Object Databases Data Model

Mathematical representation

Matematikos
ir informatikos
fakultetas

Arūnas Janeliūnas

Things out of model

Infinite sets of:

- object identifiers **obj** = { o_1 , o_2 , … };
- class names **class** = { c_1 , c_2 , … };
- attribute names **att** = { a_1 , a_2 , … };
- method names **meth** = { m_1 , m_2 , . . . }.

Types

Atomic data types

- Long,
- Short,
- Unsigned long,
- Unsigned short,
- Float,
- Double,
- Boolean,
- Octet,
- Char,
- String,
- Enum.

Values of those types constitute a set denominated by **dom**.

Values (*literals*)

Given a set $O \subset$ **oid**, the set of values over O is defined as:

1. *nil* is a value over O;
2. all values from **dom** are values over O;
3. all elements from O are values over O;
4. if $v_1 , \dots , v_n$ are values over O and $a_1 , \dots , a_n$ are attribute names from **att**, then the tuple $[a_1 : v_1 , \dots , a_n : v_n]$ is a value over O;
5. if $v_1 , \dots , v_n$ are values over O then the collection $\{v_1 , \dots , v_n\}$ is a value over O.

The set of values over O is denoted by **val**(O).

Value examples

```
1,

„Some Value",

oid12,

[ cinema: oid12,
  time: "16.30",
  price: nil,
  movie: oid4
],

{ "G.Massina", "S.Loren", "M.Mastroianni" },

[ title: "La Strada",
  director: "F.Fellini",
  actors: {oid25, oid14, oid51}
]
```

Values

Objects

Objects

Object is a pair <*id, val*>, where *id* is an element of **oid**, and *val* is a value of the form of a tuple or a collection

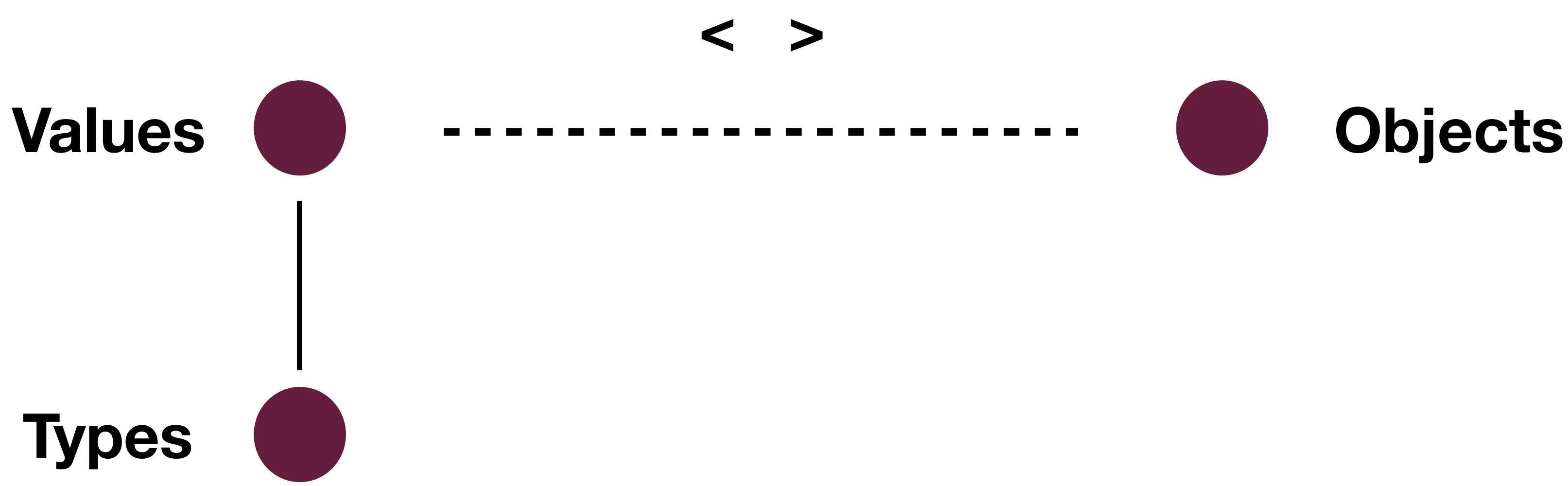
<   >

Values ● - ● **Objects**

Object examples

```
< oid123, { “G.Massina”, “S.Loren”, “M.Mastroianni” } >,

< oid672354, [ title: “La Strada”,
               director: “F.Fellini”,
               actors: {oid25, oid14, oid51}
             ]
>
```


< >
Values
Objects
Types

Types

Given the set of class names $C \subset$ **class**, types over C are defined as:

- class name **any** is a type over C;
- all atomic types (**short, long, unsigned short** ir t.t.) are types over C;
- class names from C are types over C;
- if t_1 , … , t_n are types over C and a_1 , … , a_n and a_1 , … , a_n are attribute names from **att**, then the tuple $[a_1 : t_1 , \ldots , a_n : t_n]$ is a tuple type over O
- if t is a type over C then $\{t\}$ is a collection type over C.

