

# 50) Transconsistent Composition and Active Documents for Component-Based Document Engineering (CBDE)

Prof. Dr. Uwe Aßmann

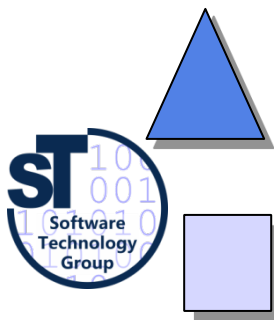
Florian Heidenreich

Technische Universität Dresden

Institut für Software- und  
Multimediatechnik

<http://st.inf.tu-dresden.de>

Version 11-0.1, Juli 6, 2011



1. Problems of Document Composition
2. Invasive Document Composition
3. Invasive Architectures for Active Documents
4. Transconsistency
  1. A Graph-Theoretic Definition of Transconsistency
  2. Transconsistent Architectures
5. Architectural Styles for Transconsistent Architectures



## Literature

- ▶ U. Aßmann. Architectural Styles for Active Documents.  
<http://dx.doi.org/10.1016/j.scico.2004.11.006>



# Overview

- ▶ Some problems in document processing
  - And why they require document architecture
- ▶ Invasive composition of active documents
- ▶ Export declarations as a basis for architecture of active documents
- ▶ Features of acyclic, interactive architectures
  - Transconsistency, a novel evaluation concept for active documents
  - Transconsistent architectural styles for active documents
- ▶ Conclusions for web engineering



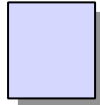
# Architecture and Composition

- ▶ One of the central insights of the software engineering in the 1990s is:

Separate architecture (composition)  
from  
the base components

- ▶ Purpose: Get a second level of variability
  - Architecture and components can be varied independently of each other
  - Scale better by different binding times of composition programs
  - Be *uniform* for many products of a product family
- ▶ However, how to be uniform also for documents?

# *50.1) Problems in Document Construction*





# Some Problems

## 1 – `\cite in LaTeX`

- ▶ As already McIlroy.68 has shown, we need components for a ripe industry

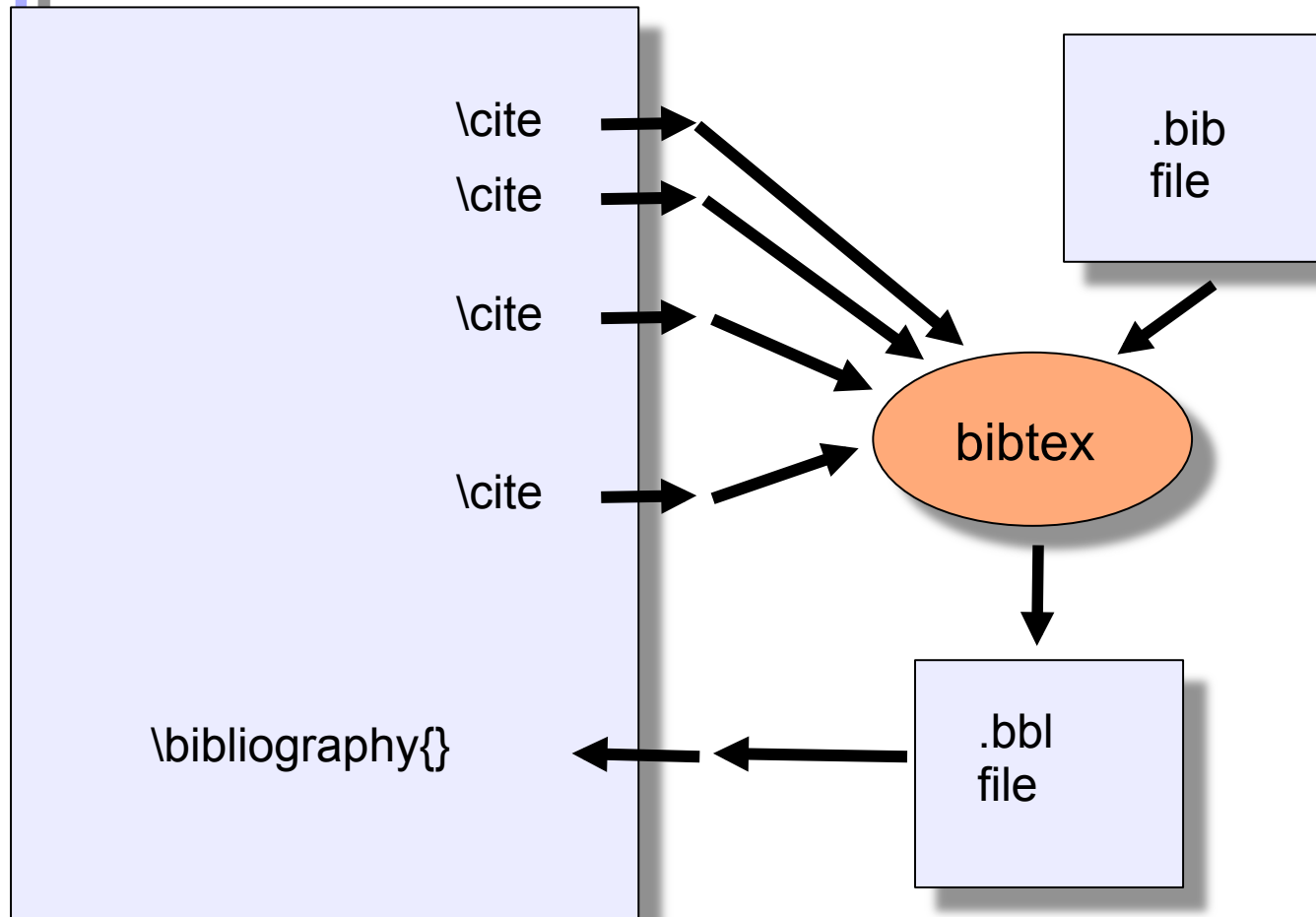
```
@InProceedings{ mcilroy.68b,  
  author      = "M. Douglas McIlroy",  
  title       = "Mass-Produced Software Components",  
  booktitl    = "Software Engineering Concepts and Techniques (1968 {NATO}  
                Conference of {S}oftware Engineering)",  
  editor      = "J. M. Buxton and Peter Naur and Brian Randell",  
  publisher   = {NATO Science Committee},  
  pages       = "88--98",  
  month       = oct,  
  year        = "1968"  
}
```



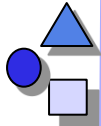
## Usual Solution

- ▶ Problem: Document is *active*, i.e., contains generated components
- ▶ Procedure:
  - Latex writes citation to .aux-file
  - bibtex greps them and produces a .bbl file
  - .bbl file is included into document
- ▶ How does the architecture of a latex document look like that regenerates all generated components?

# Maybe Like This...



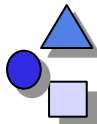




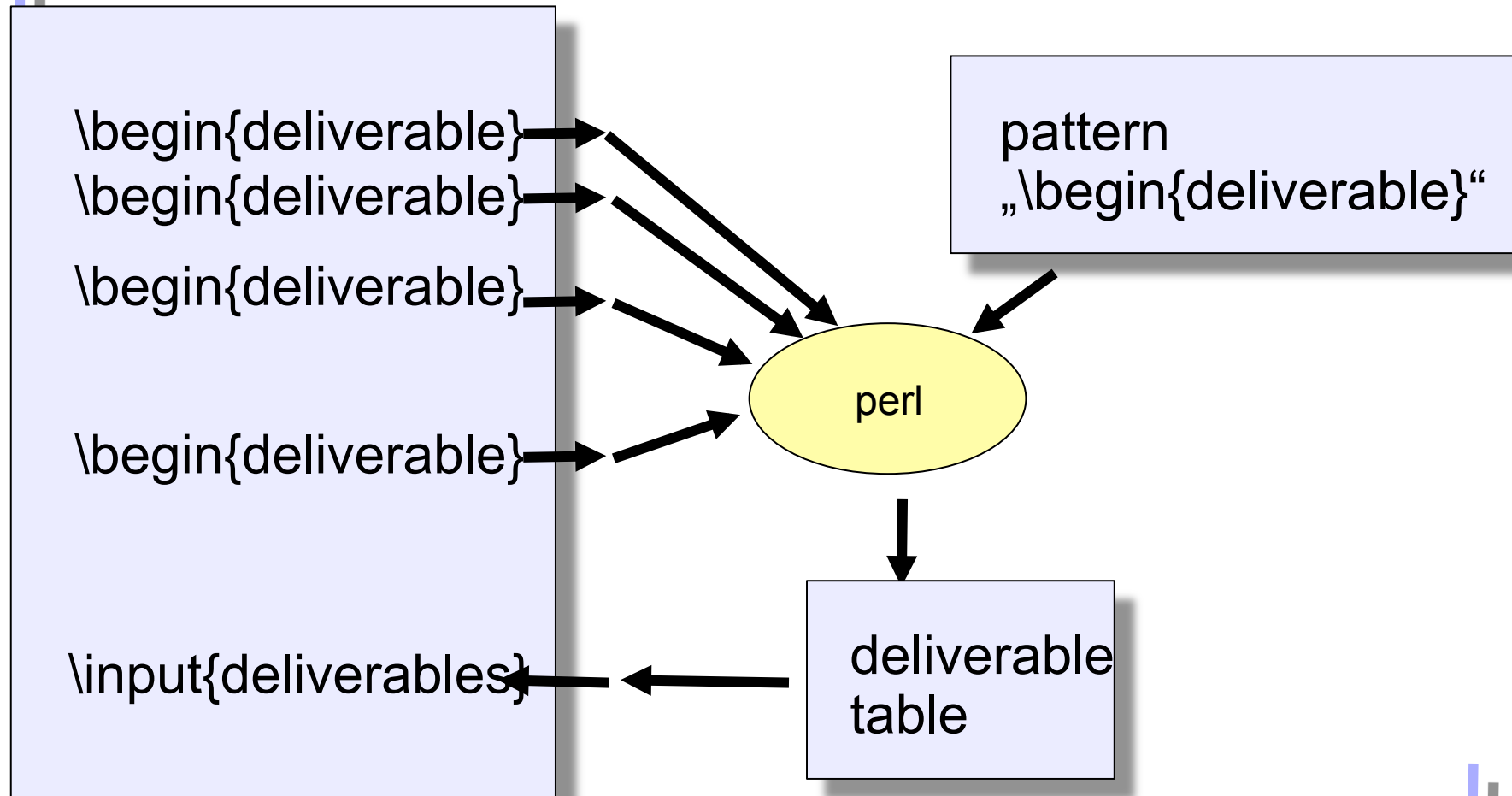
## ***Problem 2 – Deliverable Definitions in LaTeX Project Plan***

```
\begin{deliverables}
EASYCOMP workshop I      &\DIS.1.1 & \UKA & 12 & W & PU & 18 \\
EASYCOMP workshop II    &\DIS.1.2 & \UKA & 12 & W & PU & 30 \\
Web-based Composition Centre &\DIS.2 & \UKA & 3 & H & PU & 36 \\
Composition Handbook     &\DIS.3 & \UKA & 14 & R & PU & 24 \\
Final Report             &\DIS.4 & \UKA & 6.5 & R & CO/PU & 36 \\
\end{deliverables}
```

- ▶ Procedure:
  - extract deliverables by perl script
  - concat to latex table
  - include table
  
- ▶ How does the architecture of that document look like?

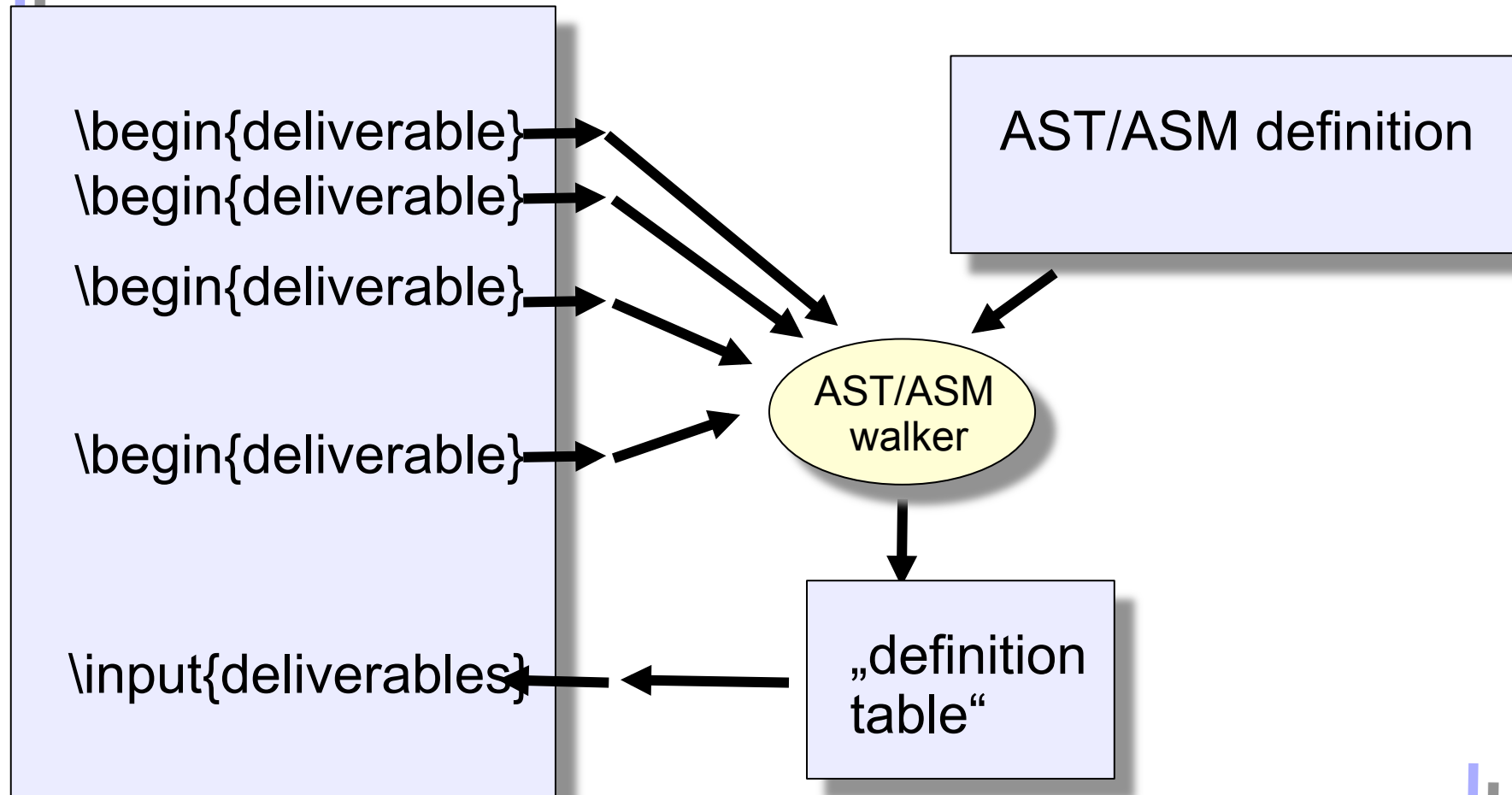


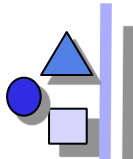
## Like This...



