LiveDroid: Identifying and Preserving Mobile App State in Volatile Runtime Environments

Umar Farooq, Zhijia Zhao, Manu Sridharan, and Iulian Neamtiu*

University of California, Riverside, *New Jersey Institute of Technology

Volatile Runtime Environment

- Unlike traditional applications, mobile apps suffer **frequent restarts**

Scenario 1: Runtime Config. Changes

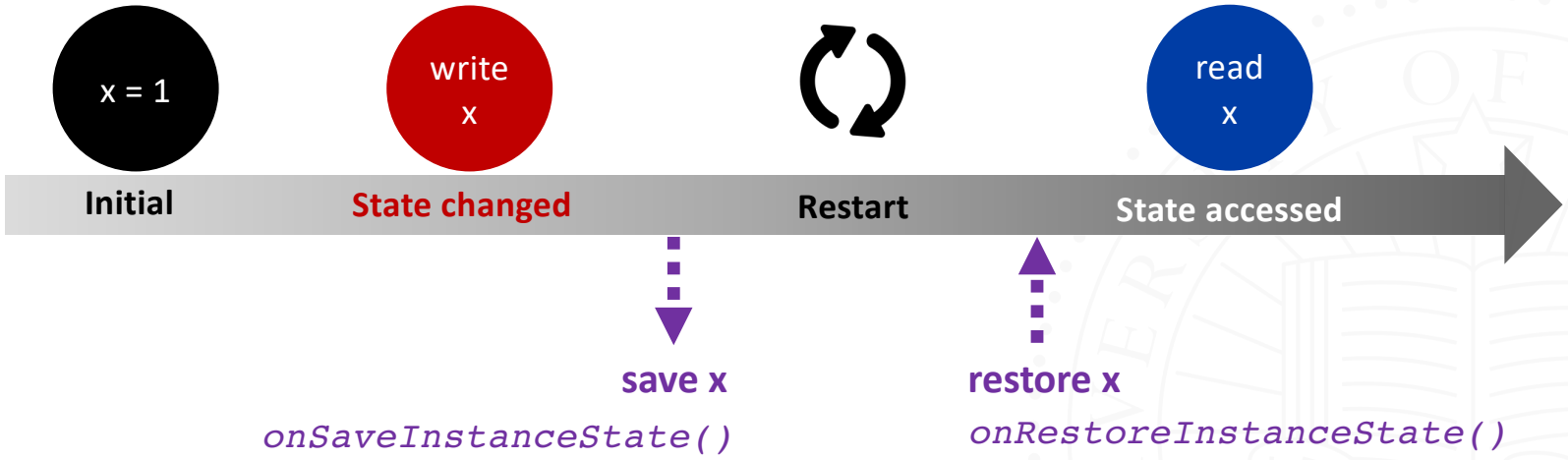
- rotate phone
- attach keyboard
- change language
- …

