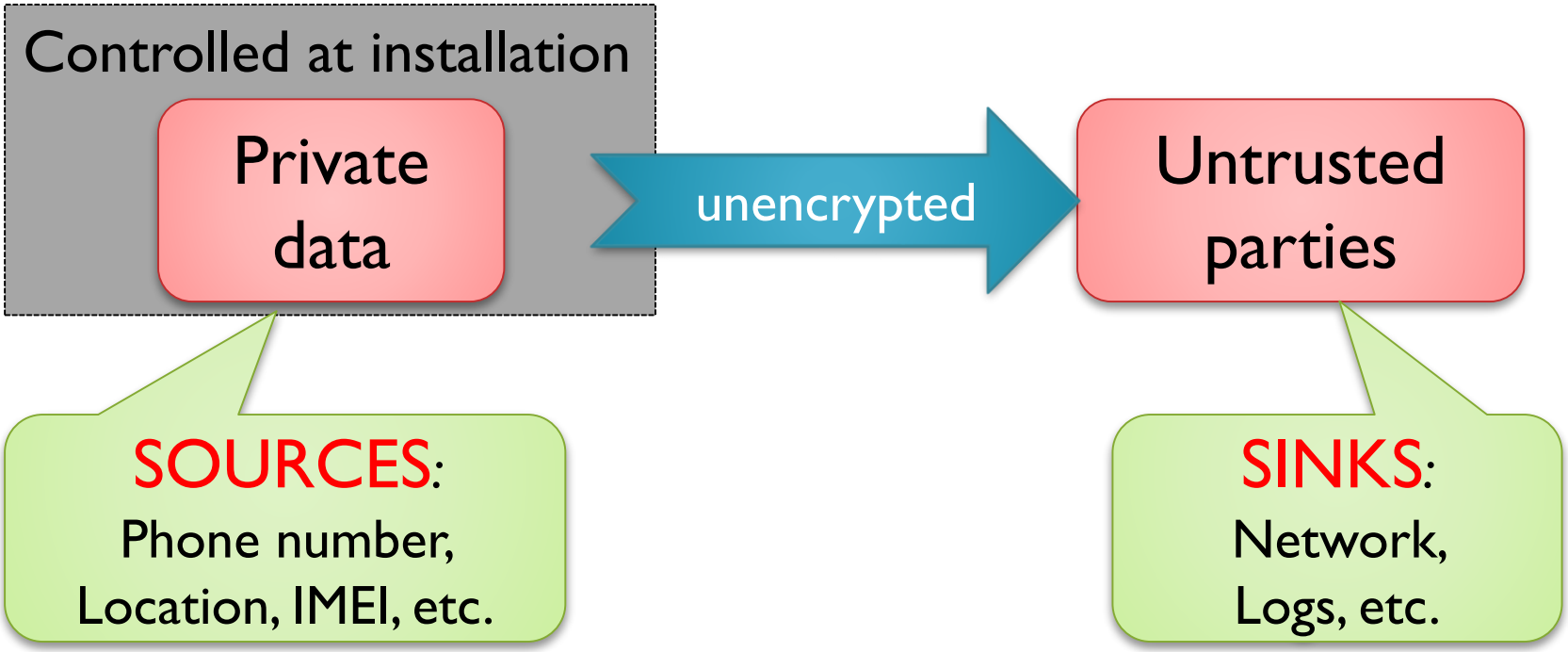
Scalable and Precise Taint Analysis for Android

Wei Huang[1,2],
Yao Dong[1], Ana Milanova[1],
Julian Dolby[3]

[1]Rensselaer Polytechnic Institute
[2]Google
[3]IBM Research

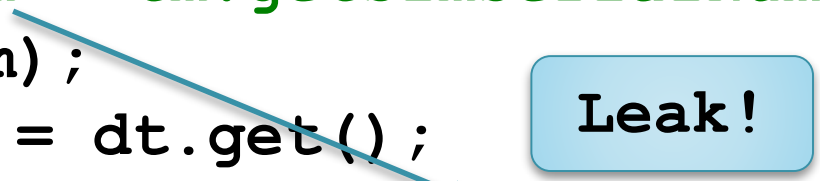
Taint Analysis for Android

- Tracks flow of private data



Motivating Example [From DroidBench]

```
public class Data {
   String f;
   String get() { return f; }
   void set(String p) { f = p; }
}
public class FieldSensitivity3 {
   protected void onCreate(Bundle b) {
      Data dt = new Data();
      …
      String sim = tm.getSimSerialNumber();
      dt.set(sim);
      String sg = dt.get();
      sms.sendTextMessage(…,sg,…); // sink
   }
}
```



Leak!

Solution – DFlow/DroidInfer

```
public class Data {
   String f;
   String get() { return f; }
   void set(String p) { f = p; }
}
public class FieldSensitivit
   protected void onCreate(Bund      b) {
      tainted Data dt = new Da      a();
      tainted String sim =
         tm.getSimSerialNumber()
      dt.set(sim);
      tainted String sg = dt.get();
      sms.sendTextMessage(…,sg,…); // sink
   }
}
```

Subtyping:
safe <: tainted

Source: the return value is tainted

Sink: the parameter is safe

Type error!

