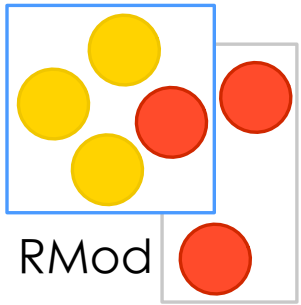


RMOD

Dr. Stéphane Ducasse
stephane.ducasse@inria.fr
http://stephane.ducasse.free.fr/

A word of presentation

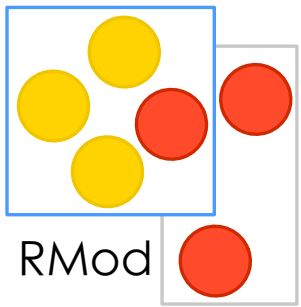
Since 1996 Moose (reengineering platform)

Object-Oriented Reengineering Patterns

Grounded in reality
Maintainer of open-source projects

Worked with:
- Harman-Becker AG
- Bedag AG,
- Nokia, Daimler

RMOD expertise



Supporting software evolution and software composition

Axis 1: Reengineering

Maintaining large software systems

Moose: a powerful platform for reengineering

Nokia, Daimler, Harman-Becker, Siemens, Cincom

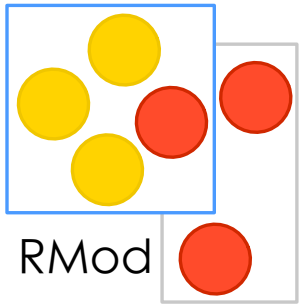
Axis 2: Dynamic languages to support evolution

Revisiting fundamental aspects of OO languages

Reuse Traits: Fortress (SUN Microsystems), Perl-6, Scala (EPFL), Squeak, Dr-Scheme,

Security and Dynamic Languages

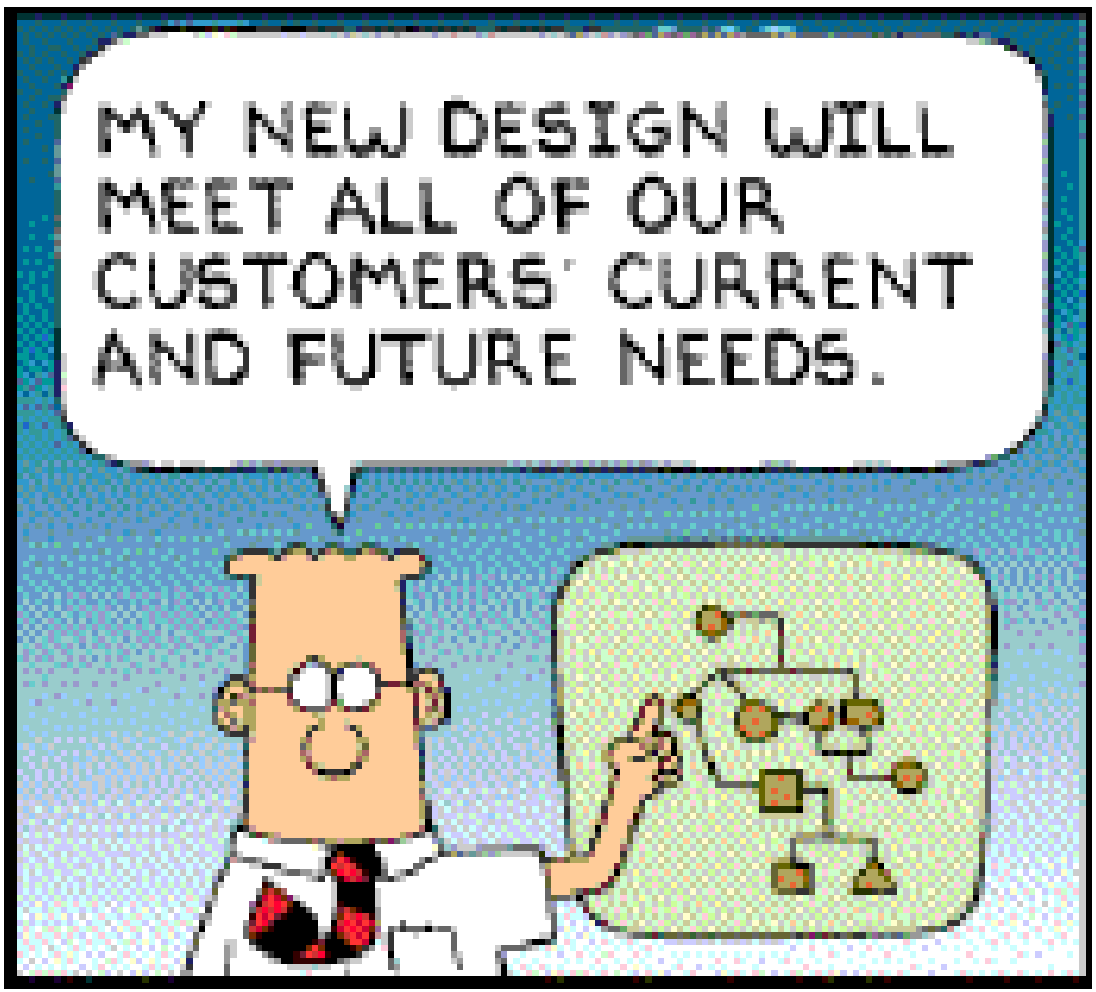
Axis 1: Reengineering



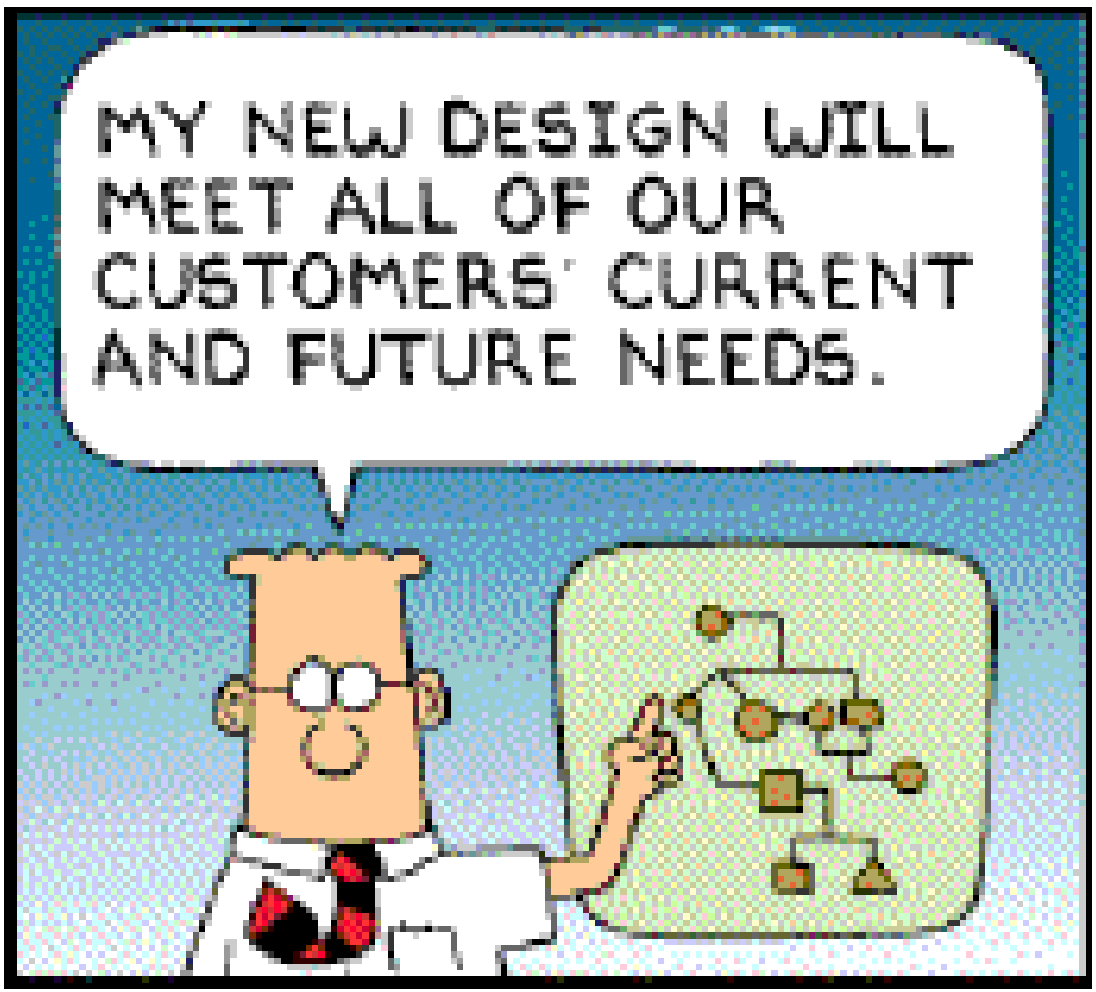
Maintaining large software systems

Moose: a powerful platform for reengineering

Nokia, Daimler, Harman-Becker, Siemens, Cincom


MY NEW DESIGN WILL MEET ALL OF OUR CUSTOMERS' CURRENT AND FUTURE NEEDS.

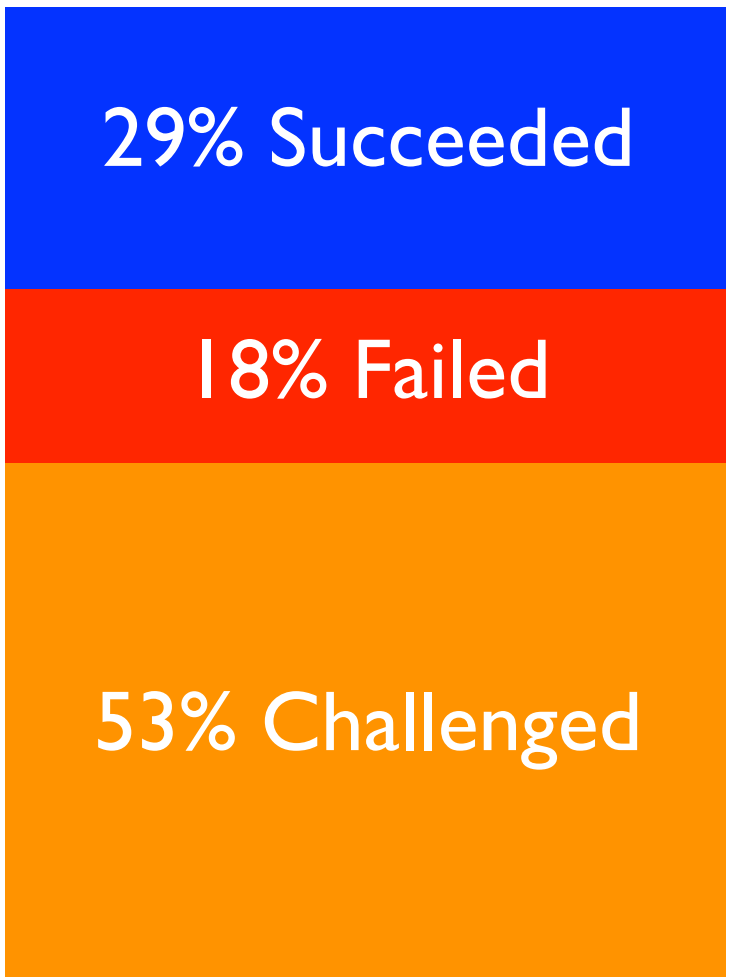
Let's face it, this is the Graal



Roadmap

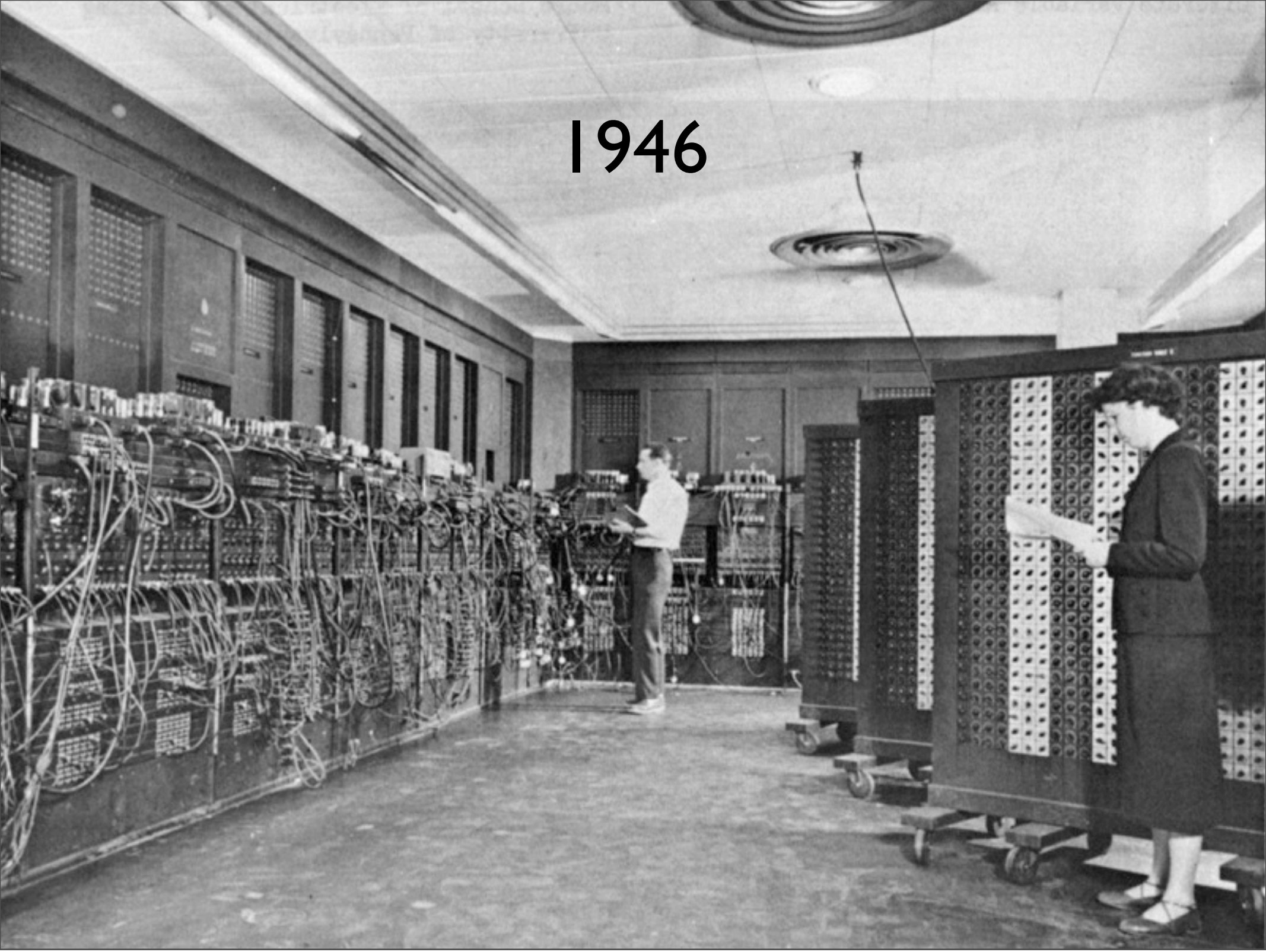
- ***Some software development facts***
- Our approach
  - Supporting maintenance
  - Moose an open-platform
- *Some* visual examples
- Conclusion

Software...



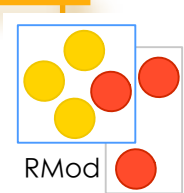
The Standish Group, 2004

Software is complex.

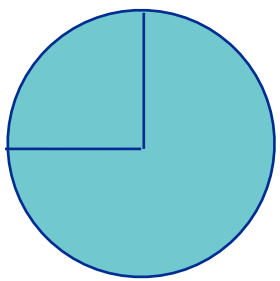

1946

How large is your project?

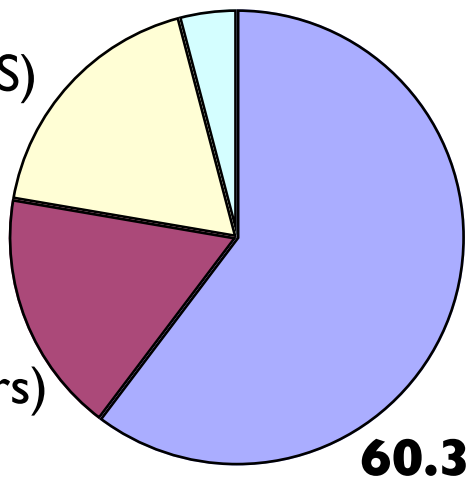
1'000'000 lines of code
* 2 = 2'000'000 seconds
/ 3600 = 560 hours
/ 8 = 70 days
/ 20 = 3 months



Maintenance: ***Continuous*** Development



Between 50% and 75% of global effort is spent on "maintenance" !

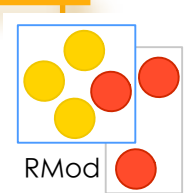


The bulk of the maintenance cost is due to new functionality
even with better requirements, it is ***hard*** to predict new functions

Lehman's Software Evolution Laws

Continuous Change: "A program that is used in a real-world environment must change, or become progressively less useful in that environment."

Software Entropy: "As a program evolves, it becomes more complex, and extra resources are needed to preserve and simplify its structure."



System evolution is like... SimCity

Software are living…

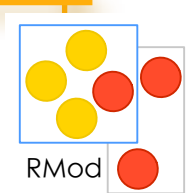
Early decisions were certainly good at that time

But the context ***changes***

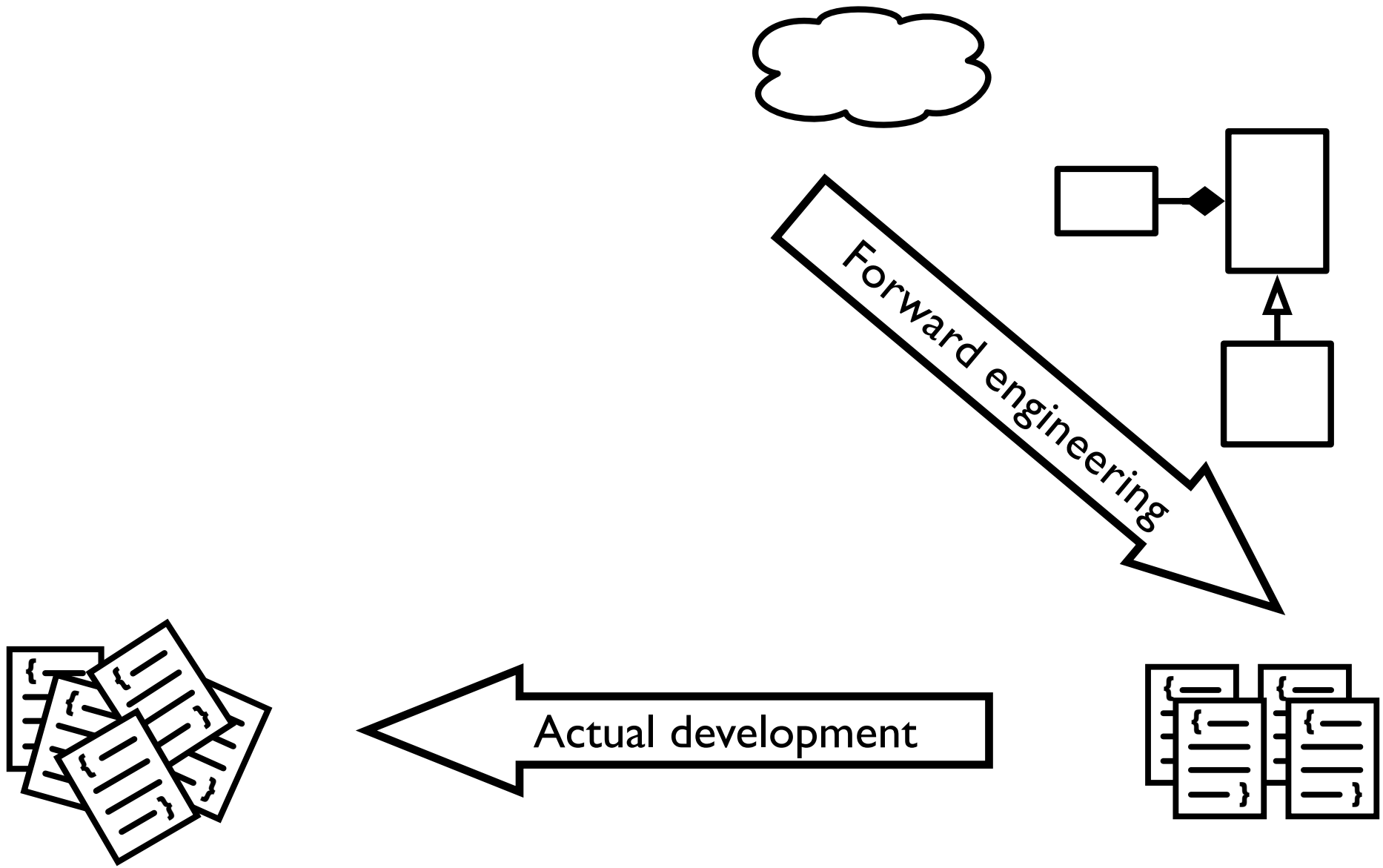
Customers ***change***

Technology ***changes***

People ***change***

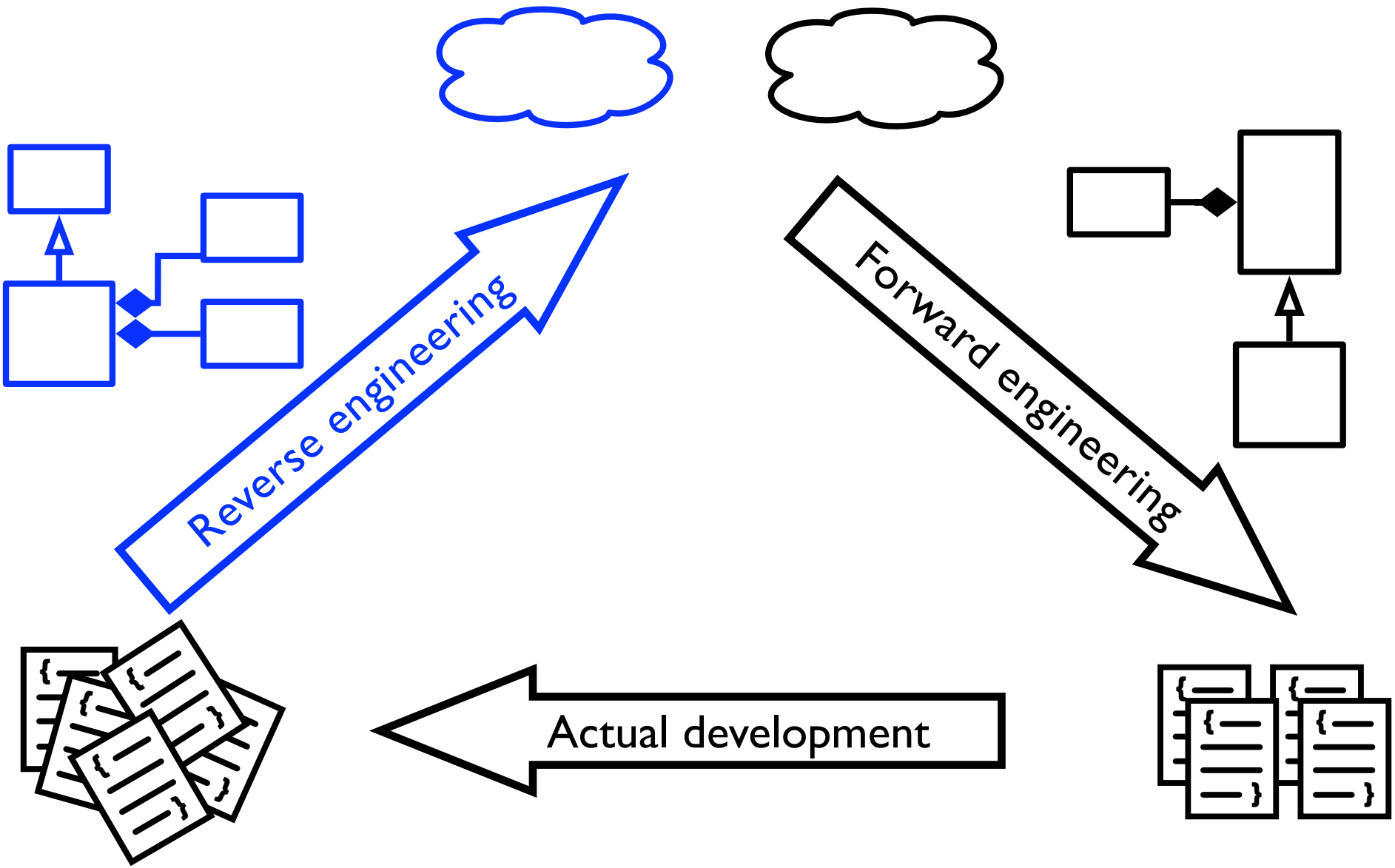


Software development is ***more*** than forward engineering.



