UNIVERSITÄT PASSAU

Fakultät für Informatik und Mathematik

Static Analyses of Interlanguage Inter-operations

Jyoti Prakash

Rigorosum, 07.10.2024

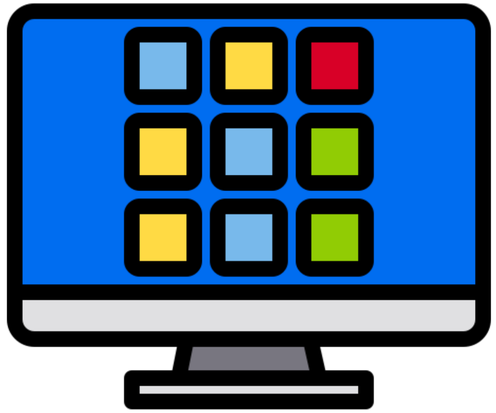
Superviser: Prof. Dr-Ing. Christian Hammer

Multilingual Programming is Everywhere

Java/Kotlin + CPP
Java/Kotlin + JavaScript

android

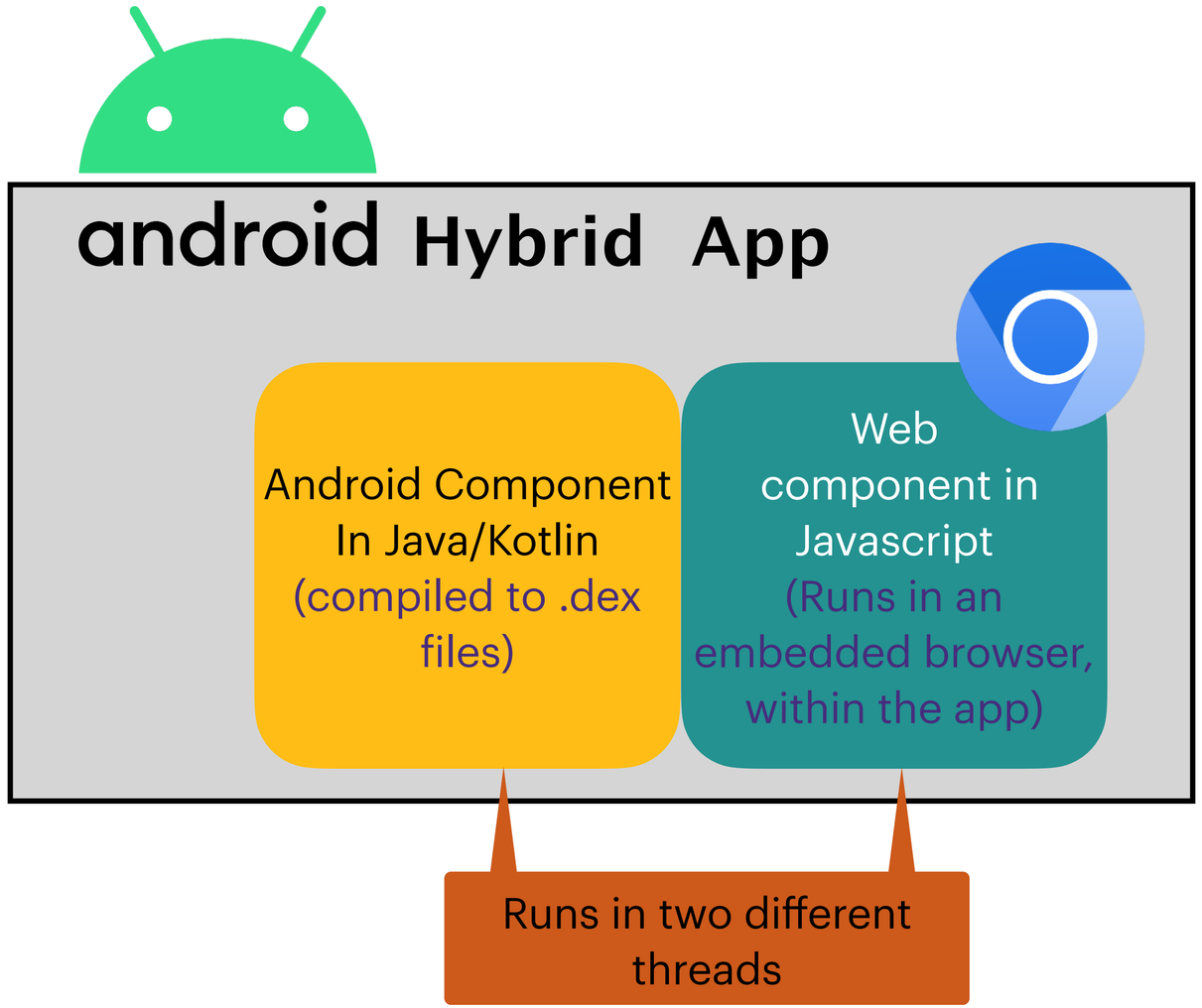
Python + C



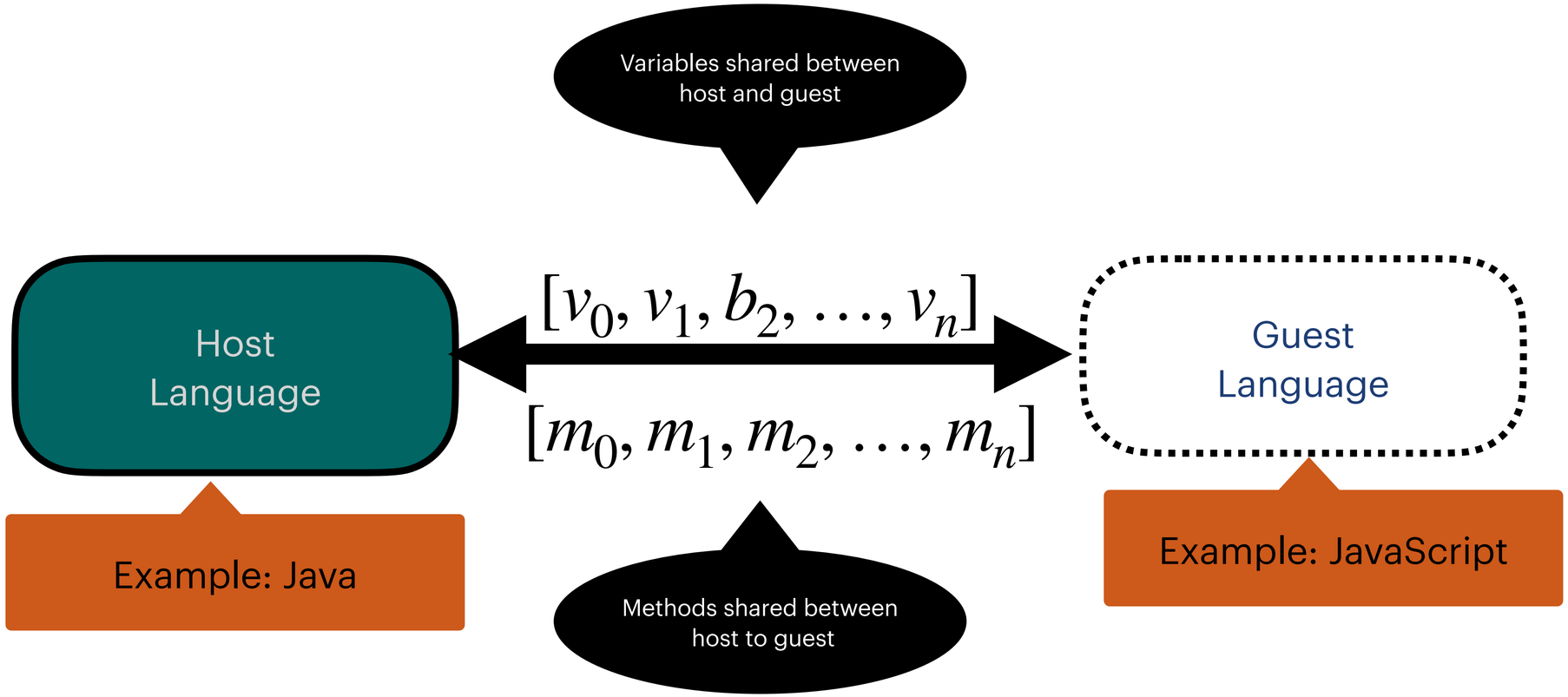
Python + C

https://medium.com/jeremy-gottfrieds-tech-blog/a-complete-guide-to-modern-web-applications-793ae71b57ad

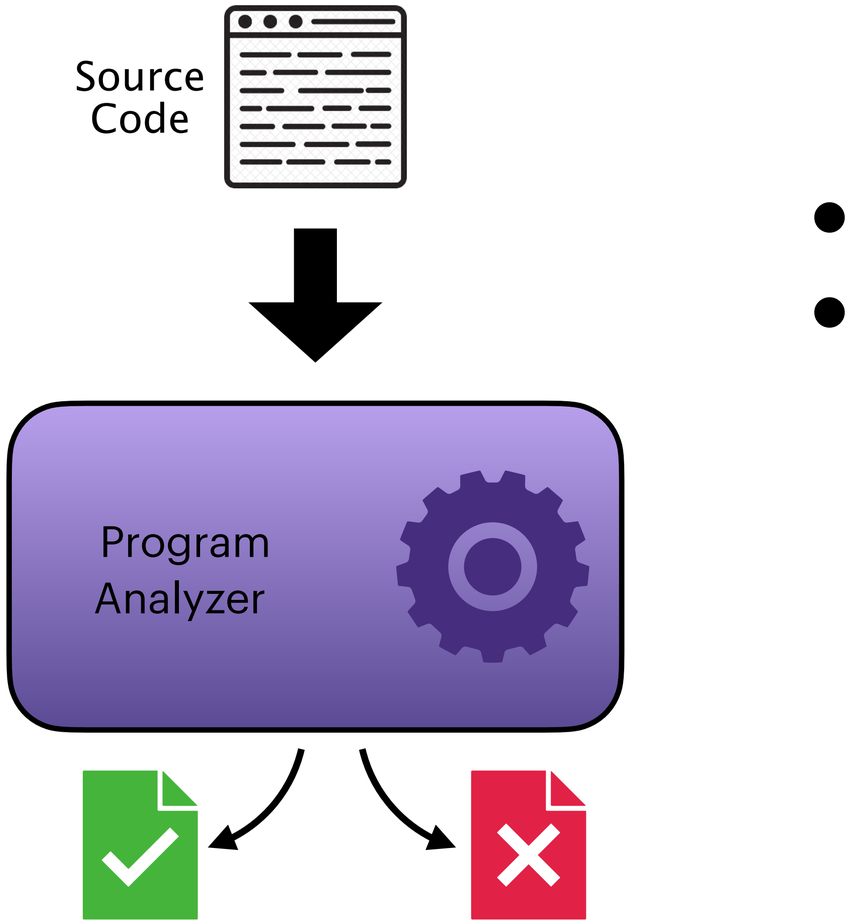
Android Hybrid Apps



Multilingual Programming Model



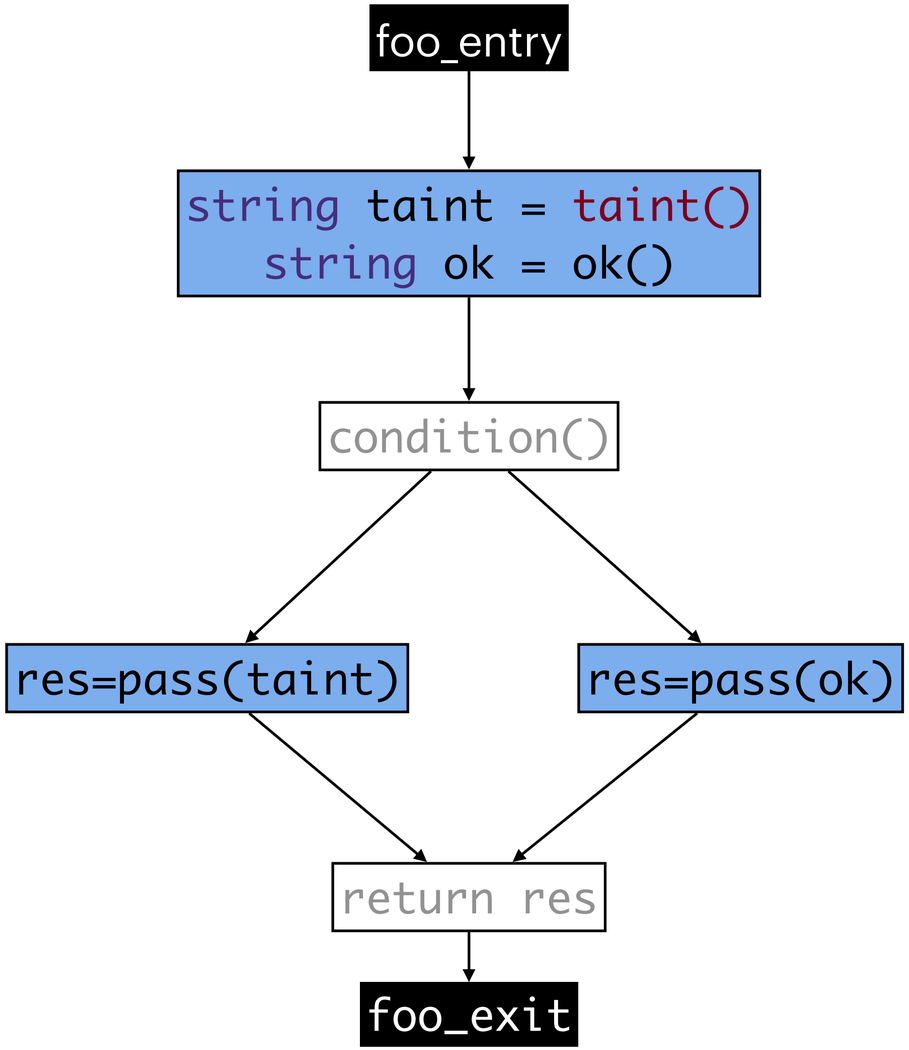
Static Program Analysis



- Analyse source-code without running it
- Balance between soundness, completeness, and scalability
  - Soundness:
    don't miss any errors
  - Completeness:
    no false positives
  - Scalability:
    report errors within limited time budgets

Scaling Static Analyses with Function Summaries

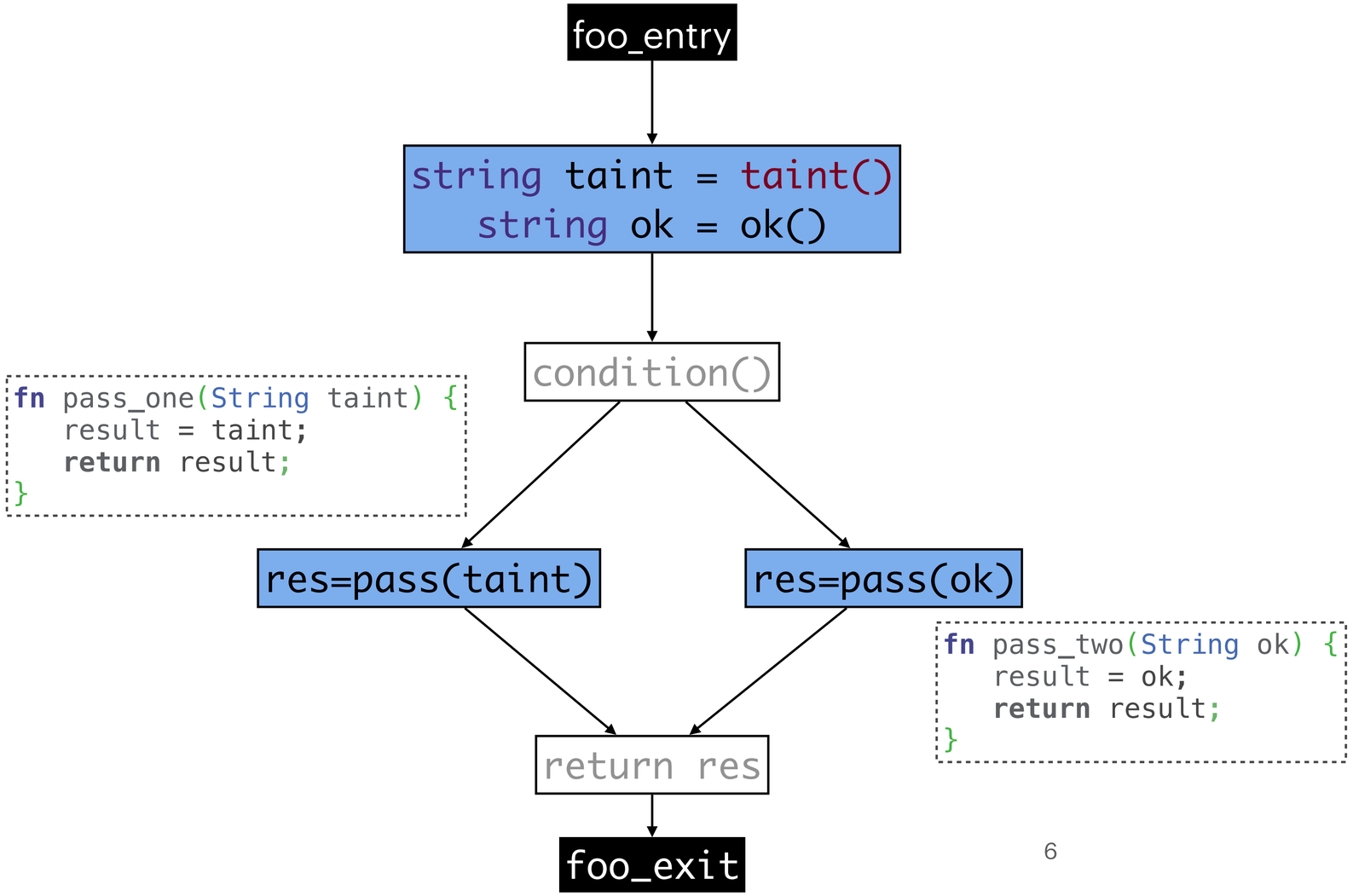
Taint Analysis




```
fn pass(String p) {
    result = p;
    return result;
}
```

Scaling Static Analyses with Function Summaries

Taint Analysis



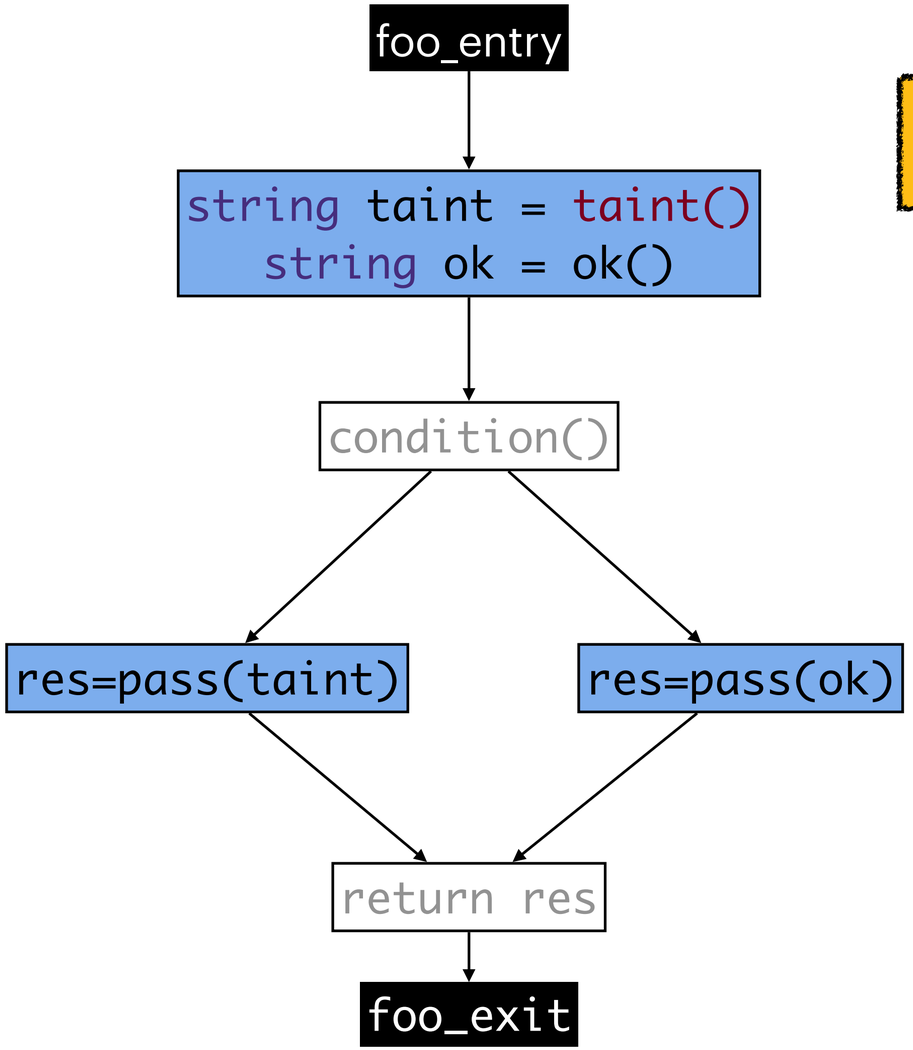

```
fn pass_one(String taint) {
    result = taint;
    return result;
}
```

```
fn pass_two(String ok) {
    result = ok;
    return result;
}
```

```
fn pass(String p) {
    result = p;
    return result;
}
```

Scaling Static Analyses with Function Summaries

Taint Analysis

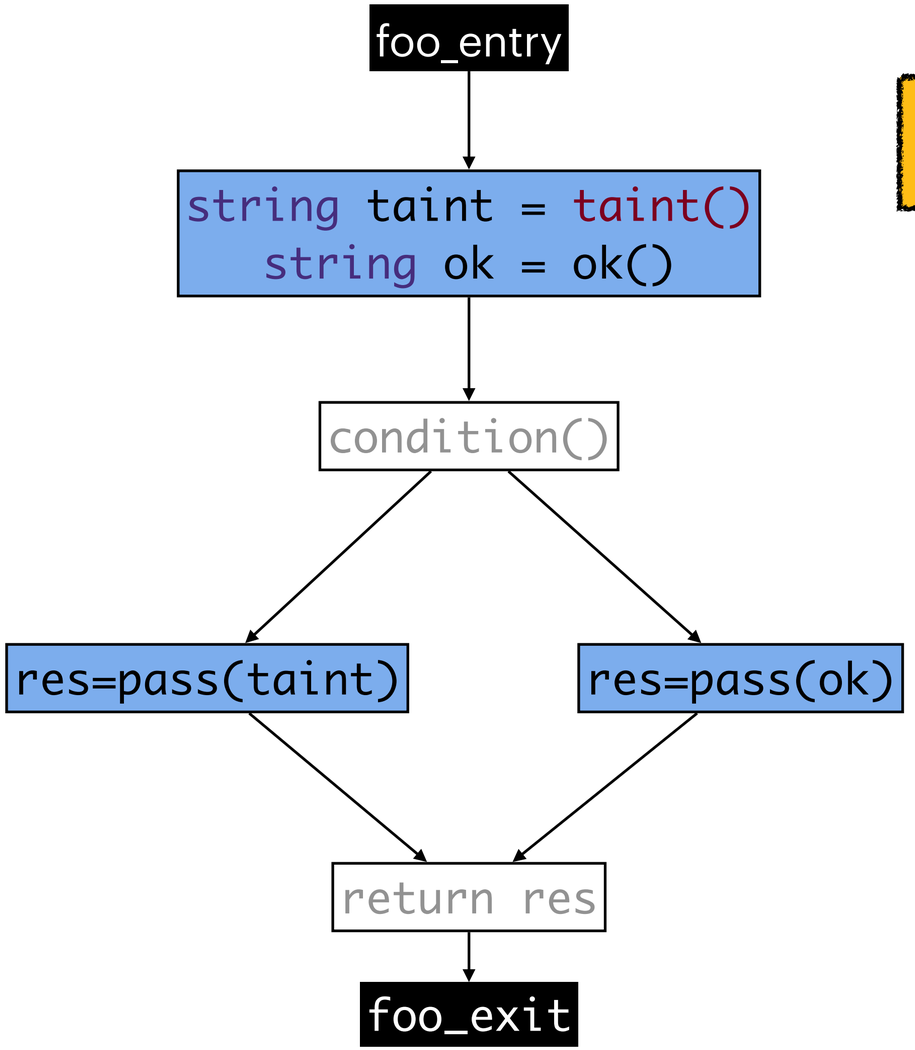


Escape multiple analysis of the same function

```
fn pass(String p) {
    result = p;
    return result;
}
```

