Software Composition Paradigms

Sommersemester 2015

Radu Muschevici

Software Engineering Group, Department of Computer Science

TECHNISCHE
UNIVERSITÄT
DARMSTADT

2015-05-19

How to Read a Scientific Article

Why Reading Scientific Articles?

Reading papers is a significant activity for a researcher

- To keep current in your field
- To survey a new field you are interest in
- To review a paper for class or a conference/journal
- To learn how to write papers yourself

Goal: understand the scientific contributions the authors make

Reading Scientific Articles

- Reading a scientific article is a complex task (can take many hours even for experienced people)
- There are many approaches to reading scientific articles
- Over time you will develop own method that works best for you
- Here are some suggestions (but keep in mind – there is not one single “right” way)

Some General Tips

- Do **not** read a paper like a novel (beginning to end).
- Instead, do several passes over the paper, paying increasing attention to details.
- Take breaks between passes.
- Take notes as you read; Highlight important statements.
- You don't need to understand every detail about the paper.
- Read critically: You should not assume that the authors are always correct.
- Write a short critique (5 sentences): What is the main idea of the paper? What did you like about the idea? What did you not like about the idea? How reading this paper influenced you?
- Discuss with your peers, try to clarify points that you don't understand.

The Three Pass Approach[1]: First Pass

Read title, abstract, introduction and conclusion; section headings.

- To get the "big picture"
- 10–15 minutes

Title, Abstract and Conclusion

- What is the problem addressed? The proposed solution?
- What are the paper's main contributions/results?

Introduction

- Motivation: What is the problem addressed in the paper, why is it a problem, and why is it important?
- How is the paper structured (outline)?
- What are the main contributions?
- Background information

Also, take note **when** the paper was published.

[1] Adapted from [Keshav 2007]

The Three Pass Approach: Second Pass

- Read the paper with greater care, focus on the main sections (these contain the contributions)
- Read critically: what are the paper's innovations, but also its hidden failings and assumptions?
- You should be able to summarize the main contributions of the paper.
- 1–2 hours

The Three Pass Approach: Third Pass

- Optional
- Carefully re-read sections that you didn't fully understand
- Look up references if necessary
- Read creatively: What are the good ideas in the paper? What are other applications and extensions? What improvements would you make?
- Check proofs, appendices

In Summary…

- First, get a bird's-eye-view
- Then dig into the details
- Read critically
- Don't worry if you don't understand all the small details
- But make sure you understand the scientific contributions
- Do not leave reading to the last minute (before the exam).

Metaprogramming & Reflection

(Based on slides by Kim Mens, Université Catholique de Louvain)

What is “meta”

In Greek

- The preposition “μετα” means “after”, “beside” or “with”

In English

- The prefix “meta-” means “about (its own category)”
- For example, metadata is data **about** data: who has produced it, when, what format is the data in, …

Douglas Hofstadter popularised the meaning of the term

- In his books “Gödel, Escher, Bach” and “Metamagical Themas”
- For example “going meta” is a rhetorical trick to take a debate or analysis to a higher abstraction level: “This debate isn’t going anywhere”

Metaprogramming

In computer science metaprogramming is the writing of computer programs that write or manipulate other programs as their data.

Some examples of metaprogramming

- A compiler
- An interpreter
- A code analysis tool
- A code generator

Reflective programming (**reflection**) is a case of metaprogramming

- Where a program is its own metaprogram
- i.e. when a program manipulates itself

