UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB DISTRIBUTED DATA INTELLIGENCE AND TECHNOLOGIES LAB

https://www.snap4City.org

NeuroSymbolic Artificial Intelligence at Scale

Marco Fanfani, marco.fanfani@unifi.it

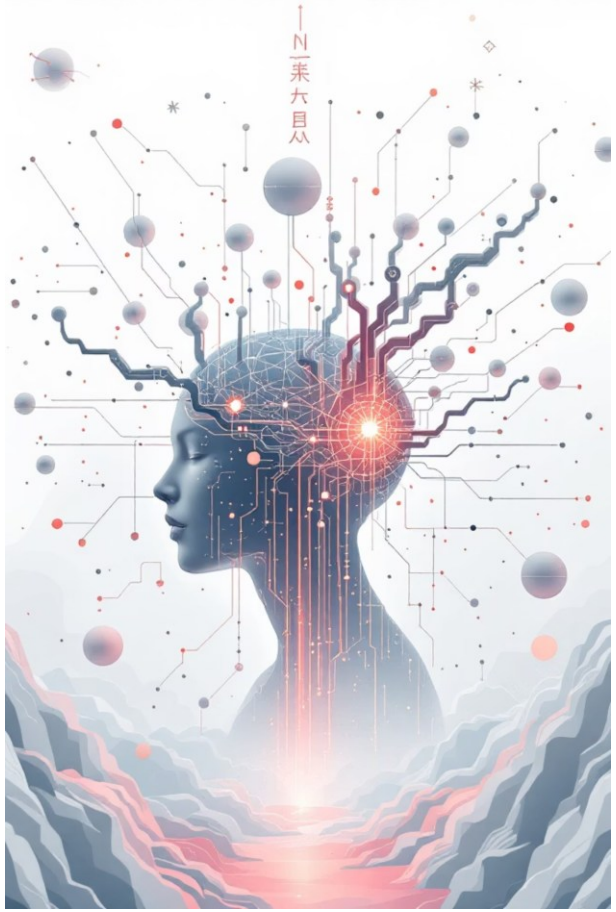
https://www.disit.org/

Parte: P6.0 (2025-26) - (Quick) Introduction to Knowledge Engineering

(Quick) Introduction to Knowledge Engineering

Knowledge Engineering and Semantic Modeling

- This lecture serves as the introduction to the neuro-symbolic integration of **ontologies and knowledge graphs** with modern **neural architectures.**
- The primary objective of this session is to establish the **symbolic foundations** required to build hybrid AI systems that combine the **perception of neural networks** with the **formal logic of symbolic engineering**
- We will introduce hot to transition from raw, unstructured data toward **structured, machine-interpretable knowledge bases** that act as the "symbolic brain" for advanced artificial intelligence



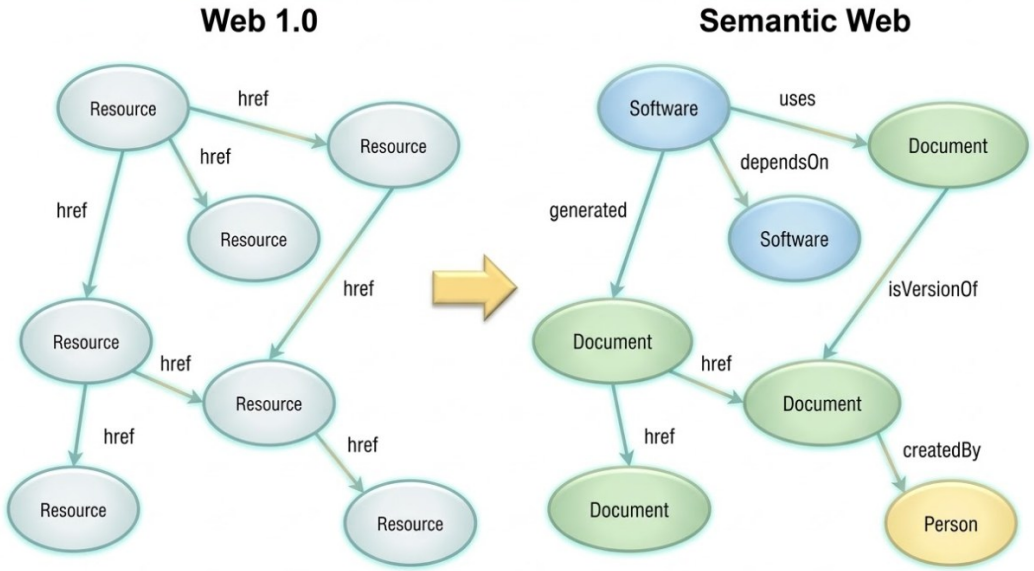
Overview and Objectives

This introduction is organized into three main pillars:

- **Formal modelling**: defining the core components of ontologies and the methodology of Knowledge Engineering used to model real-world domains
- **Technical implementation**: presenting the Semantic Web Stack, focusing on standards such as RDF for data interchange and OWL for rich semantic descriptions
- **Principles of automated reasoning**: the mechanics of deductive reasoning and identify the structural limitations of purely symbolic systems, which necessitates the integration of neural techniques

From Documents to Data

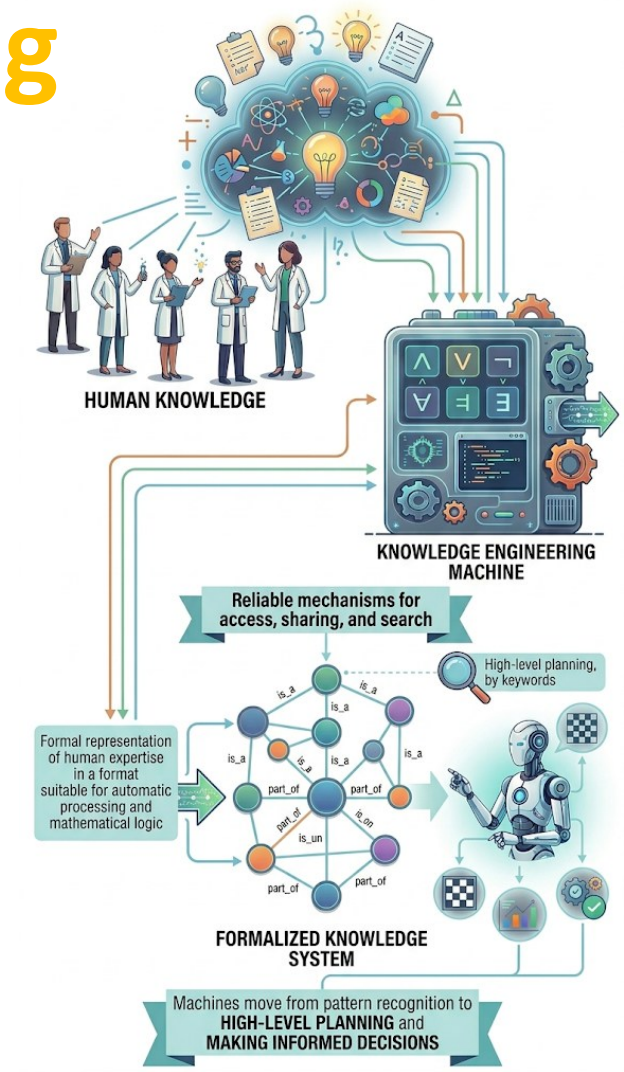
- The traditional Web is primarily a "Web of Documents" composed of HTML pages designed for human consumption, where machines often perceive resources as **unstructured text without deep context**.
- Knowledge Engineering facilitates the transition toward a **Web of Data** or **Linked Data**, where information is structured to be machine-friendly and **interoperable** across different systems.



- In this paradigm, every **real-world entity** is identified by a unique URL and accessed via standard protocols like HTTP, allowing for the creation of a global, interconnected **database of knowledge**

Knowledge Engineering

- Knowledge Engineering is a field dedicated to the **formal representation of human expertise and domain information** in a format that supports automated processing and mathematical logic
- The goal is to provide reliable mechanisms for accessing, sharing, and searching for knowledge beyond simple keyword matching
- By formalizing a domain, we enable machines to move from simple pattern recognition to **high-level planning and informed decision-making**



Knowledge Engineering

- Formalizing a domain does not only mean writing rules by hand but creating **Ontologies** and **Knowledge Graphs** that act as a **semantic infrastructure**.
- These tools make it possible to organize knowledge in such a way that the machine does not have to "guess" the relationships, but can consult them as a **structured map**

Ontology

- An ontology is defined as a **formal**, **structured**, and **systematic** representation of knowledge within a specific domain of interest
- It serves as a **blueprint** that specifies
  - the types of **entities** that exist,
  - the **properties** they possess, and
  - the rules governing their **interactions**
- Unlike simple classifications, ontologies provide a clear framework for organizing domain knowledge, facilitating a shared understanding between human experts and software agents

Ontology

- Developing an ontology is essential for **sharing a common understanding** of information structures among people or autonomous software agents
- Ontologies promote **knowledge reuse**, allowing developers to extend existing domain models (like Schema.org) rather than building from scratch for every new application

Ontology

- Furthermore, they make **domain assumptions explicit** and allow for the **separation of domain-specific knowledge from the general operational logic** of the software system

Making Domain Assumptions Explicit

- In traditional software development, the "rules of the world" are often hidden within **thousands of lines of if-else statements** or database constraints.
- Ontologies transform these hidden assumptions into **Axioms**—foundational statements of truth that are clear, unambiguous, and readable by both humans and machines

