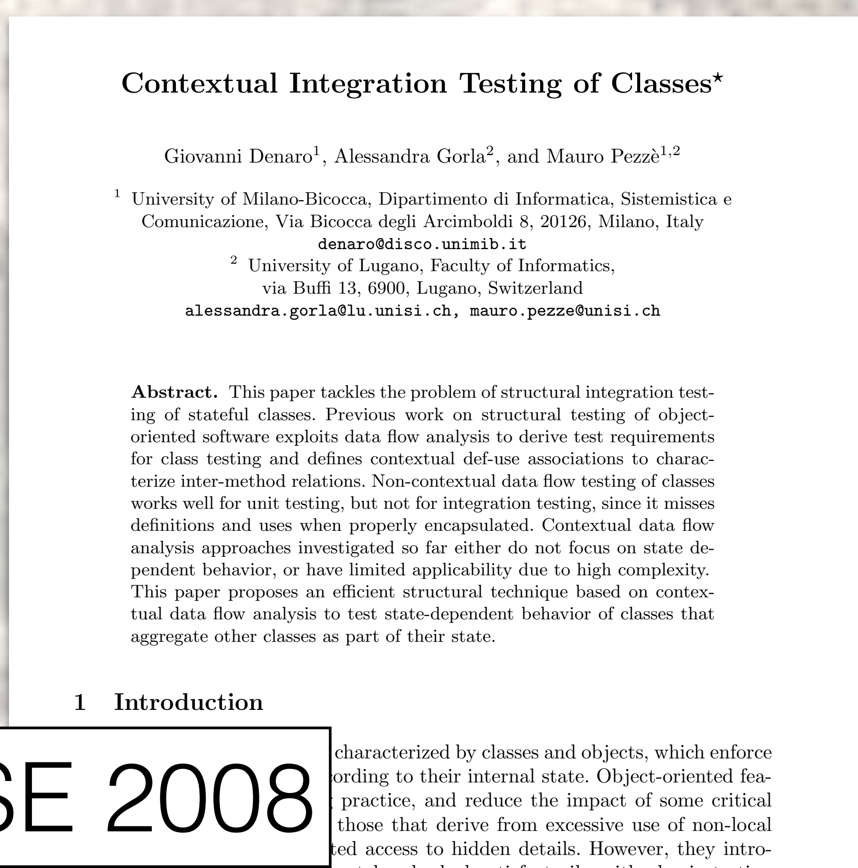
Challenges and Opportunities in Mobile Testing

Alessandra Gorla
IMDEA Software Institute, Madrid, Spain

Intro

B.Sc. and M.Sc. in Milano-Bicocca, Italy

Data-flow testing of Java Applications



Contextual Integration Testing of Classes*

Giovanni Denaro[1], Alessandra Gorla[2], and Mauro Pezzè[1,2]

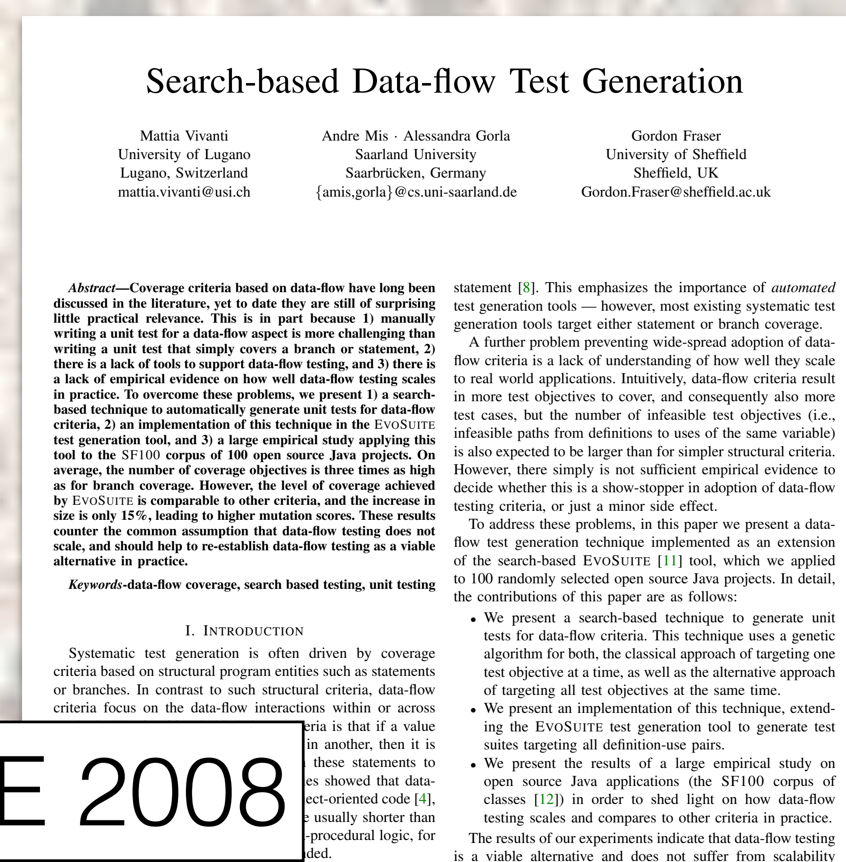
[1] University of Milano-Bicocca, Dipartimento di Informatica, Sistemistica e Comunicazione, Via Bicocca degli Arcimboldi 8, 20126, Milano, Italy
denaro@disco.unimib.it
[2] University of Lugano, Faculty of Informatics, via Buffi 13, 6900, Lugano, Switzerland
alessandra.gorla@lu.unisi.ch, mauro.pezze@unisi.ch

Abstract. This paper tackles the problem of structural integration testing of stateful classes. Previous work on structural testing of object-oriented software exploits data flow analysis to derive test requirements for class testing and defines contextual def-use associations to characterize inter-method relations. Non-contextual data flow testing of classes works well for unit testing, but not for integration testing, since it misses definitions and uses when properly encapsulated. Contextual data flow analysis approaches investigated so far either do not focus on state dependent behavior, or have limited applicability due to high complexity. This paper proposes an efficient structural technique based on contextual data flow analysis to test state-dependent behavior of classes that aggregate other classes as part of their state.

1 Introduction

characterized by classes and objects, which enforce
ording to their internal state. Object-oriented fea-
practice, and reduce the impact of some critical
those that derive from excessive use of non-local
ted access to hidden details. However, they intro-

FASE 2008



Search-based Data-flow Test Generation

Mattia Vivanti
University of Lugano
Lugano, Switzerland
mattia.vivanti@usi.ch

Andre Mis · Alessandra Gorla
Saarland University
Saarbrücken, Germany
{amis,gorla}@cs.uni-saarland.de

Gordon Fraser
University of Sheffield
Sheffield, UK
Gordon.Fraser@sheffield.ac.uk

***Abstract*—Coverage criteria based on data-flow have long been discussed in the literature, yet to date they are still of surprising little practical relevance. This is in part because 1) manually writing a unit test for a data-flow aspect is more challenging than writing a unit test that simply covers a branch or statement, 2) there is a lack of tools to support data-flow testing, and 3) there is a lack of empirical evidence on how well data-flow testing scales in practice. To overcome these problems, we present 1) a search-based technique to automatically generate unit tests for data-flow criteria, 2) an implementation of this technique in the EvoSuite test generation tool, and 3) a large empirical study applying this tool to the SF100 corpus of 100 open source Java projects. On average, the number of coverage objectives is three times as high as for branch coverage. However, the level of coverage achieved by EvoSuite is comparable to other criteria, and the increase in size is only 15%, leading to higher mutation scores. These results counter the common assumption that data-flow testing does not scale, and should help to re-establish data-flow testing as a viable alternative in practice.**

***Keywords*-data-flow coverage, search based testing, unit testing**

I. Introduction

Systematic test generation is often driven by coverage criteria based on structural program entities such as statements or branches. In contrast to such structural criteria, data-flow criteria focus on the data-flow interactions within or across
eria is that if a value
in another, then it is
these statements to
es showed that data-
ect-oriented code [4],
usually shorter than
-procedural logic, for
ded.

statement [8]. This emphasizes the importance of *automated* test generation tools — however, most existing systematic test generation tools target either statement or branch coverage.

A further problem preventing wide-spread adoption of data-flow criteria is a lack of understanding of how well they scale to real world applications. Intuitively, data-flow criteria result in more test objectives to cover, and consequently also more test cases, but the number of infeasible test objectives (i.e., infeasible paths from definitions to uses of the same variable) is also expected to be larger than for simpler structural criteria. However, there simply is not sufficient empirical evidence to decide whether this is a show-stopper in adoption of data-flow testing criteria, or just a minor side effect.

To address these problems, in this paper we present a data-flow test generation technique implemented as an extension of the search-based EvoSuite [11] tool, which we applied to 100 randomly selected open source Java projects. In detail, the contributions of this paper are as follows:

- We present a search-based technique to generate unit tests for data-flow criteria. This technique uses a genetic algorithm for both, the classical approach of targeting one test objective at a time, as well as the alternative approach of targeting all test objectives at the same time.
- We present an implementation of this technique, extending the EvoSuite test generation tool to generate test suites targeting all definition-use pairs.
- We present the results of a large empirical study on open source Java applications (the SF100 corpus of classes [12]) in order to shed light on how data-flow testing scales and compares to other criteria in practice.

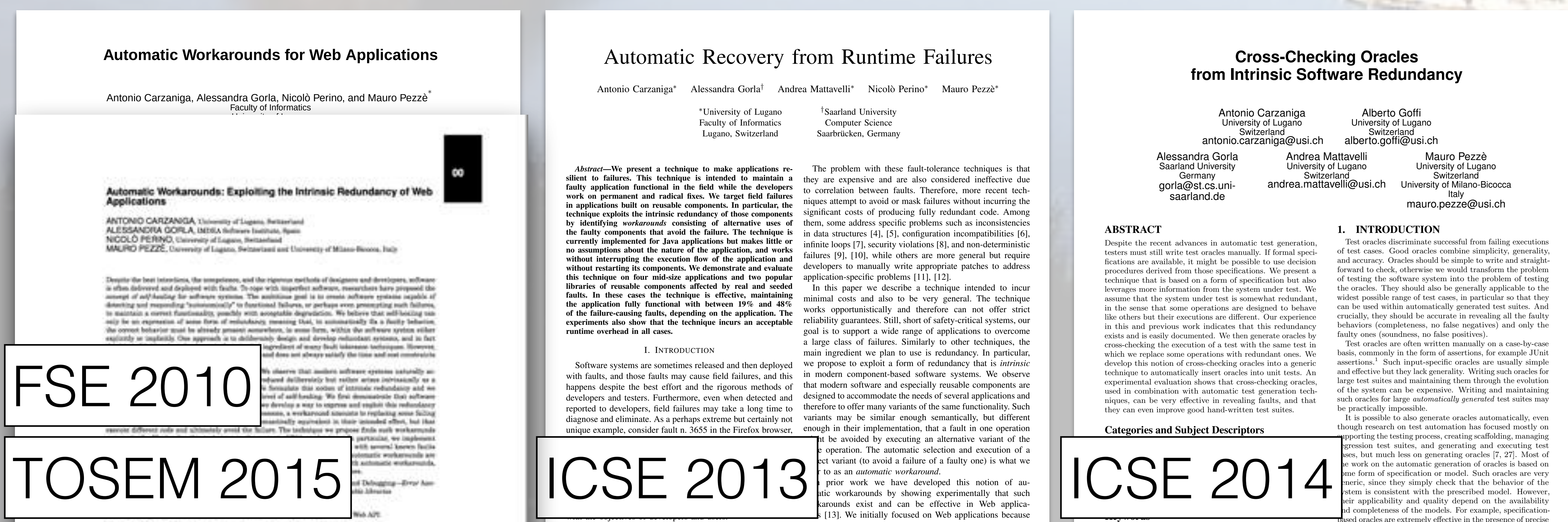
The results of our experiments indicate that data-flow testing is a viable alternative and does not suffer from scalability

ISSRE 2008

Intro

PhD in Informatics in Lugano, Switzerland

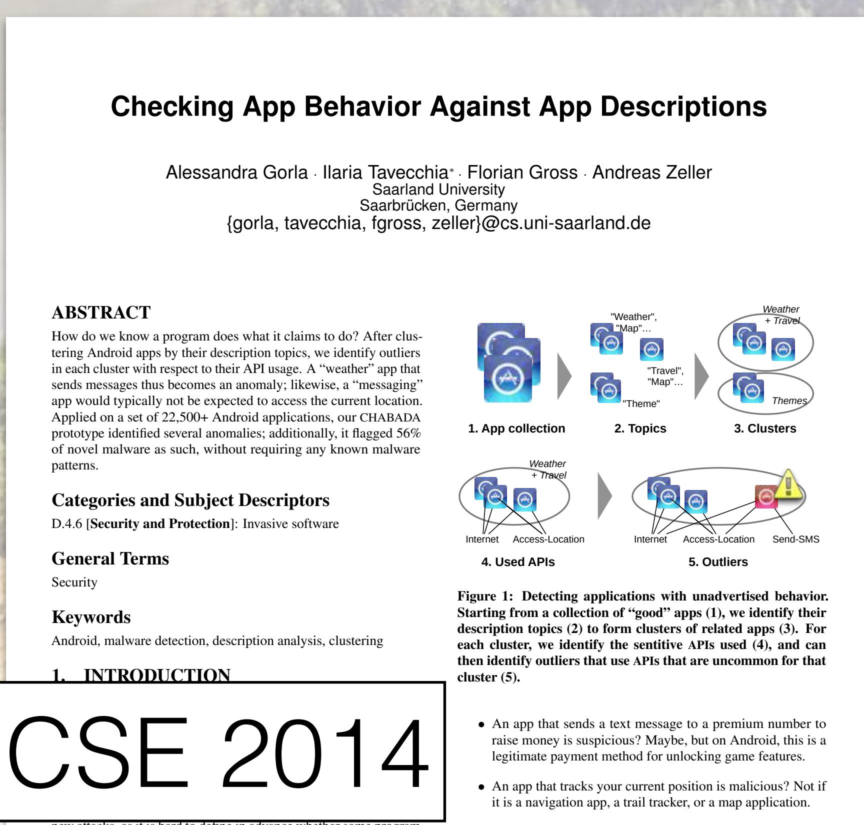
Automatic Workarounds for Web Applications



Intro

Postdoc Saarland University, Germany

Malware detection in Android applications


Checking App Behavior Against App Descriptions

Alessandra Gorla · Ilaria Tavecchia · Florian Gross · Andreas Zeller
Saarland University
Saarbrücken, Germany
{gorla, tavecchia, fgross, zeller}@cs.uni-saarland.de

ABSTRACT
How do we know a program does what it claims to do? After clustering Android apps by their description topics, we identify outliers in each cluster with respect to their API usage. A “weather” app that sends messages thus becomes an anomaly; likewise, a “messaging” app would typically not be expected to access the current location. Applied on a set of 22,500+ Android applications, our CHABADA prototype identified several anomalies; additionally, it flagged 56% of novel malware as such, without requiring any known malware patterns.

Categories and Subject Descriptors
D.4.6 [Security and Protection]: Invasive software

General Terms
Security

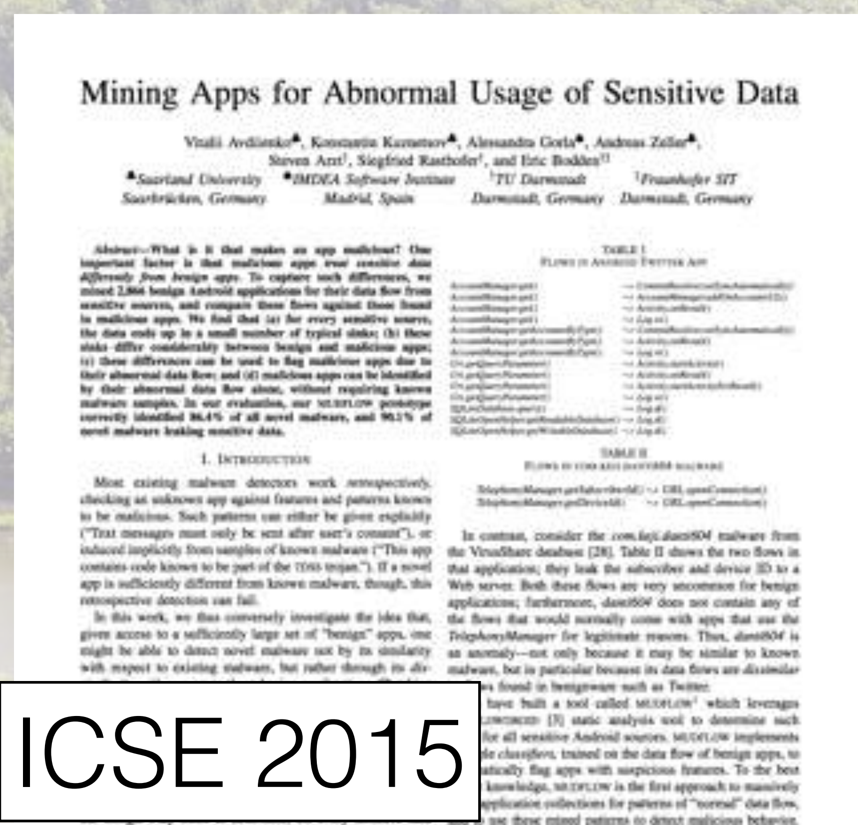
Keywords
Android, malware detection, description analysis, clustering

1. INTRODUCTION

Figure 1: Detecting applications with unadvertised behavior. Starting from a collection of “good” apps (1), we identify their description topics (2) to form clusters of related apps (3). For each cluster, we identify the sentitive APIs used (4), and can then identify outliers that use APIs that are uncommon for that cluster (5).

- An app that sends a text message to a premium number to raise money is suspicious? Maybe, but on Android, this is a legitimate payment method for unlocking game features.
- An app that tracks your current position is malicious? Not if it is a navigation app, a trail tracker, or a map application.

ICSE 2014


Mining Apps for Abnormal Usage of Sensitive Data

ICSE 2015


Search-Based Security Testing of Web Applications

Julian Thomé, Alessandra Gorla, Andreas Zeller
Saarland University, Saarbrücken, Germany

SBST 2014

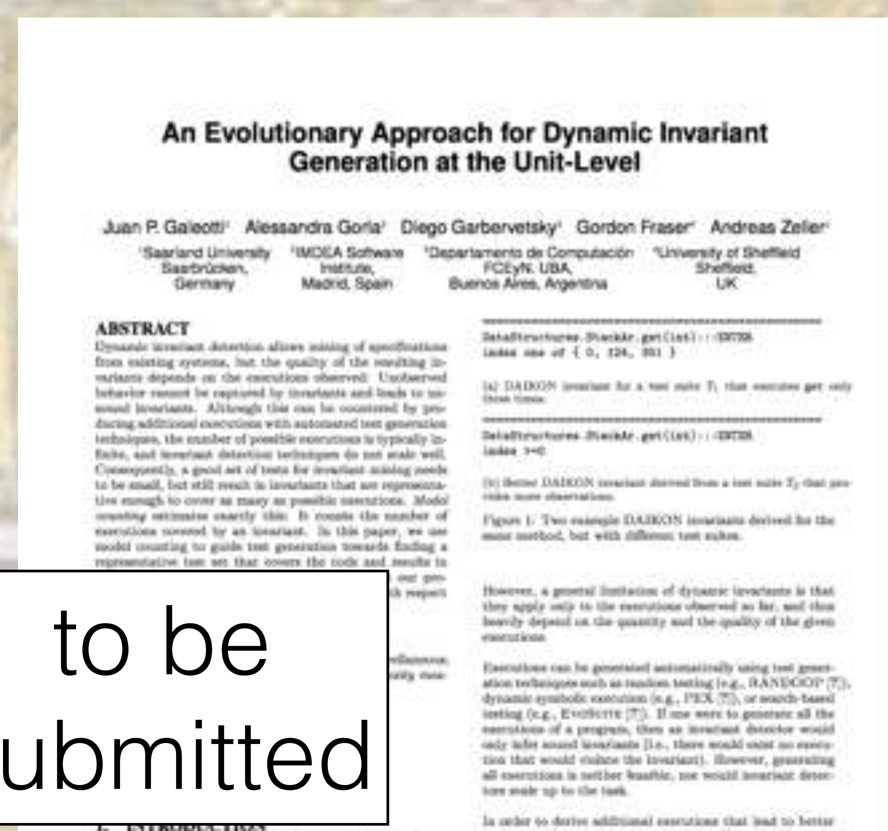
Intro

Assistant professor @ IMDEA software
Madrid, Spain
since January 2015


Automated Test Input Generation for Android: Are We There Yet?

