Improve Android System Component Performance

Jim Huang (黃敬群) <**jserv**@0xlab.org>

Developer & Co-Founder, 0xlab

http://0xlab.org/

Feb 14, 2012 / Android Builders Summit

Rights to copy

contact@0xlab.org

Corrections, suggestions, and contributions are welcome!

Latest update: Feb 14, 2012

$0\text{x}1\text{ab} = 16^2 + 16\text{x}10 + 11 = 427$

(founded on April 27, 2009)

0xlab is another Hexspeak.
(pronounce: *zero-aks-lab*)

About Me

(1) Come from Taiwan

(2) Contributor of Android Open Source Project (AOSP)

(3) Developer, Linaro

(4) Focus: system performance and virtualization at 0xlab

connect your device to application

0xlab

Mission of 0xlab development:

Improve UX in SoC

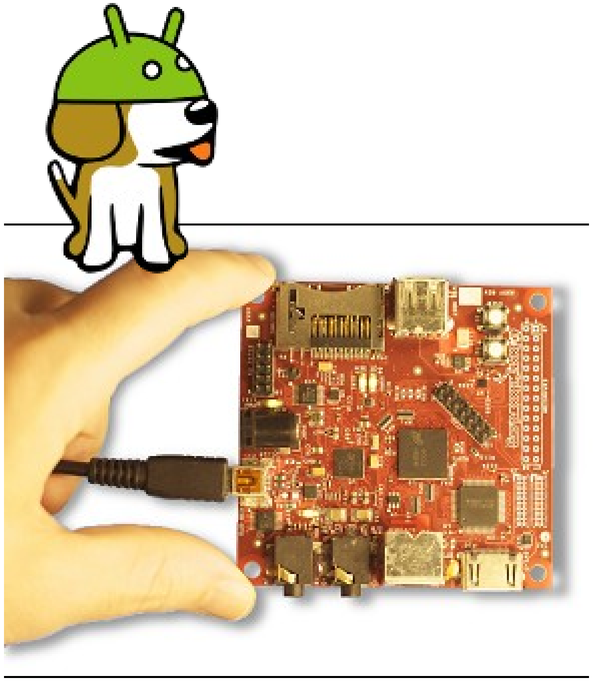
UX = User Experience

SoC = Integrated Computing Anywhere

Strategy and Policy

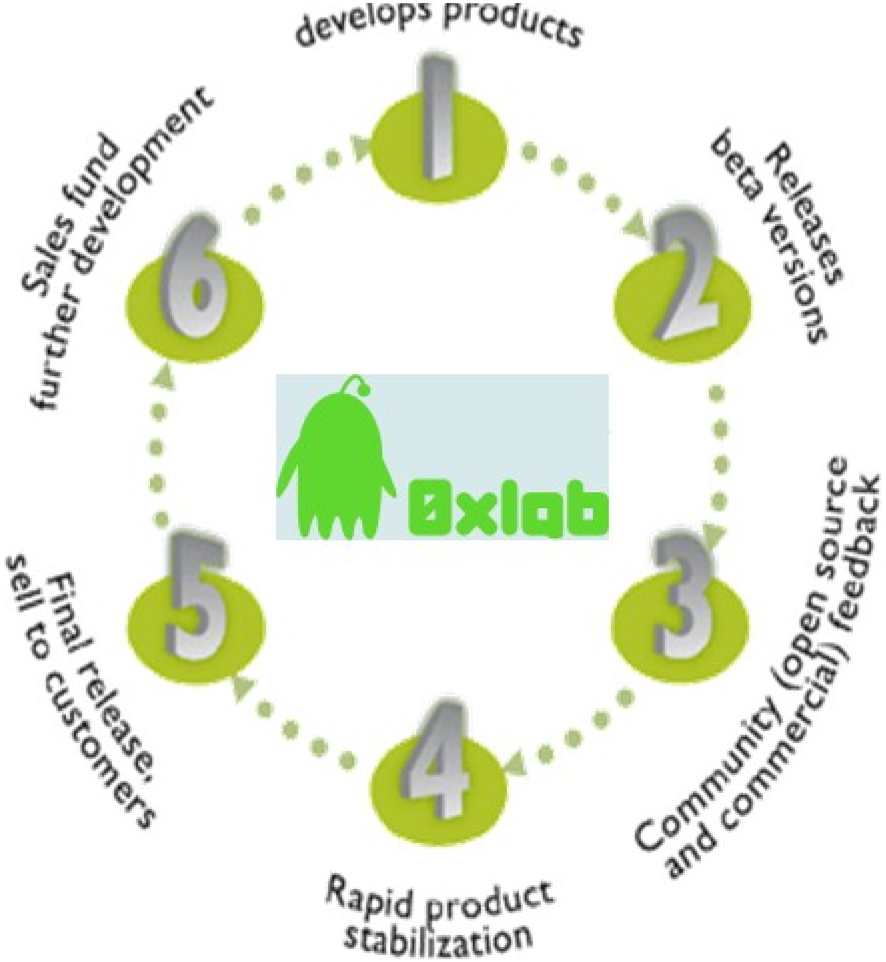
- open source efforts to improve AOSP
- We focus on small-but-important area of Android.
  - toolchain, libc, dynamic linker, skia, GLES, system libraries, HAL
- Develop system utilities for Android
  - benchmark, black-box testing tool, validation infrastructure
- Value-added features
  - Faster boot/startup time, Bluetooth profile, visual enhancements
- Submit and share changes to community
  - AOSP, CyanogenMod, Android-x86
  - Linaro

Working Model



Linaro

Rowboat **CyanogenMod** **Android-x86**



develops products

1

2

Releases beta versions

3

Community (open source and commercial) feedback

4

Rapid product stabilization

5

Final release, sell to customers

6

Sales fund further development

0xlab

Hidden Bugs in AOSP

- AOSP is dedicated to mobile product devices shipped by OHA members
  - Fixed hardware and specifications
  - Not well verified for other configurations
- Performance is important, but we frequently hit the hidden bugs when apply aggressive optimizations.
  - Quality is the first priority!

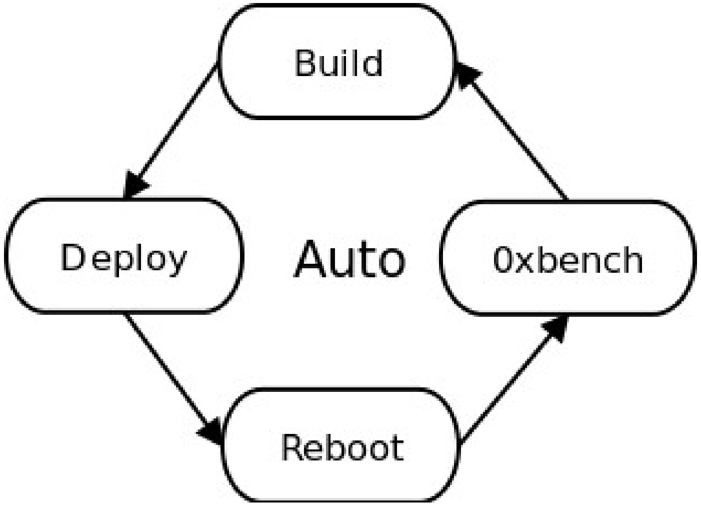
Quality in custom Android Distribution

- 0xlab delivers the advantages of open source software and development.
  - Quality relies on two factors: continuous development + strong user feedback
- Several utilities are developed to ensure the quality and released as open source software.
  - 0xbench (Android benchmarking tool)
  - ASTER (Android System Testing Environment and Runtime)
  - LAVA (Linaro Automated Validation Architecture)
- In the meanwhile, performance is improved by several patches against essential components.

Tip: Automate system before optimizing

LAVA: Automated Validation Infrastructure for Android

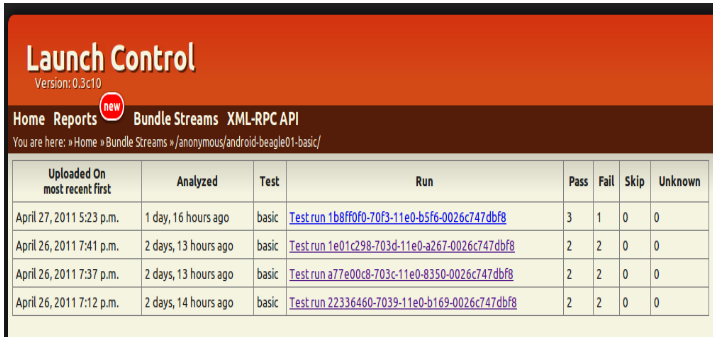
Android benchmark running on **LAVA**. Automated Validation flow includes from deploy, then reboot, testing, benchmark running, and result submit.



Check "LAVA Project Update" by Paul Larson, 2012 Embedded Linux Conference

Android support on LAVA
https://wiki.linaro.org/Platform/Validation/LAVA

```
Android related commands in LAVA:
 * deploy_linaro_android_image
 * boot_linaro_android_image
 * test_android_basic
 * test_android_monkey
 * test_android_0xbench
 * submit_results_on_host
```

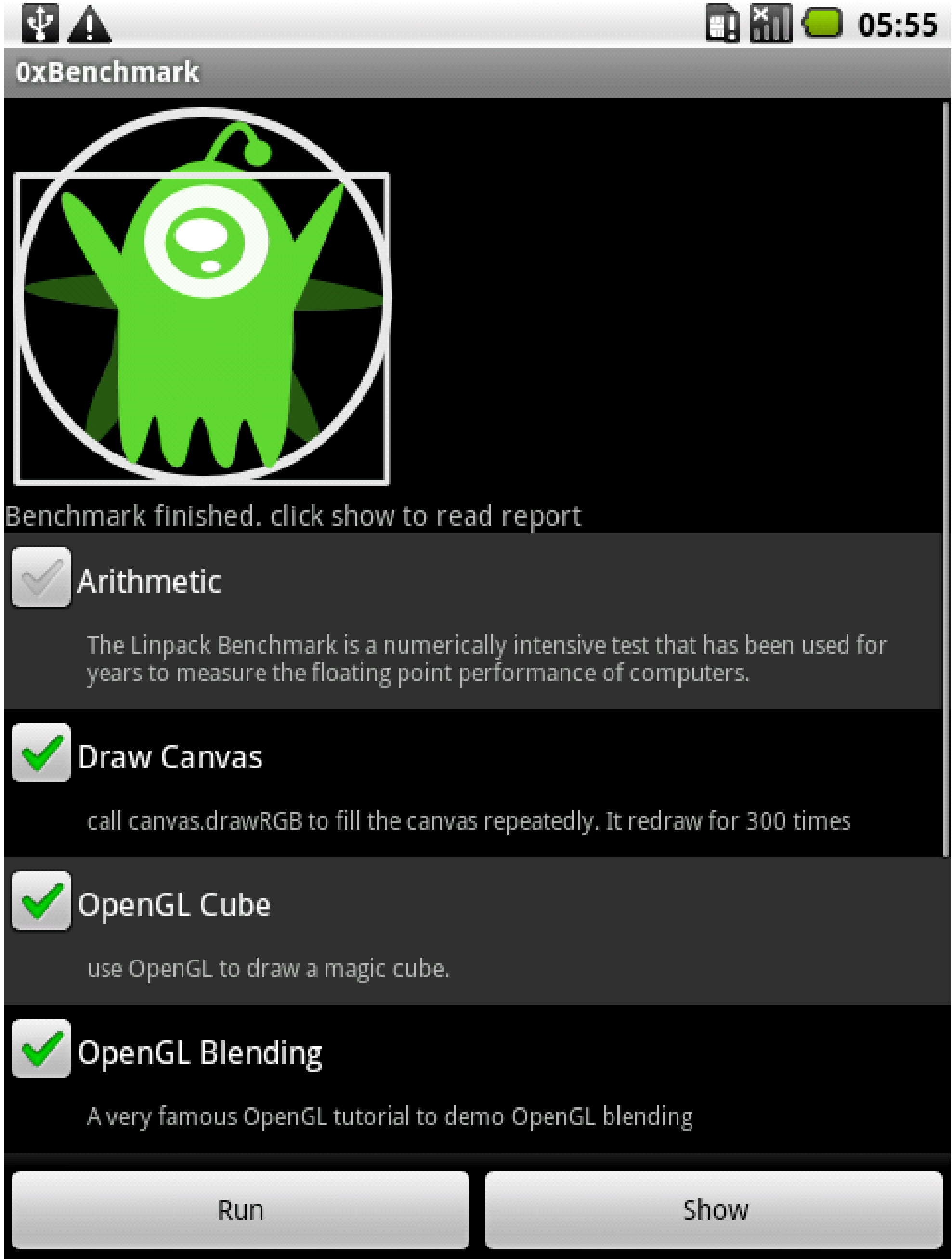


You are here: » Home » Bundle Streams » /anonymous/android-beagle01-basic/

Uploaded On most recent first	Analyzed	Test	Run	Pass	Fail	Skip	Unknown
April 27, 2011 5:23 p.m.	1 day, 16 hours ago	basic	Test run 1b8ff0f0-70f3-11e0-b5f6-0026c747dbf8	3	1	0	0
April 26, 2011 7:41 p.m.	2 days, 13 hours ago	basic	Test run 1e01c298-703d-11e0-a267-0026c747dbf8	2	2	0	0
April 26, 2011 7:37 p.m.	2 days, 13 hours ago	basic	Test run a77e00c8-703c-11e0-8350-0026c747dbf8	2	2	0	0
April 26, 2011 7:12 p.m.	2 days, 14 hours ago	basic	Test run 22336460-7039-11e0-b169-0026c747dbf8	2	2	0	0

0xbench: comprehensive open source benchmark suite for Android

- A set of system utilities for Android to perform comprehensive system benchmarking
  - Dalvik VM performance
  - OpenGL|ES performance
  - Android Graphics framework performance
  - I/O performance
  - JavaScript engine performance
  - Connectivity performance
  - Micro-benchmark: stanard C library, system call, latency, Java invocation, ...



Benchmark finished. click show to read report

□ Arithmetic

The Linpack Benchmark is a numerically intensive test that has been used for years to measure the floating point performance of computers.

☑ Draw Canvas

call canvas.drawRGB to fill the canvas repeatedly. It redraw for 300 times

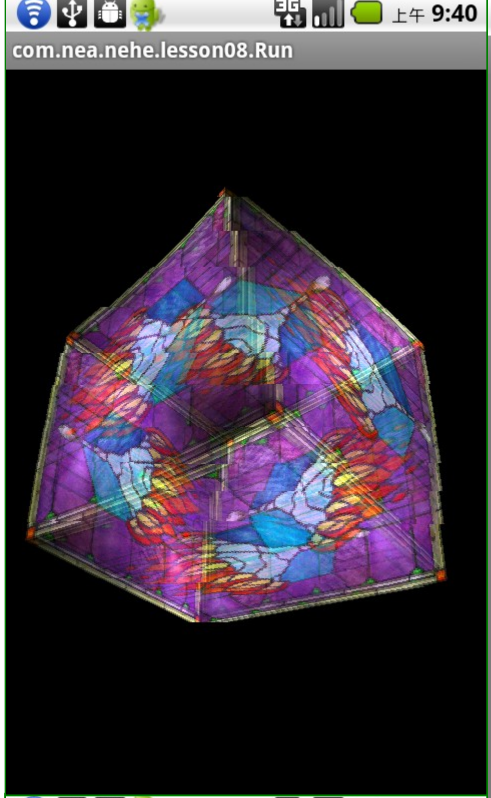
☑ OpenGL Cube

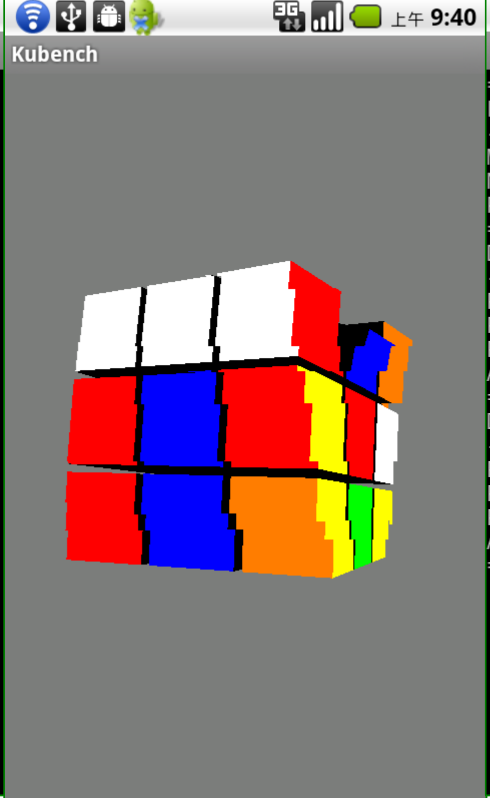
use OpenGL to draw a magic cube.

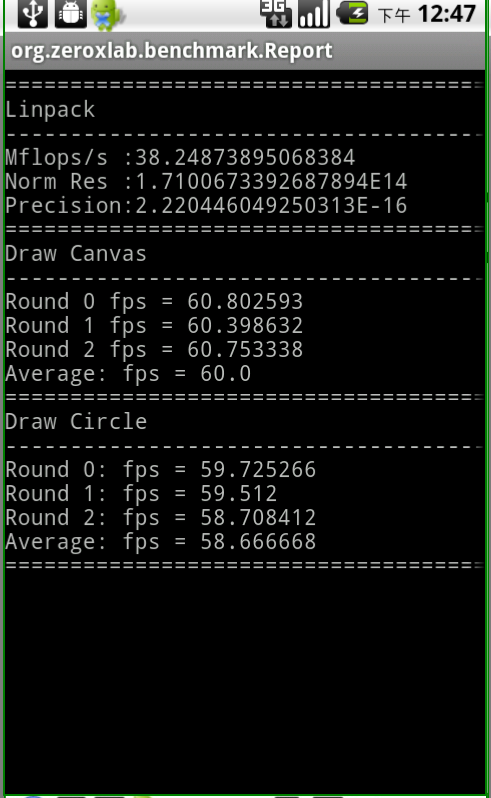
☑ OpenGL Blending

A very famous OpenGL tutorial to demo OpenGL blending

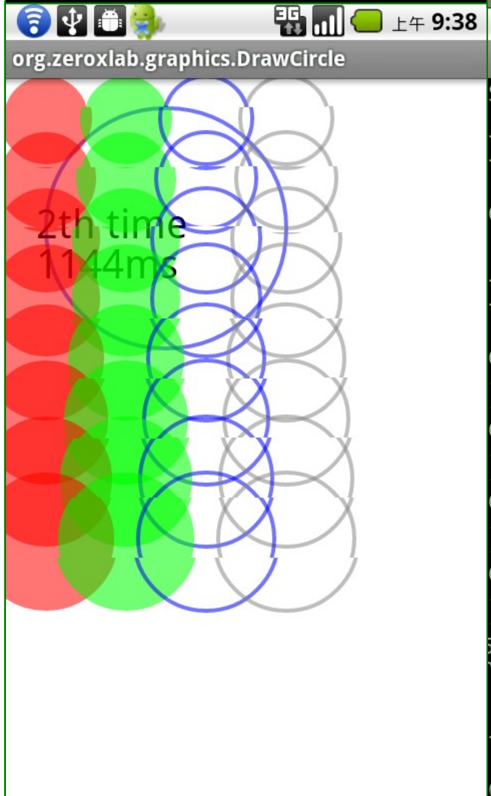
Project page: http://code.google.com/p/0xbench/

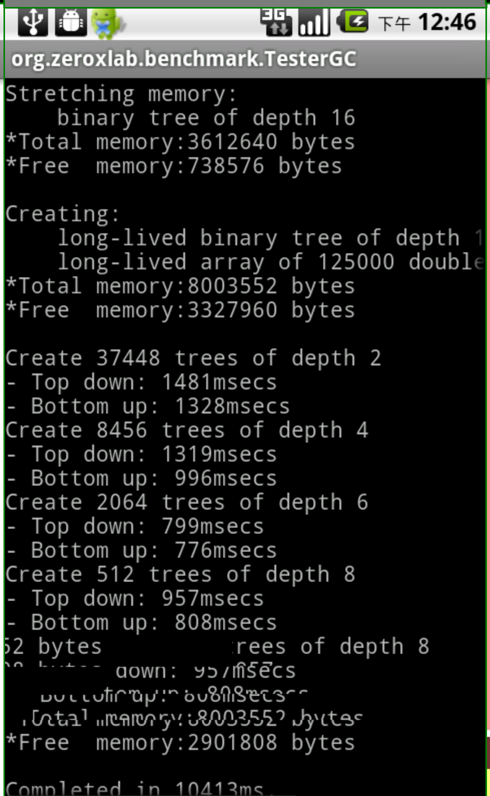




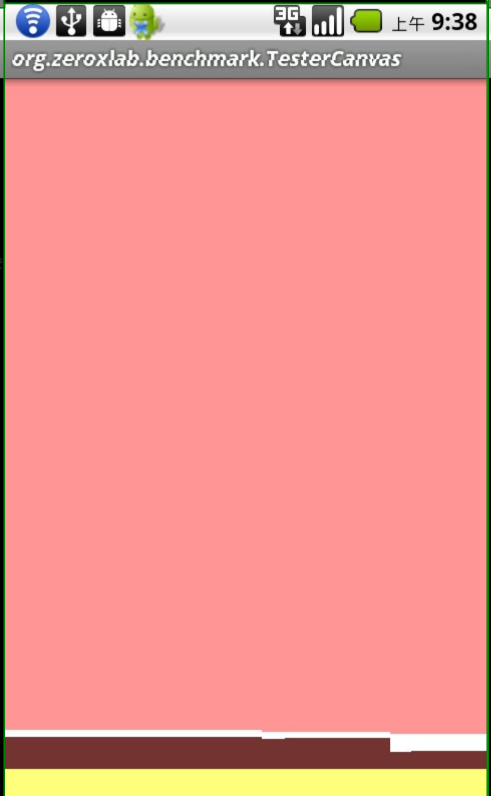


```
=====================================
Linpack
-------------------------------------
Mflops/s :38.24873895068384
Norm Res :1.7100673392687894E14
Precision:2.220446049250313E-16
=====================================
Draw Canvas
-------------------------------------
Round 0 fps = 60.802593
Round 1 fps = 60.398632
Round 2 fps = 60.753338
Average: fps = 60.0
=====================================
Draw Circle
-------------------------------------
Round 0: fps = 59.725266
Round 1: fps = 59.512
Round 2: fps = 58.708412
Average: fps = 58.666668
=====================================
```

