

# Jordan Samhi

## *Research Scientist*

University of Luxembourg, SnT  
TruX Research Group

*Dr. Jordan Samhi*  
[jordan.samhi@uni.lu](mailto:jordan.samhi@uni.lu)  
<https://jordansamhi.com>



# Mobile Security

*Years of improvement, yet challenges remain*



# ANDROID





1<sup>1</sup>  
1%

of you have malware on your  
smartphone

2<sup>2</sup>  
90%

of you have critical vulnerabilities  
on your smartphone

1 - <https://www.networkworld.com/article/961916/malware-infection-rate-of-smartphones-is-soaring-android-devices-often-the-target.html>

2 - <https://www.linkedin.com/pulse/90-android-devices-left-exposed-critical-filipi-tassinari/>





**6 billion people own a smartphone**



**71% are Android-based**



**Sensitive Data**





# Android Security



Bank



Photos



Contacts



Social Medias



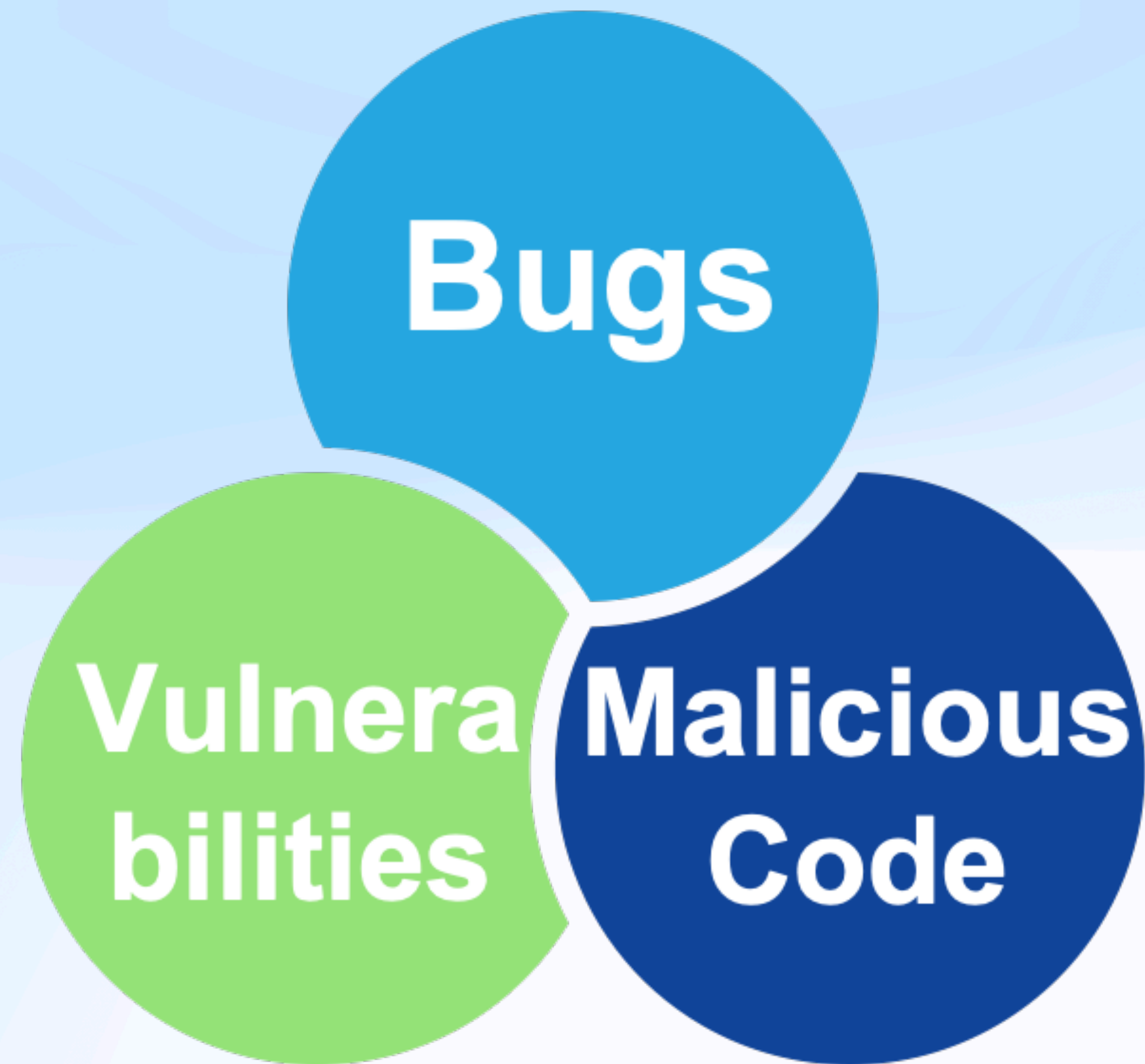
Sensors



Health Data



# High Security Risks



# \$10.5 trillion

Cost of Cybercrime in 2025

# 500 000

new malware detected in 2023



 android malware



Google Search

I'm Feeling Lucky



All

**News**

Images

Videos

Short videos

Forums

Web

More ▾

Tools ▾

Recent ▾

Sorted by relevance ▾

Advanced Search

Clear

About 12 200 results (0,22 seconds)

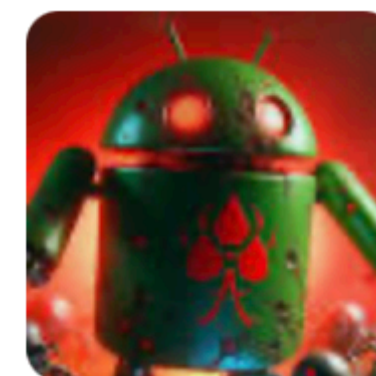


Génération NT

## Triada : un malware préinstallé sur des milliers de smartphones Android

Injectée dans le firmware de smartphones Android, une nouvelle variante du malware Triada représente une menace persistante et difficile à...

1 day ago

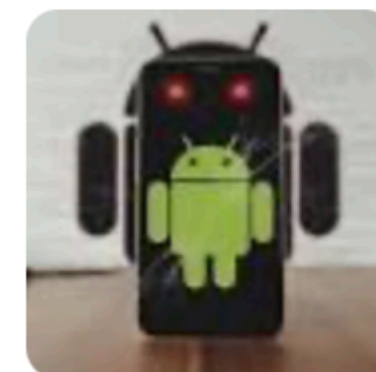


MSN

## Triada : des smartphones Android livrés avec un virus déjà installé !

Une nouvelle version du cheval de Troie Triada a été découverte préinstallée sur des smartphones Android contrefaits, selon les chercheurs...

6 days ago

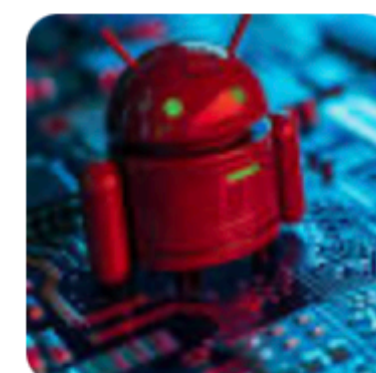


BleepingComputer

## Counterfeit Android devices found preloaded with Triada malware

A new version of the Triada trojan has been discovered preinstalled on thousands of new Android devices, allowing threat actors to steal...

1 week ago



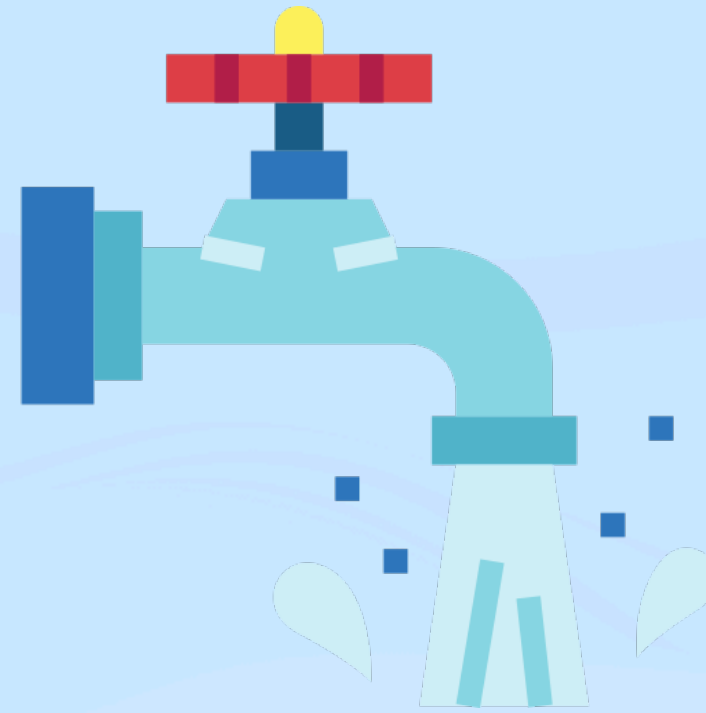


We need automated  
techniques!

