I Object

or ... How I Learned to Stop Worrying and Love OOP

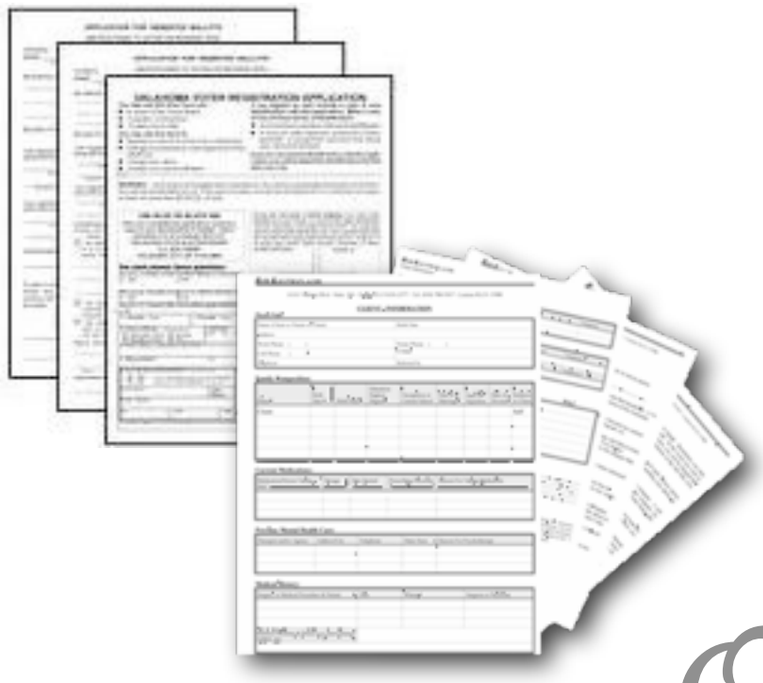
Oscar Nierstrasz
scg.unibe.ch

ECOOP 2013

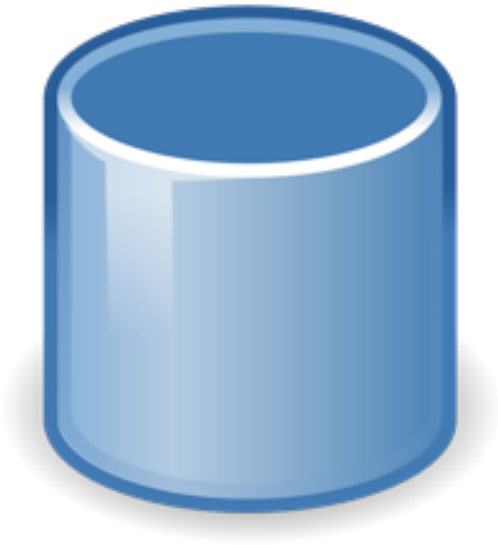
1. Office Objects

How to build the “electronic office”?

TLA



OFS



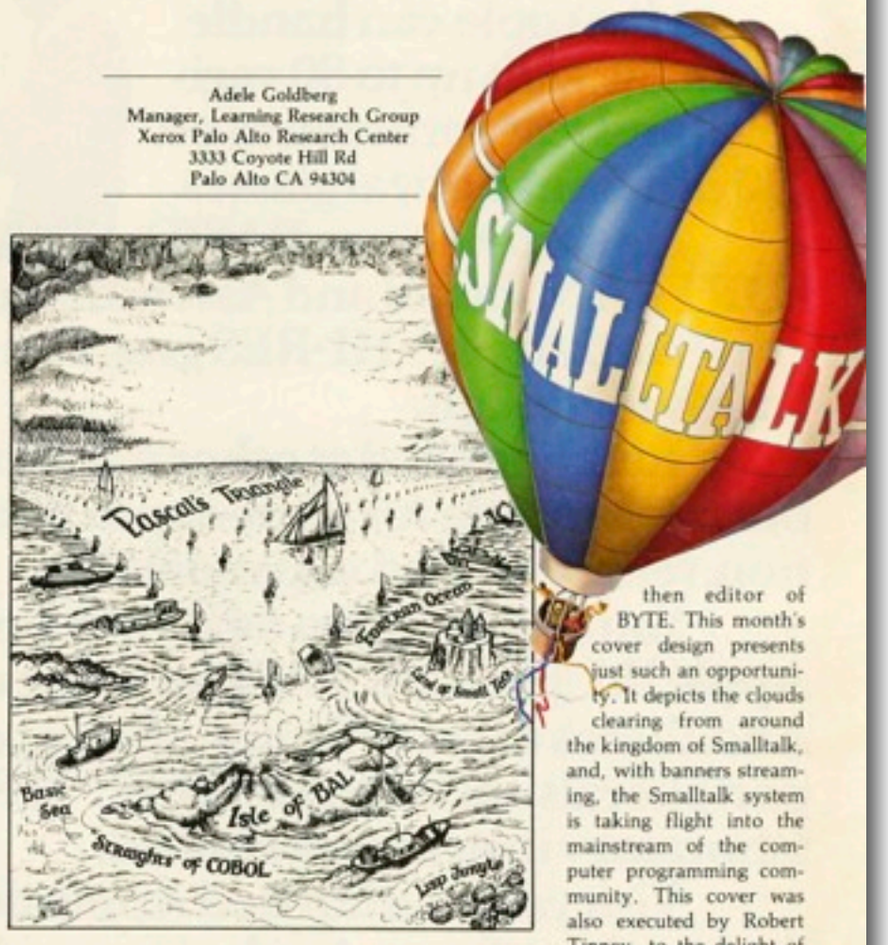
MRS

Tsichritzis, et al. ***A System for Managing Structured Messages.*** *IEEE Transactions on Communications, 1982.*

Uh, where are the objects?

Introducing the Smalltalk-80 System

Adele Goldberg
Manager, Learning Research Group
Xerox Palo Alto Research Center
3333 Coyote Hill Rd
Palo Alto CA 94304



It is rare when one can indulge in one's prejudices with relative impunity, poking a bit of good humored fun to make a point.

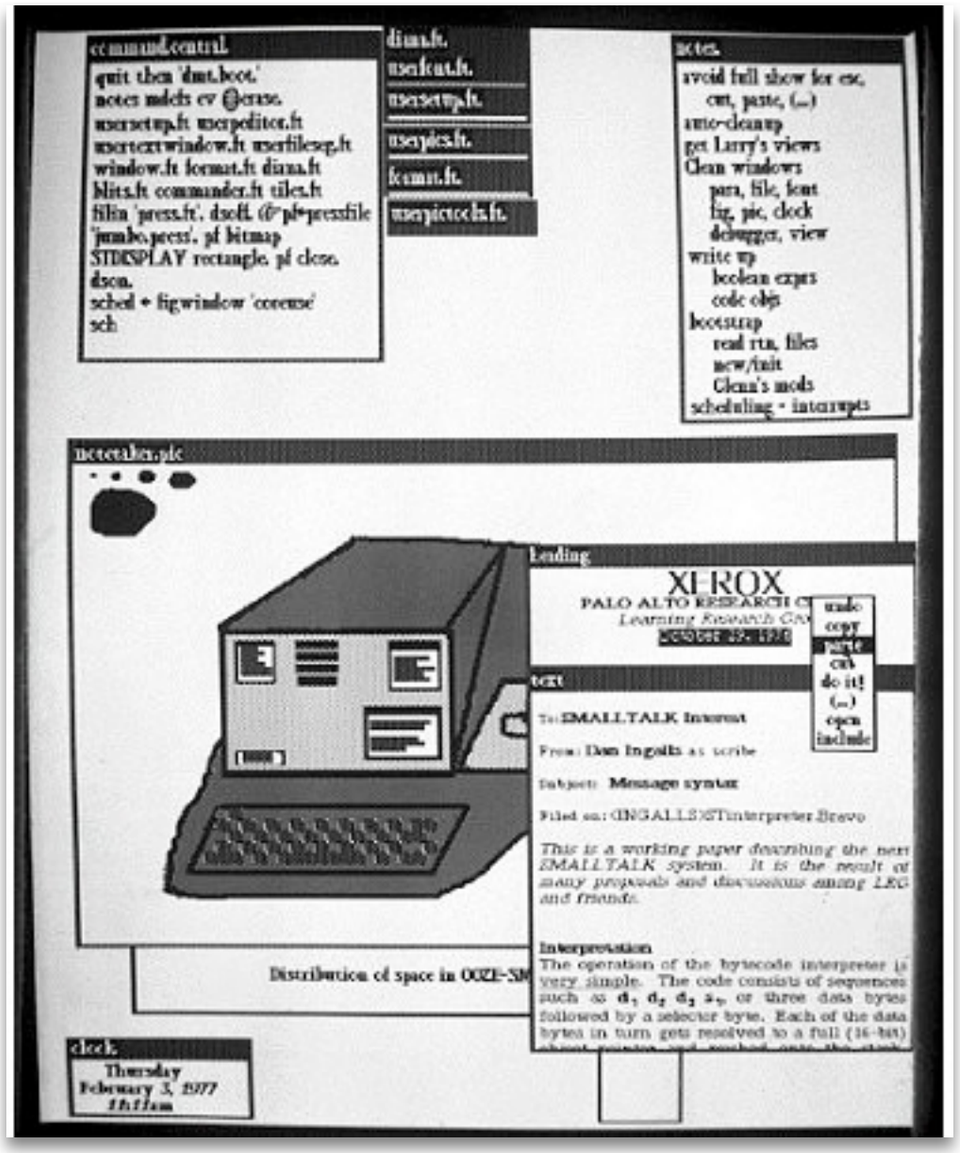
With this statement, Carl Helmers opened his remarks in the "About the Cover" section of the August 1978 issue of BYTE. The issue was a special on the language Pascal, so Helmers took the opportunity to present Pascal's triangle as drawn by artist Robert Tinney. The primary allegory of the cover was the inversion of the Bermuda Triangle myth to show smooth waters within the area labeled "Pascal's Triangle." In explaining the allegory, Helmers guided the traveler through the FORTRAN Ocean, the BASIC Sea, around the Isle of BAL, and up to the Land of Smalltalk.

Traveling upward (in the picture) through heavy seas we come to the pinnacle, a snow white island rising like an ivory tower out of the surrounding shark infested waters. Here we find the fantastic kingdom of Smalltalk, where great and magical things happen. But alas . . . the craggy aloofness of the kingdom of Smalltalk keeps it out of the mainstream of things.

It is rare when one can indulge in one's fantasies to respond to so pointed a remark as that provided by the then editor of BYTE. This month's cover design presents just such an opportunity. It depicts the clouds clearing from around the kingdom of Smalltalk, and, with banners streaming, the Smalltalk system is taking flight into the mainstream of the computer programming community. This cover was also executed by Robert Tinney, to the delight of the Learning Research Group (LRG) of the Xerox Palo Alto Research Center. LRG is the group that has designed, implemented, and evaluated several generations of Smalltalk over the past ten years.

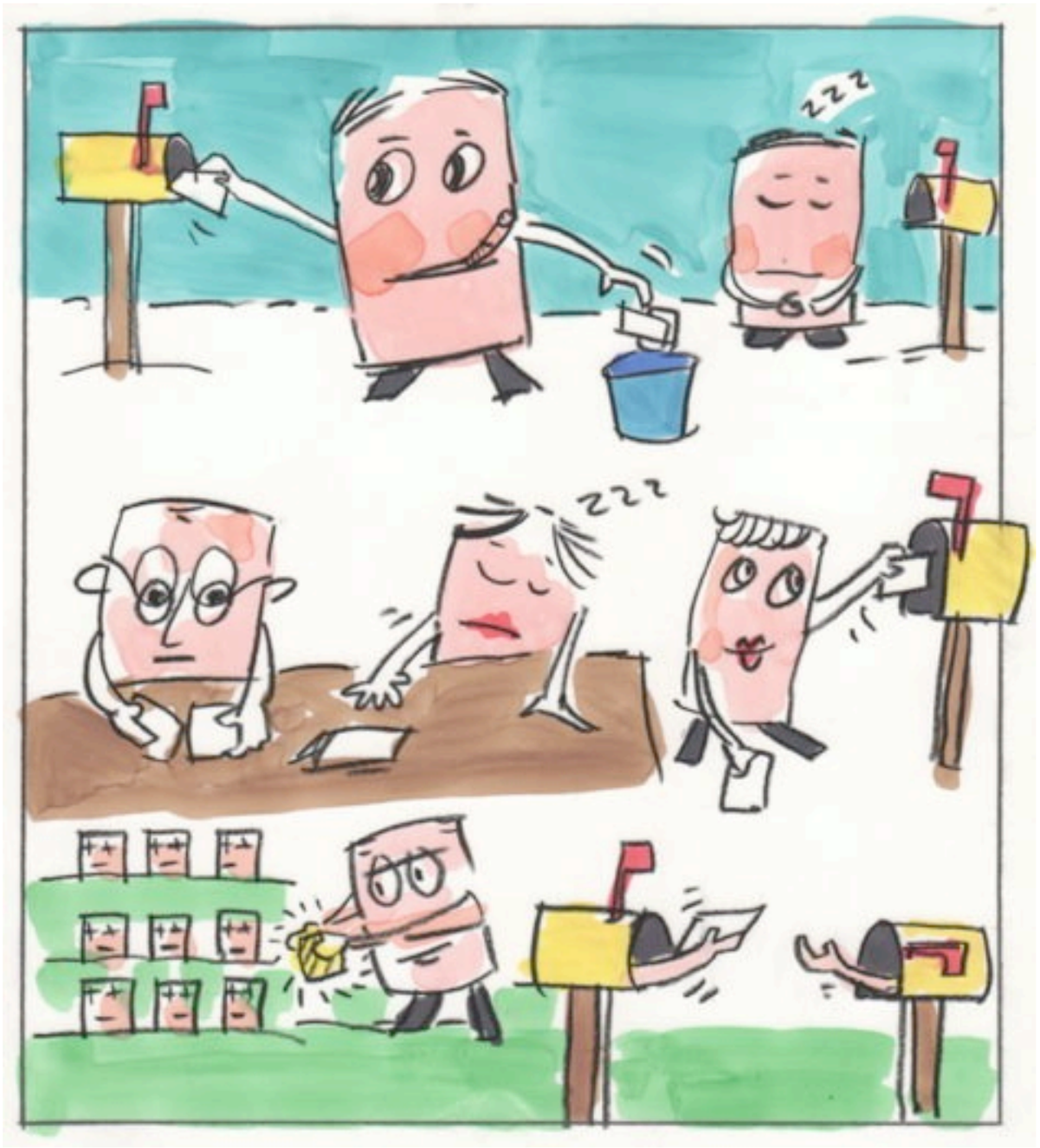
The balloon on the cover symbolizes the Smalltalk-80 system that is being released this year for more general access. The release is in the form of publications and a file containing the Smalltalk-80 programming system. Twelve articles describing the system appear in this issue of BYTE. Through such publication, LRG's research will become generally accessible, dispelling the clouds.

