

Develop Community-based Android Distribution and Upstreaming Experience

Jim Huang (黃敬群) <jserv@0xlab.org>

Developer & Co-Founder, 0xlab

<http://0xlab.org/>

Rights to copy

© Copyright 2011 **0xlab**

<http://0xlab.org/>

contact@0xlab.org



Attribution – ShareAlike 3.0

You are free

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Corrections, suggestions, contributions and translations are welcome!

Latest update: Nov 23, 2011

Under the following conditions

- **BY:** **Attribution.** You must give the original author credit.
- **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.
- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text: <http://creativecommons.org/licenses/by-sa/3.0/legalcode>





connect your device to application

0xlab

$0x1ab = 16^2 + 16 \times 10 + 11 = 427$
(founded on April 27, 2009)

0xlab is another Hexspeak.



About Me

- (1) Come from Taiwan
- (2) Contributor of Android
Open Source Project (AOSP)
- (3) Developer, Linaro
- (4) Contributed to GNU
Classpath / Kaffe, Linux
internationalization (i18n),
Openmoko



Commercial Partners of Oxlab

ARM / Linaro

Oxlab contributes to Linaro Android since the first line of code

AzureWave

Build wireless networking & image processing solutions

Mediatek

Android based consumer products

Open Embedded Software Foundation

Oxlab contributes to the reference implementation

<http://Oxlab.org/partners.html>

Eventually, partners can benefit from open source efforts and our experience.



Agenda

- (1) Reasons to build Android distribution
- (2) Lesson learned from AOSP
- (3) The methods to enhance Android software stack
- (4) Go upstream





Reasons to build Android Distribution

based on non typical open source projects



The reason why we built community Android Distribution:

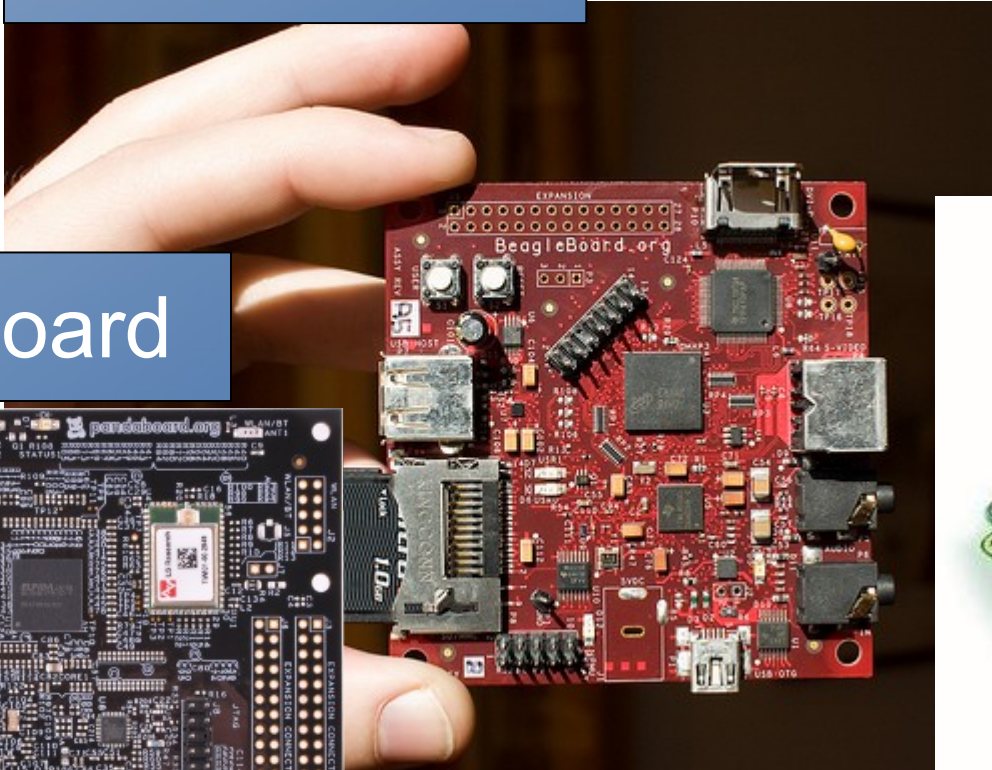
Initially, we just wanted to enable wireless connectivity features on Android per our hardware partners requested.

But, we never thought that it was difficult to do things efficiently.

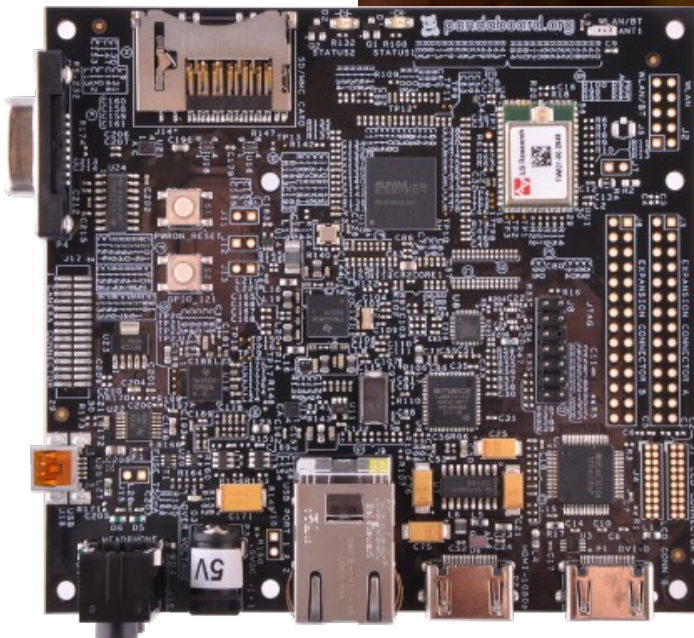


Oxdroid: enhanced Android distribution

Beagleboard



Pandaboard



DevKit8000



We suffered from performance and usability issues in AOSP. Oxdroid is basically the environment where we can develop and experiment.



What does Oxdroid deliver?

- Hardware enablement: Beagleboard (TI OMAP3), Pandaboard (TI OMAP4), Snowball (ST-Ericsson Ux500)
- Provide full source code for HAL
- Usability: software cursor, window manager fix, large screen tweaks, network connectivity fix
- Performance: ARM specific optimizations, graphics enhancement
- Features: Bluetooth HID (keyboard/mouse), external modem, 3D effects, customized Launcher



But, Oxlab is not really making yet another Android distribution.

Our idea is to help community.

Oxdroid is just a *testbed* (or reference implementation), and the valuable changes should be merged in upstream or other community projects.



