

A case of Visitor versus Interpreter Pattern

Paul Klint, Mark Hills, Tijs van der Storm,
Jurgen Vinju

Zürich, June 30th 2011

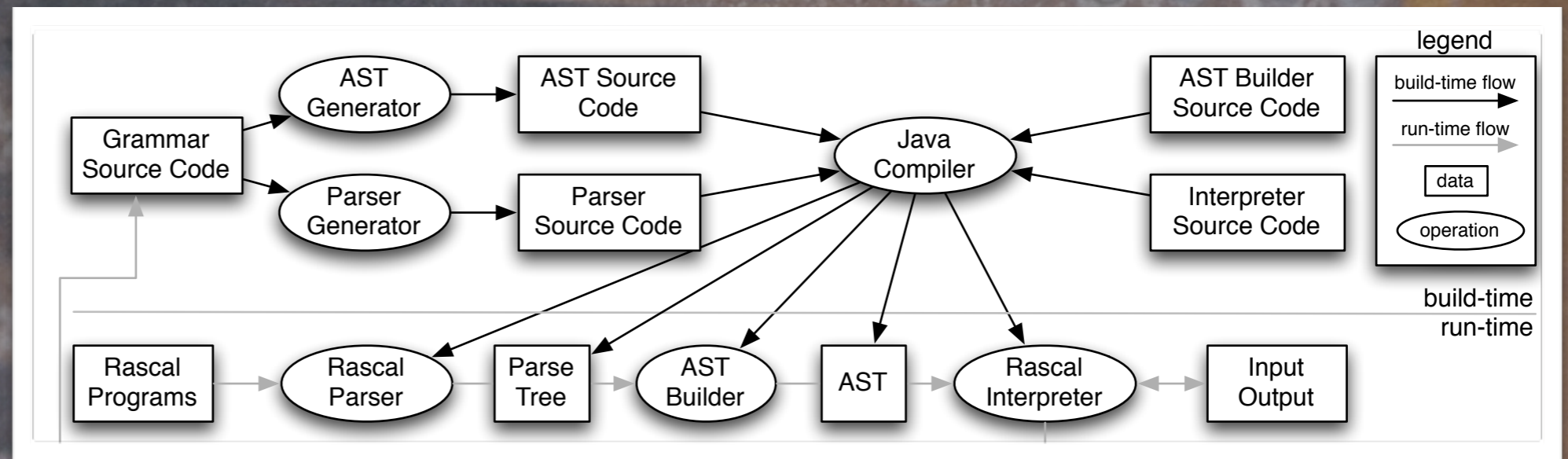
Why?

- Why this experiment?
- Why this “laboratory” setup?
- Why trust the conclusions?

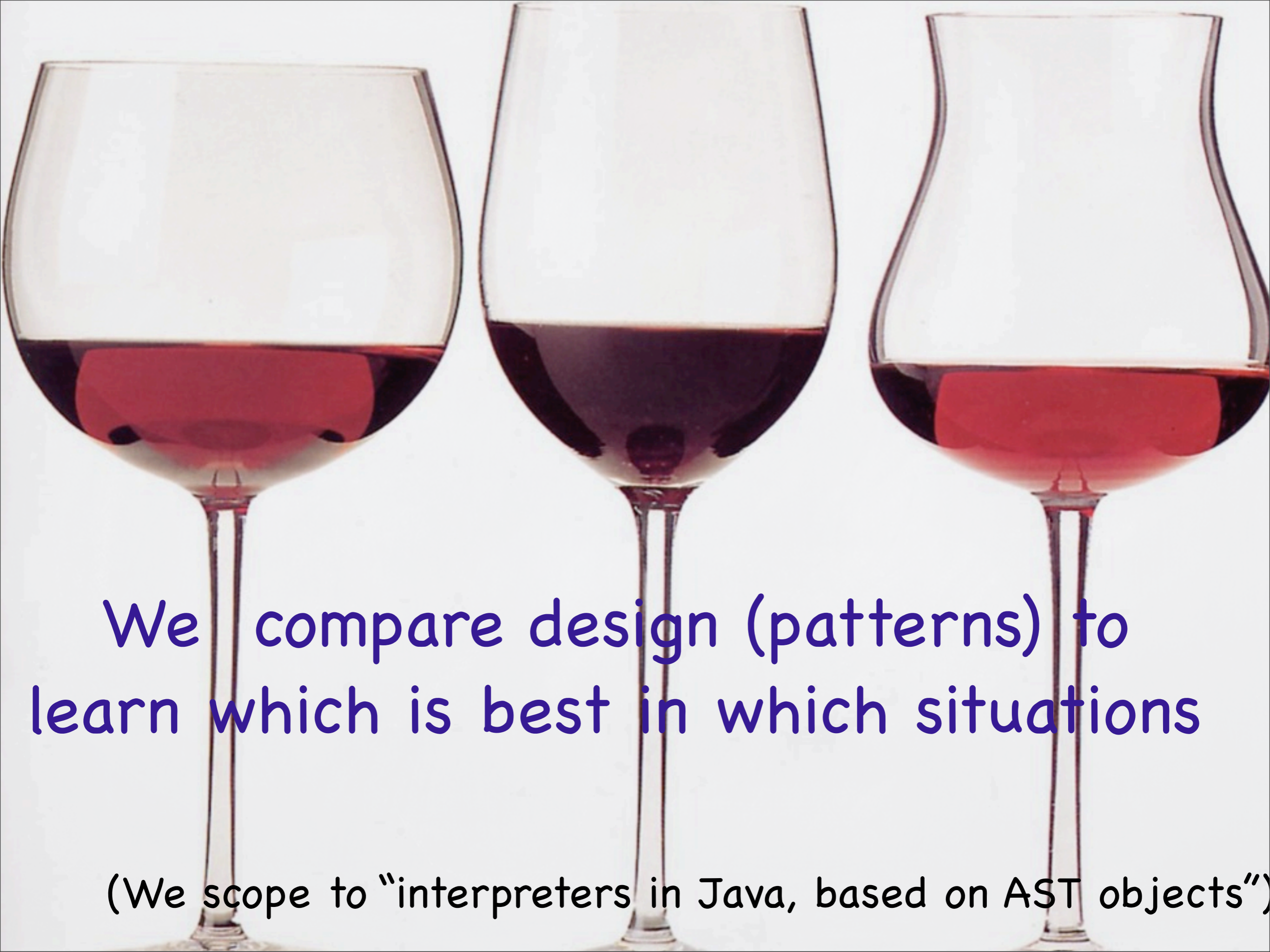


“Long Live Incremental Research!”

Case:



- Abstract syntax trees (ASTs)
- Operations on ASTs
- 400 concrete classes, 140 abstract classes
- AST classes are generated from a grammar
- Dispatch, dispatch, dispatch
- Evolution of the \pm 100 kLOC java code



We compare design (patterns) to learn which is best in which situations

(We scope to "interpreters in Java, based on AST objects")

