GROPG: A Graphical On-Phone Debugger

May 22, 2013

Tuan Anh Nguyen
tanguyen@mavs.uta.edu

Christoph Csallner
csallner@uta.edu

Computer Science and Engineering Department
University of Texas at Arlington

Nikolai Tillmann
nikolait@microsoft.com

Microsoft Research
Redmond, WA

Debugger: Important tool

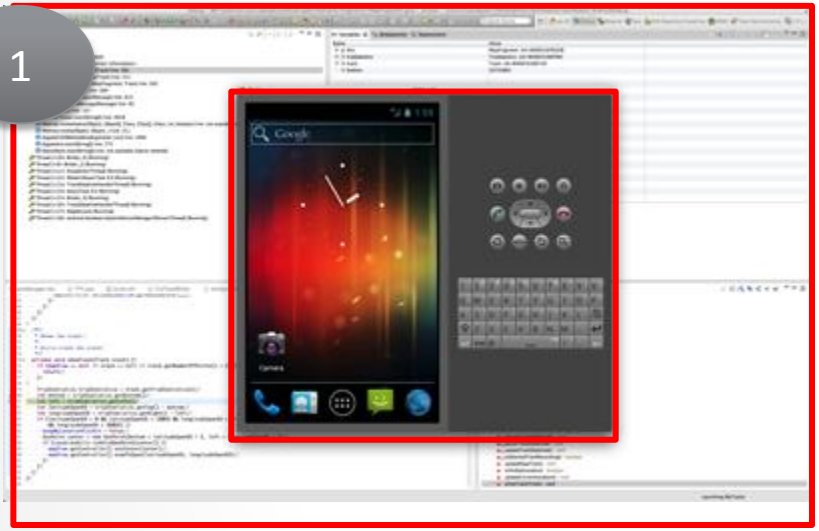
- Debugging common during development and maintenance
- Murphy'06 study: Java developers using Eclipse:
  >90% of developers used built-in debugger to inspect memory values during program execution
- Roehm'12 study: Professional developers use debuggers heavily for program comprehension

G. C. **Murphy**, M. Kersten, and L. Findlater, "How are Java software developers using the Eclipse IDE?" IEEE Software, vol. 23, no. 4, pp. 76–83, Jul. 2006.
T. **Roehm**, R. Tiarks, R. Koschke, and W. Maalej, "How do professional developers comprehend software?" in Proc. 34th ACM/IEEE International Conference on Software Engineering (ICSE). IEEE, Jun. 2012, pp. 255– 265.

Debugging mobile phone applications today: 2 mainstream techniques

Common on: iOS, Android, Windows Phone, etc.

1



Attach desktop debugger to a virtual device

Cannot emulate all phone features (e.g., Accelerometer, Gyroscope, GPS)

2

Attach desktop debugger to a phone

Need two physical devices

Pay for two physical devices

Real sensor readings: restricted movement

Mainstream techniques ultimately require two physical devices

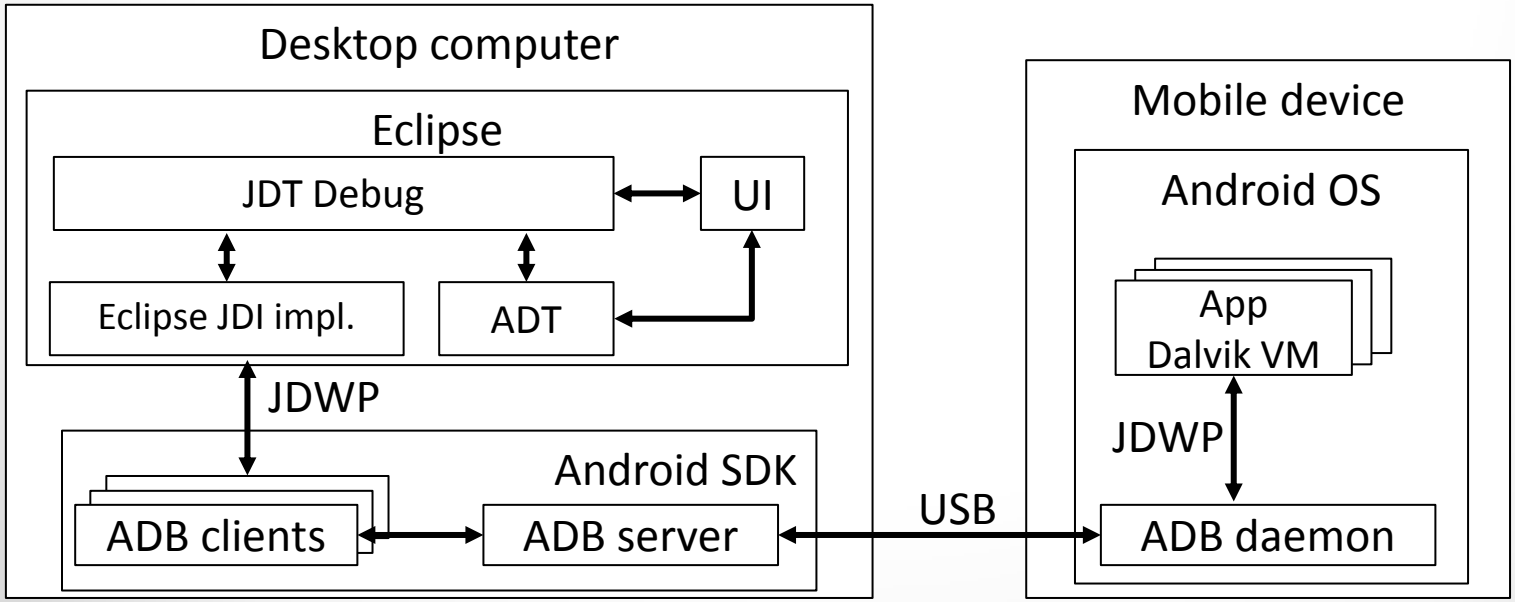
Example: Mainstream debugger architecture for Android

Android

- Implemented in C, but almost all Android applications are written in Java
- Consists of OS services, a Linux-based kernel, and the Dalvik virtual machine.