# Query Should Use the AST/ASM

- ▶ Regular expressions are too weak





## Problem 3 – A Simple Web Page, Generated By a Database

```
<html>
..
<table>
  <tr> <td> Employee </td> <td> Address </td> </tr>
  <tr> <td> Uwe Assmann </td> <td> Farhagsvägen 128 </td> </tr>
  <tr> <td> Robert Kaminski </td> <td> Platensgatan 9 </td> </tr>
  <tr> <td> Jens Gustavsson </td> <td> Stora Torget 14 </td> </tr>
</table>
..
</html>
```

### ► Procedure:

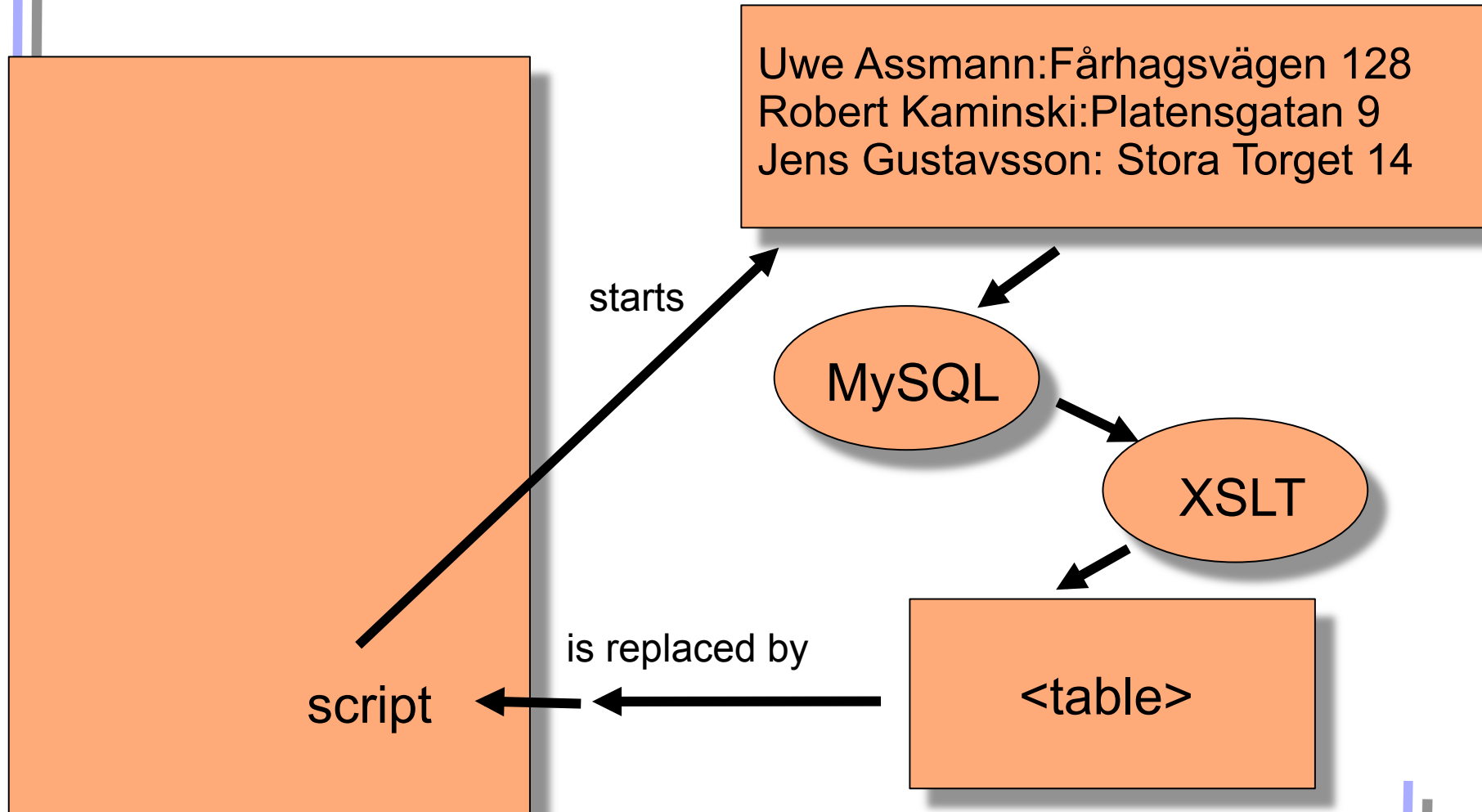
- Run the embedded script of an HTML template
- Start SQL query in MySQL
- Transform (with XSLT) the plain text to HTML
- Include table and replace the embedded script

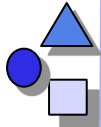
### ► How does the architecture of that document look like?





## Like This...





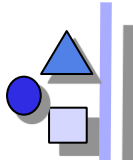
## *Problem 4: Electra Spreadsheet*

- ▶ Used for contract negotiations about project budgets with the EC
- ▶ About 10 summary pages, generated from participant figures
- ▶ 4 pages per participant
  
- ▶ No architecture available....



## Conclusion

- ▶ Why don't we define document architectures?
  - That allows for extracting the architecture and separating it from „components“
- ▶ Software architecture and composition have been successful for
  - Developing in the large
  - Software reuse
- ▶ Why don't we define a *document architecture language*?
  - That allows for expressing the coarse grain structure of documents?
  - And unify it with software architecture / software composition?

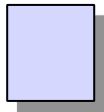
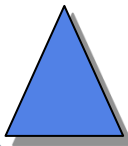


## ***But An Architectural Language For Documents is Difficult..***

- ▶ Well, connectors as binding elements between components don't suffice
  - It must be composition operations or other mechanisms (such as AG) that glue the components together
  - We need composition languages for uniform composition
- ▶ There are some other problems...
  - Invasiveness
  - Transconsistency

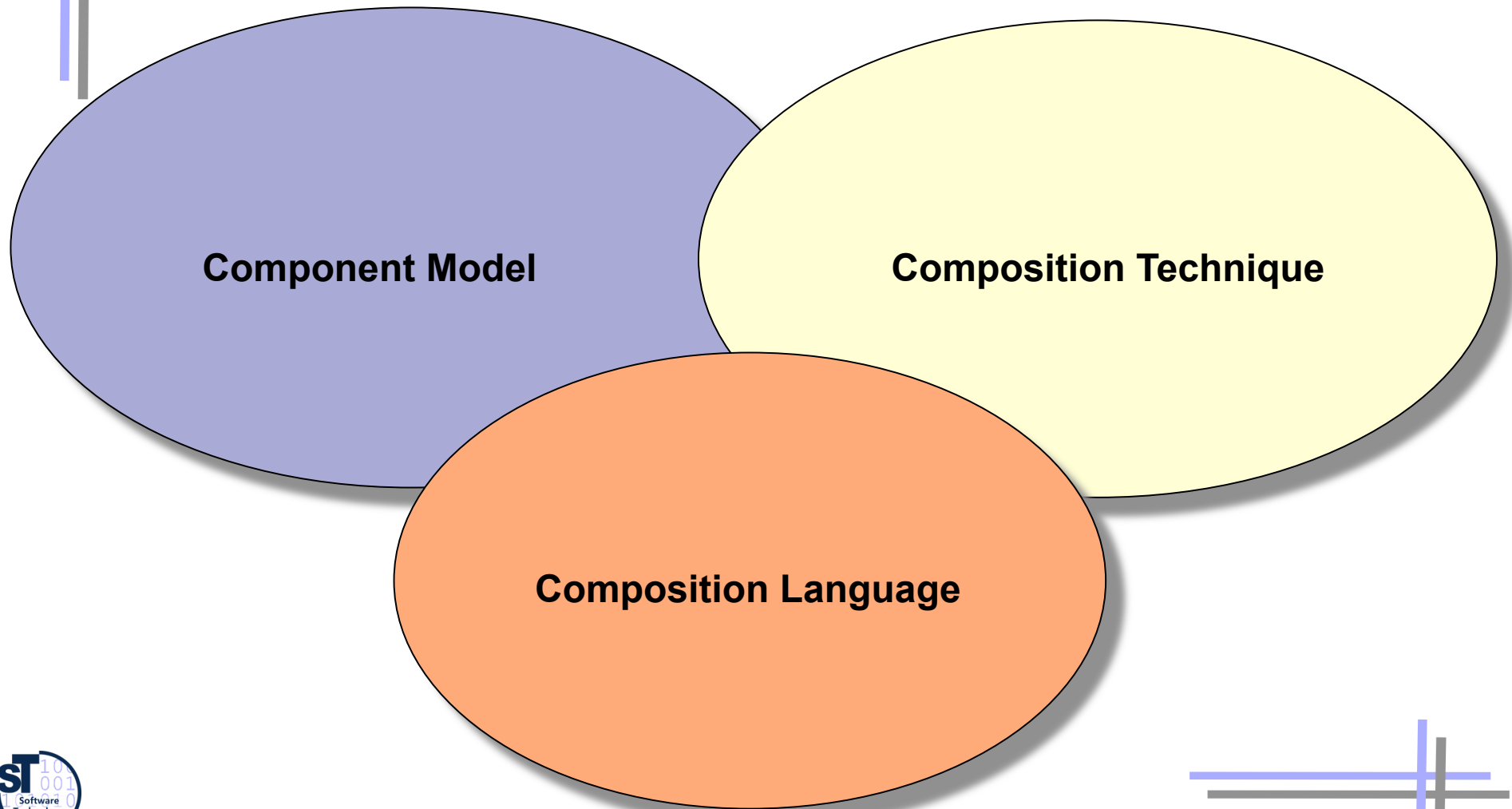


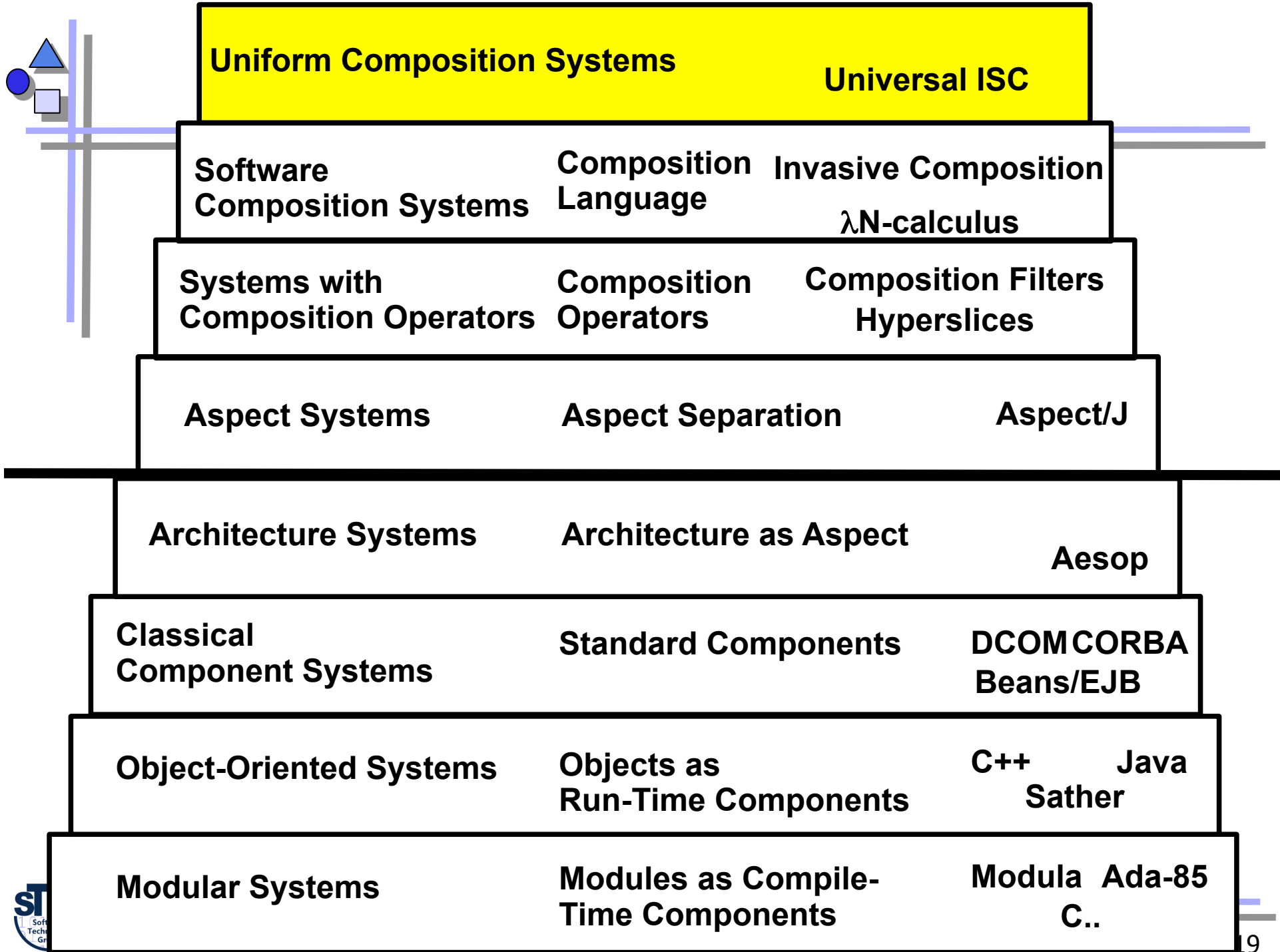
## *50.2) Invasive Composition of Active Documents*





