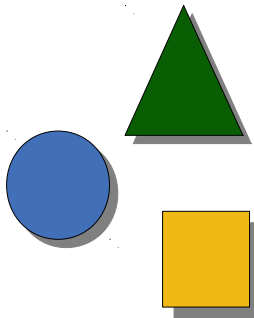


52. Staged Software Architectures with Staged Composition

Prof. Dr. Uwe Aßmann
Technische Universität
Dresden

Institut für Software- und
Multimediatechnologie
Version 13-1.0, 13.07.13

- 1) Web programming considered harmful
 - 1) Problem 1: Untyped template expansion
 - 2) Problem 2: Staging
 - 3) Problem 3: Spaghetti Code
- 2) Staged Architectures

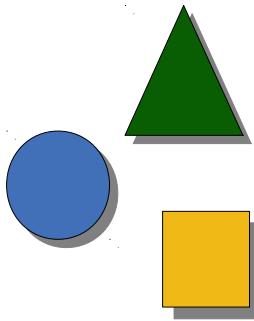


A Staged Architecture from Nature



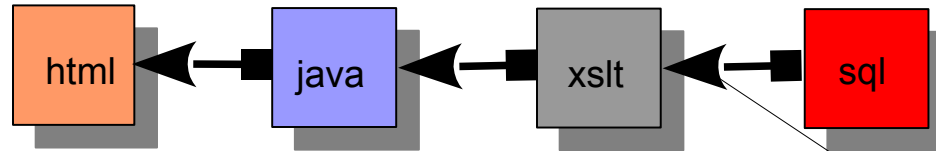
Prof. U. Aßmann, CBSE

52.1 Web Programming Considered Harmful



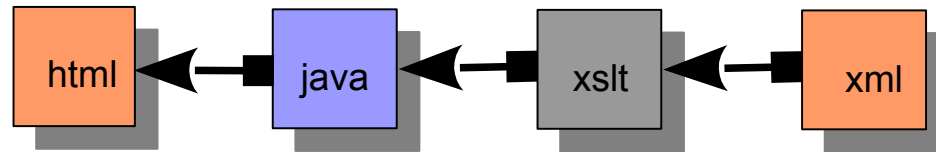
Web Programming: Staged, Untyped Template Expansion

Prof. U. Alsmann, CBSE



Stage 1

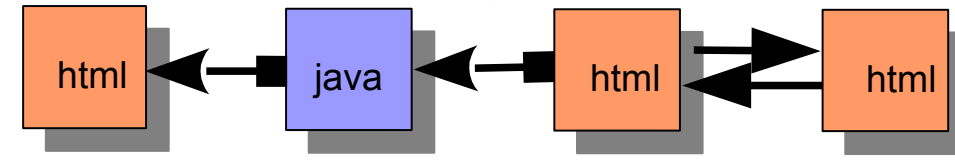
Sql expansion



Embedding after Expansion

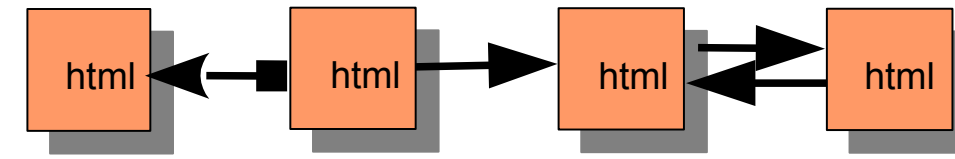
Stage 2

Xslt transformation



Stage 3

java expansion



Stage 4

Runtime execution or interpretation of data

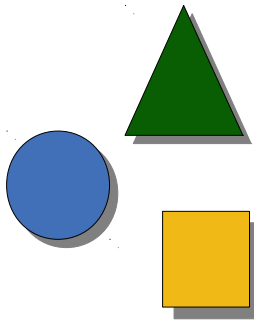
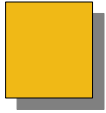




Problems of Web Programming

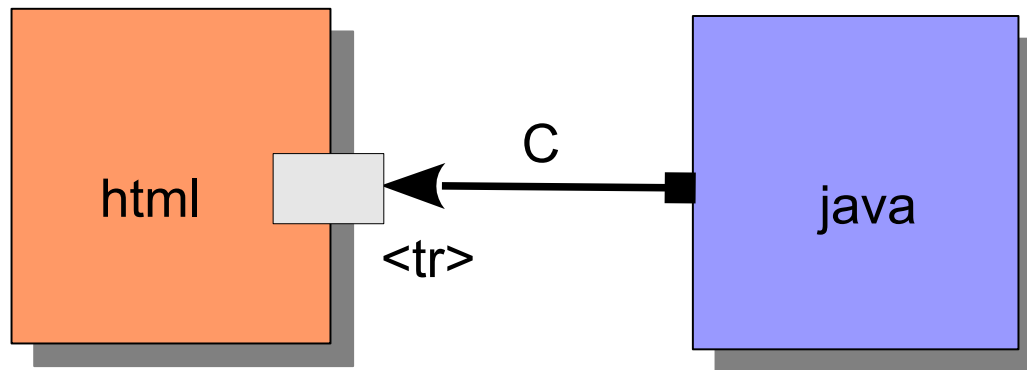
- ▶ Untyped extensions of templates
 - Error-prone
- ▶ Comprehension very difficult, due to the different stages
- ▶ Spaghetti-code-like programs
 - Scripts mixed with templates
 - Only valuable for programming-in-the-small

52.1.1 Problem 1: Untyped Template Expansion



Type-Safe Template Expansion

- ▶ How can you be sure that table rows are filled in?



- ▶ Answer: in an invasive document composition system, the type checker of the invasive composition program will tell you, when checking the composition step C



Universality of Invasive Composition

- ▶ Invasive composition only depends on a metamodel of the language
 - New hook and slot models can be derived from any language
 - Typing controls the composition of artifacts
- ▶ Hence, the method is *universal*
- ▶ and can be applied for typed document composition
- ▶ See www.reuseware.org, the universal invasive composition environment,
 - Can be tailored for text-based and diagrammatic languages
 - OpenOffice
 - XML dialects
 - EMF-based

Elements of Web Composition Systems

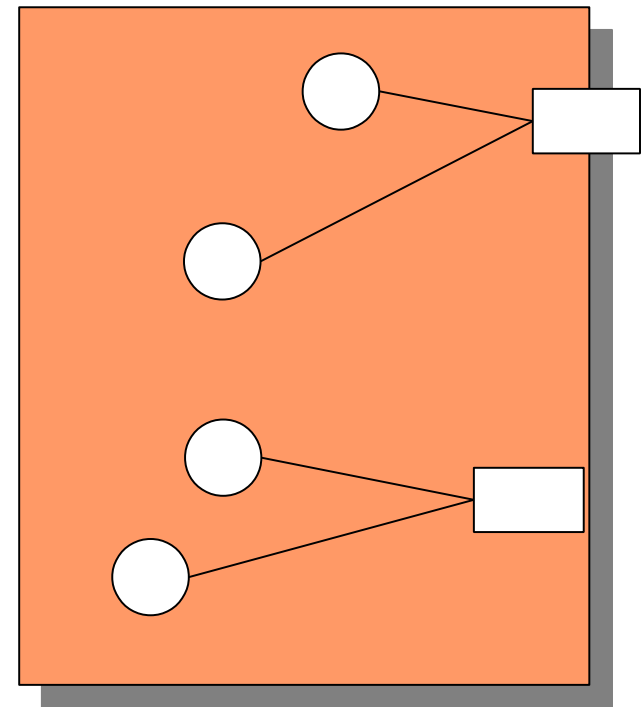
Component Model
XML templates

Composition Technique
Parameterization
Extension
In-place-expansion of scripts

Composition Language
none

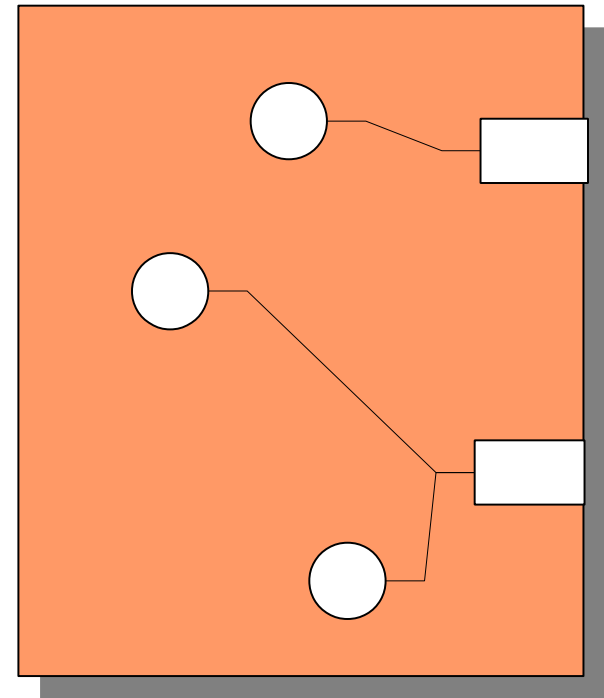
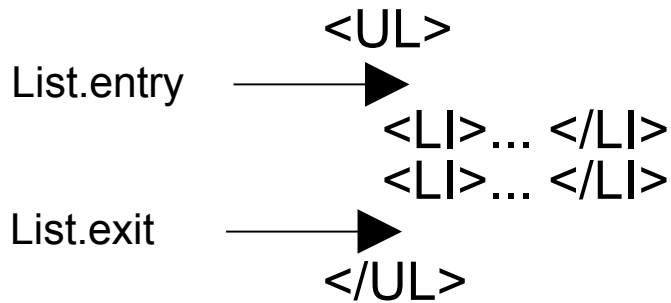
The Component Model of Invasive XML Composition

- ▶ The component is a fragment component (template)
 - A subword of the language, with *holes*
- ▶ Slots are variation points of a component
 - Parameters
 - Positions, which are subject to change
- ▶ Hooks are extension points
- ▶ Example:
 - A generic XML tree
 - A XML list with extension points



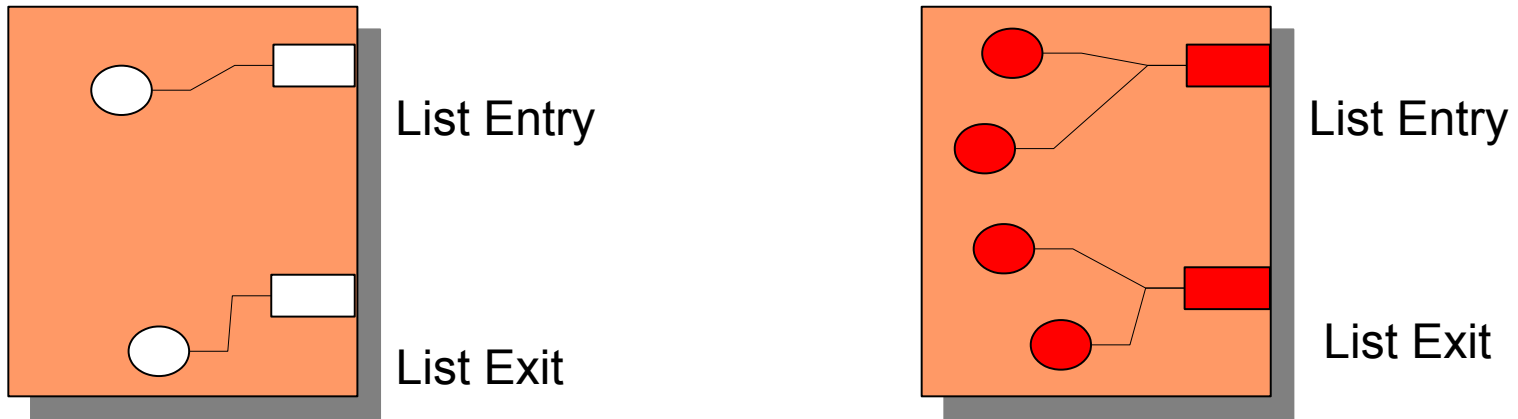
Extension of XML Fragment Components Should can be Typed

- ▶ What can be placed into an XML list entry/exit?



Slot and hook types are given by an XSchema, a metamodel of the XML document

Typed Hook Expansion for XML Components



```
<UL>  
  <LI>... </LI>  
  <LI>... </LI>  
</UL>
```

```
<UL>  
  <LI>... </LI>  
  <LI>... </LI>  
  <LI>... </LI>  
  <LI>... </LI>  
</UL>
```

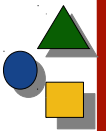
```
XMLcomponent.findHook(„ListEntry“).extend(„<LI>... </LI>“);  
XMLcomponent.findHook(„ListExit“).extend(„<LI>... </LI>“);
```



Insight: Web Systems Need Typed Template Processing

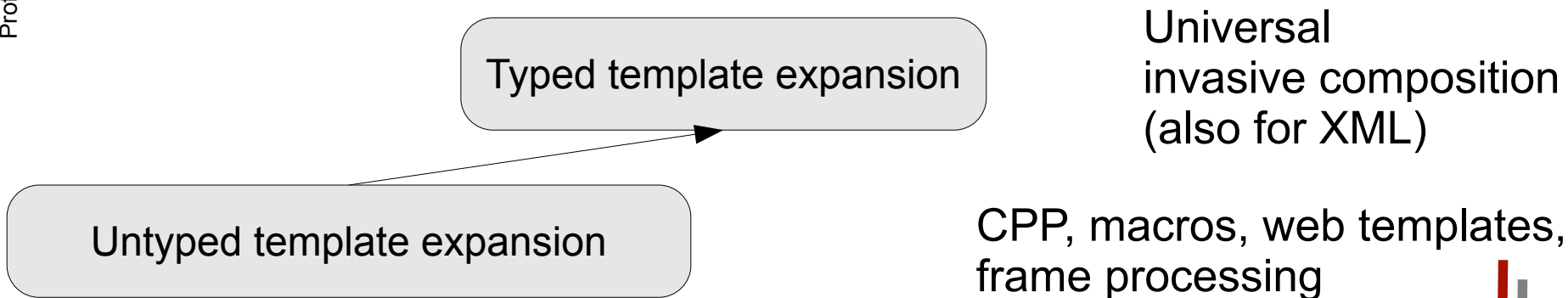
Problem: Web programming is based on *untyped template expansion (frame processing)*

It should be based on typed template expansion (invasive composition)

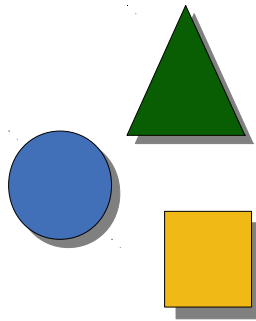


The Hierarchy of Staged Architectures

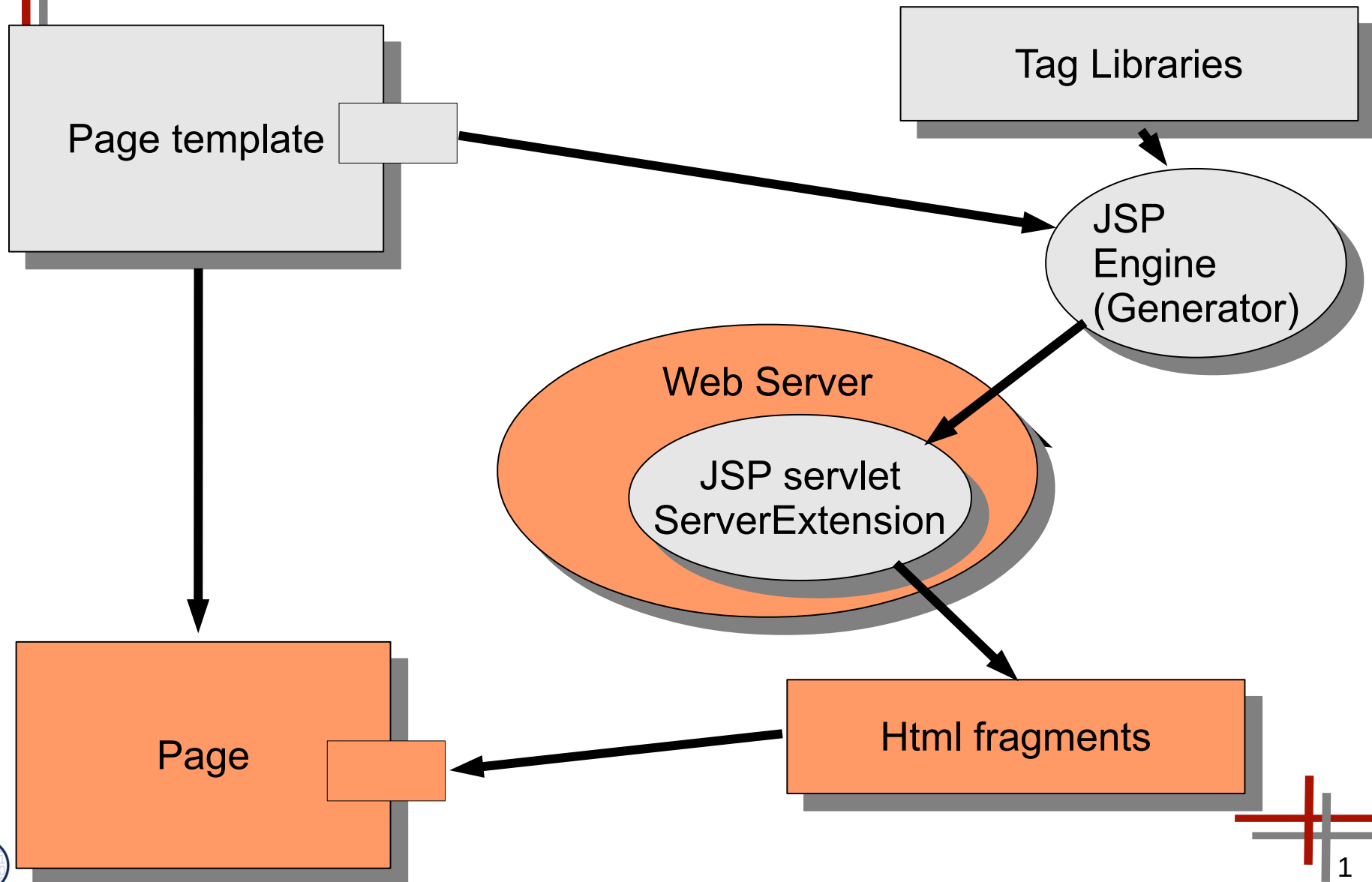
Prof. U. Aßmann, CBSE



Problem 2: Staging



The JSP Mechanism





Spagetti Code from JSP Tutorial - Belongs to Different Execution Stages

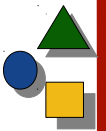
```
<html>
<%@page language="java" imports="java.util.*" %>
<h1> Welcome! </h1>
<jsp:useBean id="clock" class="jspCalendar" />
<p> Today is
<%=clock.getYear() %>-<%=clock.dayOfTheMonth() %>
</p>
<p>
<% if (Calender.getInstance().get(Calendar.AM_PM) == Clalender.AM) %>
    Good Morning!
<% }else { %>
    Good afternoon...
<% } %>
</p>
<html>
```



