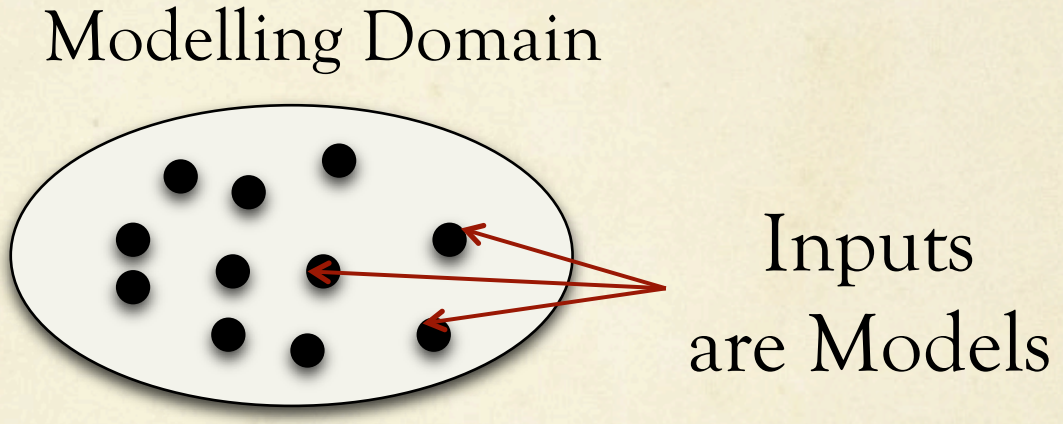
Building Effective Modelling Domains for Testing

Sagar Sen
Equipe Triskell, INRIA/IRISA
35042 Rennes, France

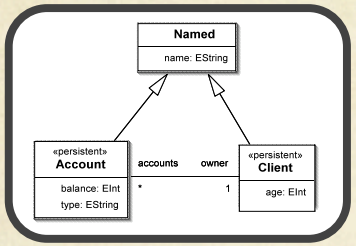
Outline

- Motivation
- Challenge : Building Effective Modelling Domains
- Solution Overview : Cartier
- Cartier : A Guided Tour
- Instance Generation using the Alloy API
- Validation of Test Cases: Mutation Analysis
- Experiments and Results
- Conclusion

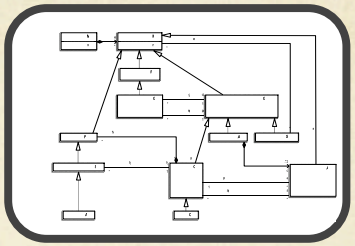
Motivation : The Modelling Domain



Motivation : Transformations



Input meta-model



Output meta-model

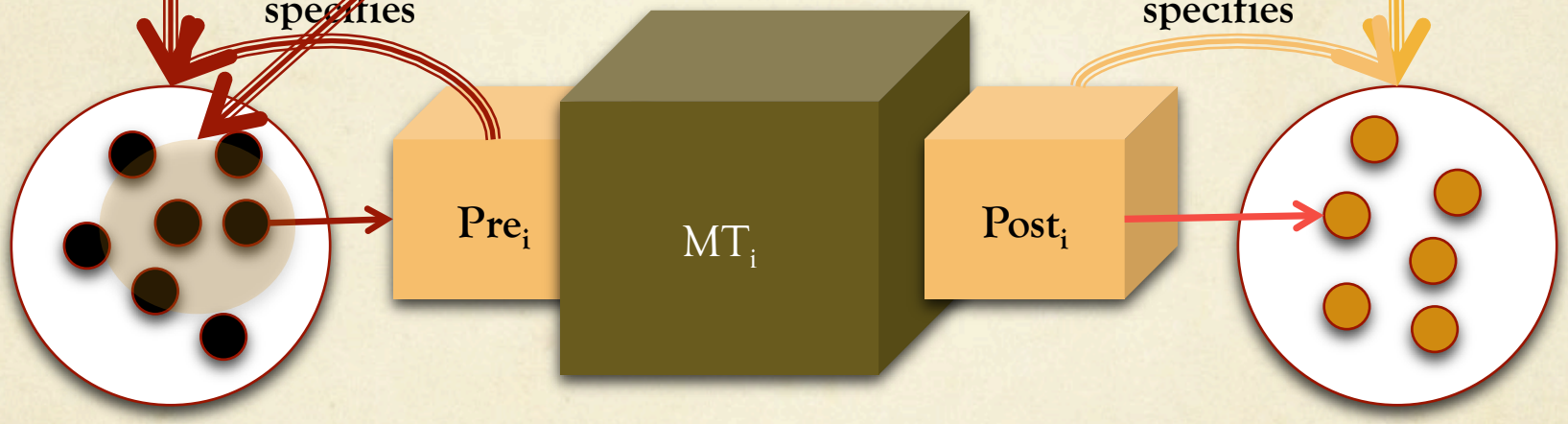
Test model knowledge

1. Model fragments
2. Test Model Objectives

specifies test models

specifies

specifies



Input Domain

Output Domain

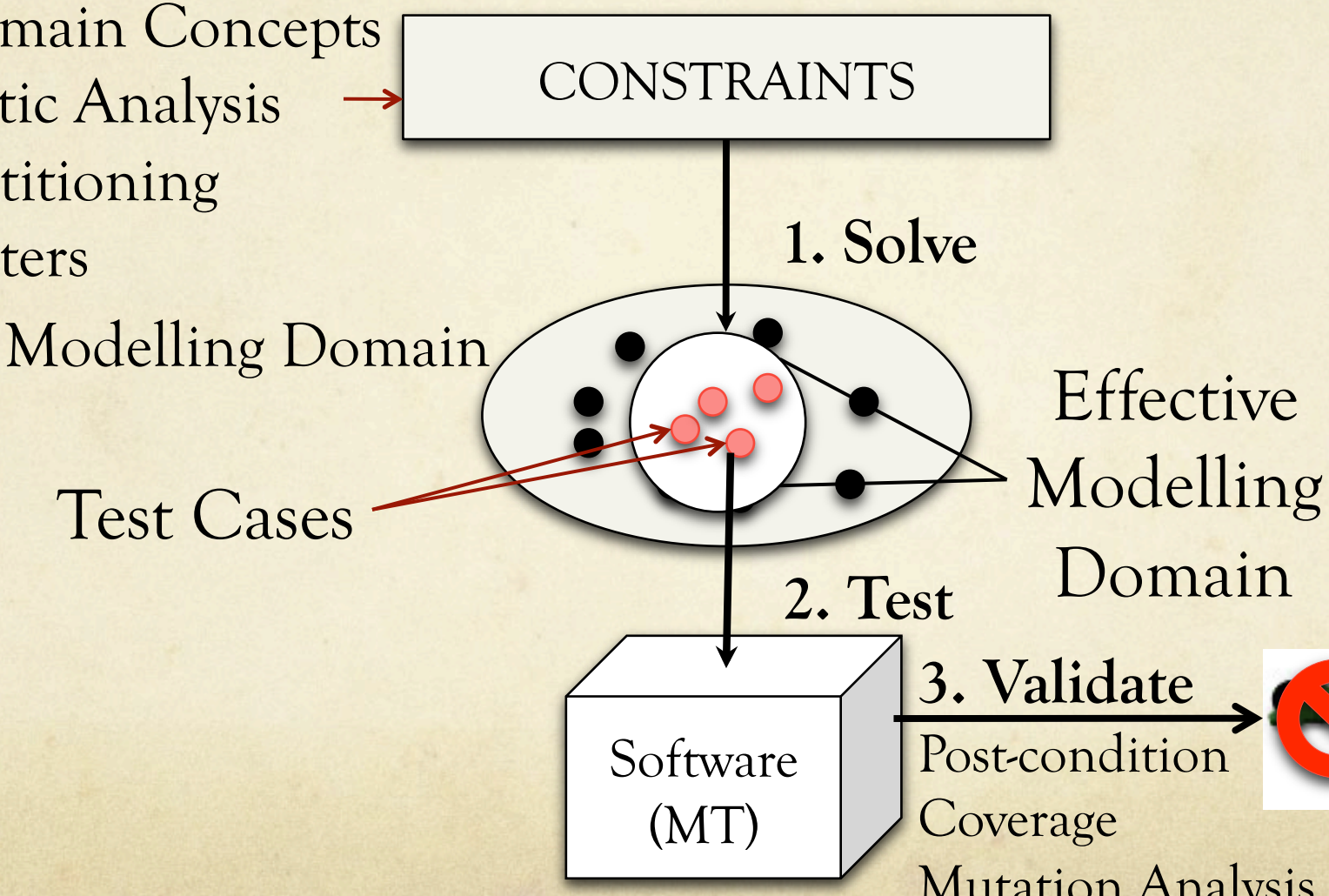
Question: *How to select test models using knowledge from* ***Input meta-model + Pre$_i$ + Test Model Knowledge?***

