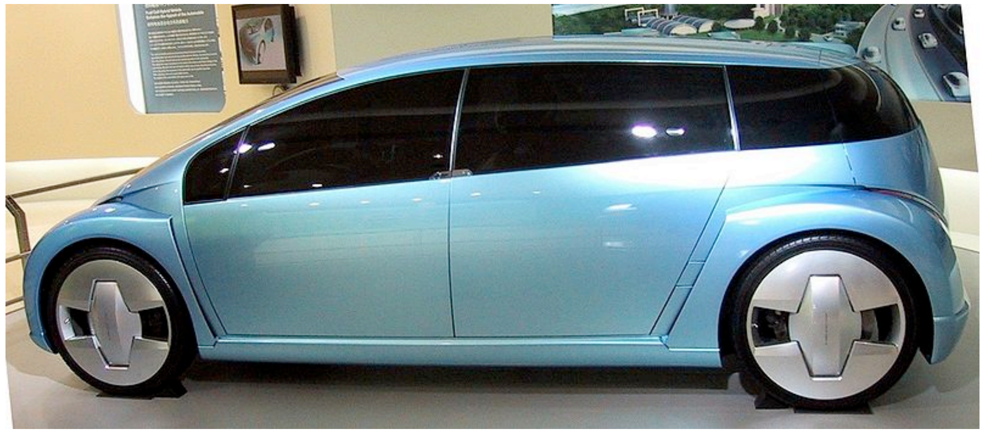
SOFISMO

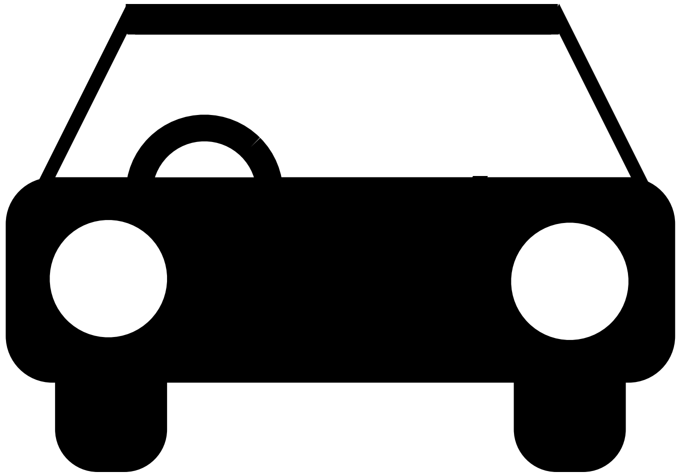
Your code pilot

From Muddling to Modelling

in software, economics, engineering, science



instance ?



class ?

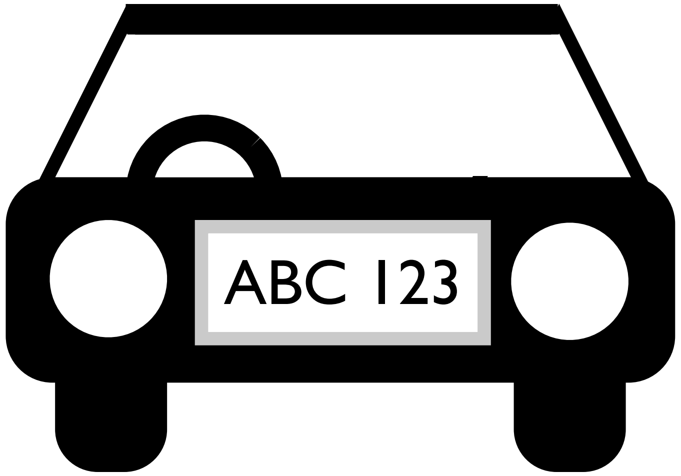
encrypting ?

coding ?

modelling ?

muddling ?

model ?



object ?

Dictionary definition: to code

express (a statement or communication) in an indirect way

Coding can be viewed as having to deal with someone else's representation (program notation or otherwise)

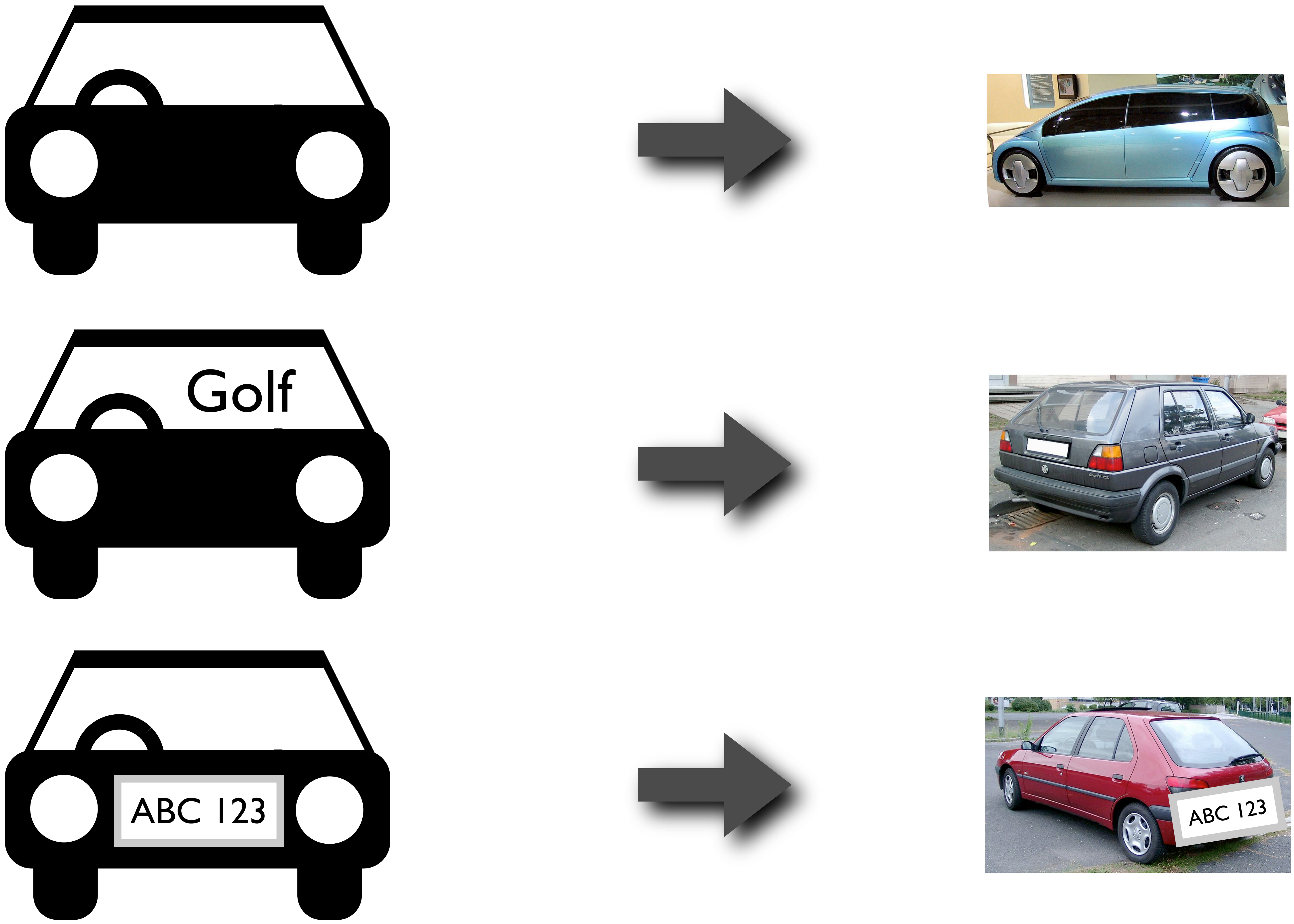
Dictionary definition: to model

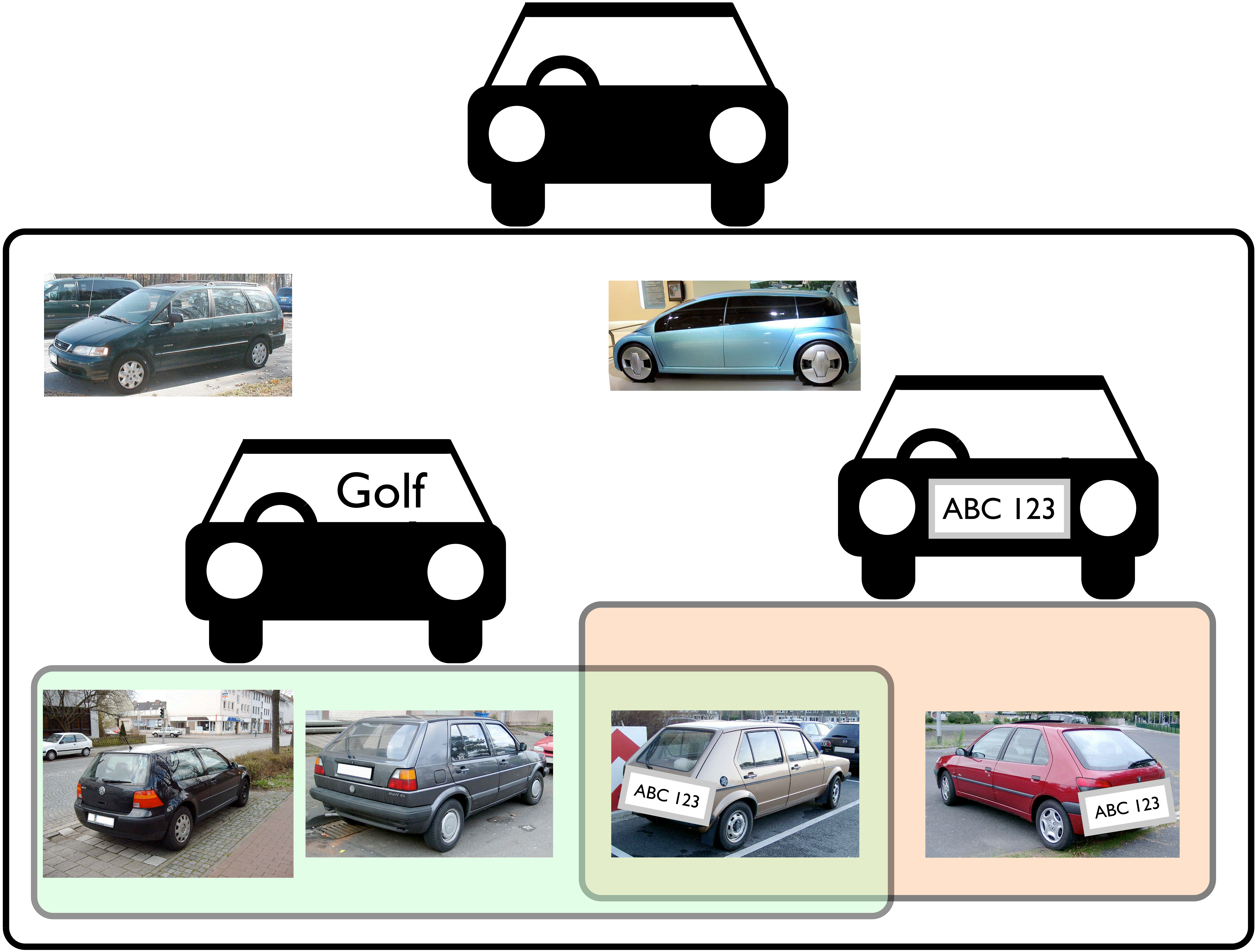
devise a representation, especially a mathematical one of (a phenomenon or system)

Modelling can be viewed as dealing with a representation that is fit for purpose

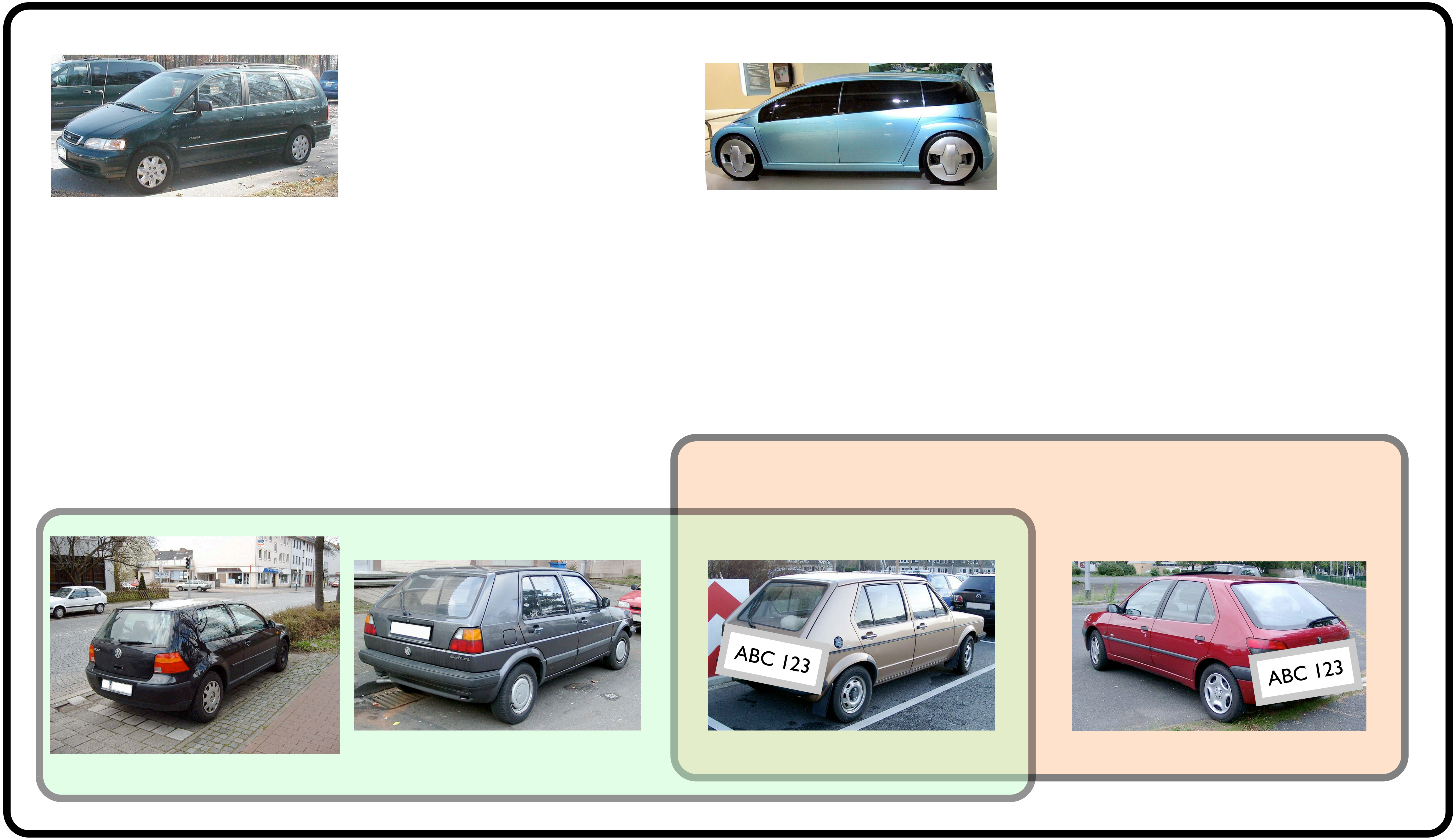
Notation

Semantics ?

