Category Theory

without math

well, maybe just a wee bit

Lecture #2

Don Batory
University of Texas at Austin
batory@cs.utexas.edu

2023

Lecture #2: Two MDE Applications using Functors

- App #1: verifying class & object diagram refactorings

  equiv: schema and database refactorings

  - Will use core 40-year-old relational database ideas & terminology

- App #2: Relationship of ***CT*** + relational algebra = foundation for OCL neat result

- Purpose of this lecture: show non-trivial examples of how ***CT*** organizes difficult problems and "how to think" in a structured way

Very Different than Lecture #1

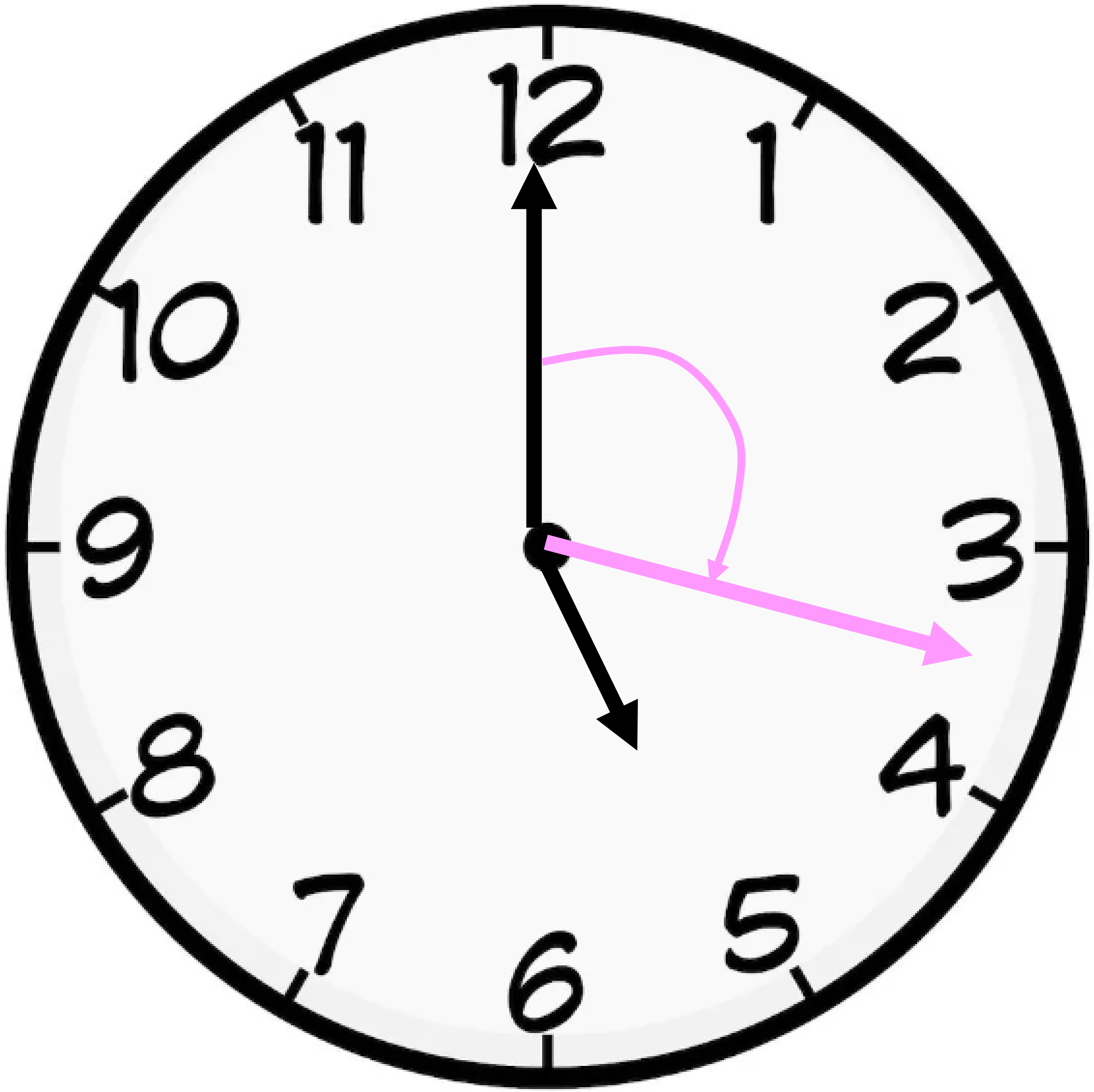
- Tour of an old, but still open, research area open for at least 2+ decades
  - Refactoring class diagrams + OCL constraints + object diagrams

- Refactoring is central to modern OO software development
  - refactoring should have a central role in MDE meta-model development too

- And it is surprisingly hard…
  - give you an insight why
  - explain most recent progress w. ***CT*** viewpoint
  - show what it would take to solve
  - open problems suitable for PhDs

Sit back, relax, to learn some neat ideas!

Tutorial: Lecture #2a



Ends at 17:00

Most controversial part is Lecture #2b Ends at 17:17

Functors and Embeddings

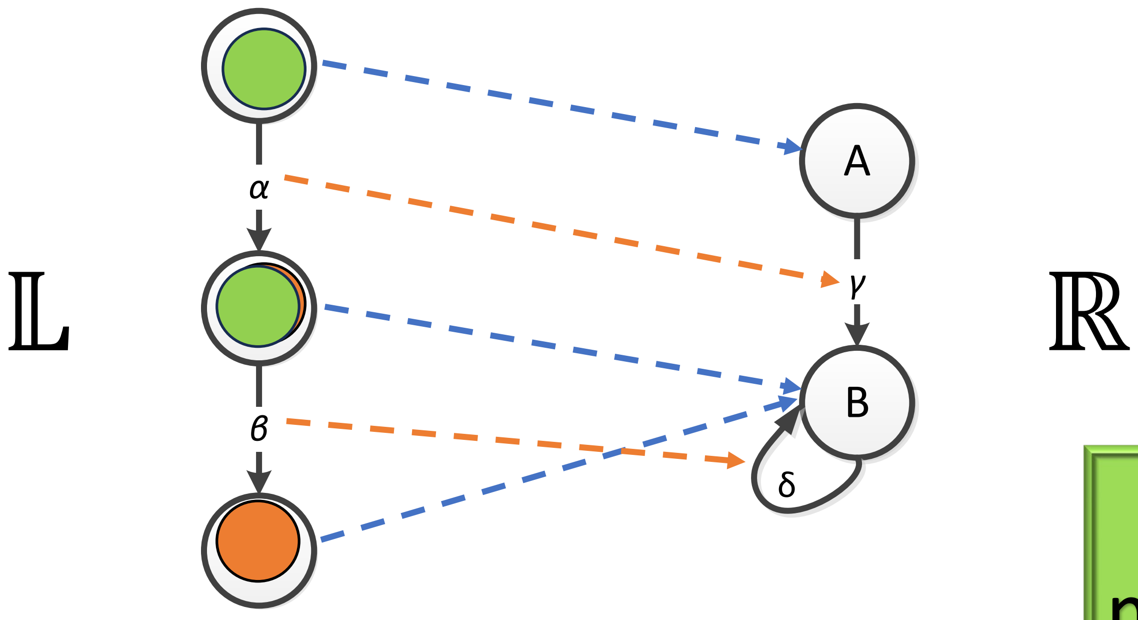
definition and examples

Functor Definition

Functor $\mathbb{F}: \mathbb{L} \rightarrow \mathbb{R}$ is an embedding of category $\mathbb{L}$ in category $\mathbb{R}$

1. Arrow from each domain D in $\mathbb{L}$ to a domain $\mathbb{F}(\mathrm{D})$ in $\mathbb{R}$
2. Arrow from each arrow $\theta: \mathrm{D}_1 \rightarrow \mathrm{D}_2$ in $\mathbb{L}$ to $\mathbb{F}(\theta): \mathbb{F}(\mathrm{D}_1) \rightarrow \mathbb{F}(\mathrm{D}_2)$ in $\mathbb{R}$

"Embedding" = every pt and arrow and every inferable arrow of $\mathbb{L}$ is accounted for in $\mathbb{R}$



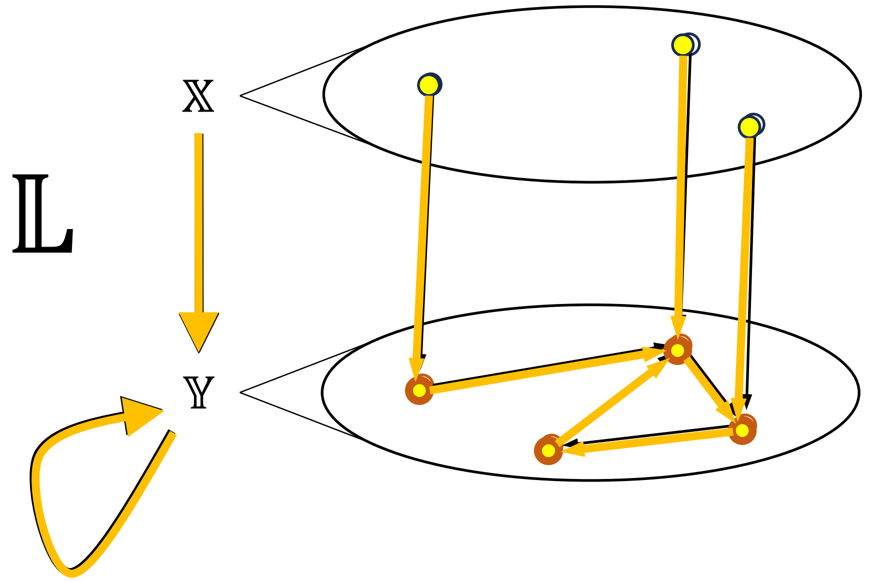
$\mathbb{L} \hookrightarrow \mathbb{R}$

Simple but much is unsaid

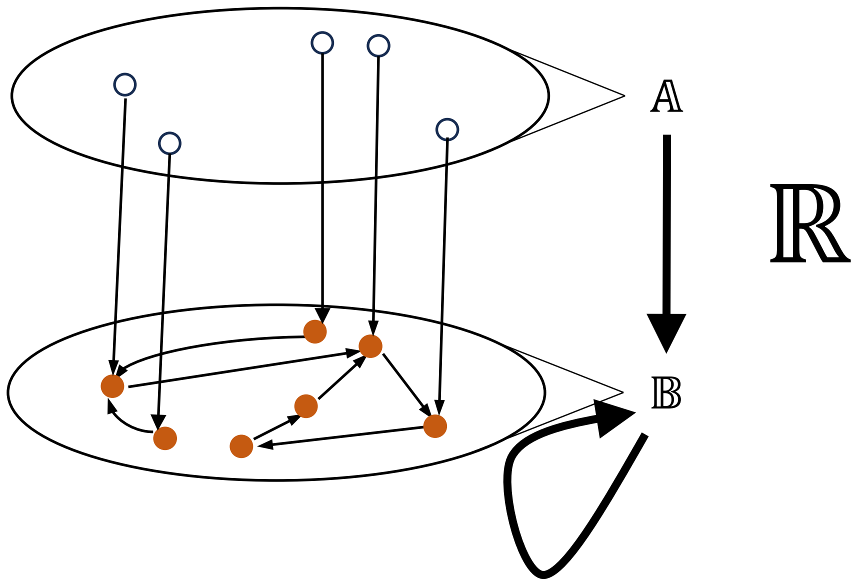
Functor $\mathbb{F}$ embeds $\mathbb{L}$ into $\mathbb{R}$

definition meaning

$$\mathbb{L} \hookrightarrow \mathbb{R}$$



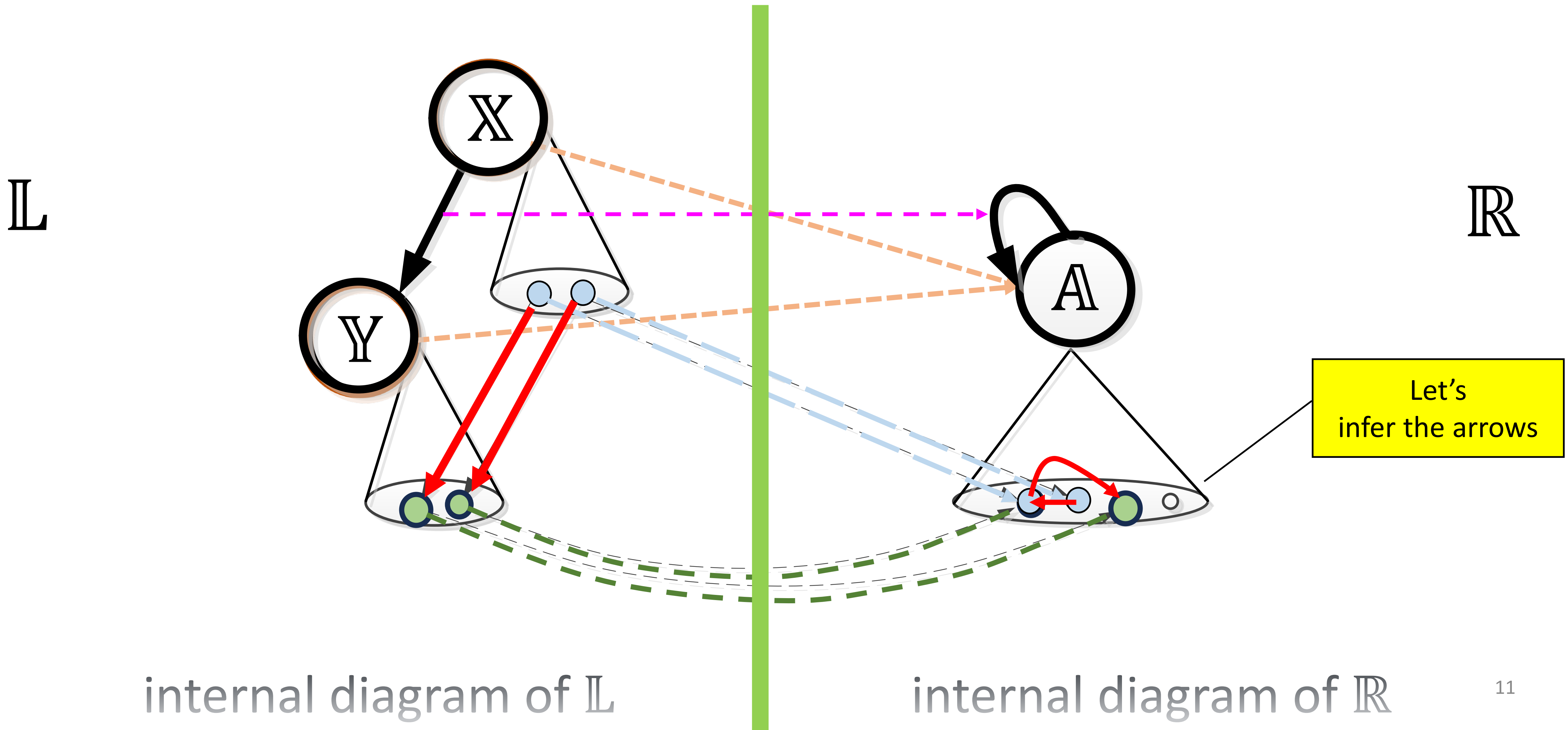
internal diagram of $\mathbb{L}$



internal diagram of $\mathbb{R}$

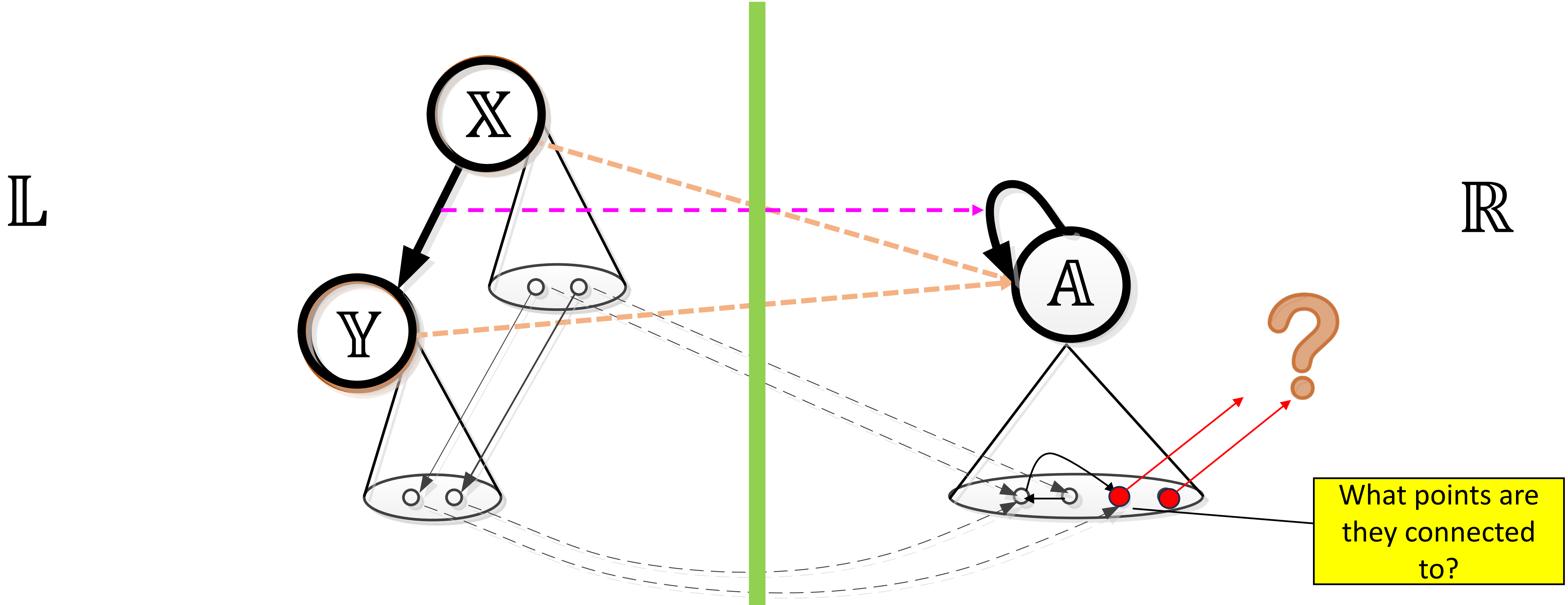
$\mathbb{F}$ preserves the ext/int multi-graph structure of $\mathbb{L}$ in $\mathbb{R}$

Simple Example: Functor $\mathbb{F}: \mathbb{L} \to \mathbb{R}$



internal diagram of $\mathbb{L}$ internal diagram of $\mathbb{R}$

Simple Example: Functor $\mathbb{F}: \mathbb{L} \to \mathbb{R}$



Ans: for this example, it doesn't matter -- pick any point in $\mathbb{A}$

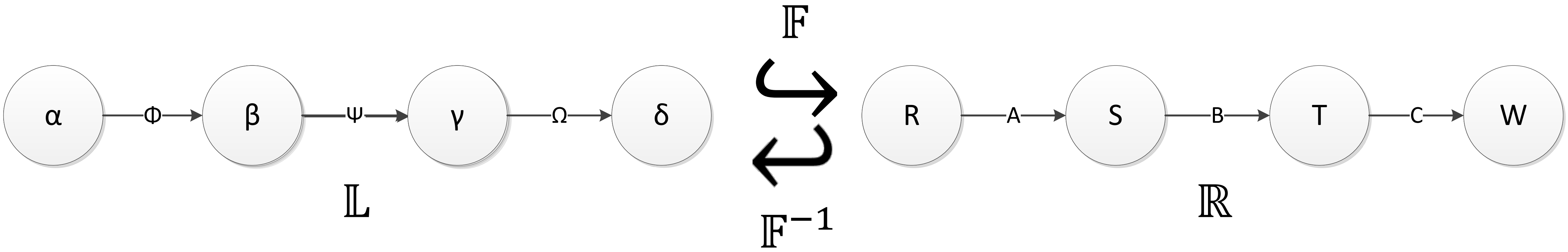
internal diagram of $\mathbb{L}$ internal diagram of $\mathbb{R}$

That’s a lot!

