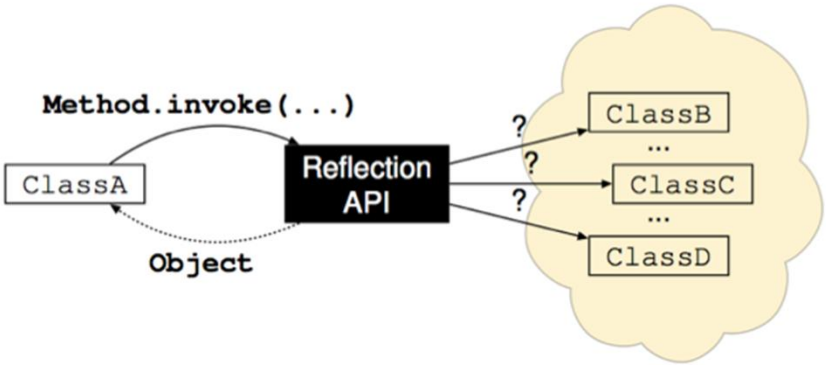
Static Resolution of Implicit Control Flow for Reflection and Message-Passing

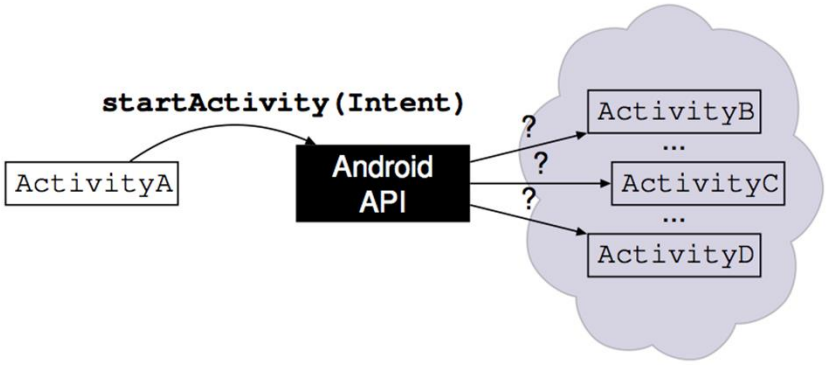
Paulo Barros, René Just, Suzanne Millstein,
Paul Vines, Werner Dietl, Marcelo d'Amorim and
Michael D. Ernst

Implicit control flow

- Indirect method call
- Design pattern that allows coding flexibility



Reflection



Message-Passing
(Android Intents)

Problem: imprecise summaries for static analyses

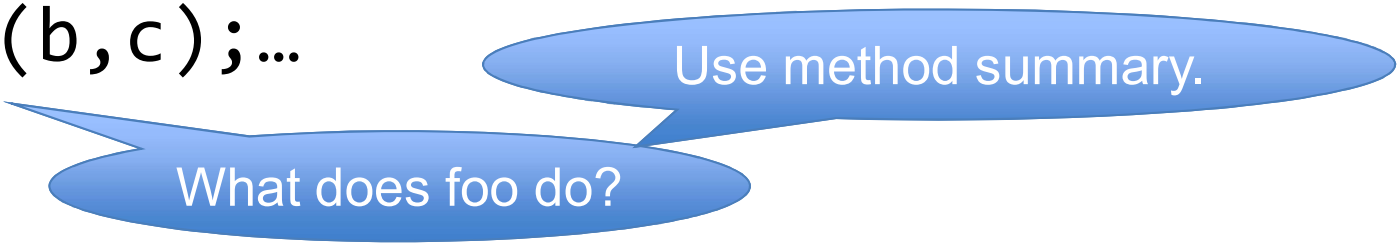
```
…a.foo(b,c);…
```

Problem: imprecise summaries for static analyses

```
…a.foo(b,c);…
```

Problem: imprecise summaries for static analyses

```
…a.foo(b,c);…
```



Problem: imprecise summaries for static analyses

```
…a.foo(b,c);…
```

Use method summary.

What does foo do?

```
…myMethod.invoke(a,b,c);…
```

Problem: imprecise summaries for static analyses

```
…a.foo(b,c);…
```

Use method summary.

What does foo do?

```
…myMethod.invoke(a,b,c);…
```

What does invoke do?

Problem: imprecise summaries for static analyses

```
…a.foo(b,c);…
```

Use method summary.

What does foo do?

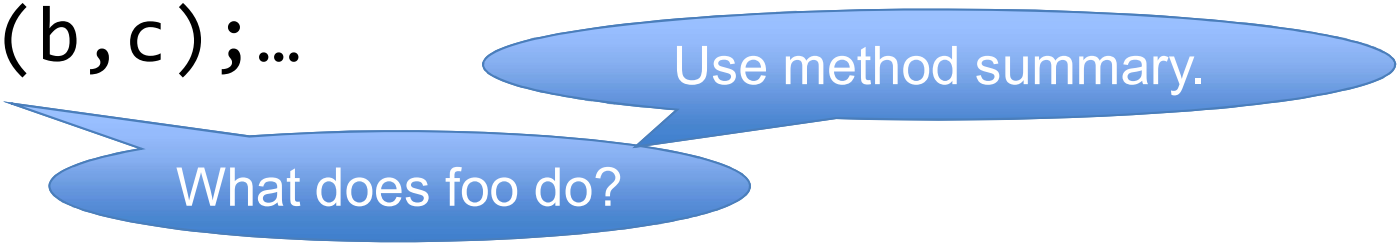
```
…myMethod.invoke(a,b,c);…
```

Anything!

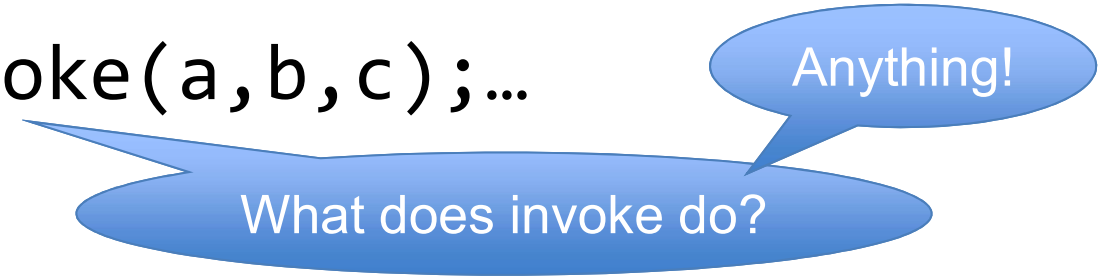
What does invoke do?