# *The Elements of Composition*







## *For Active Documents, We Need Invasiveness*

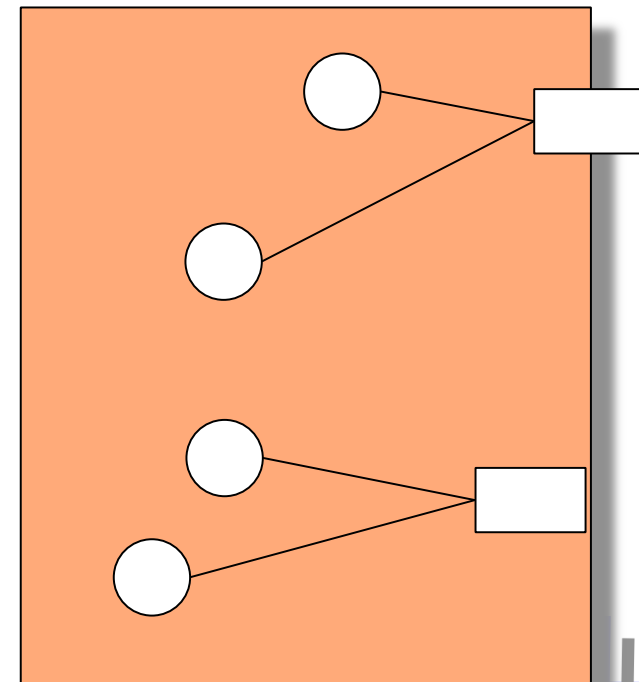
- ▶ Active documents require invasive patching
- ▶ If some parts are changed, others need to be updated
- ▶ Question: are there invasive component models?
- ▶ Answer: yes



# A Greybox Component Model For Uniform Compoc

Invasive **document** composition **adapts** and **extends**  
**Document fragment components**  
**at hooks**  
**by transformation**

- ▶ A **fragment component** is a fragment group of a document language
  - OpenOffice XML, Word XML, AbiWord, many others
- ▶ Uniform representation for
  - Text
  - Pictures
  - Sheets





# *Document Components have Hooks*

**Hooks** are change points of a box:  
fragments which are subject to change

## ■ XML Variation Points

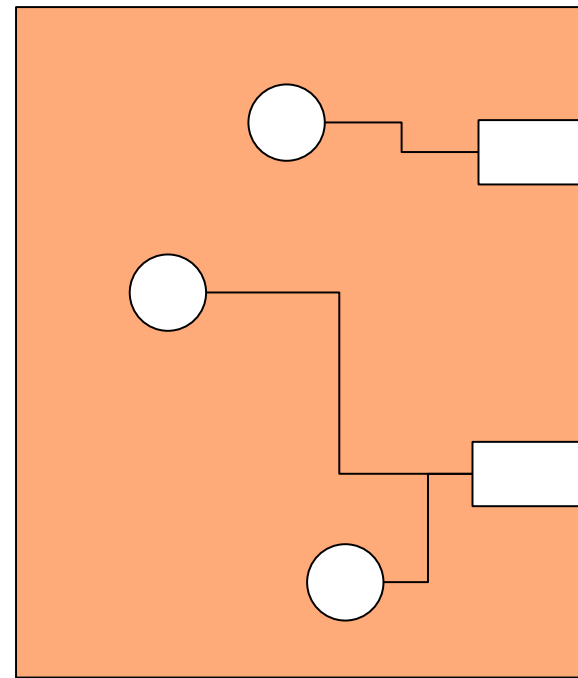
- beginning/end of tag lists
- anchors

## ▶ Software Variation Points

- method entries/exits
- generic parameters



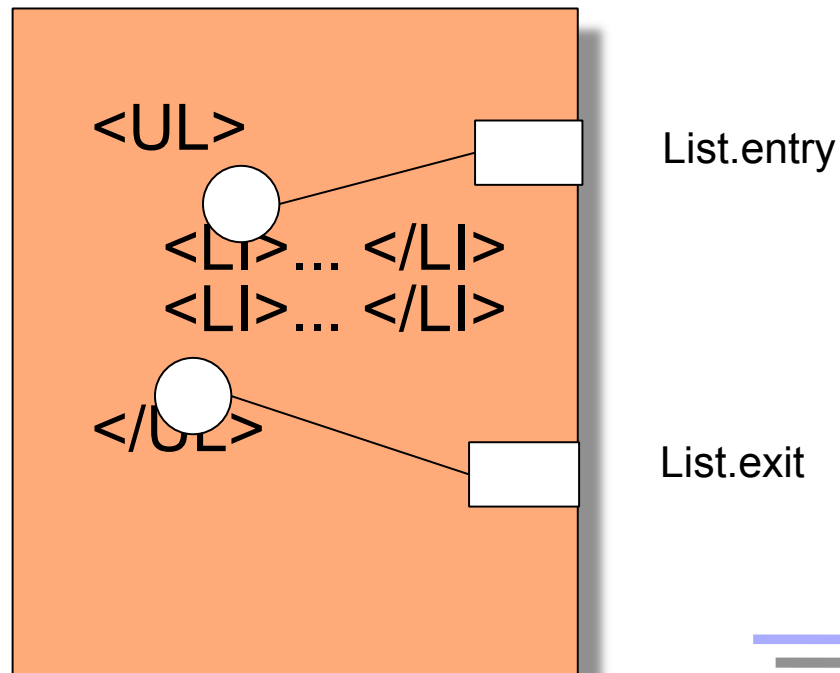
```
<UL>  
→ <LI>... </LI>  
  <LI>... </LI>  
→ </UL>
```





## Implicit Hooks For XML

- ▶ A **hook (extension point)** is given by the document language
  - ▶ In XML given by the DTD or Xschema
- ▶ Hooks can be *implicit* or *explicit (declared)*
  - We draw implicit hooks *inside* the component, at the border
- ▶ Example List Entry/Exit

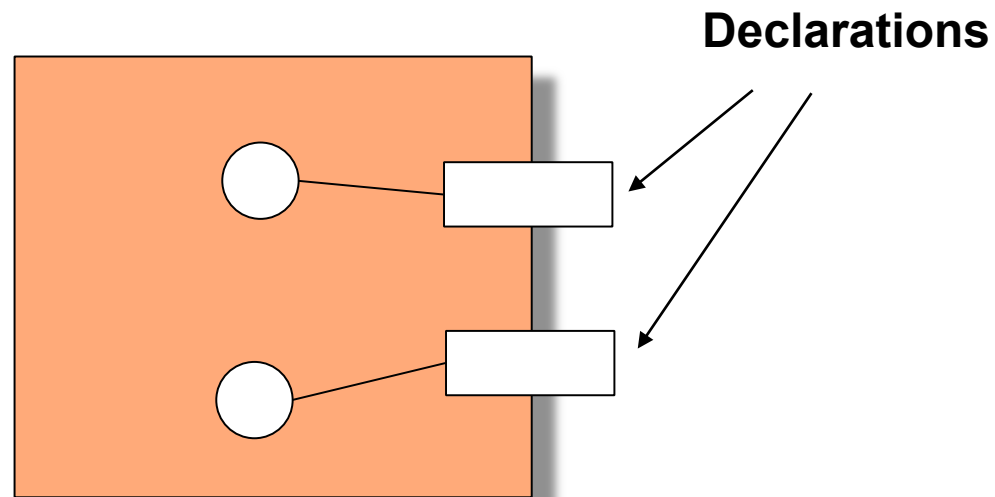






## Slots (Declared Hooks)

- ▶ A **slot** is a variation point (a code parameter)
- ▶ Slots are always *declared*, i.e., declared or explicit hooks
  - They are never implicit, i.e., must be declared by the component writer
  - We draw slots as crossing the border of the component

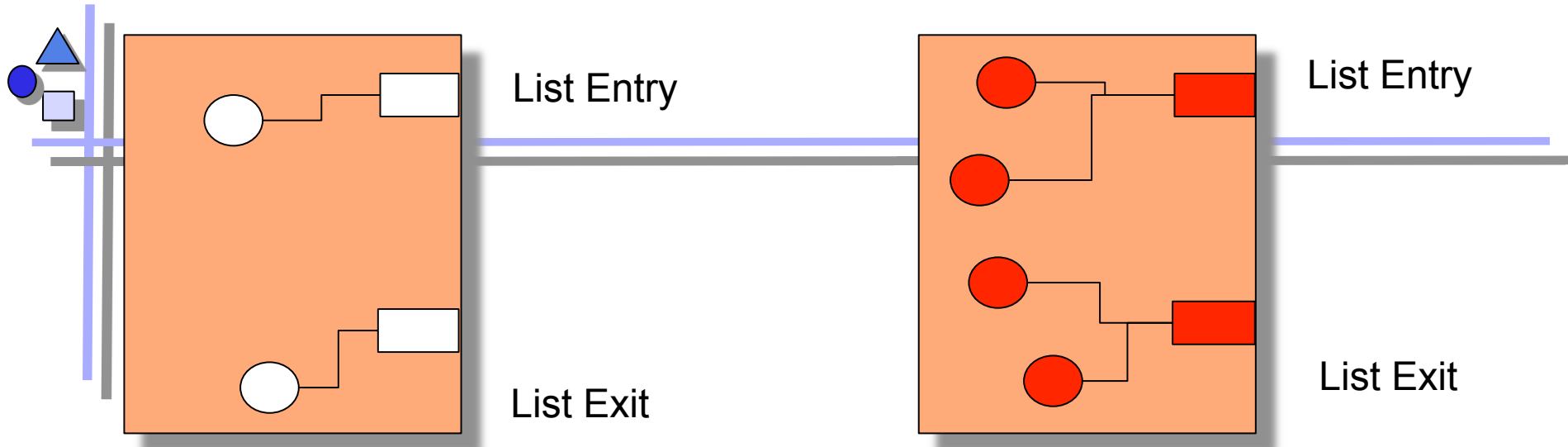




# *The Composition Technique of Invasive Composition*

- ▶ A composer is a tag transformer from unbound to bound hooks  
composer: box with hooks --> box with tags

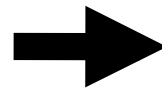
**Invasive Document Composition  
parameterizes and extends  
document components  
at hooks  
by transformation**



```

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

```



```

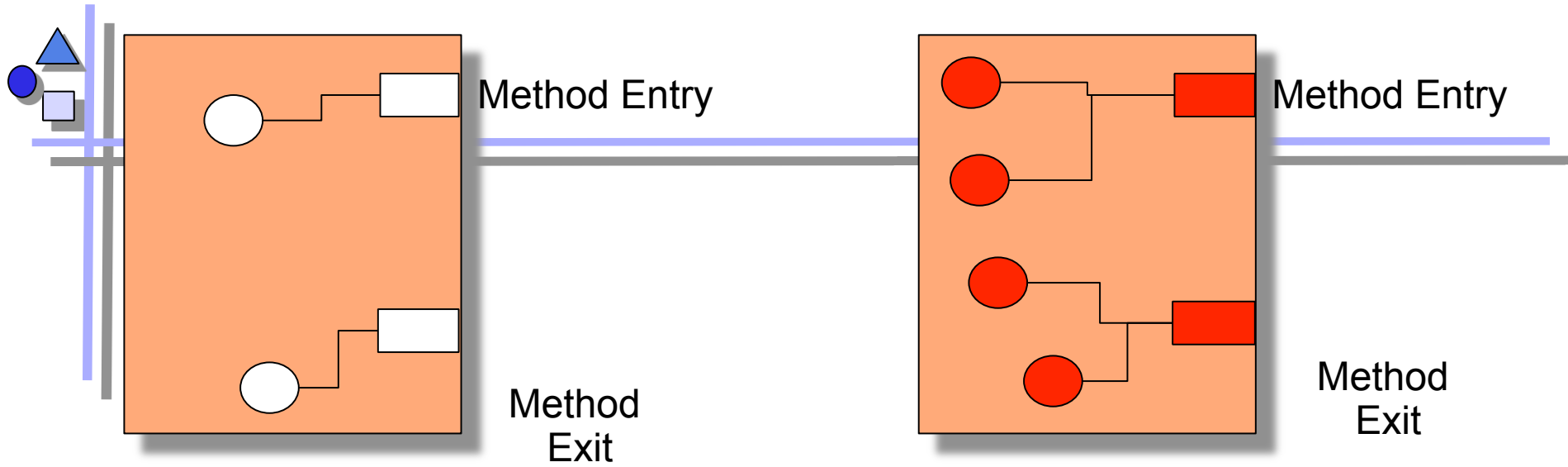
<UL>
  <LI>... </LI>
  <LI>... </LI>
  <LI>... </LI>
  <LI>... </LI>
</UL>

```

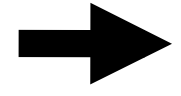
```

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