Motivation : Applications

1. *Model transformation pipelines such as XML to XML Pipelines (Eg: Xproc)*
2. *Code Generators (Eg: Simulink Model to Code)*
3. *Object Persistence (Eg: Hibernate 3.0)*
4. *Simulation Traces (Eg: Simulink, Modelica)*
5. *UML Model Design Re-factoring (Eg: Rhapsody)*
6. *Code migration (Cobol to Java/C#, VPLUS UI to XML/J2EE)*
7. *Game development pipelines (Eg: XNA Studio)*
8. *Business Process Models (Eg: Websphere business process model)*
9. *Code reverse engineering*
10. *Graphical Editors ...*

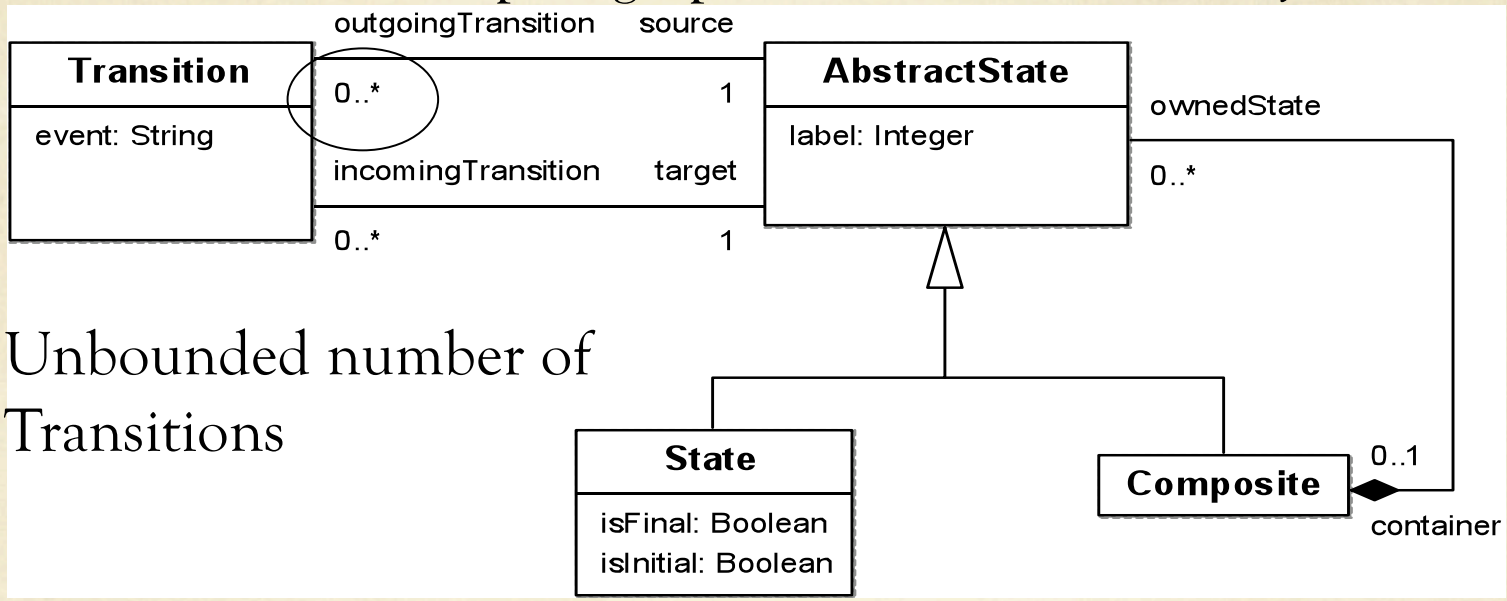
Motivation : The Testing Context

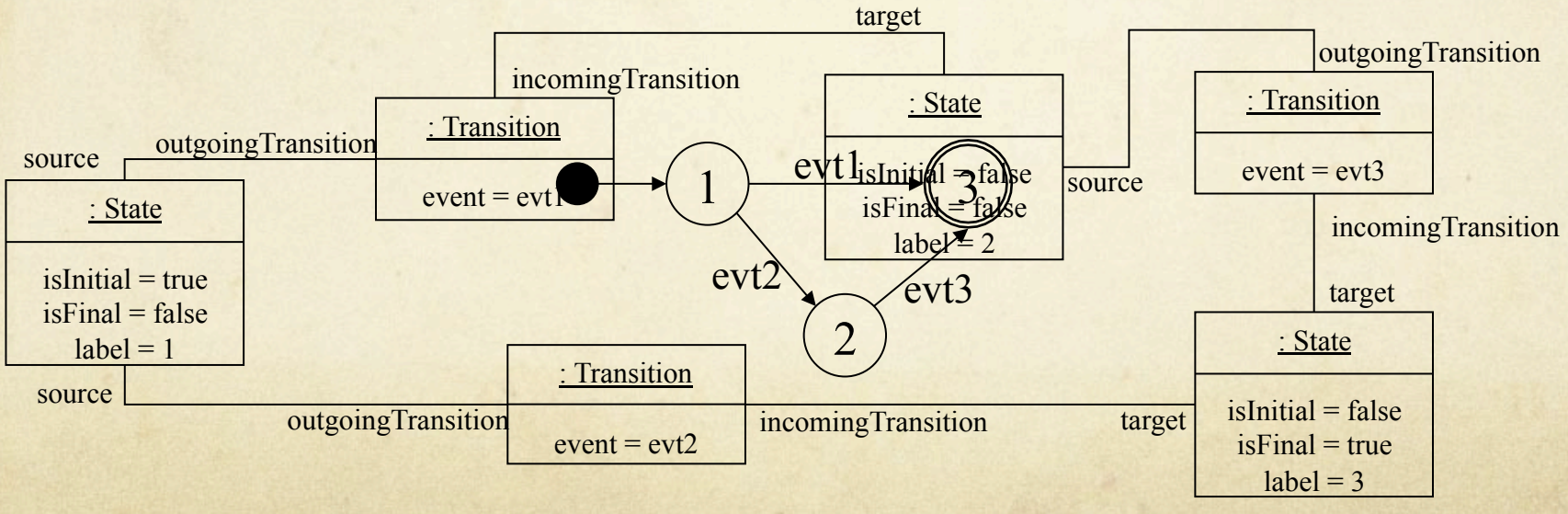
Knowledge
Domain Concepts
Static Analysis
Partitioning
Testers