Scaling Static Analyses with Function Summaries

Taint Analysis



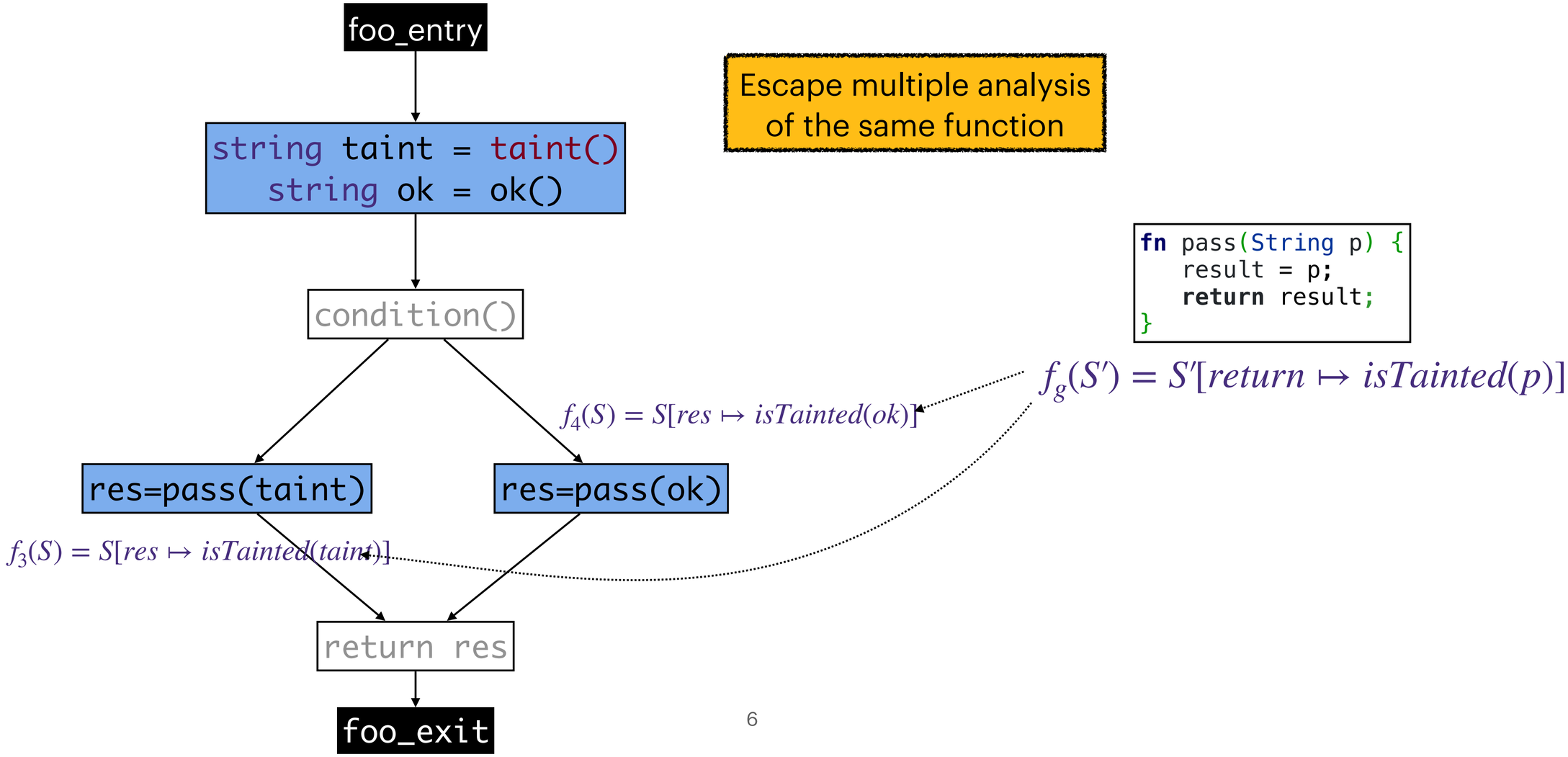
Escape multiple analysis of the same function

```
fn pass(String p) {
   result = p;
   return result;
}
```

$$f_g(S') = S'[return \mapsto isTainted(p)]$$

Scaling Static Analyses with Function Summaries

Taint Analysis

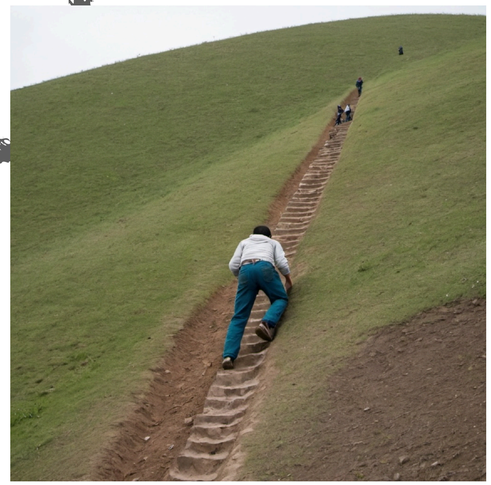


Multilingual Programming

Interoperability and reusability of parts of software components written in different languages

Convenient for developers

Challenging for static analysis

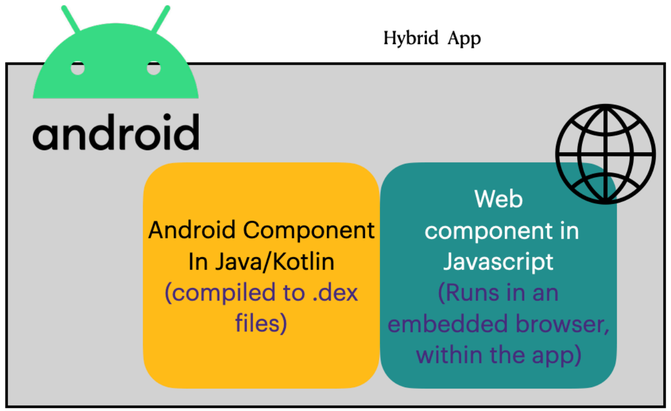


Images have been generated using AI

Thesis Objective

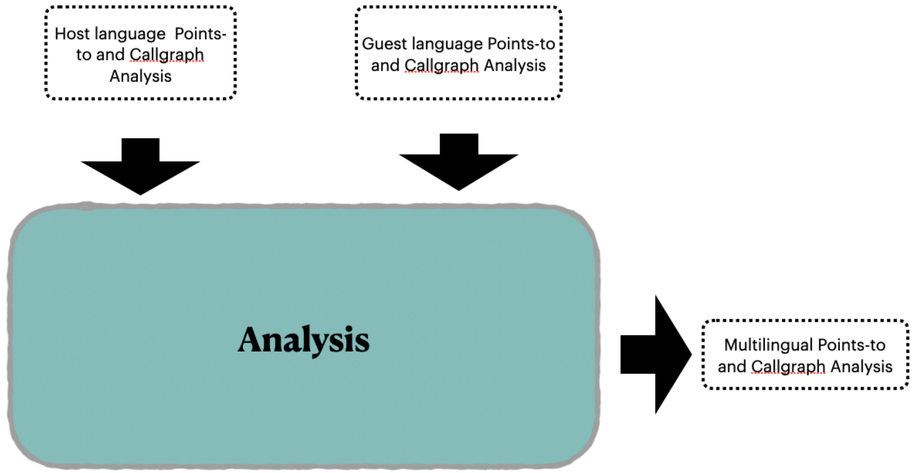
Devise scalable techniques for static analysis of multilingual programs through function summaries

Objective of the Thesis



1

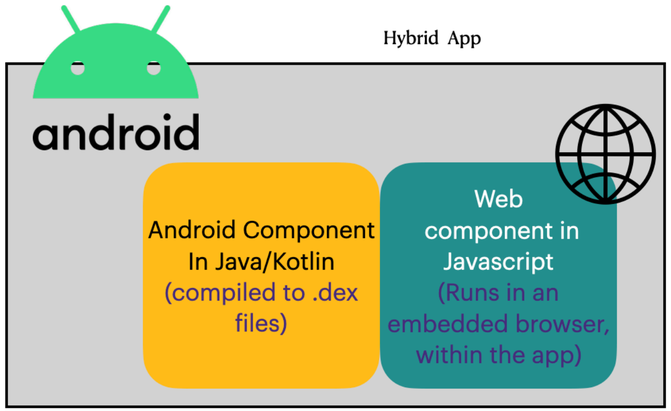
How vulnerable is the communication in Android Hybrid Apps?



2

Can we combine monolingual static analyses for analysing multilingual applications?

Objective of the Thesis



1

How vulnerable is the communication in Android Hybrid Apps?

2

Can we combine monolingual static analyses for analysing multilingual applications?

Multilingual Programming in Hybrid Apps

android

```
1. foo(WebView myWebView) {
2.   JSObj js = new JSObj(); //obj@iface
3.   myWebView.addJavaScriptInterface(js,"jsobj") //webview
4.   myWebView.evaluateJavascript("set()");
5.   myWebView.loadUrl("javascript:alert('Hello World')");
6.   myWebView.loadUrl("https://www.google.com");
7. }
```

APIs

```
1. class JSObj {
   …

10.    @JavaScriptInterface
11.    public void setValue(Object p) {
12.       this.f = p;
13.    }

200. }
```

Java

```
21. set() {
22.   x = new Object() // obj@x
23.   x.f = JSON.parse(api.getResult())
24.   v = x.f
25.   jsobj.setValue(v)
26. }
```

Javascript

Multilingual Programming in Hybrid Apps

android

```
1. foo(WebView myWebView) {
2.    JSObj js = new JSObj(); //obj@iface
3.    myWebView.addJavaScriptInterface(js,”jsobj”) //webview
4.    myWebView.evaluateJavascript(“set()”);
5.    myWebView.loadUrl("javascript:alert('Hello World')");
6.    myWebView.loadUrl(“https://www.google.com“);
7. }
```

```
1. class JSObj {
   …

10.     @JavaScriptInterface
11.     public void setValue(Object p) {
12.        this.f = p;
13.     }

200. }
```

Shared/Bridge methods

Java

```
21. set() {
22.    x = new Object() // obj@x
23.    x.f = JSON.parse(api.getResult())
24.    v = x.f
25.    jsobj.setValue(v)
26. }
```

Javascript

Multilingual Programming in Hybrid Apps

android

Bridge/Interface variable

```
1. foo(WebView myWebView) {
2.   JSObj js = new JSObj(); //obj@iface
3.   myWebView.addJavaScriptInterface(js,"jsobj") //webview
4.   myWebView.evaluateJavascript("set()");
5.   myWebView.loadUrl("javascript:alert('Hello World')");
6.   myWebView.loadUrl("https://www.google.com");
7. }
```

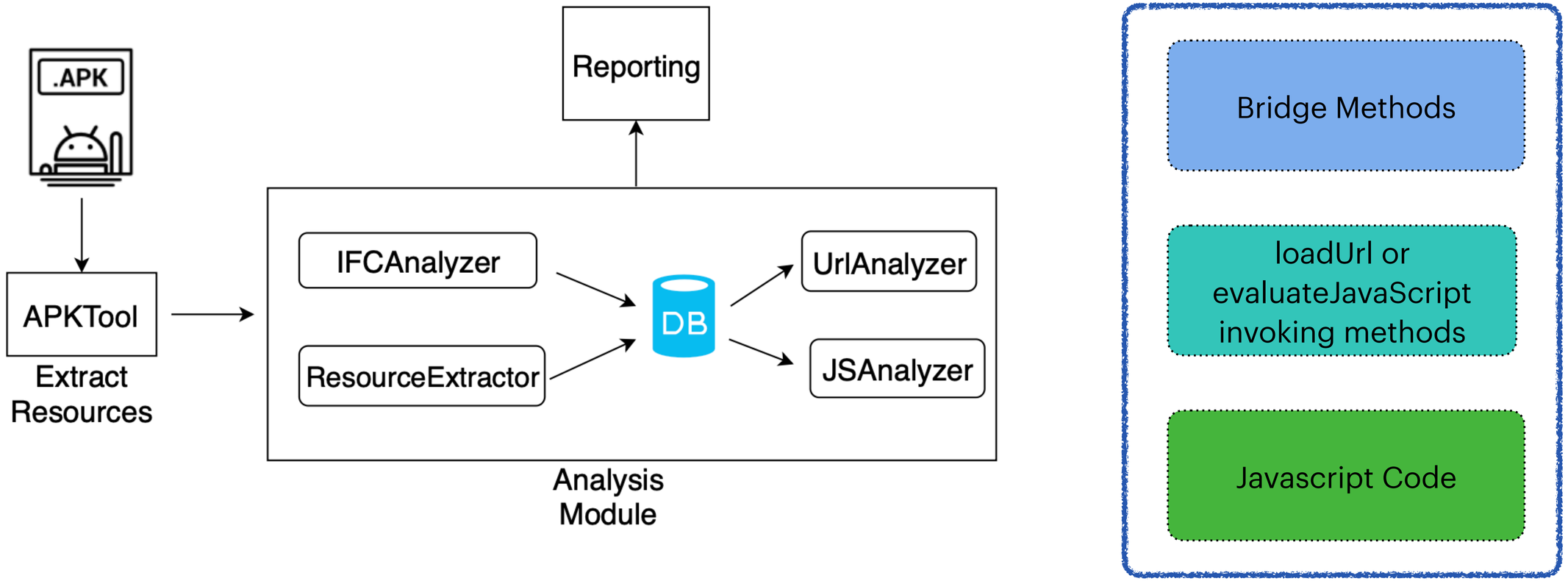
```
1. class JSObj {
   …

10.    @JavaScriptInterface
11.    public void setValue(Object p) {
12.       this.f = p;
13.    }

200. }
```

Java

```
21. set() {
22.    x = new Object(); // obj@x
23.    x.f = JSON.parse(api.getResult());
24.    v = x.f;
25.    jsobj.setValue(v)
26. }
```

Javascript

LuDroid: Tool to understand Multilingual programs in Hybrid Apps



A Large Scale Analysis of Android - Web Hybridization
Abhishek Tiwari, Jyoti Prakash, Sascha Groß, and Christian Hammer
Journal of Systems and Software 170

LuDroid — A Large Scale Analysis of Android - Web Hybridization
Abhishek Tiwari, Jyoti Prakash, Sascha Groß, and Christian Hammer
19th International Working Conference on Source Code Analysis and Manipulation (SCAM'2019)

Common Patterns

Shared "setter" methods

```
1 javascript :window.SynchJS.setValue((function(){
2     try{
3         return  JSON.parse(Sponsorpay.MBE
      .SDKInterface.do_getOffer()).uses_tpn;
4     }catch( js_eval_err ){
5         return  false;
6 }})());
```

Shared "setter" methods

```
window.HTMLOUT.processHTML(
  document.getElementById('SearchResults')
  .innerHTML
);
```

Pattern observed in 30% of benign apps (N=7500) and 60% of malware apps (N=1000)

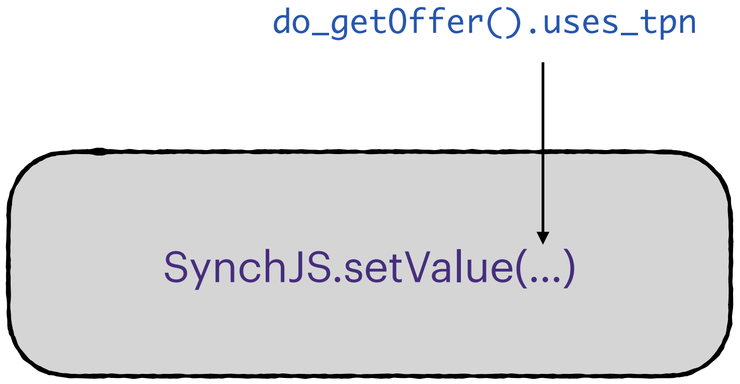
Information Flow Analysis

Inter-language Threat Model For Bridge Objects

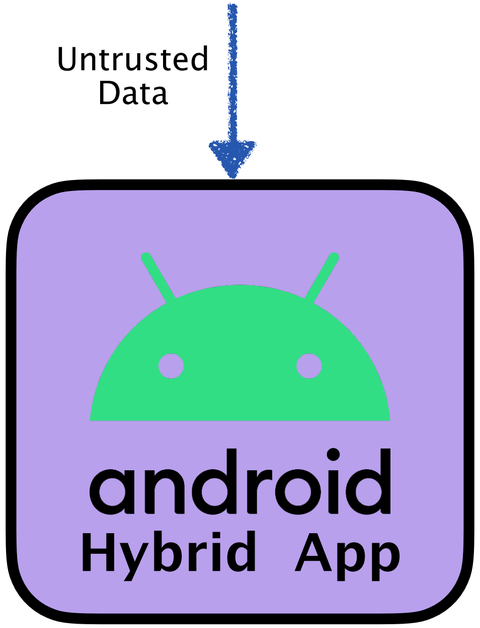
Information Flow Analysis

Inter-language Threat Model For Bridge Objects

```
1 javascript :window.SynchJS.setValue((function(){
2       try{
3             return  JSON.parse(Sponsorpay.MBE
        .SDKInterface.do_getOffer()).uses_tpn;
4       }catch( js_eval_err ){
5             return   false ;
6 }})());
```



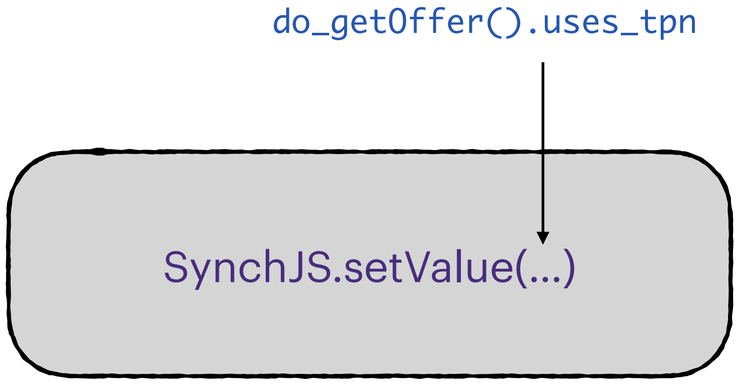
Integrity Violation



Information Flow Analysis

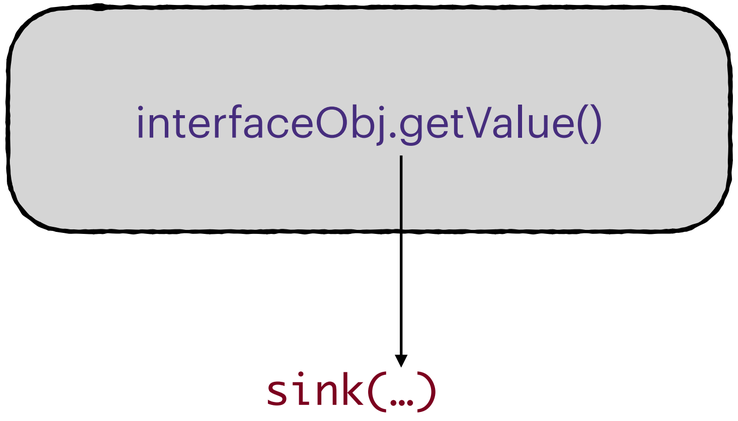
Inter-language Threat Model For Bridge Objects

```
1 javascript :window.SynchJS.setValue((function(){
2     try{
3         return JSON.parse(Sponsorpay.MBE
      .SDKInterface.do_getOffer()).uses_tpn;
4     }catch( js_eval_err ){
5         return false;
6 }})());
```



Integrity Violation

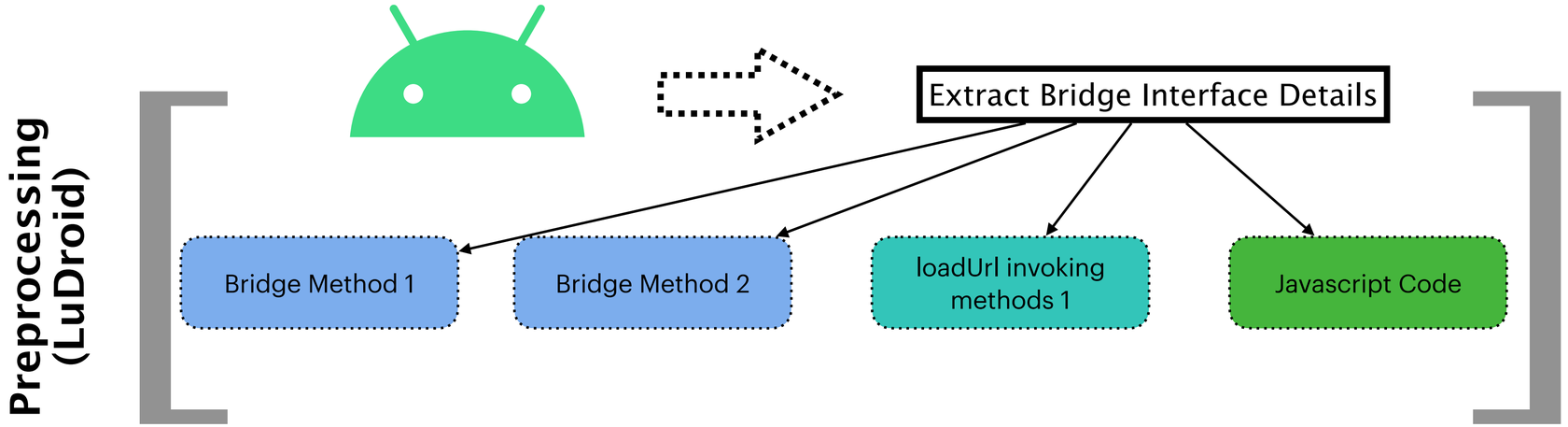

```
1 function set() {
2         var x = interfaceObj.getValue();
3         Sink(x); //New code to Sink secret x
4 }
```



Confidentiality Violation

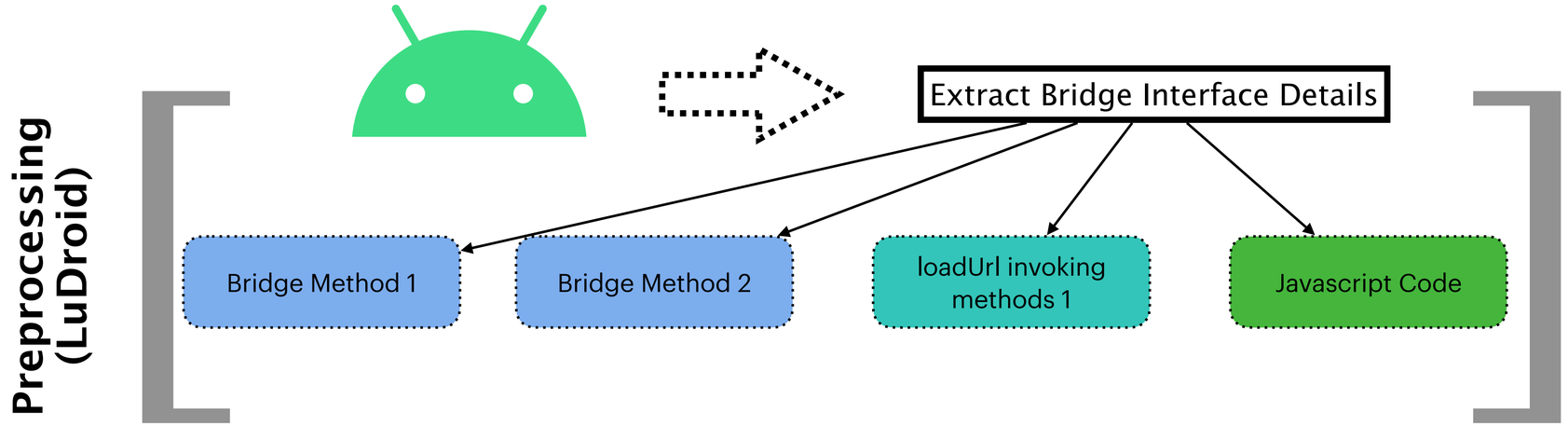
IWanDroid: Information Flow Analysis on WebView