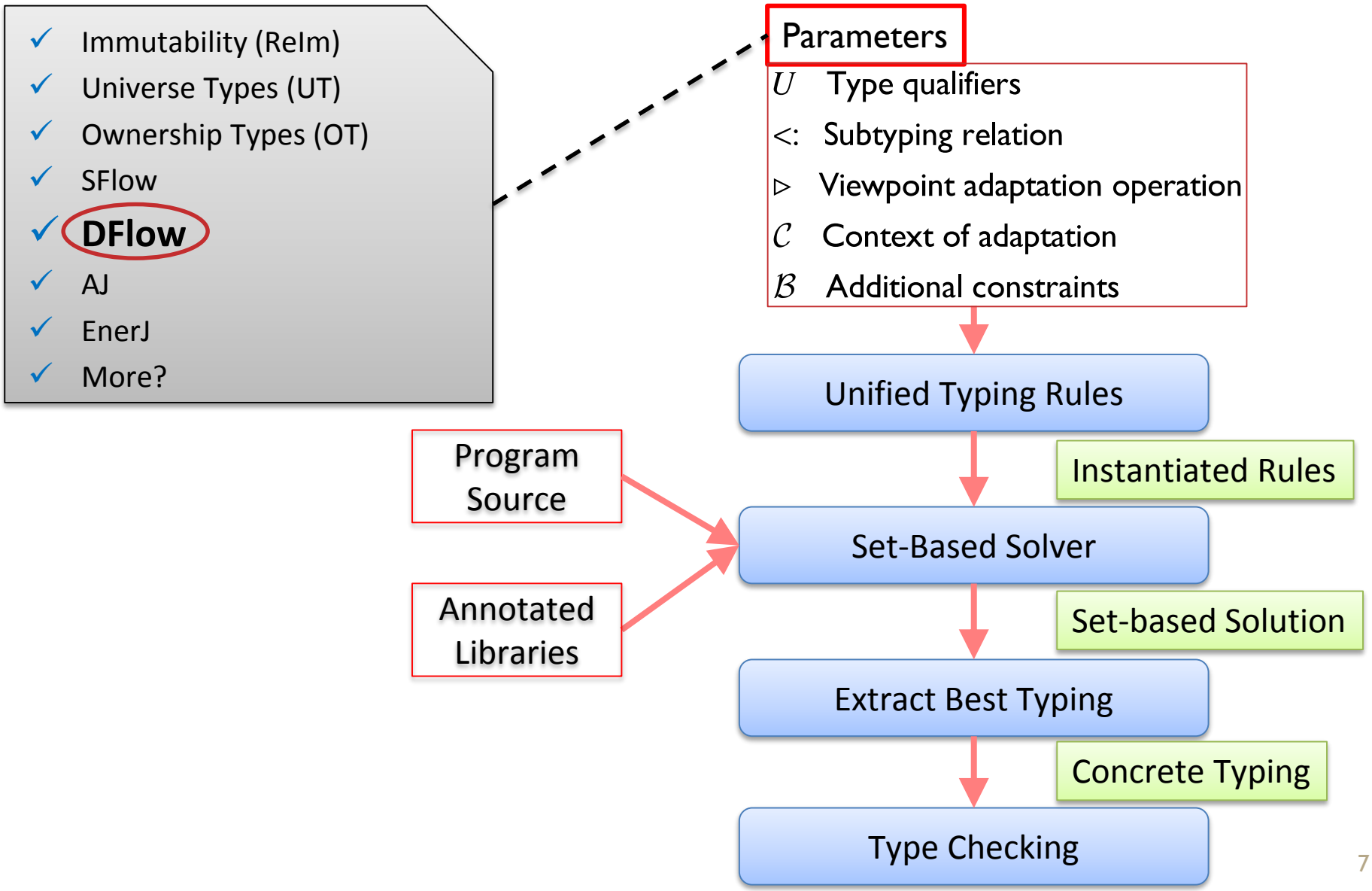
Contributions

- DFlow: A context-sensitive information flow type system
- DroidInfer: An inference algorithm for DFlow
- CFL-Explain: A CFL-reachability algorithm to explain type errors
- Effective handling of Android-specific features
- Implementation and evaluation
  - DroidBench, Contagio, Google Play Store

Inference and Checking Framework

- Build DFlow/DroidInfer on top of our type inference and checking framework
  - Programmers provide parameters to instantiate their own type system
    - Context sensitivity is encoded with viewpoint adaptation
  - Framework infers the "best" typing
    - If inference succeeds, this verifies the absence of errors
    - Otherwise, this reveals errors in the program

Framework Structure



DFlow

- Type qualifiers:
  - tainted: A variable x is tainted, if there is flow from a sensitive source to x
  - safe: A variable x is safe if there is flow from x to an untrusted sink
  - poly: The polymorphic qualifier, is interpreted as tainted in some contexts and as safe in other contexts
- Subtyping hierarchy:
  - safe <: poly <: tainted

DFlow Typing Rules (Simplified)

(TWRITE)

$$\frac{\Gamma(x) = q_x \quad \Gamma(y) = q_y \quad typeof(f) = q_f \quad q_x <: q_y \triangleright q_f}{\Gamma \vdash y.f = x}$$

(TREAD)

$$\frac{\Gamma(x) = q_x \quad \Gamma(y) = q_y \quad typeof(f) = q_f \quad q_y \triangleright q_f <: q_x}{\Gamma \vdash x = y.f}$$

(TCALL)

$$\frac{\Gamma(x) = q_x \quad \Gamma(y) = q_y \quad \Gamma(z) = q_z \quad typeof(m) = q_{this}, q_p \rightarrow q_{ret} \quad q_y <: q^i \triangleright q_{this} \quad q_z <: q^i \triangleright q_p \quad q^i \triangleright q_{ret} <: q_x}{\Gamma \vdash x = y.m^i(z)}$$

Inference Example

```
public class Data {
    {poly, tainted} String f;
    {safe, poly, tainted} String get({safe, poly, tainted} Data
this) {return this.f;}
    void set({safe, poly, tainted} Data this,
             {safe, poly, tainted} String p) {this.f = p;}
}
public class FieldSensitivity3 {
    protected void onCreate(Bundle b) {
        {safe, poly, tainted} Data dt = new Data();
        {safe, poly, tainted} String sim =
            tm.getSimSerialNumber(); // source
        dt.set(sim);
        {safe, poly, tainted} String sg = dt.get();
        sms.sendTextMessage(…,sg,…); // sink
    }
}
```

Inference Example

```
public class Data {
    {poly, tainted} String f;
    {safe, poly, tainted} String get({safe, poly, tainted} Data
this) {return this.f;}
    void set({safe, poly, tainted} Data this,
             {safe, poly, tainted} String p) {this.f = p;}
}
public class FieldSensitivity3 {
    protected void onCreate(Bundle b)
        {safe, poly, tainted} Data dt = ne    ta();
        {safe, poly, tainted} String sim
            tm.getSimSerialNumber()    // source
        dt.set(sim);
        {safe, poly, tainted} String sg = dt.get();
        sms.sendTextMessage(…,sg,…); // sink
    }
}
```

sg <: $q \triangleright$ safe

Inference Example

```
public class Data {
    {poly, tainted} String f;
    {safe, poly, tainted} String get({safe, poly, tainted} Data
this) {return this.f;}
    void set({safe, poly, tainted} Data this,
             {safe, poly, tainted} String p) {this.f = p;}
}
public class FieldSensitivity3 {
    protected void onCreate(Bundle b) {
        {safe, poly, tainted} Data dt = new
        {safe, poly, tainted} String sim =
            tm.getSimSerialNumber(); // source
        dt.set(sim);
        {safe, poly, tainted} String sg = dt.get();
        sms.sendTextMessage(…,sg,…); // sink
    }
}
```