Composite Pattern

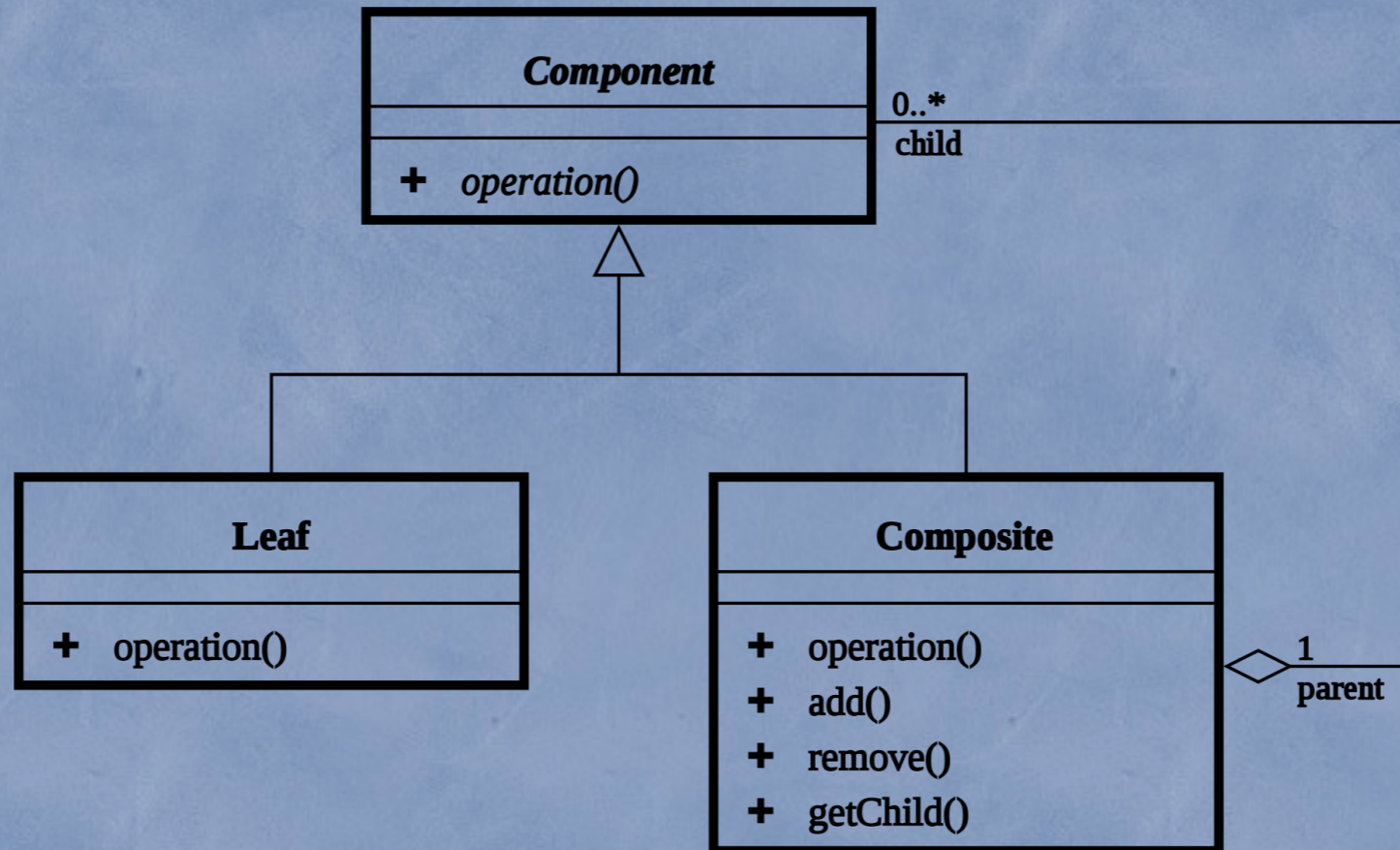


Fig. 2. The Composite Pattern³

Composite Pattern

Statement

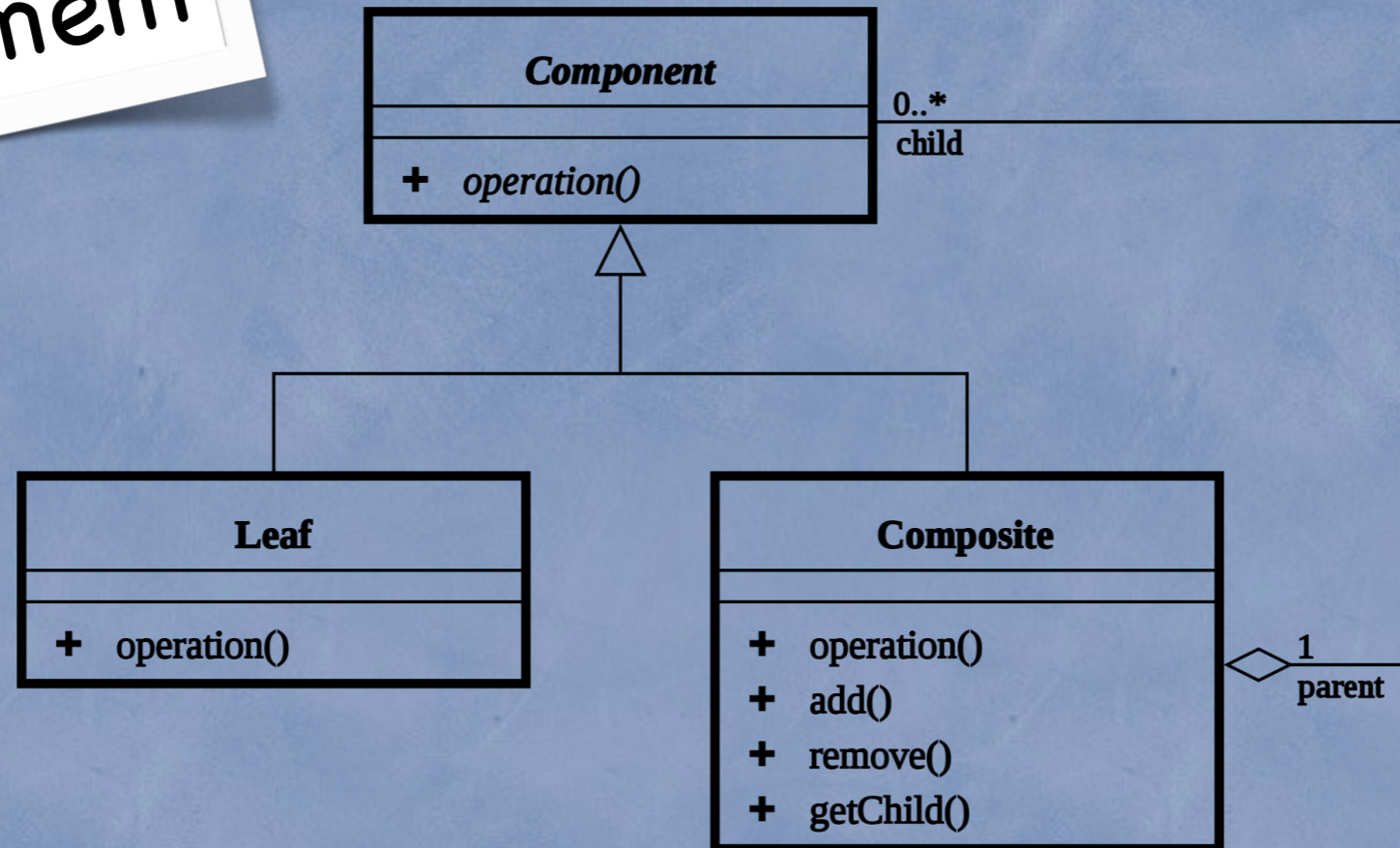


Fig. 2. The Composite Pattern³

Composite Pattern

Statement

NoOp

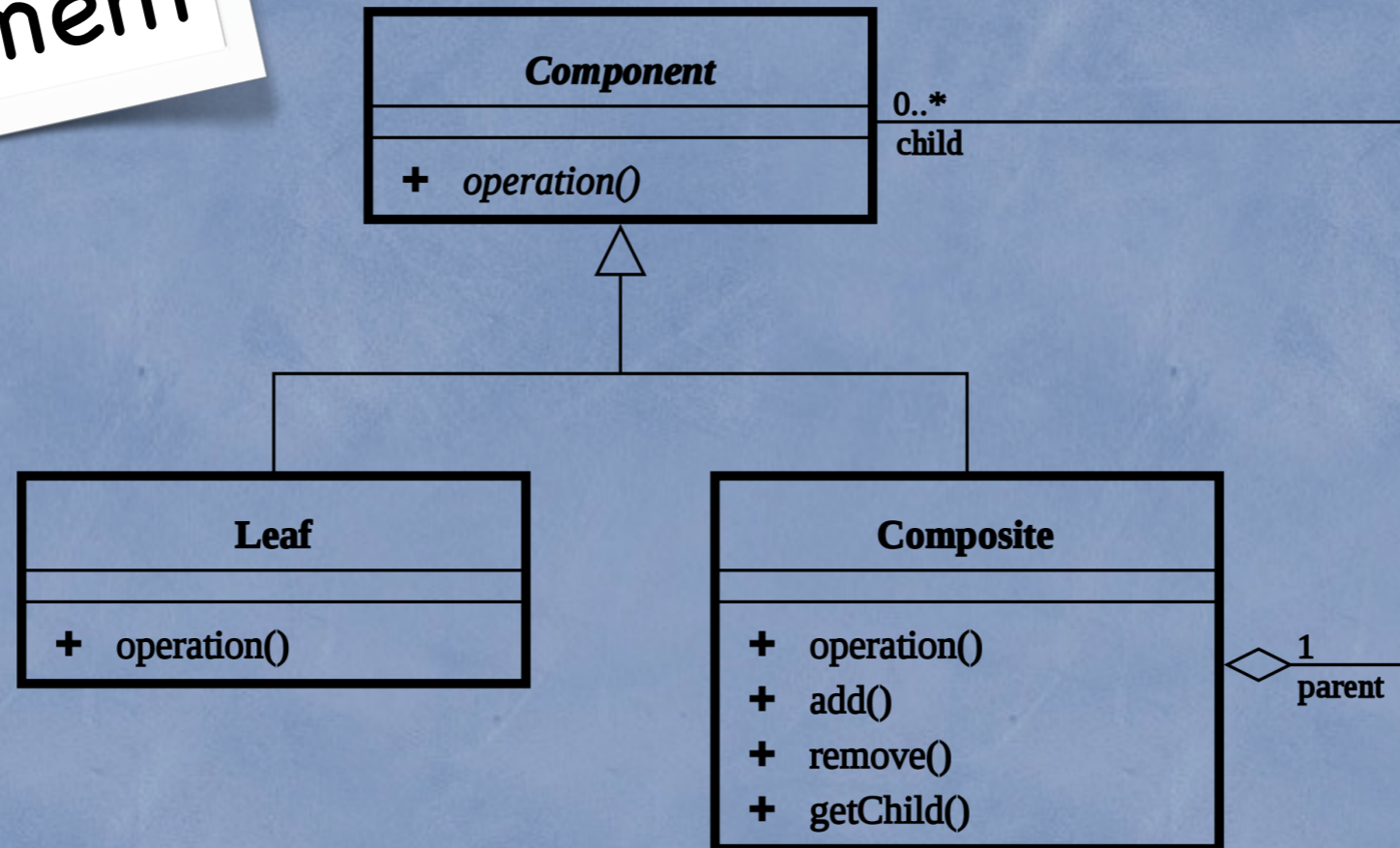


Fig. 2. The Composite Pattern³

Composite Pattern

Statement

NoOp

IfThenElse

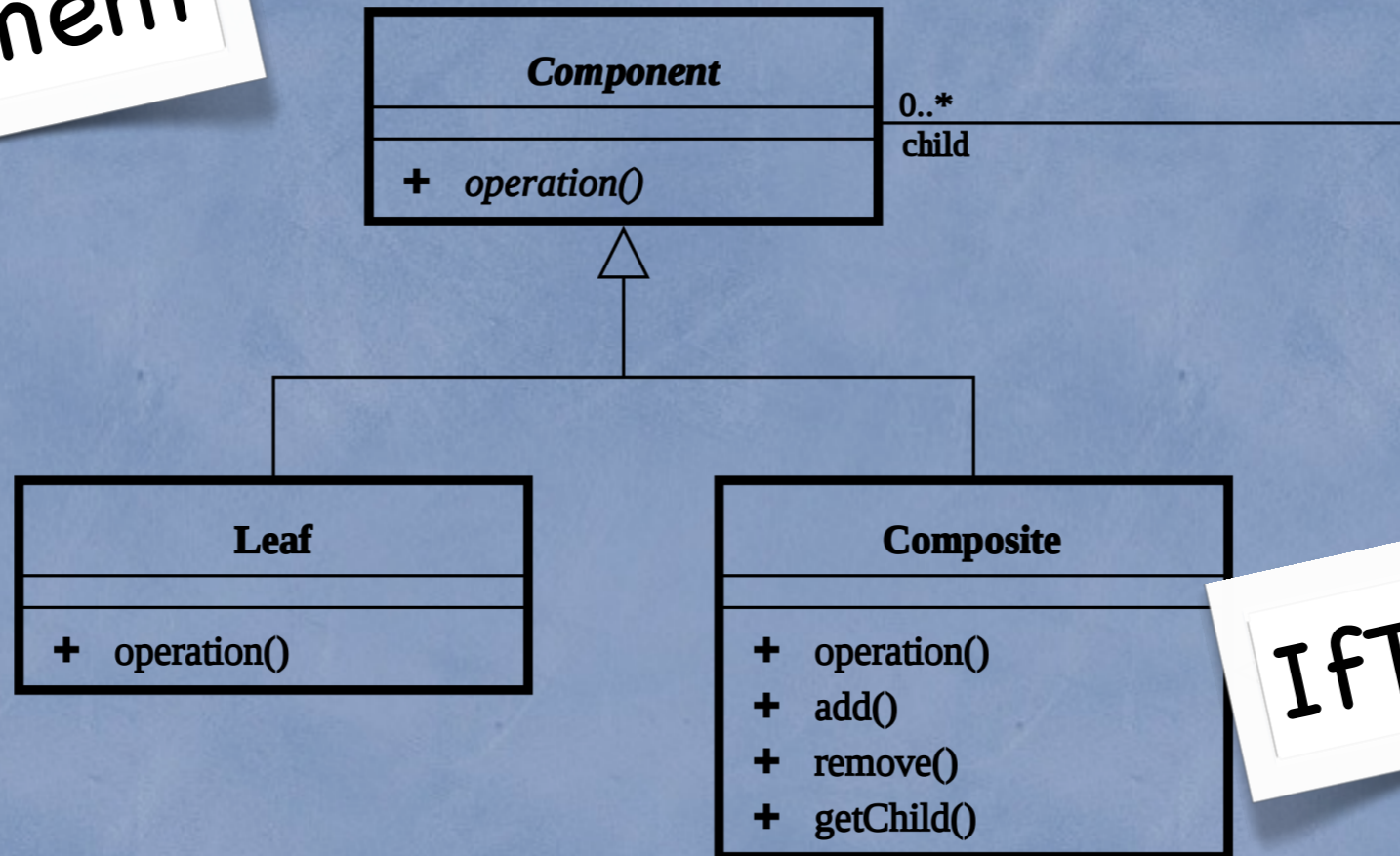


Fig. 2. The Composite Pattern³

Composite Pattern

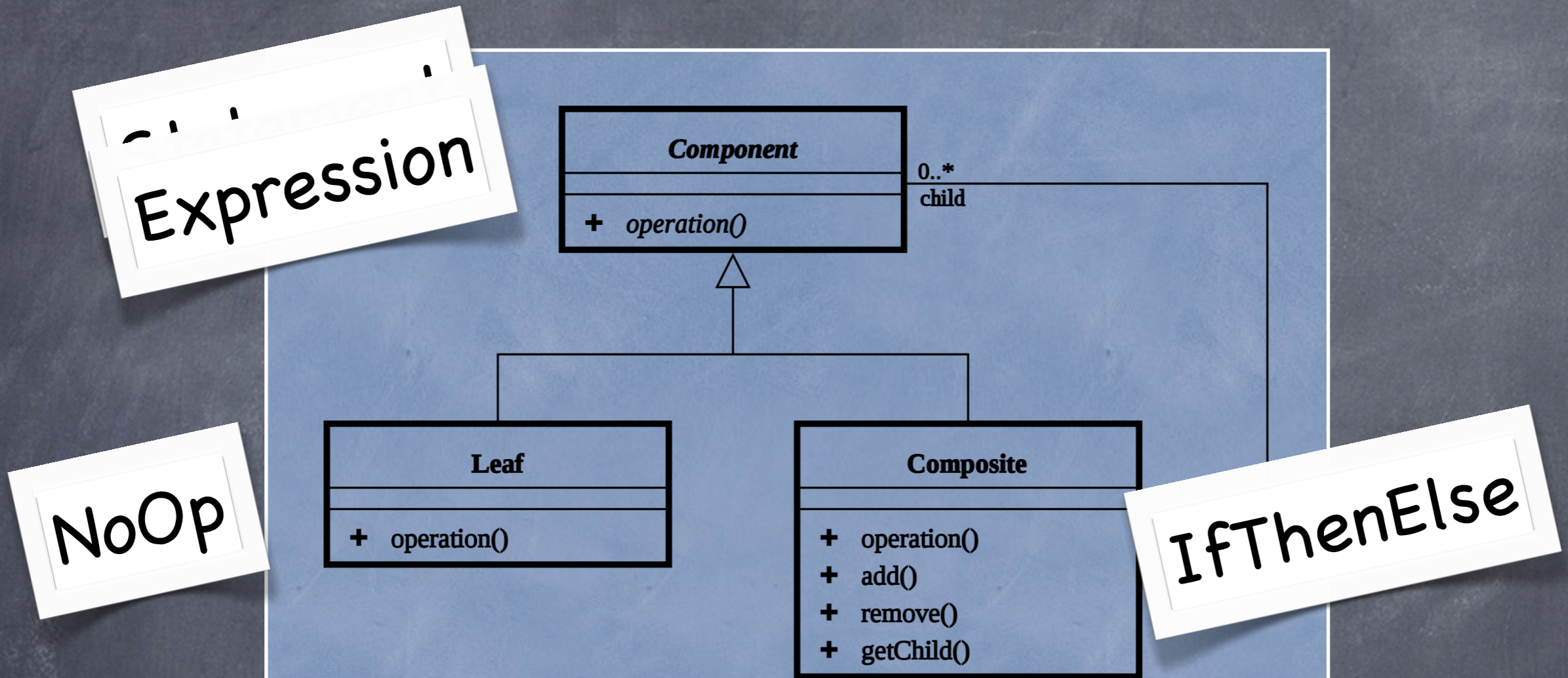


Fig. 2. The Composite Pattern³

Composite Pattern

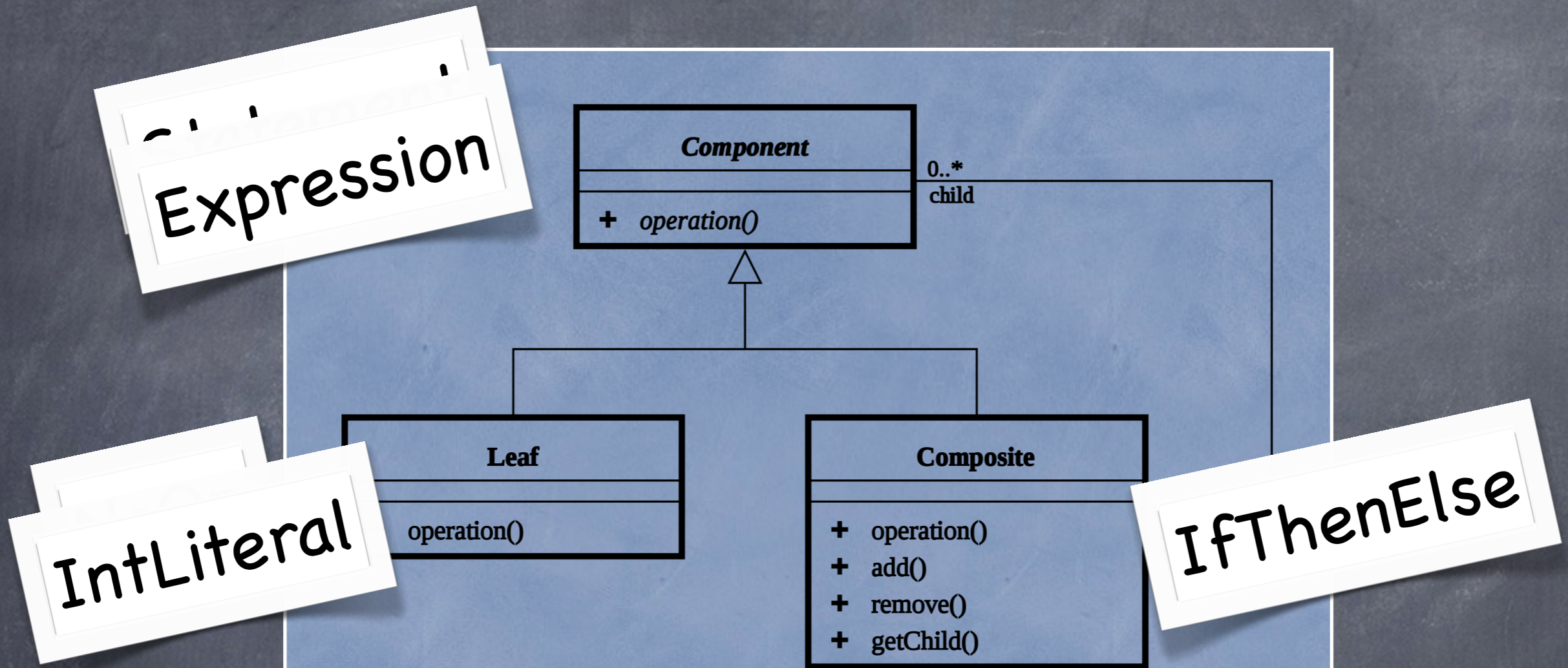


Fig. 2. The Composite Pattern³

Composite Pattern

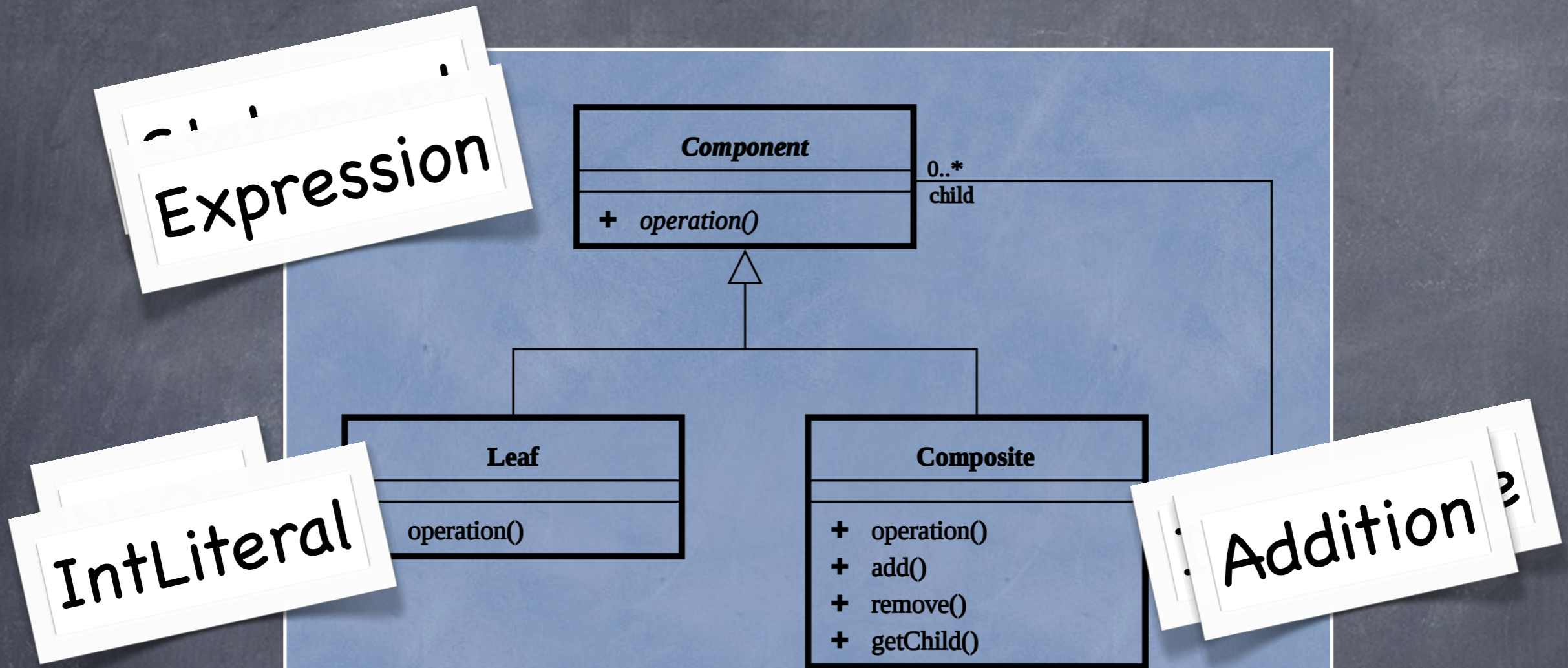


Fig. 2. The Composite Pattern³

AST instance

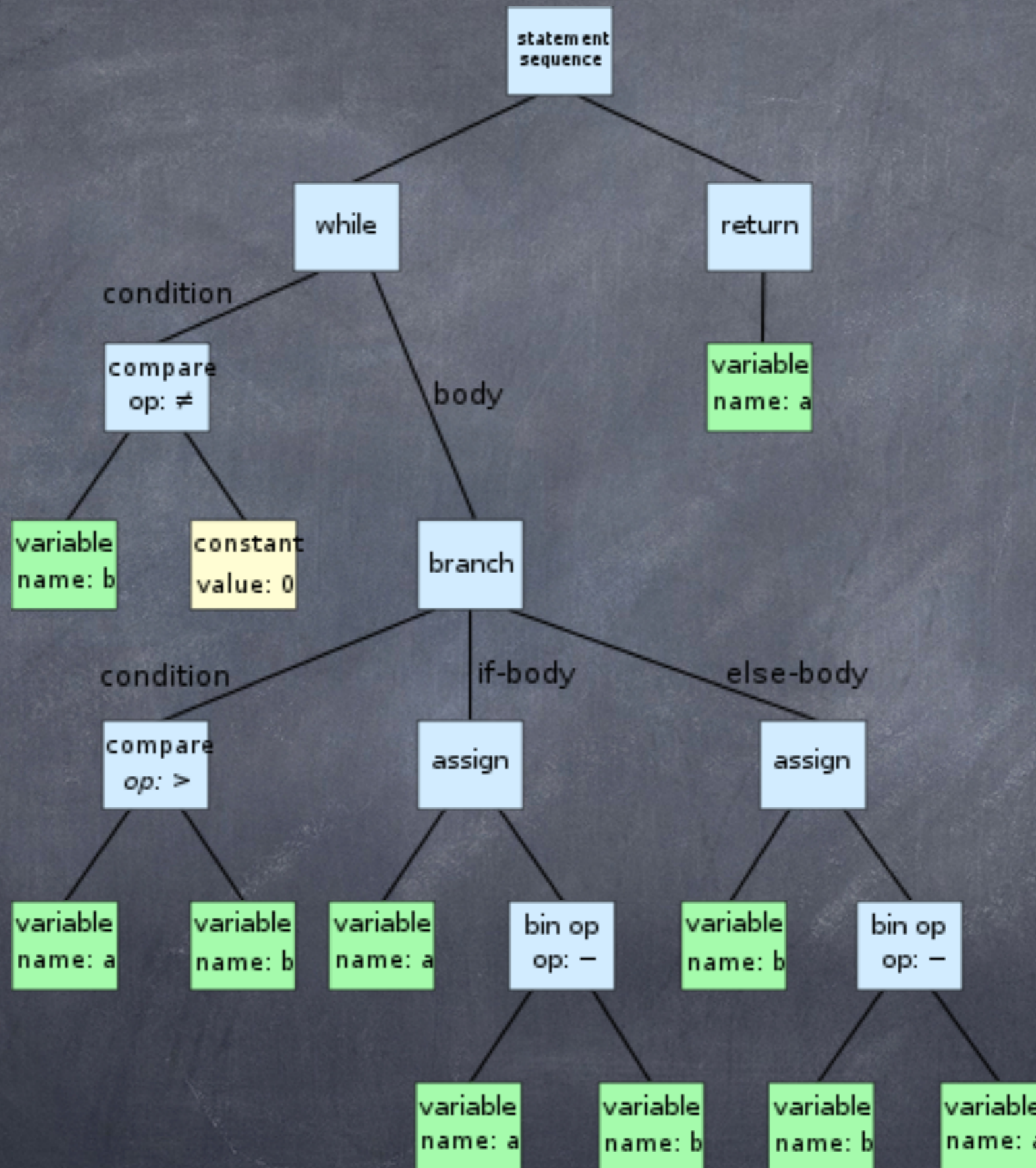


image from wikipedia.org

Interpreter Pattern

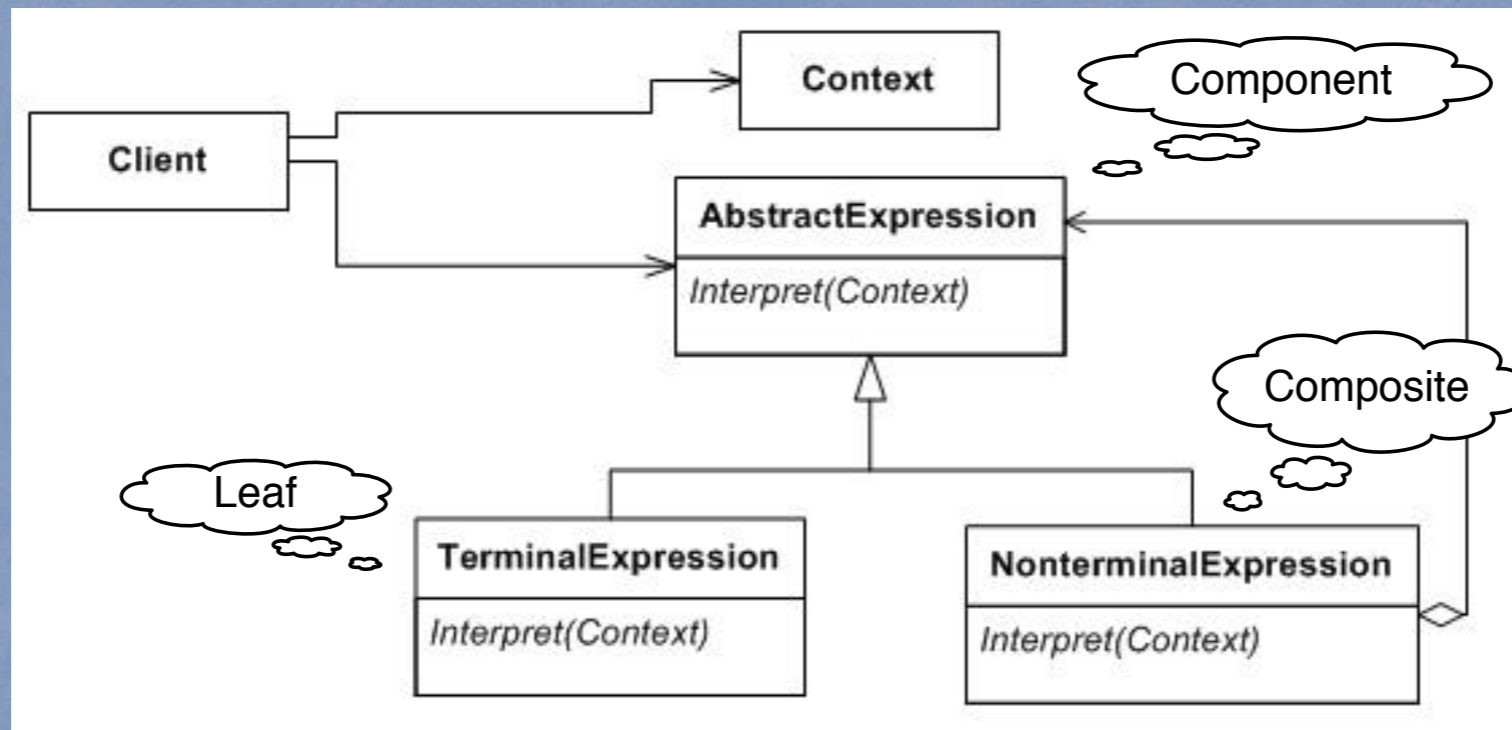


