

Source-code quality

Part 2. Software Visualizations

Andy Kellens



*“If I could say it in words, there would be no
reason to paint”
-- Edward Hopper*

In this lecture ...

- ▶ **What is (software) visualization?**
- ▶ **What makes a good visualization?**
- ▶ **Software visualizations**
- ▶ **Moose tool suite**
- ▶ **Build your own visualization**

What is software visualization?

What is software visualization?

► Use of graphic means

- Typography
- Graphic design
- Animation
- ...

► To facilitate human understanding and effective use of computer software

(Stasko et al. , Software Visualization - Programming as a Multimedia Experience, MIT Press, 1998)

Why Software Visualizations?

- ▶ **Process large quantities of information**
- ▶ **Human brain good at combining complex information from visual clues**
- ▶ **Pre-attentative process: draw attention to important parts**

Need for tools

- ▶ **Manual effort**
 - Review meetings
 - Find quality issues
 - 2 seconds per line of code
 - 250 KLOC * 2
 - 500 000 seconds
 - 140 hours
 - 18 days

Need for tools that help in assessing quality attributes



The diagram illustrates the manual effort required for code review. It lists 'Manual effort' with sub-points: 'Review meetings', 'Find quality issues', and '2 seconds per line of code'. Below this, it calculates the total effort: '250 KLOC * 2', '500 000 seconds', '140 hours', and '18 days'. To the right, there are book covers for 'BORNA & MRP', 'The Fellowship of the Ring', 'The Hobbit', 'The Lord of the Rings', and 'ULYSSES'. A wrench and gear icon is at the bottom right.

For example: commit logs

r19692 | cderoove | 2012-02-19 20:28:21 +0100 (Sun, 19 Feb 2012) | 2 lines

intro of three

r19691 | cderoove | 2012-02-19 19:14:58 +0100 (Sun, 19 Feb 2012) | 2 lines

minor changes to section 2 + bib

r19690 | cderoove | 2012-02-19 18:44:00 +0100 (Sun, 19 Feb 2012) | 2 lines

proper (;)) capitalization of SOOT and SOUL

r19689 | cderoove | 2012-02-19 18:32:59 +0100 (Sun, 19 Feb 2012) | 2 lines

soul section

r19681 | akellens | 2012-02-17 15:52:38 +0100 (Fri, 17 Feb 2012) | 1 line

Bigger boxplots

r19680 | cnoguera | 2012-02-17 15:39:29 +0100 (Fri, 17 Feb 2012) | 2 lines

Added ref to acceleo, also explained a bit more on the transformation

r19679 | akellens | 2012-02-17 14:53:39 +0100 (Fri, 17 Feb 2012) | 1 line

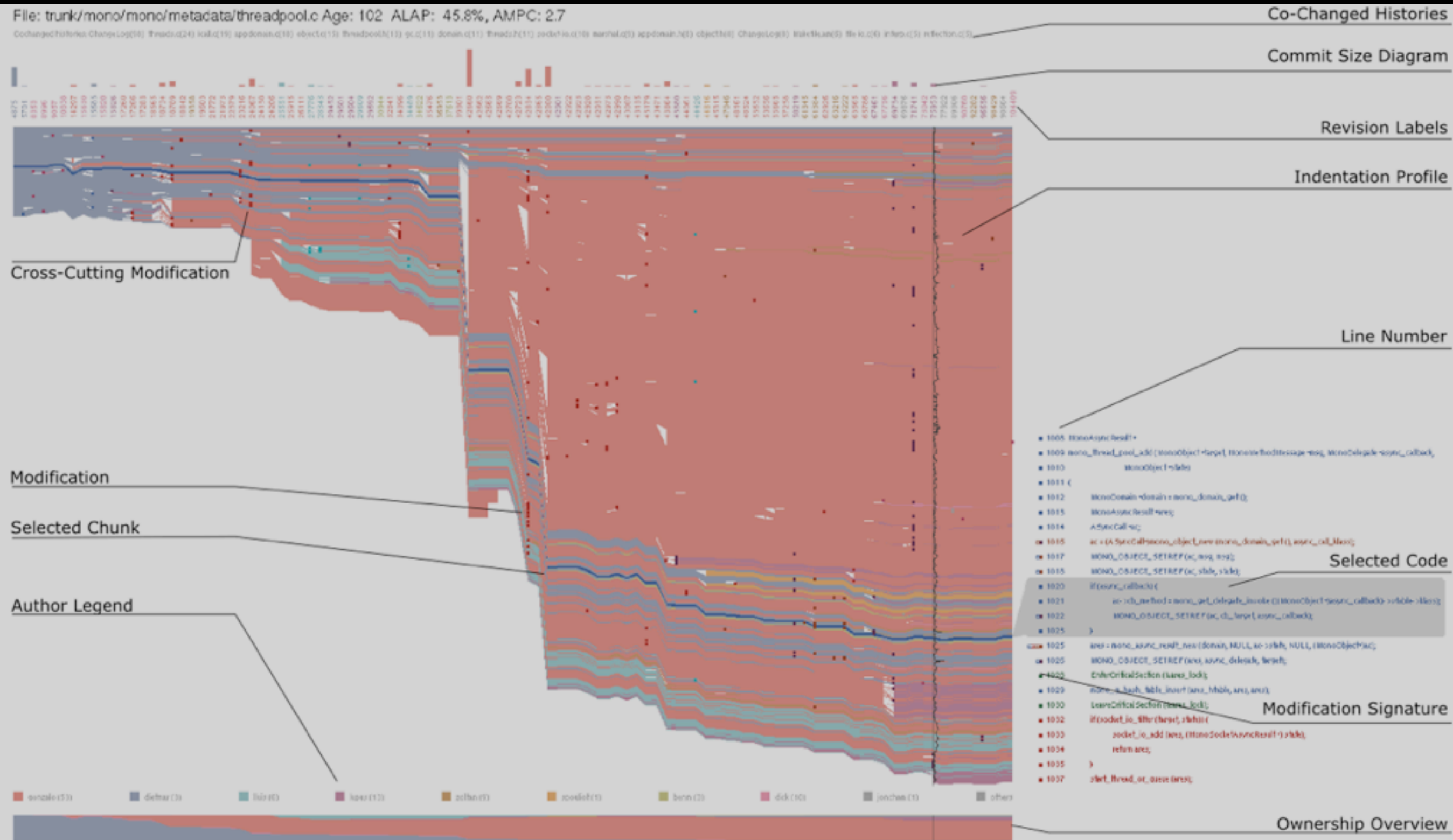
Iteration done

r19678 | cnoguera | 2012-02-17 14:44:16 +0100 (Fri, 17 Feb 2012) | 2 lines

Structured data

What can we see?

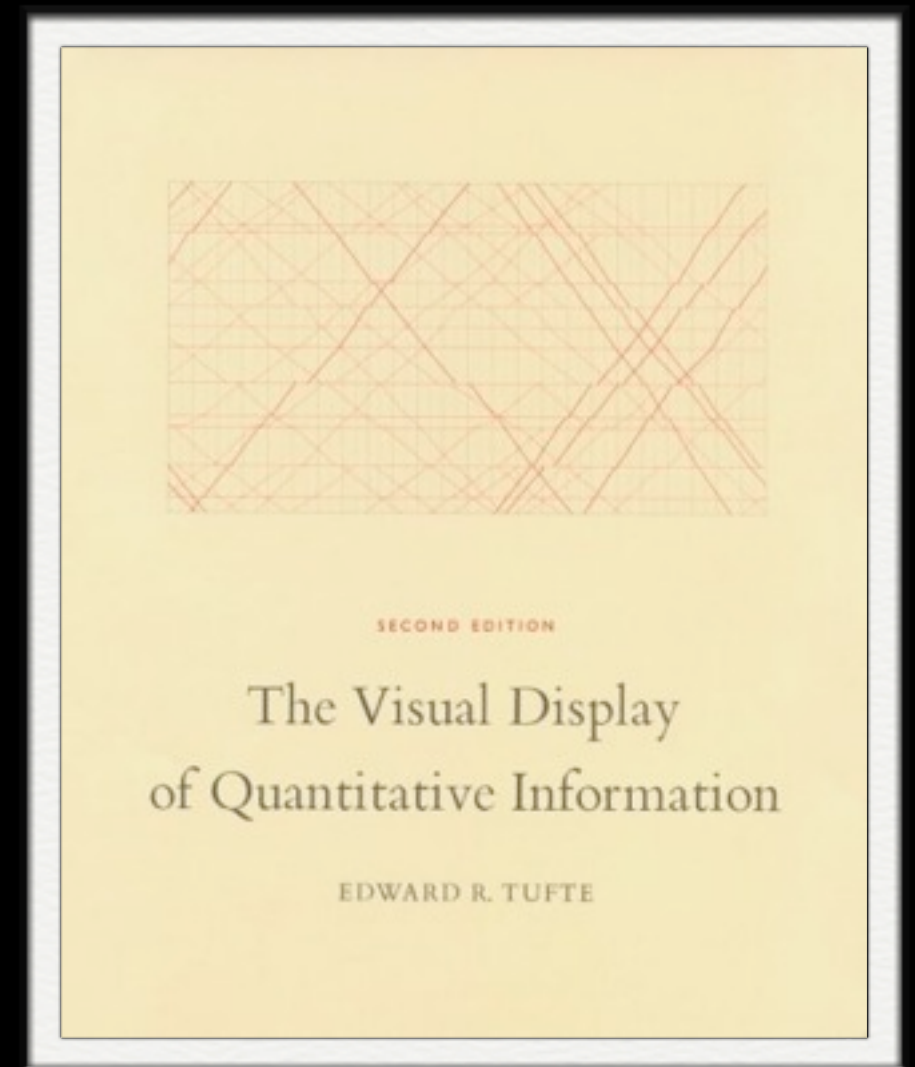
Visualizing commits



Properties of good visualizations

What makes a good visualization?

