COMP318: Ontology based Information Systems

www.csc.liv.ac.uk/~valli/Comp318

Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

V.Tamma@liverpool.ac.uk

Where were we

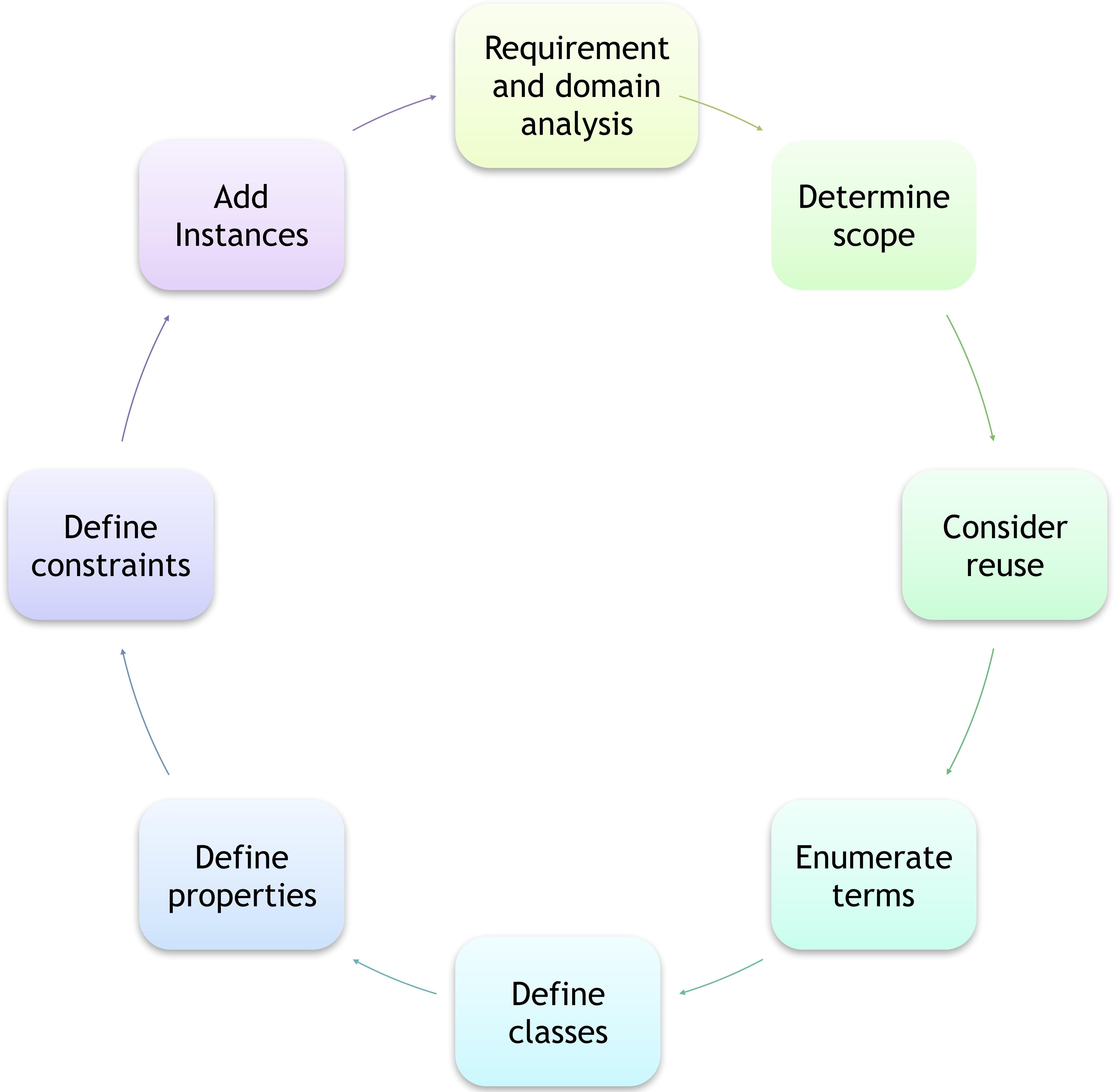
- Motivation behind the need for ontologies
- Ontology engineering
  - Principles
  - Methodologies

Ontology 101

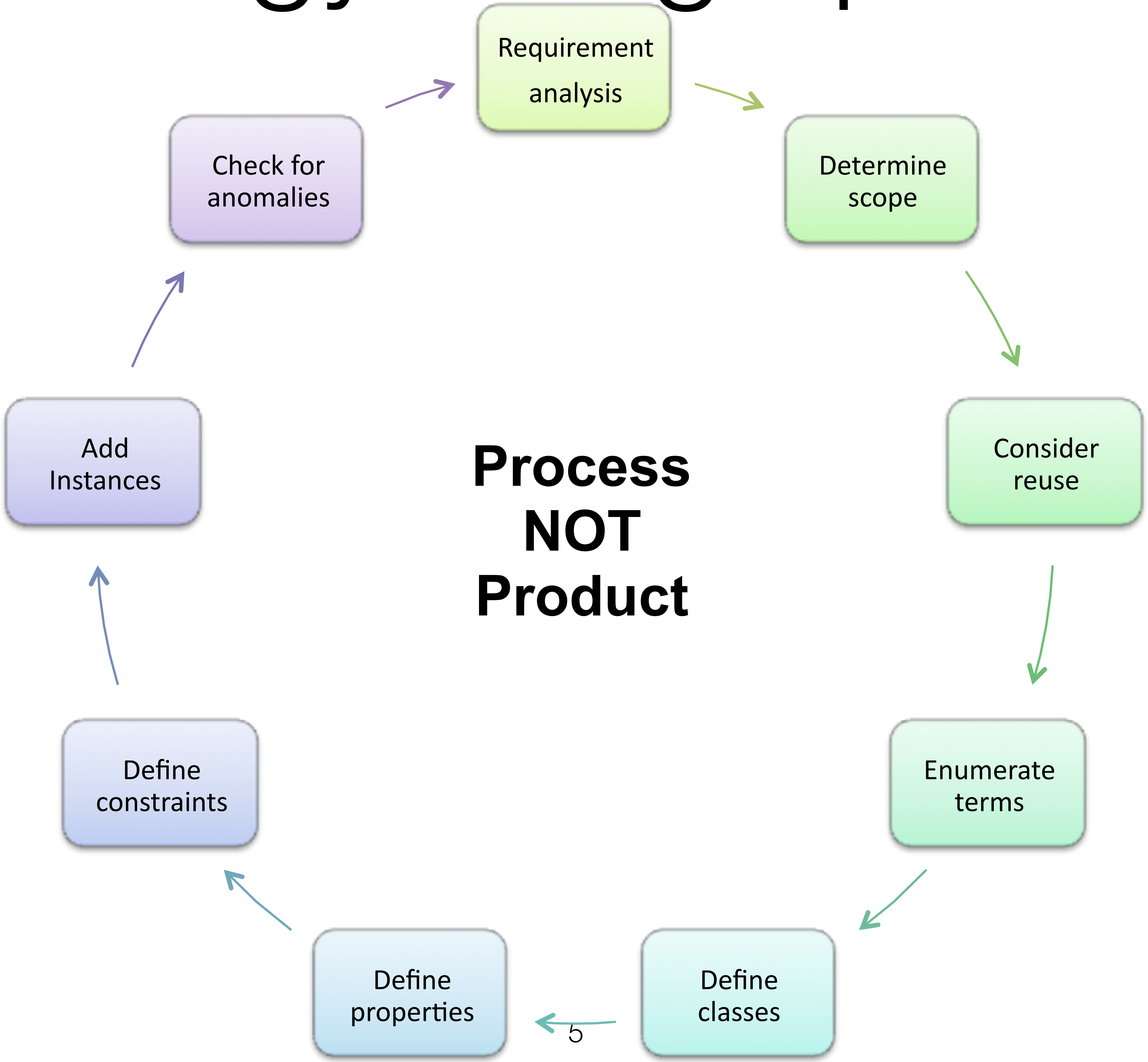
Requirement and domain analysis → Determine scope → Consider reuse → Enumerate terms → Define classes → Define properties → Define constraints → Add Instances

But really more like...

