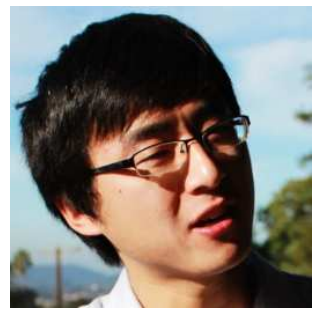
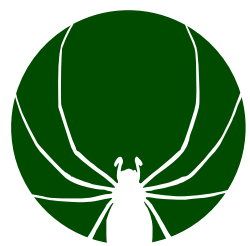
Improving Efficiency of Dynamic Analysis with Dynamic Dependence Summaries


Vijay Krishna Palepu Guoqing Xu James A. Jones

University of California, Irvine, USA

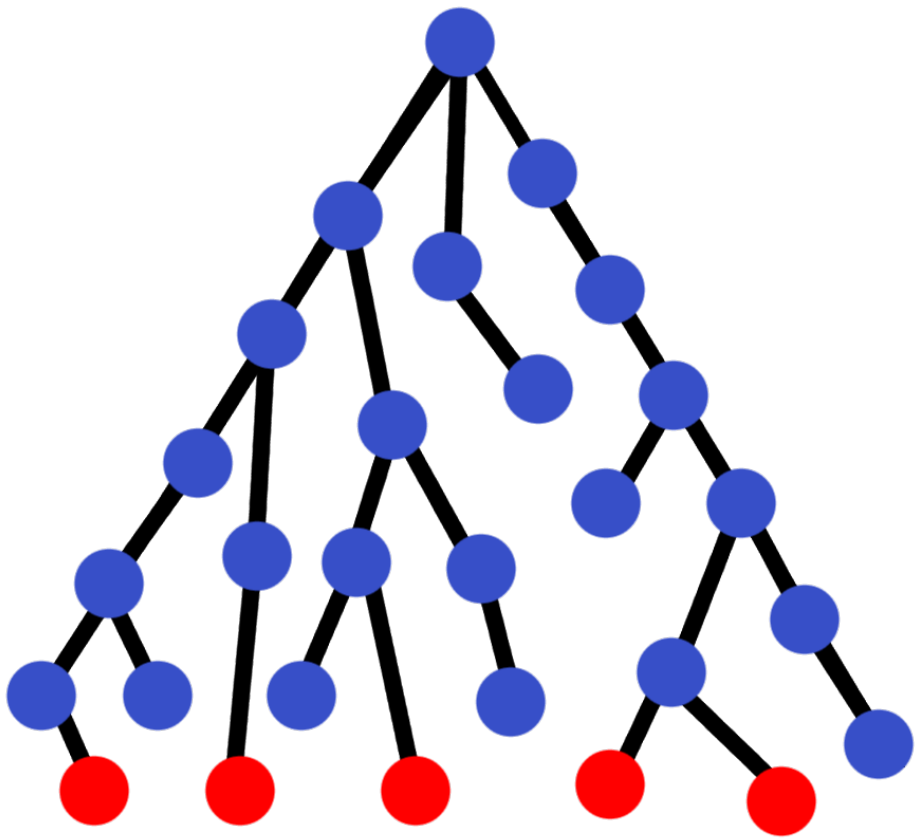
28th IEEE/ACM International Conference on Automated Software Engineering, 2013