```



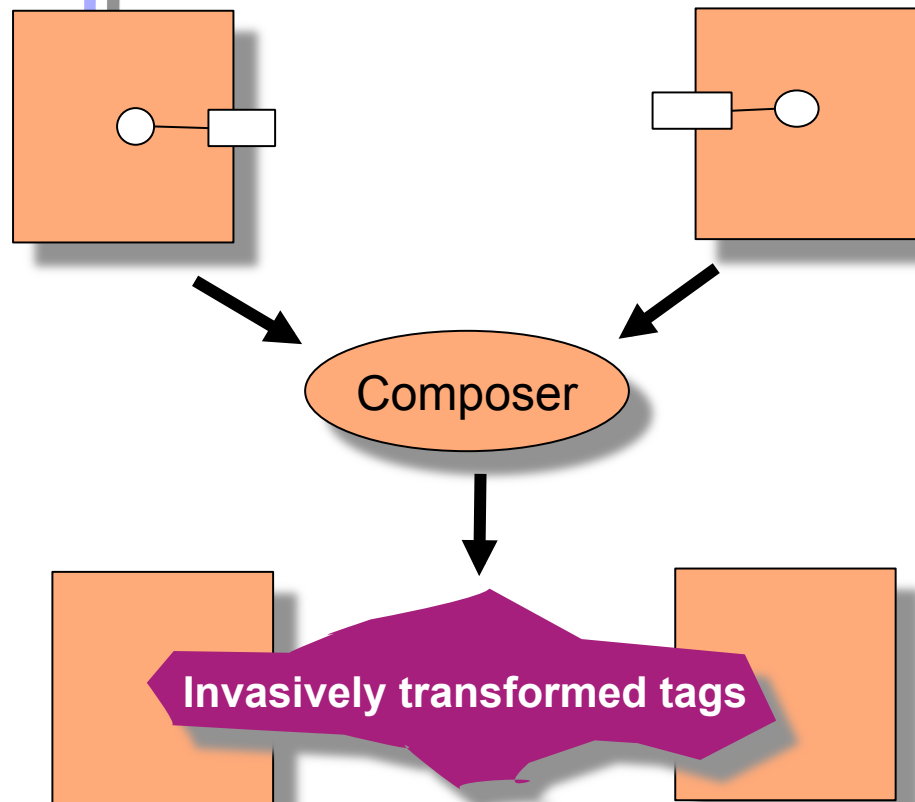
```
m (){
  abc..
  cde..
}
```



```
m (){
  print("enter m");
  abc..
  cde..
  print("exit m");
}
```

```
box.findHook(„MethodEntry“).extend(“print(\\”enter m\\”);”);
box.findHook(„MethodExit“).extend(“print(\\”enter m\\”);”);
```

## ▶ *Invasive Composition*



- ▶ Invasive Composition works uniformly over code and data
- ▶ Allows to compose XML documents uniformly
- ▶ Extend operation implements what we need for document architectures

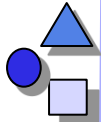


## *Basic Operations on Hooks*

- ▶ bind (parameterize)
- ▶ extend
- ▶ rename
- ▶ copy

## *50.3 Explicit Invasive Architectures for Active Documents*





# Documents Must be Decomposed

- ▶ For architecture of active documents, we need fragment *composition* and *decomposition*
- ▶ For fragment-based composition of documents, other documents need to be decomposed
  - Fragment extraction
  - Fragment selection or query
  - Fragment component search
  - A *fragment query language* is needed
- ▶ In the simplest case, components export all fragments (white-box)
  - Visibility can be controlled by *fragment export languages* forming export interfaces





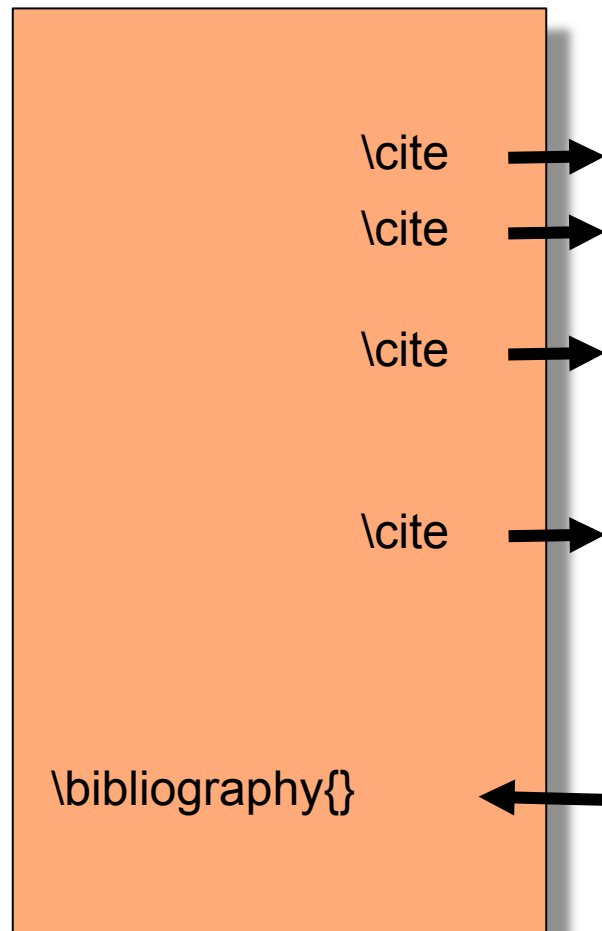
# Fragment Query Languages

- ▶ A *exported fragment* (provided fragment) is defined by a component of an active document and exposes to the external world
- ▶ The programmer declares the exported item in
  - a *fragment export language*
    - a markup language (explicit definition, embedded)
    - Often the explicit specification of exports of fragments is too cumbersome
  - a *fragment query language*
    - a match language (implicit definition, embedded), to select fragments from a component
    - a query language (implicit definition, embedded)
    - a position addressing language (implicit, embedded)
- ▶ In whitebox reuse, fragment export and query language coincide



# Export (and Query) Language 1

- ▶ Basic Operation to Extract Fragments:
- ▶ Match: `ExprInQueryLanguage` → `ExportedDefinitions`



Example 1:  
Query language  
Regular expressions like  
`\cite{.+}`



## *Export (and Query) Language 2*

`\begin{deliverable}` →

`\begin{deliverable}` →

`\begin{deliverable}` →

`\begin{deliverable}` →

`\input{deliverables}` ←

Query language based on  
AST/ASG, together with regular  
expressions



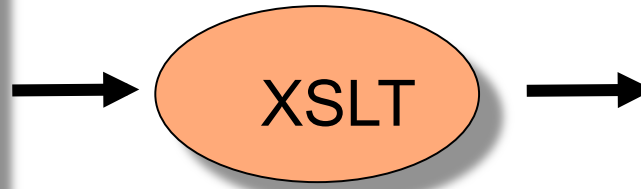
## Query Language 3

Uwe Assmann:Fårhagsvägen 128  
Robert Kaminski:Platensgatan 9  
Jens Gustavsson: Stora Torget 14

Query language:  
Relational algebra,  
started by script

# Another query Language is XSLT

```
<html>
..
<table>
  <tr>
    <td>Employee</td>
    <td>Address</td>
  </tr>
  <tr>
    <td>Uwe Assmann</td>
    <td>Farhagsvägen 128</td>
  </tr>
  <tr>
    <td>Robert Kaminski</td>
    <td>Platensgatan 9</td>
  </tr>
</table>
</html>
```

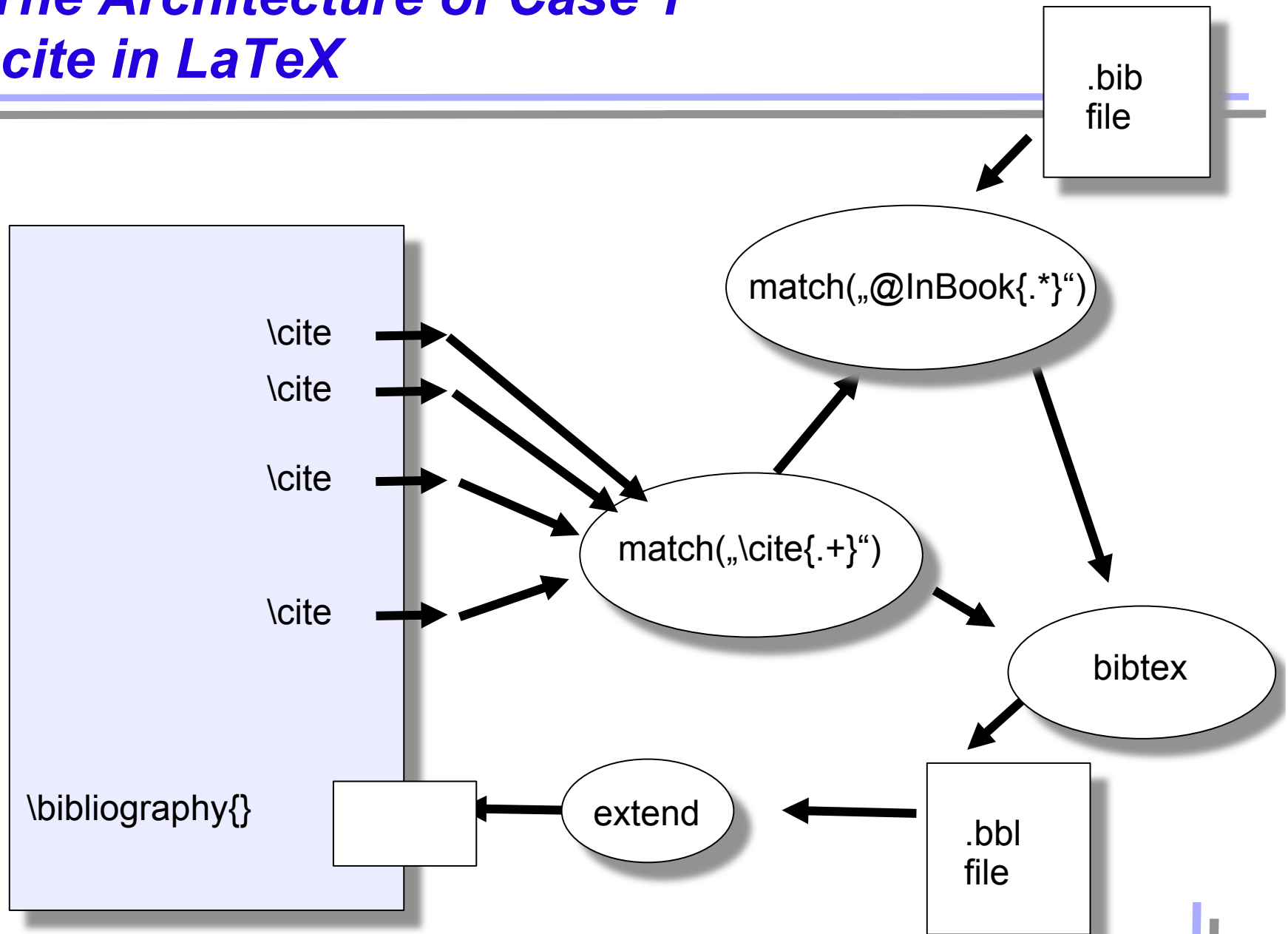




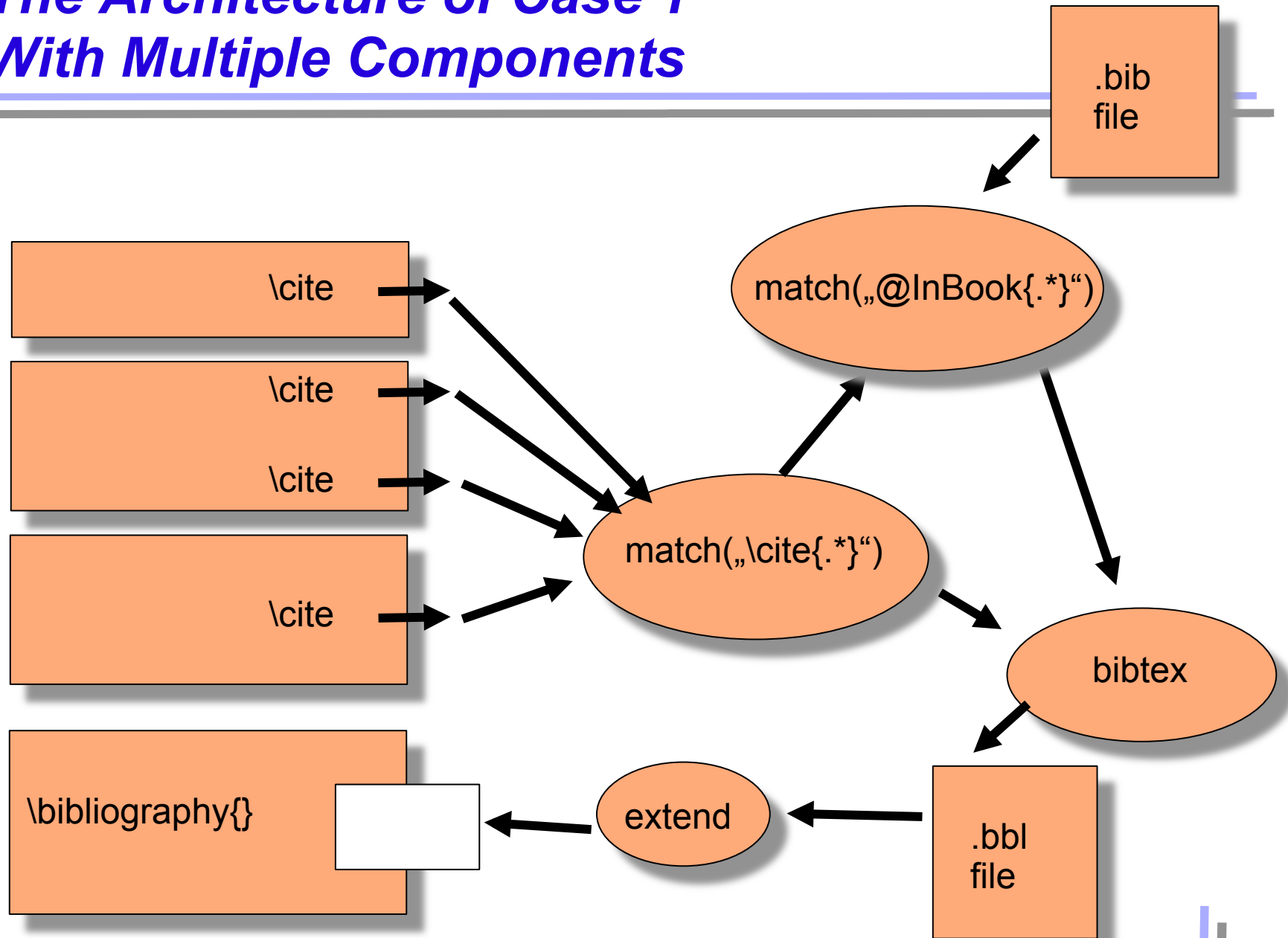
# *Basic Operations on Hooks of Active Documents*