Problem: imprecise summaries for static analyses

…`a.foo(b,c);`…

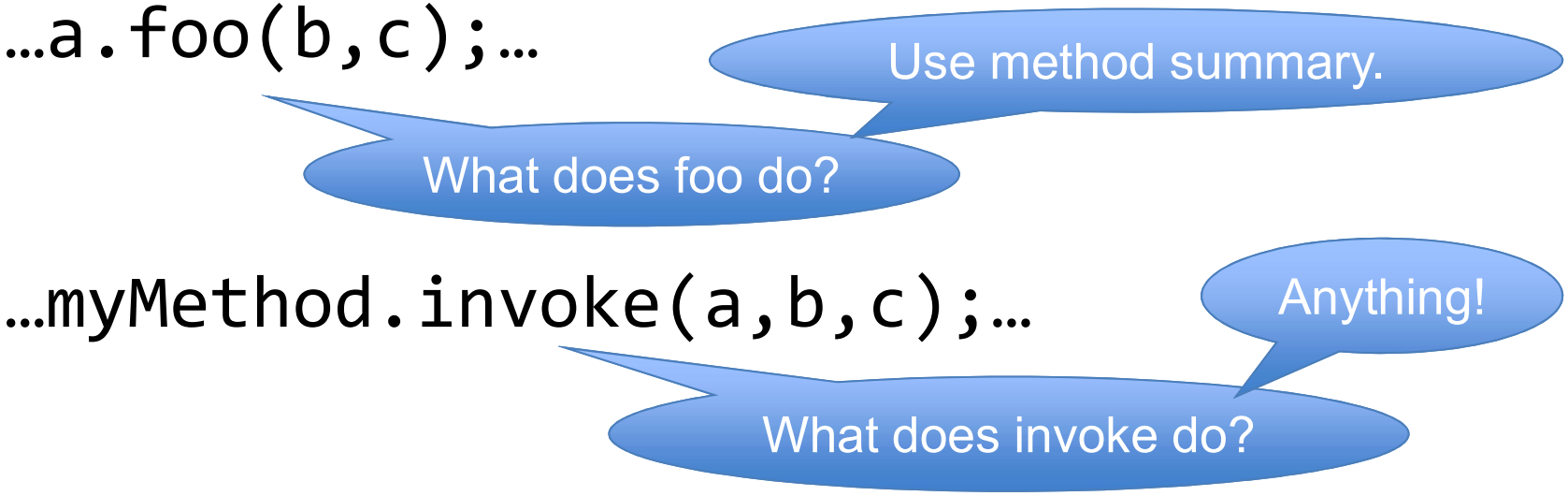


…`myMethod.invoke(a,b,c);`…



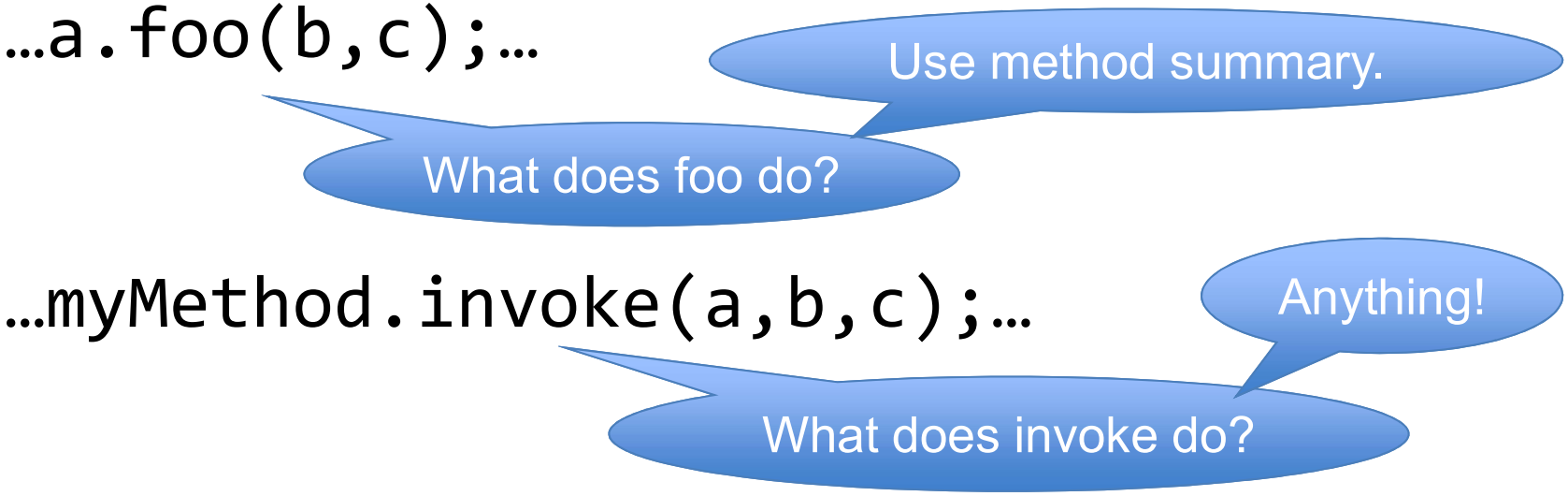
- Sound analysis → Imprecise

Problem: imprecise summaries for static analyses



- Sound analysis → Imprecise
- Unsound analysis → Precise but unsafe

Problem: imprecise summaries for static analyses



- Sound analysis → Imprecise
- Unsound analysis → Precise but unsafe
- Goal → Soundness and high precision

Android

- Over 1 billion active users
- Over 1.6 million apps

- Analyzing apps is important
- Example: Malware detection
  - Soundness is crucial

Implicit control flow is pervasive in Android

- F-Droid is a repository of Android apps

- F-Droid apps
  - 39% use reflection
  - 69% share data through intents

- **Conclusion** → Static analysis on Android apps must handle implicit control flow

Resolving implicit control flow

- **Goal** → Soundly resolve implicit control flows
- **Observation** → Statically resolvable in F-Droid
  - 93% of reflective calls
  - 88% of sent intents
- **Solution** → We developed type systems that model implicit control flows
- **Results**
  - Improves the precision by 400x
  - Soundness is maintained
  - Low developer effort

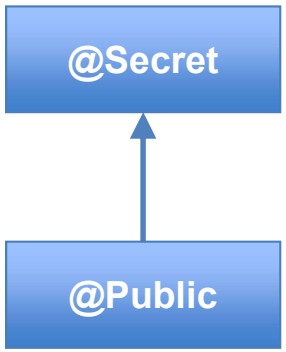
Resolving implicit control flow

- **Goal** → Soundly resolve implicit control flows
- **Observation** → Statically resolvable in F-Droid
  - 93% of reflective calls
  - 88% of sent intents
- **Solution** → We developed type systems that model implicit control flows
- **Results**
  - Improves the precision by 400x
  - Soundness is maintained
  - Low developer effort

Reflection and intents in real apps

Non-interference type system

- Guarantees that the program does not leak sensitive data
- **Privacy-types:**
  - **@Secret:** Sensitive-data values
  - **@Public:** Non-sensitive-data values

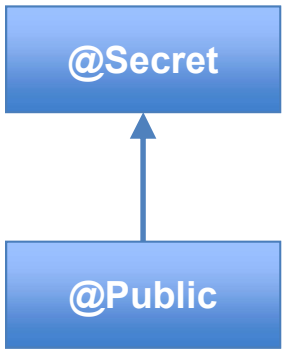



```
@Public String var;
@Secret String password = getPassword();
var = password;
```

@Public ← @Secret

Non-interference type system

- Guarantees that the program does not leak sensitive data
- **Privacy-types:**
  - **@Secret:** Sensitive-data values
  - **@Public:** Non-sensitive-data values




```
@Public String var;
@Secret String password = getPassword();
var = password;
```

@Public ⇐ **@Secret**

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
  // In Android SDK ≥ 11.
  @Public ActionBar getActionBar() {...}
}
class Method {
  @Secret Object invoke(Object obj, Object... args) {...}
}
```

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
   Class<?> clazz = Activity.class;
   Method mtd = clazz.getMethod("getActionBar");
   @Public Object actionBar = mtd.invoke(this);
   …
}…
```

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
  // In Android SDK ≥ 11.
  @Public ActionBar getActionBar() {...}
}
class Method {
  @Secret Object invoke(Object obj, Object... args) {...}
}
```

Conservative annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
   Class<?> clazz = Activity.class;
   Method mtd = clazz.getMethod("getActionBar");
   @Public Object actionBar = mtd.invoke(this);
   …
}…
```

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
  // In Android SDK ≥ 11.
  @Public ActionBar getActionBar() {...}
}
class Method {
  @Secret Object invoke(Object obj, Object... args) {...}
}
```

Conservative annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
   Class<?> clazz = Activity.class;
   Method mtd = clazz.getMethod("getActionBar");
   @Public Object actionBar = mtd.invoke(this);
   …
}…
```

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
  // In Android SDK ≥ 11.
  @Public ActionBar getActionBar() {...}
}
class Method {
  @Secret Object invoke(Object obj, Object... args) {...}
}
```

Conservative annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
   Class<?> clazz = Activity.class;
   Method mtd = clazz.getMethod("getActionBar");
   @Public Object actionBar = mtd.invoke(this);
   …
}…
```

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
  // In Android SDK ≥ 11.
  @Public ActionBar getActionBar() {...}
}
class Method {
  @Secret Object invoke(Object obj, Object... args) {...}
}
```

Conservative annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
   Class<?> clazz = Activity.class;
   Method mtd = clazz.getMethod("getActionBar");
   @Public Object actionBar = mtd.invoke(this);
   …
}…
```

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
  // In Android SDK ≥ 11.
  @Public ActionBar getActionBar() {...}
}
class Method {
  @Secret Object invoke(Object obj, Object... args) {...}
}
```

Conservative annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
   Class<?> clazz = Activity.class;
   Method mtd = clazz.getMethod("getActionBar");
   @Public Object actionBar = mtd.invoke(this);
   …
}…
```

@Public ← @Secret

Use of reflection – Aarddict

```
// Library Annotations:
class Activity {
  // In Android SDK ≥ 11.
  @Public ActionBar getActionBar() {...}
}
class Method {
  @Secret Object invoke(Object obj, Object... args) {...}
}
```