- ▶ **Designing good visualizations not trivial**
- ▶ **Sufficiently rich to convey info**
- ▶ **Not overwhelming**
- ▶ **Studied in cognitive sciences**



(Edward Tufte, 1983)

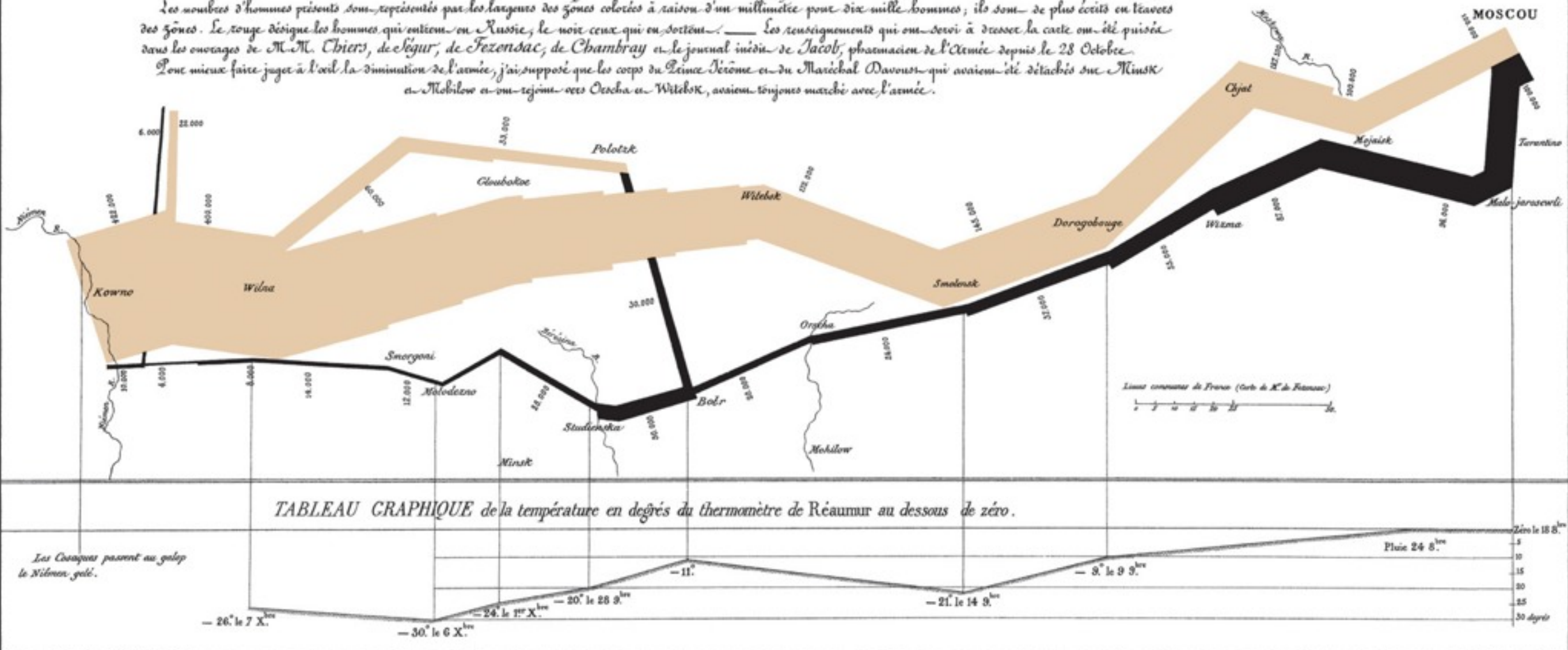


Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813.

Dessiné par M. Minard, Inspecteur Général des Ponts et Chaussées en retraite. Paris, le 20 Novembre 1869.

Les nombres d'hommes présents sont représentés par les largeurs des zones colorées à raison d'un millimètre pour dix mille hommes; ils sont de plus écrits en lettres des zones. Le rouge désigne les hommes qui entrent en Russie, le noir ceux qui en sortent. Les renseignements qui ont servi à dresser la carte ont été puisés dans les ouvrages de M. M. Chiers, de Ségur, de Fezensac, de Chambray et le journal inédit de Jacob, pharmacien de l'Armée depuis le 28 Octobre.

Pour mieux faire juger à l'œil la diminution de l'armée, j'ai supposé que les corps du Prince Jérôme et du Maréchal Davoust qui avaient été détachés sur Minsk et Mohilow et qui rejoignent vers Orescha et Witebsk, avaient toujours marché avec l'armée.



(Charles Minard, 1861)

Visualization Pitfalls

▶ Colors:

- Attribute meaning to color
- Max. 10 colors

▶ Mapping to reality:

- Link with domain concepts; intuitive mapping

▶ Information density:

- Every visual element should have a meaning

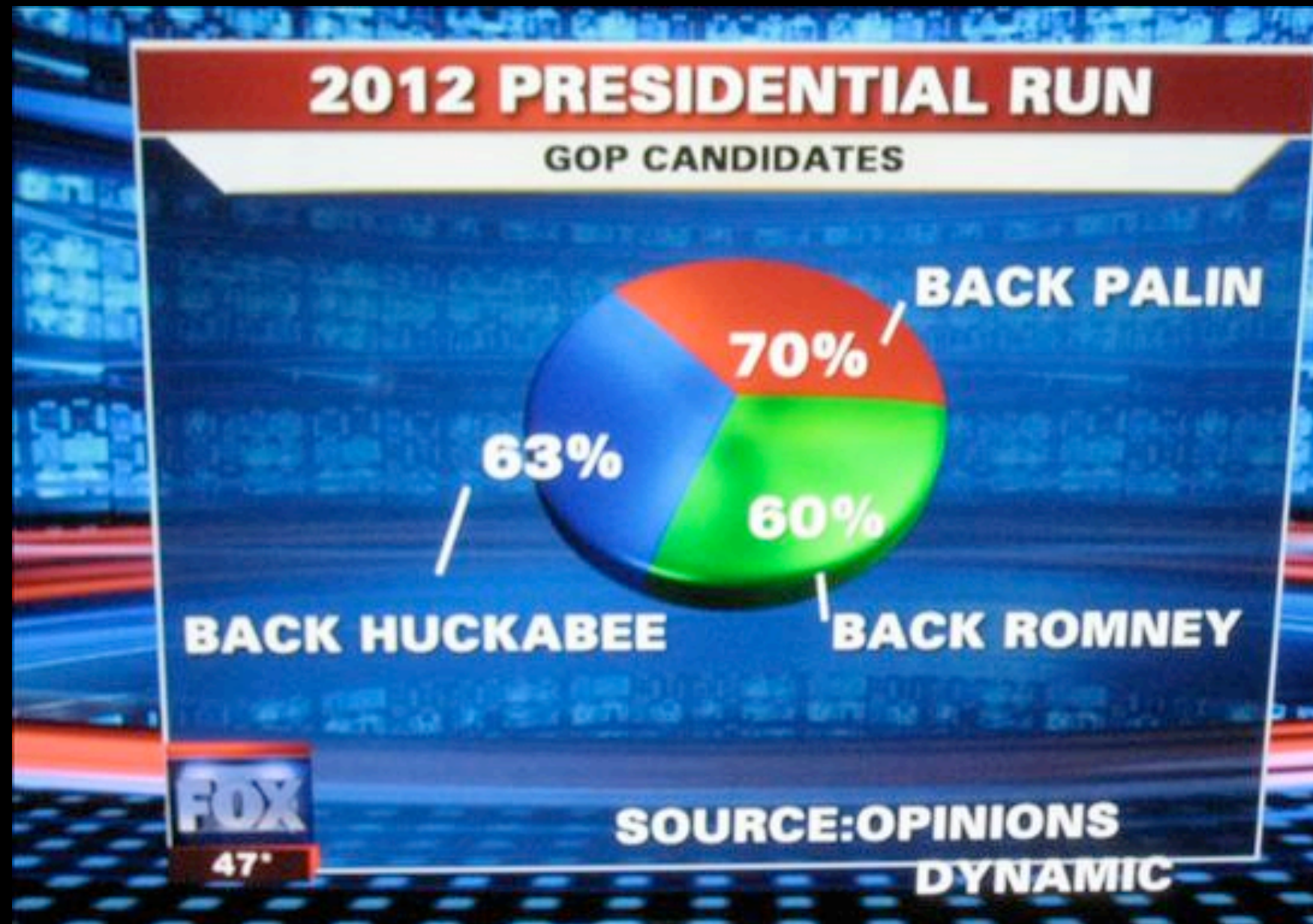
▶ Scalability:

- Deal with lots of data

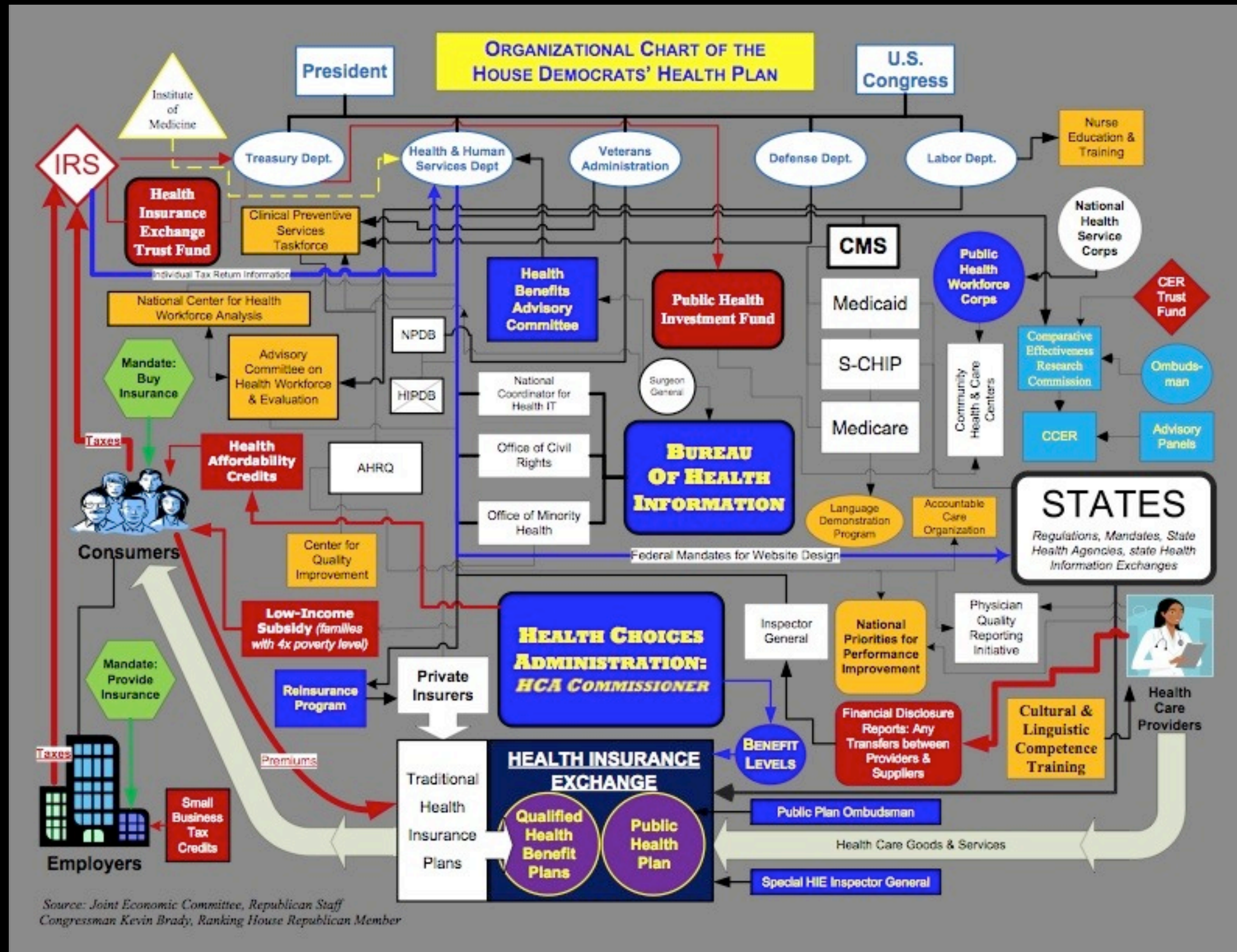
▶ Interactivity:

- Intelligent use of interaction to increase usefulness

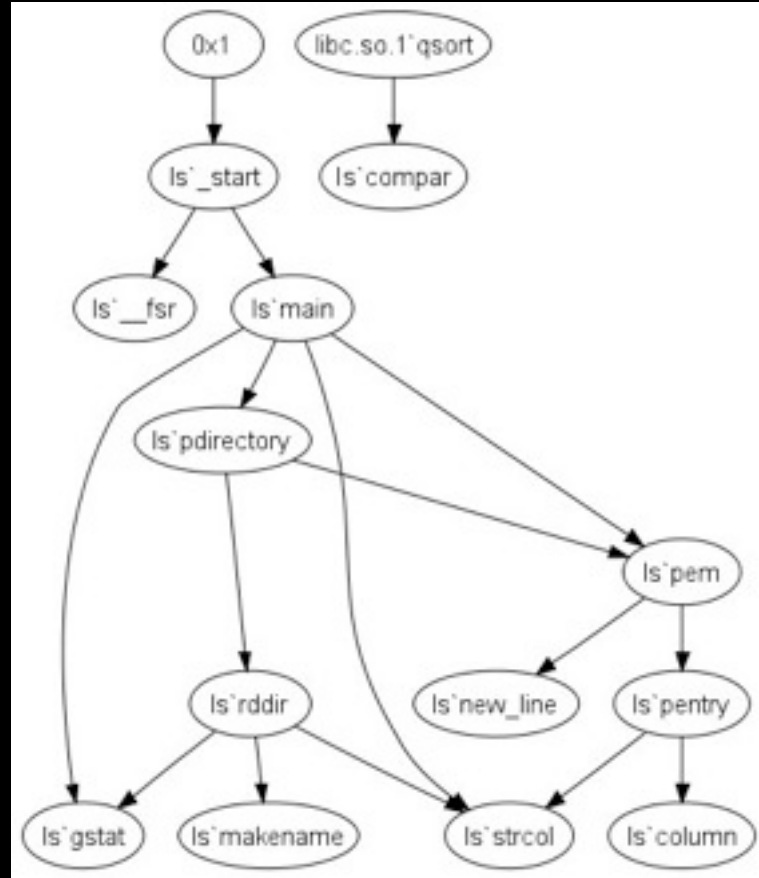
Poor visualization 1



Poor visualization 2



Poor visualization 3



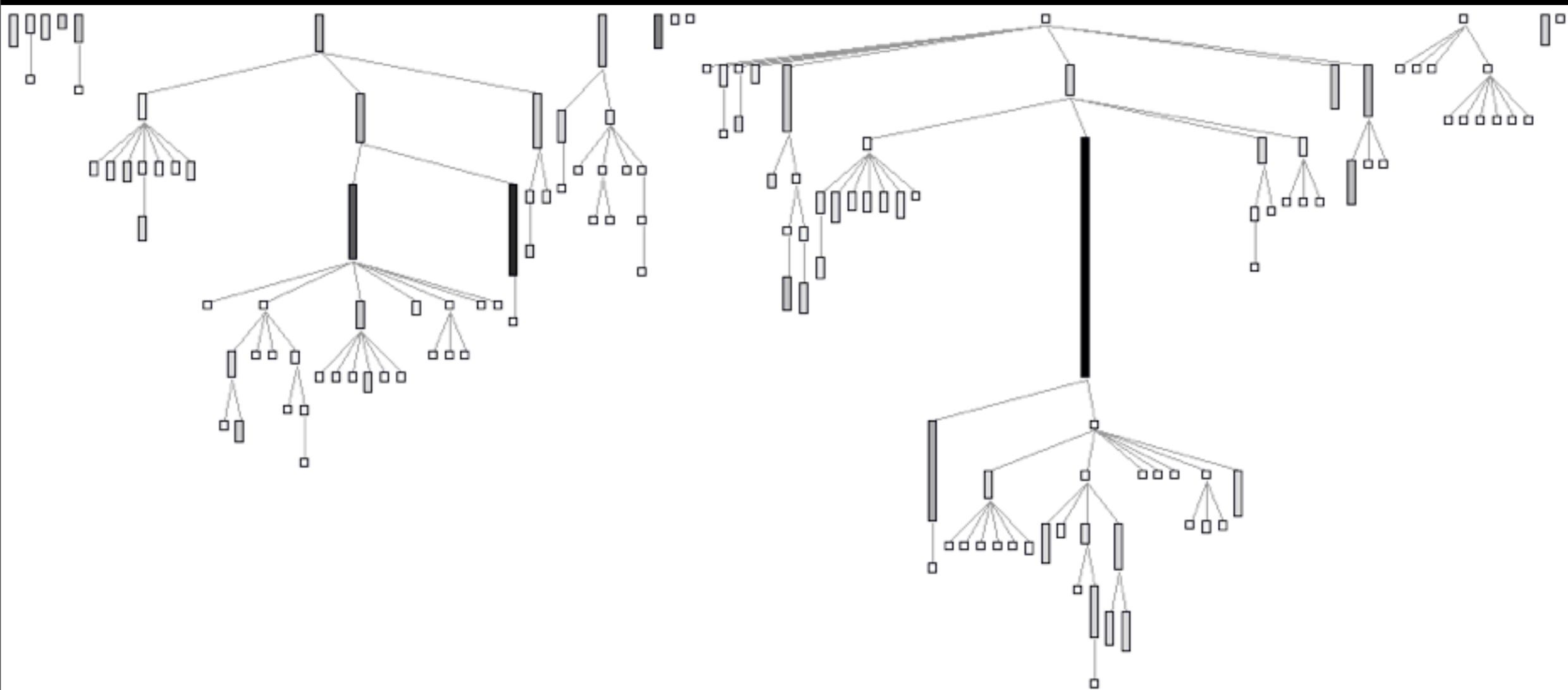
Call graph



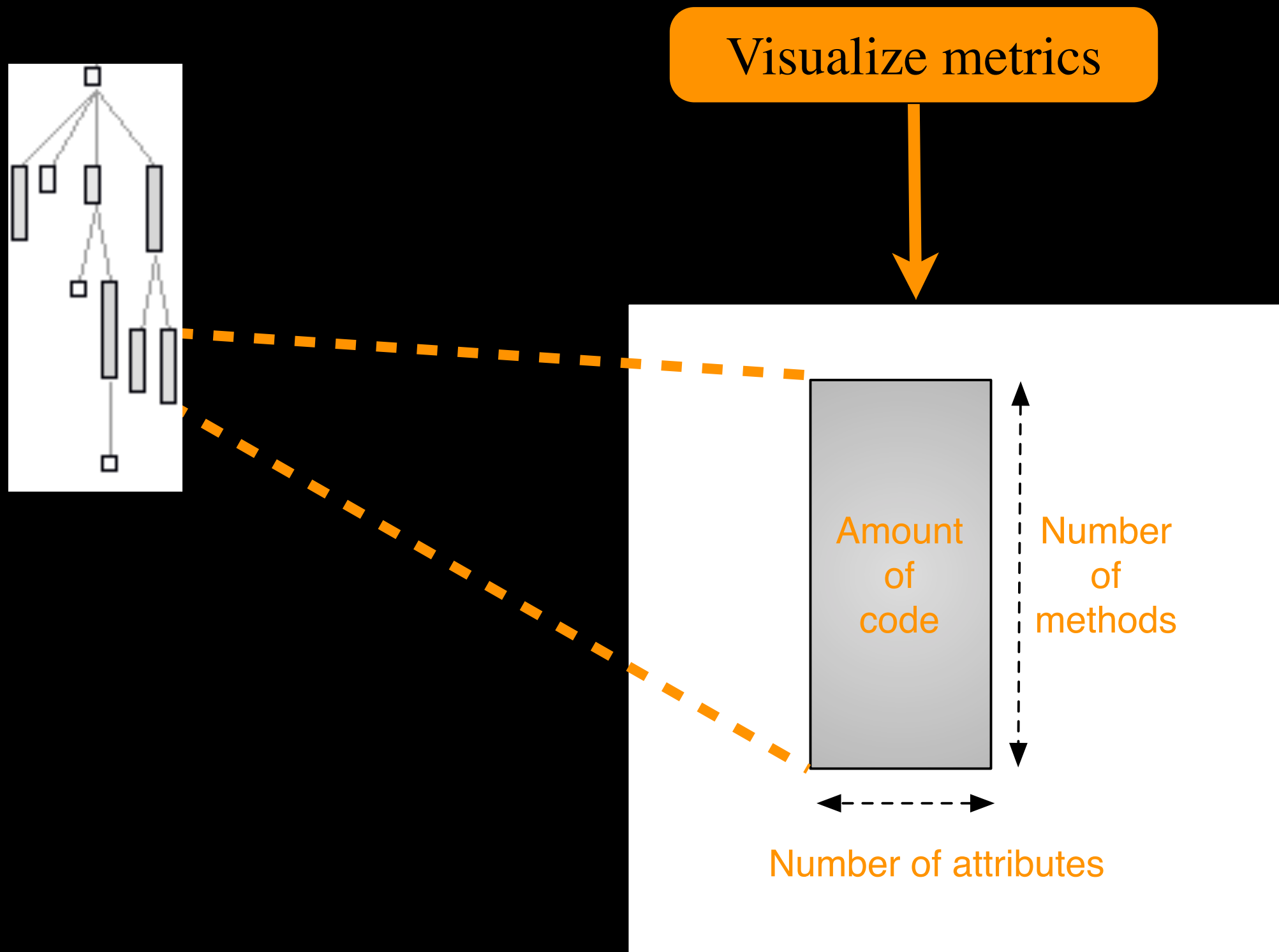
Software Visualizations

System complexity view [Lanza&Ducasse] (2003)

Visualize the system's structure



What does it mean?



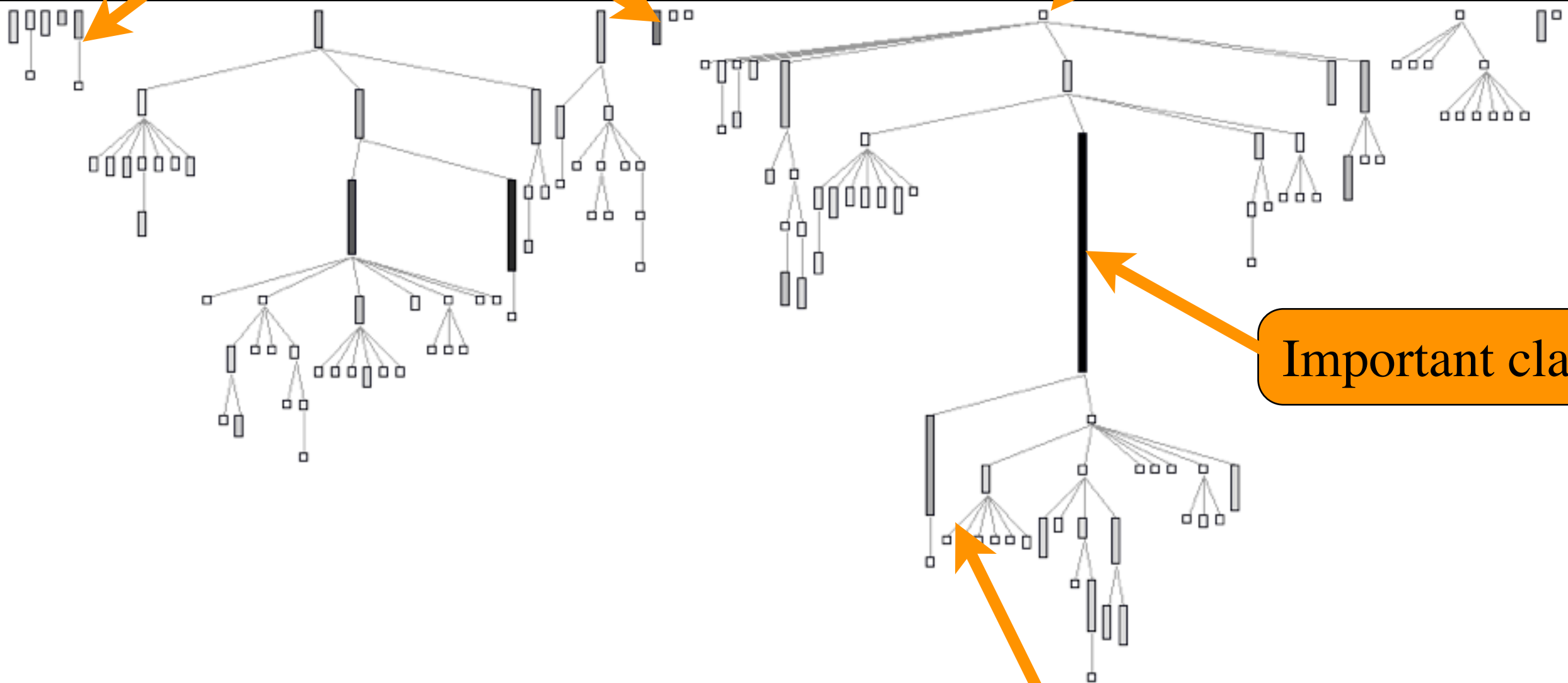
What do we see?