Challenge: Effective Domains

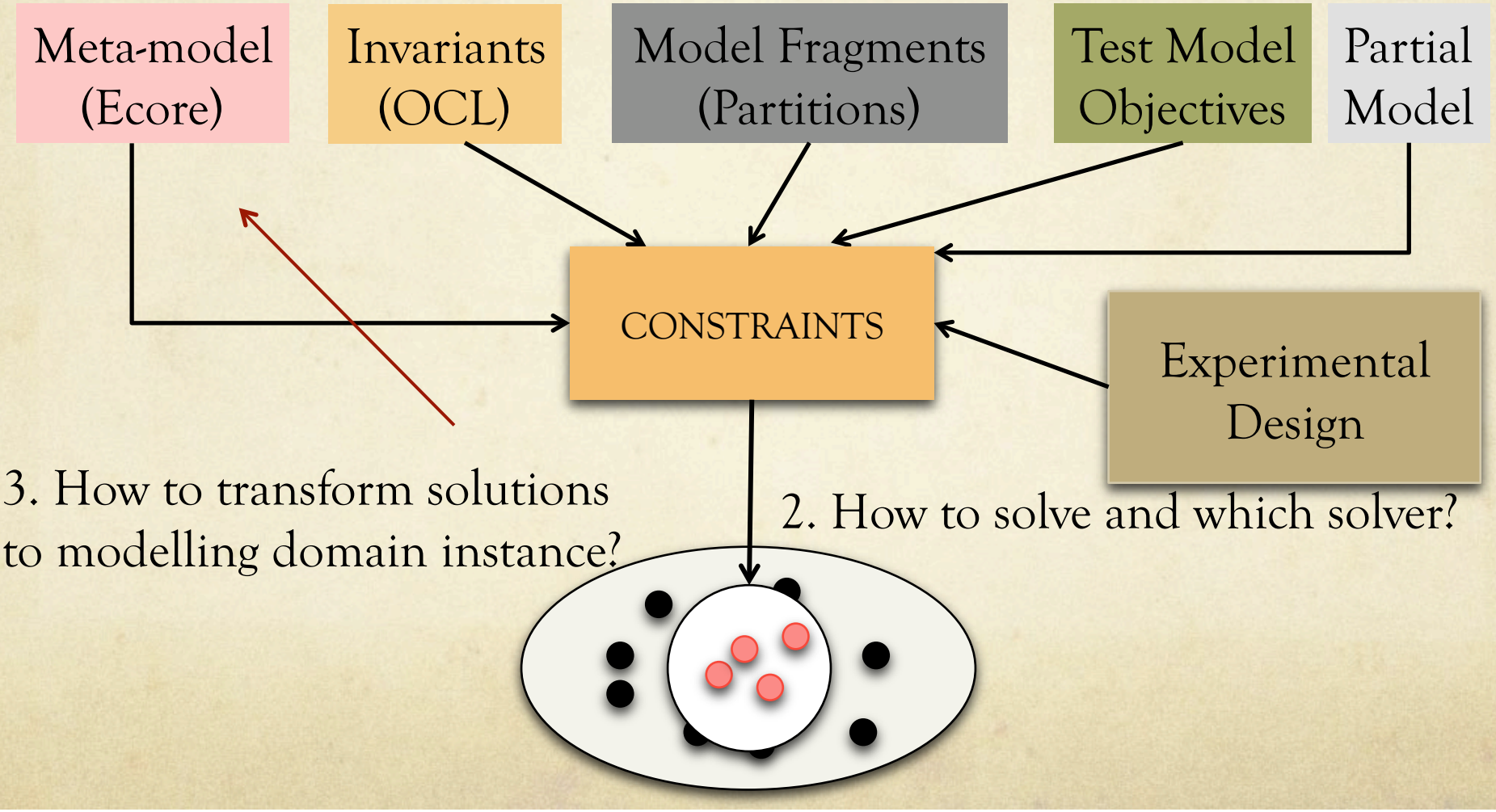
To select a model (a complex graph) from a *constrained infinite* domain



