CSSE 490 Model-Based Software Engineering: Even More on Domain Specific Languages ☺

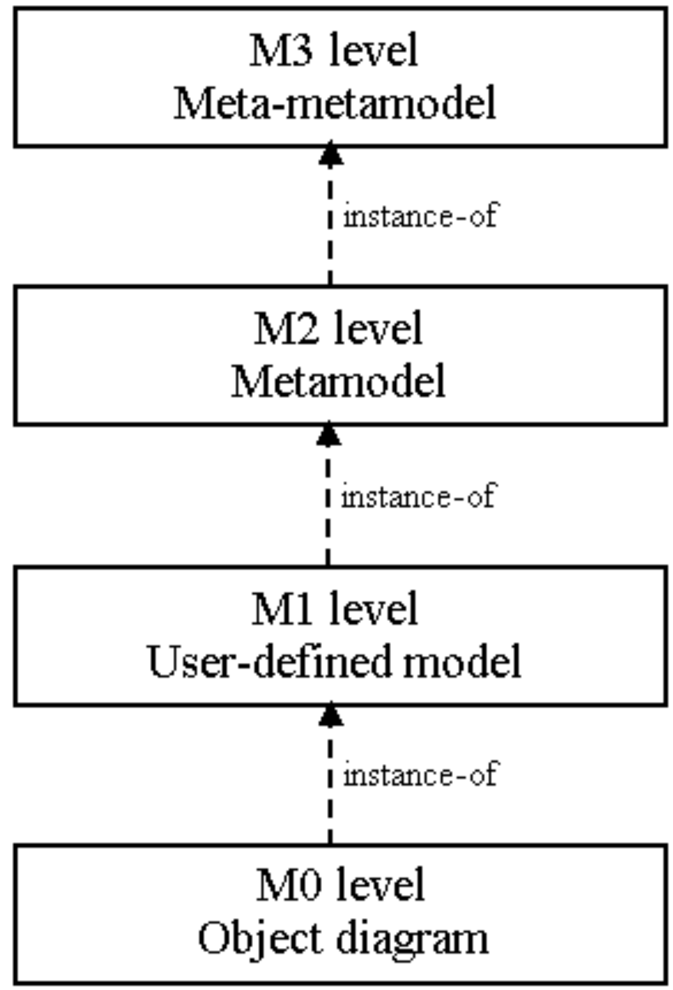
Shawn Bohner
Office: Moench Room F212
Phone: (812) 877-8685
Email: bohner@rose-hulman.edu

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Learning Outcomes: Metamodels

Design a metamodel for a model-based software system.

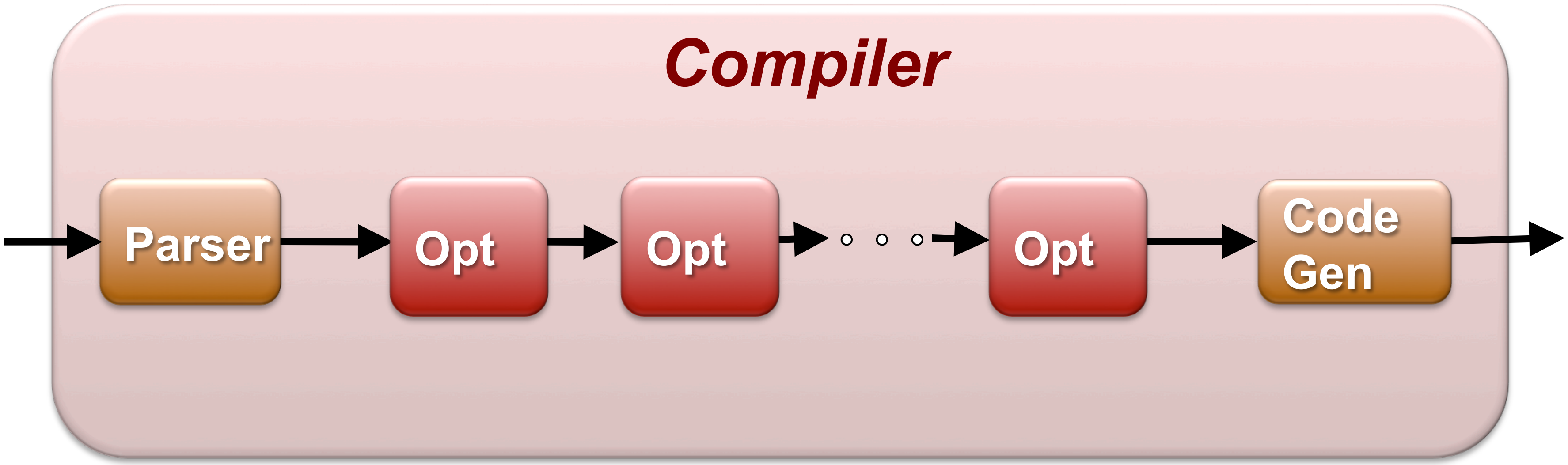
- **Contrast DSLs with compilers**
- **Examine Benefits and Risks of DSL Approaches**
- **Introduce Eclipse Modeling Framework (EMF)**



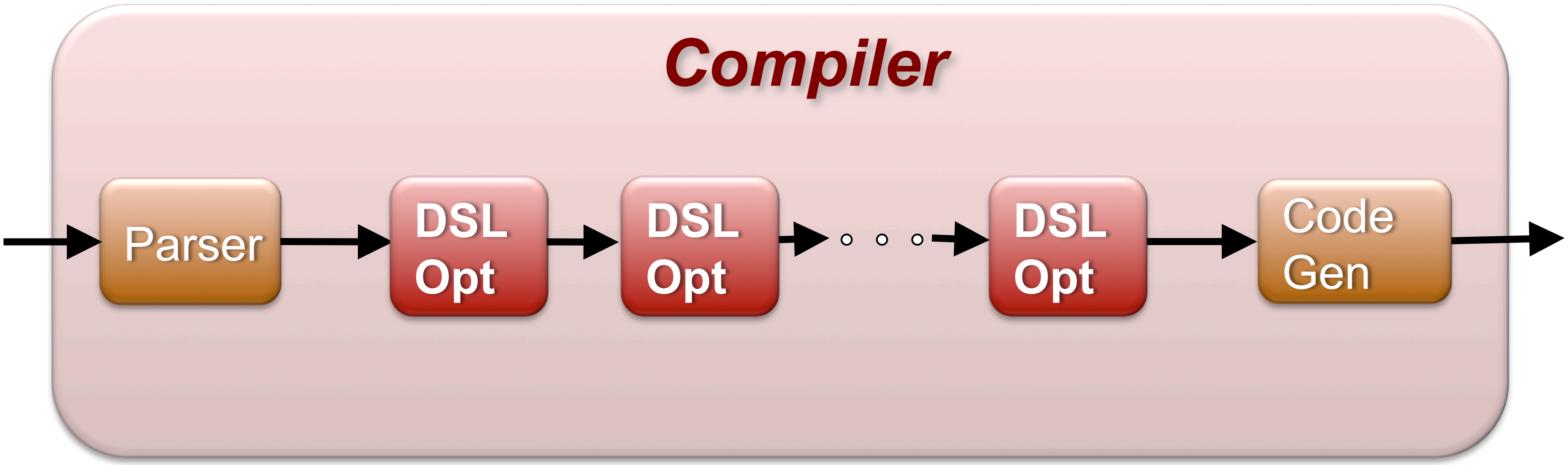
"GPL" is to a "Compiler" as DSL is to a(n) ________?

