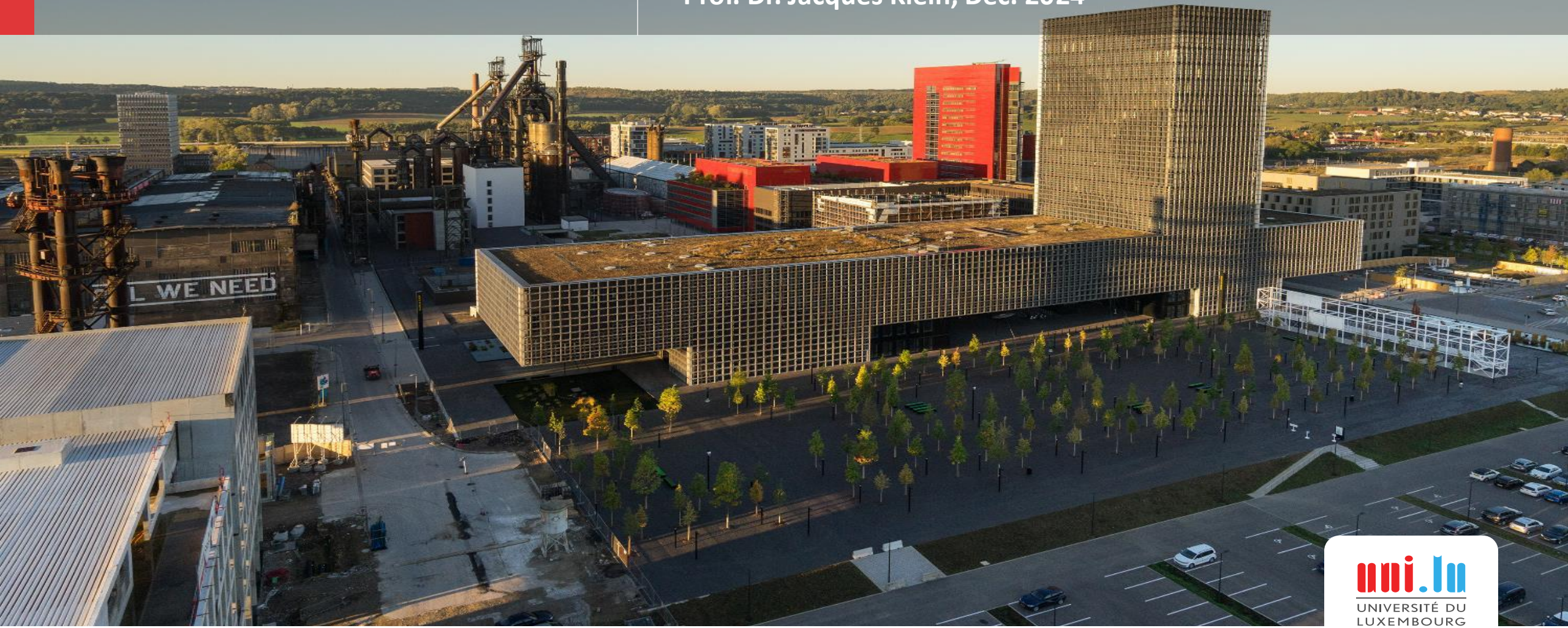
University of Luxembourg

Multilingual. Personalised. Connected.

AI for Software Vulnerabilities and Android Malware Detection

31st Asia-Pacific Software Engineering Conference (APSEC 2024)

Prof. Dr. Jacques Klein, Dec. 2024



UNIVERSITÉ DU LUXEMBOURG

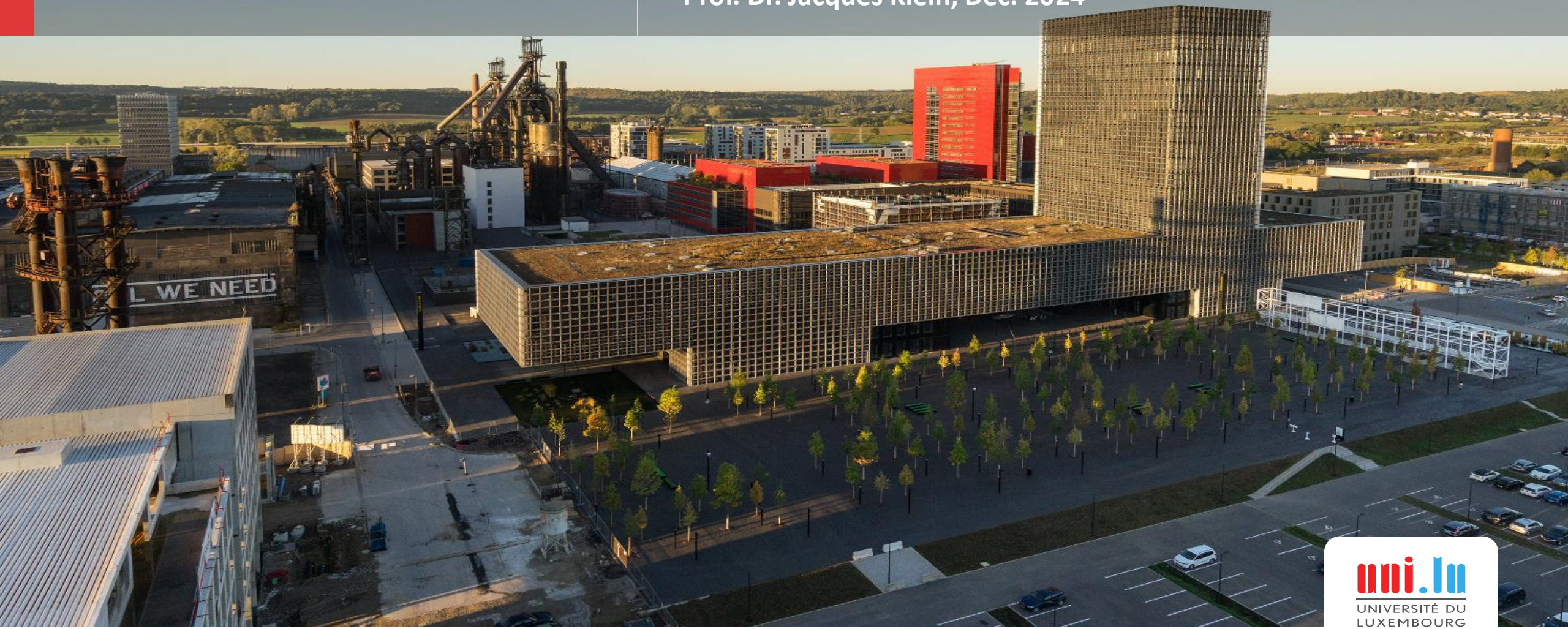
University of Luxembourg

Multilingual. Personalised. Connected.

~~AI for Software Vulnerabilities and Android Malware Detection~~

31st Asia-Pacific Software Engineering Conference (APSEC 2024)

Prof. Dr. Jacques Klein, Dec. 2024



uni.lu
UNIVERSITÉ DU LUXEMBOURG

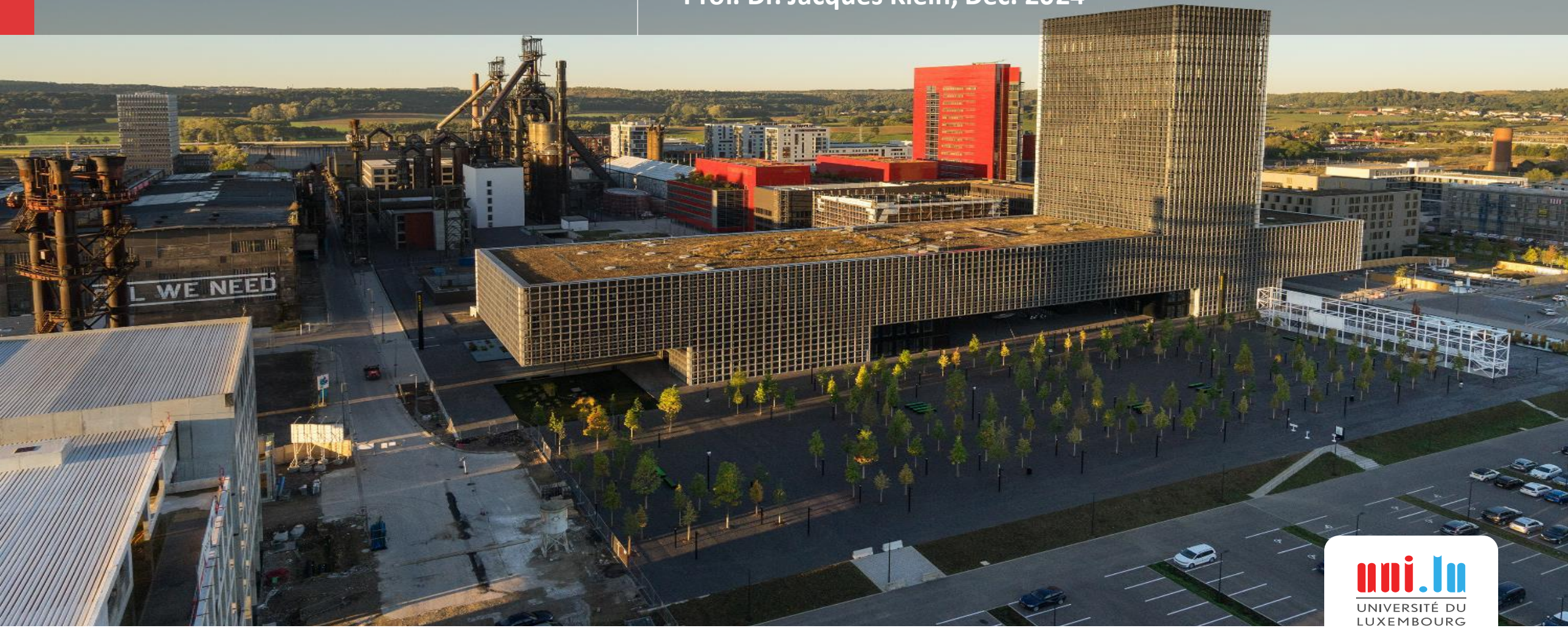
University of Luxembourg

Multilingual. Personalised. Connected.

Mobile App Analysis

31st Asia-Pacific Software Engineering Conference (APSEC 2024)

Prof. Dr. Jacques Klein, Dec. 2024



UNIVERSITÉ DU LUXEMBOURG

Why another topic?

Plenty of young and fearless researchers!

The LLM adventurers

Why not ask old and wise researchers?

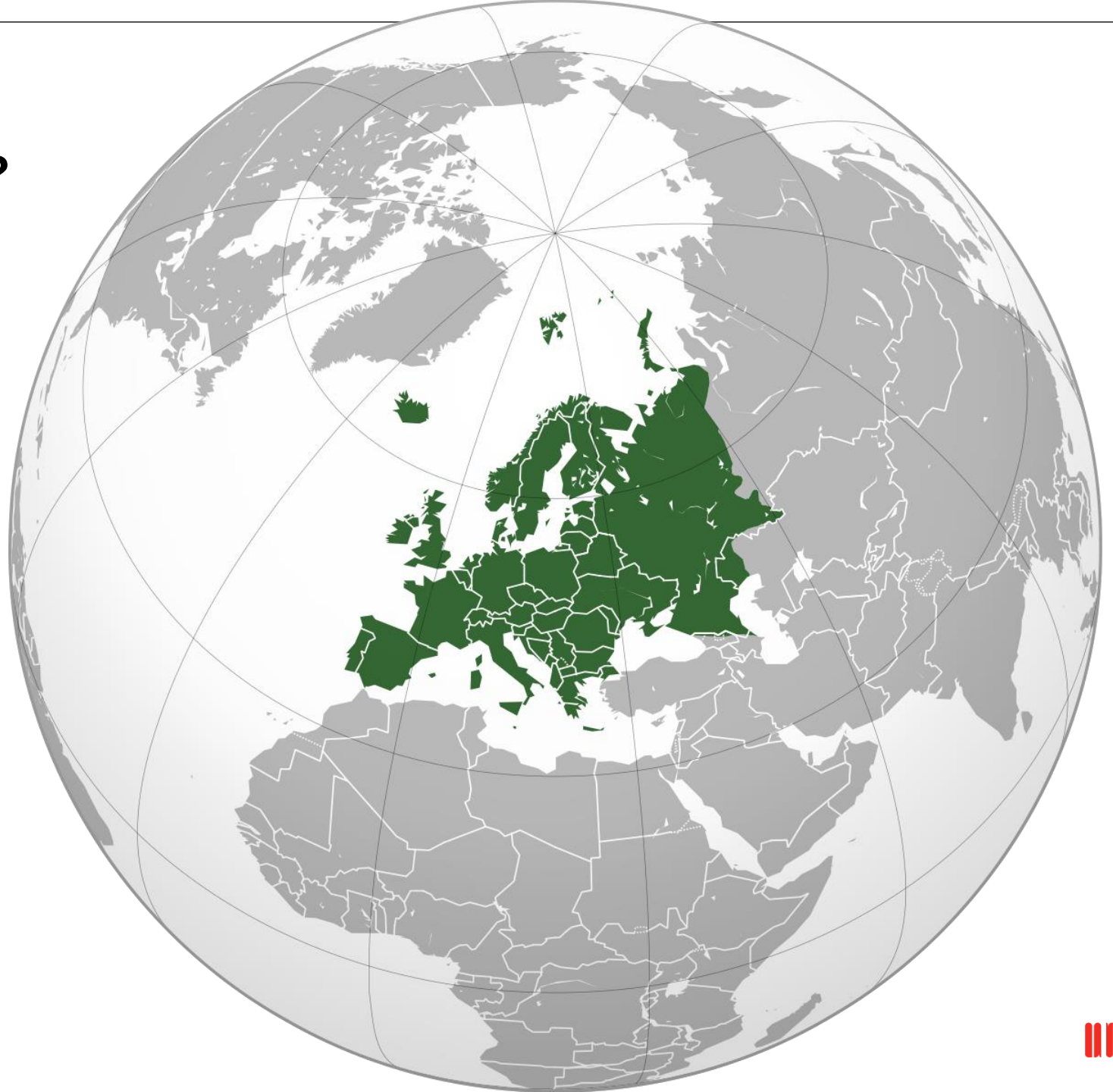
Traditional SE researchers

Let's go back to the roots of Software Engineering

Where is Luxembourg?

UNIVERSITY OF LUXEMBOURG

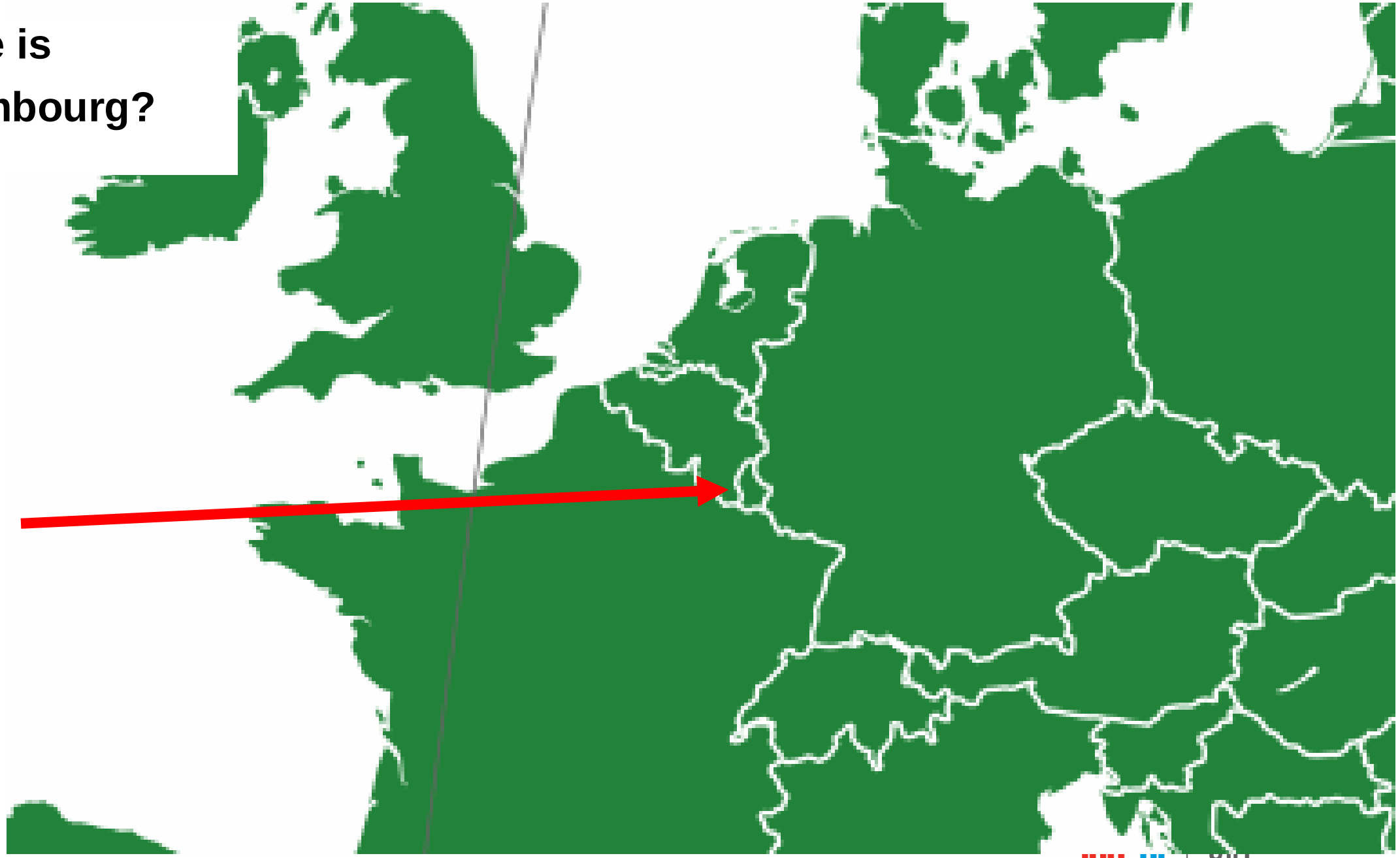
Where is Luxembourg?



Where is Luxembourg?

Where is Luxembourg?

Where is Luxembourg?



The University of Luxembourg

The University of Luxembourg is a research university with a distinctly **international**, **multilingual** and **interdisciplinary** character.

The University's ambition is to provide the **highest quality research** and teaching in its chosen fields and to generate a positive scientific, educational, social, cultural and societal impact in Luxembourg and the Greater Region.

