S.T.A.M.P

Program Analysis for Android

1. Objective and goals

What?
1) Tell whether a mobile app includes malicious behavior or not.
2) Verify the absense of malicious behavior.

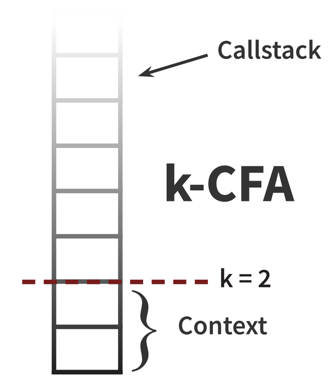
Why is it hard?
Context dependent: "malicious behavior" depends on what the app is supposed to do.

How?
Allow human auditors to inspect application behavior: *information flow*, values, usage of dangerous features. Open the blackbox of apk + permissions.

2. Static Analysis

Two main phases:
Points-to analysis (aliasing).
Information (taint) flow analysis: uses model annotations and points-to results.

Analysis charactersitics
- Bottom-up exhaustive solving
- Field-sensitive
- Bounded context sensitive (k-CFA)
- Path, flow and object-insensitive

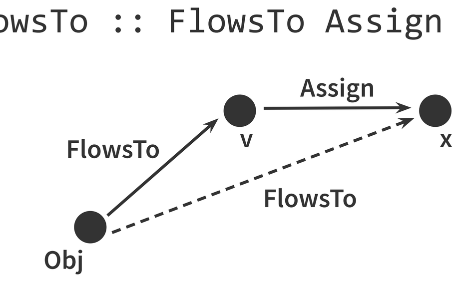



```
...
FlowsTo(O,X) :-
  FlowsTo(O,V),
  Assign(V,X)
...
```

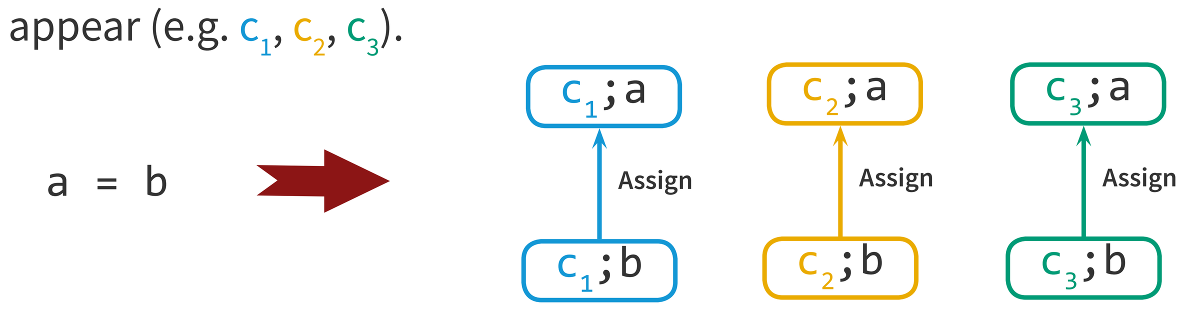
Old implementation (using Chord)
Facts represented as BDD relations.
Datalog-based analysis using a BDD solver.
Produces correct source-sink flows, *but* makes it harder to extract meaningful flow paths.

New implementation (SolverGen)
Soot based (www.sable.mcgill.ca/soot).
Explicitly represented program graphs.
Transitive CFL-based graph reachability: we only add edges if they form a word in the grammar.

