Guides To Analyzing WebKit Performance

– Looking at the internals –

Holger Freyther

Developer, WebKit Project

Jim Huang (黃敬群) <jserv@0xlab.org>

Developer & Co-founder, 0xlab

April 28, 2011 / Android System Development Forum

Rights to copy

Corrections, suggestions, contributions and translations are welcome!

Latest update:April 28, 2011

The Goal of This Talk(1)

- Optimize WebKit for the Content?
- Optimize the Content for WebKit?

The Goal of This Talk(2)

- For optimizing content, see the remote inspector work
  - Another example: Opera Mobile Accelerator

    http://www.opera.com/press/releases/2004/06/09/
- This talk will be about *approaching* WebKit

The Goal of This Talk(3)

- Take the fear from working on a big project.
- Show ways to approach the codebase.
- Establish "do not guess but **measure**" attitude.

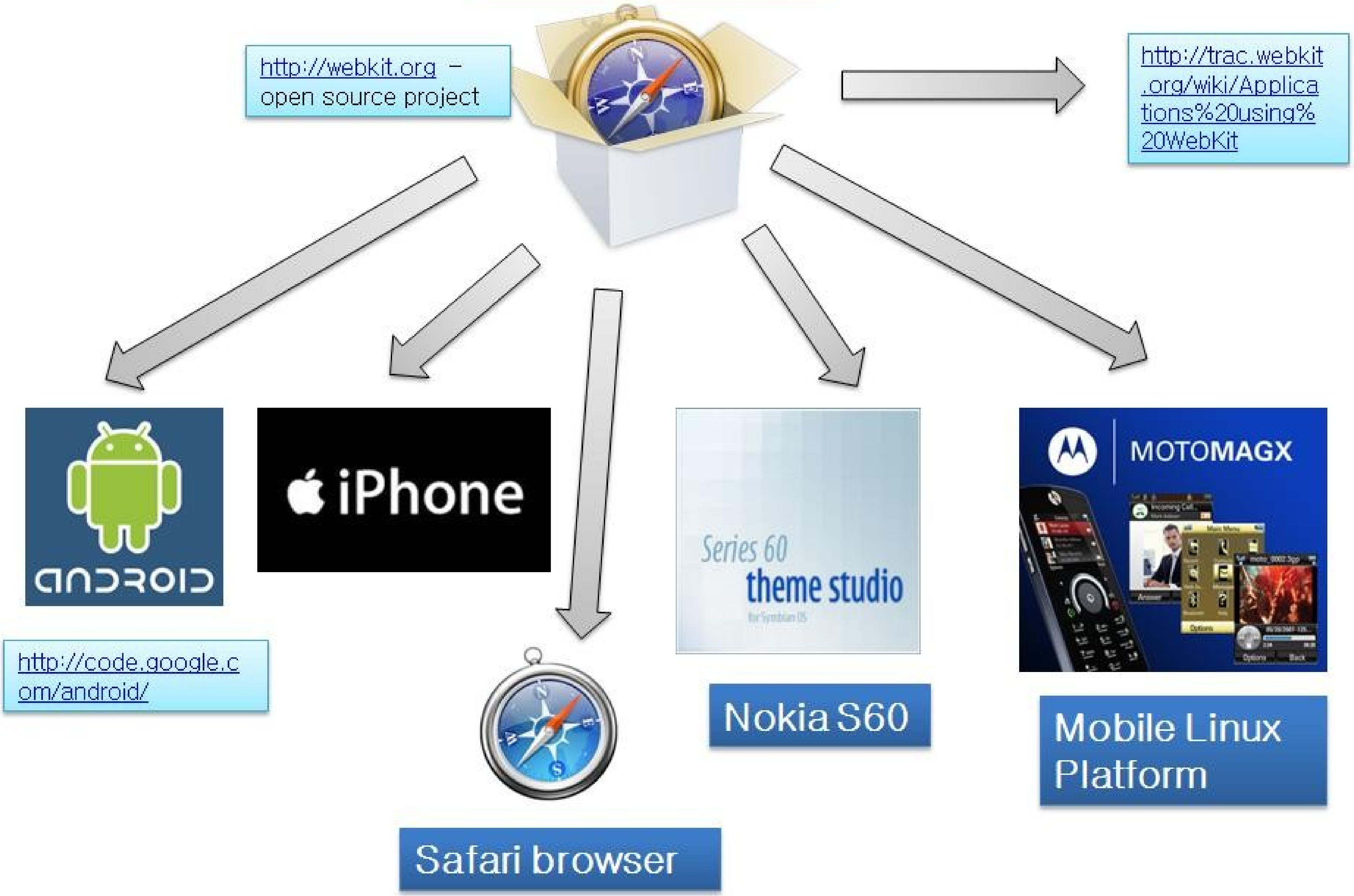
Agenda

(1) What is WebKit (project)?

(2) Android & WebKit

(3) How to prepare Android to measure?

