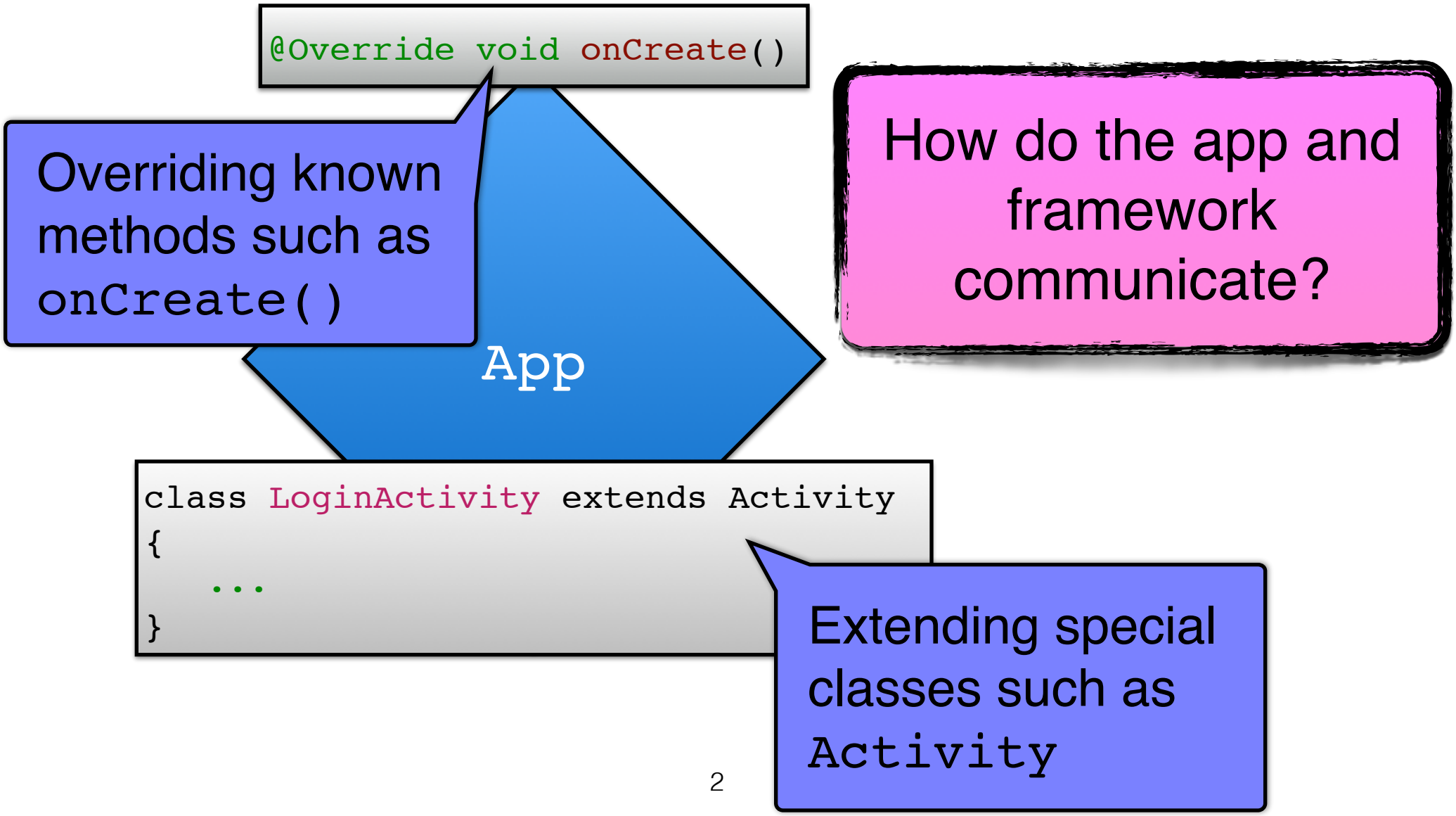
Droidel: A General Approach to Android Framework Modeling

