香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Characterizing and Detecting Performance Bugs for Smartphone Applications

Yepang Liu[1], Chang Xu[2], and S.C. Cheung[1]

[1]The Hong Kong University of Science and Technology

[2]State Key Lab for Novel Software Technology, Nanjing University

Smartphone Era

1 million+ apps

Apps for different purposes

App performance is critical

11,108 of *60,000* Android apps randomly sampled from Google Play suffer from *performance bugs!*

App performance is critical

App performance is critical

App performance is critical

Bad performance → User complaints → Market failure

Assuring good performance is NOT easy

- Small team
- No dedicated QA

Assuring good performance is NOT easy

- Small team
- No dedicated QA

- Limited bug understanding
- Lack of useful tool support

Assuring good performance is NOT easy

- Small team
- No dedicated QA

- Limited bug understanding
- Lack of useful tool support

- Fierce competition
- Short time to market

How can we help?

Understanding performance bugs

Designing performance assurance tools

Overview

- Empirical study: understanding performance bug
  - Research questions and study design
  - Empirical findings and implications
- PerfChecker: a performance bug detection tool
  - Tool design and implementation
  - Detected bugs and developers' feedback

Overview

- Empirical study: understanding performance bug
  - Research questions and study design
  - Empirical findings and implications
- PerfChecker: a performance bug detection tool
  - Tool design and implementation
  - Detected bugs and developers' feedback

Research questions

- RQ1: Bug types and impact
- RQ2: Bug manifestation
- RQ3: Debugging and bug-fixing effort
- RQ4: Common bug patterns

Application and bug selection

8 *popular Android apps with* well-maintained bug tracking system and code repository

Selected apps

Application name	Category	Size (LOC)	Downloads
Firefox	Communication	122.9K	10M ~ 50M
Chrome	Communication	77.3K	50M ~ 100M
AnkiDroid	Education	44.8K	500K ~ 1M
K-9 Mail	Communication	76.2K	1M ~ 5M
My Tracks	Health & Fitness	27.1K	10M ~ 50M
c:geo	Entertainment	44.7K	1M ~ 5M
Open GPS Tracker	Travel & Local	18.1K	100K ~ 500K
Zmanim	Books & Reference	5.0K	10K ~ 50K

Application and bug selection

8 *popular Android apps with* well-maintained bug tracking system and code repository

70 fixed performance bugs labeled by original developers

Empirical study process

70 Performance bugs
Bug reports, comments
Bug fixing patches
Patch reviews
Revision commit logs …

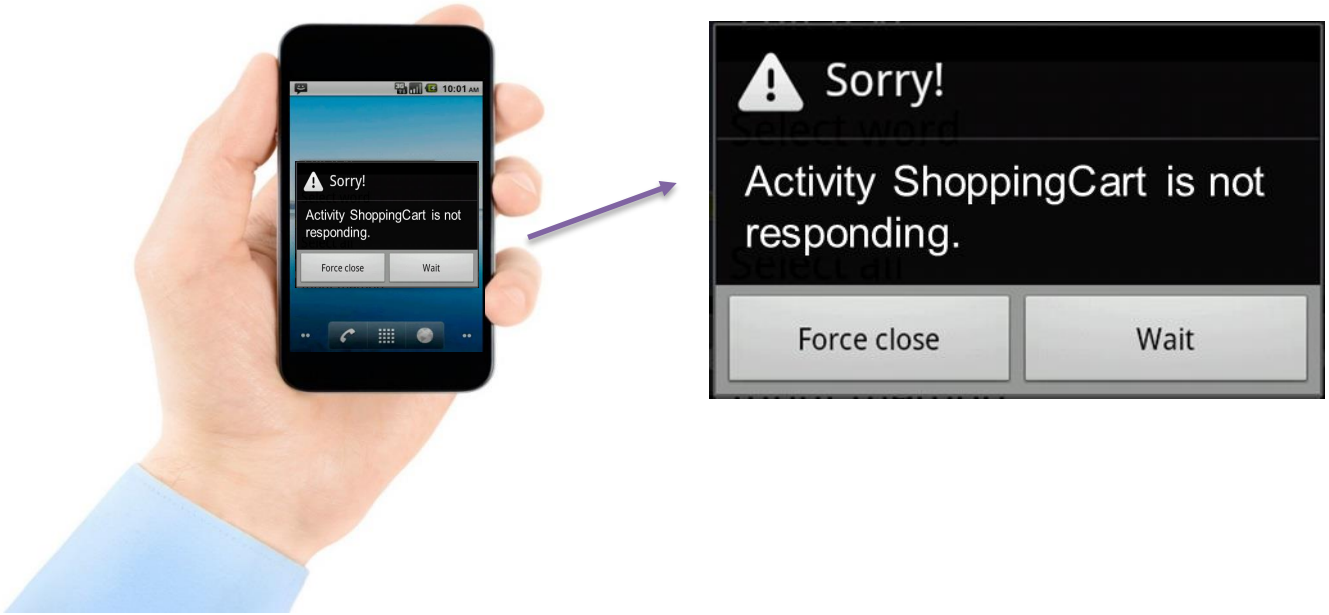
Research questions
Bug types and impacts
Bug manifestation
Debugging and fixing effort
Common bug patterns

Overview

- Empirical study: understanding performance bug
  - Research questions and study design
  - Empirical findings and implications
- PerfChecker: a static performance analysis tool
  - Analysis tool design and features
  - Detected bugs and developers' feedback

Three dominant bug types

1. **GUI lagging:** Significantly reducing the responsiveness and smoothness of an application's GUI (75.7%)



Three dominant bug types

1. **GUI lagging:** Significantly reducing the responsiveness and smoothness of an application's GUI (75.7%)

*"Switching tabs is **too slow**, sometimes can take **5 – 10 seconds**, triggering **Application Not Responding error**."* (Firefox bug 719493)

Three dominant bug types

2. **Energy leak:** Applications quickly and silently consume much battery power (14.3%)