Separation of Domain Knowledge from Operational Logic

- This concept refers to a "bipartite" architecture where the what (the knowledge) is decoupled from the how (the software engine that processes it)

Ontologies versus Databases

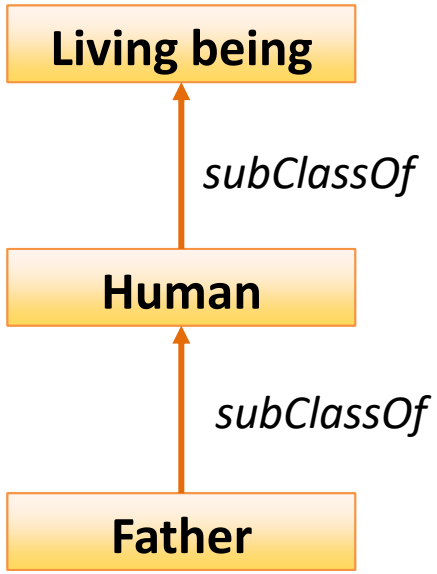
- An ontology represents an evolutionary step beyond a traditional database by adding a **layer of rich vocabulary and formal semantics** to facts
- While standard databases focus primarily on the structure and storage of data, **ontologies capture the inherent meaning and nuances of relationships**

Example:

- a database might simply link *Italy* to *Southern Europe*, and miss that Italy is in Europe,
- an ontology can formalize that *Southern Europe is part of Europe* allowing for deductive reasoning (or inference)

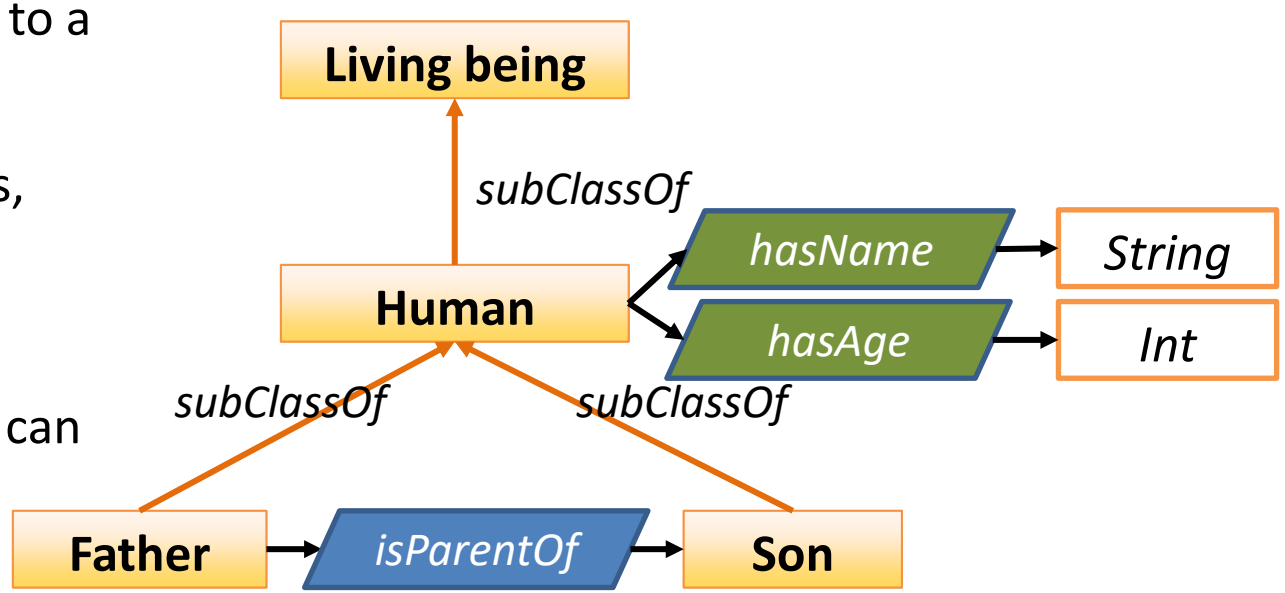
Classes and Taxonomies

- **Classes**, also known as Concepts, represent the general categories or types of entities found within a domain, such as "*Person*," "*City*," or "*Disease*"
- These concepts are often organized into **hierarchical taxonomies** using the "*subClassOf*" relationship, which allows for a structured classification of knowledge
- A critical feature of this structure is **inheritance**: subclasses automatically **inherit all logical restrictions and properties** from their parent classes in the hierarchy
- In the formal architecture of a Knowledge Base, this layer is designated as the **TBox** (**Terminological Box**). The TBox serves as a **structural blueprint** or schema that defines the vocabulary of the domain and the logical rules that govern conceptual relationships



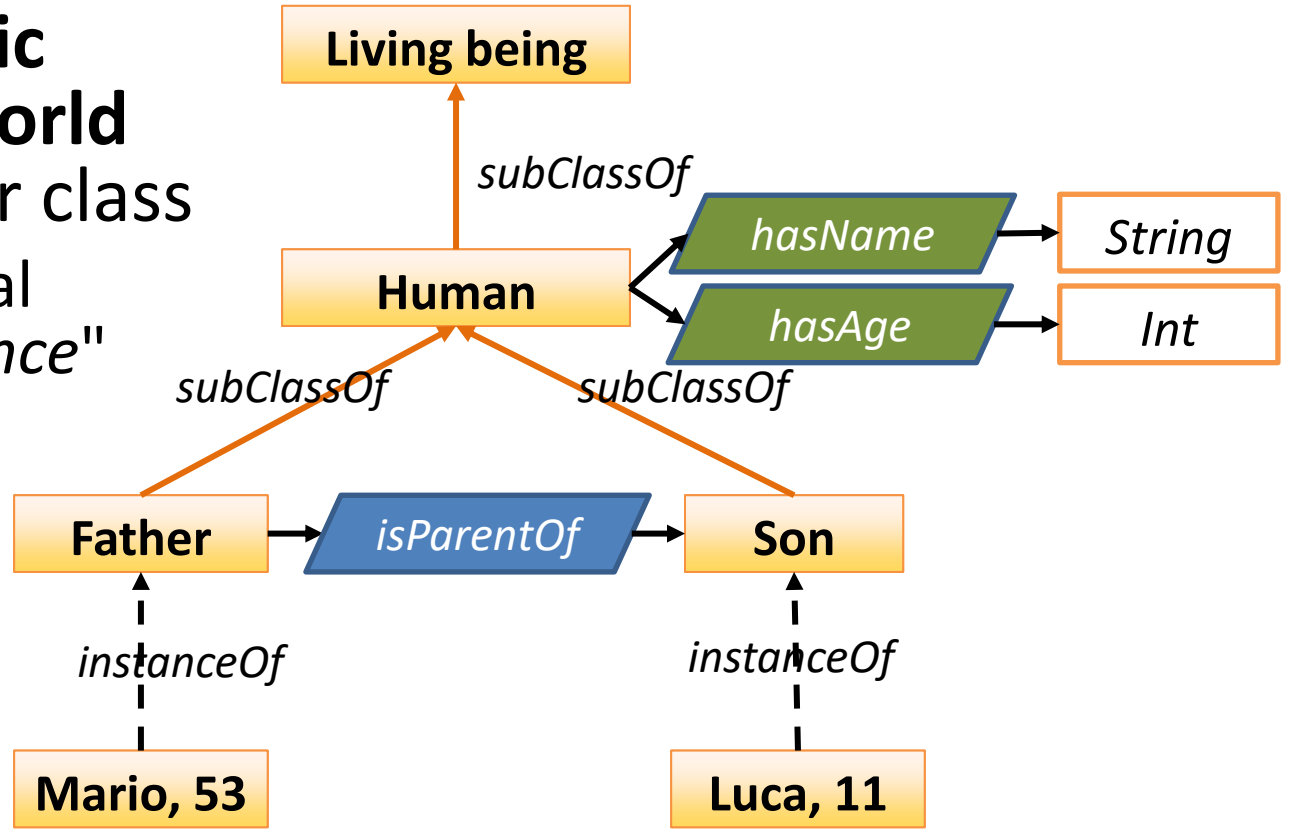
Properties

- **Properties** represent the predicates that **connects entities**
- Properties allow us to move beyond simple categorization to describe the **internal structure of concepts and the complex relationships** between individuals
- In the Semantic Web framework, properties are formally categorized into three distinct types:
  - **Object Properties**: These define binary relations between two individuals
  - **Datatype Properties**: These link an individual to a specific data value or "Literal,"
  - **Annotation Properties**: These are utilized to associate non-logical metadata, such as labels, comments, or versioning info, with resources
- Properties are further refined through logical axioms that constrain their use:
  - **Domain** specifies the class of individuals that can act as the subject of a property,
  - **Range** specifies the class of individuals or the data type that can act as the object



Instances & Facts

- Instances, also referred to as Individuals or Nominals, represent the **specific occurrences, members, or real-world entities** that belong to a particular class
  - For example, while "*City*" is a general conceptual class in the TBox, "*Florence*" is a specific individual instance
- This layer of specific, **assertional information** is known as the **ABox** (**Assertional Box**).
- The ABox describes specific facts and the relations between individuals



Development Cycle

- Ontology engineering is not a linear task but a **formal**, **iterative process** of modelling domain knowledge
- There is no single "correct" way to model a specific domain; instead, there are always **viable alternatives** depending on the intended application and anticipated future extensions
- The development cycle typically involves
  - defining classes,
  - arranging them in hierarchies,
  - specifying properties, and
  - filling in values for specific instances
- Because domains evolve and requirements change, the **ontology must be continuously refined through testing and expert feedback** to ensure it remains a faithful representation of reality