UNIVERSITY OF LUXEMBOURG

Ranked

12th Young University

worldwide and #1 worldwide for its "international outlook" in the Times Higher Education (THE) World University Rankings 2020

~7000 students

~1000 PhDs

270 faculty members

129 nationalities

56% international students

TruX

SnT

Trustworthy Software Engineering
TruX Research Group

Prof. Tegawendé F. BISSYANDE

Prof. Jacques KLEIN

UNIVERSITY OF LUXEMBOURG

TruX People

Professors

- Tegawendé F. BISSYANDE (head)
- Jacques KLEIN (co-head)

Research Associates

1. Yinghua LI

Research Scientist

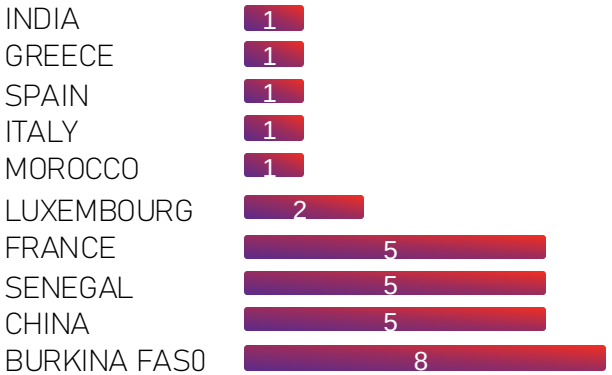
1. Jordan SAMHI

Assistant

- Fiona LEVASSEUR

Coming Soon

1. Paweł BORSUKIEWICZ



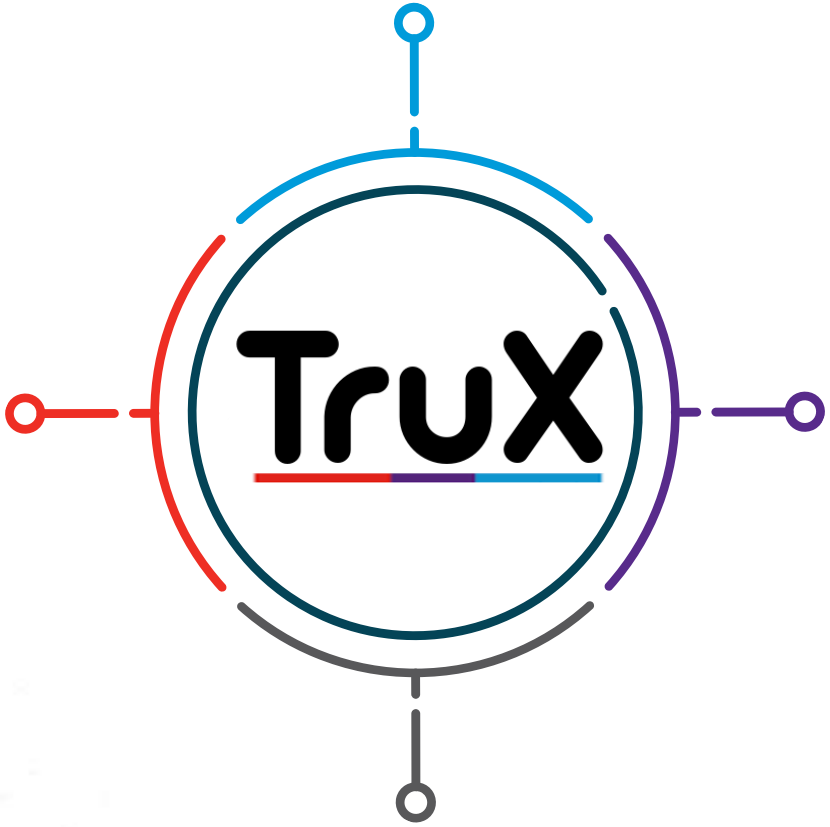
PhD Students

1. Fatou Ndiaye MBODJI (Apr. 2021)
2. Tiezhu SUN (Apr. 2021)
3. Xunzhu TANG (Oct. 2021)
4. Damien FRANCOIS (Nov. 2021)
5. Weiguo PIAN (Jan 2022)
6. Alioune DIALLO (Feb. 2022)
7. Christian OUEDRAOGO (Apr. 2022)
8. Aicha WAR (May 2022)
9. Yewei SONG (Jun. 2022)
10. Despoina GIARIMPAMPA (Sep. 2022)
11. Marco ALECCI (Oct. 2022)
12. Fred PHILIPPY (Mar. 2023)
13. Jules WAX (Mar. 2023)
14. Moustapha DIOUF (Apr. 2023)
15. Micheline MOUMOULA (Oct. 2023)
16. Pedro RUIZ JIMÉNEZ (Nov. 2023)
17. Omar EL BACHYR (Feb. 2024)
18. Prateek RAJPUT (Mar. 2024)
19. Albérick DJIRE (Mar. 2024)
20. Maimouna Tamah DIAO (Apr. 2024)
21. Maimouna OUATTARA (May 2024)
22. Aziz BONKOUNGOU (Jul. 2024)
23. Serge Lionel NIKIEMA (Jul. 2024)
24. Loic TALEB (Dec, 2024)

Trustworthy Software Engineering

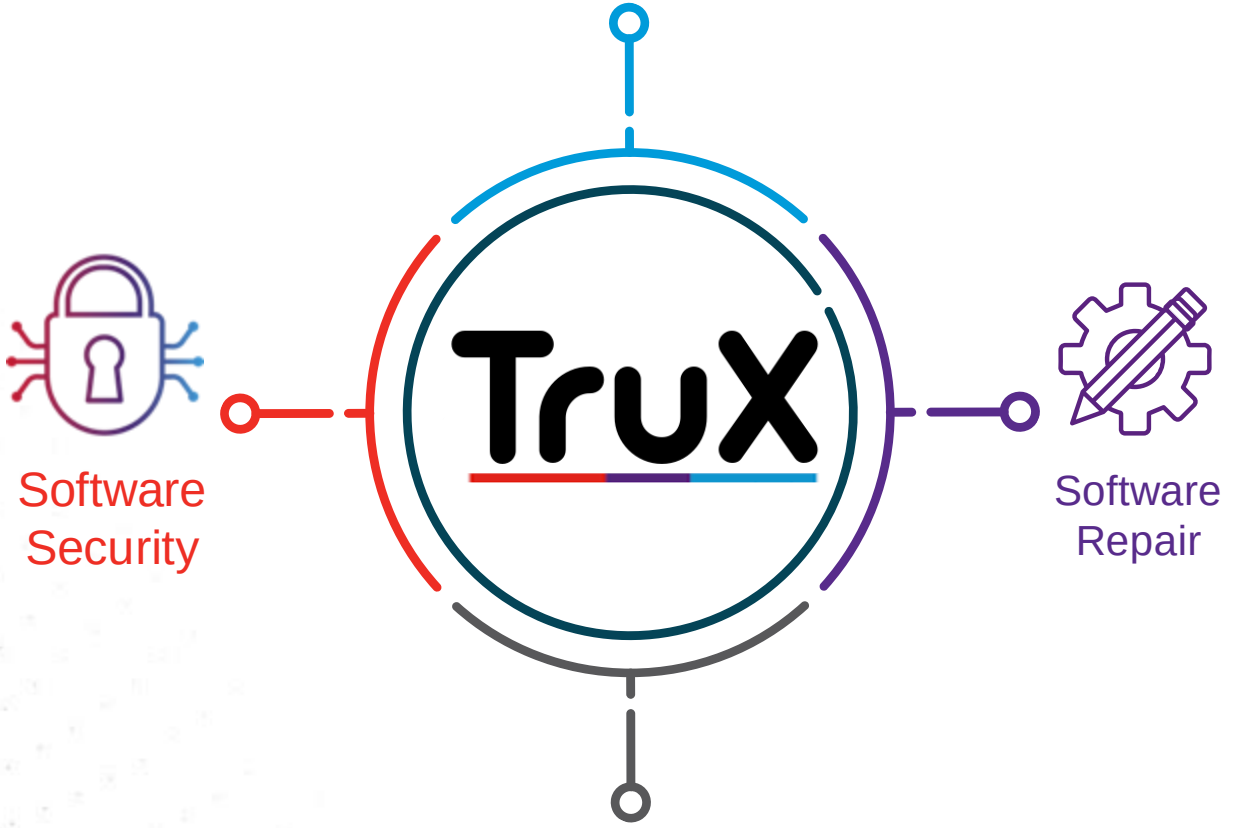
- **Vulnerability detection, Android app Analysis (e.g., Data Leaks)**
- GDPR compliance
- Malware Detection, Piggybacking Detection

Software Security



Trustworthy Software Engineering

- **Vulnerability detection, Android app Analysis (e.g., Data Leaks)**
- GDPR compliance
- Malware Detection, Piggybacking Detection

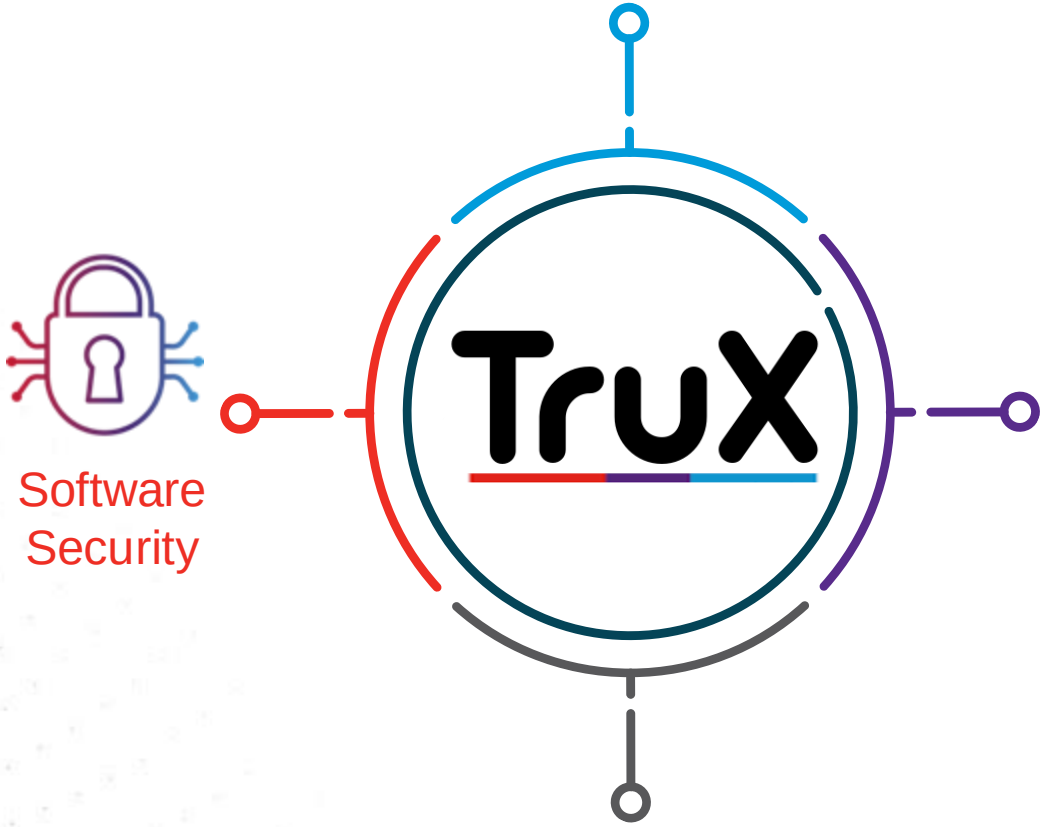


- Patch Recommendation
- Automated Program Repair
- **Bug Detection**
- Vulnerability patching

Trustworthy Software Engineering

Explainable Software

- Information Retrieval
- Natural Language Processing
- Time Series Pattern Recognition
- Machine learning, Explainable ML



Software Security

- **Vulnerability detection, Android app Analysis (e.g., Data Leaks)**
- GDPR compliance
- Malware Detection, Piggybacking Detection

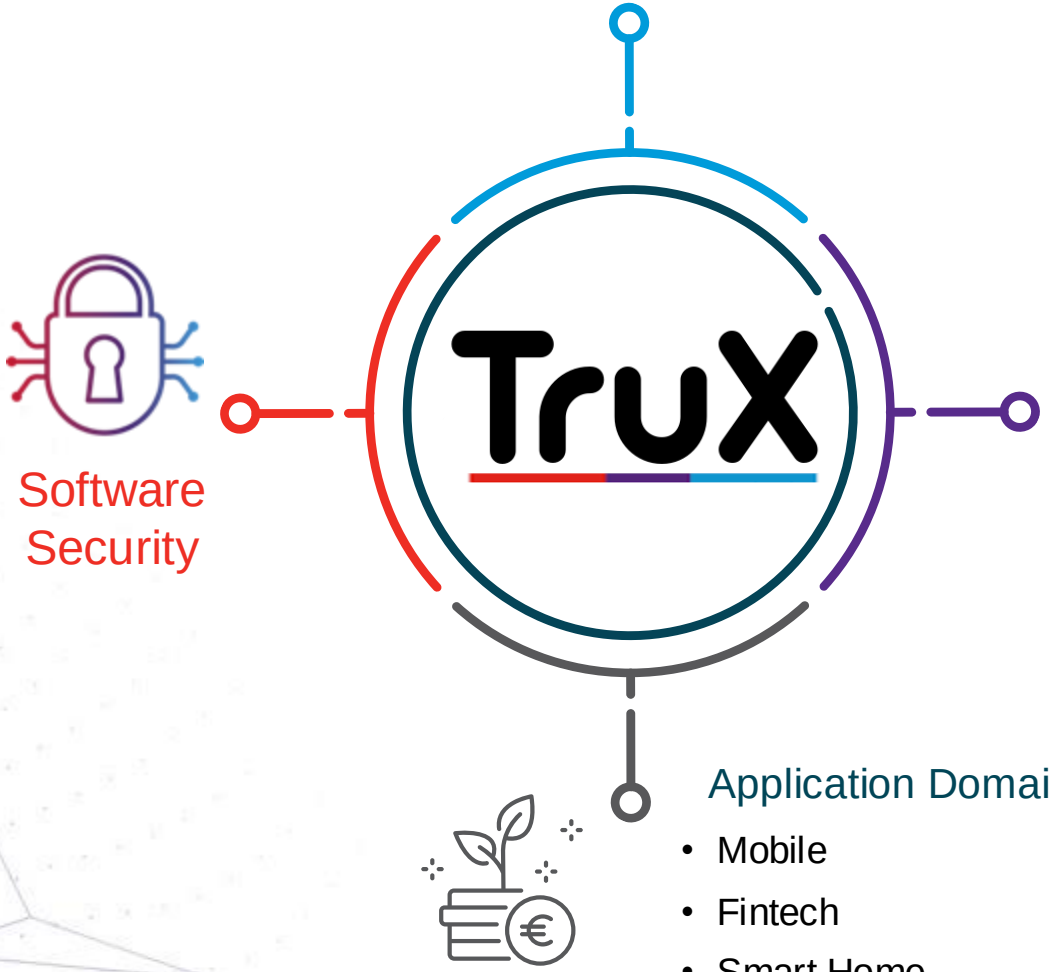
Software Repair

- Patch Recommendation
- Automated Program Repair
- **Bug Detection**
- Vulnerability patching

Trustworthy Software Engineering

Explainable Software

- Information Retrieval
- Natural Language Processing
- Time Series Pattern Recognition
- Machine learning, Explainable ML



- **Vulnerability detection, Android app Analysis (e.g., Data Leaks)**
- GDPR compliance
- Malware Detection, Piggybacking Detection

- Patch Recommendation
- Automated Program Repair
- **Bug Detection**
- Vulnerability patching

- Mobile
- Fintech
- Smart Home
- Business Critical Systems

PhD students

Post-Docs


SNT
Mobile
App
Analysis
UNIVERSITY OF
LUXEMBOURG


SNT
Mobile
App
Analysis
UNIVERSITY OF
LUXEMBOURG

Android App Analysis

UNIVERSITY OF LUXEMBOURG

Why Android App Analysis is important?

More than 6 billion **people** own a smartphone

Almost three-quarters are Android-based

We manipulate a lot of sensitive data

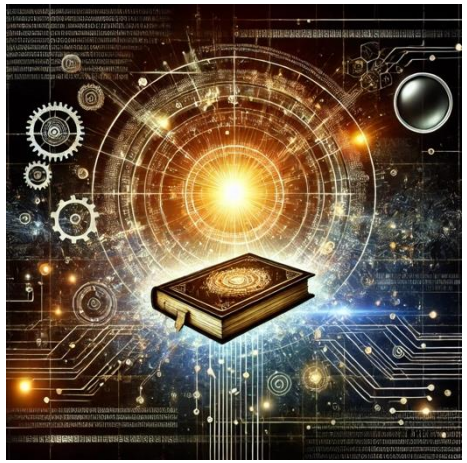

Google patches critical Android vulnerability that allows for es privileges
Dozens of vulnerabilities fixed in Xiaomi, Google Android flavors … slowly
Oversecure details bugs spotted and stamped since private disclosure
ESET researchers discovered a zero-day Telegram for Android exploit that allows sending malicious files disguised as videos
g the EvilVideo Telegram for Android
lusa banking trojan returns to steal your words and cash — how to stay safe
Anthony Spadafora published June 25, 2024
ightweight variant can capture screenshots and use full-
rlays
Samsung says a critical security patch is making its way to Galaxy devices soon
MANDRAKE ANDROID SPYWARE FOUND IN FIVE APPS IN GOOGLE PLAY WITH OVER 32,000
New 'Brokewell' Android Malware Spread Through Fake Browser Updates
Apr 26, 2024
Ravie Lakshmanan
Mobile Security / Cybercrime
RAFEL RAT, ANDROID MALWARE RANSOMWARE OPERATIONS
June 20, 2024
Research by: Antonis Terefos, Bohdan Melnykov
his notorious Android bank lets hackers remotely cont how to stay safe
By Anthony Spadafora published April 3, 2024
News
New upgrade lets hackers bypass the
MOBILE
Tik
Have trouble with TikTok? Here are some
By Adam Birney • June 28, 2024
Elizabeth Montalbano
July 17, 202
Android Mal
in mobile Trojans like TeaBot and others,
4 Min Read
for Android allowed masquerade as videos
a zero-day exploit for the Telegram messaging app on that could have allowed attackers to send malicious payloads disguised as
make sure to che
carefully
News
By Sead Fadilpašić published 18 hours ago
Updated version of Mandrake Android malware
New Vulnerability Discovered in Android Devices Sparks Security Concerns
By Michał Nawrocki
2024-06-24
ode Execution Vulnerability Patched in
Android's February 2024 security patches resolve 46 vulnerabilities, including a critical remote code execution bug.
Just a fraction of 2024!

Agenda

1. The need for a large set of Apps

AndroZoo

2. Static Analysis The Genesis



The Past

3. Static Analysis Soundness?

The Present

4. Better Analysis!

The Future

Agenda

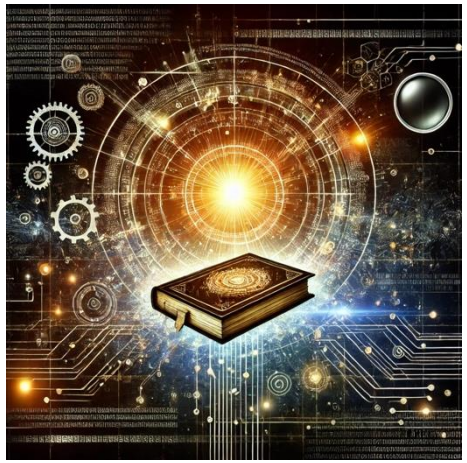
1

The need for a large set of Apps

AndroZoo

2

Static Analysis The Genesis



The Past

3

Static Analysis Soundness?

