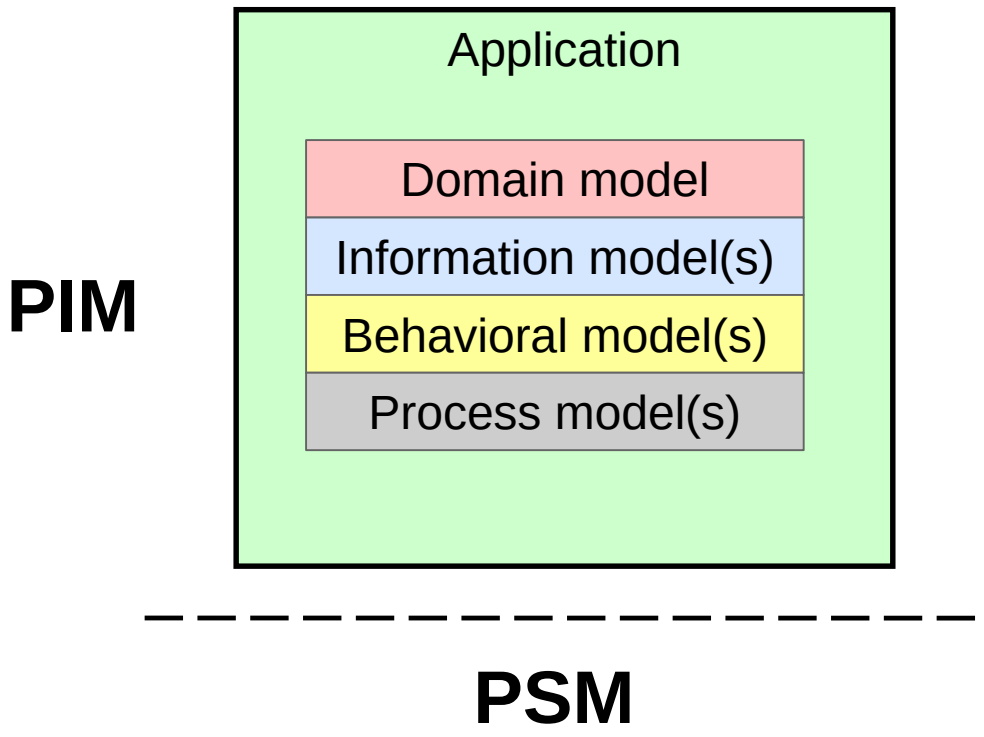
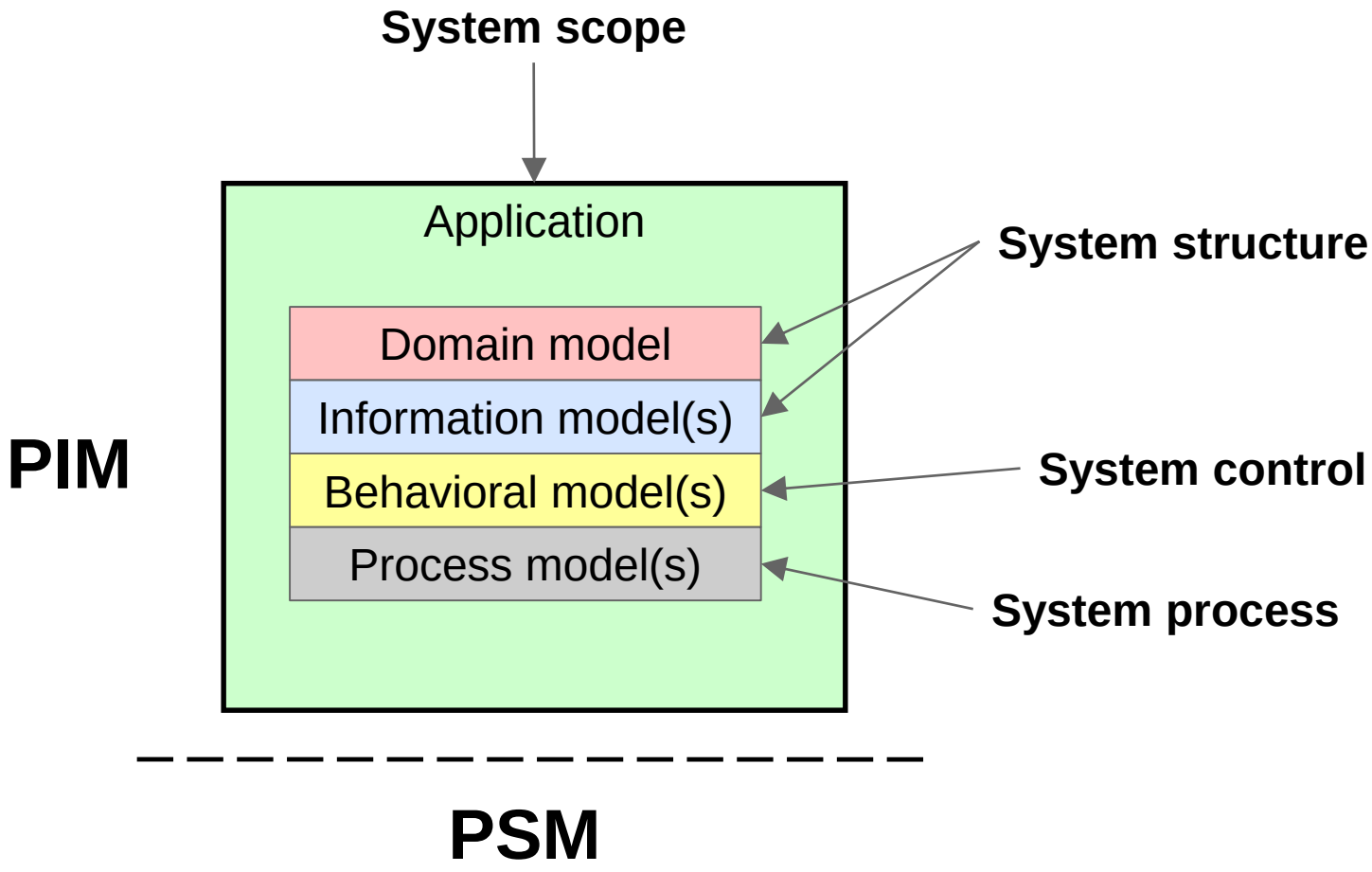
# TYPE SYSTEM WITH ABSTRACTION.