Formal Naming Conventions

- To ensure **interoperability** and **clarity**, an ontology must follow strict naming conventions.
- It is recommended to use English words and maintain consistency across the entire model.
- Common standards include the **use of CamelCase**,
  - class names start with an upper-case letter (e.g., "*PopulatedPlace*") and
  - properties start with a lower-case letter (e.g., "*hasPopulation*")
- Utilizing standard prefixes like "*has*" for object properties or "*is...Of*" for inverse relationships helps both human developers and automated reasoners interpret the intended semantics of the graph more accurately

Ontology Classification (1)

Foundational and Domain Models

- Ontologies are often classified by their level of generality. **Foundational** or Upper-Level ontologies provide very high-level, generic concepts applicable across many diverse domains
- **Domain** ontologies are more specialized, focusing on a specific area of interest—such as aviation, medicine, or smart cities—from a particular viewpoint
- By aligning domain ontologies with foundational structures, engineers can ensure that their data **remains interoperable** with other global datasets in the Linked Data cloud

Ontology Classification (2)

- Knowledge models also vary in their logical depth. Linguistic or terminological ontologies focus on **managing labels**, **taxonomies**, and **thesauri** for search and indexing purposes
- Formal ontologies, such as those written in **OWL** (Web Ontology Language), are grounded in **Description Logics**, i.e., formal knowledge representation languages
- Formal ontologies are designed for **automated inference** and **reasoning**, allowing systems to derive new conclusions that were never explicitly stated in the source data

Semantic Web Assumptions

- Anyone Can Say Anything (AAA)
  - The architecture of the Semantic Web is decentralized by design. The fundamental assumption is that "*Anyone can say Anything about Anything*" (AAA)
  - This means that information about a single resource can be **distributed across multiple independent sites**.
  - Because there is no central authority to manage the "truth," the system must be **flexible enough to integrate potentially conflicting or complementary statements** about the same real-world entity from different providers

Semantic Web Assumptions

- Open World Assumption (OWA)
  - A critical distinction between ontologies and traditional databases is the Open World Assumption (OWA).
  - In ontology, if a fact is not explicitly stated in the knowledge base, **it is considered "unknown" rather than "false"**
  - This is essential for neuro-symbolic systems: while neural models often operate in a "closed" environment where missing data might lead to "hallucinations," the symbolic layer uses OWA to **acknowledge gaps in knowledge**, preventing the system from making incorrect negative deductions

Semantic Web Assumptions

- Open World Assumption (OWA), example:
  - Consider a Knowledge Graph that contains the triple

    <*Mary*, *holds*, *Apple*>.

  - Under a standard database (Closed World), if you ask, "Does Mary hold a banana?", the system will answer "No" because that fact is missing.
  - Under the OWA, the system would answer "Unknown." A "No" or "False" answer is only provided if there is an explicit axiom stating that Mary is not holding a banana
  - This logic ensures that the system does not over-reach its conclusions when dealing with incomplete web data.

Semantic Web Assumptions

- Non-Unique Name Assumption (NUNA):
  - In a decentralized environment, different people will inevitably create different identifiers (URIs) for the same real-world object.
  - This leads to the "Non-Unique Name Assumption" (NUNA), which states that **different names (URIs) do not necessarily refer to different objects**
  - Unless two entities are explicitly declared as distinct, a reasoner must allow for the possibility that they are the same. This allows for **data integration across different sources**, where one site might refer to "Firenze" and another to "Florence"

Semantic Web Assumptions

- Non-Unique Name Assumption (NUNA):
  - To resolve the naming issues presented by NUNA, the Semantic Web uses identity **management properties**
  - The "*owl:sameAs*" predicate is a powerful tool that **explicitly tells the machine that two different URIs refer to the exact same individual**
  - Once this link is established, a reasoner can **merge the information from both sources**, providing a unified view of the entity
  - For example, linking the DBpedia URI for "*Florence*" with the GeoNames URI for the same city allows the system to combine population data from one with geographical coordinates from the other

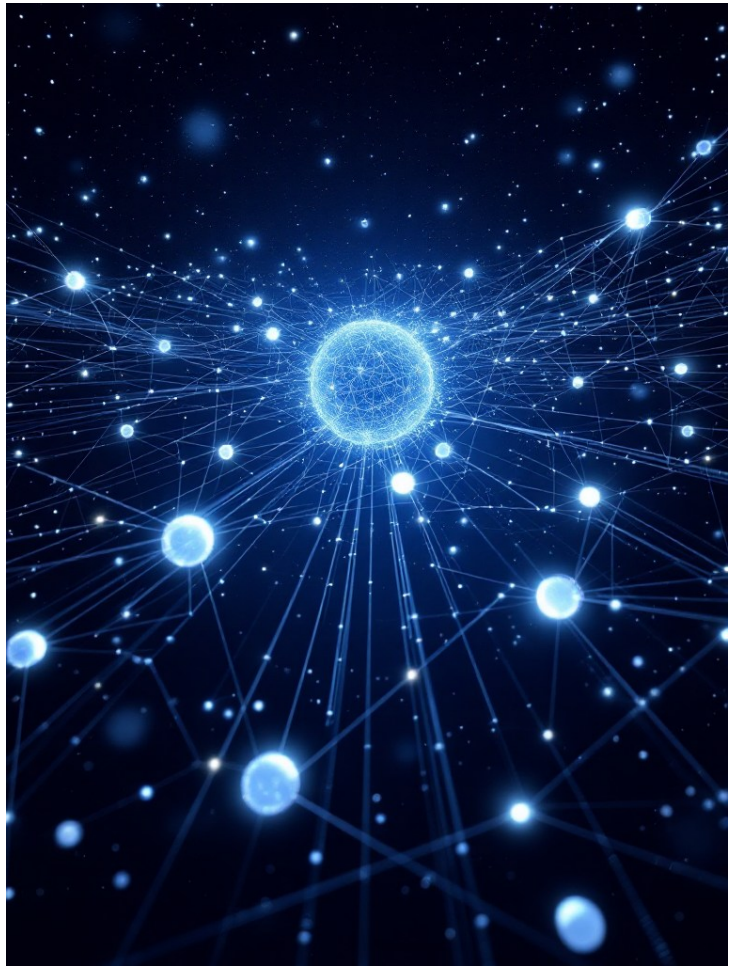
Semantic Web Assumptions

- Non-Unique Name Assumption (NUNA):
  - Conversely, it is often necessary to **explicitly state that two resources are not the same** to prevent a reasoner from incorrectly merging them during inference
  - The "*owl:differentFrom*" axiom is used to declare this distinction formally.
  - This is particularly useful in large-scale data integration where similar names or attributes might otherwise lead the system to assume identity where none exists.
  - Maintaining these distinctions is vital for the logical consistency of the knowledge base.

From Concepts to Implementation

- We have now established the conceptual and logical foundations of Knowledge Engineering.
- We have moved from defining basic components like **classes** and **properties** to understanding the complex logic of **OWA** and **NUNA** that allows ontologies to handle the ambiguity of the real world.
- The separation of concerns between the **TBox** (schema) and **ABox** (data) provides the structural blueprint necessary for the next part of this lecture
- Next, we will explore the **technical formats like RDF** and the **query languages like SPARQL** that turn these models into working Knowledge Graphs.

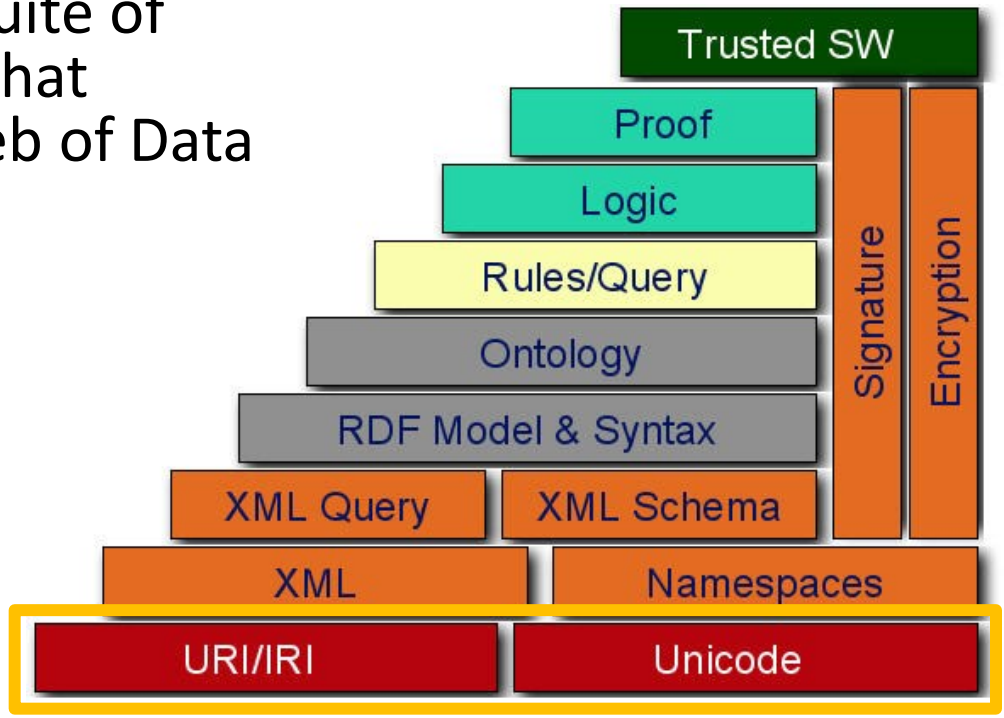
Introduction to Semantic Web Technologies