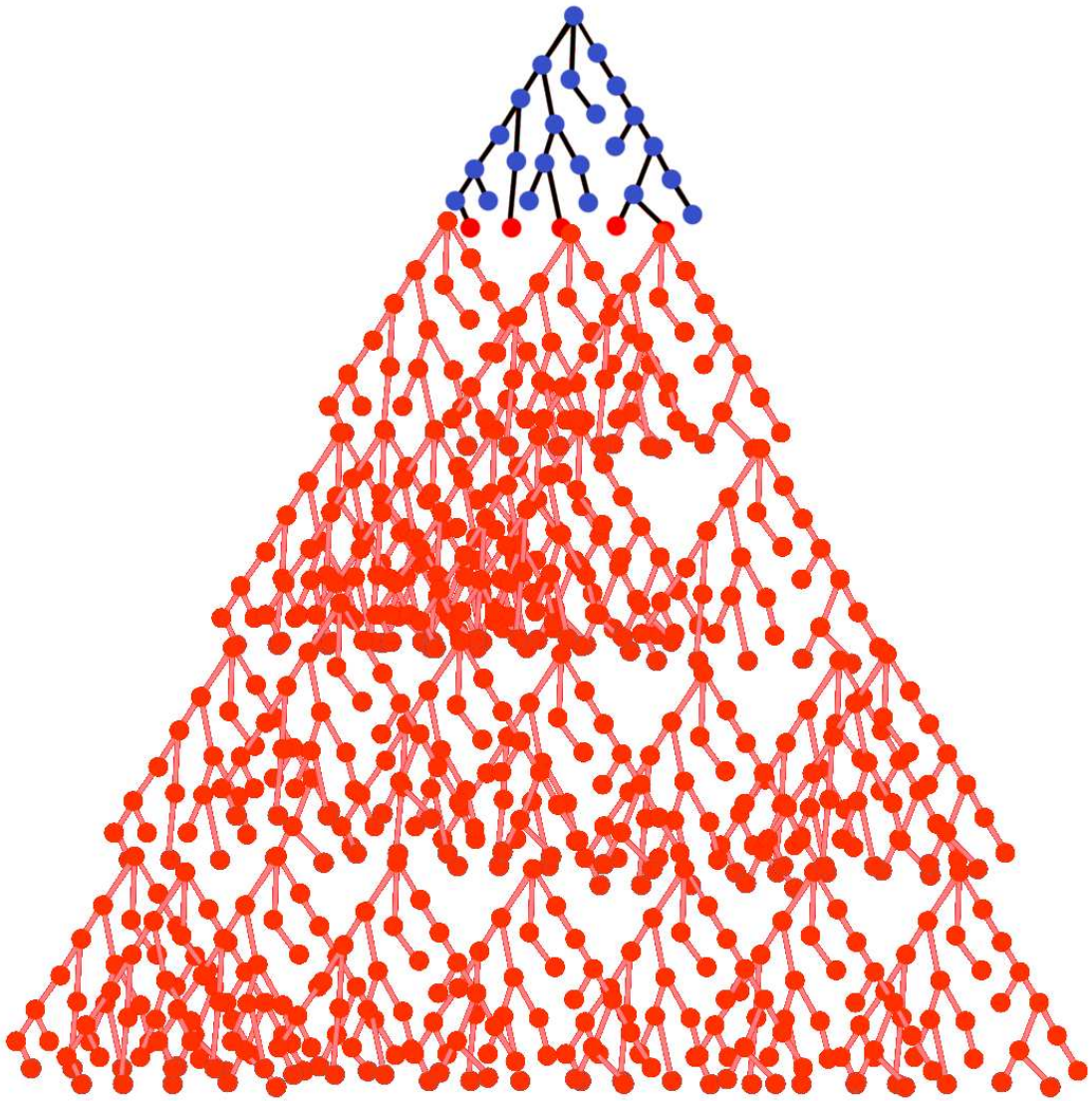
The Hut and The Mountain

Image Credit: "Eagle's Eye KHMR", Doug Zwick@Flickr

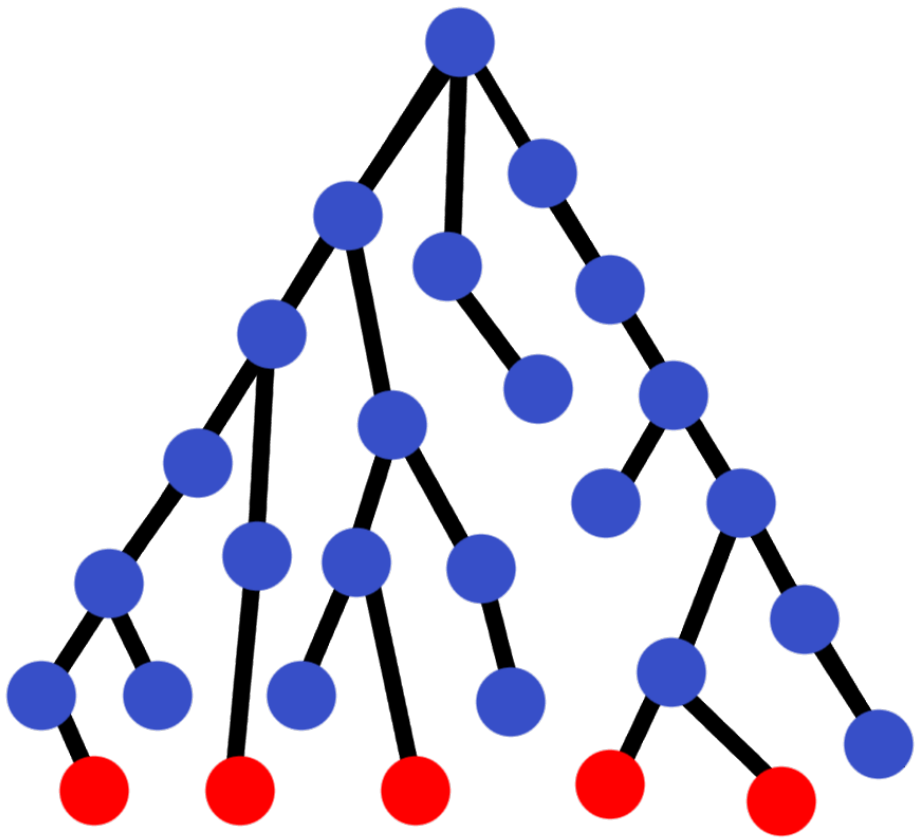
The Hut and The Mountain



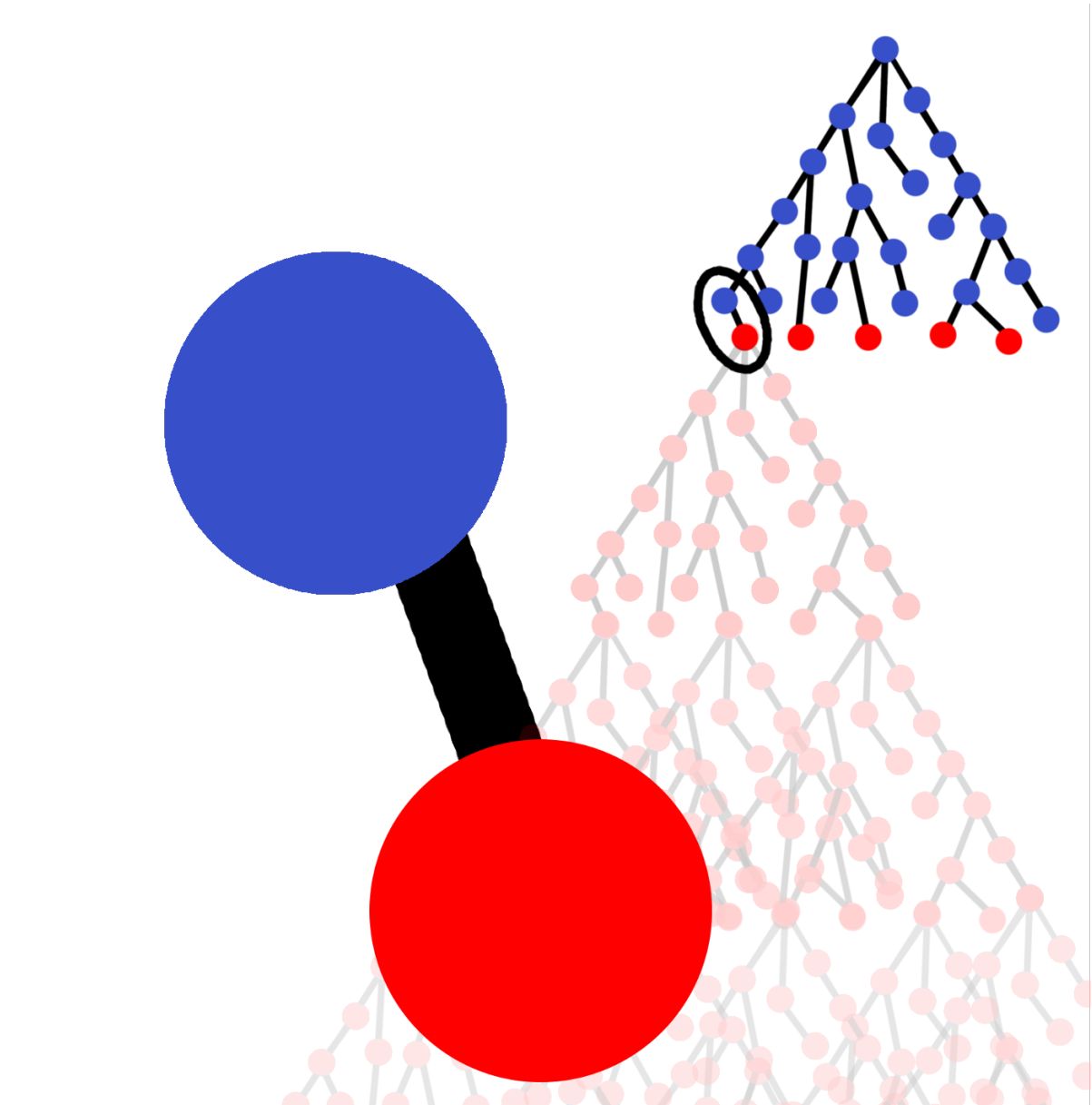
The Hut and The Mountain



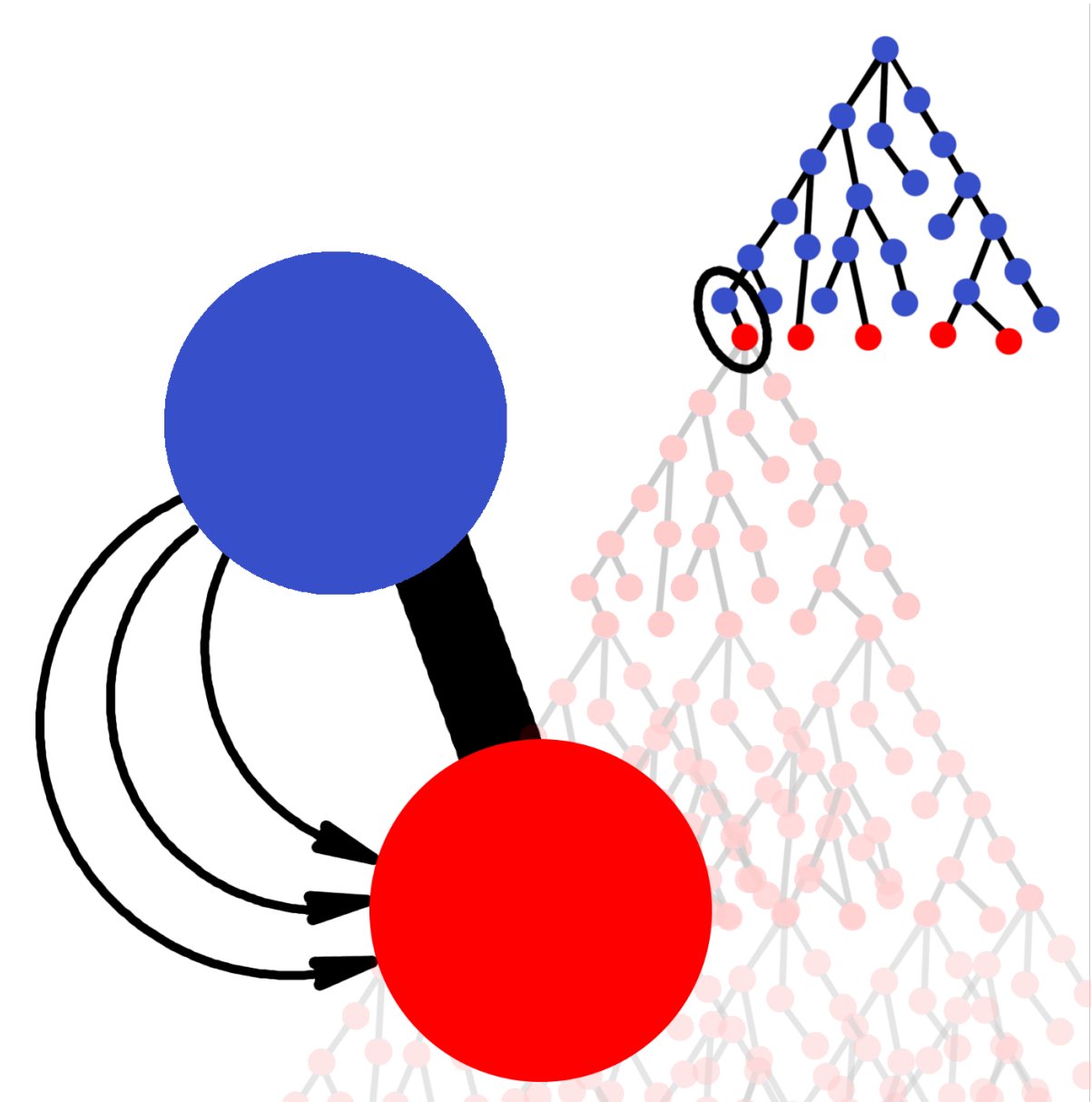
The Hut and The Mountain



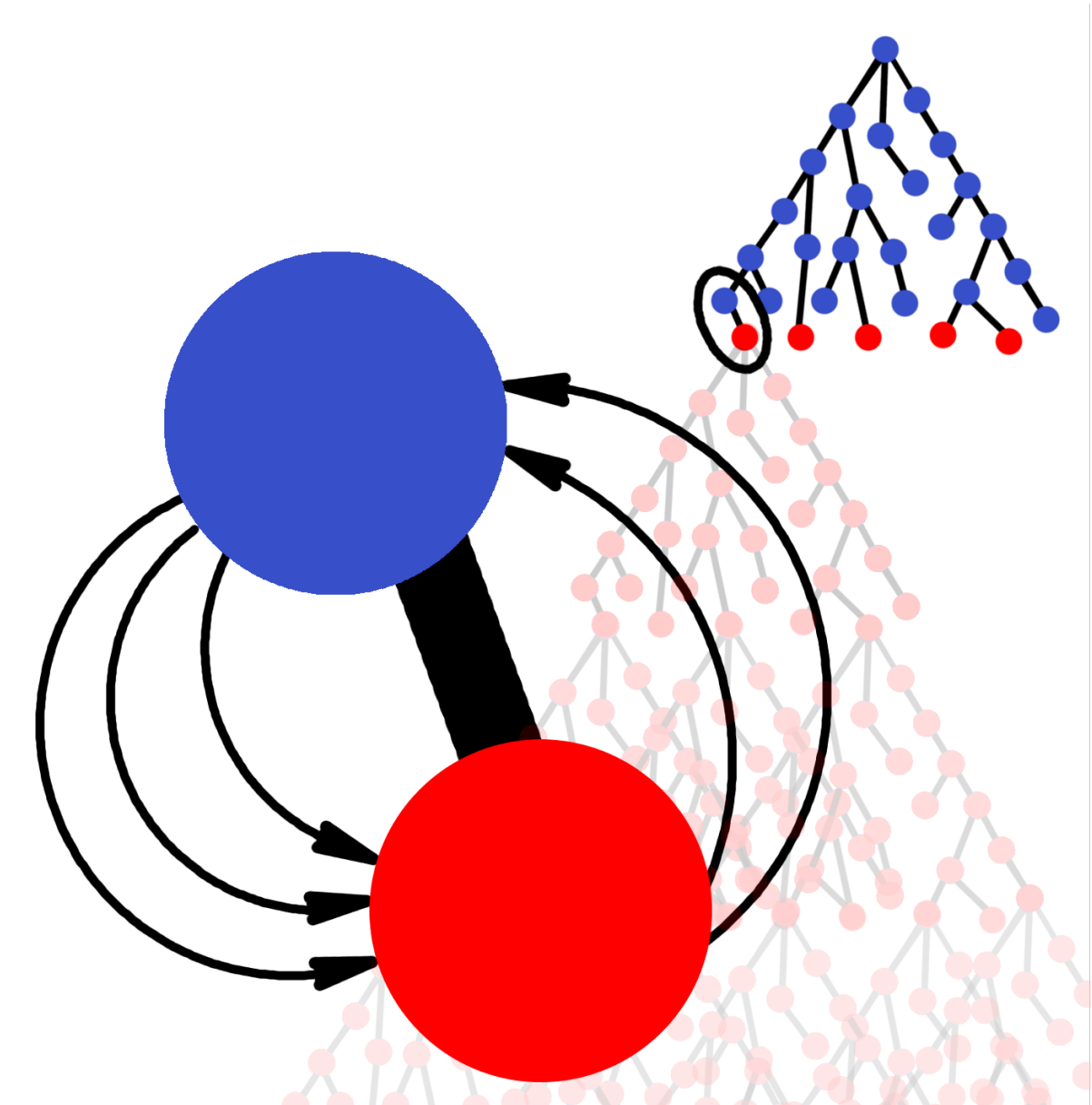
Method Summaries



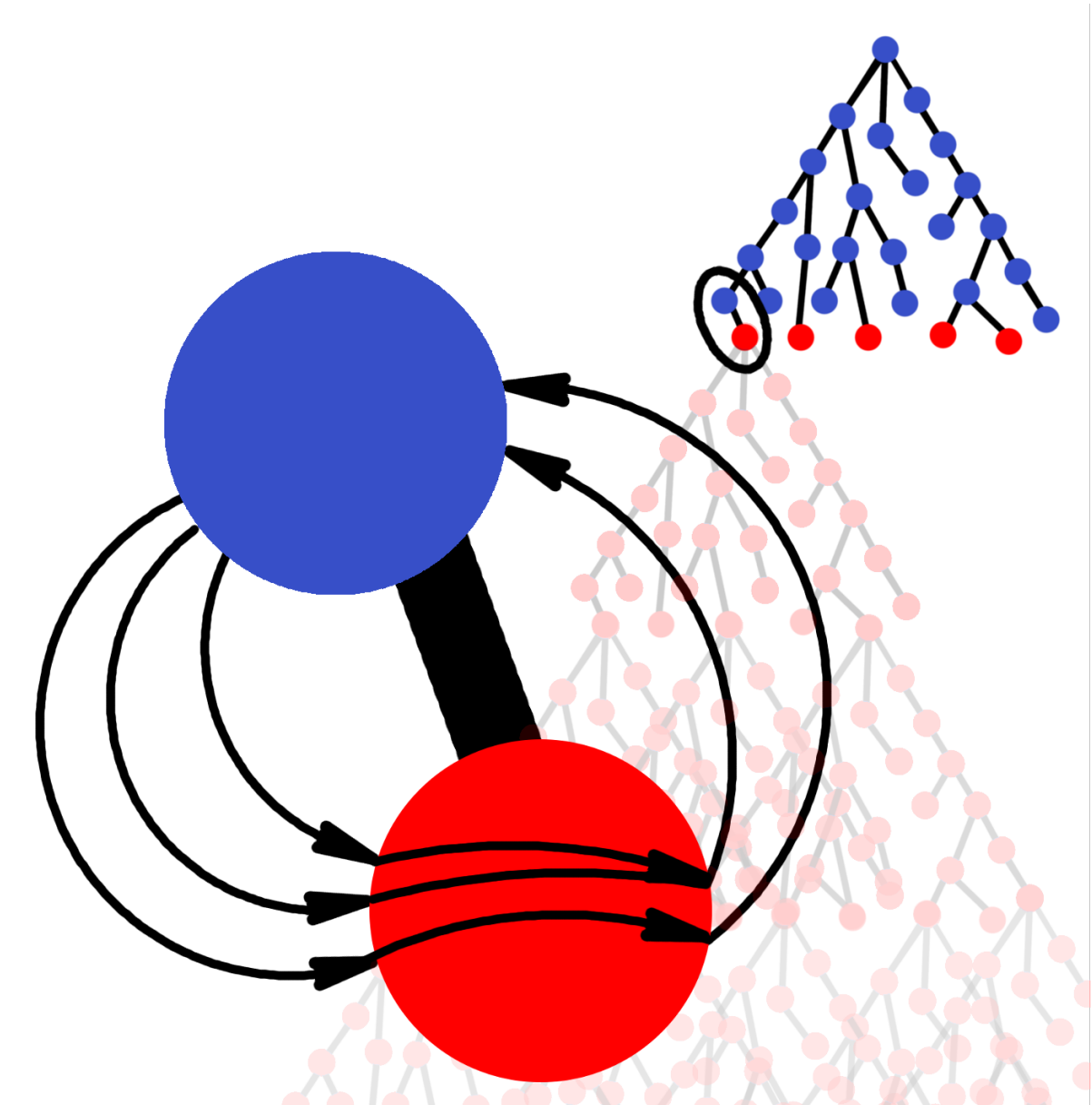
Method Summaries



Method Summaries



Method Summaries



Example.

```
void main() {
    IntList k =  new IntList();
    int num = 1;
    k.add(num);
    ...
    ...
    ...
}
```

