foaf:title, “Dr.”),
(ex:sattler, foaf:lastName, “Sattler”),
(ex:jchen, foaf:title, “Dr.”),
(ex:sattler, foaf:knows, ex:npaton),
(ex:jchen, foaf:knows, ex:npaton) }

RDF syntaxes



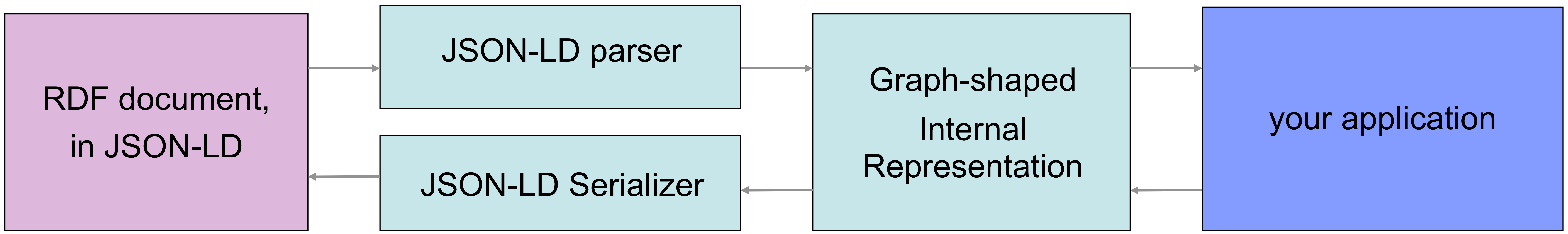
- "serialisation formats"
  – for ExtRep of RDF graphs
    – graphs are IntReps!
- there are several:
  –Turtle
  –N-Triples
  –**JSON-LD**
  –N3
  –RDF/XML
  –…

Triples in **JSON-LD**:

```
{"@context": {
   "Person": "http://xmlns.com/foaf/0.1/Person",
   "title": "http://xmlns.com/foaf/0.1/title",
   "lastName": "http://xmlns.com/foaf/0.1/lastName",
   "knows": "http://xmlns.com/foaf/0.1/knows"
 }, "@graph": [{
    "@id": "http://www.cs.man.ac.uk/sattler",
    "@type": "Person" ,
    "title": "Dr." ,
    "lastName": "Sattler",
    "knows": "http://www.cs.man.ac.uk/npaton"
        }, {
    "@id": "http://www.cs.man.ac.uk/jchen",
    "@type": "Person" ,
    "title": "Dr." ,
    "lastName": "Chen",
    "knows": ["http://www.cs.man.ac.uk/npaton",
                  "http://www.cs.man.ac.uk/sattler"]}
]}
```

~~{(ex:jchen, foaf:knows, ex:sattler),
(ex:jchen, rdf:type, foaf:Person),
(ex:jchen, rdf:type, foaf:Agent),
(ex:sattler, foaf:title, "Dr."),
(ex:jchen, foaf:title, "Dr."),
(ex:sattler, foaf:knows, ex:npaton),
(ex:jchen, foaf:knows, ex:npaton) }~~

Parsing/serialising RDF graphs



- See eg https://json-ld.org/
- See eg https://github.com/RDFLib/rdflib
  - for Python parsers/serialisers/libraries
  - for RDF/XML, N3, NTriples, N-Quads, Turtle, …
  - with support for SPARQL for querying

Day 1: SPARQL

a query language for RDF

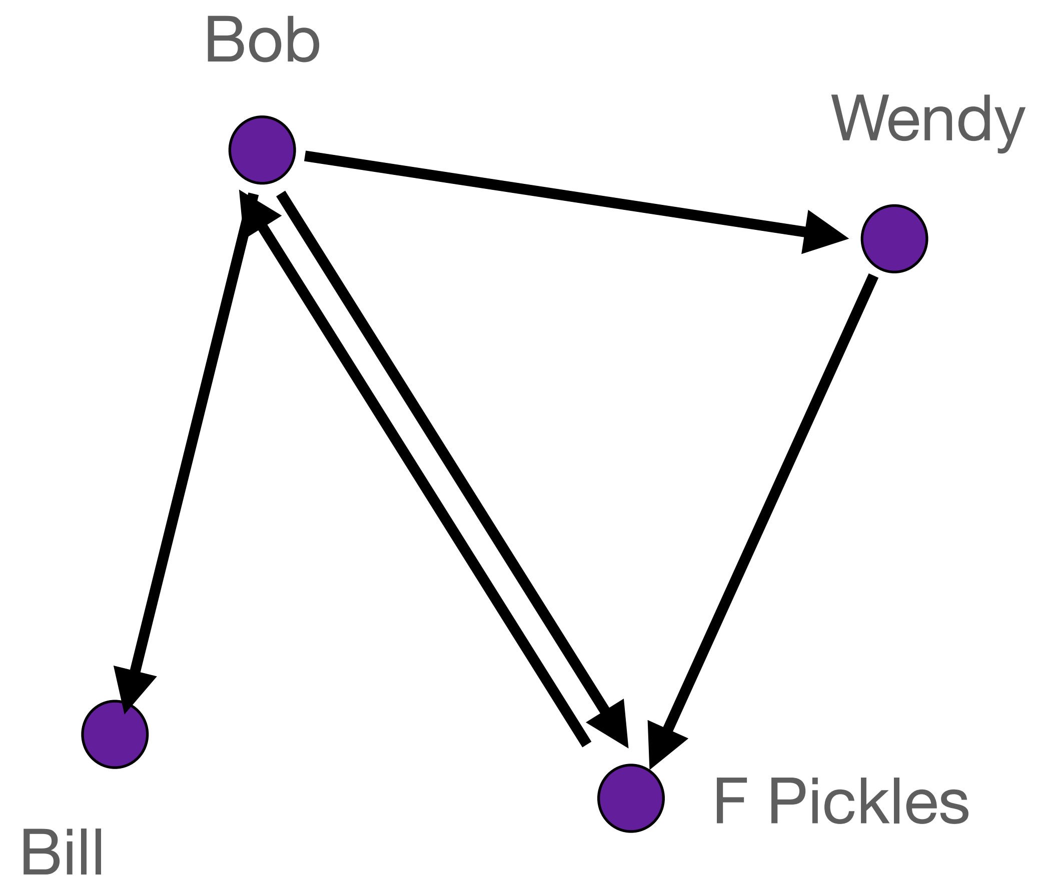
Uli Sattler
Professor in Computer Science
University of Manchester

ESSAI 2024 Athens

SPARQL

- We have
  - a data structure/internal representation: graphs!
  - schema languages (later: RDF, SHACL)
    - plus various external representions (Turtle, N3, N-triples, JSON-LD,..)
- For manipulating RDF graphs: you can use
  - libraries for your favourite programming language:
    - rdflib in Python
    - Jena, RDF4J, CommonsRDF, … in Java
    - …
  - a query language
    - SPARQL, a W3C standardised QL
    - Cypher, supported by Neo4j
      - http://neo4j.com/developer/cypher/
      - has “graph structural” features like “shortest path”
      - lacks “regular path” queries