Type Error!

dt <: sg

CFL-Explain

- Type error:

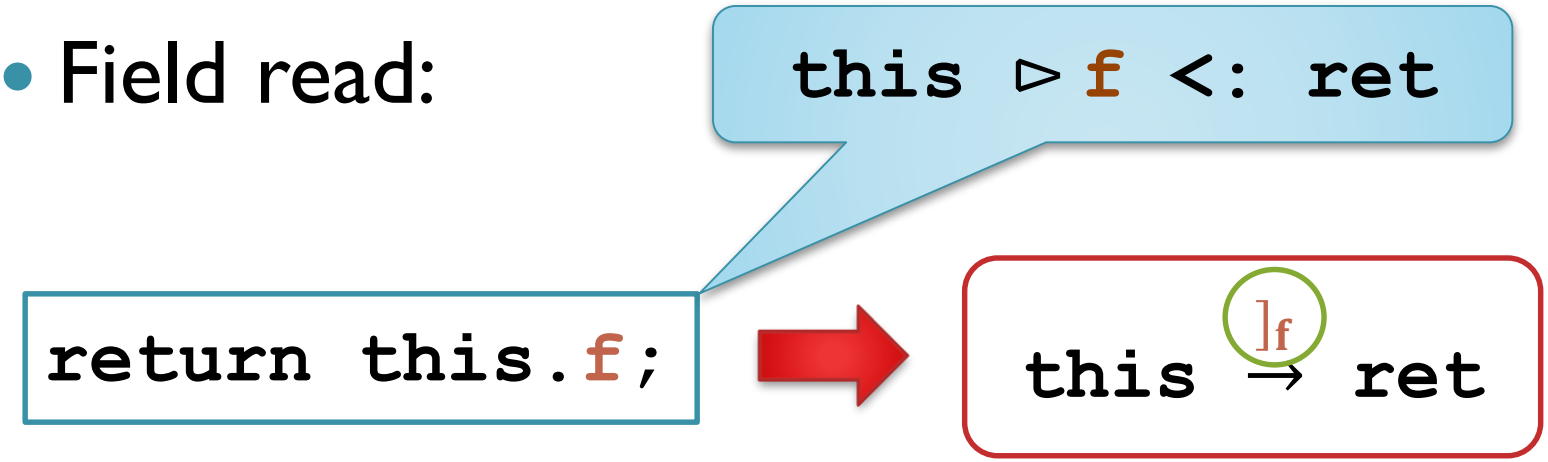
$$q \triangleright \texttt{ret}_{\texttt{getSimSerialNumber}}\ \{\textbf{tainted}\}\ \texttt{<:}\ \texttt{sim}\ \{\textbf{safe}\}$$

- Construct a dependency graph based on CFL-reachability

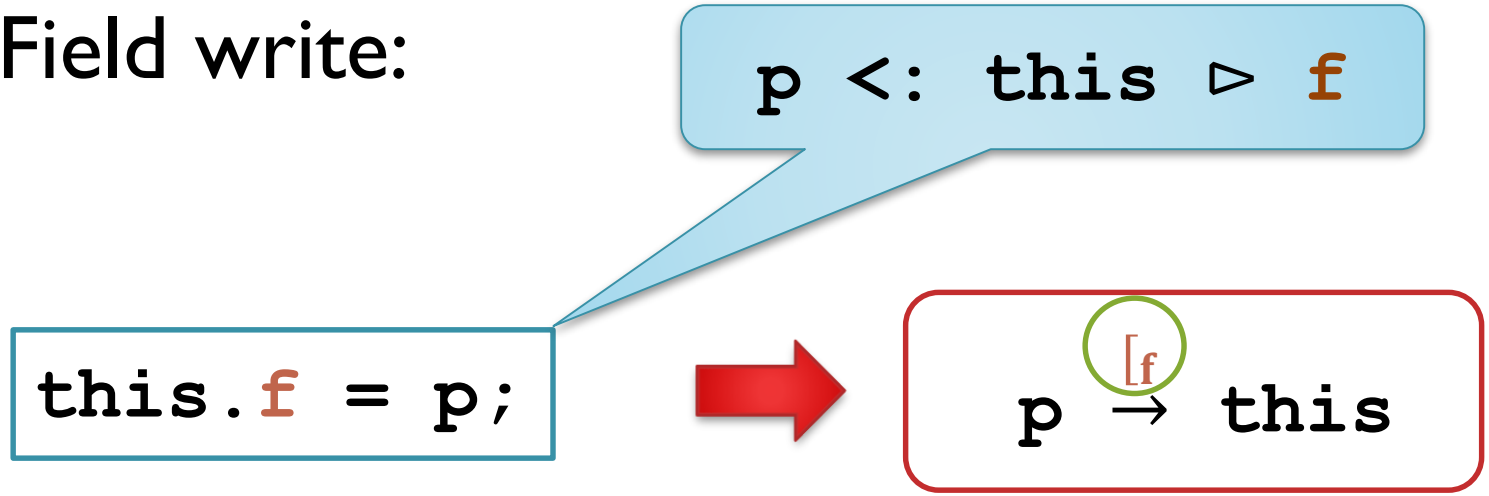
- Map a type error into a source-sink path in the graph