- An iterative Process that repeats continuously and improves the ontology
  - there are always different approaches for modelling an ontology
  - in practice the designated application decides about the modelling approach

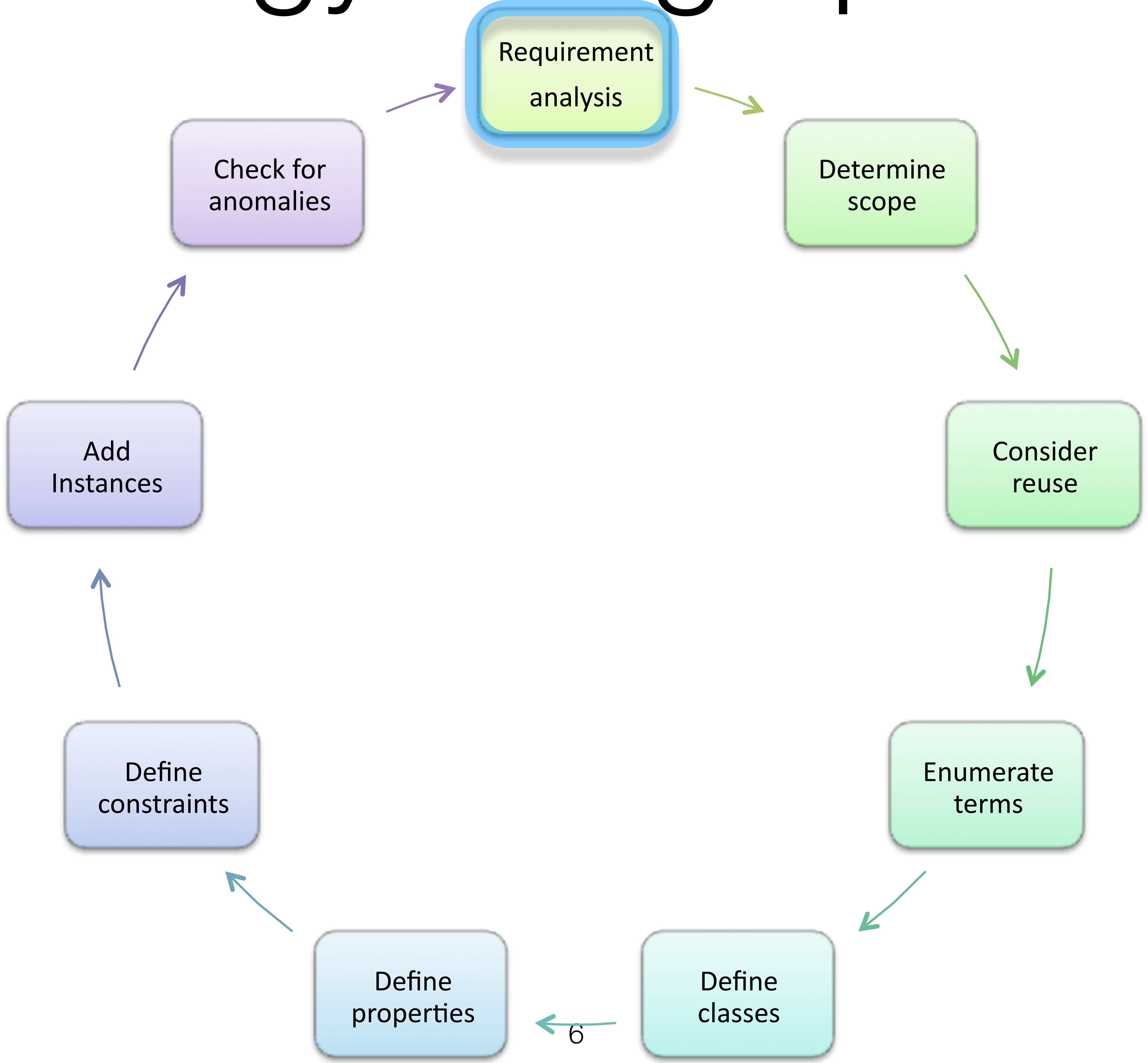


Ontology design process



Process NOT Product

Ontology design process

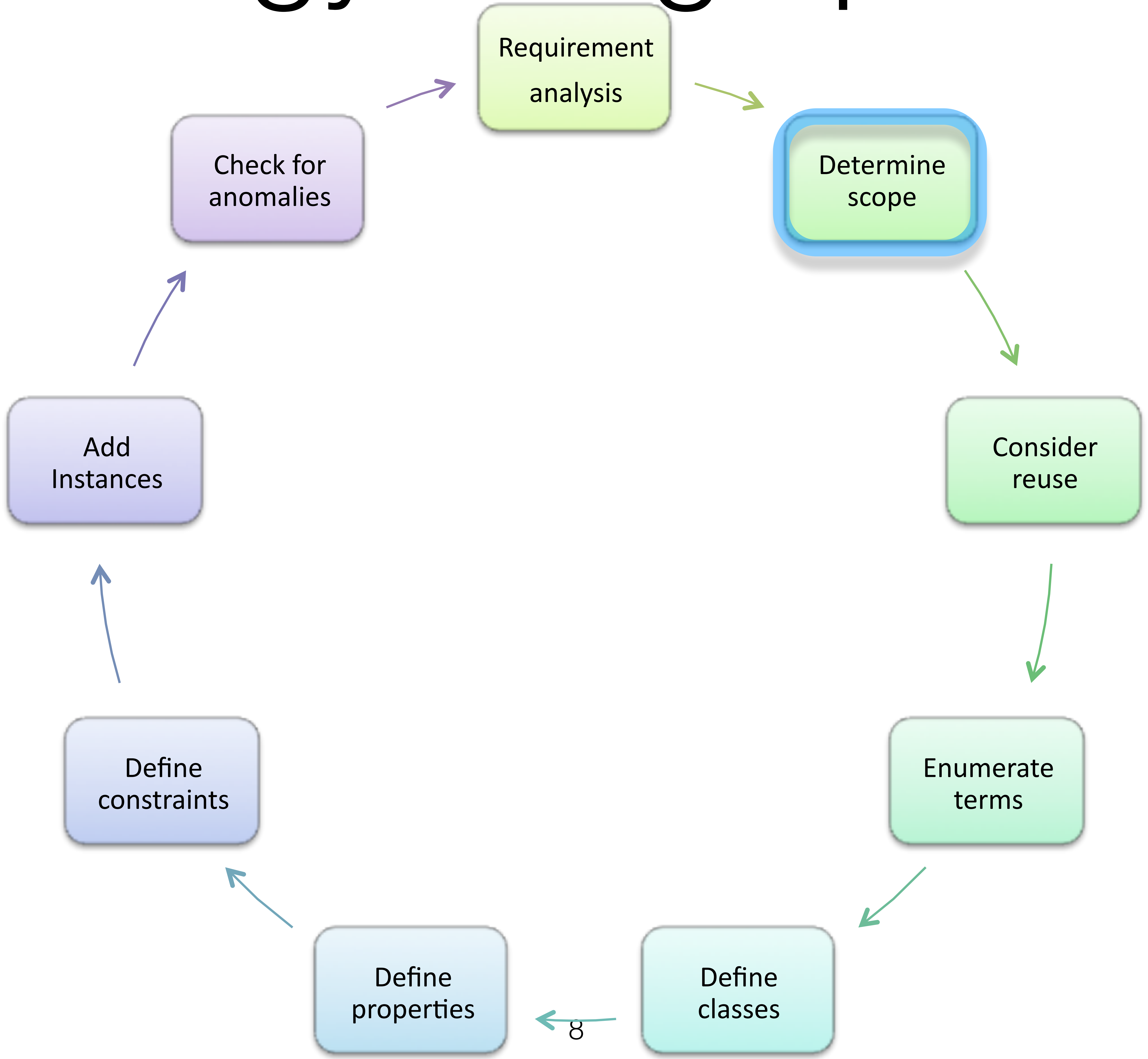


Requirement analysis

- Requirements, Domain & Use Case Analysis are critical phases as in any software engineering design.
  - they allows ontology engineers to ground the work and prioritise.