WebKit Engine



http://webkit.org – open source project

http://trac.webkit.org/wiki/Applications%20using%20WebKit

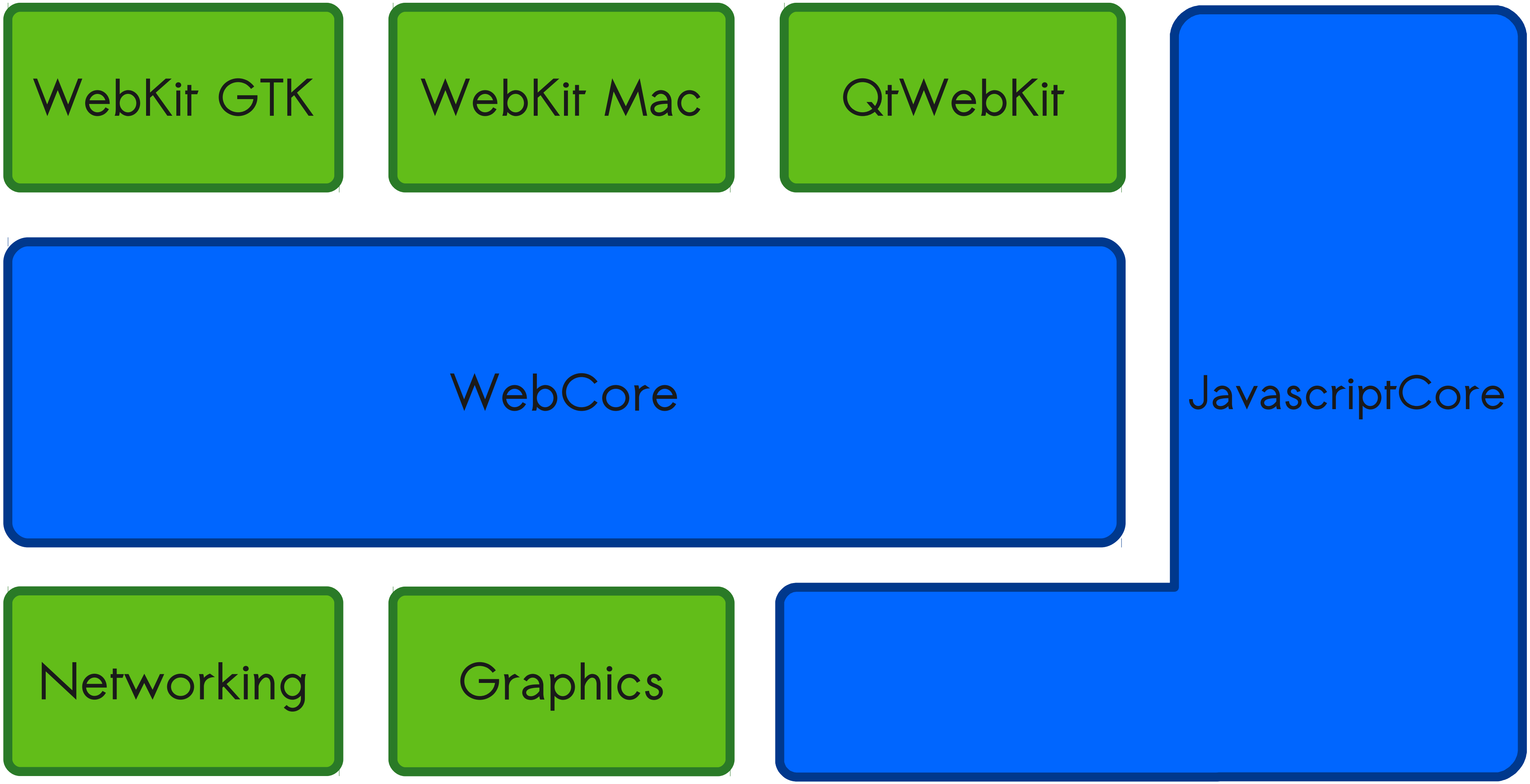
http://code.google.com/android/

Safari browser

Nokia S60

Mobile Linux Platform

WebKit Architecture



WebKit GTK

WebKit Mac

QtWebKit

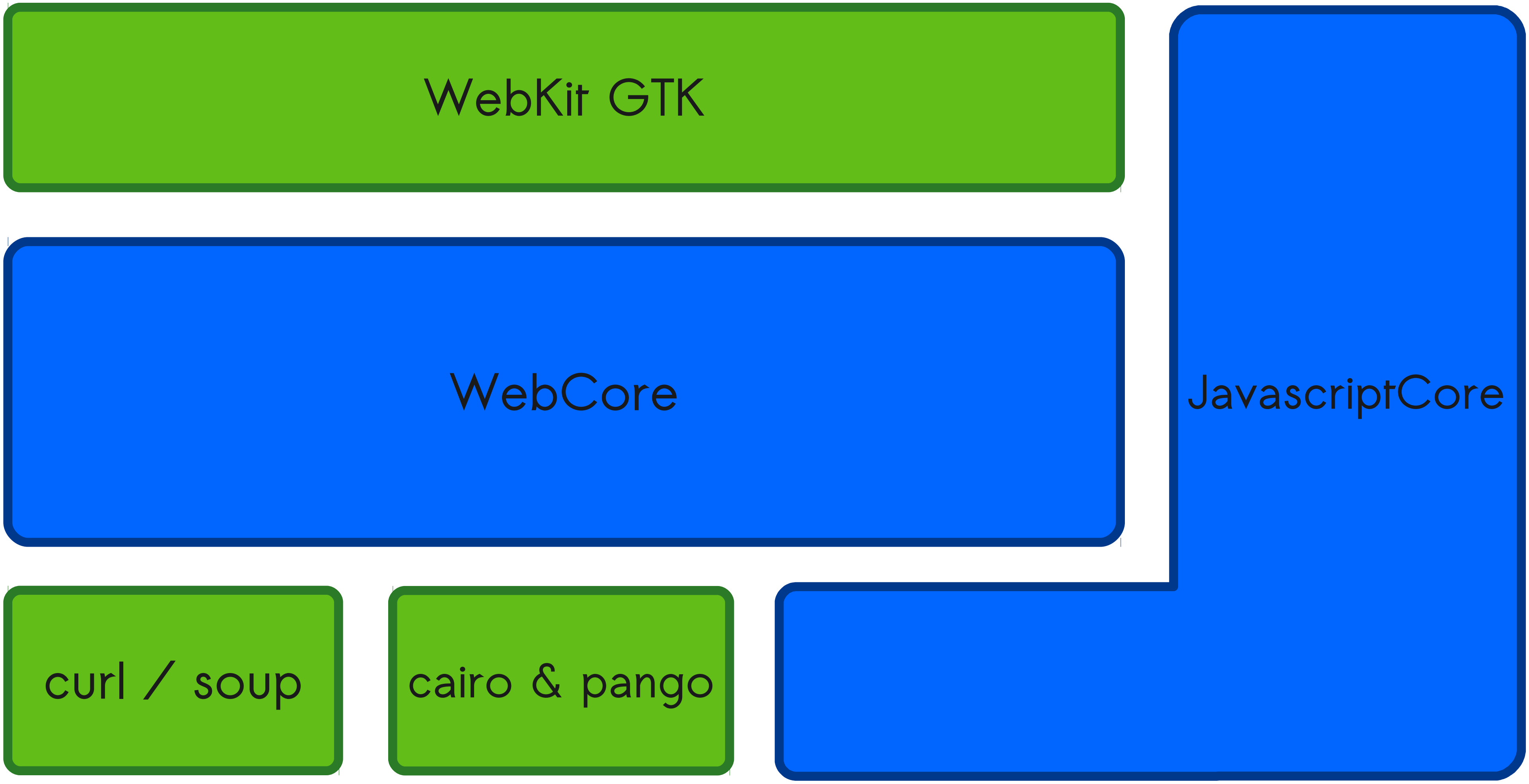
WebCore

JavascriptCore

Networking

Graphics

(Flexible) WebKit Implementations



WebKit GTK

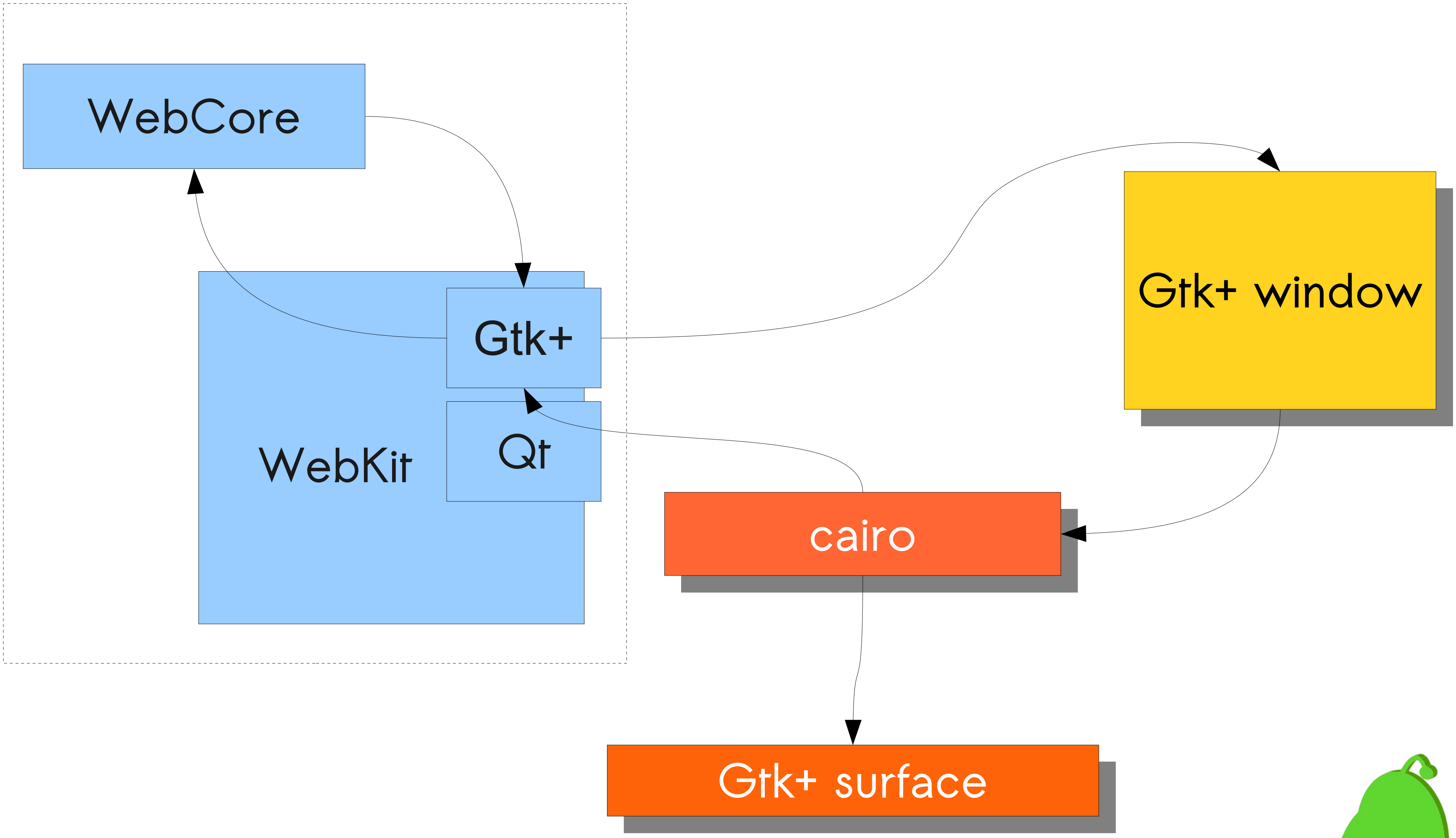
WebCore

JavascriptCore

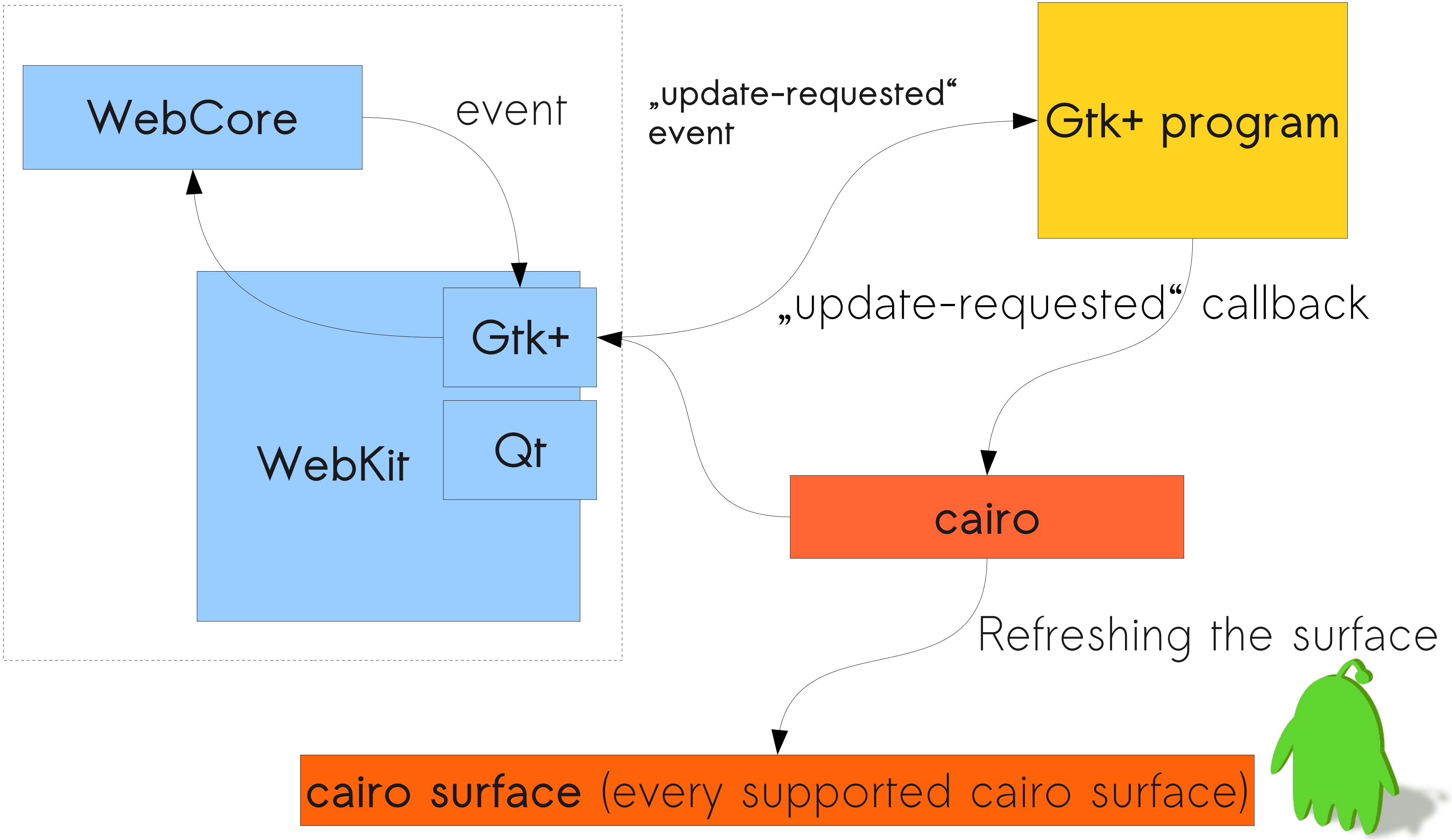
curl / soup

cairo & pango

WebKit drawn to Gtk+



WebKit drawn to Gtk+ applications



What Is the WebKit Project?

