Linux Integrated System Analysis

Tooling for Scheduler and PM

ARM

Brendan Jackman and Patrick Bellasi
brendan.jackman@arm.com patrick.bellasi@arm.com

OSPM Summit, Scuola Superiore Sant'Anna Pisa
April 2017

ARM

Agenda

- Introduction
  What is LISA
- Example plots and analysis
  some of the standard plots we generate
- Hands on examples
  walking-through to some simple usage examples
- Discussion
  feedbacks collection

Introduction - Goals & Motivation

*A **toolkit** to support **interactive analysis***

- Support **study of existing behaviour**

e.g. *how does this PELT thing work?*

- Support **analysis of new features**

How does this tweak to __update_load_avg change the PELT signal?

What impact does that have on schedutil's response time?

- Help get insights into **what's not working and why**
- Provide a **common language** for **reproducible** experiments & analysis
- Framework for **automated testing** of kernel behaviours

Did I break frequency invariant load tracking when I rebased those topology.c patches?

Introduction - What is LISA?

Linux Integrated System Analysis

- Python-based framework for analysis of kernel behaviour
  https://github.com/ARM-software/lisa (Apache 2.0 license)

- Brings together other frameworks & tools
  - Interacting with (Linux/Android/Localhost) target devices (devlib)
  - Describing & running synthetic workloads (rt-app)
    while collecting (ftrace/systrace) traces and energy samples (HWMON, AEP, ACME Cape, Monsoon, ...)
  - Parsing, analysing, asserting on trace data (Pandas, TRAPpy, BART)
  - Plotting & interactive analysis (Jupyter, a.k.a IPython Notebooks)

- Provide tests to verify regressions on scheduler behaviors

Introduction - Fundamental Idea: Data Analysis on Trace Events

- From collected (ftrace/systrace) trace files…

```
trace-cmd-2204  [000]  1773.509207:  sched_load_avg_task:  comm=trace-cmd pid=2204 cpu=0 load_avg=452 util_avg=176 util_est=176 load_sum=21607277 util_sum=8446887 period_contrib=125
trace-cmd-2204  [000]  1773.509223:  sched_load_avg_task:  comm=trace-cmd pid=2204 cpu=0 load_avg=452 util_avg=176 util_est=176 load_sum=21607277 util_sum=8446887 period_contrib=125
   <idle>-0     [002]  1773.509522:  sched_load_avg_task:  comm=sudo pid=2203 cpu=2 load_avg=0 util_avg=0 util_est=941 load_sum=7 util_sum=7 period_contrib=576
    sudo-2203   [002]  1773.511197:  sched_load_avg_task:  comm=sudo pid=2203 cpu=2 load_avg=14 util_avg=14 util_est=941 load_sum=688425 util_sum=688425 period_contrib=219
     udo-2203   [002]  1773.511219:  sched_load_avg_task:  comm=sudo pid=2203 cpu=2 load_avg=14 util_avg=14 util_est=14 load_sum=688425 util_sum=688425 period_contrib=219
```

- … to PANDAS::DataFrames

```
# Load the LISA::Trace parsing module
from trace import Trace

# Define which event we are interested into
trace = Trace(platform, trace_file, [
            "sched_switch",
            "sched_load_avg_cpu",
            "sched_load_avg_task",
            "sched_boost_cpu",
            "sched_boost_task",
            "cpu_frequency",
            "cpu_capacity",
    ])
```

```
# Trace events are converted into tables, let's have a look at one
# of such tables
load_df = trace.data_frame.trace_event('sched_load_avg_task')
load_df.head()
```

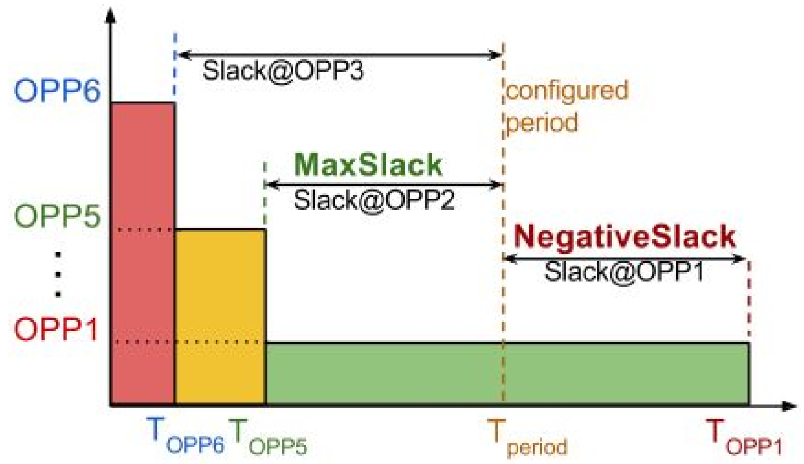
	__comm	__cpu	__pid	comm	cpu	load_avg	load_sum	period_contrib	pid	util_avg	util_est		
Time													
0.000065	trace-cmd	0	2204	trace-cmd	0	452	21607277	125	2204	176	176		
0.000081	trace-cmd	0	2204	trace-cmd	0	452	21607277	125	2204	176	176	8446887	LI
0.000380	<idle>	2	0	sudo	2	0	7	576	2203	0	941	7	bi
0.002055	sudo	2	2203	sudo	2	14	688425	219	2203	14	941	688425	bi
0.002077	sudo	2	2203	sudo	2	14	688425	219	2203	14	14	688425	bi