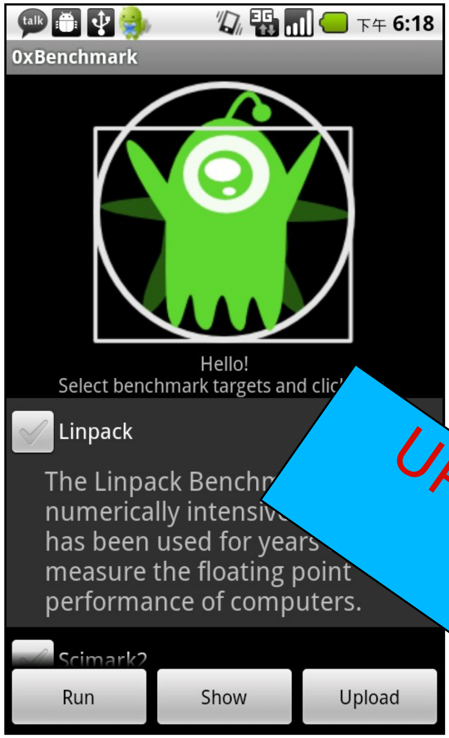




```
Stretching memory:
    binary tree of depth 16
*Total memory:3612640 bytes
*Free  memory:738576 bytes

Creating:
    long-lived binary tree of depth 1
    long-lived array of 125000 double
*Total memory:8003552 bytes
*Free  memory:3327960 bytes

Create 37448 trees of depth 2
- Top down: 1481msecs
- Bottom up: 1328msecs
Create 8456 trees of depth 4
- Top down: 1319msecs
- Bottom up: 996msecs
Create 2064 trees of depth 6
- Top down: 799msecs
- Bottom up: 776msecs
Create 512 trees of depth 8
- Top down: 957msecs
- Bottom up: 808msecs
52 bytes          rees of depth 8
        down: 957msecs
  Bottom up: 808msecs
 Total memory:8003552 bytes
*Free  memory:2901808 bytes

Completed in 10413ms.
```



Collect and Analyze results on server-side

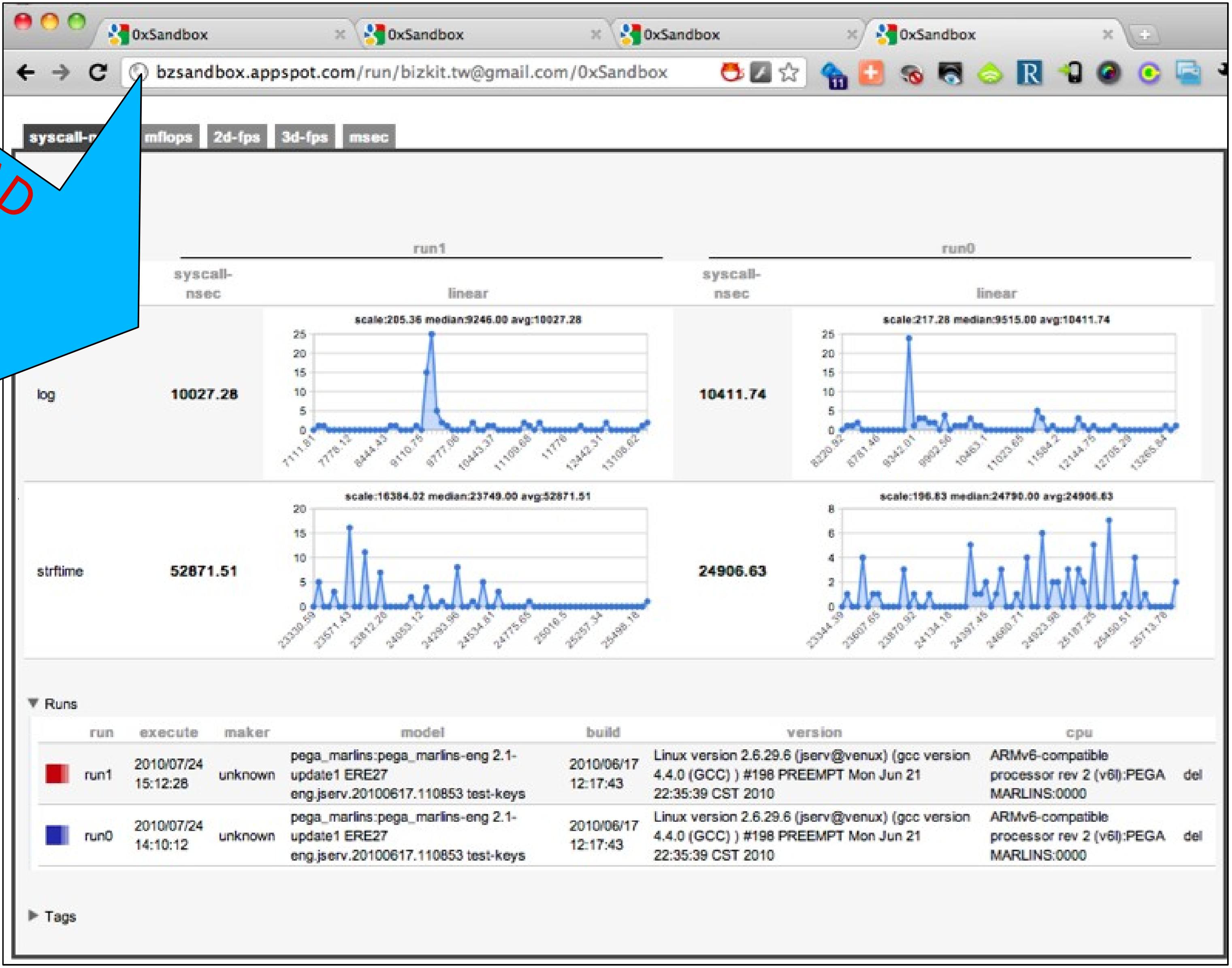


Hello!
Select benchmark targets and clic

☑ Linpack

The Linpack Benchm
numerically intensiv
has been used for years
measure the floating point
performance of computers.

☑ Scimark2



	run1		run0	
	syscall-nsec	linear	syscall-nsec	linear
log	10027.28	scale:205.36 median:9246.00 avg:10027.28	10411.74	scale:217.28 median:9515.00 avg:10411.74
strftime	52871.51	scale:16384.02 median:23749.00 avg:52871.51	24906.63	scale:196.63 median:24790.00 avg:24906.63

▼ Runs

	run	execute	maker	model	build	version	cpu	
🟥	run1	2010/07/24 15:12:28	unknown	pega_marlins:pega_marlins-eng 2.1-update1 ERE27 eng.jserv.20100617.110853 test-keys	2010/06/17 12:17:43	Linux version 2.6.29.6 (jserv@venux) (gcc version 4.4.0 (GCC)) #198 PREEMPT Mon Jun 21 22:35:39 CST 2010	ARMv6-compatible processor rev 2 (v6l):PEGA MARLINS:0000	del
🟦	run0	2010/07/24 14:10:12	unknown	pega_marlins:pega_marlins-eng 2.1-update1 ERE27 eng.jserv.20100617.110853 test-keys	2010/06/17 12:17:43	Linux version 2.6.29.6 (jserv@venux) (gcc version 4.4.0 (GCC)) #198 PREEMPT Mon Jun 21 22:35:39 CST 2010	ARMv6-compatible processor rev 2 (v6l):PEGA MARLINS:0000	del

▶ Tags

Android Functional Testing

- stress test
  - Utilizing 'monkey', which is part of framework
- Automated test
  - Both blackbox-test and whitebox-test are required

Stress Test

- According to CDD (Compatibility Definition Document), Device implementations MUST include the Monkey framework, and make it available for applications to use.
- `monkey` is a command that can directly talks to Android framework and emulate random user input.

```
adb shell monkey -p your.package.name -v 500
```

- Decide the percentage of touch events, keybord events, etc., then run automatically.

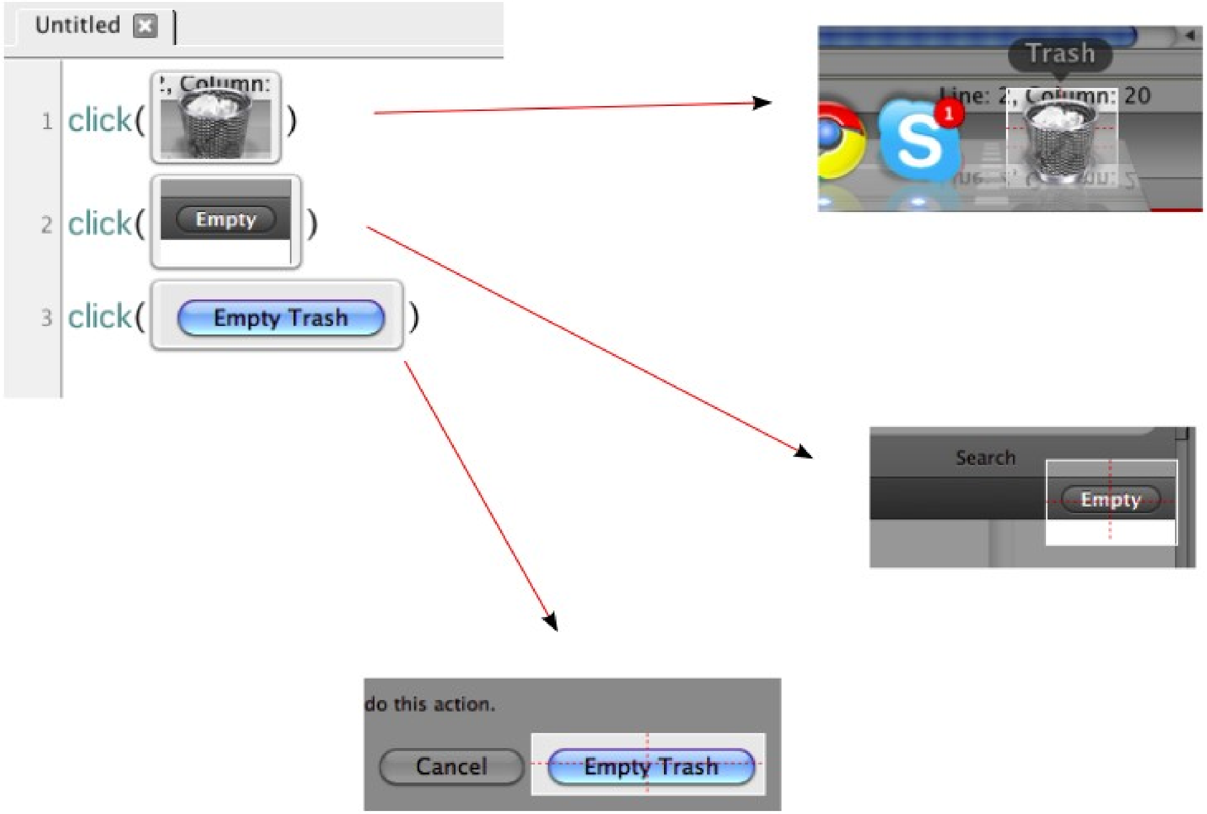
ASTER: Automated Test

- Blackbox-test vs. Whitebox-test
- An easy to use automated testing tool with IDE
  - Built upon MoneyRunner
- Batch execution of visual test scripts
- Multiple chains of recall commands
- Designed for non-programmer or Q&A engineers
- Use OpenCV to recognize icons or UI hints

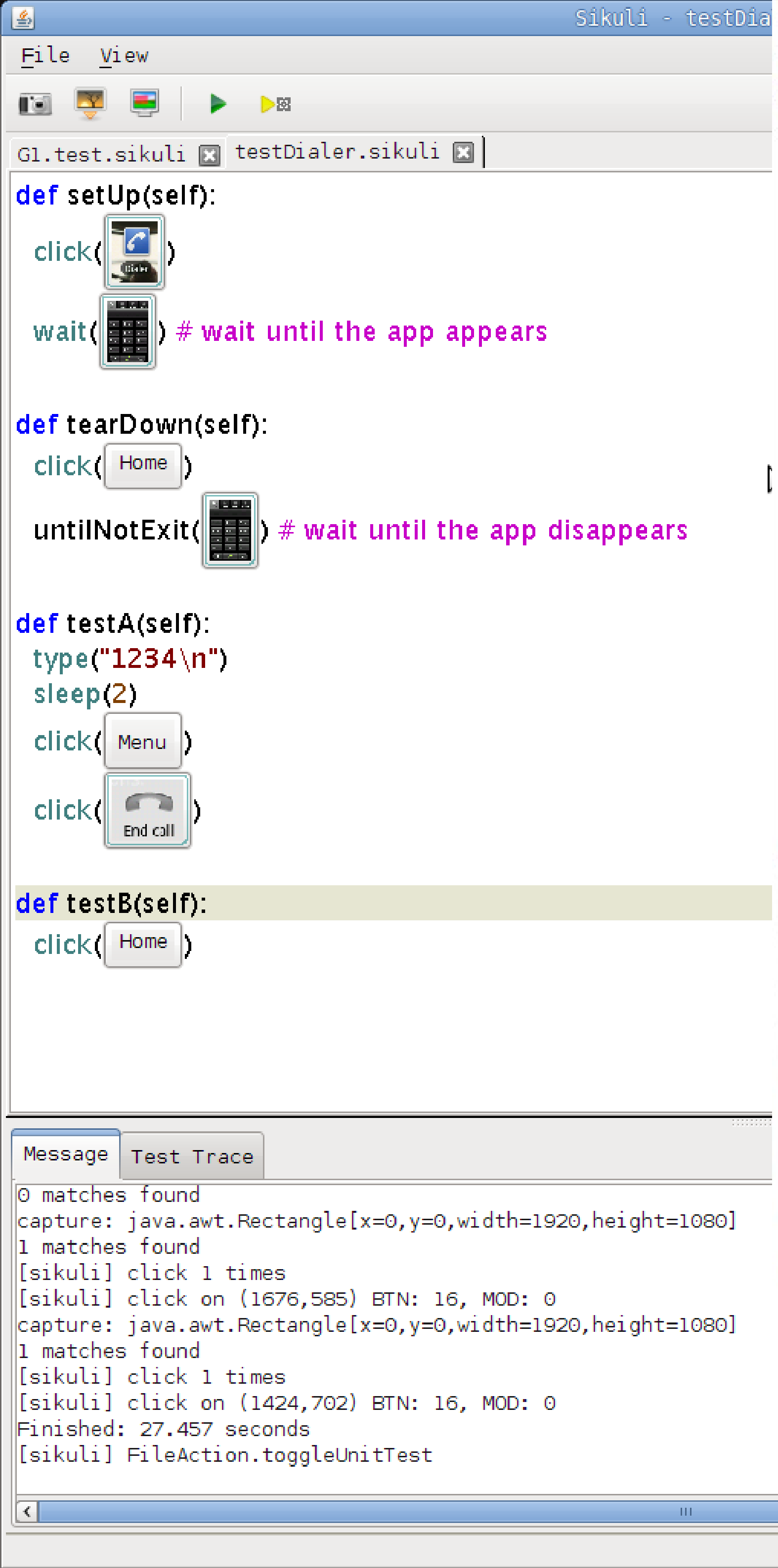
Project page: http://code.google.com/p/aster/

Functional Test

Desktop: Sikuli



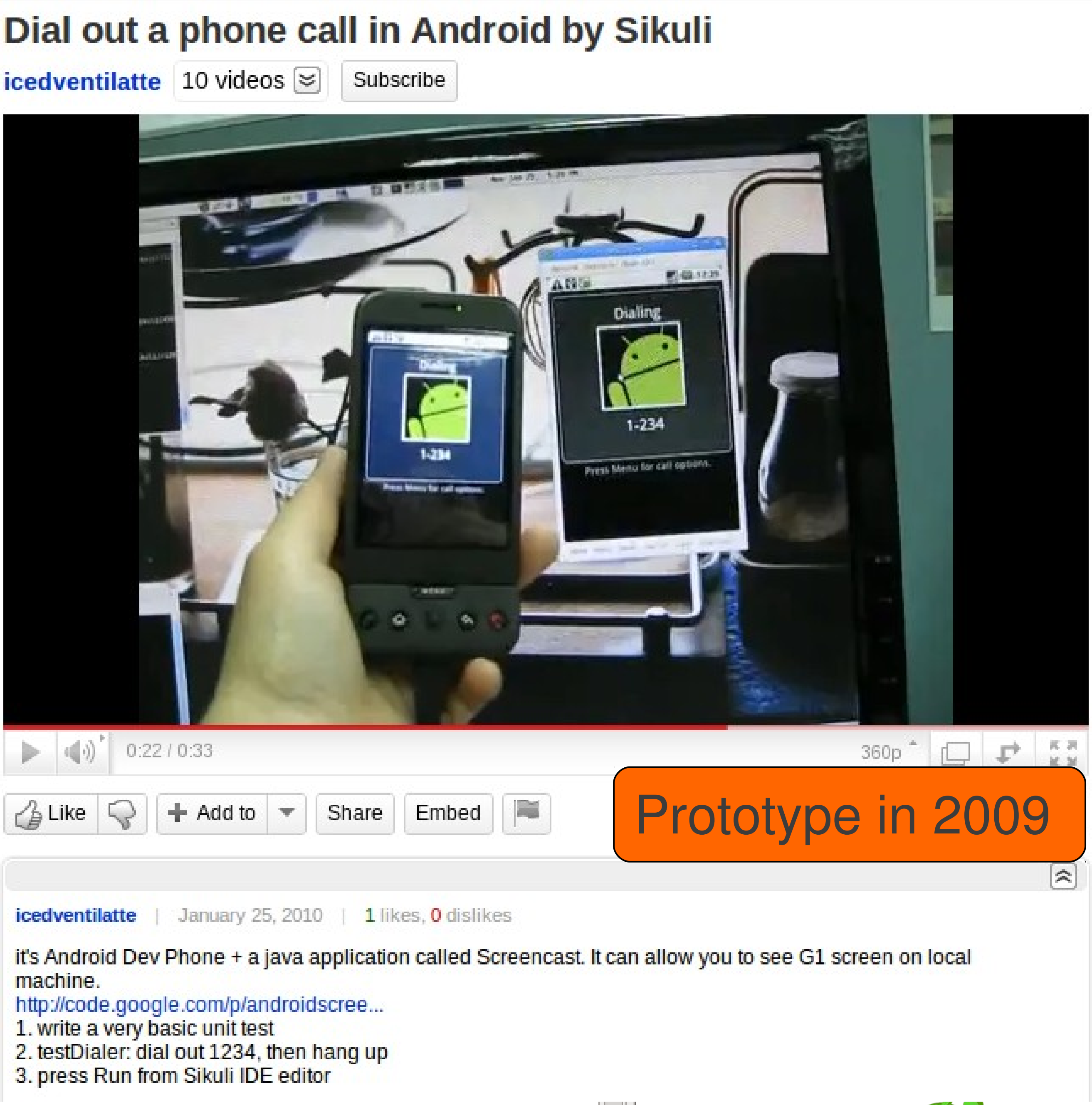
```
click( )
click( )
click( )
```



```
def setUp(self):
  click( )
  wait( ) # wait until the app appears

def tearDown(self):
  click(Home)
  untilNotExit( ) # wait until the app disappears

def testA(self):
  type("1234\n")
  sleep(2)
  click(Menu)
  click(End call)

def testB(self):
  click(Home)
```

```
0 matches found
capture: java.awt.Rectangle[x=0,y=0,width=1920,height=1080]
1 matches found
[sikuli] click 1 times
[sikuli] click on (1676,585) BTN: 16, MOD: 0
capture: java.awt.Rectangle[x=0,y=0,width=1920,height=1080]
1 matches found
[sikuli] click 1 times
[sikuli] click on (1424,702) BTN: 16, MOD: 0
Finished: 27.457 seconds
[sikuli] FileAction.toggleUnitTest
```

Line: 97, Column: 19



Dial out a phone call in Android by Sikuli

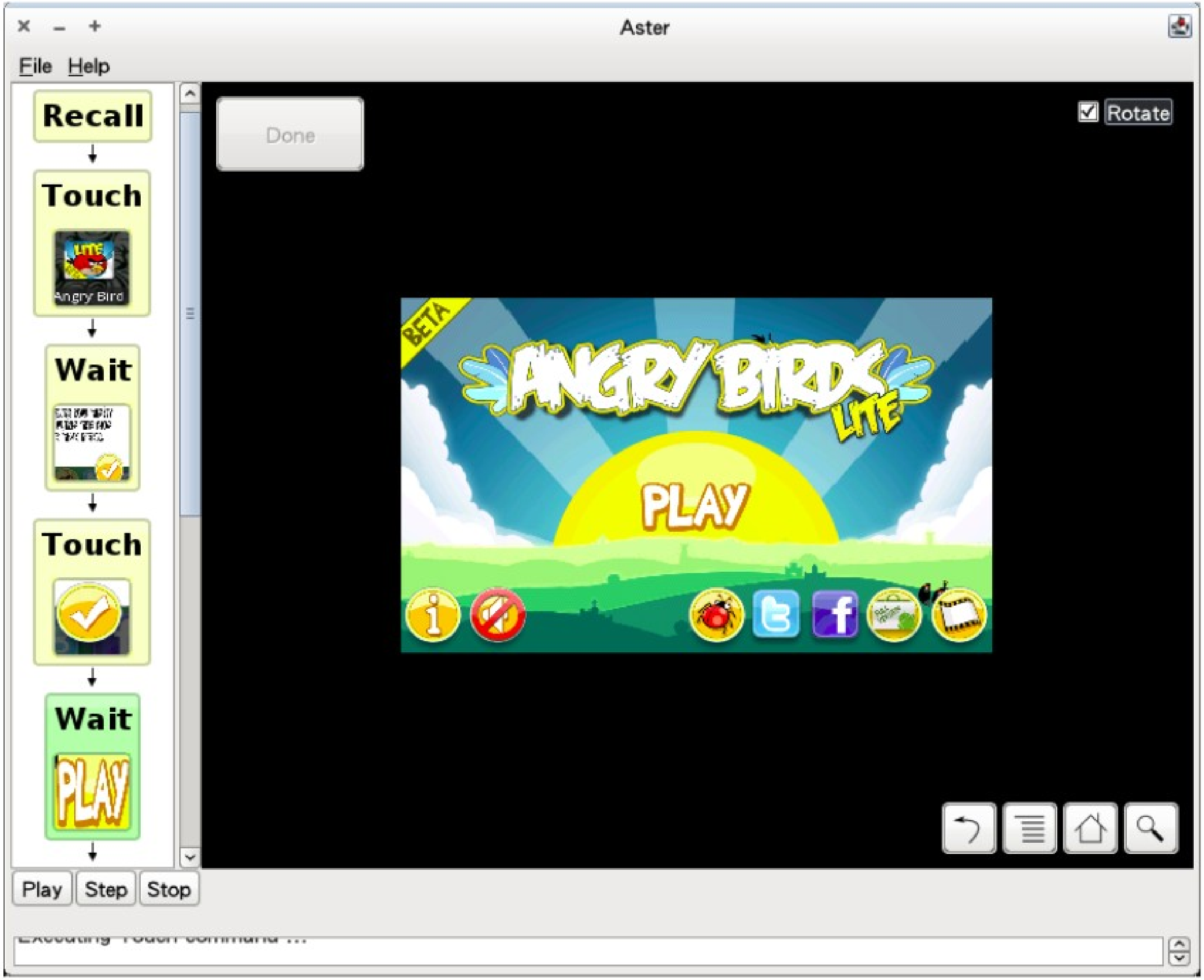
Prototype in 2009

icedventilatte | January 25, 2010 | 1 likes, 0 dislikes

it's Android Dev Phone + a java application called Screencast. It can allow you to see G1 screen on local machine.
http://code.google.com/p/androidscree...

