Ontology, OWL, and RDF

Schema, Language, and Data

Motivation

1. We know that eating poisonous food can kill people (domain).
2. We want to build a logical structure can help people avoid poisonous food (language).
3. To do that, we need to represent knowledge about food, people, and restrictions in a structured way (data).

How can we make machines understand this knowledge?

Our thinking Process

1. We have an idea about a **domain**.

- We make conceptual descriptions of the domain.
- We call it modeling.

2. To express the model formally, we need a **language**.

- We need the grammar to express meaning (semantics) through syntax.

3. Then, finally, we can use the language to store/share the **data** about the domain in a structured way.

Ontology/OWL/RDF: Definitions

Onto**log**y = Logic of 'CD (Conceptual Description, in other words, model) in D (domain)'

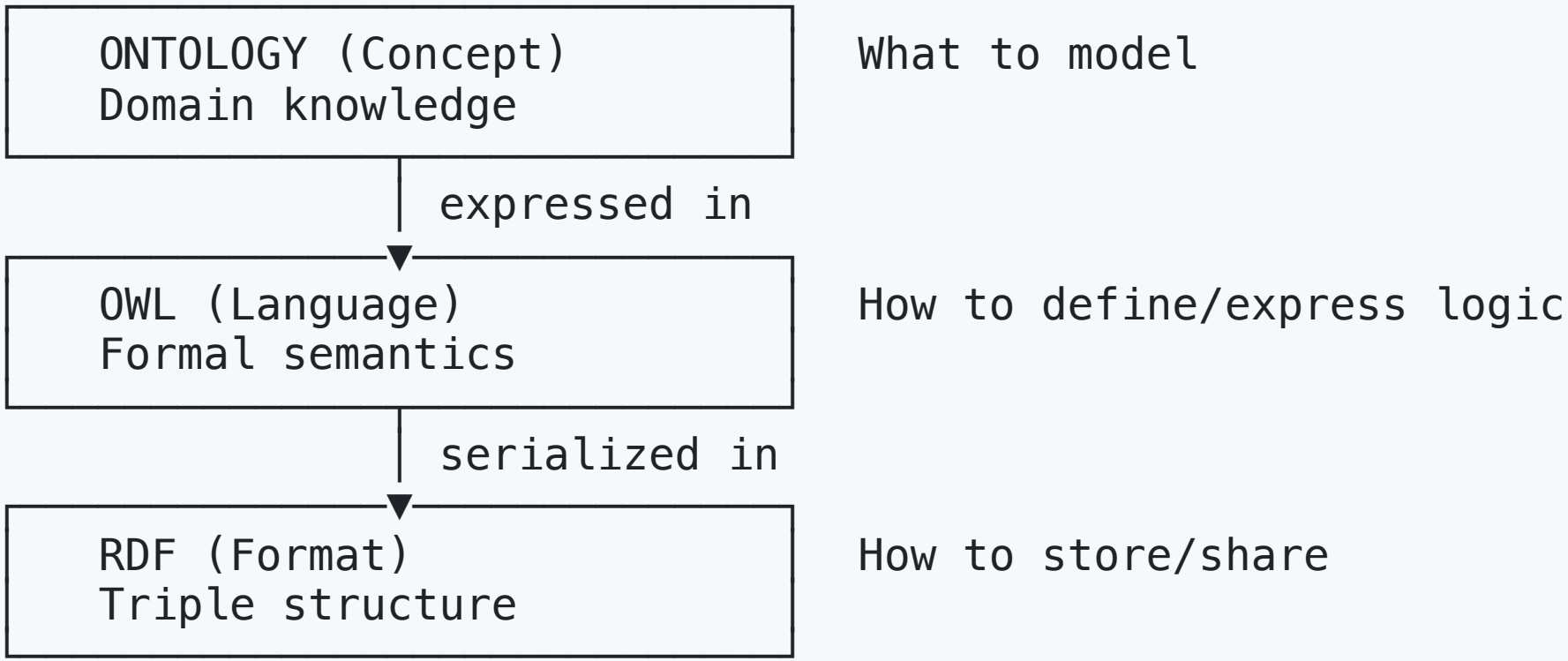
OW**L** = Web Ontology **Language** (for expressing/defining the logic)

RDF = **Resource Description** (not concept description) Framework (for storing the data)

- Focus on **Description** of logical structure in a domain

The Big Picture

They form a layered stack:



Each layer has a different responsibility

What Is an Ontology?

