Accelerate your Mobile Apps and Games for Android™ on ARM

Matthew Du Puy
Software Engineer, ARM

Presenter

Matthew Du Puy

Software Engineer, ARM

Matthew Du Puy is a software engineer at ARM and is currently working to ensuring mobile app performance on the latest ARM technologies. Previously a self employed embedded systems software contractor working primarily on the Linux Kernel and a mountain climber.

Contact Details:

Email: **matthew.dupuy@arm.com**

Problem: This is not a desktop

- Mobile apps require special design considerations that aren't always clear and tools to solve increasingly complex systems are limited
  - Animations and games drop frames
  - Networking, display, real time audio and video processing eat battery
  - App won't fit in memory constraints

Analysis

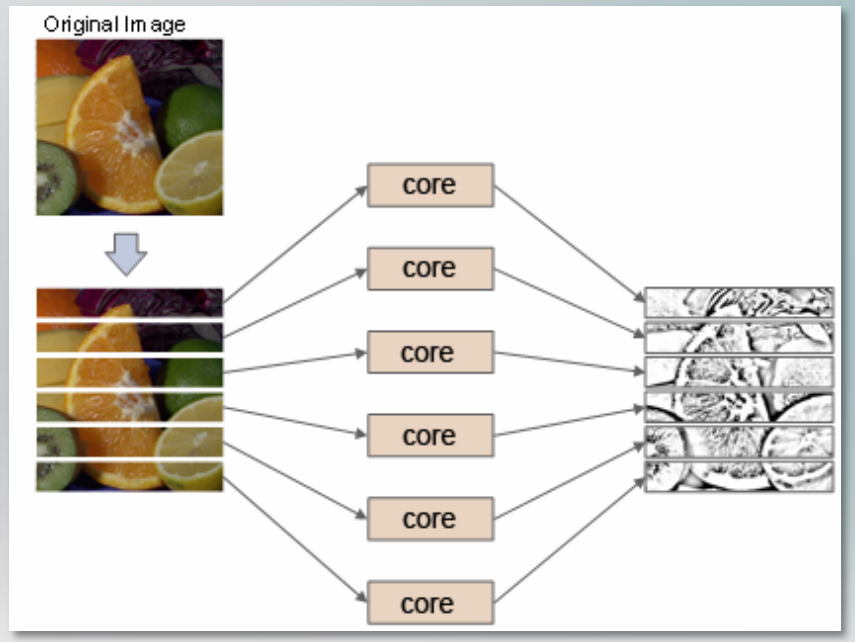
- Fortunately Google, ARM and many others are developing analysis tools and solutions to these problems
- Is my app … ?
  - CPU/GPGPU bound
  - I/O or memory constrained
  - Power efficient
- What can I do to fix it?
  (short of buying everyone who runs my app a Quad-core ARM® Cortex™-A15 processor & ARM Mali™-T604 processor or Octo phone)

Analysis of Java SDK Android Apps

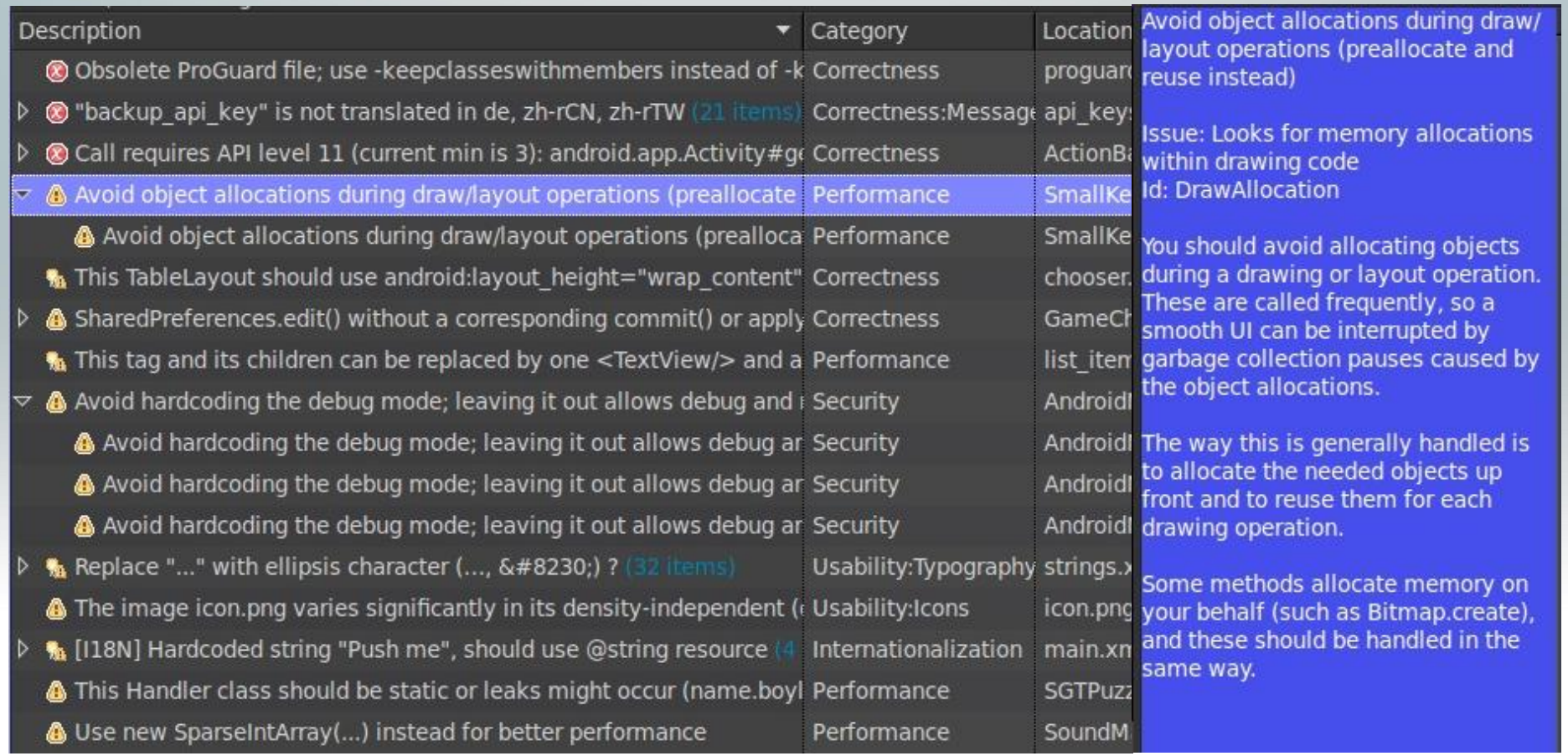
- Static analysis with SDK Lint tool
- Dynamic analysis with DDMS
  - Allocation/heap
  - Process and thread utilization
  - Traceview (method)
  - Network
- Hierarchy Viewer
- Systrace

But ask yourself these questions

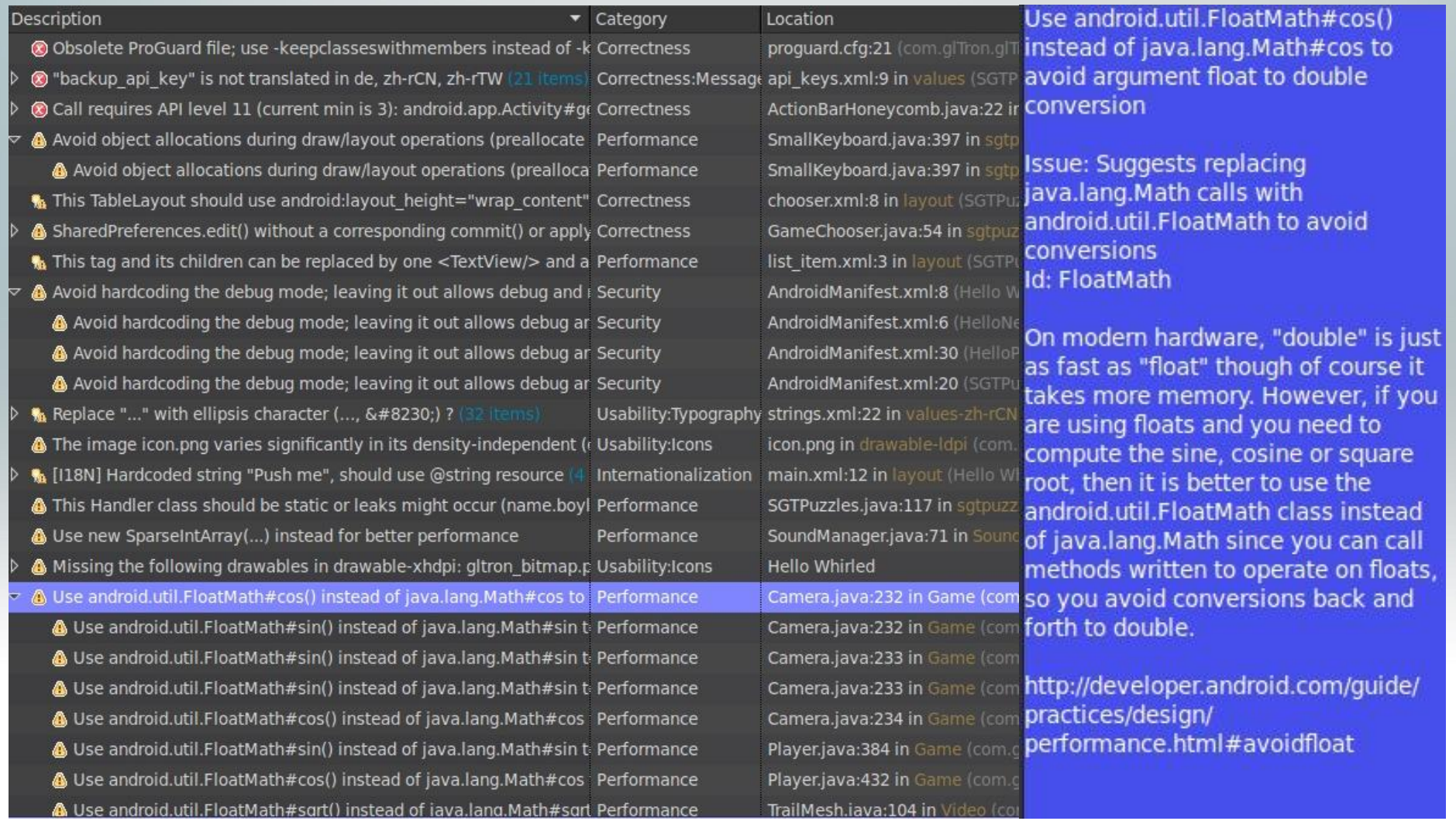
- Is this performance bottleneck parallelizable?
- Is this Java or Native?
  Would it be better the
  other way around?
- Has this been done
  before? Don't reinvent the wheel.
- Am I being smart with resources?
- What version of Android should I target?



Starting EASY
Static analysis: LINT

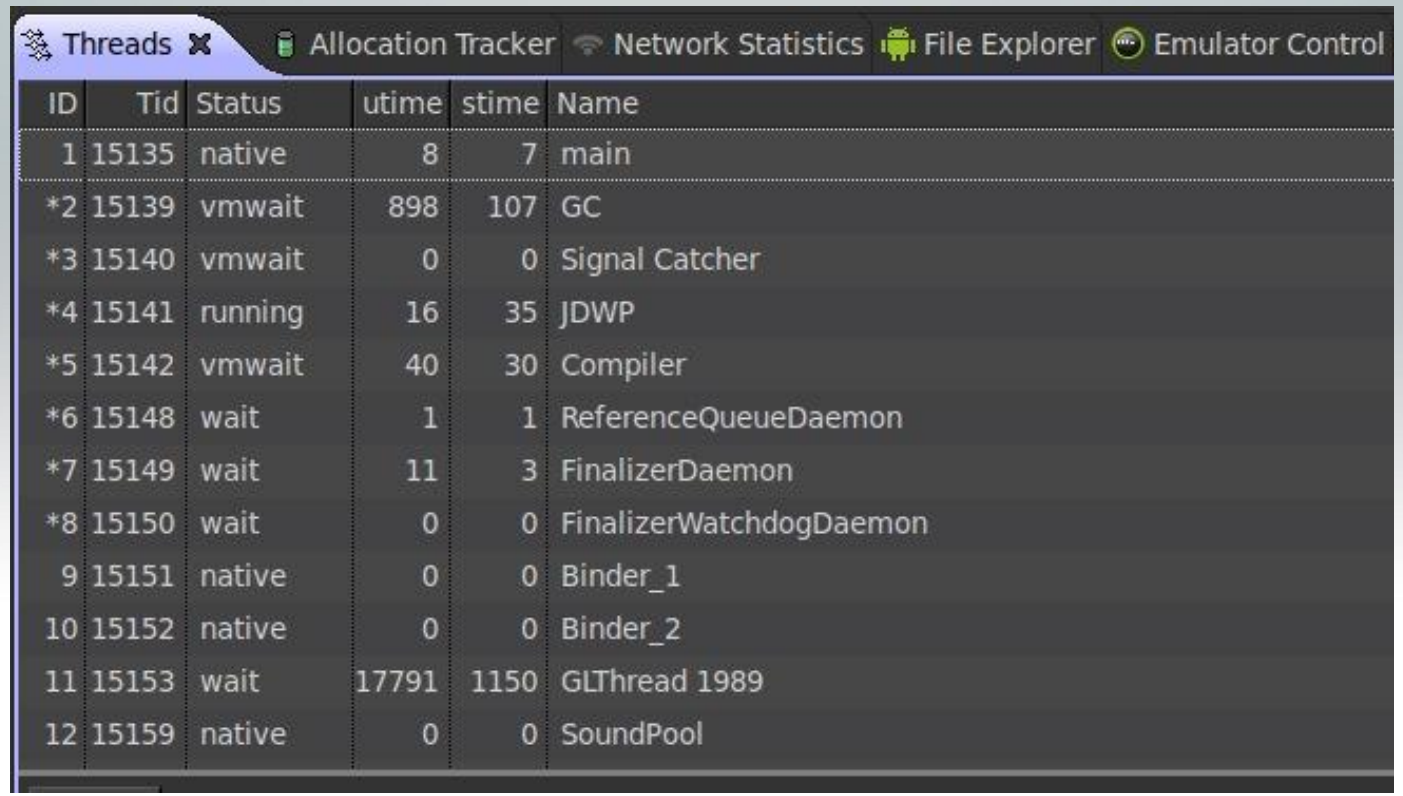


Static analysis: LINT



Beyond static analysis
Dalvik Debug Monitor Server (DDMS)