Maintenance is

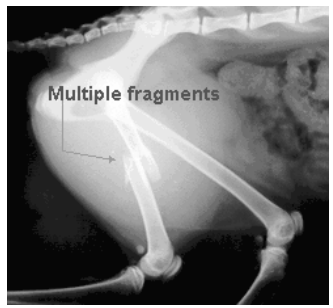
is needed to evolve the code.



Roadmap

- Some software development facts
- ***Our approach***
  - Supporting maintenance
  - Moose an open-platform
- *Some* visual examples
- Conclusion

Help teams maintaining large software



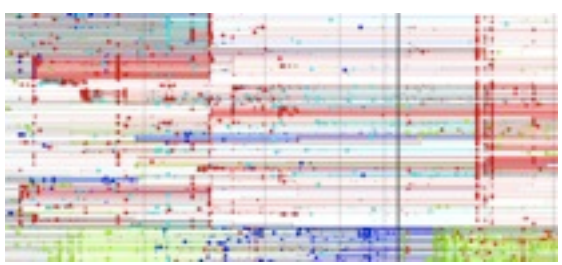
What is the xray for software?

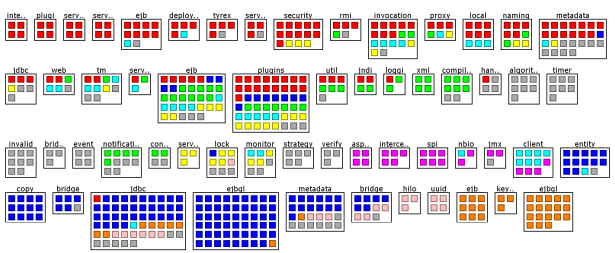
code, people, practices

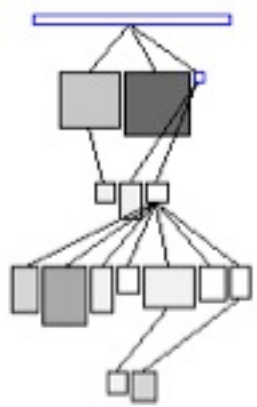
Which analyses?

How can you monitor your system (dashboards....)

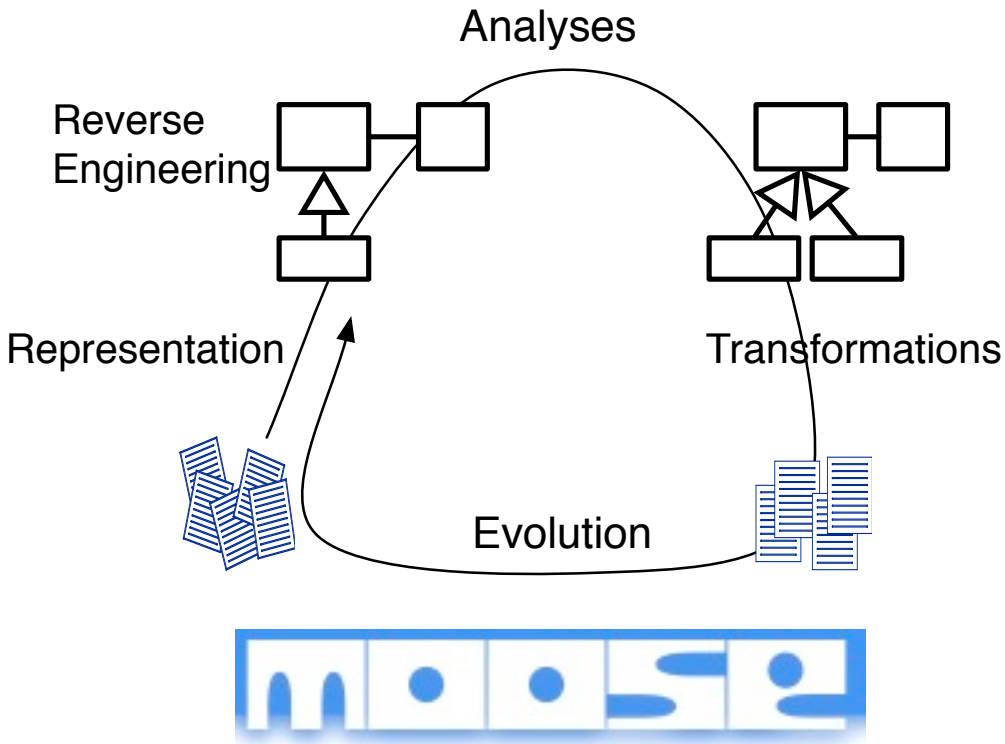
How to present extracted information?







Since 1996...



Topics

Metamodeling, metrics, program understanding, visualization, evolution analysis, duplicated code detection, code Analysis, refactorings, test generation...

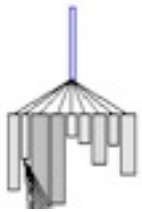
Contributions

Moose: an open-source extensible reengineering environment: (Lugano, Bern, Annecy, Anvers, Louvain la neuve, ULB, UTSL)

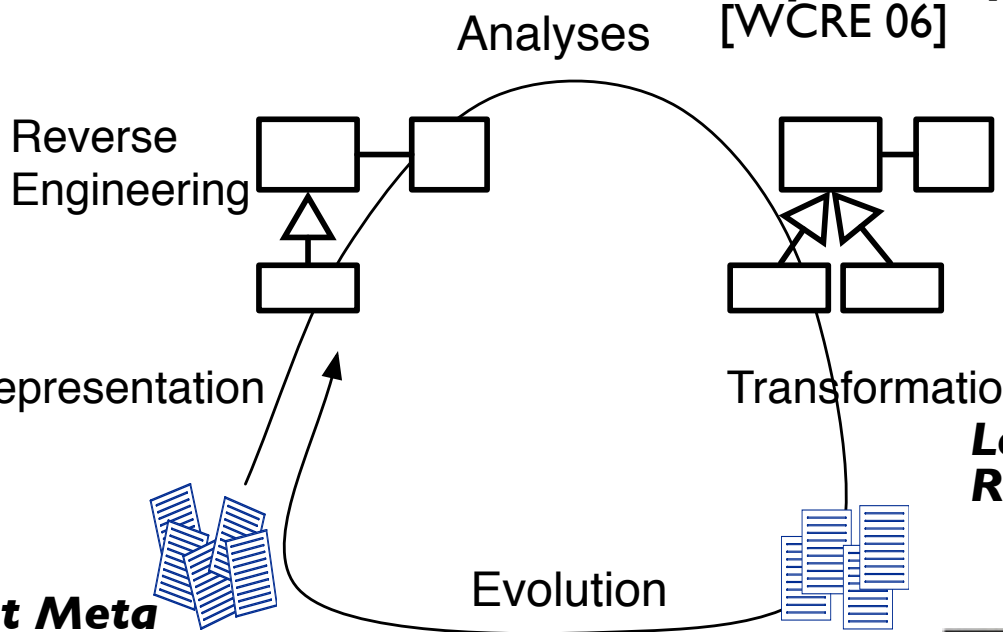
Contacts

Harman-Becker (3 Millions C++), Bedag (Cobol), Nokia, ABB, IMEC

Understanding Large Systems
[WCRE99, TSI00, TSE03]
Static/Dynamic Information
[ICSM99]
Feature Analysis
[JSME 06]
Class Understanding
[OOPSLA01, TSE04]
Package Blueprints
[ICSM 07]
Distribution Maps
[ICSM 06]



Software Metrics
[LMO99, OOPSLA00]
Duplicated Code Identification
[ICSM99, ICSM02]
Group Identification
[ASE03]
Test Generation
[CSMR 06]
Concept Identification
[WCRE 06]

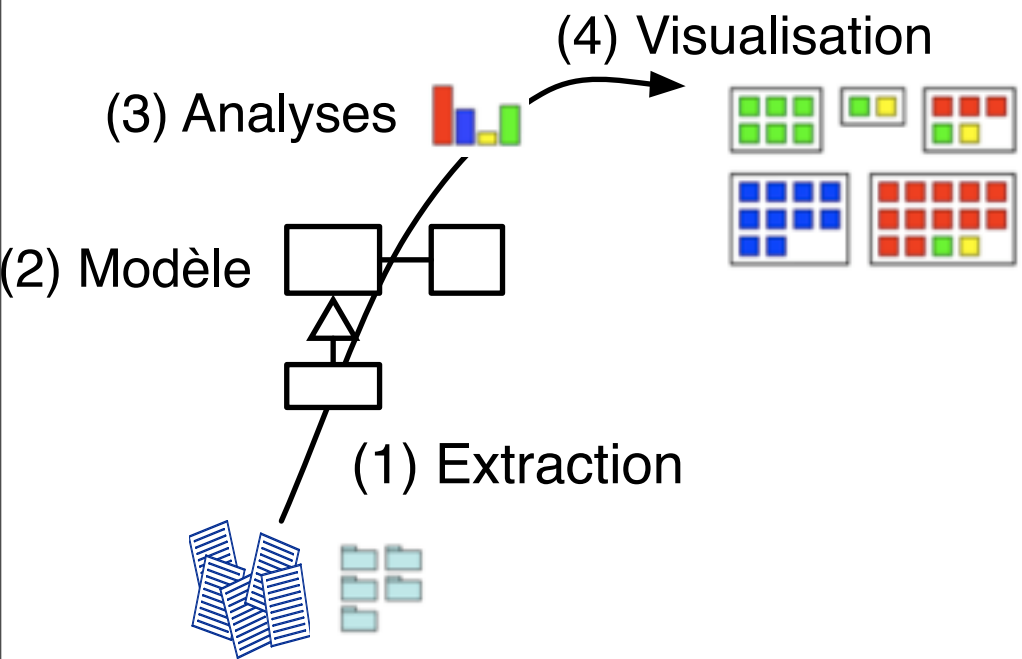


Language Independent Refactorings
[IWPSE 00]

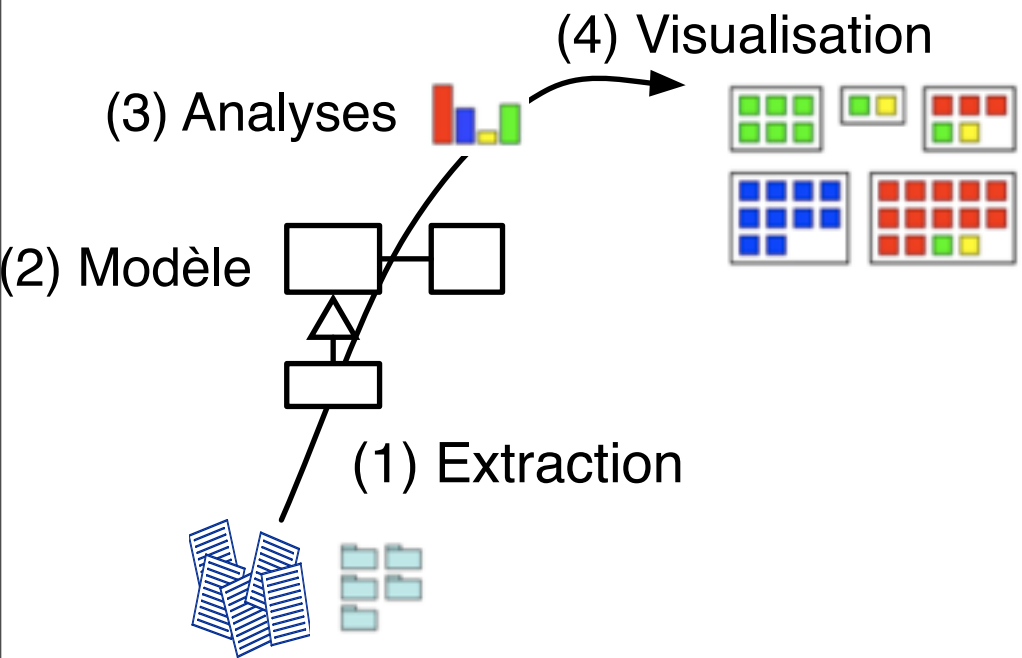
Language Independent Meta Model (FAMIX)
[UML99]
An Extensible Reengineering Environment (Moose)
[Models 06]

Reengineering Patterns
Version Analyses
[ICSM 05]
HISMO metamodel
[JSME 05]

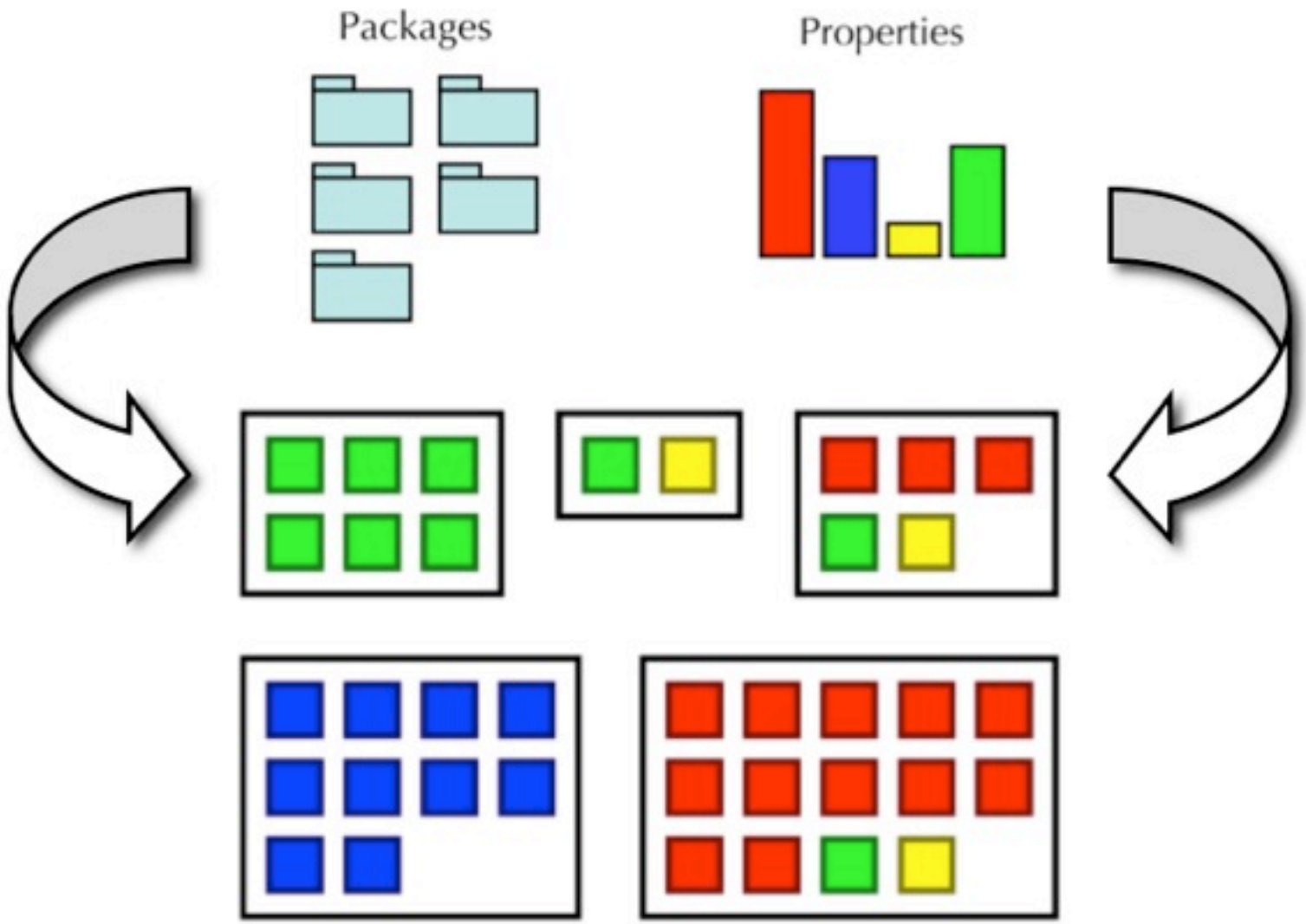
An example: who is responsible of what?



An example: who is responsible of what?



Distribution Map

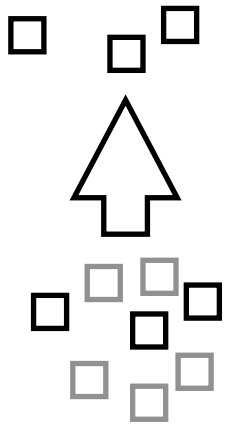


Moose is a powerful environment

McCabe = 21

NOM = 102

LOC = 753,000

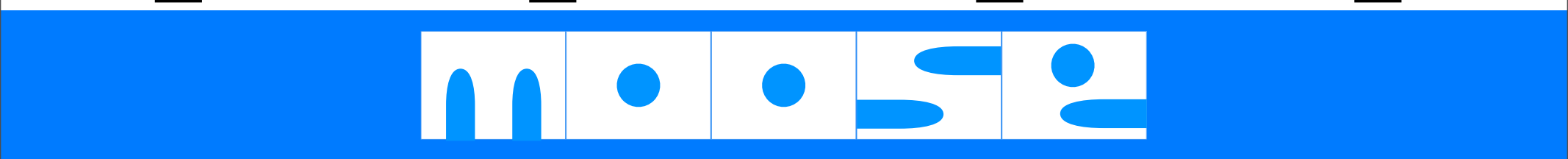


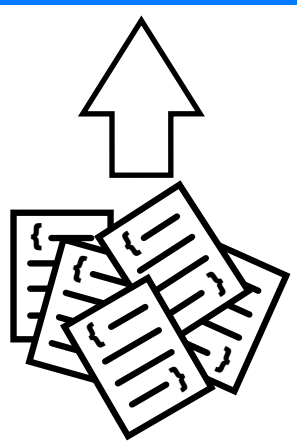
Metrics

Queries

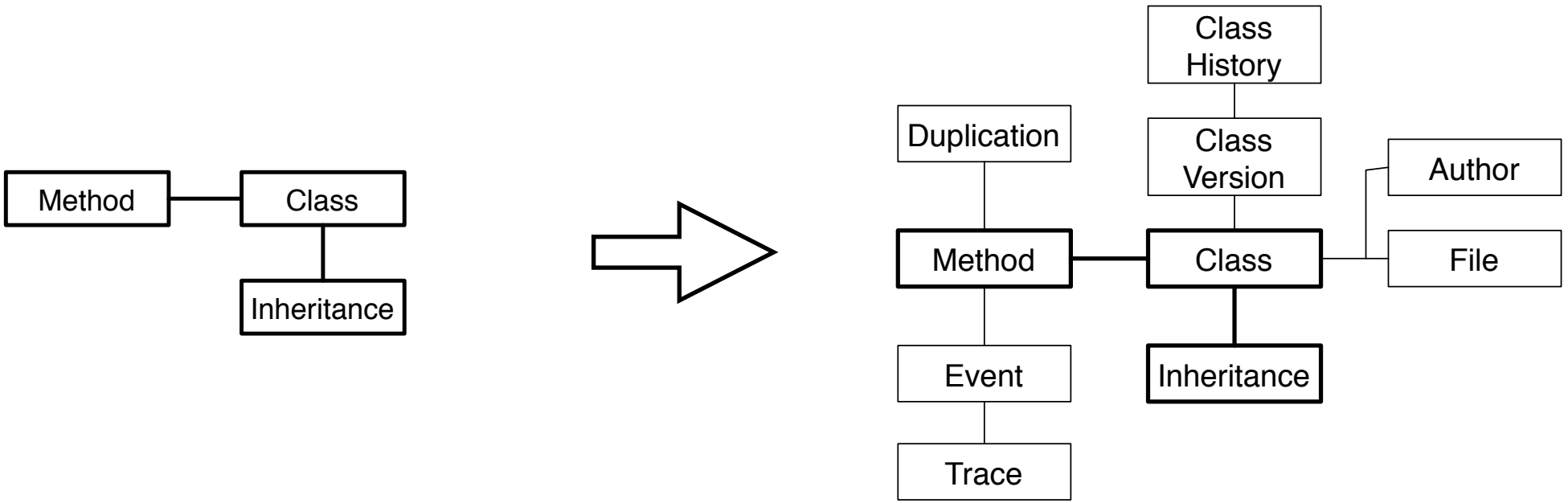
Visualizations

• • •





Moose is designed to be extensible



open

meta-described

Moose has been validated on real life systems

Several large, industrial case studies (NDA)

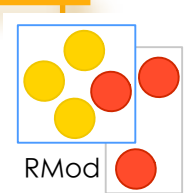
- Harman-Becker
- Nokia
- Daimler
- Siemens

Different implementation languages (C++, Java, Smalltalk, Cobol)

- We use external C++ parsers

Different sizes

Moose is used in several research groups



Roadmap

- Some software development facts
- Our approach
  - Supporting maintenance
  - Moose an open-platform
- ***Some* visual examples**
- Conclusion

Challenges in Visualization

Screen size
Max 12 colors
Edge-crossing
Limited short-term memory (three to nine)
Extracting semantics out
Beauty cannot be a goal

Get some help from
- Gestalt principles
- pre-attentive visualization

Understanding large systems

Understanding code is difficult!

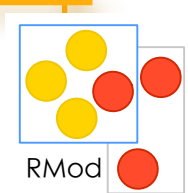
Systems are large

Code is abstract

Should I really convinced you?