ARM

Example - RTApp Task Performances

https://github.com/ARM-software/lisa/blob/master/ipynb/examples/wlgen/rtapp_example.ipynb

RTApp Performance Index

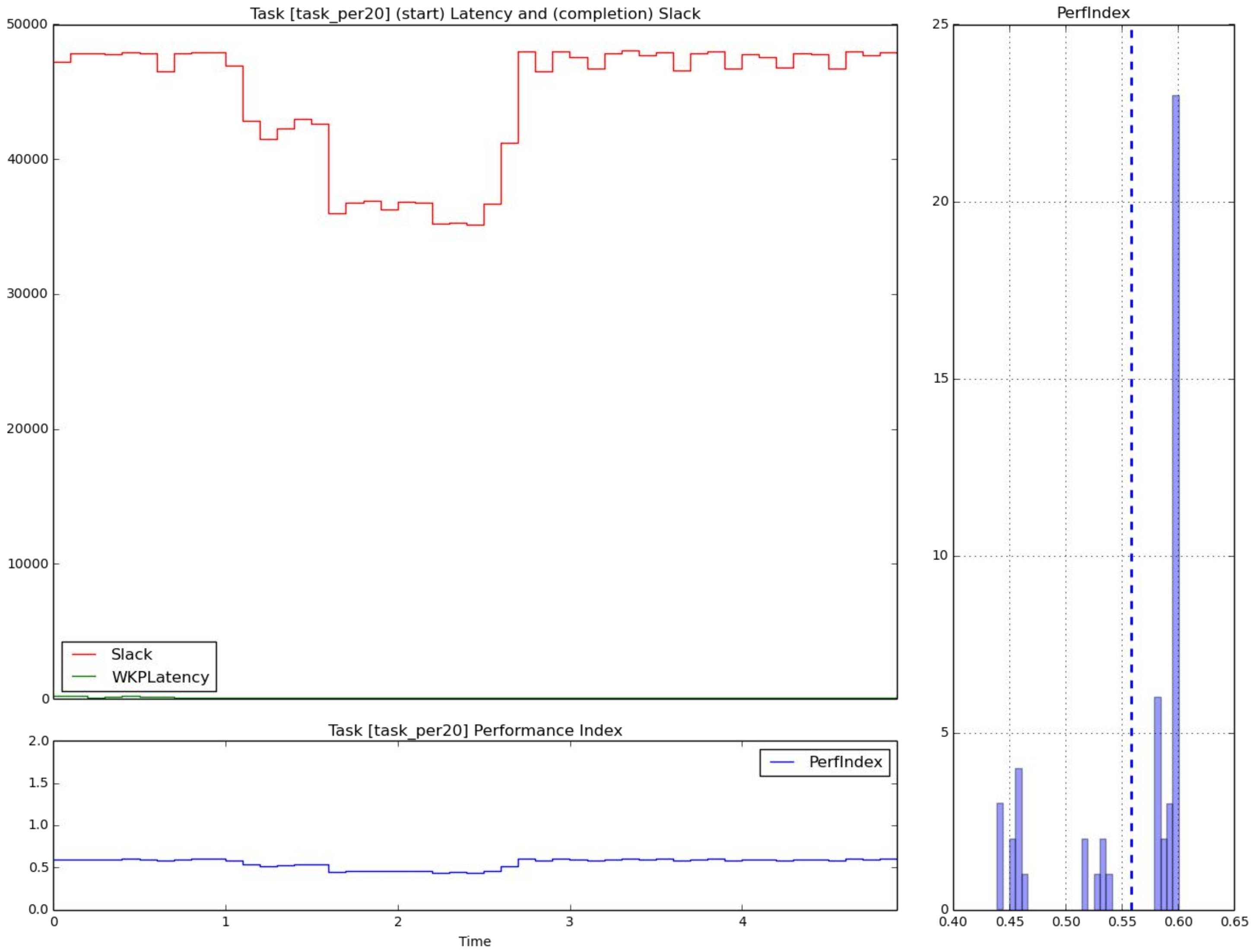


$$MaxSlack = Period_{conf} - RunTime_{conf}$$