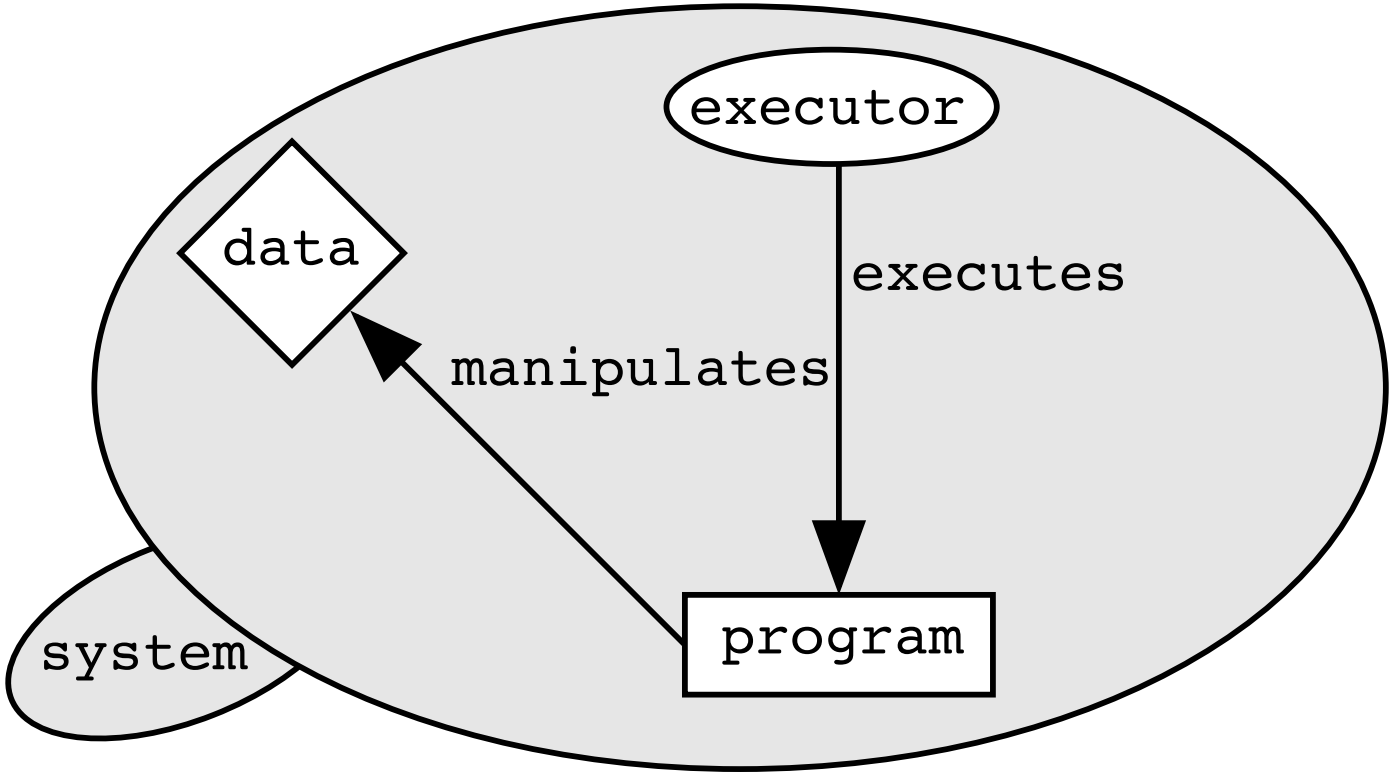
Metaprogramming: Concepts

To understand the notion of metaprogramming, let us first define some important concepts:

- Computational system
- Programming language and program
- Meta system
- Meta programming language and meta program
- Meta language and base language

Computational System

A computational system is a system that reasons about and acts upon some part of the world, called the **domain** of that system.

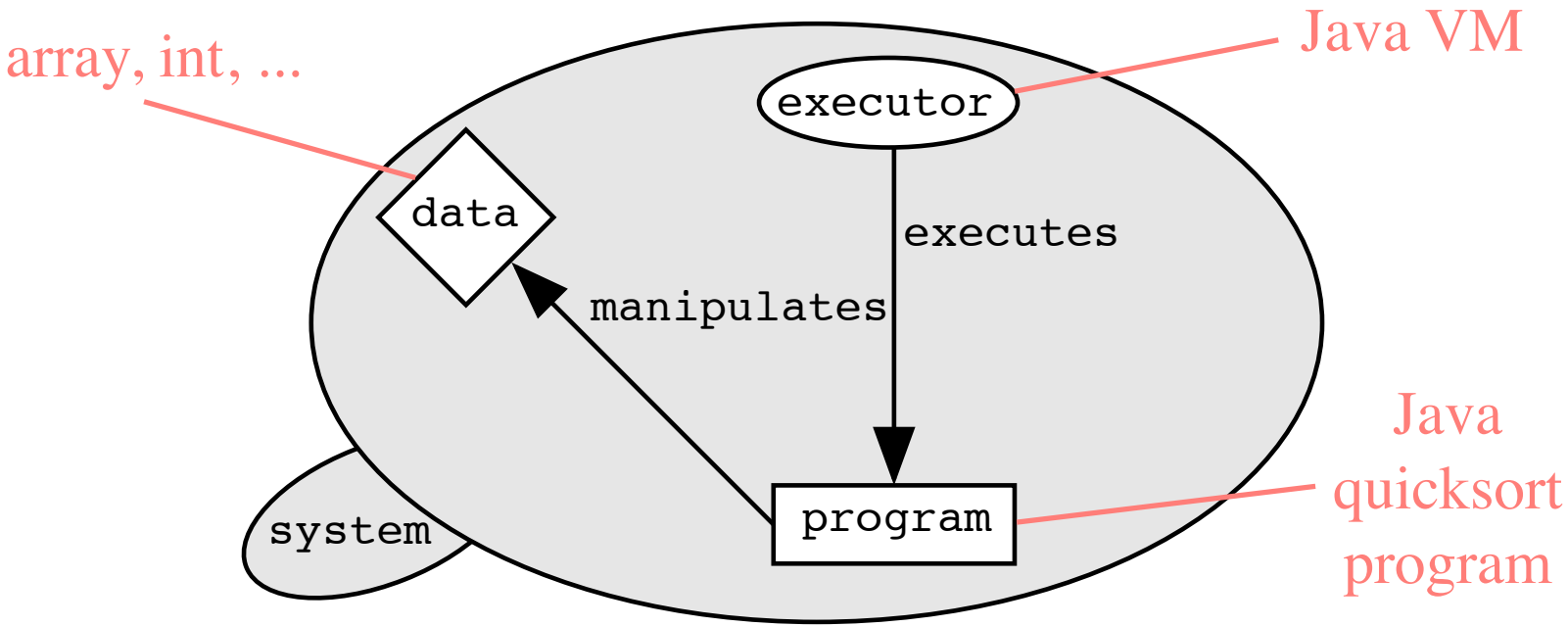


Domain of a Computational System

- The **domain** of a computational system (or its **universe of discourse**) is the collection of elements (entities or concepts) it can reason about.
- Example: an address book application is a system that reasons about a domain which contains (real) persons, names, phone numbers, addresses, birthdays, etc.
- The computational system manipulates **representations** of these elements.

Computational System: Example

- **Domain:** lists and numbers
- **Computation:** sorting



Program and Programming Language

A program is a formal, executable specification of a computational system.