1. write a very basic unit test
2. testDialer: dial out 1234, then hang up
3. press Run from Sikuli IDE editor

ASTER IDE in 2011



Executing Touch command ...

Aster

It is time to improve the performance of Android system components

No Silver Bullet

to Improve the whole

Possibly Premature optimizations in Android

- **“Premature optimization is the root of all evil”** – Donald Knuth
- bionic libc
  - glibc incompatibility, No SysV IPC, partial Pthread, incomplete prelink
  - inactive/incorrect kernel header inclusion
  - May not re-use existing system utilities
- Assumed UI behavior
  - Input event dispatching and handler
  - Strict / non-maintainable state machine (policy)
  - Depending on a certain set of peripherals
- Unclear HAL design and interface
  - Wifi, Bluetooth, GPS, ...

Think Difficult

- To make performance improvement visible
  - Modifications from Application level, Android framework, system libraries, and kernel
- Slowdown in newer Android version
  - Example: Graphics in Eclair (2.0/2.1) is much slower than 1.5 or 1.6
- To optimize or not to optimize, that is the question.
  - Merge Local optimizations != Optimized globally
  - Many Android applications don't take various devices into consideration. Thus, performance issues occur all the way.

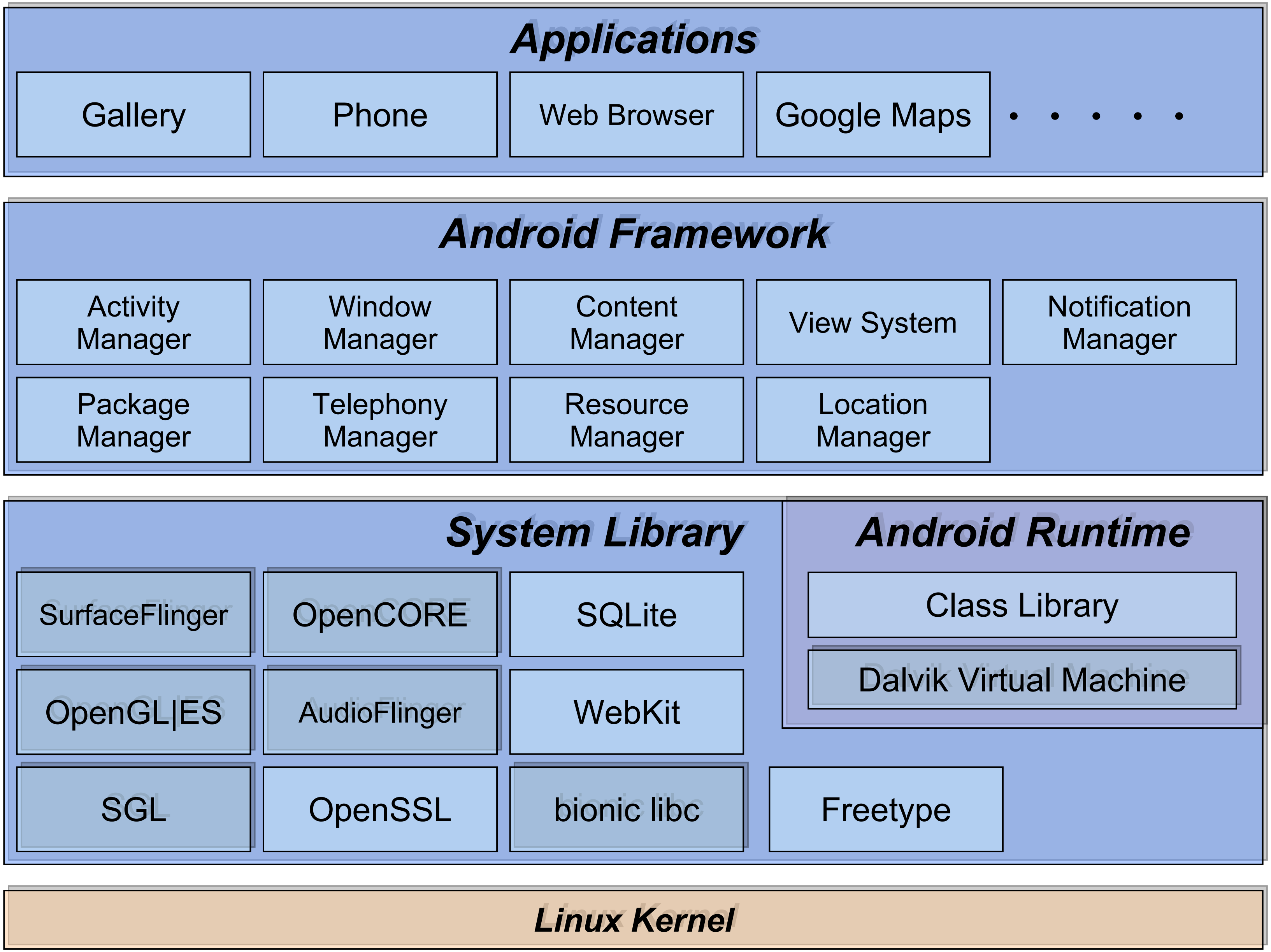
Which parts will be Improved?

- 2D/3D Graphics
- Android Runtime
- Boot time

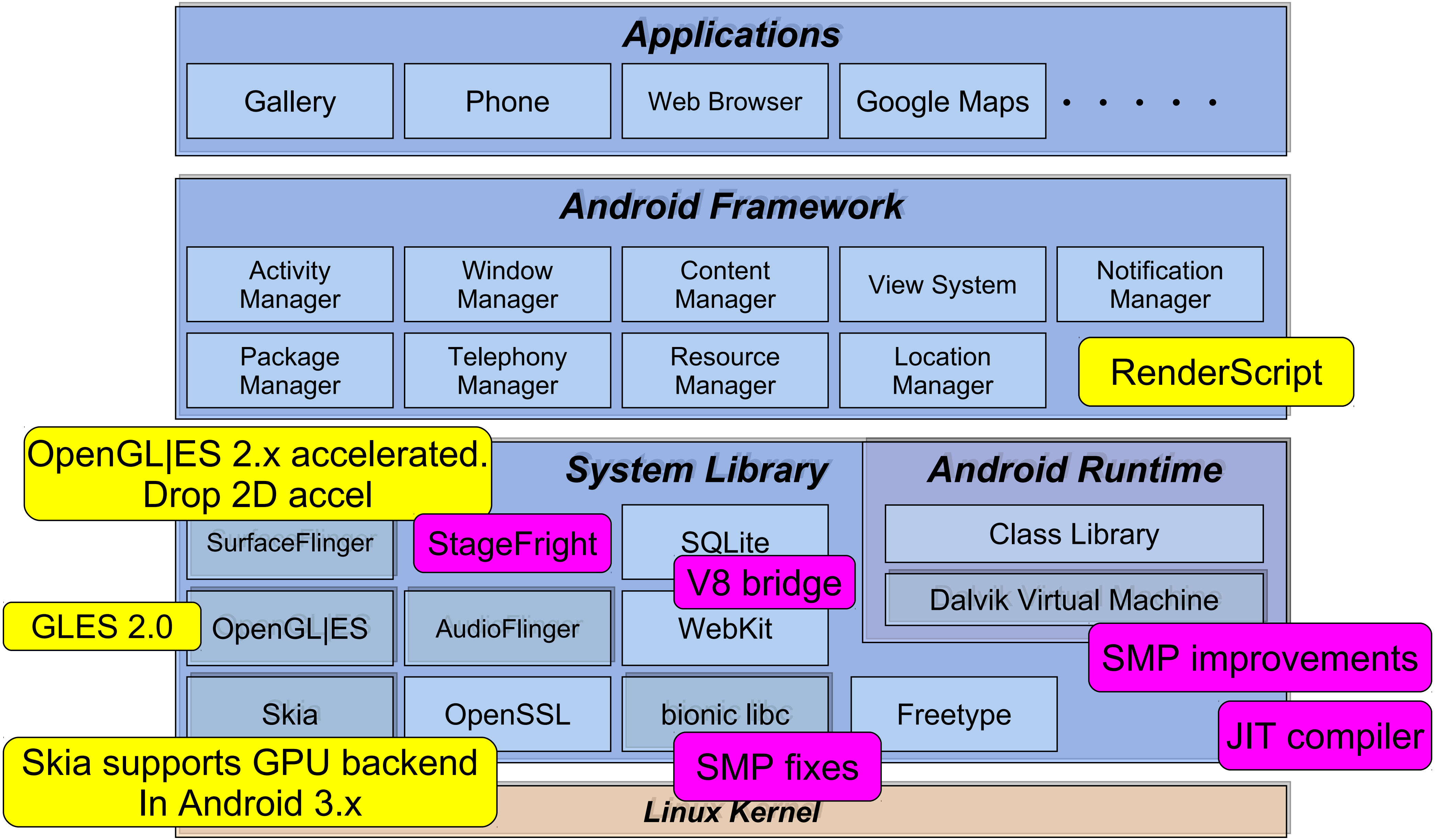
Three frequently mentioned items in Android engineering are selected as the entry points: 2D/3D graphics, runtime, and boot time.

Android Graphics

Functional View (1.5)

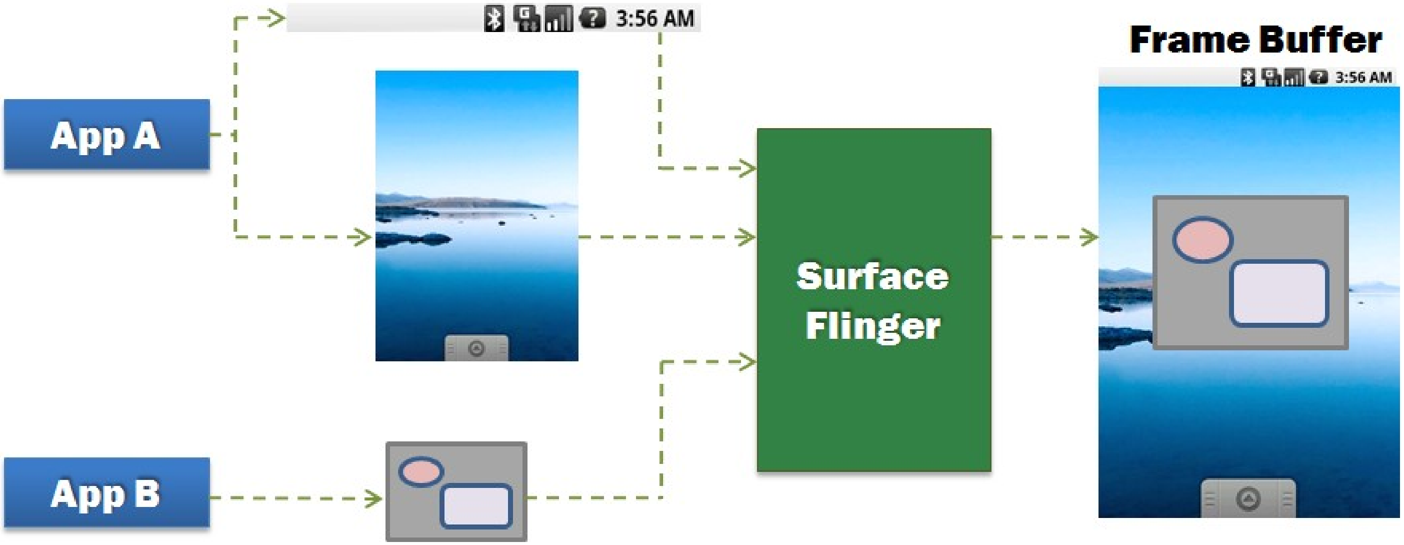


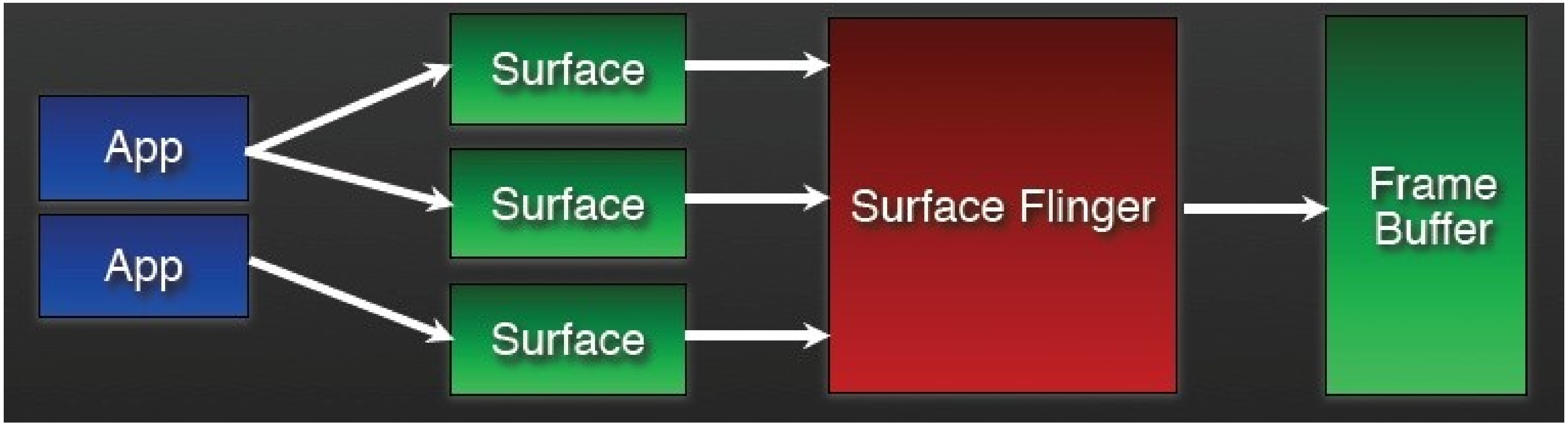
Functional View (2.3)



Android SurfaceFlinger

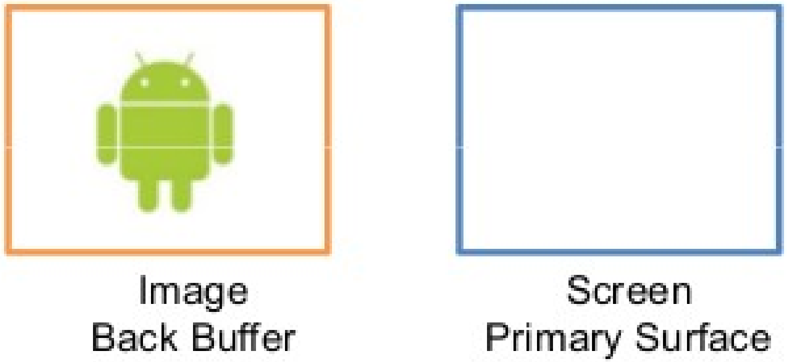
- Properties
  - Can combine 2D/3D surfaces and surfaces from multiple applications
  - Surfaces passed as buffers via Binder IPC calls
  - Can use OpenGL ES and 2D hardware accelerator for its compositions
    - Double-buffering using page-flip



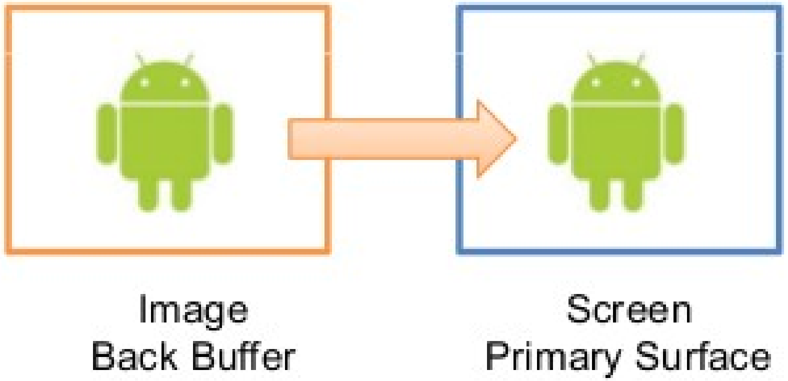


Double Buffering

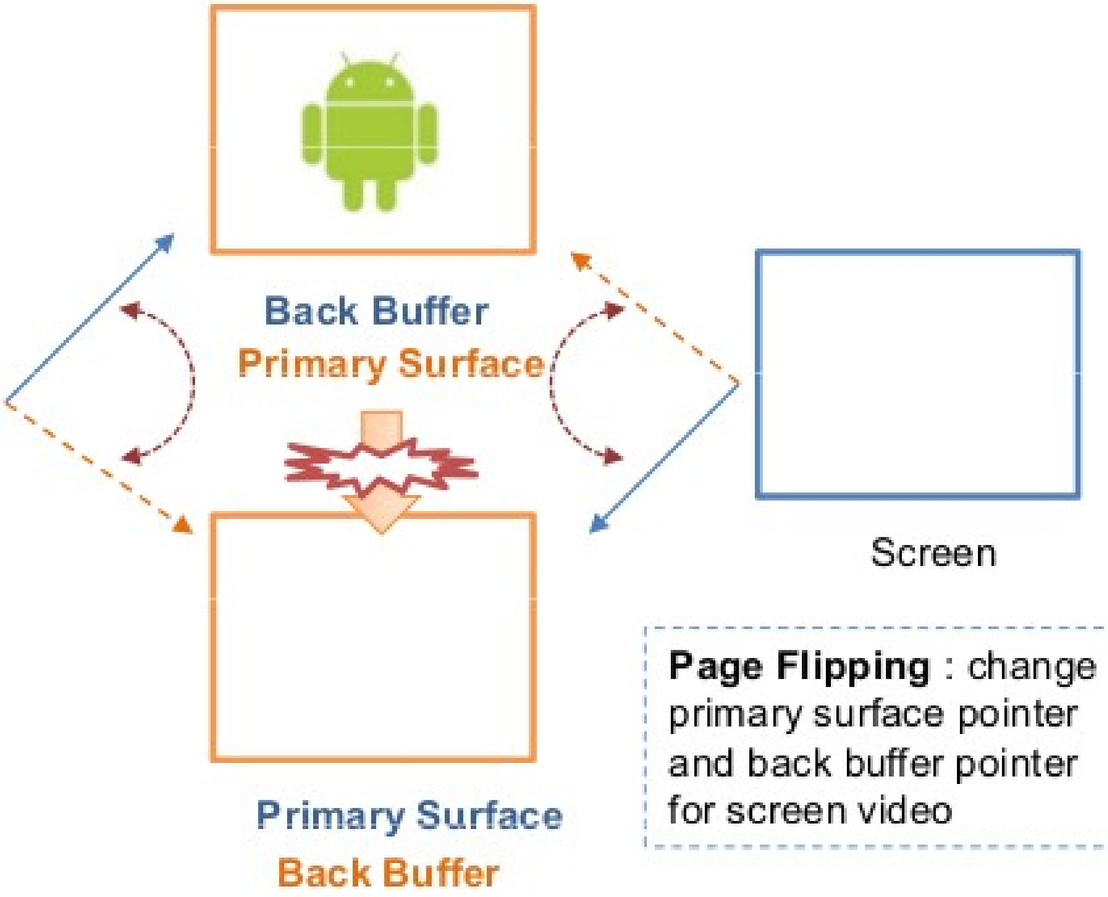
Draw



Copy (BLT : Block Line Transfer)