- ▶ bind (parameterize)
- ▶ extend
- ▶ rename
- ▶ copy
- ▶ **match**

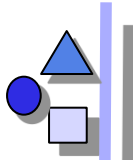
# The Architecture of Case 1 `\cite` in `LaTeX`



# The Architecture of Case 1 With Multiple Components



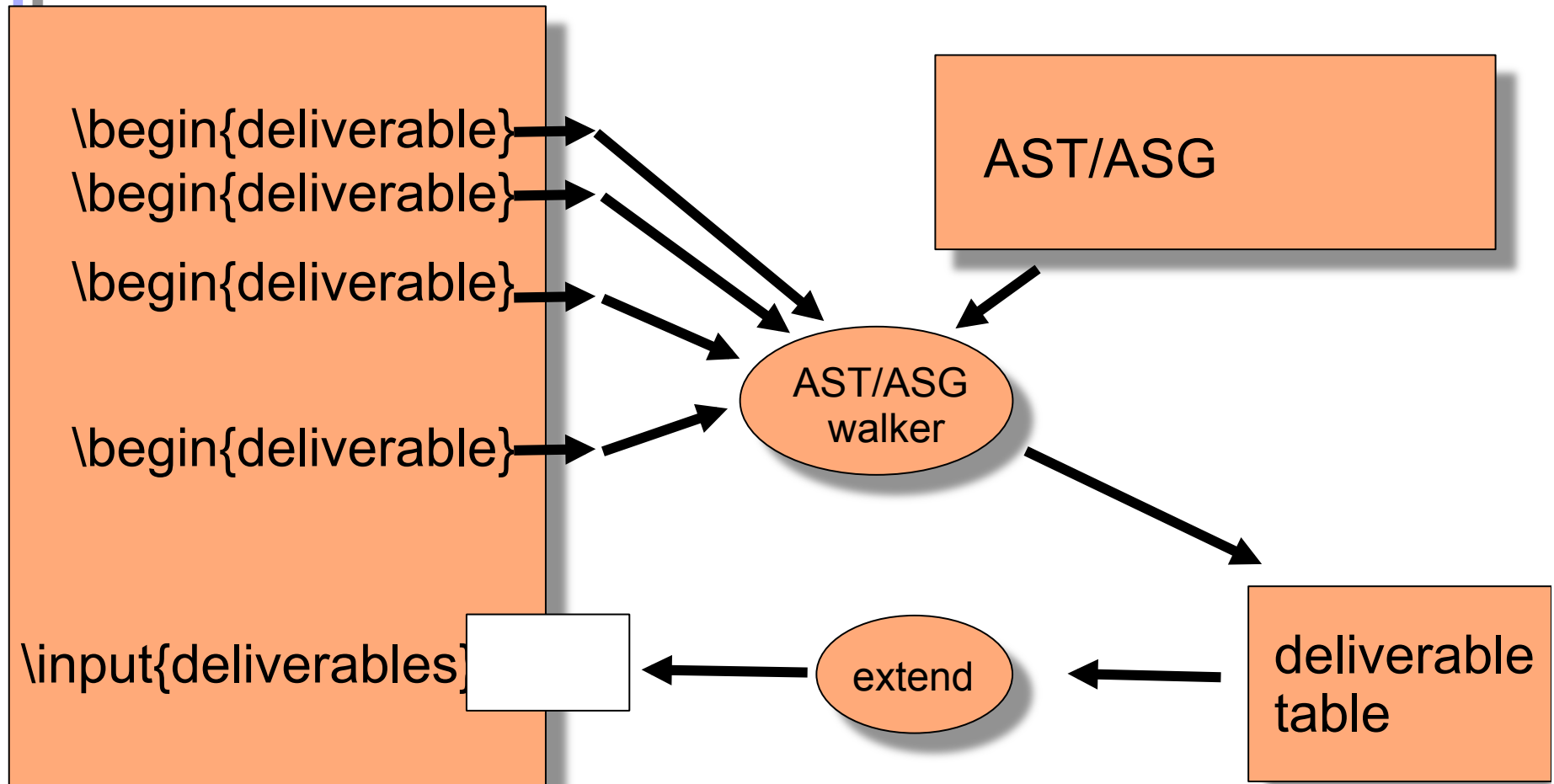




# *Advantages of Export Declarations For Example 1*

- ▶ We have extracted the document's architecture
- ▶ LaTeX becomes simpler
  - query is separated into the composition level
- ▶ Standard language to write the compositions
  - no architectural language required
- ▶ Documents are real components, with a composition interface

# The Architecture of Case 2 Deliverables



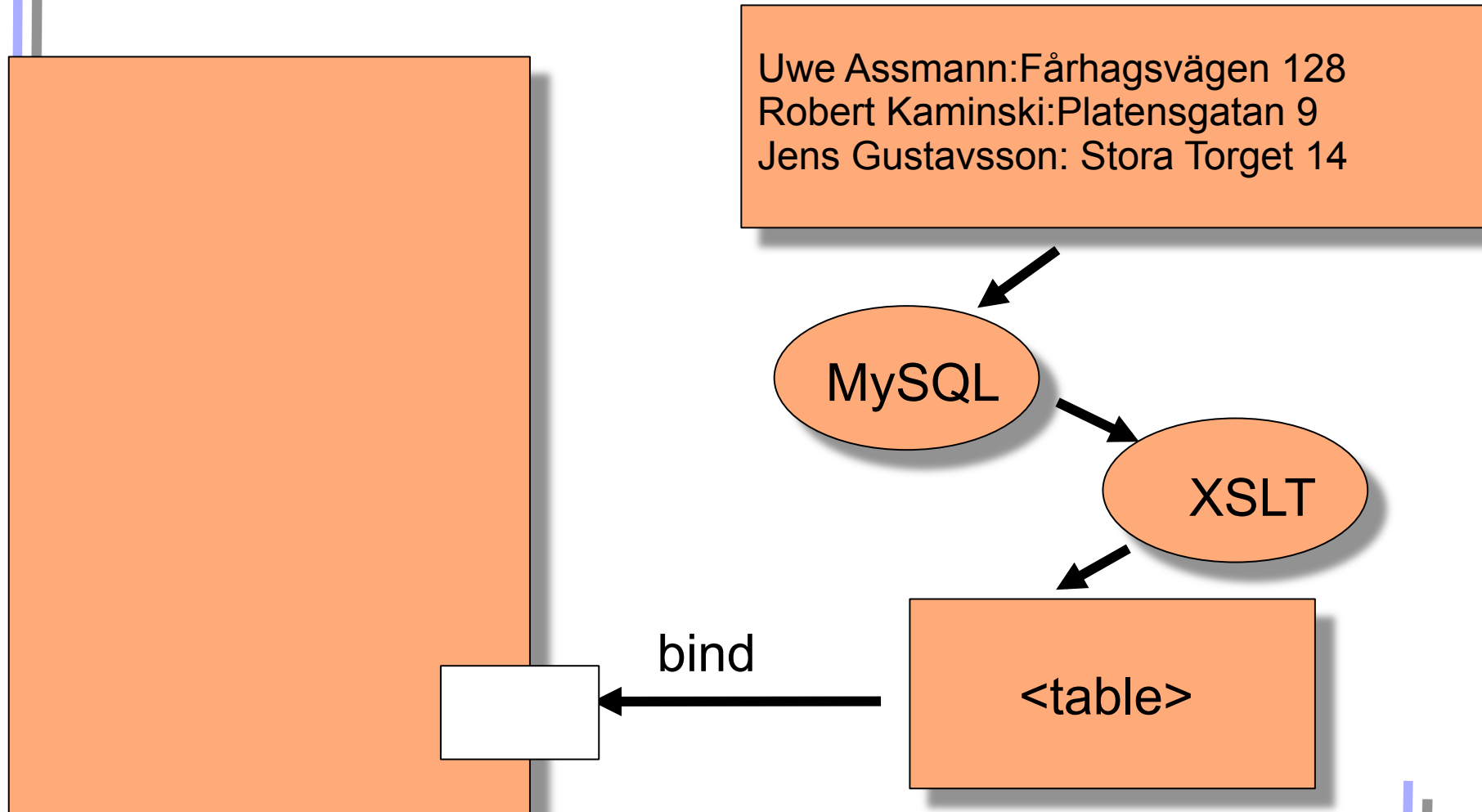


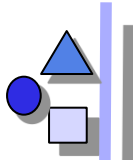
## *Advantages for Example 2*

- ▶ LaTeX cannot interpret the AST
  - and cannot treat relational algebra either
- ▶ We can employ many different definition (query, markup) languages
- ▶ We can employ many different connection and composition languages
  - and write connectors with them
- ▶ Flexible composition approach

# The Architecture of Case 3

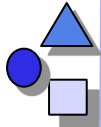
## Database-driven Web Document





# Advantages of Architectures for Active Documents

- ▶ Better reuse
  - Scripts are removed from HTML pages
  - The template can be reused in other contexts where the table expansion is not required
- ▶ A lot of embedded scripts in HTML is composition code
  - let's move it out!
- ▶ Simplifying web engineering



# Afterthought: What Flows Through an Active Document

- ▶ In contrast to a software architecture, in active documents document fragments flow
  - Like in a spreadsheet, the dataflow graph is acyclic (spreadsheet-documents)
  - Generation and modification of values are modeled with export declaration languages (script languages)
- ▶ In contrast to a software architecture, the values only change when the user changes a component
  - Pushed once through that graph, the document is updated
  - Transclusion works for dataflow graphs!
- ▶ **Requirements for Active Document Architectures**
  - *Fragment queries* or export definitions
  - *Invasive embedding* of results
  - *Hot update* of all computations (aka transconsistency)