Strategy and Policy

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



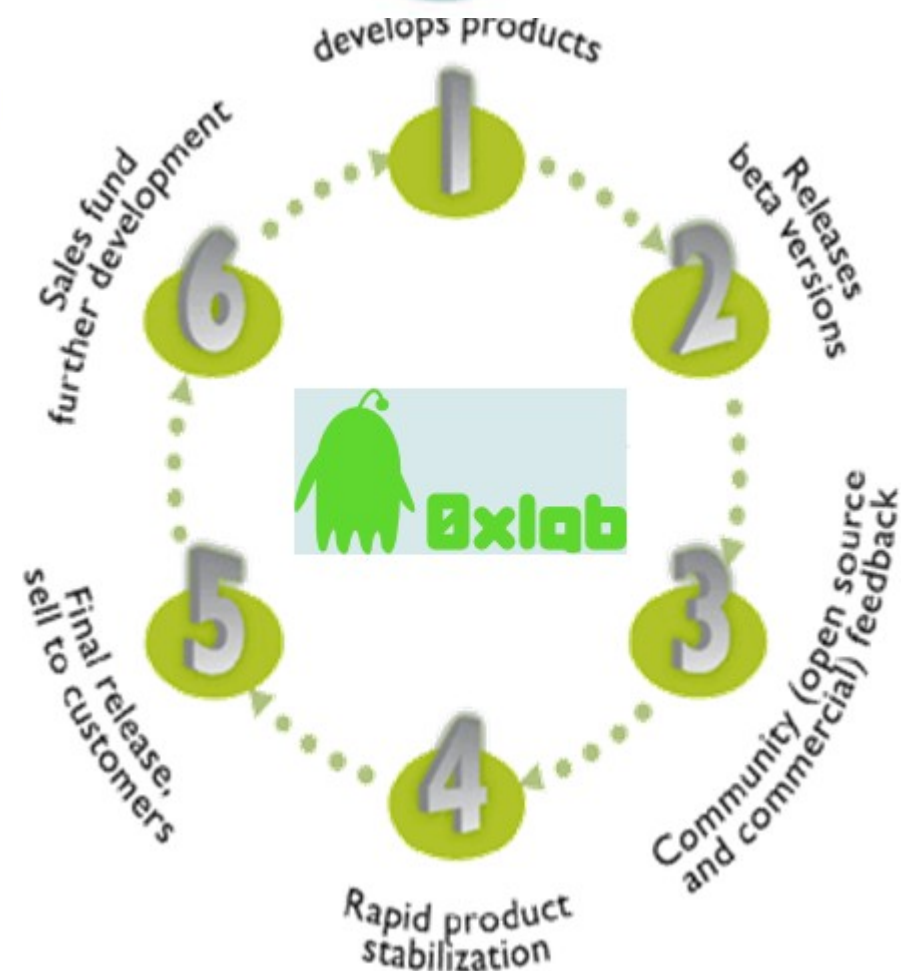


Working Model

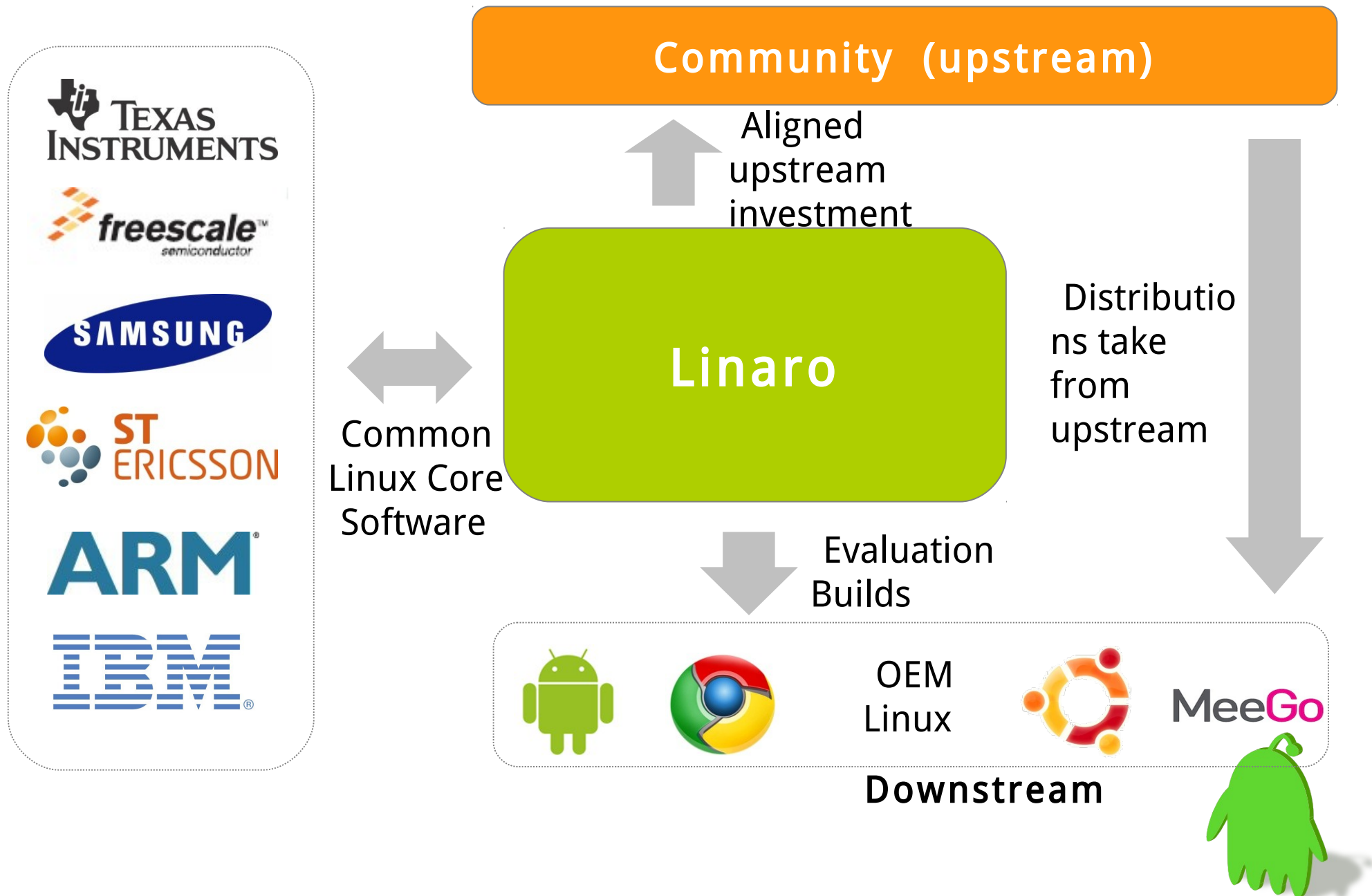
Rowboat

CyanogenMod

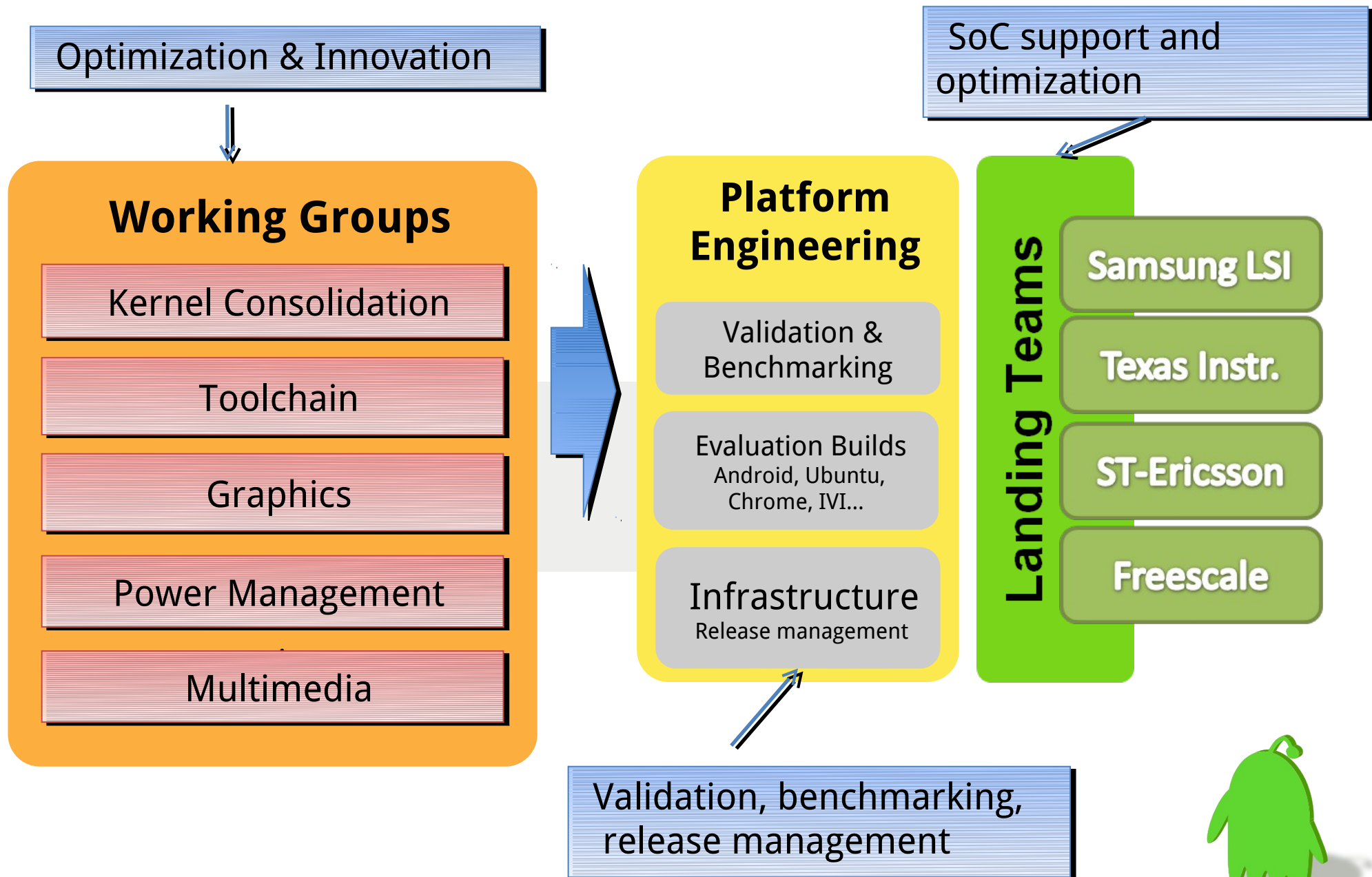
Android-x86



Where does Linaro fit?



Linaro Engineering Unit



Joint-Development Efforts

(demo video)

- Milestone 0x4
 - <http://www.youtube.com/watch?v=OGpYk1p1UPI>
simplified deployment, theme-able Android launcher, Automated Testing Infrastructure (esuit + CTS), external modem support, 3D Eye-candy Effects, Performance Improvements for ARM
- Milestone 0x7
 - http://www.youtube.com/watch?v=_3dT68MOzz0
Linaro powered infrastructure, further ARM optimizations, real fast boot-time solution for Android, validation framework





Lesson Learned from AOSP



Let's go Upstream!

Unfortunately, contributing to AOSP
is an __art__.

You never know how Google thinks of your
patches exactly, even through Gerrit (code review system).



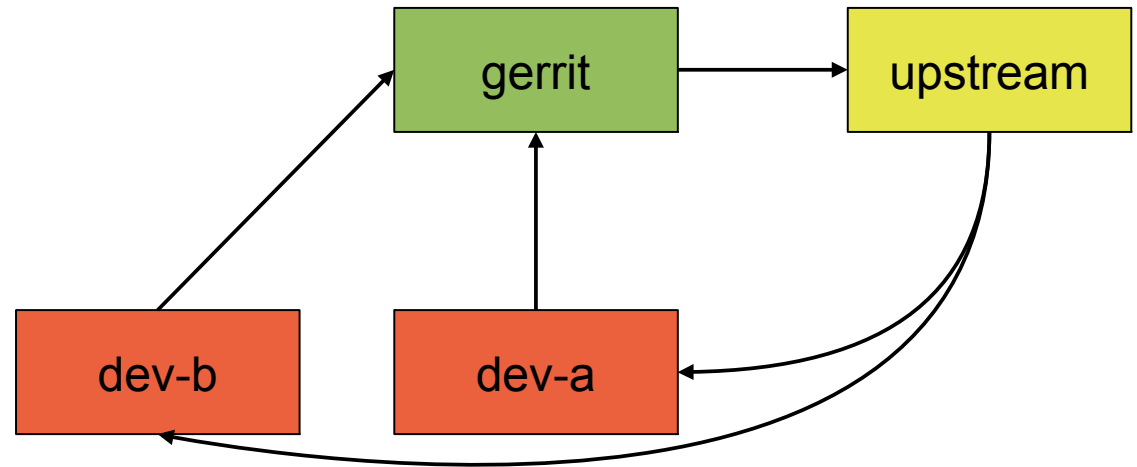
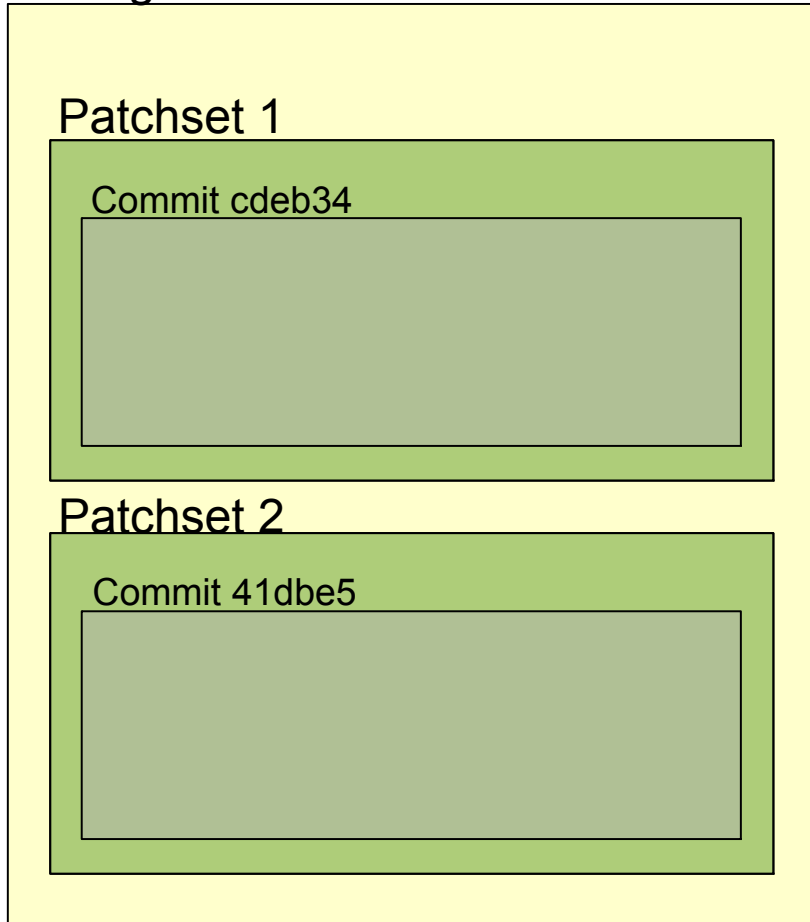
Problems We faced

- AOSP looks like “An Open Source Protector”
- No public roadmap
 - Therefore, we ignore the modifications against Android framework in public activities.
- The merged changes usually show up in next 1 or 2 public release.
 - It is really hard to introduce/track the relevant changes.
- Not clear discussions on android-contrib mailing-list. Sometimes, you have to have private communications to Google engineers.
- Version control / Code Review on invisible repositories (internal and far-away GIT tree)



