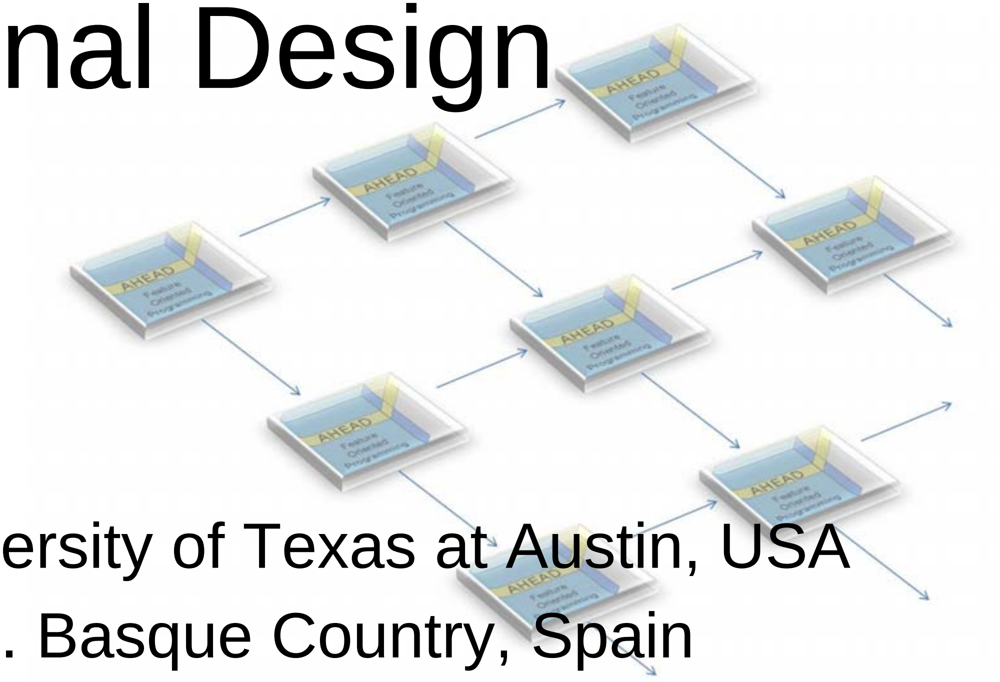
The Objects And Arrows Of Computational Design



Don Batory – University of Texas at Austin, USA
Maider Azanza – Univ. Basque Country, Spain
João Saraiva – Univ. Minho, Portugal

Introduction

- Future of software design and development is automation
  - mechanize repetitive tasks
  - free programmers for more creative activities

- Entering the age of **Computational Design**
  - program design and synthesis is a computation

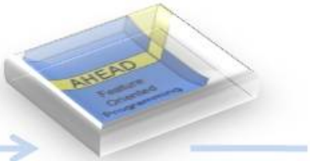
- **Design**: steps to take to create an artifact
  - metaprogram

- **Synthesis**: evaluate steps to produce the artifact
  - metaprogram execution

Forefront of Automated Development

- **Model Driven Engineering (MDE)**
  - high-level models define applications
  - transformed into lower-level models
  - general-purpose approach
- **Software Product Lines (SPL)**
  - domain-specific approach
  - we know the problems, solutions of a domain
  - we want to automate the construction of these programs
- Both complement each other
  - strength of MDE is weakness of SPLs, and vice versa
  - not disjoint, but I will present their strengths as such

My Background



- In prior lifetime, I was a database researcher
  - program generation was **relational query optimization (RQO)**
  - query evaluation programs were relational algebra expressions
  - designs of such programs could be optimized

- Took me years to recognize the significance of RQO
  - compositional paradigm, computational design

- Fundamentally shaped my view of automated software development
  - you'll see impact…

My Work

- SPLs with emphasis on language and tool support

- I needed a simple language to express program design and synthesis as a computation
    - **modern algebra fits the bill**

- Programs are **structures**
    - tools transform, manipulate, analyze

    - OO structures are methods, classes, packages
    - compilers transform source structures
    - refactoring tools transform source structures
    - meta-models of MDE define allowable structures of instances; transformations map instances for analysis or synthesis

So What??

- Well … mathematics is the science of structure and the manipulation of structure

- Once I recognized that transformations are fundamental to software development

    - I was on the road to mathematical thinking

    - basic ideas are relevant

    - once I understood the connection...

My Work

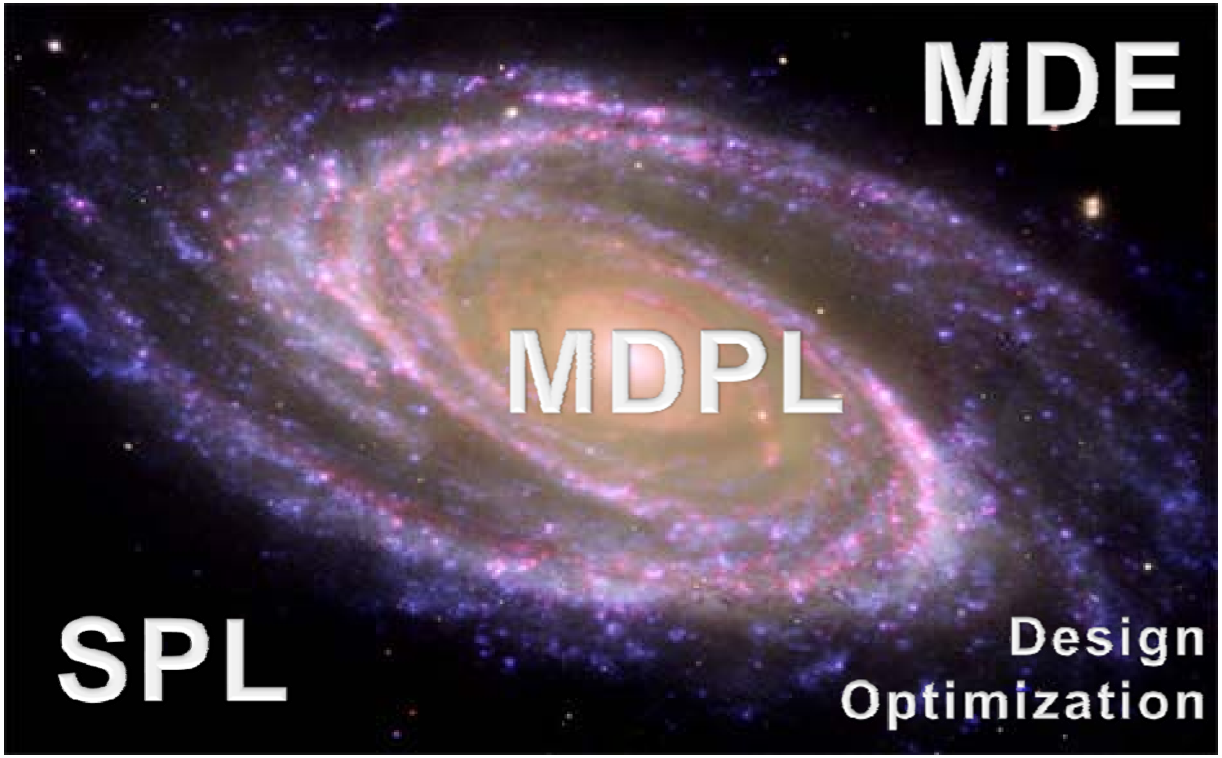
- I use mathematics as an informal design methodology and language to explain computational designs
  - **not a formalism!**

- This is a modeling talk aimed at practitioners
  - no special mathematical background

- Core ideas inspired from category theory
  - theory of mathematical structures
  - result of a domain analysis of geometry, topology, algebra…
  - basic concepts in CT are core ideas in MDE, SPL

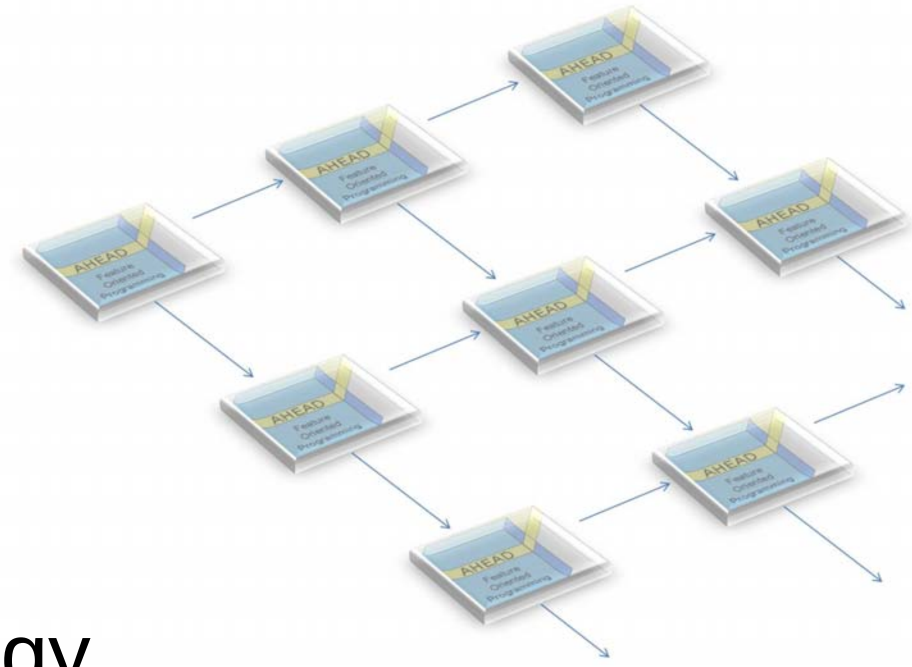
This Talk

- Expose underlying principles of MDE and SPL
  - not category theory – functors, pushouts, products of categories, …
- Series of mini-tutorials (10 minutes apiece)



categories
on an
industrial
scale

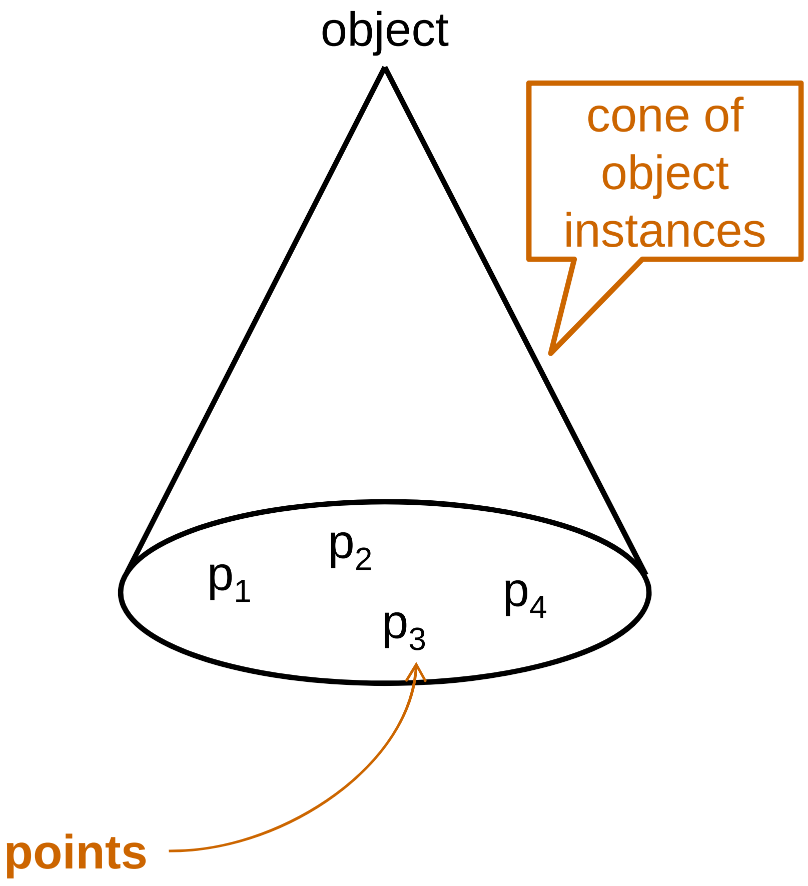
Part 1: Categories in MDE

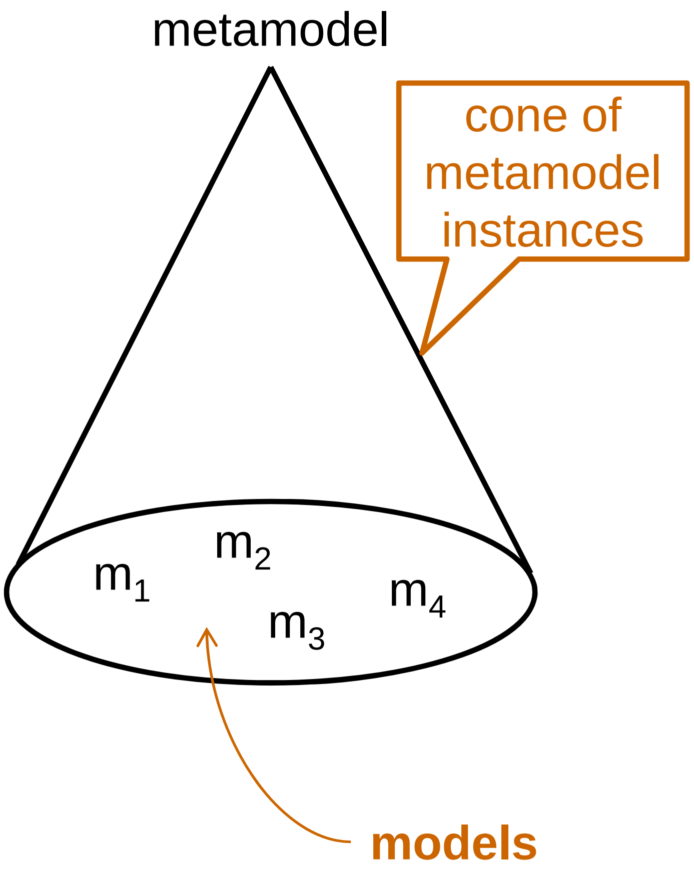

AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming

let’s start with some unfortunate terminology…

Objects

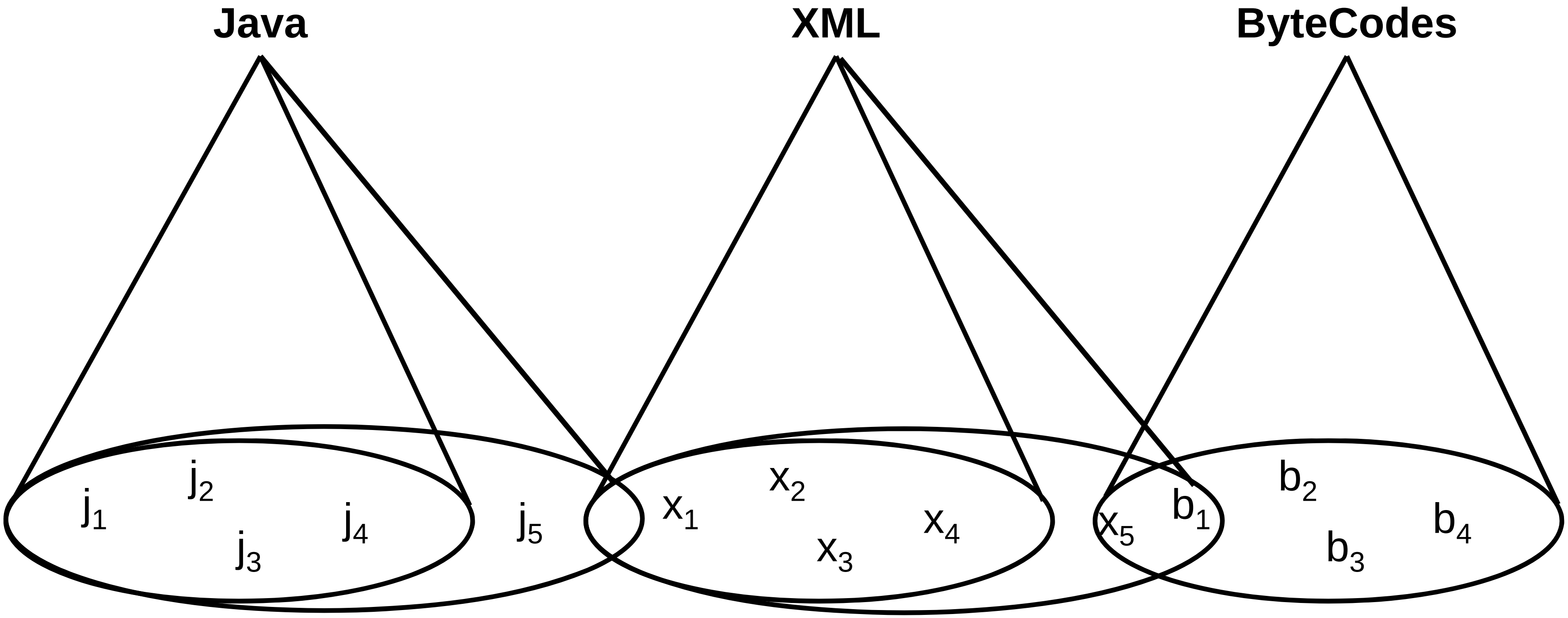
- An **object** is a domain of **points** (no standard term)
- **Metamodel** defines a domain of **models**