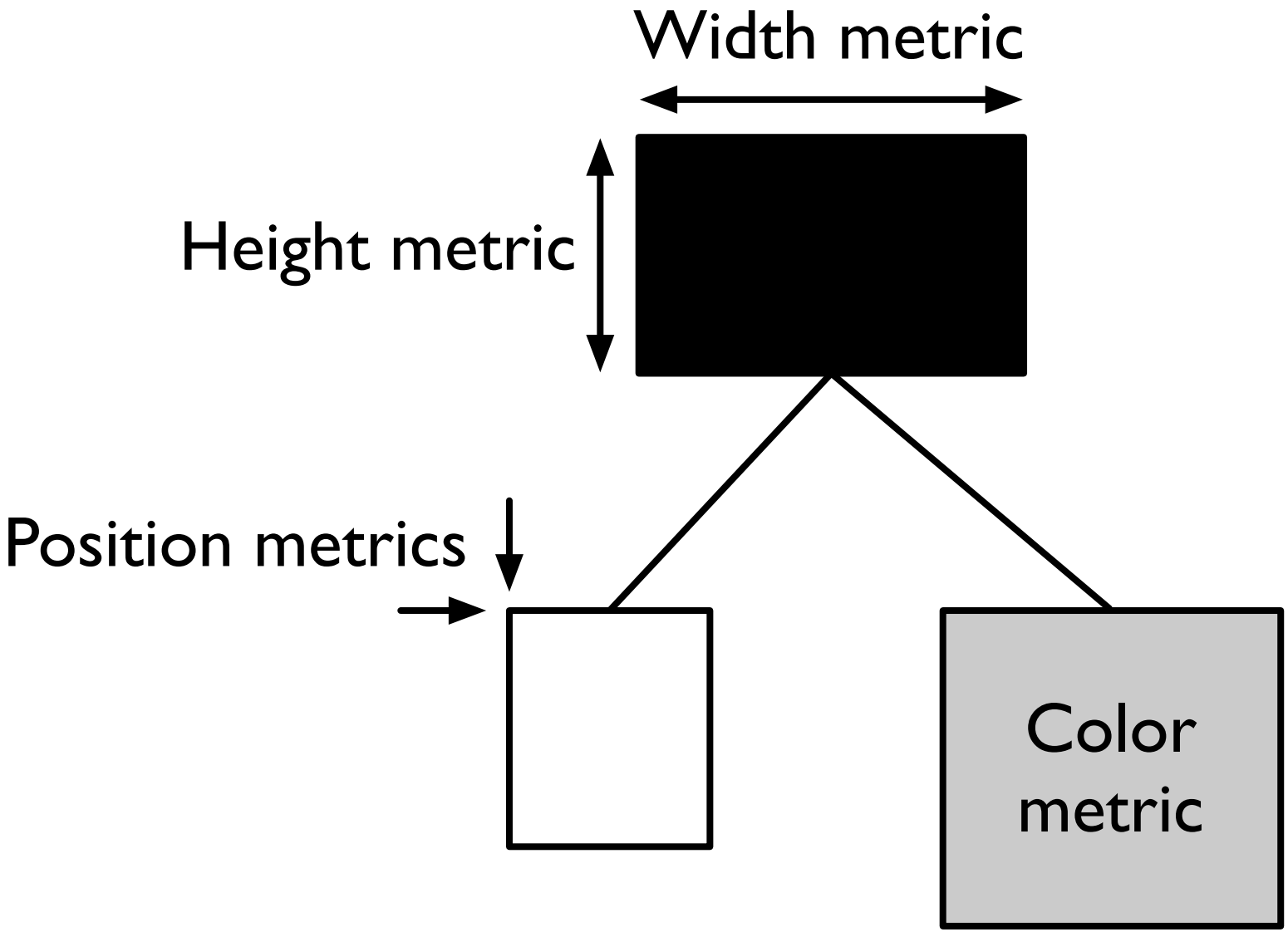
Some existing approaches

- Metrics: you often get ***meaningless*** results once ***combined***
- Visualization: often beautiful but ***with little*** meaning

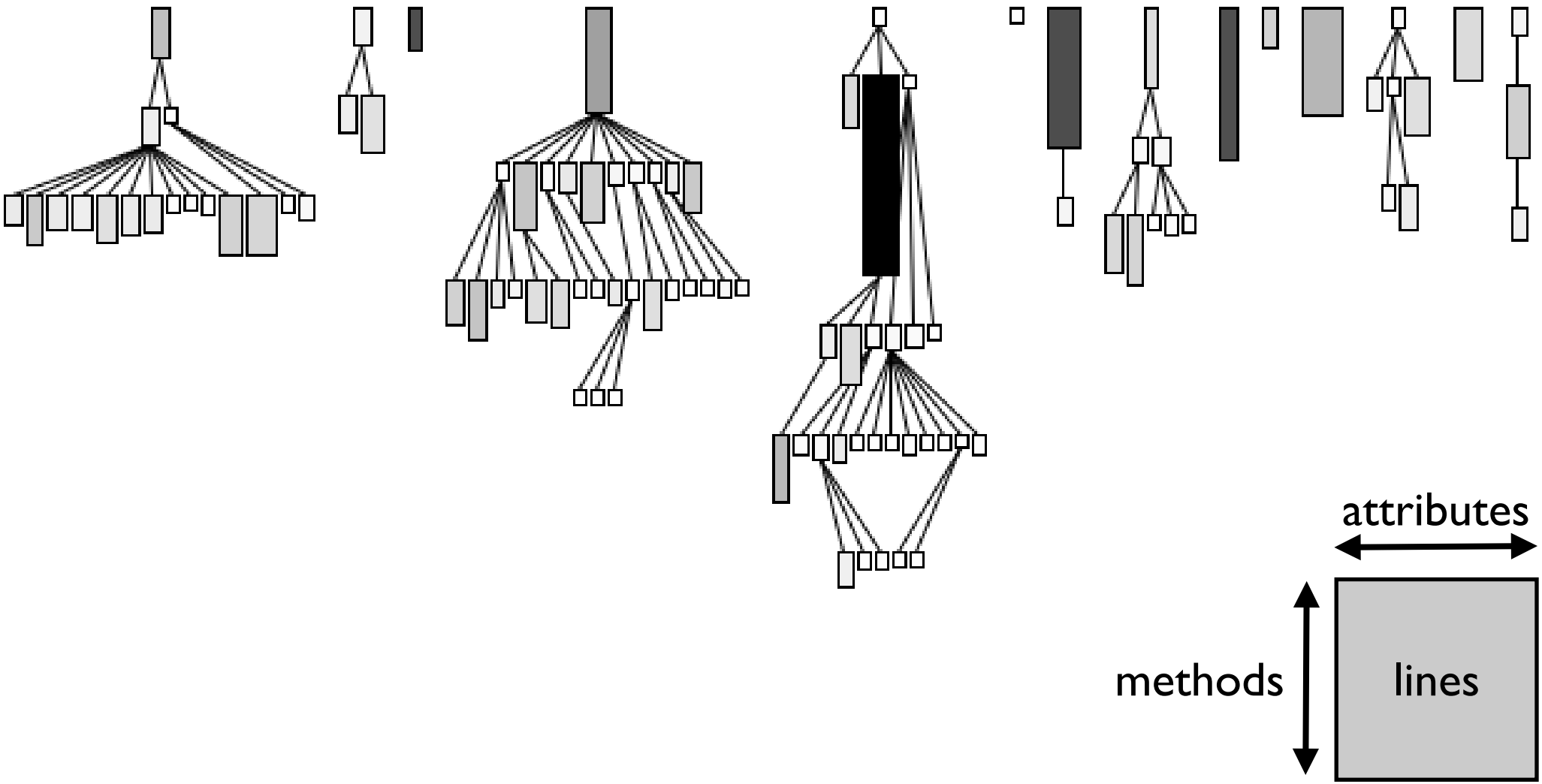


Polymetric views show up to 5 metrics.

Lanza etal, 03

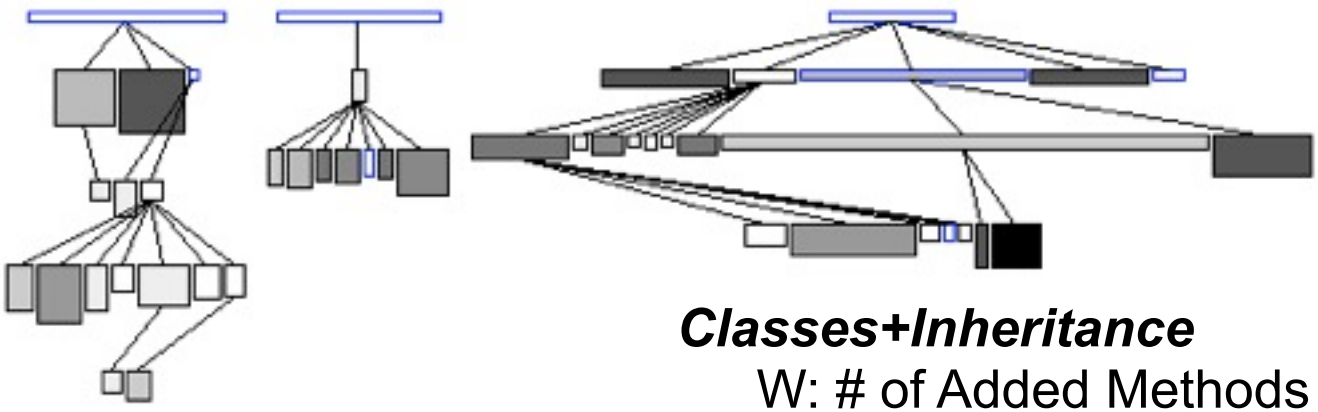


System Complexity shows class hierarchies.



Polymetric views condense information

To get a feel of the inheritance semantics: adding vs. reusing

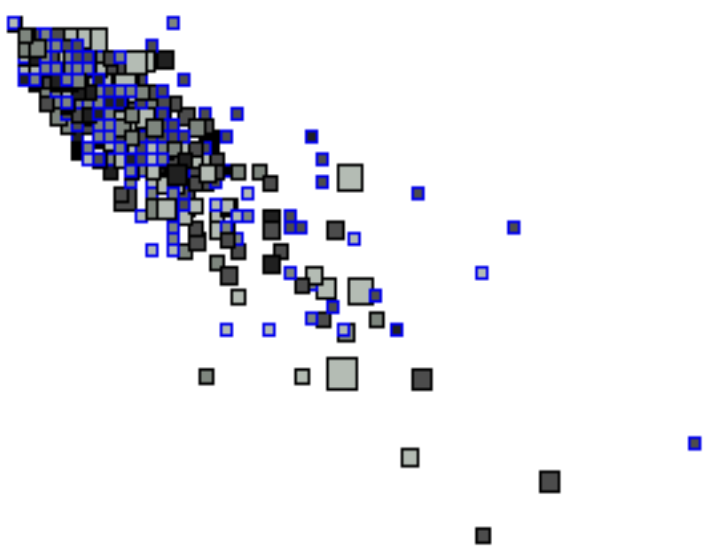


Classes+Inheritance

W: # of Added Methods

H: # of Overridden Metho

C: # of Method Extended



methods

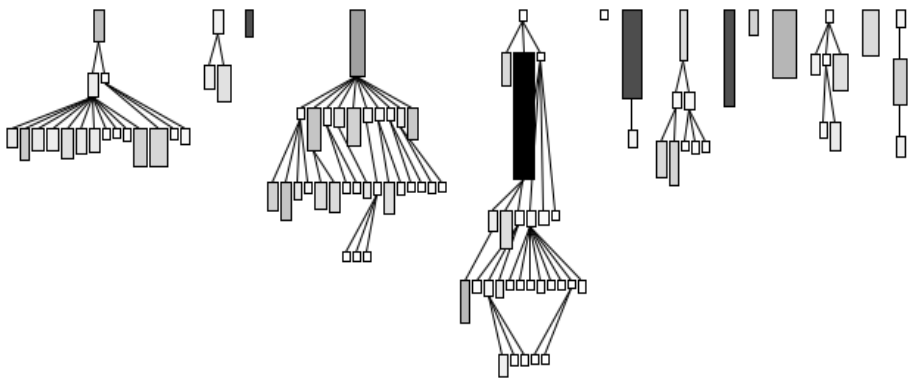
LOC

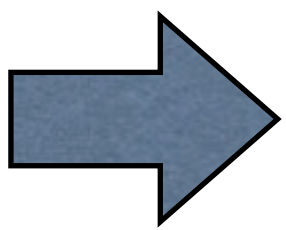
statements

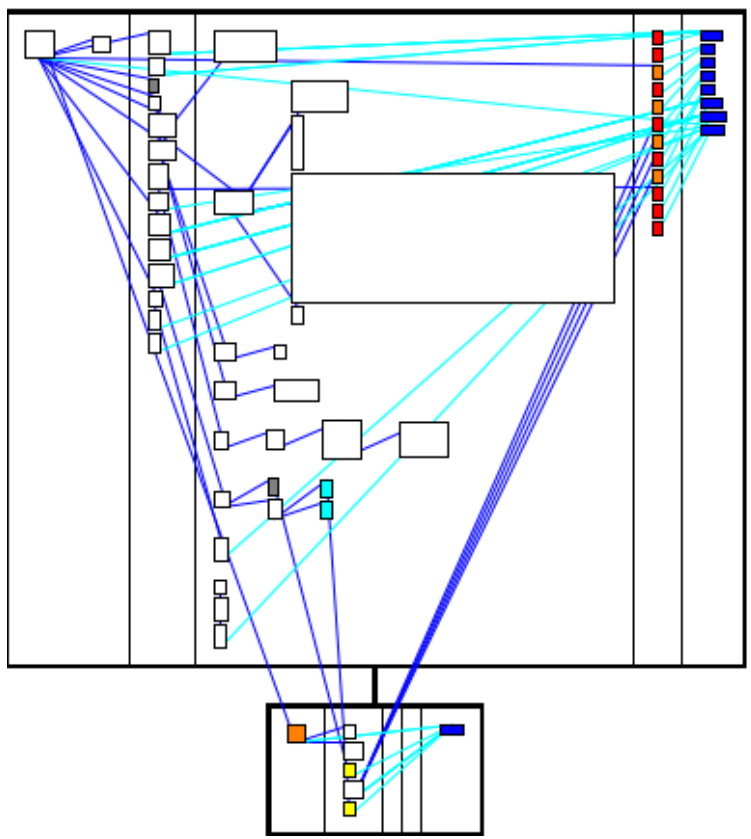
parameters

Understanding classes

Understanding even a class is difficult!

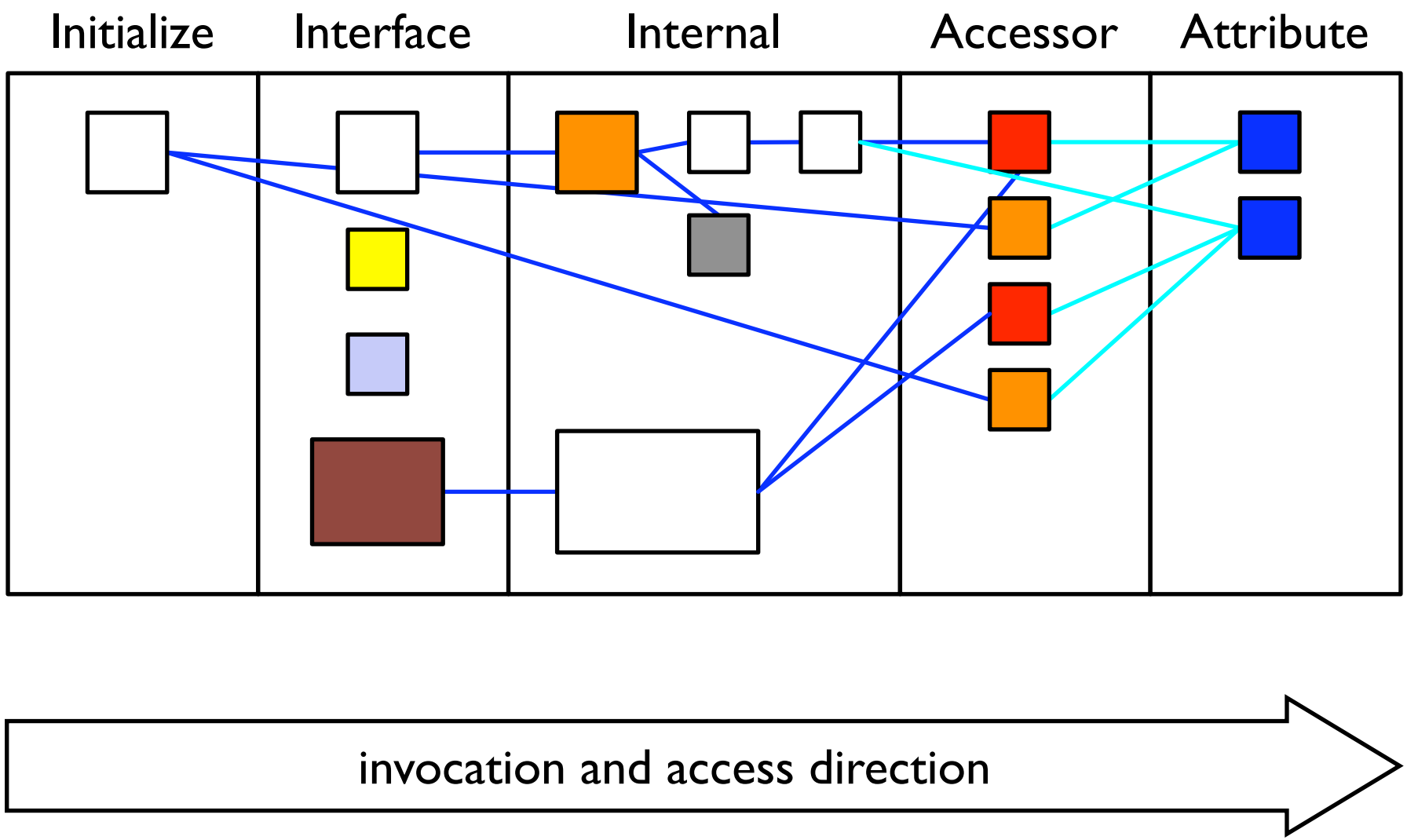




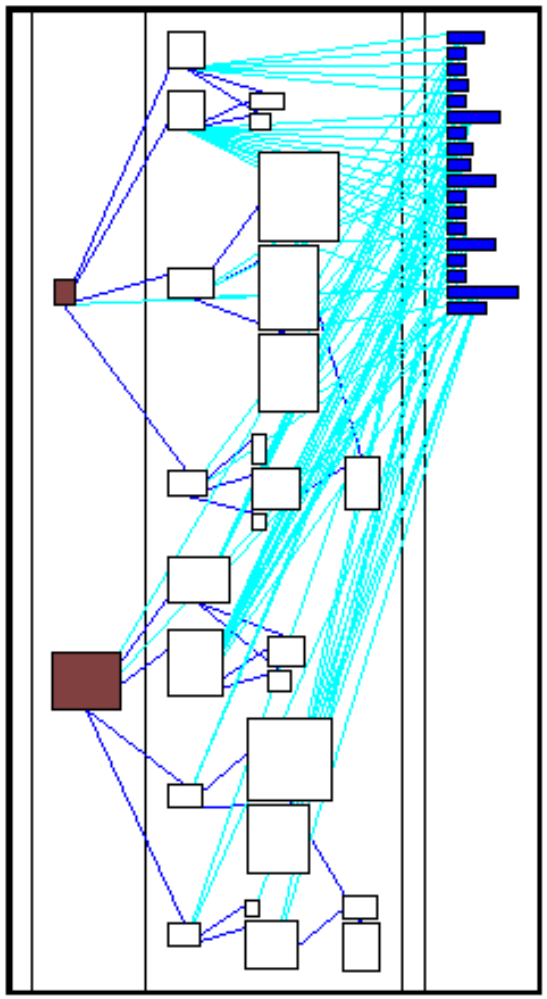


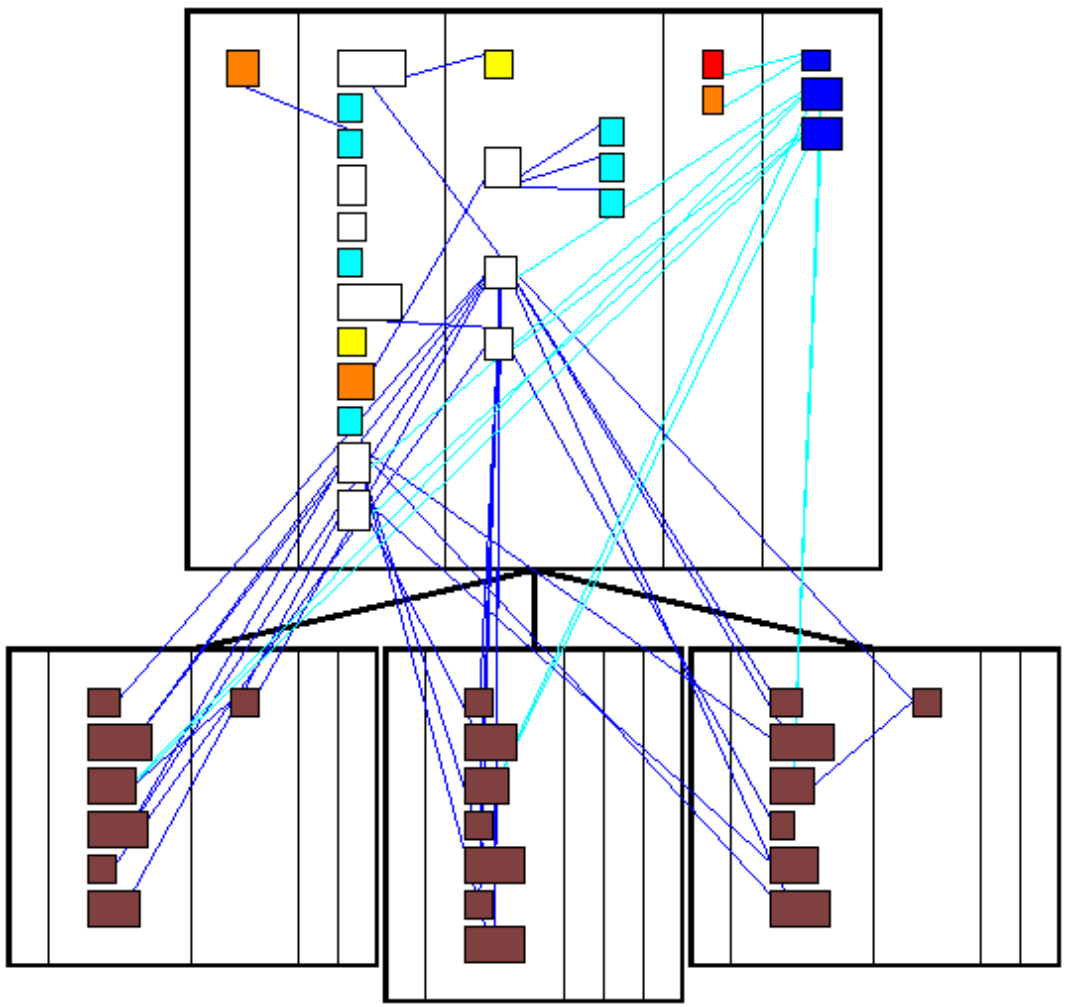
Class Blueprint shows class internals.

Ducasse, Lanza, 05



Class Blueprint shows class internals.

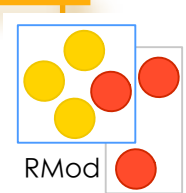




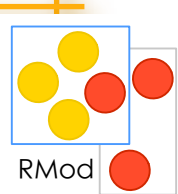
Cycles?

Identify
Understand
Fix

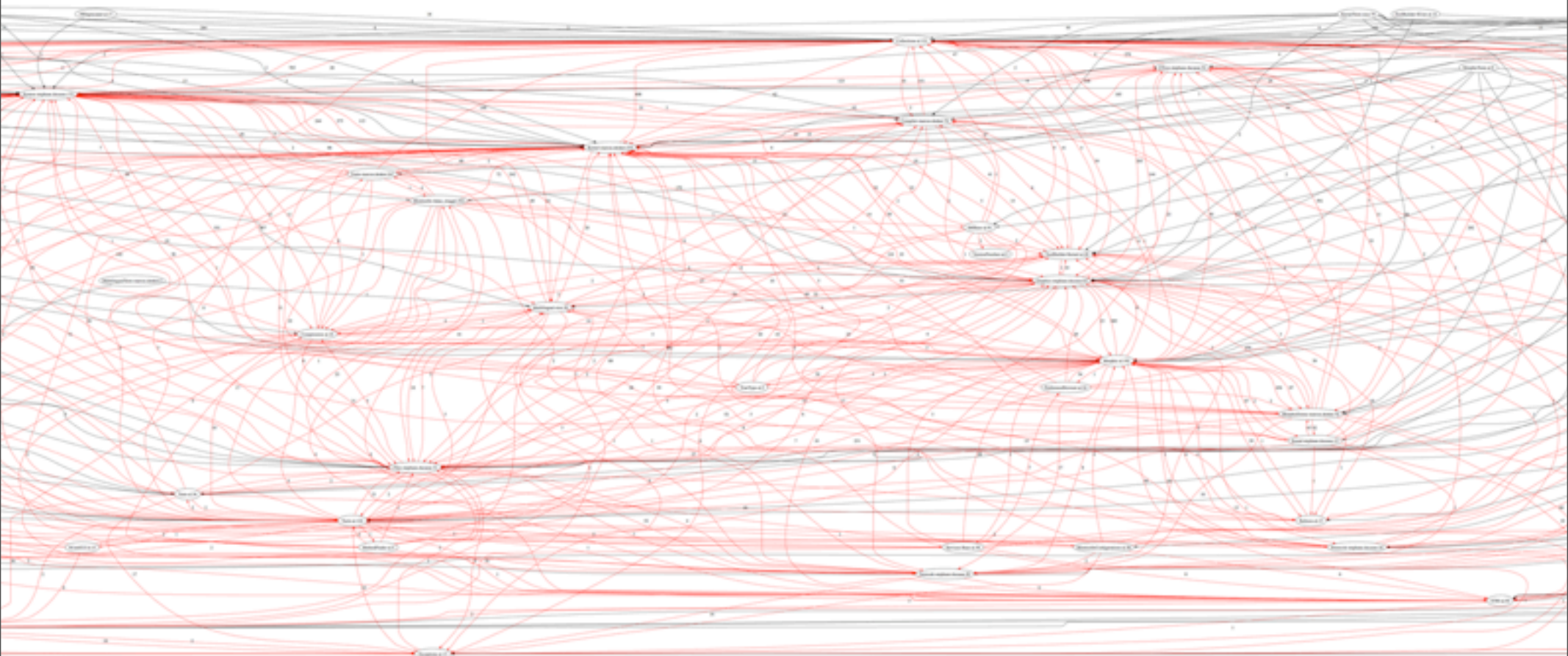
Enhancing Dependency Structural Matrix



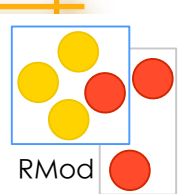
Graph you said?



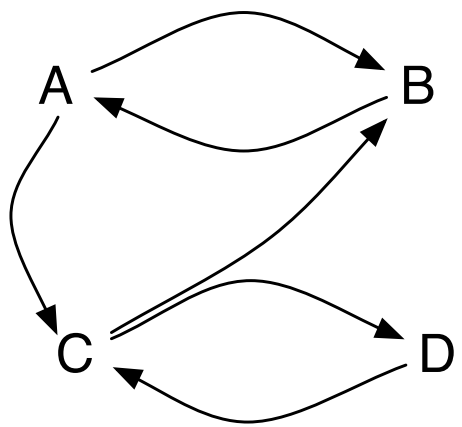
Graph you said?



Graph you said?