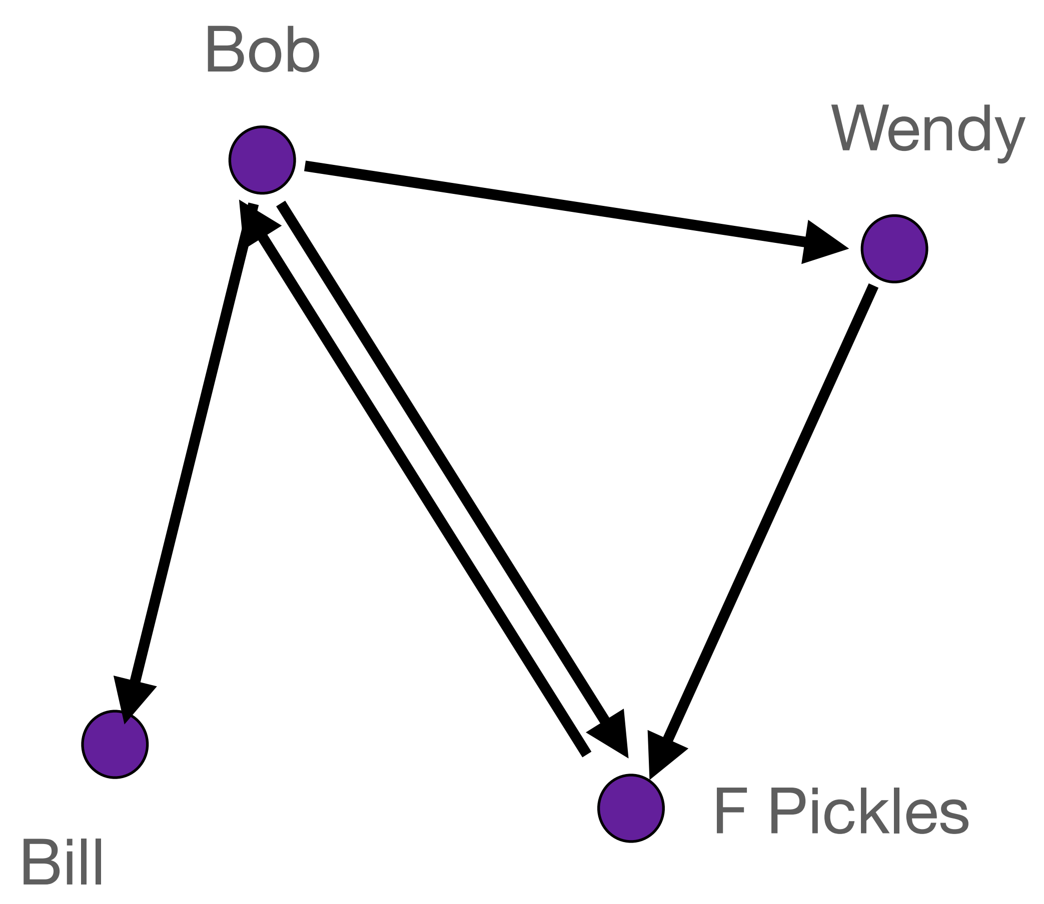
SPARQL: Basic Graph Patterns



- are at the core of SPARQL queries:
  - a *BGP* is a list/set of *triple patterns*
    - e.g., →
    - with abbreviations for shared subjects or predicates
    - separated by .
  - a *triple pattern* is a triple where *variables* can be used as subject, predicate, or object
    - e.g., {?x rdf:type foaf:Person}

```
ex:bobthebuilder
        foaf:firstName "Bob";
        foaf:lastName "Builder";
        foaf:knows ex:Wendy .
```

SPARQL: Clauses (1)



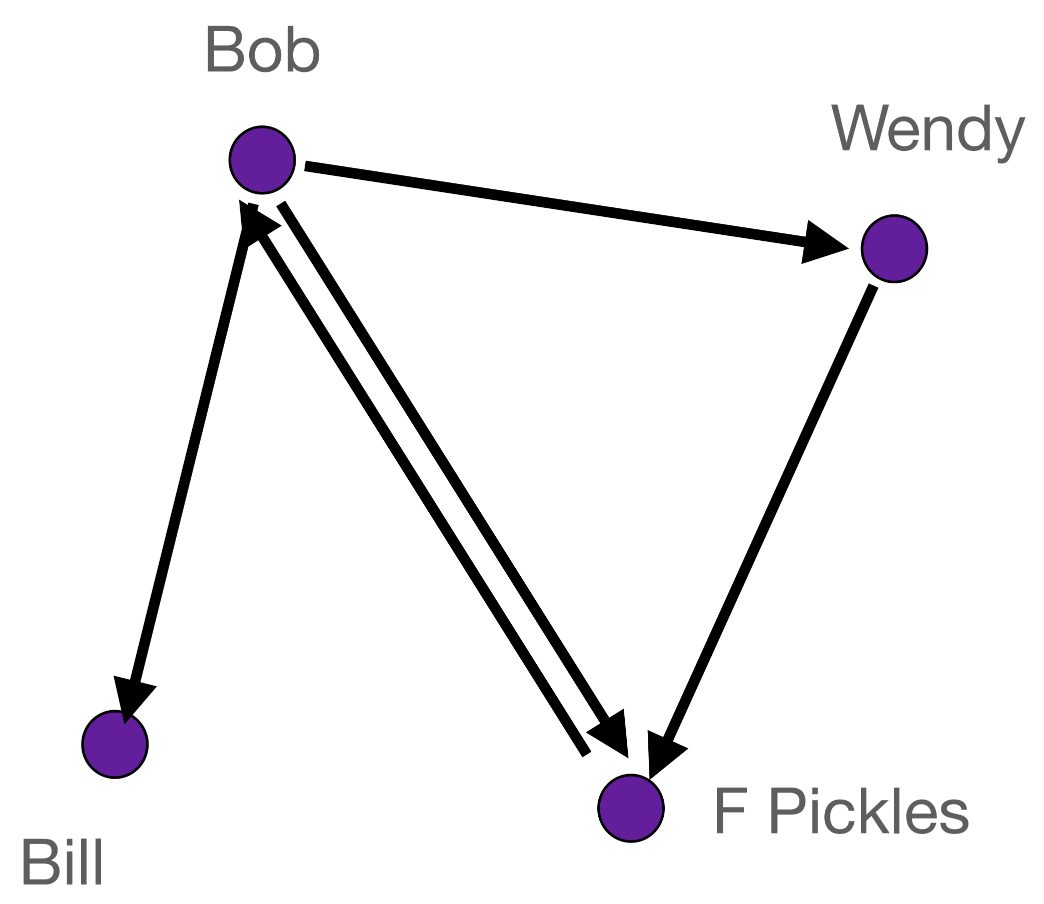
- We combine a BGP with a **query type**
  - ASK
    - e.g., ASK WHERE {ex:sattler rdf:type foaf:Person}
    - returns true or false (only)
  - SELECT
    - e.g., SELECT ?p WHERE {?p rdf:type foaf:Person}
    - very much like SQL SELECT
  - Careful:
    - ASK returns a Boolean (not an RDF graph!)
    - SELECT returns a table (not an RDF graph!)
    - SPARQL is *not* closed over graphs!
      - unusual: compare to SQL or XQuery!

SPARQL Clauses (2)

- There are two query types that return graphs:
  - CONSTRUCT
    - e.g., CONSTRUCT {?p rdf:type :Befriended}
WHERE {?p foaf:knows ?q}
    - like XQuery element and attribute constructors
  - DESCRIBE
    - e.g., DESCRIBE ?p WHERE {?p rdf:type foaf:Person}
    - implementation dependent!
    - returns a“description”
      - as a graph
      - whatever the service deems helpful!
      - similar to querying system tables in SQL

Examples: Data

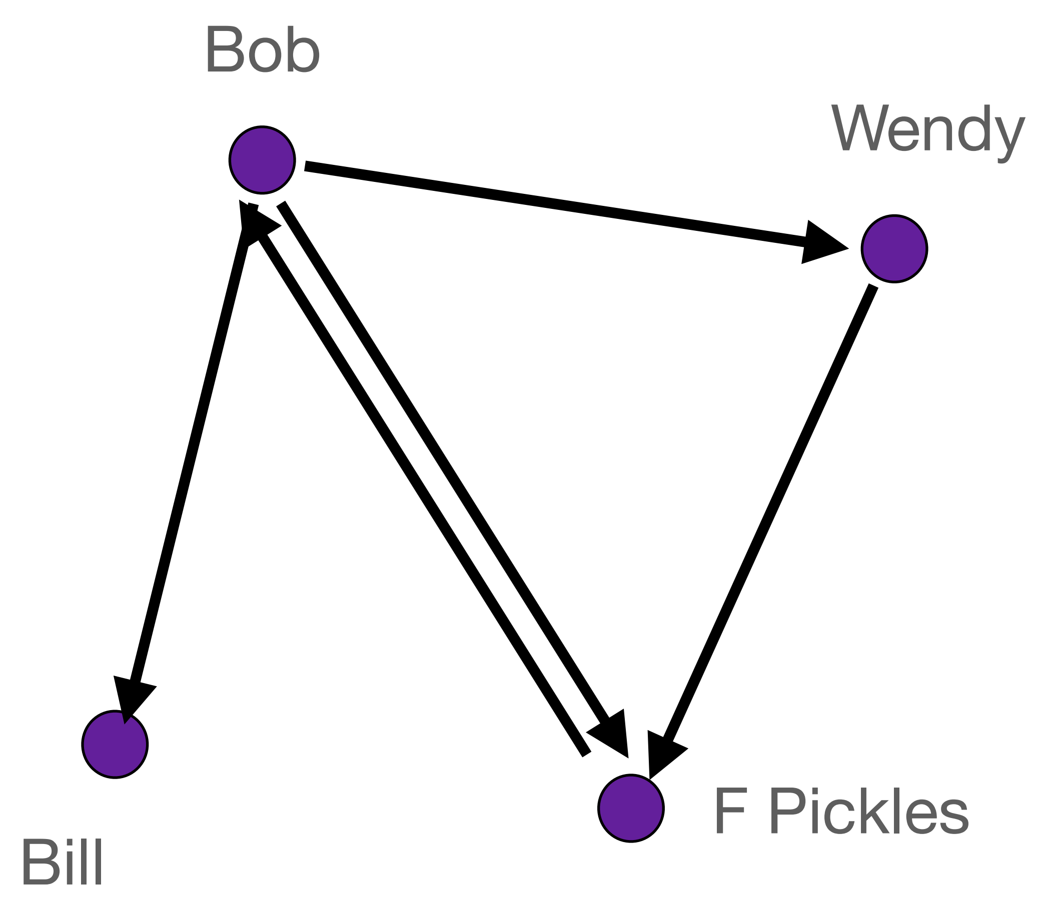
```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex:   <http://www.cs.man.ac.uk/> .

ex:bobthebuilder
        foaf:firstName "Bob";
        foaf:lastName "Builder";
        foaf:knows ex:wendy ;
        foaf:knows ex:farmerpickles;
        foaf:knows ex:billbibs.
ex:wendy
        foaf:firstName "wendy";
        foaf:knows ex:farmerpickles.

ex:farmerpickles
        foaf:firstName "Farmer";
        foaf:lastName "Pickles";
        foaf:knows ex:bobthebuilder.

ex:billbibs
        foaf:firstName "Bill";
        foaf:lastName "Bibby".
```



Example: Count Friends!