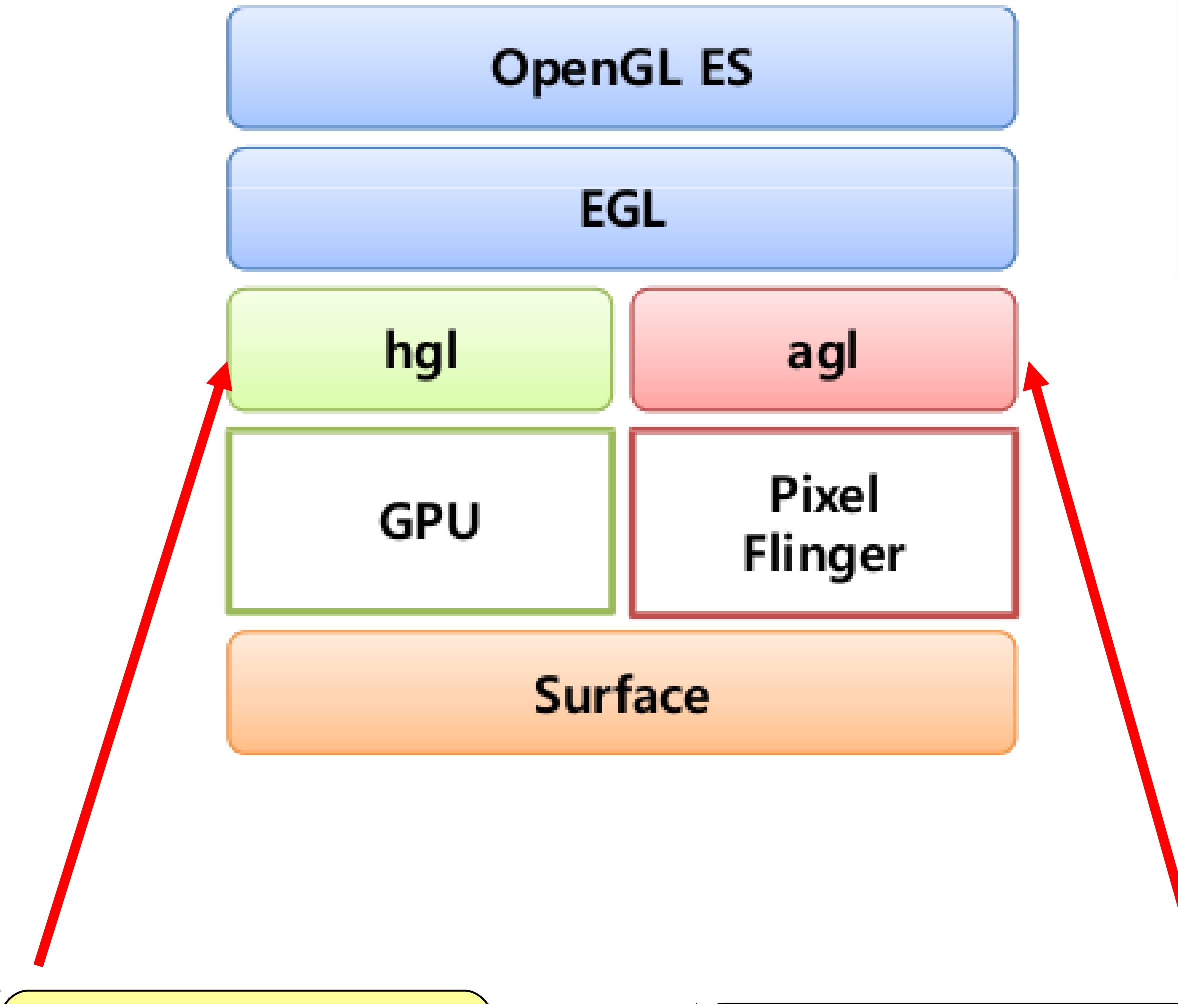


Page Flipping



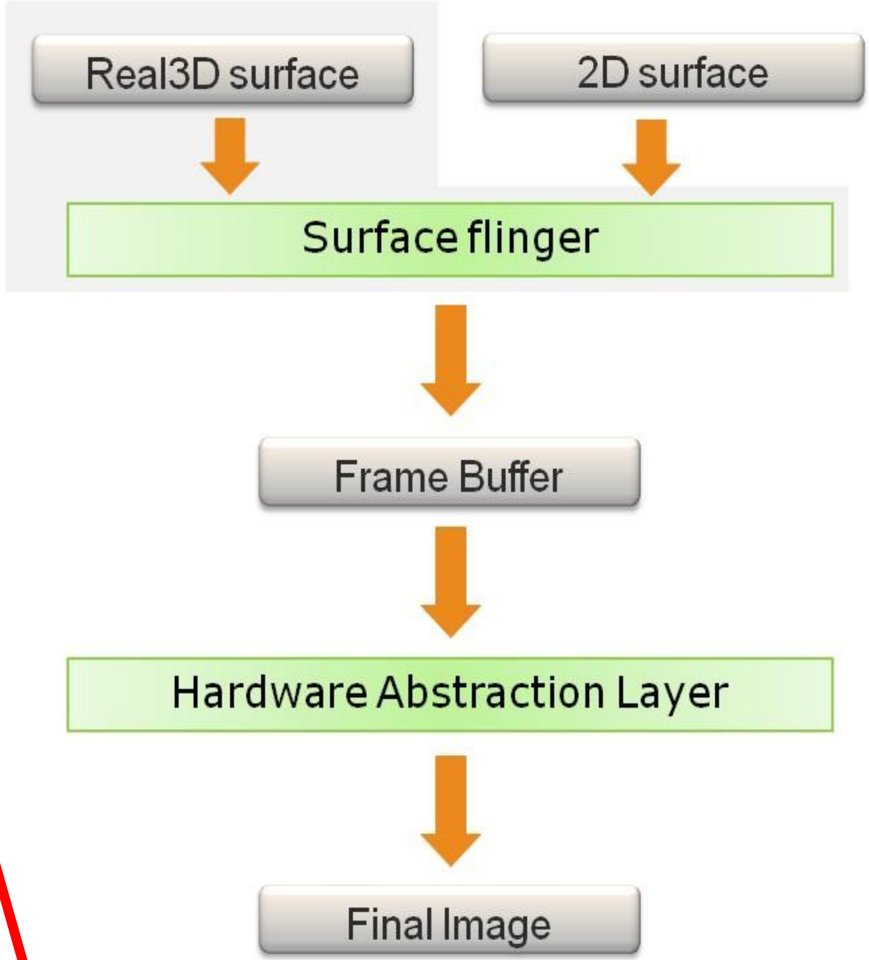
Page Flipping : change primary surface pointer and back buffer pointer for screen video

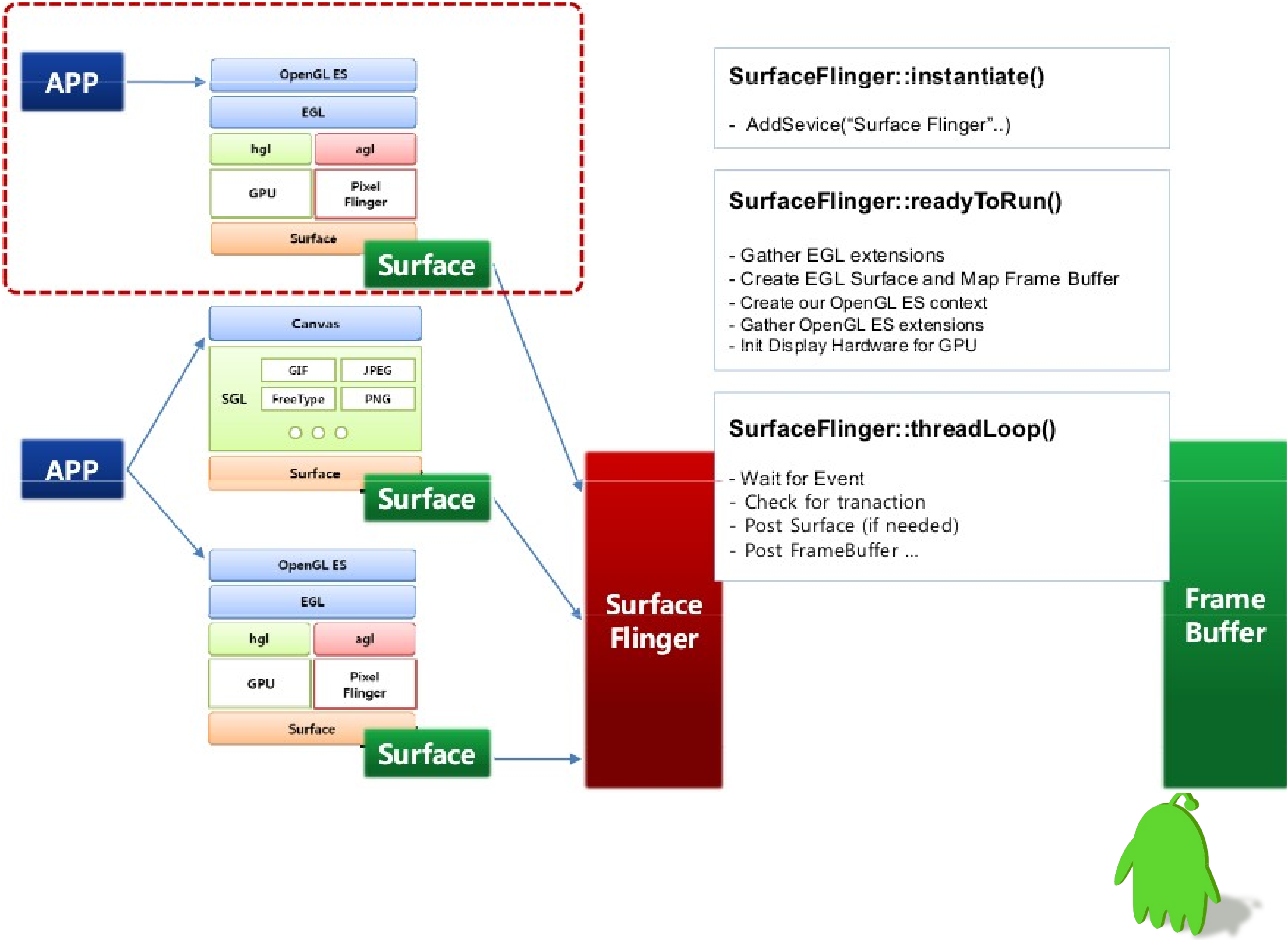
from EGL to SurfaceFlinger



hgl = hardware OpenGL|ES

agl = android software OpenGL|ES renderer





SurfaceFlinger::instantiate()

- AddSevice("Surface Flinger"..)

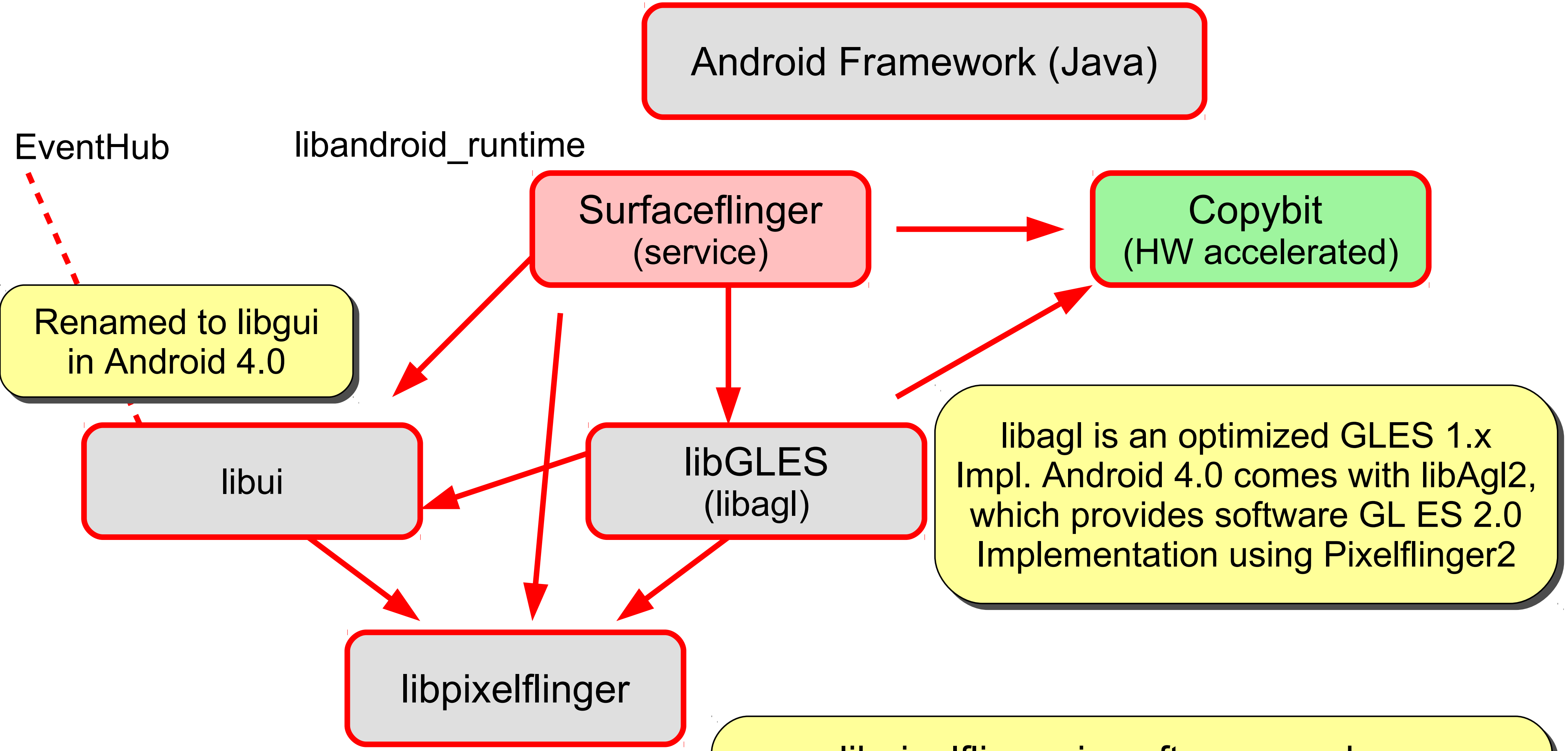
SurfaceFlinger::readyToRun()

- Gather EGL extensions
- Create EGL Surface and Map Frame Buffer
- Create our OpenGL ES context
- Gather OpenGL ES extensions
- Init Display Hardware for GPU

SurfaceFlinger::threadLoop()

- Wait for Event
- Check for tranaction
- Post Surface (if needed)
- Post FrameBuffer ...

Android Graphics without OpenGL|ES Hardware



Renamed to libgui in Android 4.0

libagl is an optimized GLES 1.x Impl. Android 4.0 comes with libAgl2, which provides software GL ES 2.0 Implementation using Pixelflinger2

libpixelflinger is software renderer Android 4.0 comes with a new implementation, PixelFlinger2, which Is based on LLVM and Mesa (glsl2-llvm): external/mesa3d

When GLES doesn't work, software is used

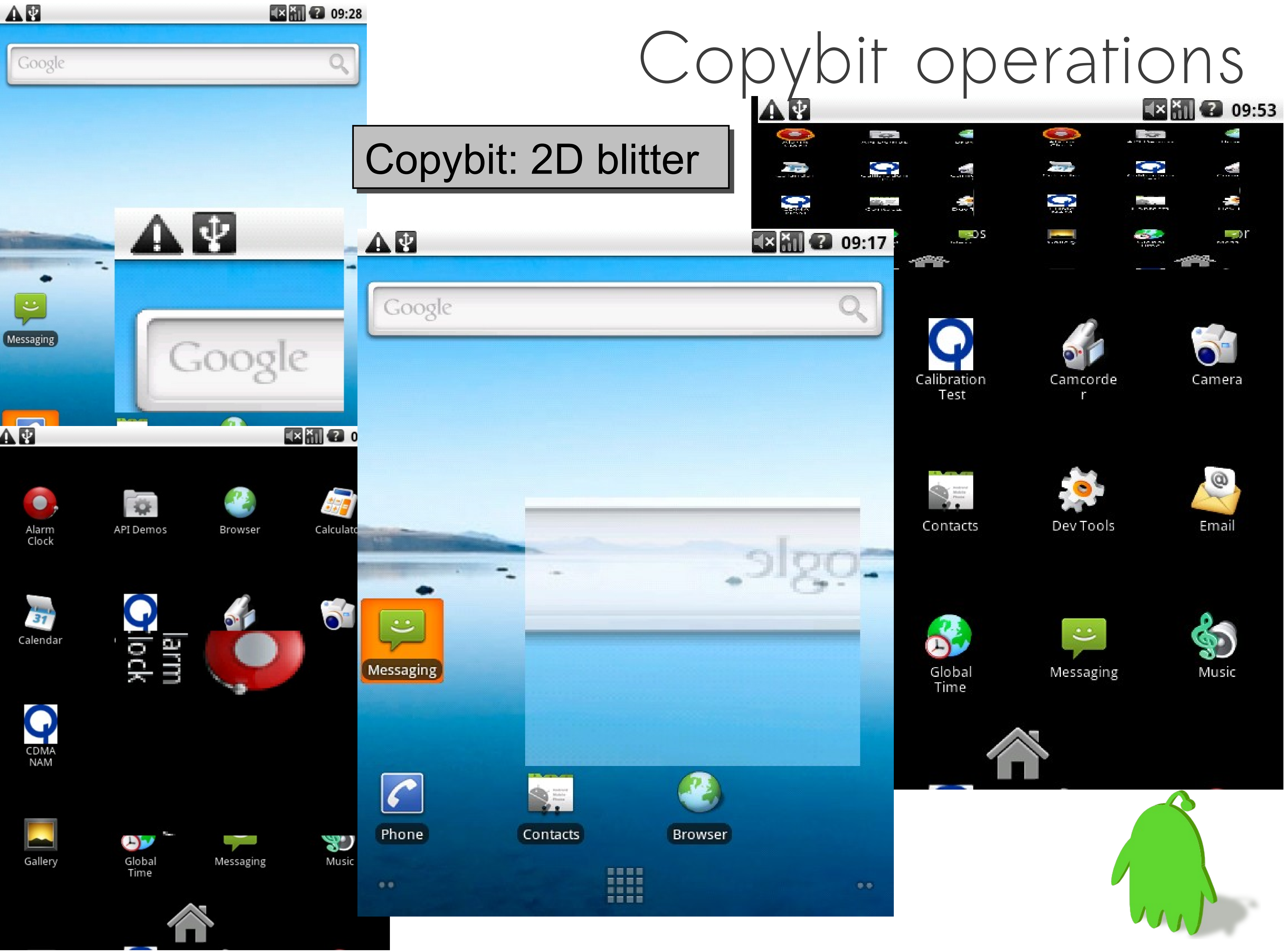
2D Accelerator for Android Graphics

- lib**copybit** provides hareware bitblit operations which includes moving, scaling, rotation, mirroring, and more effects, like blending, dithering, bluring, etc.
- Removed since Android 2.3
  - But adding it back might improve UX in large screen.
- Android has two copybit interfaces:
  - Blit: moving / blending
  - Stretch: scaling besides moving
- libcopybit is called by libagl which can do swapBuffers to do the framebuffer page flipping that can also be accelerated by libcopybit.

Copybit could improve the performance of page flipping

Copybit operations

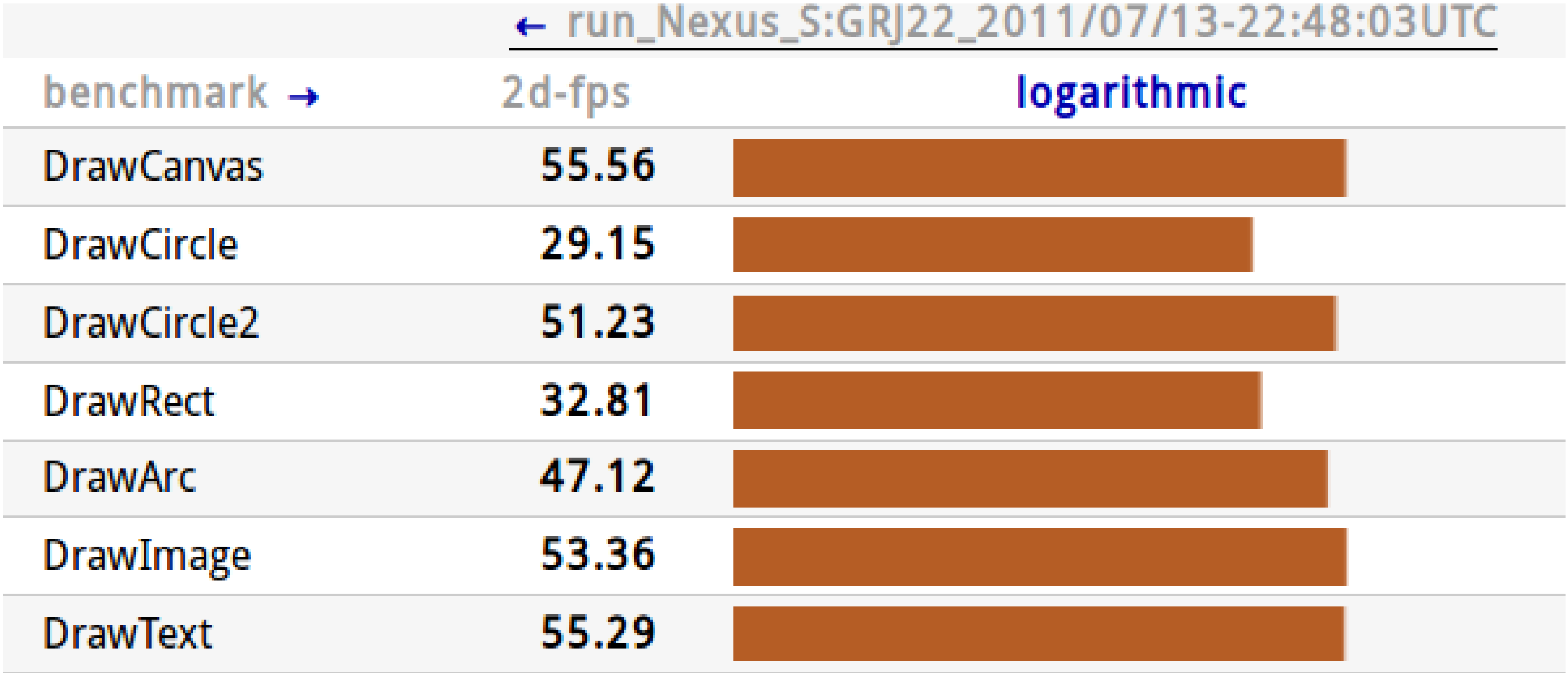
Copybit: 2D blitter



Optimizing Graphics without 3D/HW

- Implement copybit HAL carefully
  - Minimize clip region
  - Eliminate data copy
- Check ioctl for page flipping in framebuffer driver
  - Efficiency and consistency
- Without 3D/HW, Android Graphics is CPU bound
  - Reduce the amount of surfaces to manipulate
  - Optimizing skia (2D vector library) is important
  - Optimize color space conversion
  - Optimize blitter and primitive operations like matrix using ARM VFP and NEON