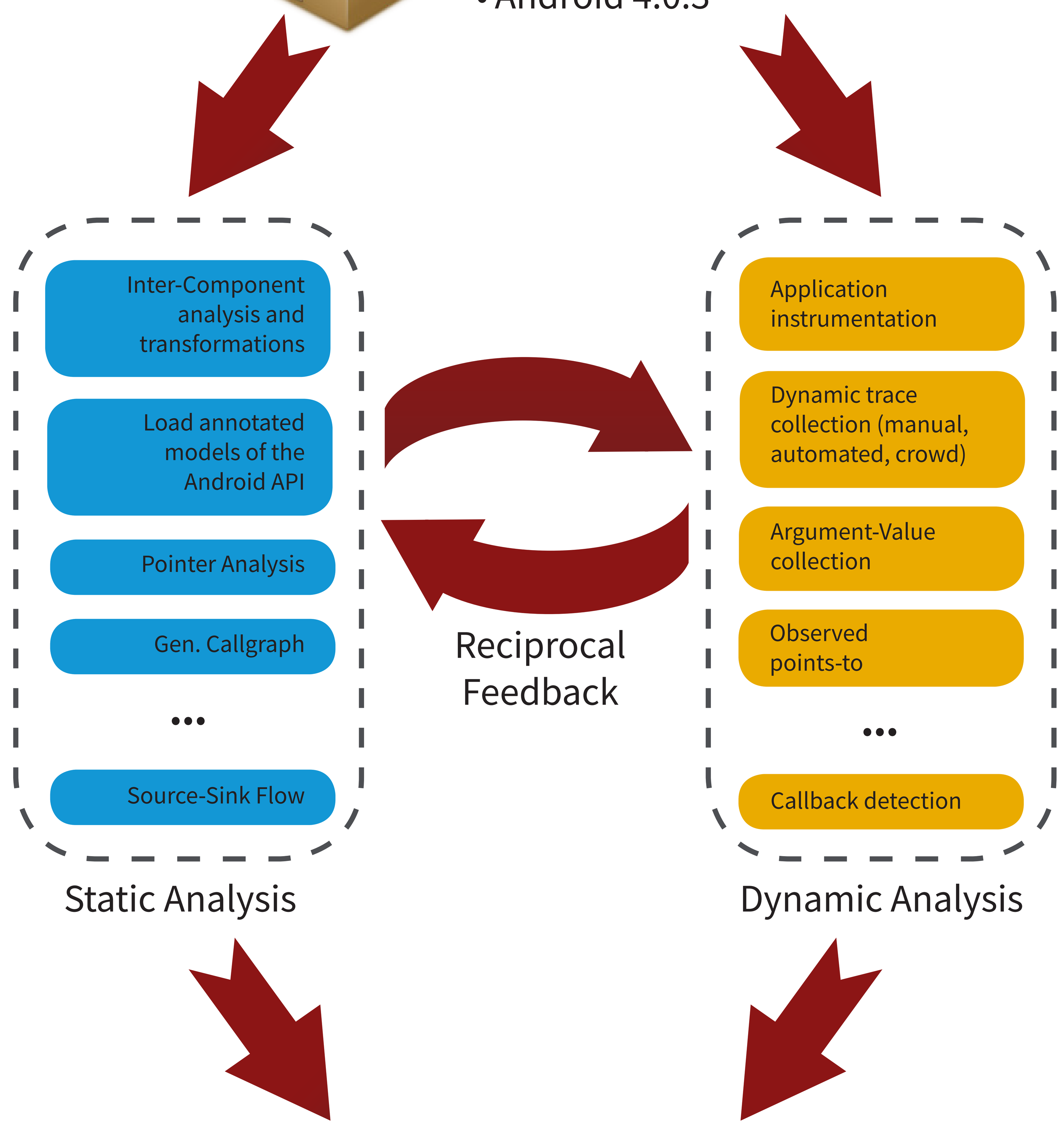
`FlowsTo :: FlowsTo Assign`

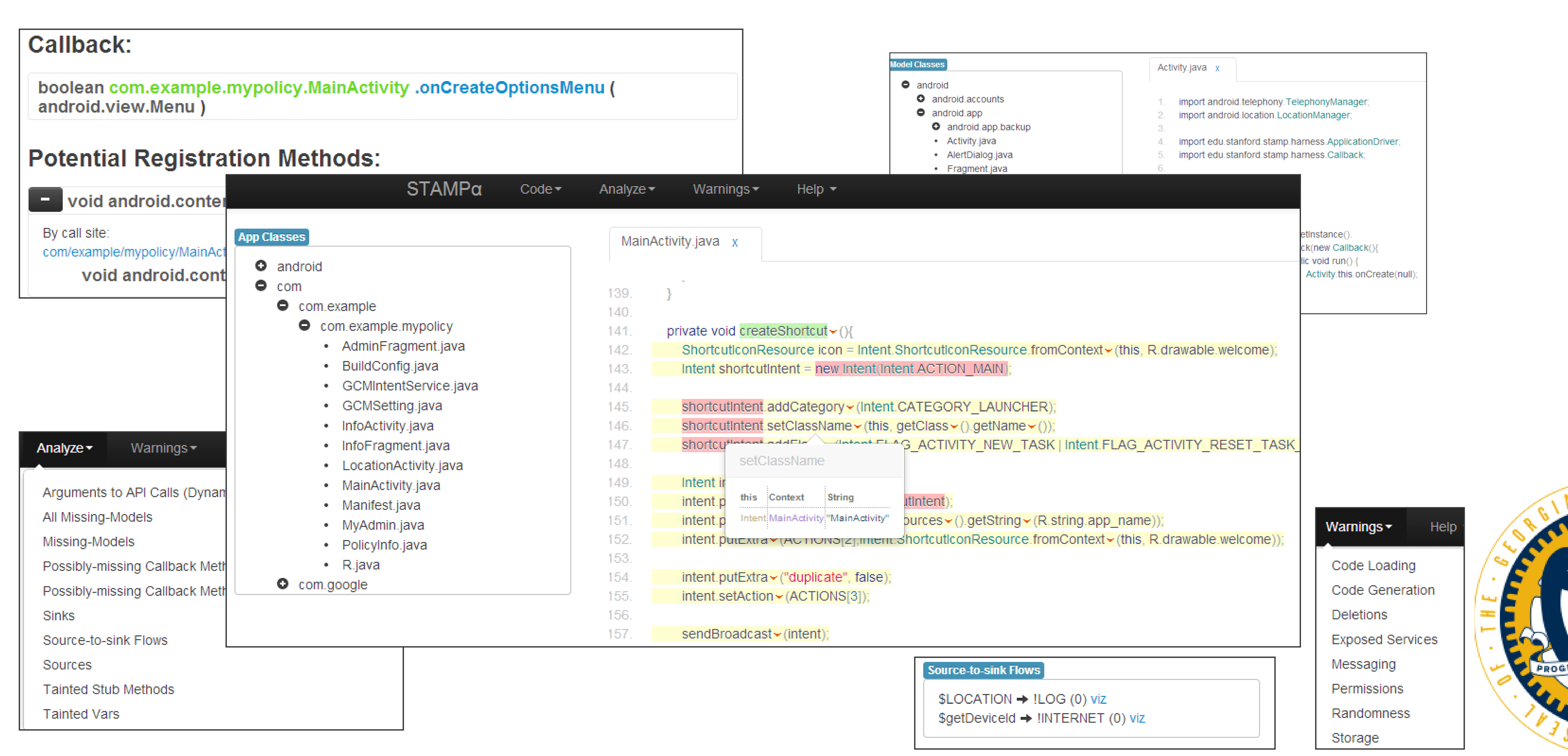


Conceptually: Variables are copied for every context in which they appear (e.g. c_1, c_2, c_3).



Android App: Source or APK
- AndroidManifest.xml
- Resources, layout files, libraries
- Android 4.0.3





3. Models

Why model the API?
Static analysis does not scale well to the size of the Android platform.
Language (native calls) & process (Android's root process) boundaries.

```
@STAMP(flows={@Flow(from="$LOCATION",
to="@return")})
   private Location getLocation() {
      return new Location((String)null);
   }
```

Source and sink flow annotations:
Model sensitive data comming in and out of the app

Callback registration:
Android lacks a main method. Callbacks are registered with our execution harness.

```
ApplicationDriver.getInstance().
   registerCallback(new CB() {...
      listener.onLocChanged(...);
  ...}});
```

We have models for thousands of Android API 4.0.3 methods

4. Dynamic Analysis

Step #1: Instrument app code:
DroidRecord modifies the app's bytecode to record: method calls, loads, stores, etc. Android platform code is not instrumented.