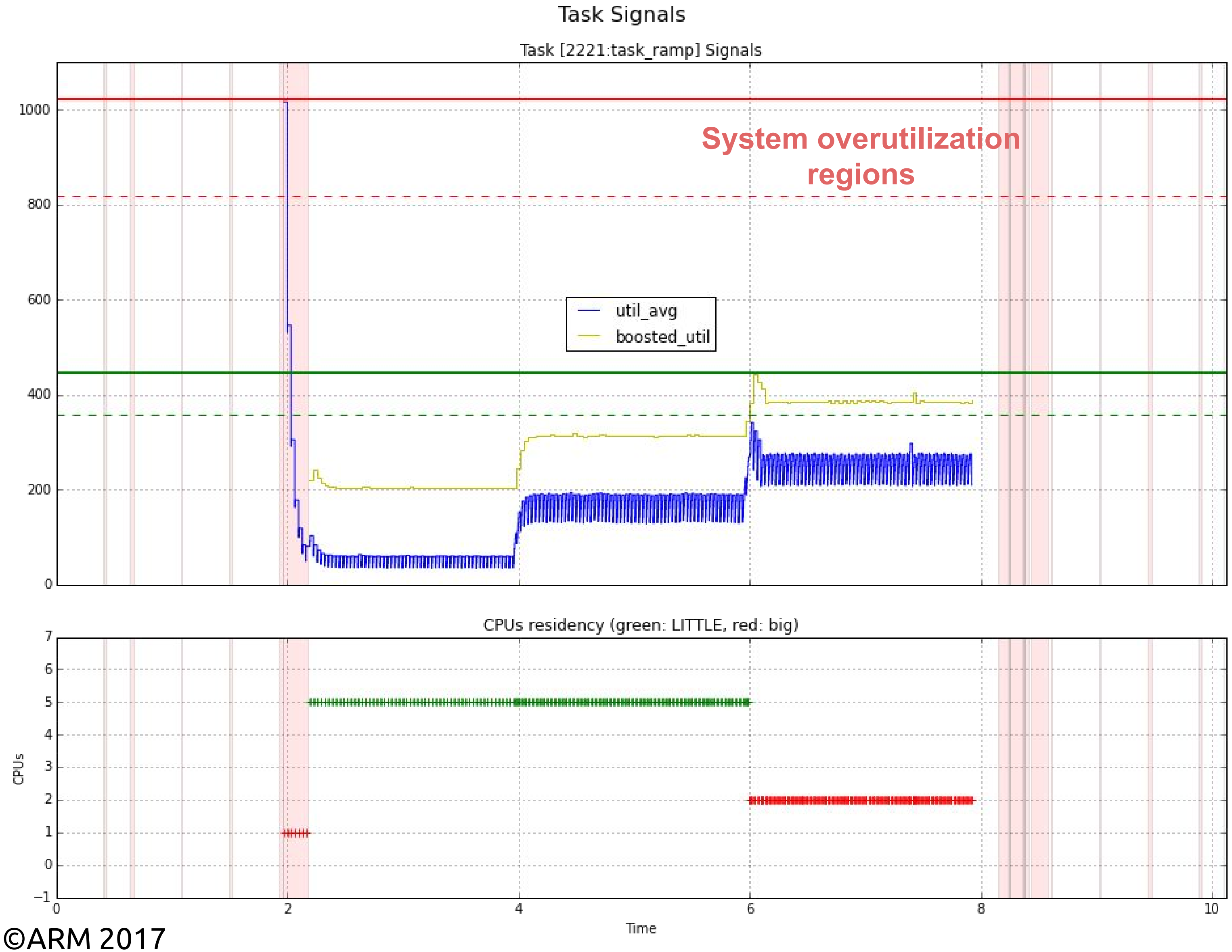
$$PerfIndex = \frac{Period_{conf} - RunTime_{meas}}{MaxSlack}$$

$$NegSlack_{percent} = \frac{\sum Max(0, RunTime_{meas} - Period_{conf})}{\sum RunTime_{meas}}$$

Performance plots for task [task_per20]