All types over C are denoted by **types**(C).

Collections

ODMG data model has several types for collections:

- *Set*;
- *Bag* (multi-set);
- *List* (has an order in it);
- *Array*.

Tuple types

ODMG data model also has several predefined tuple types:

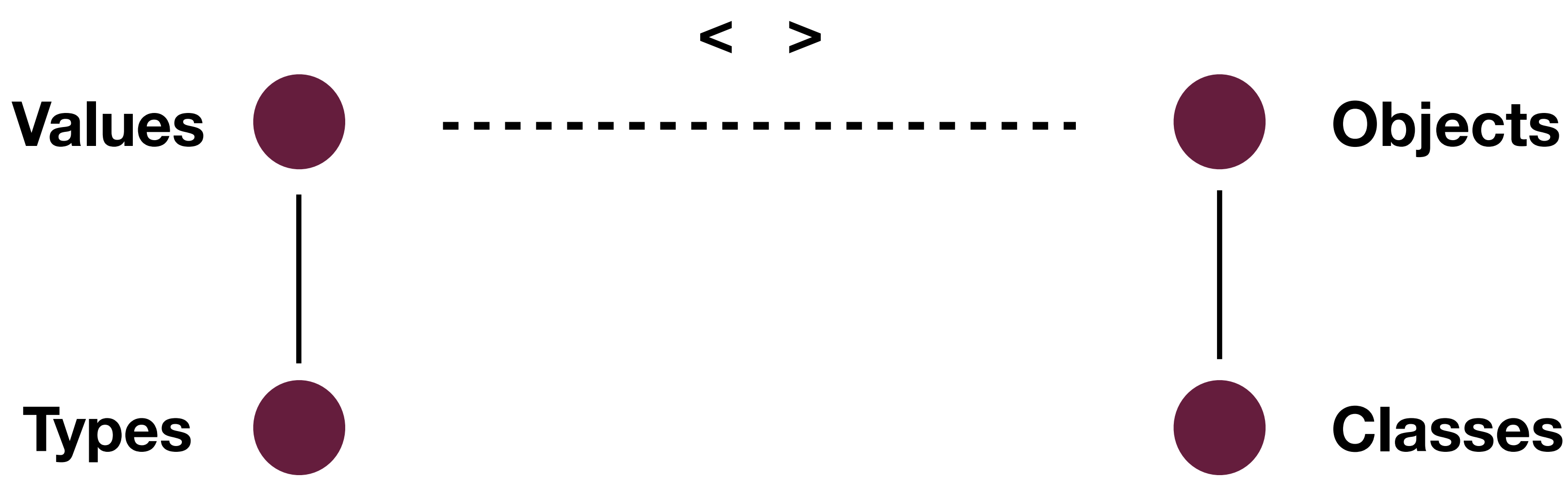
- *Date*;
- *Interval*;
- *Time*;
- *Timestamp*.

Type examples

```
Cinema,  // class name

{ Time },

[ cinema: Cinema,
    time: String,
   price: Short,
   movie: Movie   // yet another class name
]
```


