Data-Driven Synthesis for Object-Oriented Frameworks

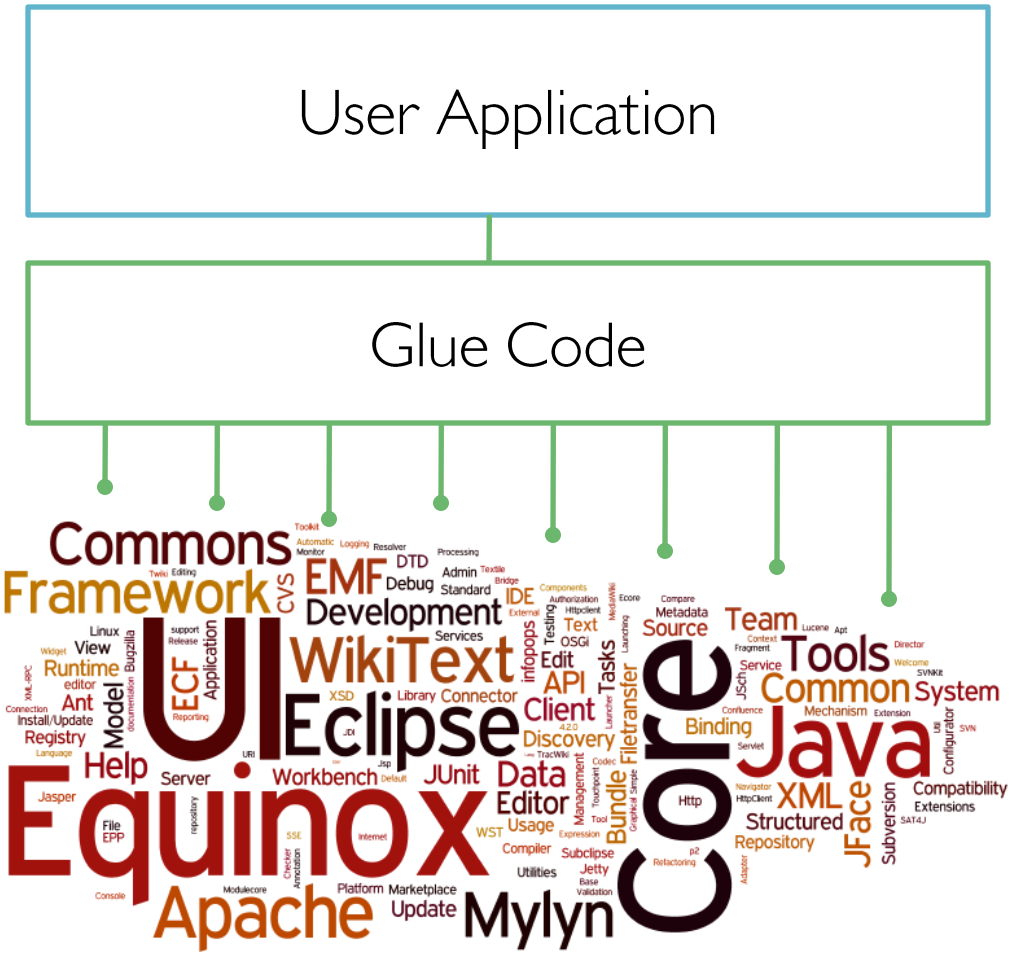
Armando Solar-Lezama, Zhilei Xu, Kuat Yessenov

OOPSLA 2011

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

CSAIL

Programming with Frameworks



Example: Eclipse Syntax Highlighting

File Edit Source Navigate Search Project Run Window Help

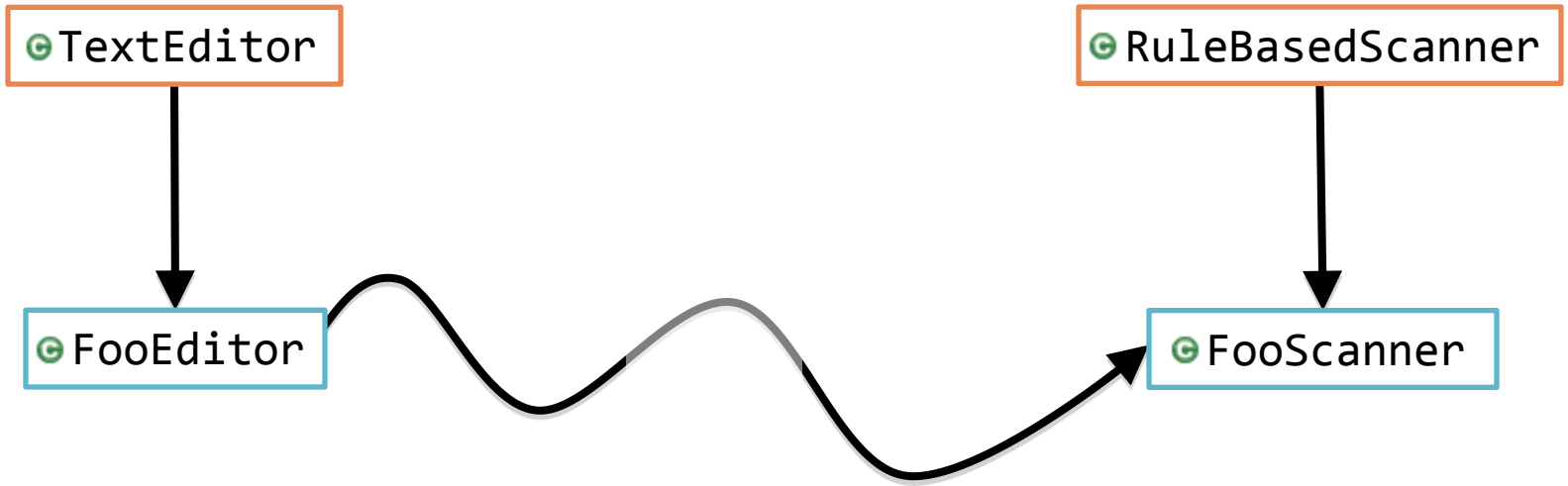
NewFile.xml

```
<!--  This is a simple configuration file -->

<?xml version="1.0" encoding="UTF-8"?>

<body>
<name>Configuration</name>
<property p="234">Property</property>
<outer>
    <inner>inner</inner>
</outer>
</body>
```

Design Source

Writable Sm...rt

What Novice Programmer Knows



~~FooEditor setScanner(FooScanner);~~

Editor and Scanner Glue Code

```
class FooConfig extends SourceViewerConfiguration {
  @Override
  public getPresentationReconciler(…) {
    DefaultDamagerRepairer dr = new DefaultDamagerRepairer(new FooScanner());
    PresentationReconciler rcr = new PresentationReconciler();
    rcr.setDamager(dr, …);
    rcr.setRepairer(dr, …);
    return rcr;
  }
}
class FooEditor extends TextEditor {
  @Override
  void initializeEditor() {
    setSourceViewerConfiguration(new FooConfig());
  }
}
class FooScanner extends RuleBasedScanner {…}
```

Welcome to MatchMaker!