https://iwandroid.github.io/



IWanDroid: Information Flow Analysis on WebView

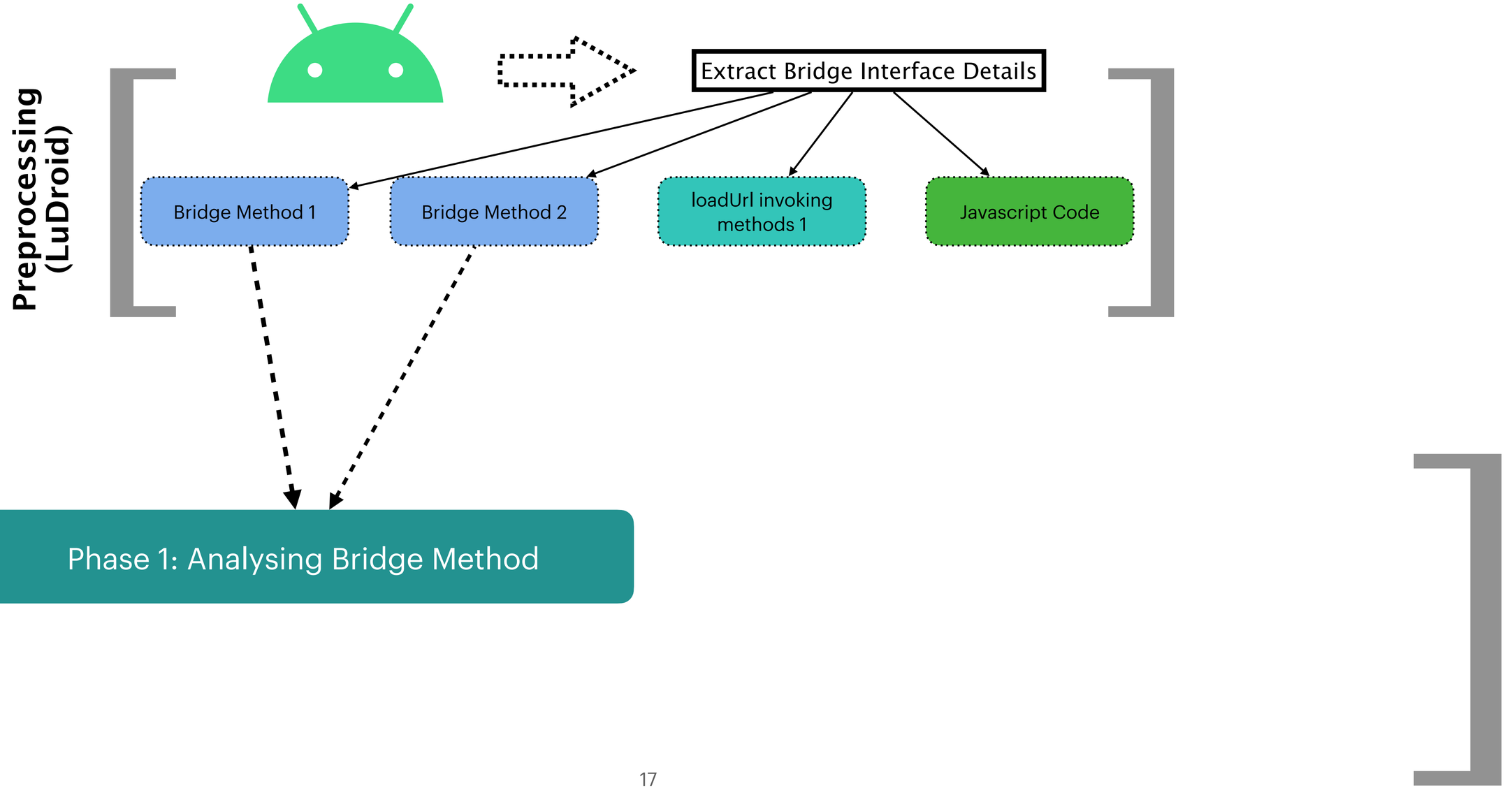
https://iwandroid.github.io/



Information Flow Analysis

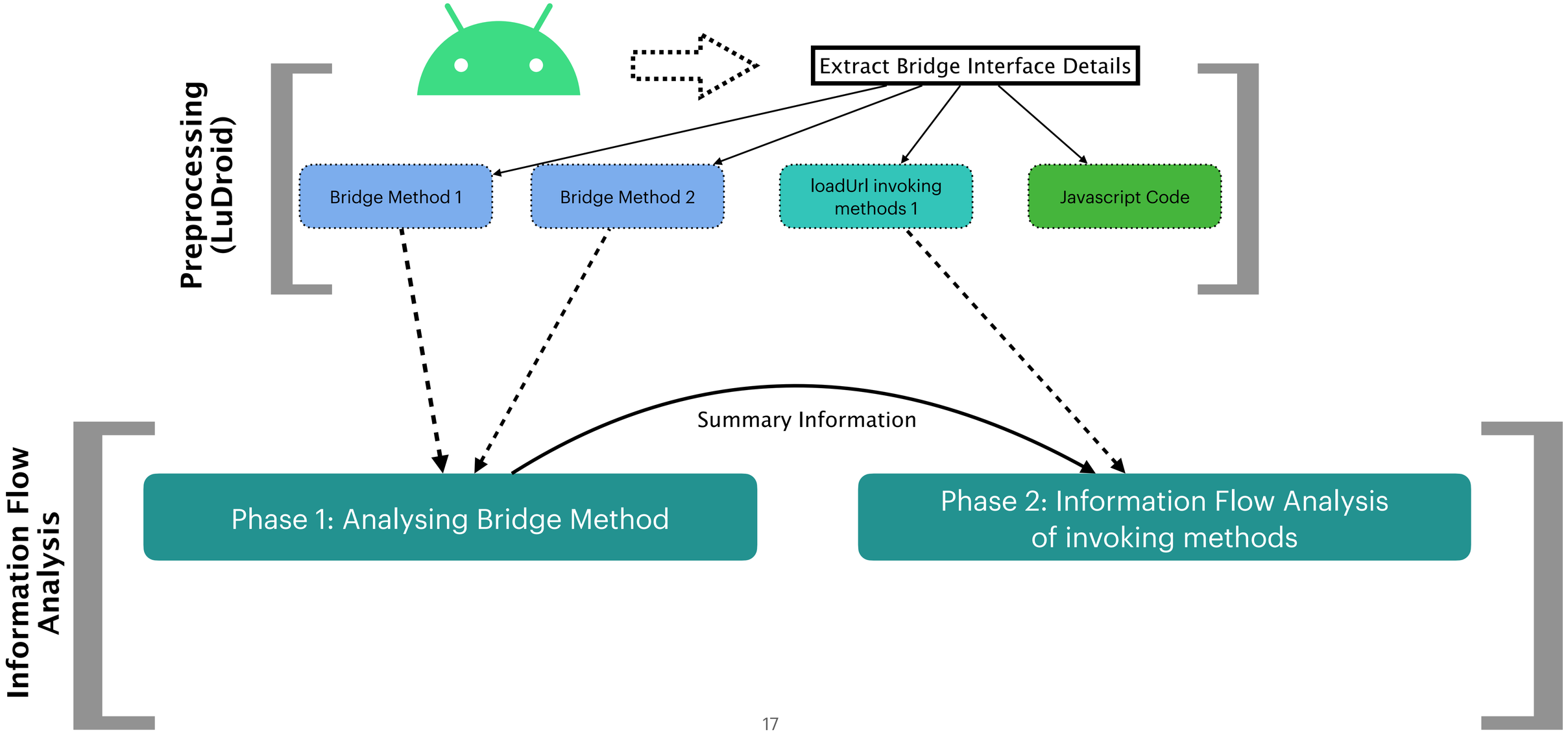
IWanDroid: Information Flow Analysis on WebView

https://iwandroid.github.io/



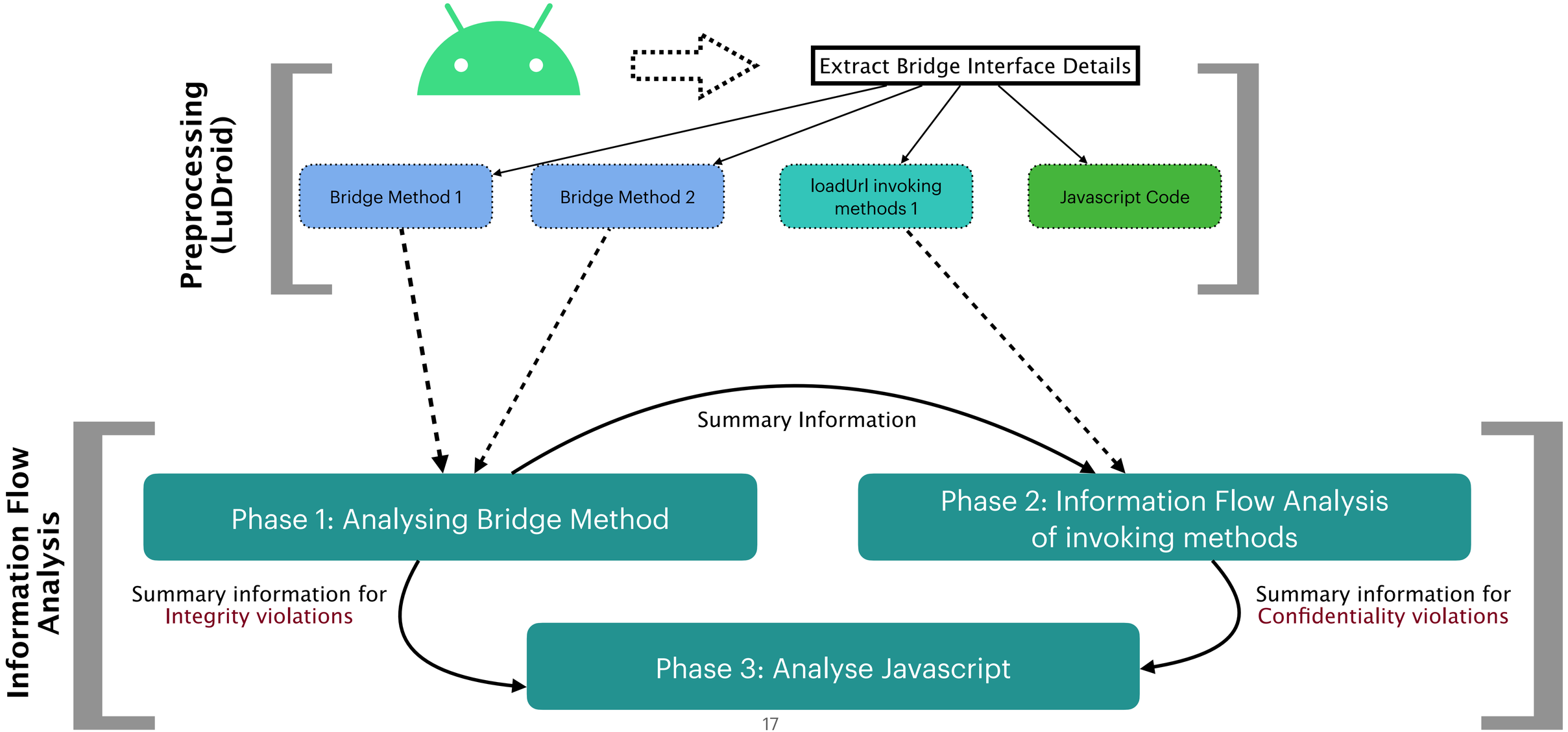
IWanDroid: Information Flow Analysis on WebView

https://iwandroid.github.io/



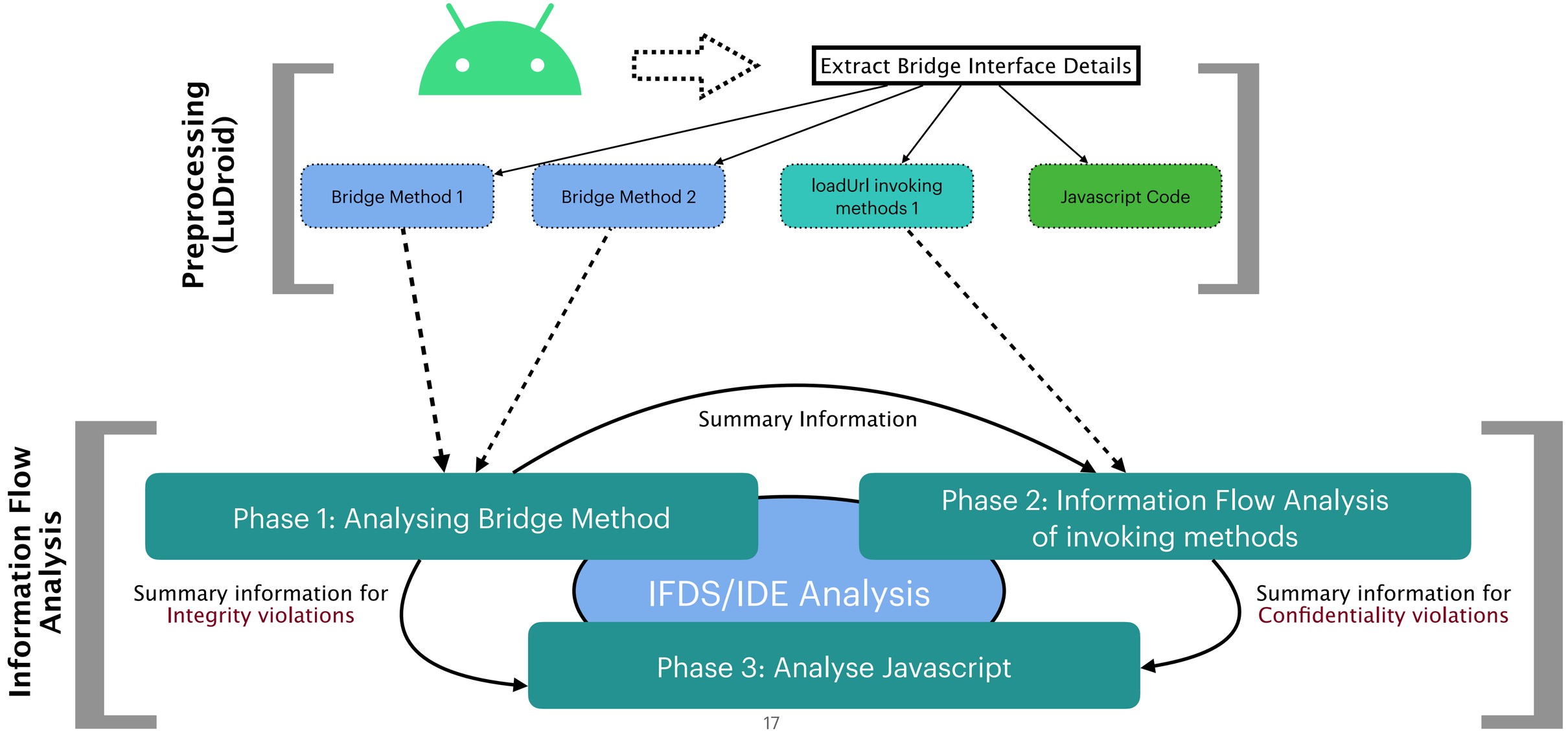
IWanDroid: Information Flow Analysis on WebView

https://iwandroid.github.io/



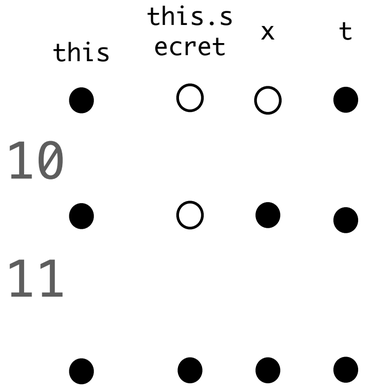
IWanDroid: Information Flow Analysis on WebView

https://iwandroid.github.io/



Analysis: Phase 1
Identify variables which are likely to influenced

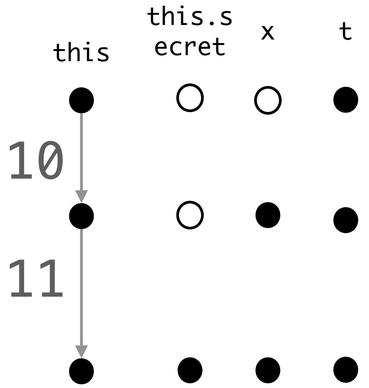
```
   @JavaScriptInterface
 9 set(Object t) {
10     var x = t;
11     this.secret = x;
   }
```



Forward analysis to discover
the set of access-paths modified
by the input parameters to the function

Analysis: Phase 1
Identify variables which are likely to influenced

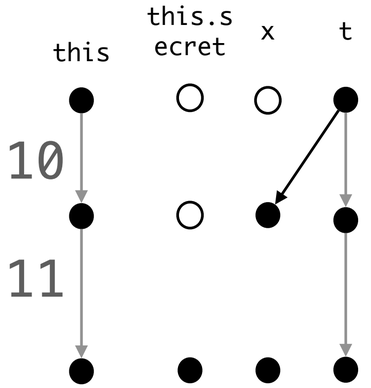
```
   @JavaScriptInterface
 9 set(Object t) {
10     var x = t;
11     this.secret = x;
   }
```



Forward analysis to discover
the set of access-paths modified
by the input parameters to the function

Analysis: Phase 1
Identify variables which are likely to influenced

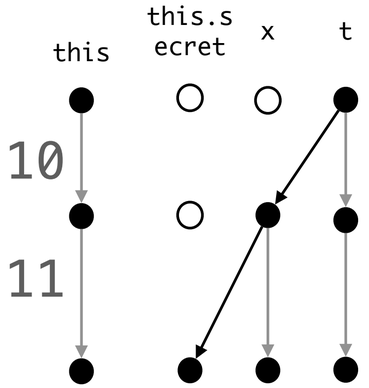
```
   @JavaScriptInterface
 9 set(Object t) {
10     var x = t;
11     this.secret = x;
   }
```



Forward analysis to discover
the set of access-paths modified
by the input parameters to the function

Analysis: Phase 1
Identify variables which are likely to influenced

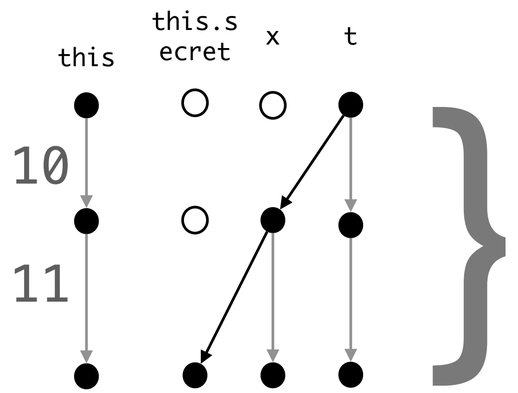
```
   @JavaScriptInterface
 9 set(Object t) {
10     var x = t;
11     this.secret = x;
   }
```



Forward analysis to discover
the set of access-paths modified
by the input parameters to the function

Analysis: Phase 1
Identify variables which are likely to influenced

```
   @JavaScriptInterface
 9 set(Object t) {
10     var x = t;
11     this.secret = x;
   }
```

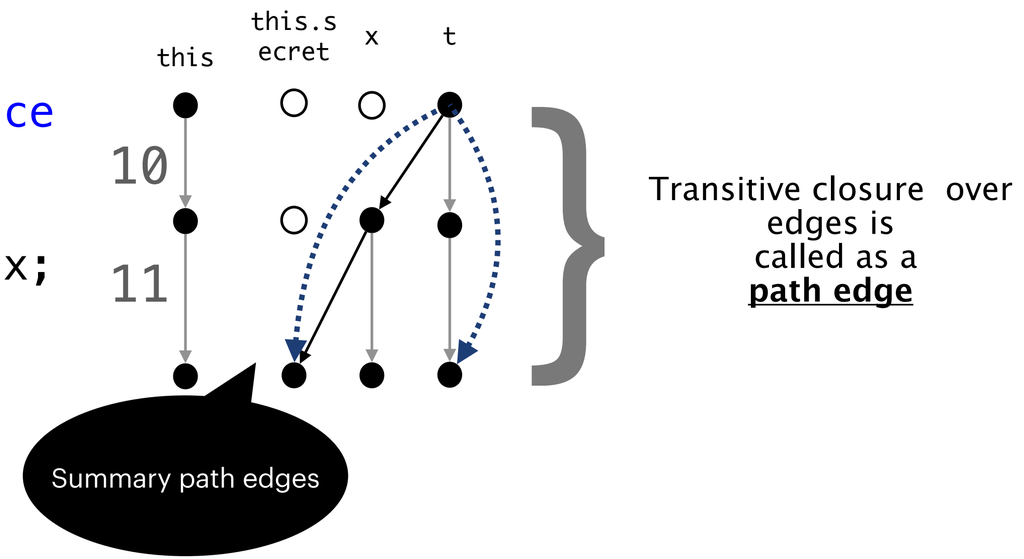


Transitive closure over edges is called as a **path edge**

Forward analysis to discover
the set of access-paths modified
by the input parameters to the function

Analysis: Phase 1
Identify variables which are likely to influenced

```
   @JavaScriptInterface
 9 set(Object t) {
10     var x = t;
11     this.secret = x;
   }
```

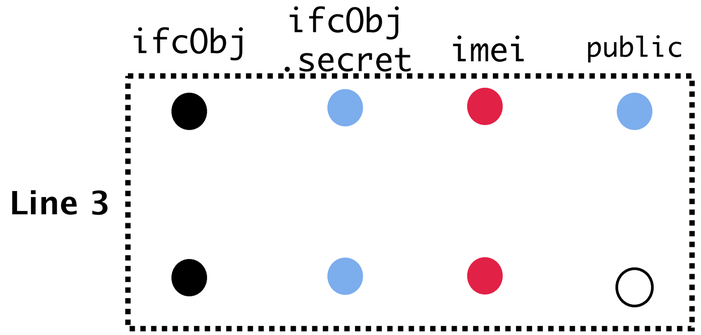


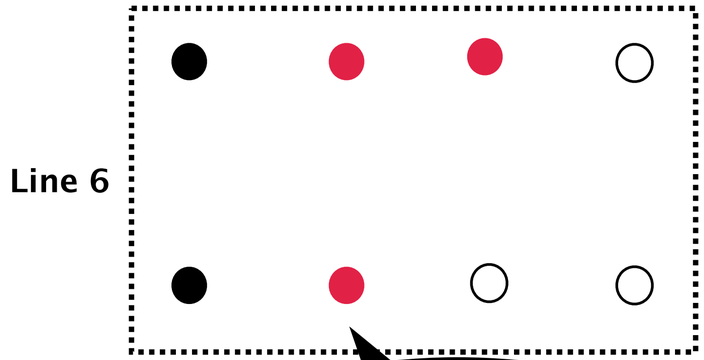
Transitive closure over edges is called as a **path edge**

Forward analysis to discover the set of access-paths modified by the input parameters to the function

Analysis: Phase 2
Analyse Invoking Methods: Backward Taint Analysis

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.set("public");
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.set(imei);
7. }
```





Backward Taint Analysis using access graphs from Phase 1

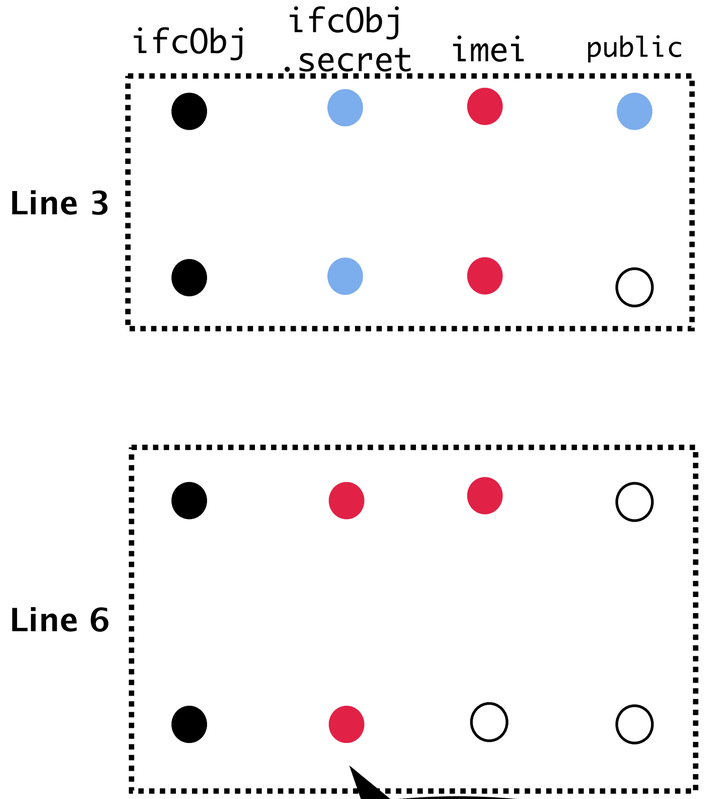
Access paths summaries from Phase 1

Analysis: Phase 2
Analyse Invoking Methods: Backward Taint Analysis

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.set("public");
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.set(imei);
7. }
```

Access paths from Phase-1

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.secret = "public";
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.secret = imei;
7.  }
```



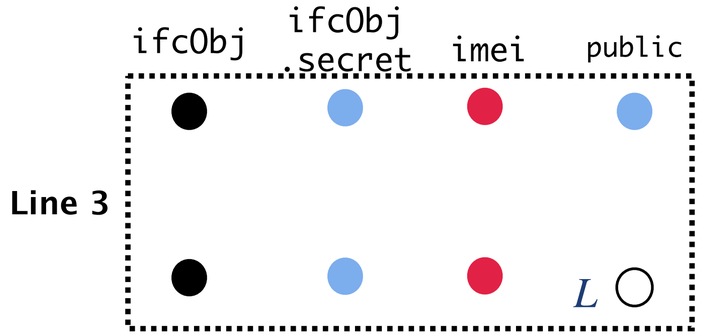
Backward Taint Analysis using access graphs from Phase 1

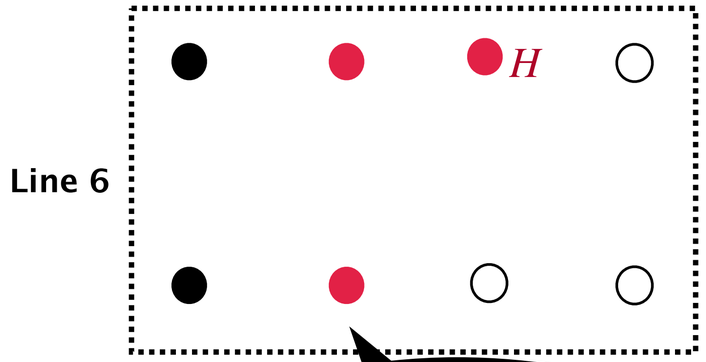
Analysis: Phase 2
Analyse Invoking Methods: Backward Taint Analysis

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.set("public");
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.set(imei);
7. }
```

Access paths from Phase-1

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.secret = "public";
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.secret = imei;
7.  }
```





Backward Taint Analysis using access graphs from Phase 1

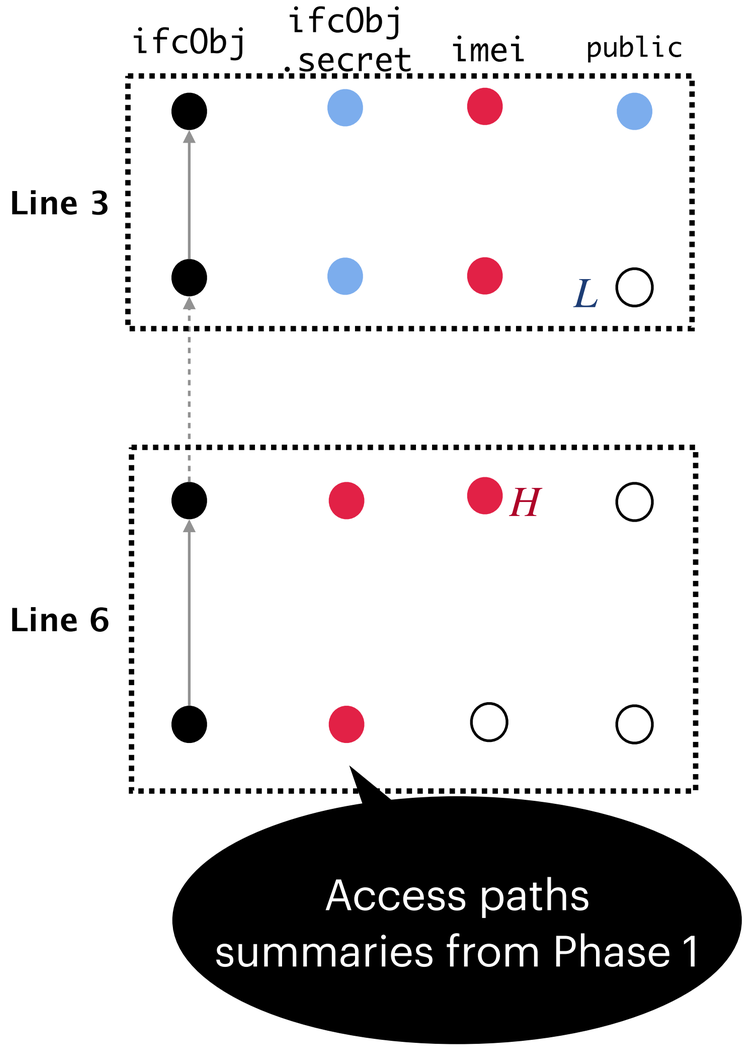
Access paths summaries from Phase 1

Analysis: Phase 2
Analyse Invoking Methods: Backward Taint Analysis

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.set("public");
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.set(imei);
7. }
```