< >
Values
Objects
Types
Classes

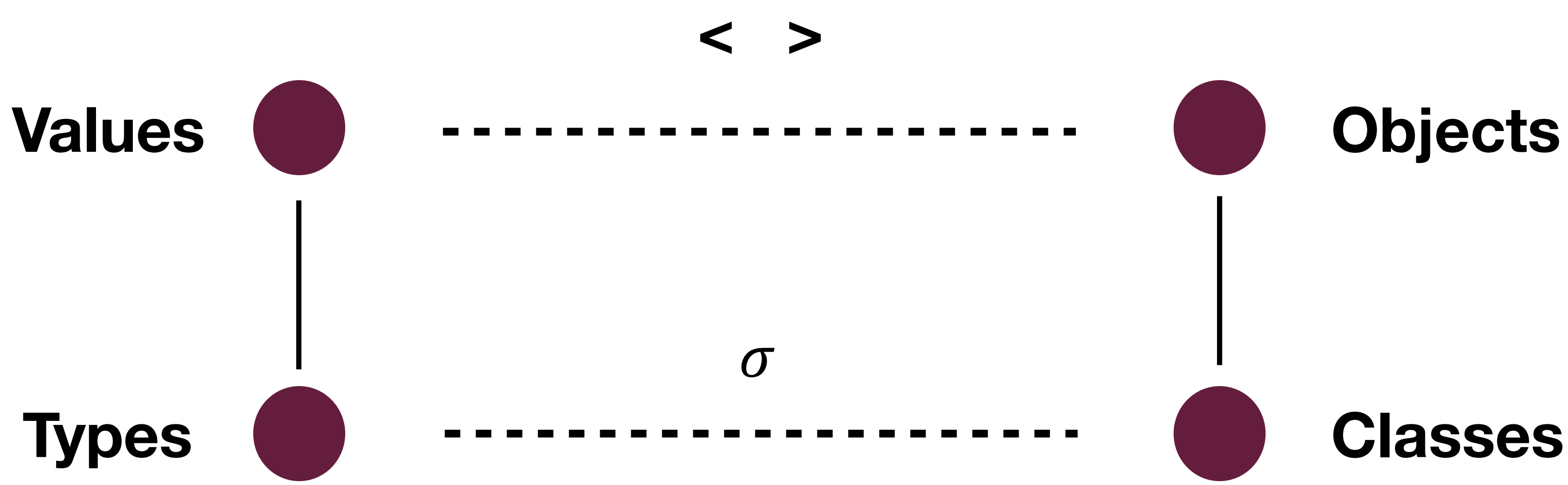
Classes

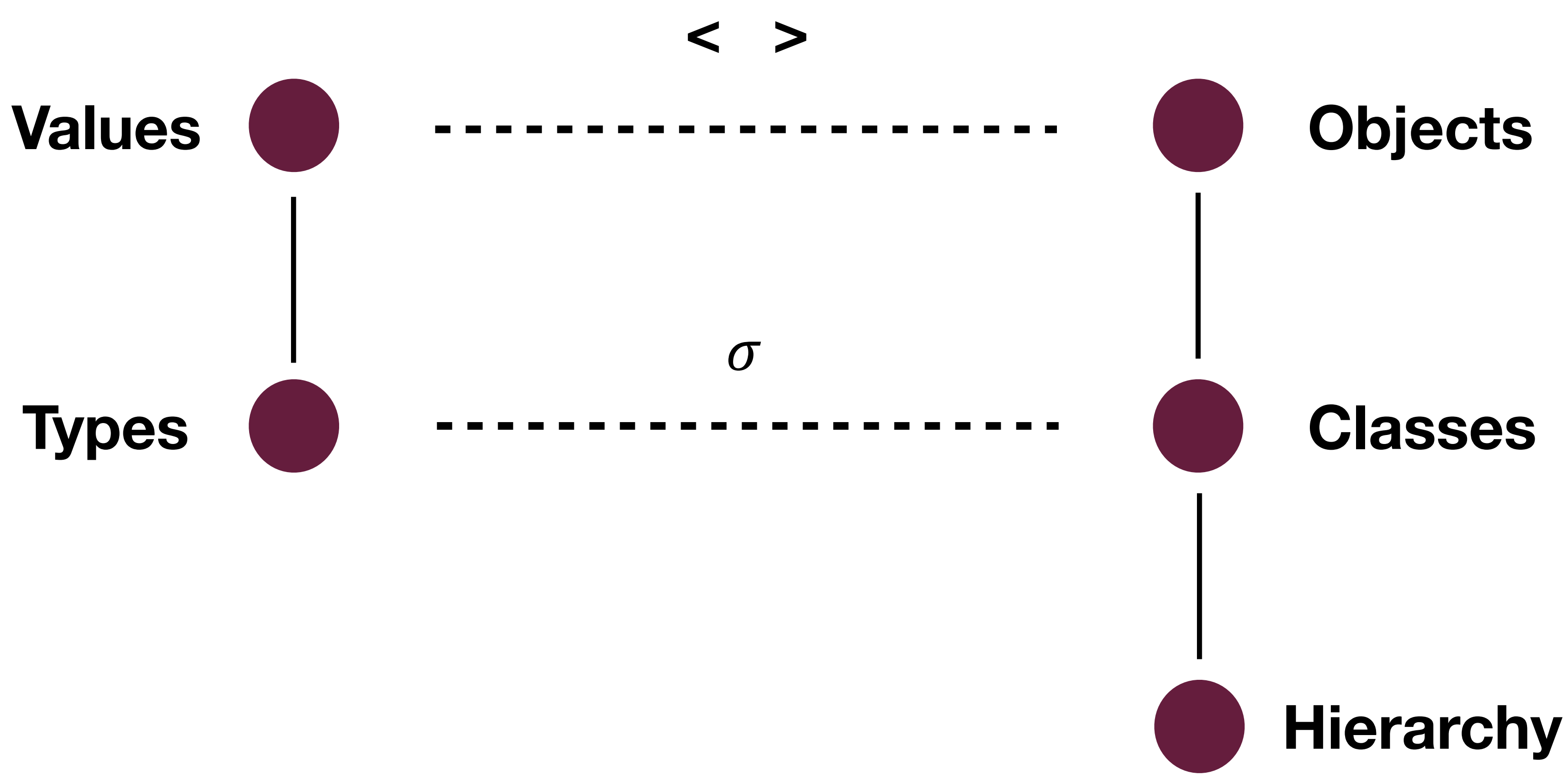
Class is a set of objects holding inside values of the same type.