- DDMS Thread analysis (like "top" but better)

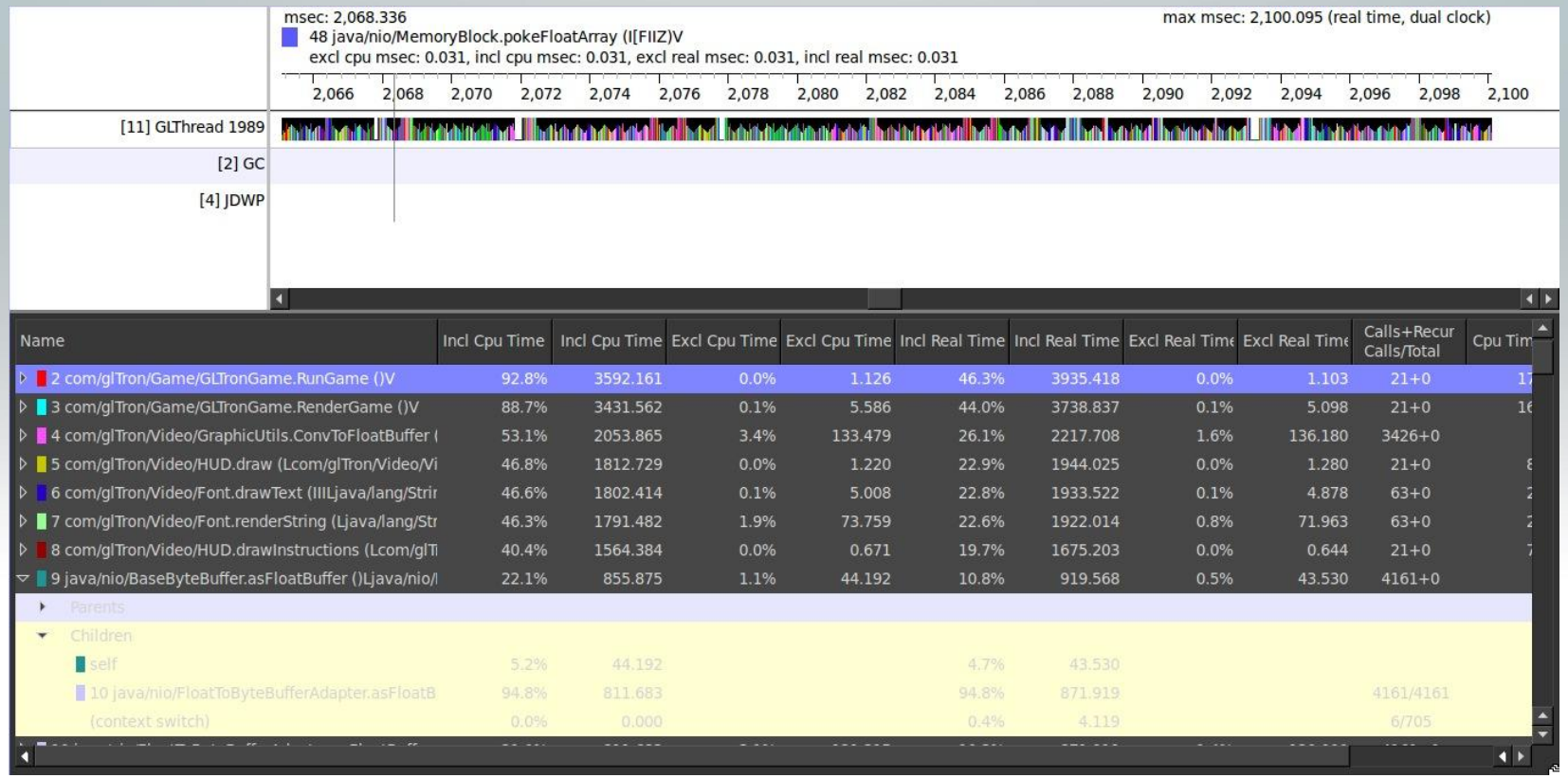


ID	Tid	Status	utime	stime	Name
1	15135	native	8	7	main
*2	15139	vmwait	898	107	GC
*3	15140	vmwait	0	0	Signal Catcher
*4	15141	running	16	35	JDWP
*5	15142	vmwait	40	30	Compiler
*6	15148	wait	1	1	ReferenceQueueDaemon
*7	15149	wait	11	3	FinalizerDaemon
*8	15150	wait	0	0	FinalizerWatchdogDaemon
9	15151	native	0	0	Binder_1
10	15152	native	0	0	Binder_2
11	15153	wait	17791	1150	GLThread 1989
12	15159	native	0	0	SoundPool

DDMS: Traceview

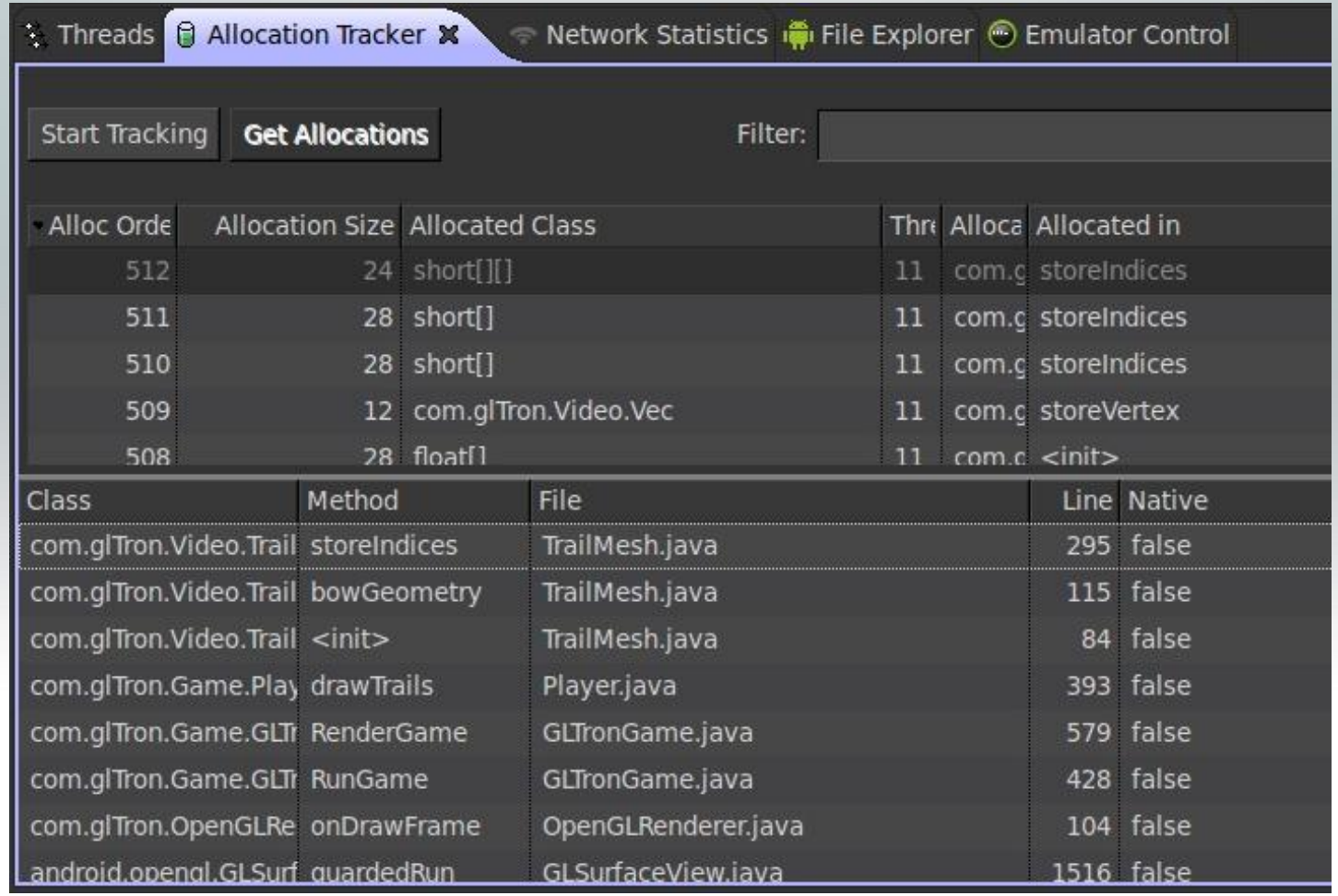
How much CPU time is each method consuming?

- Traceview (start method profiling button)

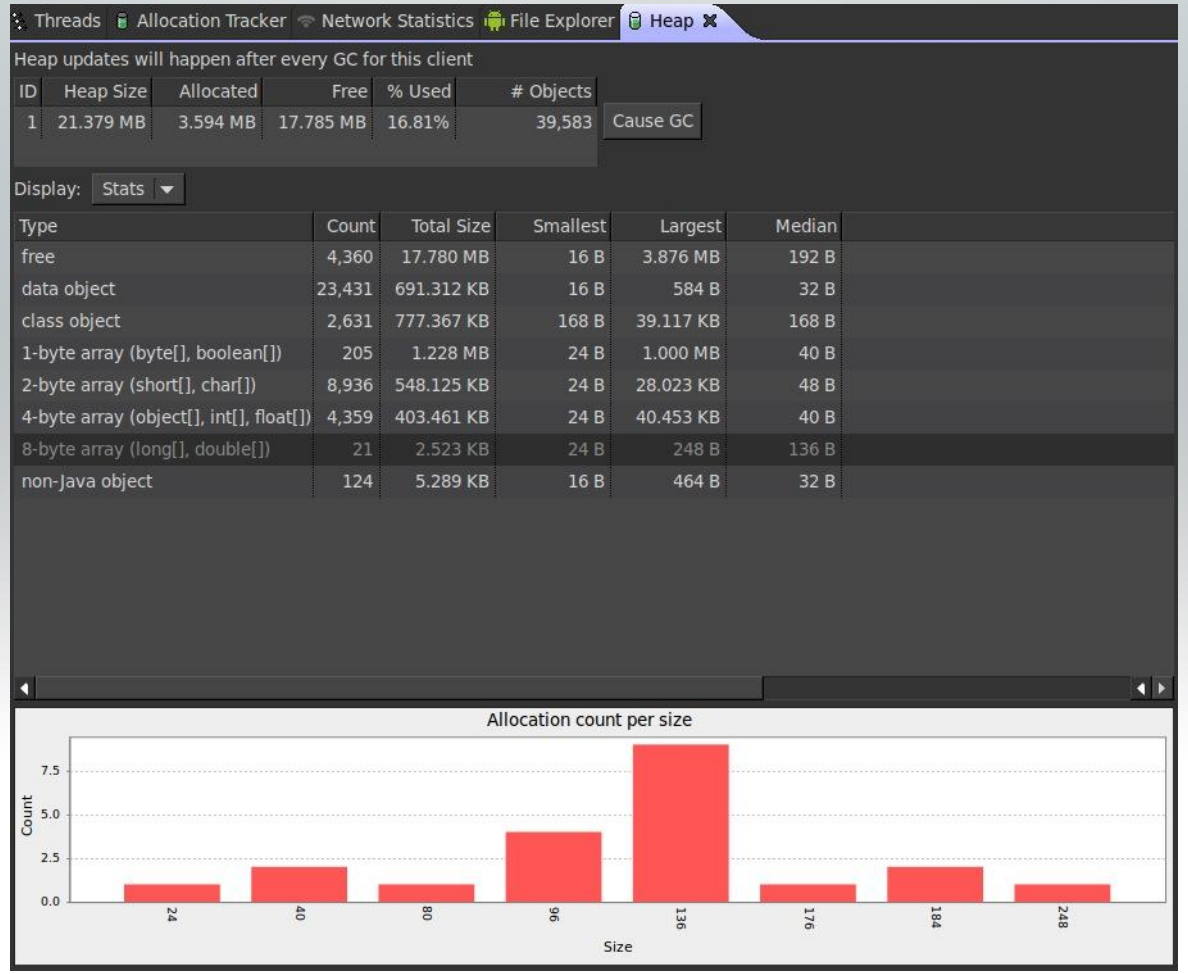


Allocations and HEAP

are you allocating in a high frequency method?

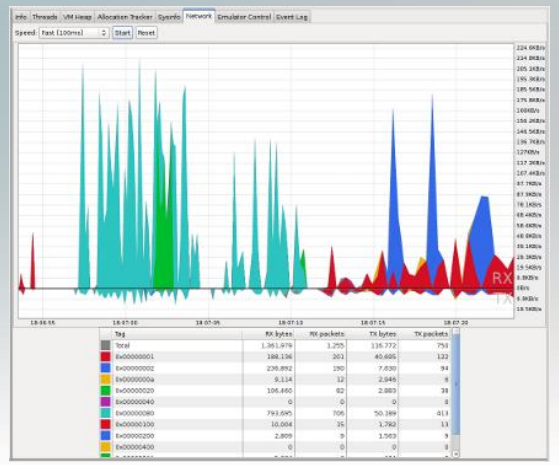


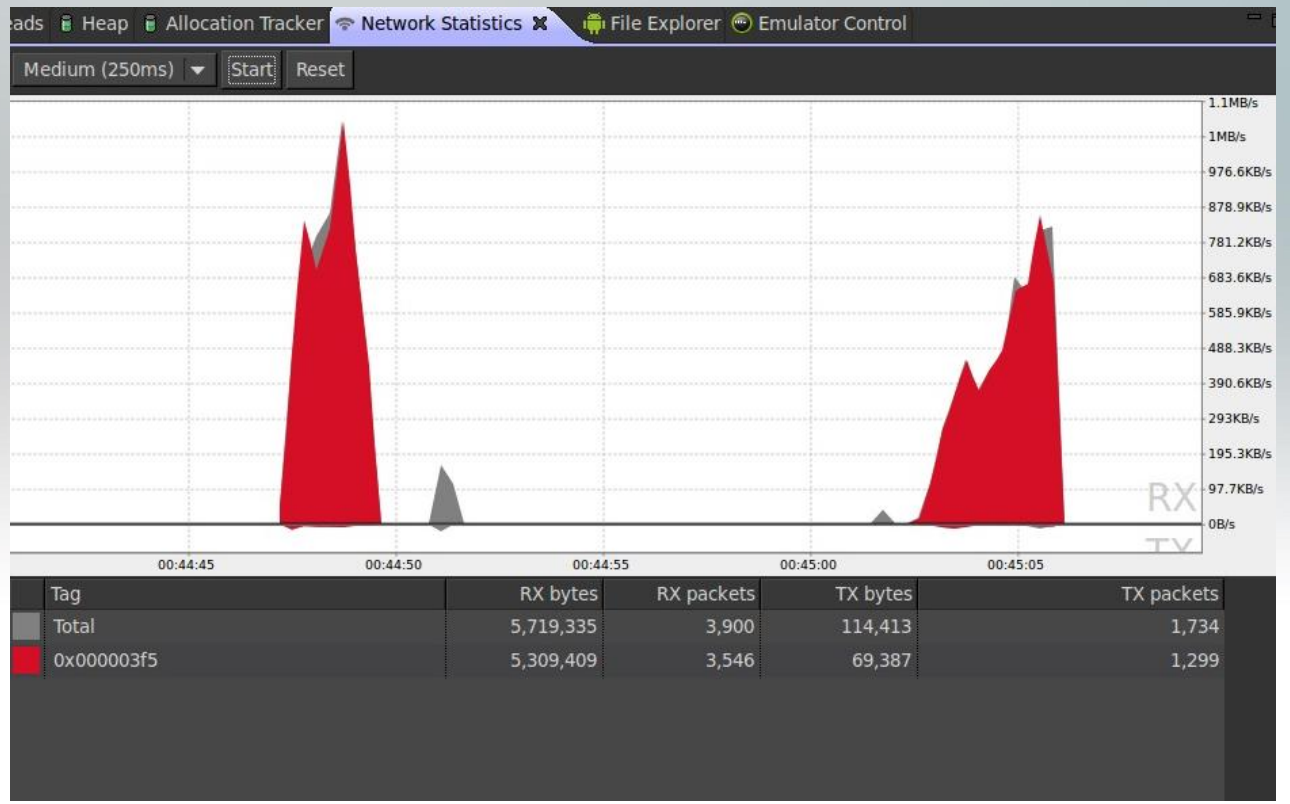
HEAP:
Is your app running out of memory?