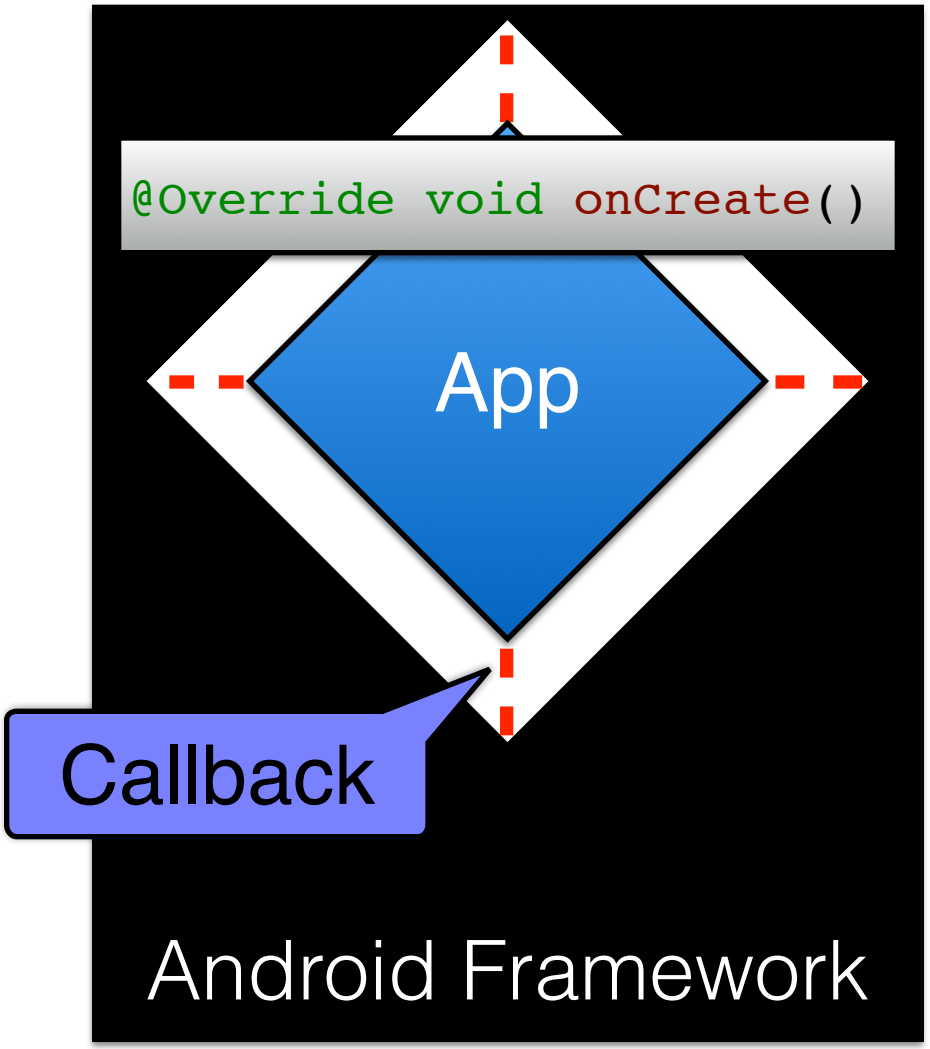
Sam Blackshear Alexandra Gendreau Bor-Yuh Evan Chang

University of Colorado Boulder

Implementing an Android app

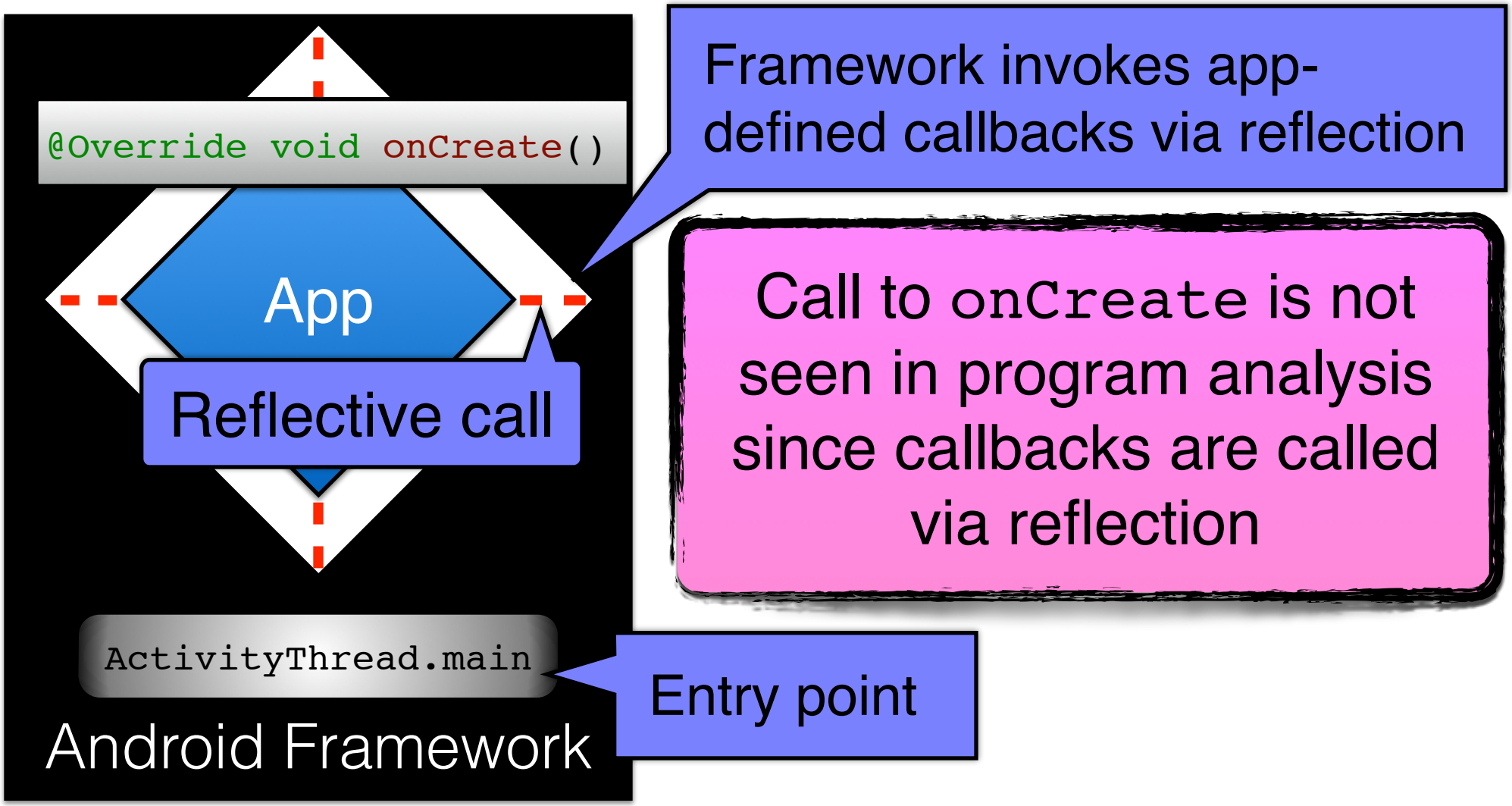


How does the app hook into the Android framework?