Fig. 4. The Interpreter Pattern with references to Composite (Figure 2).⁷

Interpreter Pattern

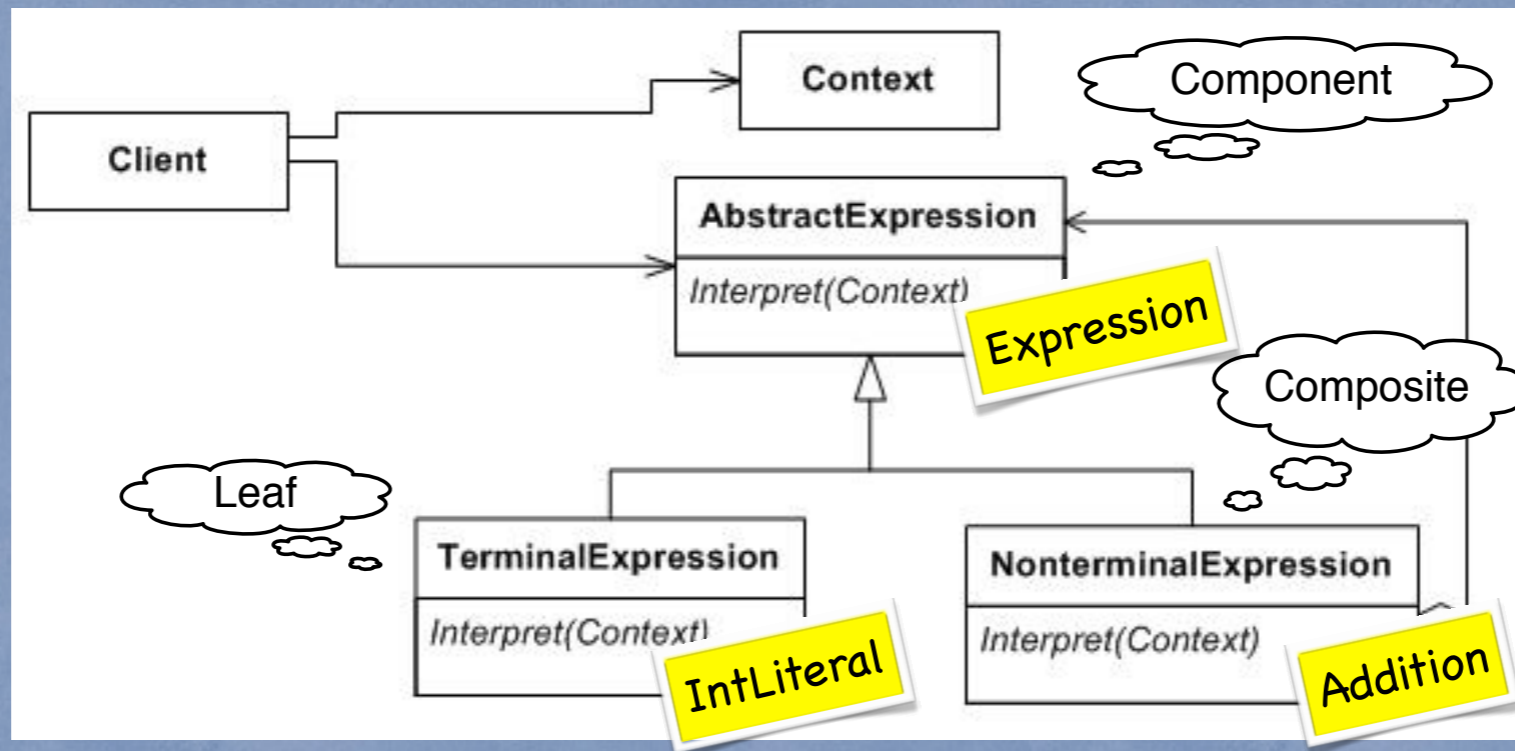


Fig. 4. The Interpreter Pattern with references to Composite (Figure 2).⁷

Visitor Pattern

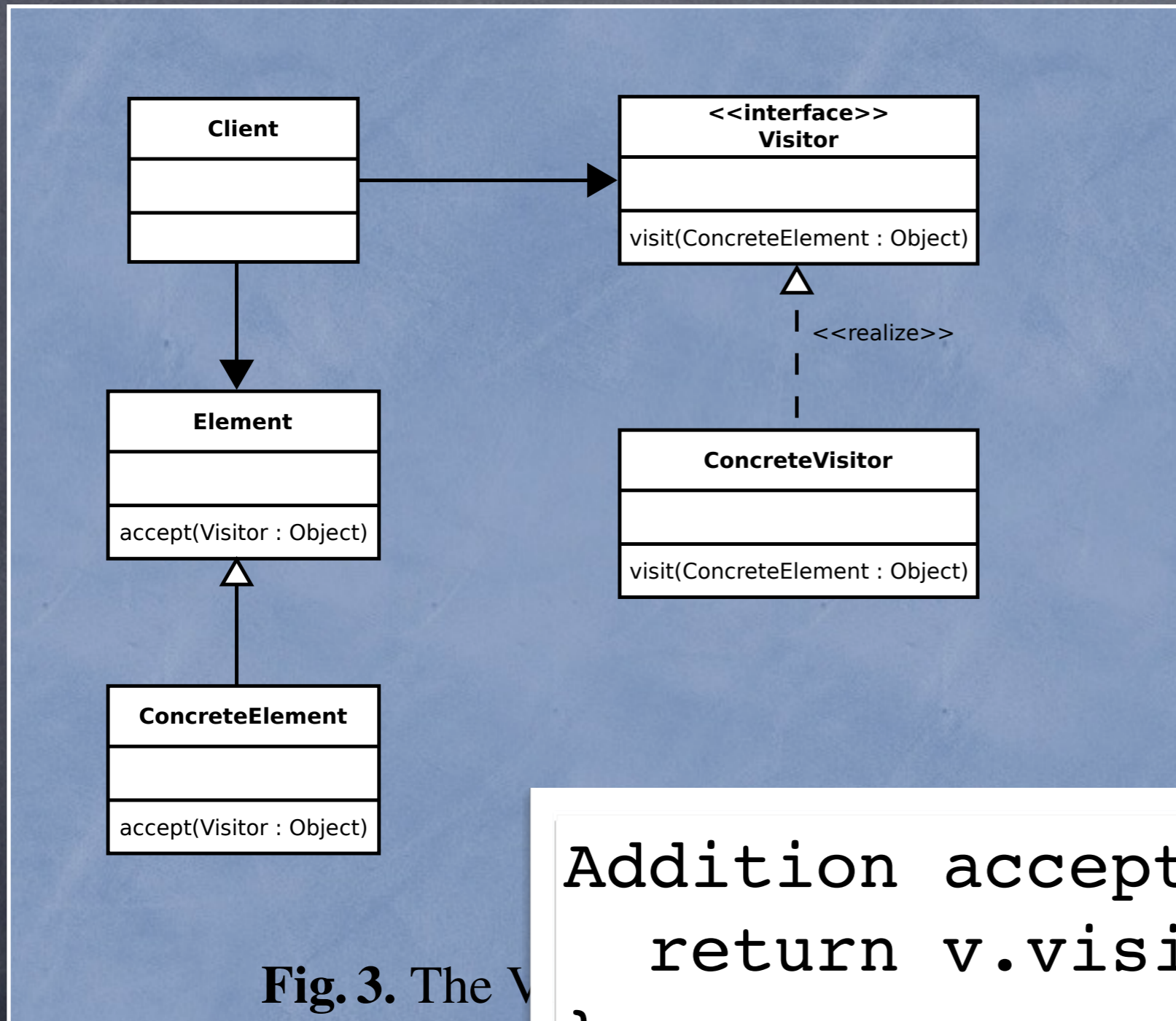
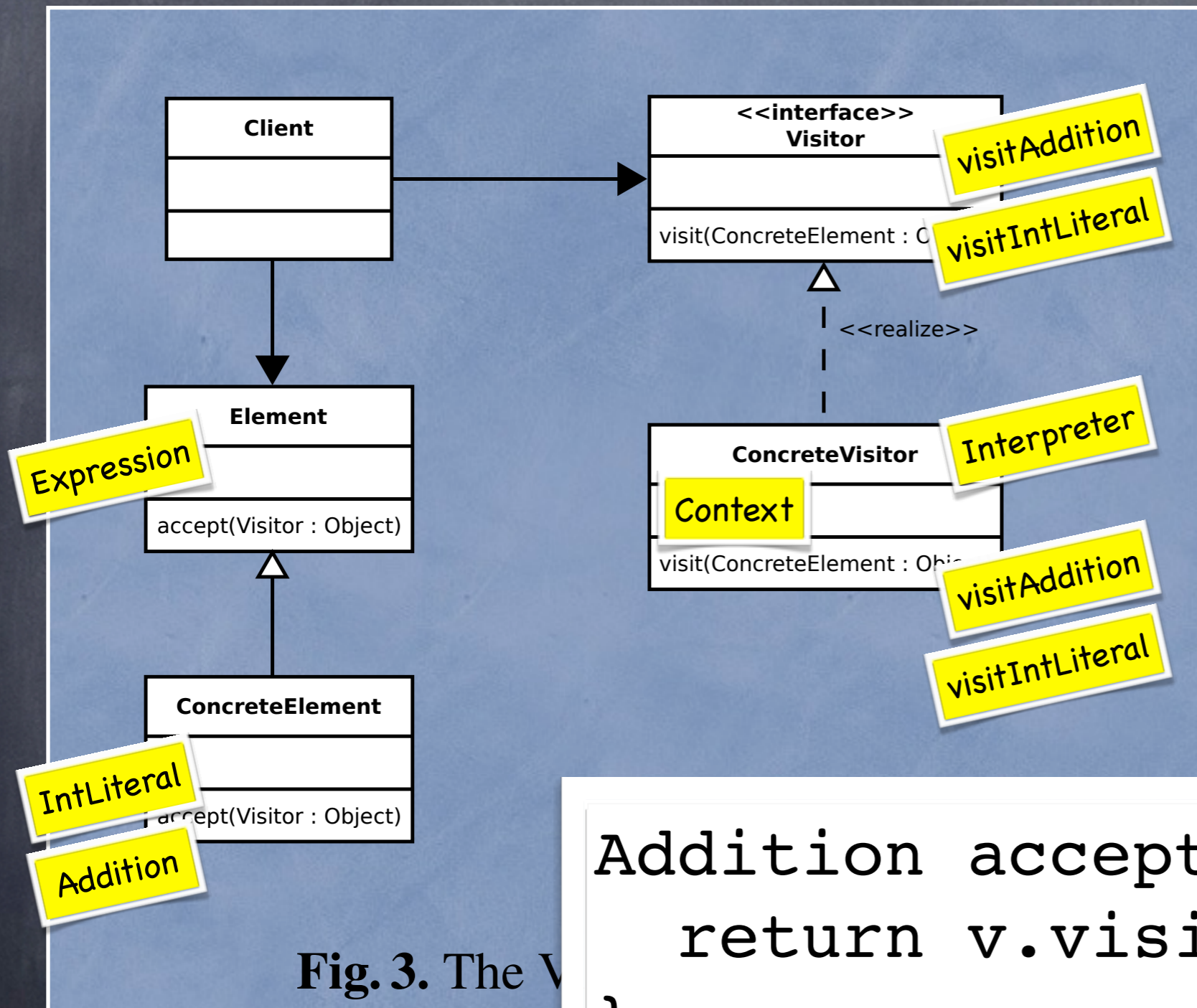


Fig. 3. The V

```
Addition accept(Visitor v) {  
    return v.visitAddition(this);  
}
```

Visitor Pattern



```

Addition accept(Visitor v) {
    return v.visitAddition(this);
}
    
```


Visitor design pattern and the
Interpreter design pattern are
functionally inter-changeable

Visitor design pattern and the
Interpreter design pattern are
functionally inter-changeable



But, they are different
in **non-functional**
properties

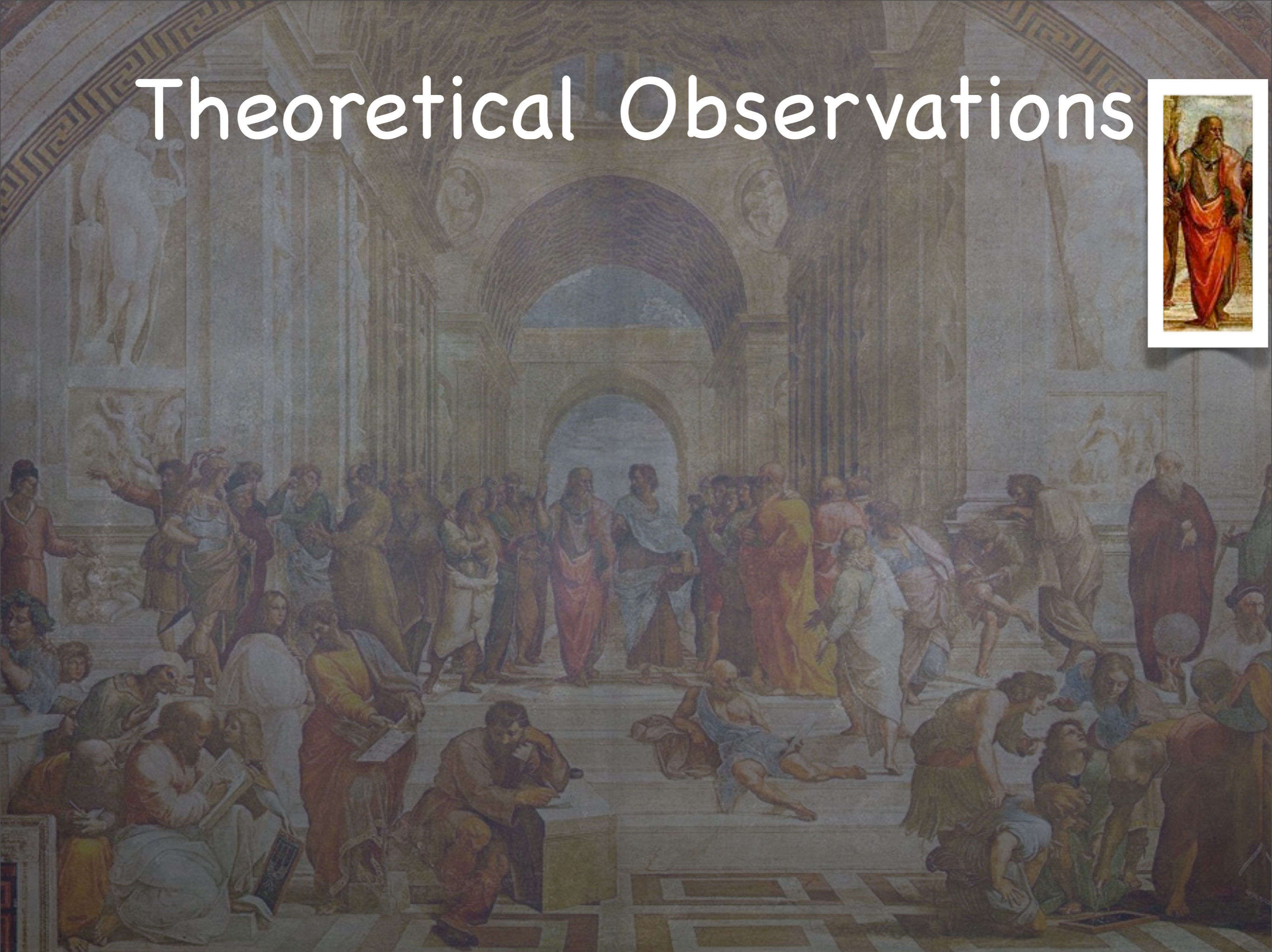