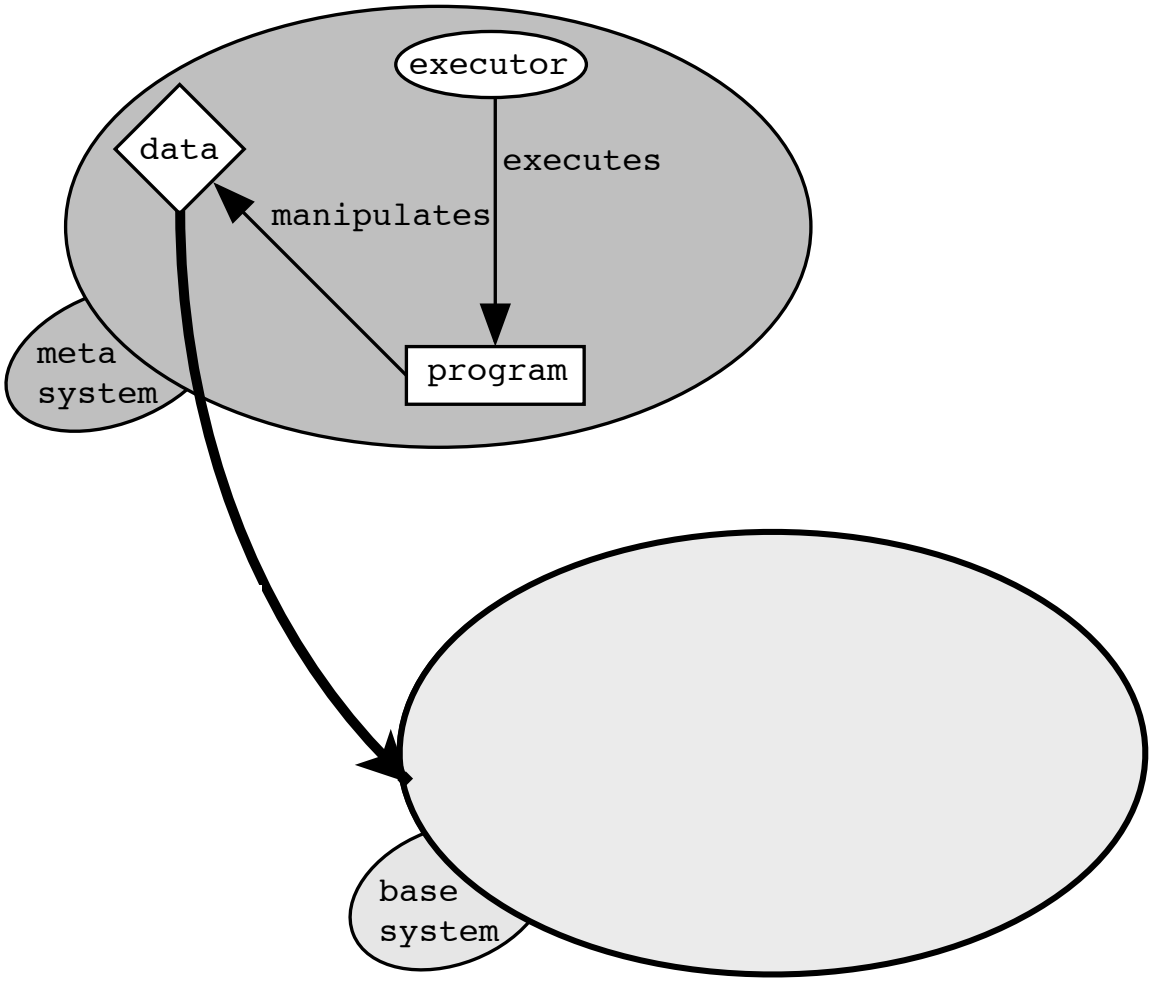
Example: a Java quicksort program

A programming language is a formalism that can be interpreted in an automatic manner in order to obtain the computational system specified by the program written in it.