- The analysis has to elicit and make explicit:
  - The nature of the knowledge and the questions (competency questions) that the ontology (through a reasoner) needs to answer;
    - *This process is crucial for scoping and designing the ontology, and for driving the architecture;*
  - Architectural issues;
  - The effectiveness of using traditional approaches with knowledge intensive approaches;

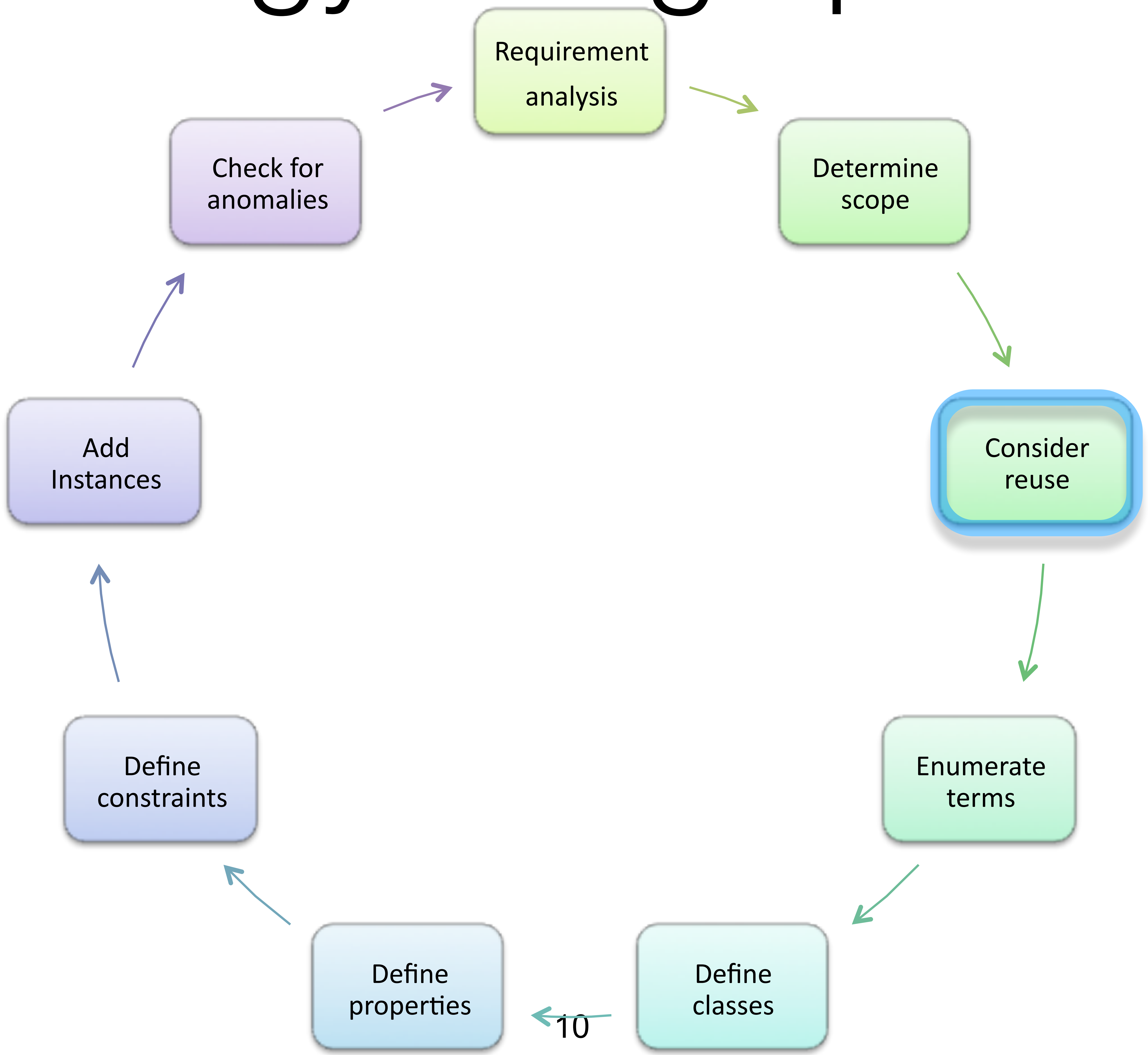
Ontology design process



Determine ontology scope

- There is no correct ontology of a specific domain
  - An ontology is an abstraction of a particular domain, and there are always viable alternatives
- What is included in this abstraction should be determined by
  - the use to which the ontology will be put
  - by future extensions that are already anticipated
- Addresses straight forward questions such as:
  - What is the ontology going to be used for
  - How is the ontology ultimately going to be used by the software implementation?
  - What do we want the ontology to be aware of?
  - What is the scope of the knowledge we want to have in the ontology?

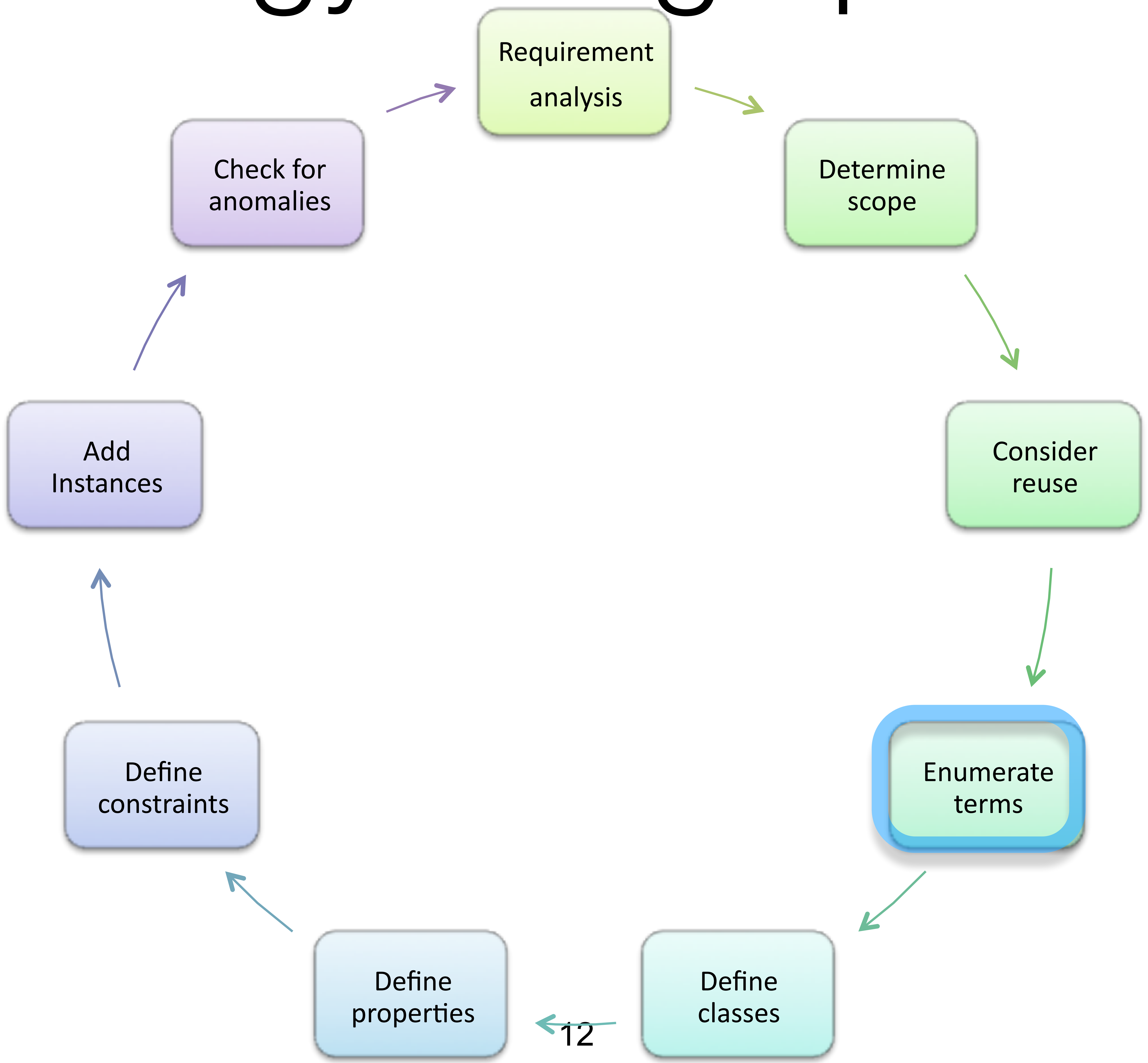
Ontology design process



Consider Reuse

- We rarely have to start from scratch when defining an ontology:
  - There is almost always an ontology available from a third party that provides at least a useful starting point for our own ontology
- Reuse allows to:
  - to save the effort
  - to interact with the tools that use other ontologies
  - to use ontologies that have been validated through use in applications:
    - standard vocabularies are available for most domains, many of which are overlapping
  - Identify the set that is most relevant to the problem and application issues
- A component-based approach based on modules facilitates dealing with overlapping domains:
  - Reuse an ontology module as one would reuse a software module
  - Standards, complex relationships are defined such that term usage and overlap is unambiguous and machine interpretable