How many friends does Bob Builder have?

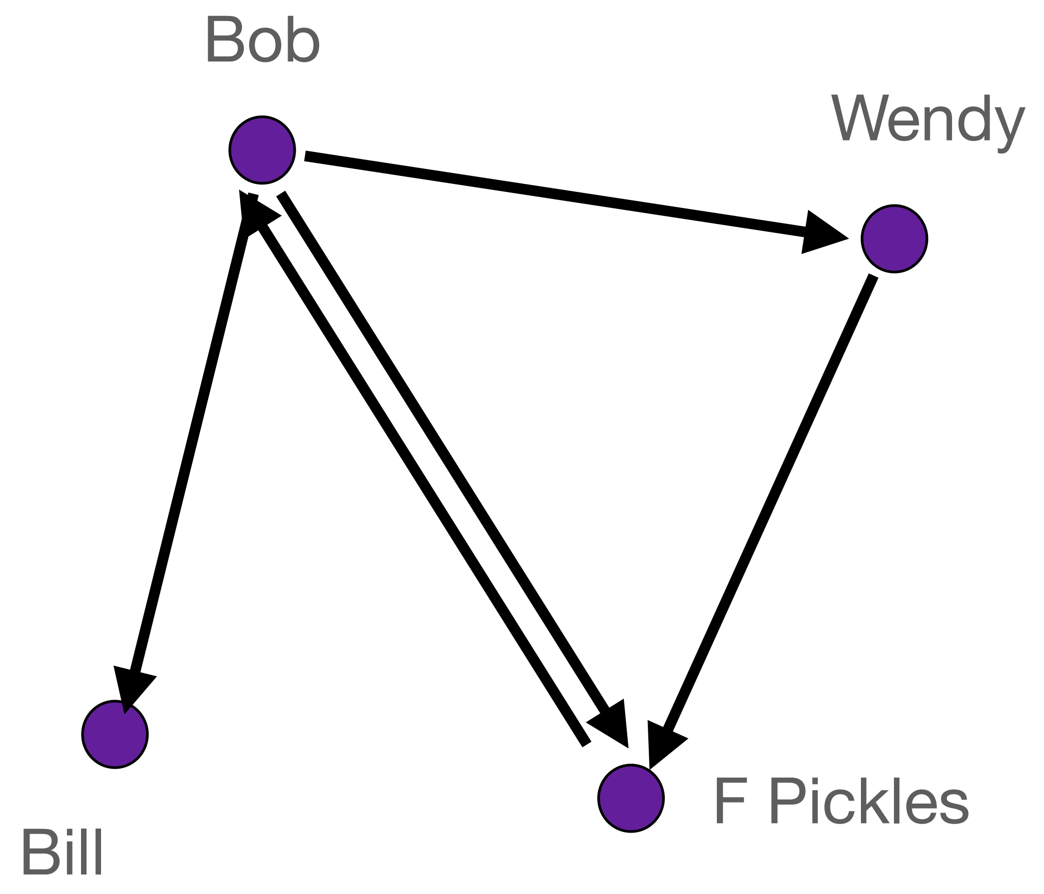
```
SELECT DISTINCT COUNT(?friend)
WHERE {ex:bobthebuilder
              foaf:firstName "Bob";
              foaf:lastName "Builder";
              foaf:knows ?friend };
```



Quite similar to a SQL query:

```
SELECT COUNT(DISTINCT k.Whom)
FROM Persons P, knows k
WHERE ( P.PersonID = k.Who AND
            P.FirstName = "Bob" AND
            P.LastName = "Builder" );
```

Example: Find Friends' Friends?



Give me Bob Builder's friends' friends' names?

```
SELECT ?first, ?last
WHERE {ex:bobthebuilder
                foaf:firstName "Bob";
                foaf:lastName "Builder";
                foaf:knows ?x.
          ?x foaf:knows ?y.
          ?y foaf:firstName ?first;
              foaf:lastName ?last}
```

As a SQL query:

```
SELECT P3.FirstName , P3.LastName
FROM knows k1, knows k2, Persons P1, Persons P3
WHERE (  k1.whom = k2.who AND
         P1.PersonID = k1.Who AND
         P3.PersonID = k2.Whom AND
         P1.FirstName = "Bob" AND
         P1.LastName = "Builder" );
```

Friends network?

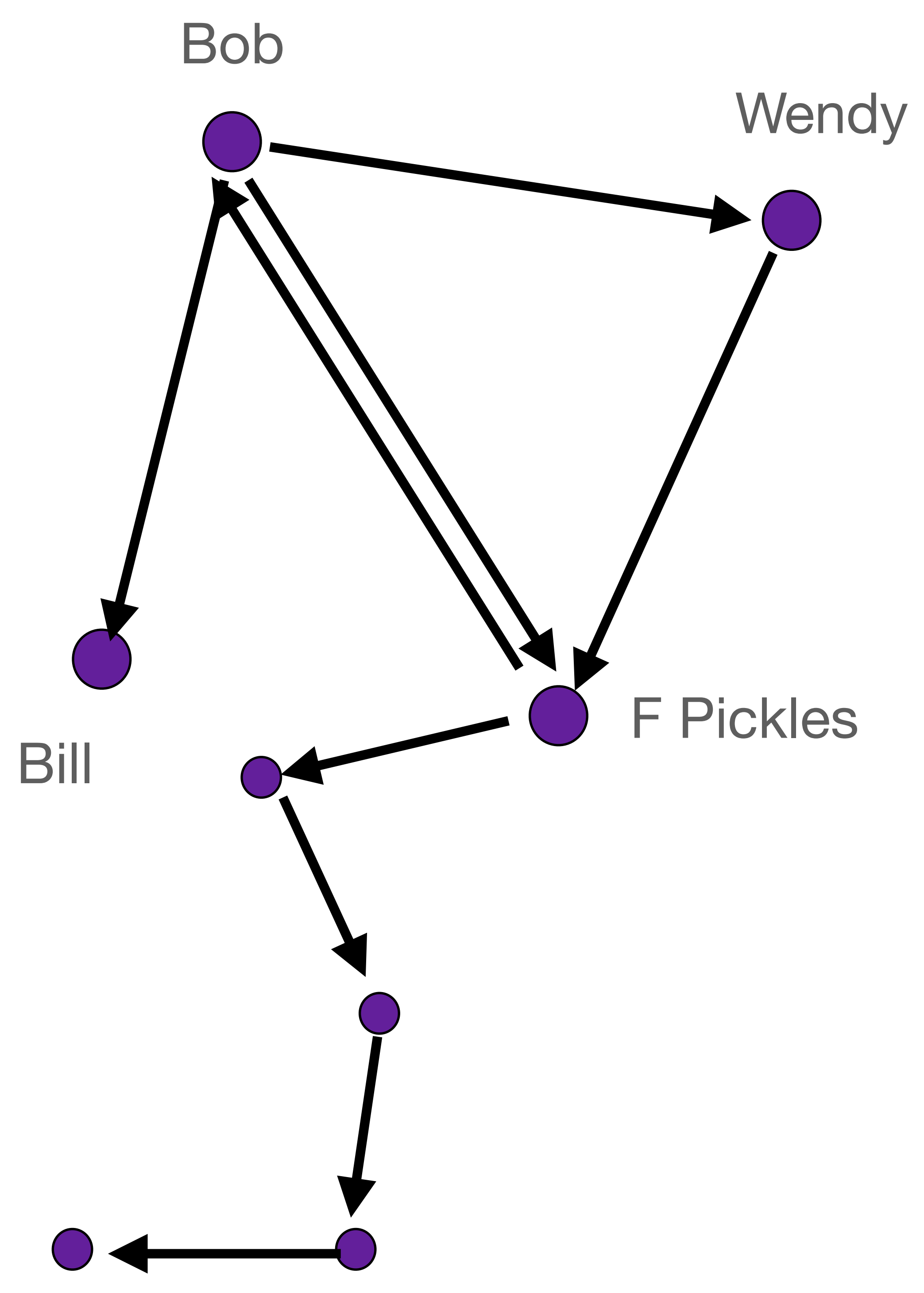
Give me everybody in Bob Builder's friends' friends…?

```
SELECT ?first, ?last
WHERE {ex:bobthebuilder
                 foaf:firstName "Bob";
                 foaf:lastName "Builder";
                 foaf:knows+ ?friend.
          ?friend foaf:firstName ?first;
                     foaf:lastName ?last}
```

a path query !!!!

