28/11/2022

Ontology Engineering

Dr Eleni Tsalapati

Marie Curie Fellow

Ontology Engineering

- Ontology engineering is **knowledge engineering**.
- Developing ontology engineering techniques, methodologies and tool support has been a **core research problem.**
- There are various interesting ontology engineering methodologies (some accompanied by relevant tools):
  - Methontology (Gomez-Perez and colleagues, 1997)
  - Uschold and King (2005)
  - Gruninger and Fox (2005)
  - …
  - SAMOD (2016)
  - …

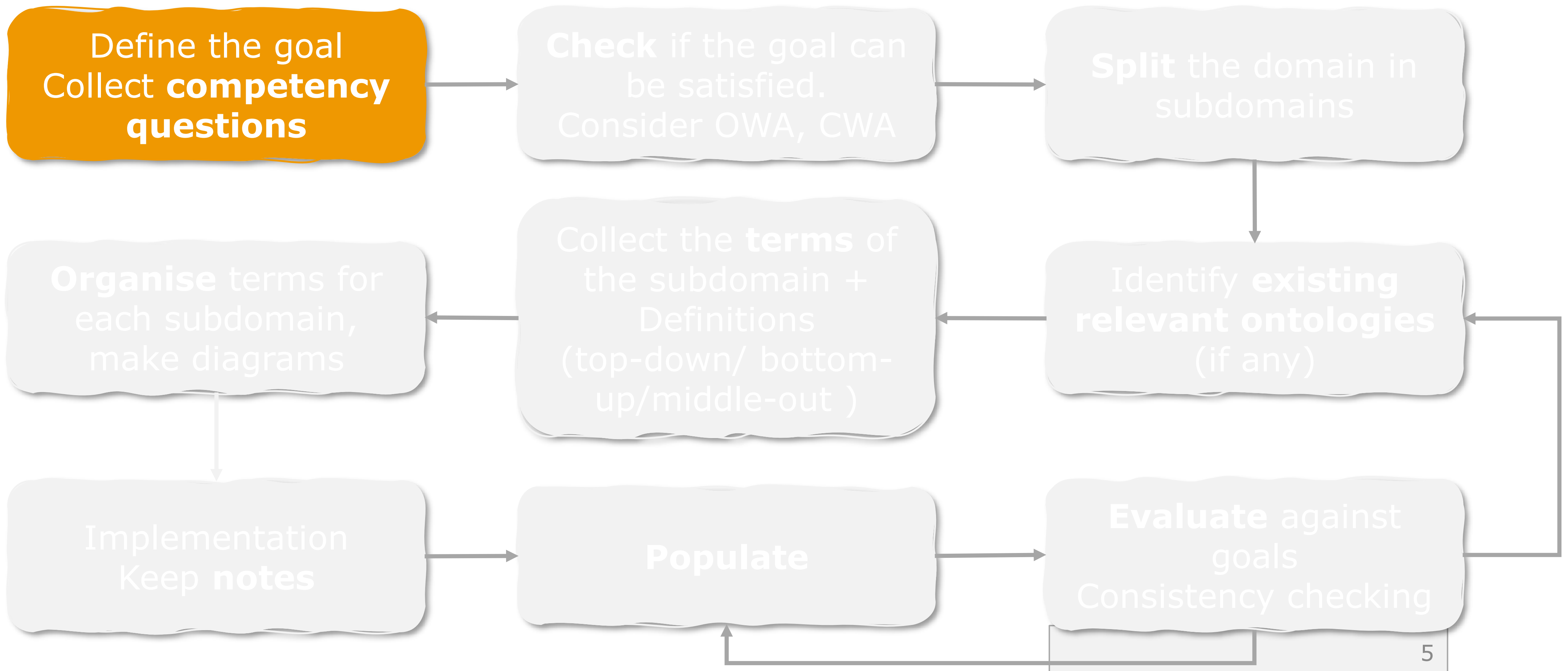
When to use OWL?

- Consider carefully the following features of OWL:
  - **Object-centered** (based on individuals with unique identity, classes and properties).
  - **Terminological**: Supports the building of complex terms (noun phrases) in the form of classes. Individuals are asserted to belong to these classes. There is no way to express arbitrary quantifications or disjunctions (as in FOL).
  - **Deductive**: not just a passive repository of assertions.
  - **Incremental**: partial, incomplete descriptions of individuals are acceptable and can be refined later.
  - Based on self-organization of concepts in a **subsumption** hierarchy.
  - Based on the **open world assumption**.

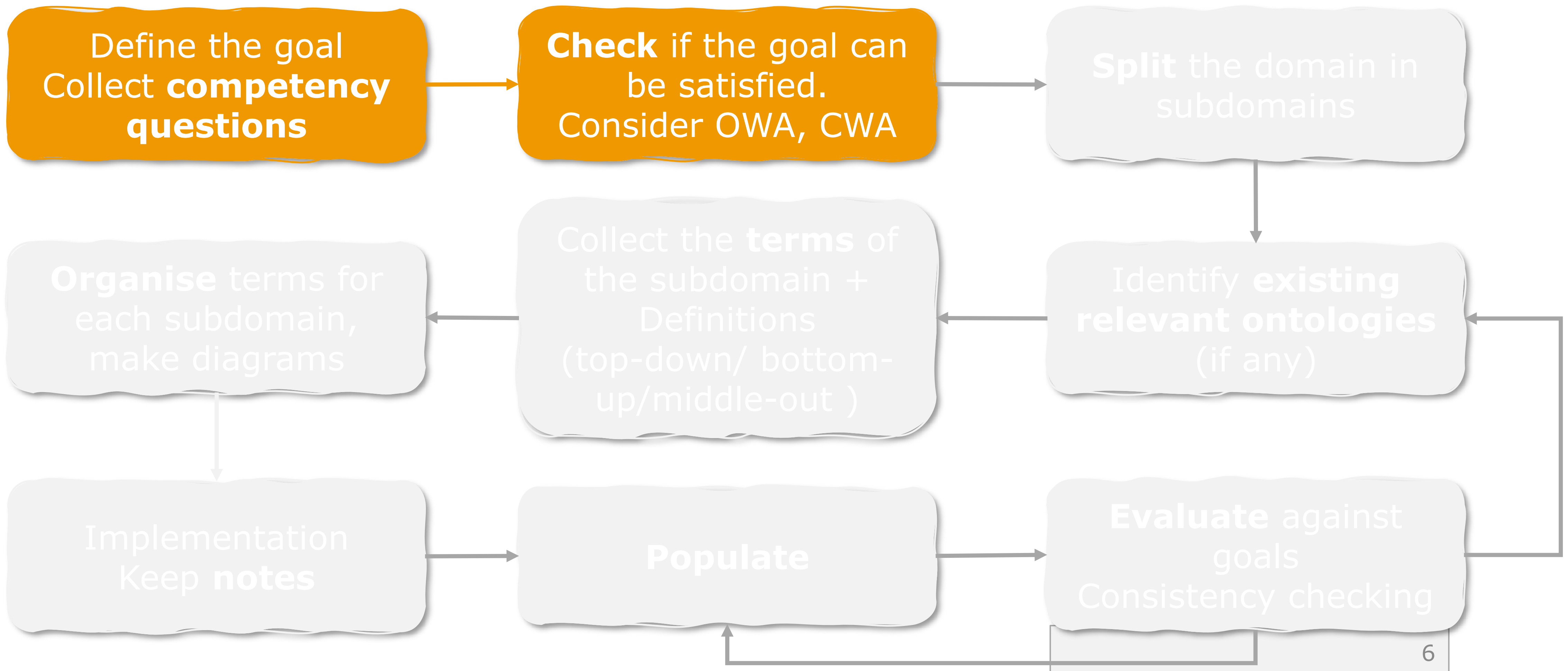
Key elements

- Main elements of an ontology:
  - Entities
    - Classes
    - Individuals (not always)
    - Properties
  - Property restrictions (OWL)
  - Class expressions (OWL)
  - General axioms/rules (OWL/SWRL)
  - Annotations

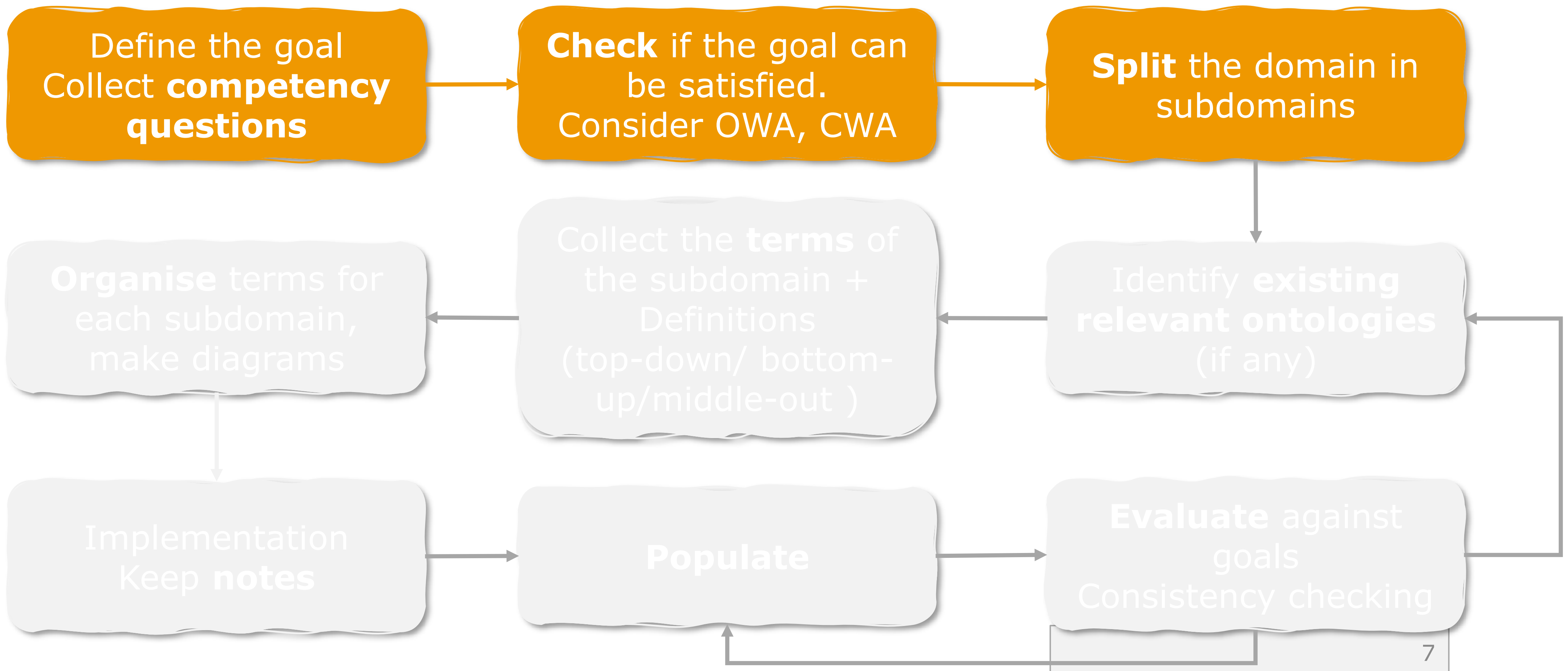
Ontology Development Pipeline


Define the goal
Collect **competency questions**
Check if the goal can be satisfied. Consider OWA, CWA
Split the domain in subdomains
Identify **existing relevant ontologies** (if any)
Collect the **terms** of the subdomain + Definitions (top-down/ bottom-up/middle-out)
Organise terms for each subdomain, make diagrams
Implementation
Keep **notes**
Populate
Evaluate against goals
Consistency checking