Fortunately, all examples of functors familiar to me in SE are special cases
that are easy to understand

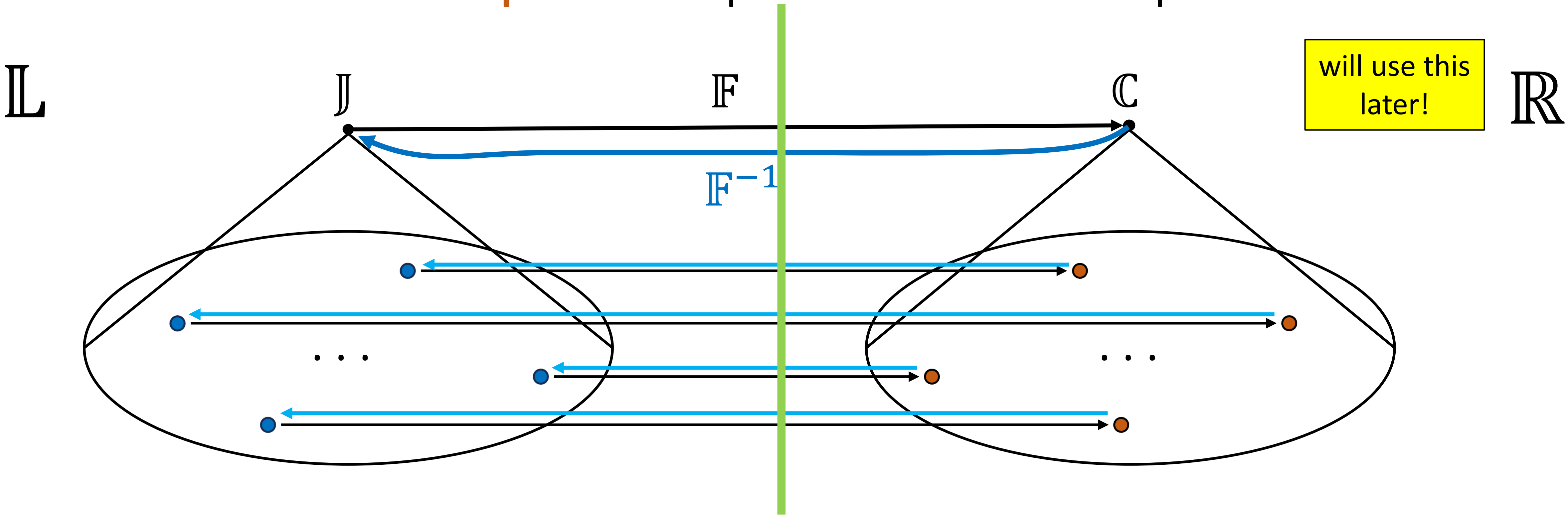
Special Case #1: Category Equivalence

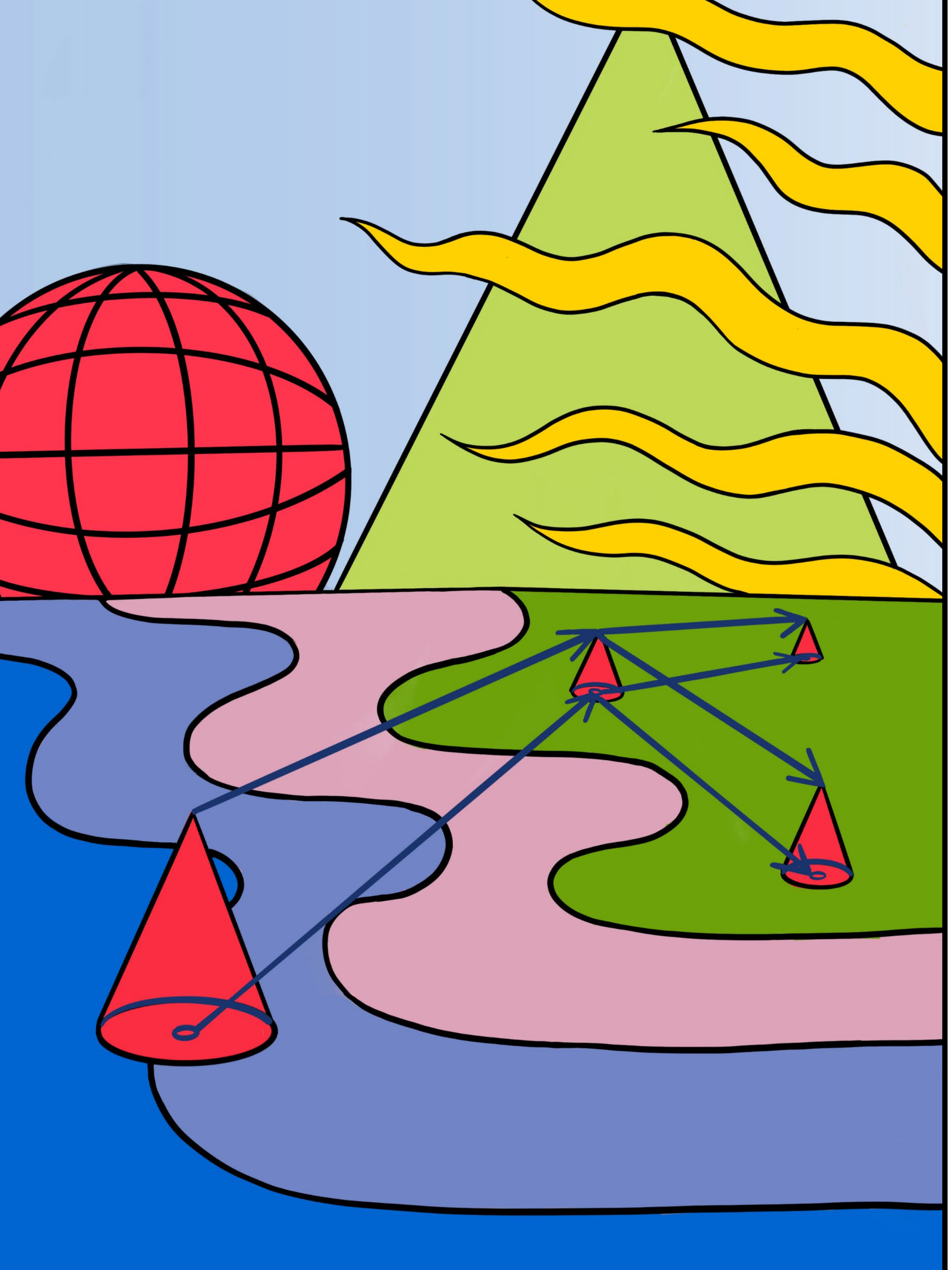
- Categories with the same "shape" are **isomorphic**: $\mathbb{L} \hookrightarrow \mathbb{R}$ and $\mathbb{R} \hookrightarrow \mathbb{L}$
  - only difference between $\mathbb{L}$ and $\mathbb{R}$ are the names of points, domains, arrows
  - functors of isomorphic categories have inverses



Special Case #2: Domain Equivalence

- Two categories each with a single domain $\mathbb{J}$ and $\mathbb{C}$; functor $\mathbb{F}: \mathbb{J} \rightarrow \mathbb{C}$ relates them
- Two domains are **isomorphic** if their points are in 1:1 correspondence





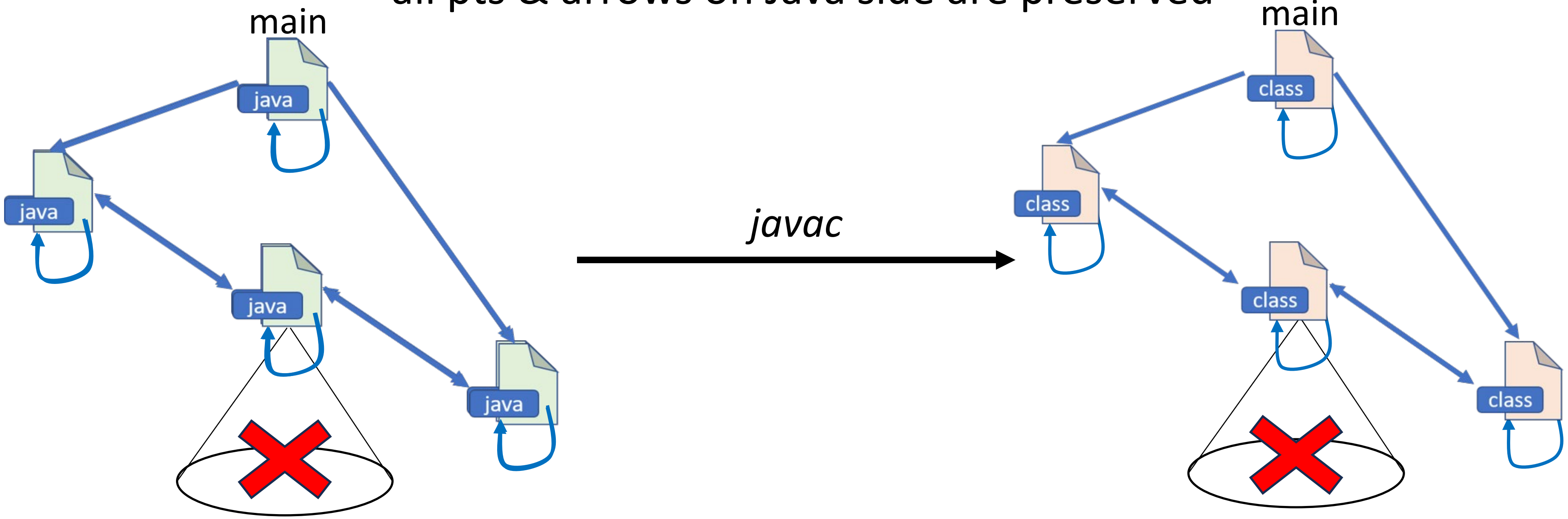
A Common Example of a Functor

and correct ***CT*** terminology

Revisit the Java Compiler

- A **Java program** is a category of Java files rooted at Main.java
- ***javac*** translates a JavaProgram into an isomorphic category of dotClass files

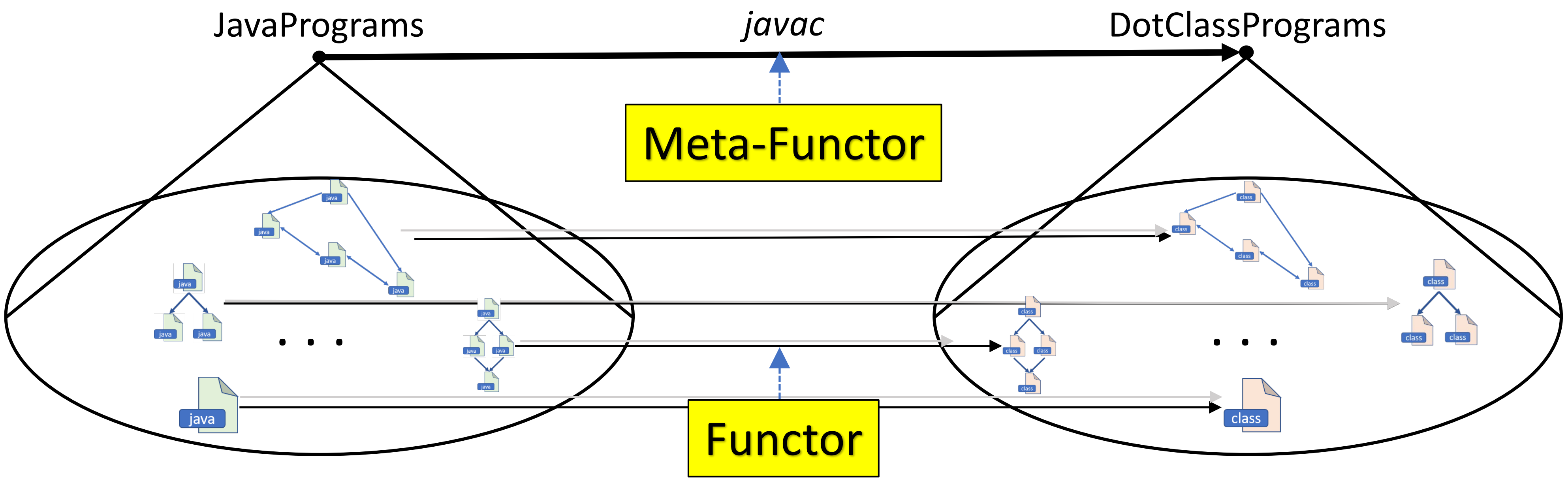
all pts & arrows on Java side are preserved



- Note: these categories are at the internal level only – they don't have internal diagrams!!

Meta-Functor

- ***javac***, a meta-functor, translates every Java program a category into a DotClassProgram an isomorphic category



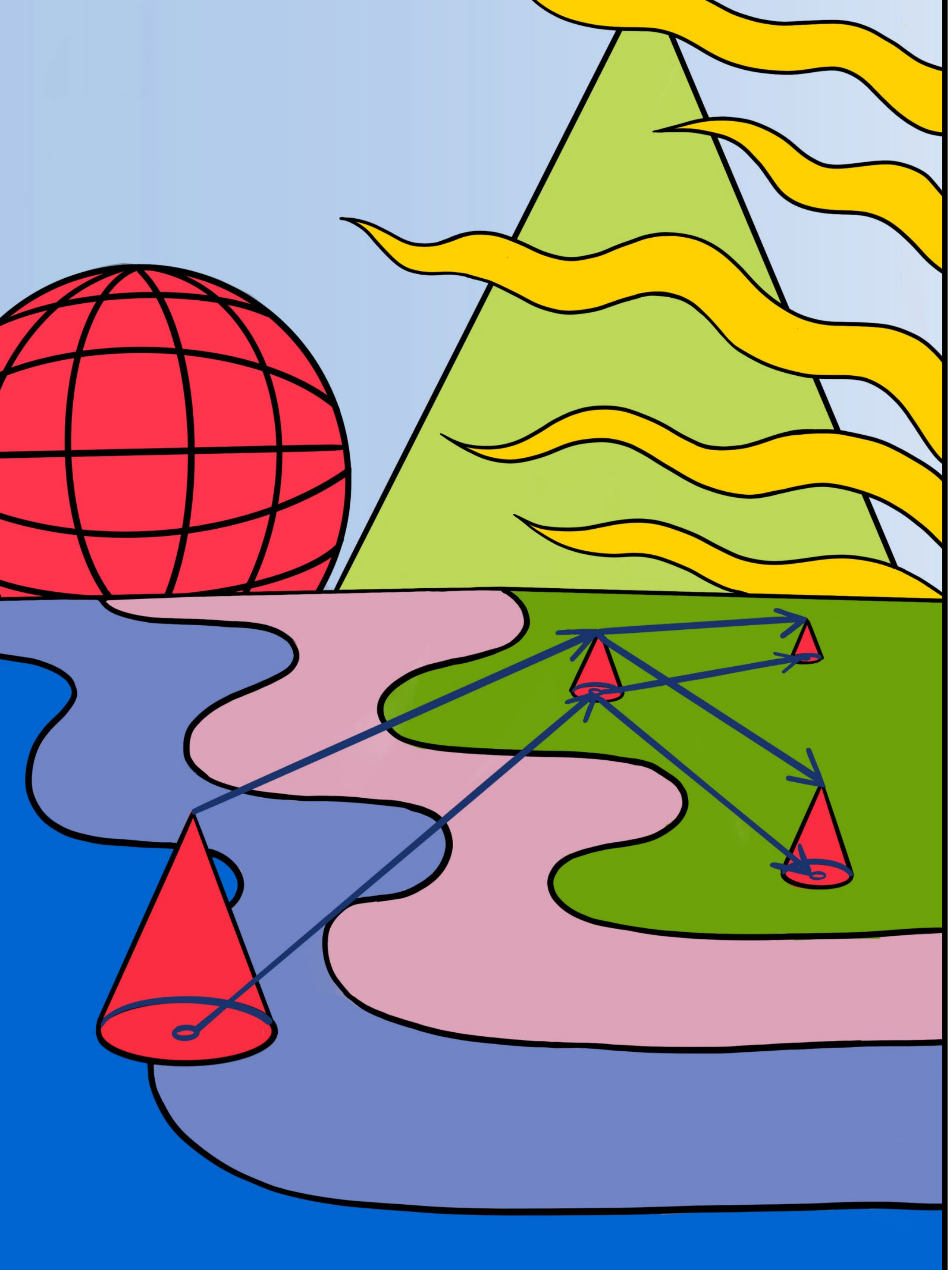
Correct ***CT*** Terminology for Purists

My Terminology

Correct ***CT*** Terminology

My Terminology		Correct CT Terminology
Arrow	⇨	Arrow
Meta-Arrow	⇨	Arrow
Functor	⇨	Functor
Meta-Functor	⇨	Functor
Point, Element	⇨	Object
Domain	⇨	Object