Three dominant bug types

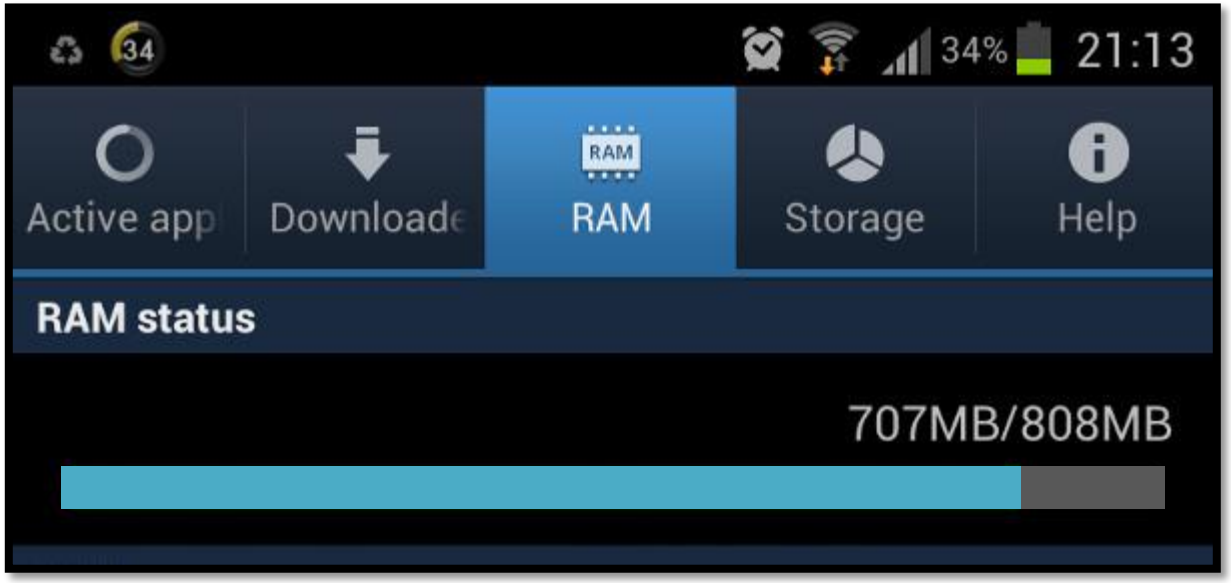
2. **Energy leak:** Applications quickly and silently consume much battery power (14.3%)

*"My Tracks is a massive battery drain. Battery lost **10%** in **standby** just **20 minutes** after a full charge."*

*"It is **destroying my battery**. I will have to **uninstall** it if there isn't a fix soon."* (My Tracks bug 520)

Three dominant bug types

3. **Memory bloat:** Applications consume significantly more memory than necessary (11.4%)



Three dominant bug types

3. **Memory bloat:** Applications consume significantly more memory than necessary (11.4%)

*"I went to **a few websites**, played **< 10 minutes**, **5.6 MB** memory is consumed … causing **crashes on Galaxy S4**."* (Chrome bug 245782)

Three dominant bug types

3. **Memory bloat:** Applications consume significantly more memory than necessary (11.4%)

*"I went to **a few websites**, played **< 10 minutes**, **5.6 MB** memory is consumed … causing **crashes on Galaxy S4**."* (Chrome bug 245782)

GUI lagging, energy leak and memory bloat are **three dominant types** in our studied performance bugs.

Manifesting performance bugs

Observation 1: Special user interactions needed to expose performance bugs (25 / 70)

Manifesting performance bugs

Zmanim energy leak reproducing steps

- Step 1: Switch on GPS
- Step 2: Configure Zmanim to use current location
- Step 3: Start Zmanim's main activity
- Step 4: Press "Home" button when GPS is acquiring a location

Manifesting performance bugs

Zmanim energy leak reproducing steps

- Step 1: Switch on GPS
- Step 2: Configure Zmanim to use current location
- Step 3: Start Zmanim's main activity
- Step 4: Press "Home" button when GPS is acquiring a location

These bugs make performance testing difficult ☹

How to trigger performance bugs?

Zmanim energy leak reproducing steps

- Step 1: Switch on GPS
- Step 2: Configure Zmanim to use current location
- Step 3: Start Zmanim's main activity
- Step 4: Press "Home" button when GPS is acquiring a location

Sequence

How to trigger performance bugs?

Zmanim energy leak reproducing steps

- Step 1: Switch on GPS
- Step 2: Configure Zmanim to use current location
- Step 3: Start Zmanim's main activity
- Step 4: Press "Home" button when GPS is acquiring a location

Sequence + Timing (app state)

Performance oracles

Observation 2: No well-defined performance oracle

- Performance bugs rarely cause fail-stop consequences

Performance oracles

Observation 2: No well-defined performance oracle

- Performance bugs rarely cause fail-stop consequences

How do developers judge performance degradation in reality?

Performance oracles

Three common judgment strategies in practice:

- Manual judgment

Performance oracles

Three common judgment strategies in practice:

- Manual judgment
- Product comparison

K-9 Mail

vs.

Gmail

Performance oracles

Three common judgment strategies in practice:

- Manual judgment
- Product comparison
- Developers' consensus

"Generally, ***100 ms*** *is the threshold beyond which users will perceive slowness in application."*

Performance oracles

Three common judgment strategies in practice:

- Manual judgment (manual effort)
- Product comparison (manual effort)
- Developers' consensus (not well defined)

Automated and well-defined oracles are desirable to facilitate performance testing and analysis.

Performance oracles

Three common judgment strategies in practice:

- Manual judgment (manual effort)
- Product comparison (manual effort)
- Developers' consensus (not well defined)

General oracles may not exist. Bug specific oracles are still helpful.
(Zhang et al. CODES+ISSS'12, Liu et al. PerCom'13, TSE'14)

Common bug patterns

Observation: More than one third of performance bugs are amenable to automated detection.