Smalltalk is the name LRG assigned to the software



Uh, what's a Dorado?

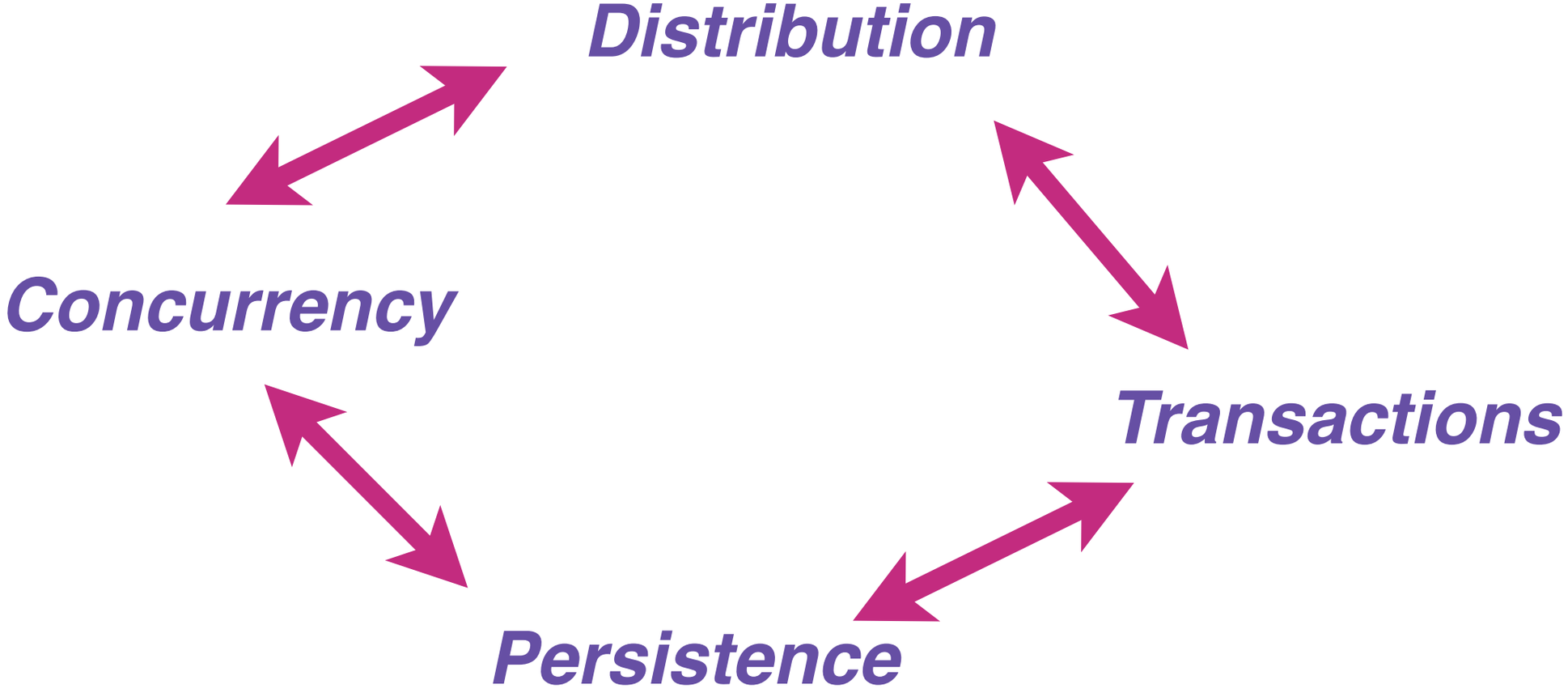
Oz: Objects with Rules




```
customer : office {
        name, owner : string ;
        set_name (n) {
                ~ : office ;
                n : string ;
                ~.owner = owner ;
                name := n ;
        }
}
```

Nierstrasz and Tsichritzis. ***An Object-Oriented Environment for OIS Applications****. VLDB 1985*


OOP
All Purpose


Distribution
Concurrency
Transactions
Persistence

What I learned ...

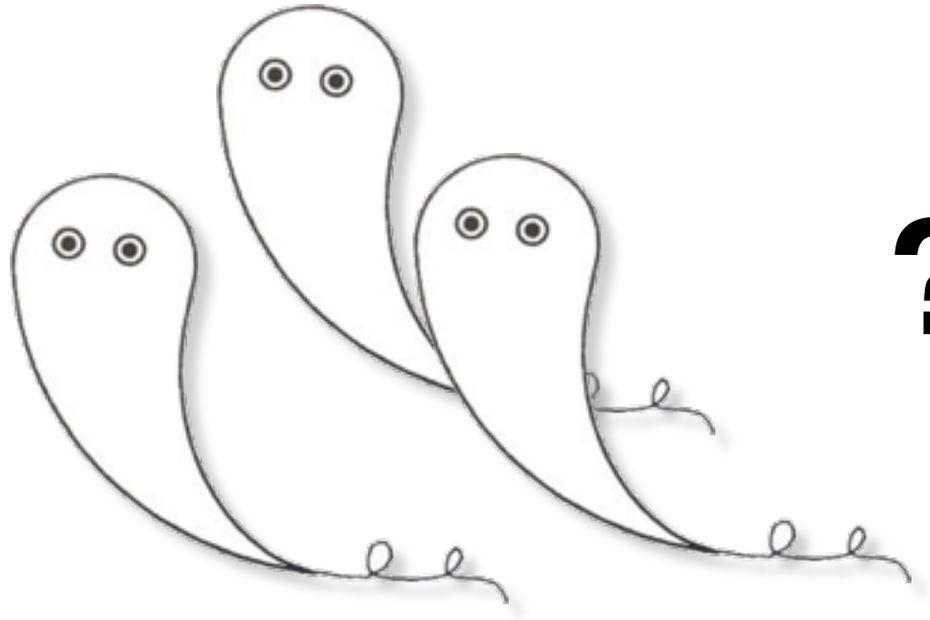
Objects are

complicated

2. Active Objects

How to meld objects and concurrency?

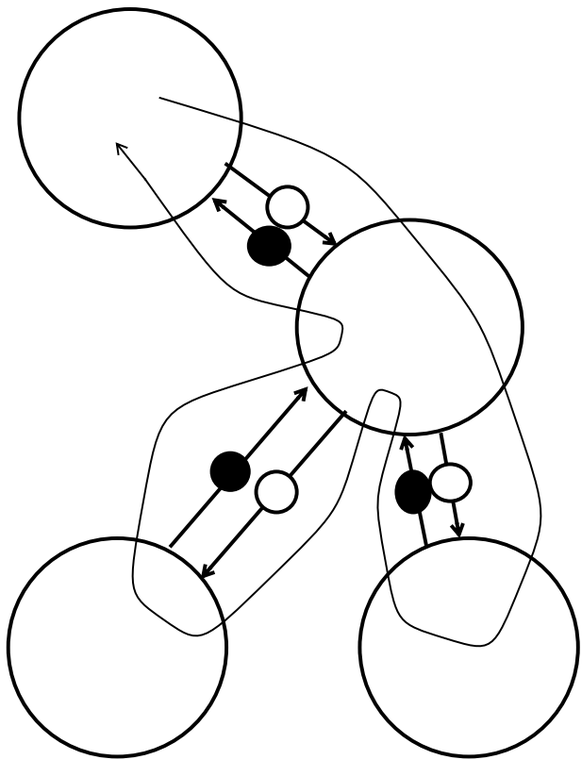
=



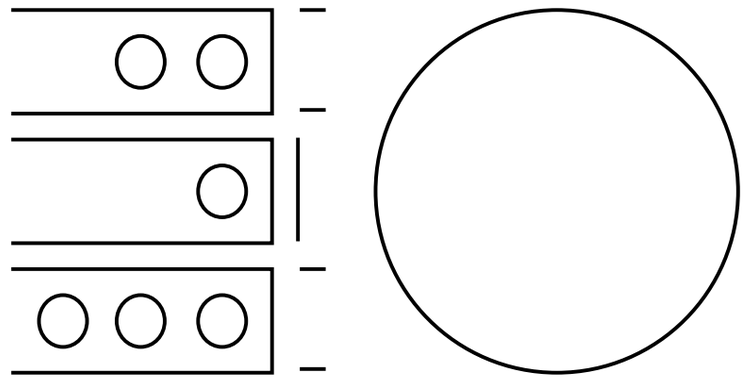
?

Tsichritzis, et al. ***KNOs: KNowledge Acquisition, Dissemination and Manipulation Objects****. ACM TOOIS 1987*