I will continue to use my terminology



Main Event

Verifying Class Diagram Refactorings

Ph.D. Topic

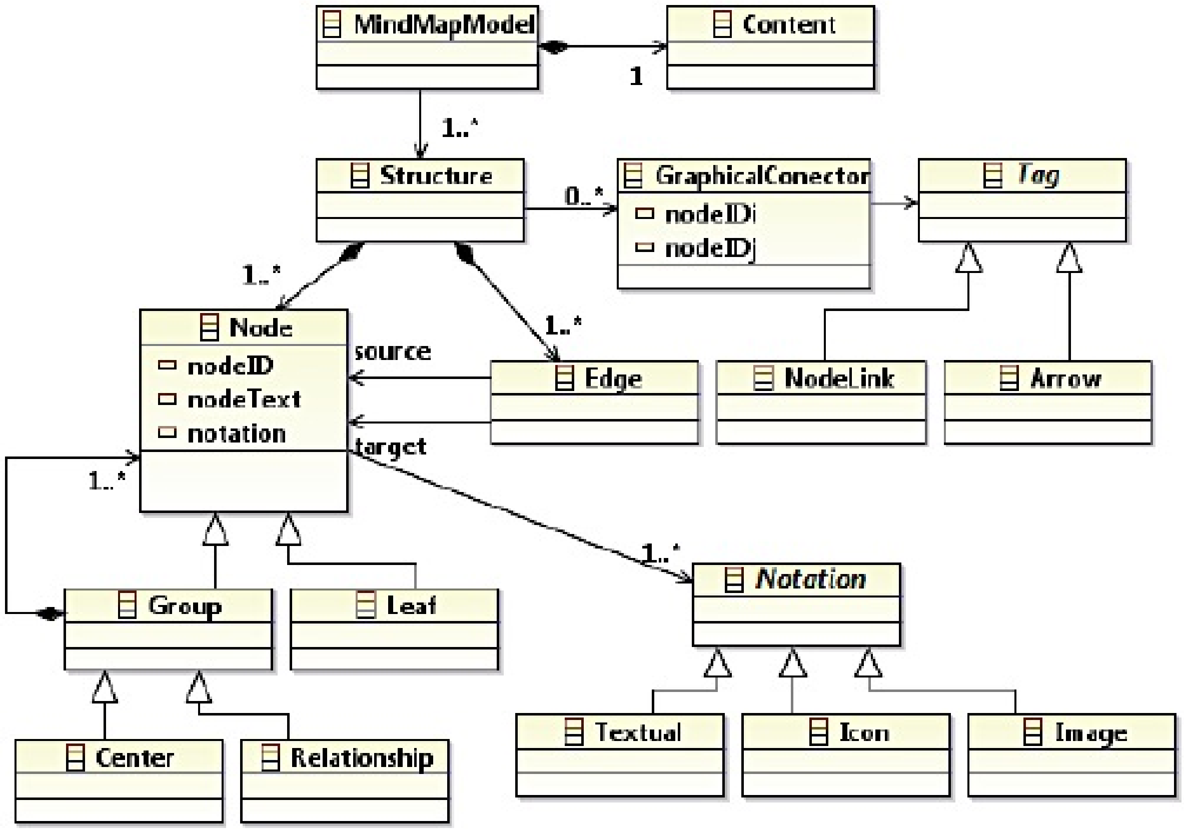
Metamodels in MDE

- 3 ways to declare metamodels in MDE:

	Metamodel Decl	Instance
This Lecture	Class diagram + OCL constraints	Object Diagram

Focus on Subset of UML Class Diagrams

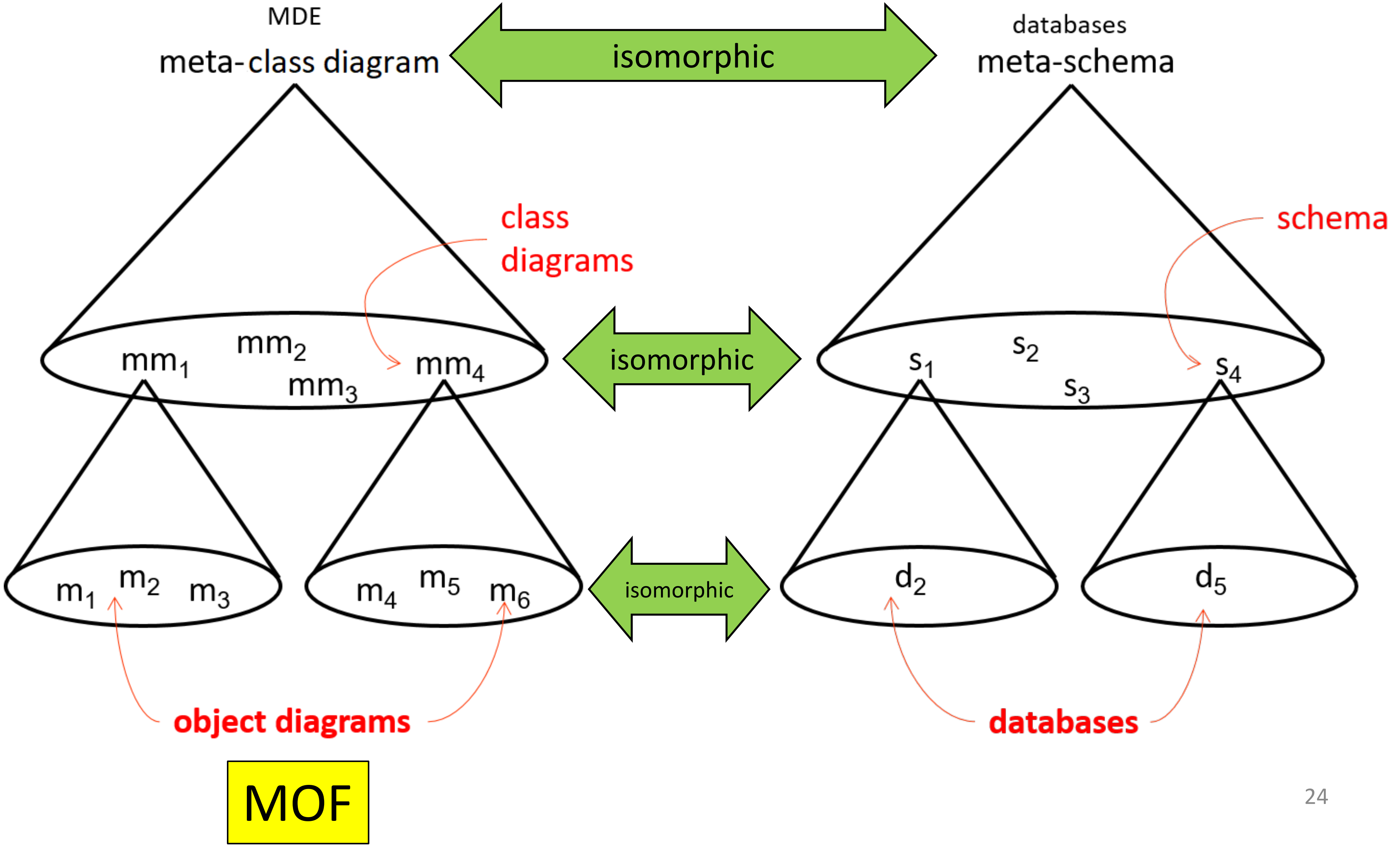
- No interfaces, no methods
- Only classes, data members, associations, and class inheritance
- Subset known for 25 years
  IBM's Rational Rose depicts relational database schemas with table inheritance
- Instances of schema are relational databases with table inheritance



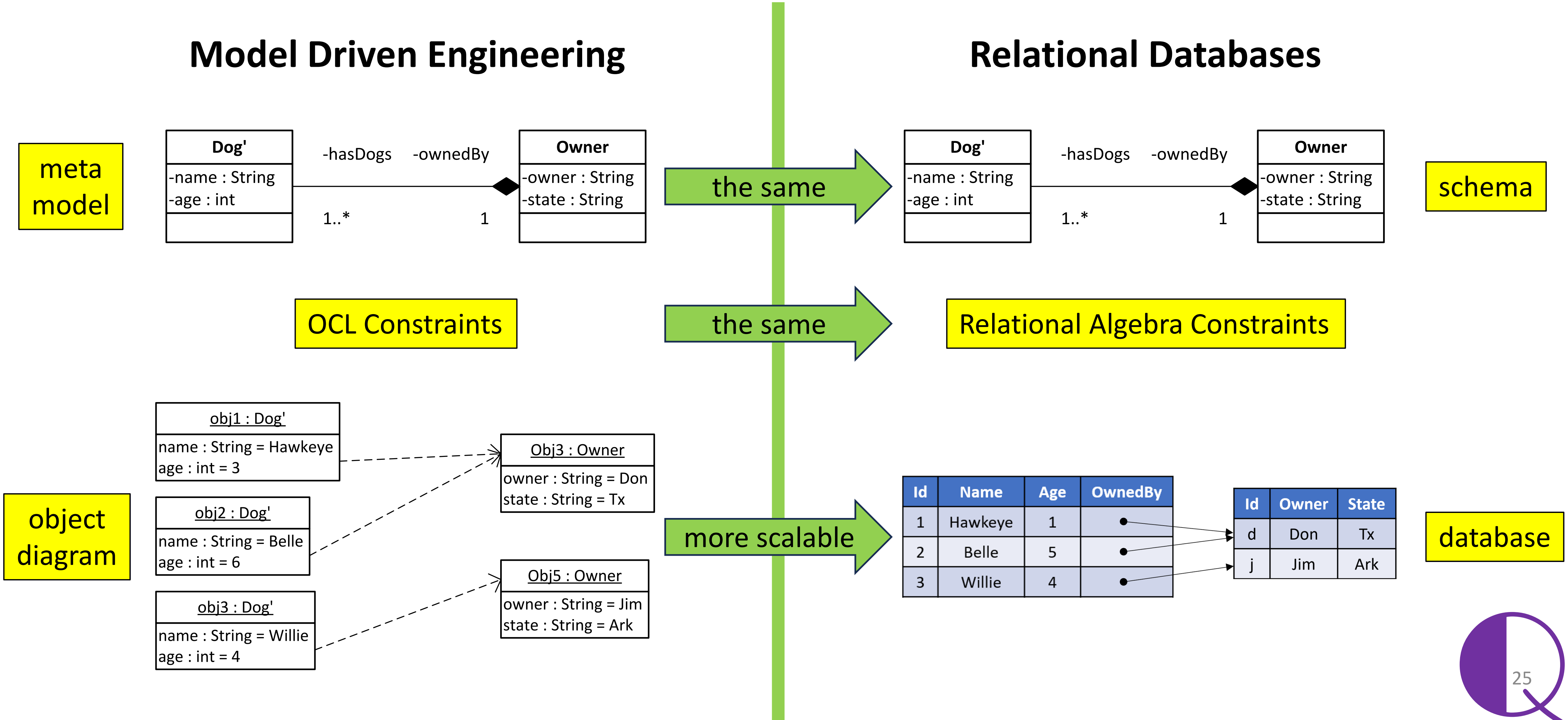
From Lecture #1

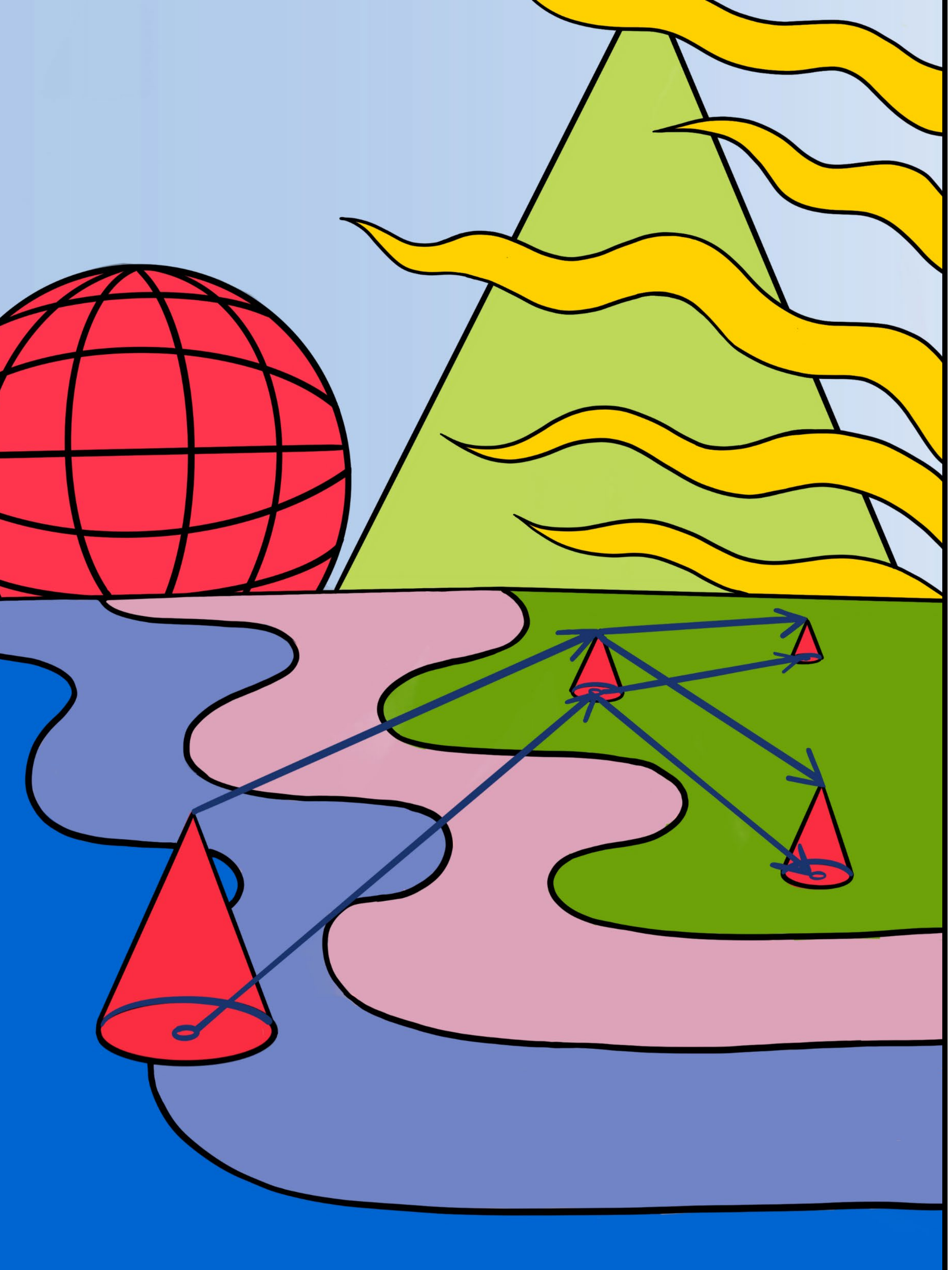
Abstractions are largely interchangeable

even though... MDE & DBMSs have very different implementations



Equalities or Isomorphisms





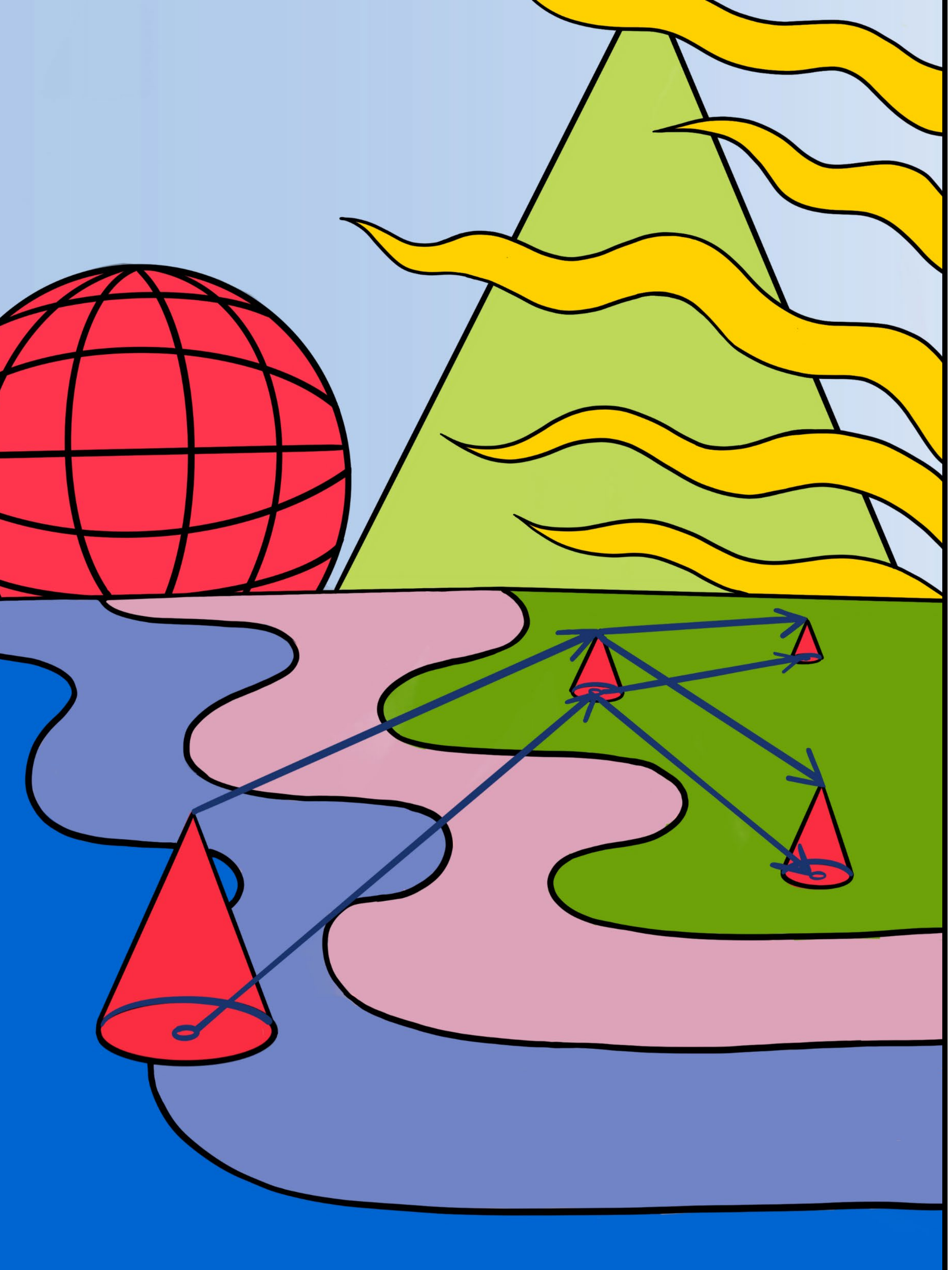
Enter the World of Refactorings

What are Refactorings?