CFL-Explain – Construct Graph

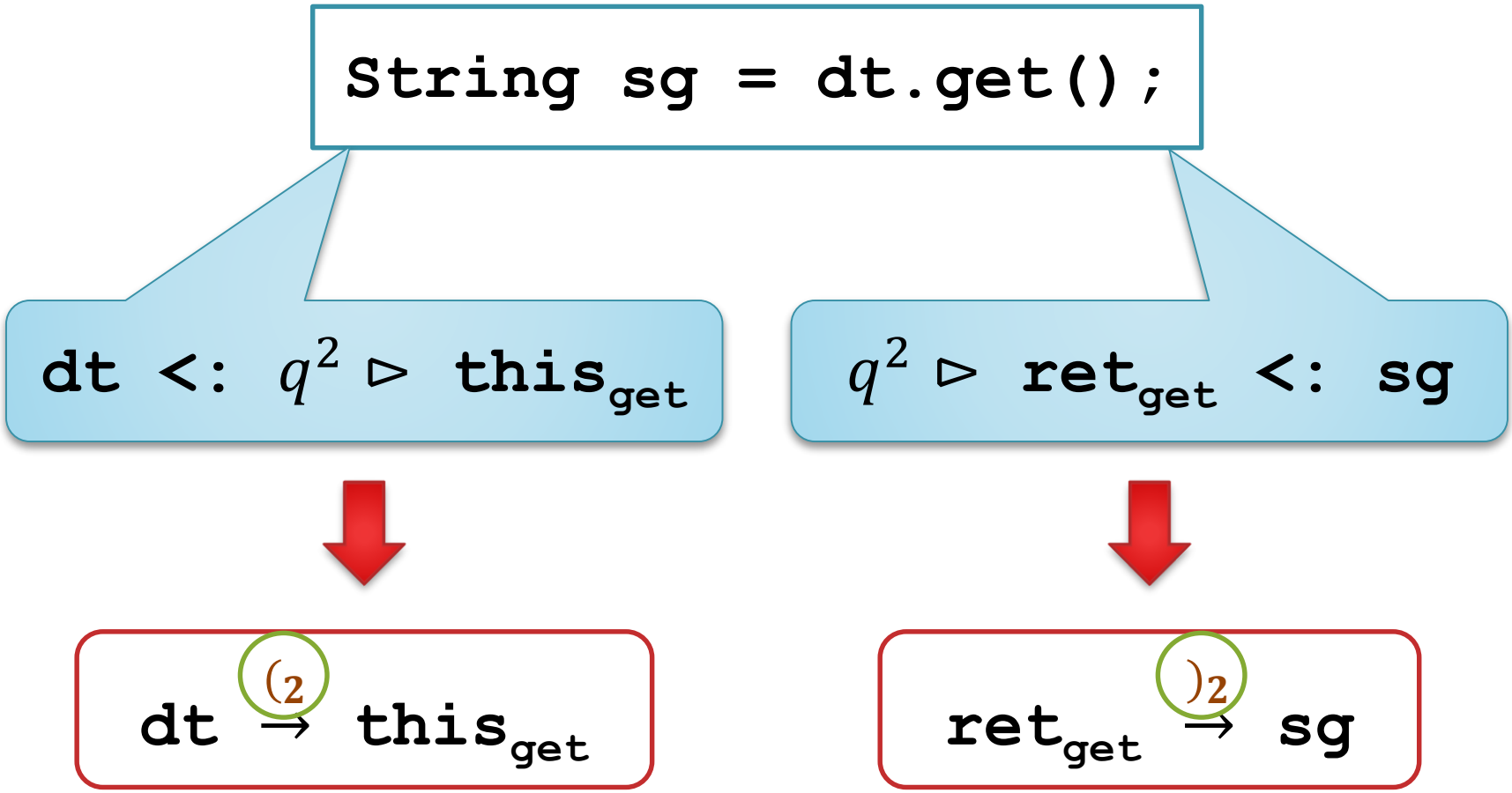
- Field read:



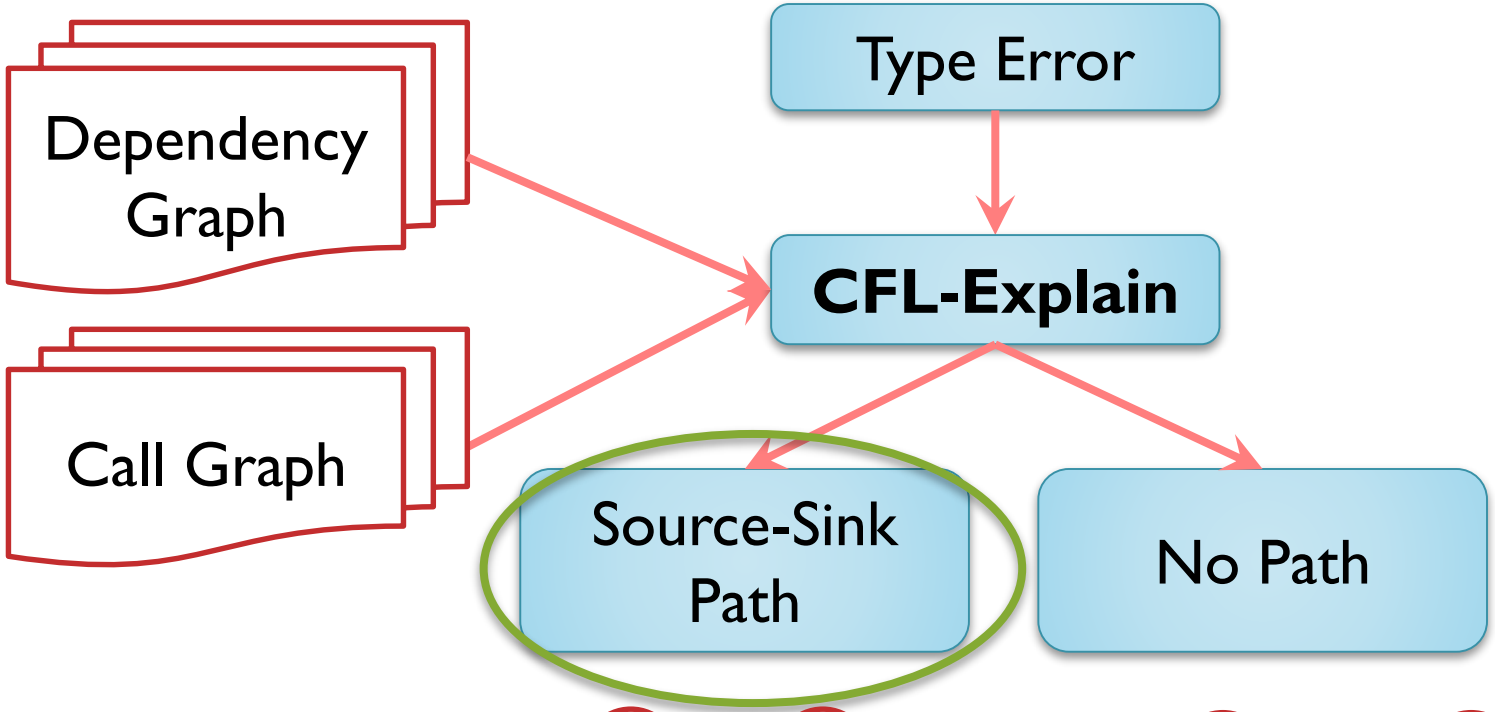
- Field write:



CFL-Explain – Construct Graph (Cont'd)