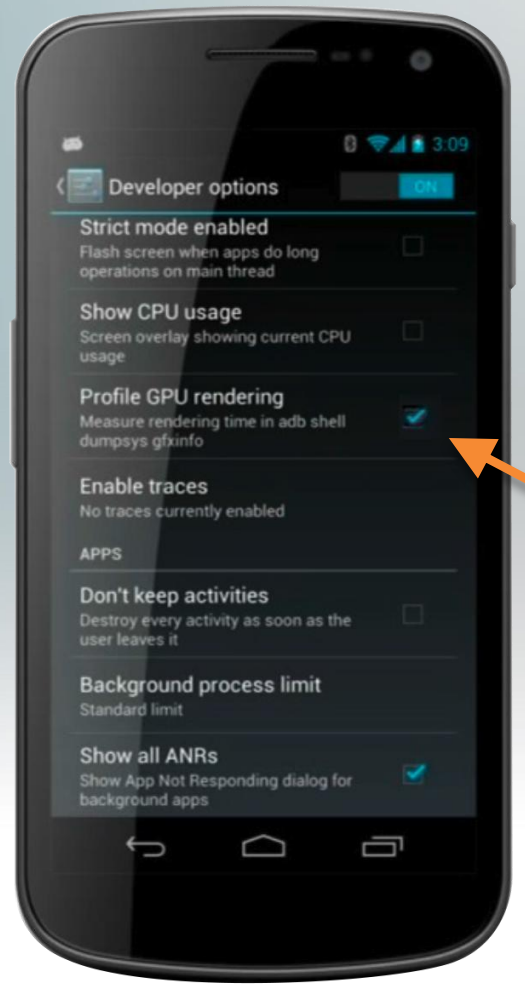
Network Statistics

- Save battery, look for short spikes that can be delayed
- TrafficStats API allows you to tag individual sockets





Adb shell DUMPsys

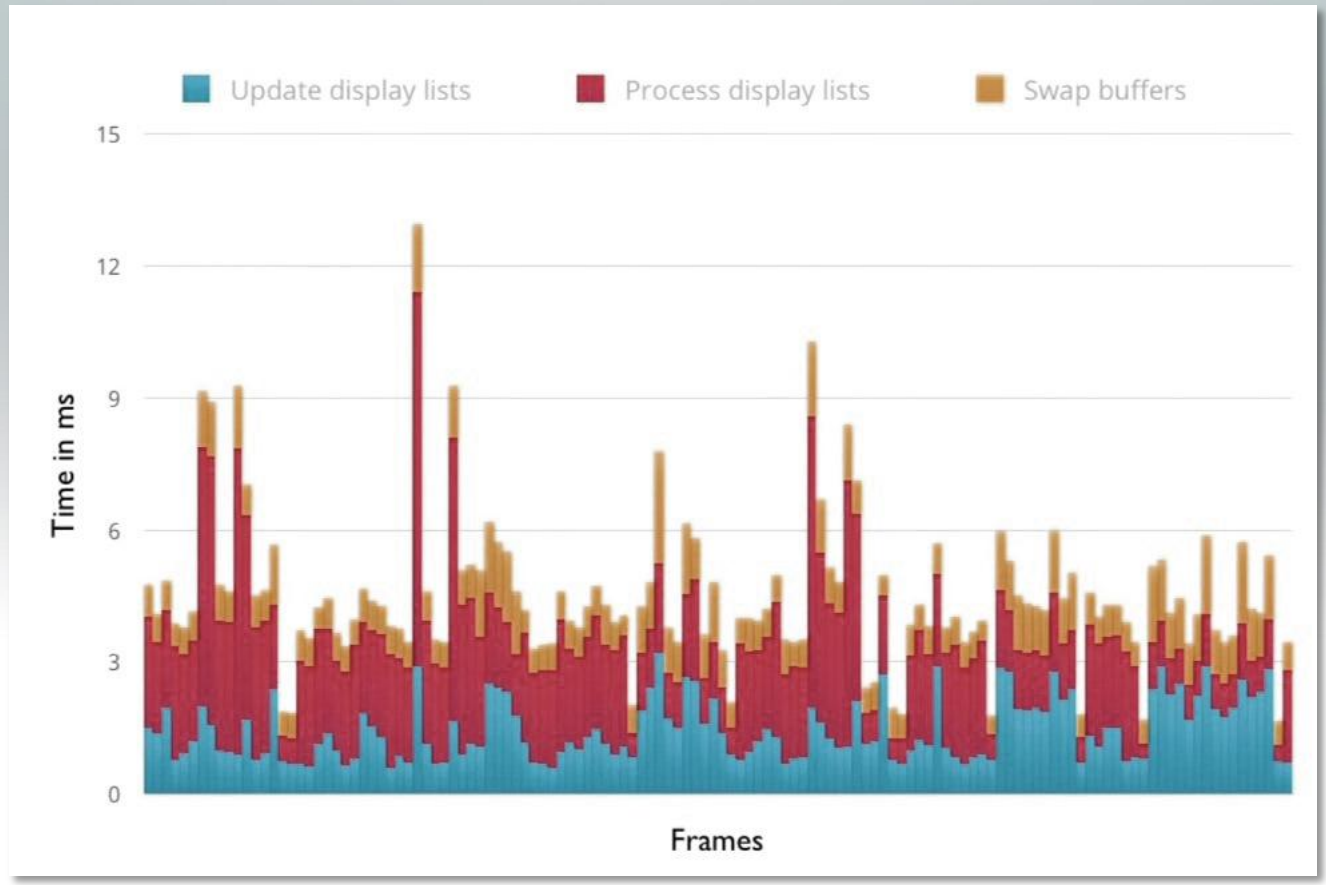


- With dumpsys you can check:
  - Event Hub State
  - Input Reader State
  - Input Dispatcher State
  - any number of other systems
    e.g. dumpsys gfxinfo

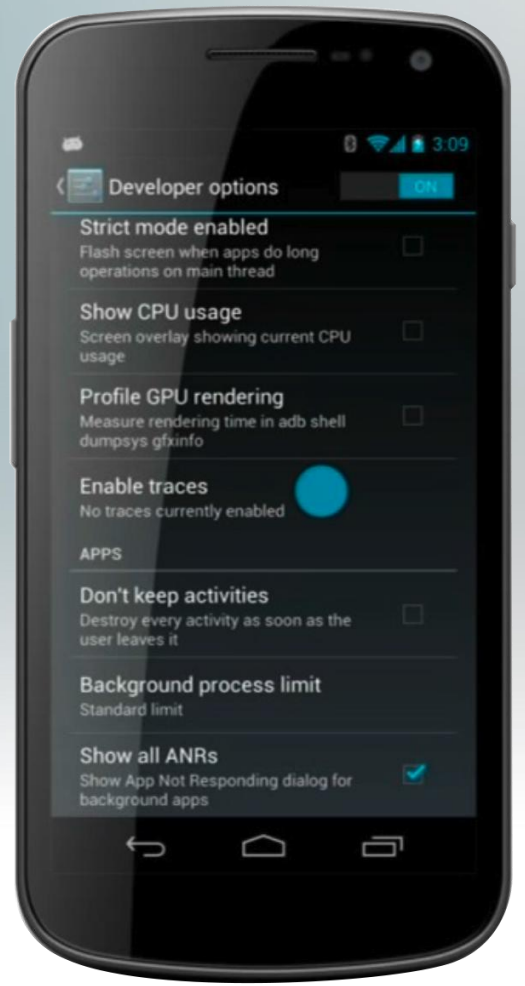
```
Profile data in ms:

        com.android.launcher/com.android.launcher2.Launcher/android.view.ViewRootIm
pl@4161f120
        Draw     Process  Execute
        0.07     0.60     0.92
        0.07     0.68     0.96
        0.10     0.96     1.34
        0.10     0.94     1.20
        0.09     0.94     1.19
        0.10     0.83     1.28
        0.06     0.83     1.21
        0.06     0.61     0.73
        0.05     0.55     0.67
```

Dumpsys gfxinfo

- Drop dumpsys data columns in to a spreadsheet and visualize…
  e.g. Will my animation drop frames?



Systrace



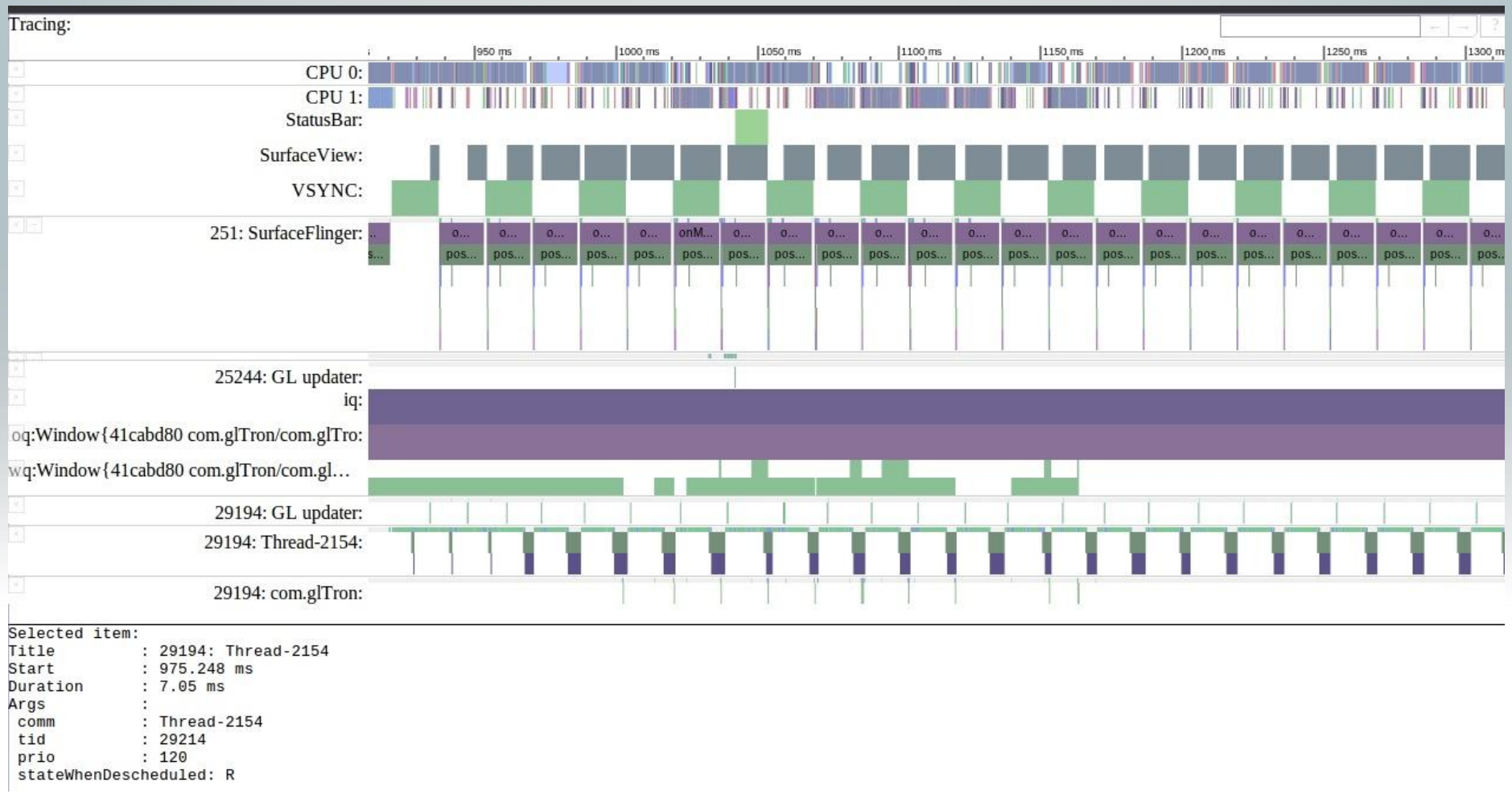
- I've done all I can to analyze inside my app but still can't find the bottleneck.

  Systrace to
  the rescue!