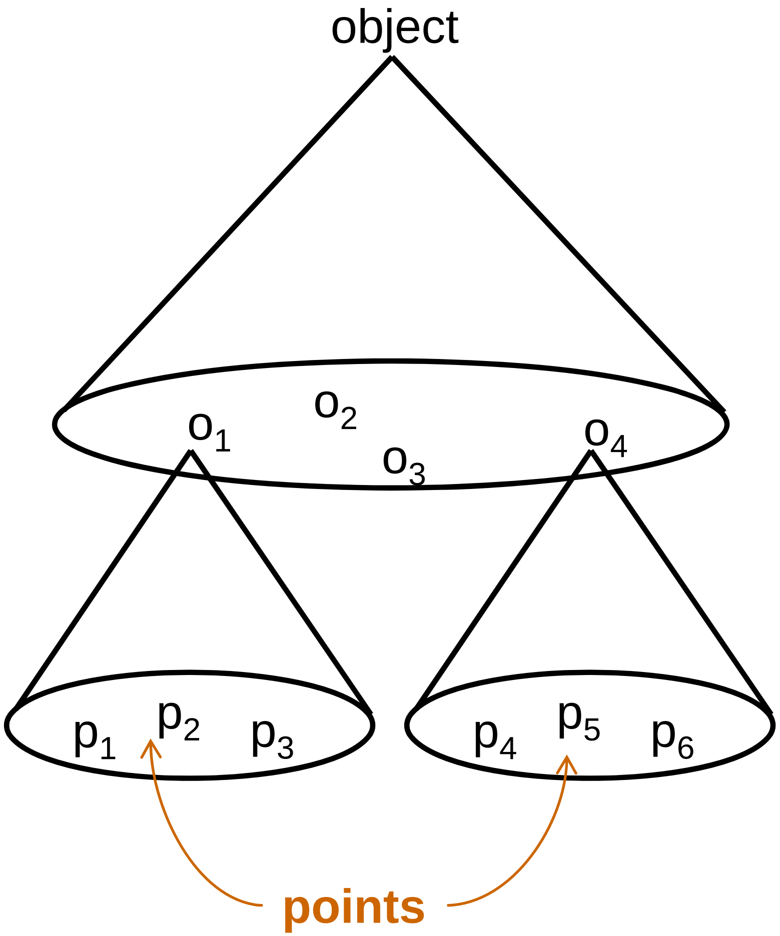
Examples

- MDE focuses on UML metamodels and their instances
- Ideas of objects & instances also apply to non-MDE artifacts
  - **technical spaces** of Bezivin, et al.

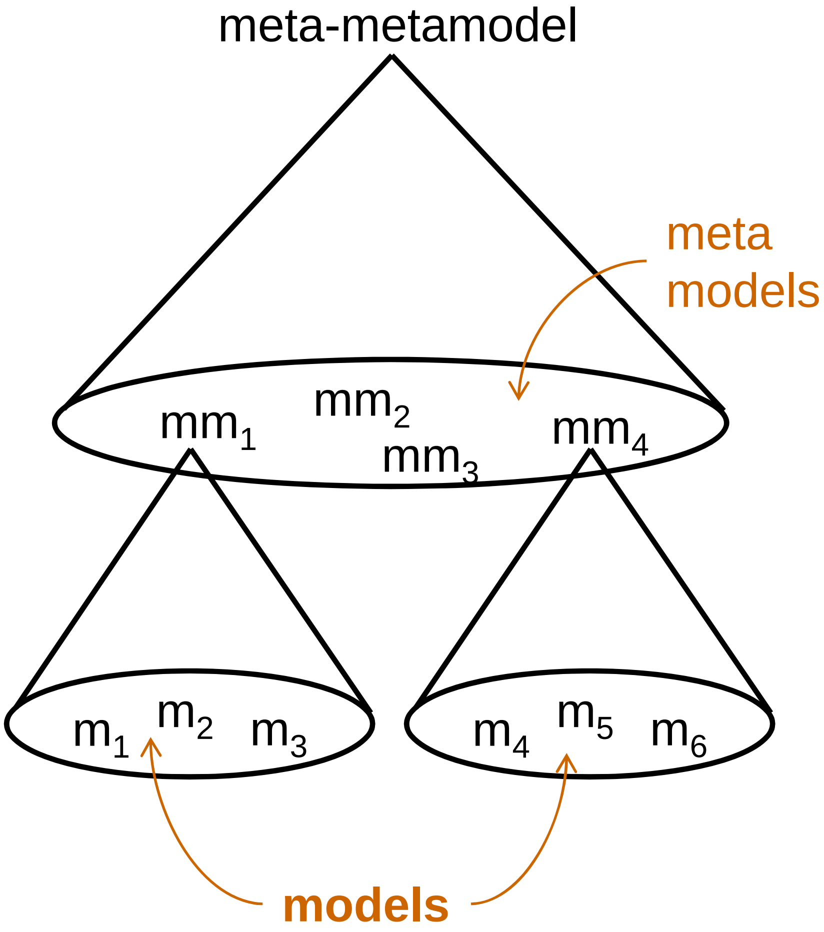


Recursion

- A point can be an object

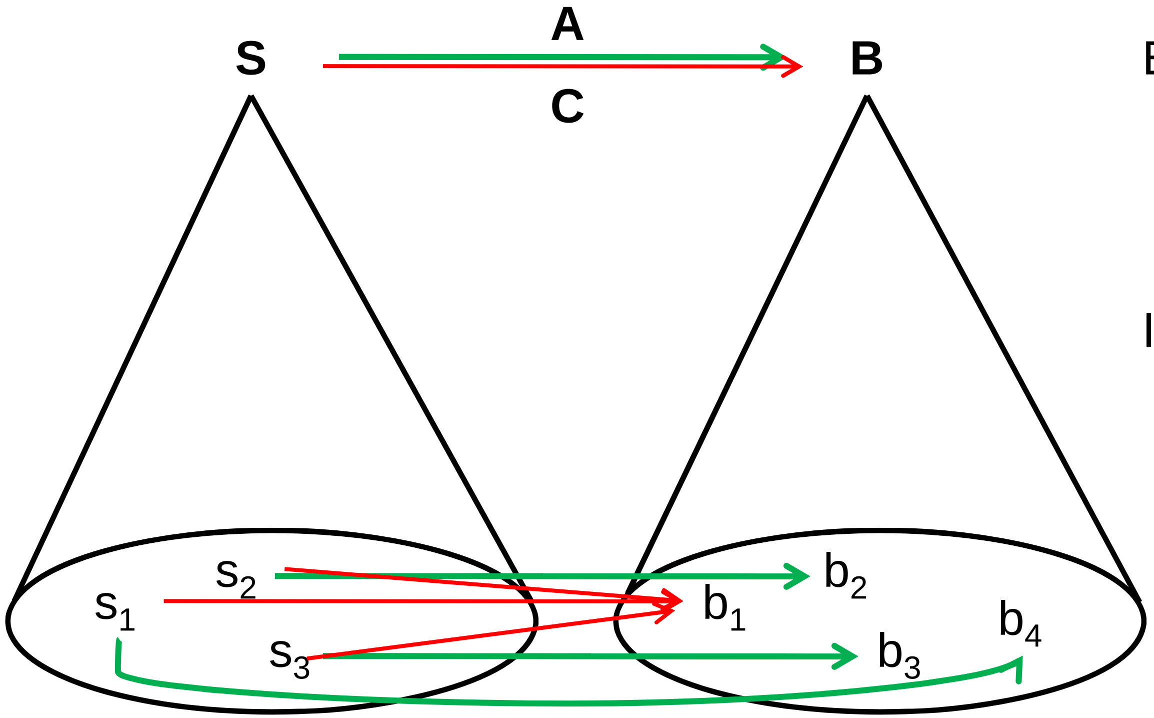


- Standard MOF architecture



Arrow

- Is **map** or **function** or **transformation** or **morphism** between objects (all names are used)
    - implementation is unspecified



External Diagram

Internal Diagram

My Terminology (for this talk)

- **Arrow** – denotes a map
- **Transformation** – an MDE implementation of an arrow
  - ATL, RubyTL, GReAT, QVT …
- **Tool** – is a non-MDE implementation of an arrow
  - standard tools of software engineers

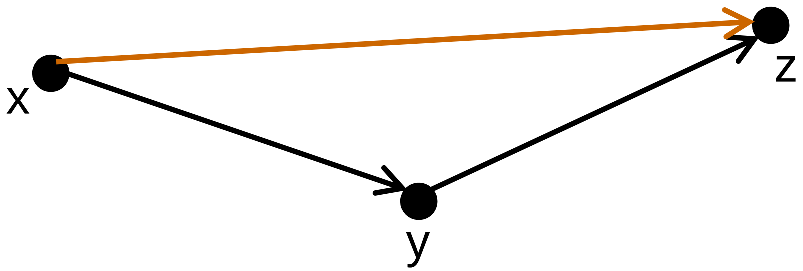
External Diagrams

$$\begin{tikzcd}
\text{MSC} \arrow[loop above] \arrow[r, "\text{M2MX}"] & \text{SC} \arrow[loop above] \arrow[r, "\text{M2TX}"] & \text{Java} \arrow[loop above] \arrow[r, "\text{javac}"] & \text{ByteCode} \arrow[loop above]
\end{tikzcd}$$

- **Category** – a collection of objects and arrows
  - above is a category of 4 objects, 3 non-identity arrows
  - categories satisfy 3 simple properties…

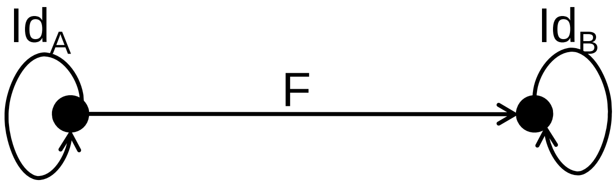
Properties of Categories

- Arrows are composable



- Composition is associative: $A \bullet (B \bullet C) = (A \bullet B) \bullet C$

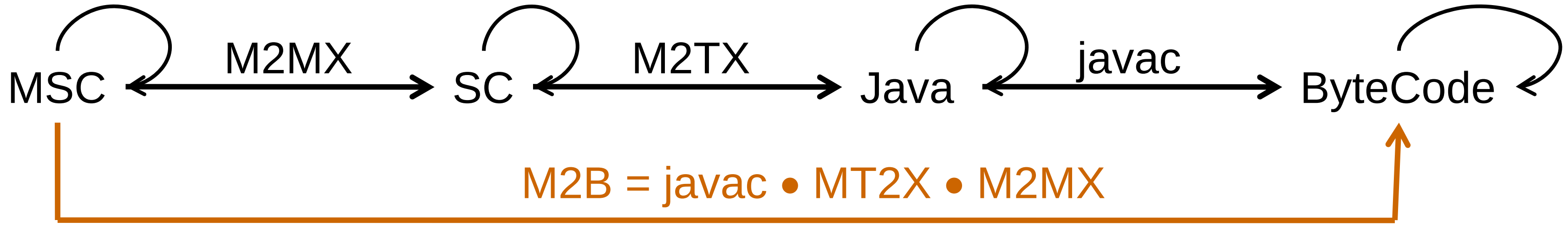
- Identities



$F \bullet Id_B = F$

$Id_A \bullet F = F$

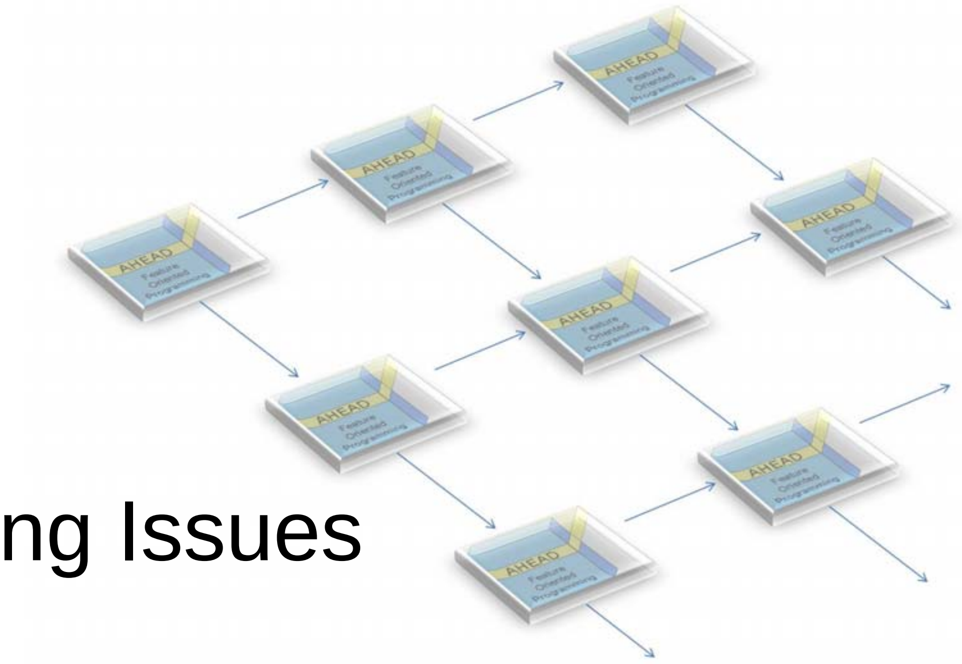
Support for These Abstractions



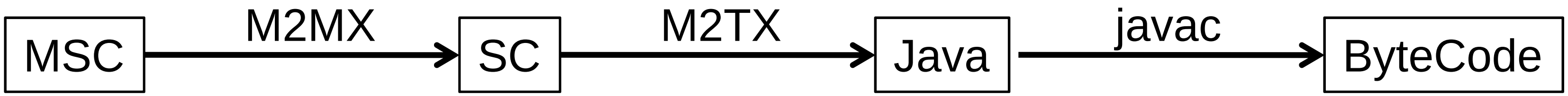
**tool chains,
makefiles,
metaprogram**

- Treat arrows and objects uniformly
  - hide their implementation technologies
  - GROVE, UniTI, etc.
  - lack of support obscures fundamental relationships
  - remove artificial complexity, expose essential complexity