A Web Scripting Language with 5 Stages

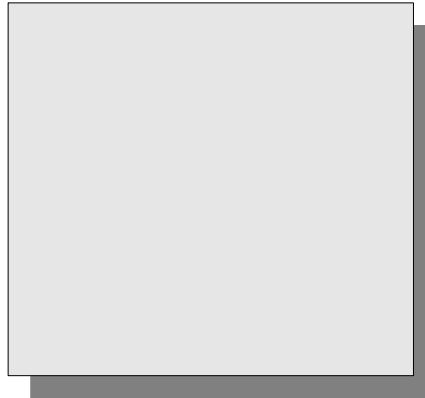
```
<xfa1:profession>
  <xfa2:ref pop-up>
    <sql>select arbitrary lastName from bakers</sql> baker
  <xfa2:ref pop-up>
</xfa1:profession>
<xfa:function hello>
  <body>
    <h1>This is My Personal Page with XFA</h1>
    <xfa:if Odd(environment^DATE)>
      <xfa:ref message>
    <xfa:else>
      Even day. No money for <xfa1:profession> :-(
    </xfa:if>
  </body>
</xfa:function>
<xfa:function message>
  Odd day today, dear student. You may visit the <xfa1:profession> shop.
</xfa:function>
```

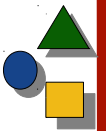
[until 2003: www.xml4all.com]



A Possible Solution: Staged Programming

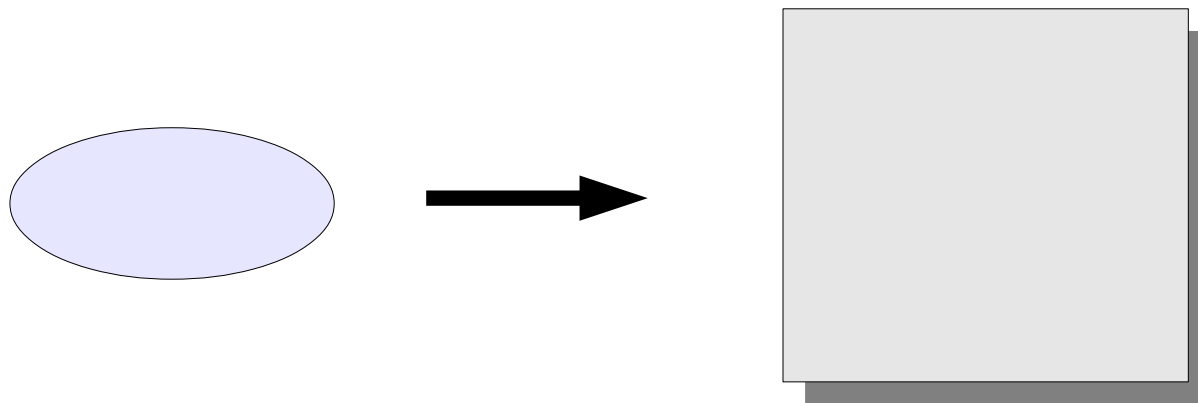
- ▶ In the Beginning, there was the Data

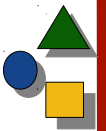




Then Came the Programs

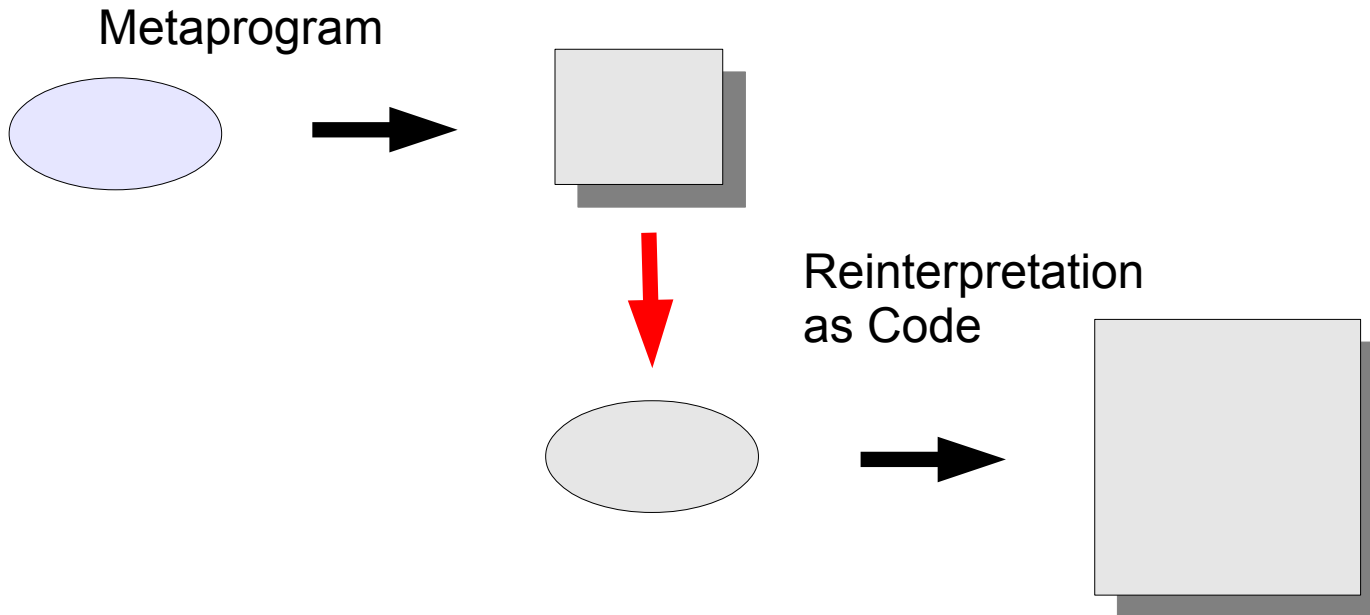
- ▶ Producing lots of data out of little code

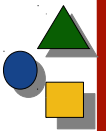




Then Came the Metaprograms

- ▶ Producing lots of programs from few metaprograms



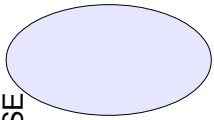


Then Came the Staged Metaprograms

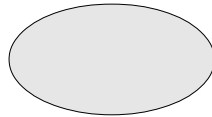
- ▶ Invented by Chiba, Sheard, Taha

Runtime 0

Metaprogram



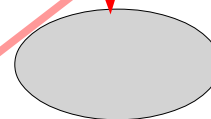
Reinterpretation
as Code



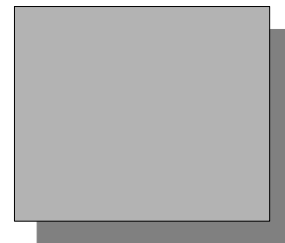
Runtime 1



Reinterpretation
as Code



Runtime 2





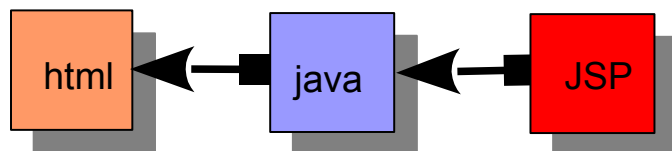
Staged Programming

- ▶ Staged programming (e.g., MetaML, MetaOCaML) has pioneered the mix of static metaprograms and programs
 - The metaprograms are expanded statically (stage 1) to produce the final program (stage 2)
 - Metaprograms are typed in the metamodel of the programs (type-safe expansion of metaprograms)

- ▶ Example [Taha]:

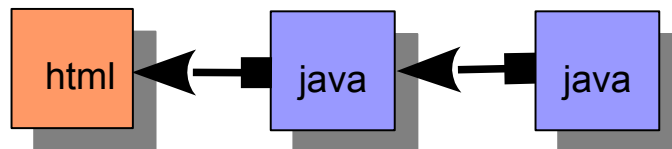
```
# let a = 1+2;;  
val a: int = 3  
# let a = .<1+2>.;;  
val a: int code = .<1+2>.  
# let b = .! a;;  
val b = 3
```

JSP Uses Staged Programming



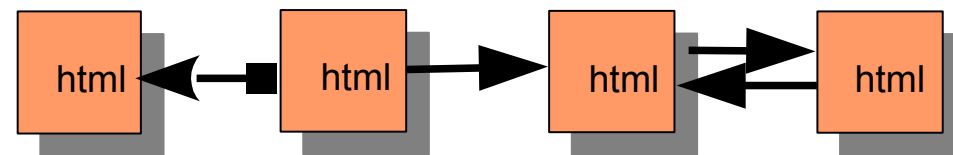
Stage 1

JSP expansion



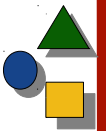
Stage 2

java expansion



Stage 3

interpretation of data

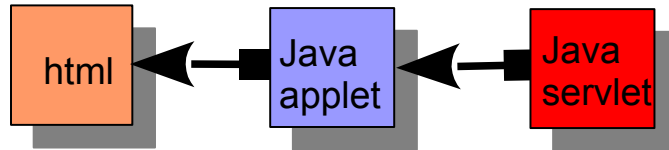


Spagetti Code Revisited

```
<html>
<%@page language="java" imports="java.util.*" %>
<h1> Welcome! </h1>
<jsp:useBean id="clock" class="jspCalendar" />
<p> Today is
<%=clock.getYear() %>-<%=clock.dayOfMonth() %>
</p>
<p>
<% if (Calender.getInstance().get(Calendar.AM_PM) == Clalender.AM) %>
    Good Morning!
<% }else { %>
    Good afternoon...
<% } %>
</p>
<html>
```

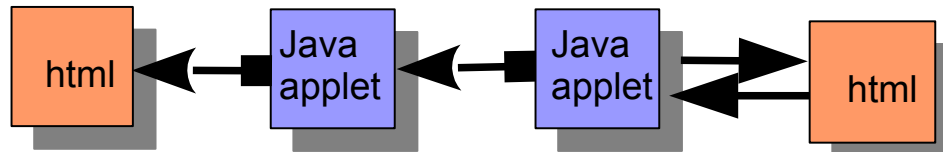
Servlet generator expands
blue lines to Java code

Example 2: Staged Servlet/Applet Processing



Stage 1

JSP expansion



Stage 2

java expansion

interpretation of data



Insight 2: Web Systems Need Staged Programming

Web programming is often based on
staged programming

- ▶ Because for dynamic web pages, code is generated
 - E.g., servlet or applet generation
- ▶ Because of the client-server stage separation
- ▶ Because legacy tools must be encapsulated into a stage (e.g., databases)

Staged programming should additionally be typed, otherwise
it is chaotic

N.B.: Configuration and Variant Selection works with Staged Programming

```
# fun f variant =  
  if variant = 1 then .<.fun q x = x*x.>.  
    else .<.fun q x = x/x.>.  
  
;;
```

```
# let variant = 1;;  
# fun g = (f variant) 2;;  
val g: int code = .<let q x =  
  x*x>.  
# let res = g 3;;  
val res = 9
```

Different behavior
of second stage

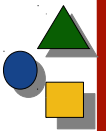
```
# let variant = 2;;  
# let g = (f variant) 2;;  
val g: int code = .<let q x =  
  x/x>.  
# let res = g 3;;  
val res = 1
```



Staging Is Used for Variant Management

On stage $n-1$, control-flow denotes variant selection for stage n

Platforms are often selected by evaluating control-flow in previous stages



Spagetti Code Revisited

```
#ifdef HTML
<html>
#else
<wap>
#endif
<%@page language="java" imports="java.util.*" %>
#ifdef HTML
<h1> Welcome! </h1>
#else
<bold>Welcome!</bold>
#endif
<jsp:useBean id="clock" class="jspCalendar" />
#ifdef HTML
<p>
#endif
.....
```

CPP stage selects
HTML or WAP

Evaluating the CPP script
chooses the platform

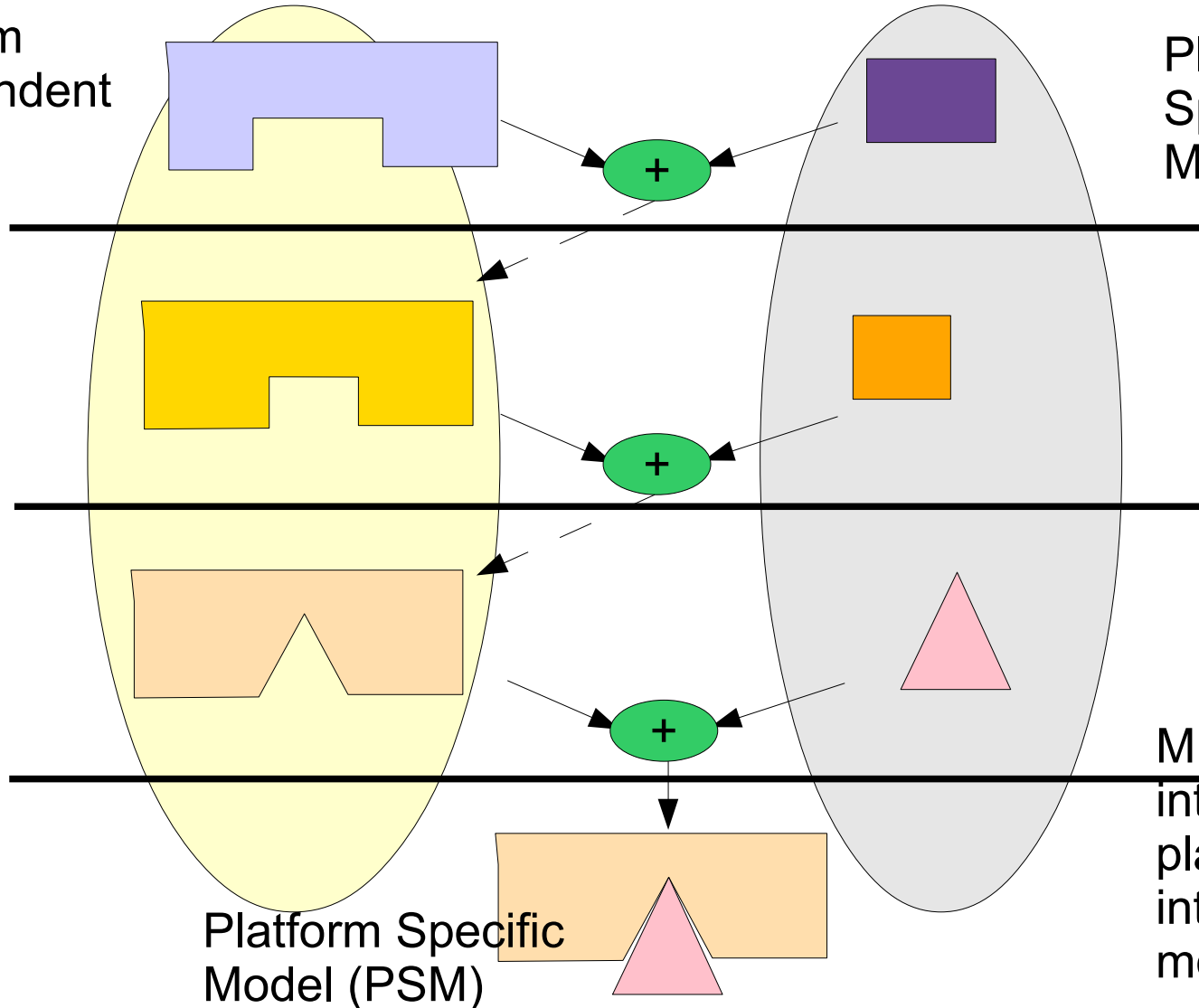
The C Preprocessor as Staged Programming System

- ▶ Insight: C with `#ifdef` language is a real staged programming system with CPP-C (State 0) and core-C (Stage 1)
 - That's why it's being used...
 - That's why it's so hard to deal with
- ▶ However, there is no component model, not even respect of the syntax of core-C
- ▶ The composition language of CPP-C is simple (macros, if-expressions, constant definitions)

A Staged Programming System: MDA

Platform Independent Models (PIM)

Platform Specific Models



Prof. U. Aßmann, CBSE

Platform Specific Model (PSM)