- Systrace.py will generate a
  5 second system level snapshot

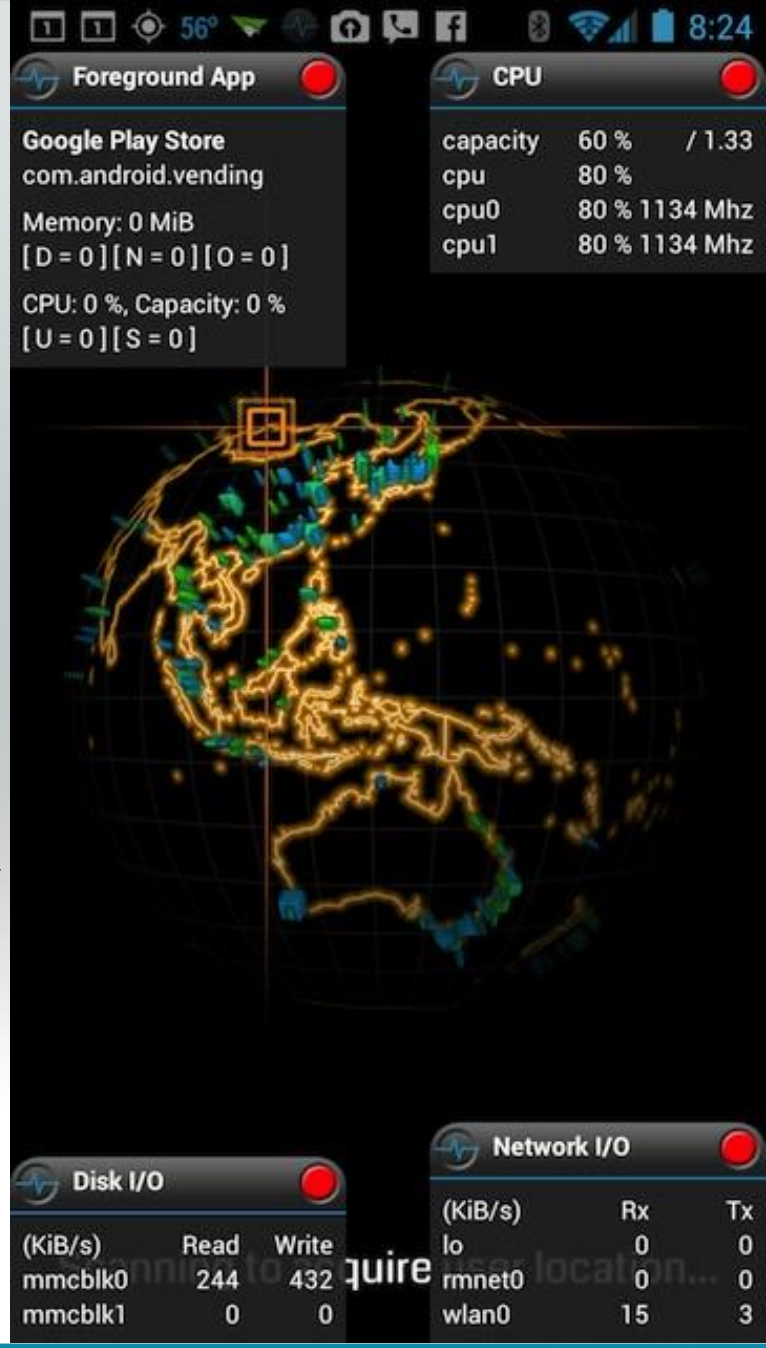
```
android@android ~/android-sdk-linux/tools/systrace $ ./systrace.py
capturing trace... done
downloading trace... done

    wrote file:///home/android/android-sdk-linux/tools/systrace/trace.html

android@android ~/android-sdk-linux/tools/systrace $ xdg-open trace.html
```

Systrace
html5 page of info: Navigate with ‘w’a’s’d’



Other system profilers to consider:

- Chainfire PerfMon App - Free on XDA-Developers
  - Foreground App
  - CPU
  - Disk I/O
  - Network I/O
- From Qualcomm
  - Trepn Profiler App – overlay mode similar to PerfMon but can monitor Android Intents, log states, allows external control and power monitoring.
  - Adreno SDK and Profiler for profiling the Adreno GPU

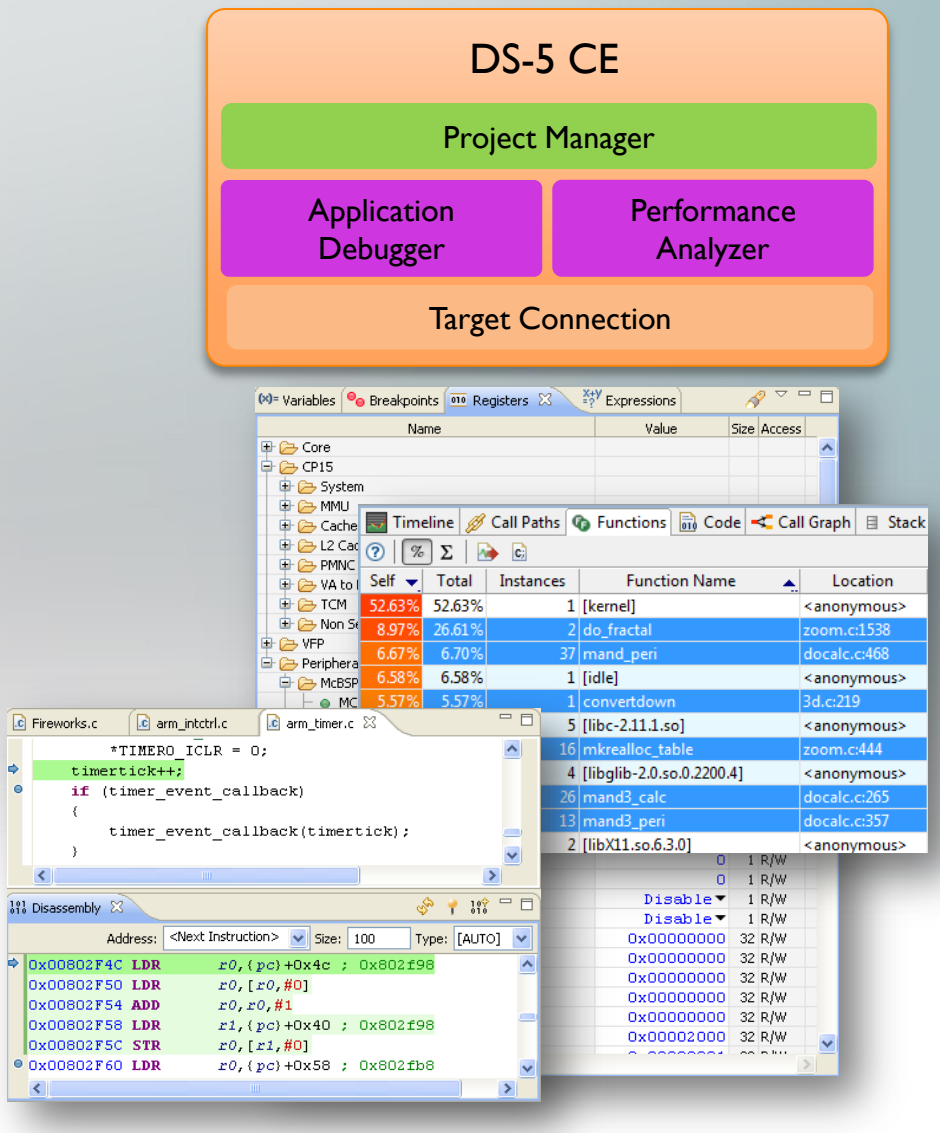


Analyzing native C/C++ (NDK)

- But I didn't use the Java SDK to write my app! How do I analyze my already wicked fast native (or iOS app objective-C port) code?
- What about the Linux kernel part of system analysis?
- Notes of caution
  - Applications that use NDK well will be faster and slicker
  - Ones that don't will be cursed by unhappy users
  - If you build .so libraries for ARM, only ARM devices will be able to run your apps
  - Fortunately not many Android Platforms that aren't ARM
  - Good use of the NDK will narrow the difference between high and low end devices
  - Moving inefficient code to Native doesn't magically make it better code

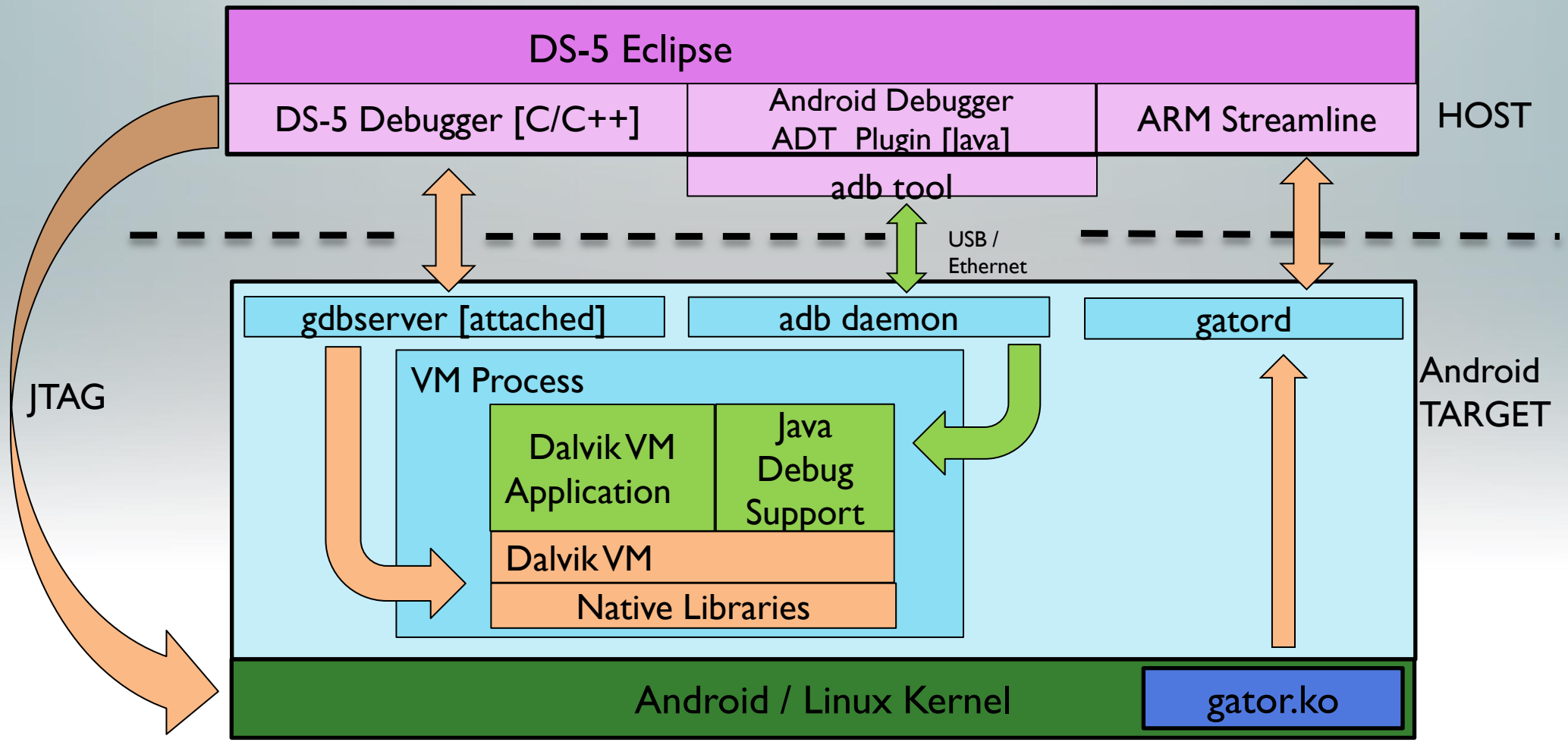
DS-5 CE for Android App Developers

- Friendly, Reliable App Debugger
  - Powerful graphical user interface
  - ADB integration for native debug
  - Java* and native debug in the same IDE
- System-wide Performance Analyzer
  - In-depth system performance statistics
  - Process to function level profiling (native)
- Integrated, validated solution
  - Comprehensive documentation
  - Support via ARM forums
  - Delivered as Eclipse plug-in on arm.com
  - Free of charge



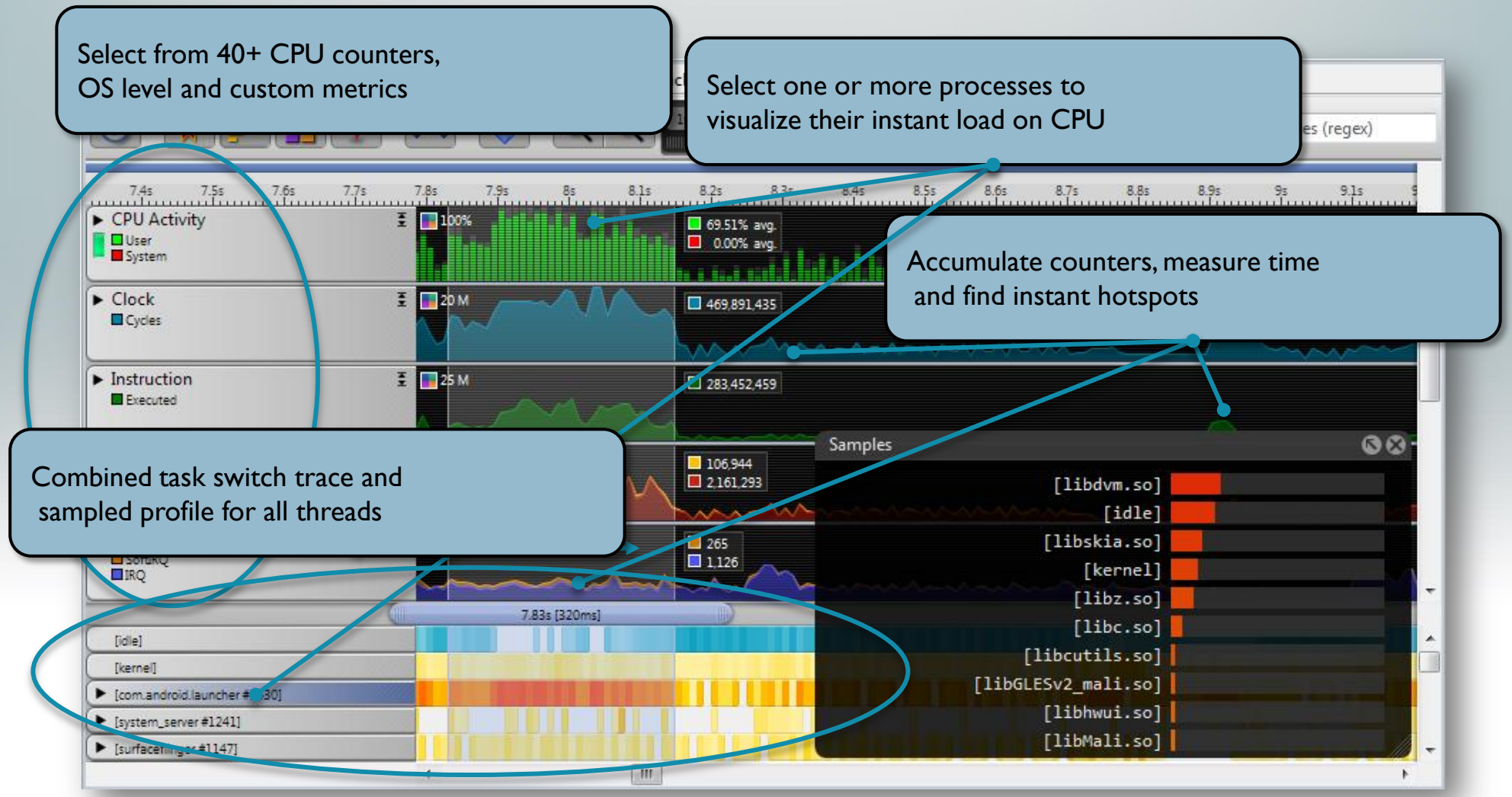
* Java debug for Android requires SDK and ADT