Example.

```
void main() {
    IntList k =  new IntList();
    int num = 1;
    k.add(num);
    ...
    ...
    ...
}
```

```
void add(int i) {
        int t = this.size;
        int[] a = this.arr;
        a[t] = i;
        t = t + 1;
        this.size = t;
}
```

Example.

```
void main() {
    IntList k =  new IntList();
    int num = 1;
    k.add(num);
    ...
    ...
    ...
}
```

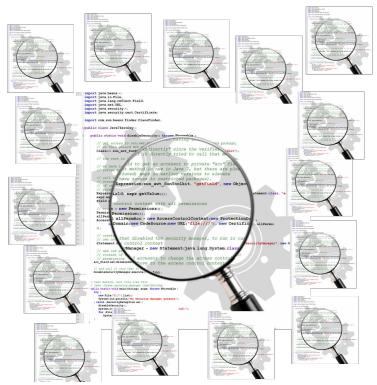
```
void main() {
    IntList k =  new IntList();
    int num = 1;
    k.arr[k.size] = num;
    k.size = k.size + 1;
    ...
    ...
    ...
}
```

Background.

- Sharir and Pnueli. **Two Approaches to Interprocedural Data Flow Analysis.** Program Flow Analysis: Theory and Applications, 1981.
- Horwitz, Reps and Binkley. **Interprocedural Slicing using Dependence Graphs.** TOPLAS, 1990.
- Rountev, Sharp and Xu. **IDE Dataflow Analysis in the Presence of Large Object-Oriented Libraries.** CC, 2008.
- Yorsh, Yahav and Chandra. **Generating Precise and Concise Procedure Summaries.** POPL, 2008.
- Xu, Rountev and Sridharan. **Scaling CFL-reachability-based Points-to Analysis using Context-sensitive Must-not-alias Analysis.** ECOOP, 2009.

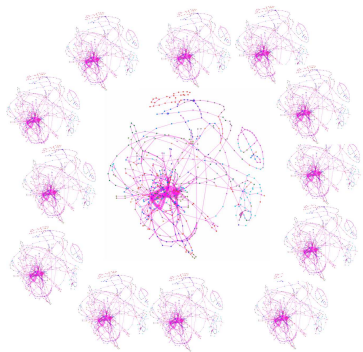
Typical Dynamic Analysis

Instrument Program & Libraries.

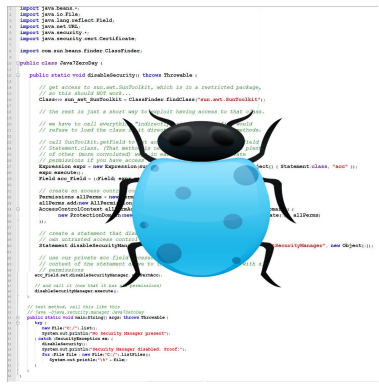


Monitor Execution (Program & Libraries).

After Some Waiting



Extract Dataflow (Program & Libraries).



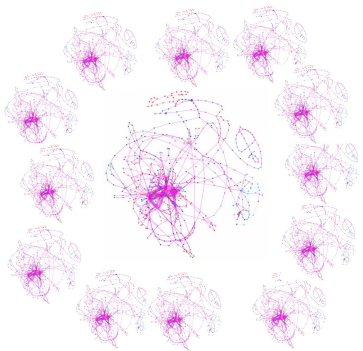
Client Analysis. *e.g.*, Find Bug!

Approach Overview.

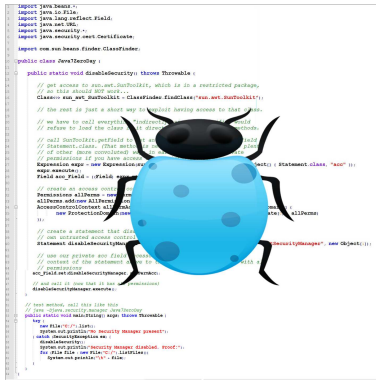
Instrument Program & Libraries.

Monitor Execution (Program & Libraries).

After Some Waiting



Extract Dataflow (Program & Libraries).



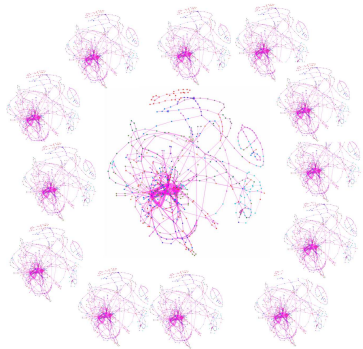
Client Analysis. *e.g.*, Find Bug!

Approach Overview.

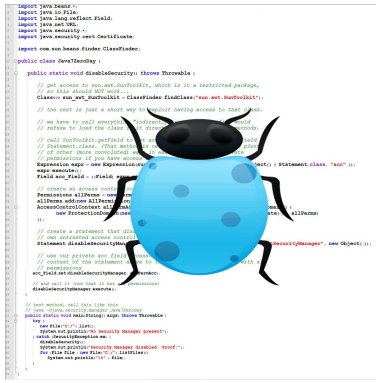
Instrument Program & Libraries.

Monitor Execution (Program & Libraries).

After Some Waiting



Extract Dataflow (Program & Libraries).



Client Analysis. *e.g.*, Find Bug!

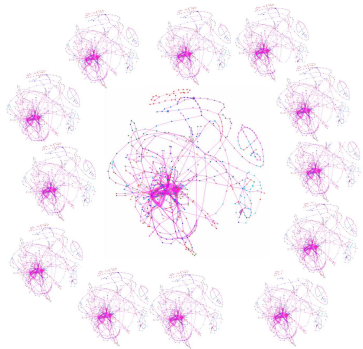
Library Method's Dataflows.

Approach Overview.

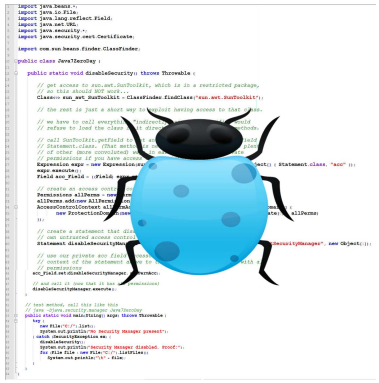
Instrument Program & Libraries.

Monitor Execution (Program & Libraries).

After Some Waiting

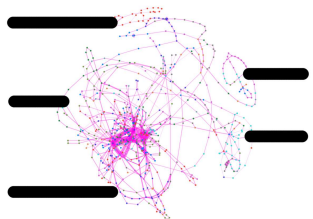


Extract Dataflow (Program & Libraries).



Client Analysis. *e.g.*, Find Bug!

Library Method's Dataflows.



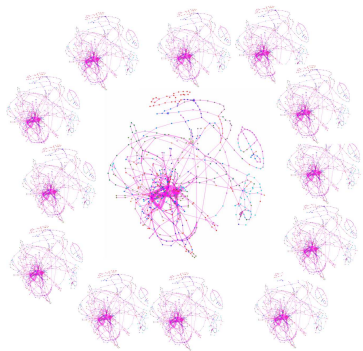
Recognize Method Inputs/Outputs.

Approach Overview.

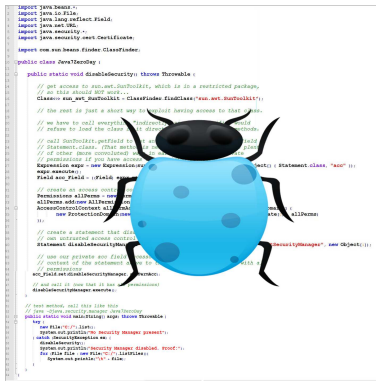
Instrument Program & Libraries.

Monitor Execution (Program & Libraries).

After Some Waiting

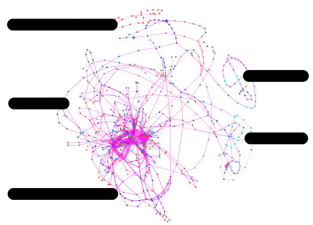


Extract Dataflow (Program & Libraries).

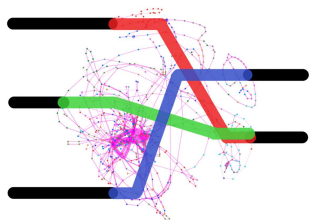


Client Analysis. *e.g.*, Find Bug!

Library Method's Dataflows.



Recognize Method Inputs/Outputs.



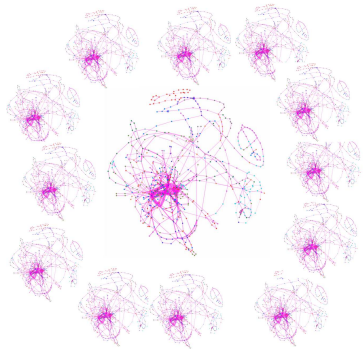
Summarize Dataflows.

Approach Overview.

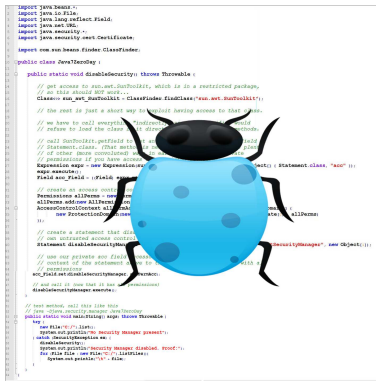
Instrument Program & Libraries.

Monitor Execution (Program & Libraries).

After Some Waiting

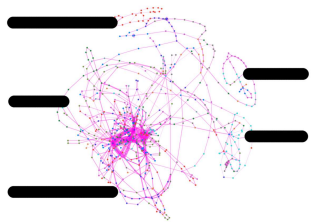


Extract Dataflow (Program & Libraries).

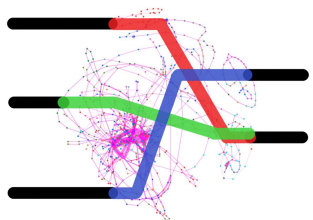


Client Analysis. *e.g.*, Find Bug!

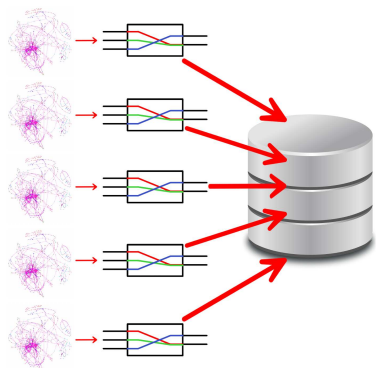
Library Method's Dataflows.