Shallow inheritance

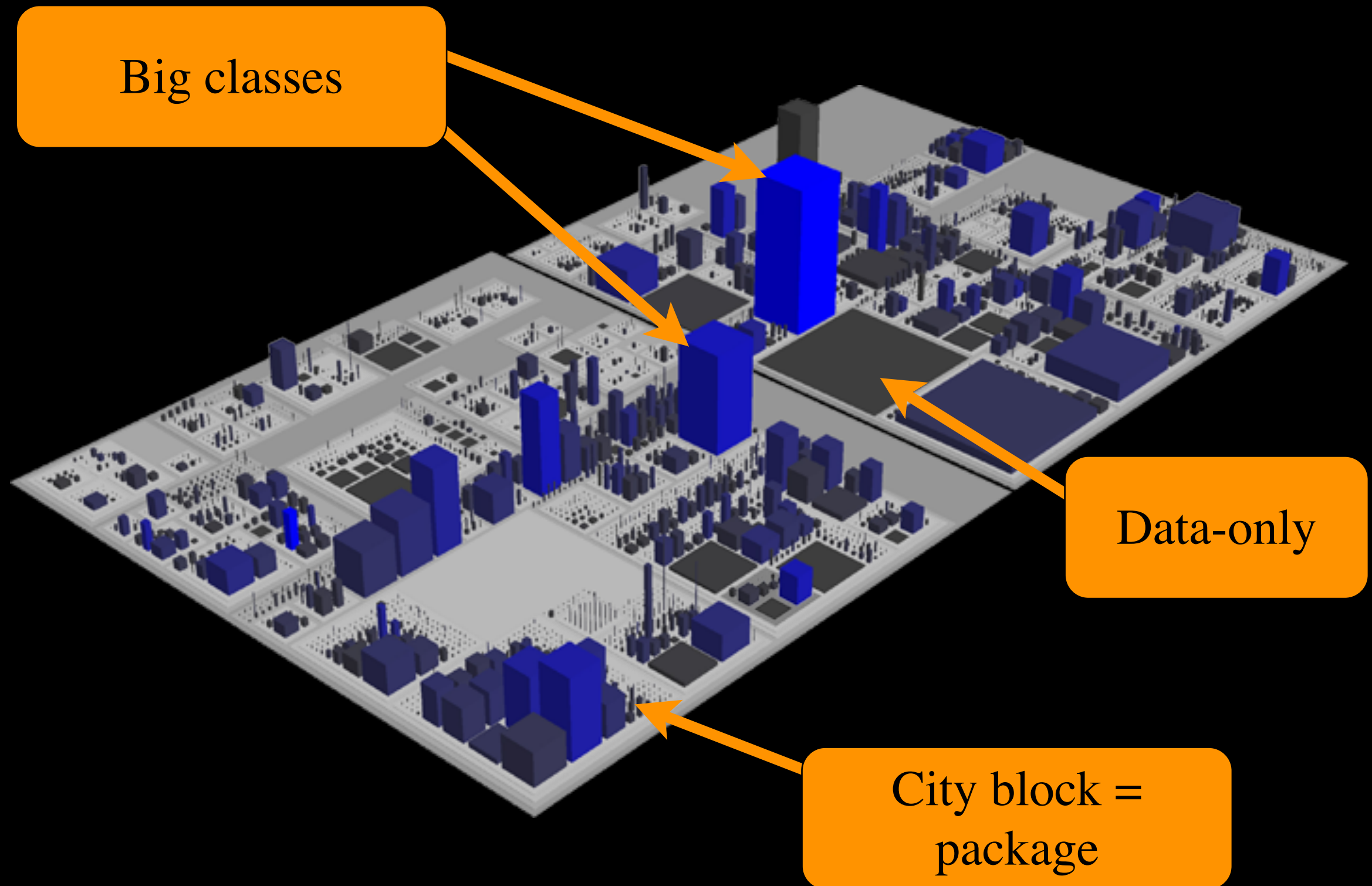
Large hierarchy

Important class

Classes without behavior

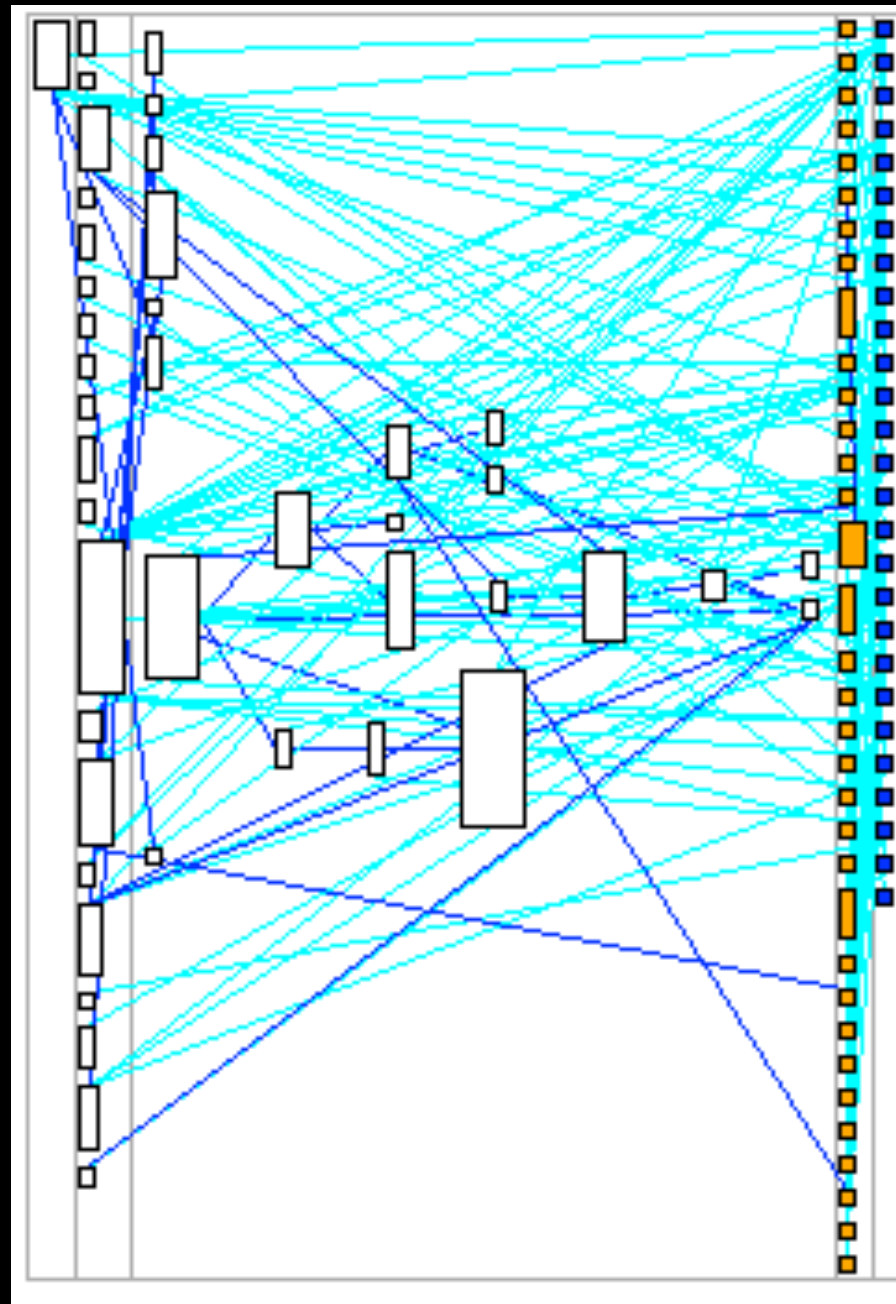


Structure of a system as a city

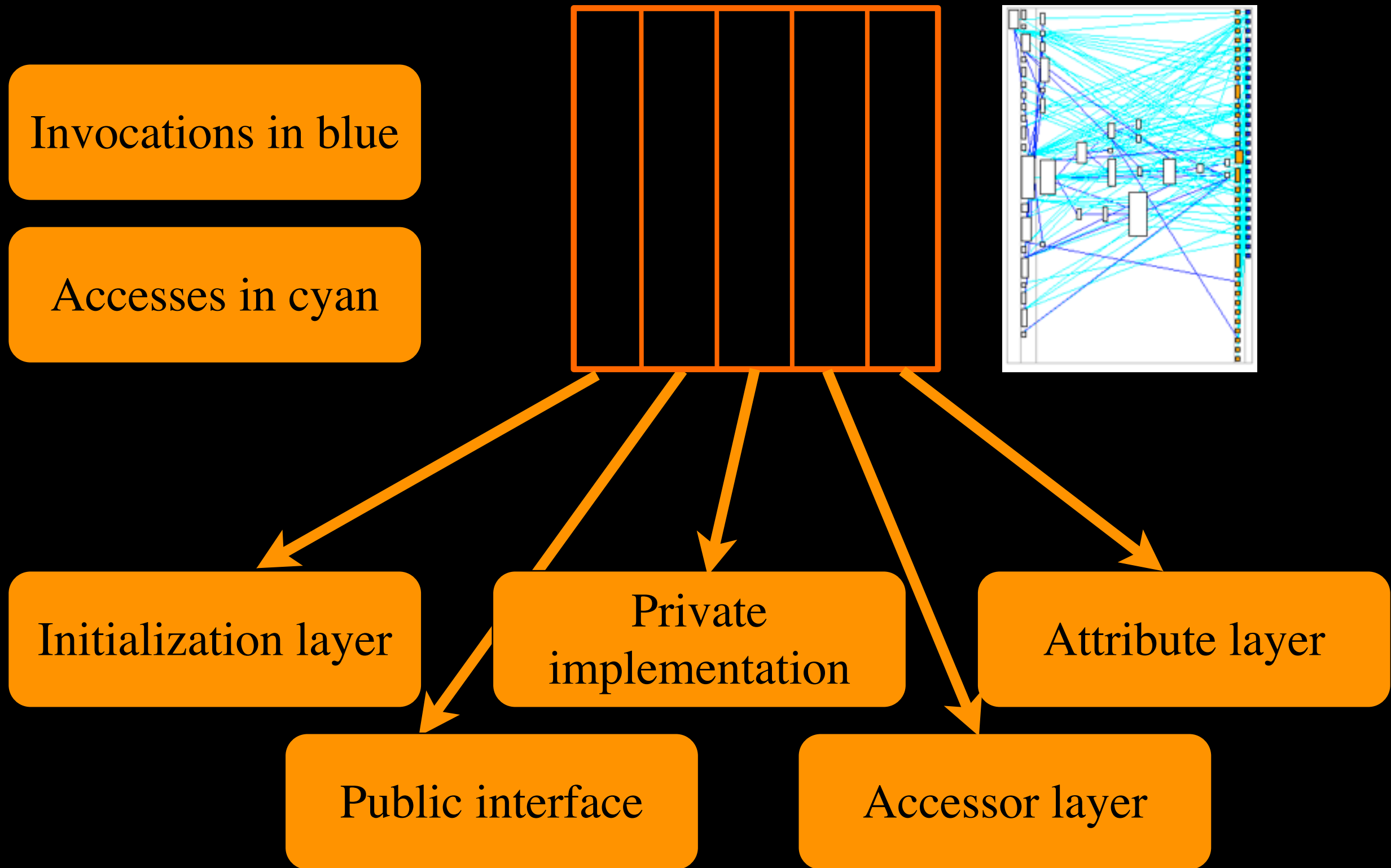


Class blueprint

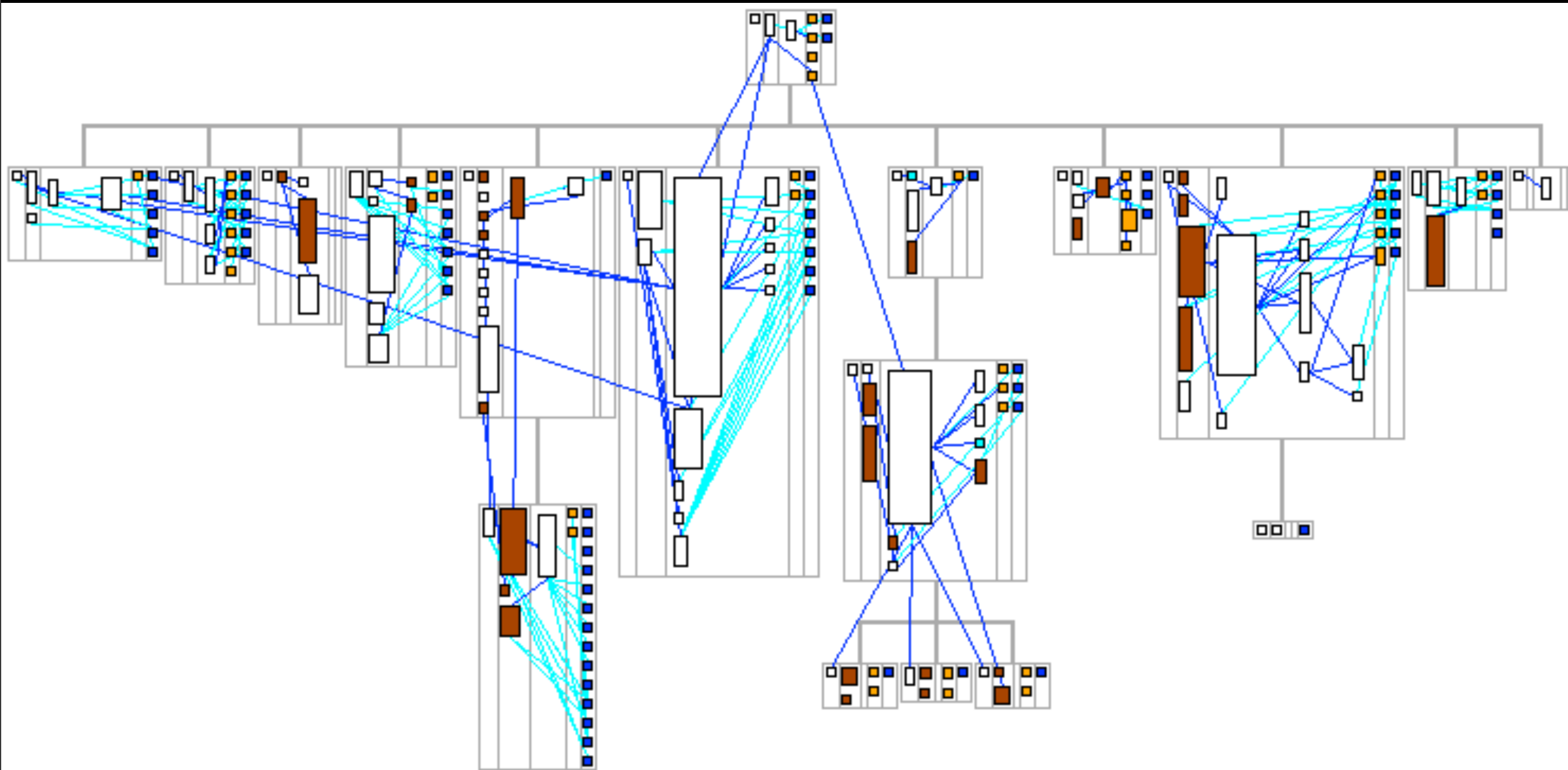
Visualize complexity and
cohesion of a class



What does it show?