works regardless of path length

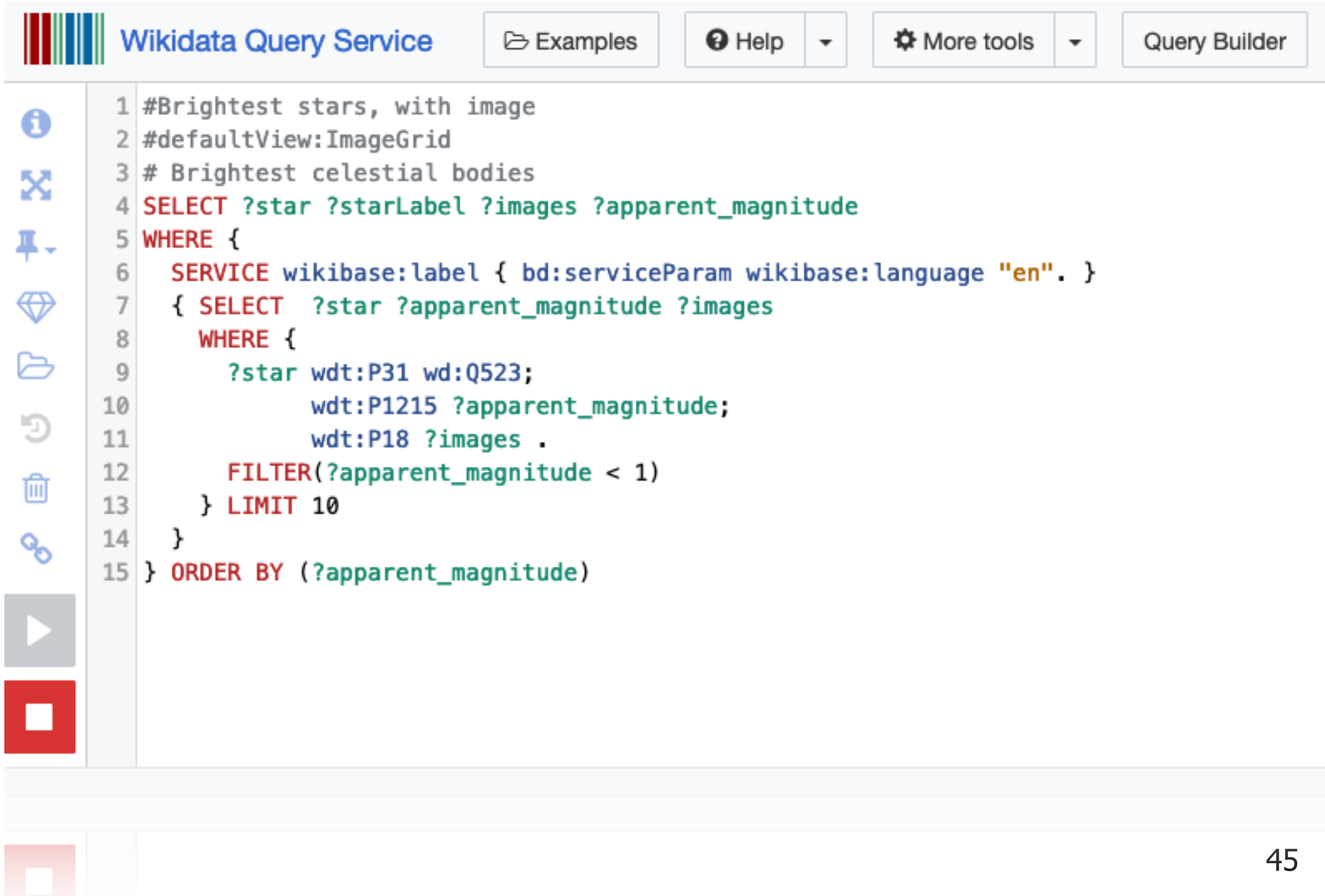
works without cycle detection



SPARQL supports
full regular expressions
in path queries!