- Again, think for 15 seconds…
- Let's talk…

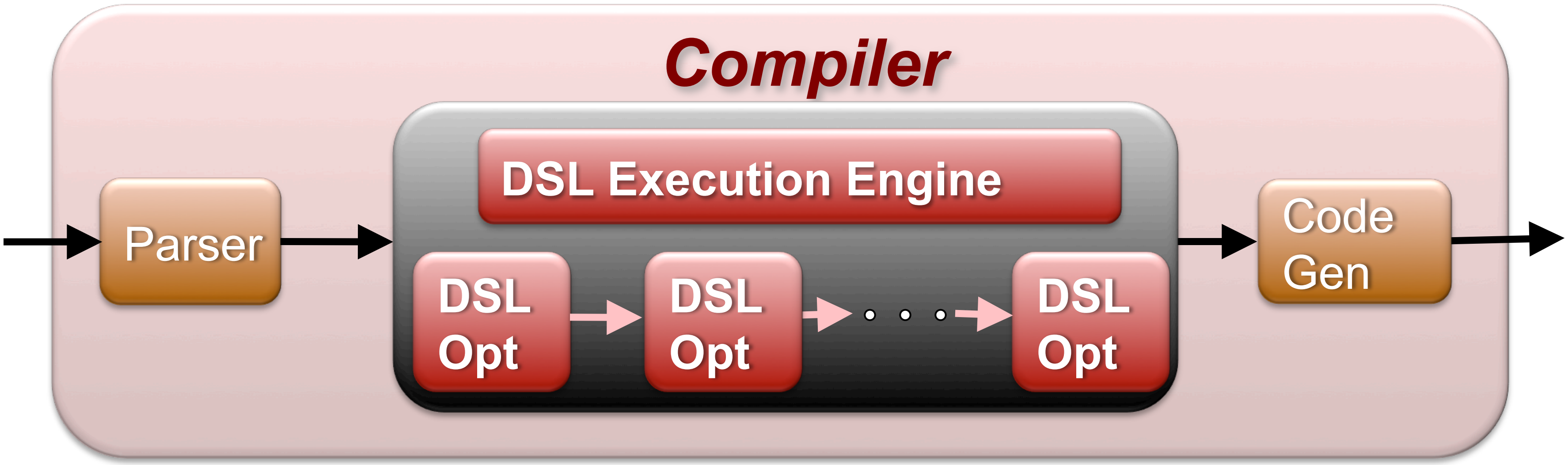
Traditional Compilers



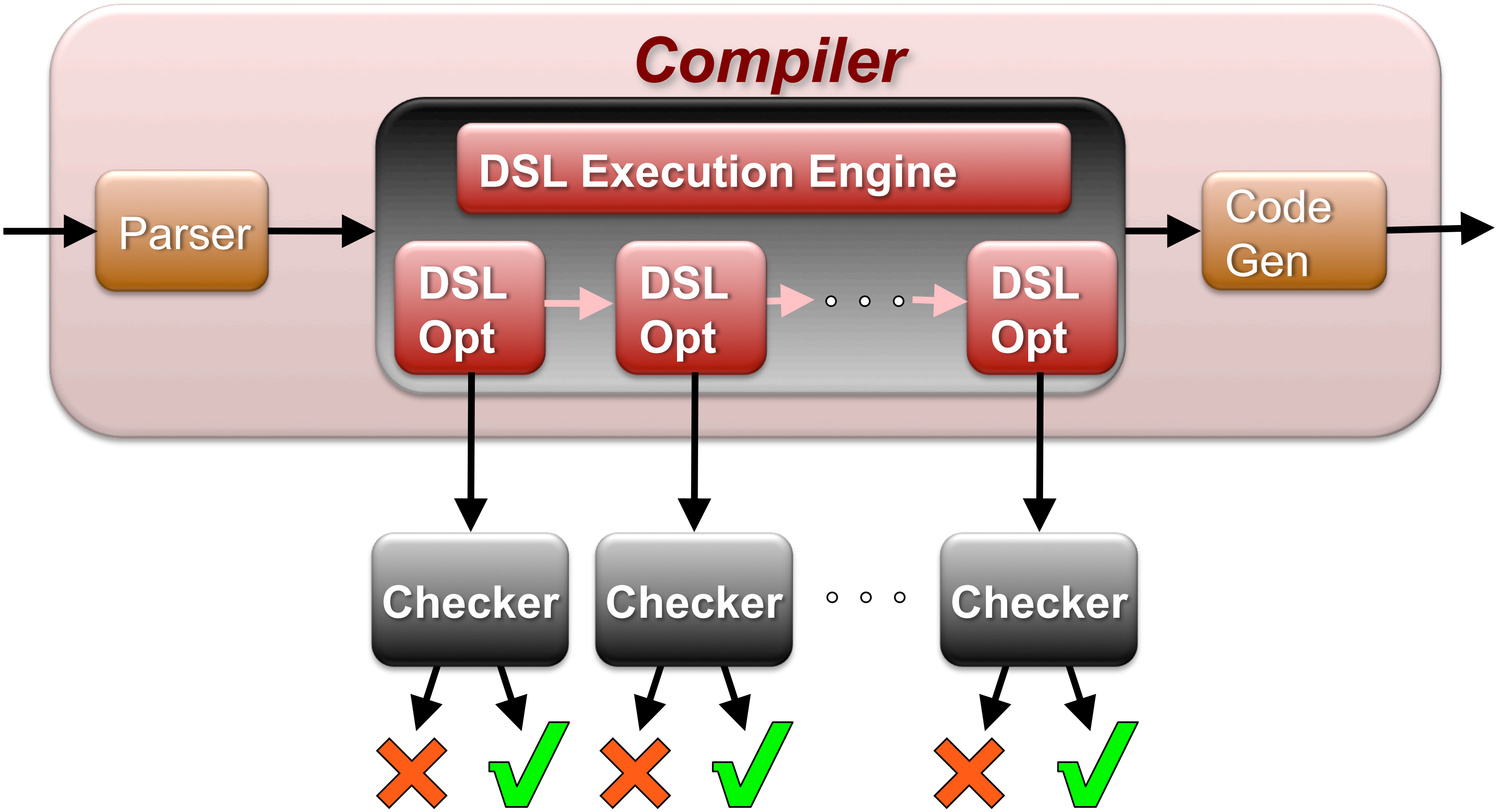
Using a Domain Specific Language



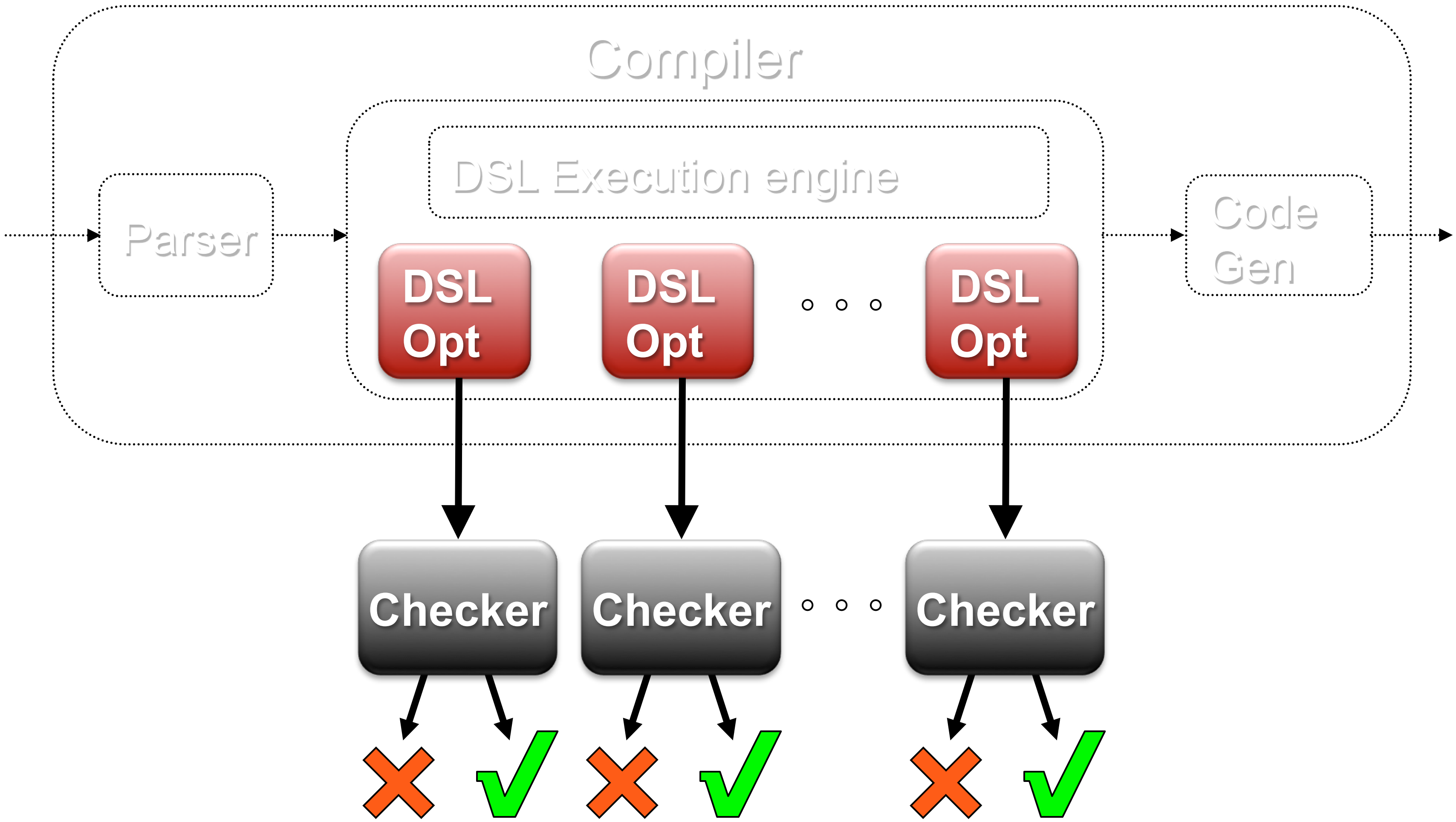
Using a Domain Specific Language



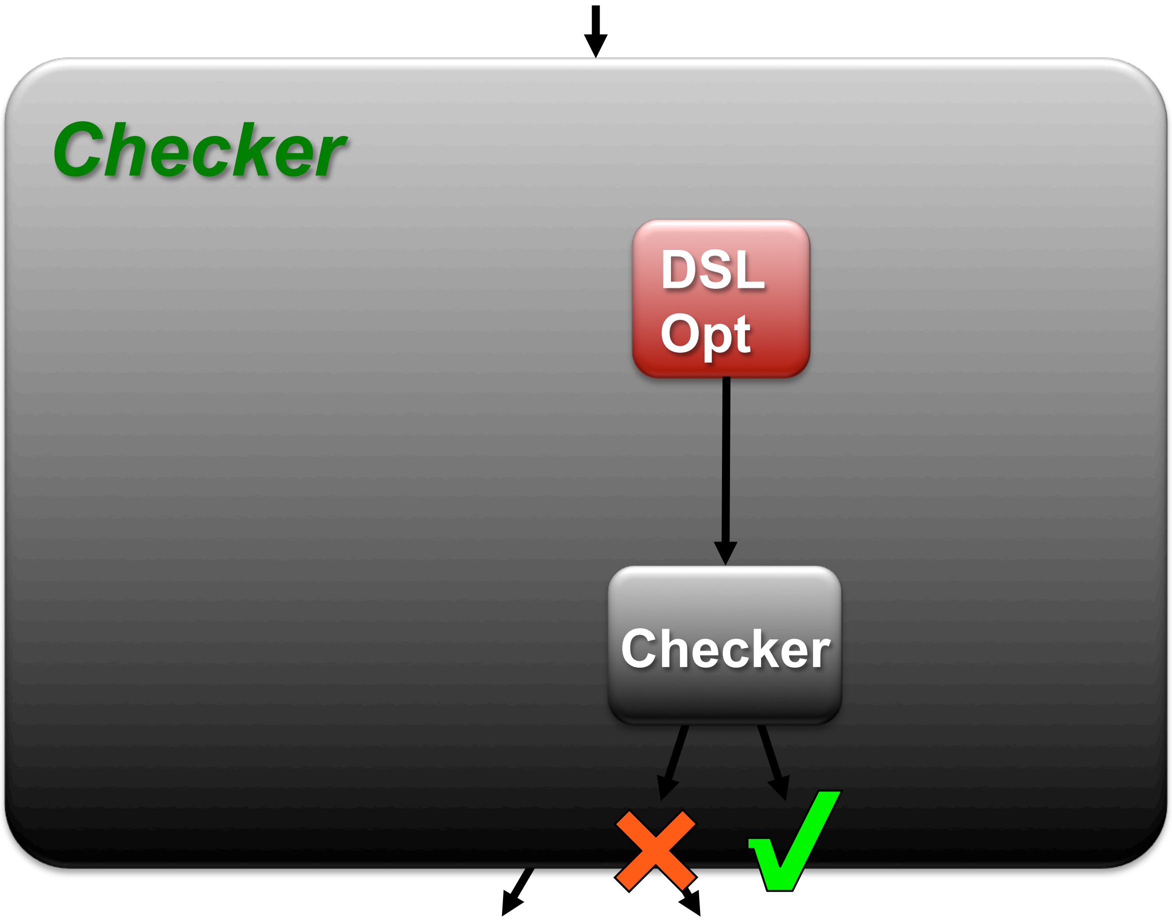
Checking Correctness



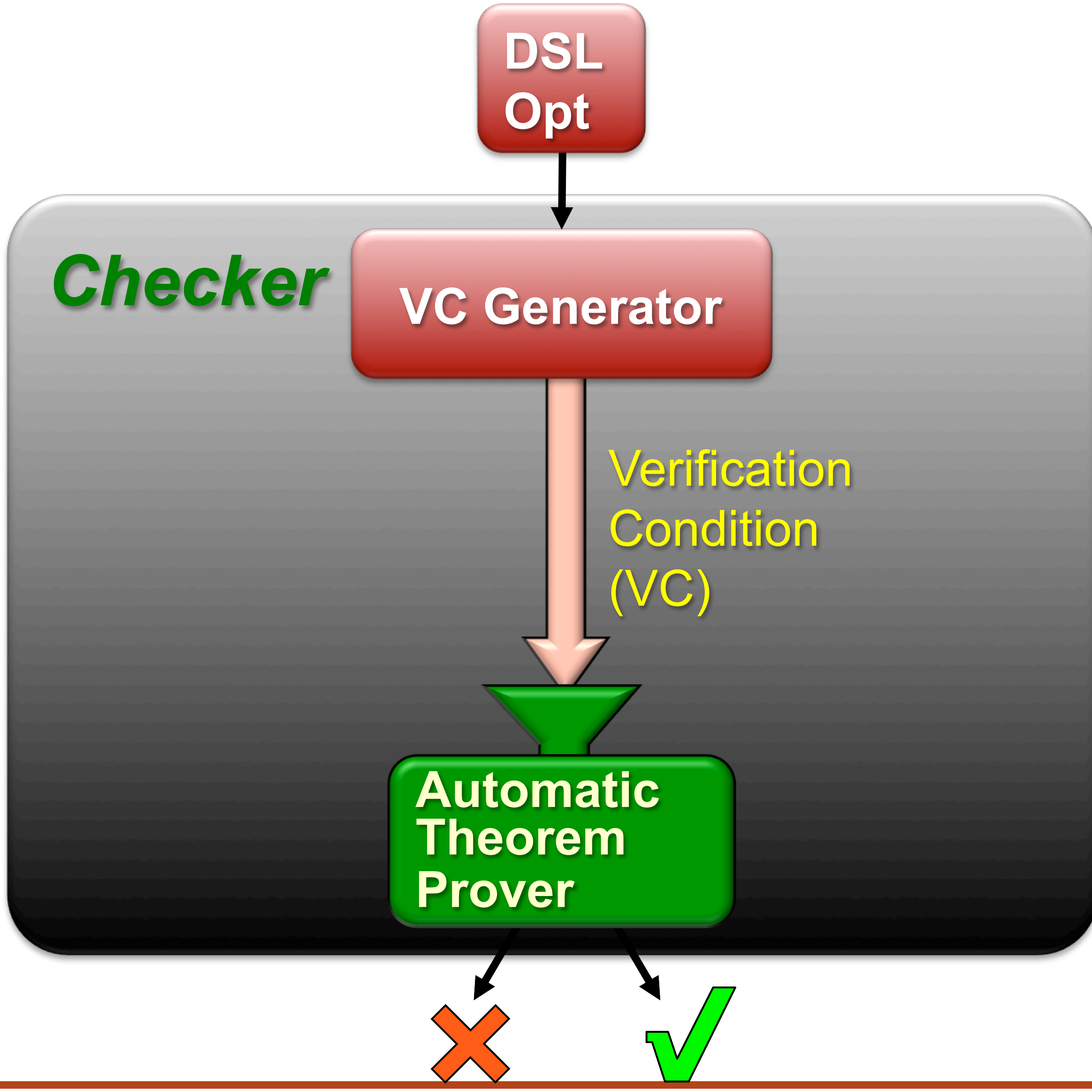
Checking Correctness



