UNIVERSITÄT BERN

Software Architecture Extraction

Andrea Caracciolo

Adapted from slides by **Oscar Nierstrasz** and **Mircea Lungu**

Roadmap

- Introduction to SAR
- The Architecture of Architecture Recovery
- Top-down SAR
- Bottom-up SAR
- Tool Demo

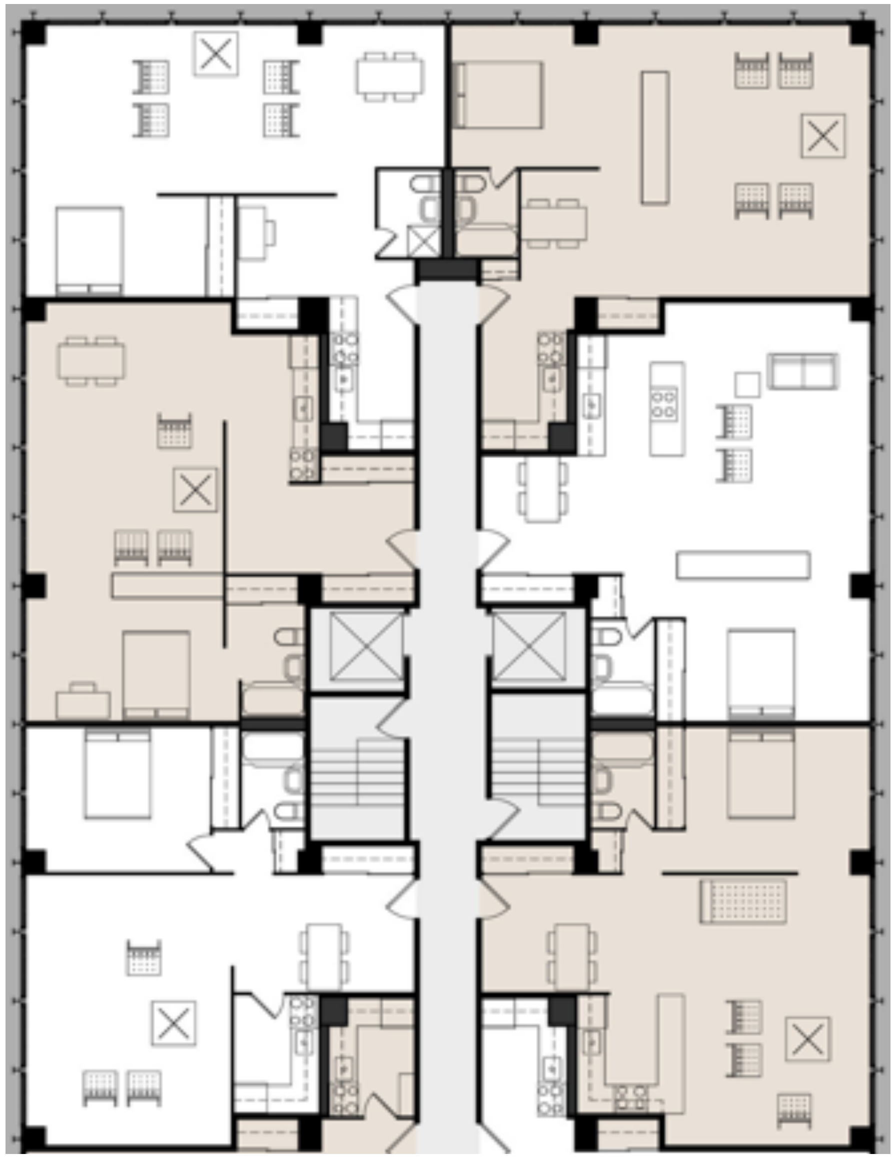
Roadmap

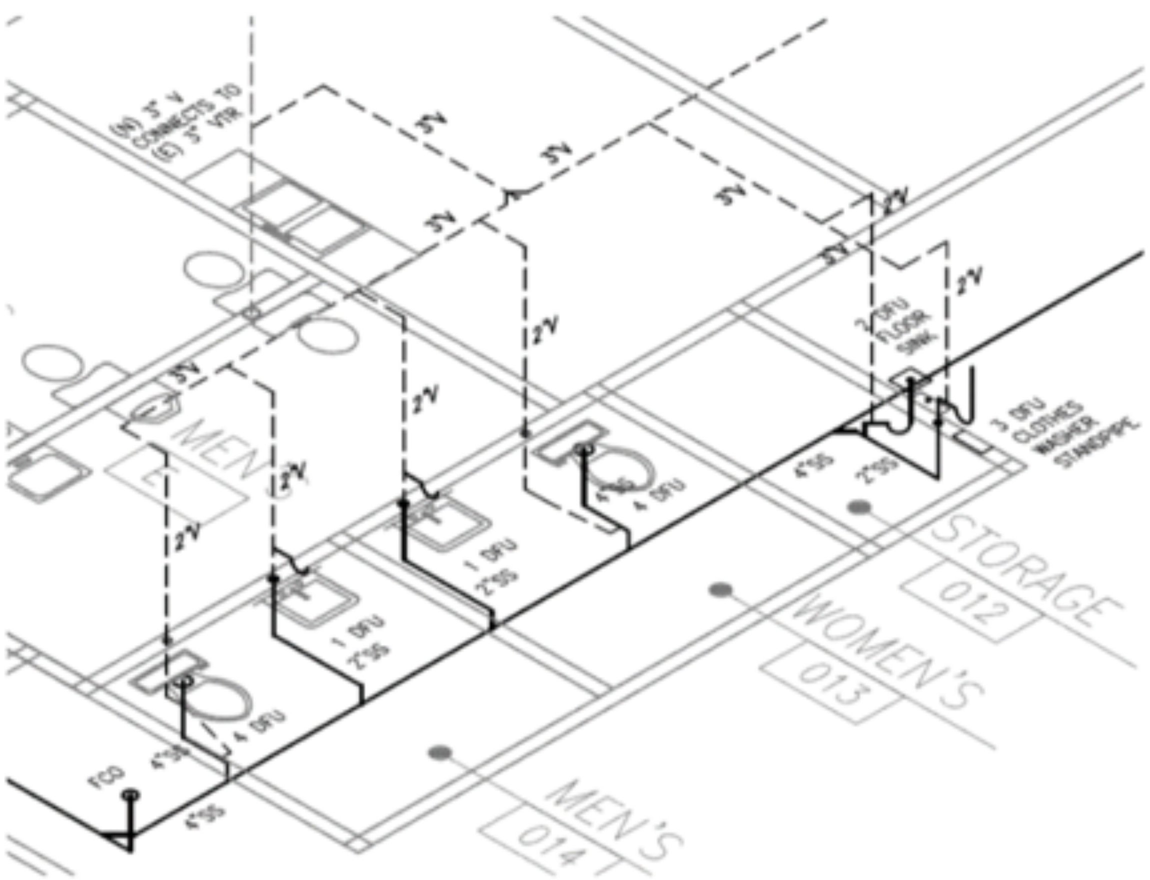
> **Introduction to SAR**
  - **Architecture**
  - Viewpoints, Styles, ADL's
  - Architecture Recovery
> The Architecture of Architecture Recovery
> Top-down SAR
> Bottom-up SAR
> Tool Demo

Structure: Elements and Form

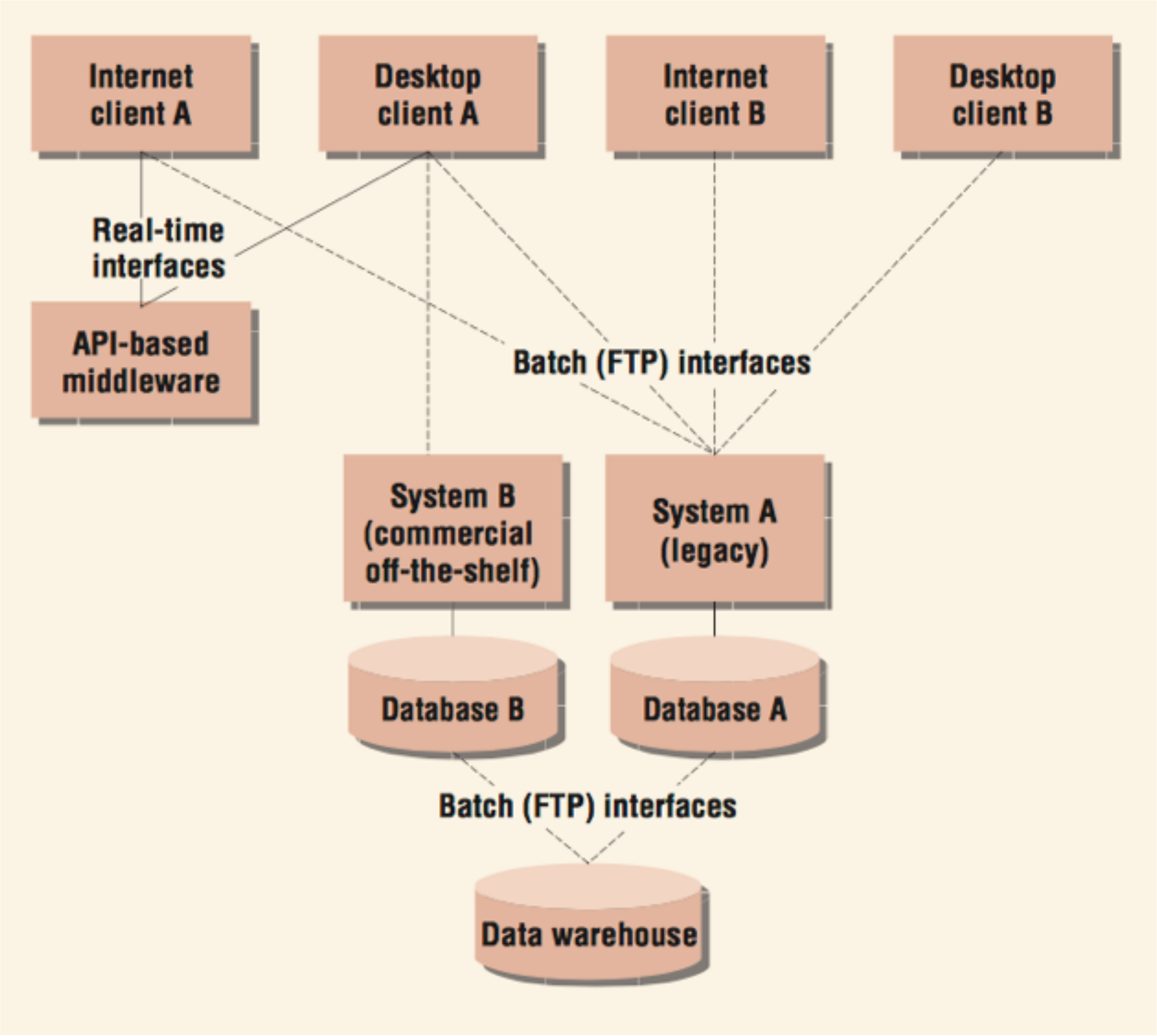
"[...] the fundamental organization of a system embodied in its **components**, their **relationships** to each other [..]"

[IEEE 1421, 2000]





Structure: Elements and Form

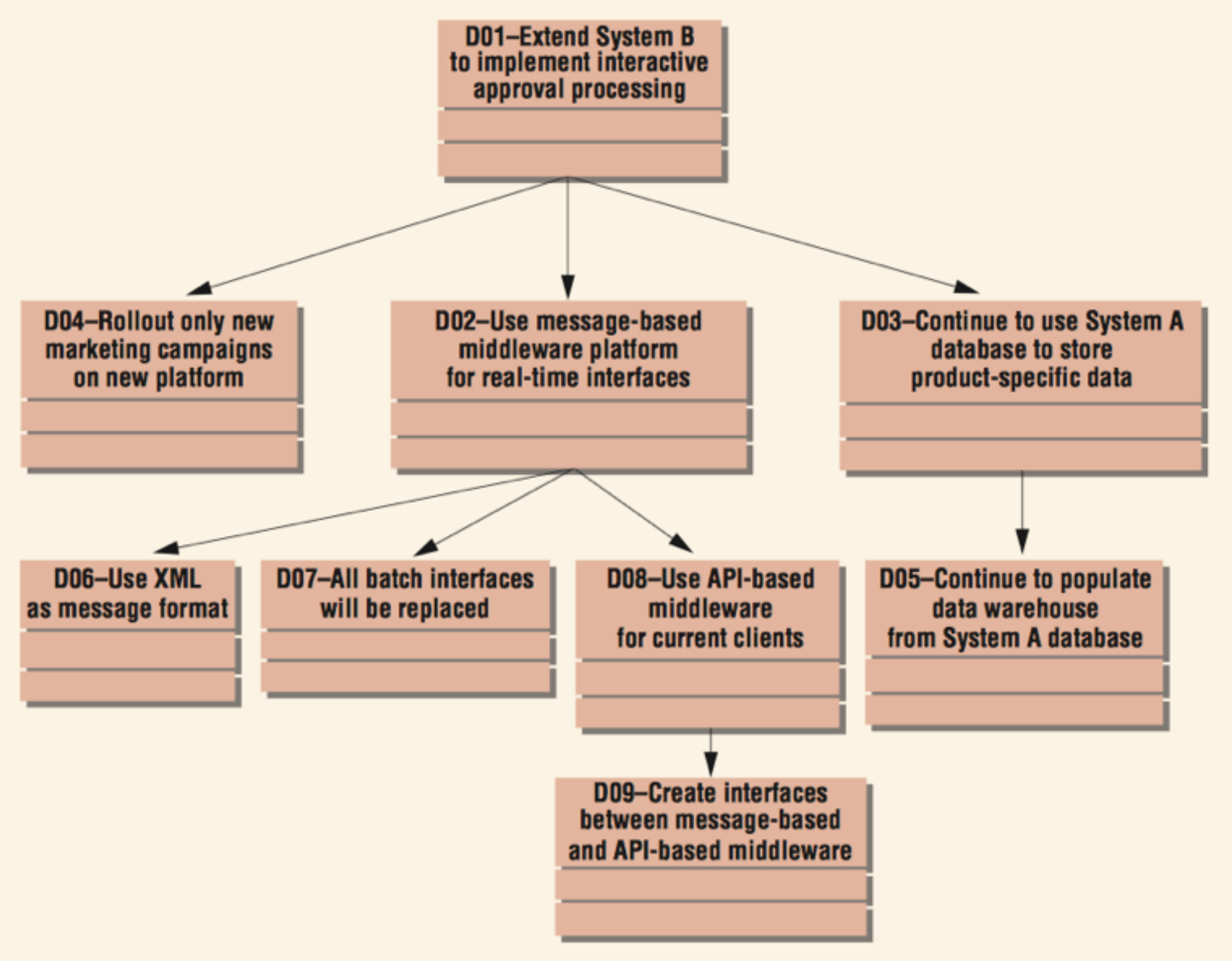


Rationale: Design Decisions

"The structure of components, their interrelationships, and **principles** and **guidelines** governing their design and evolution over time."