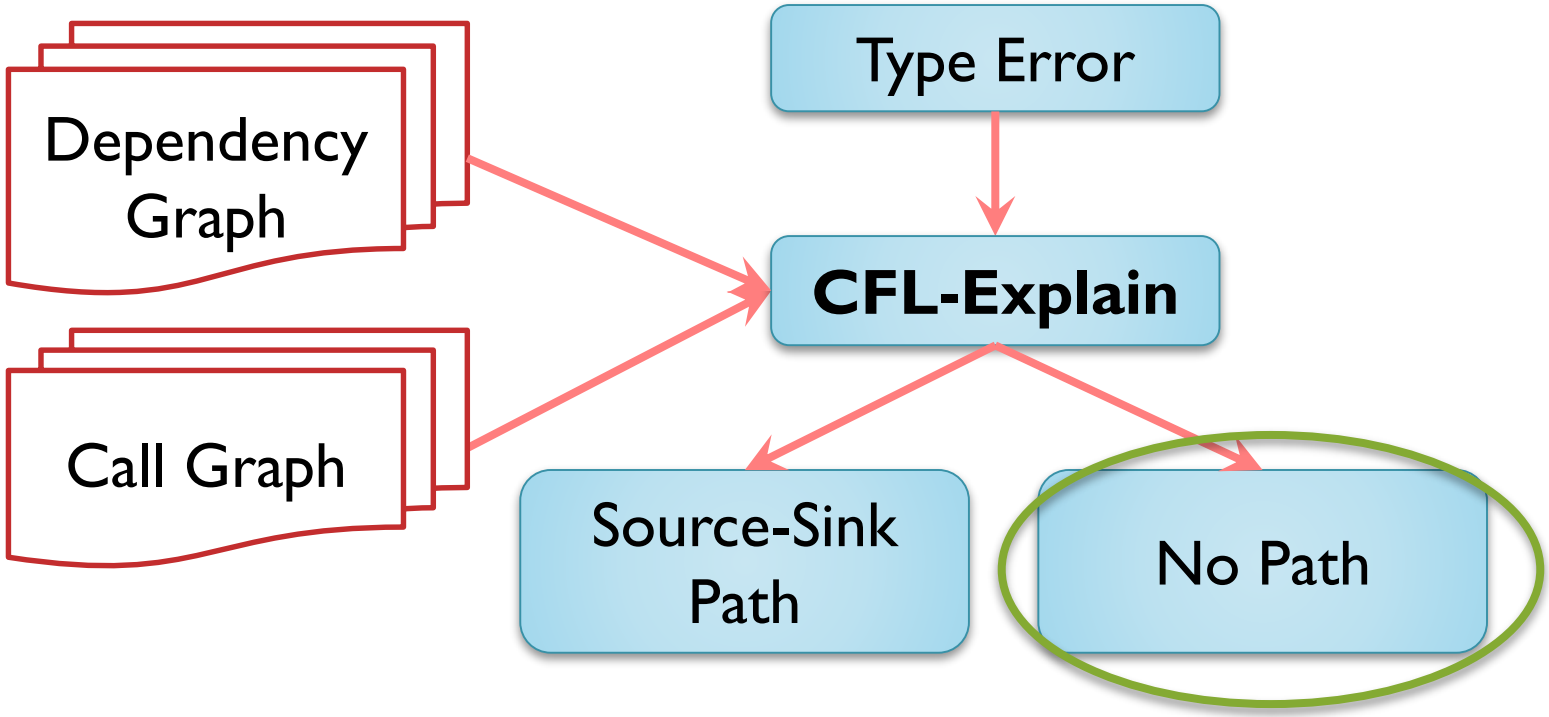
CFL-Explain Output



$$\texttt{source} \rightarrow \texttt{sim} \xrightarrow{(_4} \texttt{p} \xrightarrow{[_f} \texttt{this}_{\texttt{set}} \xrightarrow{)_4} \texttt{dt} \xrightarrow{(_2}$$

$$\texttt{this}_{\texttt{get}} \xrightarrow{]_f} \texttt{ret}_{\texttt{get}} \xrightarrow{)_2} \texttt{sg} \rightarrow \texttt{sink}$$

CFL-Explain Output



Reasons:

- Unreachable methods on the call graph
- False positive due to partial field insensitivity

Outline

- DFlow type system
- Inference algorithm for DFlow
- CFL-Explain
- Handling Android-specific features
- Implementation and evaluation

Android-Specific Features

- Libraries
  - Flow through library method
- Multiple Entry Points and Callbacks
  - Connections among callback methods
- Inter-Component Communication(ICC)
  - Explicit/implicit Intents

Libraries

- Insert annotations into Android library
  - source → {tainted} sink → {safe}
- Type all parameters/returns of library methods as
  - poly, poly → poly
- Method **n** overrides **m**:

$$(this_n, p_n \rightarrow ret_n) <: (this_m, p_m \rightarrow ret_m)$$

$$this_m <: this_n \quad p_m <: p_n \quad ret_n <: ret_m$$

Example

- **Library source:**
  `LocationListener.onLocationChanged (tainted Location l)`
- **Type library method as:**
  `poly double getLatitude (poly Location this)`

`l <: loc`

`loc <: q ▷ poly`
`q ▷ poly <: lat`

```
public class MyListener {
   @Override
   public void onLocationChanged(Location
loc) {
      double lat = loc.getLatitude();
      Log.d(…, "Latitude: " + lat); // sink
   }
}
```