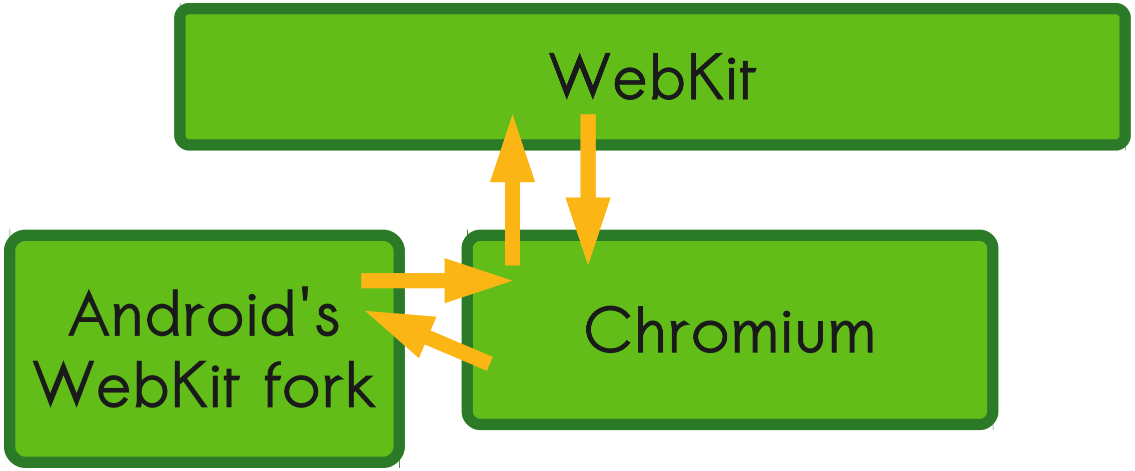
- ~80 Reviewers
- ~110 Committers
- 8 ports in the tree
- Apple and Google (Chromium) are major contributors
- Many commits per day

How Does WebKit Work?

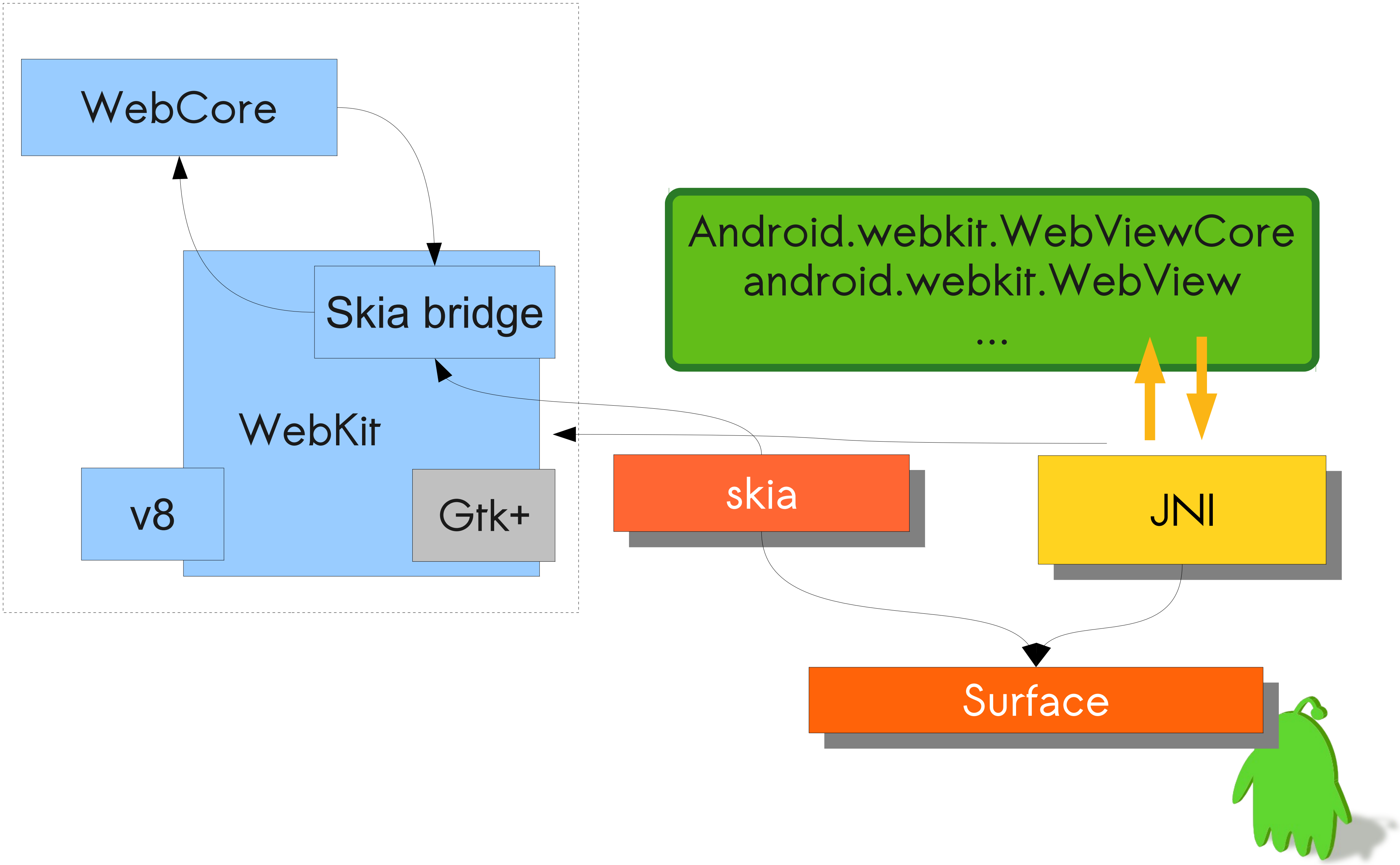
- Check http://webkit.org/projects/goals.html
  - Goals vs. Non-Goals
- Content Engine, Security, Performance and more
- Every change needs review, no performance regression allowed
- But performance tests are private due to copyright laws

Android & WebKit

- Android is not involved with the WebKit project
  - Android style open source model
- Android is using Chromium as upstream
- Android does not include the data for Quality Assurance (tests)
- Who is fixing known security issues in the Android code?

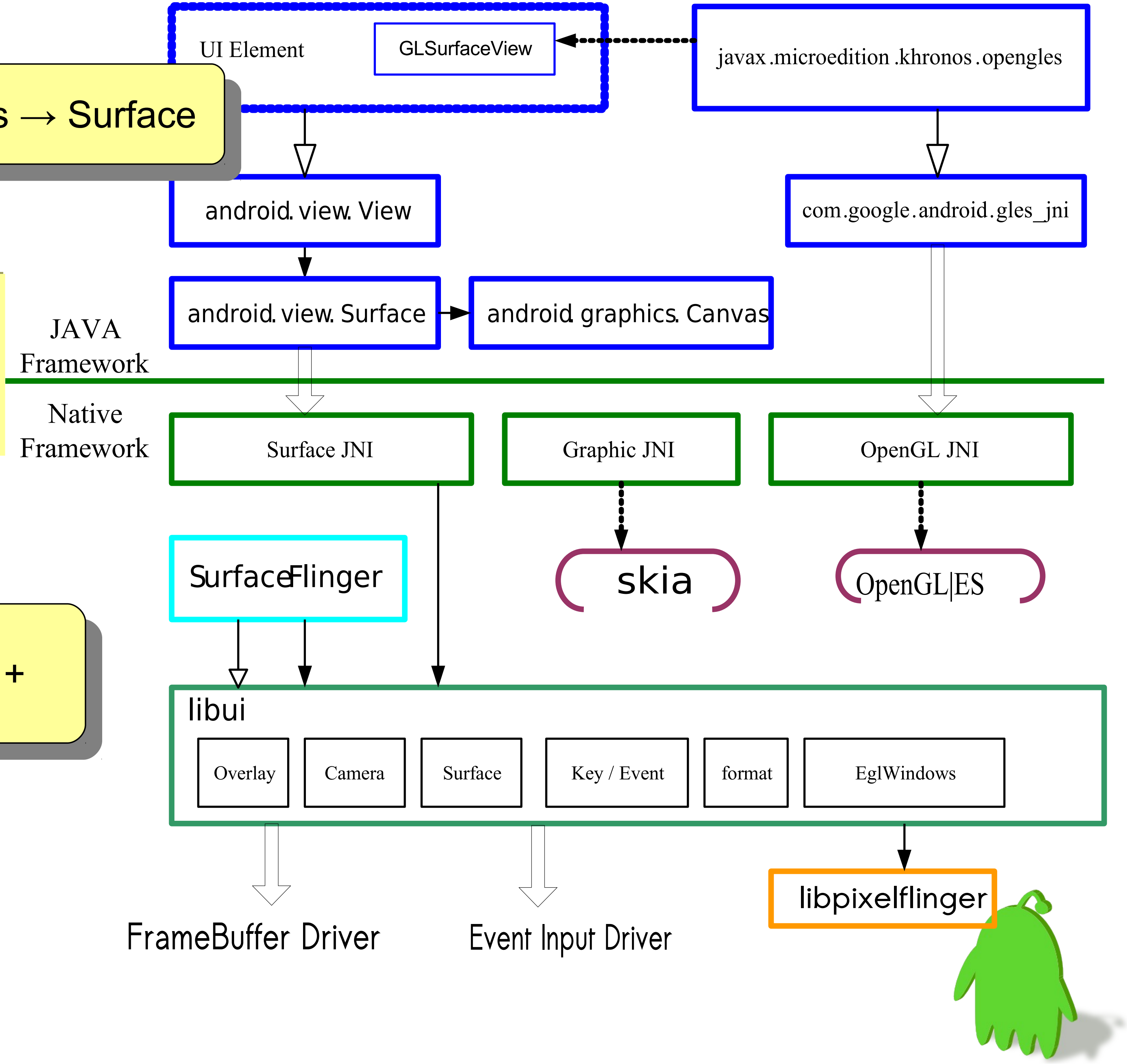


WebKit in Android



View and Canvas → Surface

Java Graphics is implemented via JNI



libui + SurfaceFlinger + libpixelflinger

Moving To Performance Now

- What is performance?
- How to measure it on GNU/Linux?
- How to do it on Android/ARM?