# The mobile threat landscape



Malware



Data Exfiltration



Privileges



Obfuscation



Fake apps



Vulnerabilities

# Who are threat actors?



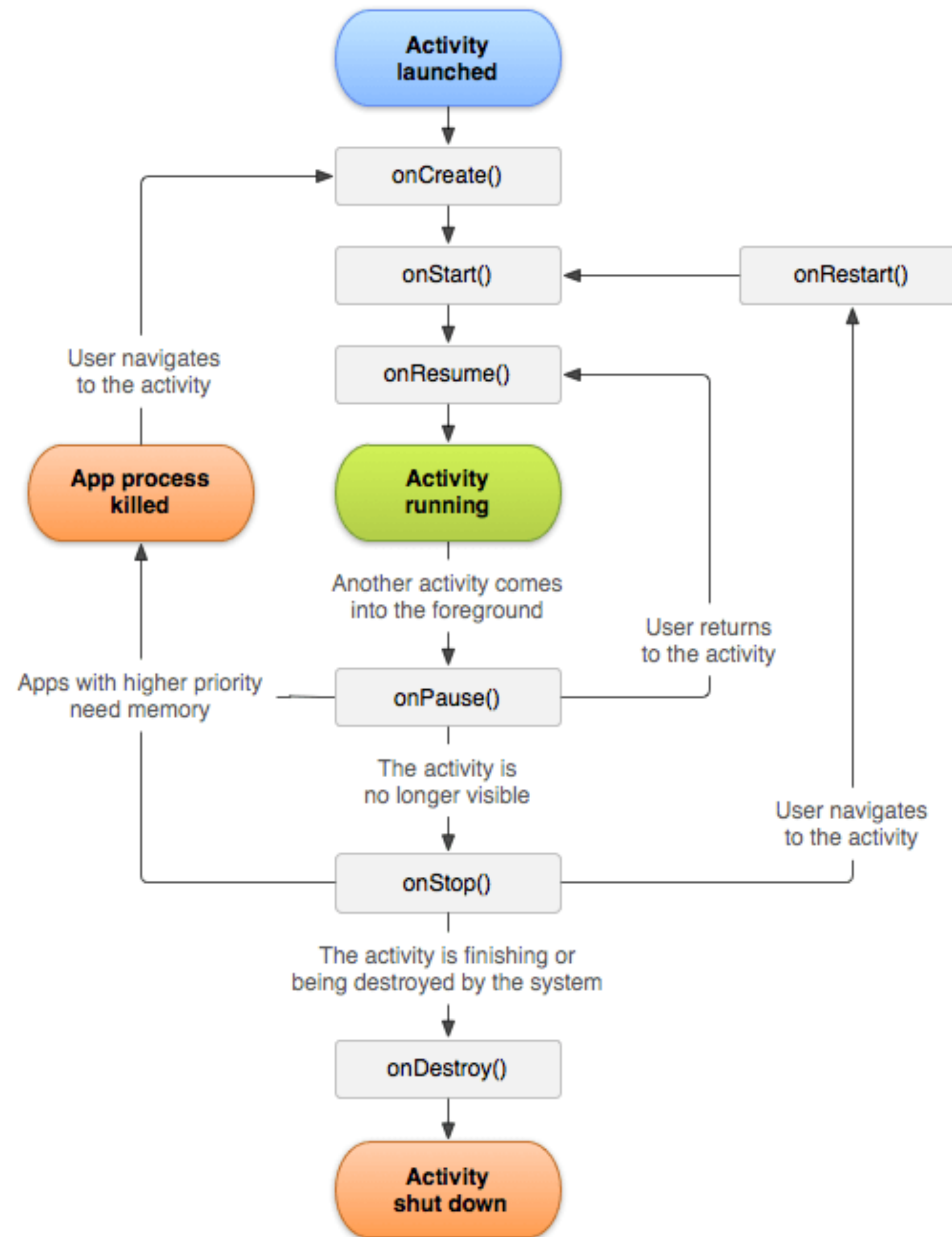


# Our Contributions

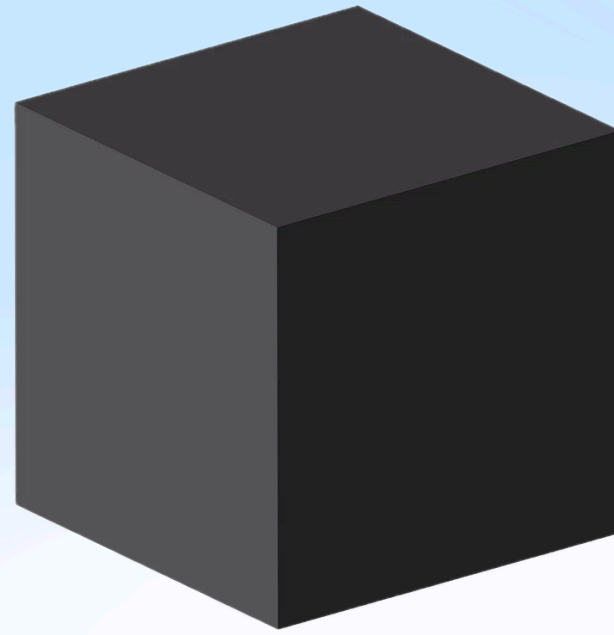
# Static Analysis



# Android

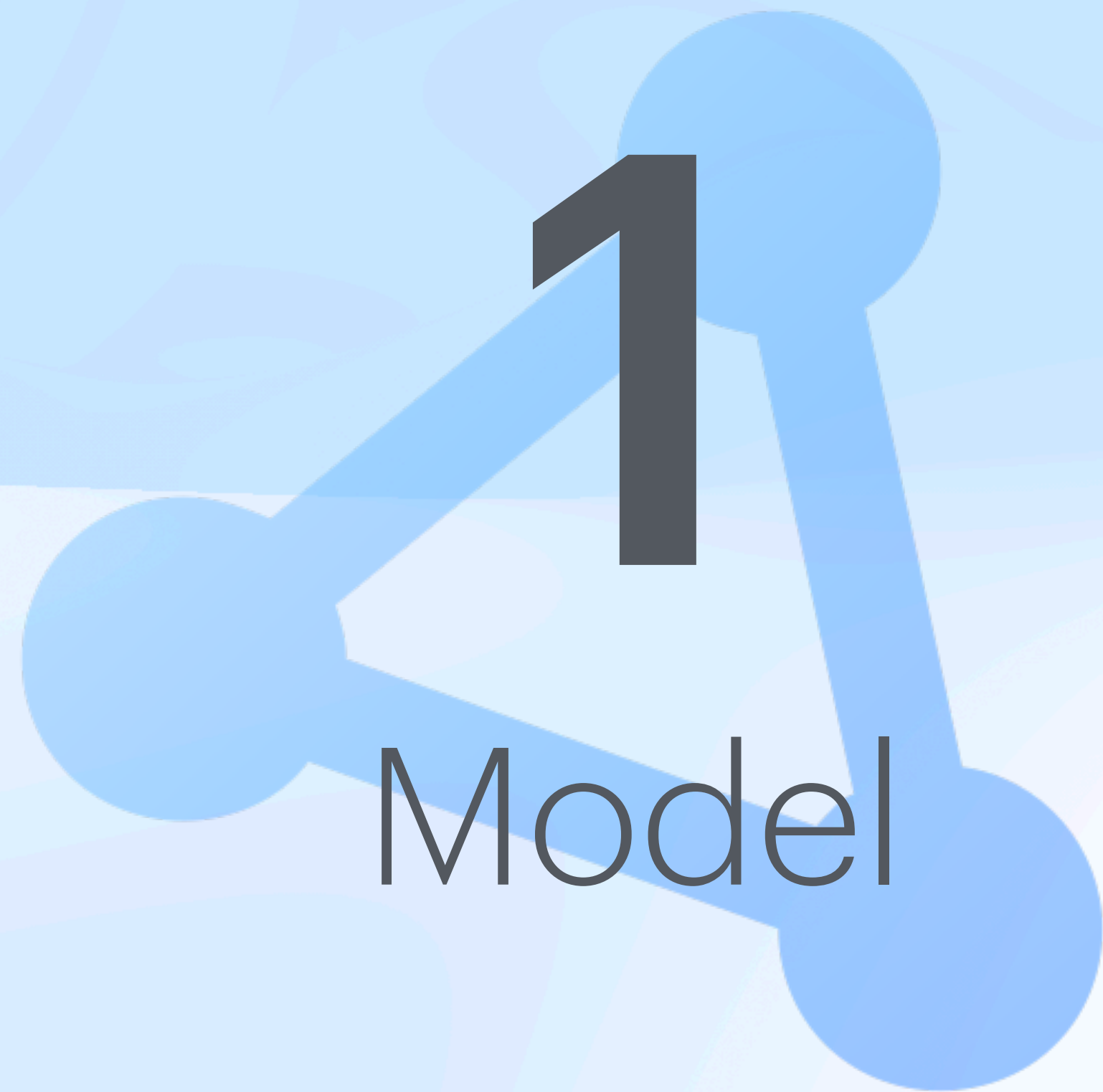


# Discontinuity

startActivity() .....  ..... onCreate()

There is a  
**discontinuity!**

# Static Analysis



# Static Analysis



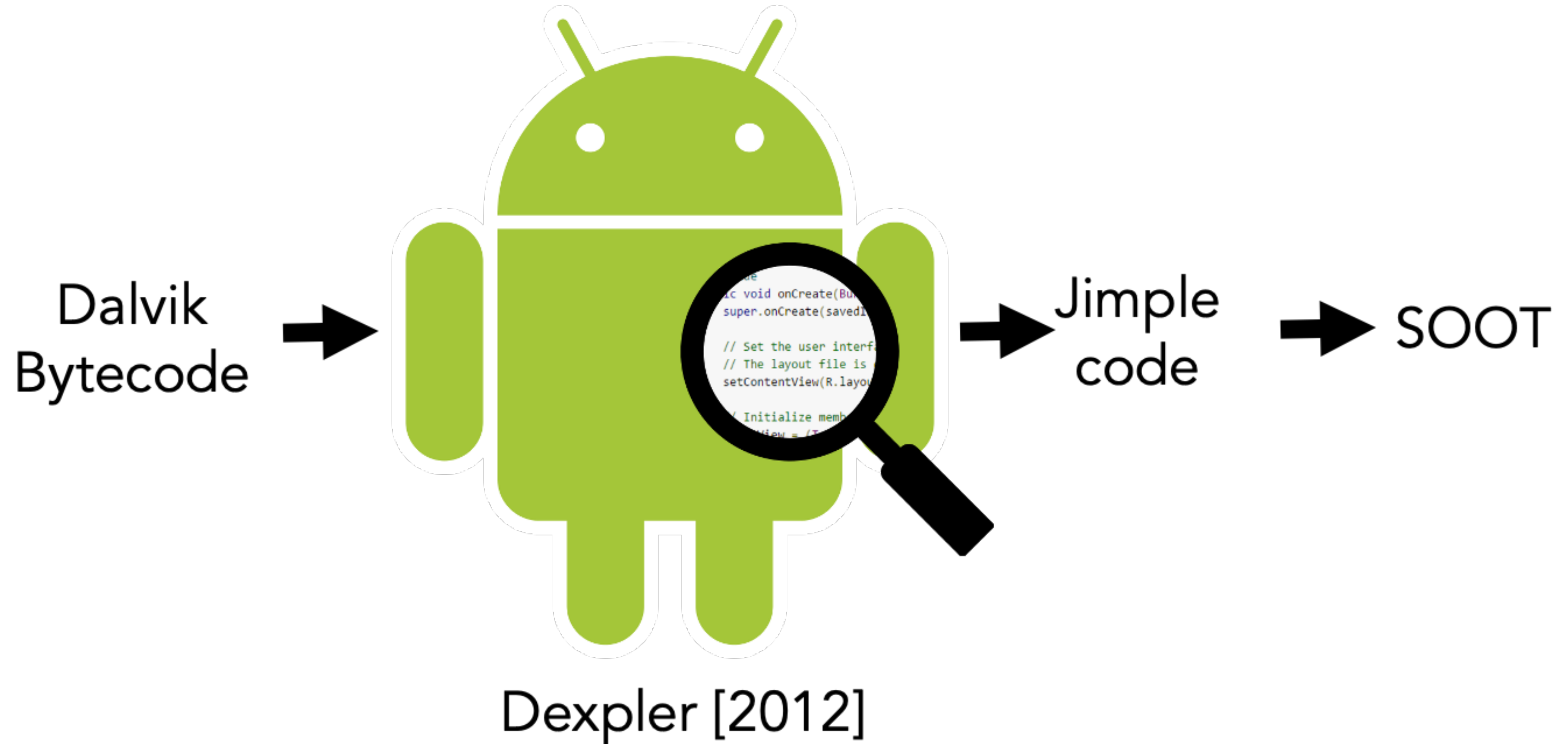
Modeling Android apps is  
challenging!





# Analyzing Android Apps (static)

Frist, need of decompiling Android App



# Data Leaks



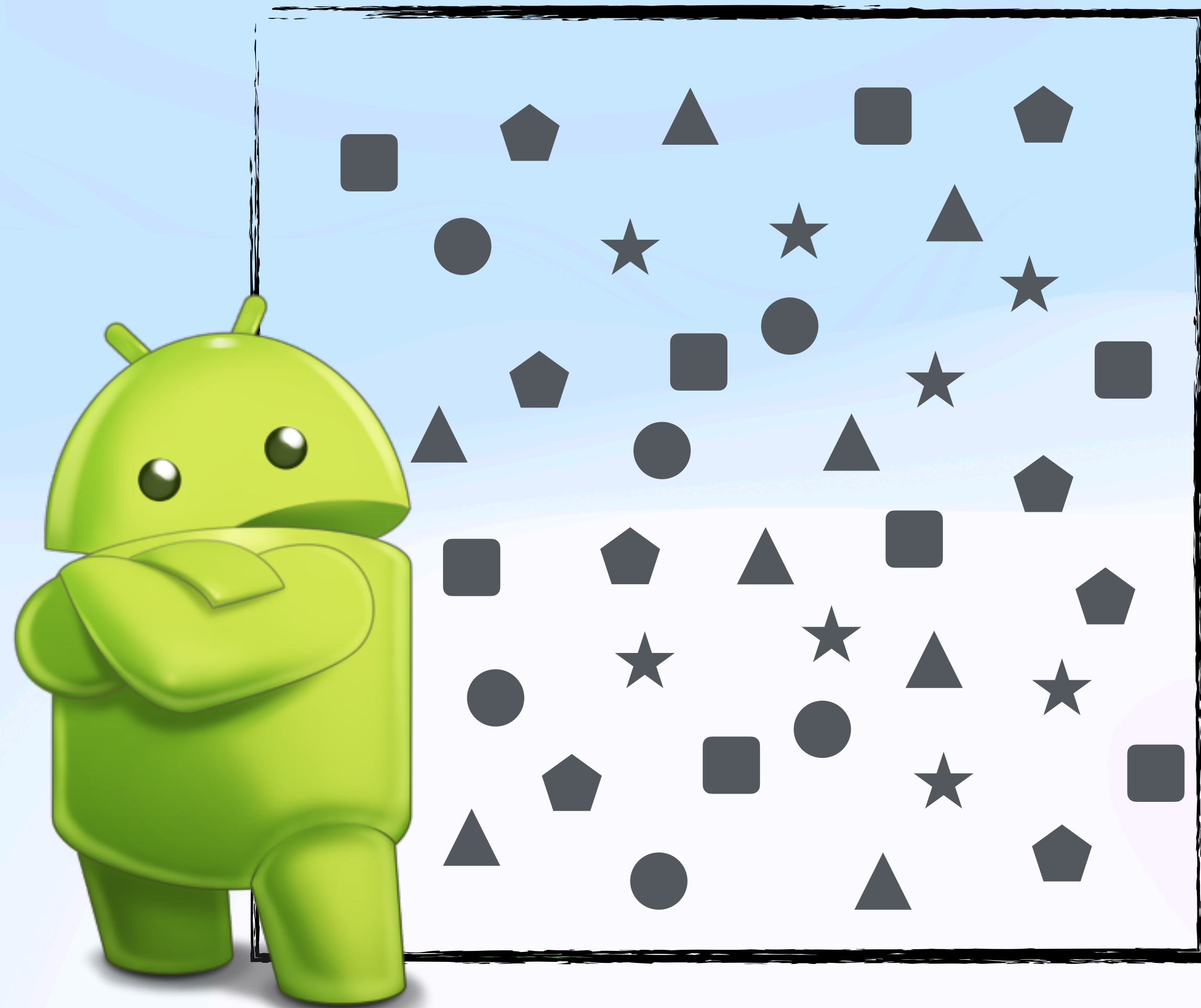
Data Leaks for  
Android Apps  
FlowDroid  
[PLDI'14]

- PLDI, 10 years Most Influential Paper
- Over 2,700 citations

So far so good,...

But in Android, do not forget  
Inter-Component  
Communication  
(ICC)





# Inter-Component Communication

startComponent1()

startComponent2()

...

# Example of Leak between Components

```
public class Activity_A extends ActionBarActivity {  
  
    TelephonyManager telManager;  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_activity_);  
  
        String id = telManager.getDeviceId();  
        Intent intent = new Intent(Activity_A.this, Activity_B.class);  
        intent.putExtra("sensitive", id);  
        Activity_A.this.startActivity(intent);  
    }  
}
```

## Difficulty: ICC

Inter Component Communication

```
public class Activity_B extends ActionBarActivity {  
  
    SmsManager sms;  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_activity__b);  
  
        Intent i = getIntent();  
        String s = i.getStringExtra("sensitive");  
        String number = "+3524666445600";  
        sms.sendTextMessage(number, null, s, null, null);  
    }  
}
```



# Example of Leak between Components

```
public class Activity_A extends ActionBarActivity {
```

```
    TelephonyManager telManager;
```

```
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_activity_);
```

```
        String id = telManager.getDeviceId();  
        Intent intent = new Intent(Activity_A.this, Activity_B.class);  
        intent.putExtra("sensitive", id);  
        Activity_A.this.startActivity(intent);
```

```
    }  
    Activity_B ab = new Activity_B();  
    ab.onCreate(...)
```

source

```
public class Activity_B extends ActionBarActivity {
```

```
    SmsManager sms;
```

```
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_activity__b);
```

```
        Intent i = getIntent();  
        String s = i.getStringExtra("sensitive");  
        String number="+3524666445600";  
        sms.sendTextMessage(number, null, s, null, null);
```

sink

## Difficulty: ICC

Inter Component Communication

# Inter-Component Communication

setAlarm()

sendSMS()

...