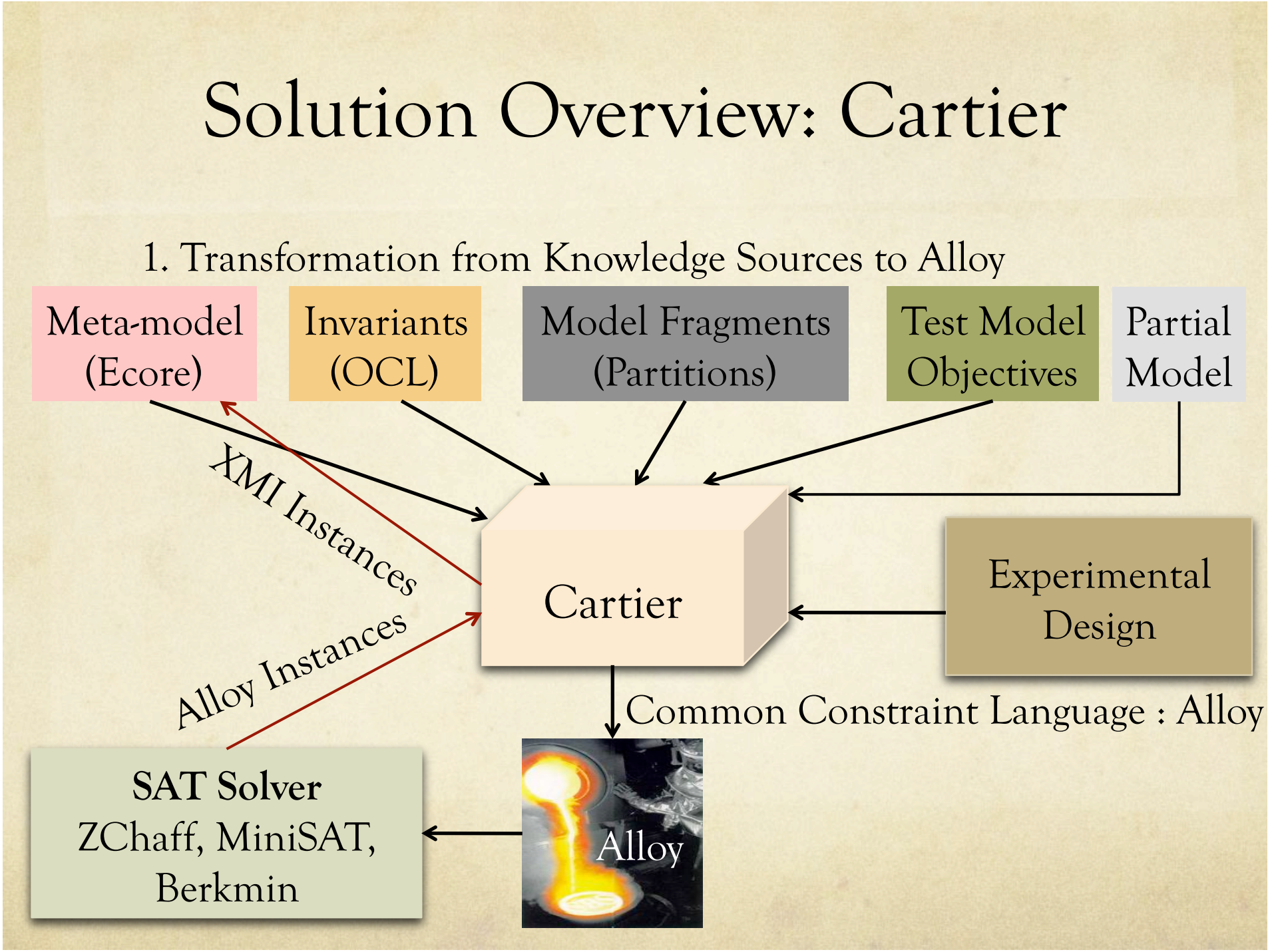


Challenge: Building Effective Modelling Domains for Testing

1. How to transform sources of knowledge for testing?



Solution Overview: Cartier

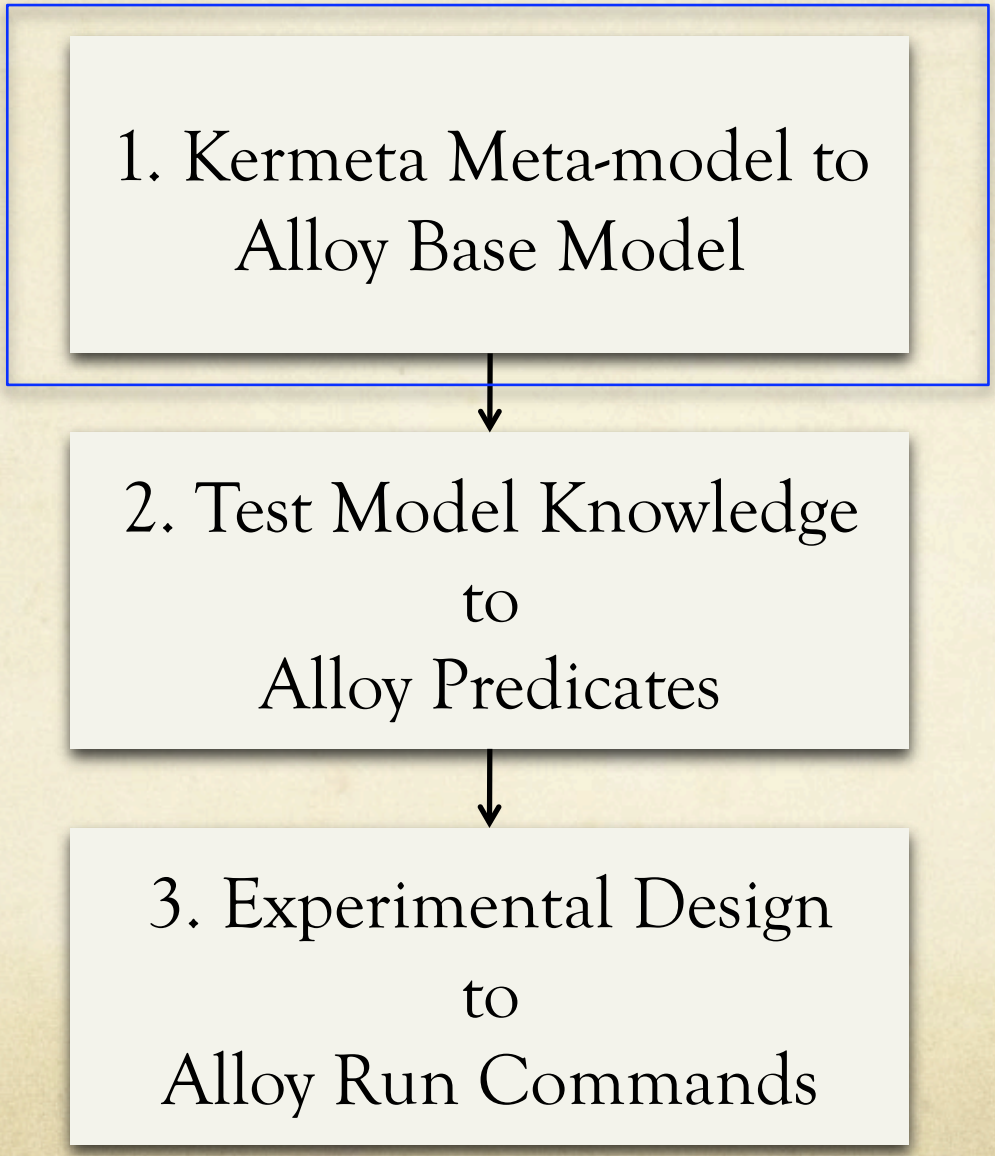


Cartier : A Guided Tour

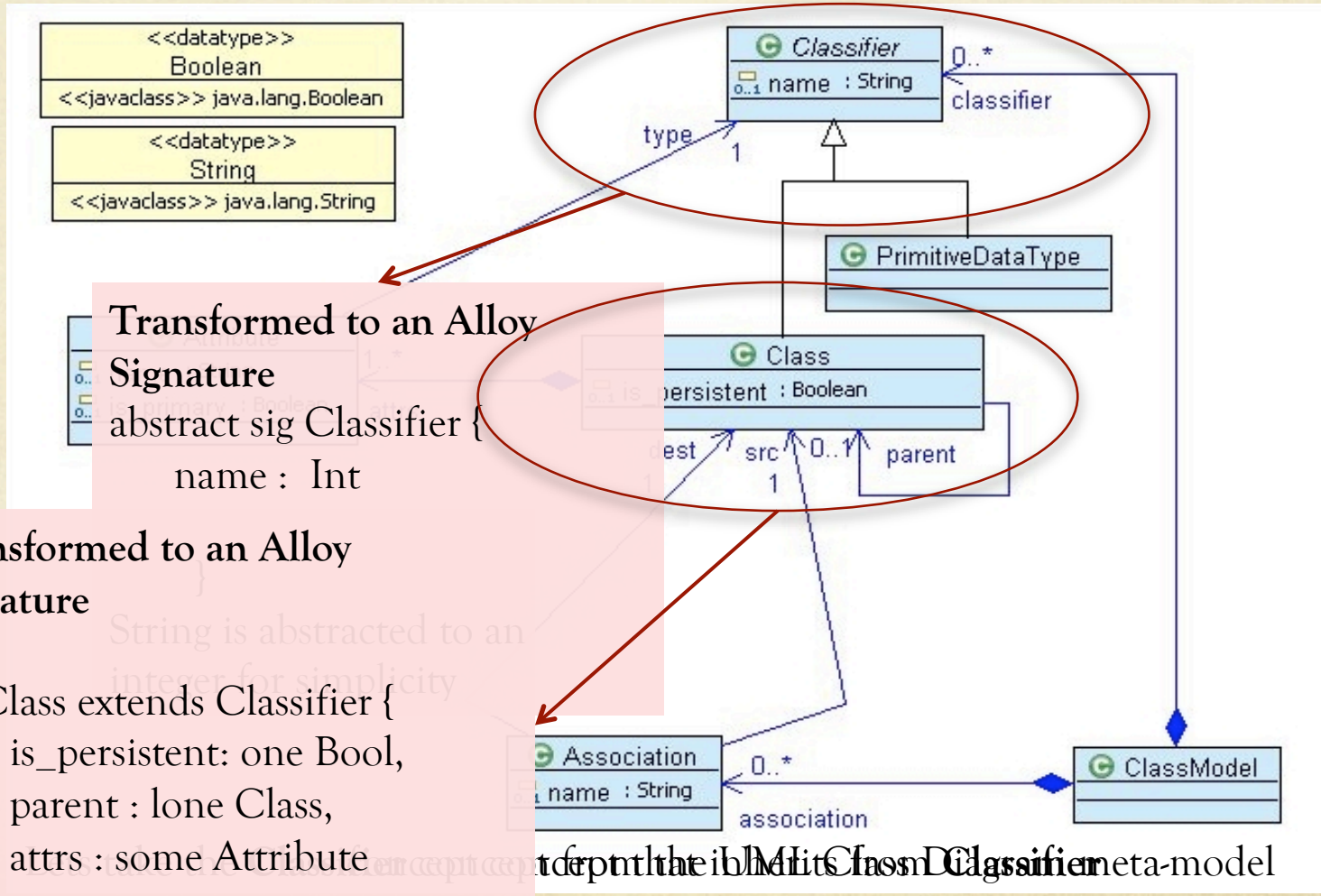
Cartier is implemented using Kermeta
(http://www.kermeta.org)

- Kermeta (Meta-modelling Kernel) is a language to represent meta-models and models of the different sources of knowledge.
- Cartier uses "aspect-oriented modelling" to transform *aspects* in source language to target language Alloy.
- So, what are the essential aspects in Kermeta that are transformed to Alloy ? ....