Computer Performance

- Amount of useful work accomplished
- Examples:
  - how fast does the page load?
  - How many frames per second are drawn?
  - How little/much bandwidth is used?
- Optimization mostly trade off between Memory and CPU usage

Performance Experiments

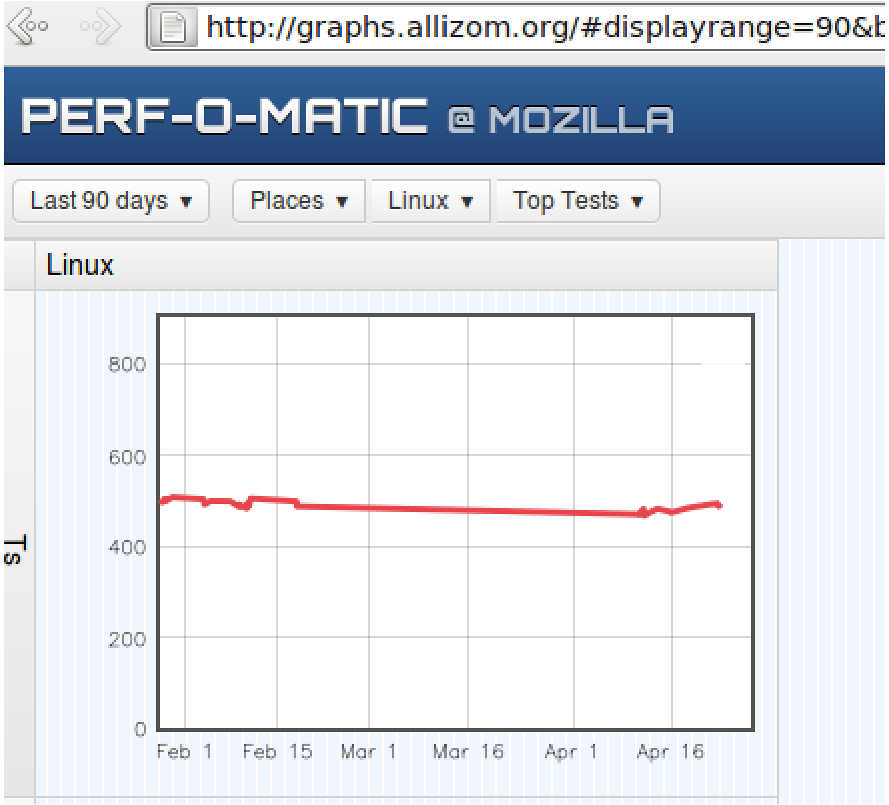
- **Do not assume, meassure it!**
- Have a manual or automatic testcase
- Observe the system while running the testcase
- Analyze the situation, make changes
- Repeat until considered good enough

Performance Experiments - Manual

- Easy to setup
- Open a site and wait, or scroll
- Good for getting an idea
- Bad for repeating and comparing results

Performance Experiments - Automatic

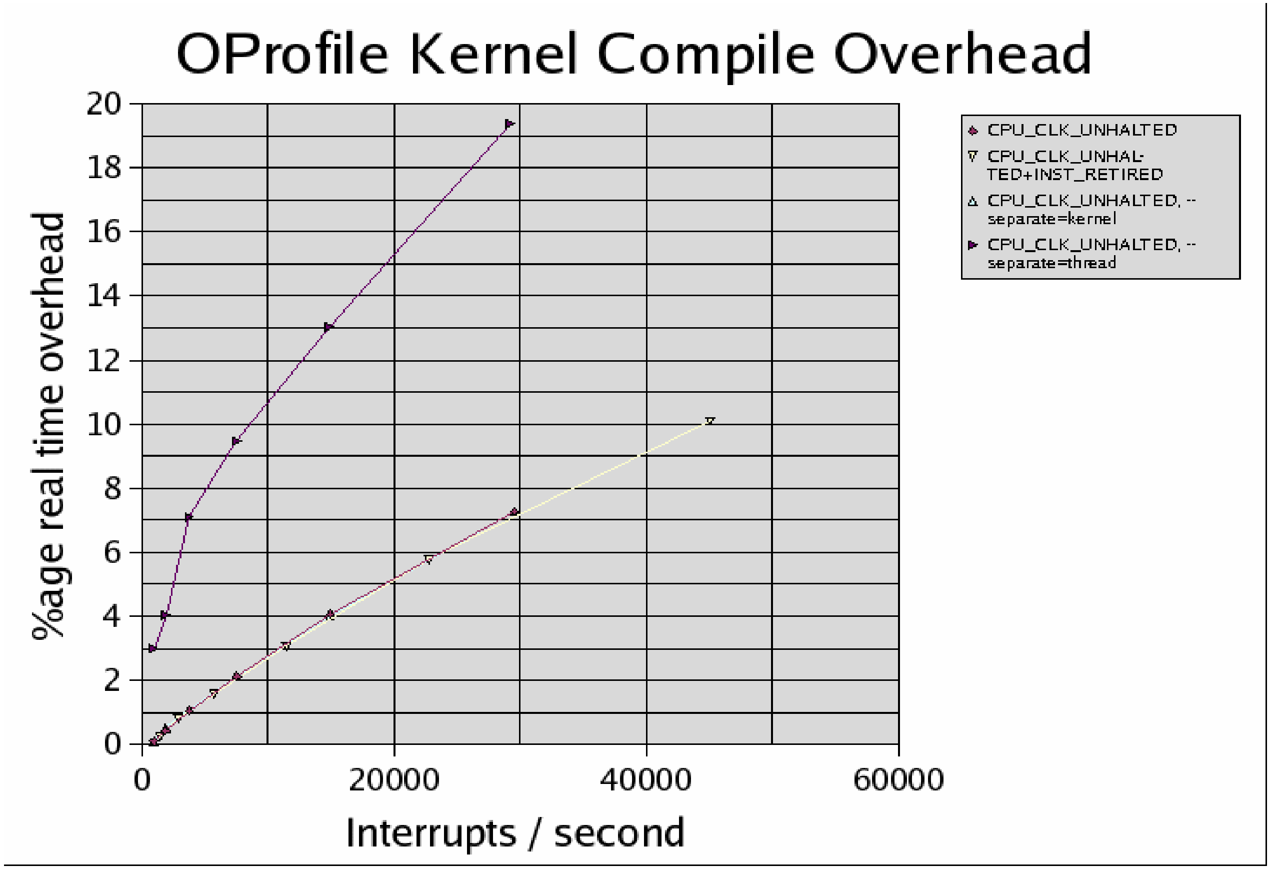
- More difficult to create
- Requires stable content
- Should allow to compare results
- Talos and others as a framework
  - https://wiki.mozilla.org/Buildbot/Talos



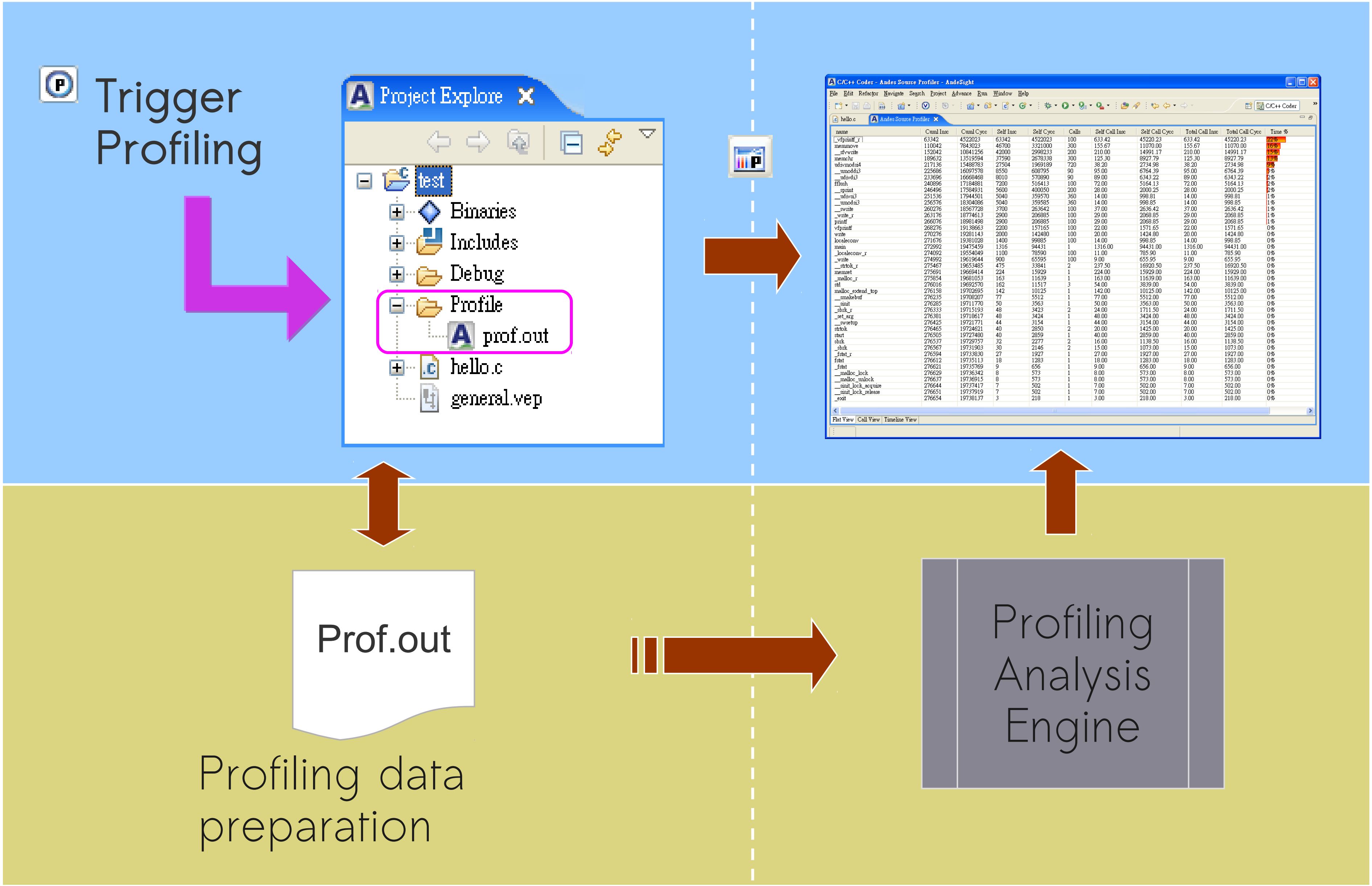
http://graphs.allizom.org/#displayrange=90&b