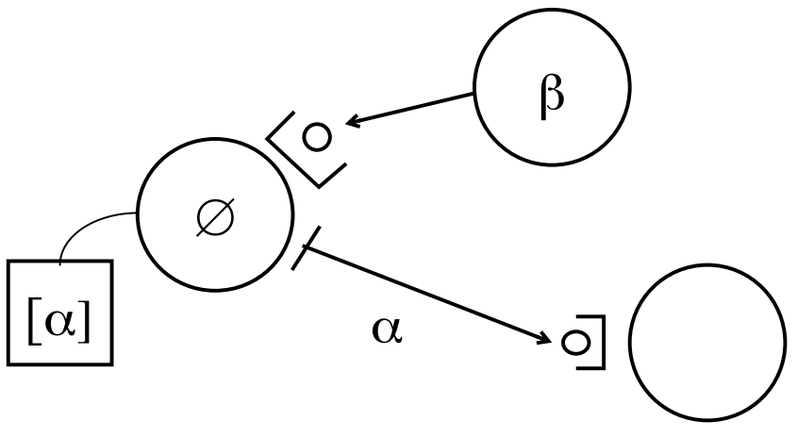
Hybrid



Domains & Activities



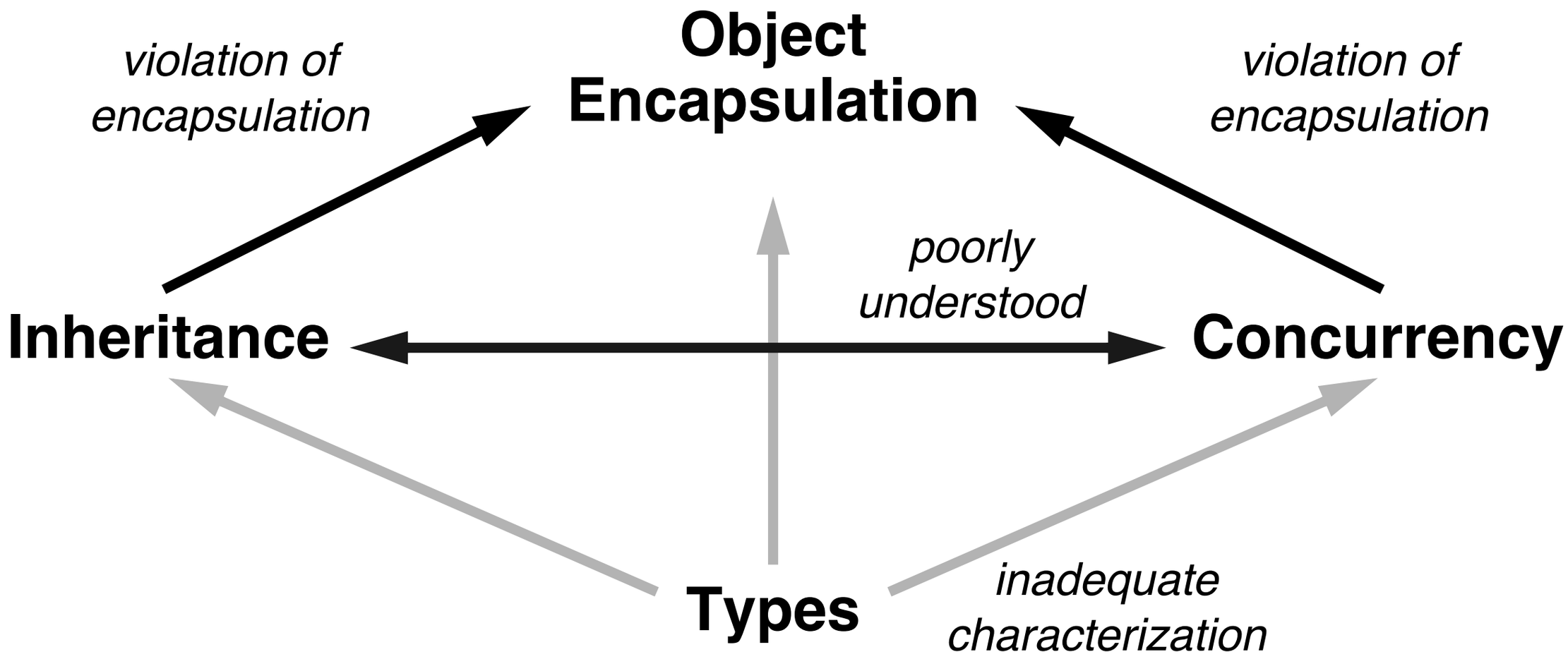
Delay Queues



Delegation

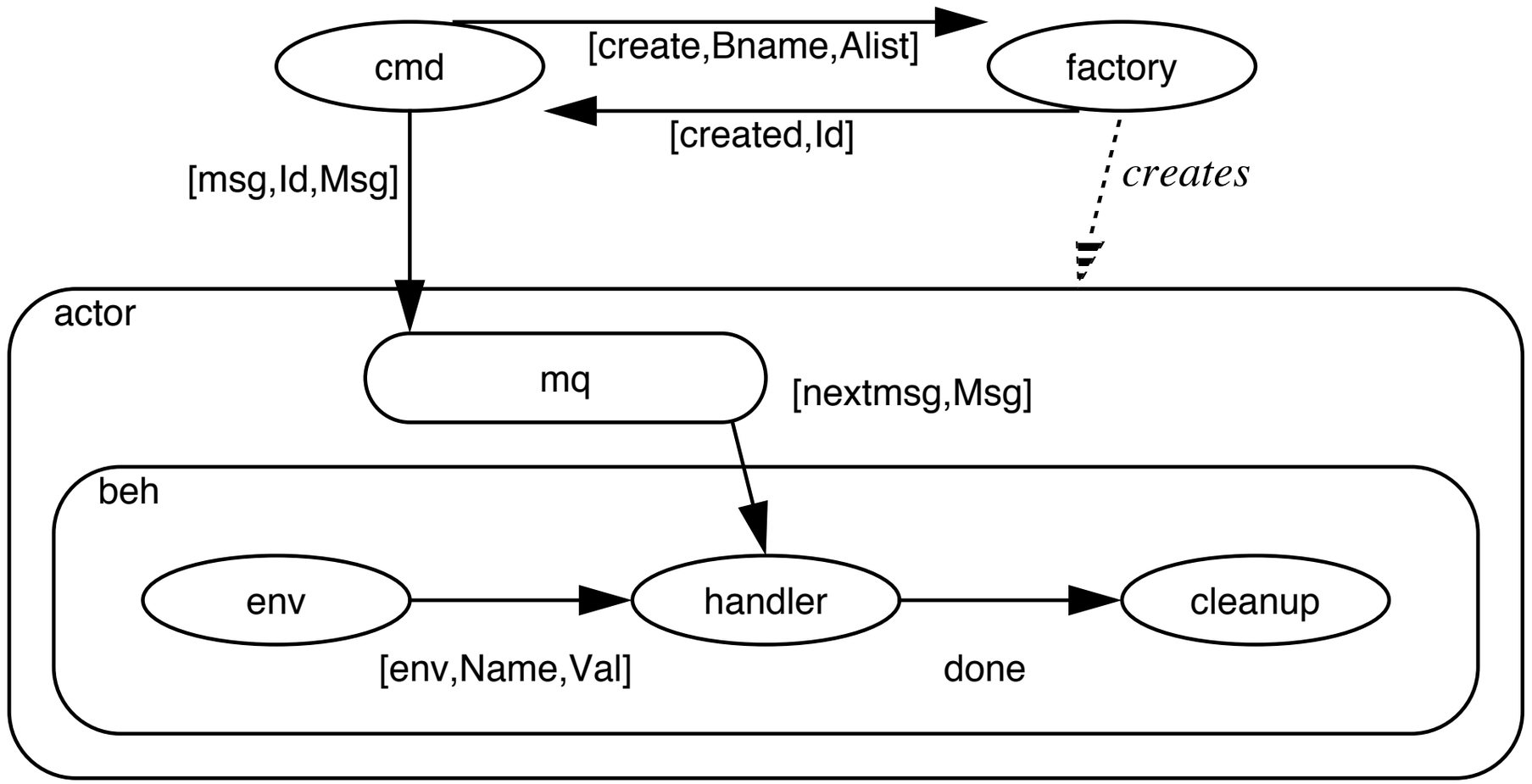
Nierstrasz. ***Active Objects in Hybrid****. OOPSLA 1987*

Semantics?



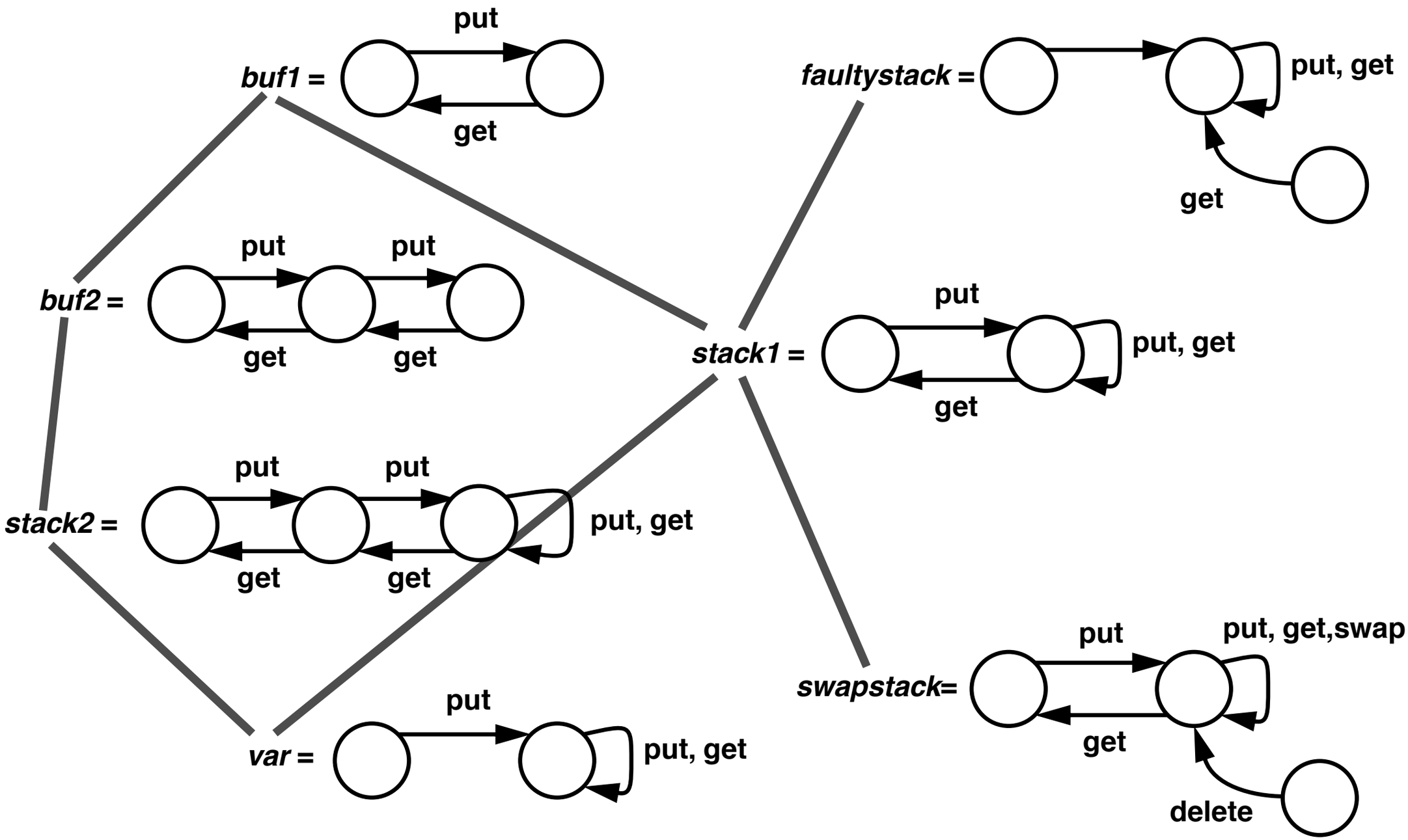
Nierstrasz. ***Composing Active Objects — The Next 700 Concurrent Object-Oriented Languages****. Research Directions in Concurrent Object-Oriented Programming, 1993*

Object calculi



Nierstrasz and Papathomas. ***Viewing Objects as Patterns of Communicating Agents****. OOPSLA/ECOOP 1990*

Regular Types



Nierstrasz. ***Regular Types for Active Objects****. OOPSLA 1993*

What I learned ...

Active objects

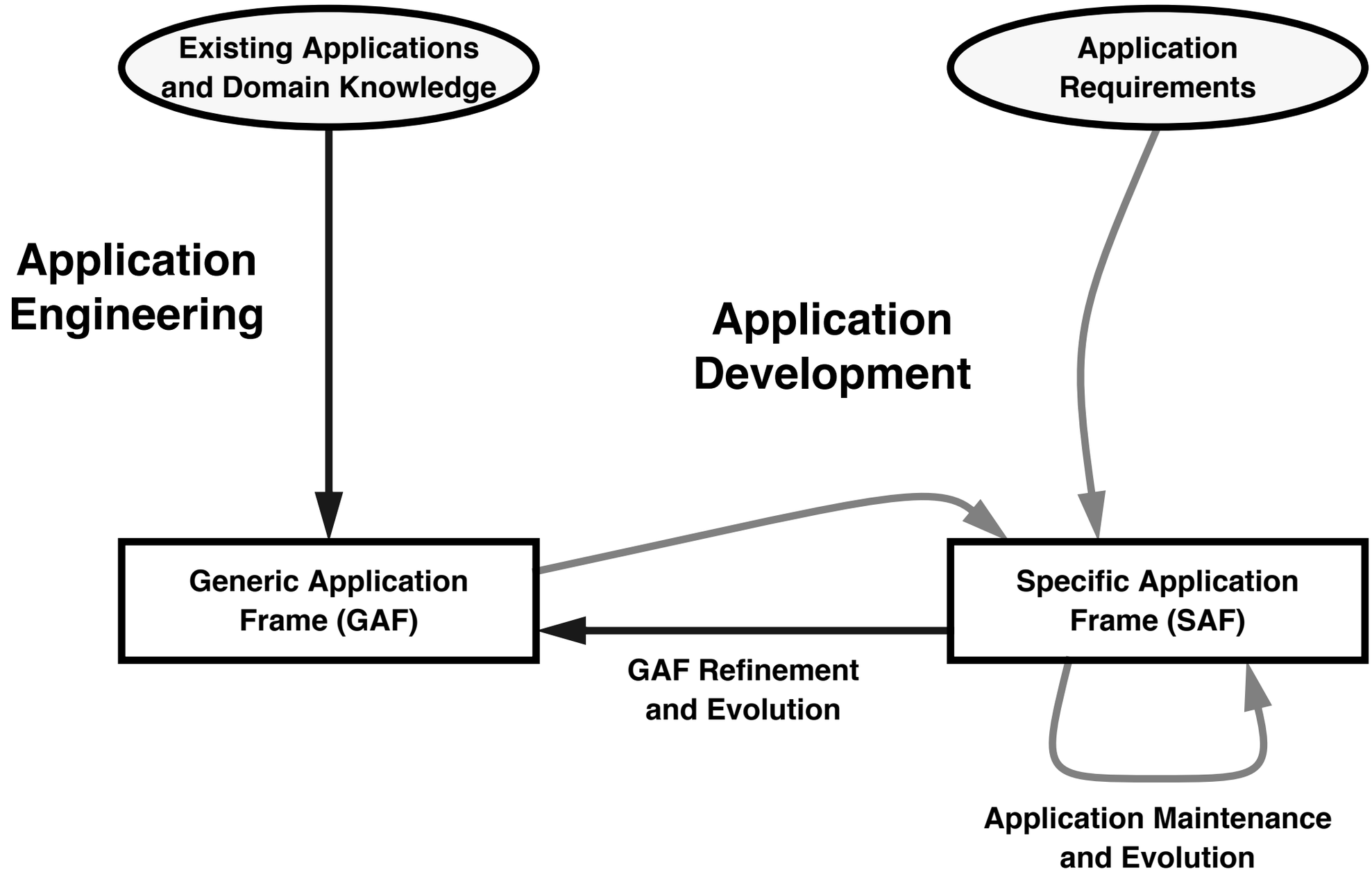
are very

complicated

3. Components

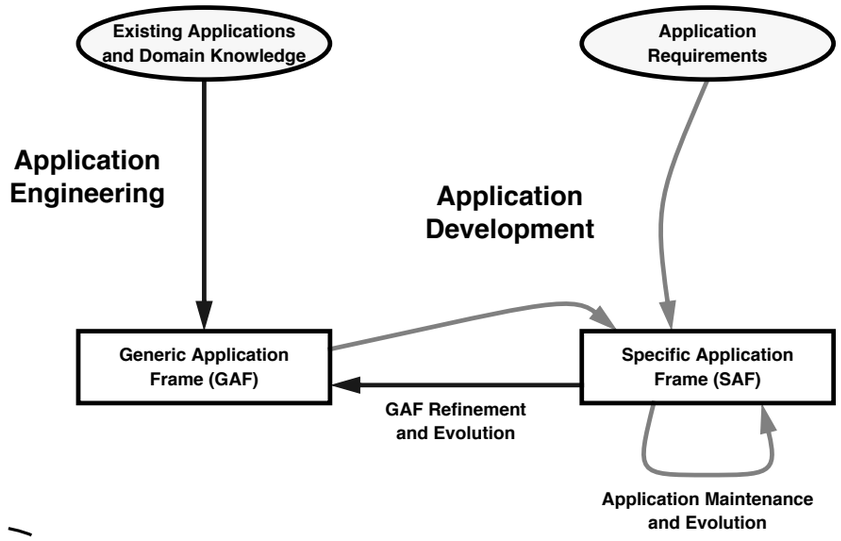
How to compose applications from “reusable” parts?