Cartier : A Guided Tour



Cartier: A Guided Tour (Object Persistence Case Study)



Transformed to an Alloy Signature

```
abstract sig Classifier {
        name :  Int
}
```

Transformed to an Alloy Signature

```
sig Class extends Classifier {
        is_persistent: one Bool,
        parent : lone Class,
        attrs : some Attribute
}
```

t of the UML Class Diagram meta-model

Cartier: A Guided Tour

Transforming Invariants

No Cyclic Inheritance

OCL Version:

```
context Class:

inv :

not self.allParents()->includes(self)
```

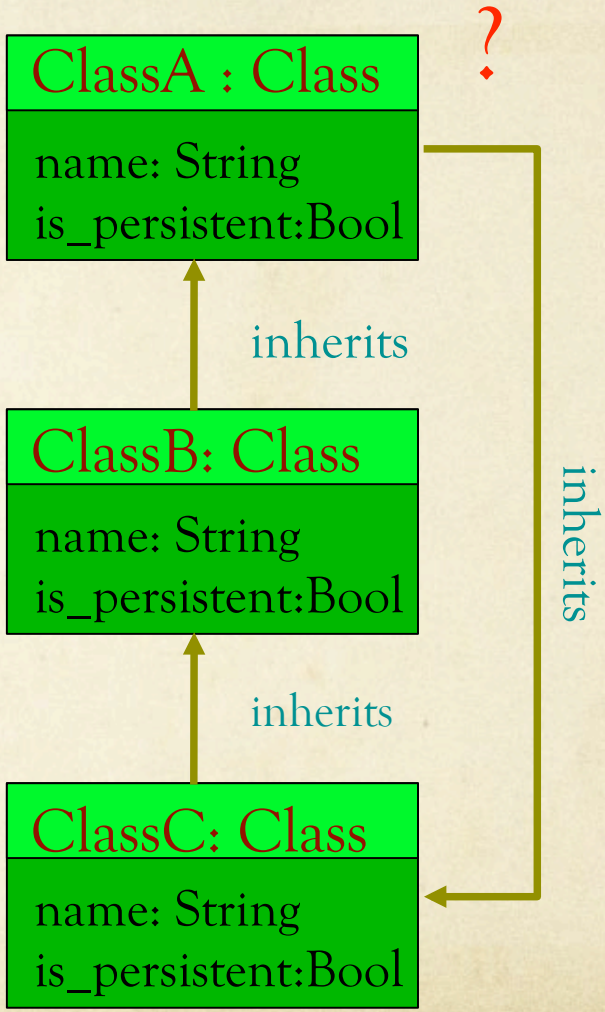
↓ to an Alloy fact (specified for a subset)

Alloy Version:

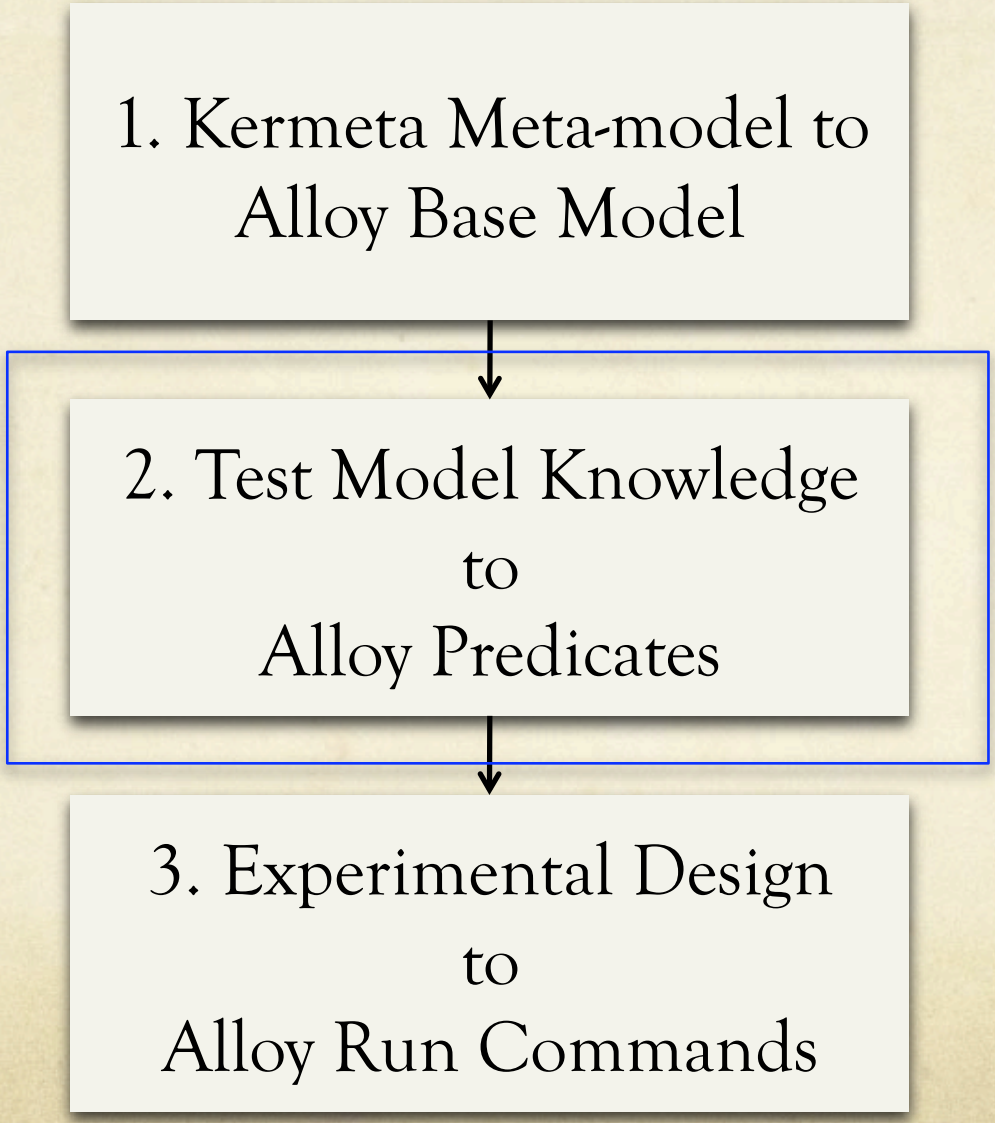
```
fact noCyclicInheritance {

    no c: Class | c in c.^parent

}
```



Cartier : A Guided Tour



Cartier : A Guided Tour

Transforming Model Fragments

Consider the model fragment (partition of the modelling domain) obtained using MMCC (Frank Fleurey et al. 2007) on the UML Class Diagram meta-model