Current **activity** is **destroyed**

Scenario 2: High Memory Pressure

The whole **app** (process) is **killed**

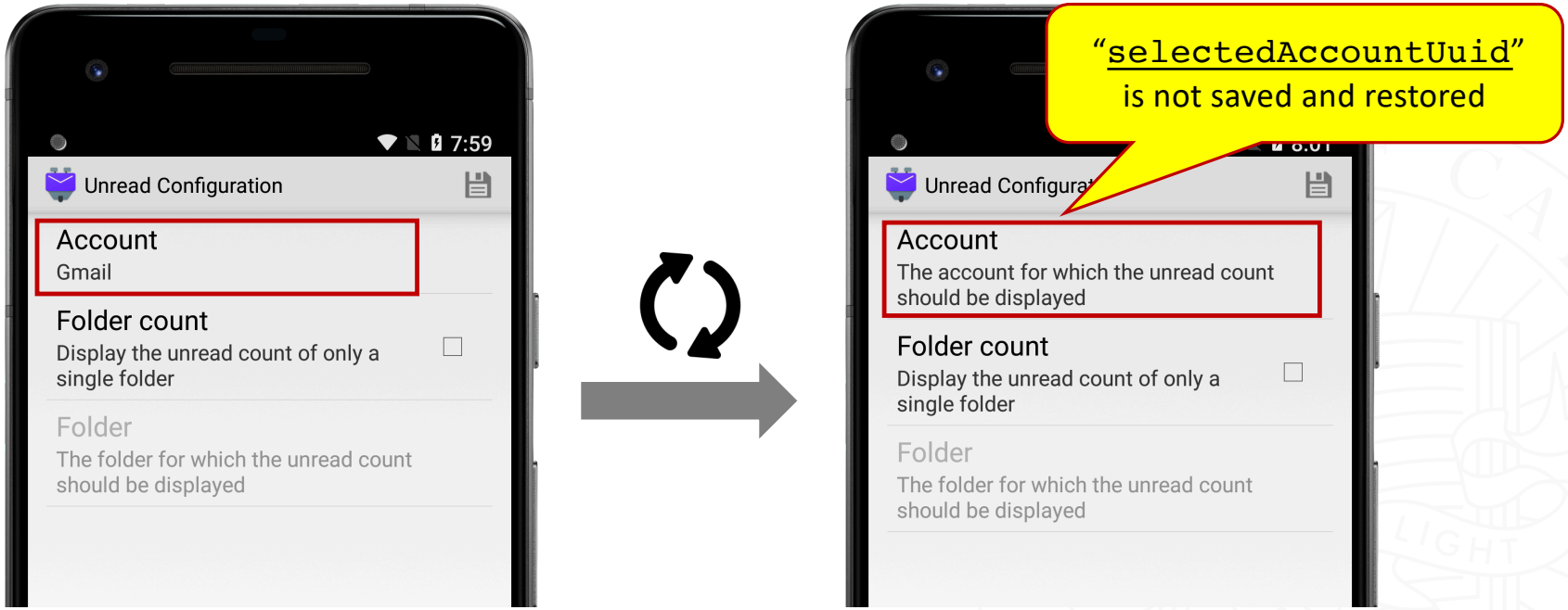
Volatile Runtime Environment



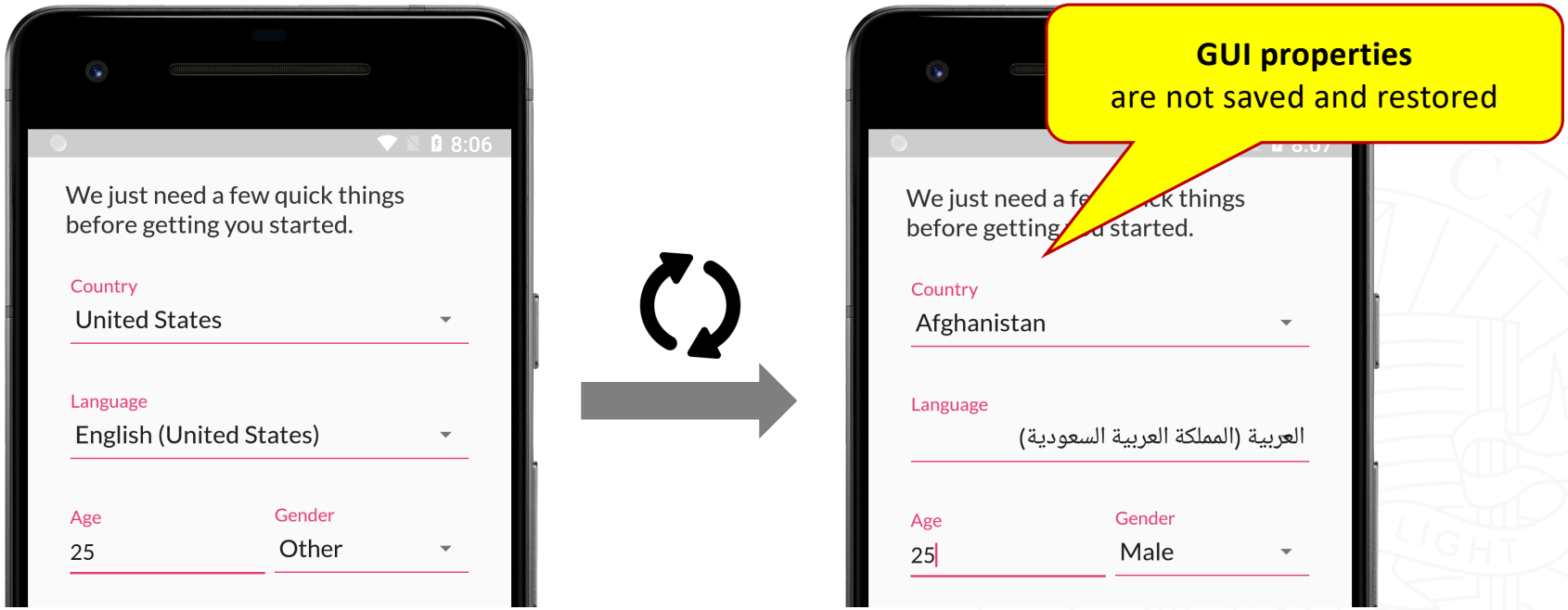
State Issues: *state lost*, *unresponsiveness*, *UI malfunctioning*, *app crashes*

[MobiSys'18 by Farooq etl.]