MDA weavers
integrate
platform variants
into staged
models

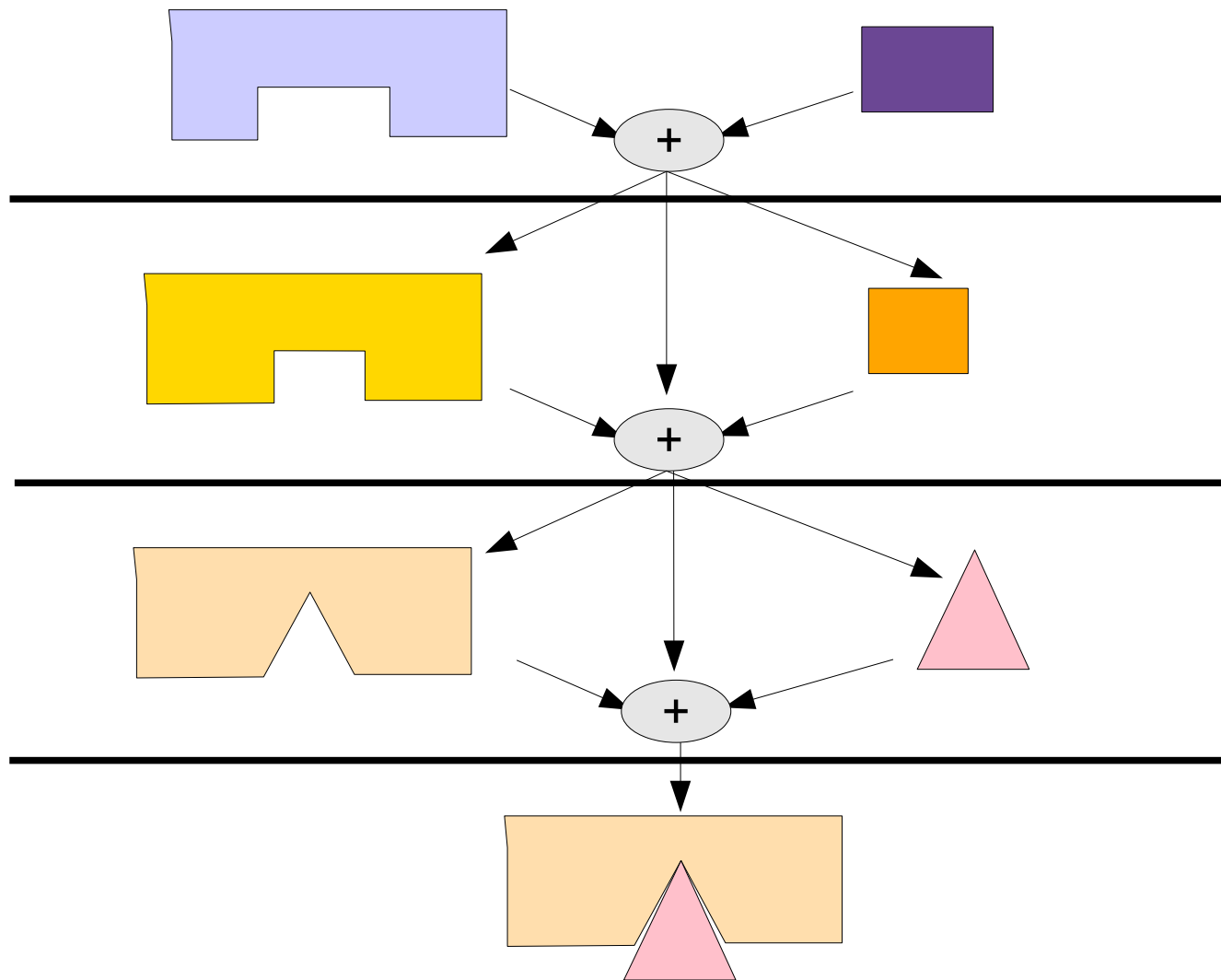


Staged Programming Architectures vs MDA

- ▶ MDA is a staged programming approach, but *not* a staged programming architecture, since no architecture, no component models are given
- ▶ ... but a staged programming technology for variant selection

... but we can build more powerful forms of MDA by taking in the ideas of staged programming and staged architectures

Staged Architectures Written as Layers





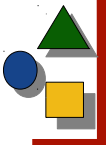
Advantages of Staged Programming

- ▶ Typed
 - Type-safe development, less error-prone
- ▶ Concise representation of system
 - Representation is expanded through every stage
- ▶ Easy to code variants
 - Control flow on a build stage does variant selection
- ▶ Problems:
 - Still, lots of spaghetti code.

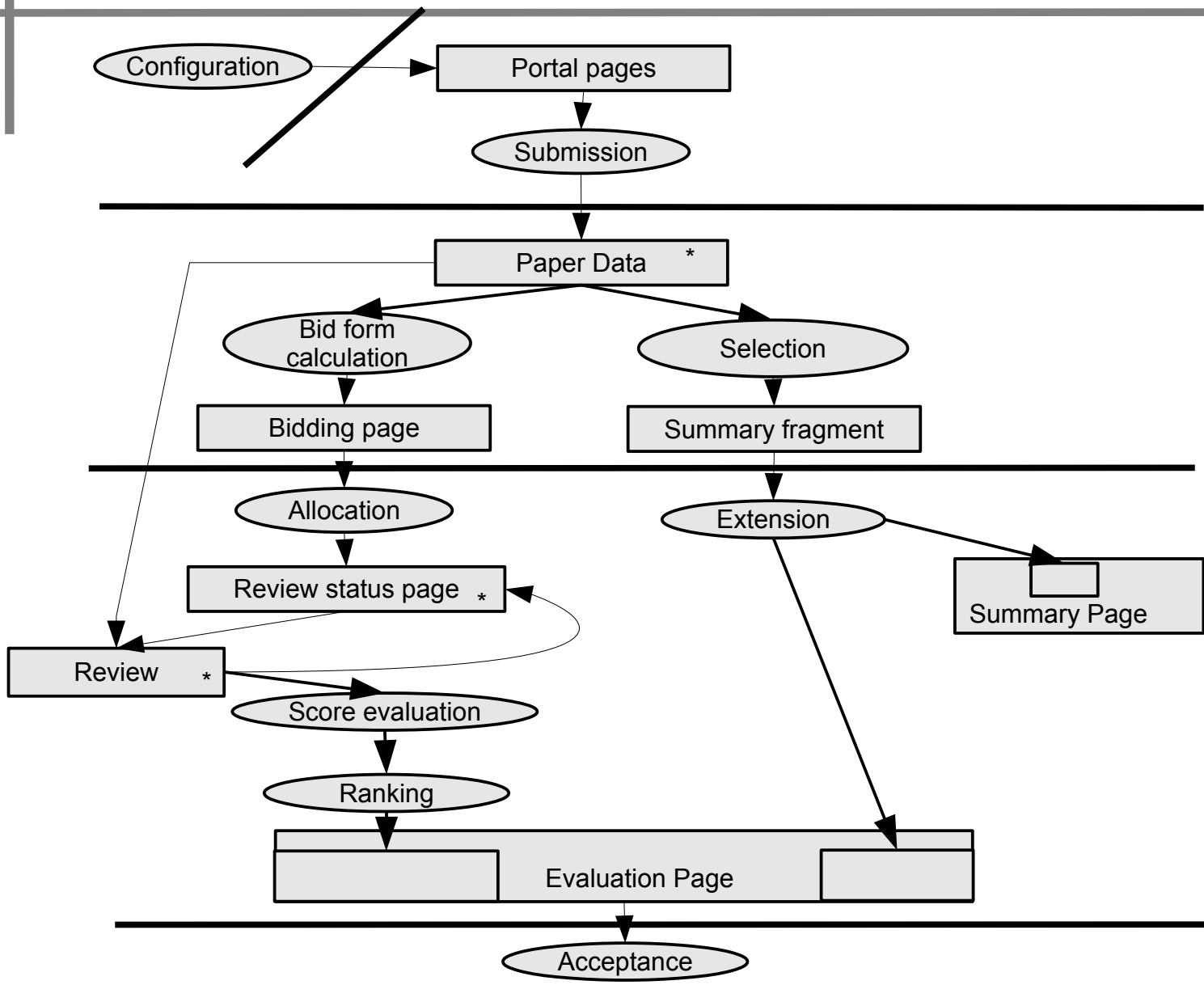


Example: The *START* Conference Management System

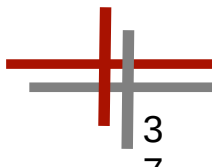
- ▶ START is a review management system
 - It has a 5-phase staged template expansion architecture
 - START servlets are composition scriptlets that compose (parameterize, extend) html-templates
- ▶ Using invasive composition, we developed a *staged typed template expansion* system
- ▶ It is no problem to generate servlets, too. Then we have real staged programming

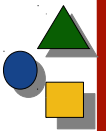


The Staged Template Expansion Architecture of START



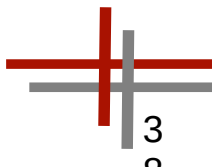
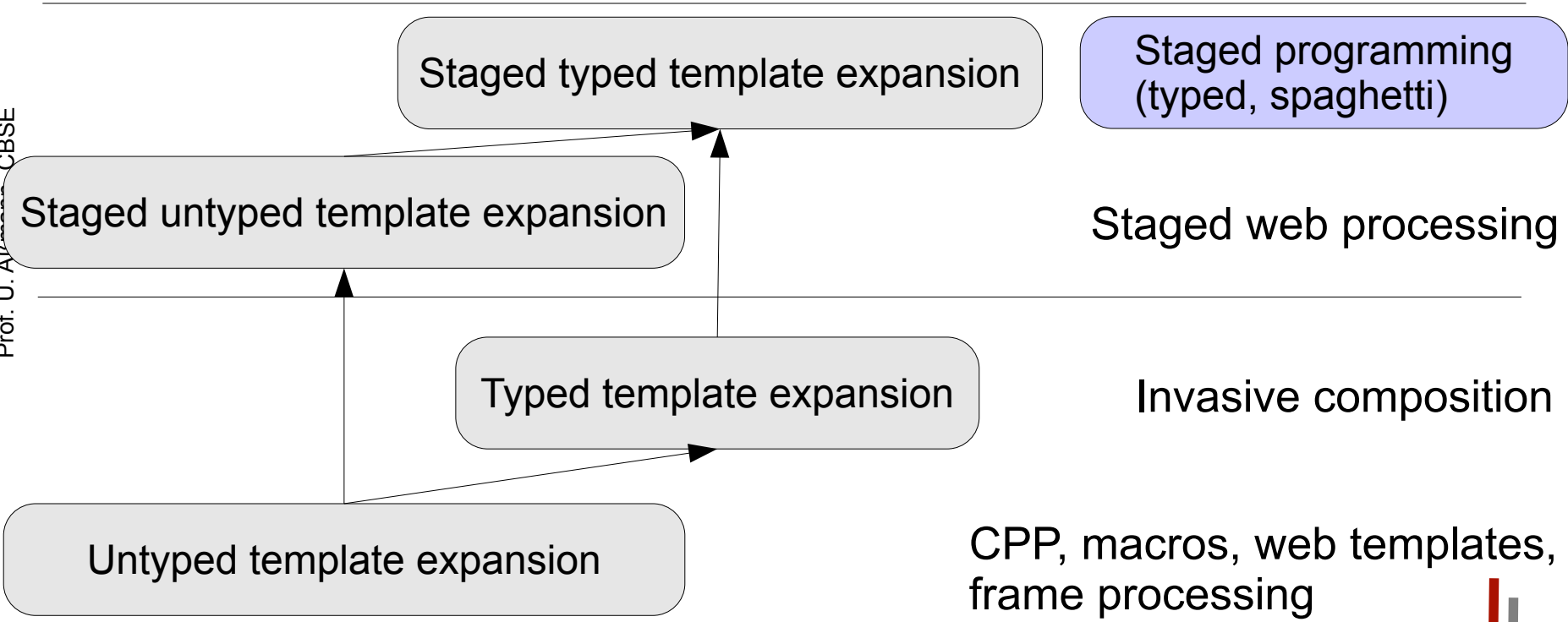
Prof. U. Aßmann, CBSE





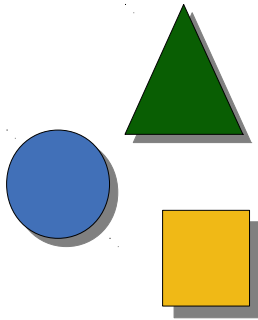
The Hierarchy of Staged Architectures

Prof. U. AP
CBSE



54.1.3 Problem 3: Spaghetti Code

and a possible remedy:
staged architectures





Architecture and Composition

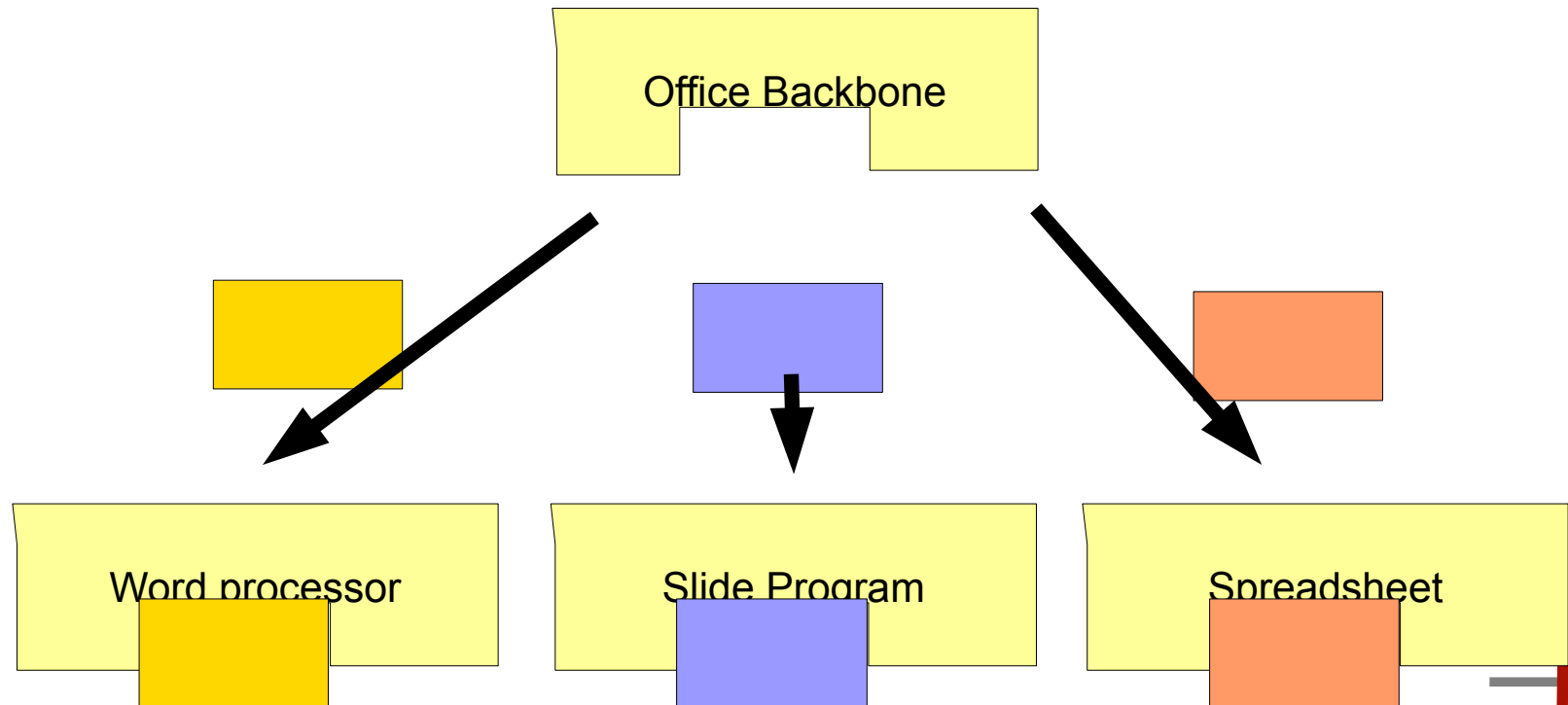
- ▶ Two of the central insights of the software engineering in the 1990s are:

Separate architecture from the components

Compose components by a *composition language*

Benefit of Architectures

- ▶ Comprehensibility
- ▶ Commonalities into the architectural level, variabilities into the application-specific components
- ▶ Does this also hold for web programming?