- In physics, there are **coordinate transformations**
  - that reposition an object to get a particular view of it
  - **does not change object semantics**
- Watch this video of a 3D sculpture by James Hopkins. A rotation is a coordinate transformation

Refactorings are the software counterpart to coordinate transformations

preserve semantics
invertible
R is discrete, not continuous



The Normalize Refactoring

Running Example/Exemplar

Initial Model

Model $\mathfrak{m}_{\mathbb{L}}$

Dog

-name : String
-age : int
-owner : String
-state : String

Schema = class diagram

Database constraints called **functional dependencies**
Names of dogs is unique: name $\rightarrow$ age, owner, state
Names of owners are unique: owner $\rightarrow$ state

not OCL constraints (technically)

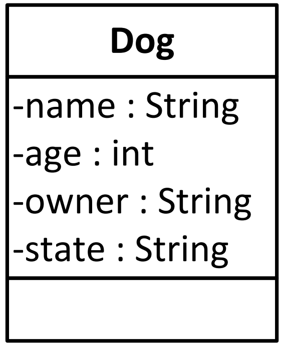
Id	Name	Age	Owner	State
1	Hawkeye	1	Don	Tx
2	Belle	5	Don	Tx
3	Willie	4	Jim	Ark
4	Lassie	7	Timmy	Cal
5	Pancake	6	Greg	Tx

database instance of schema

Problem with this design:
owner-state data is replicated

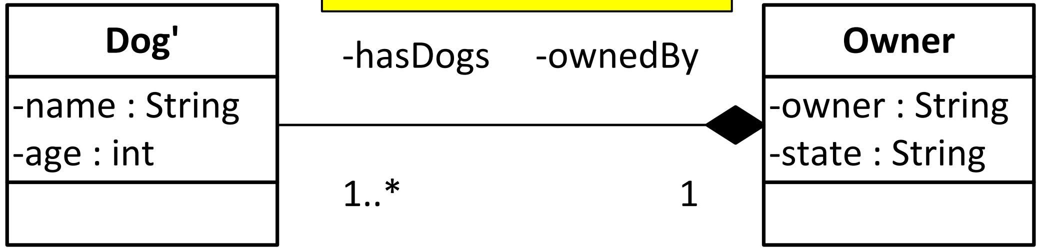
Normalize Schema and Database Refactoring

Model $\mathfrak{m}_{\mathbb{L}}$



Normalize schema

Model $\mathfrak{m}_{\mathbb{L}}$



Database constraints called **functional dependencies**
Names of dogs is unique: name → age, owner, state
Names of owners are unique: owner → state

same

Functional dependencies stay the same
Names of dogs is unique: name → age, owner, state
Names of owners are unique: owner → state

Id	Name	Age	Owner	State
1	Hawkeye	1	Don	Tx
2	Belle	5	Don	Tx
3	Willie	4	Jim	Ark
4	Lassie	7	Timmy	Cal
5	Pancake	6	Greg	Tx

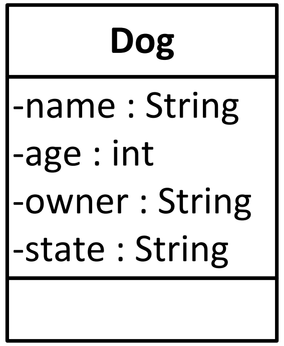
Normalize database

Id	Name	Age	OwnedBy
1	Hawkeye	1	•
2	Belle	5	•
3	Willie	4	•
4	Lassie	7	•
5	Pancake	6	•

Id	Owner	State
d	Don	Tx
j	Jim	Ark
t	Timmy	Cal
g	Greg	Tx

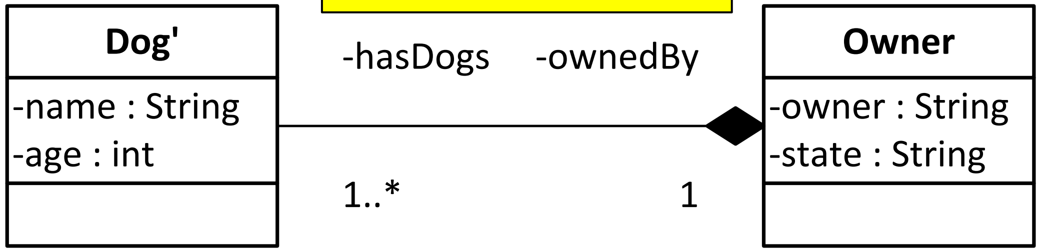
Unnormalize Schema and Database Refactoring

Model $\mathfrak{m}_{\mathbb{L}}$



Unnormalize schema

Model $\mathfrak{m}_{\mathbb{L}}$



Database constraints called **functional dependencies**
Names of dogs is unique: name → age, owner, state
Names of owners are unique: owner → state

same

Functional dependencies stay the same
Names of dogs is unique: name → age, owner, state
Names of owners are unique: owner → state

Id	Name	Age	Owner	State
1	Hawkeye	1	Don	Tx
2	Belle	5	Don	Tx
3	Willie	4	Jim	Ark
4	Lassie	7	Timmy	Cal
5	Pancake	6	Greg	Tx

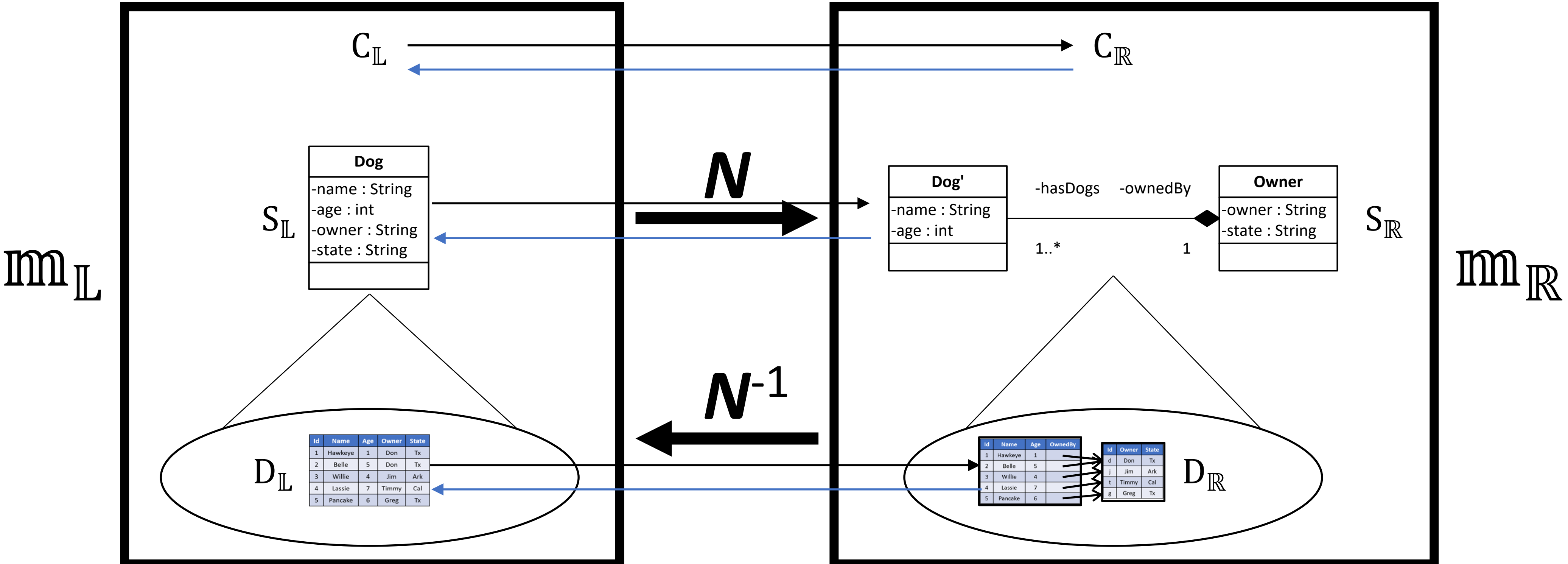
Unnormalize database

Id	Name	Age	OwnedBy
1	Hawkeye	1	
2	Belle	5	
3	Willie	4	
4	Lassie	7	
5	Pancake	6	

Id	Owner	State
d	Don	Tx
j	Jim	Ark
t	Timmy	Cal
g	Greg	Tx

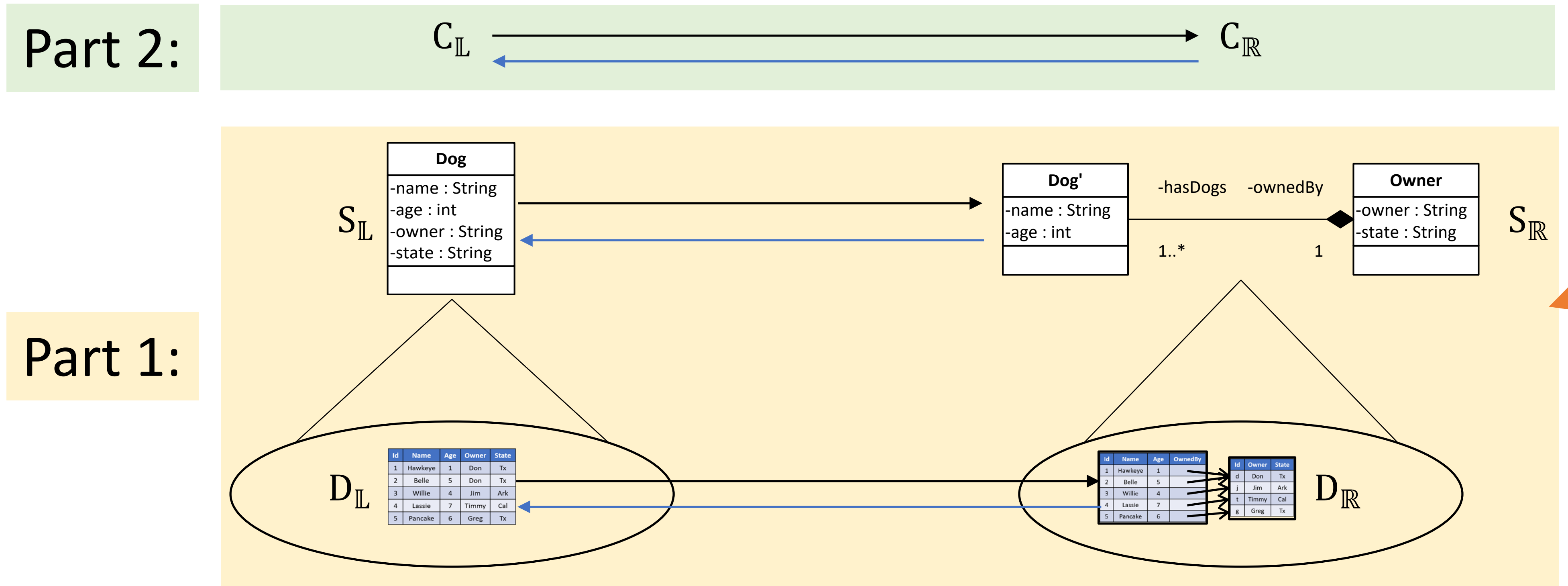
Big Picture of Upcoming Slides

- "Model" is a 3-tuple (OCL Constraints, Schema, Database)
- Normalize: $\boldsymbol{N}(\mathfrak{m}_{\mathbb{L}}) = \mathfrak{m}_{\mathbb{R}}$ and $\boldsymbol{N}^{-1}(\mathfrak{m}_{\mathbb{R}}) = \mathfrak{m}_{\mathbb{L}}$



Big Picture of Upcoming Slides

- **Part 1:** how we verified refactorings of schemas and their databases
  - MDE: a transformation of a type and its instances is a ***co-transformation***
- **Part 2:** how OCL constraints *might* be refactored (and other interesting ***CT*** ideas)



Part 1:

Verifying Schema & Database Refactorings

~12 minutes

What to Prove?

RoundTrip Theorems:

schema:

$\mathbb{A} = \boldsymbol{N}_{\boldsymbol{s}}^{-1} \cdot \boldsymbol{N}_{\boldsymbol{s}}(\mathbb{A})$

$\mathbb{B} = \boldsymbol{N}_{\boldsymbol{s}} \cdot \boldsymbol{N}_{\boldsymbol{s}}^{-1}(\mathbb{B})$

database:

$\forall\, a \in \mathbb{A}:\; a = \boldsymbol{N}_{\boldsymbol{d}}^{-1} \cdot \boldsymbol{N}_{\boldsymbol{d}}(a)$

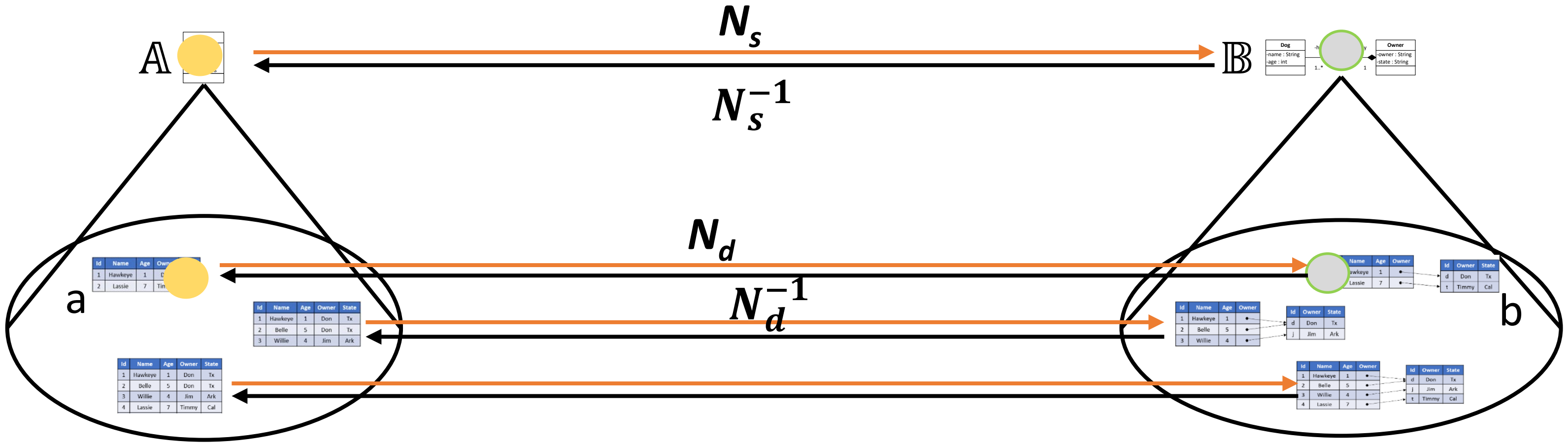
$\forall\, b \in \mathbb{B}:\; b = \boldsymbol{N}_{\boldsymbol{d}} \cdot \boldsymbol{N}_{\boldsymbol{d}}^{-1}(b)$

fixed schemas
Easy!

∀ schema instances
Need a Theorem Prover

Prove domains $\mathbb{A}$ and $\mathbb{B}$ are isomorphic

Now a vanilla problem in theorem proving



Described in *ACM TOSEM April 2023*

On Proving the Correctness of Refactoring Class Diagrams of MDE Metamodels

NAJD ALTOYAN* and DON BATORY, The University of Texas at Austin, USA

Model Driven Engineering (MDE) is general-purpose engineering methodology to elevate system design, maintenance, and analysis to corresponding activities on models. Models (graphical and/or textual) of a target application are automatically transformed into source code, performance models, Promela files (for model checking), and so on for system analysis and construction.

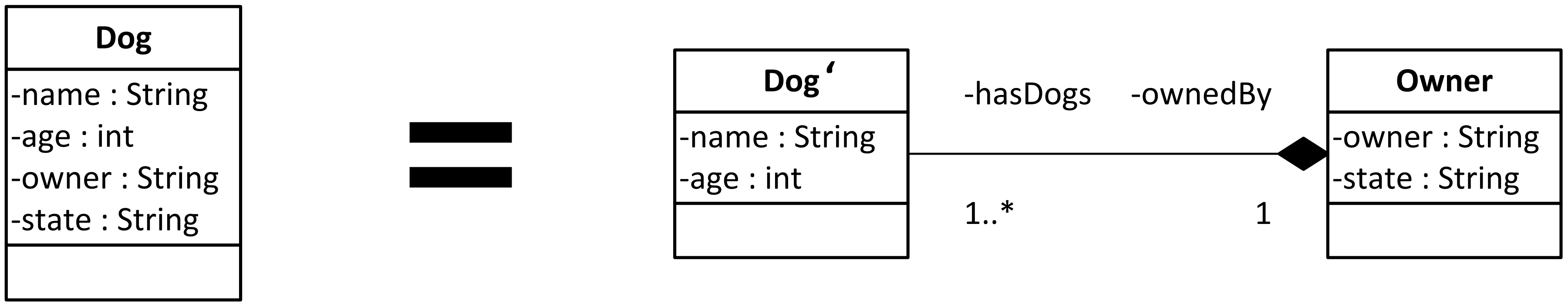
Models are instances of *metamodels*. One form an MDE metamodel can take is a [class diagram, constraints] pair: the class diagram defines all object diagrams that could be metamodel instances; OCL constraints eliminate semantically undesirable instances.

A metamodel *refactoring* is an invertible semantics-preserving co-transformation, i.e., it transforms both a metamodel *and* its models without losing data. This paper addresses a subproblem of metamodel refactoring: how to prove the correctness of refactorings of class diagrams without OCL constraints using the Coq Proof Assistant.

- We identified a set non-trivial meta-model refactorings (including normalize)
- Proved these refactorings correct using the ***CT*** structure of prior slides and used the Coq Proof Assistant (Rooster theorem prover)
- Proof details are in the paper

What we learned ⇒ You should learn too!

- About Coq (Rooster) Proof Assistant
  - Remarkable tool, but is ***not*** the theorem prover to use in the future for these problems
  - Why?   Answer: it is at the wrong level of abstraction
  - Proofs are much too difficult
  - **All meta-model refactorings we encountered define relational algebra equalities**

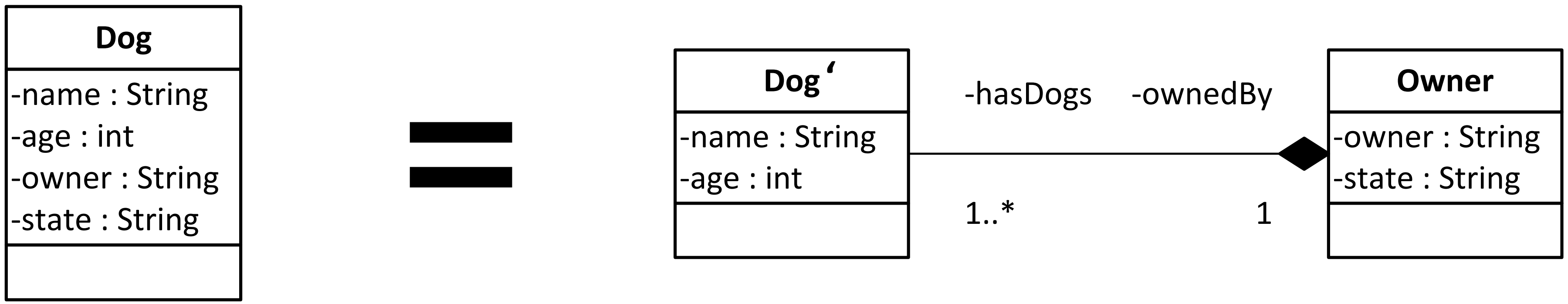


$$\textbf{Dog} = \textbf{Dog}' \bowtie \textbf{Owner}$$

$$\textbf{Dog}' = \Pi_{\text{name,age,owner}}\ \textbf{Dog}$$
$$\textbf{Owner} = \Pi_{\text{owner,state}}\ \textbf{Dog}$$

What we learned ⇒ You should learn too!