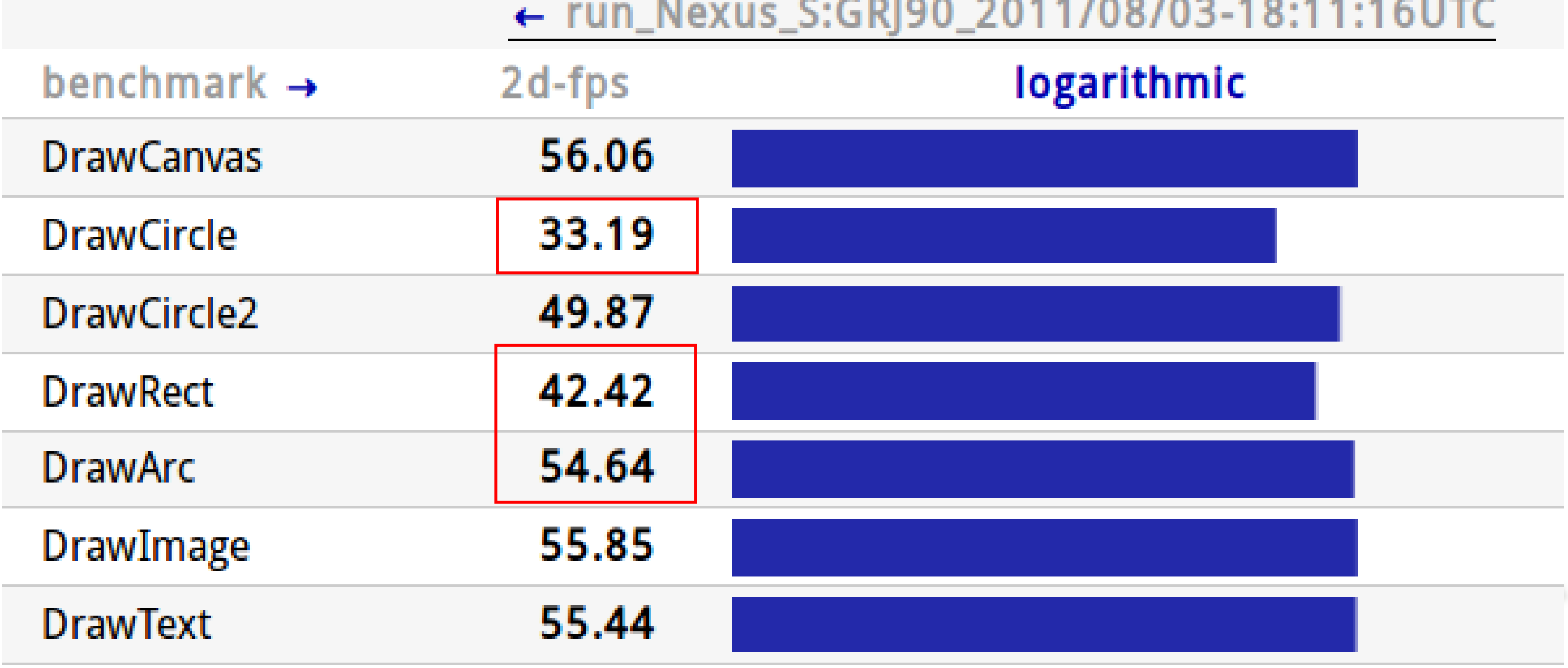
2D on Nexus S



← run_Nexus_S:GRJ22_2011/07/13-22:48:03UTC

benchmark →	2d-fps	logarithmic
DrawCanvas	55.56	
DrawCircle	29.15	
DrawCircle2	51.23	
DrawRect	32.81	
DrawArc	47.12	
DrawImage	53.36	
DrawText	55.29	

Apply extra performance tweaks against optimized build (NEON)



← run_Nexus_S:GRJ90_2011/08/03-18:11:16UTC

benchmark →	2d-fps	logarithmic
DrawCanvas	56.06	
DrawCircle	33.19	
DrawCircle2	49.87	
DrawRect	42.42	
DrawArc	54.64	
DrawImage	55.85	
DrawText	55.44	

2D Improvement (1)

external/skia/

```
ccommit ae265ac7f132f5d475040edf134e312b3987eade
    Add NEON optimized blitter: RGB565 to ABGR8888 without filter
and blending


commit 4b9b68bb9b8f82d6f70d98449851bc4bb19958bd
    optimize blend32_16_row and unroll SkRGB16_Blitter::blitRect

    Reference benchmark using 0xbench 2D on Nexus S (1 GHz)
    [before]
    Draw Rect:       28.52 fps

    [after]
    Draw Rect:       37.89 fps
```

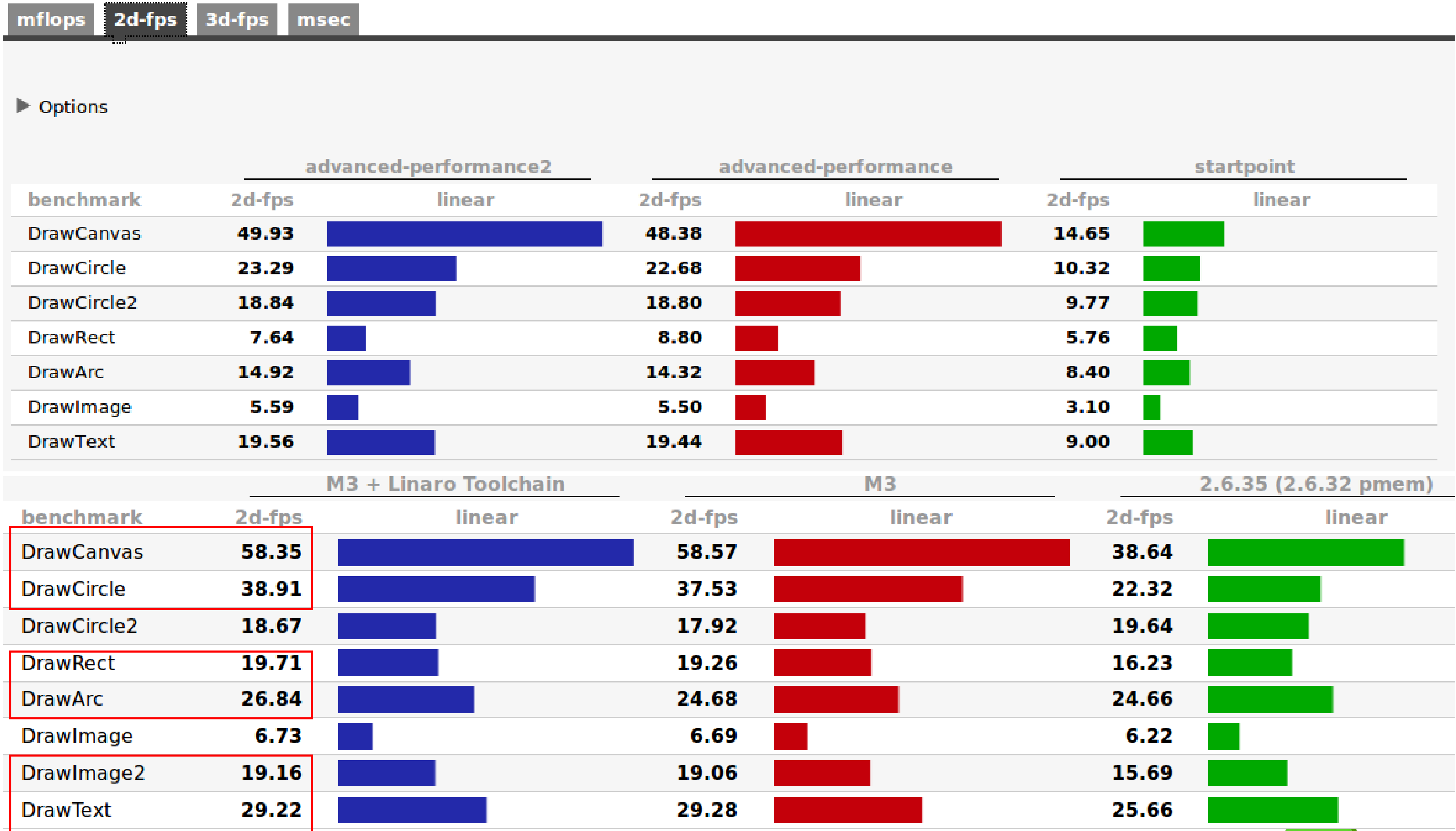
This presentation takes the contributions in CyanogenMod as example including SHA-1 hash

2D Improvement (2)

external/skia/

```
commit cb837750a37d59c979768320a7cf5ced96c7231c
    Add NEON optimized SkARGB32_Black_Blitter::blitMask

    Reference benchmark results on Nexus S (ARM Cortex-A8; 1 GHz) using
    skia_bench: (time in ms, smaller is better)
    [before]
    running bench [640 480]          text_48_linear_pos
      8888: cmsecs =  88.18
       565: cmsecs =  61.51
    running bench [640 480]              text_48_linear
      8888: cmsecs =  85.85
       565: cmsecs =  60.18

    [after]
    running bench [640 480]          text_48_linear_pos
      8888: cmsecs =  38.52
       565: cmsecs =  59.11
    running bench [640 480]              text_48_linear
      8888: cmsecs =  36.24
       565: cmsecs =  57.37
```

Benchmark: 2D (arm11-custom)

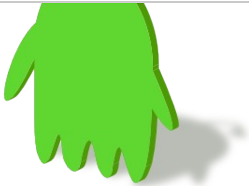


mflops | 2d-fps | 3d-fps | msec

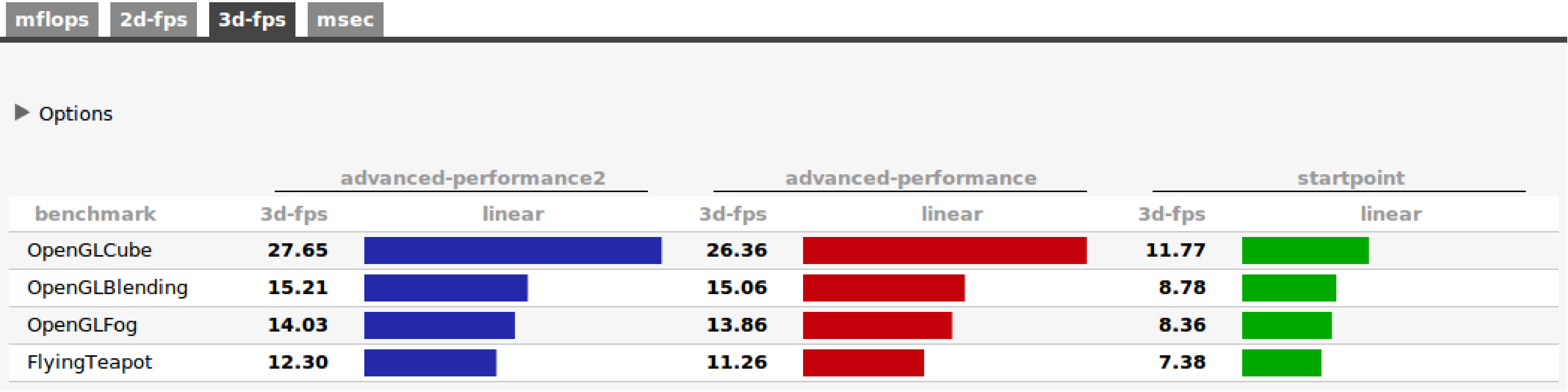
▶ Options

	advanced-performance2		advanced-performance		startpoint	
benchmark	2d-fps	linear	2d-fps	linear	2d-fps	linear
DrawCanvas	49.93		48.38		14.65	
DrawCircle	23.29		22.68		10.32	
DrawCircle2	18.84		18.80		9.77	
DrawRect	7.64		8.80		5.76	
DrawArc	14.92		14.32		8.40	
DrawImage	5.59		5.50		3.10	
DrawText	19.56		19.44		9.00	

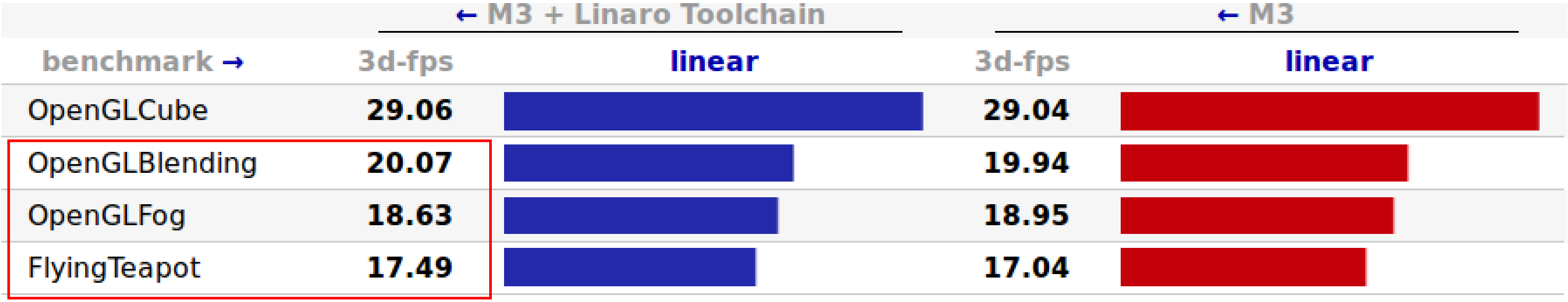
	M3 + Linaro Toolchain		M3		2.6.35 (2.6.32 pmem)	
benchmark	2d-fps	linear	2d-fps	linear	2d-fps	linear
DrawCanvas	58.35		58.57		38.64	
DrawCircle	38.91		37.53		22.32	
DrawCircle2	18.67		17.92		19.64	
DrawRect	19.71		19.26		16.23	
DrawArc	26.84		24.68		24.66	
DrawImage	6.73		6.69		6.22	
DrawImage2	19.16		19.06		15.69	
DrawText	29.22		29.28		25.66	



Benchmark: 3D (arm11-custom; no GPU)



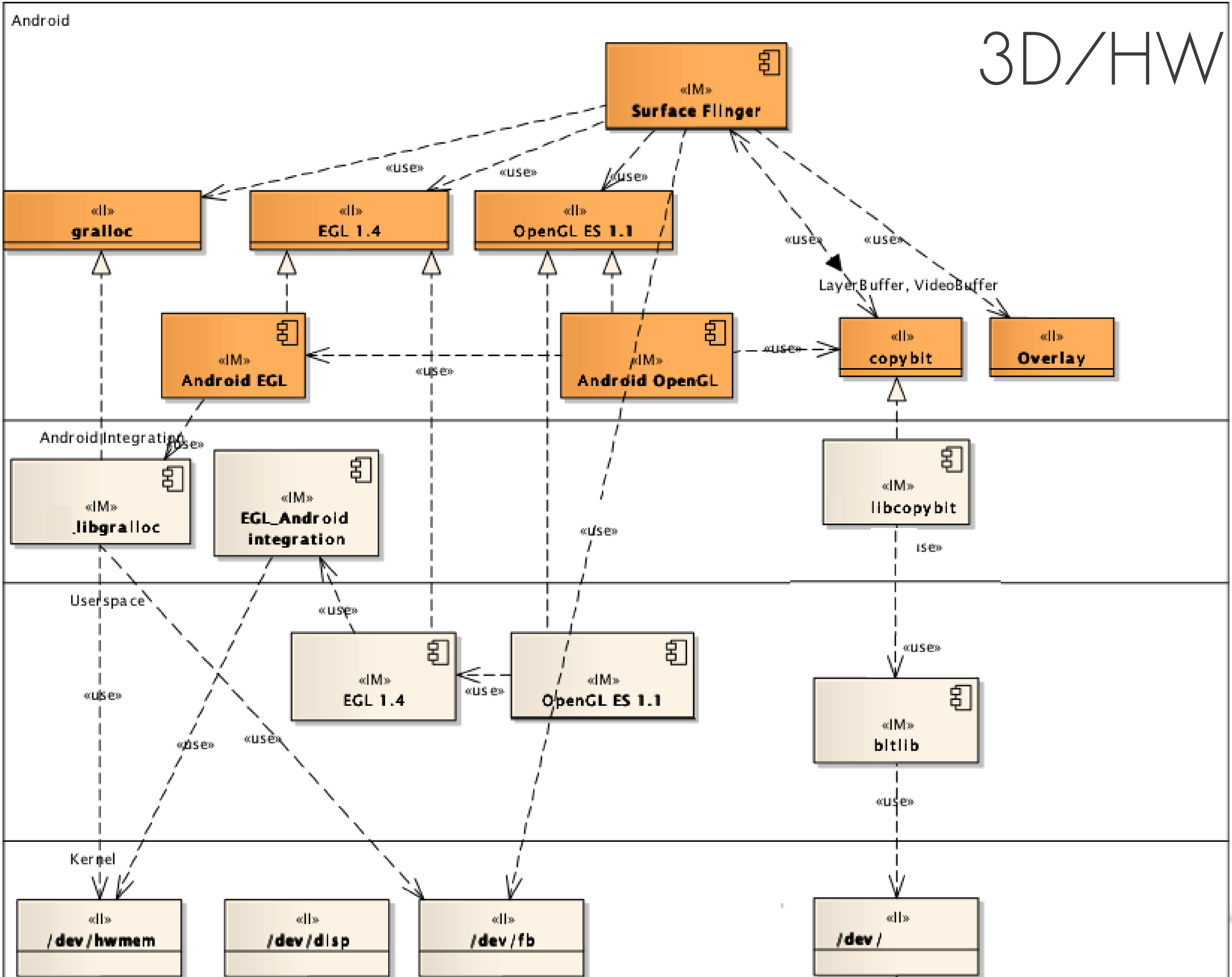
	advanced-performance2		advanced-performance		startpoint	
benchmark	3d-fps	linear	3d-fps	linear	3d-fps	linear
OpenGLCube	27.65	🟦	26.36	🟥	11.77	🟩
OpenGLBlending	15.21	🟦	15.06	🟥	8.78	🟩
OpenGLFog	14.03	🟦	13.86	🟥	8.36	🟩
FlyingTeapot	12.30	🟦	11.26	🟥	7.38	🟩



	← M3 + Linaro Toolchain		← M3	
benchmark →	3d-fps	linear	3d-fps	linear
OpenGLCube	29.06	🟦	29.04	🟥
OpenGLBlending	20.07	🟦	19.94	🟥
OpenGLFog	18.63	🟦	18.95	🟥
FlyingTeapot	17.49	🟦	17.04	🟥

This explains that we have several system tools and development flow to help customers/community to verify the performance and improve.

3D/HW


Android
«IM»
Surface Flinger
«use»
«use»
«use»
«II»
gralloc
«II»
EGL 1.4
«II»
OpenGL ES 1.1
«use»
«use»
LayerBuffer, VideoBuffer
«IM»
Android EGL
«use»
«IM»
Andrdid OpenGL
«use»
«II»
copybit
«II»
Overlay
AndroidIntegration
«use»
«IM»
libgralloc
«IM»
EGL_Android integration
«IM»
libcopybit
«use»
ise»
Userspace
«use»
«IM»
EGL 1.4
«use»
«IM»
OpenGL ES 1.1
«use»
«IM»
bltlib
«use»
«use»
«use»
«use»
Kernel
«II»
/dev/hwmem
«II»
/dev/disp
«II»
/dev/fb
«II»
/dev/

Optimizing Graphics with 3D/HW

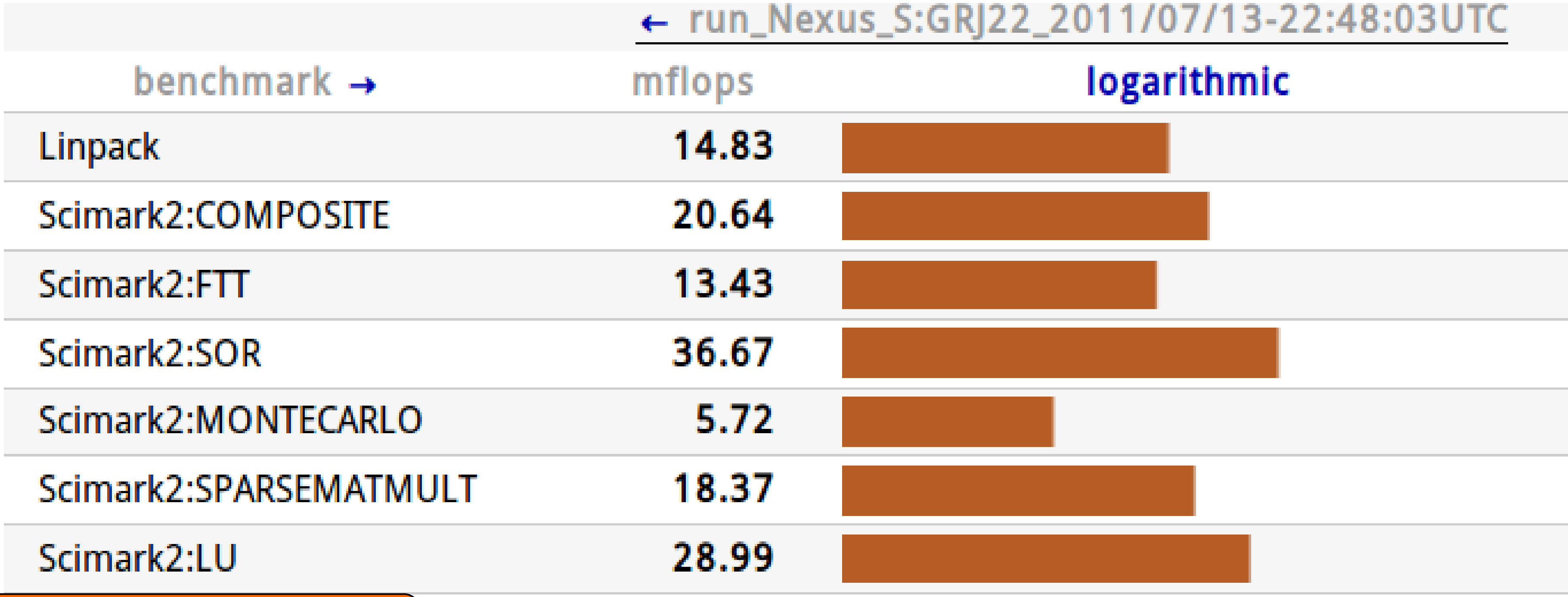
- The significant changes happen in applications and Android (Java) framework usage