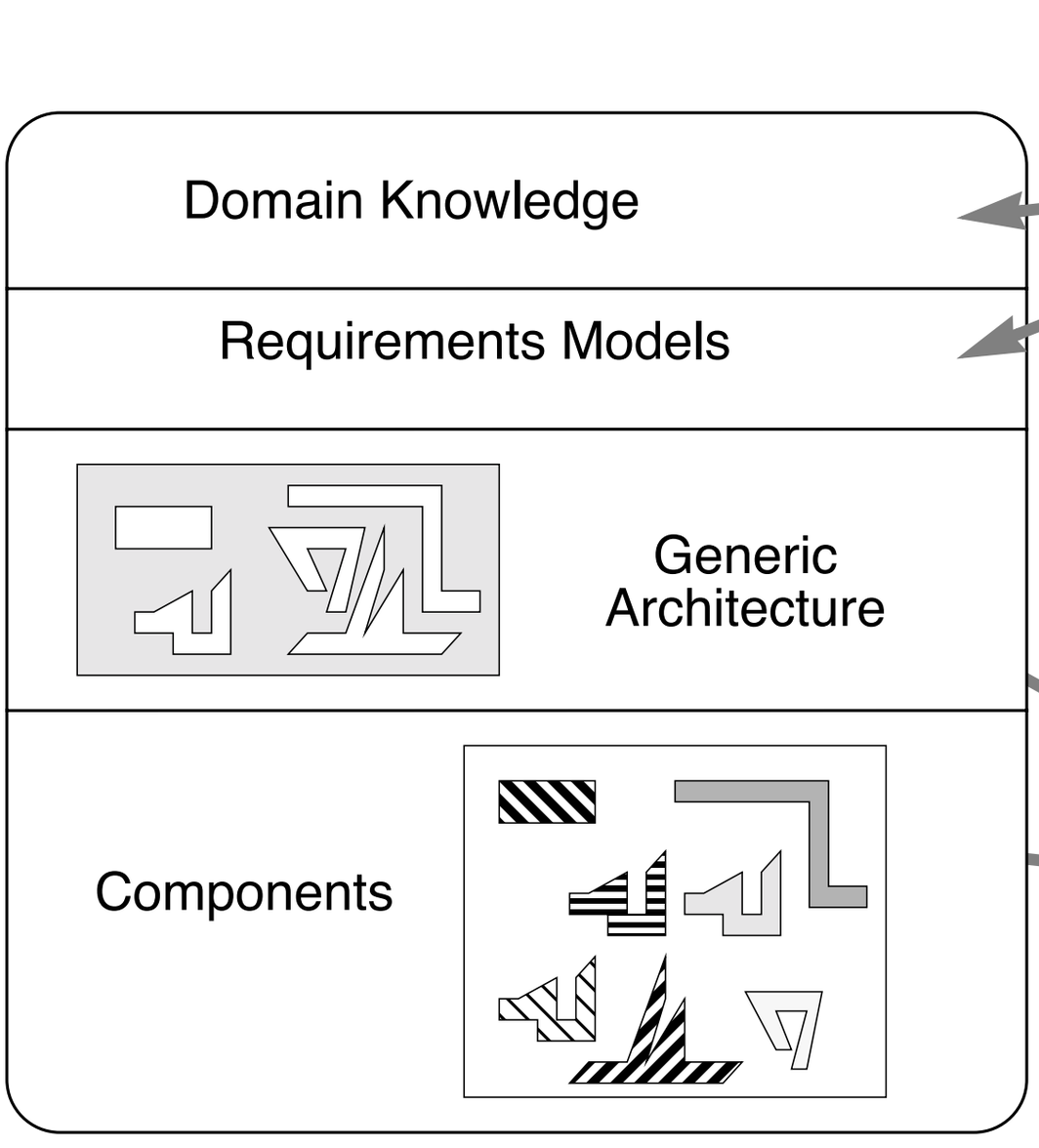
Ithaca

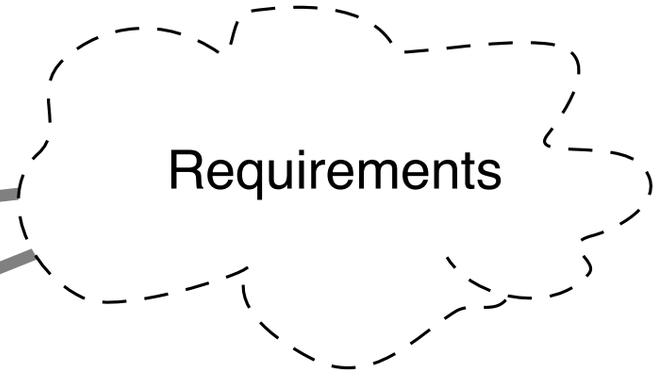


Nierstrasz, et al. ***Component-Oriented Software Development****. CACM 1992*

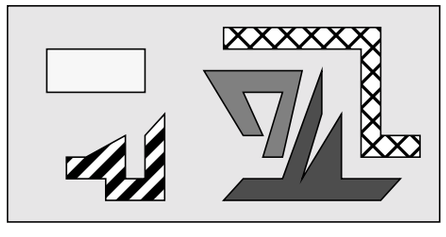
Ithaca







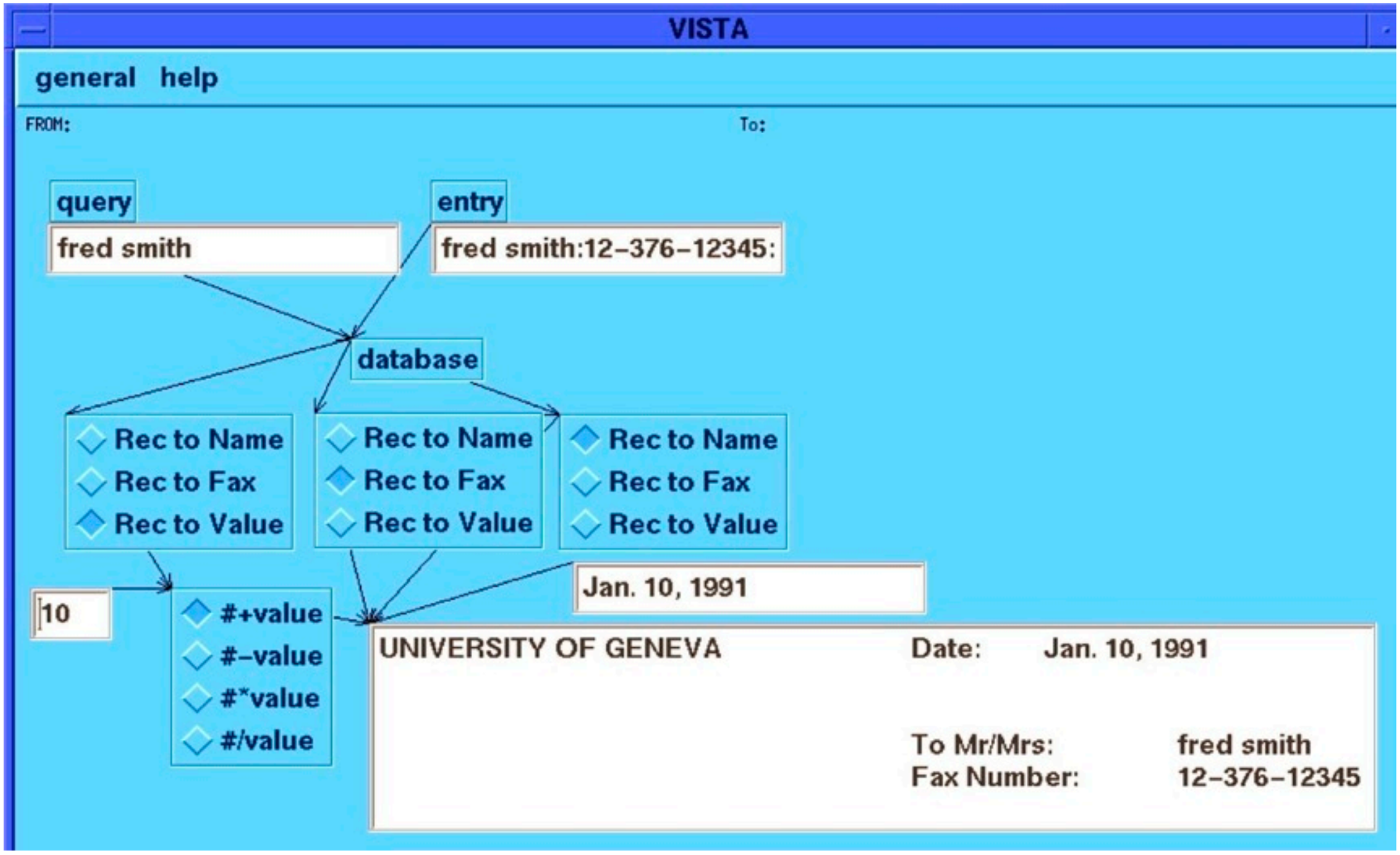
Component-Oriented Software Development is *Framework-Driven*



Specific Application

Nierstrasz, et al. ***Component-Oriented Software Development****. CACM 1992*

Visual Scripting



Nierstrasz, et al. ***Objects + Scripts = Applications****. Esprit 1991 Conference*

What would be a *pure* composition language?

Nierstrasz and Meijler. ***Research Directions in Software Composition****. ACM Computing Surveys 1995*

Applications = Components + Scripts (+ Glue)

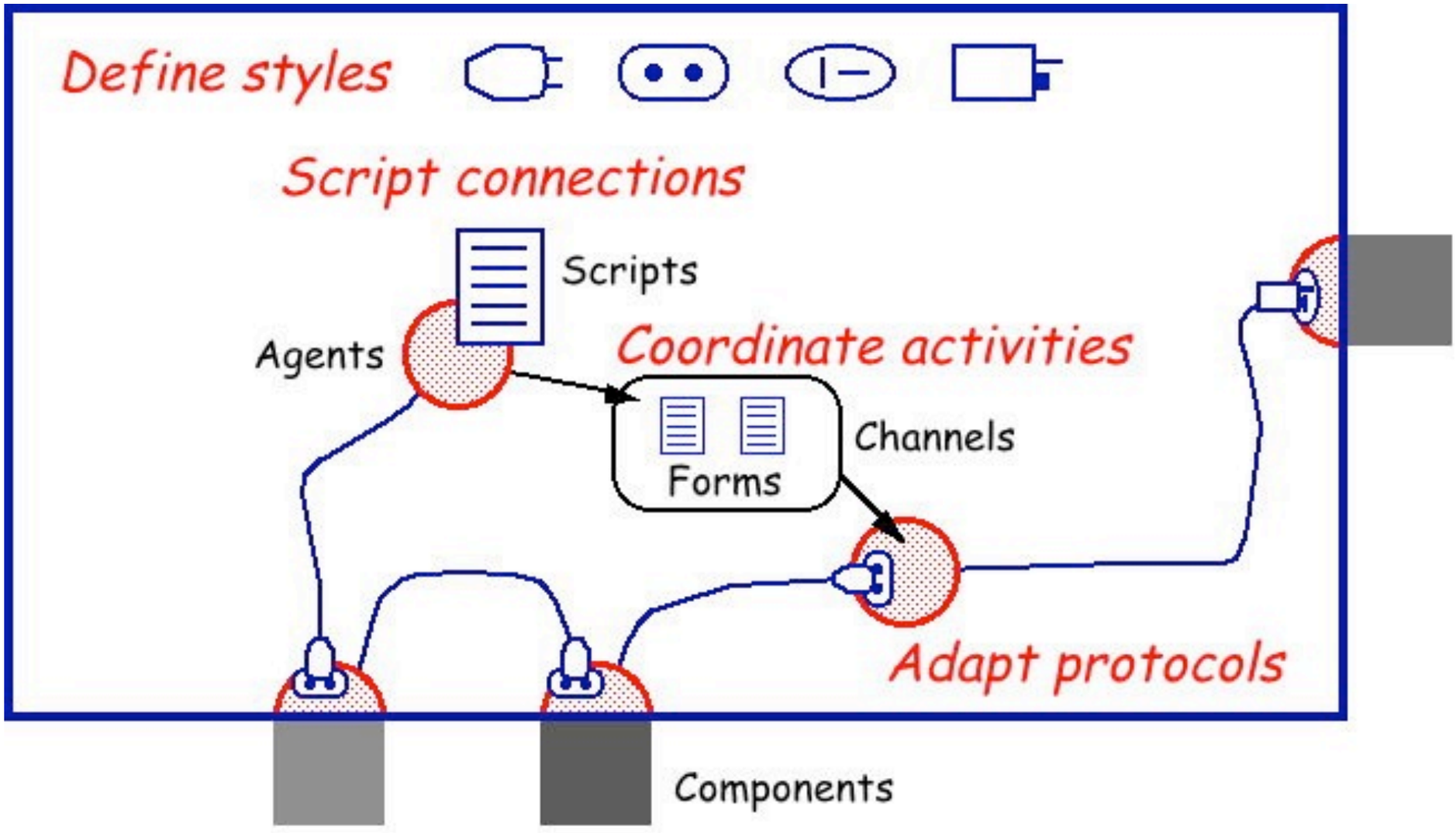
Components both *import* and *export* services

Scripts *plug* components together

A scripting language is a dedicated language for for orchestrating a set of tasks (or components).

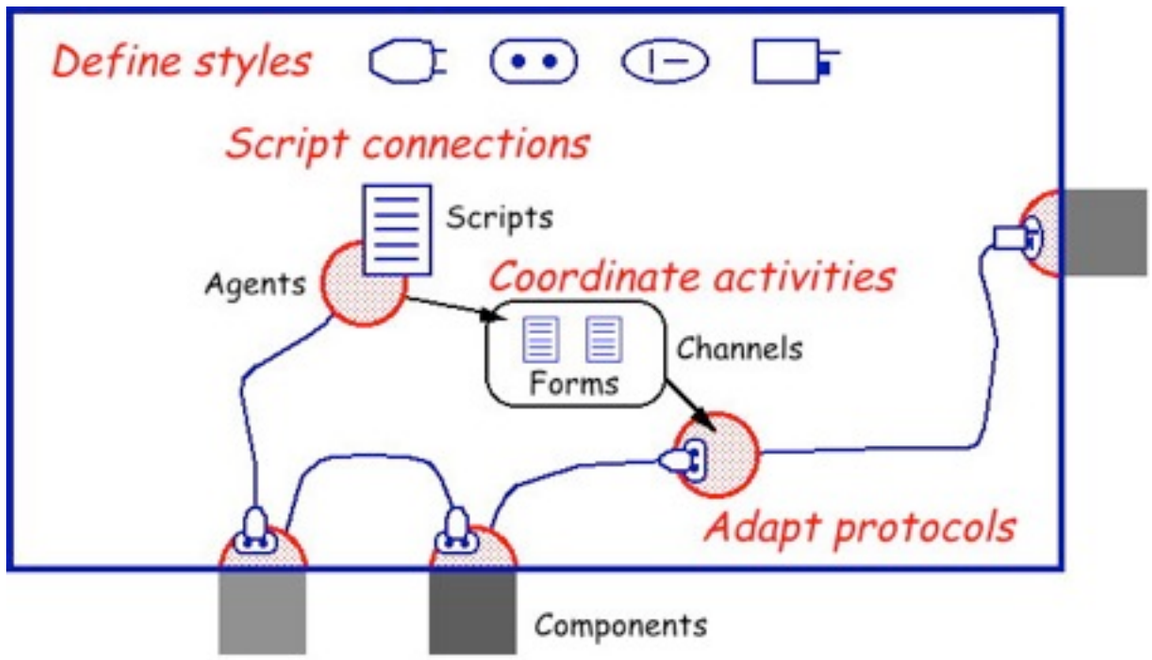
Schneider and Nierstrasz. ***Components, Scripts and Glue****.*
In Software Architectures — Advances and Applications, 1999.