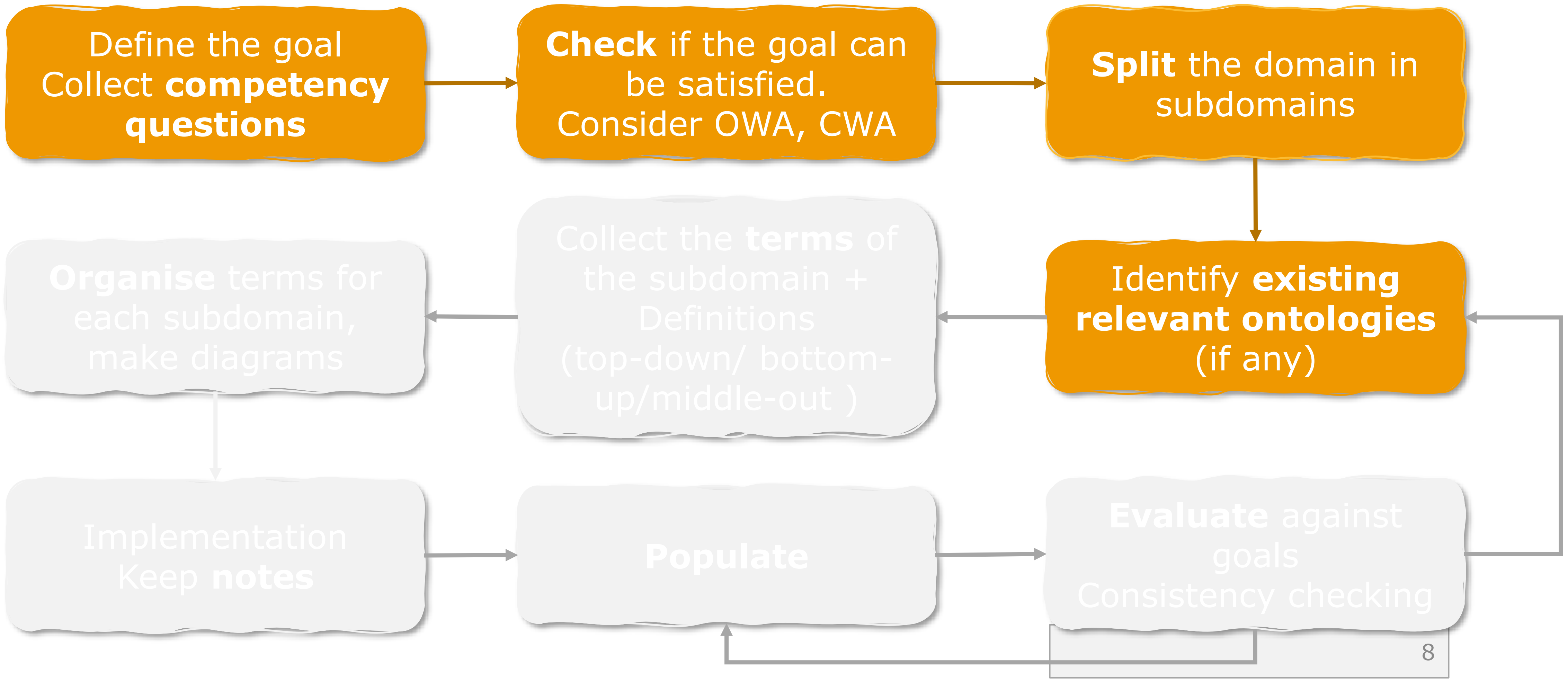
Ontology Development Pipeline


Define the goal
Collect **competency questions**
Check if the goal can be satisfied.
Consider OWA, CWA
Split the domain in subdomains
Identify **existing relevant ontologies** (if any)
Collect the **terms** of the subdomain + Definitions (top-down/ bottom-up/middle-out)
Organise terms for each subdomain, make diagrams
Implementation
Keep **notes**
Populate
Evaluate against goals
Consistency checking

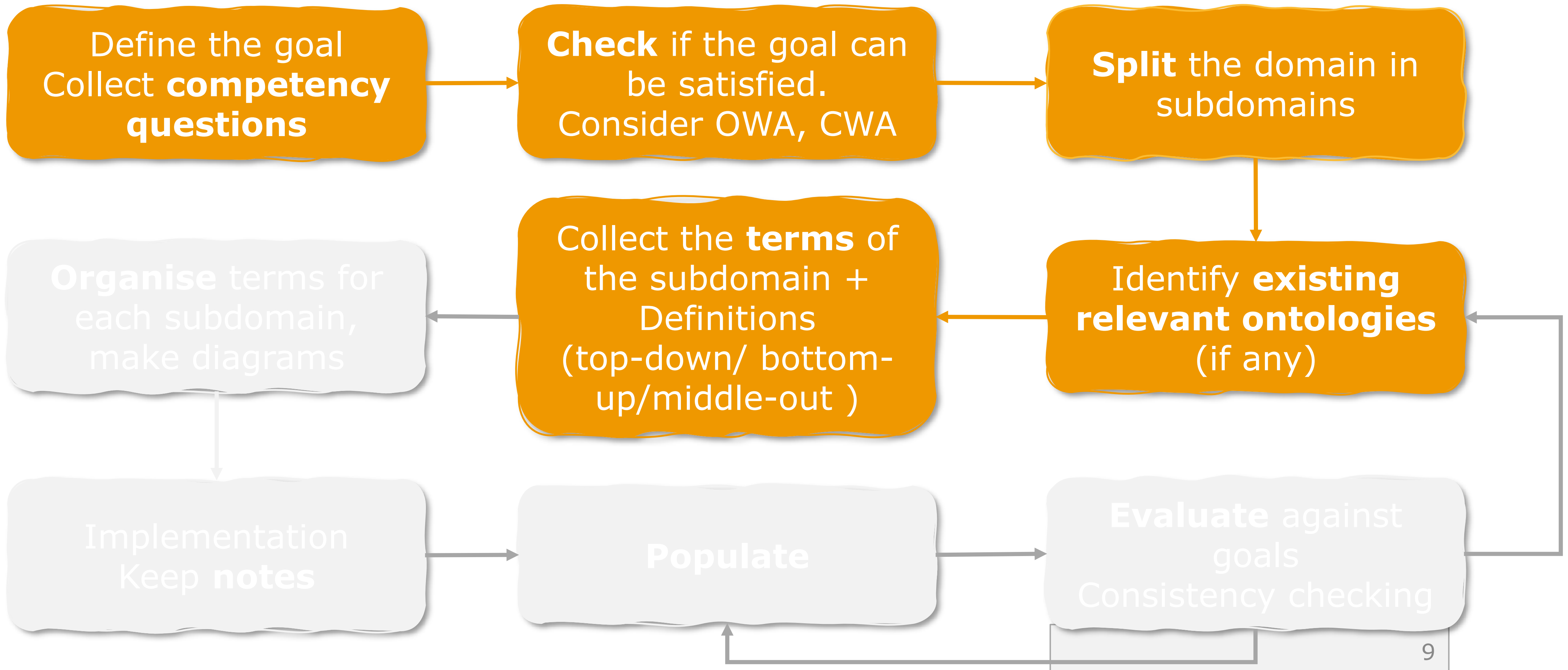
Ontology Development Pipeline


Define the goal
Collect **competency questions**
Check if the goal can be satisfied.
Consider OWA, CWA
Split the domain in subdomains
Identify **existing relevant ontologies** (if any)
Collect the **terms** of the subdomain + Definitions (top-down/ bottom-up/middle-out)
Organise terms for each subdomain, make diagrams
Implementation
Keep **notes**
Populate
Evaluate against goals
Consistency checking

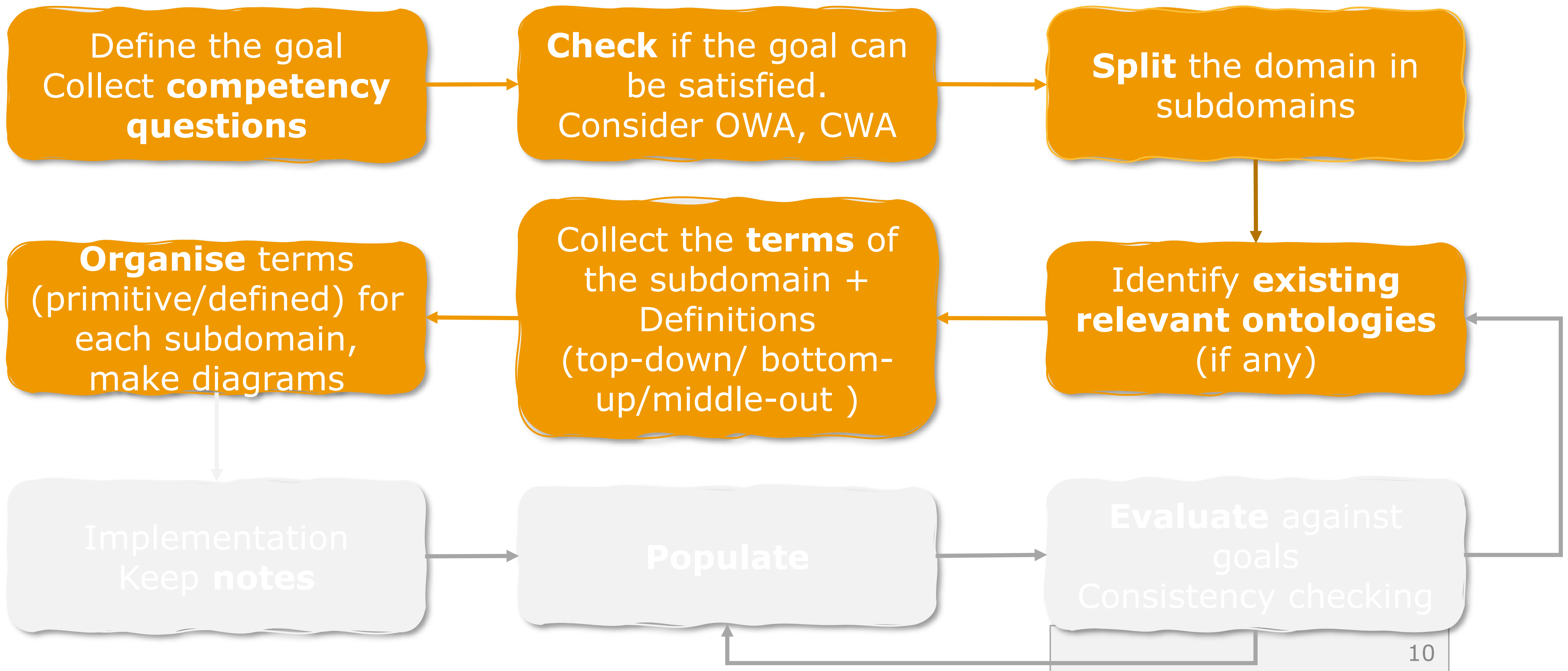
Ontology Development Pipeline


Define the goal
Collect **competency questions**
Check if the goal can be satisfied.
Consider OWA, CWA
Split the domain in subdomains
Identify **existing relevant ontologies** (if any)
Collect the **terms** of the subdomain + Definitions (top-down/ bottom-up/middle-out)
Organise terms for each subdomain, make diagrams
Implementation
Keep **notes**
Populate
Evaluate against goals
Consistency checking