Example: the Java programming language

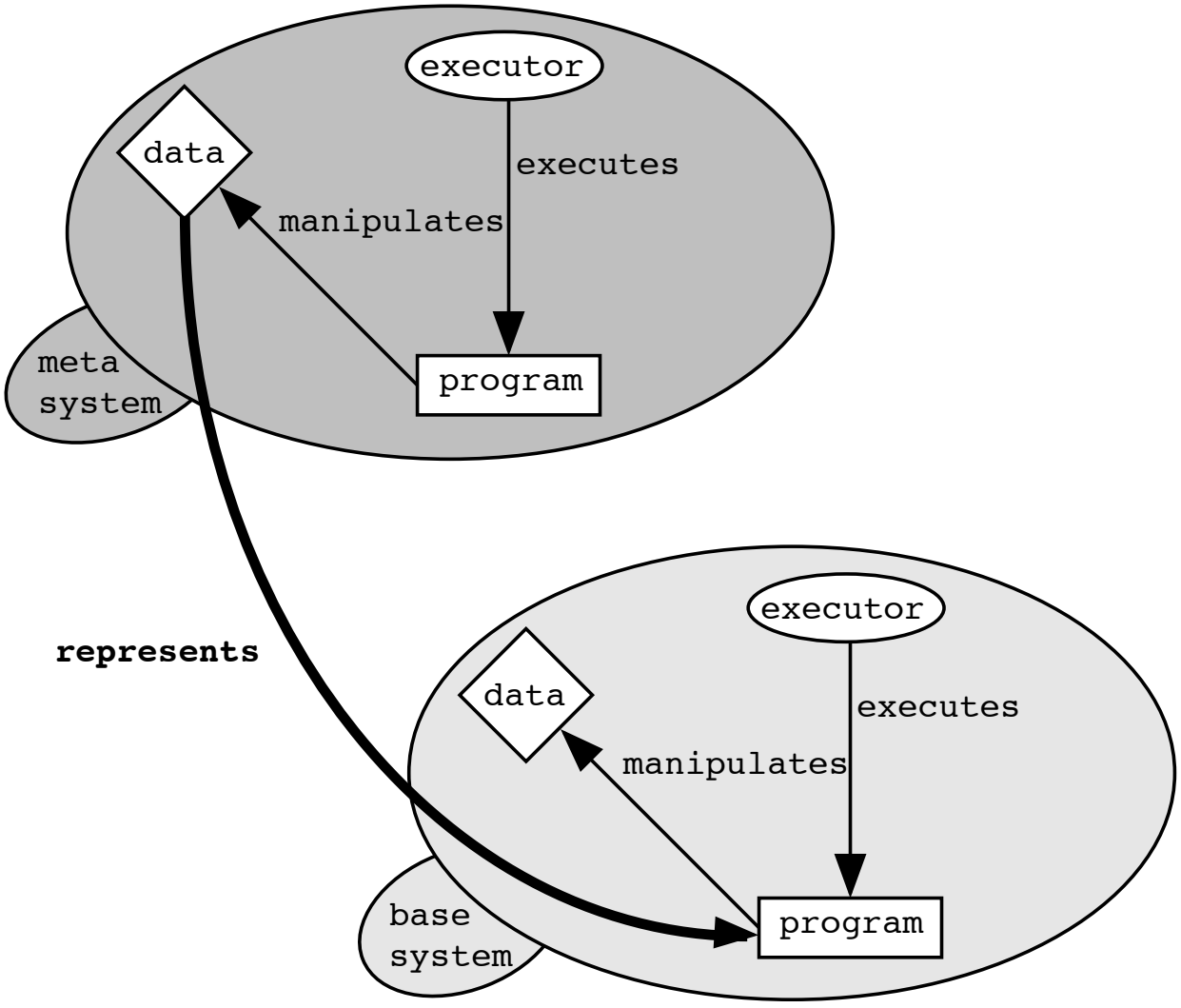
Meta System

A meta system is a system that has as its domain another computational system, called its base system.



Meta System (cont.)

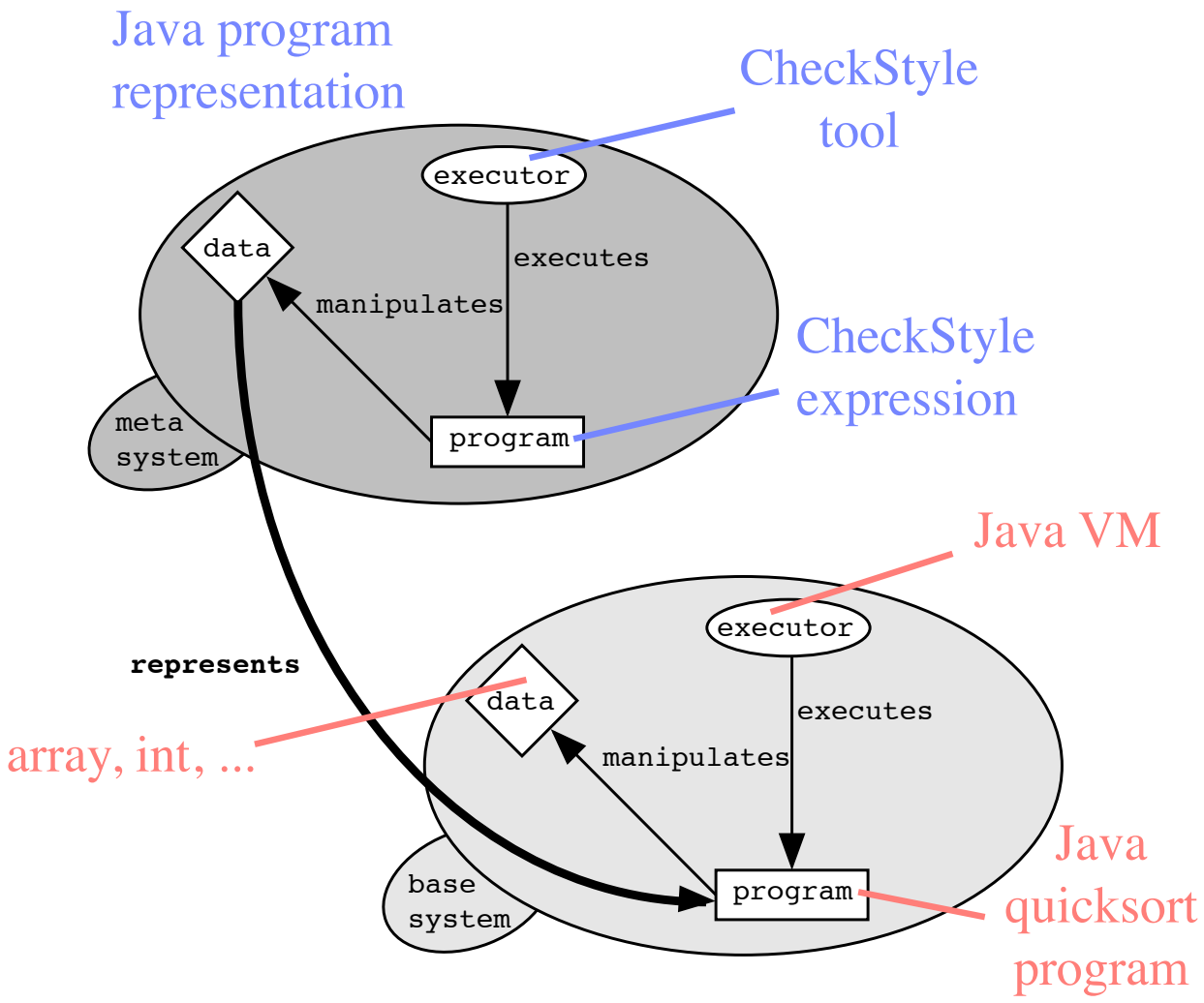
Somehow the data of the meta system needs to **represent** programs of the base system.



Meta System Example: CheckStyle[2]

"Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard.