Shauvik Roy Choudhary, Georgia Institute of Technology, USA
Alessandra Gorla, IMDEA Software Institute, Spain
Alessandro Orso, Georgia Institute of Technology, USA

under submission


An Evolutionary Approach for Dynamic Invariant Generation at the Unit-Level

Juan P. Galeotti, Alessandra Gorla, Diego Garbervetsky, Gordon Fraser, Andreas Zeller

ABSTRACT

to be submitted

Interested in internships
short visits
giving talks??

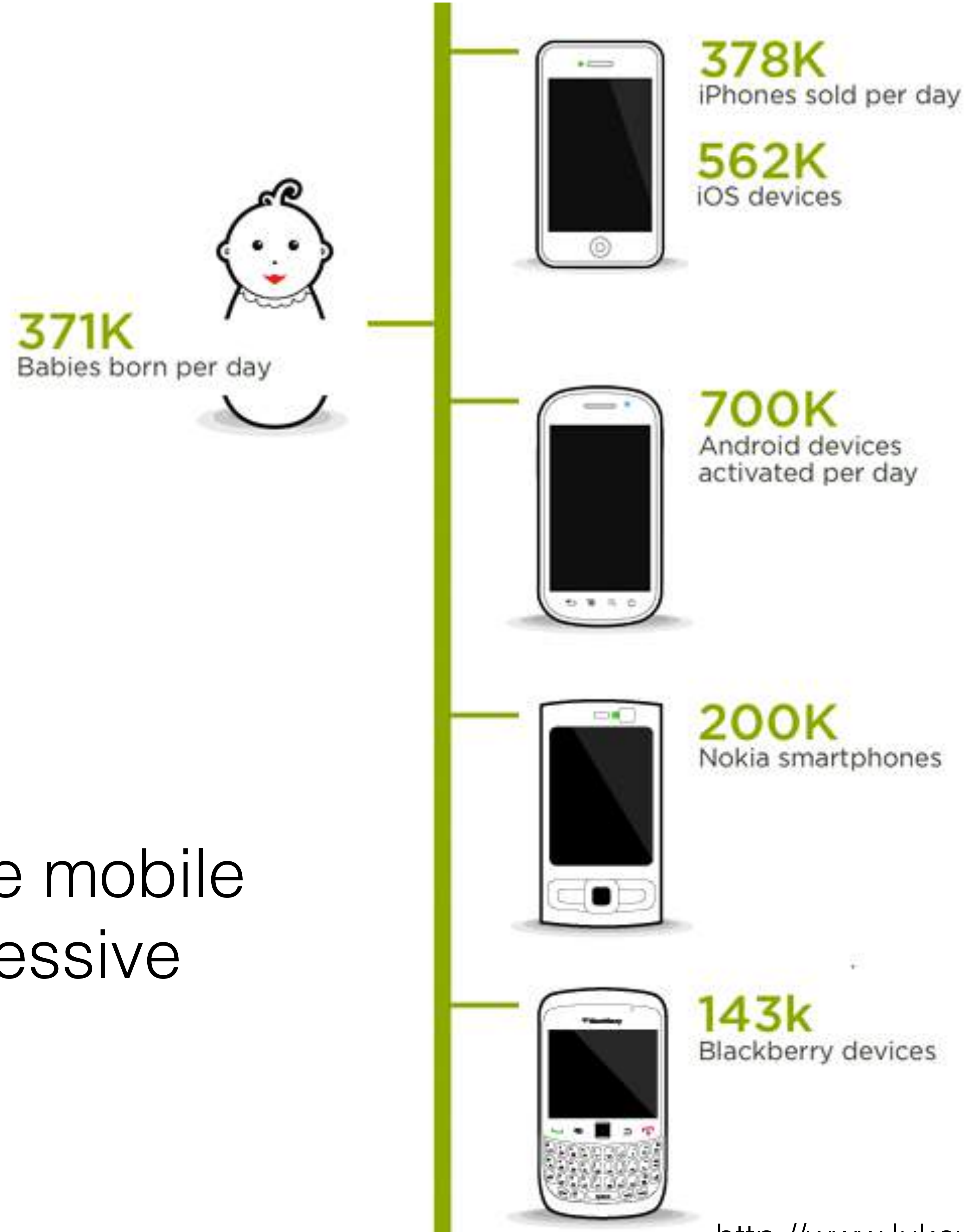
https://www.software.imdea.org/~alessandra.gorla

alessandra.gorla@imdea.org

About this talk

- *intro to Android*
- *state of the art in Android testing*
- *open challenges and opportunities ahead*

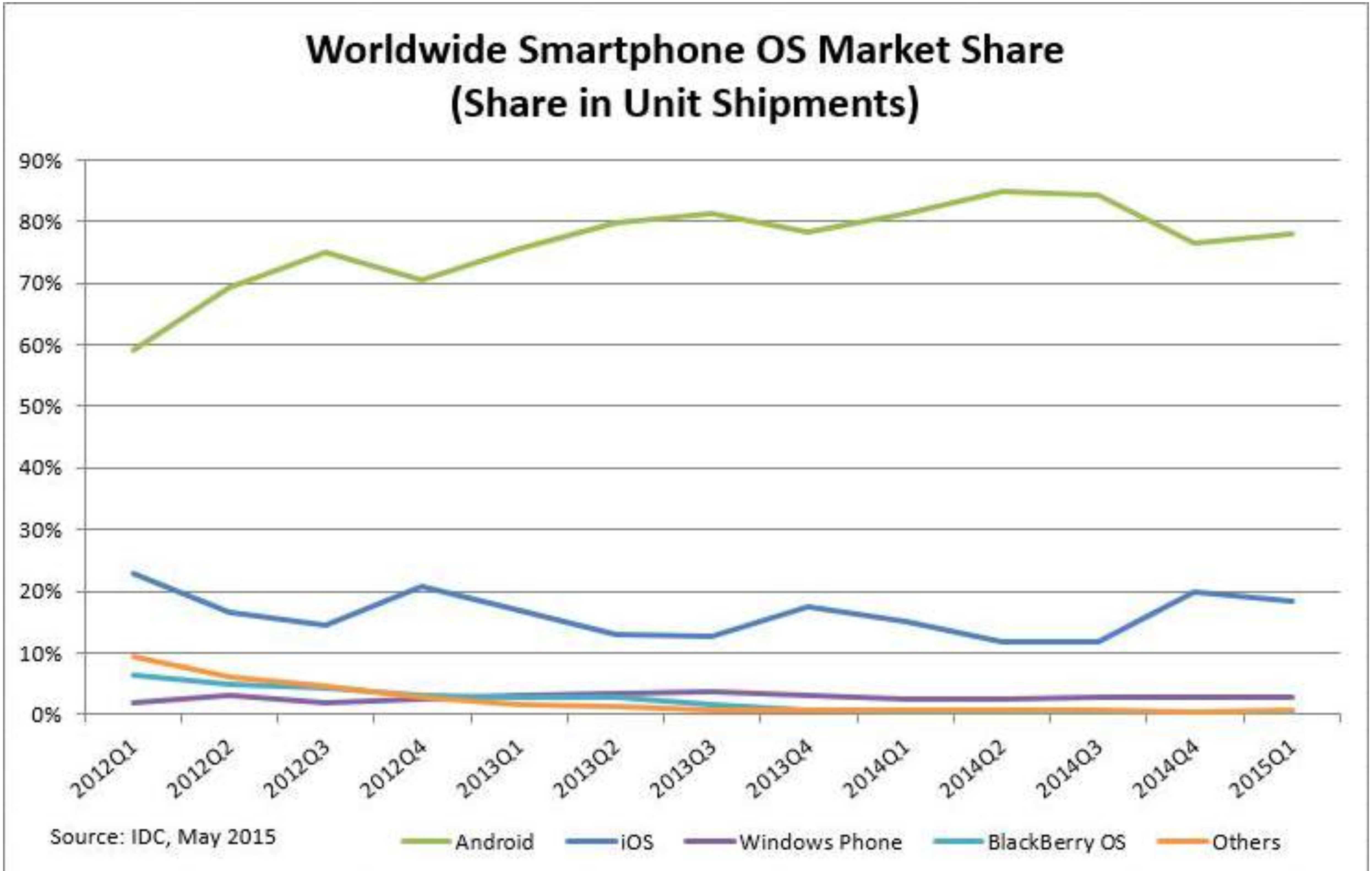
The mobile market and the Android ecosystem



The growth of the mobile market is impressive

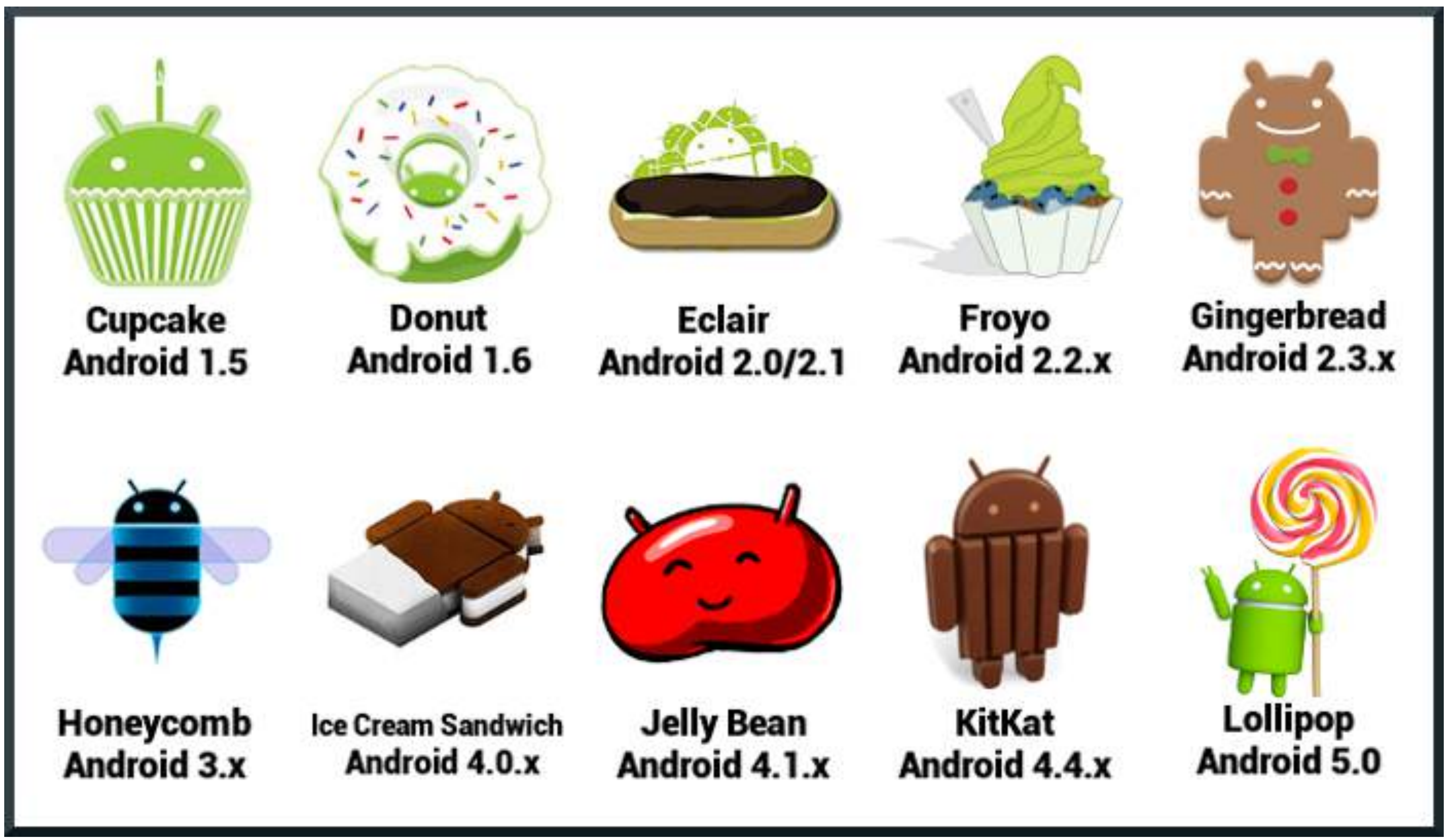
http://www.lukew.com/

Mobile market



Android history

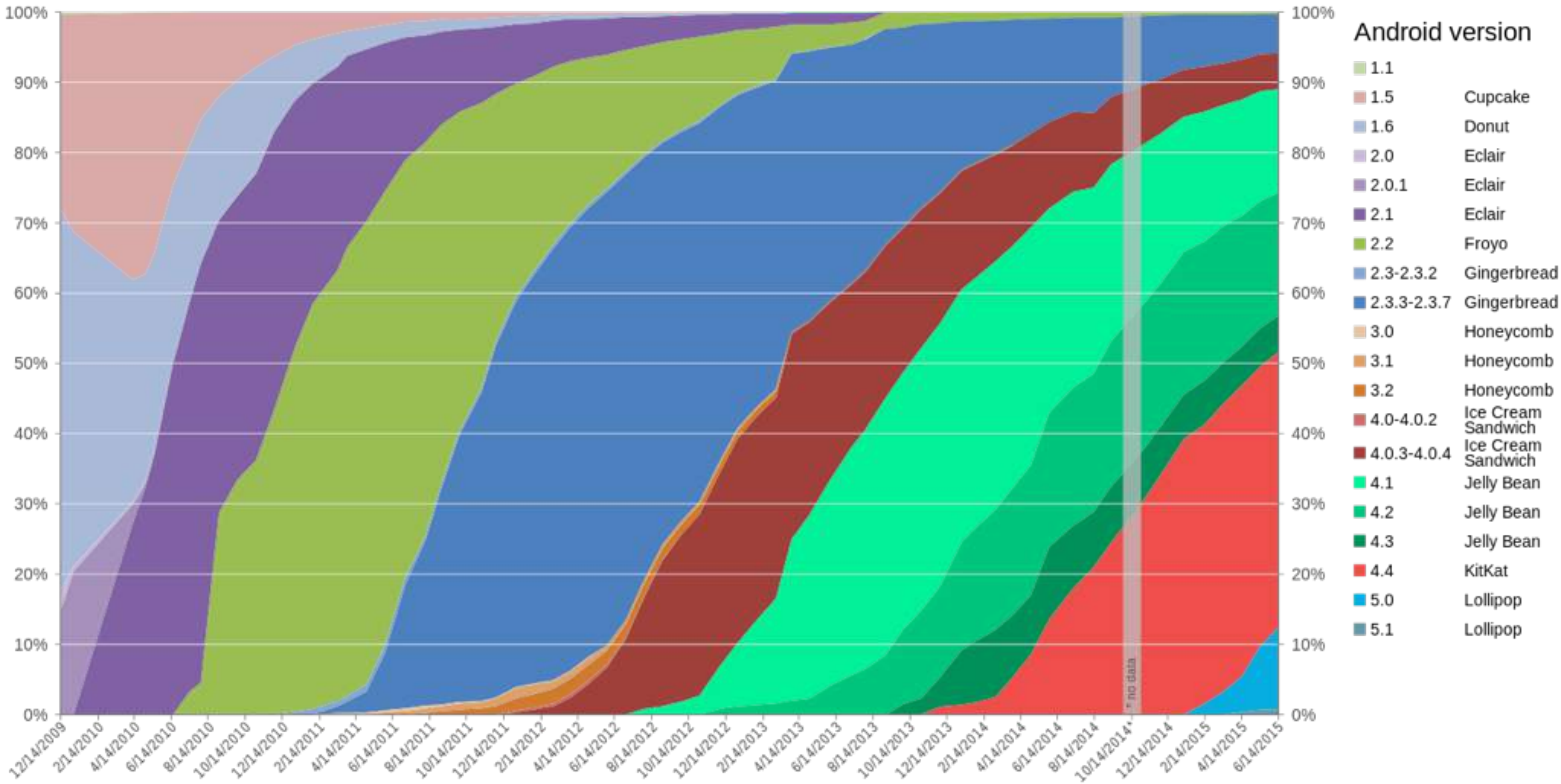
2009



M

Android devices

Release adoption



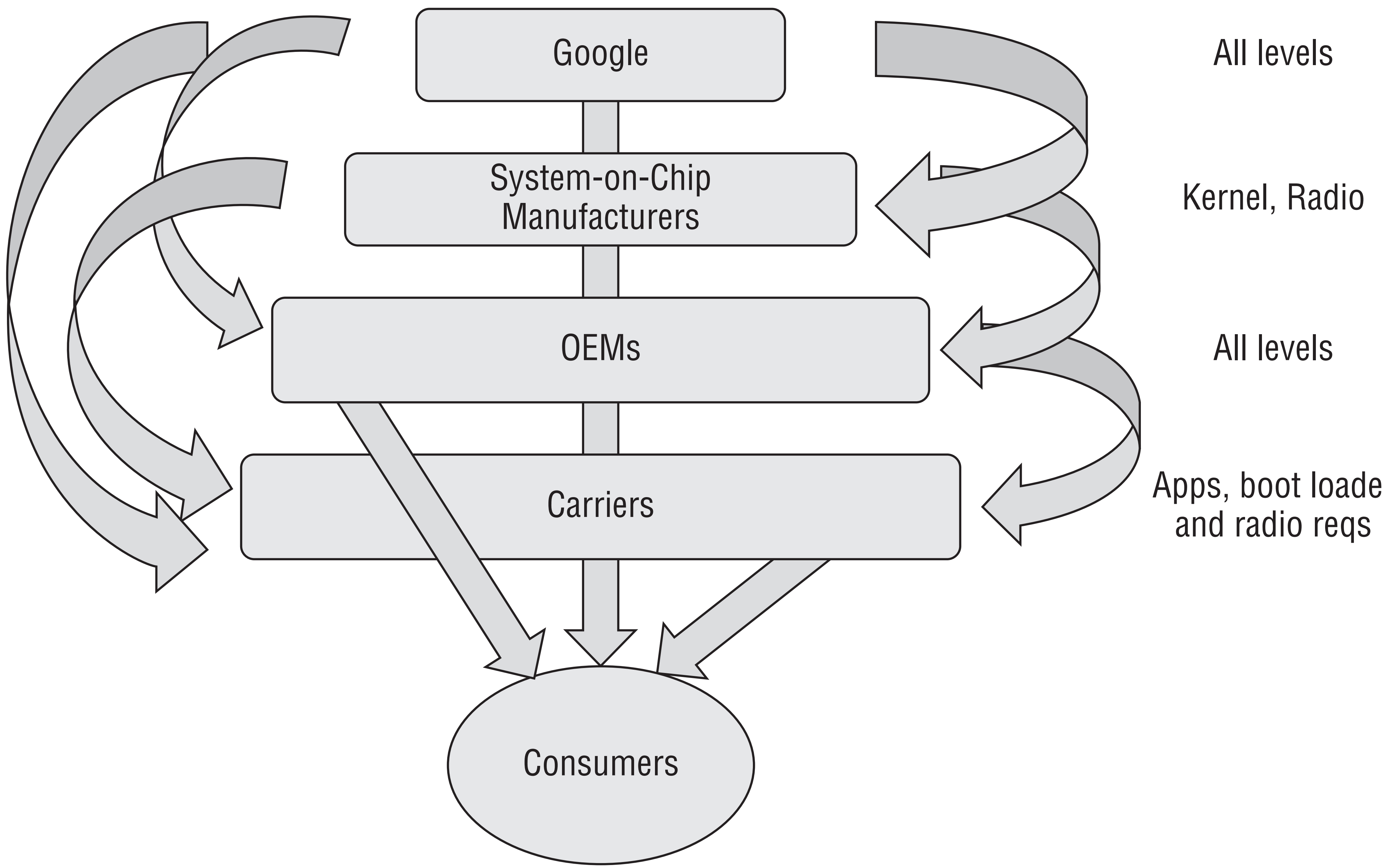
Open source culture

- Android operating system is build upon many different open source components.
    - libraries
    - Linux kernel
    - user interface
    - applications