Example Issue: Selected Account Lost



Example Issue: User selections lost



Existing Work

[OOPSLA'16 by Shan et. al]

- Save and restore *all **mutable** activity fields; **ignore** GUI elements*
- Detect *inconsistent* data saving and restoring

[MobiSys'18 by Farooq et. al]

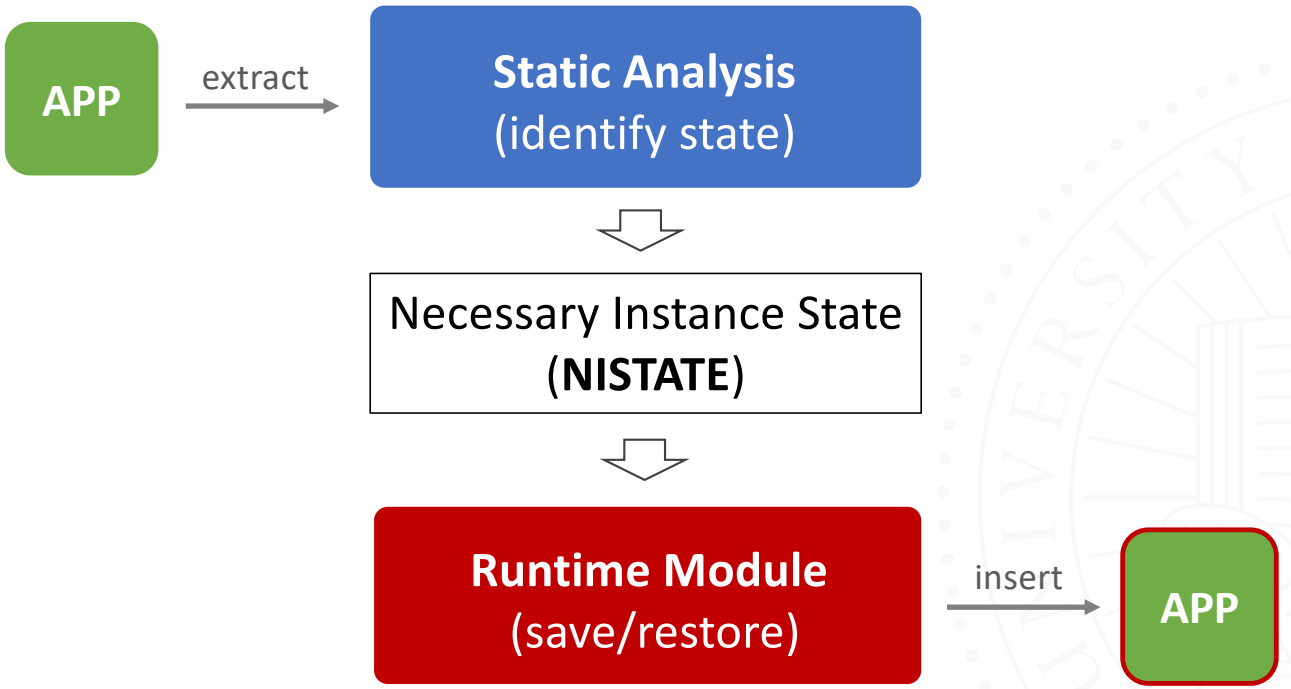
- *Prevent activity restarting* during configuration changes
- App ***may still get restarted*** due to ***low memory***

Open Questions:

Q1: How to identify the ***necessary state*** that needs to preserve?

Q2: How to ***automatically*** save and restore the state?