[Garlan and Perry, 1995]

Rationale: Design Decisions



Rationale: Design Decisions

- architectural decisions are ones that permit a system to meet its **quality attribute** and **behavioral requirements**.

- architecture is design, but not all design is architecture

- design decisions resulting in element properties that are *not visible* - that is, make no difference outside the element - are non-architectural.

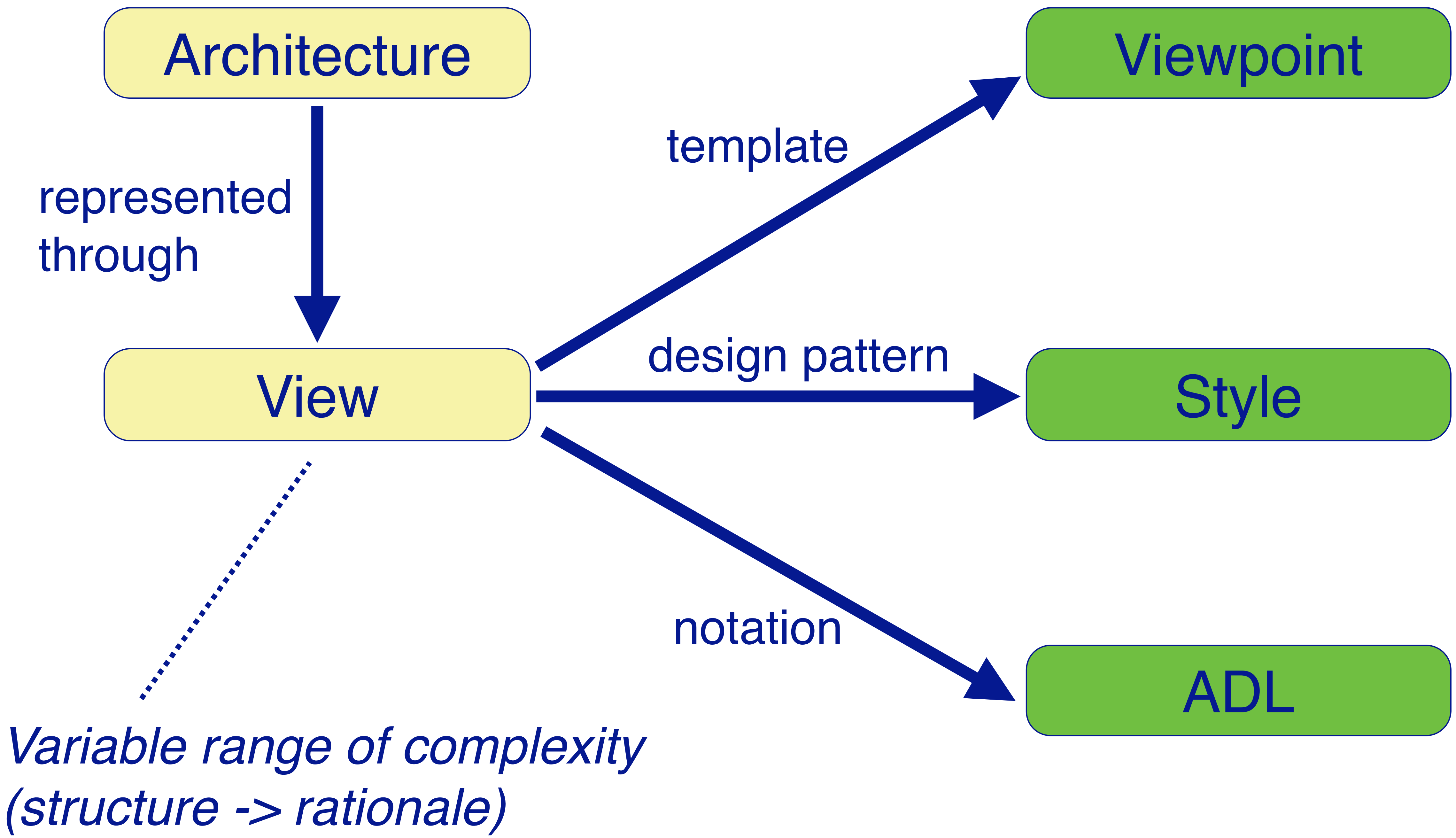
[Clements et al., Software Architectures and Documentation]

http://msdn.microsoft.com/en-us/library/ee658098.aspx

Roadmap

> **Introduction to SAR**
  - Architecture
  - **Viewpoints, Styles, ADL's**
  - Architecture Recovery
> The Architecture of Architecture Recovery
> Top-down SAR
> Bottom-up SAR
> Tool Demo

Architectural View



Architectural View

A **view** is a representation of a whole system from the perspective of a related set of concerns.

A **concern** is an interest which pertains to the system's development, its operation or any other aspects that are important to one or more stakeholders.

- e.g.: performance, security, distribution, maintenance

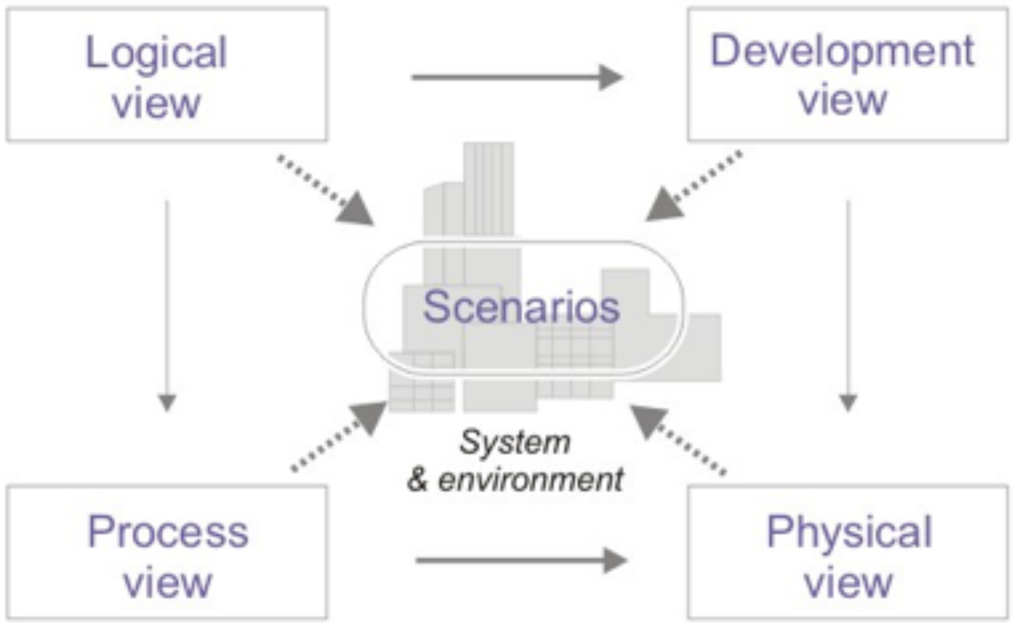
A **stakeholder** is an individual, team, or organization with interests in, or concerns relative to, a system.

- e.g.: development team, operational staff, project manager

Architectural Viewpoint

- A **viewpoint** is
  - a specification of the conventions for constructing and using views
  - a template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis.
- Consensus in software engineering community
- Viewpoints catalogues
  - Kruchten ’95
  - Hofmeister ’99

Kruchten 4+1



Logical view: Logical representation of the system's functional structure

- **stakeholders**: end-user
- **formalization**: UML Class diagram

Development view: design time software structure, modules, sub-systems and layers

- **stakeholders**: developer
- **formalization**: UML Component diagram

Process view: system processes and how they communicate. Focuses on the runtime behavior

- **stakeholders:** developer, system engineer
- **formalization**: UML Activity diagram

Physical view: topology, physical connections, mapping of architectural elements to nodes

- **stakeholders**: system engineer
- **formalization**: UML deployment diagram

Classical Architectural Viewpoints

Run-time	How are responsibilities distributed amongst run-time entities?
Process	How do processes communicate and synchronize?
Dataflow	How do data and tasks flow through the system?
Deployment	How are components physically distributed?
Module	How is the software partitioned into modules?
Build	What dependencies exist between modules?

Architectural Style

An **architectural style** defines a family of systems in terms of a pattern of structural organization.

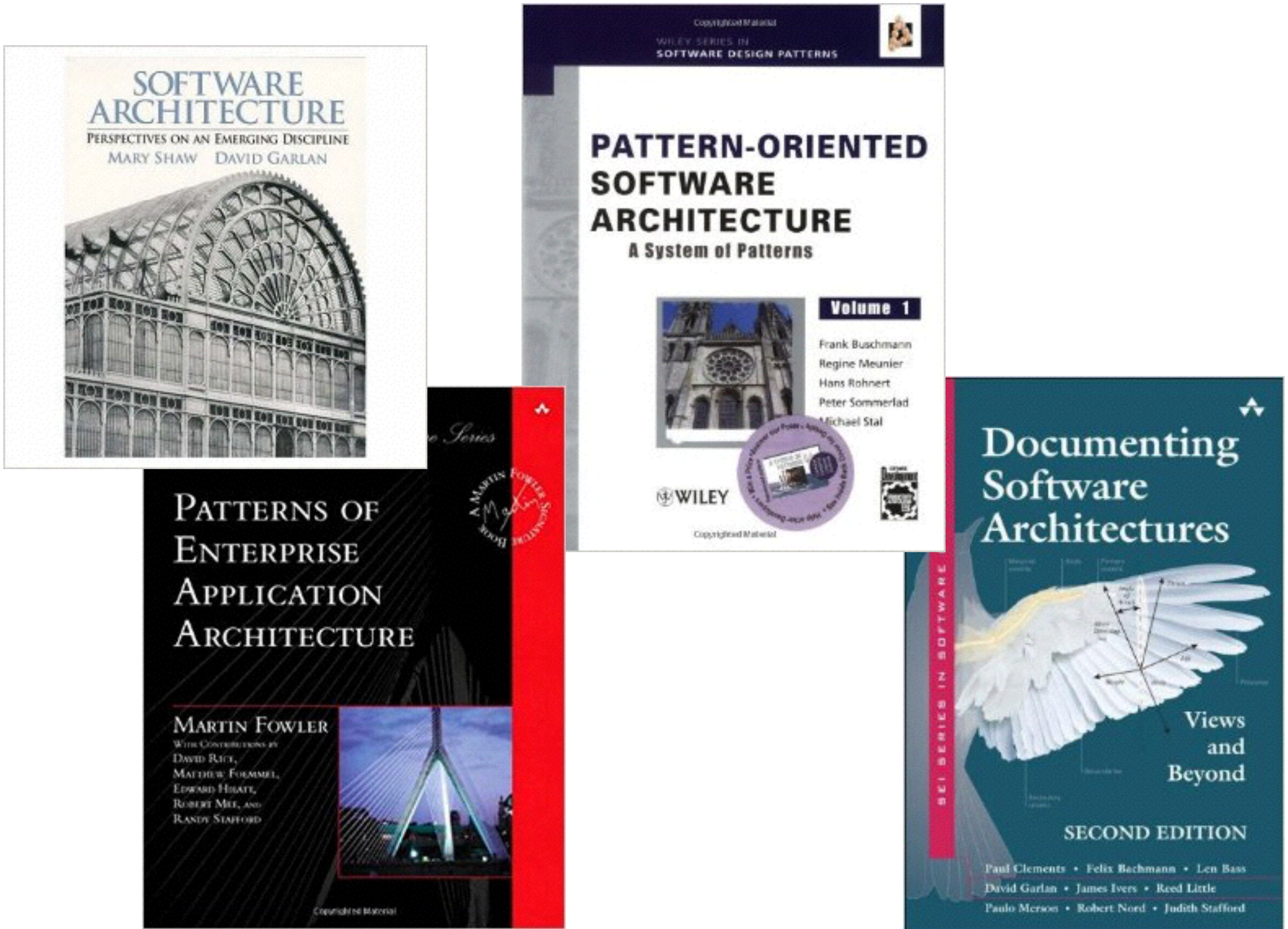
More specifically, an architectural style defines a vocabulary of **components** and **connector** types, and a set of **constraints** on how they can be combined.

[Shaw and Garlan]

Classical Architectural Styles

Layered	Elements in a given layer can only see the layer below. Callbacks used to communicate upwards
Client-Server	Separate application logic from interaction logic. Clients may be “fat” or “thin”
Dataflow	Data or tasks strictly flow “downstream”.
Blackboard	Tools or applications coordinate through shared repository.

Architectural Style “Catalogues”



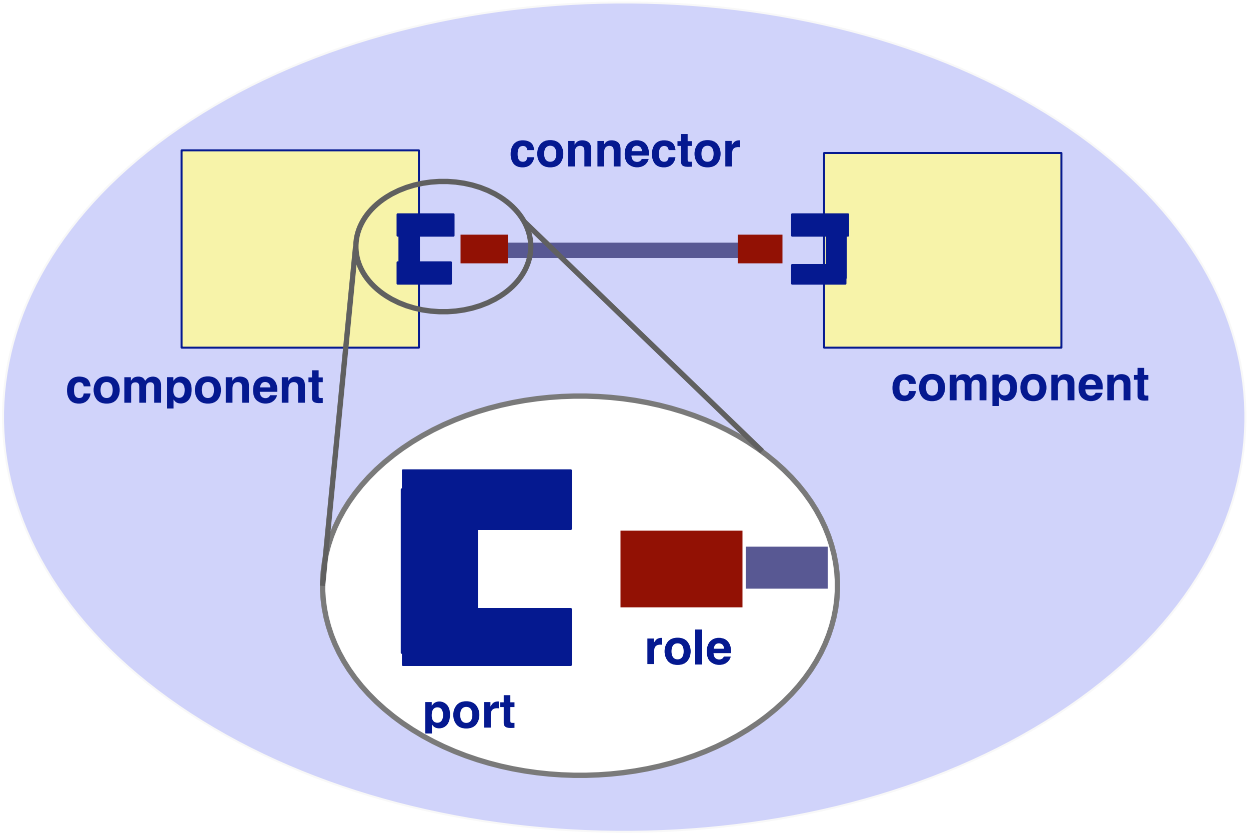
Architectural Description Languages (ADLs)

Formal languages for representing and reasoning about software architecture.