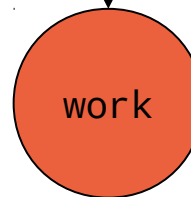
Change represented in Gerrit

Change 123



Create
Local
Branch

`% git checkout -b topic-branch`



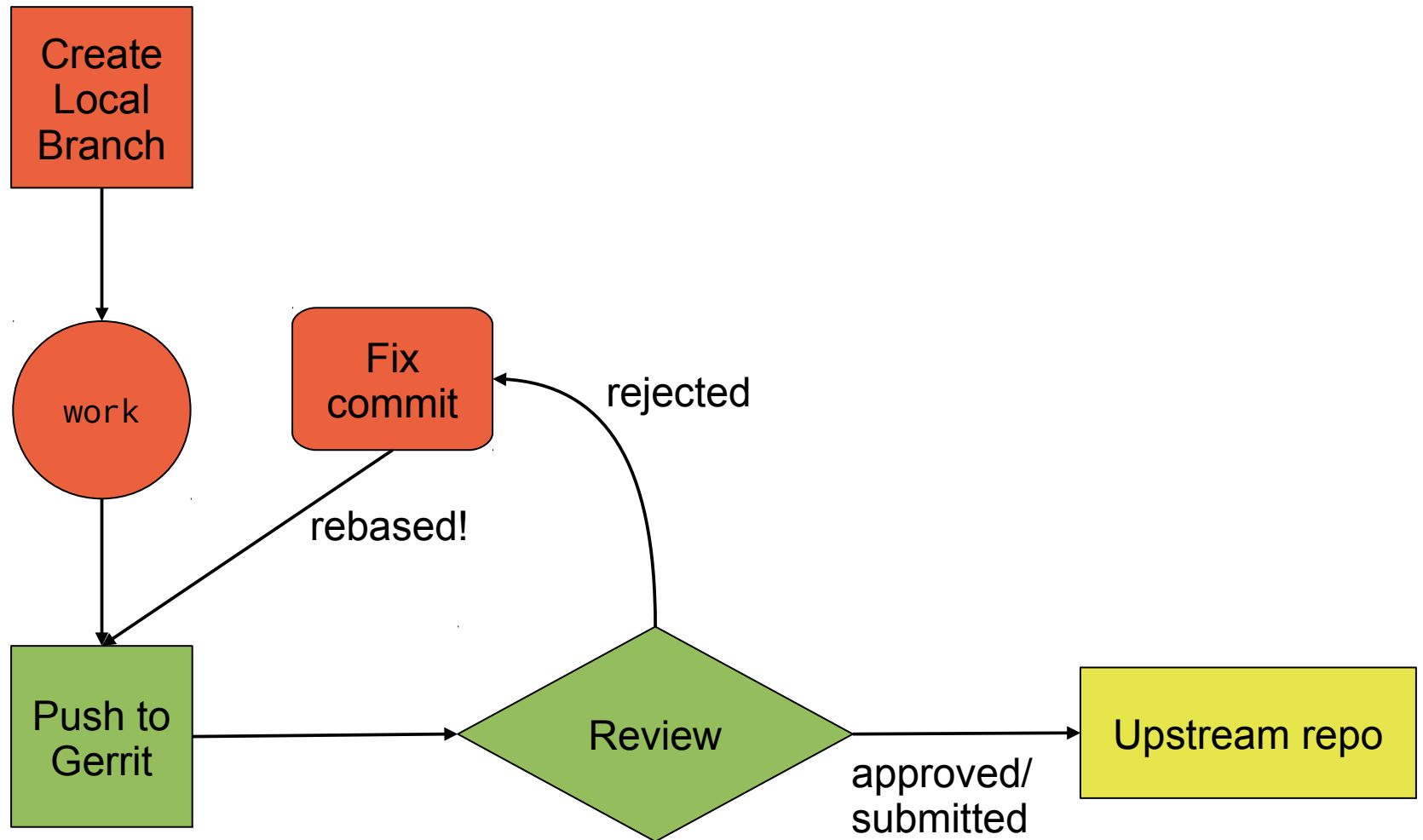
Push
to
Gerrit

`% git push gerrit \`
`HEAD:refs/for/master`



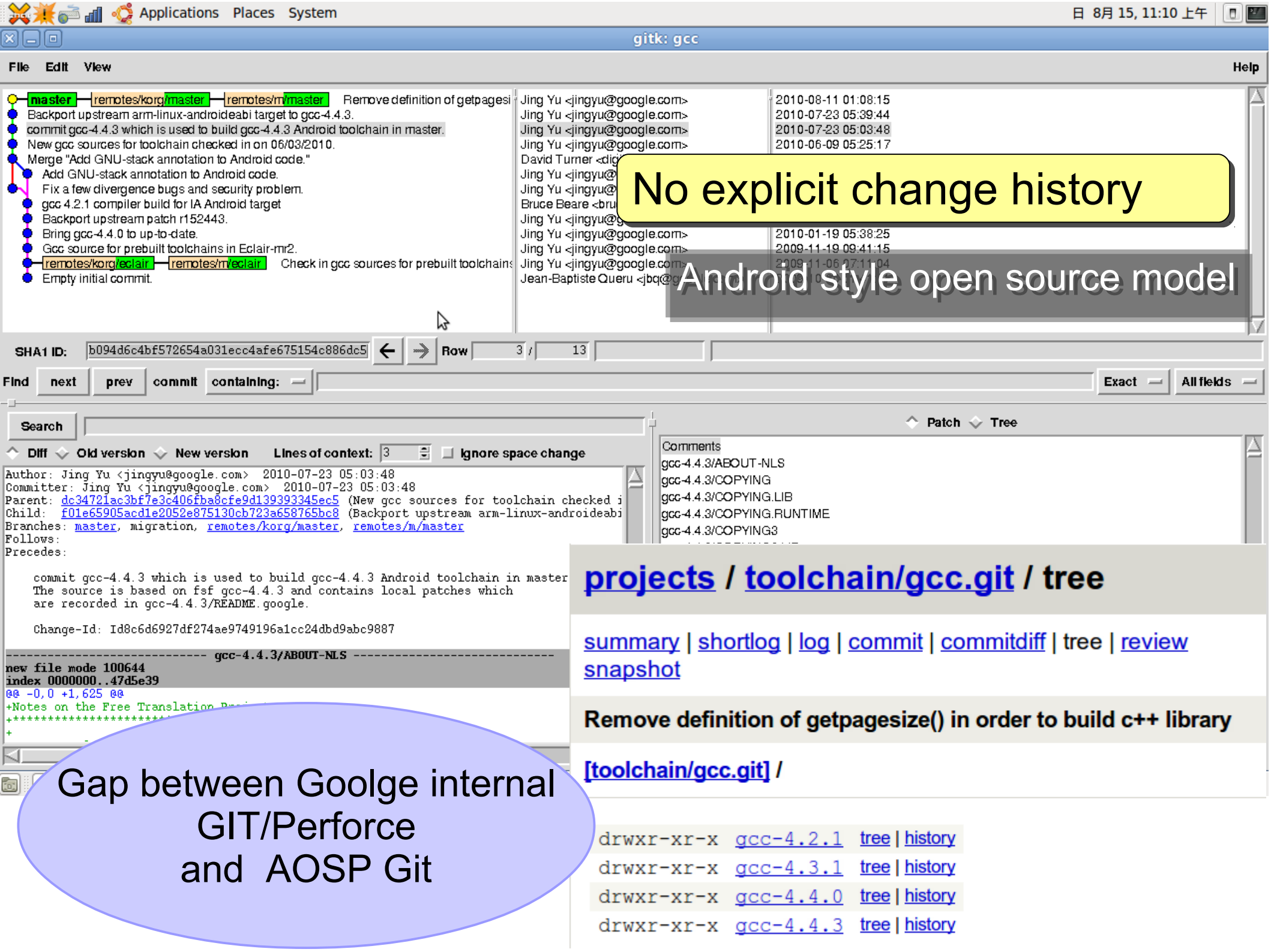
Google provides the great code review tool for AOSP, but...

Flow of AOSP submitted changes



Here “upstream” means AOSP master and Google internal tree.





No explicit change history

Android style open source model

[projects](#) / [toolchain/gcc.git](#) / [tree](#)

[summary](#) | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#) | [review snapshot](#)

Remove definition of getpagesize() in order to build c++ library

[\[toolchain/gcc.git\]](#) /

- drwxr-xr-x [gcc-4.2.1](#) [tree](#) | [history](#)
- drwxr-xr-x [gcc-4.3.1](#) [tree](#) | [history](#)
- drwxr-xr-x [gcc-4.4.0](#) [tree](#) | [history](#)
- drwxr-xr-x [gcc-4.4.3](#) [tree](#) | [history](#)

Gap between Goolge internal
GIT/Perforce
and AOSP Git

After Gingerbread, it gets much clear for toolchain part

```
prebuilt/  
commit 81cce608ab19dcd0aaf7d08d57a4460229e43c45  
Author: Jing Yu <jingyu@google.com>  
Date: Tue Dec 14 10:55:23 2010 -0800
```

Patched toolchain to fix a few gcc and binutils bugs.


Sources to build this toolchain are listed on arm-eabi-4.4.3/SOURCES

```
linux-x86/toolchain/arm-eabi-4.4.3/SOURCES
```

```
build/ synced to  
commit 4cc02faaa7e8828f9458b1828a6f85e7791ae2aa  
Author: Jim Huang <jserv@0xlab.org>  
Date: Fri Aug 20 23:30:37 2010 +0800
```

And rollback the following 3 patches.

```
commit de263c26a7680529baca731c003bc58b68d72511  
Author: Jing Yu <jingyu@google.com>  
Date: Thu Aug 12 15:52
```



Although we can check git log, we
still have no idea why they changed

Observed AOSP Working Model

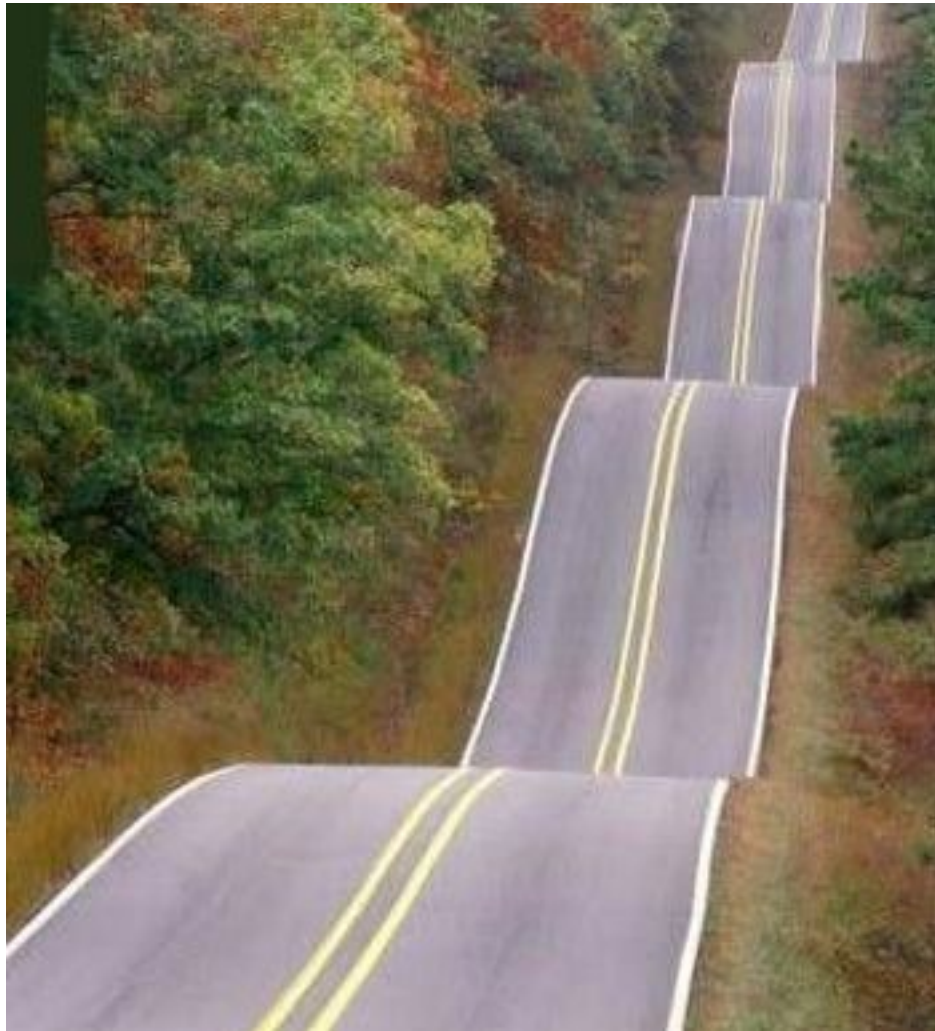
- “master” branch in Android is the bridge between AOSP and Google internal tree. There are many contributions merged from companies, organizations, and individuals. But no efficient code review available for non-existing repositories. And, only few Google engineers do review changes.
- Master branch = the latest AOSP + Partial changes by Google (bug-fixes from internal tree)
 - Not fully verified codebase.
- The best hints are the opinions written by Google engineers inside Gerrit.
 - Send patches if possible