The Present

4

Better Analysis!

The Future

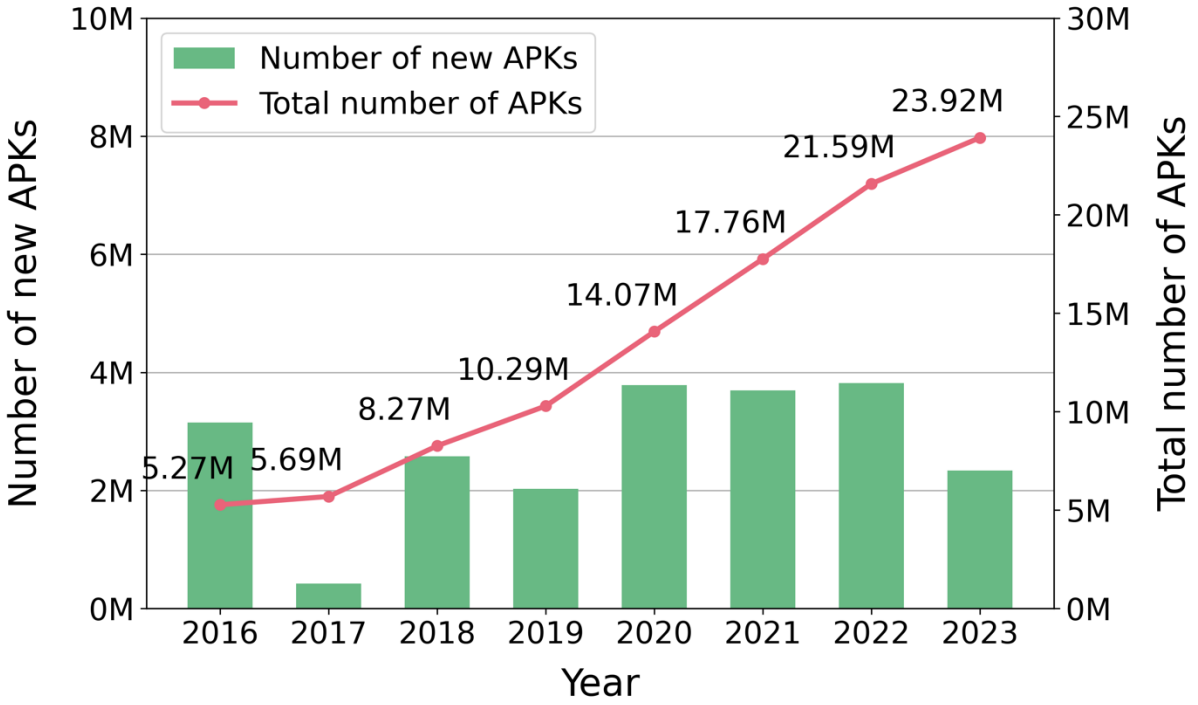
AndroZoo
A repository of Android Apps

[MSR 2016] AndroZoo: Collecting Millions of Android Apps for the Research Community

AndroZoo: A Retrospective

AndroZoo is currently the **biggest dataset of Android apps, with 24 million entries.**
It was created in **2016 at the University of Luxembourg.**

Constantly growing



[MSR 2024]: AndroZoo: A Retrospective with a Glimpse into the Future

AndroZoo: A Retrospective

App ≠ Apk

24 million apks, but 8 708 304 apps (average of 2.7 apks for each app)

Table 1: Top 10 apps by number of APKs

Package Name	#APKs
com.chrome.canary	1986
org.mozilla.fenix	1811
wp.wpbeta	910
dating.app.chat.flirt.wgbcv	826
com.blackforestapppaid.blackforest	822
com.brave.browser_nightly	787
com.topwar.gp	728
com.opodo.reisen	688
com.edreams.travel	679
com.styleseat.promobile	675

Table 2: Lifespan of apps in AndroZoo

#Years	#Apps	#Years	#Apps	#Years	#Apps
10	9347	6	37 099	2	315 206
9	20 072	5	84 931	1	432 536
8	20 171	4	108 962	0	2 732 016
7	37 378	3	186 800		

[MSR 2024]: AndroZoo: A Retrospective with a Glimpse into the Future

AndroZoo: A Retrospective

From November 2021 to November 2023:

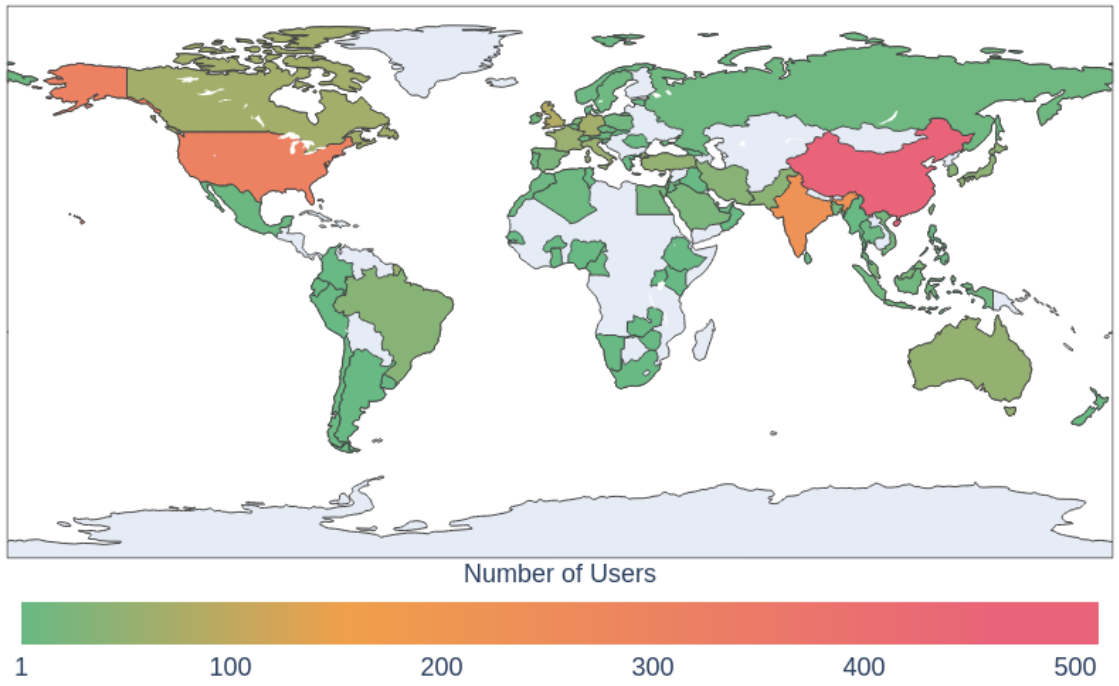
365 604 948 download requests from 692 different users

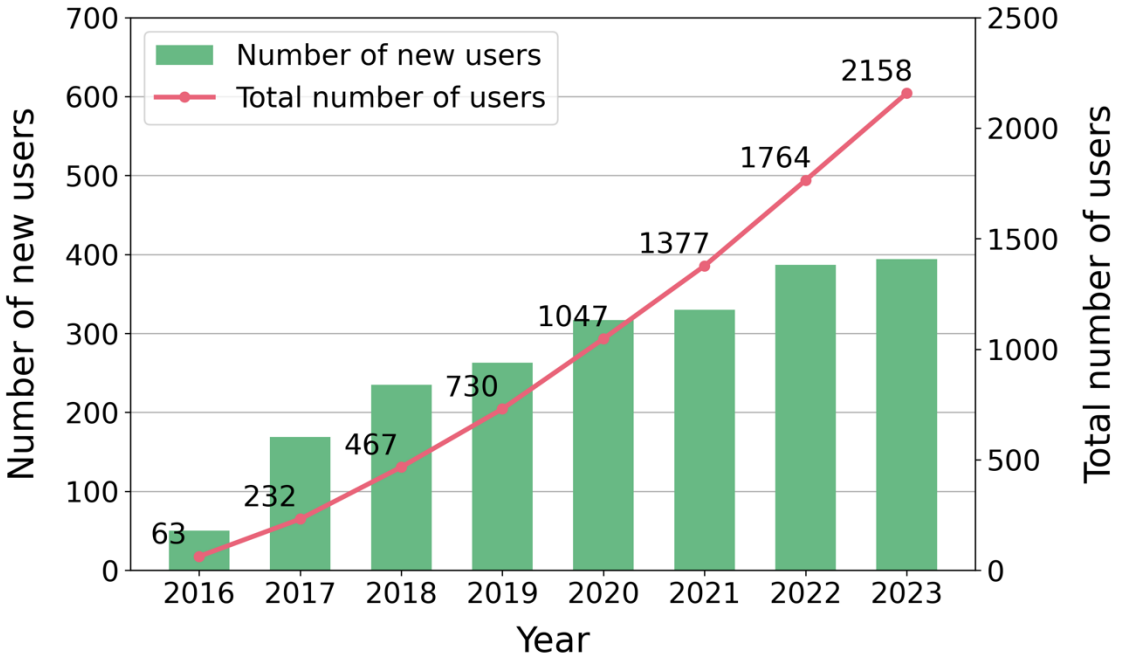
=> 4 PiB of data sent

[MSR 2024]: AndroZoo: A Retrospective with a Glimpse into the Future

AndroZoo: A Retrospective

AndroZoo is currently used by **more then 2000 users worldwide.**





AndroZoo: A Glimpse into the Future

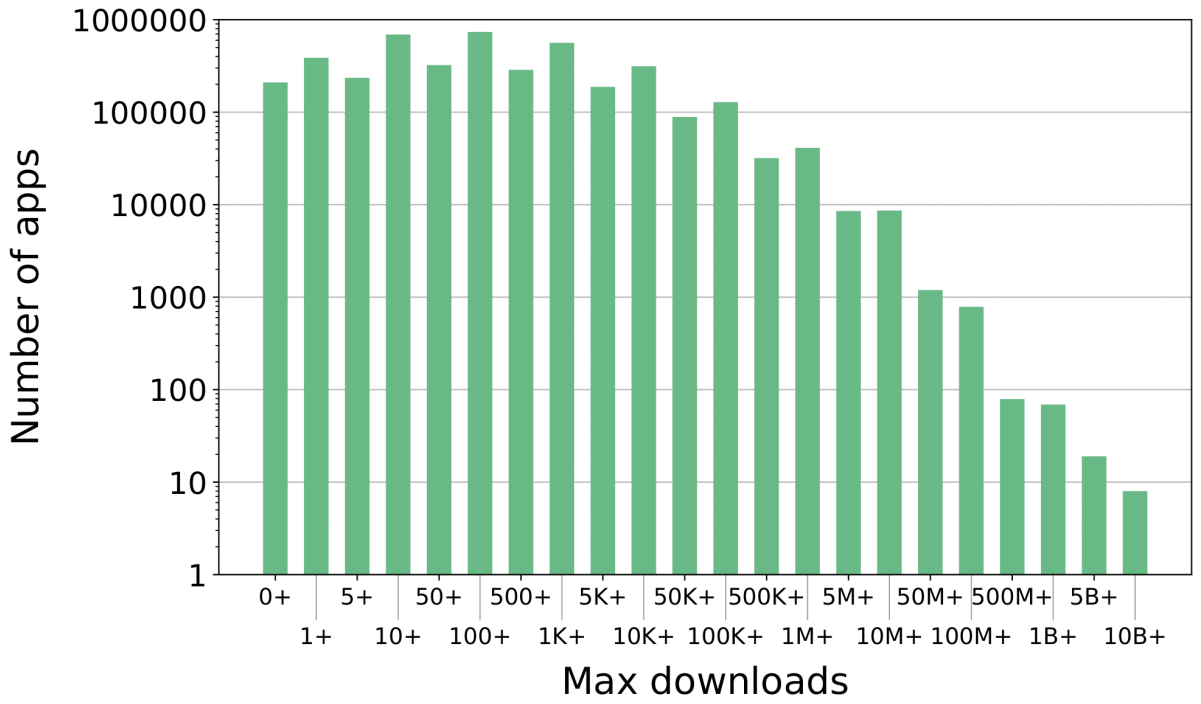
NEW

We started **collecting metadata since 2020**, and we are now **releasing them in AndroZoo** together with the apps.

EXAMPLE

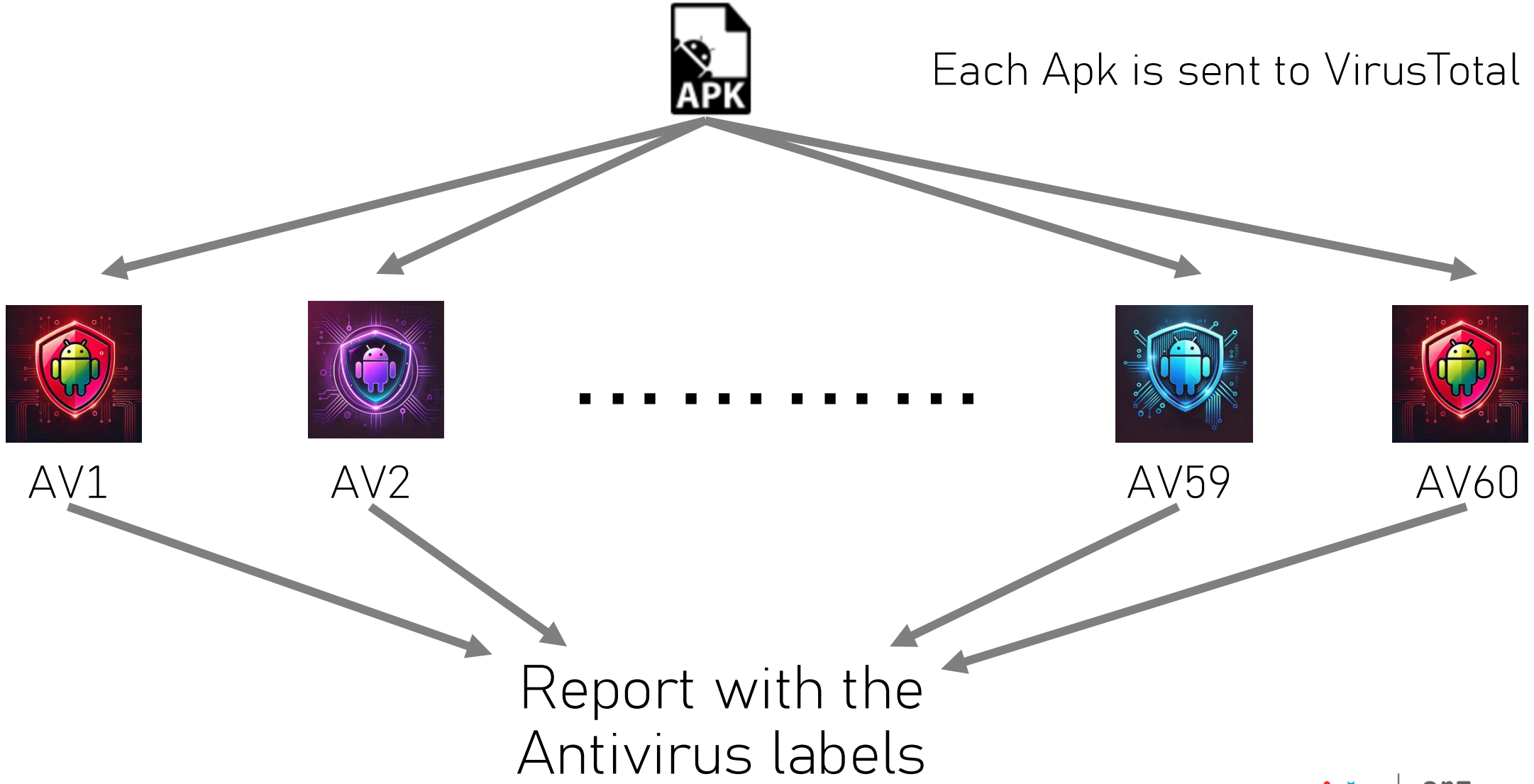
A few examples:

- Description
- Number of Downloads
- Ratings
- Permissions
- Upload Date
- Privacy Policy Link
- …. many others ….



What can you do with AndroZoo?

AndroZoo for Malware Investigation



AndroZoo for Malware Investigation

On 21,570,017 apks from Google Play
sent to VirusTotal,
85,782 have been tagged
by at least 10 Antivirus products

What can you do with AndroZoo?

Another Example

AndroZoo for Large Scale Empirical Studies

Let's start with a simple question

AndroZoo for Large Scale Empirical Studies

Let's start with a simple question

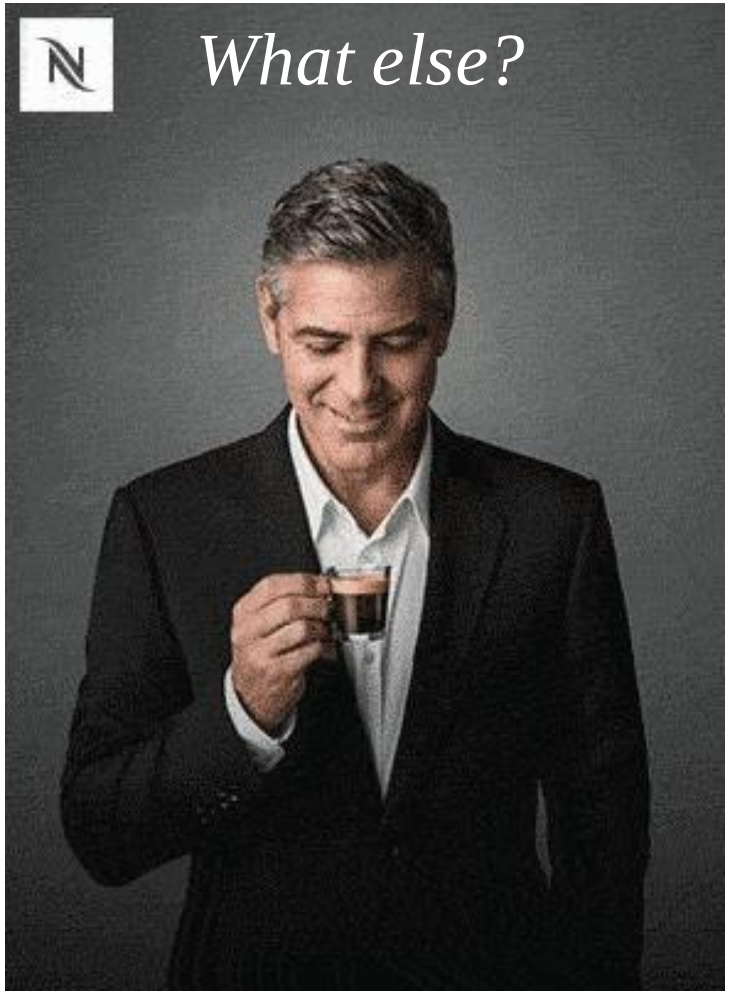
Do you know what is inside an Android App?