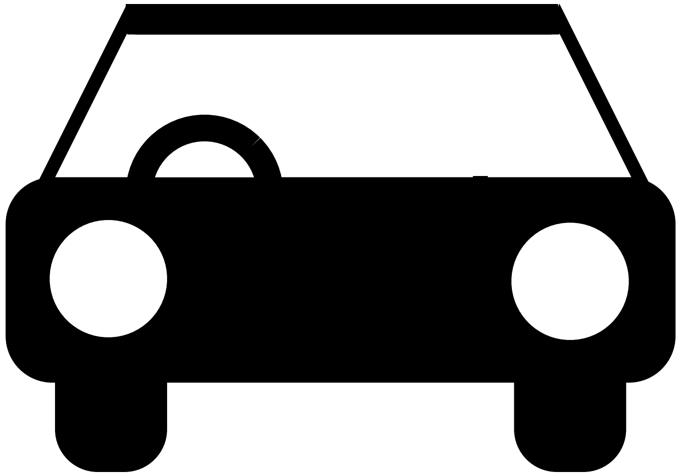


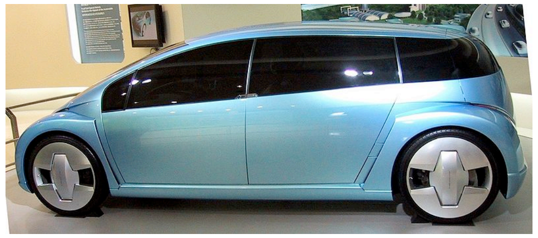

Golf
ABC 123
ABC 123
ABC 123

A familiar setting from elementary school maths



cars





Observation:
Modelling of abstractions relies on concepts from **pure mathematics**, it requires no statistics or other applied mathematics

Observation: Models are a way of referring to useful sets or subsets in a domain

Observation:

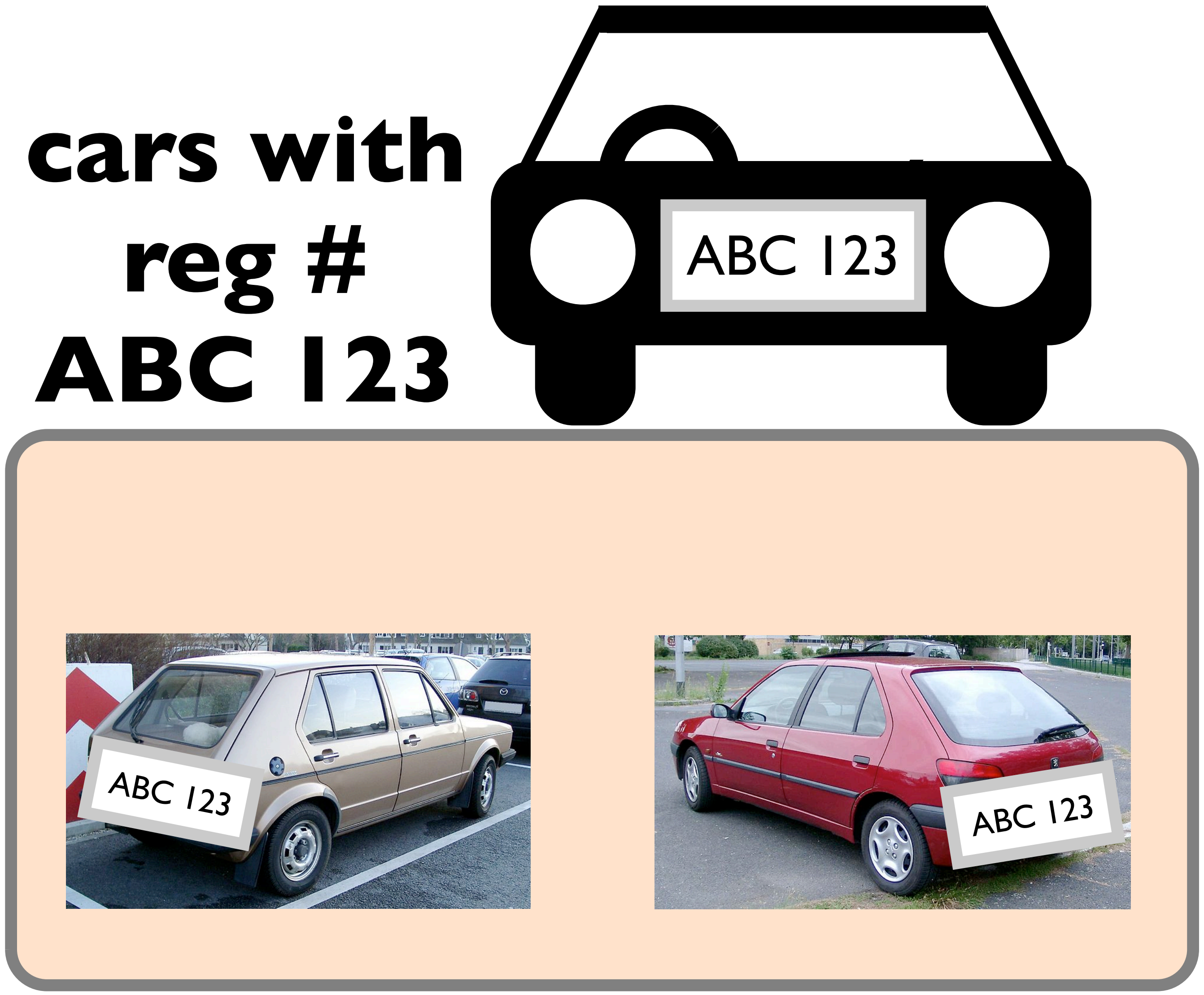
The elements of a set may change over time

Definition:

A **query** is a function that returns the content of a set at a given point in time

Golf cars

Observation: Only one car with reg # ABC 123 can exist at any given time

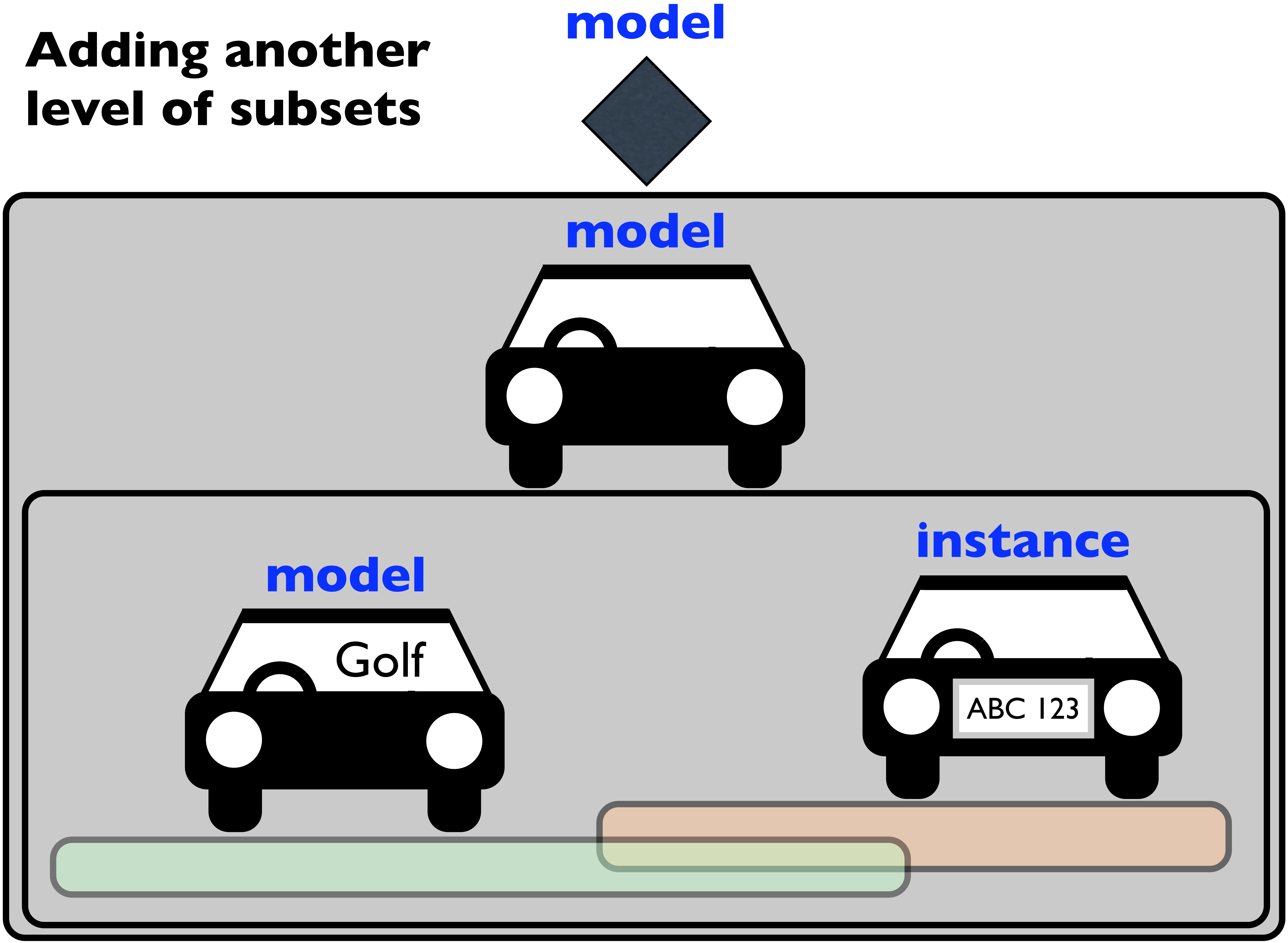


cars with reg # ABC 123

Definition:

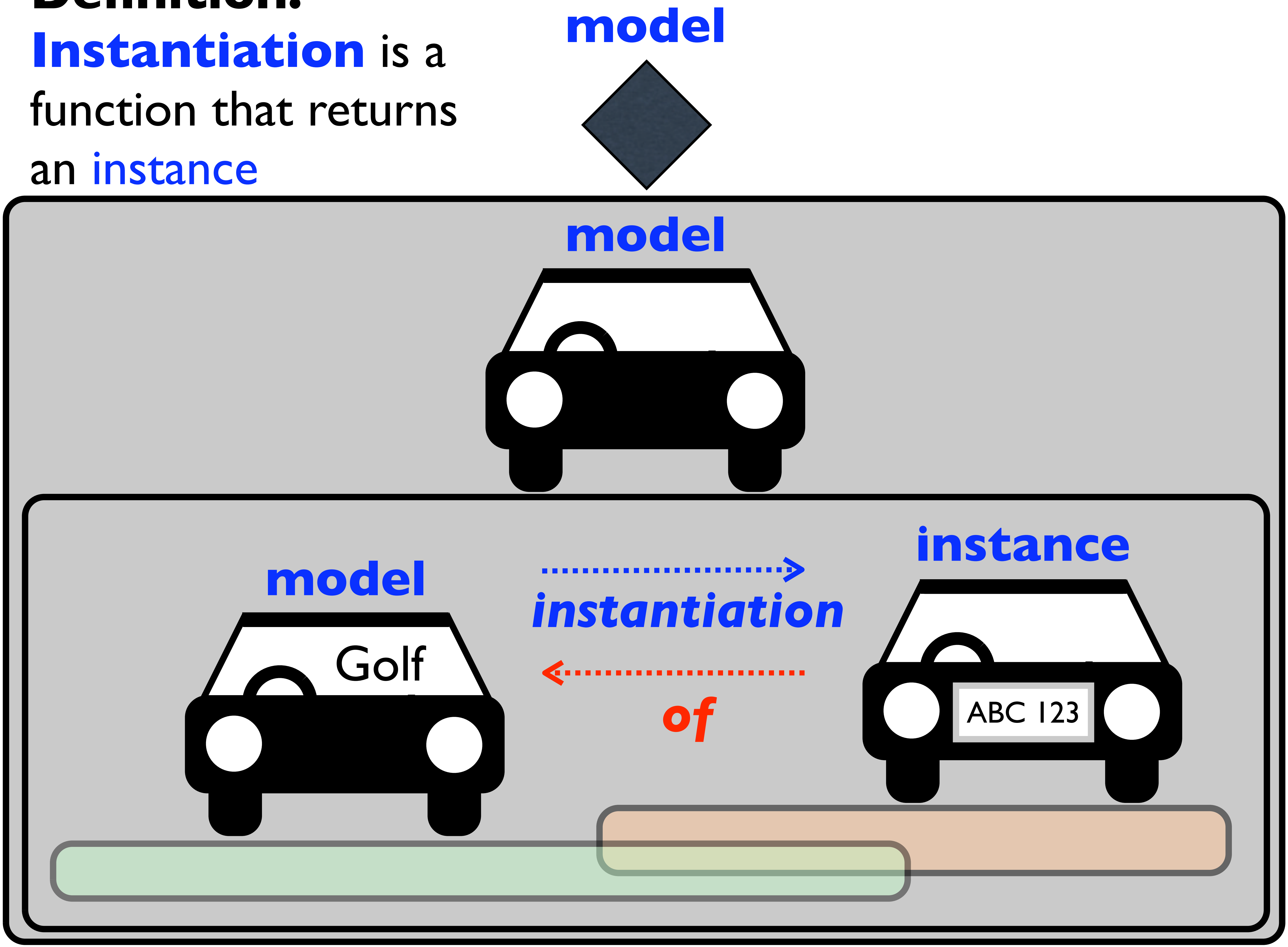
An **instance** is a set that contains one and only one element at any given point in time