Nils Paulsson, PhD
Saab Aeronautics

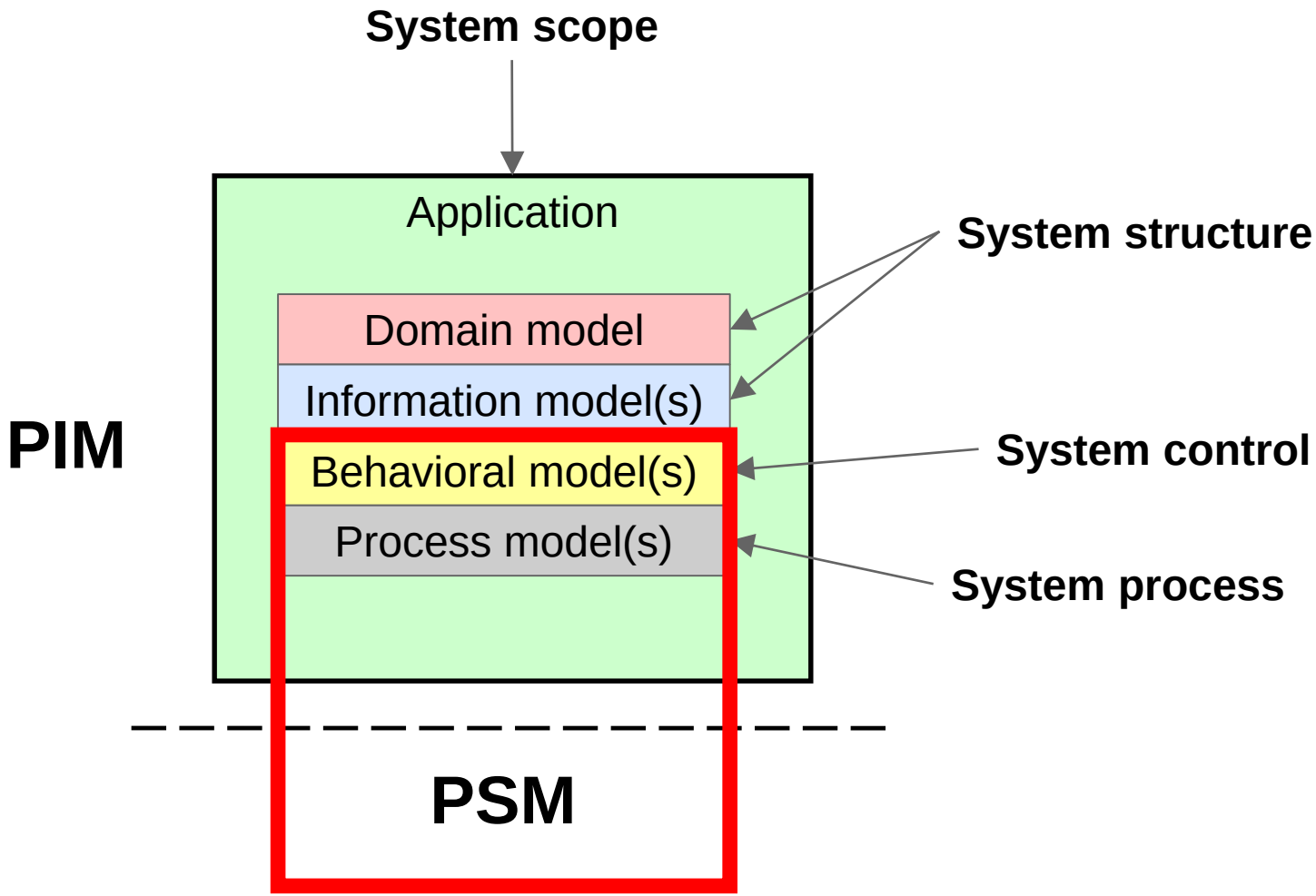
# THE XT-UML STRUCTURE



# THE XT-UML STRUCTURE



# COMMON TYPE SYSTEMS



# WHAT IS THE PROBLEM?

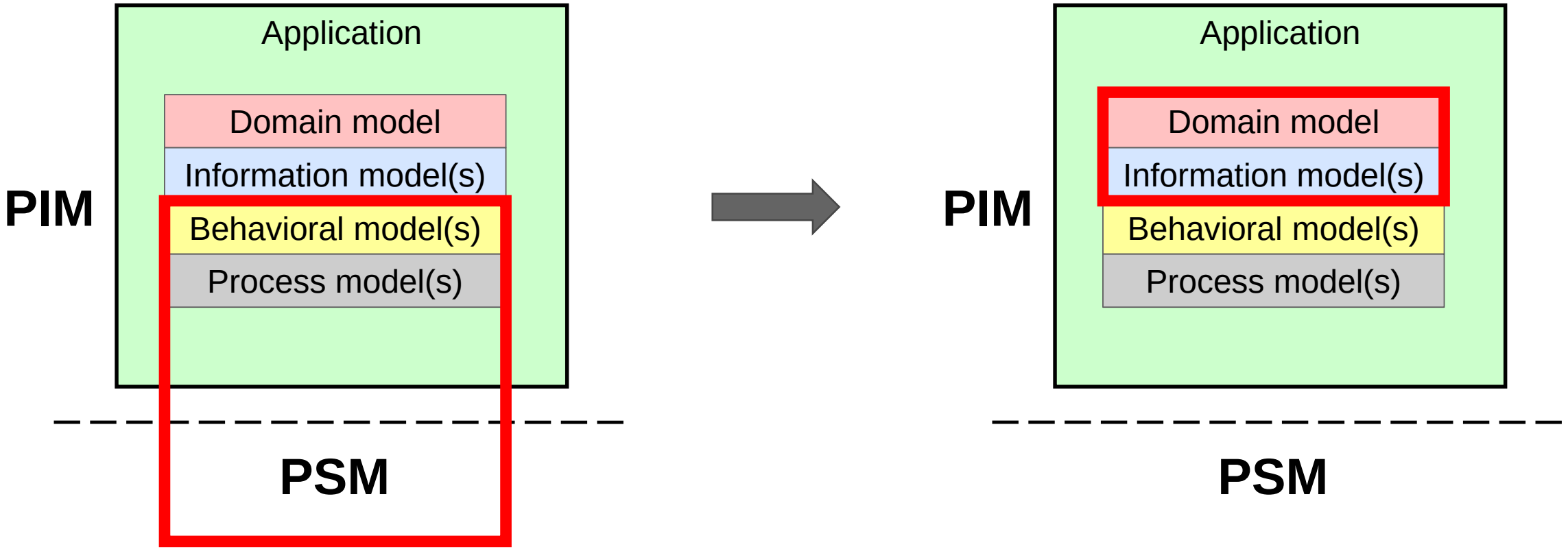
Common type systems are mixtures of:

- Type system
- Instantiation aspects (constness, volatility, scope, etc.)
- Usage aspects (access rights, type interface, etc.)
- Implementation aspects (variable sizes, pointers/references, static/dynamic inference, etc.)
- Usability aspects (typing shortcuts, expressiveness, perceived clarity, etc. )

Type == Class

- Suggests that *Set theory* == *Relational model / Relational theory*

# AN XT-UML CENTRIC TYPE SYSTEM



# HOW?

*Databases, Types, and The Relational Model: The Third Manifesto*

by

H. Darwen and C.J. Date

# D&D TYPE SYSTEM

## Some key points:

- Types can be arbitrarily complex.
- Constraints are a part of the type.
- Types are declarative.
- Types have various representations.
- There are platform independent representations and platform specific representations.