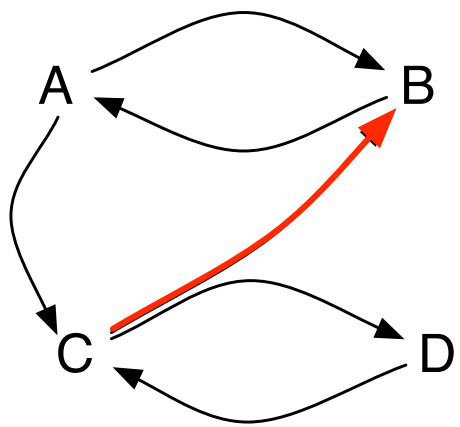
Building a DSM



	A	B	C	D
A		X		
B	X		X	
C	X			X
D			X	

	A	B	C	D
A	0	1	0	0
B	1	0	1	0
C	1	0	0	1
D	0	0	1	0

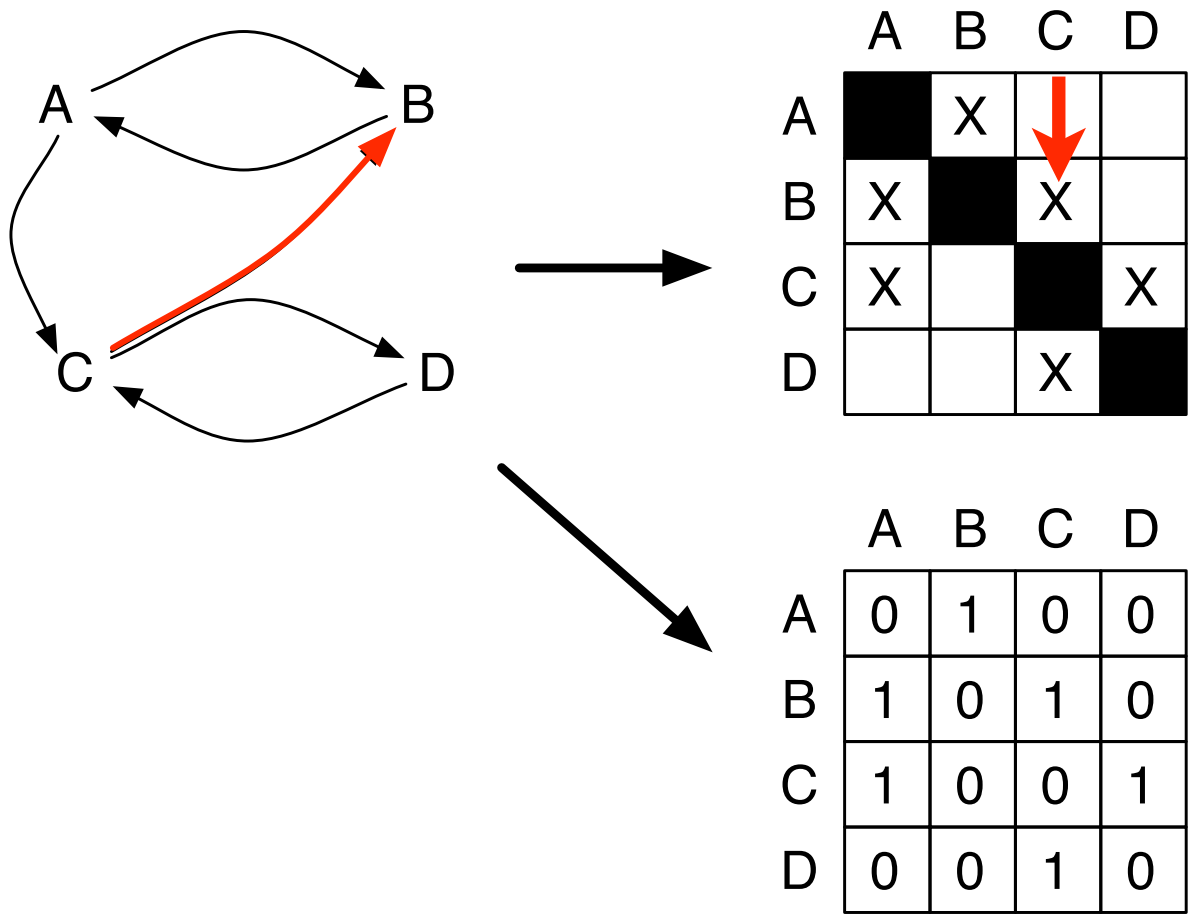
Building a DSM



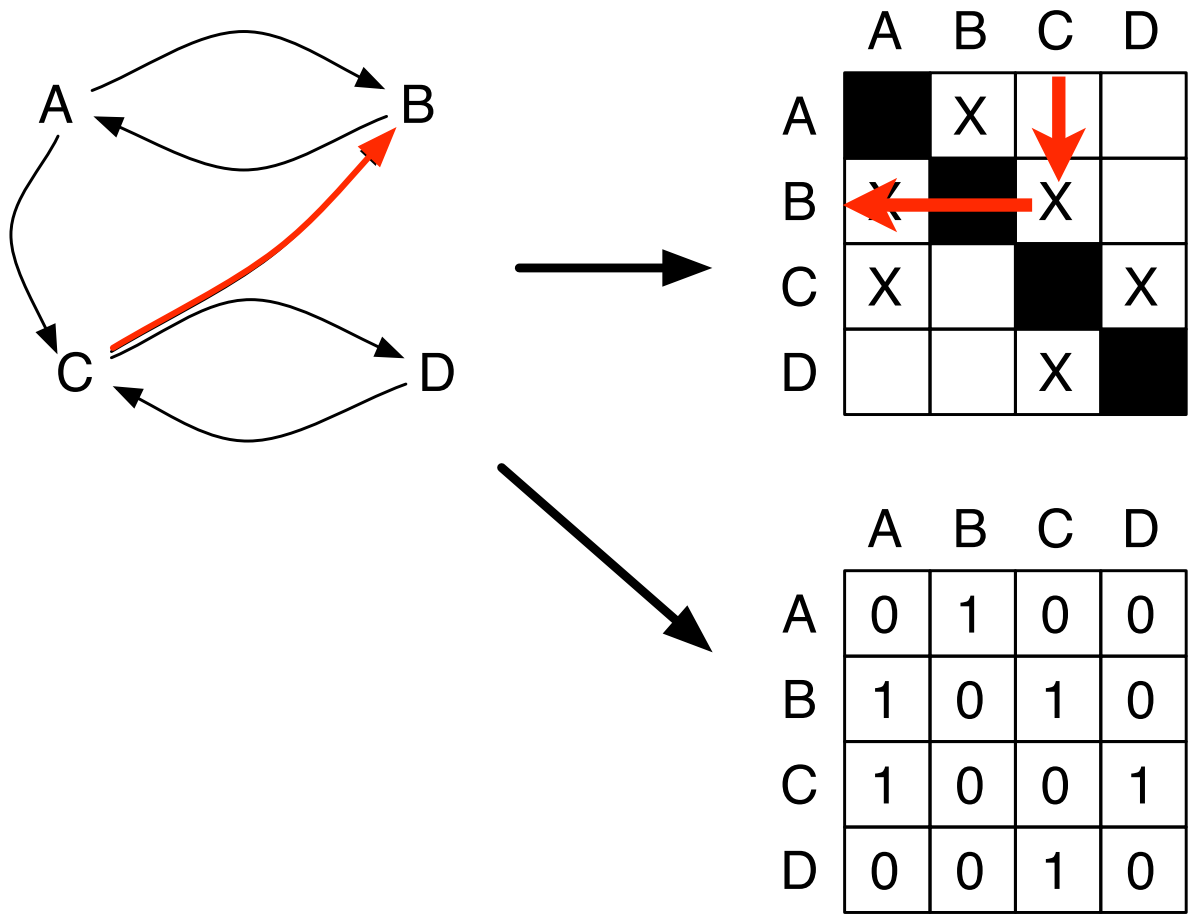
	A	B	C	D
A		X		
B	X		X	
C	X			X
D			X	

	A	B	C	D
A	0	1	0	0
B	1	0	1	0
C	1	0	0	1
D	0	0	1	0

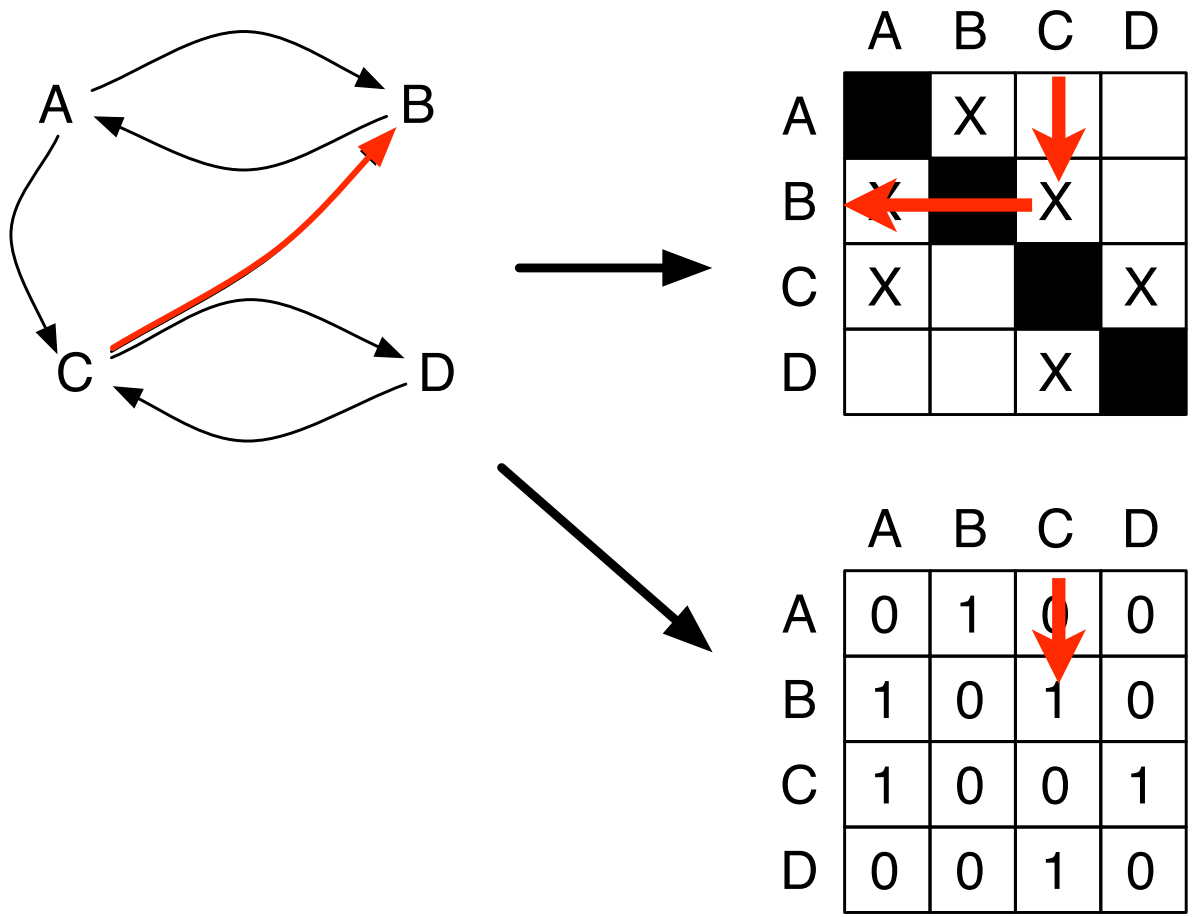
Building a DSM



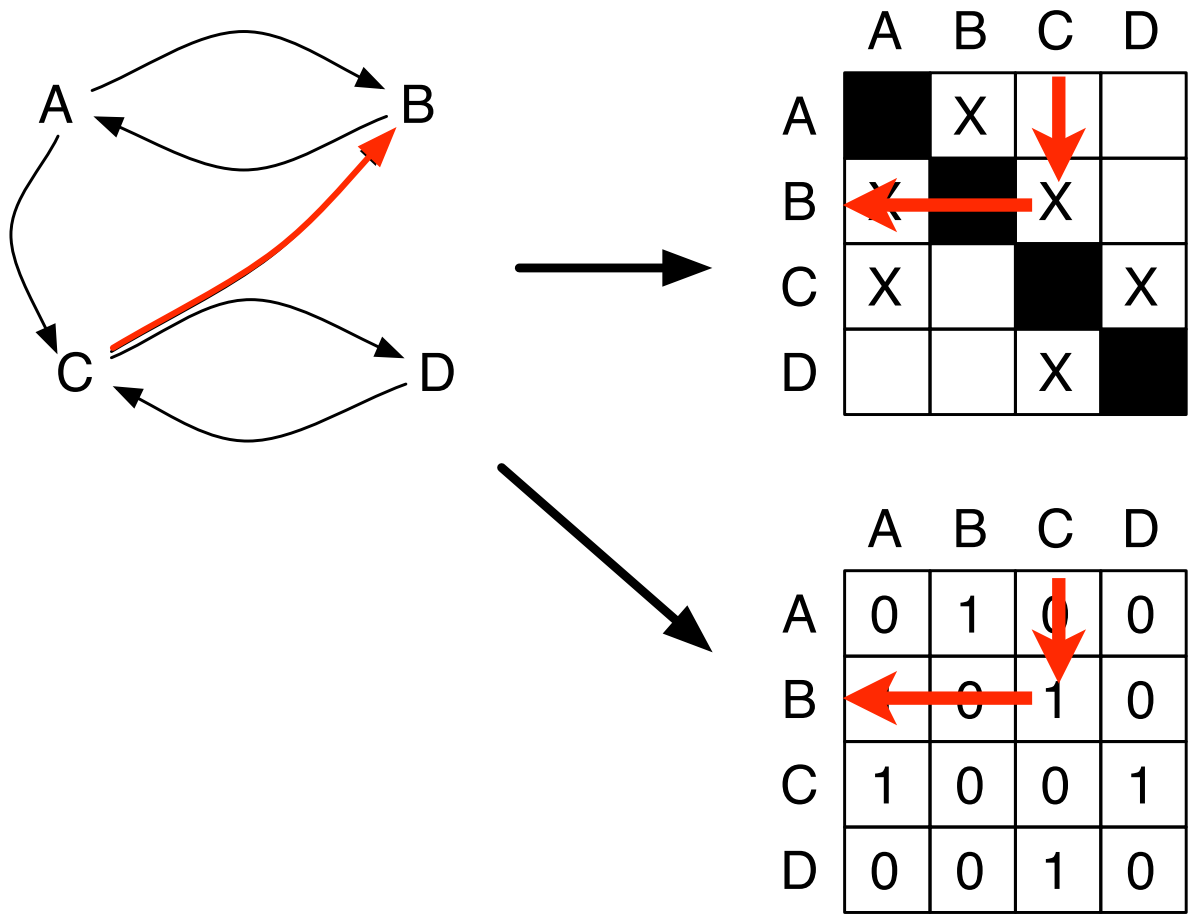
Building a DSM



Building a DSM



Building a DSM



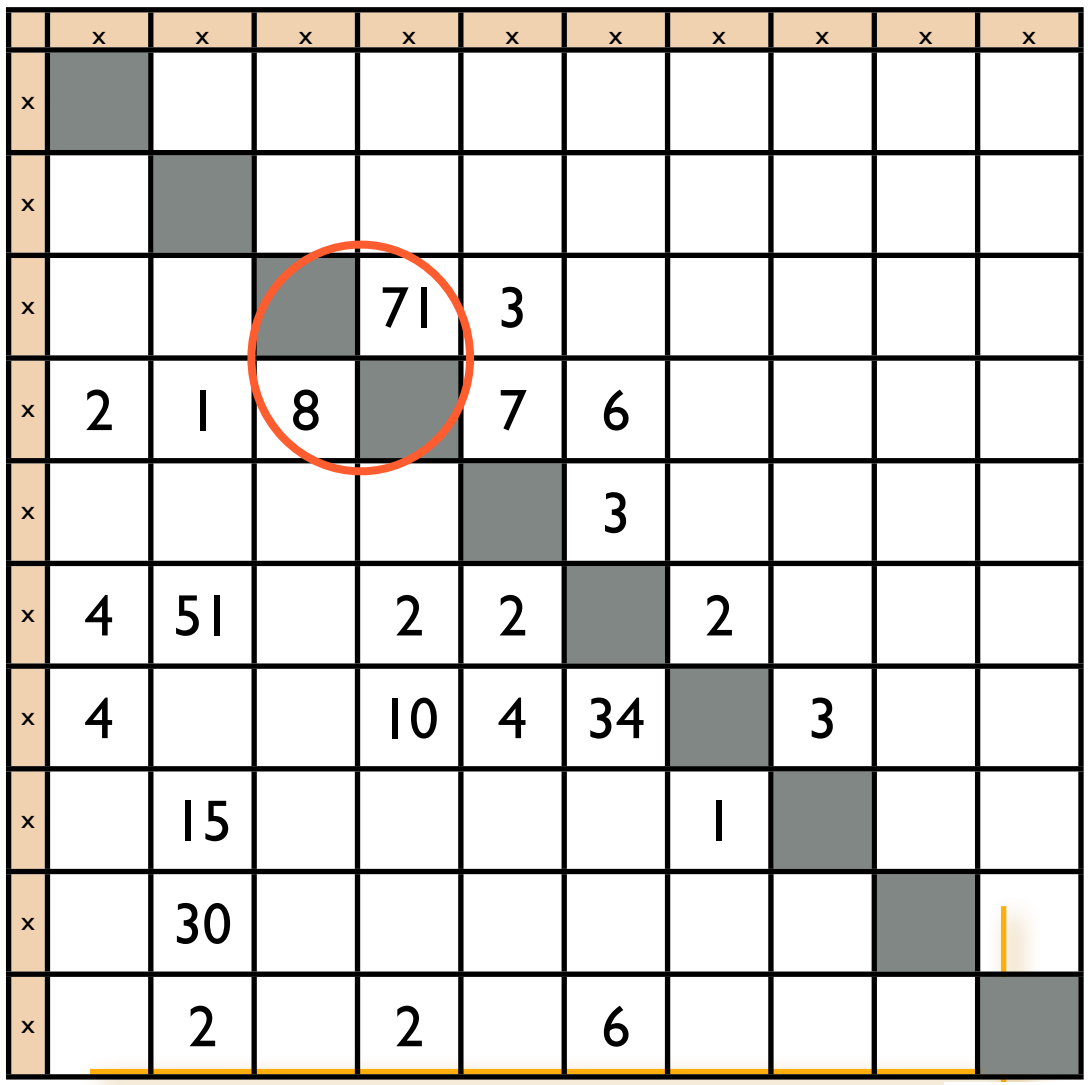
7 Packages visualization

- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages

	x	x	x	x	x	x	x	x	x	x
x										
x										
x				71	3					
x	2	1	8		7	6				
x						3				
x	4	51		2	2		2			
x	4			10	4	34		3		
x		15					1			
x		30								
x		2		2		6				