We observed three common bug patterns:

- Long running operations in main threads
- Wasted computation for invisible GUI
- Inefficient callbacks (frequently invoked)

1. Long running operations in main threads

How to keep your applications responsive?

"Android applications normally run entirely on a single thread. By default, it is the *UI thread* or *main thread*, which drives the user interface event loop. Any method that runs in the main thread should do as little work as possible." (Android documentation)

1. Long running operations in main threads

Long running operations will prevent the main thread from handling user events timely

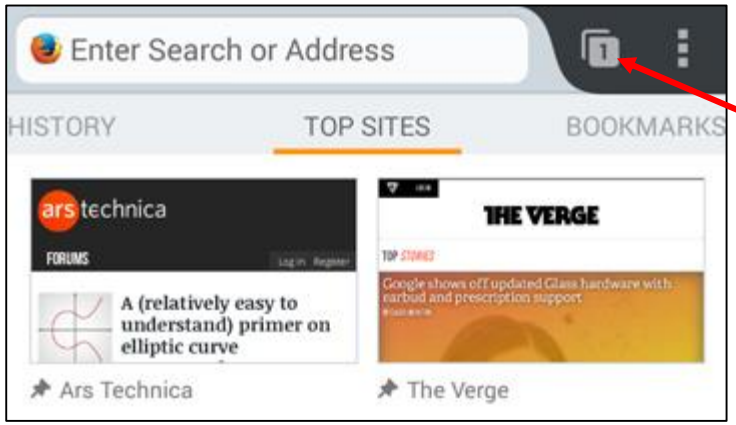
2. Wasted computation for invisible GUI

A dilemma:

- One great feature of Android is multitasking
- Potential drawback: bad apps conducting useless computation in background can eat up precious battery life

"*I have noticed there are a few people who have phones where there is software running in the background that just sort of exhausts the battery quickly.*" (Larry Page, Google's co-founder)

2. Wasted computation for invisible GUI

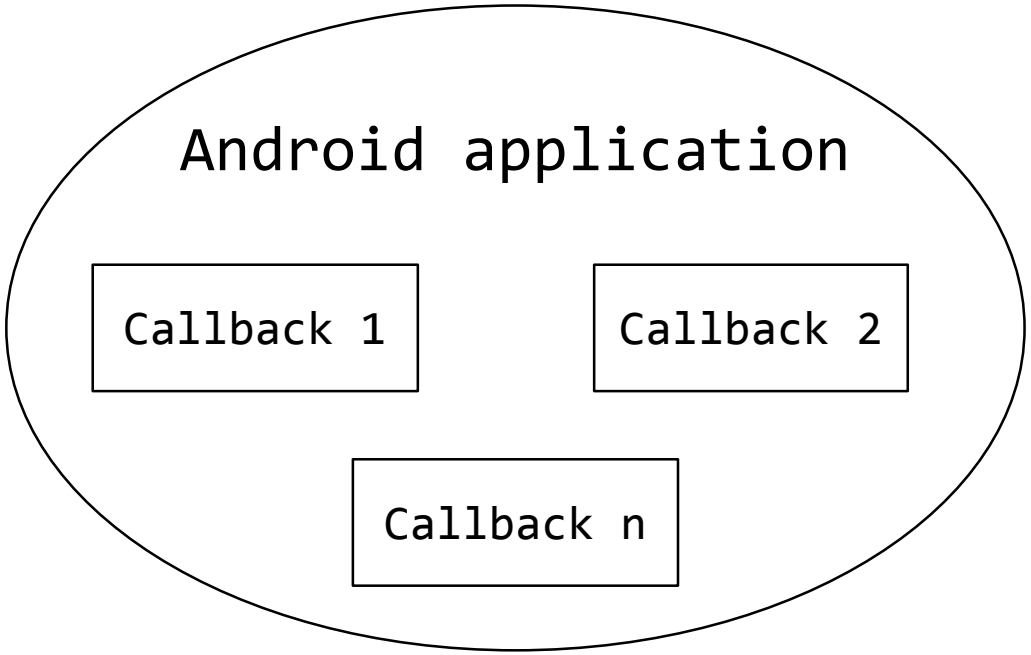


Firefox energy leak:

- Video keeps running on background tabs

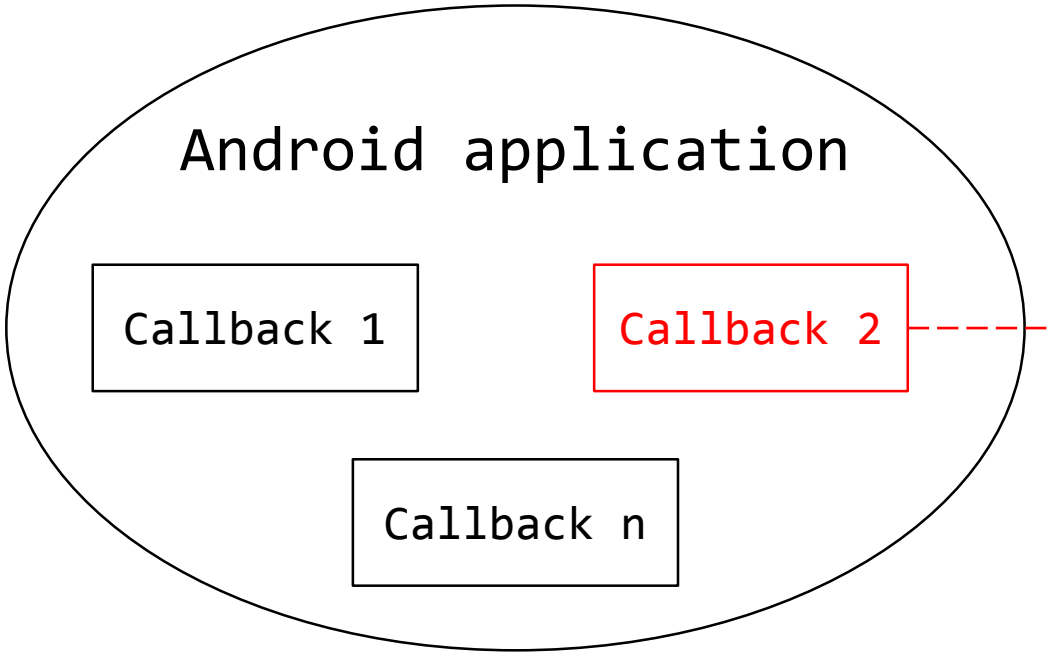
"When Fennec is in the background, these things should be suspended ideally: timers / JavaScript, animated images, Dom events, audio / video, flash plugins." (Firefox bug 736311)