## In addition:

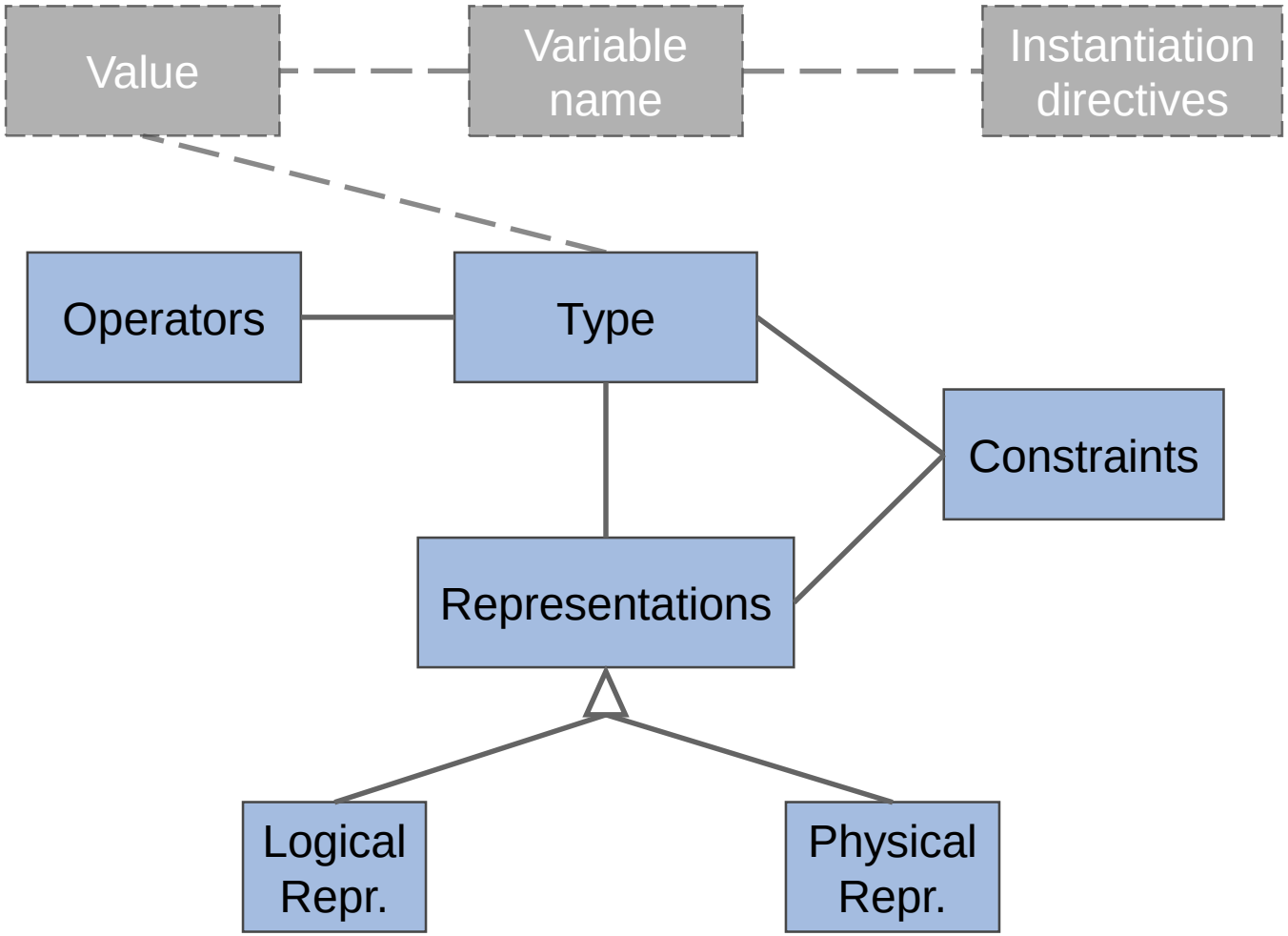
- A type definition is done from a mathematical perspective & platform independent perspective.

# D&D TYPE SYSTEM

## Type examples used in this presentation:

- **Real** : A rational number
- **Integer** : A whole number
- **LengthType** : A physical length
- **AngleType** : An angle
- **PointType** : A point in a 2D space
- **EllipseType** : A ellipse

# D&D TYPE SYSTEM



# D&D TYPE SYSTEM

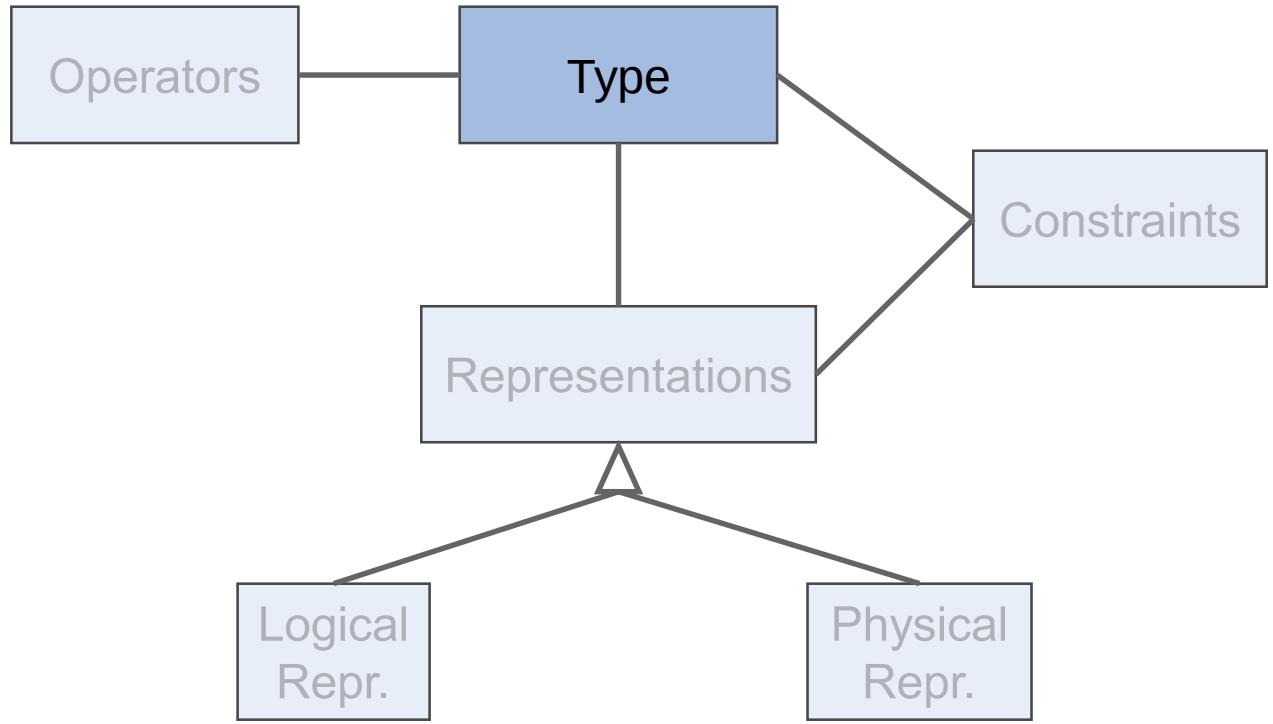
**Type**
A named ***set of values*** of arbitrary complexity.

New types are defined by subtyping and/or composition.