AndroZoo for Large Scale Empirical Studies

Let's start with a simple question

Do you know what is inside an Android App?

AndroZoo for Large Scale Empirical Studies



AndroZoo for Large Scale Empirical Studies

We dissected 410 125 apks

How many files?

270 million files
661 files on average

How many file extensions (.dex,.jpg, .png)?

Over 15,000 file extensions

How many file types?

1000 file types

Other interesting facts

- Several apks embed another apk file
- 10% of apks contain compressed files

Agenda

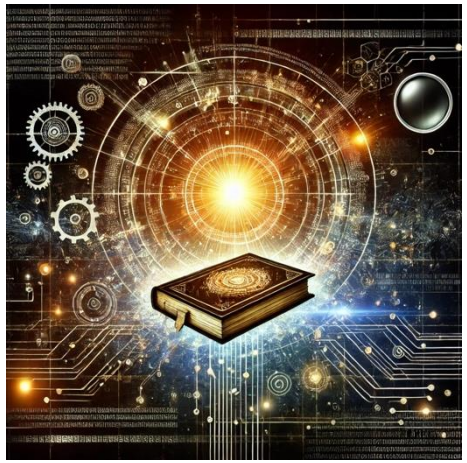
1

The need for a large set of Apps

AndroZoo

2

Static Analysis The Genesis



The Past

3

Static Analysis Soundness?

The Present

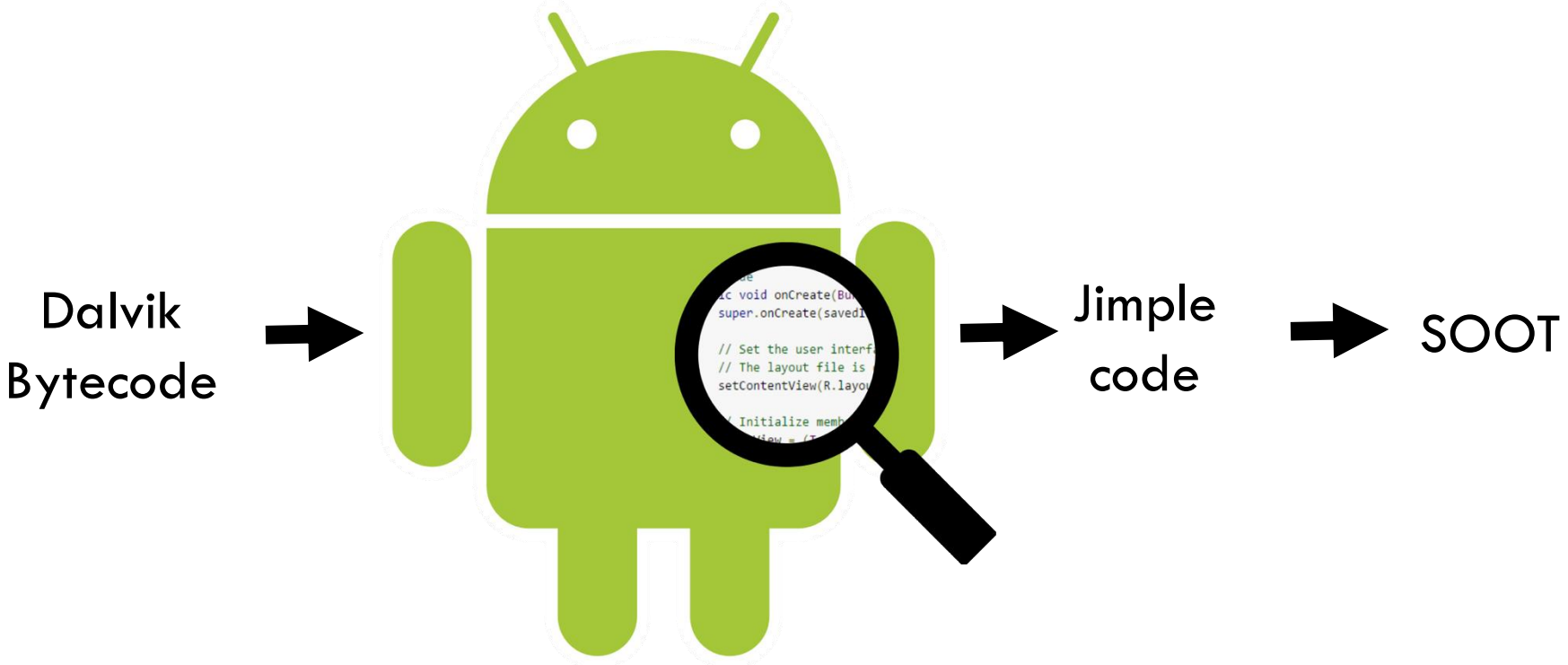
4

Better Analysis!

The Future

Analyzing Android Apps (static)

Frist, need of decompiling Android App



Dexpler [2012]

Data Leaks


Popular Android Apps Leaking
Report Finds
By Chloe Albanesius
Skype for Android leaks sensitive data
WhatsApp leaks use
and messages
19 MAY 2011
APPLICATIONS
SPYING BIRDS
eaking
velopers may
Angry Birds and other Mobile Gaming apps leaking your private information to NSA
by Swati Khandelwal on Monday, January 27, 2014
Hackers
By DANIEL BATE
PUBLISHED: 10:13
Published January
Appthority Security Team
apps leak user privacy data
find permitted apps transmit phone numbers, location, and SIM card IDs

Data Leaks

Data Leaks for Android Apps FlowDroid [PLDI'14]

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

Example of Leak within a single component

```
public class Activity1 extends ActionBarActivity {

    TelephonyManager telManager;
    SmsManager sms;

    protected void onCreateSimpleCase(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_activity1);
        String id = telManager.getDeviceId();
       //...
        String number="+3524666445600";
        sms.sendTextMessage(number,null,id,null,null);
    }
```

Example of Leak within a single component

```
public class Activity1 extends ActionBarActivity {

    TelephonyManager telManager;
    SmsManager sms;

    protected void onCreateSimpleCase(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_activity1);
        String id = telManager.getDeviceId();
       //...
        String number="+3524666445600";
        sms.sendTextMessage(number,null,id,null,null);
    }
```

source

Example of Leak within a single component

```
public class Activity1 extends ActionBarActivity {

    TelephonyManager telManager;
    SmsManager sms;

    protected void onCreateSimpleCase(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_activity1);
        String id = telManager.getDeviceId();
        //...
        String number="+3524666445600";
        sms.sendTextMessage(number,null,id,null,null);
    }
```

source

sink

Example of Leak within a single component

```
public class Activity1 extends ActionBarActivity {

    TelephonyManager telManager;
    SmsManager sms;

    protected void onCreateSimpleCase(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_activity1);
        String id = telManager.getDeviceId();
        //...
        String number="+3524666445600";
        sms.sendTextMessage(number,null,id,null,null);
    }
```

source

sink

One of the main contributions of FlowDroid

Modeling of the lifecycle methods

So far so good,...

But in Android, do not forget

Inter-Component

Communication

(ICC)

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);
    }
```

source

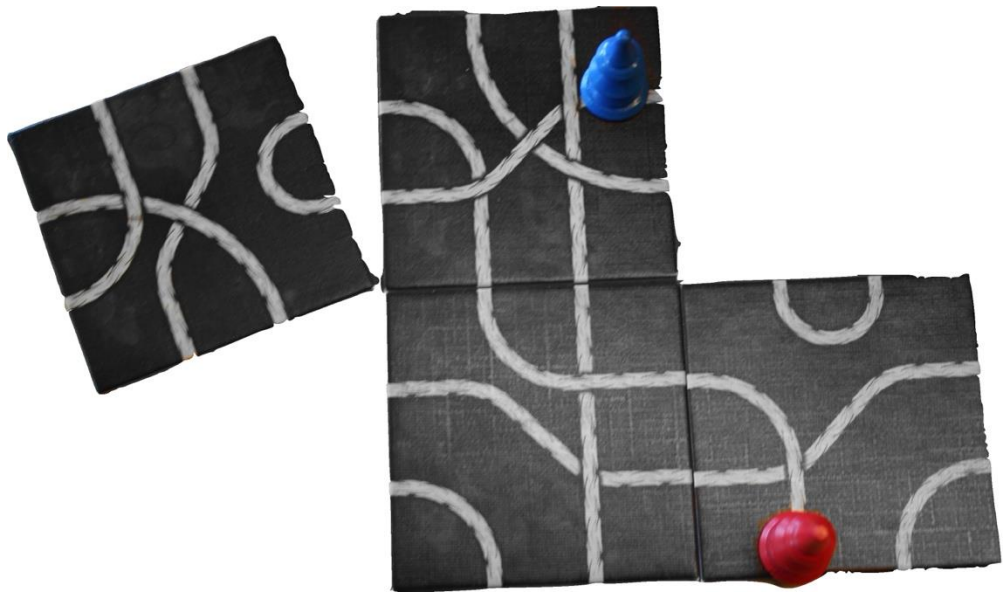
Difficulty: ICC

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);

    }
```

sink

Data Leaks



To solve this issue, we proposed

ICCTA (ICSE 2015)

- Leverage a string retrieval approach that we presented at Usenix Security 2013
- We instrument the app to add an explicit method call

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);
    }
```

source

Activity_B ab = new Activity_B();
ab.onCreate(…)

```
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

Thanks to our Colleagues from

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Resolving reflection: FlowDroid+ICCTA Extensions

- DroidRA: Taming Reflection to Support Whole-Program Analysis of Android Apps [ISSTA 2016, TOSEM 2020]

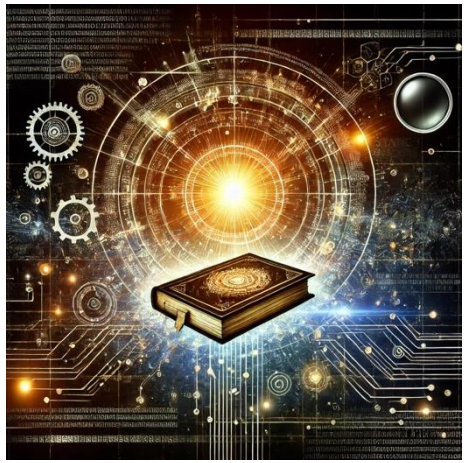
```
1  TelephonyManager telephonyManager = //default;
2  String imei = telephonyManager.getDeviceId();
3  Class c =
       Class.forName("de.ecspride.ReflectiveClass");
4  Object o = c.newInstance();
5  Method m = c.getMethod("setIme" + "i",
       String.class);
6  m.invoke(o, imei);
7  Method m2 = c.getMethod("getImei");
8  String s = (String) m2.invoke(o);
9  SmsManager sms = SmsManager.getDefault();
10 sms.sendTextMessage("+49 1234", null, s, null,
       null);
```

Agenda

1. The need for a large set of Apps

AndroZoo

2. Static Analysis The Genesis



The Past

3. Static Analysis Soundness?

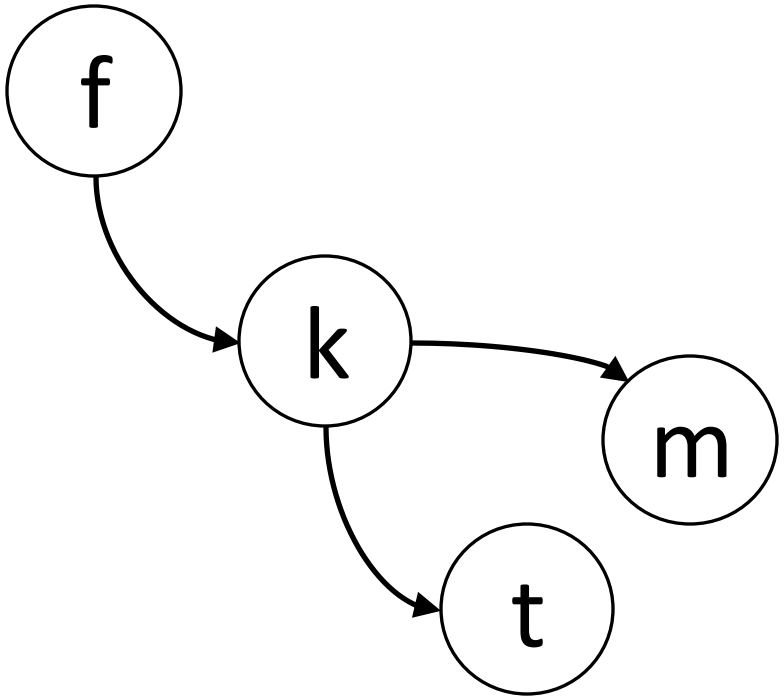
The Present

4. Better Analysis!

The Future

Call Graph

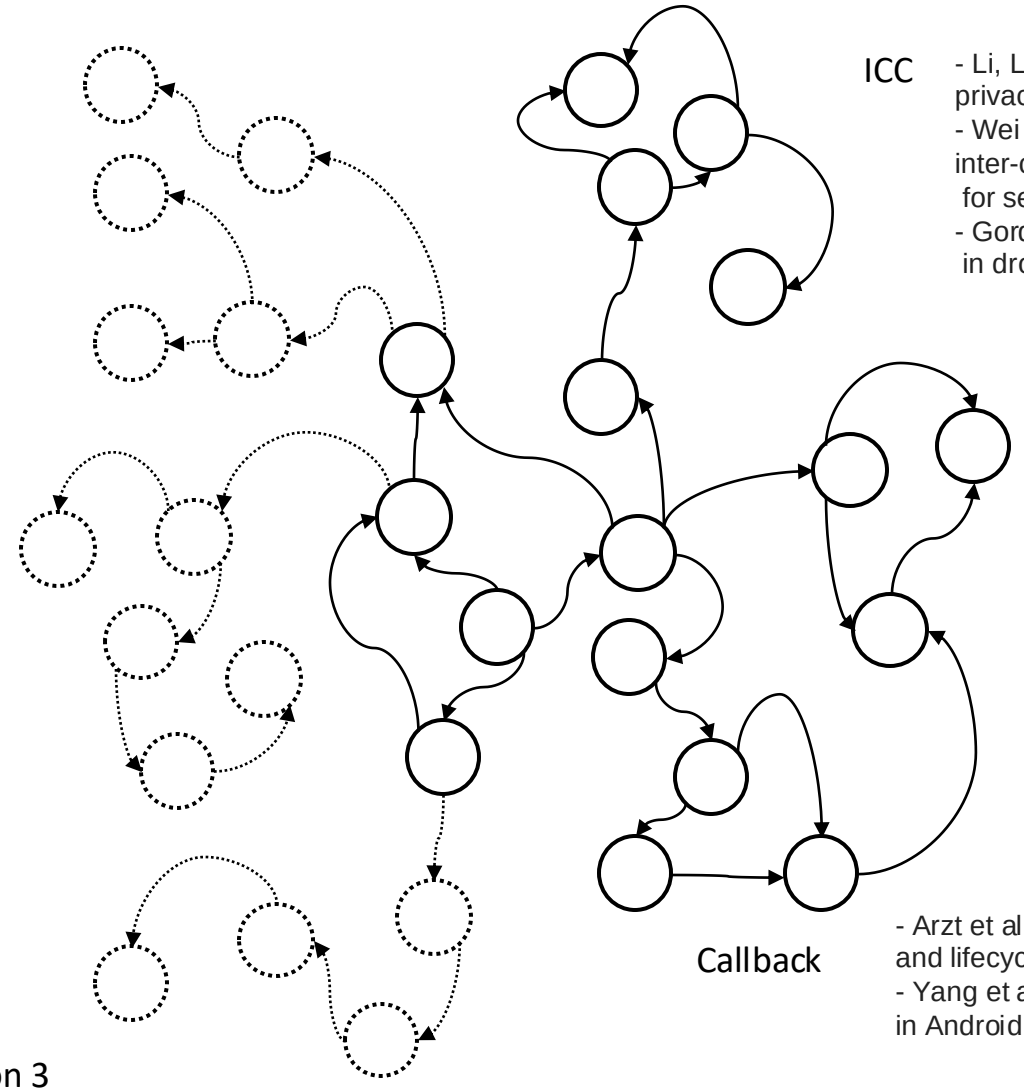
```
f()  {
        a = 4;
        b = 2 * a;
        k(b);
}
g(a) {
        b = t(a);
        m(b);
}
```



Contribution 1

Contribution 2

Contribution 3



ICC

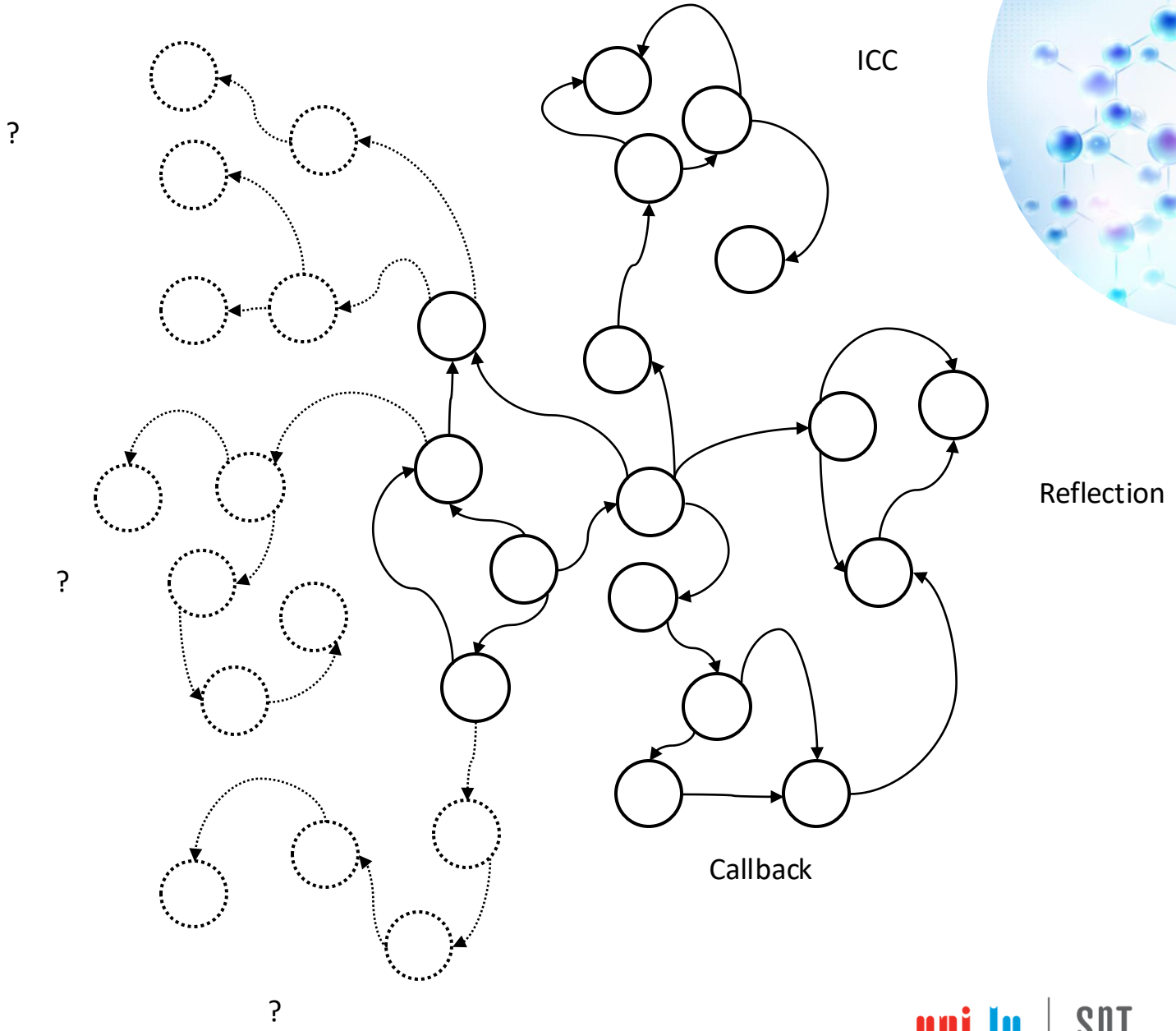
- Li, Li et al. Iccta: Detecting inter-component privacy leaks in android apps. ICSE 2015.
- Wei et al., Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. *TOPS 2018*.
- Gordon et al. Information flow analysis of android applications in droidsafe. NDSS 2015.