Please specify source and target classes:

Source: org.eclipse.ui.editors.text.TextEditor

Target: org.eclipse.jface.text.rules.RuleBasedScanner

(type auto-completion is case-sensitive; click button below to generate code)

Match!

```
class MyConfig extends SourceViewerConfiguration {
  @Override
  public getPresentationReconciler(ISourceViewer arg0) {
    DefaultDamagerRepairer dr = new DefaultDamagerRepairer(new MyScanner());
    PresentationReconciler rcr = new PresentationReconciler();

    rcr.setRepairer(dr, ??);
    return rcr;
  }
}
class MyEditor extends TextEditor {
  @Override
  void initializeEditor() {
    setSourceViewerConfiguration(new MyConfig());
  }
}
class MyScanner extends RuleBasedScanner {}
```

Learn from Observation

User applications

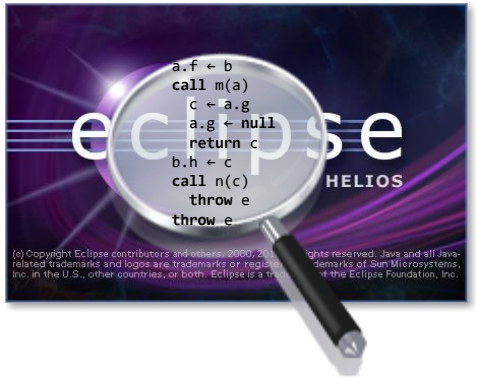
reuse the framework

Monkey See,
Monkey Do

Observe how the user application

dynamically exercises the framework

to infer the glue code.

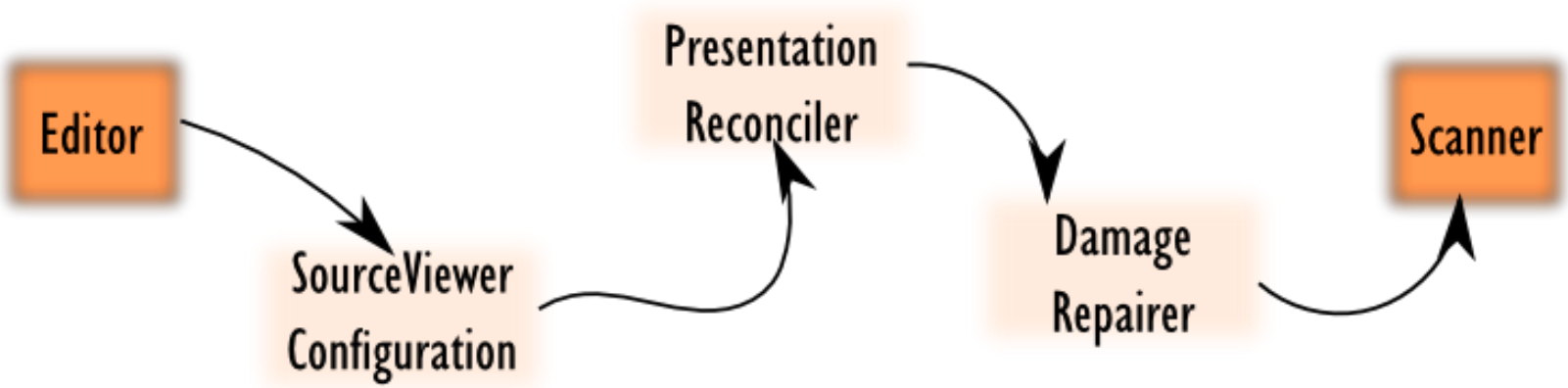


Data-Driven Approach

Query: two types

Answer: (imperfect)
Glue Code

MatchMaker Hypothesis



Objects interact via **pointer chains**

Critical chain is the first chain between two objects

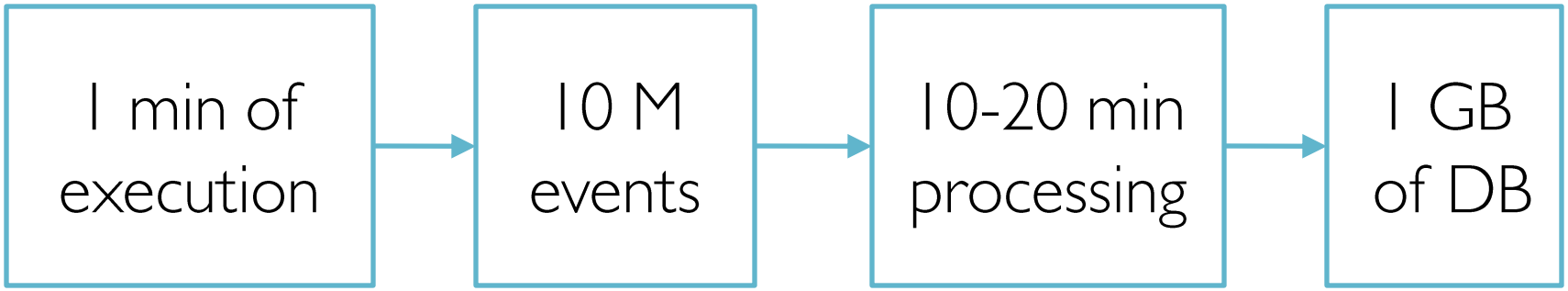
MatchMaker Algorithm

0	*Collect* execution data	Set-up
1	Build *heap series* and call tree	
2	Find *critical chains* between A and B	Query: "How do I connect A and B?"
3	*Thin-slice* events creating the chain	
4	*Project* the slice to the user code.	

Output: synthesized code snippets modulo simple transformations

Data Collection

- Dynamic byte code instrumentation using ASM
- Records internal program events:
  - Field reads and writes
  - Method enters and exits, exceptions
- Filters out internal library events (e.g. JDK)



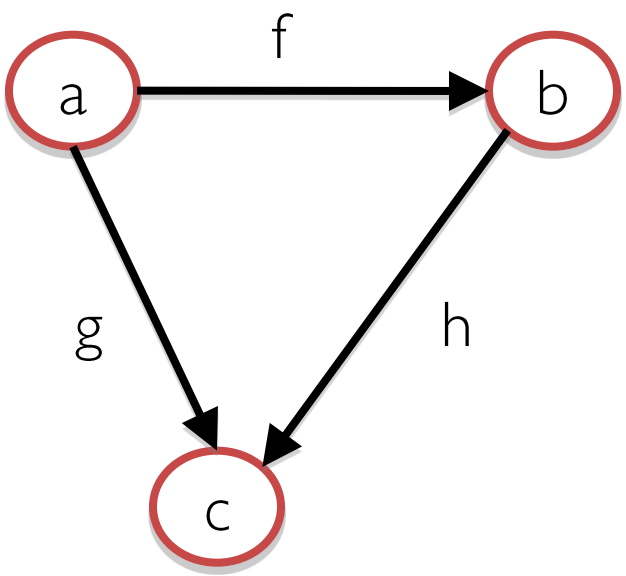
Data Analysis

Call Tree:

```
1  a.f ← b
2.  call m(a)
3.    c ← a.g
4.    a.g ← null
5.    return c
6.  b.h ← c
7.  call n(c)
8.    throw e
9.  throw e
```

Heap Series:

Statements update the heap



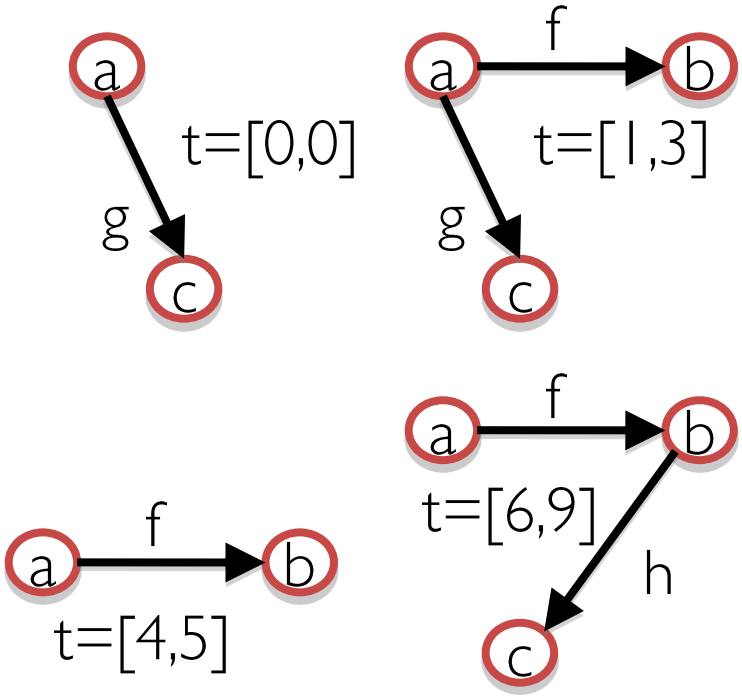
Data Analysis

Call Tree:

```
1.  a.f ← b
2.  call m(a)
3.    c ← a.g
4.    a.g ← null
5.    return c
6.  b.h ← c
7.  call n(c)
8.    throw e
9.  throw e
```

