Component-Based Software Engineering (CBSE)
10. Introduction

Dr.-Ing. Sebastian Götz
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de/teaching/cbse
04.04.2018

1. Basics of Composition Systems
2. Black-Box Composition Systems
3. Gray-Box Composition Systems

Based on Slides by Prof. Uwe Aßmann

Obligatory Reading

- McIlroy, M. D. (1968). **Mass produced software components.**
  In: Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany. 1968.
  http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF (Pages 79-87)

- Dijkstra, E.W. (1982). **On the Role of Scientific Thought.**
  In: Selected Writings on Computing: A personal Perspective. Texts and Monographs in Computer Science. Springer, New York, NY
  https://www.cs.utexas.edu/users/EWD/ewd04xx/EWD447.PDF

The Destructive Power of Ill-Used Components: The Ariane 5 Launcher Failure

June 4th 1996

Total failure of the Ariane 5 launcher on its maiden flight

The following slides are from

Ian Summerville, Software Engineering

Credit: DLR/Thilo Kranz (CC-BY 3.0) 2013
http://commons.wikimedia.org/wiki/File:Ariane_5ES_with_ATV_4_on_its_way_to_ELA-3.jpg

http://www.astronews.com/news/artikel/2002/12/0212-009.shtml

Ariane 5 Launcher Failure

- Ariane 5 can carry a heavier payload than Ariane 4
  - Ariane 5 has more thrust, launches *steeper*
- 37 seconds after lift-off, the Ariane 5 launcher lost control
  - Incorrect control signals were sent to the engines
  - These swivelled so that unsustainable stresses were imposed on the rocket
  - It started to break up and self-destructed
- The system failure was a software failure

The Problem of Component Reuse

- The attitude and trajectory of the rocket are measured by a computer-based inertial reference system
  - This transmits commands to the engines to maintain attitude and direction
  - The software failed and this system and the backup system shut down
- Diagnostic commands were transmitted to the engines
  - ..which interpreted them as real data and which swivelled to an extreme position
- Technically: Reuse Problem
  - Integer overflow failure occurred during converting a 64-bit floating point number to a signed 16-bit integer
  - There was no exception handler
  - So the system exception management facilities shut down the software

Software Reuse Error

- The erroneous software component (Ada-83) was reused from the Ariane 4 launch vehicle.
- The computation that resulted in overflow was not used by Ariane 5.
- Decisions were made in the development
  - Not to remove the facility as this could introduce new faults
  - Not to test for overflow exceptions because the processor was heavily loaded.
  - For dependability reasons, it was thought desirable to have some spare processor capacity
- Why not in Ariane 4?
  - Ariane 4 has a lower initial acceleration and build up of horizontal velocity than Ariane 5
  - The value of the variable on Ariane 4 could never reach a level that caused overflow during the launch period.
  - The contract was not re-proven for Ariane-5
  - There was also no run-time check for contract violation in Ariane-5

10.1. Basics of Composition Systems

- Component-based software engineering is built on **composition systems.**
- A composition system has
  - a component model,
  - a composition technique, and
  - a composition language.

Software is Everywhere

- Cars, Planes, Ships, Factories, Houses, Watches, …
- Unimaginable amount of software is spread across the globe
- How to keep this manageable?

The Power of Components

Goals

- Component-based software engineering (CBSE) is the generalization of object-oriented software engineering (OOSE)
  - Understand how to **reuse** software
  - Component models are the basis of all **engineering**
- Understand how large software systems are build
- What is a *composition system?*
  - The difference of component-based and composition-based systems
  - What is a composition operator? composition expression? composition program? composition language?
- Understand the difference between **graybox** and **blackbox** systems
- Understand the ladder of composition systems
  - Understand the criteria for comparison of composition systems

Motivation for Component-Based Development

- Component-Based Development is the basis of *all* engineering
  - Development by “divide-and-conquer” (Alexander the Great)
  - Well known in other disciplines
    - Mechanical engineering (e.g., German VDI 2221)
    - Electrical engineering
    - Architecture

- “Make, reuse or buy” decisions (reuse decisions):
  - Outsourcing to component producers (Components off the shelf, COTS)
    - Reuse of partial solutions
    - Easy configurability of the systems: variants, versions, product families

- Scaling business by Software Ecosystems
  - Component models and composition systems are the technical basis for all modern software ecosystems: Linux, Eclipse, AutoSAR, Android, openHAB, ROS, …

Mass-produced Software Components

- Mass Produced Software Components [McIlroy, Garmisch 68, NATO conference on software engineering]:
  - Every ripe industry is based on components, to manage large systems
  - Components should be produced in masses and composed to systems afterwards

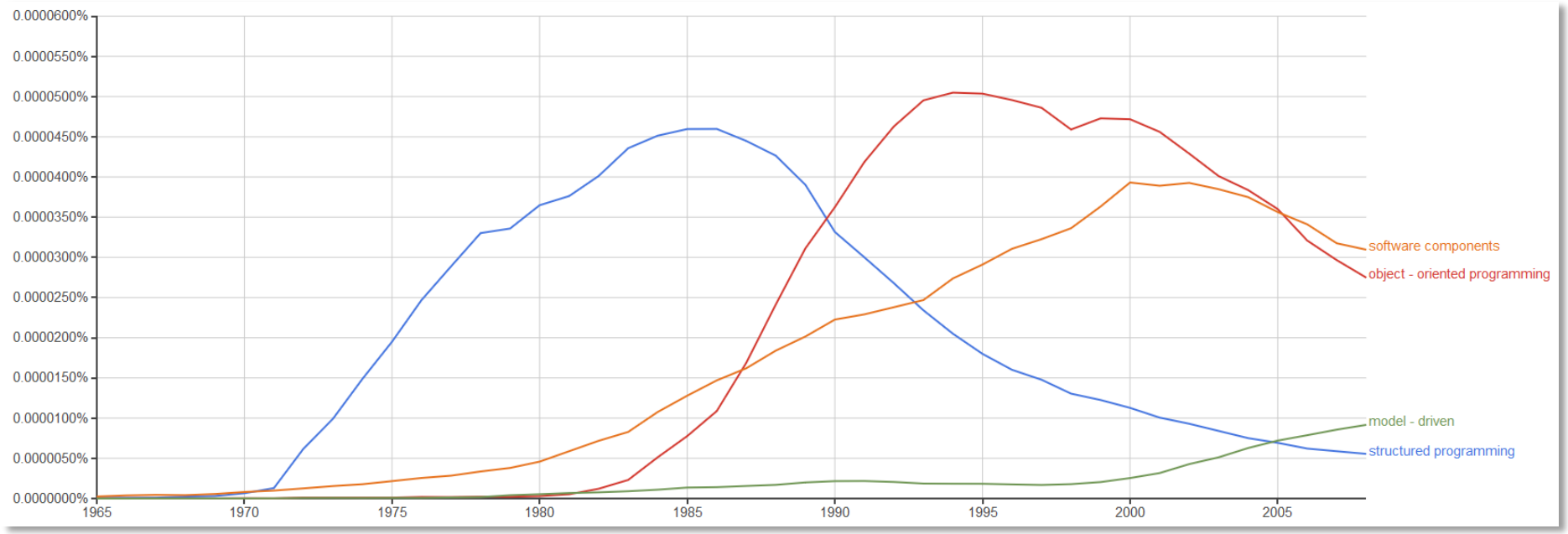
In the phrase “mass production techniques”, my emphasis is on “techniques” and not on “mass production” plain. Of course mass production, in the sense of limitless replication of a prototype, is trivial for software.

But certain ideas from industrial technique I claim are relevant.
- The idea of **subassemblies** carries over directly and is well exploited.
- The idea of **interchangeable parts** corresponds roughly to our term “modularity”, and is fitfully respected.
- The idea of **machine tools** has an analogue in assembly programs and compilers.