Recognize Method Inputs/Outputs.



Summarize Dataflows.

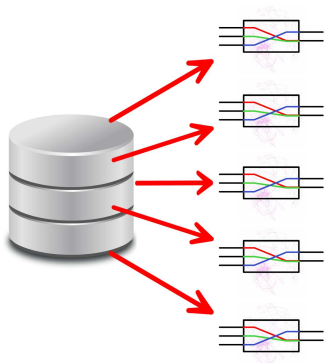


Abstract & Store.

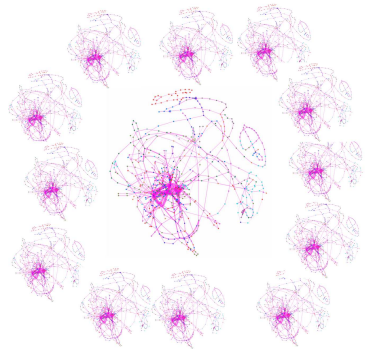
Approach Overview.

Instrument Program.

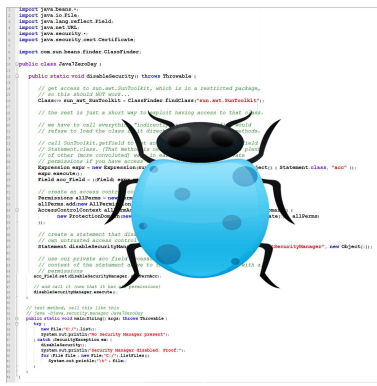
Monitor Execution (Program).



Reuse Stored Library Method Summaries.



Extract Dataflow (Program & Libraries).

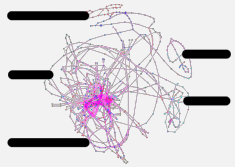


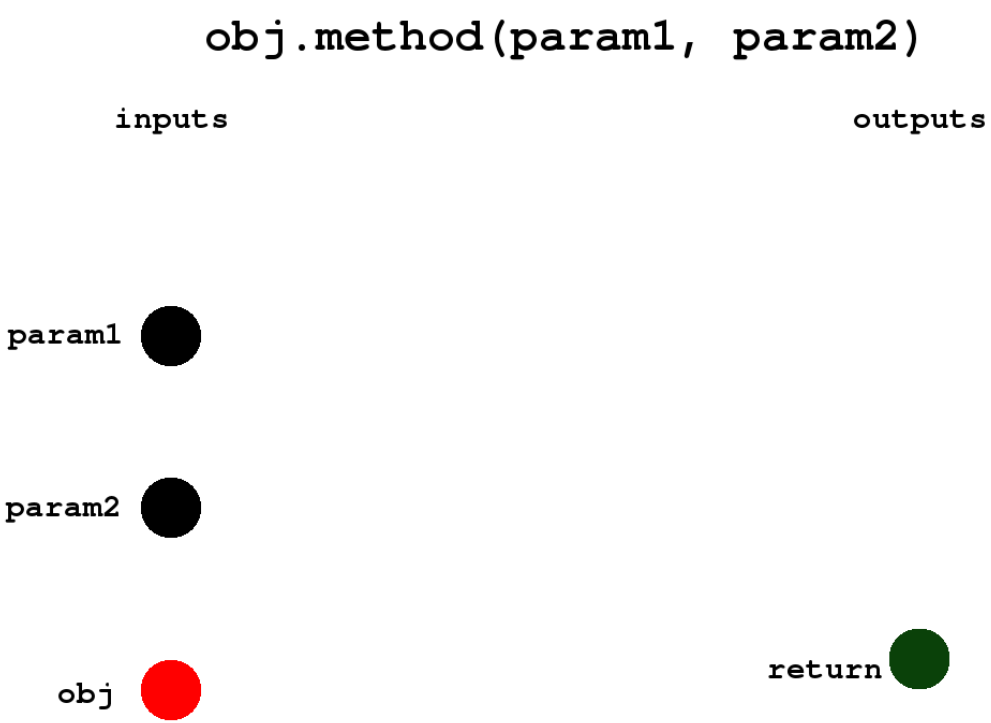
Client Analysis. *e.g.*, Find Bug!

Dynamic Dependence Summaries.

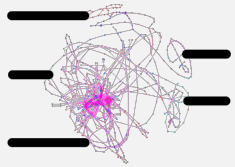
- Summarize
- Abstract
- Reuse

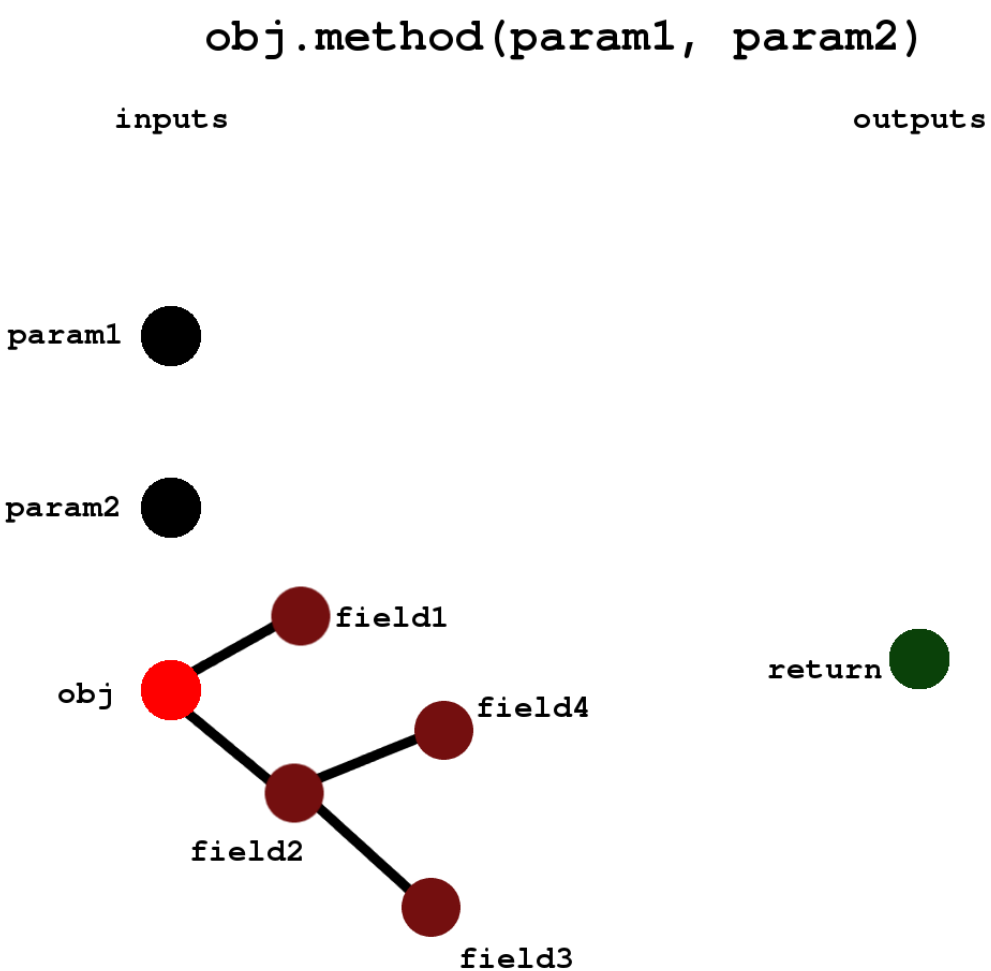
Summarize.

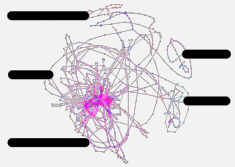



obj.method(param1, param2)
inputs
outputs
param1
param2
obj
return

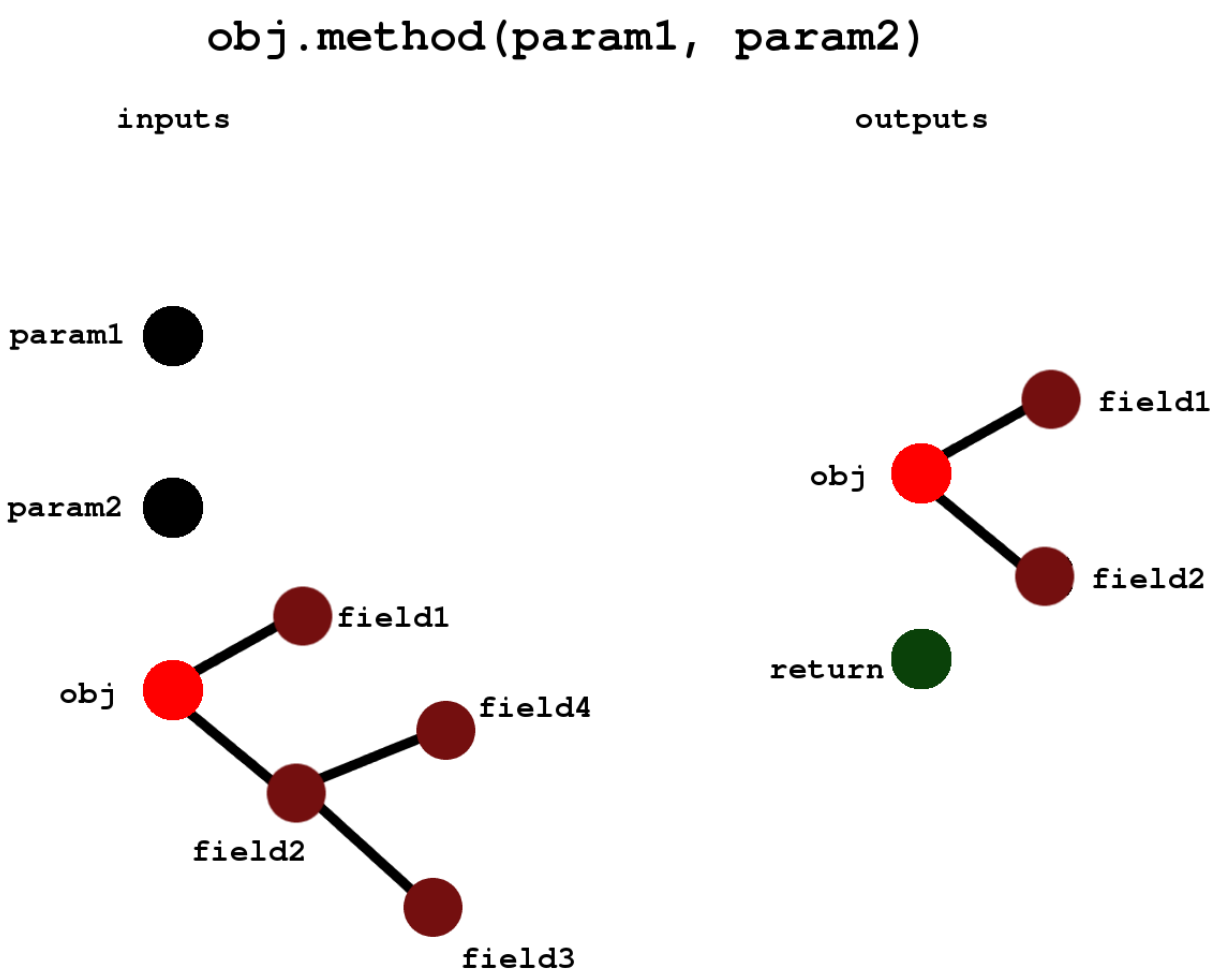
Summarize.