Classes / types

If C is a set of class names $C \subset$ **class**, then $\sigma(C)$ is a function

$$\sigma : C \rightarrow \mathbf{types}(C)$$


< >
Values
Objects
σ
Types
Classes



< >
Values
Objects
σ
Types
Classes
Hierarchy

Class hierarchy

Class hierarchy is a triplet $< C, \sigma, < >$, where:

- C is a finite set of class names,
- $\sigma : C \rightarrow \textbf{types}(C)$,
- $<$ is a partial order relationship in the set C.

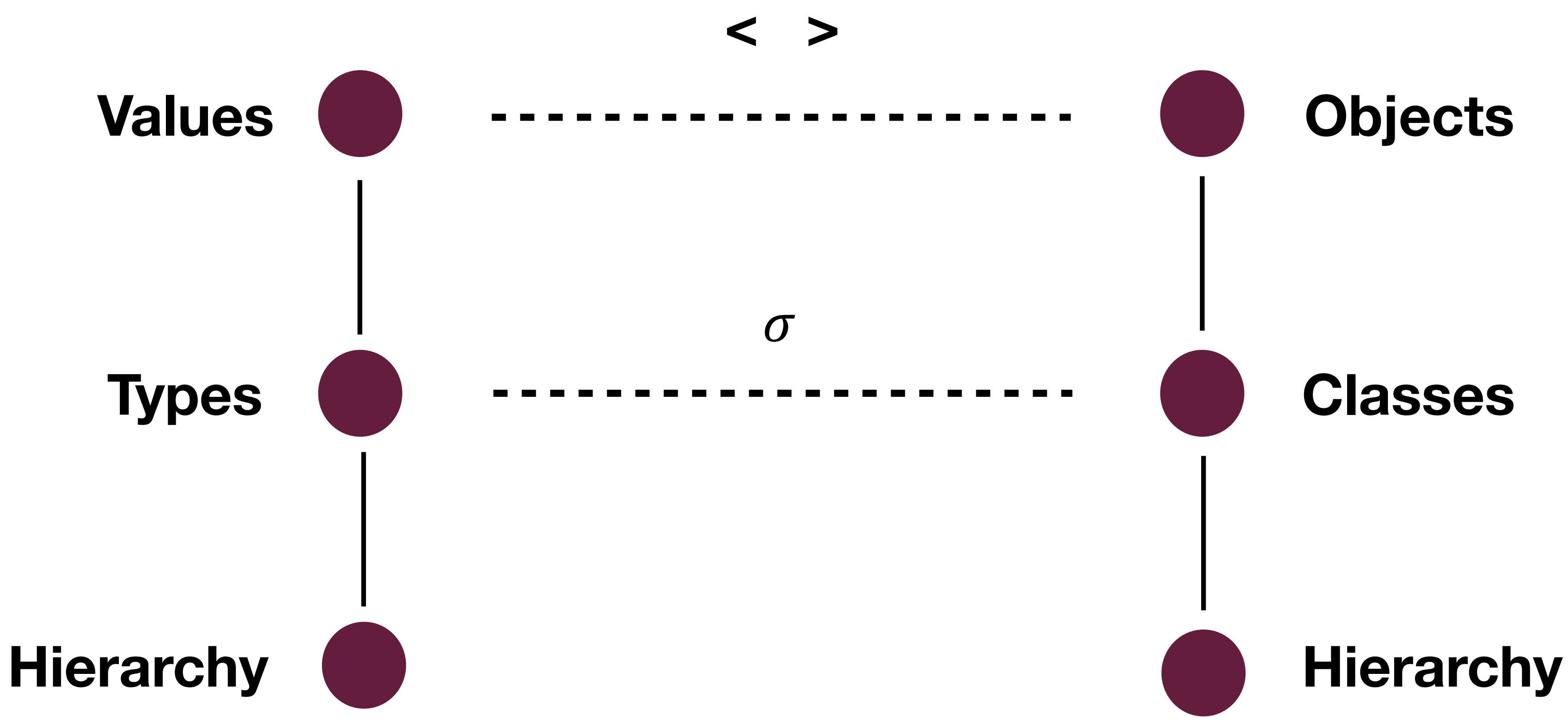
Transitional and non-comutative relationship in the set is called an *order*. The order relationship in the set which exists between any given pair of the set elements is called *total order* and *partial order* otherwise.