Less Spaghetti Code: A Fragment-Based Template and its Architecture

Component

```
<html>
    <hook id="imports">
<h1> Welcome! </h1>
    <hook id="use">
&ltp> Today is
    <hook id="year"/>
    -<hook id="day"/>
</p>
&ltp>
    <hook id="greeting"/>
</p>
</html>
```

Prof. U. Alsmann, CBSE

Composition Program (Architecture)

```
public class composeTemplate {
    String use = „jspCalendar“
    String imports=„java.util.*“;

    compose() {
        Template template = read();
        Bean clock = new jspCalendar();
        String year = clock.getYear();
        String day = clock.dayOfTheMonth();
        if (Calender.getInstance().get(Calendar.AM_PM) ==
            Calender.AM)
            greeting = “Good Morning!”;
        else
            greeting = “Good afternoon...”;
        this.merge(template);
    }
}
```

Separation of Components and Architecture Allows for Variants

```
public class composeTemplate {  
    String use =  
    String imports=  
    compose() {  
        String year =  
        String day =  
        greeting =  
    }  
}
```

Composition Program (Architecture)

Component 1

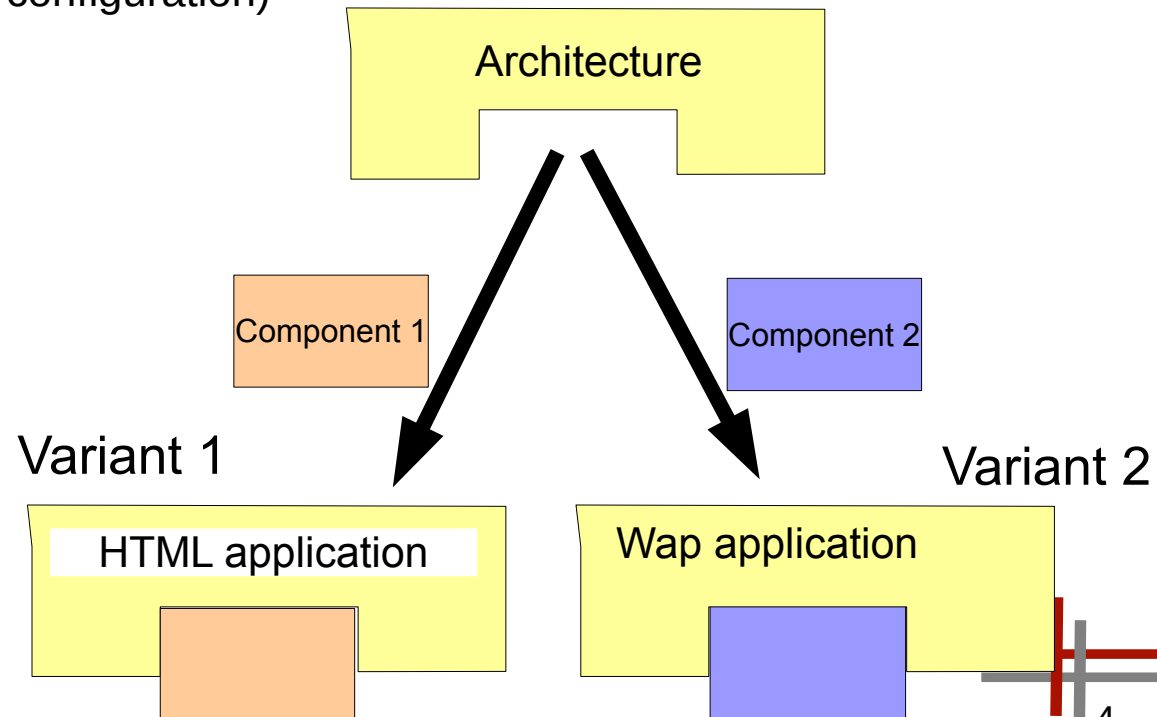
Component 2

```
<html>  
    <hook id="imports">  
<h1> Welcome! </h1>  
    <hook id="use">  
<p> Today is <hook id="year"/>  
    -<hook id="day"/>  
</p><p> <hook id="greeting"/>  
</p>  
</html>
```

```
<wap>  
    <hook id="imports">  
<bold> Welcome! </bold>  
    <hook id="use">  
<p> Today is <hook id="year"/>  
    -<hook id="day"/>  
</p><p> <hook id="greeting"/>  
</p>  
</wap>
```

Architecture and Variants in a Product Line

- ▶ Advantages for Separating Architecture From Application Components
 - Isolation of commonalities into frameworks
 - Comprehensibility
 - Programming-in-the-large is separated from programming-in-the-small, components can be abstracted away
 - Less spaghetti
 - Easy variability (variant configuration)



Variant Management by Control Flow in Architectural Composition Programs

```
public class composeTemplate {  
    if (HTML) then use component 1  
        else use component 2  
  
    String use =  
    String imports=  
    compose() {  
        String year =  
        String day =  
        greeting =  
    }  
}
```

Variant 1

Variant 2

SE

```
<html>  
    <hook id="imports">  
<h1> Welcome! </h1>  
    <hook id="use">  
<p> Today is <hook id="year"/>  
    -<hook id="day"/>  
</p><p> <hook id="greeting"/>  
</p>  
</html>
```

```
<wap>  
    <hook id="imports">  
<s1> Welcome! </h1>  
    <hook id="use">  
<p> Today is <hook id="year"/>  
    -<hook id="day"/>  
</p><p> <hook id="greeting"/>  
</p>  
</wap>
```



Definition: Staged Data-Flow Architectures

Staged data-flow architectures add an explicit architectural level to staged template processing

- ▶ Every stage is executed to produce **data** for the next stage (data-flow)
- ▶ Every stage is executed at a specific time
- ▶ On every stage, there is
 - an architecture,
 - a component model
 - a composition technique,
 - and a composition language
- ▶ Every composition language has its own interpreter
 - and is reduced (expanded) at different interpretation times



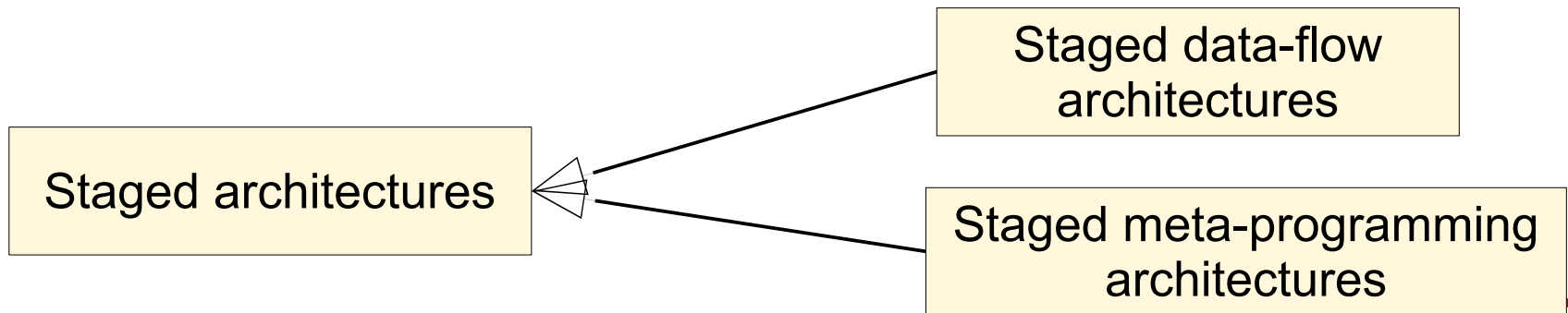
Web Programming needs Staged Data-Flow Architectures

- ▶ It would be nice to extend staged typed template expansion in web engineering to
- ▶ staged data-flow architectures.

Definition: Staged Architectures

Staged meta-programming architectures combine *staged programming with an explicit architectural level*

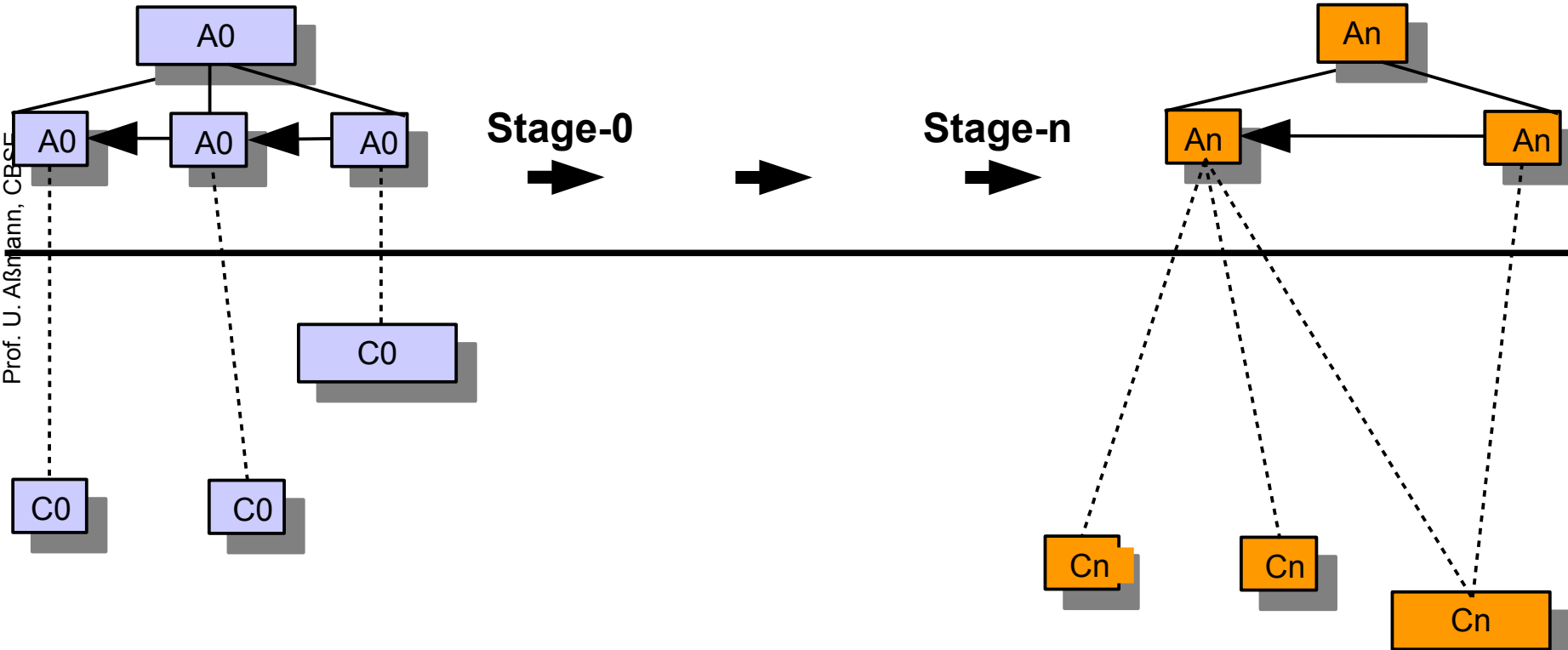
- ▶ Every stage is executed to produce **code** for the next stage
 - The final runtime code (architecture and components) is computed over several stages
 - The initial architecture is very small, the final architecture can be very large
 - Composition expressions, specifications, or programs may be hidden in components of a previous stage



Staged Metaprogramming Architectures Separate Large from Small

Stage-A0 architecture in composition language A0
Component language C0

Generated Stage-An architecture in composition language An
Component language Cn



Prof. U. Alsmann, CBSE

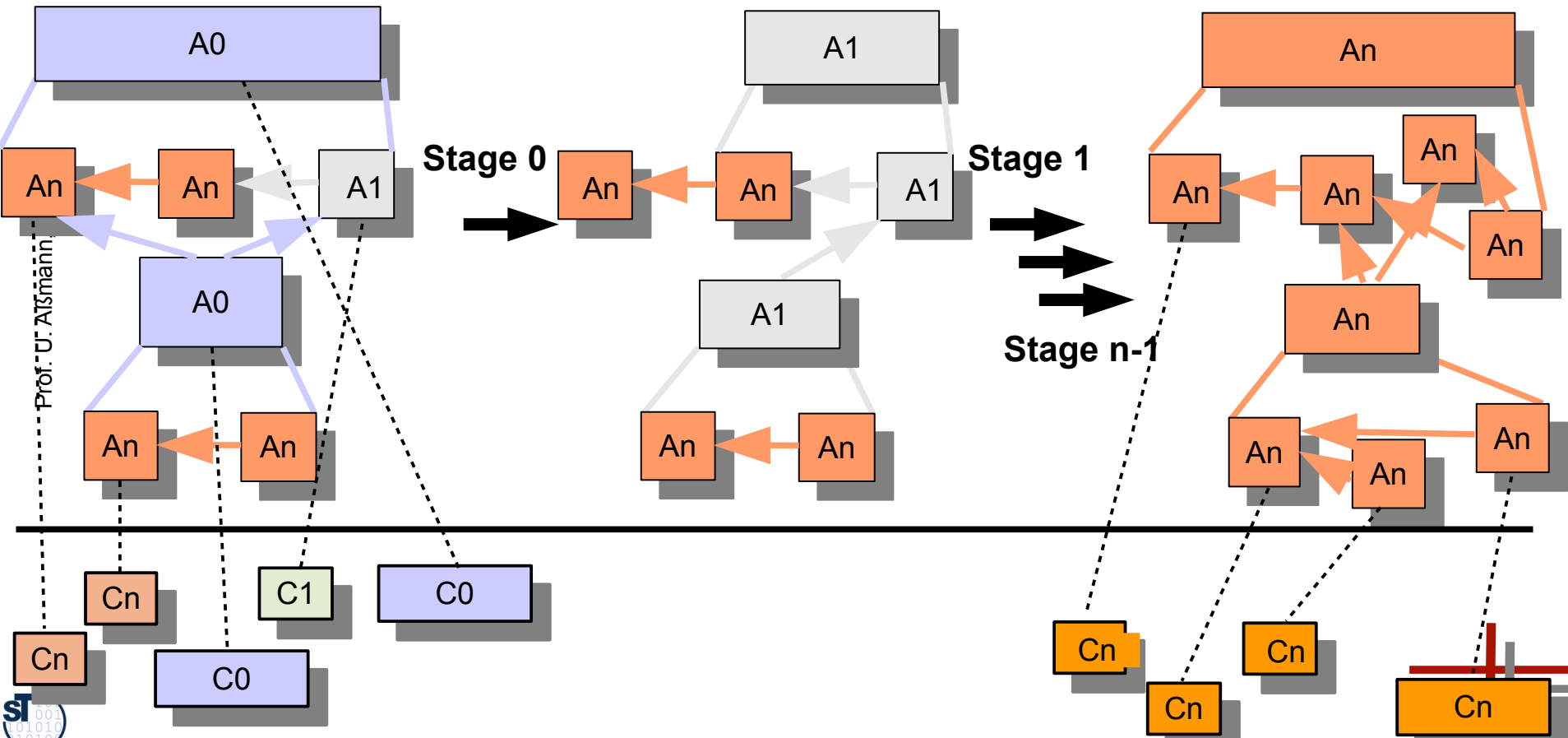


Staged Metaprogramming Architectures may have Different Component Models on Each Stage

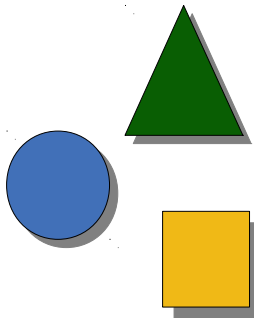
Stage-0 architecture in composition language A0
Component language C0

Stage 0 produces Stage-1 architecture in composition language A1
Component language C1

Stage n-1 produces Stage-n architecture in composition language An
Component language Cn



52.4 Staged Metaprogramming Architectures in Software Engineering



Build Management is Staged Composition

- ▶ Software build management is code composition in several stages
- ▶ Composition language: Make, ant, maven, etc.
 - Make is a composition tool with a lazy rule-based language
 - Expressions are applications of UNIX tools (compiler, linker, generator, preprocessor)
- ▶ Different component models on all stages

**Compiler
component model**

**Linker
component model**

**Runtime
component model**

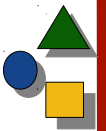


Modules

**Binary object
files**

**Runtime
components**





Invasive Software Composition

- ▶ Produces code from typed templates by parameterization and expansion

Stage-0
Composition level
language: Java

Stage-1
language: Java

**Fragment
component model**

**Runtime
component model
(objects)**



Stage-0

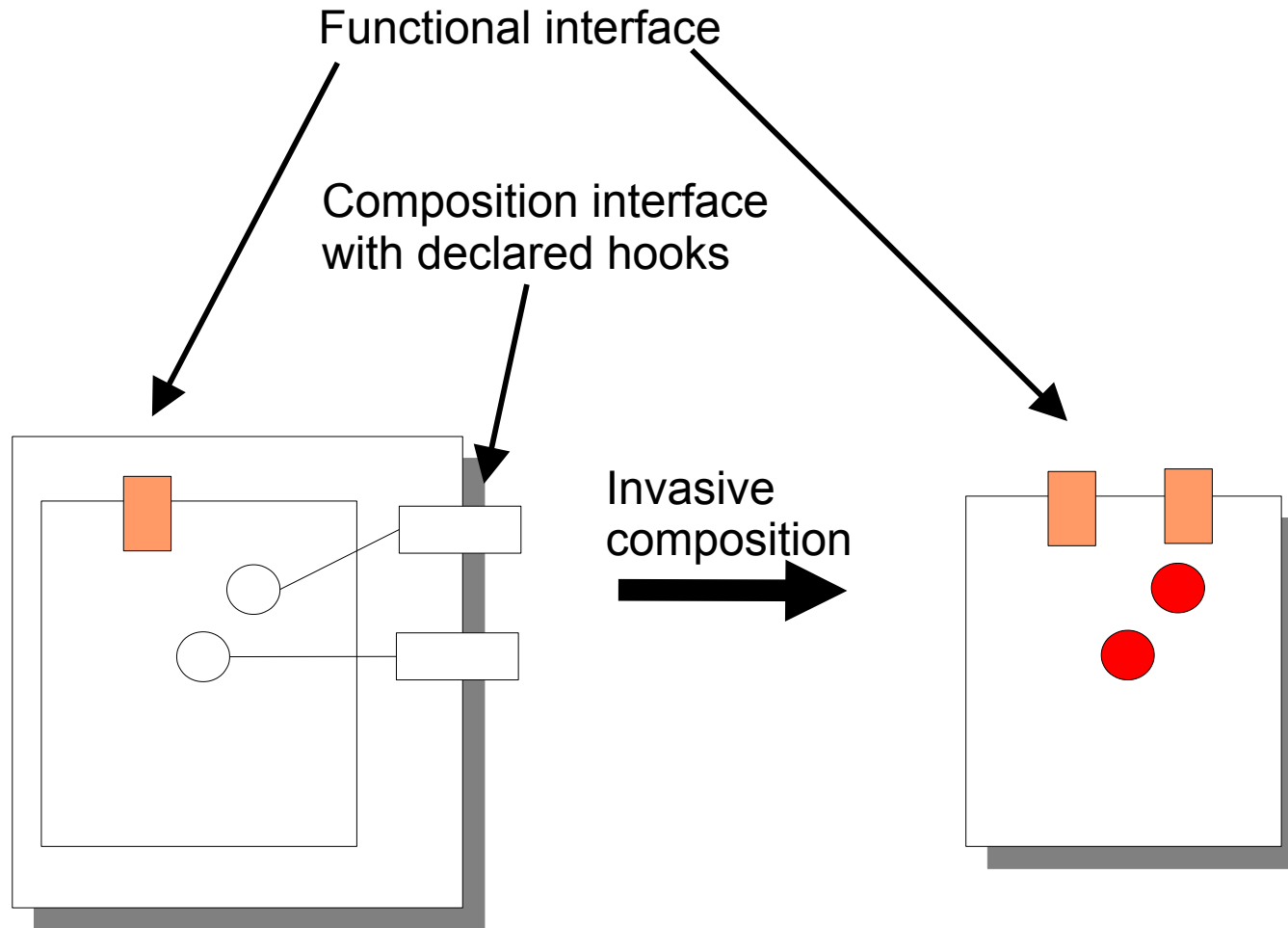


**Code Fragment
Components**

**Runtime
components**

Invasive Composition Produces Functional from Composition Interfaces

- ▶ Two different component models



Component Models on Different Levels in the Software Process

- ▶ Standard COTS models are just models for binary code

**Fragment
component model**

**Generic COTS
component model**

**Run time
component model**

**Code Fragment
Components**

**COTS
components**

**Run time
components**

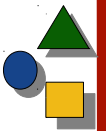




The Dresden Staged Architecture Development Process

- ▶ Fix the stages
 - Decide on a staged processing or programming architecture
- ▶ Fix the component models for every stage
 - Interface concepts, composition operations, composition language
 - Design a concrete component model with Reuseware toolkit
- ▶ Fix the architectures
 - Decide on a composition language on each level
- ▶ Fix the variant management
- ▶ Fix the components