How to Observe on GNU/Linux

- perf (new way)
- oprofile (old way)
- Both are sampling profilers
  - ARM Performance Counter; PMU (Performance Measurement Unit)



Profiling



Trigger Profiling

Project Explore
test
Binaries
Includes
Debug
Profile
prof.out
hello.c
general.vep

C/C++ Coder - Andes Source Profiler - AndeSight
File Edit Refactor Navigate Search Project Advance Run Window Help
C/C++ Coder
hello.c
Andes Source Profiler

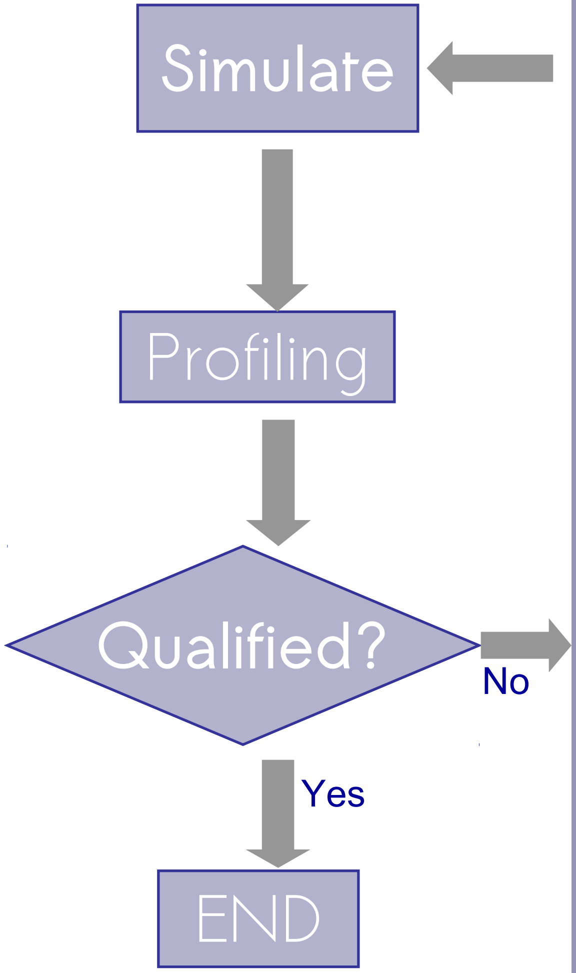
name	Cuml Insc	Cuml Cycc	Self Insc	Self Cycc	Calls	Self Call Insc	Self Call Cycc	Total Call Insc	Total Call Cycc	Time %
_vfprintf_r	63342	4522023	63342	4522023	100	633.42	45220.23	633.42	45220.23	22%
memmove	110042	7843023	46700	3321000	300	155.67	11070.00	155.67	11070.00	16%
__sfvwrite	152042	10841256	42000	2998233	200	210.00	14991.17	210.00	14991.17	15%
memchr	189632	13519594	37590	2678338	300	125.30	8927.79	125.30	8927.79	13%
udivmodsi4	217136	15488783	27504	1969189	720	38.20	2734.98	38.20	2734.98	9%
__umoddi3	225686	16097578	8550	608795	90	95.00	6764.39	95.00	6764.39	3%
__udivdi3	233696	16668468	8010	570890	90	89.00	6343.22	89.00	6343.22	2%
fflush	240896	17184881	7200	516413	100	72.00	5164.13	72.00	5164.13	2%
__sprint	246496	17584931	5600	400050	200	28.00	2000.25	28.00	2000.25	2%
__udivsi3	251536	17944501	5040	359570	360	14.00	998.81	14.00	998.81	1%
__umodsi3	256576	18304086	5040	359585	360	14.00	998.85	14.00	998.85	1%
__swrite	260276	18567728	3700	263642	100	37.00	2636.42	37.00	2636.42	1%
_write_r	263176	18774613	2900	206885	100	29.00	2068.85	29.00	2068.85	1%
printf	266076	18981498	2900	206885	100	29.00	2068.85	29.00	2068.85	1%
vfprintf	268276	19138663	2200	157165	100	22.00	1571.65	22.00	1571.65	0%
write	270276	19281143	2000	142480	100	20.00	1424.80	20.00	1424.80	0%
localeconv	271676	19381028	1400	99885	100	14.00	998.85	14.00	998.85	0%
main	272992	19475459	1316	94431	1	1316.00	94431.00	1316.00	94431.00	0%
_localeconv_r	274092	19554049	1100	78590	100	11.00	785.90	11.00	785.90	0%
_write	274992	19619644	900	65595	100	9.00	655.95	9.00	655.95	0%
__strtok_r	275467	19653485	475	33841	2	237.50	16920.50	237.50	16920.50	0%
memset	275691	19669414	224	15929	1	224.00	15929.00	224.00	15929.00	0%
_malloc_r	275854	19681053	163	11639	1	163.00	11639.00	163.00	11639.00	0%
std	276016	19692570	162	11517	3	54.00	3839.00	54.00	3839.00	0%
malloc_extend_top	276158	19702695	142	10125	1	142.00	10125.00	142.00	10125.00	0%
__smakebuf	276235	19708207	77	5512	1	77.00	5512.00	77.00	5512.00	0%
__sinit	276285	19711770	50	3563	1	50.00	3563.00	50.00	3563.00	0%
_sbrk_r	276333	19715193	48	3423	2	24.00	1711.50	24.00	1711.50	0%
_set_arg	276381	19718617	48	3424	1	48.00	3424.00	48.00	3424.00	0%
__swsetup	276425	19721771	44	3154	1	44.00	3154.00	44.00	3154.00	0%
strtok	276465	19724621	40	2850	2	20.00	1425.00	20.00	1425.00	0%
start	276505	19727480	40	2859	1	40.00	2859.00	40.00	2859.00	0%
sbrk	276537	19729757	32	2277	2	16.00	1138.50	16.00	1138.50	0%
_sbrk	276567	19731903	30	2146	2	15.00	1073.00	15.00	1073.00	0%
_fstat_r	276594	19733830	27	1927	1	27.00	1927.00	27.00	1927.00	0%
fstat	276612	19735113	18	1283	1	18.00	1283.00	18.00	1283.00	0%
_fstat	276621	19735769	9	656	1	9.00	656.00	9.00	656.00	0%
__malloc_lock	276629	19736342	8	573	1	8.00	573.00	8.00	573.00	0%
__malloc_unlock	276637	19736915	8	573	1	8.00	573.00	8.00	573.00	0%
__sinit_lock_acquire	276644	19737417	7	502	1	7.00	502.00	7.00	502.00	0%
__sinit_lock_release	276651	19737919	7	502	1	7.00	502.00	7.00	502.00	0%
_exit	276654	19738137	3	218	1	3.00	218.00	3.00	218.00	0%