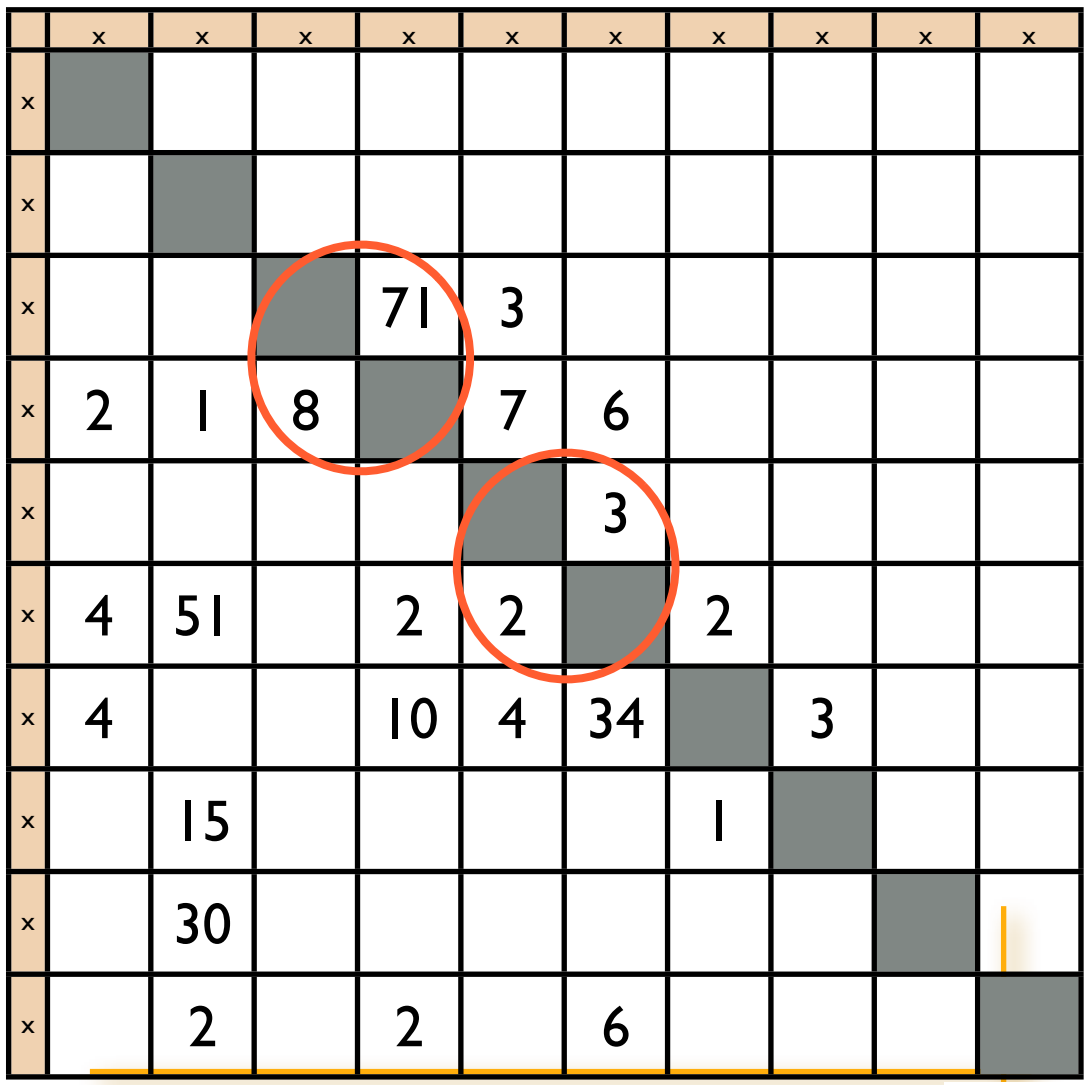
7 Packages visualization

- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages



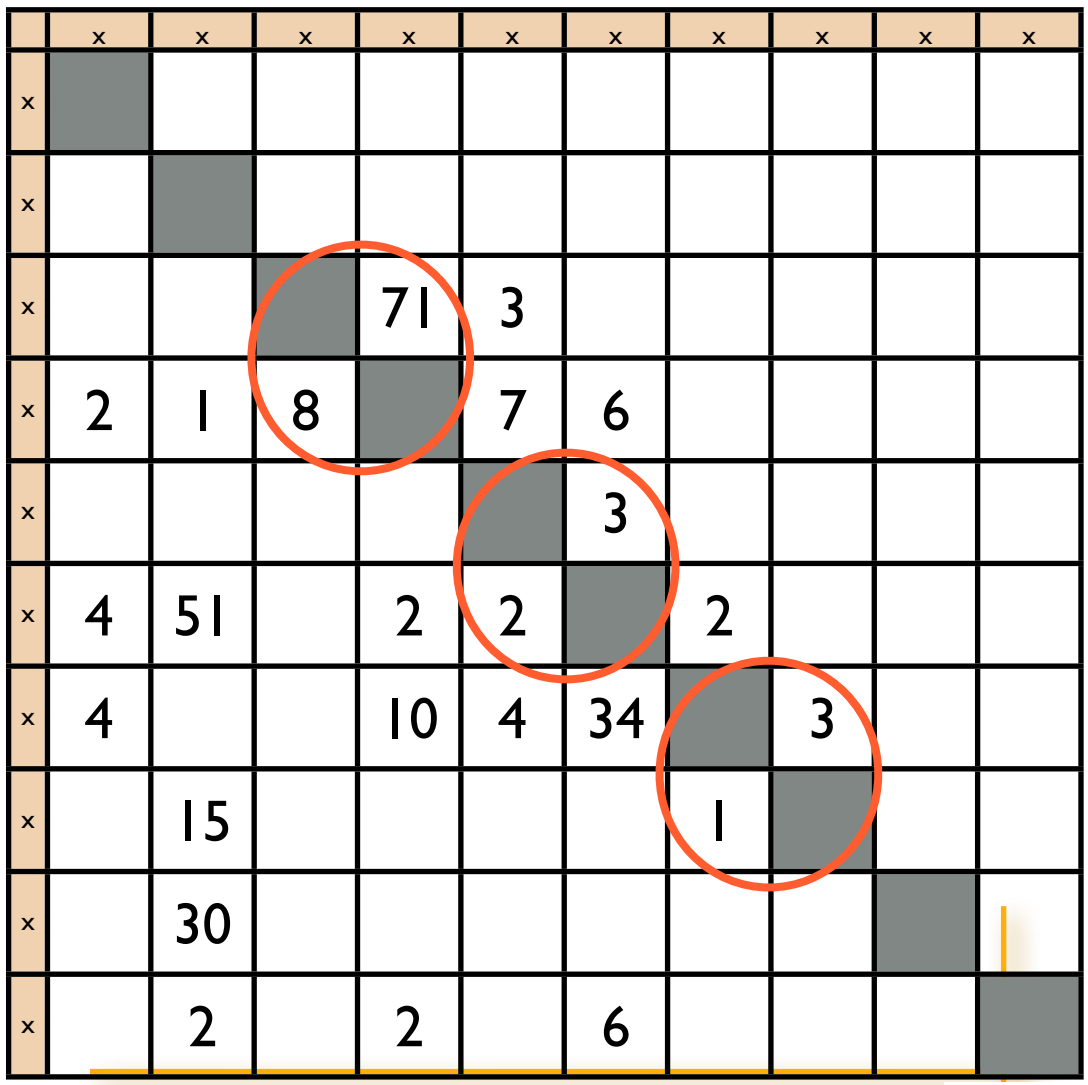
7 Packages visualization

- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages



7 Packages visualization

- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages



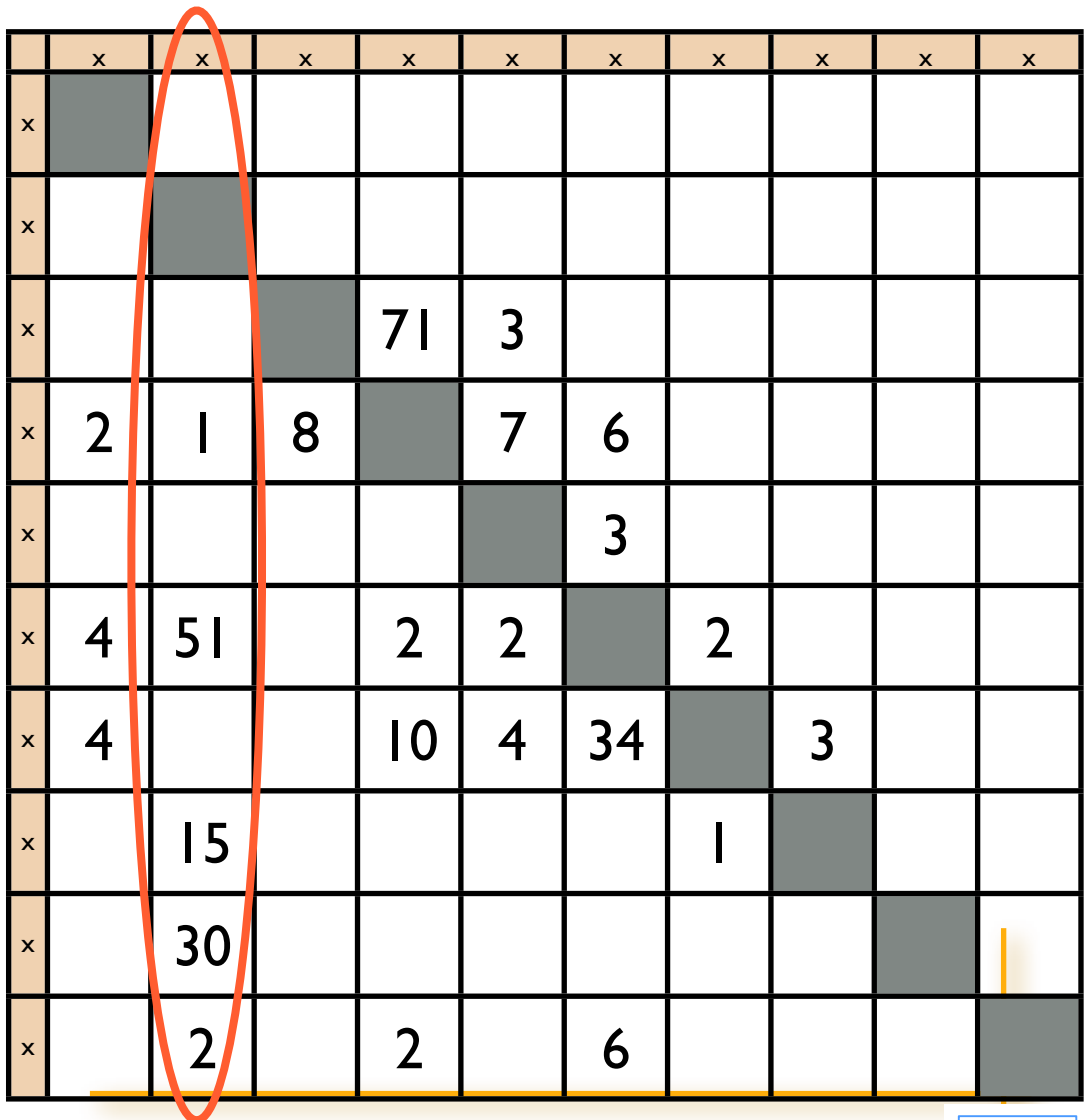
7 Packages visualization

- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages

	x	x	x	x	x	x	x	x	x	x
x										
x										
x				71	3					
x	2	1	8		7	6				
x						3				
x	4	51		2	2		2			
x	4			10	4	34		3		
x		15					1			
x		30								
x		2		2		6				

7 Packages visualization

- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages



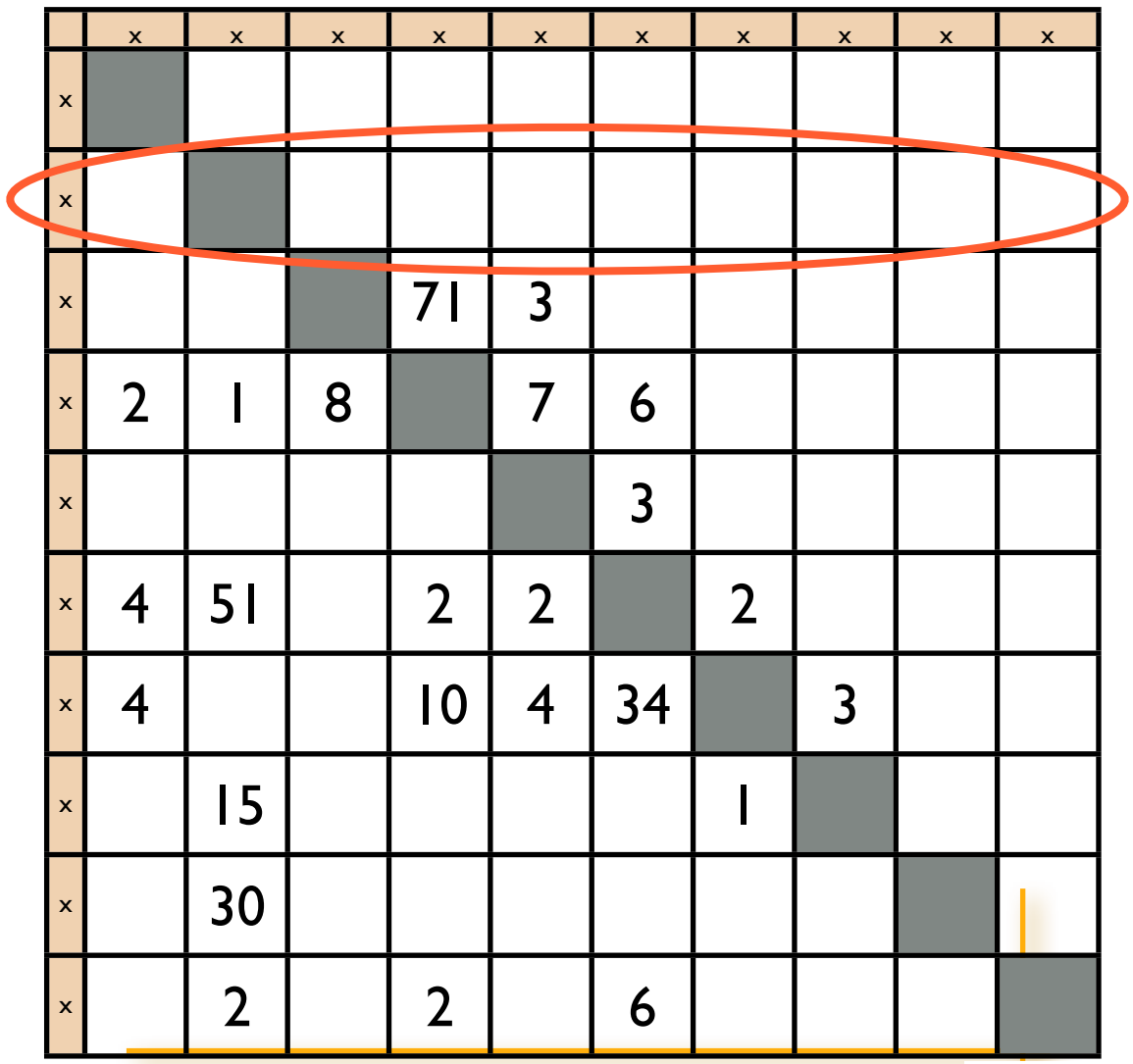
7 Packages visualization

- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages

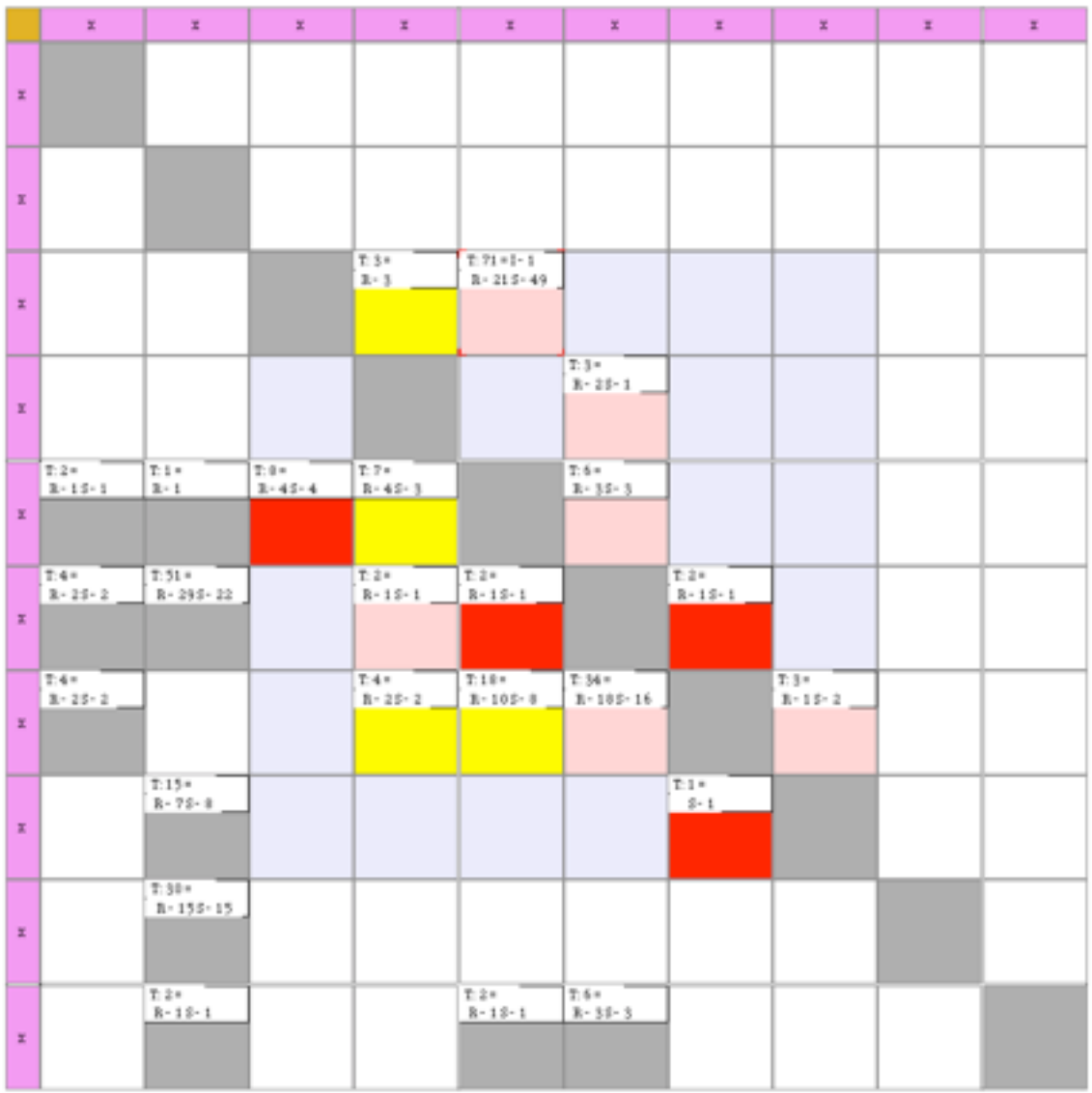
	x	x	x	x	x	x	x	x	x	x
x										
x										
x				71	3					
x	2	1	8		7	6				
x						3				
x	4	51		2	2		2			
x	4			10	4	34		3		
x		15					1			
x		30								
x		2		2		6				

7 Packages visualization

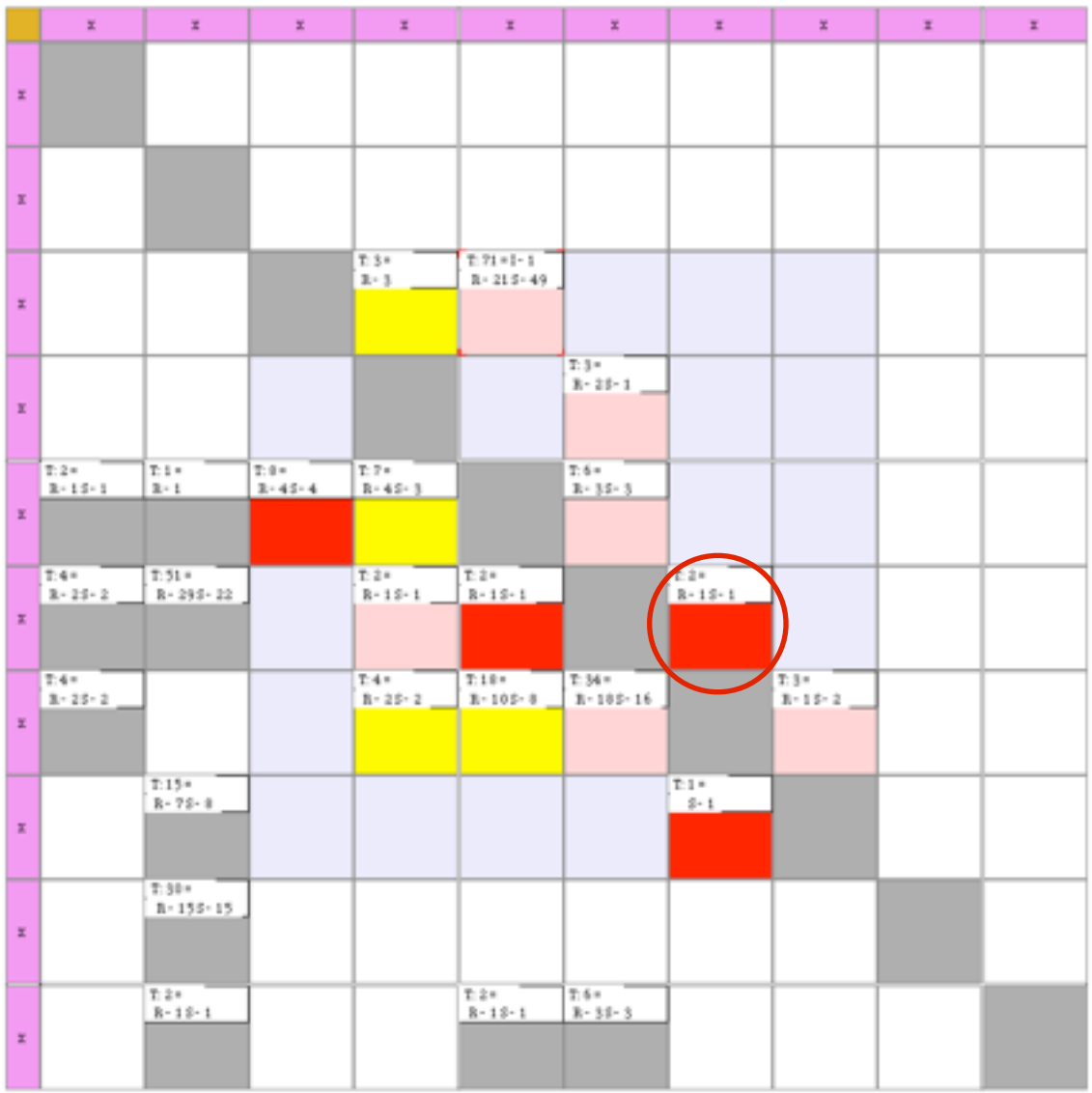
- 1 cell = 1 dependency
- 1 column = used packages
- 1 line = using packages