# Our Approach

**RAICC**

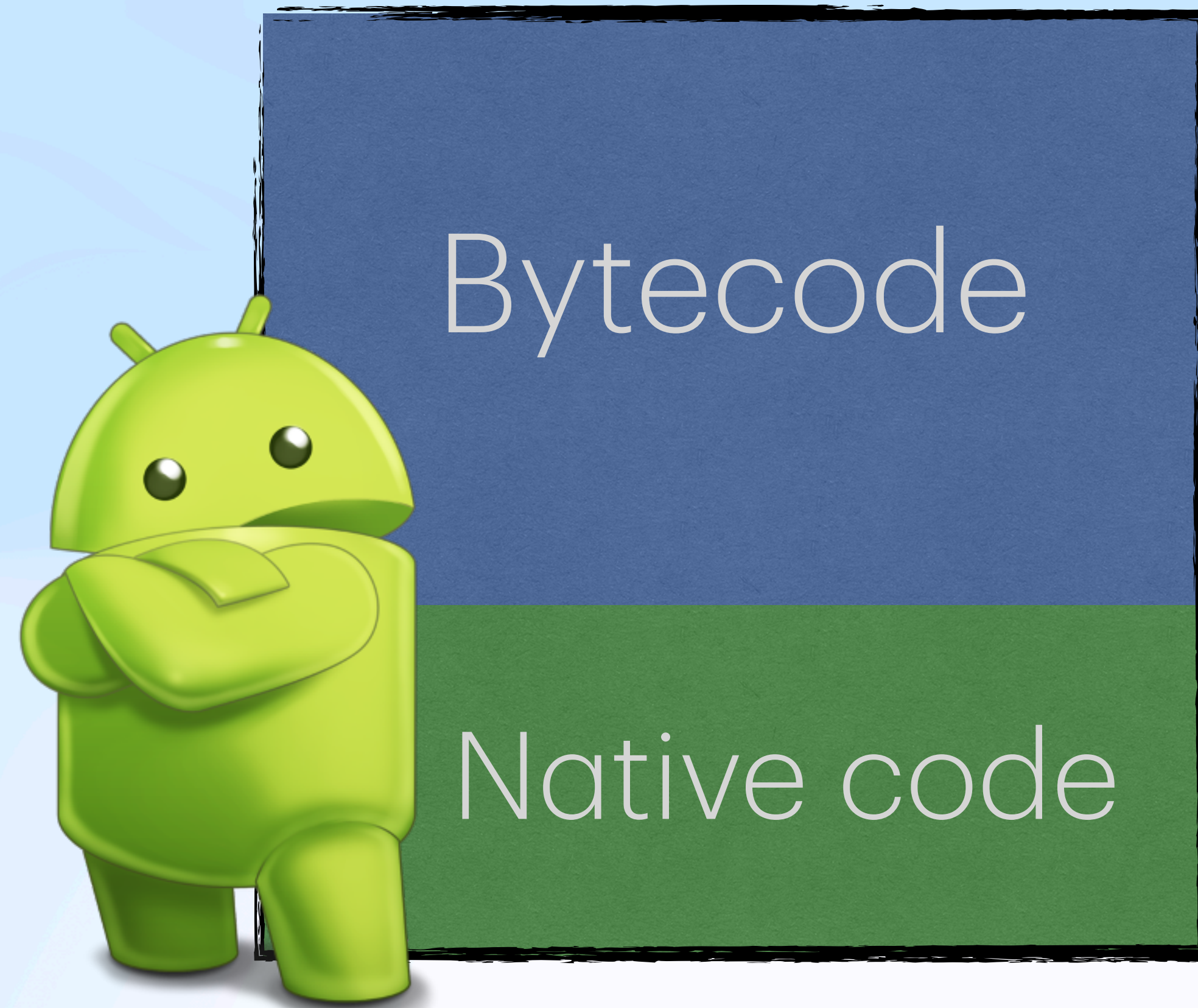


<https://github.com/JordanSamhi/RAICC>

***ICSE 2021 - Research Track***





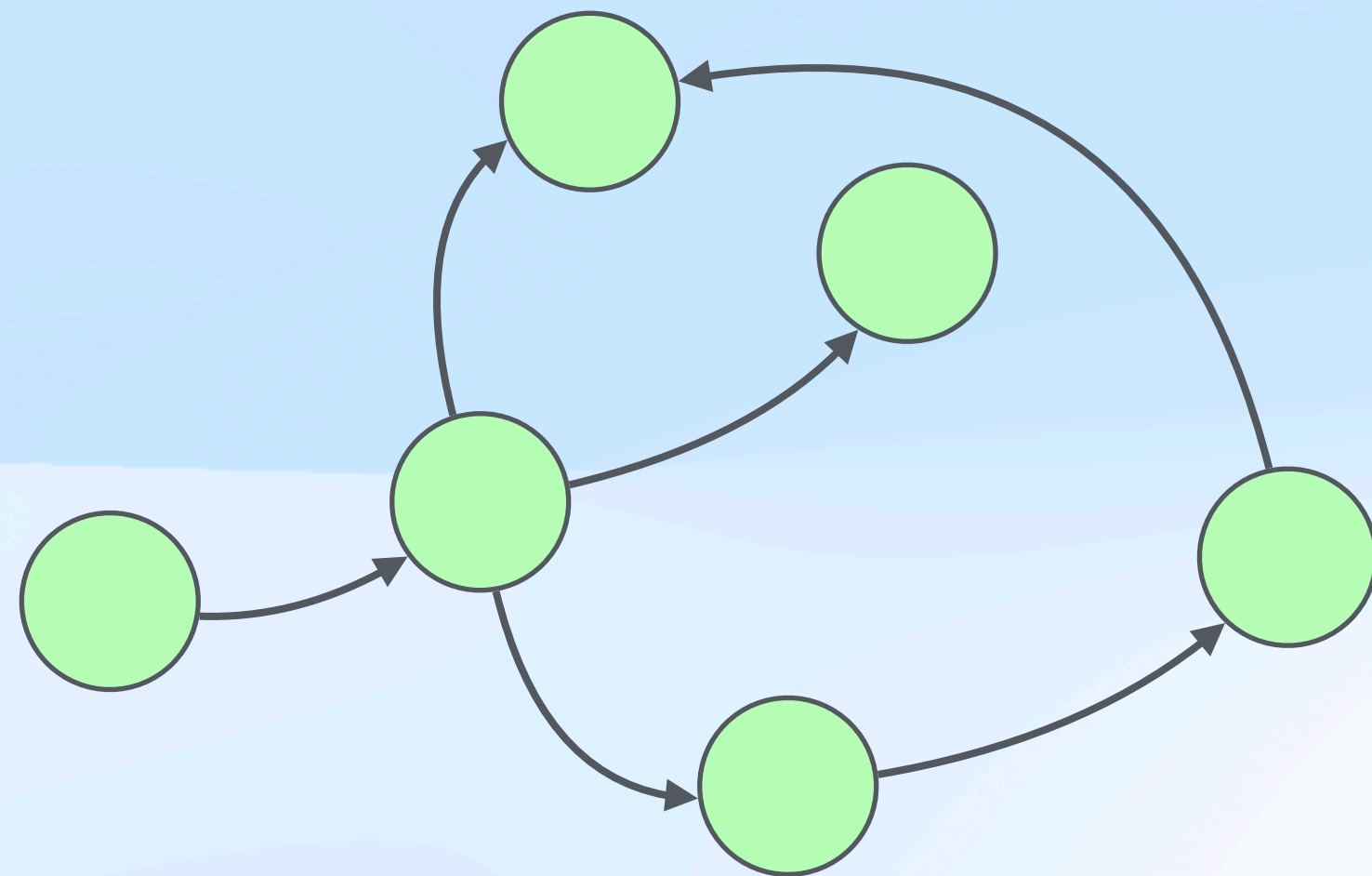


**Easy** to  
Analyze

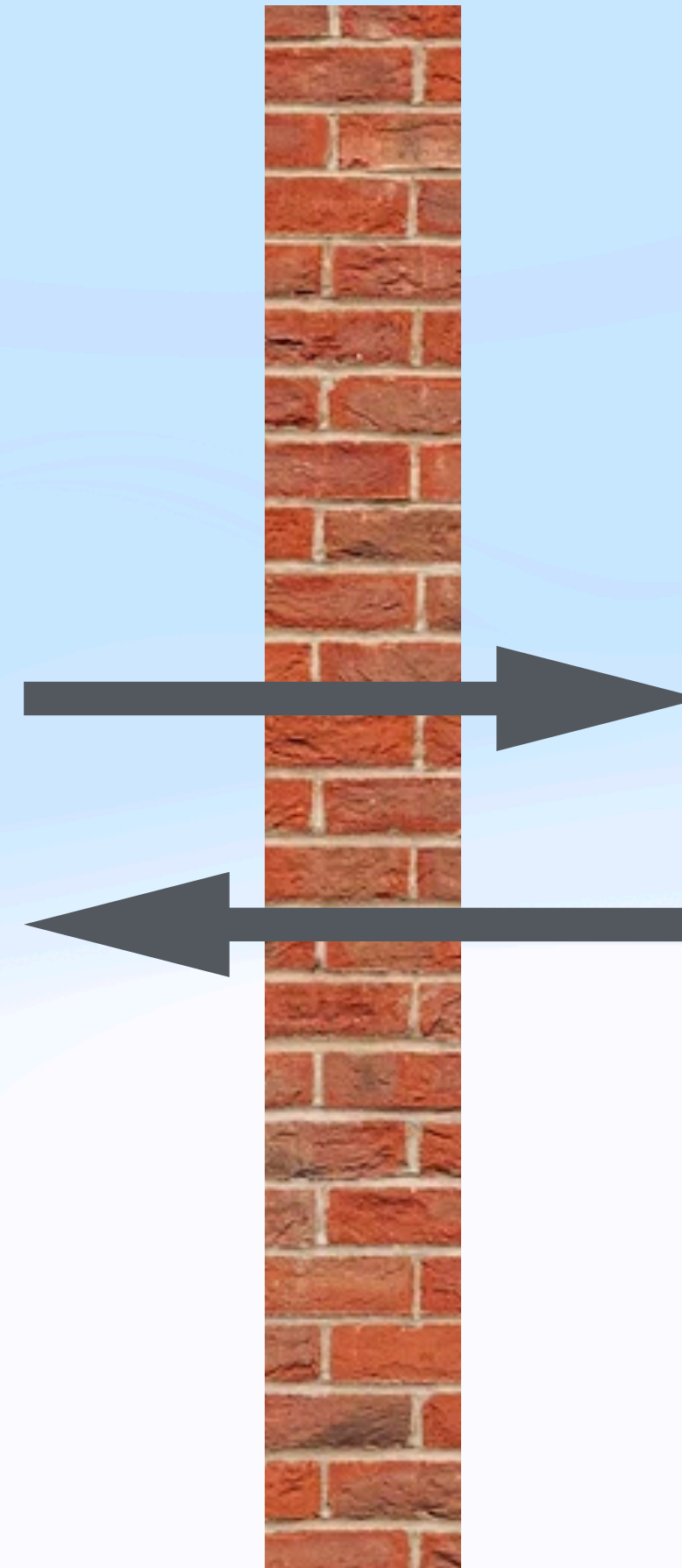
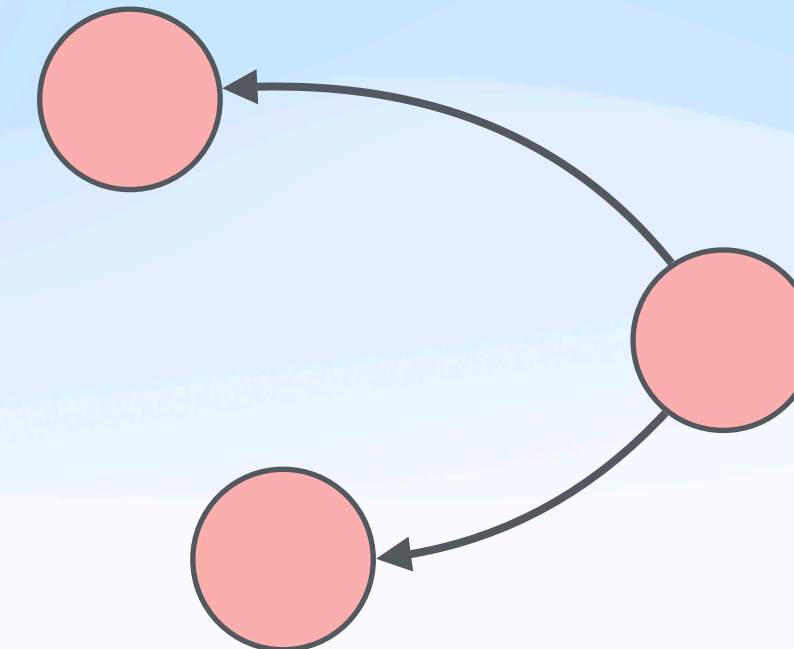
**Hard** to  
Analyze

# Model Construction

Bytecode



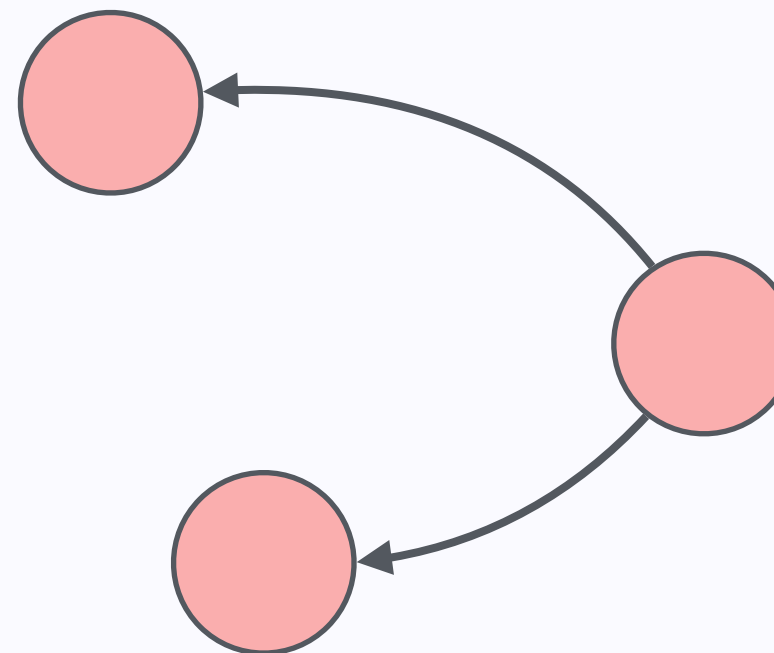
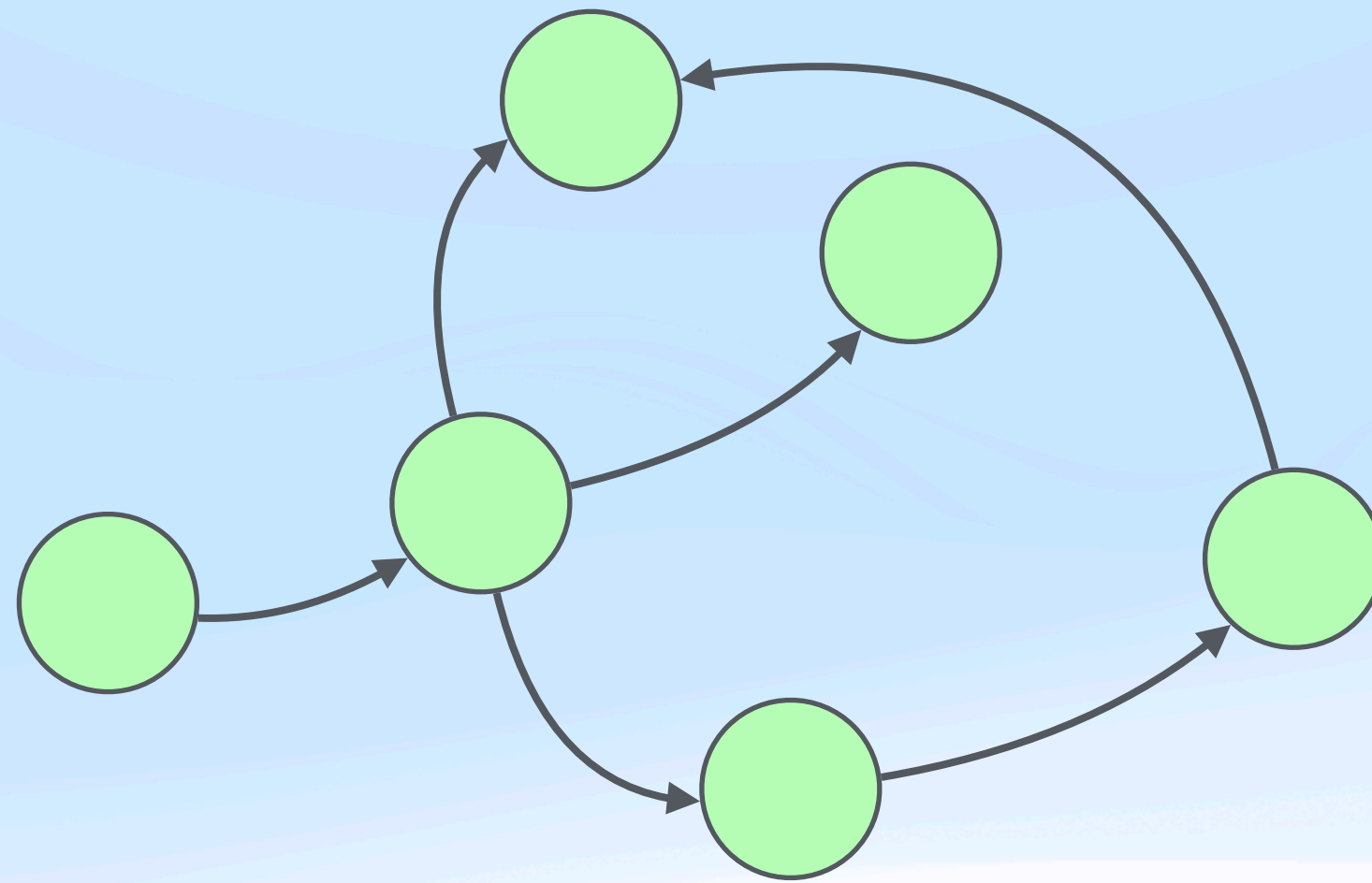
Native code