Dalvik virtual machine (VM)

- Conceptually similar to a Java virtual machine
- Implements two debug interfaces defined by the Java virtual machine: Java Debug Interface (JDI) and Java Debug Wire Protocol (JDWP)



Challenges of building on-phone debuggers

Desktop

- Developers can see during debugging several kinds of information on the screen
- Desktop debuggers use UI elements optimized for keyboard short-cuts and mouse interaction

Mobile devices

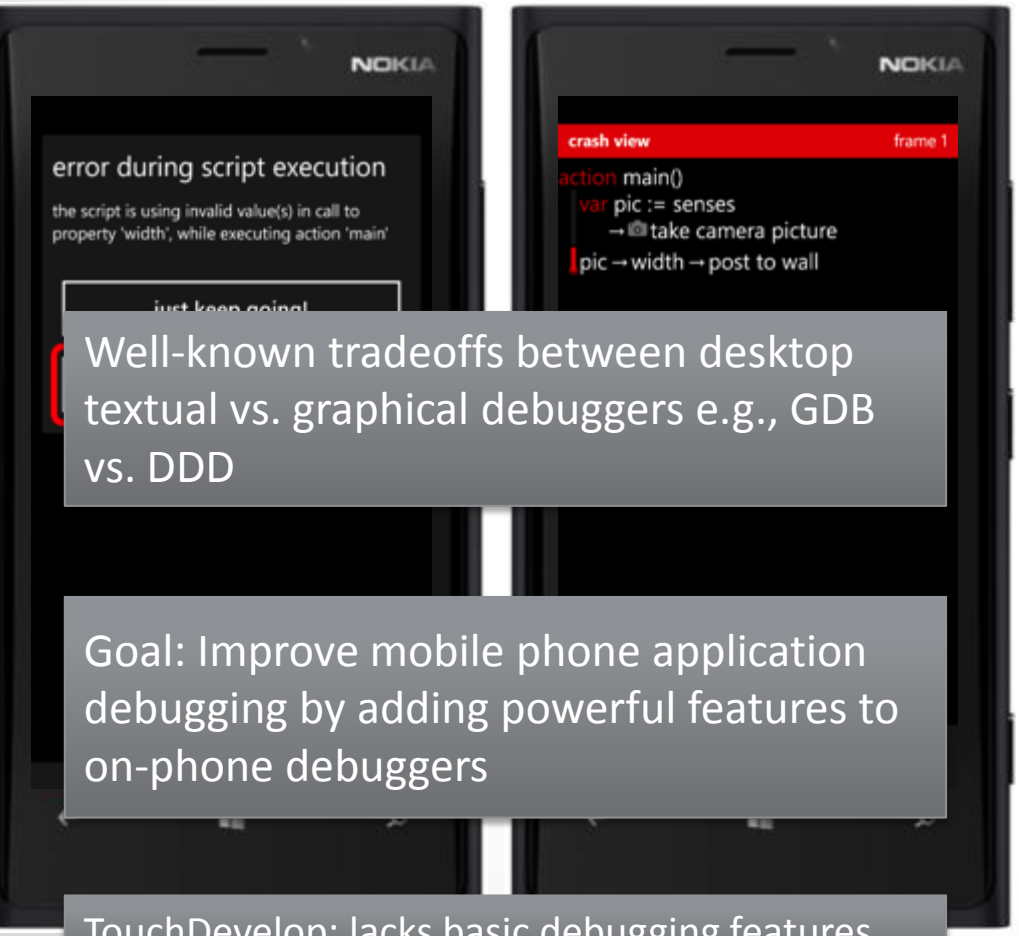
- Limited screen real estate: comparatively small mobile phone screen size
- Mobile phones use touch and gesture interaction

Mobile evolving

- Narrowing the performance gap with desktop computers: CPU, memory
- Providing high- resolution touch-screens ➔ enable interactive debugging

On-phone debuggers today

- Pioneering techniques only require one device: a mobile phone



Well-known tradeoffs between desktop textual vs. graphical debuggers e.g., GDB vs. DDD

Goal: Improve mobile phone application debugging by adding powerful features to on-phone debuggers