Provide a **conceptual framework** and a concrete syntax for characterizing architectures.

Some are **executable**, or implemented in a general-purpose programming language.

Common ADL Concepts



Component: unit of computation or data store. Typically contains interface (ports) and formal behavioral description.

Connector: architectural building block used to model interactions among components. Typically contains interface (roles) and formal behavioral description.

Configuration: connected graph of components and connectors that describe architectural structure.

ADL example

```
process implementation process1.basic
   subcomponents
      A: thread t1.basic; B: thread t2.basic; C: thread t2.basic;
   connections
      cn1: data port signal -> A.p1;
      cn2: data port A.p2 -> B.p1;
      cn3: data port B.p2 -> result1;
      cn4: data port A.p2 -> C.p1;
      cn5: data port C.p2 -> result2;
      cn6: data port A.p3 -> status;
      cn7: event port init -> C.reset;
   flows
      f1: flow path signal->cn1->A.fs1->cn2->B.fs1->cn3->result1;
      f2: flow path signal->cn1->A.fs1->cn4->C.fs1->cn5->result2;
      f3: flow sink init->cn7->C.fs2;
      f4: flow source A.fs2->cn6->status;
end process1.basic;
```

```
system implementation Software.Basic
subcomponents
   Sampler_A : process Collect_Samples {
      Source_Text => ("collect_samples.ads", "collect_samples.adb") ;
      Period => 50 ms ;
} ;
end Software.Basic ;
```

Some ADLs

- **Wright**: underlying model is CSP, focuses on connectivity of concurrent components.
- **Darwin**: focuses on supporting distributed applications. Components are single-threaded active objects.
- **Rapide**: focuses on developing a new technology for building large-scale, distributed multi-language systems.

Roadmap

- > **Introduction to SAR**
  - — Architecture
  - — Viewpoints, Styles, ADL's
  - — **Architecture Recovery**
- > The Architecture of Architecture Recovery
- > Top-down SAR
- > Bottom-up SAR
- > Tool Demo

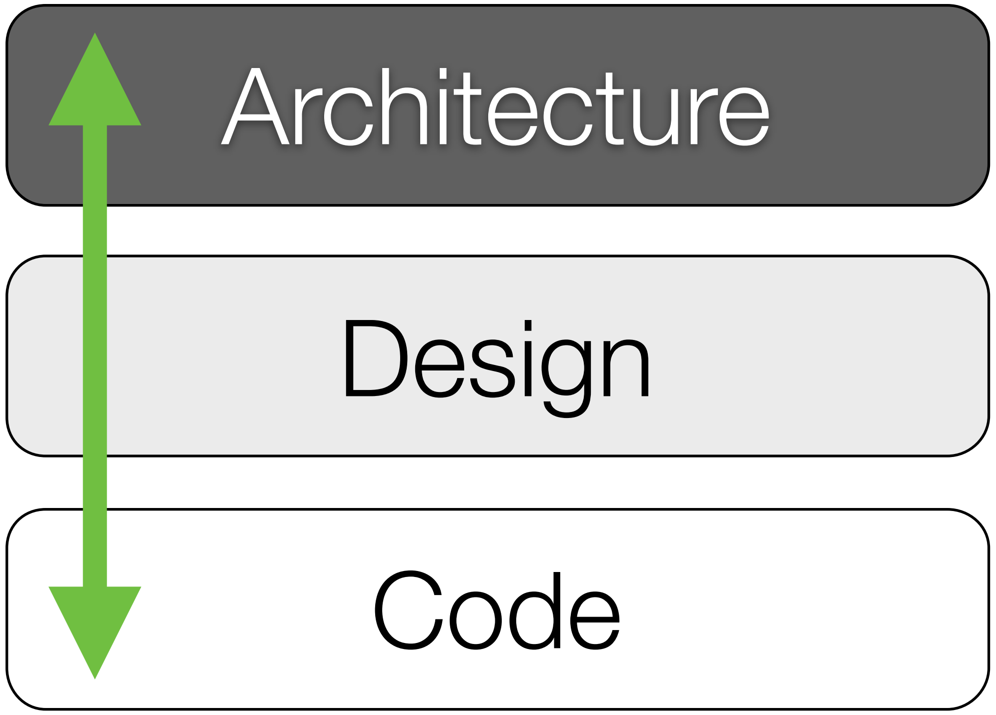
Architecture Recovery

[...] are the techniques and processes used to uncover a system's architecture from available information.

[Jazayeri]

[...] is an archaeological activity where the analysts must **unveil all the historical design decisions** by looking at the existing implementation and documentation of the system.

[Riva]

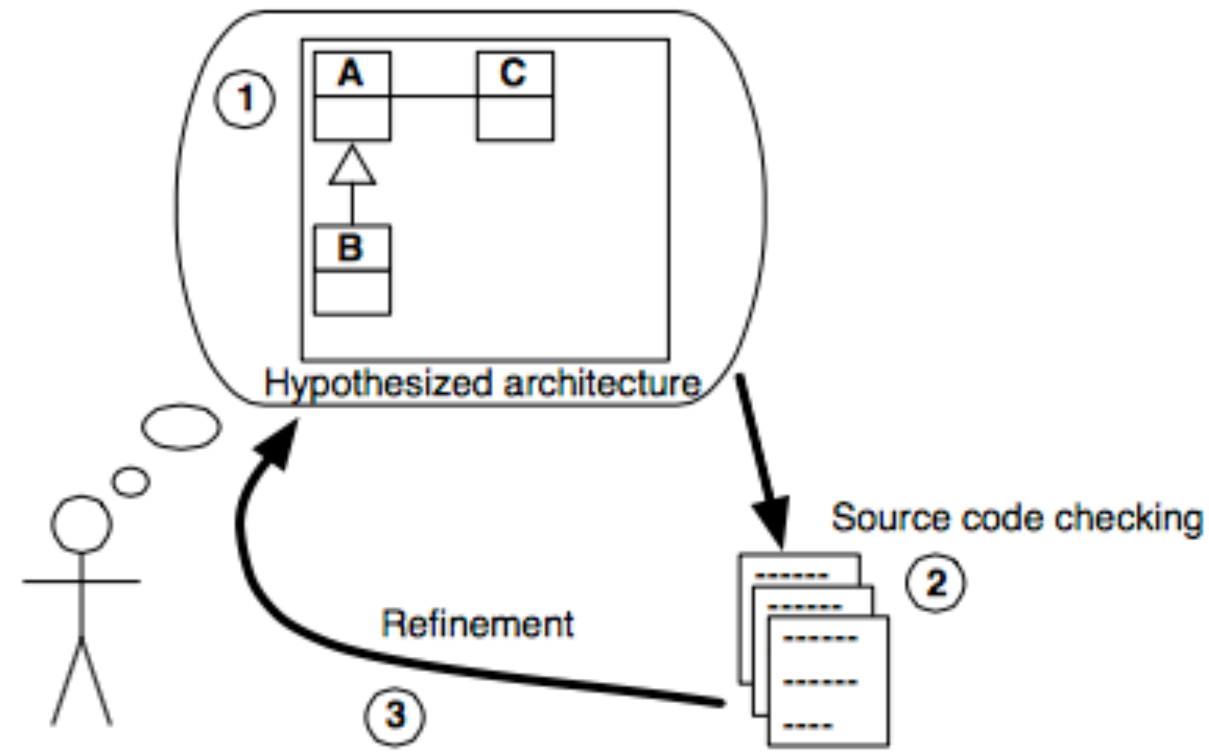


Roadmap

> Introduction to SAR
> The Architecture of Architecture Recovery
> **Top-down SAR**
  — Reflexion Models
> Bottom-up SAR
> Tool Demo

Top-Down SAR: Overview

Verifies whether the system conforms to the model the stakeholders have in mind



(1) an hypothesized architecture is defined,
(2) the architecture is checked against the src,
(3) the architecture is refined.

Roadmap

> Introduction to SAR
> The Architecture of Architecture Recovery
> **Top-down SAR**
  — **Reflexion Models**
> Bottom-up SAR
> Tool Demo

Software Reflexion Models

> A **reflexion model** indicates where the source model and high-level model differ
  - Convergences
  - Divergences
  - Absences

> Has to be interpreted by developer

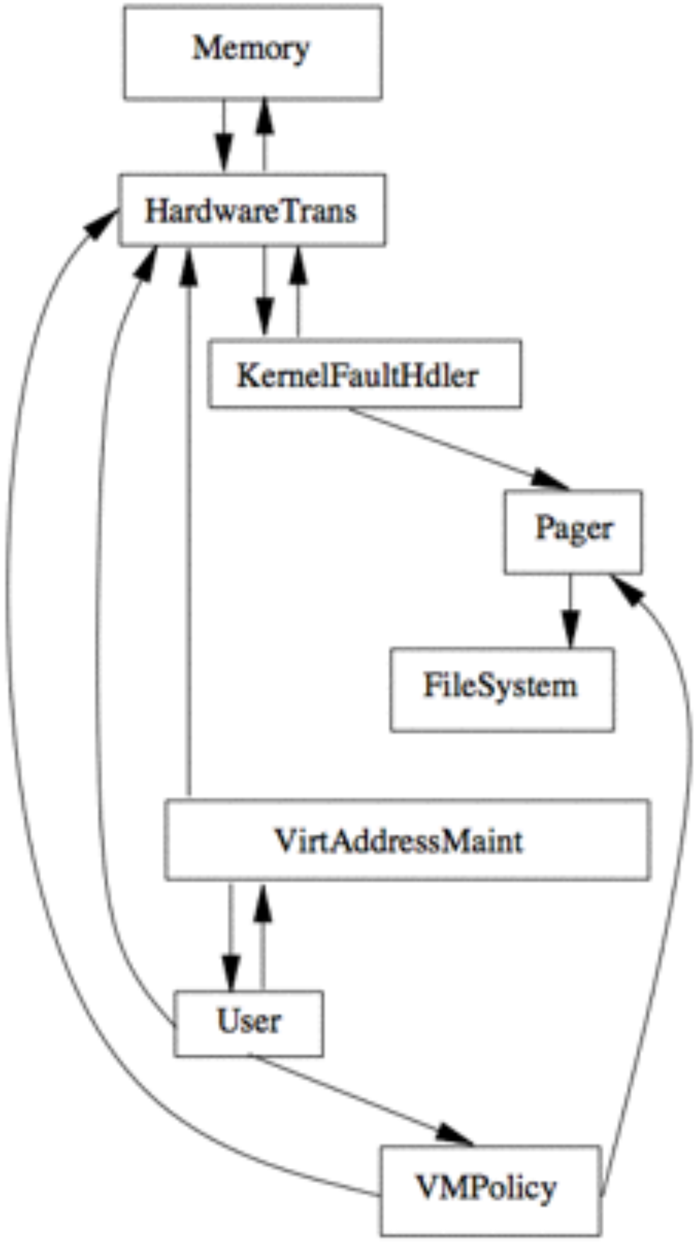
Reflexion modeling is iterative

```
Repeat
      * Define/Update high-level model of interest
      * Extract a source model
      * Define/Update declarative mapping between high-
      level model and source model
      * System computes a software reflexion model
      * Interpret the software reflexion model.
Until "happy"
```