  `http://developer.android.com/guide/practices/design/performance.html`
- Implement libgralloc carefully
  - Minimize the overhead of graphics memory allocator: the kernel helper
  - Example: UMP (Unified Memory Provider) in ARM Mali GPU
- Track the transactions inside SurfaceFlinger
  - Eliminate the invalid layer operations
  - Corresponding modifications in upper framework
- Still, page flipping benefits from libcopybit
  - but it has smaller difference with 3D/HW

Android Runtime

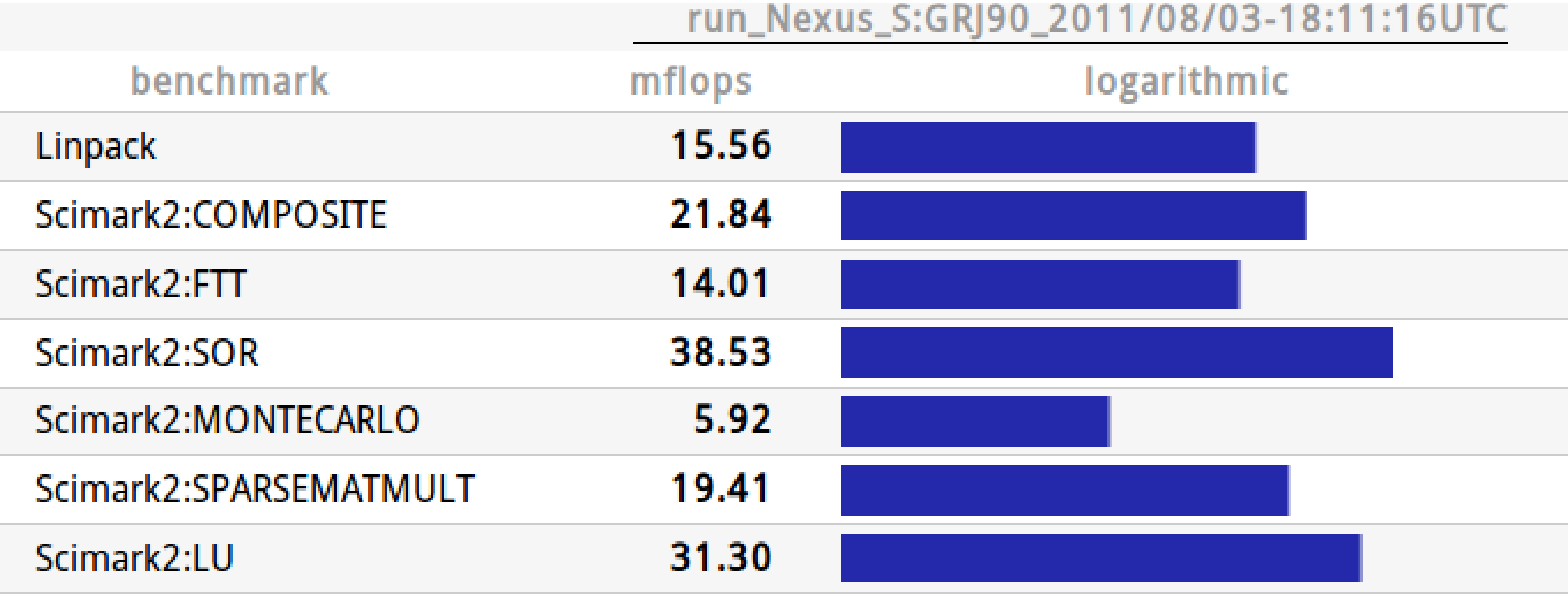
Arithmetic on Nexus S

Tune Dalvik VM performance (armv7)



← run_Nexus_S:GRJ22_2011/07/13-22:48:03UTC

benchmark →	mflops	logarithmic
Linpack	14.83	
Scimark2:COMPOSITE	20.64	
Scimark2:FTT	13.43	
Scimark2:SOR	36.67	
Scimark2:MONTECARLO	5.72	
Scimark2:SPARSEMATMULT	18.37	
Scimark2:LU	28.99	



run_Nexus_S:GRJ90_2011/08/03-18:11:16UTC

benchmark	mflops	logarithmic
Linpack	15.56	
Scimark2:COMPOSITE	21.84	
Scimark2:FTT	14.01	
Scimark2:SOR	38.53	
Scimark2:MONTECARLO	5.92	
Scimark2:SPARSEMATMULT	19.41	
Scimark2:LU	31.30	

Arithmetic Improvements

- Floating-point performance depends on Dalvik VM.
- Internally, Dalvik VM has huge amount of byte-swapped access, which can be improved by ARMv6's REV and REV16 instructions.

bionic/

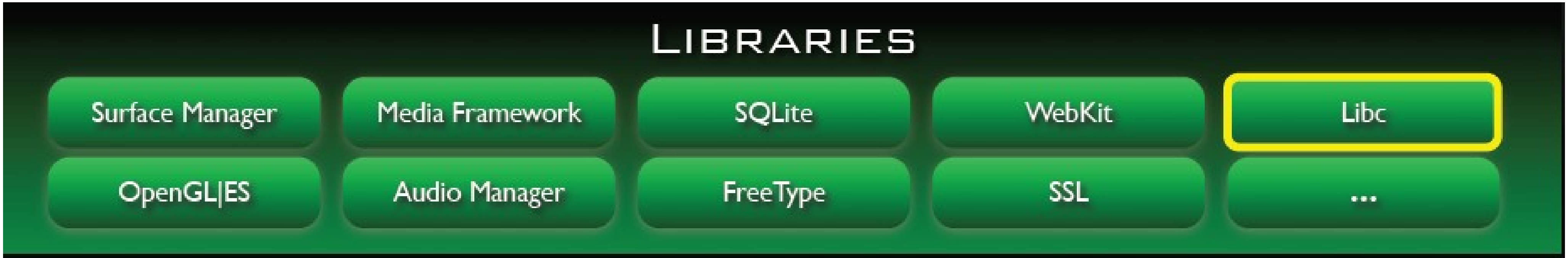
```
commit 02bee5724266c447fc4699c00e70d2cd0c19f6e1
    Use ARMv6 instruction for handling byte order

    ARMv6 ISA has several instructions to handle data in different
    byte order.
```

libcore/

```
commit 7d5299b162863ea898dd863004afe79f7a93fbce
    Optimize byte-swapped accesses.

    Brings the performance of byte-swapped accesses way down from about
    3x to less than 2x worst-case (char/short) and 20% best-case
    (long/double). The main active ingredients are switching to a
    single-pass swapped-copy (rather than copy in one pass, swap
    in a second pass), and ensuring we use ARM's REV and REV16
    instructions.
```

bionic libc

- Android C/C++ library
- 0xlab/Linaro Optimizations (merged in AOSP)
  - – Memory operations: Use ARMv6 unaligned access to optimize usual cases
    - Useful to TCP/IP (big-endian ↔ little endian)
  - – Various ARM optimized functions
    - memcpy, strcmp, strcpy, memset, memcpy, strlen
    - sha1
    - code size reduction: useful for recovery image



LIBRARIES

Surface Manager
Media Framework
SQLite
WebKit
Libc
OpenGL|ES
Audio Manager
FreeType
SSL
...

Prelinking in GNU world

(Quote from Embedded Linux optimizations – Size, RAM, speed, power, cost by Michael Opdenacker
Thomas Petazzoni, Free Electrons)

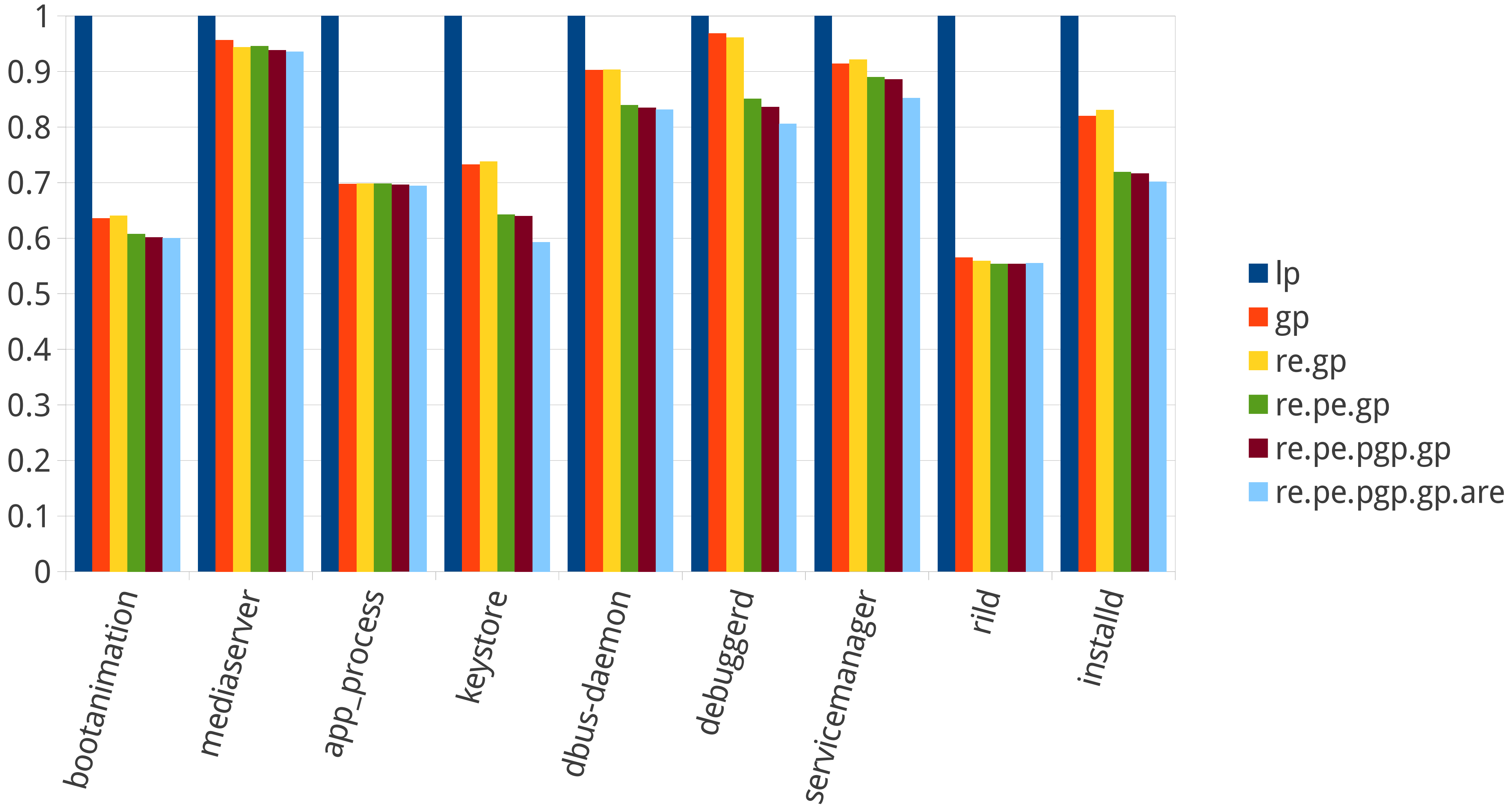
- `prelink`
  http://people.redhat.com/jakub/prelink/
- `prelink` modifies executables and shared libraries to simplify the dynamic linker relocation work.
- This can greatly reduce startup time for big applications (50% less for KDE!). This also saves memory consumed by relocations.
- Can be used to reduce the startup time of a Linux system.
- Just needs to be run again when libraries or executables are updated.

  Details on http://elinux.org/Pre_Linking

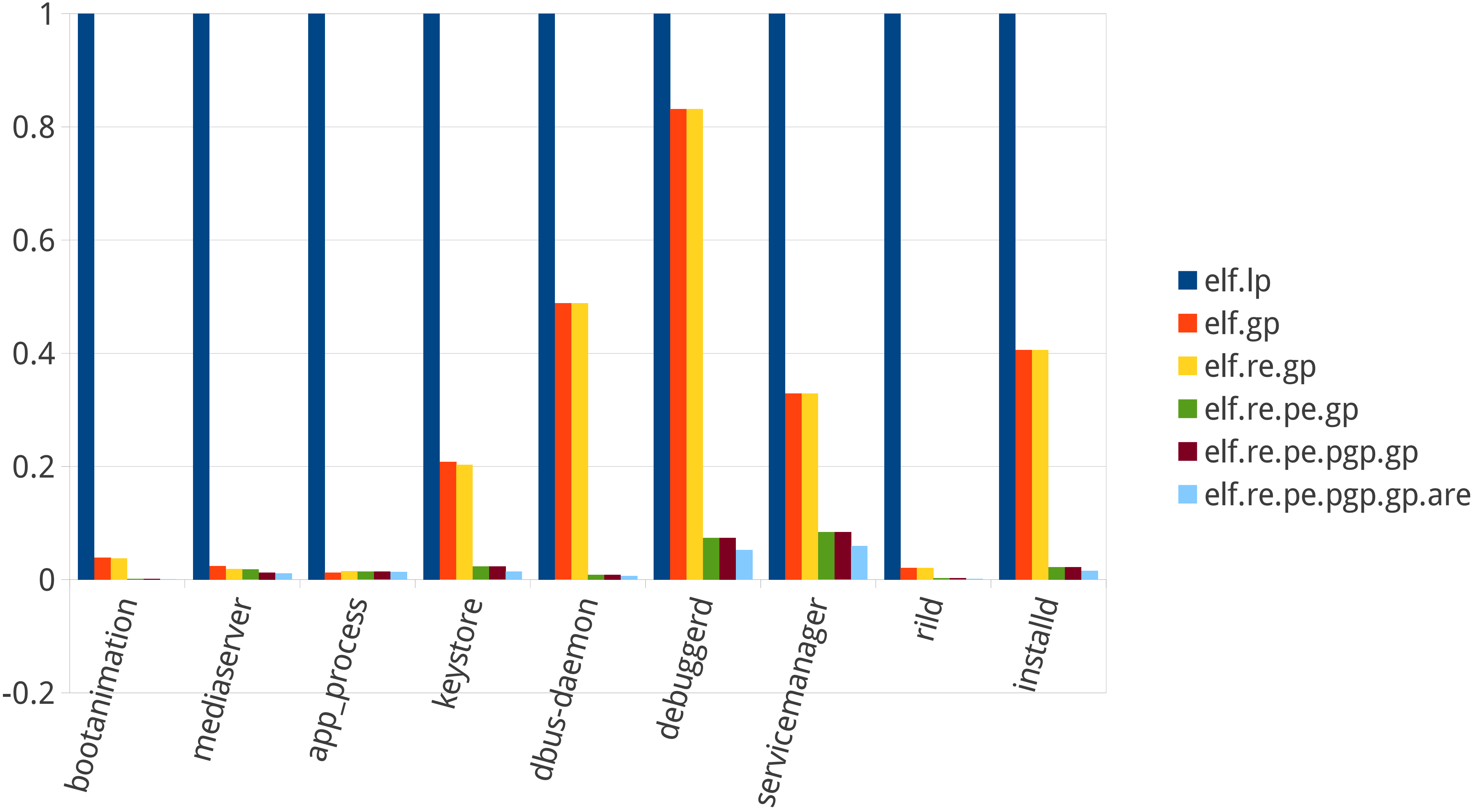
Dynamic Linker Optimization: Why and How?

- The major reason to optimize dynamic linker is to speed up application startup time.
- Approaches:
  - Implement GNU style hash support for bionic linker
  - Prelinker improvements: incremental global prelinking
    - reduce the number of ELF symbol lookup aggressively
- Changed parts
  - apriori, soslim, linker, elfcopy, elfutils

(normalized) Dynamic Link time



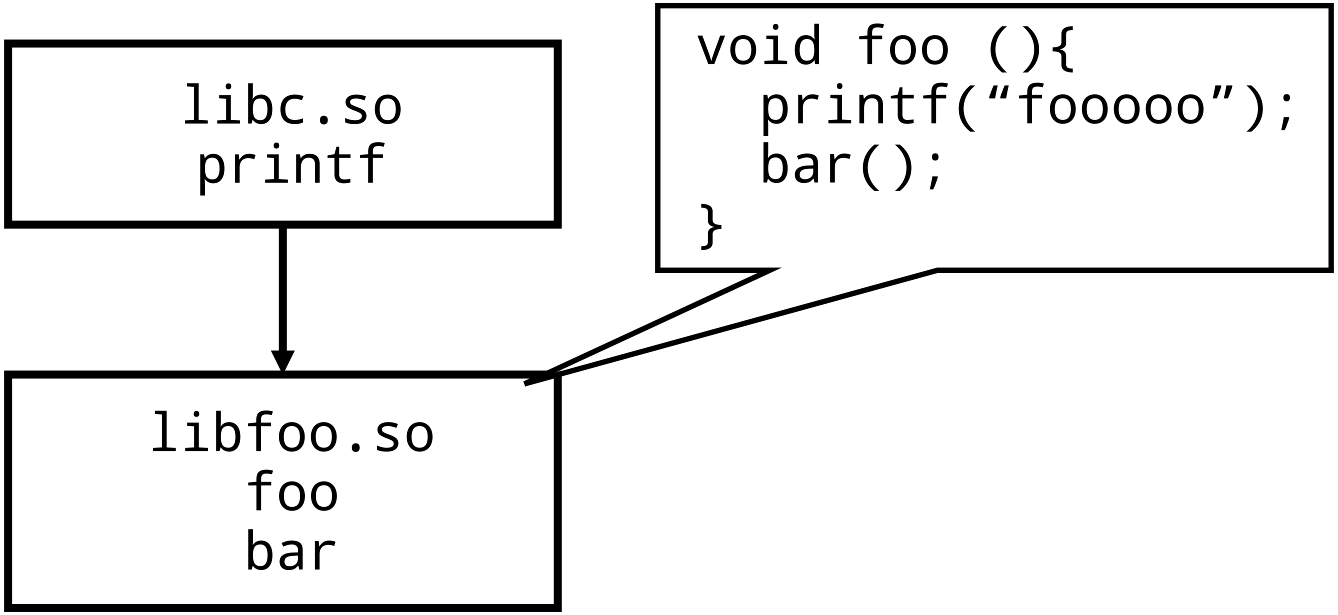
(normalized) Symbol Lookup number



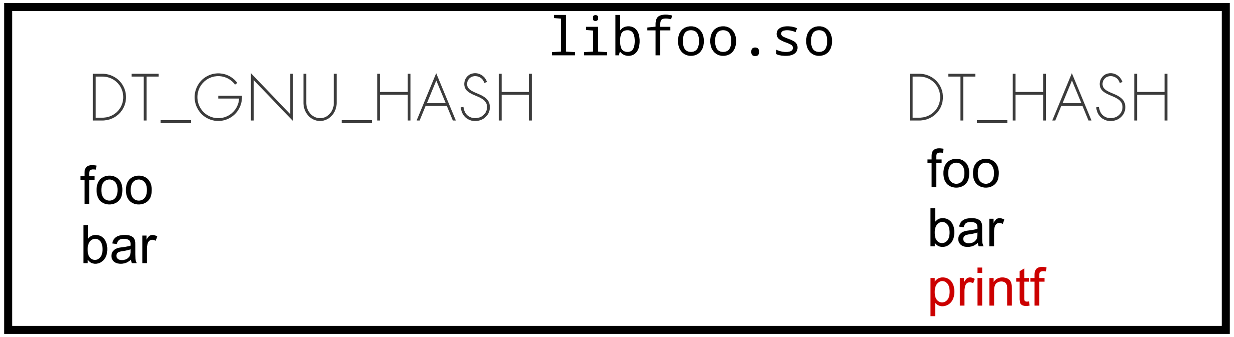
- DT_GNU_HASH: visible dynamic linking improvement = Better hash function (few collisions)

  + Drop unnecessary entry from hash

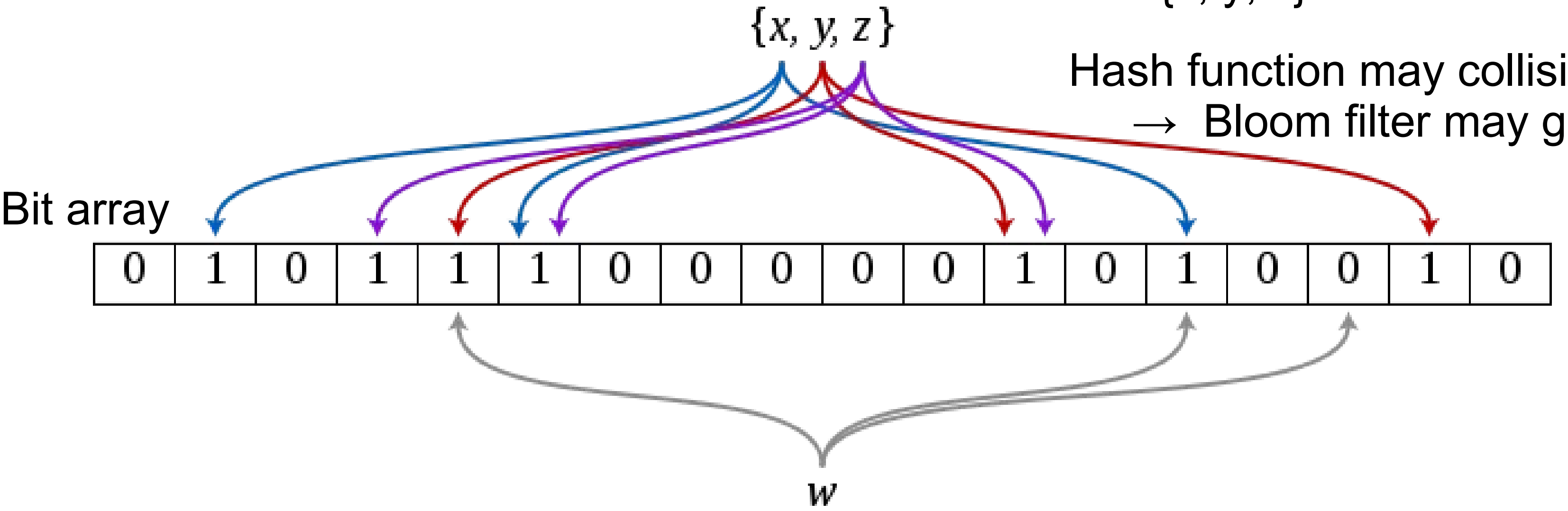
  + Bloom filter



```
void foo (){
  printf(“fooooo”);
  bar();
}
```