Reflection

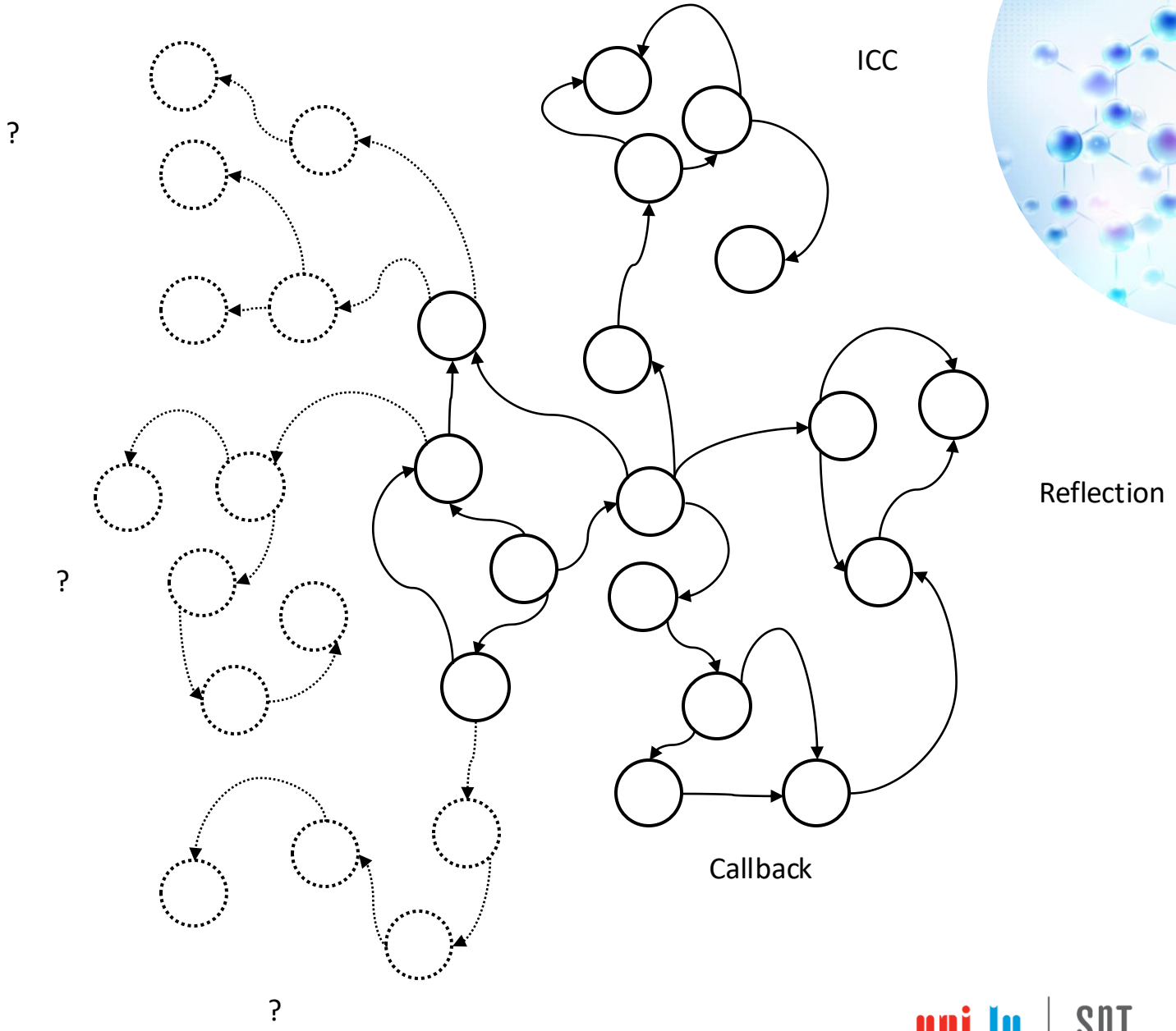
- Li, Li et al., Droidra: Taming reflection to support whole-program analysis of android apps. ISSTA 2016.
- Barros et al. Static analysis of implicit control flow: Resolving java reflection and android intents. ASE 2015.

Callback

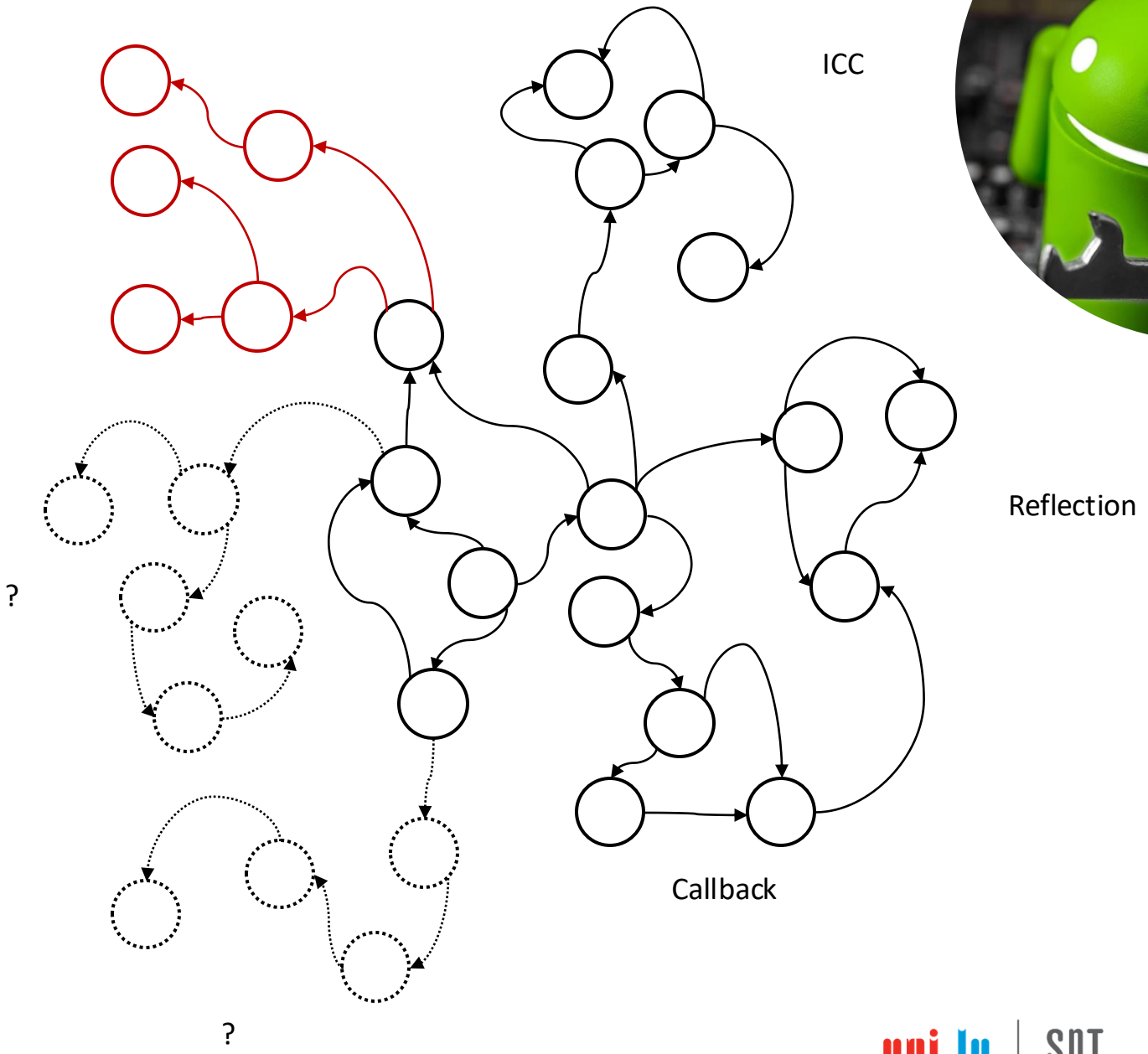
- Arzt et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. PLDI 2014.
- Yang et al. Static control-flow analysis of user-driven callbacks in Android applications. ICSE 2015.


ICC
?
Reflection
?
Callback
?

Random discoveries….



Contribution 1:
J. Samhi et al., "RAICC: Revealing Atypical Inter-Component Communication in Android apps", ICSE 2021.

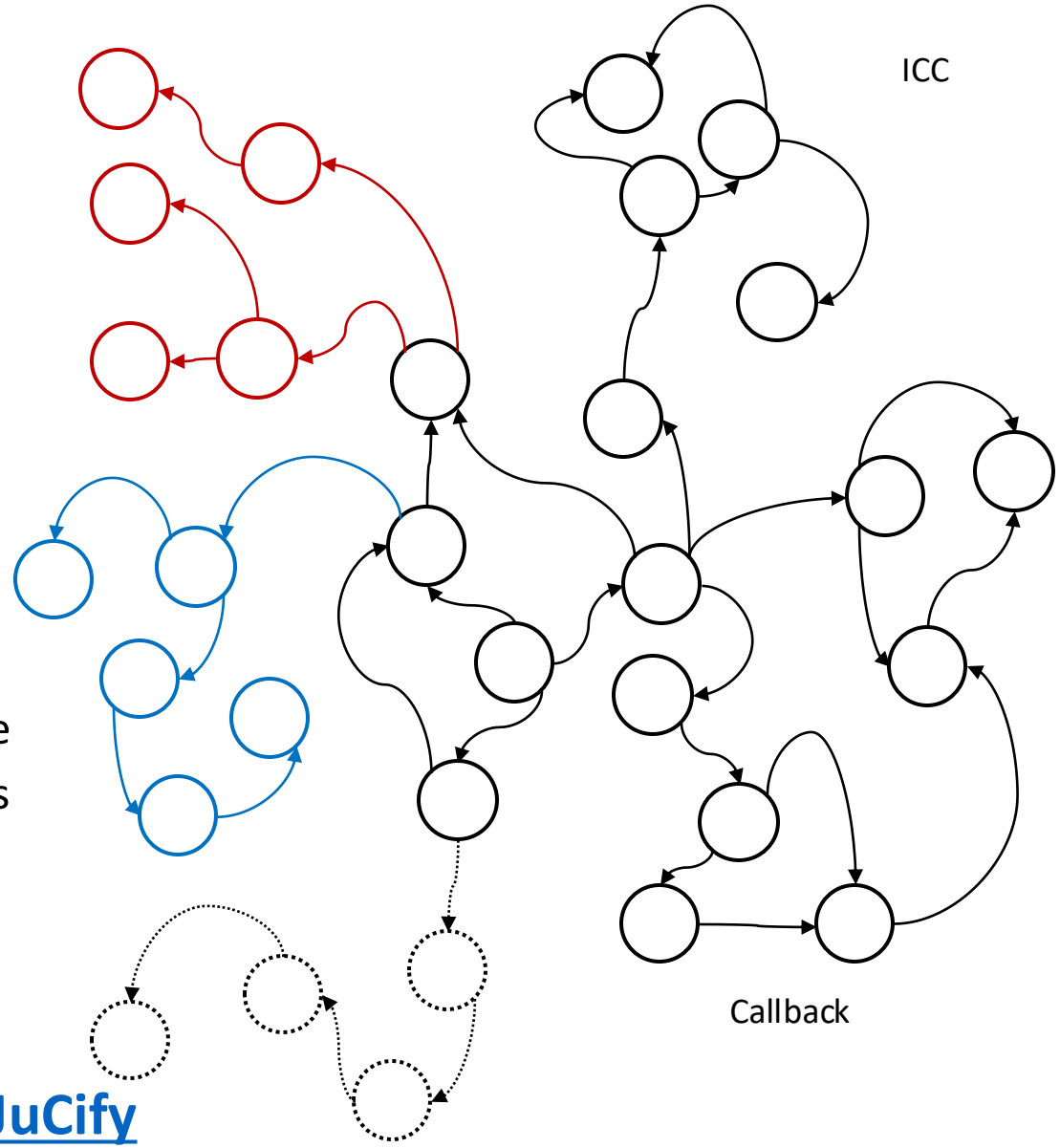


- RAICC improves ICC modeling
- It is is already used by collaborators
- It is maintained
- Improvable on-demand
- RAICC and artifacts are available at:

https://github.com/JordanSamhi/RAICC

Contribution 1:
J. Samhi et al., "RAICC: Revealing Atypical Inter-Component Communication in Android apps", ICSE 2021.

Contribution 2:
J. Samhi et al., "JuCify: A Step Towards Android Code Unification for Enhanced Static Analysis", ICSE 2022.



- We proposed a new approach to unify the bytecode and native code representations
- We demonstrated how JuCify is a step toward code unification
- JuCify and artifacts are available at:

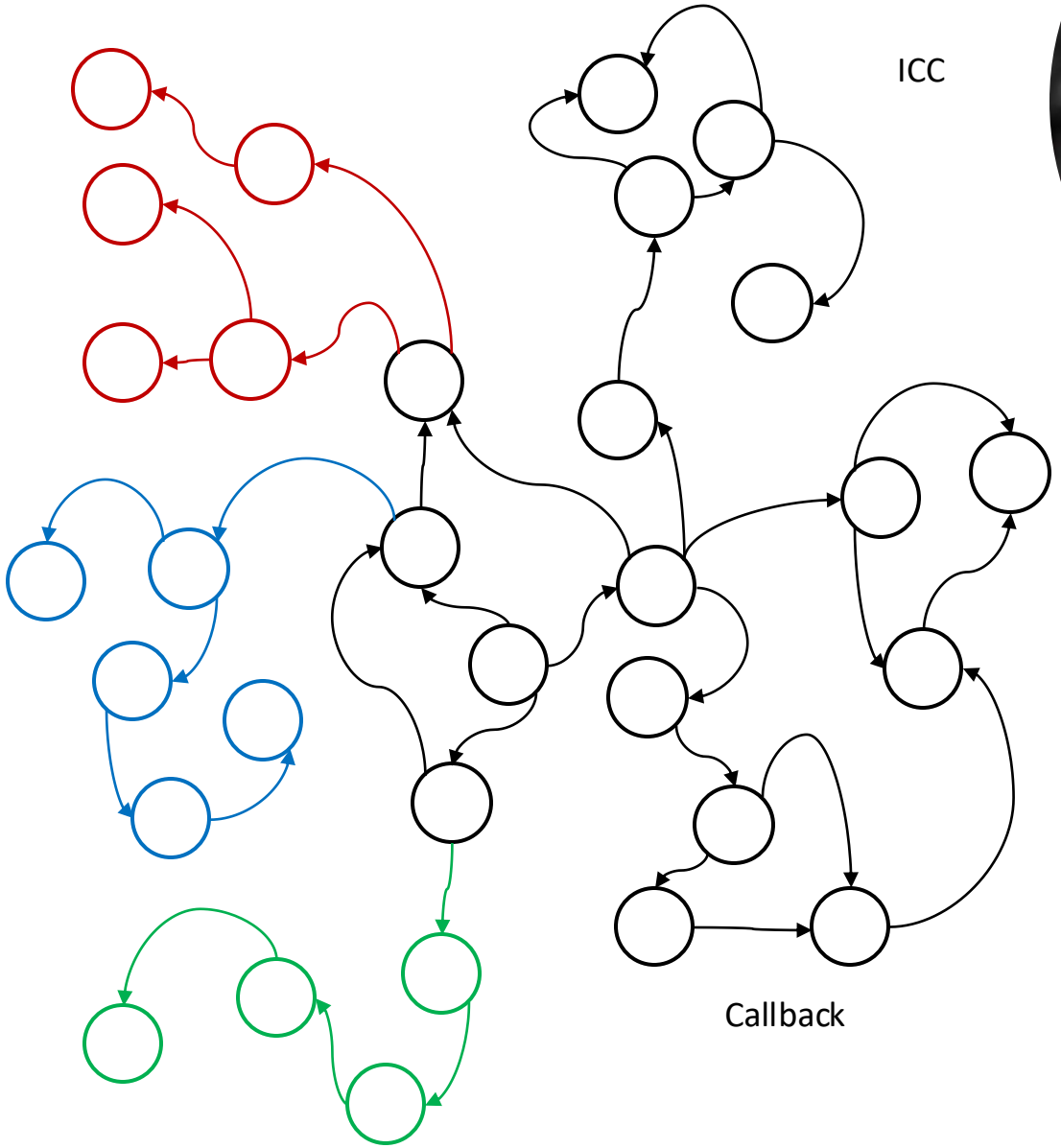
https://github.com/JordanSamhi/JuCify

Contribution 1:
J. Samhi et al., "RAICC: Revealing Atypical Inter-Component Communication in Android apps", ICSE 2021.