- The Semantic Web is not intended to be a separate entity from the existing Web, but rather a **functional extension of it**.
- The primary goal is to provide data with **well-defined meaning**, enabling humans and computers to work in closer cooperation, by **leveraging semantics-enriched data**,

Semantic Web Stack

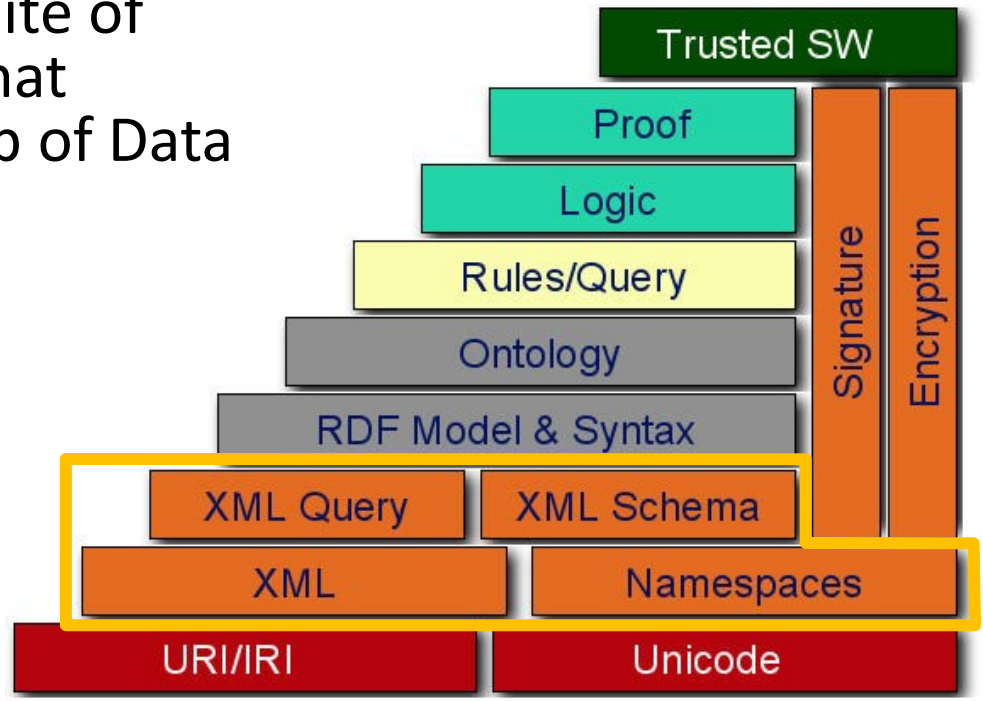
- The Semantic Web Stack represents a hierarchical suite of open, standardized W3C technology specifications that collectively define the infrastructure for a global Web of Data



At the foundational level, the stack relies on **URIs (Uniform Resource Identifiers)** and **Unicode** to provide a persistent, global mechanism for identifying real-world entities

Semantic Web Stack

- The Semantic Web Stack represents a hierarchical suite of open, standardized W3C technology specifications that collectively define the infrastructure for a global Web of Data

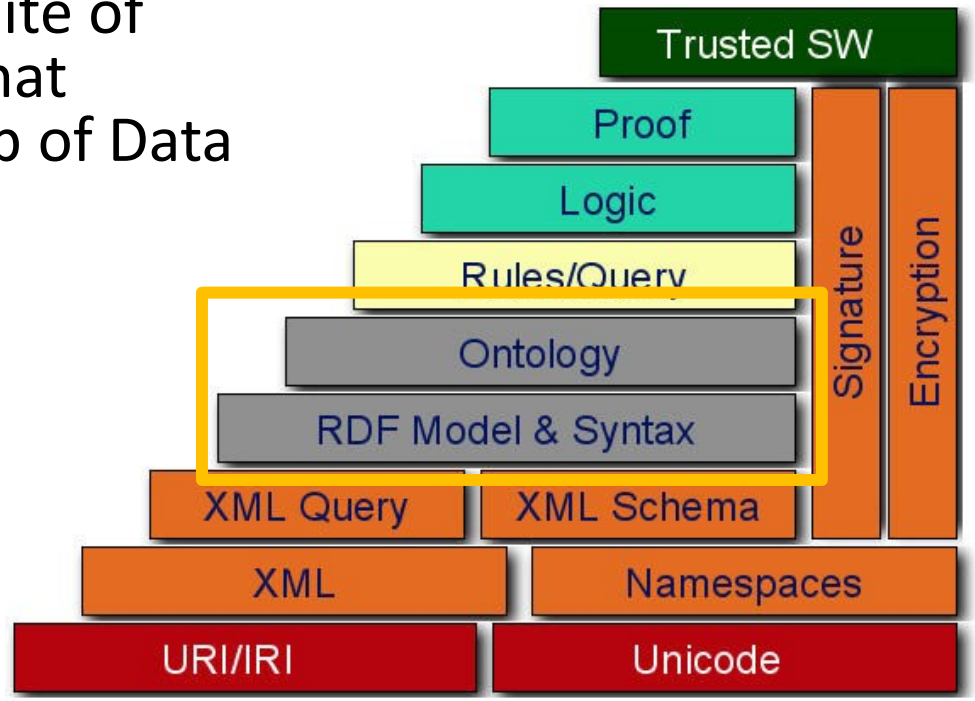


XML serves as a **common syntax** for document structure

At the foundational level, the stack relies on **URIs (Uniform Resource Identifiers)** and **Unicode** to provide a persistent, global mechanism for identifying real-world entities

Semantic Web Stack

- The Semantic Web Stack represents a hierarchical suite of open, standardized W3C technology specifications that collectively define the infrastructure for a global Web of Data



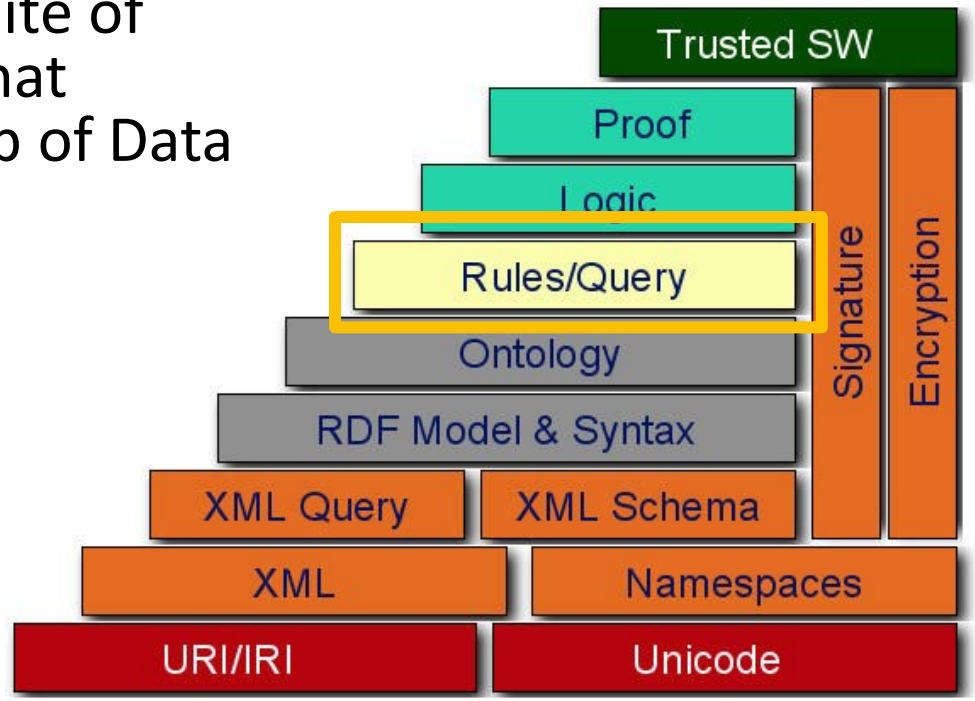
RDF Schema (**RDFS**) introduces **basic taxonomies** and **class hierarchies**, while the Web Ontology Language (**OWL**) provides a rich, logic-based **knowledge representation language**

XML serves as a **common syntax** for document structure

At the foundational level, the stack relies on **URIs (Uniform Resource Identifiers)** and **Unicode** to provide a persistent, global mechanism for identifying real-world entities

Semantic Web Stack

- The Semantic Web Stack represents a hierarchical suite of open, standardized W3C technology specifications that collectively define the infrastructure for a global Web of Data



To **access** and **manipulate** this structured knowledge, **SPARQL** serves as the standardized semantic query language