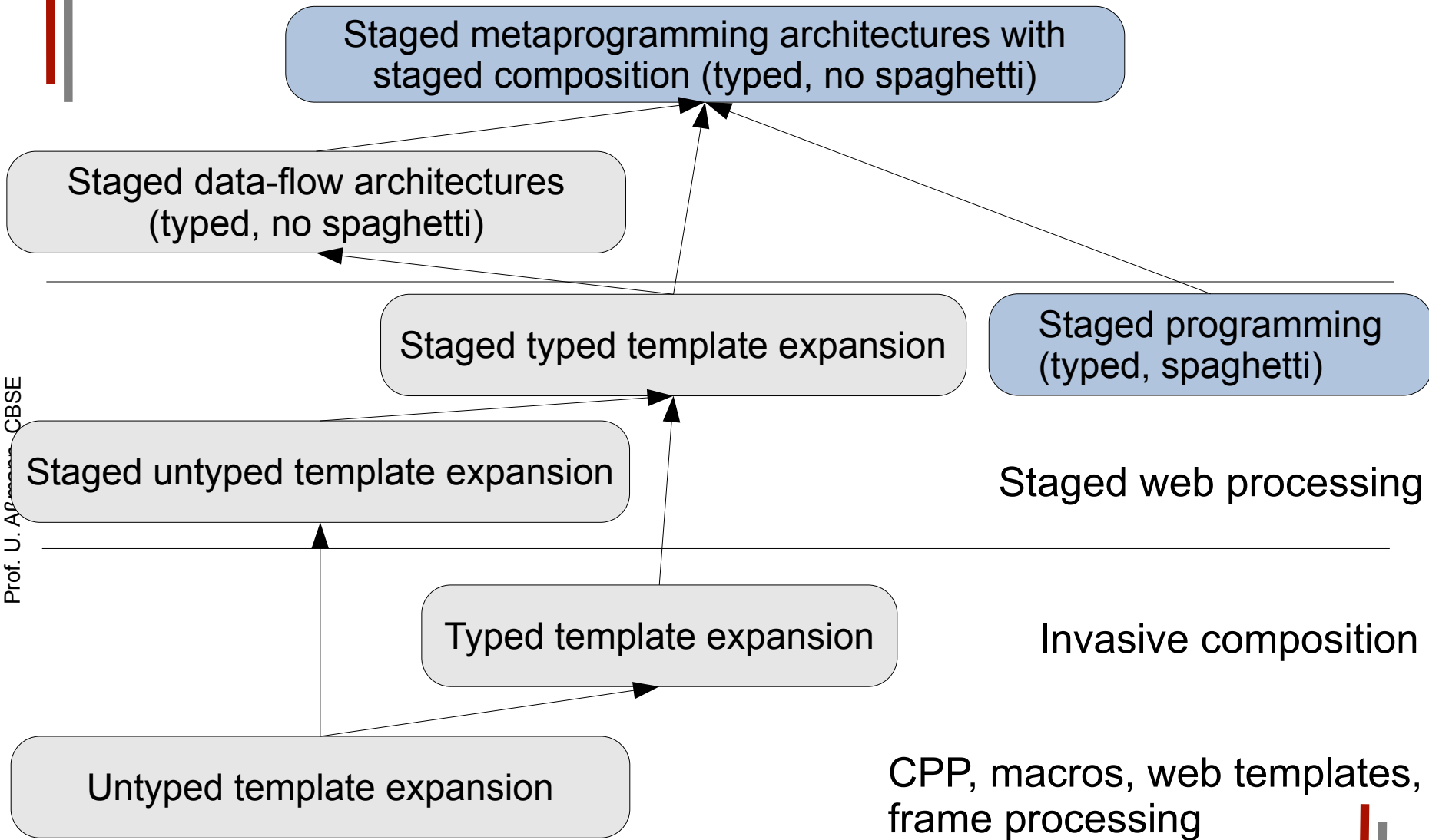
- ▶ And you'll have a pretty comprehensible product line!



The Vision of Staged Systems

- ▶ The staged programming principle is powerful, so future systems will employ it
- ▶ We need tools to support staged architectures
 - Visualize them
 - Debug them
 - Support the component models on all stages
 - that's a lot of work...

The Hierarchy of Staged Architectures





What Have We Learned?

- ▶ Large systems have *staged architectures* based on
 - *staged programming*,
 - *architectures*,
 - and *typed composition*
- ▶ On every stage, there is a component model and composition system
- ▶ All component models, composition systems and architectures have to work in synchronization
- ▶ Special cases:
 - The refinement-based software process (e.g., MDA)
 - Web systems, active documents
 - Invasive software composition
 - Standard build management

The Beauty of a Staged Programming Architecture





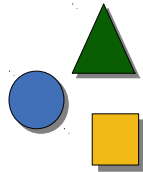
The End

- ▶ www.easycomp.org
- ▶ <http://www.the-compost-system.org>
- ▶ U. Aßmann. Invasive Software Composition, 2003, Springer.
- ▶ U. Aßmann. Architectural Styles for Active Documents. Special Issue “Software Composition” Science of Computer Programming, Elsevier, 2005.
- ▶ Walid Taha. A Gentle Introduction to Multi-Stage Programming. Domain-Specific Program Generation, 2003, LNCS, pp. 30-50
<http://www.springerlink.com/index/JEMT0D8VYN5JB2L8.pdf>
- ▶ Tim Sheard: Accomplishments and Research Challenges in Meta-programming. SAIG 2001: Proceedings of the Second International Workshop on Semantics, Applications, and Implementation of Program Generation, pp. 2-44, LNCS 2196, Springer-Verlag, 2001.

52. Staged Software Architectures with Staged Composition

Prof. Dr. Uwe Aßmann
Technische Universität
Dresden
Institut für Software- und
Multimediatechnologie
Version 13-1.0, 13.07.13

- 1) Web programming considered harmful
 - 1) Problem 1: Untyped template expansion
 - 2) Problem 2: Staging
 - 3) Problem 3: Spaghetti Code
- 2) Staged Architectures



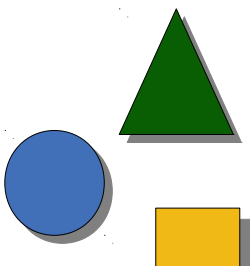
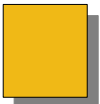


A Staged Architecture from Nature

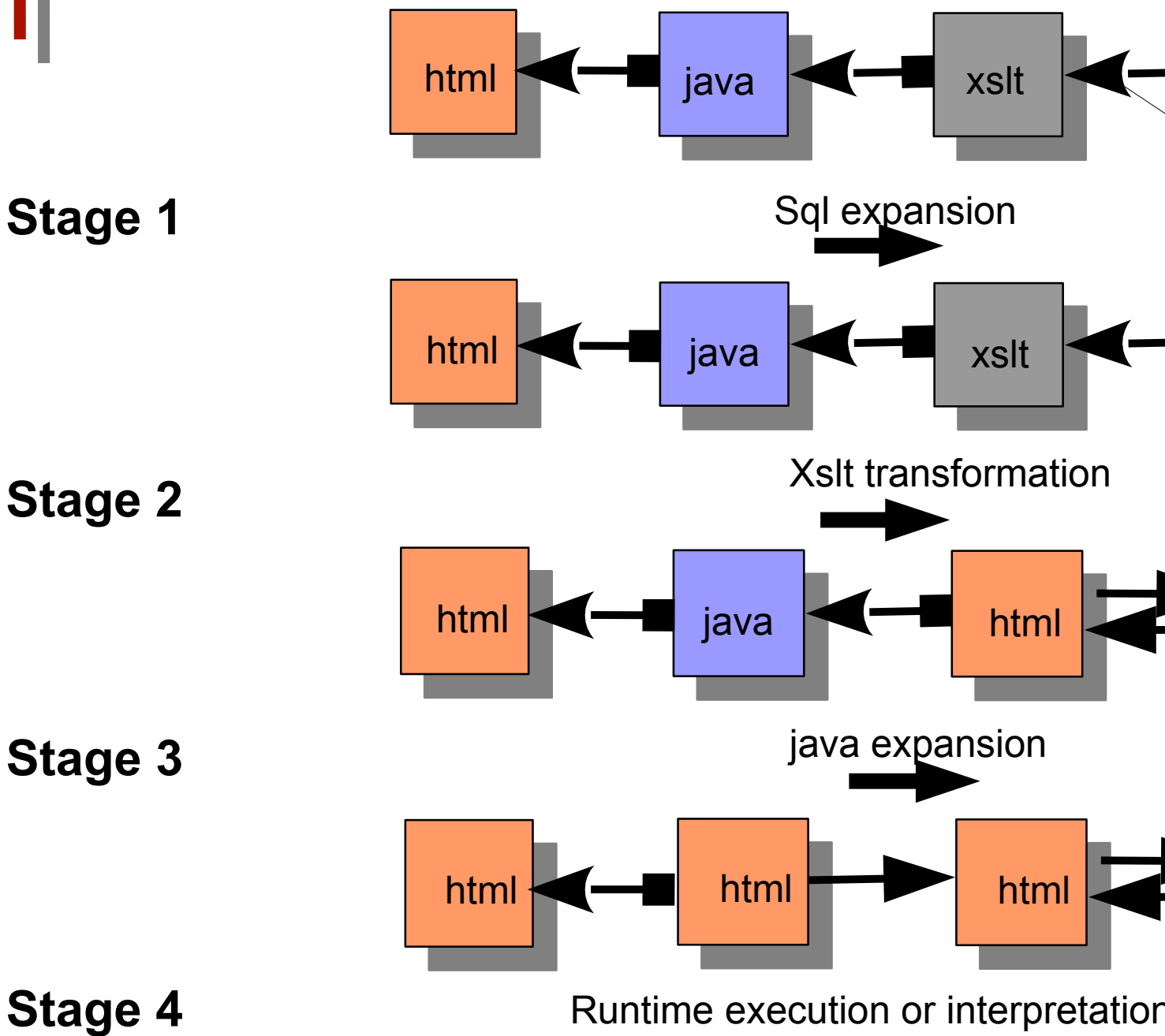
Prof. U. Alsmann, CESE



52.1 Web Programming Considerations Harmful



Web Programming: Staged, Untyped Expansion





Problems of Web Programming

- ▶ Untyped extensions of templates
 - Error-prone
- ▶ Comprehension very difficult, due to the different stages
- ▶ Spaghetti-code-like programs
 - Scripts mixed with templates
 - Only valuable for programming-in-the-small

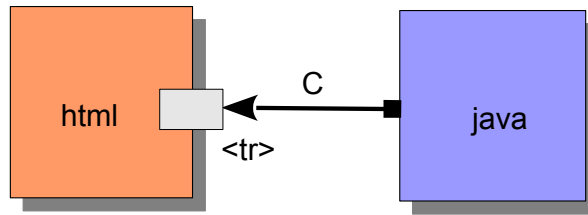
52.1.1 Problem 1: Untyped Template Expansion





Type-Safe Template Expansion

- ▶ How can you be sure that table rows are filled in?



- ▶ Answer: in an invasive document composition system, the type checker of the invasive composition program will tell you, when checking the composition step C



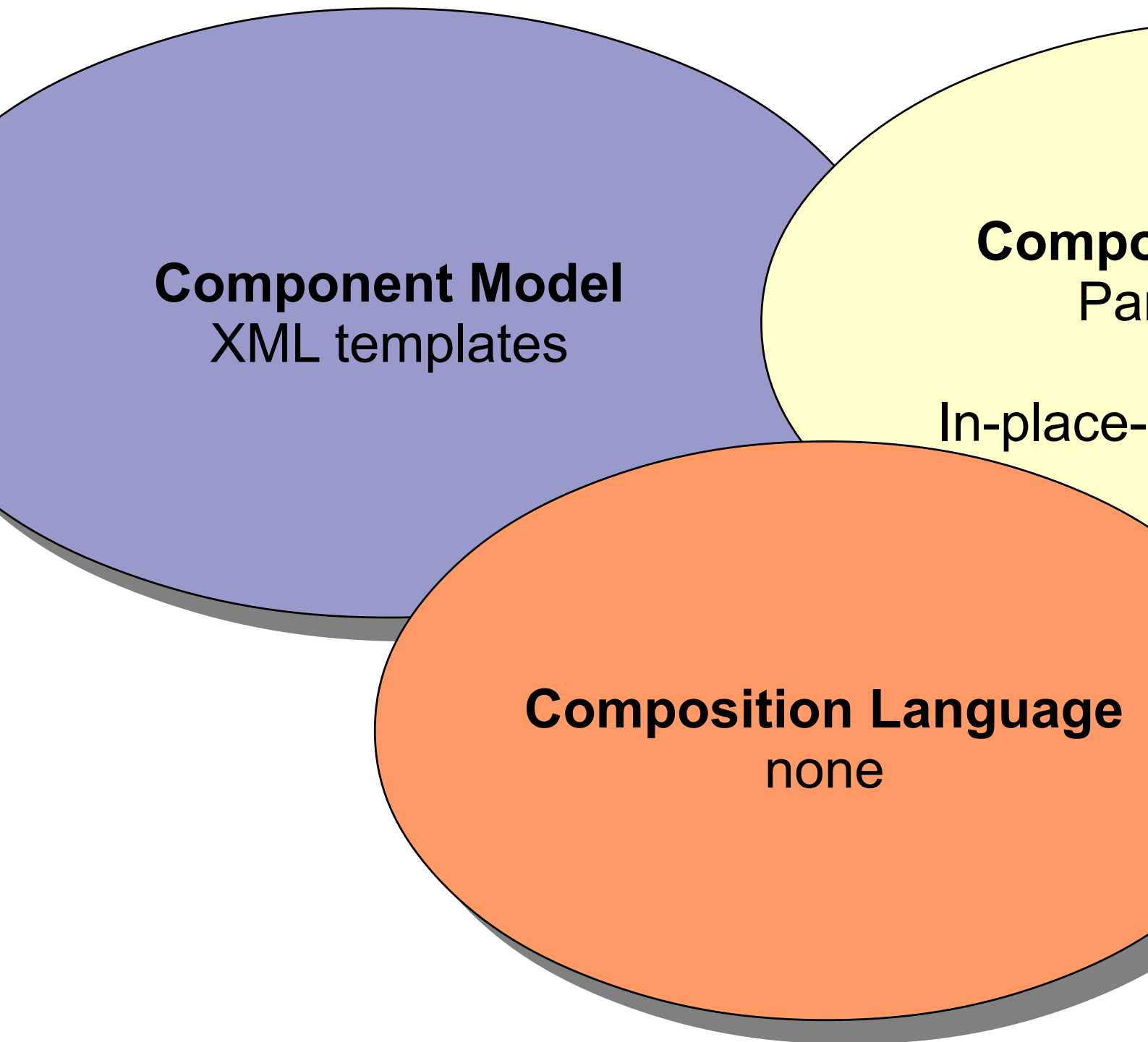
Universality of Invasive Composition

- ▶ Invasive composition only depends on a metamodel of the language
 - New hook and slot models can be derived from any language
 - Typing controls the composition of artifacts
- ▶ Hence, the method is *universal*
- ▶ and can be applied for typed document composition
- ▶ See www.reuseware.org, the universal invasive composition environment,
 - Can be tailored for text-based and diagrammatic languages
 - OpenOffice
 - XML dialects
 - EMF-based

Prof. U. Altmann, CESE



Elements of Web Composition Sys

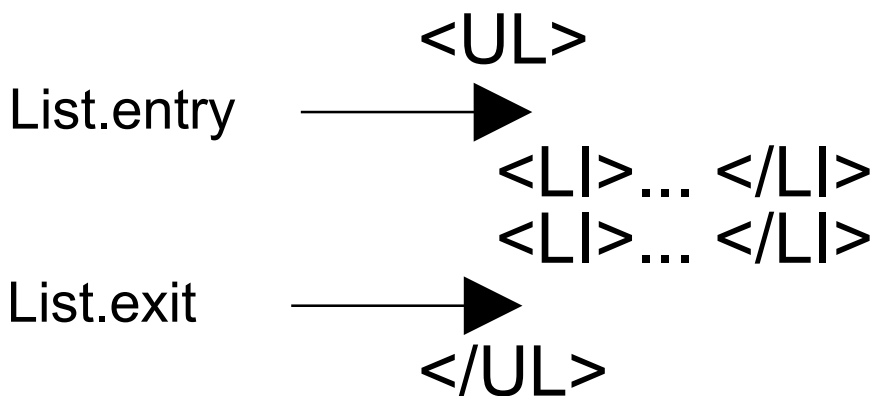


The Component Model of Invasive Composition

- ▶ The component is a fragment component (template)
 - A subword of the language, with *holes*
- ▶ Slots are variation points of a component
 - Parameters
 - Positions, which are subject to change
- ▶ Hooks are extension points
- ▶ Example:
 - A generic XML tree
 - A XML list with extension points

Extension of XML Fragment Com can be Typed

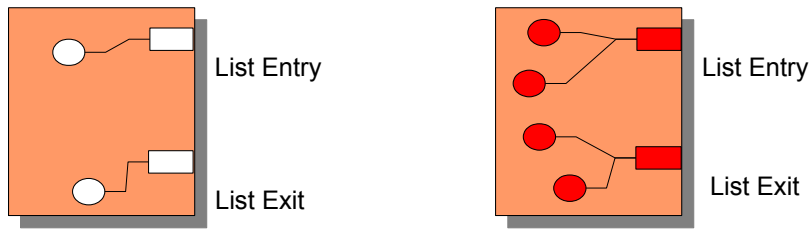
- ▶ What can be placed into an XML list entry/exit



Slot and hook types are given by an XSchema, a document



Typed Hook Expansion for XML Components



...
...