Access paths from Phase-1

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.secret = "public";
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.secret = imei;
7.  }
```



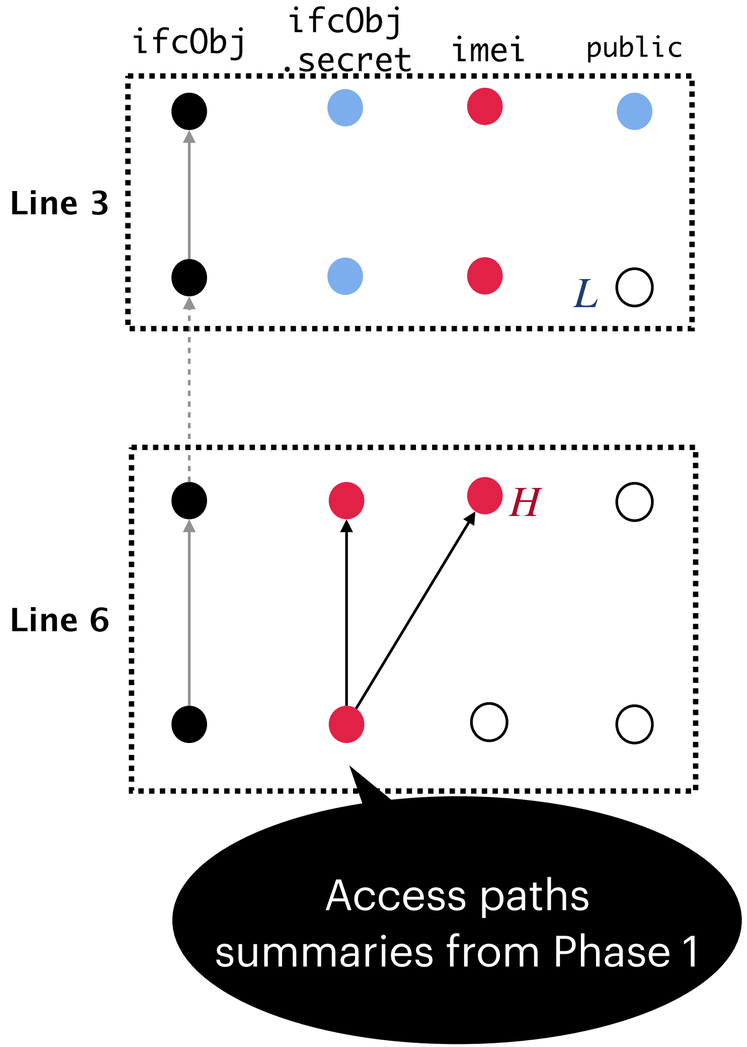
Backward Taint Analysis using access graphs from Phase 1

Analysis: Phase 2
Analyse Invoking Methods: Backward Taint Analysis

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.set("public");
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.set(imei);
7. }
```

Access paths from Phase-1

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.secret = "public";
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.secret = imei;
7.  }
```



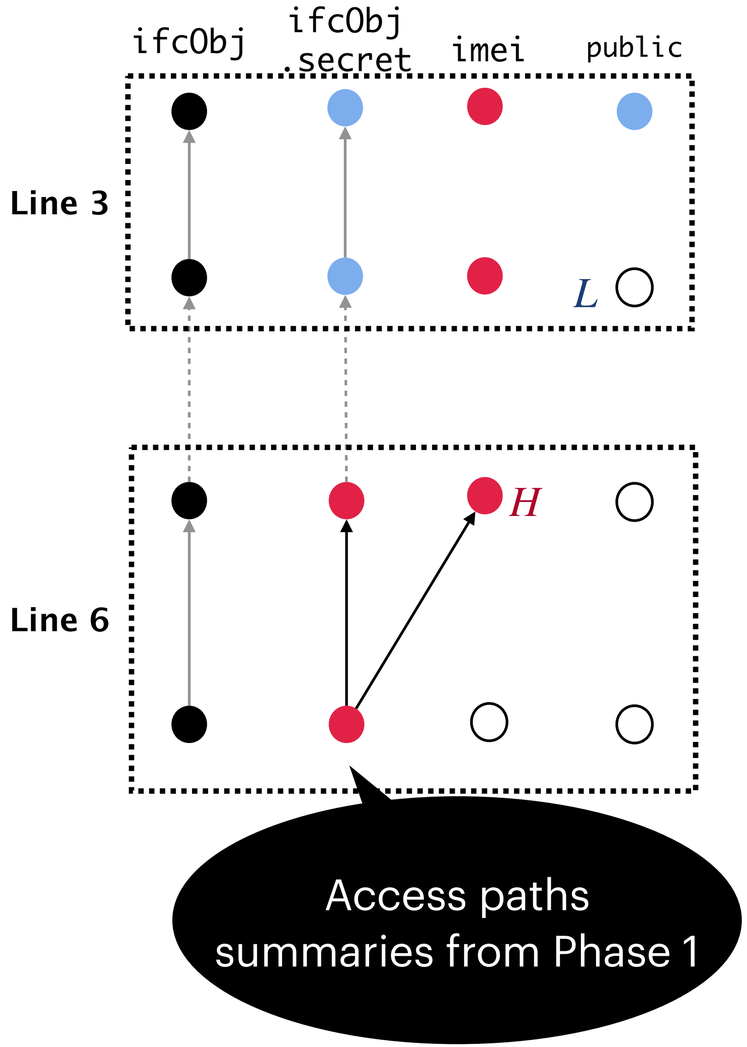
Backward Taint Analysis using access graphs from Phase 1

Analysis: Phase 2
Analyse Invoking Methods: Backward Taint Analysis

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.set("public");
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.set(imei);
7. }
```

Access paths from Phase-1

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.secret = "public";
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.secret = imei;
7.  }
```



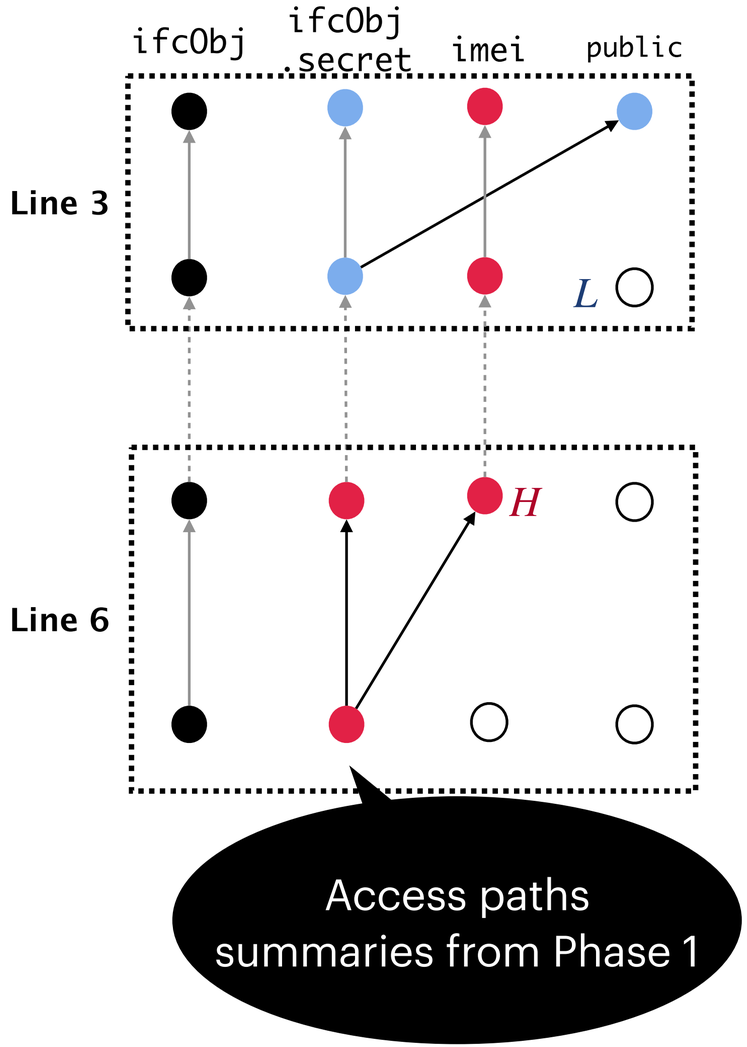
Backward Taint Analysis using access graphs from Phase 1

Analysis: Phase 2
Analyse Invoking Methods: Backward Taint Analysis

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.set("public");
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.set(imei);
7. }
```

Access paths from Phase-1

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.secret = "public";
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.secret = imei;
7.  }
```



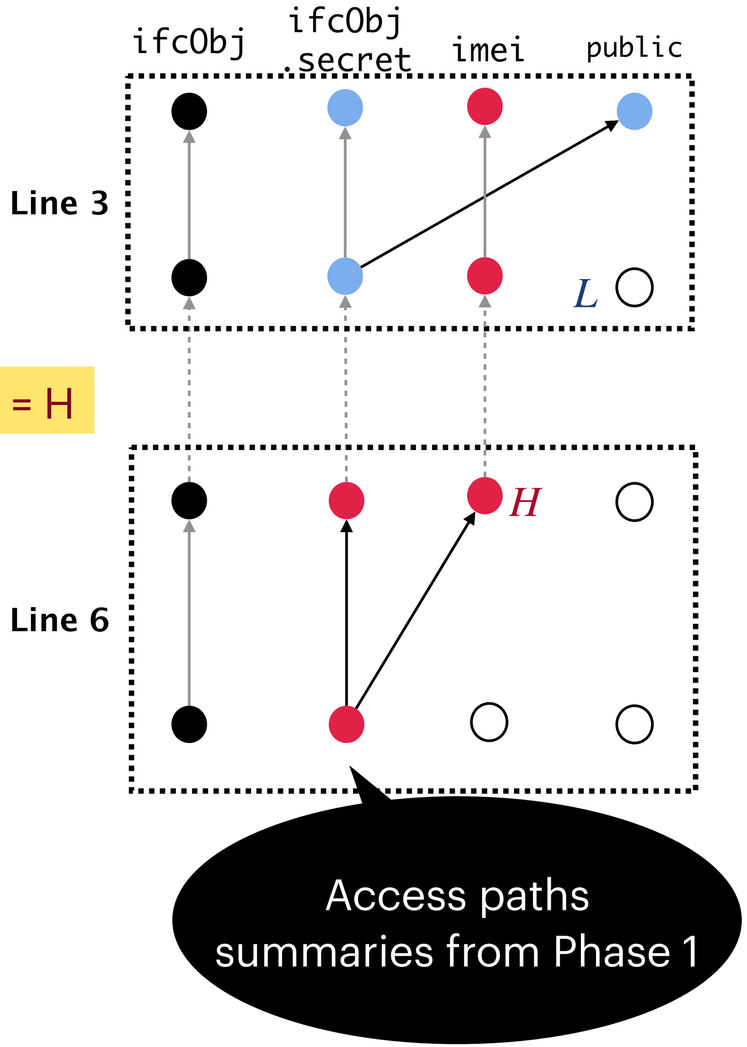
Backward Taint Analysis using access graphs from Phase 1

Analysis: Phase 2
Analyse Invoking Methods: Backward Taint Analysis

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.set("public");
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.set(imei);
7. }
```

Access paths from Phase-1

```
1. foo() {
2.    BridgeObj ifcObj = new BridgeObj(this)
3.    ifcObj.secret = "public";
4.    webView.evaluateJavascript("get()");
5.    var imei = getImei();
6.    ifcObj.secret = imei;
7.  }
```



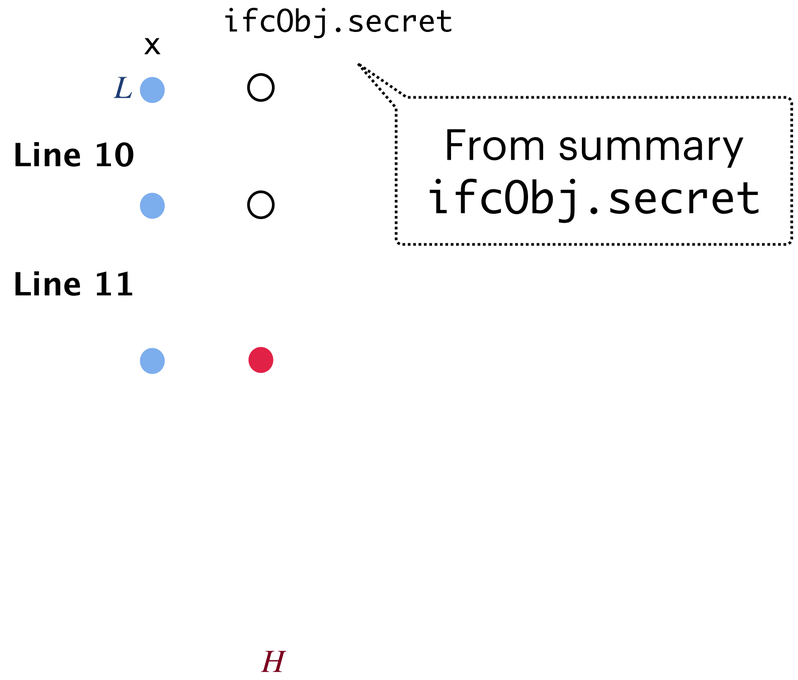
Backward Taint Analysis using access graphs from Phase 1

Analysis: Phase 3
Analyse Javascript Analysis : Forward Taint Analysis

```
 9 function function1() {
10     var x = api().getResult;
11     ifcObj.set(x); //integrity-violation
12 }
```

One of the parameters to the bridged interface is public?

ifcObj.secret = H

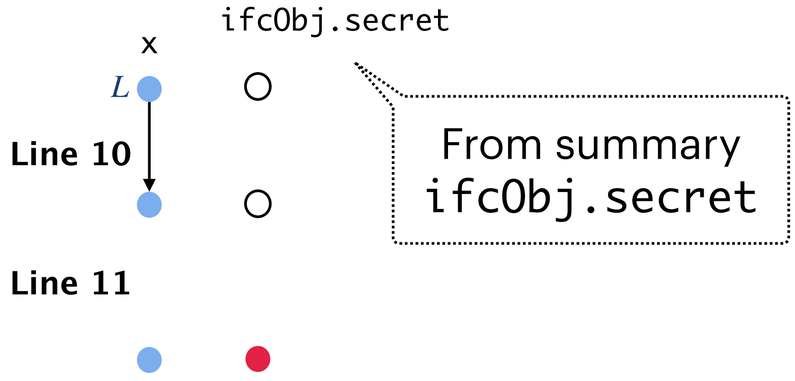


Analysis: Phase 3
Analyse Javascript Analysis : Forward Taint Analysis

```
 9 function function1() {
10     var x = api().getResult;
11     ifcObj.set(x); //integrity-violation
12 }
```

One of the parameters to the bridged interface is public?

ifcObj.secret = H



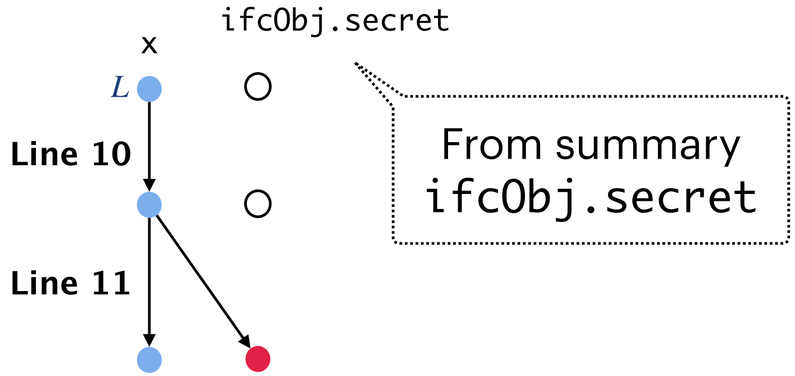
H

Analysis: Phase 3
Analyse Javascript Analysis : Forward Taint Analysis

```
 9 function function1() {
10     var x = api().getResult;
11     ifcObj.set(x); //integrity-violation
12 }
```

One of the parameters to the bridged interface is public?

ifcObj.secret = H



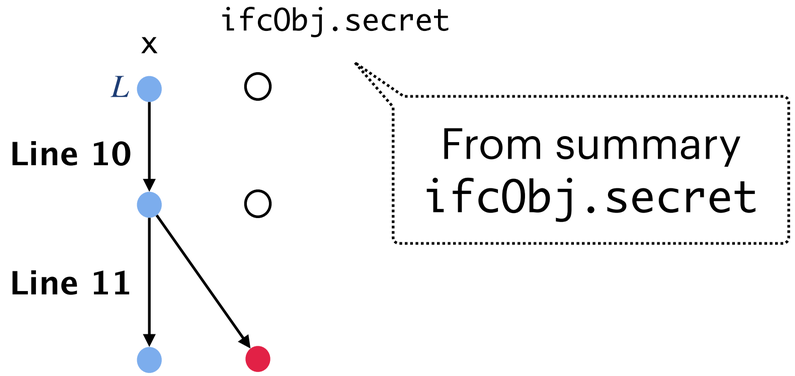
H

Analysis: Phase 3
Analyse Javascript Analysis : Forward Taint Analysis

```
 9 function function1() {
10     var x = api().getResult;
11     ifcObj.set(x); //integrity-violation
12 }
```

One of the parameters to the bridged interface is public?

ifcObj.secret = H




```
19 function function2() {
20     var x = ifcObj.get();
21     leak(x); //confidentiality-violation
22 }
```

Replace the summary information use it in taint analysis

H

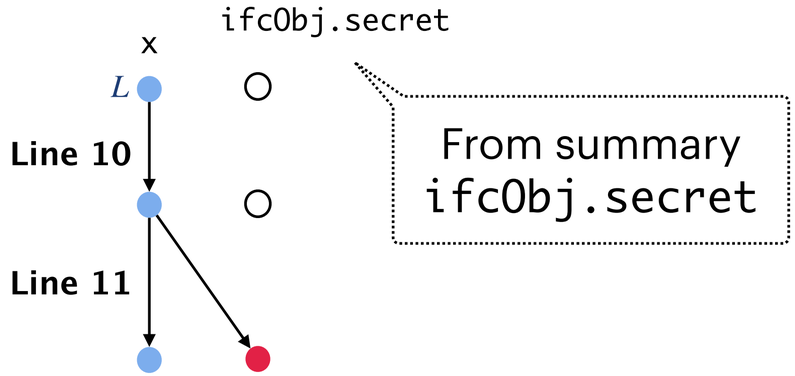
```
@JavaScriptInterface
get() {
    return this.secret;
}
```

Analysis: Phase 3
Analyse Javascript Analysis : Forward Taint Analysis

```
 9 function function1() {
10     var x = api().getResult;
11     ifcObj.set(x); //integrity-violation
12 }
```

One of the parameters to the bridged interface is public?

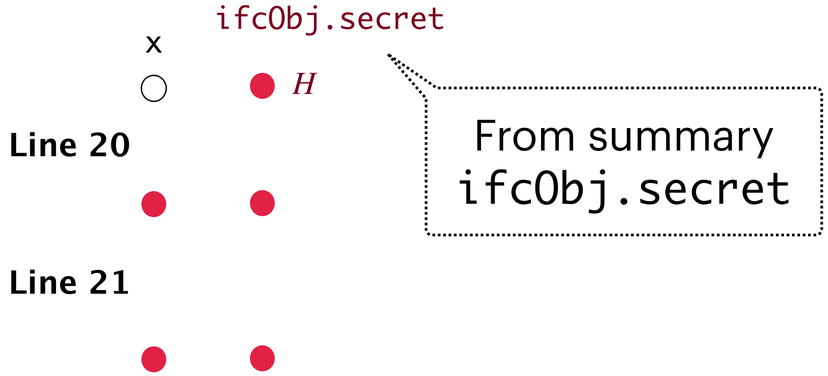
ifcObj.secret = H




```
19 function function2() {
20     var x = ifcObj.get();
21     leak(x); //confidentiality-violation
22 }
```

Replace the summary information use it in taint analysis

```
@JavaScriptInterface
get() {
    return this.secret;
}
```

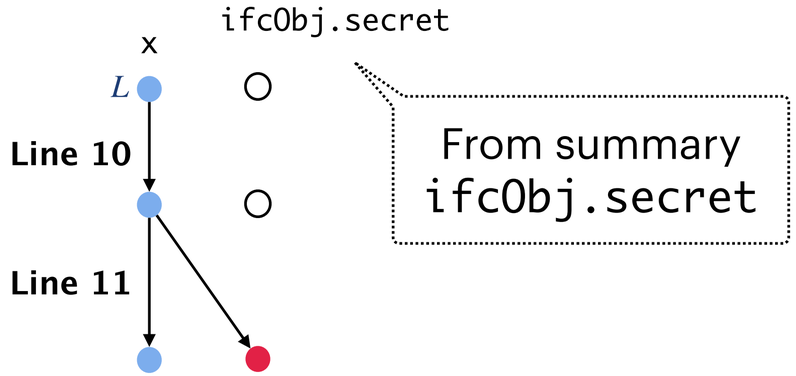


Analysis: Phase 3
Analyse Javascript Analysis : Forward Taint Analysis

```
 9 function function1() {
10     var x = api().getResult;
11     ifcObj.set(x); //integrity-violation
12 }
```

One of the parameters to the bridged interface is public?

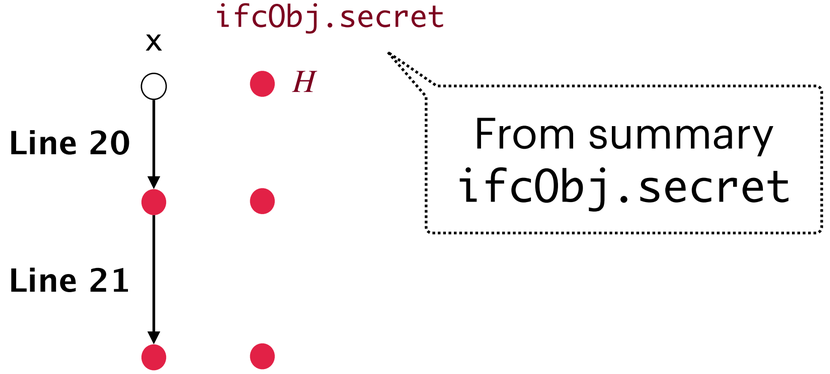
ifcObj.secret = H




```
19 function function2() {
20     var x = ifcObj.get();
21     leak(x); //confidentiality-violation
22 }
```

Replace the summary information use it in taint analysis

```
@JavaScriptInterface
get() {
    return this.secret;
}
```

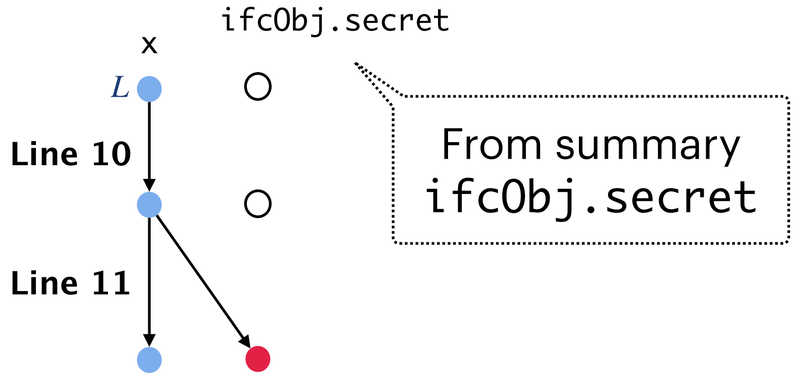


Analysis: Phase 3
Analyse Javascript Analysis : Forward Taint Analysis

```
 9 function function1() {
10     var x = api().getResult;
11     ifcObj.set(x); //integrity-violation
12 }
```

One of the parameters to the bridged interface is public?