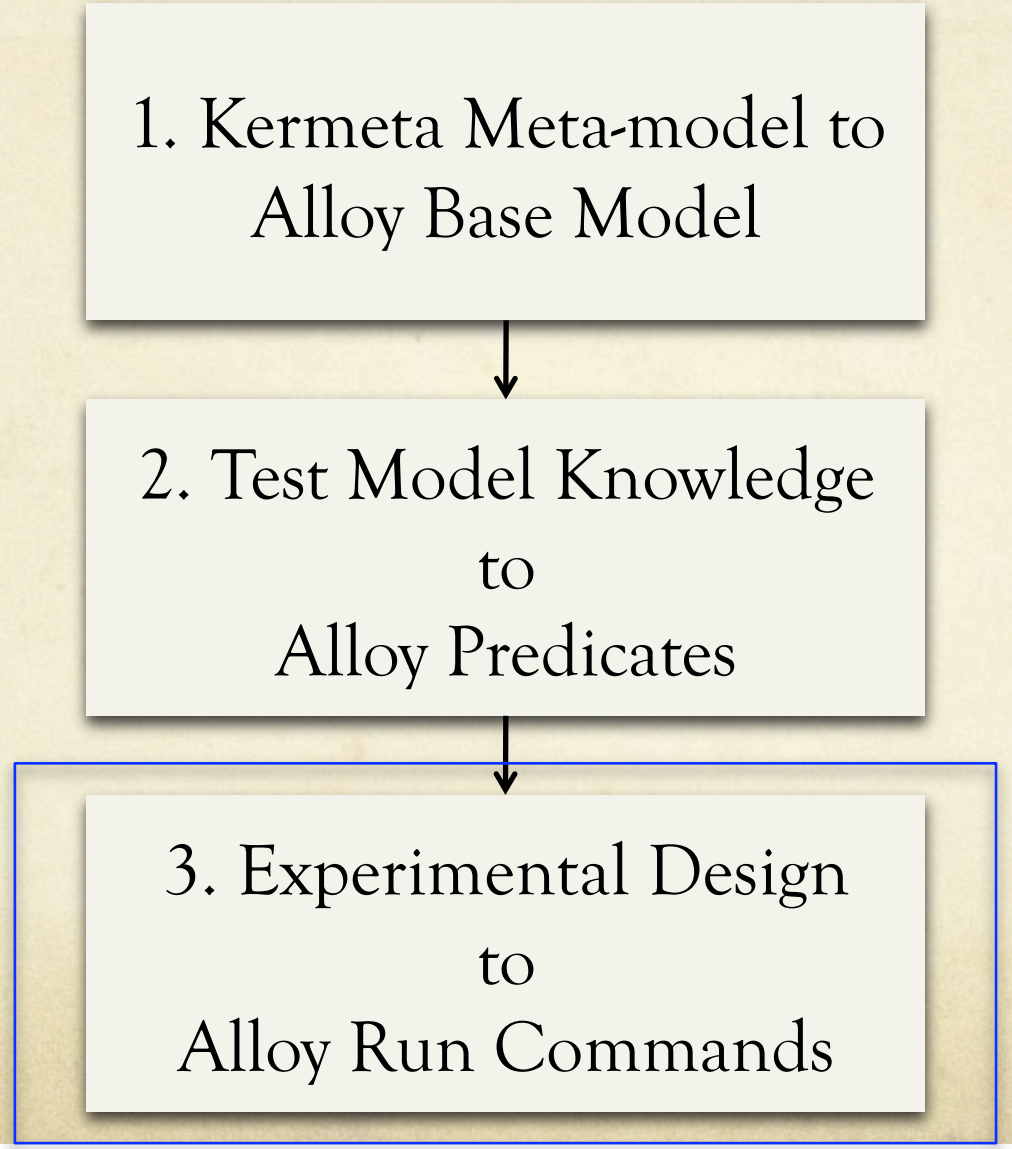
Association(source=0) and Association(source =1)

At least one Association object with no source and one Association object with one source

transformed to
an Alloy predicate

```
pred modelFragment1
{
some a1:Association,a2:Association|#a1.src=0 and #a2.src=1
}
```

Cartier : A Guided Tour



Cartier : A Guided Tour

Transforming Experimental Design

A experimental design specifies:

1) A factorial combination of number of objects of each type and different predicates.
2) Scope of Integer, String
3) Maximum size of a sequence

to Alloy Run Commands

run testModel1 for exactly 10 Class, exactly 5 Attribute, exactly 10 Association, 5 Int

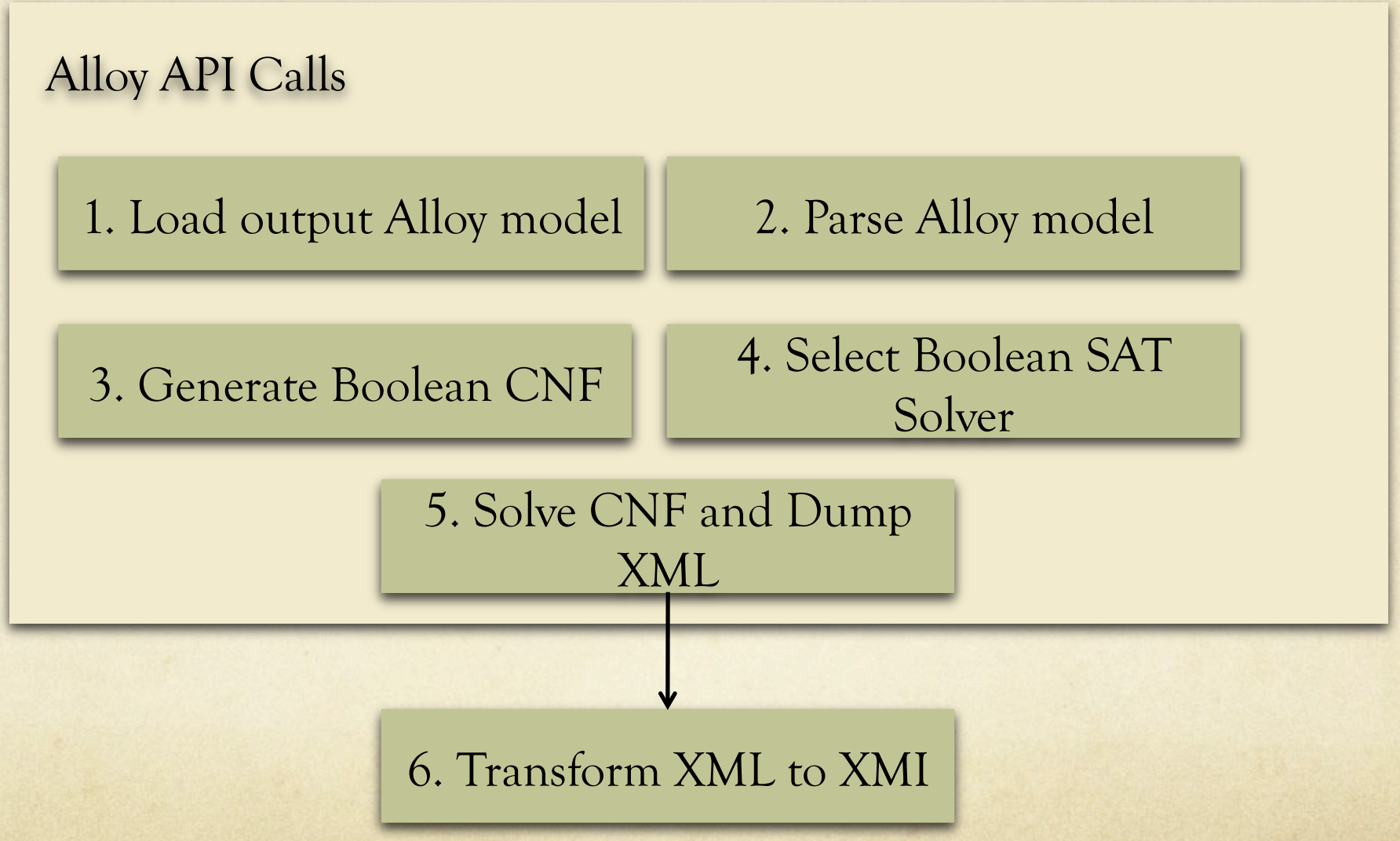
run testModel1 for exactly 10 Class, exactly 10 Attribute, exactly 10 Association, 5 Int

.

run testModel2 for exactly 5 Class, exactly 5 Attribute, exactly 10 Association, 5 Int

Cartier : A Guided Tour

Instance Generation and Output Processing



Instance Generation using Alloy

Why Alloy?