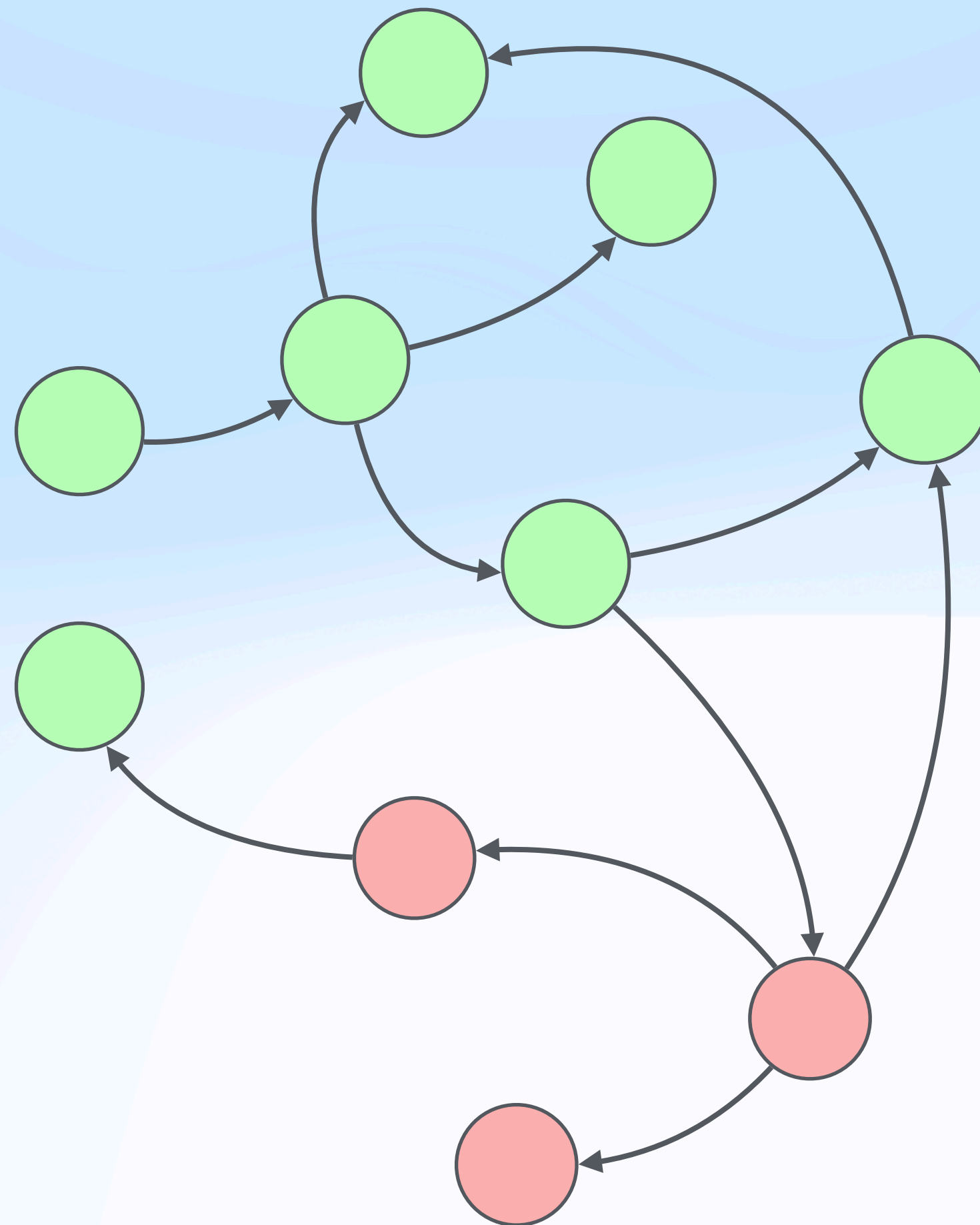
Results are bridged



# Model Construction



# Model Construction



**A Unified  
Model**

# Our Approach

JuCIFy



<https://github.com/JordanSamhi/JuCify>

**ICSE 2022 - Research Track**

# Implications for Security

**Better Static Code Modeling**

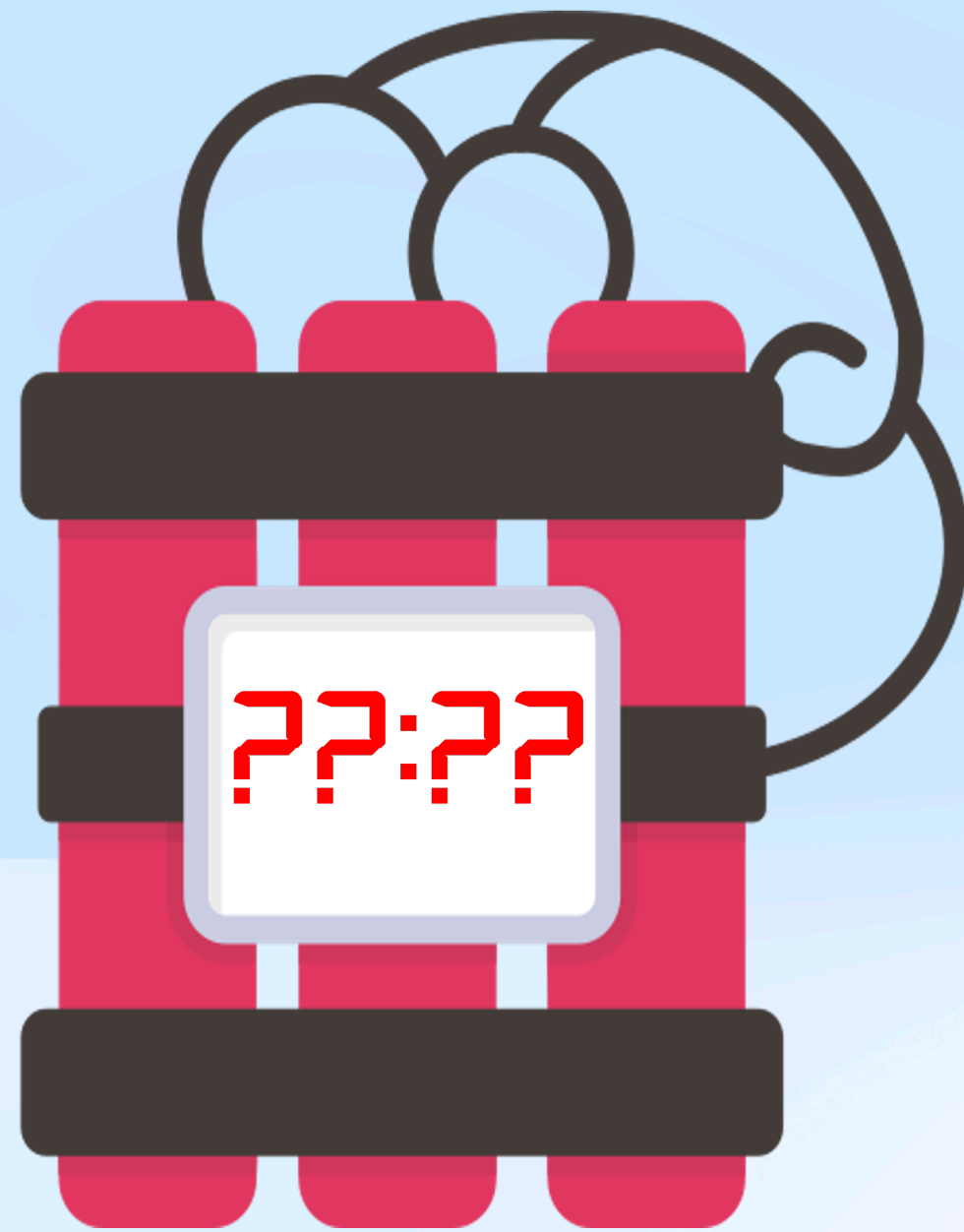


**Better Code Coverage**



# Dynamic Analysis

# Technical Challenges



Malicious code may  
be dormant



Malicious code may  
be hidden

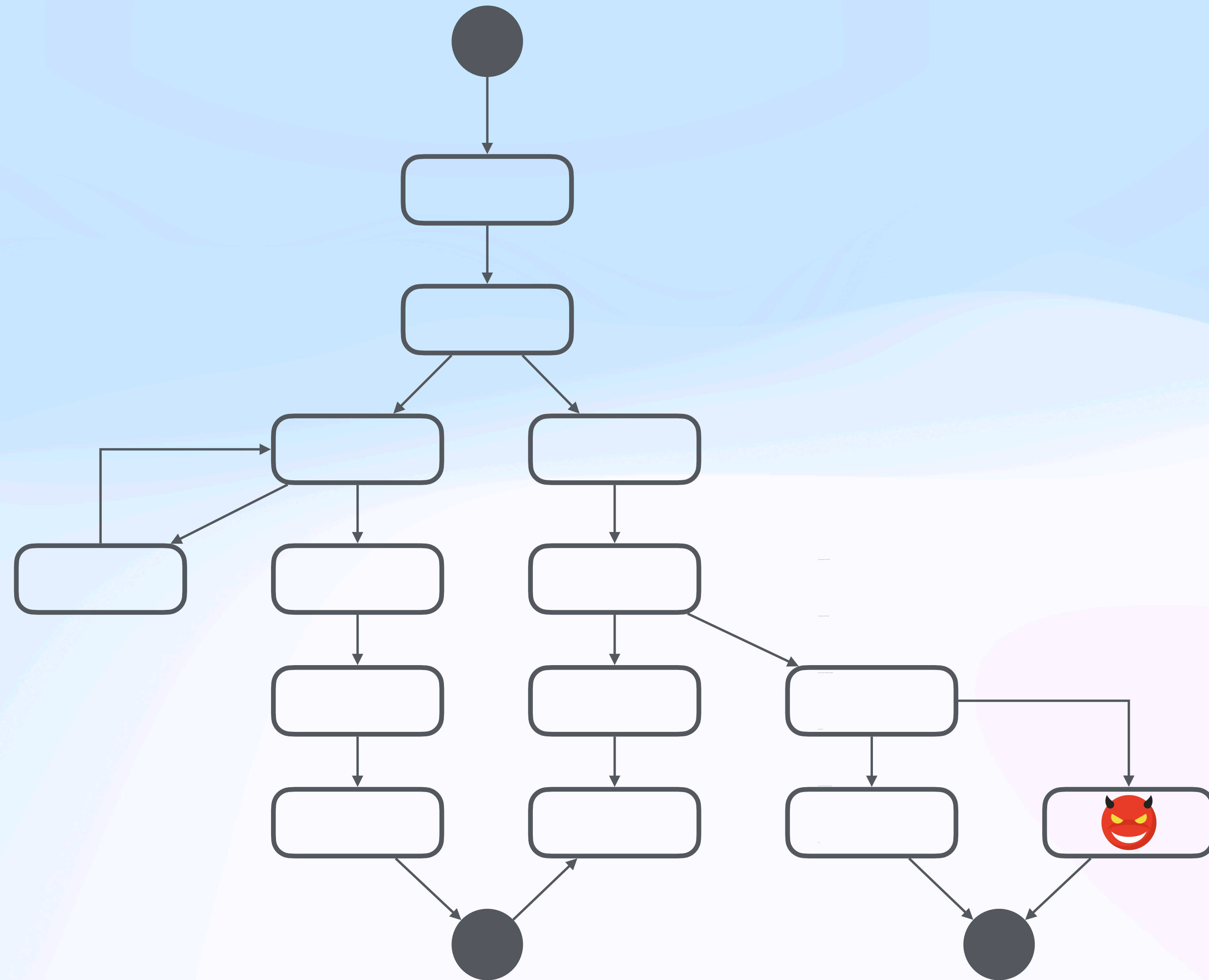
# Logic Bomb Detection

1. A **reproduction** of a static analysis approach
2. A **new approach** based on data flow analysis and anomaly detection
3. A **new dataset** of Android apps automatically infected with logic bombs