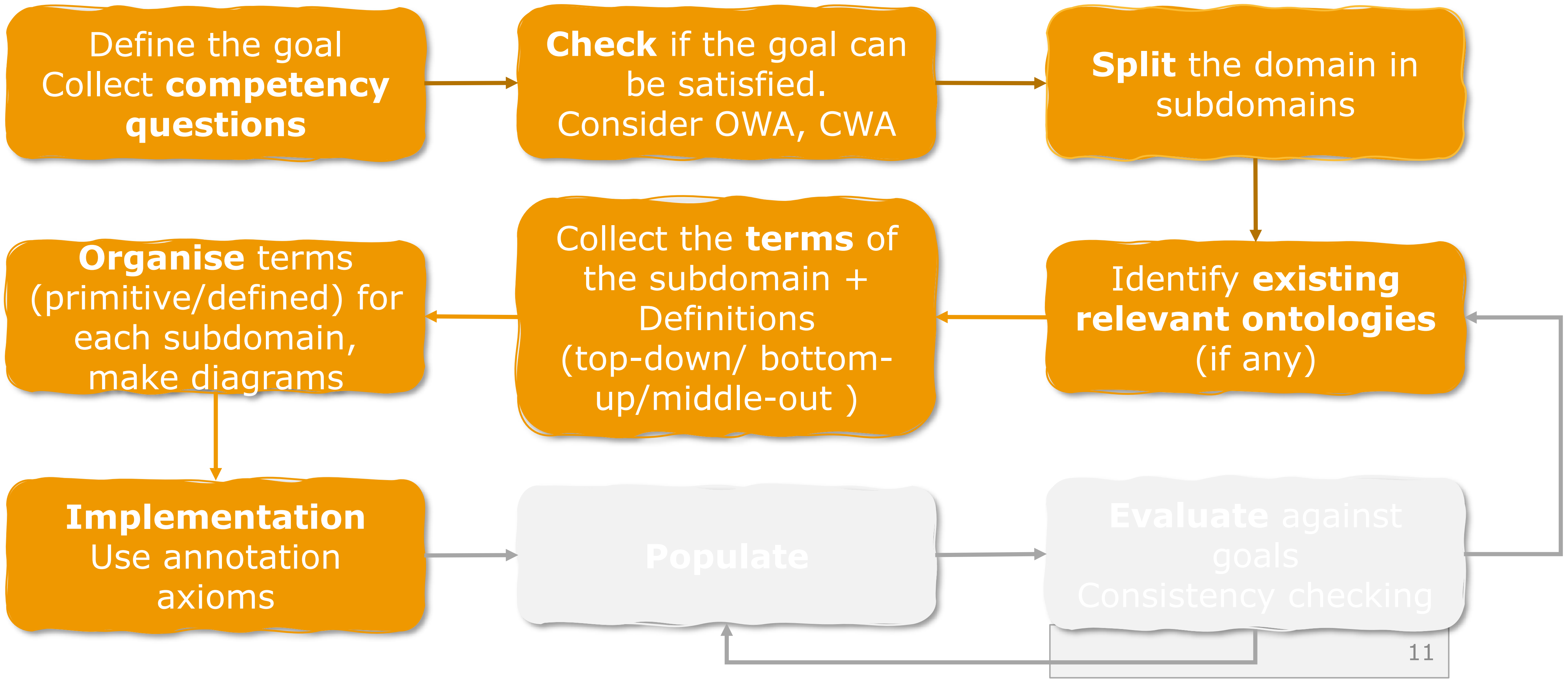
Ontology Development Pipeline


Define the goal
Collect **competency questions**
Check if the goal can be satisfied.
Consider OWA, CWA
Split the domain in subdomains
Identify **existing relevant ontologies** (if any)
Collect the **terms** of the subdomain + Definitions (top-down/ bottom-up/middle-out)
Organise terms for each subdomain, make diagrams
Implementation
Keep **notes**
Populate
Evaluate against goals
Consistency checking

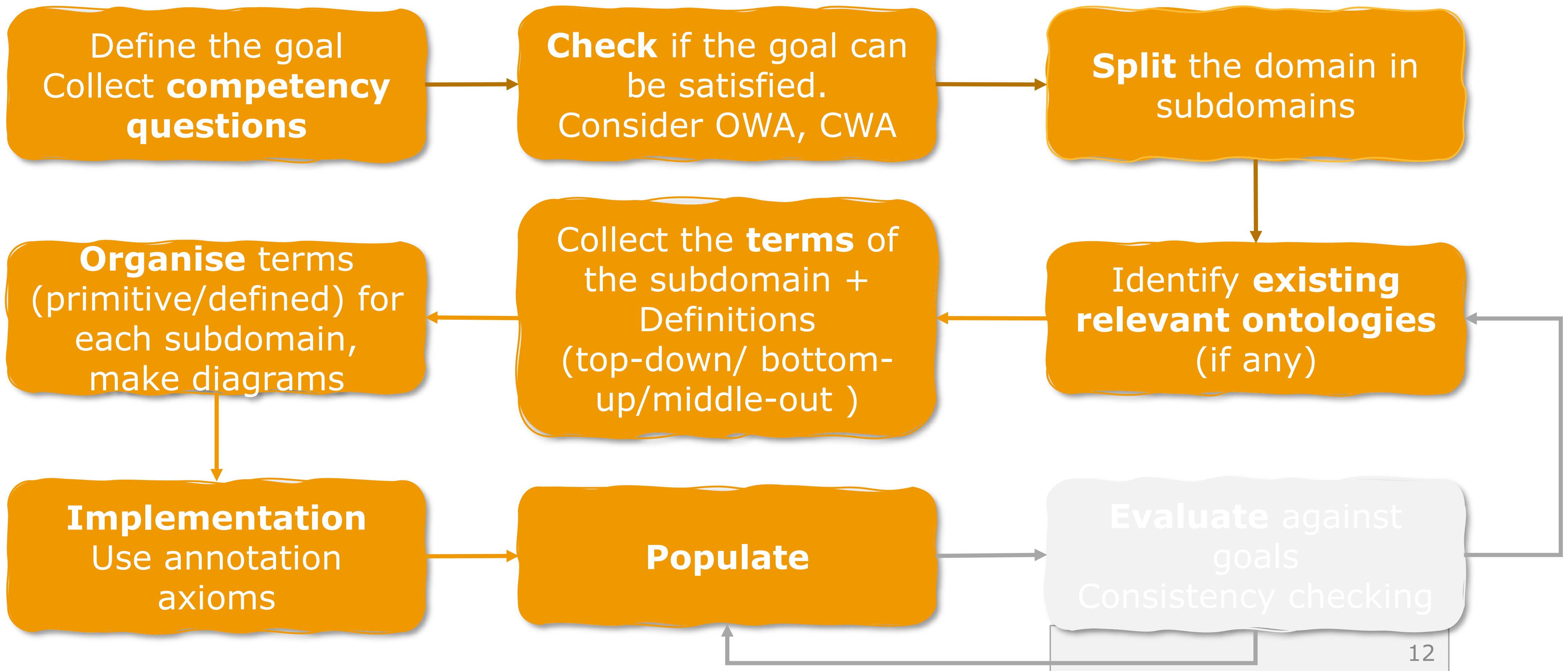
Ontology Development Pipeline


Define the goal
Collect **competency questions**
Check if the goal can be satisfied.
Consider OWA, CWA
Split the domain in subdomains
Identify **existing relevant ontologies** (if any)
Collect the **terms** of the subdomain + Definitions (top-down/ bottom-up/middle-out)
Organise terms (primitive/defined) for each subdomain, make diagrams
Implementation
Keep **notes**
Populate
Evaluate against goals
Consistency checking

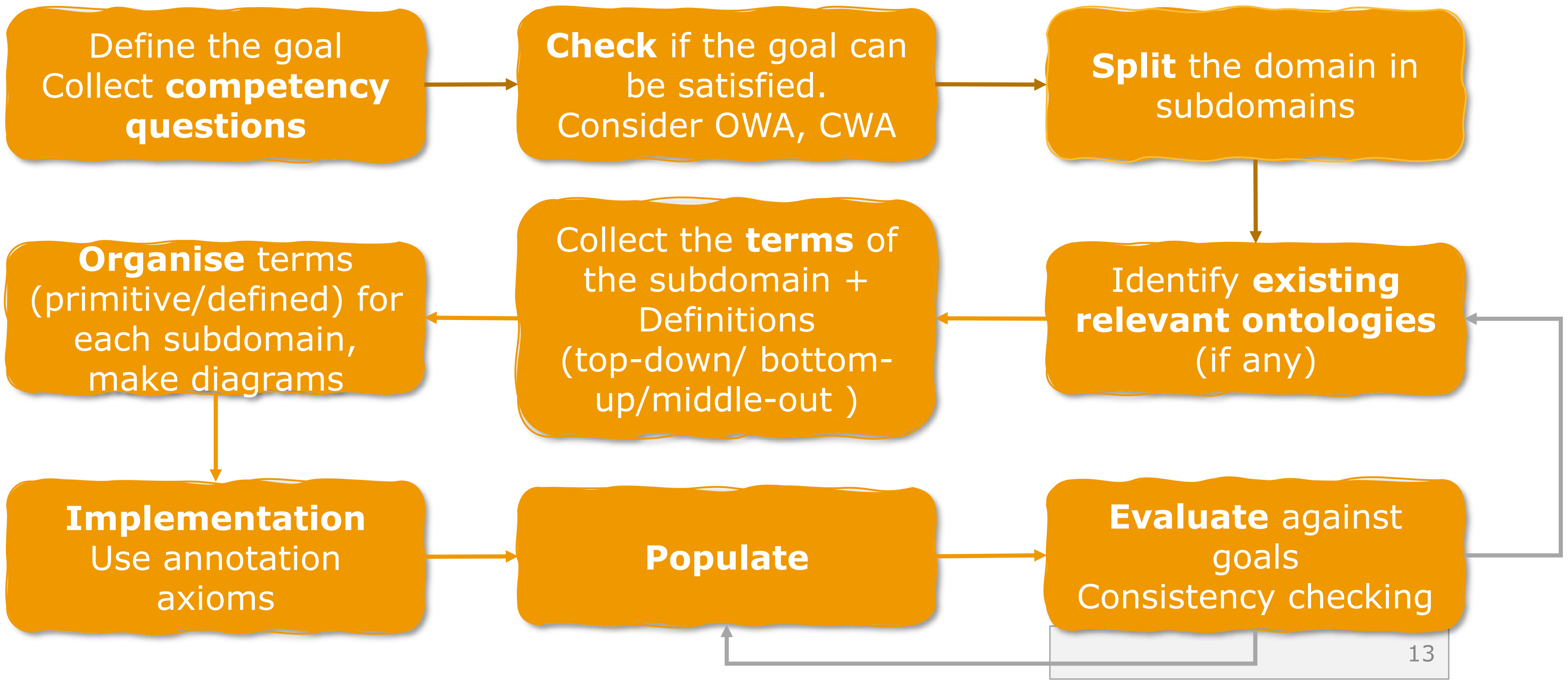
Ontology Development Pipeline


Define the goal
Collect **competency questions**
Check if the goal can be satisfied. Consider OWA, CWA
Split the domain in subdomains
Identify **existing relevant ontologies** (if any)
Collect the **terms** of the subdomain + Definitions (top-down/ bottom-up/middle-out)
Organise terms (primitive/defined) for each subdomain, make diagrams
Implementation
Use annotation axioms
Populate
Evaluate against goals
Consistency checking