… but

- There are also closed source components
  - boot loaders
  - peripheral firmware
  - radio components
  - Applications
- And changes in Android are not made available to the public immediately

Android stakeholders



Developers

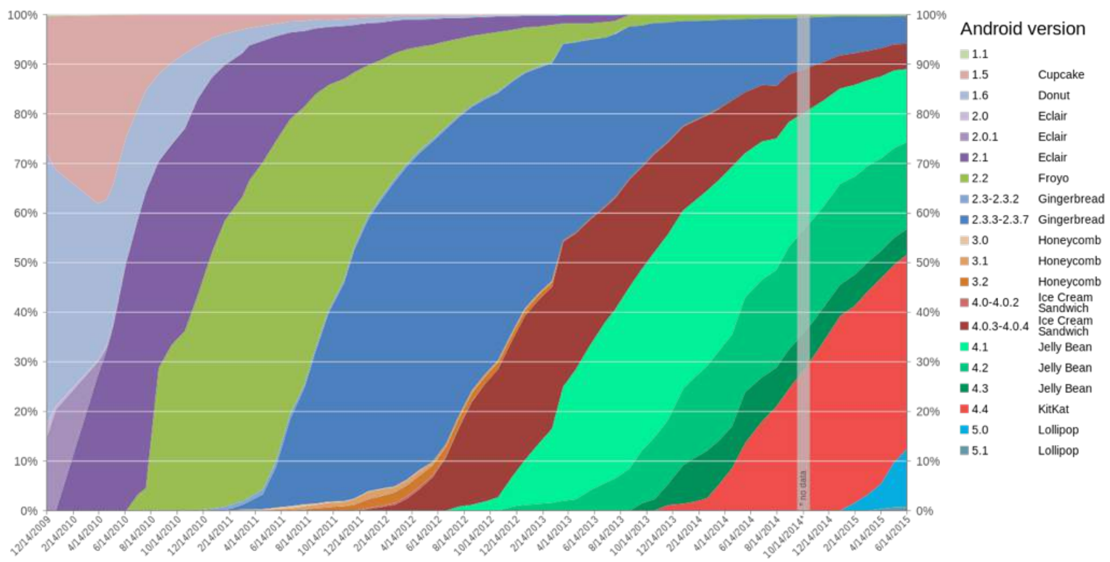
- Developers may contribute to the Android platform.
  - Code review process by Google before including external code.
- Most of external developers contribute by writing apps (through SDK and APIs)
  - Automated analysis before publishing an app in the store.
  - Ranking and report system for further quality

Ecosystem complexity

- fragmentation in hardware
- fragmentation in software
- customization

Issues for quality assurance?

QA issues

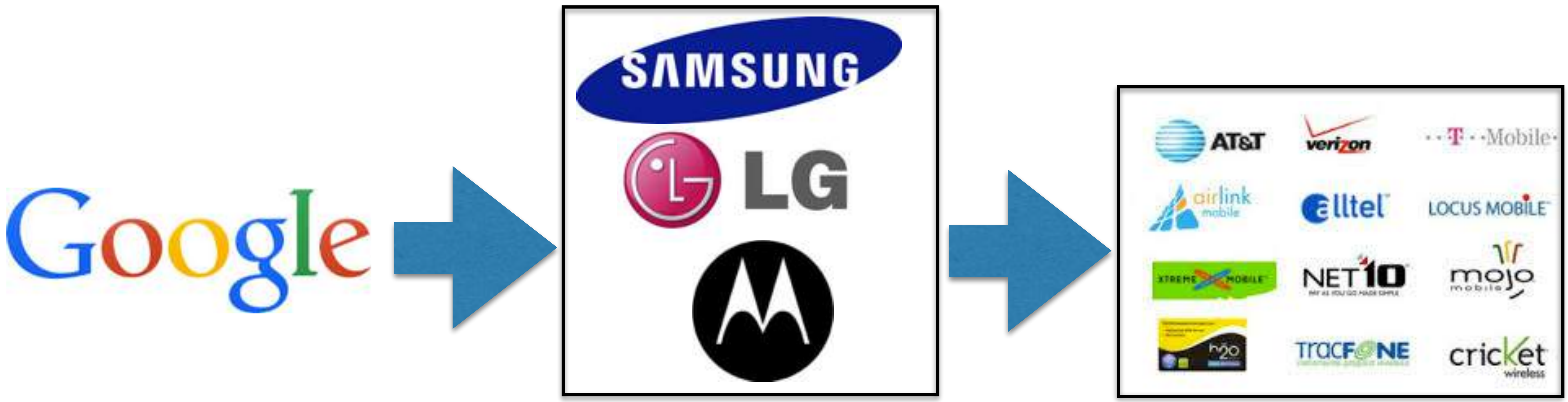


X

~**4** OS releases

+**1000** devices

Security issues



Device manufacturers

Carriers

- Updates might take a long time before being propagated to carrier specific devices.
- Security issues may be fixed after a long time (or even never).

Pros

Security issues are often specific to hw and sw configurations.

Fragmentation makes it hard to develop security attacks that are valid for most devices.

Cons

Security issues detected in the main Android components might take a long time before they are fixed on all devices

Update mechanisms

- Updates to Android are pushed to Nexus phones directly by Google. Days-weeks between security issue report and pushing a fix.

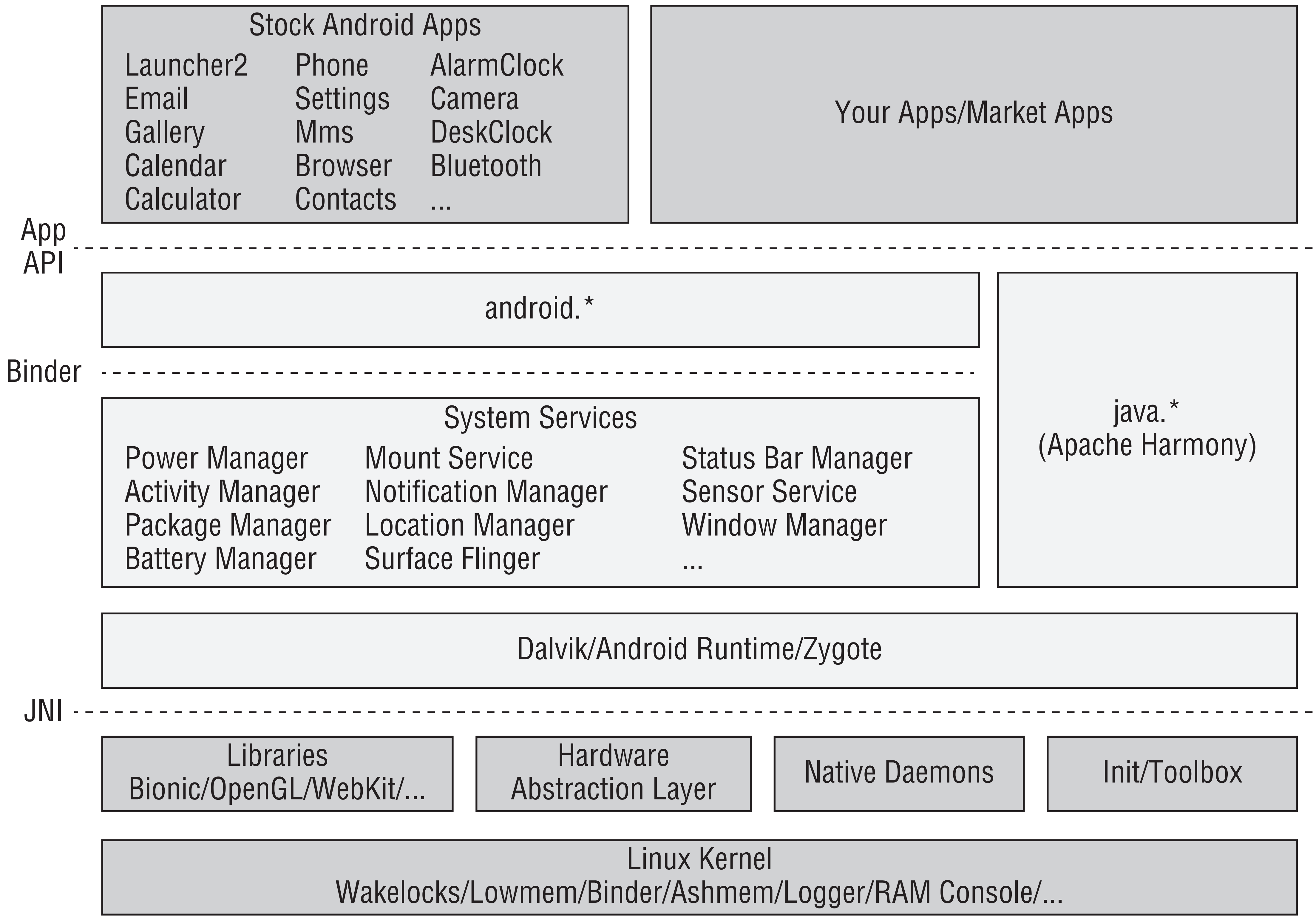
- For other devices it takes longer. Months-years or even never.

- Almost no back-porting (i.e. applying a fix to older versions of the system).

- Updates to apps are easier. Done directly by app developers through the Google store.

Android architecture

Android components



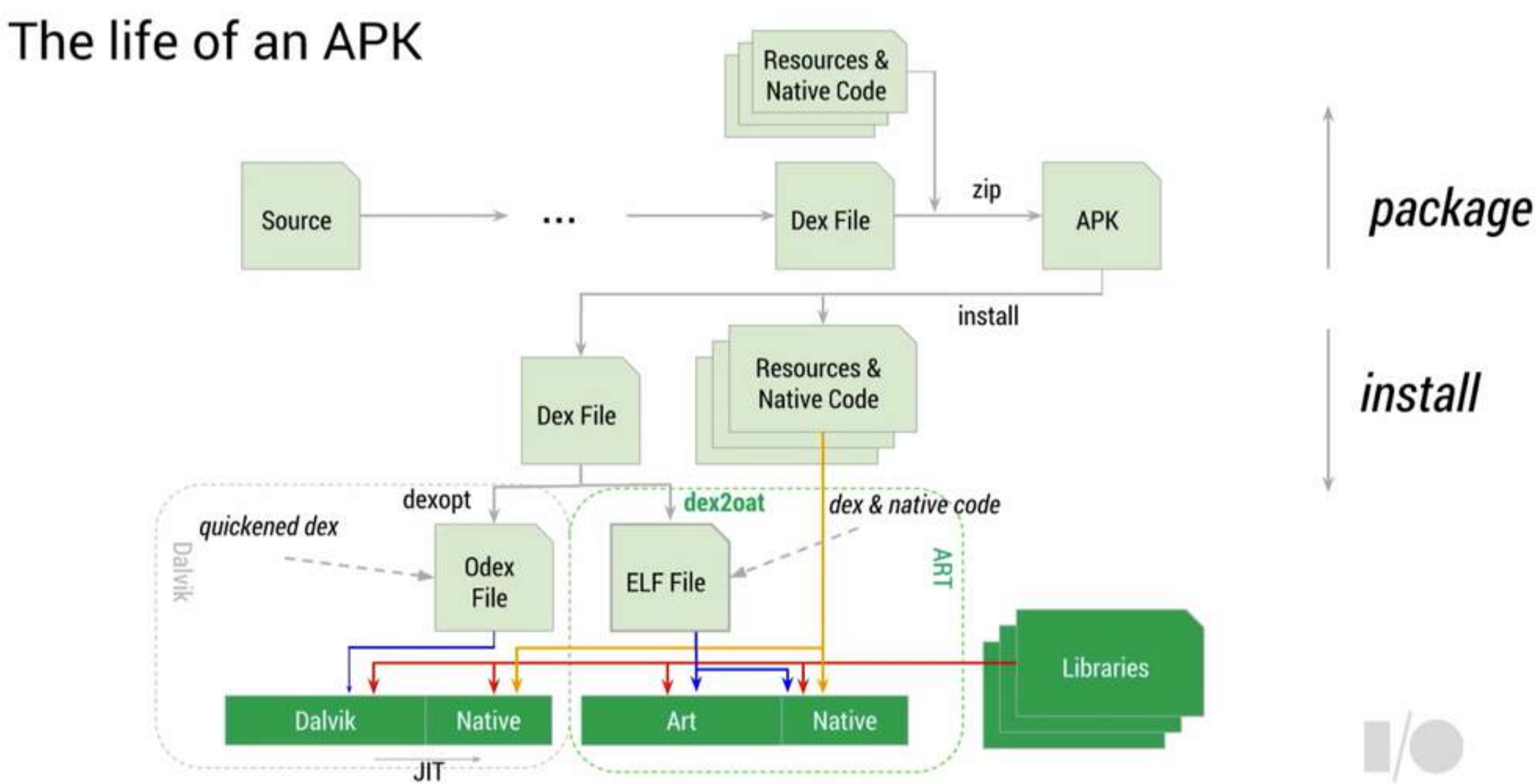
Dalvik VM

- Specifically designed to provide an efficient abstraction layer to the underlying OS
  - register-based VM
  - interprets Dalvik Executable (DEX bytecode format)
  - relies on functionalities provided by a number of supporting native code libraries

Android RunTime

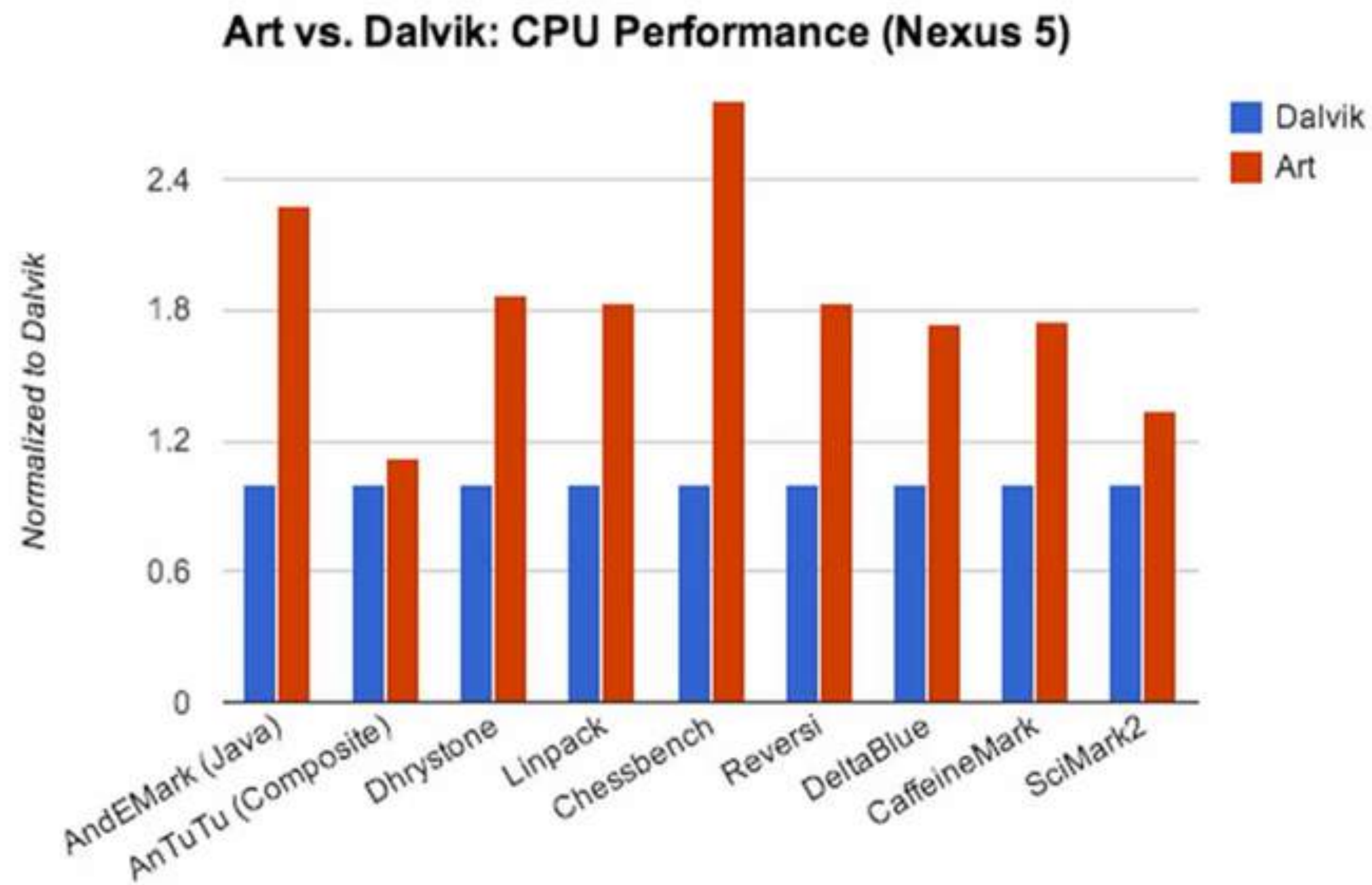
- although..
- Google recently introduce a new runtime environment: ART (Android RunTime)
    - experimental in Android 4.4 (KitKat)
    - default in Android Lollipop
    - main advantage: performance. Instead of Just In Time compiler, it now compiles Ahead Of Time

Android Runtime



Android RunTime

Performance Boosting Thing, realized



Zygote

- Daemon responsible of launching apps.
- Forks a new process for each app.