Ontology design process

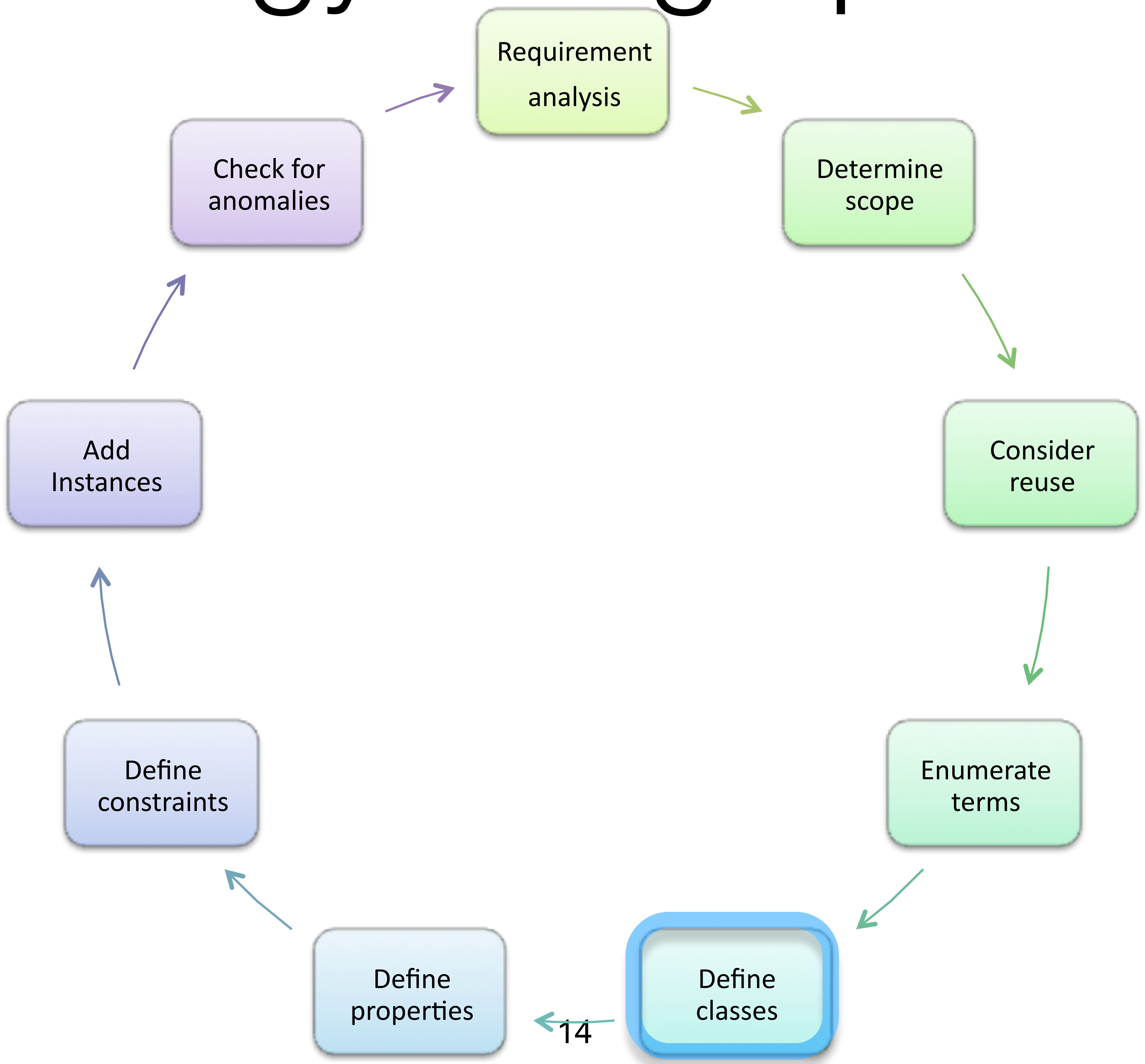


Enumerate terms

- Write down in an unstructured list all the relevant terms that are expected to appear in the ontology
  - Nouns form the basis for class names
  - Verbs (or verb phrases) form the basis for property names

- Card sorting is often the best way:
  - Write down each concept/idea on a card
  - Organise them into piles
  - Link the piles together
  - Do it again, and again
  - Works best in a small group

Ontology design process



Define classes and their taxonomy

- A class is a concept in the domain:
  - Animal
    - cow, cat, fish
  - A class of properties
    - father, mother
- A class contains necessary conditions for membership
  - type of food, dwelling
- A class is a collection of elements with similar properties
- Instances of classes
  - A particular farm animal, a particular person
  - Tweety the penguin

How do we establish the taxonomy

- Relevant terms must be organised in a taxonomic hierarchy
  - Choose some main axes:
    - add ***abstractions*** where needed
  - Identify ***relations***
  - Identify ***definable things***
    - e.g. *Father is an animal who has children, Herbivore, is an animal who only eats grass...*
    - Not everything is definable, "natural kinds" cannot be defined as precisely in terms of properties or constraints
      - a cat is....?

How do we establish the taxonomy

- Relevant terms must be organised in a taxonomic hierarchy
  - Distinguish between ***self standing*** things vs ***modifiers***:
    - ***Self standing*** things exist in their own right
      - typically indicated by nouns, e.g. cat, people, animal, action, process...
    - ***Modifiers*** modify other entities
      - typically denoted by adjectives and adverbs, e.g. wild vs domestic, male vs female, healthy vs sick, etc

- Make sure self-standing terms, modifiers and relations are represented in pure trees
  - no multiple inheritance!
  - these will become the “primitive” concepts from which all other concepts can be defined
    - no definable things

Levels in the class hierarchy

- Different modes of development
  - Top-down
    - define the most general concepts first and then specialize them
  - Bottom-up
    - define the most specific concepts and then organize them in more general classes
  - Combination (typical)
    - breadth at the top level and depth along a few branches to test design
- But... there is no single correct hierarchy
  - guidelines helps us to identify the correct ones
  - Criteria: ***is each instance of the subclass also an instance of the superclass***?

Top level

Middle level

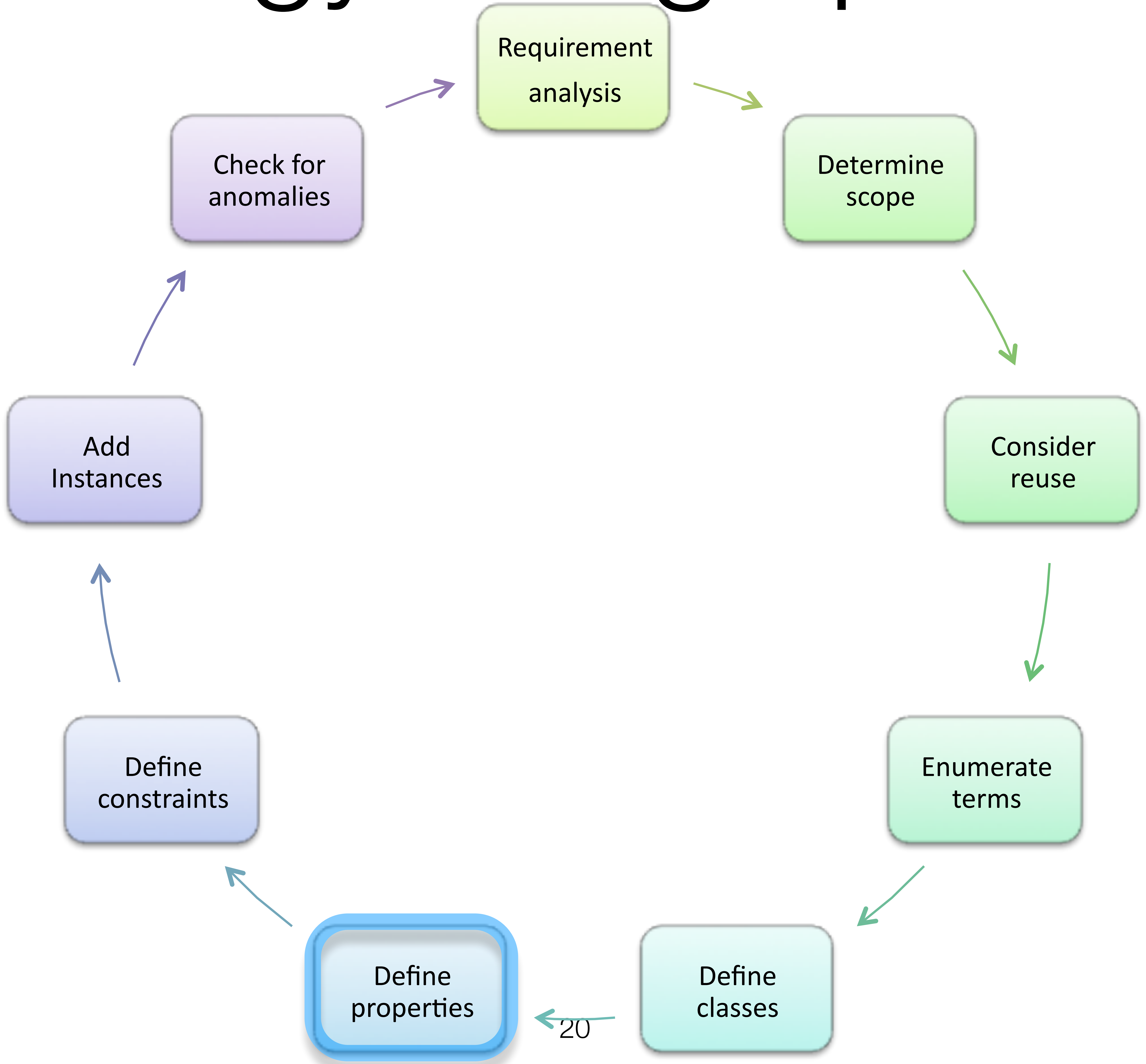
- Living Thing
  - Animal
    - Mammal
      - Cat
      - Dog
      - Cow
      - Person
    - Fish
      - Carp
      - Goldfish
  - Plant
    - Tree
    - Grass
    - Fruit