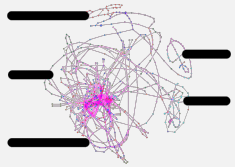

obj.method(param1, param2)
inputs
outputs
param1
param2
field1
obj
field4
field2
field3
return

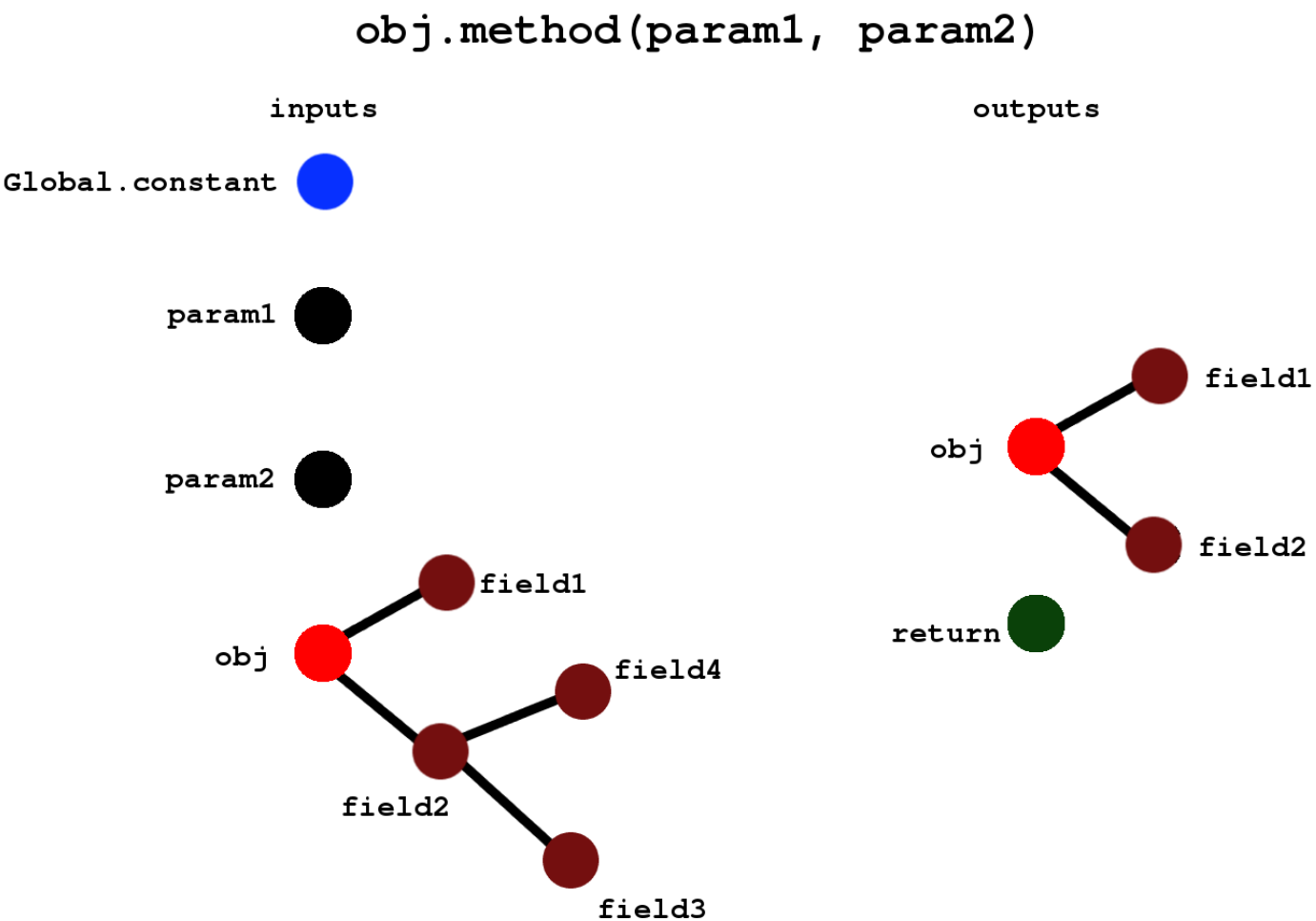


Summarize.

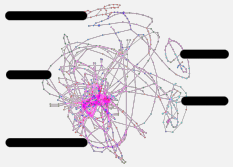

obj.method(param1, param2)
inputs
outputs
param1
param2
obj
field1
field4
field2
field3
obj
field1
field2
return

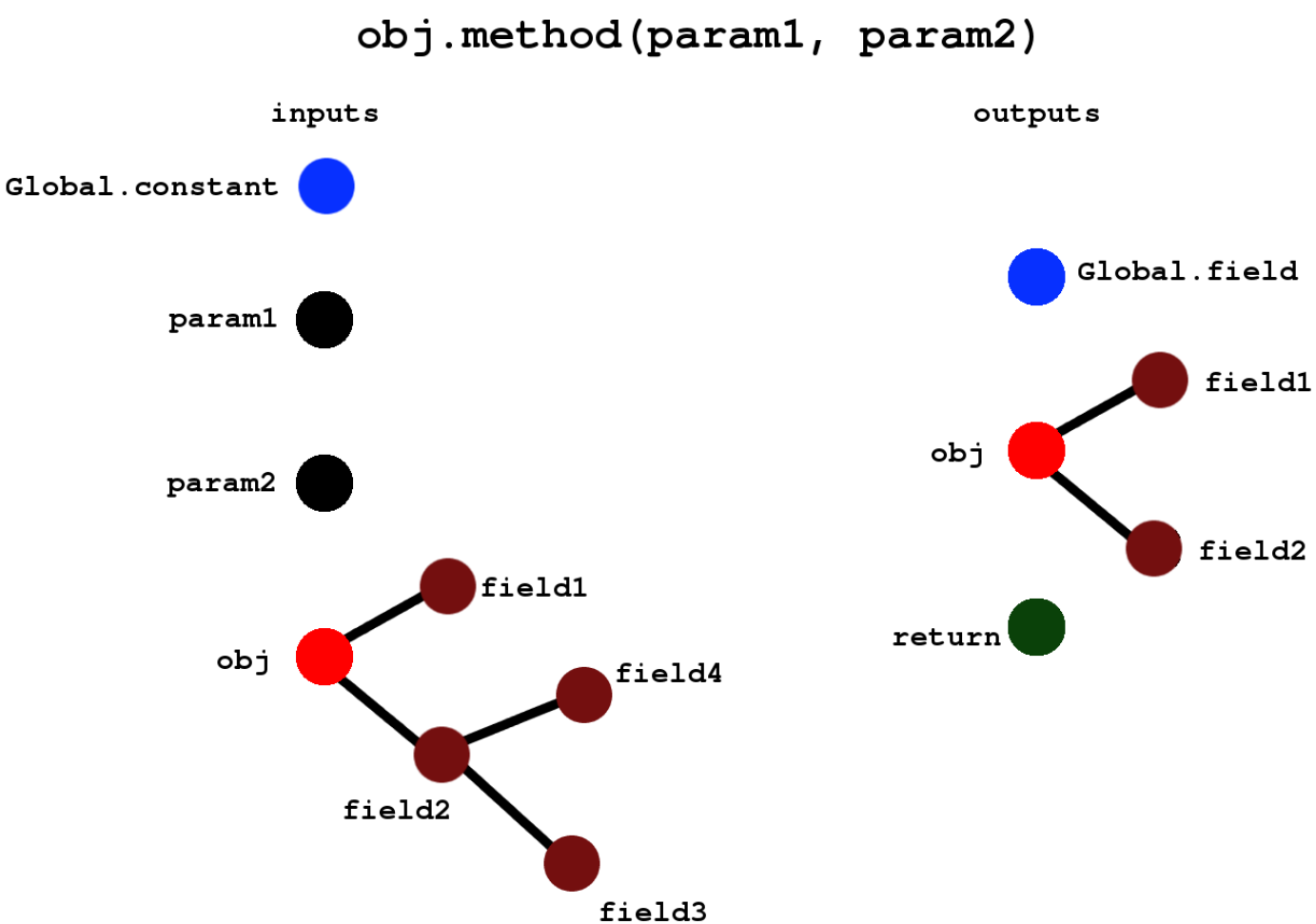
Summarize.



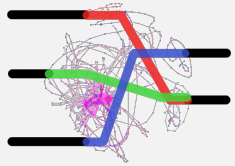

obj.method(param1, param2)
inputs
Global.constant
param1
param2
obj
field1
field4
field2
field3
outputs
obj
field1
field2
return

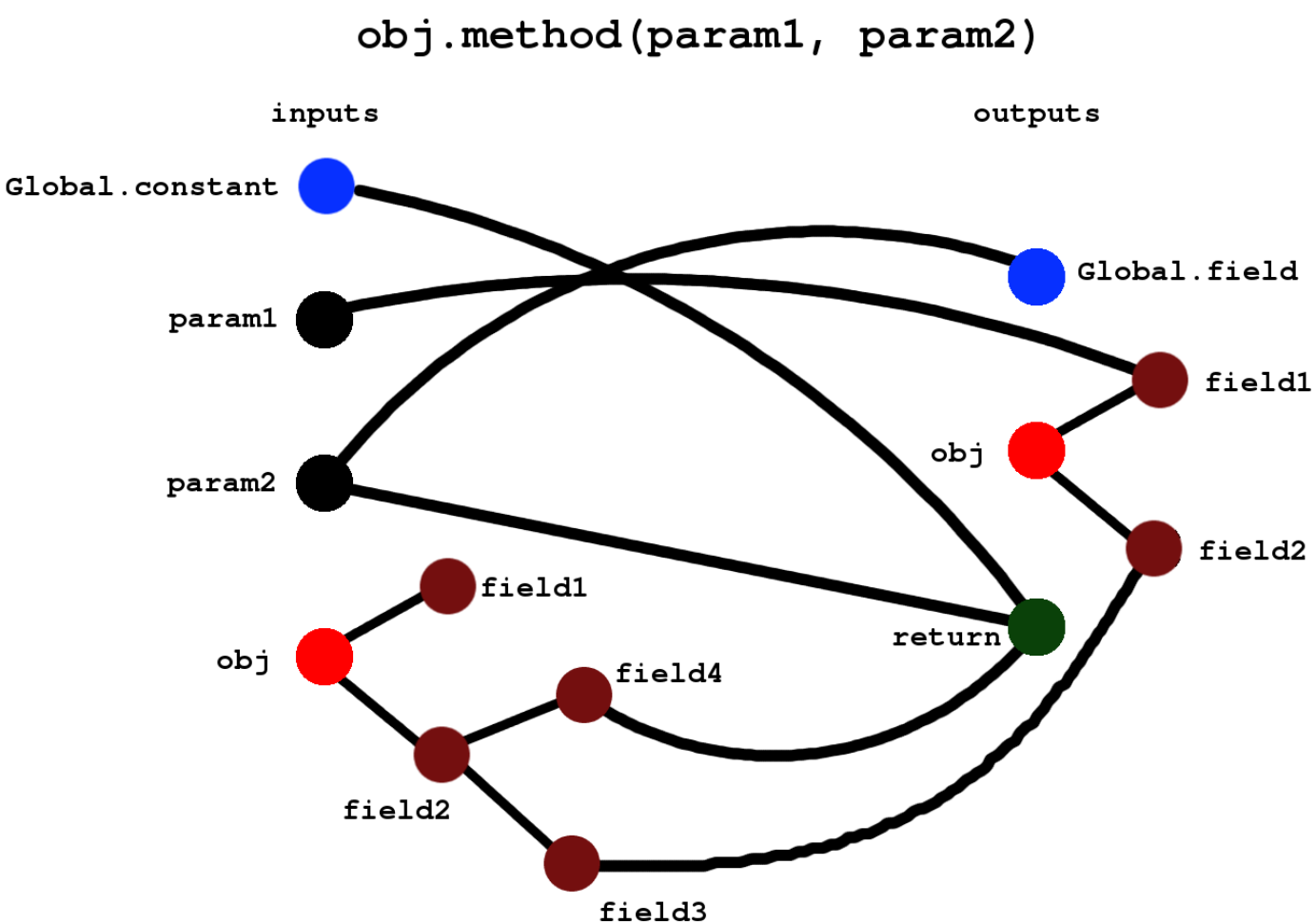
Summarize.