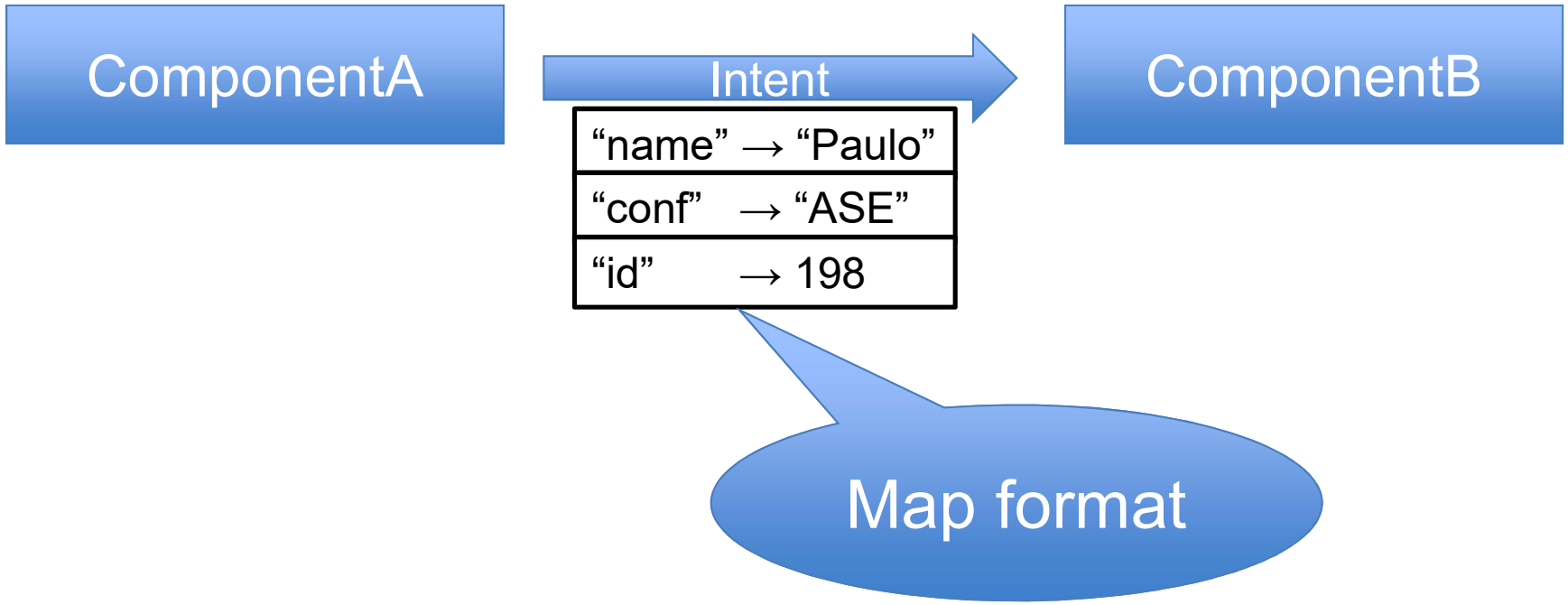
Conservative annotation

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
   Class<?> clazz = Activity.class;
   Method mtd = clazz.getMethod("getActionBar");
   @Public Object actionBar = mtd.invoke(this);
   …
}…
```

@Public ← @Public

Intent payloads

Intent payloads



Use of intent payloads – Aarddict

```
class LookupWord extends Activity {
  void translateWord(@Public String sentence) {
    Intent i = new Intent(this, WordTranslator.class);
    i.putExtra("sentence", sentence);
    startActivity(i);
}…}
```

```
// Library Annotations
class Intent {
  @Secret String
  getStringExtra(String key) {...}
}
```

```
class WordTranslator extends Activity {
  void onCreate(Bundle savedInstanceState)
    Intent i = getIntent();
    @Public String sentence = i.getStringExtra("sentence");
    …
}…}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {
  void translateWord(@Public String sentence) {
    Intent i = new Intent(this, WordTranslator.class);
    i.putExtra("sentence", sentence);
    startActivity(i);
}…}
```

```
// Library Annotations
class Intent {
  @Secret String
  getStringExtra(String key) {...}
}
```

Conservative annotation

```
class WordTranslator extends Activity {
  void onCreate(Bundle savedInstanceState)
    Intent i = getIntent();
    @Public String sentence = i.getStringExtra("sentence");
    …
}…}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {
  void translateWord(@Public String sentence) {
    Intent i = new Intent(this, WordTranslator.class);
    i.putExtra("sentence", sentence);
    startActivity(i);
}…}
```

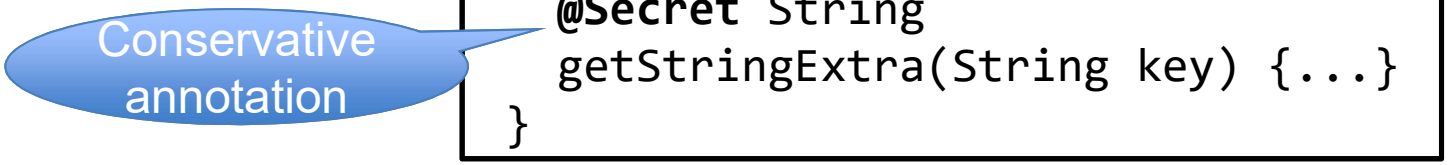
```
// Library Annotations
class Intent {
  @Secret String
  getStringExtra(String key) {...}
}
```

Conservative annotation

```
class WordTranslator extends Activity {
  void onCreate(Bundle savedInstanceState)
    Intent i = getIntent();
    @Public String sentence = i.getStringExtra("sentence");
    …
}…}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {
  void translateWord(@Public String sentence) {
    Intent i = new Intent(this, WordTranslator.class);
    i.putExtra("sentence", sentence);
    startActivity(i);
}…}
```




```
class WordTranslator extends Activity {
  void onCreate(Bundle savedInstanceState)
    Intent i = getIntent();
    @Public String sentence = i.getStringExtra("sentence");
    …
}…}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {
  void translateWord(@Public String sentence) {
    Intent i = new Intent(this, WordTranslator.class);
    i.putExtra("sentence", sentence);
    startActivity(i);
}…}
```

```
// Library Annotations
class Intent {
  @Secret String
  getStringExtra(String key) {...}
}
```

Conservative annotation

```
class WordTranslator extends Activity {
  void onCreate(Bundle savedInstanceState)
    Intent i = getIntent();
    @Public String sentence = i.getStringExtra("sentence");
    …
}…}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {
  void translateWord(@Public String sentence) {
    Intent i = new Intent(this, WordTranslator.class);
    i.putExtra("sentence", sentence);
    startActivity(i);
}…}
```

```
// Library Annotations
class Intent {
  @Secret String
  getStringExtra(String key) {...}
}
```

Conservative annotation

```
class WordTranslator extends Activity {
  void onCreate(Bundle savedInstanceState)
    Intent i = getIntent();
    @Public String sentence = i.getStringExtra("sentence");
    …
}…}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {
  void translateWord(@Public String sentence) {
    Intent i = new Intent(this, WordTranslator.class);
    i.putExtra("sentence", sentence);
    startActivity(i);
}…}
```

```
// Library Annotations
class Intent {
  @Secret String
  getStringExtra(String key) {...}
}
```

Conservative annotation

```
class WordTranslator extends Activity {
  void onCreate(Bundle savedInstanceState)
    Intent i = getIntent();
    @Public String sentence = i.getStringExtra("sentence");
    …
}…}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {
  void translateWord(@Public String sentence) {
    Intent i = new Intent(this, WordTranslator.class);
    i.putExtra("sentence", sentence);
    startActivity(i);
}…}
```

```
// Library Annotations
class Intent {
  @Secret String
  getStringExtra(String key) {...}
}
```

Conservative annotation

```
class WordTranslator extends Activity {
  void onCreate(Bundle savedInstanceState)
    Intent i = getIntent();
    @Public String sentence = i.getStringExtra("sentence");
    …
}…}
```

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {
  void translateWord(@Public String sentence) {
    Intent i = new Intent(this, WordTranslator.class);
    i.putExtra("sentence", sentence);
    startActivity(i);
}…}
```

```
// Library Annotations
class Intent {
  @Secret String
  getStringExtra(String key) {...}
}
```