Bottom level

More criteria

- All the siblings must denote concepts at the ***same level*** of generality
  - *similar to sections and subsections in a book*
- If a class has more than a dozen direct superclasses, then it an additional level of generality is needed
  - *compare to bullets in a bullet list*
    - But in some cases, if no natural classification exist, a long list might reflect the reality better.
- Class names should be either singular or plural, don’t mix!
  - *Animal is not a kind-of Animals*
- Classes represent concepts in the domain, but names do not
  - a class name can change but the concept represented will still be the same
    - Synonym names for the same concepts refer to different labels, not to different classes

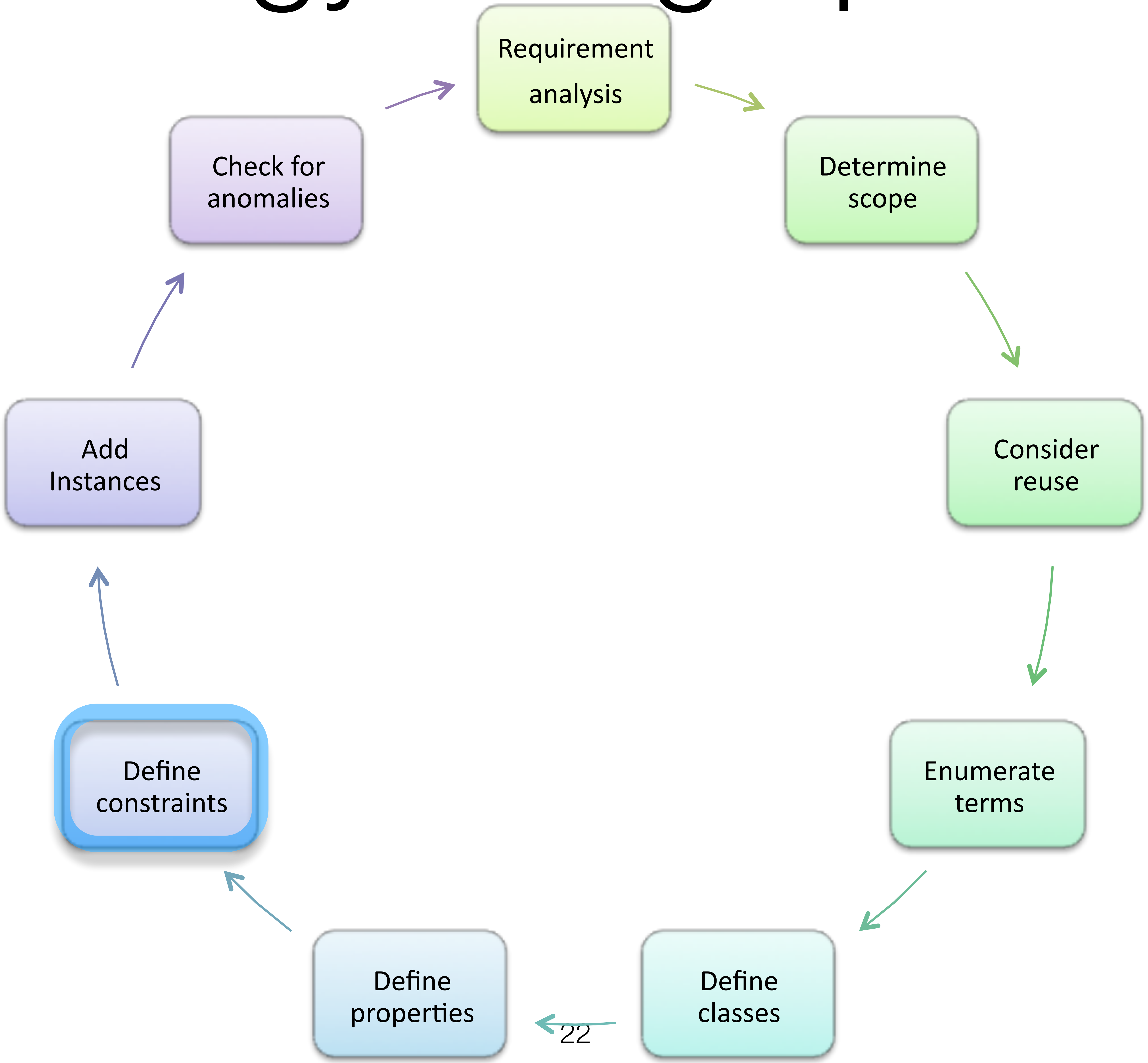
Ontology design process



Define properties

- Often interleaved with the previous step
- Properties (or roles in DL) describe the attributes of the members of a class
  - Defined in terms of domain and range constraints
    - if anything is used in a special way, then add comments
      - *Animal eat LivingThing*, domain: Animal - range: LivingThing
      - *Person owns LivingThing except Person*, domain: Person - range: LivingThing and not Person
      - *Animal parentOf Animal*, domain: Animal - range: Animal
  - Defined in terms of property restrictions
    - What can we say about all instances of a class?
      - *all Cows eat some Plants*
      - *all Cats eat some Animals*
      - *all Pigs eat some Animals and eat some Plants*
- For the semantics of subClassOf whenever A is a subclass of B, every property statement that holds for instances of B must also apply to instances of A
  - It makes sense to attach properties to the highest class in the hierarchy to which they apply

Ontology design process



State constraints: definable things

- Definitions need to be ***paraphrased*** and ***formalised*** in terms of primitive classes, relations and other definable entities
  - Add comments when providing definitions
    - Note any assumptions that need to be represented somewhere else.
  - Paraphrasing needs to achieve consensus on what we meant to represent and how we represent it.