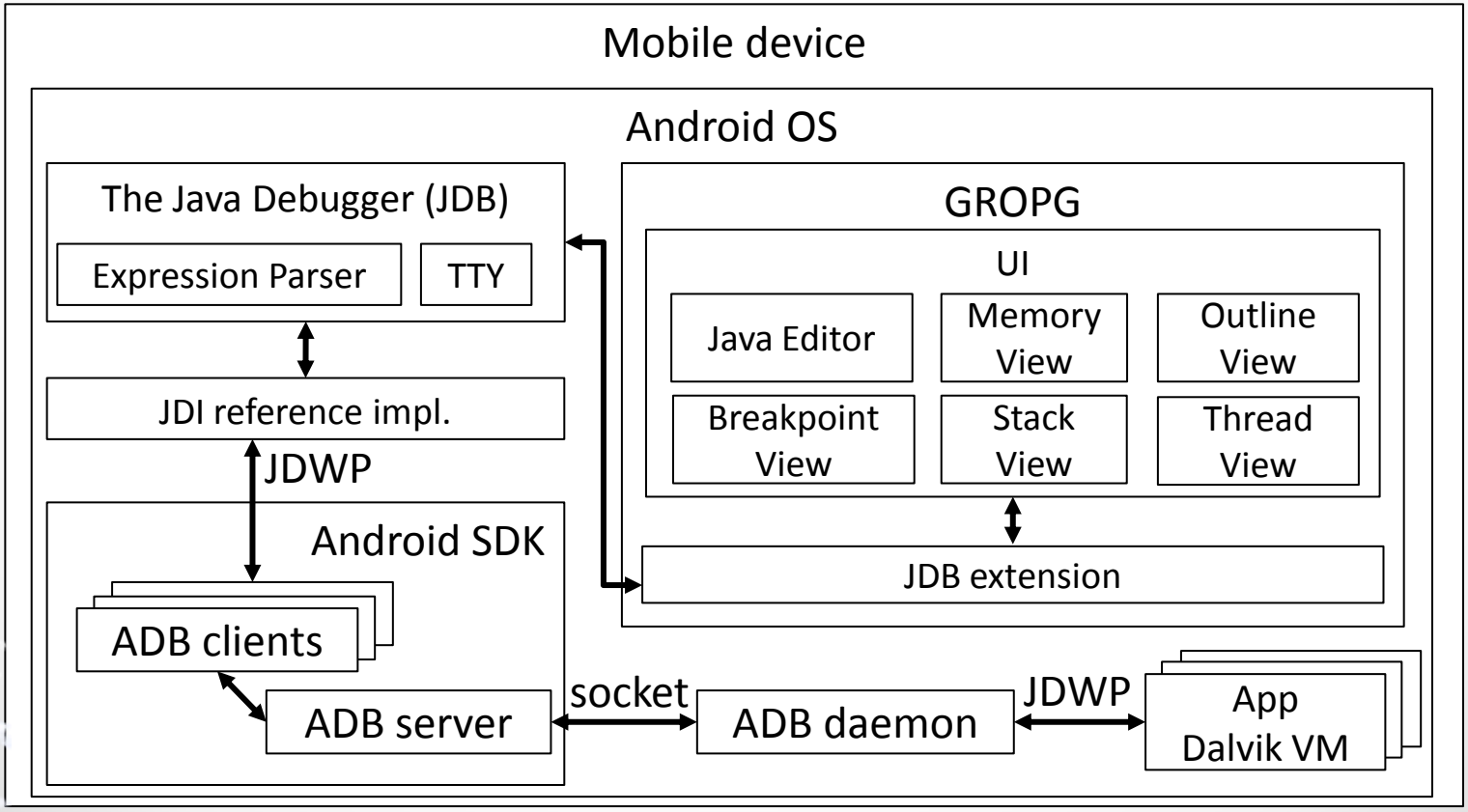
TouchDevelop: lacks basic debugging features