Visitor design pattern and the Interpreter design pattern are functionally inter-changeable



But, they are different in non-functional properties

And, these emergent properties tend to be difficult to predict

Theoretical Observations



Theoretical Observations



- Visitor is conceptually more complex
- Interpreter is only a small extension of composite



Theoretical Observations



- Visitor is conceptually more complex
 - Interpreter is only a small extension of composite
- Visitor encapsulates entire algorithms
 - Interpreter encapsulates language constructs

Theoretical Observations



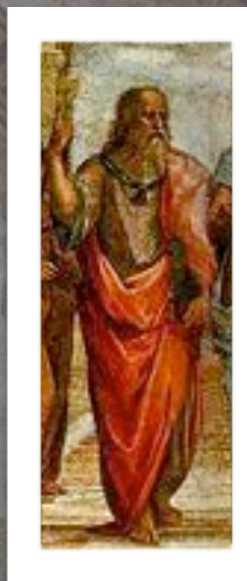
- Visitor is conceptually more complex
 - Interpreter is only a small extension of composite
- Visitor encapsulates entire algorithms
 - Interpreter encapsulates language constructs
- Visitor's decoupling implies dynamic indirection
 - Interpreter has less dynamic dispatch

Theoretical Observations



- Visitor is conceptually more complex
 - Harder to maintain, right?
 - Interpreter is only a small extension of composite
- Visitor encapsulates entire algorithms
 - Interpreter encapsulates language constructs
- Visitor's decoupling implies dynamic indirection
 - Interpreter has less dynamic dispatch

Theoretical Observations



- Visitor is conceptually more complex

Harder to maintain, right?

- Interpreter is only a small extension of composite

- Visitor encapsulates...

Easy for adding algorithm, hard for adding new language construct, right?

- Interpreter encapsulates language constructs

- Visitor's decoupling implies dynamic indirection

- Interpreter has less dynamic dispatch

Theoretical Observations



- Visitor is conceptually more complex

Harder to maintain, right?