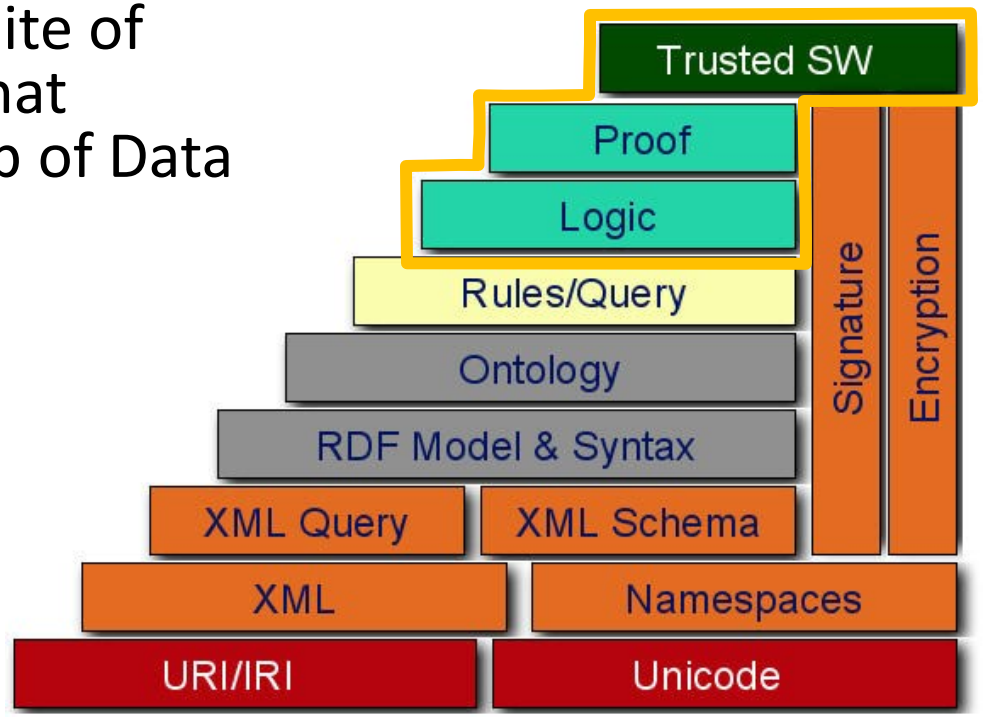
RDF Schema (**RDFS**) introduces **basic taxonomies** and **class hierarchies**, while the Web Ontology Language (**OWL**) provides a rich, logic-based **knowledge representation language**

XML serves as a **common syntax** for document structure

At the foundational level, the stack relies on **URIs (Uniform Resource Identifiers)** and **Unicode** to provide a persistent, global mechanism for identifying real-world entities

Semantic Web Stack

- The Semantic Web Stack represents a hierarchical suite of open, standardized W3C technology specifications that collectively define the infrastructure for a global Web of Data



Unifying Logic, Proof, and Trust--represent the **ongoing research**

To **access** and **manipulate** this structured knowledge, **SPARQL** serves as the standardized semantic query language

RDF Schema (**RDFS**) introduces **basic taxonomies** and **class hierarchies**, while the Web Ontology Language (**OWL**) provides a rich, logic-based **knowledge representation language**

XML serves as a **common syntax** for document structure

At the foundational level, the stack relies on **URIs (Uniform Resource Identifiers)** and **Unicode** to provide a persistent, global mechanism for identifying real-world entities

URIs and URLs

- In the Semantic Web architecture, every unique entity, concept, or relationship **must be identified** by a global and persistent identifier.
- The standard mechanism for this is the **Uniform Resource Identifier** (URI)
  - URIs can take the form of URNs (**Uniform Resource Names**),
  - URLs (**Uniform Resource Locators**) are generally used by the industry
- By using the HTTP protocol, these identifiers allow software agents to **retrieve a machine-readable description** of the resource directly from the web
- Remember **NUNA**: URIs identify resources, but do not necessarily imply uniqueness of names

RDF - The Resource Description Framework

- The Resource Description Framework (RDF) is a W3C recommendation that serves as the **foundational data model** for interchange on the web

- Unlike the relational model, which is based on tables, RDF is inherently **graph-based**. This structure is highly effective for modelling complex, real-world knowledge where entities are interconnected in a non-linear fashion

RDF Triple

- All information in an RDF-based Knowledge Graph is represented through a simple, atomic structure known as a **triple**.
- A triple consists of three components:
  - a **Subject** (denotes the entity being described)
  - a **Predicate** (specifies the type of relationship or attribute)
  - an **Object** (indicates the related entity or a literal value)

- For example:

Leonardo da Vinci painted the Gioconda

[LeonardoDaVinci] --[painted]--> [Gioconda]

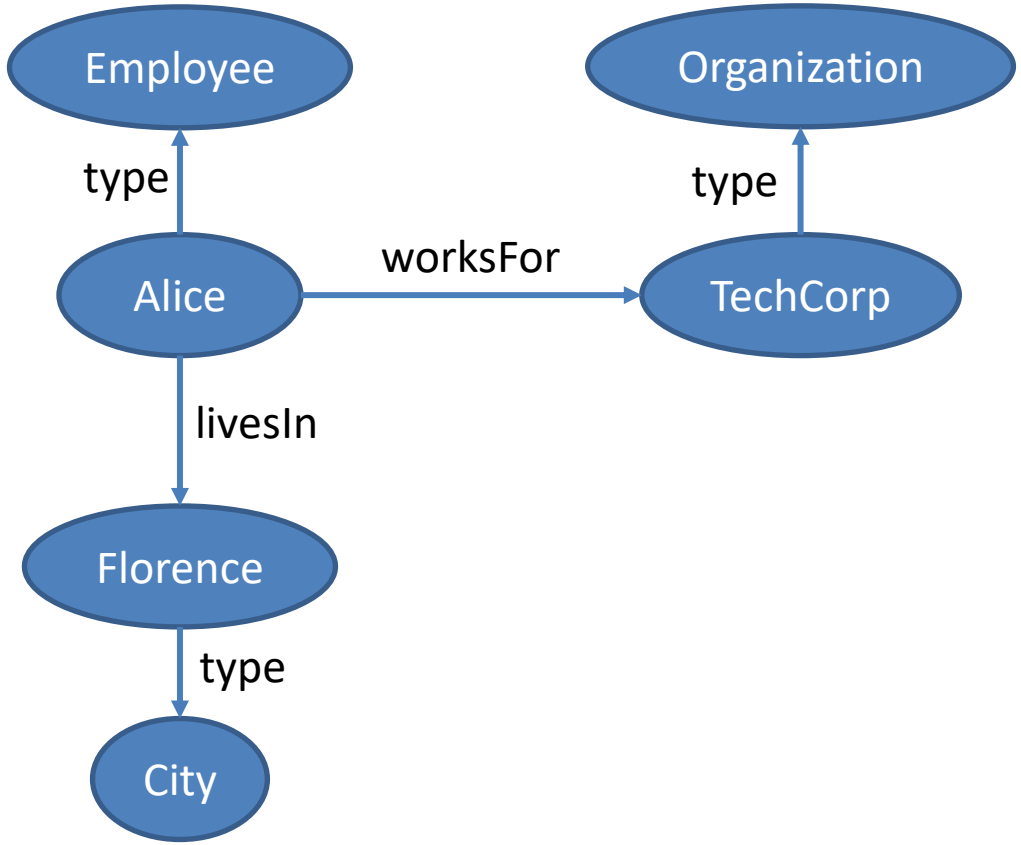
(Subject) (Predicate) (Object)

Directed Labelled Graphs

- While individual triples represent isolated facts, a collection of triples naturally forms a **directed labelled graph**
- the subjects and objects serve as **nodes**, while the predicates act as **directed**, **labelled edges**
- This graph-based approach allows for **multi-attribute** representation

[Alice] --[worksFor]--> [TechCorp]
[Alice] --[livesIn]--> [Florence]

- This interconnectedness is what transforms a simple database into a **Knowledge Graph** where the context of an entity is defined by its position and its surrounding relationships within the network

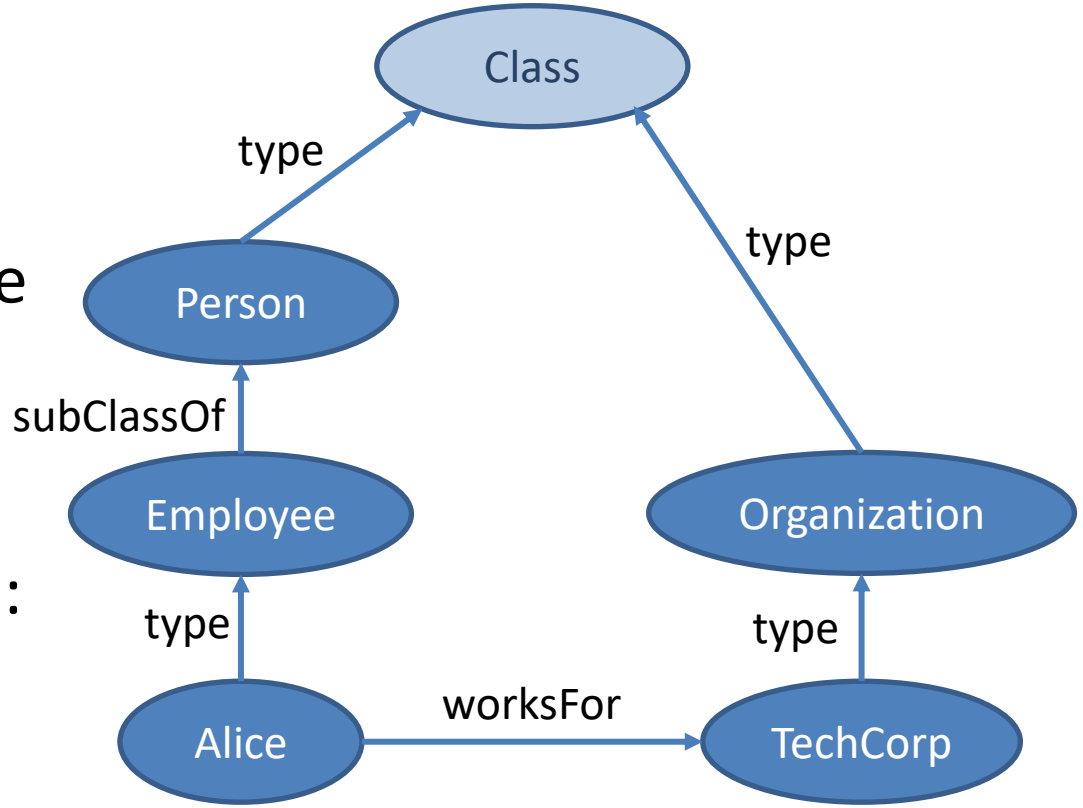


RDFS - Adding Schema to RDF

- While RDF provides the data model for triples, it lacks the vocabulary to define **categories** or **hierarchies**
- **RDF Schema** (**RDFS**) was introduced to fill this gap
- RDFS adds **basic inference semantics** to RDF. It allows us to define "Classes" (types of things) and organize them into hierarchies
- Crucially, *RDFS allows a reasoner to understand that if a resource belongs to a specific class, it also automatically belongs to all parent classes in the hierarchy*