What do we see?



Distribution map

How are different properties distributed over the system?

► Different possible properties:

- Metrics
- Concerns
- Defects/bugs
- Owners/authors
- ...

Example Distribution Map

Ownership within
JBoss

Each color is a
developer



Design structure matrices (DSMs)

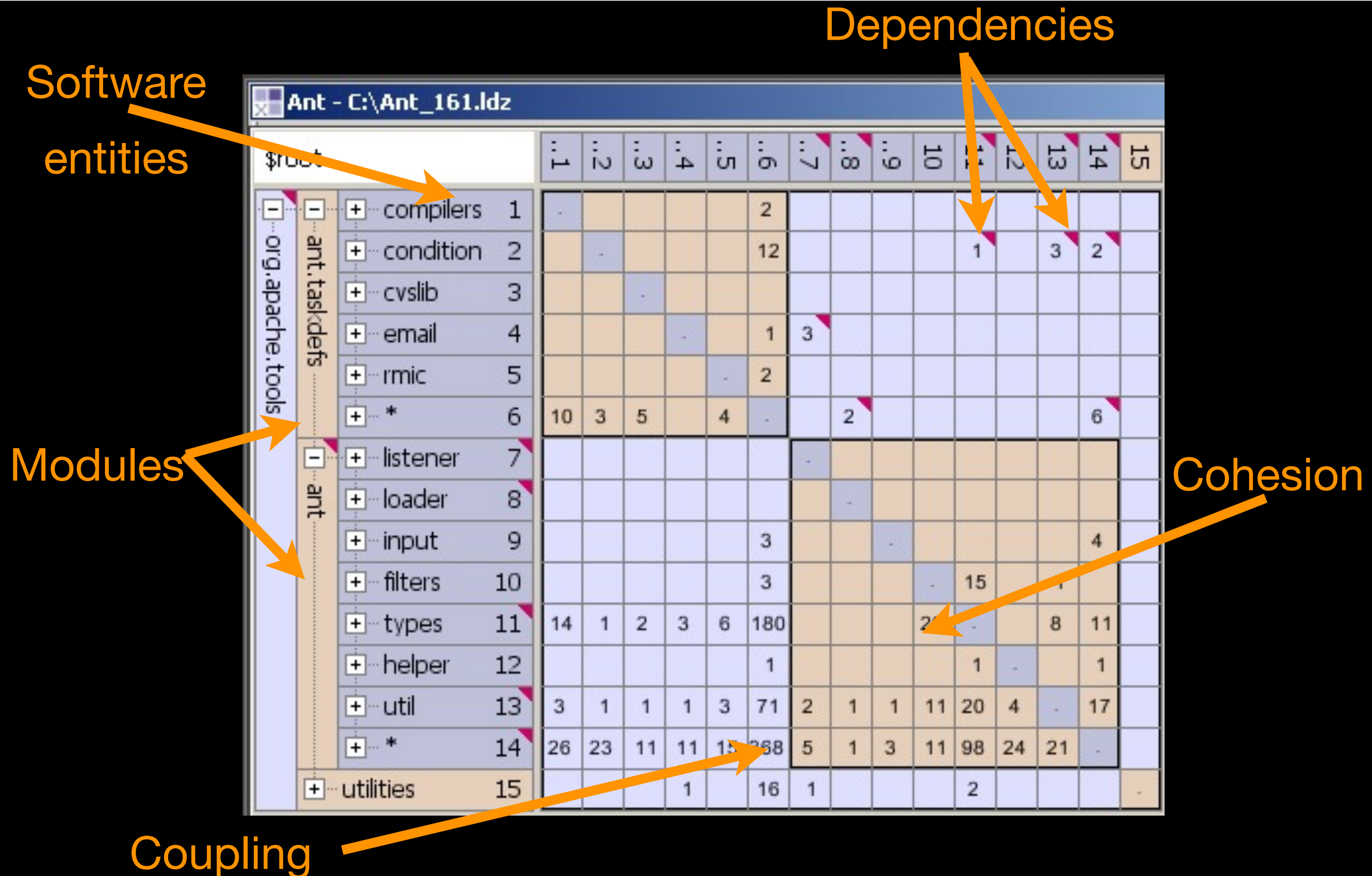
Entities

		1	2	3	4	5	6
Foundations	1	*					
Staircases	2	1	*		1		
Walls	3	1	2	*		2	
Floors	4	1		4	*		
Electricity	5					*	
Wiring	6			1		2	*