Notation and Modeling Issues



External Diagrams in MDE



MSC

M2MX

SC

M2TX

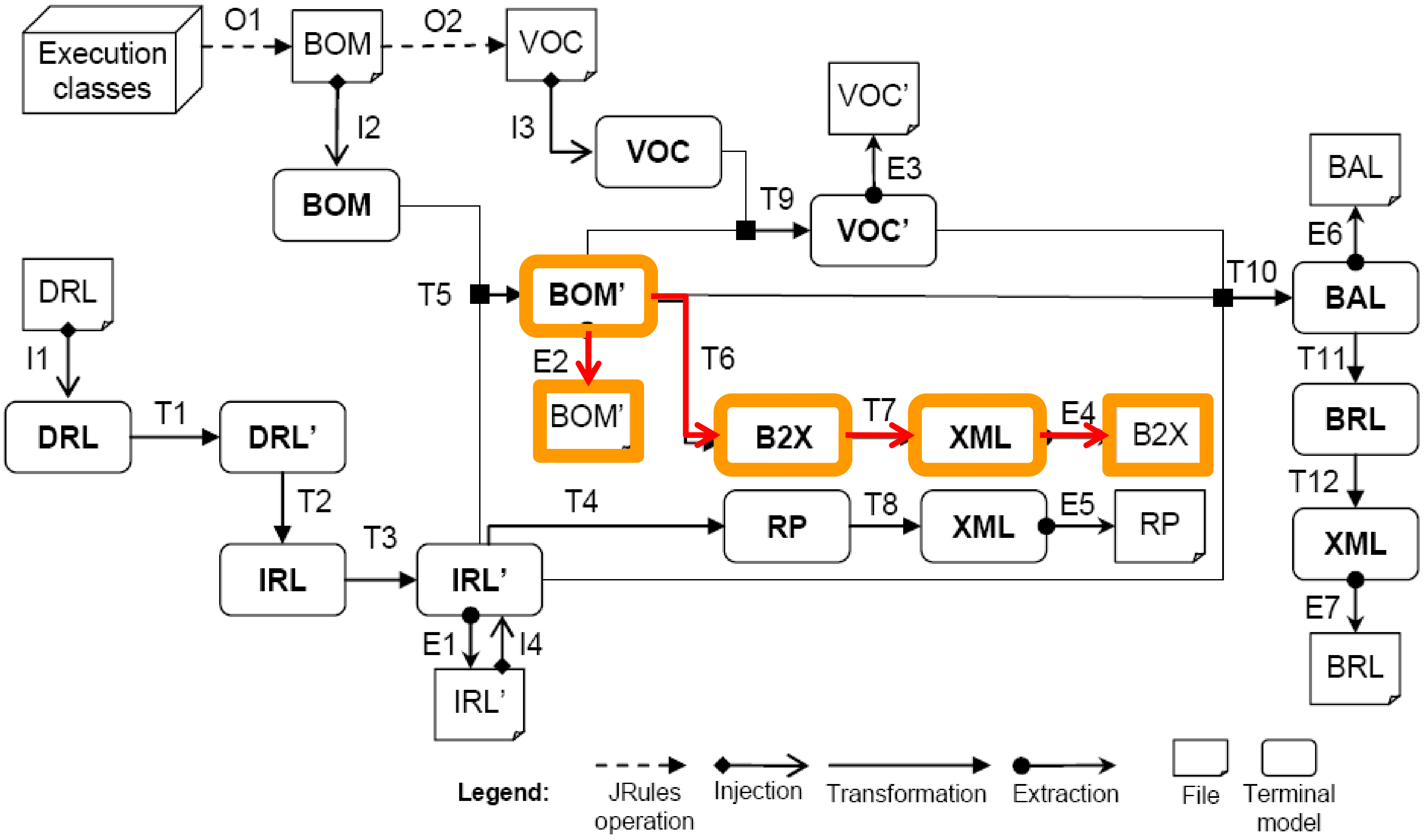
Java

javac

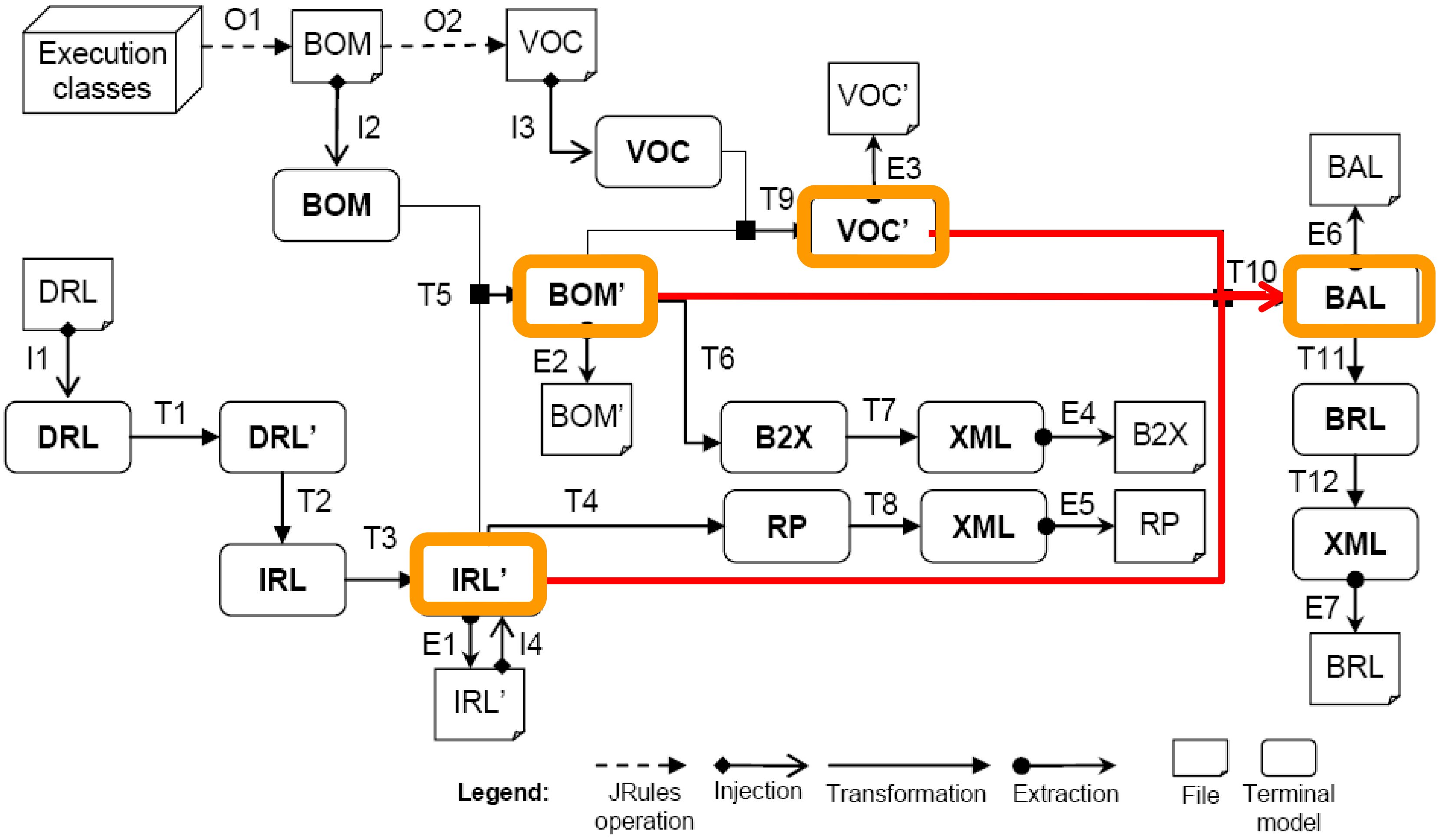
ByteCode

- No standard names for such diagrams in MDE
  - drawn differently (sans identity arrows)
  - **Toolchain diagrams** (MIC)
  - **MegaModels** (ATL)

External Diagrams in MDE



External Diagrams in MDE



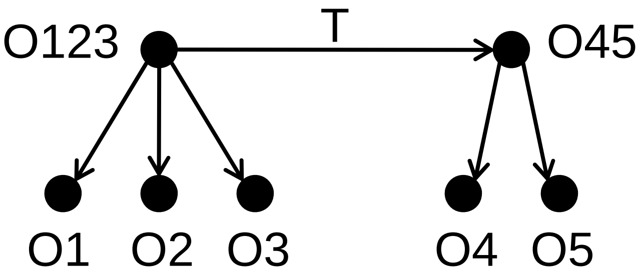
Arrows with Multiple Inputs, Outputs

- Arrows 1 input object to 1 output object

  what about T: O1, O2, O3 → O4, O5 ?
  occurs in model weaving …

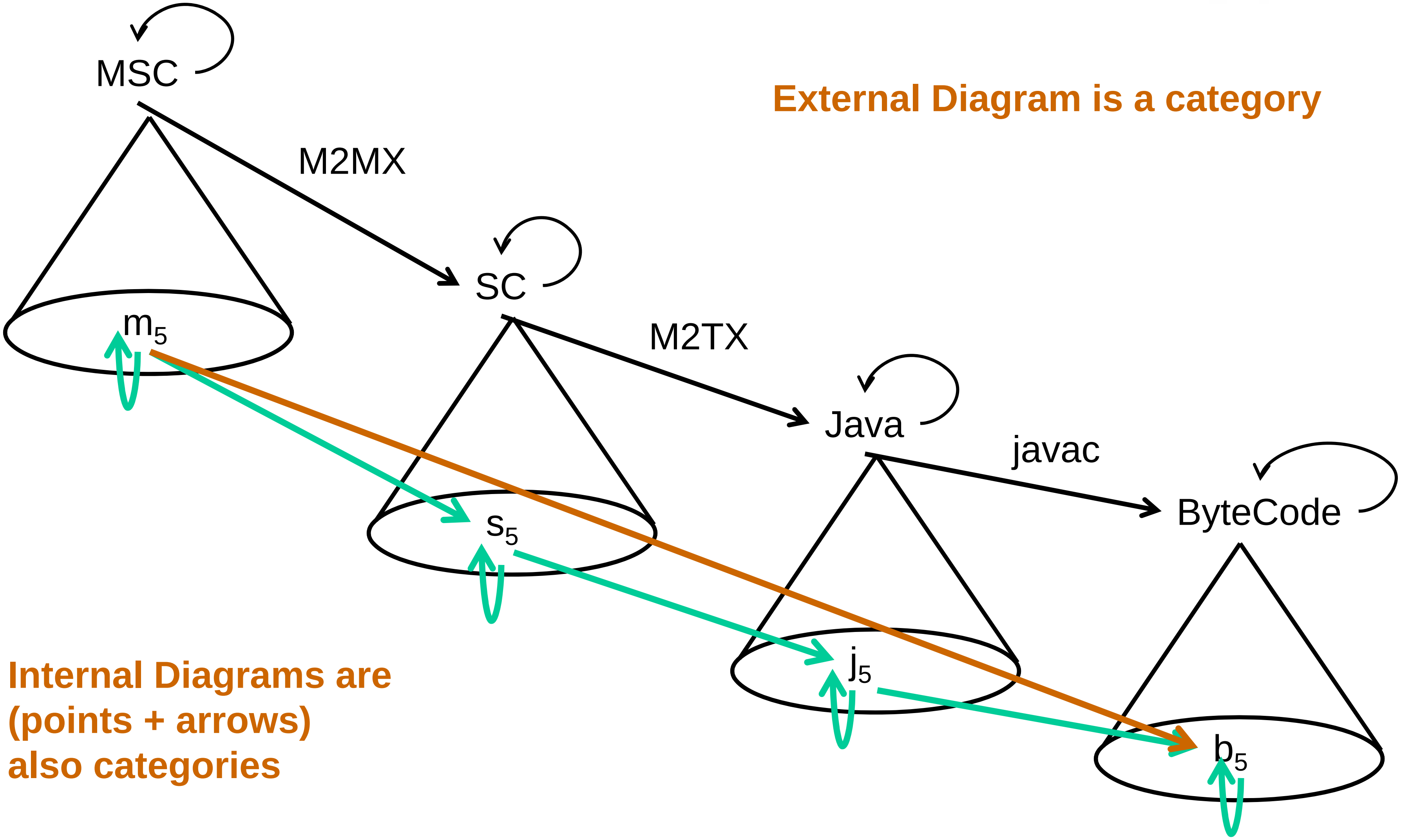
- Ans: create tuple of objects, which is itself an object

  O123 = [O1, O2, O3]
  O45 = [O4, O5]



projection arrows

Internal Diagrams



External Diagram is a category

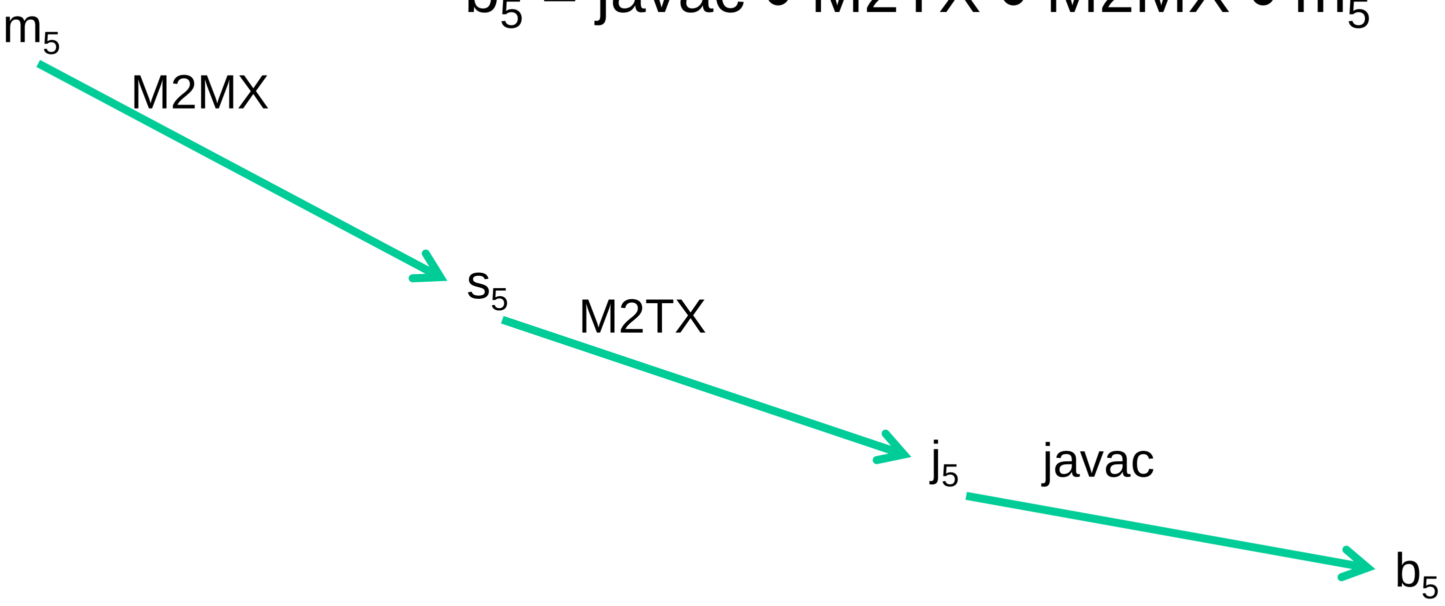
Internal Diagrams are (points + arrows) also categories

degenerate or trivial category: point is a domain with a single program

Computational Design

- Design of an artifact is an expression
  - synthesis is expression evaluation
  - RQO paradigm

$$b_5 = javac \bullet M2TX \bullet M2MX \bullet m_5$$



m_5

M2MX

s_5

M2TX

j_5

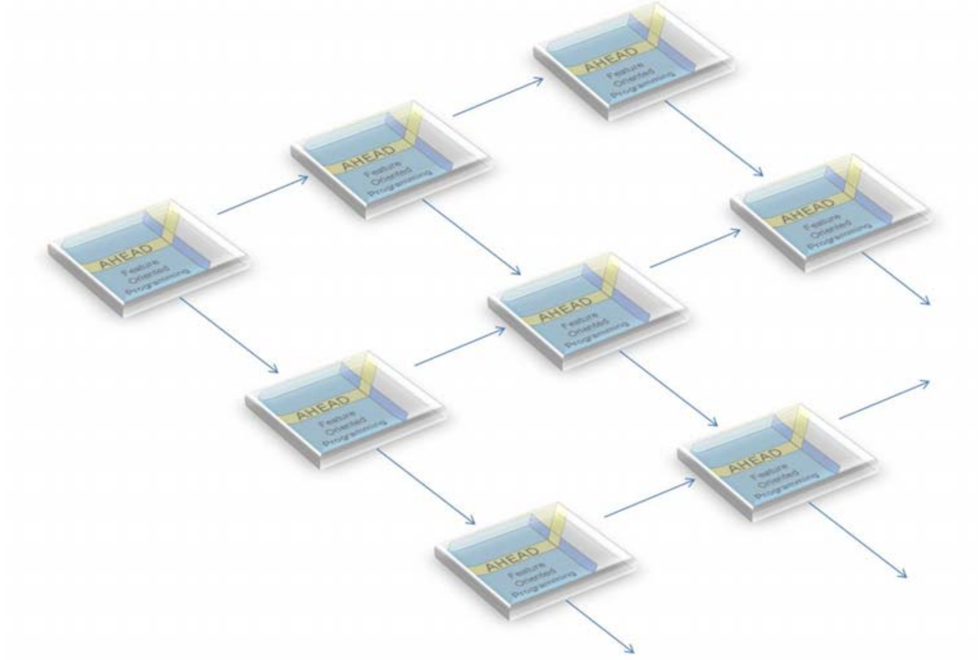
javac

b_5

Recap

- Categories lie at the heart of MDE
    - found at all levels in an MDE architecture
    - categories on an industrial scale
- Informally, categories provide a compact set of ideas to express relationships that arise among objects in MDE
    - language and terminology for MDE computational designs
    - can use CT more formally (e.g., Meseguer, Ehrig, Täntzer, Diskin) …
- Now let’s look for categories in Product Lines

Part 2: Categories in SPLs



SPL Overview

- SPL is a set of similar programs

- Programs are defined by **features**
  - increment in program functionality that customers use to distinguish one program from another

- Programs are related by features
  - program P is derived from program G by adding feature F
  - feature is a function:

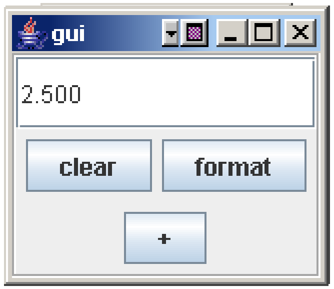
$$P = F(G)$$

4-Program Product Line

```
class calculator {
    float result;
    void add( float x ) { result=+x; }
    void sub( float x ) { result=-x; }
}

class gui {
   JButton format = new JButton(“format”);
   JButton add    = new JButton(“+”);
   JButton sub    = new JButton(“-”);

   void initGui() {
     ContentPane.add( format );
     ContentPane.add( add );
     ContentPane.add( sub );
   }

   void initListeners() {

     add.addActionListener(...);
     sub.addActionListener(...);
   }

   void formatResultString() {...}
}
```

new methods

new fields



2.500

new fields

extend existing methods

extend existing methods

$\text{format} \bullet \text{sub} \bullet \text{base} = p_4$

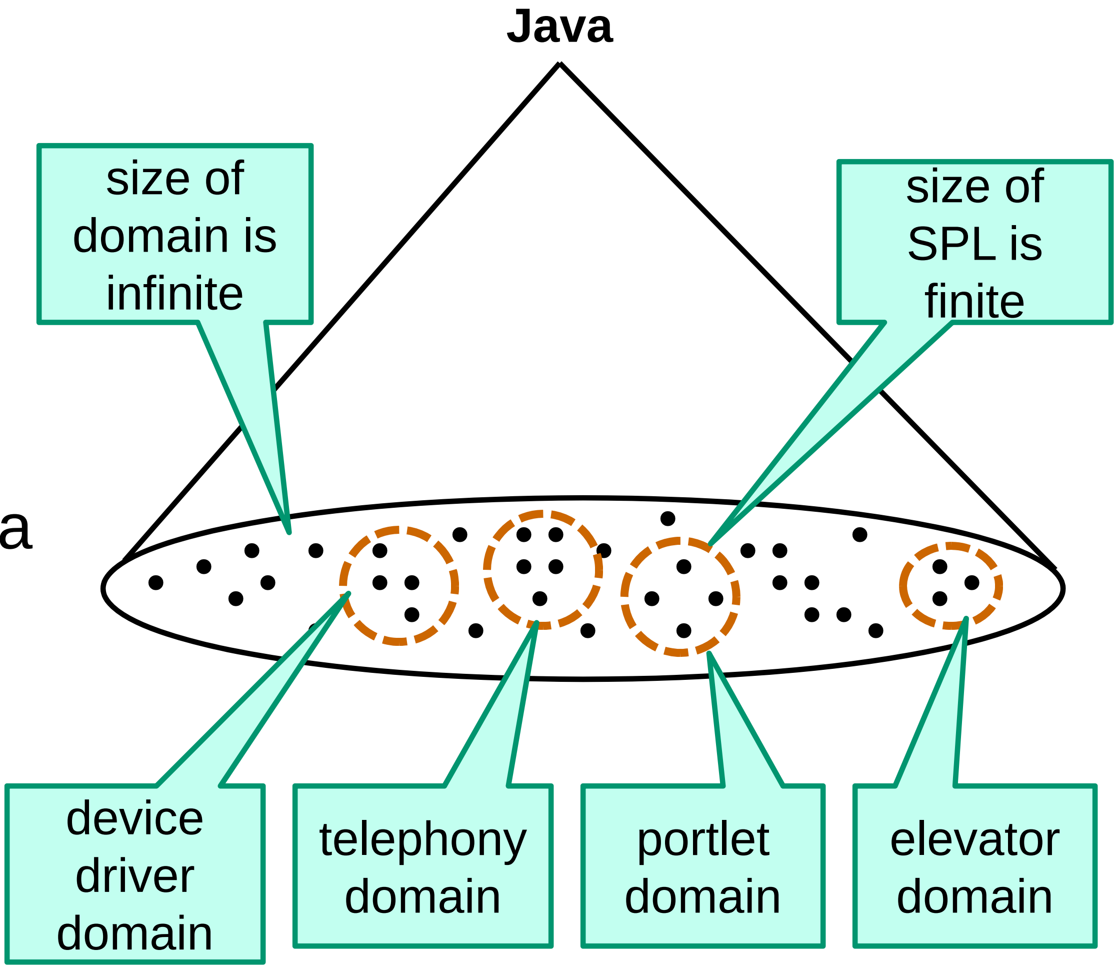
new methods

Ideas Scale...

- 1986 database systems 80K LOC
- 1989 network protocols
- 1993 data structures
- 1994 avionics
- 1997 extensible Java preprocessors 40K LOC
- 1998 radio ergonomics
- 2000 program verification tools
- 2002 fire support simulators
- 2003 AHEAD tool suite 250K LOC
- 2004 robotics controllers
- 2006 web portlets

Perspective on Product Lines

- SPL is a set of similar programs
- Is a ***miniscule*** subset of a domain
- Infinite set of SPLs in a domain



Java

size of domain is infinite

size of SPL is finite

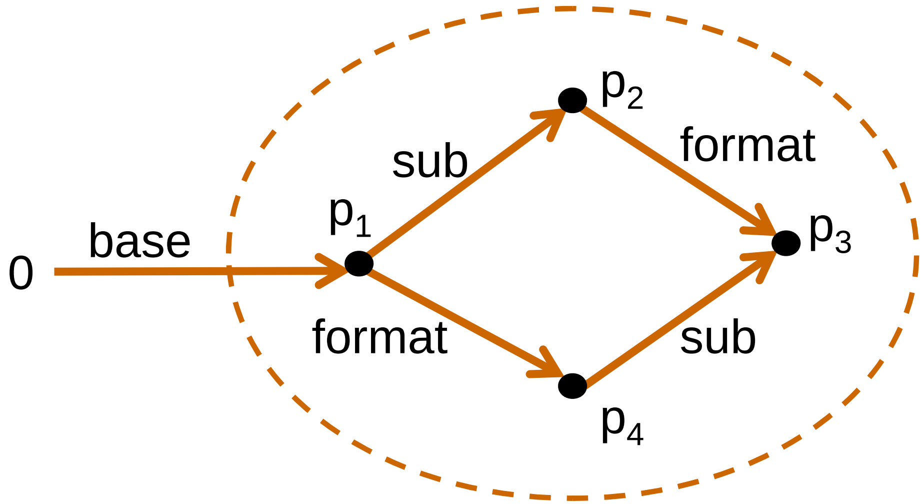
device driver domain

telephony domain

portlet domain

elevator domain

Perspective on Product Lines

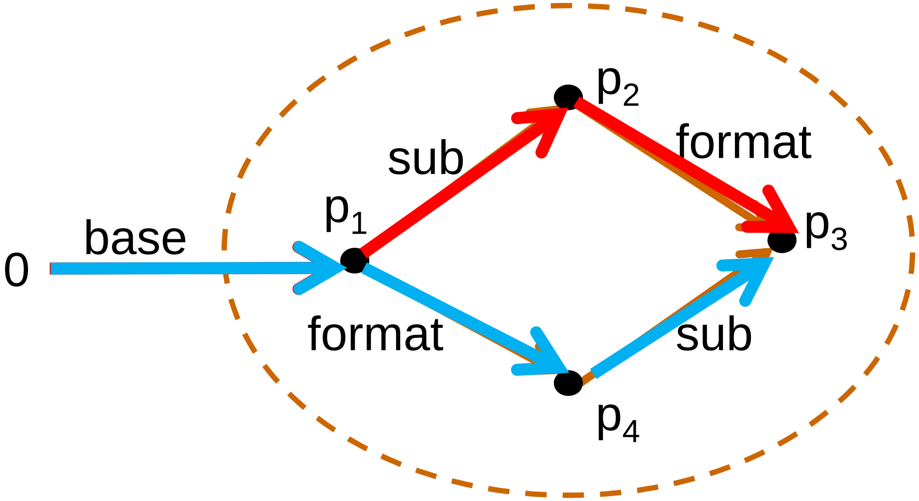


- SPL defines relationships between its programs
  - how are programs related?
  - by arrows, of course!

- Empty program (0) may (or may not) be part of SPL

- Each arrow is a **feature**

Computational Design



p_3 = format • sub • base

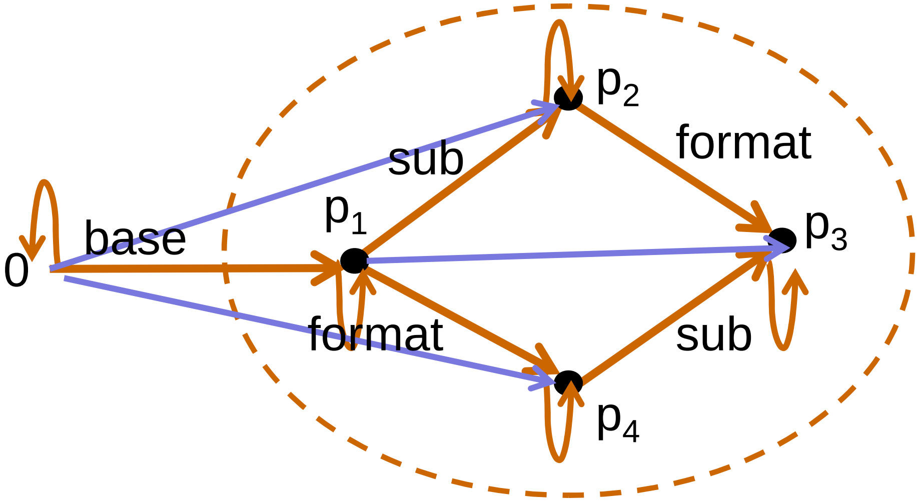
p_3 = sub • format • base

- Program design is an expression
  - RQO paradigm
  - programs can have multiple designs

evaluating both expressions yields the same program

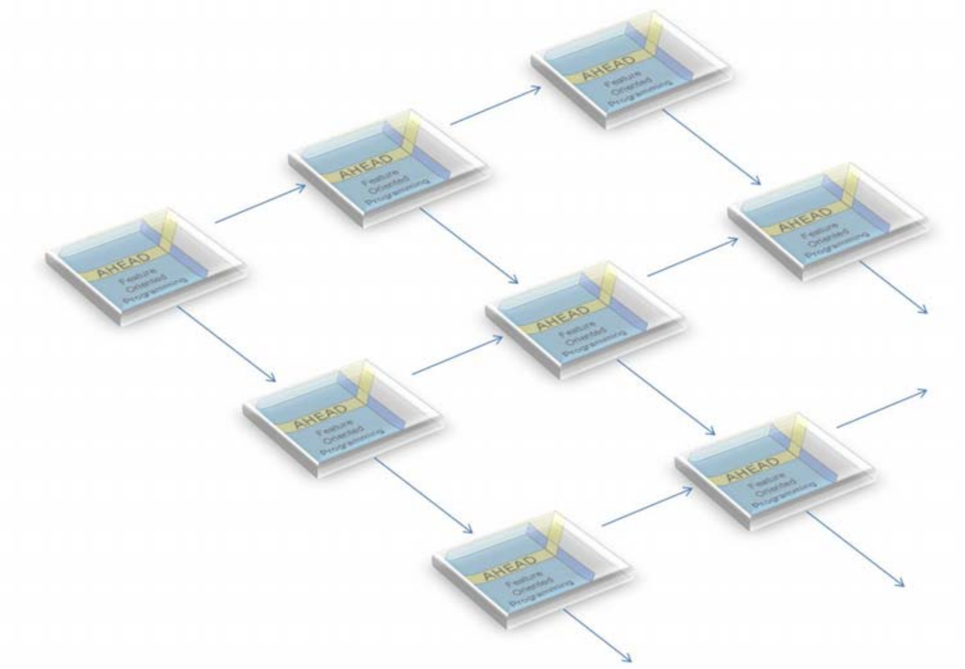
format, sub are **commutative**

A Product Line is a Category



- Degenerate or trivial category
  - point is a domain with a single program in it
- Has implied identity arrows
- Has implied composed arrows, as required

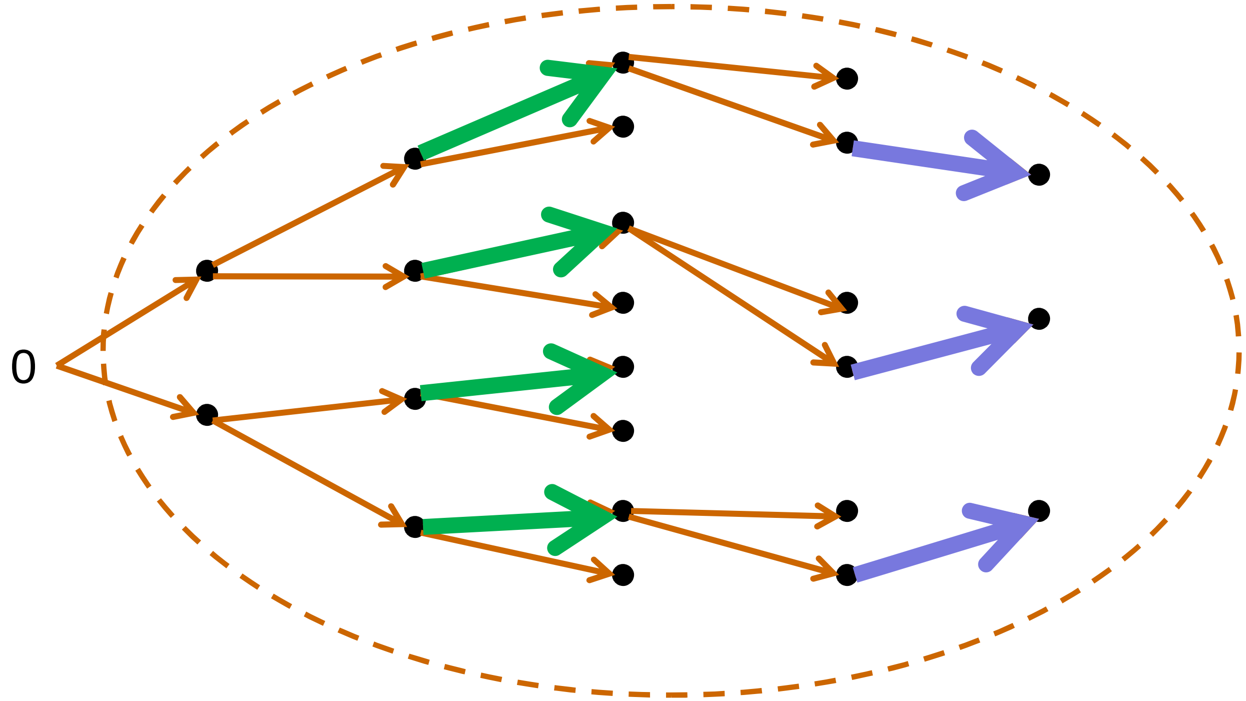
Implementing SPLs



Implementing SPLs

want this:

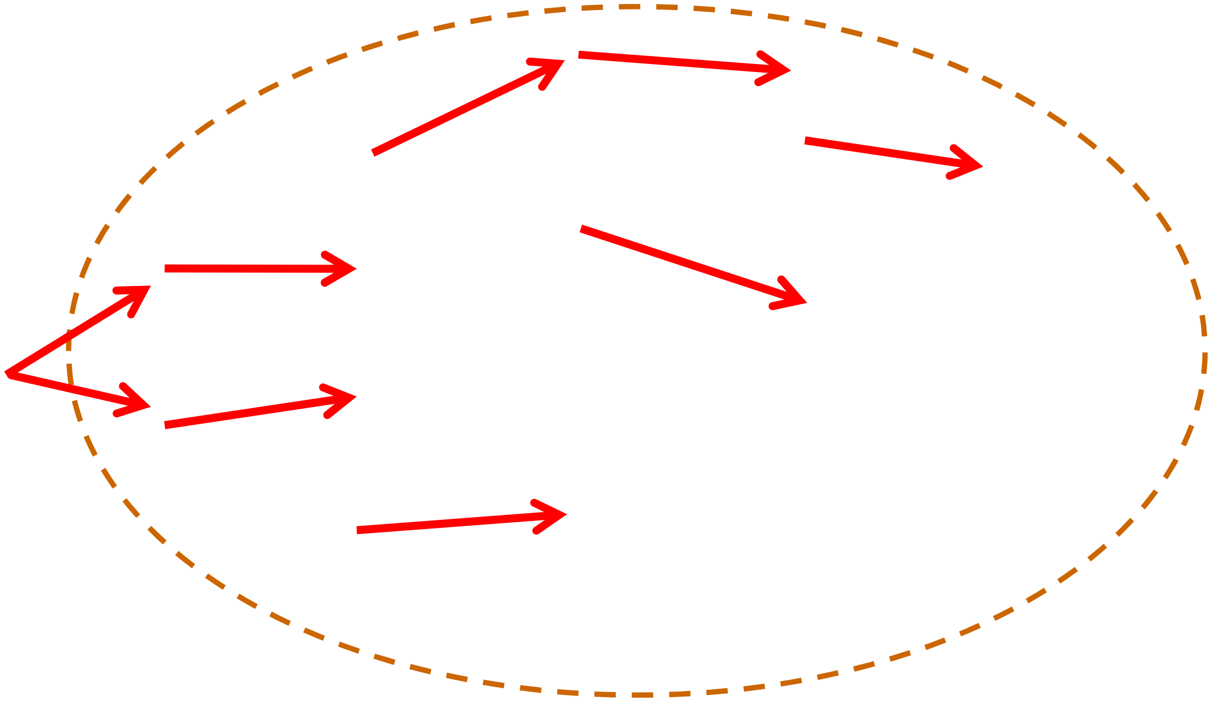
know this:



- Same function being applied to different inputs

Implementing SPLs

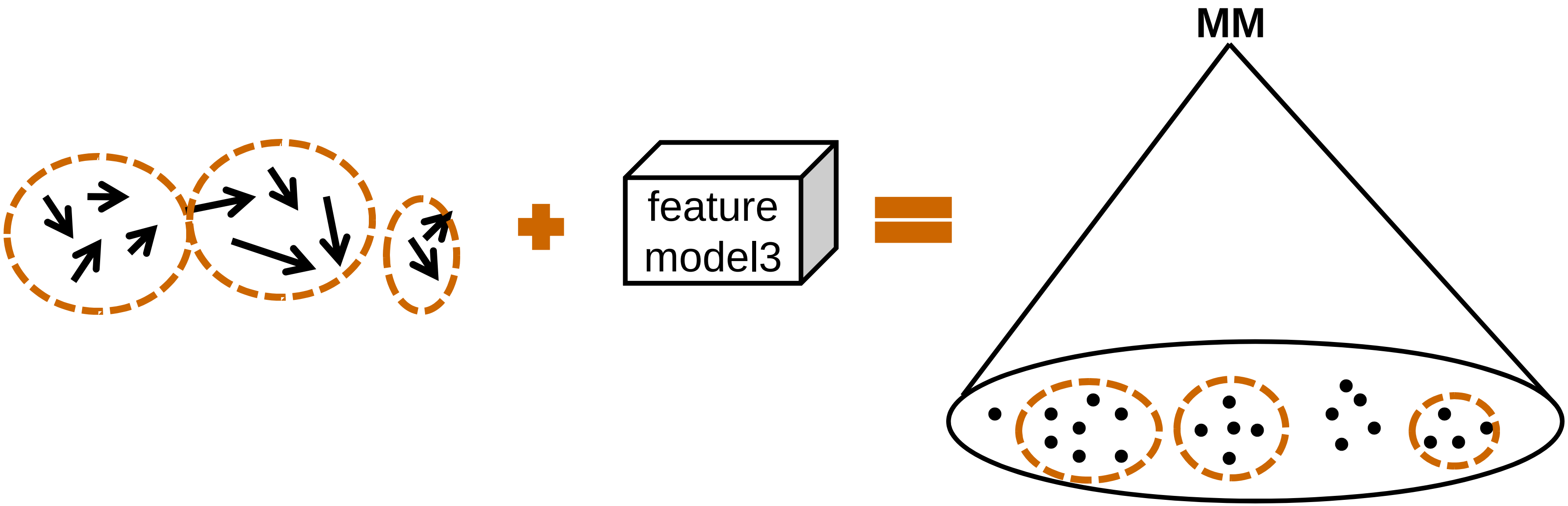
store this:



- Just store arrows regardless of how they are implemented
  - n optional features, 2^n possible programs
  - compact representation of an SPL

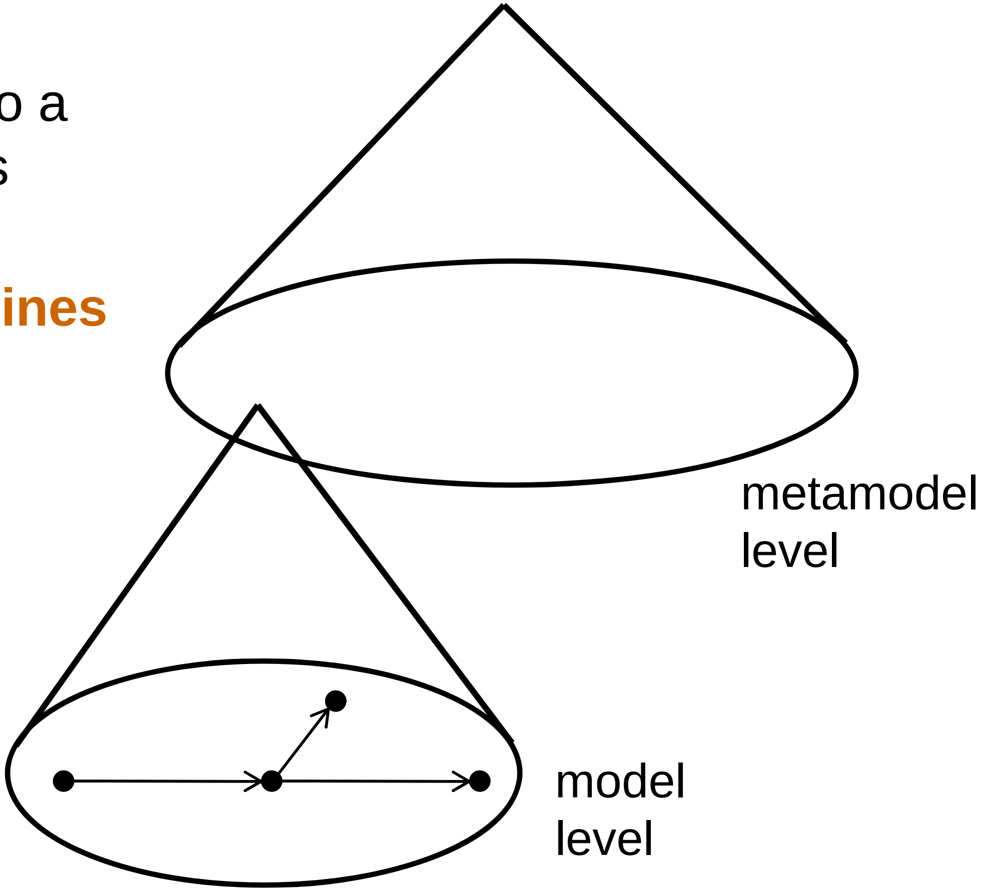
Models of SPLs

- Implement a set of arrows
- Define a feature model to define legal compositions of arrows
- Yields a product line
- See paper for more details



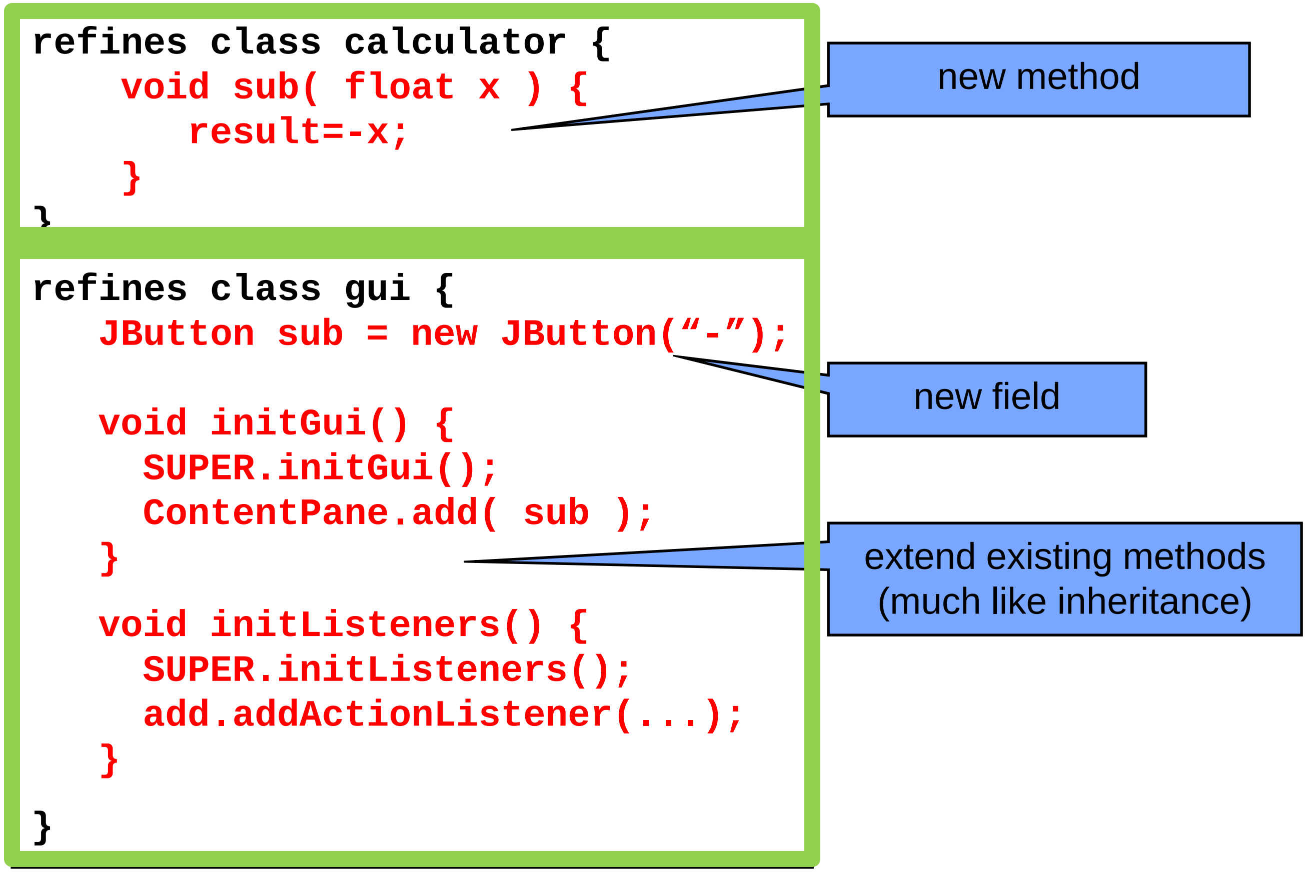
Recursion

- SPLs can appear at any level of an MDE architecture
  - arrow add same feature to a large domain of programs
  - **Model Driven Product Lines** to be discussed shortly
- **Superposition** is standard technique



Code Arrows in AHEAD

- AHEAD refines (Scala, ClassBox/J, …)
  - “sub” feature adds (superimposes) new fields, members, wrappers…



```
refines class calculator {
    void sub( float x ) {
       result=-x;
    }
}
```

```
refines class gui {
   JButton sub = new JButton(“-”);

   void initGui() {
     SUPER.initGui();
     ContentPane.add( sub );
   }

   void initListeners() {
     SUPER.initListeners();
     add.addActionListener(...);
   }
}
```

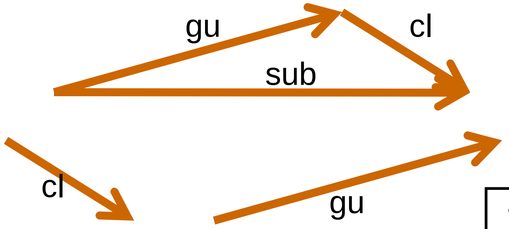
new method

new field

extend existing methods
(much like inheritance)

Code Arrows in AHEAD

- “sub” arrow is composed from 2 simpler arrows



```
refines class calculator {
    void sub( float x ) {
       result=-x;
    }
}

refines class gui {
   JButton sub = new JButton(“-”);

   void initGui() {
     SUPER.initGui();
     ContentPane.add( sub );
   }

   void initListeners() {
     SUPER.initListeners();
     add.addActionListener(...);
   }
}
```

sub

=

```
refines class calculator {
    void sub( float x ) {
       result=-x;
    }
}
```

cl

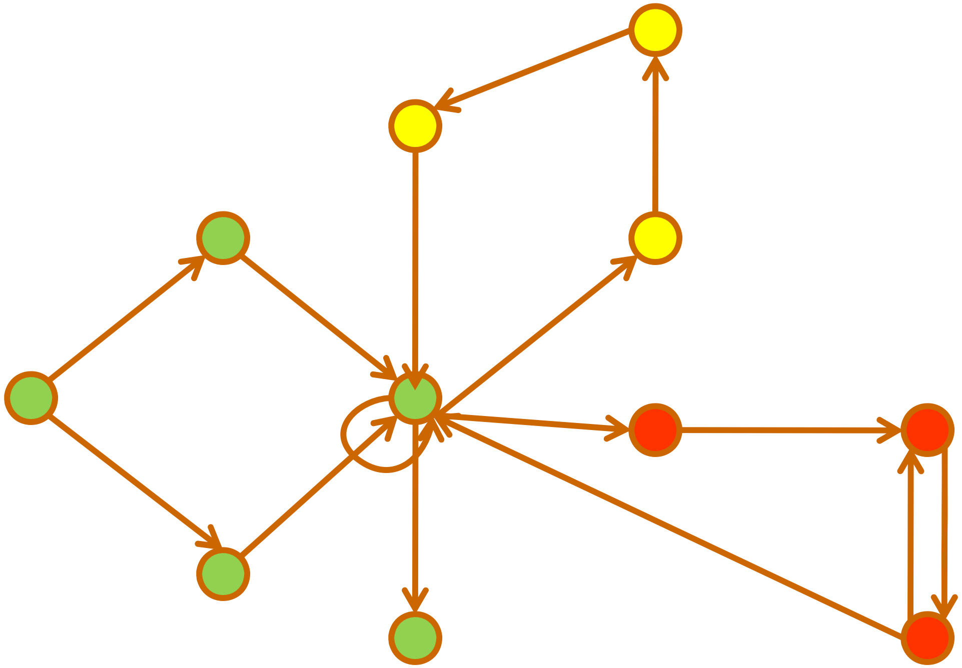
●

```
refines class gui {
   JButton sub   = new JButton(“-”);

   void initGui() {
     SUPER.initGui();
     ContentPane.add( sub );
   }

   void initListeners() {
     SUPER.initListeners();
     add.addActionListener(...);
   }
}
```

gu

Model Arrows in AHEAD

- Example: Product Lines of State Machines
  - ICSR 2000: fire support simulators
  - ICSE 2007: web portlet product line



Synthesizing customized state charts by composing features

Feature arrows are implemented by model deltas – additional elements and relationships are superimposed onto a base model

Makes SPL designs easier to understand and analyze

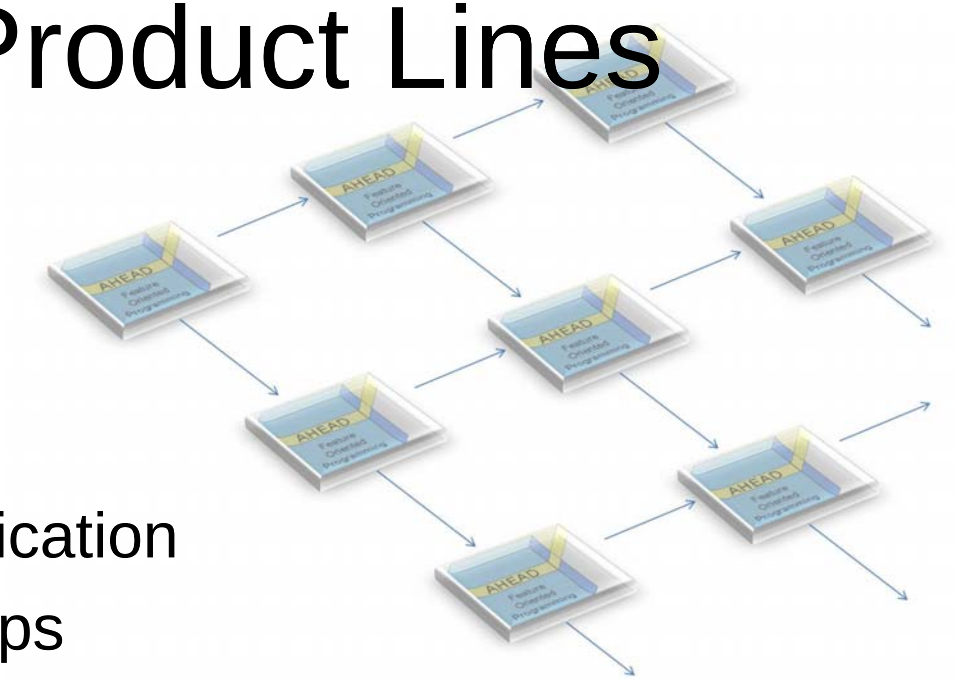
Don't use power of full transformation language to implement arrows

orange • yellow • base

Recap

- Categories lie at the heart of Software Product Lines
  - SPLs appear at all levels of an MDE architecture
- Informally, categories provides a clean set of ideas to express relationships that arise among objects in SPL
  - enabled me to place in perspective what MDE and SPL communities have been doing
- Next topic: **model-driven product lines (MDPLs)**

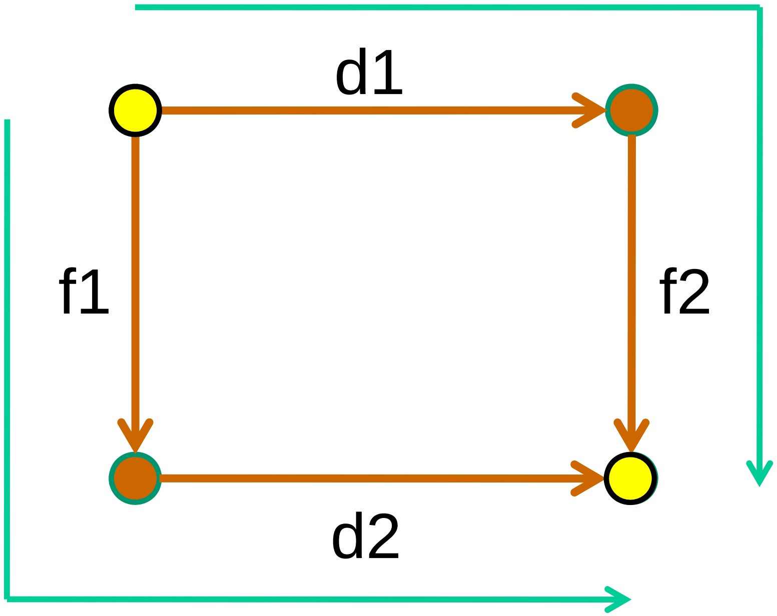
Part 3: Categories in Model Driven Product Lines