ARM DS-5™ Community Edition
Free Android Native analyzer and debugger

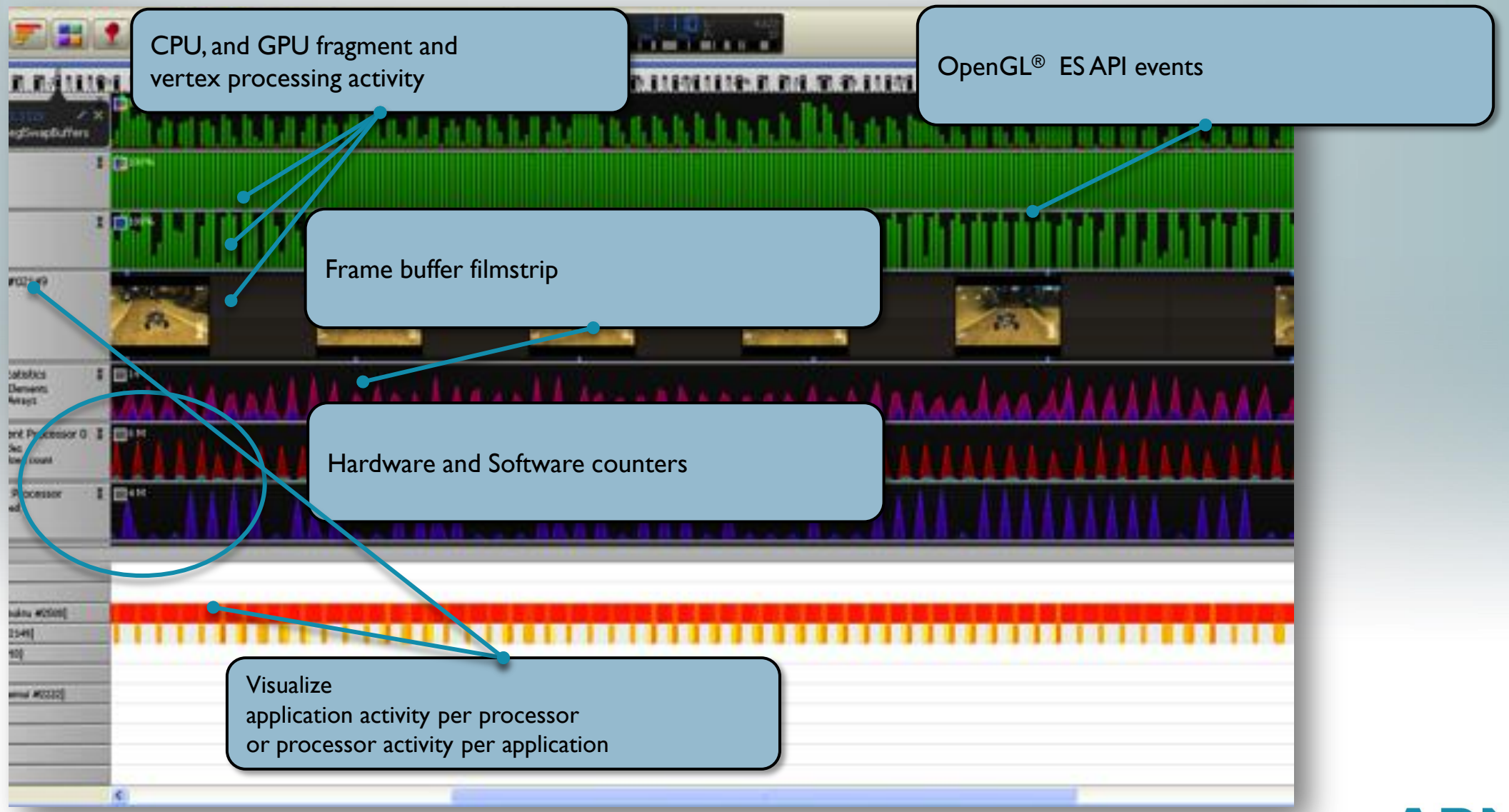


Streamline: The Big Picture

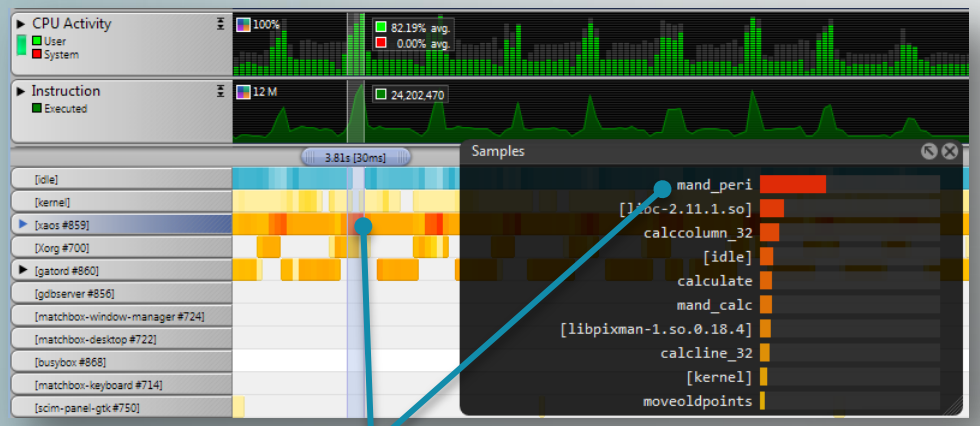
- Find hotspots, system glitches, critical conditions at a glance



Mali GPU Graphics Analysis



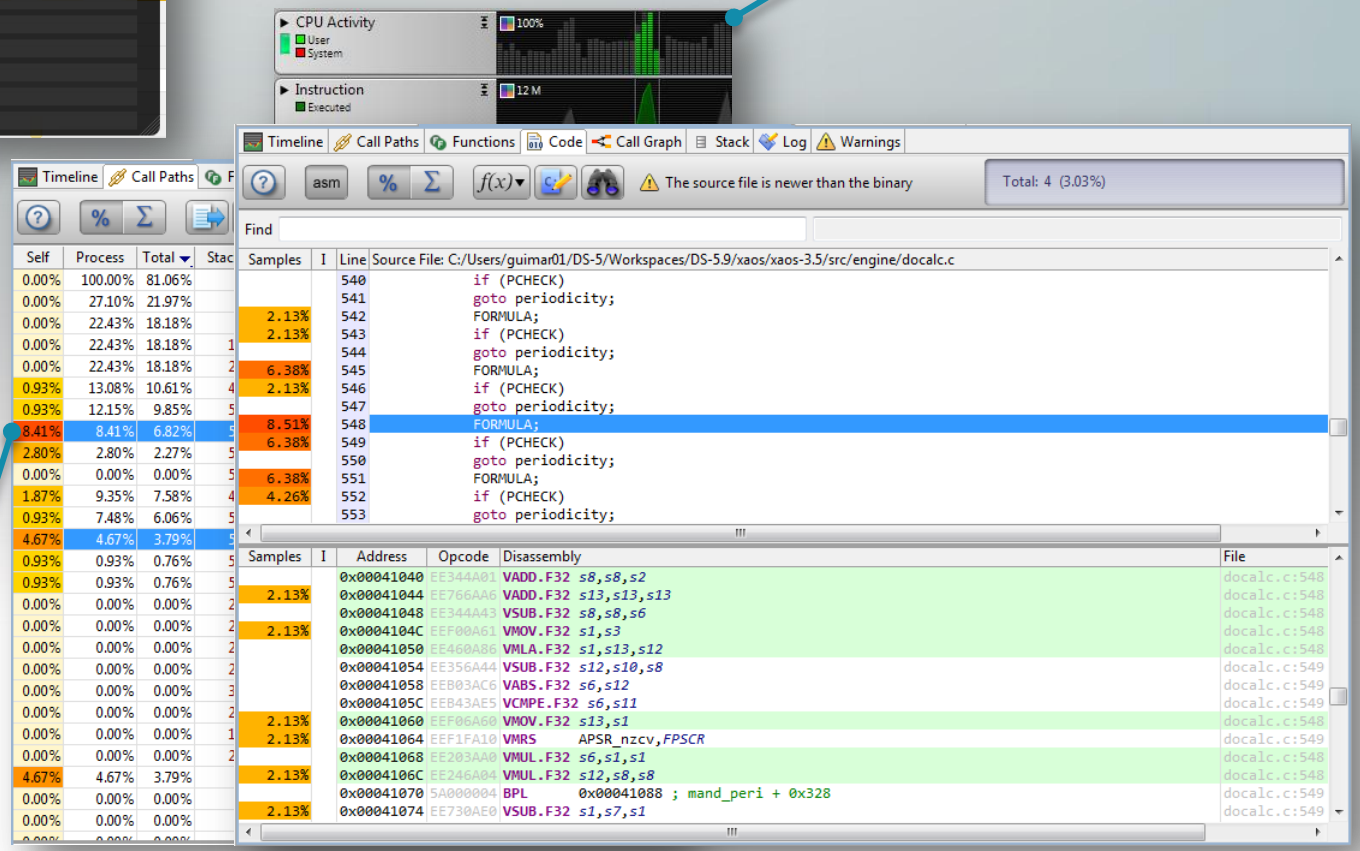
Drilldown Software Profiling



Quickly identify instant hotspots

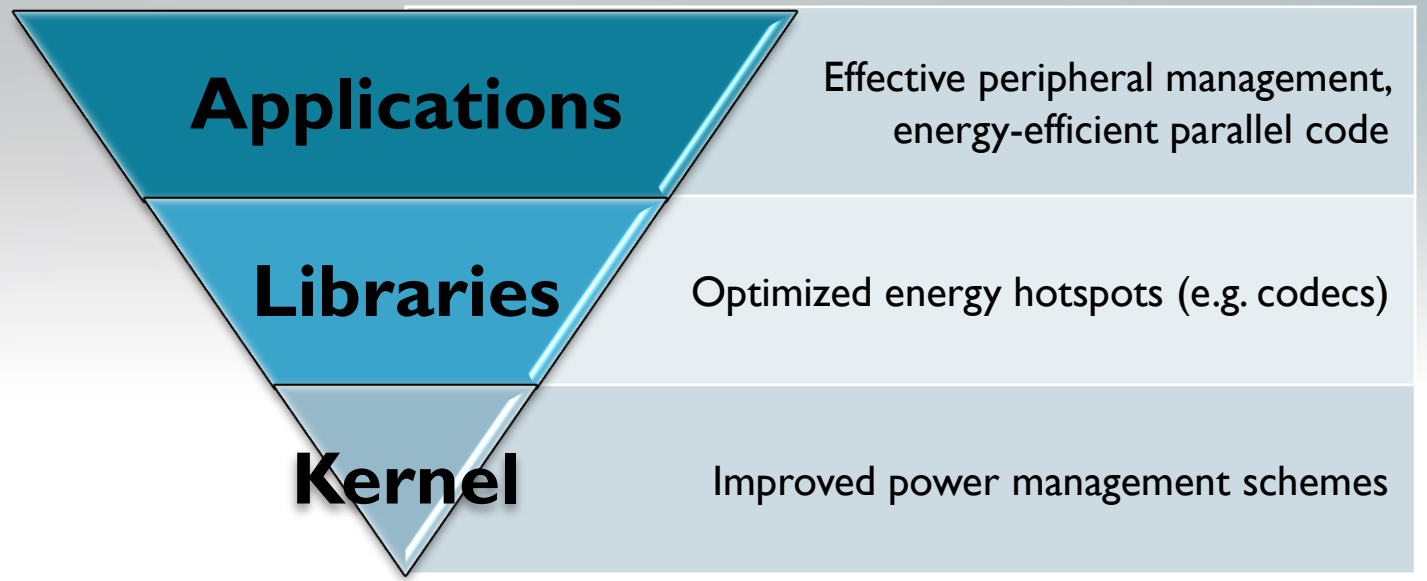
Click on the function name
to go to source code level profile