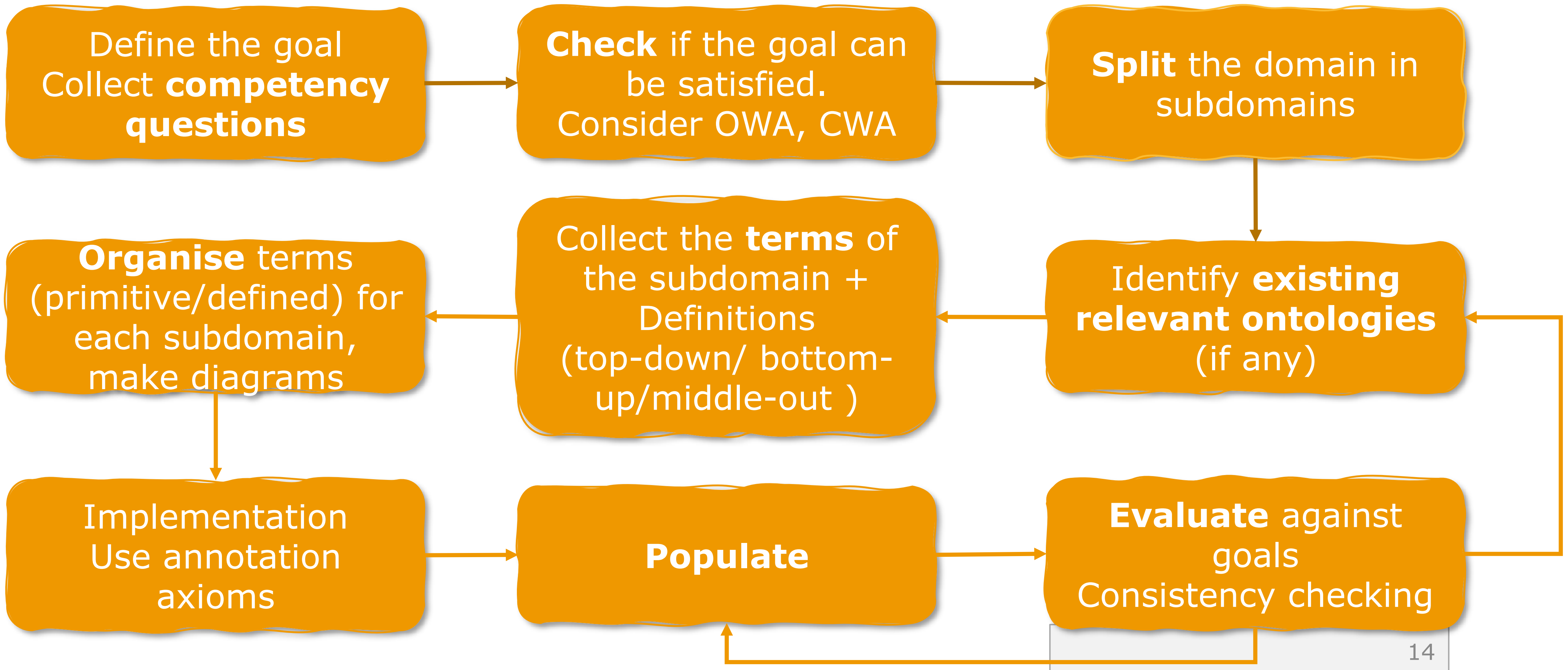
Ontology Development Pipeline


Define the goal
Collect **competency questions**
Check if the goal can be satisfied.
Consider OWA, CWA
Split the domain in subdomains
Identify **existing relevant ontologies** (if any)
Collect the **terms** of the subdomain + Definitions (top-down/ bottom-up/middle-out)
Organise terms (primitive/defined) for each subdomain, make diagrams
Implementation
Use annotation axioms
Populate
Evaluate against goals
Consistency checking

Ontology Development Pipeline


Define the goal
Collect **competency questions**
Check if the goal can be satisfied.
Consider OWA, CWA
Split the domain in subdomains
Identify **existing relevant ontologies** (if any)
Collect the **terms** of the subdomain + Definitions (top-down/ bottom-up/middle-out)
Organise terms (primitive/defined) for each subdomain, make diagrams
Implementation
Use annotation axioms
Populate
Evaluate against goals
Consistency checking

Ontology Development Pipeline


Define the goal
Collect **competency questions**
Check if the goal can be satisfied.
Consider OWA, CWA
Split the domain in subdomains
Identify **existing relevant ontologies** (if any)
Collect the **terms** of the subdomain + Definitions (top-down/ bottom-up/middle-out)
Organise terms (primitive/defined) for each subdomain, make diagrams
Implementation
Use annotation axioms
Populate
Evaluate against goals
Consistency checking

Class or Individual?

- Consider the following terms:

- Wine
- Red Wine
- Red Bordeaux Wine

Class or Individual?

- Can the class be instantiated?

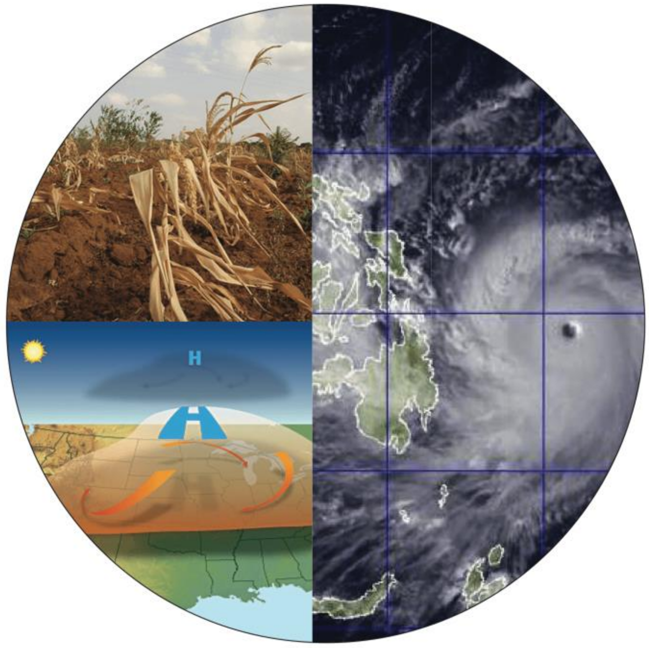
- Wine (class)
- Red Wine (class)
- Red Bordeaux Wine (class)
- Chateau Lafleur (class or individual?)

Class or Individual?

- **It depends on the goal of the ontology:**
- If the goal is to match wines with food:
  - Then an individual

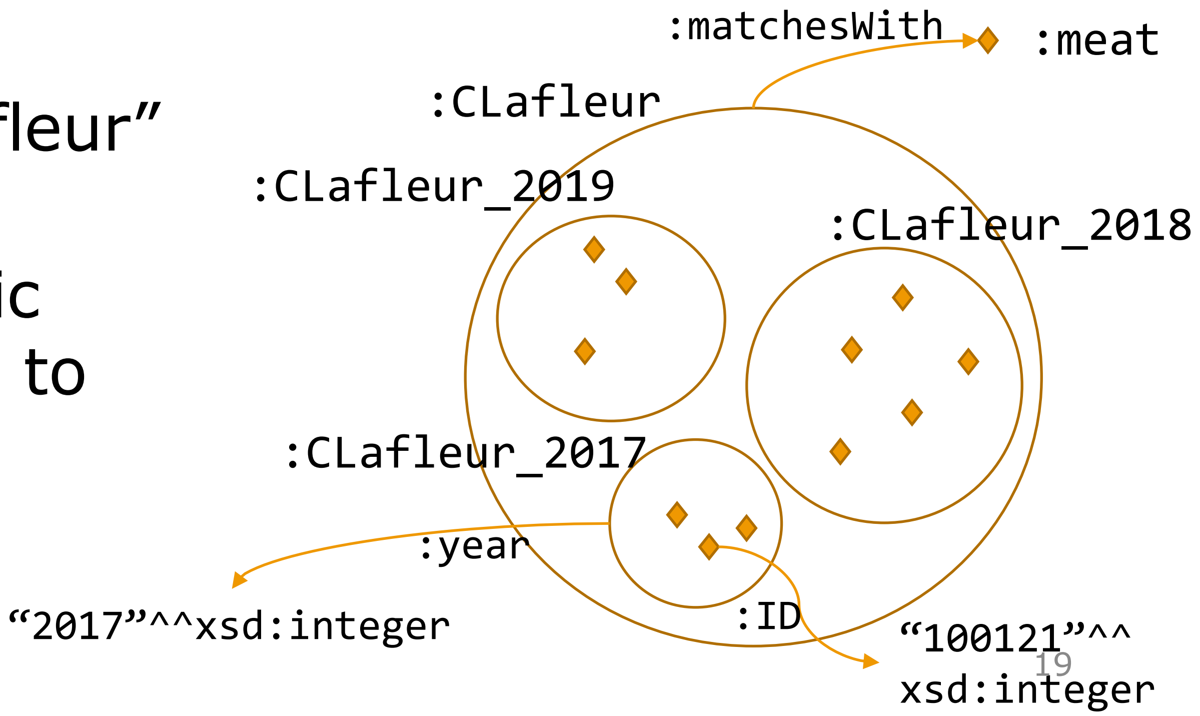
Class or Individual?

- **It depends on the goal of the ontology:**
- If the goal is to learn the weather conditions at the time of harvesting:
  - Then class
  - With what individuals?
  - 1993 Chateau Lafleur



Class or Individual?

- **It depends on the goal of the ontology:**
- If the goal is to cover the inventory of a wine cellar
  - Then class "1993 Chateau Lafleur" is a class
  - The individuals are the specific bottles in the cellar according to their ID numbers



Classes

- In OWL (as in DLs), we can distinguish two kinds of classes:
  - **Defined classes**
  - **Primitive classes**

Defined Classes

- A **defined class** is like an “if and only if” statement in logic.

```
(EquivalentClasses(CE1 ... CEn))
```

- **Example:** A driver can be defined to be exactly “a person who drives a vehicle”.
- With a defined class, we give necessary and sufficient conditions for membership in a class.
- Thus a defined class allows **deduction in two directions**. For example:
  - If someone is a driver, then he/she is a person and he/she drives a vehicle.
  - If someone is a person and he/she drives a vehicle, then he/she is a driver.

Primitive Classes

- A **primitive class** includes only necessary (but not sufficient conditions) for membership.

```
(SubClassOf(CE1 CE2))
```

- **Example:** It is hard to define a dog (or any other natural kind). However, we might want to say:
  - Among other things, a dog is something that eats bones.