Piccola



Piccola is a minimal language for defining plugs, connectors and scripts

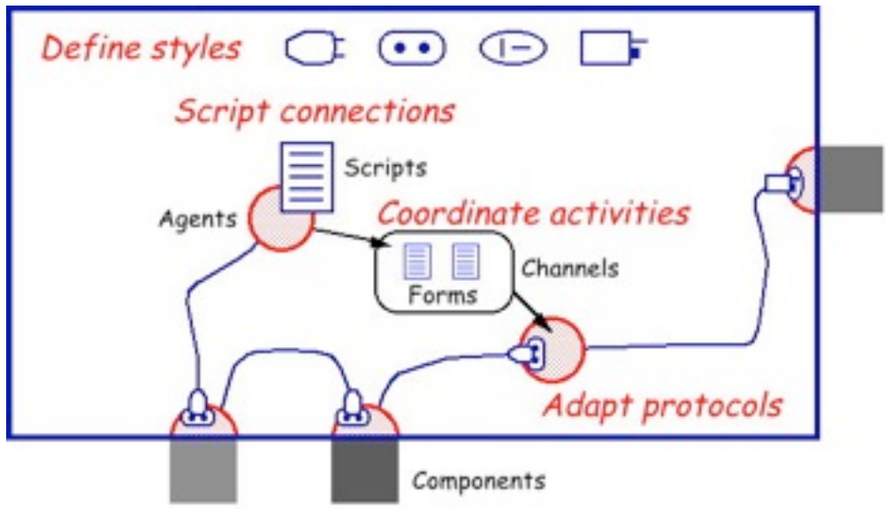
Piccola



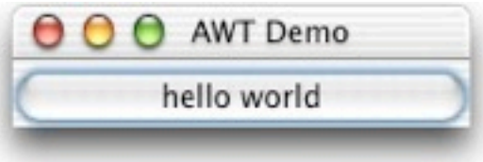
$$
\begin{array}{lcll}
A, B, C & ::= & \epsilon & \textit{empty form} \\
 & | & x\mapsto & \textit{bind} \\
 & | & x & \textit{variable} \\
 & | & A; B & \textit{sandbox} \\
 & | & hide_x & \textit{hide} \\
 & | & A \cdot B & \textit{extension} \\
 & | & \lambda x.A & \textit{abstraction} \\
 & | & AB & \textit{application} \\
 & | & \mathbf{R} & \textit{current root} \\
 & | & \mathbf{L} & \textit{inspect} \\
 & | & A \mid B & \textit{parallel} \\
 & | & \nu c.A & \textit{restriction} \\
 & | & c? & \textit{input} \\
 & | & c & \textit{output}
\end{array}
$$

built on a process calculus with explicit environments

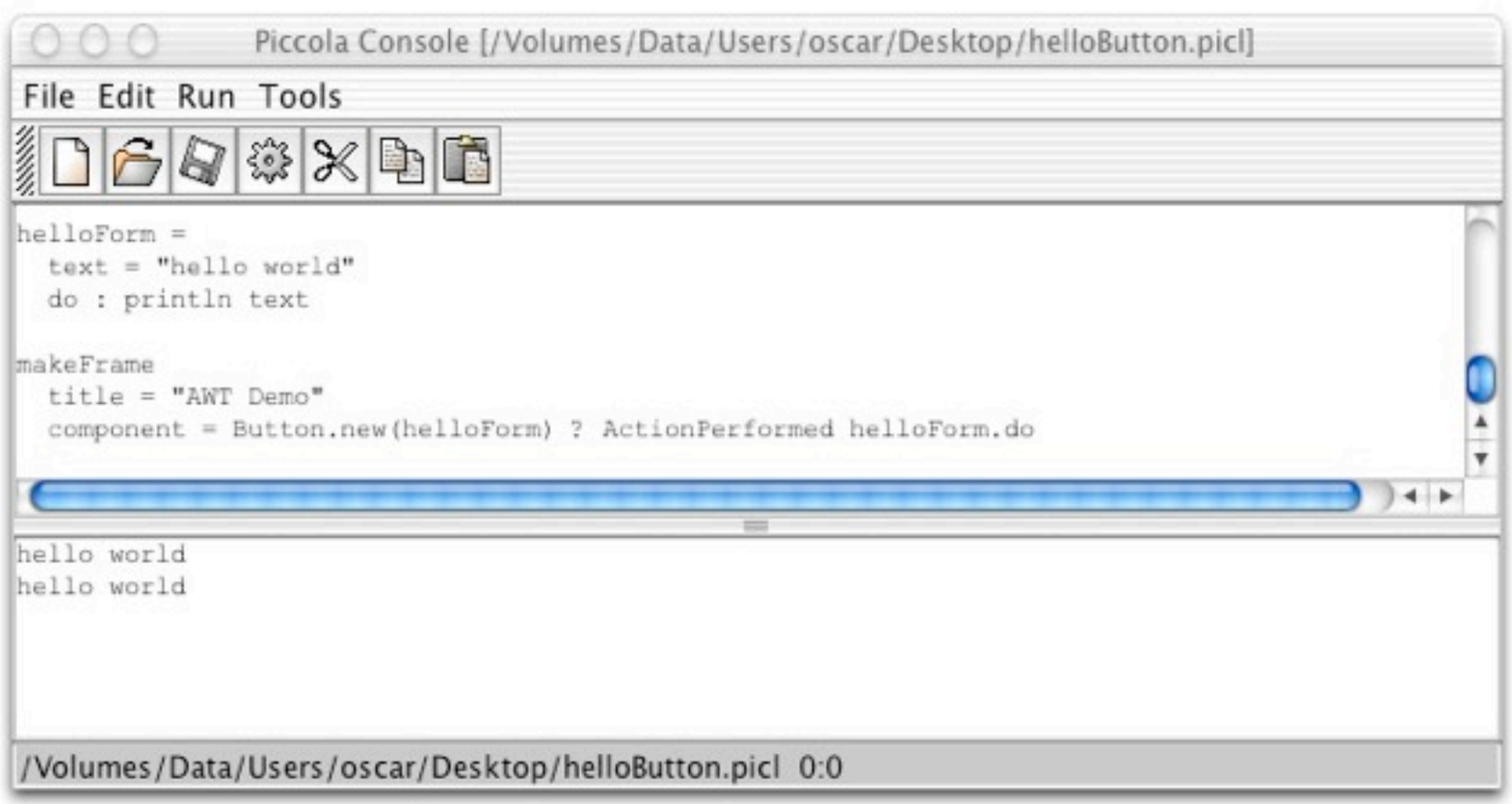
Franz Achermann and Oscar Nierstrasz. *A Calculus for Reasoning about Software Components*. Theoretical Computer Science 331(2), 2005.



$$
\begin{array}{rcll}
A, B, C & ::= & \epsilon & \textit{empty form} \\
& | & x\mapsto & \textit{bind} \\
& | & x & \textit{variable} \\
& | & A; B & \textit{sandbox} \\
& | & hide_x & \textit{hide} \\
& | & A \cdot B & \textit{extension} \\
& | & \lambda x.A & \textit{abstraction} \\
& | & AB & \textit{application} \\
& | & \mathbf{R} & \textit{current root} \\
& | & \mathbf{L} & \textit{inspect} \\
& | & A \mid B & \textit{parallel} \\
& | & \nu c.A & \textit{restriction} \\
& | & c? & \textit{input} \\
& | & c & \textit{output}
\end{array}
$$



for **scripting** components written in Java



Achermann, et al. ***Piccola — a Small Composition Language****. Formal Methods for Distributed Processing, Cambridge, 2001*

API = Metamodel = DSL

Configuration = Model = Script

What I learned ...

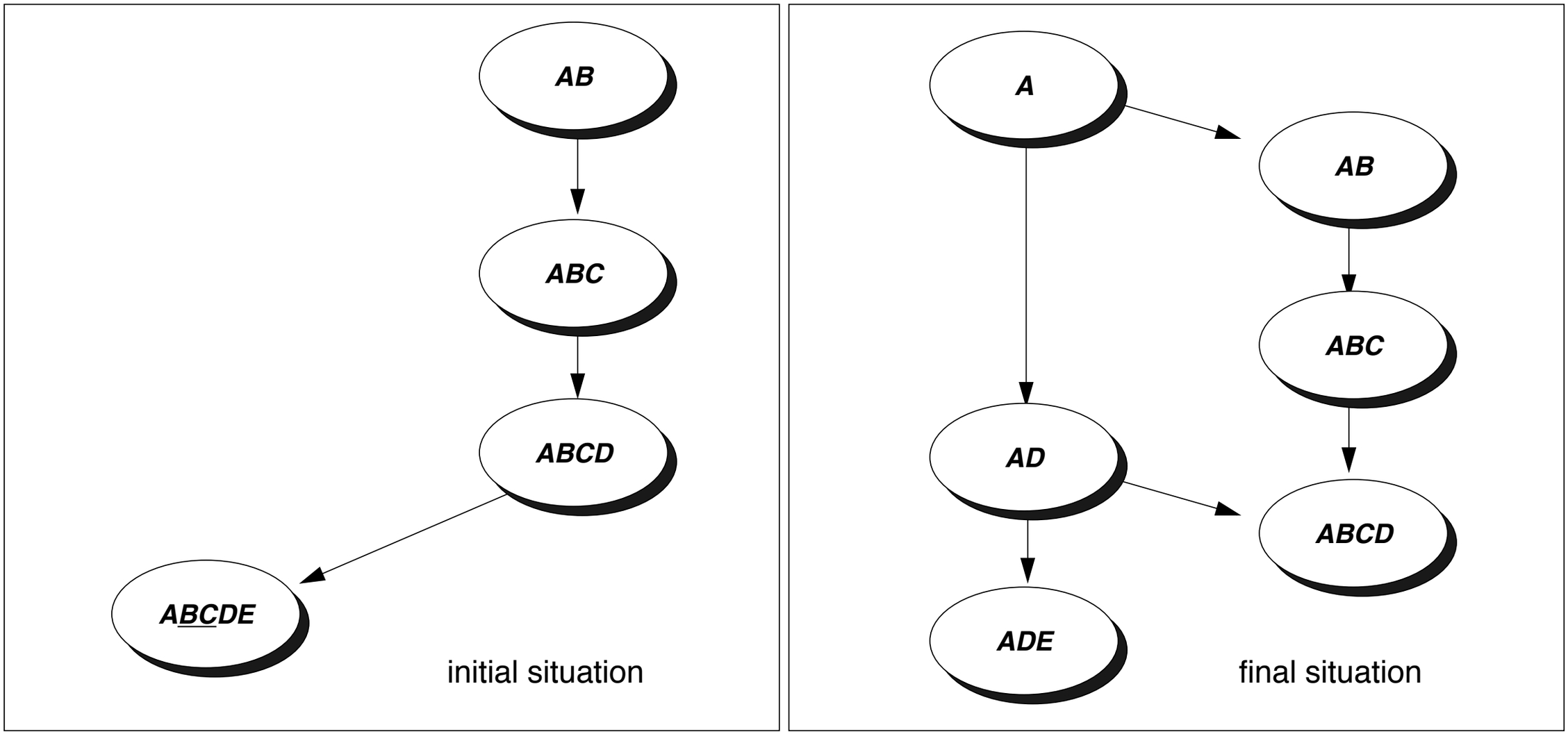
Scripts, not components, are the key to composition

4. Legacy OOP


O O P
FIRST PREFIX
SECOND PREFIX
THIRD PREFIX
AUTO DESTRUCT TEST

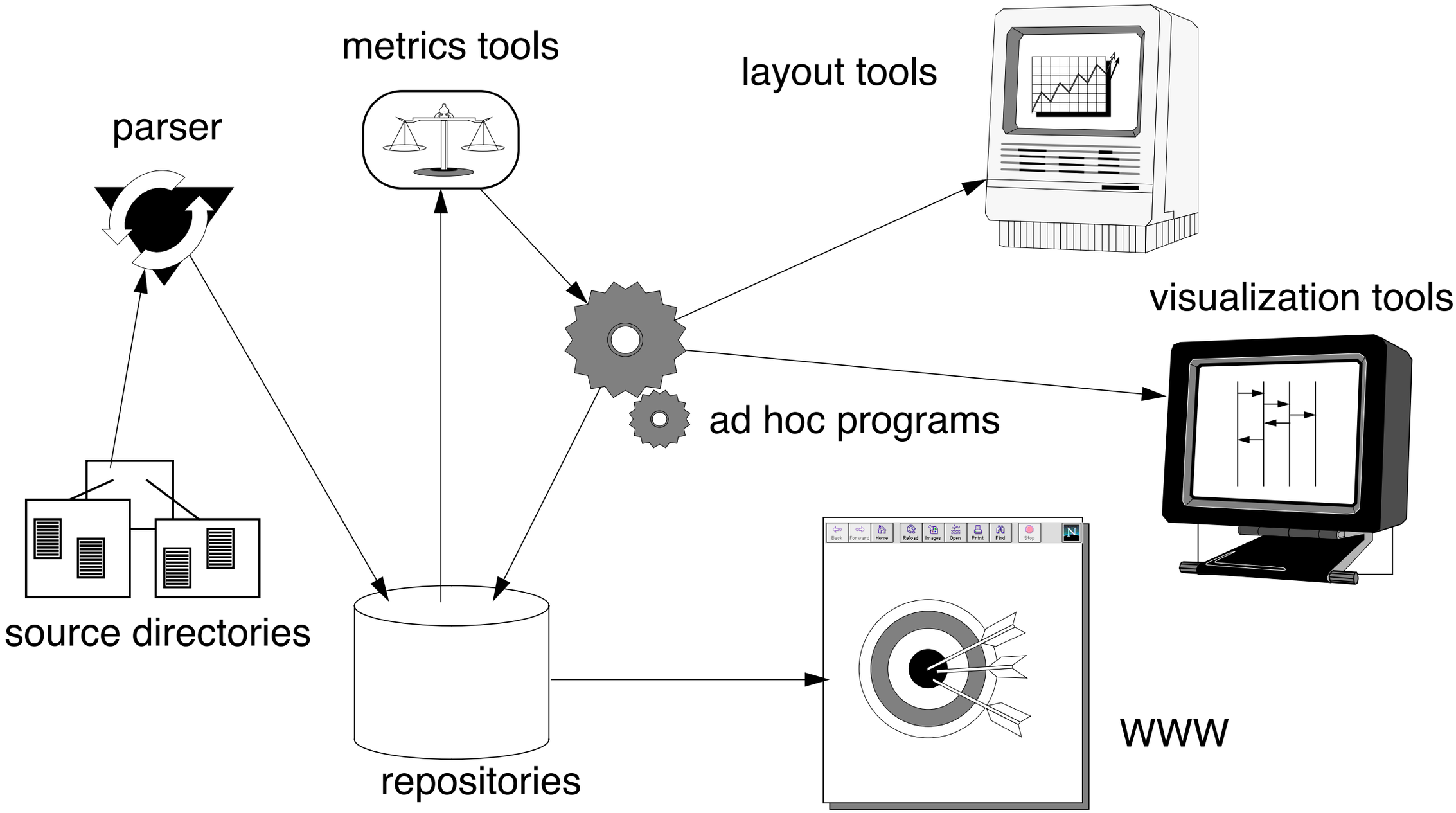
How to reengineer OO legacy systems towards component-based frameworks?