Working with RDF graphs through SPARQL endpoints

- by hand
  - eg Wikidata
- programmatically
  - eg through REST APIs



Day 1: RDFS

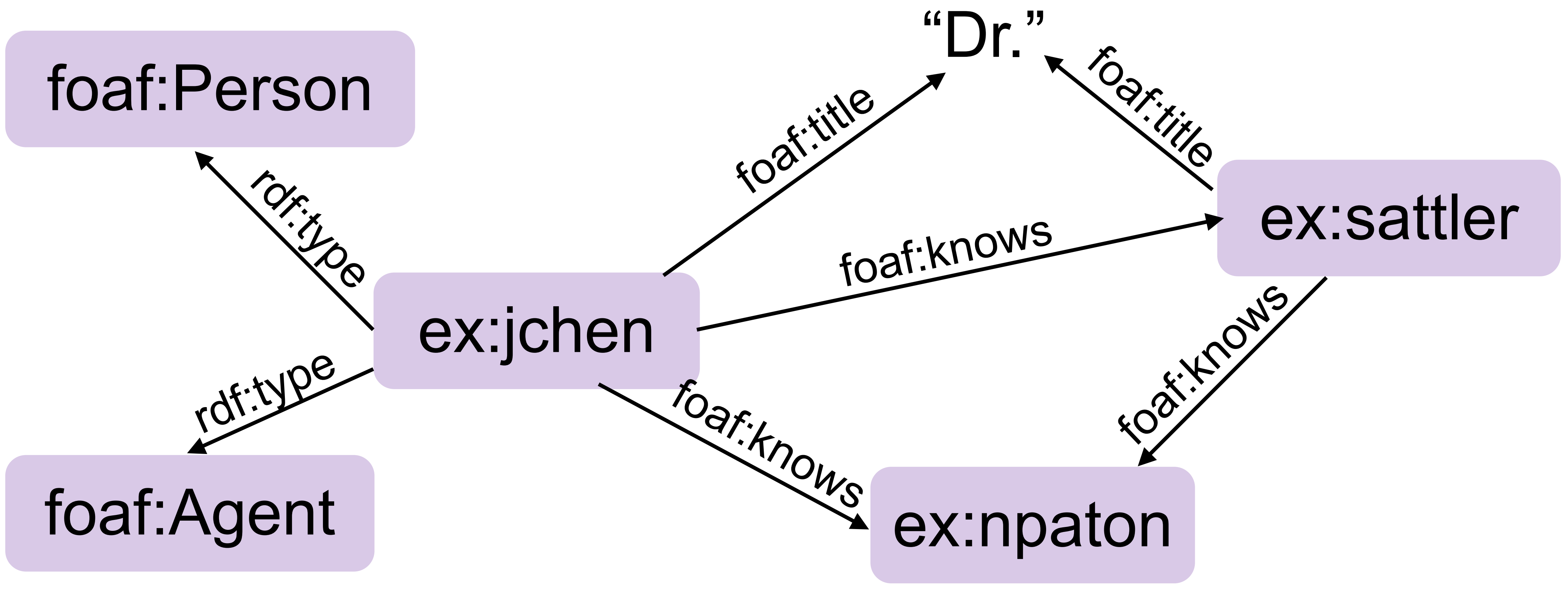
a schema language for RDF

Uli Sattler
Professor in Computer Science
University of Manchester

ESSAI 2024 Athens

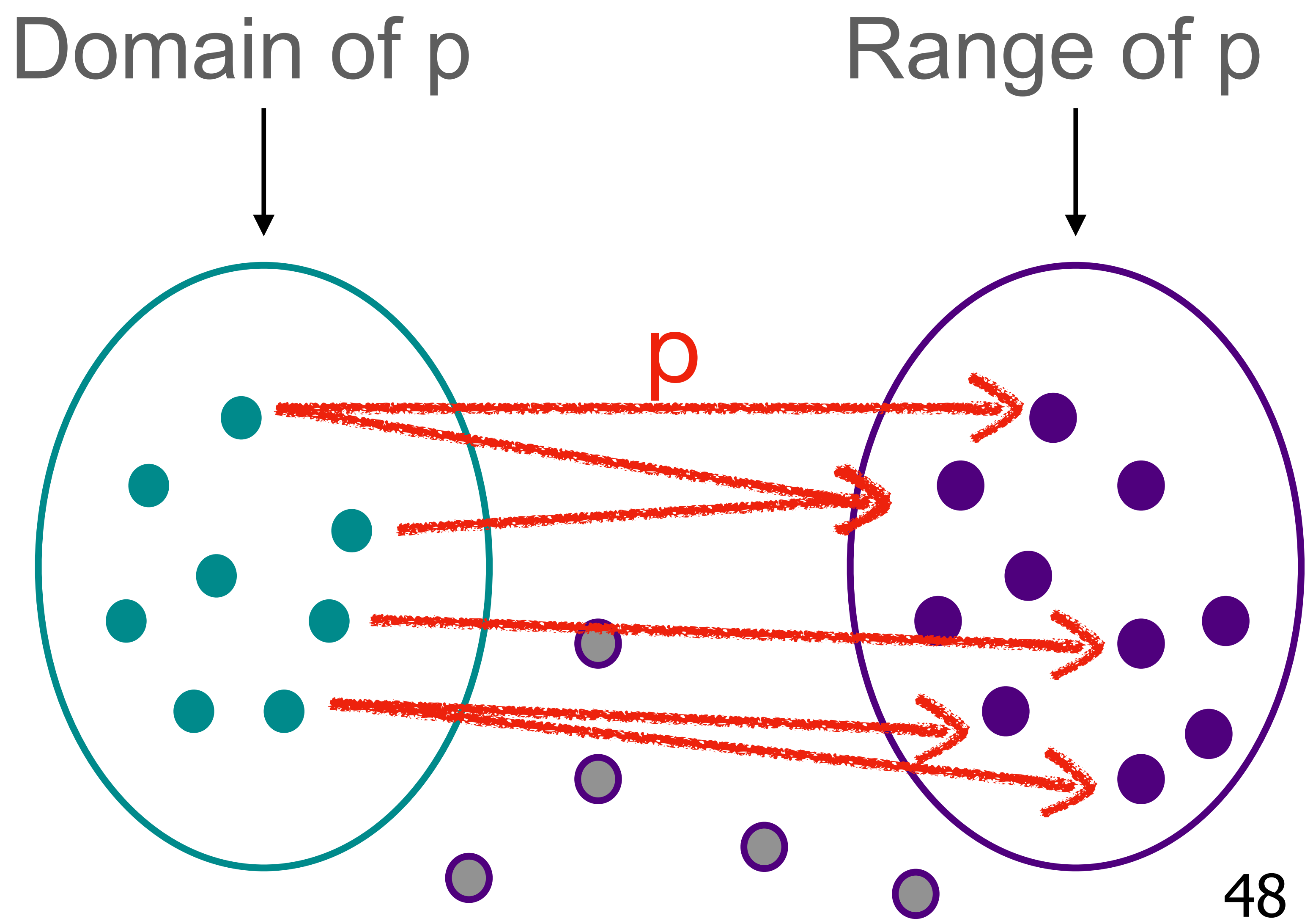
RDF

- in RDF, we can state *factual* knowledge:
  - how 2 nodes relate
    - e.g. (ex:sattler, foaf:knows, foaf:npaton)
  - how a node relates to a literal
    - e.g. (ex:jchen, foaf:title, “Dr.”)
  - what type a node has via rdf:**type**
    - e.g. (ex:sattler, rdf:type, foaf:Person)
- but we can’t say anything about
  - classes
    - e.g., foaf:Person implies foaf:Agent
  - properties
    - e.g., worksWith implies knows



RDFS: a schema language for RDF

- in RDFS, we can state *conceptual knowledge:*
  - rdfs:**subClassOf**
    - e.g. (foaf:Person, rdfs:subClassOf, foaf:Agent)
    - (ex:Woman, rdfs:subClassOf, foaf:Person)
  - rdfs:**subPropertyOf**
    - e.g. (ex:worksWith, rdfs:subPropertyOf, foaf:knows)
  - rdfs:**domain**
    - e.g. (ex:hasChild, rdfs:domain, foaf:Person)
(foaf:currentProject, rdfs:domain, foaf:Person)
  - rdfs:**range**
    - e.g. (ex:hasChild, rdfs:range, foaf:Person)
(foaf:currentProject, rdfs:range, foaf:Project)



Reasoning: Default Values++

- RDFS does **not** *describe/constrain structure*
  - that is, unlike in other schema languages,
    - in RDFS, we don't describe what *has to be the case*

      we don't write *integrity constraints*
    - RDFS can't be used to "validate" documents/graphs
- RDFS allows us to provide *extra information*
  - …a bit like default values!
  - …rather than requesting information, we infer it!

Reasoning: Default Values++

- RDFS does **not** *describe/constrain structure*
- RDFS allows us to provide *extra information*

Facts

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex:   <http://www.cs.man.ac.uk/> .

ex:sattler
  foaf:title  "Dr." ;
  foaf:knows ex:jiaoyanchen ;
  foaf:knows
   [
    foaf:title "Count";
    foaf:lastName "Dracula"
  ] .
```

\+

extra information

```
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:person rdfs:subClassOf foaf:Agent.
```

=>

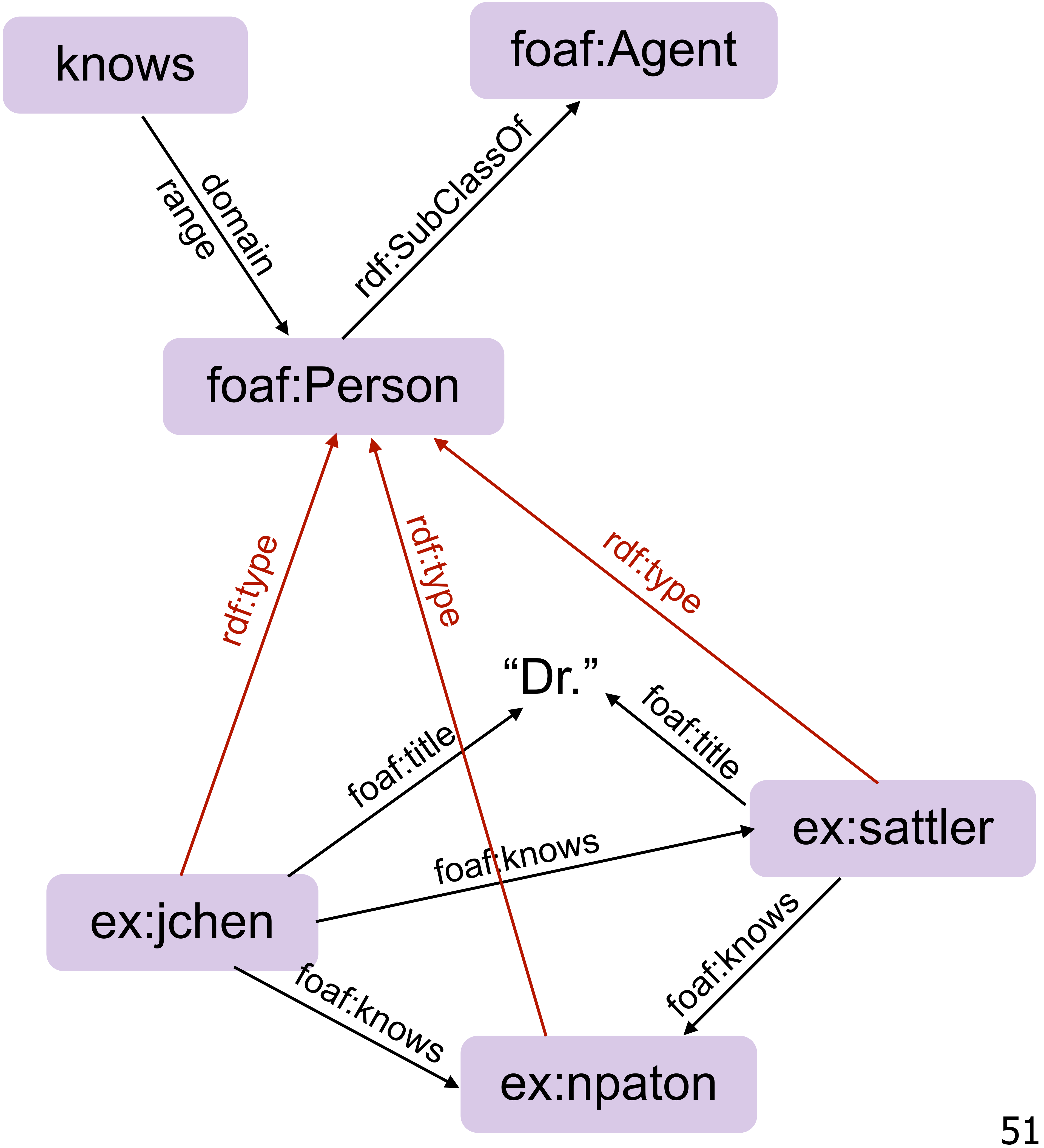
inferences

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex:   <http://www.cs.man.ac.uk/> .

ex:sattler rdf:type foaf:Person.
ex:sattler rdf:type foaf:Agent.
ex:jiaoyanchen rdf:type foaf:Person.
ex:jiaoyanchen rdf:type foaf:Agent.
```

Reasoning: Default Values++

- RDFS does **not** *describe/constrain structure*
- RDFS allows us to provide *extra information*

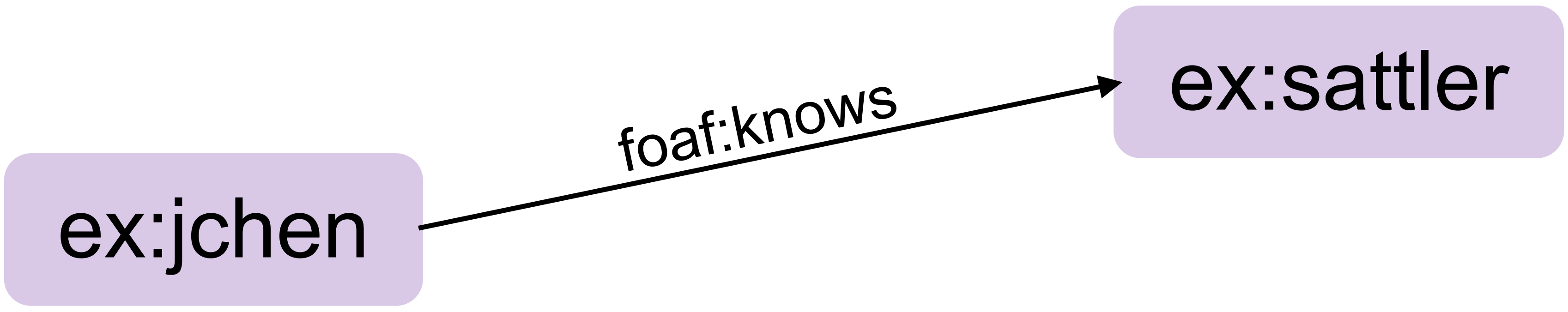
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:Person rdfs:subClassOf foaf:Agent



What do schemas usually do again?