- In contrast to defined classes, primitive classes support deductions in only one direction. For example:
  - If something is a dog, then we can infer that it eats bones.

Determining whether a class is defined or primitive

- **Defined**
  - The complete definition of the class is known and relevant.
  - When one wants **the system** to determine class membership (well, if we do not want to do this, why use OWL?).
- **Primitive**
  - They are usually found near the top of a generalization hierarchy while **defined classes** typically appear as we move further down by specializing general classes with various restrictions.

Definitional vs. Incidental Properties

- It is important to **distinguish between a class' true definition and any incidental properties**.

- **Examples:**
- Red Bordeaux wines are always dry.
- Red Bordeaux wines are red and from Bordeaux

- But the property of being dry is certainly not a part of the definition of the class `RedBordeauxWine` (only the color and the region define a wine to be a Red Bordeaux).

Definitional vs. Incidental Properties (cont’d)

- In OWL, incidental properties are asserted using **extra subclass axioms** (in addition to the axioms that define the class).

Defining Classes

- Decide for each class whether a restriction should be taken as:
  - Part of the meaning of a class (and thus participate in classification) (definitional property)
    - Bordeaux wine is made in Bordeaux
  - Derived property to be inferred once class membership is known (incidental property)
    - Bordeaux wine is made of grapes

Classes or properties?

- Usually:
  - **Nouns** as **classes**
  - **Verbs** as **properties**

- Example:
- A **Bordeaux wine** is any **wine produced** in the **Bordeaux region of France**
- **Exercise:** How would *you* model this?

Classes or properties?

- Usually:
  - **Nouns** as **classes**
  - **Verbs** as **properties**

- Example:
- A **Bordeaux wine** is any **wine** **produced in** the **Bordeaux** **(is)** **region of** **France**

Classes or properties?

- Usually:
  - **Nouns** as **classes**
  - **Verbs** as **properties**
- What about:
  - Cabernet Franc is a **grape**
  - Chateau Lafleur has **grape** Cabernet Franc
- Class or property?

Classes or properties?

- Can the description **stand on its own** without implying an unmentioned object related to the object in question?
  - If yes: class
  - Otherwise: property.

  - Cabernet Franc is a **grape**
  - Chateau Lafleur has **grape** Cabernet Franc

Classes or properties?

- Can the description **stand on its own** without implying an unmentioned object related to the object in question?
  - If yes: class
  - Otherwise: property.

  - Cabernet Franc is a **grape**
  - Chateau Lafleur **has grape** Cabernet Franc
- If the term should play both roles then we can use the prefix “has”

Things to Remember (from Ontology 101 tutorial)

- There is never a single correct way to model a domain— there are always **viable alternatives.**
- The best solution almost always depends on the **application** that you have in mind and the **extensions** that you anticipate.
- Ontology development is necessarily an **iterative process**.

Example: Purpose & scope of the animals ontology

- **To provide an ontology for an index of a children’s book of animals including**
  - Where they live
  - What they eat
    - Carnivores, herbivores and omnivores
  - How dangerous they are
  - How big they are
  - A bit of basic anatomy
    - numbers of legs, wings, toes, etc.

Organise the concepts
Example: Animals & Plants

- Dog
- Cat
- Cow
- Person
- Tree
- Grass
- Herbivore
- Male
- Female

- Carnivore
- Plant
- Animal
- Fur
- Child
- Parent
- Mother
- Father

- Healthy
- Pet
- Domestic Animal
- Farm animal
- Draft animal
- Food animal
- Fish
- Carp
- Goldfish

Organise the concepts
Example: Animals & Plants

- Dog
- Cat
- Cow
- Person
- Tree
- Grass
- Herbivore
- Male
- Female

- Carnivore
- Plant
- Animal
- Fur
- Child
- Parent
- Mother
- Father

- Healthy
- Pet
- Domestic Animal
- Farm animal
- Draft animal
- Food animal
- Fish
- Carp
- Goldfish

Organise the concepts
Example: Animals & Plants

- Dog
- Cat
- Cow
- Person
- Tree
- Grass
- Herbivore
- Male
- Female

- Carnivore
- Plant
- Animal
- Fur
- Child
- Parent
- Mother
- Father

- Healthy
- Pet
- Domestic Animal
- Farm animal
- Draft animal
- Food animal
- Fish
- Carp
- Goldfish

Extend the concepts “Laddering”

- Take a group of things and ask what they have in **common**
  - Then what other ‘siblings’ there might be
- e.g.
  - Plant, Animal → **Living Thing**
    - Might add Bacteria and Fungi but not now
  - Cat, Dog, Cow, **Person → Mammal**
    - Others might be Goat, Sheep, Horse, Rabbit,…
  - Cow, Goat, Sheep, Horse → **Hoofed animal** (“Ungulate”)
    - What others are there? Do they divide amongst themselves?
  - Wild, Domestic → **Domestication**
    - What other states – “Feral” (domestic returned to wild)

Extend the concepts
“Laddering”

- Why do we do this?
- This way we can define the properties of the most generic classes and their subclasses will inherit these properties

Choose some main axes

- Add abstractions where needed
  - e.g. “Living thing”
- Identify relations
  - e.g. “eats”, “owns”, “parent of”
- Identify definable things
  - e.g. “child”, “parent”, “Mother”, “Father”
    - Things where you can say clearly what it means
      - Try to define a dog precisely – very difficult
        - » A “natural kind”
- Make names explicit (start naming things properly)

Naming conventions

- **Choose a naming convention and stick to it!**
- **Issues:**
  - Capitalization and delimiters (e.g., `WildAnimal vs. Wild-Animal vs. Wild Animal`)
  - Singular or plural (e.g., `Wine vs. Wines`)
  - Prefix and suffix conventions (e.g., `has-father, father-of`)
  - Do **not** add strings such as “class” or “property” to names of classes or properties (e.g, WineClass).
  - Avoid abbreviations to enhance readability
  - If you prefer to use the name of a class (e.g., `Animal`) in the name of a direct subclass (e.g., `Wild Animal`), use it consistently for all subclasses.

Choose some main axes
Add abstractions where needed; identify relations;
Identify definable things, make names explicit

- Living Thing
  - Animal
    - Mammal
      - Cat
      - Dog
      - Cow
      - Person
    - Fish
      - Carp
      - Goldfish
  - Plant
    - Tree
    - Grass
    - Fruit

- Modifiers*
  - domestic
    - pet
    - Farmed
      - Draft
      - Food
  - Wild
  - Health
    - healthy
    - sick
  - Sex
    - Male
    - Female
  - Age
    - Adult
    - Child

- Relations
  - eats
  - owns
  - parent-of
  - …
- Definable
  - Carnivore
  - Herbivore
  - Child
  - Parent
  - Mother
  - Father
  - Food Animal
  - Draft Animal

Self_standing_entities

- Things that can exist on their own
  - People, animals, houses, actions, processes, …
    - Roughly nouns
- Modifiers
  - Things that modify (“inhere”) in other things (e.g., domestic animal)
    - Roughly adjectives and adverbs

Identify the domain and range constraints for *properties*

- Animal *eats* Living_thing
  - *eats* domain: Animal;
    range: Living_thing
- Person *owns* Living_thing except person
  - *owns* domain: Person
    range: Living_thing & not Person
- Living_thing *parent_of* Living_thing
  - *parent_of*: domain: Animal
    range: Animal

If anything is used in a special way, add a text comment

- Animal *eats* Living_thing — ***ignore difference between parts of living things and living things also derived from living things***
  - *eats* domain: Animal;
    range: Living_thing

For definable things

- Paraphrase and formalise the definitions in terms of the primitives, relations and other definables.

- Note any assumptions to be represented elsewhere.
  - Add as comments when implementing

- *"A 'Parent' is an animal that is the parent of some other animal" (Ignore plants for now)*
  - Parent isEquivalentTo Animal and *parent_of* some Animal

- *"A 'Herbivore' is an animal that eats only plants"*

  - Herbivore isEquivalentTo Animal and *eats* only Plant

- "An 'omnivore' is an animal that eats both plants and animals"
  - Omnivore isEquivalentTo Animal and *eats* some Animal and *eats* some Plant

Which properties can be filled in at the class level now?

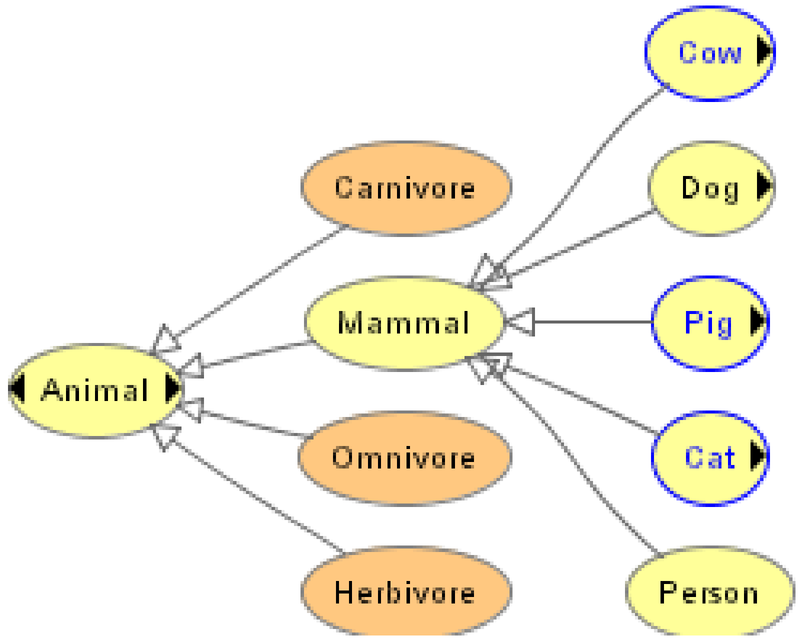
- What can we say about *all* members of a class?
  - *eats*
    - *All cows eat some plants*
    - *All cats eat some animals*
    - *All pigs eat some animals* &
      eat some plants

Check with reasoner

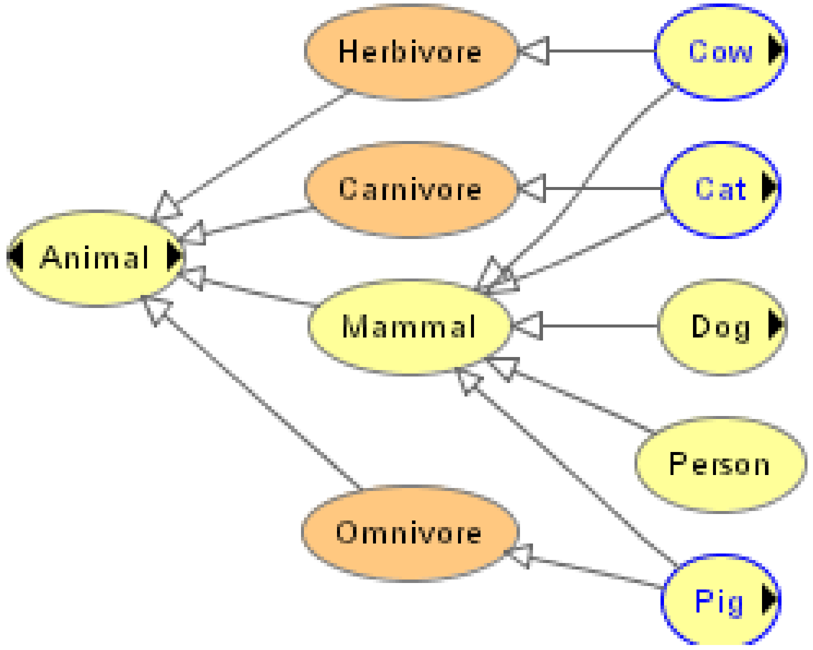
- Cows should be Herbivores
  - Are they? why not?
    - What have we said?
      - Cows are animals and, *amongst other things*,
        eat *some* grass and
        eat some leafy_plants
    - What do we need to say:
      Closure axiom
      - Cows are animals and, *amongst other things*,
        eat *some* plants and eat *only* plants
        - » (See “Vegetarian Pizzas” in OWL tutorial)