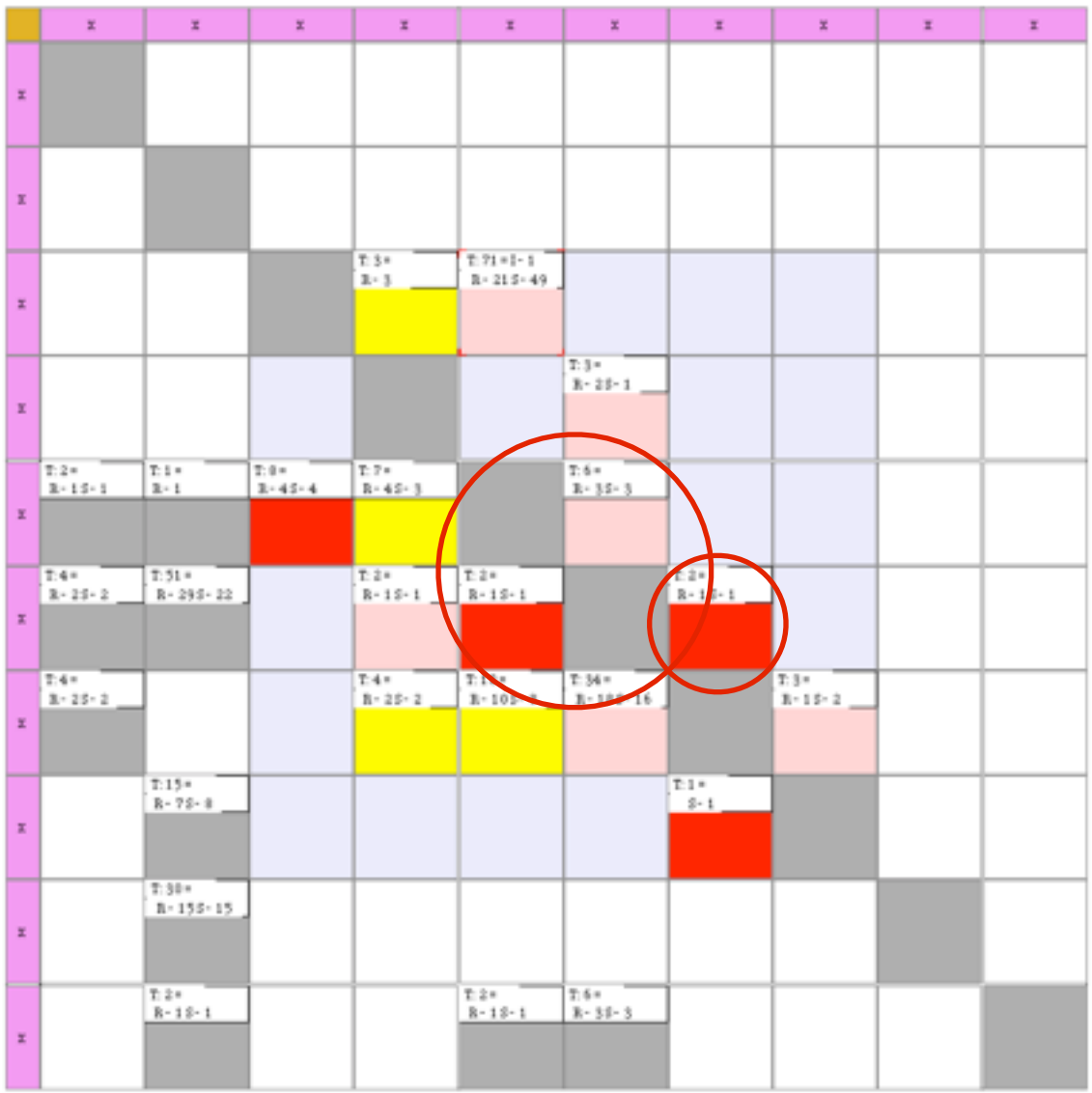
Identify cycles



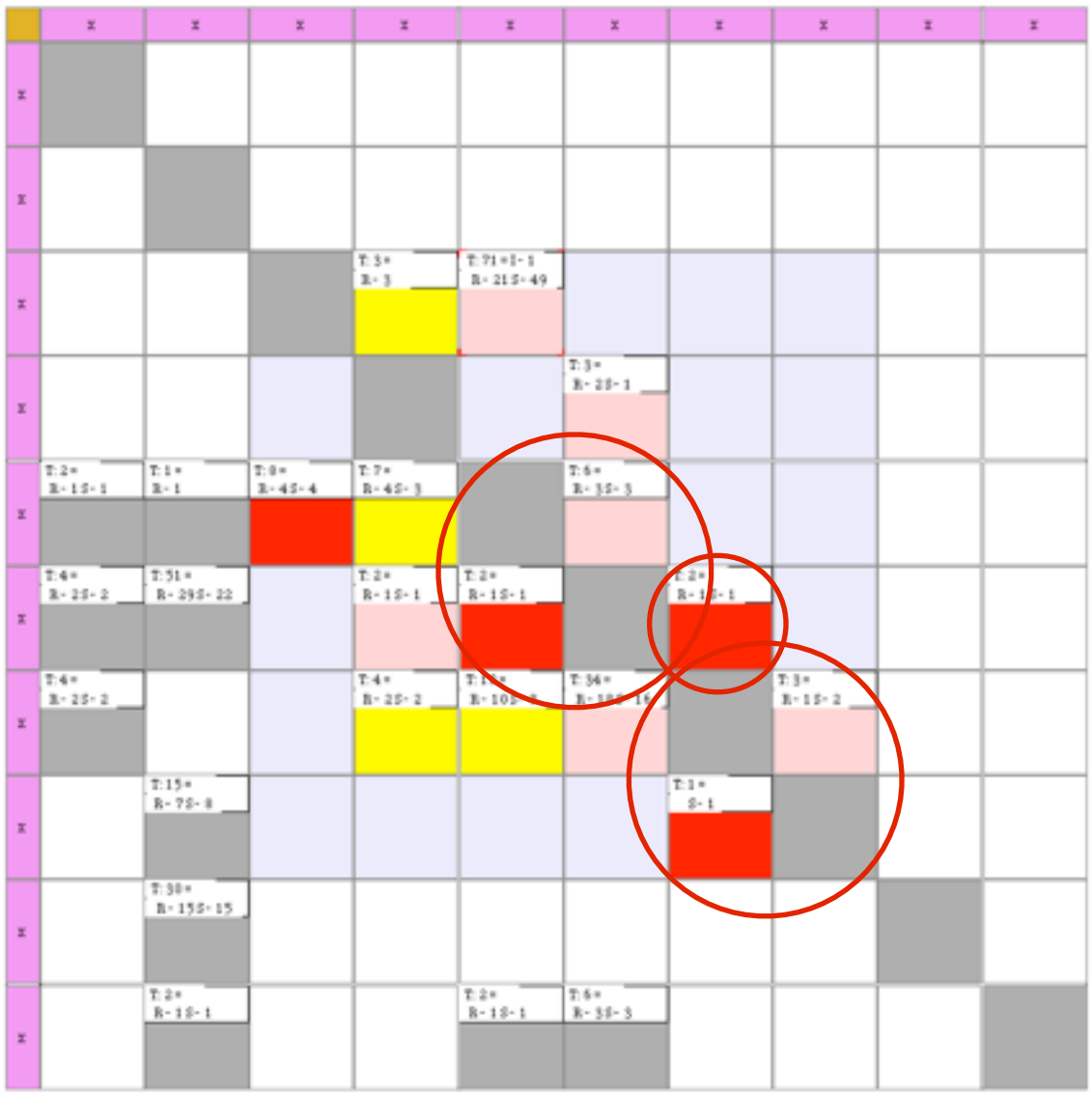
Identify cycles



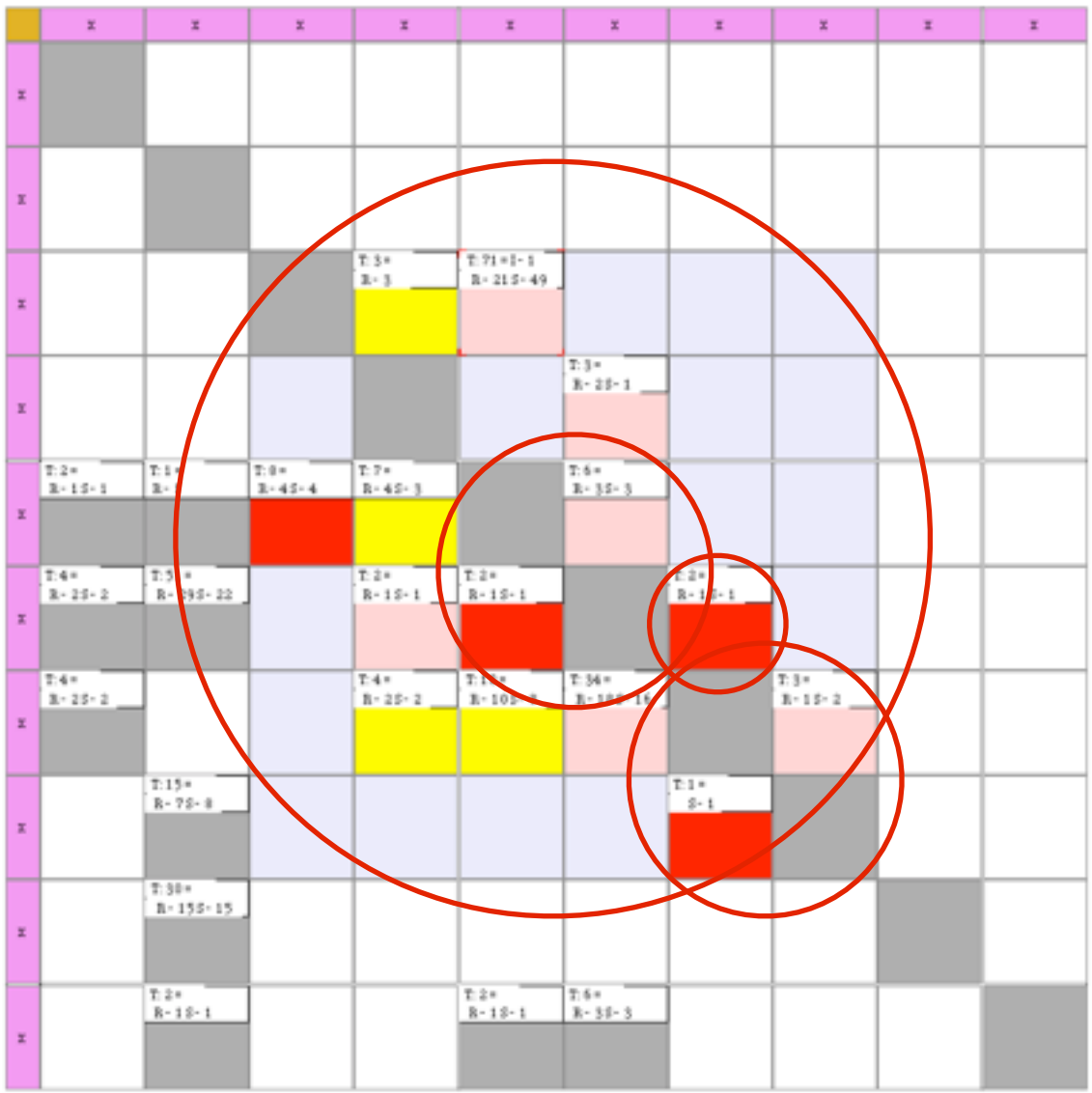
Identify cycles



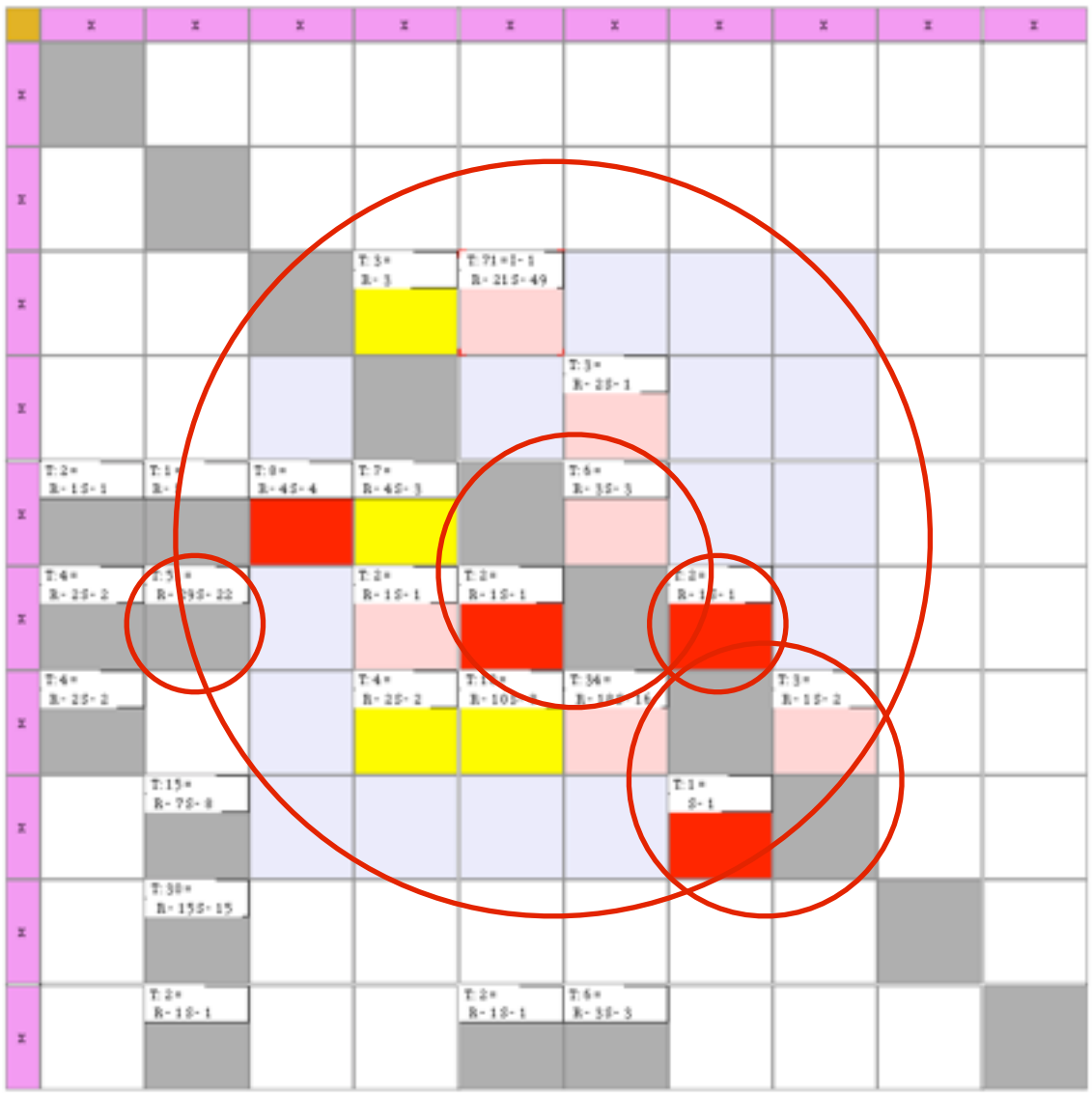
Identify cycles



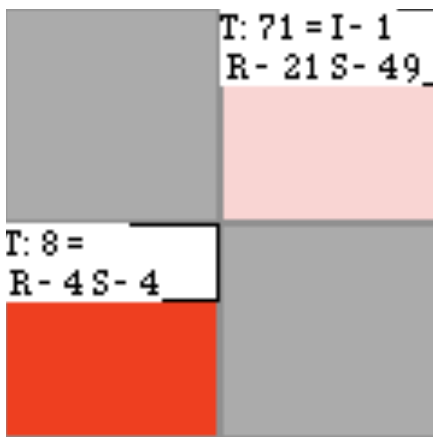
Identify cycles



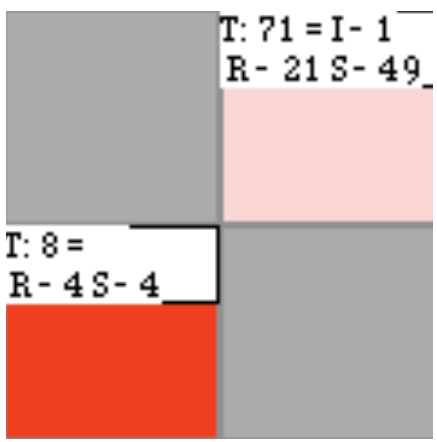
Identify cycles

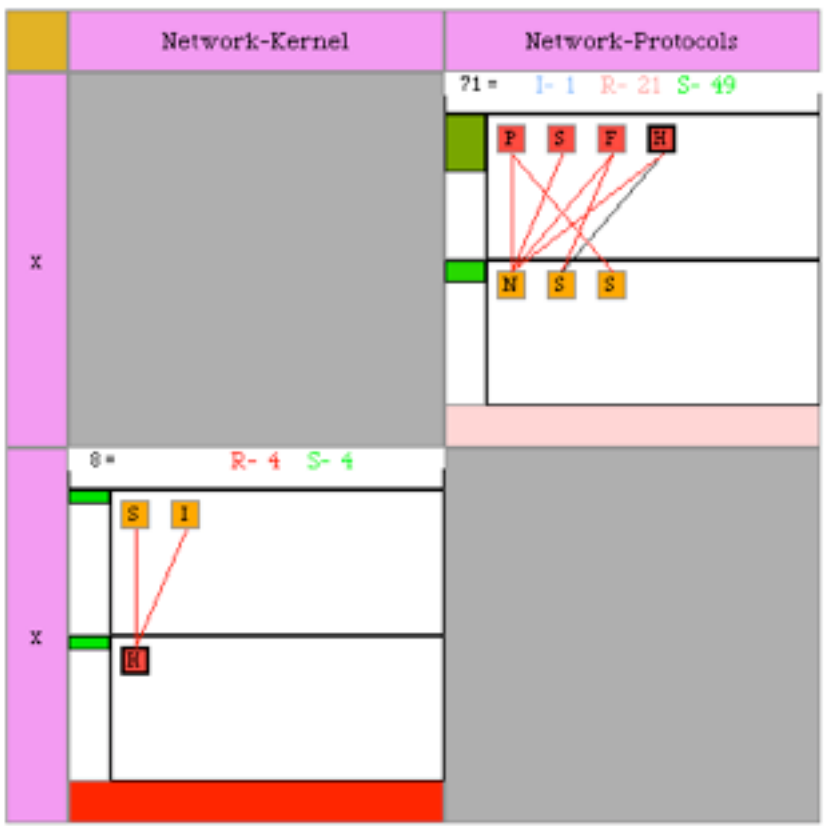


Causes and distribution



Causes and distribution






I: outgoing funnel
D: two classes referring each other
F: candidate for direct cycle fix
A: indirect cycle
E: high % of source
F: candidate for direct cycle fix
I: incoming funnel
G: invocations
B: complex cycle
B: complex cycle
E: high % of target impacted
C: accesses
C: accesses
H: inheritance + other

Language Independent

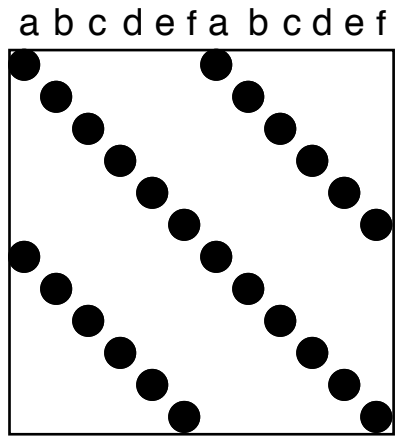
Language independent, Textual,
[ICSM'99], M. Rieger's PhD. Thesis

Duploc handled
Pascal, Java, Smalltalk, Python,
Cobol, C++, PDP-11, C

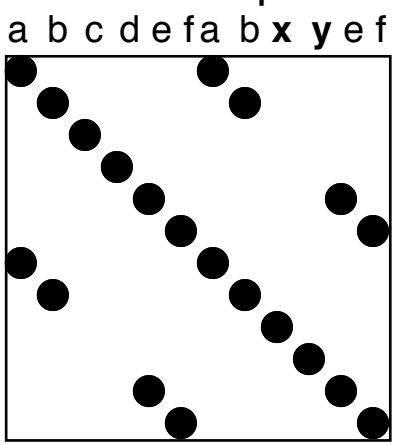
Slower than other approaches but...

Max 45 min to adapt our approach to
a new language

Between 3% and 10%
less identification than parametrized match

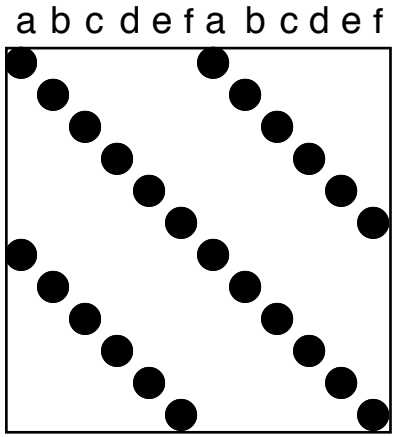


Exact Copies

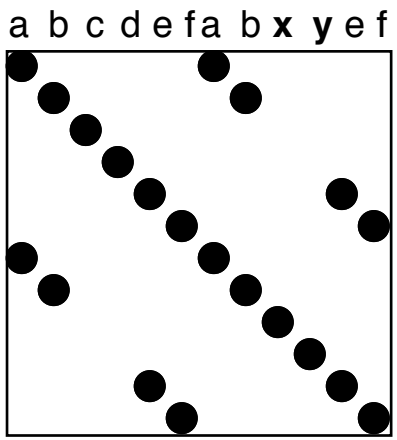


Copies with

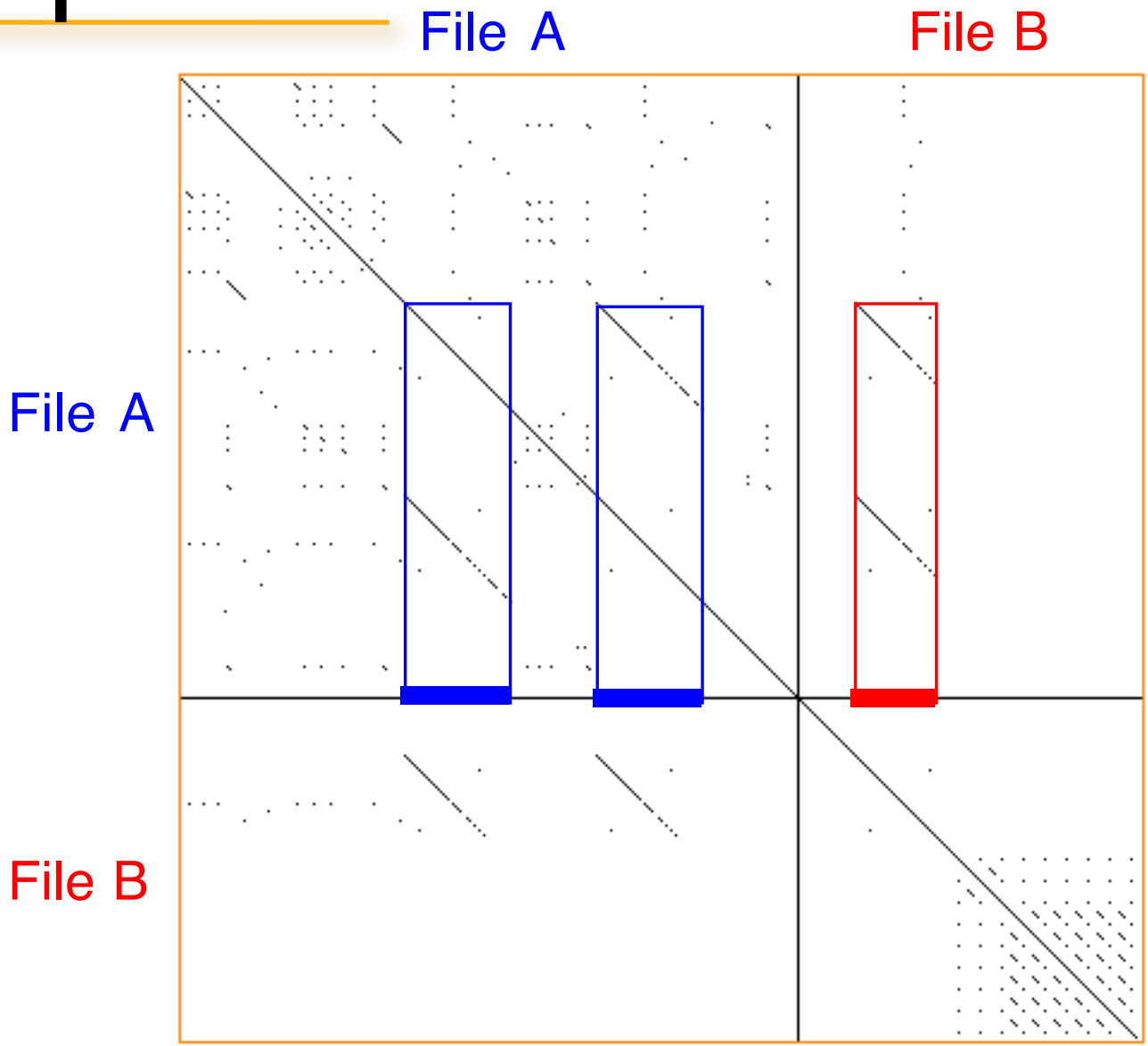
A Conceptual Matrix



Exact Copies



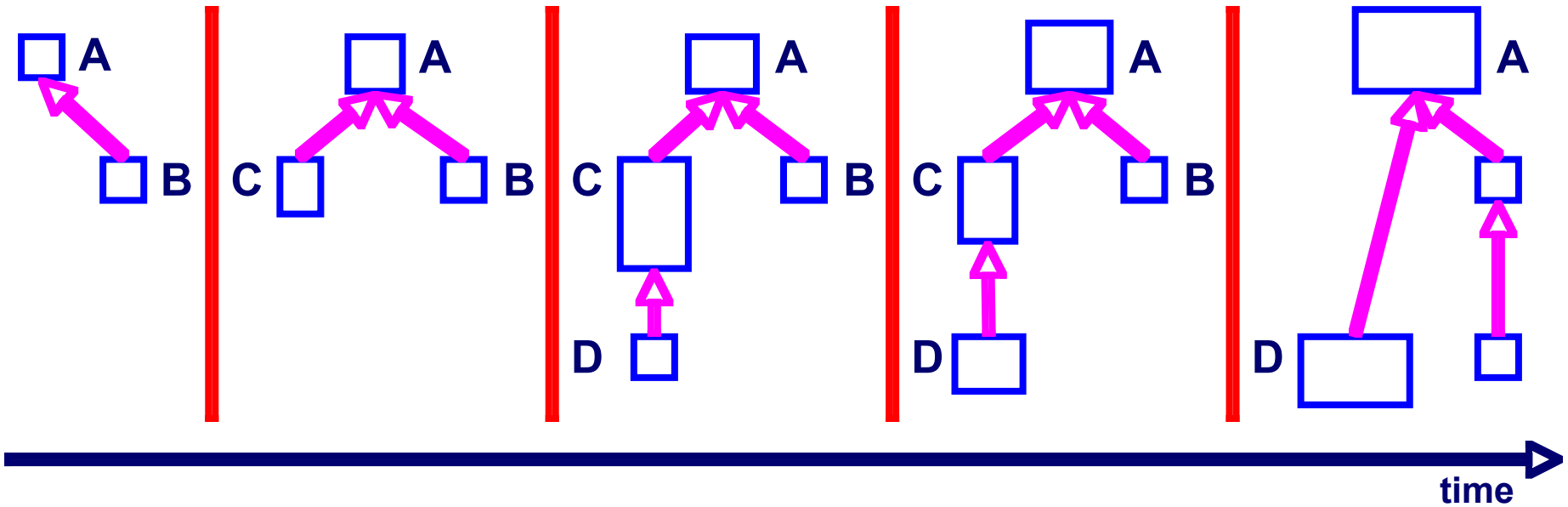
Copies with Variations



S.Ducasse

RMod

Evolution holds useful information



A is persistent

B is stable

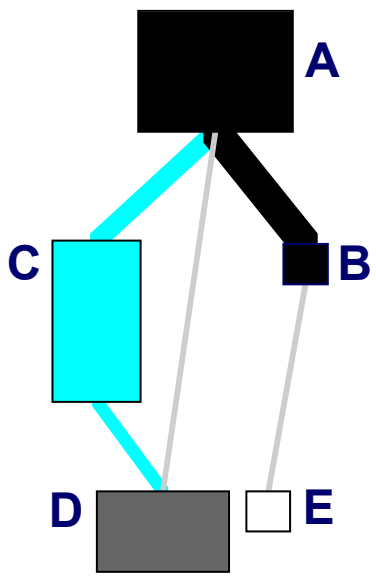
D inherited from C and then from A

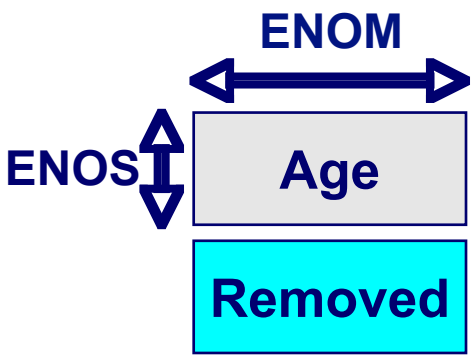
C was removed

E is newborn

…

Hierarchy Evolution Complexity View characterizes class hierarchy histories





Class History

Inheritance History

A is persistent

B is stable

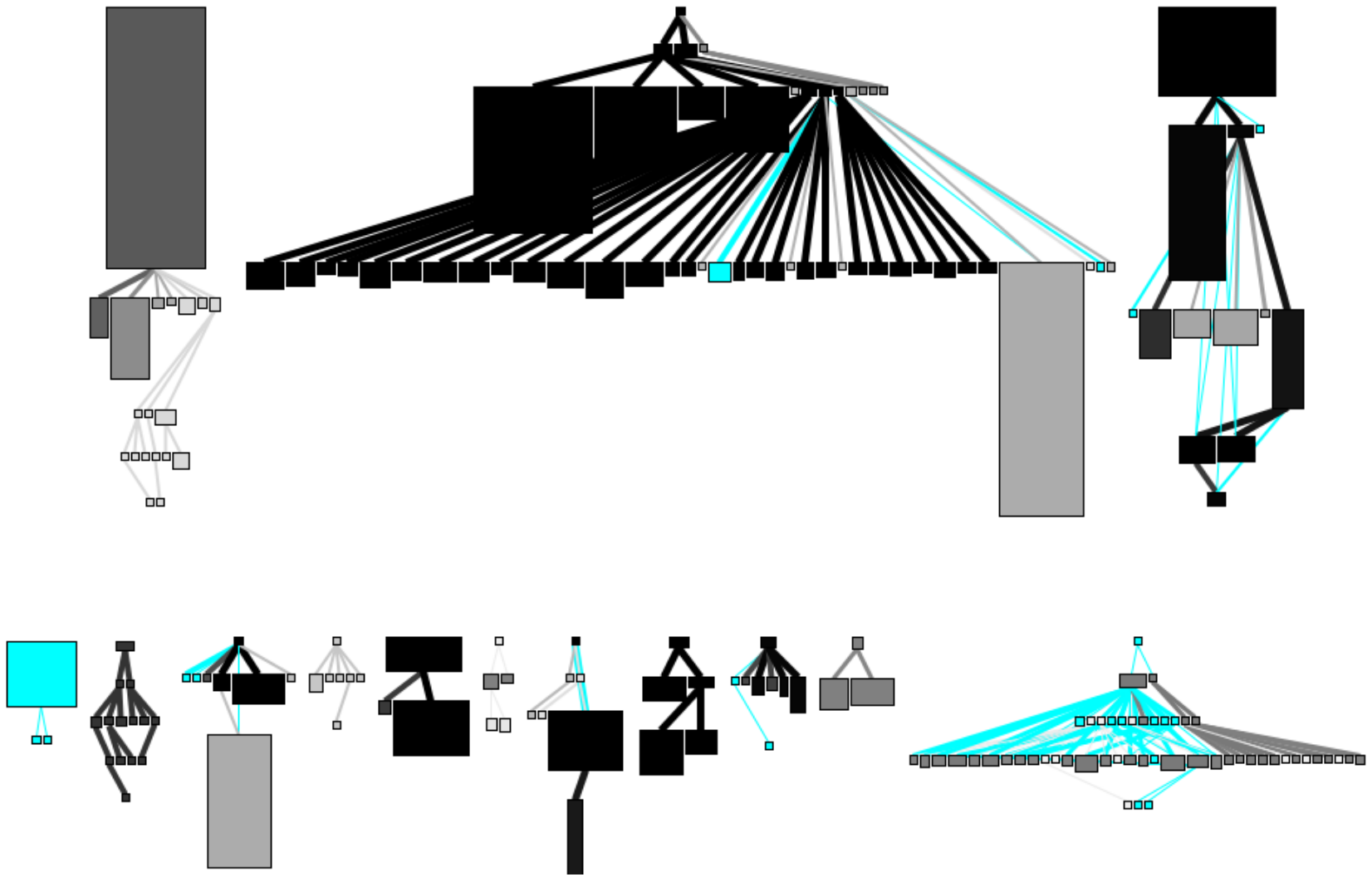
D inherited from C and then from A

C was removed

E is newborn

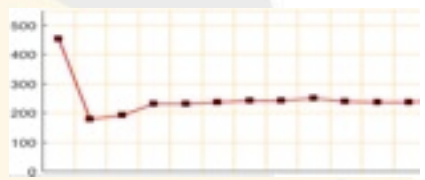
…

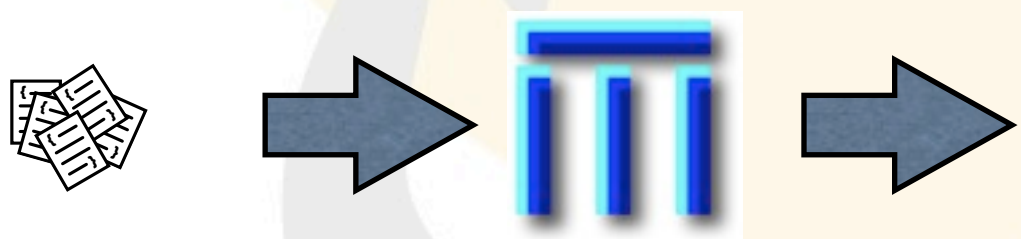
Class hierarchies over 40 versions of Jun - a 740 classes, 3D framework



Evolution is difficult

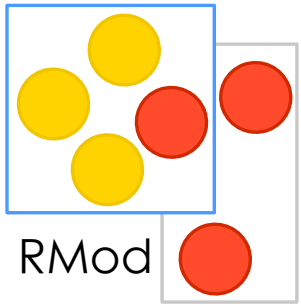
- We are interested in ***your*** problems!
- Moose is open-source, you can use it, extend it, change it
- We can collaborate!





