



# Model-based software development

## Lecture X.

### Model transformation, Graph pattern matching

Dr. Semeráth Oszkár

# Model transformation

**Definitions**

**Model Transformation Chains**

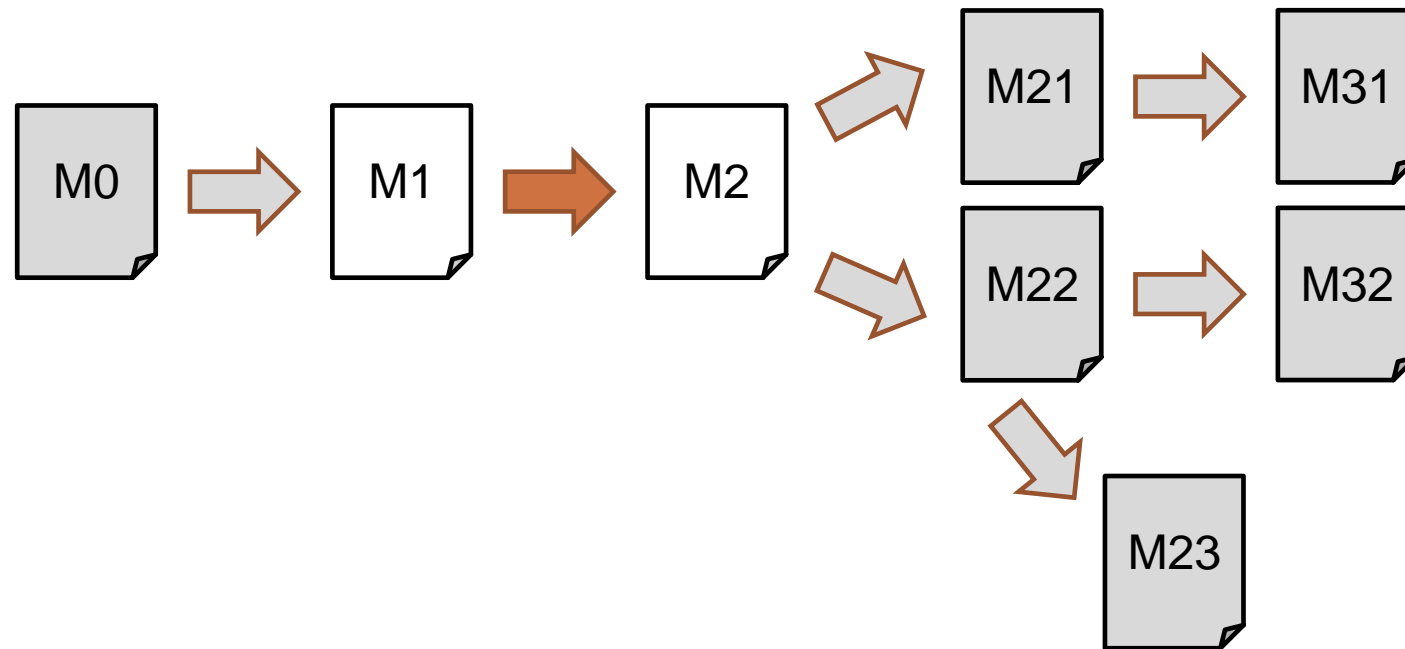
**Rule-based model transformations**

**Technologies**



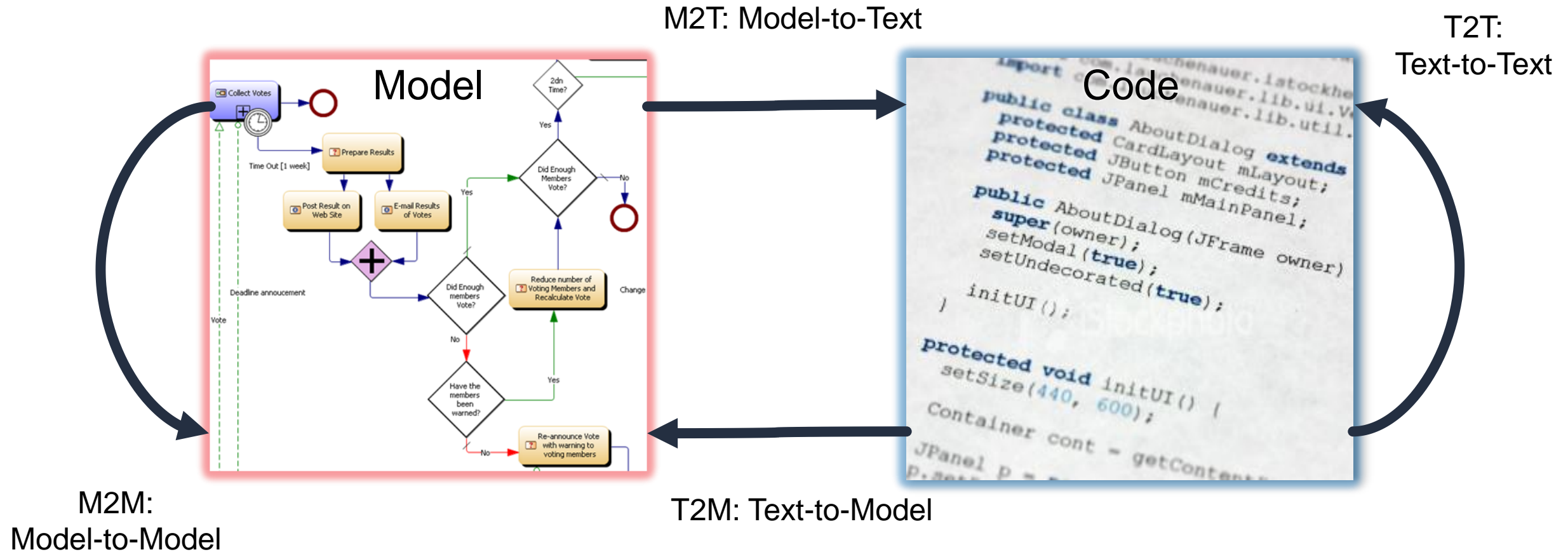
# Motivation: Transformation of models

- **Model-based development:** Models as primary documents
- Developing models, automating model processing



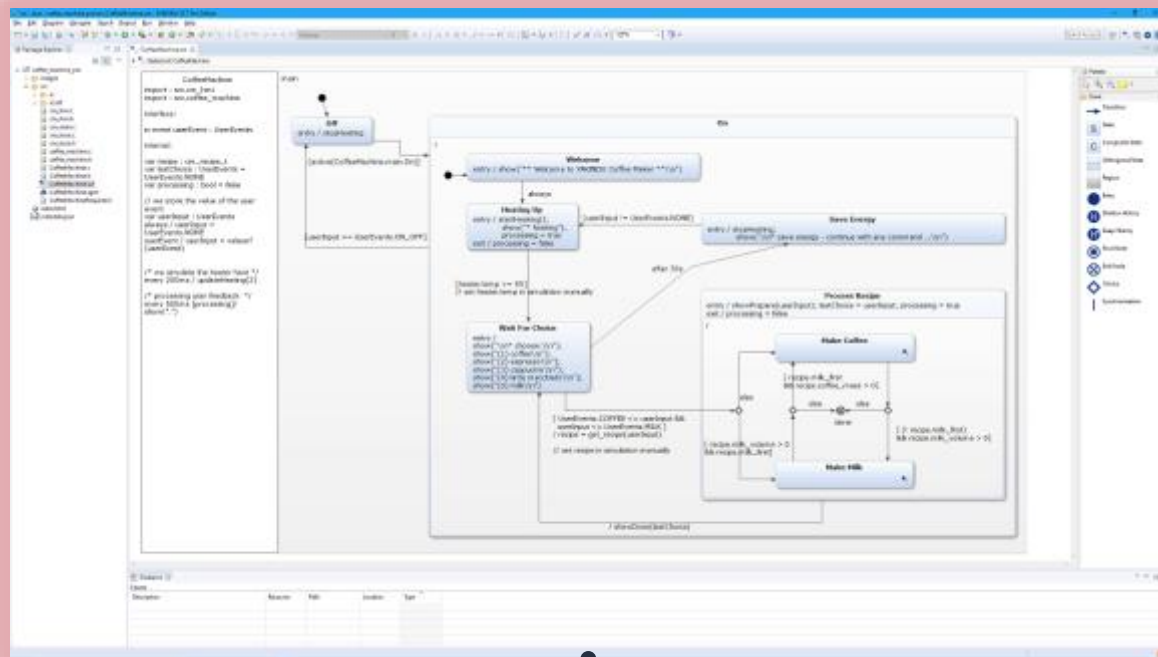
- **Goal:** to efficiently formulate and implement model transformations

# Types of transformation



# M2T Example

## ■ Code generation



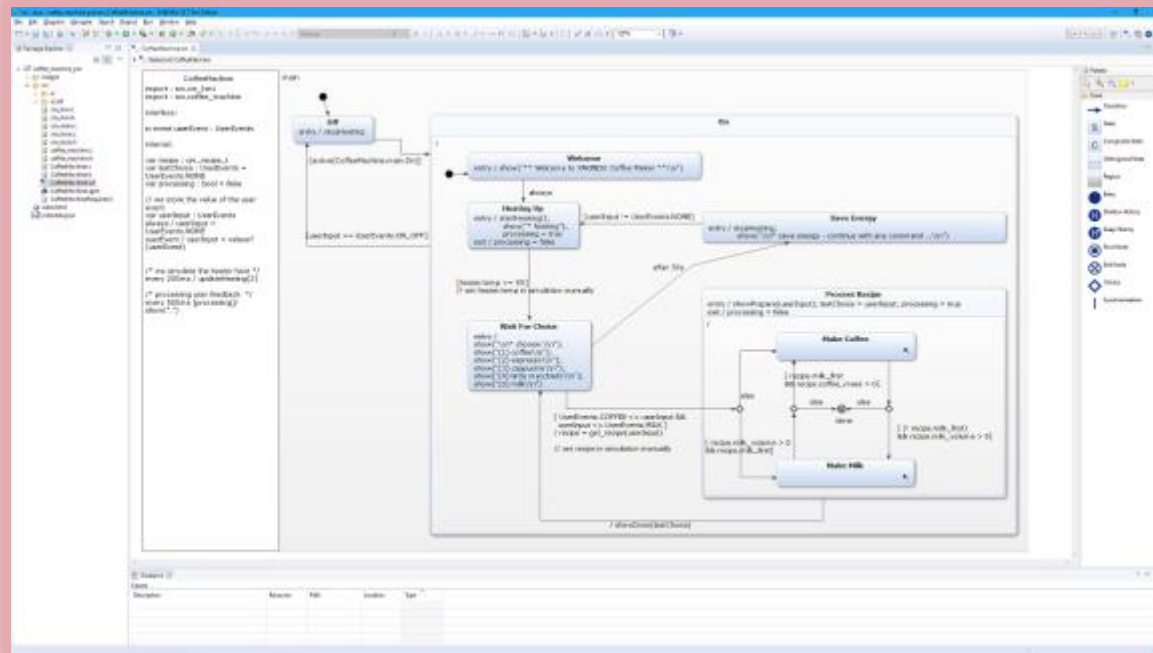
```
#ifndef DEFAULTSM_H_
#define DEFAULTSM_H_
#include "sc_types.h"
#include "StateMachineInterface.h"

class DefaultSM : public StateMachineInterface
{
public:
    DefaultSM();
    ~DefaultSM();
    /*! Enumeration of all states */
    typedef enum
    {
        main_region.MyState,
        DefaultSM_last_state
    } DefaultSMStates;
    /*! Inner class for Sample interface scope.
    class SCI_Sample
    {
    public:
        /*! Gets the value of the variable 'a' that is defined in the interface scope 'Sample'. */
        sc_boolean get_a();
        /*! Sets the value of the variable 'a' that is defined in the interface scope 'Sample'. */
        void set_a(sc_boolean value);
        /*! Raises the in event 'evA' that is defined in the interface scope 'Sample'. */
        void raise_evA(sc_boolean value);
        /*! Checks if the out event 'evB' that is defined in the interface scope 'Sample' has been raised. */
        sc_boolean isRaised_evB();
        /*! Gets the value of the out event 'evB' that is defined in the interface scope 'Sample'. */
        sc_integer get_evB_value();
    private:
        friend class DefaultSM;
        sc_boolean a;
        sc_boolean evA_raised;
        sc_boolean evA_value;
        sc_boolean evB_raised;
        sc_integer evB_value;
    };
    /*! Returns an instance of the interface class 'SCI_Sample'. */
    SCI_Sample* getSCI_Sample();
    void init();
    void enter();
    void exit();
    void runCycle();
    sc_boolean isActive();
    sc_boolean isFinal();
    sc_boolean isStateActive(DefaultSMStates state);
private:
    static const sc_integer maxOrthogonalStates = 1;
    DefaultSMStates stateConfVector[maxOrthogonalStates];
    sc_ushort stateConfVectorPosition;
};
```