Ontology = Conceptual description of a domain

It defines CPC for Conceptual Description (CD) in a Domain (D):

- **What exists** (Classes: Person, Food, Restriction)
- **How things relate** (Properties: eats, hasAllergy, incompatibleWith)
- **What rules apply** (Constraints: Vegetarians don't eat meat)

Using Software Architecture Analogy

If you finished ASE 456, you may remember the concept of software architecutre with three elements (ERC):

- **Entity** (modules, classes)
- **Relationship** (relationships between components)
- **Constraints** (rules about how components can interact)

Ontology is similar but for knowledge domains

Examples from diet domain

- Person, Food, Ingredient, Restriction
- hasRestriction, containsIngredient
- "Vegetarian is incompatible with Meat"

Ontology is the **idea and design (such as CPC) in a domain**, not the file format

What Is OWL?

OWL (Web Ontology Language) = Language to write ontologies

OWL allows you to define:

- Class hierarchies (Beef ⊆ Meat ⊆ Ingredient)
- Logical rules (If X is Vegan, X is Vegetarian)
- Restrictions (Person eats only Food)
- Inference patterns (Reasoning about relationships)

Example rules:

```
Vegetarian ⊆ DietaryRestriction
Vegan ⊆ Vegetarian
Human eats only Food
```

OWL provides **logic and formal meaning**

What Is RDF?

RDF (Resource Description Framework) = Data model

Everything is represented as triples:

```
Subject – Predicate – Object
```

Examples:

```
Alice – type – Person
Alice – hasRestriction – Vegetarian
Pizza – containsIngredient – Cheese
Vegetarian – incompatibleWith – Meat
```

RDF provides **structure and data exchange format**

How They Connect

Layer	Role	Example
Ontology	Knowledge design	"Vegetarians don't eat meat"
OWL	Logic & schema language	`Vegetarian incompatibleWith Meat`
RDF	Storage & transport	`:Vegetarian :incompatibleWith :Meat .`

All three work together to create machine-understandable knowledge

Example 1: From Idea to Data

Step 1 — Ontology Idea (Concept)

Domain knowledge:

> "All vegetarians are people with dietary restrictions"
> "Vegetarians cannot eat meat"

Step 2 — OWL Representation (Logic)

OWL expresses these rules:

```
:Vegetarian rdf:type owl:Class ;
    rdfs:subClassOf :DietaryRestriction .

:Vegetarian owl:disjointWith :MeatEater .

:incompatibleWith rdf:type owl:ObjectProperty ;
    rdfs:domain :DietaryRestriction ;
    rdfs:range :IngredientCategory .

:Vegetarian :incompatibleWith :Meat .
```

This adds logical meaning and constraints

Step 3 — RDF Storage (Data)

Same information stored as RDF triples:

```
@prefix : <http://example.org/diet#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

:Vegetarian rdf:type owl:Class .
:Vegetarian rdfs:subClassOf :DietaryRestriction .
:Vegetarian :incompatibleWith :Meat .
```

Now machines can store, query, and exchange it

Example 2: Instance Data (Knowledge Graph)

Real-world fact stored in RDF:

```
:Alice rdf:type :Person .
:Alice :hasRestriction :Vegetarian .
```

Using OWL rules:

```
:Vegetarian rdfs:subClassOf :DietaryRestriction .
```

Reasoner automatically infers:

```
:Alice :hasRestriction :DietaryRestriction .
```

Logic creates new knowledge automatically

Example 3: Diet Restriction Reasoning

Step 1: Define the ontology

```
# Classes
:Person rdf:type owl:Class .
:Food rdf:type owl:Class .
:Ingredient rdf:type owl:Class .
:Meat rdfs:subClassOf :Ingredient .
:Beef rdfs:subClassOf :Meat .

# Properties
:eats rdf:type owl:ObjectProperty ;
    rdfs:domain :Person ;
    rdfs:range :Food .

:containsIngredient rdf:type owl:ObjectProperty ;
    rdfs:domain :Food ;
    rdfs:range :Ingredient .
```