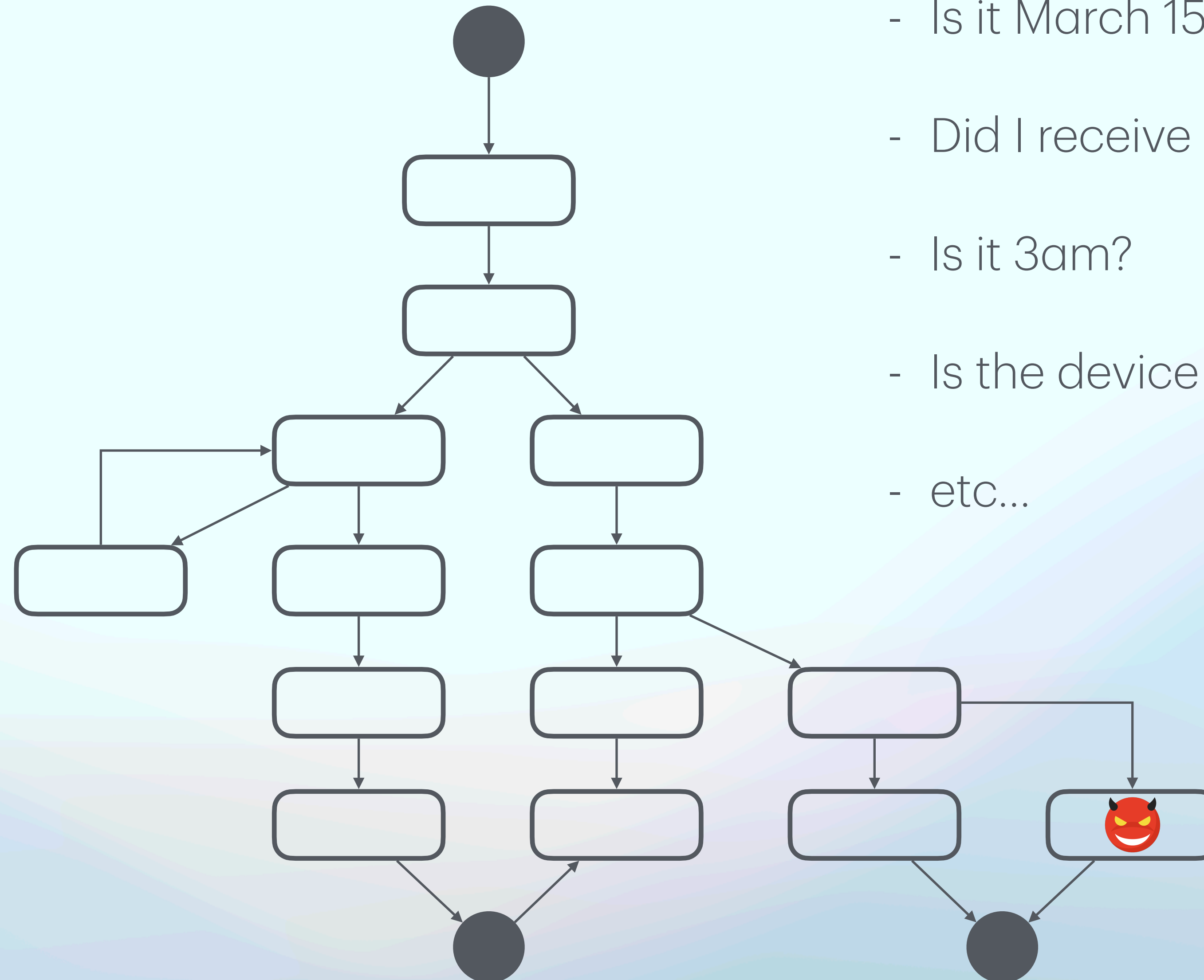


# What is a logic bomb?





# What is a logic bomb?

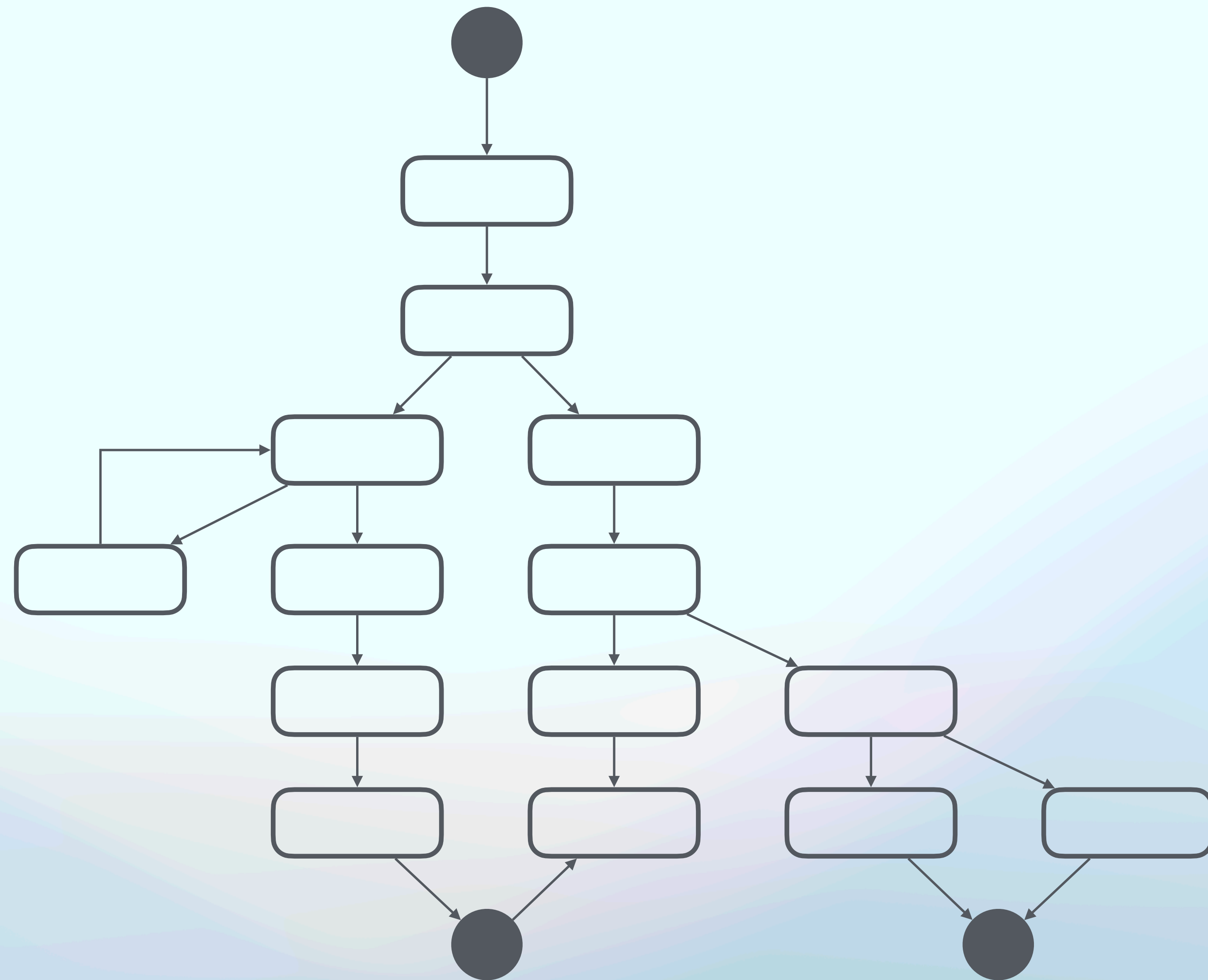


- Is the device in China?
- Is it March 15<sup>th</sup> 2024?
- Did I receive a specific SMS?
- Is it 3am?
- Is the device an emulator?
- etc...

# How to detect logic bombs?

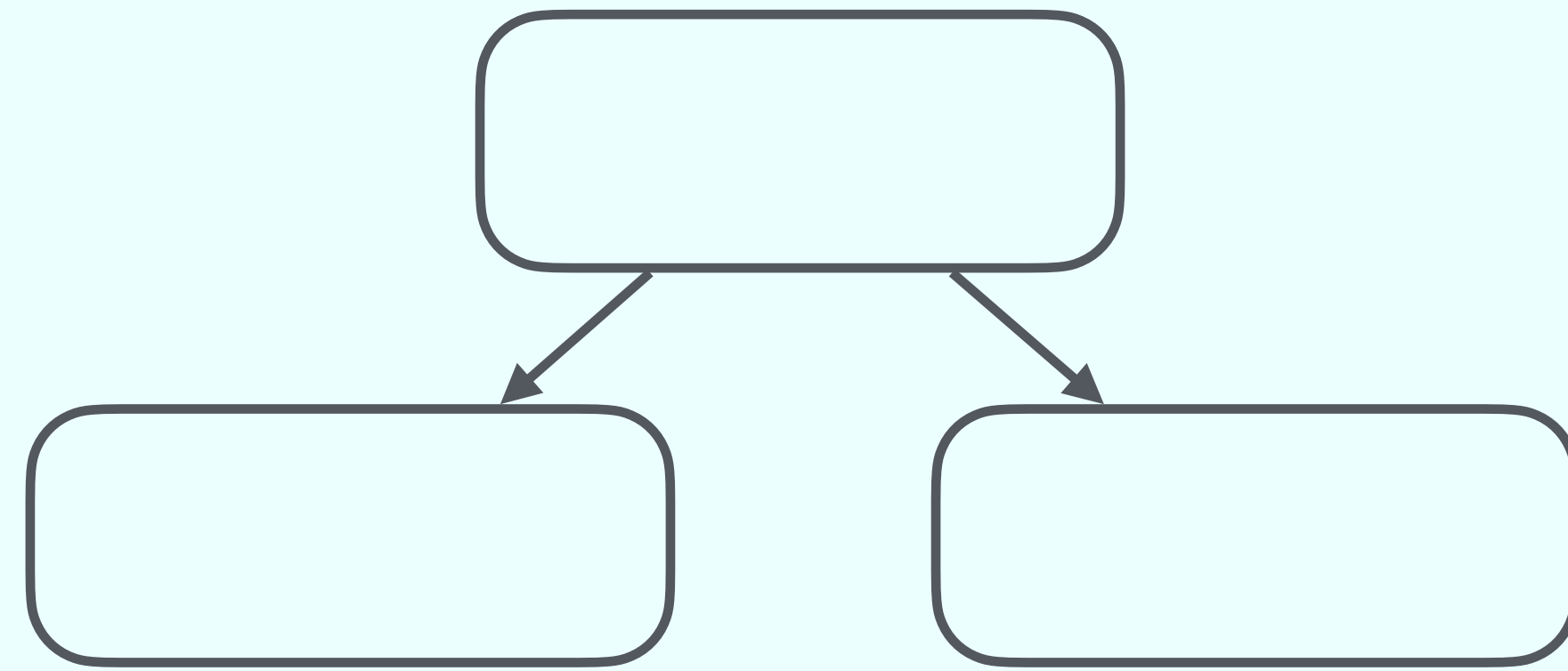
- Too many conditional statements
- How to discriminate legitimate conditions from malicious ones?