Heap Series:

Combine heap configurations



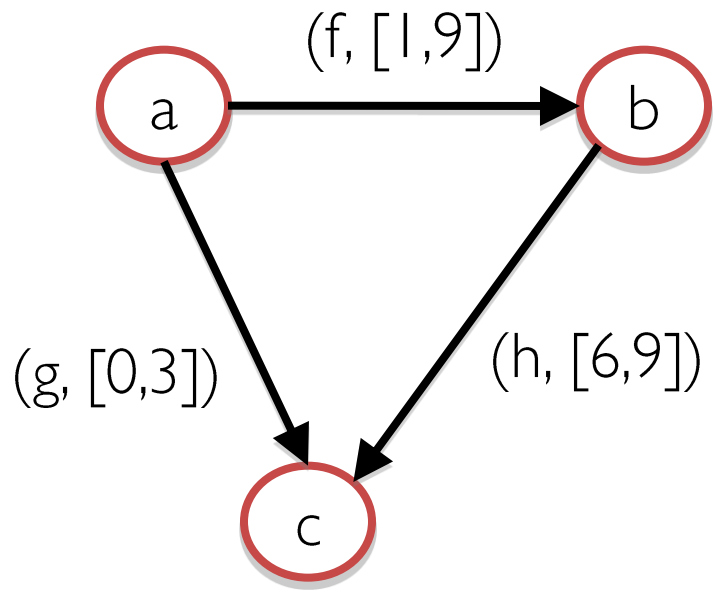
Data Analysis

Call Tree:

```
1.  a.f ← b
2.  call m(a)
3.    c ← a.g
4.    a.g ← null
5.    return c
6.  b.h ← c
7.  call n(c)
8.    throw e
9.  throw e
```

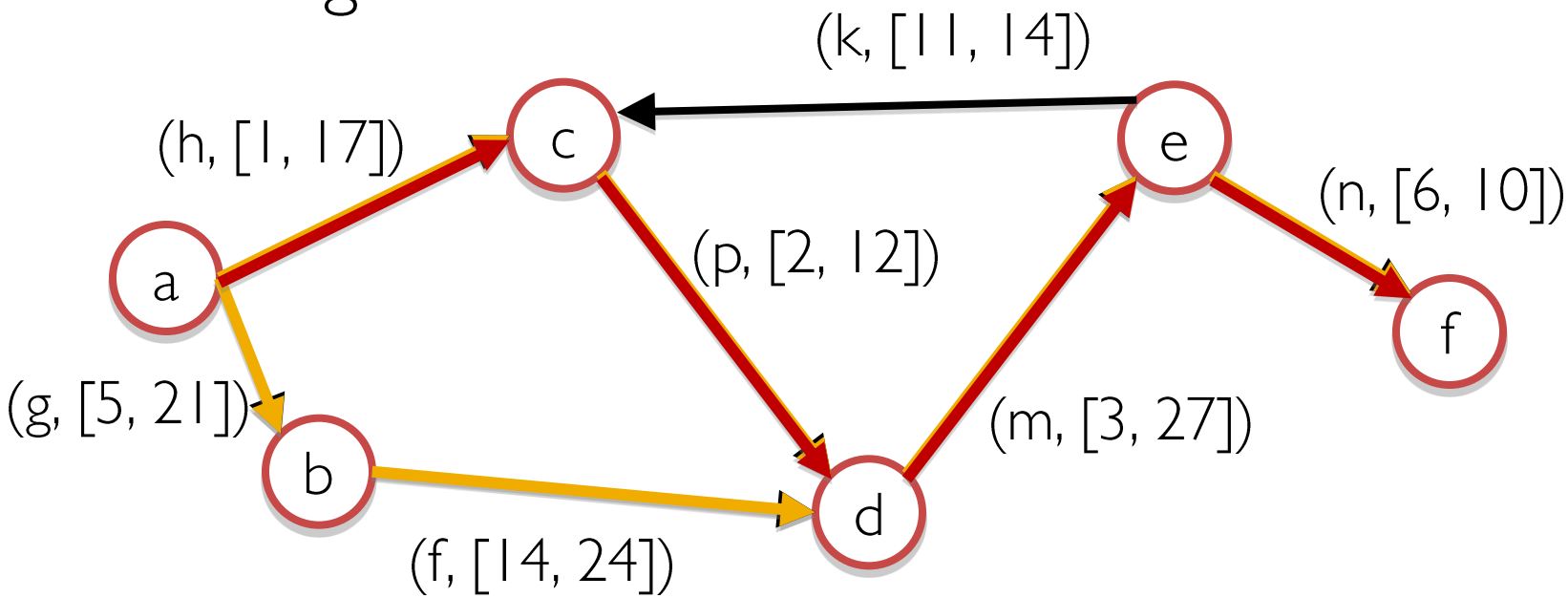
Heap Series:

Graph on objects labeled with fields and time intervals



Critical Chain in Heap Series

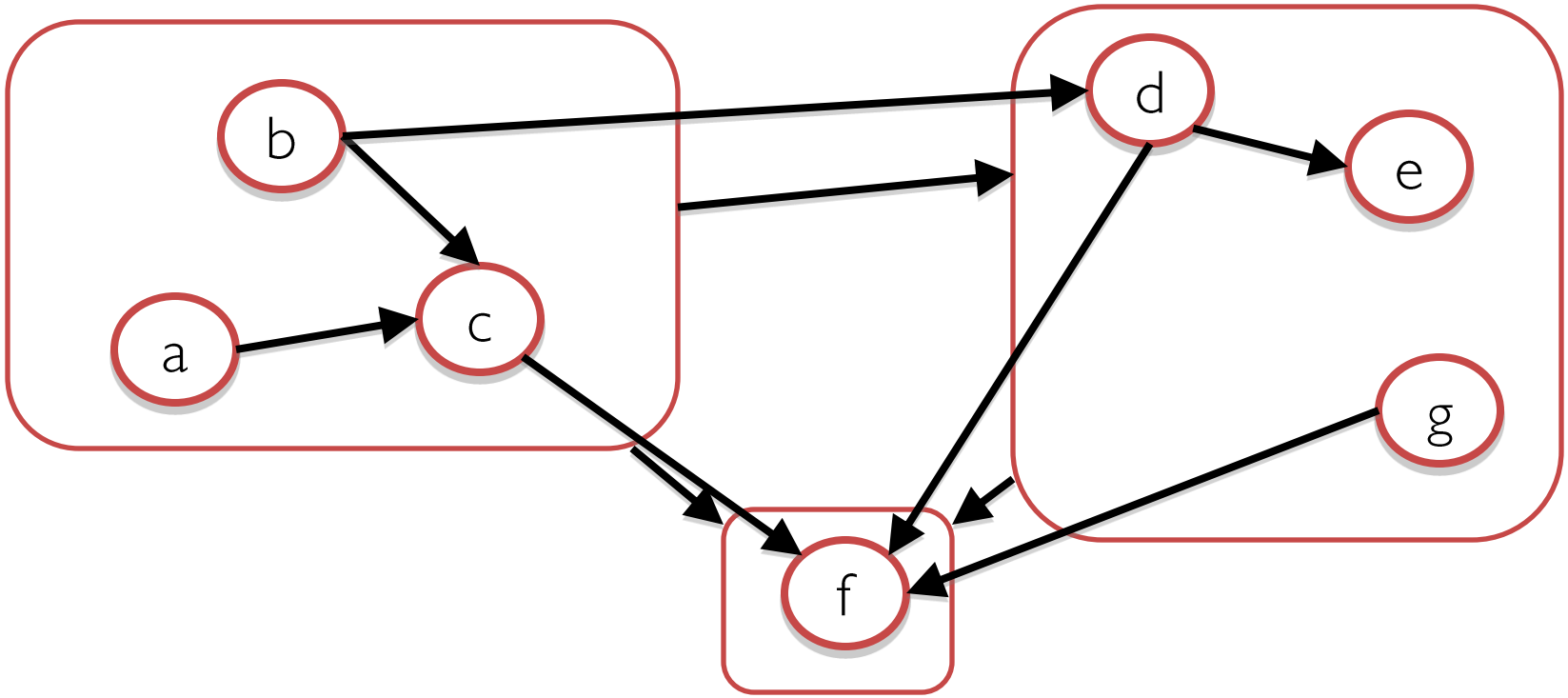
Naïve algorithm:



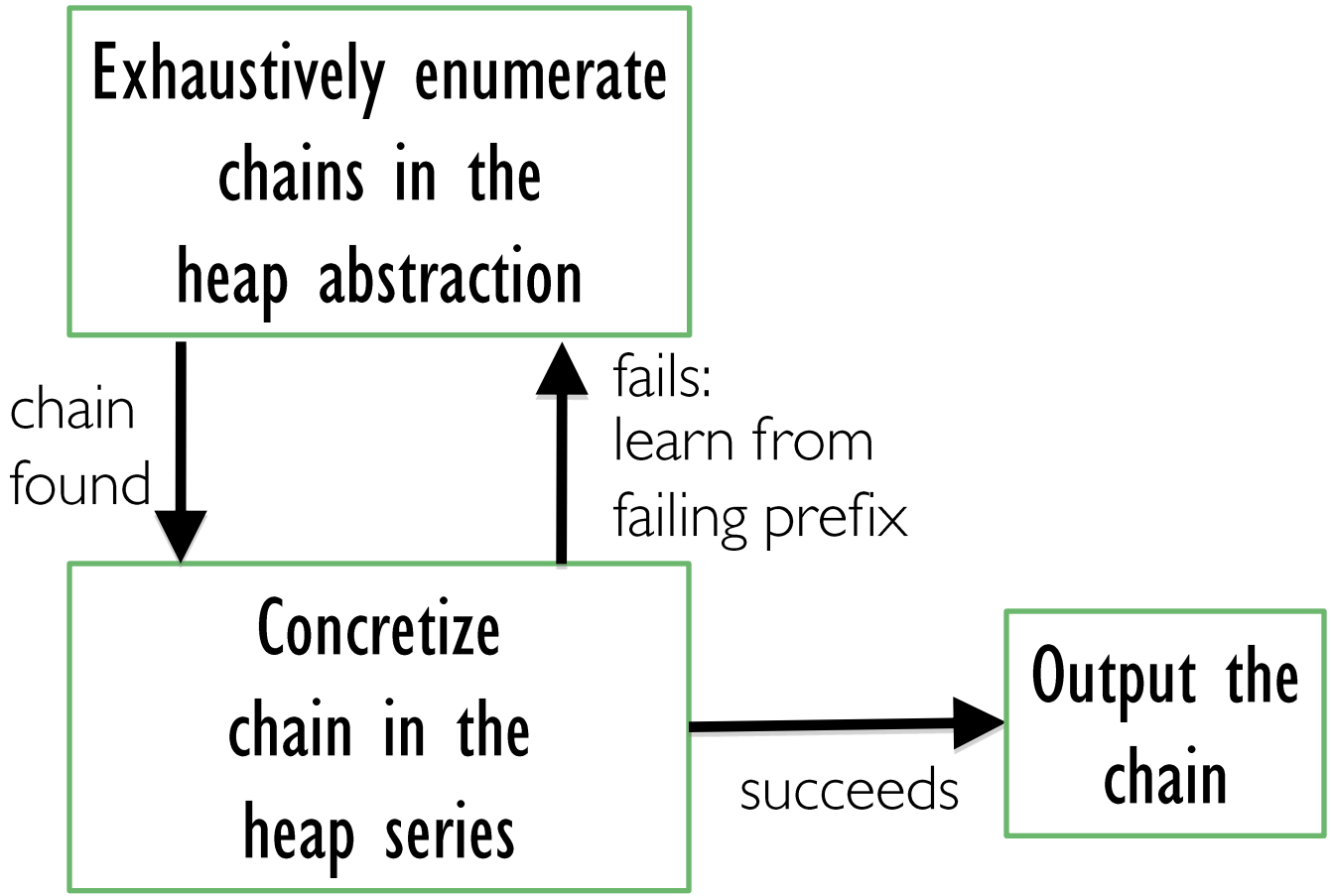
Chain lifetime: [6, 10]

Challenge: Heaps Are Large

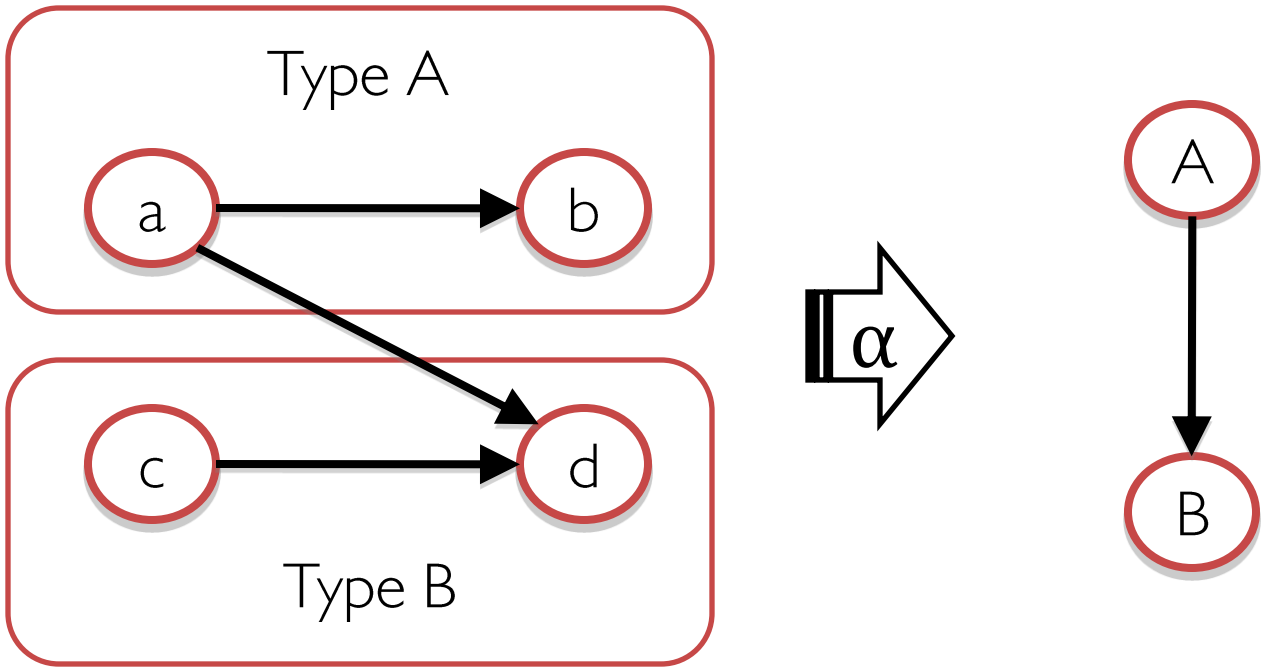
Solution: connectivity-preserving abstractions of the heap series