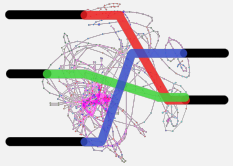

obj.method(param1, param2)
inputs
Global.constant
param1
param2
obj
field1
field4
field2
field3
outputs
Global.field
obj
field1
field2
return

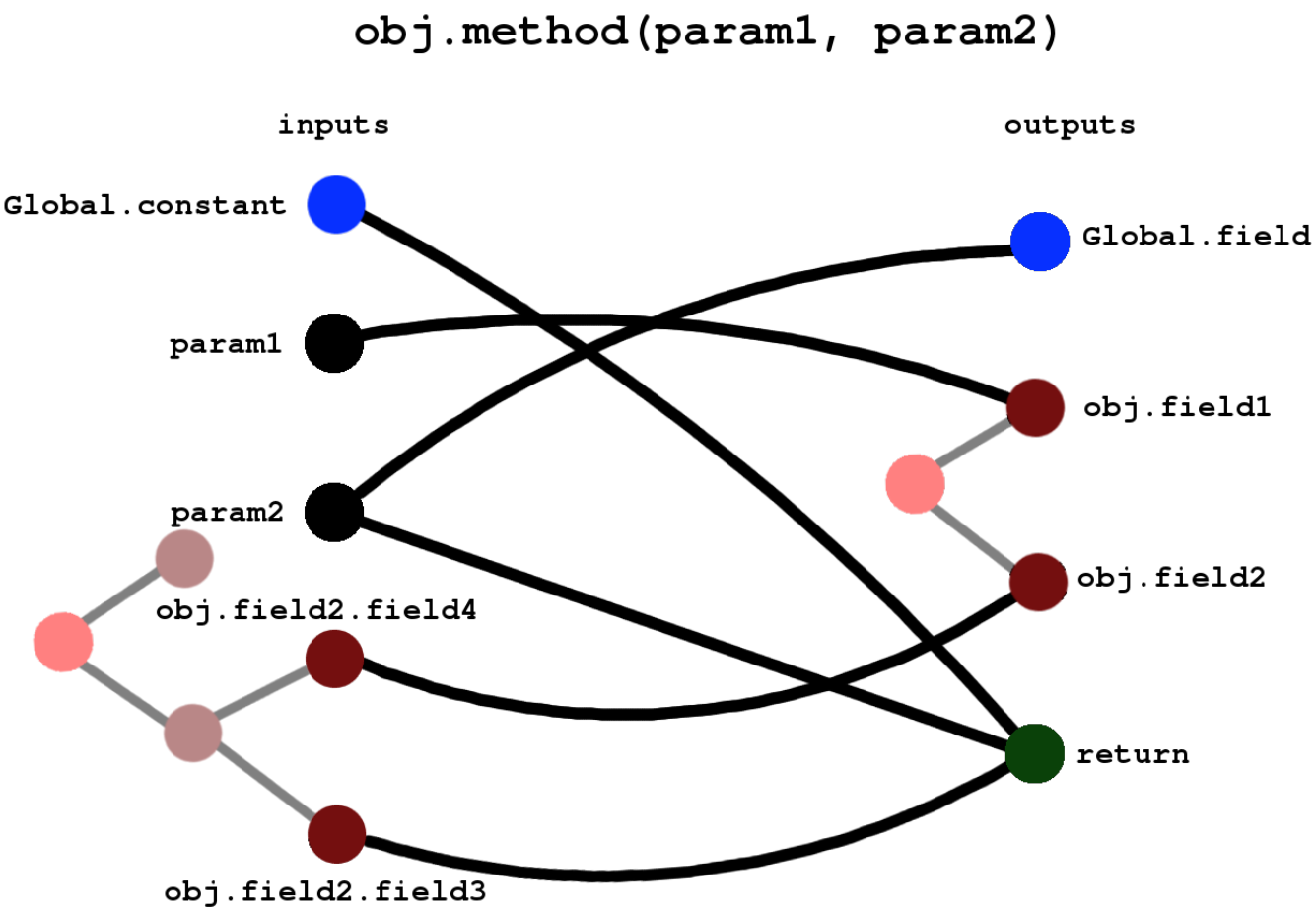
Summarize.



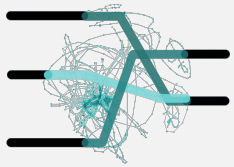

obj.method(param1, param2)
inputs
outputs
Global.constant
param1
param2
field1
obj
field4
field2
field3
Global.field
field1
obj
field2
return

Summarize.




obj.method(param1, param2)
inputs
outputs
Global.constant
param1
param2
obj.field2.field4
obj.field2.field3
Global.field
obj.field1
obj.field2
return

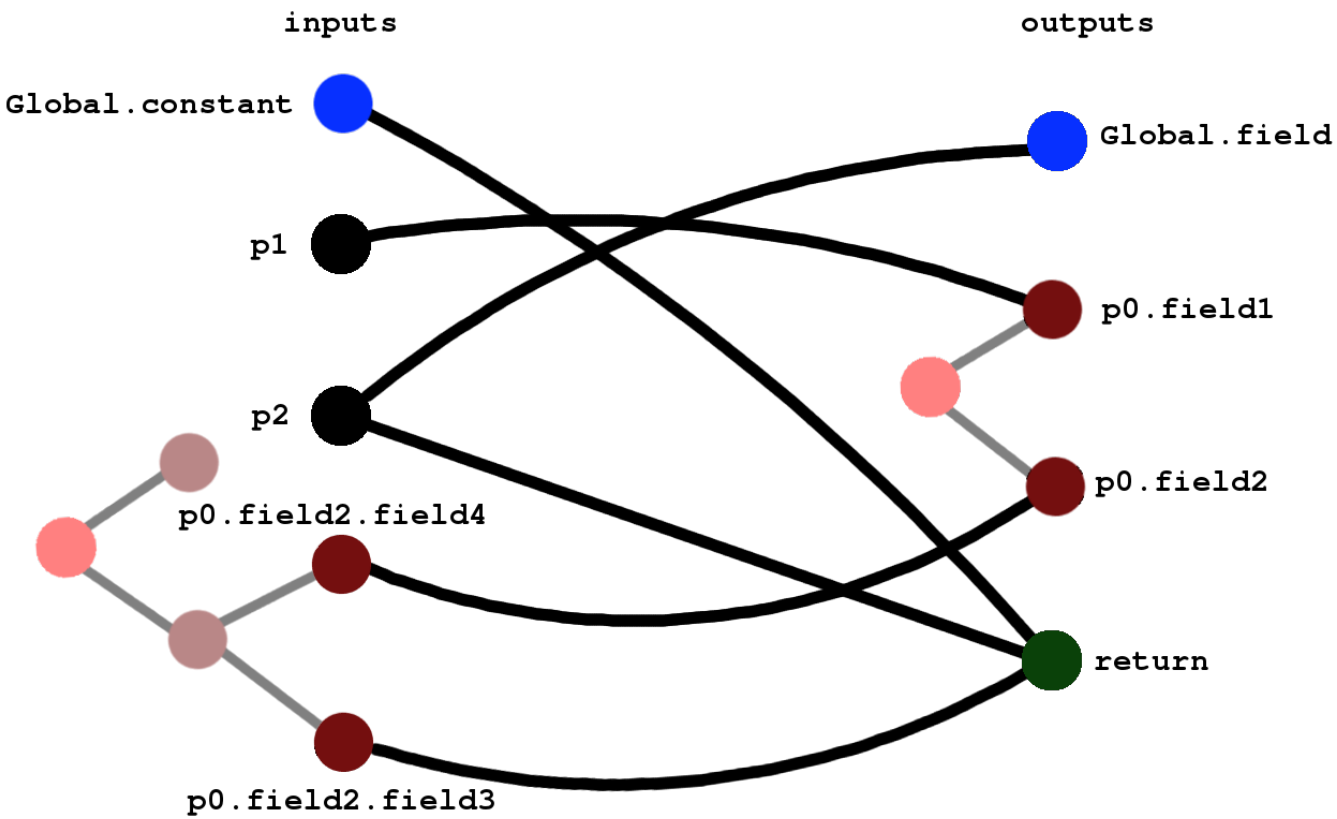
Abstract.

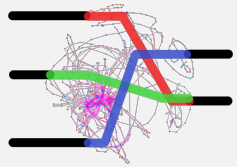


```
obj.method(param1, param2)
```

```
obj -> p0
param1 -> p1
param2 -> p2
```

```
p0.method(p1, p2)
```



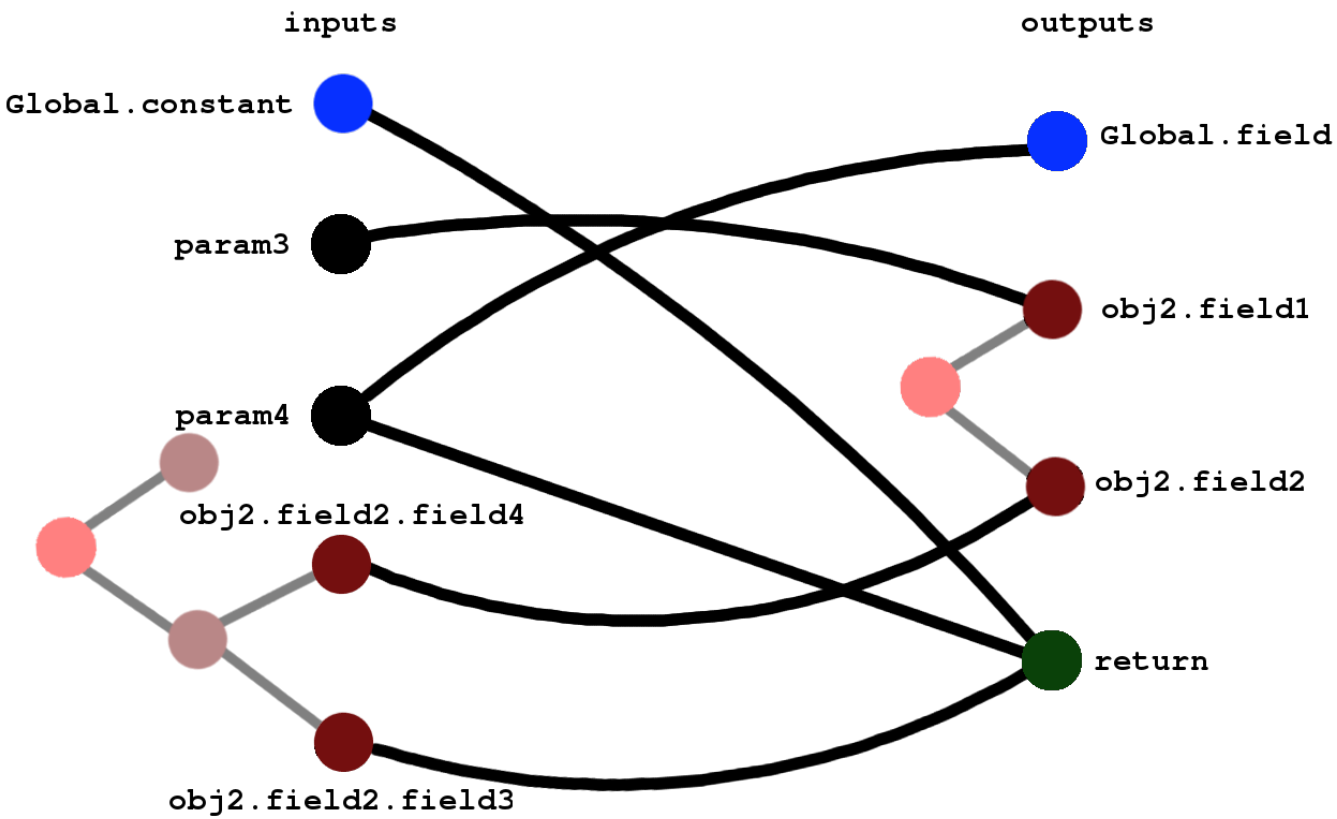


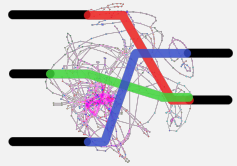
Reuse.

```
p0.method(p1, p2)
```

```
p0 -> obj2
p1 -> param3
p2 -> param4
```

```
obj2.method(param3, param4)
```