Conservative annotation

```
class WordTranslator extends Activity {
  void onCreate(Bundle savedInstanceState)
    Intent i = getIntent();
    @Public String sentence = i.getStringExtra("sentence");
    …
}…}
```

@Public ← @Secret

Use of intent payloads – Aarddict

```
class LookupWord extends Activity {
  void translateWord(@Public String sentence) {
    Intent i = new Intent(this, WordTranslator.class);
    i.putExtra("sentence", sentence);
    startActivity(i);
}…}
```

```
// Library Annotations
class Intent {
  @Secret String
    getStringExtra(String key) {...}
}
```

Conservative annotation

```
class WordTranslator extends Activity {
  void onCreate(Bundle savedInstanceState)
    Intent i = getIntent();
    @Public String sentence = i.getStringExtra("sentence");
    …
}…}
```


@Public ← @Public

Reflection Analysis

Reflection resolution

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
  Class<?> clazz = Activity.class;
  Method mtd = clazz.getMethod("getActionBar");
  @Public Object actionBar = mtd.invoke(this);
  …
}…
```

Reflection resolution

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
  Class<?> clazz = Activity.class;
  Method mtd = clazz.getMethod("getActionBar");
  @Public Object actionBar = mtd.invoke(this);
  …
}…
```

The type of `clazz` is inferred to represent Activity

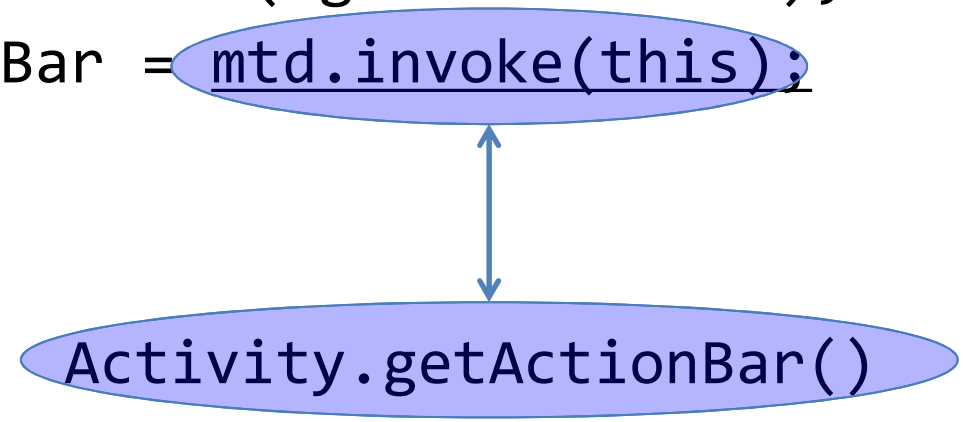
Reflection resolution

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
  Class<?> clazz = Activity.class;
  Method mtd = clazz.getMethod("getActionBar");
  @Public Object actionBar = mtd.invoke(this);
  …
}…
```

The type of `mtd` is inferred to represent Activity.getActionBar()

Reflection resolution

```
if (android.os.Build.VERSION.SDK_INT >= 11) {
   Class<?> clazz = Activity.class;
   Method mtd = clazz.getMethod("getActionBar");
   @Public Object actionBar = mtd.invoke(this);
   …
}…
```



*Conceptual replacement

Reflection type system

Refines the Java type system

- Indicates an exact class
  - Example
    - **@ClassVal**(“java.util.HashMap”)

- Indicates an upper bound of a class
  - Example
    - **@ClassBound**(“java.util.HashMap”)

Reflection type system

Refines the Java type system

- Indicates an exact class
  - Example
    - **@ClassVal**(“java.util.HashMap”)
- Indicates an upper bound of a class
  - Example
    - **@ClassBound**(“java.util.HashMap”)

Reflection type system

Refines the Java type system

- Indicates a method
  - Example
    - **@MethodVal**("java.util.HashMap.containsKey(Object)")

Constant value analysis

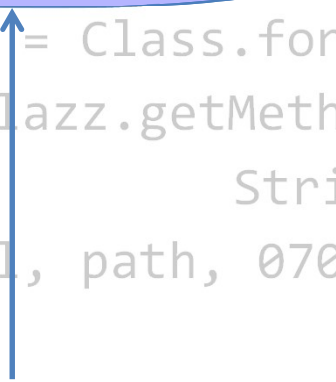
- Constant folding
- Constant propagation
- Multiple values, not just one
- Evaluate side-effect-free methods
- Infer and track length of arrays
- Implemented as a type system and dataflow analysis

Constant value inference

```
void restrictFileAccess(String path) {
    String fileUtilsClassName = "android.os.FileUtils";
    Class<?> clazz = Class.forName(fileUtilsClassName);
    Method mtd = clazz.getMethod("setPermissions",
                                 String.class, int.class);
    mtd.invoke(null, path, 0700);
}
```

Constant value inference

```
void restrictFileAccess(String path) {
    String fileUtilsClassName = "android.os.FileUtils";
    Class<?> clazz = Class.forName(fileUtilsClassName);
    Method mtd = clazz.getMethod("setPermissions",
                                 String.class, int.class);
    mtd.invoke(null, path, 0700);
}
```



@StringVal(“android.os.FileUtils”)

Constant value inference

```
void restrictFileAccess(String path) {
    String fileUtilsClassName = "android.os.FileUtils";
    Class<?> clazz = Class.forName(fileUtilsClassName);
    Method mtd = clazz.getMethod("setPermissions",
                                 String.class, int.class);
    mtd.invoke(null, path, 0700);
}
```

@ClassVal(“android.os.FileUtils”)

- Inference of **@ClassVal**
  - C.class
  - Class.forName(arg)
  - ClassLoader.loadClass(arg)

Constant value inference

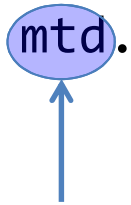
```
void restrictFileAccess(String path) {
    String fileUtilsClassName = "android.os.FileUtils";
    Class<?> clazz = Class.forName(fileUtilsClassName);
    Method mtd = clazz.getMethod("setPermissions",
                                 String.class, int.class);
    mtd.invoke(null, path, 0700);
}
```

@MethodVal(“android.os.FileUtils.setPermissions(String,int)”)

- Inference of **@MethodVal**
  - Class.getMethod(String n, Class<?> pT)
  - Class.getConstructor(String n, Class<?> pT)

Constant value inference

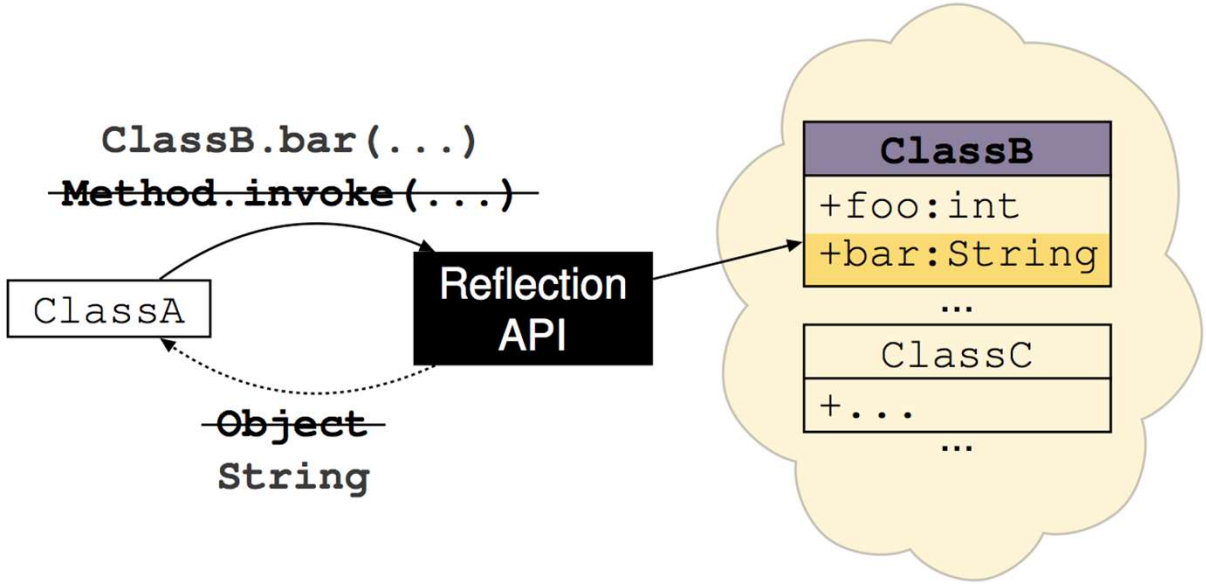
```
void restrictFileAccess(String path) {
    String fileUtilsClassName = "android.os.FileUtils";
    Class<?> clazz = Class.forName(fileUtilsClassName);
    Method mtd = clazz.getMethod("setPermissions",
                                 String.class, int.class);
    mtd.invoke(null, path, 0700);
}
```



@MethodVal(“android.os.FileUtils.setPermissions(String,int)”)

*Conceptual replacement

Reflection resolver



- Procedure summary is narrowed based on the Reflection type system
- Program remains unchanged
- Downstream analysis remains unchanged

Message-passing analysis (Android Intents)

Intent analysis

ComponentA