# Training

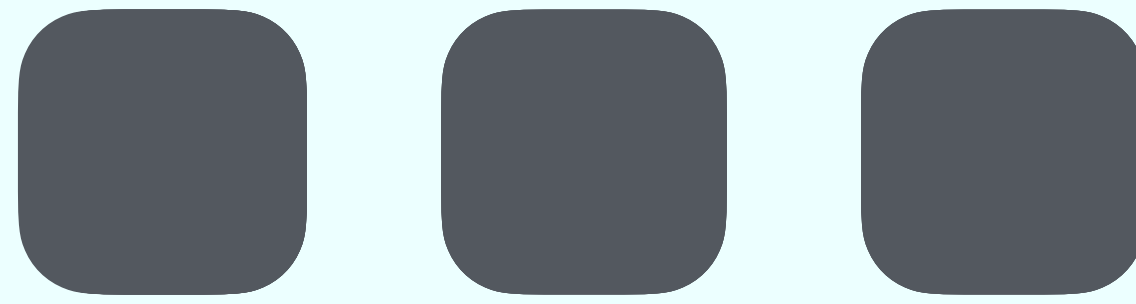




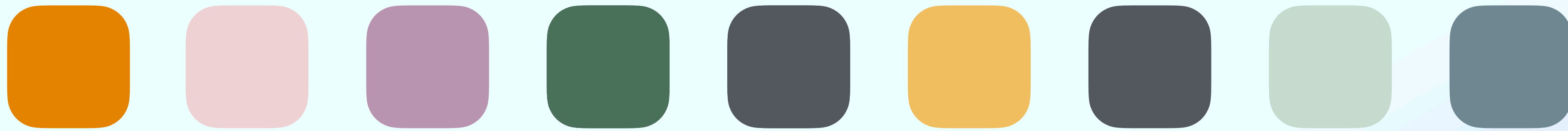
# Training



# Training

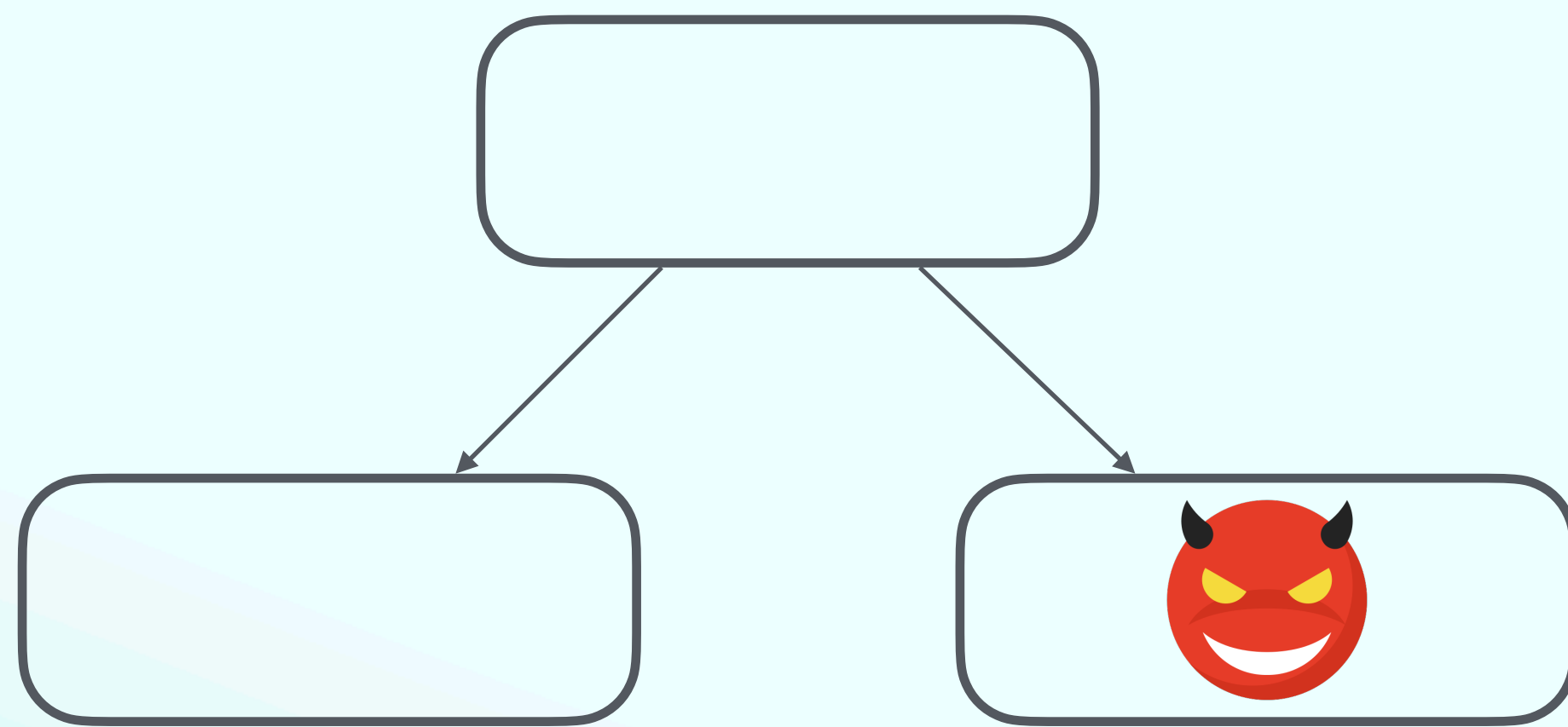


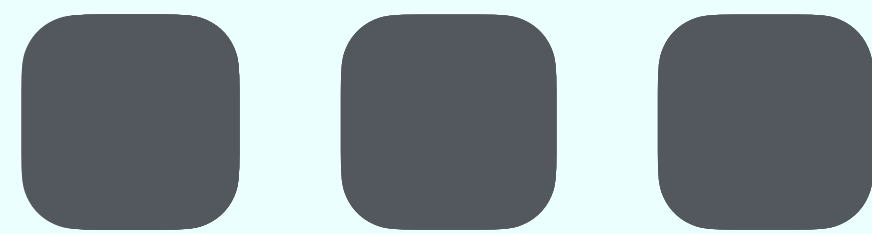
# Training





Training









# What is a logic bomb?

**NOI**

## DIFUZER

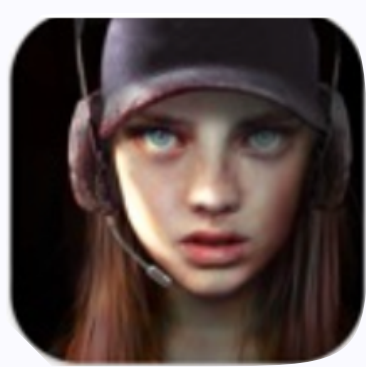


<https://github.com/JordanSamhi/Difuzer>

**ICSE 2022 - Research Track**

**ABN**





Difuzer





# A Dataset



# A Dataset





# TRIGGERZOO



<https://github.com/JordanSamhi/TriggerZoo>

**MSR 2022 - Dataset Track**


# Instrumentation



How to find  
properties?



# Dynamic Analysis



How do you know  
it is good?

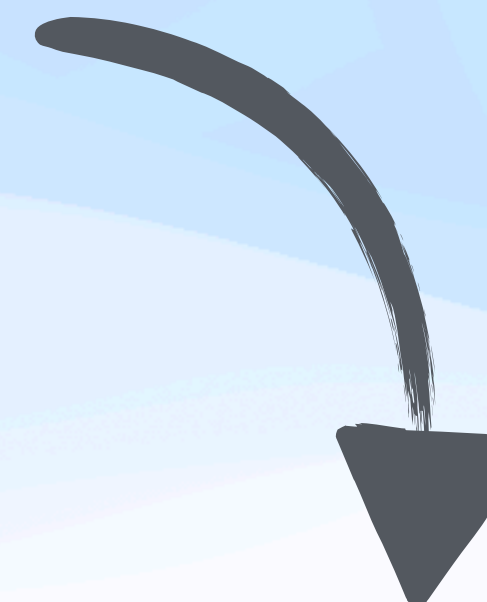
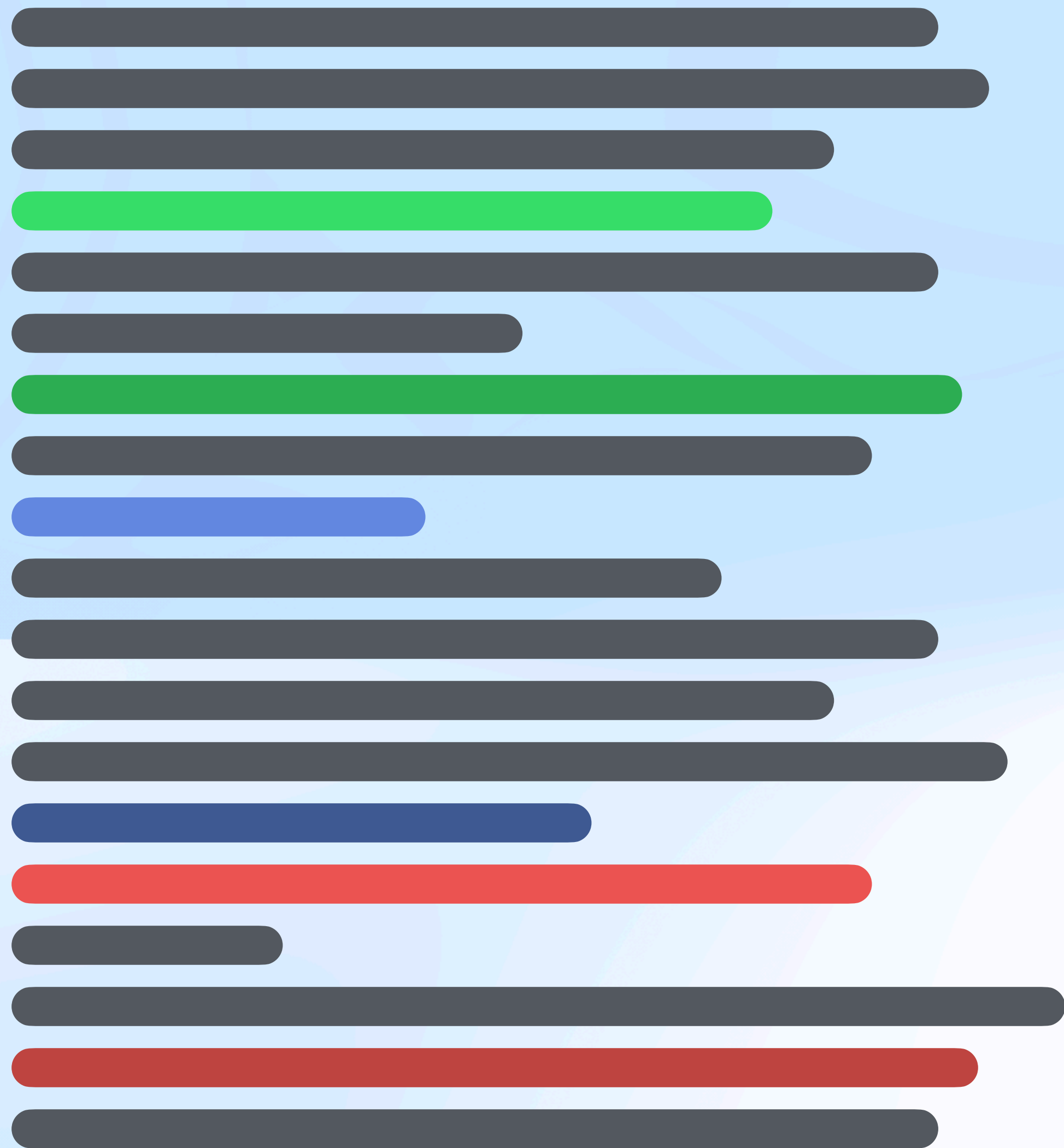
# CODE COVERAGE



[Redacted text block containing multiple lines of obscured content]

But, how do you  
do with  
Android?



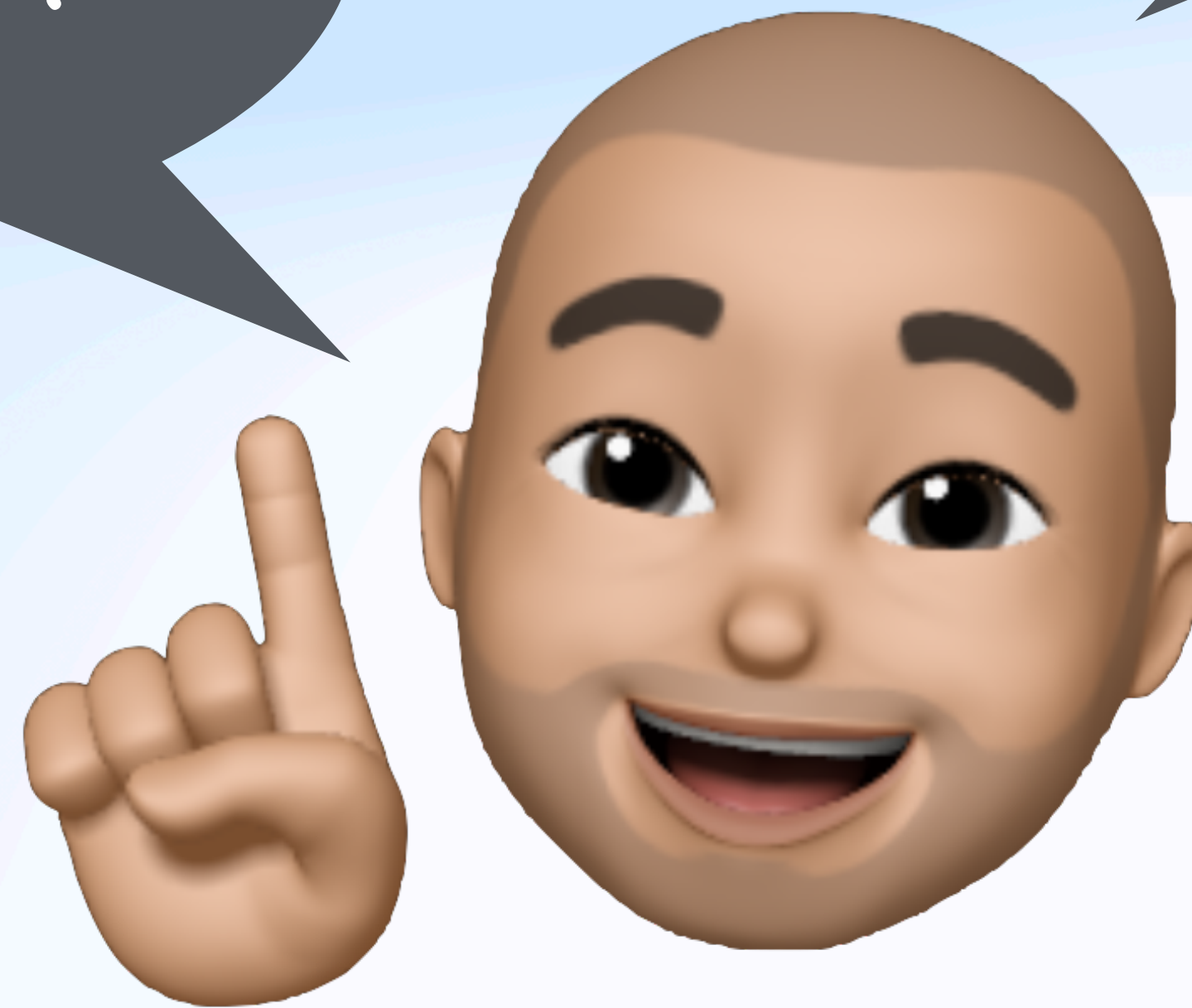


# REPORT

Good idea !



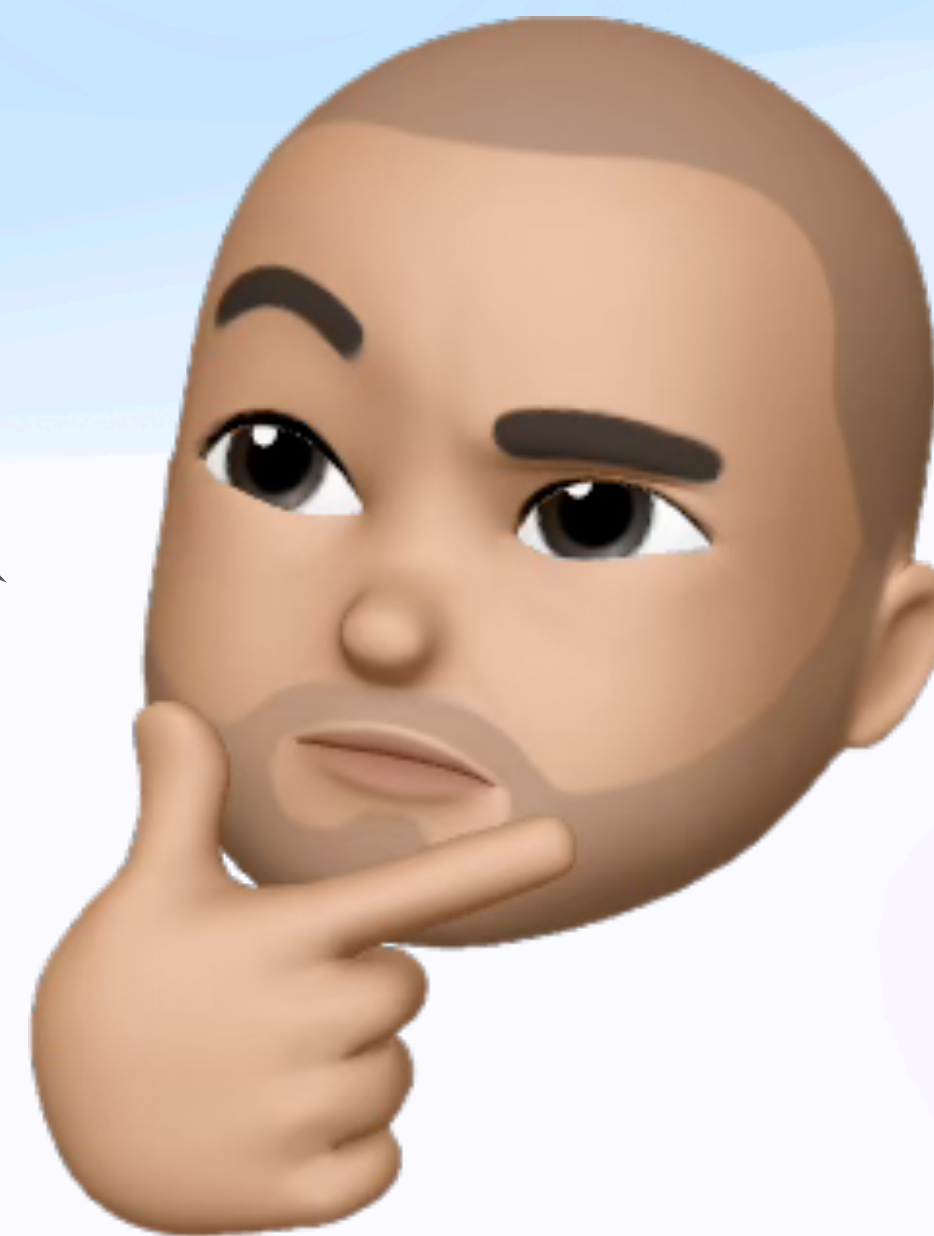
But what if you do not have access to source code?