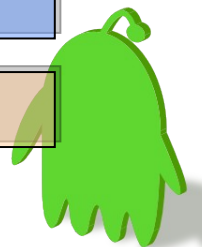
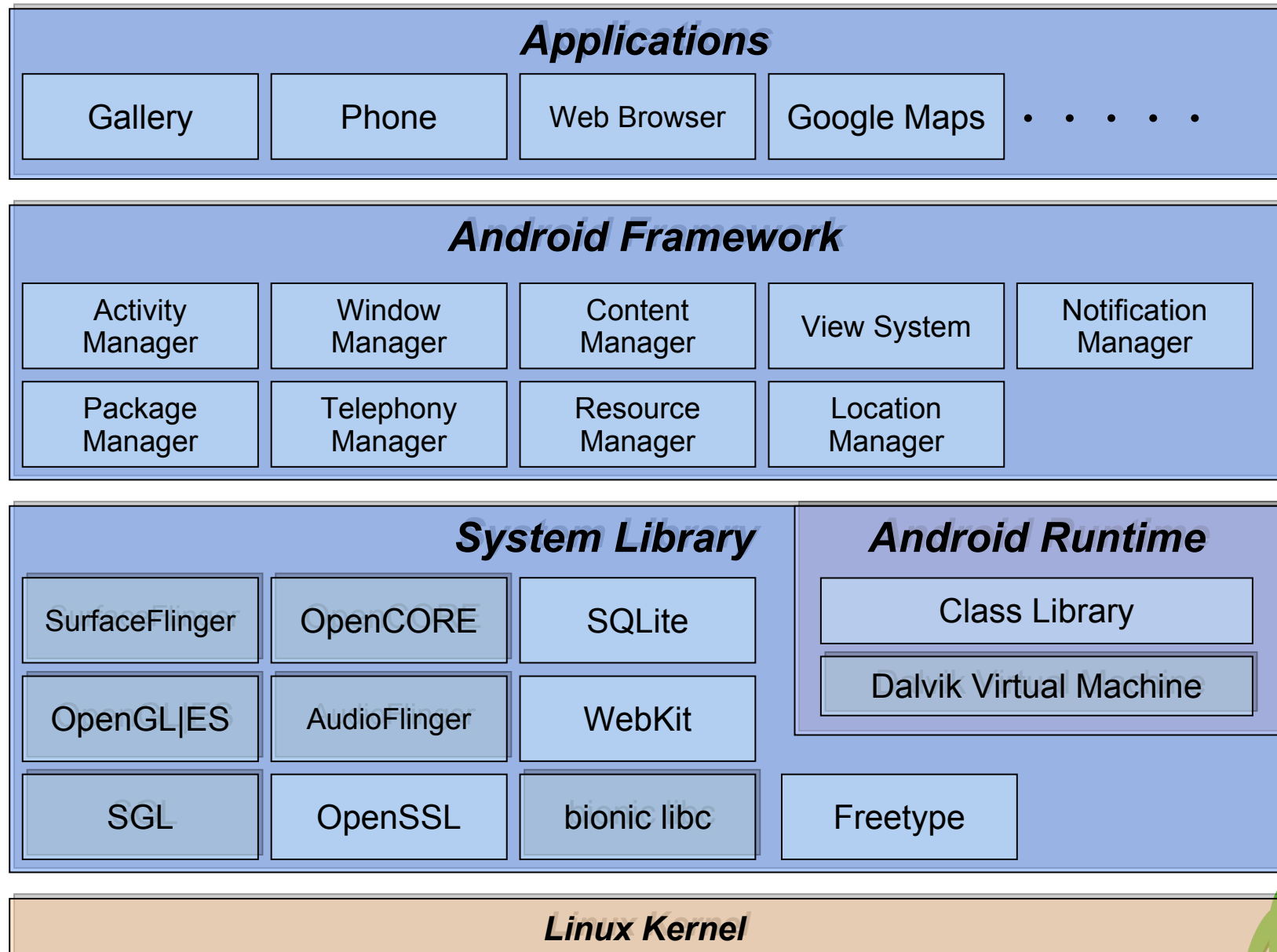
My interpretation of Android:

Hardware is Revolution;
Software is basically
Evolution;

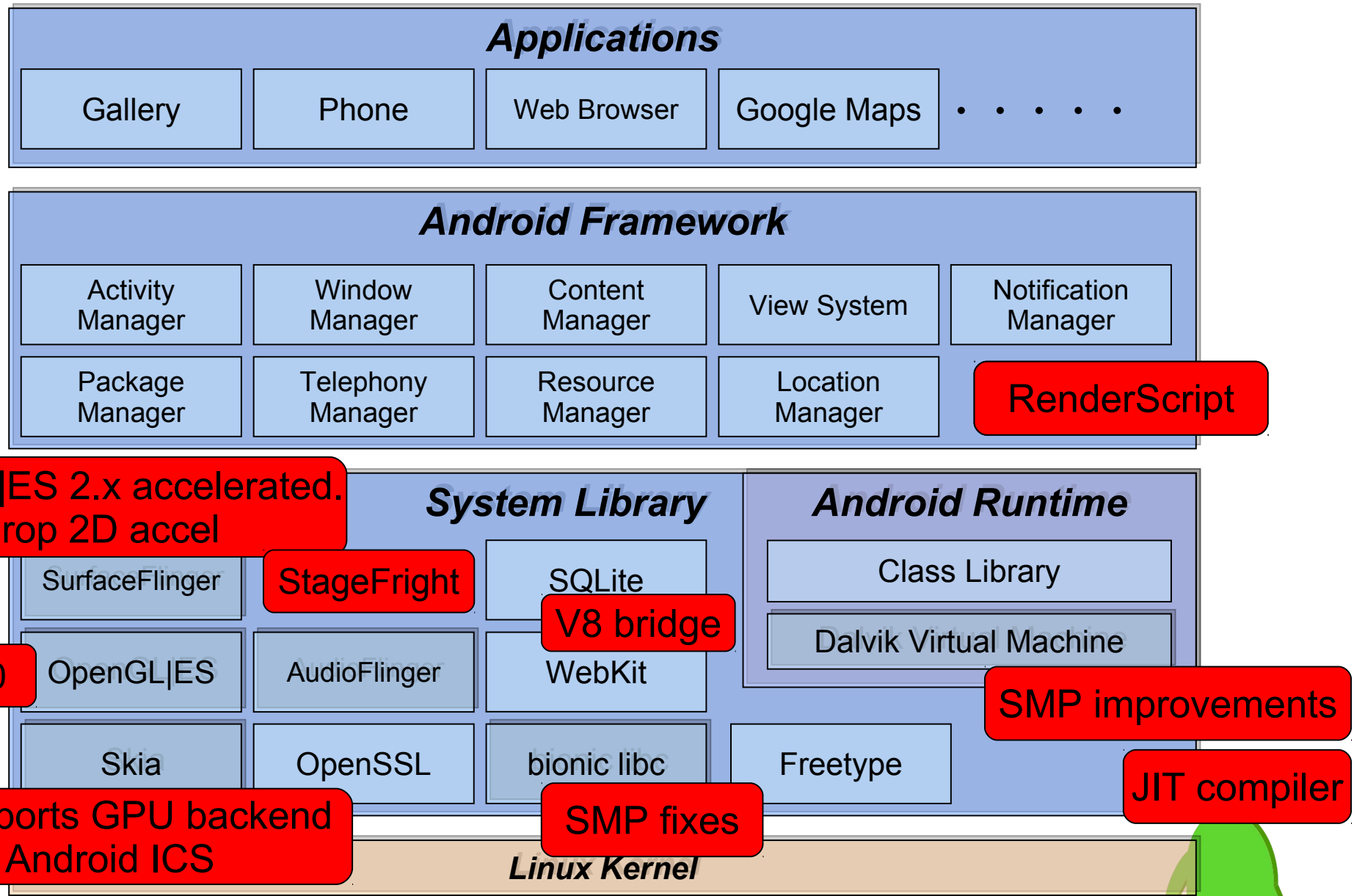
Android is Hardware-driven
Software Revolution



Functional View (Android 1.5)



Functional View (Android 2.3)



The overall design is consistent, but the current model prevents from diverse community contributions.

AOSP statistics for Gingerbread (Dec 2010)

| | | | |
|--------|-------------------|------|----------------|
| ■ 4204 | google.com | ■ 17 | openbossa.org |
| ■ 1354 | android.com | ■ 11 | nxp.com |
| ■ 98 | sonyericsson.com | ■ 11 | linux.org.tw |
| ■ 71 | gmail.com | ■ 10 | ti.com |
| ■ 39 | codeaurora.org | ■ 10 | acer.com.tw |
| ■ 39 | samsung.com | ■ 8 | themaw.net |
| ■ 38 | intel.com | ■ 8 | garmin.com |
| ■ 32 | nokia.com | ■ 7 | snpe.rs |
| ■ 32 | holtmann.org | ■ 7 | motorola.com |
| ■ 29 | 0xlab.org | ■ 7 | mc.pp.se |
| ■ 25 | trusted-logic.com | ■ 7 | googlemail.com |

Because Android
imported BlueZ
as Bluetooth stack

The number are commits since Froyo release.

However, the valuable changes from community such as CyanogenMod are usually absent due to long-time review process.





Methods to Enhance Android Software Stack

when building custom Android distribution and
merging changes from community



Mission in our development:

Improve UX in SoC

UX = User Experience

SoC = Integrated Computing Anywhere



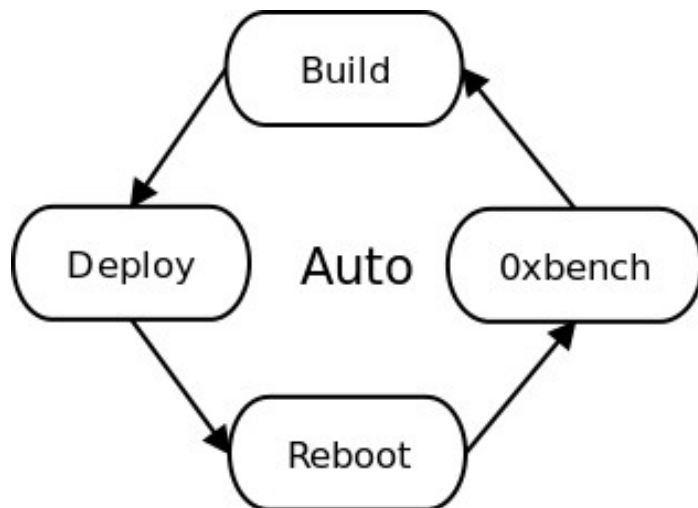
Quality in custom Android Distribution

- 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.



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.



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`

The screenshot shows the Launch Control web interface. At the top, it says "Launch Control" with a version of 0.3c10. Below this, there are navigation links: Home, Reports, Bundle Streams, and XML-RPC API. The "Reports" link is highlighted with a red "new" badge. The breadcrumb trail shows the user is in the "Bundle Streams" section for an "anonymous/android-beagle01-basic" device. Below this is a table with columns: Uploaded On, Analyzed, Test, Run, Pass, Fail, Skip, and Unknown. The table contains four rows of test results, all showing a "Pass" status.