```
Intent i = buildIntent();
i.putExtra("key",getPass());
startActivity(i);
```

ComponentB

```
Intent i = getIntent();
int val = i.getIntExtra("key");
sendToEverybody(val);
```

- Intents present two challenges to static analyses:
  - Control flow
    - Component Communication Pattern (CCP)

    [D. Octeau *et al. USENIX '13*]
  - Data flow analysis
    - Intent type system

Intent analysis

ComponentA

```
Intent i = buildIntent();
i.putExtra("key",getPass());
startActivity(i);
```

ComponentB

```
Intent i = getIntent();
int val = i.getIntExtra("key");
sendToEverybody(val)
```

Who receives this message?

Who sent this message?

- Intents present two challenges to static analyses:
  - Control flow
    - Component Communication Pattern (CCP)

    [D. Octeau *et al. USENIX '13*]
  - Data flow analysis
    - Intent type system

Intent analysis

ComponentA

```
Intent i = buildIntent();
i.putExtra("key",getPass());
startActivity(i);
```

ComponentB

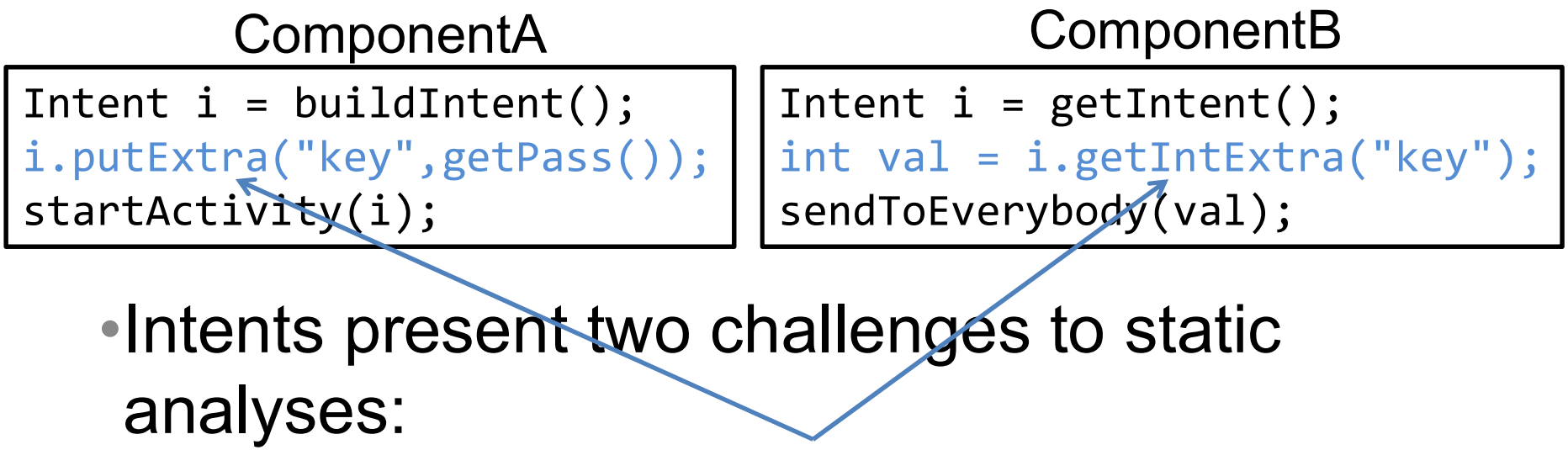
```
Intent i = getIntent();
int val = i.getIntExtra("key");
sendToEverybody(val)
```

Who receives this message?

Who sent this message?

- Intents present two challenges to static analyses:
  - Control flow
    - Component Communication Pattern (CCP) ←
    - [D. Octeau *et al. USENIX '13*]
  - Data flow analysis
    - Intent type system

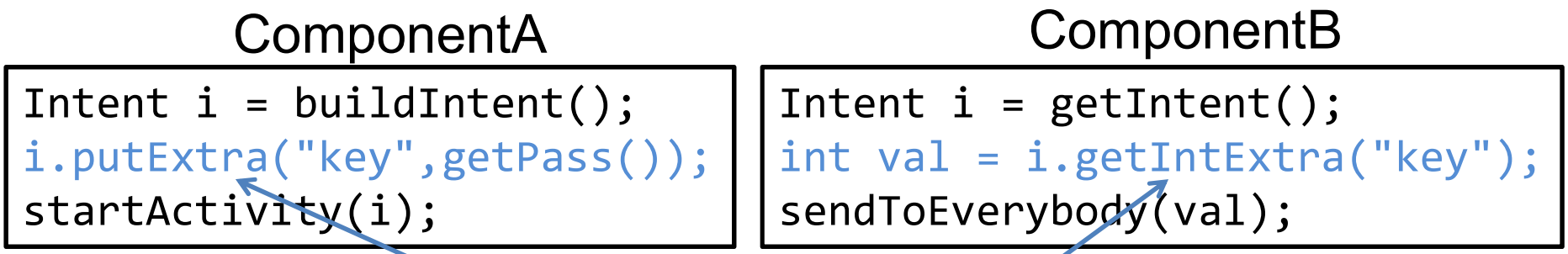
Intent analysis



- Intents present two challenges to static analyses:
  - Control flow
    - Component Communication Pattern (CCP)

    [D. Octeau *et al. USENIX '13*]
  - Data flow analysis ←
    - Intent type system

Intent analysis



- Intents present two challenges to static analyses:
  - Control flow
    - Component Communication Pattern (CCP)

    [D. Octeau *et al. USENIX '13*]
  - Data flow analysis
    - Intent type system ← Our contribution

Intent type system

- Syntax
  - `@Intent("K1" → t1, ..., "Kn" → tn) Intent i = …;`

T

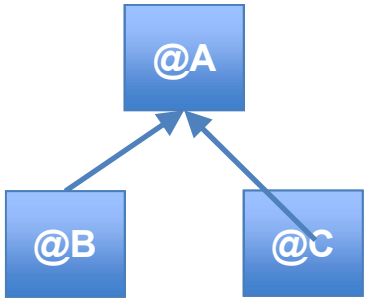
- Semantics
  - (C1): Keys accessed in *i* must be a subset of T's keys
  - (C2): *∀k ∈ domain(*T*) . i.get*Extra(k) :* `t[k]`

- Example


```
@Intent("k" → @C) Intent i = …
@A int e1 = i.getIntExtra("k");        // Legal
i.getIntExtra("otherKey");             // Violates (C1)
@B int e3 = i.getIntExtra("k");        // Violates (C2)
```

Intent type system

- Syntax
  - `@Intent("K1" → t1, ..., "Kn" → tn) Intent i = …;`
  - T
- Semantics
  - (C1): Keys accessed in *i* must be a subset of T's keys
  - (C2): *∀k ∈ domain(T) . i.get*Extra(k) :* `t[k]`
- Example




```
@Intent("k" → @C) Intent i = …
@A int e1 = i.getIntExtra("k");      // Legal
i.getIntExtra("otherKey");           // Violates (C1)
@B int e3 = i.getIntExtra("k");      // Violates (C2)
```

Intent type system inference

```
Intent i = new Intent();
@Secret int secret = …;
i.putExtra("akey", secret);
// i now has type @Intent("akey" → @Secret)
```

i:T

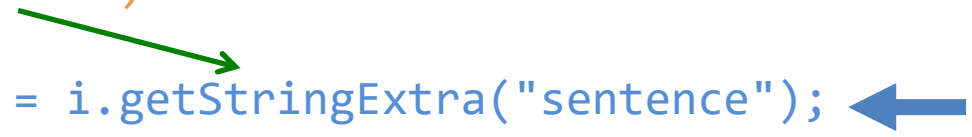
- Calls *i.putExtra(key, value)* always refine the type of i, **except when**:
  - *i* has aliases

    or
  - The declared type of *i* has *key* in its domain and *T[key] is a subtype of the refined type*

Revisiting example
Aarddict