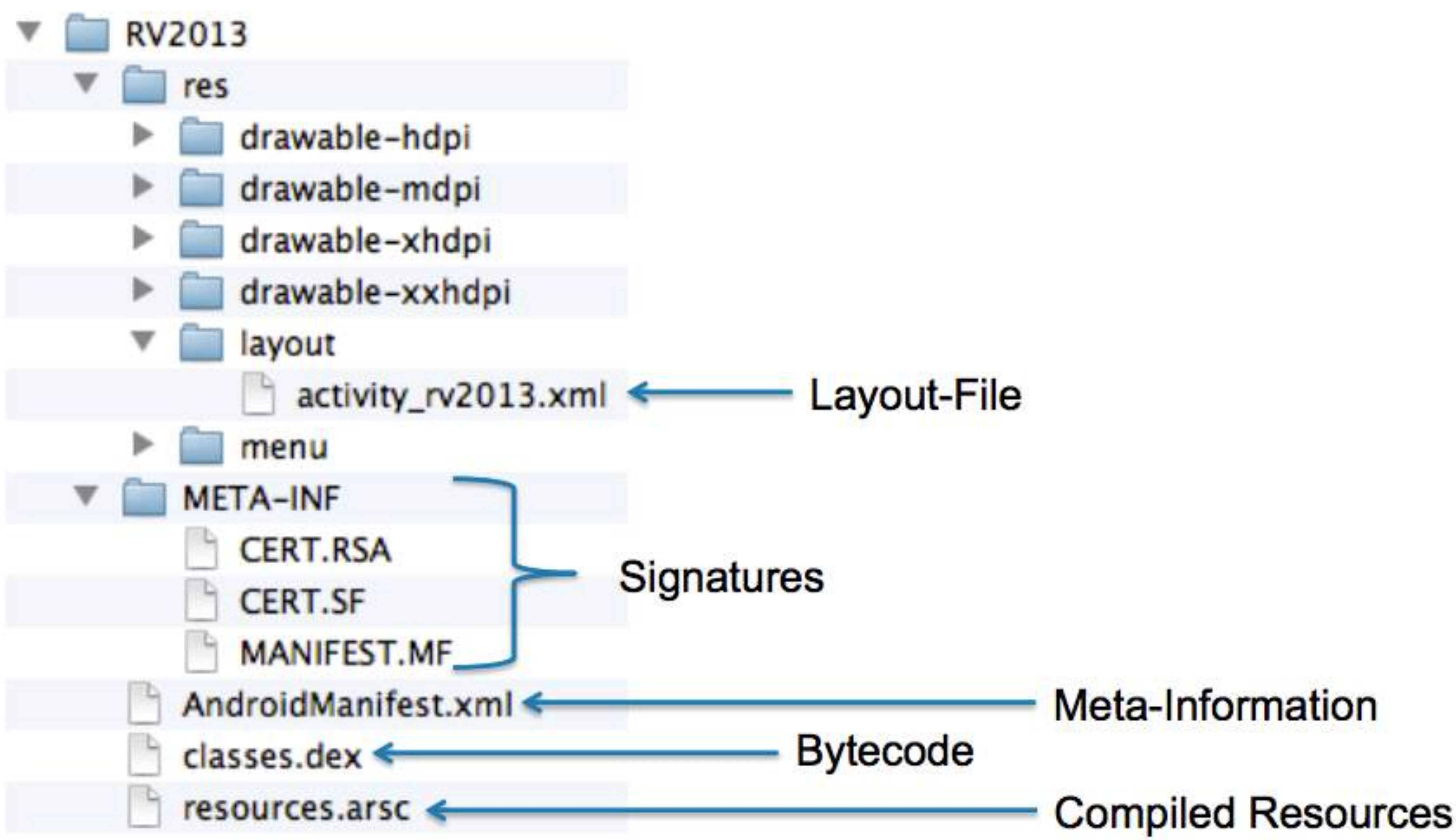
User-space native code components

- Include system services and libraries
  - they communicate with the kernel-level services and drivers.
  - facilitate the low-level operations

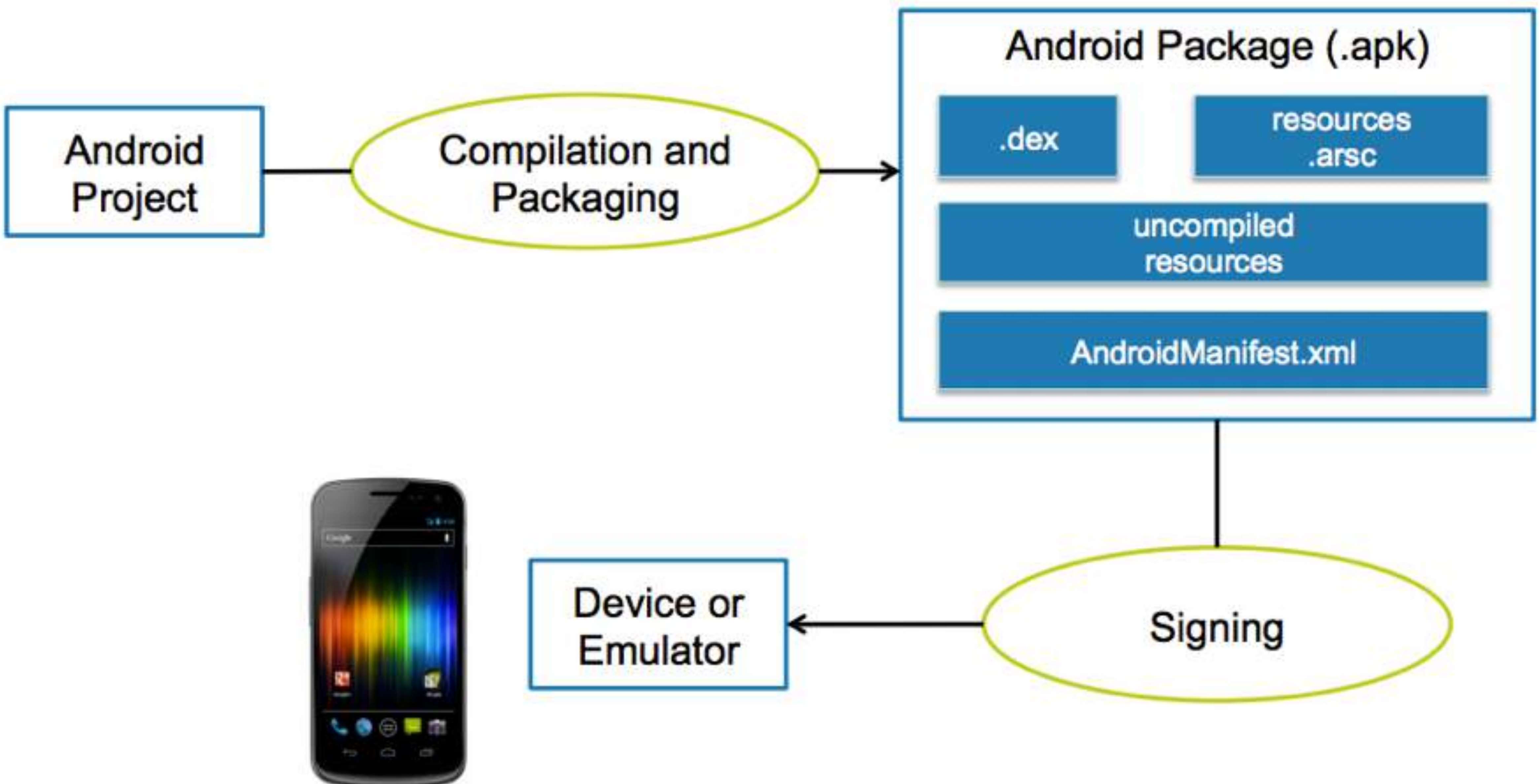
Linux Kernel

- Android made numerous additions and changes to the kernel.
- provide additional functionalities such as
  - camera access
  - wi-fi
  - binder driver (for inter-processes communication)

Main components of an Android app



APK building process



Source: http://developer.android.com/tools/building/index.html

Android Manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.novaapps.findevents" android:versionCode="1"
    android:versionName="1.0"
    android:installLocation="preferExternal">
    <uses-sdk android:minSdkVersion="4" />
    <supports-screens
        android:largeScreens="true"
        android:normalScreens="true"
        android:smallScreens="true"
        android:resizeable="true"
        android:anyDensity="true" />

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".FindEventsGADroidActivity"
            android:label="@string/app_name" android:configChanges="orientation|keyboardHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHEI
            </intent-filter>
        </activity>

        <activity android:name="com.phonegap.DroidGap" android:label="@
            android:configChanges="orientation|keyboardHidden">
            <intent-filter>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Unique package name

List of activities, services…

Permission definitions

External libraries

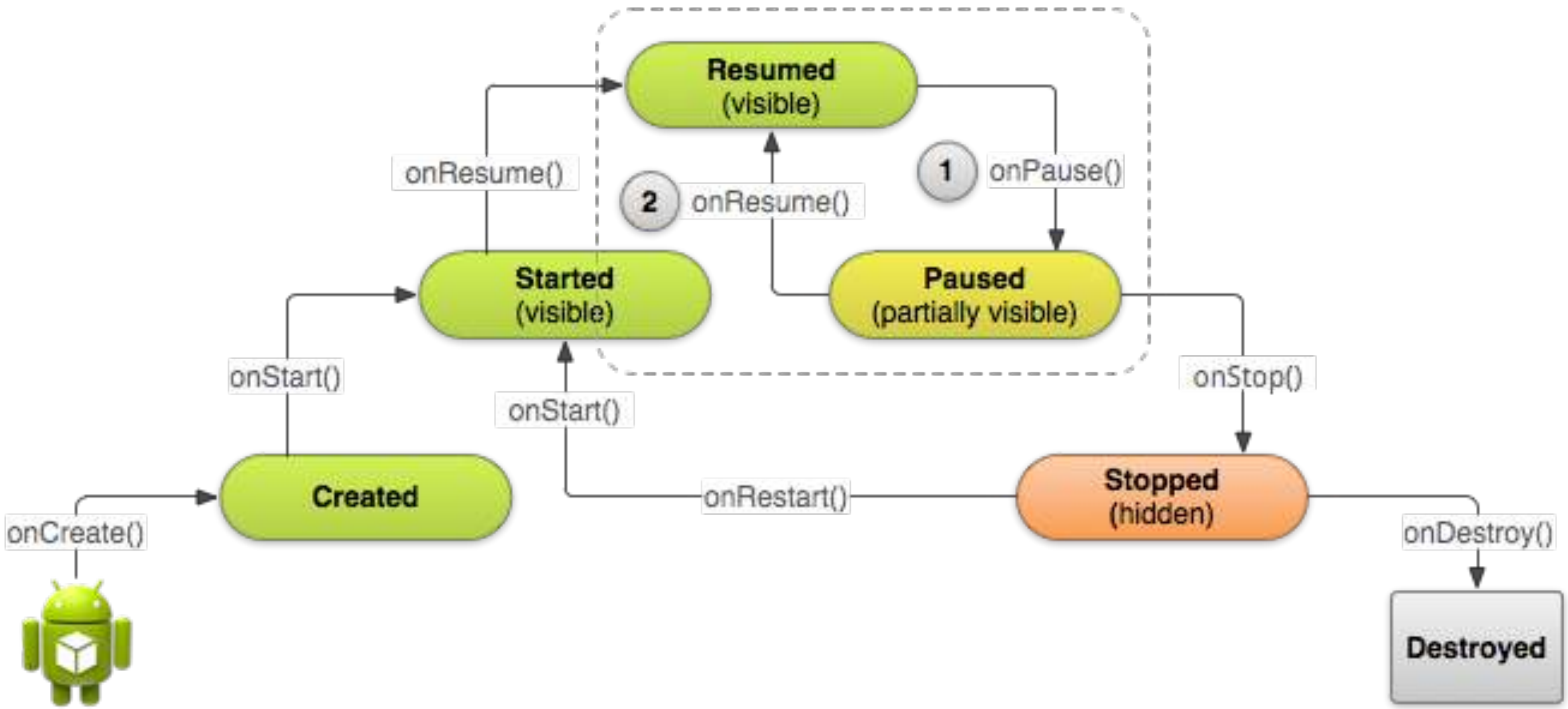
shared UID information

preferred installation location

Activities

- In essence it is the UI.
- An activity consists of a window along with several other UI elements.
- Activities are managed by the activity manager service (which also processes intents that are sent to invoke activities).

Activity life cycle



Services

- Application components without UI that run in the background.
- For example, SmsReceiver or BluetoothService
- Services can typically be stopped, started or bound all by way of Intents.

Intents

- Intents are the key part of inter-app communications.

- they are message objects that contain information about an operation to be performed (e.g. make a phone call)

- Intent can also be *implicit,* when they do not have a specific destination.

Broadcast Receivers

- Another component of the IPC.
- Commonly found where applications want to receive an implicit intent matching certain criteria (e.g. receive a SMS).
- They can also be registered at runtime (i.e. not necessarily in the Android Manifest)

Content providers

- Act as a structured interface to common shared data stores (typically SQLite).
- E.g. Contacts and Calendar providers manage centralized repositories with different entries
- Applications may have they own content provider, and may expose it to other apps.

Android security model

Security Boundaries

- Places in the system where the level of trust differs on either side
  - Boundary between kernel-space and user-space.
  - Code in kernel space is trusted to perform low-level operation and access physical memory.
  - Code in user-space cannot access all the memory.

Permissions in Android

- Android OS uses two separate but cooperative permission models
  - Low level: Linux kernel enforces permissions using users and groups (inherited by Linux)
  - Low level permission system is usually referred to as the Android sandbox.
  - High level: app permissions, which limit the abilities of Android apps.
  - The Android runtime/Dalvik VM enforce the high level model

Android’s sandbox

- Unix-like process isolation
- Principle of least privilege

Android sandbox

- Processes run as separate users and cannot interfere with each other (e.g. send signals or access one another's memory space)
- Unique user IDs for most processes
- Tightly restricted file system permissions

UID's

- Android shares Linux's UID/GID paradigm, but does not have the traditional *passwd* and *group* files for credentials.

- Android defines a map of names to unique identifiers known as Android IDs (AIDs)

- In addition to AIDs, Android uses supplementary groups to enable processes to access shared/protected resources (e.g. sdcard_rw)

At runtime

- When apps execute their UID, GID and supplementary groups are assigned to a newly created process.
- Running under unique UID and GID enables the operating system to enforce lower-level restrictions in the kernel
- Inter-app interaction is possible, and it is controlled by the runtime environment.

output of PS command

app_16	4089	1451	304080	31724	...	S	com.htc.bgp
app_35	4119	1451	309712	30164	...	S	com.google.android.calendar
app_155	4145	1451	318276	39096	...	S	com.google.android.apps.plus
app_24	4159	1451	307736	32920	...	S	android.process.media
app_151	4247	1451	303172	28032	...	S	com.htc.lockscreen
app_49	4260	1451	303696	28132	...	S	com.htc.weather.bg
app_13	4277	1451	453248	68260	...	S	com.android.browser

File system permissions

```
root@android:/ # ls -l /data/data
drwxr-x--x u0_a3    u0_a3 ... com.android.browser
drwxr-x--x u0_a4    u0_a4 ... com.android.calculator2
drwxr-x--x u0_a5    u0_a5 ... com.android.calendar
drwxr-x--x u0_a24   u0_a24 ... com.android.camera
...
drwxr-x--x u0_a55   u0_a55 ... com.twitter.android
drwxr-x--x u0_a56   u0_a56 ... com.ubercab
drwxr-x--x u0_a53   u0_a53 ... com.yougetitback.androidapplication.virgin.
mobile
drwxr-x--x u0_a31   u0_a31 ... jp.co.omronsoft.openwnn
```

Android permissions

- Permissions are required for:
  - System API calls
  - Database operations (content providers)
  - Inter Process Communications (send and receive *Intents*)

Application's permissions

- Extracted from the application's manifest at install time by the PackageManager and stored in /data/system/packages.xml

```
<package name="com.android.chrome"
codePath="/data/app/com.android.chrome-1.apk"
nativeLibraryPath="/data/data/com.android.chrome/lib"
flags="0" ft="1422a161aa8" it="1422a163b1a"
ut="1422a163b1a" version="1599092" userId="10082"
installer="com.android.vending">
<sigs count="1">
<cert index="0" />
</sigs>
<perms>
<item name="com.android.launcher.permission.INSTALL_SHORTCUT" />
<item name="android.permission.NFC" />
...
<item name="android.permission.WRITE_EXTERNAL_STORAGE" />
<item name="android.permission.ACCESS_COARSE_LOCATION" />
...
<item name="android.permission.CAMERA" />
<item name="android.permission.INTERNET" />
...
</perms>
</package>
```

API permissions

- e.g. READ_PHONE_STATE: Read only access to the phone state.
- An app that requires this permission would therefore be able to call a variety of methods related to querying the phone state

getDeviceSoftwareVersion()

getDeviceId()

IPC permissions

- e.g. CALL_PHONE: permission to start a phone call
- An application requires permissions to communicate with another app.

```
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse(...));
startActivity(intent);
```

Content Provider permissions

- e.g. READ_CONTACTS, WRITE_CONTACTS: read or write access to the contacts provider.
- An application requires permissions to access a resource at a given URI

State of the art in test input generation for Android

Inputs?

Android apps are highly interactive and event driven.

UI events (clicks, longclicks, text)

System events (sms received…)

Environment

Different strategies

Random

Systematic

- Model-based (static - dynamic)
- Search-based algorithms
- Symbolic-execution

Many useful available frameworks!

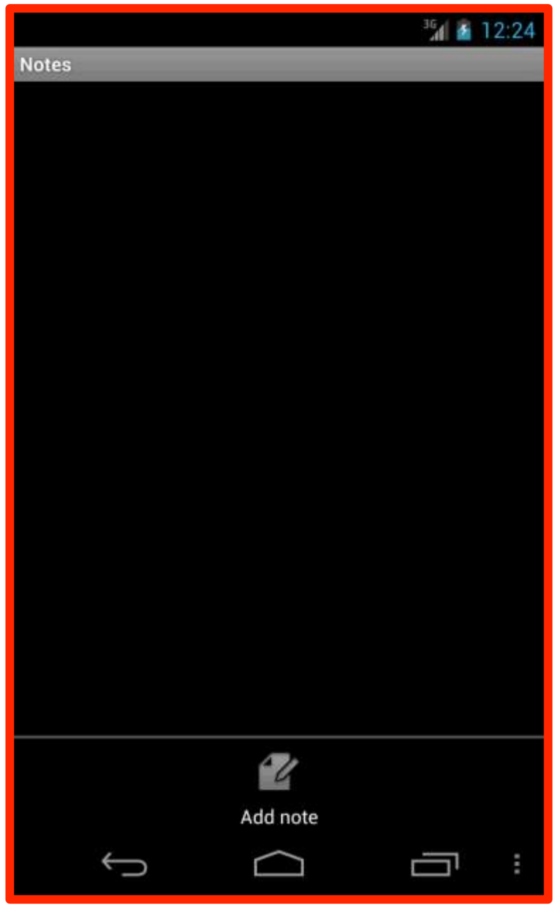
Useful Frameworks

- *UI automation*
  - Robotium
  - Espresso
  - UI automator
- *Static analysis*
  - DARE
  - Dex disassemblers
  - Soot and Flowdroid

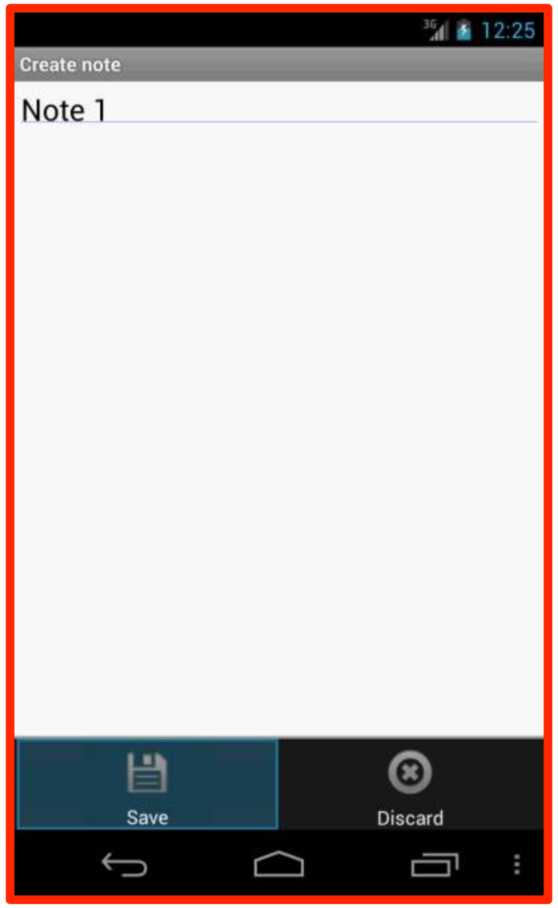
Robotium

An open source test framework
Used to write black or white box tests
Tests can be executed on an Android Virtual Device (AVD) or a real device
Built on Java and Android JUnit Test Framework