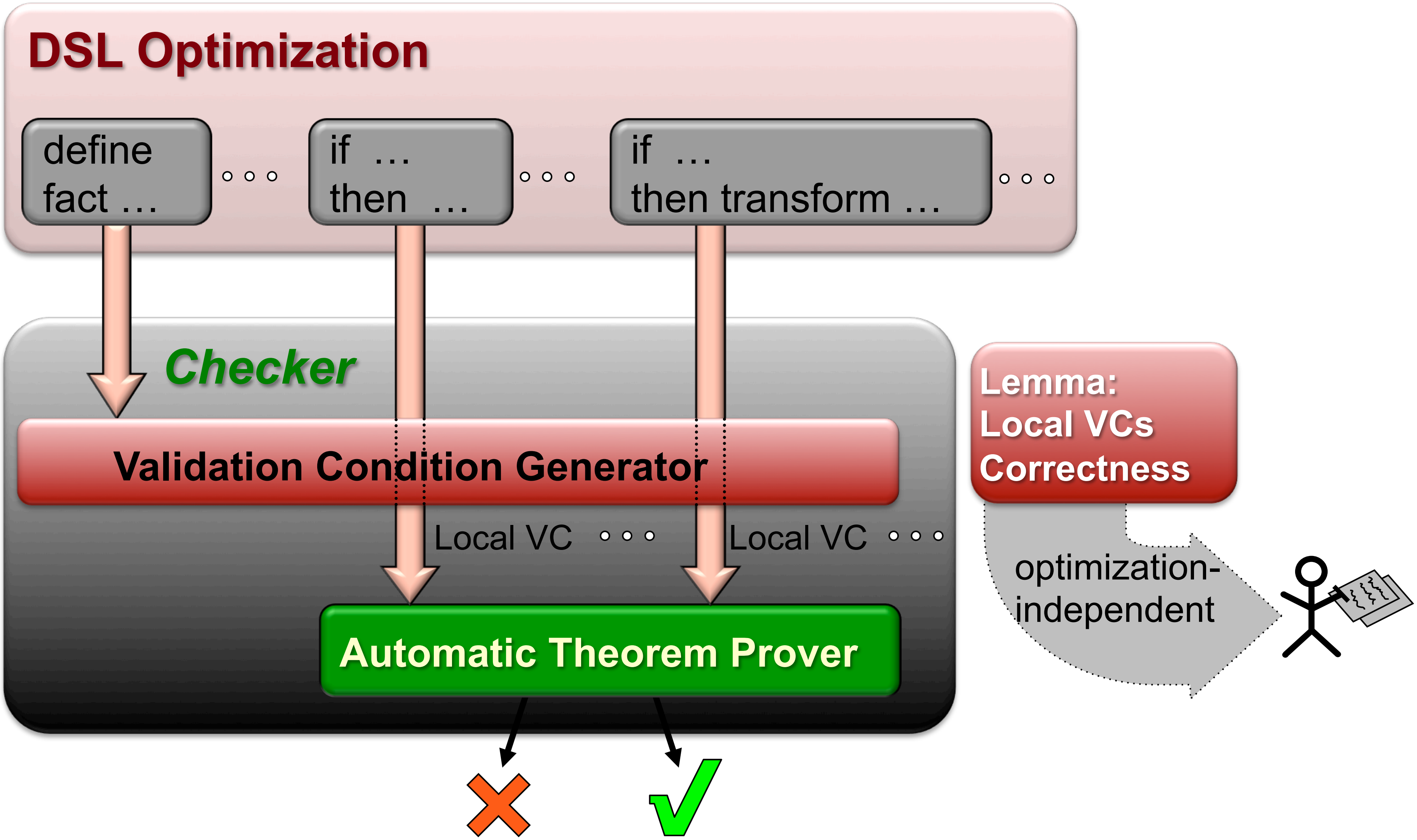
Checking Correctness



Correctness Checker



Correctness Checker

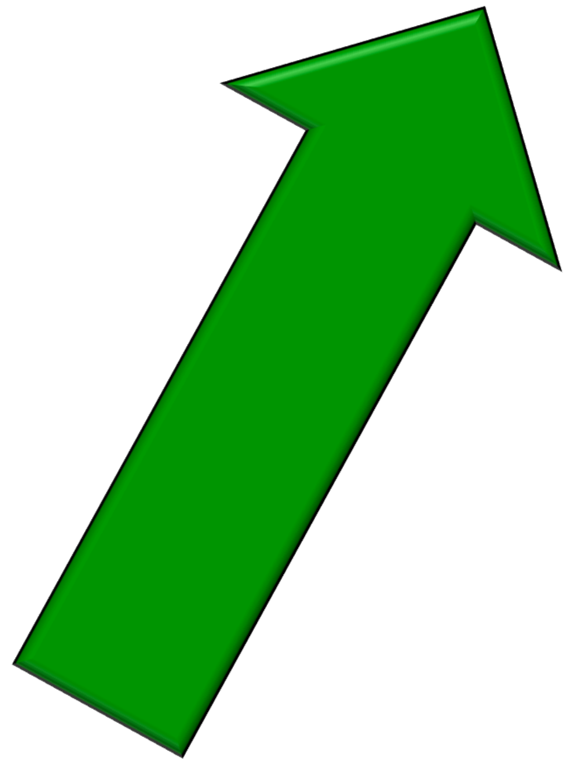


What are some of the benefits of DSLs?
What are some of the Risks?

- Again, think for 15 seconds…
- Let's talk…

Benefits of using DSLs

- **Expressiveness**: DSLs allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain
- **Reusability:** DSL programs are concise, and can be reused for different purposes