- Interpreter is only a small extension of composite

- Visitor encapsulates

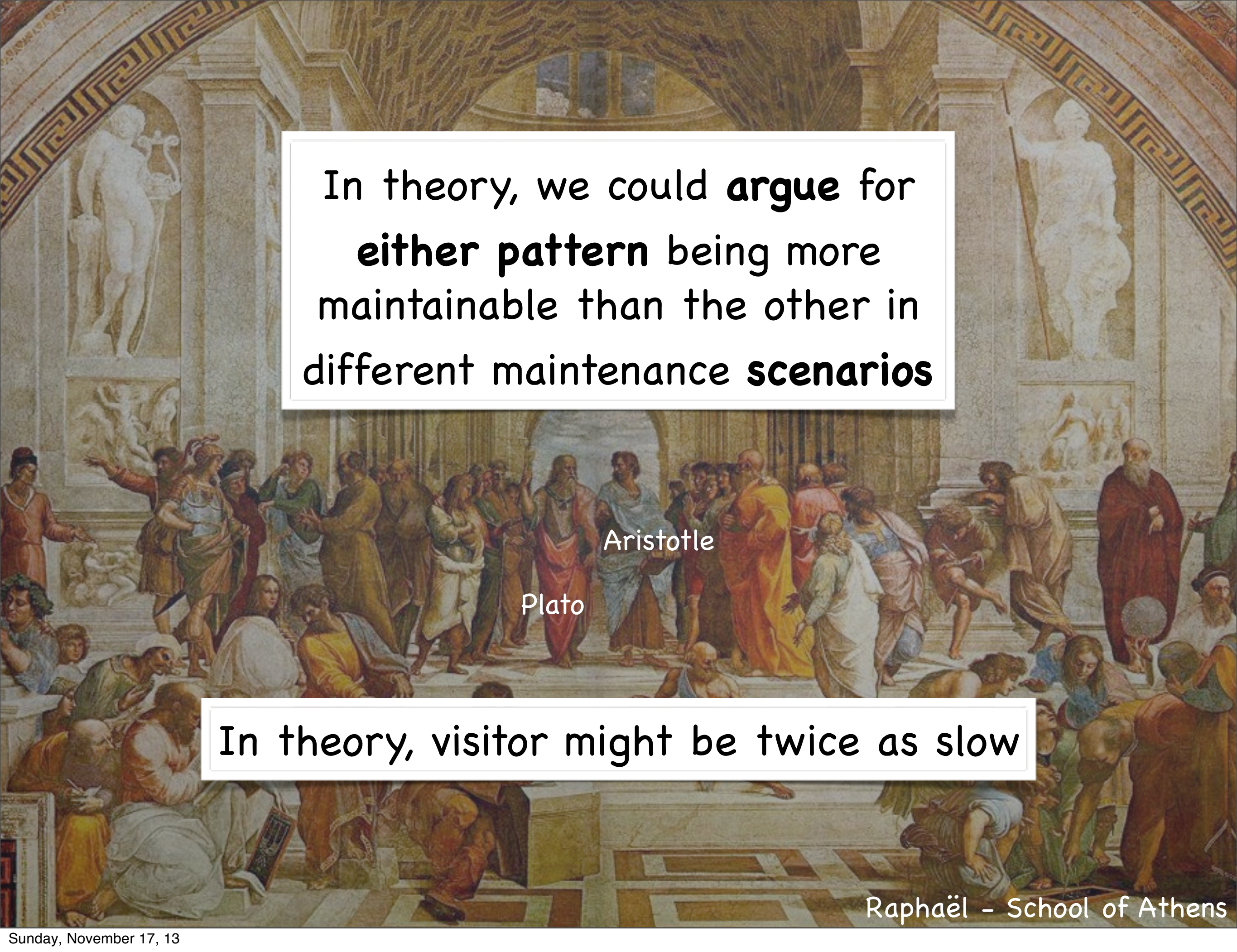
Easy for adding algorithm, hard for adding new language construct, right?

- Interpreter encapsulates language constructs

- Visitor's decision is dynamic indirection

Slower, right?

- Interpreter has less dynamic dispatch

The background of the slide is Raphael's fresco 'The School of Athens'. It depicts a group of ancient Greek philosophers in a grand, vaulted hall. In the center, Plato and Aristotle are walking towards the viewer. Plato is on the left, pointing upwards, and Aristotle is on the right, gesturing downwards. They are surrounded by other philosophers in various poses of teaching and learning. The architecture features arches with statues in niches and a coffered ceiling.

In theory, we could **argue** for
either pattern being more
maintainable than the other in
different maintenance **scenarios**

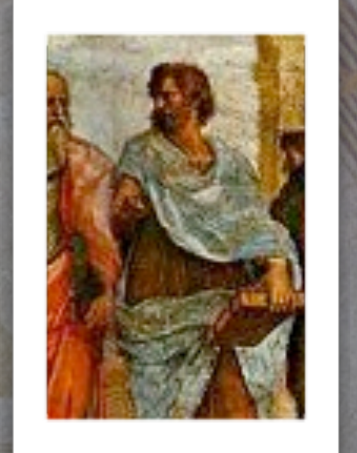
Aristotle

Plato

In theory, visitor might be twice as slow

Raphaël - School of Athens

Empirical Observations



- Visitor-based interpreter is complex
- Many visitors classes
- Main interpreter is a "God class"
- Interpreter should run faster than this

Why this experiment?

- ⑤ Is the difference between Interpreter and Visitor **causing** a part of these two problems, or not at all?



- ⑤ How does one answer such a question?

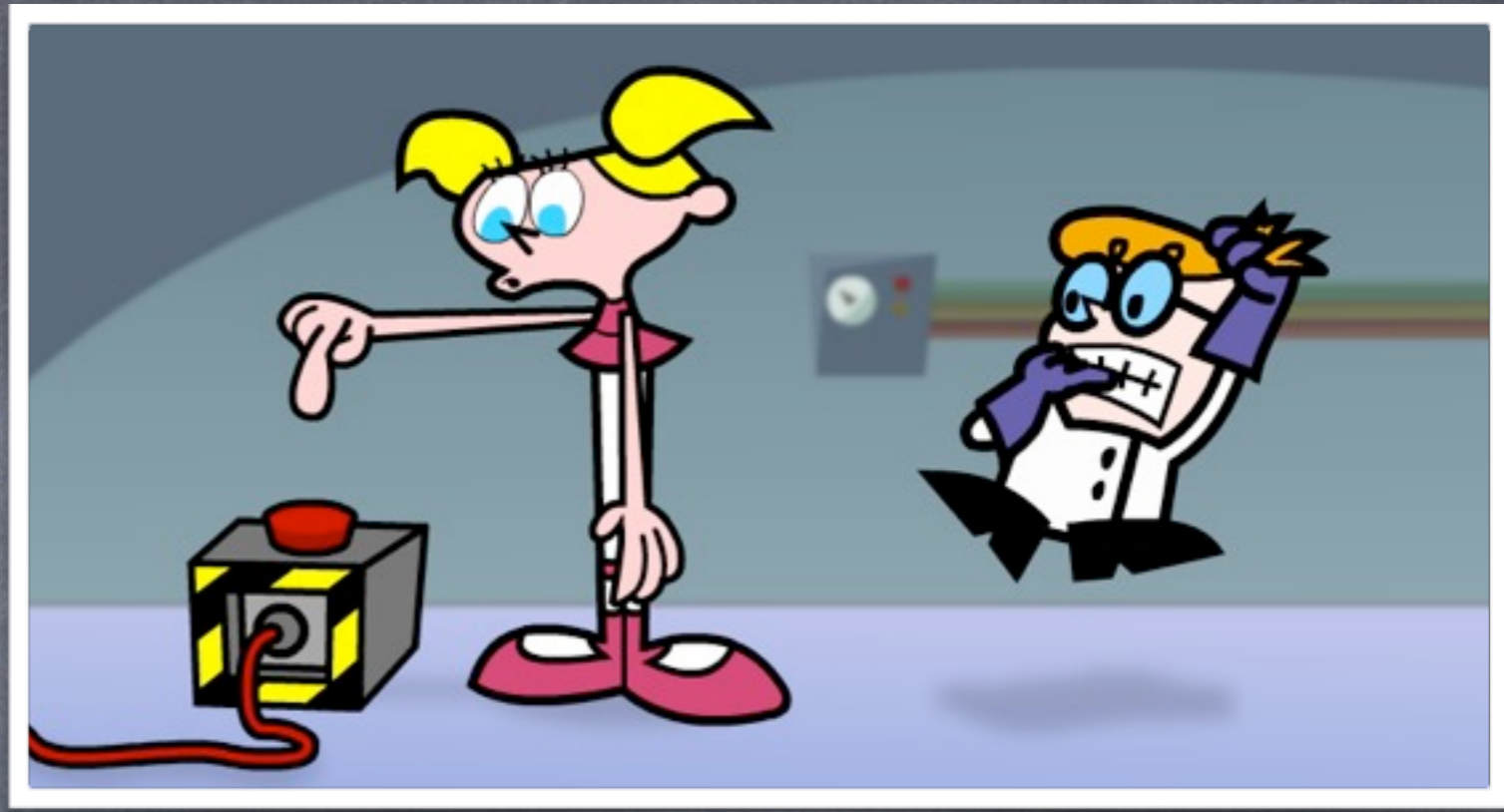
Why this lab setup?

Observing software “in the wild”



- In reality, there exist no two different versions of the same interpreter
- In reality, there are many other factors influencing maintenance and efficiency other than this design choice
- Reality is perhaps easy to see, but it is **very hard to understand**

Lab Experiment



- In a lab we may isolate a factor
- In the lab we may focus on the effect
- In the lab we can observe causality more directly

Possible lab experiments



- Source code metrics for maintainability
- Construction of Cognitive Models
- New method based on "Evolution complexity"

Source Code Metrics are (perhaps) good for observing reality **statistically**, but not for observing implications of design choices

Source Code Metrics are (perhaps) good for observing reality **statistically**, but not for observing implications of design choices

Maintainability Index I&II

Source Code Metrics are (perhaps) good for observing reality **statistically**, but not for observing implications of design choices

Maintainability Index I&II

SIG maintainability model

Source Code Metrics are (perhaps) good for observing reality **statistically**, but not for observing implications of design choices

Maintainability Index I&II

Maintenance Complexity Metric

SIG maintainability model

Source Code Metrics are (perhaps) good for observing reality **statistically**, but not for observing implications of design choices

Maintainability Index I&II

Maintenance Complexity Metric

SIG maintainability model

Computing and aggregating metrics values, **independent** of maintenance scenario, predicting long-term expectations on maintenance costs

Source Code Metrics are (perhaps) good for observing reality statistically, but not for observing implications of design choices

Maintainability Index I&II

Maintenance Complexity Metric

SIG maintainability model

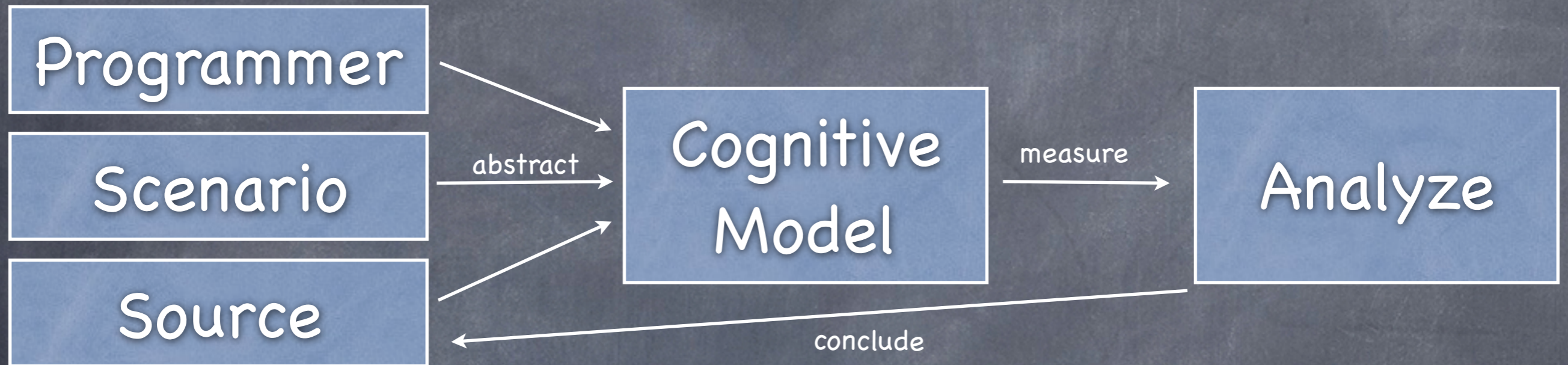
Computing and aggregating metrics values, independent of maintenance scenario, predicting long-term expectations on maintenance costs