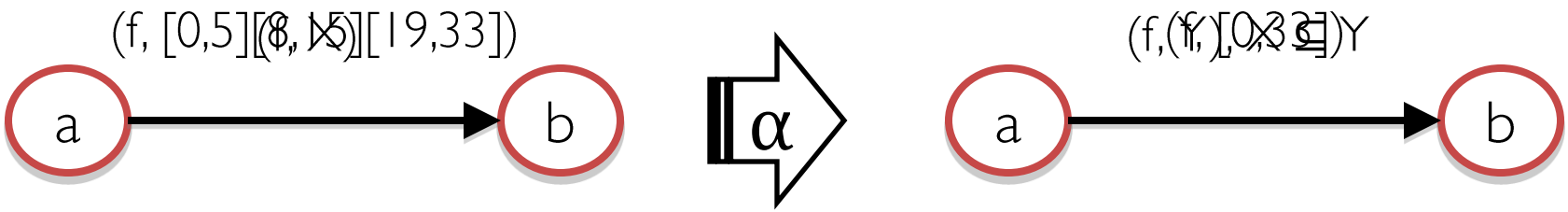
Challenge: Heaps Are Large



Type-Based Heap Abstraction



Time-Based Heap Abstraction



Projection of Thin Slices

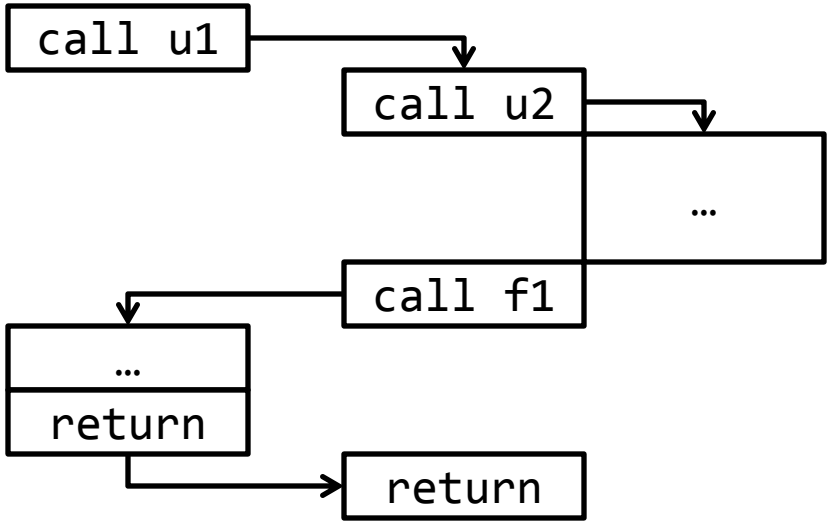
Thin slicing computes a data-dependency relation effective for program understanding.

Projection of Thin Slices

Thin slicing computes a data-dependency relation effective for program understanding.

Projection generalizes to the essential code creating the chain.

- calls from the user to the framework
- returns from the user code
- user producers of framework consum



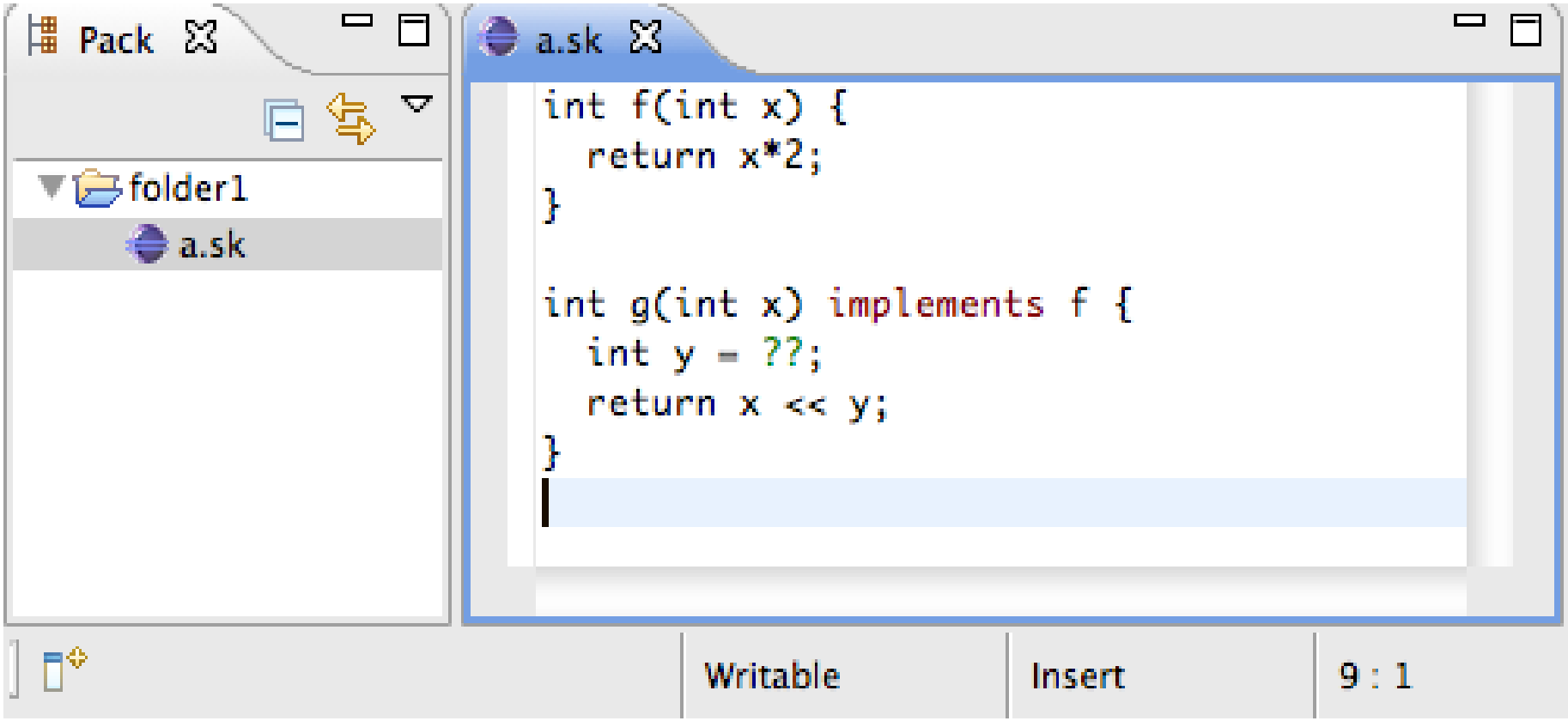
Evaluation

- User Study
  - MatchMaker generates imperfect glue code.
  - Does it still help improve productivity?

- Case Study
  - Can MatchMaker generate glue code for many class pairs other than editor-scanner?

User Study - Task

- Implement Syntax Highlighting in an editor, highlighting keywords "implements" and "??"