Yet this fragile analogy is belied when we seek for analogues of other tangible symbols of mass production.
- There do not exist manufacturers of standard parts, much less catalogues of standard parts.
- One may not order parts to individual specifications of size, ruggedness, speed, capacity, precision or character set.

Mass-produced Software Components

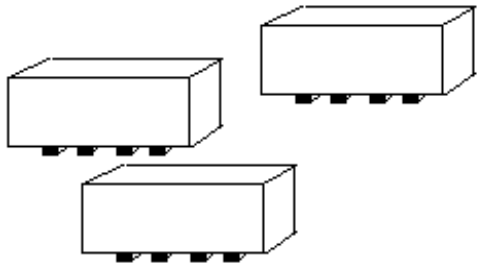
- Later McIlroy was with Bell Labs,
  - ..and invented pipes, diff, join, echo (UNIX).
  - Pipes are still today the most employed component system!
- Where are we today?



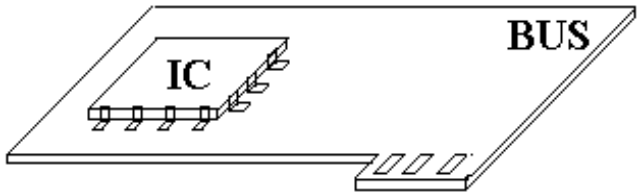
(Google Ngram Viewer)

“Real” Component Systems

- Lego
- Square stones
- Building plans
- IC’s
- Hardware bus



- How do they differ from software?



Definitions of Software Components

A software component is a unit of composition
- with contractually specified interfaces
- and explicit context dependencies only.

A software component
- can be deployed independently and
- is subject to composition by third parties.

(Clemens Szyperski)

A reusable software component is a
- logically cohesive,
- loosely coupled module
- that denotes a single abstraction. (Grady Booch)

A software component is a static abstraction with plugs.

(Nierstrasz/Dami)

What is a Software Component?

- A component is a container with
  - Hidden inner
  - Public outer interface
  - Stating all dependencies explicitly
- Example: a method
  - Public outer interface: return type
  - Explicit dependencies: parameters
  - Hidden inner: method body

```
int add(int a, int b) {
  return a+b;
}
```

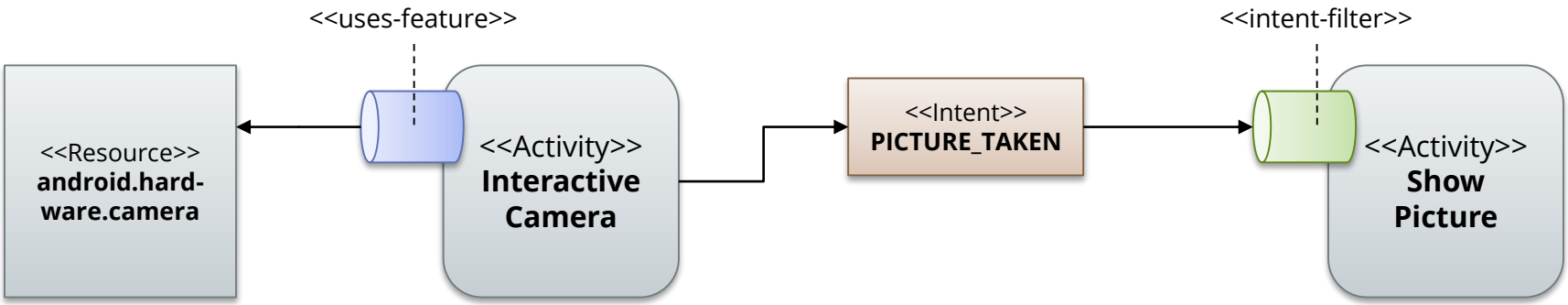
- A component is a reusable unit for composition!
- A component underlies a component model
  - that fixes the abstraction level
  - that fixes the grain size (widget or OS?)
  - that fixes the time (static or runtime?)

What is a Software Component?

- A component is a container with
  - Hidden inner
  - Public outer interface
  - Stating all dependencies explicitly

Learn more here:
https://developer.android.com/topic/libraries/architecture/index.html

- Example: Android Activity Components
  - Interfaces specified in **`AndroidManifest.xml`**
  - Public outer interface:
    - *Intent-filter*, lists intents the activity shall respond to
  - Explicit dependencies:
    - *Uses-feature*, to list hard- and software features required by your activity
  - Hidden inner:
    - Implementation of Activity class

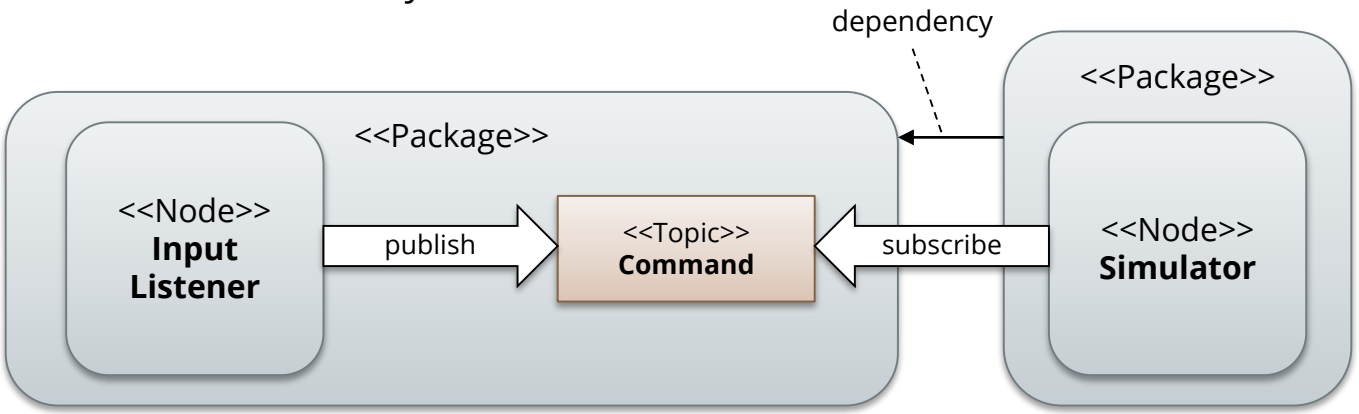


What is a Software Component?

Learn more here: http://wiki.ros.org/ROS/Tutorials/

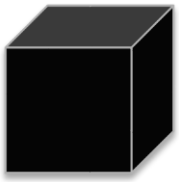
- A component is a container with
  - Hidden inner
  - Public outer interface
  - Stating all dependencies explicitly

- Example: Robot Operating System (ROS) Components
  - Public outer interface:
    - *ROS Nodes* are executables. Their outer interface is defined by start-up parameters.
  - Explicit dependencies:
    - *ROS Packages (package.xml).* Dependencies are specified per package.
    - *ROS Nodes* communicate via topics
  - Hidden inner:
    - Implementation in C++ or Python.

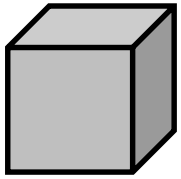


What Is a Component-Based System?