Prof.out

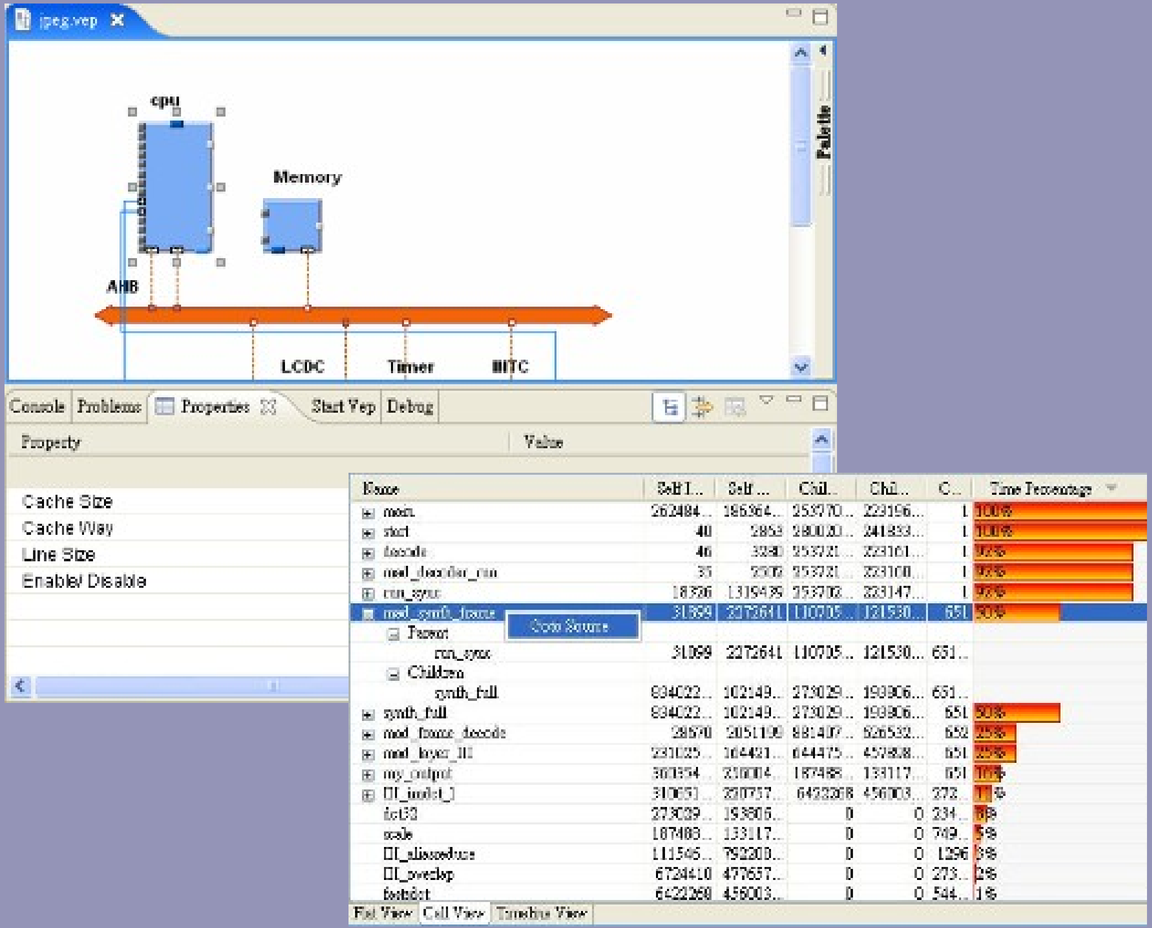
Profiling data preparation

Profiling Analysis Engine

Evaluating and Tuning



Tune Performance by CPU Configuration



Property	Value
Cache Size	
Cache Way	
Line Size	
Enable/ Disable	

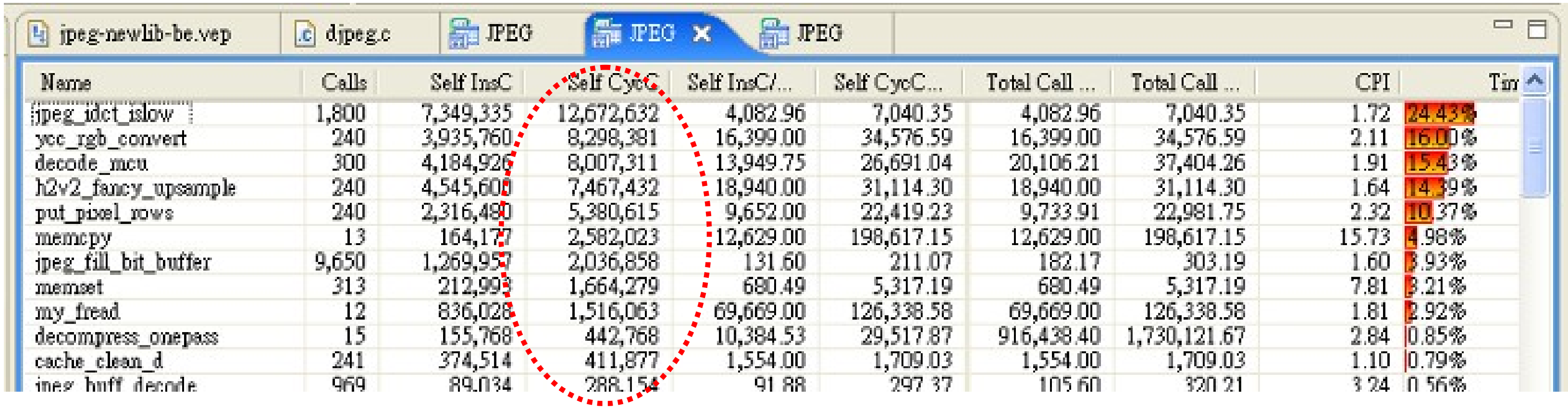
Name	Self T...	Self ...	Chil...	Chil...	C...	Time Percentage
main	262484...	186364...	253770...	223196...	1	100%
start	40	2853	280020...	241833...	1	100%
decode	46	3280	253721...	223161...	1	92%
mad_decoder_run	35	2509	253721...	223100...	1	92%
run_sync	18326	1319439	253702...	223147...	1	92%
mad_synth_frame	31899	2272641	110705...	121530...	651	50%
Parent						
run_sync	31099	2272641	110705...	121530...	651...	
Children						
synth_full	834022...	102149...	273029...	193806...	651...	
synth_full	834022...	102149...	273029...	193806...	651	50%
mad_frame_decode	28570	2051190	881407...	526532...	652	25%
mad_layer_III	231025...	164421...	644475...	457808...	651	25%
my_output	300354...	210004...	187488...	133117...	651	16%
III_imdct_l	310651...	220737...	6422268	456003...	272...	11%
dct32	273029...	193806...	0	0	234...	8%
scale	187488...	133117...	0	0	749...	5%
III_aliasreduce	111546...	792200...	0	0	1296	3%
III_overlap	6724410	477657...	0	0	273...	2%
fastsdct	6422268	456003...	0	0	544...	1%

Reference oprofile usage

```
# prepare the setup
$ rm -rf /var/lib/oprofile
$ opcontrol --start-daemon -p library -c 10

# run the app once to force loading it from nfs into the cache
$ ./tst_something

# start profiling
$ opcontrol -start
$ ./tst_something -iterations enough

# stop profiling
$ opcontrol -h

# generate reports
$ opreport -l
$ opreport -c …
```

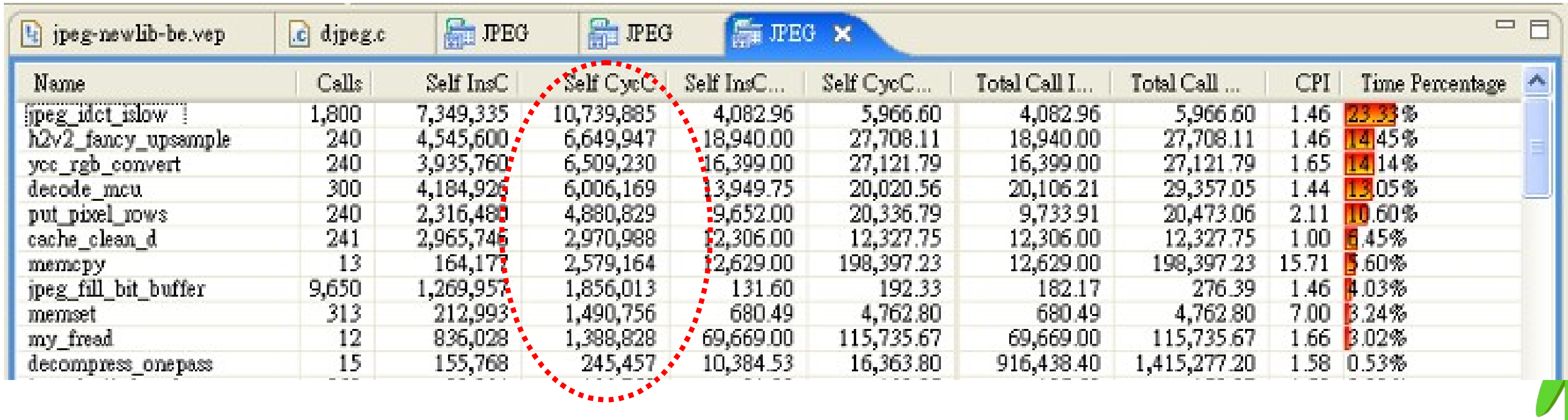
Profiler sample

Profile Result (A)

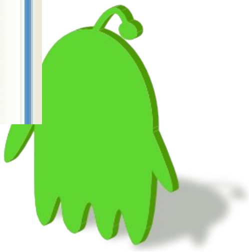


Name	Calls	Self InsC	Self CycC	Self InsC/...	Self CycC...	Total Call ...	Total Call ...	CPI	Tim
jpeg_idct_islow	1,800	7,349,335	12,672,632	4,082.96	7,040.35	4,082.96	7,040.35	1.72	24.43%
ycc_rgb_convert	240	3,935,760	8,298,381	16,399.00	34,576.59	16,399.00	34,576.59	2.11	16.00%
decode_mcu	300	4,184,926	8,007,311	13,949.75	26,691.04	20,106.21	37,404.26	1.91	15.43%
h2v2_fancy_upsample	240	4,545,600	7,467,432	18,940.00	31,114.30	18,940.00	31,114.30	1.64	14.39%
put_pixel_rows	240	2,316,480	5,380,615	9,652.00	22,419.23	9,733.91	22,981.75	2.32	10.37%
memcpy	13	164,177	2,582,023	12,629.00	198,617.15	12,629.00	198,617.15	15.73	4.98%
jpeg_fill_bit_buffer	9,650	1,269,957	2,036,858	131.60	211.07	182.17	303.19	1.60	3.93%
memset	313	212,993	1,664,279	680.49	5,317.19	680.49	5,317.19	7.81	3.21%
my_fread	12	836,028	1,516,063	69,669.00	126,338.58	69,669.00	126,338.58	1.81	2.92%
decompress_onepass	15	155,768	442,768	10,384.53	29,517.87	916,438.40	1,730,121.67	2.84	0.85%
cache_clean_d	241	374,514	411,877	1,554.00	1,709.03	1,554.00	1,709.03	1.10	0.79%
jpeg_huff_decode	969	89,034	288,154	91.88	297.37	105.60	320.21	3.24	0.56%

Profile Result (B)