- Relational algebra operations are **co-transformations**
- **One proof suffices for both the schema AND the database level!**
- Still some problems to solve…



$$\mathbf{Dog} = \mathbf{Dog}' \bowtie \mathbf{Owner}$$

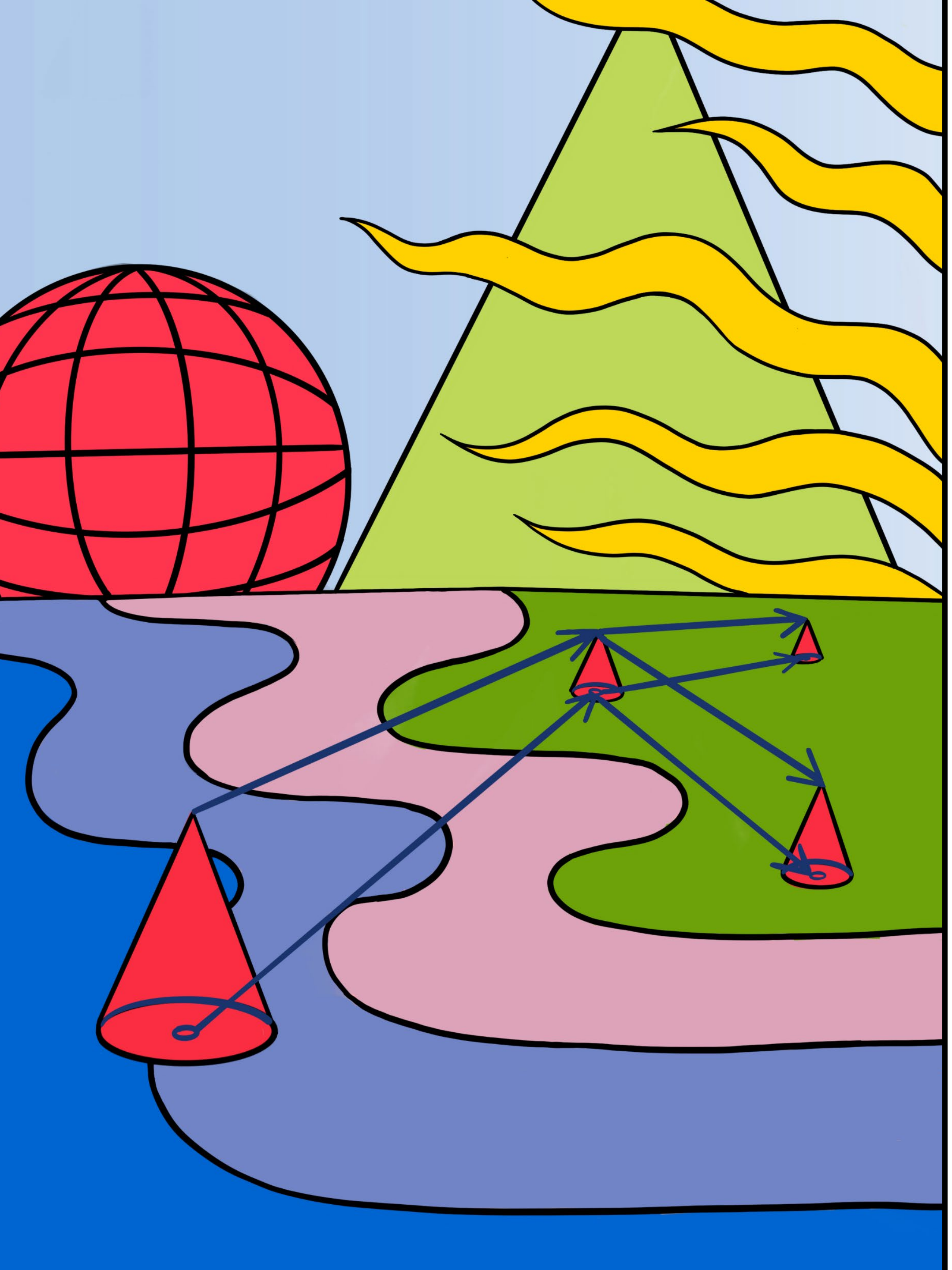
$$\mathbf{Dog}' = \Pi_{\mathbf{name,age,owner}}\ \mathbf{Dog}$$
$$\mathbf{Owner} = \Pi_{\mathbf{owner,state}}\ \mathbf{Dog}$$

What we learned ⇒ You should learn too!

- What is needed doesn't yet exist

- Need a **theorem prover of relational algebra (*R_A*)** identities
- Define MDE refactorings as compositions of ***R_A*** operations
- State refactoring as a ***R_A*** identity + verify!
- Potential for a big impact

Ph.D. Topic



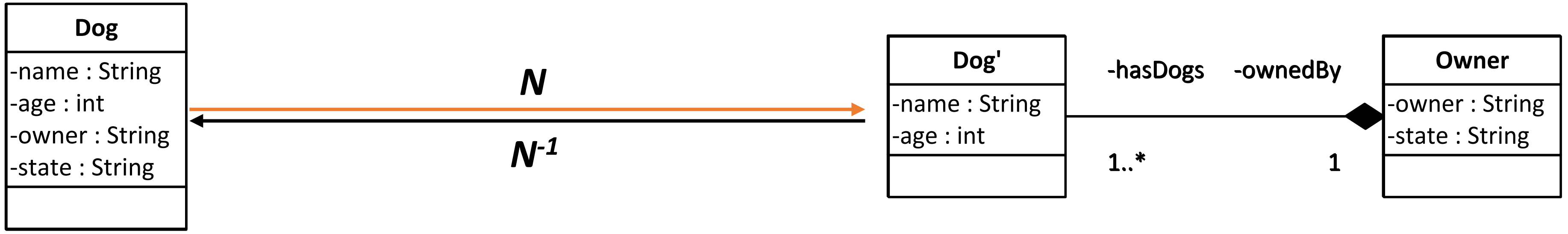
Not Finished!
More Embeddings

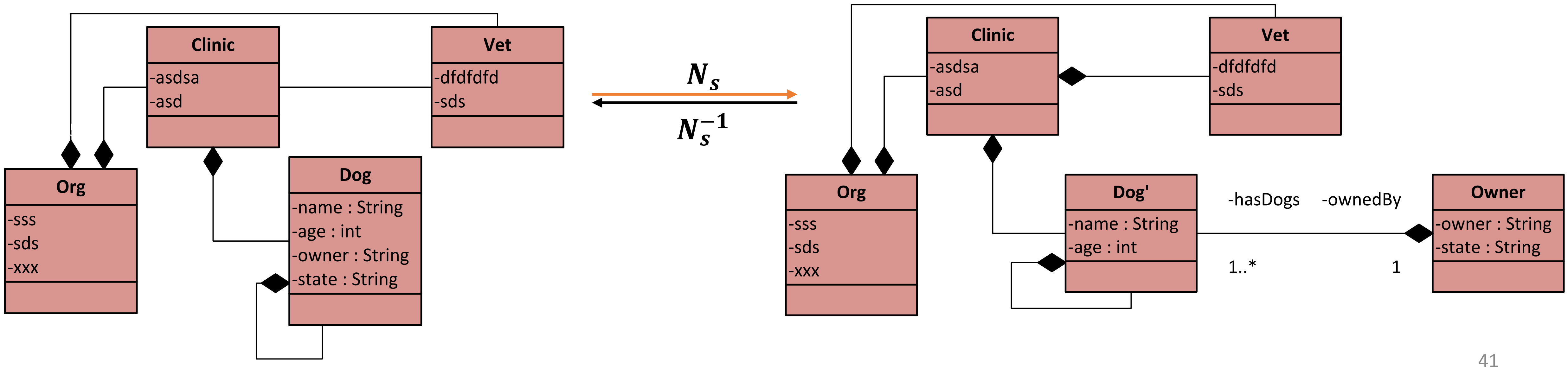
may be controversial

Ph.D. Topic

Embeddings Again!

- Class diagrams are rarely this isolated:





What to Prove?

RoundTrip Theorems:

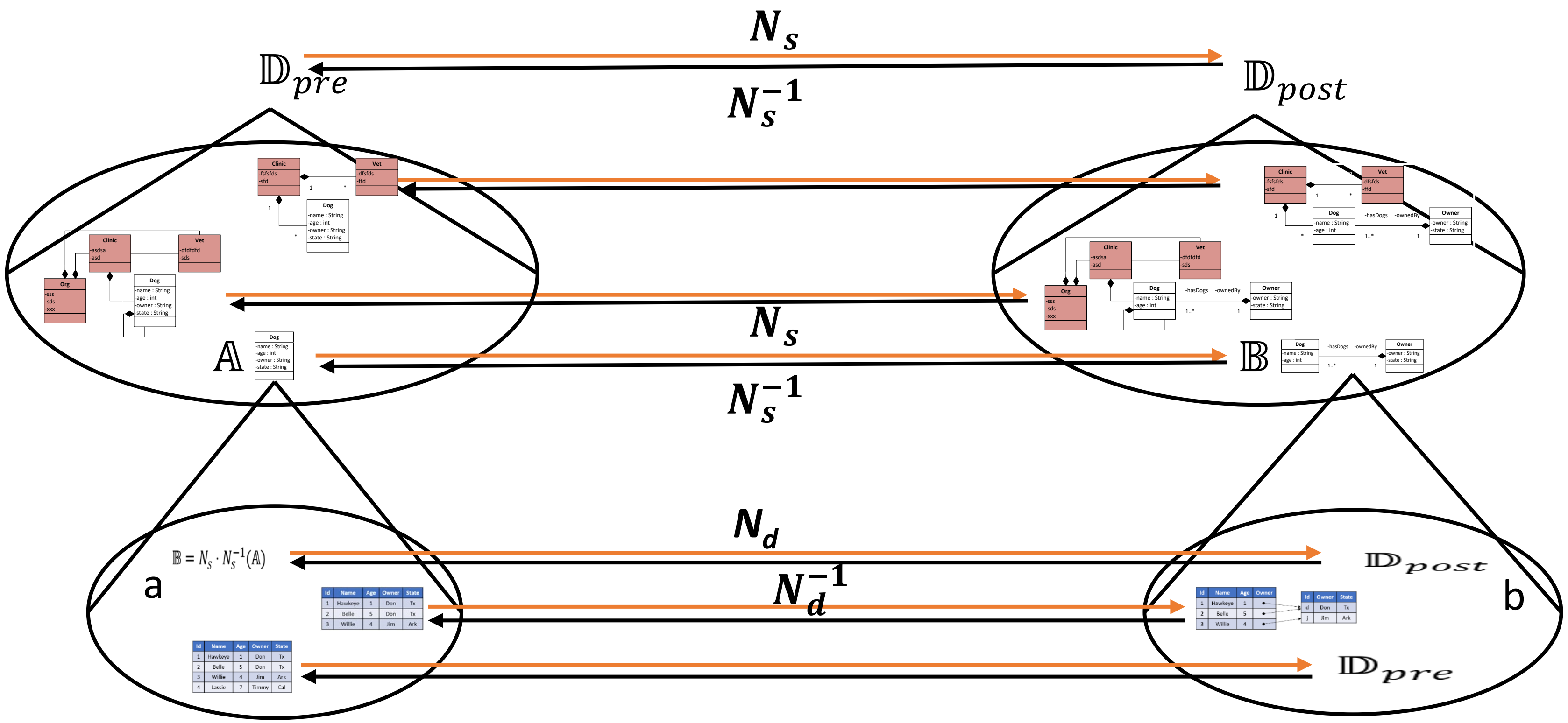
$\forall\, \mathbb{A} \in \mathbb{D}_{pre}:$	$\mathbb{A} = N_s^{-1} \cdot N_s(\mathbb{B})$
$\forall\, \mathbb{B} \in \mathbb{D}_{post}:$	$\mathbb{B} = N_s \cdot N_s^{-1}(\mathbb{A})$

Need a Theorem Prover

$\forall\, a \in \mathbb{A}:$	$a = \mathbb{R}^{-1} \cdot \mathbb{R}(a)$
$\forall\, b \in \mathbb{B}:$	$b = \mathbb{R} \cdot \mathbb{R}^{-1}(b)$

Need a Theorem Prover

domain of all schemas that satisfy ***N***'s preconditions



domain of all schemas that satisfy ***N***'s postconditions

What to Prove?

RoundTrip Theorems:

$$\forall\, \mathbb{A} \in \mathbb{U}_{pre} : \quad \mathbb{A} = \boldsymbol{N_s^{-1}} \cdot \boldsymbol{N_s}(\mathbb{B})$$

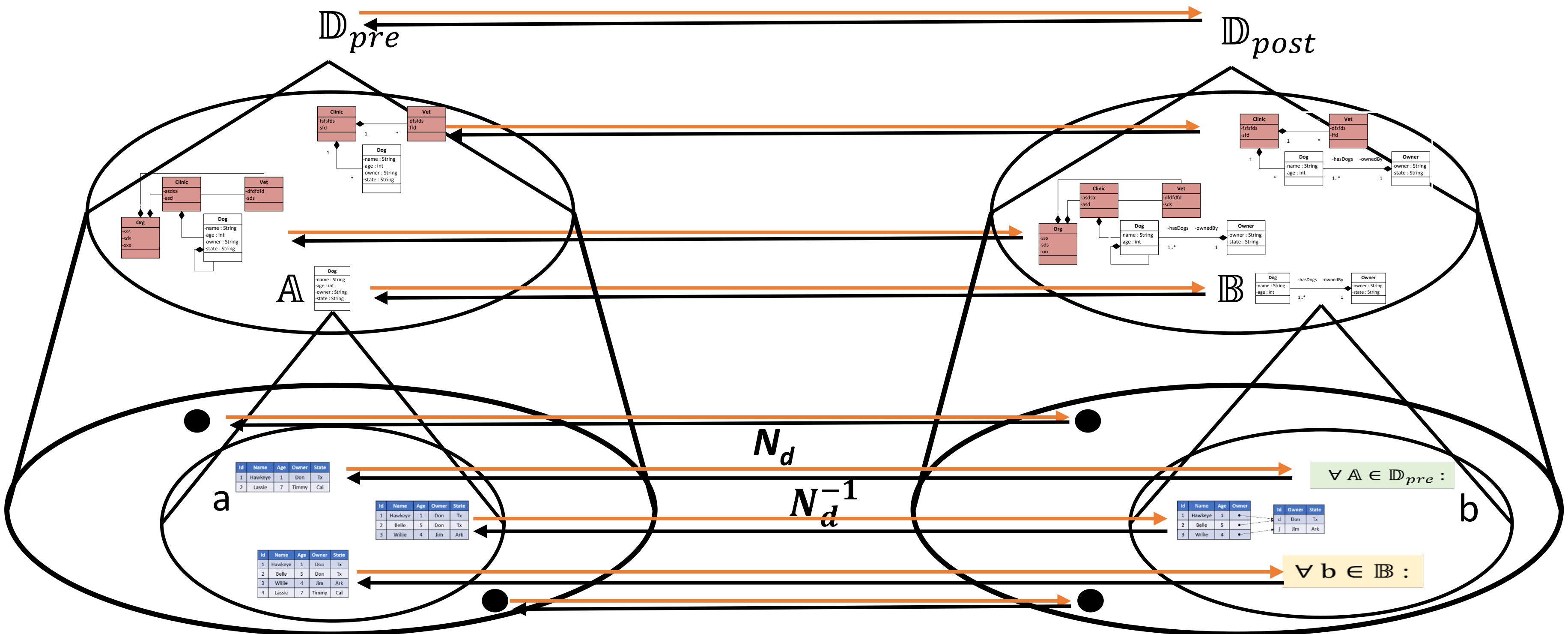
$$\forall\, \mathbb{B} \in \mathbb{U}_{post} : \quad \mathbb{B} = N_s \cdot N_s^{-1}(\mathbb{A})$$

Need a Theorem Prover

$$\forall\, \mathbb{A} \in \mathbb{D}_{pre} : \quad \forall\, \mathrm{a} \in \mathbb{A} : \quad \mathrm{a} = \mathbb{R}^{-1} \cdot \mathbb{R}(\mathrm{a})$$

$$\forall\, \mathbb{B} \in \mathbb{D}_{post} : \quad \forall\, \mathrm{b} \in \mathbb{B} : \quad \mathrm{b} = \mathbb{R} \cdot \mathbb{R}^{-1}(\mathrm{b})$$

Need a Theorem Prover



What to Prove?