- DSLs enhance **productivity**, **reliability**, **maintainability**, **portability**, and **testability**

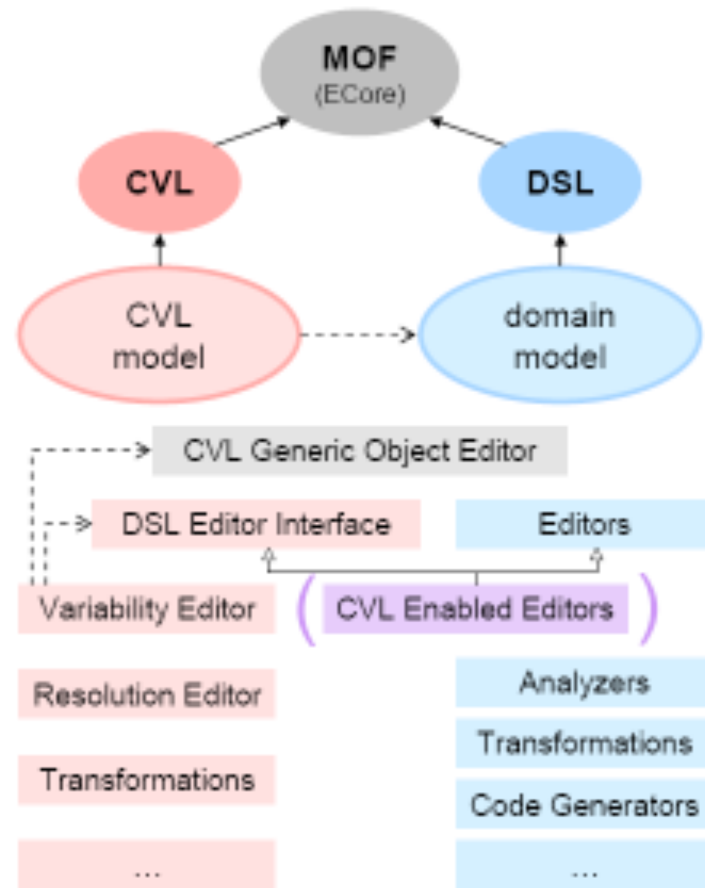


Risks of using DSLs

- Costs shift towards
  - Designing, implementing and maintaining DSLs
- Language issues
  - Limited availability of DSLs
- Issues in practice
  - Potential loss of efficiency