RDFS Class Hierarchies and subClassOf

- The ***rdfs:subClassOf*** property is the cornerstone of taxonomic modelling in the Semantic Web.
- It defines a relationship where every instance of Class A (child) is also an instance of Class B (father)
- This enables the **creation of deep inheritance trees**
- This mechanism supports knowledge reuse: domain experts can link their specific local classes to well-known global classes, ensuring **interoperability** across different Knowledge Graphs



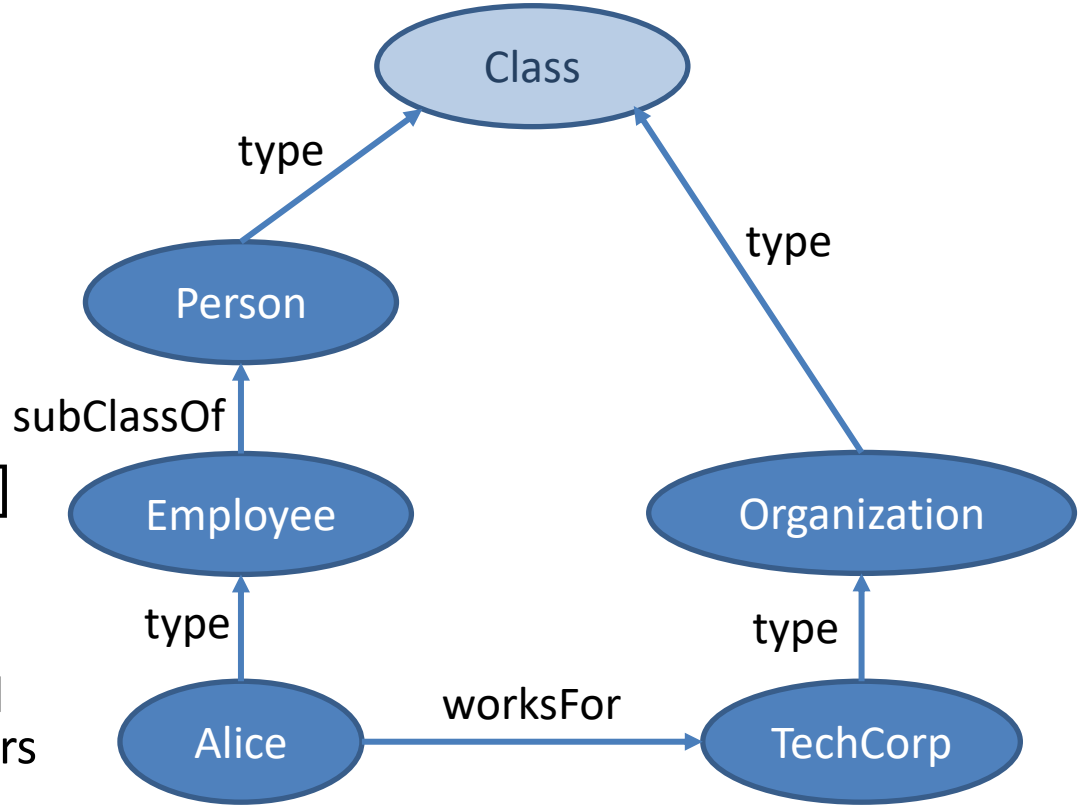
Can infer [Alice] --[type]--> [Person]

RDFS Property Hierarchies and Domain/Range

- RDFS extends hierarchical logic to properties via *rdfs:**subPropertyOf***. This allows for more granular relations to inherit the semantics of broader ones (e.g., "*paints*" could be a subproperty of "*creates*")
- Furthermore, RDFS introduces *rdfs:**domain*** and *rdfs:**range*** to constrain how properties are used

```
[worksFor] --(domain)--> [Person]
[worksFor] --(range)--> [Organization]
```

- These constraints act as **powerful inference tools**:
  - if a property has a domain of [Person] and it is applied to a subject, e.g. [Alice], the system automatically infers that [Alice] is an instance of a [Person], even if not explicitly stated



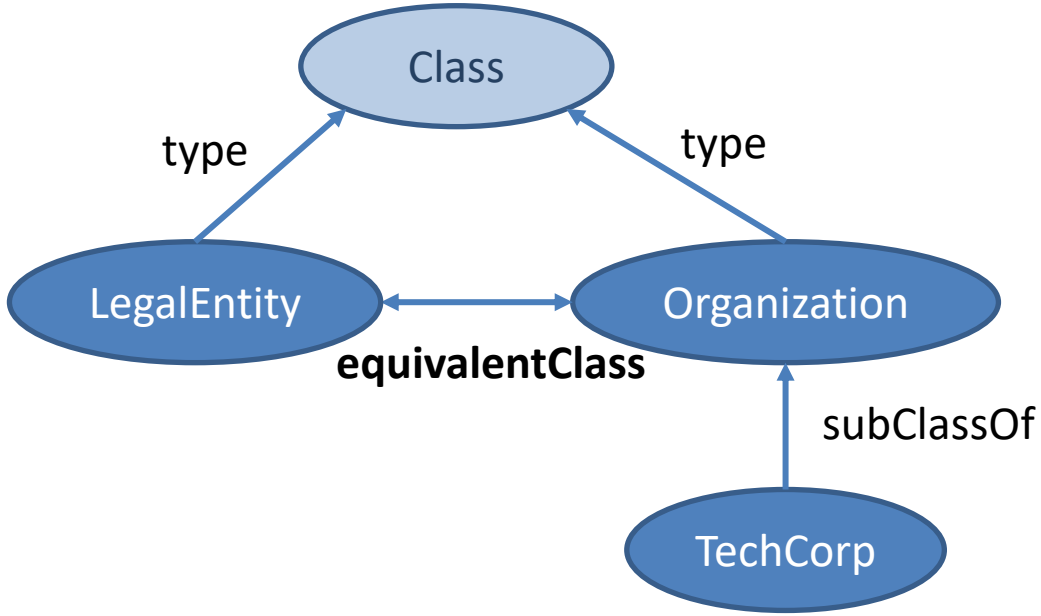
Since [Alice] [workFor] an [Organization],
I can infer that [Alice] is a [Person]

Web Ontology Language (OWL)

- While RDF Schema (RDFS) provides a solid foundation for defining classes and property hierarchies, its knowledge expressivity is often too limited for complex, real-world domains
- The transition to OWL is driven by the need for a **more expressive language** that can formalize common-sense domain rules and handle requirements that RDFS cannot address, such as specifying that two classes are mutually exclusive
- Technically, OWL builds upon RDFS by adopting its hierarchical constructs and **introducing many additional constructs**, all of which are represented as RDF triples within the knowledge graph