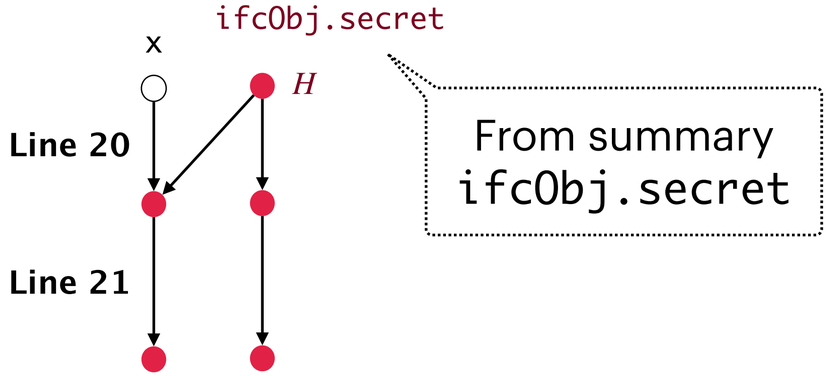
ifcObj.secret = H




```
19 function function2() {
20     var x = ifcObj.get();
21     leak(x); //confidentiality-violation
22 }
```

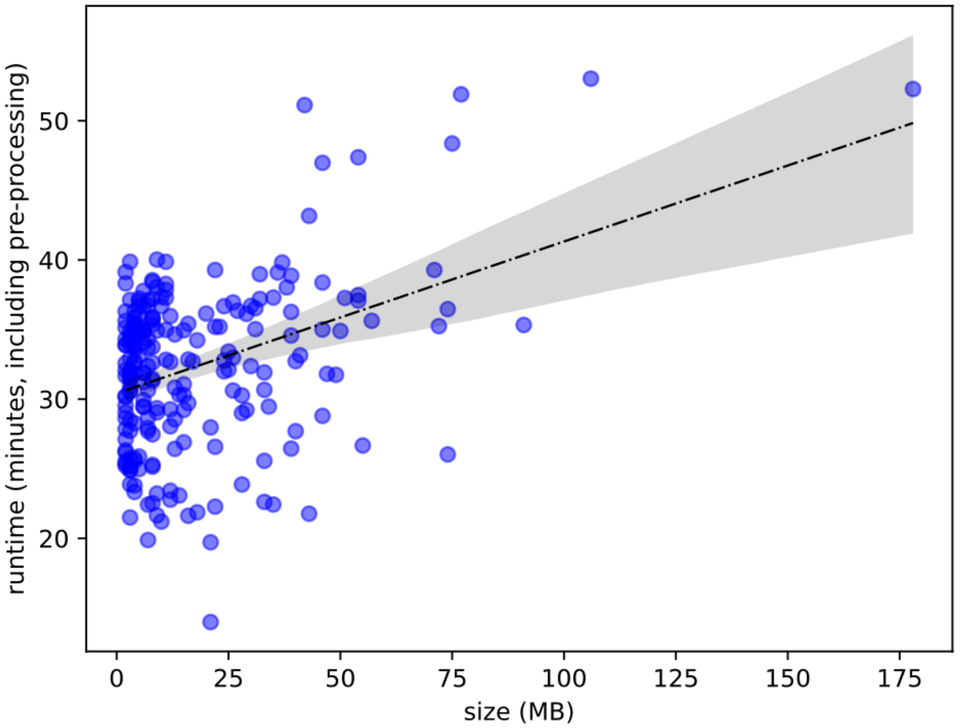
Replace the summary information use it in taint analysis

```
@JavaScriptInterface
get() {
    return this.secret;
}
```



Evaluation

Benchmark	Results
IWandDroid	Surpasses HybridDroid and LuDroid. **Analyses 19/23 (2 False Positives) applications** **IFC analysis is not possible with LuDroid and HybridDroid**
Apps from F-Droid	Detects around **123** integrity and confidentiality violations from a benchmark suite of 687 applications



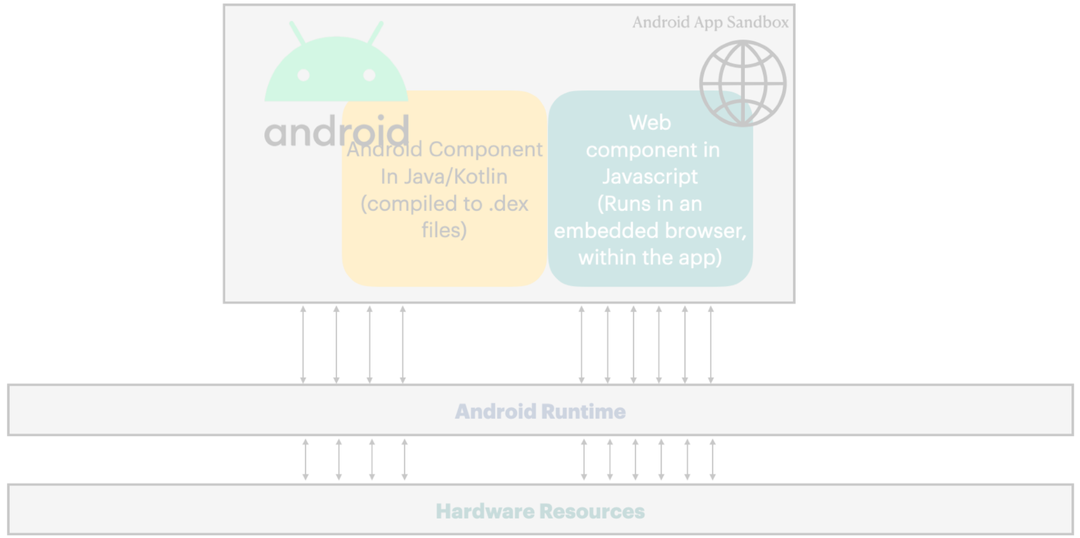
Runtime distribution over benchmark application

Reflections

How vulnerable is the communication in Android Hybrid Apps?

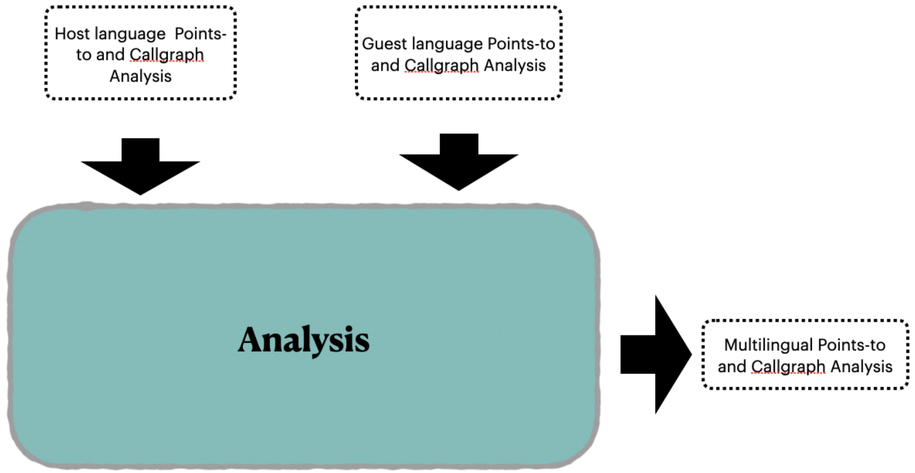
- Developed LuDroid to study the patterns in WebView
  - About 60% of the apps use WebView — interlanguage setters are a common pattern
- Defined a confidentiality and integrity threat model for Hybrid apps
  - Developed IWanDroid to detect the confidentiality and integrity violations

Objective of the Thesis



1

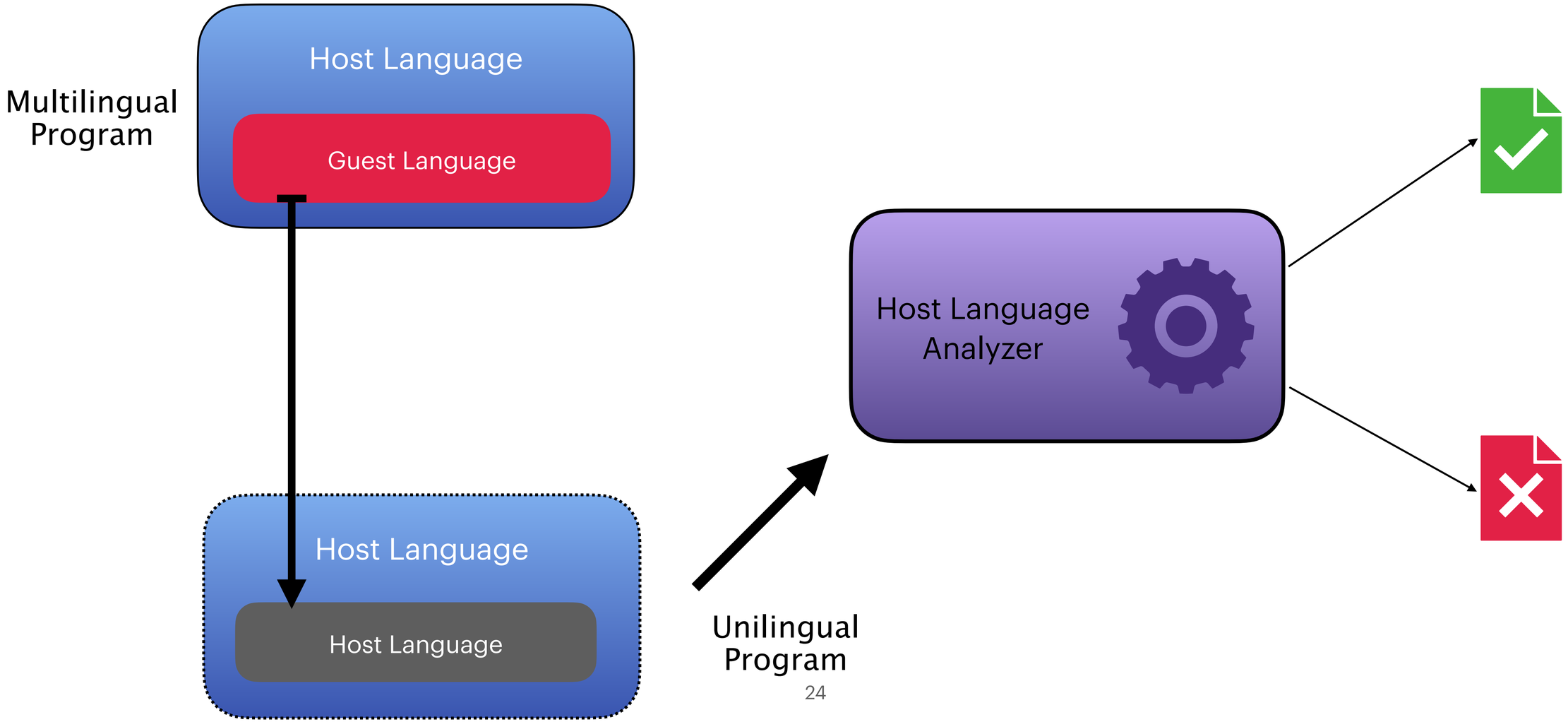
How vulnerable is the communication in Android Hybrid Apps?



2

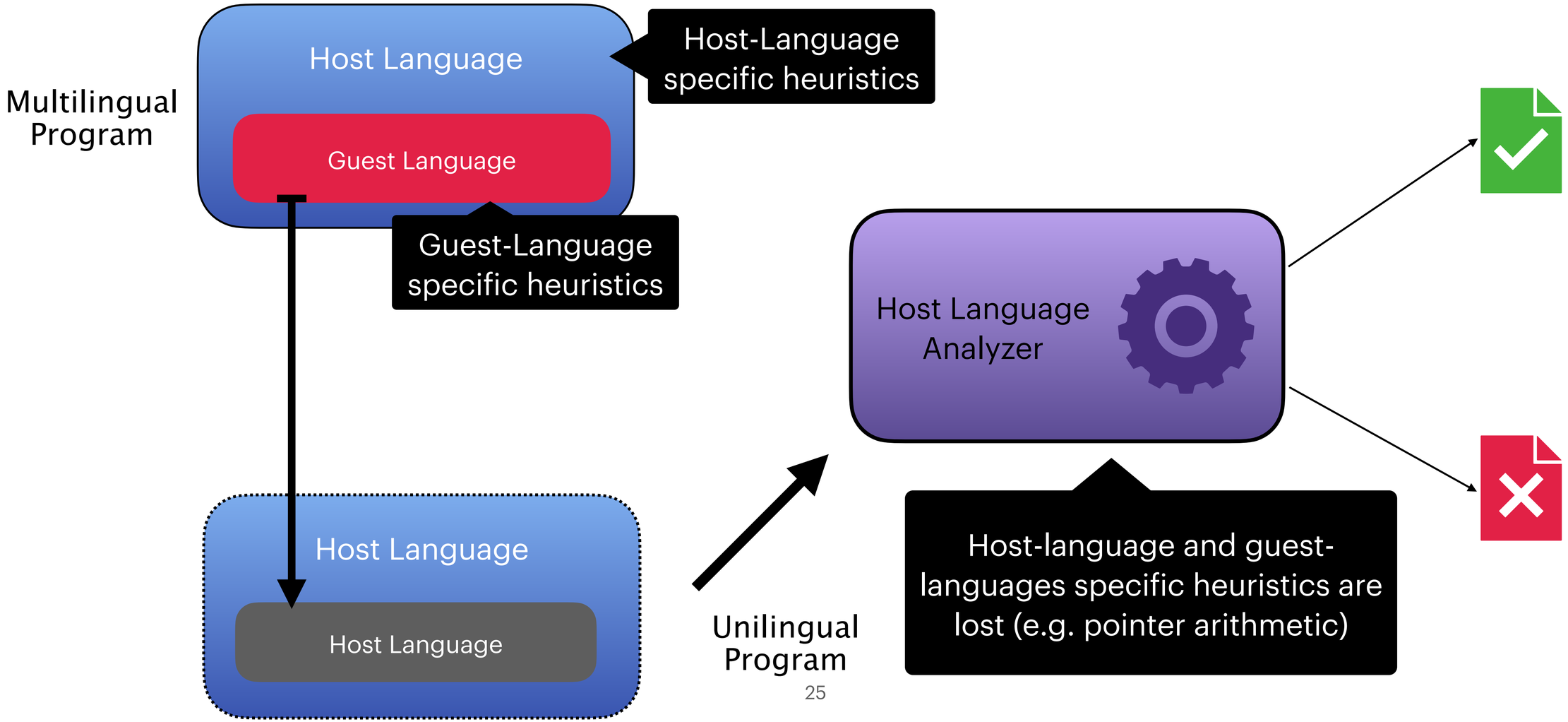
Can we combine monolingual static analyses for analysing multilingual applications?

Transform Guest Language Code to Host Language



Transform Guest Language Code to Host Language

Limitation

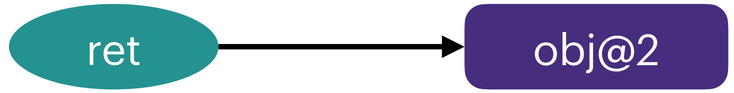


Points-to Analysis

- Static Analysis to compute the set of objects referred by the variables

```
1. Object mkalloc() {
2.     Object ret = new Object(); // obj@2
3.     return ret
4. }
```

$$\langle ret \rangle = \{obj@2\}$$

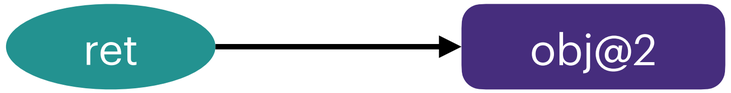


Points-to Analysis

- Static Analysis to compute the set of objects referred by the variables

```
1. Object mkalloc() {
2.    Object ret = new Object(); // obj@2
3.    return ret
4. }

5. main() {
6.   Object x = mkalloc(); //line 6
7.   Object y = mkalloc(); //line 7
8. }
```

$$\langle ret \rangle = \{obj@2\}$$



Points-to Analysis

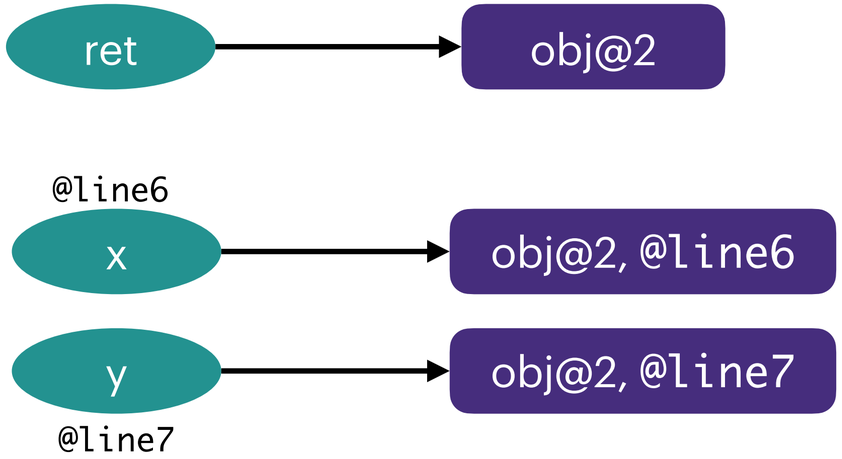
- Static Analysis to compute the set of objects referred by the variables

```
1. Object mkalloc() {
2.     Object ret = new Object(); // obj@2
3.     return ret
4. }

5. main() {
6.   Object x = mkalloc(); //line 6
7.   Object y = mkalloc(); //line 7
8. }
```

$$\langle ret \rangle = \{obj@2\}$$

$$\langle x \rangle = \{obj@2, @\texttt{line6}\}$$

$$\langle y \rangle = \{obj@2, @\texttt{line7}\}$$



Call-Graph Analysis

- Static analysis which computes the caller-callee relationships

```
1. class A {
2.     void func() {}
3. }

4. class B extends A {
5.     void func() {}
6. }

7. main() {
8.     A obj = null;
9.     obj = (condition) ? new A() : new B();
10.    obj.func();
11.}
```

Call-Graph Analysis

- Static analysis which computes the caller-callee relationships

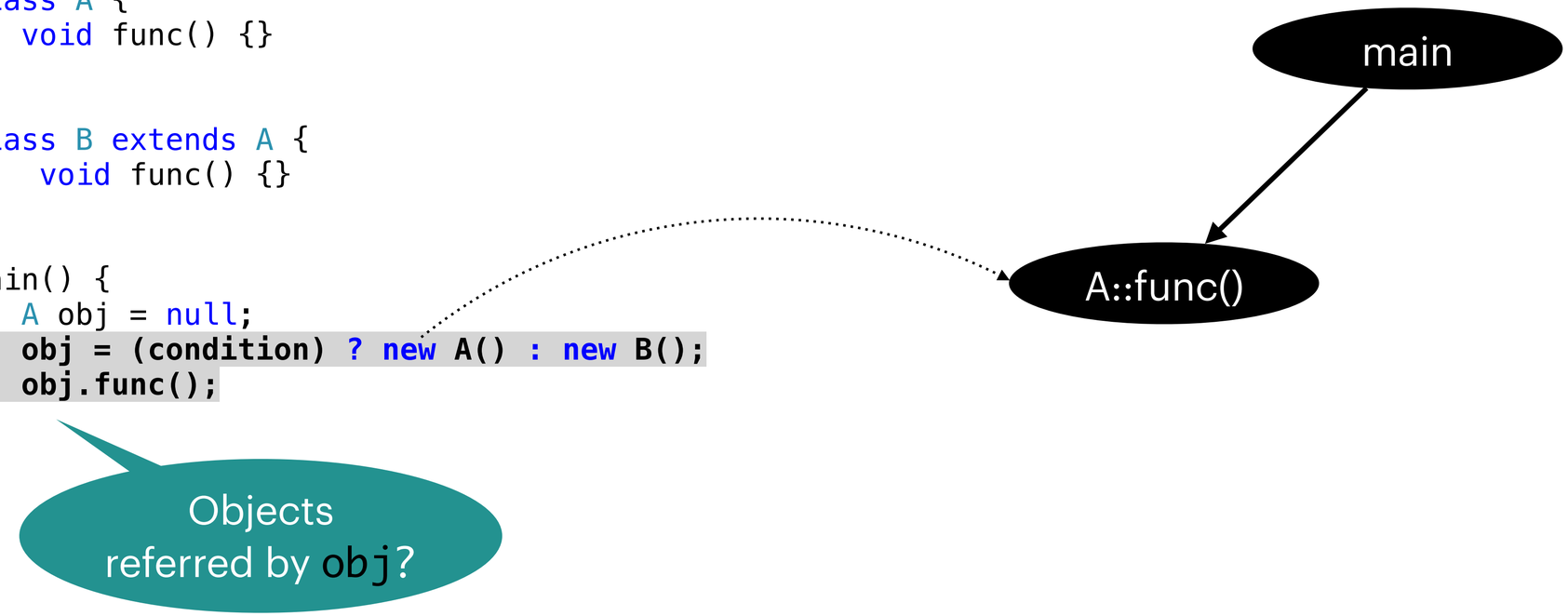
```
1. class A {
2.     void func() {}
3. }

4. class B extends A {
5.     void func() {}
6. }

7. main() {
8.     A obj = null;
9.     obj = (condition) ? new A() : new B();
10.    obj.func();
11.}
```

Objects referred by `obj`?

Call-Graph Analysis

- Static analysis which computes the caller-callee relationships

```
1. class A {
2.     void func() {}
3. }

4. class B extends A {
5.     void func() {}
6. }

7. main() {
8.     A obj = null;
9.     obj = (condition) ? new A() : new B();
10.    obj.func();
11.}
```

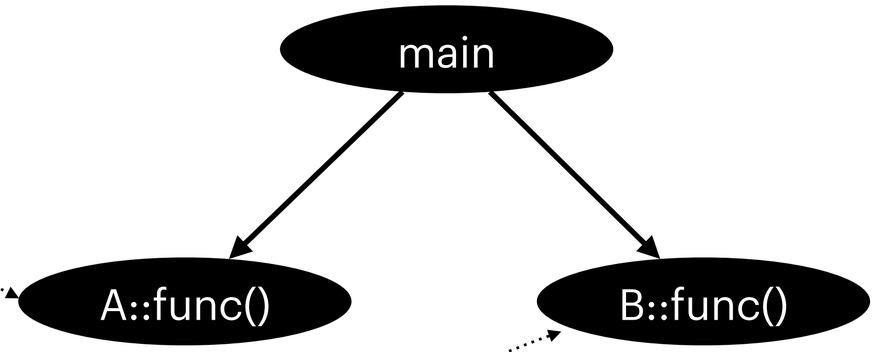


Call-Graph Analysis

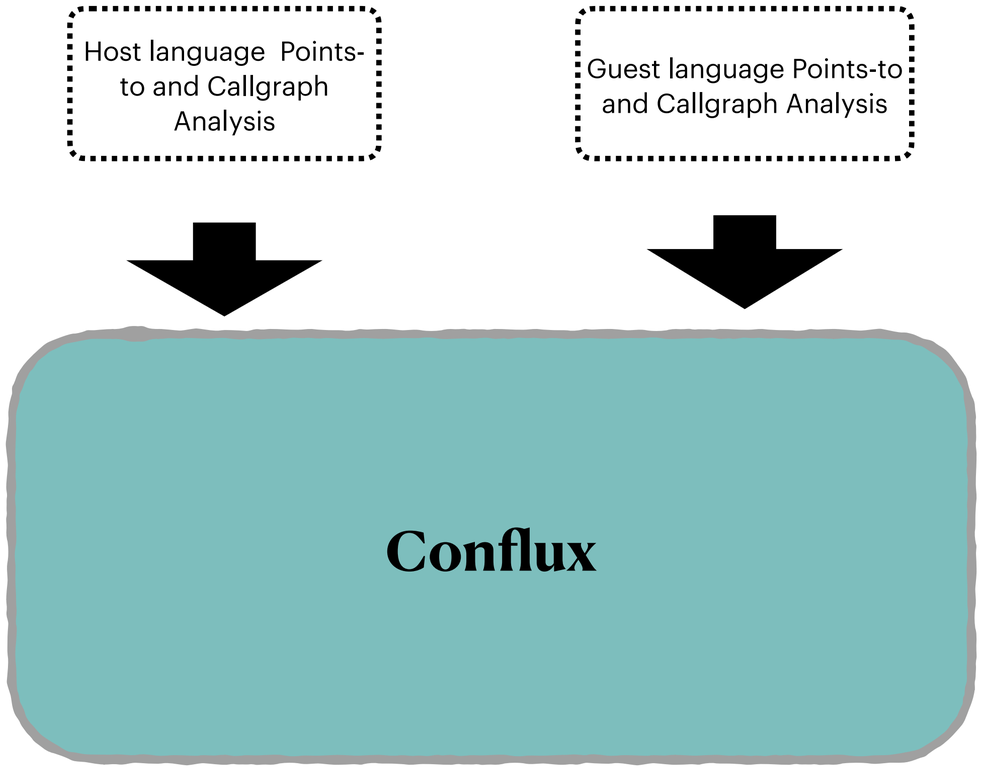
- Static analysis which computes the caller-callee relationships

```
1. class A {
2.     void func() {}
3. }

4. class B extends A {
5.     void func() {}
6. }

7. main() {
8.     A obj = null;
9.     obj = (condition) ? new A() : new B();
10.    obj.func();
11.}
```

Objects referred by `obj`?