  - Integrating various DSLs is difficult

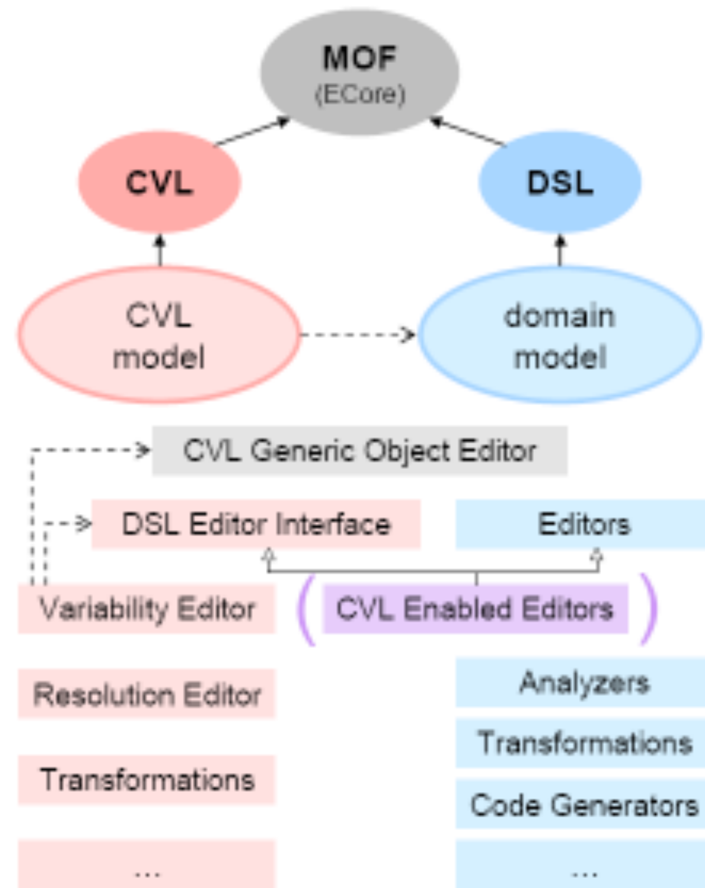
Example Implementation Approaches 1/3

- Macro processing, lexical processing, source-to-source transformation, pipeline
  - Pros: easy in implementation
  - Cons: absence of semantic analysis; problematic error reporting
- Embedding/Internal
  - Pros: Reused compiler/ interpreter
  - Cons: Limited expressiveness; problematic error reporting