Normalisation: From Trees to DAGs

- Before classification
  - A tree



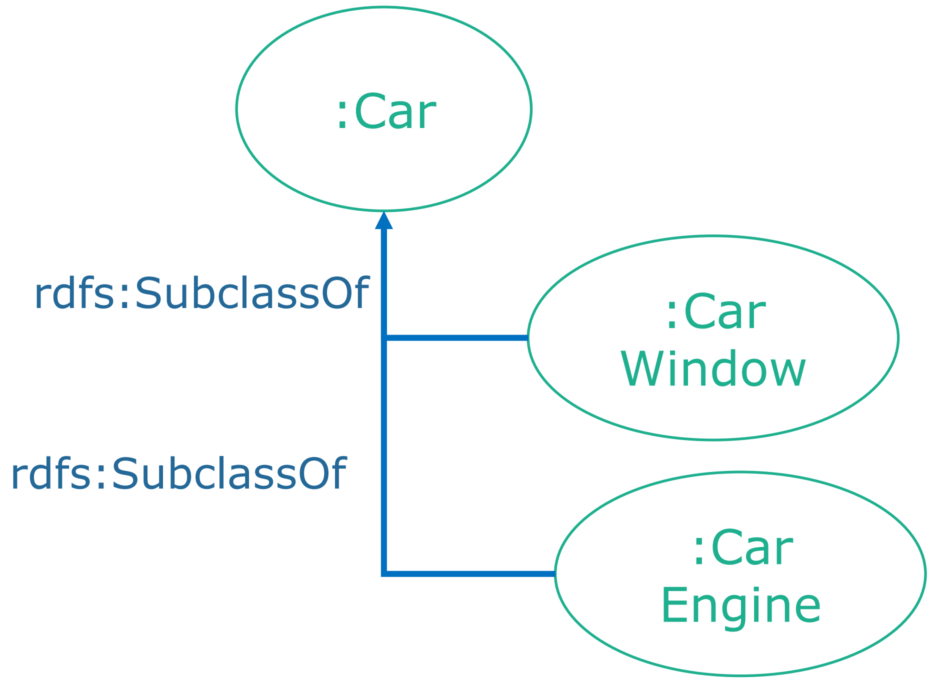
- After classification
  - A DAG
    - **D**irected **A**cyclic **G**raph



Common mistakes in OWL

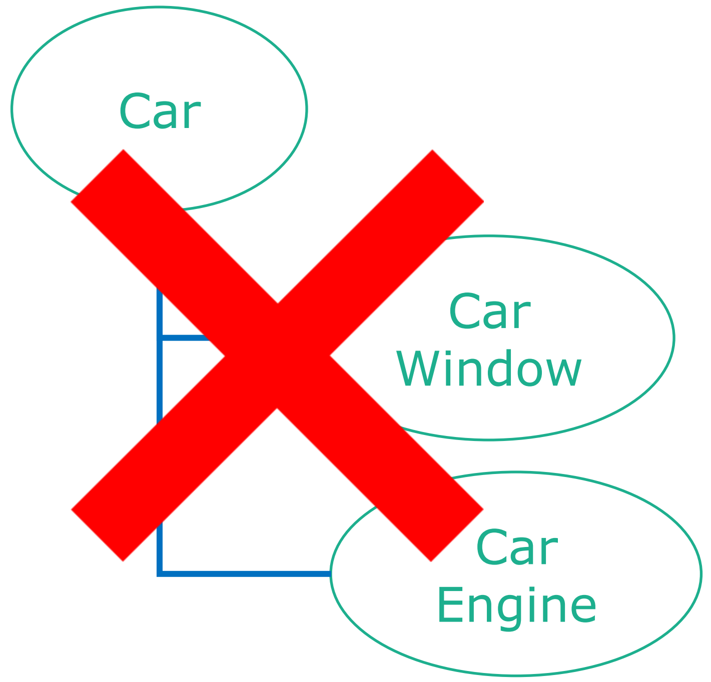
- Partonomic vs Subclass relationships
- Forgetting to make classes disjoint
- Open World Assumption
- Unique Name Assumption
- Domain and range definitions

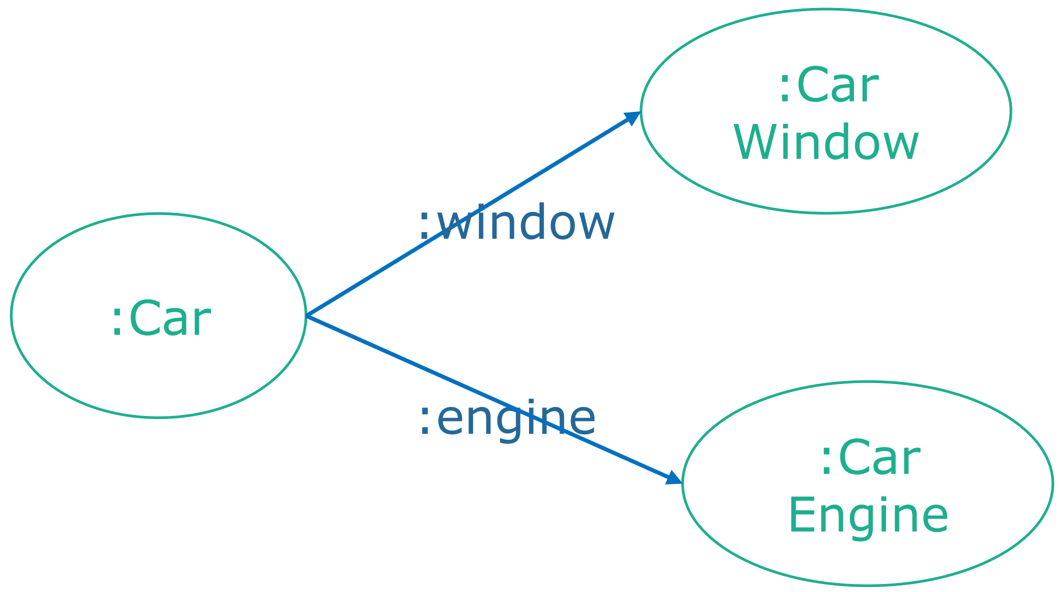
Partonomic vs Subclass relationships



Exercise: Is this correct?

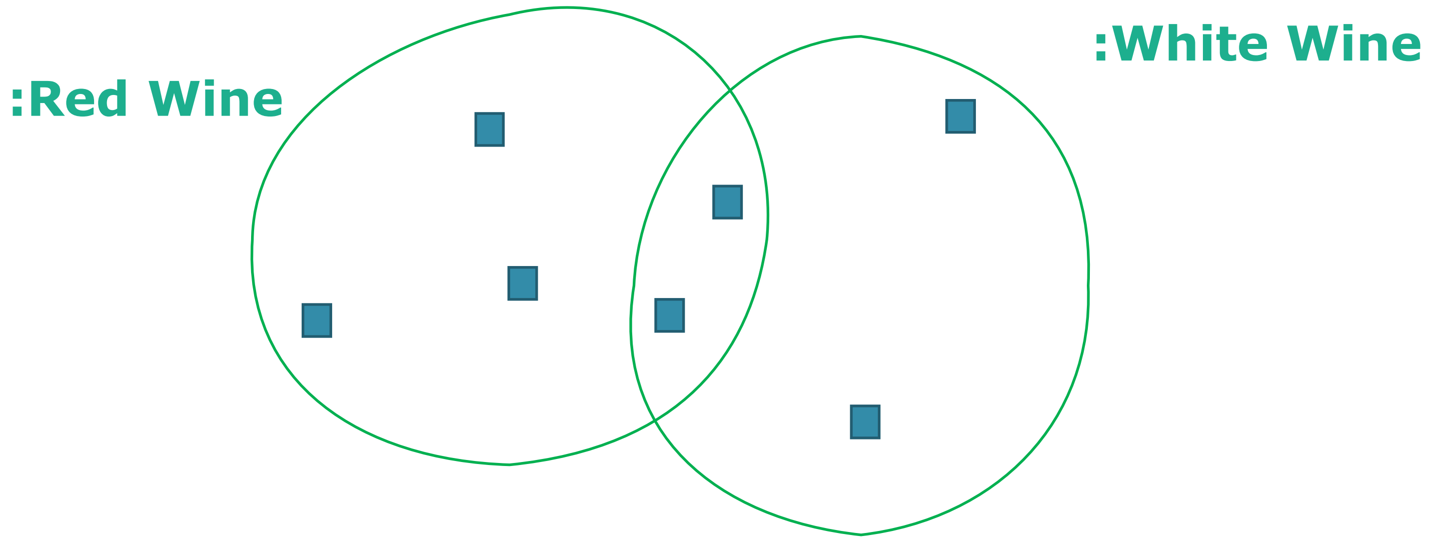
Partonomic vs Subclass relationships



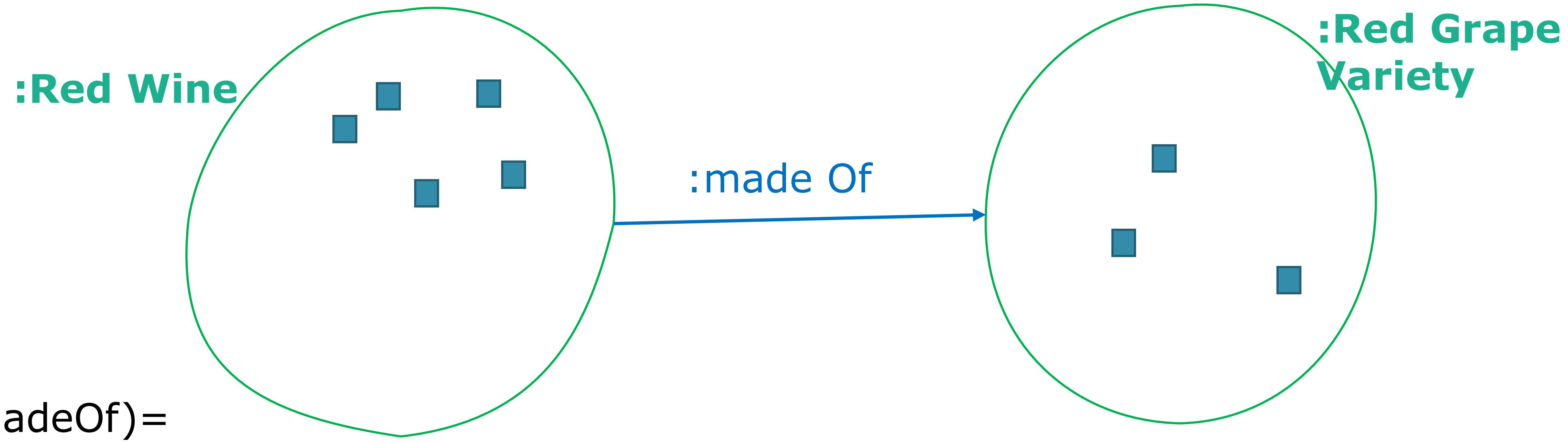


Disjoint Classes

- OWL Classes are assumed to overlap
- It is safer to explicitly state that two classes are disjoint