3. Inefficient frequently-invoked callbacks

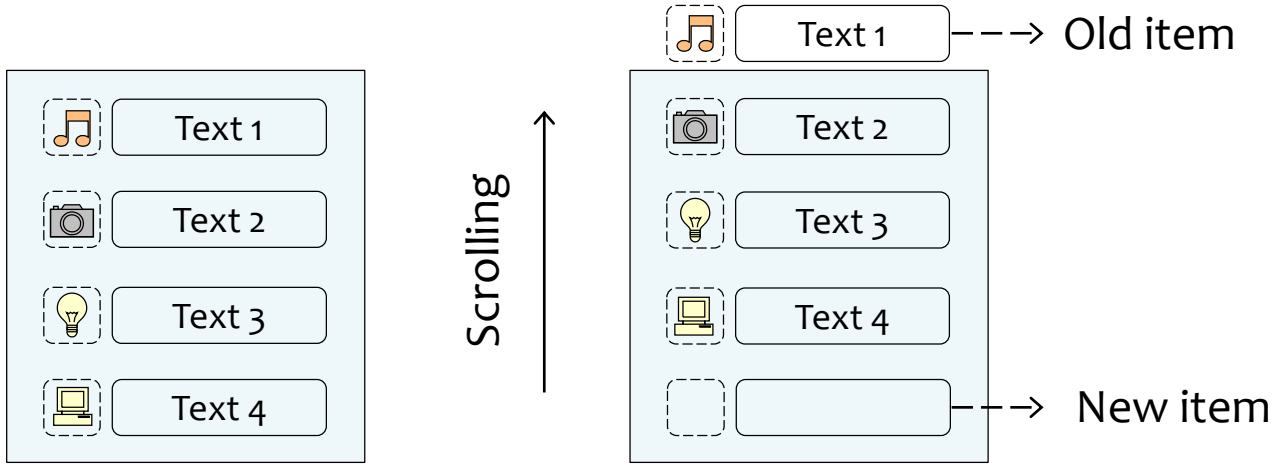


3. Inefficient frequently-invoked callbacks

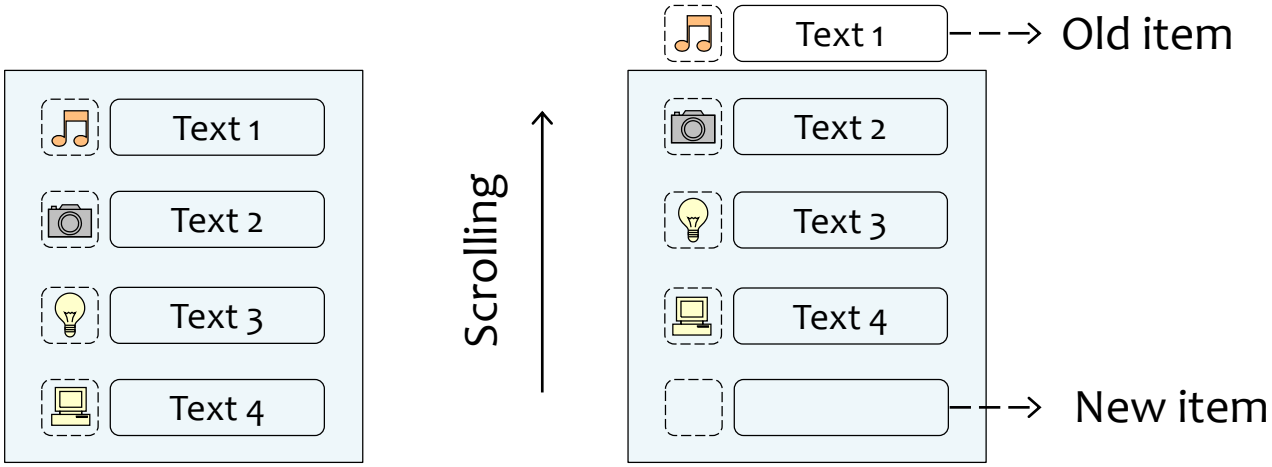
Frequently-invoked callbacks should be highly efficient.



ListView callback example



ListView callback example

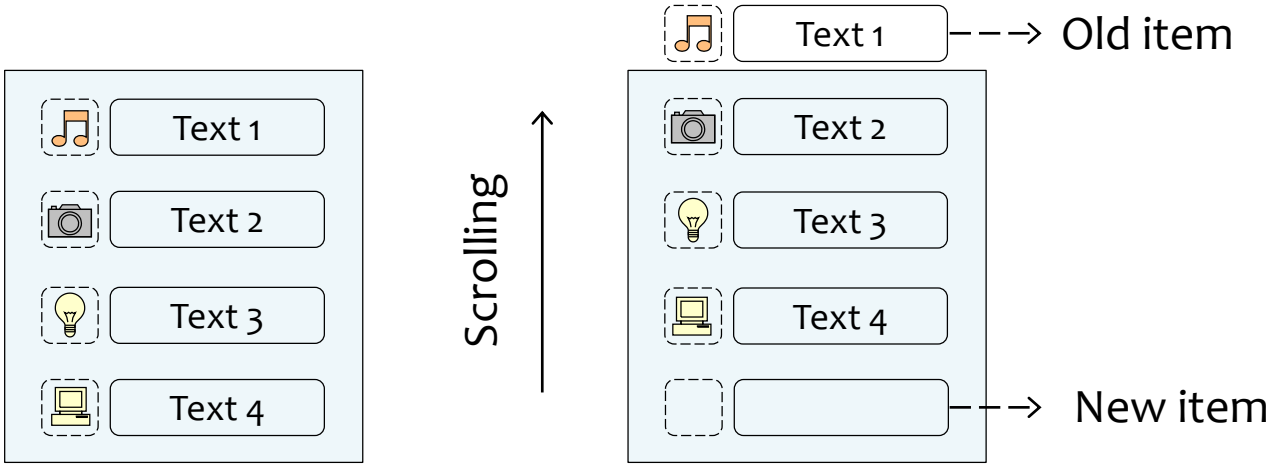



```
public View getView(int pos, View recycledView, ...)
```