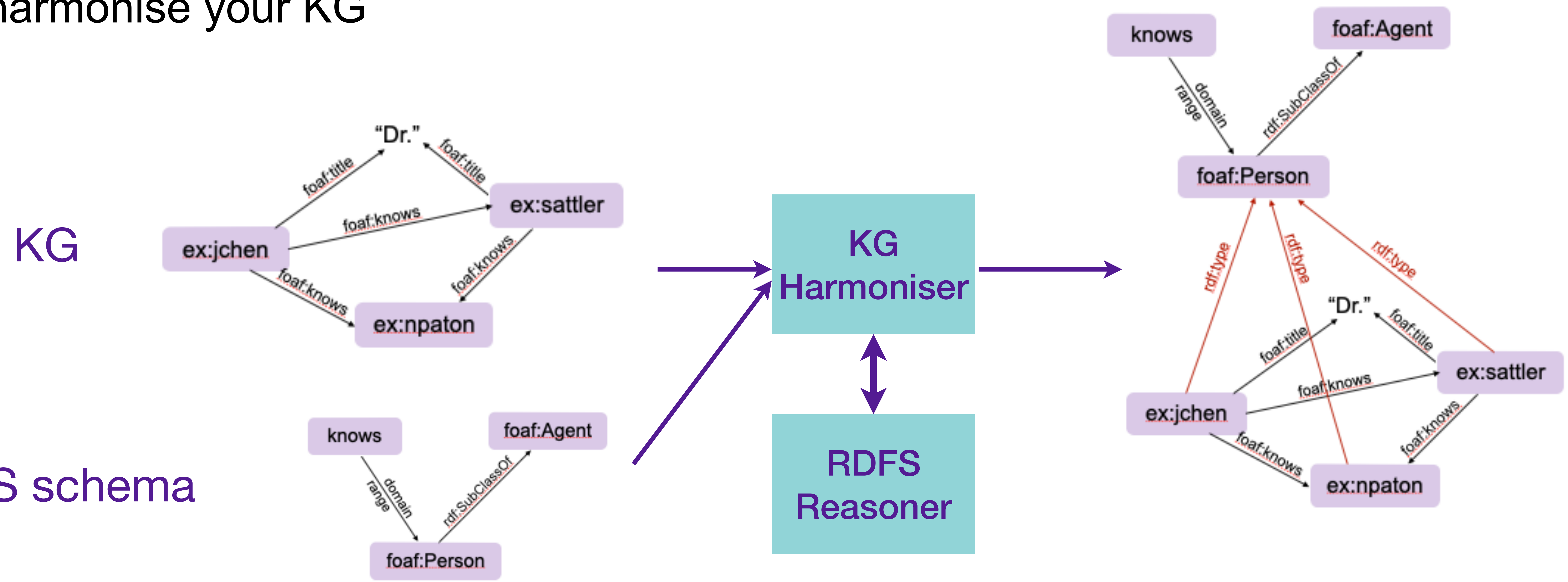
- In other schema languages, we usually **describe** ExtReps:
  - what’s allowed
  - what’s required
  - what’s assumed
    - default values
  - what’s expected
  - what’s forbidden (e.g., in Schematron)
- In RDFS, we can only state
  - what’s assumed/known, and thus
  - what can be inferred
    - here: ex:jchen rdf:type foaf:Person.
      ex:sattler rdf:type foaf:Person.

foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person



SPARQL, RDFS, and Reasoning

- Inferences can be *materialised*
  - add reasoning results to your KG
    - make background knowledge *explicit* in KG
    - harmonise your KG



SPARQL, RDFS, and Reasoning

- SPARQL queries are sensitive to RDF(S) inference
  - the way XPath is sensitive to default values!
  - also sensitive to more expressive language inferences
    - like OWL - tomorrow!
- Inference has a cost
  - results may be surprising
  - query answering may be (!) computationally expensive!

Solves all problems?

- No!
  - RDFS can’t express complex conceptual knowledge
    - see OWL tomorrow
  - we need to decide *which* additional information to make explicit in KG
    - too much: KG size may increase dramatically
    - too little: missing knowledge
- No validation!
  - this is a formalism specific quirk
  - there is SHACL

Day 1: SHACL

another schema language for RDF

Uli Sattler
Professor in Computer Science
University of Manchester

ESSAI 2024 Athens

SHACL: another schema language for RDF

- in SHACL, we have *shapes*
  - to describe constraints on nodes and edges:
  - eg in our KG, each Person has to have a first name or a last name

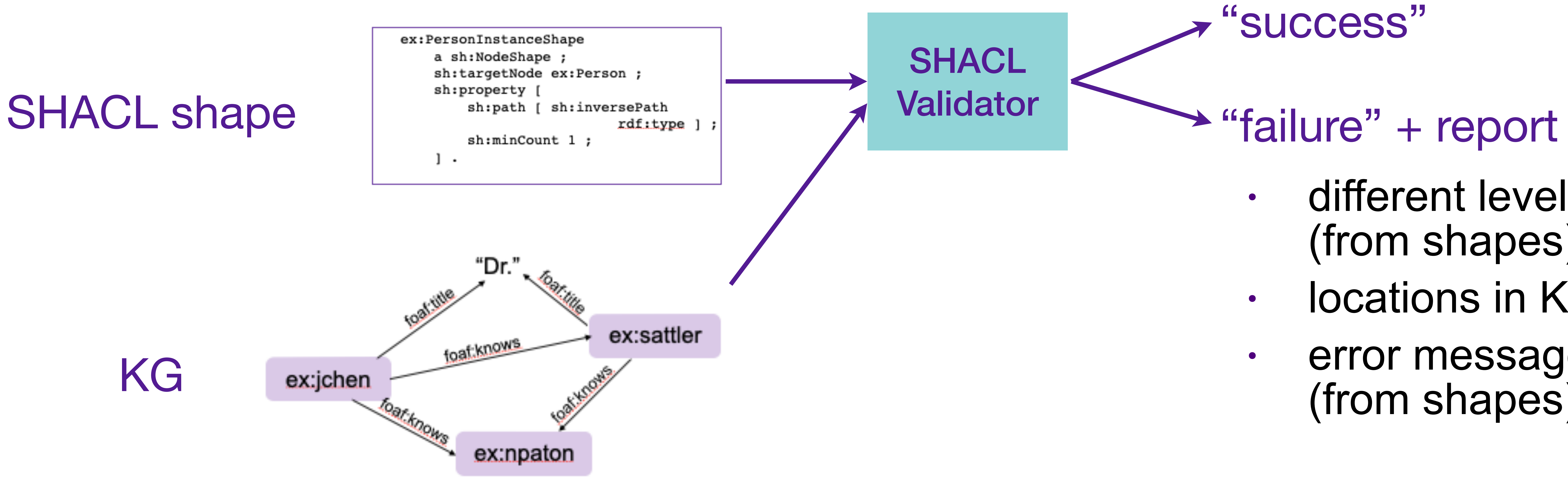
```
ex:PersonInstanceShape
    a sh:NodeShape ;
    sh:targetNode ex:Person ;
    sh:property [
        sh:path [ sh:alternativePath
                (foaf:firstName
                 foaf:lastName)] ;
        sh:minCount 1 ;
    ] .
```

- eg our KG must have at least 1 instance of Person

```
ex:PersonInstanceShape
    a sh:NodeShape ;
    sh:targetNode ex:Person ;
    sh:property [
        sh:path [ sh:inversePath
                          rdf:type ] ;
        sh:minCount 1 ;
    ] .
```

SHACL: validation

- given a KG and shapes, we can ask a *validator* to test whether KG satisfies constraints in shapes:



- different levels of severity (from shapes)
- locations in KG
- error messages (from shapes)