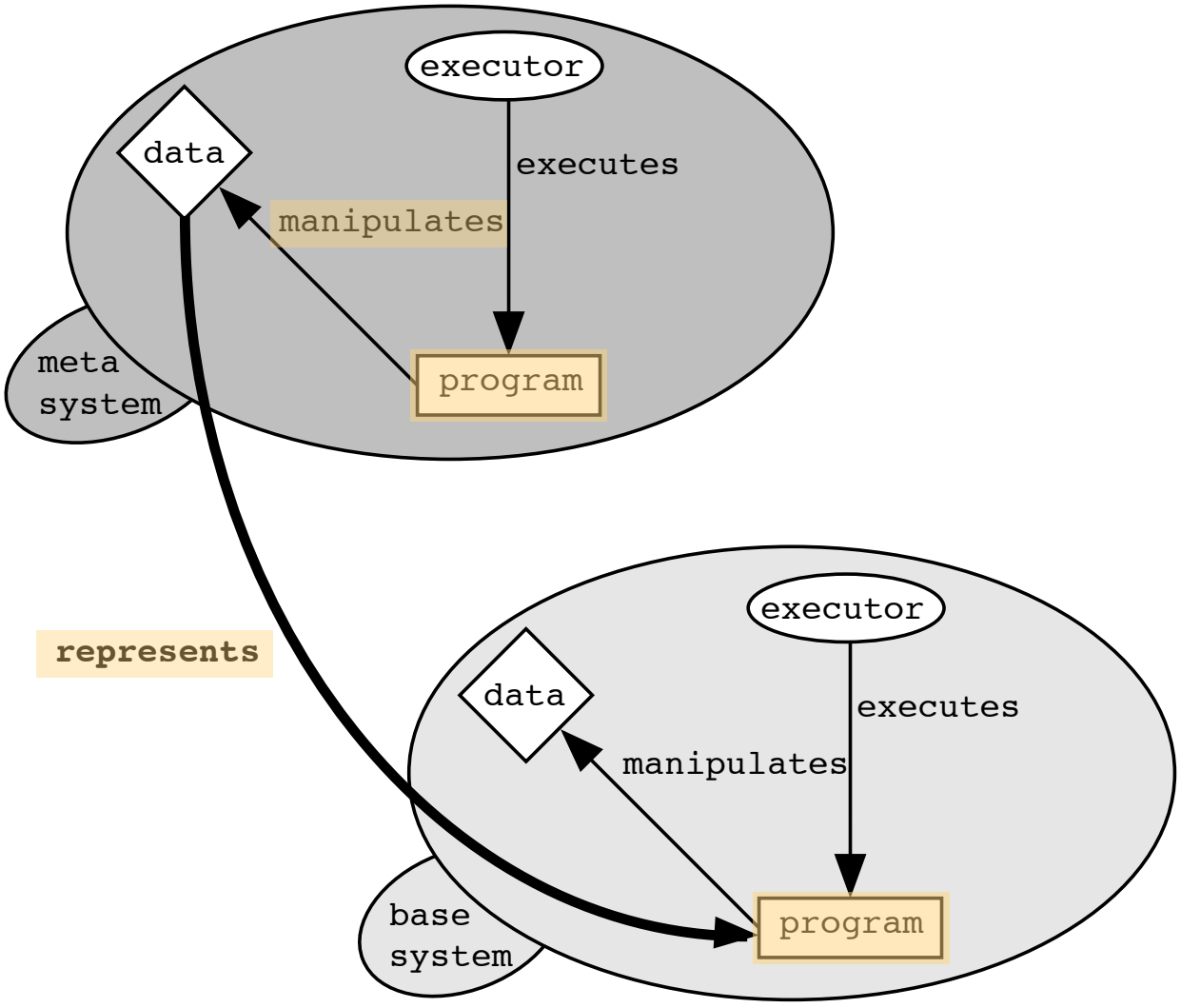
It automates the process of checking Java code to spare humans of this boring (but important) task This makes it ideal for projects that want to enforce a coding standard."



[2] http://checkstyle.sourceforge.net/

Meta Program

A meta program is the program specifying the meta system of a computational system.



Meta Program (cont.)

The meta program manipulates programs of the base system.

- Example: A CheckStyle expression to check for a particular code smell, such as “duplicate code”

The meta system does not directly manipulate its base system. It needs some kind of explicit representation of the base program.

- Example: CheckStyle represents Java programs as parse trees

Base and Meta Language

Metaprogramming involves two languages:

- Base language: in which the base programs are written
- Meta language: in which the meta program is written

A meta program is written in the meta language and reasons about a base program written in the base language.

- The meta language can be a general-purpose programming language, or specifically tuned for specifying meta programs – example: CheckStyle

Both languages have to be integrated somehow

- Meta language needs explicit representation of (concepts relevant to meta programs in) the base language
- For example, a “meta-object protocol” (MOP)

Base and Meta Language: Examples

CheckStyle tool

- Goal: meta “program” checks conventions in Java programs
- Meta language = configuration files accepted by CheckStyle
- Base language = Java

Aspect-oriented programming

- Goal: meta program “weaves” aspects in base program
- Meta language = AspectJ
- Base language = Java

Reflection

- Goal: meta program manipulates and modifies base program
- Meta program = Base program

Reflection (Language)

Meaning

- One of the meanings of reflection (in English) is "introspection": contemplation of self
- In other words, reflection is the ability of a thing to reason about itself

Reflection in natural language

- "This sentence contains 37 characters."
- "This sentence contains 27 letters."

Reflection can easily introduce paradoxes:
"This sentence is false."

- Suppose true. Then claim is correct. So false. Contradiction.
- Suppose false. Then claim is wrong. So true. Contradiction.

Computational Reflection

Computational reflection is the ability of a program to examine and change its own implementation at runtime.

- A reflective program is a meta program (= a program that manipulates a program)
- A reflective program is **its own** meta program (= a program that manipulates **itself**)

Reflection is an instance of metaprogramming where metaprogram and base program are the same.

Examples of reflective programming systems:
Smalltalk, Common Lisp Object System (CLOS), Java, Ruby, Perl, PHP, Python, VBScript, JavaScript, etc.

