



## Benchmarking Techniques

Michael Hope <[michael.hope@linaro.org](mailto:michael.hope@linaro.org)>

# About Linaro

Compilers, debuggers, emulation, profiling, trace

Upstream first, then backport

GCC, GDB, QEMU,  
perf, valgrind, ltrace

A selection of improvements

Vectoriser

Thumb-2 support in libraries

Cross-debug support

Cortex-A9 model in QEMU

# Methods

Open, accessible group

Bug process with timing

Open planning

Auto builder

Merge requests

Verification

Performance testing

Why benchmark?

Issues are

Relevance

Accuracy

Repeatability

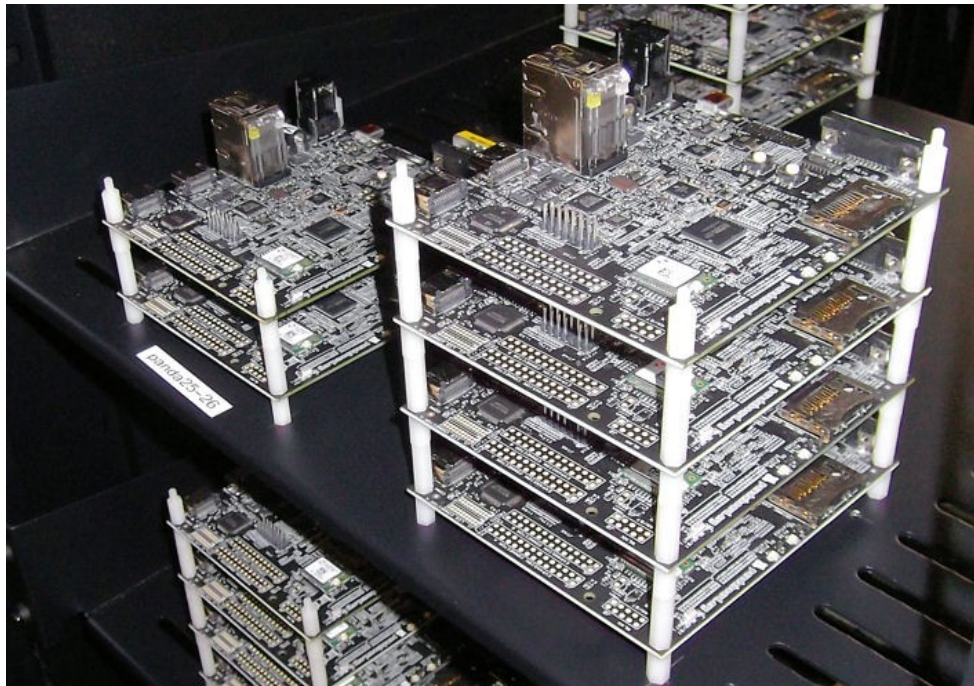
Picking relevant benchmarks:

Profile  
Workload  
Features

We use SPEC 2000 and EEMBC

We'd like shareable benchmarks

# Test Platform



## Build hosts

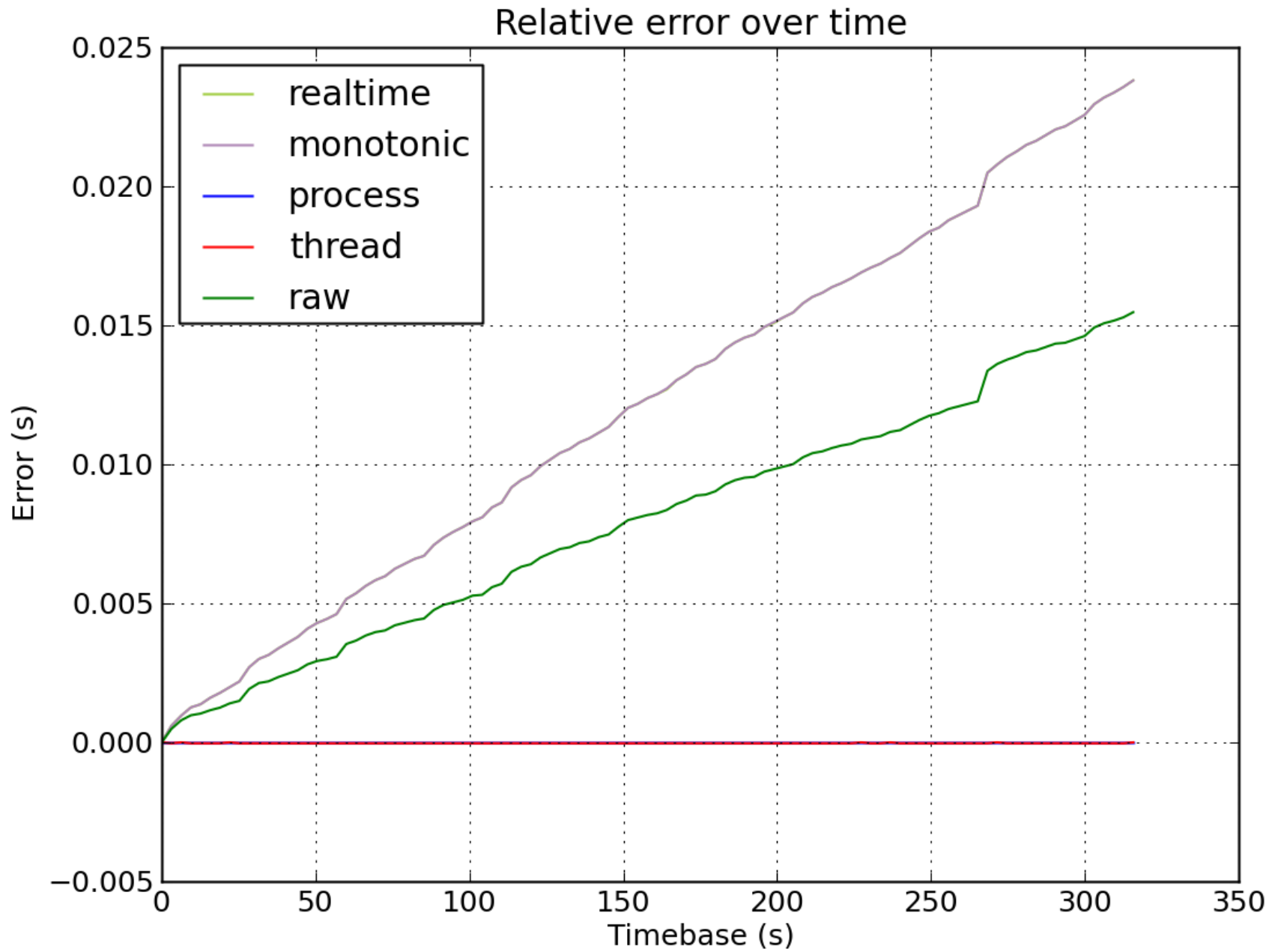
Name	State	Details	Last seen	
leo1	idle	load 0.04 users cbuid	1.2 minutes	
leo2	idle	load 0.04 users cbuid	1.5 minutes	
oort1	lurking	load 1.16 users cbuid	1.7 minutes	
oort2	lurking	load 0.42 users cbuid	0.5 minutes	
pavo1	idle	load 0.0 users cbuid	1.5 minutes	
tcpanda01	idle	load 0.0 users cbuid	1.0 minutes	
tcpanda02	idle	load 0.05 users cbuid	1.1 minutes	
tcpanda03	updating	load 0.0 users cbuid	0.3 minutes	
tcpanda04	idle	load 0.03 users cbuid	2.2 minutes	
tcpanda05	updating	load 0.0 users cbuid	0.3 minutes	
tcpanda06	updating	load 0.0 users cbuid	0.2 minutes	
ursa1	reserved	load 0.05 users michaelh	michaelh1	2.7 days
ursa2	running	load 2.32 users cbuid,michaelh	denbench-o3-neon-cortexa9-build	5.2 minutes
ursa3	running	load 1.73 users cbuid,michaelh	spec2000-o3-neon-cortexa8-run	25 minutes
ursa4	running	load 0.0 users cbuid	denbench-o3-neon-publish	4.8 days