User Study - Task

```
TextEditor
```

```
MyEditor
{

}
```

```
RuleBasedScanner
```

```
MyScanner
{
 code for scanning "implements"

 code for scanning "??"

}
```

User Study - Task

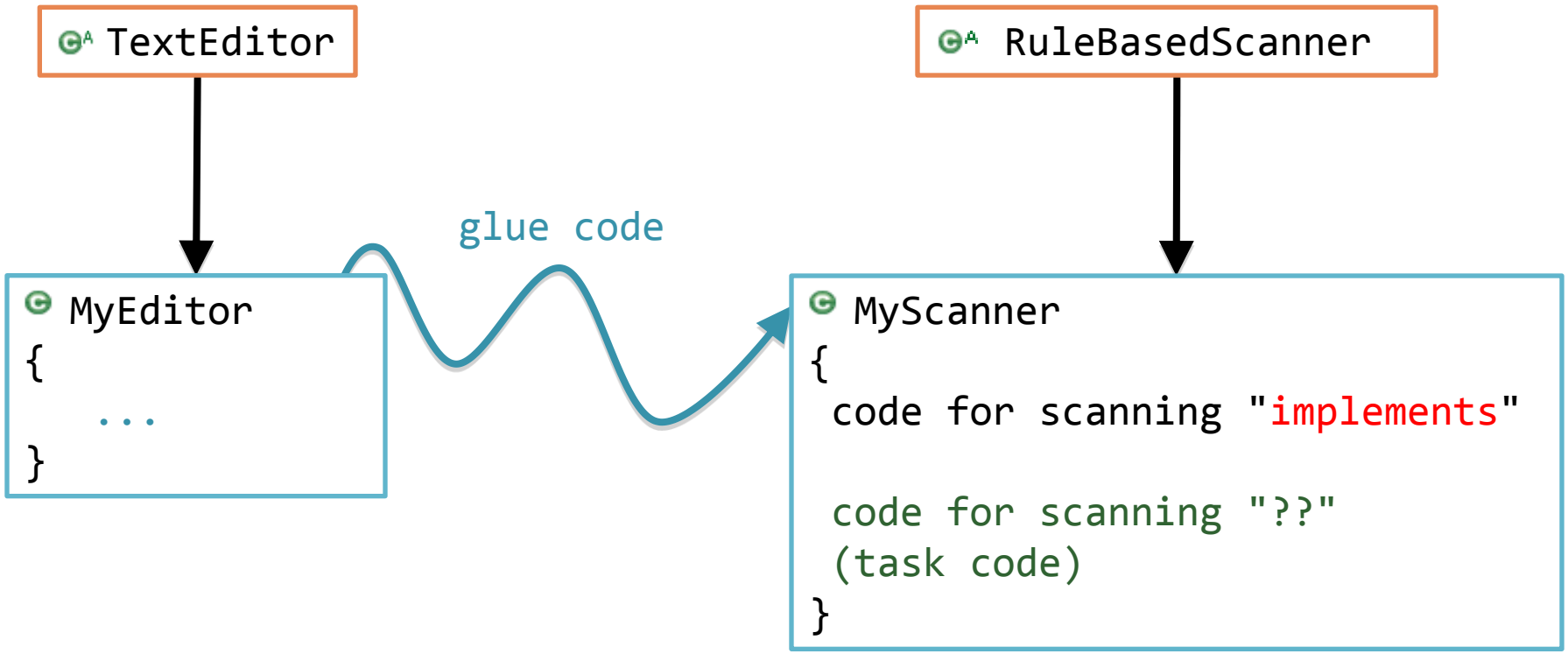
```
TextEditor
```

```
MyEditor
{

}
```

```
RuleBasedScanner
```

```
MyScanner
{
 code for scanning "implements"

 code for scanning "??"
 (task code)
}
```