```
class LookupWord extends Activity {
  void translateWord(@Public String sentence) {
    @Intent("sentence" → @Public)
    Intent i = new Intent(this, WordTranslator.class);
    i.putExtra("sentence", sentence);
    startActivity(i);
} … }
```

```
class WordTranslator extends Activity {
  void onCreate(Bundle savedInstanceState)
    @Intent("sentence" → @Public)
    Intent i = getIntent();
    @Public String sentence = i.getStringExtra("sentence");
    …
} … }
```



Evaluation

Research questions

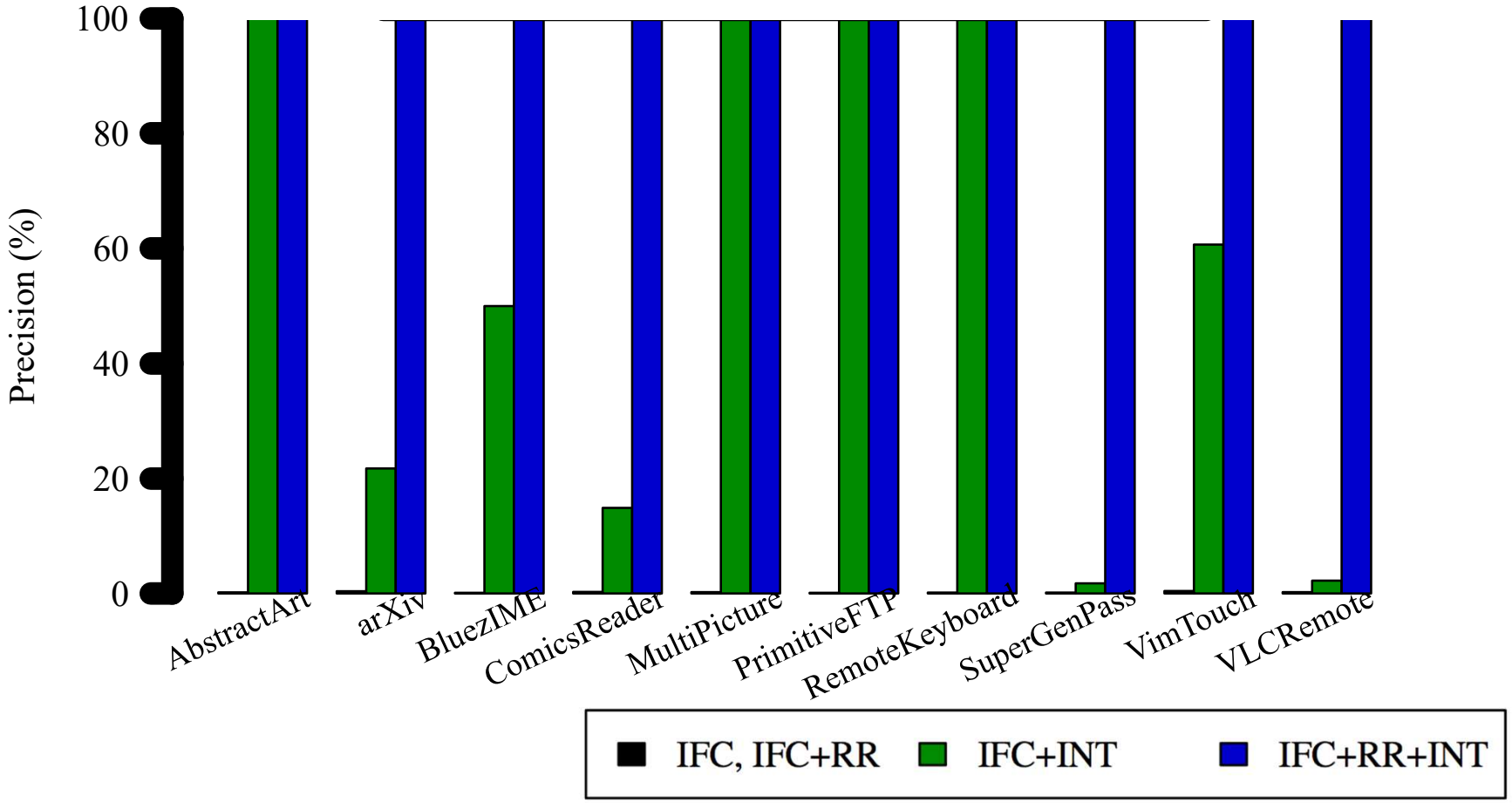
1. How much do our reflection and intent analyses improve the precision of a downstream analysis?

2. What is the annotation overhead for programmers?

Experimental setup

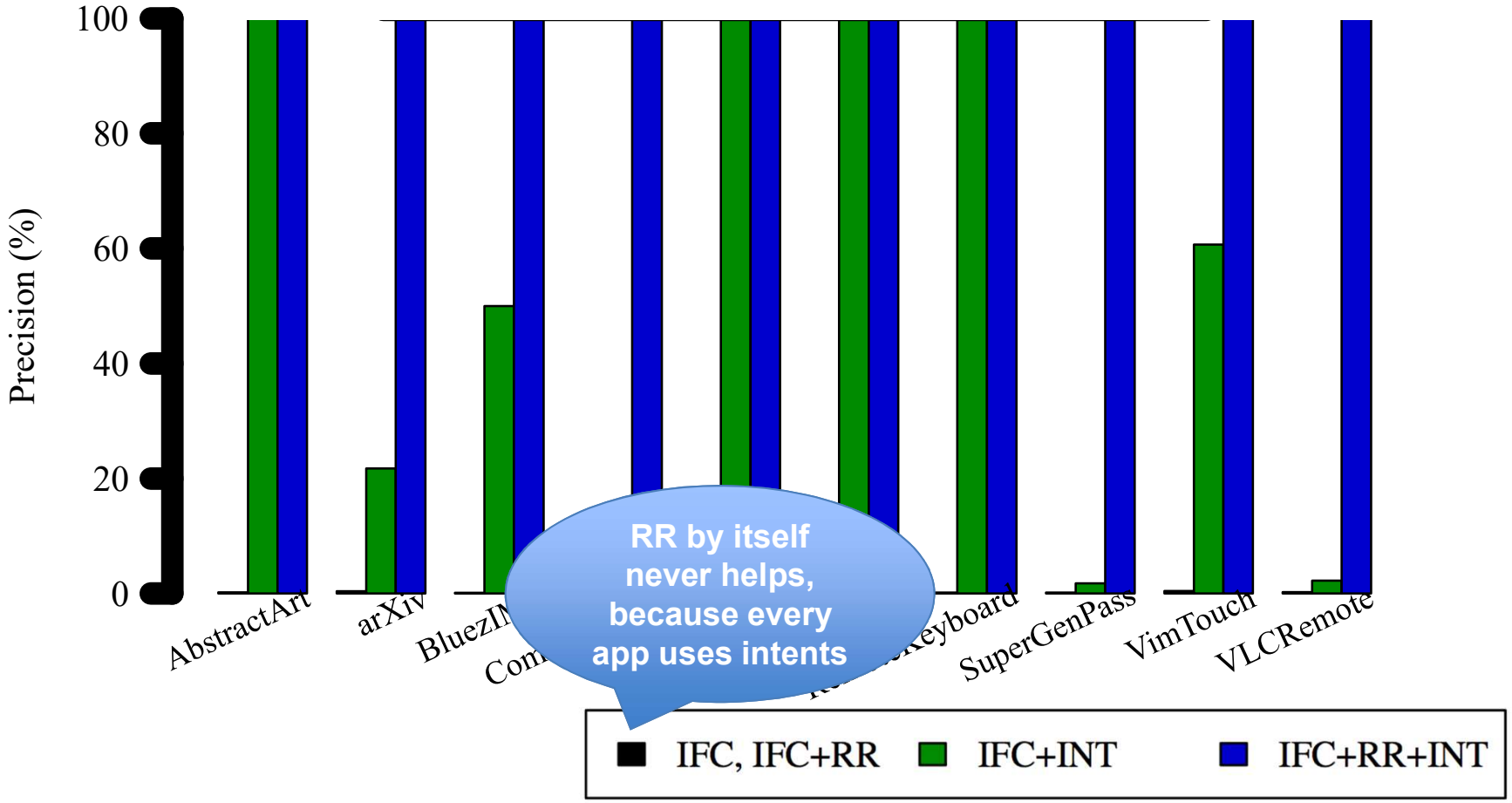
- 10 F-Droid apps
  - Each contain uses of reflection and intents
  - Average complexity → 5.3K LOC
- Downstream analysis
  - Information Flow Checker (IFC)
    https://github.com/typetools/sparta
- Metrics
  - Recall → 100%
  - Precision
    - # Real Flows / # Flows Reported
  - Programmer overhead → Number of annotations

Precision



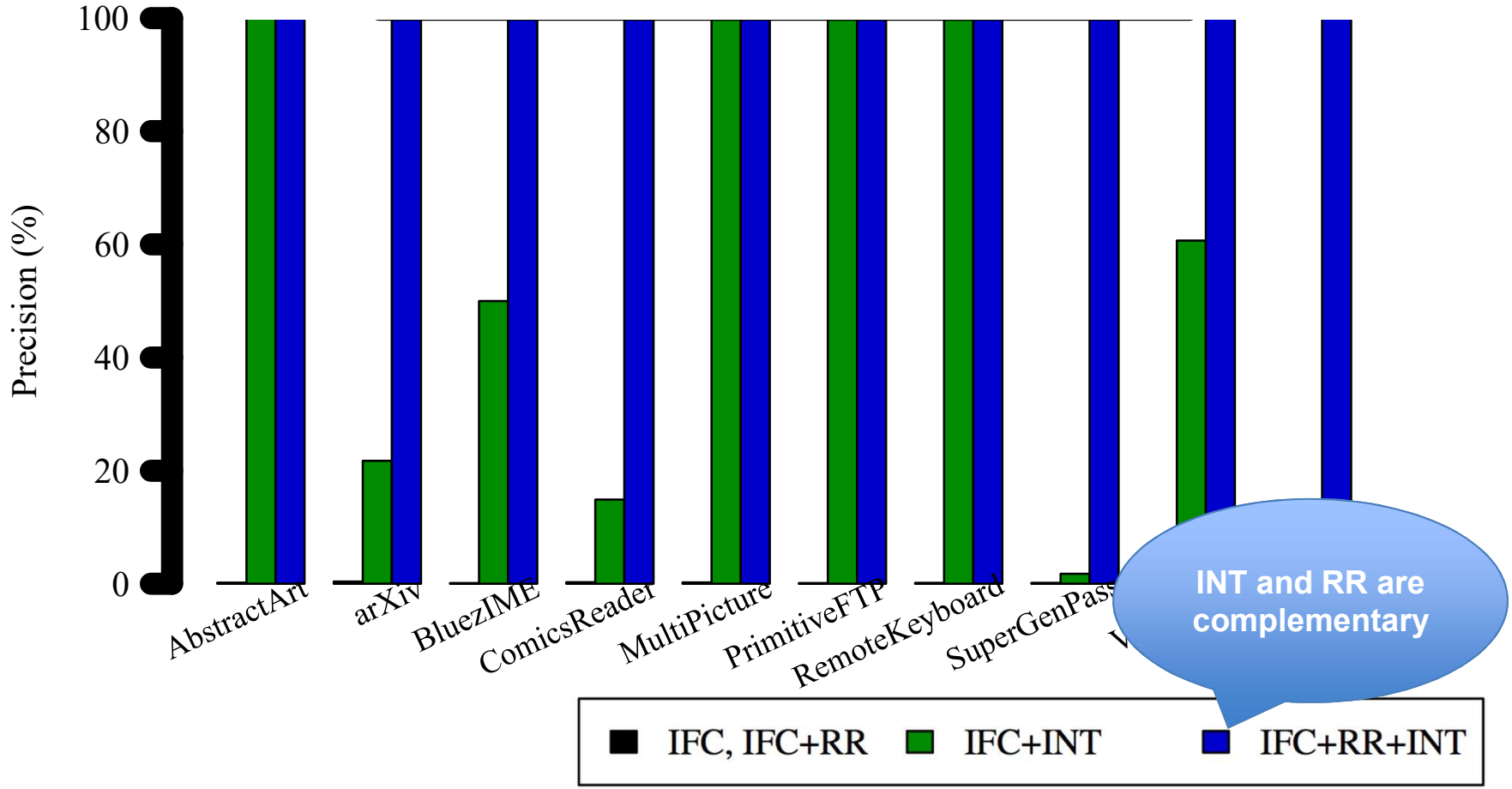
Average precision: **0.24%** **53%** **100%**

Precision



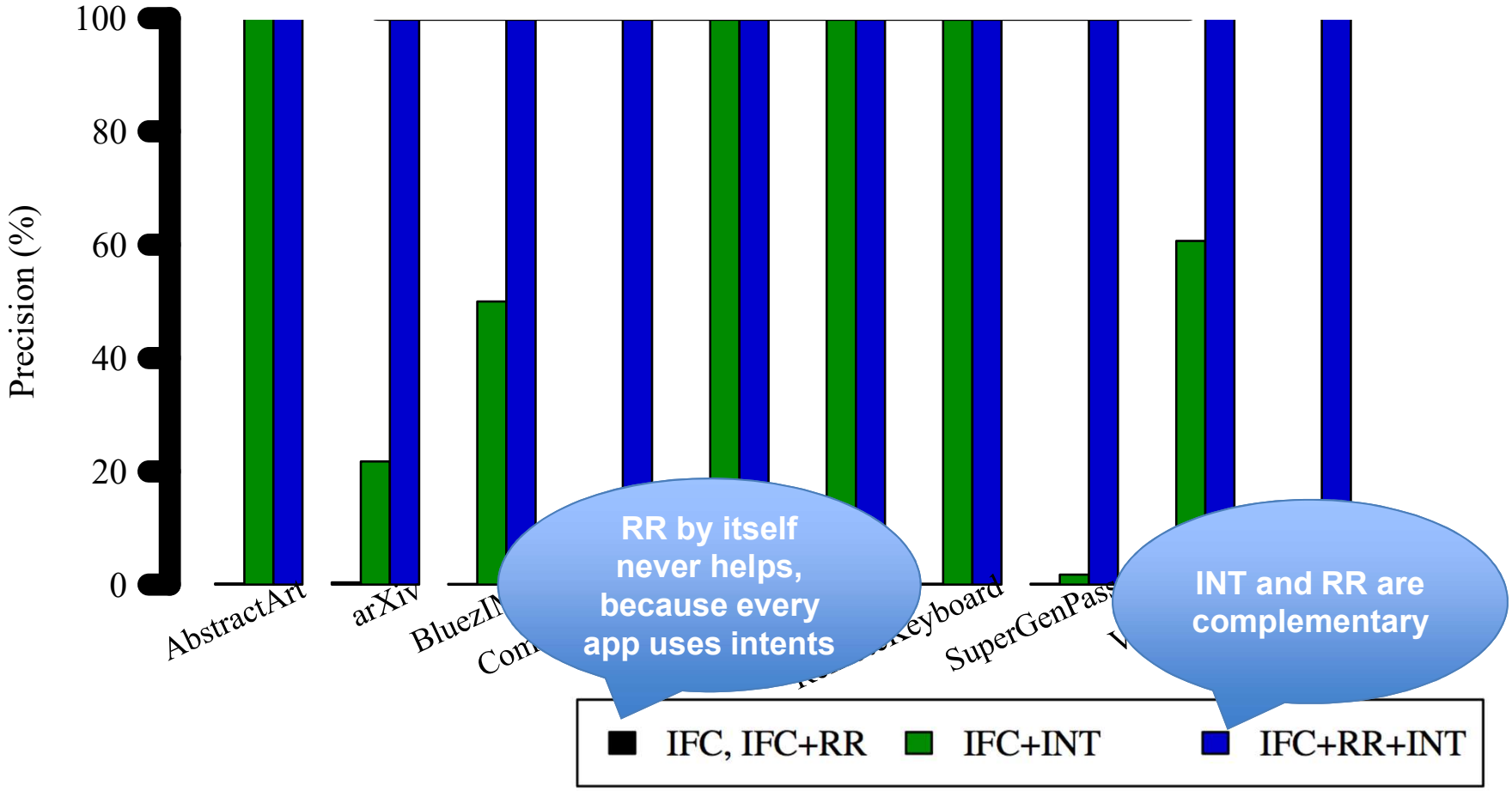
Average precision: **0.24%** **53%** **100%**

Precision



Average precision: **0.24%** **53%** **100%**

Precision



Average precision: **0.24%** **53%** **100%**

Annotation overhead

10 Android Apps	LOC	Reflection and Intent uses	# of annotations	
			IFC	RR+INT
Total	52,614	405	**5,583**	**98**

2% extra annotations

- For RR+INT → One annotation every ~2K LOC

Related work

- Reflection – sound, but limited:
  - M. S. Tschantz and M. D. Ernst, *OOPSLA'15*
  - Livshits *et al., APLAS '05*
  - M. Tatsubori *et al., PPL'04*
- Reflection – unsound:
  - Y. Li *et al., ECOOP'*14
  - Bodden *et al., ICSE'11*
- Intents – unsound:
  - L. Li *et al., ICSE'*15

Conclusion

- Two sound analyses for implicit control flows
  - Reflection
  - Message-Passing
- High precision for Android apps
  - Resolved 93% of reflective calls
  - Resolved 88% of sent intents
- Can be integrated with any downstream analysis
  - Improved precision by 400x
- Implementations are available

http://CheckerFramework.org/

Problem: imprecise summaries for static analyses

- …`a.foo(b,c);`… — Use method summary. / What does foo do?
- …`myMethod.invoke(a,b,c);`… — Anything! / What does invoke do?
- Sound analysis → Imprecise
- Unsound analysis → Precise but unsafe