Refactoring

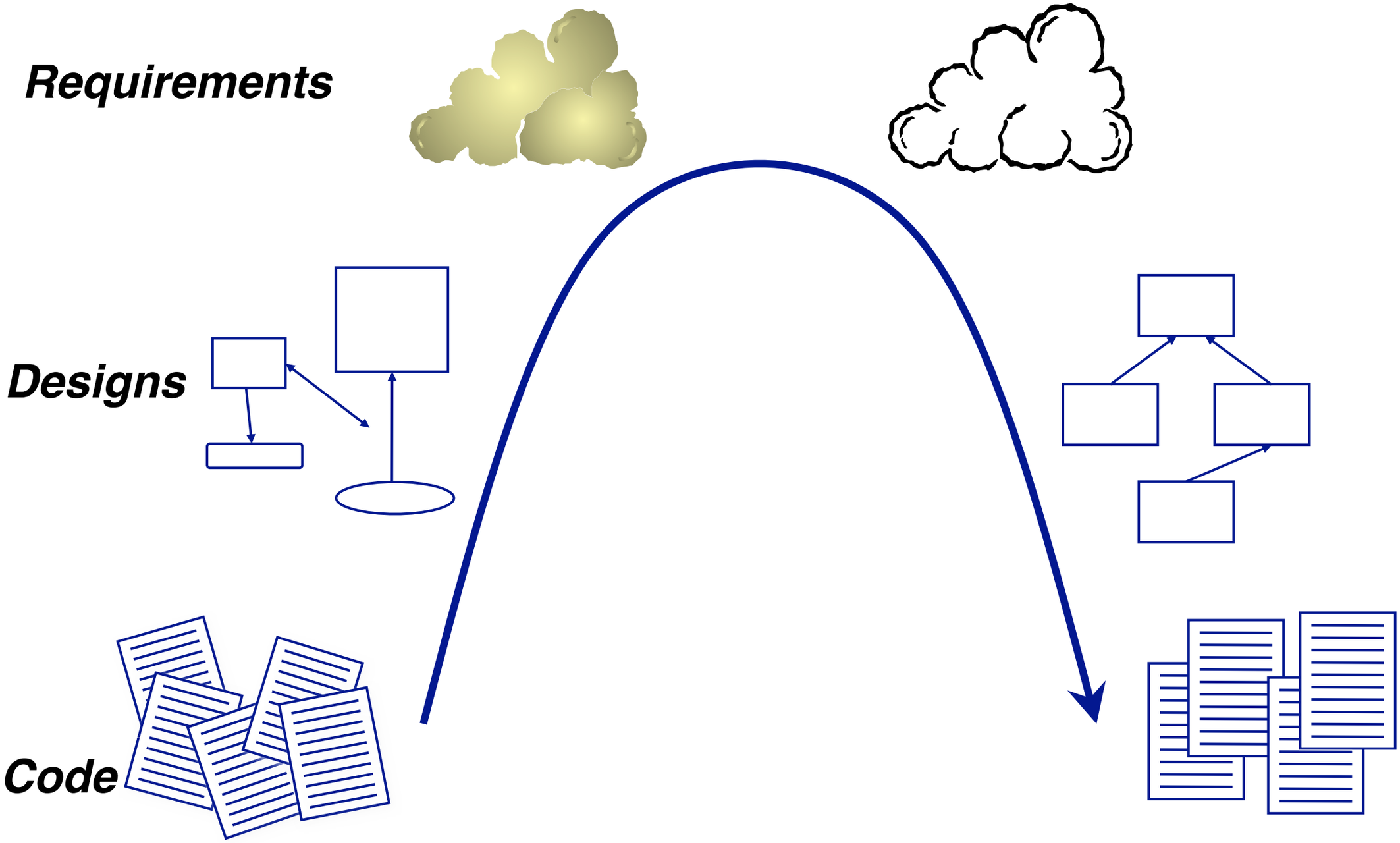


Gibbs, et al. ***Class Management for Software Communities****. CACM 1990*

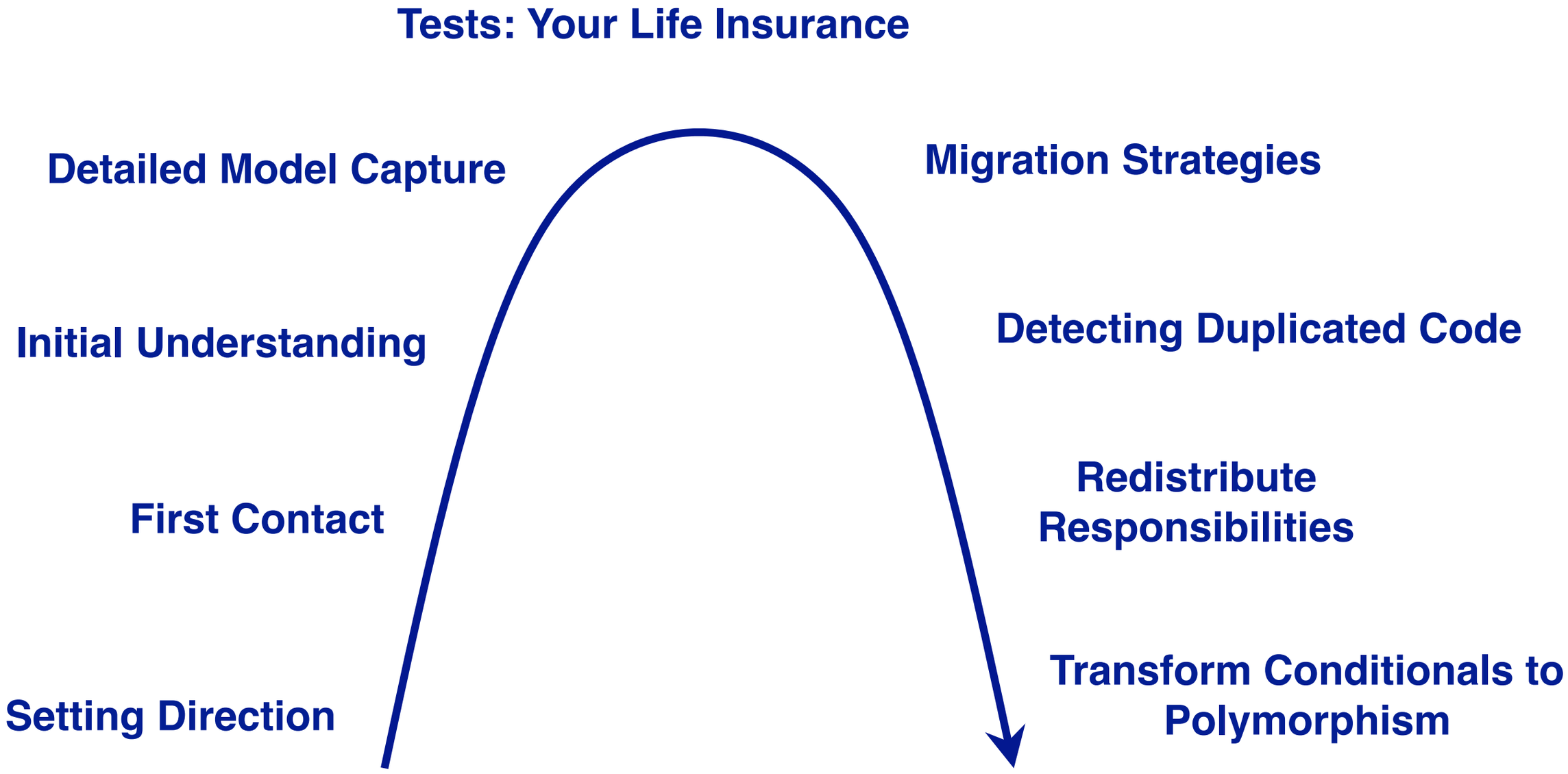
FAMOOS


metrics tools
layout tools
parser
visualization tools
ad hoc programs
source directories
repositories
www

The Reengineering Life-Cycle

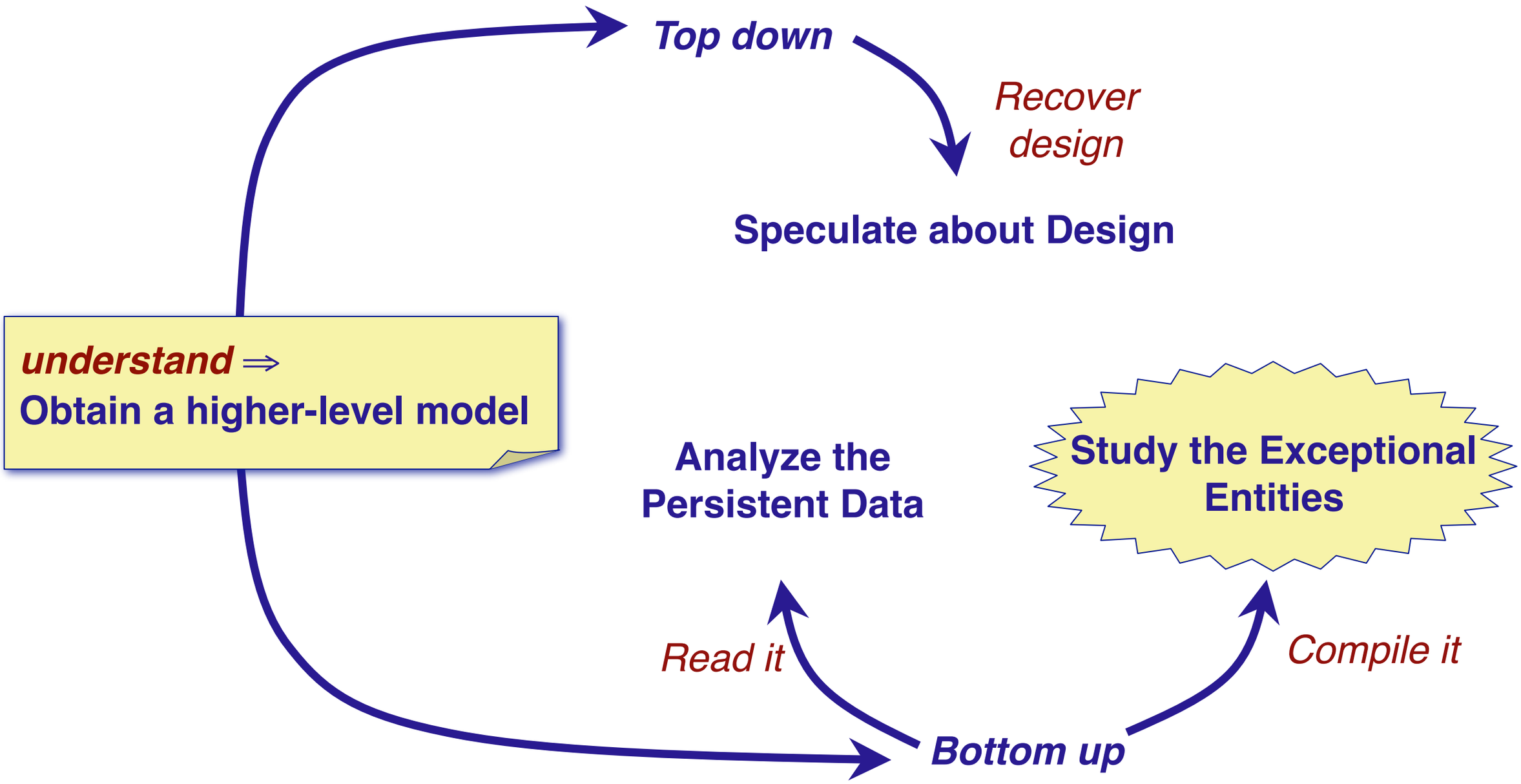


Lightweight Reengineering Patterns

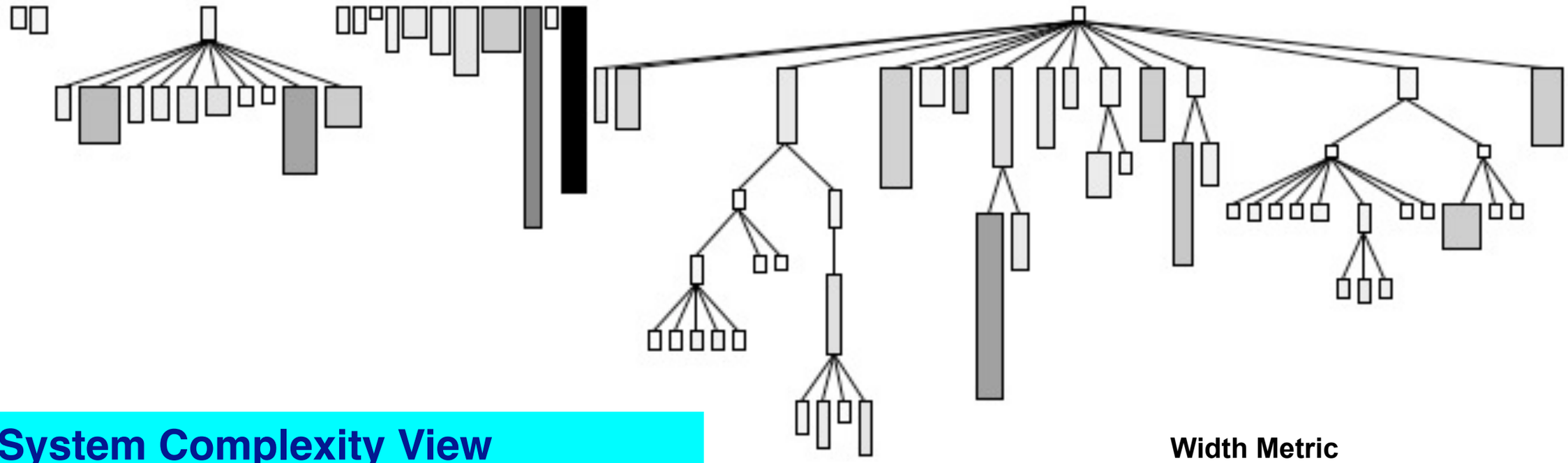


Demeyer, et al. ***Object-Oriented Reengineering Patterns****, Morgan Kaufmann, 2002*