Build / test / benchmark via Linux / web / commodity hardware

# Measuring

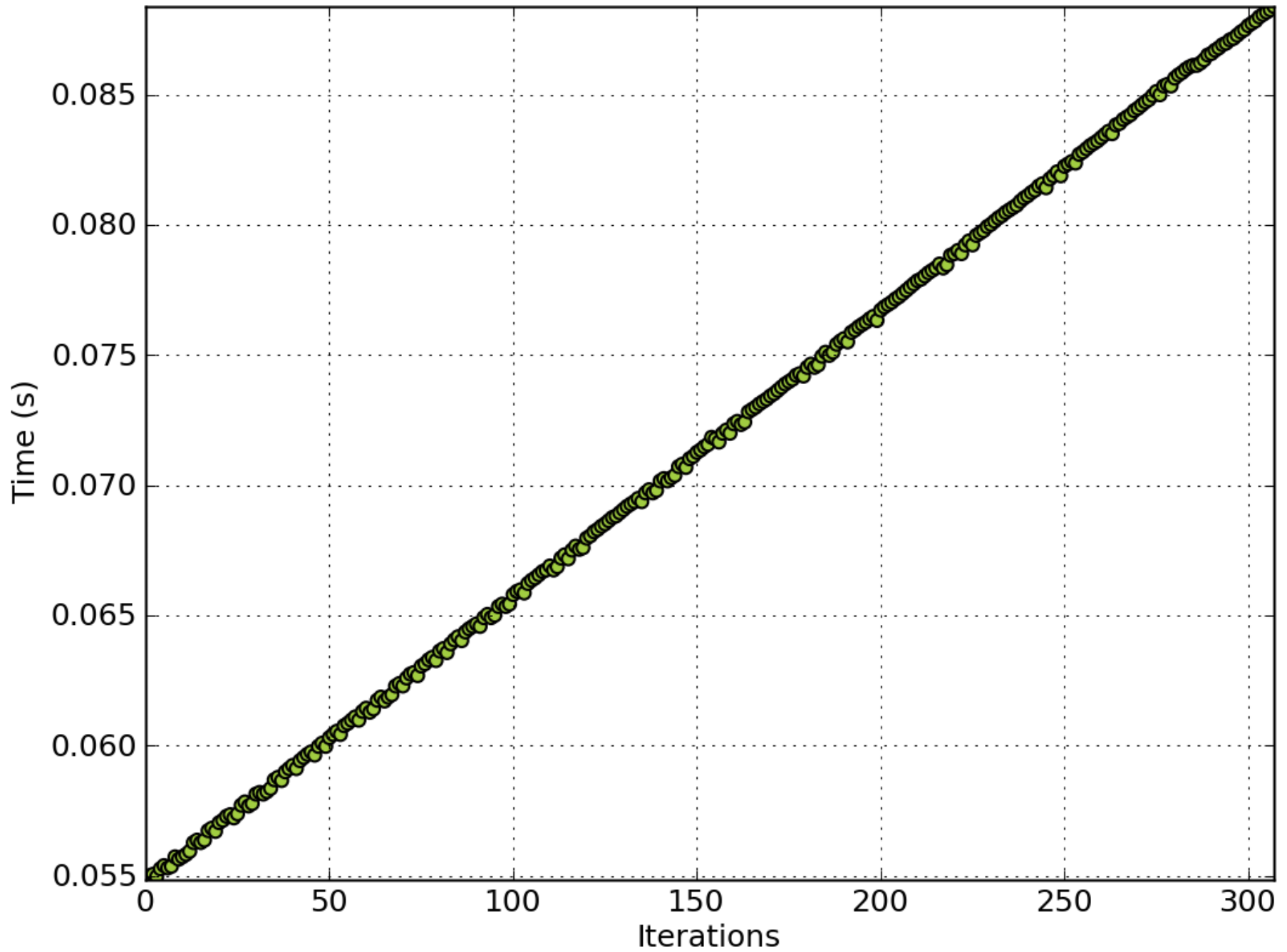
## Realtime timers

See `'man clock_gettime'`

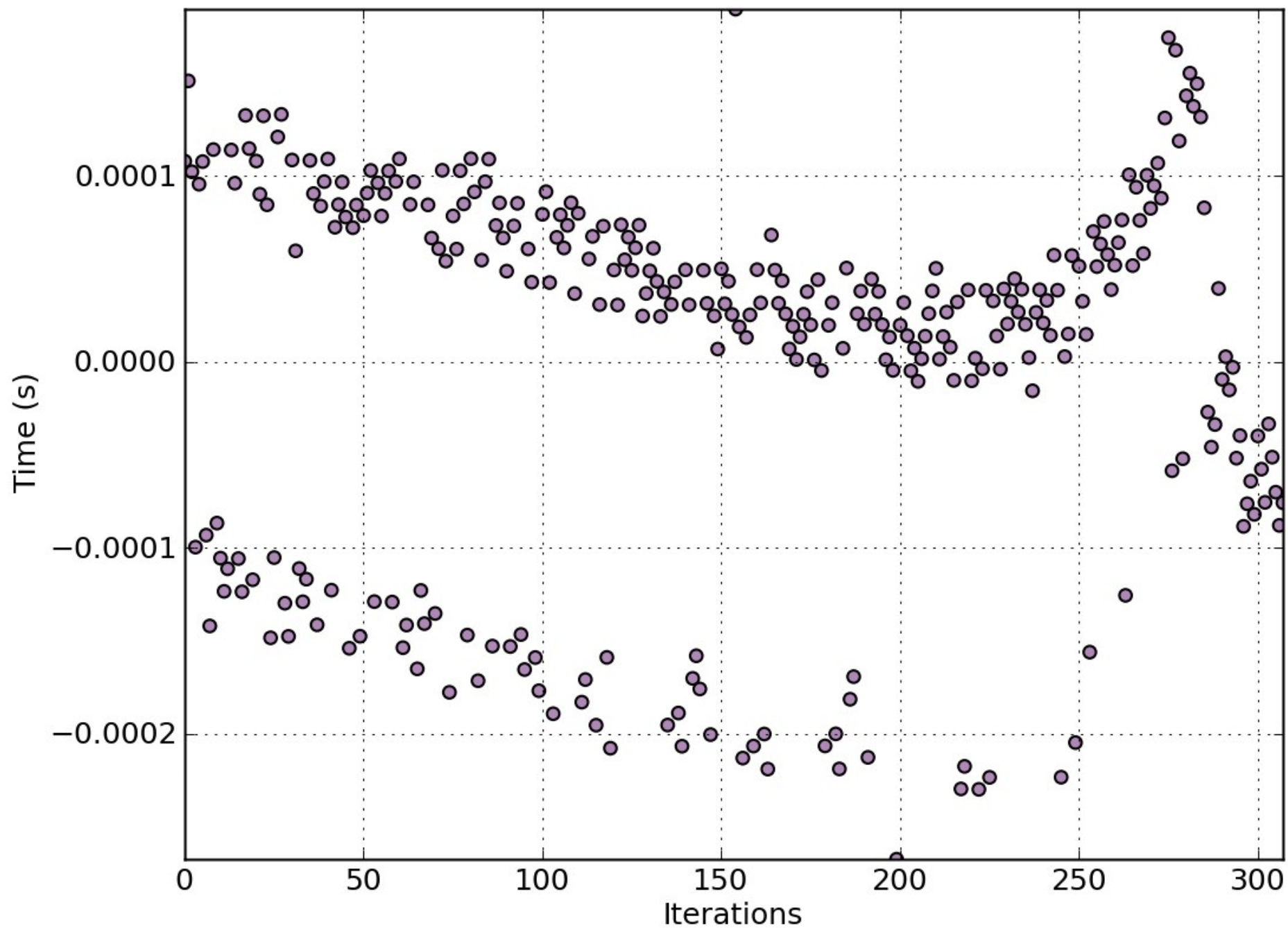


See <https://wiki.linaro.org/WorkingGroups/ToolChain/Benchmarks/TimerAccuracy> 12

Time to perform a number of iterations



Detrended time to perform a number of iterations



# External influences

## Idle

	<b>Wall mean</b>	<b>Mean</b>	<b>Stddev</b>	<b>Mean vs plain</b>	<b>Stddev vs plain</b>
Plain	4.993	4.993	0.000353		
High	4.993	4.993	0.000190	100.001%	53.875%
Nice	4.993	4.993	0.000255	99.999%	72.225%