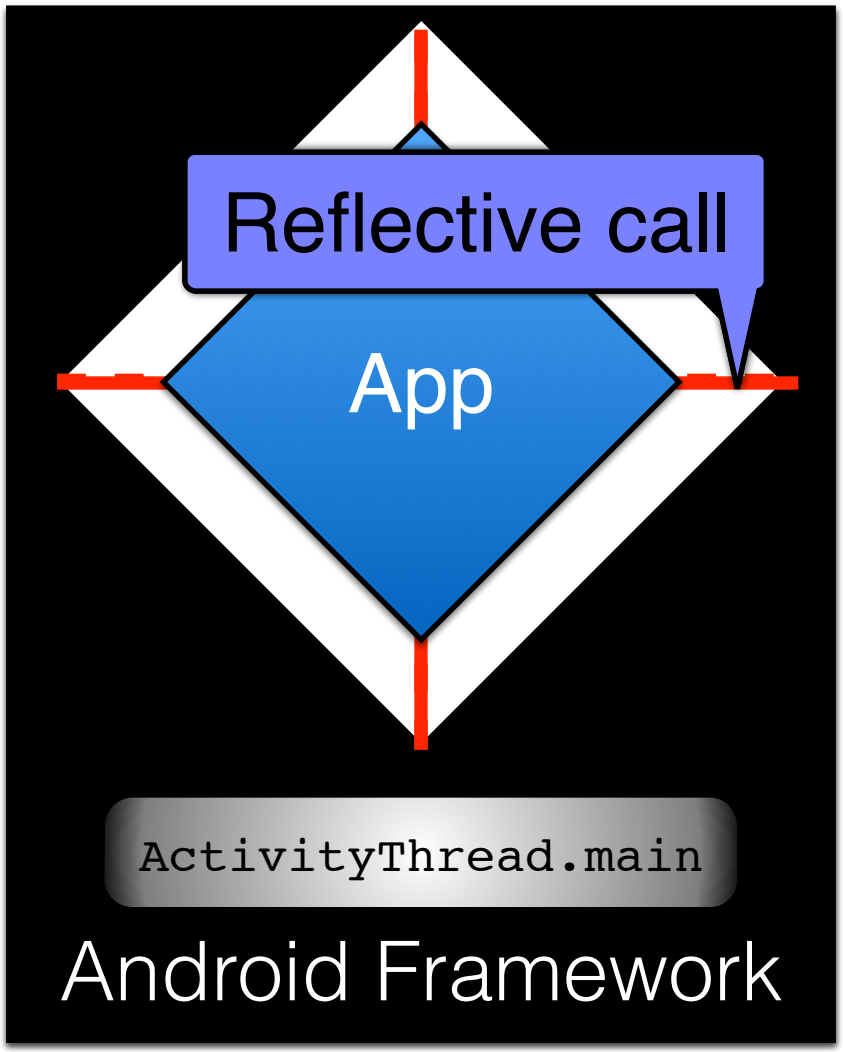


Events in the Android framework trigger callbacks on the app

Execution of an Android app



Models summarize reflective calls




```
void loginActivityHarness() {
    Activity a = new LoginActivity();
    ...
}
void androidMain() {
    ...
    loginActivityHarness();
    ...
}
```

Perfect Model:
Replace all reflective calls with explicit, app specific calls

The trouble with modeling

The Android framework is **big**

Client Specific Models

The Android framework is **complex**

Requires careful modeling of execution context

Framework is big

Android Framework

Models are client specific and thus only summarize reflective calls relevant to a particular analysis making it difficult to reuse models

behaviors of interest

Model

Framework is Complex

To be sound, the harness must invoke `l.onCancel()` with respect to this `Activity`

```
class LoginActivity extends Activity {
    …
    @Override void onCreate() {
        OnCancelListener l = …
    }
}
```

Needs to over approximate behaviors of interest

```
void loginActivityHarness() {
    Activity a = new LoginActivity();
    …
}
```

Every harness model must soundly set up the execution context

Goal: a general purpose modeling approach

The pr
the

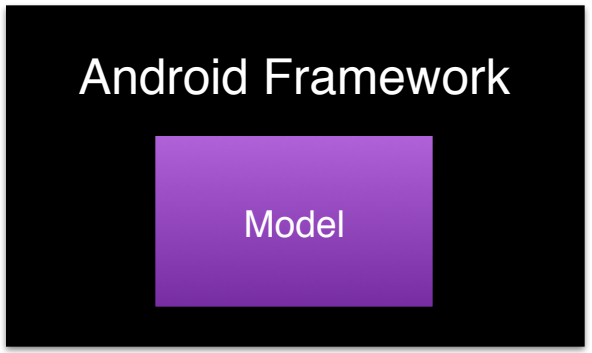
We present Droidel, a framework model for Android, built using these philosophies