Exposing fundamental verification and optimization relationships

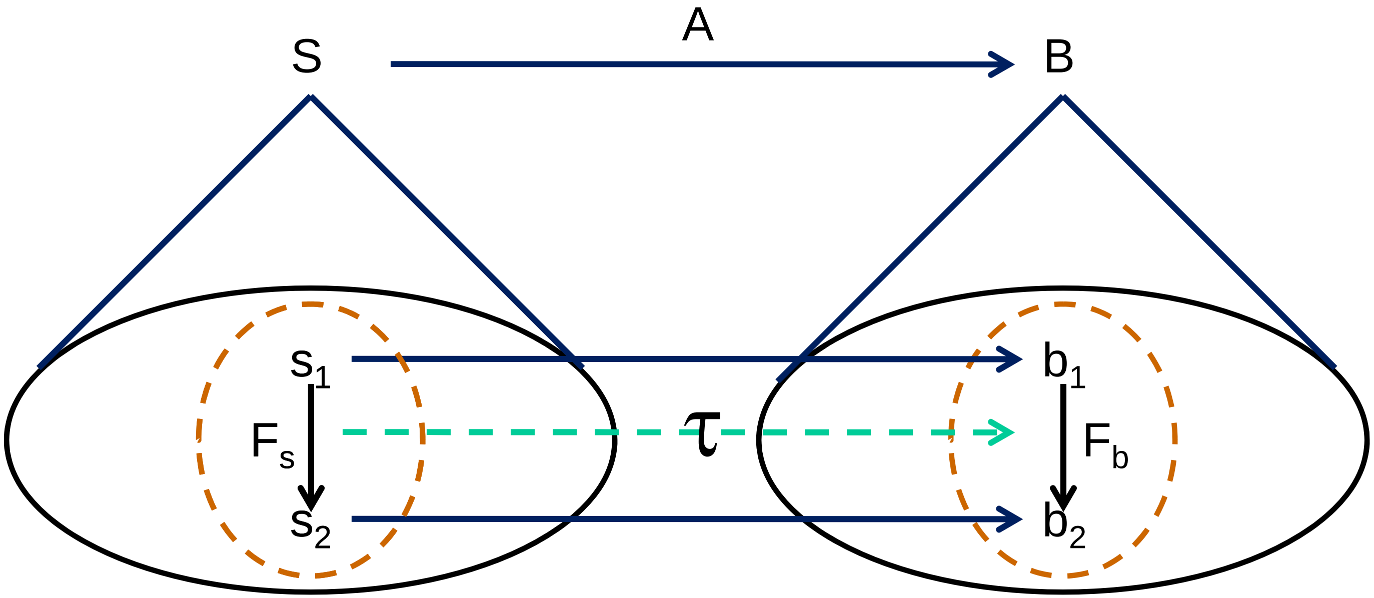
Commuting Diagram

- Fundamental concept in category theory
  - all paths between two objects yield same result
  - theorems of CT



$$f1 \bullet d2 = d1 \bullet f2$$

Diagrams in MDPLs

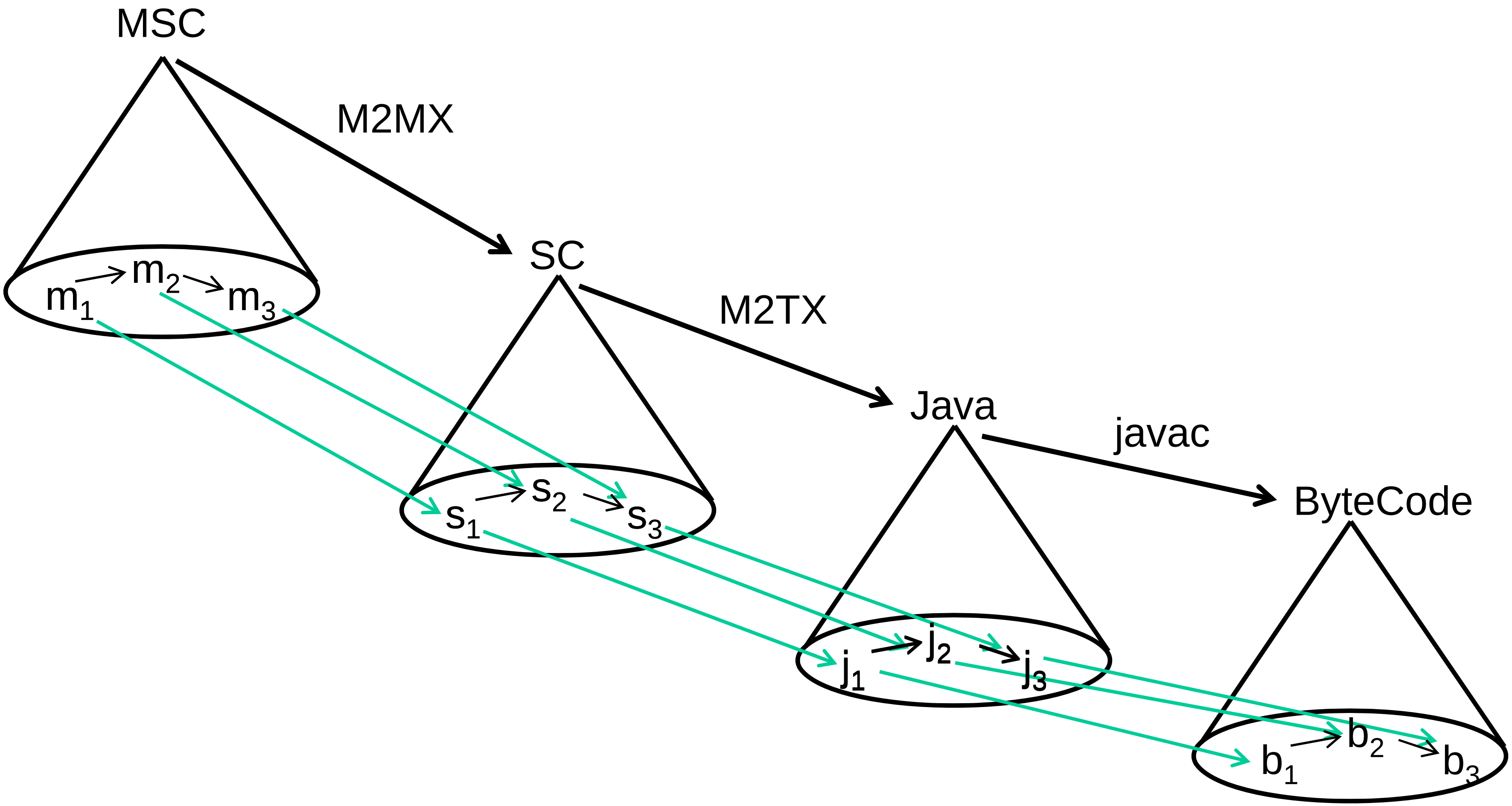


- Want to map a product line of S models into its corresponding product line of B models

- Transformations of MDE map objects

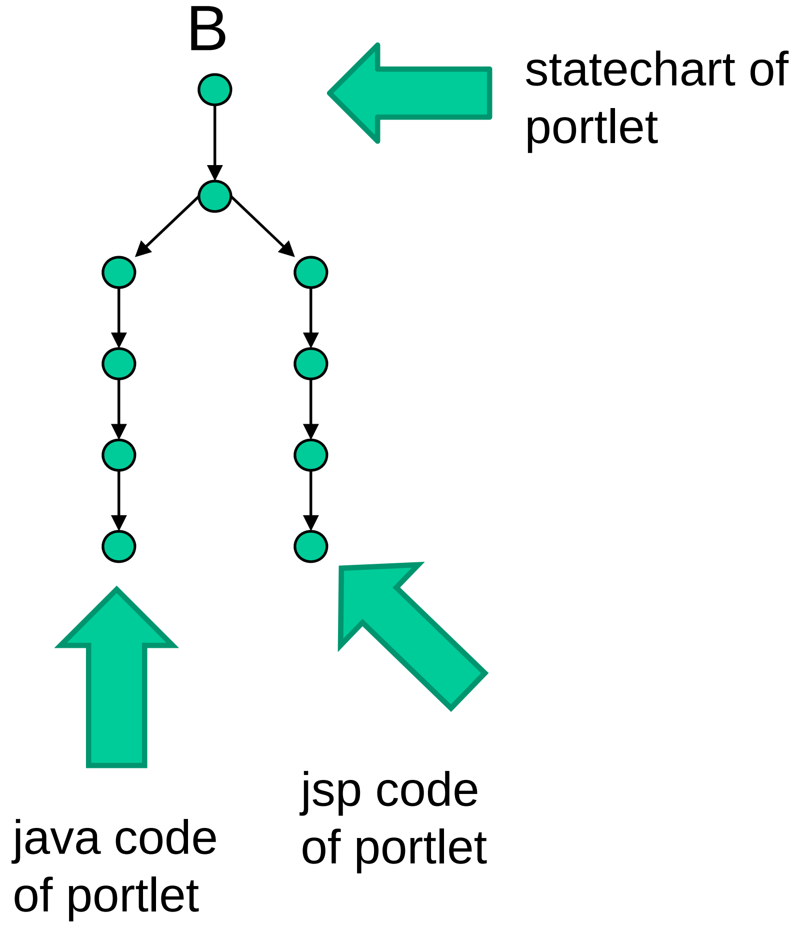
- Operator τ to map arrow F_s to arrow F_b: $\tau(F_s) = F_b$

How Commuting Diagrams Arise



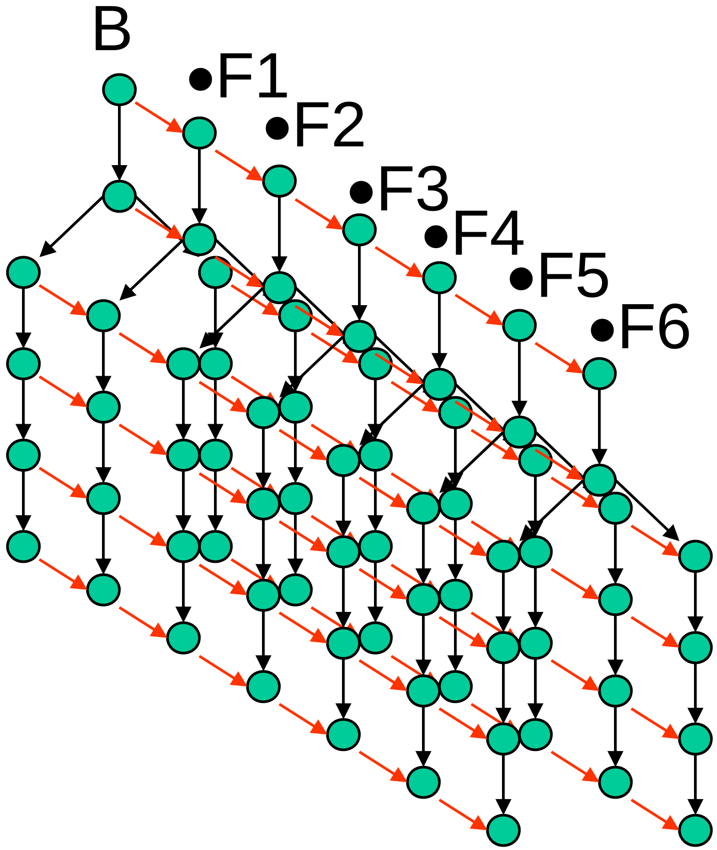
Commuting Diagrams in PinkCreek

- Trujillo, et al. ICSE 2007
- Portlet synthesis
- Transform state chart into a series of lower level representations until Java and JSP code reached



Commuting Diagrams in PinkCreek

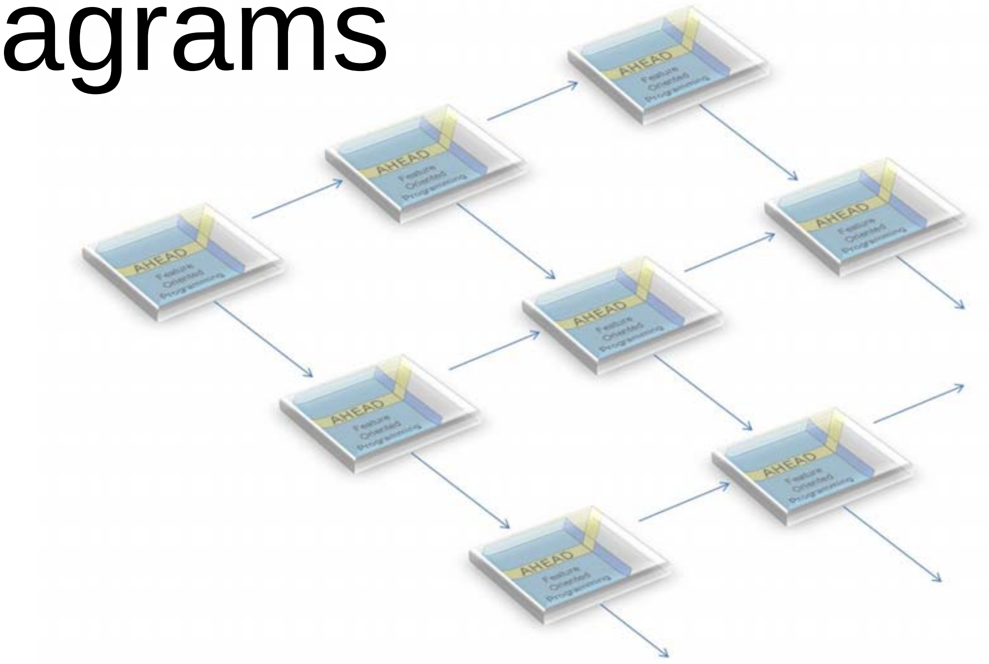
- Trujillo, et al. ICSE 2007
- Portlet synthesis
- Transform state chart into a series of lower level representations until Java and JSP code reached



Warning!