NOM > 10 &
LOC > 100

Axis 2: Dynamic Languages



Revisiting fundamental aspects of OO languages

Reuse Traits: Fortress (SUN Microsystems), Perl-6, Scala (EPFL), Squeak, Dr-Scheme,

Security and Dynamic Languages

Axis 2: Dynamic Languages Infrastructure

La perfection est atteinte, non pas lorsqu'il n'y a plus rien à ajouter, mais lorsqu'il n'y a plus rien à retirer. St-Exupery

Topics

Components for field devices (Pecos IST Project)

Classboxes: Modules for open-classes [OOPSLA'05]

OOPAL: OOP + APL Generalizing message passing [OOPSLA'03]

Language symbiosis (Jour. Program)

Encapsulation for dynamic languages [ECOOP '04, OOPSLA'04]

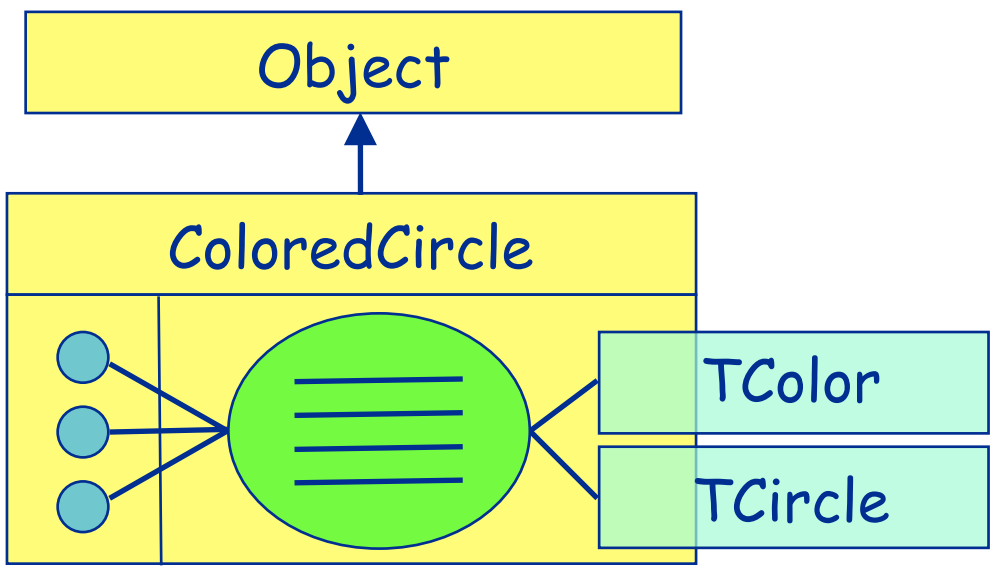
Reusable behavior: ***Traits*** [ECOOP'03, OOPSLA'03, Toplas, ..., OOPSLA'07]

Impacts

Traits used by Fortress (***SUN Microsystems***), Scala (EPFL), Perl-6, Squeak, Slate, Dr-Scheme, Multiple type systems (Drossopoulos, Reppy, Liquori, Bono...)

Reconciling reuse and single inheritance

class = superclass + state + traits + glue



Contributions

Traits

Stateful traits

Freezable traits

Impacts

Fortress (SUN Microsystems), Scala (EPFL), Perl-6, Squeak, Slate, Dr-Scheme

Multiple type systems (Drossopoulos, Reppy, Liquori, Bono...)

Class

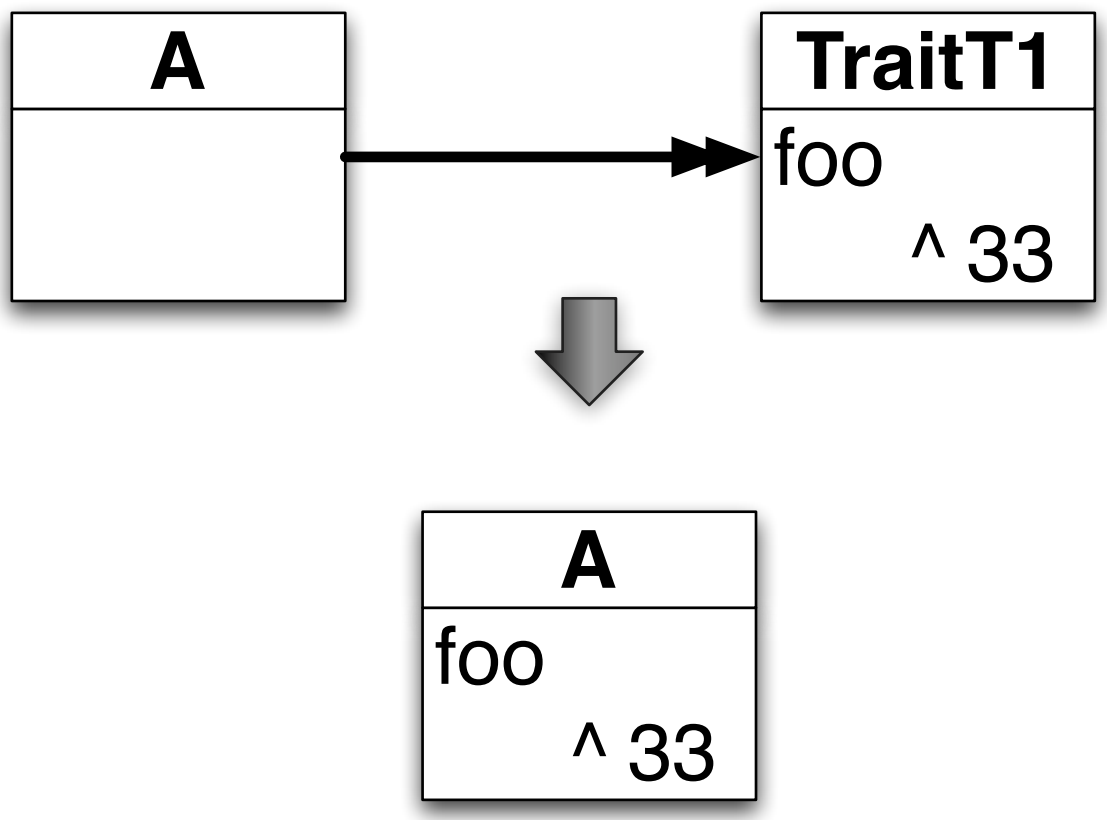
=

Superclass
+ State
+ Traits
+ Methods

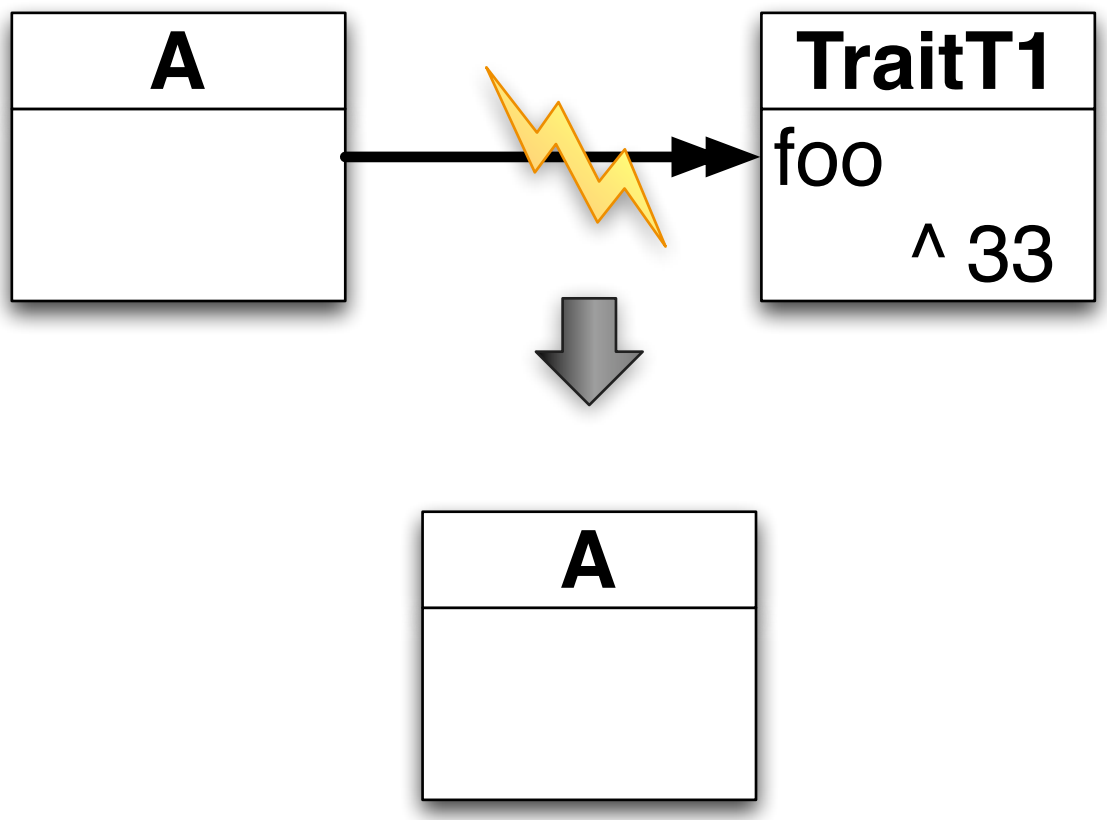
Traits do ***NOT*** exist at runtime

- Traits are like macros
- Method defined in class take precedence over trait methods