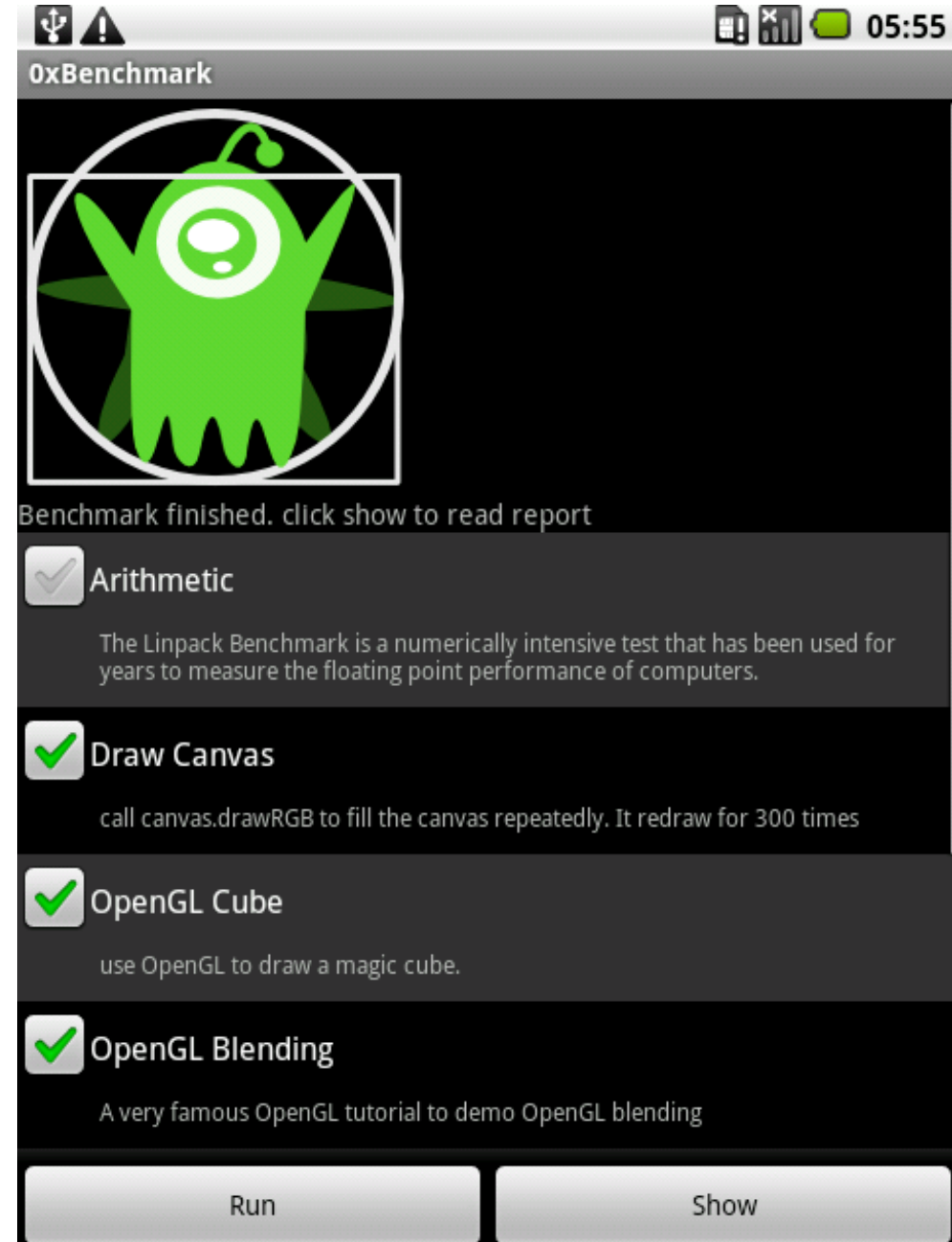
| Uploaded On most recent first | Analyzed | Test | Run | Pass | Fail | Skip | Unknown |
|----------------------------------|----------------------|-------|---|------|------|------|---------|
| April 27, 2011 5:23 p.m. | 1 day, 16 hours ago | basic | Test run 1b8ff0f0-70f3-11e0-b5f6-0026c747dbf8 | 3 | 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 |

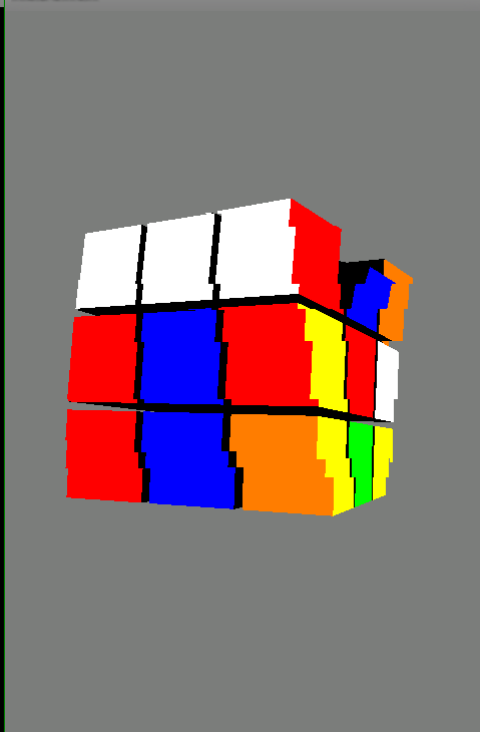
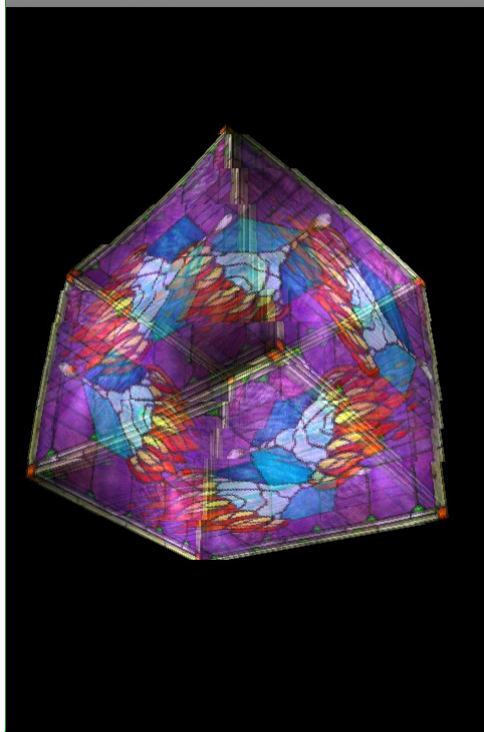


Oxbench: comprehensive open source benchmark suite for Android

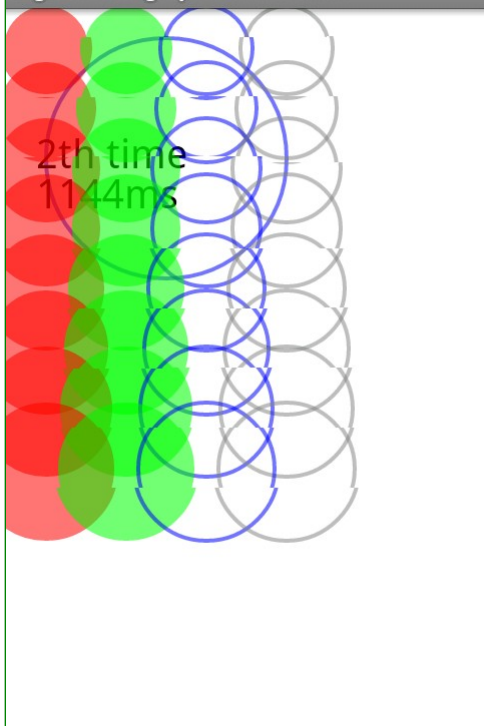
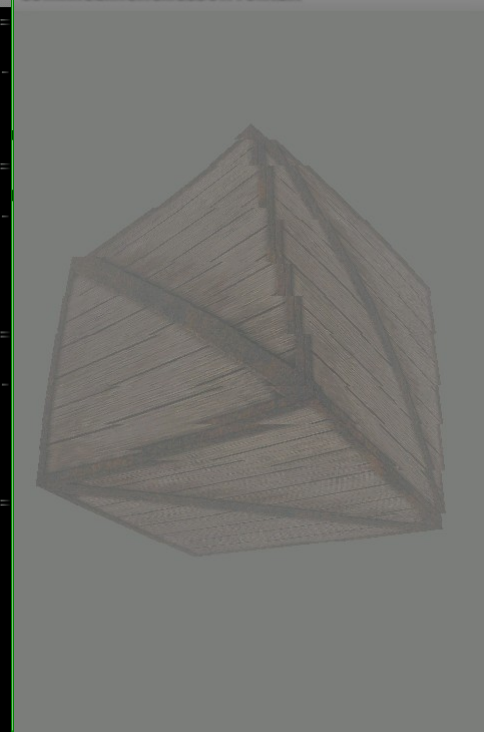
- A set of system utilities for Android to perform comprehensive system benchmarking
 - Dalvik VM performance
 - OpenGL|ES performance
 - Android Graphics framework performance
 - I/O performance
 - JavaScript engine performance
 - Connectivity performance
 - Micro-benchmark: standard C library, system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

<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
=====
```

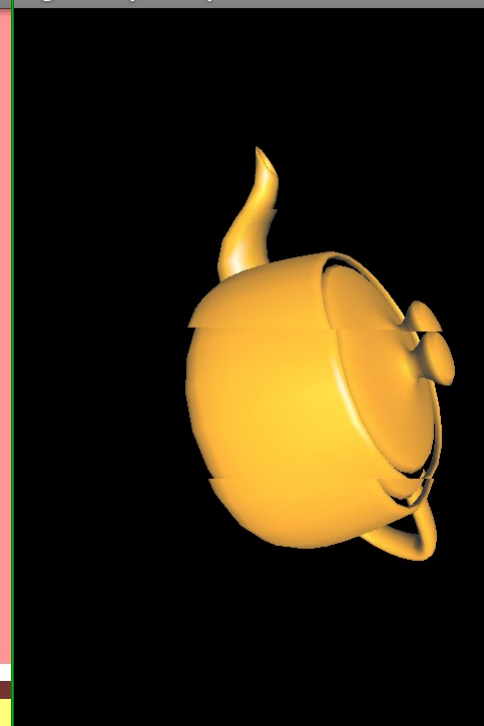


```
Stretching memory:
  binary tree of depth 16
*Total memory:3612640 bytes
*Free memory:738576 bytes

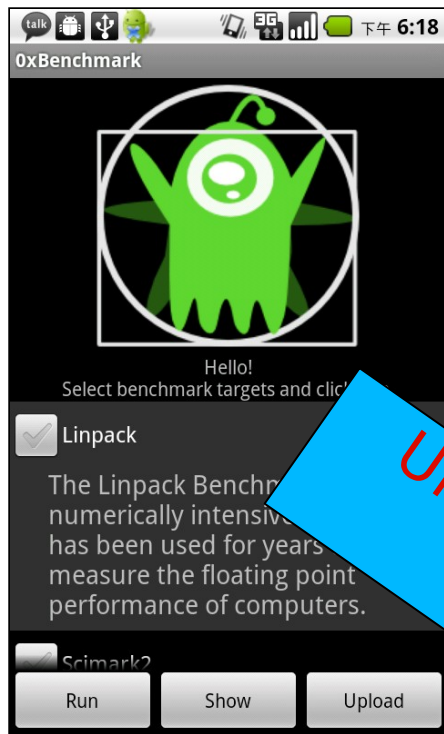
Creating:
  long-lived binary tree of depth 16
  long-lived array of 125000 double
*Total memory:8003552 bytes
*Free memory:3327960 bytes

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
62 bytes
- Top down: 957msecs
- Bottom up: 808msecs
*Total memory:8003552 bytes
*Free memory:2901808 bytes