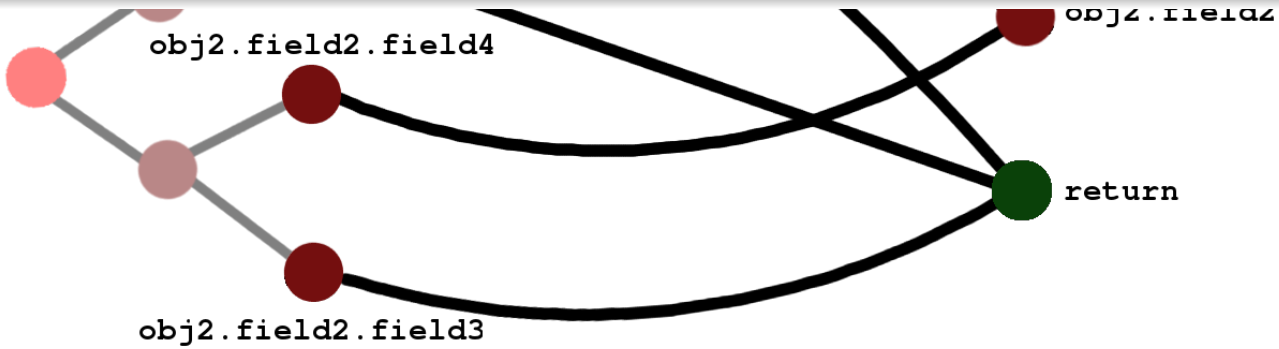


Reuse.

```
p0.method(p1, p2)
```

Key Technical Challenges Addressed.

- Summary Abstraction and Reuse.
- Precise modeling of Array element accesses.
- Accounting for Varying Method Behavior due to polymorphism.
- Handling object-graph mismatch.
- Object sensitivity.



Implementation.

Key Parts

Implementation.

Key Parts

- Includes:

Java Bytecode Instrumenter

`uses:Java; ASM (asm.org)`

Implementation.

Key Parts

- Includes:

Java Bytecode Instrumenter

`uses:Java; ASM (asm.org)`

Trace Analyzer (Dataflow or Dependencies)

`uses:Java`

Implementation.

Key Parts

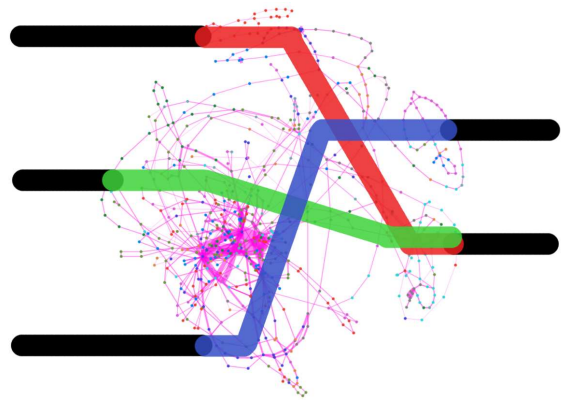
- Includes:

Java Bytecode Instrumenter

`uses:Java; ASM (asm.org)`

Trace Analyzer (Dataflow or Dependencies)

`uses:Java`



Dependence Summarizer

`uses:Java`

Implementation.

Key Parts

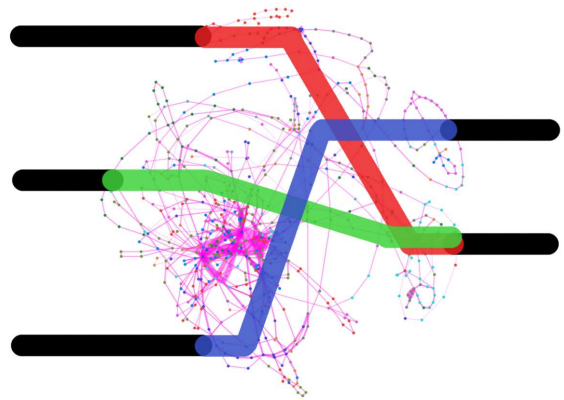
- Includes:

Java Bytecode Instrumenter

`uses:Java; ASM (asm.org)`

Trace Analyzer (Dataflow or Dependencies)

`uses:Java`



Dependence Summarizer

`uses:Java`

- Built using: `Java; ASM (asm.org)`

Experiment One: Performance.

RQ1

How does the reuse of dynamic dependence summaries affect the costs of dynamic analysis?

Experiment One: Performance.

RQ1

How does the reuse of dynamic dependence summaries affect the costs of dynamic analysis?

Metrics

Execution Trace Size.
Execution Running Time.

Experiment One: Performance.

RQ1

How does the reuse of dynamic dependence summaries affect the costs of dynamic analysis?

Metrics

Execution Trace Size.
Execution Running Time.

Treatments

Exhaustive.
Summary-based.

Experiment One: Performance.

RQ1

How does the reuse of dynamic dependence summaries affect the costs of dynamic analysis?

Metrics

Execution Trace Size.
Execution Running Time.

Treatments

Exhaustive.
Summary-based.

Client Subjects

ANTLR (35KLOCs)
BLOAT (41KLOCS)
FOP (102KLOCs)
JYTHON (245KLOCs)
PMD (60KLOCs)

Experiment One: Performance.

RQ1

How does the reuse of dynamic dependence summaries affect the costs of dynamic analysis?

Metrics

Execution Trace Size.
Execution Running Time.

Treatments

Exhaustive.
Summary-based.

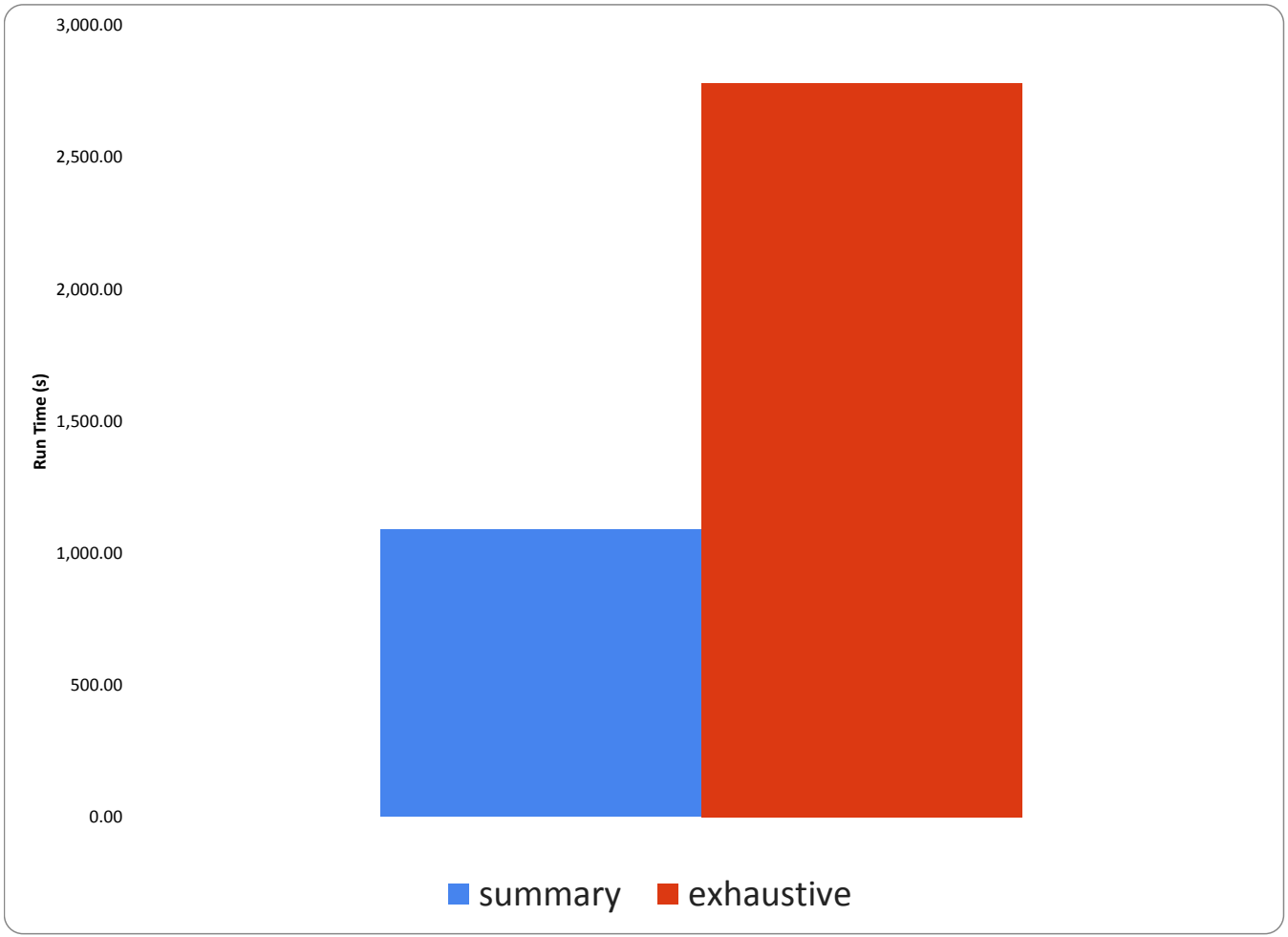
Client Subjects

ANTLR (35KLOCs)
BLOAT (41KLOCS)
FOP (102KLOCs)
JYTHON (245KLOCs)
PMD (60KLOCs)

Library Subject

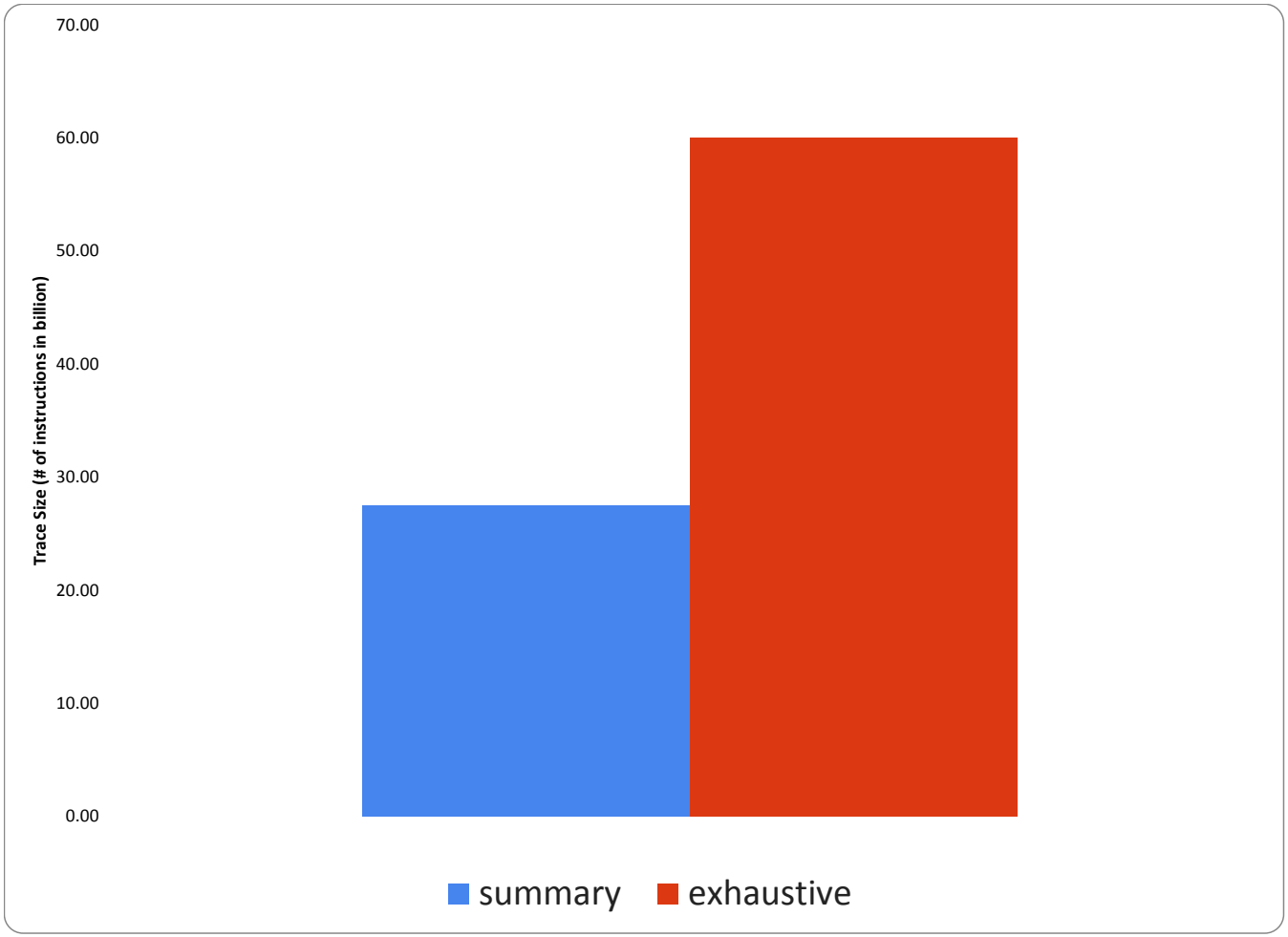
JAVA DEVELOPMENT KIT (`rt.jar`)

Results: Runtime (RQ1).