Class hierarchy

Can you see $< C, \sigma, < >$ here?

```
class Person {
  String name;
  Integer age;
};

class Lecturer extends Person {
  String title;
};
```



< >
Values
Objects
σ
Types
Classes
Hierarchy
Hierarchy

Type hierarchy

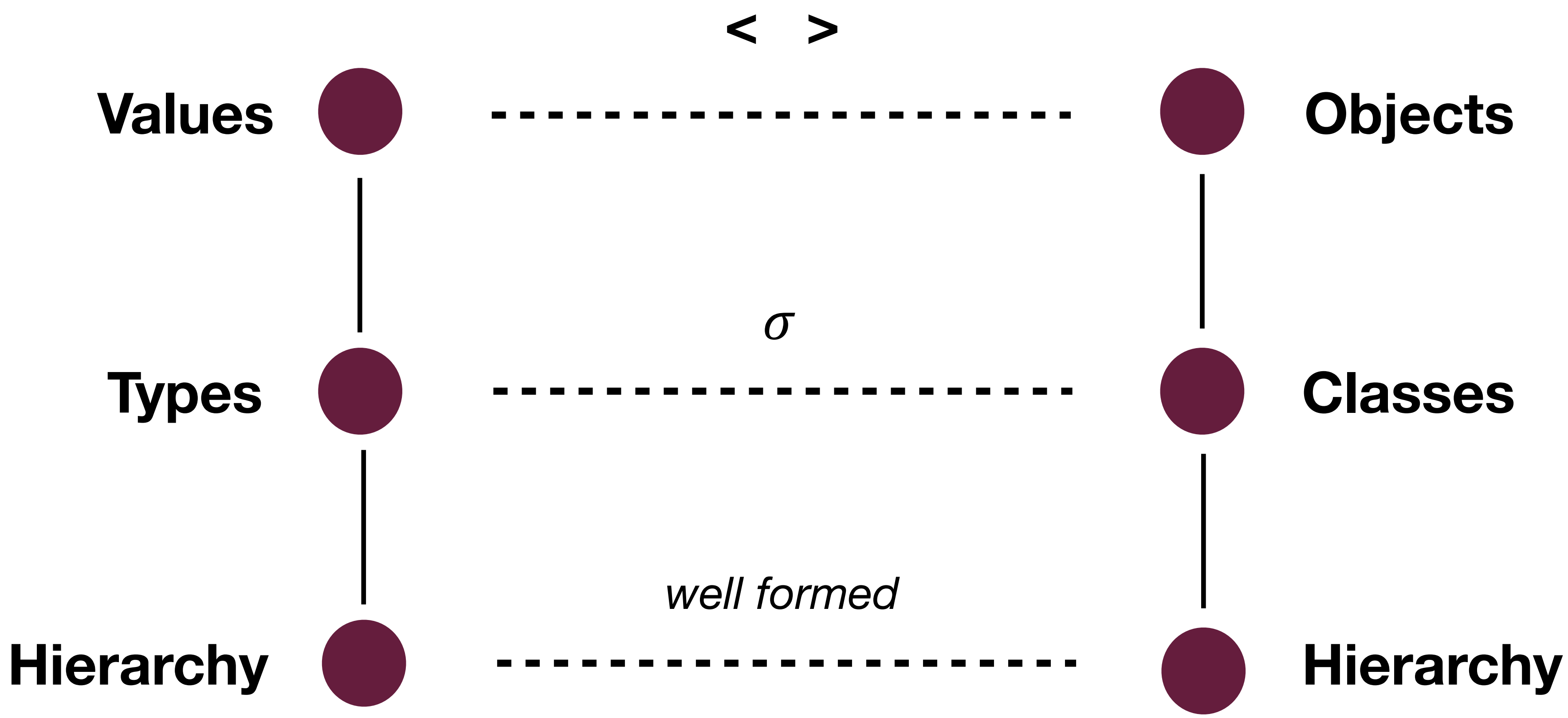
Let $< C, \sigma, < >$ be a class hierarchy. Then the sub-type/super-type relationship $\leq$ is a partial order in the set **types**(C), described by the following rules:

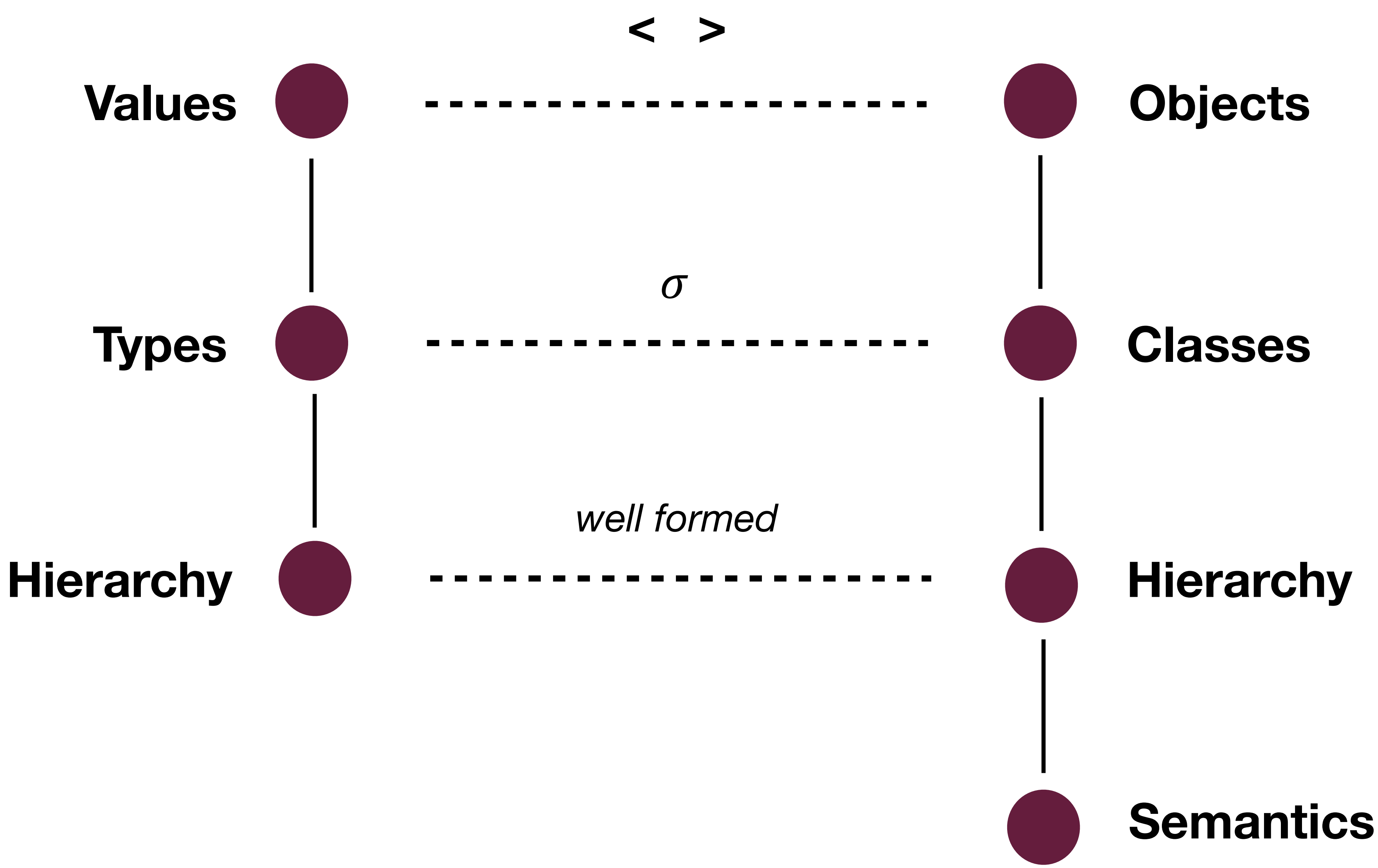
- $\forall\, t : t \leq$ **any** ,
- $c < k \;\Rightarrow\; c \leq k$,
- $(\forall\, i \in [1, n], n \leq m : t_i \leq t'_i) \;\Rightarrow\; [a_1 : t_1 , \ldots , a_m : t_m] \leq [a_1 : t'_1 , \ldots , a_n : t'_n]$,
- $t \leq t' \;\Rightarrow\; \{ t \} \leq \{ t' \}$.

Well formed structure

The class hierarchy $< C, \sigma, < >$ is called to be of a well formed structure if for any given pair of classes c and k