# T2M Example

- Representing code artifacts in models



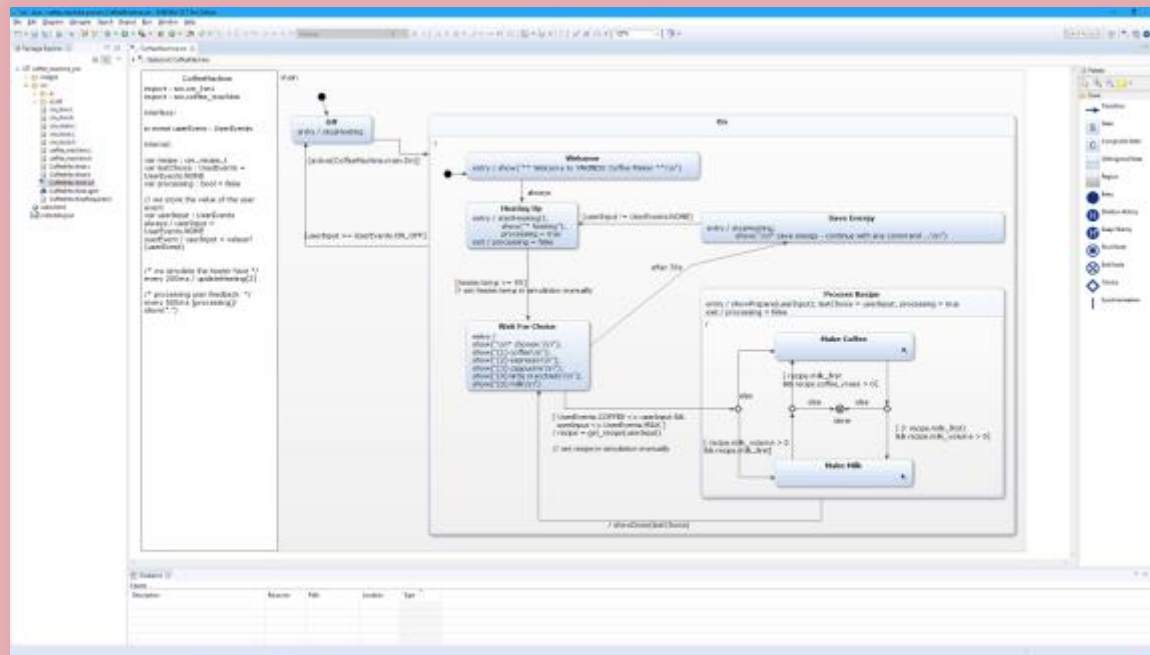
```
class Point
{
    public:
        int32_t get_x();
        void set_x(int32_t x);
        int32_t get_y();
        void set_y(int32_t y);
    private:
        int32_t x;
        int32_t y;
};
```





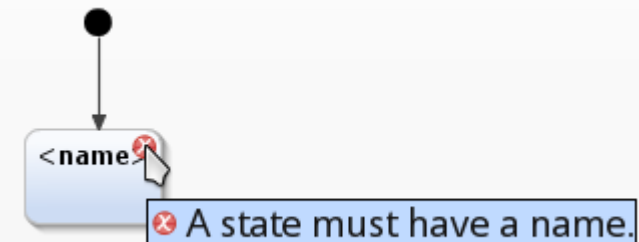
# 1. M2M Example

- M2M: Model validation: error pattern → error message



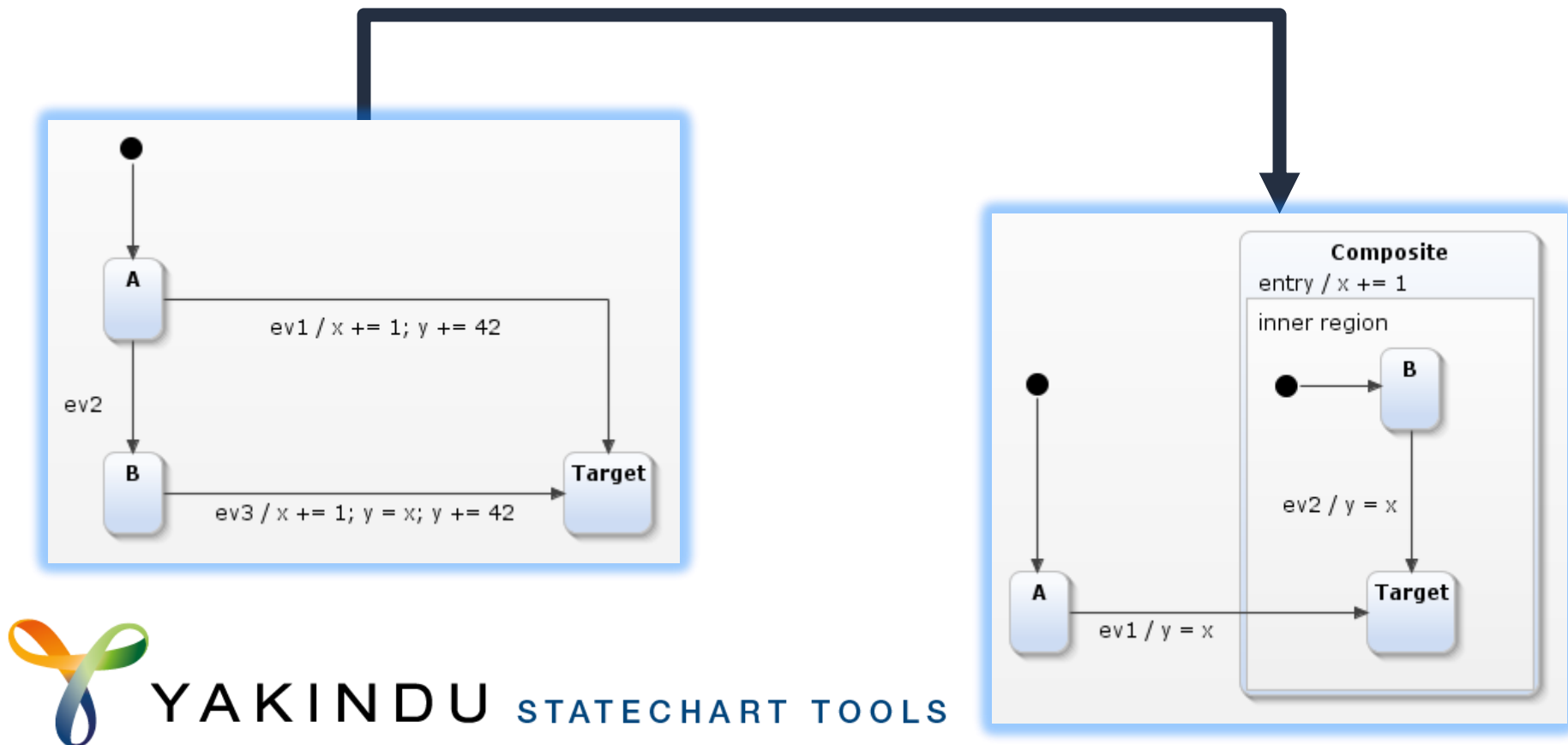
**Compare this with the OCL:**

- No error message
- Bound to a single object
- Positively worded (when correct vs when incorrect)
- When to run
- Precise semantics
- Performance