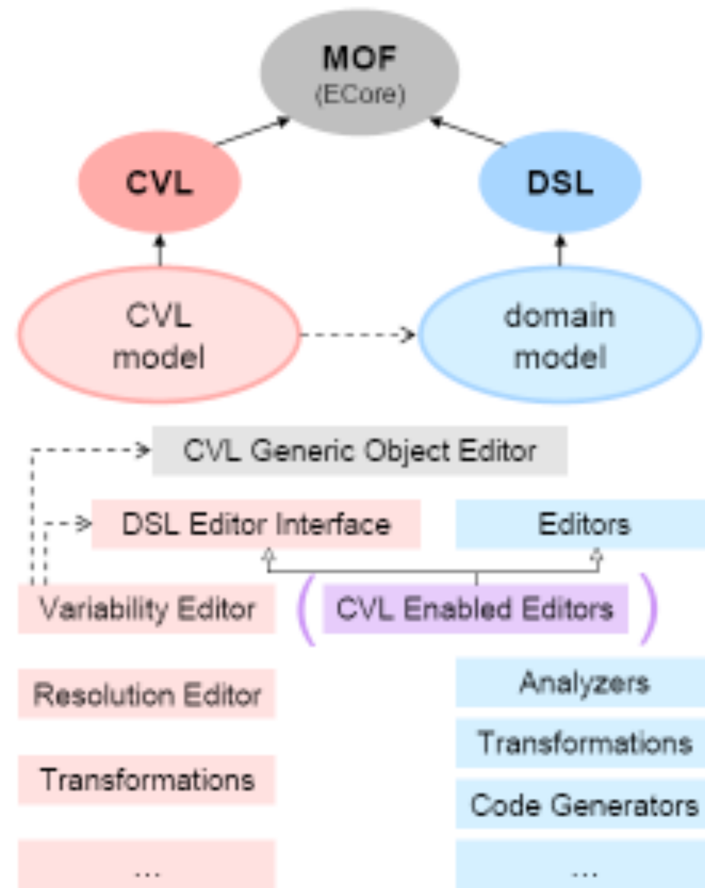
Example Implementation Approaches 2/3

- Compiler/interpreter
  - Pros: Ability to domain-level optimization, analysis
  - Cons: High building cost
- Compiler generator
  - Pros: Ability to domain-level optimization, analysis with minimized building efforts
  - Cons: still, implementing compiler is hard even if compiler generator are used



Example Implementation Approaches 3/3

- Extensible compiler/ interpreter
  - Pros: Reused compiler with minimized effort
  - Cons: Extreme caution to prevent interference



Eclipse Modeling Framework (EMF)

- Most programs manipulate some data model
  - It might be defined using Java, UML, XML Schemas, or some other definition language
- EMF extracts this intrinsic "model" and generates some of the implementation code
  - Can be a tremendous productivity gain
- EMF is one implementation of MOF
  - Not EMF = MOF

EMF Model Definition 1/2

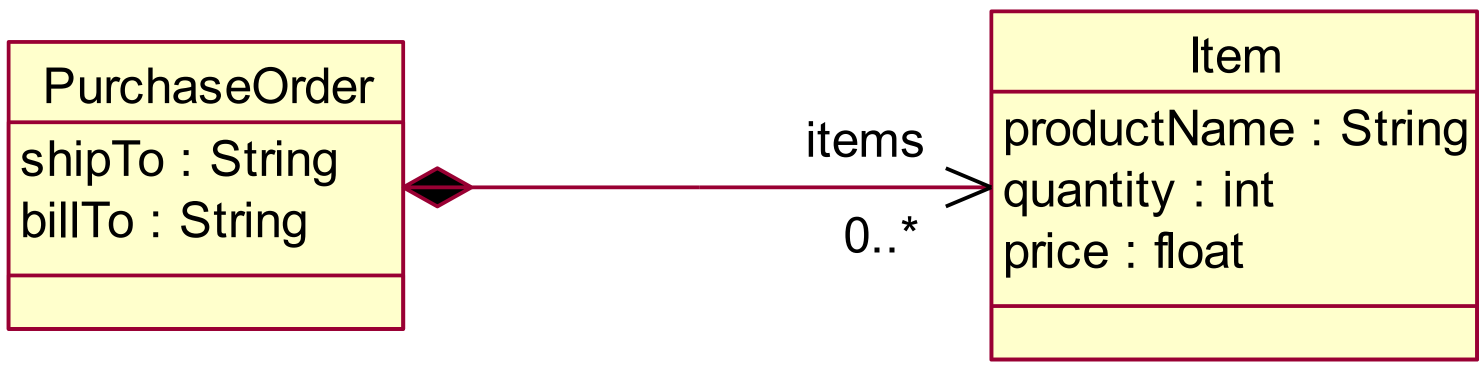
- Specification of an application's data
  - Object attributes
  - Relationships (associations) between objects
  - Operations available on each object
  - Simple constraints (e.g., multiplicity) on objects and relationships
- Essentially the Class Diagram subset of UML

EMF Model Definition 2/2

- EMF models can be defined in three ways:
  1. Java interfaces
  2. UML Class Diagram
  3. XML Schema

- Choose the one matching your perspective or skills, and EMF can generate the others as well as the implementation code

EMF Model Definition: UML class diagrams



EMF Model Definition: Java Interfaces

```
public interface PurchaseOrder {
  String getShipTo();
  void setShipTo(String value);
  String getBillTo();
  void setBillTo(String value);
  List getItems(); // List of Item
}

public interface Item {
  String getProductName();
  void setProductName(String value);
  int getQuantity();
  void setQuantity(int value);
  float getPrice();
  void setPrice(float value);
}
```