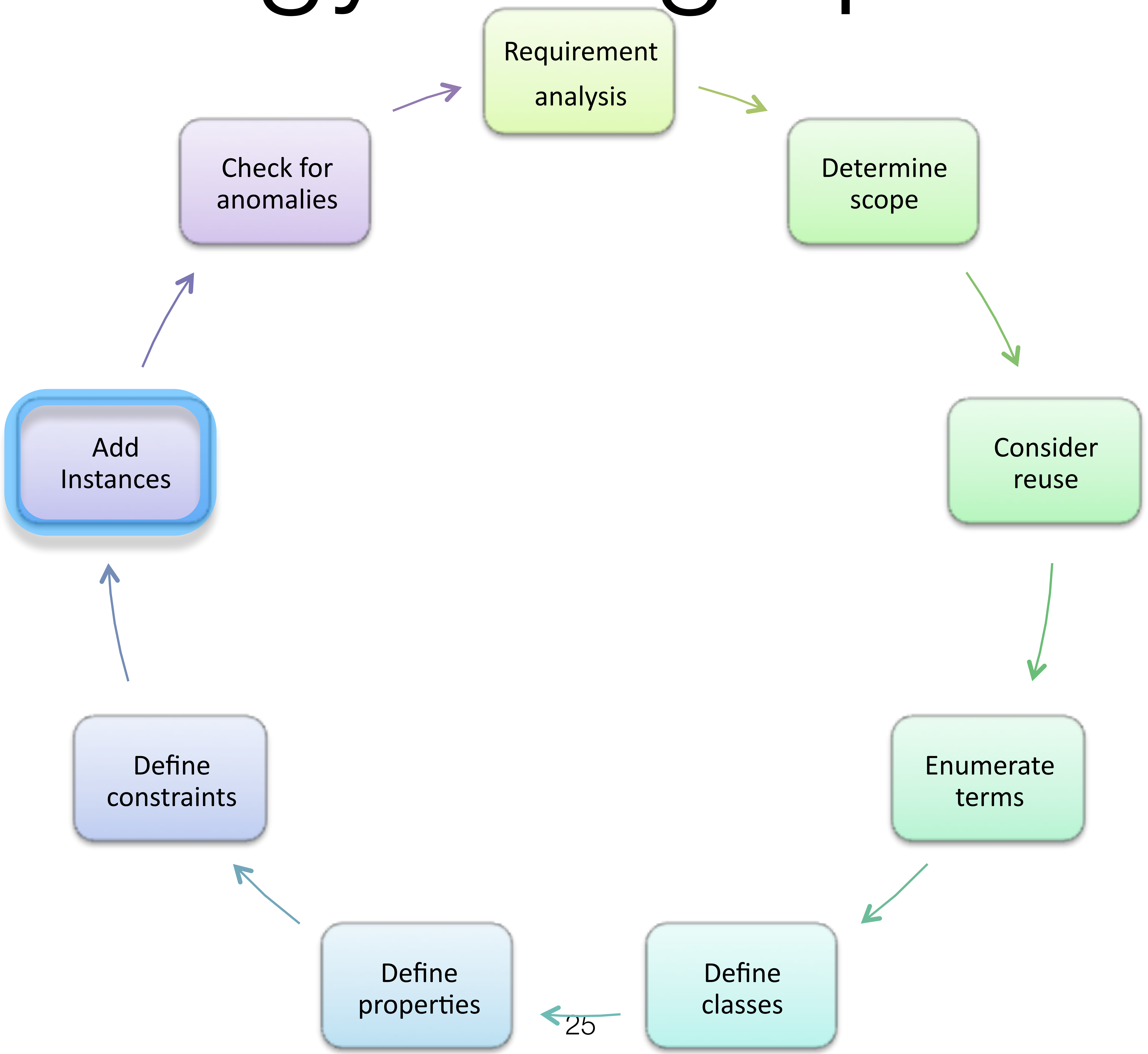
```
:Parent  owl:equivalentClass  [
   rdf:type               owl:Class;
   owl:intersectionOf  (:Animal [
    rdf:type               owl:Restriction ;
    owl:onProperty        :hasChild ;
    owl:someValuesFrom   :Animal .])
 ] .
```

```
:Herbivore  owl:equivalentClass  [
   rdf:type               owl:Class;
   owl:intersectionOf  (:Animal [
   rdf:type               owl:Restriction ;
   owl:onProperty        :eats ;
   /* eats range LivingThing */
   owl:allValuesFrom    :Plant .])
 ] .
```

State constraints: definable things

- *A **Parent** is an **Animal** that is a parent of some **Animal***
  - `Parent = Animal and parentOf some Animal`
- *A **Herbivore** is an **Animal** that eats only **Plants***
  - assume that Animals eat some LivingThing
  - ```
    Herbivore ≡ Animal and eats
                  only Plant
    ```
- *An* **Omnivore** *is an* **Animal** *that eats both* **Plants** *and* **Animals**
  - ```
    Omnivore ≡ Animal and eats
                 some Plant and eats some Animal
    ```

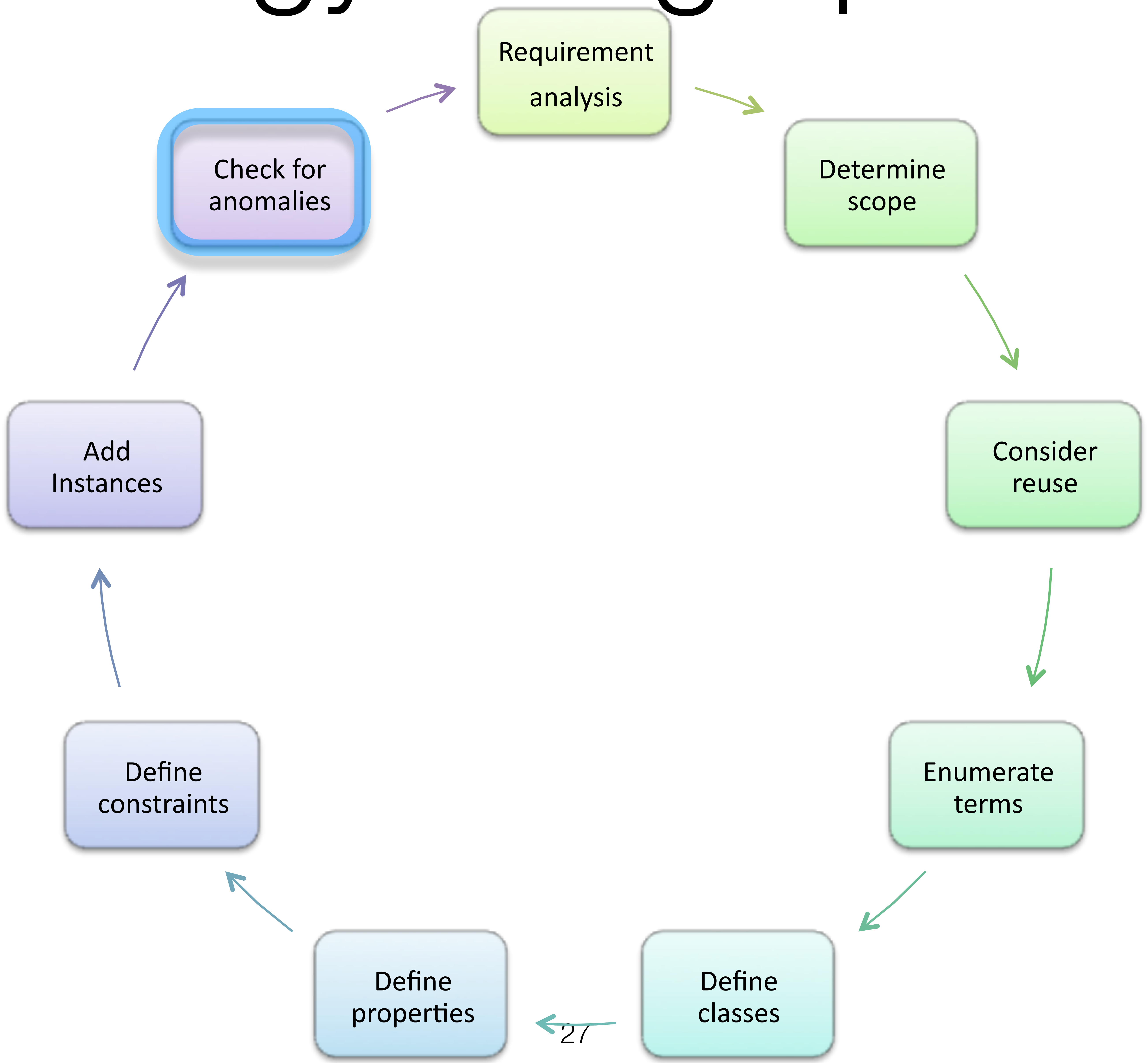
Ontology design process



Creating instances

- Create an instance of a class
    - The class becomes a direct type of the instance
    - Any superclass of the direct type is a type of the instance
- Assign property values for the instance description
    - property values should conform to the constraints asserted for the property
    - Knowledge-acquisition tools often check that constraints are satisfied