- Operation 1: item layout inflation
- Operation 2: inner view updating

ListView callback example

Efficiency is critical (tens of invocations during a scrolling)



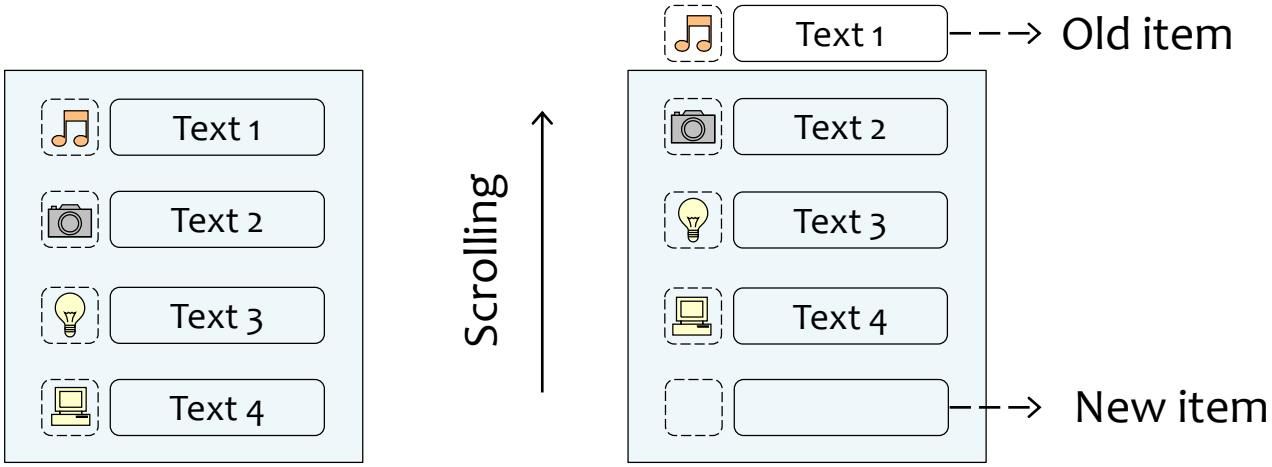
Heavy operations!

```
public View getView(int pos, View recycledView, ...)
```

- Operation 1: item layout inflation
- Operation 2: inner view updating

View holder design pattern

Observation: List items have identical layout



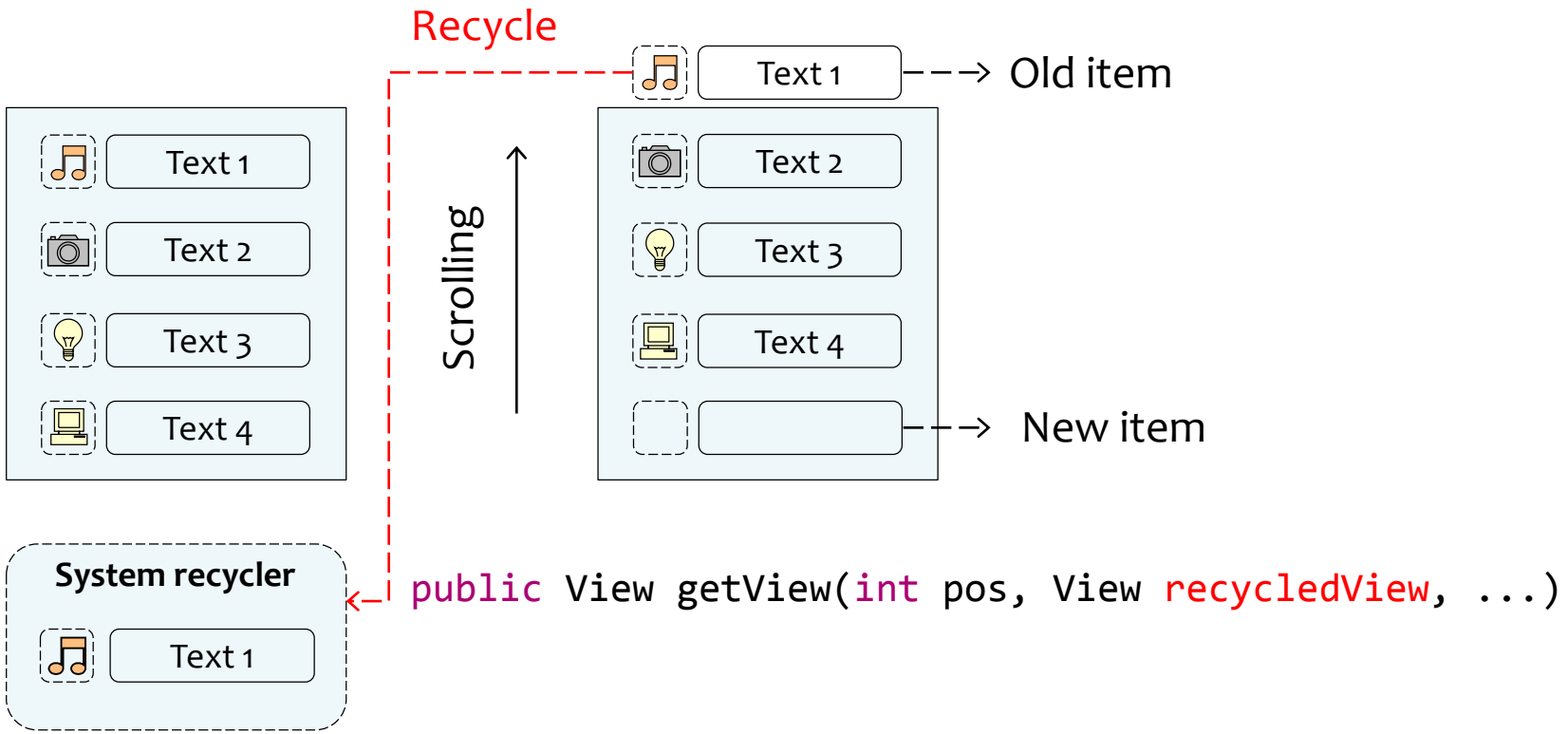
System recycler

```
public View getView(int pos, View recycledView, ...)
```