	Symbols in ELF	lookup#	fail#	gnu hash	filtered by bloom
gnu.gp	3758	23702	19950	23310	18234 (78%)
gnu.gp.re	3758	20544	16792	19604	14752 (75%)
gnu.lp	61750	460996	399252	450074	345032 (76%)
gnu.lp.re	61750	481626	419882	448492	342378 (76%)

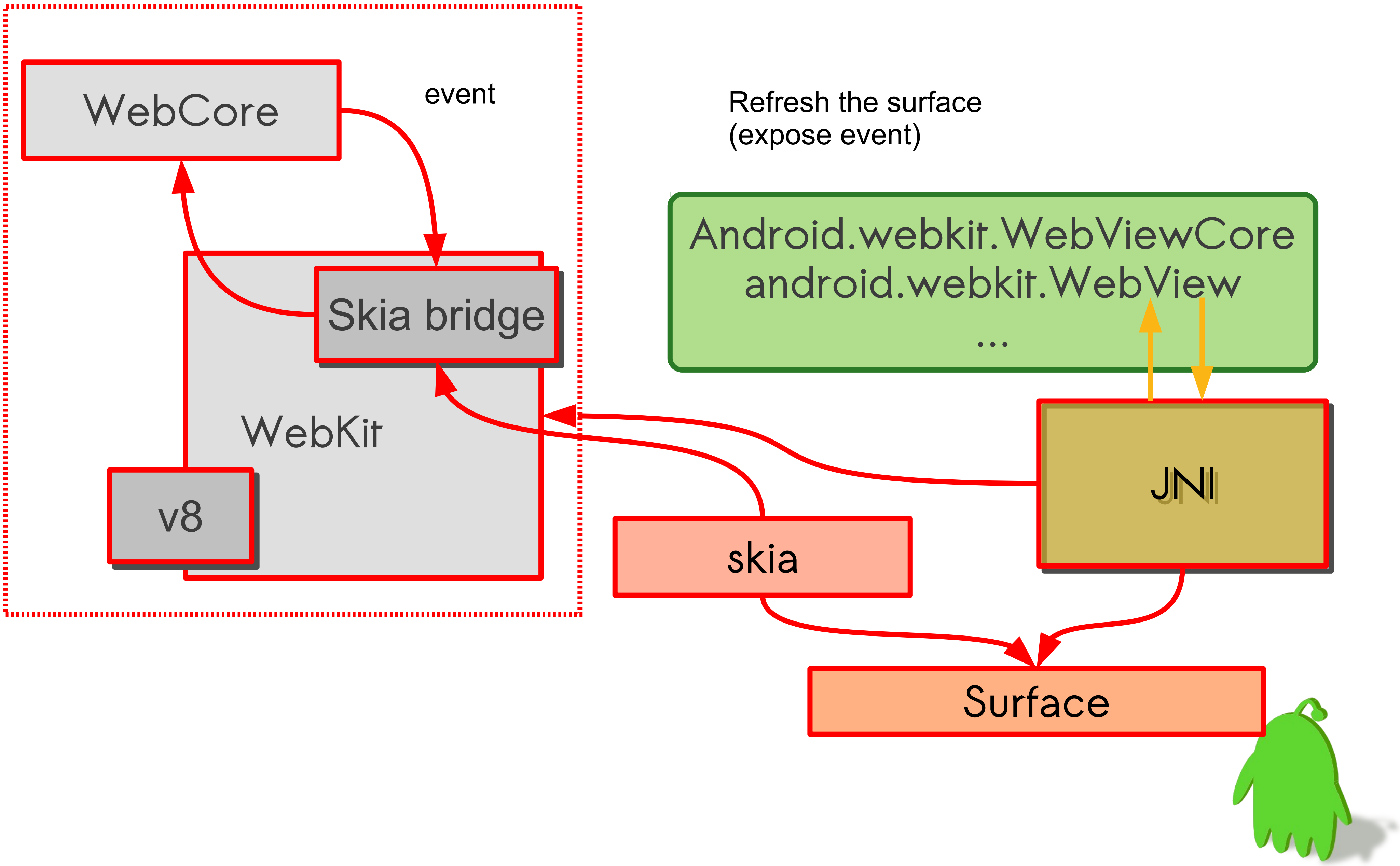


H = {x, y, z} = hash functions

Hash function may collision
→ Bloom filter may got false positives

NOTE: Android 4.0 removes the support of prelinker, but gnu style hash is still useful.

Case Study: WebKit in Android



How to Measure On Android/ARM?

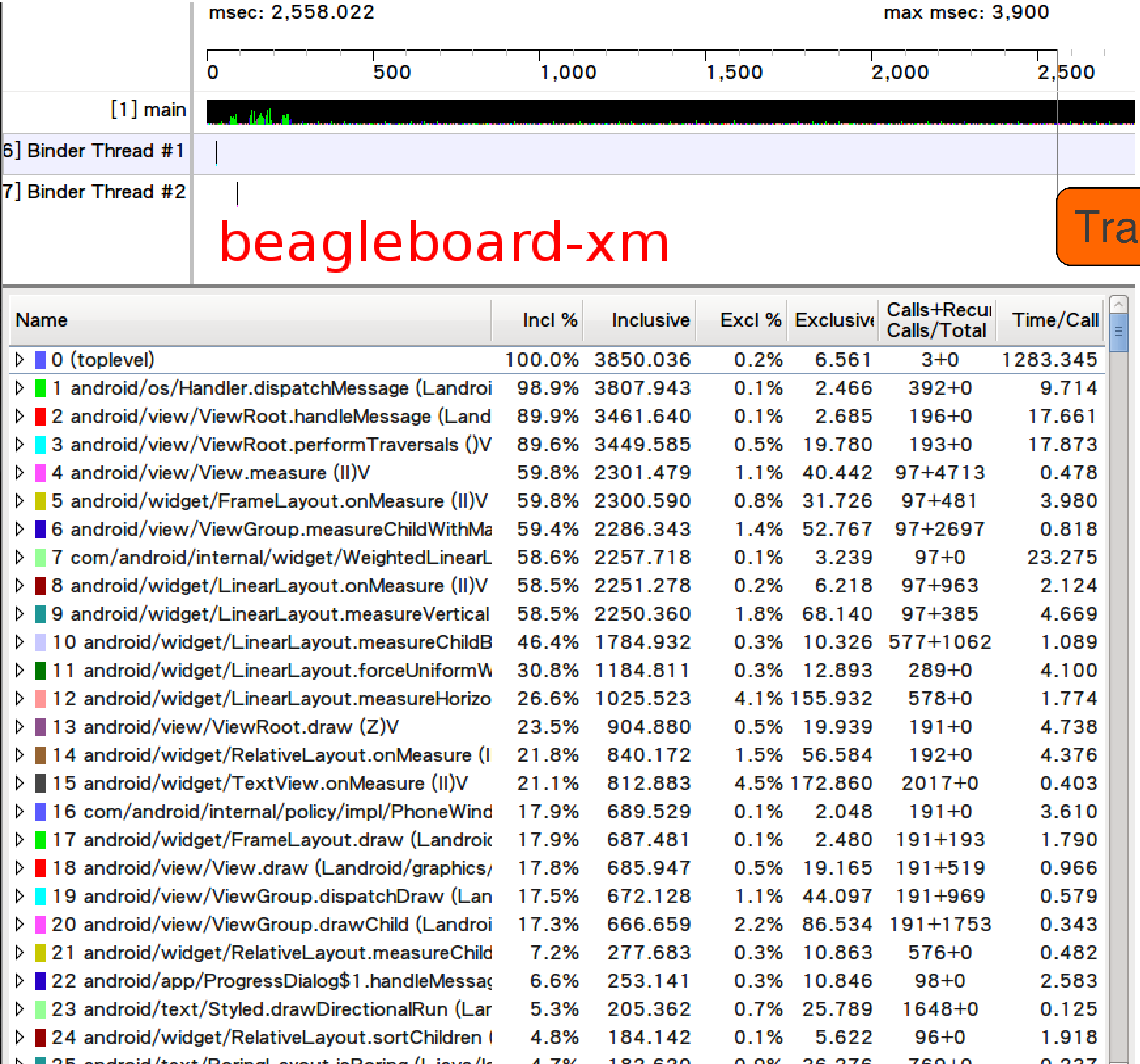
- for Native libraries →
  - Use '**perf**' built without libperl, libpython
  - oprofiled and opcontrol are there, CPU data is missing
  - Binaries for ARM need frame pointers to have backtraces
- Java part is the performance hell always.
  - **traceview** is a great tool for Java performance analysis.
  - JVMTI / JDWP (Java Debug Wire Protocol, normally spoken between a VM and a debugger)

```
# Overhead          Command          Shared Object   Symbol
# ........  ...............  .....................  ......
#
    89.23%    system_server                2b0c6c  [.] 0x000000002b0c6c
     1.26%     MLVdo_thread  [kernel_helper]         [k] 0x0000000017aa90
     1.05%  d.process.acore  libskia.so              [.] S32A_Opaque_BlitRow32_arm
     0.83%  d.process.acore  libcutils.so            [.] android_memset32
     0.63%    system_server  libc.so                 [.] memcpy
     0.63%  d.process.acore  libc.so                 [.] memset
```

system_server is the process name of Android Framework runtime. It occupies most of CPU resources, but it is hard to figure out details only by native tools like perf.

We can always optimize known performance hotspot routines such as S32A_Opaque_BlitRow32_arm but should be measured in advance.

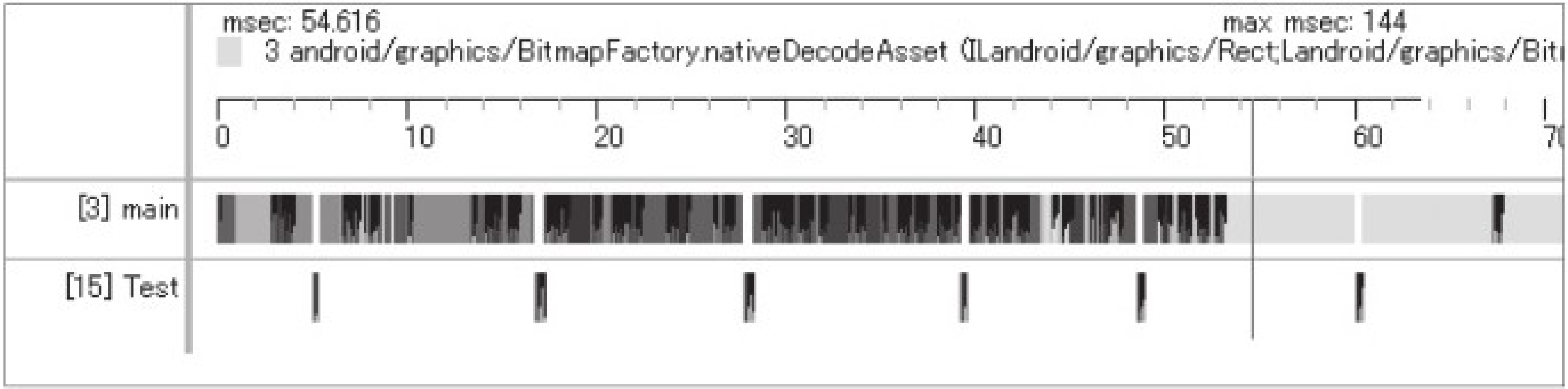
Traceview (java)



beagleboard-xm

Name	Incl %	Inclusive	Excl %	Exclusive	Calls+Recu Calls/Total	Time/Call
▷ ■ 0 (toplevel)	100.0%	3850.036	0.2%	6.561	3+0	1283.345
▷ ■ 1 android/os/Handler.dispatchMessage (Landroi	98.9%	3807.943	0.1%	2.466	392+0	9.714
▷ ■ 2 android/view/ViewRoot.handleMessage (Land	89.9%	3461.640	0.1%	2.685	196+0	17.661
▷ ■ 3 android/view/ViewRoot.performTraversals ()V	89.6%	3449.585	0.5%	19.780	193+0	17.873
▷ ■ 4 android/view/View.measure (II)V	59.8%	2301.479	1.1%	40.442	97+4713	0.478
▷ ■ 5 android/widget/FrameLayout.onMeasure (II)V	59.8%	2300.590	0.8%	31.726	97+481	3.980
▷ ■ 6 android/view/ViewGroup.measureChildWithMa	59.4%	2286.343	1.4%	52.767	97+2697	0.818
▷ ■ 7 com/android/internal/widget/WeightedLinearL	58.6%	2257.718	0.1%	3.239	97+0	23.275
▷ ■ 8 android/widget/LinearLayout.onMeasure (II)V	58.5%	2251.278	0.2%	6.218	97+963	2.124
▷ ■ 9 android/widget/LinearLayout.measureVertical	58.5%	2250.360	1.8%	68.140	97+385	4.669
▷ ■ 10 android/widget/LinearLayout.measureChildB	46.4%	1784.932	0.3%	10.326	577+1062	1.089
▷ ■ 11 android/widget/LinearLayout.forceUniformW	30.8%	1184.811	0.3%	12.893	289+0	4.100
▷ ■ 12 android/widget/LinearLayout.measureHorizo	26.6%	1025.523	4.1%	155.932	578+0	1.774
▷ ■ 13 android/view/ViewRoot.draw (Z)V	23.5%	904.880	0.5%	19.939	191+0	4.738
▷ ■ 14 android/widget/RelativeLayout.onMeasure (I	21.8%	840.172	1.5%	56.584	192+0	4.376
▷ ■ 15 android/widget/TextView.onMeasure (II)V	21.1%	812.883	4.5%	172.860	2017+0	0.403
▷ ■ 16 com/android/internal/policy/impl/PhoneWind	17.9%	689.529	0.1%	2.048	191+0	3.610
▷ ■ 17 android/widget/FrameLayout.draw (Landroic	17.9%	687.481	0.1%	2.480	191+193	1.790
▷ ■ 18 android/view/View.draw (Landroid/graphics/	17.8%	685.947	0.5%	19.165	191+519	0.966
▷ ■ 19 android/view/ViewGroup.dispatchDraw (Lan	17.5%	672.128	1.1%	44.097	191+969	0.579
▷ ■ 20 android/view/ViewGroup.drawChild (Landroi	17.3%	666.659	2.2%	86.534	191+1753	0.343
▷ ■ 21 android/widget/RelativeLayout.measureChild	7.2%	277.683	0.3%	10.863	576+0	0.482
▷ ■ 22 android/app/ProgressDialog$1.handleMessag	6.6%	253.141	0.3%	10.846	98+0	2.583
▷ ■ 23 android/text/Styled.drawDirectionalRun (Lar	5.3%	205.362	0.7%	25.789	1648+0	0.125
▷ ■ 24 android/widget/RelativeLayout.sortChildren (	4.8%	184.142	0.1%	5.622	96+0	1.918
▷ ■ 25 android/text/BoringLayout.isBoring (Ljava/la	4.7%	182.620	0.9%	36.376	760+0	0.237

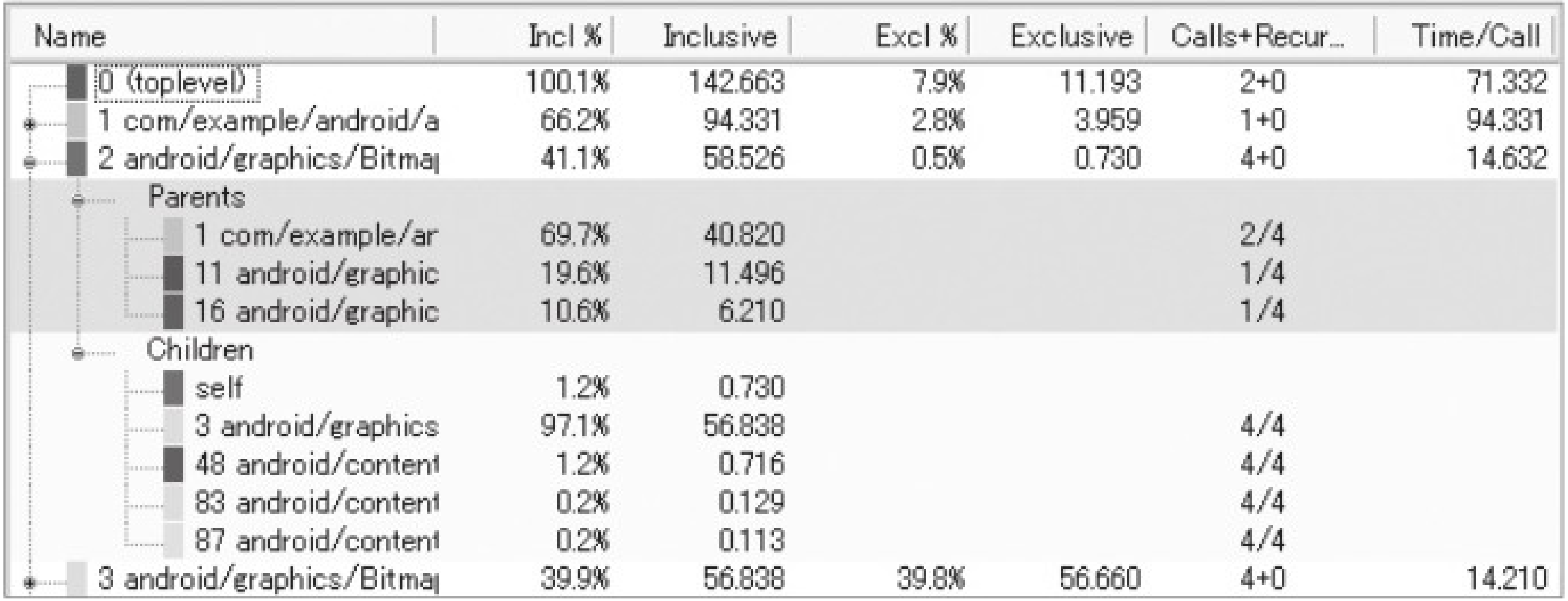
Timeline Panel



msec: 54.616 max msec: 144

3 android/graphics/BitmapFactory.nativeDecodeAsset (ILandroid/graphics/Rect;Landroid/graphics/Bit

Profile Panel



Name	Incl %	Inclusive	Excl %	Exclusive	Calls+Recur...	Time/Call
0 (toplevel)	100.1%	142.663	7.9%	11.193	2+0	71.332
1 com/example/android/a	66.2%	94.331	2.8%	3.959	1+0	94.331
2 android/graphics/Bitma	41.1%	58.526	0.5%	0.730	4+0	14.632
Parents						
1 com/example/ar	69.7%	40.820			2/4	
11 android/graphic	19.6%	11.496			1/4	
16 android/graphic	10.6%	6.210			1/4	
Children						
self	1.2%	0.730				
3 android/graphics	97.1%	56.838			4/4	
48 android/content	1.2%	0.716			4/4	
83 android/content	0.2%	0.129			4/4	
87 android/content	0.2%	0.113			4/4	
3 android/graphics/Bitma	39.9%	56.838	39.8%	56.660	4+0	14.210

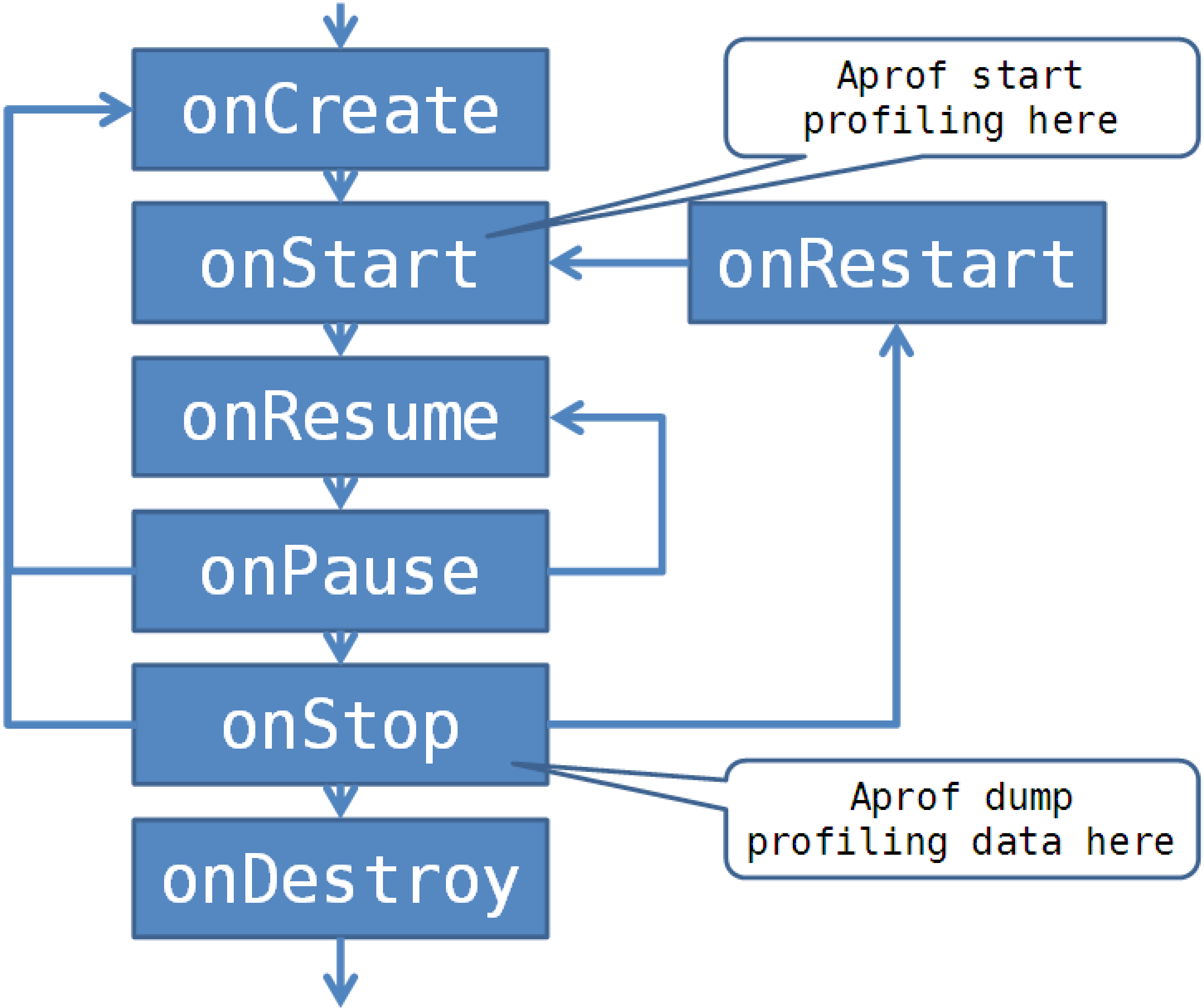
Approaches to Optimize WebKit

- Cherry-pick upstream enhancements
  - Example: ARM NEON optimized renderer and blur effects
- Track JNI bridge in WebKit – Avoid memory leaks
- Use hardware accelerated backing store for certain UI actions such as scrolling
  - Check Qualcomm's QAEP
- Image caching in both skia and webkit
- Since skia supports GL backend, webkit can utilize the accelerated paths
  - That's what Android 4.0 emphasize on.