Dependency

Example: building a house

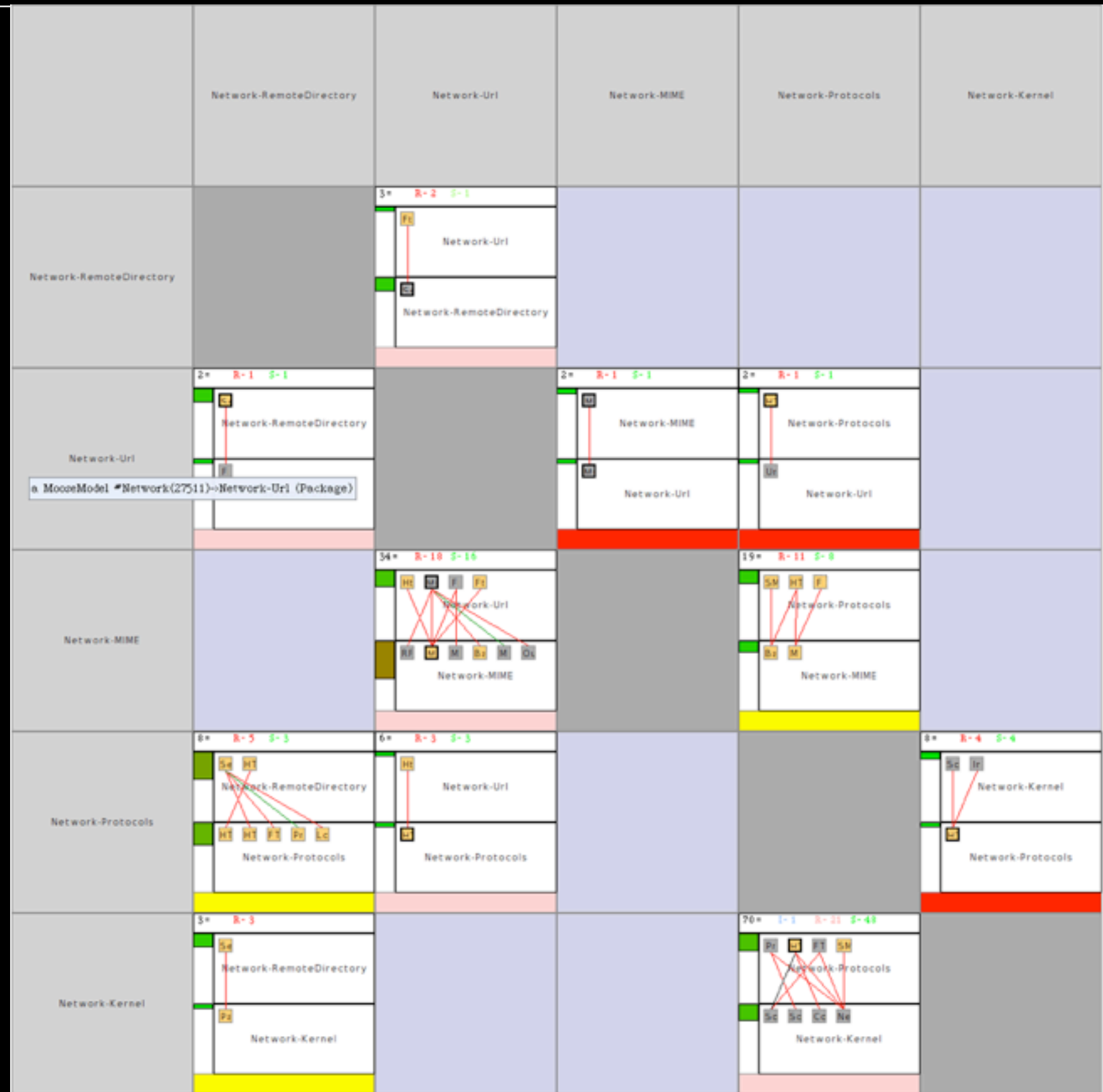
Assessing modularity with DSMs



Enriched DSMs

Kinds of dependencies

Identify cycles

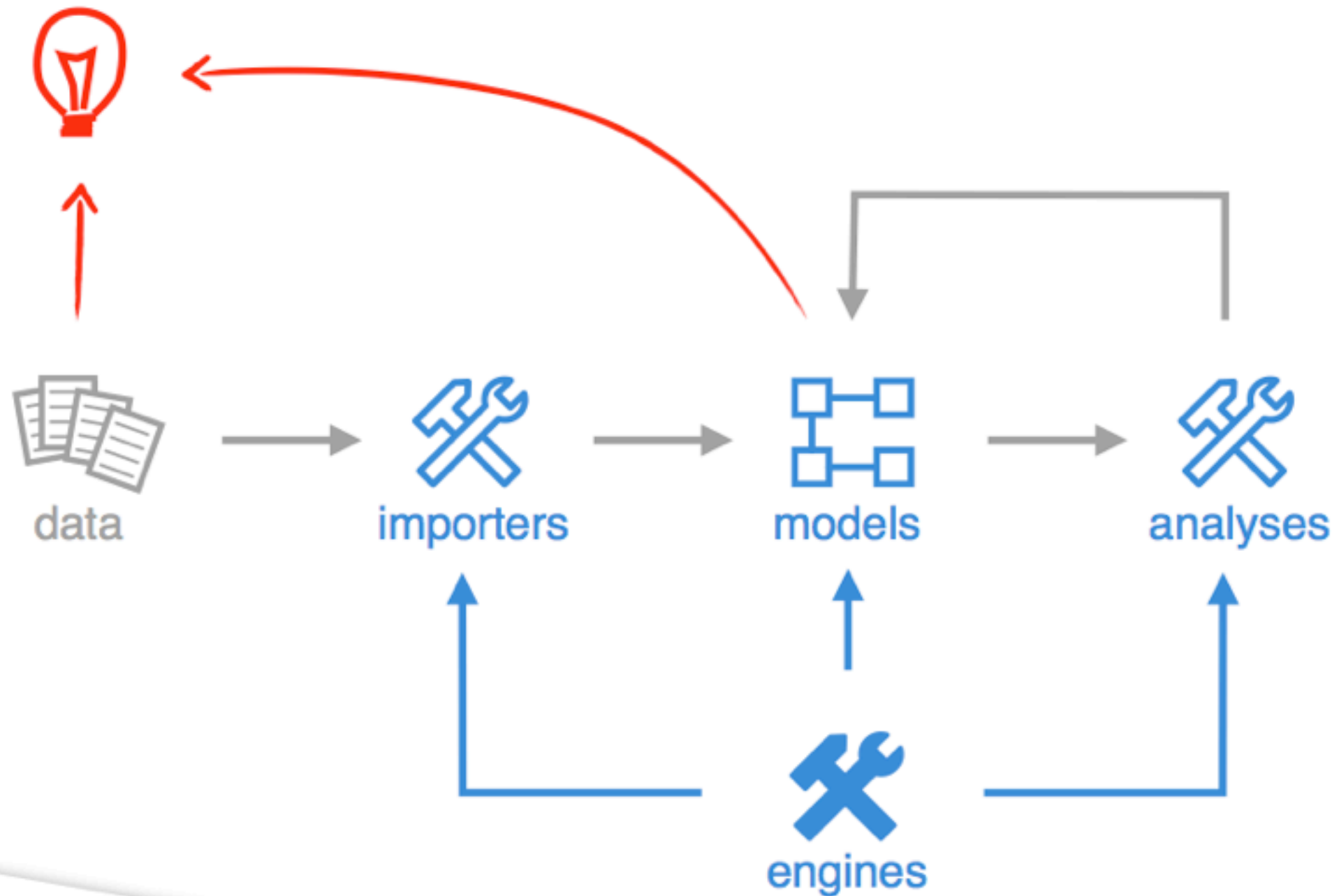


The Moose tool suite

Moose tool suite

- ▶ **Tool suite written in Pharo Smalltalk**
- ▶ **Lots of tools for analyzing software**
- ▶ **Provides:**
 - A meta-model to represent code
 - Metrics
 - Visualizations
- ▶ **Extensible/scriptable**
- ▶ **Lots of documentation:**
 - <http://www.moosetechnology.org>
 - <http://www.themoosebook.org/book>

Assessing software



moosetechnology.org

(slide from Tudor Girba)

Demo

The screenshot shows the 'Moose Panel' application window. On the left, a 'Models' sidebar lists 'web_portal'. The main panel, titled 'web_portal x', shows 'web_portal (MooseModel)' and a list of 25 model classes with their respective counts:

- All famixaccesses (654)
- All famixattributes (66)
- All famixcaughtexceptions (77)
- All famixclasses (100)
- All famixcomments (245)
- All famixdeclaredexceptions (105)
- All famiximplicitvariables (31)
- All famixinheritances (131)
- All famixinocations (1275)
- All famixlocalvariables (209)
- All famixmethods (299)
- All famixnamespaces (22)
- All famixparameterizableclasses (13)
- All famixparameterizedtypes (20)
- All famixparameters (261)
- All famixparametertypes (3)
- All famixprimitivetypes (5)
- All famixreferences (166)
- All famixthrownexceptions (69)
- All model classes (48)
- All model namespaces (7)
- All model types (48)
- SourceLanguage: Java (1)

An orange rounded rectangle with the text 'Brief demo' is overlaid on the right side of the interface.

Build your own visualization with Mondrian and Moose

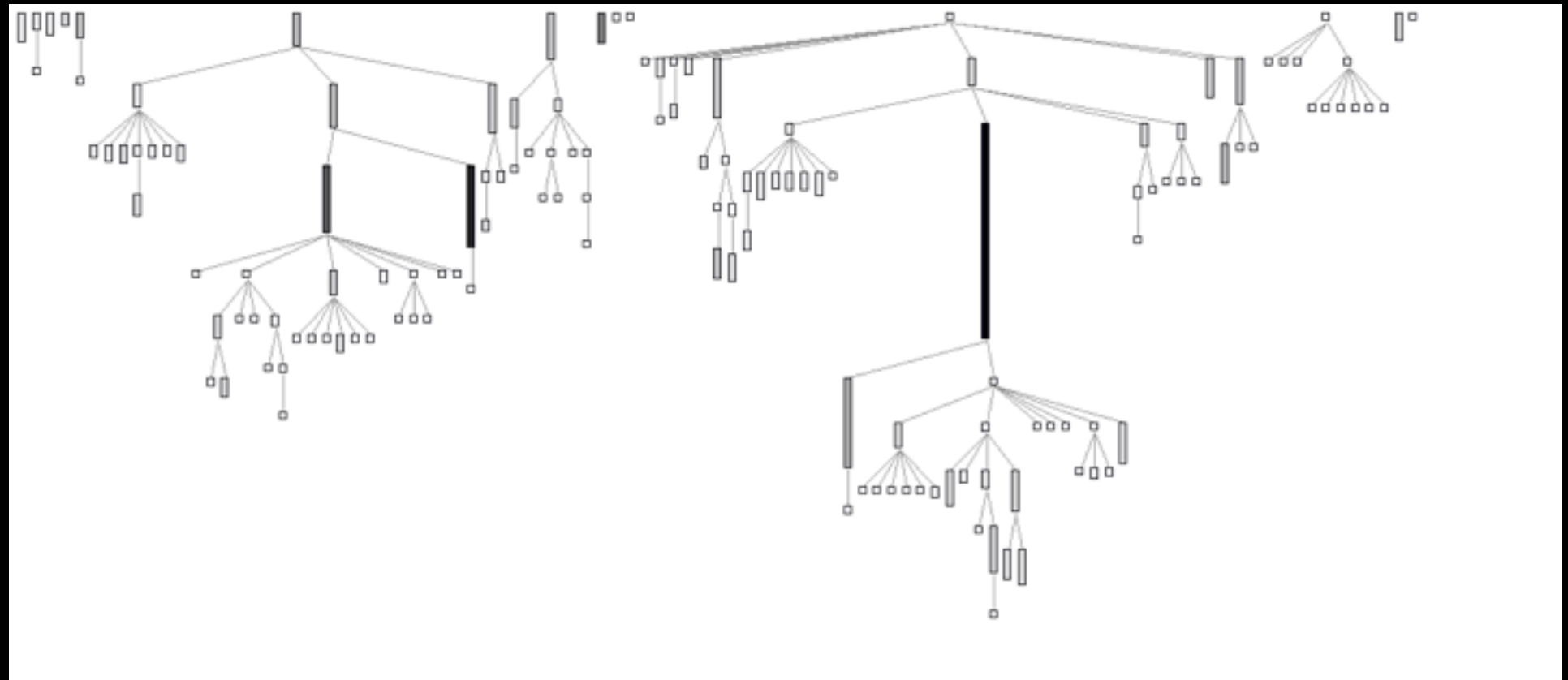
What is Mondrian?

- ▶ **Scripting language**
- ▶ **On top of Smalltalk**
- ▶ **Integrated with Moose**
- ▶ **Allow to easily prototype interactive visualizations**
- ▶ **Most visualizations shown before are implemented in Mondrian**
- ▶ **For more information:**
 - Chapter 15 of the Moose book
(<http://www.themoosebook.org/book/internals/mondrian>)