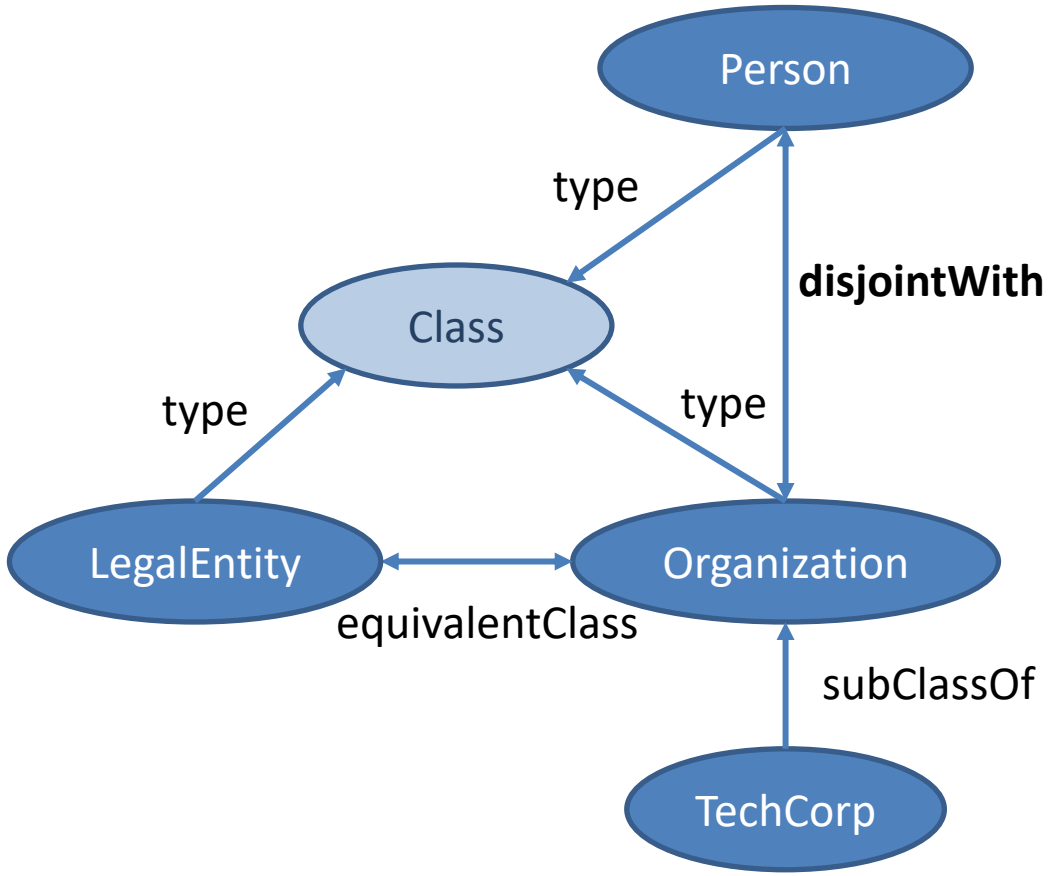
Equivalence

- Equivalence in OWL allows an engineer to declare that **two different identifiers refer to the same conceptual entity or relationship**
- The *owl:equivalentClass* predicate is used in the TBox to state that two class descriptions have the exact same set of individuals as members
- Similarly, *owl:equivalentProperty* is used to define that two property names are semantically identical; for instance, the properties "*beside*" and "*nextTo*" can be declared equivalent so that a triple using one automatically implies the other



Disjunction

- Disjunction is used explicitly separate concepts that cannot overlap
- The *owl:disjointWith* axiom is used to declare that two classes are **mutually exclusive**, meaning no individual can be a member of both classes simultaneously
- For example, an ontology can formalize that "*Person*" and "*Organization*" are disjoint classes; if a reasoning engine detects an individual belonging to both, it flags a **logical inconsistency**



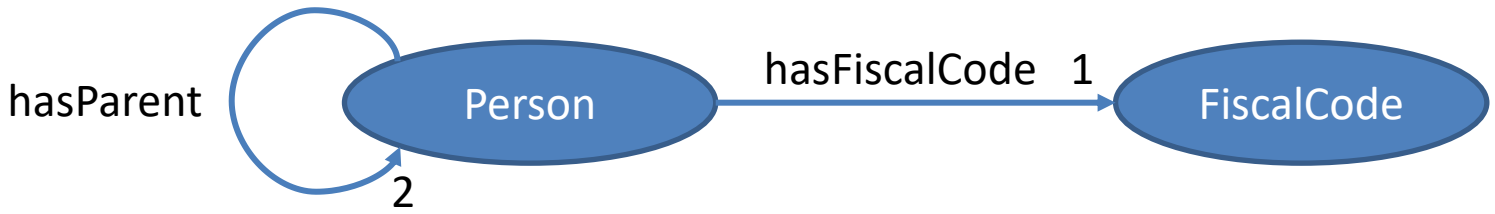
Property Axioms and Characteristics

- OWL include also construct working on properties
- For example:
  - *owl:TransitiveProperty* allows for **chains of relations** to be inferred (A is in B, B is in C -> A is in C)
  - *owl:SymmetricProperty* ensures that **bidirectional relations** (like "friendOf")
  - *owl:inverseOf* links two properties as **opposites** (e.g., "hasParent" and "isChildOf")
- These axioms provide the "glue" that allows machines to navigate and understand complex relationship networks

Restrictions

- Cardinality constraints (*owl:cardinality*, *owl:maxCardinality*) allow us to **restrict the number of relationships** an entity can have

- For example, an ontology can specify that a "*Person*" must have exactly one "*FiscalCode*" or exactly two biological parents



Class Axioms	rdfs:subClassOf	Defines a hierarchical relationship where every instance of a subclass is also an instance of the superclass.
	owl:equivalentClass	Declares that two class descriptions have exactly the same set of individuals as members.
	owl:disjointWith	States that two classes are mutually exclusive; no individual can belong to both simultaneously.
	owl:AllDisjointClasses	A concise way to declare that a large set of classes are all pairwise disjoint.
Property Axioms	rdfs:subPropertyOf	Establishes a hierarchy for properties, where one relation is a more specific type of another.
	owl:equivalentProperty	Declares that two different property names are semantically identical.
	owl:inverseOf	Defines two properties as opposites (e.g., hasChild is the inverse of hasParent).
	rdfs:domain	Restricts the class of individuals that can act as the **subject** of a property.
	rdfs:range	Restricts the class of individuals or data types that can act as the **object** of a property.
Property Characteristics	owl:TransitiveProperty	If A is related to B, and B to C, then A is automatically inferred to be related to C.
	owl:SymmetricProperty	Defines a bidirectional relationship; if A is related to B, then B is related to A.
	owl:AsymmetricProperty	Explicitly forbids a bidirectional relationship, enabling consistency checks for logic violations.
	owl:FunctionalProperty	Constrains a property so an individual can be related to at most one unique value through it.
	owl:InverseFunctionalProperty	States that the inverse of the property is functional (often used for unique identifiers).
	owl:ReflexiveProperty	States that every individual is related to itself via this property.
	owl:IrreflexiveProperty	Forbids an individual from being related to itself via this property.
	owl:propertyChainAxiom	Allows defining a property based on a sequence of other properties (e.g., parent's parent is grandparent).

Identity Axioms	owl:sameAs	States that two different URIs refer to the exact same real-world individual (critical for NUNA).
	owl:differentFrom	Explicitly declares that two individuals are distinct to prevent incorrect reasoning merges.
	owl:AllDifferent	Declares that all members of a specific set of individuals are distinct from one another.
Restrictions & Constructors	owl:someValuesFrom	**Existential Restriction:** An individual must have at least one relationship of this type to a specific class.
	owl:allValuesFrom	**Universal Restriction:** All relationships of this type for an individual must point to members of a specific class.
	owl:hasValue	Restricts a class to individuals that are linked to a specific, unique individual or literal value.
	owl:cardinality	Specifies the exact number of relationships an individual must have via a specific property.
	owl:intersectionOf	Defines a class as the logical **AND** (overlap) of multiple other classes.
	owl:unionOf	Defines a class as the logical **OR** (combination) of multiple other classes.
	owl:complementOf	Defines a class containing everything that is **NOT** a member of the specified class.

RDF vs RDFS vs OWL

RDF (Resource Description Framework)

- fundamental data model
- based on a triples (*subject*, *predicate*, *object*)
- graph model
- no inherent semantics
- no logical inference

The Graph

RDFS (RDF Schema)

- provide a basic vocabulary to structure RDF data
- introduces classes (*rdfs:Class*) and hierarchies (*rdfs:subClassOf, rdfs:subPropertyOf)*
- specify subjects (*rdfs:domain*) and objects (*rdfs:range*) of a given property
- Basic Inference Semantics

The Hierarchy

OWL (Web Ontology Language)

- represent knowledge in a rich and complex way
- based on Descriptive Logics
- constraint declaration
  - disjunction,
  - cardinality
  - property characteristics
- can infer new knowledge
- can verify the logical coherence

Logic Rigor

Serialization Formats

- Because RDF is an abstract data model, the triples must be "serialized" into a textual format for storage and transmission between systems
- Several standard formats exist, each catering to different needs. **N-Triples** is a very simple format with one triple per line, while **Turtle** provides a more human-readable and compact syntax.
- For machine-heavy integration, **RDF/XML** is the legacy W3C standard, and **JSON-LD** is the modern choice for web-based applications and search engine optimization

- Turtle

```
@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

# Using a semicolon (;) to add more info about Alice
ex:Alice
    rdf:type ex:Employee ;
    ex:worksFor ex:TechCorp .

# A separate block for TechCorp
ex:TechCorp
    rdf:type ex:Organization .
```

- N-Triples

```
<http://example.org/Alice> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/Employee> .
<http://example.org/Alice> <http://example.org/worksFor> <http://example.org/TechCorp> .
<http://example.org/TechCorp> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/Organization> .
```

- Turtle

```
@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

# Using a semicolon (;) to add more
ex:Alice
    rdf:type ex:Employee ;
    ex:worksFor ex:TechCorp .

# A separate block for TechCorp
ex:TechCorp
    rdf:type ex:Organization .
```

Prefixes

- To prevent the syntax from becoming cluttered with long URIs, Turtle uses "Prefixes"
- A prefix acts as a shorthand for a namespace URI.
- For example, the prefix "dbr:" can represent "http://dbpedia.org/resource/". Thus, "dbr:Florence" is automatically expanded to the full URI by the RDF parser
- Standard prefixes like "rdf:", "rdfs:", and "owl:" are used across almost all Knowledge Graphs

- N-Triples

```
<http://example.org/Alice> <http://www.w3.
<http://example.org/Alice> <http://example.
<http://example.org/TechCorp> <http://www.w3.
```

- RDF/XML

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://example.org/">

  <ex:Employee rdf:about="http://example.org/Alice">
    <ex:worksFor rdf:resource="http://example.org/TechCorp"/>
  </ex:Employee>