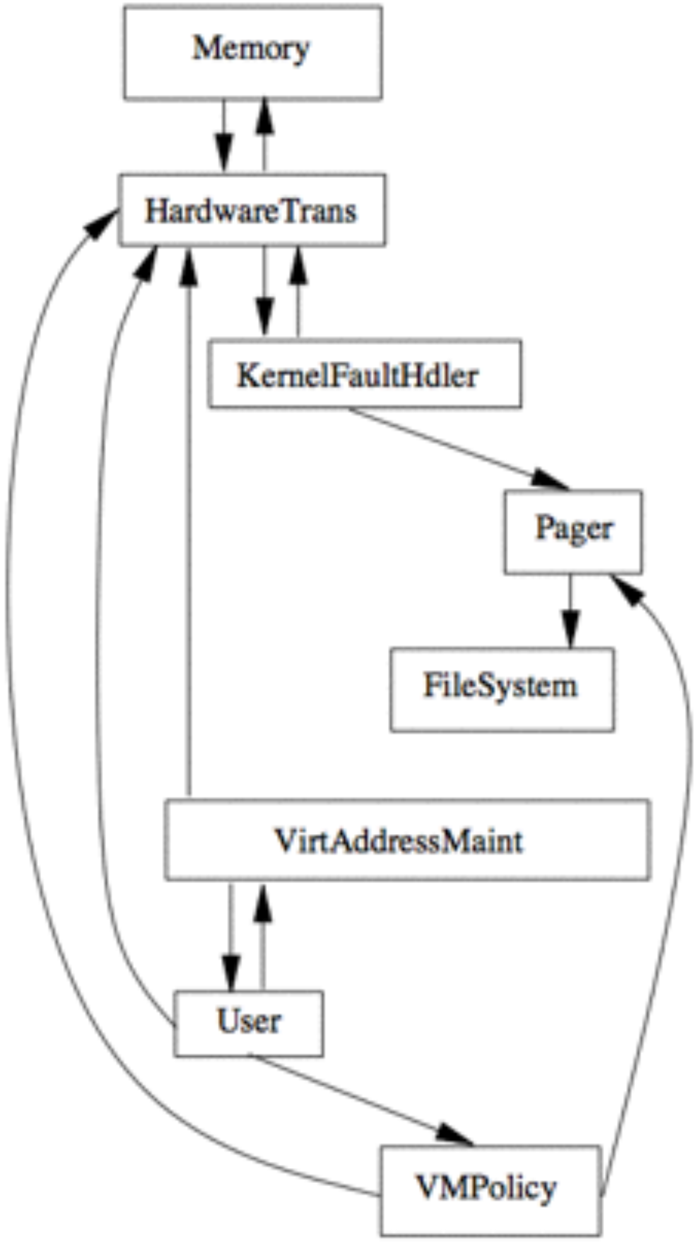
Case Study

The VMS of NetBSD

The High-level Model



The High-level Model



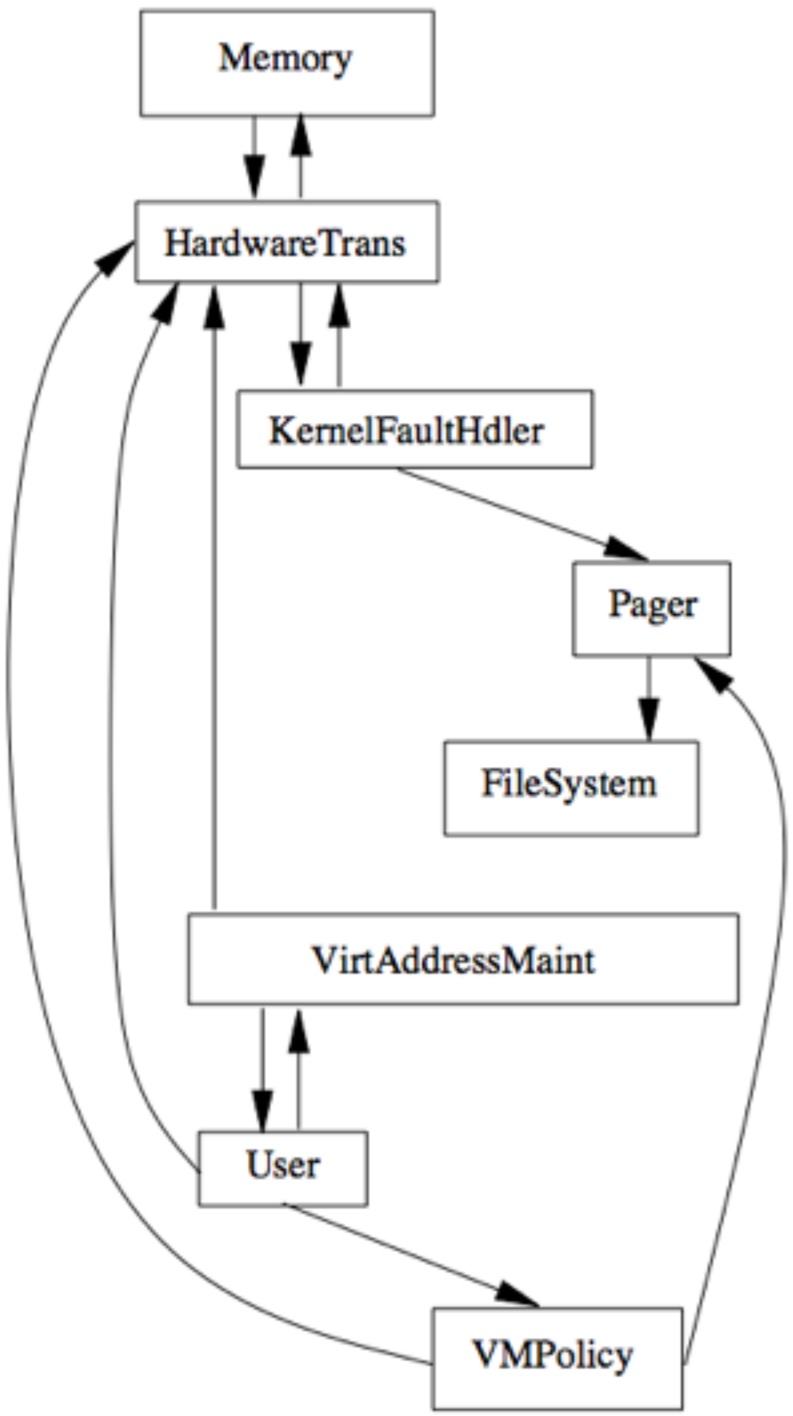
The Mapping

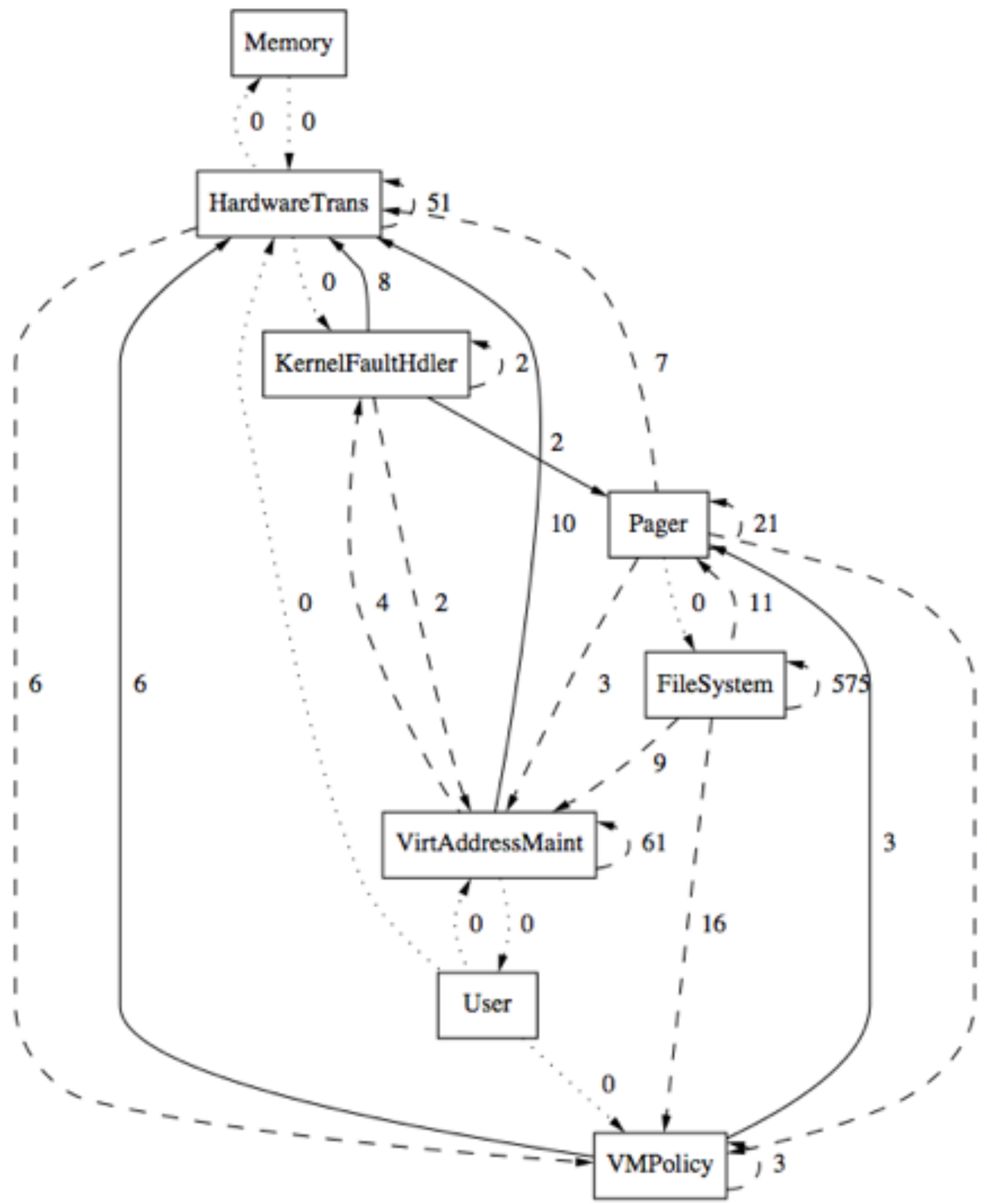
file= .*pager.*	mapTo=Pager
file= vm_map.*	mapTo=VirtAddressMaint
file=vm_fault\.c	mapTo=KernelFaultHandler
dir=[un]fs	mapTo=FileSystem
dir=sparc/mem.*]	mapTo=Memory
file=pmap.*	mapTo=HardwareTrans
file=vm_pageout\.c	mapTo=VMPolicy

Source Model

- Particular information extracted from source code
- Calculated with lightweight source extraction
  - Flexible: few constraints on source
  - Tolerant: source code can be incomplete, not compilable, …
- Lexical Approach

A Reflexion Model



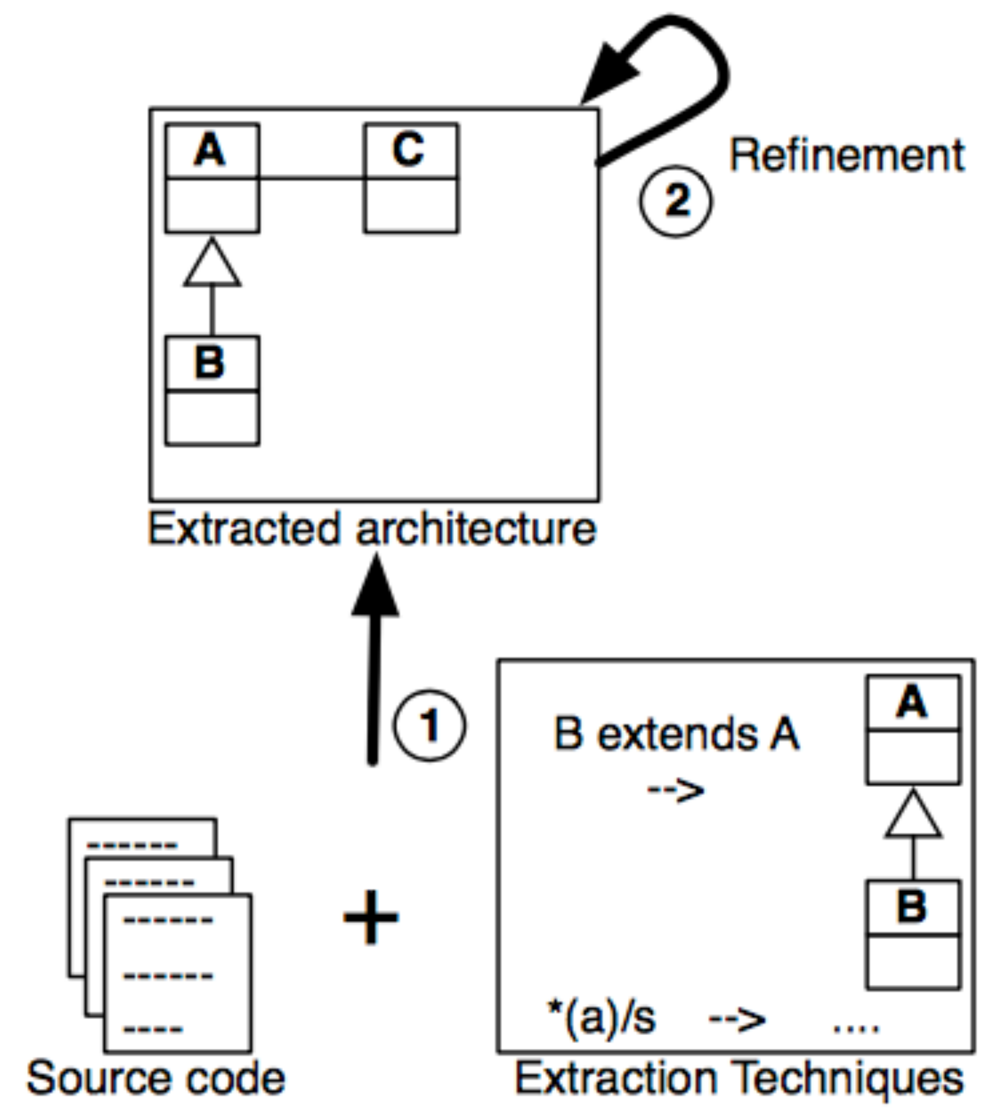


Roadmap

- > Introduction to SAR
- > The Architecture of Architecture Recovery
- > Top-down SAR
- > **Bottom-up SAR**
  - — Data Extraction
  - — Knowledge Organization
  - — Analysis & Exploration
- > Tool Demo

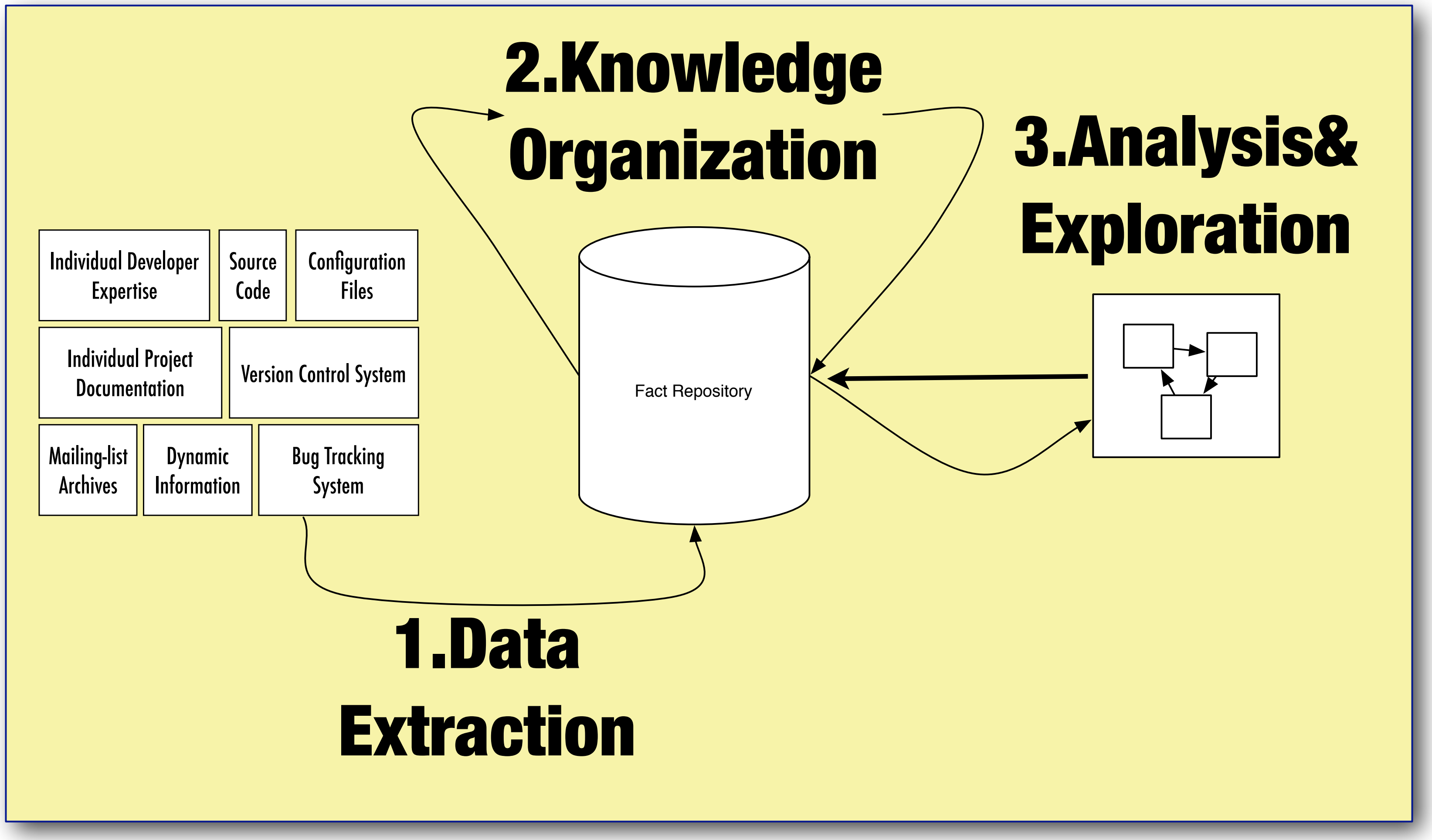
Bottom-Up SAR: Overview

Starts without any assumptions about the code and tries to recover the architecture *as-is*