- Operators easy to draw…
  - may (or may not) be easy to implement
  - may (or may not) be practical to implement
  - CT is not constructive – it doesn’t say how to implement arrow
  - no more than UML class diagrams tell you how to implement a method
- Tells you certain relationships exist, and if you can implement arrows, you can exploit commuting diagrams

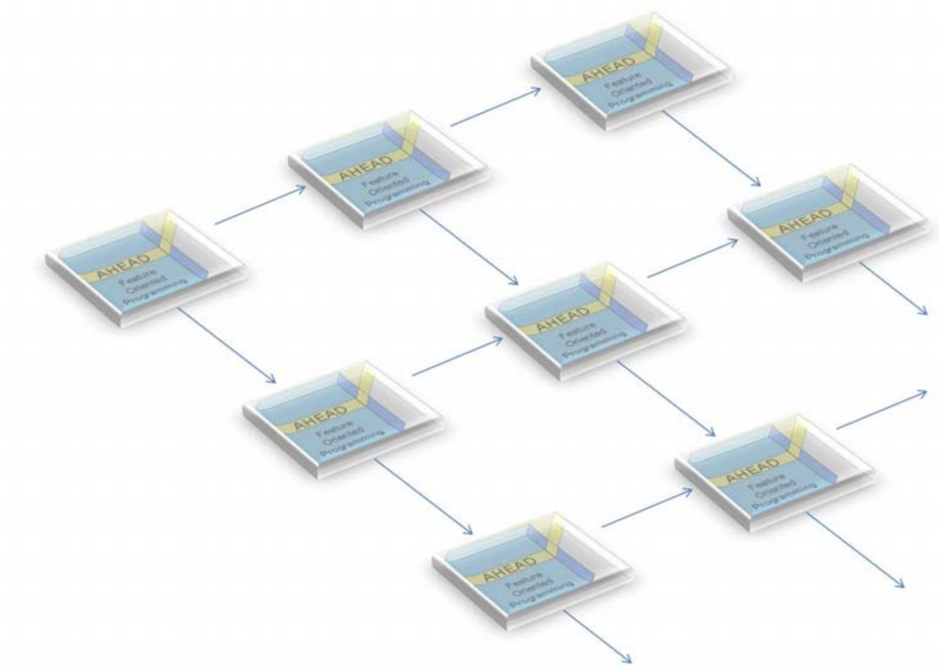
Examples that Exploit Commuting Diagrams


AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming

Writing Operators

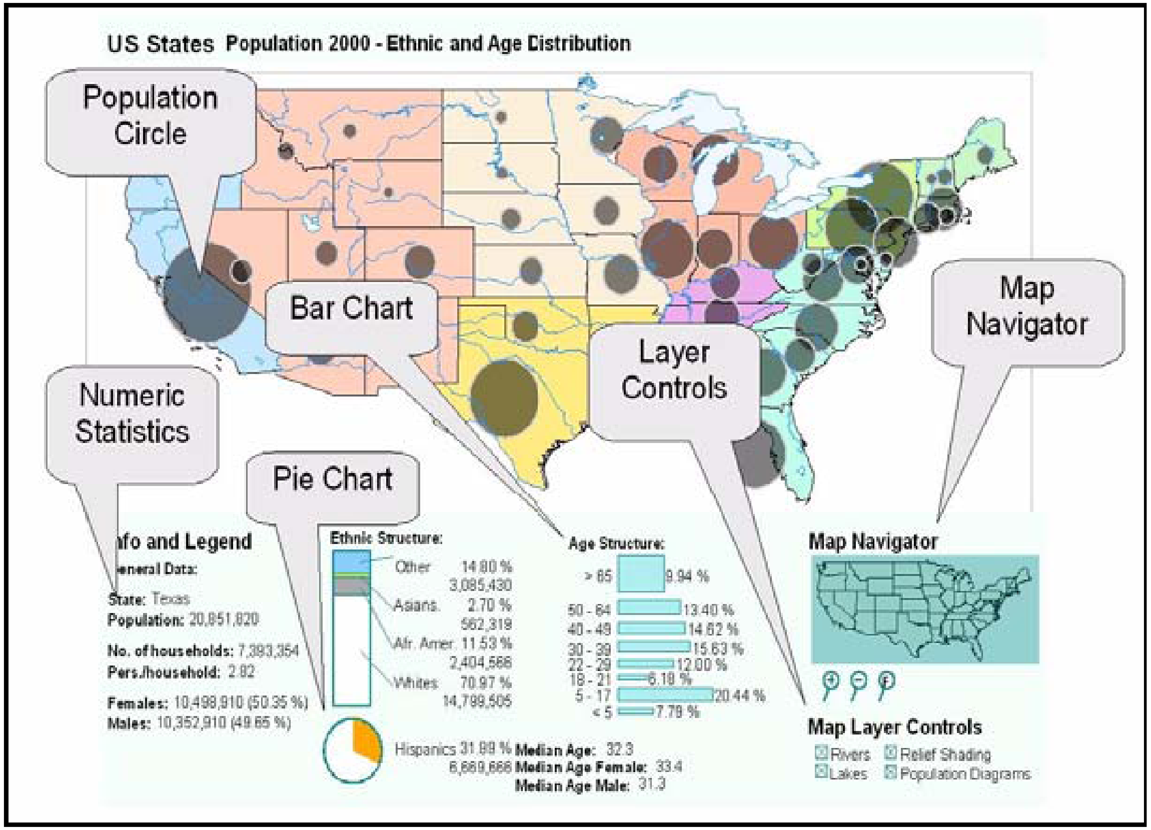
- In last 2 years, we found uses for commuting diagrams and arrow operators in MDPLs:
  - **simplifying implementation (ICMT 2008)**
  - **improving test generation (SIGSOFT 2007)**
  - understanding feature interactions (GPCE 2008)
  - understanding AHEAD (GPCE 2008)
  - improving synthesis performance (ICSE 2007)
- Briefly review the first two of these…

General Technique for Implementing MDPLs



Example 1: ICMT 2008 Paper

- Work with G. Freeman and G. Lavender
- MDPL of applications written in SVG and JavaScript
  - selectively customize application (removing, adding charts, controls)



US States Population 2000 - Ethnic and Age Distribution

nfo and Legend

eneral Data:

State: Texas
Population: 20,851,820

No. of households: 7,393,354
Pers./household: 2.82

Females: 10,498,910 (50.35 %)
Males: 10,352,910 (49.65 %)

Ethnic Structure:

Other	14.80 %	3,085,430
Asians.	2.70 %	562,319
Afr. Amer.	11.53 %	2,404,566
Whites	70.97 %	14,799,505
Hispanics	31.99 %	6,669,666

Age Structure:

> 65	9.94 %
50 - 64	13.40 %
40 - 49	14.62 %
30 - 39	15.63 %
22 - 29	12.00 %
18 - 21	6.18 %
5 - 17	20.44 %
< 5	7.79 %

Median Age: 32.3
Median Age Female: 33.4
Median Age Male: 31.3

Map Navigator

Map Layer Controls

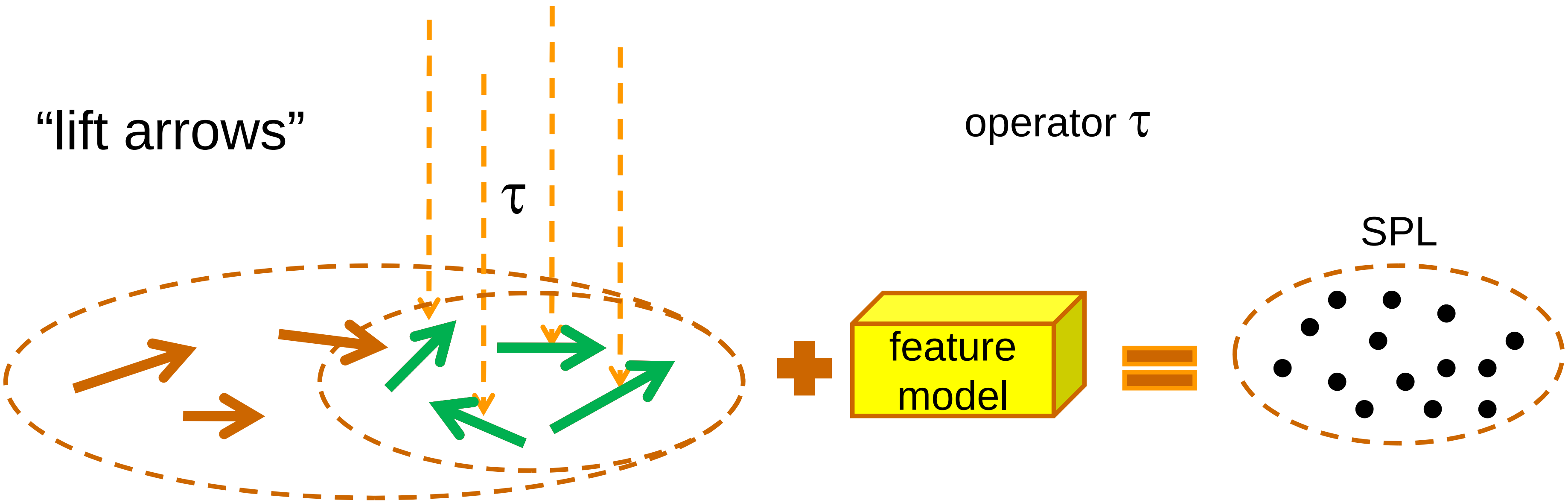
☒ Rivers ☒ Relief Shading
☒ Lakes ☒ Population Diagrams

Example 1: ICMT 2008 Paper

- Created a set of arrows and a feature model for our MDPL
    - red arrows (defining a product line of charts) were tedious to write
    - created DSL for charts, where arrows were easy to write, compose
    - defined an operator τ to map 1:1 from green arrows to red arrows

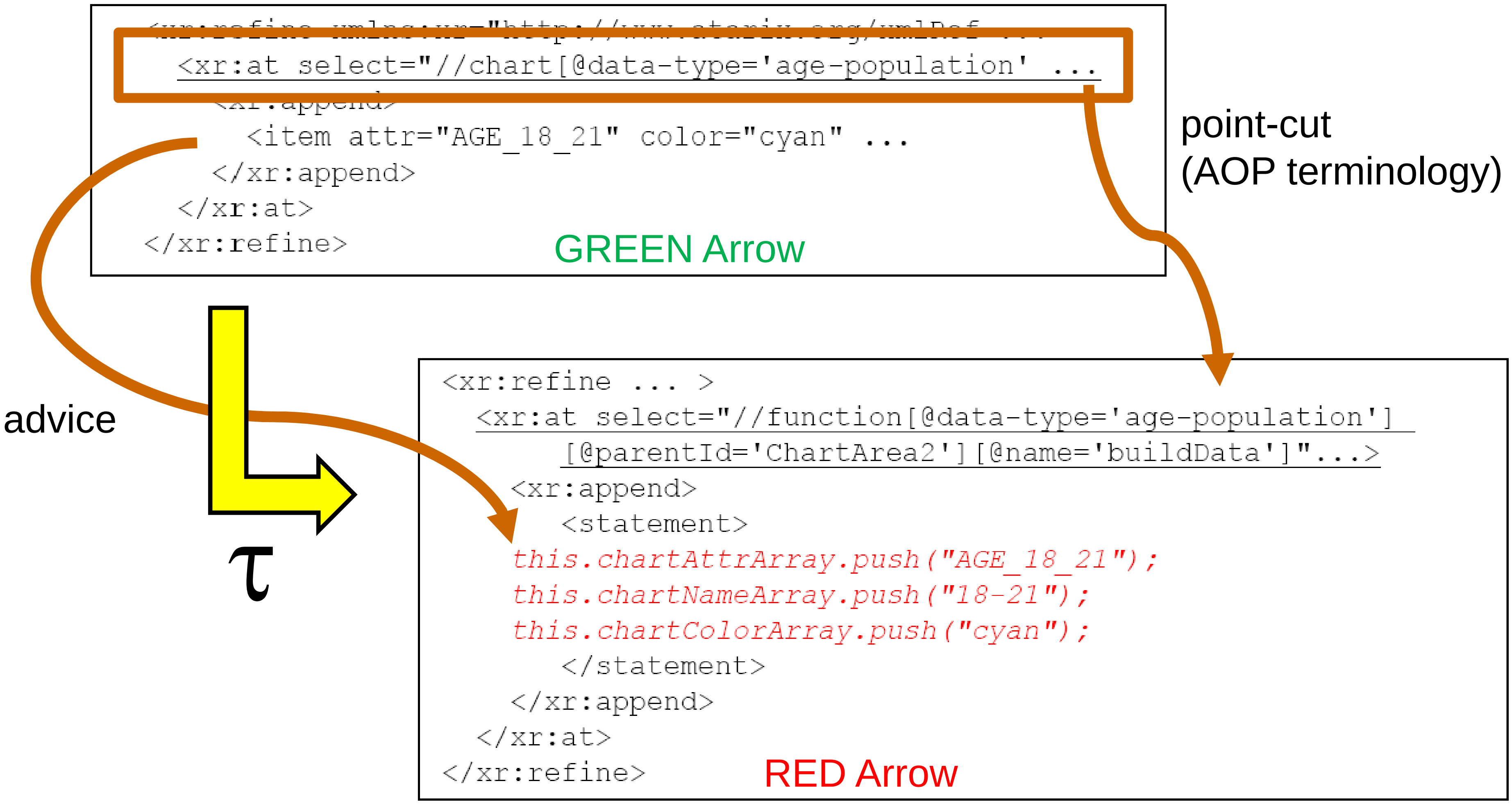
DSL for chart arrows

operator τ



“lift arrows”

τ Translation Example



```
<xr:refine xmlns:xr="http://www.atarix.org/xmlRef ...
  <xr:at select="//chart[@data-type='age-population' ...
    <xr:append>
      <item attr="AGE_18_21" color="cyan" ...
    </xr:append>
  </xr:at>
</xr:refine>
```

GREEN Arrow

point-cut
(AOP terminology)

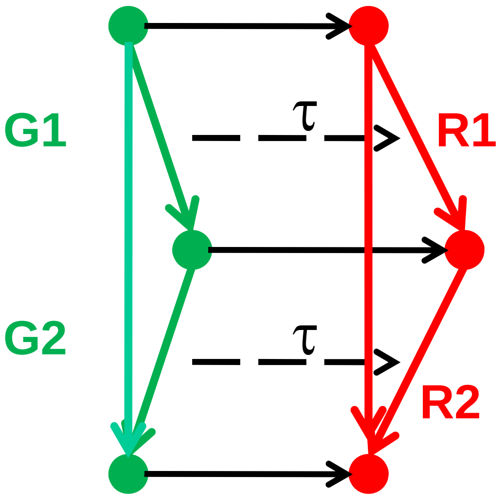
advice

τ

```
<xr:refine ... >
  <xr:at select="//function[@data-type='age-population']
       [@parentId='ChartArea2'][@name='buildData']"...>
    <xr:append>
       <statement>
    this.chartAttrArray.push("AGE_18_21");
    this.chartNameArray.push("18-21");
    this.chartColorArray.push("cyan");
       </statement>
    </xr:append>
  </xr:at>
</xr:refine>
```

RED Arrow

Diagram Constraints



$$\tau(G1 \bullet G2) = \tau(G1) \bullet \tau(G2)$$

$$= R1 \bullet R2$$

- Same ranslate OR we tr rows

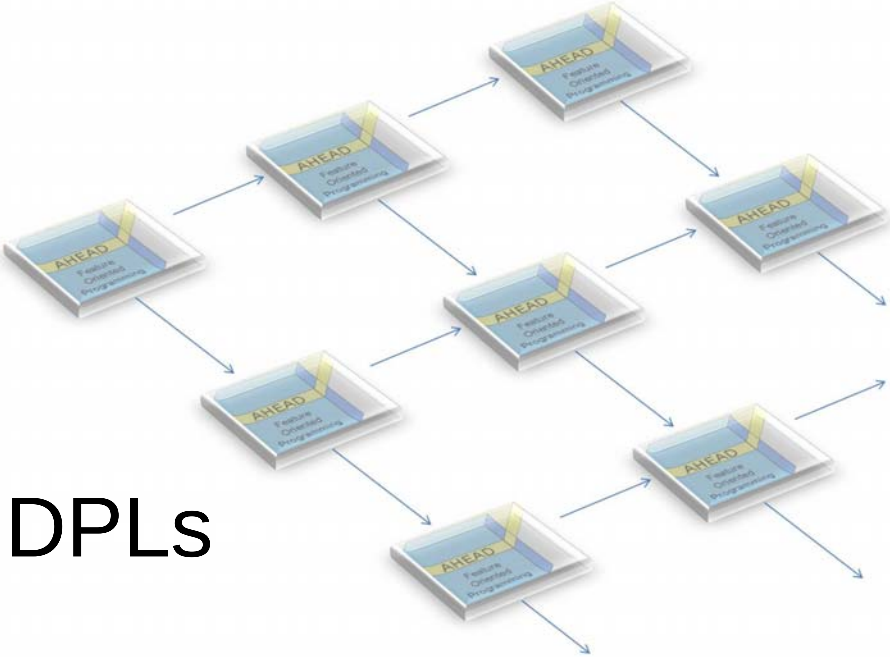
- **Hom** ne algebra (**GRE** ther (**RED**)

Verification condition: Implementation is correct if this equality holds!

Guess What?