User Study - Task

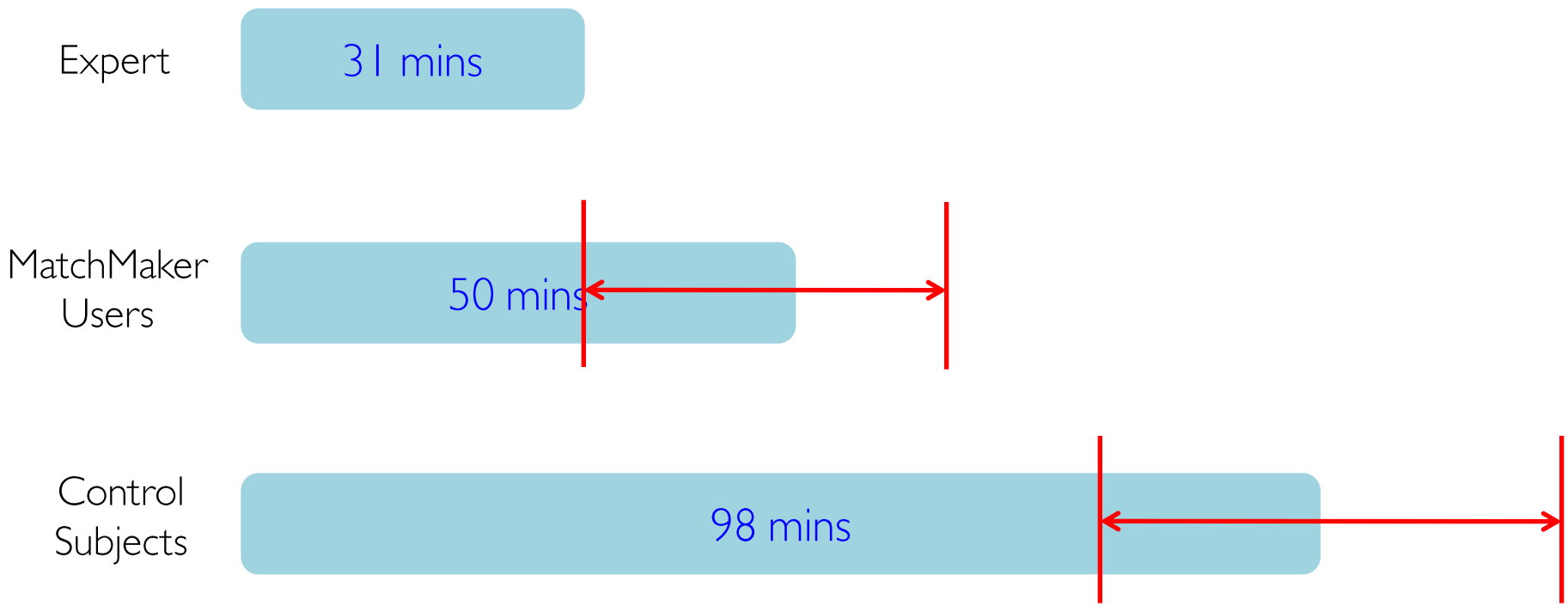


User Study - Participants

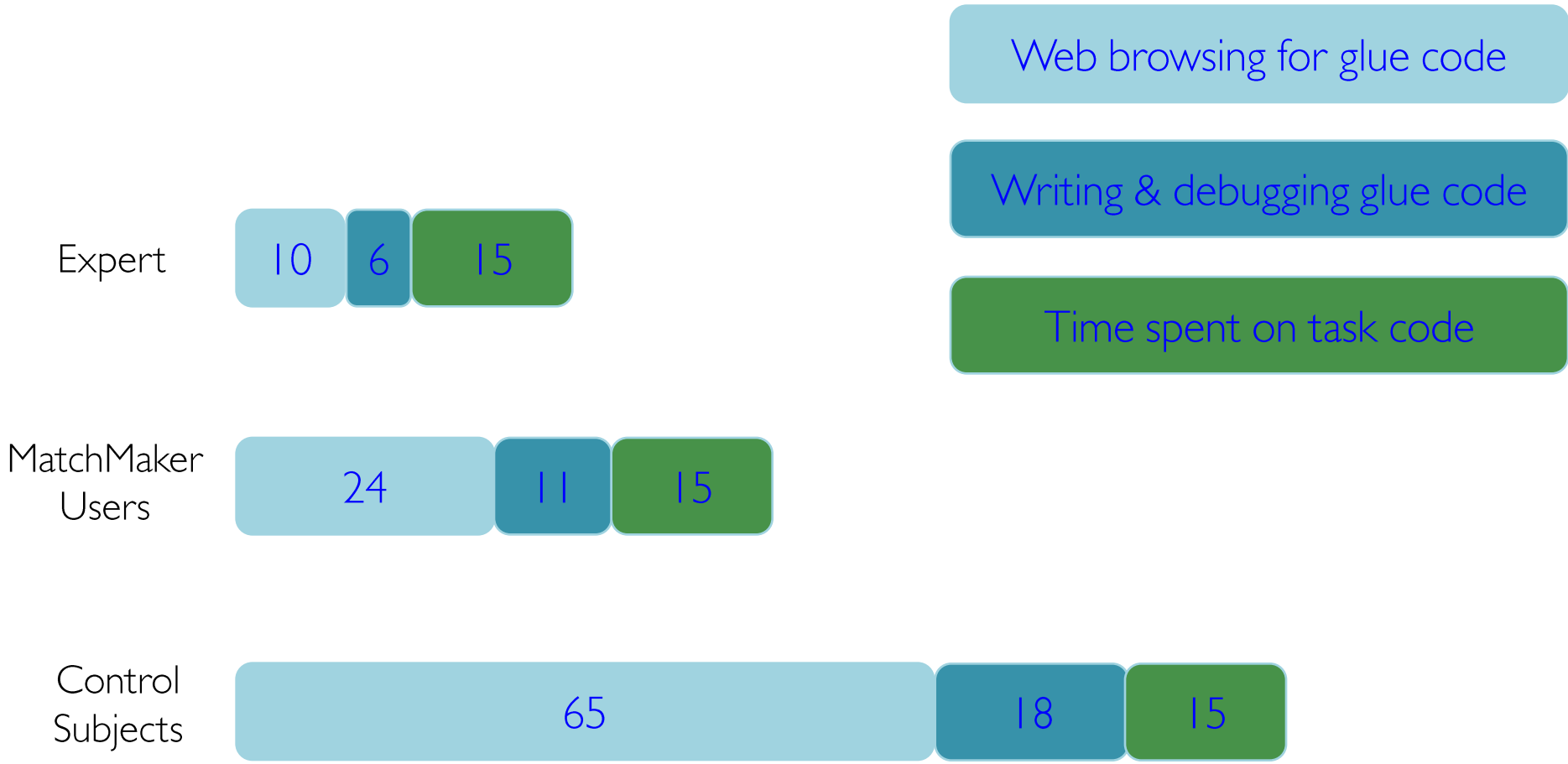
- 9 participants
  - One Eclipse framework expert
    - Years of experience writing Eclipse plug-ins
    - No experience implementing syntax highlighting
  - 8 novice users
    - 4 control subjects
    - 4 MatchMaker users
    - All of them are experienced Java programmers

User Study - Result

- Average time to complete the whole task:



User Study - Result



User Study - Observation

- Documentation on the web is not enough
  - JavaDoc: class-by-class, not helpful for interactions
  - Tutorials: lengthy, hard to get the essential piece

- MatchMaker makes browsing more efficient
  - Generated glue code depicts relations among classes
  - Only use web to find holes / missing statements
  - Users have correct class/method names as keyword to quickly skim the documentation

MatchMaker improves productivity

even when generated glue code is imperfect

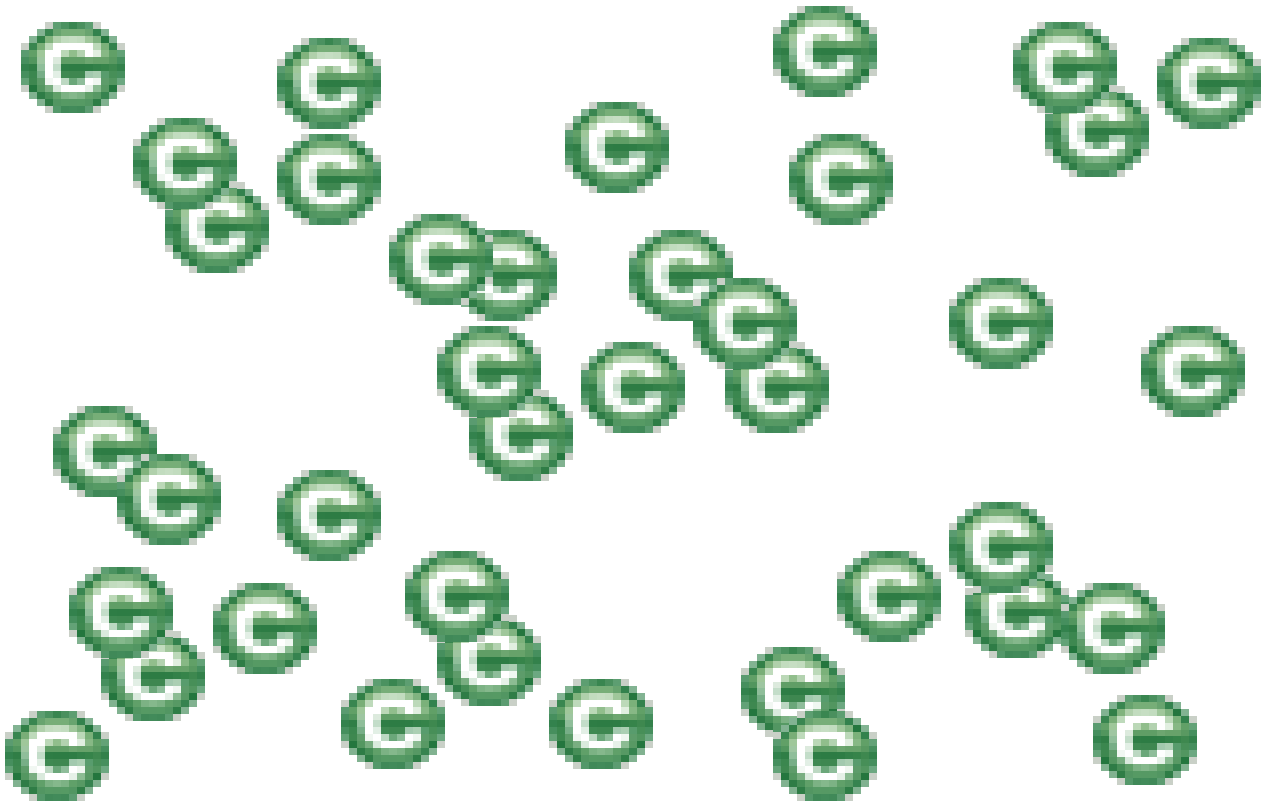
Case Study

Can MatchMaker do more than connect editors and scanners?