Filter timeline data to generate
focused software profile reports



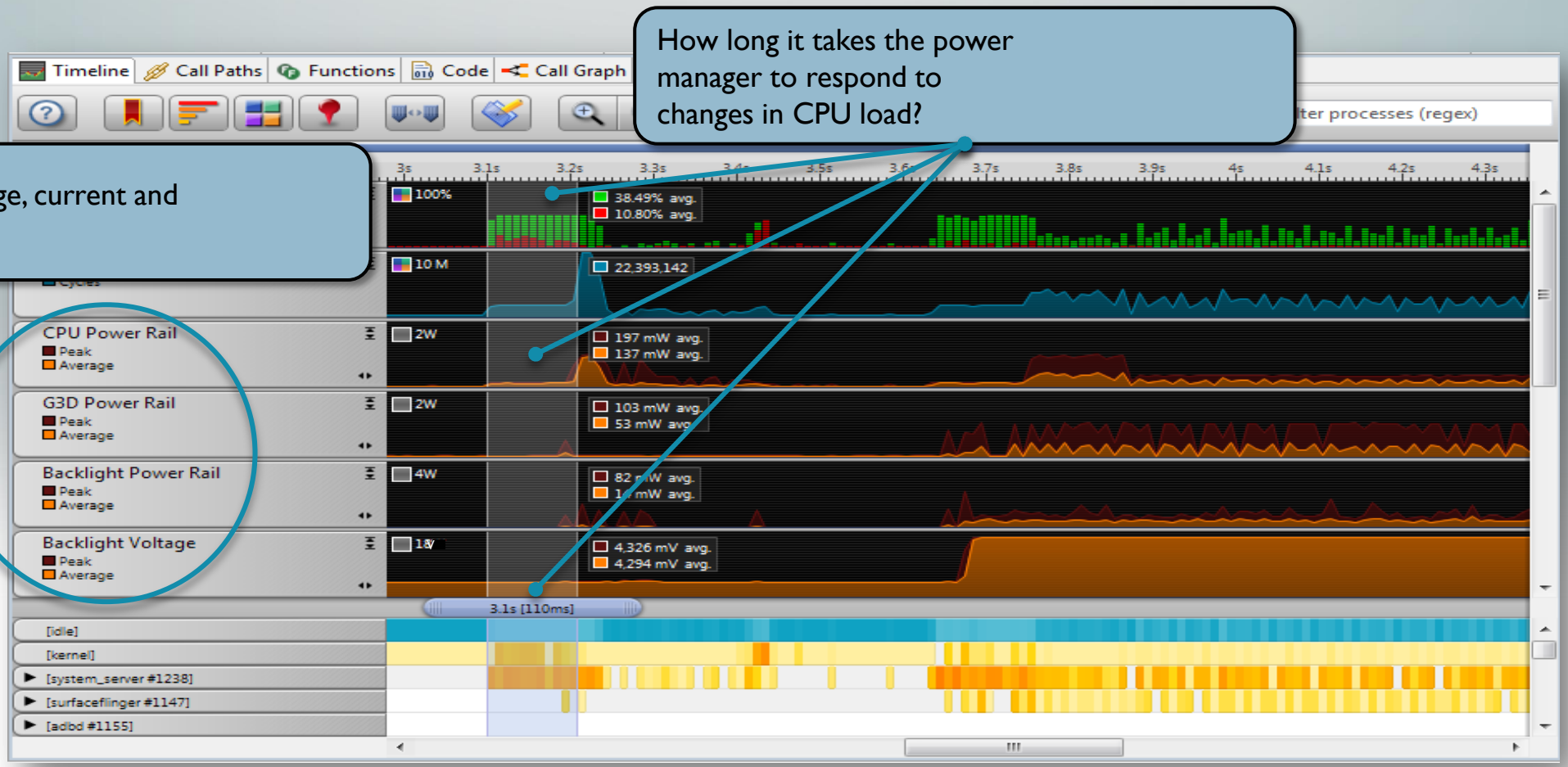
Enabling Energy-Aware Coding

- ARM Energy Probe
  - Lightweight power measurement for software developers
  - Correlates power consumption with software execution in Streamline
  - Monitor up to three voltage rails simultaneously
  - Helps developers to make informed decisions at all layers of the software stack



The *Power* of Having It All in One Place

- How effective are you managing your energy budget?



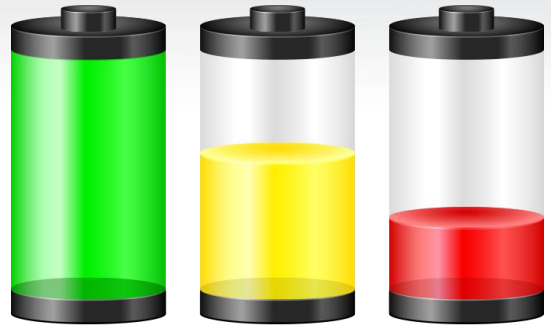
Application Resource Optimizer (ARO)

- Free / Open Source Network-centric diagnostic tool
  - (yes, it is by AT&T but you don't need an AT&T device)
  - Requires root for pcap/data collection
  - APK on device, java desktop app for captured data analysis

Test Your Application **Transfer Trace Files** **Process Trace**

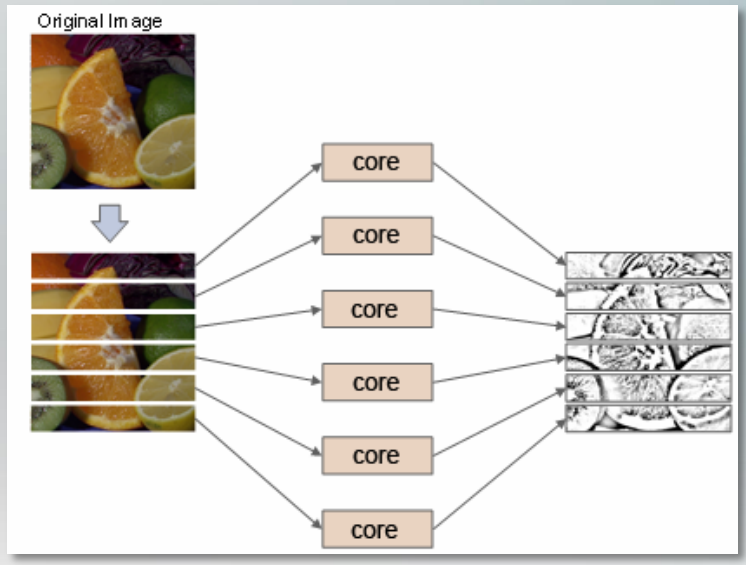
How Can ARO Make Apps Faster?

- The fixes identified by ARO will tune your application to higher performance and speed
  - App-specific Analysis
  - Highlight Key Areas to Improve
  - Increase Network Availability
  - Improve Battery Life
  - Get Faster Response Times
- Simple, common sense development best practices in network environments
  - Reducing connection times
  - Caching files
  - Eliminating errors
- Cross Platform and Network Agnostic



Analysis overload: Fixing the problems…

- **My leading questions:**
- Am I being smart with resources?
- Is this performance bottleneck parallelizable?
- Is this Java or Native? Would it be better the other way around?
- Has this been done before? Don't reinvent the wheel.
- What version of Android should I target?



Networking Resources

- Close Connections
  - >80% of applications do NOT close connections when they are finished
  - 38% more power on LTE (18% more power on 3G)
- Cache Your Data
  - 17% of all mobile traffic is duplicate download of the same unaltered HTTP content (1)
  - "It's just a 6 KB logo" -- 6 KB * 3 DL/session *10,000 users/day = 3.4GB/month
  - Reading from local cache is 75-99% faster than downloading from the web
  - Even if caching IS supported – it is OFF by default
- Manage Every Connection
  - Group your connections
  - Save battery, speed up applications

(1)"Web Caching on Smartphones: Ideal vs. Reality", http://www.research.att.com/~sen/pub/Caching_mobisys12.pdf

Closing Connections: CODE

- MultiRes Sample app from Android SDK

```
HttpURLConnection getimagecloseconn = (HttpURLConnection) urln.openConnection();

getimagecloseconn.setRequestProperty("connection", "close");

getimagecloseconn.connect();
String cachecontrol = getimagecloseconn.getHeaderField("Cache-Control");
InputStream isclose = getimagecloseconn.getInputStream();
        bitmap = BitmapFactory.decodeStream(isclose);
        getimagecloseconn.disconnect();
```

Caching Methods (How do I do it?)

ETags

```
HTTP/1.1 304 Not Modified
Date: Mon, 29 Oct 2012 20:53:03 GMT
Server: Apache
Connection: close
ETag: "2d4f-4cd3729619900"
Expires: Tue, 29 Oct 2013 20:53:03 GMT
Cache-Control: max-age=0
```

- Each file has a Unique Tag
- Revalidated on server for each request
  - High Performance Web Sites: Rule 1 – Make Fewer HTTP Requests (1)
  - Adding a connection drains battery, adds 500-3,000 ms latency

Cache Control Headers

```
HTTP/1.1 200 OK
Date: Mon, 29 Oct 2012 20:51:38 GMT
Server: Apache
Last-Modified: Mon, 28 Jun 2004 00:03:33 GMT
Accept-Ranges: bytes
Content-Length: 27007
Cache-Control: max-age=1437313
Connection: close
Content-Type: image/jpeg
```

- Important to carefully assign Max-Age times
- App will not check file on server until Max-Age is reached
  - Retrieval is strictly file processing time

(1) http://developer.yahoo.com/blogs/ydn/posts/2007/04/rule_1_make_few/

Caching: Worth the Effort?