**E.g.**
Real

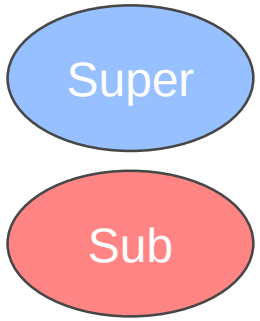
LenghType : Real

EllipseType
{
MinorAxis : LengthType,
MajorAxis : LenghType
}



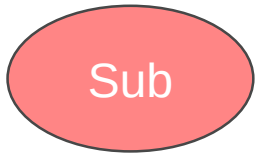
# D&D TYPE SYSTEM

## Inheritance vs Subtyping

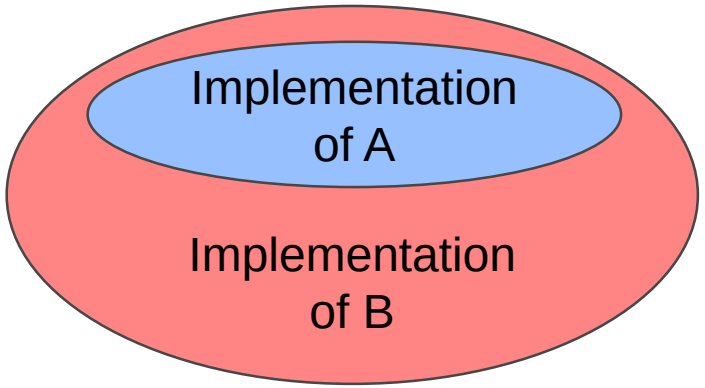


# D&D TYPE SYSTEM

## Inheritance vs Subtyping

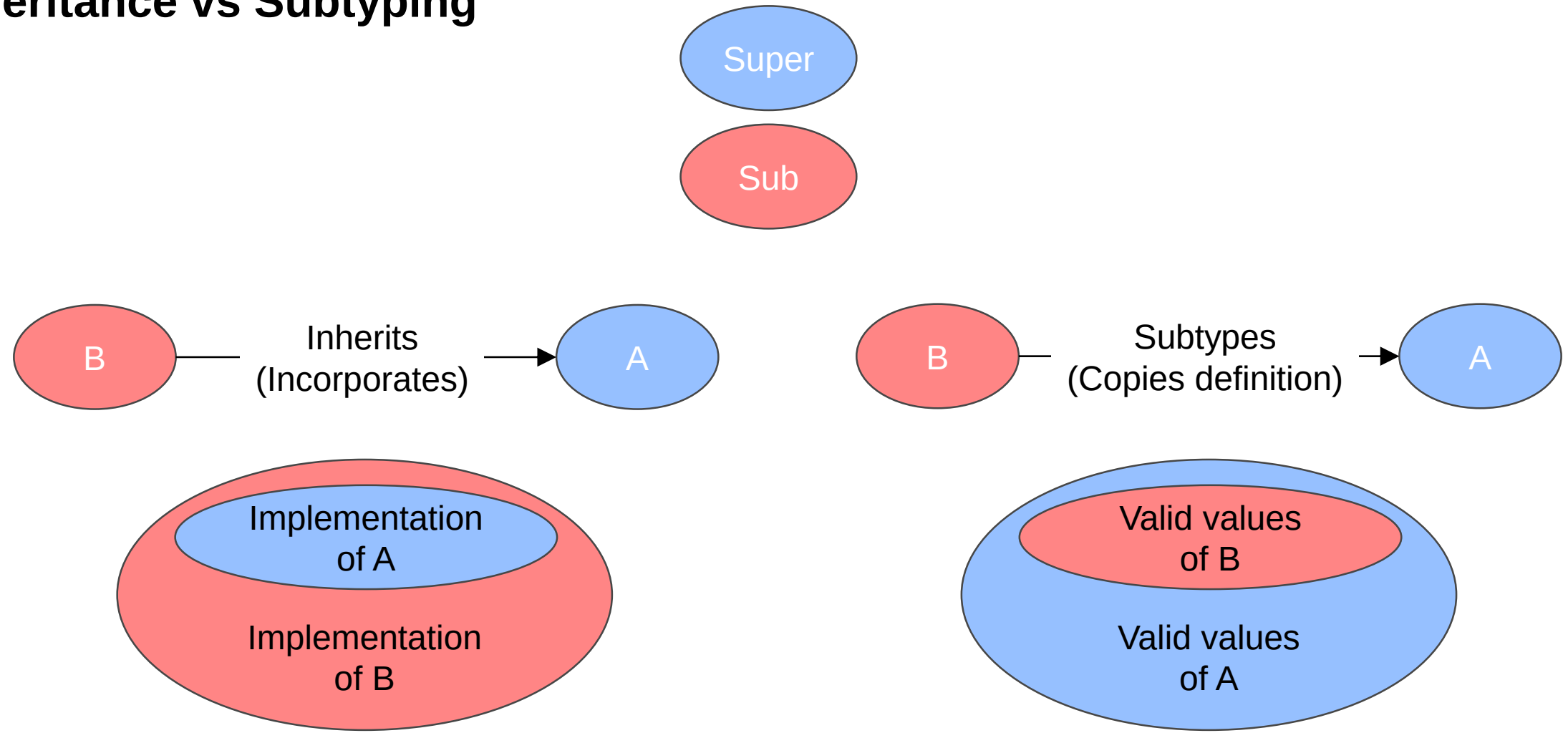







# D&D TYPE SYSTEM

## Inheritance vs Subtyping



# D&D TYPE SYSTEM

## Operators

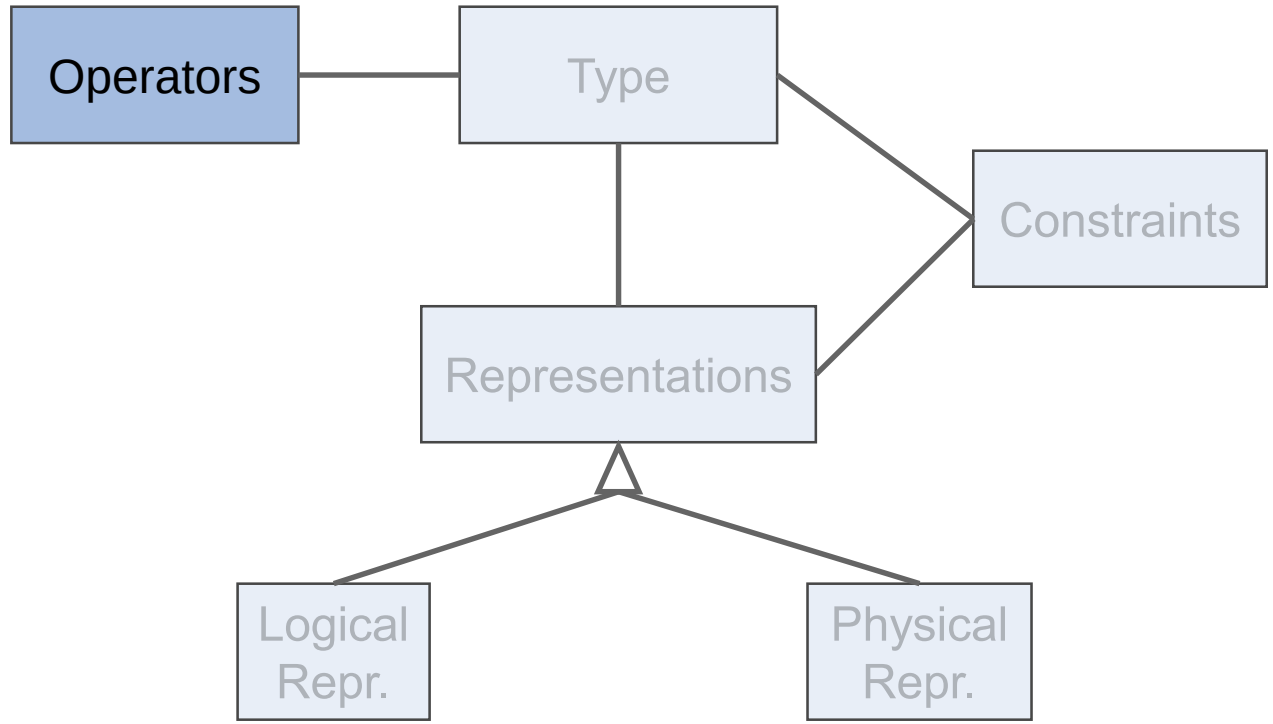
Define valid operators and their arguments for a type.

**E.g.**

Real : + - / * *etc.*

LenghType : + - / * *etc.*

EllipseType : + - / * *etc.*



# D&D TYPE SYSTEM

## Constraints

Restricts valid set of values for a ***type*** or ***representation***.

Constraints are declarative.