- Combines the power of first-order predicate logic with quantifiers and operators in relational logic.
- A restricted but theoretically sound meta-level declarative language.
- Operators such as transitive closure improve the expressive power of Alloy over for example Prolog

Eg: A constraint can be expressed on:

c.^parent ={c.parent, c.parent.parent, c.parent.parent.parent....}

- Operators such as the relational join (box/dot) allow you to specify powerful navigation expressions on sets and relations.

Eg:

all c: State |c.label !=5

- Automatic transformation to Boolean CNF for instance and counterexample generation for a finite scope.

Instance Generation using Alloy

Some limitations

- No support for higher-order relations.
- That means no constraint on relations of relations
- Constraints can be expressed on binary relations and their transitive closure.
- No fundamental support for String type from the MDE world. We developed one.
- No extensive support for functions on relations. Functions in Alloy are named expressions.

Instance Generation using Alloy

- For a given scope or exact number of objects Alloy creates relations which are tuples of atoms.

Example:

```
sig Association {
      name: Int,
      dest: one Class,
      src: one Class
}
```



```
Association ={
      (14,Class8,Class4),
      (11,Class8,Class9),
      (8,Class8,Class3),
      (1,Class7,Class5),
      (-10,Class6,Class9),
      (-14,Class5,Class3),
      (-16,Class5,Class6),
      (-16,Class5,Class2),
      (-1,Class4,Class1)
                  }
```

run mfAllRanges6a for 1 ClassModel, 5 int, exactly 10 Class, exactly 20 Attribute, exactly 4 PrimitiveDataType, exactly **10 Association**

Instance Generation using Alloy

```
Association ={
    (14,Class8,Class4),
    (11,Class8,Class9),
    (8,Class8,Class3),
    (1,Class7,Class5),
    (-10,Class6,Class9),
    (-14,Class5,Class3),
    (-16,Class5,Class6),
    (-16,Class5,Class2),
    (-1,Class4,Class1)
        }
```

Where do these values come from?

That satisfies facts such as…

```
/*An associations have the same name either
they
are the same association or they have
different sources*/
fact uniqueNameAssocSrc {
all a1:Association, a2:Association |
a1.name == a2.name => (a1 = a2 or a1.src !=
a2.src)
}
```

Instance Generation using Alloy

```
a[1..10] ={
      (v1,v2,v3),
      (v4,v5,v6),
      (v7,v8,v9),
      (v10,v11,v12),
      (v13,v14,v15),
      (v16,v17,v18),
      (v19,v20,v21),
      (v22,v23,v24),
      (v25,v26,v27),
            }
```

Boolean variables

KodKod Engine

Symmetry Detection & Breaking

Compact Boolean Circuit

```
((v1==v4)→(a1=a2) or (v3!=v6)) and
((v1==v7)→(a1=a3) or (v3!=v9)) and
((v1==v10)→(a1=a4) or (v3!=v12)) and
...
```

Conjunctive Normal Form

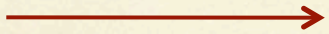
```
cnf 267395 317287
-2 3884 0
-3 3884 0
-4 3884 0
...
```

```
fact uniqueNameAssocSrc {
all a1:Association, a2:Association |
a1.name == a2.name => (a1 = a2 or a1.src !=
a2.src)
}
```

Instance Generation using Alloy

Conjunctive Normal Form

```
cnf 267395 317287
-2 3884 0
-3 3884 0
-4 3884 0
...
```



SAT Solver
ZChaff
MiniSAT
BerkMin
SAT4J

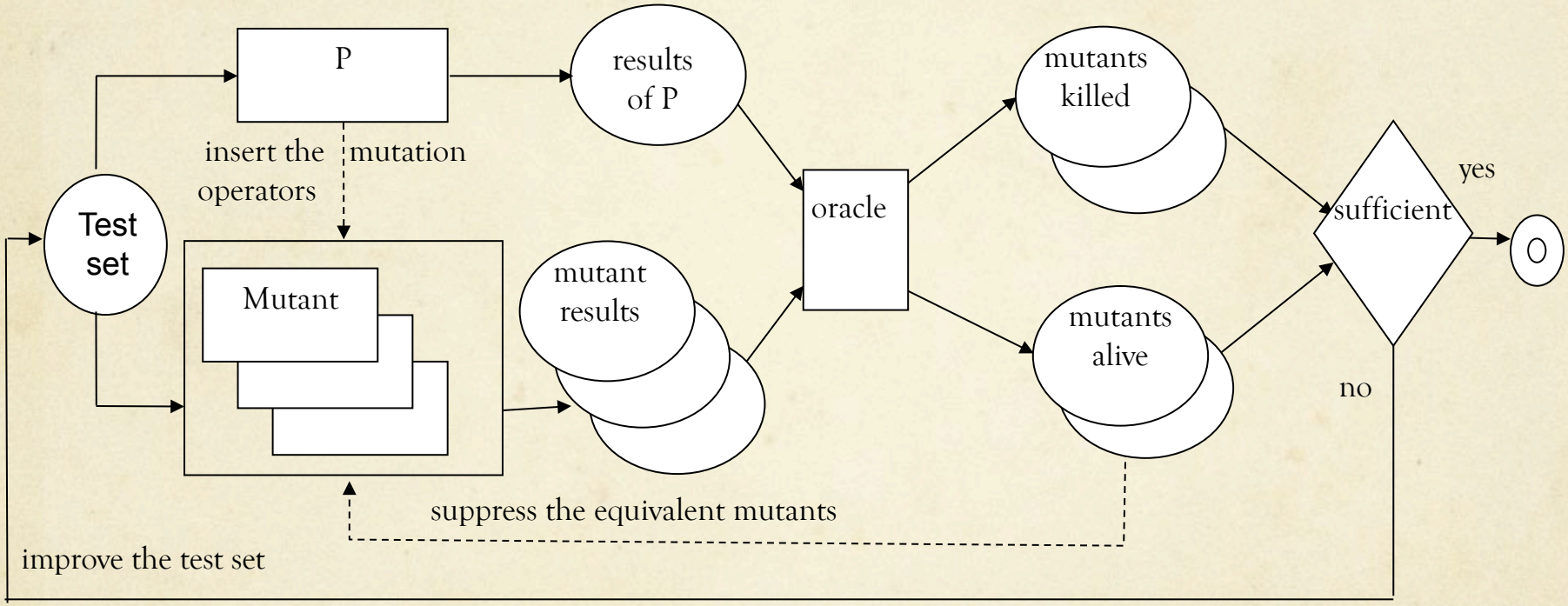
Alloy
XML Instances

Cartier

XMI Instances

Validation : Mutation Analysis

Question: Are the generated test models able to detect bugs?