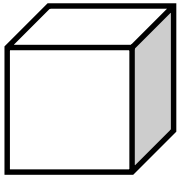
- A component-based system has the following **divide-and-conquer feature**:
  - A component-based system is a system in which a major relationship between the components should be **tree-shaped or reducible**.
  - In other words: components are hierarchically organized
- Consequence: the entire system can be reduced to one abstract node
  - at least along the structuring relationship
- Because of the divide-and-conquer property, component-based development is attractive.
  - However, we have to choose the structuring relation and the composition model
- Mainly, 2 types of component models are known
  - Modular decomposition (blackbox)
  - Separation of concerns (graybox)



everything hidden



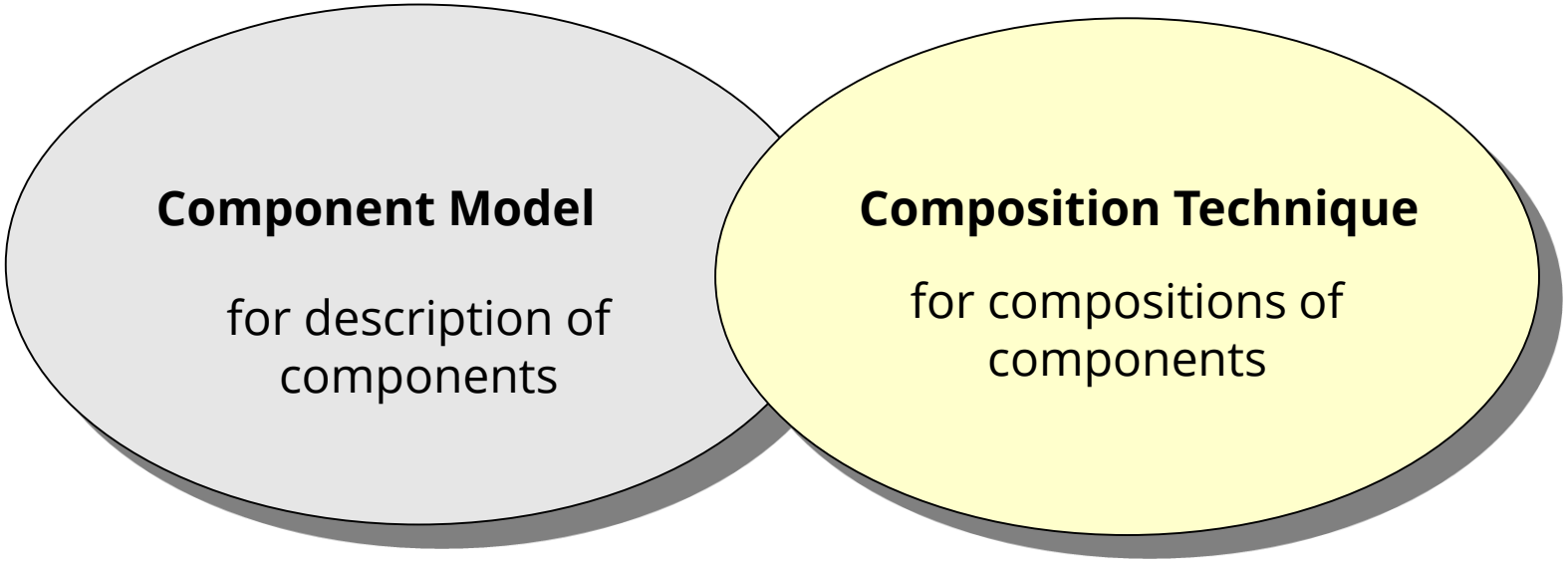
parts visible



everything visible

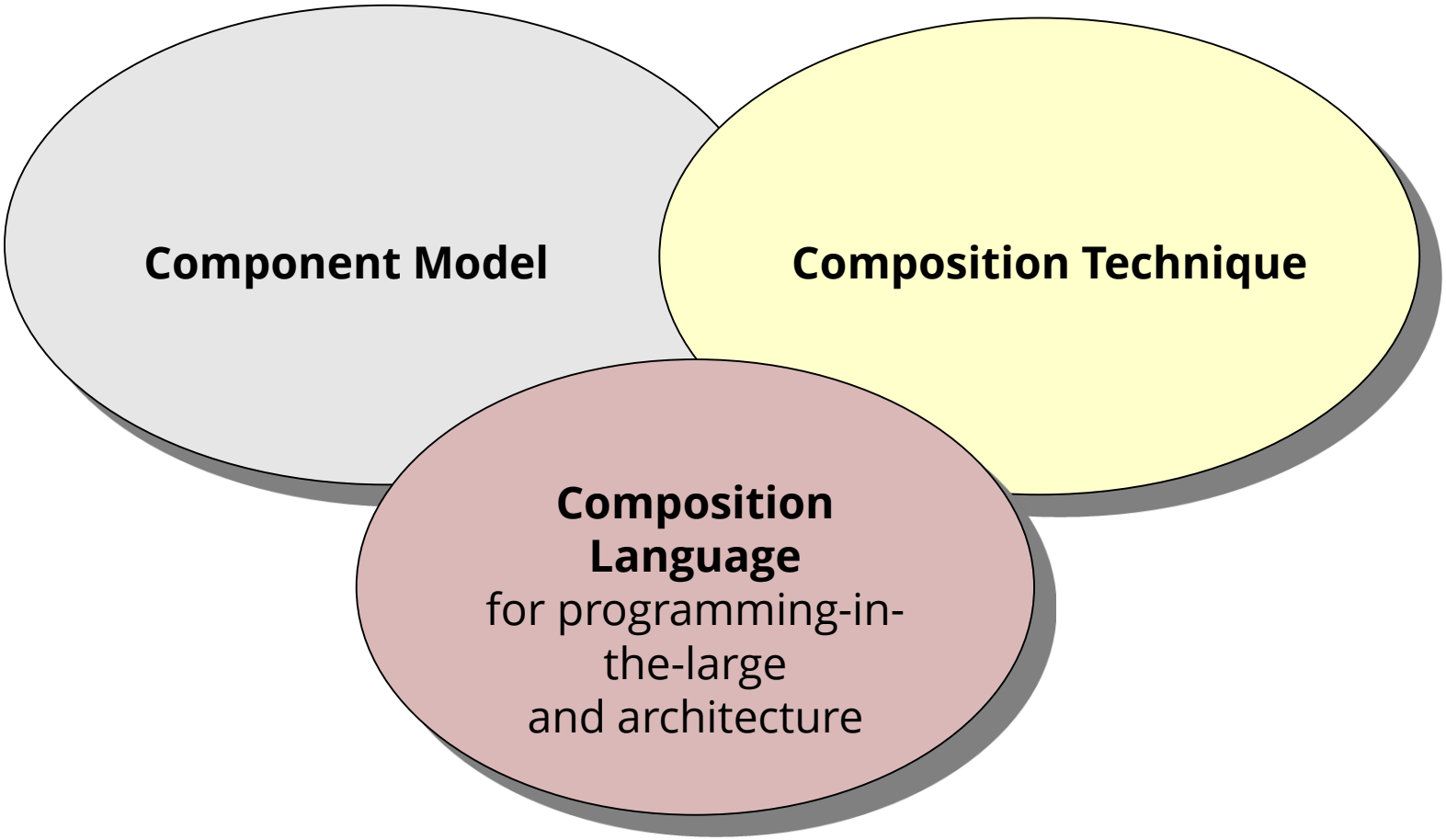
Component Systems

- We call a technology in which component-based systems can be produced a *component system.*
- A component system has