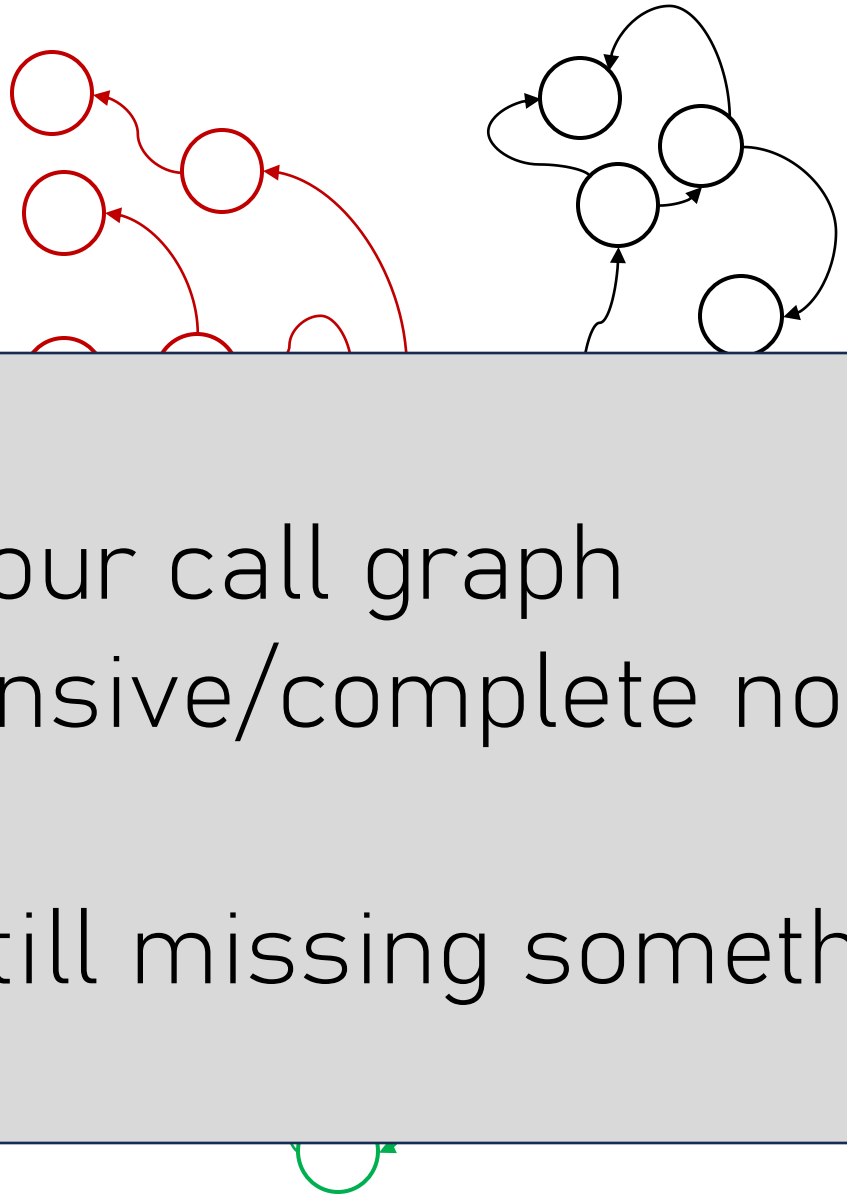
Contribution 2:
J. Samhi et al., "JuCify: A Step Towards Android Code Unification for Enhanced Static Analysis", ICSE 2022.

Contribution 3:
J. Samhi et al., "Archer: Resolving Conditional Implicit Calls in Android Apps", under submission

- We proposed a new approach for Conditional Implicit Calls
- We demonstrated how Archer improves static analysis
- We demonstrated how Archer aids dynamic analysis



Contribution 1:
J. Samhi et al., "RAICC: Revealing Atypical Inter-Component Communication in Android apps", ICSE 2021.



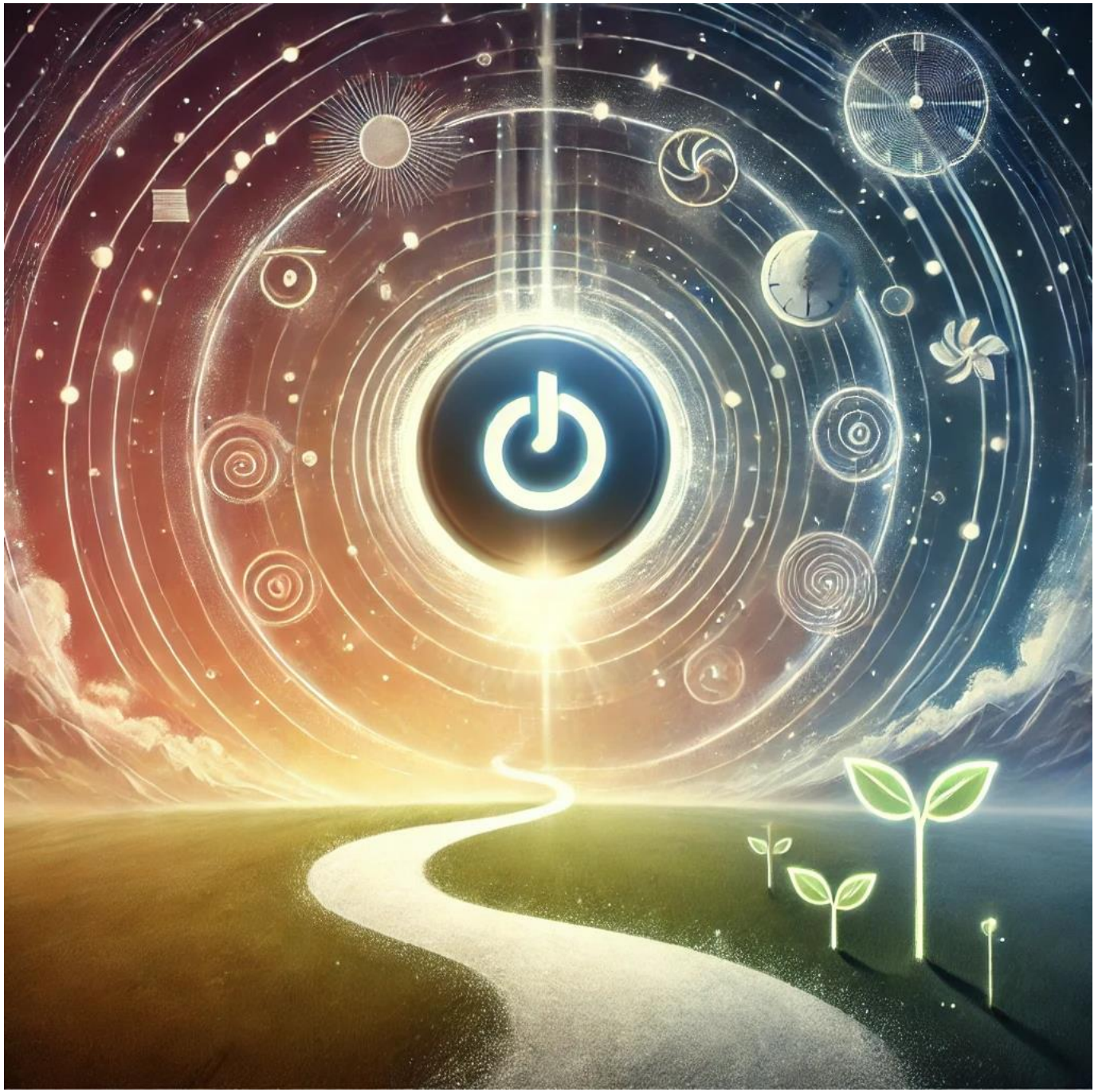
Contribution 2:
J. Samhi et al.
Code Unificat
ICSE 2022.

Contribution
J. Samhi et al.
Implicit Calls i
submission

Is our call graph comprehensive/complete now?

Or are we still missing something?

- We propo
  Implicit Ca
- We demo
  analysis
- We demonstrated how Archer aids dynamic analysis



Let's restart from the beginning

Two main techniques to analyse a program

1

Dynamic Analysis

2

Static Analysis

Dynamic Analysis

*"Dynamic analysis operates by **executing** a program and **observing** the executions"**

Dynamic analysis is precise!

*"Dynamic analysis is **precise** because no approximation or abstraction need be done"**

*Ernst, Michael D. "Static and dynamic analysis: Synergy and duality." WODA 2003: ICSE Workshop on Dynamic Analysis. 2003

Static Analysis

"Static analysis examines program code and reasons over **all possible behaviors** that might arise at run time"*

Static analysis is sound!

"Typically, static analysis is **conservative and sound**"*

"Soundness guarantees that analysis results are an **accurate description of the program's behavior**, no matter on what inputs or in what environment the program is run"*

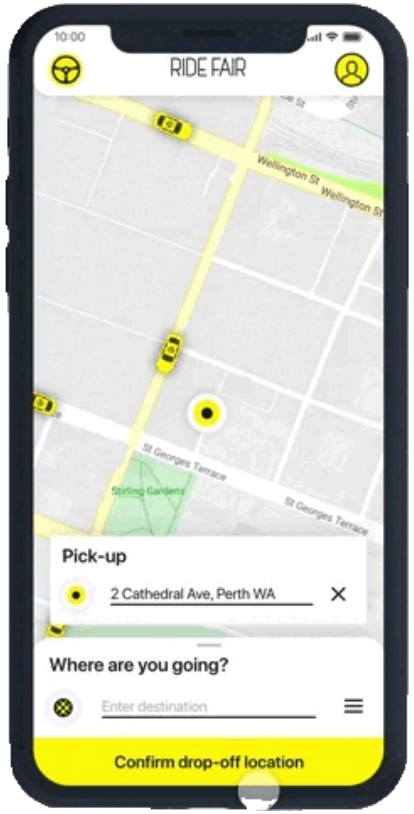
Is it?

*Ernst, Michael D. "Static and dynamic analysis: Synergy and duality." WODA 2003: ICSE Workshop on Dynamic Analysis. 2003

Objective

Measure and understand the level of **unsoundness** in Android static analysis tools

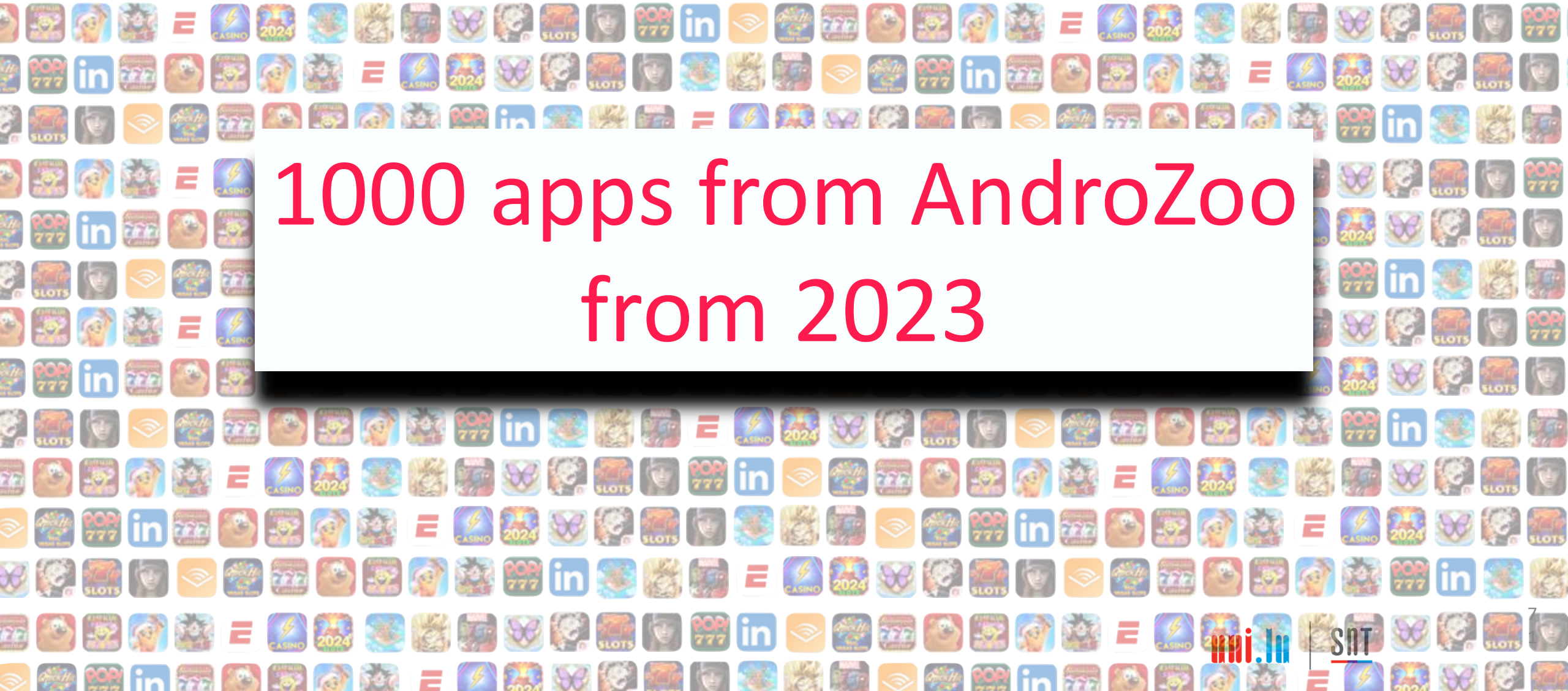
How?



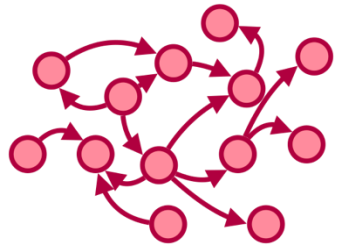
Dynamic Analysis

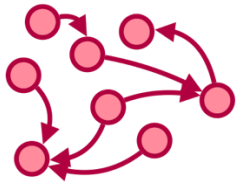
Static Analysis

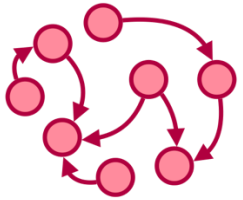
Dataset

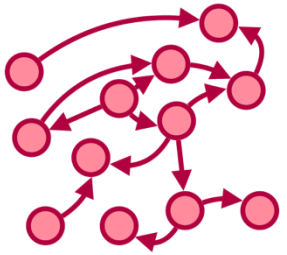


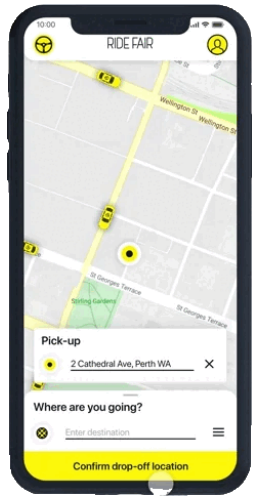
Dynamic Analysis



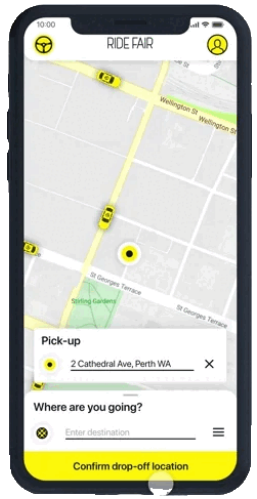








Dynamic Analysis



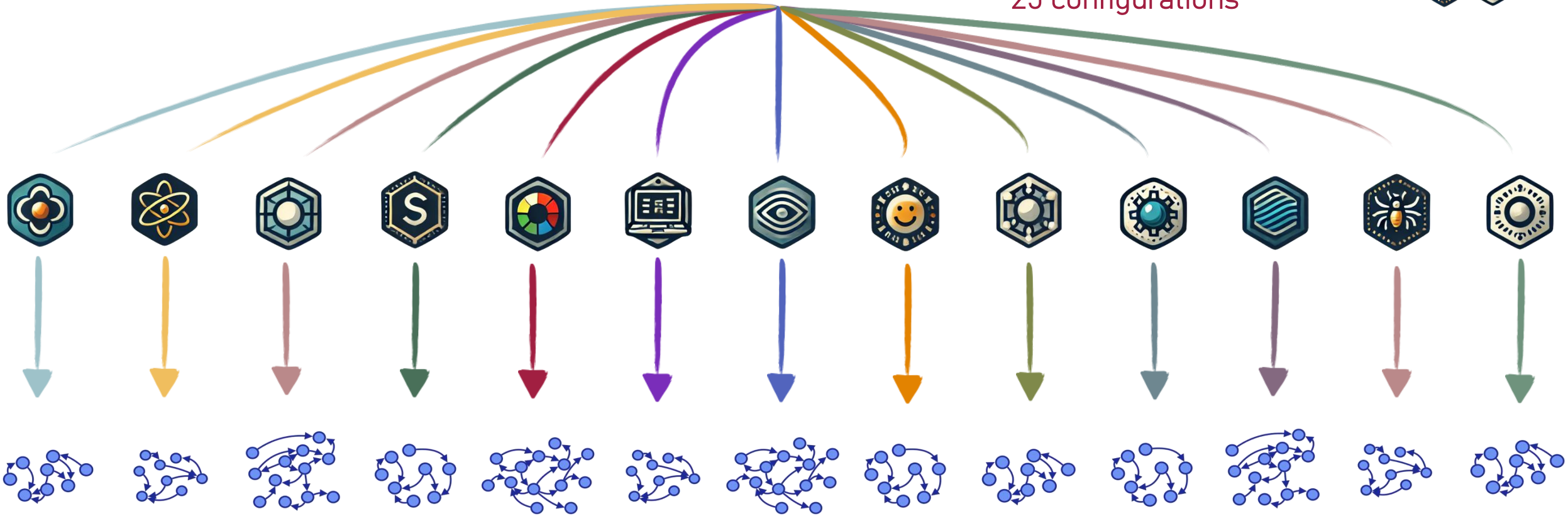
1000
call graphs

Average Code Coverage

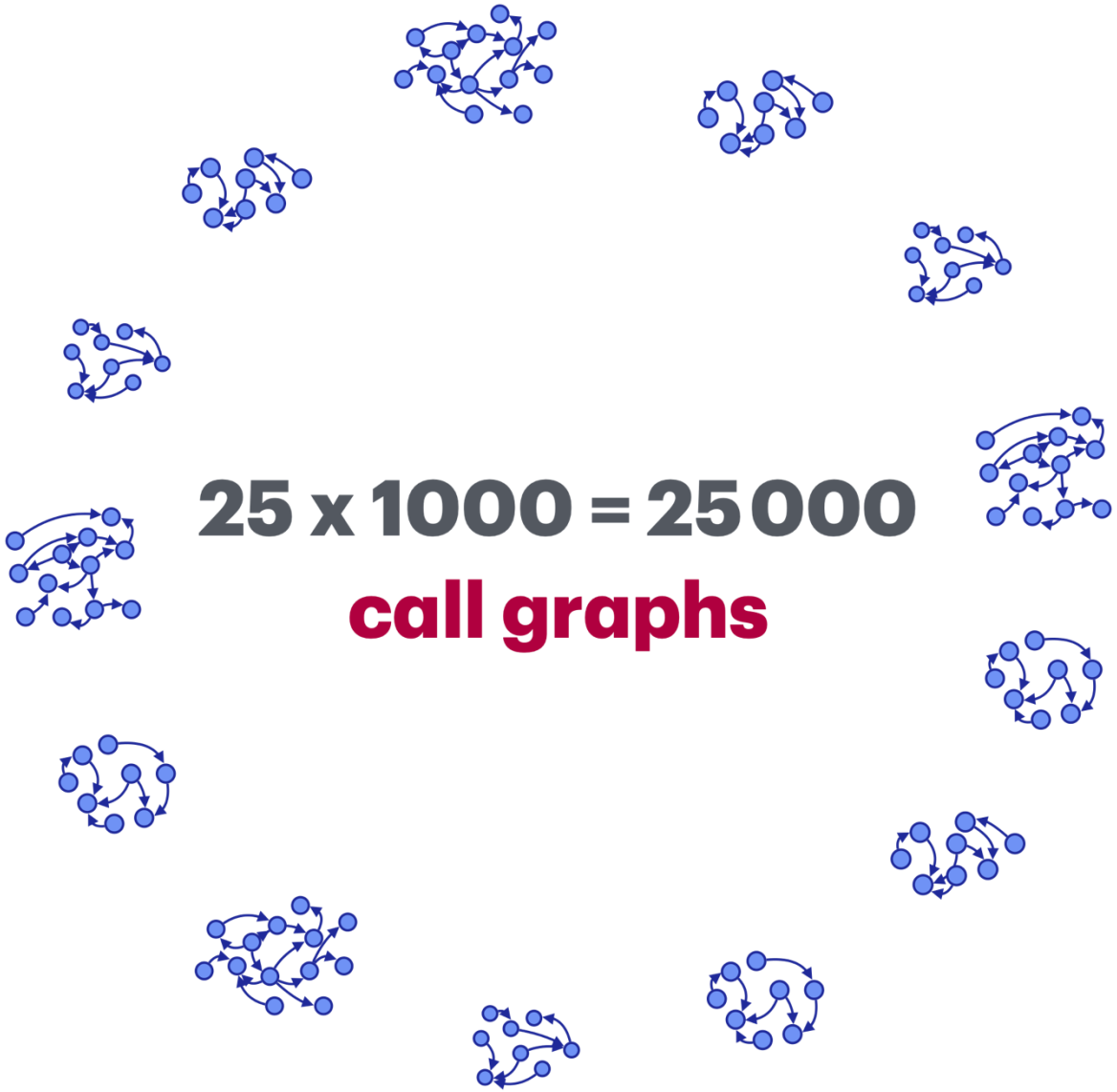
Static Analysis

Each app has been processed by a static analyzer:

When possible, we parametrized the call graph construction algorithm : 25 configurations



Static Analysis



Static Analysis

126

Apps successfully analyzed by all tools

25 x 126 = 3150

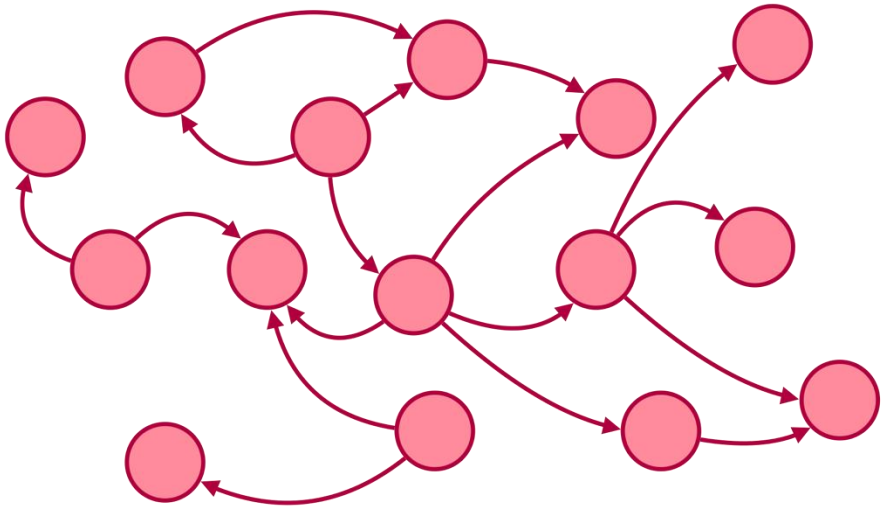
call graphs

		With libraries			Without libraries		
		Avg. $\lvert SM \rvert$	% M. in CG	Avg. $\lvert SE \rvert$	Avg. $\lvert SM^{\neg l} \rvert$	% M. in CG	Avg. $\lvert SE^{\neg l} \rvert$
FlowDroid	CHA	71 051	38%	399 975	6651	66%	48 218
	RTA	71 046	24%	227 493	6651	52%	33 802
	VTA	71 045	18%	109 519	6651	42%	16 788
	SPARK	71 031	5%	15 250	6649	12%	2391
IccTA	CHA	71 051	38%	399 981	6651	66%	48 220
	RTA	71 046	24%	227 541	6651	52%	33 746
	VTA	71 045	18%	109 023	6651	41%	16 703
	SPARK	71 031	5%	15 249	6649	12%	2391
RAICC	CHA	71 051	38%	397 791	6651	66%	47 894
	RTA	71 046	24%	224 574	6651	52%	33 271
	VTA	71 045	19%	111 151	6651	41%	16 605
	SPARK	71 031	6%	16 264	6650	12%	2434
DroidRA	CHA	71 053	38%	397 872	6652	66%	47 903
	RTA	71 048	24%	224 992	6652	52%	33 452
	VTA	71 047	19%	111 188	6652	42%	16 749
	SPARK	71 033	6%	16 437	6650	12%	2491
NatiDroid	CHA	61 758	81%	469 025	4837	88%	40 398
MaMaDroid	SPARK	60 500	5%	12 592	4791	14%	2007
BackDroid	SPARK	60 500	5%	12 592	4791	14%	2007
SootFX	SPARK	61 707	0%	101	4798	1%	9
ACID	SPARK	61 707	8%	54 169	4798	48%	4124
Gator	CHA	110 824	73%	1 920 412	31 342	90%	655 813
Jicer	SPARK	71 144	6%	15 763	6651	11%	2302
ArpDroid	SPARK	60 500	5%	12 593	4791	14%	2007
Difuzer	CHA	60 567	34%	245 987	4809	65%	31 060

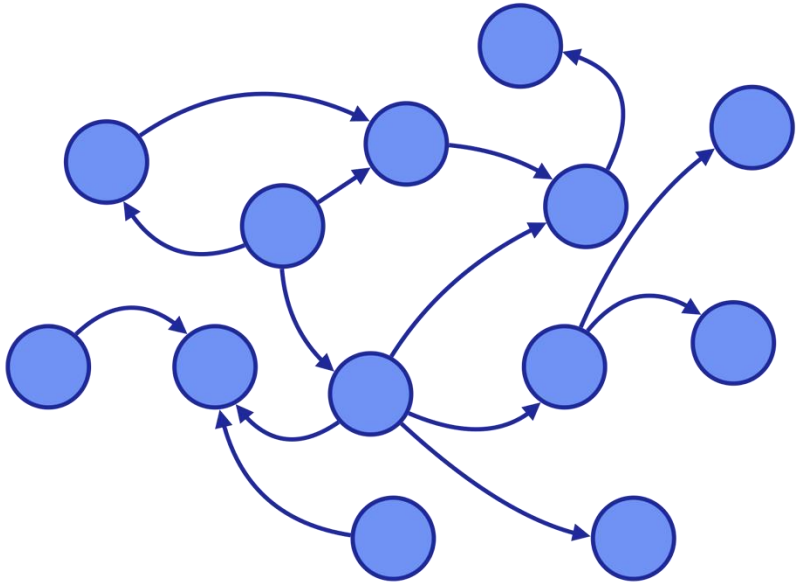
Comparison of Static Analysis Tools

- Tools find different numbers of methods in apps
- Some tools supposed to add edges have **fewer edges** than baselines
- More precise call graph algorithms lead to significantly **fewer edges** in the call graph
- The same call graph construction algorithm leads to **different call graphs**

Comparison



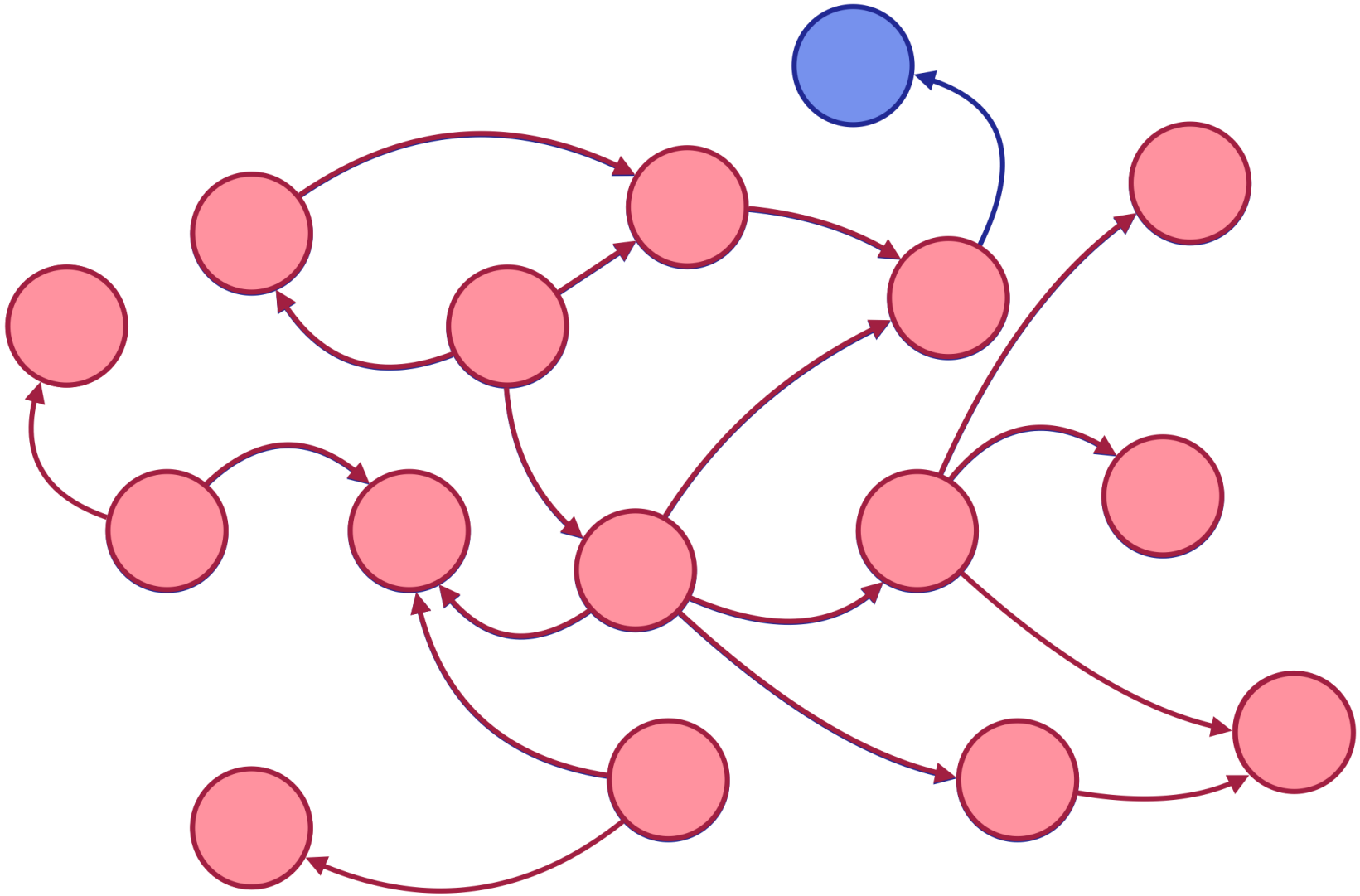
Dynamic Call Graph



Static Call Graph

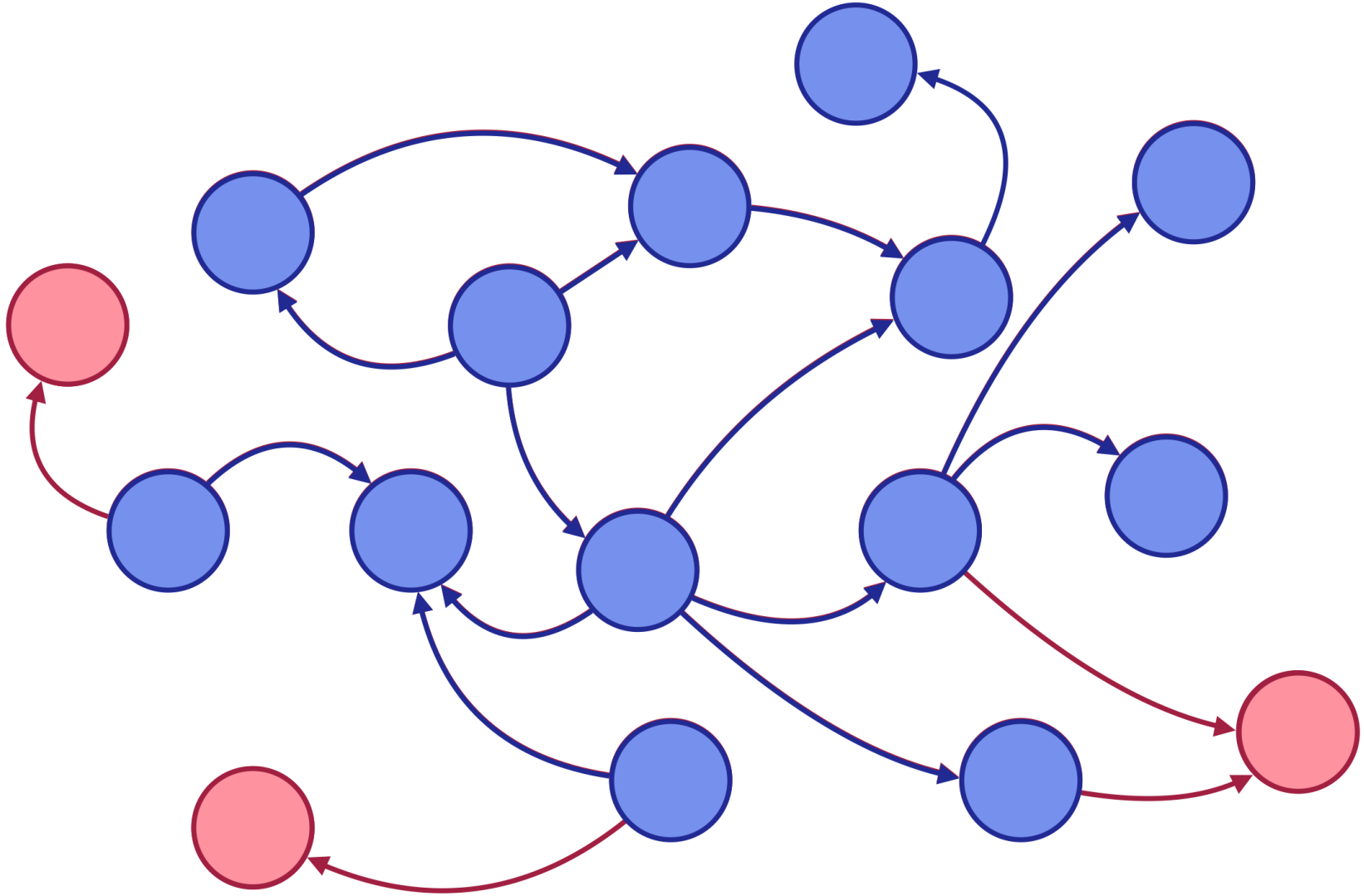
Comparison

The dynamic call Graph can miss some method calls (i.e., some nodes)
=> This is expected



Comparison

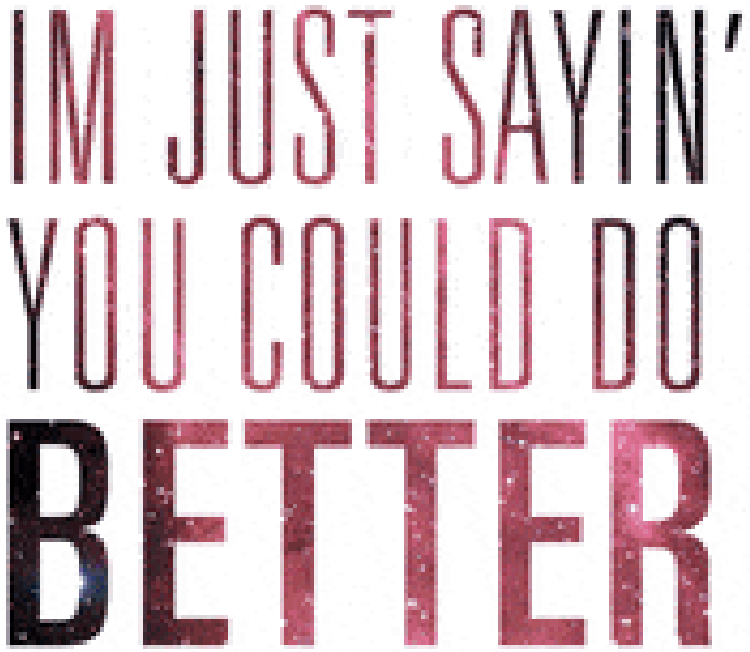
More interestingly, the static call Graph can miss some method calls
=> This is NOT expected



40%
methods missed with the
biggest over-approximation

Comparison of dynamic and static analysis

- More precise call graph
- The more precise an alg
- CHA-based tools have le
- Even if CHA is the bigges