(1) views are extracted from src
(2) view are refined

The Architecture of Architecture Recovery

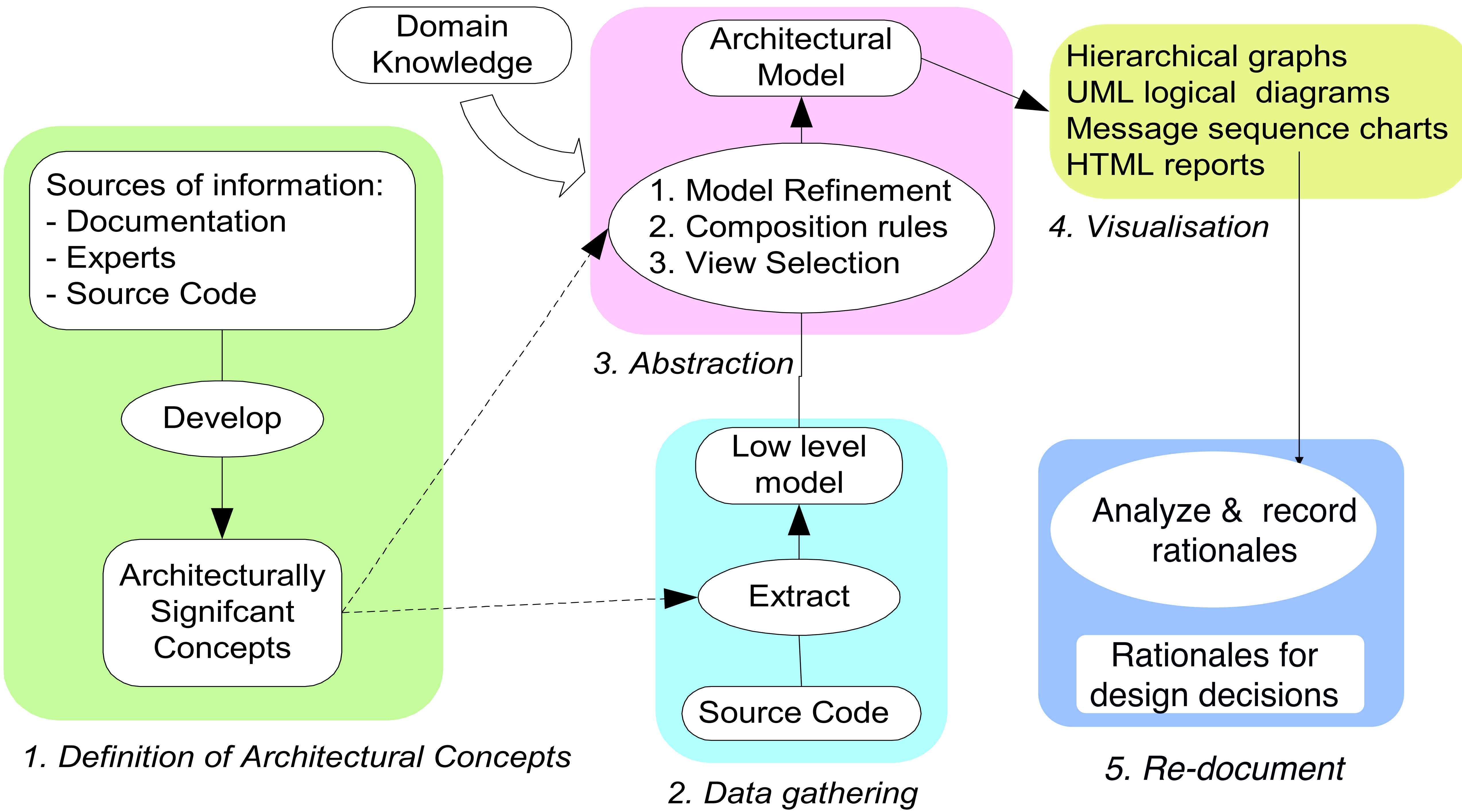


“extract-abstract-present” [Tilley]

Roadmap

- Introduction to SAR
- The Architecture of Architecture Recovery
- Top-down SAR
- **Bottom-up SAR**
  - **Data Extraction**
  - Knowledge Organization
  - Analysis & Exploration
- Tool Demo

Architecture Reconstruction



1. Data Extraction - Tools

	src	text	dyn	phys	hist	stk	style
Alborz [110]	x		x			x	
ArchView [99]	x		x		x	x	
ArchVis [45]	x	x	x	x			x
ARES [26]	x					x	
ARM [40]	x					x	
ARMIN [58]	x					x	
ART [32]	x					x	x
Bauhaus [13, 25, 62]	x		x			x	
Bunch [79, 90]	x					x	
Cacophony [28]						x	
Dali [56, 57]	x					x	
DiscoTect [146]	x		x			x	x
Focus [18, 84]	x					x	x
Gupro [24]	x					x	
Intensive [87, 145]	x					x	
ManSART [4, 43]	x			x		x	x
MAP [117]	x					x	x
PBS/SBS [8, 31, 49, 113]	x			x		x	
PuLSE/SAVE [61, 103]	x					x	
QADSAR [118, 119]	x					x	
Revealer [100, 101]	x	x				x	
RMTool [92, 93]	x					x	
SARTool [30, 64]	x					x	
SAVE [89, 94]	x					x	
Softwarenaut [77]	x	x		x	x	x	
Symphony,Nimeta [106, 135]			x			x	
URCA			x			x	
W4 [44]	x				x	x	
X-Ray [86]	x				x	x	x

src - source code
text - textual information
dyn - dynamic analysis
phys - physical organiation
stk - human expertise / stakeholder
style - architectural style

Roadmap

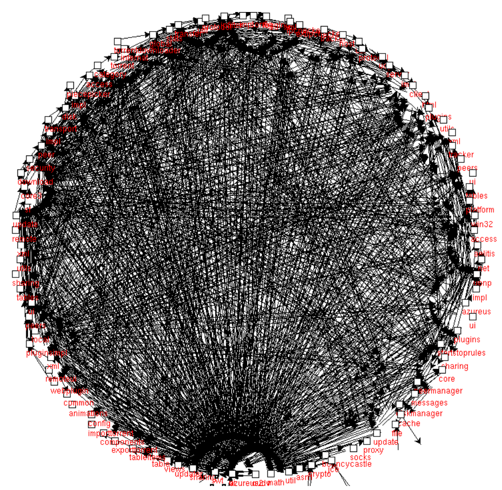
> Introduction to SAR
> The Architecture of Architecture Recovery Tools
> Top-down SAR
> **Bottom-up SAR**
  — Data Extraction
  — **Knowledge Organization**
  — Analysis & Exploration
> Tool Demo

Knowledge Organization

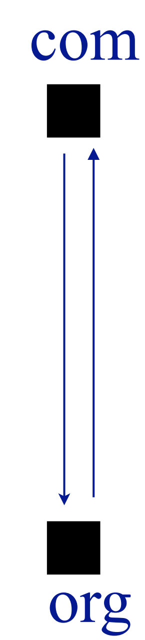
- Different techniques
  a) Aggregation
  b) Clustering
  c) Concept Analysis

a. Aggregation

Package
Dependencies

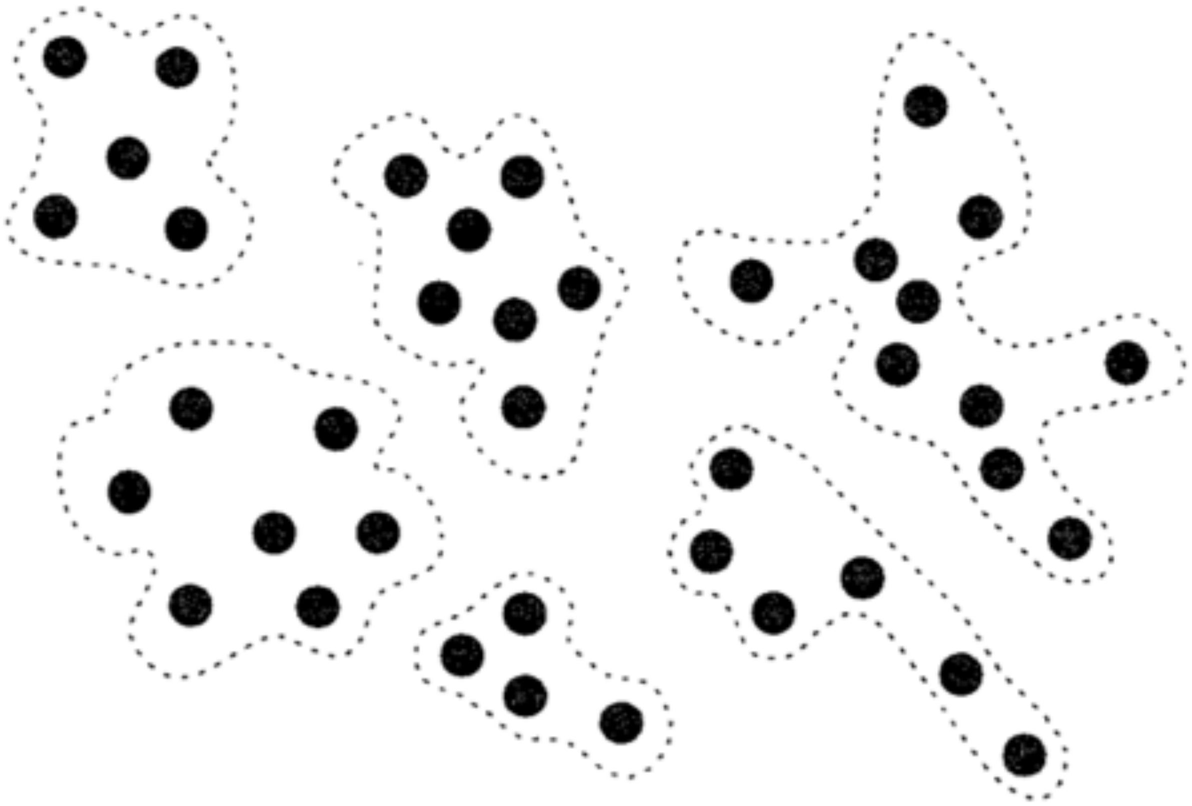


Highest-Level
Dependency View



Hierarchical Graph Data Structure

b. Clustering



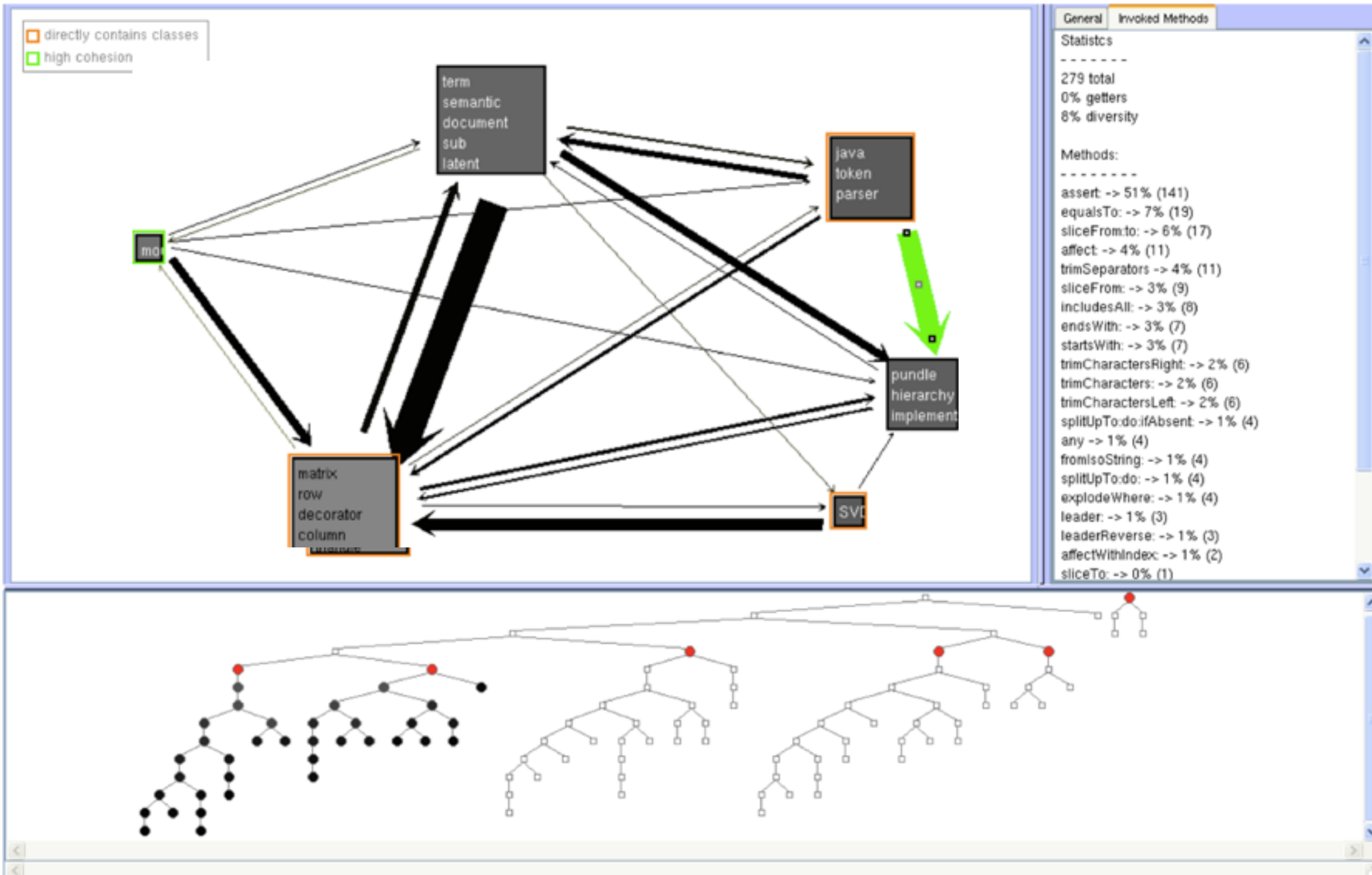
> Concepts
  - Entities
  - Similarity Metric
  - Algorithms