Composition Systems

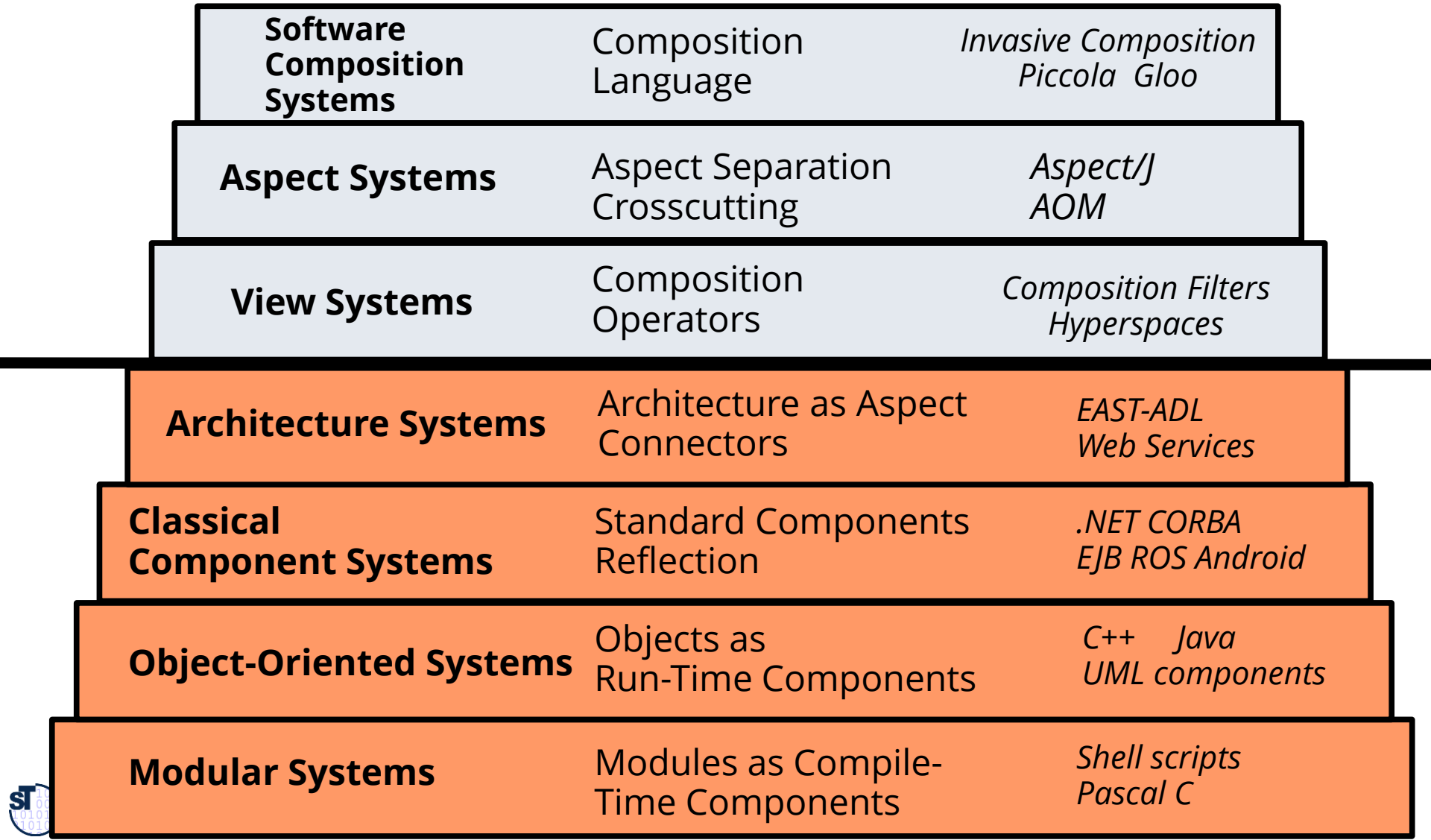
- A composition system has



Composition Systems

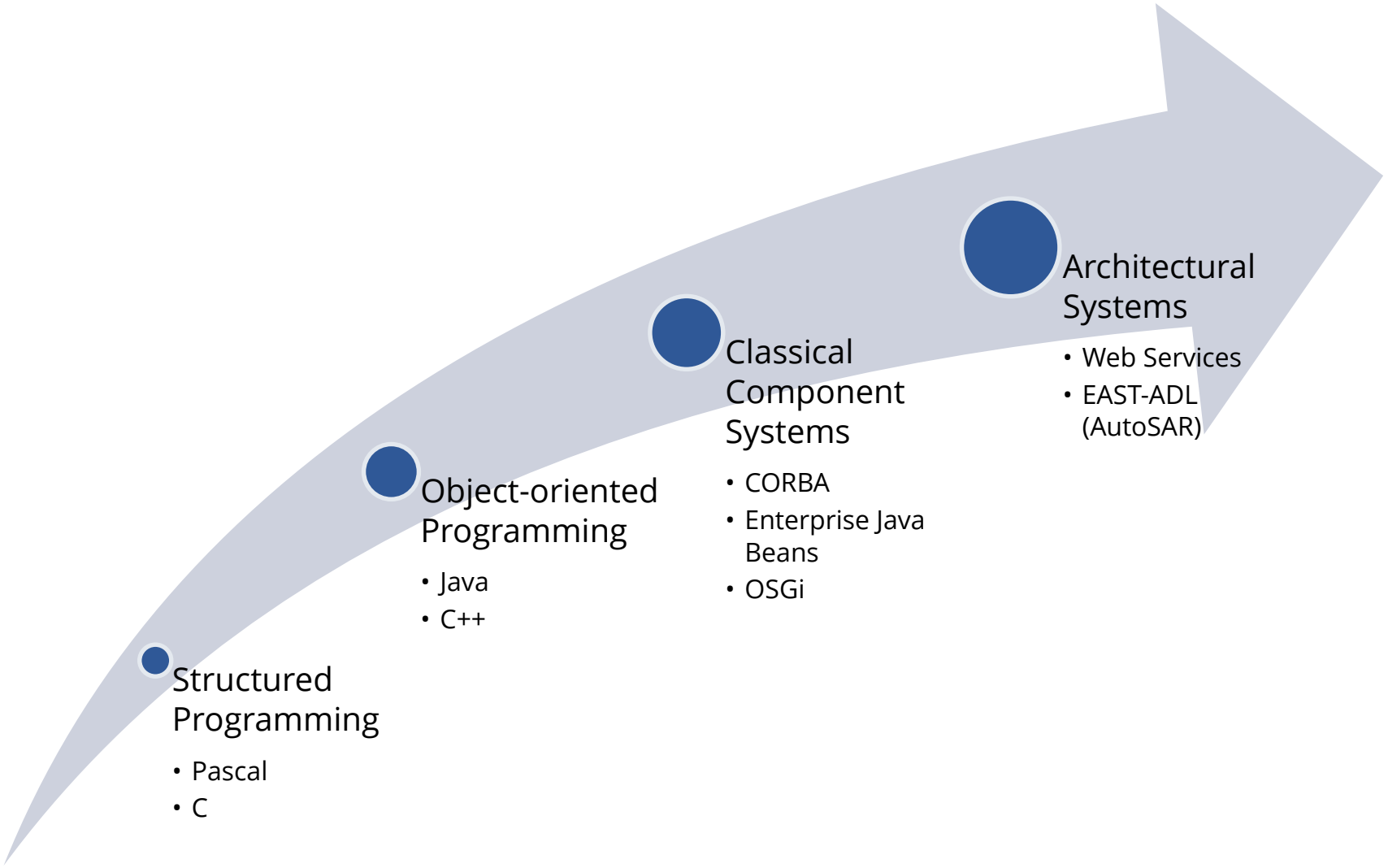
- Component Model:
  - How do components look like? What types of components are provided?
  - Secrets, interfaces, substitutability
- Composition Technique
  - How are components plugged together, composed, merged, applied?
  - Composition time (Compile-time, Deployment-time, Runtime)
- Composition Language
  - How are compositions of large systems described?
  - How are system builds managed?