## 2. M2M Example

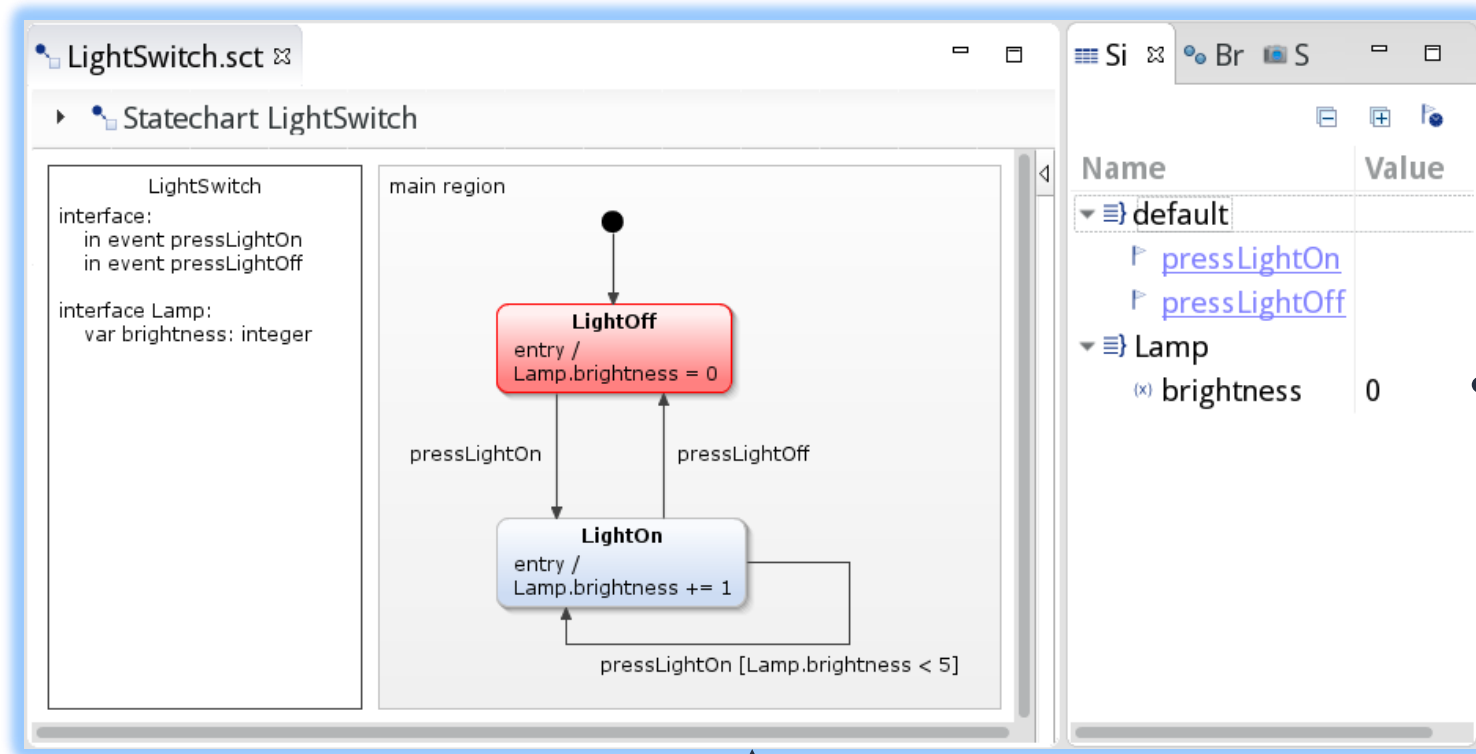
- Model refactoring





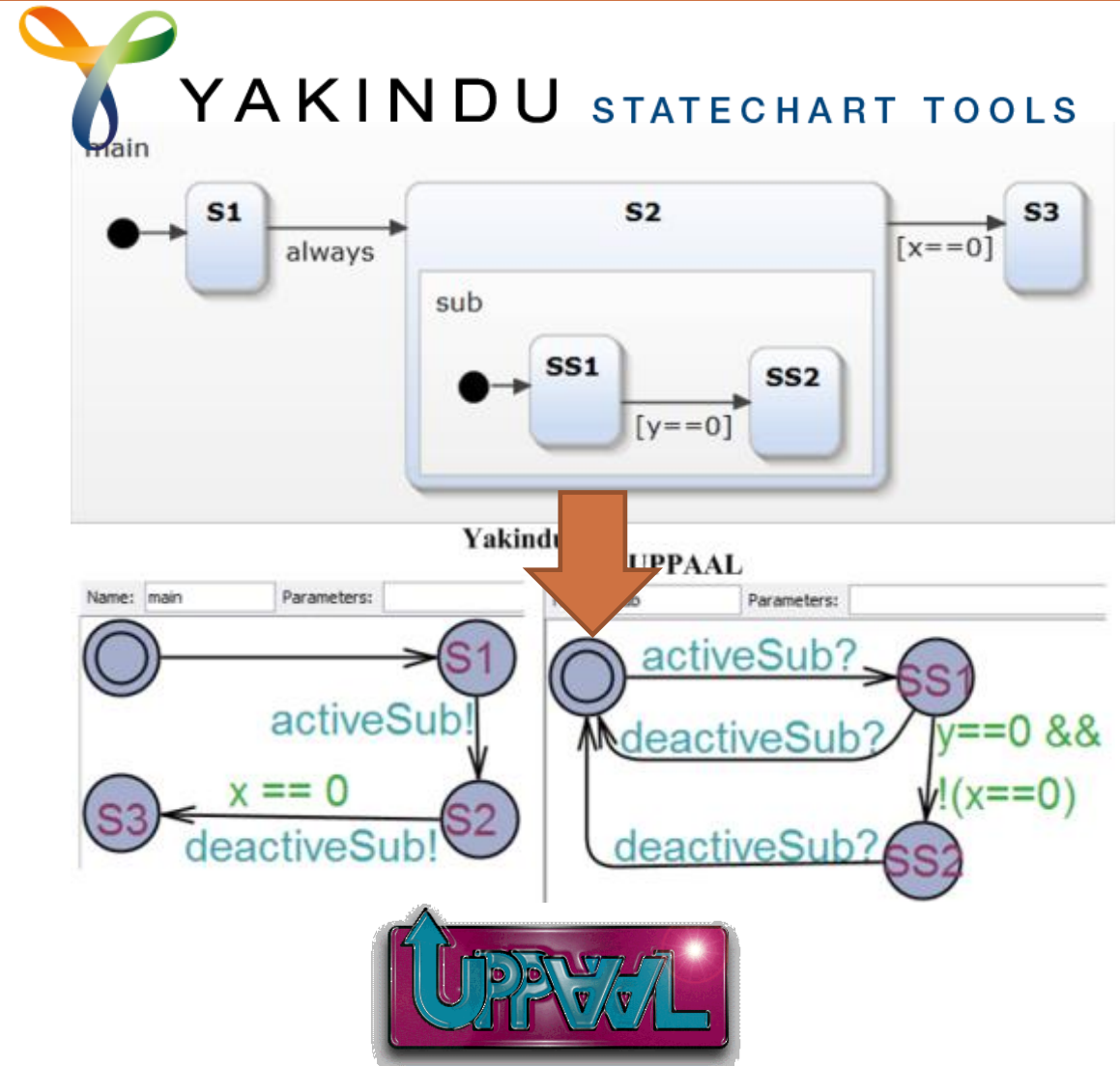
### 3. M2M Example

- Simulation
- Semantics

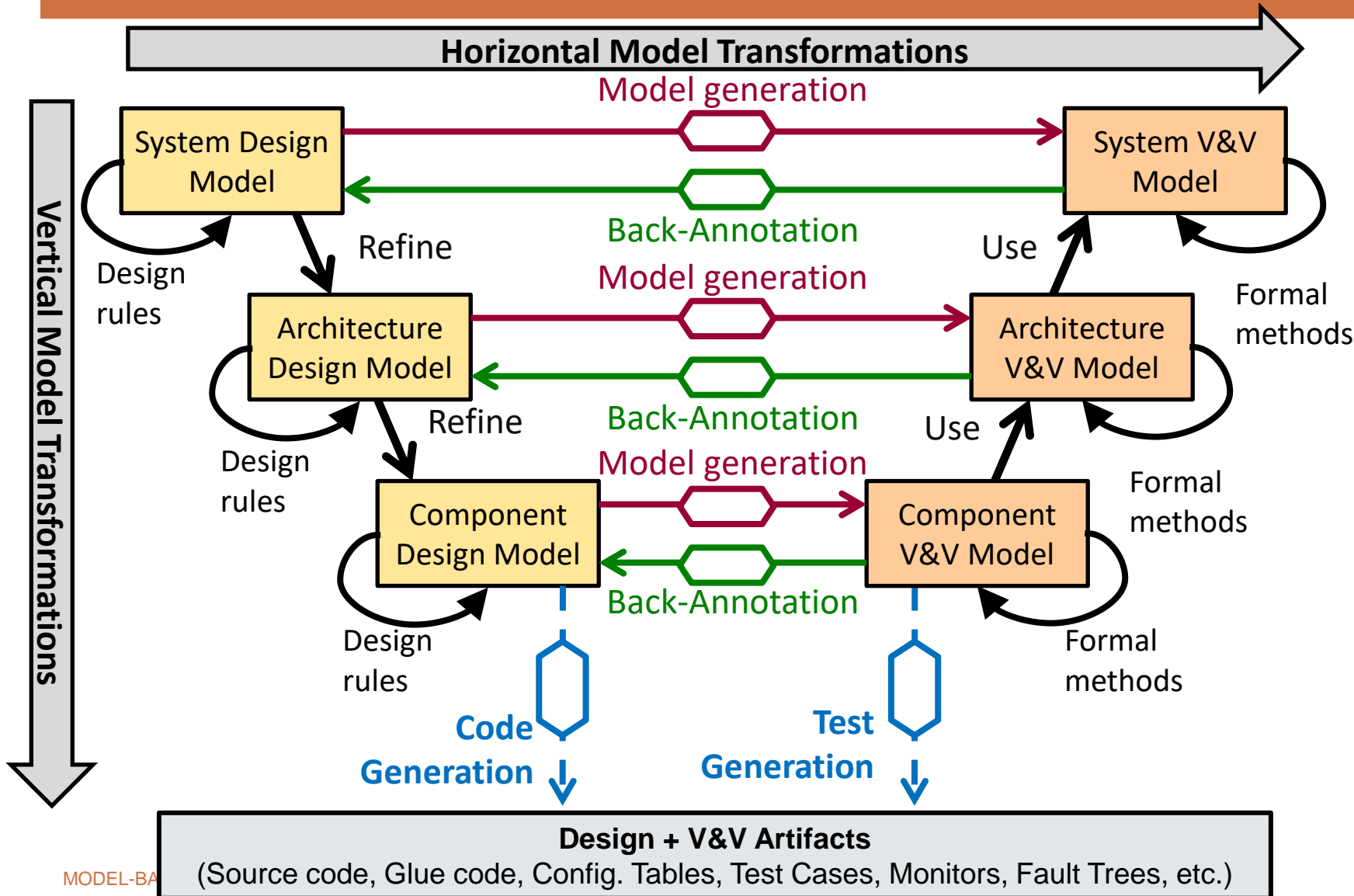


## 4. M2M Example

- Applying formal methods
- Hidden formal methods:  
Algorithms that can be applied without special expertise
  - > Tool support
  - > Projection of results



# Models and Transformations in Critical Systems



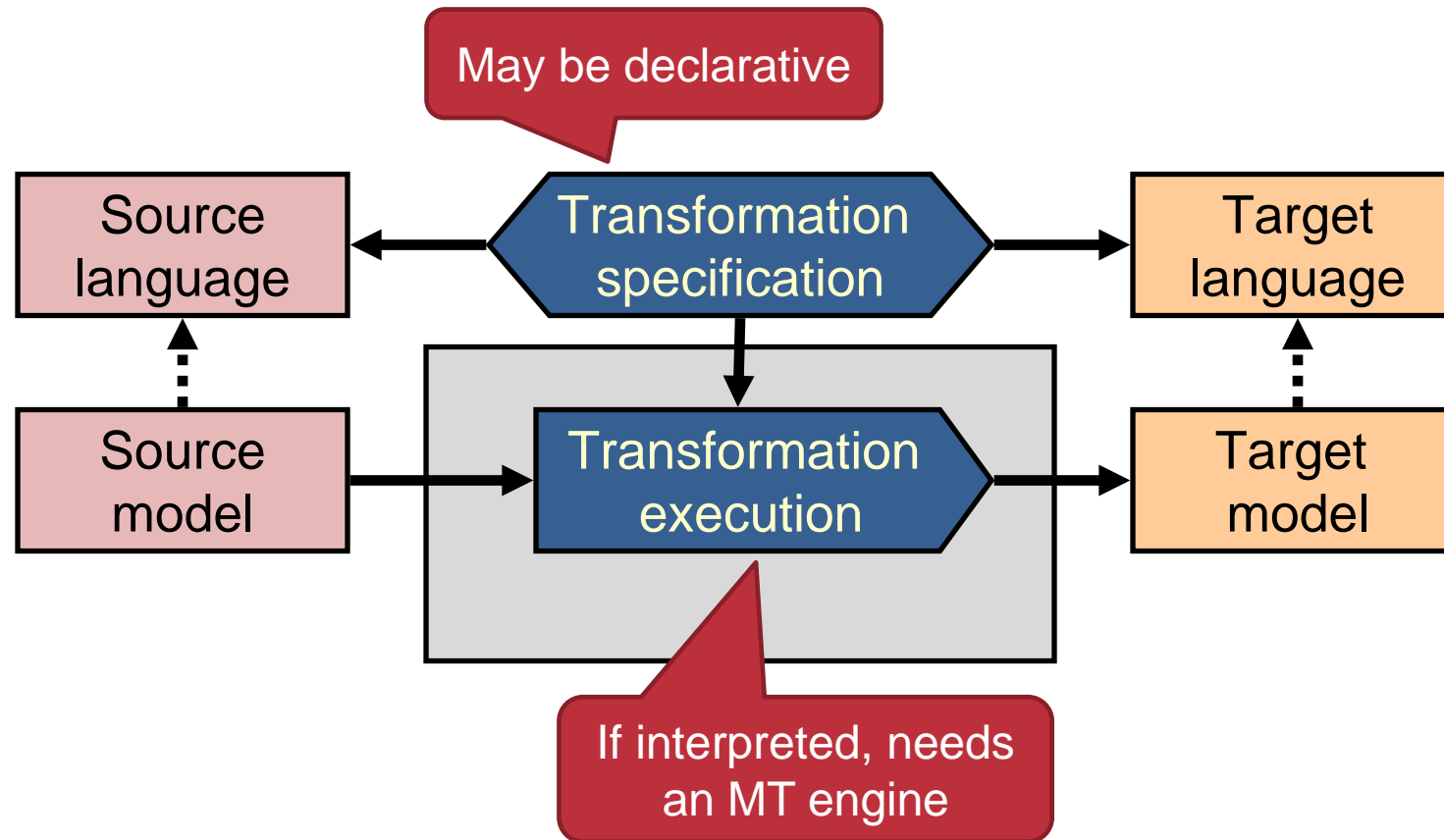
## Model Transformations:

- systematic foundation of knowledge transfer:

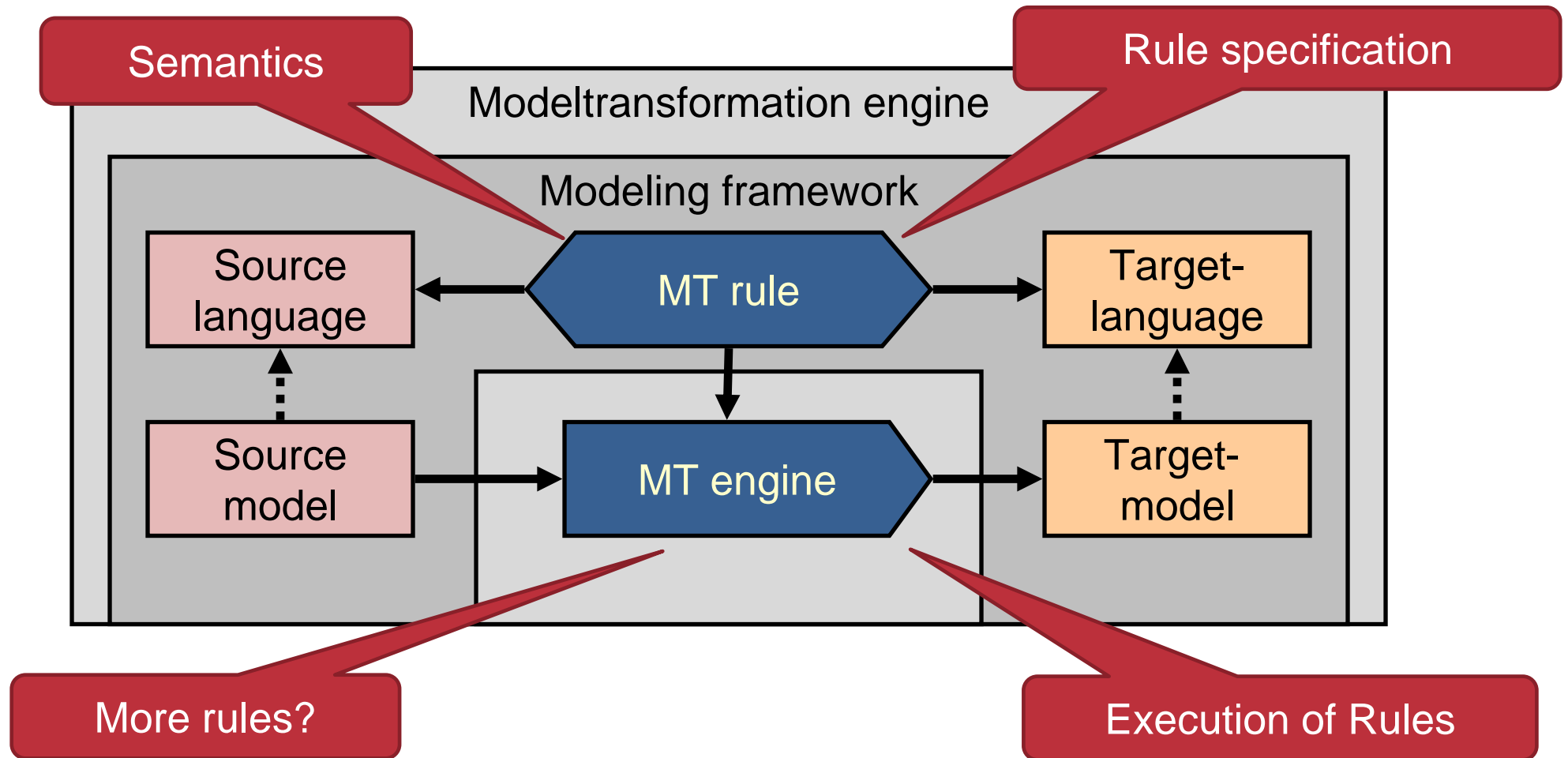
*Theoretical results* → *tools*

- bridge / integrate existing languages&tools

# Definition of Model Transformation

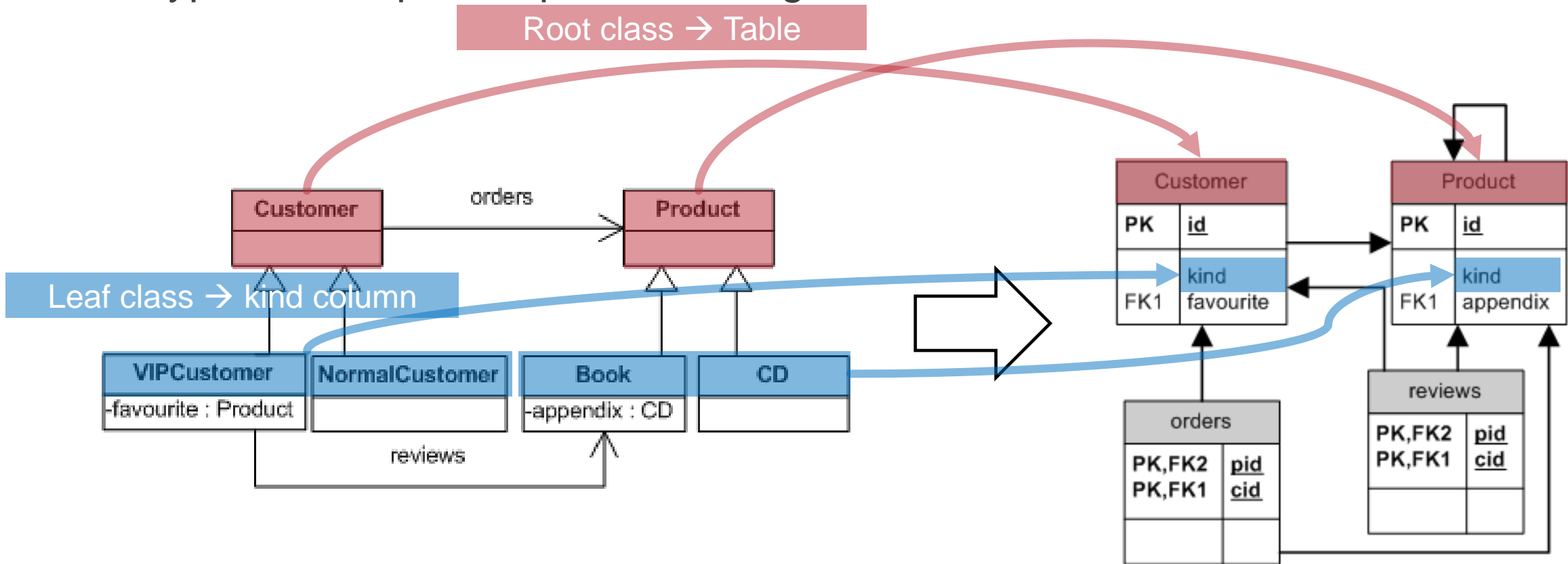


# Defintions and Questions



# Example: Object-relational mapping

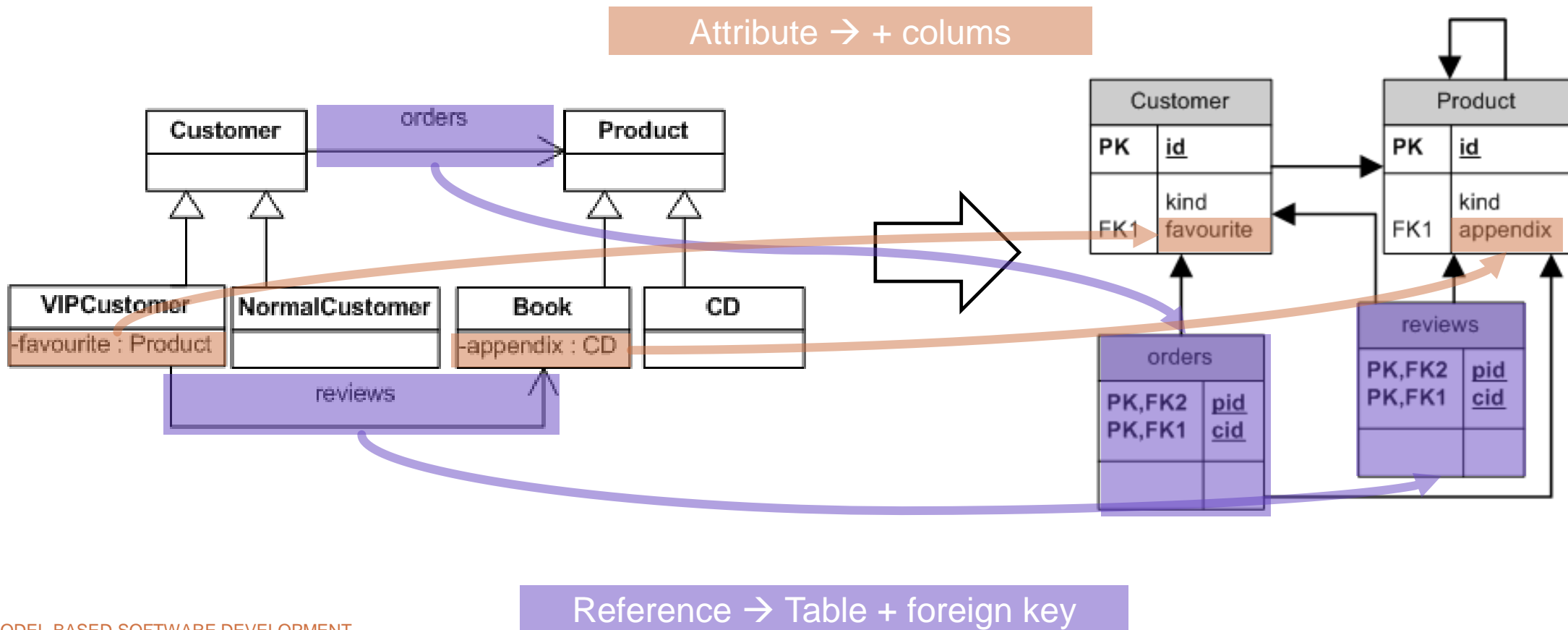
- Typical example: map a class diagram to database tables!





# Example: Object-relational mapping

- Typical example: map a class diagram to database tables!

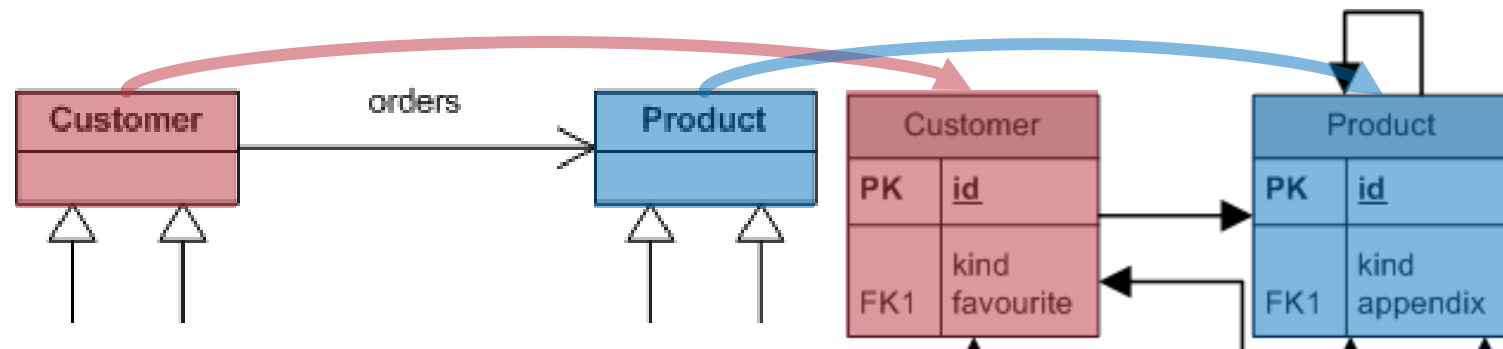
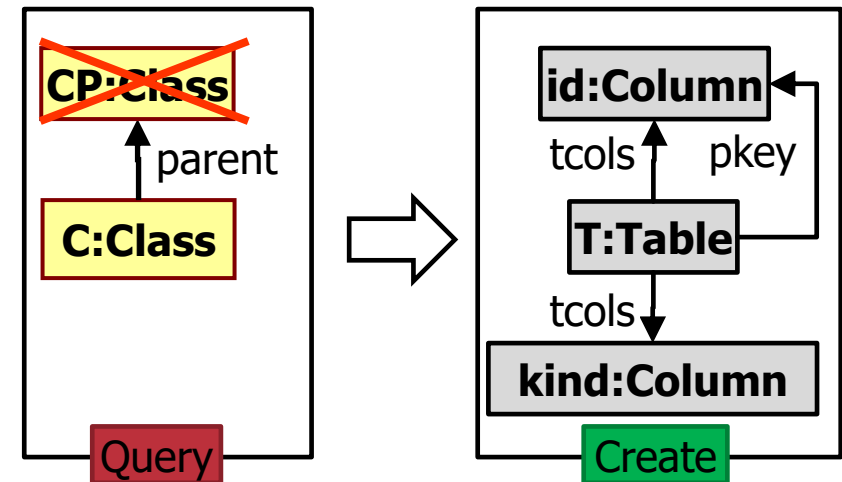


# Example Transformation

- How would we solve the problem of creating tables representing root classes?