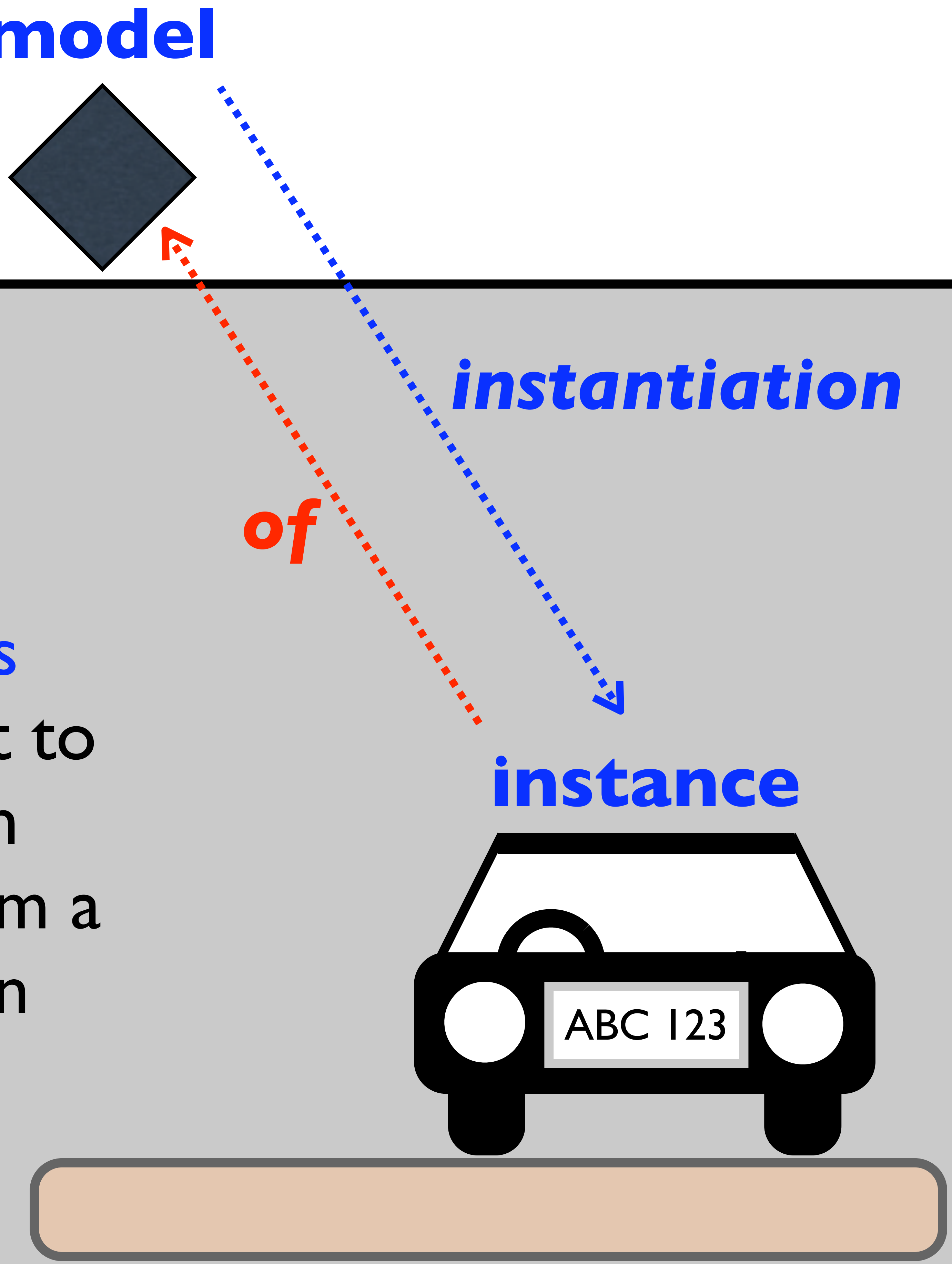
Adding another level of subsets



Definition:

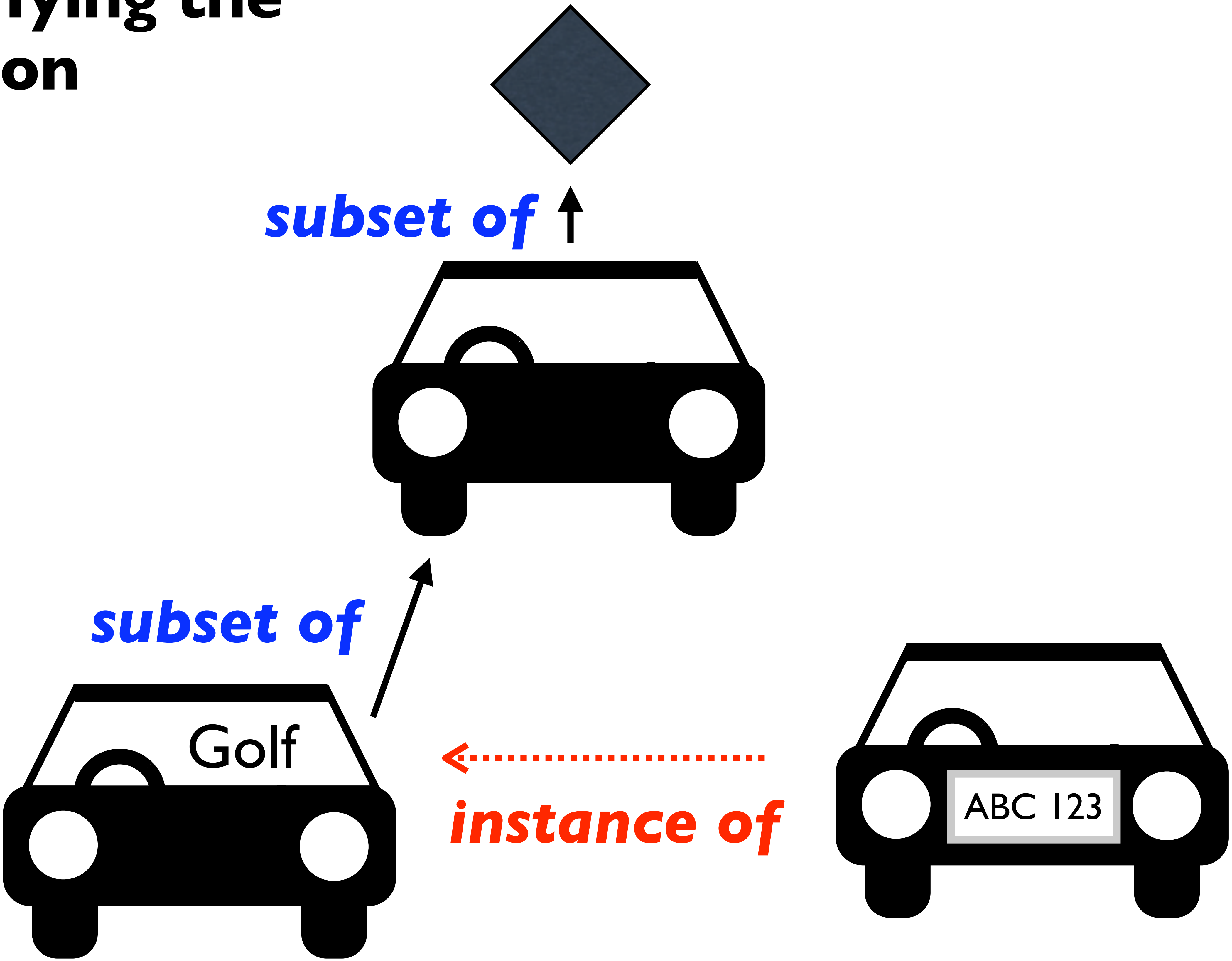
Instantiation is a function that returns an instance



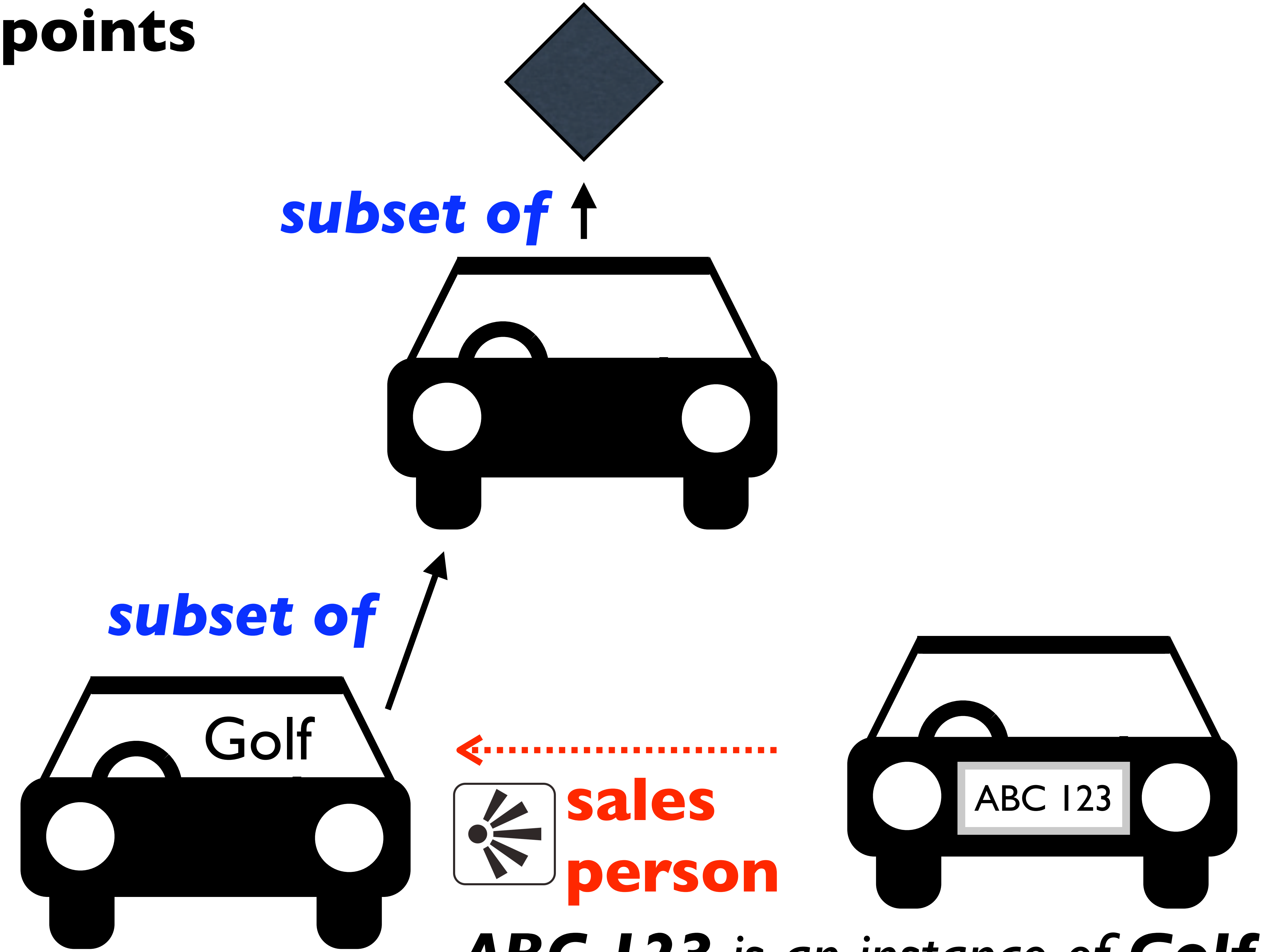


Observation: If the intermediate subsets cars and Golf are not relevant to our model, we can use an instantiation function from a higher level of abstraction