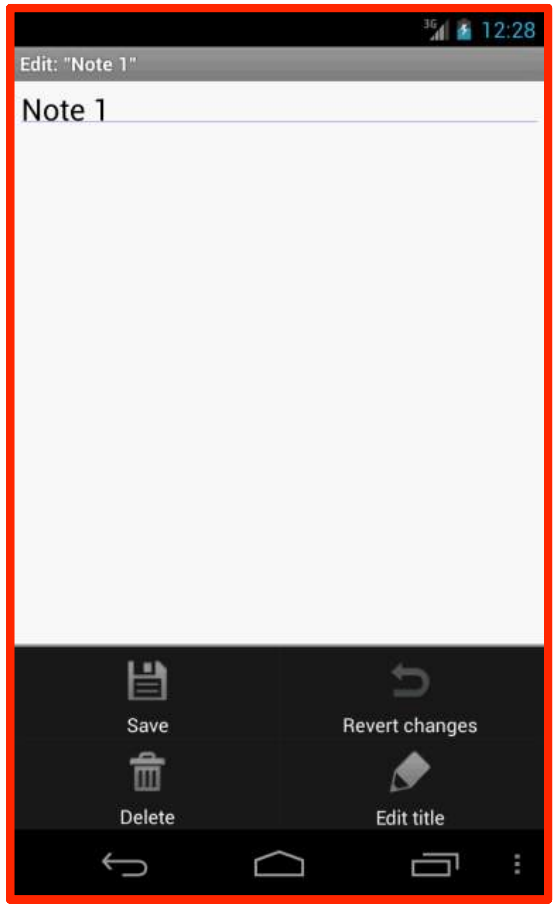
Notepad with Robotium



Add note



Save note



Edit note

Robotium

```
public void testAddNote() throws Exception {
  solo.clickOnMenuItem("Add note");
  //Assert that NoteEditor activity is opened
  solo.assertCurrentActivity("Expected NoteEditor activity", "NoteEditor");
  //In text field 0, enter Note 1
  solo.enterText(0, "Note 1");
  solo.goBack();
  //Clicks on menu item
  solo.clickOnMenuItem("Add note");
  //In text field 0, type Note 2
  solo.typeText(0, "Note 2");
  //Go back to first activity
  solo.goBack();
  //Takes a screenshot and saves it in "/sdcard/Robotium-Screenshots/".
  solo.takeScreenshot();
  boolean expected = true;
  boolean actual = solo.searchText("Note 1") && solo.searchText("Note 2");
  //Assert that Note 1 & Note 2 are found
  assertEquals("Note 1 and/or Note 2 are not found", expected, actual);
}
```

UIAutomator and Espresso

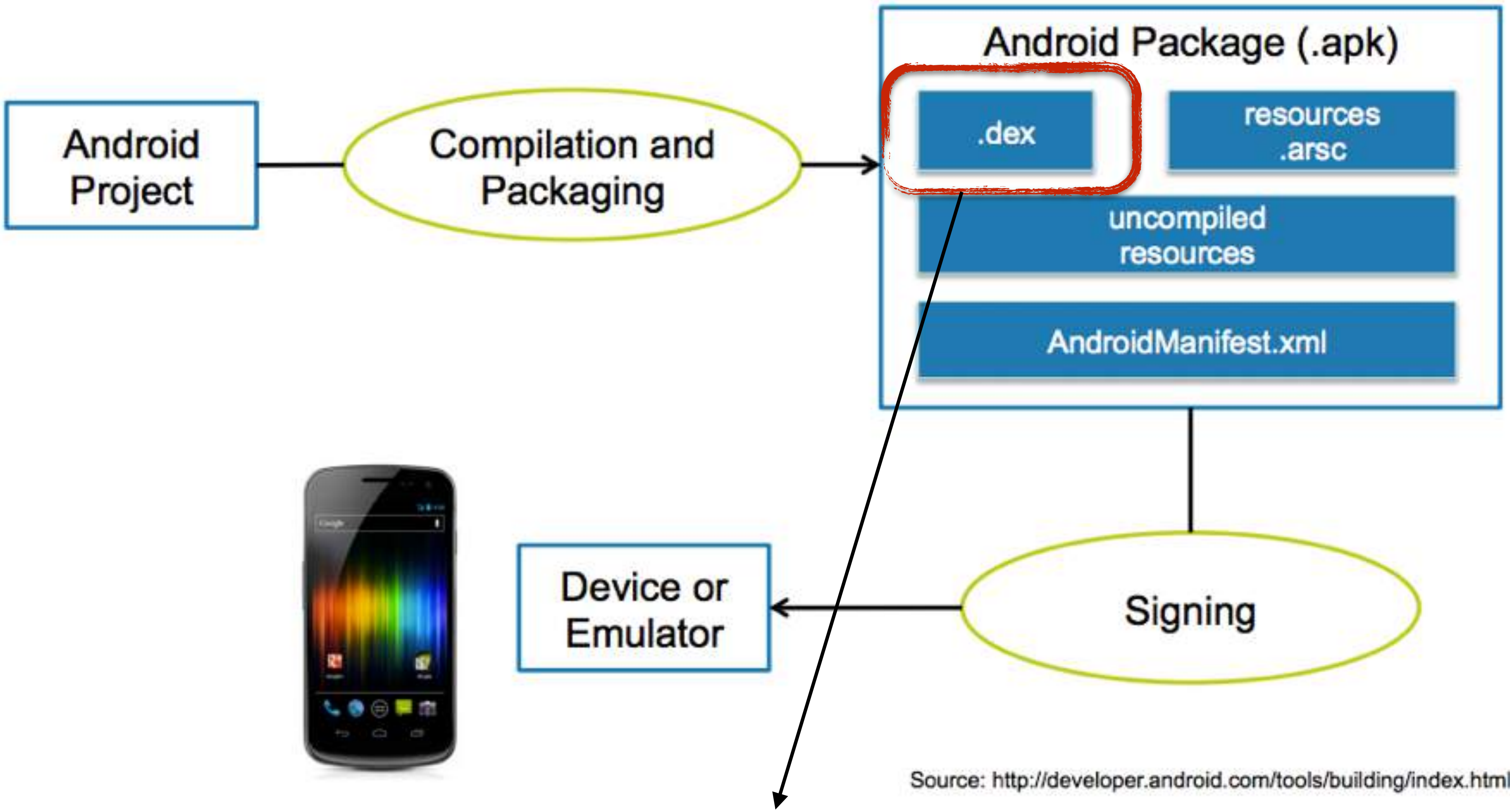
- UIAutomator is another framework that allows to build tests for user apps and system apps. (integration)
  - Perfect for implementing blackbox testing techniques.
  - Provide means to inspect the layout elements in activities.
- Espresso is another framework, more suitable for implementing whitebox testing techniques (single app)

Code coverage with emma

`ant emma debug install`

Program transformation for static analysis

Get the binary code



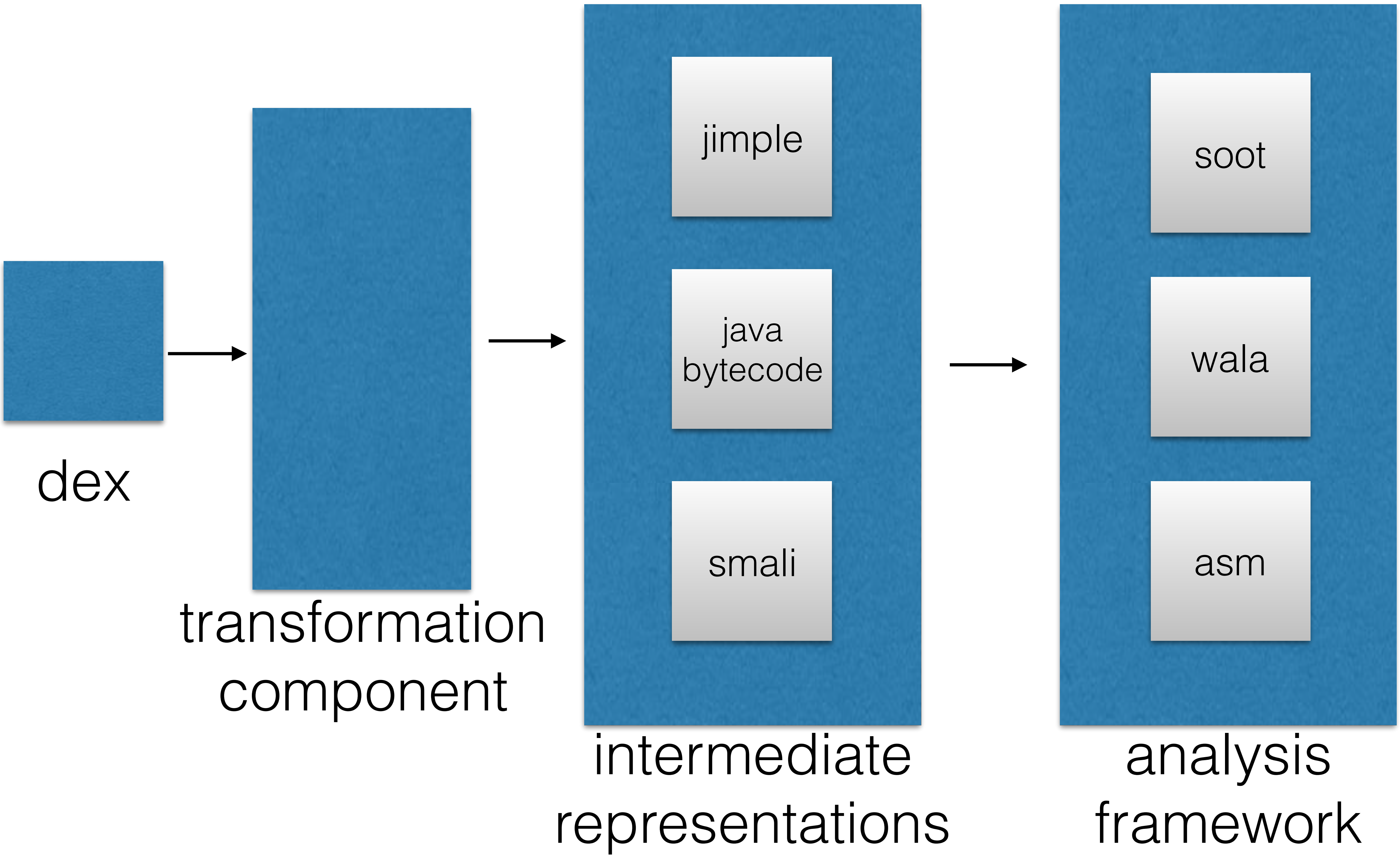
dexdump

dexdump

```
000418: 2b02 0c00 0000                          |0000: packed-switch v2, 0000000c // +0000000c
00041e: 12f0                                    |0003: const/4 v0, #int -1 // #ff
000420: 0f00                                    |0004: return v0
000422: 1220                                    |0005: const/4 v0, #int 2 // #2
000424: 28fe                                    |0006: goto 0004 // -0002
000426: 1250                                    |0007: const/4 v0, #int 5 // #5
000428: 28fc                                    |0008: goto 0004 // -0004
00042a: 1260                                    |0009: const/4 v0, #int 6 // #6
00042c: 28fa                                    |000a: goto 0004 // -0006
00042e: 0000                                    |000b: nop // spacer
000430: 0001 0300 faff ffff 0500 0000 0700 ... |000c: packed-switch-data (10 units)
```

not really easy to understand

Android app analysis



DEX disassemblers

- Other DEX disassembles can produce “more readable” outputs

  - **Dedexer**: turns the DEX format into an “assembly like” format. Influenced by Jasmin syntax but with Dalvik opcodes

  - **Smali/baksmali**: similar to dedexer, but well maintained (and acts as assembler as well)

  - **Androguard:** written in python. Provides some basic static analyses (check for similarities, navigate through cfgs, visualization)

Smali example

```
# class name, also determines file path when dumped
.class public Lcom/packageName/example;

# inherits from Object (could be activity, view, etc.)
# note class structure is L<class path="">;
.super Ljava/lang/Object;

# these are class instance variables
.field private someString:Ljava/lang/String;

# finals are not actually used directly, because references
# to them are replaced by the value itself
# primitive cheat sheet:
# V - void, B - byte, S - short, C - char, I - int
# J - long (uses two registers), F - float, D - double
.field public final someInt:I  # the :I means integer
.field public final someBool:Z # the :Z means boolean

# Do you see how to make arrays?
.field public final someCharArray:[C
.field private someStringArray:[Ljava/lang/String;

# this is the <init> of the constructor
# it calls the <init> of it's super, which in this case
# is Ljava/lang/Object; as you can see at the top
# the parameter list reads: ZLjava/lang/String;I
# Z - boolean
# Ljava/lang/String; - java String object
#   (semi-colon after non-primitive data types)
# I - integer
```

```
# these are not always present and are usuaully taken
# out by optimization/obfuscation but they tell us
# the names of Z, Ljava/lang/String; and I before
# when it was in Java
.parameter "someBool"
.parameter "someInt"
.parameter "exampleString"

# the .prologue and .line directives can be mostly ignored
# sometimes line numbers are useful for debugging errors
.prologue
.line 10

# p0 means parameter 0
# p0, in this case, is like "this" from a java class.
# we are calling the constructor of our mother class.
# what would p1 be?
invoke-direct {p0}, Ljava/lang/Object;-><init>()V

# store string in v0
const-string v0, "i will not fear. fear is the mind-killer."

# store 0xF hex value in v0 (or 15 in base 10)
# this destroys previous value string in v0
# variables do not have types they are just registers
# for storing any type of value.
# hexadecimal is base 15 is used in all machine languages
# you normally use base 10
# read up on it:
# http://en.wikipedia.org/wiki/Hexadecimal
```

Dare

- Retargeting android apps to Java bytecode
- Motivation (back in 2012): Reuse analyses that were already implemented on top of frameworks such as WALA and SOOT
- Aim: produce verifiable Java bytecode, which ensures it is analyzable by these frameworks.

Retargeting challenges

- **Type systems** are very different in DVM and JVM:
  - **Primitive assignments**: in Dalvik they specify only the width of the constant (32 vs 64 bits). No difference between float and int.
  - **Array load/store instructions**: DVM has array-specific load and store instructions for int and float arrays (a-get aput) and for long and double (aget-wide aput-wide). Type ambiguity again
  - **Object references**: Java bytecode uses null reference to detect undefined refs. Dalvik instead uses 0 to represent both number 0 and null refs.

DARE

- Works well in practice:
  - ~262,110 classes (top 50 apps of each of the 22 categories) —> successful retargeting for 99.09% of apps

Retargeting Android Applications to Java Bytecode
FSE 2012

Dexpler

- Converts Dalvik bytecode to Jimple intermediate representation.
- Jimple is the representation used in the Soot framework
- Built on top of dedexer
- Uses typing inference algorithm of soot (but deals with typing ambiguities)

Converting Android Dalvik Bytecode to Jimple for Static Analysis with Soot — SOAP12

Jimple

```
void foo() {
  double d1 = 3.0;
  double d2 = 2.0;
  int i1 = (int) (d1*d2);
  bar(this,i1);
}
```

```
void foo()
{
    Main this;
    double d1, d2, temp$0;
    int i1;

    this := @this: Main;
    d1 = 3.0;
    d2 = 2.0;
    temp$0 = d1 * d2;
    i1 = (int) temp$0;
    virtualinvoke this.<Main: void bar(Main,int)>(this, i1);
    return;
}
```

Challenges of the Android life cycle

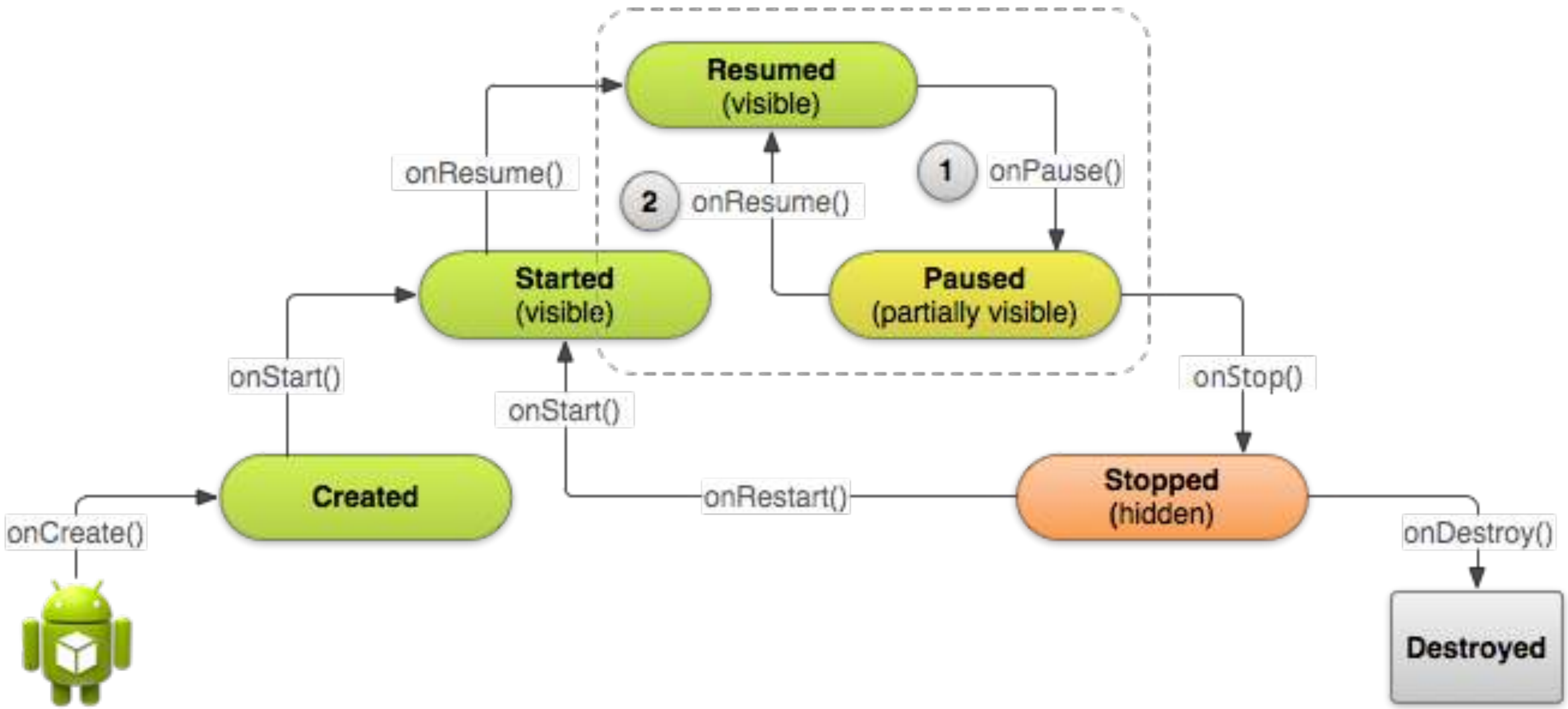
```java
public class LeakageApp extends Activity{
private User user = null;
protected void onRestart(){
  EditText usernameText =
    (EditText)findViewById(R.id.username);
  EditText passwordText =
    (EditText)findViewById(R.id.pwdString);
  String uname = usernameText.toString();
  String pwd = passwordText.toString();
  if(!uname.isEmpty() && !pwd.isEmpty())
    this.user = new User(uname, pwd);
}
//Callback method in xml file
public void sendMessage(View view){
  if(user == null) return;
  Password pwd = user.getpwd();
  String pwdString = pwd.getPassword();
  String obfPwd = "";
  //must track primitives:
  for(char c : pwdString.toCharArray())
    obfPwd += c + "_"; //String concat.

  String message = "User: " +
     user.getName() + " | Pwd: " + obfPwd;
  SmsManager sms = SmsManager.getDefault();
  sms.sendTextMessage("+44 020 7321 0905",
    null, message, null, null);
}
```

read pwd from text field when the app restarts

when the user presses a button the pwd is sent via sms

Important to model app life cycle and callbacks!!

Activity life cycle



Automated testing in Android

Automated Testi Input Generation for Android: Are We There Yet? — *under submission*
http://arxiv.org/abs/1503.07217

Fuzzing

“ab’d&gfdfggg” grep • sh • sed … 25%–33%

…

Send "!o%888888888f" as command to the csh command-line shell

Invoke this with string ="%888888888f":

```
char *string = …
printf(string);
```

…and made the shell hang

Fuzzing in Android

- Mildly widely used so far.
- Fuzzing mainly focused on IPC

Null intent fuzzer

- Very simple fuzzer: Null intents
    - Create null intents and see whether the broadcast receivers registered to those intents crash.

Null intent fuzzer

- **Identify targets:**
  - thanks to PackageManager
- **Generate intents**
  - Intent i = new Intent()
- **Deliver inputs**
  - sendBroadcast(i)
- **Monitor**
  - logcat.. —> NullPointerExceptions

Null intent fuzzer

iSECpartners
part of nccgroup

"can either fuzz a single component or all components. It works well on Broadcast receivers, and average on Services".