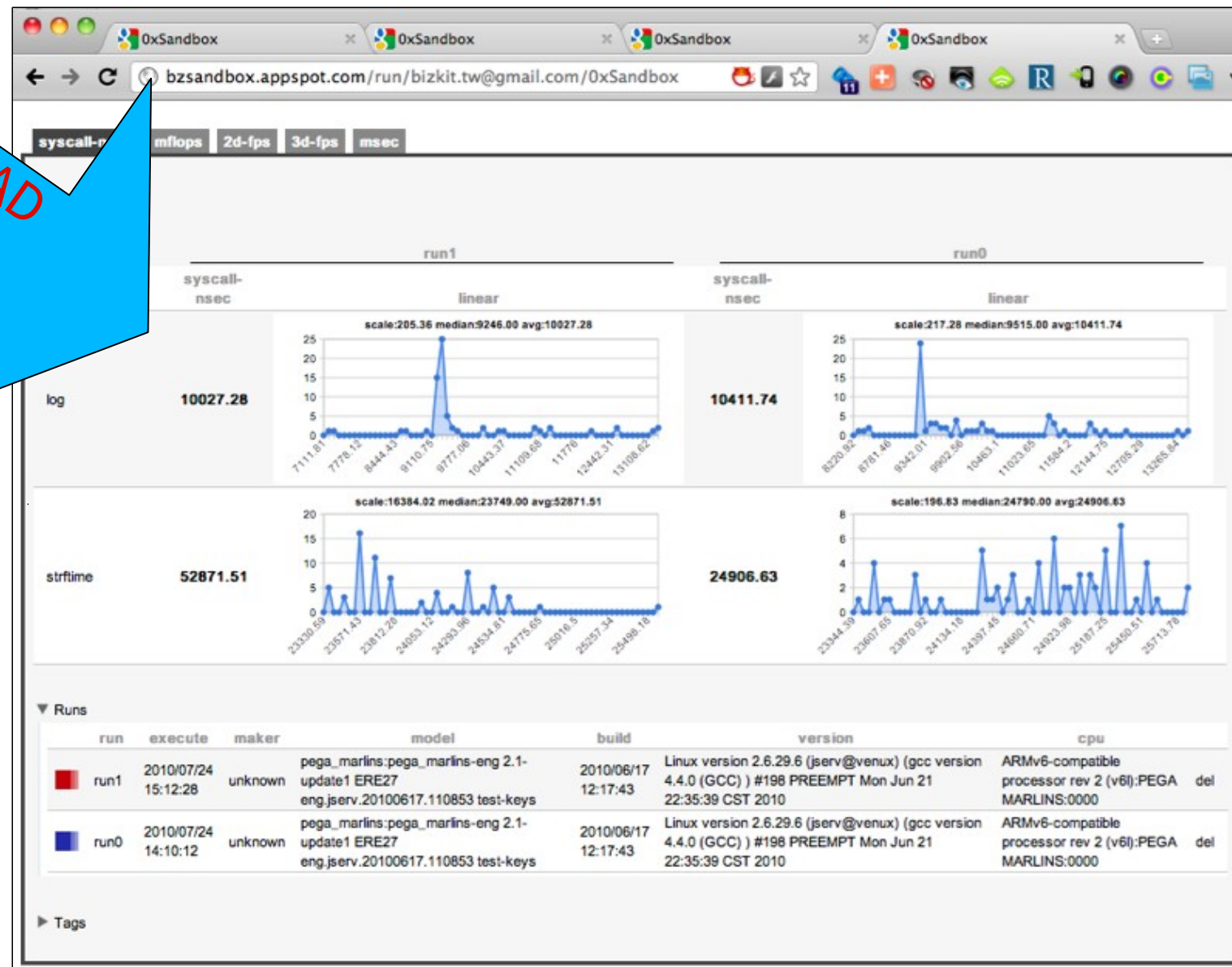
Completed in 10413ms.
```



Collect and Analyze results on server-side



UPLOAD



Android Functional Testing

(1) stress test

(2) Automated test



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, keyboard events, etc., then run automatically.



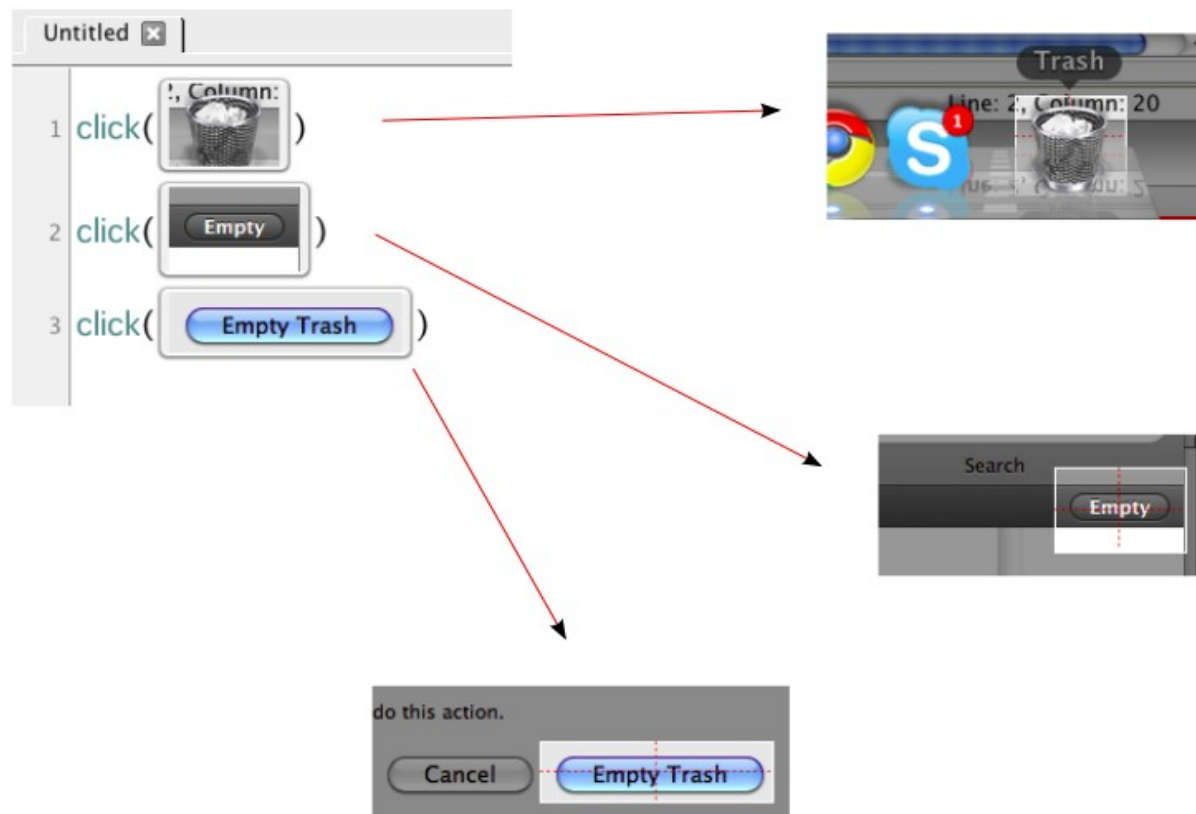
ASTER: Automated Test

- 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



Functional Test



Desktop: Sikuli







Sikuli - testDialer


File View

G1.test.sikuli x testDialer.sikuli x

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

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

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

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

Message Test Trace