EMF Model Definition - XML

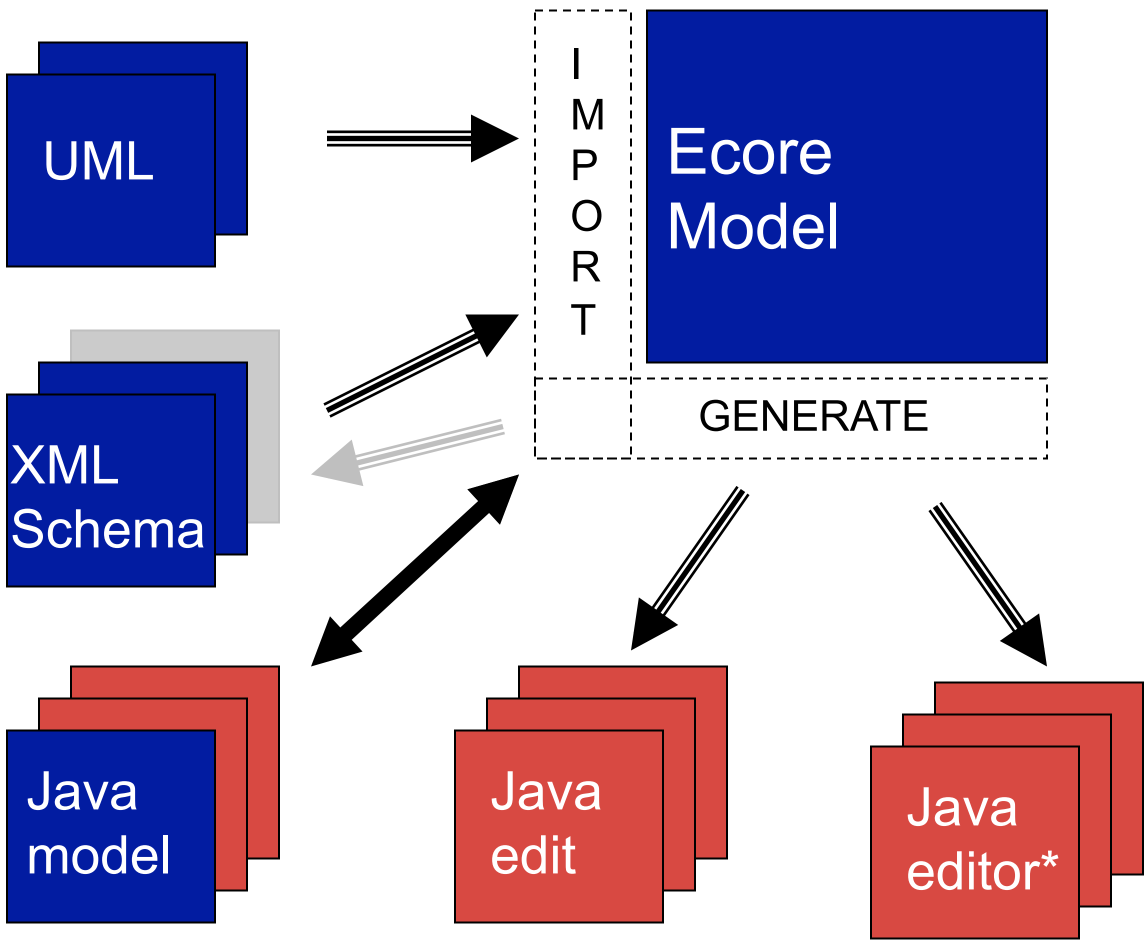
```
<xsd:complexType name="PurchaseOrder">
 <xsd:sequence>
  <xsd:element name="shipTo" type="xsd:string"/>
  <xsd:element name="billTo" type="xsd:string"/>
  <xsd:element name="items"  type="PO:Item"
               minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Item">
 <xsd:sequence>
  <xsd:element name="productName" type="xsd:string"/>
  <xsd:element name="quantity" type="xsd:int"/>
  <xsd:element name="price" type="xsd:float"/>
 </xsd:sequence>
</xsd:complexType>
```

Unifying Java, XML, and UML

- All three forms provide the same information
  - Different visualization/representation
  - The application's "model" of the structure
- From a model definition, EMF can generate:
  - Java implementation code, including UI
  - XML Schemas
  - Eclipse projects and plug-in

EMF Architecture:
Model Import and Generation



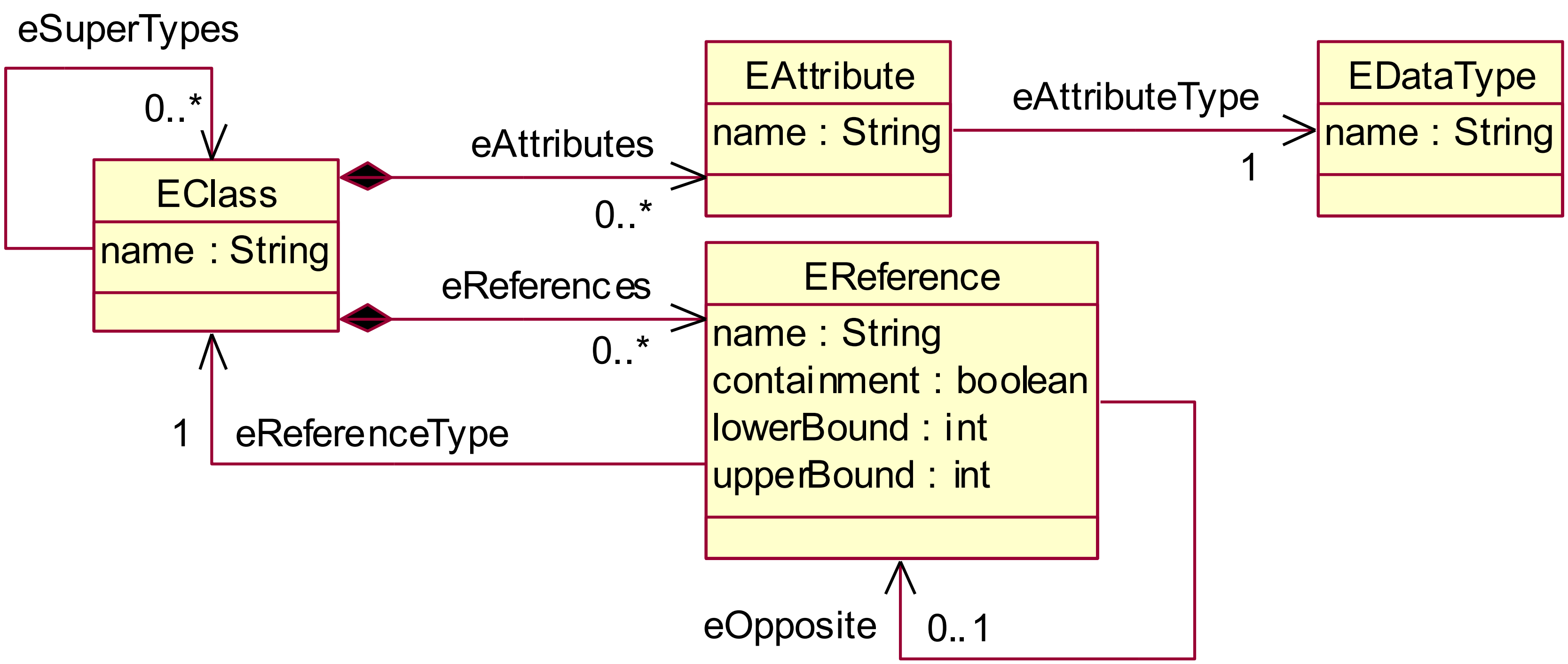
Generator features:

- **Customizable JSP-like templates (JET)**
- **Command-line or integrated with Eclipse JDT**
- **Fully supports regeneration and merge**