Only single Activities can be fuzzed.

Runs on device as an app, opensource

Detected a serious bug in a google package that makes the phone hang

Intent fuzzer

- Works exactly like null intent fuzzer
- Static analysis component that can detect the expected structure of an intent.
  - Works with inputs of primitive types

Intent Fuzzer: Crafting intents of death
WODA+PERTEA 2014

DroidFuzzer

- It focuses on generating inputs for activities that accept MIME data types (AVI, MP3, HTML files)
- It can make video player apps crash
- Tool not available

DroidFuzzer: Fuzzing the Android apps with Intent-filter tag — MoMM 2013

Automated GUI testing in Android

Randomized GUI testing

Monkey

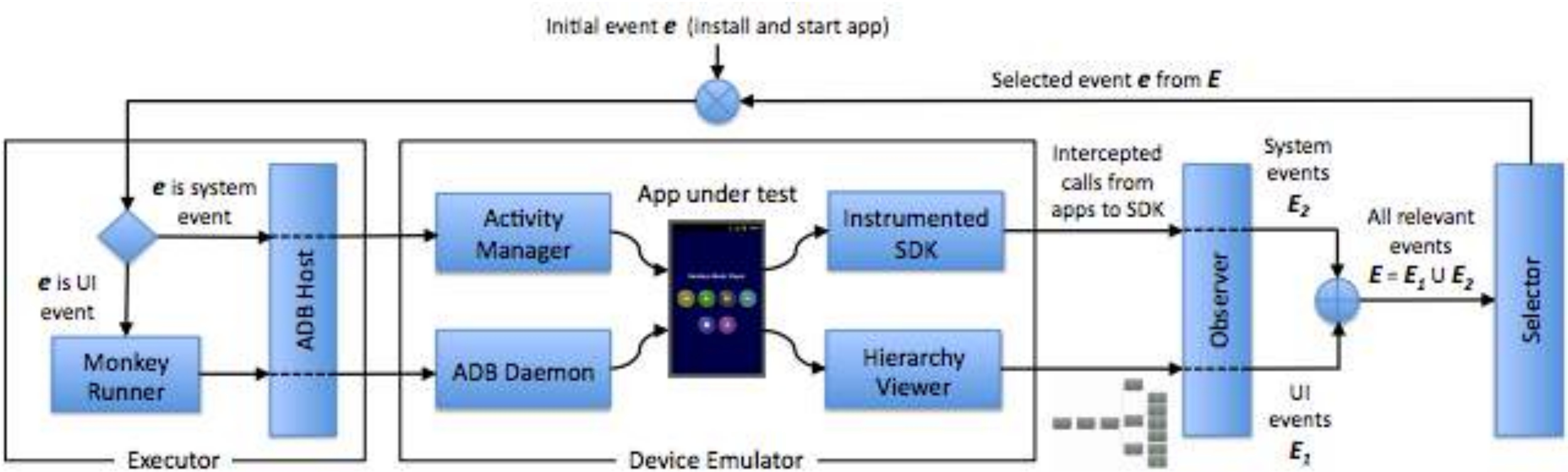
Tests Android apps at the GUI level
Randomly generates UI events

Runs on emulator or real device

$ adb shell monkey

Dynodroid

- Executor executes the event in the current state to yield a new emulator state (that overwrites the current state)
- Observer computes which events are relevant in the new state
- Selector selects one of the events to execute

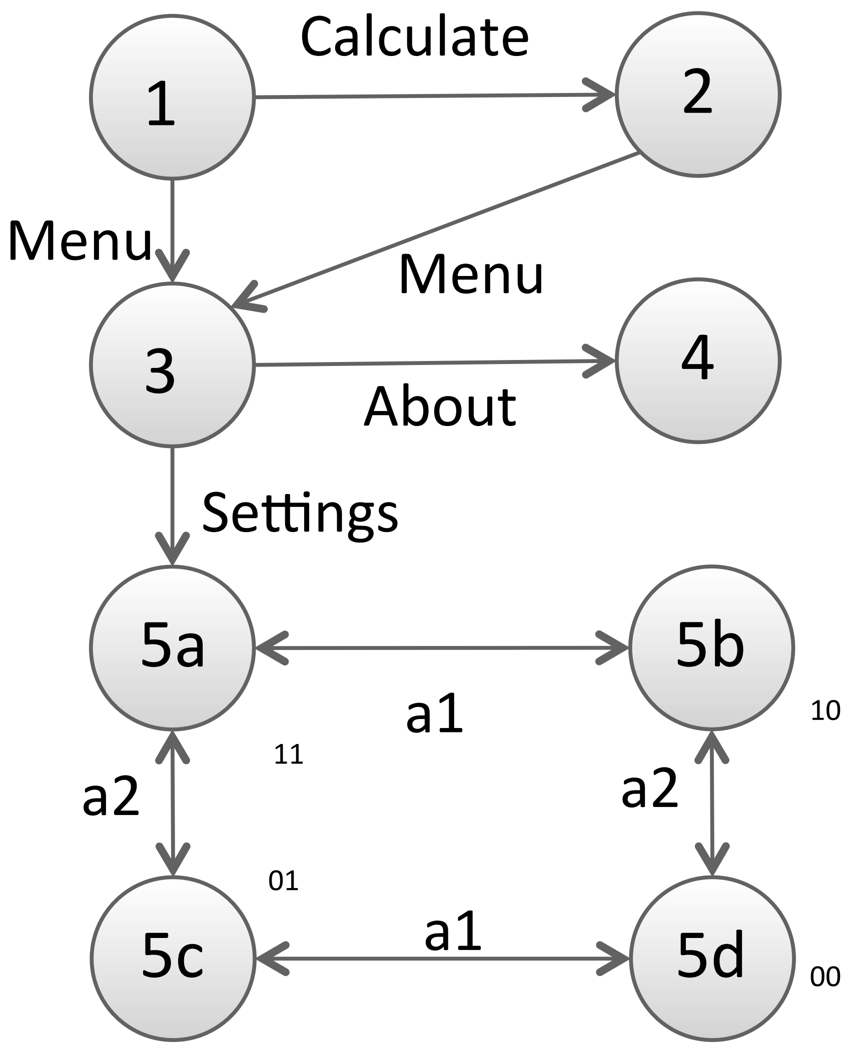


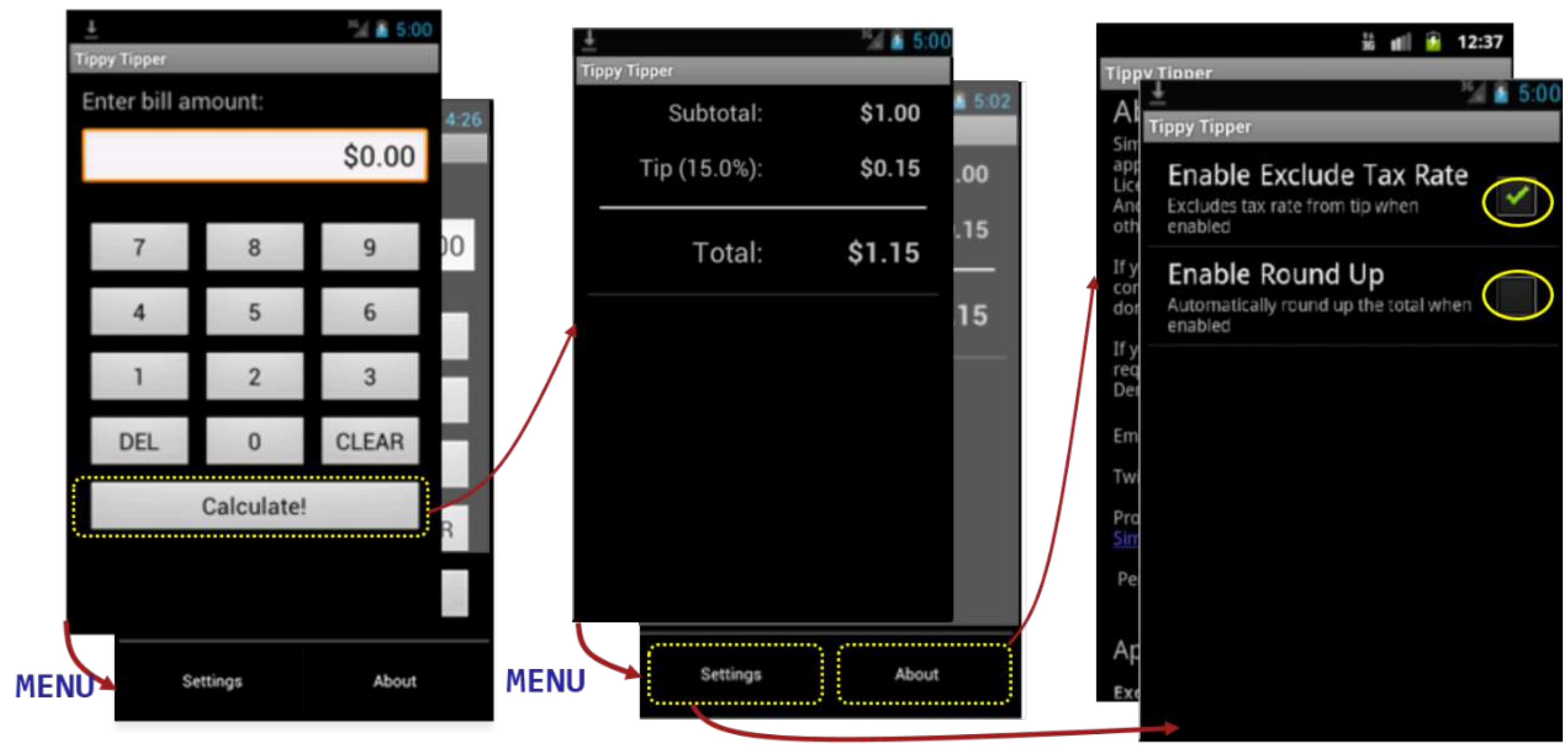
Dynodroid

- How to generate relevant inputs?
  - First generate it randomly but… It lets *users* pause the automated crawling and let them provide an input.

Dynodroid: An Input Generation System for Android Apps — ESEC/FSE13

Model-based techniques



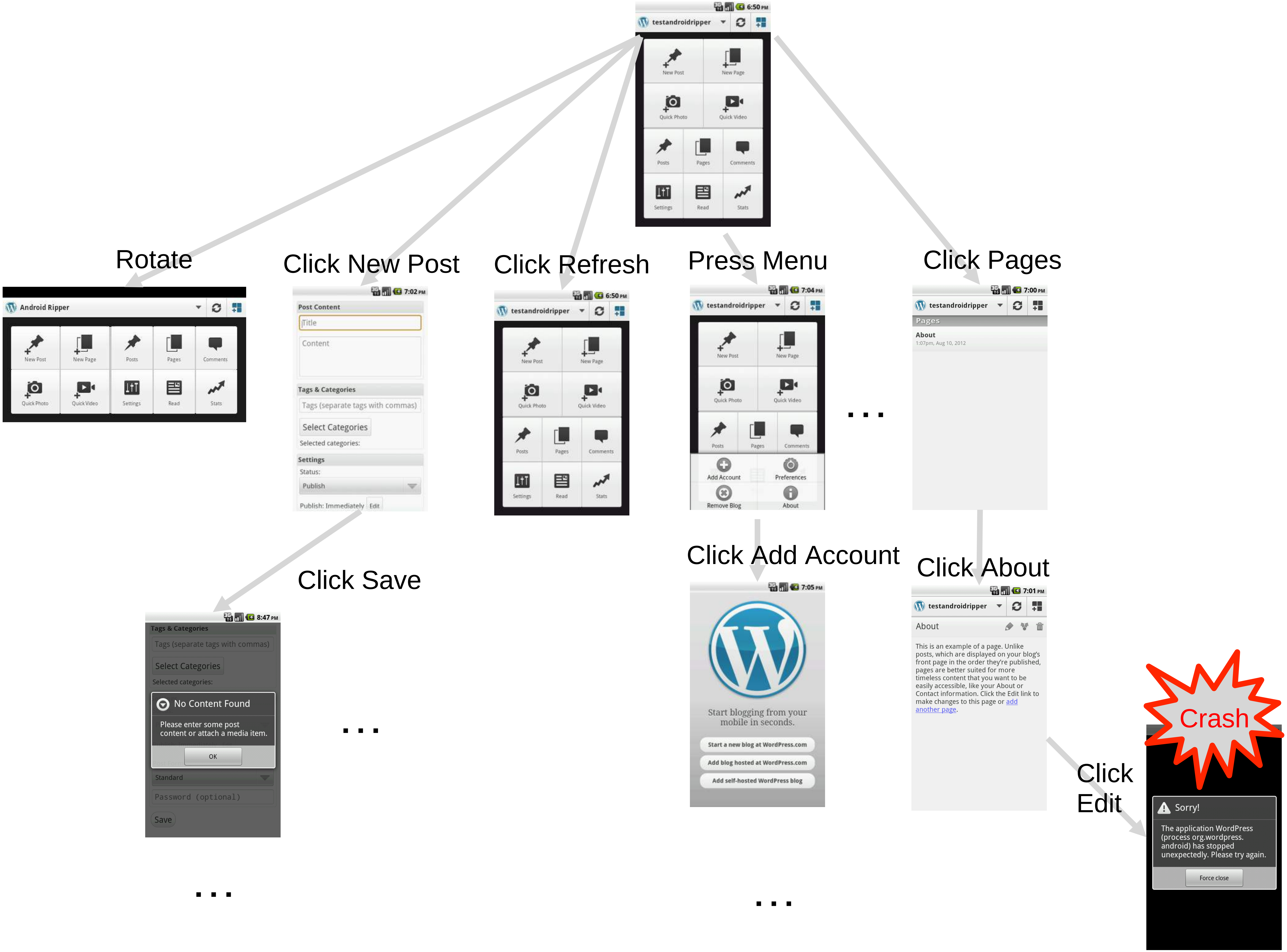


GUIRipper

- Dynamically builds FSM model
- DFS exploration strategy
- At each step it keeps list of relevant UI events

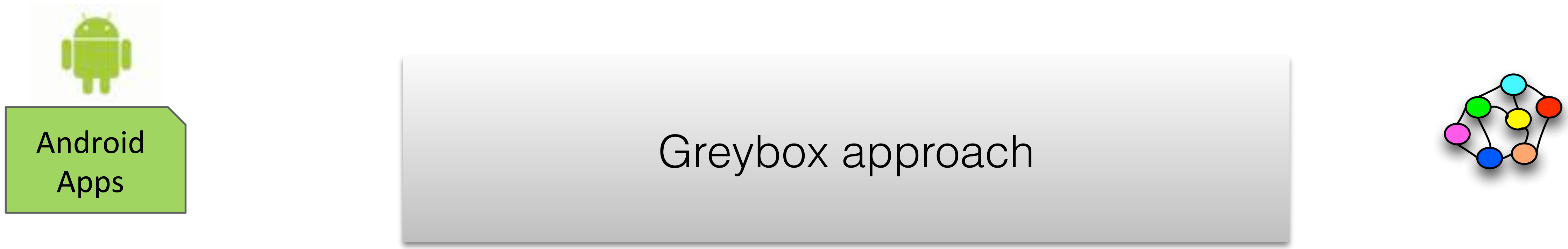
Allows users to create snapshots
and provide custom inputs

Using GUI Ripping for Automated Testing of Android Applications — ASE12



testandroidripper
New Post
New Page
Quick Photo
Quick Video
Posts
Pages
Comments
Settings
Read
Stats
Rotate
Click New Post
Click Refresh
Press Menu
Click Pages
Android Ripper
Post Content
Title
Content
Tags & Categories
Tags (separate tags with commas)
Select Categories
Selected categories:
Settings
Status:
Publish
Publish: Immediately
Edit
Add Account
Preferences
Remove Blog
About
Pages
About
1:07pm, Aug 10, 2012
Click Save
Click Add Account
Click About
No Content Found
Please enter some post content or attach a media item.
OK
Standard
Password (optional)
Save
Start blogging from your mobile in seconds.
Start a new blog at WordPress.com
Add blog hosted at WordPress.com
Add self-hosted WordPress blog
This is an example of a page. Unlike posts, which are displayed on your blog's front page in the order they're published, pages are better suited for more timeless content that you want to be easily accessible, like your About or Contact information. Click the Edit link to make changes to this page or add another page.
Click Edit
Crash
Sorry!
The application WordPress (process org.wordpress.android) has stopped unexpectedly. Please try again.
Force close

Orbit GUI testing



Statically extracts all the possible set of events supported by the GUI on an app.

Dynamically exercises these events on the app.

A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications — FASE13

Proposed GUI model

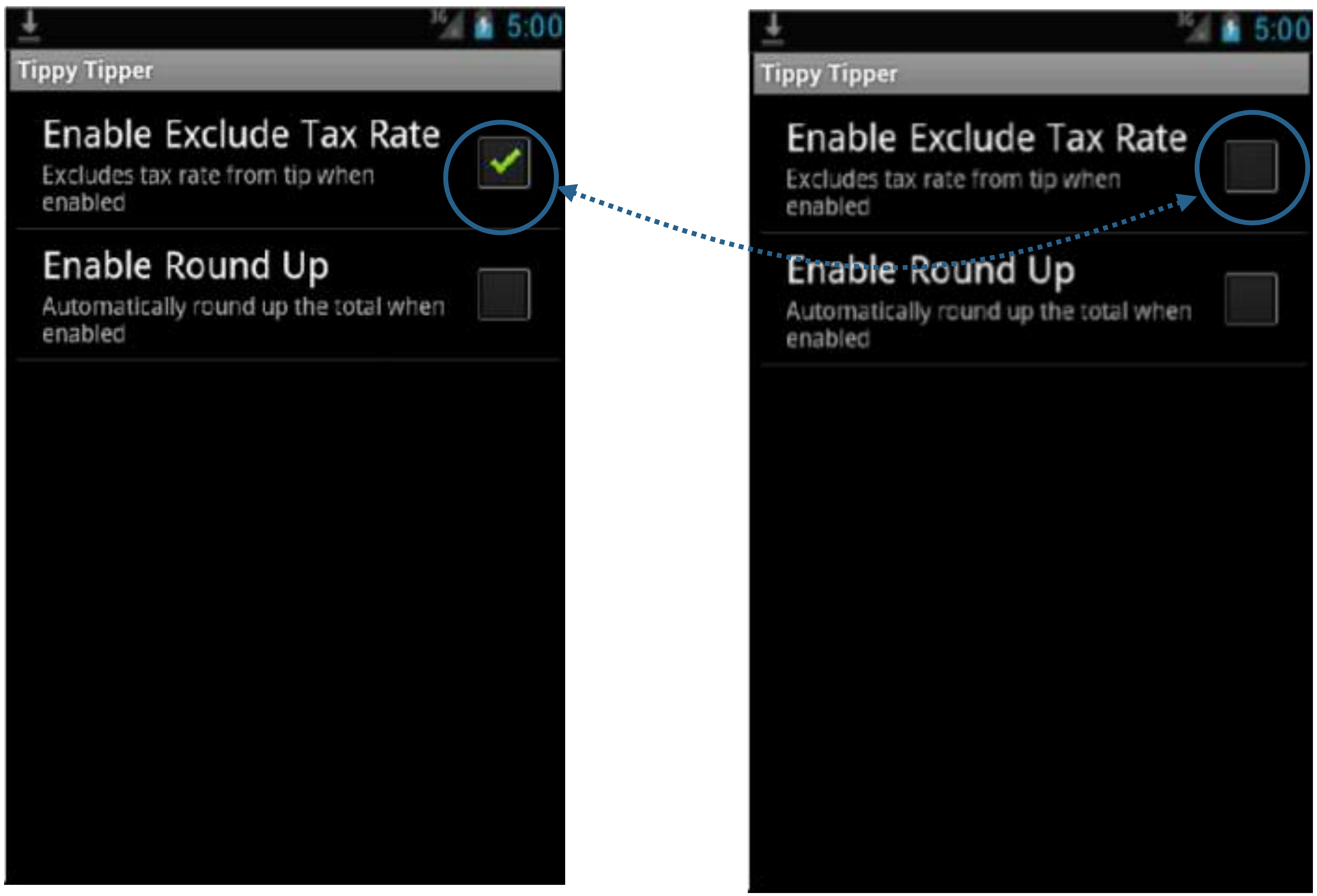
Visual Observable State

Composition of the state

Model

A finite-state machine over visual observable states with the user actions constituting the transitions between these states