Simplifying the notation



View points



subset of

subset of

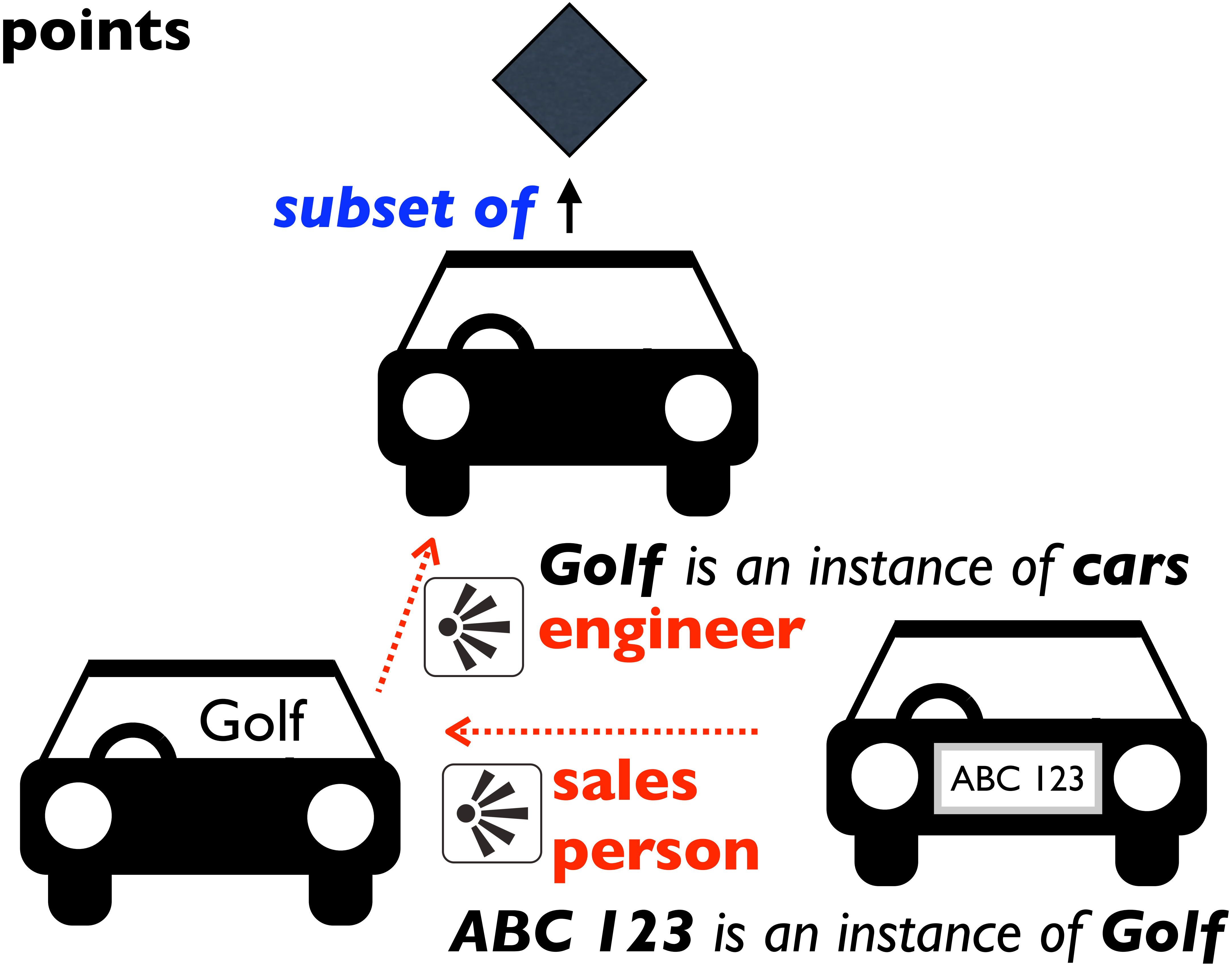
Golf

sales person

ABC 123

ABC 123 *is an instance of* ***Golf***

View points



subset of

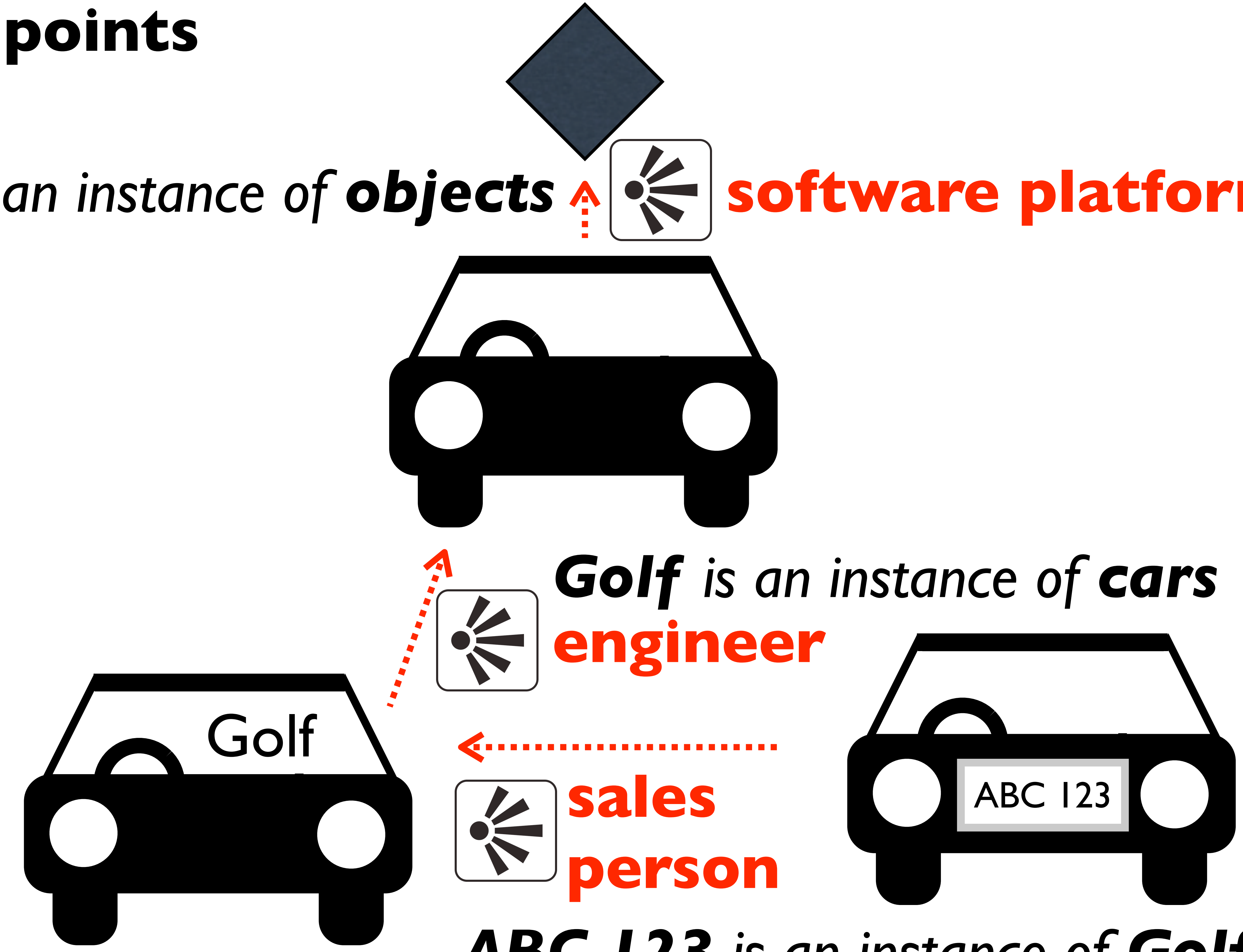
Golf *is an instance of* ***cars***

engineer

sales person

ABC 123 *is an instance of* ***Golf***

View points



cars *is an instance of* ***objects***

software platform

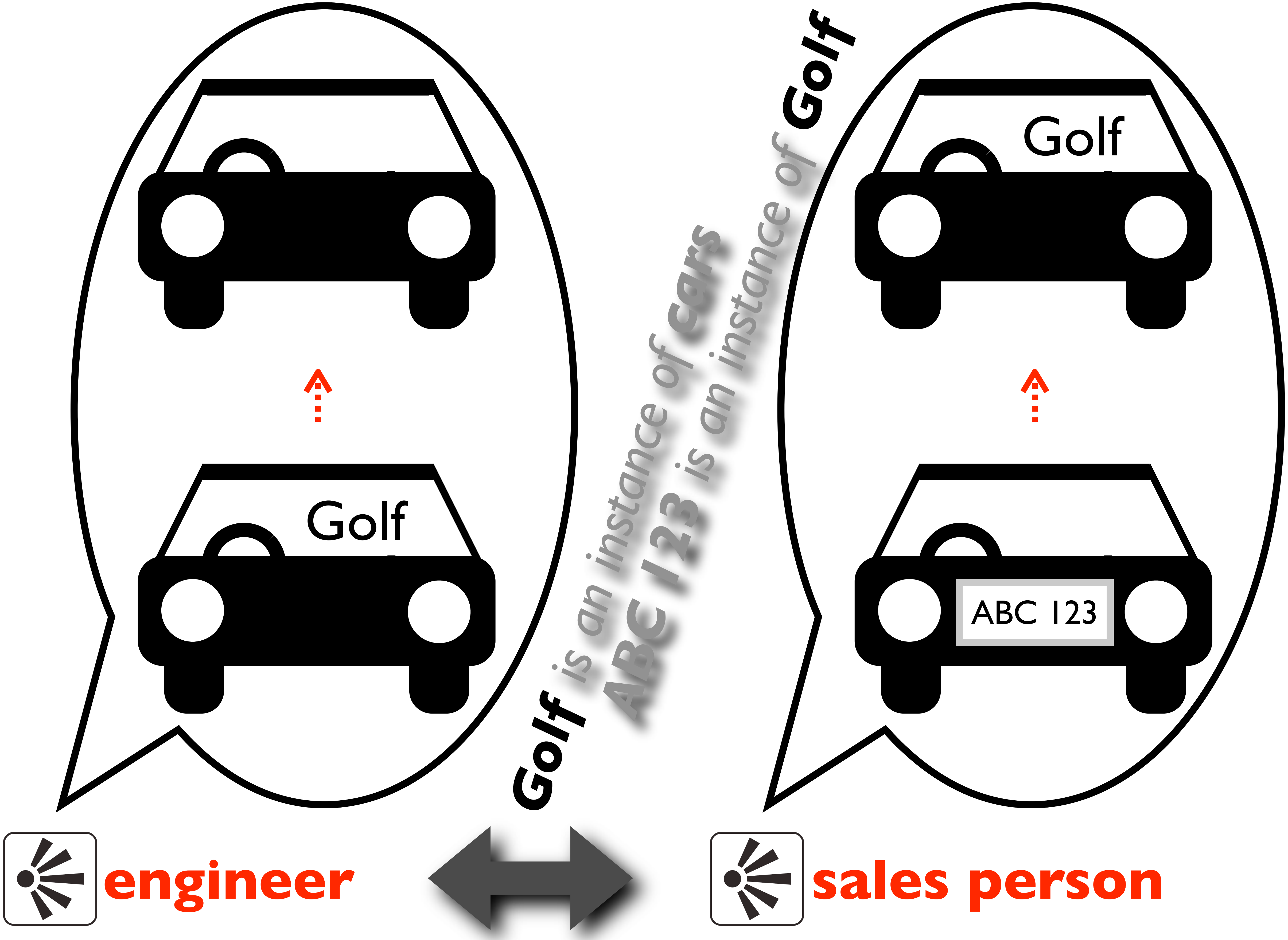
Golf *is an instance of* ***cars***

engineer

sales person

ABC 123 *is an instance of* ***Golf***

Communication ?



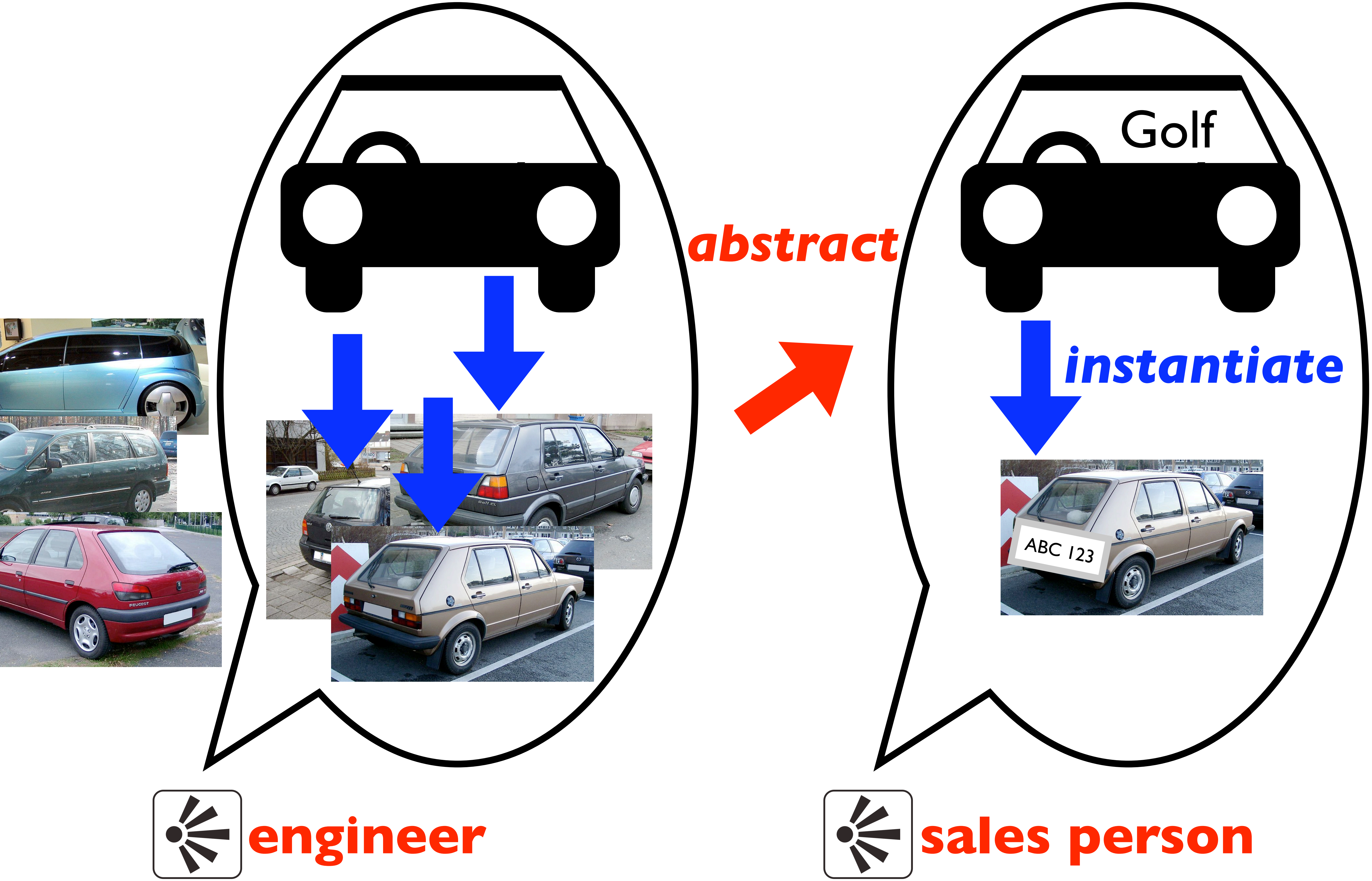
Golf is an instance of cars

ABC 123 is an instance of Golf

engineer

sales person

Abstractions must be validated via instantiation



abstract

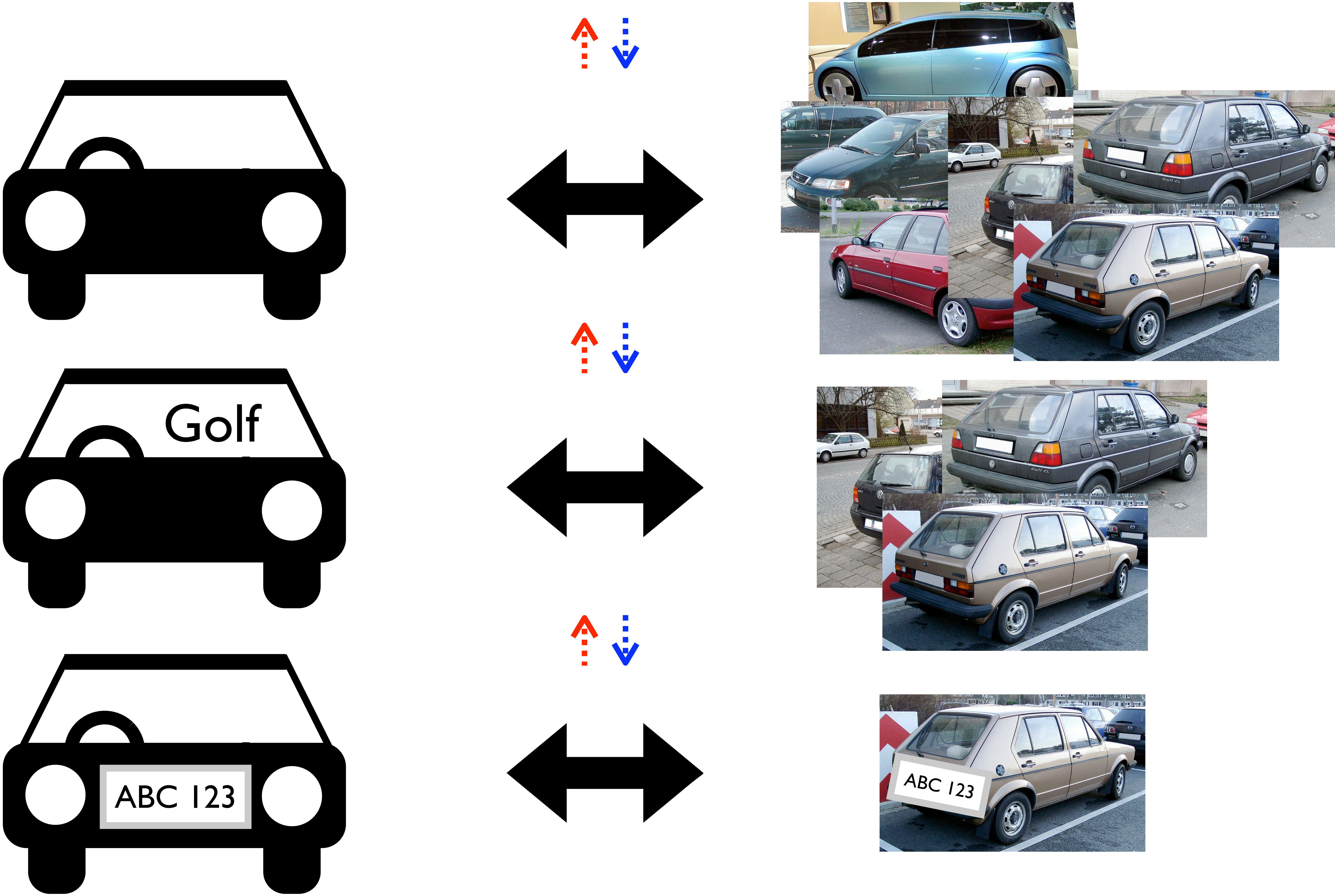
instantiate

engineer

sales person

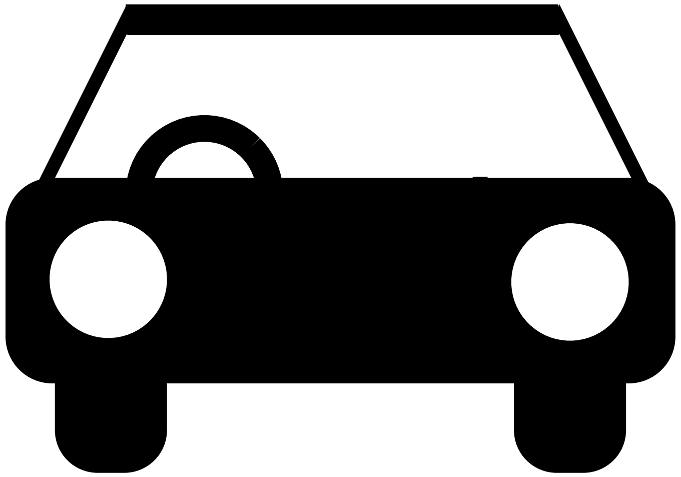
Notation

Semantics



No support for **multi-level instantiation** in any industry standard modelling/programming language !