Combining Points-to and Callgraph Analyses

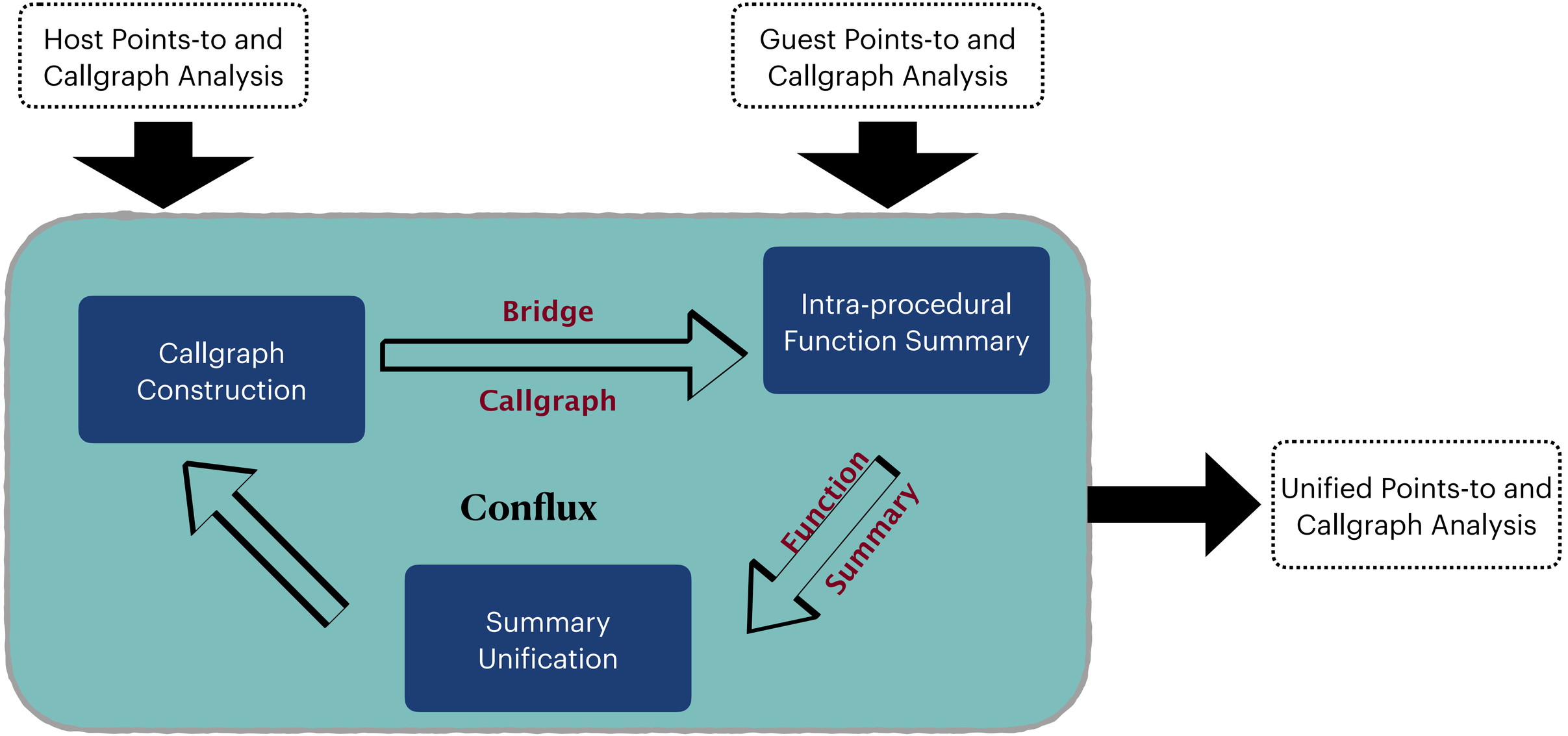


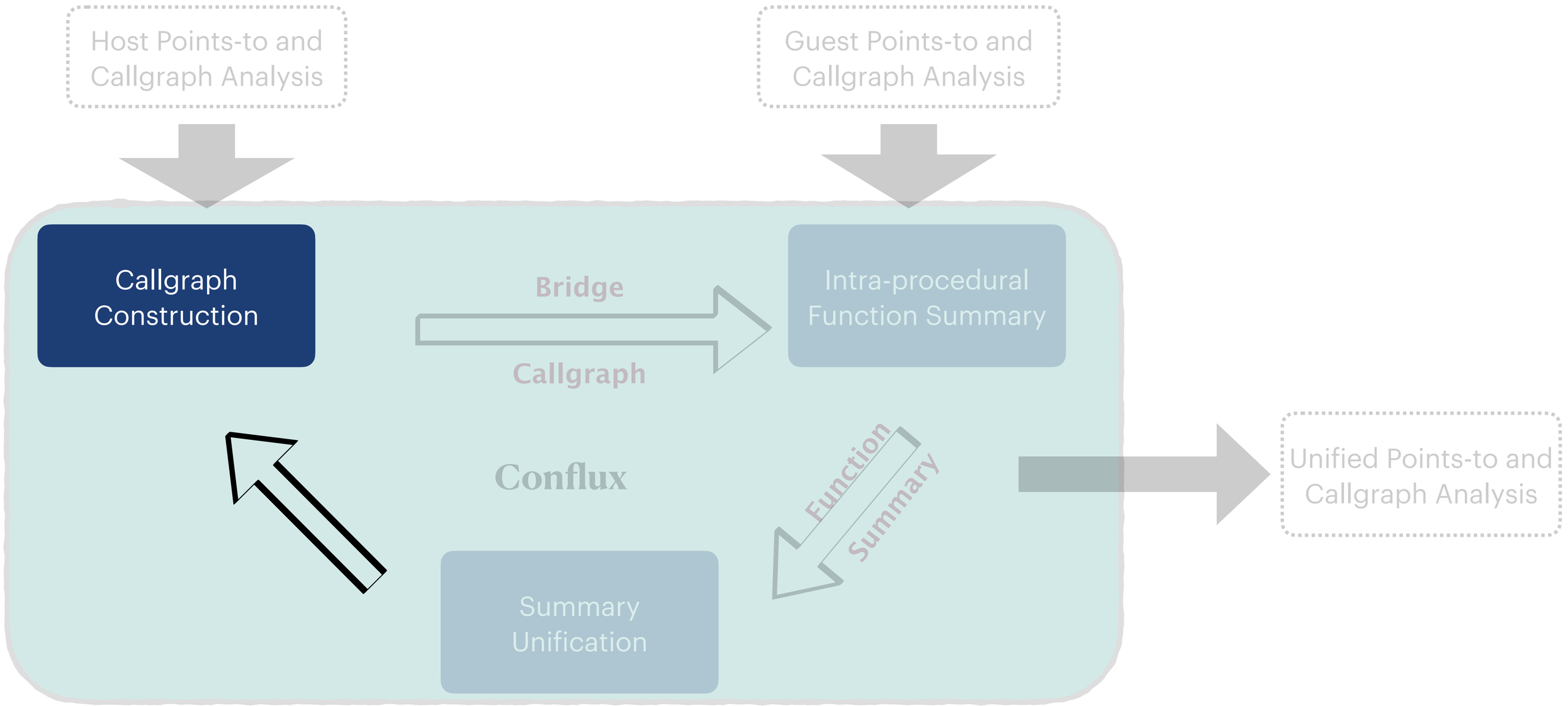
Technique

Exchange

Function Summaries

Conflux: Combining Static Analyses

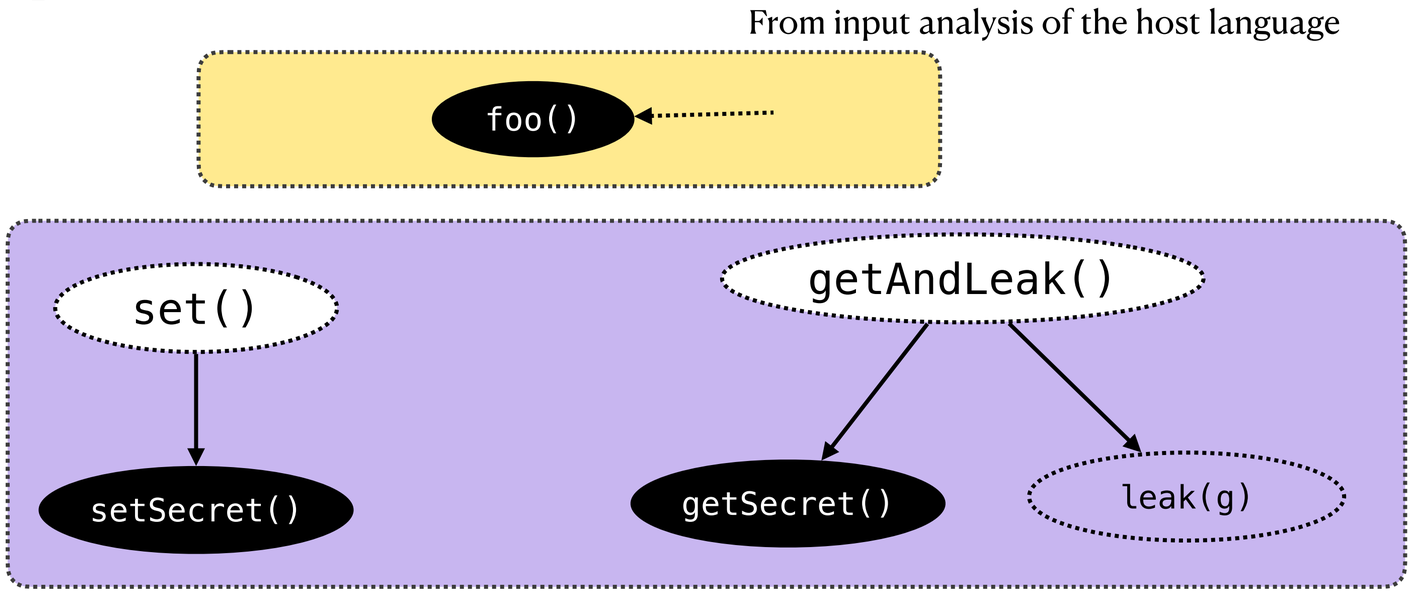



Host Points-to and Callgraph Analysis
Guest Points-to and Callgraph Analysis
Callgraph Construction
Bridge
Callgraph
Intra-procedural Function Summary
Conflux
Function Summary
Summary Unification
Unified Points-to and Callgraph Analysis

CallGraph Construction

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

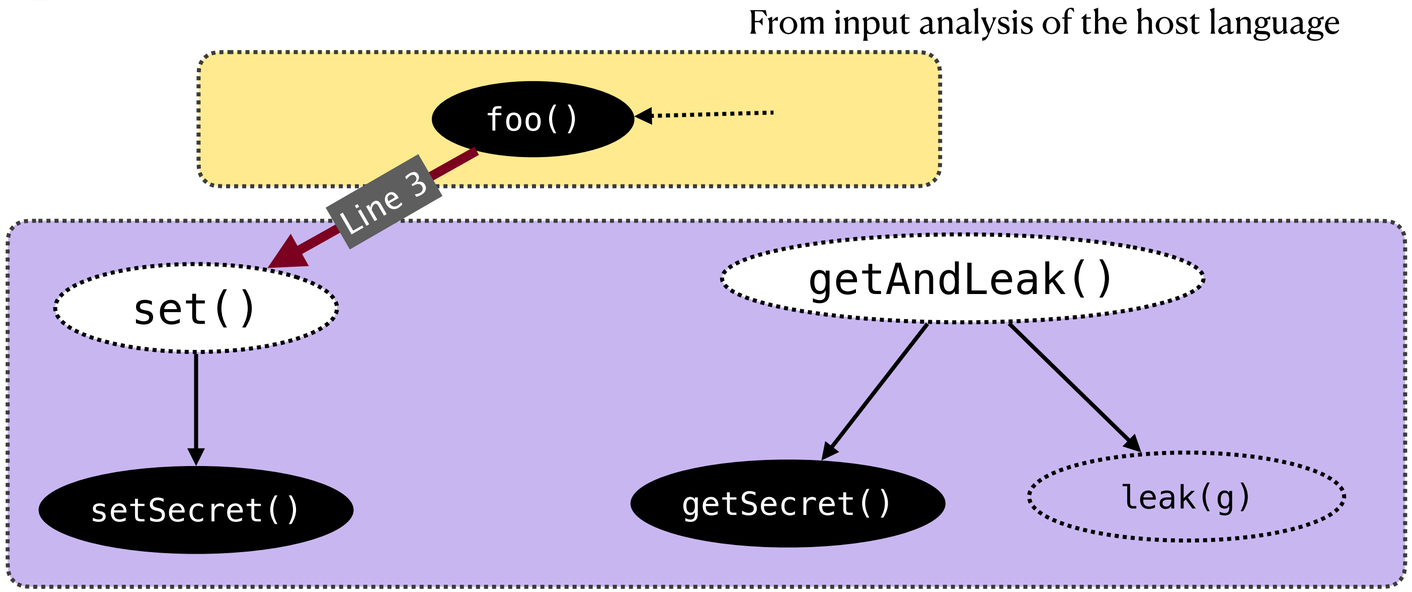
```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }

24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```



CallGraph Construction

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

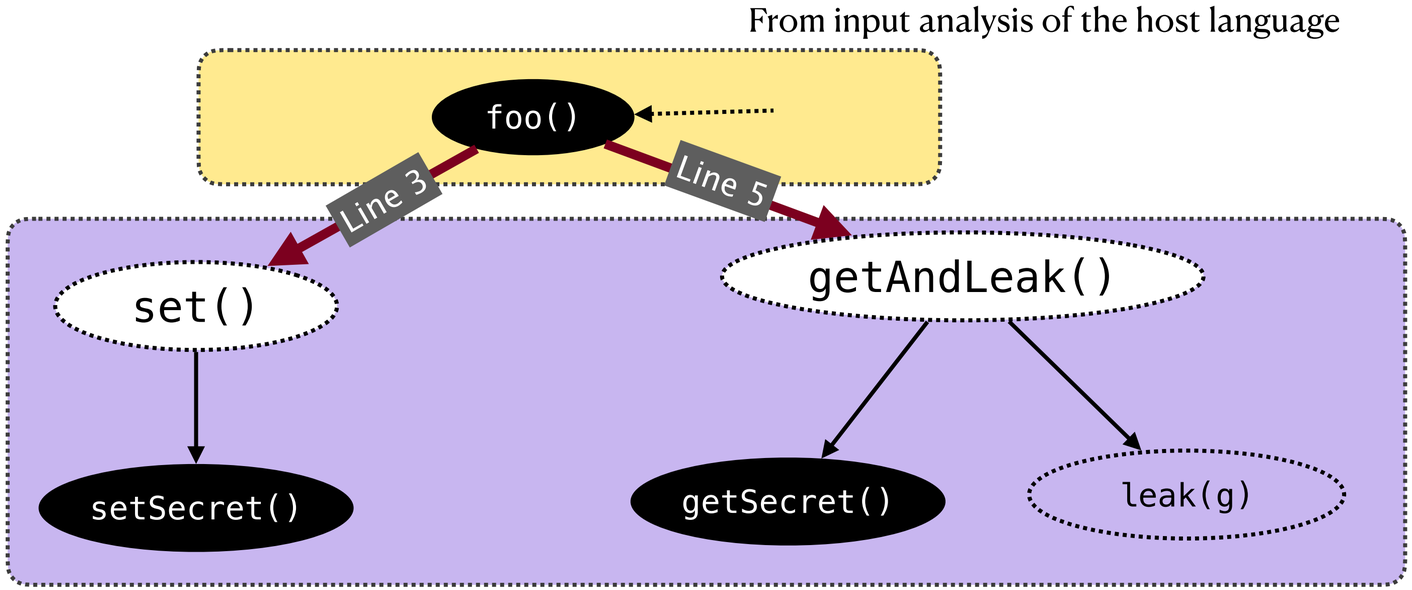
```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }

24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```



CallGraph Construction

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }

24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```

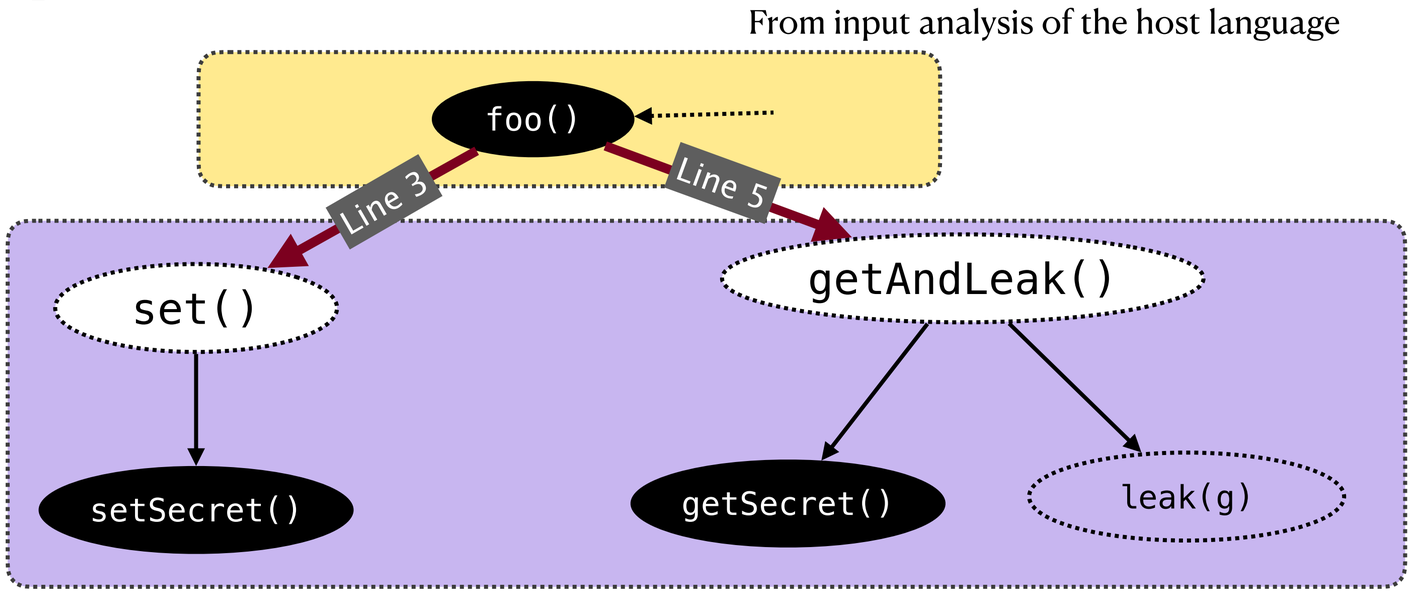


CallGraph Construction

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

Call between host methods through a guest method

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }

24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```



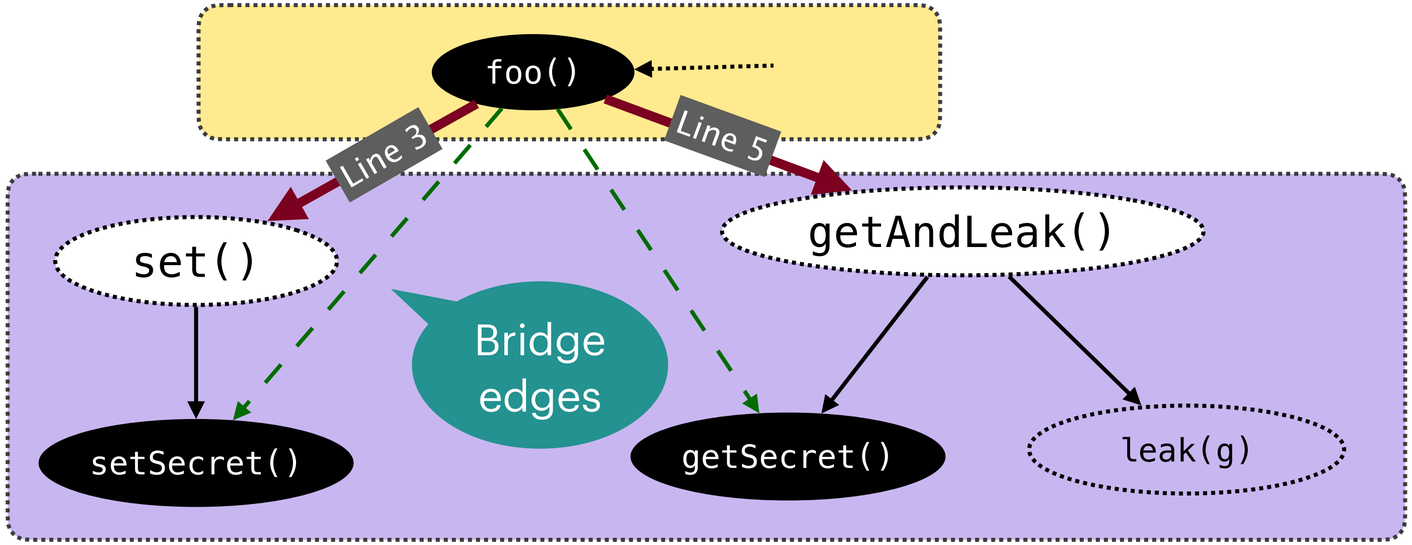
CallGraph Construction

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

Call between host methods through a guest method

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }

24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```

From input analysis of the host language



From input analysis of the guest language

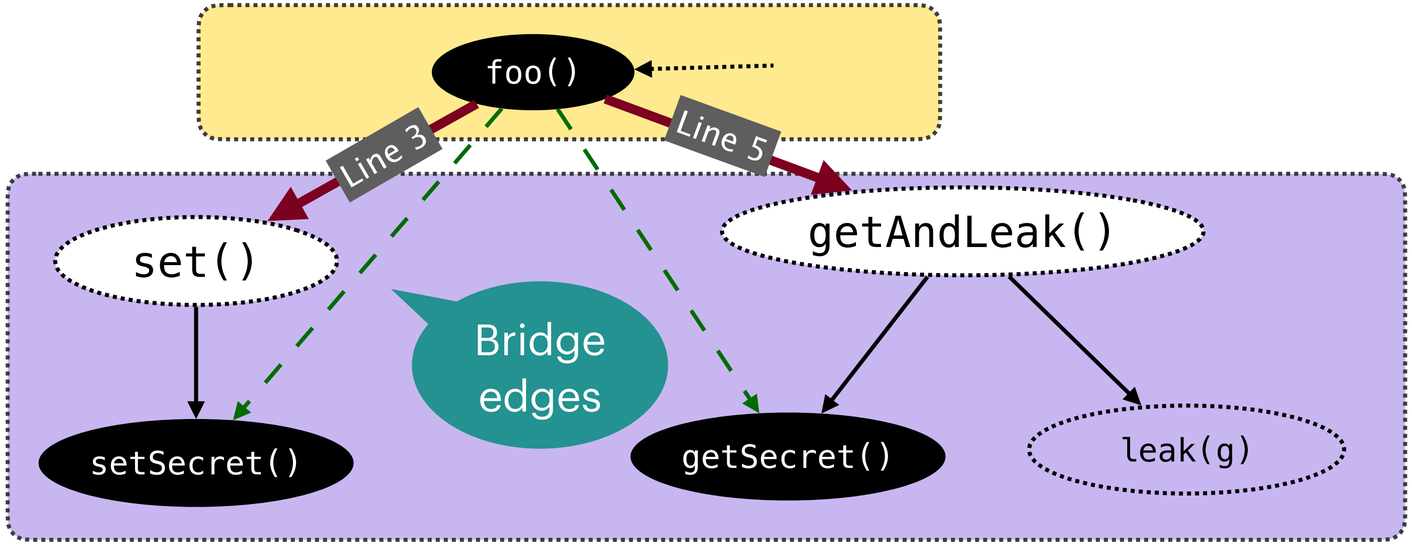
CallGraph Construction

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

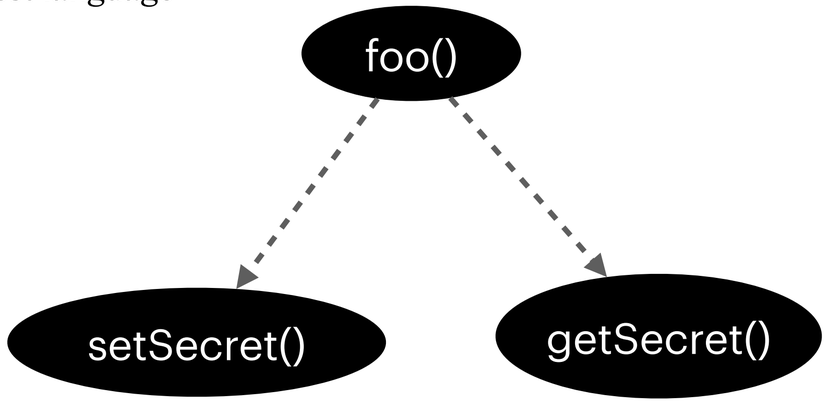
Call between host methods through a guest method

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }

24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```

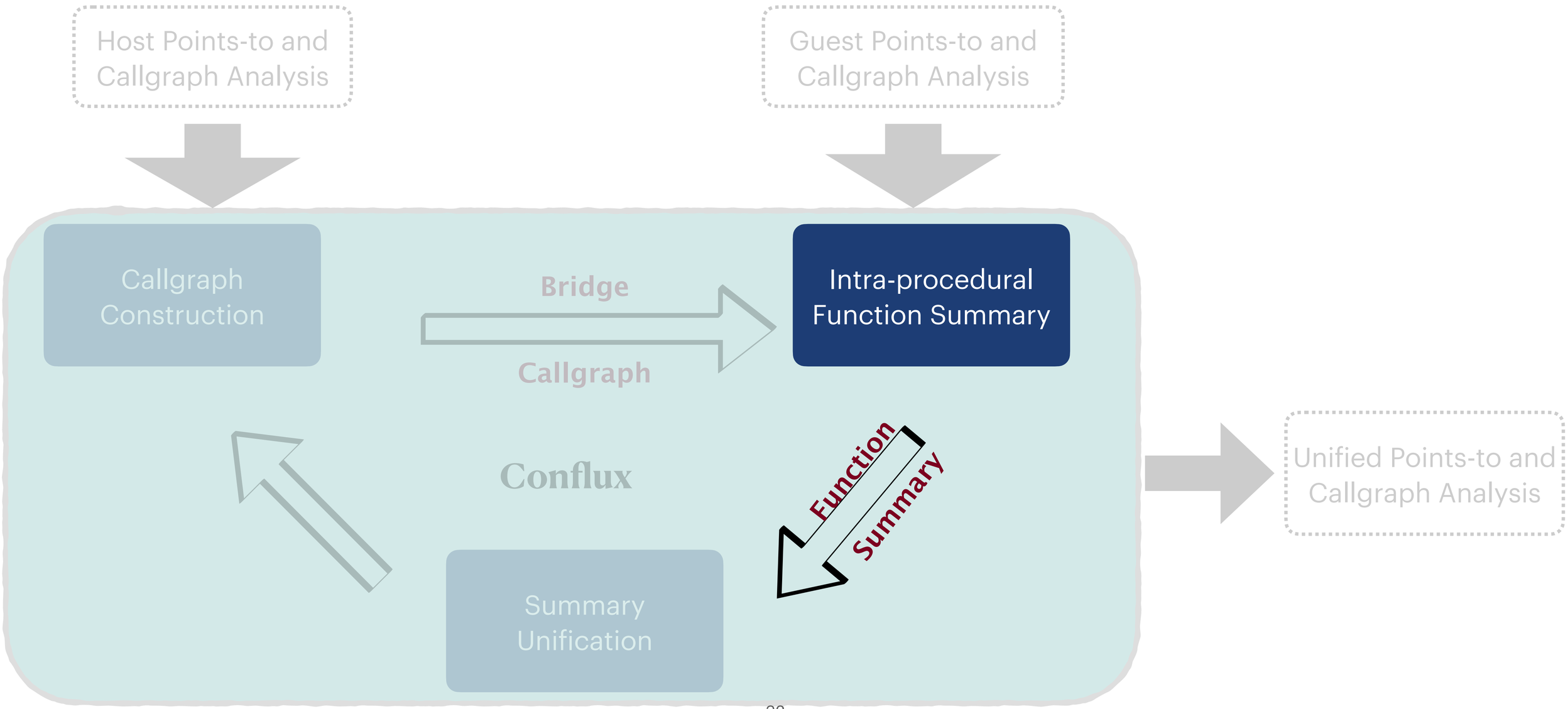
From input analysis of the host language