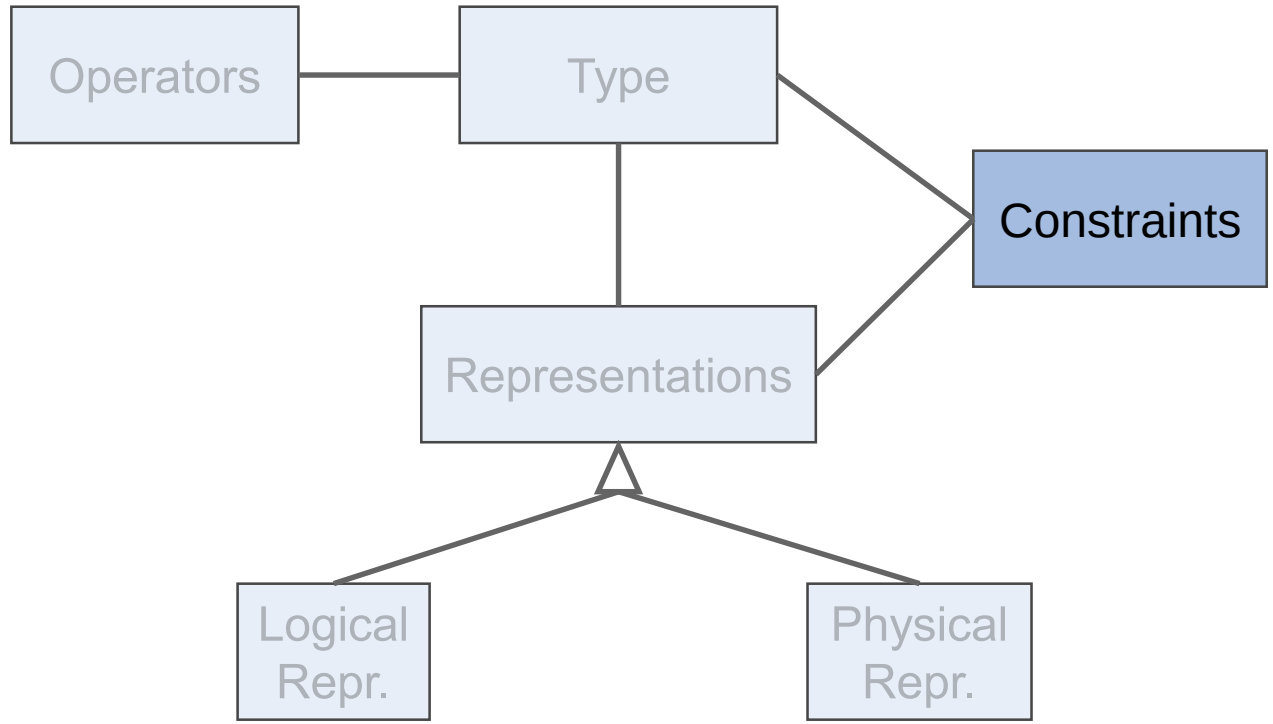
**E.g.**

Real : [-∞, +∞] *(infinite resolution)*

Integer : [-∞, +∞] *(no fractions)*

LenghType : [0.0, +∞]

EllipseType : MinorAxis <= MajorAxis

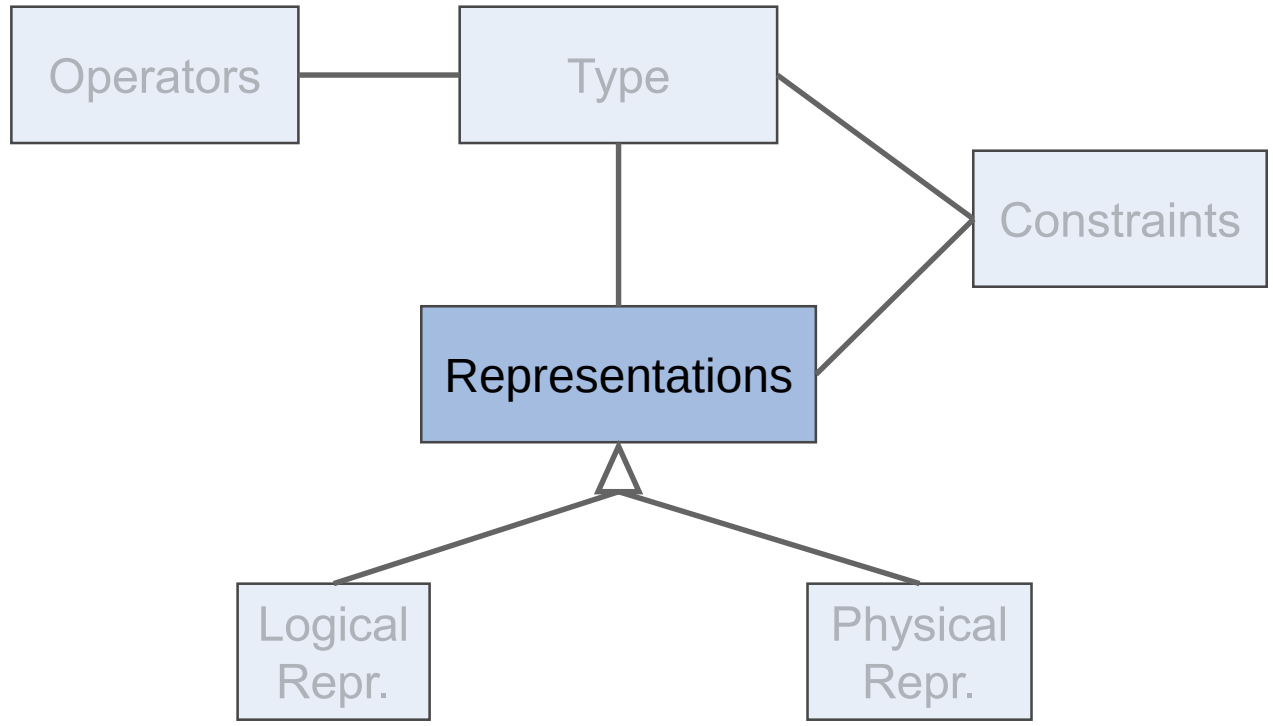


# D&D TYPE SYSTEM

**Representation**

Defines various views of a type.

Representations are declarative.



# D&D TYPE SYSTEM

## Logical representation

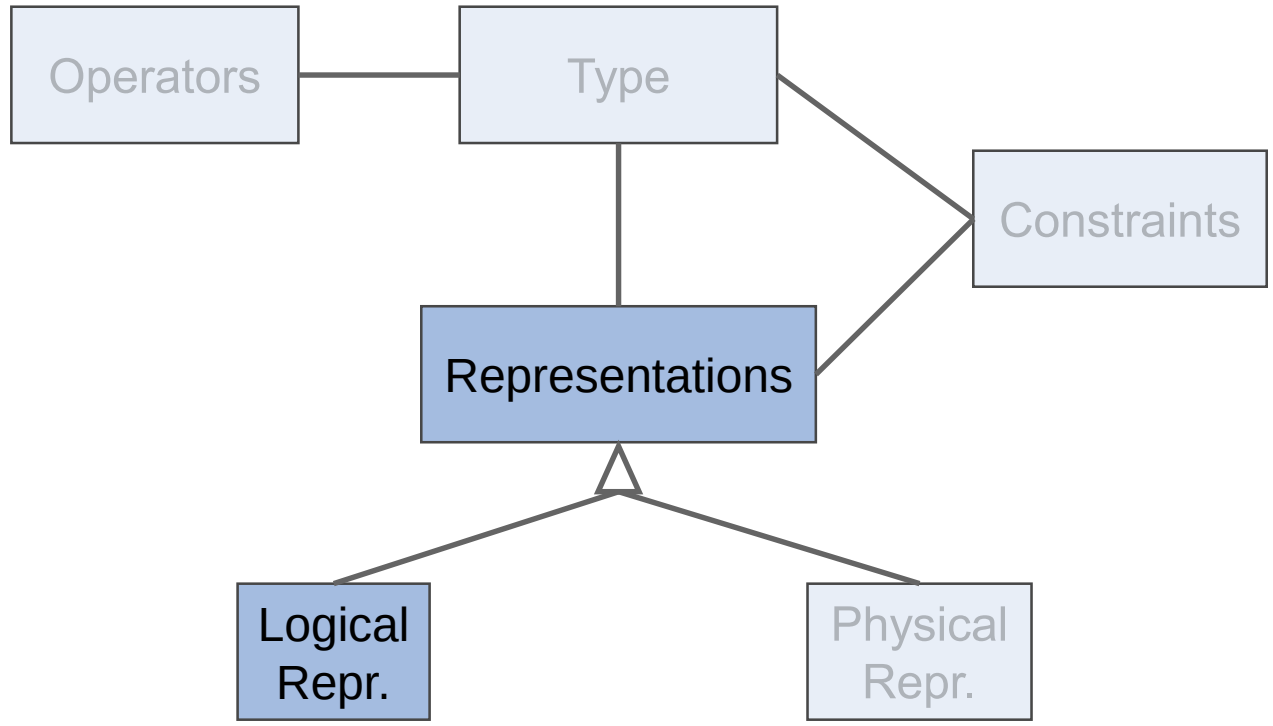
The user's views of a type.