## CPU load

	<b>Wall mean</b>	<b>Mean</b>	<b>Stddev</b>	<b>Mean vs plain</b>	<b>Stddev vs plain</b>
Plain	14.966	4.992	0.007007		
High	5.096	4.992	0.004316	99.989%	61.589%
Nice	65.311	4.991	0.005056	99.971%	72.149%

## Network load

	<b>Wall mean</b>	<b>Mean</b>	<b>Stddev</b>	<b>Mean vs plain</b>	<b>Stddev vs plain</b>
Plain	5.134	5.099	0.005952		
High	5.112	5.099	0.005798	100.009%	97.397%
Nice	5.134	5.099	0.005952	100.000%	100.000%

Other features to be wary of

scheduler

governor

cpuidle, power management, thermal limiting

SMP

bugs, like core lockdown

NEON startup

See “Understanding the Linux Kernel” ch10:

<http://oreilly.com/catalog/linuxkernel/chapter/ch10.html>

# Dispersion across boards ...and across tests

# Putting things together and running

We use:

timer built into the app

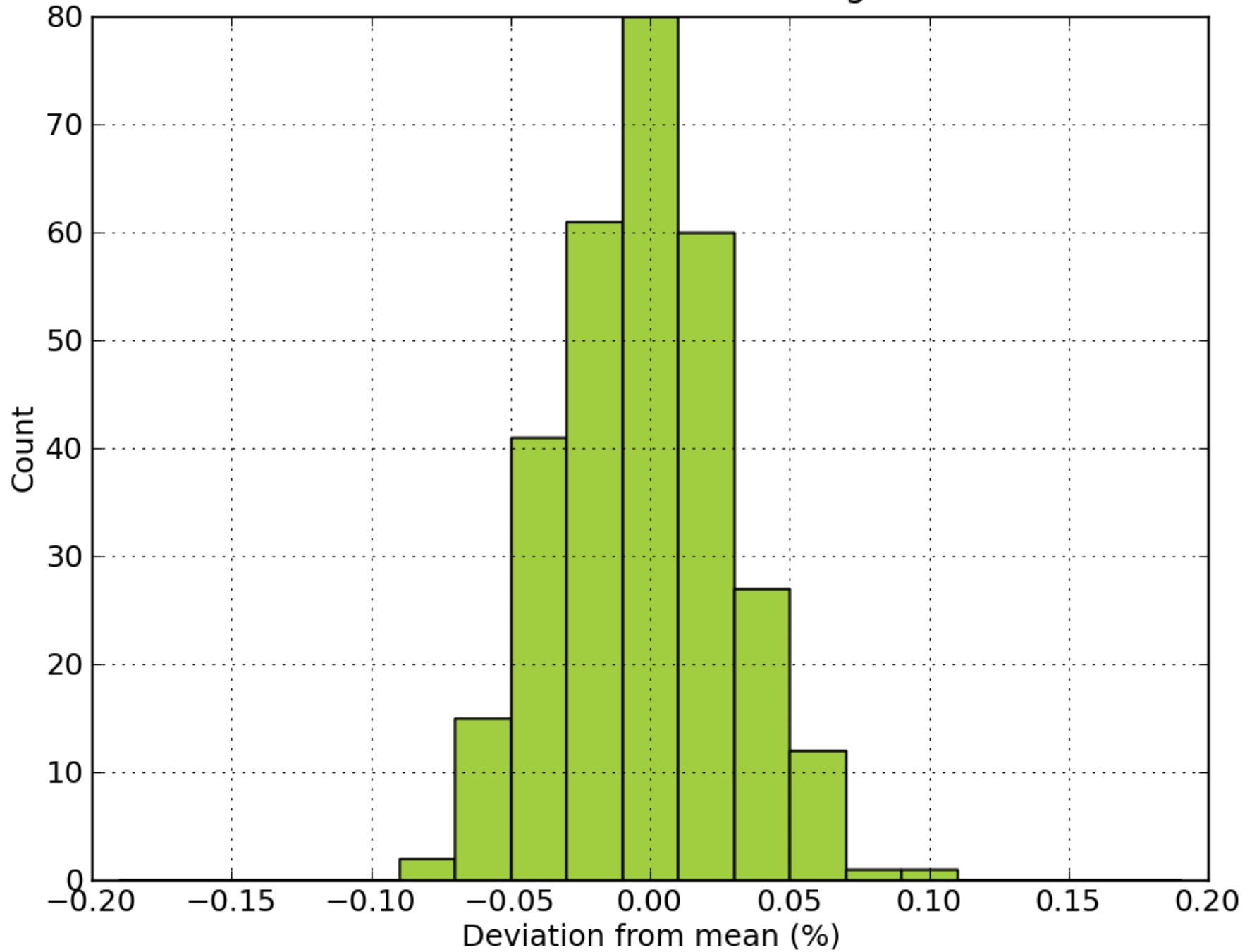
run five times

collect everything

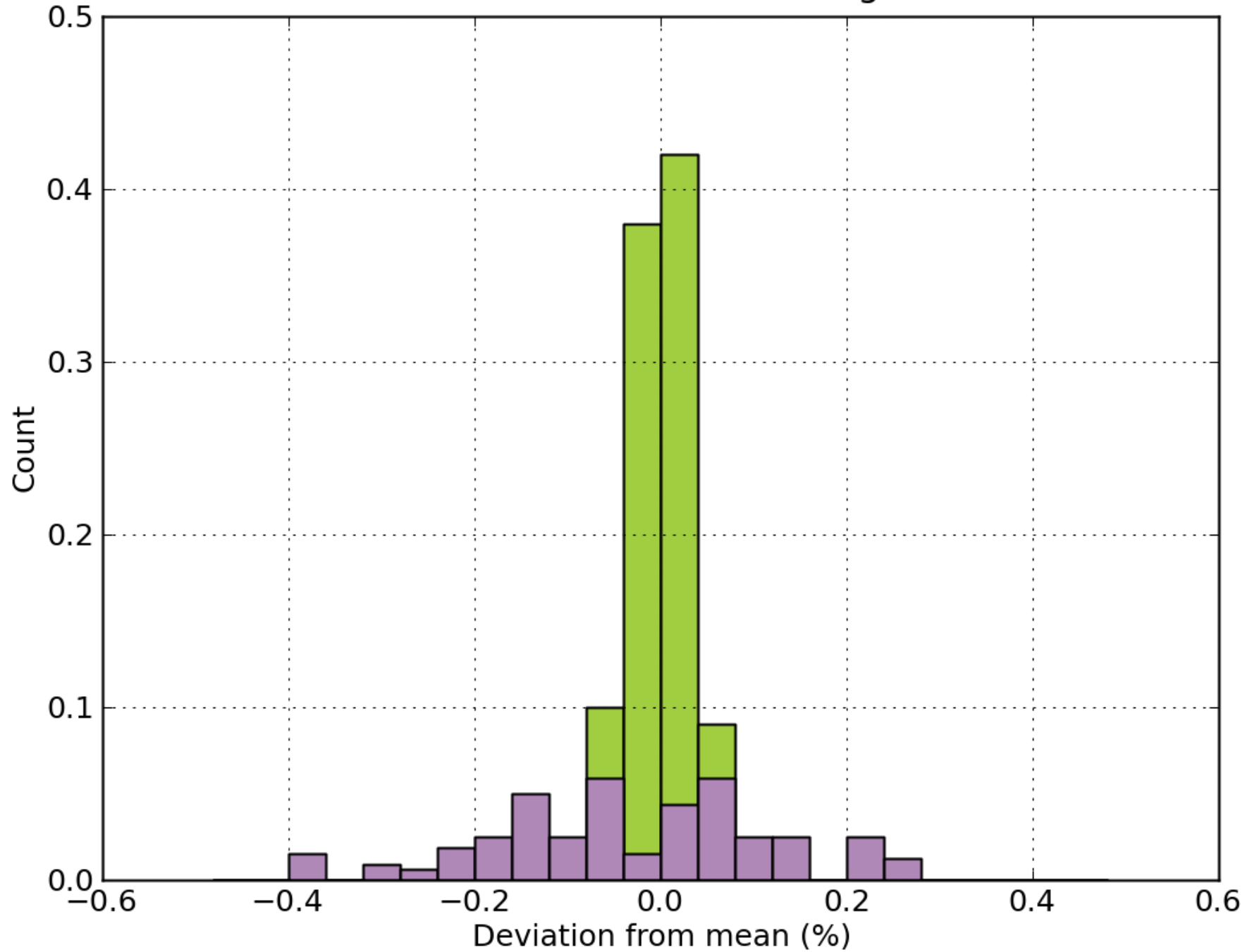
post process

# Statistics

coremark normalised histogram



cacheb01 normalised histogram

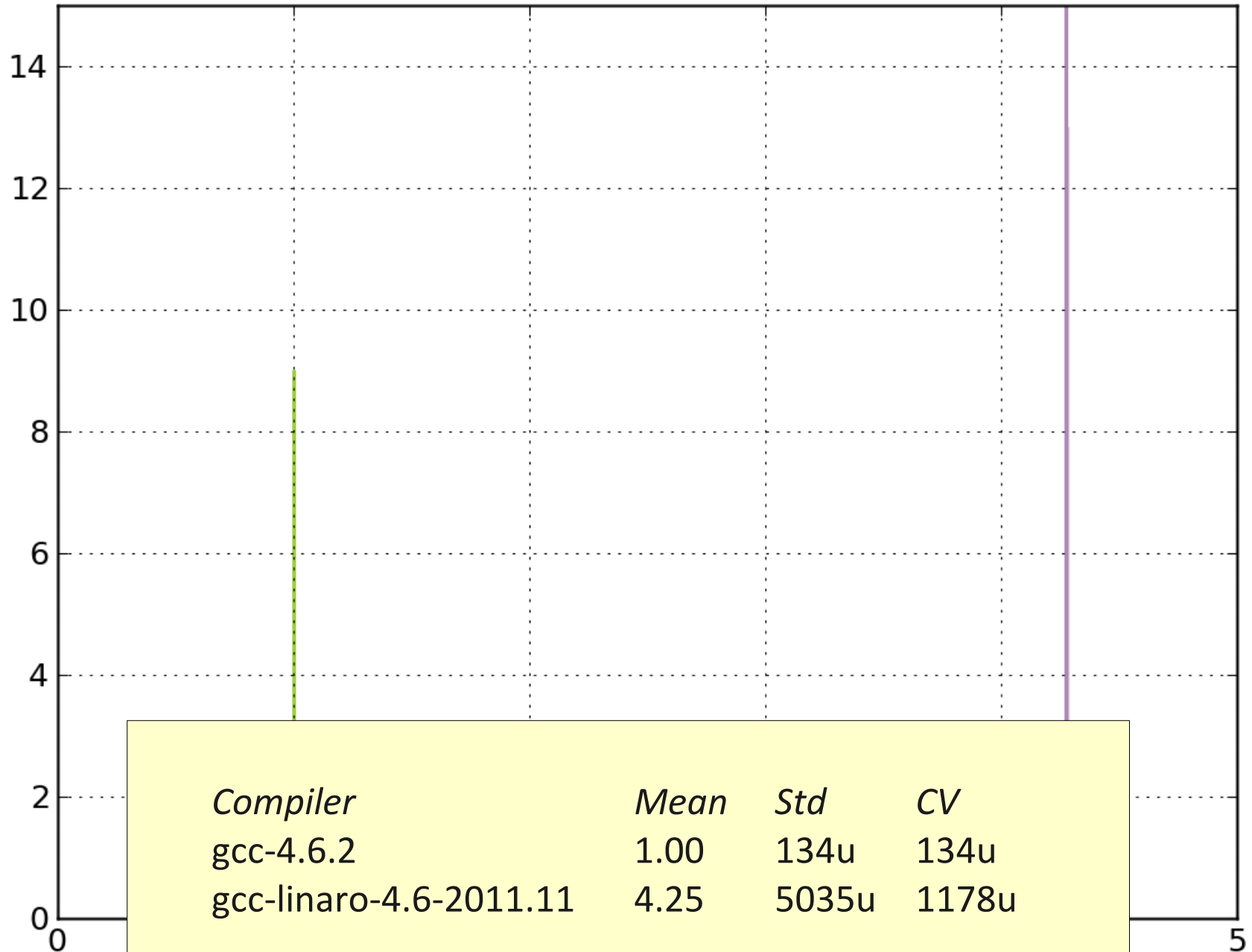


Standard deviation  
Dispersion / coefficient of variance  
t-scores  
t-test

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}$$

See [http://en.wikipedia.org/wiki/Welch%27s\\_t\\_test](http://en.wikipedia.org/wiki/Welch%27s_t_test)<sup>24</sup>