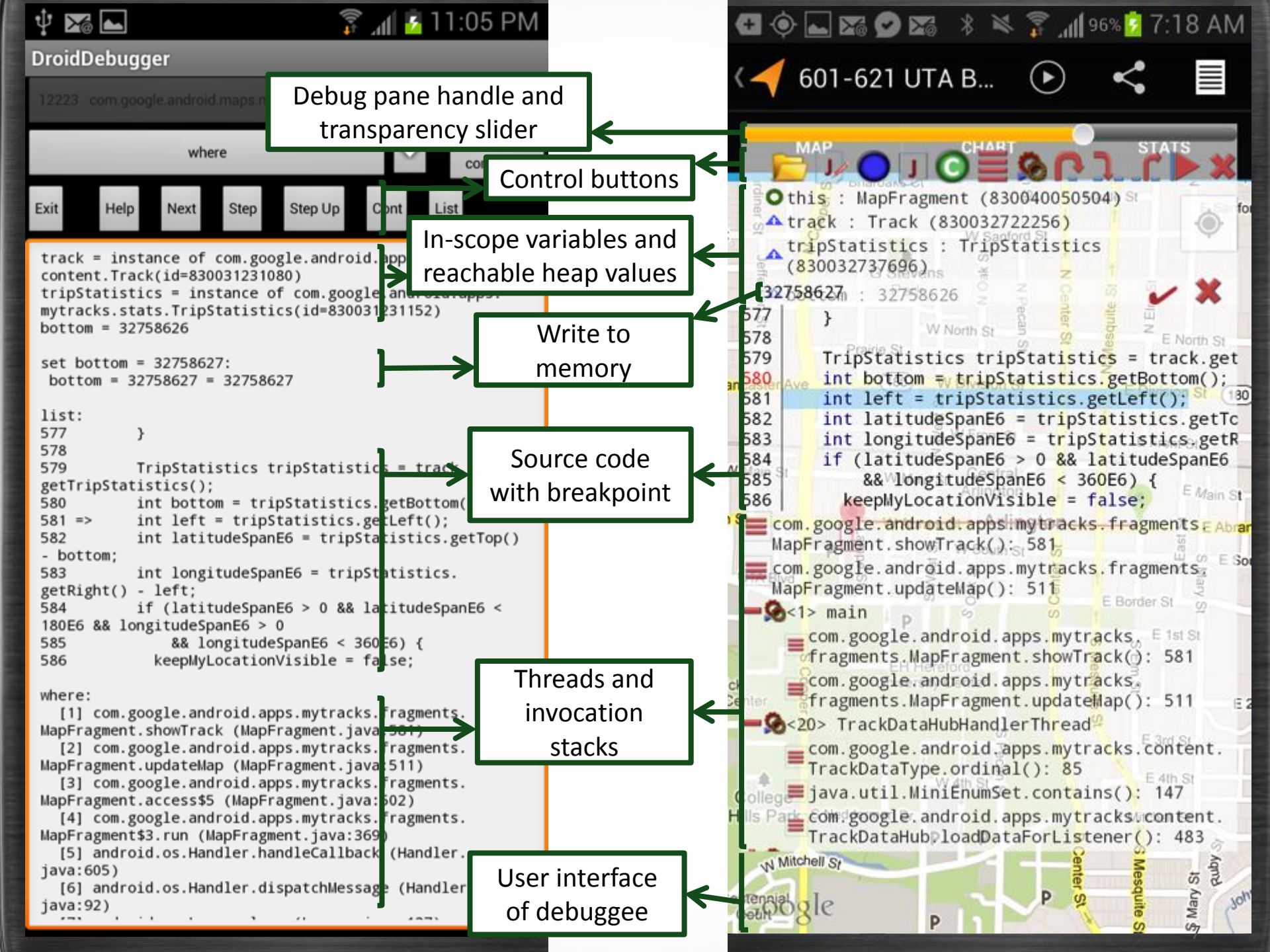
3

Text-based on-phone debugger:
less powerful than graphical debugger; developers cannot:
- quickly navigate and manipulate the debuggee memory
- view the debuggee's current source code location, memory values, call stack, and and debuggee UI side-by-side.

GROPG: Graphical on-phone debugger

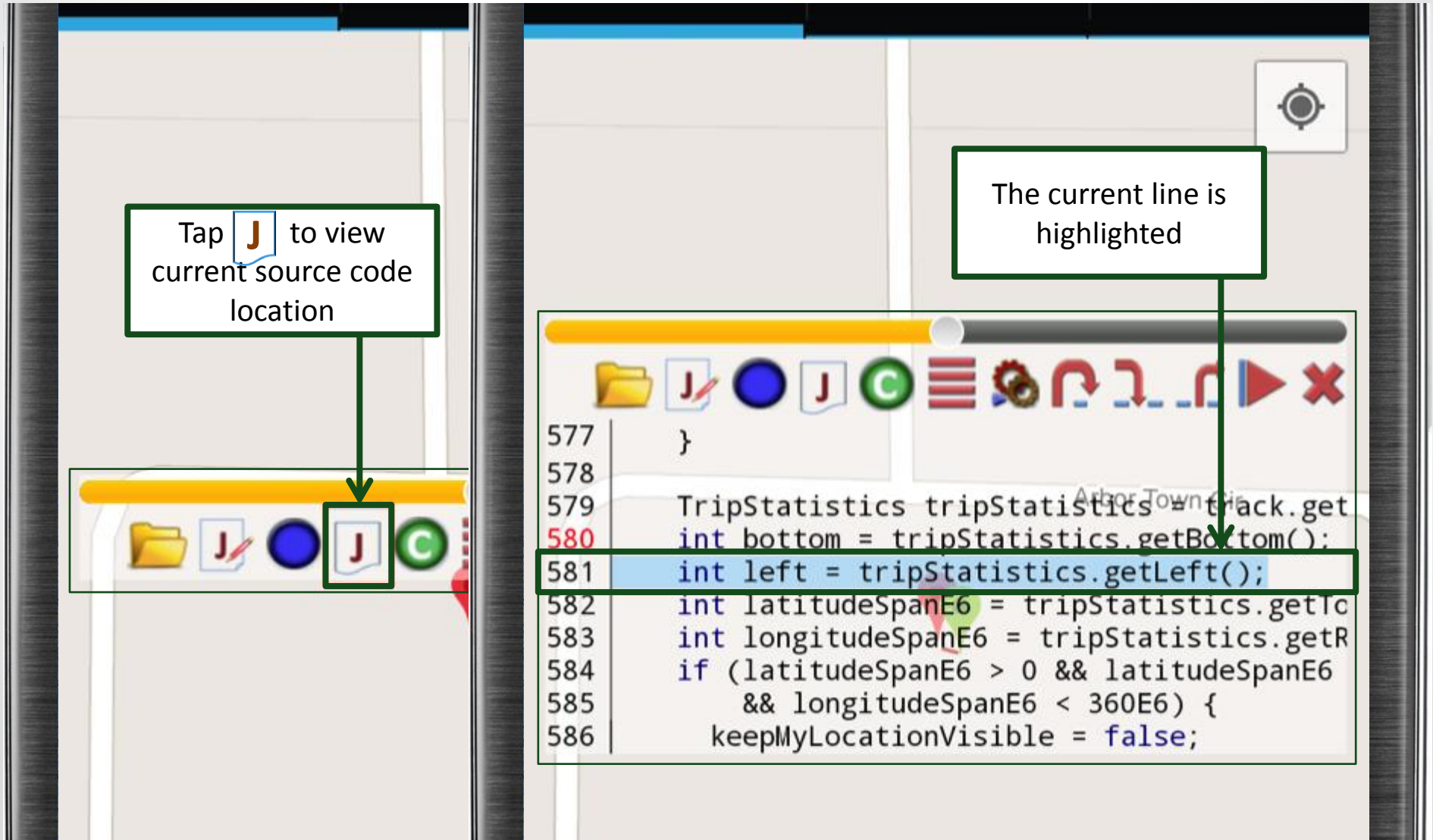
- GROPG is built on the same ADB, JDWP, and JDI debugging infrastructure as the two-device debugger
- The Java Debugger (JDB) is a simple command-line debugger included JDK