View holder design pattern

Observation: List items have identical layout

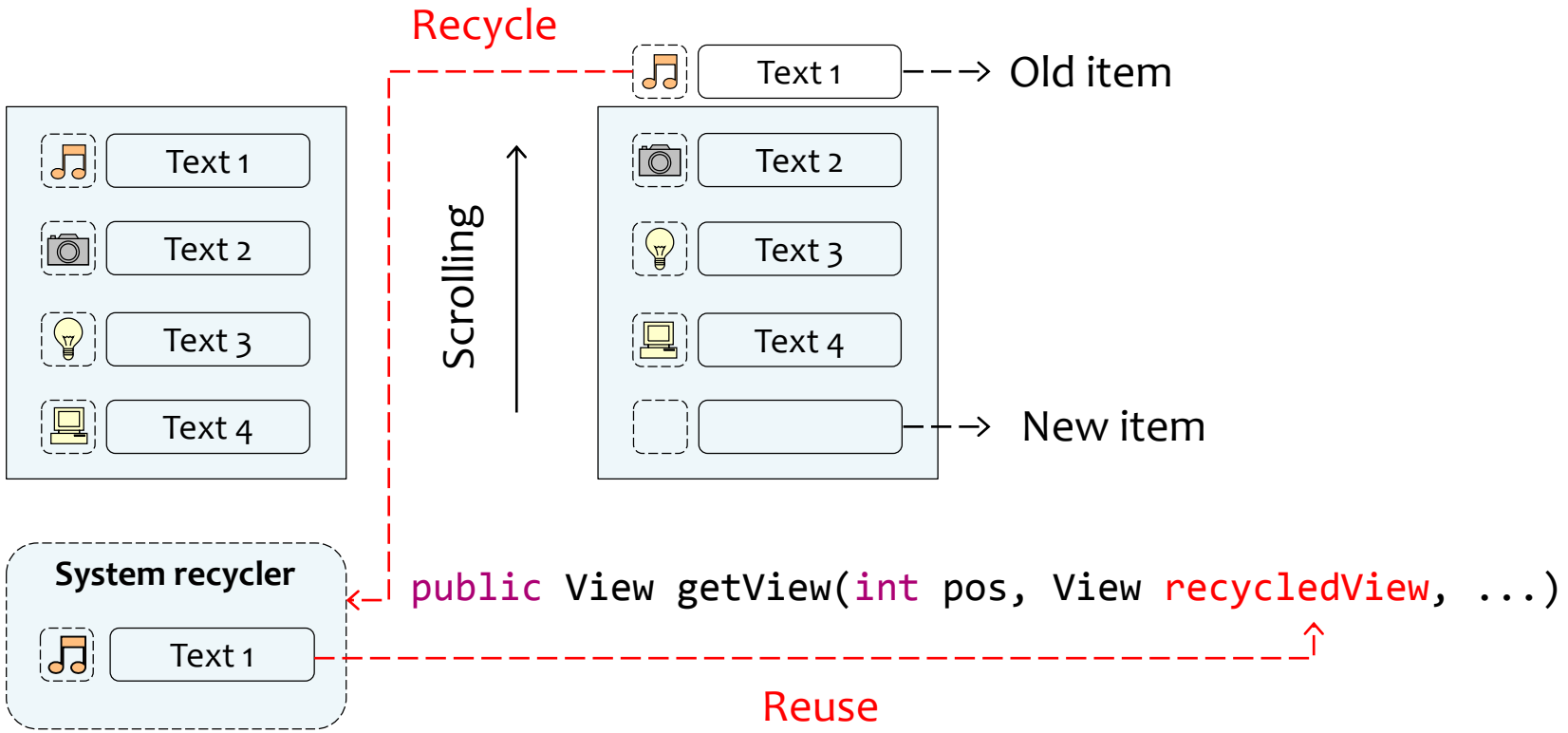
Idea: Recycle old item and cache inner view references for reuse



View holder design pattern

Observation: List items have identical layout

Idea: Recycle old item and cache inner view references for reuse



Violating view holder design pattern

```
//Simplified from Firefox bug 735736
public View getView(int pos, View recycledView, ViewGroup parent) {
  View item = mInflater.inflate(R.layout.listItem, null);
  TextView txtView = (TextView) item.findViewById(R.id.text);
  ImageView imgView = (ImageView) item.findViewById(R.id.icon);
  txtView.setText(DATA[pos]);
  imgView.setImageBitmap((pos % 2) == 1 ? mIcon1 : mIcon2);
  return item;
}
```

Violating view holder design pattern

Recycled item not used at all
item inflation and inner view updating on each call!

```
//Simplified from Firefox bug ~~735736~~
                                (735636)
public View getView(int pos, View recycledView, ViewGroup parent) {
  View item = mInflater.inflate(R.layout.listItem, null);
  TextView txtView = (TextView) item.findViewById(R.id.text);
  ImageView imgView = (ImageView) item.findViewById(R.id.icon);
  txtView.setText(DATA[pos]);
  imgView.setImageBitmap((pos % 2) == 1 ? mIcon1 : mIcon2);
  return item;
}
```