- Goal → Soundness and high precision

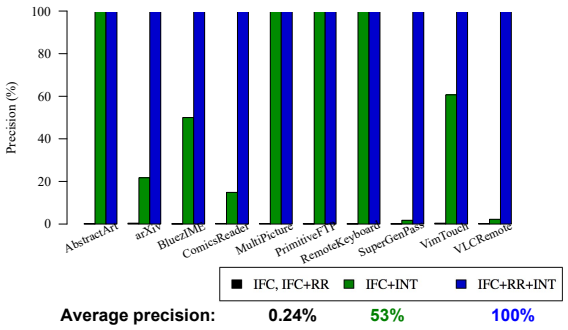
Resolving implicit control flow

- **Goal** → Soundly resolve implicit control flows
- **Observation** → Statically resolvable in F-Droid
  - –93% of reflective calls
  - –88% of sent intents
- **Solution** → We developed type systems that model implicit control flows
- **Results**
  - –Improves the precision by up to 400x
  - –Soundness is maintained
  - –Low developer effort

Research questions

1. How much do our reflection and intent analyses improve the precision of a downstream analysis?
2. What is the annotation overhead for programmers?

Precision



Average precision: 0.24% 53% 100%

Annotation overhead

10 Android Apps	LOC	Reflection and Intent uses	# of annotations	
			IFC	REF+INT
Total	52,614	405	**5,583**	**98**

2% extra annotations

- For REF+INT → One annotation every ~2K LOC

Conclusion

- Two sound analyses for implicit control flows
  - –Reflection
  - –Message-Passing
- High precision for Android apps
  - –Resolved 93% of reflective calls
  - –Resolved 88% of sent intents
- Can be integrated with any downstream analysis
  - Improved precision by 400x
- Implementations are available

CHECKER framework http://CheckerFramework.org/

UFPE — UNIVERSITY of WASHINGTON — UNIVERSITY OF WATERLOO

Paulo Barros - pbsf@cin.ufpe.br

Uses of reflection in Android apps

- 35 F-droid apps were evaluated (10+25)
  - Total of 142 reflective invocations
    - 81% to provide backward compatibility
    - 6% to access non-public/hidden methods
    - 13% are for other cases (duck-typing)

Reflection type inference rules

$$\frac{e : \texttt{String} \quad val \text{ is the statically computable value of } e}{e : \texttt{@StringVal}(val)}$$

$$\frac{e : \texttt{int} \quad val \text{ is the statically computable value of } e}{e : \texttt{@IntVal}(val)}$$

$$\frac{e : \texttt{@IntVal}(\pi)}{\texttt{new C[}e\texttt{]} : \texttt{@ArrayLen}(\pi)}$$

$$\frac{}{\texttt{new C[]\{}e_1, \ldots, e_n\texttt{\}} : \texttt{@ArrayLen}(n)}$$

$$\frac{fqn \text{ is the fully-qualified class name of C}}{\texttt{C.class} : \texttt{@ClassVal}(fqn)}$$

$$\frac{s : \texttt{@StringVal}(\nu)}{\texttt{Class.forName}(s) : \texttt{@ClassVal}(\nu)}$$

$$\frac{fqn \text{ is the fully-qualified class name of the static type of } e}{e\texttt{.getClass()} : \texttt{@ClassBound}(fqn)}$$