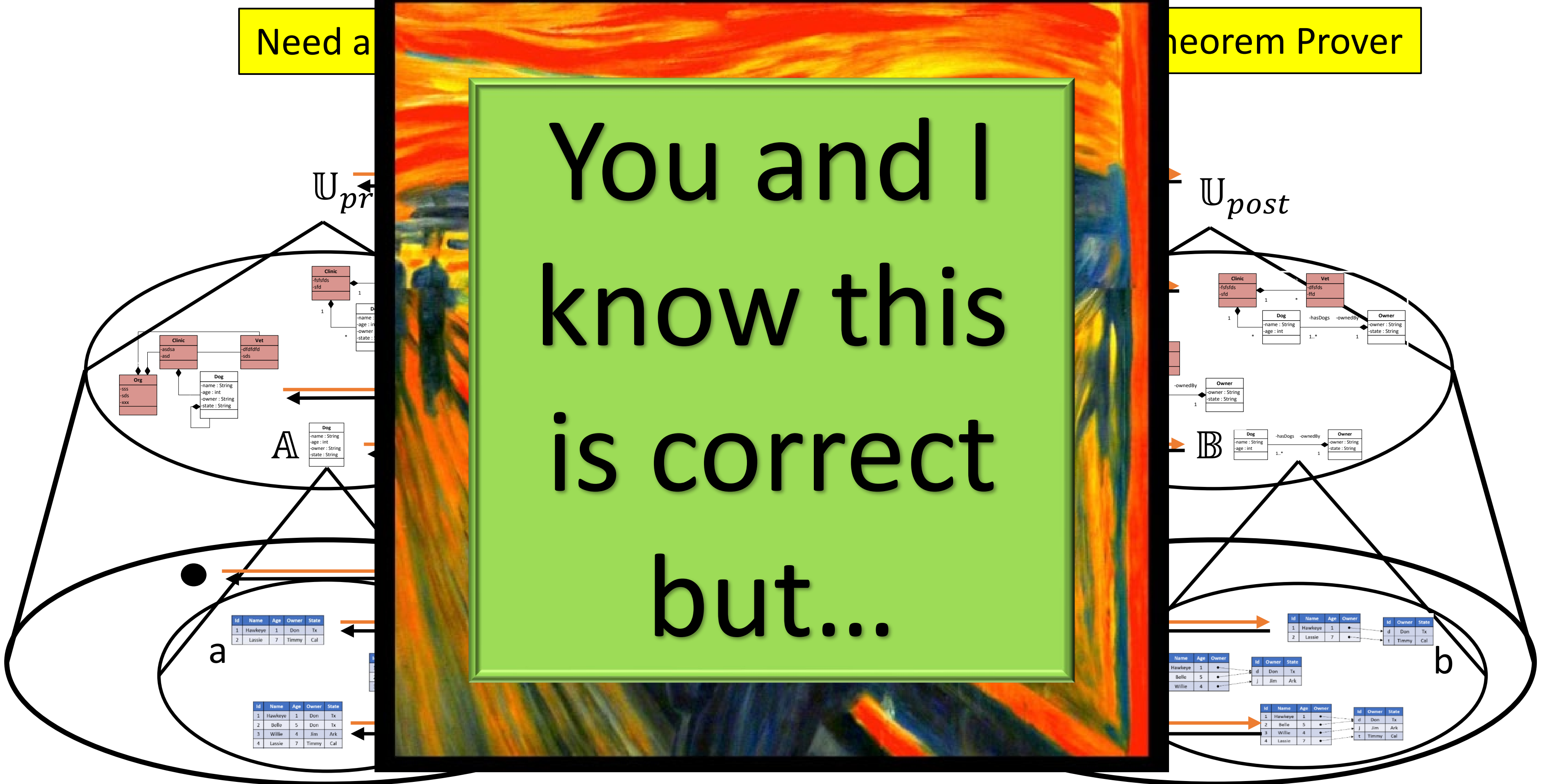
RoundTrip Theorems:

$\forall\, \mathbb{A} \in \mathbb{U}_{pre} : \quad \mathbb{A} = \boldsymbol{N_s^{-1}} \cdot \boldsymbol{N_s}(\mathbb{B})$

$\forall\, \mathbb{B} \in \mathbb{U}_{post} : \quad \mathbb{B} = N_s \cdot N_s^{-1}(\mathbb{A})$

$\forall\, \mathbb{A} \in \mathbb{D}_{pre} : \quad \forall\, a \in \mathbb{A} : \quad a = \mathbb{R}^{-1} \cdot \mathbb{R}(a)$

$\forall\, \mathbb{B} \in \mathbb{D}_{post} : \quad \forall\, b \in \mathbb{B} : \quad b = \mathbb{R} \cdot \mathbb{R}^{-1}(b)$



Time To Rethink the Problem

- The Problem: MDE thinking deals with explicit pointers and associations
- Pointers are the problem!!   Their copies are ***not*** identical
- We took this into account in our Rooster proofs and it complicated things…

Id	Name	Age	OwnedBy
1	Hawkeye	1	
2	Belle	5	
3	Willie	4	
4	Lassie	7	
5	Pancake	6	

Id	Owner	State
d	Don	Tx
j	Jim	Ark
t	Timmy	Cal
g	Greg	Tx

Id	Name	Age	OwnedBy
1	Hawkeye	1	
2	Belle	5	
3	Willie	4	
4	Lassie	7	
5	Pancake	6	

Id	Owner	State
d	Don	Tx
j	Jim	Ark
t	Timmy	Cal
g	Greg	Tx

Time To Rethink the Problem

- My Conjecture: MDE thinking deals with explicit pointers and associations
- Pointers are the problem!! Their copies are *not* identical
- We took this into a … complicated things…

4	Lassie	7	
5	Pancake	6	

lesson RDB community learned 45 years ago ☹

~~For refactoring verification,~~

avoid pointers

~~in proofs!~~

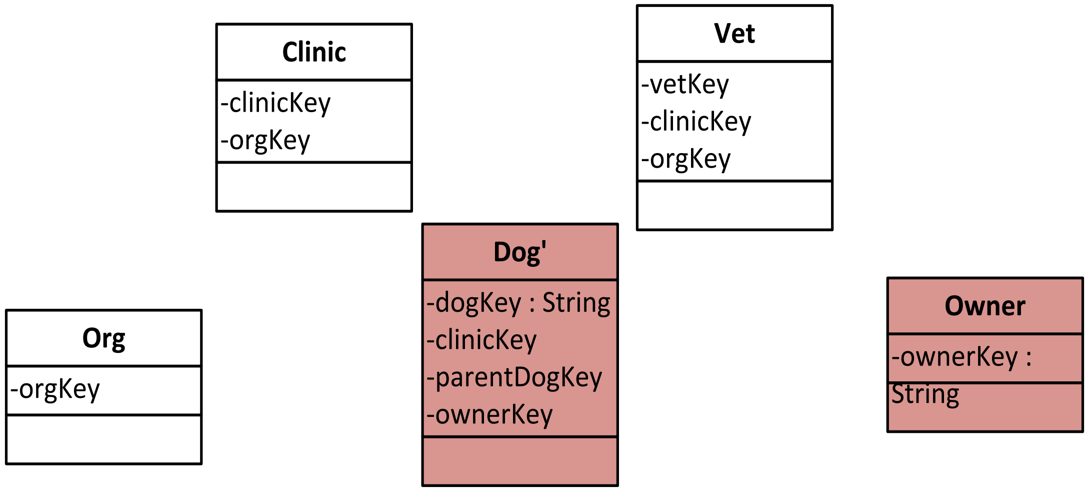
~~Too low level~~

…wnedBy

Id	Owner	State
d	Don	Tx
j	Jim	Ark
t	Timmy	Cal
g	Greg	Tx

Use Database Abstractions Instead

- No explicit pointers and associations! Still there: They are implemented differently



Keys

Id	Name	Age	OwnedBy
1	Hawkeye	1	d
2	Belle	5	d
3	Wille	4	j
4	Lassie	7	t
5	Pancake	6	g

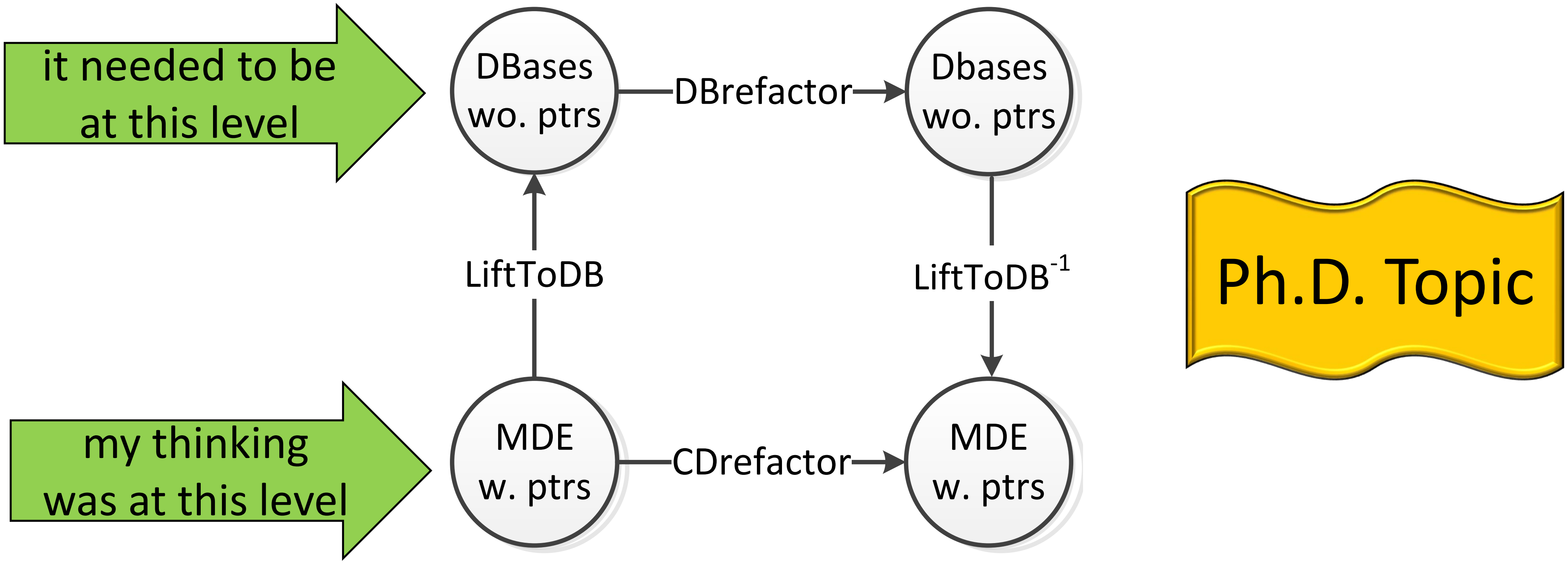
Id	Owner	State
d	Don	Tx
j	Jim	Ark
t	Timmy	Cal
g	Greg	Tx

- Why is this important?

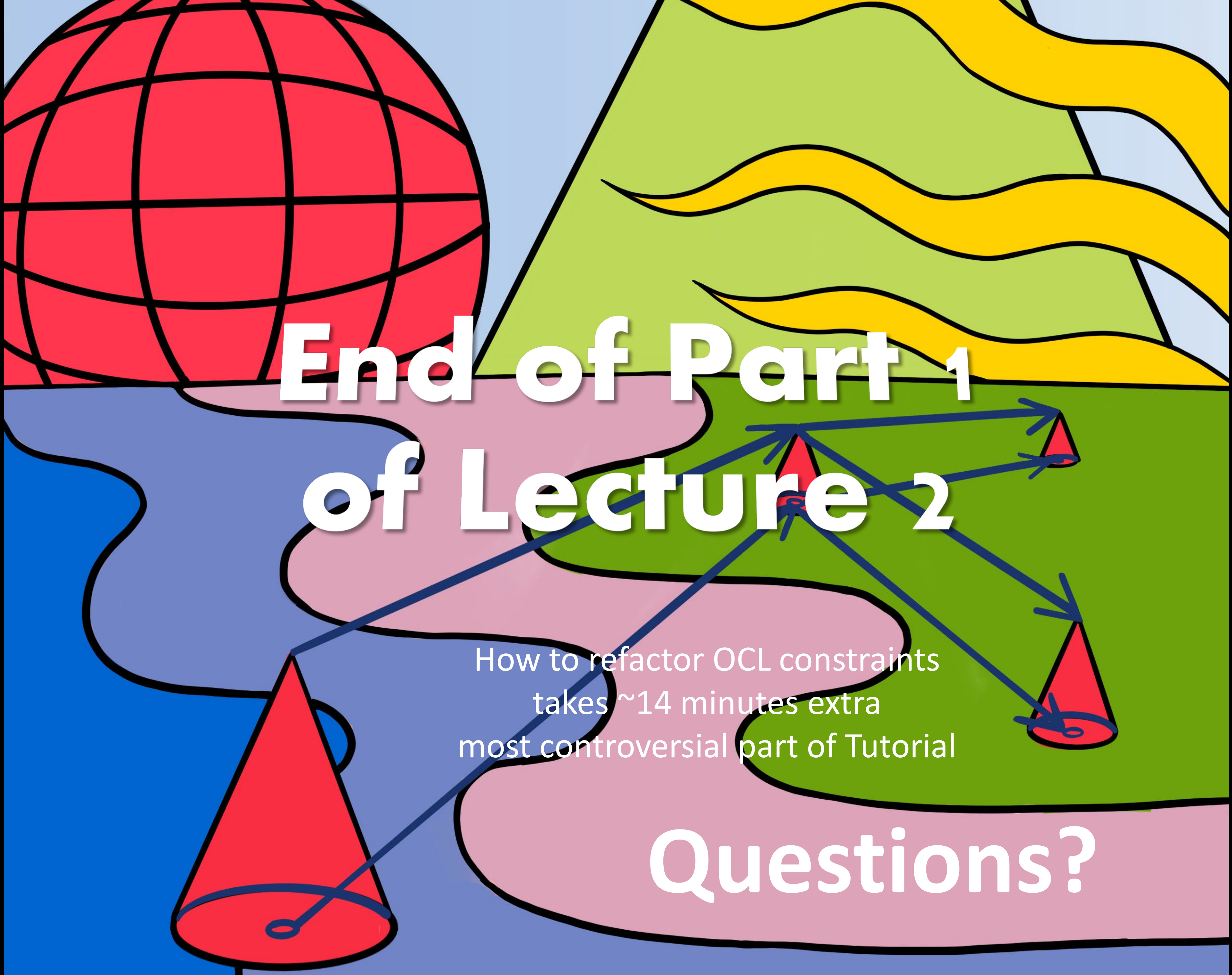
Ans: Refactorings are localized

Proofs using non-embedded schemas may suffice!

Think in Terms of Categories!



- Simplify proofs when dealing with domains like $\mathbb{U}_{pre}$ and $\mathbb{U}_{post}$
- Limit proofs to largely local changes, independent of embeddings


End of Part 1
of Lecture 2
How to refactor OCL constraints
takes ~14 minutes extra
most controversial part of Tutorial
Questions?

Part 2:

How OCL constraints might be refactored

and other *CT* facts

~14 minutes

Part 2: How to Refactor OCL constraints

- Recall **Relational Algebra (*R_A*)**
  - select, project, join, count, exists… generic operations on tables
- OCL is a subset of ***R_A*** in OO syntax; OCL is ***R_A*** without proper join & projection operations

Aocl : A Pure-Java Constraint and Transformation Language for MDE

Don Batory and Najd Altoyan

Department of Computer Science
The University of Texas at Austin, Austin, Texas
{batory, naltoyan}@cs.utexas.edu

Abstract: OCL is a standard MDE language to express constraints. OCL has been criticized for being too complicated, over-engineered, and difficult to learn. But beneath OCL's complicated exterior is an elegant language based on relational algebra. We call this language $\mathbb{A}_{ocl}$, which has a straightforward implementation in Java. $\mathbb{A}_{ocl}$ can be used to write OCL-like constraints and model transformations in Java. A simple MDE tool generates an $\mathbb{A}_{ocl}$ Java 8.0 package from an input class diagram for $\mathbb{A}_{ocl}$ to be used.

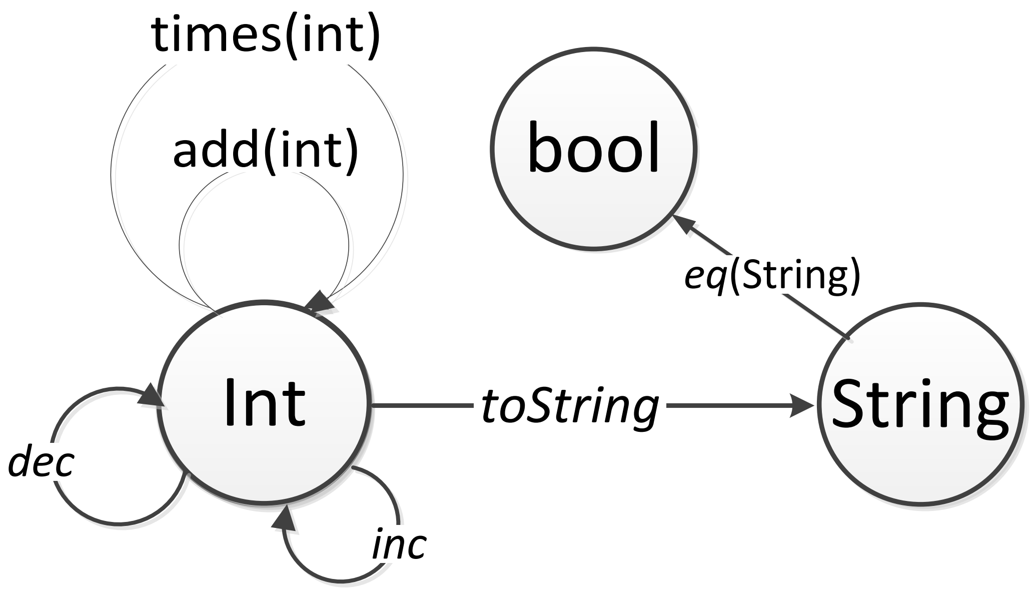
MODELSWARD 2020

- I'll present a sequence of facts about ***CT*** that seem unrelated but together you will see the above result unfold and why it is important

Fact #1: Every Category is an Algebra

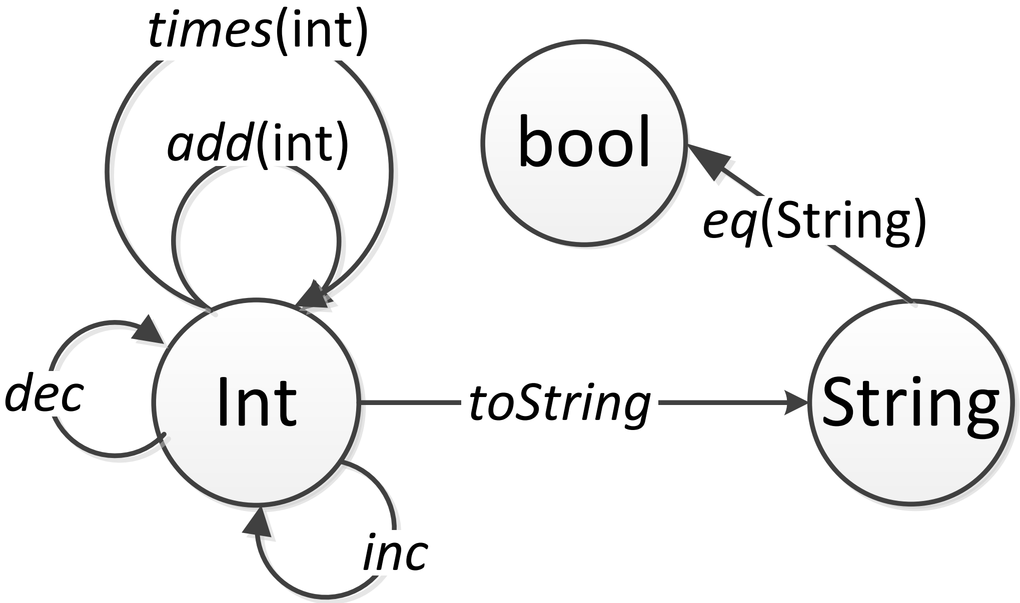
- An **algebra** is a set of operations with typed inputs and typed outputs
- A category diagram depicts an algebra

```
Int inc(Int)
Int dec(Int)
Int add(Int, Int)
Int times(Int,Int)
String toString(Int)
bool eq(String)
```