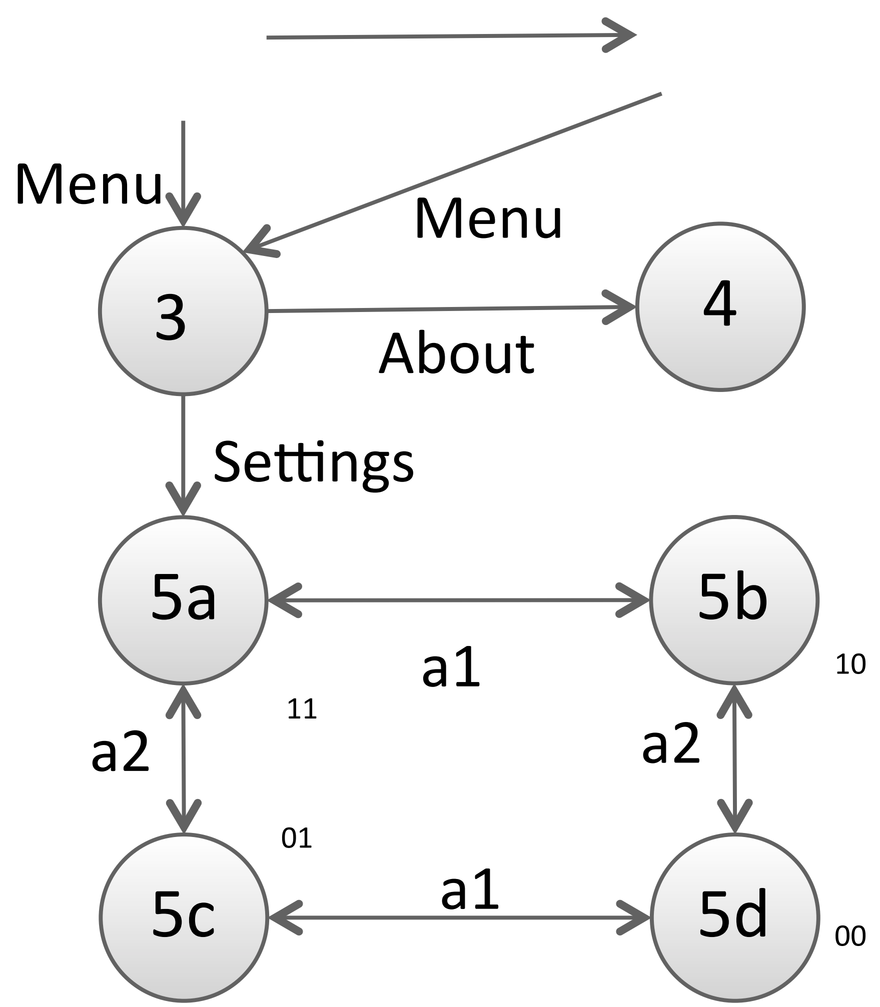
States



(5b) (5d)

These two states differ

Model for Simple TippyTipper



a1: Toggle exclude tax rate option.
a2: Toggle round up option.

Action Inference

TippyTipper.java

```
View btn_delete = findViewById(R.id.btn_delete);
    ⋮
Btn_delete.setOnClickListener(new onClickListener() {
        public void onClick(View v) {
            removeBillAmount();
            FlurryAgent.onEvent("Delete Button");
        }
    });
    ⋮
Btn_delete.setOnLongClickListener(new onLongClickListener() {
        public void onLongClick(View v) {
            clearBillAmount();
            return true;
        }
    });
```

R.Id.java

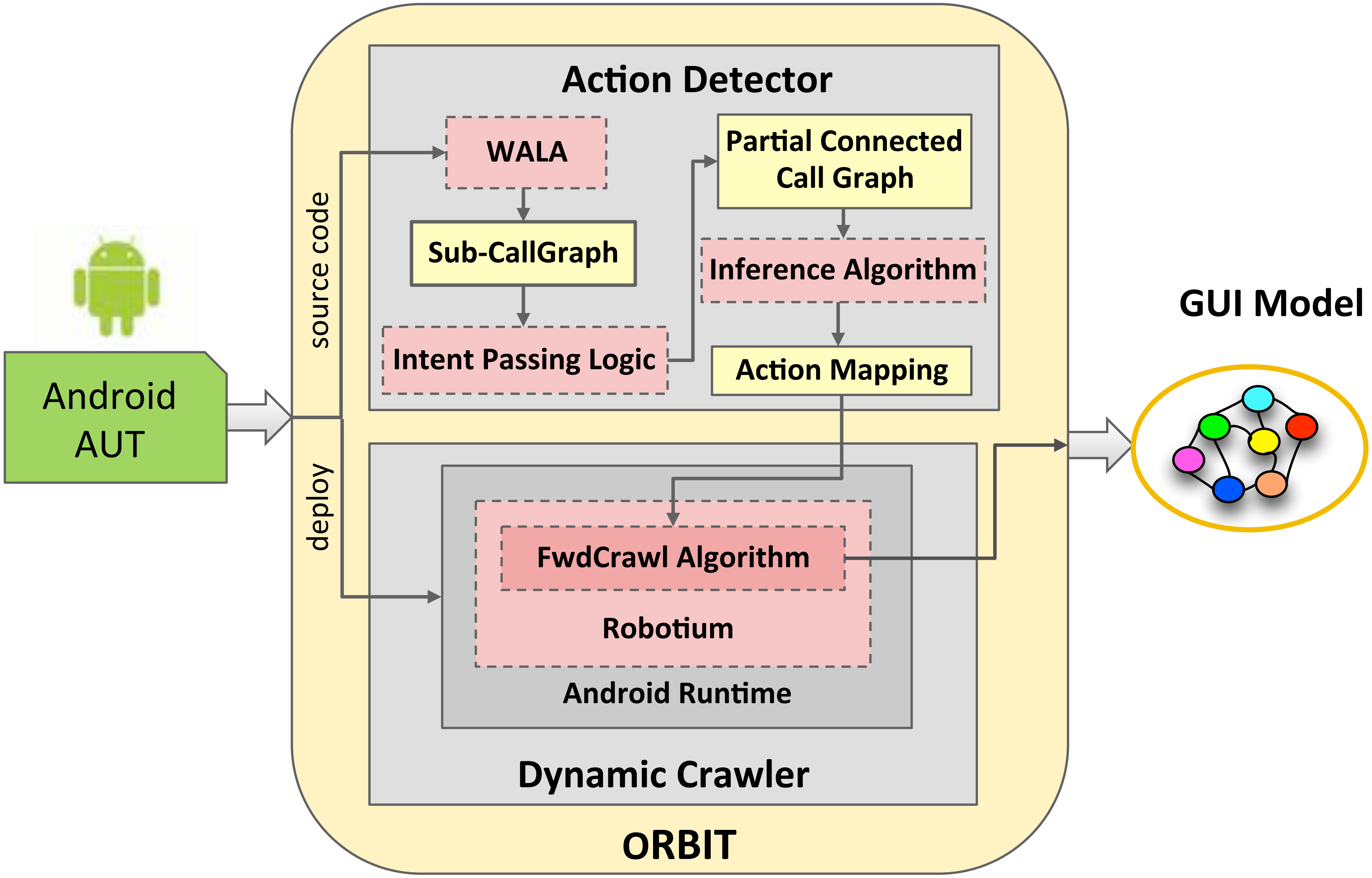
```
public static final class id {
    public static final int about=0x7f0a002d;
    public static final int btn_SplitBill=0x7f0a002b;
    public static final int btn_TipAmount1=0x7f0a0026;
    public static final int btn_TipAmount2=0x7f0a0027;
    public static final int btn_TipAmount3=0x7f0a0028;
    public static final int btn_add_person=0x7f0a001a;
    public static final int btn_clear=0x7f0a000c;
    public static final int btn_delete=0x7f0a000a;
    public static final int btn_eight=0x7f0a0002;
```

Inference: Widget *btn_delete* with *Id* = *0x7f0000a* supports actions *click* and *longClick*

ORBIT: static analysis

- Identify components on which to fire an event (e.g. longClick):
  - build call graph to find methods that call setOnLongClickListerer
  - locate statement in the caller method and get the object the listener is registered to.
  - backward analysis to get to the object initialization to get ID
  - add ID+action to list of actions to be triggered dynamically

Implementation



Automatic Android App explorer (A3E)

- Does not require access to source code
- Targeted and Depth-first visiting strategy
  - Higher level of abstraction (1 activity, 1 state)
  - Targeted strategy uses static analysis to compute all the activities as entry points (to analyse all of them)

Targeted and Depth-first Exploration for Systematic Testing of Android Apps — OOPSLA13

Swifthand

- Dynamic model of the app. Exploration algorithm aims to reduce the number of restarts as much as possible.
- limited to touching and scrolling events

Guided GUI Testing of Android Apps with Minimal Restart and Approximate Learning — OOPSLA13

PUMA

- Framework that provides a basic monkey-like implementation.
- provides a model-based representation of an app
- possible to implement different levels of abstraction

PUMA: Programmable UI-Automation for Large Scale Dynamic Analysis of Mobile Apps — Mobysys14

Limitation of model-based strategy?

- Changes in internal states not represented in the model
- Problem for services

EvoDroid

- Evodroid: Uses evolutionary algorithms to guide the test-case generation towards unexplored code
- individuals as sequences of test inputs
  - mutation and crossover operators to recombine inputs
- tool not available

EvoDroid: Segmented Evolutionary Testing of Android Apps — FSE14

ACTEve

- Concolic testing tool that symbolically tracks events from their generation up to the point where they are handled in the app.
- Works both on system and UI events

Automated Concolic Testing of Smartphone Apps — FSE12

JPF-Android

- extends JPF, the popular model-checking tool for Java.
- aims to explore all paths to detect deadlocks and runtime exceptions
    - limitation: assumes that user provides the list of inputs.

Summary of tools

Name	Available	Instrumentation		Events		Exploration strategy	Needs source code	Testing strategy
		Platform	App	UI	System			
Monkey [10]	✓	×	×	✓	×	Random	×	Black-box
Dynodroid [11]	✓	✓	×	✓	✓	Random	×	Black-box
DroidFuzzer [12]	✓	×	×	×	×	Random	×	Black-box
IntentFuzzer [13]	✓	×	×	×	×	Random	×	White-box
Null IntentFuzzer [14]	✓	×	×	×	×	Random	×	Black-box
GUIRipper [15]	✓[a]	×	✓	✓	×	Model-based	×	Black-box
ORBIT [16]	×	×	×	✓	×	Model-based	✓	Grey-box
A^3E -Depth-first [17]	✓	×	✓	✓	×	Model-based	×	Black-box
SwiftHand [18]	✓	×	✓	✓	×	Model-based	×	Black-box
PUMA [19]	✓	×	✓	✓	×	Model-based	×	Black-box
A^3E -Targeted [17]	×	×	✓	✓	×	Systematic	×	Grey-box
EvoDroid [20]	×	×	✓	✓	×	Systematic	×	White-box
ACTEve [21]	✓	✓	✓	✓	✓	Systematic	✓	White-box
JPF-Android [22]	✓	×	×	✓	×	Systematic	✓	White-box

Aim of the study

Ease of use

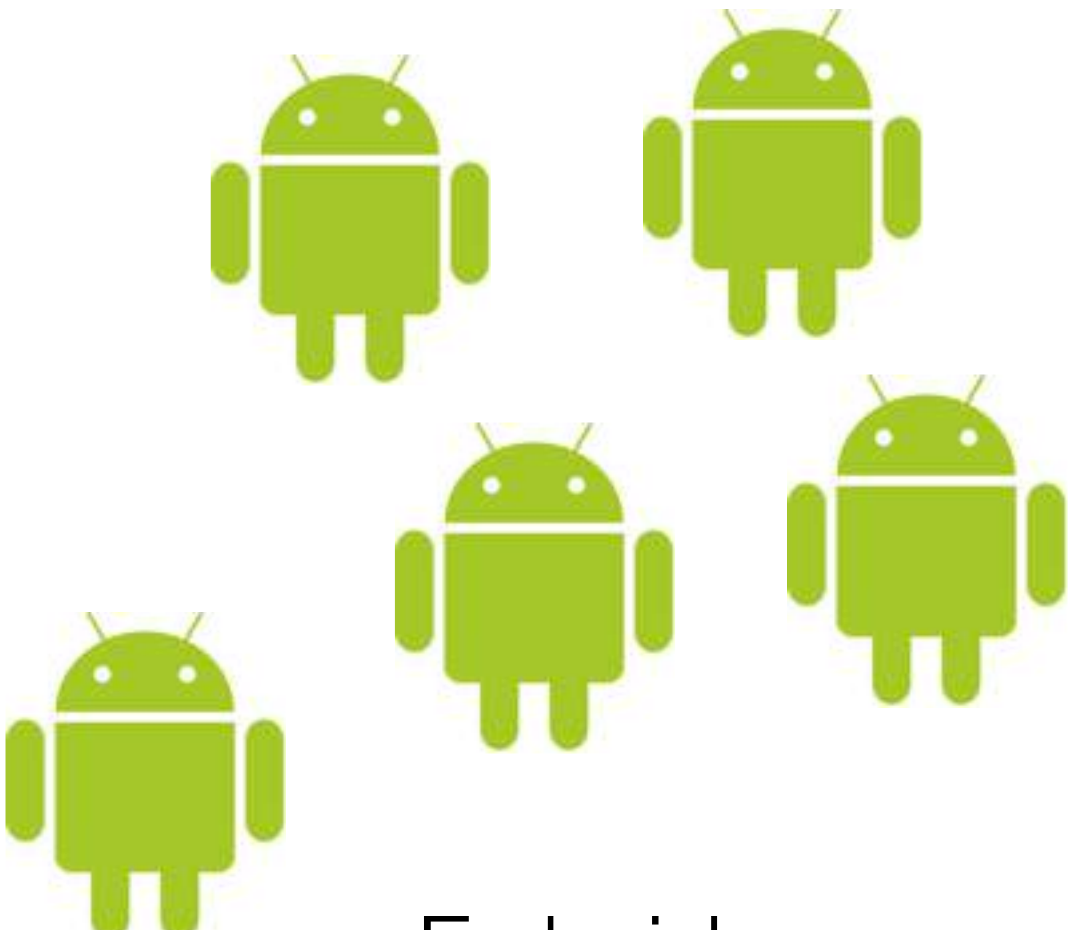
Android framework compatibility

Effectiveness of exploration strategy

Fault detection ability

Automated Test input Generation for Android:
Are we there yet? S. Roy Choudhary, A.Gorla, A.Orso - under submission