Example - Task Signals



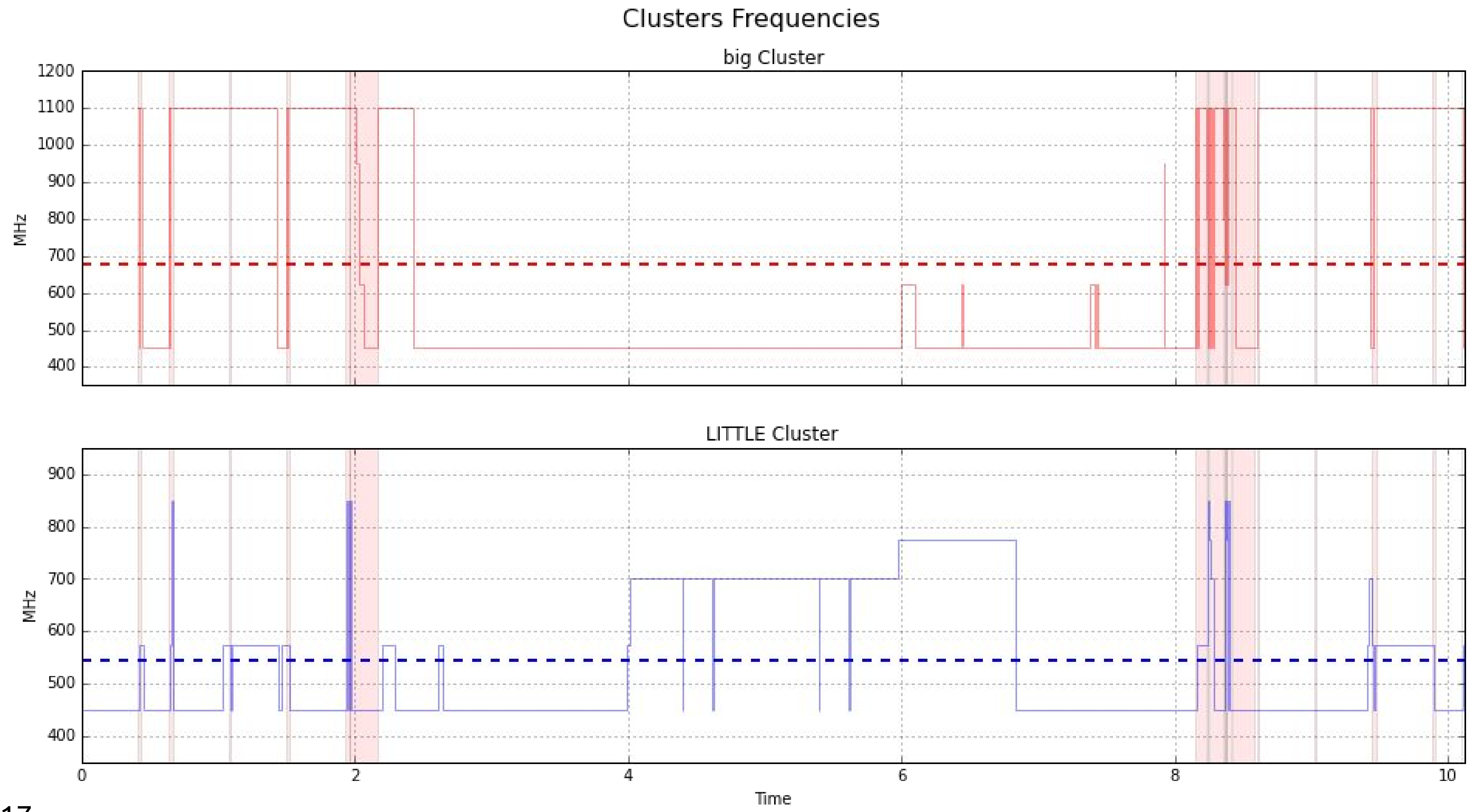
big Capacity and tipping point

LITTLE Capacity and tipping point

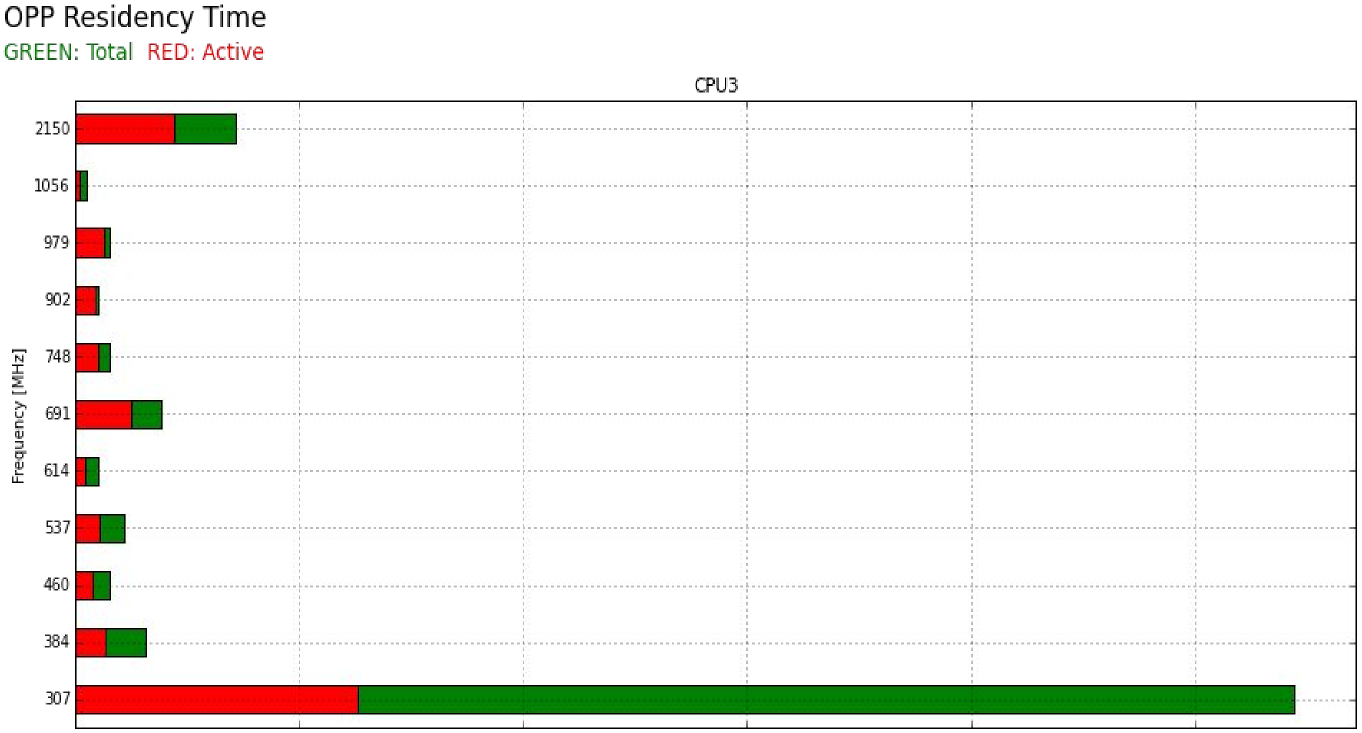
CPUs swim-lines: task residency
Color: LITTLE vs big CPUs