```
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
```

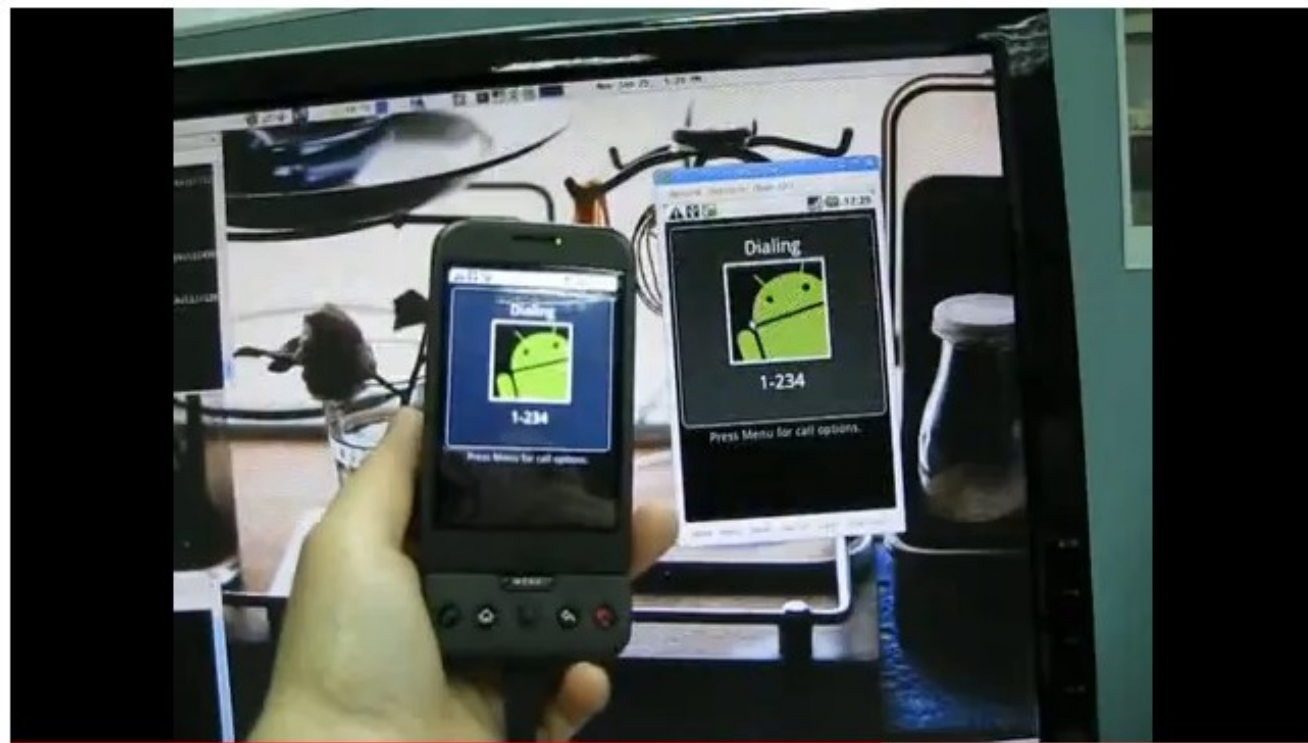
Line: 97, Column: 19

Dial out a phone call in Android by Sikuli

icedventilatte

10 videos

Subscribe



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/androidscreencast/>

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



Aster





Bring Enhancements back to
Community










What do we deliver to community?

- Patches merged in AOSP, CyanogenMod, and Android-x86
- Implement 100% open source OpenGL|ES adaptation based on Mesa/3D into Android
 - The world-first, important to Android-x86
- Performance: ARM specific optimizations, graphics enhancement
- Features: Bluetooth HID (keyboard/mouse), external modem, 3D effects, customized Launcher







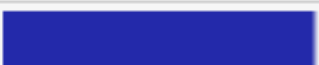


← run_Nexus_S:GRJ22_2011/07/13-22:48:03UTC

| 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 |  |








Arithmetic on Nexus S

Tune Dalvik VM performance (armv7)

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 |  |








← 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 |  |

2D on Nexus S

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 |  |

Benchmark: 2D (arm1 1-custom)

mflps

2d-fps

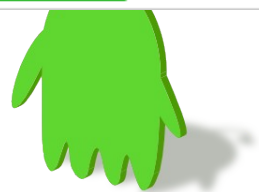
3d-fps

msec

Options

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

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



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

mflops 2d-fps 3d-fps msec

► Options

| benchmark | advanced-performance2 | | advanced-performance | | startpoint | |
|----------------|-----------------------|-------------|----------------------|-------------|------------|-------------|
| | 3d-fps | linear | 3d-fps | linear | 3d-fps | linear |
| OpenGLCube | 27.65 | <div></div> | 26.36 | <div></div> | 11.77 | <div></div> |
| OpenGLBlending | 15.21 | <div></div> | 15.06 | <div></div> | 8.78 | <div></div> |
| OpenGLFog | 14.03 | <div></div> | 13.86 | <div></div> | 8.36 | <div></div> |
| FlyingTeapot | 12.30 | <div></div> | 11.26 | <div></div> | 7.38 | <div></div> |

| benchmark → | ← M3 + Linaro Toolchain | | ← M3 | |
|----------------|-------------------------|-------------|--------|-------------|
| | 3d-fps | linear | 3d-fps | linear |
| OpenGLCube | 29.06 | <div></div> | 29.04 | <div></div> |
| OpenGLBlending | 20.07 | <div></div> | 19.94 | <div></div> |
| OpenGLFog | 18.63 | <div></div> | 18.95 | <div></div> |
| FlyingTeapot | 17.49 | <div></div> | 17.04 | <div></div> |

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



- Android C/C++ library
- 0xlab's Optimizations (merged in AOSP)
 - Memory operations: Use ARMv6 unaligned access to optimize usual cases
 - reused by Dalvik to accelerate primitive byte order
 - Endian/Data Type conversion: Use ARMv6 fast endian primitives. Useful for TCP/IP (big endian / little endian converting)
 - Various ARM optimized string operations
 - memcpy, strcmp, strcpy, memset



Dynamic Linker Optimizations

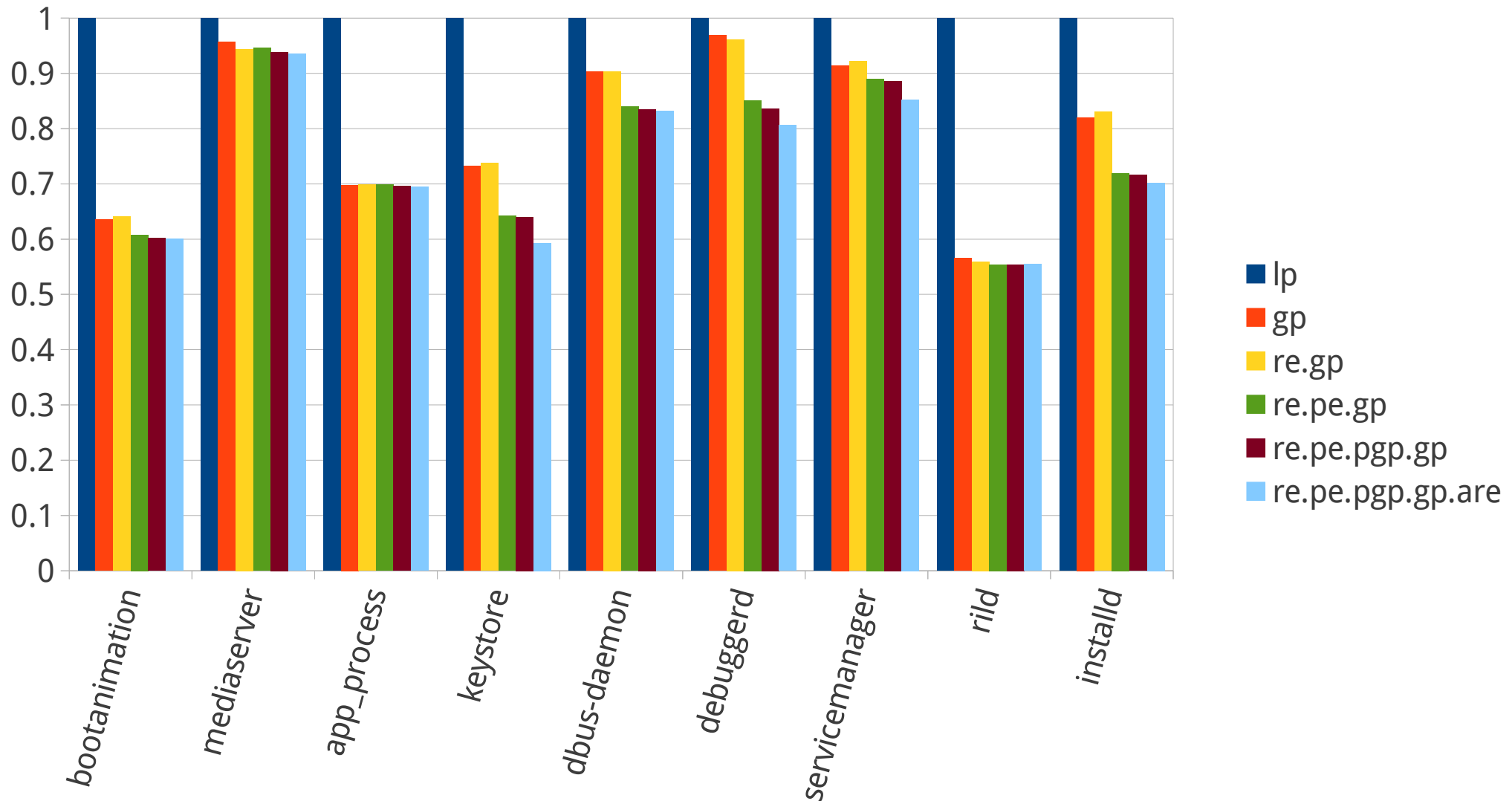


Why and How?

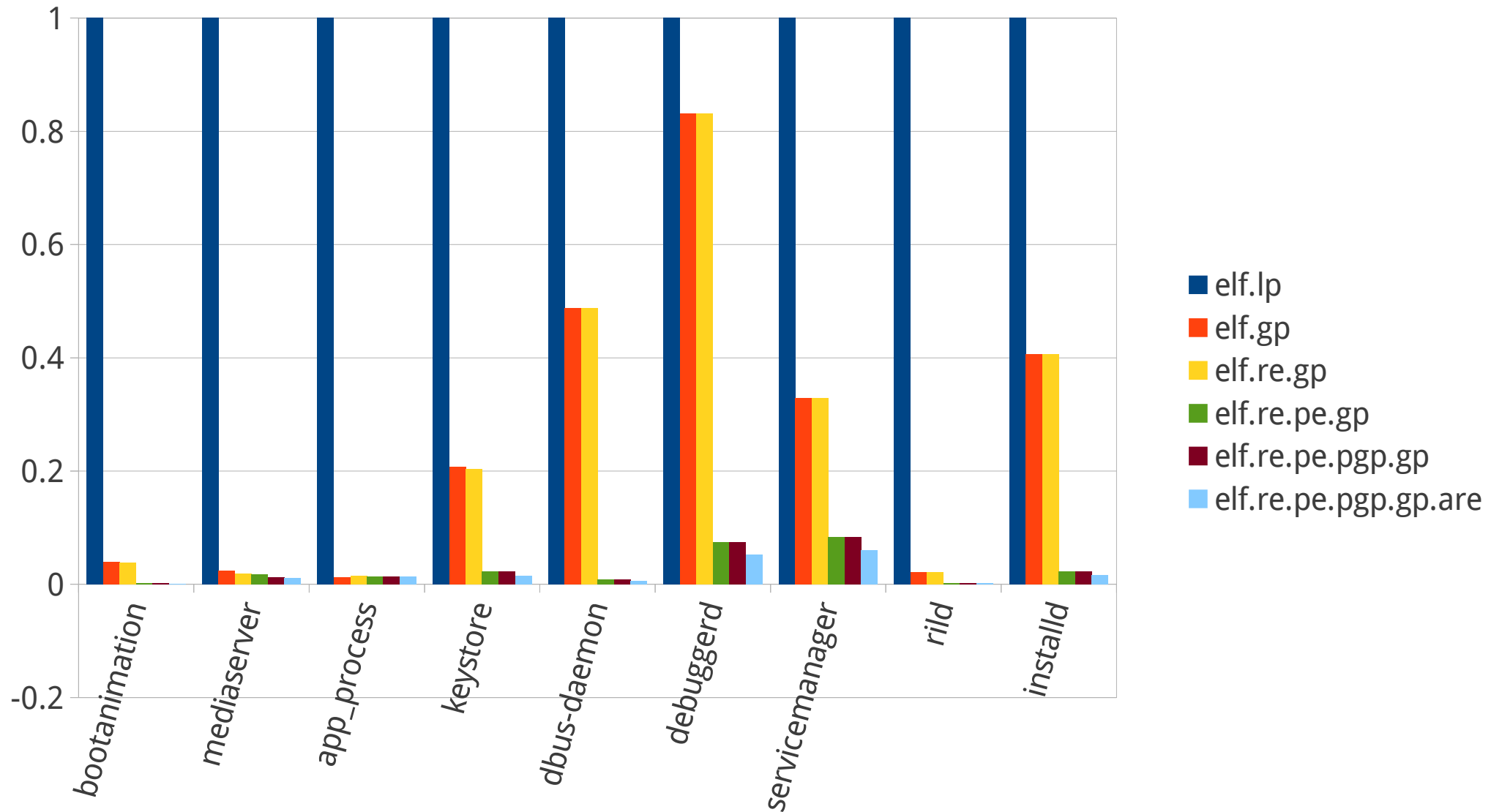
- The major reason to optimize dynamic linker is to speed up application startup time.
- Approaches:
 - Implement GNU style hash support for bionic linker
 - Prelinker improvements: incremental global prelinking
 - reduce the number of ELF symbol lookup aggressively
- Changed parts
 - apriori, soslim,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

libc.so
printf

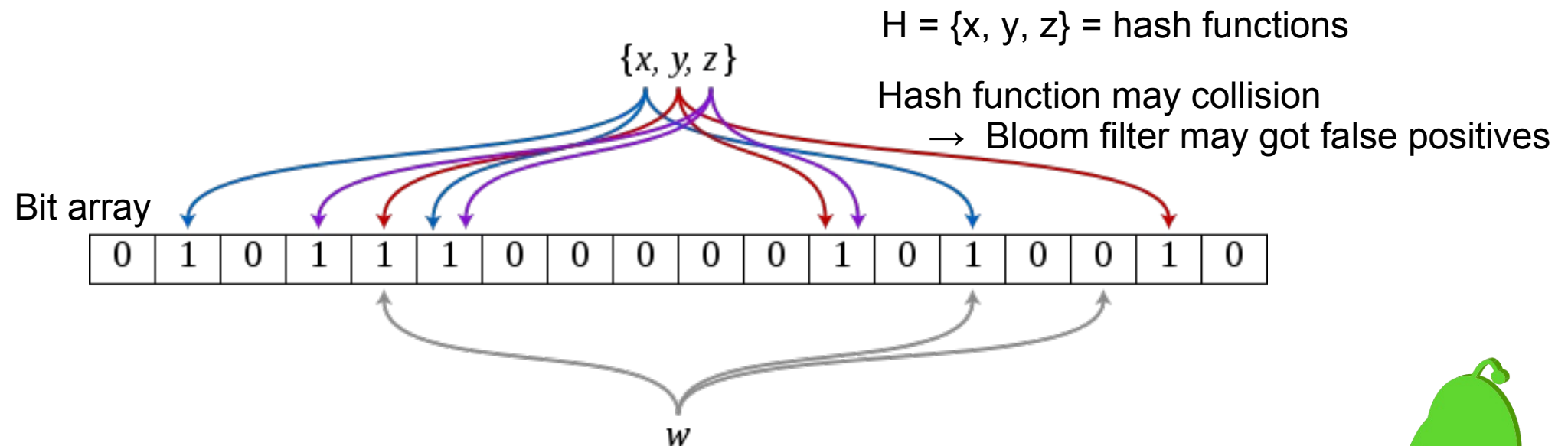
```
void foo (){  
    printf("fooooo");  
    bar();  
}
```

libfoo.so
foo
bar