# BLACK BOX PROBE INSERTION



*Any existing tools?*







YES!

YES

YES



YES!

YES



YES

YES!

YES



YES

YES



✓ YES

YES!

YES

YES!

YES!



✓ YES

YES



YES

YES

YES



YES



- They are invasive
- They write on SD card
- They are hard to use
- They modify the app Manifest
- They add permissions
- They can alter behavior
- etc.



# ANDROLOG



- A novel framework
- It allows black box instrumentation
- It is designed to be simple to use and flexible
- No new permission
- No invasive behavior
- No new resource
- Just log statements

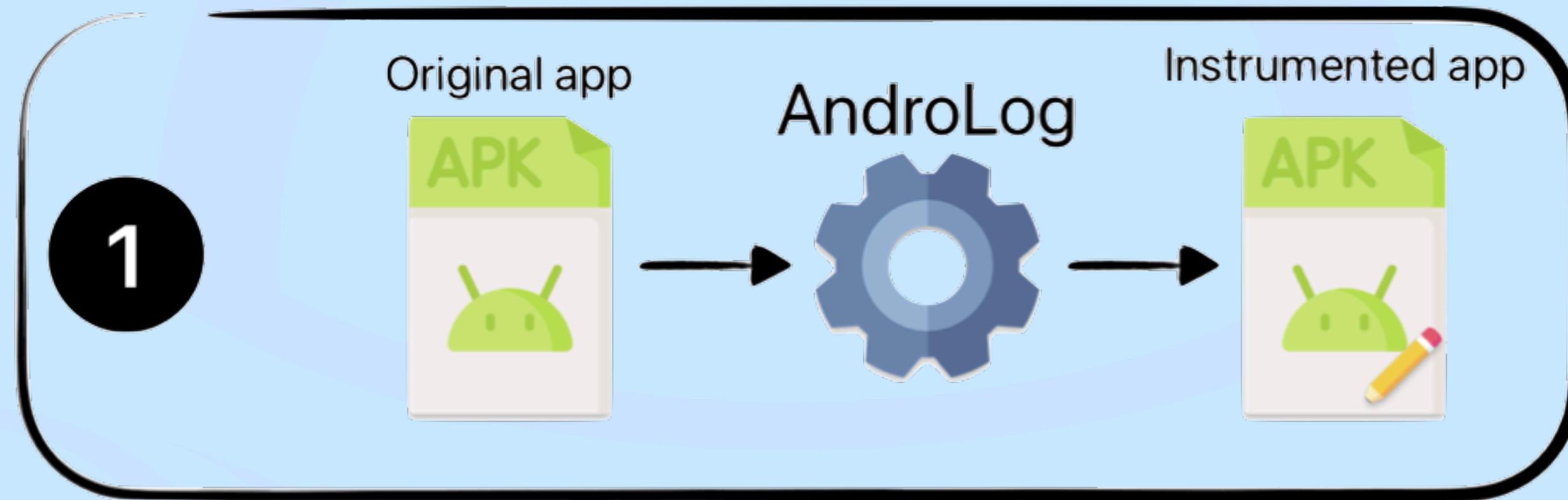
```
public void run() {  
    [...]  
}
```

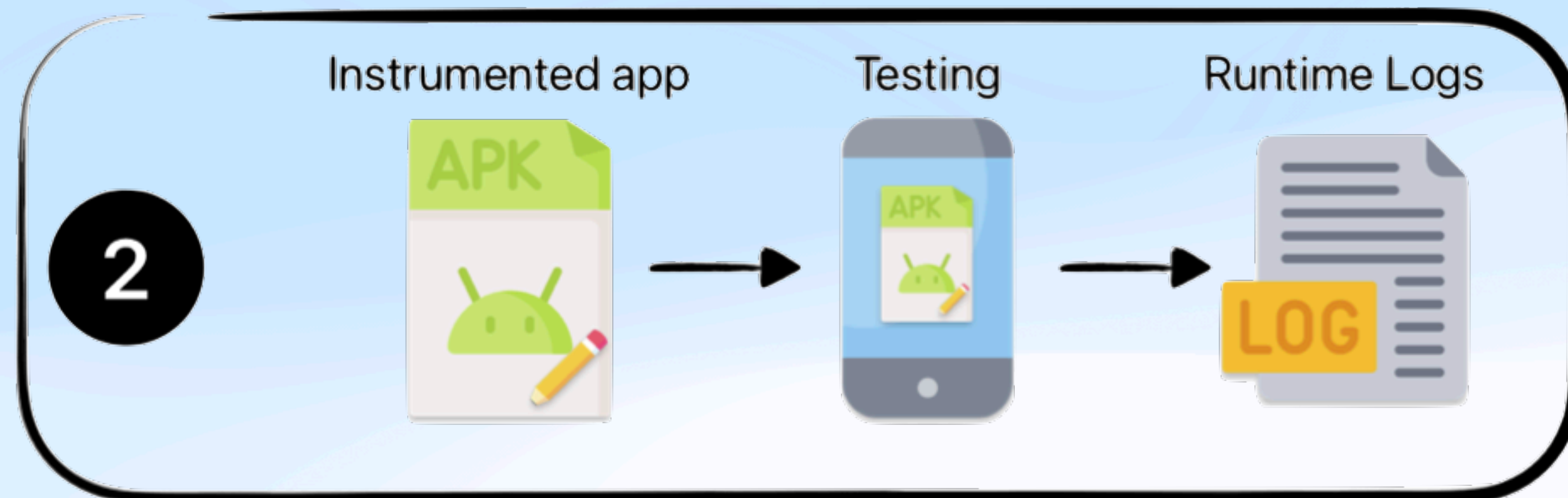
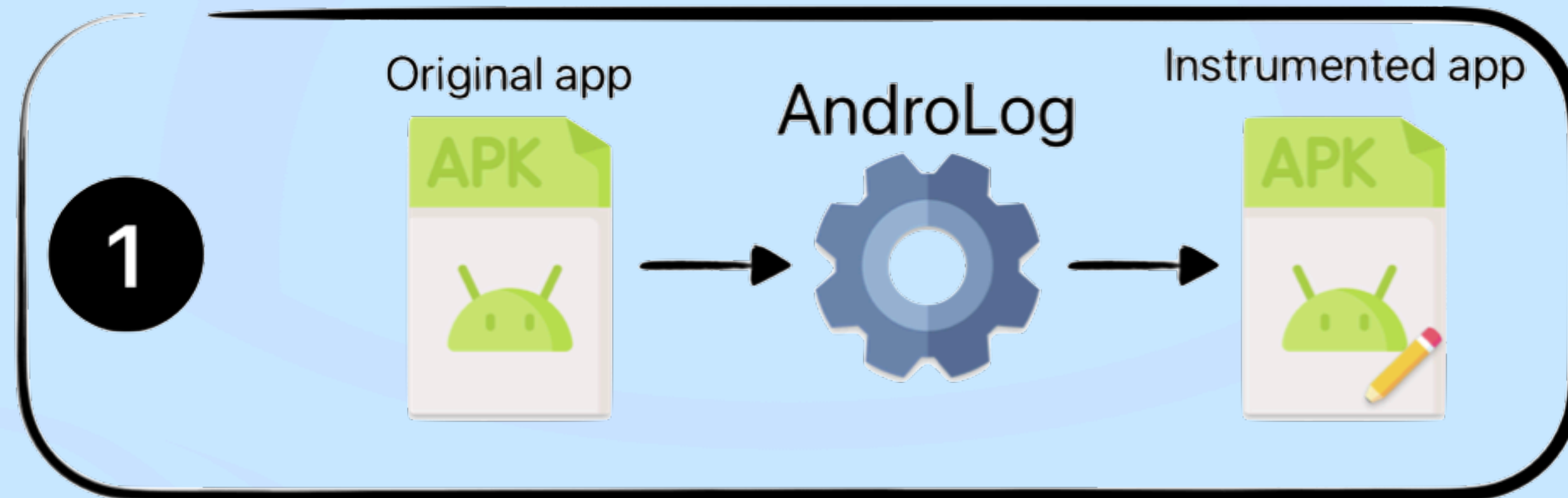
```
public void run() {  
+   LogCheckerClass.log("METHOD=<e: void run()>", "TAG");  
    [...]  
}
```



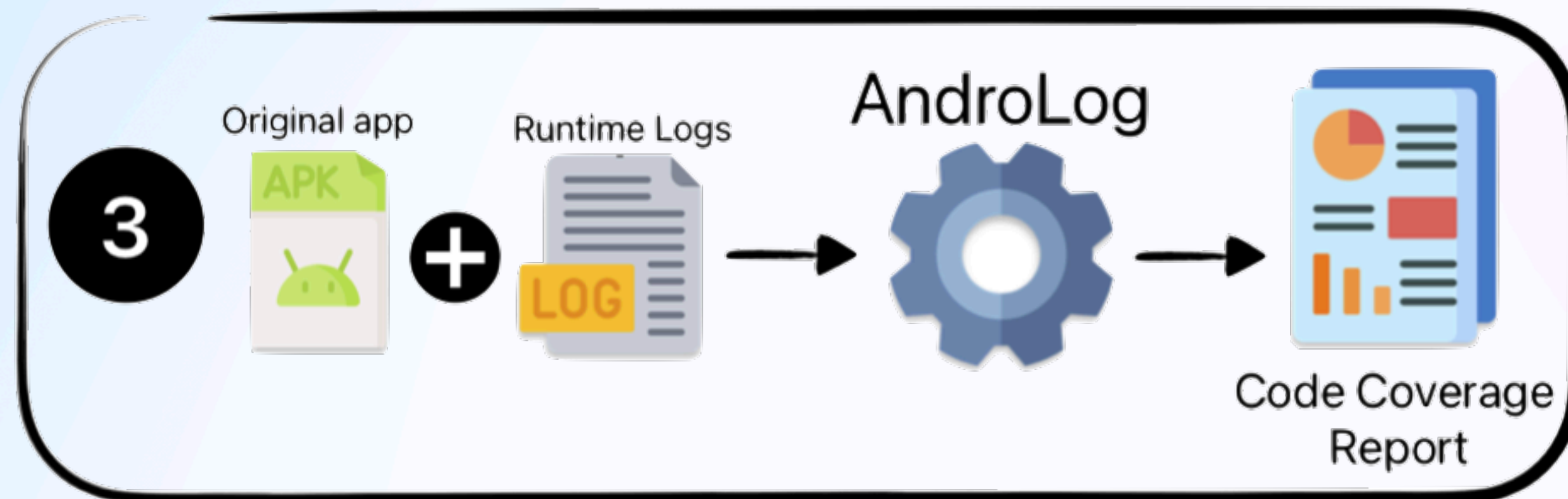
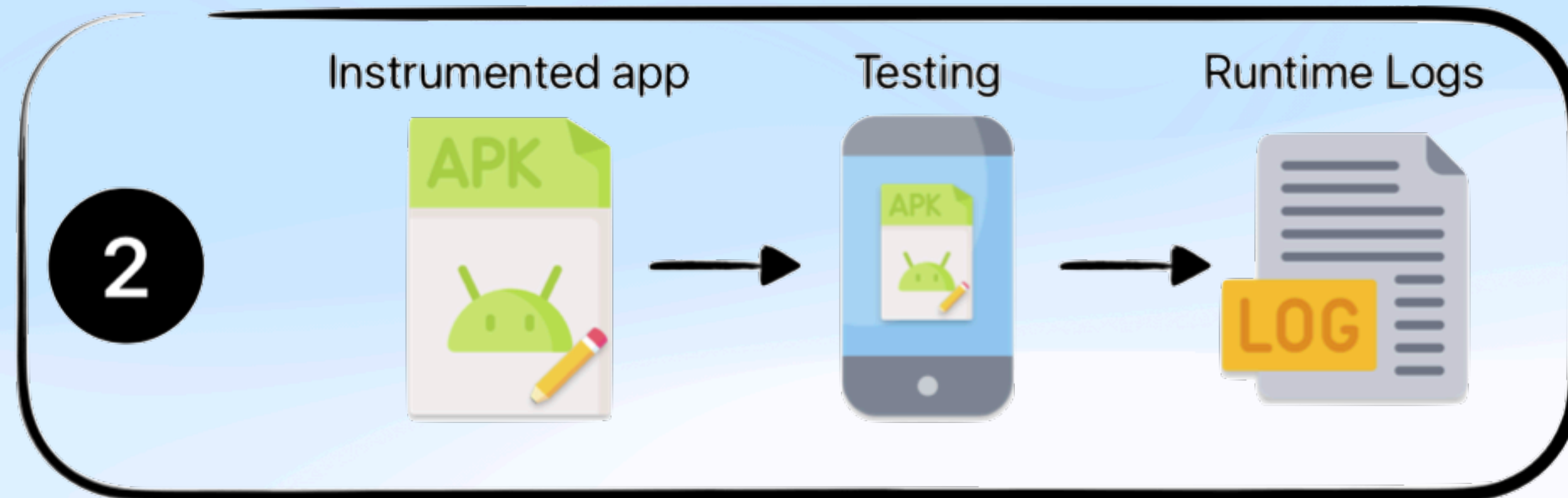
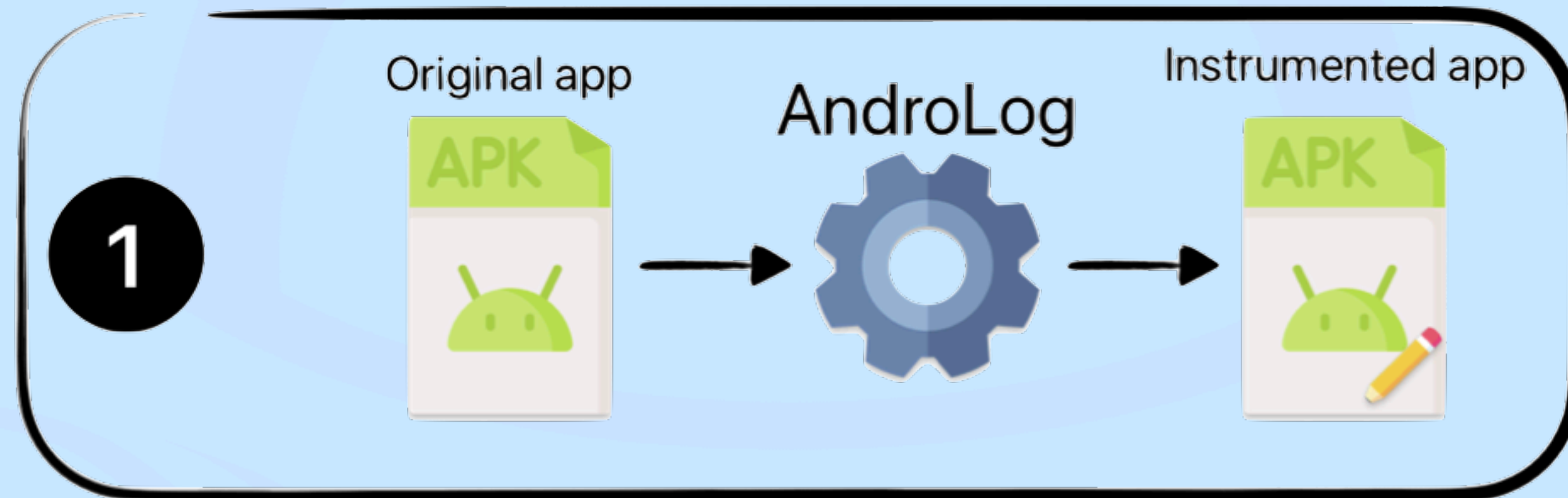
- Bases on well-established Android analysis framework
- AndroLog is test independent
- Analyst chooses how to test the app
- Analyst chooses how to process logs and generate report
- Analyst can build up its own log processor
- AndroLog provides an interface to compute the report