> Solutions: Hapax, Bunch

Similarity Metric

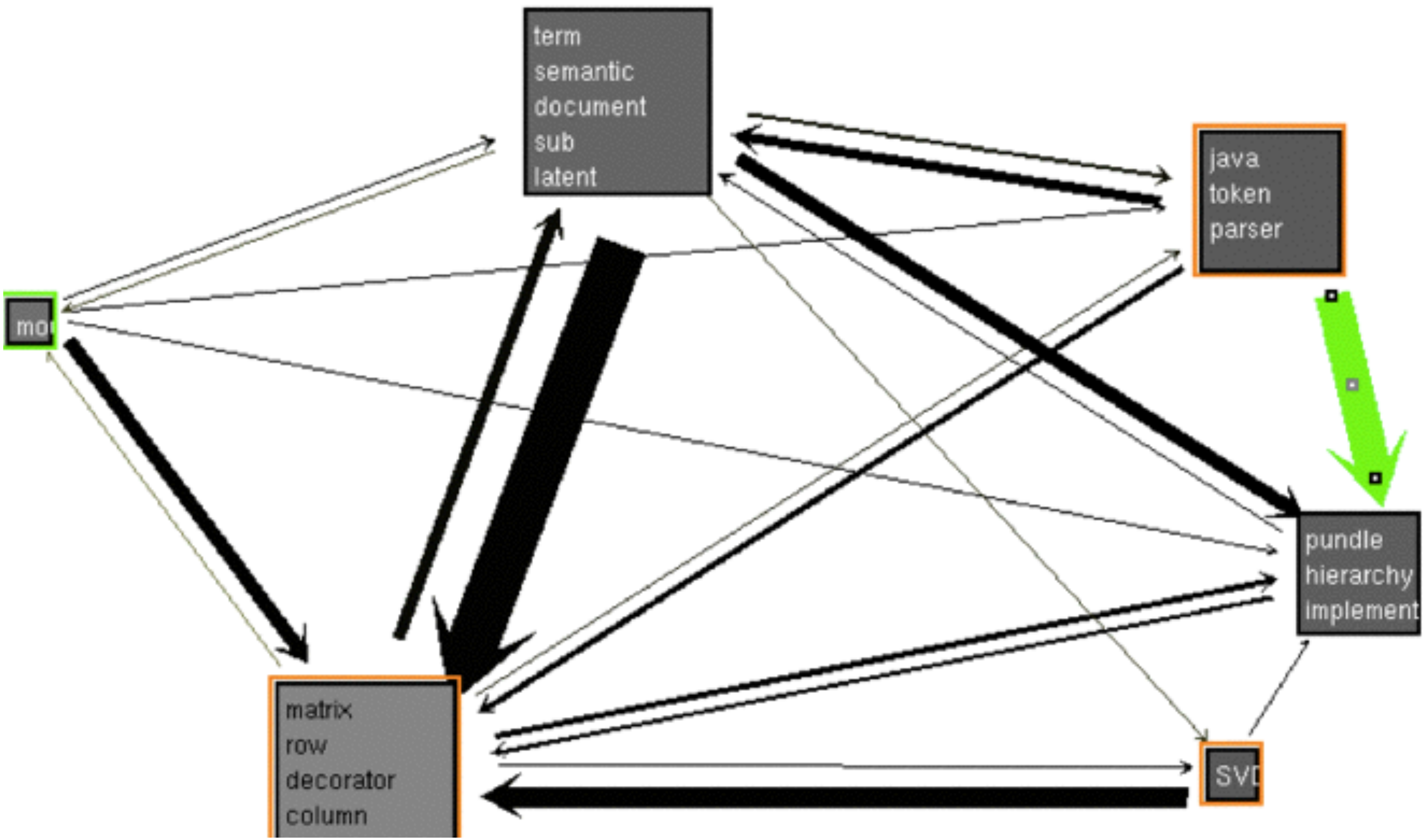
- Based on **relationships** between the elements or common **properties**
  - relationships (e.g. invocations)
  - natural language similarity
  - …

Similarity Metric: (natural) language



[Lungu et al.'05]

Similarity Metric: (natural) language



[Lungu et al.'05]

Similarity Metric: Arch

> Arch [Schwanke]

— similarity between procedures:
  - *number of common features (non-local symbols used in procedures)*
  - *feature weight*
  - *interactions*

$$Sim(A,B) = \frac{W(a \cap b) + k \times Linked(S,B)}{n + W(a \cap b) + d \times (W(a-b) + W(b-a))}$$

Algorithms

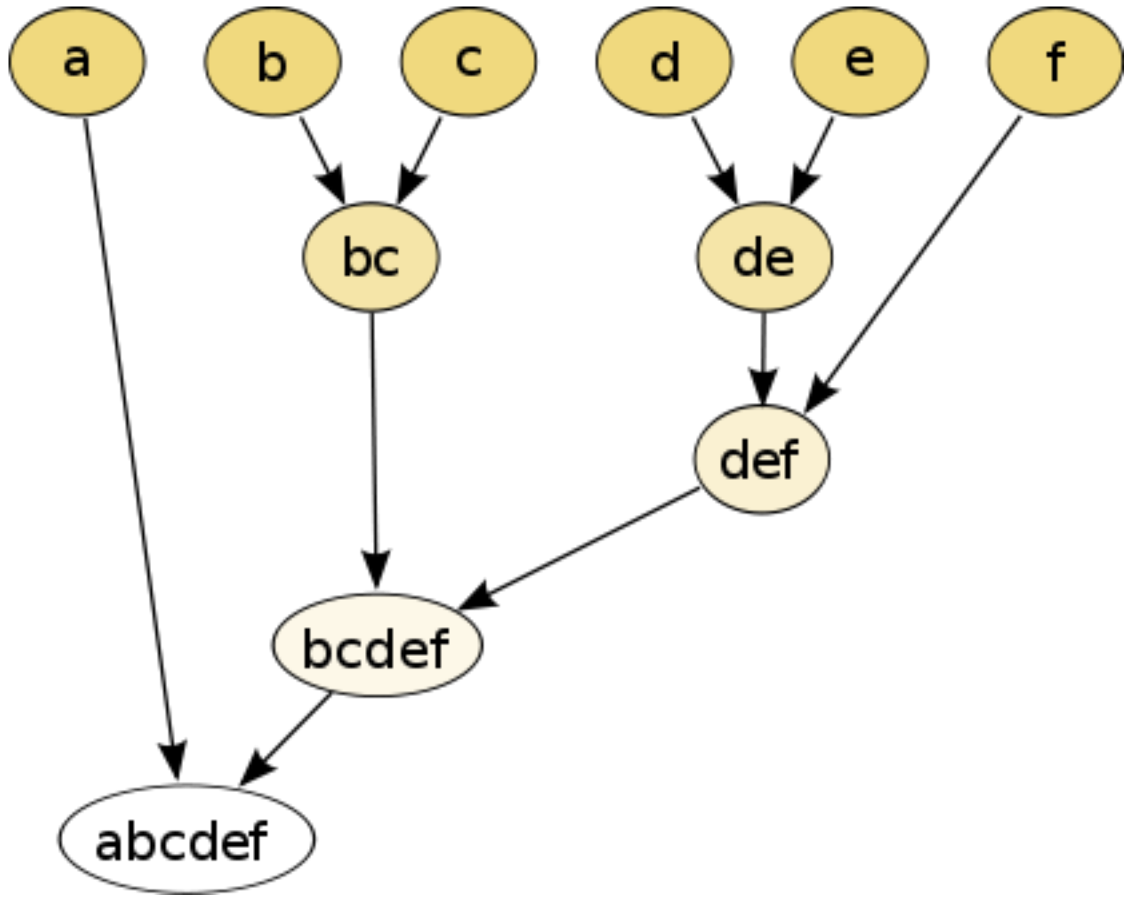
Flat

place each entity in a group by itself
repeat
  identify the *two most similar groups*
  combine them
until the existing groups are satisfactory

Hierarchical

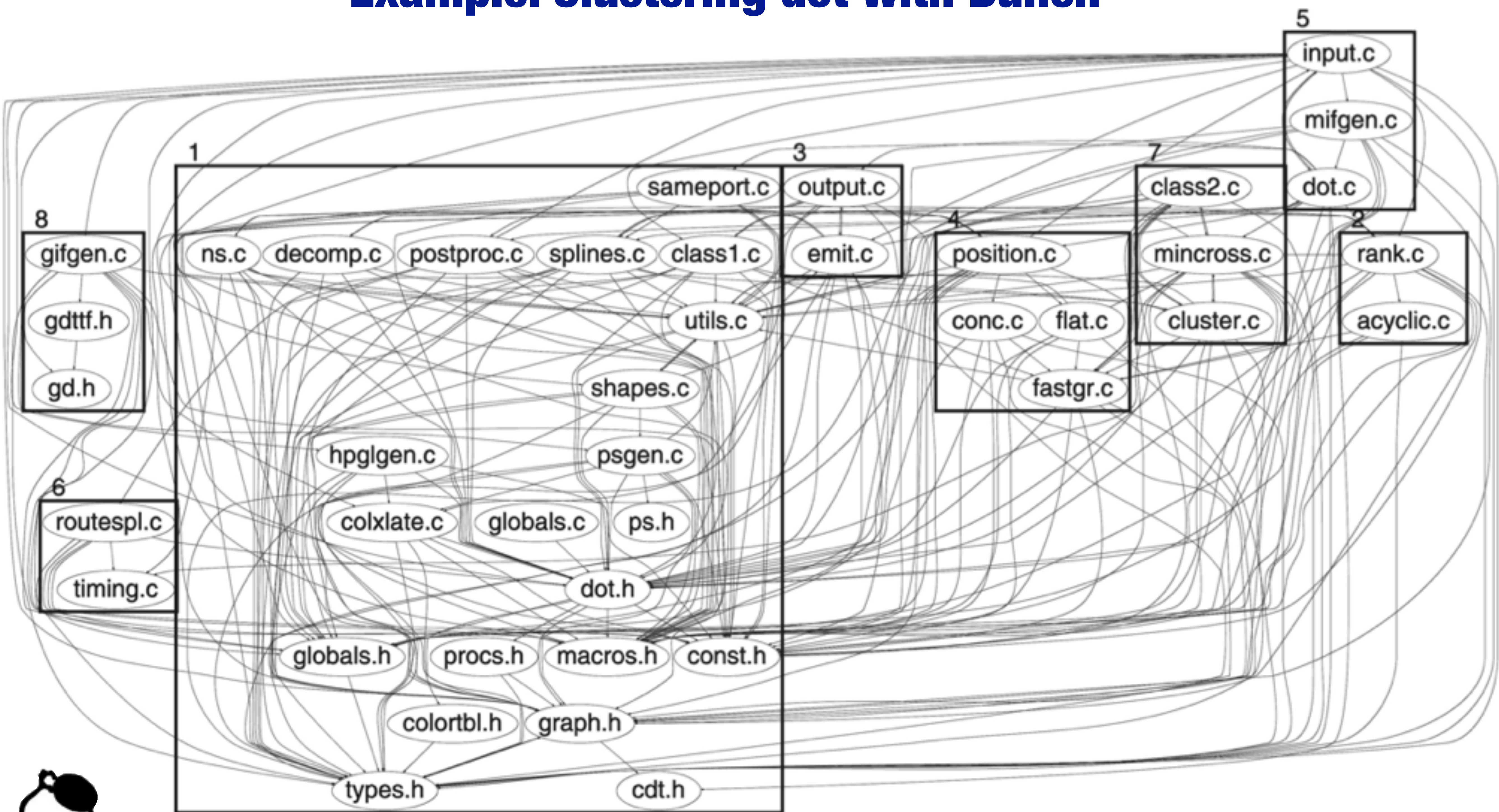
place each entity in a group by itself
repeat
  identify *the most similar groups* Si and Sj
  combine Si and Sj
  add a subtree with children Si and Sj to the clustering tree
until the existing groups are satisfactory or only one group is left

Result of Hierarchical Clustering

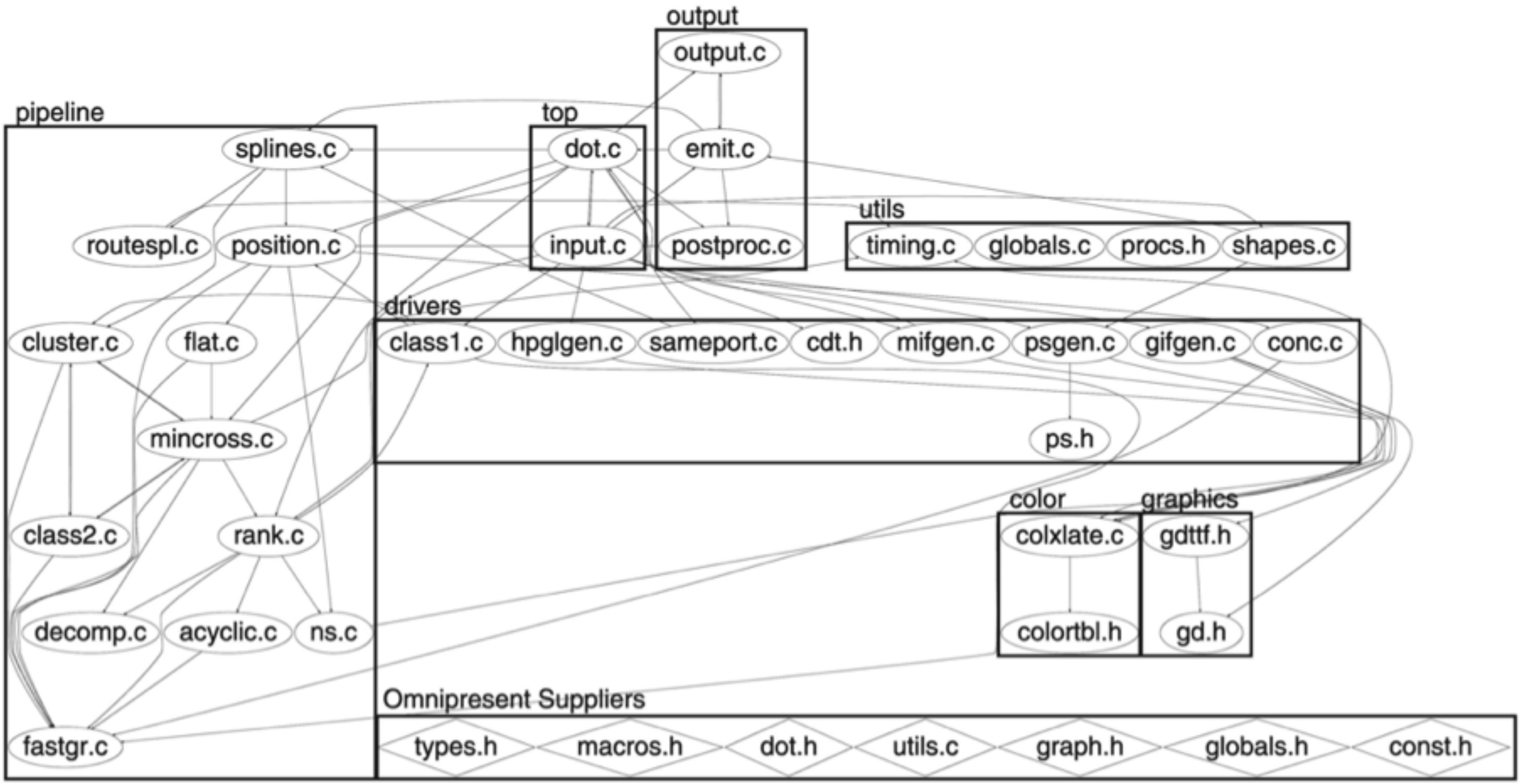


A Dendrogram: How do you select the cutoff factor?