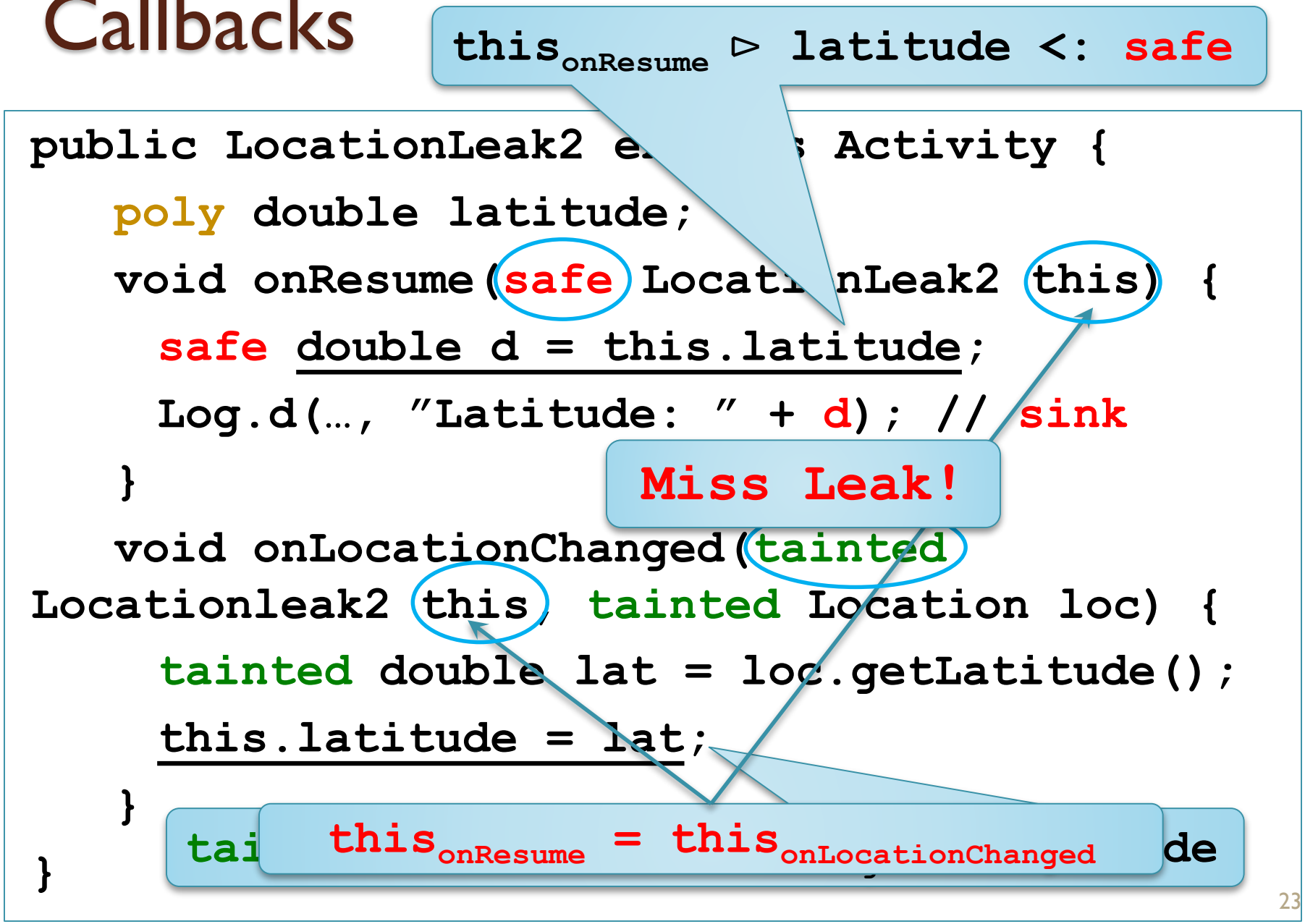
`loc <: lat`
Type error: leak!

Callbacks

- Component objects (e.g., Activity) are instantiated by the Android framework
- No explicit instance to "link" the **`this`** parameters of callback methods
- DroidInfer creates equality constraints for **`this`** parameters to "link" callback methods

$$\texttt{this}_{\texttt{callbackMethod1}} = \texttt{this}_{\texttt{callbackMethod2}}$$

Callbacks


this onResume ▷ latitude <: safe
public LocationLeak2 e Activity {
poly double latitude;
void onResume(safe Locati nLeak2 this) {
safe double d = this.latitude;
Log.d(…, "Latitude: " + d); // sink
}
Miss Leak!
void onLocationChanged(tainted
Locationleak2 this, tainted Location loc) {
tainted double lat = loc.getLatitude();
this.latitude = lat;
}
tai
this onResume = this onLocationChanged
de
}

Inter-Component Communication (ICC)

- Android components interact through Intents
- Explicit Intent
  - Have an explicit target component
  - DroidInfer connects them using placeholders
- Implicit Intent
  - Do not have a target component
  - DroidInfer conservatively considers them as sinks

ICC Example

```
public class SmsReceiver extends BroadcastReceiver {
    public void onReceiver(Context c, Intent i) {
        tainted String s = …;    // source
        Intent it = new Intent(c, TaskService.class);
        it.putExtra("data", s);
        startService(i);
    }
}
public class TaskService exennds Service {
    public void onStart(Intent it, int d) {
        String body = it.getSerializableExtra("data");
        list.add(body);
        Entity e = new UrlEncodedFormEntity(list, "UTF8");
        post.setEntity(e); // sink
    }
}
```

ICC Example

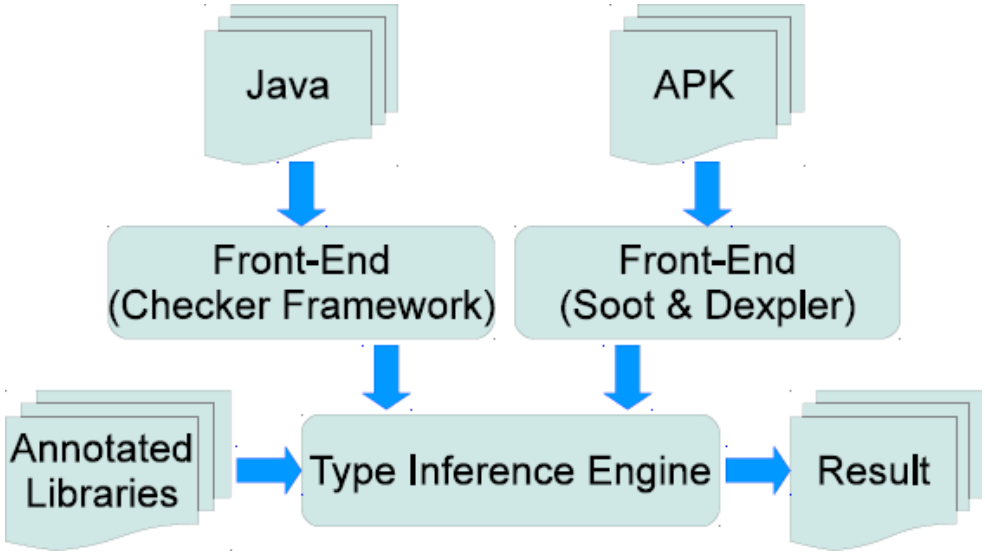
```
public class SmsReceiver extends BroadcastReceiver {
    public void onReceiver(Context c, Intent i) {
        tainted String s = …;    // source
        TaskService_Intent it = new TaskService_Intent();
        TaskService_Intent.data = s; // it.putExtra("data", s);
        startService(i);
    }
}
public class TaskService exennds Service {
    public void onStart(Intent it, int d) {
        String body = TaskService_Intent.data; //
        list.add(body);          //it.getSerializableExtra("data");
        Entity e = new UrlEncodedFormEntity(list, "UTF8");
        post.setEntity(e); // sink
    }
}
```