del

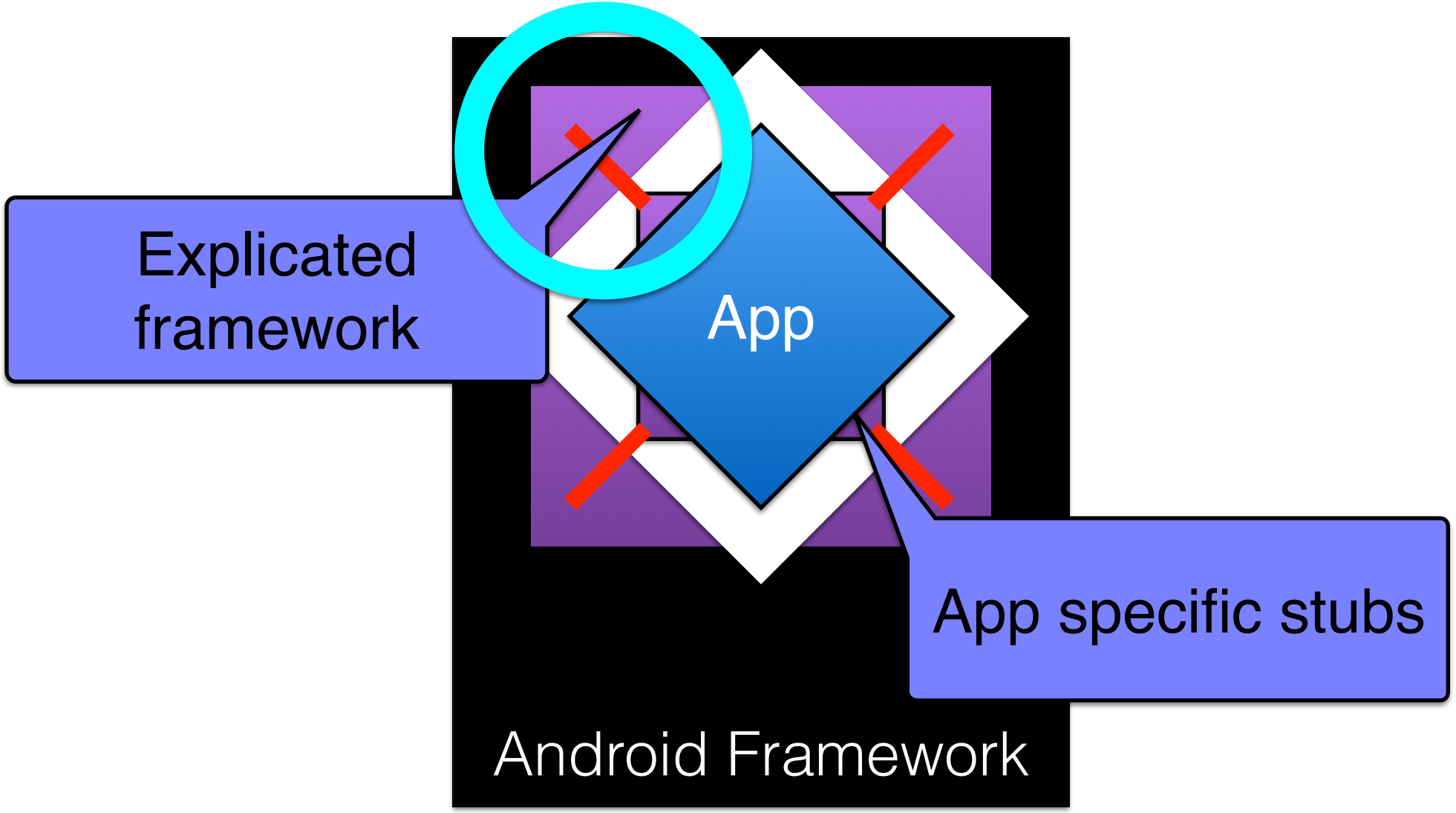
A Different Approach

- Independent of the client analysis
- Avoids modeling the execution context

Model and augment the Android framework



Contribution of Droidel: model and augment



One time manual explication of the Android framework