Step #2: Execute modified app:
Manually, automatically (DynoDroid) or crowdsourced (MTurk).

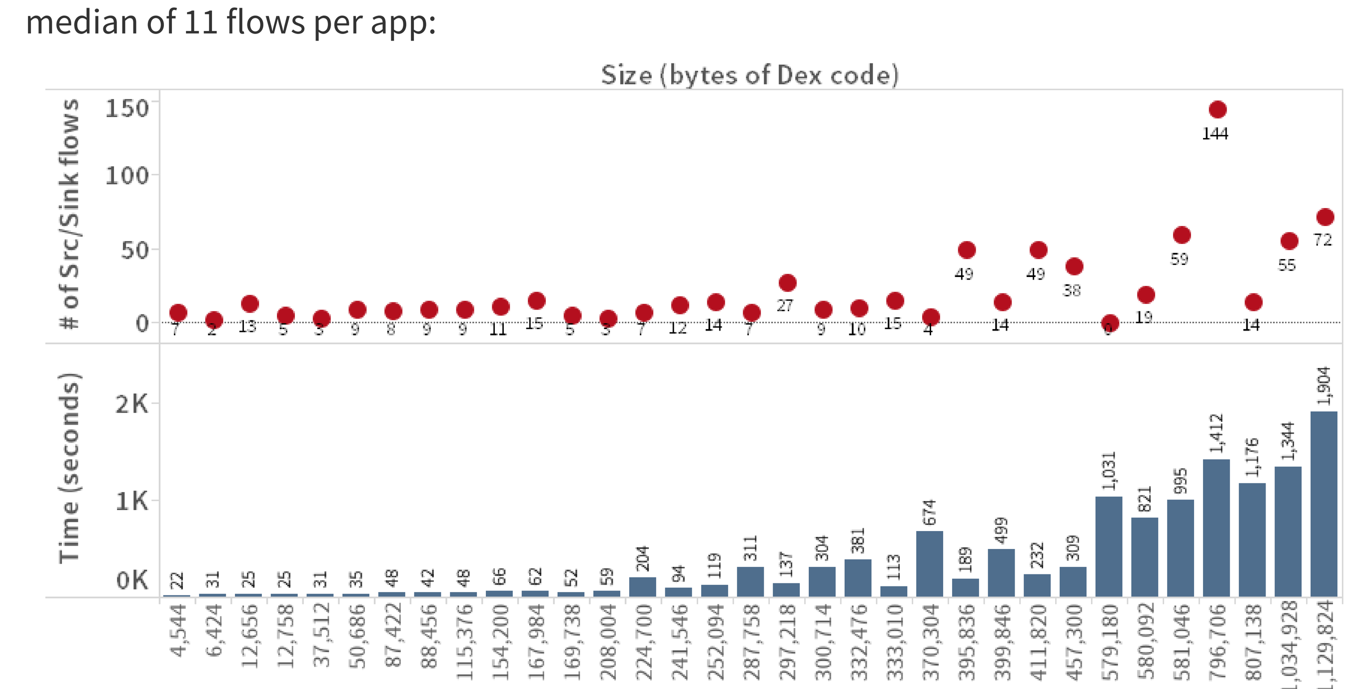
Step #3: Analyse resulting execution trace:
One log, many analyses: callback detection, observed values, aliases.

Example: Inter-Component Analysis
Use observed values (dynamic) and reaching definitions (static) to specialize strings in Intent and Bundle operations.

```
s = enc("secret"); bundle.put(dec(s), deviceId); ➡ bundle.put_secret(deviceId);
```

5. Results

We ran STAMP on 39 real apps provided by an antivirus vendor, the tool pinpointed a median of 11 flows per app:



Stanford University
Saswat Anand, Manolis Papadakis, Lazaro Clapp, Rahul Sharma, John C. Mitchell
Jason Franklin, Patrick Mutchler, Osbert Bastani, Bryce Cronkite-Ratcliff, Alex Aiken

Georgia Tech
Aravind Machiry, Xin Zhang
Ravi Mangal, Mayur Naik

U.T. Austin
Yu Feng, Isil Dillig
Thomas Dillig