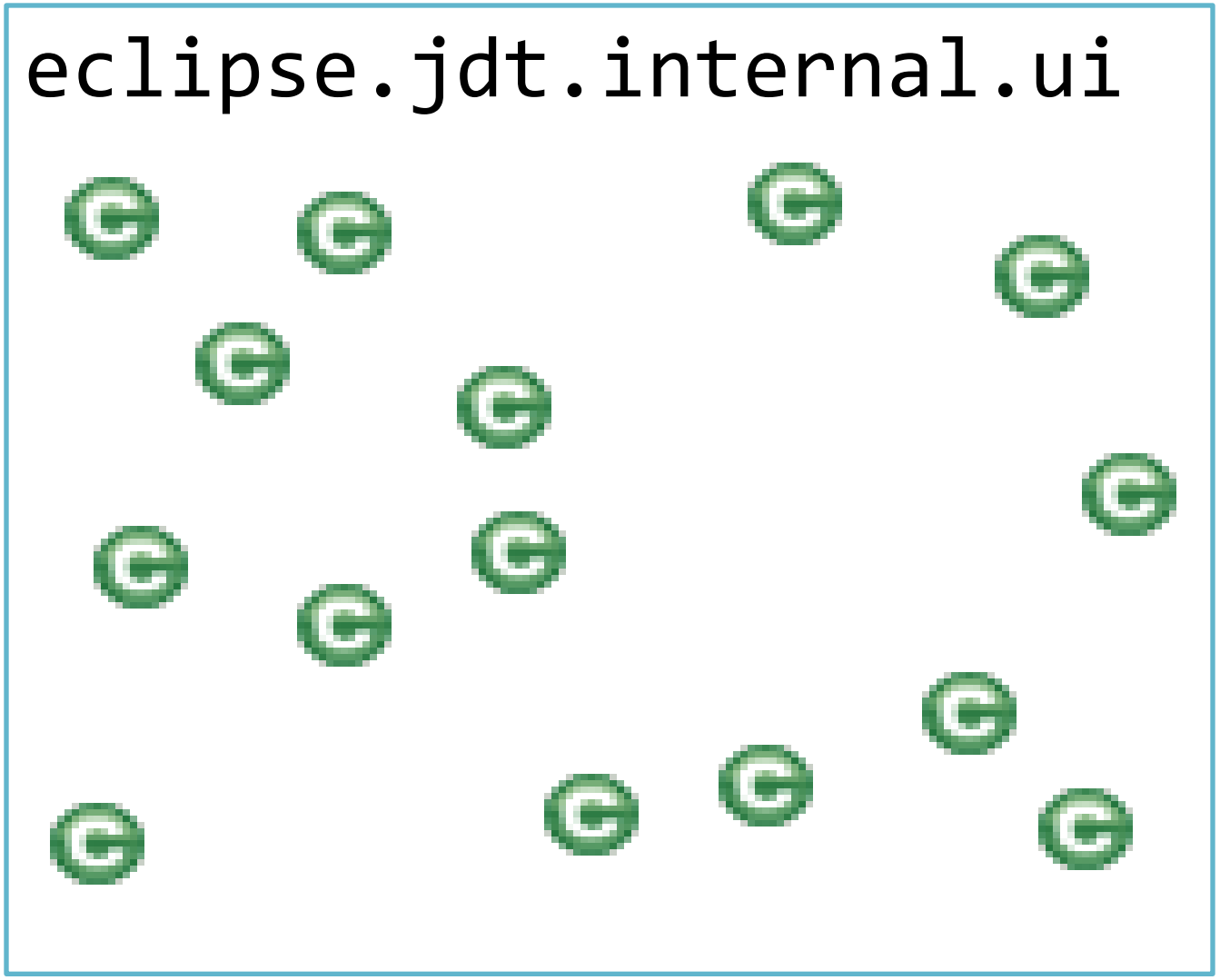
Can MatchMaker generate glue code for other interesting class pairs?

Case Study

eclipse.jdt.internal.ui



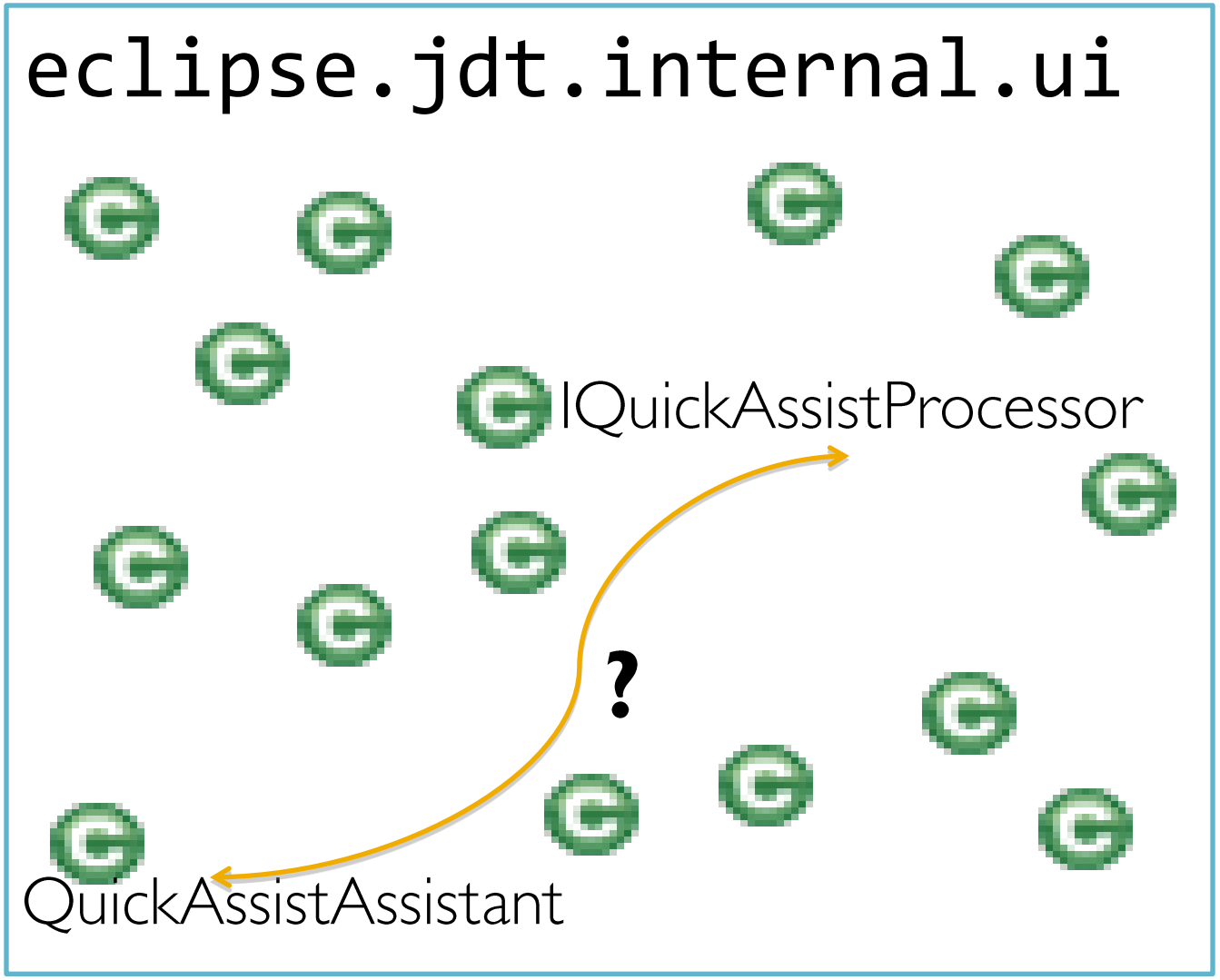
Case Study



grep

200+ classes extend the framework

Case Study



16 interesting class pairs

Case Study - Result

No.	Class pair	# holes	# missing statements
1	QuickAssistAssistant-IQuickAssistProcessor	0	0
2	ITextEditor-ITextHover	0	0
3	ITextEditor-TextFileDocumentProvider	0	0
4	ITextEditor-QuickAssistAssistant	1	0
5	ITextEditor-SelectionHistory	1	0
6	ITextEditor-IAutoEditStrategy	0	1
7	ISpellCheckEngine-ISpellChecker	0	1
8	ITextEditor-IContentOutlinePage	0	1
9	MonoReconciler-IReconcilingStrategy	1	1
10	IContentAssistProcessor-ContentAssistInvocationContext	2	0
11	ITextEditor-ContextBasedFormattingStrategy	2	0
12	ITextEditor-SemanticHighlighting	0	2

Case Study - Result

MatchMaker generates reasonable glue code for 12 /16 class pairs we have tested.

even the program execution database is quite limited.

Conclusion

Software today becomes so large and dynamic
Seeing the code is not enough to learn about it

Data-driven approach: the power of reasoning about dynamic behaviors of programs **in real time**

Opens a new gate for programming tools

Thank you!

Try MatchMaker at

`sketch1.csail.mit.edu/matchmaker`

still a prototype, not mature

the program execution database is very limited