Step 2: Add restriction rules (OWL)

```
:Vegetarian rdf:type owl:Class ;
    rdfs:subClassOf :Person ;
    owl:disjointWith [
        rdf:type owl:Restriction ;
        owl:onProperty :eats ;
        owl:someValuesFrom [
            rdf:type owl:Restriction ;
            owl:onProperty :containsIngredient ;
            owl:someValuesFrom :Meat
        ]
    ] .
```

Translation: "A Vegetarian cannot eat any food that contains meat"

Step 3: Add data (RDF)

```
:Alice rdf:type :Vegetarian .
:BeefBurger rdf:type :Food ;
    :containsIngredient :Beef .

# Wrong data
:Alice :eats :BeefBurger .
```

Step 4: Reasoner detects inconsistency

```
❌ Logical inconsistency detected!

Reason:
- Alice is Vegetarian
- Vegetarian cannot eat food with Meat
- Beef is a type of Meat
- BeefBurger contains Beef
- Alice eats BeefBurger → CONFLICT!
```

Example 4: Constraint Checking with OWL

OWL Rule:

```
:Person rdf:type owl:Class ;
    rdfs:subClassOf [
        rdf:type owl:Restriction ;
        owl:onProperty :eats ;
        owl:allValuesFrom :Food
    ] .
```

Translation: "A person can only eat things that are Food"

Valid Data:

```
:Alice rdf:type :Person .
:Apple rdf:type :Food .
:Alice :eats :Apple .
```

Result: ✓ Valid (Apple is Food)

Invalid Data:

```
:Bob rdf:type :Person .
:Rock rdf:type :Mineral .  # Not Food!
:Bob :eats :Rock .
```

Result: ❌ Inconsistent (Rock is not Food)

OWL helps detect data errors automatically

Example 5: Multi-level Inference

Ontology hierarchy:

```
:Ingredient rdf:type owl:Class .
:Meat rdfs:subClassOf :Ingredient .
:Beef rdfs:subClassOf :Meat .
:Chicken rdfs:subClassOf :Meat .

:Dairy rdfs:subClassOf :Ingredient .
:Cheese rdfs:subClassOf :Dairy .
:Milk rdfs:subClassOf :Dairy .

# Vegan restrictions
:Vegan :incompatibleWith :Meat .
:Vegan :incompatibleWith :Dairy .
```

Data:

```
:Carol rdf:type :Person .
:Carol :hasRestriction :Vegan .

:Pizza :containsIngredient :Cheese .
```

Reasoning chain:

```
1. Carol is Vegan
2. Vegan incompatible with Dairy
3. Cheese is subclass of Dairy
4. Pizza contains Cheese
→ Carol cannot eat Pizza ✓
```

Multi-level reasoning through class hierarchies

Knowledge Graph Relationship

Knowledge Graphs use all three:

- **RDF** to store facts (triples)
- **OWL** to define meaning (logic)
- **Ontology** to organize structure (schema)

Example from diet domain:

```
# Ontology/OWL (Schema)
:Person rdf:type owl:Class .
:hasRestriction rdfs:domain :Person .

# Knowledge Graph (Data)
:Alice rdf:type :Person .
:Alice :hasRestriction :Vegetarian .
:Bob rdf:type :Person .
:Bob :hasAllergy :Peanut .
```

RDF Serialization Formats

The SAME data can be written in different formats:

Turtle (human-readable):

```
:Alice rdf:type :Person ;
    :hasRestriction :Vegetarian ;
    :age 25 .
```

RDF/XML:

```
<rdf:Description rdf:about="Alice">
    <rdf:type rdf:resource="Person"/>
    <hasRestriction rdf:resource="Vegetarian"/>
    <age>25</age>
</rdf:Description>
```

JSON-LD:

OWL Levels of Expressiveness

OWL comes in different flavors:

1. **OWL Lite** - Basic hierarchies

```
:Beef rdfs:subClassOf :Meat .
```

2. **OWL DL** - Description Logic (most common)

```
:Vegetarian owl:disjointWith :MeatEater .
```

3. **OWL Full** - Maximum expressiveness

```
# Complex reasoning patterns
```

Trade-off: More expressiveness = Slower reasoning

Common Misunderstandings

❌ Ontology = OWL

✓ Correct understanding:

- Ontology is the **conceptual model** (the ideas)
- OWL is the **language** to express it
- Like: Architecture (ontology) vs Blueprint notation (OWL)

❌ RDF is only for data

✓ Correct understanding:

RDF stores **everything**:

- Instance data (Alice is 25)
- Ontology structure (Person is a Class)
- OWL rules (Vegetarian ⊆ Person)

Example:

```
# All are RDF triples!
:Person rdf:type owl:Class .          # Ontology
:Alice rdf:type :Person .             # Data
:Vegetarian rdfs:subClassOf :Person . # Schema
```

❌ You need OWL for RDF

✓ Correct understanding:

- You can use RDF **without** OWL (just data)
- But OWL **adds reasoning and validation**

Without OWL:

```
:Alice :eats :Rock .  # No validation
```

With OWL:

```
:Person eats only :Food .
:Alice :eats :Rock .  # ❌ Error detected!
```

Practical Example: Complete System

1. Ontology Design (Concept)

```
Domain: Diet restrictions
Concepts: Person, Food, Ingredient, Restriction
Rules: Restrictions are incompatible with ingredients
```

2. OWL Implementation (Logic)

```
:DietaryRestriction rdf:type owl:Class .
:incompatibleWith rdf:type owl:ObjectProperty ;
    rdfs:domain :DietaryRestriction ;
    rdfs:range :Ingredient .
```

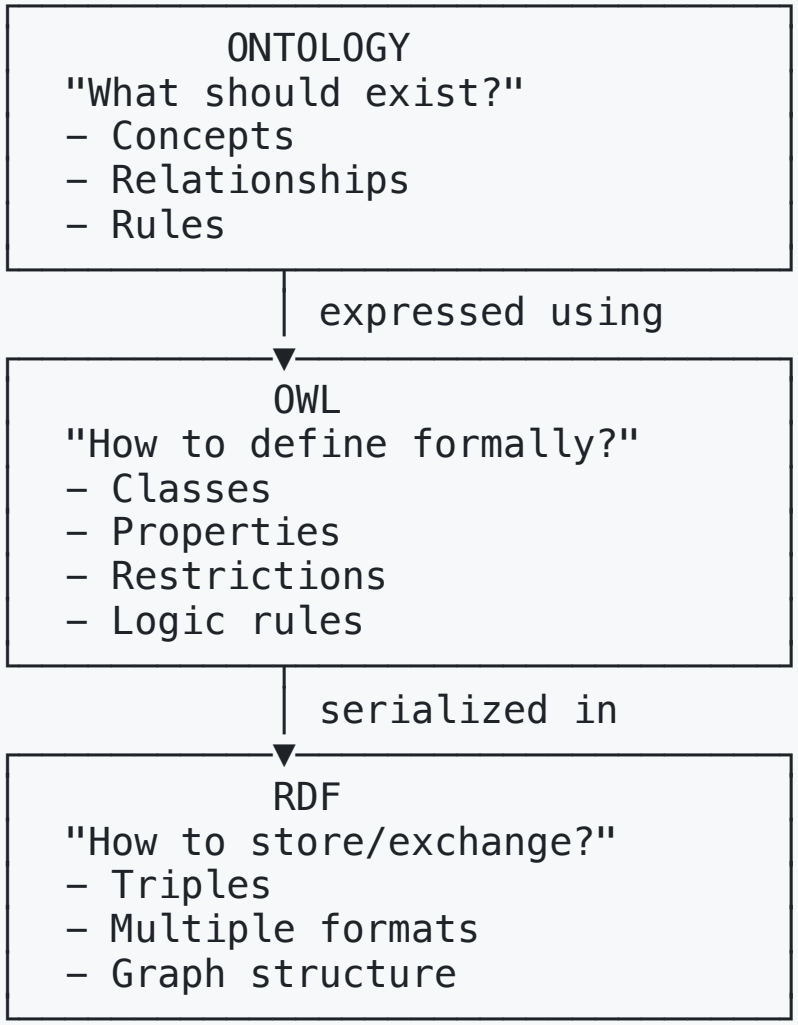
3. RDF Data (Instances)

```
:Alice rdf:type :Person ;
    :hasRestriction :Vegetarian .

:Vegetarian :incompatibleWith :Meat .

:BeefBurger :containsIngredient :Beef .
:Beef rdfs:subClassOf :Meat .
```