The Ladder of Composition Systems



10.2 Black-box Composition Systems

Black-box Composition



Structured Programming

- Component Model
  - Procedures (alias Methods, Functions, Operations)
  - Variables (i.e., data)
- Composition Technique
  - Procedure calls (data exchanged via parameters and return values)
- Composition Language
  - none

Example: Pascal

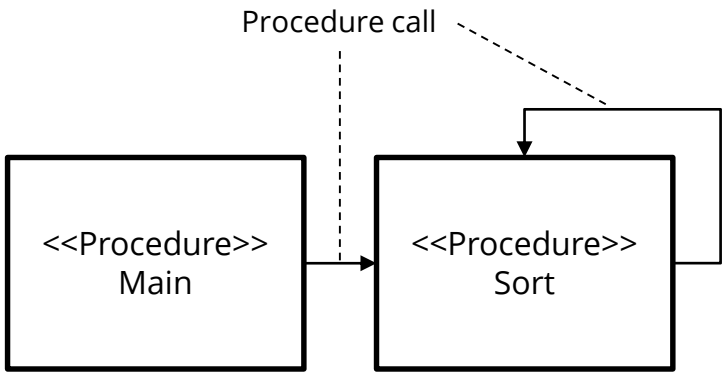
```
PROGRAM ProgramName;

VAR
   VariableName : VariableType;
   ...

PROCEDURE ProcedureName;
  variable declarations
BEGIN
    ...
END;

FUNCTION FunctionName(variableList): VariableType;
   variable declarations
BEGIN
   ...
   FunctionName := some expression
   ...
END;

BEGIN
    ...
END.
```



Object-oriented Programming

- Component Model
  - Classes, which capsule
    - Methods (alias Procedures, Functions, Operations)
    - Variables (i.e., data)
  - Objects
- Composition Technique
  - Method calls (Delegation)
  - Inheritance (Polymorphism)
- Composition Language
  - none

Example: Java

```
class HelloWorld {
  protected String text;

  public HelloWorld() {
    text = "Hello World.";
  }

  public void sayHello() {
    System.out.println(text);
  }
}

class HelloCBSE extends HelloWorld {
  public HelloCBSE() {
    text = "Hello CBSE!";
  }
}

class Client {
  public static void main(String args[]) {
    HelloWorld a = new HelloCBSE();
    a.sayHello();
  }
}
```

Classical Component-based Systems

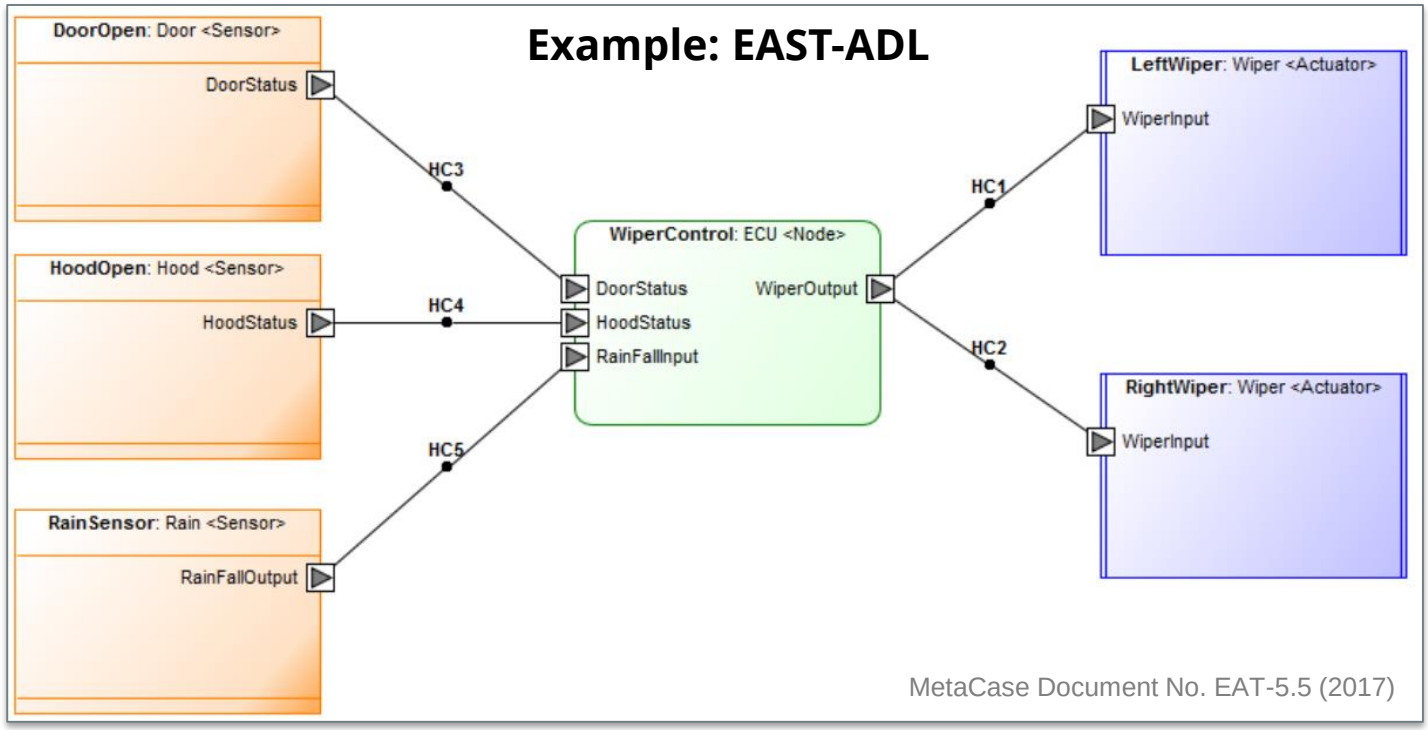
- Component Model
  - Standardized components, typically defined via interfaces
  - Components are technology-specific:
    - Android: *Apps*, Activities, Services, Broadcast Receivers, Content providers
    - Robot OS: *Packages*, Nodes, Services
    - Enterprise Java: *Enterprise archive (EAR)*, Entity/Session/Message-driven Beans
    - OSGi: *Bundles*, Services

- Composition Technique
  - Interplay of components managed by **middleware** services
  - Composition techniques are technology-specific, too:
    - Android: Intents, Android Interface Definition Language (AIDL)
    - Robot OS: Topics
    - Enterprise Java: Remote Method Invocation (RMI)
    - OSGi: Package ex-/import, Method calls

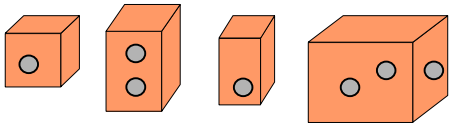
- Composition Language
  - none

Architectural Systems

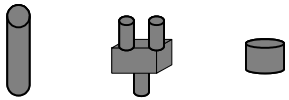
- Component Model
  - Standardized components, typically defined via interfaces
- Composition Technique
  - Interconnection of components via **connectors**
  - Connectors encapsulate communication
- Composition Language
  - Architecture Description Languages