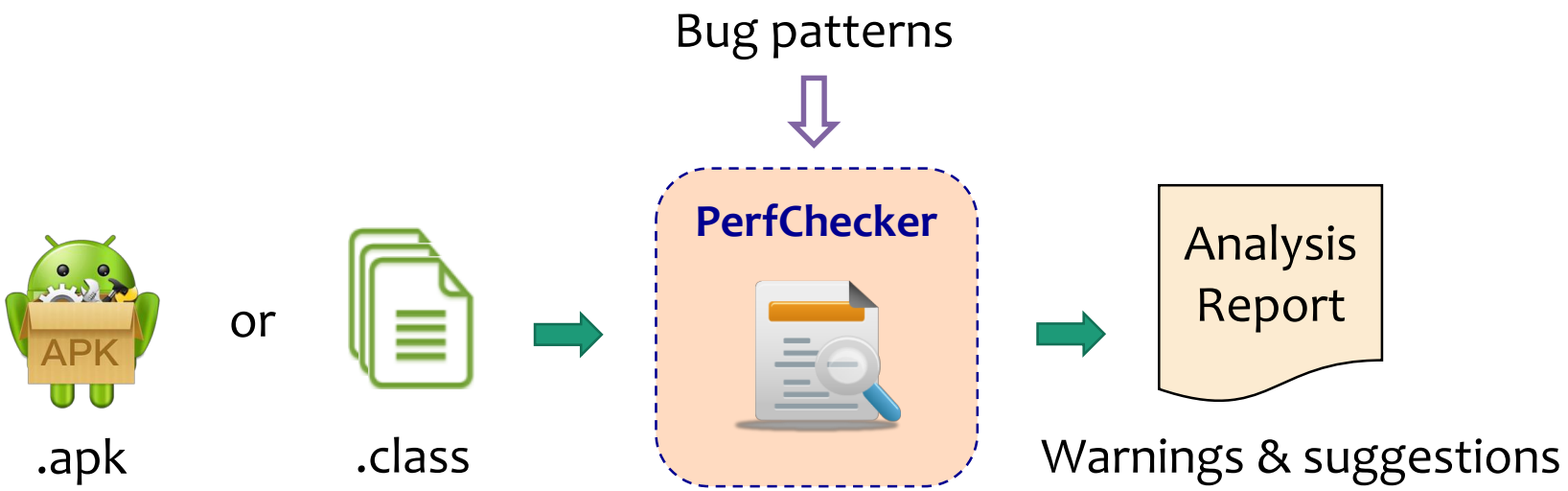
Consequence: bad scrolling performance!

Overview

- Empirical study: understanding performance bug
  - Research questions and study design
  - Empirical findings and implications
- PerfChecker: a performance bug detection tool
  - Tool design and implementation
  - Detected bugs and developers' feedback