Initial Understanding



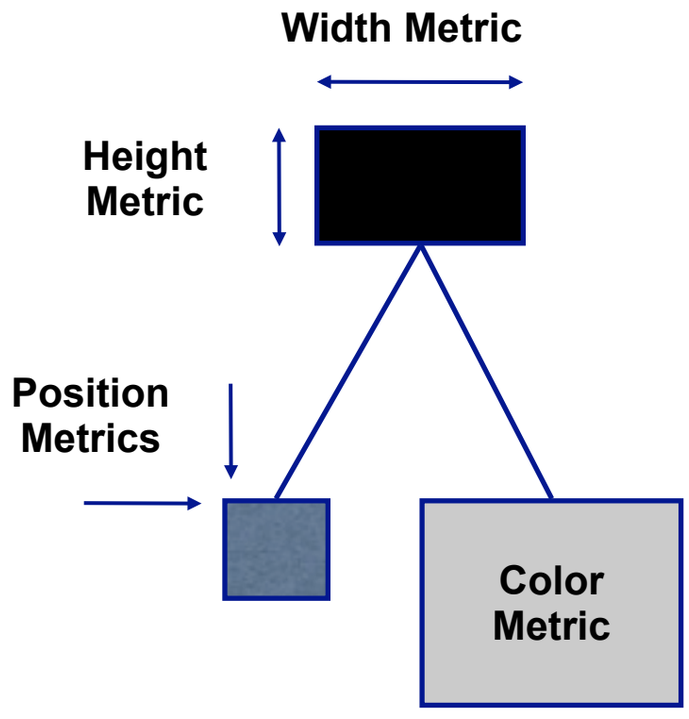
System Complexity View

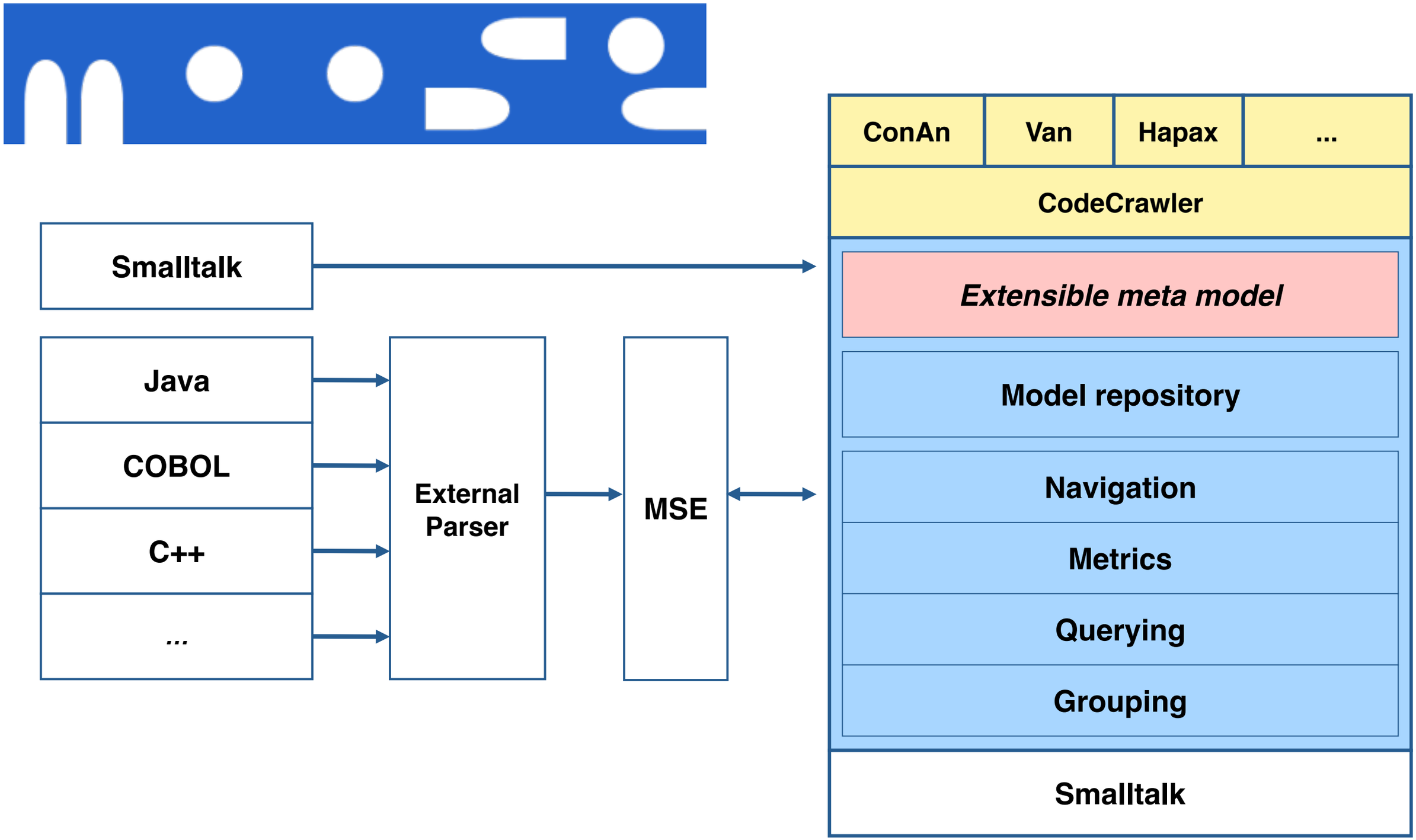


System Complexity View

Nodes = Classes
Edges = Inheritance Relationships

Width = Number of Attributes
Height = Number of Methods
Color = Number of Lines of Code

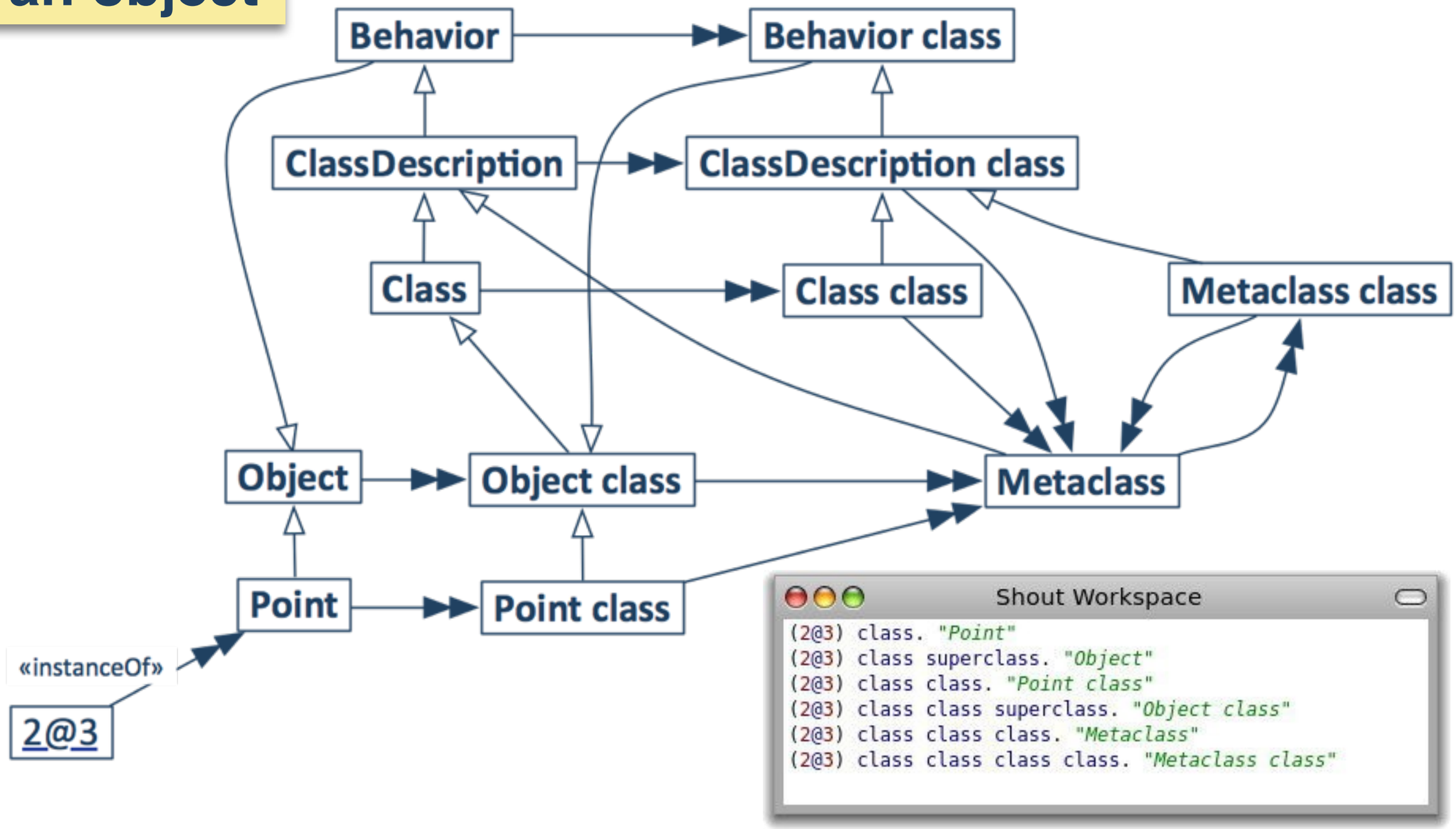




Nierstrasz, et al. ***The Story of Moose: an Agile Reengineering Environment****. ESEC/FSE 2005*

(Re)discovering Smalltalk

Everything
is an object



Shout Workspace

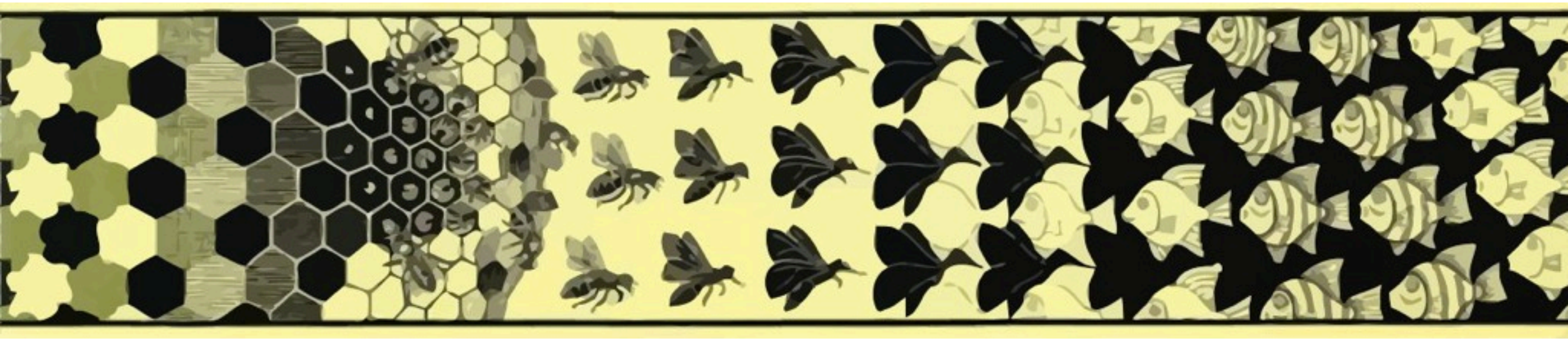
```
(2@3) class. "Point"
(2@3) class superclass. "Object"
(2@3) class class. "Point class"
(2@3) class class superclass. "Object class"
(2@3) class class class. "Metaclass"
(2@3) class class class class. "Metaclass class"
```

What I learned ...

Less is more

5. Software Evolution

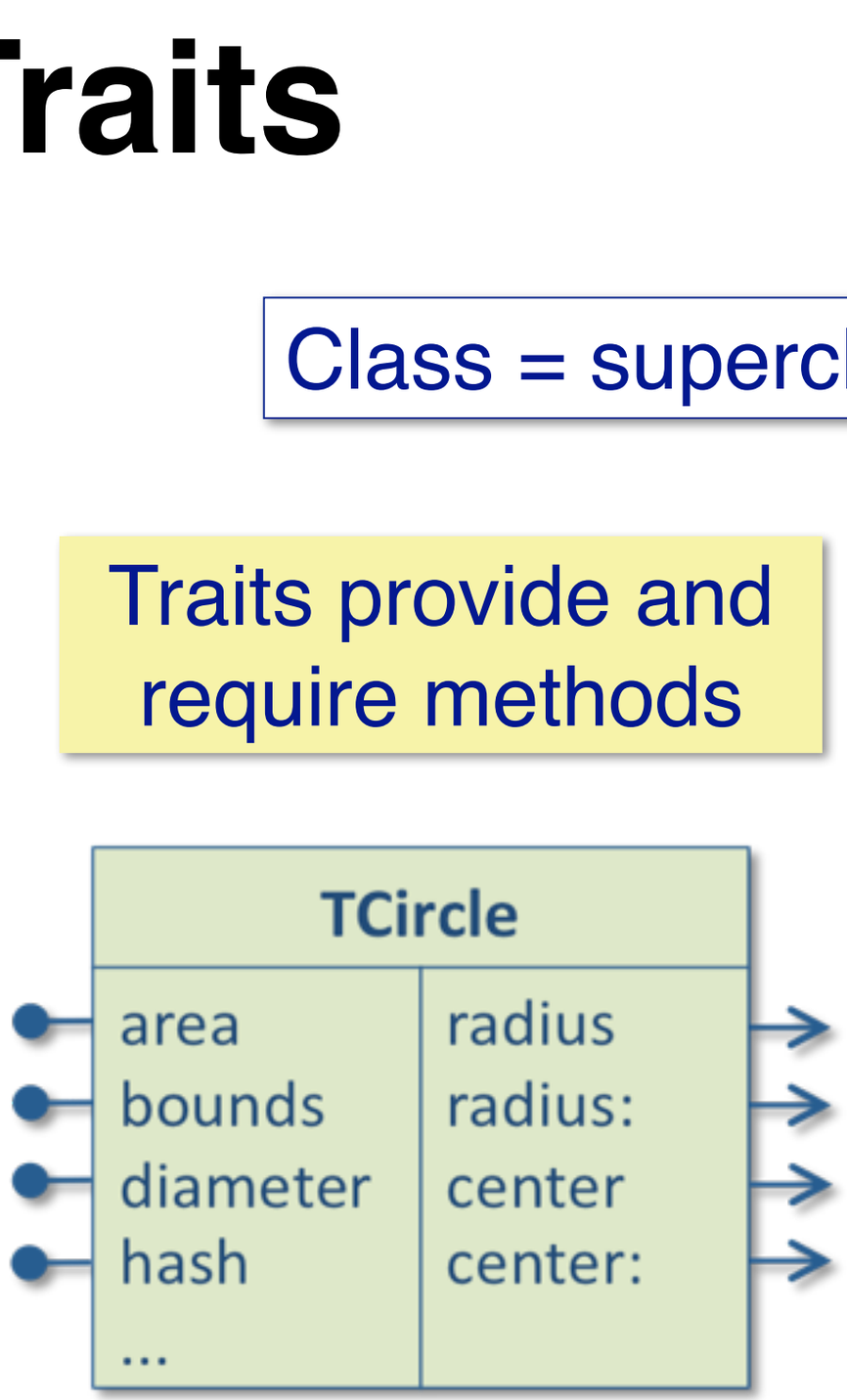
How to gracefully evolve running software systems?