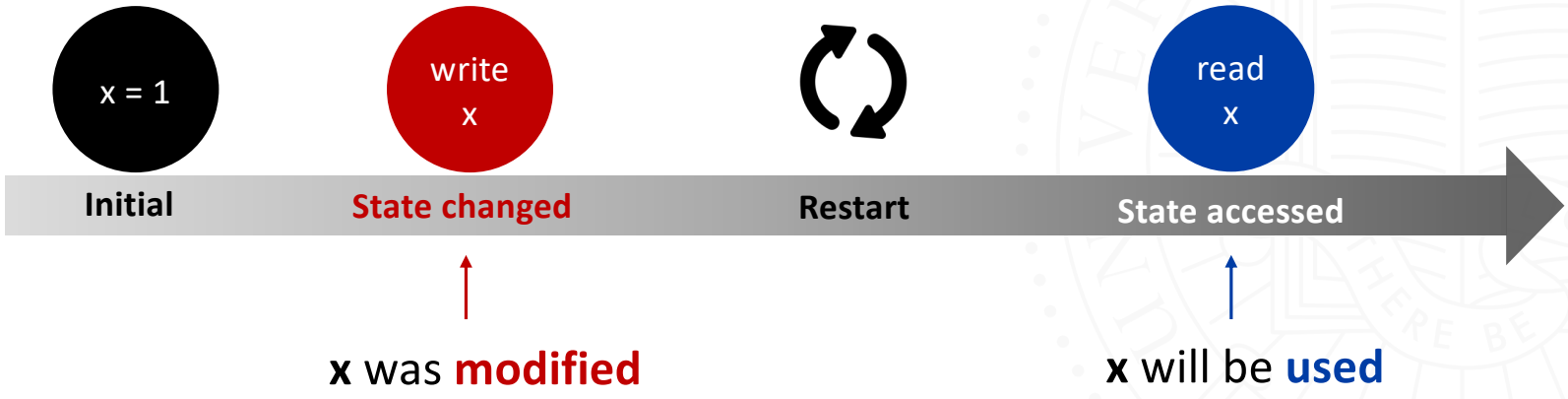
This Work


APP
extract
Static Analysis
(identify state)
Necessary Instance State
(NISTATE)
Runtime Module
(save/restore)
insert
APP

Necessary Instance State (NISTATE)

- **NISTATE**: *data that are **necessary** to preserve to maintain the feeling that the app is **always running***

- Two Key Requirements:



Static NISTATE Analysis

What variables may be modified?

```
void onClick1() {
  …
  this.v = this.u + 1;
}
```

What variables will be used?

```
void onClick2() {
  this.w = this.x + 2;
  this.y = this.v + this.w;
}
```

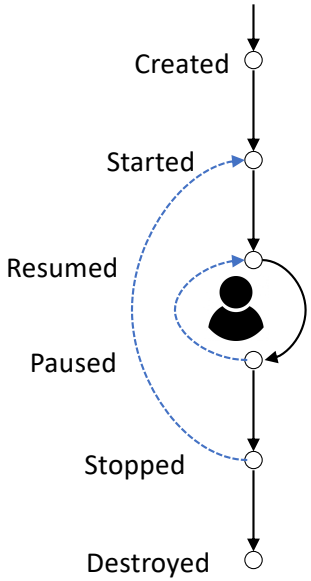
May-Modify Analysis

Entry-Liveness Analysis

NISTATE = MOD ⋂ LIVE

{this.v} = {this.v} ⋂ {this.x, this.v}

Analysis Details: *Callbacks Modeling*



Pre-Interaction onCreate(), onStart(), onResume()	skip
Interaction onClick(), onCheckedChanged(), onLocationChanged(), ...	**MOD** and **LIVE**
Post Interaction onPause(), onStop(), onDestroy()	**LIVE**