Ontology design process



Check for anomalies

- An important advantage of the use of OWL over RDF Schema is the possibility to detect inconsistencies
  - In ontology
    - incoherent ontology: at least an unsatisfiable class, class that cannot have any instance
  - In ontology+instances
    - inconsistent ontology: every class is interpreted as the empty set
- Examples of common inconsistencies
  - incompatible domain and range definitions for transitive, symmetric, or inverse properties
  - cardinality properties
  - requirements on property values can conflict with domain and range restriction
- Examples from the Pizza tutorial for Protege
  - http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/

Ontology representation in OWL

- We use turtle syntax for the examples
- Class definition: SubClassOf vs EquivalentClasses
  - All PizzaMargherita have, amongst other things, some mozzarella topping and some tomato topping

```
:Margherita rdf:type owl:Class ;
            rdfs:subClassOf :NamedPizza ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom :TomatoTopping
                            ] ,

                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom :MozzarellaTopping
                            ] ;
```

Ontology representation in OWL

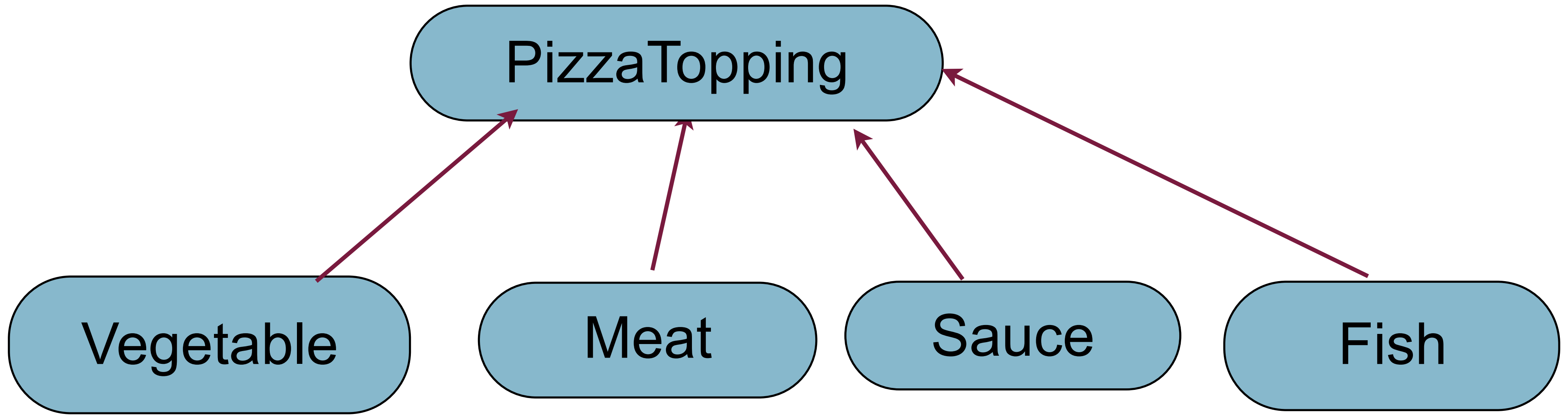
- Class definition: SubClassOf vs EquivalentClasses
  - A MeatyPizza is any pizza that has, amongst other things, at least one meat topping

```
:MeatyPizza rdf:type owl:Class ;
            owl:equivalentClass [ rdf:type owl:Class ;
                                  owl:intersectionOf (
                                      :Pizza
                                      [ rdf:type owl:Restriction ;
                                        owl:onProperty :hasTopping ;
                                        owl:someValuesFrom :MeatTopping
                                      ]
                                  )
                                ] ;
```

Ontology representation in OWL

- Disjointness
  - States that all disjoint classes belong to different branches in the ontology tree!
    - Classes can overlap unless a disjointness axiom is added.

```
:MeatTopping rdf:type owl:Class ;
             rdfs:subClassOf :PizzaTopping ;
             owl:disjointWith :FishTopping ,
                              :SauceTopping ,
                              :VegetableTopping .
```



Ontology representation in OWL

- ***Existential property restriction: someValuesFrom***
  - Can be used to declare primitive or defined classes
  - *Indicates some or at least one*
    - *A cheeseyPizza is any pizza that has some (at least one) cheese topping*

```
:CheeseyPizza rdf:type owl:Class ;
              owl:equivalentClass
                 [ rdf:type owl:Class ;
                   owl:intersectionOf ( :Pizza
                                        [ rdf:type owl:Restriction ;
                                          owl:onProperty :hasTopping ;
                                          owl:someValuesFrom :CheeseTopping
                                        ]
                                      )
                 ] ;
```

Ontology representation in OWL

- ***Existential property restriction: someValuesFrom***
  - Can be used to declare primitive or defined classes
  - *Indicates some or at least one*
    - *CheeseyPizza are pizza and have some (at least one) cheese topping*

```
:CheeseyPizza rdf:type owl:Class ;
              owl:subClassOf
                 [ rdf:type owl:Class ;
                   owl:intersectionOf ( :Pizza
                                        [ rdf:type owl:Restriction ;
                                          owl:onProperty :hasTopping ;
                                          owl:someValuesFrom :CheeseTopping
                                        ]
                                      )
                 ] ;
```

Understanding restrictions

- ***Understanding OWL requires some understanding of how DL models the world.***
  - In DL we ofter use subsumption to model classes
    - OWL uses the rdfs:subClassOf for representing subsumption.
  - Suppose we want to state the cheesy toppings have some ingredient "cheese" as their “main ingredient”.
    - *"Cheesy topping is a subclass of all things that have as main ingredient some cheese.”*

```
:CheeseTopping
  a owl:Class ;
  rdfs:subClassOf
    [ a owl:Restriction ;
      owl:onProperty :mainIngredient ;
      owl:someValueFrom Cheese ].
```

Understanding restrictions

- All cheesy toppings form a subset of all things that have as main ingredient some cheese.
  - If you omit the subClassOf construct you can read it as a UML class with an attribute mainIngredient and a value type constraint for the attribute.

- The subclass relation is essential for understanding the semantics of the restriction:
  - it is a necessary but not sufficient condition for the class.
    - There might be things that have cheese as main ingredient but are not a cheesy topping, hence the subset/subclass definition.

Understanding restrictions

- The definition states that the set of CheeseTopping things is exactly the same as the class of things that have as main ingredient at least one thing that is cheese.

- These are also sometimes called "defined classes";

- Classes declared with just necessary conditions are sometimes called "primitive classes”.

```
:CheeseTopping
  a owl:Class ;
  owl:EquivalentTo
    [ a owl:Restriction ;
      owl:onProperty :mainIngredient ;
      owl:someValueFrom Cheese ].
```

Ontology representation in OWL

- Existential property restriction: someValuesFrom
  - Can be used to declare primitive or defined classes
  - Indicates some or at least one
    - A cheeseyPizza is any pizza that has some (at least one) cheese topping

```
:CheeseyPizza rdf:type owl:Class ;
              owl:equivalentClass
                 [ rdf:type owl:Class ;
                   owl:intersectionOf ( :Pizza
                                        [ rdf:type owl:Restriction ;
                                          owl:onProperty :hasTopping ;
                                          owl:someValuesFrom :CheeseTopping
                                        ]
                                      )
                 ] ;
```

Ontology representation in OWL

- Existential property restriction: someValuesFrom
  - Can be used to declare primitive or defined classes
  - Indicates some or at least one
    - CheeseyPizza are pizza and have some (at least one) cheese topping

```
:CheeseyPizza rdf:type owl:Class ;
              owl:subClassOf
                 [ rdf:type owl:Class ;
                   owl:intersectionOf ( :Pizza
                                        [ rdf:type owl:Restriction ;
                                          owl:onProperty :hasTopping ;
                                          owl:someValuesFrom :CheeseTopping
                                        ]
                                      )
                 ] ;
```

Ontology representation in OWL

- Universal property restriction: allValuesFrom
  - Can be used to declare primitive or defined classes
  - Indicates only or no values except
    - A thin and crispy pizza is any pizza where the base is only a thin and crispy base

```
:ThinAndCrispyBase rdf:type owl:Class ;
                   rdfs:subClassOf :PizzaBase .
:ThinAndCrispyPizza rdf:type owl:Class ;
                    owl:equivalentClass
                       [ rdf:type owl:Class ;
                         owl:intersectionOf (
                            :Pizza [ rdf:type owl:Restriction ;
                                     owl:onProperty :hasBase ;
                                     owl:allValuesFrom :ThinAndCrispyBase ]
                                            )
                        ] .
```