1. Query the root classes (class that has no ancestor)
2. Create the tables and with them the necessary columns
3. Repeat as long as we can

- Goal: To formulate the whole transformation with similar rules



# Model transformation

**Definitions**

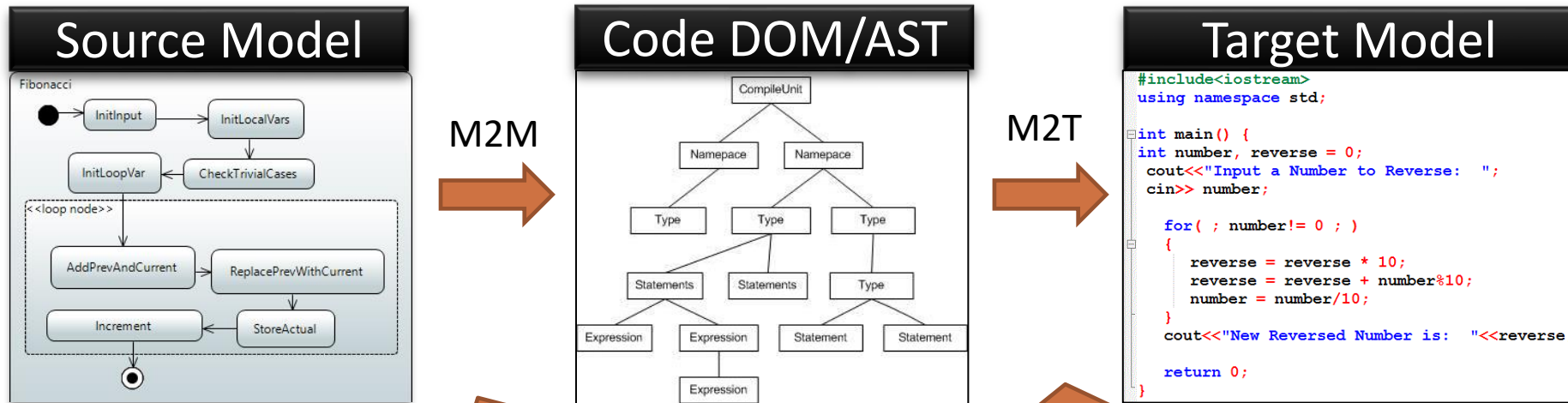
**Model Transformation Chains**

**Rule-based model transformations**

**Technologies**



# Code Generation by Model Transformations



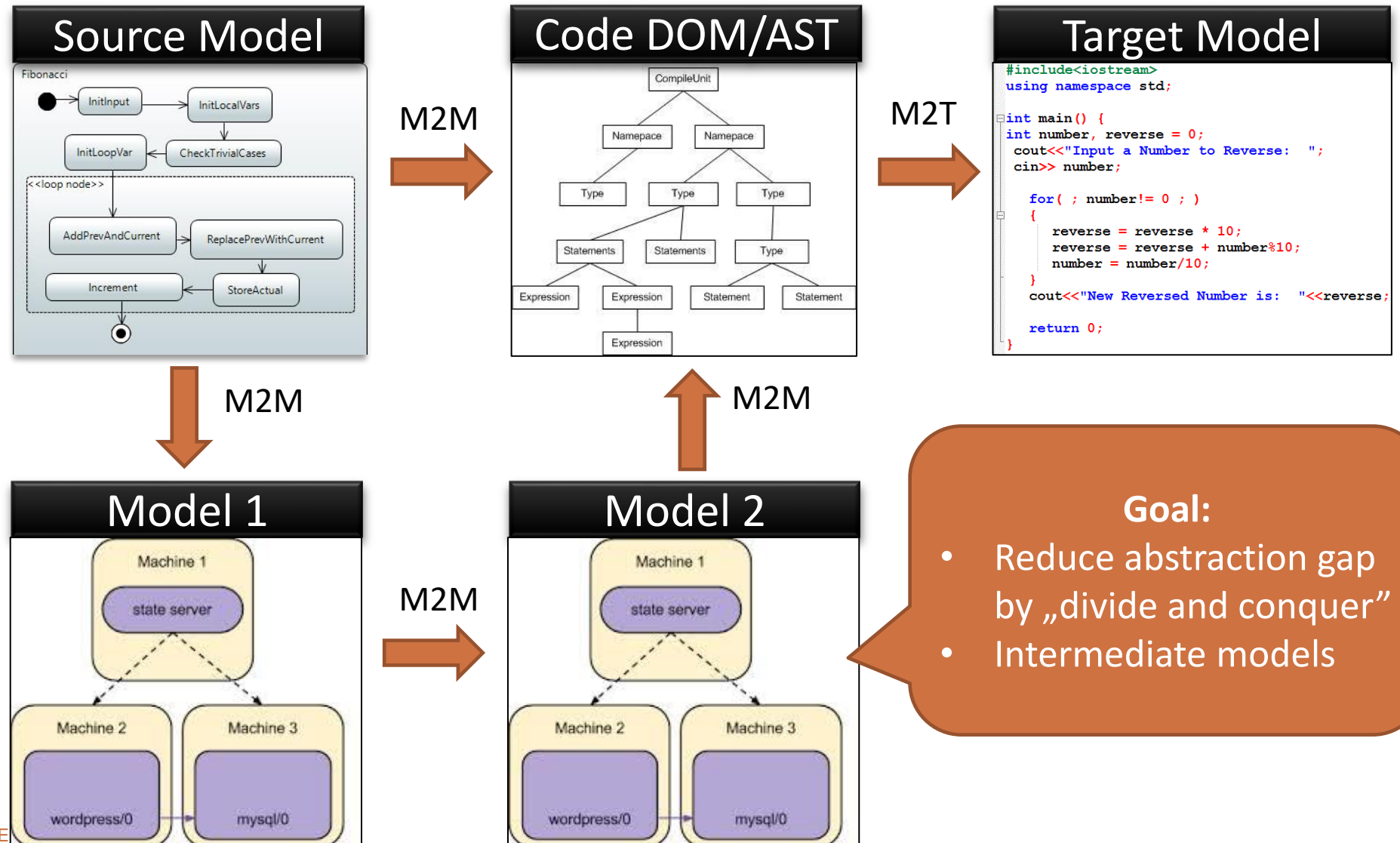
## Model-to-Model (M2M)

- SRC: In-memory model (objects)
- TRG: In-memory model (objects)

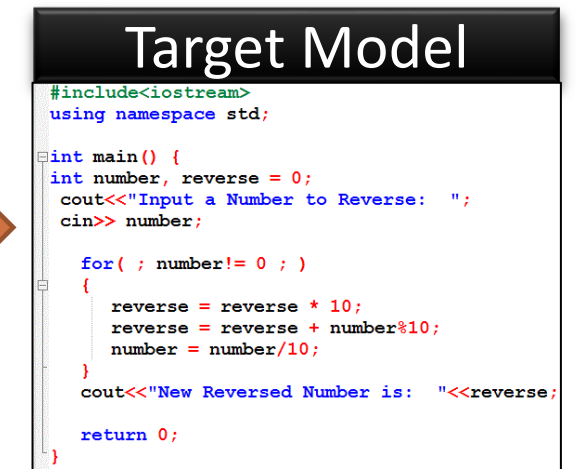
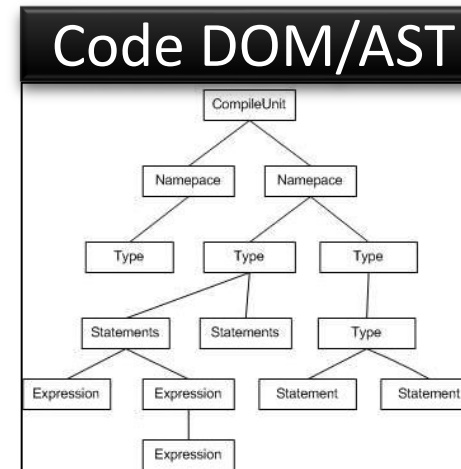
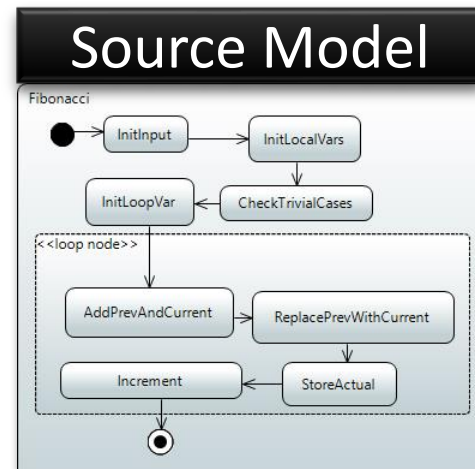
## Model-to-Text (M2T)

- SRC: In-memory model (objects)
- TRG: Textual code (string)