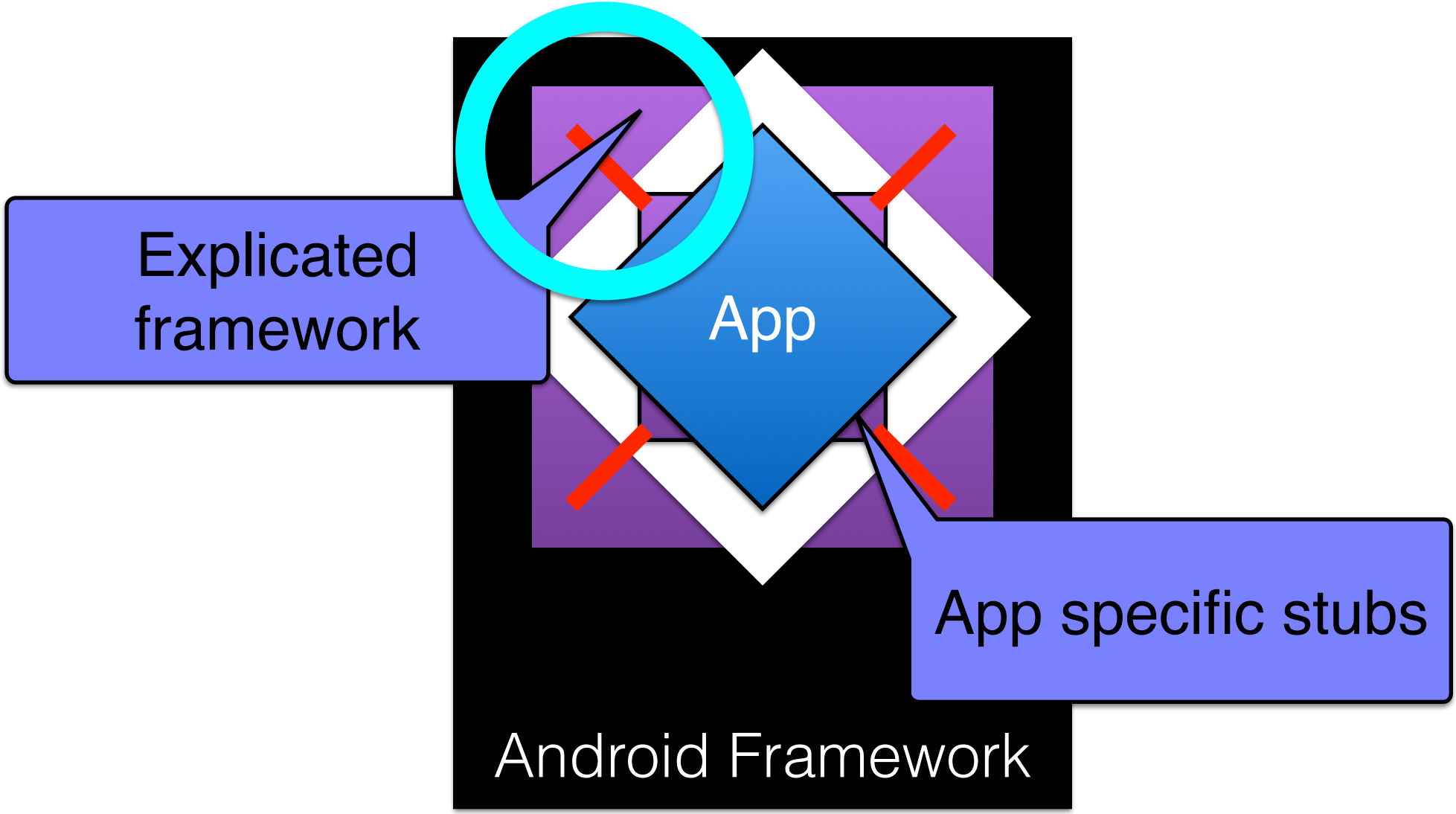
```
public interface DroidelStubs {
    …
     Activity getActivity(String cls);
    …
}
```

Replace with an **explicit** call to `DroidelStubs`

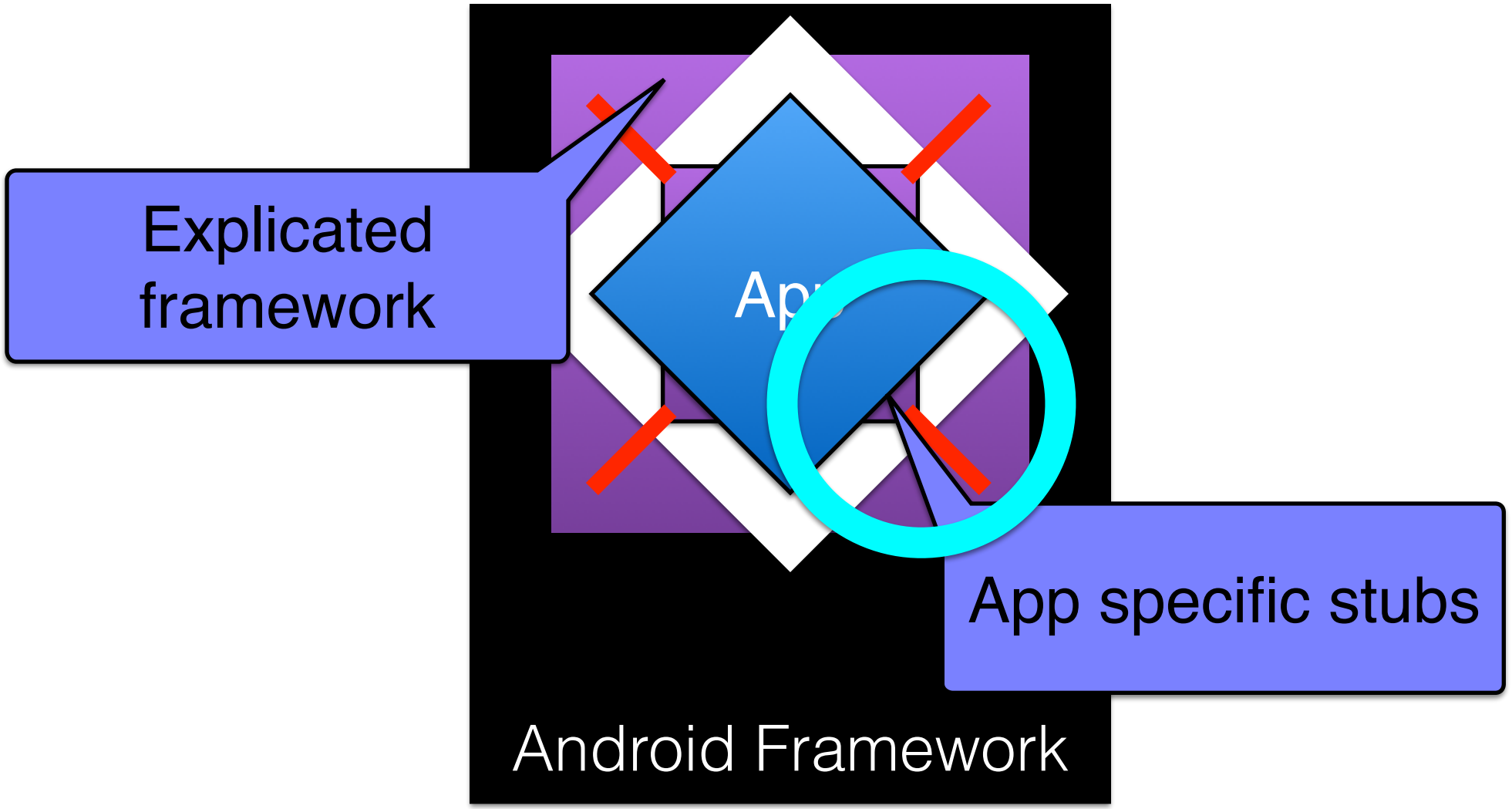
```
Activity a = (Activity) clazz.newInstance();
```