Example: Clustering dot with Bunch

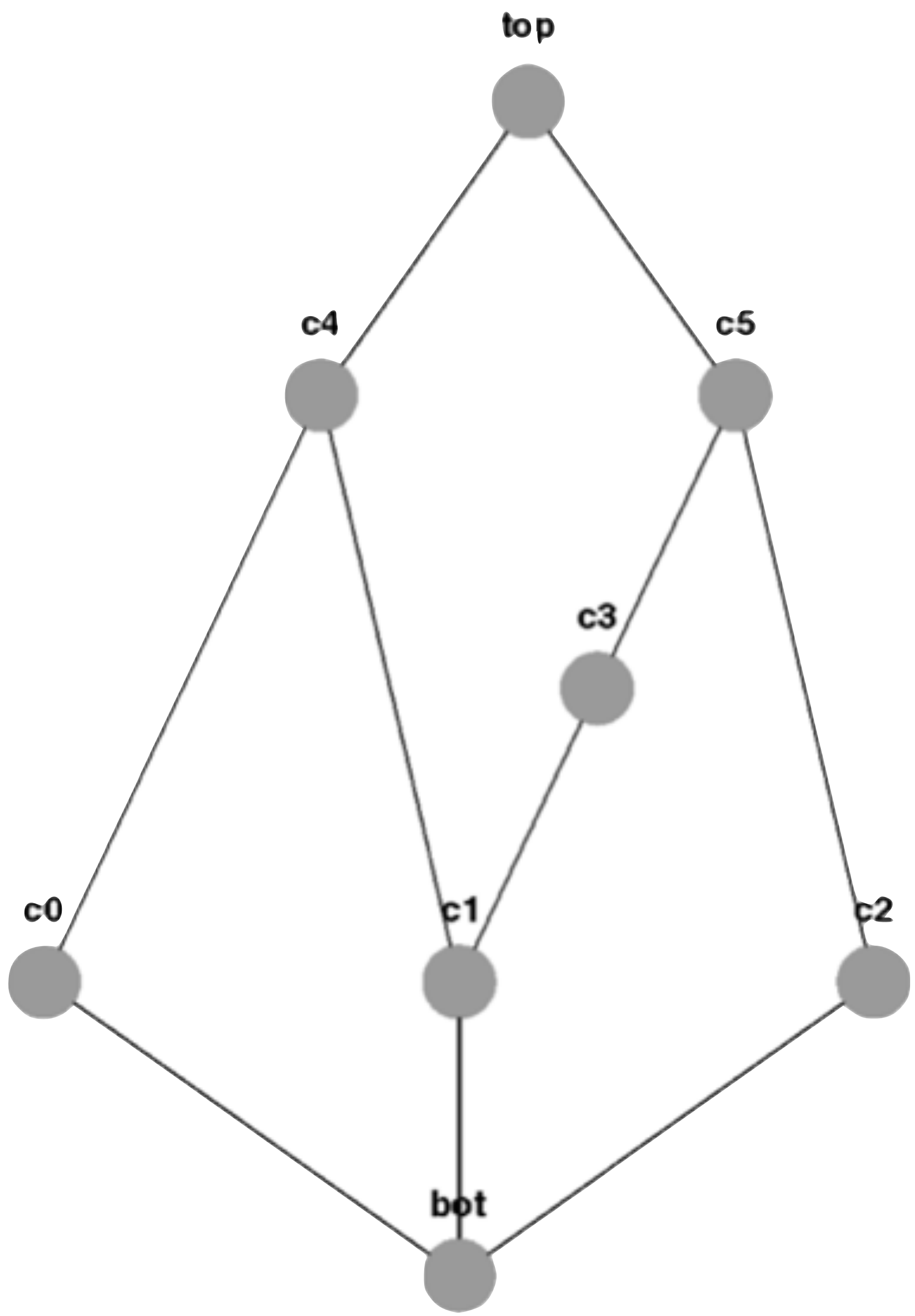


Clustering dot with Bunch



c. Formal Concept Analysis

- Identify meaningful groupings of elements that have common properties
- Concept: (objs, props)
  - props(obj) includes props
  - obj_with(props) == objs

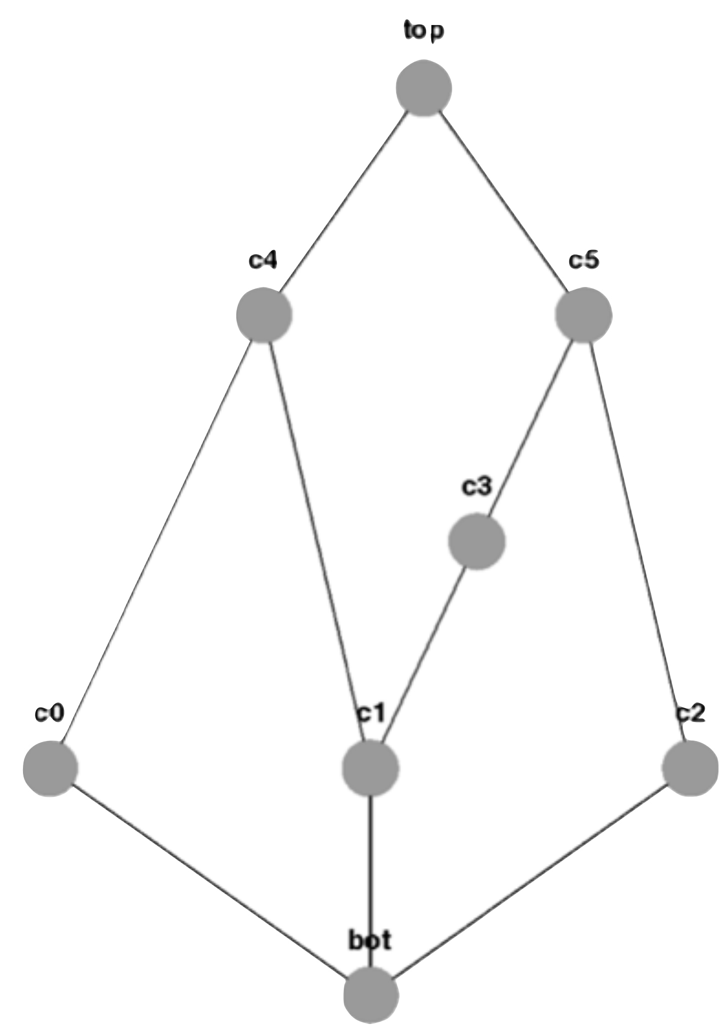


A Concept Analysis Example

— props(obj) includes props
— obj_with(props) == objs

		attributes				
		four-legged	hair-covered	intelligent	marine	thumbed
objects	cats	√	√			
	dogs	√	√			
	dolphins			√	√	
	gibbons		√	√		√
	humans			√		√
	whales			√	√	

top	({cats, gibbons, dogs, dolphins, humans, whales}, ∅)
c_5	({gibbons, dolphins, humans, whales}, {intelligent})
c_4	({cats, gibbons, dogs}, {hair-covered})
c_3	({gibbons, humans}, {intelligent, thumbed})
c_2	({dolphins, whales}, {intelligent, marine})
c_1	({gibbons}, {hair-covered, intelligent, thumbed})
c_0	({cats, dogs}, {hair-covered, four-legged})
bot	(∅, {four-legged, hair-covered, intelligent, marine, thumbed})



The Concept Lattice

A Concept Analysis Problem

```
#define QUEUE_SIZE 10
struct stack { int *base, *sp, size; };
struct queue { struct stack *front, *back; };

struct stack* initStack(int sz) {
 struct stack* s =
   (struct stack*) malloc(sizeof(struct stack));
 s->sp = (int*)malloc(sz * (sizeof(int)));
 s->base = s->sp;
 s->size = sz;
 return s; }

struct queue* initQ() {
 struct queue* q =
   (struct queue*) malloc(sizeof(struct queue));
 q->front = initStack(QUEUE_SIZE);
 q->back = initStack(QUEUE_SIZE);
 return q; }

int isEmptyS(struct stack* s) {
 return (s->sp == s->base); }

int isEmptyQ(struct queue* q) {
 return (isEmptyS(q->front)
           && isEmptyS(q->back)); }
```

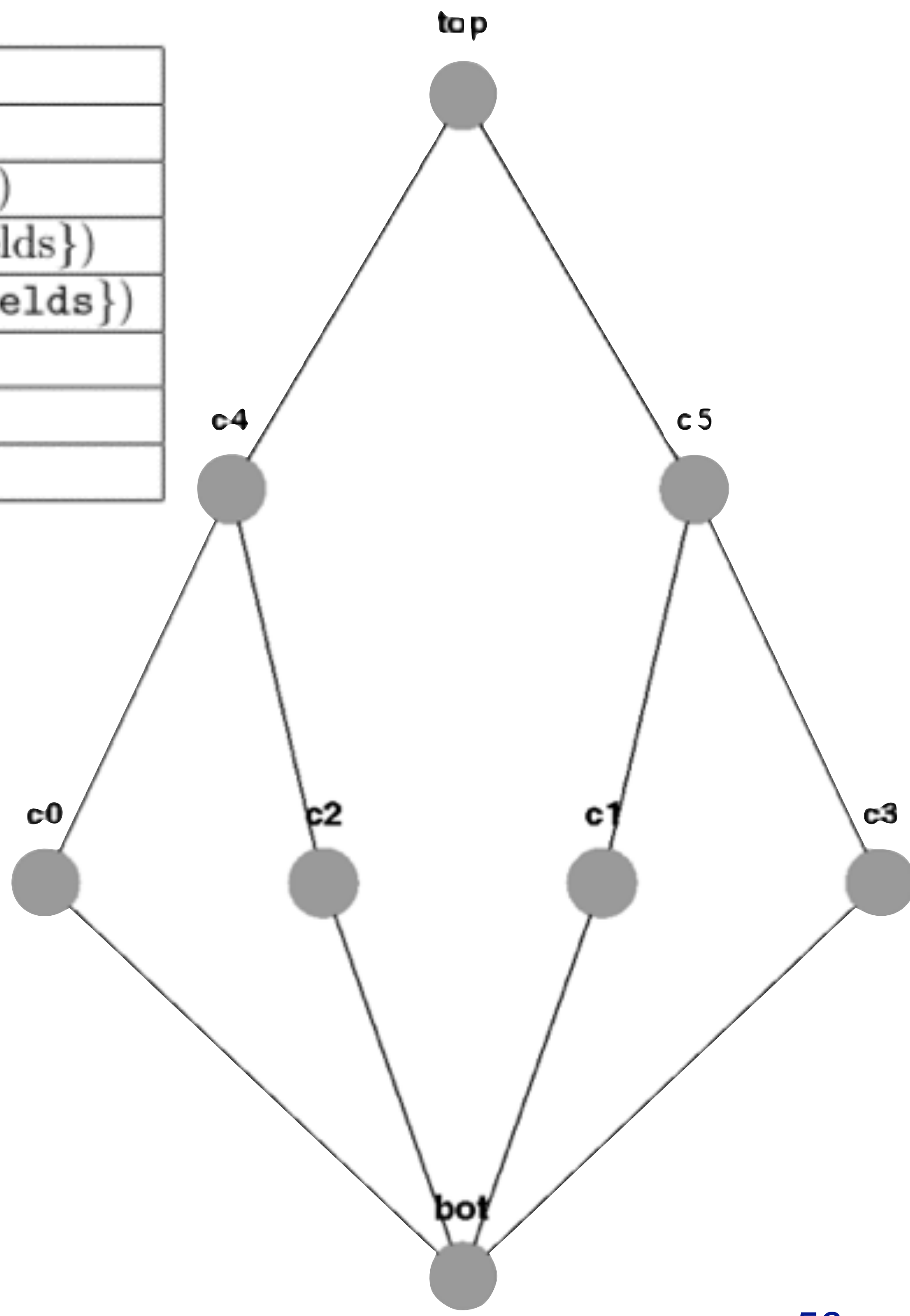
```
void push(struct stack* s, int i) {
 /* no overflow check */
 *(s->sp) = i; s->sp++; }

void enq(struct queue* q, int i) {
 push(q->front, i); }

int pop(struct stack* s) {
 if (isEmptyS(s)) return -1;
 s->sp--;
 return (*(s->sp)); }

int deq(struct queue* q) {
 if (isEmptyQ(q)) return -1;
 if (isEmptyS(q->back))
  while(!isEmptyS(q->front))
   push(q->back, pop(q->front));
 return pop(q->back); }
```

A Concept Analysis Problem

	returns `stack`	*returns* `queue`	*has* `stack` *arg.*	*has* `queue` *arg.*	*uses* `stack` *fields*	*uses* `queue` *fields*
`initStack`	√				√	
`initQ`		√				√
`isEmptyS`			√		√	
`isEmptyQ`				√		√
`push`			√		√	
`enq`				√		√
`pop`			√		√	
`deq`				√		√

A Concept Analysis Problem

top	(*all objects*, $\emptyset$)
c_5	({initQ, isEmptyQ, enq, deq}, {uses queue fields})
c_4	({initStack, isEmptyS, push, pop}, {uses stack fields})
c_3	({isEmptyQ, enq, deq}, {has queue argument, uses queue fields})
c_2	({isEmptyS, push, pop}, {has stack argument, uses stack fields})
c_1	({initQ}, {returns queue})
c_0	({initStack}, {returns stack})
bot	($\emptyset$, *all attributes*)