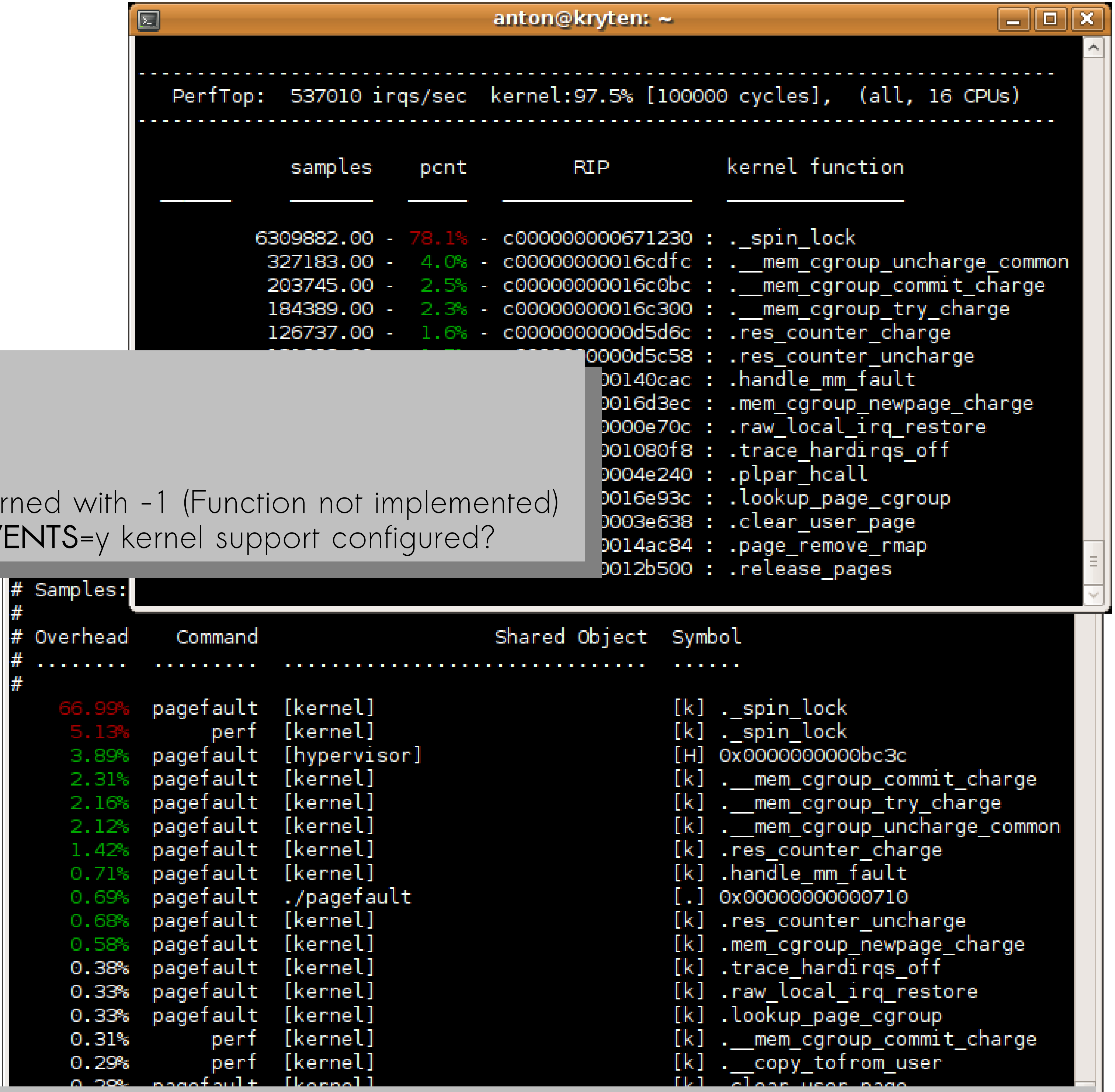


Name	Calls	Self InsC	Self CycC	Self InsC...	Self CycC...	Total Call I...	Total Call ...	CPI	Time Percentage
jpeg_idct_islow	1,800	7,349,335	10,739,885	4,082.96	5,966.60	4,082.96	5,966.60	1.46	23.33%
h2v2_fancy_upsample	240	4,545,600	6,649,947	18,940.00	27,708.11	18,940.00	27,708.11	1.46	14.45%
ycc_rgb_convert	240	3,935,760	6,509,230	16,399.00	27,121.79	16,399.00	27,121.79	1.65	14.14%
decode_mcu	300	4,184,926	6,006,169	13,949.75	20,020.56	20,106.21	29,357.05	1.44	13.05%
put_pixel_rows	240	2,316,480	4,880,829	9,652.00	20,336.79	9,733.91	20,473.06	2.11	10.60%
cache_clean_d	241	2,965,746	2,970,988	12,306.00	12,327.75	12,306.00	12,327.75	1.00	6.45%
memcpy	13	164,177	2,579,164	12,629.00	198,397.23	12,629.00	198,397.23	15.71	5.60%
jpeg_fill_bit_buffer	9,650	1,269,957	1,856,013	131.60	192.33	182.17	276.39	1.46	4.03%
memset	313	212,993	1,490,756	680.49	4,762.80	680.49	4,762.80	7.00	3.24%
my_fread	12	836,028	1,388,828	69,669.00	115,735.67	69,669.00	115,735.67	1.66	3.02%
decompress_onepass	15	155,768	245,457	10,384.53	16,363.80	916,438.40	1,415,277.20	1.58	0.53%



perf is powerful.

```
# perf record -s CMD
  Error: perfcounter syscall returned with -1 (Function not implemented)
  Fatal: No CONFIG_PERF_EVENTS=y kernel support configured?
```



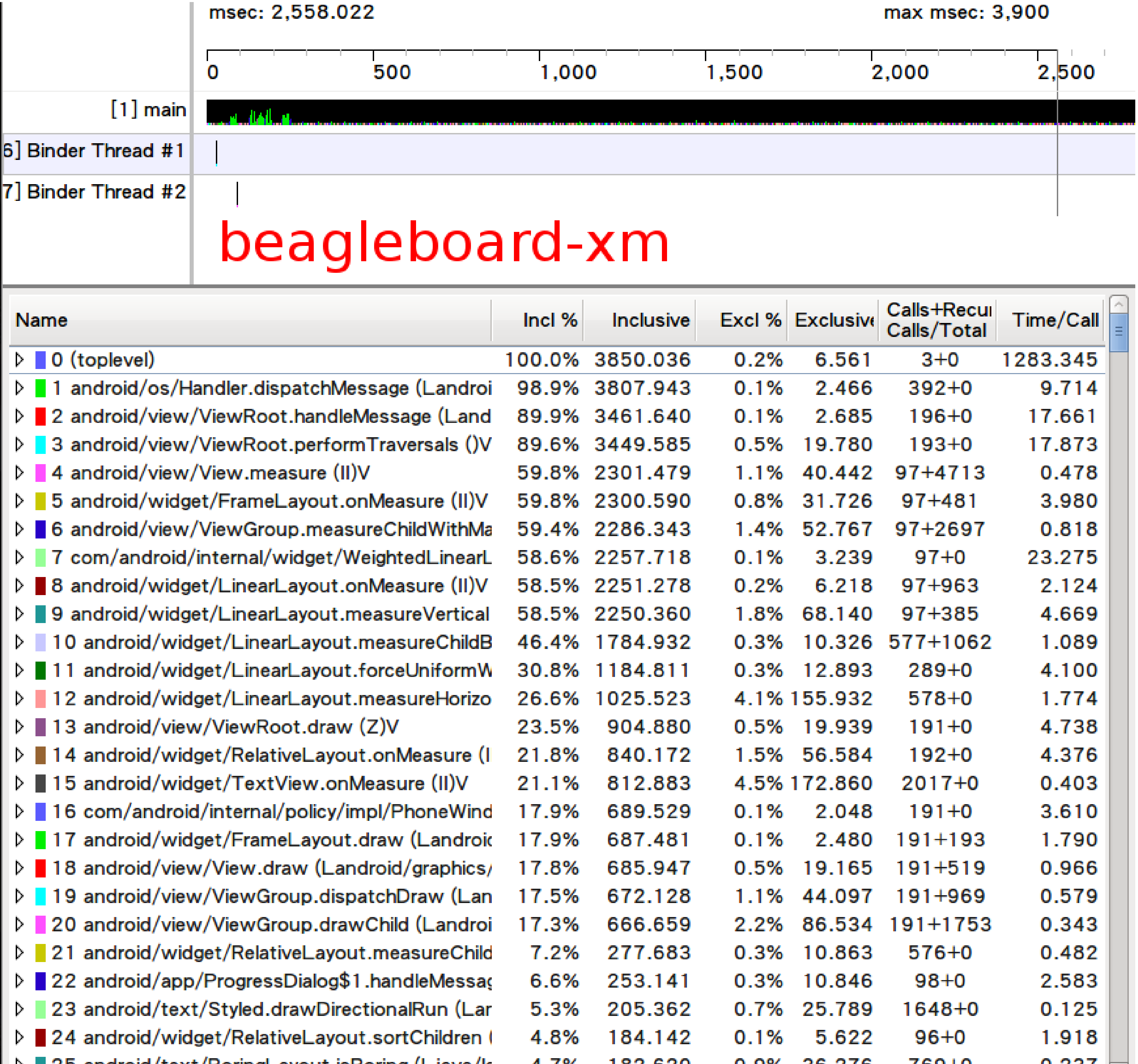
```
------------------------------------------------------------------------------
   PerfTop:  537010 irqs/sec  kernel:97.5% [100000 cycles],  (all, 16 CPUs)
------------------------------------------------------------------------------

             samples    pcnt         RIP          kernel function
  ______     _______   _____   ________________   _______________

          6309882.00 - 78.1% - c000000000671230 : ._spin_lock
           327183.00 -  4.0% - c00000000016cdfc : .__mem_cgroup_uncharge_common
           203745.00 -  2.5% - c00000000016c0bc : .__mem_cgroup_commit_charge
           184389.00 -  2.3% - c00000000016c300 : .__mem_cgroup_try_charge
           126737.00 -  1.6% - c0000000000d5d6c : .res_counter_charge
                                     0000d5c58 : .res_counter_uncharge
                                     00140cac : .handle_mm_fault
                                     0016d3ec : .mem_cgroup_newpage_charge
                                     0000e70c : .raw_local_irq_restore
                                     001080f8 : .trace_hardirqs_off
                                     0004e240 : .plpar_hcall
                                     0016e93c : .lookup_page_cgroup
                                     0003e638 : .clear_user_page
                                     0014ac84 : .page_remove_rmap
                                     0012b500 : .release_pages
```

```
# Samples:
#
# Overhead    Command                     Shared Object  Symbol
# ........  .........  ..............................  ......
#
    66.99%  pagefault  [kernel]                        [k] ._spin_lock
     5.13%       perf  [kernel]                        [k] ._spin_lock
     3.89%  pagefault  [hypervisor]                    [H] 0x0000000000bc3c
     2.31%  pagefault  [kernel]                        [k] .__mem_cgroup_commit_charge
     2.16%  pagefault  [kernel]                        [k] .__mem_cgroup_try_charge
     2.12%  pagefault  [kernel]                        [k] .__mem_cgroup_uncharge_common
     1.42%  pagefault  [kernel]                        [k] .res_counter_charge
     0.71%  pagefault  [kernel]                        [k] .handle_mm_fault
     0.69%  pagefault  ./pagefault                     [.] 0x00000000000710
     0.68%  pagefault  [kernel]                        [k] .res_counter_uncharge
     0.58%  pagefault  [kernel]                        [k] .mem_cgroup_newpage_charge
     0.38%  pagefault  [kernel]                        [k] .trace_hardirqs_off
     0.33%  pagefault  [kernel]                        [k] .raw_local_irq_restore
     0.33%  pagefault  [kernel]                        [k] .lookup_page_cgroup
     0.31%       perf  [kernel]                        [k] .__mem_cgroup_commit_charge
     0.29%       perf  [kernel]                        [k] .__copy_tofrom_user
     0.28%  pagefault  [kernel]                        [k] .clear_user_page
```

http://anton.ozlabs.org/blog/2009/09/04/using-performance-counters-for-linux/

How to Do On Android/ARM?