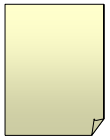
Black-box Composition Systems



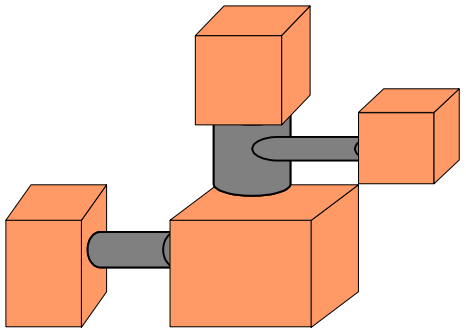
Components



Connectors

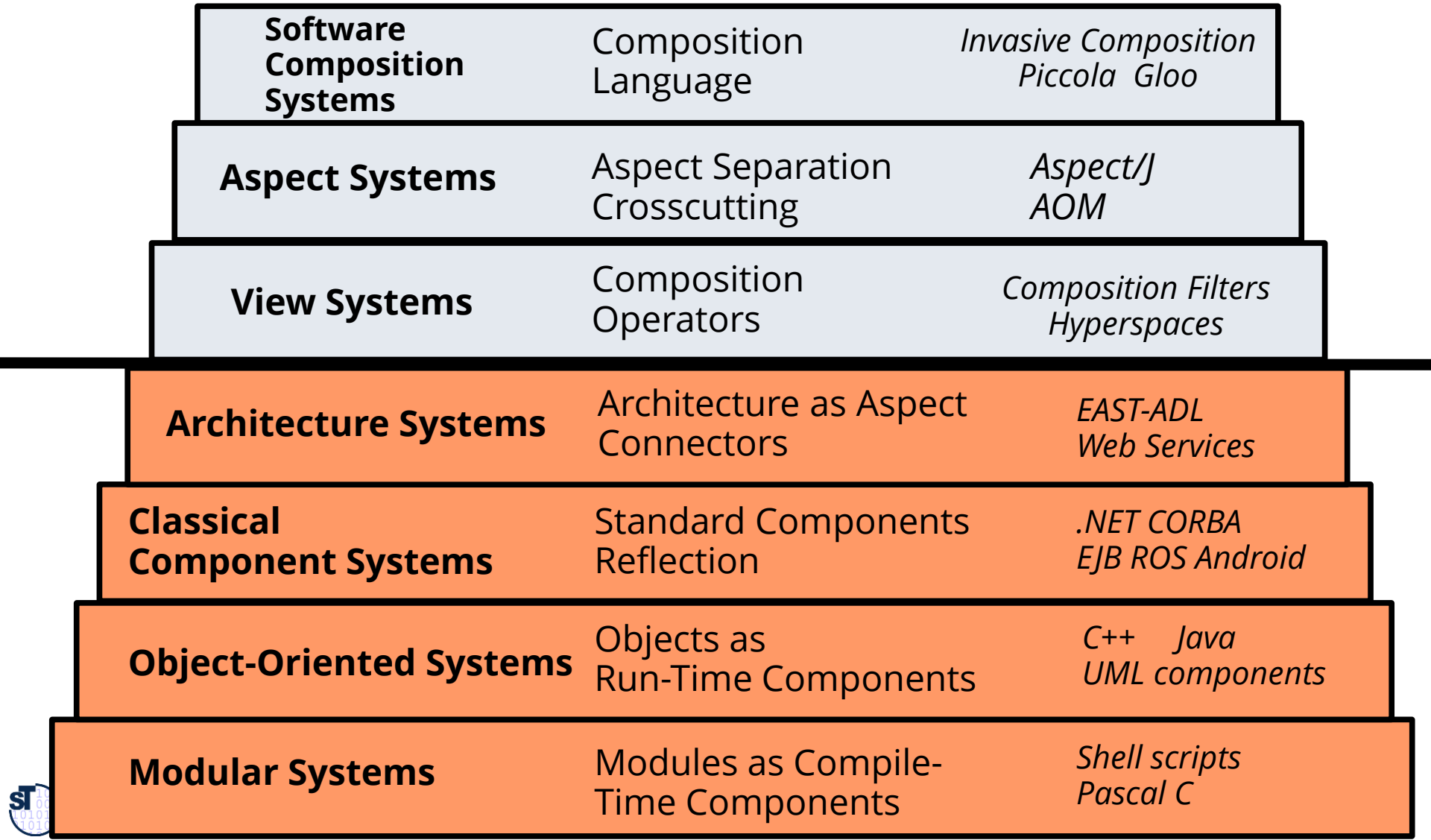


Composition recipe



Component-based applications

The Ladder of Composition Systems



10.3 Gray-box Composition Systems

Beyond Black-box Components

- Black-box component systems fully hide what's inside:
  - Components (e.g., implementation language)
  - Connectors (i.e., how components communicate)

- But, this strict decomposition of systems is in contrast to the way we think:
  - Dijkstra E.W. (1982) **On the Role of Scientific Thought**. In: Selected Writings on Computing: A personal Perspective. Texts and Monographs in Computer Science. Springer, New York, NY

> „Let me try to explain to you, what to my taste is characteristic for all intelligent thinking. It is, that one is willing to study in depth **an aspect of one's subject matter in isolation** for the sake of its own consistency, all the time knowing that one is occupying oneself **only with one of the aspects**."
>
> (Edgar Dijkstra)

- We need a more elaborate way of decomposition: **separation of concerns**

Aspects in Architecture

Electricity | Water | Air conditioning | Statics | ...

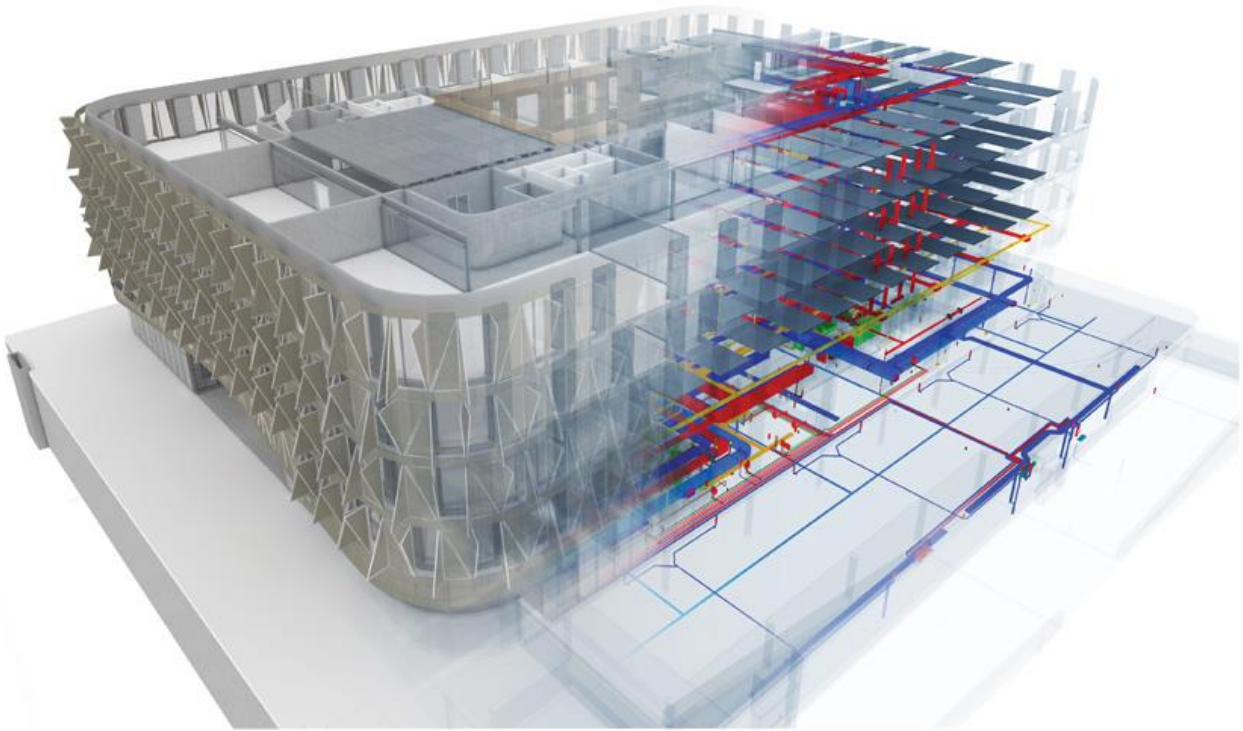
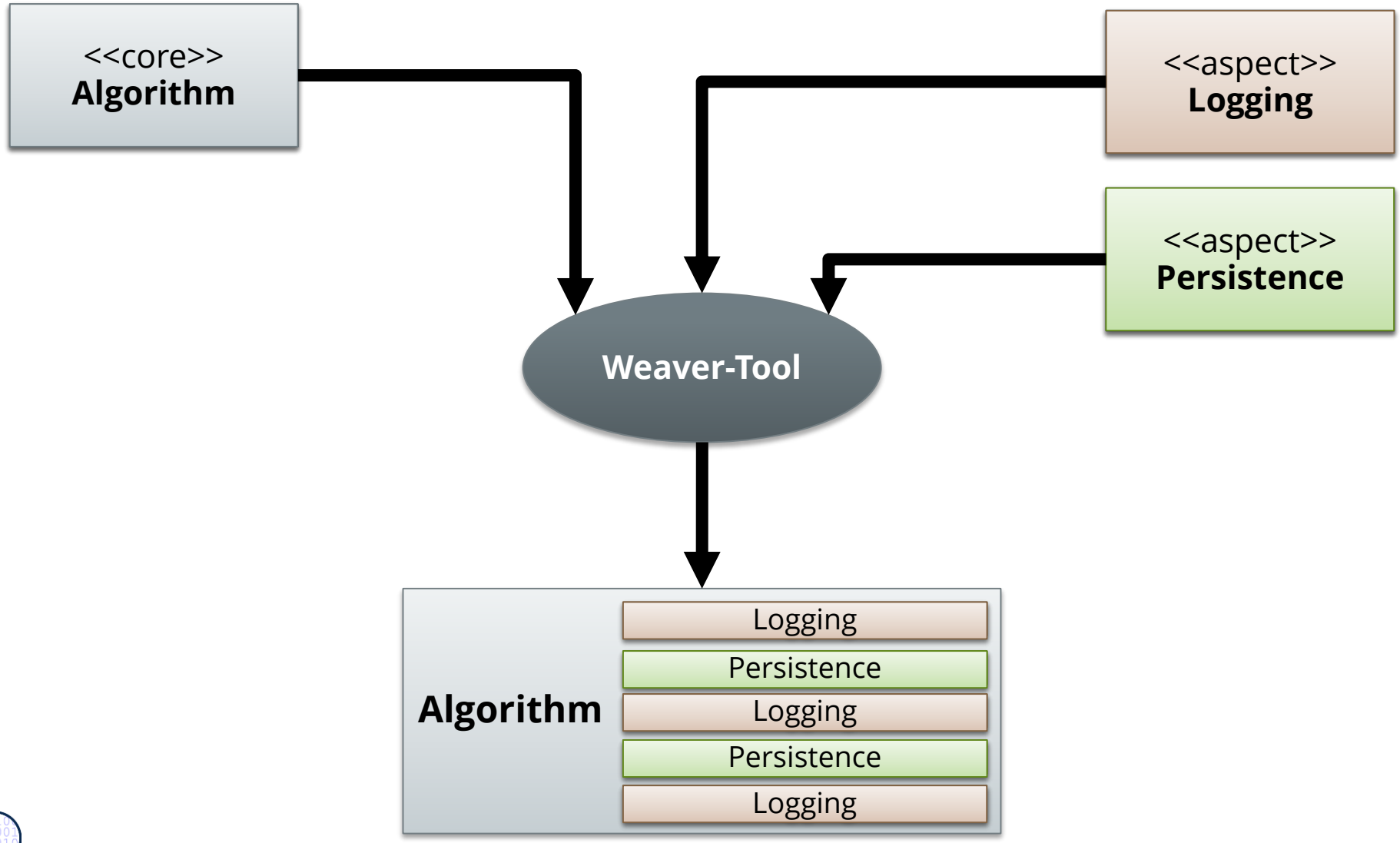