Reflection: Concepts

To understand the notion of reflection, let us first define some important concepts:

- Reification
- Causal connection
- Reflective system
- Introspection
- Intercession

Reification

Definition (Merriam-Webster Dictionary)

- Re-ify [verb]: to regard (something abstract) as a material or concrete thing

Definition (Wikipedia)

- Re-ify [verb]: to make an abstract concept or low-level implementation detail of a programming language accessible to the programmer

In reflective languages, reification is the act of making concepts of the language available as data, for manipulation by the programmer.

Reification (cont.)

Every aspect of the internal workings of a computational system that is explicitly **represented as data** of that system is said to be reified.

Examples

- The C programming language reifies the low-level detail of memory addresses $\Rightarrow$ the abstract notion of memory addresses becomes manipulable by the programmer.
- Languages, such as Curl, JavaScript, and Lisp provide an `eval` or `evaluate` procedure that effectively reifies the language interpreter $\Rightarrow$ the language interpreter can be invoked by the programmer.

Reification in Java

Most aspects of the internal workings of the Java virtual machine are not reified.

But you can explicitly import a reflection API to open up (some of) the inner workings of the VM

- `java.lang`
- `java.lang.reflect`

Example: `java.lang.Class` reifies the notion of a Java class

- The abstract notion of a Java class becomes manipulable from within normal Java programs
- Provides methods to examine the runtime properties of the object, including its members and type information
- Provides the ability to create new classes and objects

Reification in Java: `java.lang.Class`

```
static Class forName(String className)
```

Returns the Class object associated with the class or interface with the given string name.

```
Constructor[] getConstructors()
```

Returns an array containing `Constructor` objects reflecting all the public constructors of the class represented by this `Class` object.

```
Field[] getDeclaredFields()
```

Returns an array of `Field` objects reflecting all the fields declared by the class or interface represented by this `Class` object.

```
Method getDeclaredMethod(String name, Class[] parameterTypes)
```

Returns a `Method` object that reflects the specified declared method of the class or interface represented by this `Class` object.

```
Object newInstance()
```

Creates a new instance of the class represented by this `Class` object.

Reification: Causal Connection

Reification builds a **model** (an abstraction) of the system. To be useful to the programmer, the model needs to have a causal connection to the computational system it represents:

- Changes made to the model affect the system
- Changes to the system update the model

Reflective System

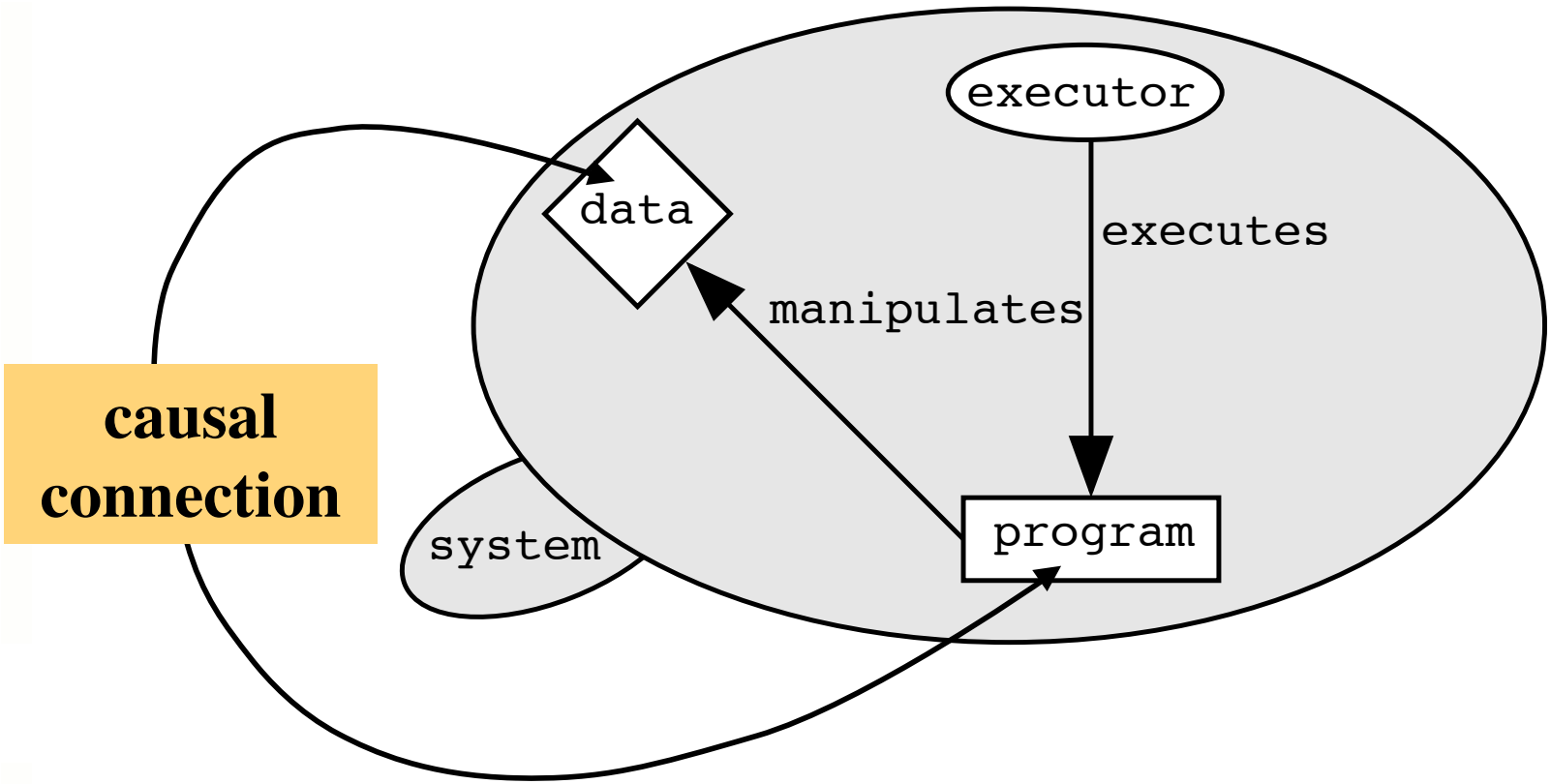
A computational system is causally connected to its domain if the computational system is linked with its domain in such way that, if one of the two changes, this leads to an effect on the other.