- **1.5× — 3.6× speedup in execution runtimes.**
  - Exhaustive: 112× runtime overhead
  - Summary Times: 43× runtime overhead

Results: Trace Size (RQ1).



- **44% smaller traces with summary usage.**

Experiment Two: Accuracy.

RQ2

How does the reuse of dynamic dependence summaries affect the accuracy of dynamic analysis?

Experiment Two: Accuracy.

RQ2

How does the reuse of dynamic dependence summaries affect the accuracy of dynamic analysis?

Metrics

Found Bugs. **Runtime Overhead.**

Experiment Two: Accuracy.

RQ2

How does the reuse of dynamic dependence summaries affect the accuracy of dynamic analysis?

Metrics

Found Bugs.

Runtime Overhead.

Treatment

Exhaustive.
Summary-based.

Experiment Two: Accuracy.

RQ2

How does the reuse of dynamic dependence summaries affect the accuracy of dynamic analysis?

Metrics

Found Bugs.

Runtime Overhead.

Treatment

Exhaustive.
Summary-based.

Client Subject

NanoXML (7KLOC)

Experiment Two: Accuracy.

RQ2

How does the reuse of dynamic dependence summaries affect the accuracy of dynamic analysis?

Metrics

Found Bugs.

Runtime Overhead.

Treatment

Exhaustive.
Summary-based.

Client Subject

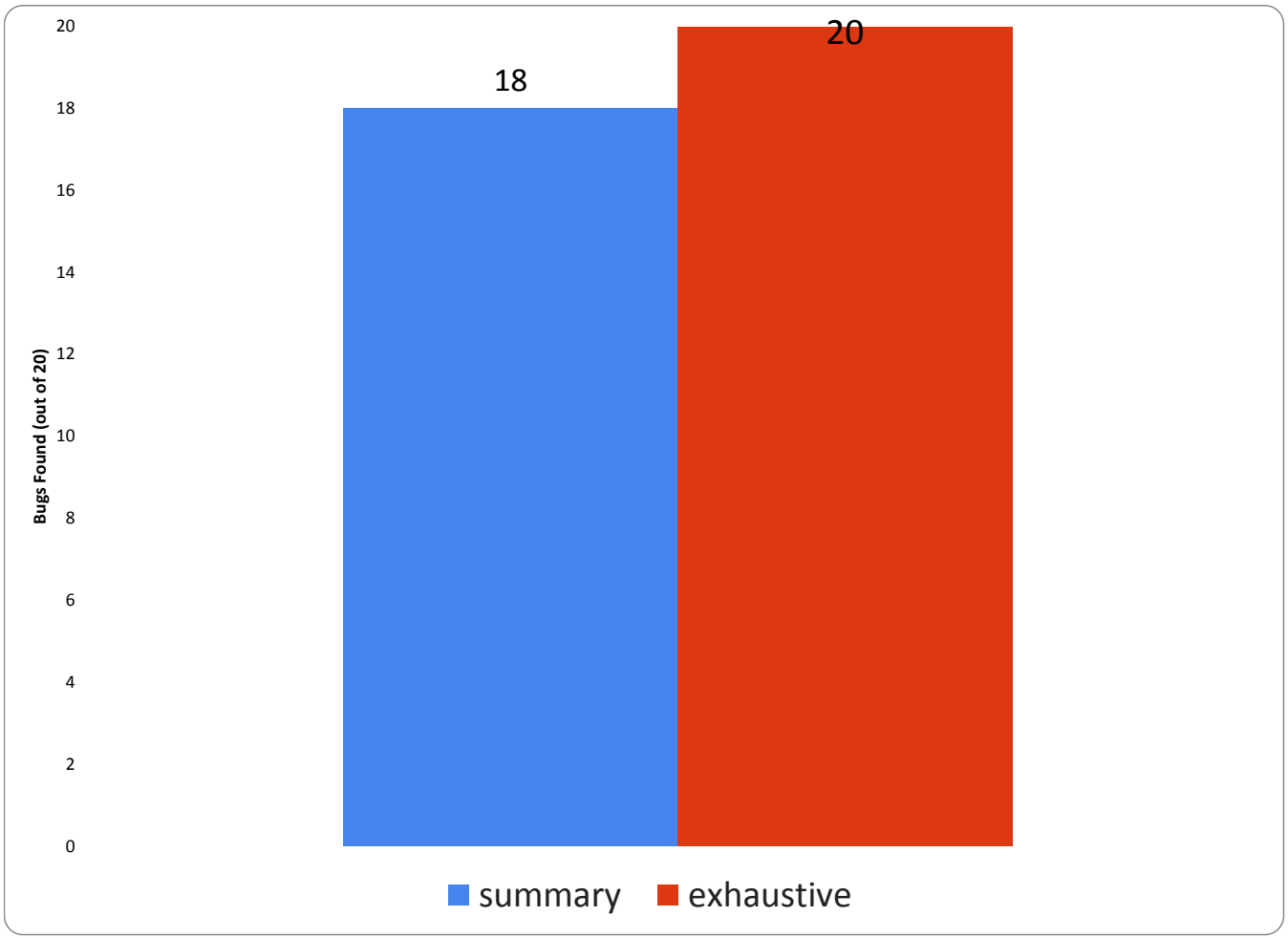
NANOXML (7KLOC)

Library Subject

JAVA DEVELOPMENT KIT (`rt.jar`)

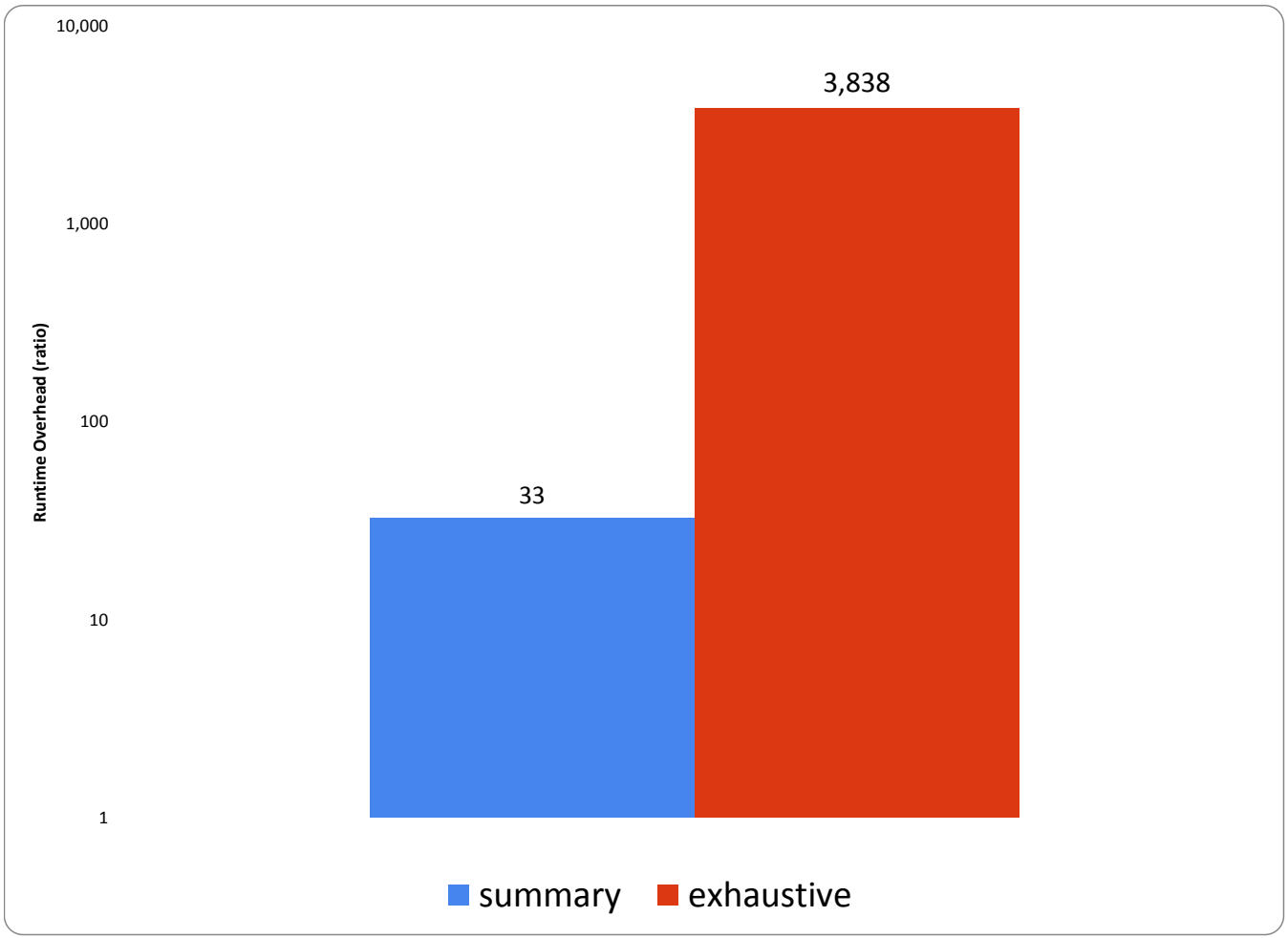
Results: Found Bugs (RQ2).

NanoXML: Exhaustive vs. Summary



Results: Runtime Overhead (RQ2).

NanoXML: Exhaustive vs. Summary



Open Issues.

Open Issues.

- Assess suitability for summarization; adequacy criteria.

Open Issues.

- Assess suitability for summarization; adequacy criteria.
- Comparison with static summaries.

Open Issues.

- Assess suitability for summarization; adequacy criteria.
- Comparison with static summaries.
- Accuracy analysis with multiple test subjects and client analyses.

Takeaways.

Takeaways.

- **Theory, Models, Implementation** for construction and reuse of **Dynamic Summaries**.

Takeaways.

- **Theory, Models, Implementation** for construction and reuse of **Dynamic Summaries**.
- **2$\times$ performance gains** (best case: 3.6$\times$) while analyzing large software benchmarks.

Takeaways.

- **Theory, Models, Implementation** for construction and reuse of **Dynamic Summaries**.
- **2$\times$ performance gains** (best case: 3.6$\times$) while analyzing large software benchmarks.
- Empirical study indicates **cost savings with modest accuracy losses.**