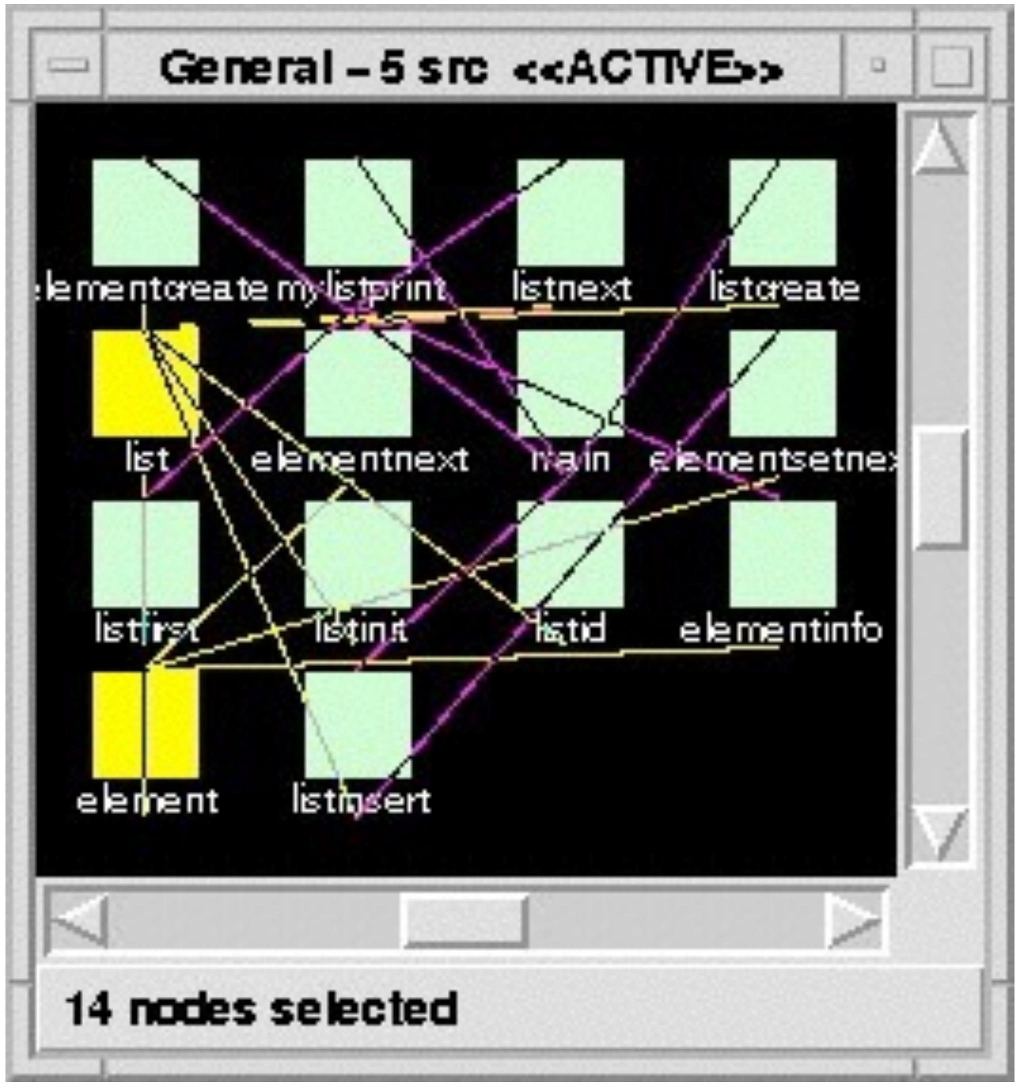
Roadmap

- Introduction to SAR
- The Architecture of Architecture Recovery
- Top-down SAR
- **Bottom-up SAR**
  - Data Extraction
  - Knowledge Organization
  - **Analysis & Exploration**
- Tool Demo

3. Analysis & exploration - Rigi

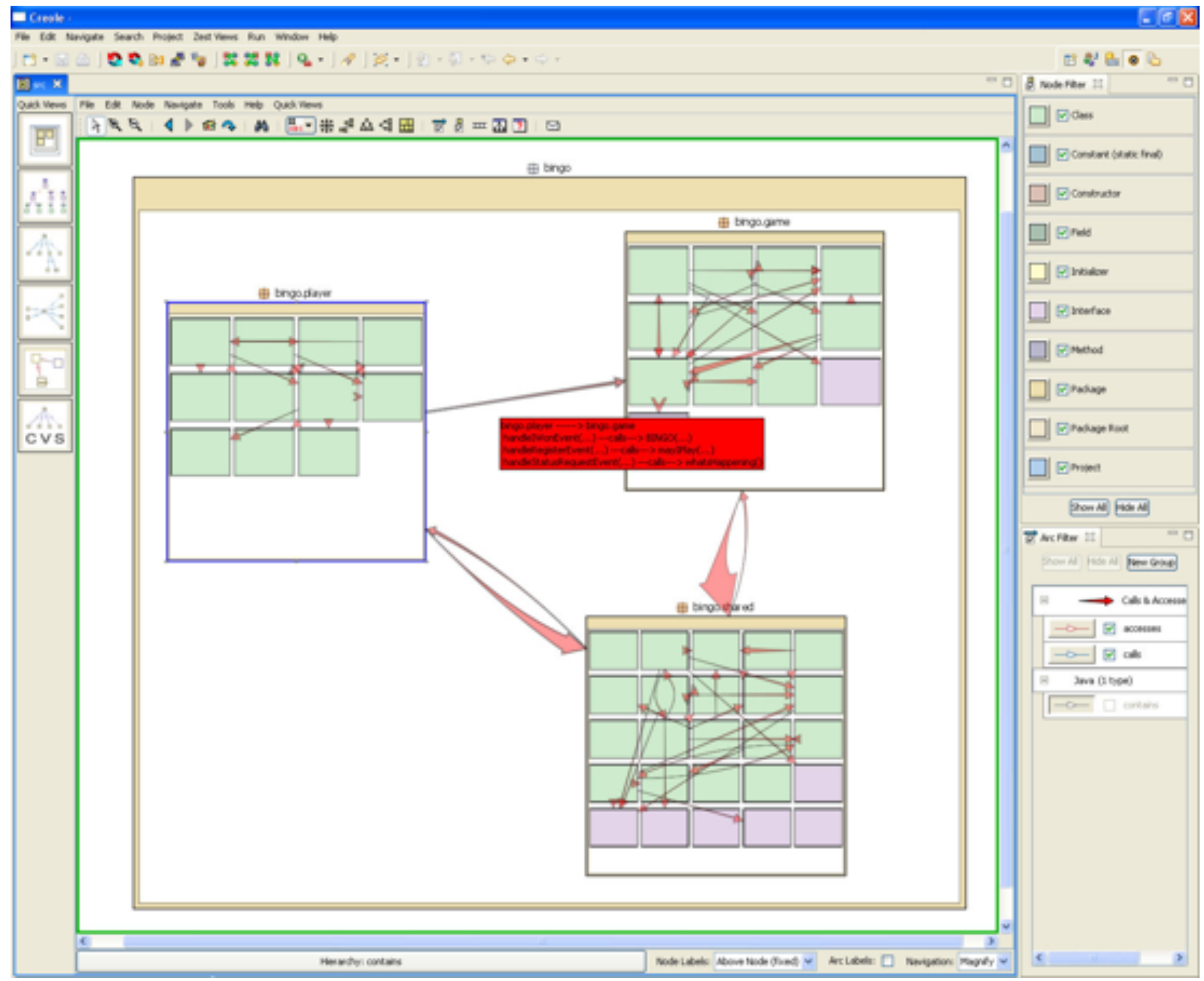
Programmable reverse engineering environment

- C parser; relational data import
- Visualization of hierarchical typed graphs
- Graph manipulation, filtering, layout
- Tcl-programmable
- www.rigi.csc.uvic.ca/



3. Analysis & exploration - Creole

> Eclipse Integration
> Semantic Zooming
> Simple Aggregation

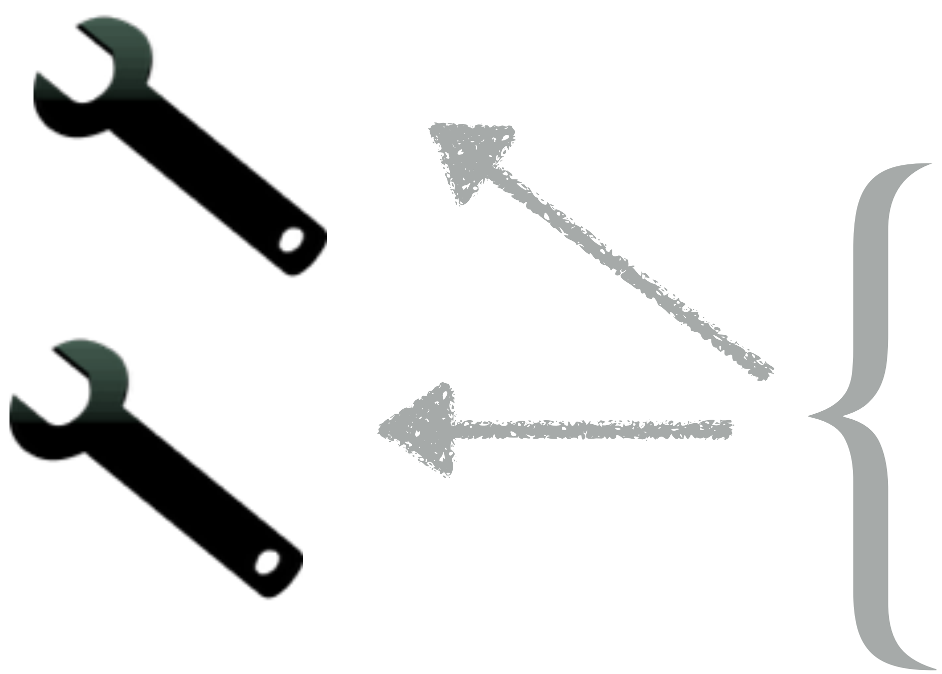


http://thechiselgroup.org/2003/07/06/creole/

Roadmap

- Introduction to SAR
- The Architecture of Architecture Recovery
- Top-down SAR
- Bottom-up SAR
- **Tool Demo**

Dicto (Top-down)

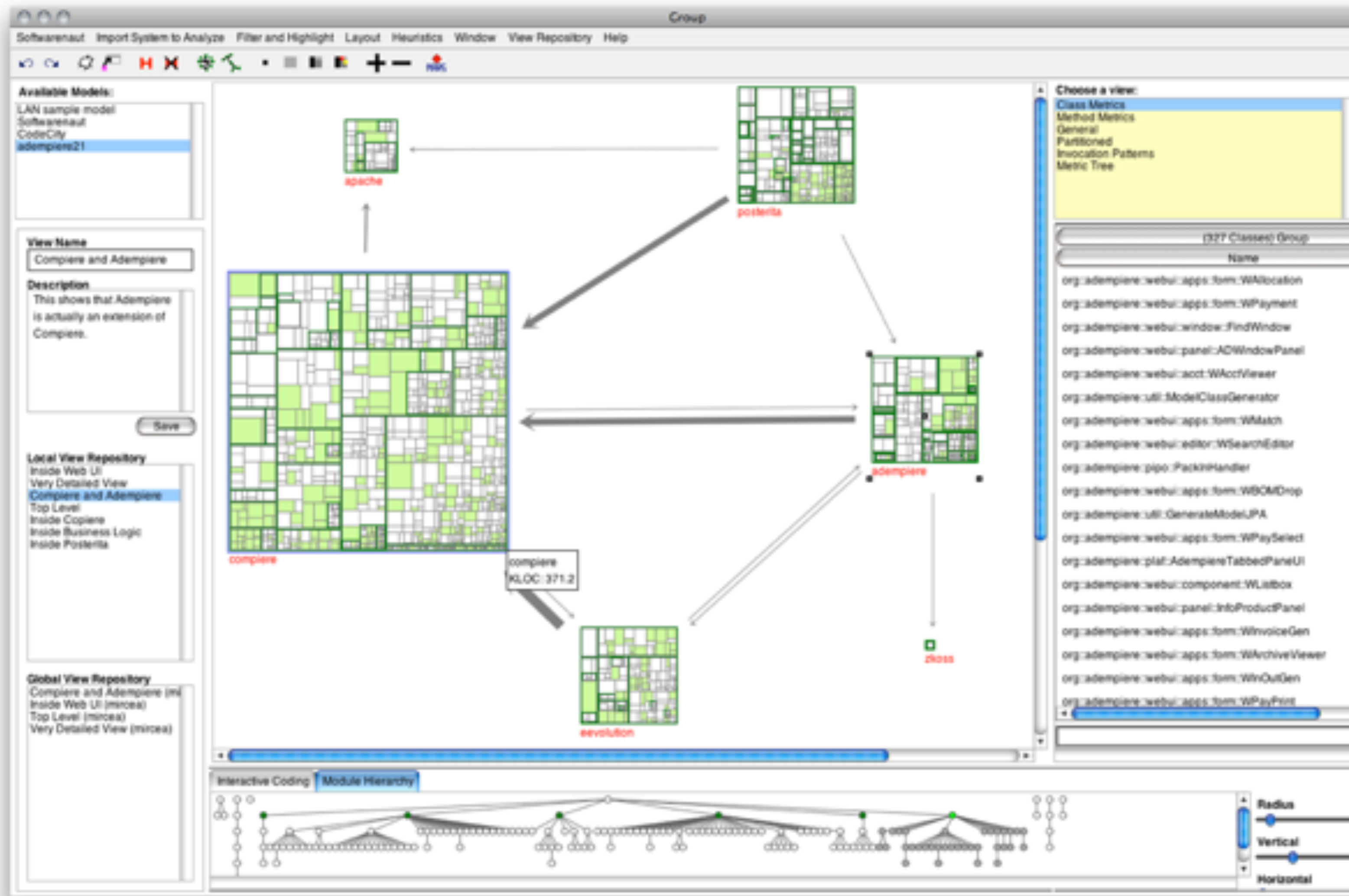


```
// Dependencies
Syntax: Package with name="org.app.Syntax"
Core: Package with name="org.app.Core"
Parser: Package with name="org.app.Parser"

Parser can only depend on Syntax
Core, Syntax cannot depend on Parser

// Performance
Google: Website with url="http://www.google.com"

Google must handle load from "10 users"
Google must have latency < "100 ms"
```

A uniform notation for keeping SA under control

http://scg.unibe.ch/dicto/

SoftwareNaut (Bottom-up)



- Based on FAMIX
- Hierarchical Graphs
- Collaboration & Sharing

http://scg.unibe.ch/softwarenaut

What you should know!

- Architecture, Architectural styles, Architectural viewpoints
- What is architecture recovery
- The two main types of architecture recovery processes
- How clustering software artefacts works
- How concept analysis works

Can you answer these questions?

- What is formal concept analysis and how can you use it in architecture recovery?
- How would you cluster the classes in an object-oriented software system if you want to discover its architecture?
- What are the limitations of top-down AR? Of bottom-up?
- What are Mavericks in Schwanke’s approach?
- What are the limitations of clustering?
- What are the limitations of concept analysis?

Further Reading

An intelligent tool for re-engineering software modularity, Schwanke R.

Software Reflexion Models: Bridging the gap between Source and High-Level Models, Murphy et al.

Identifying Modules via Concept Analysis, Siff and Reps

Constructive Architecture Compliance Checking -- An Experiment on Support by Live Feedback, Knodel et al.

Maintaining Hierarchical Graph Views, Bauchsbaum et al.

Evolutionary and Collaborative Software Architecture Recovery With Softwarenaut, Lungu et al.

Towards A Process-Oriented Software Architecture Reconstruction Taxonomy, Pollet et al.