Past

Future

(a) lifecycle (b) callback modeling (c) static analysis

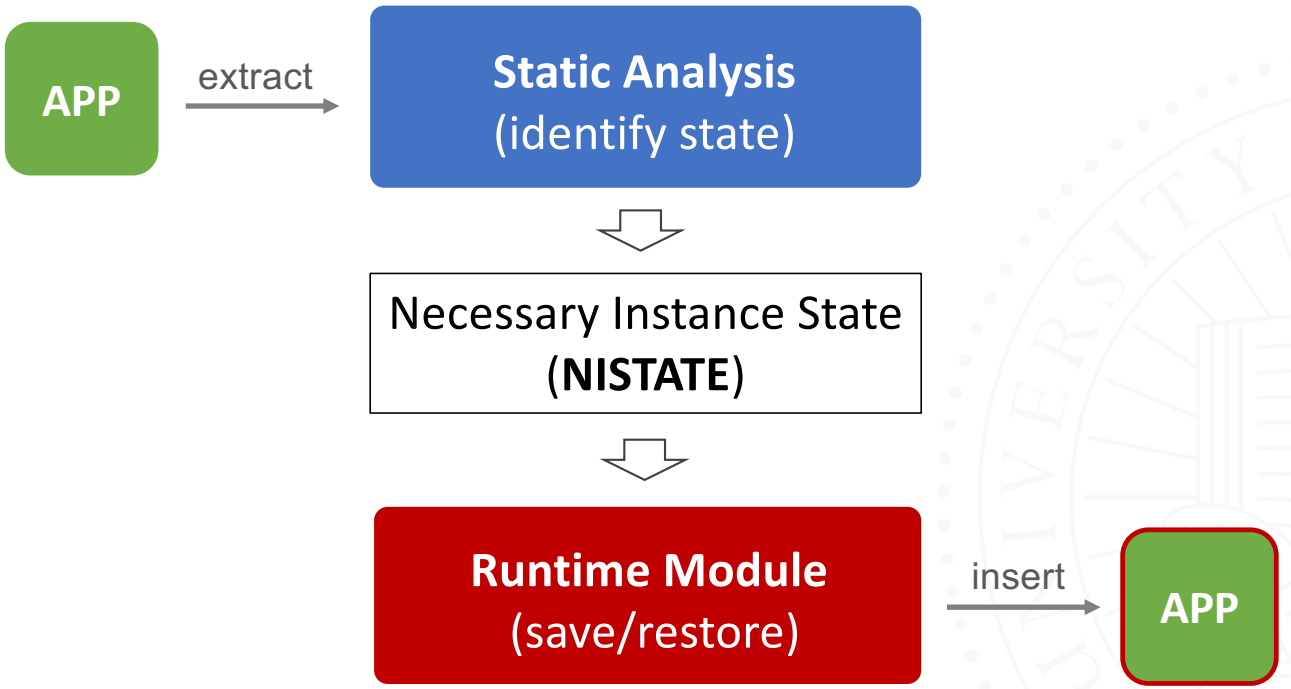
Analysis Details: *Complexities*

- GUI elements defined in layout files (XML) **UI Property Analysis**
  - unlike Java variable, user can **"read"** and "**modify**" GUI elements **directly**
  - referred to as **"External State"**

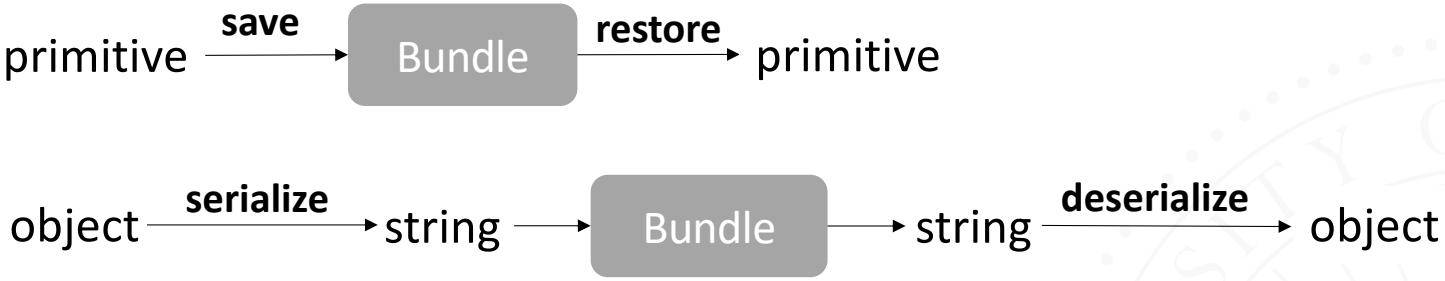
- Aliases **Alias Analysis** + **Runtime Checking** *(Alias Grounding)*
  - need to preserve for correctness e.g., `if(this.a == this.b)`
  - duplicate saving and restoring

- Field/object sensitivity

This Work


APP
extract
Static Analysis
(identify state)
Necessary Instance State
(NISTATE)
Runtime Module
(save/restore)
insert
APP

Runtime: Save/Restore NISTATE




```
//Save primitive
s.putInt("int_x", this.x);

//Restore primitive
this.x = s.getInt("int_x");
```

```
//Save object
s.putString("obj_b", gson.toJson(this.b));

//Restore object
String str = s.getString("obj_b");
this.b = gson.fromJson(str, B.class);
```