Fact #2: Every Category is a Finite State Machine

- Start at any node and end at any node…

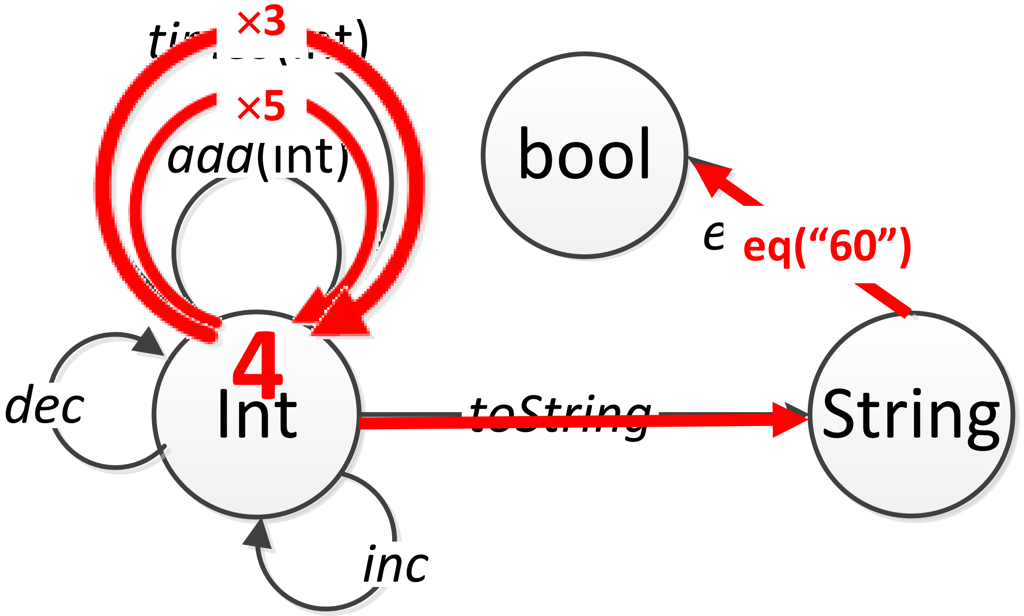


Fact #3: Every Path is a Type-Correct Expression

- A ***path*** of in algebra's category is an expression a composition of algebra operations

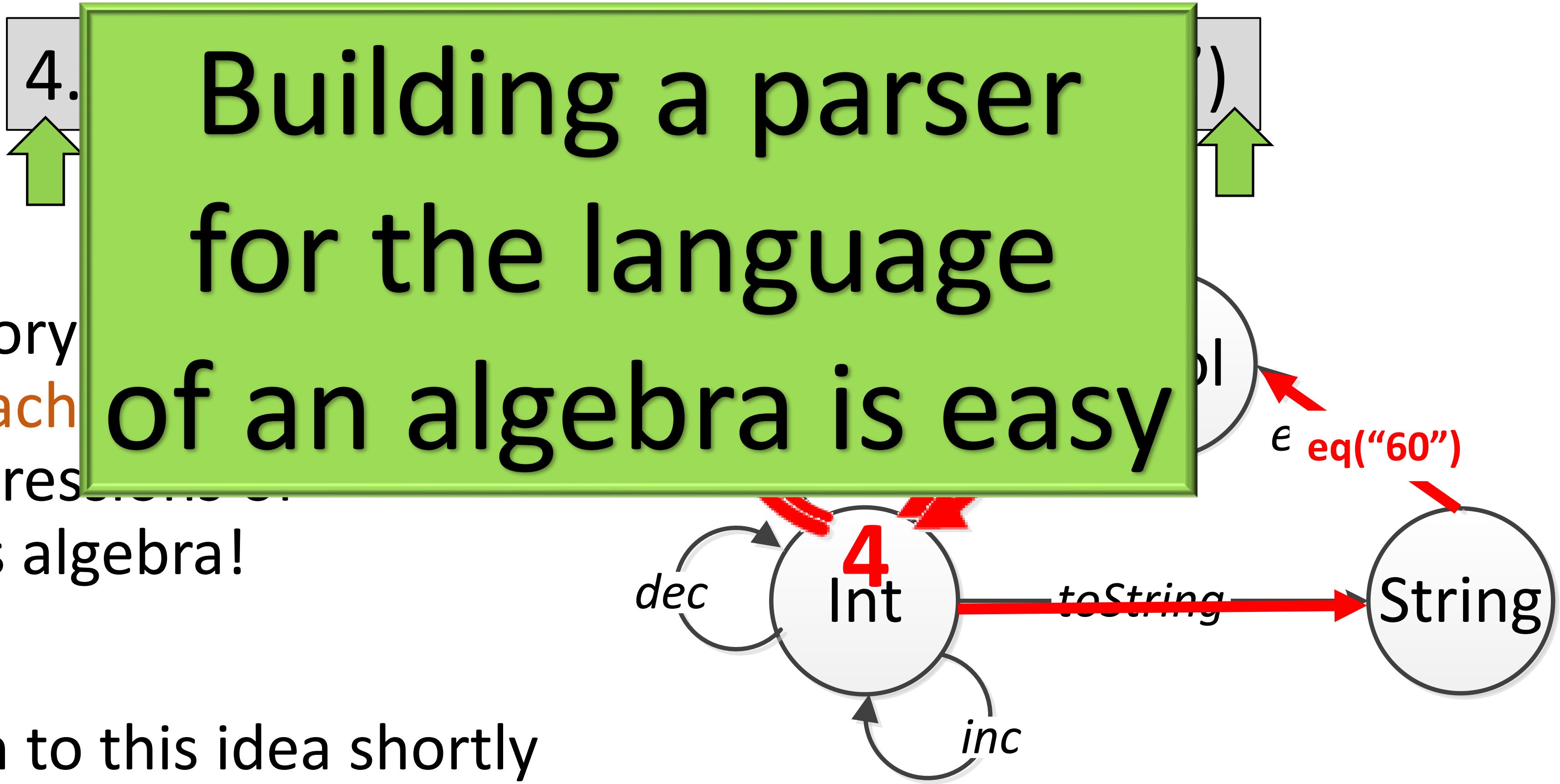
```
4.times(5).times(3).toString().eq("60")
```

- Given a category is a finite state machine ⇒ parser for expressions of the category's algebra!



Fact #3: Every Path is a Type-Correct Expression

- A ***path*** of in algebra's category is an expression a composition of algebra operations

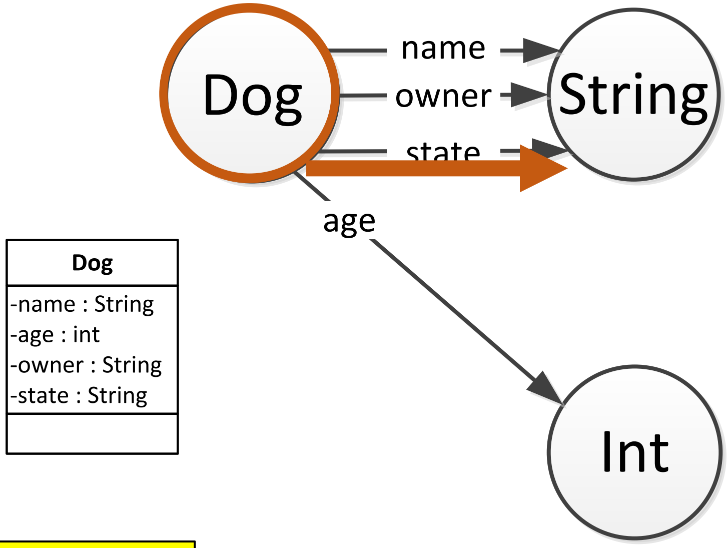


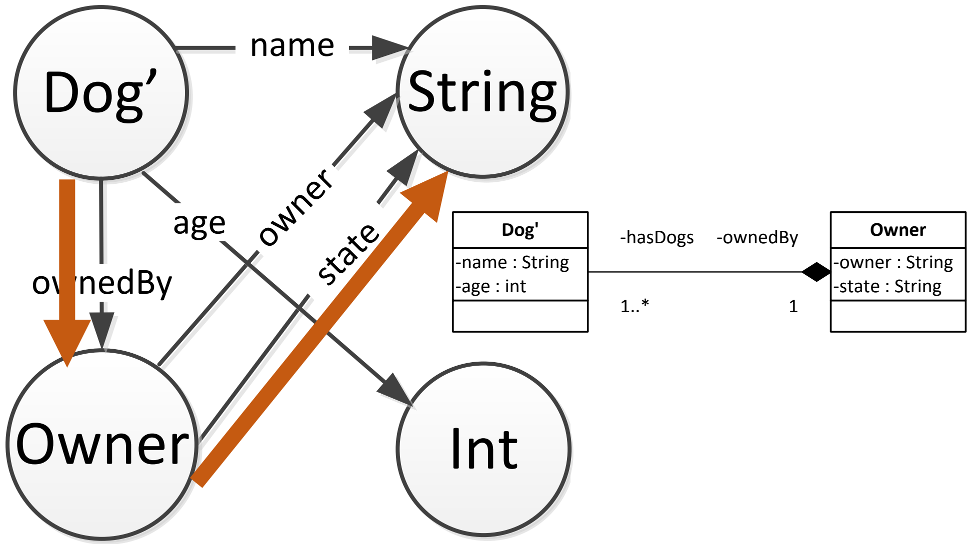
- Given a category
finite state mach
parser for expres
the category's algebra!

- We will return to this idea shortly

Fact #4: Functors seem "almost" perfect

- To translate an expr of one algebra into corresponding expr of another





Expression to refactor:

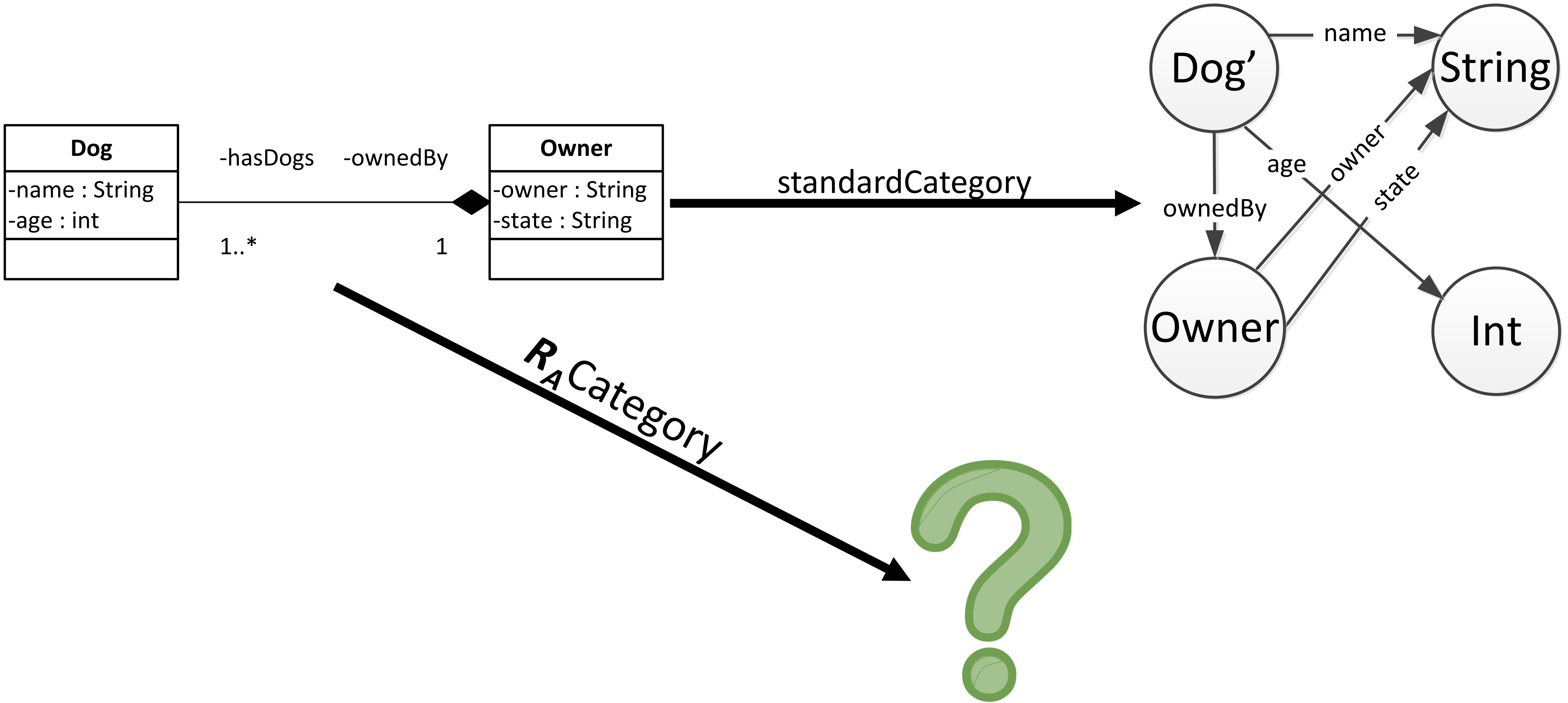
Dog.state()

Refactored expression:

Dog'.ownedBy().state()

Fact #5: $\boldsymbol{R_A}$ Categories

- Lecture #1 showed every class diagram has a **standard category**



Fact #5: Relational Algebra (R_A) Categories #1

Dog

-name : String
-age : int
-owner : String
-state : String

Dog Table

Id	Name	Age	OwnedBy
1	Hawkeye	1	Don
2	Belle	5	Don
3	Wille	4	Jim

Dog.sort(name).count() =

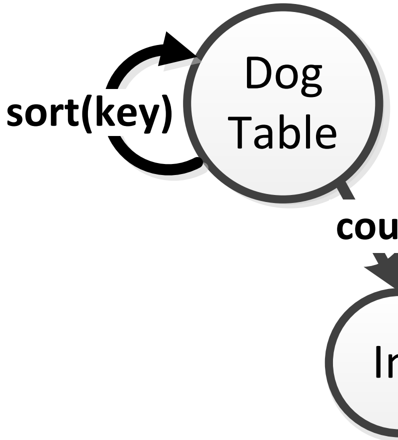
***R_A* Expression**

id	name	age	ownedBy
2	Belle	5	Don
1	Hawkeye	1	Don
3	Wille	4	Jim

.count() = 3

syntax & semantic similarity with OCL

Derived R_A Category



Fact #5: Relational Algebra ($\boldsymbol{R_A}$) Categories #2

Dog
-name : String -age : int -owner : String -state : String

Dog Table

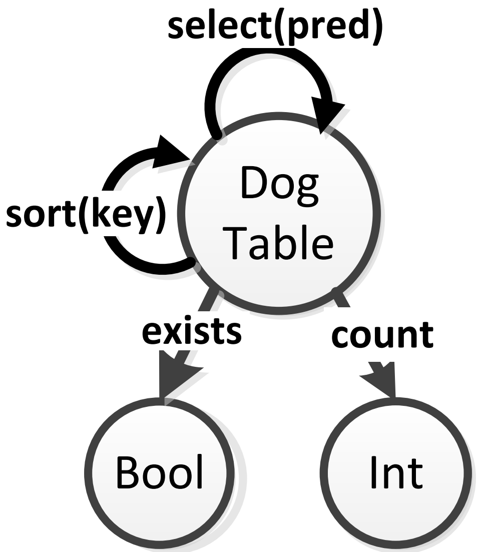
Id	Name	Age	OwnedBy
1	Hawkeye	1	Don
2	Belle	5	Don
3	Wille	4	Jim

Dog.select(d→d.age==4).exists() =

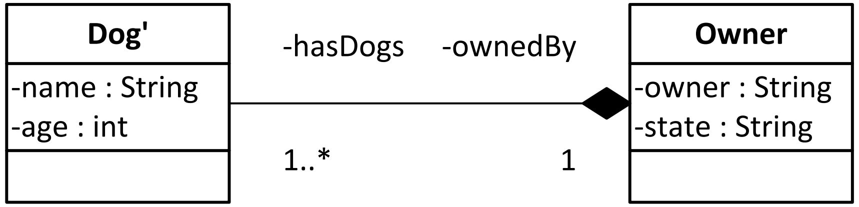
Id	Name	Age	OwnedBy
3	Wille	4	Jim

.exists() = True

Derived $\boldsymbol{R_A}$ Category

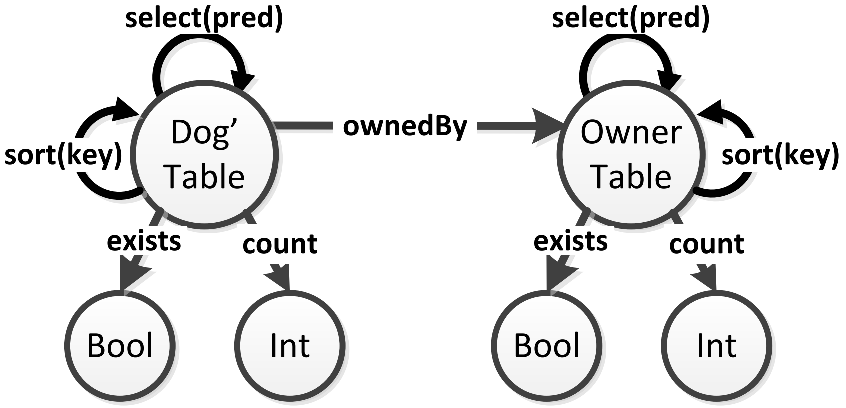


Fact #5: Association Traversal is a $R_A \rtimes$ (right-semijoin) #3



Id	Name	Age	OwnedBy
1	Hawkeye	1	d
2	Belle	5	d
3	Wille	4	j
4	Lassie	7	t
5	Pancake	6	g

Id	Owner	State
d	Don	Tx
j	Jim	Ark
t	Timmy	Cal
g	Greg	Tx



R_A Category

Dog'.select(d→d.age≤5).ownedBy() =

Id	Name	Age	OwnedBy
1	Hawkeye	1	d
2	Belle	5	d
3	Wille	4	j

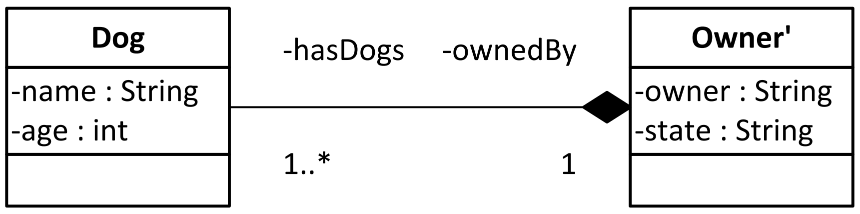
.ownedBy() =

Id	Owner	State
d	Don	Tx
j	Jim	Ark

owndBy()
in R_A is a
right semijoin

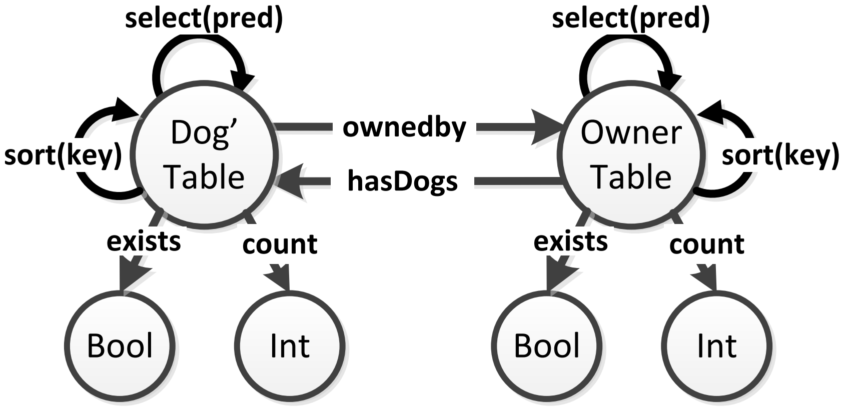
given table of Dogs, produce table of their owners