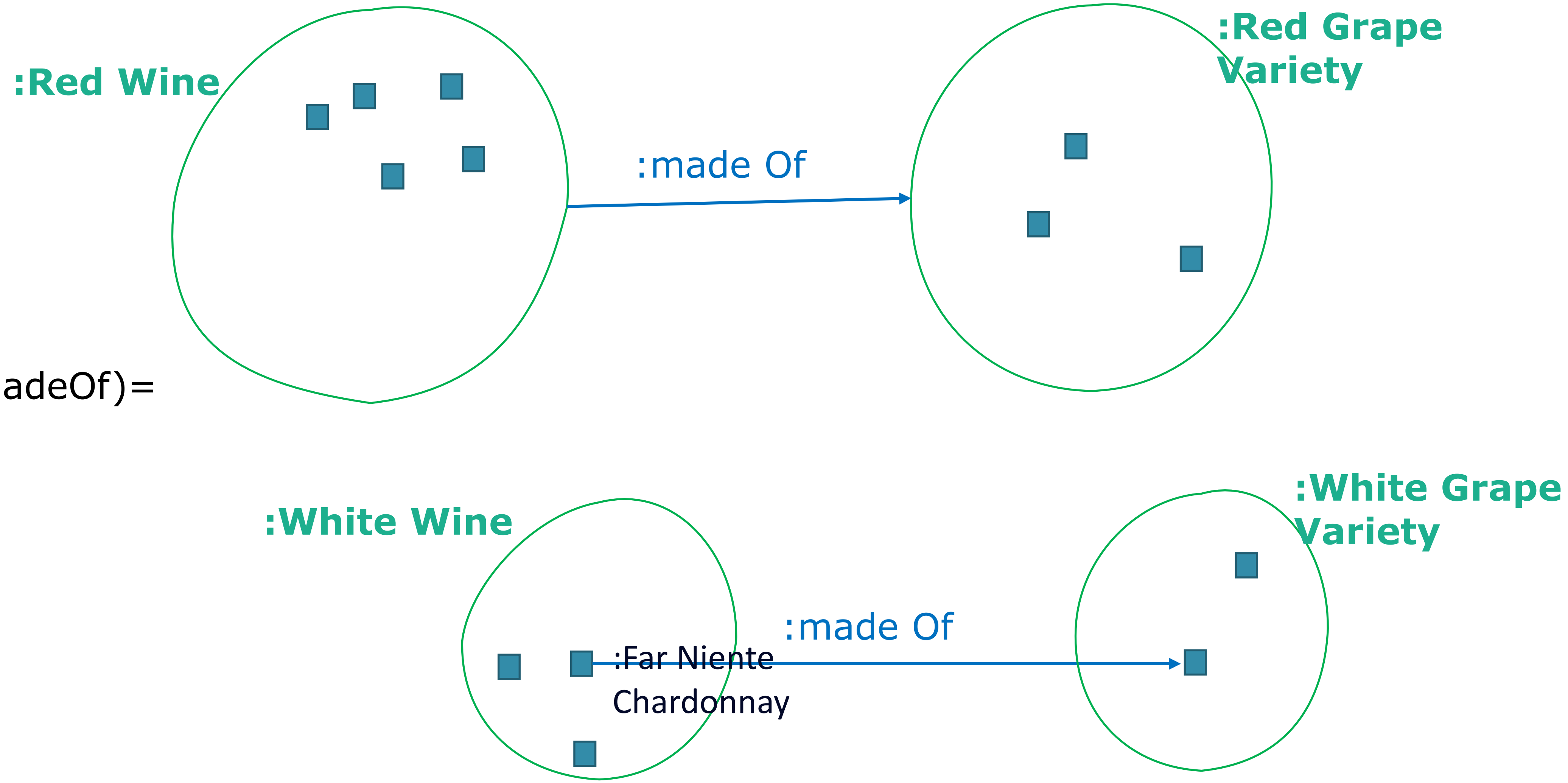


Disjoint Classes



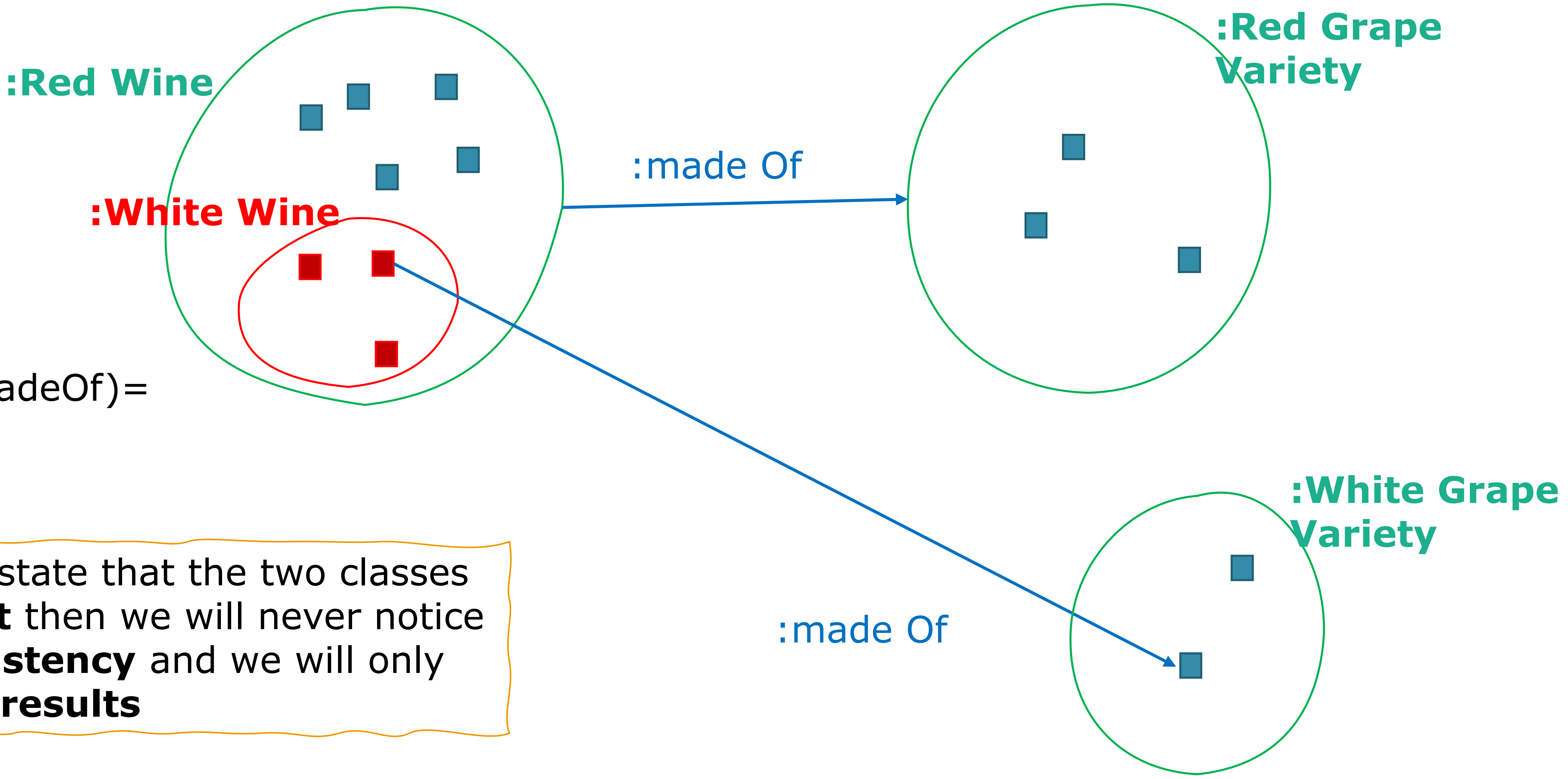
Domain(:madeOf)=
:RedWine

Disjoint Classes



Domain(:madeOf)= :RedWine

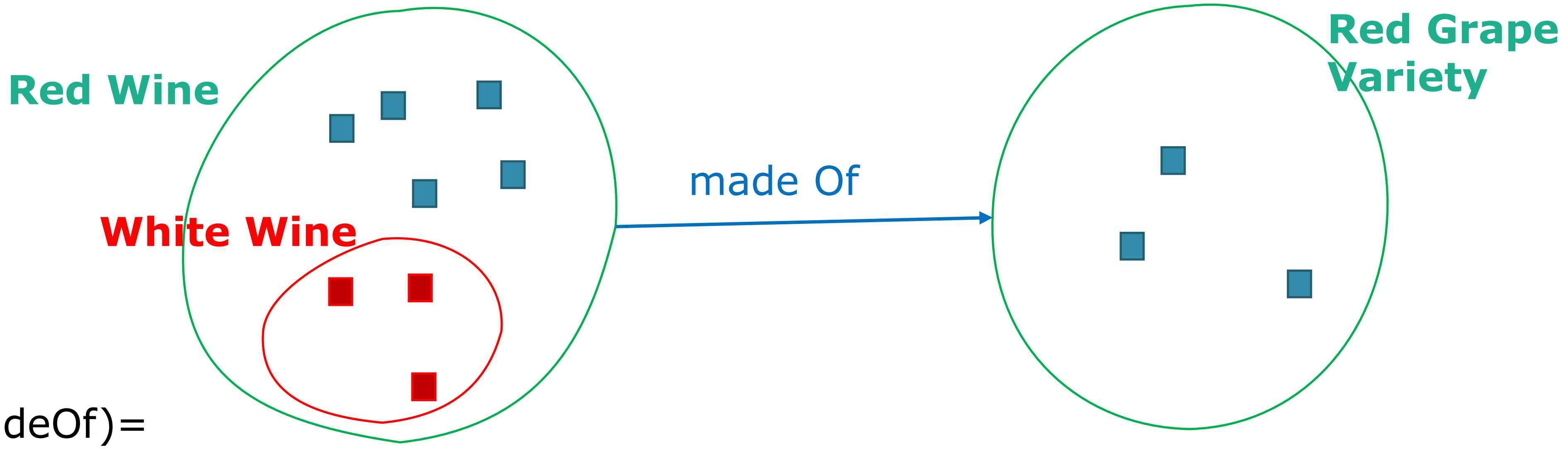
Disjoint Classes



Domain(:madeOf)= :RedWine

If we don't state that the two classes are **disjoint** then we will never notice the **inconsistency** and we will only get **wrong results**

Disjoint Classes



Domain(madeOf)=
RedWine

If we don’t state that the two classes are **disjoint** then we will never notice the **inconsistency** and we will only get **wrong results**