Runtime: Alias Grounding

- Verifies **statically identified** aliases for correctness
- **Less data** to save/restore

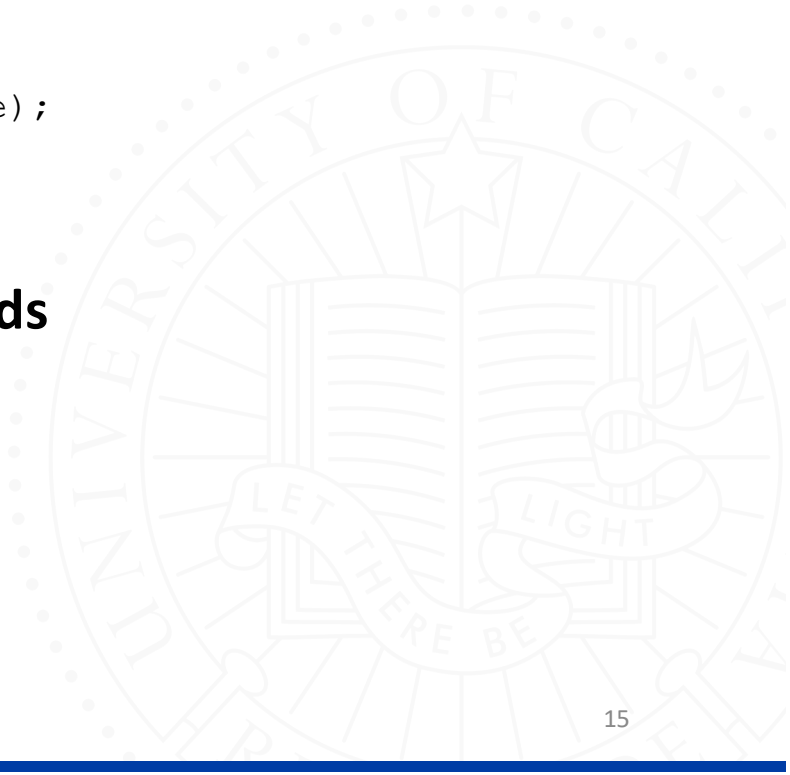
```
//Save alias
if(this.a.b == this.b)
    s.putBoolean("a_b=b", true); //save alias
else
    s.putString("obj_a_b", gson.toJson(this.a.b));  //save object
```

Runtime: Other Complexities

- Save/restore **GUI properties**

```
//Save GUI elements
TextView view = findViewById(R.id.text_time);
s.putString("text_time", view.getText());
```

- Save/restore **parital objects** and **private fields**



Implementation

- **Static Analyzer**
  - Built on Soot and FlowDroid: ***backward*** inter-procedural analysis
- **IDE Plugin**
  - Android Studio Plugin, and takes in NISTATE (in XML report)
  - Interactively generates code as developers direct
- **Patching Tool**
  - Soot-based binary patching tool, takes in NISTATE (in XML report)
  - Automatically injects code

Evaluation: Methodology

- Two Groups of Android Apps (from F-Droid/Github)

	Group-L	Group-S
# Apps	**966**	**36**
# Activities	**4,808**	**469**

K-9, **LeafPic**, **RDP Remote Desktop**, etc.
(highly popular ones)

- Devices: PC (3.5GHz CPU and 16 GB RAM)
  Nexus 5X (Android 8.0)

Evaluation: Results

Applicability (Group-L)

- 452 apps (**46.8%**) with non-empty **external state**
- 322 apps (**33.2%**) with non-empty **internal state**

Efficiency & Effectiveness (Group-S)

- Analysis finishes **within 1 minute** (30/36)
- Internal state is only **1.5%** of [OOPSLA'16]
  - Reduces state saving by **16.6X** on average
  - Reduces state restoring by **9.5X** on average

Evaluation: Results

State Issues (Group-S)
• **231/393** access paths (primitives/objects) are not saved/restored in **21 apps** • Contributated to **46 new state issues** • **7.9% false positives** (due to unrealizable paths, coarse-grained UI analysis, etc.) • All the new issues are **fixed** by LiveDroid

Takeaway

- States of mobile apps are **destroyed** in volatile runtime environments
  - Developers required to find the app state and save/restore it

- This work defines **necessary instance state (NISTATE)**
- designs and develops **LiveDroid**:
  - **Statically** identifies NISTATE
  - Automatically save and restore NISTATE at **Runtime**

- Github: https://github.com/ucr-riple/LiveDroid