Fact #5: Association Traversal is a R_A ⋊ (right-semijoin) #4



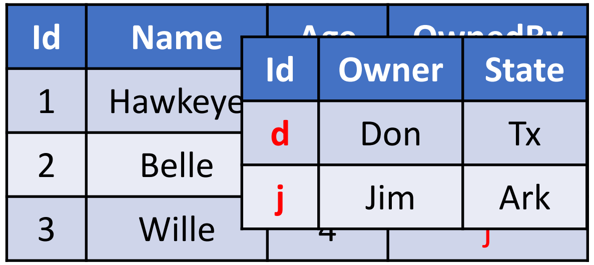
Id	Name	Age	OwnedBy
1	Hawkeye	1	d
2	Belle	5	d
3	Wille	4	j
4	Lassie	7	t
5	Pancake	6	g

Id	Owner	State
d	Don	Tx
j	Jim	Ark
t	Timmy	Cal
g	Greg	Tx



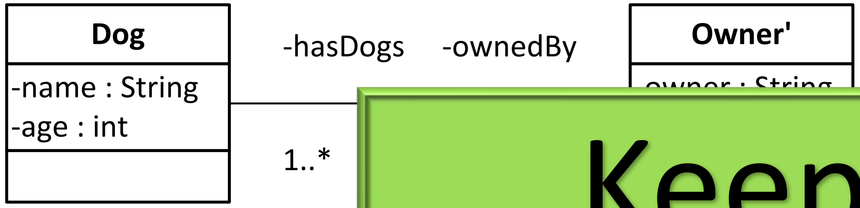
R_A Category

.hasDogs() =



hasDogs()
in R_A is a
right semijoin

Fact #5: Association Traversal is a $R_A \rtimes$ (right-semijoin) #4

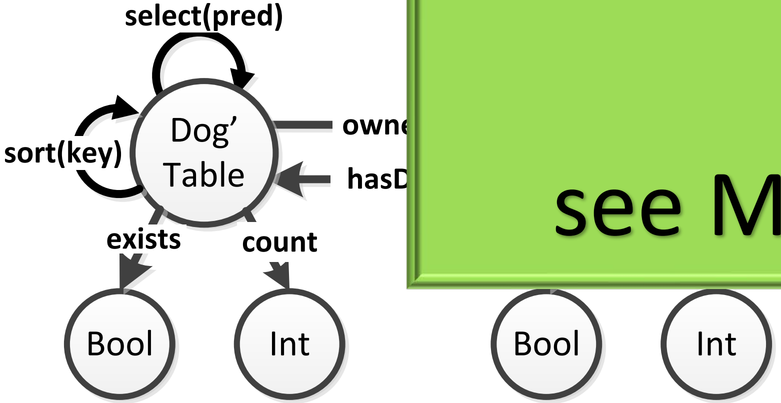


Id	Name	Age	OwnedBy
1	Hawkeye	1	d
2	Belle	5	d

Id	Owner	State
d	Don	Tx
[illegible]	[illegible]	Ark
[illegible]	[illegible]y	Cal
[illegible]	[illegible]	Tx

Keep going and you'll define an R_A approximation of OCL in *CT*

see MODELSWARD paper for details



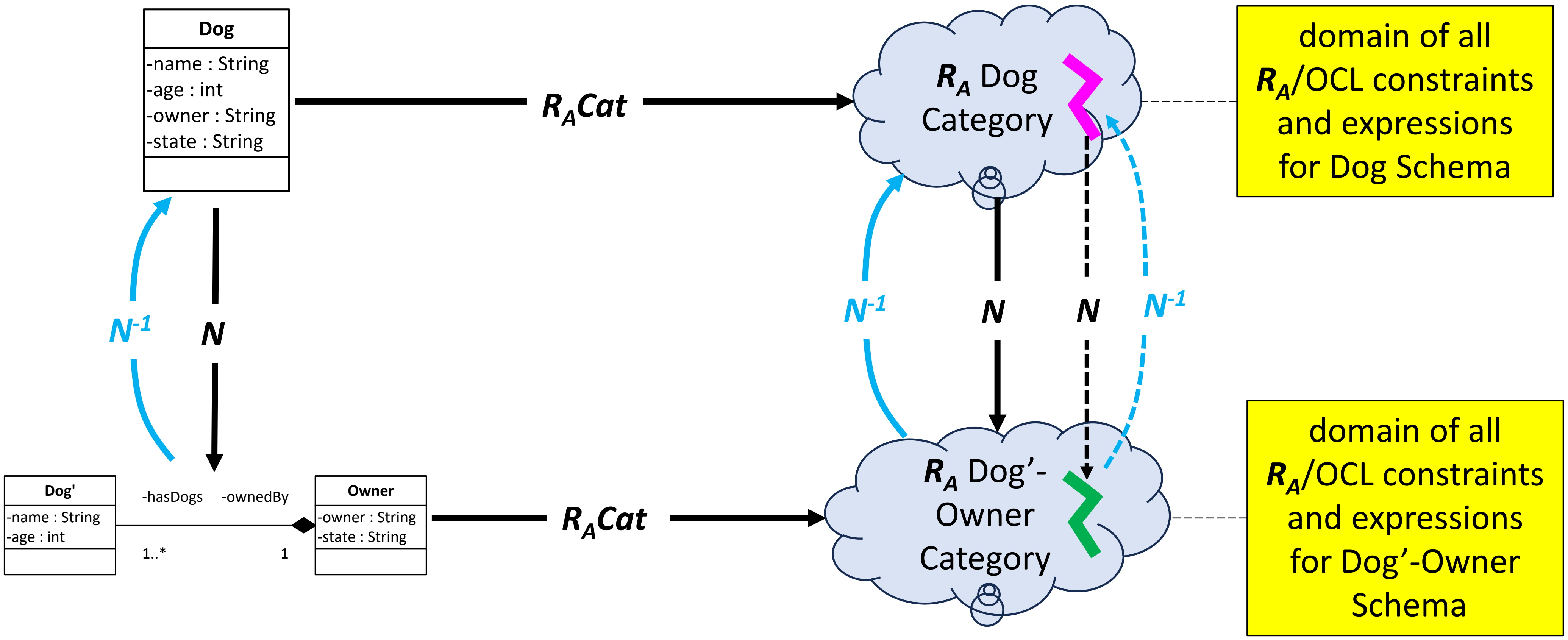
j	Jim	Ark

Id	Name	Age	OwnedBy
[illegible]	[illegible]e	1	d
[illegible]	[illegible]	5	d
3	Wille	4	j

hasDogs() in R_A is a **right semijoin**

R_A Category

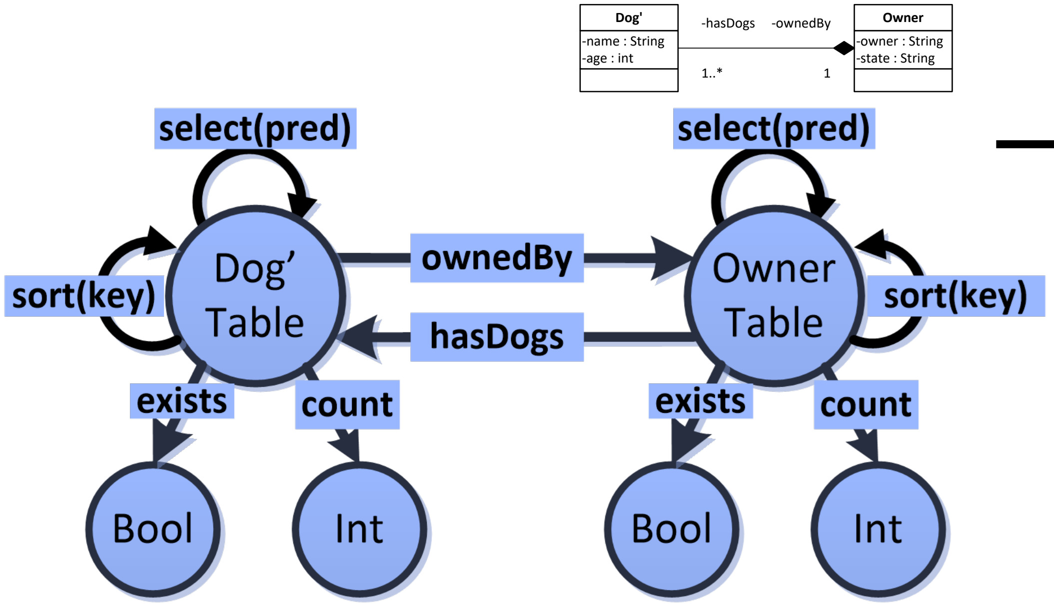
Putting it Together



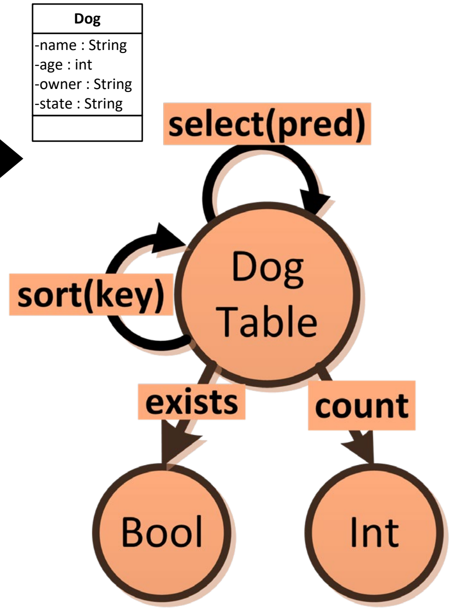
Looks promising to refactor OCL constraints!

What I learned that you should know!

- ***Refactorings that delete domains cause problems in static models!***



N^{-1} →



Database Solution: Dynamically Reconstruct Owner Table!

Owner
.select(o->o.state==“Tx”).count()<6

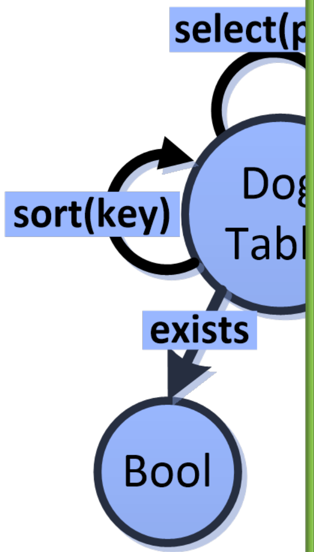
constraint’

how does constraint’ translate?

Dog.project(owner,state)
.select(o->o.state==“Tx”).count()<6

# of owners in Tx must be less than 6

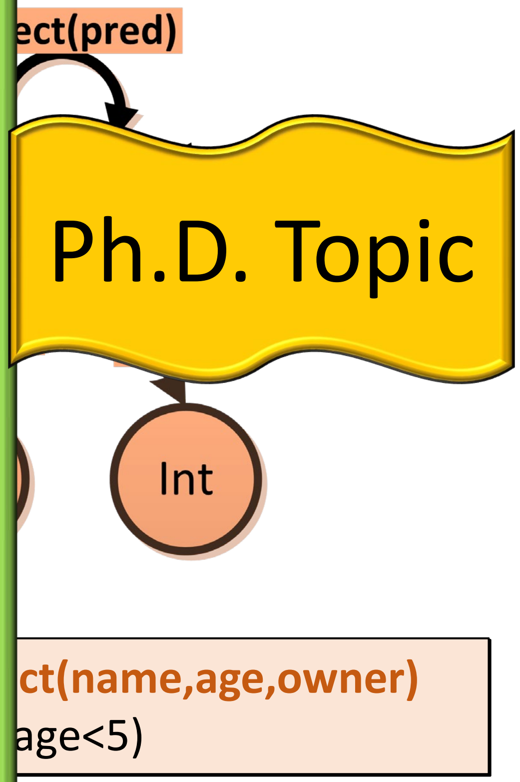
- ***Refacto*** ... ***models!***



Problem: Not sure this is possible in OCL!

Must dynamically recreate classes (tables) – that seems verboten to class diagrams!

If you could, then you can refactor OCL constraints!!

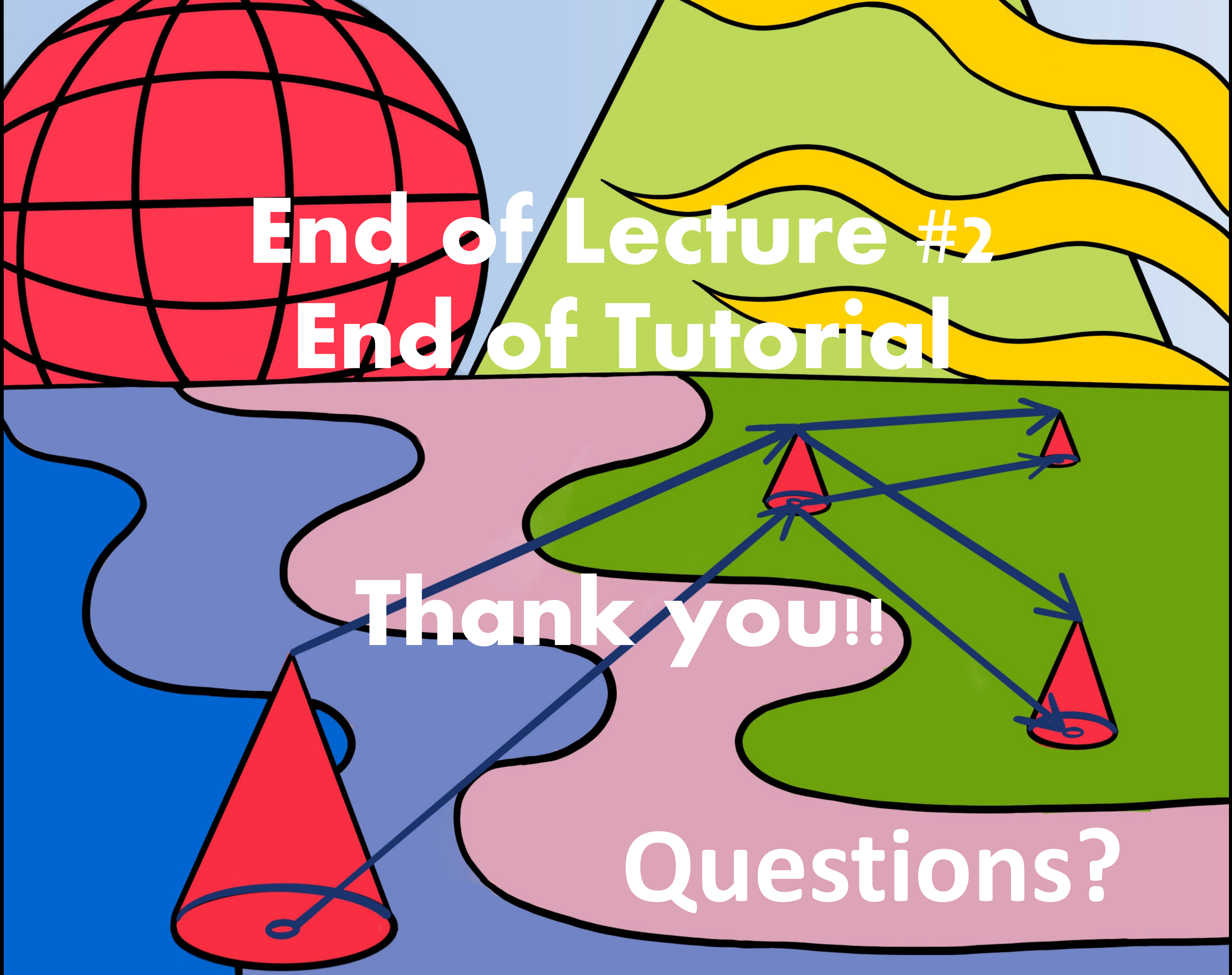


What You Should Have Learned

- ***CT*** is very powerful and practical way to think
  - Uses simple graphics to express complex ideas

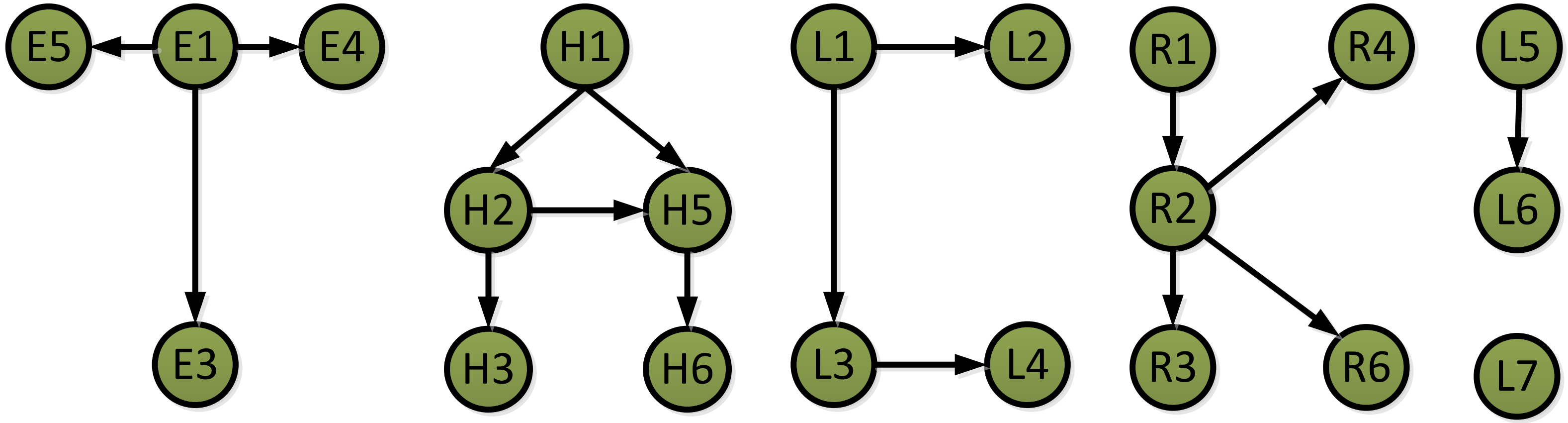
- Given
  - Refactorings in MDE are important
  - Refactoring of class diagrams & object diagrams & constraints are still open problems

- ***CT***
  - reveals the tasks required to verify refactorings
  - showed us where serious challenges lie (nested quantifiers)
  - encouraged broader thinking (using databases) to solve these problems


End of Lecture #2
End of Tutorial
Thank you!!
Questions?

The End of This Tutorial

- Most famous category in Swedish Language:



(Thank You!)