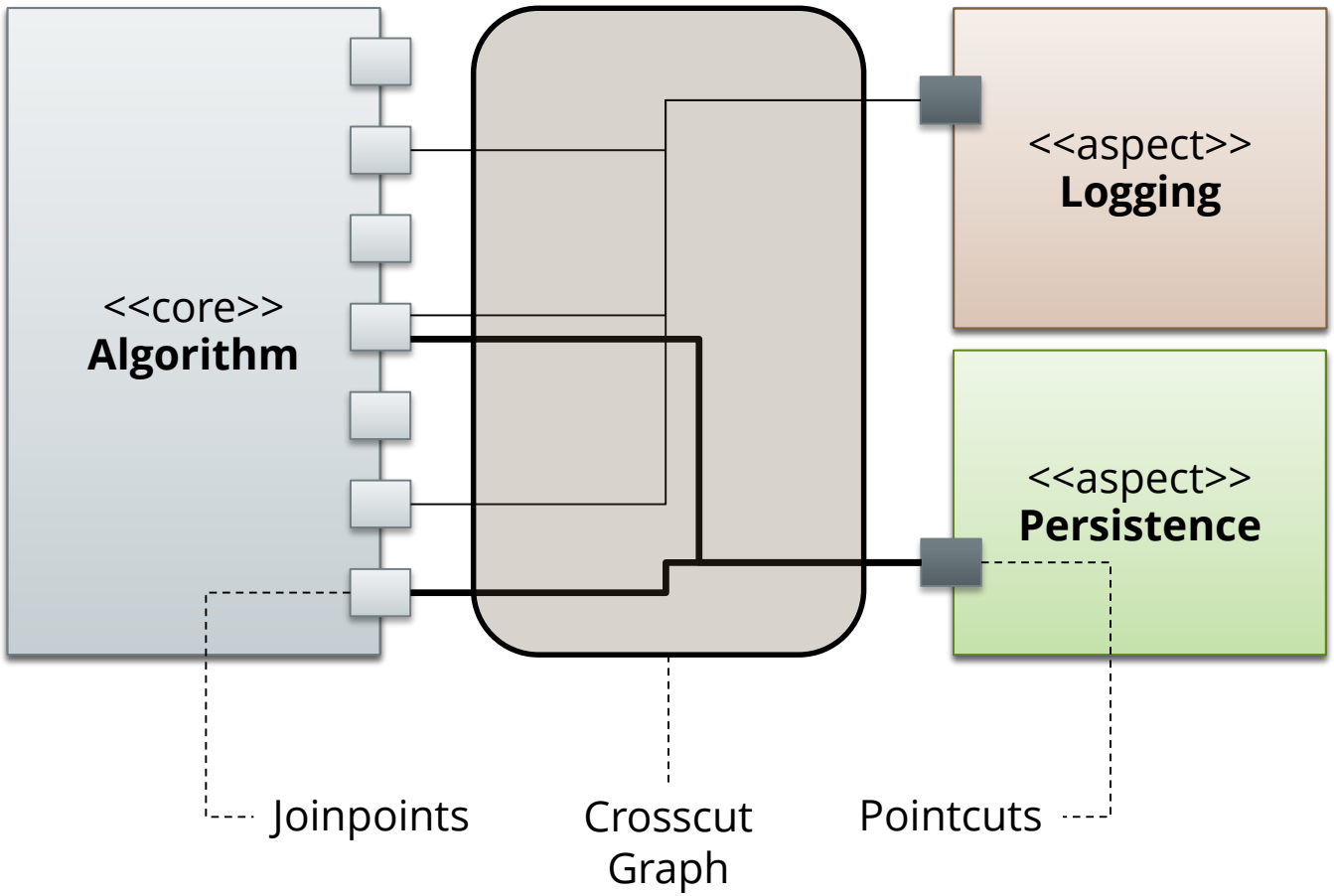
Image courtesy of ITTEN-BRECHBÜHL AG, BERN, SWITZERLAND | HEADQUARTER SCOTT SPORTS SA, GIVISIEZ FR, SWITZERLAND