ARM

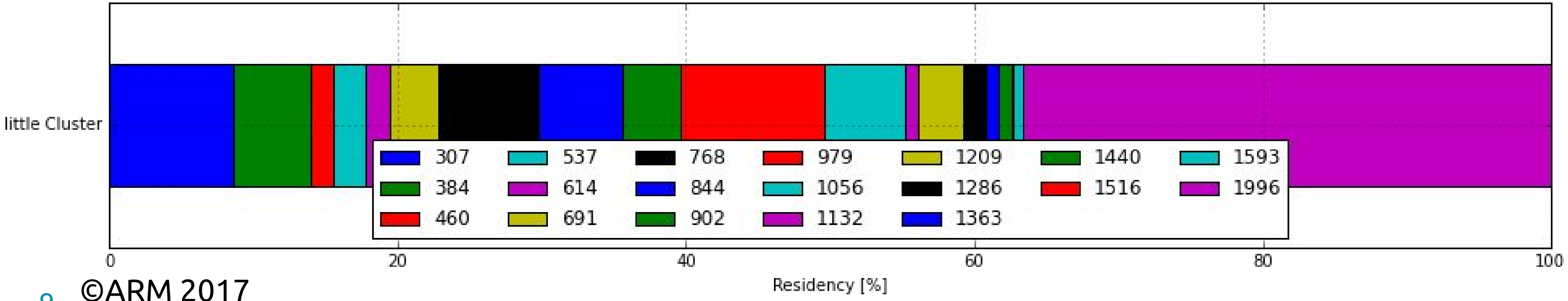
Example - CPU Frequencies over Time



Example - CPU Frequencies Analysis



Active and Total Time spent on each OPP both in absolute and percentage

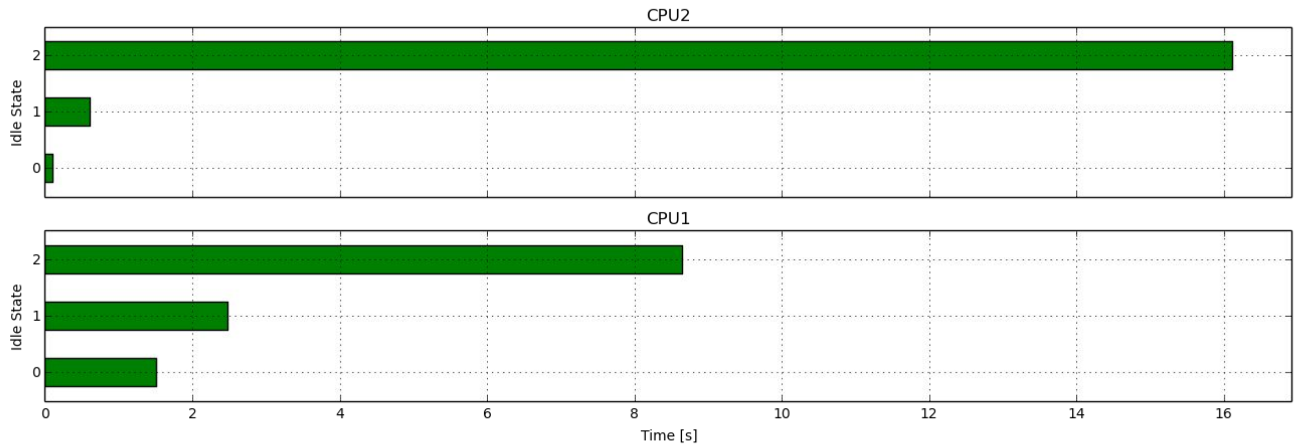


ARM

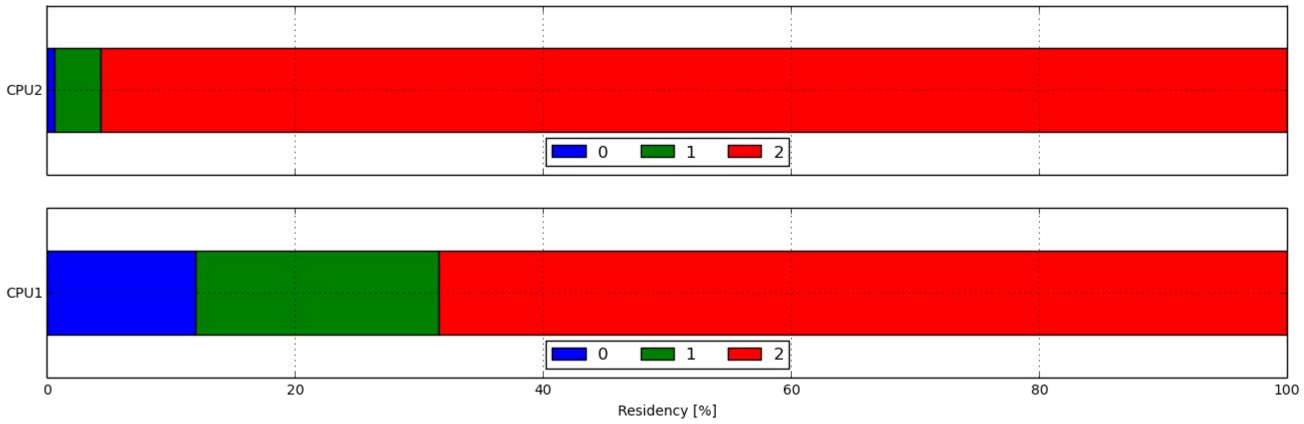
Example - Idle States Analysis

https://github.com/ARM-software/lisa/blob/master/ipynb/examples/trace_analysis/TraceAnalysis_IdleStates.ipynb