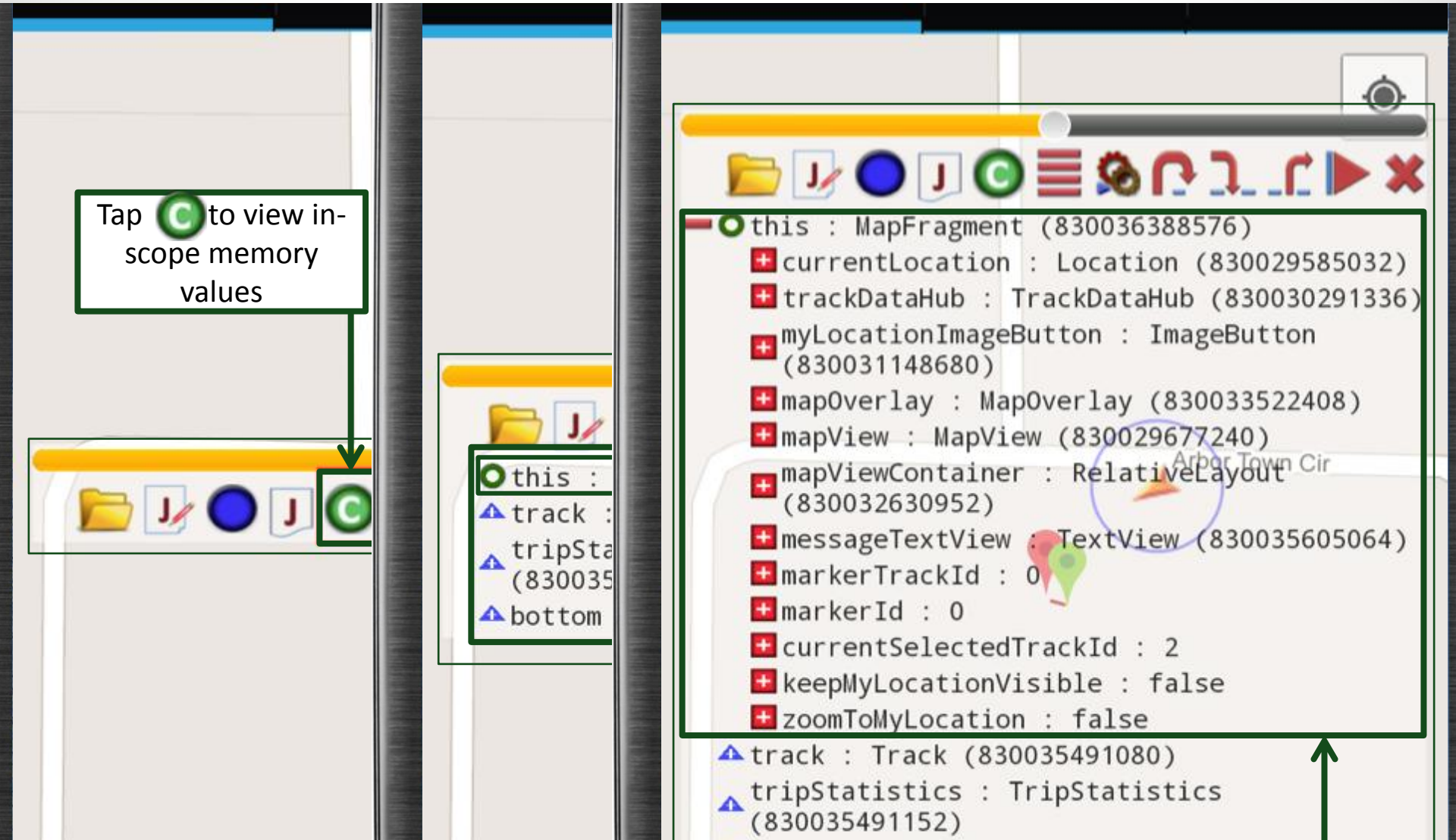

11:05 PM
DroidDebugger
where
Exit
Help
Next
Step
Step Up
Cont
List
track = instance of com.google.android.app
content.Track(id=830031231080)
tripStatistics = instance of com.google.and
mytracks.stats.TripStatistics(id=830031231152)
bottom = 32758626
set bottom = 32758627:
bottom = 32758627 = 32758627
list:
577 }
578
579 TripStatistics tripStatistics = track.
getTripStatistics();
580 int bottom = tripStatistics.getBottom(
581 => int left = tripStatistics.getLeft();
582 int latitudeSpanE6 = tripStatistics.getTop()
- bottom;
583 int longitudeSpanE6 = tripStatistics.
getRight() - left;
584 if (latitudeSpanE6 > 0 && latitudeSpanE6 <
180E6 && longitudeSpanE6 > 0
585 && longitudeSpanE6 < 360E6) {
586 keepMyLocationVisible = false;
where:
[1] com.google.android.apps.mytracks.fragments.
MapFragment.showTrack (MapFragment.java:581)
[2] com.google.android.apps.mytracks.fragments.
MapFragment.updateMap (MapFragment.java:511)
[3] com.google.android.apps.mytracks.fragments.
MapFragment.access$5 (MapFragment.java:502)
[4] com.google.android.apps.mytracks.fragments.
MapFragment$3.run (MapFragment.java:369)
[5] android.os.Handler.handleCallback (Handler.
java:605)
[6] android.os.Handler.dispatchMessage (Handler
java:92)
Debug pane handle and transparency slider
Control buttons
In-scope variables and reachable heap values
Write to memory
Source code with breakpoint
Threads and invocation stacks
User interface of debuggee
7:18 AM
601-621 UTA B...
MAP
CHART
STATS
this : MapFragment (830040050504)
track : Track (830032722256)
tripStatistics : TripStatistics (830032737696)
32758627
bottom : 32758626
577 }
578
579 TripStatistics tripStatistics = track.get
580 int bottom = tripStatistics.getBottom();
581 int left = tripStatistics.getLeft();
582 int latitudeSpanE6 = tripStatistics.getTc
583 int longitudeSpanE6 = tripStatistics.getR
584 if (latitudeSpanE6 > 0 && latitudeSpanE6
585 && longitudeSpanE6 < 360E6) {
586 keepMyLocationVisible = false;
com.google.android.apps.mytracks.fragments.
MapFragment.showTrack(): 581
com.google.android.apps.mytracks.fragments.
MapFragment.updateMap(): 511
<1> main
com.google.android.apps.mytracks.
fragments.MapFragment.showTrack(): 581
com.google.android.apps.mytracks.
fragments.MapFragment.updateMap(): 511
<20> TrackDataHubHandlerThread
com.google.android.apps.mytracks.content.
TrackDataType.ordinal(): 85
java.util.MiniEnumSet.contains(): 147
com.google.android.apps.mytracks.content.
TrackDataHub.loadDataForListener(): 483