# Model Transformation Flows / Chains



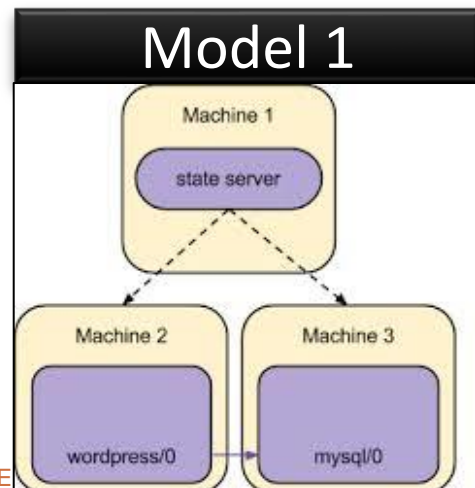
# Model Transformation Flows / Chains



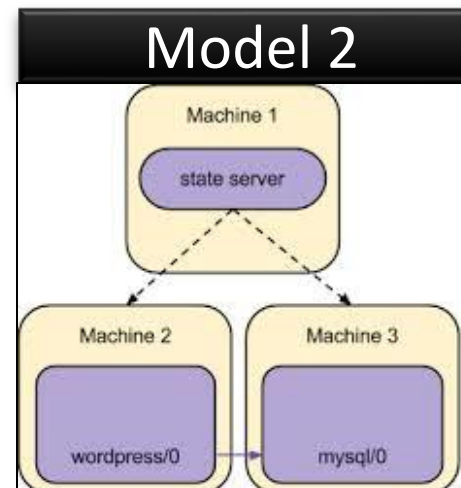
M2M

M2M

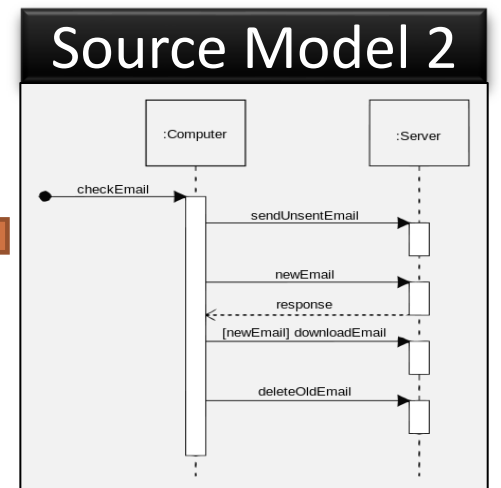
Joint optimization steps



M2M

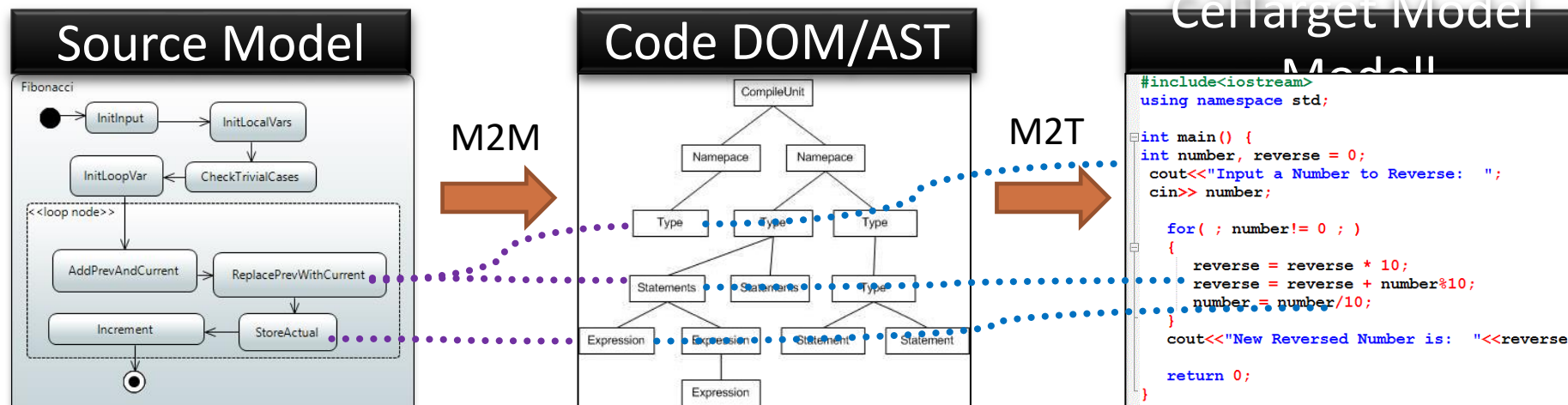


M2M



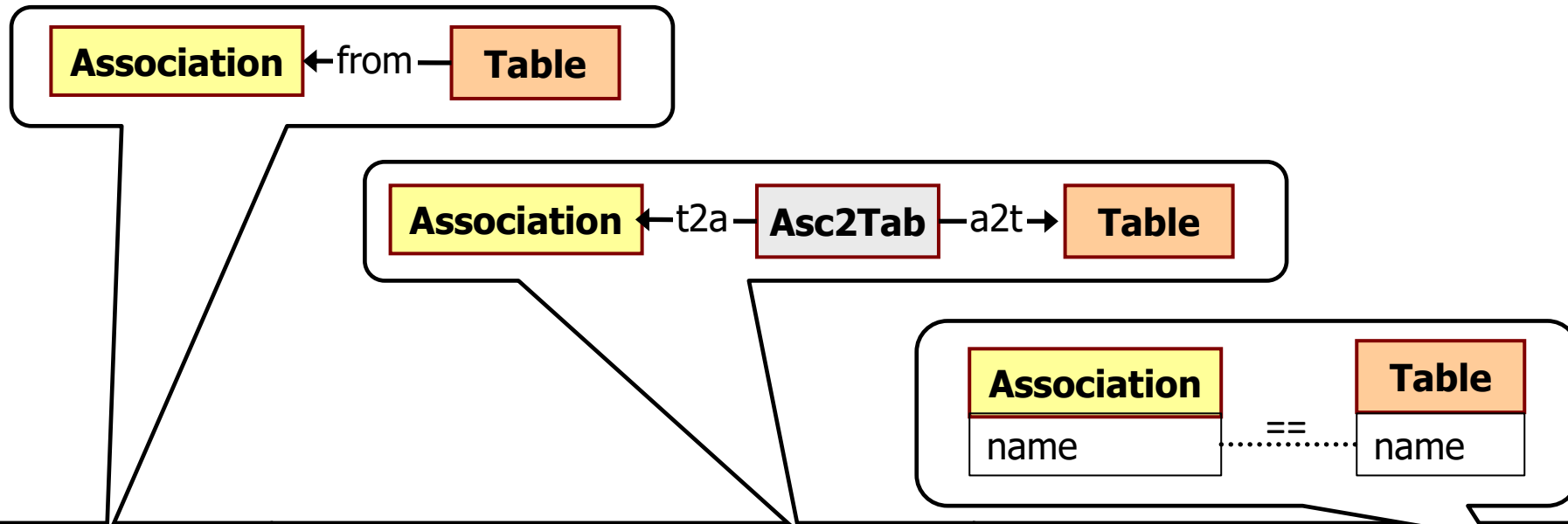


# Traceability in Model Transformations



- **Traceability / correspondence links:** Connect source and target models
- Make transformation specification easier
- Support end-to-end traceability
- Make incrementality possible

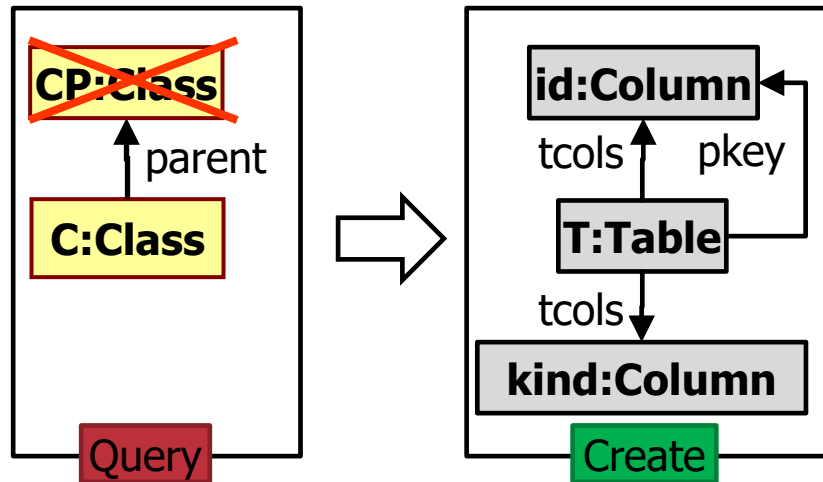
# Forms of Traceability



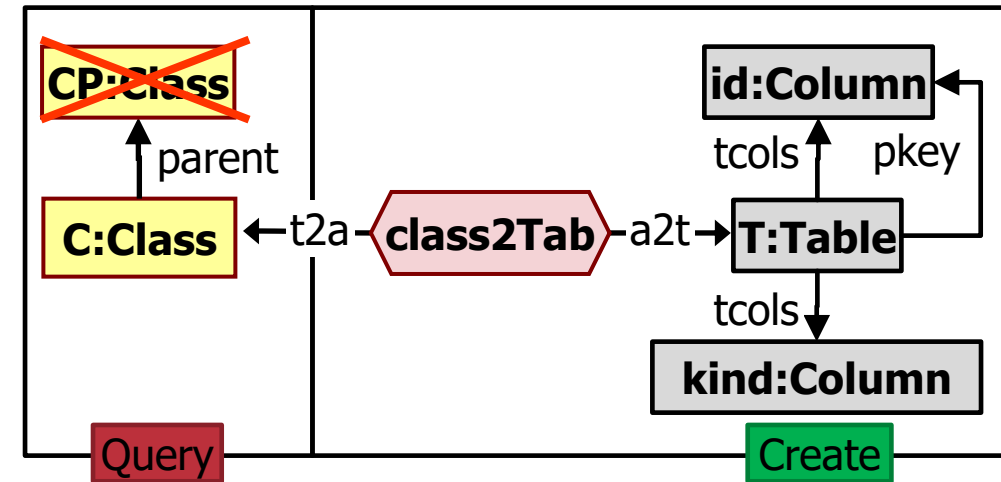
Direct links	Correspondence model	Soft links
Cross-reference between SRC↔TRG	Stored in separate metamodel & model	Match by id / qualified name using model query / index
Intrusive: must extend meta & instance models	Complex, large overhead	Requires unique identifier; limited expressiveness (tells which one, not why)

# Example: ORM

- How to connect the generated tables with the classes using a traceability model?



How do we know  
a class is done?



# Model transformation

**Definitions**

**Model Transformation Chains**

**Rule-based model transformations**

**Technologies**



# Model Transformation Specification

## ■ Imperative with direct model manipulation

- > Quick and easy to start
- > But what if we need something complicated?
  - More complex rules, new cases?
  - Incrementality?
  - Bidirectionality?

## ■ Rule-based declarative

- > *Graph Transformation* based
- > Hybrid: query + imperative action (VIATRA etc.)
- > „Relational” (QVT-R, TGG, ATL, etc.)

# Rule-based model transformations

- Unit: **MT rule**

For each occurrence of...	...transform it like this
Root class in inheritance hierarchy	Create entity table with default columns
Attribute of class	Add columns to table of class
Association between classes	Create switch with foreign key columns
<b>PRECONDITION</b> Declarative Model Query	<b>ACTION</b> May be imperative



# Rule-based Systems

Where have I seen rule-based systems?

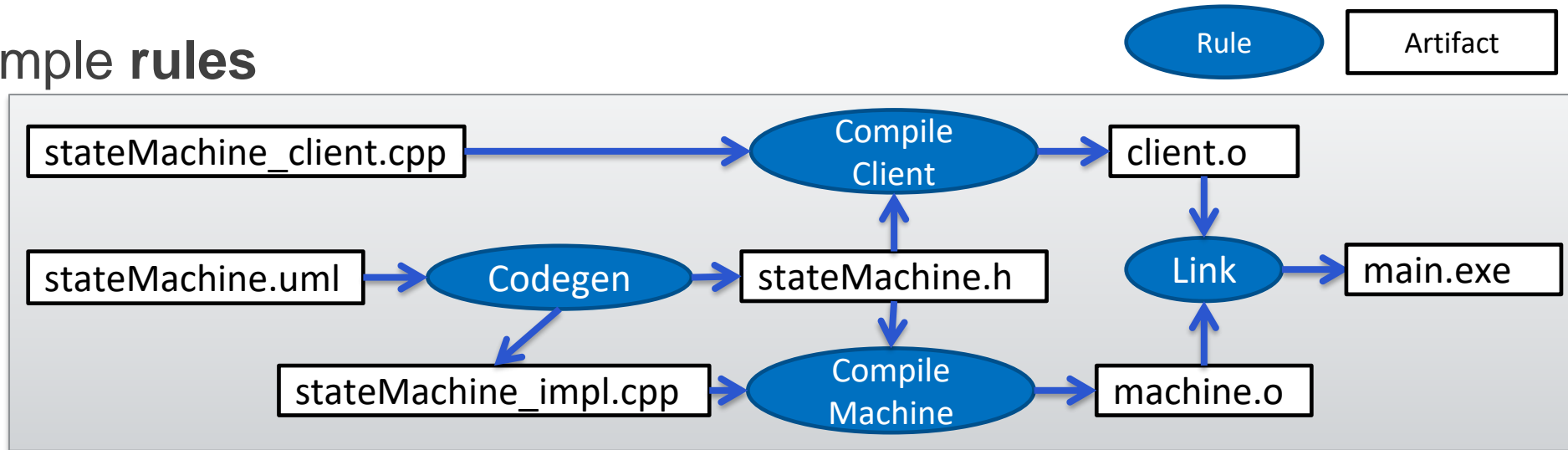
- **Model transformations**
- Build scripts (MAKEFILE, Maven, gradle, etc.)
  - > Rule: build this artifact based on the specification (*akció*)
  - > When all necessary artifacts are ready and dirty (precondition)
- Business rule & expert systems (Jboss Drools, etc.)
- Context-free grammars (see Textual Syntax Lecture)
- CSS