If validated and calibrated these make sense on huge long-lived systems, but they say **nothing about the next maintenance scenario** applied to the system

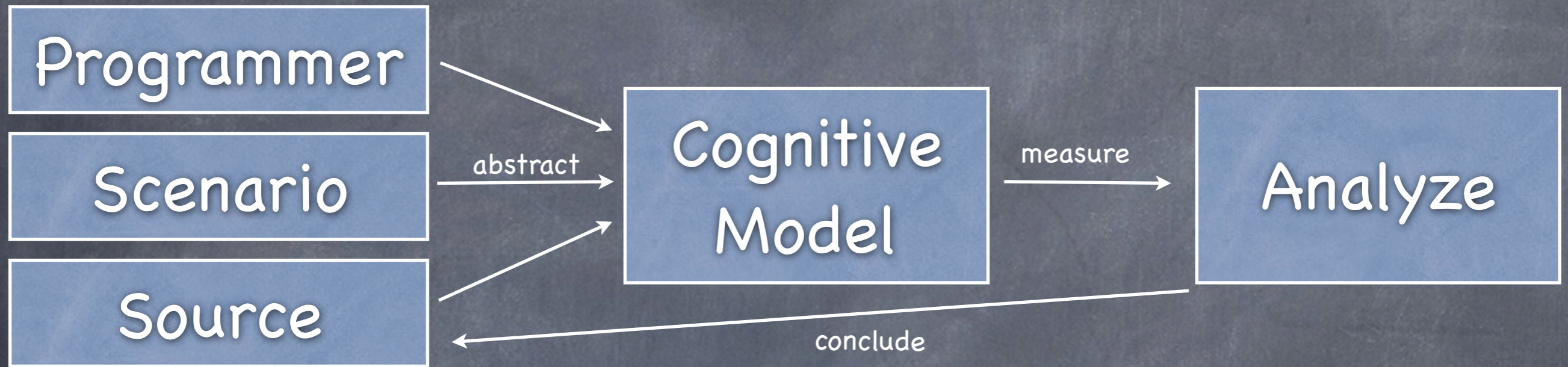


The Problem

What about using **Cognitive Models** of understanding the source code then?



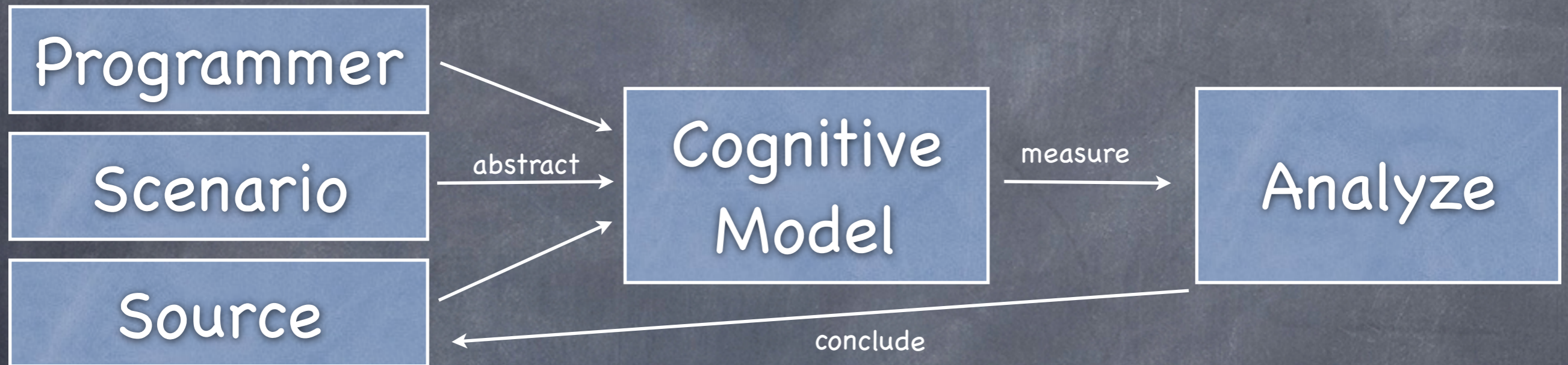
What about using **Cognitive Models** of understanding the source code then?



Unfortunately, we neither understand nor trust these models



What about using **Cognitive Models** of understanding the source code then?



Unfortunately, we neither understand nor trust these models



There is no Free Lunch.

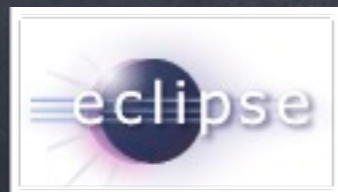
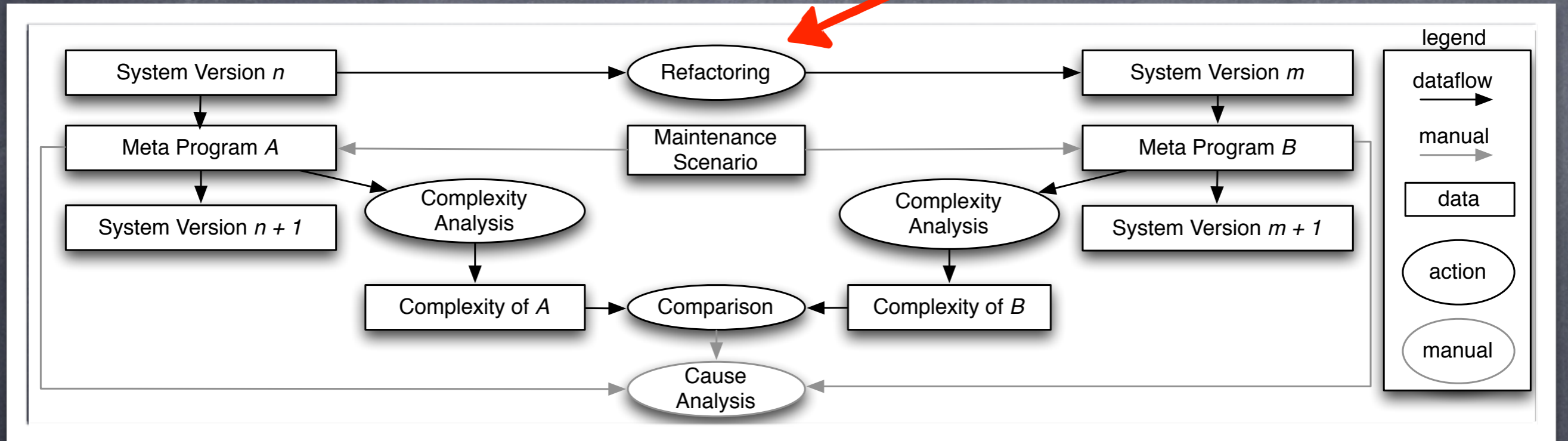
IDE + source code + human => very complex models of cognition

Our Lab Setup

- Refactoring to get two versions
- Applying realistic maintenance scenarios
- Measuring the optimal “effort” of doing maintenance
- Analyzing differences by tracing back to code

Isolating the variable

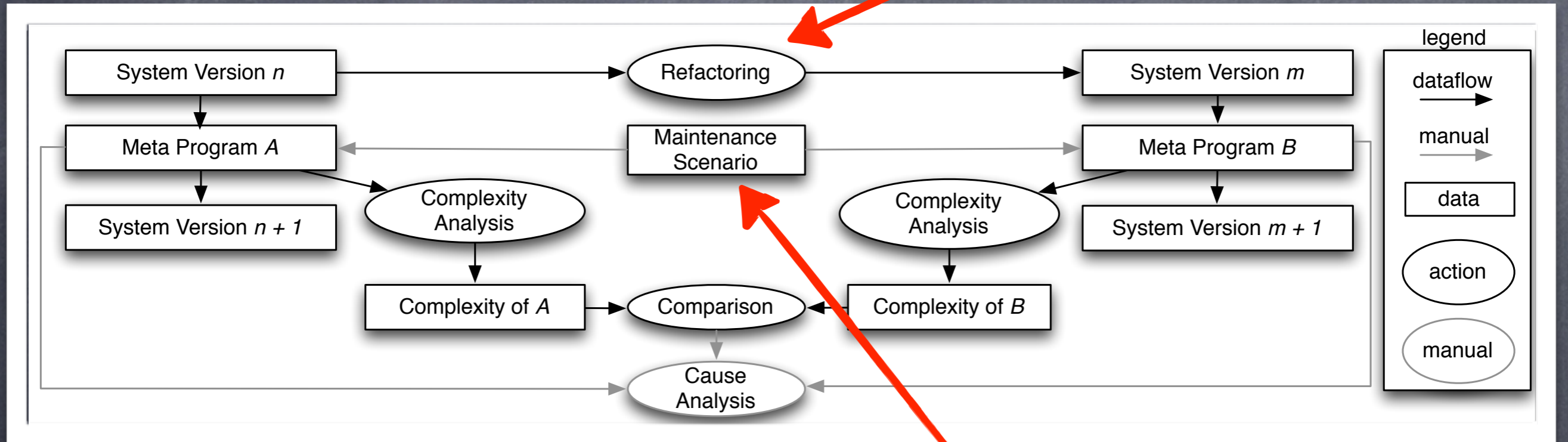
Key enabler



Rascal & JDT to implement Visitor
to Interpreter refactoring

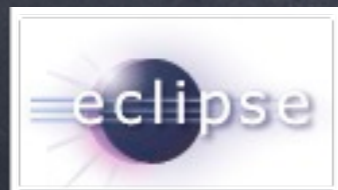
Isolating the variable

Key enabler

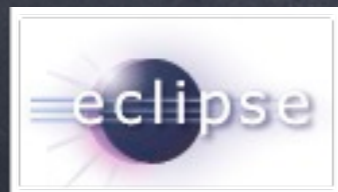
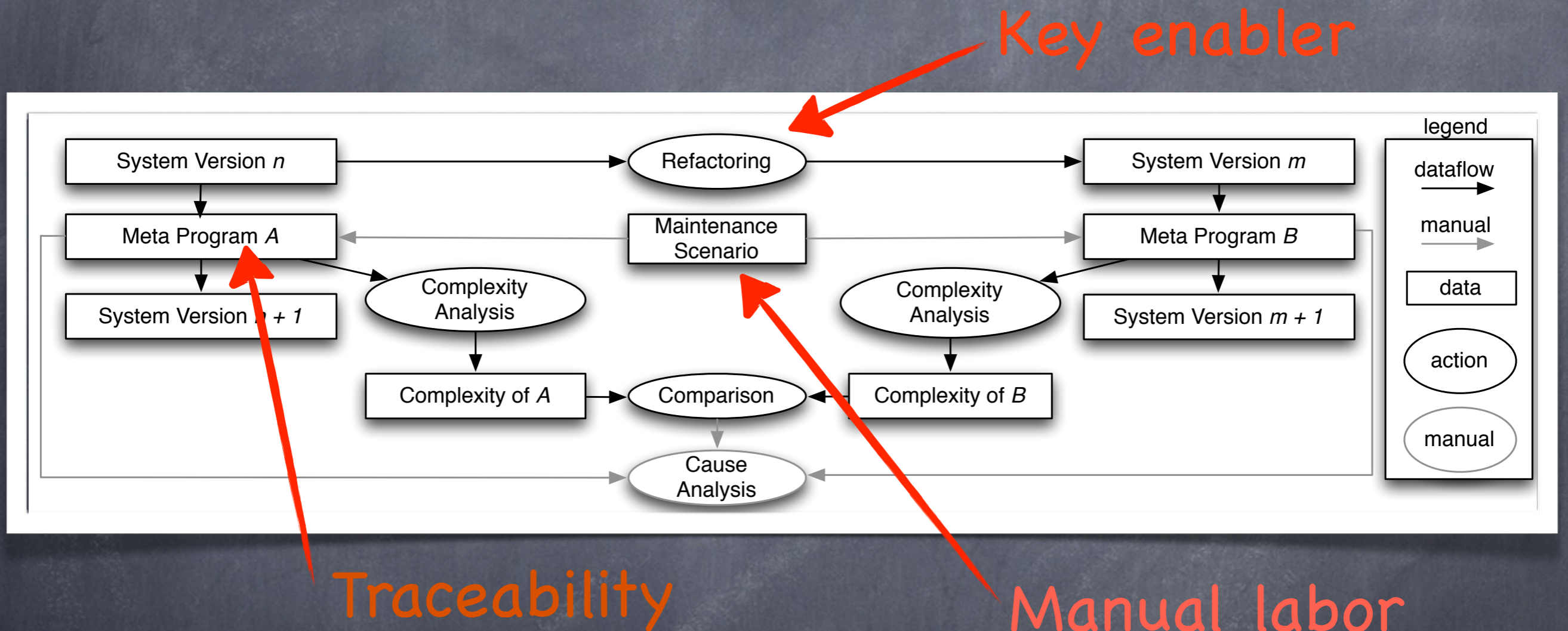