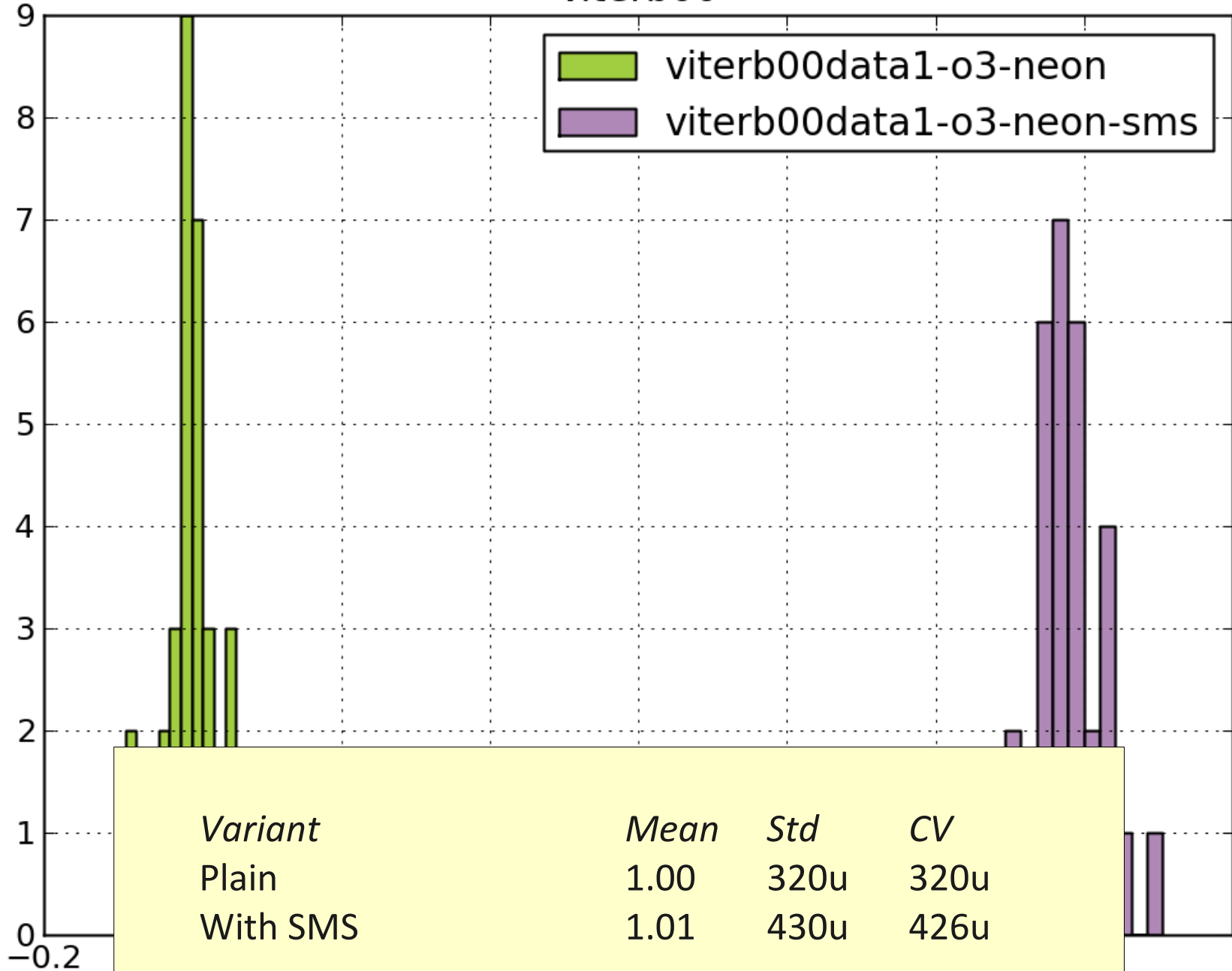
# rgbcmy01



t = 30,300

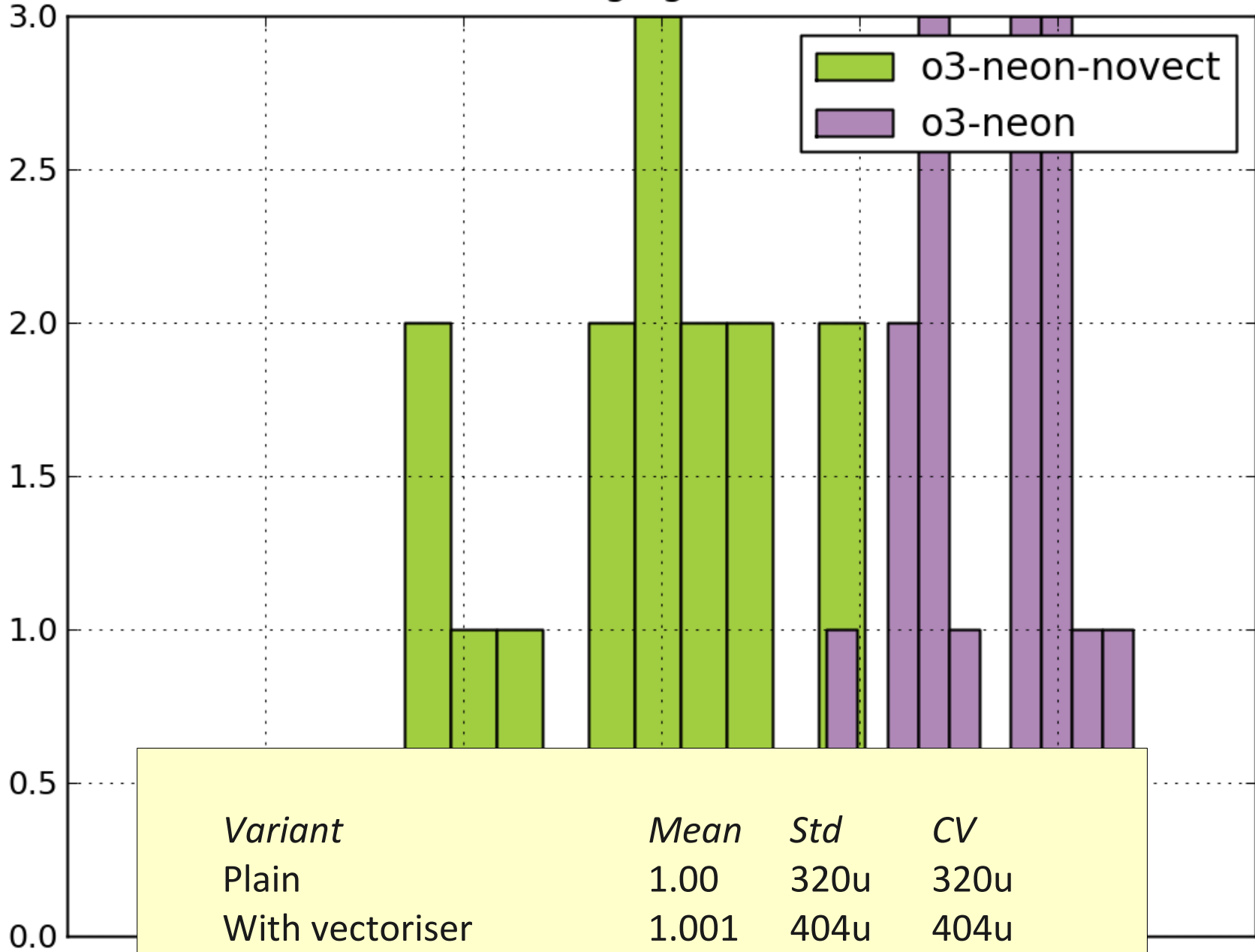
Closer example:  
Effect of modulo scheduling (SMS)

# viterb00



t = 118

# galgel



t = 8.27 - significant

## Our tools

perf

difftest

bettters

tabulate

Python

LibreOffice!

Other statistical tools like scipy.stat, R, Judge, and  
ministat

results.csv - LibreOffice Calc

File Edit View Insert Format Tools Data Window Help

Arial 10

B149  $f(x)$   $\Sigma$  = gcc-4.6.1

	A	B	C	D	E	F	G
1	testnam	version	klas	variant	subname	nsample	min
140	spec2000	gcc-linaro-4.6-2011.11		All	176_gcc	5	5.4042
141	spec2000	gcc-linaro-4.6-2011.11		Top 10	176_gcc	5	5.42
142	spec2000	gcc-linaro-4.6-2011.11		Standard Filter...	176_gcc	5	5.4442
143	spec2000	gcc-linaro-4.6-2011.11		- empty -	176_gcc	5	5.4442