Outline

- DFlow type system
- Inference algorithm for DFlow
- CFL-Explain
- Handling Android-specific features
- Implementation and evaluation

Implementation

- Built on top of Soot [Vall´ee-Rai et al. CASCON'99] and Dexpler [Bartel et al. SOAP'12]

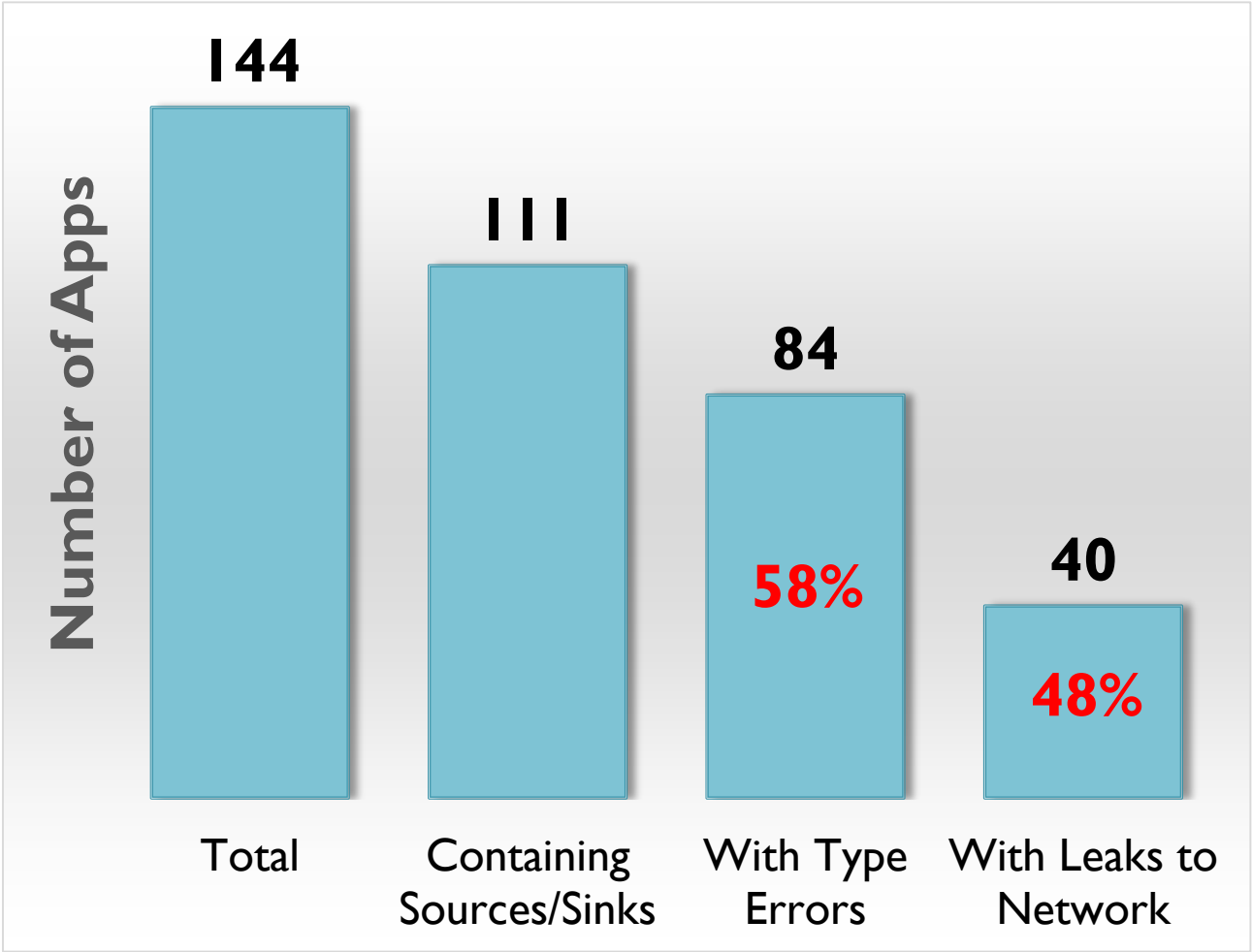


- Publicly available at
  - https://github.com/proganalysis/type-inference

Evaluation

- DroidBench 1.0
  - Recall: 96%, precision: 79%
- Contagio
  - Detect leaks from 19 out of total 22 apps
- Google Play Store
  - 144 free Android apps (top 30 free apps)
  - Maximal heap size: 2 GB
  - Time: 139 sec / app on average
  - False positive rate: 15.7%