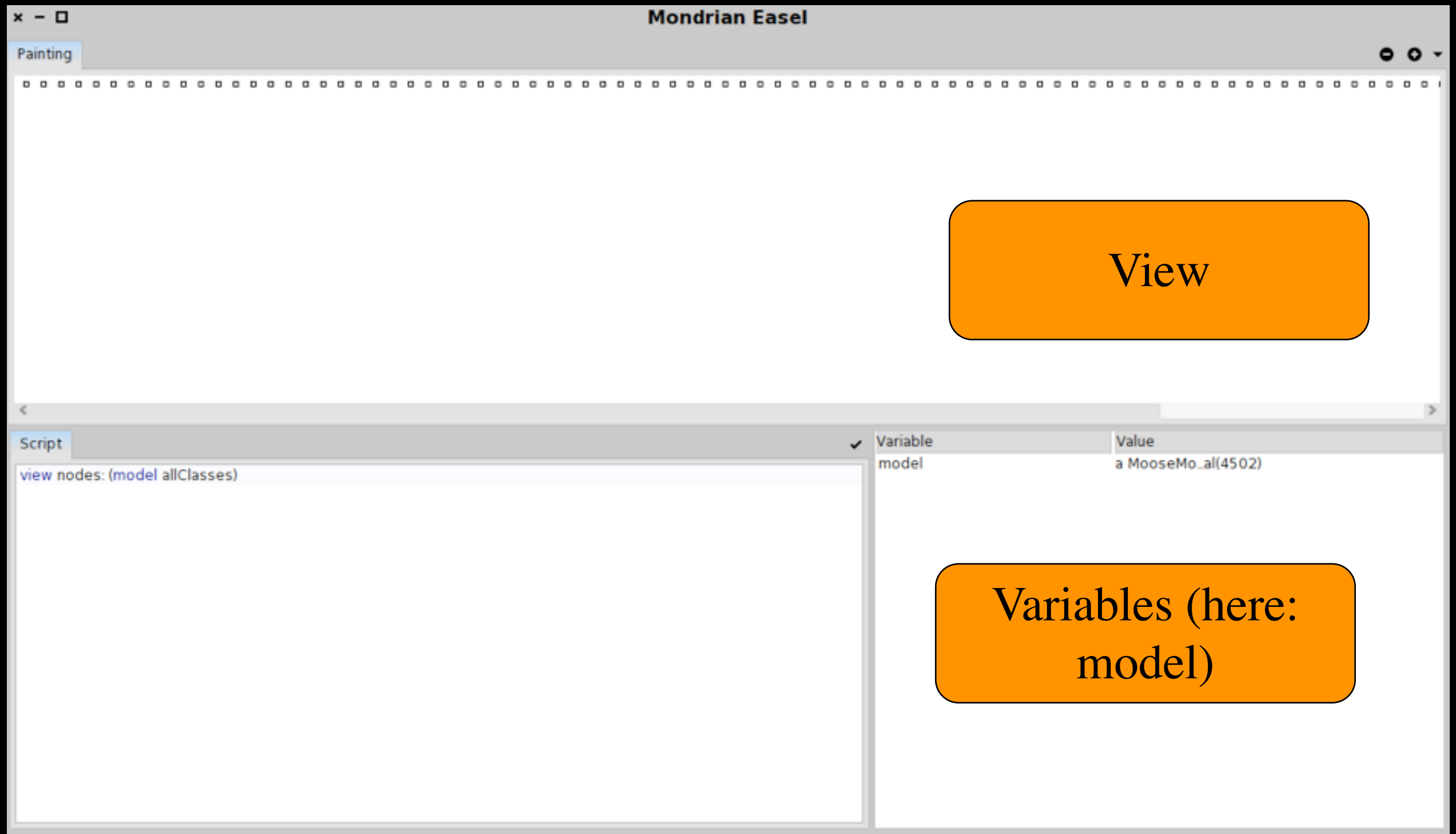
Example

► Constructing the System Complexity View step-by-step

- Visualizing nodes
- Visualizing edges
- Picking a shape
- Simple interaction



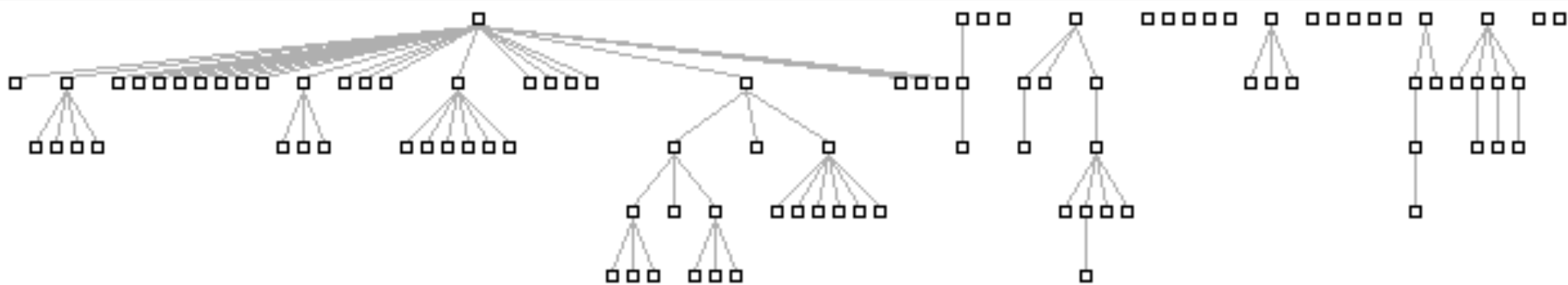
Visualizing nodes



Visualizing edges

Script

```
view nodes: (model allClasses).  
view edges: (model allClasses) from: [:class | class superclass] to: #yourself.  
view treeLayout
```



Picking a shape

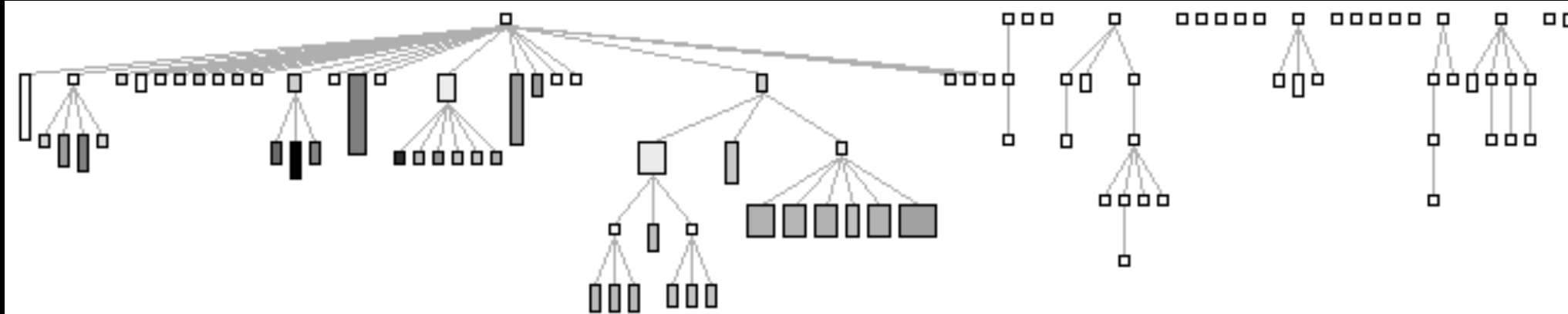
Script

```
view shape rectangle
width: [:class | class numberOfAttributes * 2];
height: [:class | class numberOfMethods * 2];
linearFillColor: [:class | class numberOfLinesOfCode] within: (model allClasses).

view nodes: (model allClasses).

view edges: (model allClasses) from: [:class | class superclass] to: #yourself.
view treeLayout
```

Set shape before
drawing



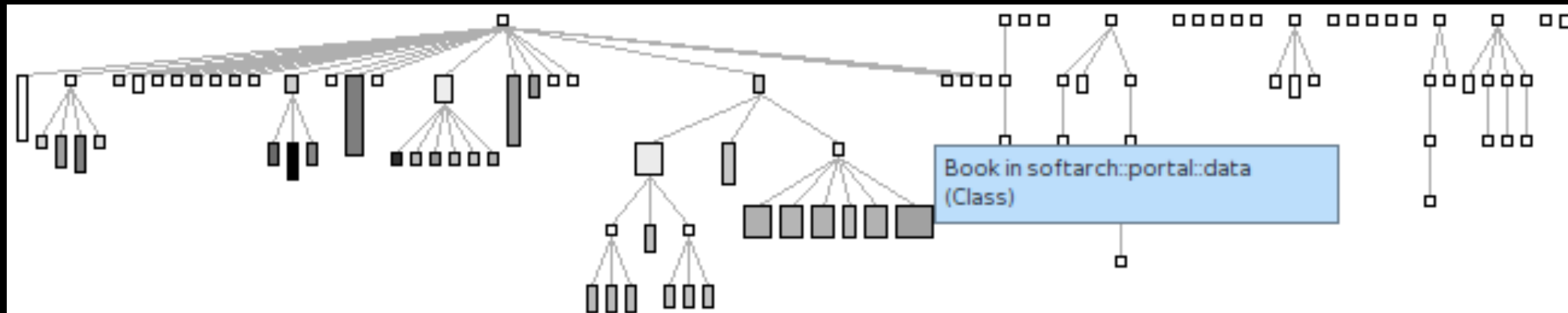
Adding interactions

Script

```
view shape rectangle
  width: [:class | class numberOfAttributes * 2];
  height: [:class | class numberOfMethods * 2];
  linearFillColor: [:class | class numberOfLinesOfCode] within: (model allClasses).

view nodes: (model allClasses).
view interaction popupText[:class | class name].

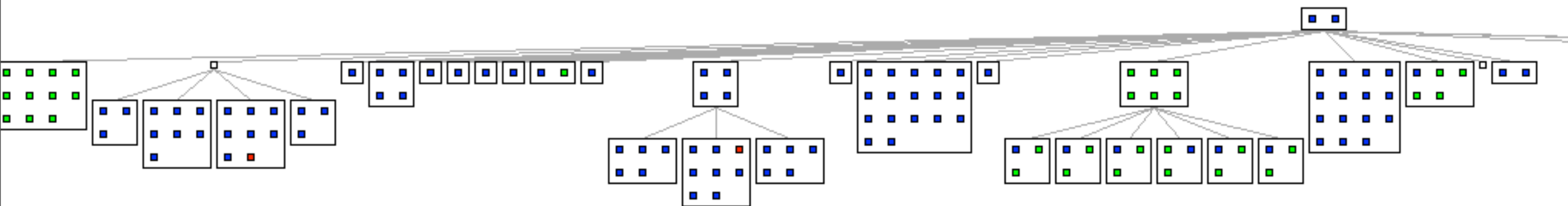
view edges: (model allClasses) from: [:class | class superclass] to: #yourself.
view treeLayout
```



A more complex example

Script

```
view shape rectangle
  width[:cl | cl attributes size * 2];
  height[:cl | cl methods size * 2].
view nodes: (model allClasses) forEach[:class |
  view shape rectangle
    fillColor[:method |
      method isPublic
        ifTrue:[Color blue]
        ifFalse:[method isPrivate ifTrue:[Color red] ifFalse:[Color green]]].
    view nodes: (class methods).
    view GridLayout].
view edges: (model allClasses) from[:cl | cl superclass ] to: [:cl | cl].
view treeLayout
```



Conclusion

▶ **Software visualization:**

- Represent structural data
- Good/bad visualizations
- A way to interpret metrics

▶ **Mondrian:**

- Build your own custom visualizations