  <ex:Organization rdf:about="http://example.org/TechCorp"/>

</rdf:RDF>
```

- JSON-LD

```
{
  "@context": {
    "ex": "http://example.org/",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "type": "@type",
    "worksFor": { "@id": "ex:worksFor", "@type": "@id" }
  },
  "@id": "ex:Alice",
  "type": "ex:Employee",
  "worksFor": "ex:TechCorp"
}
```

N-Triples - Turtle - RDF/XML - JSON-LD

Format	Why it exists	The "Vibe"
N-Triples	**Simplicity.** Each line is a self-contained fact.	The "Raw Log File"
Turtle	**Human-friendliness.** It's easy to write and read.	The "Developer's Choice"
RDF/XML	**Compatibility.** It was created when XML ruled the web.	The "Legacy Standard"
JSON-LD	**Web Integration.** It makes RDF look like standard JSON.	The "Modern API"

SPARQL - Querying the Web of Data

- SPARQL (SPARQL Protocol and RDF Query Language) serves as the standardized **semantic query language for the Semantic Web**, acting as a counterpart to SQL for relational databases
- Its primary purpose is to enable users and software agents to **retrieve and manipulate data stored in RDF format** across various distributed Knowledge Graphs

```
PREFIX ex: <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?person ?company
WHERE {
  ?person rdf:type ex:Person .
  ?person ex:worksFor ?company .
}
```

SPARQL - Querying the Web of Data

- The core of a SPARQL query is the "Graph Pattern," which consists of a set of triple patterns
- These patterns resemble RDF triples but **incorporate variables** (prefixed with a interrogation mark, e.g., ?f) to represent unknown elements
- A typical query structure includes a **SELECT** clause to specify which variables to return and a **WHERE** clause containing the triple patterns to be matched against the dataset

```
PREFIX ex: <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?person ?company
WHERE {
  ?person rdf:type ex:Person .
  ?person ex:worksFor ?company .
}
```

SPARQL - Querying the Web of Data

- To refine search results, SPARQL provides the **FILTER** keyword, which allows for the application of constraints on **variable values**, such as date ranges, language tags, or string comparisons
- Additionally, because the Semantic Web operates under the Open World Assumption, data is often incomplete. The **OPTIONAL** clause is essential here;
  - it allows the query to return results even if certain parts of the graph pattern are missing for a specific individual, preventing the entire result row from being discarded due to a lack of a specific triple

```
PREFIX ex: <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?name ?age ?email
WHERE {
  ?person rdf:type ex:Employee ;
        ex:worksFor ex:TechCorp ;
        ex:hasName ?name ;
        ex:hasAge ?age .

  OPTIONAL { ?person ex:hasEmail ?email .  }
  FILTER (?age > 18)
}
```

Foundations of Symbolic Reasoning

- Reasoning, in the context of Knowledge Engineering, is the application of mathematical proof theory to derive implicit knowledge from explicitly stated facts
- By applying logical rules (axioms), a machine can infer new triples that were never manually entered
- For example, if the system knows that

Michelangelo painted the Sistine Chapel

and that

The Sistine Chapel is in the Vatican

reasoning can automatically conclude that

Michelangelo worked in the Vatican

Description Logics (DL)

- The Web Ontology Language (OWL) is grounded in **Description Logics**, a family of knowledge representation languages tailored for the Semantic Web. Specifically, OWL 2 DL corresponds to the highly expressive SROIQ logic
- Description Logics are designed to be "decidable fragments" of First-Order Logic (FOL), meaning that they are powerful enough to **describe complex concepts** while ensuring that a computer can always **finish the reasoning process in a predictable amount of time**

Subsumption

- Subsumption is the foundational reasoning task of checking if one class description is more general than another
- This determines hierarchical "is-a" relationships
- A reasoner uses subsumption to build the inferred class hierarchy of an ontology
- For instance, if the TBox defines a "*Full Professor*" as a subclass of "*Teacher*", the reasoner verifies that every instance of "*Full Professor*" inherits membership in "*Teacher*"

Subsumption

$\texttt{Girl} \sqsubseteq \texttt{Woman}$

- reads as "Girl is subsumed by Woman" or "Woman subsumes Girl"
- it means: "a girl is a woman"
- More generally, $C \sqsubseteq D$ indicate that every individual in C is also described by D
- Note that: if $C \sqsubseteq D$ and $D \sqsubseteq C$, then C and D are equivalent, i.e., $C \equiv D$

Satisfiability and Consistency

- Satisfiability checks whether a complex concept description is **logically possible** or if it contains internal **contradictions**
- Consistency checking evaluates the entire Knowledge Base (TBox + ABox) to ensure that the **facts do not violate the rules**
- If a fact is asserted that contradicts an axiom—such as an individual belonging to two classes defined as disjoint—the reasoner will flag a "Clash," indicating a logical error in the model

Instance Checking

- Instance checking determines whether a **specific individual belongs to a certain class** based on its properties and the ontology's rules.
- This is often used for semantic classification. If an individual "X" has the property "hasDegreeInMedicine," and the ontology defines a "Doctor" as any person with a Medical Degree, the reasoner concludes that "X" is a "Doctor."

Instance Checking

```
# TBox: Defining the Doctor class
:Doctor owl:equivalentClass [
    rdf:type owl:Class ;
    owl:intersectionOf (
        :Person
        [ rdf:type owl:Restriction ;
          owl:onProperty :hasDegree ;
          owl:someValuesFrom :MedicalDegree
        ]
    )
] .

# ABox: Asserting facts
:individual_X rdf:type :Person ;
              :hasDegree :degree_Y .

:degree_Y rdf:type :MedicalDegree .
```

:individual_X rdf:type :Doctor.

Materialization

- Materialization is the process of **computing all implicit triples** and adding them explicitly to the Knowledge Graph
- This effectively **completes the data**, making it more accessible for standard query engines that do not have built-in reasoning capabilities
- Materialization
  - makes data retrieval faster ✅
  - increases storage requirements ❌

The Role of Transitivity

- Transitivity is one of the most powerful inference tools. If a property like "*locatedIn*" is declared as an *owl:TransitiveProperty*, the reasoner can follow chains of relations
- Example: If

[*Florence*] --(*locatedIn*)--> [*Tuscany*]

and

[*Tuscany*] --(*locatedIn*)--> [*Italy*]

the reasoner automatically materializes the fact that

[*Florence*] --(*locatedIn*)--> [*Italy*]

Inverse Relationships

- Inverse properties allow the system to understand **bidirectional** links.
- By declaring *owl:inverseOf*, the machine knows that if "A is the parent of B," then "B is the child of A" and can **materialize the missing fact**
- This ensures that the Knowledge Graph remains consistent regardless of which direction the information was originally entered

Domain, Range, Equality...

- Similarly, new triples can be inferred exploiting
  - *rdfs:domain* and *rdfs:range*: If the property "paints" has a domain of "Artist," and we assert "Michelangelo paints the Sistine Chapel," the reasoner automatically infers "Michelangelo is an Artist." This **type inference** allows Knowledge Graphs to be self-populating.
  - *owl:sameAs*: If *my:Florence* is the same as *dbr:Florence*, any property known about one identifier is automatically inferred for the other, allowing for seamless **data integration** across the web.

Limits of Classical Reasoning - Scalability

- While expressive and logically sound, classical reasoning faces a "Scalability vs. Expressivity" trade-off

- As the size of the Knowledge Graph (ABox) and the complexity of the rules (TBox) grow, the time required for reasoning can increase exponentially

The Practical Constraints of Pure Symbolic Systems

- Symbolic Knowledge Representation and Reasoning (KRR) offer unparalleled precision and explainability...
- ...but they **face significant hurdles in real-world deployment**!

- The most prominent issue is the **Knowledge Acquisition Bottleneck**:
  - translating messy, noisy, and unstructured real-world data into neat logical symbols is a laborious, manual process that requires specialized ontological engineers
- Symbolic systems are notoriously brittle:
  - they typically operate under the assumption that **all data is certain and consistent**. In case of noise reasoners often fail (i.e., no answer or crash).
- Problem with **unstructured domains**:
  - e.g., image recognition or sensor data analysis, where knowledge is better represented as continuous vectors rather than discrete symbols

Neuro-Symbolic KGQA

- Knowledge Graph Question Answering (KGQA) aims to bridge the gap between natural language queries and structured graph data.

- Purely symbolic methods often fail if the generated query (e.g., SPARQL) contains a **single syntax error** or if the **KG is incomplete**
- And, writing **SPARQL queries manually is complex**, require knowledge on the KG, and time consuming

- Neural approaches, particularly Large Language Models (LLMs), excel at **understanding intent** but suffer from **hallucinations** and a **lack of domain-specific technical knowledge**

Incrementing Symbolic Knowledge via NN

- Knowledge Graphs are often incomplete, missing factual triples required to answer specific queries—a problem known as IKGQA (Incomplete KGQA)
- Neural networks solve this by acting as **automated knowledge generators**
- For example, by using NN (LLM)
  - to generate missing factual triples exploiting internal weights
  - to extract new entities and relations from unstructured documents

The Neuro-Symbolic Cycle

- The ultimate goal of Knowledge Engineering is now the creation of a **Continuous Neuro-Symbolic Cycle**
- In this cycle, symbolic knowledge (Ontologies/KGs) serves as the *System 2* bedrock, providing **constraints and priors** for neural learning (*System 1*).
- Conversely, the regularities and patterns learned by the neural layers from **raw data are extracted and lifted back into the symbolic layer** as new axioms or rules
- This mutual improvement ensures that the system is not only **data-efficient** and **robust to noise** but also **remains auditable and explainable**