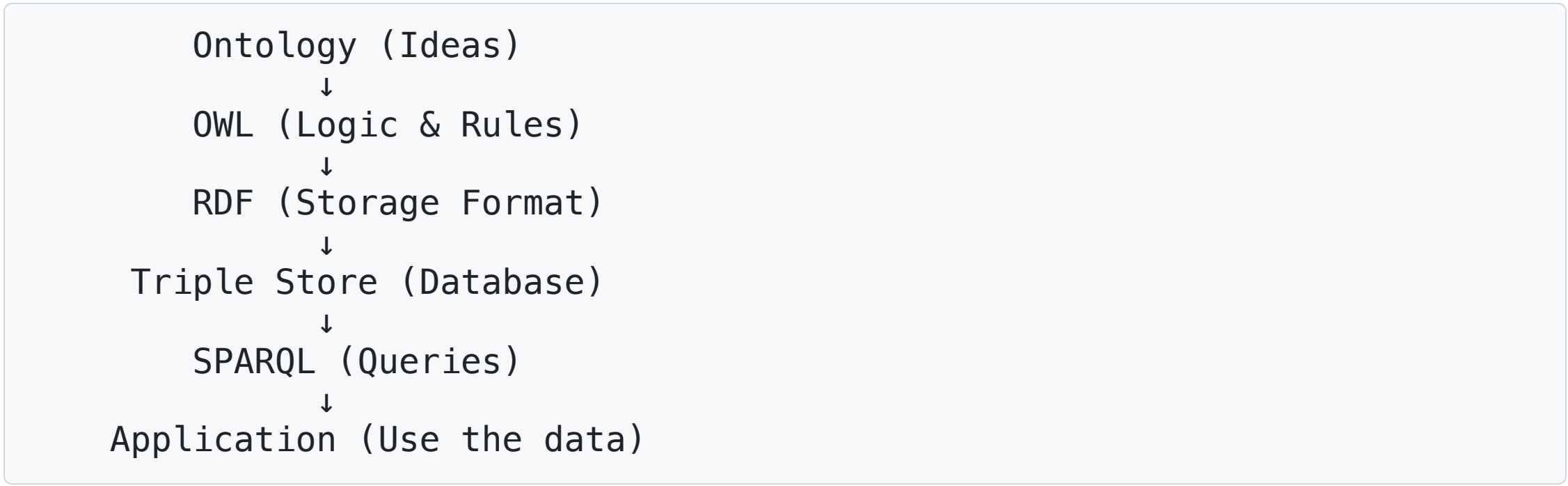
4. SPARQL Query (Usage)

```
SELECT ?food WHERE {
    :Alice :hasRestriction ?restriction .
    ?food rdf:type :Food .
    FILTER NOT EXISTS {
        ?food :containsIngredient ?ingredient .
        ?restriction :incompatibleWith ?category .
        ?ingredient rdfs:subClassOf* ?category .
    }
}
```

Visual Summary



The Complete Picture



When to Use What?

Use Ontology when:

- Designing domain model
- Defining concepts
- Planning knowledge structure

Use OWL when:

- Need logical reasoning
- Want automatic inference
- Need validation rules
- Building semantic systems

Use RDF when:

Final Takeaway

Together they create:

- ✓ Machine-readable meaning
- ✓ Logical reasoning
- ✓ Interoperable knowledge
- ✓ Flexible data integration
- ✓ Automatic validation

OWL adds intelligence

RDF adds structure

Ontology adds semantics

All three are needed for powerful semantic systems!

Key Points to Remember

1. **Ontology** = WHAT to model (concepts)
2. **OWL** = HOW to add logic and rules (language)
3. **RDF** = HOW to store and share (format)

Analogy:
hey create:**

- ✓ Machine-readable meaning
- ✓ Logical reasoning
- ✓ Interoperable knowledge
- ✓ Flexible data integration
- ✓ Automatic validation

OWL adds intelligence

Key Points to Remember

1. **Ontology** = WHAT to model (concepts)
2. **OWL** = HOW to add logic and rules (language)
3. **RDF** = HOW to store and share (format)

Analogy:

- Ontology = Architecture design
- OWL = Technical specifications
- RDF = Construction materials

They work together, not separately!