One time manual identification of the uses of reflection in the Android framework and replace those calls with explicit calls to `DroidelStubs`

Contribution of Droidel: model and augment



Contribution of Droidel: model and augment



Automatic app specific stub generation

```
public interface DroidelStubs {
    …
    Activity getActivity(String cls);
    …
}
```

getter method for Activities

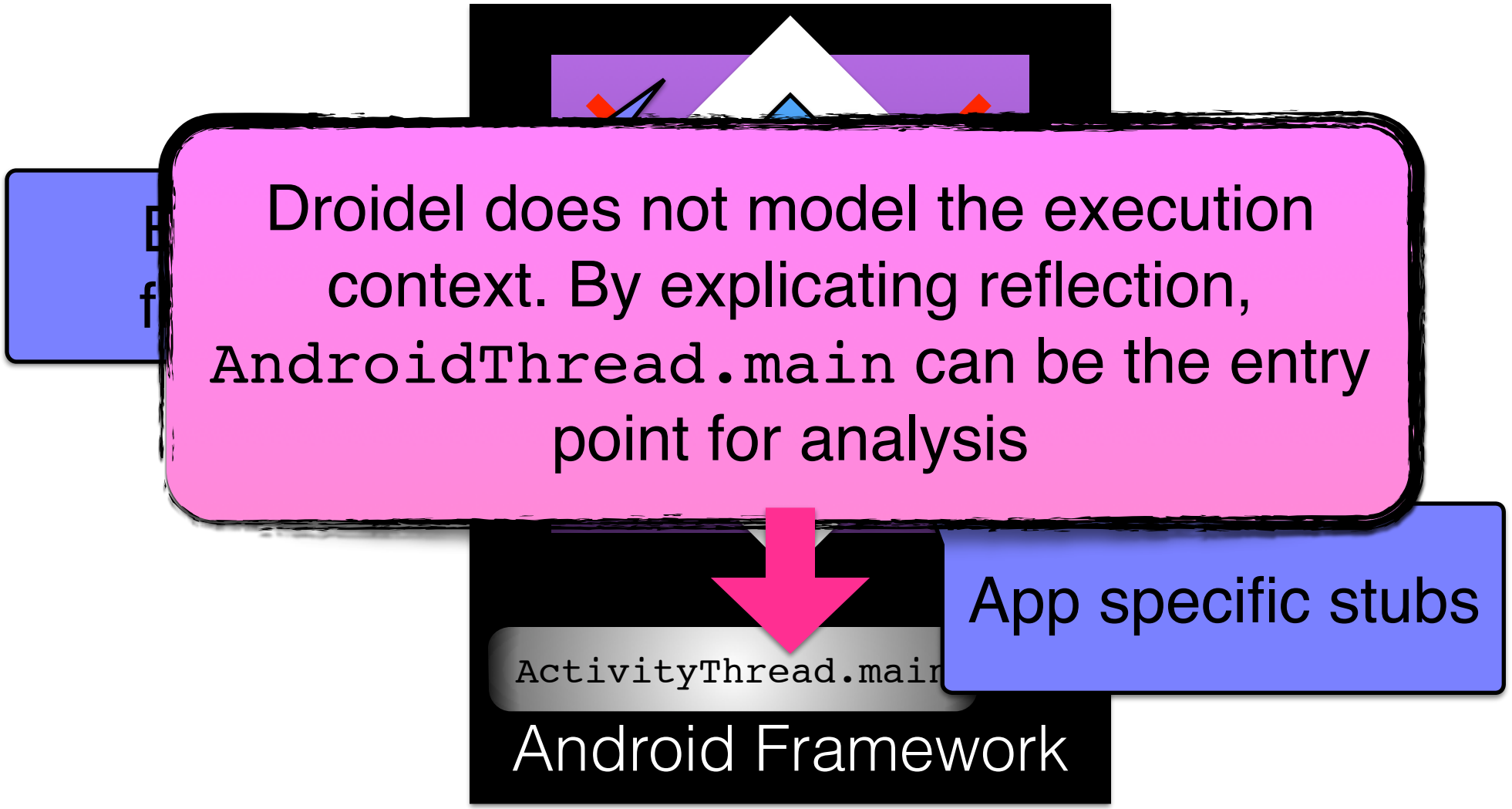
ntation

Droidel generates an implementation of `DroidelStubs` for each app

Dispatche
name and calls the zero
argument constructor
based as per the
instructions in the
documentation for
`newInstance`

```
lStubs {
cls) {
{
        return new ActivityA();
    } else if (cls == "Activity B") {
        return new ActivityB();
    } else { return new Activity(); }
  }
}
```