144	spec2000	gcc-linaro-4.6-2011.11		- not empty -	176_gcc	5	5.4462
145	spec2000	gcc-linaro-4.6-2011.11		o2	176_gcc	5	5.4
146	spec2000	gcc-4.6.1		o3	176_gcc	5	5.5171
147	spec2000	gcc-4.6.1		o3-neon	176_gcc	5	5.5227
148	spec2000	gcc-4.6.1		o3-neon-novect	176_gcc	5	5.5346
149	spec2000	gcc-4.6.1		o3-vfpv3	176_gcc	5	5.5491
594	spec2000	gcc-4.6.1		o3-neon	176_gcc	5	5.6
611	spec2000	gcc-linaro-4.6-2011.11		o2	254_gap	5	10.419
647	spec2000	gcc-linaro-4.6-2011.11		o3-neon	254_gap	5	10.4
661	spec2000	gcc-4.6.1		o2	254_gap	5	10.639
706	spec2000	gcc-4.6.1		o3-neon	254_gap	5	10.7
731	spec2000	gcc-linaro-4.6-2011.11		o3-neon-novect	254_gap	5	11.128
732	spec2000	gcc-linaro-4.6-2011.11		o3-vfpv3	254_gap	5	11.281
733	spec2000	gcc-4.6.1		o3-neon-novect	254_gap	5	11.281
733	spec2000	gcc-linaro-4.6-2011.11		o3	254_gap	5	11.285