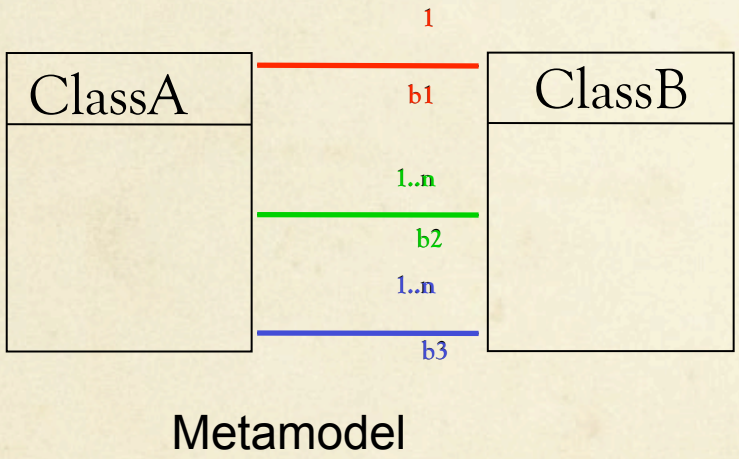


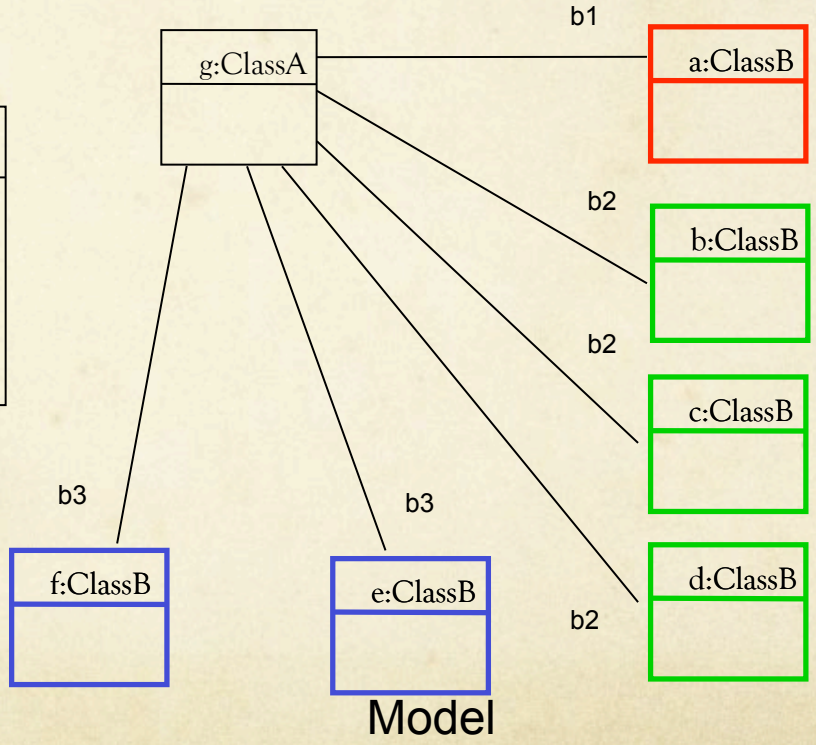
(Mottu, J.; Baudry, B. & Traon, Y. L.
Mutation Analysis Testing for Model Transformations
Proceedings of ECMDA'06 (European Conference on Model Driven Architecture), **2006**)

Validation : Mutation Analysis

A Navigation Mutation Operator Example

- Relation to the Same Class Change - RSCC





Experiments and Results

Experimental Design

Objective: To compare test generation strategies using mutation scores of sets of test models:

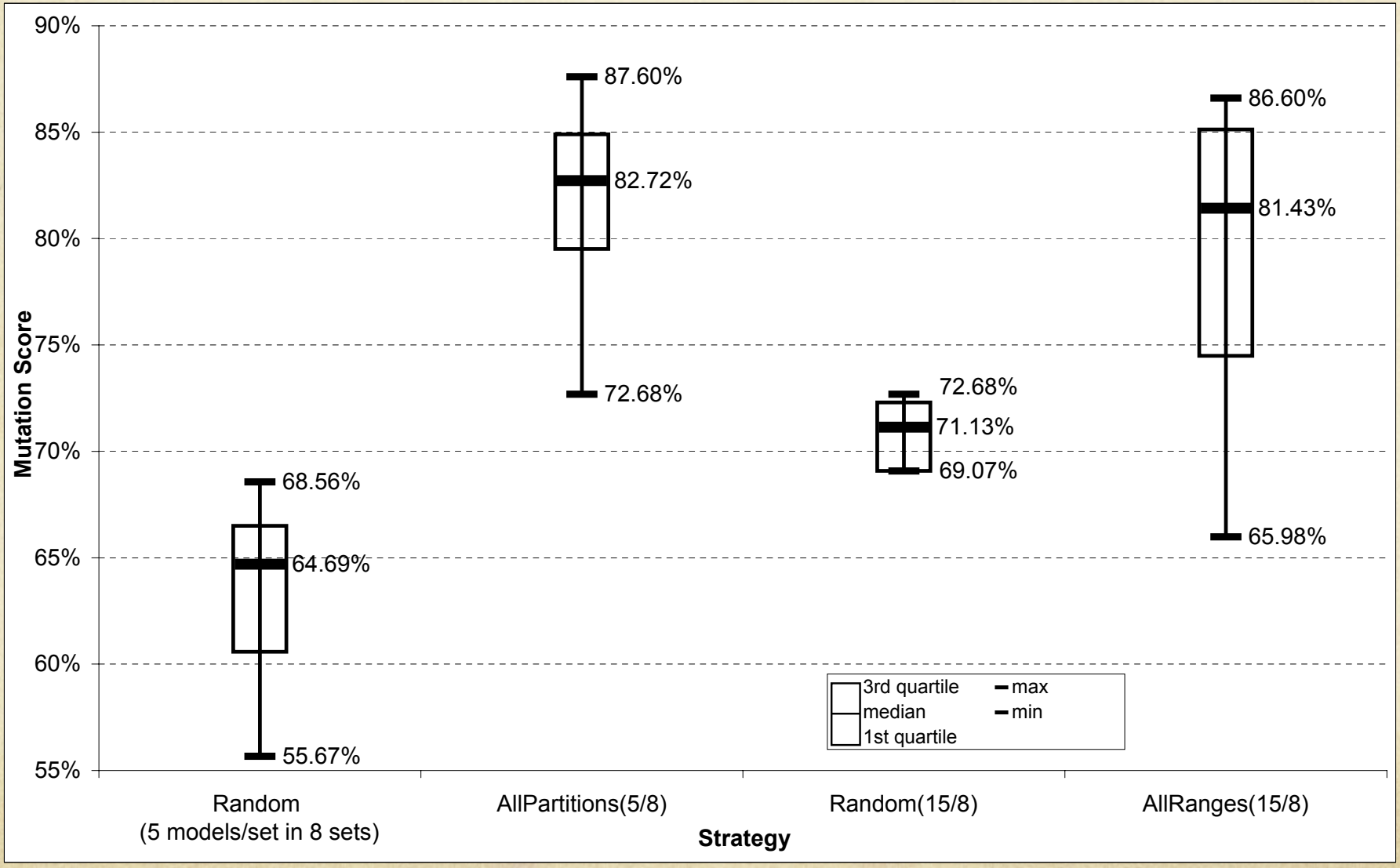
a) Using model fragments and conforming to modelling domain

b) Randomly selected in a pool of models of various sizes conforming to modelling domain.

We compare four strategies for model selection : AllRanges, Random(15 models/set), AllPartitions, Random(5 models/set)

Models Generated for AllRanges = 8 x 15 = 120
Models Generated for AllPartitions = 8 x 5 = 40
Models Generated for Random = 200
Total = 360 models

Experiments and Results



Conclusion

- We present a tool Cartier to transform a modelling domain to Alloy
- We show how we invoke the Alloy API to generate instances or models.
- We show how the effectiveness of these generated instances can be improved for testing using test model knowledge such as model fragments.
- We want to apply Cartier and Alloy for several applications in MDE such as test generation, partial model completion, and improving modelling domain specifications.