Manual labor

Rascal & JDT to implement Visitor
to Interpreter refactoring



Isolating the variable



Rascal & JDT to implement Visitor
to Interpreter refactoring

"Complexity of Maintenance"



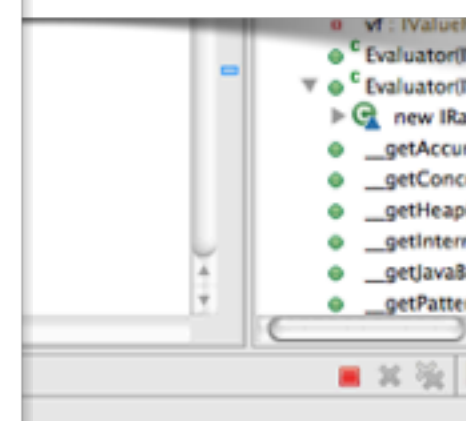
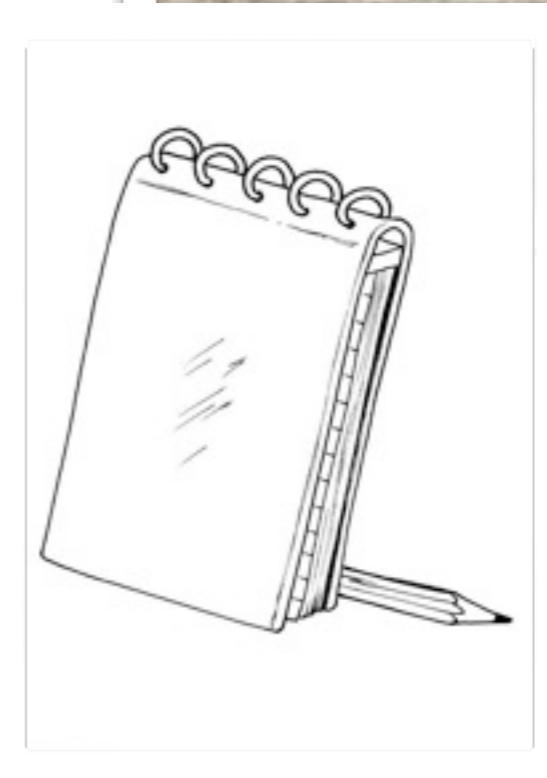
Precise definitions in the paper

- Maintainability = Understandability + Modifiability
- Complexity of a maintenance scenario is =
 - #steps to learn facts about a Program +
 - #steps to modify the Program
- Reify steps as a "Meta Program" that operates the IDE

Inspired by "Measuring Software Flexibility"
by Mens & Eden, IEE Software 2006

Collecting data

```
Java - rascal/src/org/rascalimpl/interpreter/Evaluator.java - Eclipse - /Users/jurgenv/Wa  
rchy JUnit  
AtEndOfLineRequirement.java AtColumnRequirement.java AtStartOfLineRequirement.java ParserGenerator.java Meta  
rascal src org.rascalimpl.interpreter Evaluator  
110 public class Evaluator extends NullASTVisitor<Result<IValue>> implements IEvaluator<Result<IValue>  
111     private IValueFactory vf;  
112     private static final TypeFactory tf = TypeFactory.getInstance();  
113     protected Environment currentEnv;  
114     private StrategyContextStack strategyContextStack;  
115  
116     private final GlobalEnvironment heap;  
117     private boolean interrupt = false;  
118  
119     private final JavaBridge javaBridge;  
120  
121     private AbstractAST currentAST; // used in runtime error messages  
122  
123     private static boolean doProfiling = false;  
124     private Profiler profiler;  
125  
126     private final TypeDeclarationEvaluator typeDeclaration;  
127     protected IEvaluator<IMatchingResult> patternEvaluator;  
128  
129     private final List<ClassLoader> classLoaders;  
130     private final ModuleEnvironment rootScope;  
131     private boolean concreteListsShouldBeSpliced;  
132  
133     private final PrintWriter stderr;  
134     private final PrintWriter stdout;  
135  
136     private ITestResultListener testReporter;  
137     /**  
138      * To avoid null pointer exceptions, avoid passing this directly to other classes,  
139      * the result of getMonitor() instead.  
140      */  
141     private IRascalMonitor monitor;  
142  
143  
144     private Stack<Accumulator> accumulators = new Stack<Accumulator>();  
145     private Stack<Inteaer> indentStack = new Stack<Inteaer>();
```



Results



Results

<i>S</i>	Visitor	(COM)	Interpreter	(COM)	Vis.>Int.
S1	$ci^{11}(g^2a)^2$	(18)	$m^2b(ef^2)^3(ga)^2$	(16)	yes
S1(<i>N</i>)	$ci^{11}(g^Na)^2$	(14 + 2 <i>N</i>)	$m^Nb(ef^N)^3(ga)^N$	(4 + 6 <i>N</i>)	if $N \leq 2$
S1'(<i>N</i> ,2)	$ci^{11}(g^Na)^2$	(14 + 2 <i>N</i>)	$m^N(ga)^N$	(3 <i>N</i>)	if $N \leq 14$
S1'(<i>N</i> , <i>M</i>)	$ci^{9+M}(g^Na)^M$	(10 + <i>NM</i> + 2 <i>M</i>)	$m^N(ga)^{MN}$	(<i>N</i> + 2 <i>MN</i>)	if $N \leq \frac{2M+10}{M+1}$
S2	i^2g^3iga	(8)	$i^2g^3gaig^3aiga$	(14)	no
S3	$dg^5egcg^{15}g^2a(eea)^4i^2h(ga)^3$	(43)	$d(ig)^2a(iga)^{15}(ig)^3gai$ $(ig^2)a(igg)^2anigaih(ga)^3$	(83)	no
S3'	$d(ga)^5egac(ga)^{15}(ga)^2$ $(eea)^4i^2h(ga)^3$	(70)	$d(ig)^2a(iga)^{15}(ig)^3gai$ $(ig^2)a(igg)^2anigaih(ga)^3$	(83)	no
S4	$mg^{11}a$	(13)	$bga(bga)^{11}$	(36)	no
S5	$biga$	(4)	bga	(3)	yes

Table 2. A comparison of all maintenance programs (see Table 1).

Results

S	Visitor	(COM)	Interpreter	(COM)	Vis.>Int.
S1	$ci^{11}(g^2a)^2$	(18)	$m^2b(ef^2)^3(ga)^2$	(16)	yes
S1(N)	$ci^{11}(g^Na)^2$	$(14 + 2N)$	$m^2b(ef^2)^3(ga)^2$	$(4 + 6N)$	if $N \leq 2$
S1'(N,2)	$ci^{11}(g^Na)^2$	$(14 + 2N)$	$m^N(ga)^N$	$(3N)$	if $N \leq 14$
S1'(N,M)	$ci^{11}(g^Na)^2$	$(10 + NM + 2M)$	$m^N(ga)^{MN}$	$(N + 2MN)$	if $N \leq \frac{2M+10}{M+1}$
S2	i^2g^3iga	(8)	$i^2g^3gaig^3aiga$	(14)	no
S3	$dg^5egcg^{15}g^2a(eea)^4i^2h(ga)^3$	(43)	$d(ig)^2a(iga)^{15}(ig)^3gai$ $(ig^2)a(igg)^2anigaih(ga)^3$	(83)	no
S3'	$d(ga)^5egac(ga)^{15}(ga)^2$ $(eea)^4i^2h(ga)^3$	(70)	$d(ig)^2a(iga)^{15}(ig)^3gai$ $(ig^2)a(igg)^2anigaih(ga)^3$	(83)	no
				(6)	no
				(3)	yes

steps to
add N
constructs
to Visitor
 $14 + 2N$

steps to add
N constructs
to
Interpreter
 $3N$

break-even at
 $N = 14$



Why trust this?



Why trust this?

- **Construct validity:** are all aspects of maintainability observable in this experiment?



Why trust this?

- **Construct validity:** are all aspects of maintainability observable in this experiment?
- **Internal validity:** did you really do the best job possible in all scenarios?



Why trust this?

- **Construct validity:** are all aspects of maintainability observable in this experiment?
- **Internal validity:** did you really do the best job possible in all scenarios?
- **External validity:** does this say anything about the next interpreter I write in Java? The next maintenance? What if I don't use Eclipse? What if <blablabla>?



Why trust this?

other factors may still dominate, but that is why we compare two equivalent systems

- **Construct validity:** are all aspects of maintainability observable in this experiment?
- **Internal validity:** did you really do the best job possible in all scenarios?
- **External validity:** does this say anything about the next interpreter I write in Java? The next maintenance? What if I don't use Eclipse? What if <blablabla>?



Why trust this?

• **Construct validity:** are all aspects of maintainability observable in this experiment?

other factors may still dominate, but that is why we compare two equivalent systems

• **Internal validity:** did you really do the job possible in all scenarios?

there is no proof of that - we invite you to reproduce or invalidate the results

• **External validity:** does this say anything about the next interpreter I write in Java? The next maintenance? What if I don't use Eclipse? What if <blablabla>?



Why trust this?

- **Construct validity:** are all aspects of maintainability observable in this experiment?
- **Internal validity:** did you really do the job possible in all scenarios?
- **External validity:** does this say anything about the next interpreter I write in Java? The next maintenance? What if I do a case Eclipse? What if <blablabla>?

other factors may still dominate, but that is why we compare two equivalent systems

there is no proof of that - we invite you to reproduce or invalidate the results

we do **not** know

Summary

*given the scope of the experiment

Summary

- We used Rascal to build a refactoring tool

*given the scope of the experiment

Summary

- We used **Rascal** to build a **refactoring tool**
- to **isolate** the difference between **Visitor & Interpreter**

*given the scope of the experiment

Summary

- We used **Rascal** to build a **refactoring tool**
- to **isolate** the difference between **Visitor & Interpreter**
- and using the **"Complexity of Maintenance"** method

*given the scope of the experiment

Summary

- We used **Rascal** to build a **refactoring tool**
- to **isolate** the difference between **Visitor & Interpreter**
- and using the **"Complexity of Maintenance"** method
- we found that **Visitor is better***

*given the scope of the experiment



Summary

- We used Rascal to build a refactoring tool
- to isolate the difference between Visitor & Interpreter
- and using the "Complexity of Maintenance" method
- we found that Visitor is better*

*given the scope of the experiment

<http://www.rascal-mpl.org>



Summary

Feedback on paper
and research method
more than welcome!

- We used Rascal to build a refactoring tool
- to isolate the difference between Visitor & Interpreter
- and using the "Complexity of Maintenance" method
- we found that Visitor is better*

*given the scope of the experiment



Centrum Wiskunde & Informatica