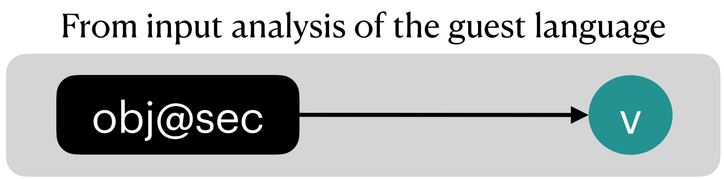
From input analysis of the guest language



Used in the next stages to resolve the interlanguage changes to the interface variable


Host Points-to and Callgraph Analysis
Guest Points-to and Callgraph Analysis
Callgraph Construction
Bridge
Callgraph
Intra-procedural Function Summary
Conflux
Function Summary
Summary Unification
Unified Points-to and Callgraph Analysis

Collect and resolve constraints for bridge variables



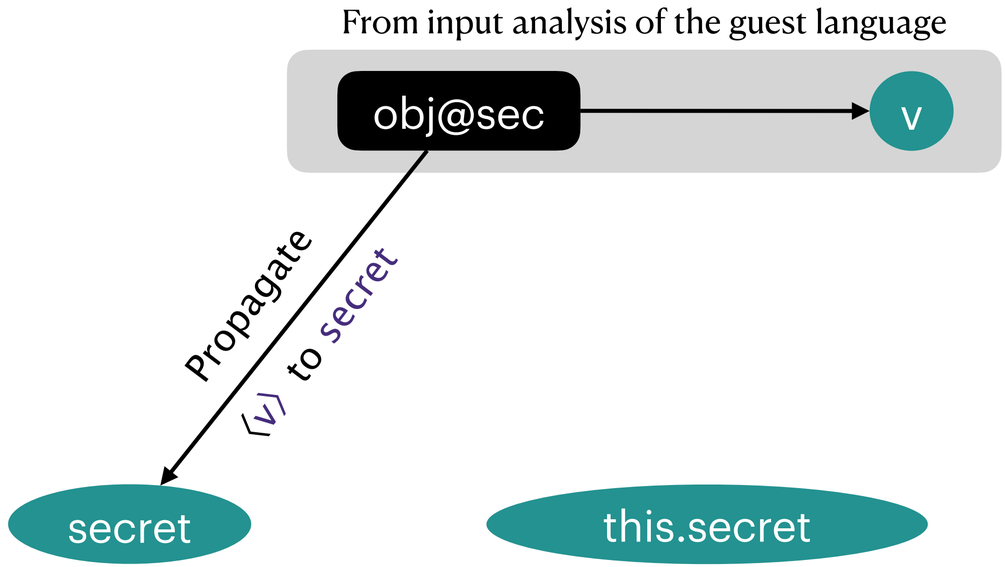

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }
```

```
 9 BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }
```

Collect and resolve constraints for bridge variables

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }
```

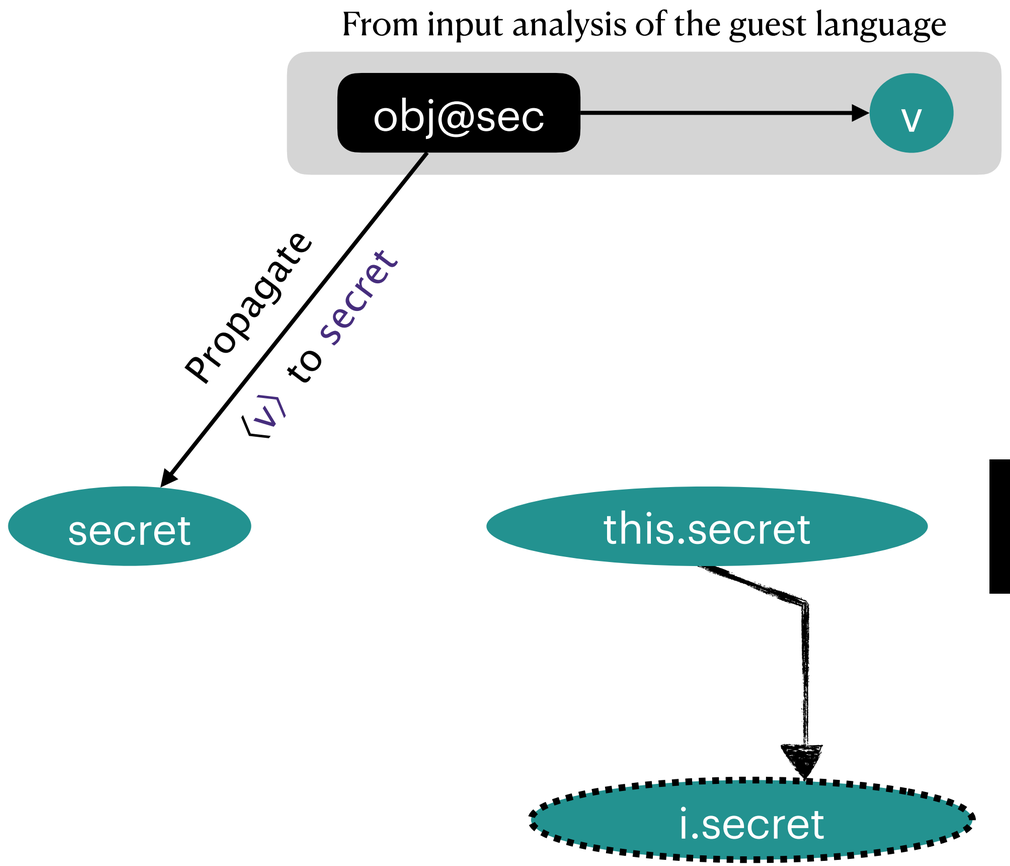
```
 9 BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }
```



Collect and resolve constraints for bridge variables

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }
```

```
 9 BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }
```

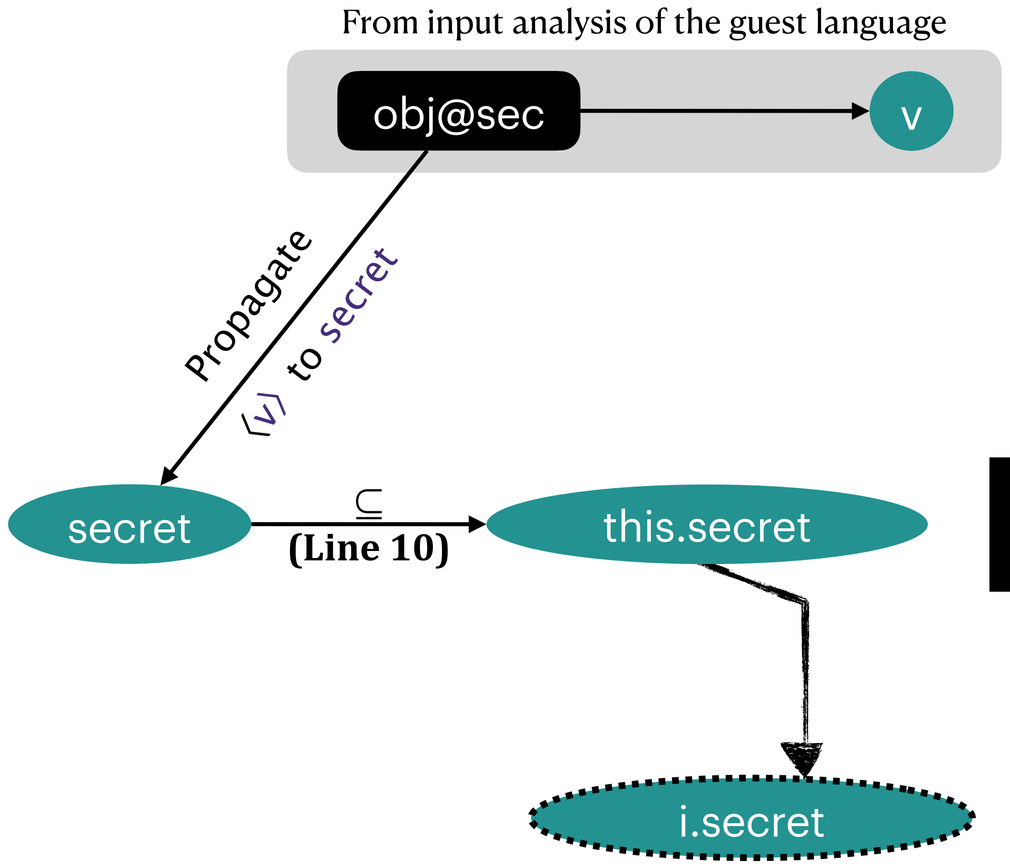


Replace it with the bridged variable

Collect and resolve constraints for bridge variables

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }
```

```
 9 BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }
```

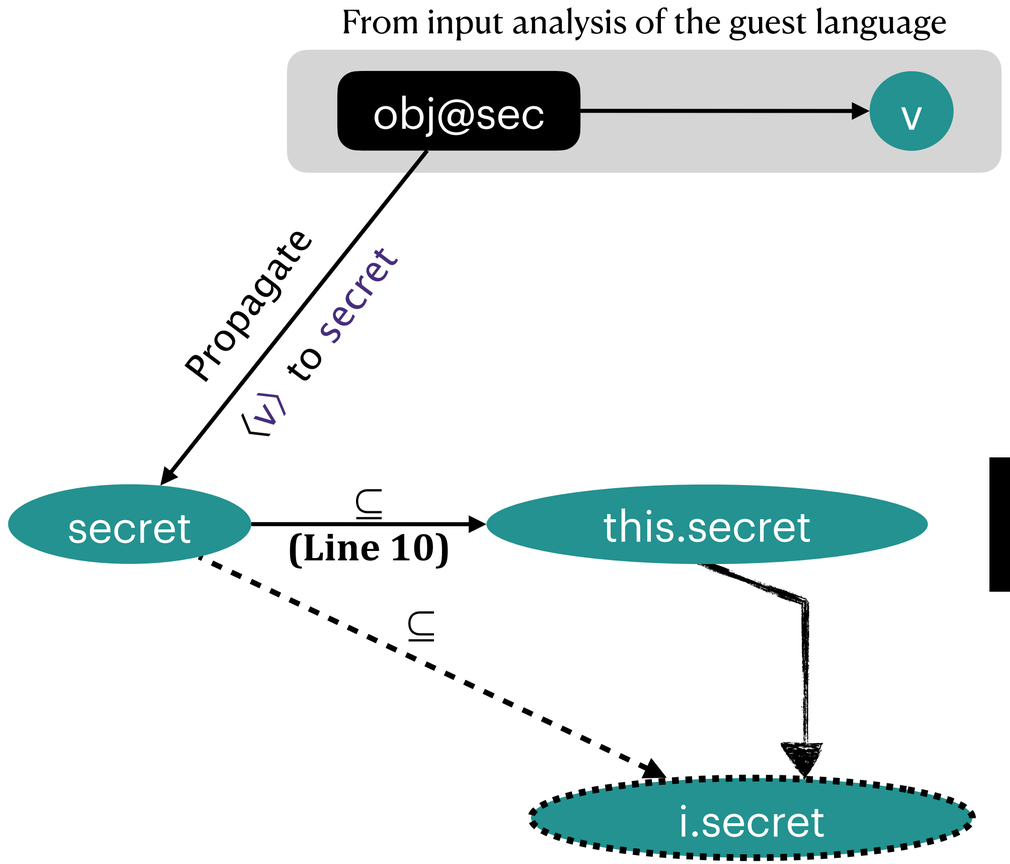


Replace it with the bridged variable

Collect and resolve constraints for bridge variables

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }
```

```
 9 BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }
```

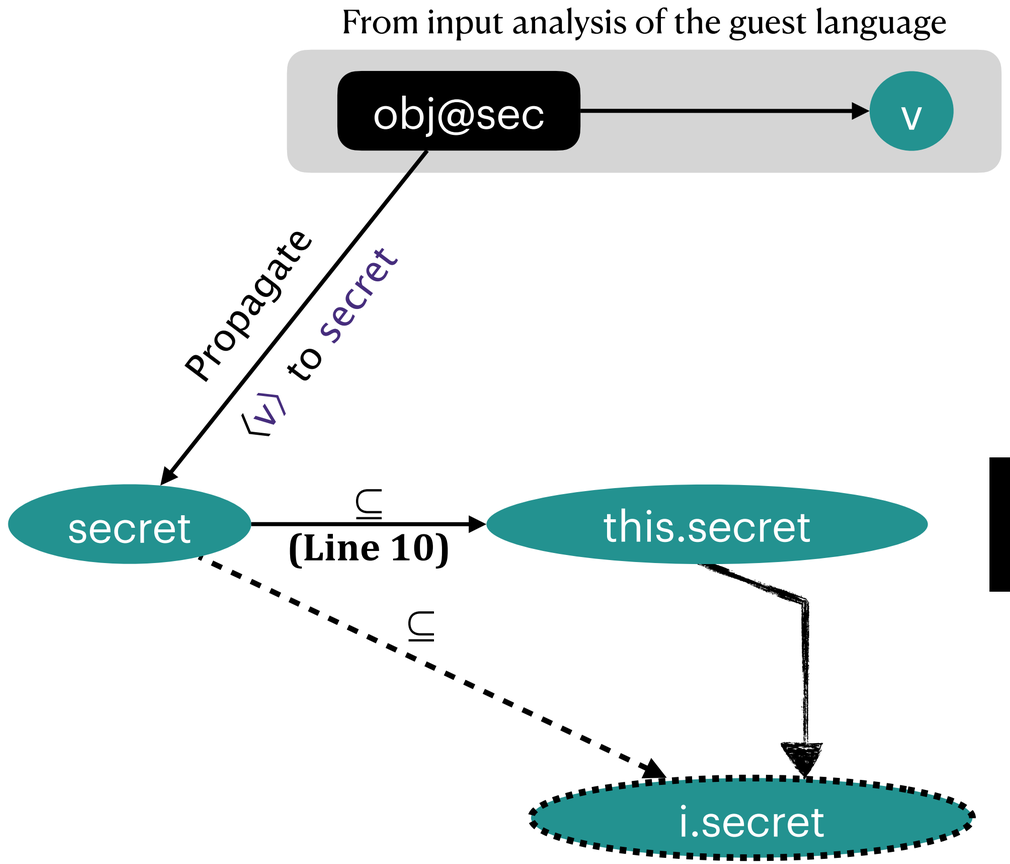


Replace it with the bridged variable

Collect and resolve constraints for bridge variables

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }
```

```
 9 BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }
```



Replace it with the bridged variable

$\langle \text{i.secret} \rangle \mapsto \{\text{obj@sec}\}$

Summary Specialisation

Resolve intra-procedural function summaries

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }

24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```

Function	Constraint	Function Summary
foo	(2) $\{obj@iface\} \in [i]$ (5) $\{\} \subseteq [x]$	$\langle i \rangle \mapsto \{obj@iface\}$ $\langle x \rangle \mapsto \{\}$

Summary Specialisation

Resolve intra-procedural function summaries

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }

19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }

24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```

Function	Constraint	Function Summary
foo	(2) $\{obj@iface\} \in [i]$ (5) $\{\} \subseteq [x]$	$\langle i \rangle \mapsto \{obj@iface\}$ $\langle x \rangle \mapsto \{\}$

Has not been resolved

Summary Specialisation

Resolve intra-procedural function summaries

```
 1 foo() {
 2     i = new BrigdeClass(); // obj@iface
 3     eval("set()"); // invoke the guest function
 4     // do some computation
 5     x = eval("getAndLeak()");
 6     return x;
 7 }

 9 BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }

24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```

Function	Constraint	Function Summary
foo	(2) {obj@iface} ∈ [i] (5) { } ⊆ [x]	⟨i⟩ ↦ {obj@iface} ⟨x⟩ ↦ { }
setSecret	(10) {obj@sec} ∈ [secret] (Spec) [secret] ⊆ [i.secret]	⟨i.secret⟩ ↦ {obj@sec}

Summary Specialisation

Resolve intra-procedural function summaries

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }

24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```

Function	Constraint	Function Summary
foo	(2) {obj@iface} ∈ [i] (5) { } ⊆ [x]	⟨i⟩ ↦ {obj@iface} ⟨x⟩ ↦ { }
setSecret	(10) {obj@sec} ∈ [secret] (Spec) [secret] ⊆ [i.secret]	⟨i.secret⟩ ↦ {obj@sec}
getSecret	(14) [i.secret] ⊆ [$ret]	⟨i.secret⟩ ↦ { } ⟨$ret⟩ ↦ { }

Summary Specialisation

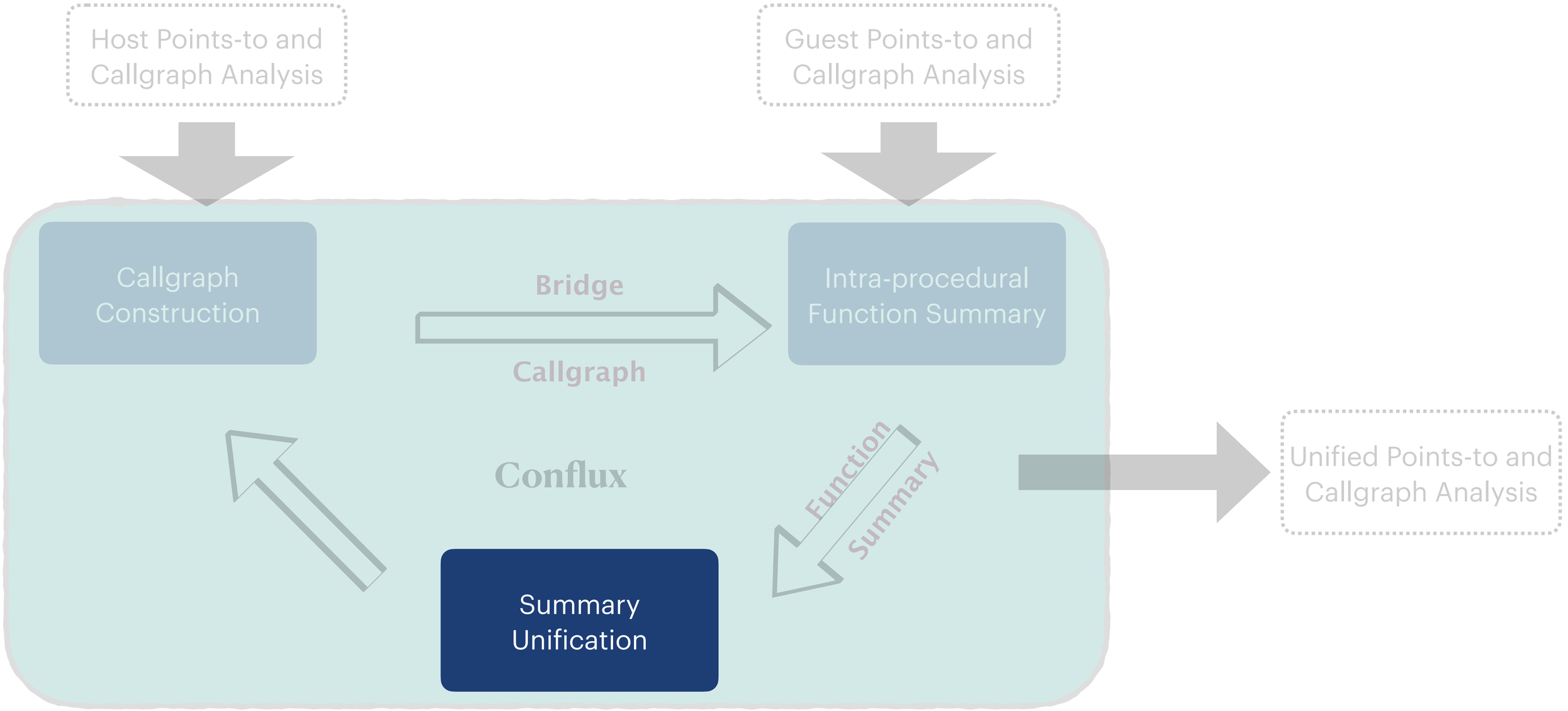
Resolve intra-procedural function summaries

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }

24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```

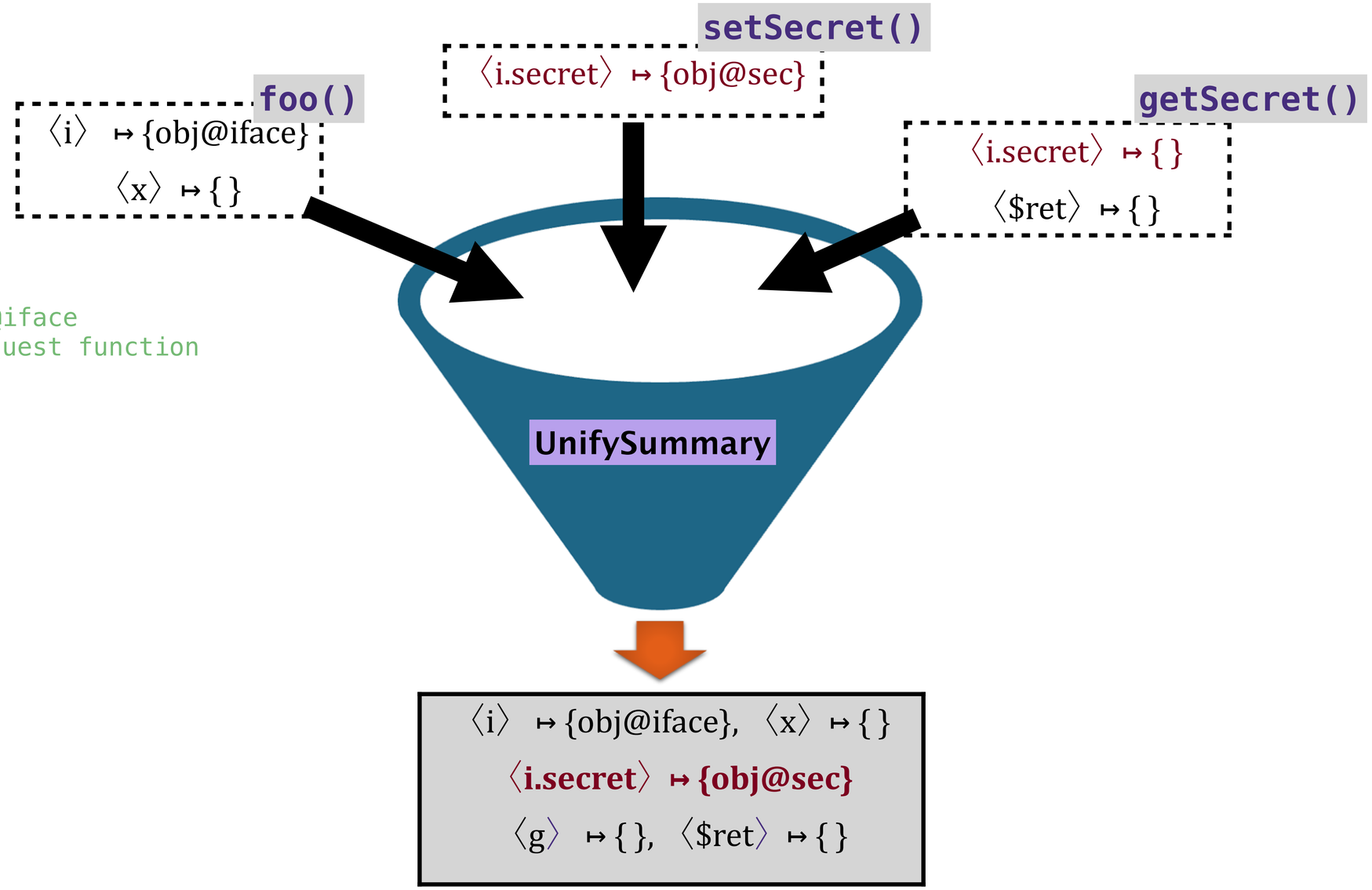
Function	Constraint	Function Summary
foo	(2) {obj@iface} ∈ [i] (5) { } ⊆ [x]	⟨i⟩ ↦ {obj@iface} ⟨x⟩ ↦ { }
setSecret	(10) {obj@sec} ∈ [secret] (Spec) [secret] ⊆ [i.secret]	⟨i.secret⟩ ↦ {obj@sec}
getSecret	(14) [i.secret] ⊆ [$ret]	⟨i.secret⟩ ↦ { } ⟨$ret⟩ ↦ { }

Has not been resolved


Host Points-to and Callgraph Analysis
Guest Points-to and Callgraph Analysis
Callgraph Construction
Bridge
Callgraph
Intra-procedural Function Summary
Conflux
Function Summary
Summary Unification
Unified Points-to and Callgraph Analysis

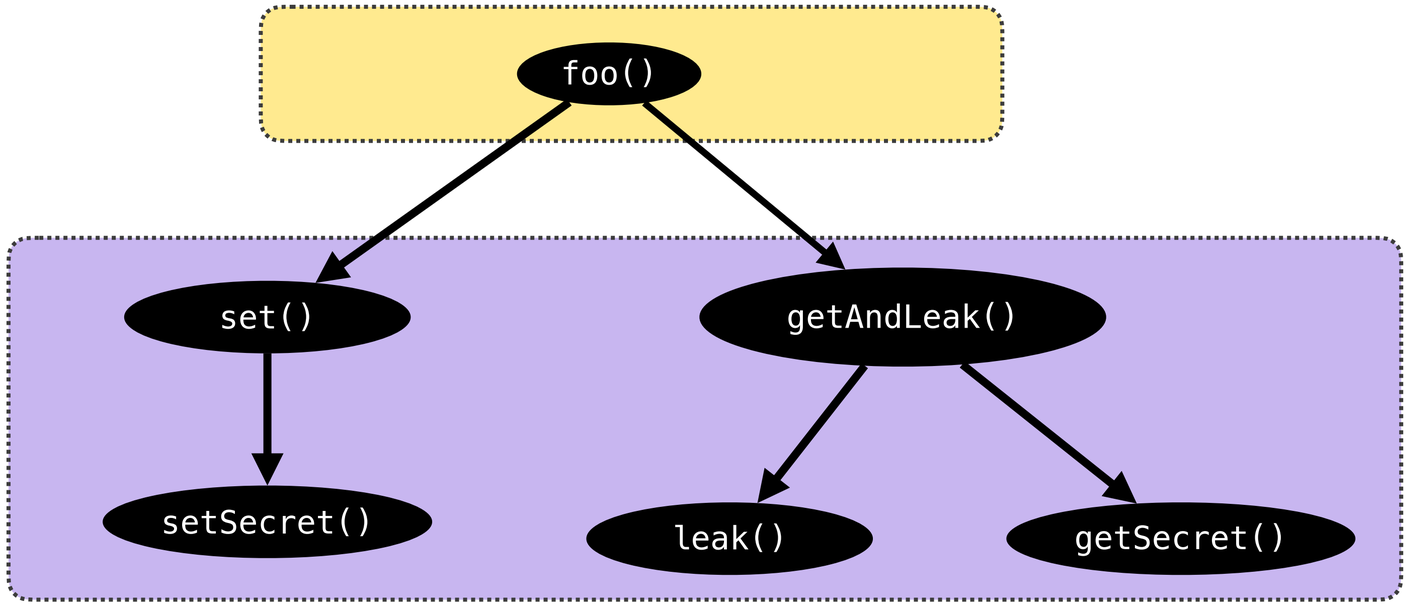
Summary Unification

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```



Summary Injection