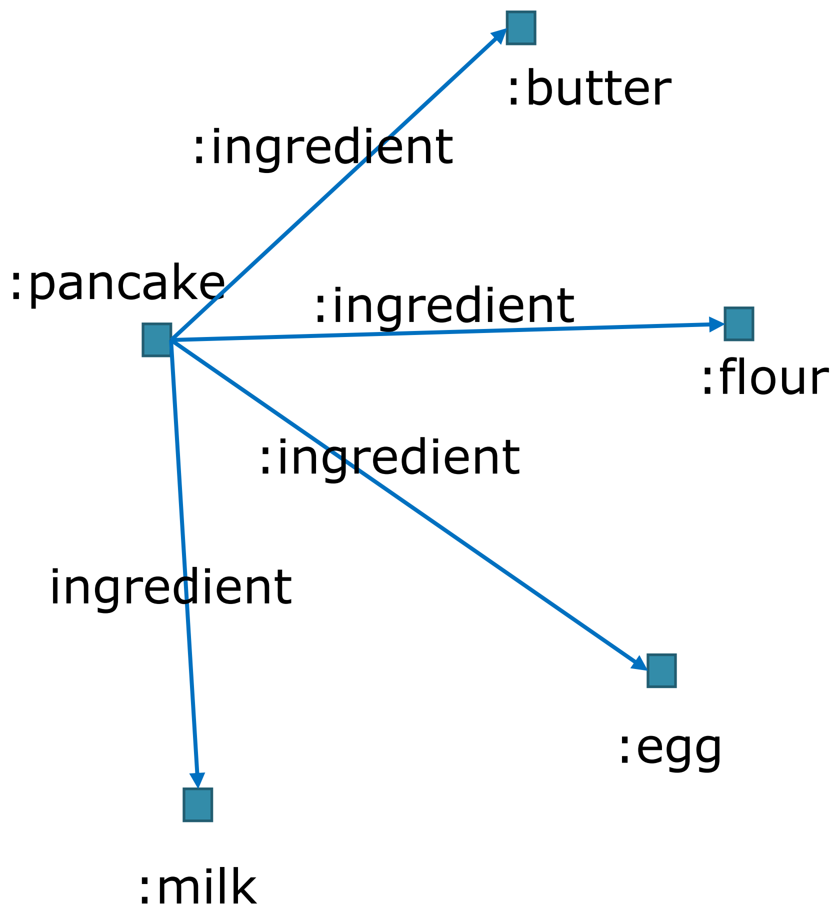
Need to be careful when defining domains and ranges
They are not constraints

Open World Assumption

- OWA assumes **incomplete** knowledge by default
- We assume that our model is going to be **reused** and **extended**
- **Everything can be true unless proven otherwise**
- OWA is good for designing knowledge in an **extensible** way

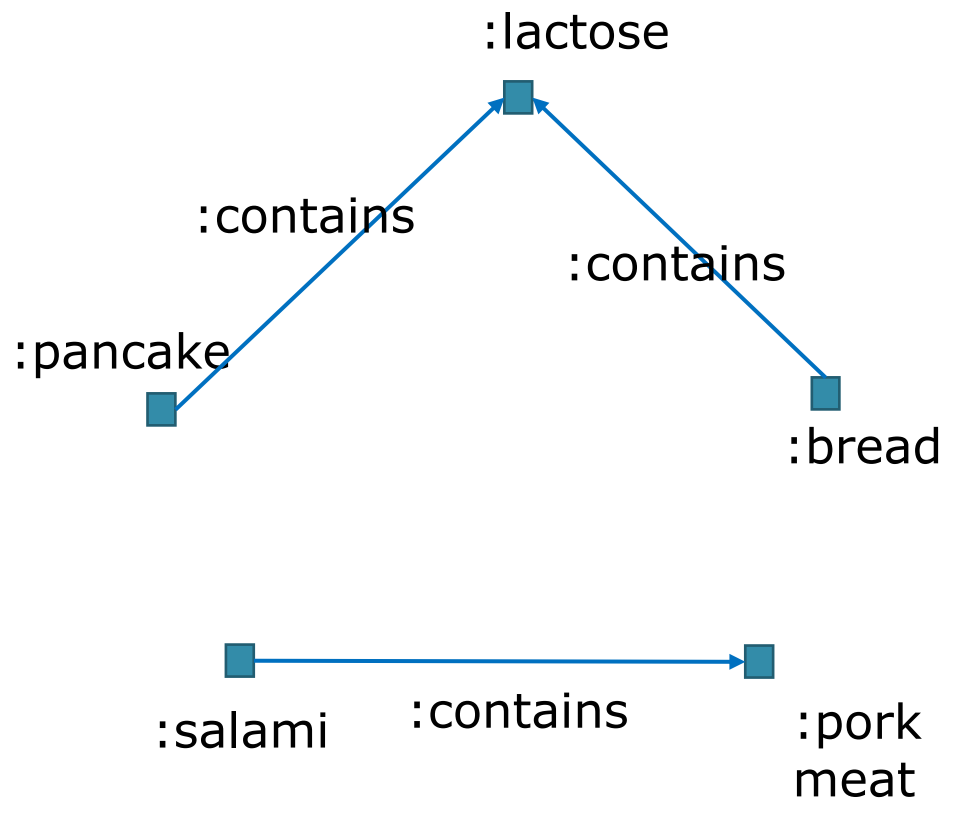
Open World Assumption

- Does the pancake have more than 3 ingredients?
- Yes
- Does the pancake have less than 3 ingredients?
- No
- Does the pancake have more than 5 ingredients?
- Unknown



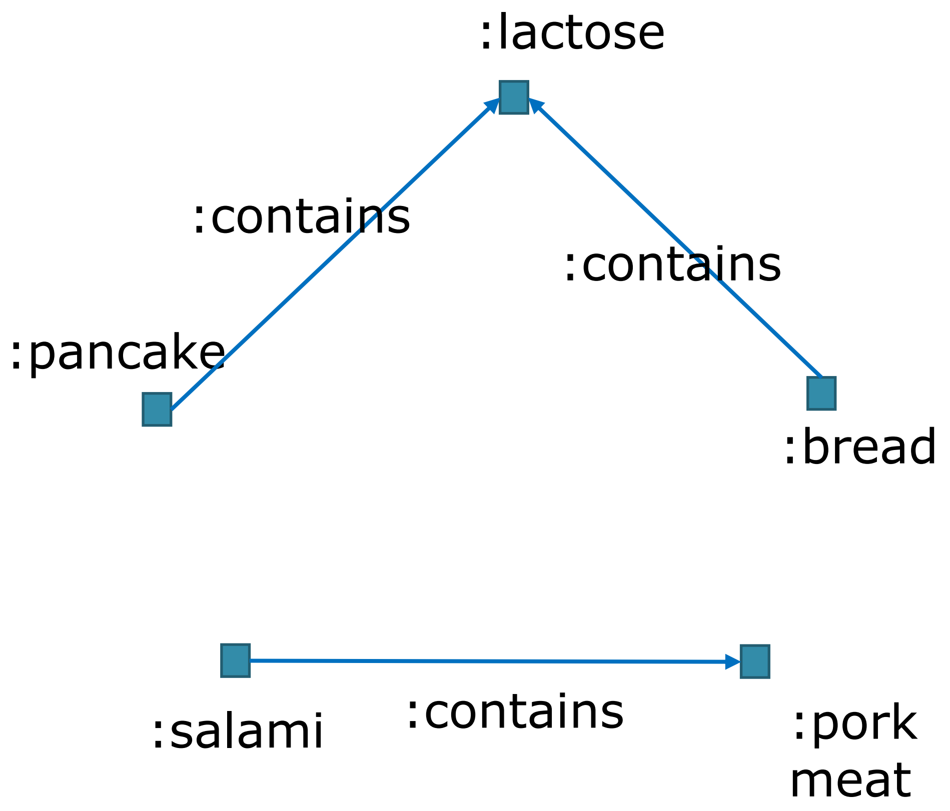
Exercise: Open World Assumption

- I want a lactose-free food
- DB: ?
- KB: ?



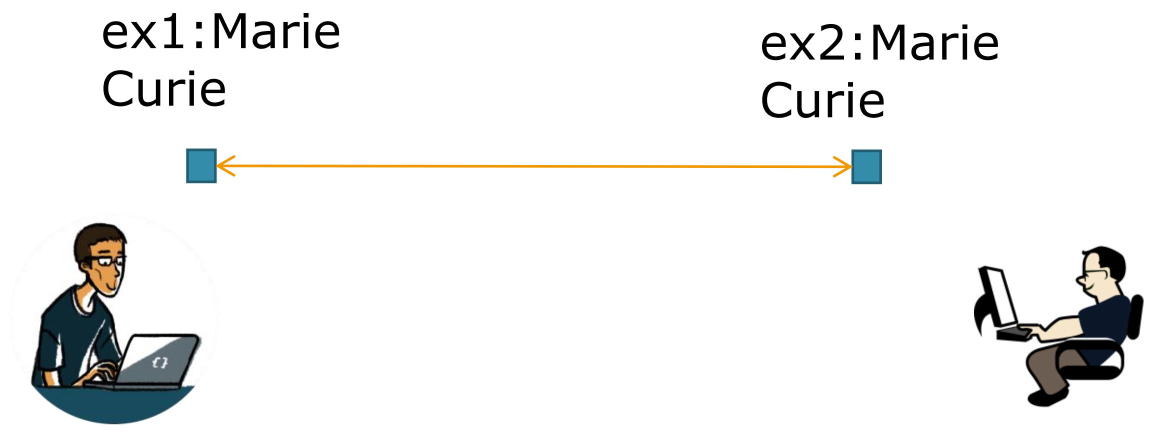
Solution

- I want a lactose-free food
- DB: salami
- KB:
- Unless it is defined that:
  - pork meat is lactose-free and
  - salami contains **only** pork meat



Unique Name Assumption

- OWL does **not** make the UNA
- Because later we may find that two individuals are the same
- We need to define this explicitly



Resources

- Various Ontology Development methodologies:
  - Olszewska JI, Houghtaling M, Gon ̧calves PJS, Fabiano N, Haidegger T, Carbonera JL, Patterson WR, Ragavan SV, Fiorini SR, Prestes E (2020) Robotic standard development life cycle in action. J Intell Robotic Syst 98(1):119–131
  - Peroni S (2016) A simplified agile methodology for ontology development. In: OWL:and Directions–Reasoner Evaluation, Springer, pp 55–69
  - Fernandez-Lopez M, Gomez-Perez A, Juristo N (1997) Methontology: From ontological art towards ontological engineering. In: AAAI 1997
  - Neches R (1993) Building large knowledge-based systems: Representation and inference in the CYC project: D.b. lenat and r.v. guha. Artificial Intelligence 61(1):65–7
- Helpful Guide:
  - A Practical Guide to Building OWL Ontologies, Using Protégé 5.5 and Plugins, Edition 3.2, 8 October 2021, Michael DeBellis: https://drive.google.com/file/d/1A3Y8T6nIfXQ_UQOpCAr_HFSCwpTqELeP/view
- Some Theory:
  - Sebastian Rudolph (2011), Foundations of Description Logics (https://www.aifb.kit.edu/images/1/19/DL-Intro.pdf)