Example: robot arm

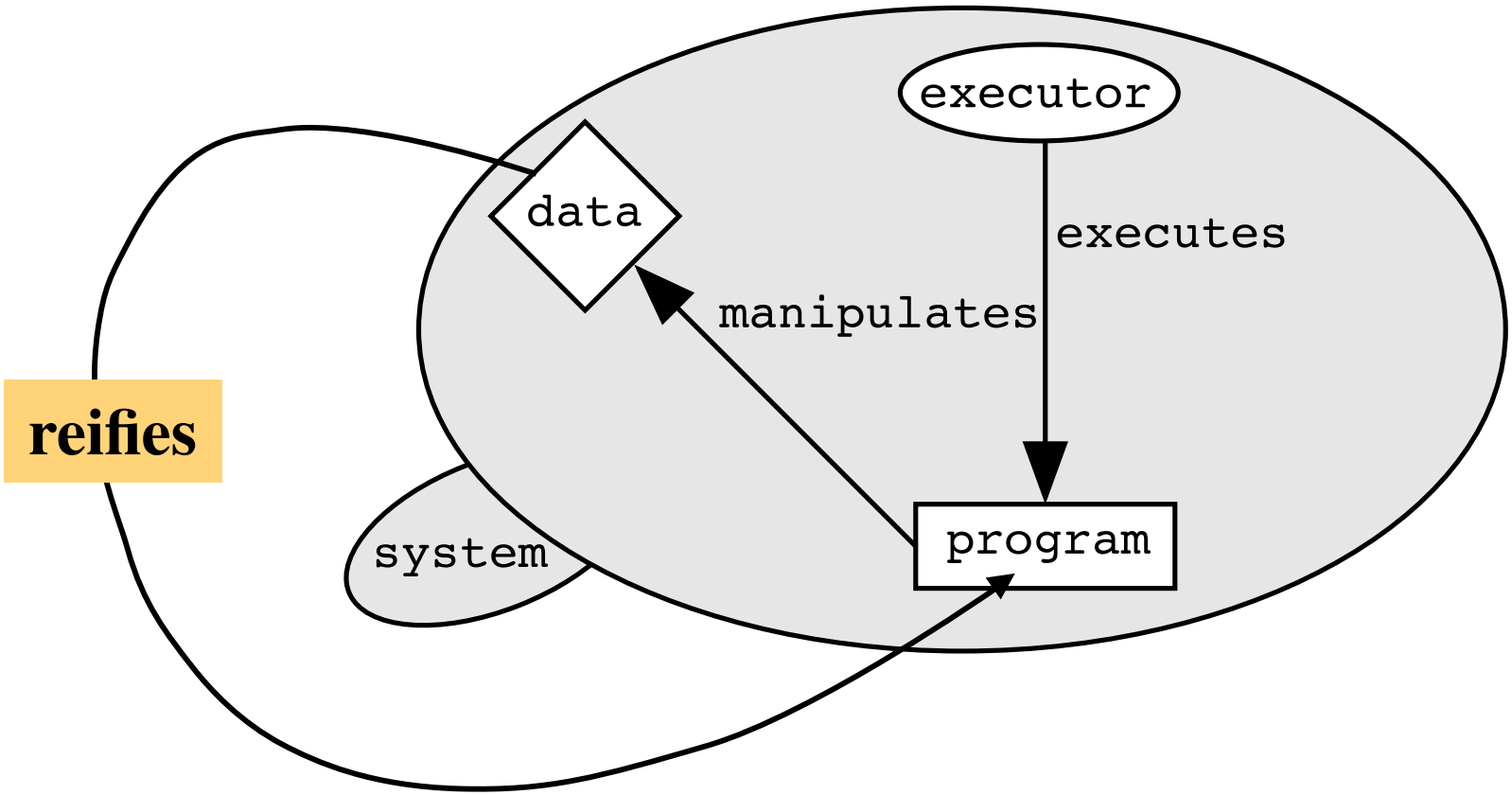
- Domain: coordinates indicating position of the arm
- Updating coordinates $\Rightarrow$ robot arm moves
- Moving the robot arm $\Rightarrow$ updates coordinates

A reflective system is a causally connected meta system that has itself as base system.

Reflective System (2)



Reflective System (3)



Introspection & Intercession

Reflective languages require language concepts to be reified so that they can be inspected and manipulated by programs as if they were ordinary data.

Reflection has two parts:

- Introspection: can only **look** at reified entities.
  Example: inspect the details of a class
- Intercession: can **change** the program behaviour by manipulating reified entities.
  Example: add a method to a class