```
1  foo() {
2      i = new BrigdeClass(); // obj@iface
3      eval("set()"); // invoke the guest function
4      // do some computation
5      x = eval("getAndLeak()");
6      return x;
7  }

9  BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }

13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```



Resolved Interlanguage call-graph

Summary Injection

For `set()` **and** `setSecret()`

```
 9 BrigdeClass::setSecret(secret) {
10     this.secret = secret;
11 }
```

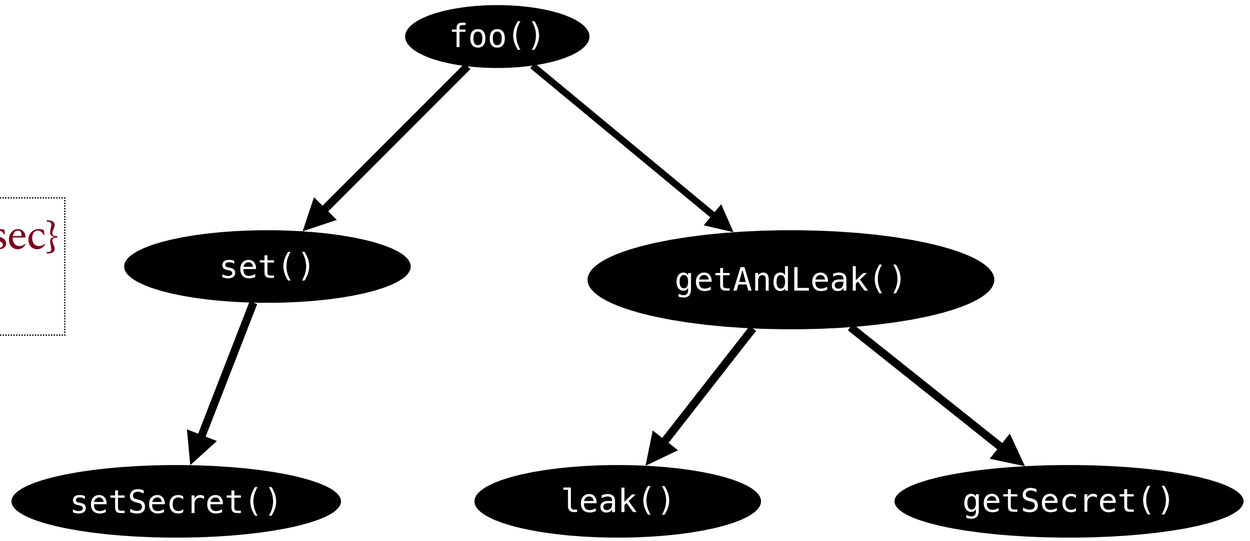
```
19 set() {
20     v = new Secret(); // obj@sec
21     i.setSecret(v);
22 }
```

⟨i, *secret*⟩ ↦ {obj@sec}

⟨v⟩ ↦ {obj@sec}

Resolved

⟨i, *secret*⟩ ↦ {obj@sec}

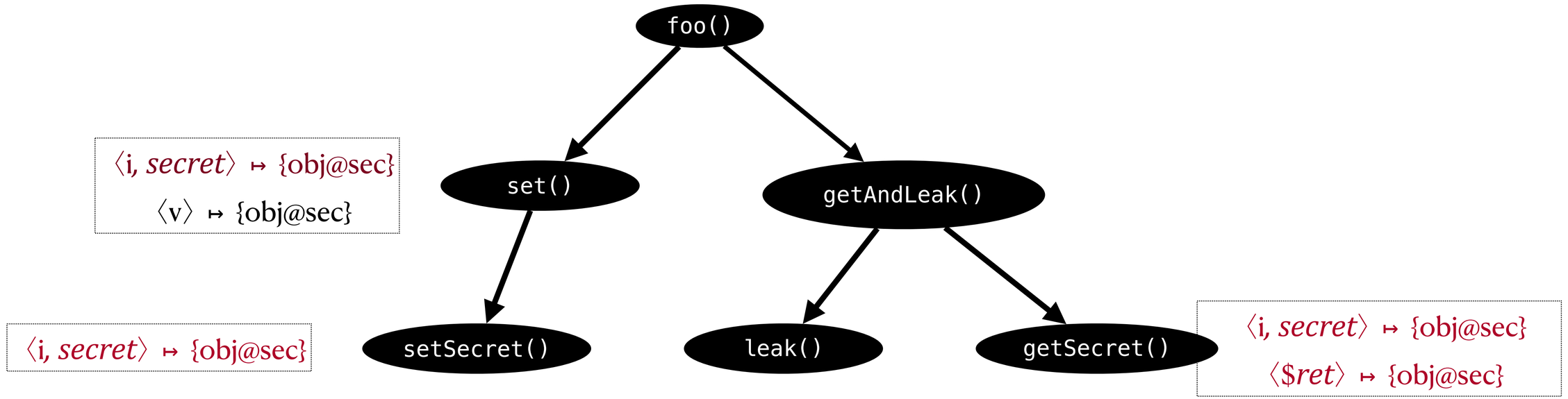


Summary Injection

For getAndLeak() **and** getSecret()

```
13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

```
24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```

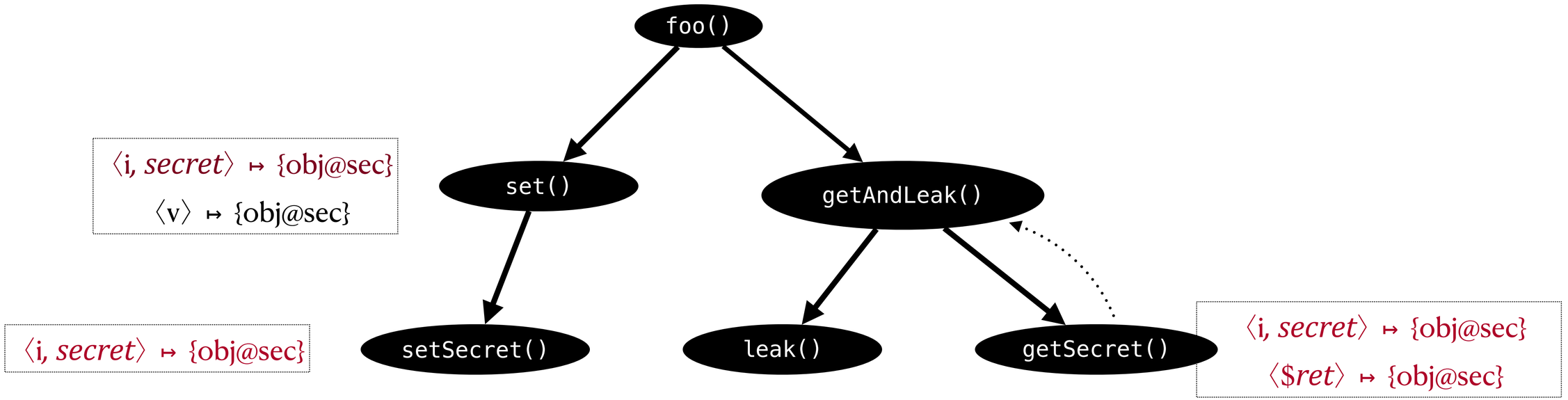


Summary Injection

For getAndLeak() **and** getSecret()

```
13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

```
24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```

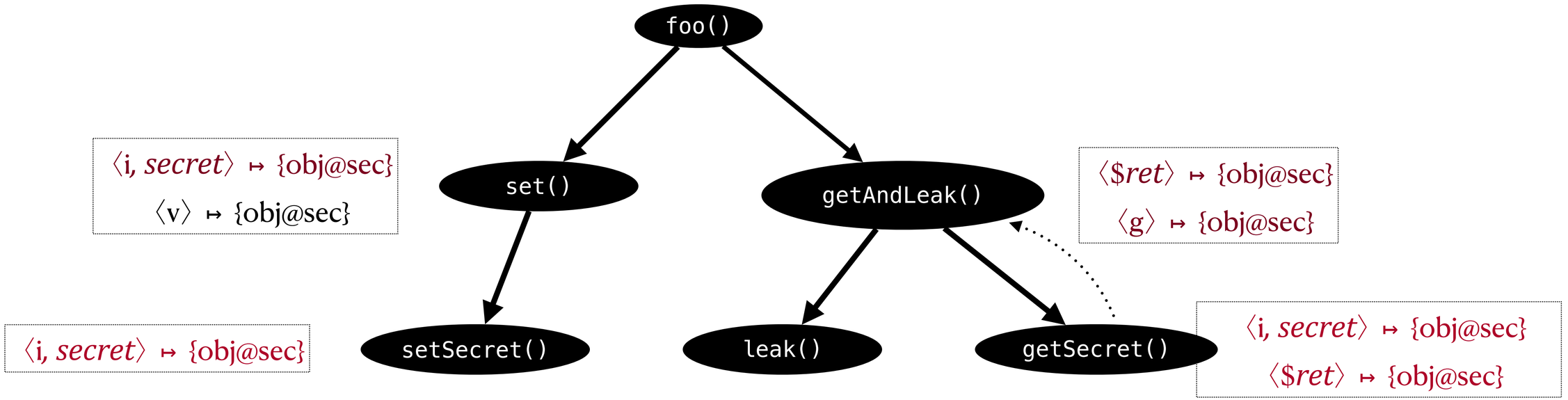


Summary Injection

For getAndLeak() **and** getSecret()

```
13 BrigdeClass::getSecret() {
14     return this.secret;
15 }
```

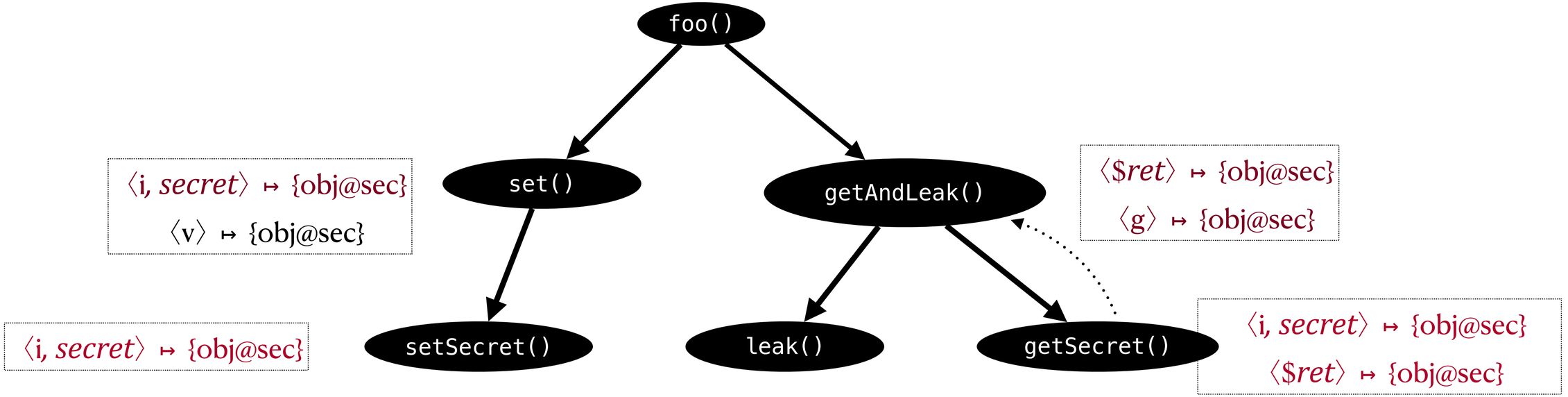
```
24 getAndLeak() {
25     g = i.getSecret();
26     leak(g);
27     return g;
28 }
```



Summary Injection

For foo()

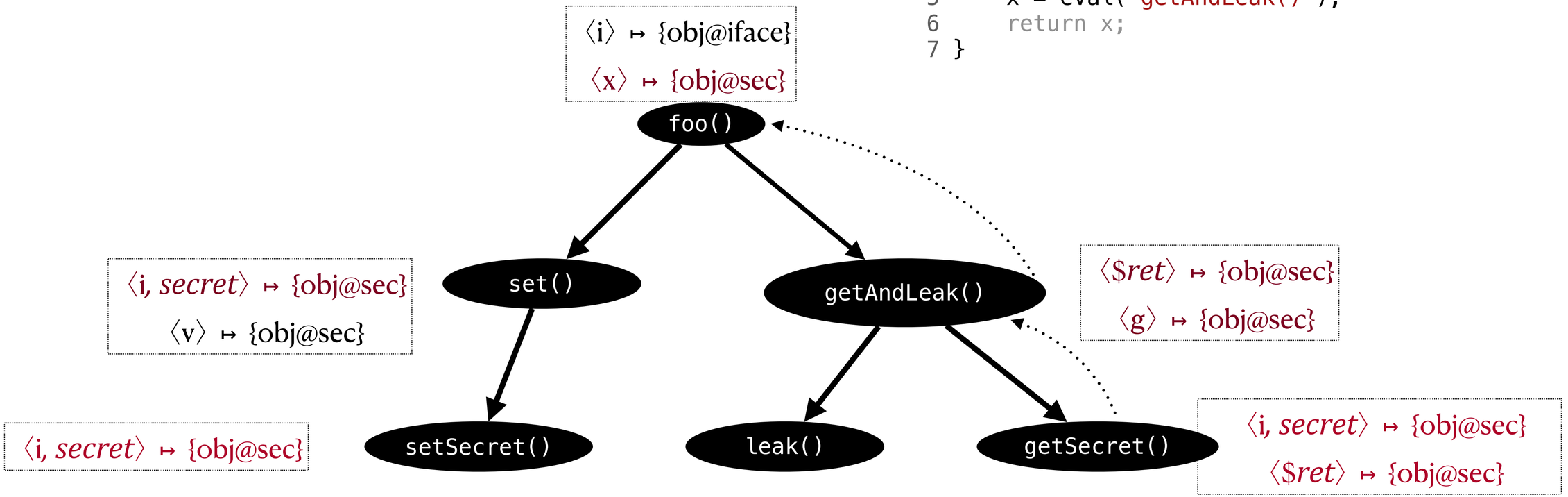
```
1 foo() {
2     i = new BrigdeClass(); // obj@iface
3     eval("set()"); // invoke the guest function
4     // do some computation
5     x = eval("getAndLeak()");
6     return x;
7 }
```



Summary Injection

For foo()

```
1 foo() {
2     i = new BrigdeClass(); // obj@iface
3     eval("set()"); // invoke the guest function
4     // do some computation
5     x = eval("getAndLeak()");
6     return x;
7 }
```



Evaluation – Precision

Android NDK (Nativeflow microbenchmarks)

Android NDK combines Java with C/C++

Host Language (Frontend)	Guest Language (Frontend)	Benchmark
Java (Wala)	C/C++ (SVF)	NativeFlowBench

Interlanguage edges Matches ground truth in 21/24 cases

Benchmark	C-Summary Generated	JuCify Analyze	N_{bj}	E_{bj}	N_{aj}	E_{aj}	IL	Conflux Analyze	N_J	E_J	N_C	E_C	IL	Time (in sec)
NativeFlowBench														
complexdata	□	✓	26	26	9	7	2	✓	364	940	8	10	2	< 1
complexdata stringop	✗	✓	26	26	9	7	2	✓	5802	13015	9	10	1	< 1
dynamic register multiple	✗	✓	18	18	0	0	2	✓	382	979	9	13	3	< 1
heap modify	□	✓	25	25	7	7	1	✓	6889	15272	11	12	1	< 1
icc javatonative	✗	✓	17	17	2	2	0	✗	366	895	13	14	0	< 1
icc nativetojava	□	✓	27	27	5	5	0	✓	7865	14891	8	9	1	< 1
leak	□	✓	18	18	4	3	1	✓	367	894	3	4	1	< 1
leak array	□	✓	18	18	5	4	1	✓	383	968	4	5	1	< 1
leak dynamic register	□	✓	18	18	4	3	1	✓	367	895	9	11	1	< 1
method overloading	□	✓	18	18	3	2	1	✓	307	751	3	5	2	< 1
multiple interactions	□	✓	18	18	3	2	1	✓	384	980	11	13	2	< 1
multiple libraries	□	✓	18	18	5	4	1	✓	382	978	3	5	1	< 1
noleak	□	✓	18	18	3	2	1	✓	367	894	1	2	1	< 1
noleak array	□	✓	18	18	5	4	1	✓	5795	13008	4	5	1	< 1
nosource	⊤	✓	13	13	1	0	0	✓	306	746	1	2	1	< 1
pure	✗	✗	No call graph was generated					✗	6581	14686	39	42	0	< 1
pure direct	✗	✗	No call graph was generated					✗	6581	14686	30	48	0	< 1
pure direct customized	✗	✗	No call graph was generated					✗	380	958	30	48	0	< 1
set field from arg	⊤	✓	13	13	12	11	1	✓	373	908	4	5	1	< 1
set field from arg field	⊤	✓	31	31	12	11	1	✓	314	763	4	5	1	< 1
set field from native	⊤	✗	25	25	7	7	0	✓	310	762	14	15	1	< 1
source	⊤	✗	27	27	9	9	0	✓	6892	15271	7	8	1	< 1
source clean	⊤	✗	27	27	9	9	0	✓	385	985	4	5	1	< 1

Fails to generate equivalent source code

Matches to ground truth in 16/24 cases

Evaluation – Scalability

Android NDK

Host Language (Frontend)	Guest Language (Frontend)	Benchmark
Java (Wala)	C/C++ (SVF)	15 applications from Google Play Store

- Conflux constructs call-graph for 12/15 (< 15 minutes⏰).
- Three apps have used inter-component communication — limitation of conflux

Evaluation

Graal VM Polyglot API

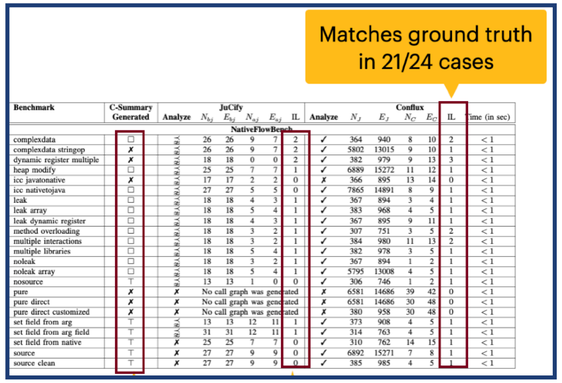
Host Language (Frontend)	Guest Language (Frontend)	Microbenchmarks
Java (Wala)	Python (Wala)	Curated microbenchmarks from Github

Benchmark	Wala-Java	Wala-Python	Conflux
BM-1	✓	✓	✓
BM-2	✓	✓	✓
BM-3	✗	✓	✗

Reflections

Can we combine monolingual static analyses for analysing multilingual applications?

- Promising approach
  - Scalable and precise; shown in experiments



Benchmark	C-Summary Generated	Analyze	JuCify N_{sj}	JuCify E_{sj}	JuCify N_{cj}	JuCify E_{cj}	JuCify IL	Analyze	Conflux N_f	Conflux E_f	Conflux N_c	Conflux E_c	Conflux IL	Time (in sec)
NativeFlowBench														
complexdata	☐	✓	26	26	9	7	2	✓	364	940	8	10	2	< 1
complexdata stringop	✗	✓	26	26	9	7	2	✓	5802	13015	9	10	1	< 1
dynamic register multiple	✗	✓	18	18	0	0	2	✓	382	979	9	13	3	< 1
heap modify	☐	✓	25	25	7	7	1	✓	6889	15272	11	12	1	< 1
icc javatonative	✗	✓	17	17	2	2	0	✗	366	895	13	14	0	< 1
icc nativetojava	☐	✓	27	27	5	5	0	✓	7865	14891	8	9	1	< 1
leak	☐	✓	18	18	4	3	1	✓	367	894	3	4	1	< 1
leak array	☐	✓	18	18	5	4	1	✓	383	968	4	5	1	< 1
leak dynamic register	☐	✓	18	18	4	3	1	✓	367	895	9	11	1	< 1
method overloading	☐	✓	18	18	3	2	1	✓	307	751	3	5	2	< 1
multiple interactions	☐	✓	18	18	3	2	1	✓	384	980	11	13	2	< 1
multiple libraries	☐	✓	18	18	5	4	1	✓	382	978	3	5	1	< 1
noleak	☐	✓	18	18	3	2	1	✓	367	894	1	2	1	< 1
noleak array	☐	✓	18	18	5	4	1	✓	5795	13008	4	5	1	< 1
nosource	⊤	✓	13	13	1	0	0	✓	306	746	1	2	1	< 1
pure	✗	✗	No call graph was generated					✗	6581	14686	39	42	0	< 1
pure direct	✗	✗	No call graph was generated					✗	6581	14686	30	48	0	< 1
pure direct customized	✗	✗	No call graph was generated					✗	380	958	30	48	0	< 1
set field from arg	⊤	✓	13	13	12	11	1	✓	373	908	4	5	1	< 1
set field from arg field	⊤	✓	31	31	12	11	1	✓	314	763	4	5	1	< 1
set field from native	⊤	✗	25	25	7	7	0	✓	310	762	14	15	1	< 1
source	⊤	✗	27	27	9	9	0	✓	6892	15271	7	8	1	< 1
source clean	⊤	✗	27	27	9	9	0	✓	385	985	4	5	1	< 1

Host Language (Frontend)	Guest Language (Frontend)	Benchmark
Java (Wala)	C/C++ (SVF)	15 applications from Google Play Store

- Conflux constructs call-graph for 12/15 (< 15 minutes⏰).
- Three apps have used inter-component communication — limitation of conflux

Host Language (Frontend)	Guest Language (Frontend)	Microbenchmarks
Java (Wala)	Python (Wala)	Curated microbenchmarks from Github

Benchmark	Wala-Java	Wala-Python	Conflux
BM-1	✓	✓	✓
BM-2	✓	✓	✓
BM-3	✗	✓	✗

Back to square one!

1. How vulnerable is the communication in Android Hybrid Apps?

2. Can we combine monolingual static analyses for analysing multilingual applications?

Back to square one!

1

How vulnerable is the communication in Android Hybrid Apps?

Understand WebView Apps

(LuDroid)

Glue bridge method summaries for demand driven IFC analysis for WebView

(IWandDroid)

2

Can we combine monolingual static analyses for analysing multilingual applications?

Back to square one!

1

How vulnerable is the communication in Android Hybrid Apps?

Understand WebView Apps

(LuDroid)

Glue bridge method summaries for demand driven IFC analysis for WebView

(IWandDroid)

2

Can we combine monolingual static analyses for analysing multilingual applications?

Combine existing monolingual analysis for analysis of multilingual programs

(Conflux)

Back to square one!

1

How vulnerable is the communication in Android Hybrid Apps?

2

Can we combine monolingual static analyses for analysing multilingual applications?

Understand WebView Apps

(LuDroid)

Glue bridge method summaries for demand driven IFC analysis for WebView

(IWandDroid)

Combine existing monolingual analysis for analysis of multilingual programs

(Conflux)

Function summaries facilitate scalable static analysis for multilingual programs

Static Analyses of Interlanguage Interoperations