...
...
...
...

Prof. U. Alsmann, CBSE

```
XMLcomponent.findHook(„ListEntry“).extend(„<LI>... </LI>“);  
XMLcomponent.findHook(„ListExit“).extend(„<LI>... </LI>“);
```





Insight: Web Systems Need Typed Template Processing

Problem: Web programming is based on *untyped template expansion (frame processing)*

Prof. U. Altmann, CESE

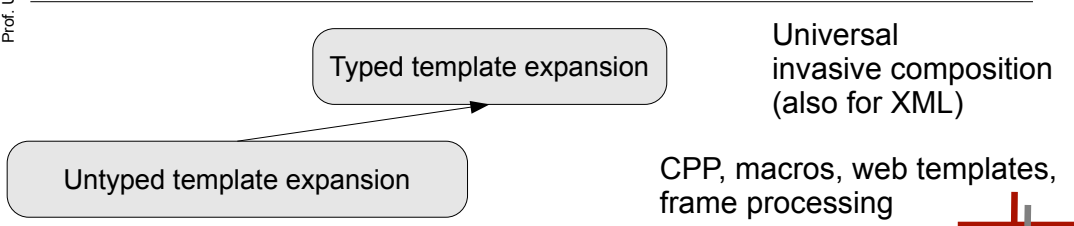
It should be based on typed template expansion (invasive composition)





The Hierarchy of Staged Architectures

Prof. U. Altmann, CESE

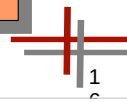
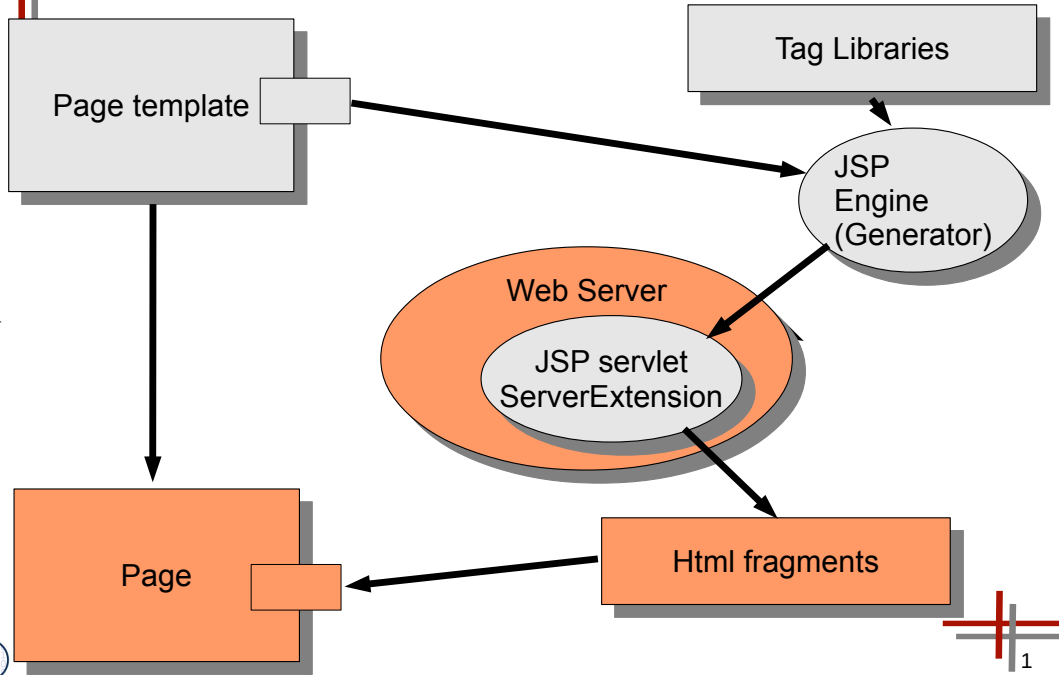


Problem 2: Staging



The JSP Mechanism

Prof. U. Aßmann, CESE

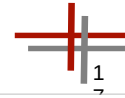




Spagetti Code from JSP Tutorial - Belongs to Different Execution Stages

Prof. U. Altmann, CESE

```
<html>
<%@page language="java" imports="java.util.*" %>
<h1> Welcome! </h1>
<jsp:useBean id="clock" class="jspCalendar" />
<p> Today is
<%=clock.getYear() %>-<%=clock.dayOfTheMonth() %>
</p>
<p>
<% if (Calender.getInstance().get(Calendar.AM_PM) == Clalender.AM) %>
    Good Morning!
<% }else { %>
    Good afternoon...
<% } %>
</p>
<html>
```





A Web Scripting Language with 5 Stages

```
<xfa1:profession>
  <xfa2:ref pop-up>
    <sql>select arbitrary lastName from bakers</sql> baker
  <xfa2:ref pop-up>
</xfa1:profession>
<xfa:function hello>
  <body>
    <h1>This is My Personal Page with XFA</h1>
    <xfa:if Odd(environment^DATE)>
      <xfa:ref message>
    <xfa:else>
      Even day. No money for <xfa1:profession> :(
    </xfa:if>
  </body>
</xfa:function>
<xfa:function message>
  Odd day today, dear student. You may visit the <xfa1:profession> shop.
</xfa:function>
```

[until 2003: www.xml4all.com]

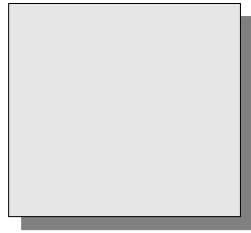
Prof. U. Altmann, CESE



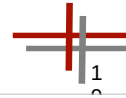


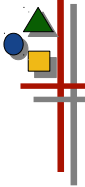
A Possible Solution: Staged Programming

- ▶ In the Beginning, there was the Data



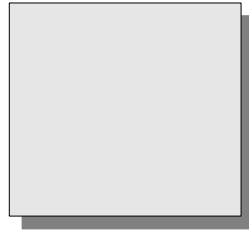
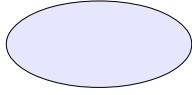
Prof. U. Altmann, CESE



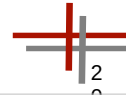


Then Came the Programs

- ▶ Producing lots of data out of little code



Prof. U. Altmann, CESE

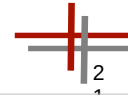
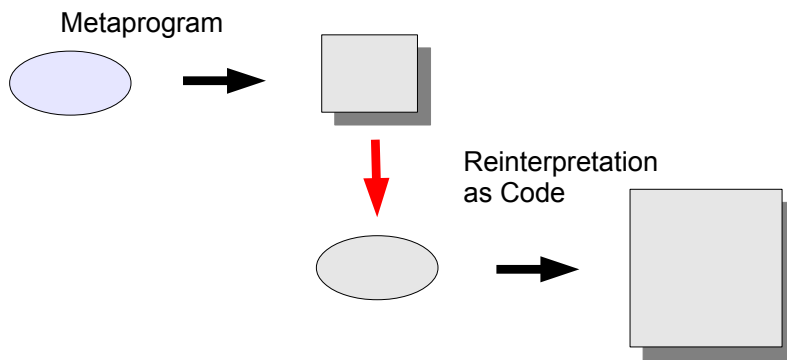




Then Came the Metaprograms

- ▶ Producing lots of programs from few metaprograms

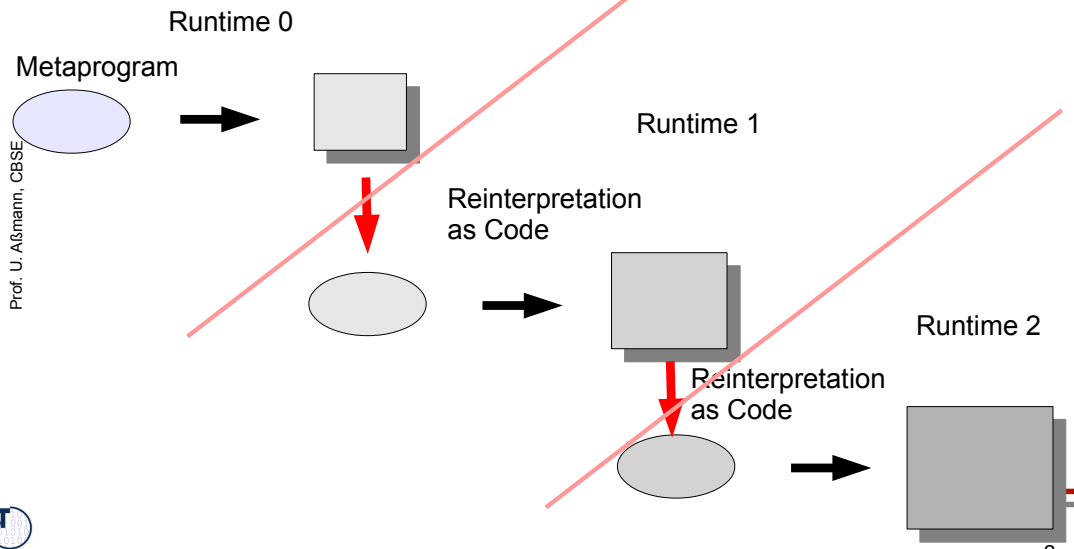
Prof. U. Alsmann, CESE





Then Came the Staged Metaprograms

- ▶ Invented by Chiba, Sheard, Taha



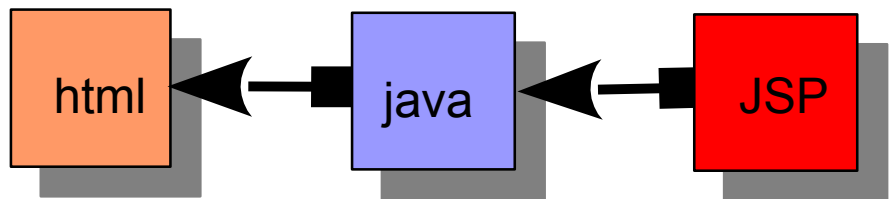


Staged Programming

- ▶ Staged programming (e.g., MetaML, MetaOCaml) separates static metaprograms and programs
 - The metaprograms are expanded statically (stage 1) and the program (stage 2)
 - Metaprograms are typed in the metamodel of the target language (of metaprograms)
- ▶ Example [Taha]:

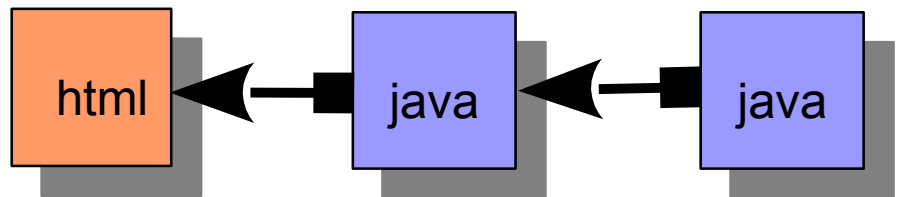
```
# let a = 1+2;;  
val a: int = 3  
# let a = .<1+2>.;;  
val a: int code = .<1+2>.  
# let b = .! a;;  
val b = 3
```

JSP Uses Staged Programming



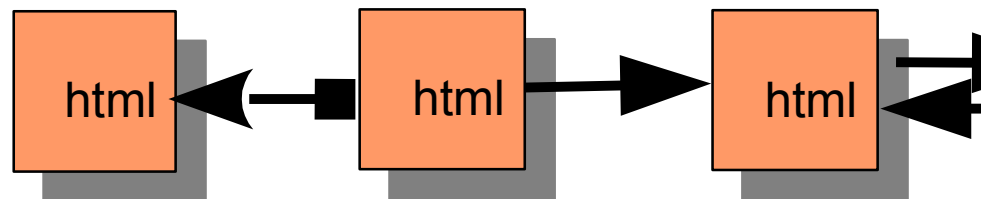
Stage 1

JSP expansion



Stage 2

java expansion



Stage 3

interpretation of data



Spagetti Code Revisited

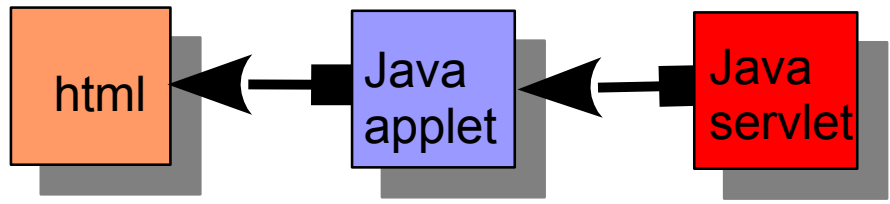
```
<html>
<%@page language="java" imports="java.util.*" %>
<h1> Welcome! </h1>
<jsp:useBean id="clock" class="jspCalendar" />
<p> Today is
<%=clock.getYear() %>-<%=clock.dayOfTheMonth() %>
</p>
<p>
<% if (Calender.getInstance().get(Calendar.AM_PM) == Clalender.AM) %>
    Good Morning!
<% }else { %>
    Good afternoon...
<% } %>
</p>
</html>
```

Servlet generator expands
blue lines to Java code

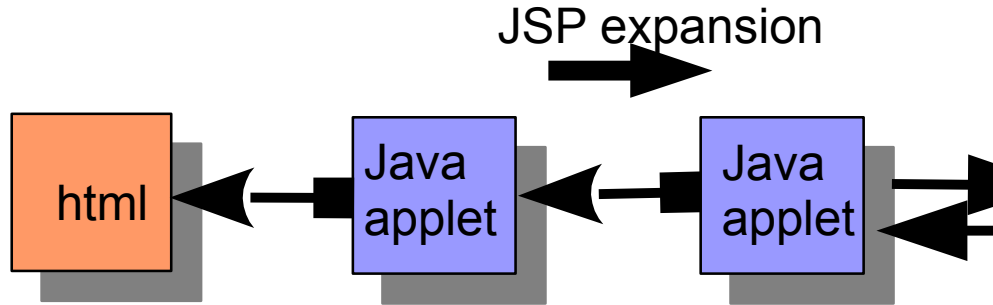
Prof. U. Altmann, CESE



Example 2: Staged Servlet/Applet Processing

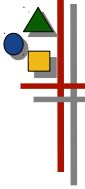


Stage 1



Stage 2





Insight 2: Web Systems Need Staged Programming

Web programming is often based on
staged programming

- ▶ Because for dynamic web pages, code is generated
 - E.g., servlet or applet generation
- ▶ Because of the client-server stage separation
- ▶ Because legacy tools must be encapsulated into a stage (e.g., databases)

Prof. U. Altmann, CESE

Staged programming should additionally be typed, otherwise
it is chaotic





N.B.: Configuration and Variant Selection works with Staged Programming

```
# fun f variant =  
  if variant = 1 then .<.fun q x = x*x.>.  
                      else .<.fun q x = x/x.>.  
  
;;
```

```
# let variant = 1;;  
# fun g = (f variant) 2;;  
val g: int code = .<let q x =  
  x*x>.  
# let res = g 3;;  
val res = 9
```

Different behavior
of second stage

```
# let variant = 2;;  
# let g = (f variant) 2;;  
val g: int code = .<let q x =  
  x/x>.  
# let res = g 3;;  
val res = 1
```