A type must have **at least** 1 logical representation.

**E.g.**
Real : 10-base

LengthType : Meter, Foot, …

EllipseType {MinorAxis, MajorAxis}



# D&D TYPE SYSTEM

## Logical representation

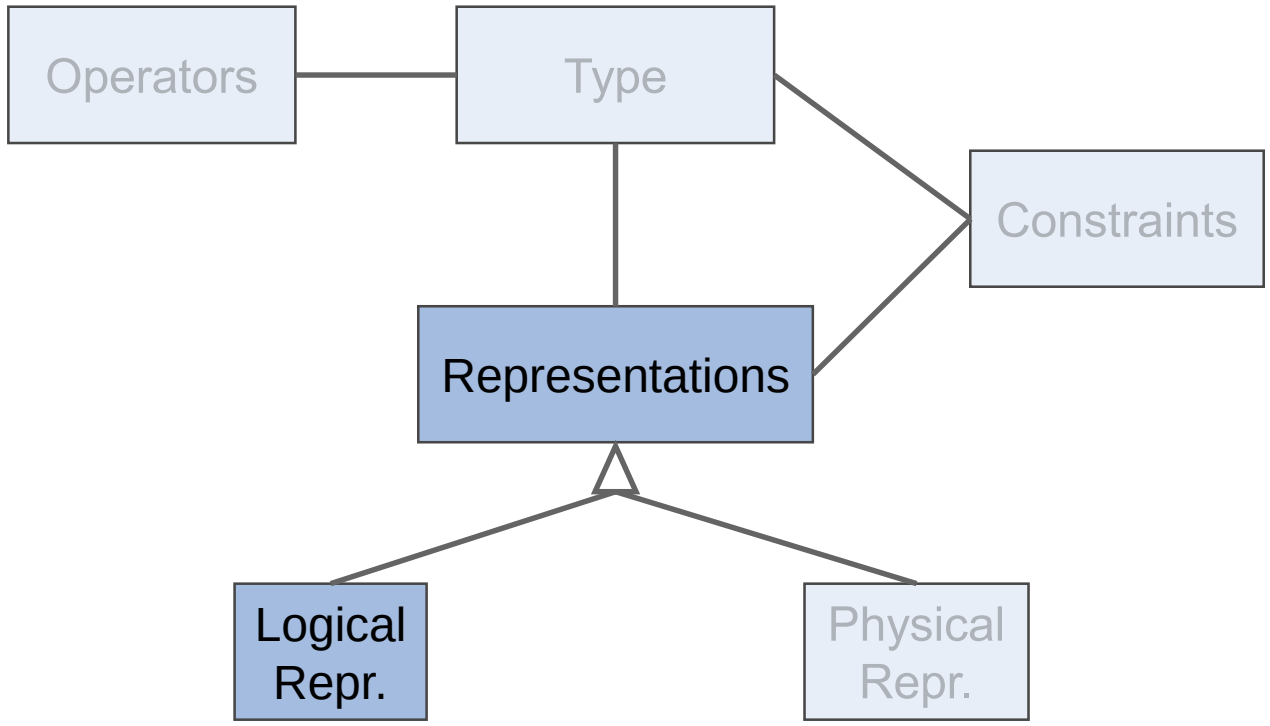
Every logical representation can have its own set of constraints.

**E.g.**

AngleType : Real

Rad : [0, 2π]

Deg : [0, 360]



# D&D TYPE SYSTEM

## Logical representation

All logical representations shall be able to ***completely represent*** all values of a specific type.

**E.g.**

AngleType

Rad : [0, 2π]

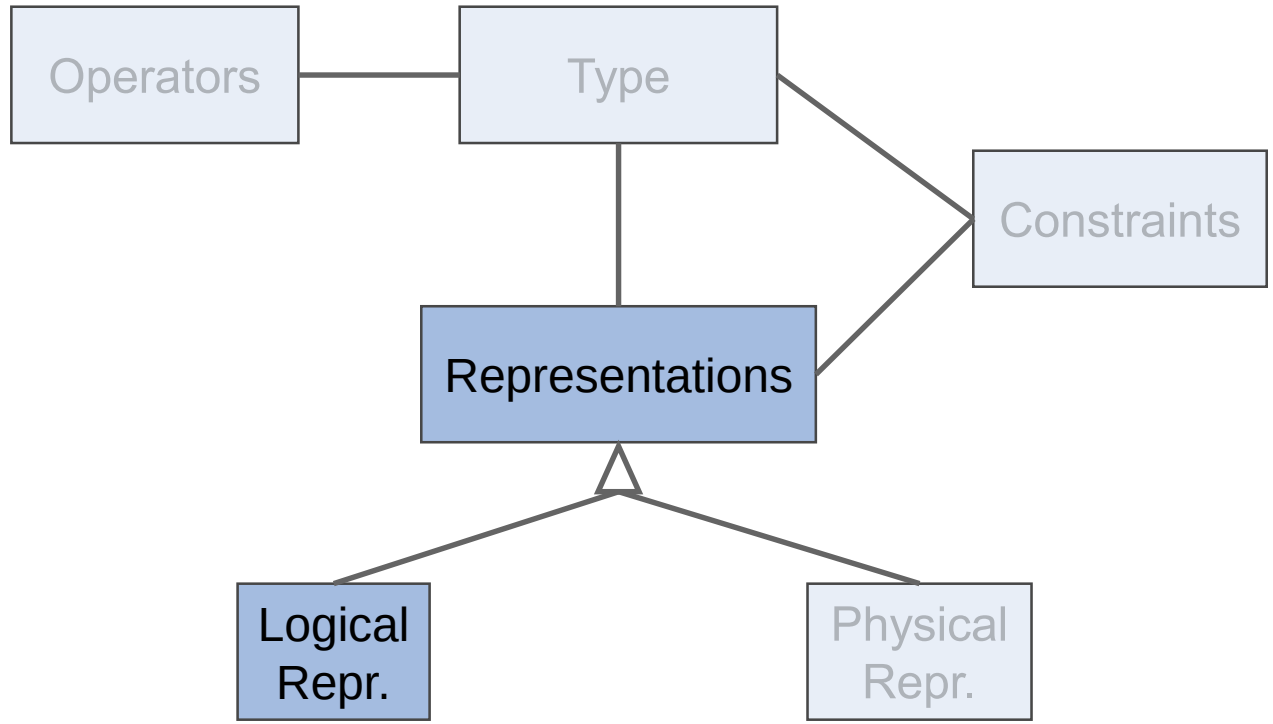
Deg : [0, 360]