Digging deeper: using perf

```
michaelh@leo1:~/coremark_v1.0$ perf report -n -d coremark.exe --stdio
```

```
# dso: coremark.exe
```

```
# Events: 15K cycles
```

```
#
```

```
# Overhead  Samples          Command          Symbol
```

```
# .....  .....
```

```
#
```

37.96%	5980	coremark.exe	[.] core_state_transition
27.21%	4197	coremark.exe	[.] core_bench_list
16.25%	2522	coremark.exe	[.] matrix_test
7.04%	1110	coremark.exe	[.] crcu32
6.50%	931	coremark.exe	[.] crc16
2.42%	396	coremark.exe	[.] core_bench_state
1.70%	279	coremark.exe	[.] crcu16
0.87%	143	coremark.exe	[.] calc_func
0.04%	6	coremark.exe	[.] core_bench_matrix
0.02%	3	coremark.exe	[.] main

```
#
```

```
# (For a higher level overview, try: perf report --sort comm,dso)
```

```
#
```

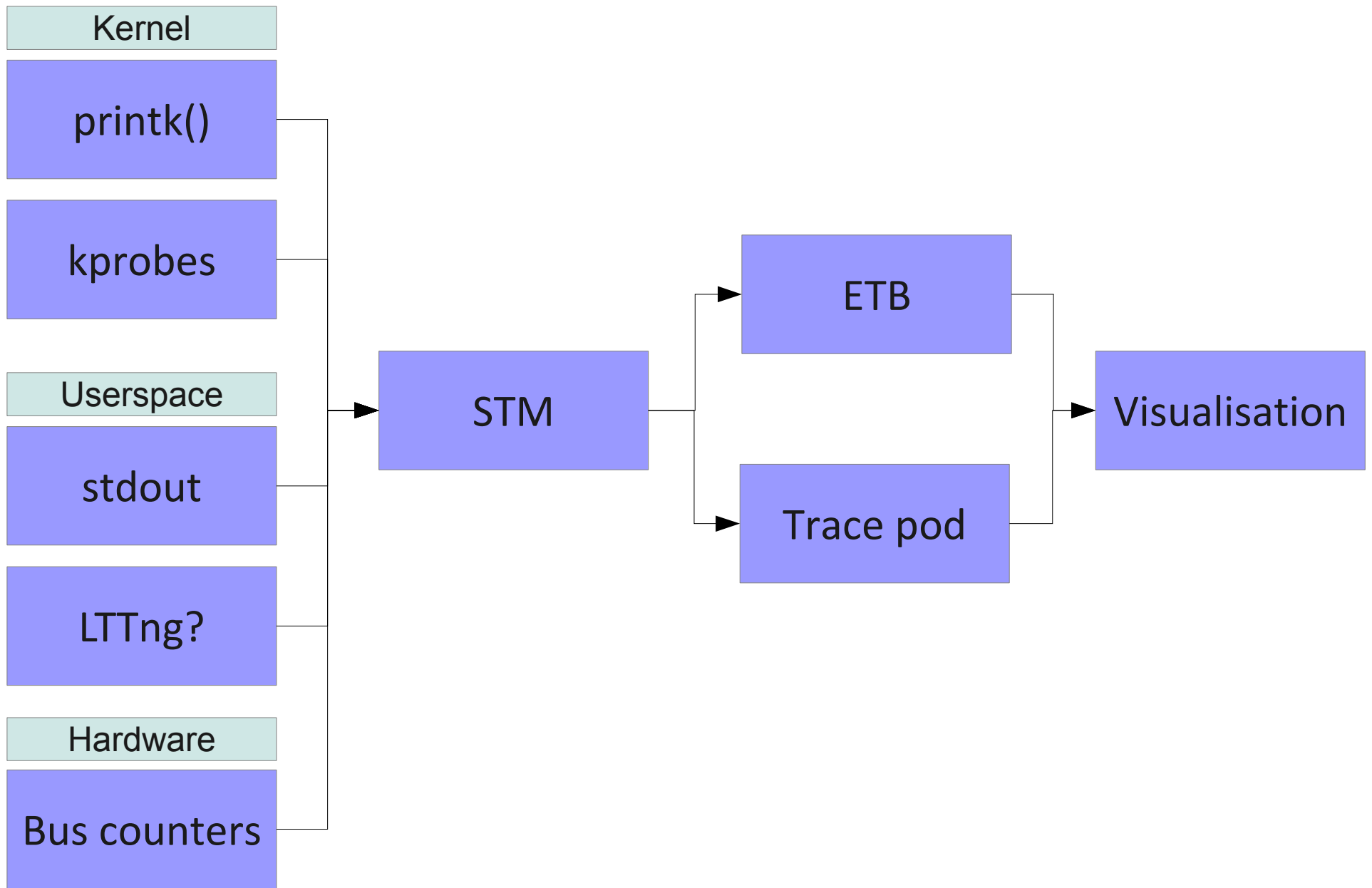
# Future

Open benchmarks

Scientific benchmarks

Locking down the environment

Future: System level trace





[www.linaro.org](http://www.linaro.org) / [wiki.linaro.org](http://wiki.linaro.org)

[people.linaro.org/~michaelh/presentations](http://people.linaro.org/~michaelh/presentations)