Idle State Residency Time



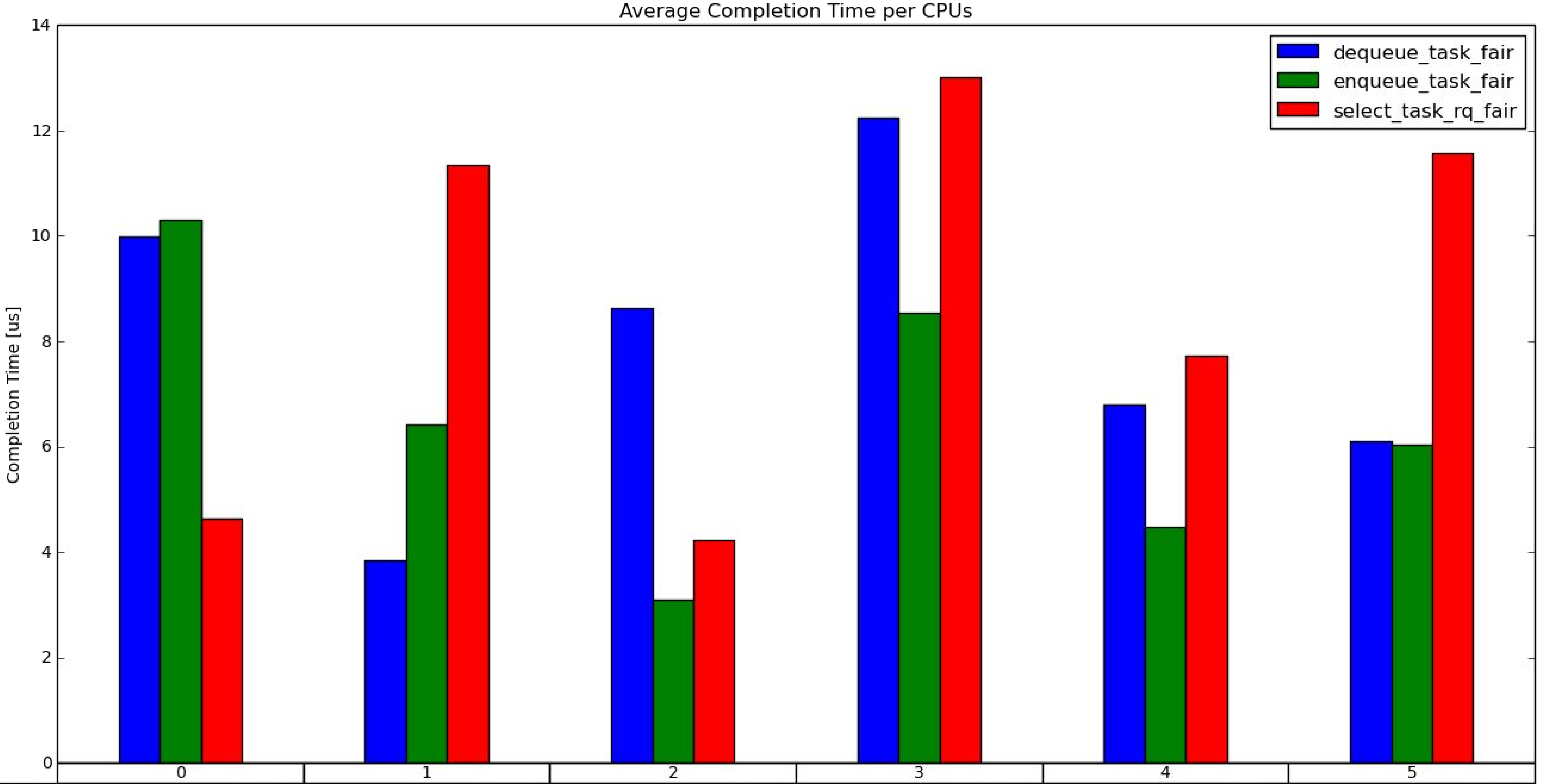
Idle State Residency Time



ARM

Example - Functions Profiling

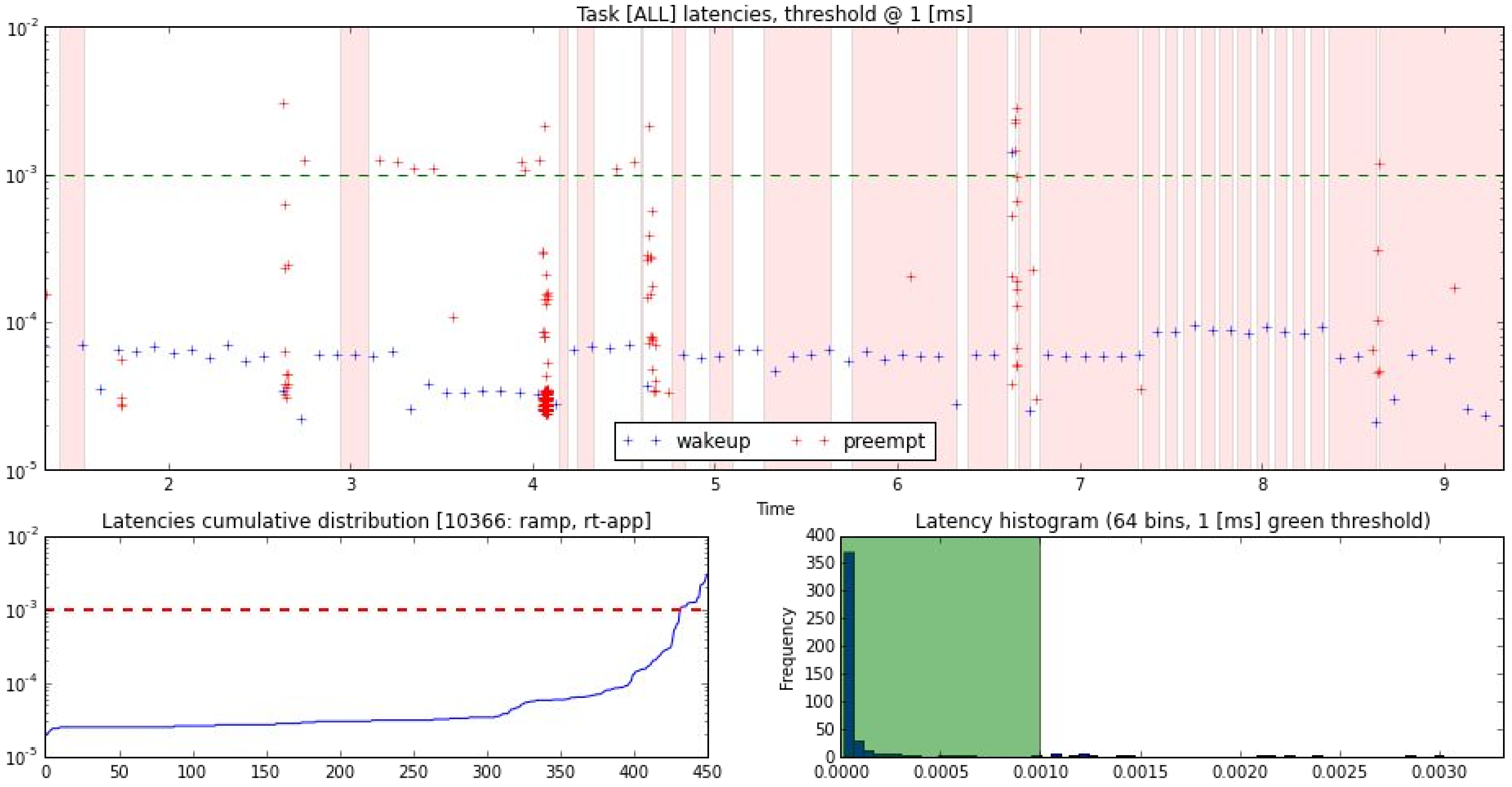
https://github.com/ARM-software/lisa/blob/master/ipynb/examples/trace_analysis/TraceAnalysis_FunctionsProfiling.ipynb



	0	1	2	3	4	5
dequeue_task_fair	9.994	3.857	8.643	12.253	6.805	6.097
enqueue_task_fair	10.302	6.415	3.091	8.536	4.474	6.029
select_task_rq_fair	4.641	11.346	4.223	13.006	7.731	11.571