# SIMPLICITY









jordan@Jordans-Macbook-Pro ~/Downloads

```
↳ java -jar androlog.jar -p /Users/jordan/git/Android-platforms/jars/stubs -o /Users/jordan/Downloads/instrumented/ -a 40C6789ABB41E108B8AE469A7469C26F8EDC4EC47CEAAEC0557C8738D9538BF2.apk -l MY_SUPER_LOG -c -m -s -cp
```

AndroLog v0.1 started on Fri Jul 12 17:28:38 CEST 2024

[\*] Setting up environment...

[✓] Done.

[\*] Instrumentation in progress...

[✓] Done.

[\*] Exporting new apk...

[✓] Apk written in: /Users/jordan/Downloads/instrumented/

[\*] Signing and aligning APK...

[✓] Done.

[\*] The apk is now instrumented, install it and execute it to generate logs.

```
jordan@Jordans-Macbook-Pro ~/Downloads
↳ java -jar androlog.jar -p /Users/jordan/git/Android-platforms/jars/stubs -o /Users/jordan/Downloads/instrumented/ -a 40C6789ABB41E108B8
AE469A7469C26F8EDC4EC47CEAAEC0557C8738D9538BF2.apk -l MY_SUPER_LOG -c -m -s -cp -pa logs
AndroLog v0.1 started on Fri Jul 12 17:31:12 CEST 2024

[*] Setting up environment...
[✓] Done.
[*] Generating Code Coverage Report...

=== Coverage Summary ===
-----
activities      : 100.0% (1/1)
methods         : 48.1% (13/27)
classes         : 100.0% (5/5)
statements      : 40.7% (131/322)
-----
```

# Different levels of granularities

- Methods
- Classes
- Statements
- Activities
- Services
- Content Providers
- Broadcast Receivers

**And more!**



```

: AndroLog -a <apk> [-c] [-cp] [-l <log-identifier>] [-m] [-n] [-o
  <output>] -p <platforms> [-pa <parse>] [-s]
-apk <apk>                Apk file
-classes                  Log classes
--components              Log Android components
-log-identifier <log-identifier> Log identifier
-methods                  Log methods
-non-libraries             Whether to include libraries (by
                           default: include libraries)
-output <output>          Instrumented APK output
-platforms <platforms>    Platform file
--parse <parse>           Parse log file
-statements               Log statements

```

```

n@Jordans-Macbook-Pro /tmp/androlog

```

```

va -jar androlog.jar -p ~/git/Android-platforms/jars/stubs/ -a FF33BA42A00DCF70EDE83DB4F2E7F2CCBCEA5D
2FD77E3385D9D223F7075.apk -o ./output/ -l MY_SUPER_LOG -c -m -s -cp
Log v0.1 started on Fri Jan 26 14:53:38 CET 2024

```

```

etting up environment...

```

```

one.

```

```

strumentation in progress...

```

```

one.

```

```

xporting new apk...

```

```

pk written in: ./output/

```

```

igning and aligning APK...

```

```

one.

```

```

he apk is now instrumented, install it and execute it to generate logs.

```

```

n@Jordans-Macbook-Pro /tmp/androlog

```

```

te

```

```

an 26 14:54:50 CET 2024

```

```

n@Jordans-Macbook-Pro /tmp/androlog

```

```

output/

```

```

26920

```

```

-r-xr-x@ 5 jordan wheel 160B Jan 26 14:52 ..

```

```

--r--@ 1 jordan wheel 13M Jan 26 14:54 FF33BA42A00DCF70EDE83DB4F2E7F2CCBCEA5D389DE2FD77E3385D9D22
75.apk

```

```

-r-xr-x@ 3 jordan wheel 96B Jan 26 14:54 .

```

```

n@Jordans-Macbook-Pro /tmp/androlog

```

```

b install output/FF33BA42A00DCF70EDE83DB4F2E7F2CCBCEA5D389DE2FD77E3385D9D223F7075.apk

```

```

forming Streamed Install

```

```

ss

```

```

n@Jordans-Macbook-Pro /tmp/androlog

```

```

b logcat -c

```

```

n@Jordans-Macbook-Pro /tmp/androlog

```

```

b logcat |grep MY_SUPER_LOG > logs

```

```

n@Jordans-Macbook-Pro /tmp/androlog

```

Running Devices: Pixel 3a API 34 extension level 7 arm64-v8a

⏻ 🔊 🔇 🔄 📷 📹 🔄 ⋮



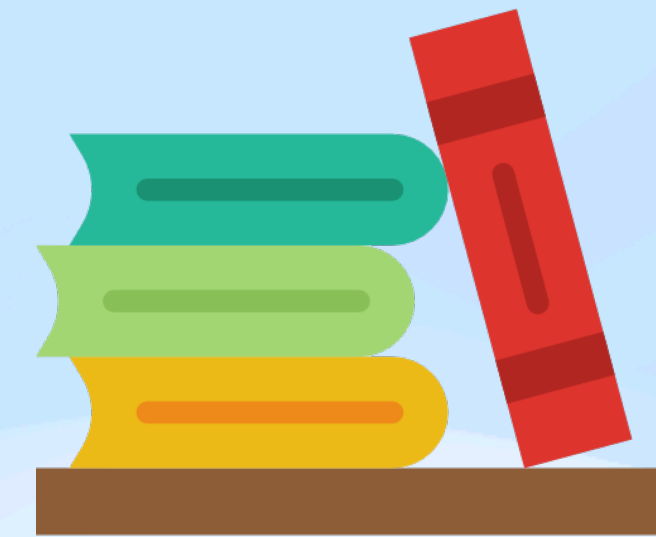
# Challenges of Android Security



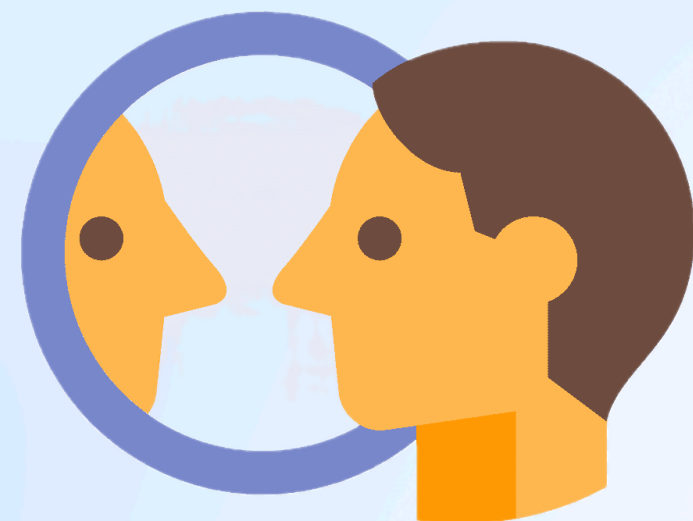
Code Obfuscation



Dynamic Code Loading



Libraries and Frameworks



Reflection



Languages



Limited Visibility







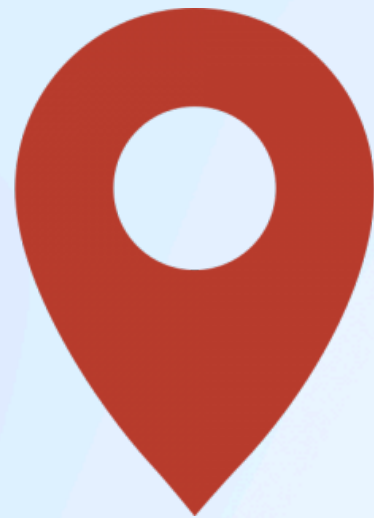
# Malware



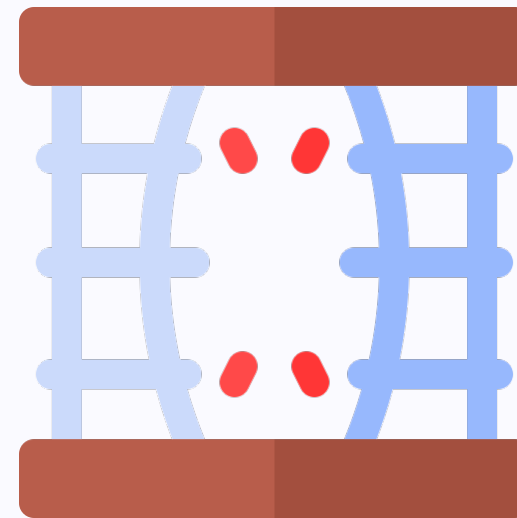
Google Play



Malware evolve rapidly!



Localization



Evasion



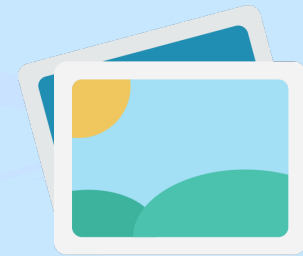
By hand



## Android Security



Bank



Photos



Contacts



Social Medias



Sensors



Health Data

Dr. Jordan Samhi

x

## Malware



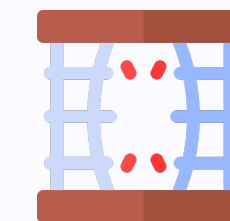
Google Play



Malware evolve rapidly!



Localization



Evasion



By hand

Dr. Jordan Samhi

x

# Jordan Samhi

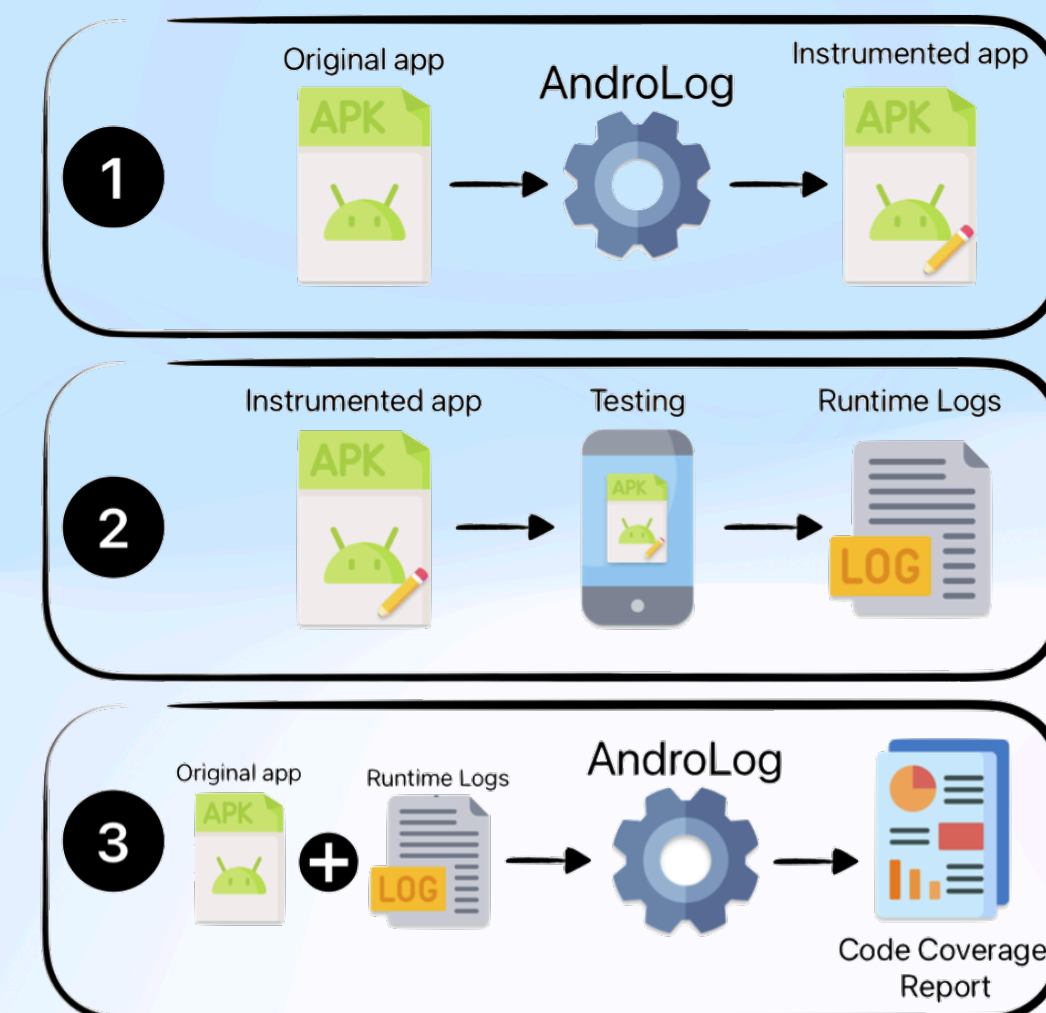
[jordan.samhi@uni.lu](mailto:jordan.samhi@uni.lu)

[jordansamhi.com](http://jordansamhi.com)

TruX

uni.lu  
UNIVERSITÉ DU  
LUXEMBOURG

SNT



x