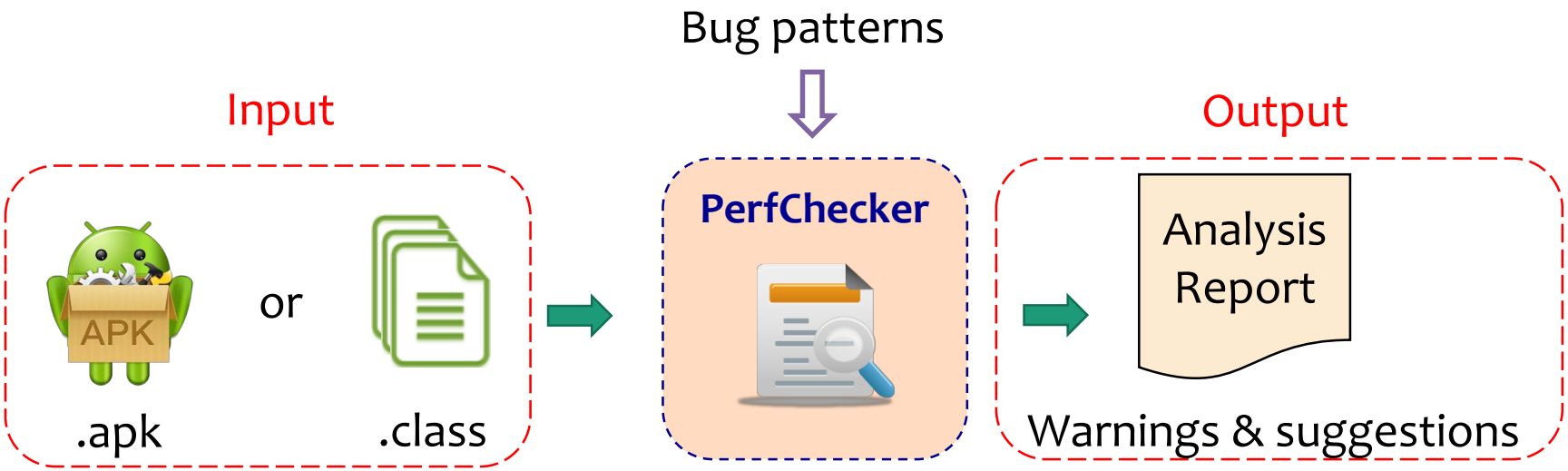
PerfChecker

- Static analysis for performance bug detection

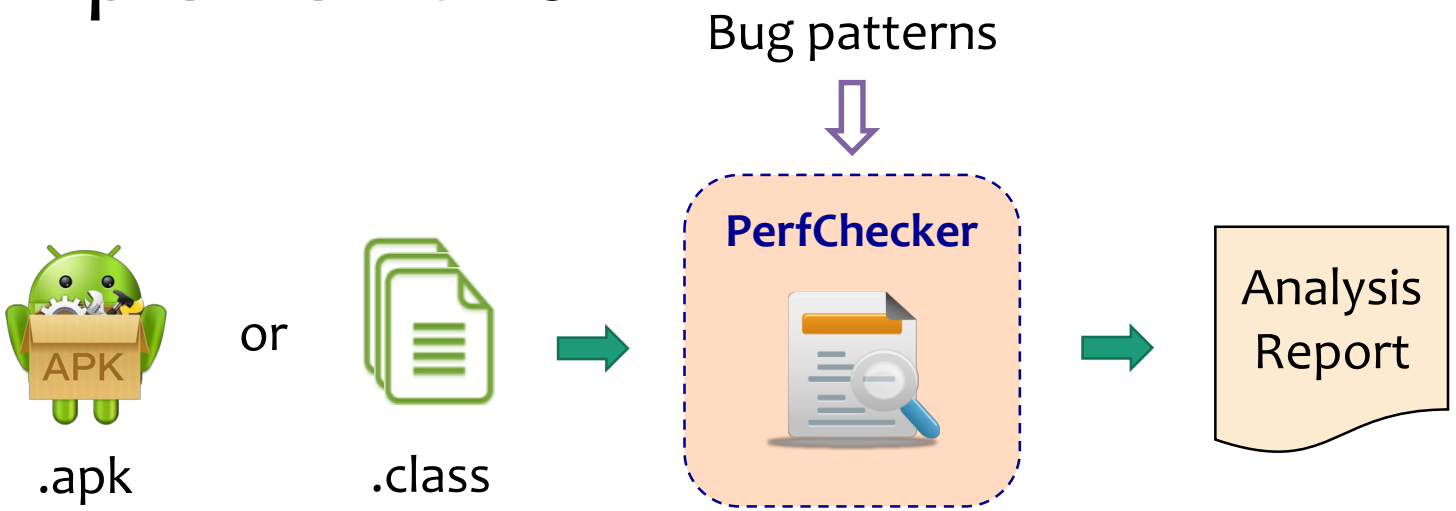


PerfChecker

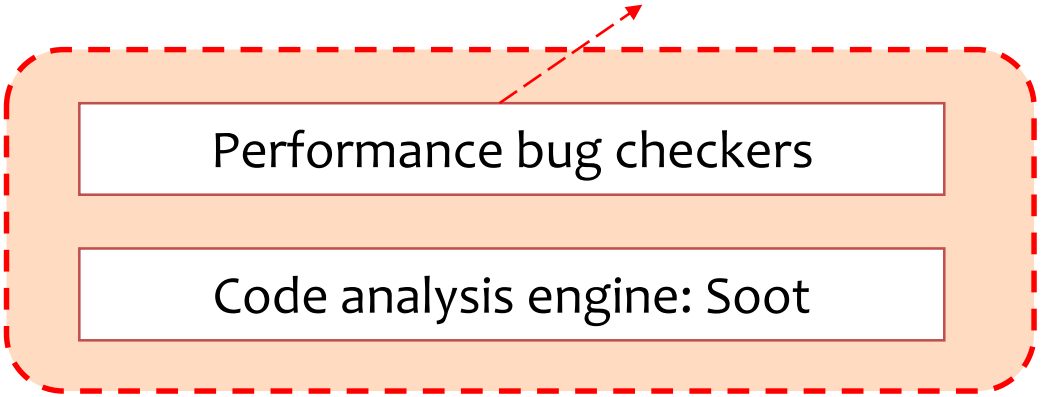
- Static analysis for performance bug detection
- Fully automated and easy to use



Implementation



- Long running operations in main threads
- View holder pattern violations



Application subjects

Latest version of 29 popular open-source Android apps

- Covering 12 different categories
- 1+ million lines of Java code in total

Application name	Category	Size (LOC)	Downloads
c:geo	Entertainment	37.7K	1M ~ 5M
Osmand	Travel & Local	77.4K	500K ~ 1M
Firefox	Communication	122.9K	10M ~ 50M
FBReaderJ	Books & Reference	103.4K	5M ~ 10M
Bitcoin Wallet	Finance	35.1K	100K ~ 500K
OI File Manager	Productivity	7.8K	5M ~ 10M
…	…	…	…

Analysis results

- PerfChecker can finish analyzing each application in a few seconds to a few minutes

Analysis results

- PerfChecker detected 126 previously-unknown issues in 18 of the 29 analyzed applications

Application name	Bug pattern instances	
	View holder pattern violation	Long running operations in main threads
Ushahidi	9	2
Firefox	1	0
FBReaderJ	6	6
OI File Manager	1	0
…	…	…

Analysis results

- 68 issues (54.0%) were confirmed as real performance bugs by original developers

Application name	Bug pattern instances	
	View holder pattern violation	Long running operations in main threads
Ushahidi	9	2
Firefox	1	0
FBReaderJ	6	6
OI File Manager	1	0
…	…	…

Analysis results

- 20 critical performance bugs were quickly fixed by original application developers

Application name	Bug pattern instances	
	View holder pattern violation	Long running operations in main threads
Ushahidi	9	2
Firefox	1	0
FBReaderJ	6	6
OI File Manager	1	0
…	…	…

Feedback from developers

- Developers are interested in performance analyzers

Henry (Ushahidi developer)

"Thanks for reporting this … Just curious, where is this static code checker? Anywhere I can play with it as well?"

Feedback from developers

- Developers act quickly with concrete suggestions

George (OI File Manager developer)

"Thanks a lot for reporting the problems. The ViewHolder pattern has just been added to the BookmarkListAdapter in 8c9c429."

Latest news

One of our checkers merged into Android Studio (for IntelliJ)

Android Studio 0.5.2 Release Log

Posted on Mar 20, 2014 by Tor Norbye

- New Lint check:

  Ensures that list view adapters use the View Holder pattern (to make scrolling smoother) …

Conclusion

- We discussed several characteristics of performance bug
- Performance bug detection tools are helpful to developers
- Future work on improving PerfChecker
  - More bug patterns to boost its detection capability
  - Improve the effectiveness of bug detection algorithms

For empirical study data and tool runnable, please visit:
http://sccpu2.cse.ust.hk/perfchecker/

Thank you ☺