Example - Latency Analysis

https://github.com/ARM-software/lisa/blob/master/ipynb/examples/trace_analysis/TraceAnalysis_TasksLatencies.ipynb



ARM

Hands-on Examples

- Run a benchmark on an Android device, plot wakeup latency of graphics tasks
- Configure and run a simple RT-App RAMP workload
  - visualize trace using trappy and plot performance index and frequencies
  - plot PELT signal and CPU capacity on same axes
- Figure out the periodicity of interrupts by analysing traces
- In a CI-friendly way
  - describe a synthetic workload
  - analyse target platform to figure out a desired task placement for workload
  - run workload, collect ftrace, collect energy readings
  - analyse trace to see if task placement matched

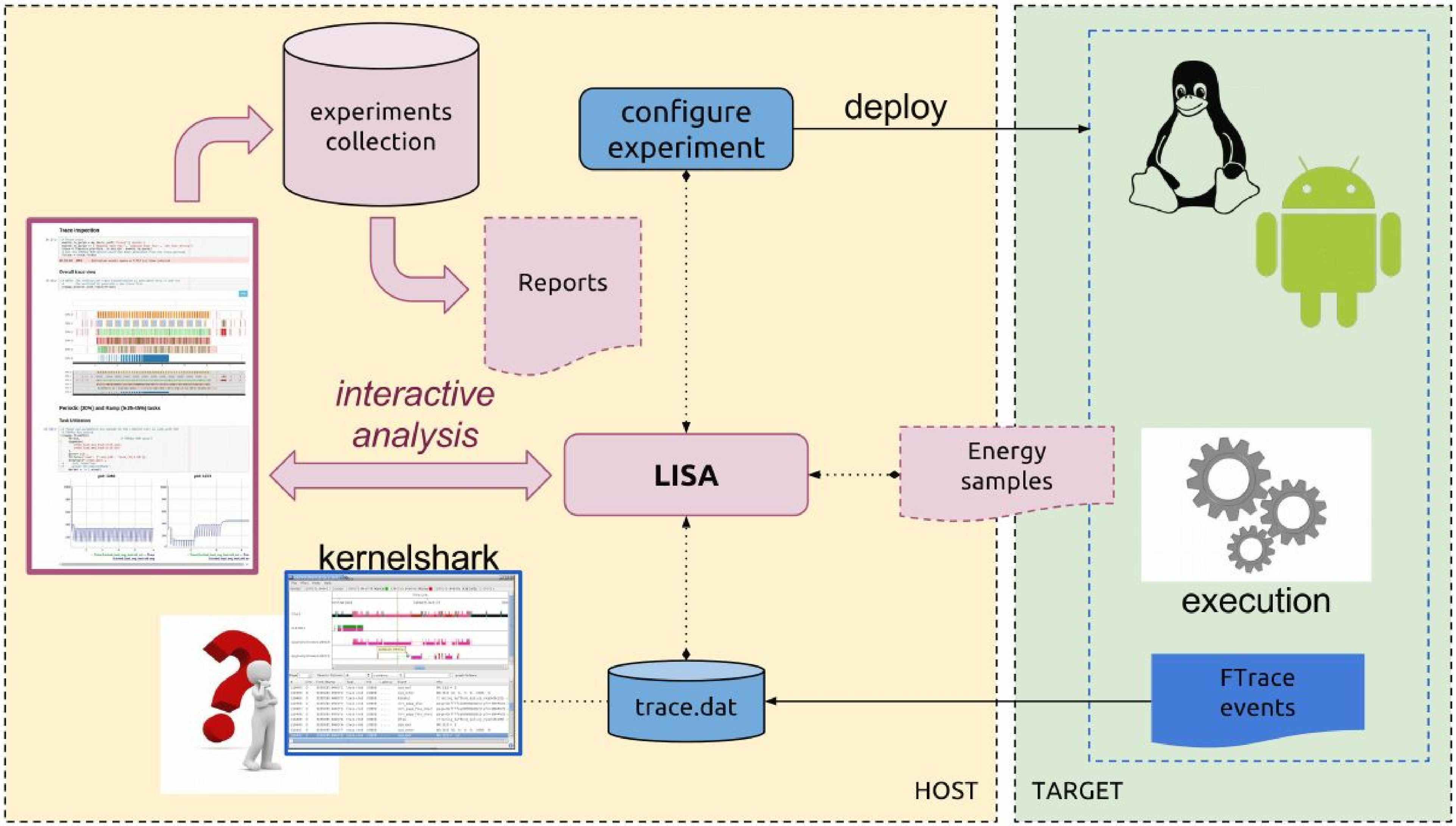
ARM

Discussion

- Overall comments and feedbacks?
- Is such a framework useful?
- How can be improved?
- Which kind of features would you like to make it more useful?

Backup Slides

Introduction: analysis workflow



Classical flow vs **LISA flow**

ARM