Android 4.0:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
```

Add this!

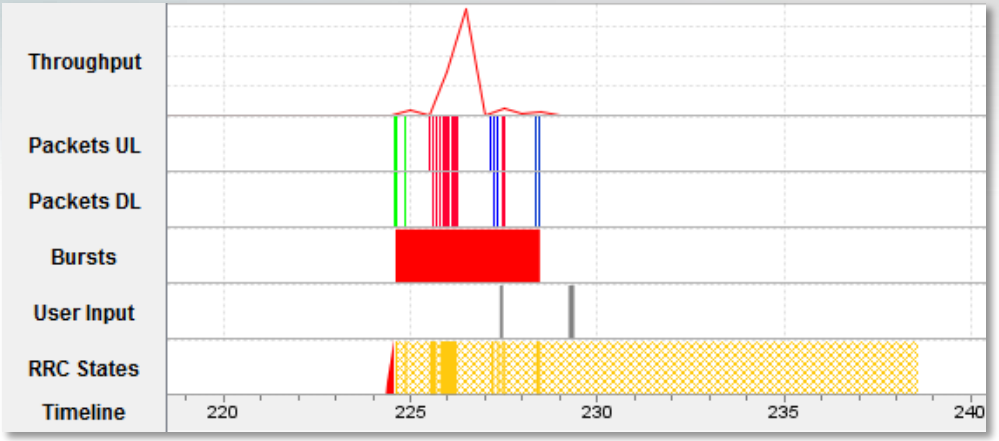
```
    //establish a cache
    try {
        File httpCacheDir = new File(getCacheDir(), "http");
        long httpCacheSize = 10 * 1024 * 1024; // 10 MiB
        HttpResponseCache.install(httpCacheDir, httpCacheSize);
        //
    }
     catch (IOException e) {
        Log.i(TAG, "HTTP response cache installation failed:" + e);
    }
```

Don't leave older devices in the cold: Consider adding reflection for older versions of Android

Grouping Connections

1. Download an image every 60s
2. Download an Ad every 60s
3. Send Analytics to a Server every 60s

Ungrouped: 38J of energy used!!

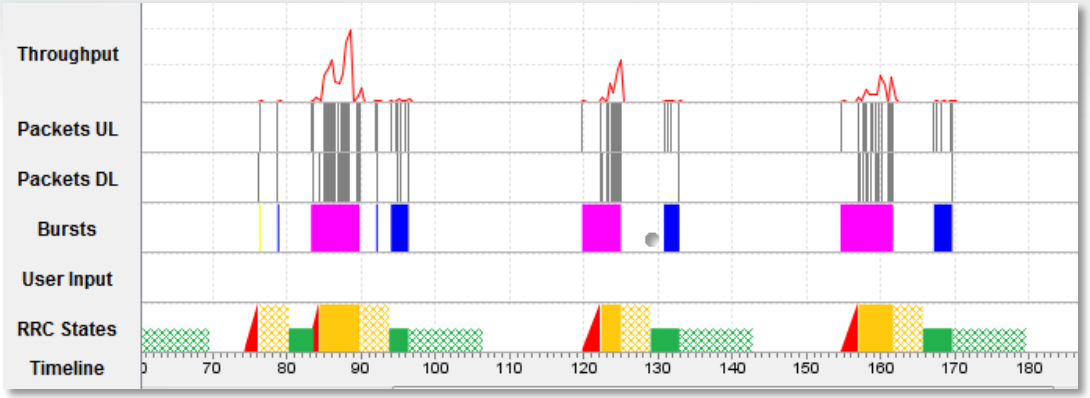


Grouped: 16J of energy used!!
58% savings!

Other best network practices

- Remove redirects to files, they ad ~2-3 seconds per request
- Pre-fetching files that are used often
- Thread file downloads instead of serial download
- No 4xx 5xx http response error codes should occur
- Decouple user feedback from network activity.
- Be careful with periodic connections
  - Regular 3 minute polls for updates could remain connected for 1.2 hours of the day consuming around 20% of your battery.

Ad download every 30s



Going Native (NDK)

http://developer.android.com/sdk/ndk/index.html

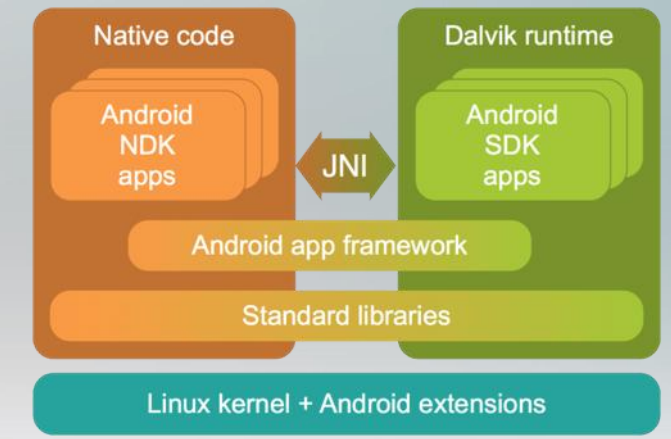
- Native Development Kit is used for writing native C/C++ code and calling your it from within an Android App through Java Native Interface (JNI).

Native Development Kit (NDK) for ARM

- NDK is a comprehensive tool kit to enable application developers to write directly for the ARM processor

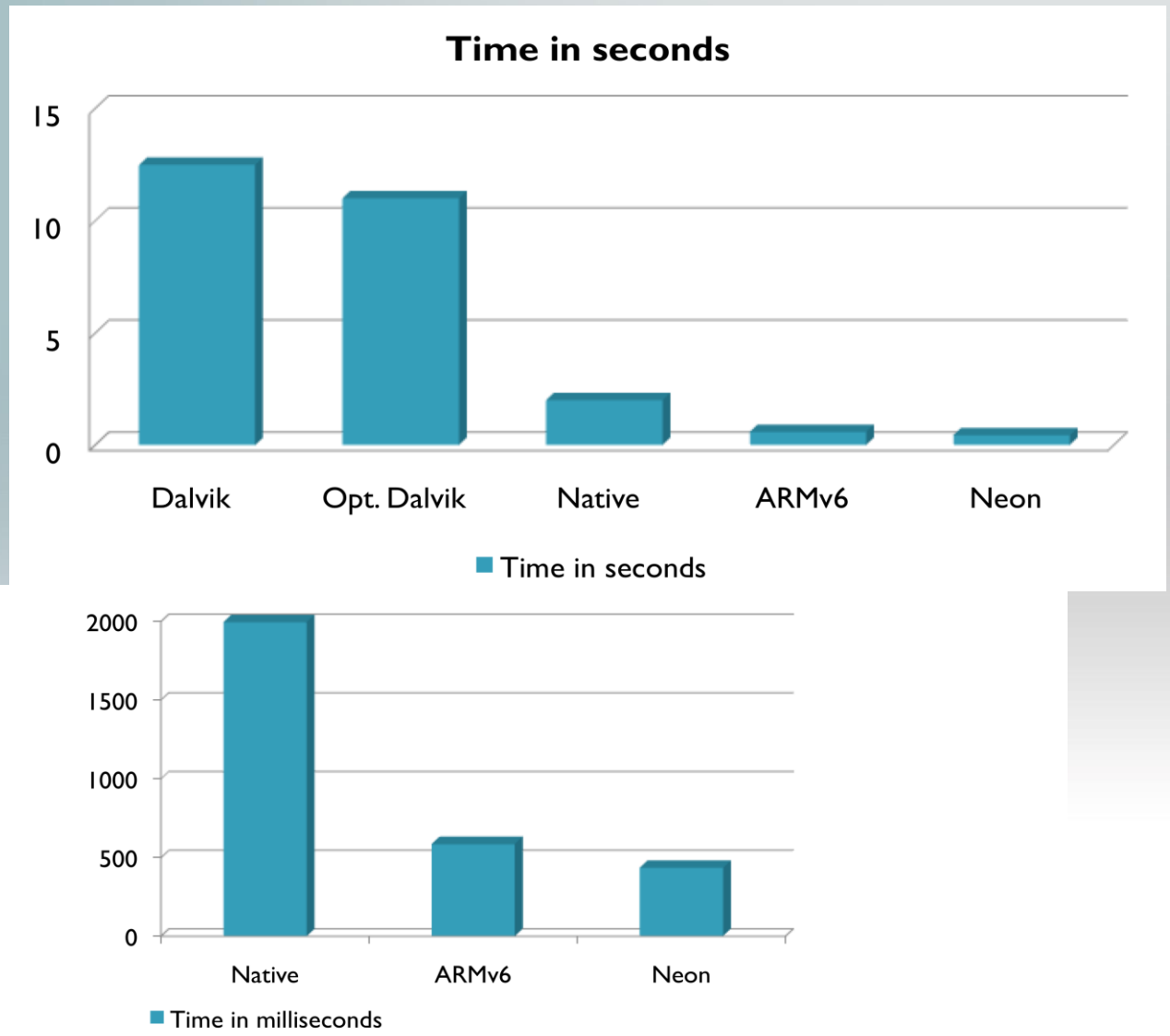
ARM Improvements	▪ Improved *performance* and *code density* with GCC 4.4.3 ▪ Optimizations for Cortex-A9 ▪ Support for VFPv3
General highlights	▪ New *NativeActivity* feature eliminates need to write Java ▪ Addition of default C++ STL ▪ New API's ▪ Input subsystem, sensor data ▪ Windows, surface subsystem ▪ OpenSL ES Audio API ▪ Access to APK graphics assets ▪ EGL library to create and manage OpenGL ES textures and services

NEON supported since the r5 release



Android™ applications can be written in Java, native ARM code, or a combination of the two

Benchmark Results - specific media intensive test case



Just Native

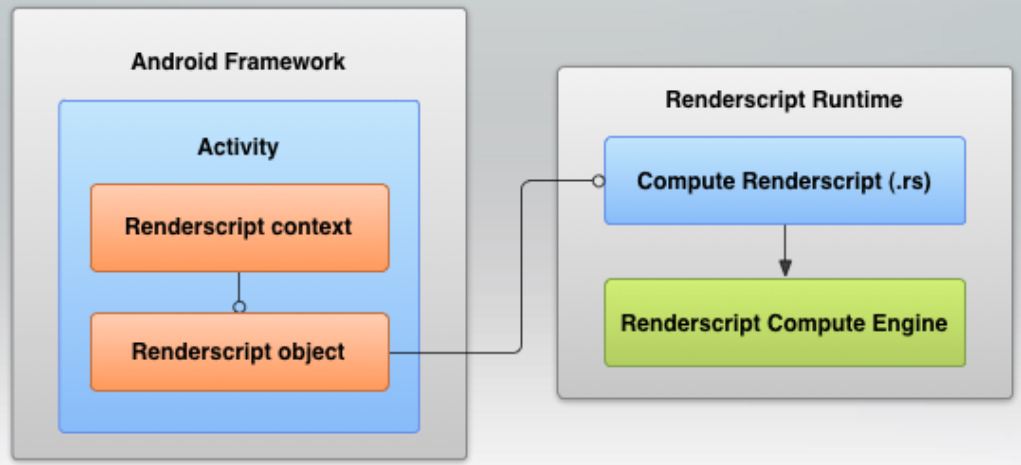
For more info on NDK, see my webinar at:
http://goo.gl/GTwPH

SMP and parallelization

- Nearly every Android and mobile device on the market today is multicore and the trend will continue – Design multi-threaded apps
- Davlik Java threads and IPC
  - AsyncTask is often the simplest way to quickly push a task onto a background worker thread with little IPC complexity
- Bionic C library implements a version of the Pthreads API
  - most of the pthread_* and sem_* functions are implemented but no SysV IPC
  - if it is declared in pthread.h or semaphore.h, it will mostly work as expected

SMP and parallelization

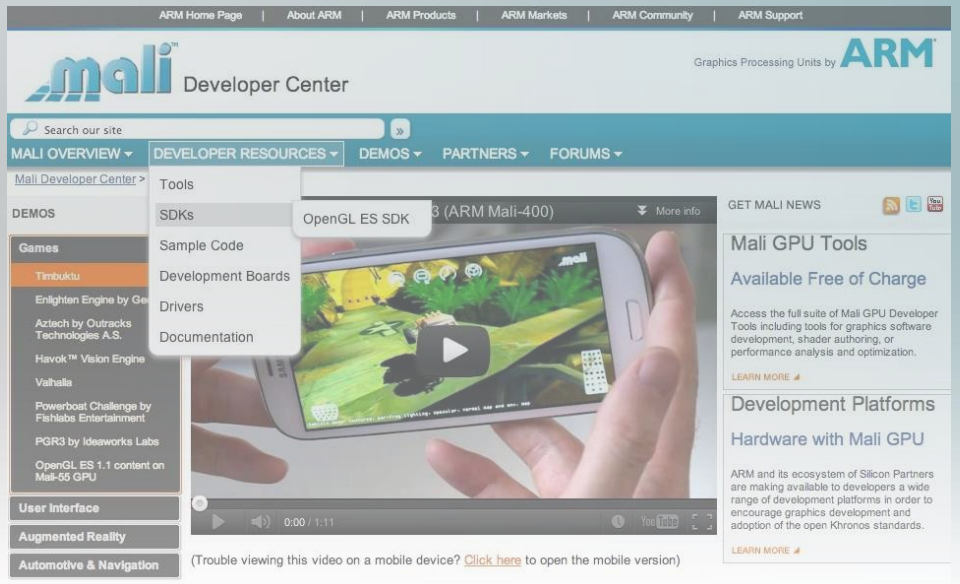
- GPU Compute: Renderscript Compute offers a high performance computation API at the native level
  - Write in C (C99 standard)
  - Run operations with automatic parallelization across all available processor cores
  - Platform independent



- Simpler to use than you might expect.
  - Your Renderscript code resides in .rs and .rsh files in the <project_root>/src/ directory
  - Call forEach_root() with your renderscript function, input and output allocations.
  - See: developer.android.com/guide/topics/renderscript

SMP and parallelization

- OpenGL® ES 2.0 enables full programmable 3D graphics for programmable embedded GPUs
  - Royalty-free, cross-platform API
  - 2D and 3D graphics
  - Supported in both Android's framework API and the NDK
- malideveloper.arm.com
  - OpenGL ES SDK and Sample Code
  - Shader libraries and complier
  - Texture compression and ASTC Codec
  - Asset compiler and conditioning tools
  - OpenGL ES 2.0 and 3.0 emulators



- Full profile OpenCL™ is an option with some GPUs and possible to use in Linux but not supported by Google in Android. Developer beware.

Write java for mobile/embedded/battery

- **new**
  - Don’t call this. Ever.
  - At least not in CPU bound/frequent activities
  - Try to use static variables or only allocate upfront or at natural pauses in activity
  - Avoid triggering Garbage Collection (use DDMS)
    watch Google IO 2009: http://goo.gl/7xCMg
- In JellyBean use features for graphics like
  - android.view.Choreographer for v-sync pulses
  - myView.postInvalidateOnAnimation()
  - don't draw stuff that won't be displayed c.quickReject(items…), Canvas.EdgeType.BW

Android Dev Pro Tips – New API tips from IO 2013

- Use Google Cloud Messaging to get notified of new info to synchronize rather than time based polling. public class MySyncAdapter extends AbstractThreadedSyncAdapter {}
  - GCM allows for persistent XMPP connections.
  - You can use to upstream data now.
- Fused Location Provider Uses Wifi and accelerometer for indoor position & GPS outside
  - Don't worry about monitoring Wifi, GPS and accelerometer yourself.
  - HIGH_ACCURACY – updates every 5 sec, 7.25%/hr bat drain
  - BALANCED_POWER – updates on 20 sec interval, 0.6%/hr
  - NO_POWER mode – accurate to 1 mile
- GeoFencing: Rather than poll a users location, just setup a fence.
  - addGeoFence saves 2/3rds power over addProximityAlert()…

SIMD: NEON

- General purpose SIMD processing useful for many applications
- Supports widest range multimedia codecs used for internet applications
  - Many soft codec standards: MPEG-4, H.264, On2 VP6/7/8, Real, AVS, …
  - Supports all internet and digital home standards in software
- Fewer cycles needed
  - NEON will give 1.6x-2.5x performance on complex video codecs
  - Individual simple DSP algorithms can show larger performance boost (4x-8x)
  - Processor can sleep sooner => overall dynamic **power saving**
- Straightforward to program
  - Clean orthogonal vector architecture
  - Applicable to a wide range of data intensive computation.
  - Not just for codecs – applicable to 2D/3D graphics and other processing
  - 32 registers, 64-bits wide (dual view as 16 registers, 128-bits wide)
  - Off-the-shelf Tools, OS, commercial & open source ecosystem support

Don't Reinvent the wheel! NEON in Open Source Today

- **Google WebM** – 11,000 lines NEON assembler!
- **Bluez** – official Linux Bluetooth protocol stack
- **Pixman** (part of cairo 2D graphics library)
- **ffmpeg (libav) – libavcodec**
  - LGPL media player used in many Linux distros and products
  - Extensive NEON optimizations
- **x264** – Google Summer Of Code 2009
  - GPL H.264 encoder – e.g. for video conferencing
- **Android** – NEON optimizations
  - **Skia** library, S32A_D565_Opaque **5x** faster using NEON
  - Available in Google Skia tree from 03-Aug-2009
- **LLVM** – code generation backend used by Android RenderScript
- **Eigen2** – C++ vector math / linear algebra template library
- **TheorARM** – libtheora NEON version (optimized by Google)
- **libjpeg / libjpeg-turbo** – optimized JPEG decode
- **libpng** – optimized PNG decode
- **FFTW** – NEON enabled FFT library
- **Liboil / liborc** – runtime compiler for SIMD processing
- **webkit** – used by Chrome Browser

How to use NEON

- Opensource libraries, e.g. OpenMAX, libav, libjpeg, Android Skia, etc.
  - **Freely available Open Source optimizations**
- Vectorizing Compilers
  - Exploits NEON SIMD automatically with existing source code
  - **Status:** Released (in DS-5 armcc, CodeSourcery, Linaro gcc and now LLVM)
- C Instrinsics
  - C function call interface to NEON operations
  - Supports all data types and operations supported by NEON
  - **Status:** Released (in DS-5 and gcc)
- Assembler
  - For those who really want to optimize at the lowest level
  - **Status:** Released (in DS-5 and gcc/gas)
- Commercial vendors
  - Optimized and supported off-the-shelf packages

What is Project Ne10?

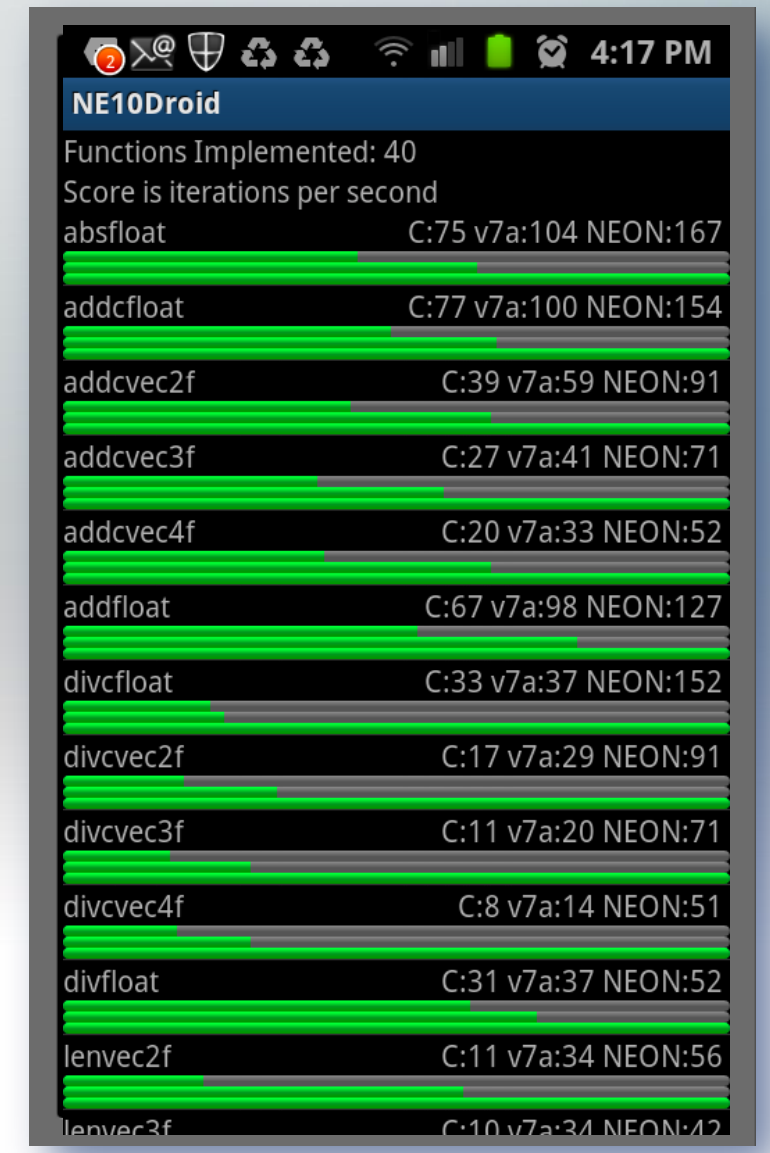
- NE10 is designed to provide a set of common, useful functions which
  - have been optimised for ARMv7 and NEON
  - provide consistent well tested behaviour
  - and that can be easily incorporated into applications
  - Is targeted at Android and Linux to maximize app performance
- Features
  - Usable from C/C++ and Java/JNI
  - The library is modular; functionality that is *not* required within an App can be discarded
  - Functions similar to the Accelerate Framework provided by iOS

Why use Project Ne10?

- It is Free
  - No commercial complications- 'build and ship' BSD License
  - No liability offered from ARM, no money paid to ARM
  - well-tested behavior with example code
- Use of the Ne10 library should be a joy, not a chore
  - Out-of-box and user experience is critical to success
  - Build and go, accessible documentation, clear code
  - Code promotes the ***best*** of the ARM Architecture- build on it
  - Lets ***you*** get the most out of ARMv7/NEON without arduous coding
  - Supported by ARM, community contributions welcome

Ne10Droid – The App in action

- NE10Droid is a benchmarking Android App that uses NE10.
- Routines are written using VFP in C, VFP in Assembly and NEON.
  - Example routines:
  - `arm_result_t normalize_vec2f(arm_vec2f_t * dst, arm_vec2f_t * src, unsigned int count);`
  - `arm_result_t normalize_vec3f(arm_vec3f_t * dst, arm_vec3f_t * src, unsigned int count);`
  - `arm_result_t normalize_vec4f(arm_vec4f_t * dst, arm_vec4f_t * src, unsigned int count);`



Conclusions

- For the Simple, Quick-to-Market option, stick with Dalvik but consider JellyBean's tools and NDK options
  - Can always optimize in version 1.1
- Be smart about your network resources
- Why write code you don't have to?
  - Look for highly optimized code with a compatible license
  - Can be beneficial without being a perfect fit
- Ideal candidates for Threads, NEON and GPU Compute: audio, image, video and game code
- There is a lot of extra performance there if you really need it
  - Particularly if you can use ARMv6, v7 extensions (NEON)
  - Learning new stuff is fun, so experiment

Solution Center for Android

- The SCA offers developers the widest range of Android resources for ARM architecture. Over 200 ARM Connected Community members come together to share their Android development expertise, solutions and services, including:
  - Development tools
  - Resources for building devices
  - Porting Guides
  - White papers
  - Android training
- community.arm.com/groups/android-community
- arm.com/solution-center-android
- androidtools.org
- projectNe10.org
- malideveloper.arm.com

Extra slides:

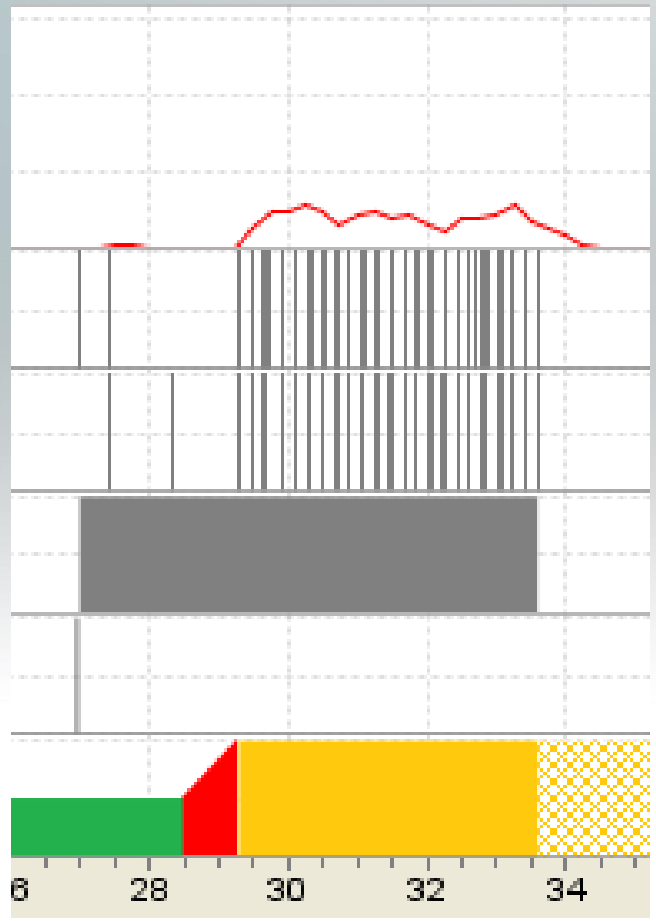
- Extra slides:
- Link to this presentation

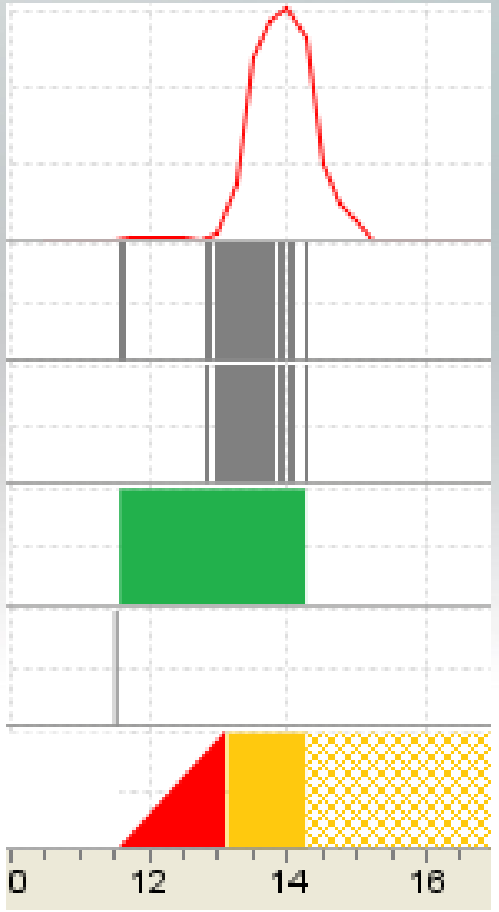
How Do I Group Connections?