Staging Is Used for Variant Management

On stage $n-1$, control-flow denotes variant selection for stage n

Prof. U. Altmann, CESE

Platforms are often selected by evaluating control-flow in previous stages





Spagetti Code Revisited

```
#ifdef HTML
<html>
#else
<wap>
#endif
<%@page language="java" imports="java.util.*" %>
#ifdef HTML
<h1> Welcome! </h1>
#else
<bold>Welcome!</bold>
#endif
<jsp:useBean id="clock" class="jspCalendar" />
#ifdef HTML
<p>
#endif
.....
```

CPP stage selects
HTML or WAP

Evaluating the CPP script
chooses the platform

Prof. U. Altmann, CESE





The C Preprocessor as Staged Programming System

- ▶ Insight: C with `#ifdef` language is a real staged programming system with CPP-C (State 0) and core-C (Stage 1)
 - That's why it's being used...
 - That's why it's so hard to deal with
- ▶ However, there is no component model, not even respect of the syntax of core-C
- ▶ The composition language of CPP-C is simple (macros, if-expressions, constant definitions)

Prof. U. Altmann, CESE



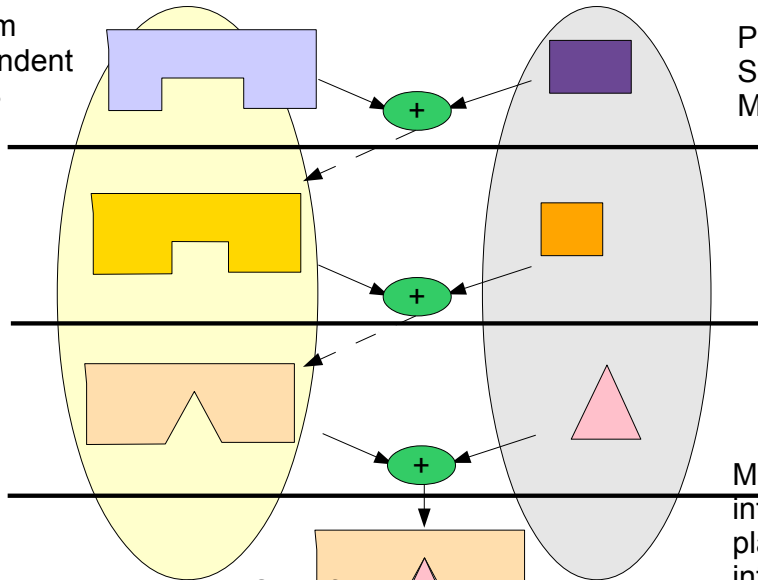


A Staged Programming System: MDA

Platform Independent Models (PIM)

Platform Specific Models

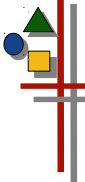
Prof. U. Altmann, CESE



Platform Specific Model (PSM)

MDA weavers integrate platform variants into staged models





Staged Programming Architectures vs MDA

- ▶ MDA is a staged programming approach, but *not* a staged programming architecture, since no architecture, no component models are given
- ▶ ... but a staged programming technology for variant selection

Prof. U. Alsmann, CESE

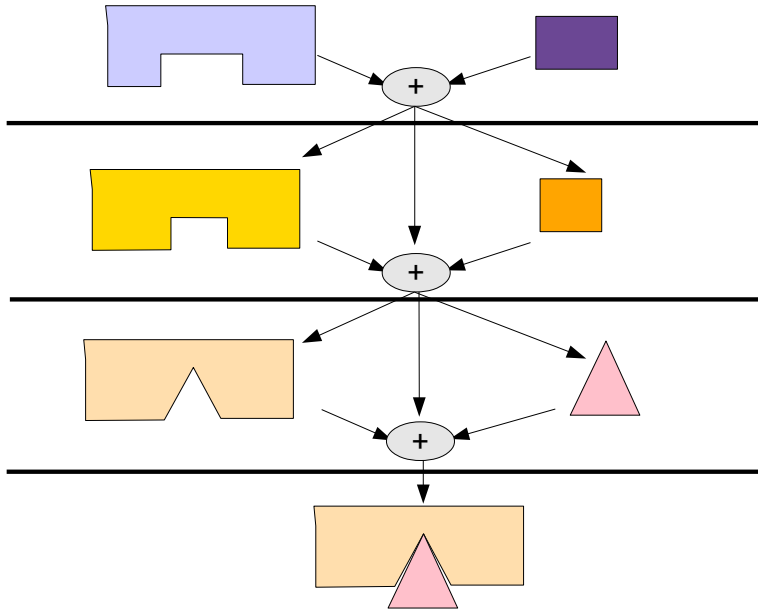
... but we can build more powerful forms of MDA by taking in the ideas of staged programming and staged architectures

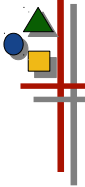




Staged Architectures Written as Layers

Prof. U. Altmann, CBSE



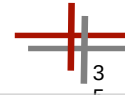


Advantages of Staged Programming

- ▶ Typed
 - Type-safe development, less error-prone
- ▶ Concise representation of system
 - Representation is expanded through every stage
- ▶ Easy to code variants
 - Control flow on a build stage does variant selection

- ▶ Problems:
 - Still, lots of spaghetti code.

Prof. U. Altmann, CESE





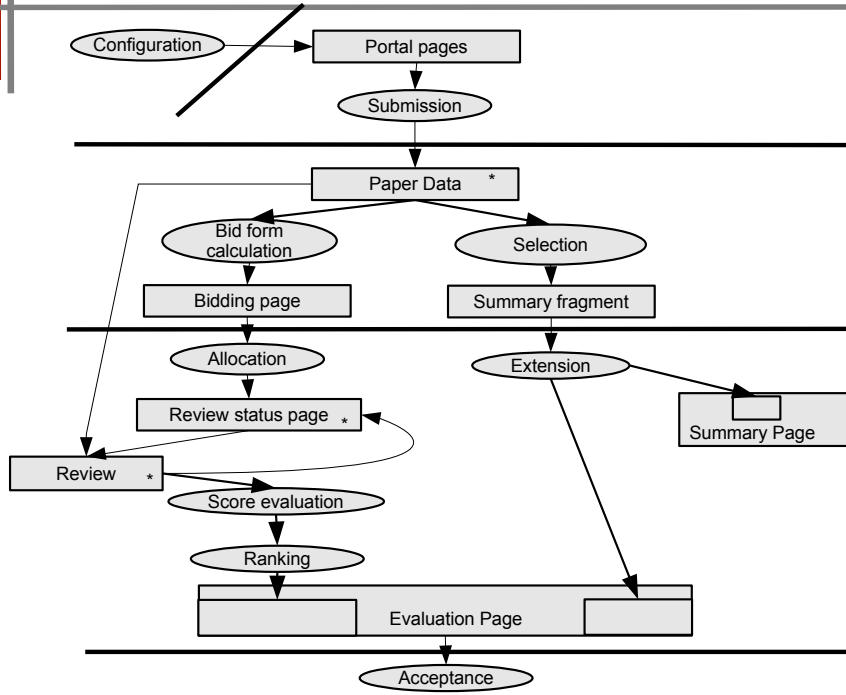
Example: The *START* Conference

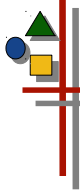
- ▶ START is a review management system
 - It has a 5-phase staged template expansion architecture
 - START servlets are composition scriptlets that generate html-templates
- ▶ Using invasive composition, we developed a staged *expansion* system
- ▶ It is no problem to generate servlets, too. There is no special programming



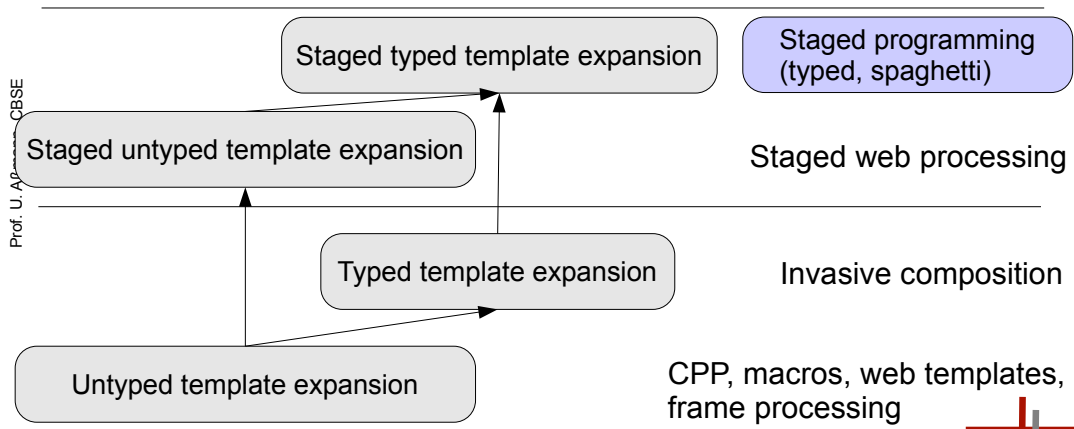
The Staged Template Expansion Architecture of START

Prof. U. Alsmann, CBSE





The Hierarchy of Staged Architectures



Prof. U. Al...
CBSE





54.1.3 Problem 3: Spaghetti Code

and a possible remedy:
staged architectures





Architecture and Composition

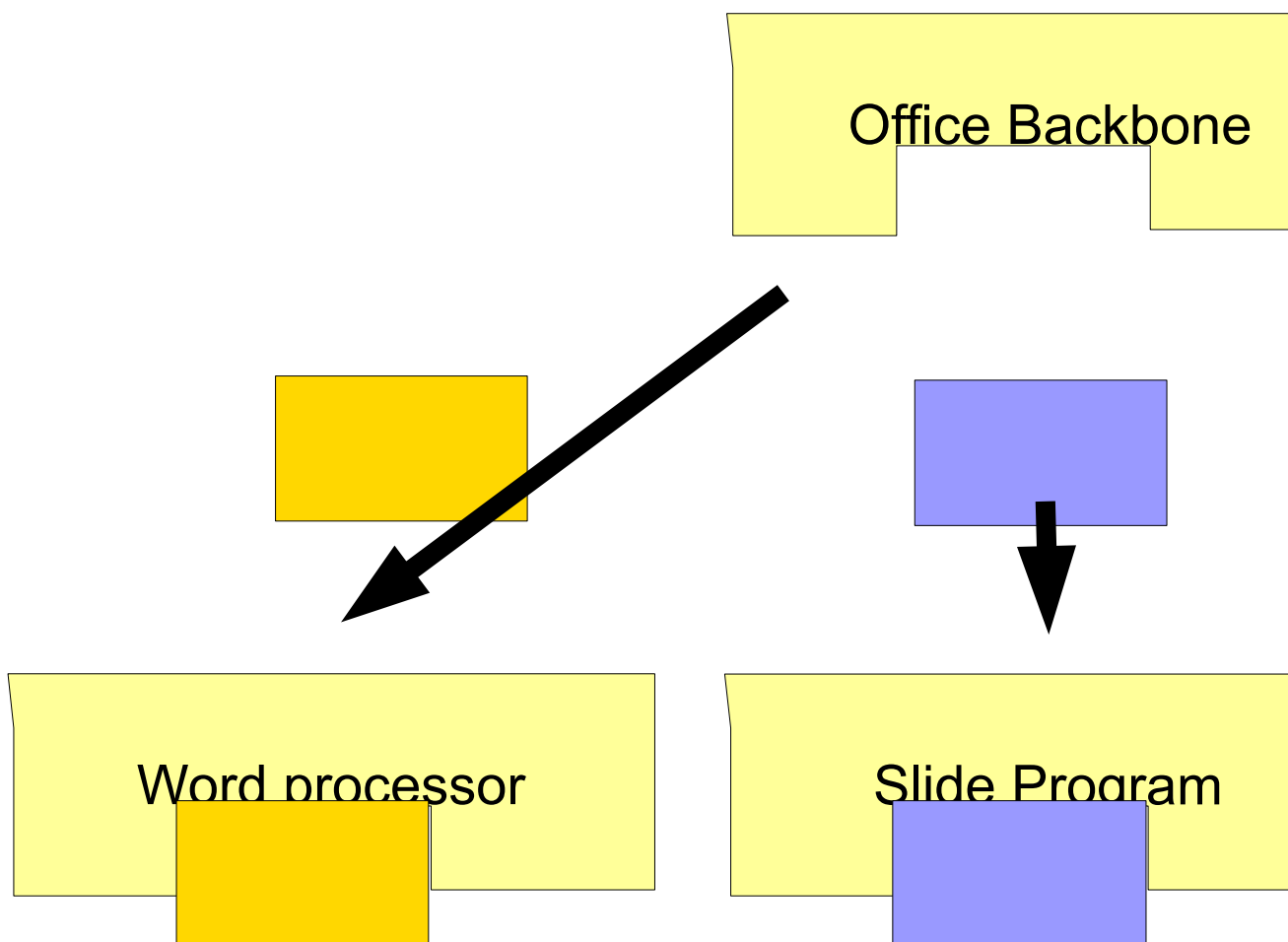
- ▶ Two of the central insights of the software engineering

Separate architecture from the code

Compose components by a *composition*

Benefit of Architectures

- ▶ Comprehensibility
- ▶ Commonalities into the architectural level, vari specific components
- ▶ Does this also hold for web programming?





Less Spaghetti Code: A Fragment-Based Template and its Architecture

Component

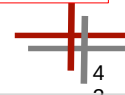
```
<html>
    <hook id="imports">
</html>
<h1> Welcome! </h1>
    <hook id="use">
<p> Today is
    <hook id="year"/>
    <-hook id="day"/>
</p>
<p>
    <hook id="greeting"/>
</p>
</html>
```

Prof. U. Asmann, CESE

Composition Program (Architecture)

```
public class composeTemplate {
    String use = „jspCalendar“;
    String imports=“java.util.*“;

    compose() {
        Template template = read();
        Bean clock = new jspCalendar();
        String year = clock.getYear();
        String day = clock.dayOfTheMonth();
        if (Calender.getInstance().get(Calendar.AM_PM) ==
            Calender.AM)
            greeting = “Good Morning!”;
        else
            greeting = “Good afternoon...“;
        this.merge(template);
    }
}
```





Separation of Components and Architecture Allows for Variants

```
public class composeTemplate {  
    String use =  
    String imports=  
    compose() {  
        String year =  
        String day =  
        greeting =  
    }  
}
```

Composition Program (Architecture)

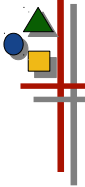
Component 1

Component 2

```
<html>  
    <hook id="imports">  
<h1> Welcome! </h1>  
    <hook id="use">  
<p> Today is <hook id="year"/>  
    -<hook id="day"/>  
</p><p> <hook id="greeting"/>  
</p>  
</html>
```

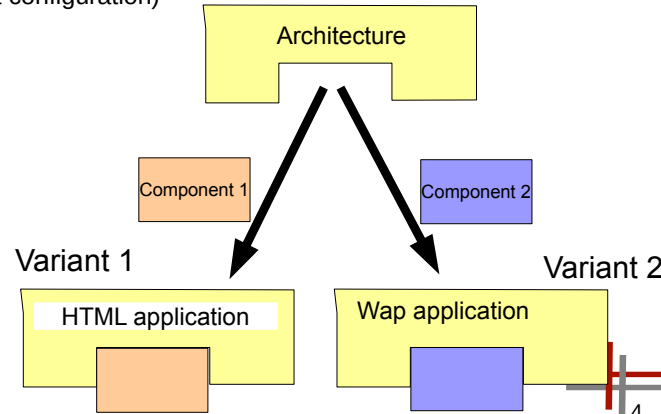
```
<wap>  
    <hook id="imports">  
<bold> Welcome! </bold>  
    <hook id="use">  
<p> Today is <hook id="year"/>  
    -<hook id="day"/>  
</p><p> <hook id="greeting"/>  
</p>  
</wap>
```





Architecture and Variants in a Product Line

- ▶ Advantages for Separating Architecture From Application Components
 - Isolation of commonalities into frameworks
 - Comprehensibility
 - Programming-in-the-large is separated from programming-in-the-small, components can be abstracted away
 - Less spaghetti
 - Easy variability (variant configuration)





Variant Management by Control Flow in Architectural Composition Programs

```
public class composeTemplate {  
    if (HTML) then use component 1  
    else use component 2  
  
    String use =  
    String imports=  
    compose() {  
        String year =  
        String day =  
        greeting =  
    }  
}
```

Variant 1

```
<html>  
    <hook id="imports">  
<h1> Welcome! </h1>  
    <hook id="use">  
<p> Today is <hook id="year"/>  
    -<hook id="day"/>  
</p><p> <hook id="greeting"/>  
</p>  
</html>
```

Variant 2

```
<wap>  
    <hook id="imports">  
<s1> Welcome! </h1>  
    <hook id="use">  
<p> Today is <hook id="year"/>  
    -<hook id="day"/>  
</p><p> <hook id="greeting"/>  
</p>  
</wap>
```





Definition: Staged Data-Flow Arc

Staged data-flow architectures add an expansion to staged template processing

- ▶ Every stage is executed to produce **data** for the next stage
- ▶ Every stage is executed at a specific time
- ▶ On every stage, there is
 - an architecture,
 - a component model
 - a composition technique,
 - and a composition language
- ▶ Every composition language has its own interpretation
 - and is reduced (expanded) at different interpretation times



Web Programming needs Staged Data-Flow Architectures

- ▶ It would be nice to extend staged typed template expansion in web engineering to
- ▶ staged data-flow architectures.