# ***50.4) Transconsistency – A New Architectural Principle for Hot Update in Composed Active Documents***

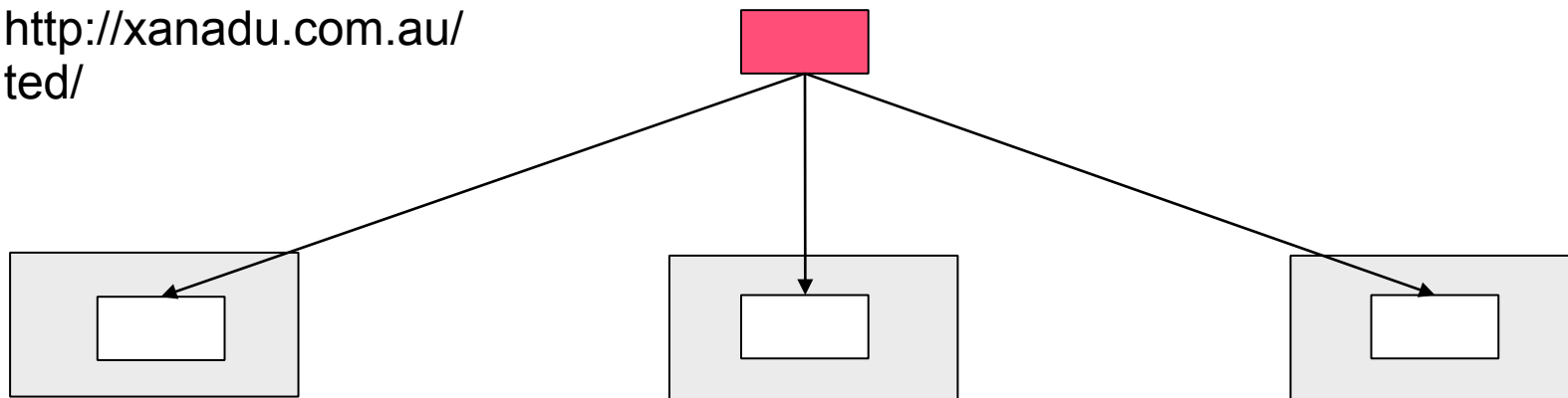




# Transclusion

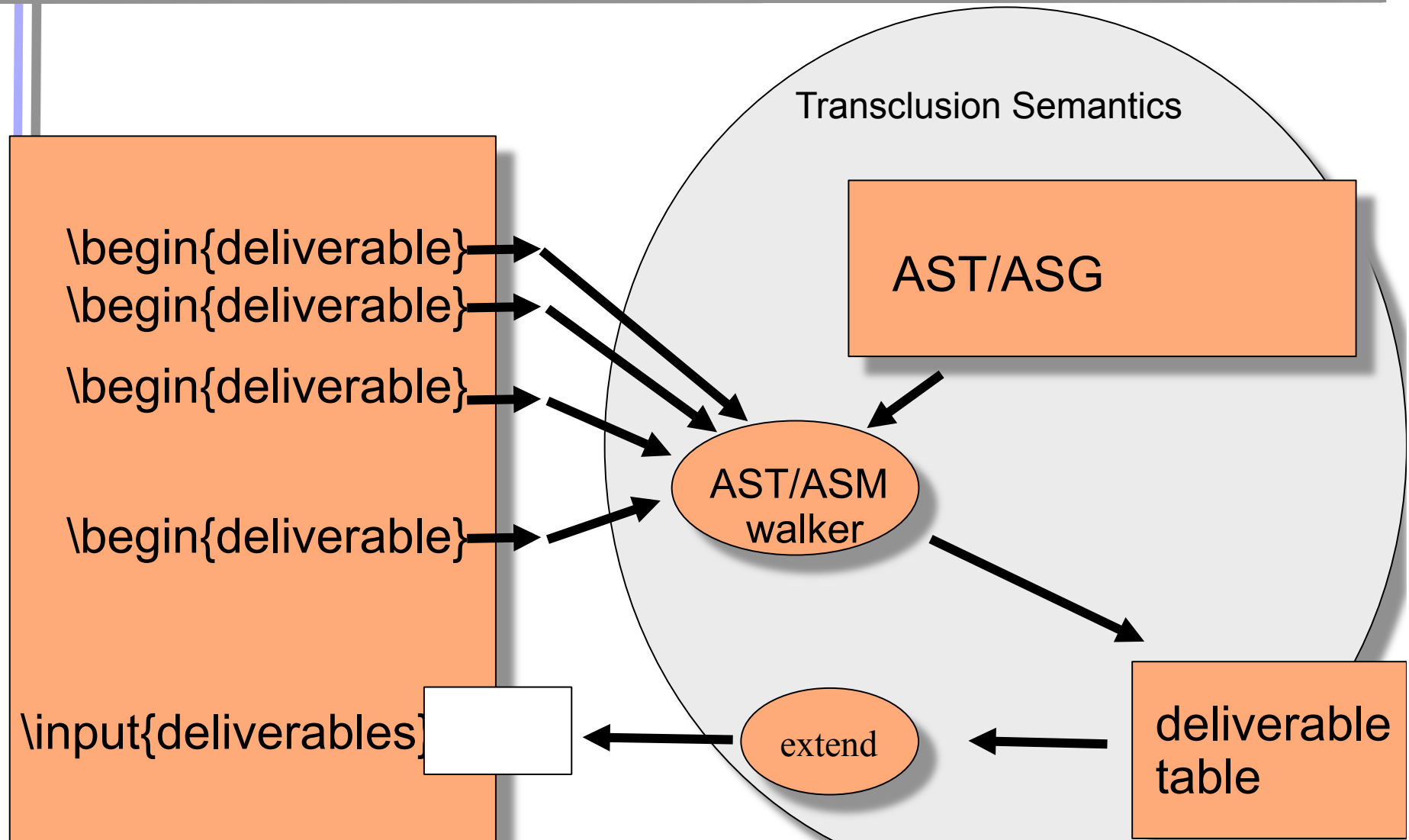
- ▶ Transclusion is *embedded sharing of document components in distributed edits*
  - Invented by Ted Nelson, the inventor of hypertext
- ▶ „hot update“ (incremental update)
  - Every change in a definition is immediately shared by all uses
  - Realized by reference and special edit protocols
  - Semantics is between call by name and call by value
- ▶ Nelson says: “That's what the computer is all about”

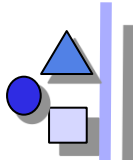
[http://xanadu.com.au/  
ted/](http://xanadu.com.au/ted/)





# Hot Update is Necessary in Active Documents





# ***Transconsistency of Active Documents (Immediate Update)***

- ▶ The architecture of an active document should obey *immediate (hot) update (transconsistency)*
  - Transclusion only deals with equality of hooks, but does not treat operations or modifications
  - Dependent components must be updated immediately
- ▶ For transconsistency, transclusion is a basis
  - Transconsistency requires a data-flow graph over operations in the document, i.e., a data-flow-based architecture
  - Whenever the input of a slice of the data-flow graph changes, recompute the result by reevaluating the slice
- ▶ Transconsistency requires invasive embedding
  - The component model of an active document must be graybox, otherwise embeddings are not possible

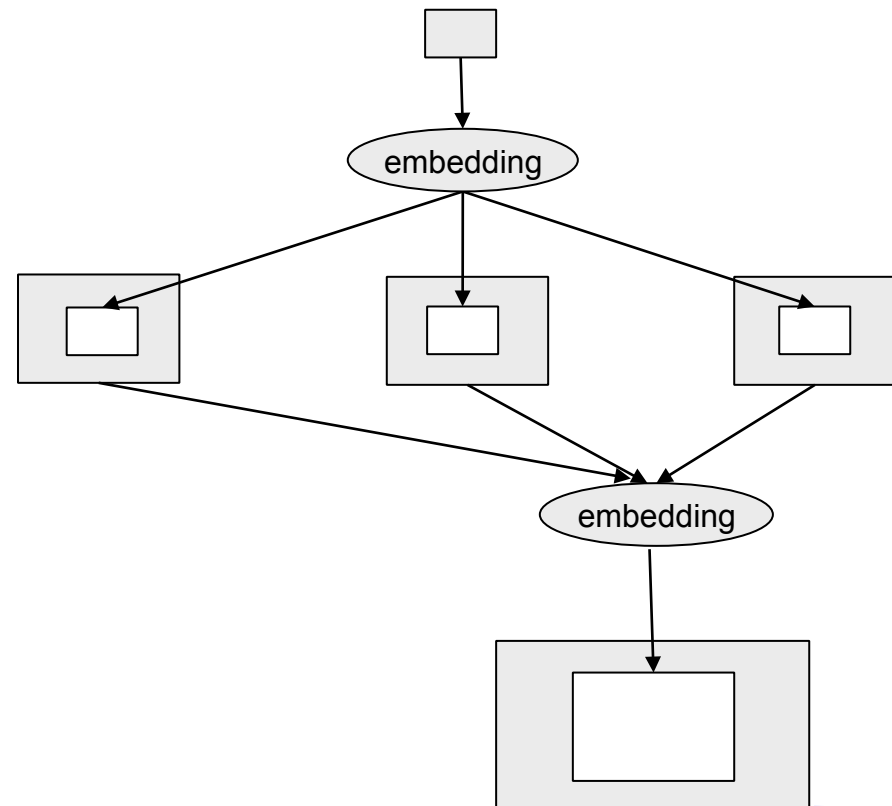
# *50.4.1. A Graph-Theoretic Definition of Transconsistency*





# Transclusion in Flow Graphs of Embedding Operations

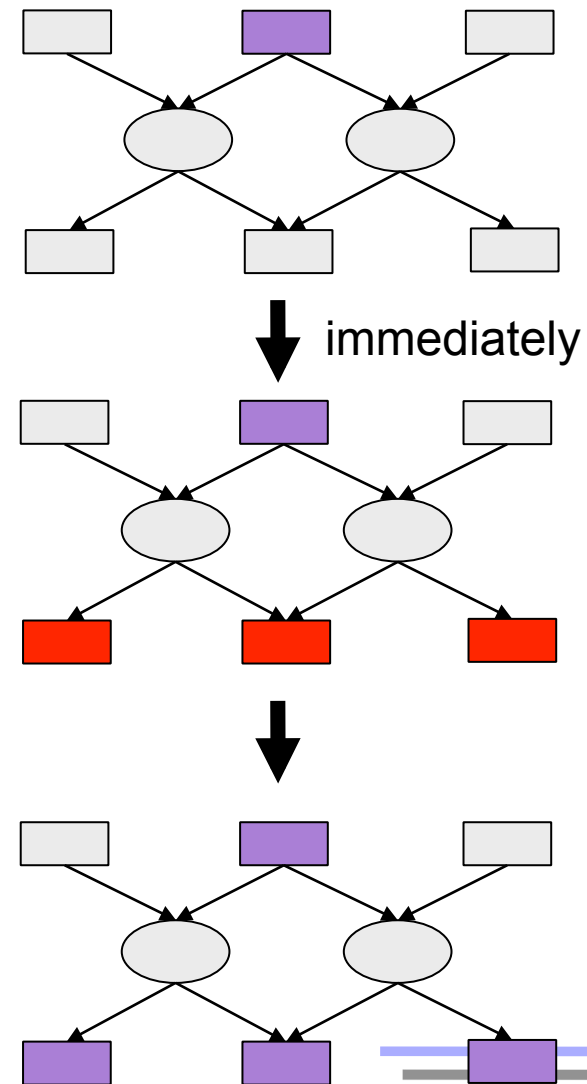
- ▶ Let  $D$  be a dataflow graph of *embedding operations*, a bipartite graph of EmbeddingOperations and Values.
- ▶  $D$  is called *transclusive*, if:
  - If an input value changes, all dependent values are declared inconsistent immediately, until they are reembedded





# Transconsistency in Data Flow Graphs

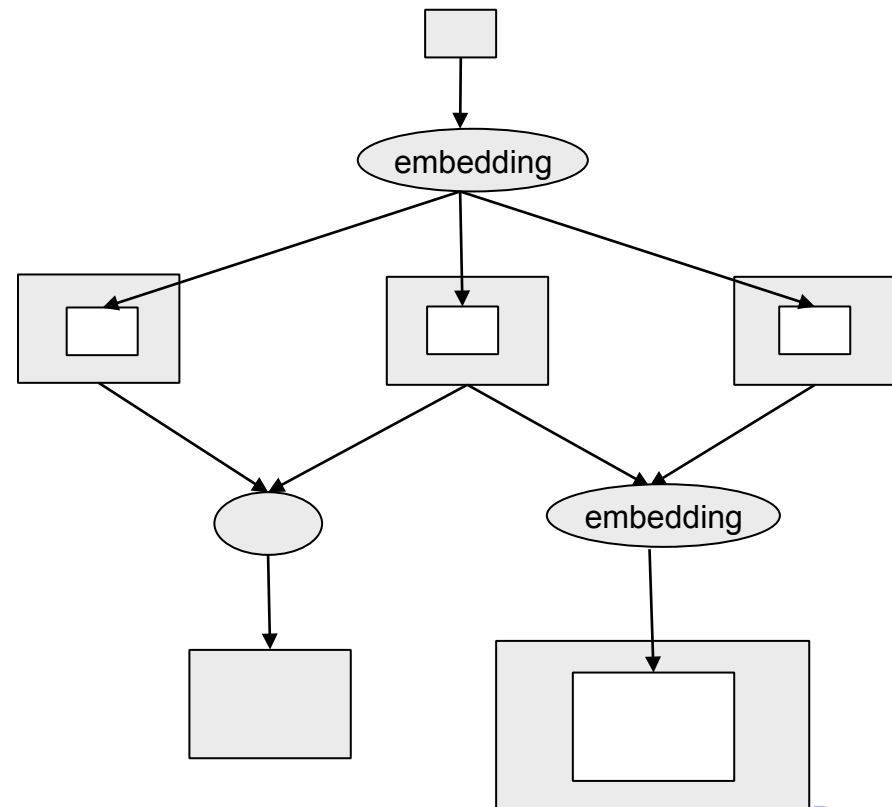
- ▶ Let  $D$  be a dataflow graph, a bipartite graph of Operations and Values.
- ▶  $D$  is called *transconsistent*, if the *hot update condition* is true:
  - If an input value changes, all dependent values are declared inconsistent immediately, until they are recomputed





# Transconsistency in Active Documents

- ▶ Let  $A$  be an active document with an underlying dataflow graph  $D$  for document parts.
- ▶ Then,  $D$  is called the *architecture* of  $A$ .
- ▶  $A$  is called *transconsistent*, if  $D$  is transconsistent



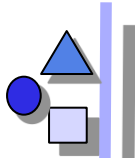


# *Transclusion and Transconsistency*

Transclusion  
=  
Invasive Embedding +  
Incrementality (hot update)

Transconsistency  
=  
Transclusion +  
Data flow graph

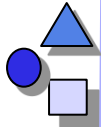
Transconsistent Architecture  
=  
Transconsistency + Architecture



# *Transconsistency Goes Beyond Transclusion*

- ▶ Transclusion only treats embedding and hot update
- ▶ It does not treat
  - Operations beyond embedding
  - Data flow graphs of these operations
  - Components





## *Examples for Transconsistency*

**Spreadsheets:** A spreadsheet relies on a dataflow graph (pipe-and-filter)

- It is a set of slices, i.e., a set of expressions, or scriptlets
  - A scriptlet describes a dataflow graph of operations
  - Every slice is independent, i.e., can be recomputed independently
- ▶ Spreadsheets are simple active document with transconsistency, i.e., immediate update
- ▶ Spreadsheets do not have architecture
- No component model nor composition interface

**Web Documents:** Servlet-based documents rely on re-expansion if users change forms or templates

- ▶ The servlets span up a data flow graph
- Templates and form inputs are the inputs
  - Result pages the output
- ▶ The regeneration is an implementation of transconsistency

## 50.4.2 Transconsistent Architectures

Uniform Composition of Active Documents with Staging  
and Transconsistency





# Transconsistent Documents

- ▶ *Transconsistent documents* are active documents with explicit *transconsistent architecture*
  - Like spreadsheets, but with explicit architecture
  - Based on a
    - Dataflow graph
    - Graybox component model (invasive embedding)
    - Incrementality (Hot update)
- ▶ Purpose of Transconsistent Architectures
  - Transconsistency copes interactive editing
  - This is fundamentally different to the so-far batch-oriented style of software construction, software build, and software execution
  - Transconsistency is needed in software editing, too