Introspective System

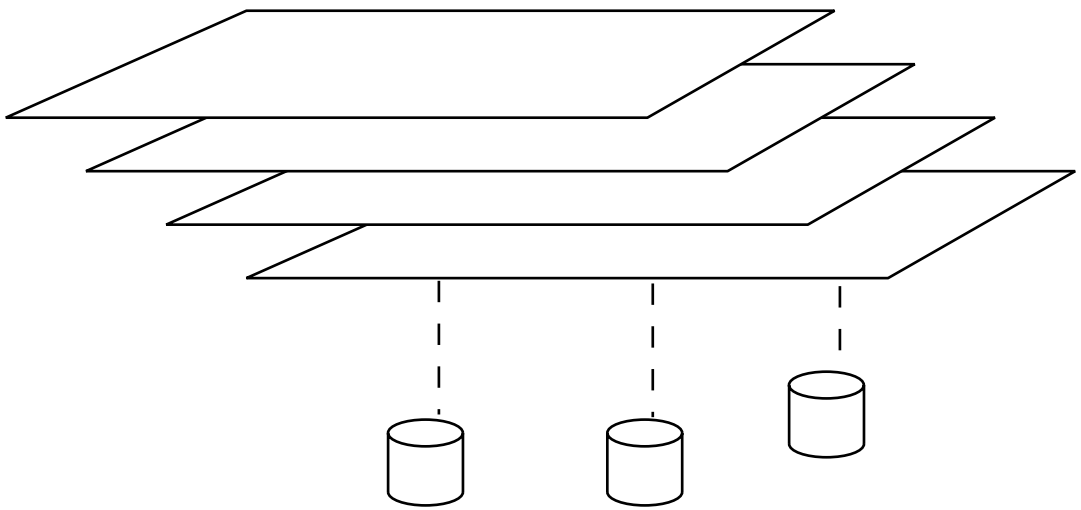
Reflection as defined means that the system can inspect and change itself. Some systems allow **only inspection**.

An introspective system is a meta system that has itself as base system.

- Not necessarily causally connected

Implementing Reflection

- Use towers of interpreters
- Analogy:



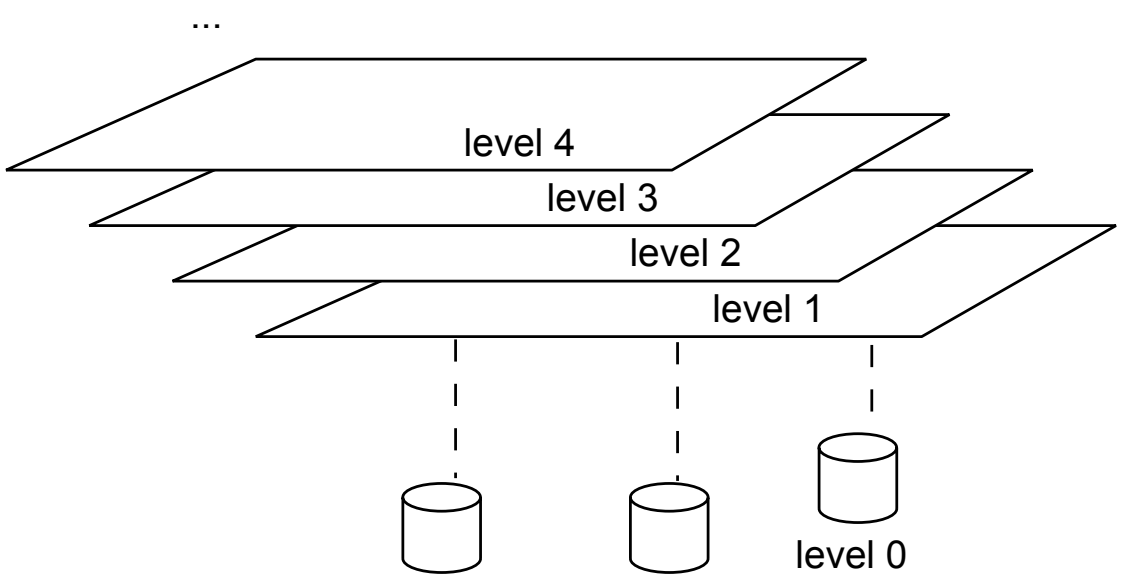
language = physics
language = electronics
language = C
language = Smalltalk

a Smalltalk program

Reflective Tower of Interpreters

The original model of reflection defined for 3-LISP [Smith 1982] is based on **meta-level interpretation**:

- The program (level 0) is interpreted by an interpreter (level 1).
- The interpreter (level 1) is interpreted by a metainterpreter (level 2).
- Leading to a (potentially infinite) tower of interpreters each defining the semantics of the program (the interpreter) it interprets.



Reflective Architecture

- A tower of interpreters is too slow in practice.
- To enable reflection in mainstream languages like CLOS, Smalltalk, Java or Ruby, the tower of interpreters is replaced with a reflective architecture consisting of **meta-objects** that model and control the reified concepts of the computational system.
- The interfaces of the **meta-objects** constitute a **meta-object protocol (MOP)** [Kiczales and Rivieres 1991].

Reflection: Uses

Computational reflection allows to solve problems elegantly, that are otherwise

- handled on an ad-hoc basis,
- hard or impossible to achieve.

Such as…

- Extending a programming language (PL design)
- Adapting a language to the application domain (DSLs)
- Building programming environments
- Advanced software development tools (debuggers, program analysers, …)
- Building knowledge and learning systems
- Dynamically self-adapting and self-optimising applications

Metaprogramming & Reflection: Summary

With metaprogramming, one system reasons about another

- Base and meta language (can be the same language or different)
- One system is meta for the other

With reflection, the system reasons about itself

- Only one system is involved
- Consequently only one language is involved
- The language is said to be reflective (or have some reflective capabilities)

This Week's Reading Assignment

- Bracha, G., and Ungar, D.
  Mirrors: design principles for meta-level facilities of object-oriented programming languages. In ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (New York, NY, USA, 2004), OOPSLA '04, ACM Press, pp. 331–344.

- Download link:
  `http://dl.acm.org/citation.cfm?id=1029004`
- Freely accessible from within the TUD campus network

References I

Keshav, S. (2007). “How to Read a Paper”. In: **ACM SIGCOMM Computer Communication Review** 37.3, pp. 83–84.

Kiczales, Gregor and Jim Des Rivieres (1991). **The Art of the Metaobject Protocol**. Cambridge, MA, USA: MIT Press.

Smith, Brian Cantwell (1982). “Procedural Reflection in Programming Languages”. PhD thesis. Massachusetts Institute of Technology, Laboratory for Computer Science.