*Valid:*

[0, 2π] ≡ [0, 360]

*Invalid:*

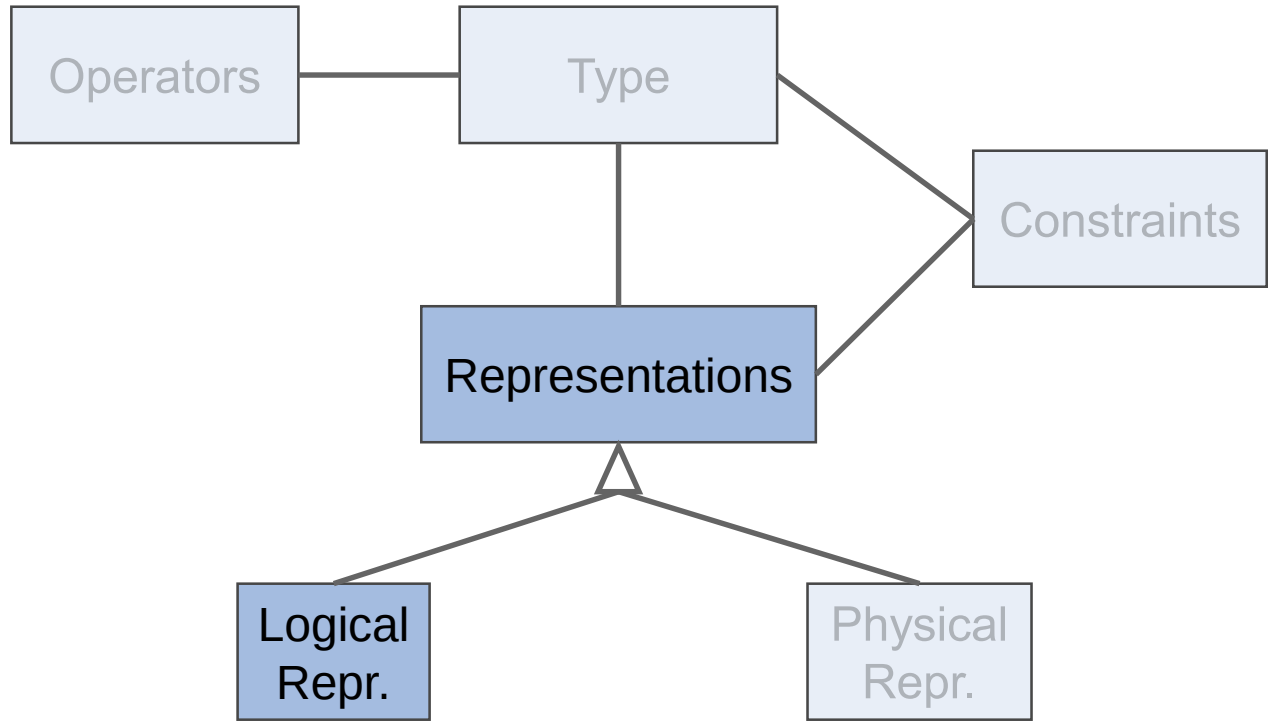
[0, 2π] ≢ [0, 360[



# D&D TYPE SYSTEM

## Logical representation

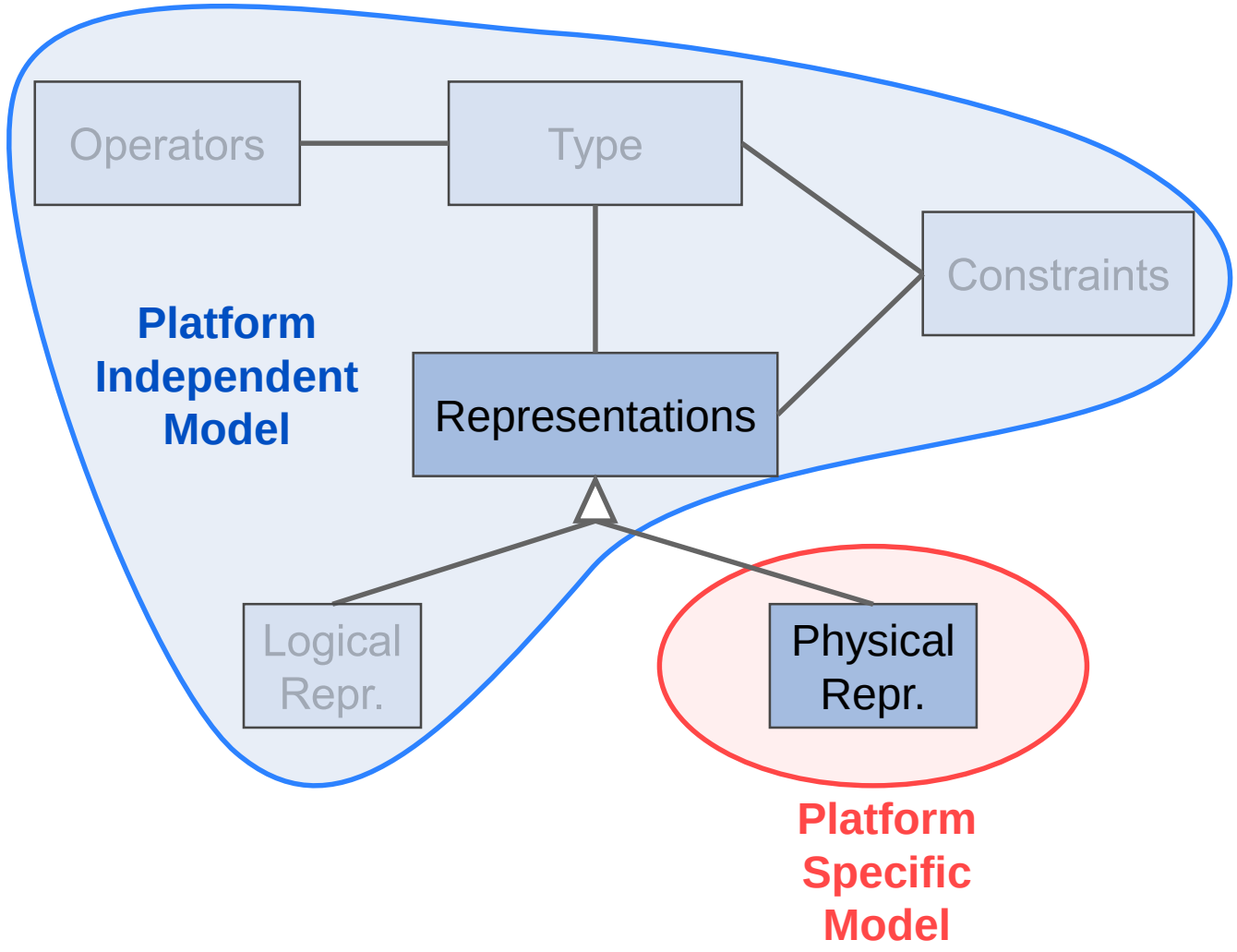
Which representation a type is using internally is only of interest to the type implementer.



# D&D TYPE SYSTEM

## Physical representation

The platform specific realization.