# Inversion of Control (IoC)

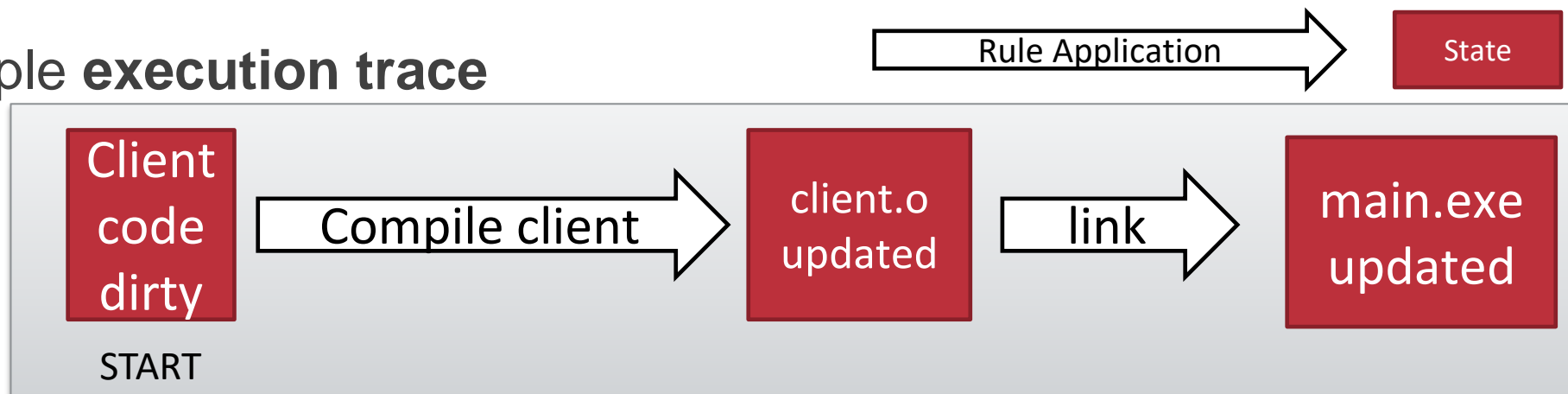
- Declarative rule execution
  - > **Transformation engine** interprets preconditions
  - > Rules are **fired** by engine when&where enabled
- Several variants
  - > „As long as possible” / „fire why possible” semantics
    - Iterate while there are **rule activations**
    - Select one activation (**conflict resolution**)
    - fire it
  - > „Fire all current” semantics
    - Select all *current* activations
    - fire them all
  - > Arbitrary control flow

# Build Script Example

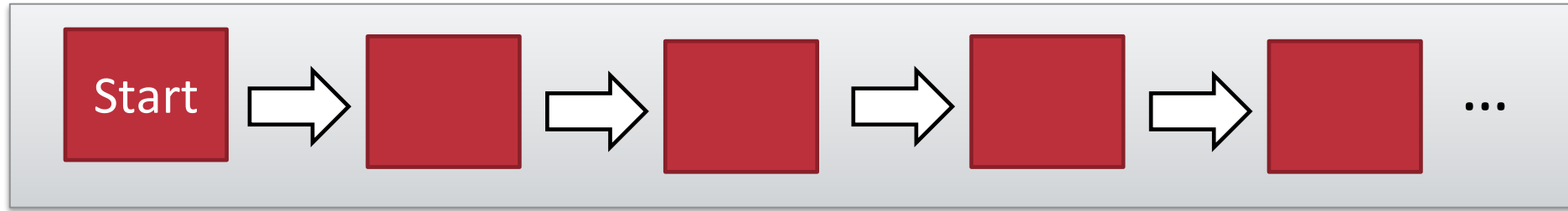
## ■ Example rules



## ■ Example execution trace



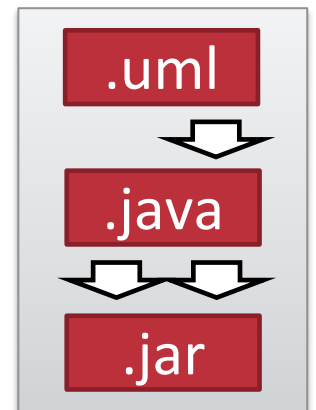
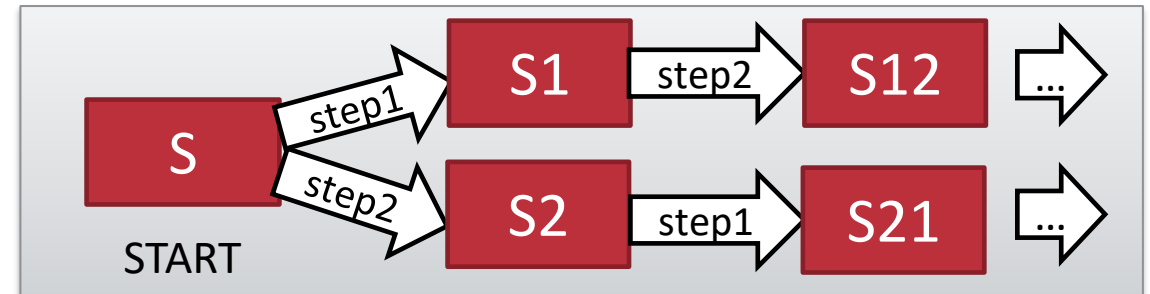
# Common Rule-based Problems: Termination



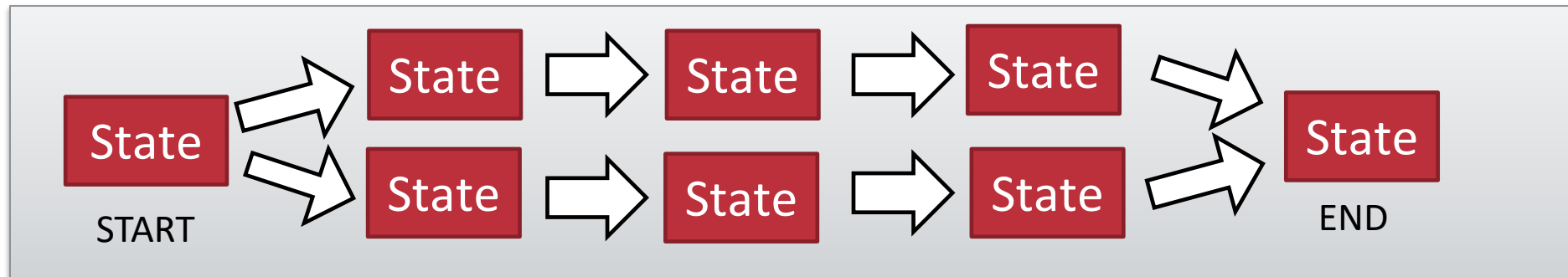
- Vital to ensure termination!
- Non-terminating examples
  - > Makefile: a build step overwrites (re-dirties) one of its inputs
  - > MT rule creates new object, has to be xformed by same rule
  - > MT Rule1 creates element, Rule2 deletes it, Rule 1 again, ...
- No systematic way to guarantee, requires thought

# Common Rule-based Problems: Ordering of steps

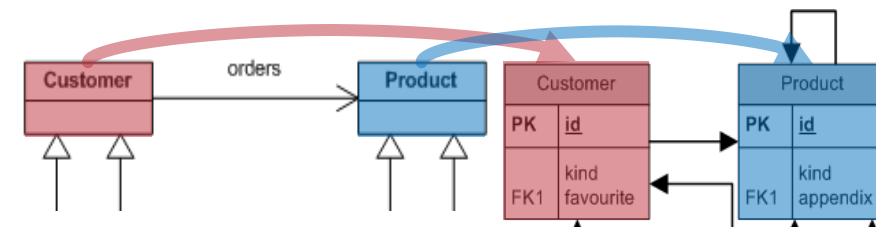
- May be required for correctness
  - > transform class →
  - > transform attribute
- In other cases, only performance is impacted
  - > Makefile: building dirty .java source code,
  - > Then build the dirty .uml artifact, which made the binary dirty again
  - > Need to build the .java artifact again
- How to manage?
  - > Smart engine (limited applicability, works for Makefile)
  - > Express ordering in precondition
  - > **Rule priorities:** assign priority to rules: low priority → high priority



# Common Rule-based Problems: Confluence



- Final state must be determined by start state
  - > No matter the internal choices (which rule to apply now?)
  - > Confluence is important; full determinism is optional
- Examples
  - > Makefile: Which dirty file to recompile first? Doesn't matter...
  - > Which root class to transform first? Doesn't matter...
- No systematic way to guarantee, requires thought





# Model transformation

**Definitions**

**Model Transformation Chains**

**Rule-based model transformations**

**Technologies**



# What technologies are available?

- **Imperative solution:** traverse the model, modify, save.
- **Template-based:** typically to generate source code or other text output
  - > Given a skeleton of text output, missing parts are filled in
  - > We will now review these
- **Graph transformation based**
  - > Next lecture, Practice

# XSLT

- EXtensible Stylesheet Language Transformations
- Processing, by matching templates
- Transformation of XML documents
  - > Declarative semantics (XML)
  - > XML or other arbitrary text output
  - > Navigation with XPath

# Example code

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<?xml-stylesheet type="text/xsl" href="example.xsl"?>
<Styles>
  <Red type="color">
    <Reddish type="color">This is reddish</Reddish>
    <Burgundy type="color">This is burgundy</Burgundy>
  </Red>
  <Blue type="color">This is blue</Blue>
  <Italic type="font">This is italic</Italic>
</Styles>
```

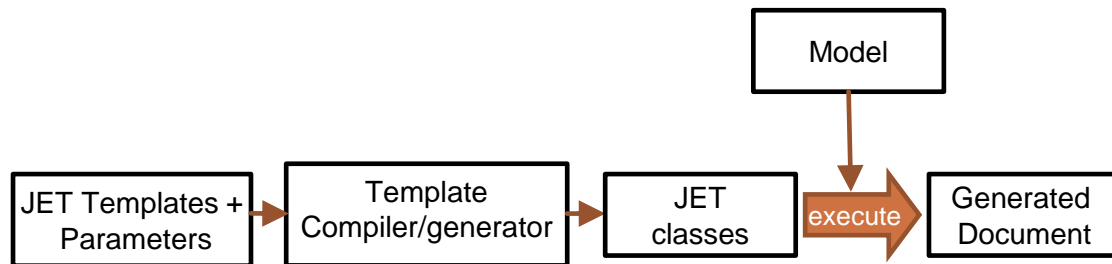
```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="Reddish">
    <font Color="#FFAA00"><xsl:value-of select="."/></font>
  </xsl:template>
  <xsl:template match="Burgundy">
    <font Color="#900000"><xsl:value-of select="."/></font>
  </xsl:template>
  <xsl:template match="Blue">
    <font Color="#0000A0"><xsl:value-of select="."/></font>
  </xsl:template>
  <xsl:template match="Italic">
    <i><xsl:value-of select="."/></i>
  </xsl:template>
</xsl:stylesheet>
```

<font Color="#FFAA00">This is reddish</font>  
<font Color="#900000">This is burgundy</font>  
<font Color="#0000A0">This is blue</font>  
<i>This is italic</i>

# Two more XML syntax

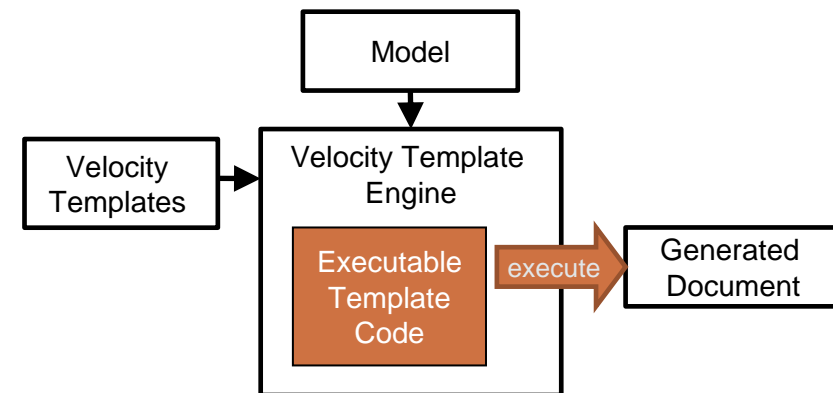
## Java Emitter Templates

- JSP-like language
- Translates to Java
- Input: java objects
- Output: text
- Part of EMF, EMF code generator



## Apache Velocity

- JSP-like language
- Interpreted
- Input: Map
- Output: text



# Example codes, <% %> vs. #( )

## Java Emitter Templates