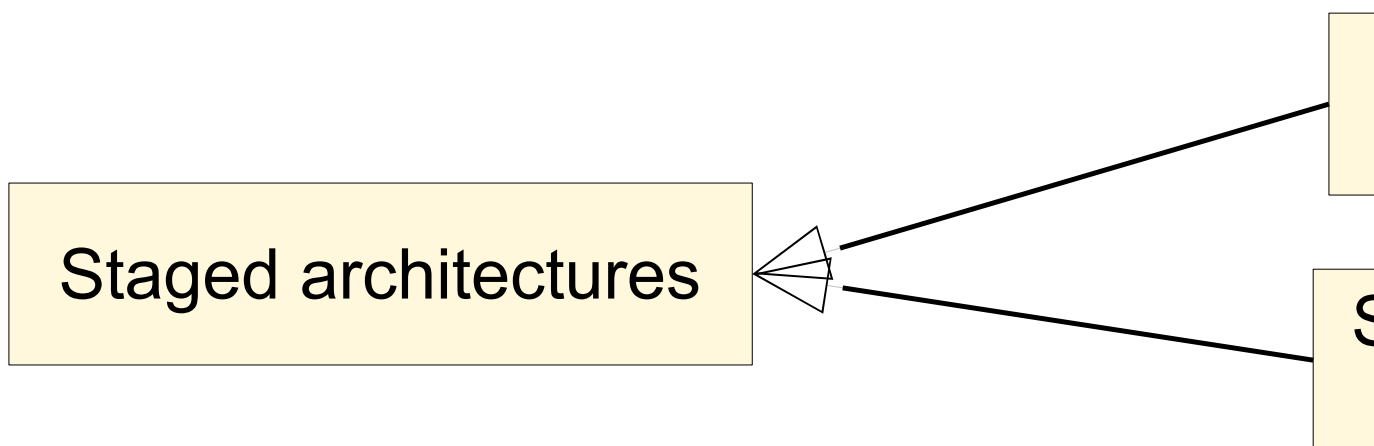
Prof. U. Altmann, CESE



Definition: Staged Architectures

Staged meta-programming architectures
programming with an explicit architecture

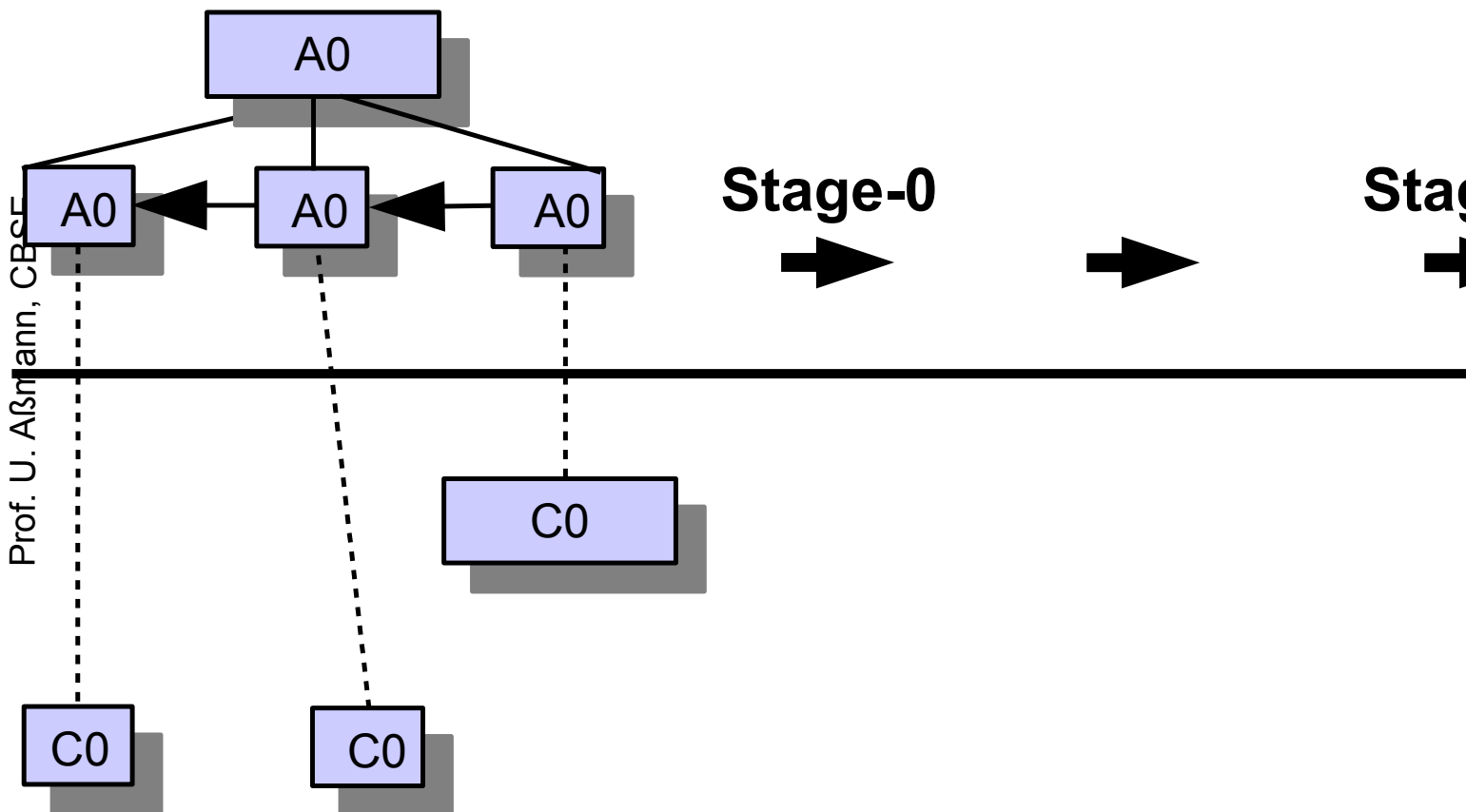
- ▶ Every stage is executed to produce **code** for the next stage
 - The final runtime code (architecture and components) is produced in the final stage
 - The initial architecture is very small, the final architecture is large
 - Composition expressions, specifications, or programs are used to generate the components of a previous stage



Staged Metaprogramming Architecture Large from Small

Stage-A0 architecture in
composition language A0
Component language C0

Gen
Stag
com
Com

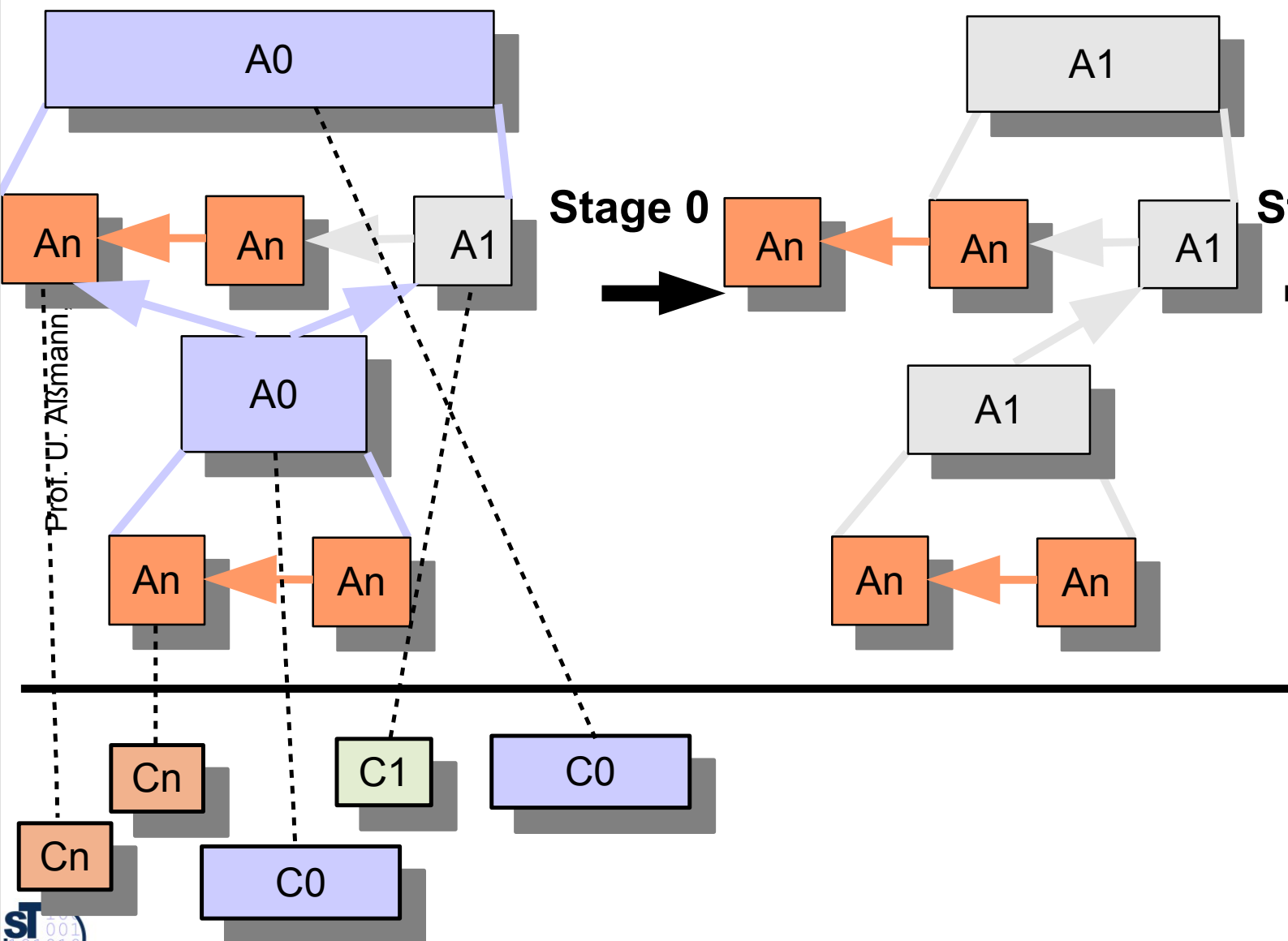


Staged Metaprogramming Architecture

Different Component Models on

Stage-0 architecture in
composition language A0
Component language C0

Stage 0 produces
Stage-1 architecture in
composition language
Component language



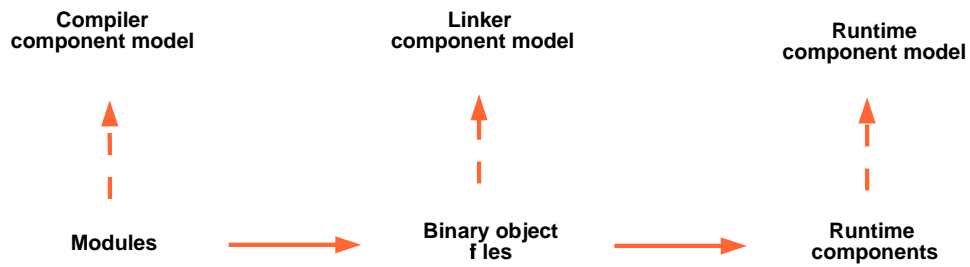
52.4 Staged Metaprogramming Architectures in Software Engineering



Build Management is Staged Composition

- ▶ Software build management is code composition in several stages
- ▶ Composition language: Make, ant, maven, etc.
 - Make is a composition tool with a lazy rule-based language
 - Expressions are applications of UNIX tools (compiler, linker, generator, preprocessor)
- ▶ Different component models on all stages

Prof. U. Alsmann, CESE





Invasive Software Composition

- ▶ Produces code from typed templates by parameterization and expansion

Prof. U. Altmann, CESE

Stage-0
Composition level
language: Java

Stage-1
language: Java

Fragment
component model

Runtime
component model
(objects)



Stage-0

Code Fragment
Components



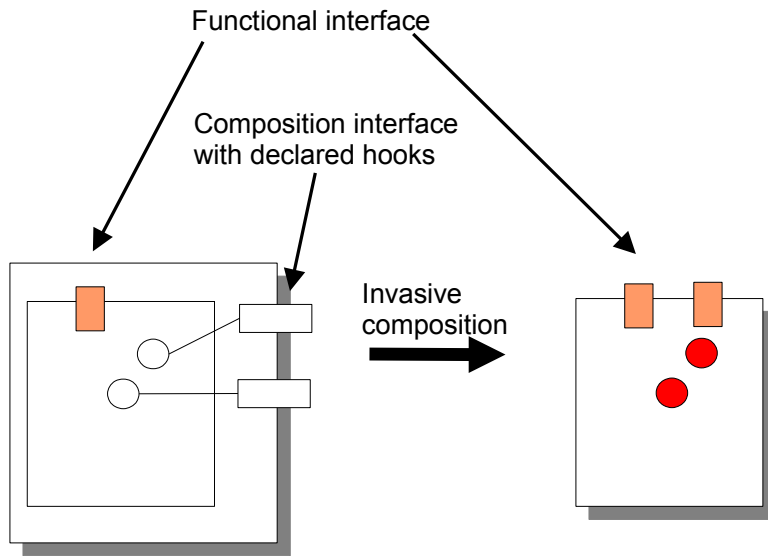
Runtime
components





Invasive Composition Produces Functional from Composition Interfaces

- ▶ Two different component models



Prof. U. Alsmann, CESE





Component Models on Different Levels in the Software Process

- ▶ Standard COTS models are just models for binary code

Prof. U. Altmann, CBSE

**Fragment
component model**



**Code Fragment
Components**



**Generic COTS
component model**



**COTS
components**



**Run time
component model**



**Run time
components**





The Dresden Staged Architecture Development Process

- ▶ Fix the stages
 - Decide on a staged processing or programming architecture
- ▶ Fix the component models for every stage
 - Interface concepts, composition operations, composition language
 - Design a concrete component model with Reuseware toolkit
- ▶ Fix the architectures
 - Decide on a composition language on each level
- ▶ Fix the variant management
- ▶ Fix the components

- ▶ And you'll have a pretty comprehensible product line!

Prof. U. Altmann, CESE





The Vision of Staged Systems

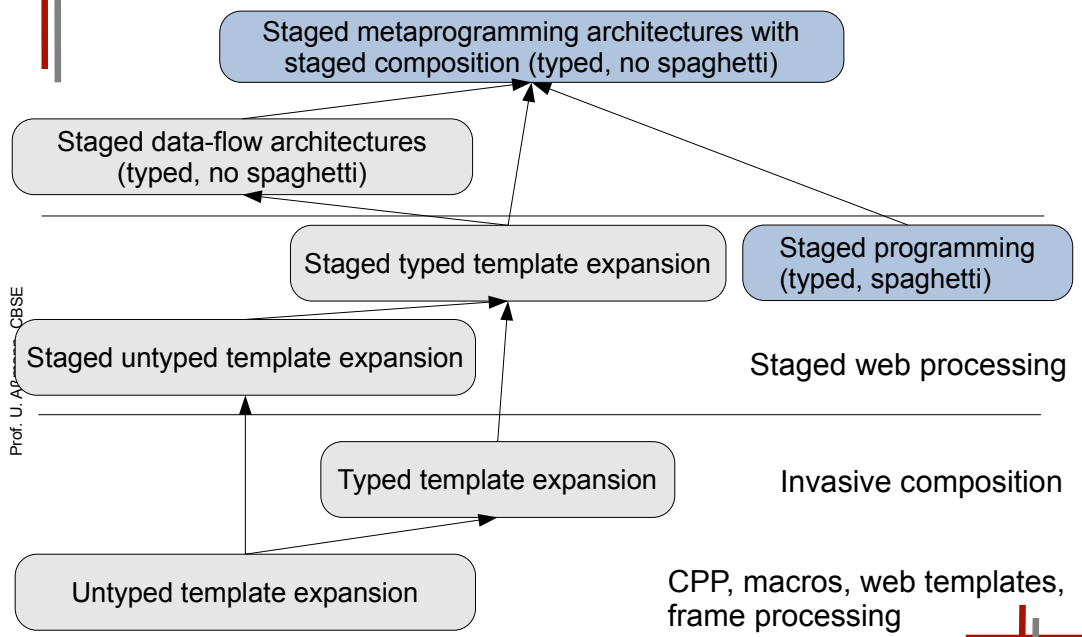
- ▶ The staged programming principle is powerful, so future systems will employ it
- ▶ We need tools to support staged architectures
 - Visualize them
 - Debug them
 - Support the component models on all stages
- that's a lot of work...

Prof. U. Altmann, CESE





The Hierarchy of Staged Architectures



Prof. U. Aditya
CBSE





What Have We Learned?

- ▶ Large systems have *staged architectures* based on
 - *staged programming*,
 - *architectures*,
 - and *typed composition*
- ▶ On every stage, there is a component model and composition system
- ▶ All component models, composition systems and architectures have to work in synchronization
- ▶ Special cases:
 - The refinement-based software process (e.g., MDA)
 - Web systems, active documents
 - Invasive software composition
 - Standard build management

Prof. U. Altmann, CESE





The Beauty of a Staged Programming Architecture

Prof. U. Alsmann, CBSE





The End

- ▶ www.easycomp.org
- ▶ <http://www.the-compost-system.org>
- ▶ U. Aßmann. Invasive Software Composition, 2003, LNCS, p. 2-44, Springer-V.
- ▶ U. Aßmann. Architectural Styles for Active Distributed Systems. “Software Composition” Science of Computer Programming, 2005, LNCS 3176, Springer-V.
- ▶ Walid Taha. A Gentle Introduction to Multi-Stage Program Generation, 2003, LNCS, p. 2-44, Springer-V. <http://www.springerlink.com/index/JEMT0D8V>
- ▶ Tim Sheard: Accomplishments and Research in Logic Programming. SAIG 2001: Proceedings of the Workshop on Semantics, Applications, and Implementation of Logic Programming, pp. 2-44, LNCS 2196, Springer-V.