Day 1: RDFS & SHACL

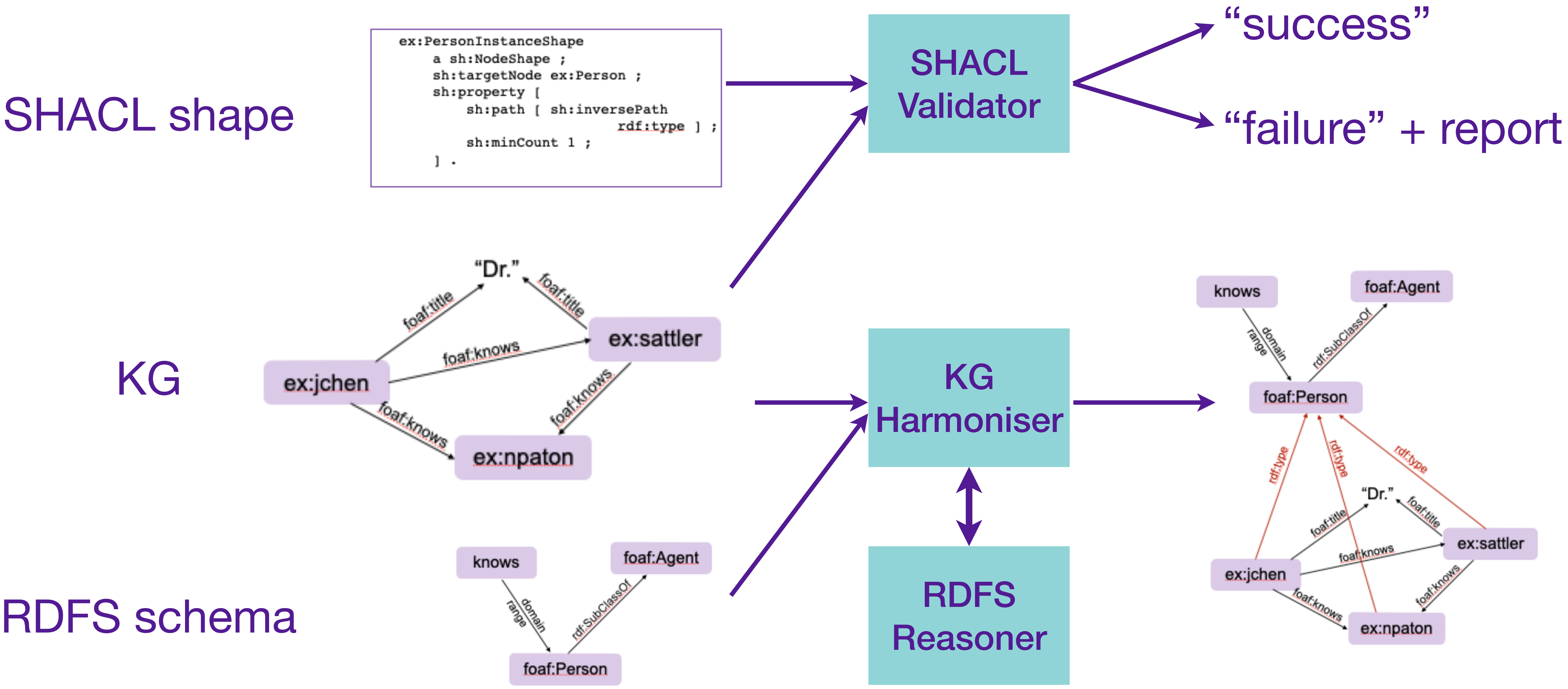
an interesting relation

Uli Sattler
Professor in Computer Science
University of Manchester

ESSAI 2024 Athens

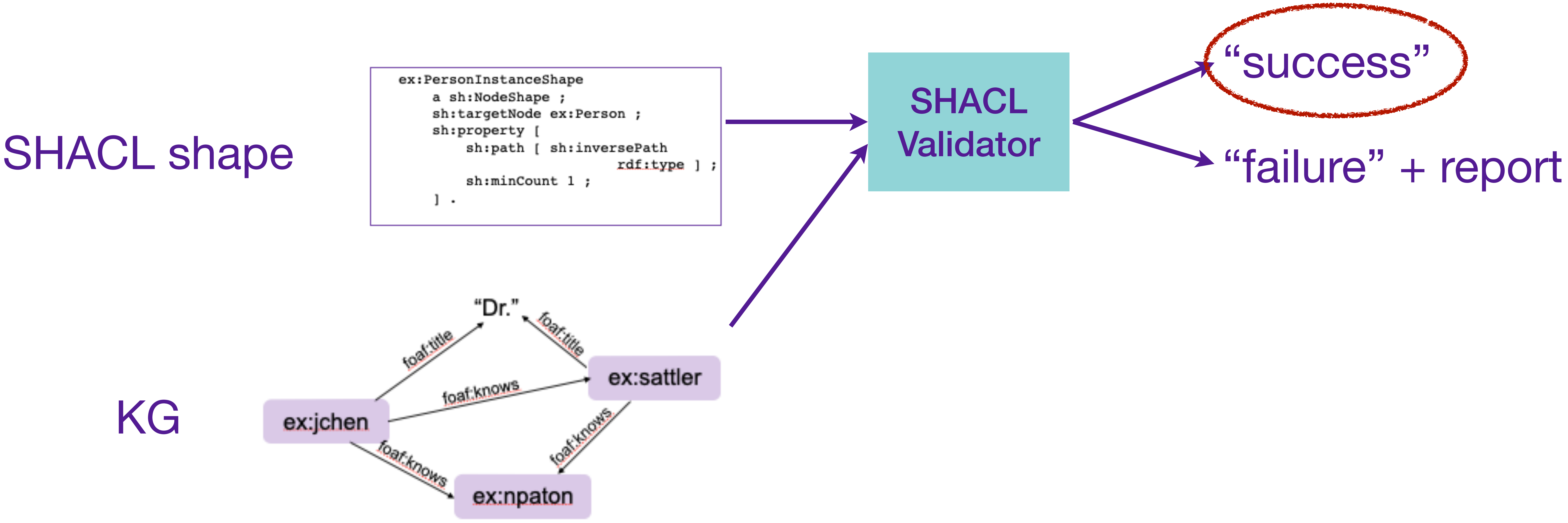
Validation and Reasoning…

- can affect each other:



Validation and Reasoning…

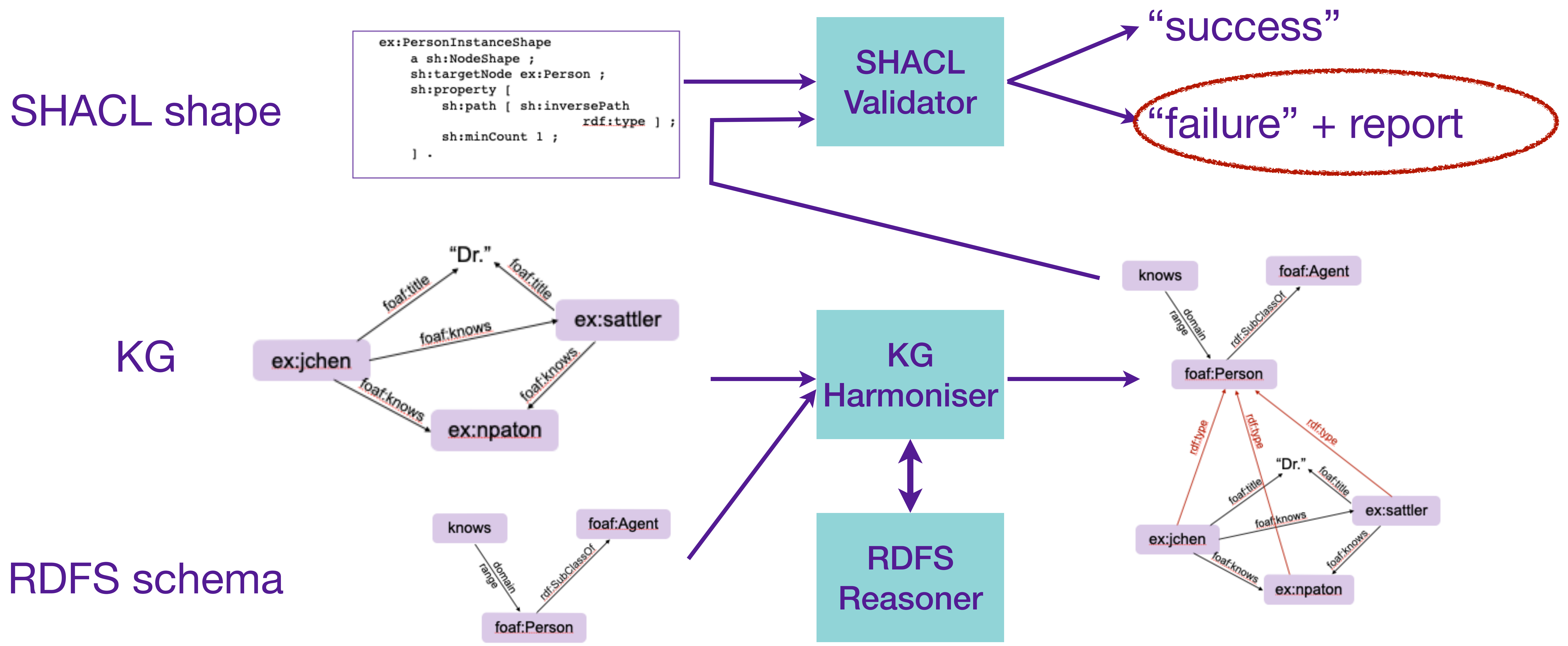
- can affect each other: it may be that



Validation and Reasoning…

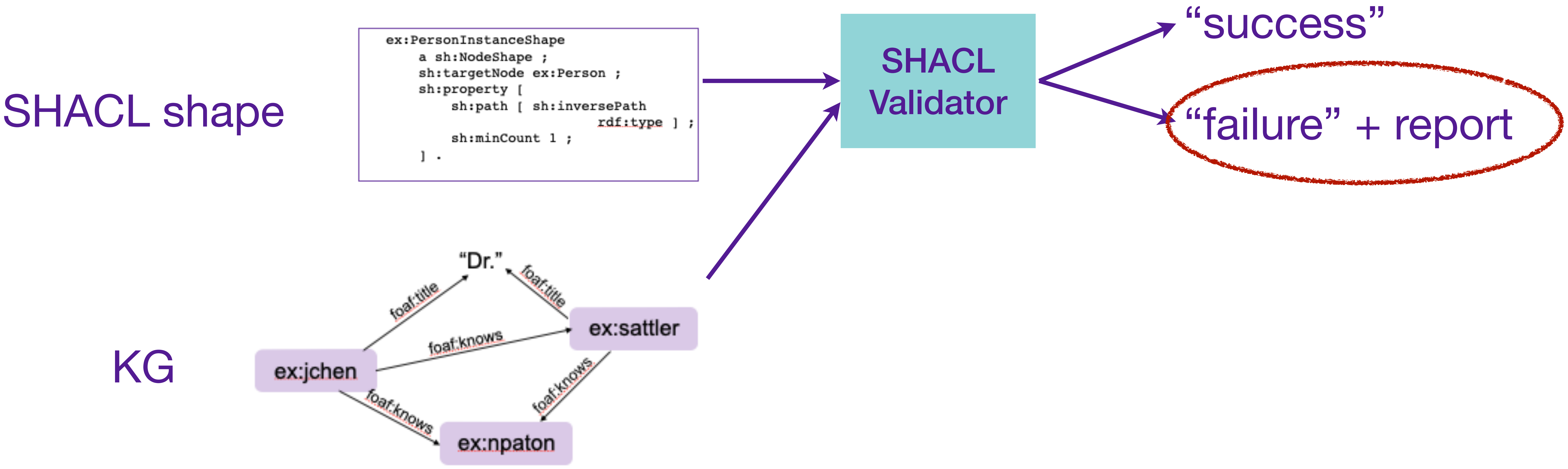
“each Person must have a name”

- can affect each other: it may be that but



Validation and Reasoning…

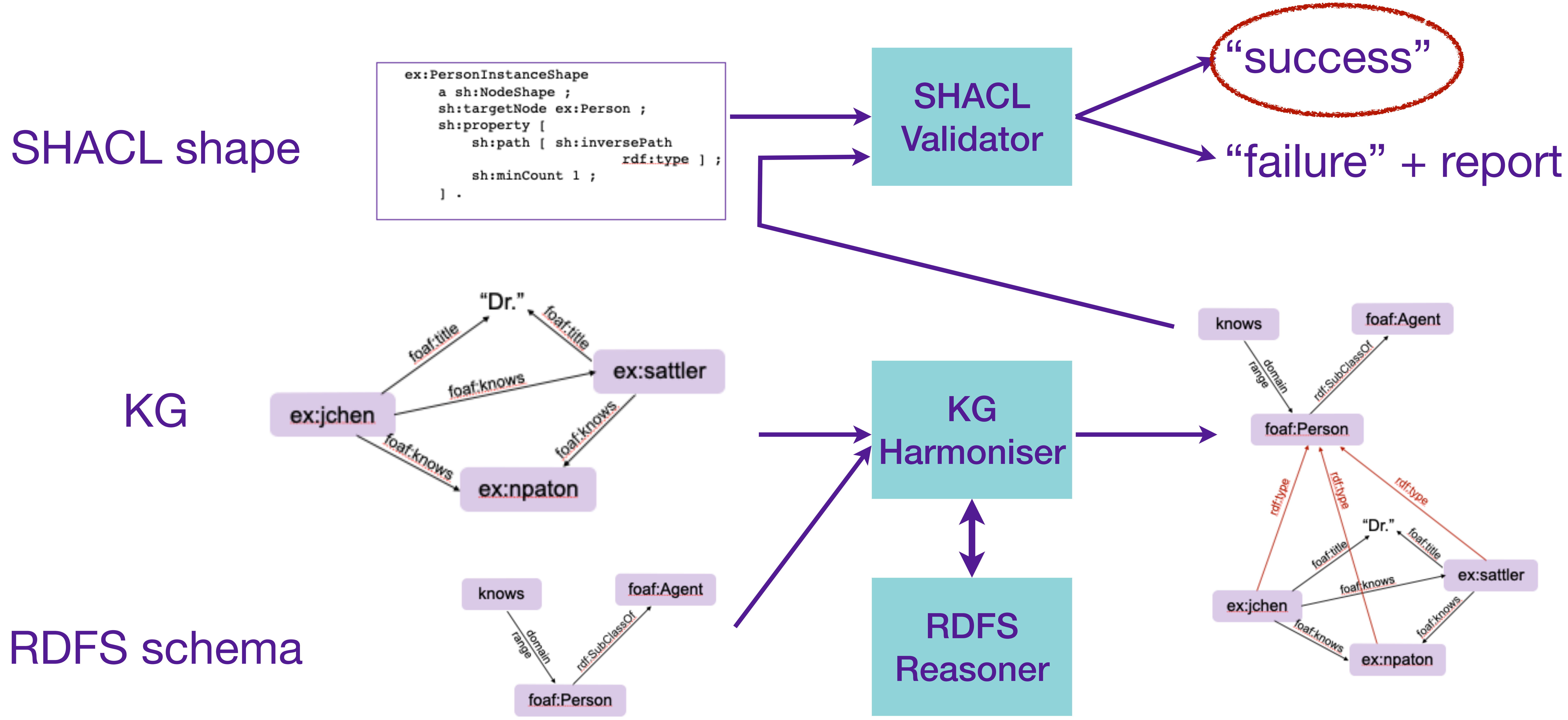
- can affect each other: it may also be that



Validation and Reasoning…

"our KG must have at least 1 instance of Person"

- can affect each other: it may be that but



Summary of today

- KGs can contain *factual* and *conceptual knowledge*
  - eg in RDF and RDFS
- *Reasoning* makes *implicit* knowledge *explicit*
- *Materialisation* of reasoning results can
  - *harmonise* a factual knowledge graph
  - prevent us from validating *invalid* documents
  - ensure that *implicitly* valid documents are validated

Any questions?

The End of Today’s session

Tomorrow: more on reasoning & OWL

Polyglott persistence

- How can we **vary?**
  - Same core data model, same implementation
    - but different domain models
  - Same core data model, same domain model
    - different implementations, e.g., SQLite vs. MySQL
  - Same shape of core data model, same conceptual model
    - different formalisms!
      - Usually, but not always, implies different implementations
      - e.g. JSON and XML
- We can be **explicitly** or **implicitly** poly-
  - If we **encode** another data model into our home model
    - We are still poly-
    - But only implicitly so
    - Key Cost: Ad hoc implementation
  - If we split our domain model across multiple formalisms/implementations

Key point

- Understand your **domain**
  - What are you trying to represent and manipulate
- Understand your use case
  - including (frequent, relevant) queries, error sources,…
- Understand the **fit** between domain and data model(s)
  - To see where there are sufficiently good fits
- Understand your infrastructure

Question 1

Consider again the Conceptual Model you started to work on last week: can you

- finish/improve/extend it?
- add adjectives?
- add examples?

- – format
- – formalism
- – core data model
- – data model
- – database
- – external repr.
- – …

- – domain model
- – schema
- – schema language
- – application
- – system
- – internal repr.
- – …

- – robust
- – extensible
- – scalable
- – self-describing
- – valid
- – expressive
- – verbose
- – …

Question 2

Consider a format for a reporting system for health & safety incidents, as exemplified by the printed example document:

- sketch a system for
  - gathering this data
  - reporting it monthly
- which kind of schema(s) would you use to describe it?
  - why?
- does this format make good use of XML's

Title Text

Good Bye!

- We hope you have learned a lot!
- It was a pleasure to work with you!
- Speak to us about projects
  - taster/MRes
  - MSc
- Enjoy the rest of your programme
  - COMP62421 query processing
  - COMP62342 rich modelling, inference
    semantic web, symbolic AI