Ontology representation in OWL

- Universal property restriction: allValuesFrom
  - Can be used to declare primitive or defined classes
  - Indicates only or no values except
    - All thin and crispy pizza have a pizza whose base is a thin and crispy base

```
:ThinAndCrispyBase rdf:type owl:Class ;
                   rdfs:subClassOf :PizzaBase .
:ThinAndCrispyPizza rdf:type owl:Class ;
                    owl: subClassOf
                       [ rdf:type owl:Class ;
                         owl:intersectionOf (
                            :Pizza [ rdf:type owl:Restriction ;
                                     owl:onProperty :hasBase ;
                                     owl:allValuesFrom :ThinAndCrispyBase ]
                                           )
                        ] .
```

Ontology representation in OWL

- Boolean Combinations
  - Union (disjunction)
    - A vegetarian pizza is any pizza which, amongst other things, has only vegetable and/or cheese toppings

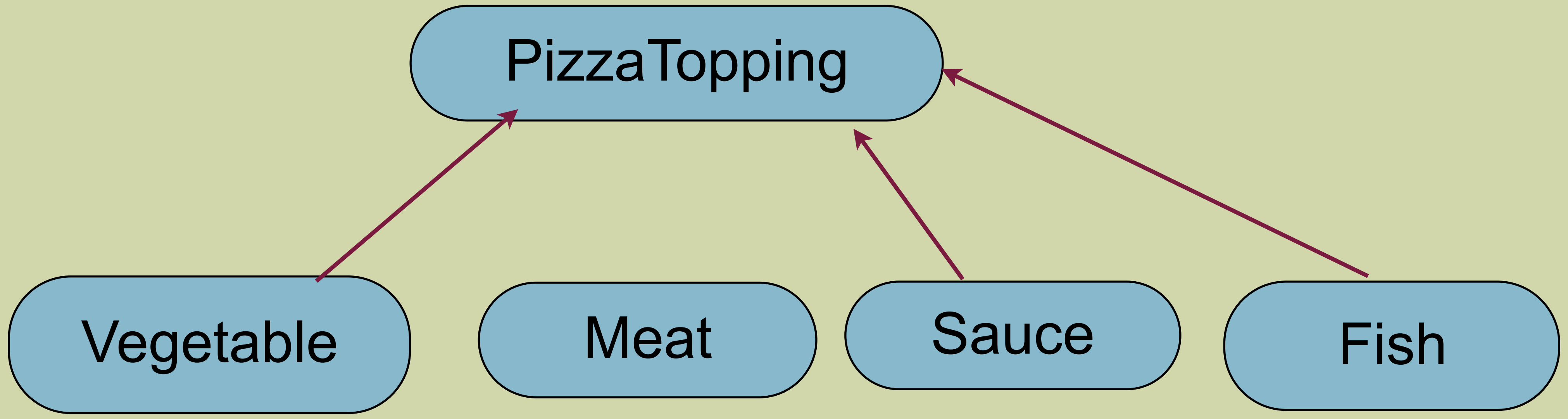
```
:VegetarianPizza rdf:type owl:Class ;
                 owl:equivalentClass
                    [ rdf:type owl:Class ;
                      owl:intersectionOf ( :Pizza
                                           [ rdf:type owl:Restriction ;
                                             owl:onProperty :hasTopping ;
                                             owl:allValuesFrom
                                                [ rdf:type owl:Class ;
                                                  owl:unionOf ( :CheeseTopping
                                                                :VegetableTopping
                                                              )
                                                ]
                                           ]
                                         )
                    ]
```

Ontology representation in OWL

- Boolean Combinations
  - Intersection (conjunction)
    - A ProteinLover’s is any pizza that has, amongst other things, has only toppings that are both meat and seafood

```
:ProteinLoverPizza rdf:type owl:Class ;
                   owl:equivalentClass
                      [ rdf:type owl:Class ;
                        owl:intersectionOf ( :Pizza
                                             [ rdf:type owl:Restriction ;
                                               owl:onProperty :hasTopping ;
                                               owl:allValuesFrom
                                                  [ rdf:type owl:Class ;
                                                    owl:intersectionOf
                                                        ( :FishTopping
                                                          :MeatTopping
                                                        )
                                                  ]
                      ]
```

Empty!



Ontology representation in OWL

- Other Boolean Combinations
  - complement
    - A non vegetarian pizza is any pizza that is not a vegetarian one

```
:NonVegetarianPizza rdf:type owl:Class ;
                    owl:equivalentClass
                       [ rdf:type owl:Class ;
                         owl:intersectionOf ( :Pizza
                                              [ rdf:type owl:Class ;
                                                owl:complementOf :VegetarianPizza
                                              ]
                                            )
                                        ] ;

                    owl:disjointWith :VegetarianPizza ;
```

Reasoning with OWL

- Reasoning with OWL is based on Description Logic reasoning

- The Open world assumption holds:
  - anything might be true unless it can be proven false
  - it states that everything we don't know is undefined
    - no single agent or observer has complete knowledge

- Reasoning is key in:
  - Designing and maintaining good quality ontologies
    - Meaningful: all named classes can have instances
    - Correct: captures intuitions of domain experts
    - Minimally redundant: no unintended synonyms
  - Answer queries, e.g.:
    - Find more general/specific classes
    - Retrieve individuals/tuples matching a given query

Common errors: “Some” does not mean “only”

- *All MargheritaPizza have, amongst other things, some mozzarella topping and some tomato topping*
  - It is an open world, hence these MargheritaPizza can have other types of toppings, e.g. spicyTopping

```
:Margherita rdf:type owl:Class ;
            rdfs:subClassOf :NamedPizza ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom :TomatoTopping
                            ] ,

                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom :MozzarellaTopping
                            ] ;
```

“*Some*” does not mean “*only*”

```
:Margherita rdf:type owl:Class ;
            rdfs:subClassOf :NamedPizza ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom :TomatoTopping
                            ] ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom :MozzarellaTopping
                            ] ,


                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:allValuesFrom [ rdf:type owl:Class ;
                                                  owl:unionOf( :MozzarellaTopping
                                                               :TomatoTopping
                                                             )
                                                ]
                            ] ,
            owl:disjointWith :Mushroom ,
                             :Napoletana ,
```

All Margherita Pizza have, amongst other things, some mozzarella topping and some tomato topping and also have ***only*** *mozzarella and/or tomato topping*

"*Some*" does not mean "*only*"

```
:EmptyPizza rdf:type owl:Class ;
            rdfs:subClassOf :Pizza ,
                            [ rdf:type owl:Restriction ;
                              owl:onProperty :hasTopping ;
                              owl:someValuesFrom owl:Thing
                            ] ;
```

All empty pizzas, amongst other things, do not have any topping

Empty pizza satisfies the definition of Vegetarian Pizza,

```
:VegetarianPizza rdf:type owl:Class ;
                 owl:equivalentClass
                    [ rdf:type owl:Class ;
                      owl:intersectionOf ( :Pizza
                                           [ rdf:type owl:Restriction ;
                                             owl:onProperty :hasTopping ;
                                             owl:allValuesFrom
                                                [ rdf:type owl:Class ;
                                                  owl:unionOf ( :CheeseTopping
                                                                :VegetableTopping
                                                              )
                                                ]
                                           ]
                                         )
                    ]
```

Domain and range constraints

- Domain and range constraints are axioms:
  - *All things have no name except classRange vs having a name implies being classDomain*
  - Violating domain and range constraints can give some unwanted results due to reasoning

```
:hasTopping rdf:type owl:InverseFunctionalProperty ,
                     owl:ObjectProperty ;
            rdfs:domain :Pizza ;
            rdfs:range :PizzaTopping ;
            rdfs:subPropertyOf :hasIngredient .
```

Brief set of guidelines

- Always **paraphrase** a description or definition before encoding it in OWL
- Make all primitives **disjoint** (requires that primitives form trees)
- Use `ObjectSomeValuesFrom` as the default quantifier in restrictions
  - Be careful to make defined classes **defined**. The classifier will place nothing (⊥) under a primitive class
    - except in the presence of axioms/domain/range constrains
- Don't forget the **open world assumption**.
  - Insert closure restrictions if that is what you mean
- Be careful to model **domain** and **range** constraints.
  - Check them carefully if classification does not work as expected
- Be careful about the use of "**and**" and "**or**" (intersection and union)
- Run the classifier frequently; spot errors early.

Summary

- Ontology engineering
  - Ontology 101