Contribution of Droidel: model and augment



Empirical Evaluation

Experimental methodology

"The fundamental law of bug finding is No Check = No Bug. If the tool can't check a **method**, then it won't find bugs in it." [1]

Evaluate the percentage of concretely reachable methods in the call graph.

1 Al Bessey , Ken Block , Ben Chelf , Andy Chou , Bryan Fulton , Seth Hallem , Charles Henri-Gros , Asya Kamsky , Scott McPeak , Dawson Engler, A few billion lines of code later: using static analysis to find bugs in the real world, Communications of the ACM, v.53 n.2, February 2010

Experimental setup

1. Manual exploration of a set of 7 android apps

2. Compute the number of concretely reachable methods

3. Compare the number of concretely reachable methods in the call graphs generated using Droidel and FlowDroid (a taint analysis framework model).

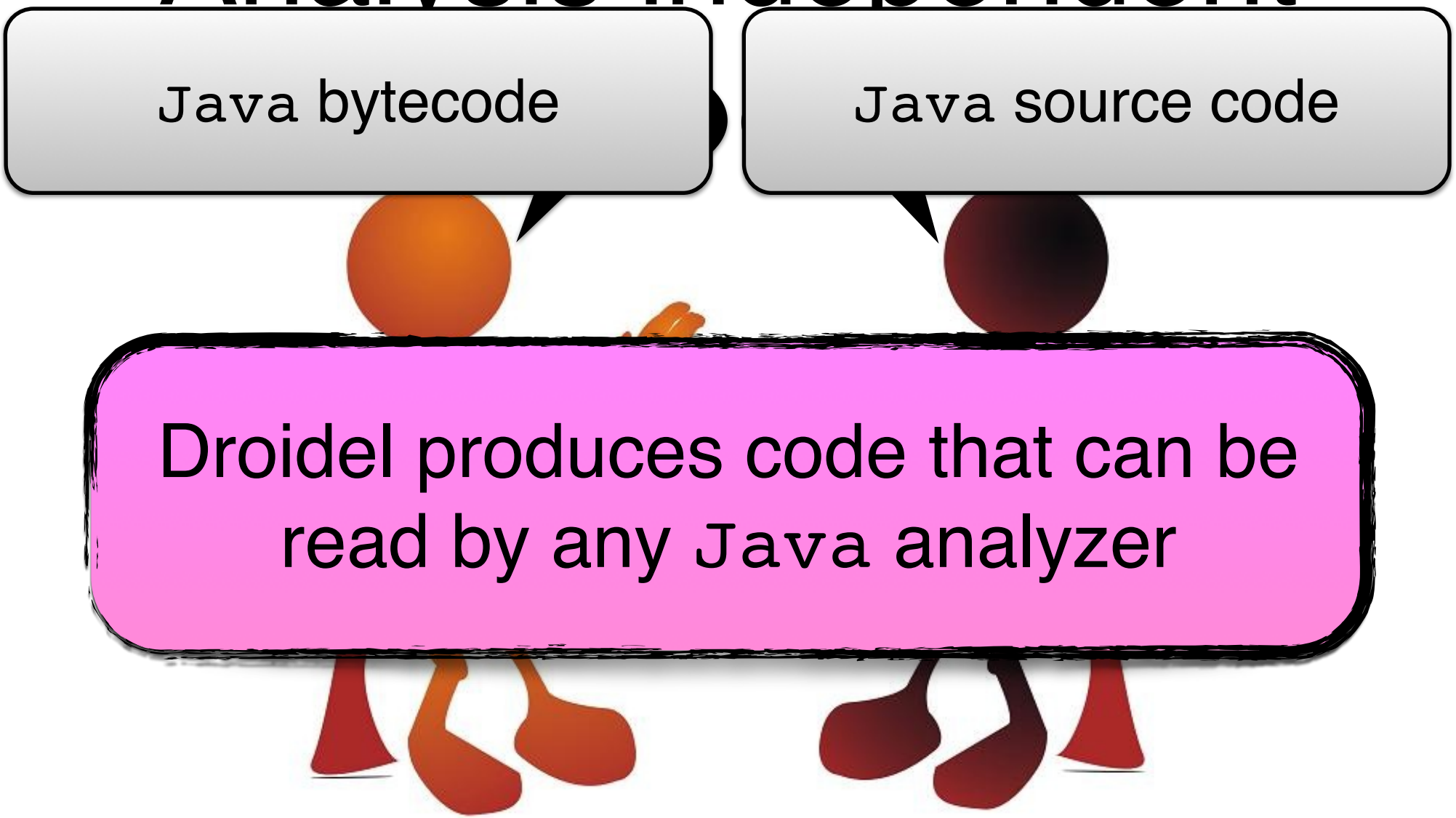
Experimental results

Benchmark	Dynamic Exploration of App Methods			Reachable methods (FlowDroid)		Reachable methods (Droidel)	
	Total	Visited	% Visited	Reachable	% Missed	Reachable	% Missed
drupaleditor	325	90	28	78	13	88	2
spycamera				40	74	151	3
npr				76	21	90	6
duckduckgo				352	32	449	14
textsecure				925	32	1141	16
wordpress	5796	2042	35	1362	33	1961	4
k9	5357	1905	36	1267	33	1773	7
Summary	17467	6173	**38**	4120	**30**	5653	**6**