Setting a breakpoint

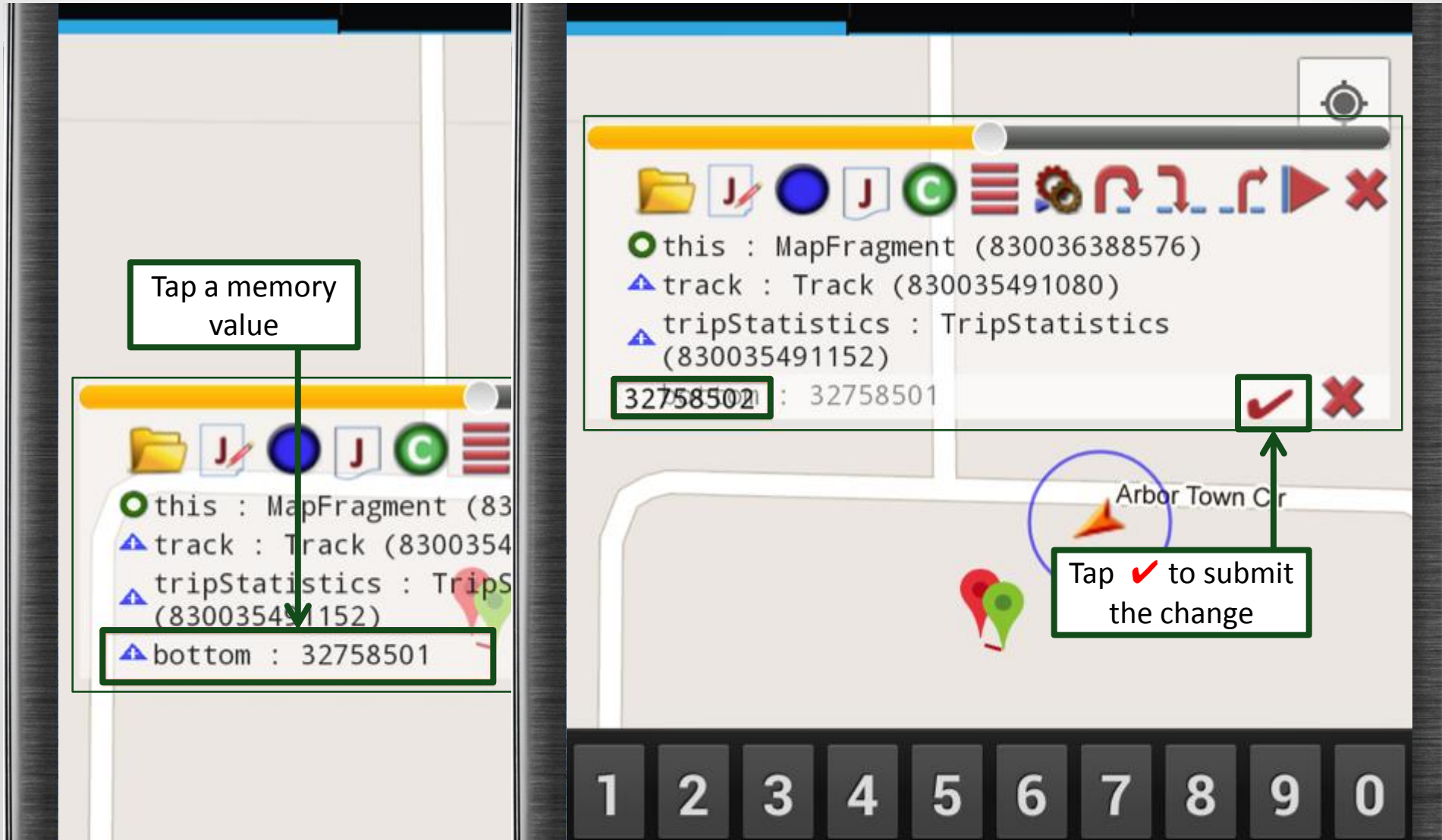
View current source code location



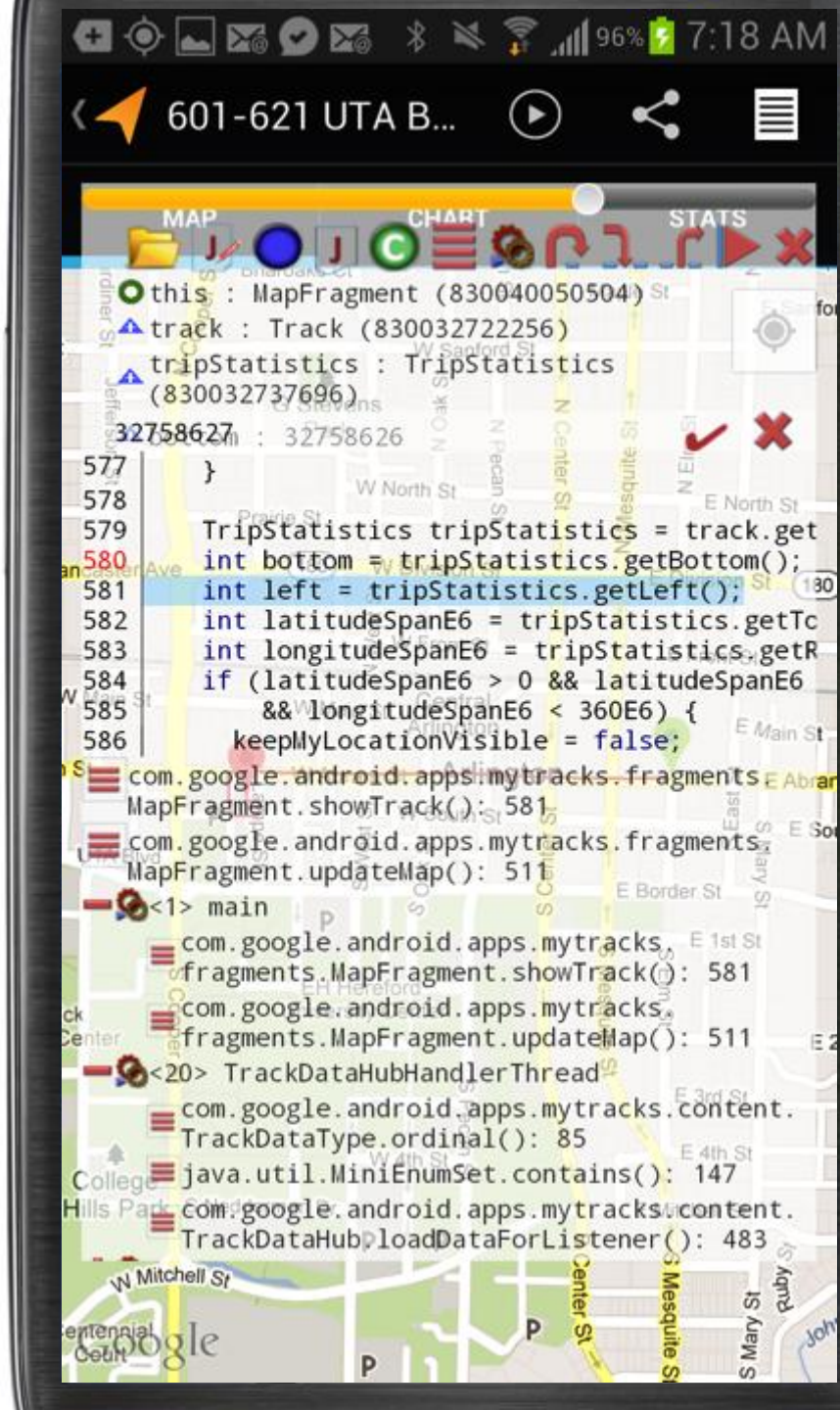
Inspect in-scope memory values



Change in-scope memory values



- Load source code files
- Set breakpoints
- View and edit active breakpoints
- Inspect & change in-scope memory values including heap
- Step into, over, and out of instructions
- Inspect threads and runtime stacks
- View current source code location



Preliminary experiments: Example debugging task

1. Start debugger and debuggee
2. Attach debugger to debuggee
3. Set one breakpoint
4. When debuggee reaches breakpoint: step to next instruction
5. Display memory values + method frame stack

Debuggee	LOC	DroidDebugger			GROPG		
		App [MB]	Debugger [MB]	Time [min:s]	App [MB]	Debugger [MB]	Time [min:s]
F2C	27	3	10	2:13	3	24	45
My Tracks	21,563	19	11	3:35	19	26	52
Zing	5,756	7	12	2:53	7	25	1:01

GROPG has a higher memory footprint but enables faster debugging

Preliminary experiments

Step 3 of previous slide:

Setting a breakpoint, for My Tracks

GROPG needs fewer steps & less time for setting a breakpoint.