↑ ↓

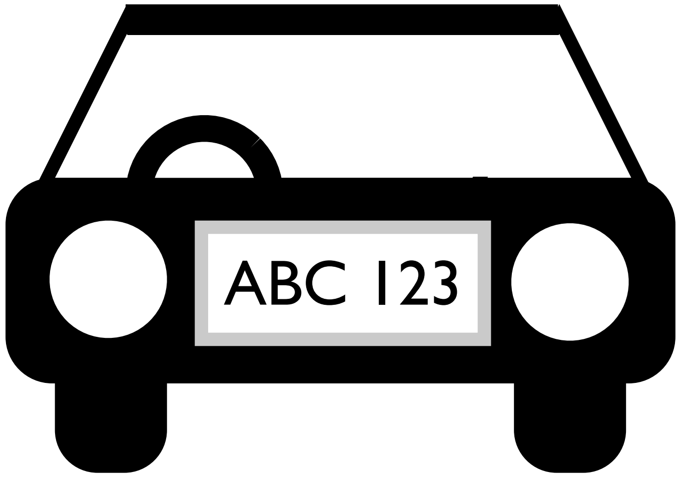


cars *is an instance of* ***objects***

↑ ↓ **engineer**

Golf *is an instance of* ***cars***

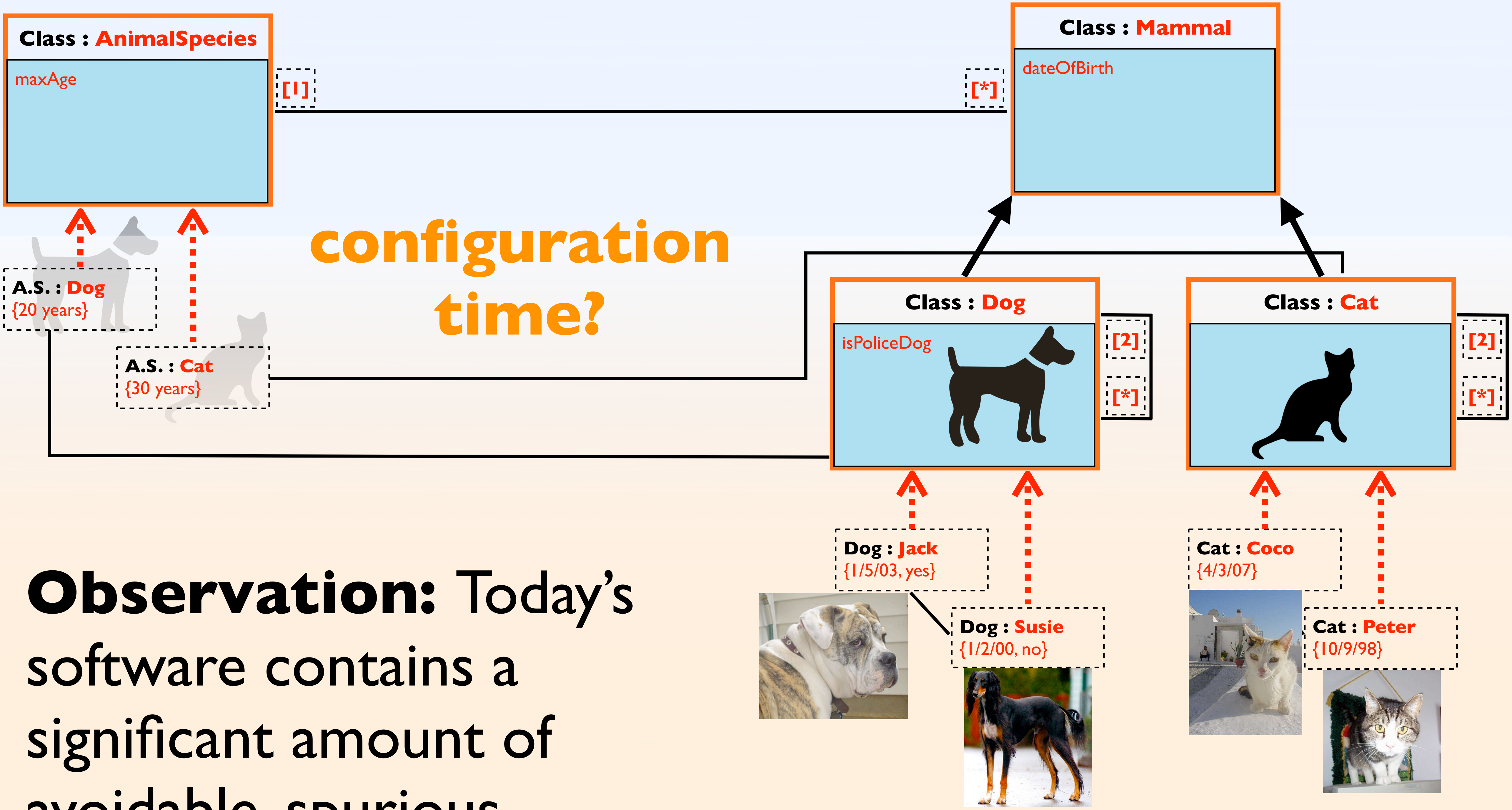
↑ ↓ **sales person**



ABC 123 *is an instance of* ***Golf***

Pragmatic kludge: The **Power Type** pattern

design time

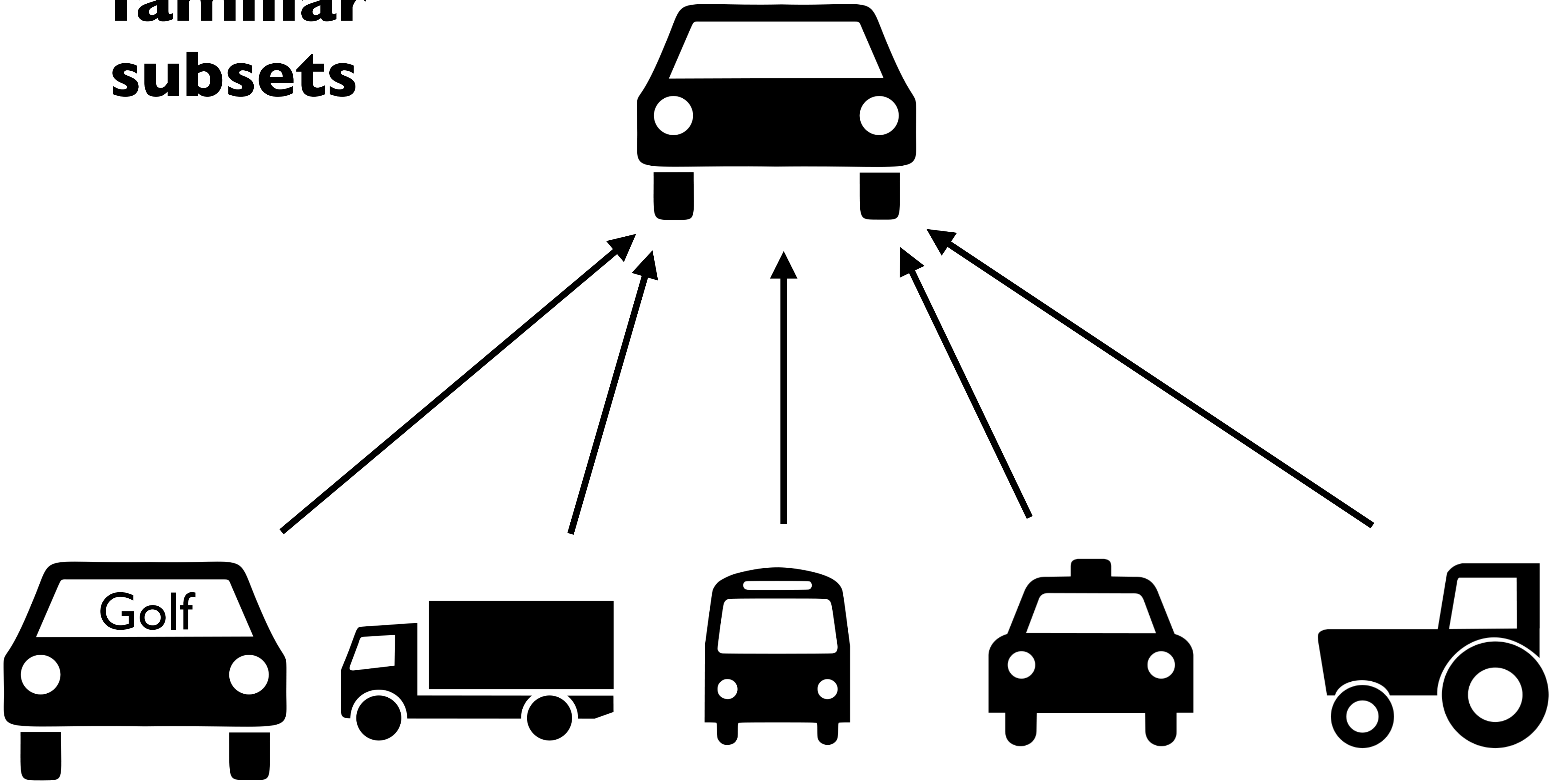


configuration time?

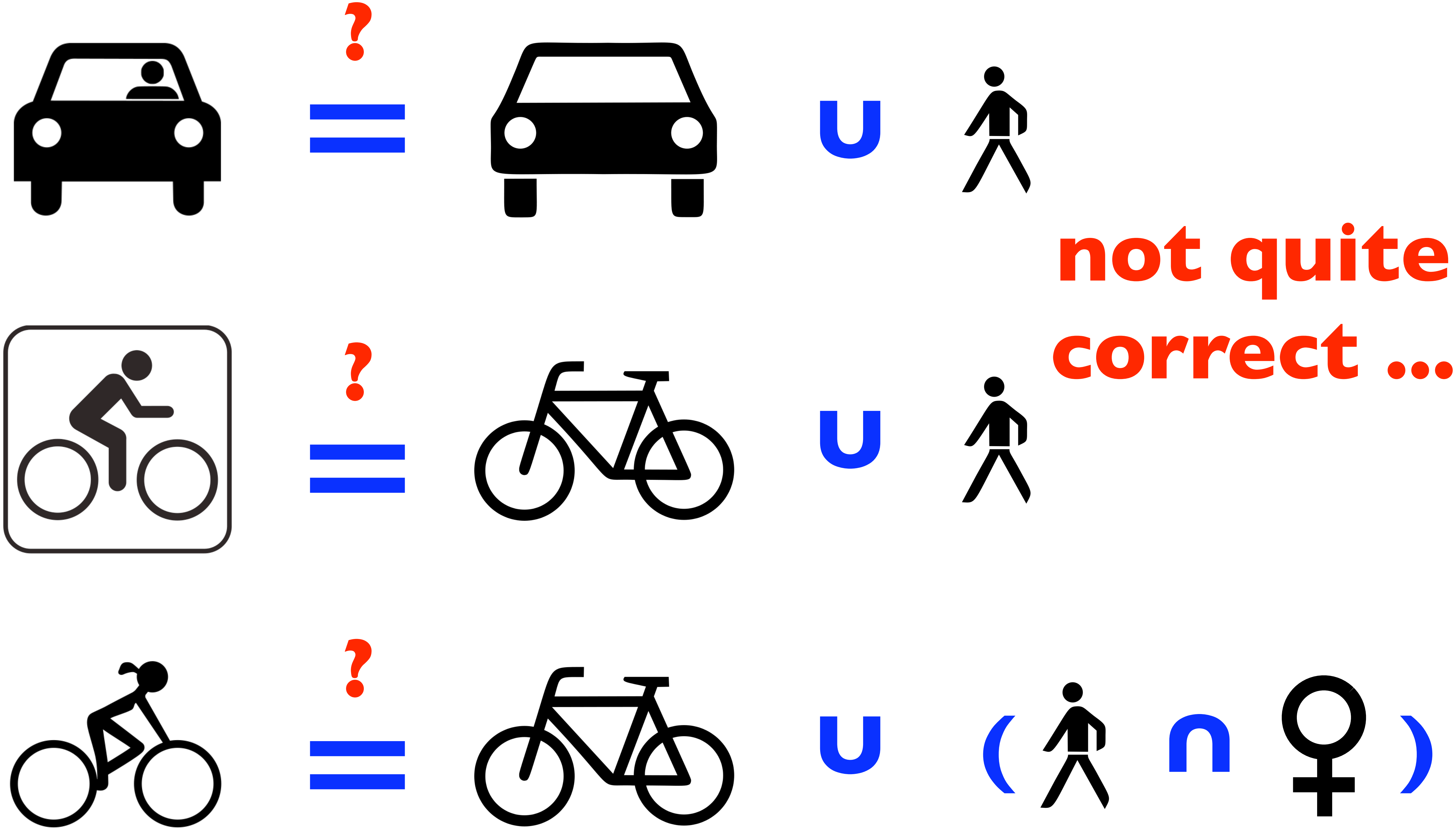
Observation: Today's software contains a significant amount of avoidable, spurious complexity