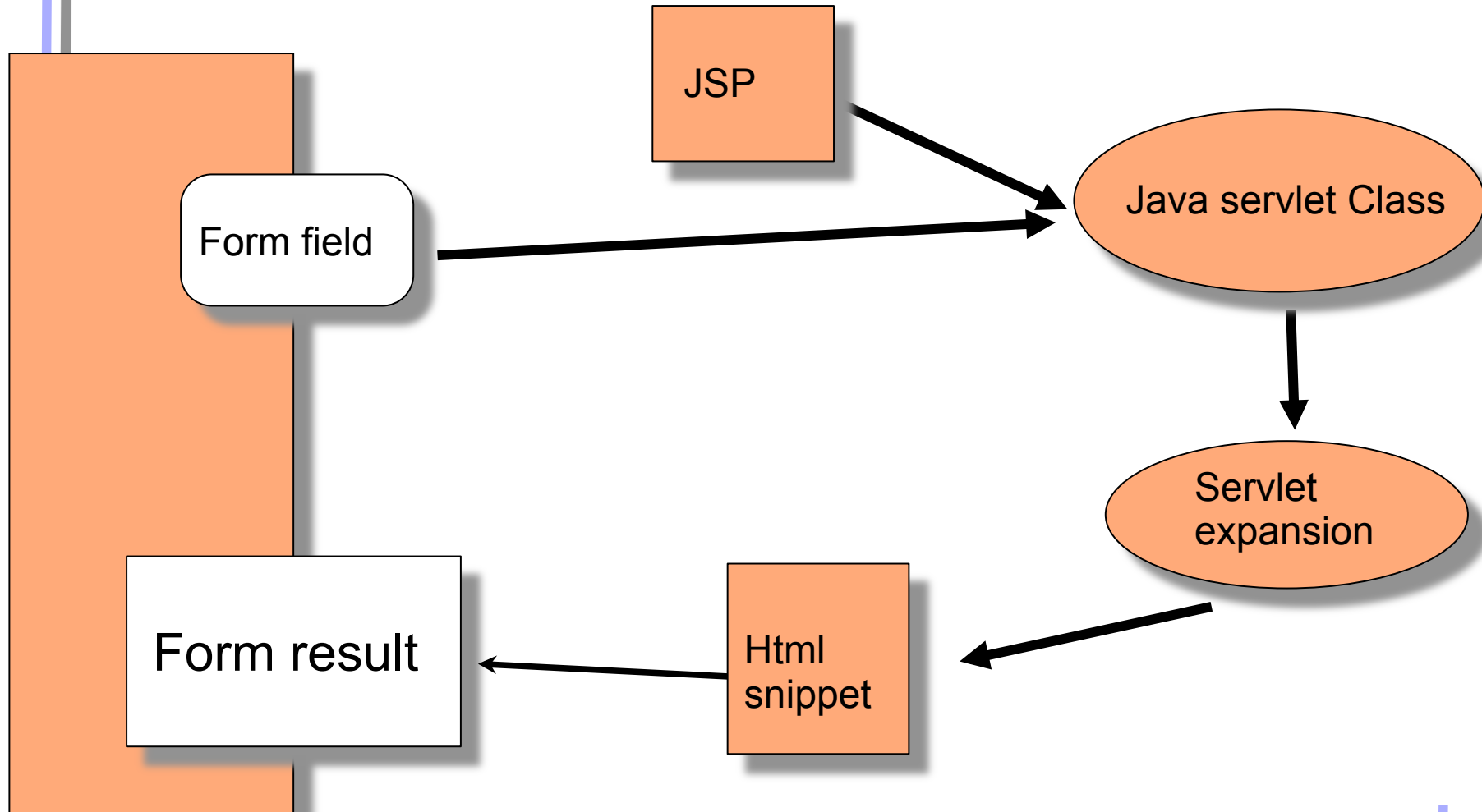
# 50.5 Transconsistent Architectural Styles

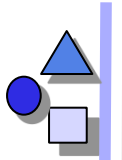
Composition of Active Documents with Staging and Transconsistency



# Web Form Processing with JSP

- ▶ Should be transconsistent...





# Spreadsheet-documents and Pipe-And-Filter Architectures

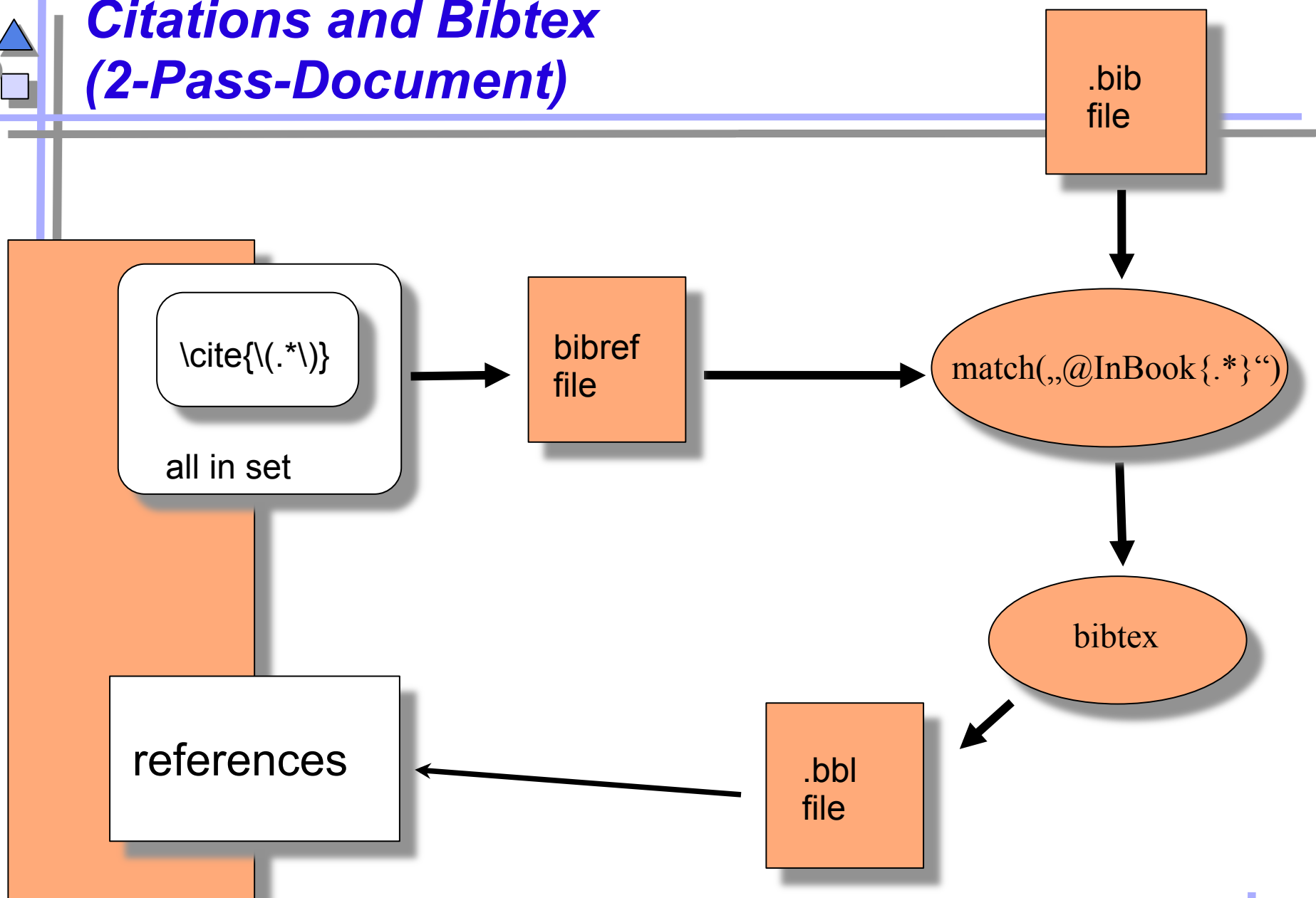
- ▶ **Spreadsheet-Documents:** A **spreadsheet-document** is a an active document with a pipe-and-filter architecture
  - Resembles spreadsheets
  - The question is how often the filter architecture is evaluated for transconsistency
  - A web form (e.g., JSP) is a *distributed spreadsheet-document*



## 2-Pass Transconsistent Documents

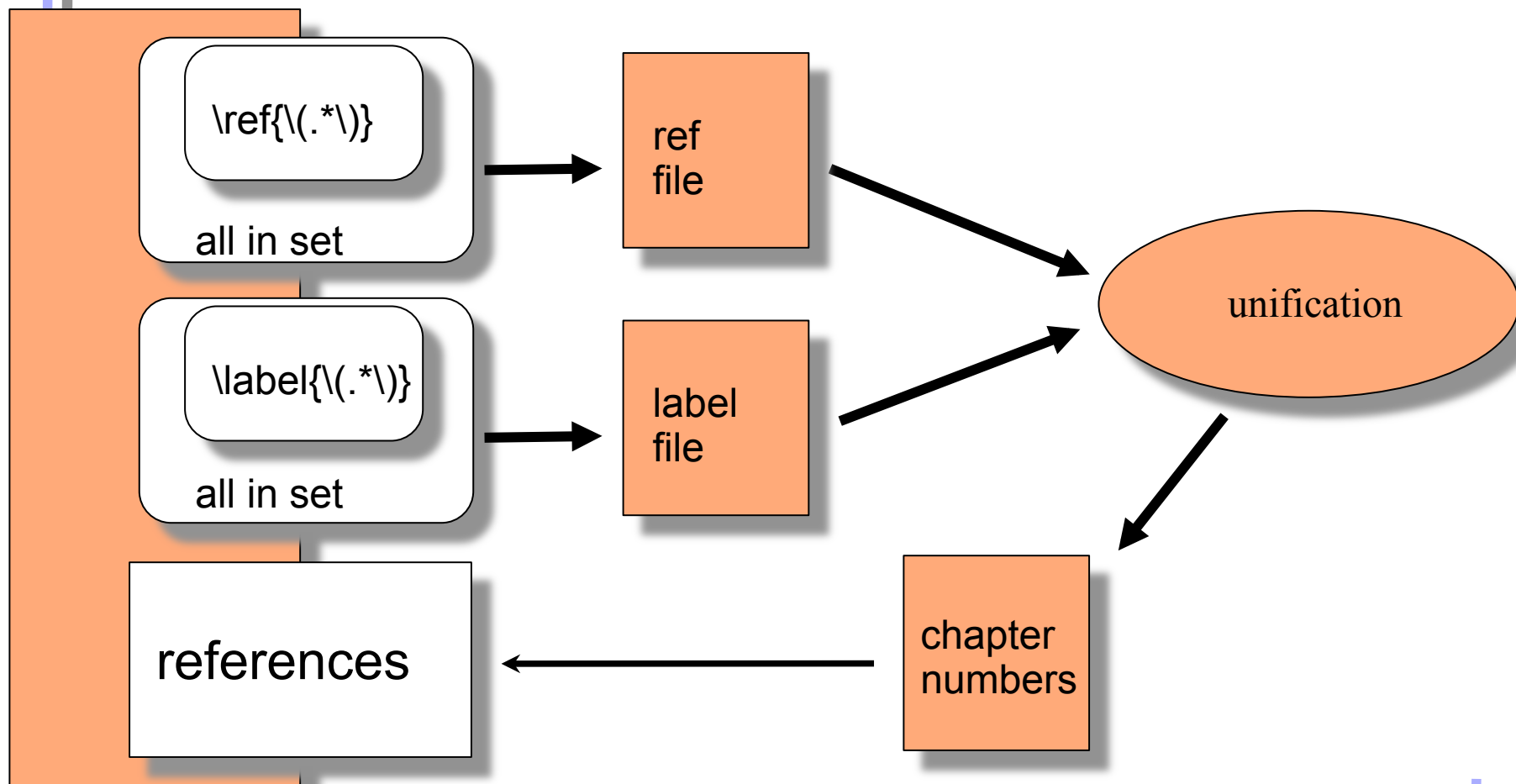
- ▶ Transconsistent documents underly a dependency graph for their update
  - This dependency graph must be acyclic
- ▶ Evaluation classes for transconsistent documents
  - 1-pass problems along the document (all definitions before uses)
  - 2-pass (backpatch problems) along the document
  - Statically orderable along the dependencies (similar to wavefront or OAG)
  - Form processing