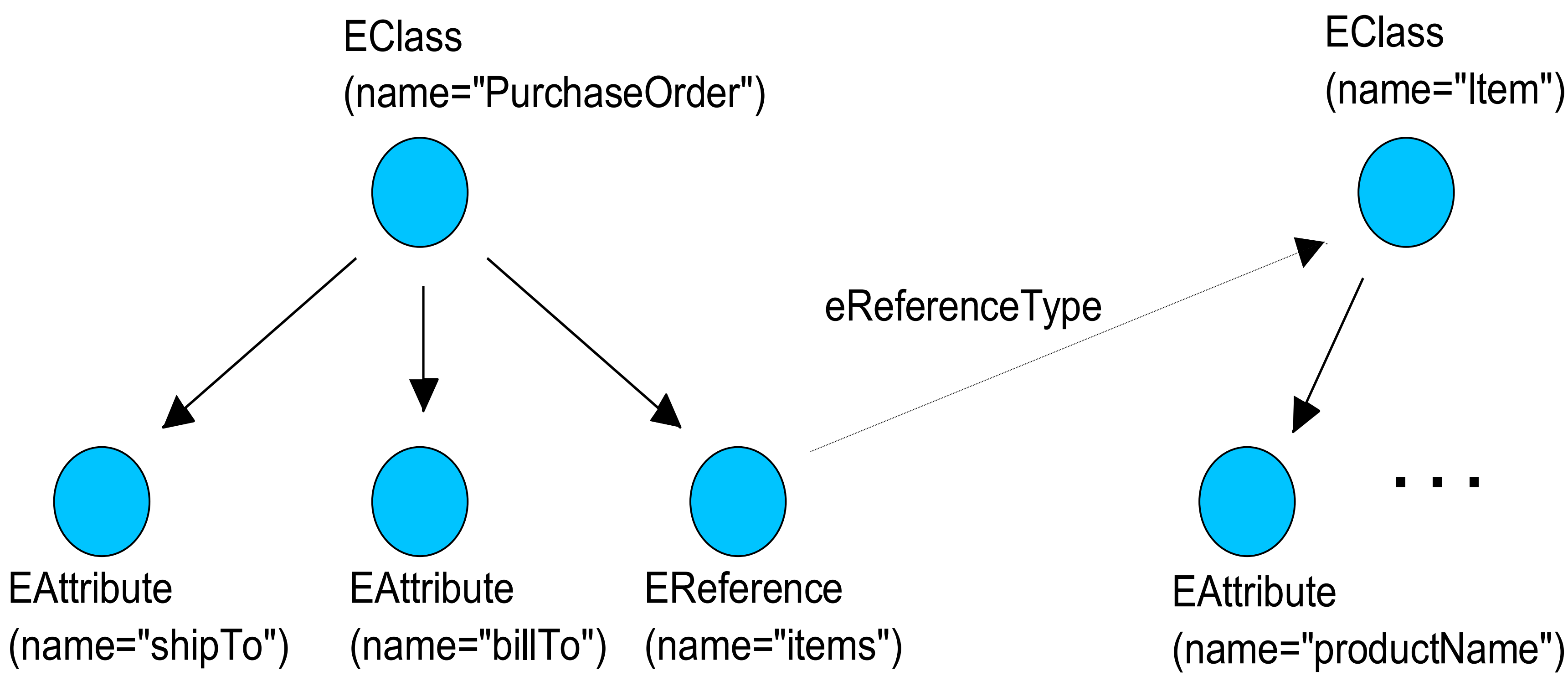
* requires Eclipse to run

EMF Architecture - Ecore

- Ecore is EMF's Metamodel (model of a model)
  - Persistent representation is XMI



EMF Architecture - PurchaseOrder Ecore Model



EMF Architecture - PurchaseOrder Ecore XMI

```
<eClassifiers xsi:type="ecore:EClass"
  name="PurchaseOrder">
 <eReferences name="items" eType="#//Item"
   upperBound="-1" containment="true"/>
 <eAttributes name="shipTo"
   eType="ecore:EDataType http:...Ecore#//EString"/>
 <eAttributes name="billTo"
   eType="ecore:EDataType http:...Ecore#//EString"/>
</eClassifiers>
```

- Alternate serialization format is EMOF
  - Part of MOF 2.0 Standard

EMF Dynamic Architecture

- Given an Ecore model, EMF also supports dynamic manipulation of instances
  - No generated code required
  - Dynamic implementation of reflective EObject API provides same runtime behavior as generated code
  - Also supports dynamic subclasses of generated classes
- All EMF model instances, whether generated or dynamic, are treated the same by the framework

Code Generation - Feature Change

- Efficient notification from "set" methods
  - Observer Design Pattern

```
public String getShipTo() {
  return shipTo;
}

public void setShipTo(String newShipTo) {
  String oldShipTo = shipTo;
  shipTo = newShipTo;
  if (eNotificationRequired())
    eNotify(new ENotificationImpl(this, ... );
}
```

Code Generation

```
public interface EObject {
  Object eGet(EStructuralFeature f);
  void eSet(EStructuralFeature f, Object v);
  ...
}
```

- All EMF classes implement interface Eobject
- Provides an efficient API for manipulating objects reflectively
  - Used by the framework (e.g., generic serializer, copy utility, generic editing commands, etc.)
  - Also key to integrating tools and applications built using EMF

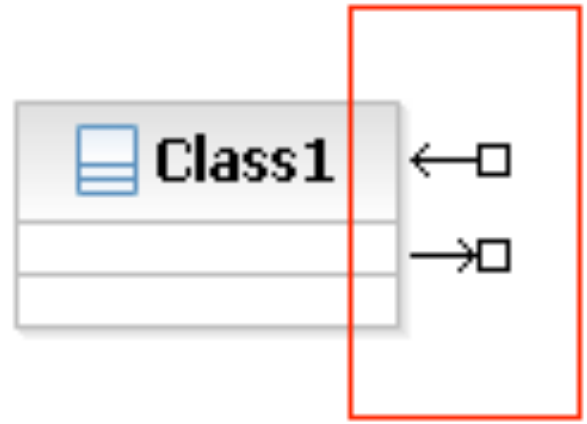
Short Discussion/Exercise:

How would you add behaviors to a declarative representation like that seen in EMF?

- What are the alternatives?
- How hard are they to implement?
- Is there support from the community?

MOF Action Semantics

- EMF has limited Behavioral Modeling support
- Action semantics capture the behavior of a model (i.e., how the model behaves)
- Actions semantics has been proposed for UML 2.0.
  - Variants appear in Executable UML
- Let's talk more about Action semantics and Object Constraint Language (OCL) on Monday



Homework and Milestone Reminders

- **Case Study/Homework: "UML 2: A model-driven development tool" by B. Selic**
  - **Be prepared to discuss and even lead the discussion**
  - **Write a brief summary of observations on the paper based on assignment (on Angel)**

- **Milestone 2: Establish a repository and structure for assembling components for your FacePamphlet application**
  - **Due by 11:55pm Friday, April ??? ~~1st~~, 2011**