run time

Some familiar subsets

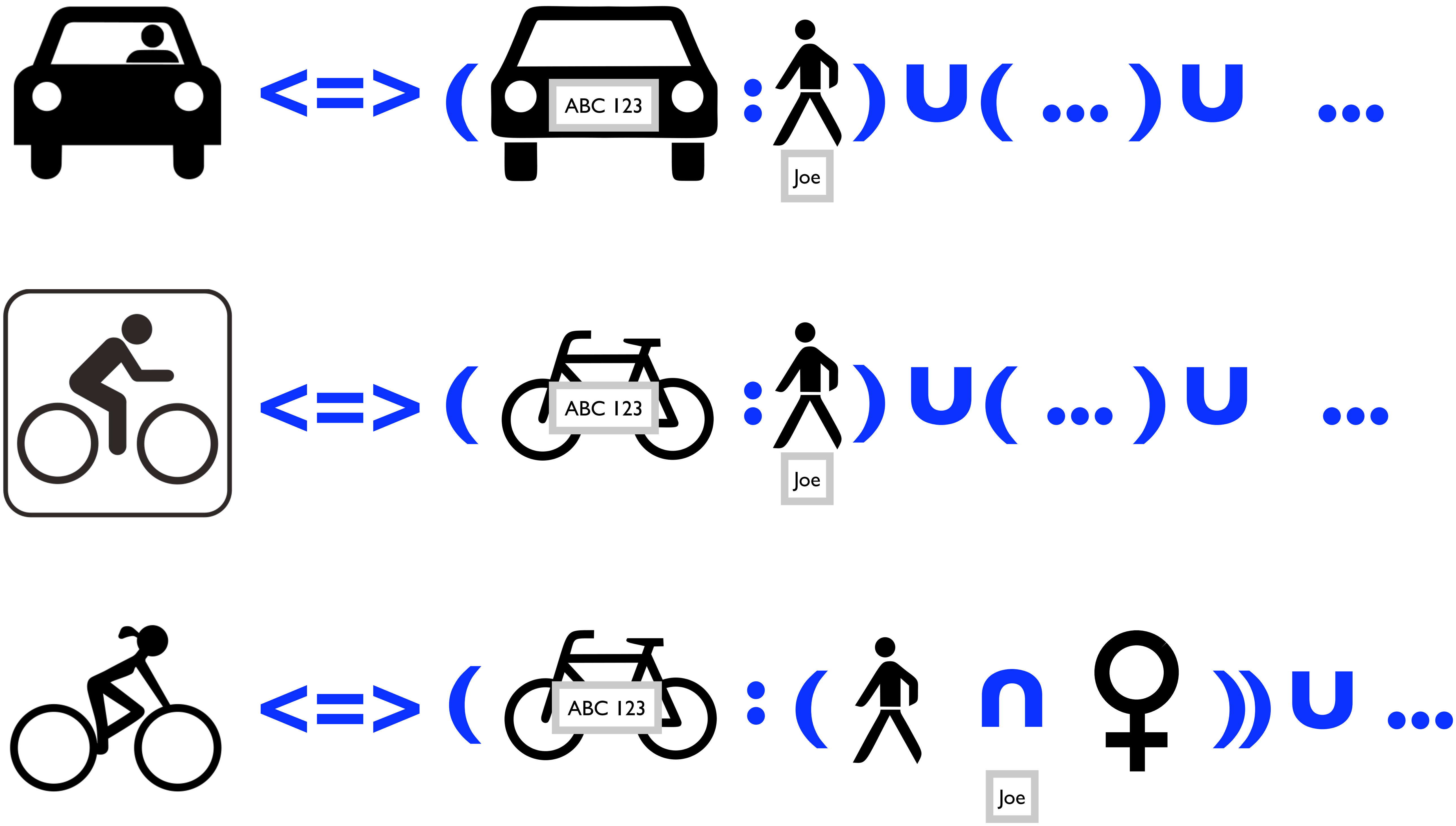


Notation matters, often less is more



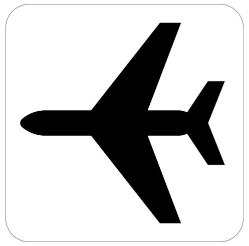
not quite correct ...

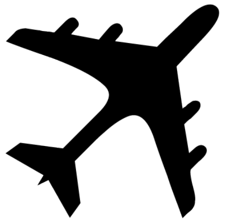
Potentially useful semantics

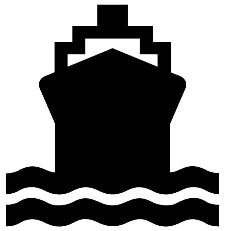


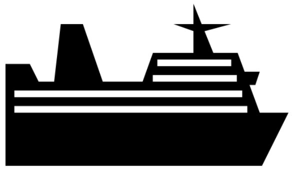
<=> (ABC 123 : Joe) U (...) U ...
<=> (ABC 123 : Joe) U (...) U ...
<=> (ABC 123 : (Joe ∩ ♀)) U ...

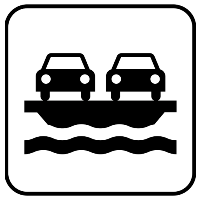
We constantly rely on speculative interpretation

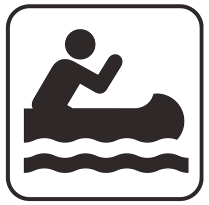

<=> airplane or aircraft ?


<=> commercial aircraft ?


<=> ship or boat ?

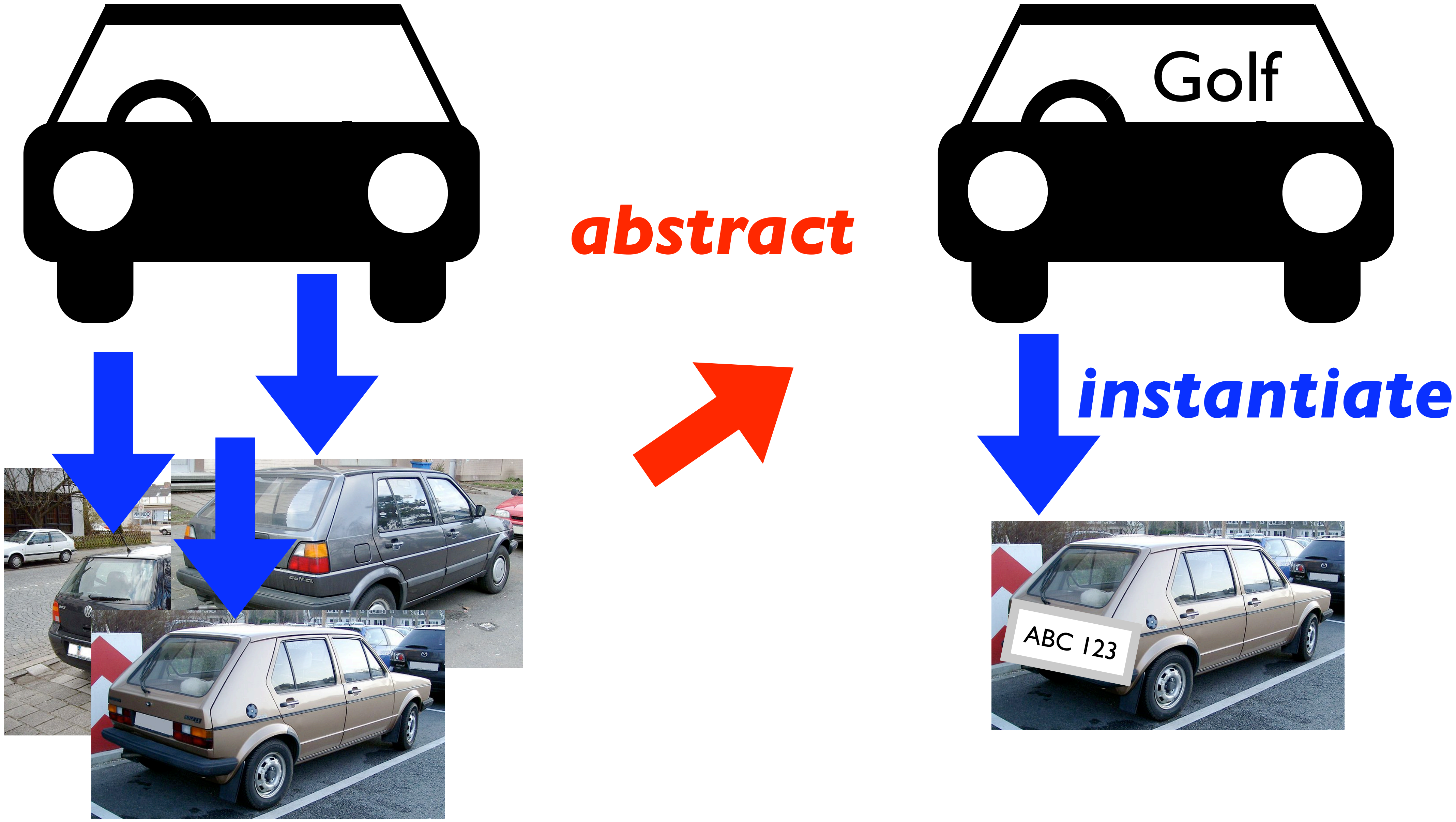

<=> ferry or cruise ship ?


<=> car ferry ?


<=> paddler or boat ?

Observation: It works 80% of the time

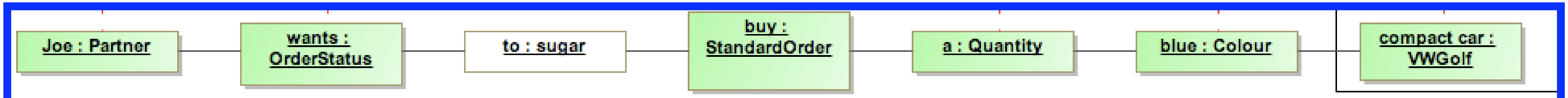
Perhaps 80% is not quite good enough for software specification !


abstract

instantiate

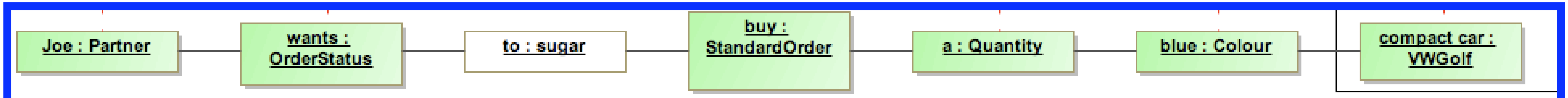
Observation: We need less **speculation** and much more **validation** via **instantiation !**

Is natural language any better?



use case step (end user needs)

Is code any better?



use case step (end user needs)

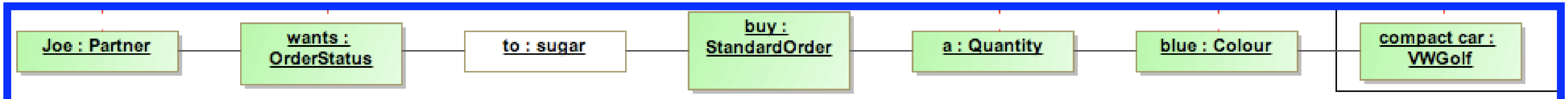
software design (voodoo)

```
whizBangTech.createStandardOrder(whizBangTech.createPartner(“Joe”), vwGolf, blue, 1);
```

code (implementation)

The code is the design.

Yeah, right!



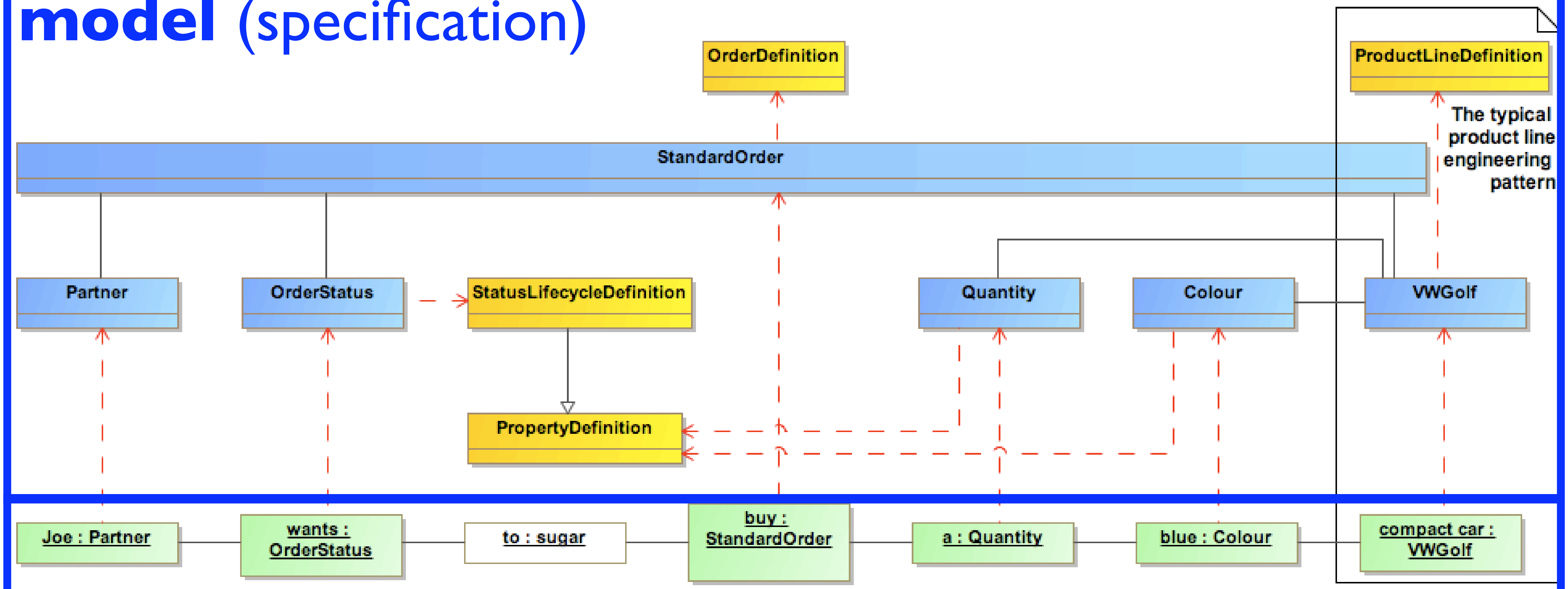
use case step (end user needs)

software design (voodoo)

```
whizBangTech.createStandardOrder(whizBangTech.createPartner(“Joe”), vwGolf, blue, 1);
```

code (implementation)

model (specification)