```
<%@ jet package="hello"
imports="java.util.*" class="XMLDemoTemplate" %>
<% List elementList = (List) argument; %>
<?xml version="1.0" encoding="UTF-8"?>
<demo>
<% for
    (Iterator i = elementList.iterator();
        i.hasNext(); ) { %>
<element><%=i.next().toString()%></element>
<% } %>
</demo>
```

## Apache Velocity

```
<?xml version="1.0" encoding="UTF-8"?>
<demo>

#set( $tempString = "Element")
#foreach( $element in $elementList)
    <element> ${element.toString()} <element>
#end
</demo>
```

### Example output

```
<demo>
  <element>A</element>
  <element>B</element>
</demo>
```

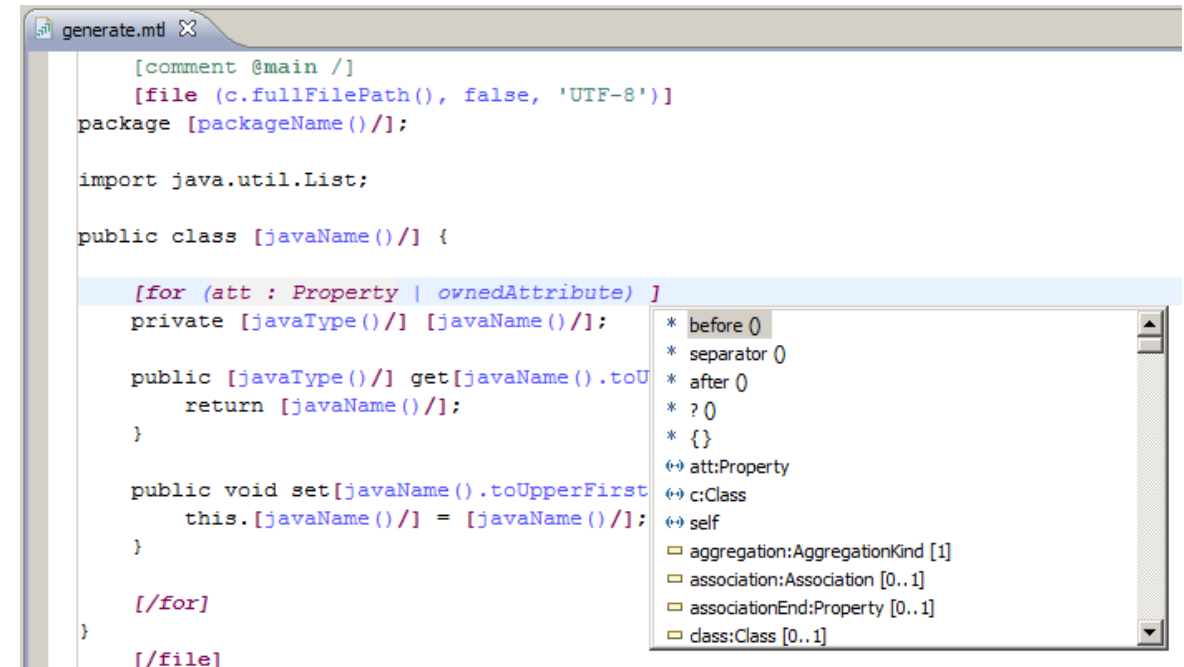
Template

Output

# Acceleo



- Code generator for EMF models
- OMG Model to Text Language (MOFM2T) implementation
- Eclipse based, stable development tool
- Consists of modules
  - > Import possible
- Language elements
  - > Templates
  - > Queries
  - > Cycle, branching, value assignment, ...

A screenshot of the Acceleo generate.mtl editor. The main text area shows a template for generating Java code. It includes a comment, a file declaration, a package declaration, an import, and a class definition. A loop is used to generate getters and setters for properties. The right-hand side of the editor shows a list of available language elements for the loop, including 'before', 'separator', 'after', '?', '{', 'att:Property', 'c:Class', 'self', 'aggregation:AggregationKind', 'association:Association', 'associationEnd:Property', and 'class:Class'.

```
[comment @main /]  
[file (c.fullFilePath(), false, 'UTF-8')]  
package [packageName()];  
  
import java.util.List;  
  
public class [javaName()] {  
  
    [for (att : Property | ownedAttribute) ]  
    private [javaType()] [javaName()];  
  
    public [javaType()] get[javaName().toUpperFirst]  
        return [javaName()];  
    }  
  
    public void set[javaName().toUpperFirst]  
        this.[javaName()] = [javaName()];  
    }  
  
    [//for]  
}  
  
[//file]
```

# Some Acceleo specifics

- Templates

```
[template public generate(c : Class)]  
  
    [comment @main /]  
    [file (c.name, false, 'UTF-8')]  
    [c.name/]  
    [/file]  
  
[/template]
```

- Preconditions (when to run the template? → Functional languages?)

```
[comment Generates the java code for a class property that belongs to an association and is ordered /]  
[template public genAssociation(p : Property) ? (owningAssociation <> null and isOrdered)]  
  
[/template]
```

- OCL Queries:

```
[query public getPublicAttributes(c : Class) : Set(Property) =  
    c.attribute->select(visibility = VisibilityKind::public)  
/]
```



# Acceleo summary

## ■ Pros

- > Effective for EMF models? → See next practice!
- > There is debug capability!

## ■ Cons

- > Java only, EMF only
- > New language to learn
  - Not difficult to learn
  - OCL was difficult to use

# Xtend

- General purpose programming language (modern java before Kotlin)
- Object oriented
- Transparent interoperability with Java
  - > Static type checking
  - > Java type system
  - > Compiles to Java code
  - > Reference back and forth



# Xtend language elements

```
import com.google.inject.Inject
```

Java interop

```
class DomainmodelGenerator implements IGenerator {
```

```
@Inject extension IQualifiedNameProvider nameProvider
```

There is no ;

```
    override void doGenerate(Resource resource, IFileSystemAccess fsa) {  
        for(e: resource.allContentsIterable.filter(typeof(Entity))) {  
            fsa.generateFile(  
                e.fullyQualifiedName.toString.replace(".", "/") + ".java",  
                e.compile)
```

Type inference

First parameter can be omitted

```
        }  
    }  
    def compile(Entity e) '''
```

''' = template

```
        «IF e.eContainer != null»
```

```
        package «e.eContainer.fullyQualifiedName»;
```

```
        «ENDIF»
```

```
        public class «e.name»
```

String interpolation

```
        «IF e.superType != null» extends «e.superType.shortName» «ENDIF»
```

```
    {
```

```
        «FOR f:e.features»
```

```
        «f.compile»
```

```
        «ENDFOR»
```

Control structures in templates

```
    }
```

```
...
```

Grey Space:  
template space vs code space

# Xtend summary

## ■ Pros

- > Easy to learn, productive coding
- > High expressiveness (complex code can be written in short)
- > Java compatible

## ■ Cons

- > Only Java supported
- > Automatic build?
- > Eclipse-based (but can now run separately)

# Microsoft T4

- **Text Templating Transformation Toolkit**
- Text blocks and control logic in one file
  - > Text block copied to output
  - > C# or VB
    - Can write to output
  - > Similar to ASP.NET, PHP, ...
- Used in: DSL Tools, Entity Framework, VMTS...

# Control blocks

- Code block: <# ... #>
- Expression block: <#= ... #>
  - > Can be evaluated
- Square of numbers:

```
<#@ template language="C#" #>
<#int top = 10;
    for (int i = 0; i<=top; i++) { #>
        The square of <#= i #> is <#= i*i #>
    } #>
```

```
The square of 0 is 0
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
...
```

# How does it work?

- It is generated (and runs):

```
<#@ template language="C#" #>
<#int top = 10;
    for (int i = 0; i<=top; i++) { #>
        The square of <#= i #> is <#= i*i #>
    } #>
```

```
public partial class MyTemplate : ... {
    public string TransformText() {
        int top = 10;
        for (int i = 0; i<=top; i++) {
            this.Write("The square of ");
            this.Write(this.ToStringHelper.ToStringWithCulture(i));
            this.Write(" is ");
            this.Write(this.ToStringHelper.ToStringWithCulture(i*i));
            this.Write("\r\n");
        }
        return this.GenerationEnvironment.ToString(); }}
}
```

# Extension of the class

- `<#+ ... #>`

- > Generate helper methods and property

```
<#+ // Class feature block
private void WriteSquareLine(int i) { #>
    The square of <#= i #> is <#= i*i #>.
<# } #>
```

- > May also include text block

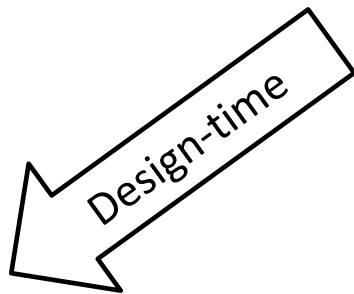
- > Added to the generated class, can be called:

```
<#int top = 10;
    for (int i = 0; i<=top; i++)
        WriteSquareLine(i);
#>
```

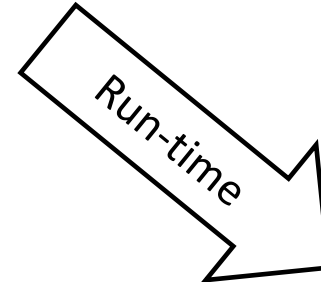


# Design-time vs run-time

```
<#int top = 5;  
    for (int i = 0; i<=top; i++) { #>  
        The square of <#= i #> is <#= i*i #>  
<# } #>
```



The square of 0 is 0  
The square of 1 is 1  
The square of 2 is 4  
The square of 3 is 9  
The square of 4 is 16  
The square of 5 is 25



```
public virtual string TransformText() {  
    int top = 5;  
    for (int i = 0; i <= top; i++) {  
        this.Write(" \r\n      The square of ");  
        this.Write(this.ToStringHelper.ToStringWithCulture(i));  
        this.Write(" is ");  
        this.Write(this.ToStringHelper.ToStringWithCulture(i * i));  
        this.Write(" \r\n");    }  
}
```

# T4 Summary

## ■ Pros

- > Easy to use, flexible
- > Excellent for task automation  
(100+ thousand lines of code)
- > Can be used to generate fast, binary code

## ■ Cons

- > Maintainability of T4 scripts
- > Debug options
- > Strange formatting requirements
- > Automatic build? Dependencies?



Thank you for your attention