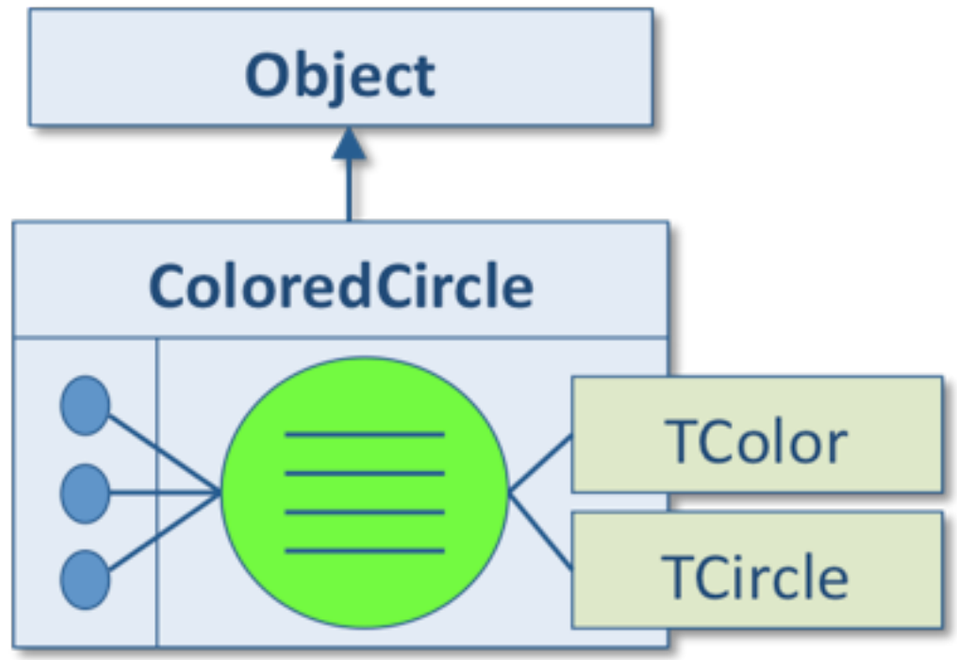


Traits

Class = superclass + state + traits + glue

Traits provide and require methods

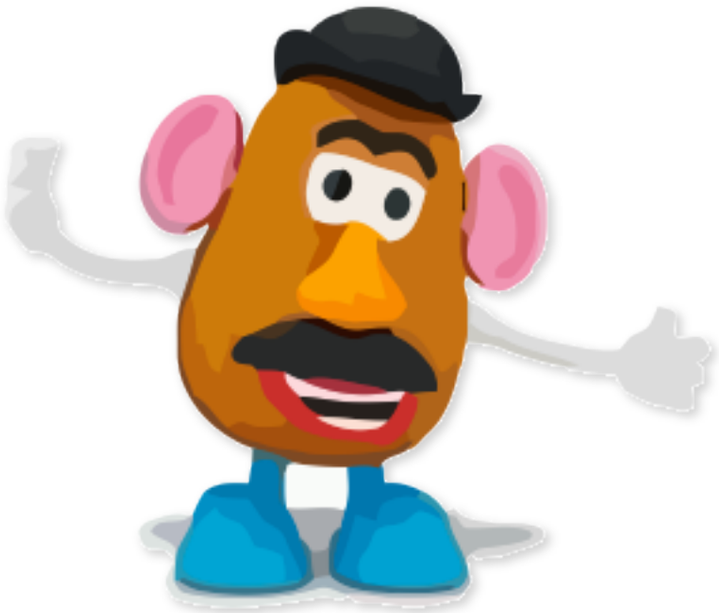




The composing class retains control

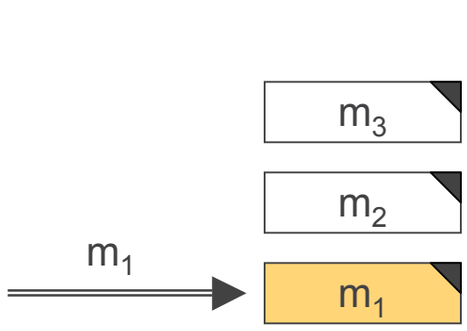
Ducasse, et al. ***Traits: A Mechanism for fine-grained Reuse****. TOPLAS 2006*

Talents

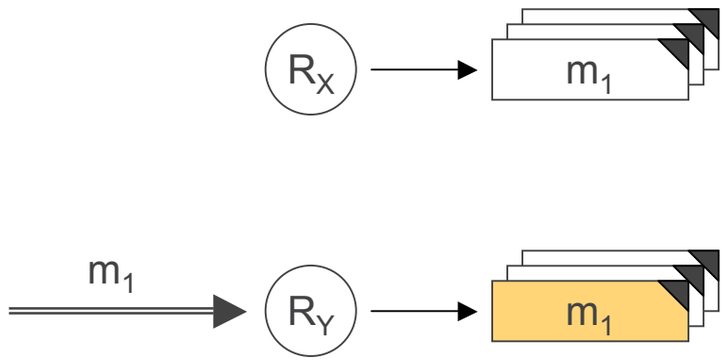


Ressia, et al. ***Talents: an environment for dynamically composing units of reuse****. Software: Practice and Experience, 2012*

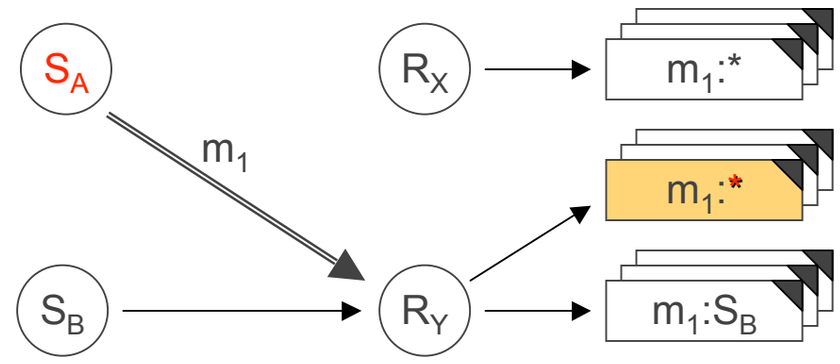
Context-Oriented Programming



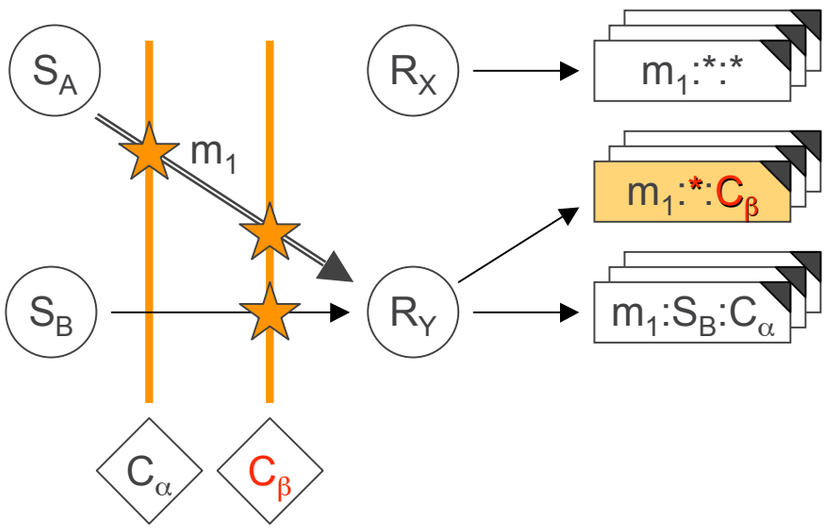
1D: procedural



2D: OOP



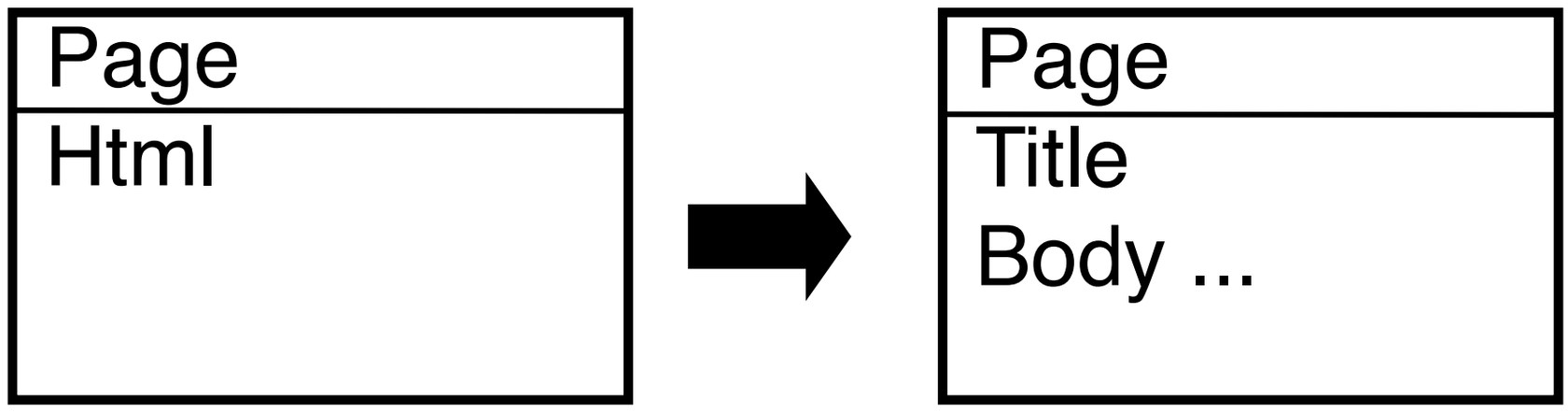
3d: subject-oriented



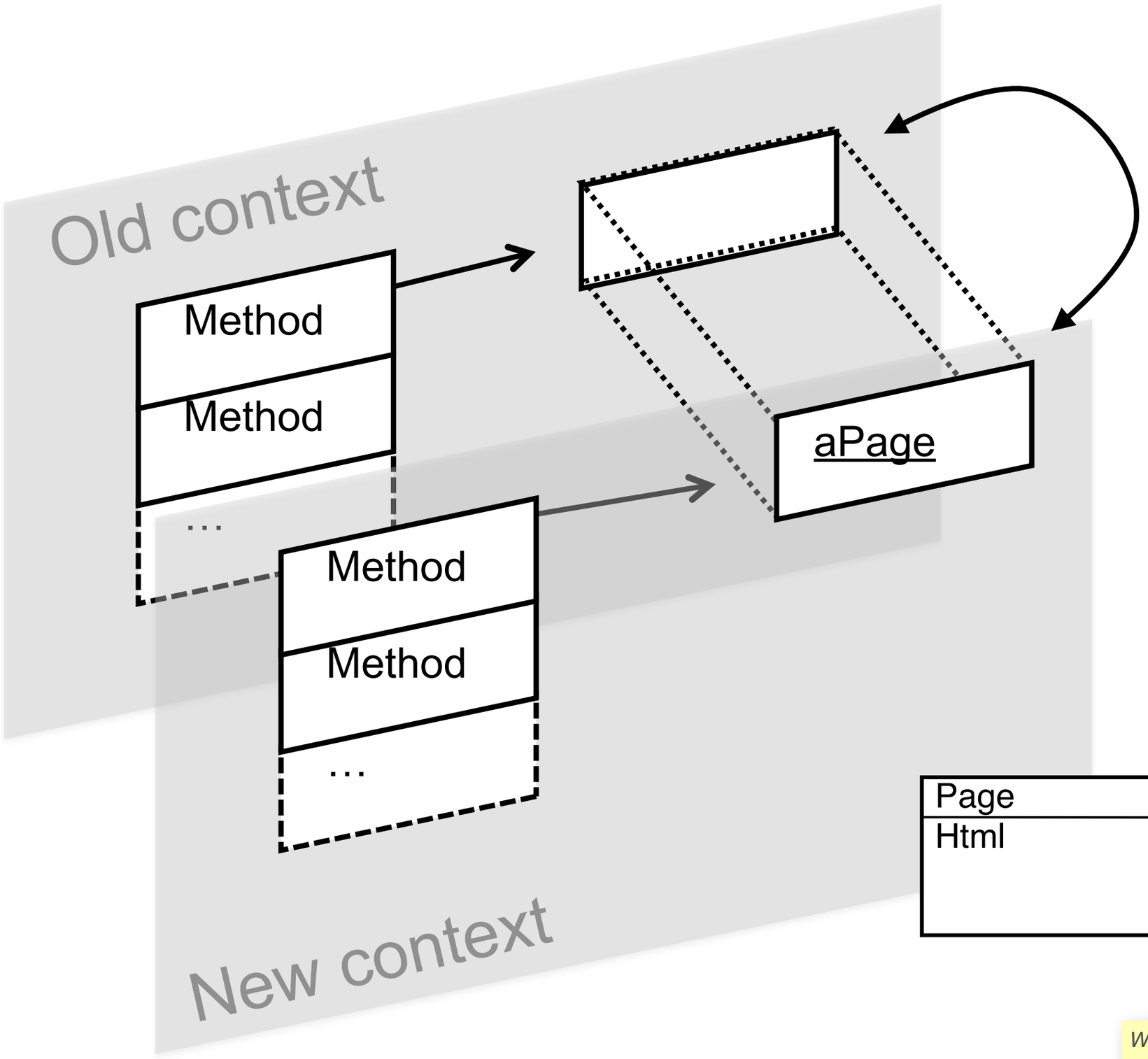
n-D: COP

Hirschfeld, et al. ***Context-Oriented Programming****. JOT 2008*

First-class contexts enable dynamic updates

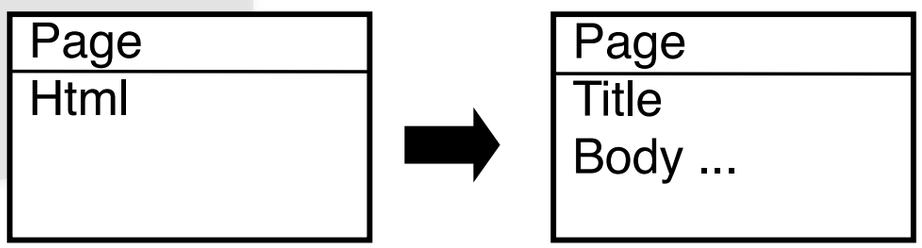


First-class contexts enable dynamic updates



bidirectional transformation

Shared object has one identity but two states



Wernli, et al. ***Incremental Dynamic Updates with First-class Contexts****. TOOLS Europe 2012*

What I learned ...

Explicit context supports software evolution

6. The end

... or is it?

Embrace Objects

Everything is an Object (Keep it Simple)

Scripts are the flip side of objects

Context is the *other* flip side of objects

SCG Present and Past