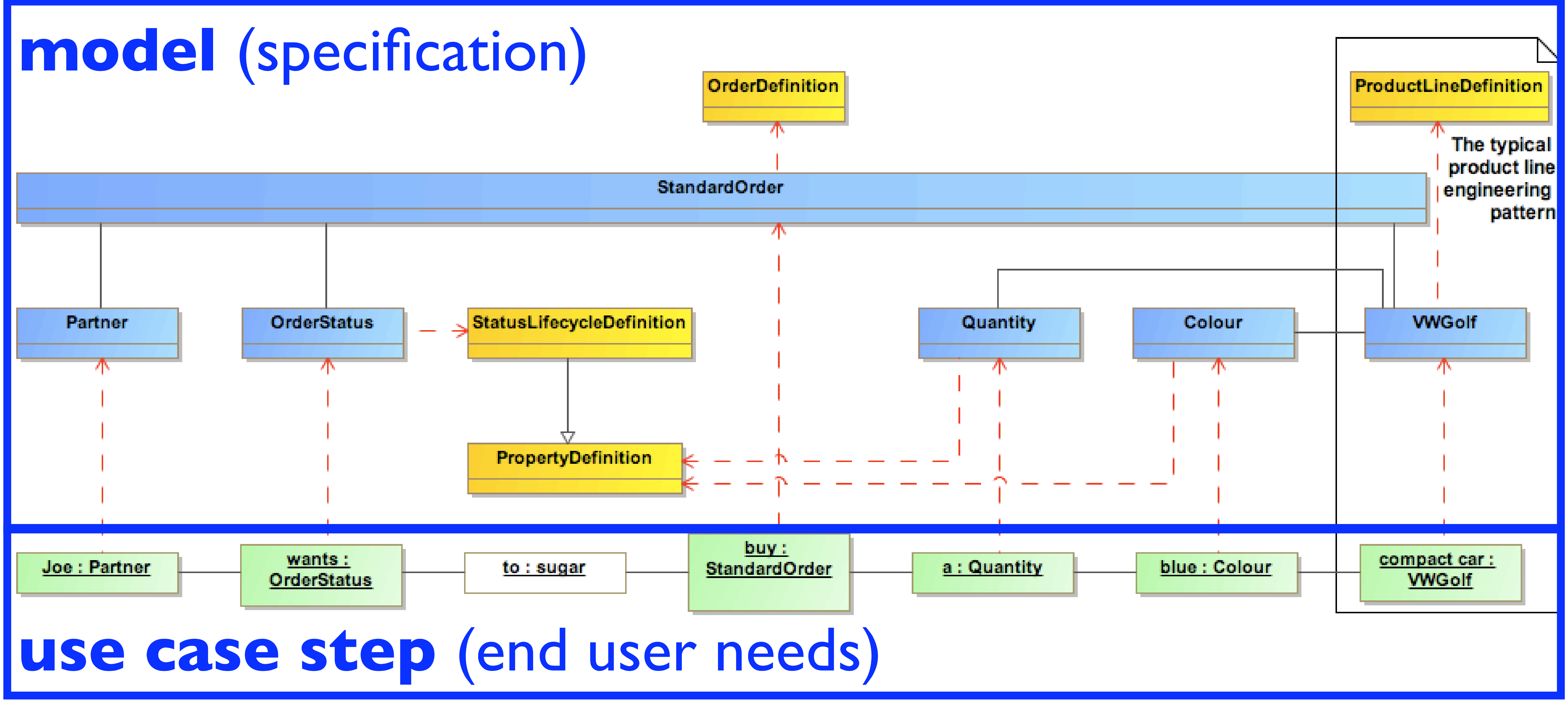


use case step (end user needs)

Automation (generation, execution)

```
whizBangTech.createStandardOrder(whizBangTech.createPartner("Joe"), vwGolf, blue, 1);
```

code (implementation)



model (specification)

use case step (end user needs)

The six essential domain analysis questions

1. **How often does a decision require revision?** -- determines whether a domain specific language is justified
2. **Who makes a decision** [about creating or changing characteristics of a <domain concept>]? -- determines **the role associated with the language** (the users of the language)
3. **When is the decision made?** -- determines **the process of using the language** and the binding time
4. **Where is the decision made** [in which work product]? -- leads to clues for a **good name for the language**
5. **What are the possible choices?** --- leads into **the details of the language definition**
6. **What heuristics apply?** --- determines **how to map the language to the underlying implementation**

Colour key

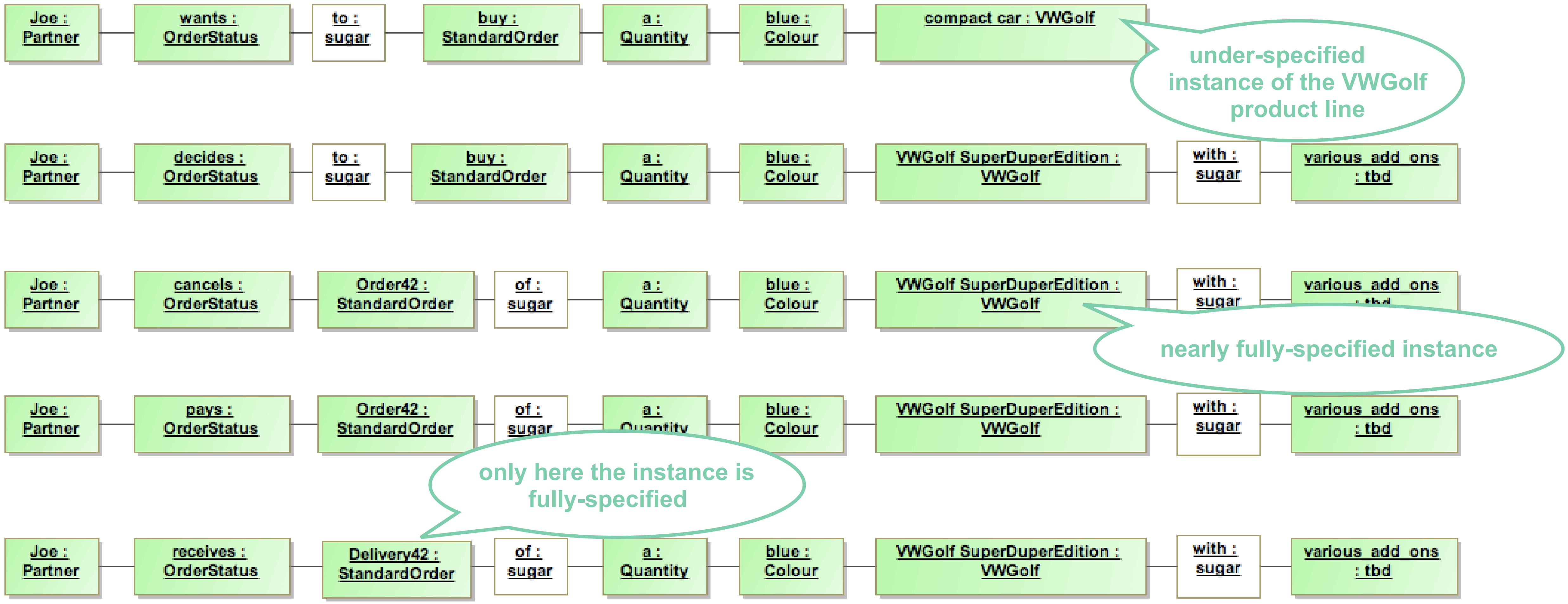
🟩 **Instantiated domain concept**

🟦 **Domain concept**

🟨 **Root element of a domain specific language definition**

Instantiation links - - - - - - - - - - >

Observation: Instantiation links do not adhere to the simplistic rules of the traditional class/object paradigm ...



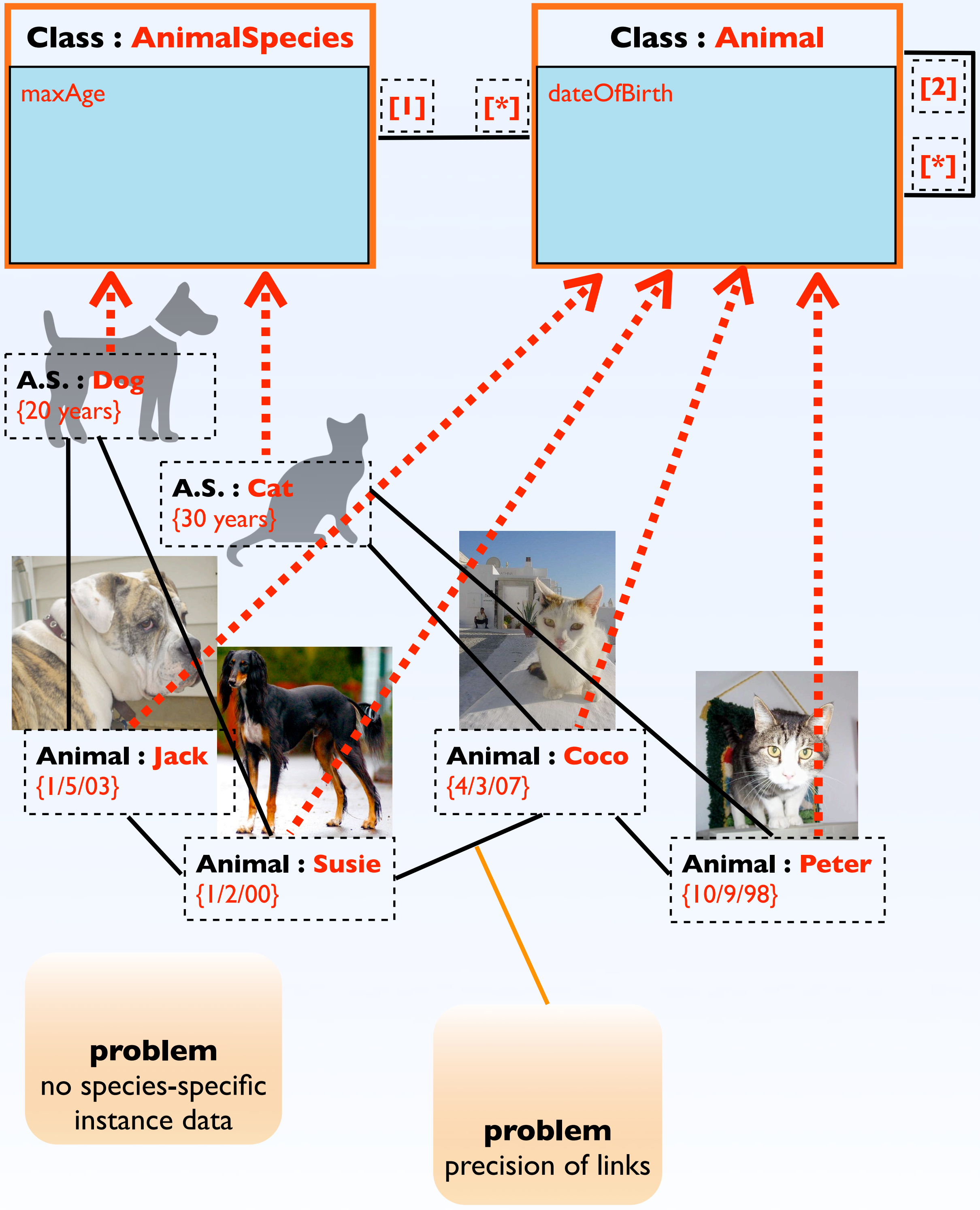
SOFISMO
Your code pilot

Observation:
Partial, incremental instantiation is common in product lines

A typical problem

object oriented model **A**

(modelling instantiation via **association**)