FlowDroid: 30%
vs.
Droidel: 6%

FlowDroid **misses more** concretely reachable methods than DROIDEL

Analysis independent



How can you help us?

Remember 6%?

Please use your Android expertise to contribute to Droidel

(https://github.com/cuplv/droidel)

EXTRA SLIDES

Current Limitations of DROIDEL

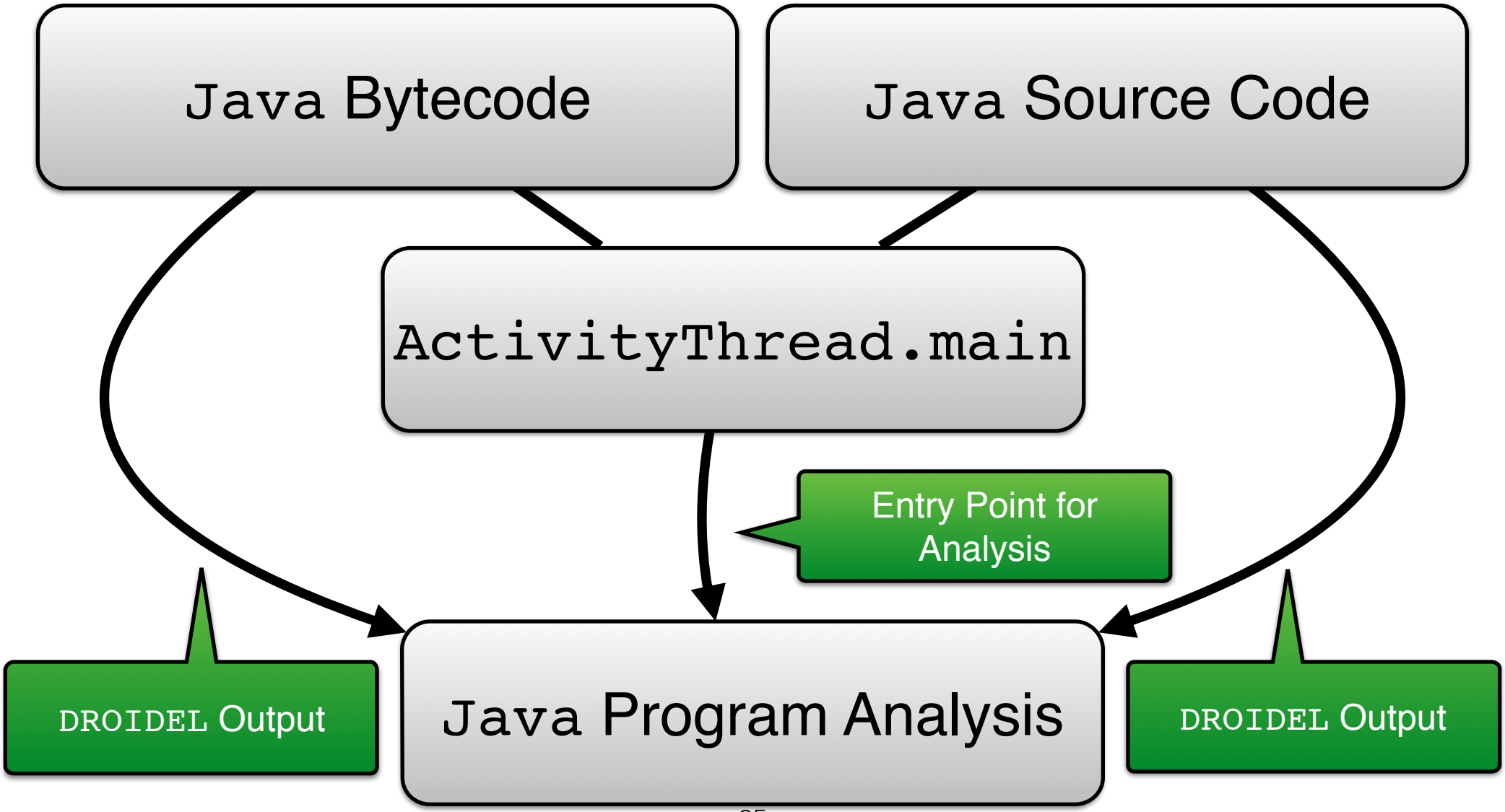
- Not all uses of reflection have been explicated yet (i.e. Reflective allocation of `Preferences` objects)
- No generated stubs for summarizing native methods in Android

Not a problem with our approach but a limitation of the current implementation

Issues with this approach

- Client analysis specific
- Targeting another client analysis causes soundness issues
- Extensive manual effort

DROIDEL Outputs



- Manually explicate each version of the Android Framework

The current model and replace approach suffers this problem as well