- Initially our tools did not satisfy diagram constraints
  - equalities of homomorphisms didn’t hold
  - our tools had bugs – we had to fix our tools
  - now we have greater confidence in tools because they implement explicit relationships of domain models
  - win from engineering perspective
    - » we have an insight into domains that we didn’t have before
    - » by imposing categorical structure on our domain, we have better understanding, better models, and better tools
- Lifting is not specific to our application,
  it is a general technique for building MDPLs

Test Generation for MDPLs


AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming
AHEAD
Feature Oriented Programming

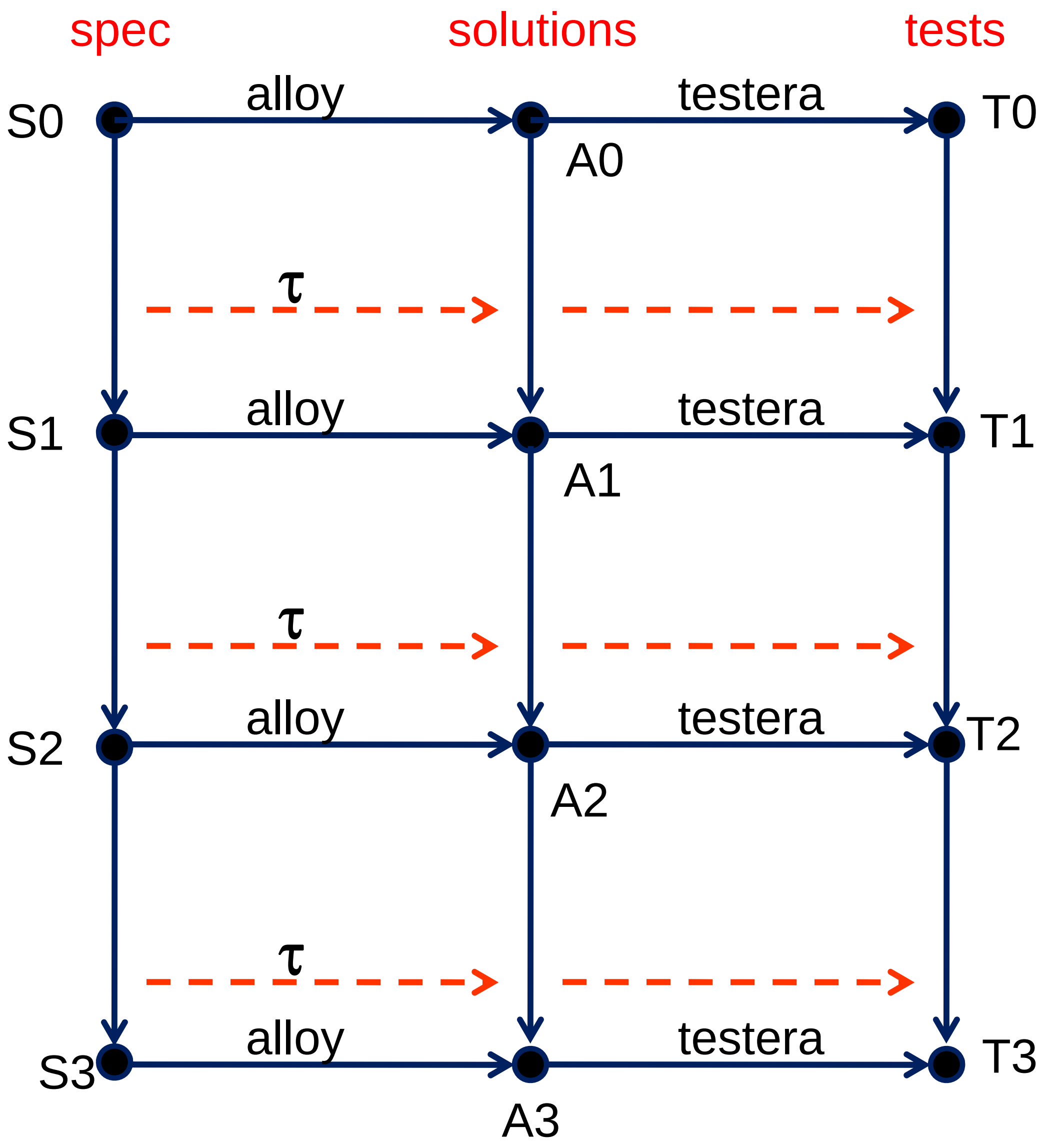
Example 2: ISSRE 2008 Paper

- Work with E. Uzuncaova and S. Khurshid (ECE@UTexas)

- Testing SPLs is a basic problem
  - we can generate different programs, but how do we know that the programs are correct?

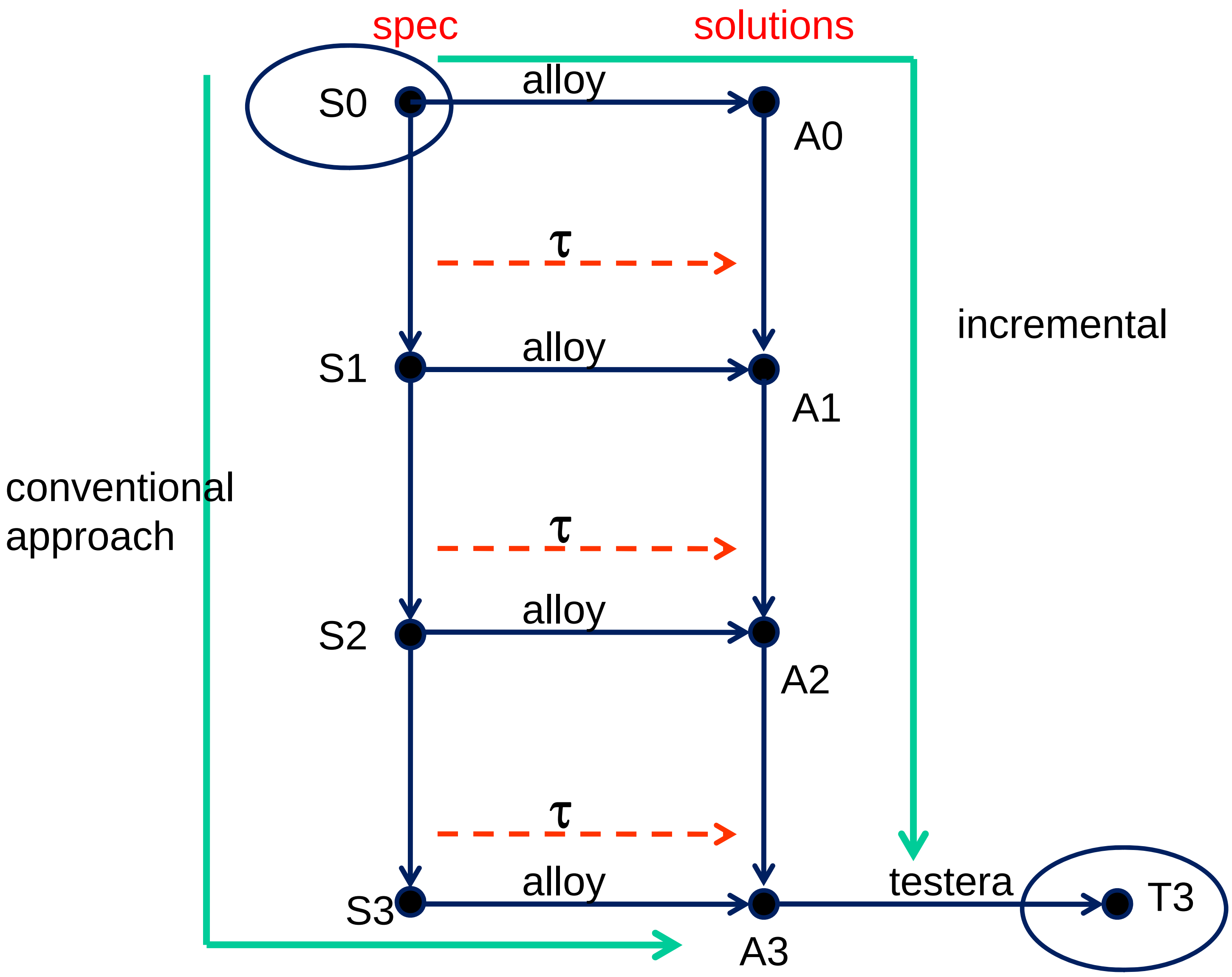
- Specification-based testing can be effective
  - start with a spec (model) of program
  - automatically derive tests
  - Alloy is example

Conventional Test Generation



Challenge:
is there
a τ
operator?

Incremental Test Generation



Implementing τ

- Spec S1 = $(A \vee B) \wedge (\neg A \vee C)$ // 20K clauses

  a solution: [A,B,C] = [$\underline{1, 0, 1}$]

- Spec S2 = $(A \vee B) \wedge (\neg A \vee C) \wedge (D \vee \neg A)$

  a solution: [A,B,C,D] = [$\underline{1, 0, 1}$, 1]

- Solution for S1 “bounds” solution for S2
  - sound, complete

- Reason for efficiency…

Preliminary Results

- In product lines that we examined (typical of Alloy research), majority of cases incremental approach is faster
  - 30-50× faster
  - can now solve larger problems with Alloy

- Although preliminary, results are encouraging

- See paper(s) for details

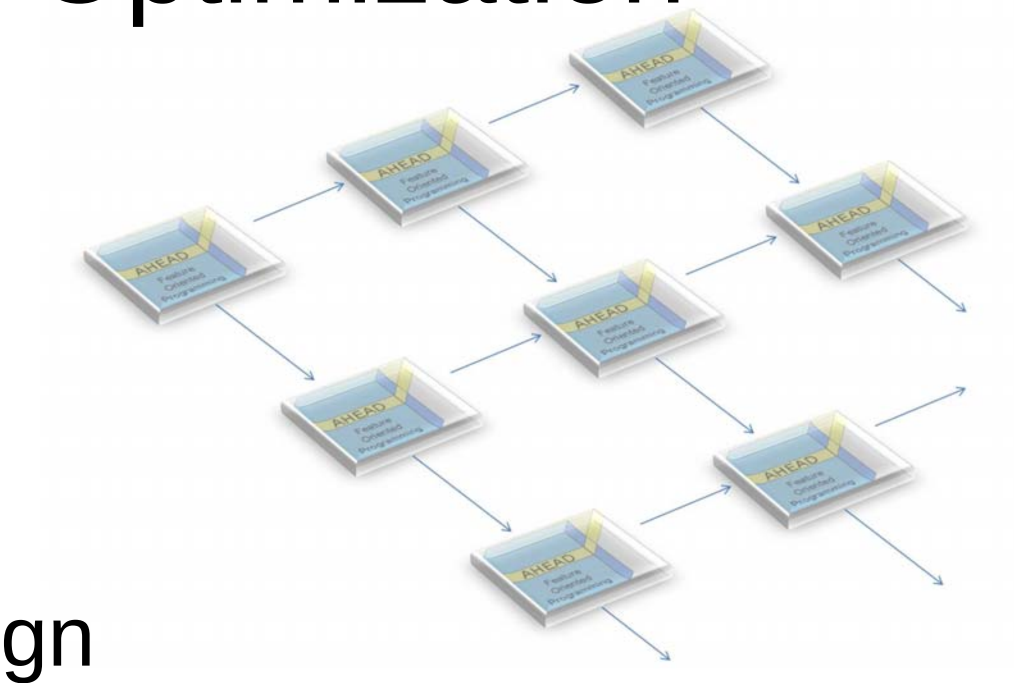
Recap

- Creating a SPL or MDE application creates industrial–sized categories

- Putting them together reveals a foundational idea of categories – commuting diagrams
  - involves mapping both objects of a category AND arrows
  - need operators (transformation – to – transformation maps)

- Can exploit exposed relationships as
  - verification conditions
  - optimization possibilities

Part 4: Design Optimization

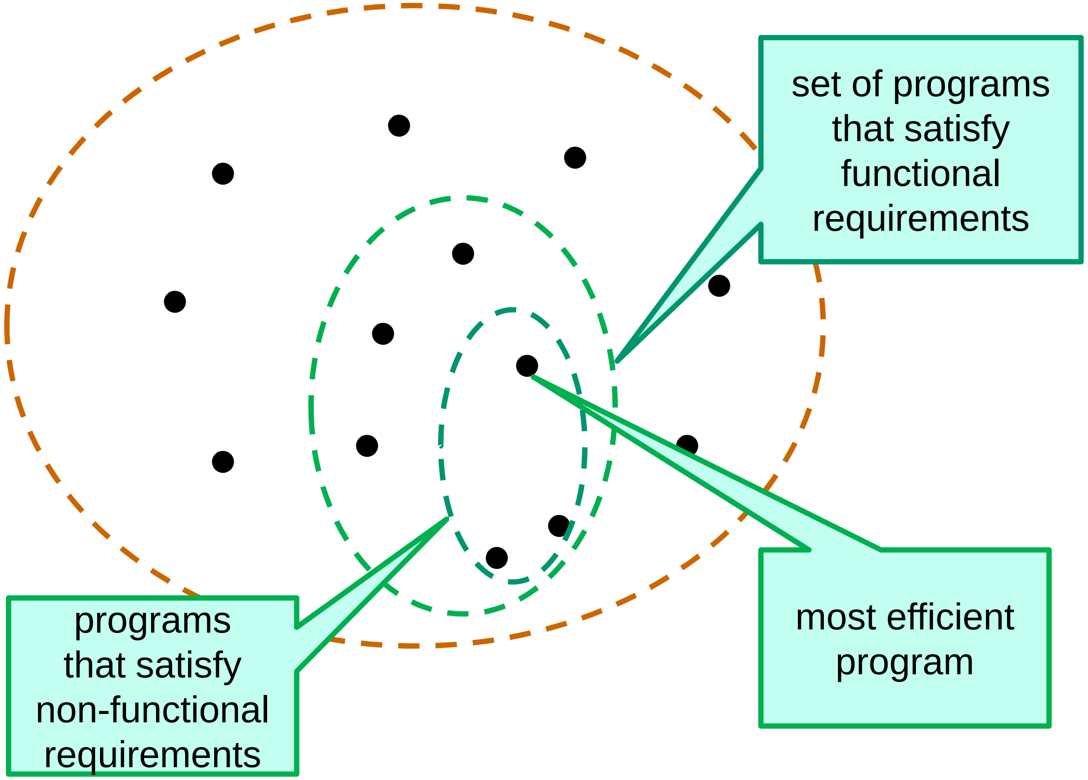


Frontier of
Computational Design

Principles of Computational Design

- Design = expression
- Synthesis = expression evaluation
- Design Optimization = expression optimization
  - find program that satisfies functional requirements and optimizes some non-functional properties (performance, energy consumption)

paradigm of relational query optimization



set of programs that satisfy functional requirements

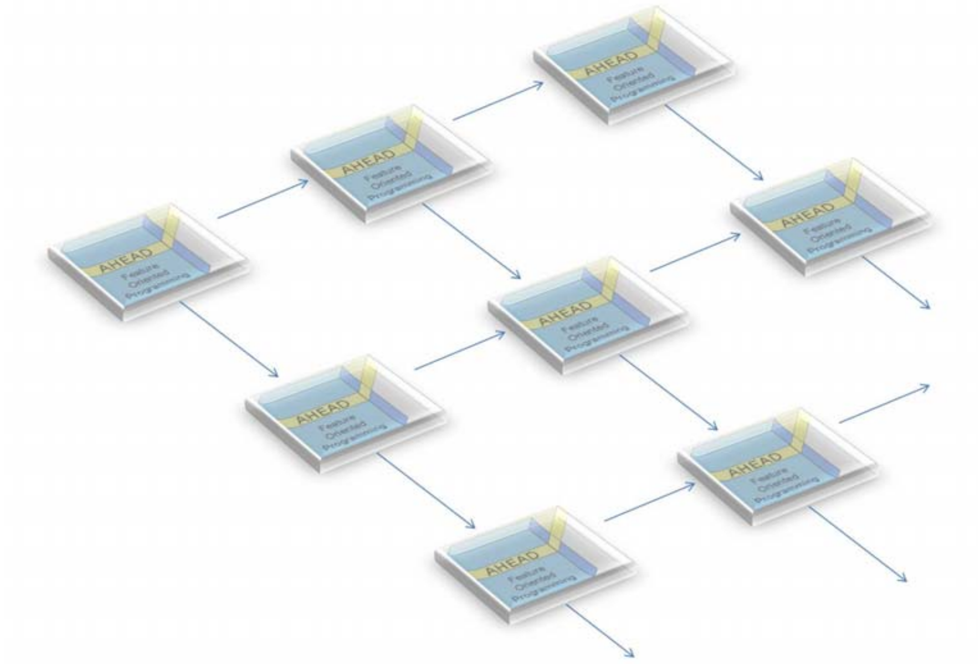
programs that satisfy non-functional requirements

most efficient program

At Present…

- I know of few examples of design optimization …
    - relational query optimization (1980s)
    - data structure optimizations (1990s)
    - Neema’s work on synthesizing adaptive computing (2001)
    - Püschel’s work on numerical library synthesis (2006)
    - Benavides work on configurators (2005)
    - …
- Main challenge: finding domains where there are different ways to implement the same functionality
    - commuting diagrams
    - this is where design optimization occurs
- If you think in terms of arrows, you have a conceptual framework and tools to explain and address design optimization in a principled, non-ad hoc way

Conclusions



Role of Mathematics in Design

- RQO helped bring database systems out of stone age

- Relational Model was based on set theory
  - this was the key to understanding a modern view of databases
  - set theory used was shallow
  - fortunate for programmers and database users
  - set select, union, join, intersect
  - disappointment for mathematicians

- Computational Design uses category theory
  - basic ideas useful – allows us to place research results in context
  - new insight on verification, optimization issues
  - whether theorems from CT are applicable, I don’t know

Key To Success

- Educational benefit of the connection
  - common terminology, concepts
  - new perspectives

- How often in MDE, SPL, MDPL do commuting diagrams arise?
  - don’t know; too early
  - but if you look, you’ll find them
  - theory says they exist
  - whether creation of operators practical decided on a per domain basis

Look for Them!

Final Comments

- Future of software design and synthesis is in automation
    - seeking principles that have a mathematical basis
- Made great strides understanding structure (UML)
- Need to take few more strides in understanding arrows
    - tools, transformations used in design and synthesis
- Software design & synthesis seems to rest on simple ideas
    - programs (models) are values
    - transformations map programs to programs
    - operators map transformations to transformations

Final Comments

- Clear that ideas are being reinvented in different contexts
  - not accidental – evidence we are working toward general paradigm
  - modern mathematics provides a simple language to express computational designs, expose useful relationships in SPL and MDE architectures
  - maybe others may be able to find deeper connections