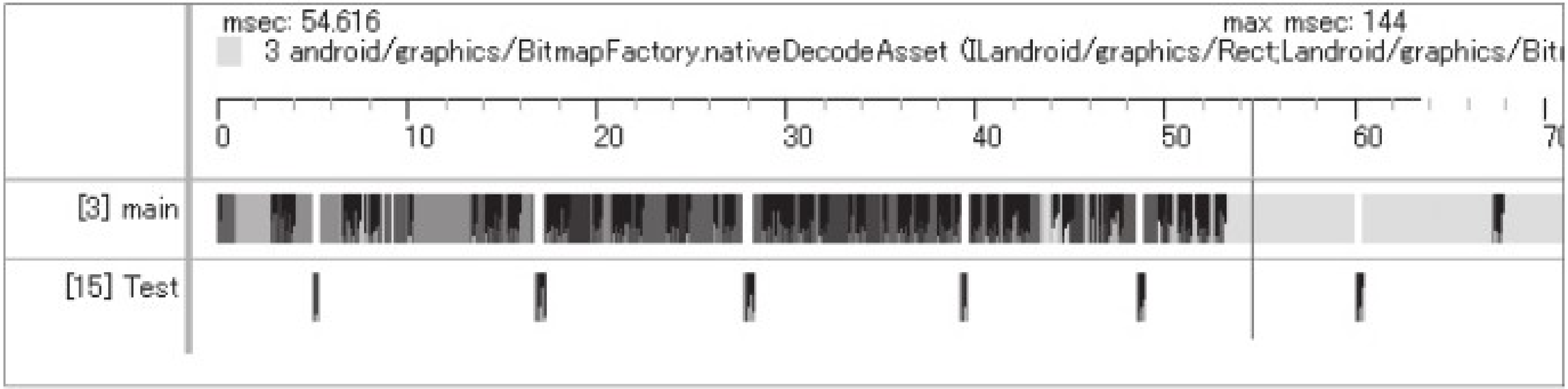
- for Native libraries →
  - Use '**perf**' built without libperl, libpython
  - oprofiled and opcontrol are there, CPU data is missing
  - Binaries for ARM need frame pointers to have backtraces
- Java part is the performance hell always.
  - **traceview** is a great tool for Java performance analysis.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



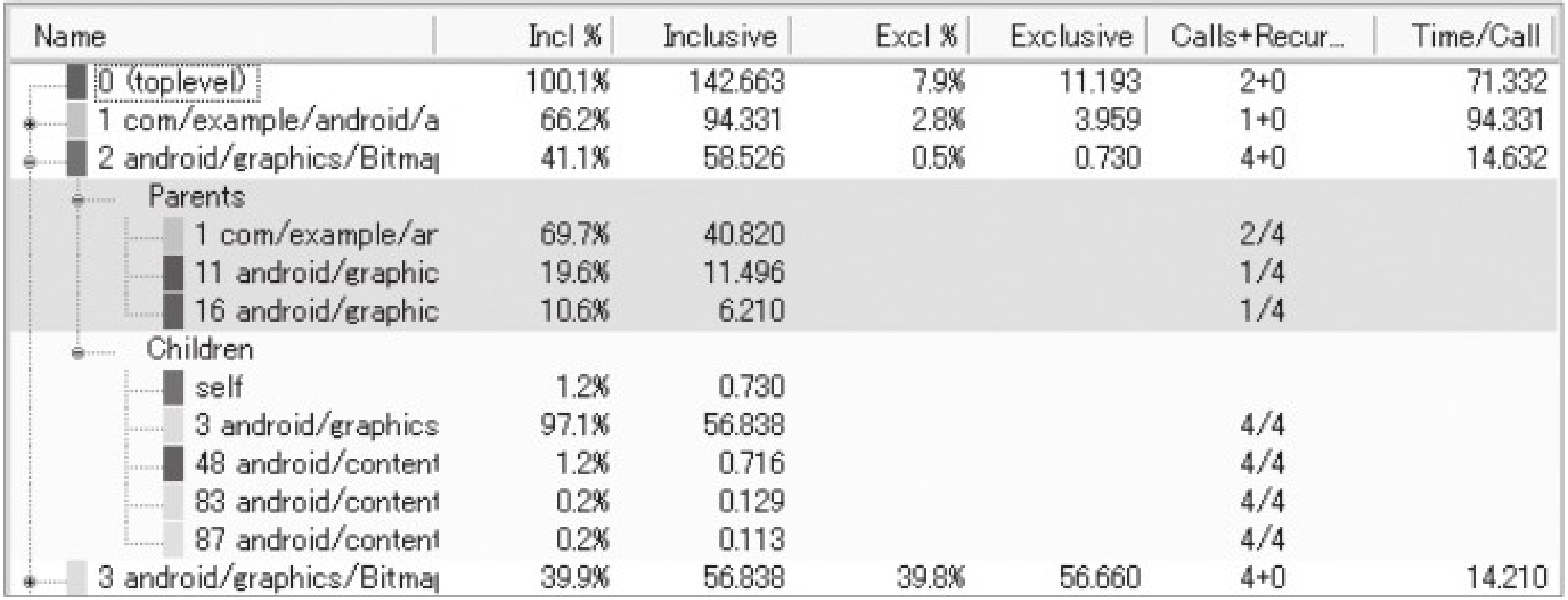
beagleboard-xm

Name	Incl %	Inclusive	Excl %	Exclusive	Calls+Recur 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%	182.630	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
0
10
20
30
40
50
60
70
[3] main
[15] Test

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

How to Do On Android/ARM?

- Upload some more files
- Start oprofile with opcontrol on the device
- Run the test on the device
- Analyze with opreport on the PC

```
# 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.

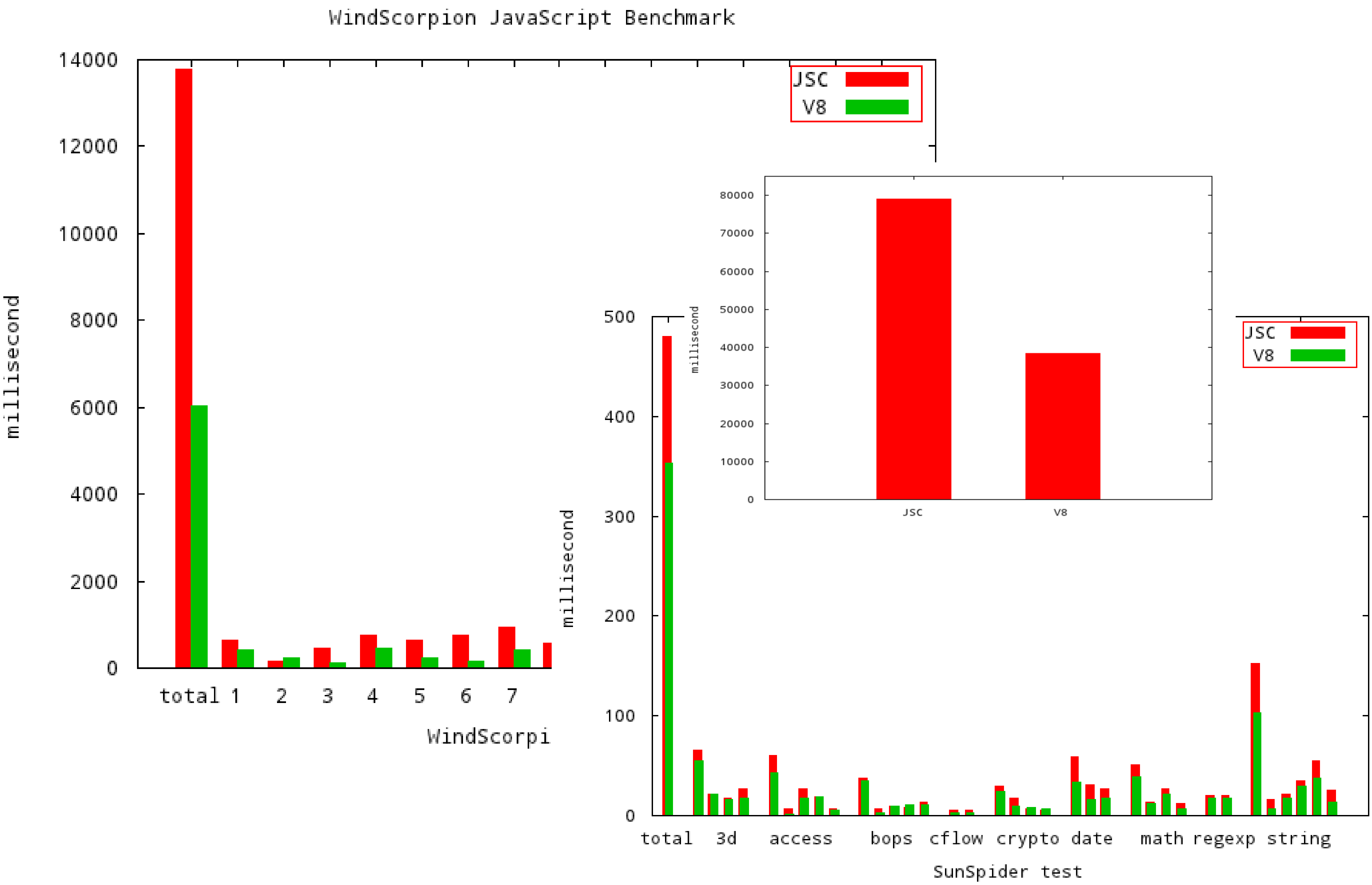
Picking or Creating a Testcase

- What to measure? Loading, Painting, Scrolling?
- No excellent benchmark suite available due to copyright issues
- Some frameworks are available, but mostly manual work

Introducing The Methanol Framework

- Small Framework from the University of Szeged
- Can load pages and count the time
- Provides a summary with error interval

http://webkit.sed.hu/blog/20101216/benchmarking-qtwebkit-v8-linux



Issues With Methanol Framework

- Webpages need to be converted
- Everything loaded from the same URL
- http://gitorious.org/methanol

Putting Everything Together

- Using methanol with the example page
- Executing it in the Browser
- Observing it with oprofile/perf

0xlab

http://0xlab.org