| libfoo.so | |
|-------------|---------|
| DT_GNU_HASH | DT_HASH |
| foo | foo |
| bar | bar |
| | printf |



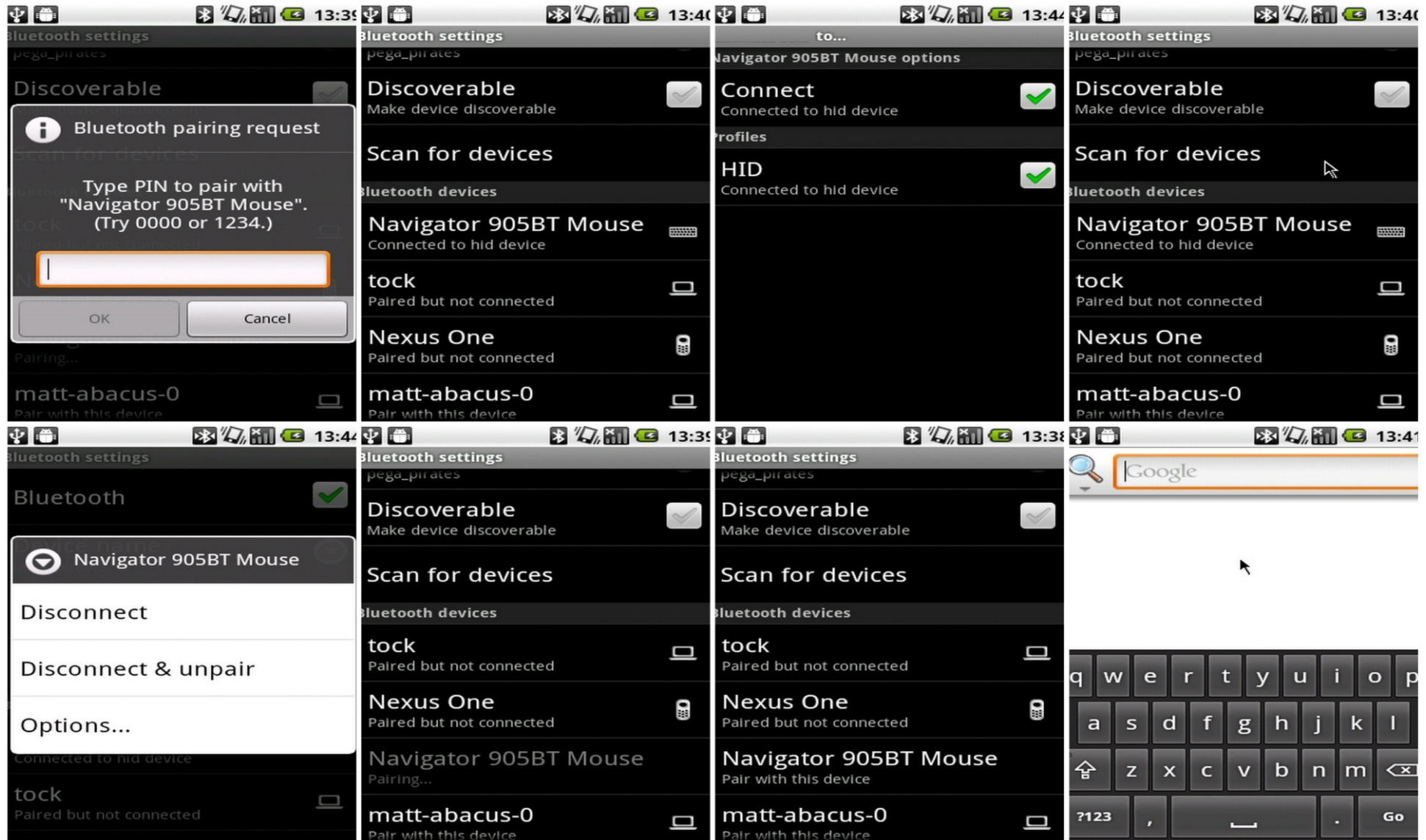
| | Symbols in ELF | lookup# | fail# | gnu hash | filtered by bloom |
|-----------|-------------------|---------|--------|----------|-------------------|
| gnu.gp | 3758 | 23702 | 19950 | 23310 | 18234(78%) |
| gnu.gp.re | 3758 | 20544 | 16792 | 19604 | 14752(75%) |
| gnu.lp | 61750 | 460996 | 399252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



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



Bluetooth HID (keyboard/mouse)

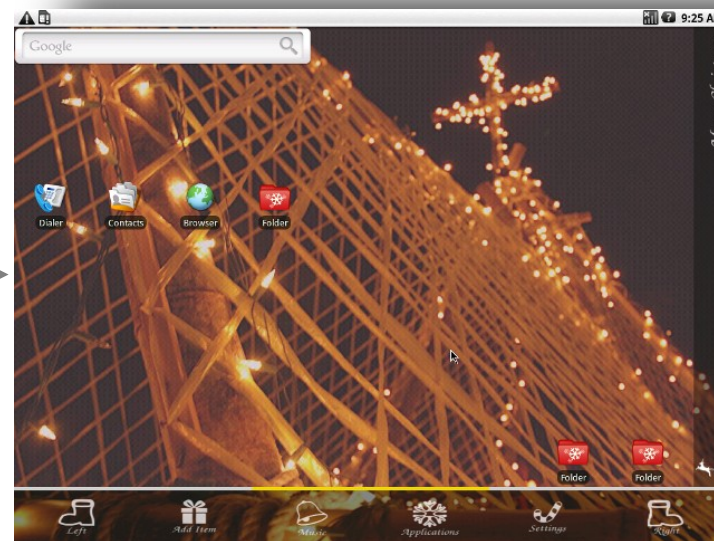
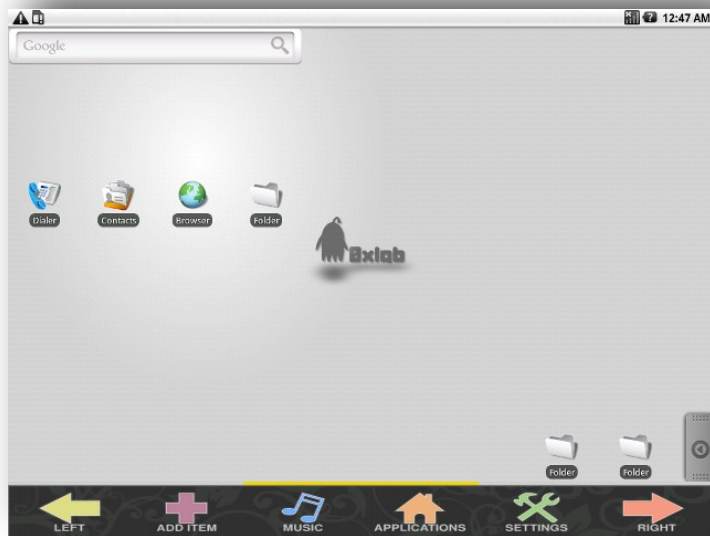


CyanogenMod merged the feature since version 6.x. (2010q2)
Android 4.0 finally provides its own implementation though. (2011q4)



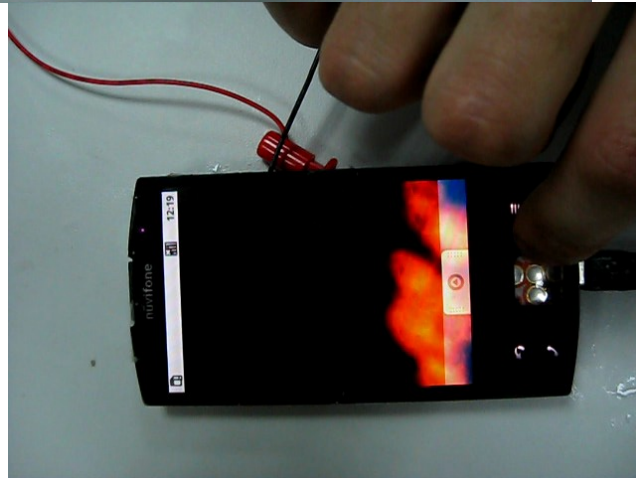
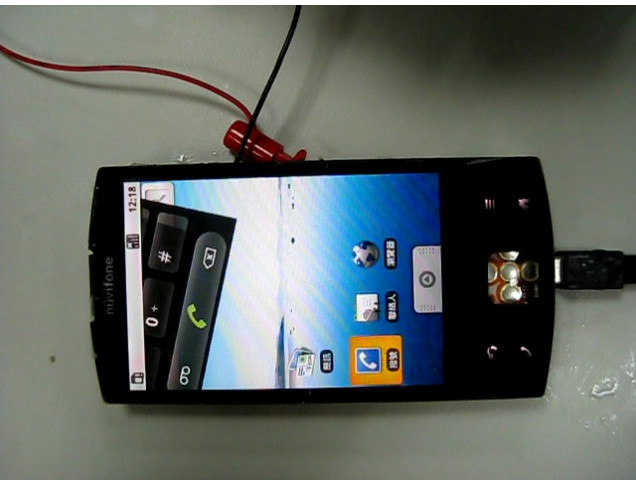
UI customizations

- Provide several UI/Launcher combination for small and large screen devices.
 - Sizing from HVGA, VGA, SVGA (Phone), to 720p/1080p (TV)
- Either modified Android Launcher or new replacement
- Licensed under Apache Software License



Products with Advanced 3D UI

- 3D effects and the ARM optimizations are enabled in Qualcomm platforms w/ and w/o GPU accelerations
- Demo video: <http://www.youtube.com/watch?v=kNyVXIEG3gg>



Android Boot Time Optimizations



Write Tiny Boot loader to Speed up

- Qi Boot-loader
 - Only **one stage** boot-loader
 - Small footprint **~30K**
 - Currently support
 - iMX31
 - Samsung 24xx
 - Beagleboard
 - KISS concept
 - Boot device and load kernel

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



Optimized ARM Hibernation

- Based on existing technologies thus requires little modification to userspace
 - 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>



Further Boot Time Optimizations

- Based on existing technologies thus requires little
- 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





Go Upstream!

Patches are dead, long live the Android

Let's think about the collaboration with AOSP,



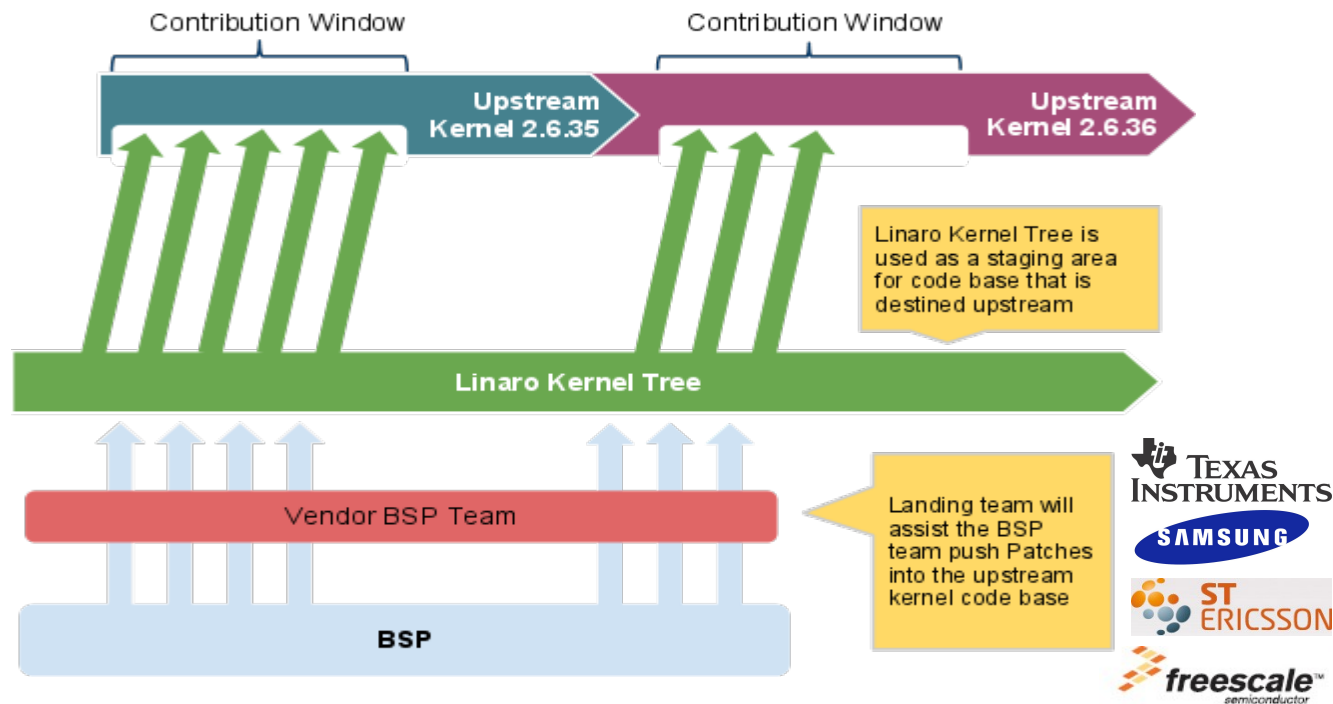
The practice way for upstreaming

- “Buffer” is required.
 - CyanogenMod
 - Linaro
- You have to jump over the gap (software revolution by Google).
- CyanogenMod is proven as the great example to work with AOSP in some way.
 - Large and active user base
 - Developers can submit verified patches to AOSP for reviewing
- Acceptable fields
 - System libraries
 - Application / framework bugs



Working with Linaro

- Linaro kernel becomes members most advanced
- Vendor BSP and Linaro landing team engineers work together
- Get latest software on latest SoCs with upstream support



"merge window" role



Conclusion

- Android is indeed a platform with source code available instead of a typical open source project.
- Collaboration is important especially to Android variants. You have to solve problems with the help from community members.
- Leverage the power of community to make your software validated well.
- Before tracking large scale of Android source code, it would be better to utilize the tools.
- Be open minded



Resources

- 0xdroid Roadmap:
<http://code.google.com/p/0xdroid/wiki/Roadmap>
- Source repository: <http://gitorious.org/+0xlab>
- Wiki: <http://code.google.com/p/0xdroid/w/list>
- Demo videos: <http://www.youtube.com/0xlab>
- Mailing-list:
 - General discussion:
<http://groups.google.com/group/0xlab-discuss>
 - Technical / Development:
<http://groups.google.com/group/0xlab-devel>
- IRC channel (FreeNode): #0xlab





<http://0xlab.org>