# Citations and Bibtex (2-Pass-Document)

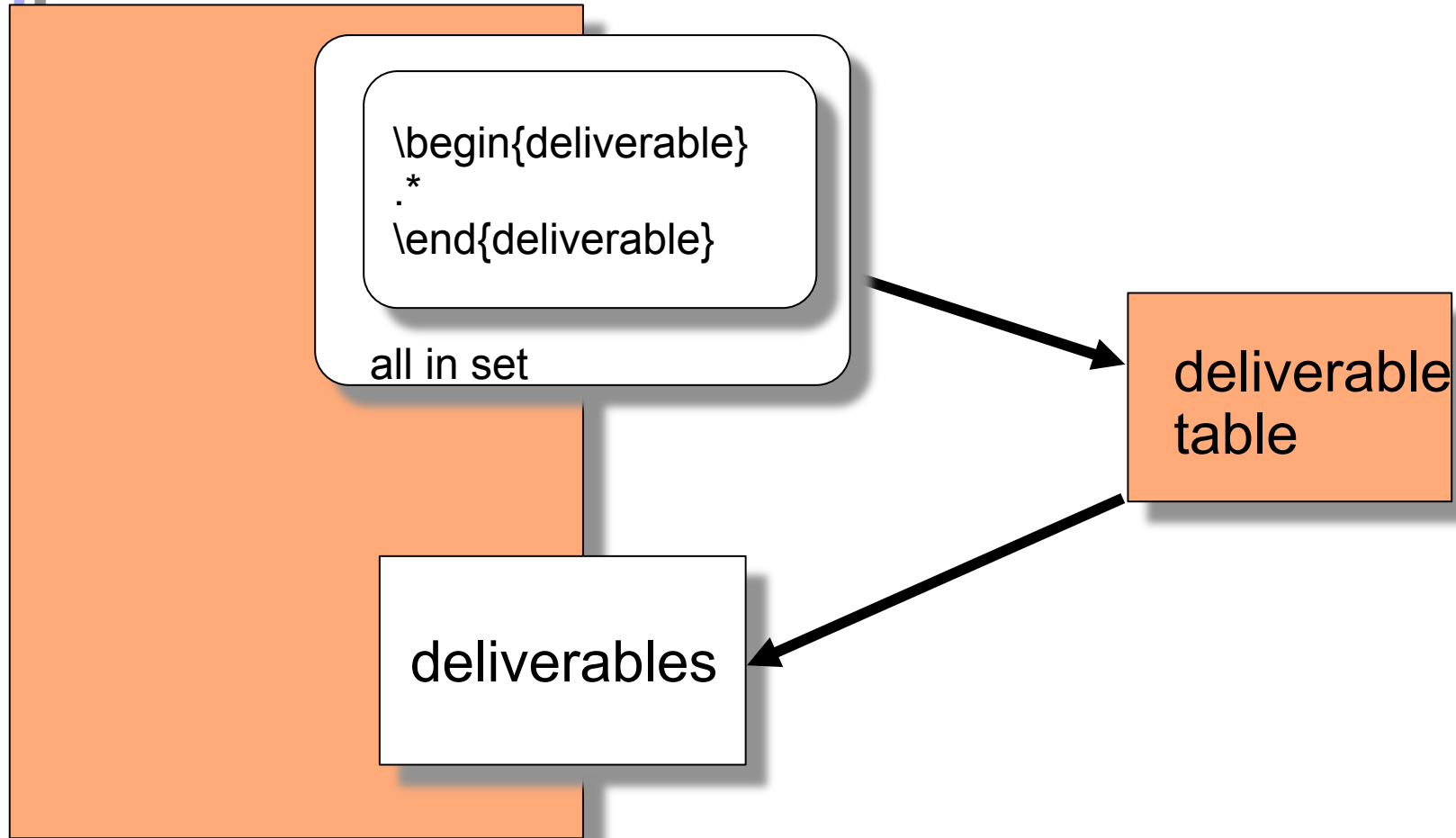




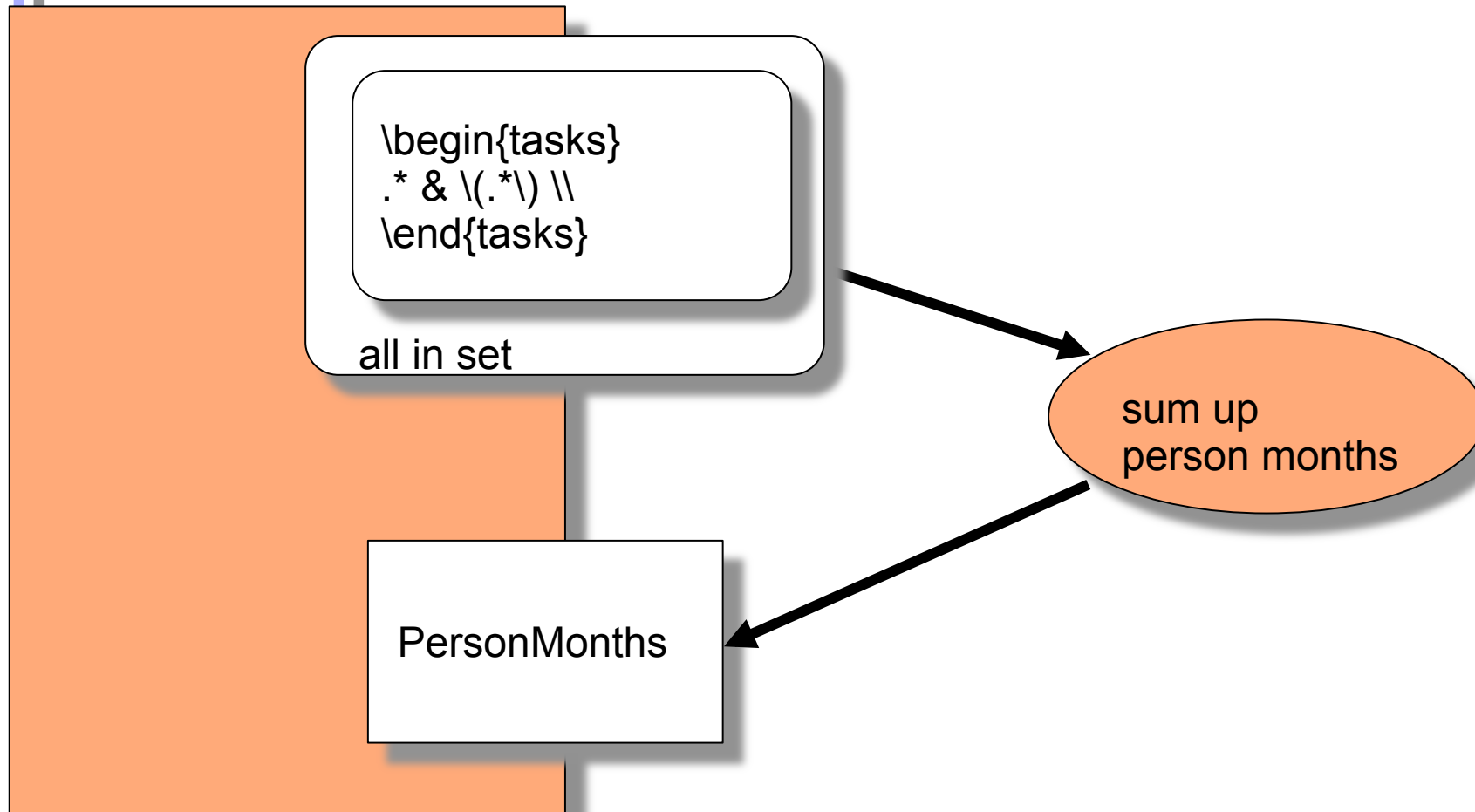
# References (2-Pass-Document)

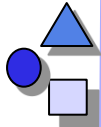


# Central Tables (2-Pass-Document)



# Person Cost Calculation Central Tables (2-Pass-Document)

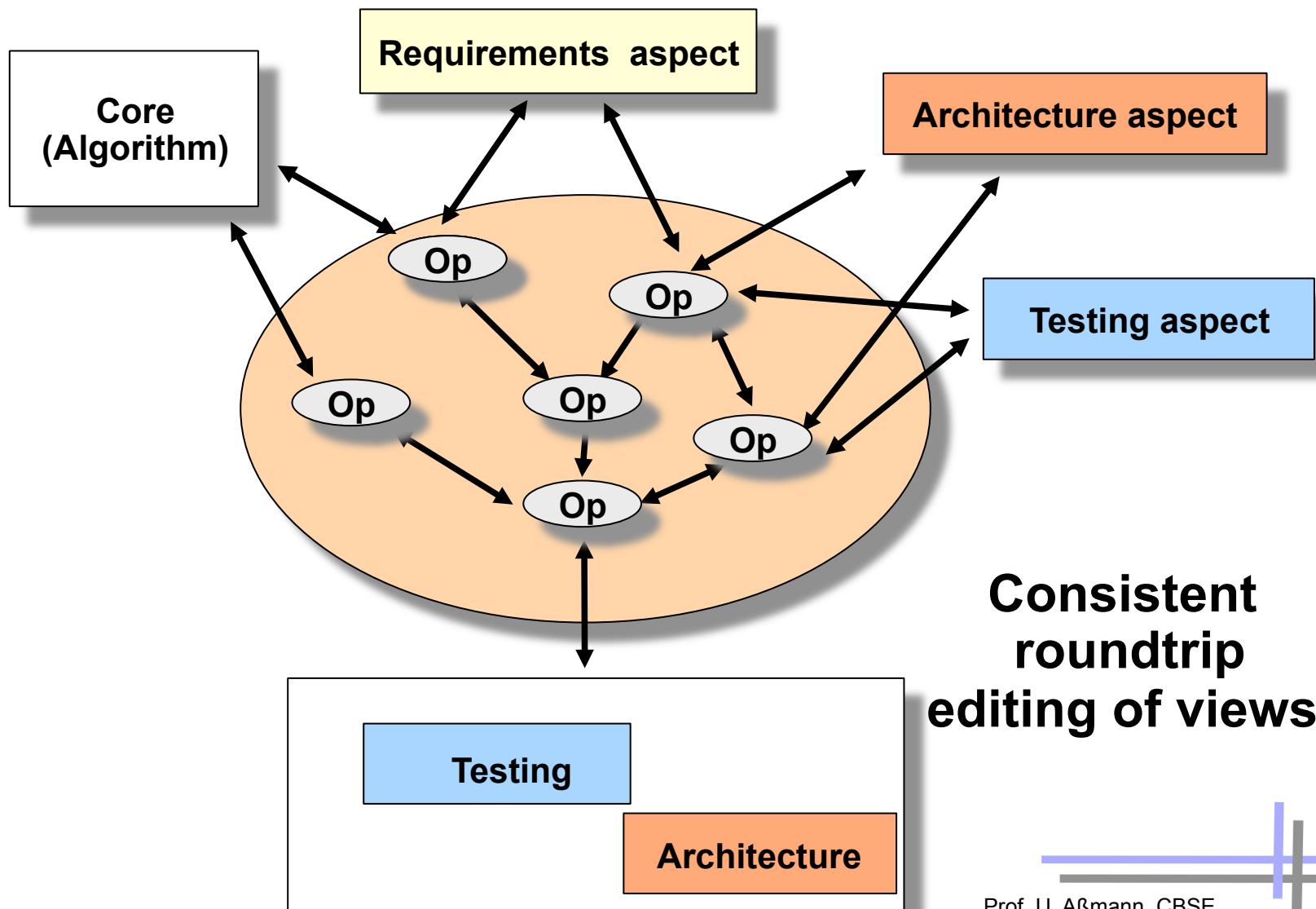




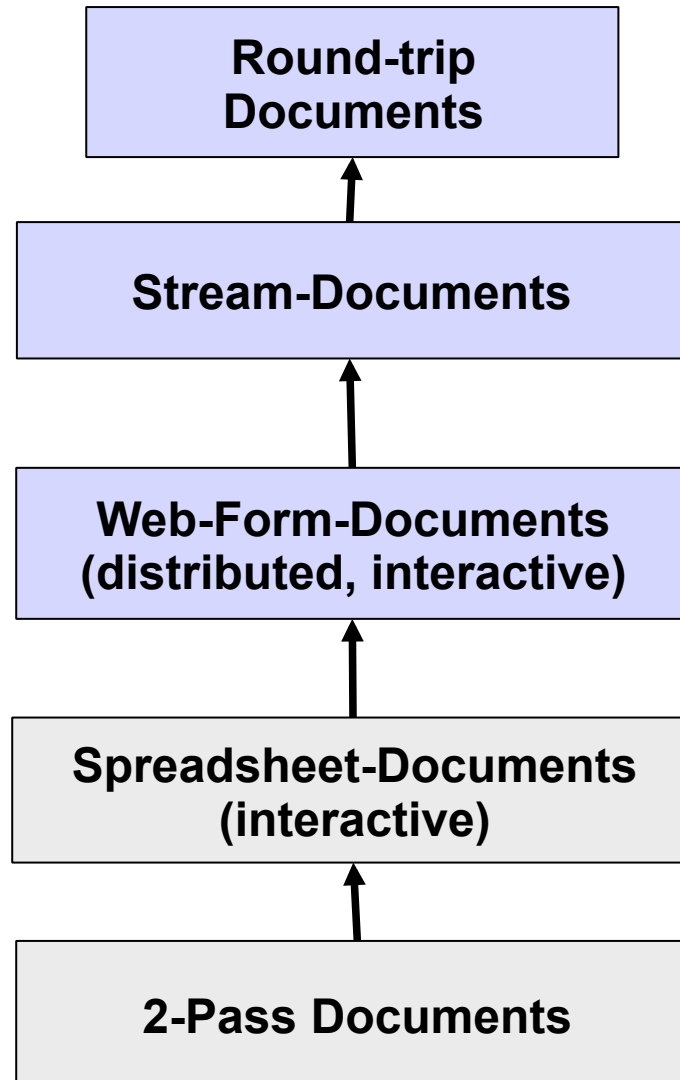
# Stream-Documents (Spreadsheet Documents with Pipe Ports)

- ▶ Instead of being a closed document, spreadsheet-documents can be open in the sense that they take in data streams over stream ports
  - START submission phase
  - START reviewing phase
- ▶ Such a change corresponds to a document extension, but works via communication channels/connectors
- ▶ User changes and sends via ports are the similar effects
  - User change: change component values
  - Send via ports: change from external world

# Transconsistent Documents: Roundtrip Engineering Documents

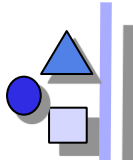


# *Transconsistent Architectural Styles for Active Documents*



# *Benefit of Transconsistent Architectures For Active Documents*





# Advantages of Transconsistent Active Documents

- ▶ **Beyond standard document models (such as OLE):**
  - Explicit distinction between architecture and content
  - Better reuse
  - Can be combined with staged composition for Web engineering
- ▶ **Beyond spreadsheets:**
  - Full table and sheet extension, not only value transconsistency (table extension hot update)
- ▶ **Beyond template-based documents:**
  - Decentralized definition of databases/relations
- ▶ **Benefits for Web Engineering**
  - Transconsistent active documents provide a first unified model for web- and document engineering
  - Beyond simple approaches such as JSP, ASP
  - Improvement of quality:
    - Documentative due to architecture
    - Gets rid of the spaghetti code in web engineering





## Summary

- ▶ For engineering of active documents, explicit distinction of architectures is important
  - Invasive embedding is required
  - Data flow graphs are required
- ▶ Transconsistent architectures are an important architectural styles for active documents
  - Rely on an extended concept of transclusion
  - Cope with streams of interactive input



*The End*