$$c < k \Rightarrow \sigma(c) \leq \sigma(k)$$


< >
Values
Objects
σ
Types
Classes
well formed
Hierarchy
Hierarchy



< >
Values
Objects
σ
Types
Classes
well formed
Hierarchy
Hierarchy
Semantics

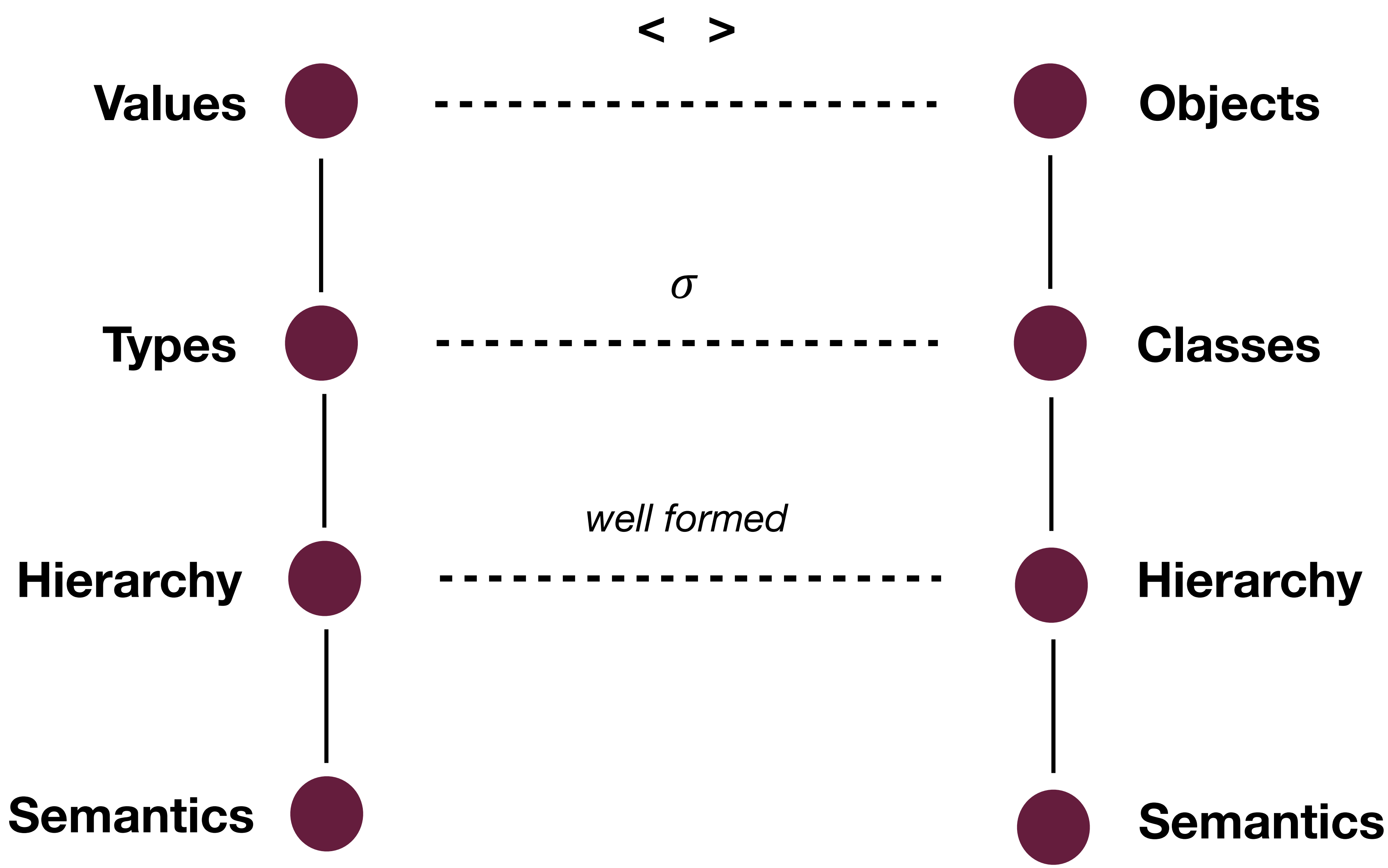
Semantics of the classes

Let $< C, \sigma, < >$ be a class hierarchy (of the well formed structure). *Oid assignment* is a function π which for every element of C assigns a particular set of object identifiers from **oid**.

Therefore $\pi(c)$ is called a *proper extent* of the class c.