Case Study: Profiling JNI

- Aprof : an Android profiler (by 0xlab, android-platform@ mailing-list)
  - a profiling tool for Android native code; aprof is not only another gprof implement on Android but also support for profiling shared
- The capability of aprof is similar to what gprof does, it provides call graph and time sampling profiling, but it's incompatible with gprof since the gprof can not profile shared library.
  - Limited by its representation and the fact of bionic libc incompatibility with GNU world.
- Integrated with Android activity life-cycle

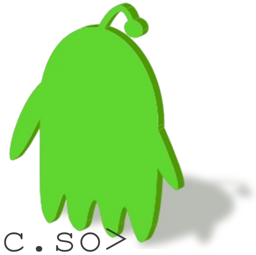
Aprof

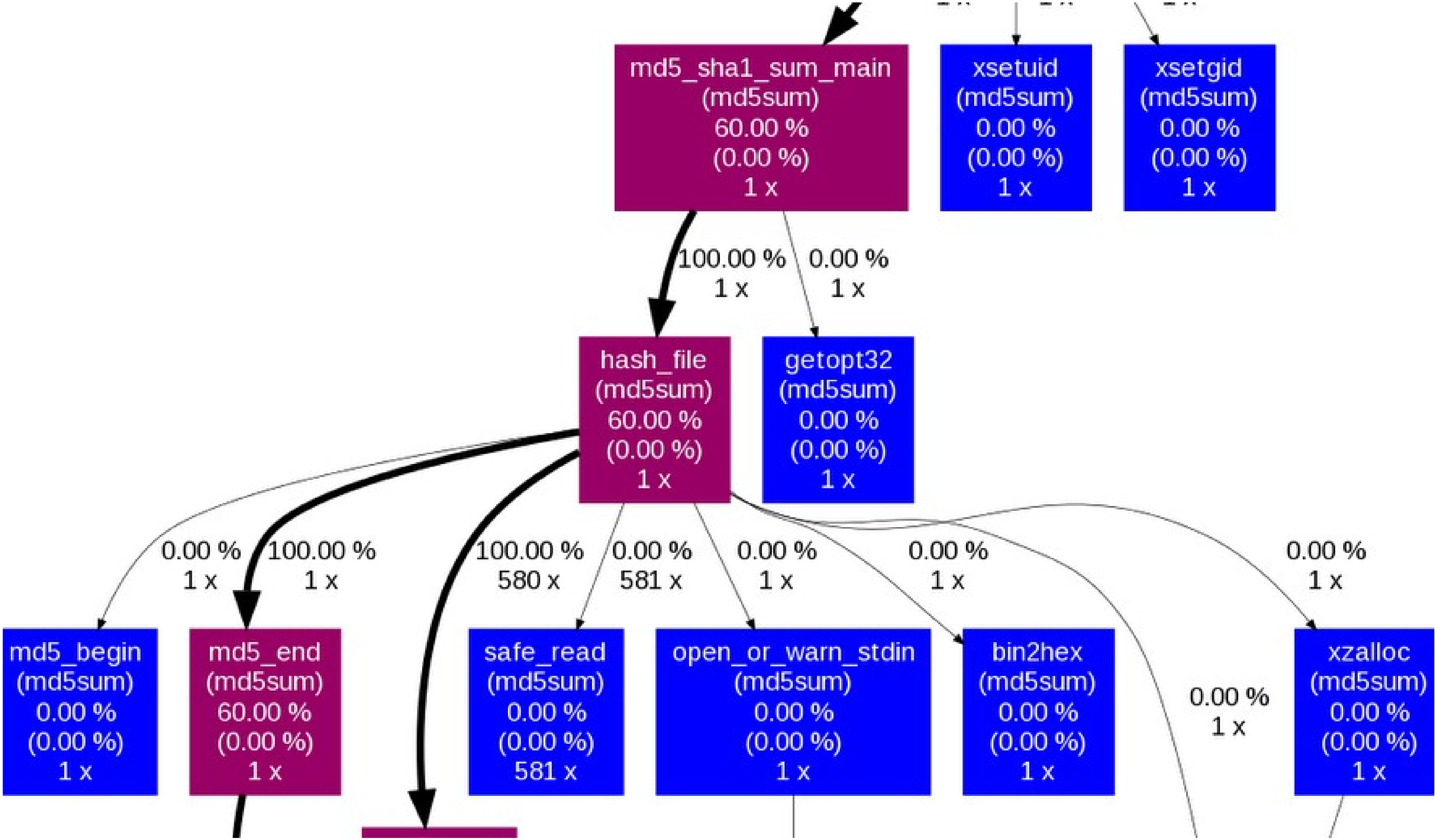


Aprof start profiling here

Aprof dump profiling data here

```
  %       cumulative     self                self      total
time        time         time       calls   ms/call   ms/call   name
99.52       2170         2140     2178309         0         0   fib
 0.00       2170            0           1         0       217   main
 0.48          0           30           0         0         0   <libc.so>
```



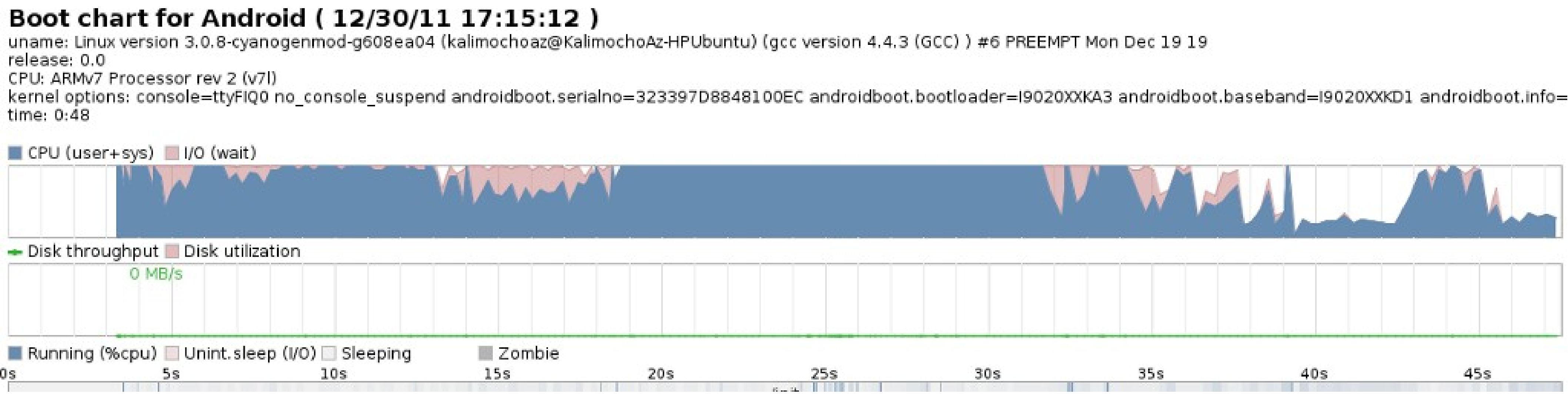


```
Android.mk
LOCAL_ENABLE_APROF := true
```

Android Boot Time Optimizations

Reducing Boot-Time is Art

- You have to take every piece of boot flow into consideration.
- Linux Kernel itself usually contributes less time than userspace.



Boot chart for Android (12/30/11 17:15:12)

```
uname: Linux version 3.0.8-cyanogenmod-g608ea04 (kalimochoaz@KalimochoAz-HPUbuntu) (gcc version 4.4.3 (GCC) ) #6 PREEMPT Mon Dec 19 19
release: 0.0
CPU: ARMv7 Processor rev 2 (v7l)
kernel options: console=ttyFIQ0 no_console_suspend androidboot.serialno=323397D8848100EC androidboot.bootloader=I9020XXKA3 androidboot.baseband=I9020XXKD1 androidboot.info=
time: 0:48
```

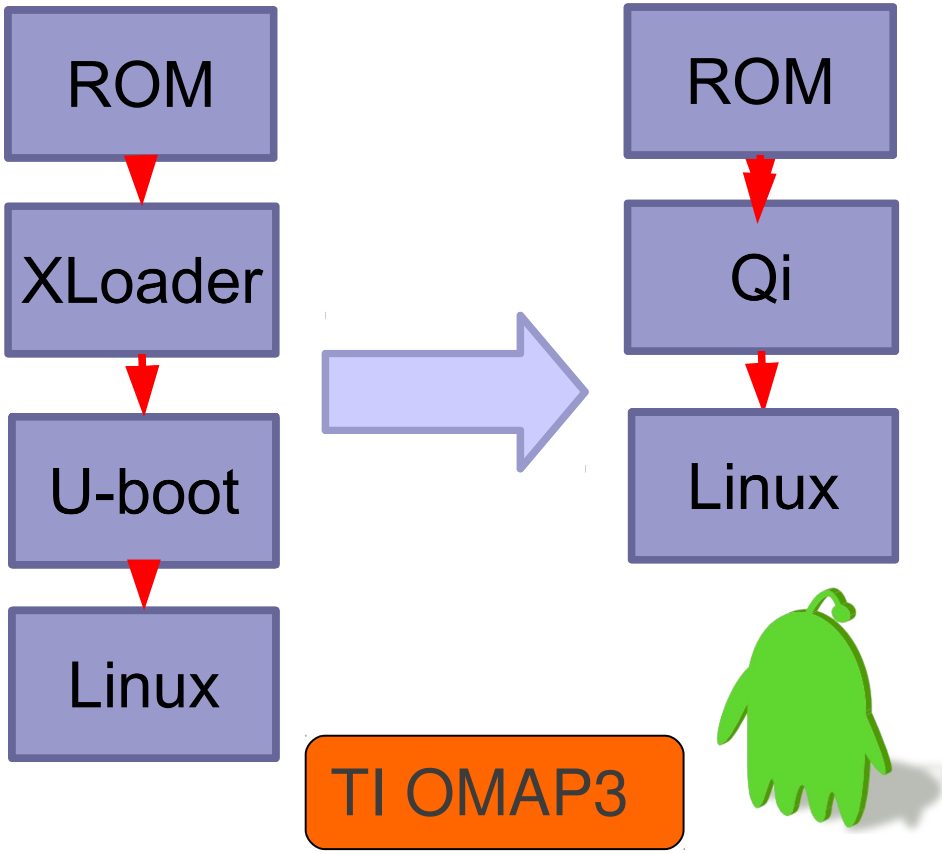
Bootchart of Android 4.0 on Nexus S
We will focus on reducing "cold" boot time,
from power on to the execution of the system application.

Write Tiny Boot loader to Speed up

	Qi Boot-oader	U-Boot + XLoader
Size	~30K	~270K+20K
Time to Kernel	< 1 s	> 5s
Usage	Product	Engineering
Code	Simple	Complicated

Qi Boot-loader

- Only **one stage** boot-loader
- Small footprint ~**30 KB**
- Currently support
  - Freescale iMX31
  - Samsung S3C24xx
  - Beagleboard
- KISS concept
  - Boot device and load kernel
  - 3 second reduction!



Optimized ARM Hibernation

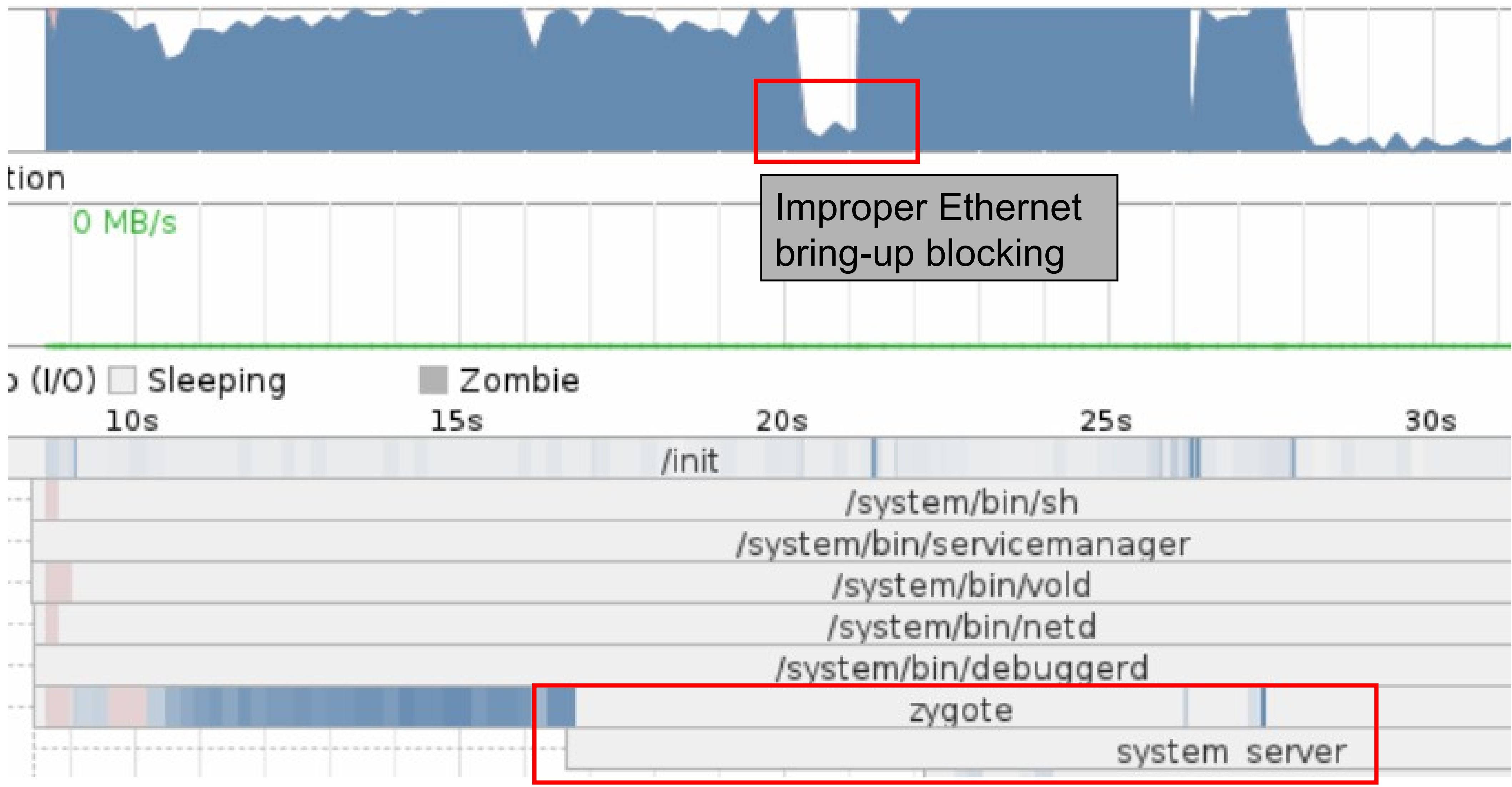
- Based on existing technologies and little modifications to userspace are required
  - TuxOnIce
- Release clean-pages before suspend
- Swap out dirty-pages before save image
- Image size reduced leads to faster resume time.

Demo video: http://www.youtube.com/watch?v=pvcQiiikJDU
Beagleboard-xM (OMAP3)
Full source tree: http://gitorious.org/0xlab-kernel

Further Boot Time Optimizations

- Save the heap image (like core dump) of Zygote after preloading classes
- Modify Dalvik to make hibernation image after system init and before Launcher startup
- Parallize Android init
- Cache & share JIT'ed code fragment

Reference: File-Based Sharing For Dynamically Compiled Code On Dalvik Virtual Machine, National Chiao Tung University in Taiwan



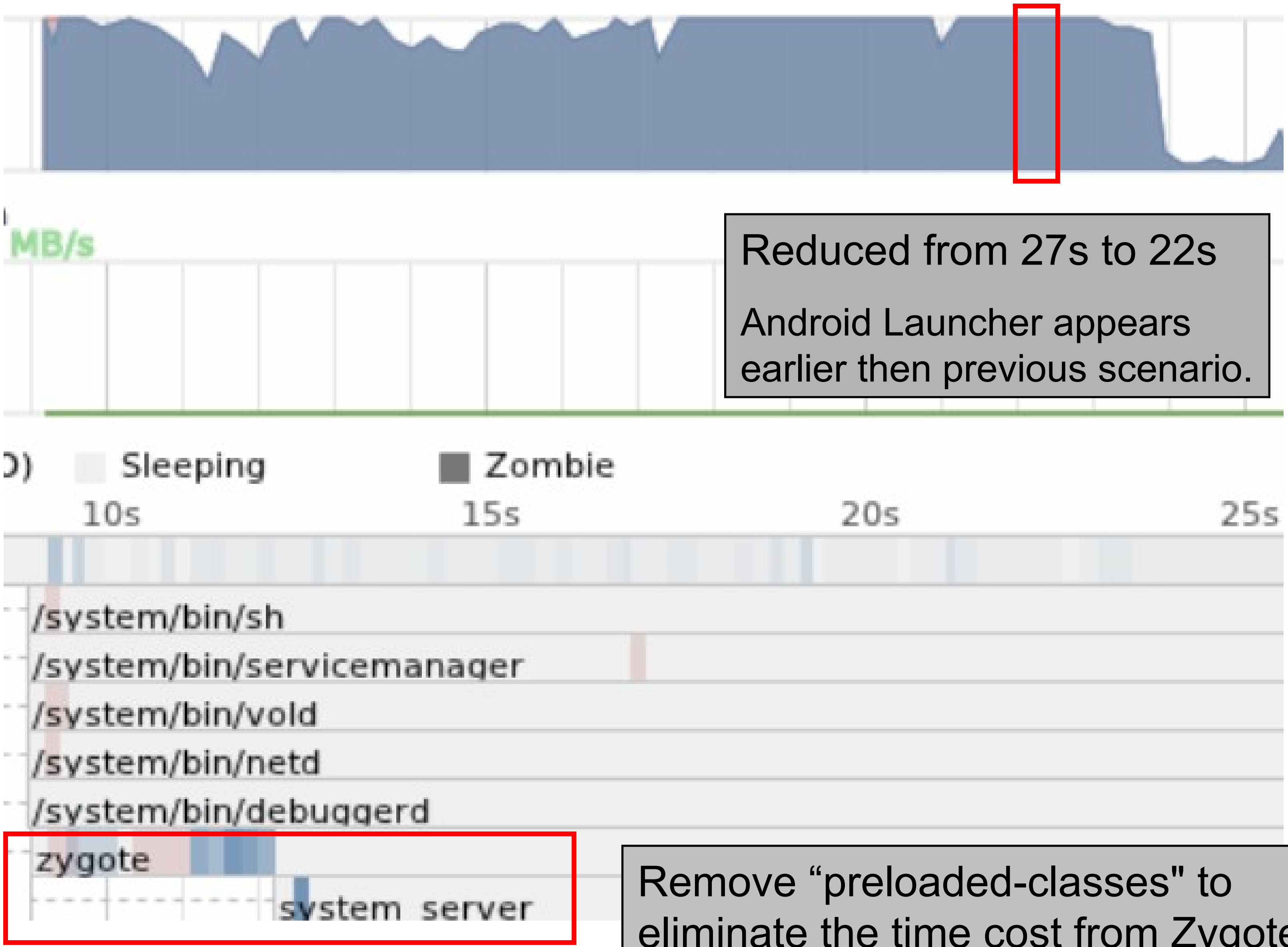
Improper Ethernet bring-up blocking

Initial bootchart analysis:

(1) It takes 27s from HW reset to Android Launcher screen.

(2) There is an improper Ethernet bring-up blocking for 2s.

(3) CPU usage looks busy.



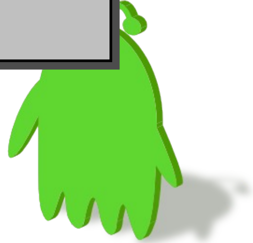
Reduced from 27s to 22s

Android Launcher appears earlier then previous scenario.

Remove “preloaded-classes" to eliminate the time cost from Zygote

Risk: potentially slower Android activity launch time

Remove unnecessary dependency to active services concurrently



Reduce boot time without Hibernation

- Zygote (init2) takes a long time to initialize Dalvik VM and Android framework, which are usually of the same context in virtual memory view
- If we can capture the state of a running process in Linux and save it to a file. This file can then be used to resume the process later on, either after a reboot or even on another machine.

  http://cryopid.berlios.de/

  https://ftg.lbl.gov/projects/CheckpointRestart/

  http://dmtcp.sourceforge.net/
- Only not zygote can benefit from from process freezing technique, but also system robustness might be improved.

Conclusion

- Optimizing Android requires the collaboration from community – verification, utilities, and upsteam
- UX is not as simple as its length.
  - Always Do measurement before taking actions
  - Hacking around the software stack
- Automated testing + continuous integration is really important.

0xlab

http://0xlab.org