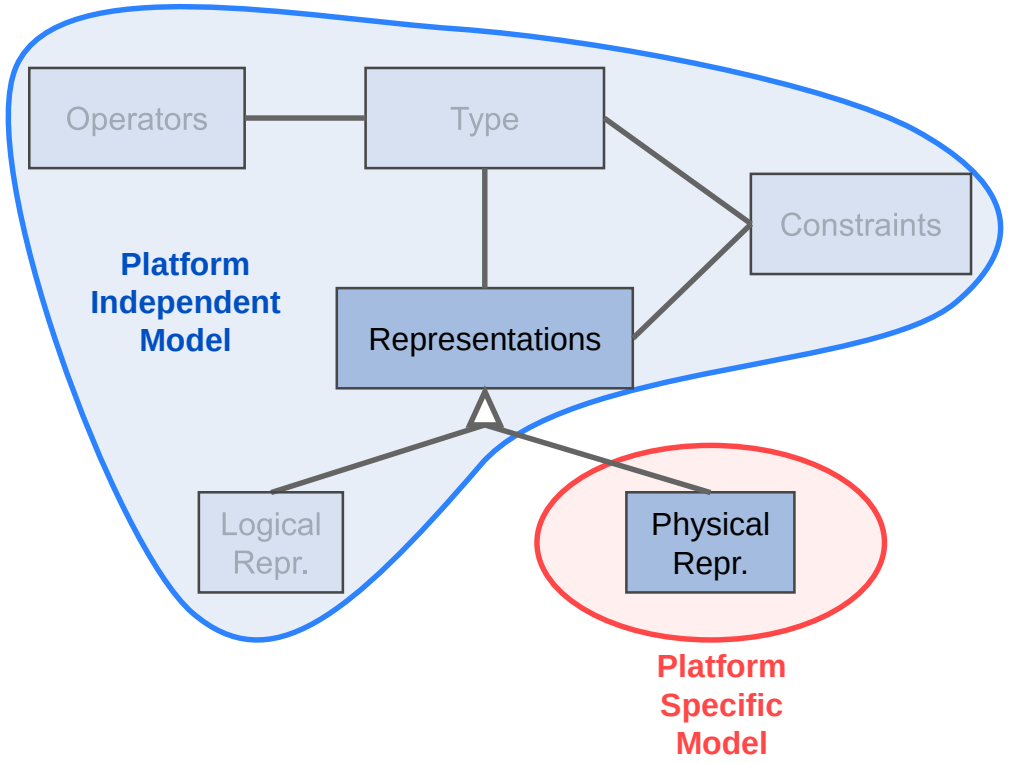
# D&D TYPE SYSTEM

## Physical representation

The platform specific realization.

Types that need PSM representations:

- Real
- Integer
- Text – Possibly endless sequence of symbols.
- Boolean – True / False



# D&D TYPE SYSTEM

## Physical representation

The platform specific realization.

**E.g.**

Real : IEEE 754 - Binary64

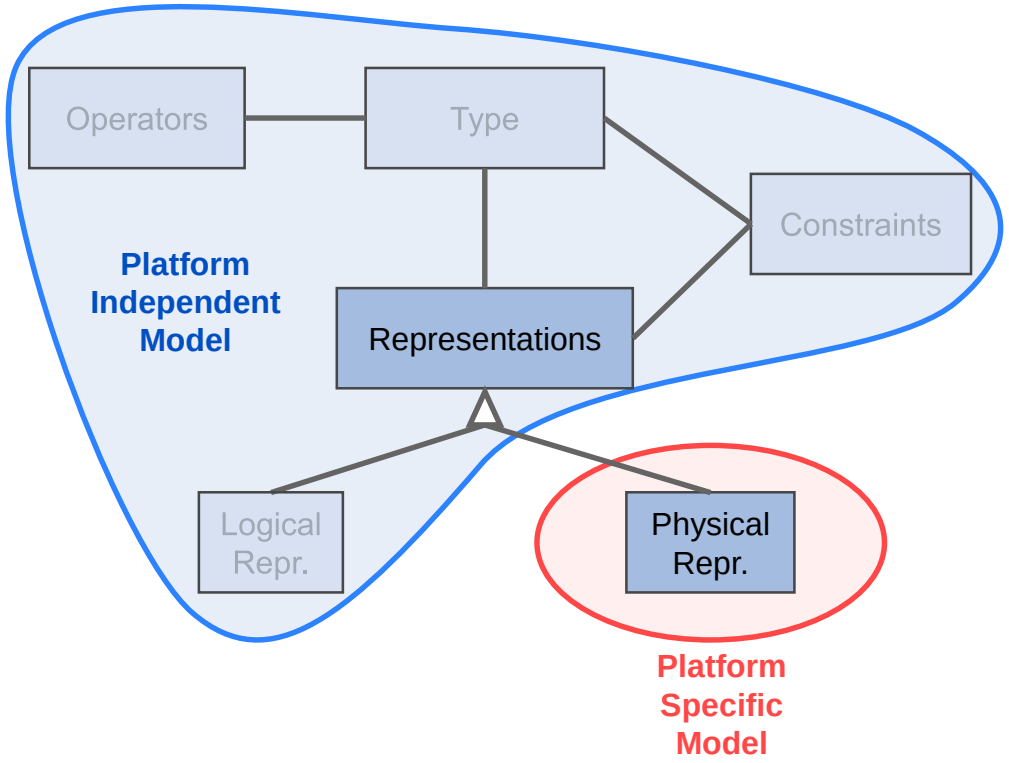
Integer : 16 bits

Text : char[]

Boolean : 1 bit

LengthType : *Given by Real*

EllipseType : *Given by Real*



# D&D TYPE SYSTEM

## Pseudo code examples

```
/* A generic length in unit Meter. */
TYPE LengthType
  REPRESENTATION Meter
    {
    Length Real;
    CONSTRAINT Length >= 0.0;
    };


/* A generic angle in unit radian. */
TYPE AngleType
   REPRESENTATION Radian
    {
    Angle Real;
    CONSTRAINT Angle >= 0.0 AND Angle <= (2 * π);
    };
```

# D&D TYPE SYSTEM

## Pseudo code examples

```
/* A geometric point in 2D space with 2 representations. */
TYPE PointType

   REPRESENTATION Cartesian
    {
    XCoord Real;
    YCoord Real;
    }

   REPRESENTATION Polar
    {
    Length LengthType;
    Angle AngleType;
    };
```

```
/* PointType instantiations */
PointType A.Cartesian(10.0, 10.0);
PointType B.Polar (20.0, π/4);
PointType C;

/* PointType operations */
C = A + B;

/* Print result */
Print C.XCoord;
Print C.Angle;
```

# D&D TYPE SYSTEM

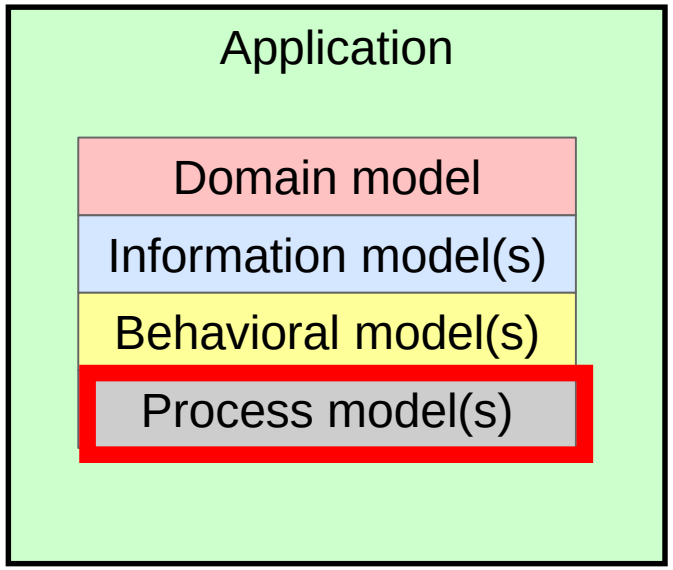
## Drawbacks, potential problems?

Types can be arbitrarily complex.

- Type <> Class
- Containers => Difficult to separate types from classes & relations.

Type operators

- Can be difficult to separate from processes.



# Questions?