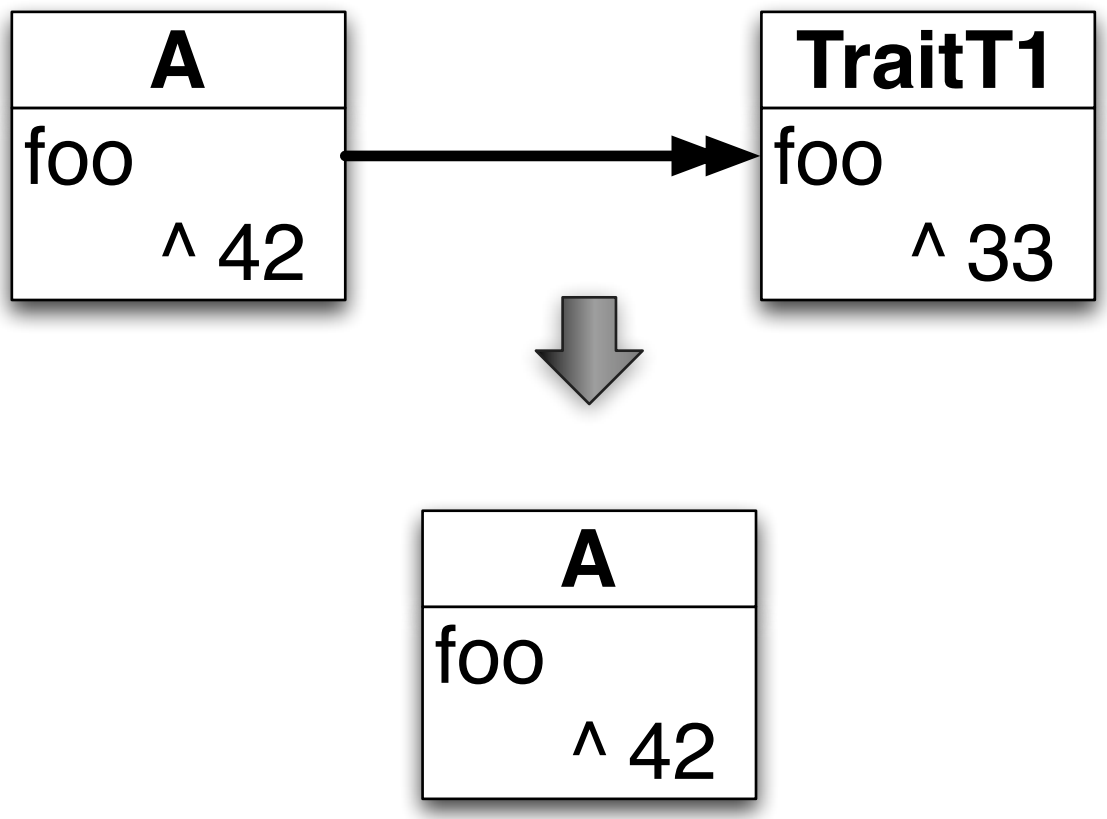
Using T I



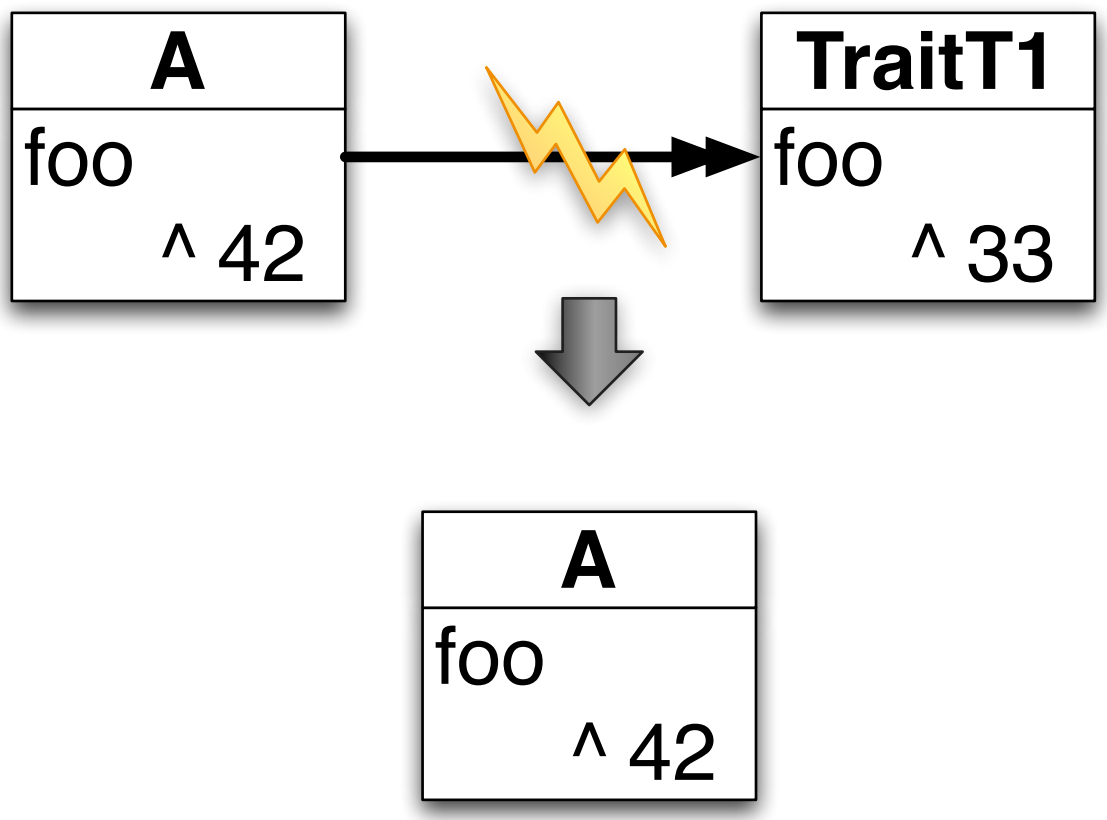
Not using anymore T1

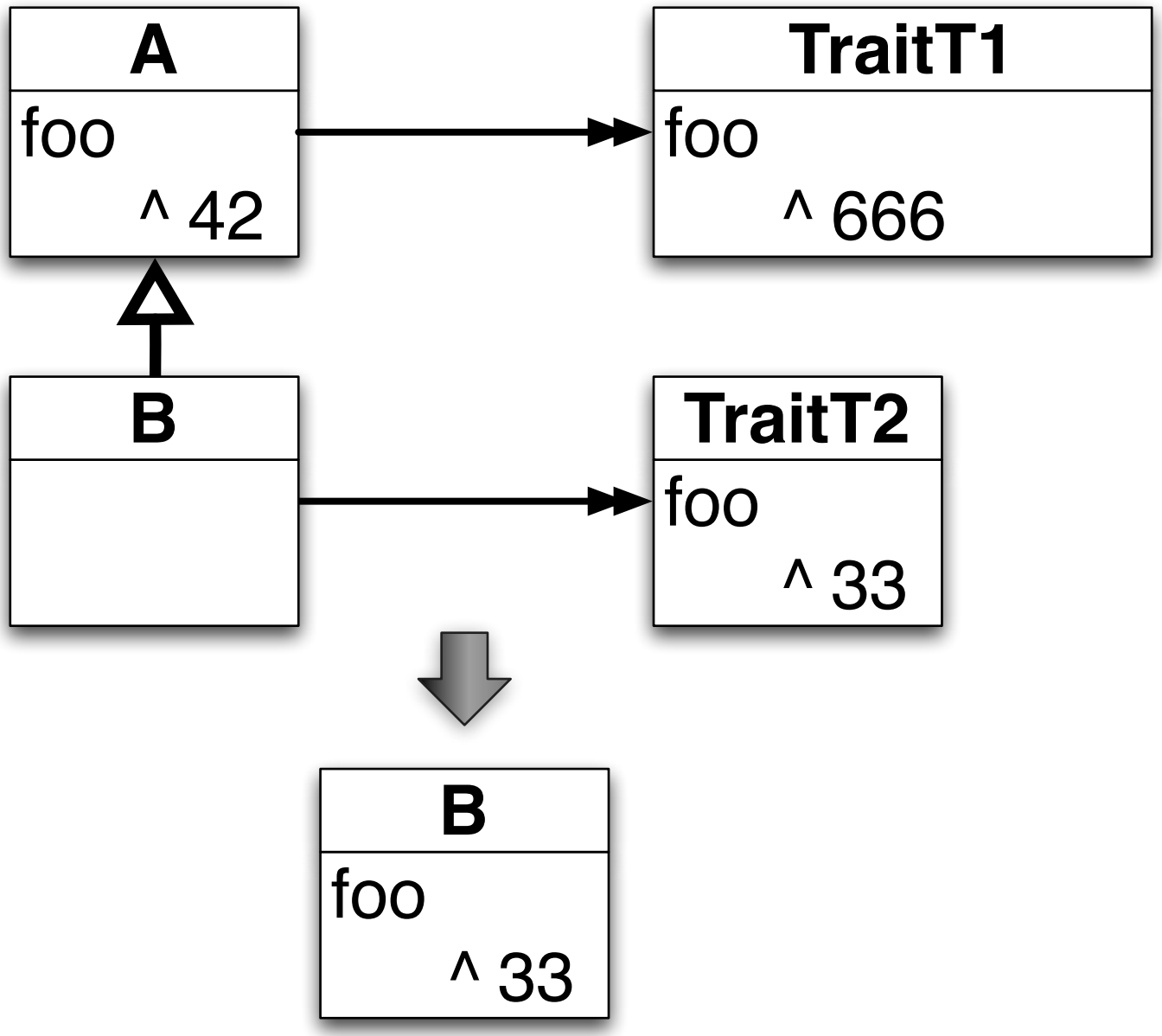


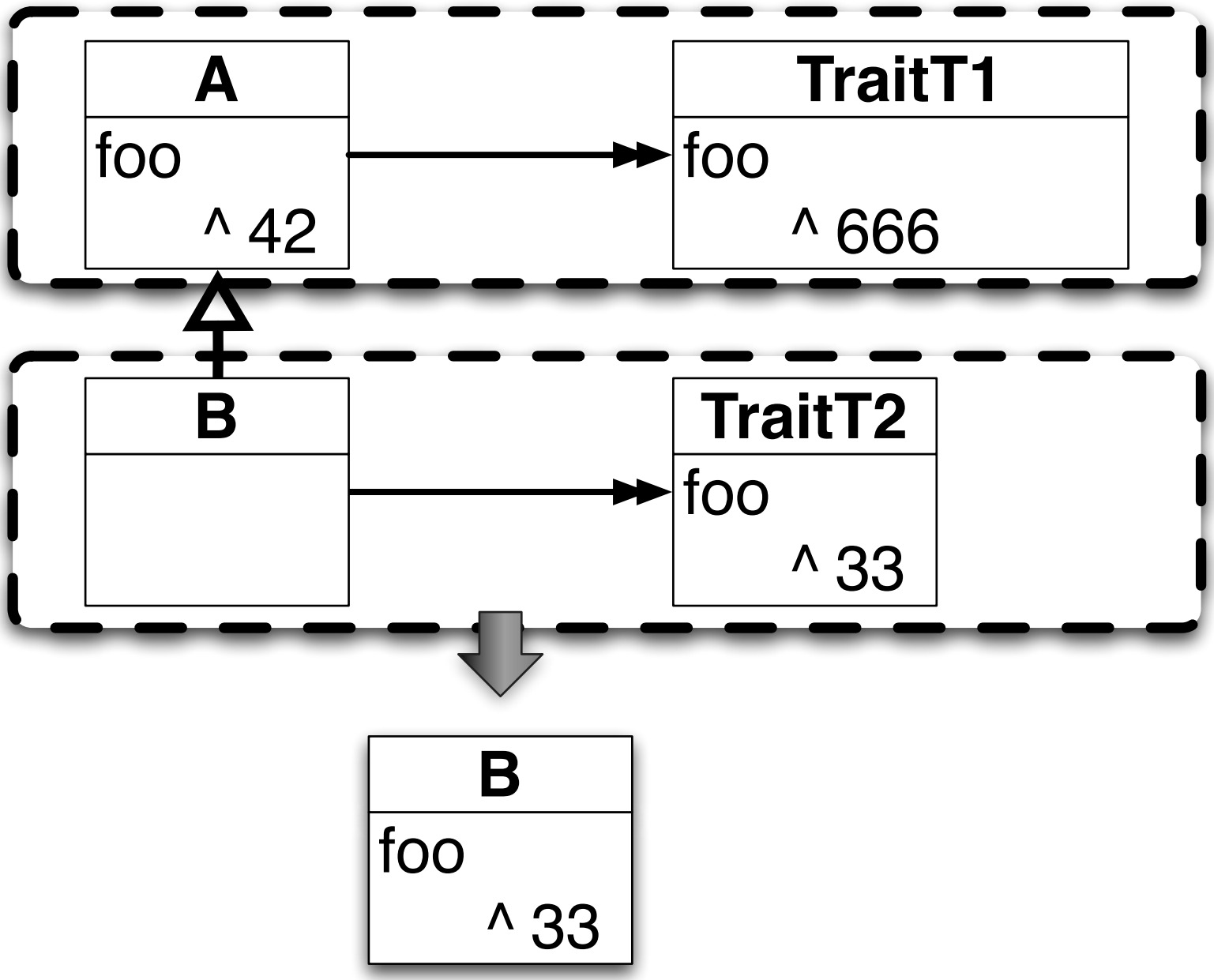
Composer has power

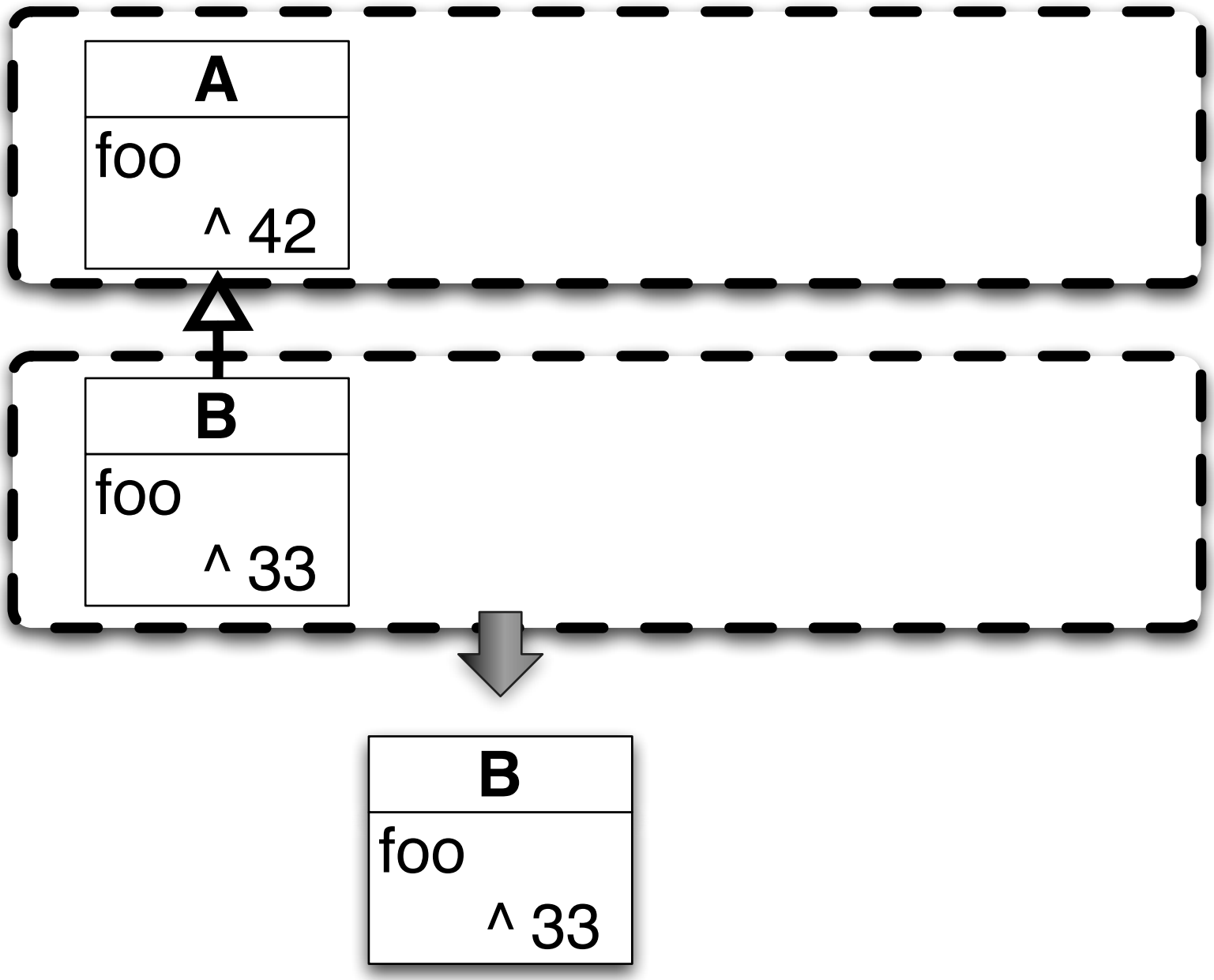


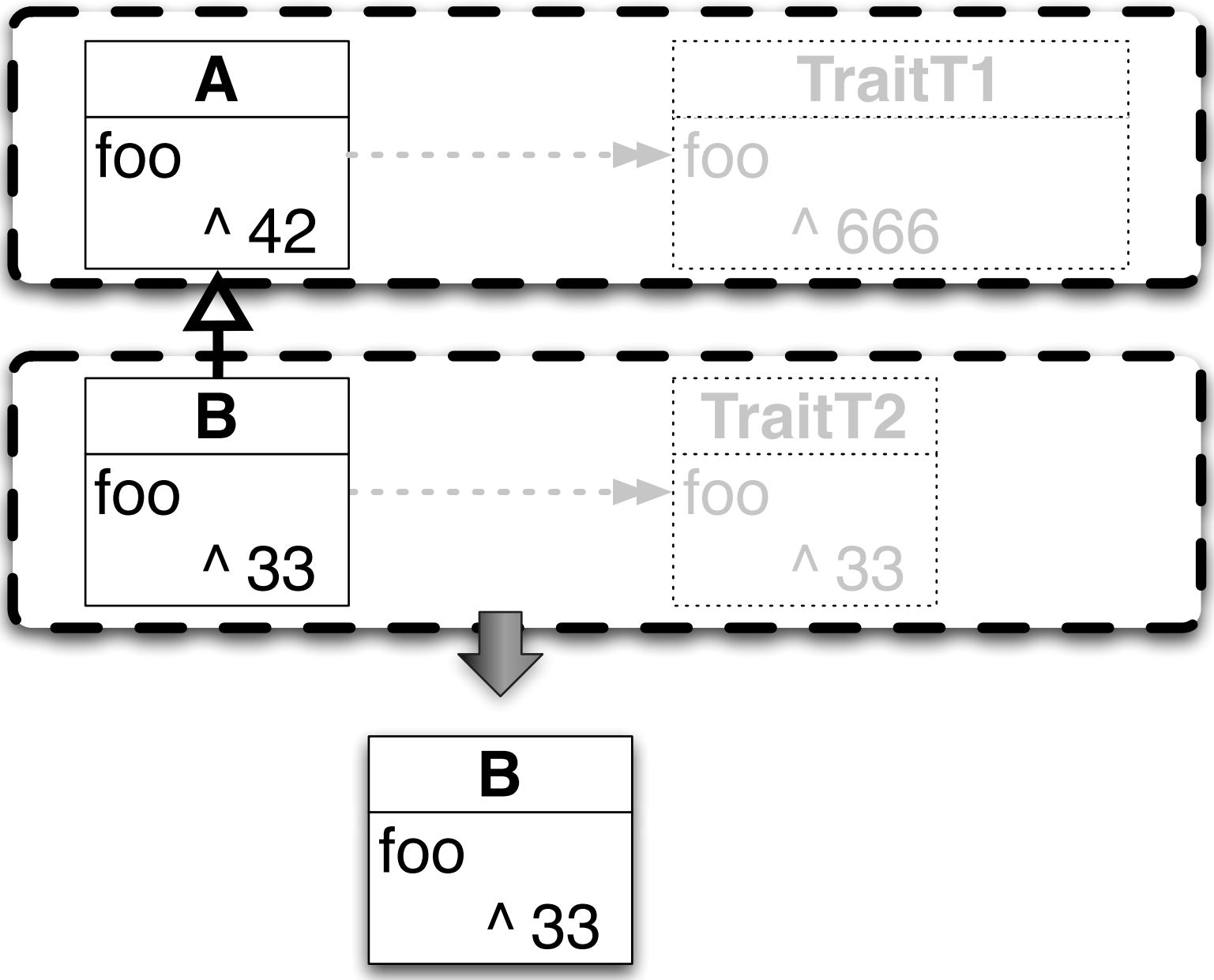
Composer has power



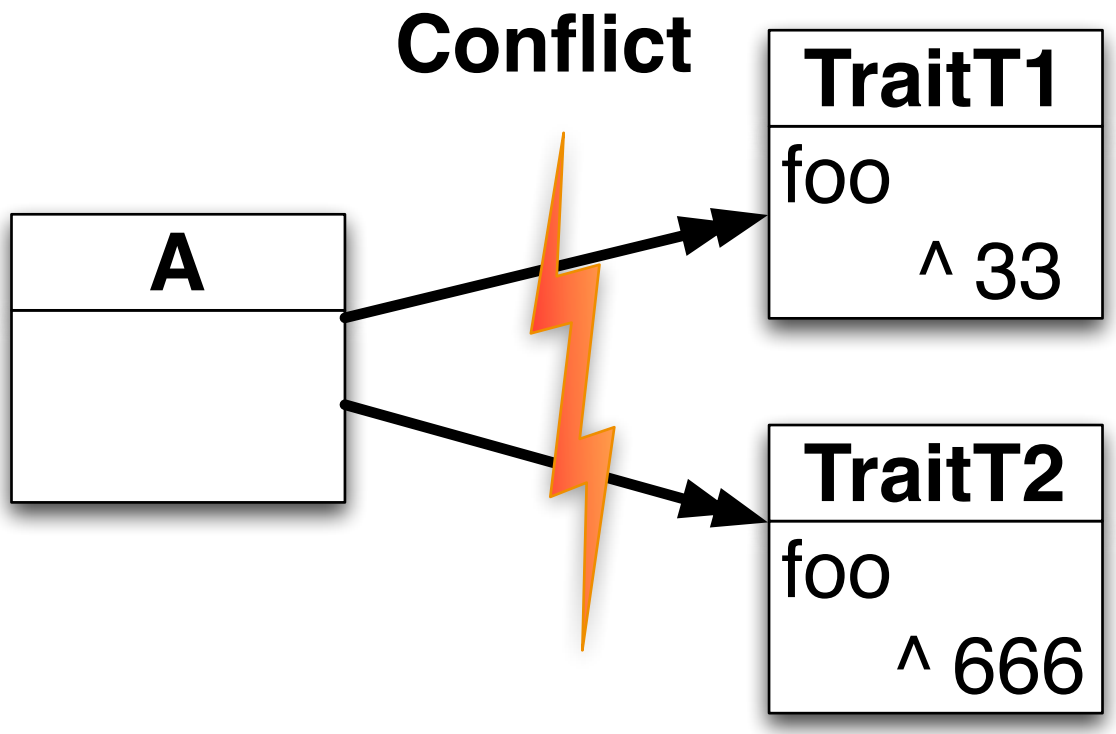

A
foo
^ 42
TraitT1
foo
^ 666
B
TraitT2
foo
^ 33
B
foo
^ 33


A
foo
^ 42
TraitT1
foo
^ 666
B
TraitT2
foo
^ 33
B
foo
^ 33

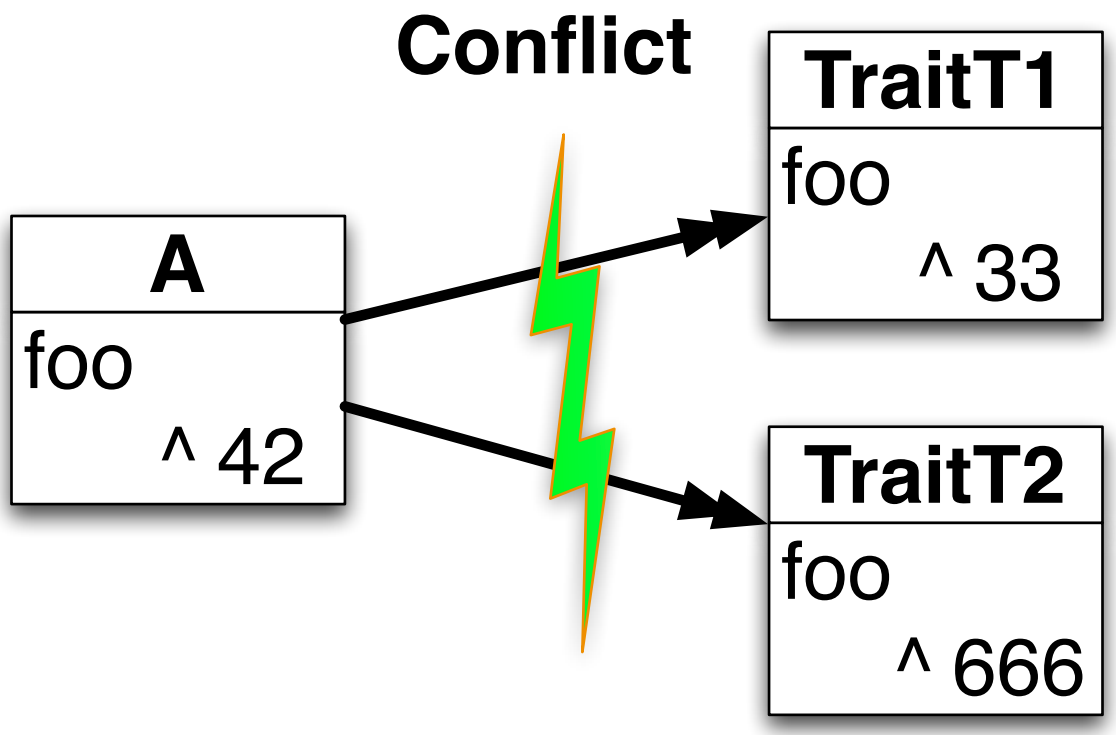

A
foo
^ 42
B
foo
^ 33
B
foo
^ 33


A
foo
^ 42
TraitT1
foo
^ 666
B
foo
^ 33
TraitT2
foo
^ 33
B
foo
^ 33

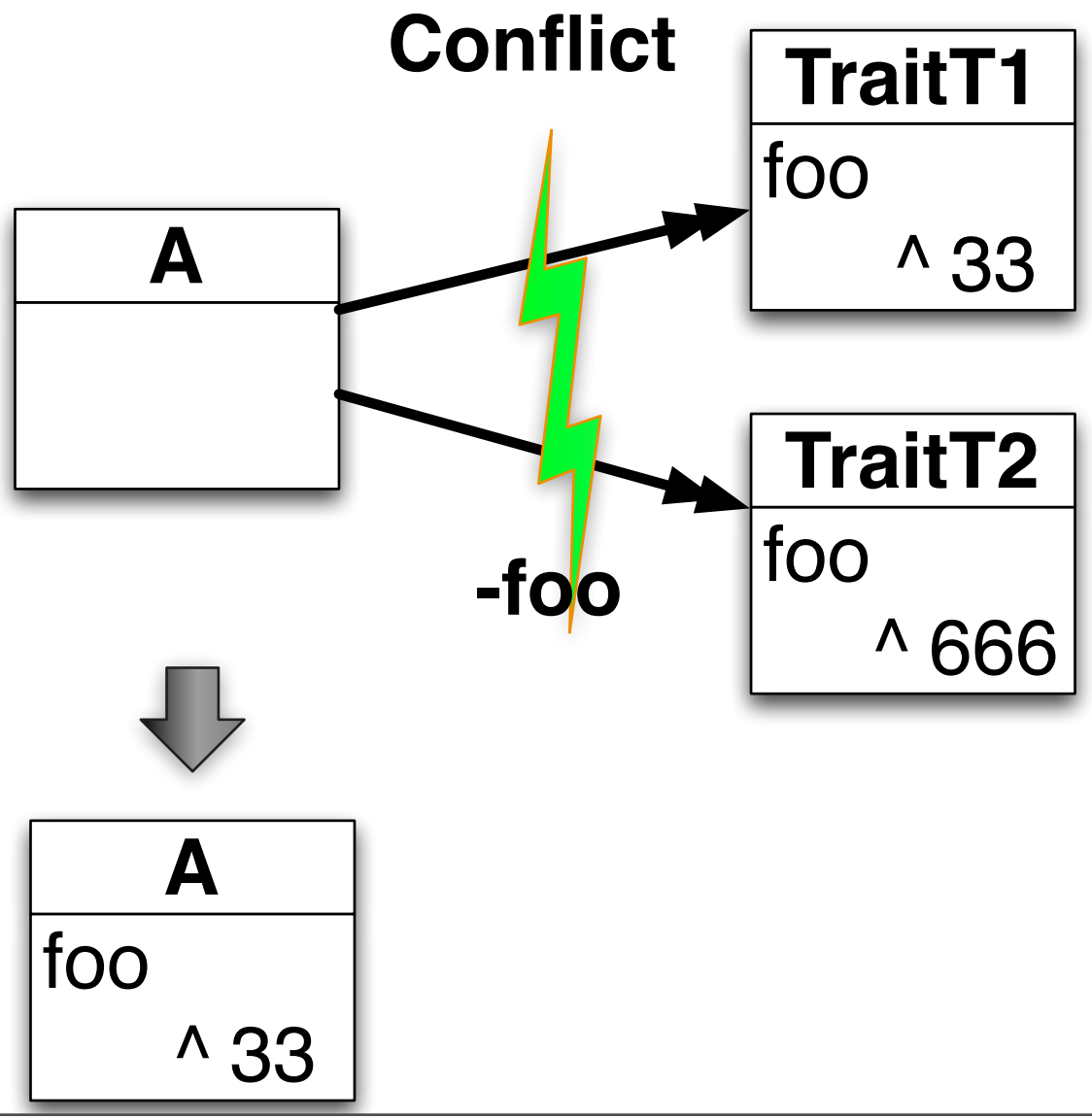
Conflicts



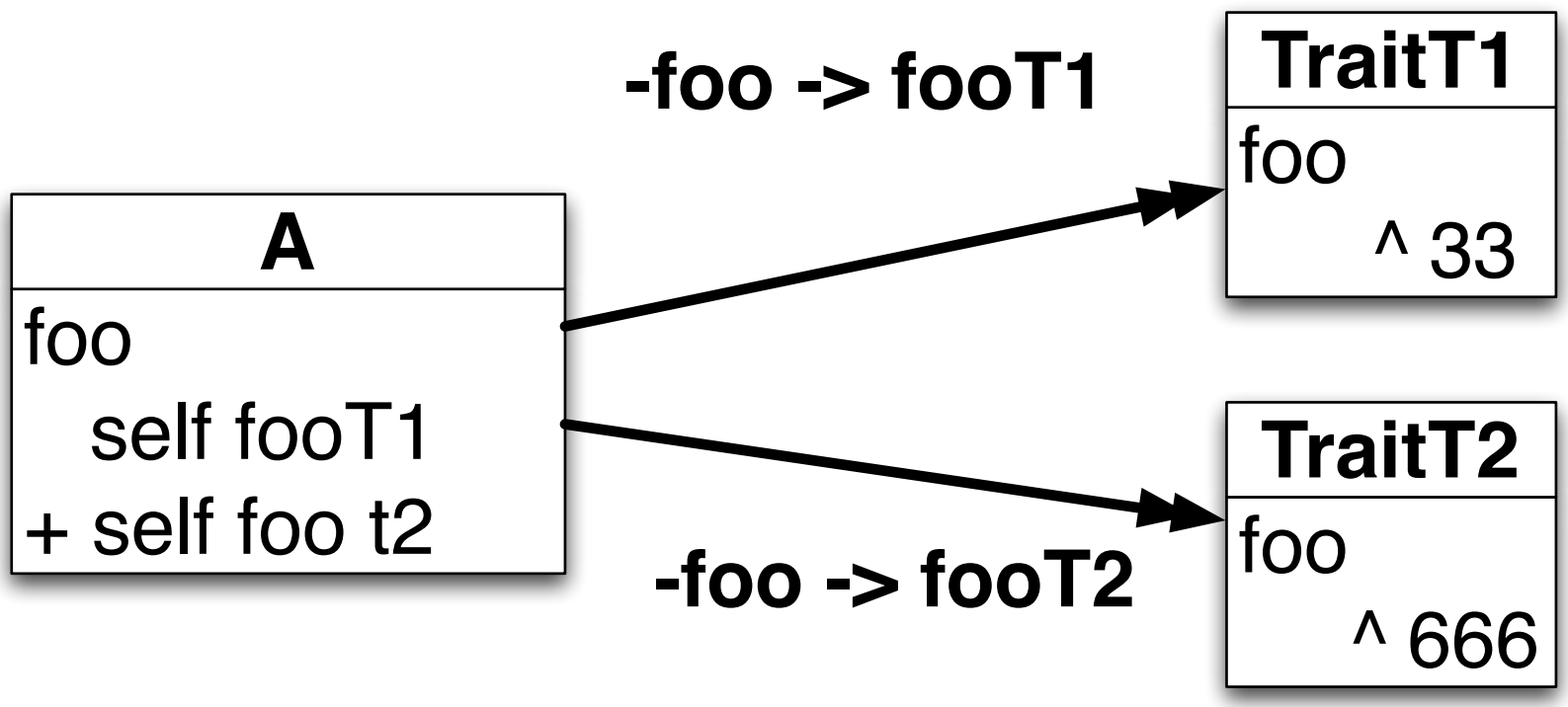
Resolved: “Overrides”



Resolved: Ignore



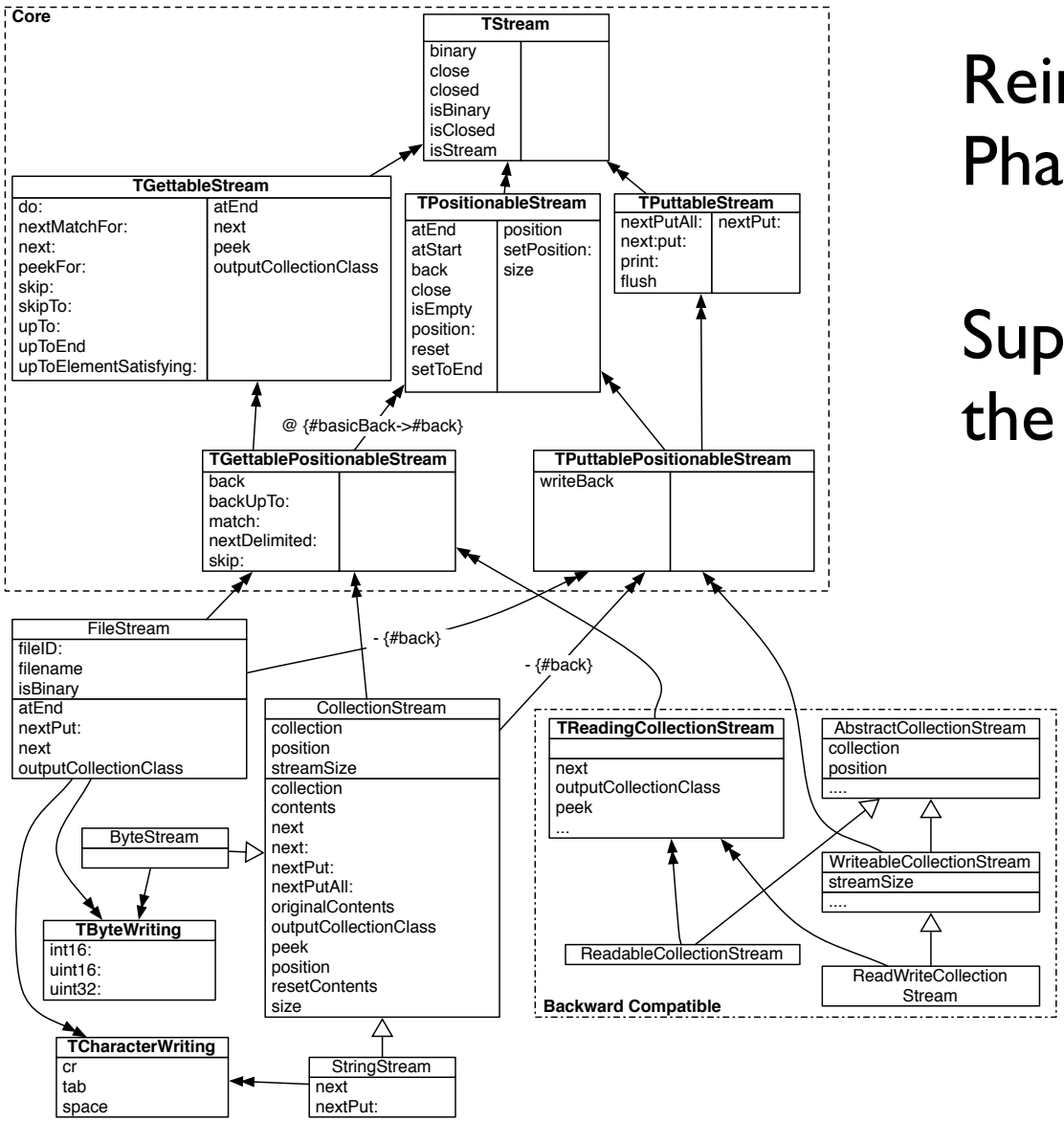
Access to ignored methods



Applications

- Building tests out of common traits
- Nile
- Polymorph
- Miro
- Large BBC software in Perl

Nile



Reimplementing streams in Pharo and Squeak

Supports old and new styles with the same traits recomposed

Traits

Implemented in Squeak/Pharo Smalltalk

- Fully backwards compatible
- No performance penalty for method lookup
- Refactored Streams
- Collection tests

In Scala (but looks more like mixins)

Replace classes in Fortress (SUN MicroSystems)

Introduced in Perl6, Slate, DrScheme, AmbiantTalk,

May be in Javascript!

Conclusion

Better tools and approaches to deal with complex system
http://moose.unibe.ch

Better languages for developing better applications