```
if (Tel.getDataActivity() >0){
        if (Tel.getDataActivity() <4){
        //ok, we are passed the minimum time to check
        //and we found network activity-
        //download the image here using image getter
         imagegetter(counter, numberofimages);
      //and show the ad
              AdRequest adRequest = new AdRequest();
              adRequest.addTestDevice(AdRequest.TEST_EMULATOR);
              adView.loadAd(adRequest);
            // Initiate a generic request to load it with an ad
              adView.loadAd(new AdRequest());
```

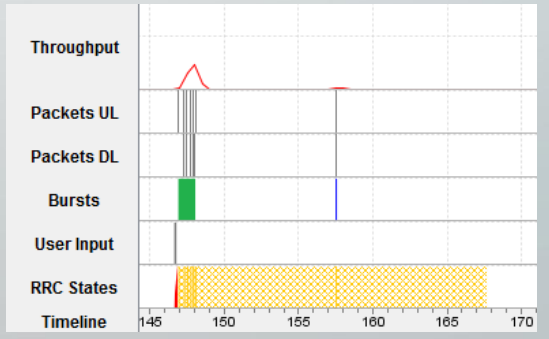
Grouping Connections for Speed

- Threading file downloads vs. serial download





Closing Connections: Example



38% more power on LTE!
(18% more power on 3G)

Time	Direction	Type	Payload Length
190.284	DOWNLINK	DATA	1257
190.284	UPLINK	ACK	0
190.288	UPLINK	CLOSE_CONN	0

Time	Direction	Type	Payloa
147.873	UPLINK	ACK	0
147.873	UPLINK	ACK	0
148.018	DOWNLINK	DATA	1296
148.037	UPLINK	ACK	0
157.484	DOWNLINK	CLOSE_CONN	0
157.556	UPLINK	ACK	0

What's in Ne10 today?

- Absolute value, multiply and accumulate and other arithmetic operations of floating point arrays with scalar and constant values along with utility functions.
- DSP FIR/IIR, CFFT/CIFFT and RFFT/RIFFT functions
- SIMD Component-wise Arithmetic on Two Vectors
- Normalize up to 4 dimensional vectors of the input array and store them in the corresponding elements of the output array.
- $$\hat{\mathbf{a}} = \frac{\mathbf{a}}{\|\mathbf{a}\|} = \frac{a_1}{\|\mathbf{a}\|}\mathbf{e_1} + \frac{a_2}{\|\mathbf{a}\|}\mathbf{e_2} + \frac{a_3}{\|\mathbf{a}\|}\mathbf{e_3}$$

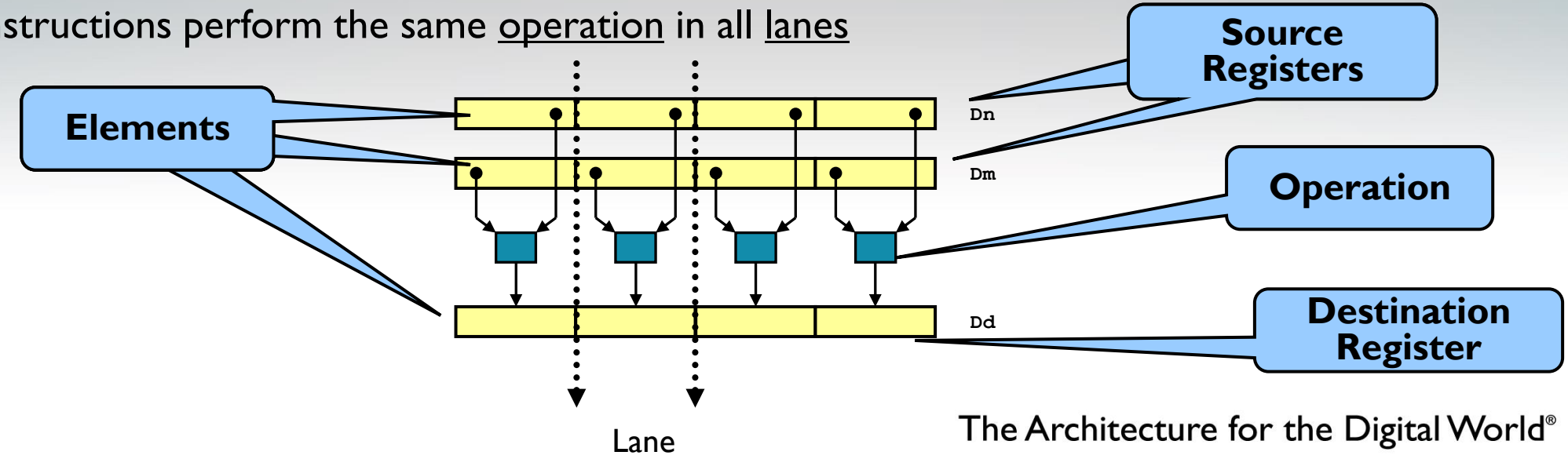
  *(*normalize_vec3f)(arm_vec3f_t * dst, arm_vec3f_t * src, unsigned int count)*
- Matrix-Constant Arithmetic e.g. add, sub, mult, div, invert, transform, identity, Matrix-Vector, Vector-Vector and Matrix-Matrix Algebra up to 4 dimensions provided

$$\mathbf{A}^{-1} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & k \end{bmatrix}^{-1} = \frac{1}{\det(\mathbf{A})} \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & K \end{bmatrix}^{T} = \frac{1}{\det(\mathbf{A})} \begin{bmatrix} A & D & G \\ B & E & H \\ C & F & K \end{bmatrix}$$

*(*invert_mat3x3f)(arm_mat3x3f_t * dst, arm_mat3x3f_t * src, unsigned int count)*

What is NEON?

- NEON is a wide SIMD data processing architecture
  - Extension of the ARM® instruction set
  - 32 registers, 64-bits wide (dual view as 16 registers, 128-bits wide)
- NEON Instructions perform "Packed SIMD" processing
  - Registers are considered as vectors of elements of the same data type
  - Data types can be: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single prec. float
  - Instructions perform the same operation in all lanes



Using NDK with a Web UI

- Prefer Web-style UI to the normal Android Widgets?
- Or just familiar with HTML and don't want to change?
  - Good news, you can still take advantage of the NDK
- Put a little extra zip in your Javascript
  - The pages don't have to be part of the assets
- Embed your web pages into an Android App Shell
  - You can extend the Javascript libraries with calls to Java
  - And those calls to Java can call NDK JNI Functions

```
WebView web = (WebView)findViewById(R.id.webview);
ProxyBridge jscriptBridge = new ProxyBridge();
web.addJavascriptInterface(jscriptBridge, "pBridge");

WebSettings settings = web.getSettings();
settings.setJavaScriptEnabled(true);

web.loadUrl("file:///android_asset/index.html");
```

Runtime Choice of Binaries

- Ideally want to ship one binary for all Android Devices
  - This can be done with a little thought
- Build a shared library for each HW variant you want
  - Remember to keep a Java implementation as fallback
- Use a Interface or Abstract Class pattern select the right code at runtime
- Use either Board (which gives device information) or a manual scan of '/proc/cpuinfo' to select which code to use
- Will bloat your App
- Use Thumb or Thumb2 instead ARM where you can
  - Native Thumb is still much faster than Dalvik
  - Thumb-2 is similar to ARM, supports NEON and V6 SIMD
- Weigh each choice of Native method carefully

Example Code

```
// Use Build, if you know exactly what you're looking for
if (Build.BOARD.compareTo("DualCoreA9DevBoard") {
   System.loadLibrary("SMPOptimisedA9Lib");
}
```

or

```
// Create a class to parse the OS's '/proc/cpuinfo' data
CPUInfo cpuinfo = CPUInfo.getCPUInfo();
if (cpuinfo != null) && (cpuinfo.isARM()) { //
   System.loadLibrary("medianFilterC");
}
if ((cpuinfo != null) &&
    (cpuinfo.getCPUArchitecture().isCompatible(CPUArch.ARMv5TEJ))) {
    System.loadLibrary("medianFilterv5");
}
if ((cpuinfo != null) &&
    (cpuinfo.getCPUArchitecture().isCompatible(CPUArch.ARMv6TEJ))) {
    System.loadLibrary("medianFilterv6");
}
if ((cpuinfo != null) && (cpuinfo.hasFeature(CPUFeature.ARM_Neon))) {
    System.loadLibrary("medianFilterNeon");
}
```

Use a Factory Pattern to hand your main code an appropriate implementation

```
MedianFilter filter = MedianFilterFactory.getMedianFilter(cpuinfo);

filter.doFilter(...parameters...);
```

NEON Visualizer

http://szeged.github.com/nevada/