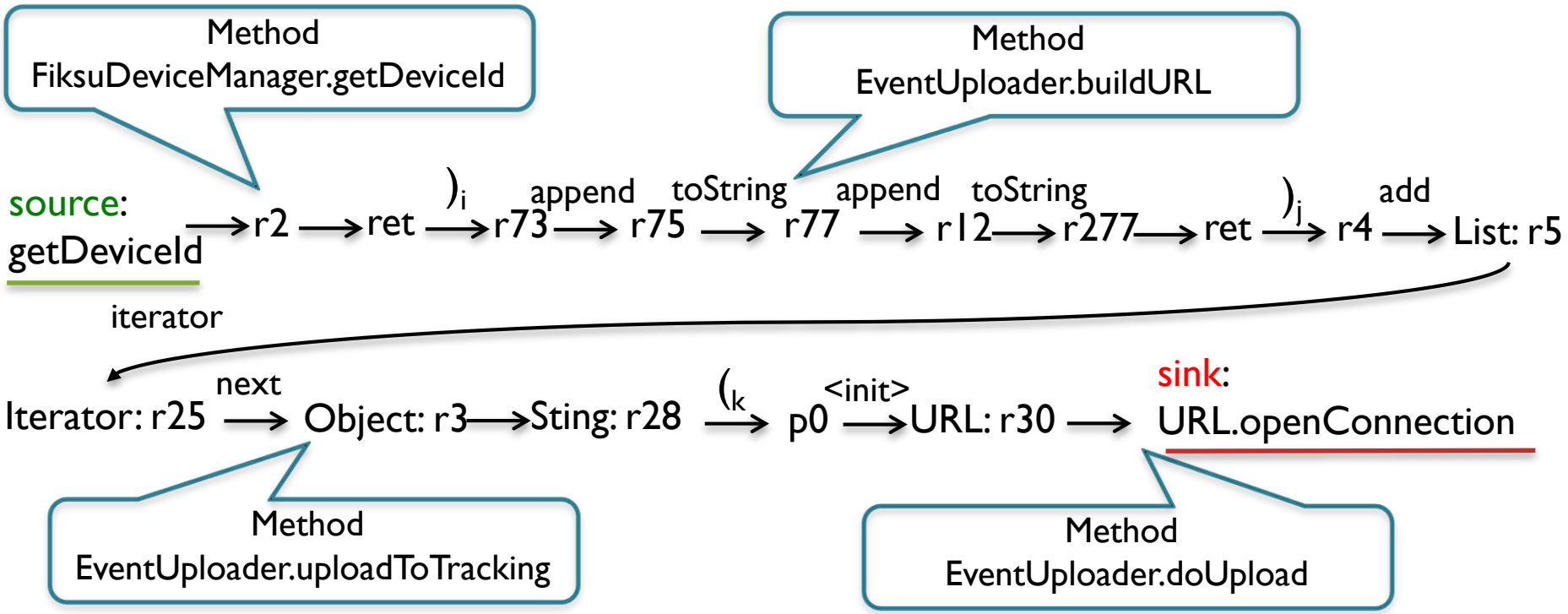
Results for Google Play Store Apps



Runtime Results

- Run 10 random apps on Android phone/tablet
- Collect and analyze logs using Android Device Monitor
- Cover 14 out of 76 true flows in 8 apps (18.4%)

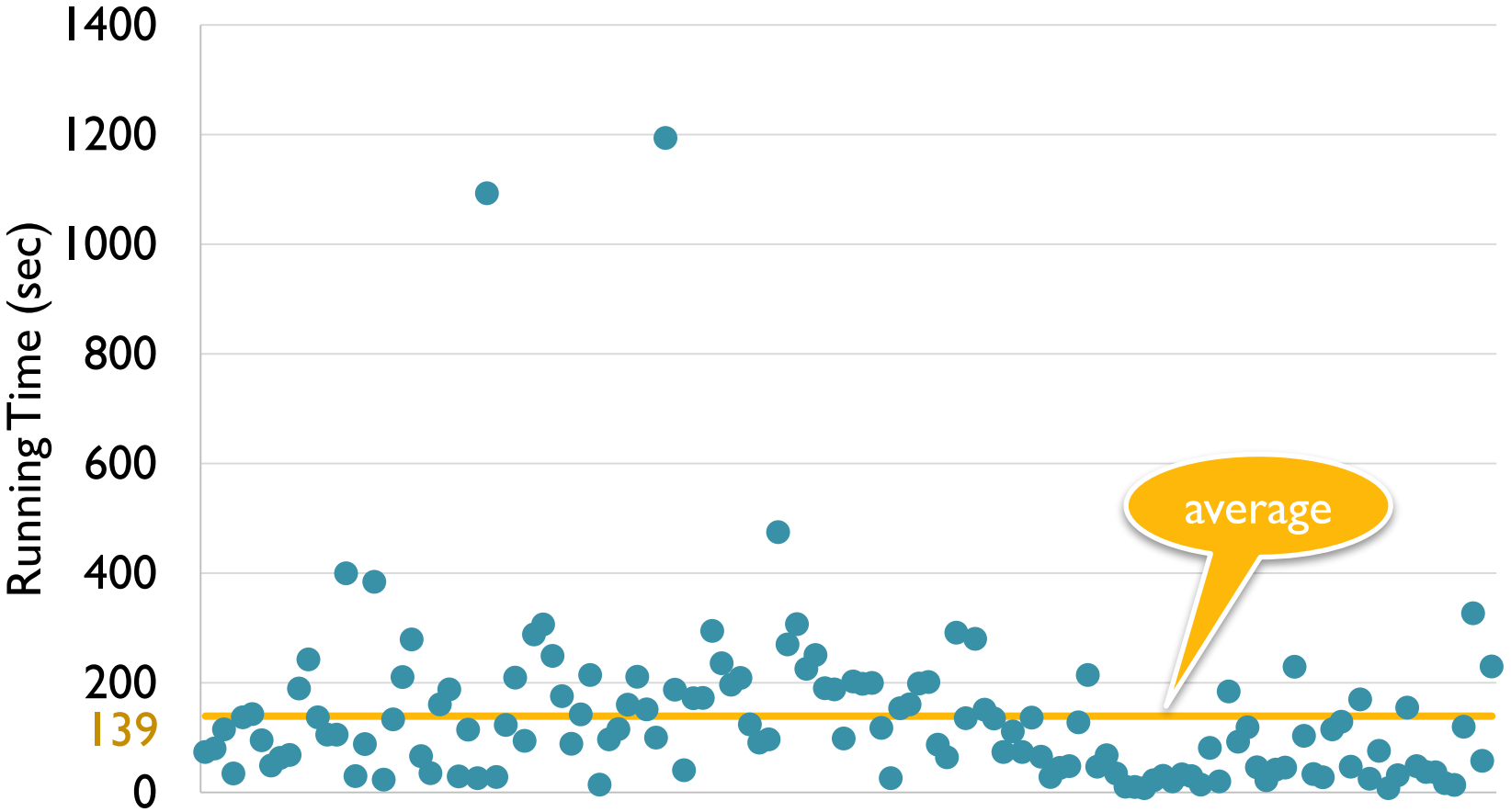
Runtime Example



A source-sink path in **Zillow** App

DroidInfer Running Time

- Maximal heap size is set to 2GB!



Related Work

- FlowDroid [Arzt et al. PLDI'14]
  - Flow-sensitive
  - Memory-intensive, reports no network flows
- IFT [Ernst et al. CCS'14]
  - Enable collaborative verification of information flow
  - Need source code of apps
  - Annotation burden: 6 annotations per 100 LOC
- IccTA [Li et al. ICSE'15]
  - Focus on inter-component detection (ICC)
- Others
  - LeakMiner, Cassandra, SCANDAL, AndroidLeaks, CHEX, SCanDroid, Epicc, and so on

Conclusions

- DFlow and DroidInfer: context-sensitive information flow type system and inference
- CFL-reachability algorithm to explain type errors
- Effective handling of Android-specific features
- Implementation and evaluation

- Publicly available at
  - https://github.com/proganalysis/type-inference