Aspects in Software

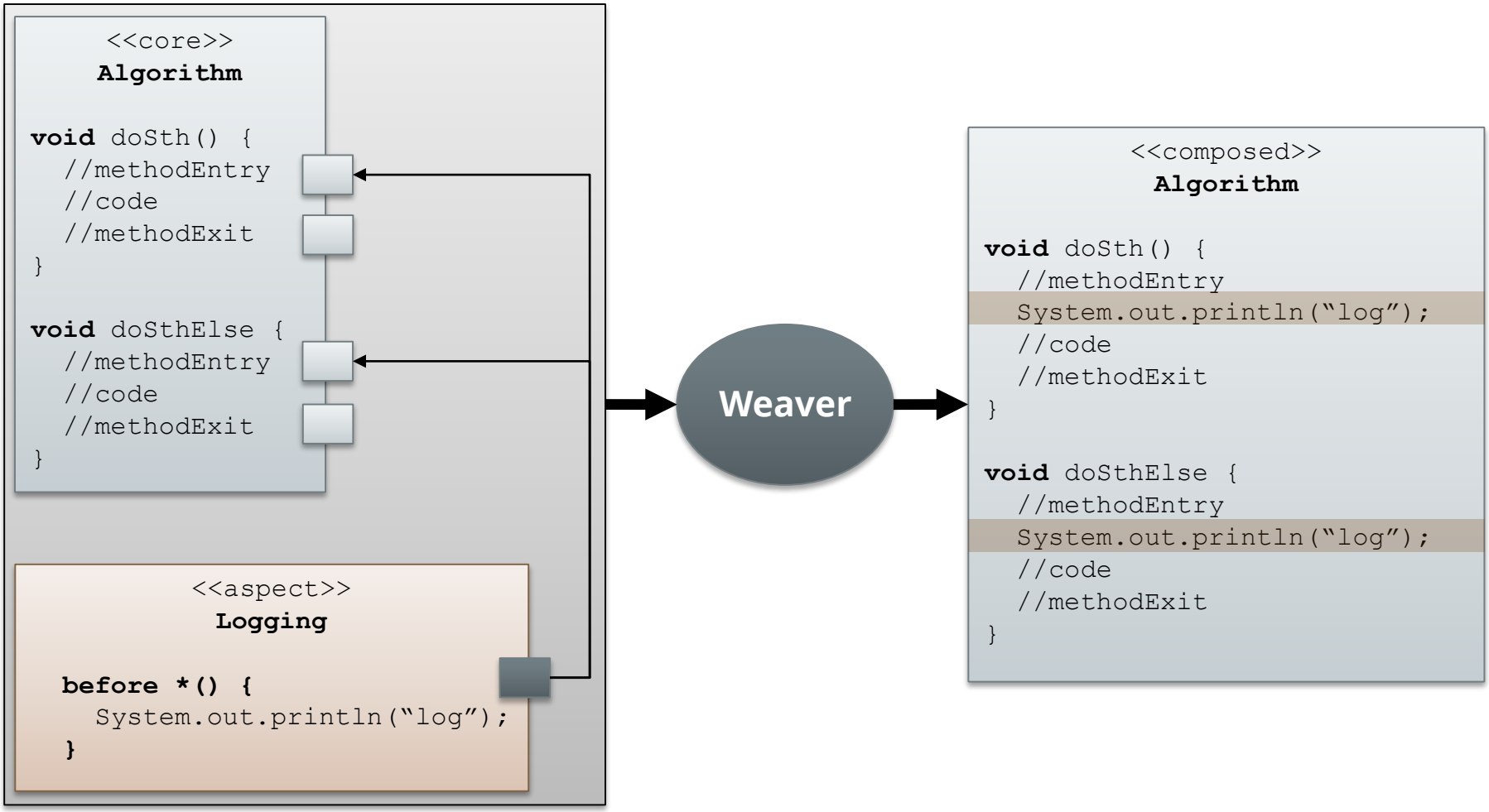


Aspects in Software

- Aspects are **Crosscutting**:
  - **scattered** across and **tangled** in the core



Aspects in Software



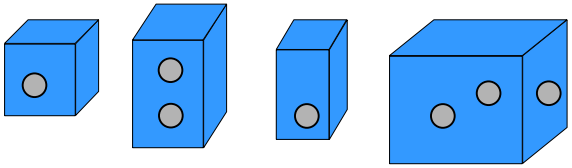
Aspects in Software

- Component Model
  - Core and aspects
    - Core contains joinpoints
    - Aspects specify pointcuts, which reference sets of jointpoints
- Composition Technique
  - Weaving (code composition)
- Composition Language
  - Usually implicit (i.e., part of the programming language)
- Besides Aspect-oriented Programming, we will investigate:
  - Composition Filters [Aksit,Bergmans]
  - Hyperspace Programming [Ossher et al., IBM]
  - Invasive software composition (ISC) [Aßmann]

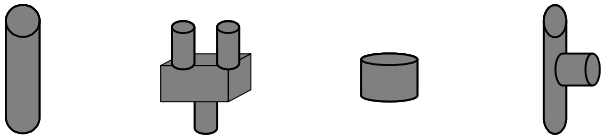
Composition Languages in Composition Systems

- Composition languages describe the structure of the system in-the-large (“programming in the large”)
  - Composition programs combine the basic composition operations of the composition language
- Composition languages can look quite different
  - Imperative or rule-based
  - Textual languages
    - Standard languages, such as Java
    - Domain-specific languages (DSL) such as Makefiles or ant-files
  - Graphic languages
    - Architectural description languages (ADL)
- Composition languages enable us to describe large systems

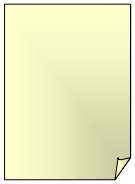
Grey-Box Composition Systems



Grey-box Components

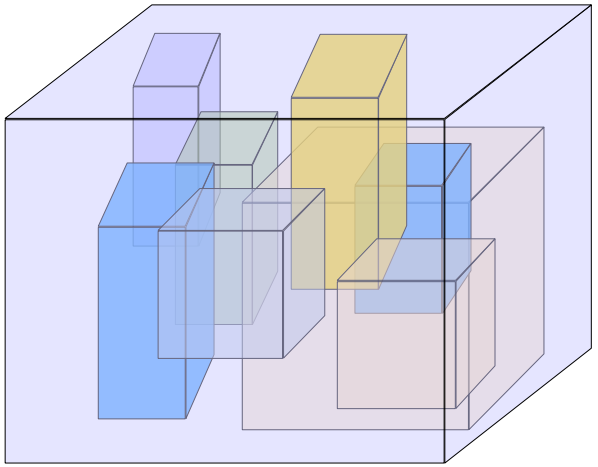


Composition Operators



Composition Recipe

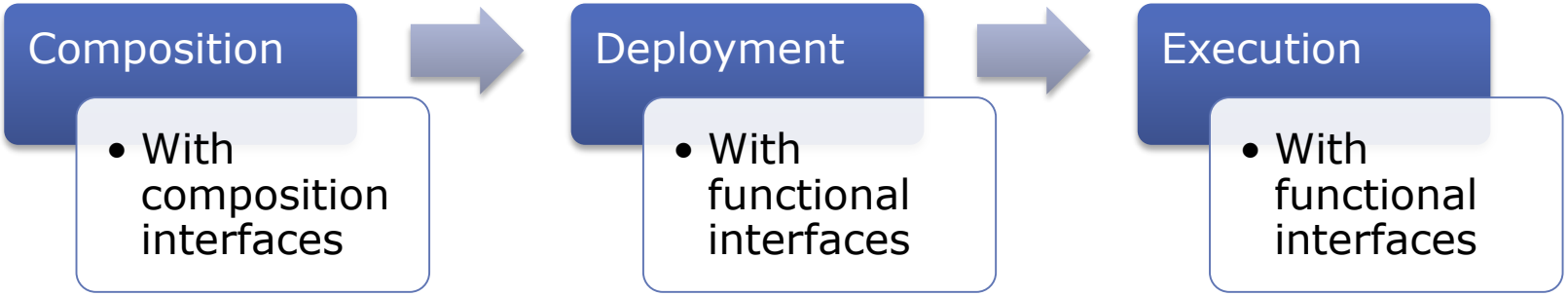
Invasive Software Composition



System Constructed with an Invasive Architecture

Conclusions for Composition Systems

- Components have a ***composition interface***
- Composition interface is different from functional interface
  - The composition is running usually *before* the execution of the system
  - From the composition interface, the functional interface is derived
- System composition becomes a new step in system build



The End / Course Outline