#	DroidDebugger	Time [s]	GROPG	Time [s]
(1)	Open source file in external tool	21	Open source file	18
(2)	Review code in external tool	19	Review code	19
(3)	***Remember class, line for breakpoint***	***5***	***n/a***	***0***
(4)	***Switch back to debugger***	***3***	***n/a***	***0***
(5)	***Recall command syntax***	***0***	***n/a***	***0***
(6)	***Type:*** stop at com.google.android.apps. Mytracks.fragments.MapFragment:580	***63***	***Tap line***	***1***
(7)	Tap Exec command	1	Tap ● button	1
	Total	112		39

Status/Limitations

- Works fully with Android version 2.3 to 4.2.1, do not need root
- From 4.2.2 need root to work
- Not support setting breakpoint in anonymous inner classes
- Not show Android library source code during debugging
- If an application is compiled for release mode, Android prevents it from being debugged

 GROPG is open source, download sources from

http://cseweb.uta.edu/~tuan/GROPG/

Install GROPG via the Google App Store

https://play.google.com/store/apps/details?id=edu.uta.cse.gropg&hl=en

Future work

- Integrate with on-phone coding which is similar to Tillmann'11 to create a full on-phone IDE
- Address traditional debugging challenges on-phone:
  - Cross-language Java and native code debugging of Lee'09
  - Support for why-questions of Ko'04

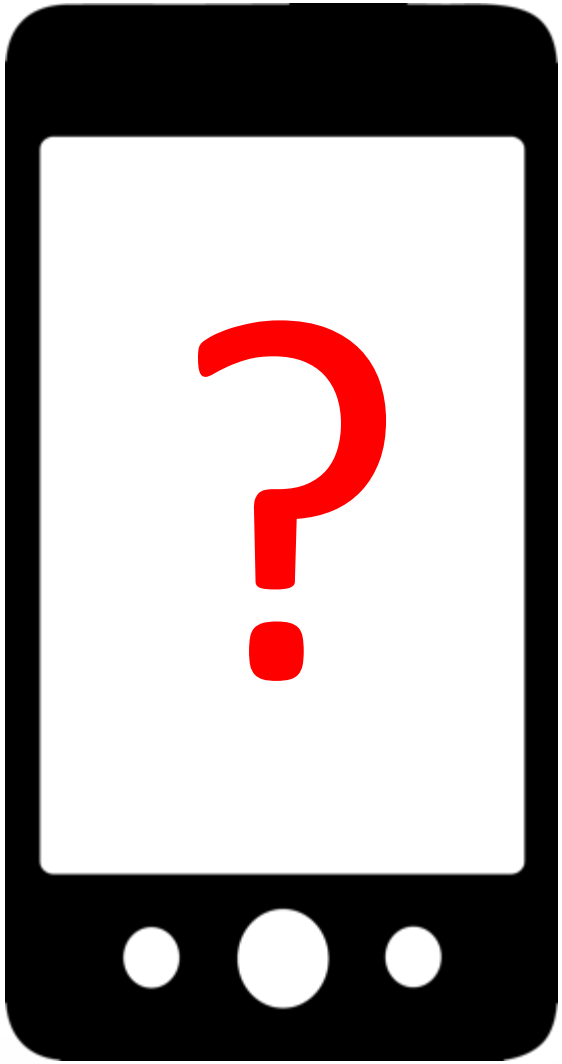
N. **Tillmann**, M. Moskal, J. de Halleux, and M. Fahndrich, "Touchdevelop: Programming cloud-connected mobile devices via touchscreen," in Proc. 10th SIGPLAN ONWARD. ACM, 2011, pp. 49–60.

B. **Lee**, M. Hirzel, R. Grimm, and K. S. McKinley, "Debug all your code: portable mixed-environment debugging," in Proc. 24th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA). ACM, Oct. 2009, pp. 207–226.

A. J. **Ko** and B. A. Myers, "Designing the Whyline: A debugging interface for asking questions about program behavior," in Proc. ACM SIGCHI CHI. ACM, Apr. 2004, pp. 151–158.

References

- G. C. Murphy, M. Kersten, and L. Findlater, “How are Java software developers using the Eclipse IDE?” IEEE Software, vol. 23, no. 4, pp. 76–83, Jul. 2006.
- T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, “How do professional developers comprehend software?” in Proc. 34th ACM/IEEE International Conference on Software Engineering (ICSE). IEEE, Jun. 2012, pp. 255– 265.
- N. Tillmann, M. Moskal, J. de Halleux, and M. Fahndrich, “Touchdevelop: Programming cloud-connected mobile devices via touchscreen,” in Proc. 10th SIGPLAN ONWARD. ACM, 2011, pp. 49–60.
- A. Zeller and D. Lütkehaus, “DDD - A free graphical front-end for Unix debuggers,” SIGPLAN Notices, vol. 31, no. 1, pp. 22–27, Jan. 1996.
- N. Matloff and P. J. Salzman, The Art of Debugging with GDB, DDD,
- and Eclipse. No Starch Press, Sep. 2008.
- B. Lee, M. Hirzel, R. Grimm, and K. S. McKinley, “Debug all your code:
- portable mixed-environment debugging,” in Proc. 24th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA). ACM, Oct. 2009, pp. 207–226.
- A. J. Ko and B. A. Myers, “Designing the Whyline: A debugging interface for asking questions about program behavior,” in Proc. ACM SIGCHI CHI. ACM, Apr. 2004, pp. 151–158.



Challenges of building on-phone debuggers

CPU: Octa-core 1.6 Ghz
Memory: 2 GB
Pixel density: 441 ppi