$$\frac{(e : \texttt{@ClassBound}(\nu) \ \vee \ e : \texttt{@ClassVal}(\nu)) \quad s : \texttt{@StringVal}(\mu) \quad p : \texttt{@ArrayLen}(\pi)}{e\texttt{.getMethod}(s, p) : \texttt{@MethodVal}(\texttt{cn=}\nu, \texttt{mn=}\mu, \texttt{np=}\pi)}$$

$$\frac{e : \texttt{@ClassVal}(\nu) \quad p : \texttt{@ArrayLen}(\pi)}{e\texttt{.getConstructor}(p) : \texttt{@MethodVal}(\texttt{cn=}\nu, \texttt{mn="<init>"}, \texttt{np=}\pi)}$$

Intent type system rules

Subtyping

$$\text{(ST)}\quad \frac{\forall\, k \in \text{keys}(\tau_2).\ \ k \in \text{keys}(\tau_1)\ \wedge\ \tau_1[k] = \tau_2[k]}{\tau_1 <: \tau_2}$$

$$\text{(CP)}\quad \frac{\forall\, k \in \text{keys}(\tau_2).\ \ k \in \text{keys}(\tau_1)\ \wedge\ \tau_1[k] <: \tau_2[k]}{\tau_1 <_{\text{copyable}} \tau_2}$$

Well-formedness

$$\text{(OR)}\quad \frac{}{\texttt{void onReceive}(\tau\ \texttt{i})}\quad \textit{No precondition}$$

Typing judgments

$$\text{(SI)}\quad \frac{\forall\ \texttt{onReceive}(b,j).\ \ i:\tau_i \qquad \langle \texttt{sendIntent}(a,i), \texttt{onReceive}(b,j)\rangle \in \mathit{CCP} \qquad j:\tau_j \qquad \tau_i <_{\text{copyable}} \tau_j}{\text{sendIntent}(a,i) : \mathit{int}}$$

$$\text{(PE1)}\quad \frac{e:\tau \qquad v:\tau[k] \qquad k \in \text{keys}(\tau) \qquad s:\texttt{@StringVal}(k)}{e.\text{putExtra}(s,v):\tau}$$

$$\text{(PE2)}\quad \frac{e:\tau \qquad k \notin \text{keys}(\tau) \qquad e \text{ is unaliased} \qquad s:\texttt{@StringVal}(k)}{e.\text{putExtra}(s,v):\tau}$$

$$\text{(GE)}\quad \frac{e:\tau \qquad k \in \text{keys}(\tau) \qquad s:\texttt{@StringVal}(k)}{e.\text{getExtra}(s):\tau[k]}$$

Type inference rules

$$\frac{e.\text{putExtra}(s,v) \qquad e:\tau \qquad v:\sigma \qquad k \notin \text{keys}(\tau) \qquad e \text{ is unaliased} \qquad s:\texttt{@StringVal}(k)}{e:\tau \cup \{k \to \sigma\}}$$

$$\frac{e.\text{putExtra}(s,v) \qquad e:\tau \cup \{k \to \_\} \qquad v:\sigma \qquad e \text{ is unaliased} \qquad s:\texttt{@StringVal}(k)}{e:\tau \cup \{k \to \sigma\}}$$

Type inference evaluation

- Inference used on reflective calls
  - 52% required intra-procedural inference
  - 41% required inter-procedural inference
  - 7% cannot be resolved by any static analysis

- Inference used on send intent calls
  - 67% required intra-procedural inference
  - 21% required inter-procedural inference
  - 12% required a better aliasing analysis

Annotation burden

- Annotations are required for two reasons
  - The downstream analysis is a modular analysis
  - Express facts that no static analysis can infer
- The average time to add an annotation was one minute