The *extent* of the class c (denoted by $\pi^*(c)$) is a set

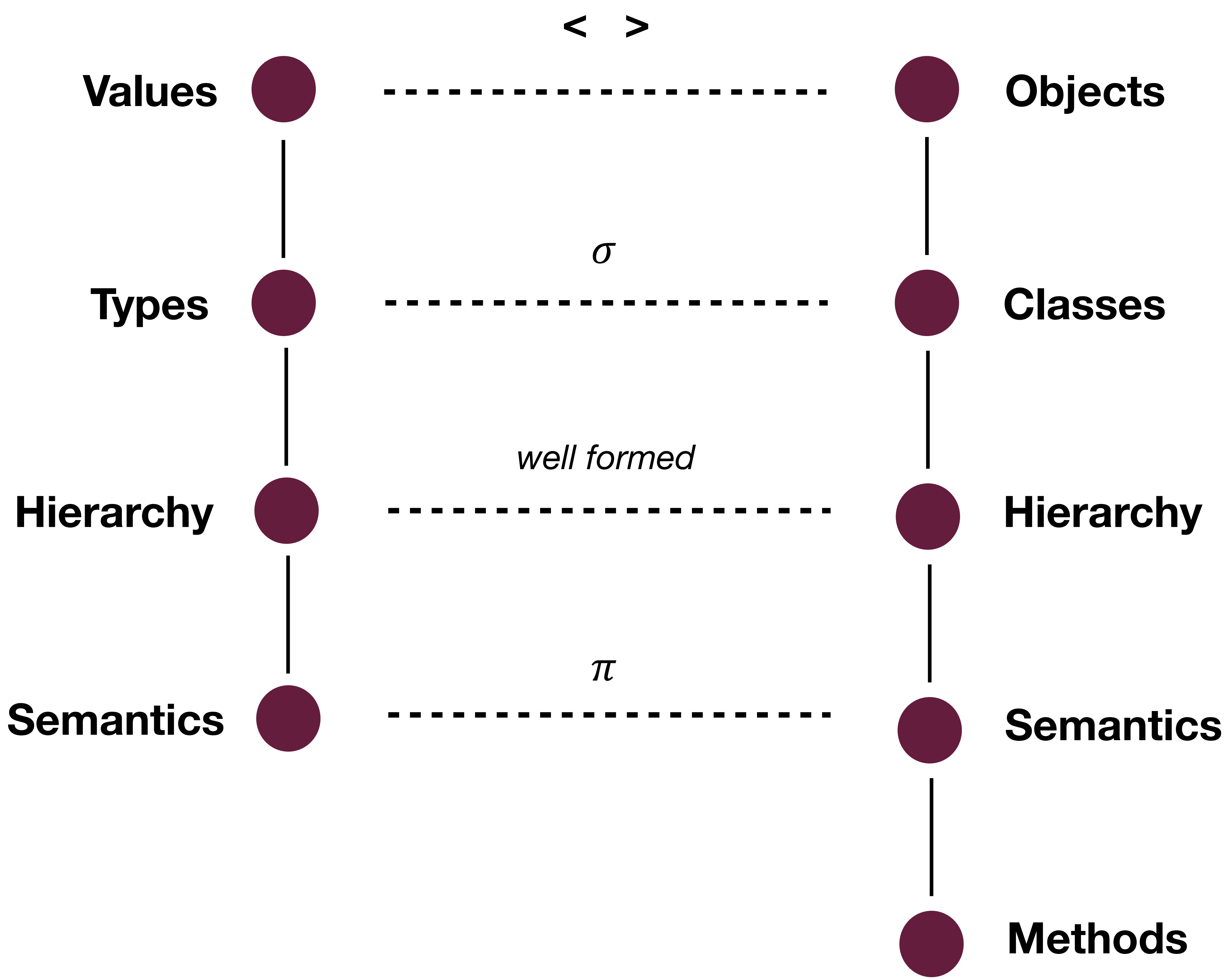
$$\pi^*(c) = \bigcup_{k} \{ \pi(k) : k = c \vee k < c \}$$


< >
Values
Objects
σ
Types
Classes
well formed
Hierarchy
Hierarchy
Semantics
Semantics

Semantics of the types

Let $< C, \sigma, < >$ be a class hierarchy and $O = \bigcup\{ \pi^*(k) : k \in C \}$. Then we can derive that $O = \pi^*(\boldsymbol{any})$. And then the *type interpretation* **dom**(t) of the type t is defined by:

- **dom**(**any**) = **val**(O)
- for every atomic type t, **dom**(t) is it's „usual“ interpretation
- $\forall c \in C$: **dom**(c) = $\pi^*(c) \cup \{nil\}$,
- **dom**($\{t\}$) = $\{ \{v_1 , \dots , v_n\} \mid v_i \in$ **dom**(t) $\}$
- **dom**($[a_1 : t_1 , \dots , a_n : t_n]$) = $\{ [a_1 : v_1 , \dots , a_n : v_n] \mid v_i \in$ **dom**(t_i) $\}$


< >
Values
Objects
σ
Types
Classes
well formed
Hierarchy
Hierarchy
π
Semantics
Semantics
Methods

Methods

A method has 3 parts:

- name
- signature
- implementation

Given the method name $m \in$ **meth**, its signature is

$$\mathrm{m} : \mathrm{c} \times t_1 \times \ldots \times t_n \rightarrow t_{out}$$

where $c \in C$ ($< C, \sigma, < >$ being a class hierarchy) and t_i are the types over C (that is, $t_i \in$ **types**(C)).

Inheritance

Given two classes c and *k* such that

- method *m* is defined in the class *c*
- $k < c$
- does not exists such a class *p* that $k < p < c$,

then it is said that class *k* inherits the method *m* from the class *c*.

Inheritance

Given two methods

$$m : c \times t_1 \times \ldots \times t_n \rightarrow t_{out}$$

and

$$m : k \times t'_1 \times \ldots \times t'_k \rightarrow t'_{out}$$

where $k < c$, the following rules must be followed:

1. *Consistency*. If $k < c$ and $k < p$ without any sub-class relationship between p and c, and method m is defined in both classes p and c, method m must be explicitly defined in the class k as well.
2. *Covariation*. It must be $t'_i \leq t_i$ for every i, and $t'_{out} \leq t_{out}$ as well.

Database scheme

Database scheme is a quintuplet $\mathbf{S} = < C, \sigma, < , M, G >$, where:

- $< C, \sigma, < >$ is a class hierarchy
- M is a set of method signatures
- G is a set of names, such that $G \cap C = \varnothing$
- $\sigma : C \cup G \rightarrow \mathbf{types}(C)$