Benchmark



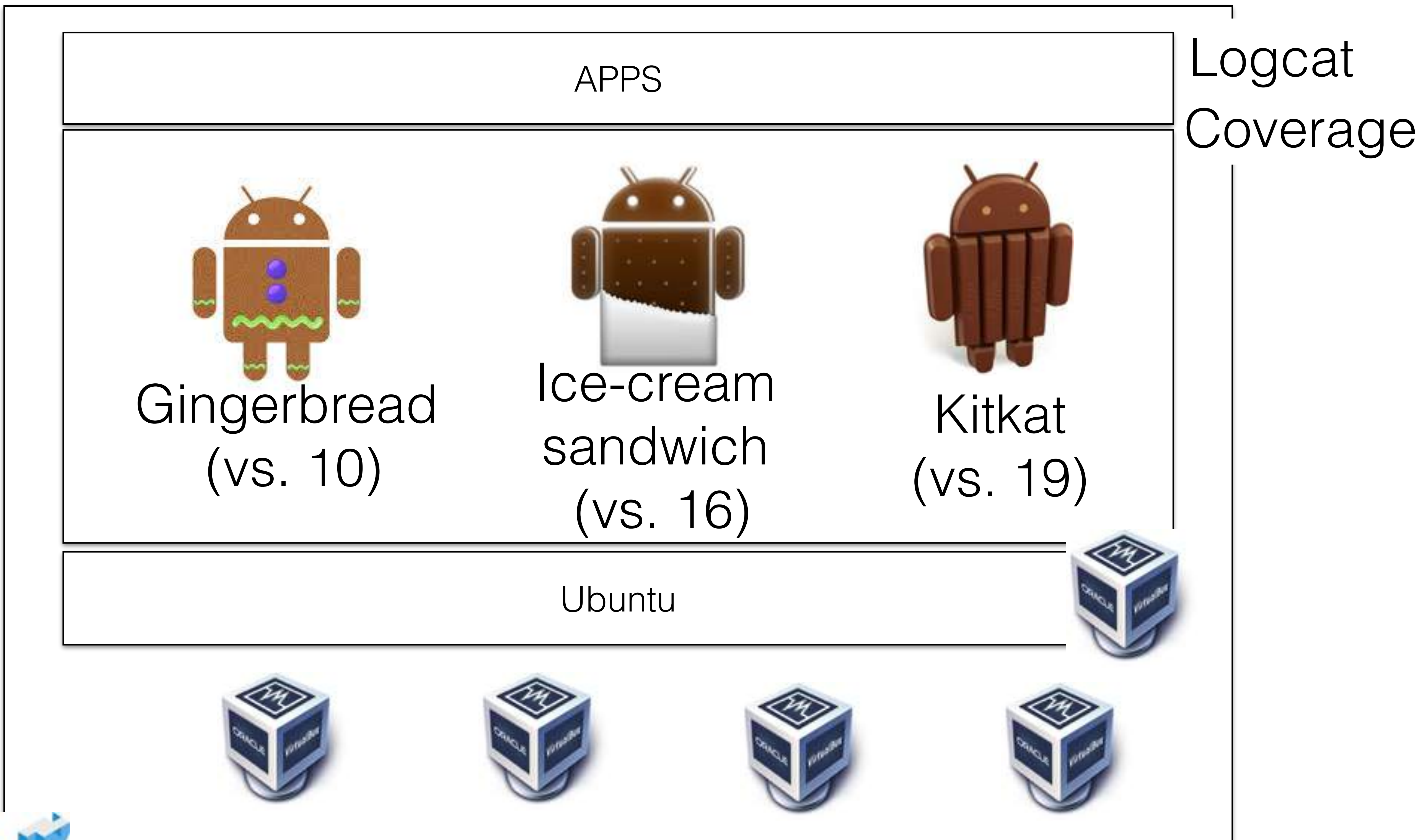
F-droid
68 apps

50 from Dynodroid

3 from GUIRipper

5 from ACTEve

10 from Swifthand

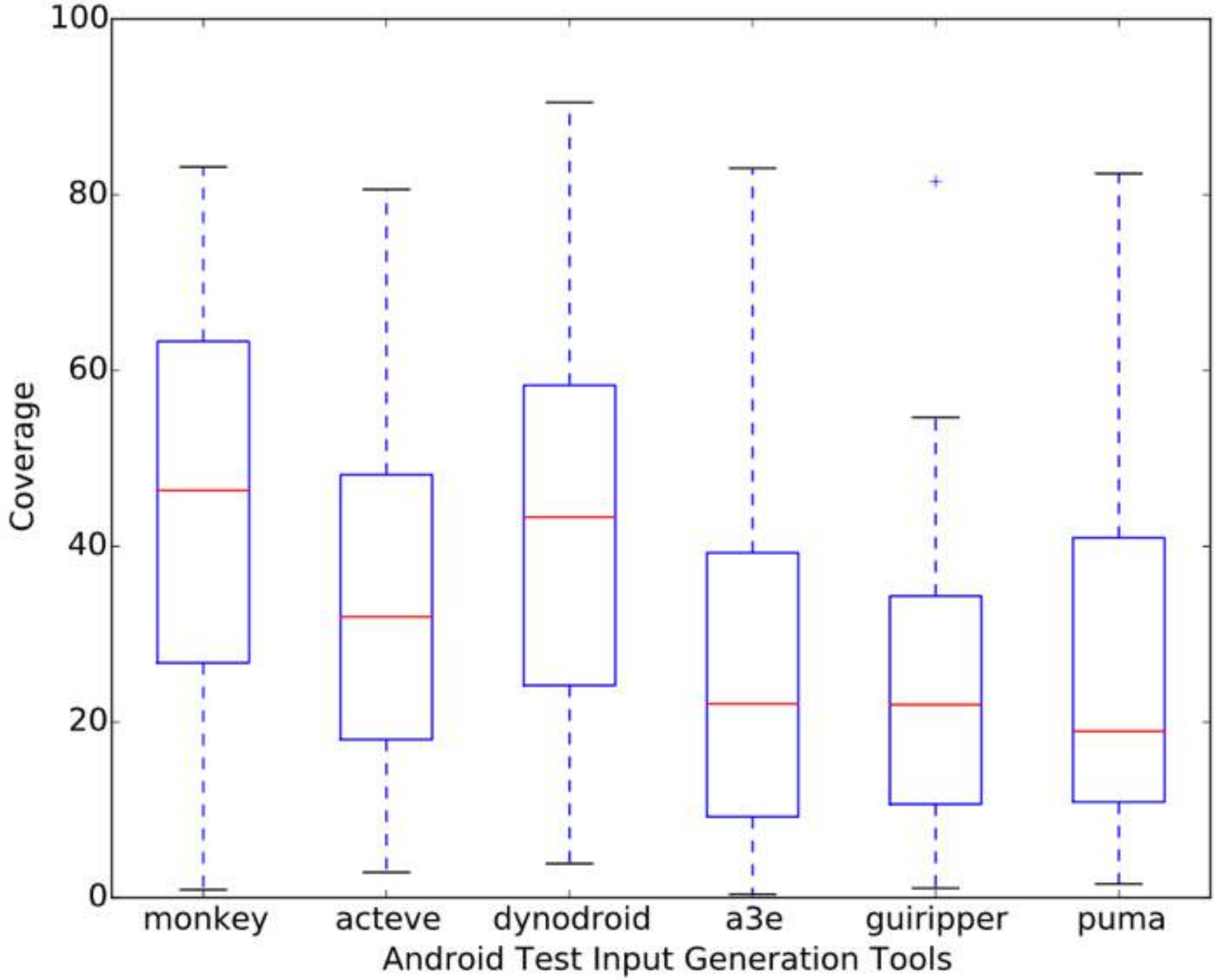


10 runs of 1 hour for each tool on each app

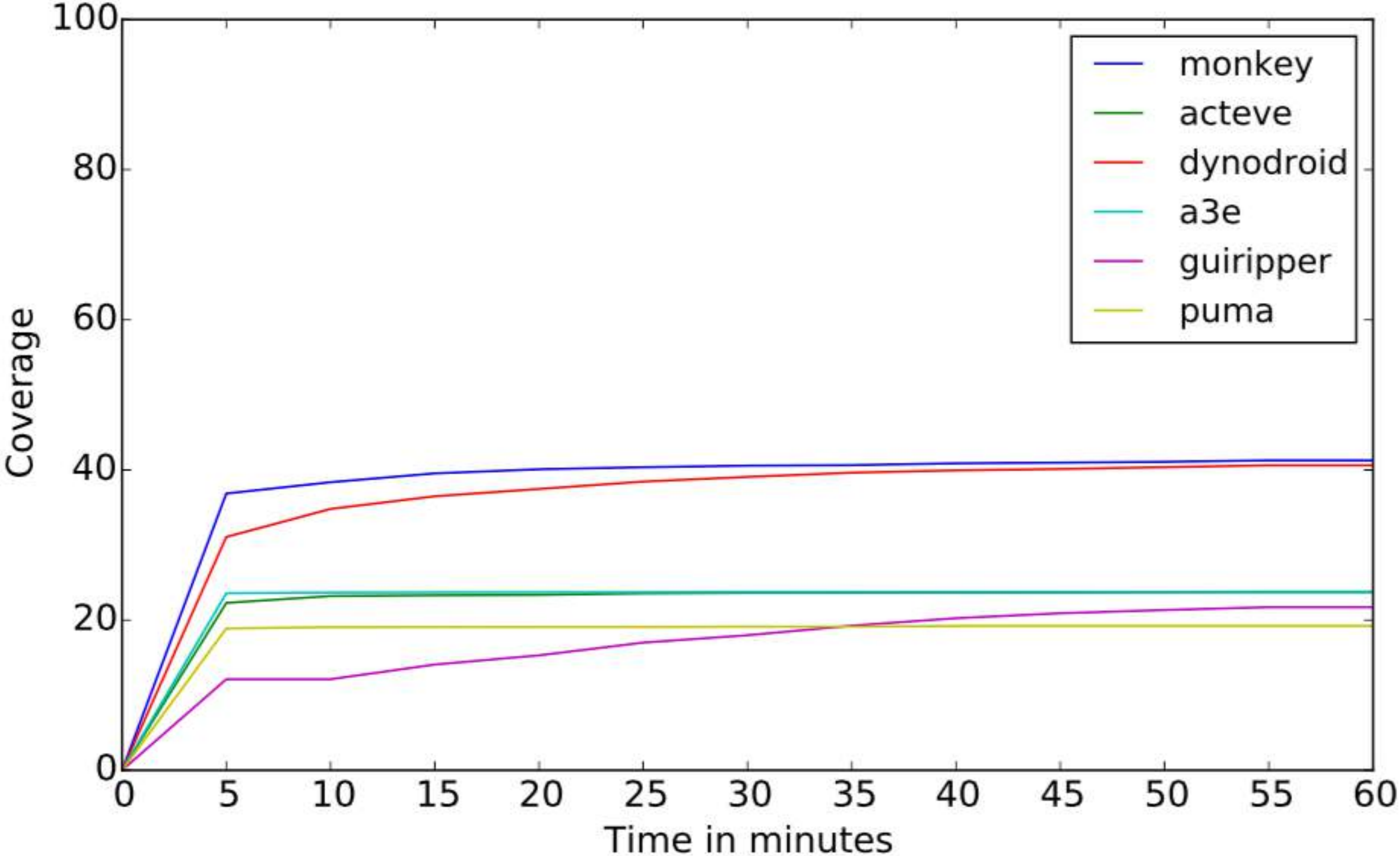
Ease of use and compatibility

Name	Ease Use	Compatibility
Monkey [10]	NO_EFFORT	any
Dynodroid [11]	NO_EFFORT	v.2.3
GUIRipper [15]	MAJOR_EFFORT	any
A^3E -Depth-first [17]	LITTLE_EFFORT	any
SwiftHand [18]	MAJOR_EFFORT	v.4.1+
PUMA [19]	LITTLE_EFFORT	v.4.3+
ACTEve [21]	MAJOR_EFFORT	v.2.3

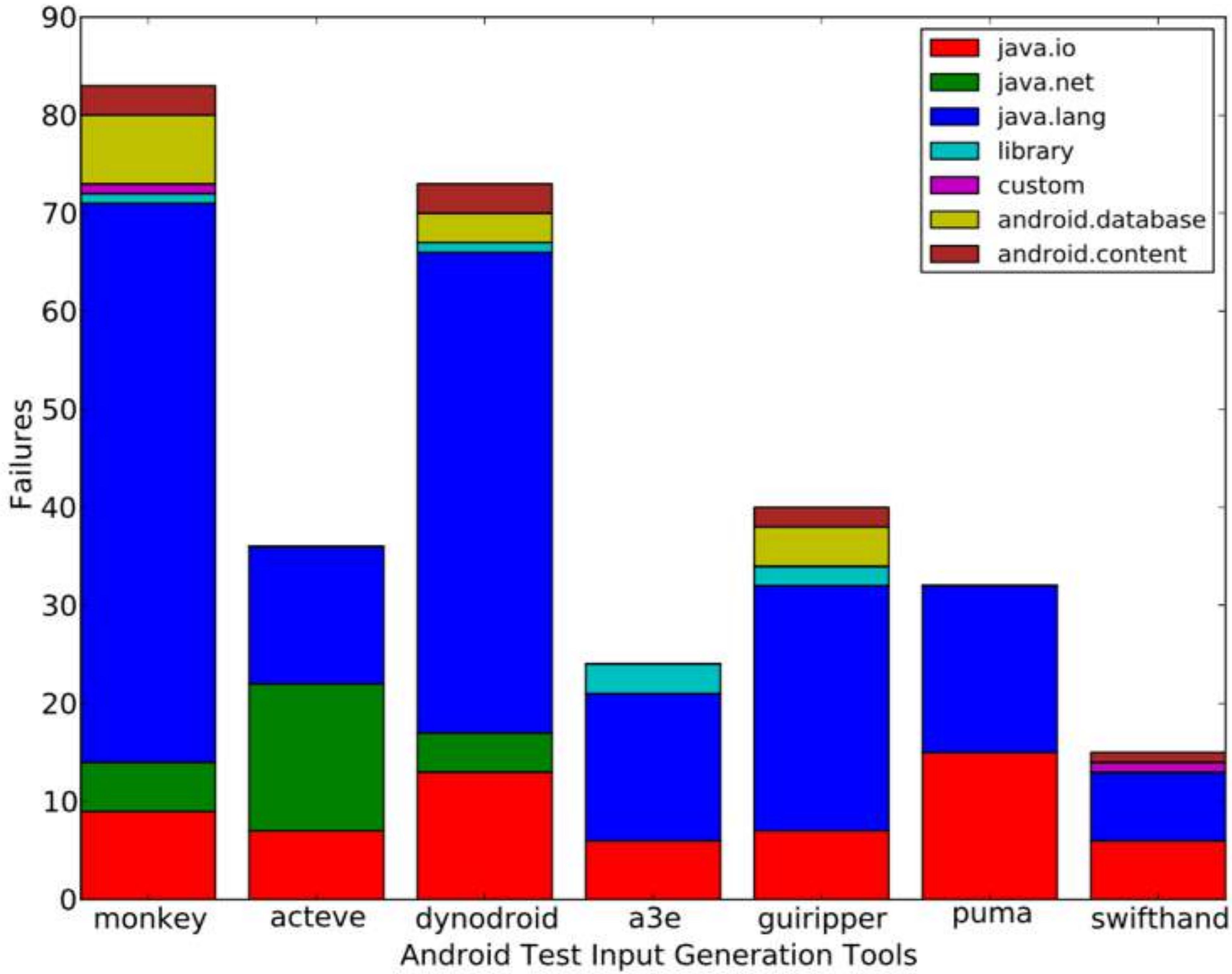
Exploration Strategy Effectiveness

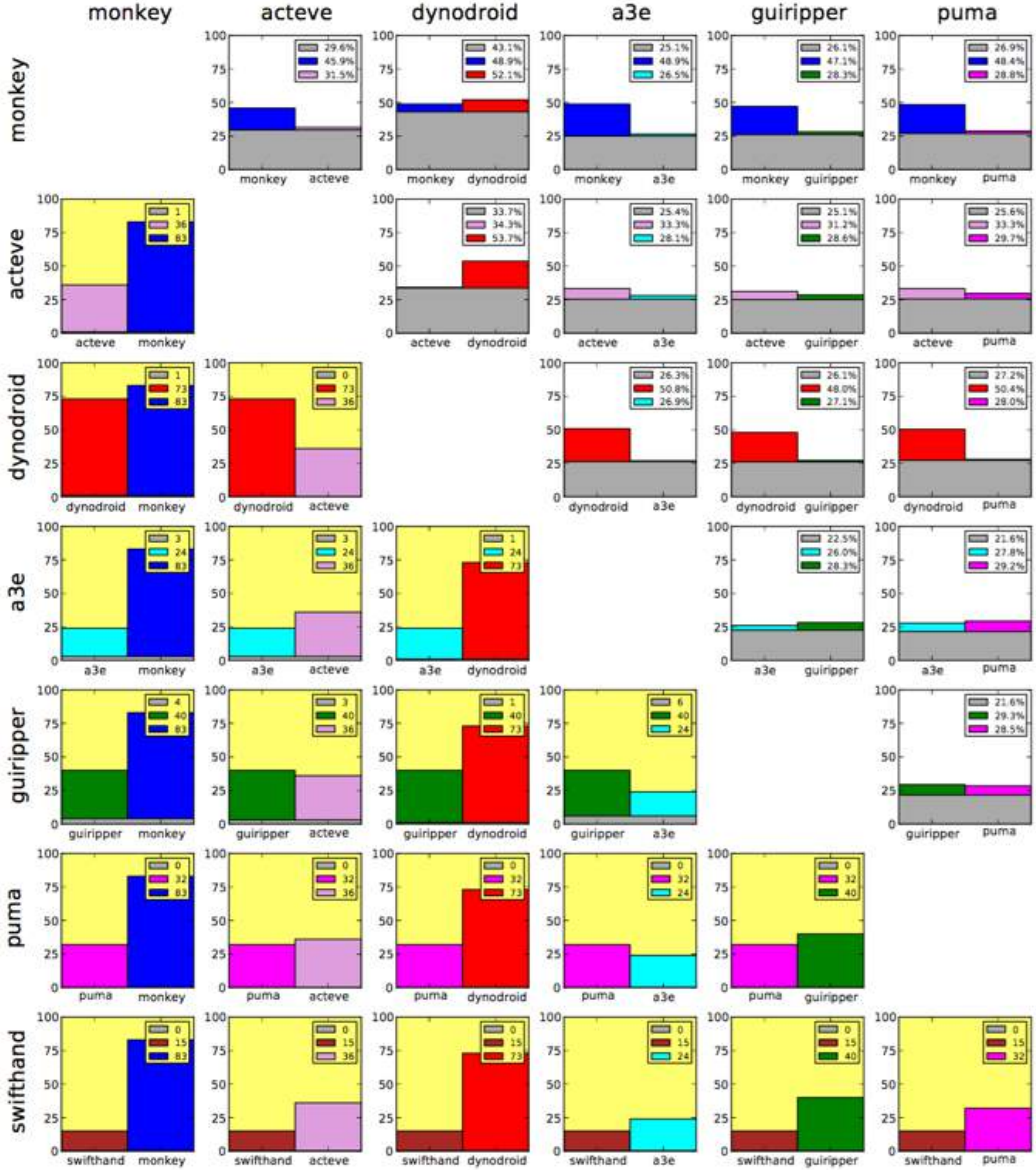


Progressive coverage



Fault detection ability



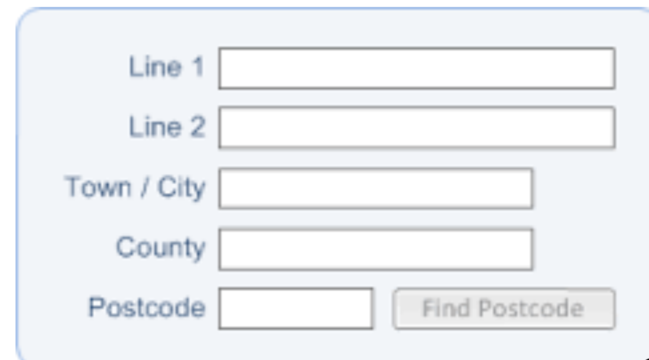

monkey
acteve
dynodroid
a3e
guiripper
puma
monkey
acteve
dynodroid
a3e
guiripper
puma
swifthand
0
25
50
75
100
29.6% 45.9% 31.5%
monkey acteve
43.1% 48.9% 52.1%
monkey dynodroid
25.1% 48.9% 26.5%
monkey a3e
26.1% 47.1% 28.3%
monkey guiripper
26.9% 48.4% 28.8%
monkey puma
1 36 83
acteve monkey
33.7% 34.3% 53.7%
acteve dynodroid
25.4% 33.3% 28.1%
acteve a3e
25.1% 31.2% 28.6%
acteve guiripper
25.6% 33.3% 29.7%
acteve puma
1 73 83
dynodroid monkey
0 73 36
dynodroid acteve
26.3% 50.8% 26.9%
dynodroid a3e
26.1% 48.0% 27.1%
dynodroid guiripper
27.2% 50.4% 28.0%
dynodroid puma
3 24 83
a3e monkey
3 24 36
a3e acteve
1 24 73
a3e dynodroid
22.5% 26.0% 28.3%
a3e guiripper
21.6% 27.8% 29.2%
a3e puma
4 40 83
guiripper monkey
3 40 36
guiripper acteve
1 40 73
guiripper dynodroid
6 40 24
guiripper a3e
21.6% 29.3% 28.5%
guiripper puma
0 32 83
puma monkey
0 32 36
puma acteve
0 32 73
puma dynodroid
0 32 24
puma a3e
0 32 40
puma guiripper
0 15 83
swifthand monkey
0 15 36
swifthand acteve
0 15 73
swifthand dynodroid
0 15 24
swifthand a3e
0 15 40
swifthand guiripper
0 15 32
swifthand puma

Challenges and Opportunities

System events!

- few tools support the generation of system events.
- which events to trigger and when?
- static analyses can be expensive, but may be useful to understand which events to trigger

Challenges and Opportunities



Manually provided inputs

- Dynodroid, GUIripper only tools that consider this
- Very basic. Can we do better?

Challenges and Opportunities

exploration strategy

- e.g. Minimize restarts
    - algorithm focused only on that is not enough. However, this is an interesting idea. Should be combined with other heuristics

Challenges and Opportunities

exploration strategy

- e.g. Multiple starting states
  - GUIRipper can support this, but it is very basic. Has to be done manually.

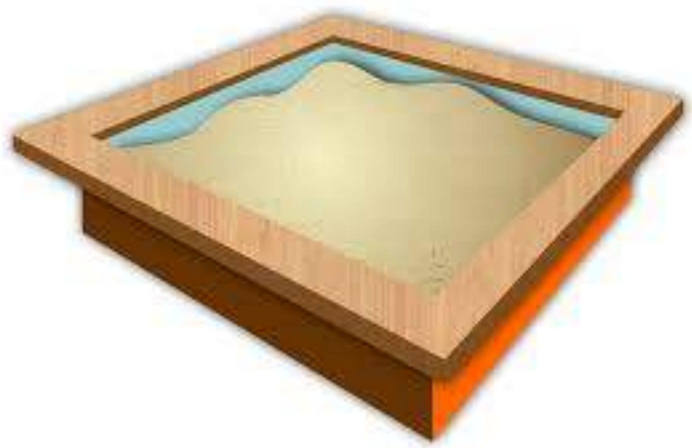
Challenges and Opportunities

avoid side effects across runs

- Dynodroid and A3E can clean state between runs (uninstalling app and clear data)

use our infrastructure!

Challenges and Opportunities



Sandboxing

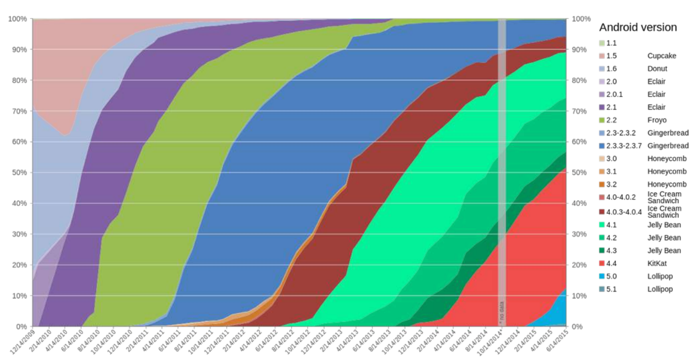
- avoid disruptive effects of some operations

Challenges and Opportunities

Reproducible test cases

- not easy to see failure reports.
- not easy to reproduce failures.
- debugging???
- NO tool is good at this.

Challenges and Opportunities



X

Fragmentation problem

- Few commercial tools are dealing with problem
- Basic solutions (lots of manual work)