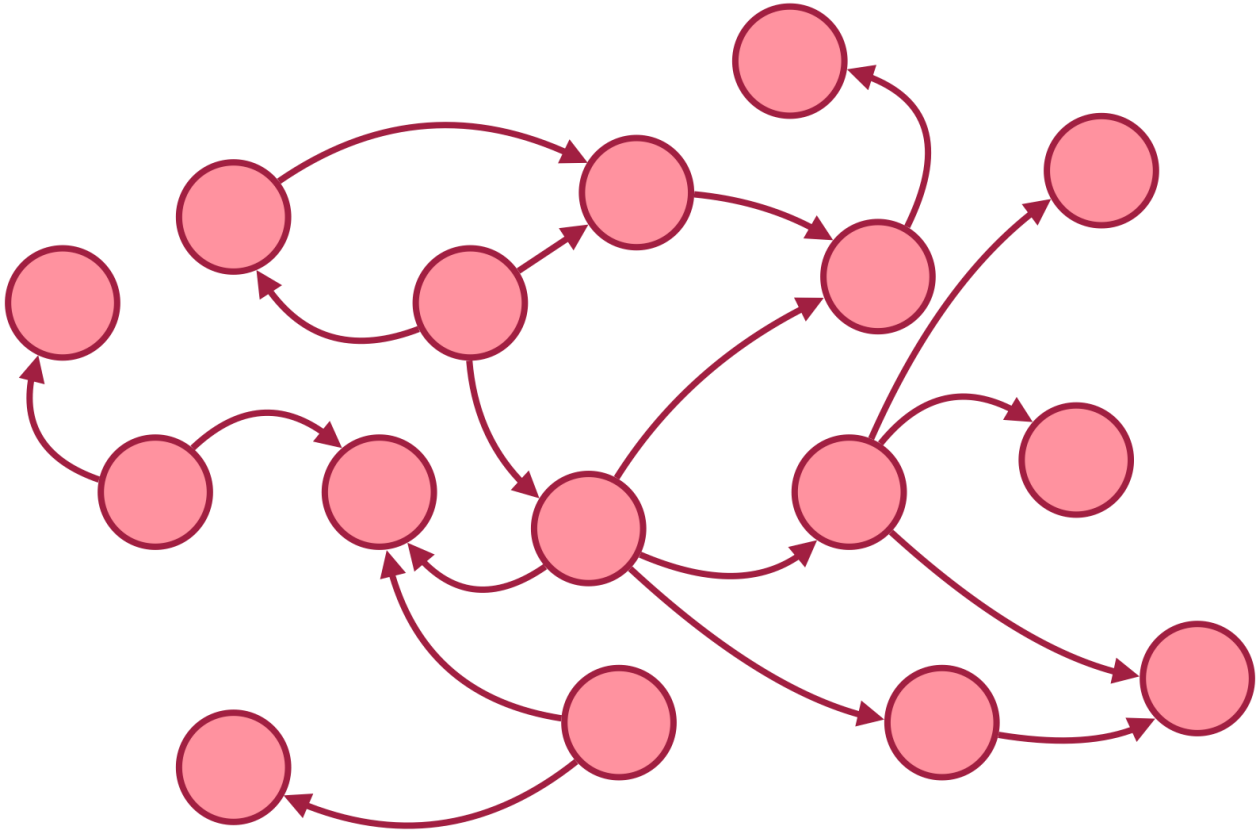


ir tasks

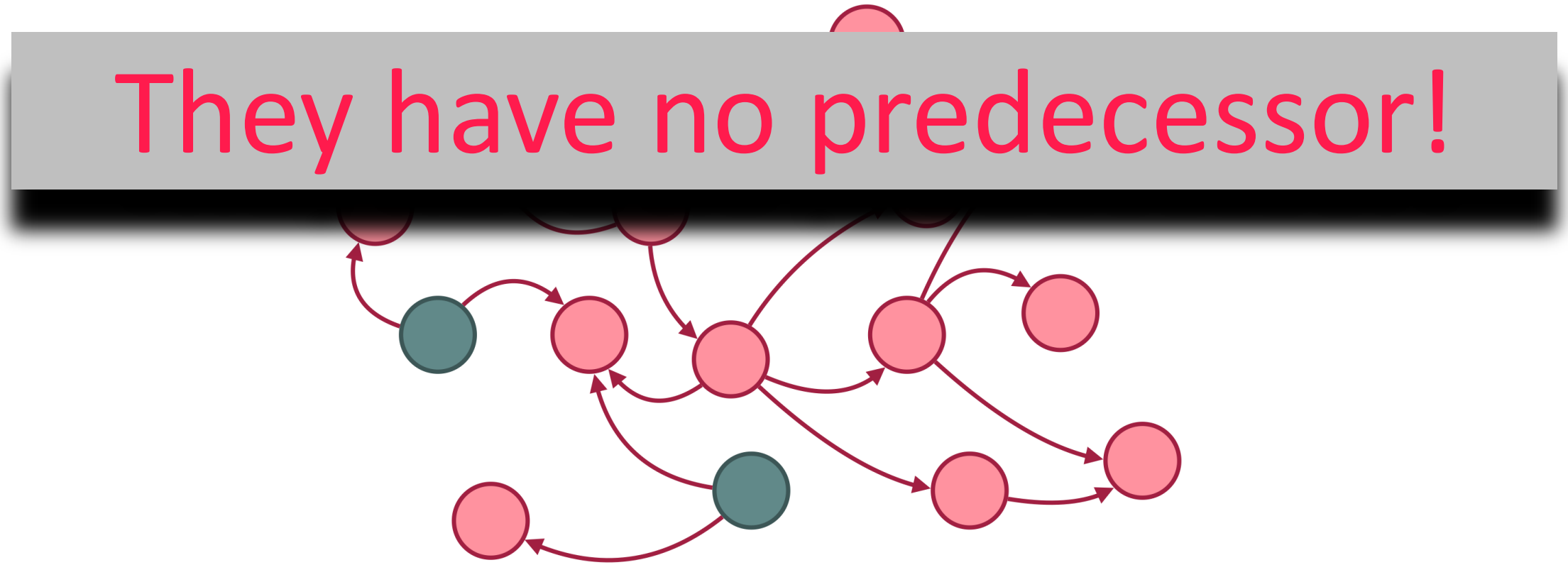
short

What is the cause of this **unsoundness**?

Remember the dynamic call graph?



Remember the dynamic call graph?



What do these nodes have in common?

We hypothesized that they are one of the main reasons for unsoundness

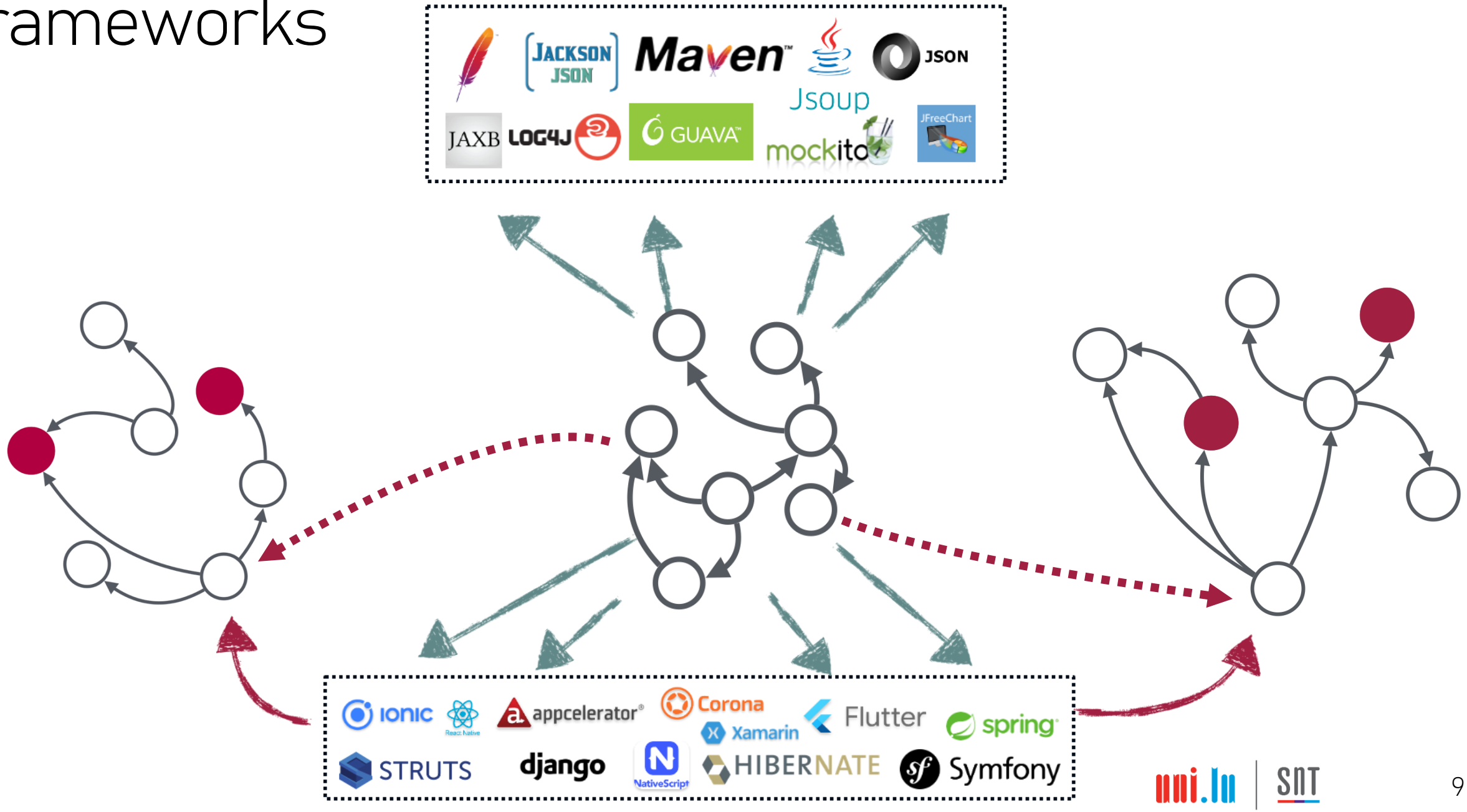
16%

of methods do not have a predecessor,
i.e., they are entrypoints

Causes of Unsoundness

- Many methods missed are derived from the Android framework methods
- Many methods missed are derived from framework methods, e.g., Google, Flutter, Ryanheise, or Unity3d

Frameworks


Jackson JSON
Maven
JSON
Jsoup
JAXB
LOG4J
GUAVA
mockito
JFreeChart
IONIC
React Native
appcelerator
Corona
Xamarin
Flutter
spring
STRUTS
django
NativeScript
HIBERNATE
Symfony

Causes of Unsoundness

- Many methods missed are derived from the Android framework methods
- Many methods missed are derived from framework methods, e.g., Google, Flutter, Ryanheise, or Unity3d
- All static analysis tools miss at least 35% of these entry points
- They represent 20% of all methods missed
- Constructors, obfuscated methods, and lifecycle methods are among the most missed methods

Implications for Security

Better Static Code Modeling

=

Better Static Code Coverage

Our study highlights many opportunities for future research and paves the way for improving the **soundness** of static analysis tools

Static analysis is NOT sound!

Agenda

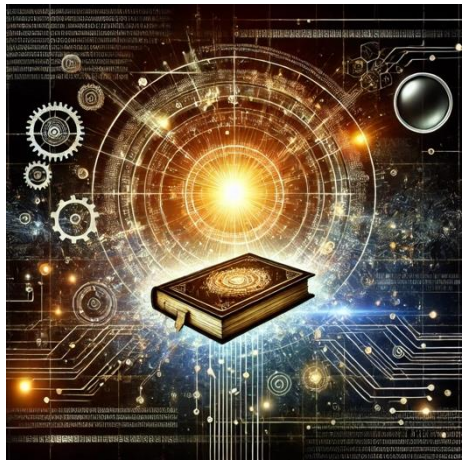
1

The need for a large set of Apps

AndroZoo

2

Static Analysis The Genesis



The Past

3

Static Analysis Soundness?

The Present

4

Better Analysis!

The Future

Using dynamic analysis to improve static analysis

Straightforward idea:

- Collect the entry point methods via dynamic analysis
- Feed these entry point methods to the static analyzer

Preliminary results:

- On 100 apps
- By dynamically analyzing the apps for 5 min each

	Average # of nodes	Median # of nodes
Without RD	50 626	25 899
With RD	65 534	46 307
	+29%	+79%

LLM for Mobile App Analysis

GUI Testing with LLMs

Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions

Zhe Liu[1,2],Chunyang Chen[3], Junjie Wang[1,2,*], Mengzhuo Chen[1,2], Boyu Wu[2,4],
Xing Che[1,2], Dandan Wang[1,2], Qing Wang[1,2,5,*]

Intent-Driven Mobile GUI Testing with Autonomous Large Language Model Agents

Juyeon Yoon
School of Computing

Robert Feldt
Dept. of Computer Science & Engineering

Shin Yoo
School of Computing

Daejeon, F
juyeon.y

Unblind Text Inputs: Predicting Hint-text of Text Input in Mobile Apps via LLM

Zhe Liu
Laboratory for Internet Software Technologies, University of Chinese Academy of Sciences, Institute of Software, Chinese Academy of Sciences, China and Laboratory for Internet Software Technologies, University of Chinese Academy of Sciences, Institute of Software, Chinese Academy of Sciences
China
liuzhe181@mails.ucas.edu.cn

Chunyang Chen
Technical University of Munich
Germany
chunyang.chen@monash.edu

Junjie Wang
University of Chinese Academy of Sciences, China and Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences
China
jujie@iscas.ac.cn

Mengzhuo Chen
University of Chinese Academy of Sciences, China and Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences
China
chenmengzhuo23@mails.ucas.edu.cn

Boyu Wu
University of Chinese Academy of Sciences, China and Institute of Software Chinese Academy of Sciences
China
boyu_wu2021@163.com

Yuekai Huang
University of Chinese Academy of Sciences
China
huangyuekai18@mails.ucas.ac.cn

Jun Hu
Laboratory for Internet Software Technologies, Institute of Software

Qing Wang
University of Chinese Academy of Sciences, China and Laboratory for

***Abstract*—GUI**
as expected whe
e.g., testing specif
scenarios. Curren
manual task since
adequacy metrics
coverage. We pro
agent for Androi
GUI testing. It is
mechanisms such
Android app, DR
quently tries to a
empirical evaluat
Themis benchmar
tasks, with a high
a messaging app,
added a first acc
without human in
61% activity cov
the-art GUI testi
that 317 out of th
and relevant to a
interacts deeply v
***Index Terms*—**
artificial intellige

Testing mobile

LLM for Static Analysis

An Empirical Study of Large Language Models for Type and Call Graph Analysis

2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering (Forge)

Ash
Ros
Mir

Rece

Abs
thei
this
anal
icall
moc
mar
wor
ing
we

Func
Nort

Ashv
Hein
E-ma

Rose
Pade
E-ma

Sam
Pade

The Emergence of Large Language Models in Static Analysis: A First Look through Micro-benchmarks

Ashwin Prasad Shivarpatna Venkatesh[§], Samkutty Sabu[¶], Amir M. Mir[‡], Sofia Reis[†], Eric Bodden[**]
[§] *ashwin.prasad@upb.de*, Heinz Nixdorf Institut, Paderborn University, Paderborn, Germany

Can Large Language Models Reason about Program Invariants?

Kexin Pei [1 2] **David Bieber** [2] **Kensen Shi** [2] **Charles Sutton** [2] **Pengcheng Yin** [2]

Abstract

Identifying invariants is an important program analysis task with applications towards program understanding, bug finding, vulnerability analysis, and formal verification. Existing tools for identifying program invariants rely on dynamic analysis, requiring traces collected from multiple executions in order to produce reliable invariants. We study the application of large language models to invariant prediction, finding that models trained on source code and fine-tuned for invariant generation can perform invariant prediction as static rather than dynamic analysis. Using a scratchpad approach where invariants are predicted sequentially through a program gives the best performance, finding invariants statically of quality comparable to those obtained by a dynamic anal-

has proved challenging even for simple programs.

In the programming languages literature, one of the most important insights is to reason at the level of *abstractions* of program states, e.g., the property "is n >= 1 when line 12 executes?", rather than *concrete states*, such as "n = 17 at line 12". This has been a fundamental insight from some of the earliest proposals to formalize program semantics (Hoare, 1969; Dijkstra, 1975). This move has computational advantages, because abstracting away details can simplify the analysis, but it is also representational, because the analysis task is often to check over all plausible inputs rather than specific concrete inputs.

If a program property is always true at a given program point, it is an *invariant*, which abstracts multiple program states by finding a common pattern that is easier to reason about. Identifying invariants is undecidable, so previous work has con-

AndroZoo for Large Scale Empirical Studies

Let's start with a simple question

Do you know what is inside an Android App?

Data Leaks

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

Data Leaks for Android Apps
FlowDroid
[PLDI'14]

Our study highlights many opportunities for future research and paves the way for improving the **soundness** of static analysis tools

Static analysis is NOT sound!

LLM for Mobile App Analysis

GUI Testing with LLMs

Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions

Zhe Liu[1,2],Chunyang Chen[3], Junjie Wang[1,2,*], Mengzhuo Chen[1,2], Boyu Wu[2,4], Xing Che[1,2], Dandan Wang[1,2], Qing Wang[1,2,5,*]

Intent-Driven Mobile GUI Testing with Autonomous Large Language Model Agents

Juyeon Yoon, *School of Computing*; Robert Feldt, *Dept. of Computer Science & Engineering*; Shin Yoo, *School of Computing*

Unblind Text Inputs: Predicting Hint-text of Text Input in Mobile Apps via LLM

Zhe Liu, Laboratory for Internet Software Technologies, University of Chinese Academy of Sciences, Institute of Software, Chinese Academy of Sciences, China and Laboratory for Internet Software Technologies, University of Chinese Academy of Sciences, Institute of Software, Chinese Academy of Sciences, China, liuzhe181@mails.ucas.edu.cn

Chunyang Chen, Technical University of Munich, Germany, chunyang.chen@monash.edu

Junjie Wang, University of Chinese Academy of Sciences, China and Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, China, jujie@iscas.ac.cn

Mengzhuo Chen, University of Chinese Academy of Sciences, China and Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, China, chenmengzhuo23@mails.ucas.edu.cn

Boyu Wu, University of Chinese Academy of Sciences, China and Institute of Software Chinese Academy of Sciences, China, boyu_wu2021@163.com

Yuekai Huang, University of Chinese Academy of Sciences, China, huangyuekai18@mails.ucas.ac.cn

Jun Hu, Laboratory for Internet Software Technologies, Institute of Software

Qing Wang, University of Chinese Academy of Sciences, China and Laboratory for

LLM for Static Analysis

An Empirical Study of Large Language Models for Type and Call Graph Analysis

2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering (Forge)

The Emergence of Large Language Models in Static Analysis: A First Look through Micro-benchmarks

Ashwin Prasad Shivarpatna Venkatesh[§], Samkutty Sabu[¶], Amir M. Mir[‡], Sofia Reis[†], Eric Bodden[**]
[§]*ashwin.prasad@upb.de*, Heinz Nixdorf Institut, Paderborn University, Paderborn, Germany

Can Large Language Models Reason about Program Invariants?

Kexin Pei[1 2] David Bieber[2] Kensen Shi[2] Charles Sutton[2] Pengcheng Yin[2]

Abstract

Identifying invariants is an important program analysis task with applications towards program understanding, bug finding, vulnerability analysis, and formal verification. Existing tools for identifying program invariants rely on dynamic analysis, requiring traces collected from multiple executions in order to produce reliable invariants. We study the application of large language models to invariant prediction, finding that models trained on source code and fine-tuned for invariant generation can perform invariant prediction as static rather than dynamic analysis. Using a scratchpad approach where invariants are predicted sequentially through a program gives the best performance, finding invariants statically of quality comparable to those obtained by a dynamic anal-

has proved challenging even for simple programs.

In the programming languages literature, one of the most important insights is to reason at the level of *abstractions* of program states, e.g., the property "is n >= 1 when line 12 executes?", rather than *concrete states*, such as "n = 17 at line 12". This has been a fundamental insight from some of the earliest proposals to formalize program semantics (Hoare, 1969; Dijkstra, 1975). This move has computational advantages, because abstracting away details can simplify the analysis, but it is also representational, because the analysis task is often to check over all plausible inputs rather than specific concrete inputs.

If a program property is always true at a given program point, it is an *invariant*, which abstracts multiple program states by finding a common pattern that is easier to reason about. Identifying invariants is undecidable, so previous work has con-

Thank You!

Beautiful Chongqing!