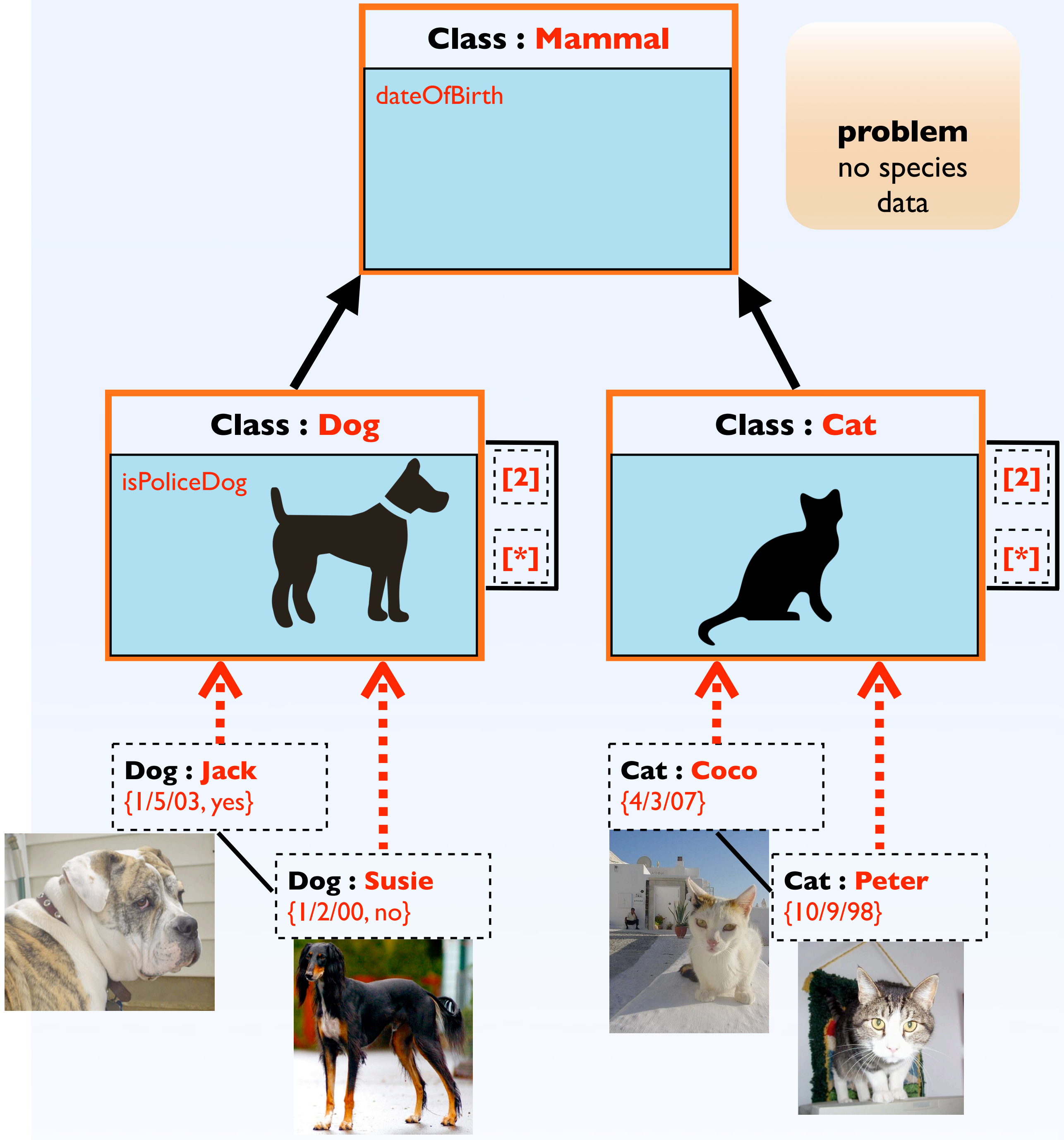


problem
no species-specific
instance data

problem
precision of links

object oriented model **B**

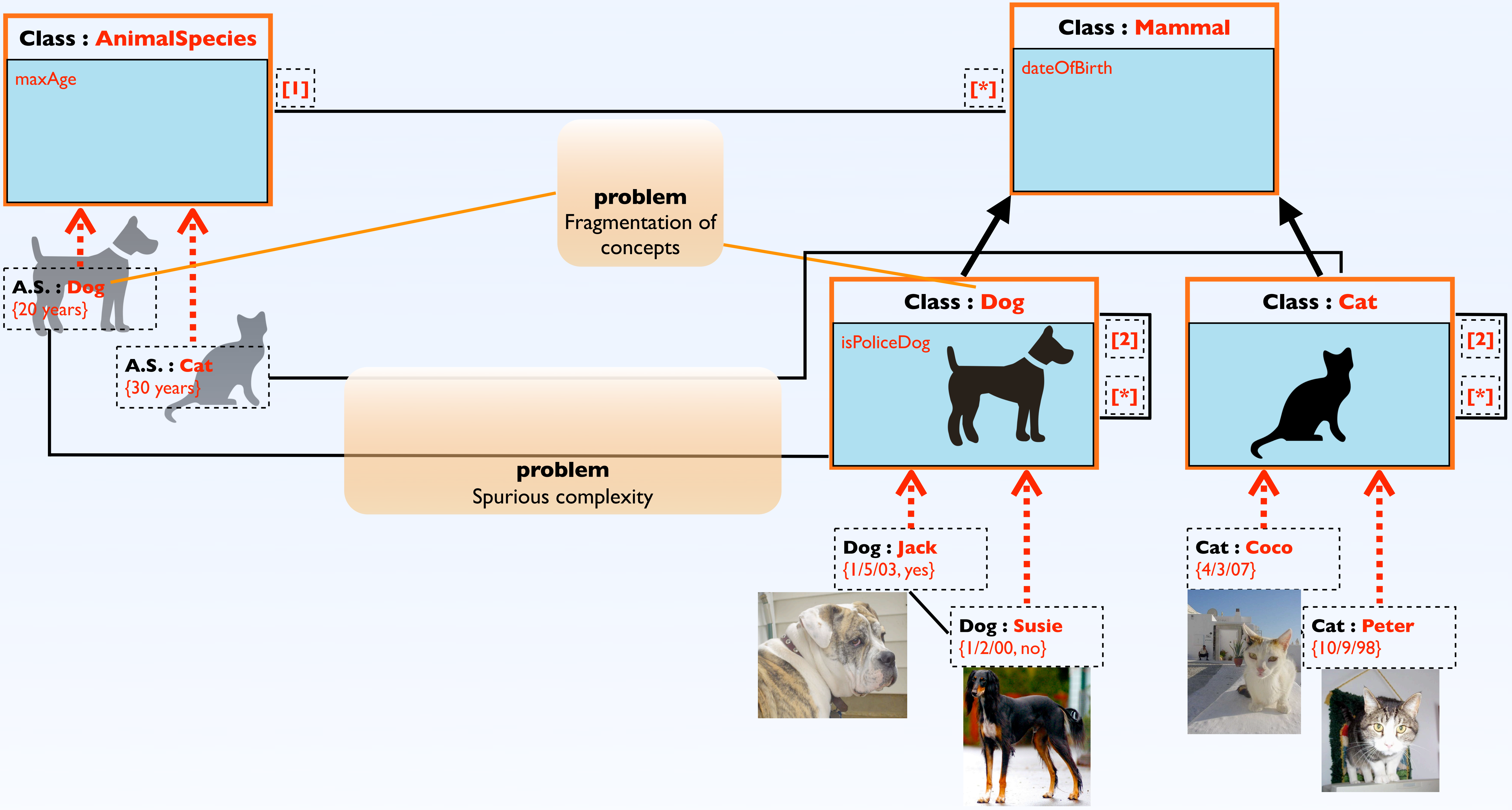
(modelling instantiation via **specialisation**)



problem
no species
data

object oriented model **C**

(**Power Type** pattern)



Class : AnimalSpecies

maxAge

[1]

[*]

Class : Mammal

dateOfBirth

problem
Fragmentation of concepts

A.S. : Dog
{20 years}

A.S. : Cat
{30 years}

Class : Dog

isPoliceDog

[2]

[*]

Class : Cat

[2]

[*]

problem
Spurious complexity

Dog : Jack
{1/5/03, yes}

Dog : Susie
{1/2/00, no}

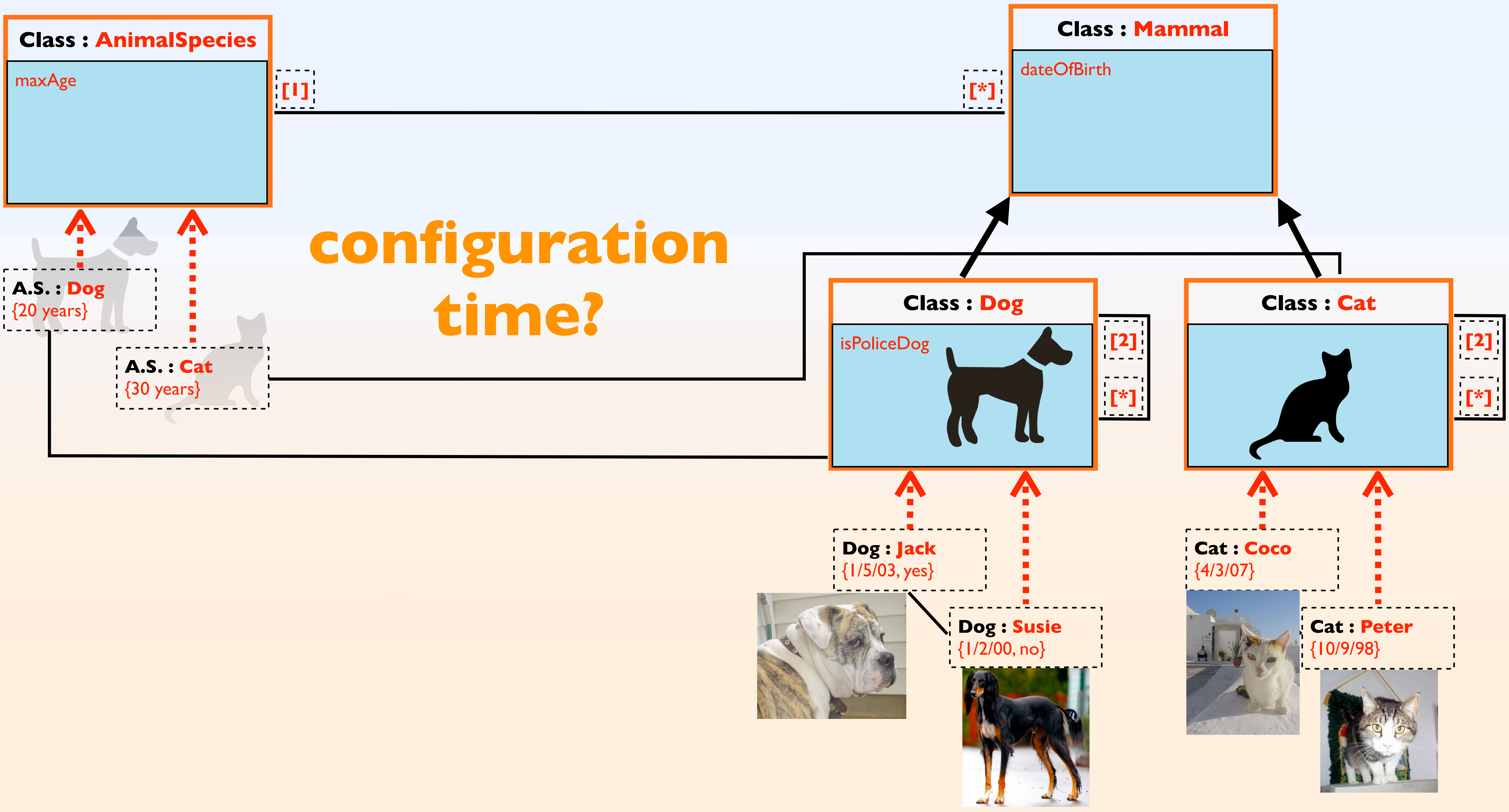
Cat : Coco
{4/3/07}

Cat : Peter
{10/9/98}

object oriented model **C**

(**Power Type** pattern)

design time



Class : AnimalSpecies

maxAge

[1]

[*]

Class : Mammal

dateOfBirth

A.S. : Dog
{20 years}

A.S. : Cat
{30 years}

configuration time?

Class : Dog

isPoliceDog

[2]

[*]

Class : Cat

[2]

[*]

Dog : Jack
{1/5/03, yes}

Dog : Susie
{1/2/00, no}

Cat : Coco
{4/3/07}

Cat : Peter
{10/9/98}

run time

The solution

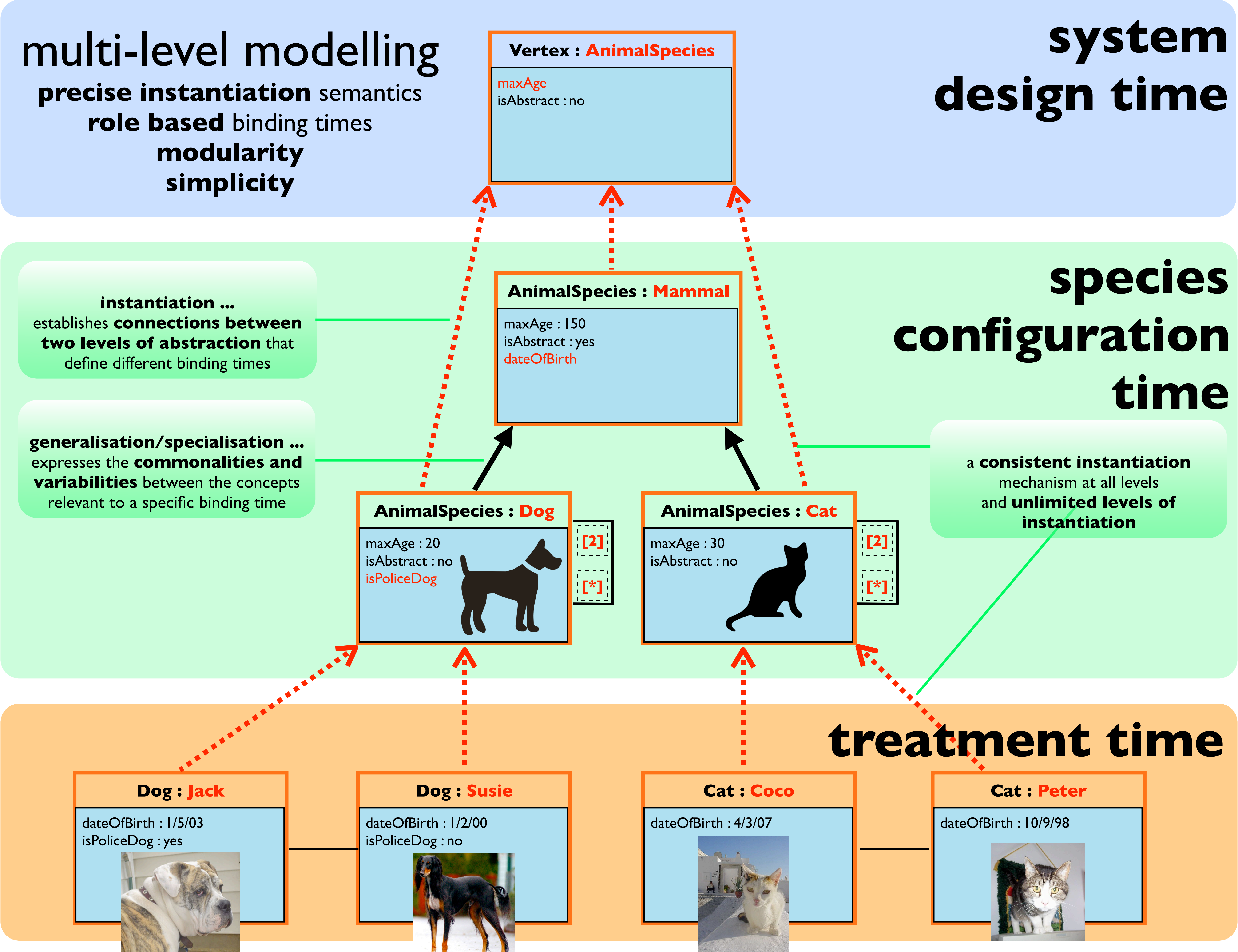
multi-level modelling

precise instantiation semantics

role based binding times

modularity

simplicity



system design time

Vertex : AnimalSpecies

maxAge
isAbstract : no

species configuration time

AnimalSpecies : Mammal

maxAge : 150
isAbstract : yes
dateOfBirth

AnimalSpecies : Dog

maxAge : 20
isAbstract : no
isPoliceDog

[2]

[*]

AnimalSpecies : Cat

maxAge : 30
isAbstract : no

[2]

[*]

instantiation ... establishes **connections between two levels of abstraction** that define different binding times

generalisation/specialisation ... expresses the **commonalities and variabilities** between the concepts relevant to a specific binding time

a **consistent instantiation** mechanism at all levels and **unlimited levels of instantiation**

treatment time

Dog : Jack

dateOfBirth : 1/5/03
isPoliceDog : yes

Dog : Susie

dateOfBirth : 1/2/00
isPoliceDog : no

Cat : Coco

dateOfBirth : 4/3/07

Cat : Peter

dateOfBirth : 10/9/98

More Information

The role of artefacts	tiny.cc/artefacts
Model Oriented Domain Analysis	tiny.cc/domainanalysis
Multi-Level Modelling	tiny.cc/gmodel
SEMAT	tiny.cc/sematpos_jbe, tiny.cc/sematslides_jbe
Denotational Semantics	tiny.cc/densem

Thank you!

Jorn Bettin
jbe @ sofismo.ch