PyConChina 2019

PyCon China
Hangzhou 2019

Python for Linux Kernel Debugging

Feng Li (李枫)
hkli2013@126.com
Oct 19，2019

PYTHON 30th

Agenda

I. Linux Kernel Debugging
- Overview
- eBPF
- Development

II. BCC
- Overview
- Development

III. LISA
- Overview
- Internals

IV. Drgn
- Overview

V. Practice on ARM
- Development Environment
- My Practice

VI. Wrap-up

I. Linux Kernel Debugging

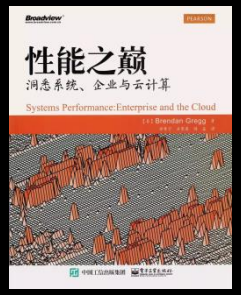
1) Overview

DTrace

- https://en.wikipedia.org/wiki/DTrace
  a comprehensive dynamic tracing framework created by Sun Microsystems for troubleshooting kernel and application problems on production systems in real time. Originally developed for Solaris, it has since been released under the free CDDL in OpenSolaris and its descendant illumos…
- https://www.oracle.com/technetwork/server-storage/solaris11/technologies/dtrace-1930301.html
- https://docs.oracle.com/en/operating-systems/oracle-linux/dtrace-guide/

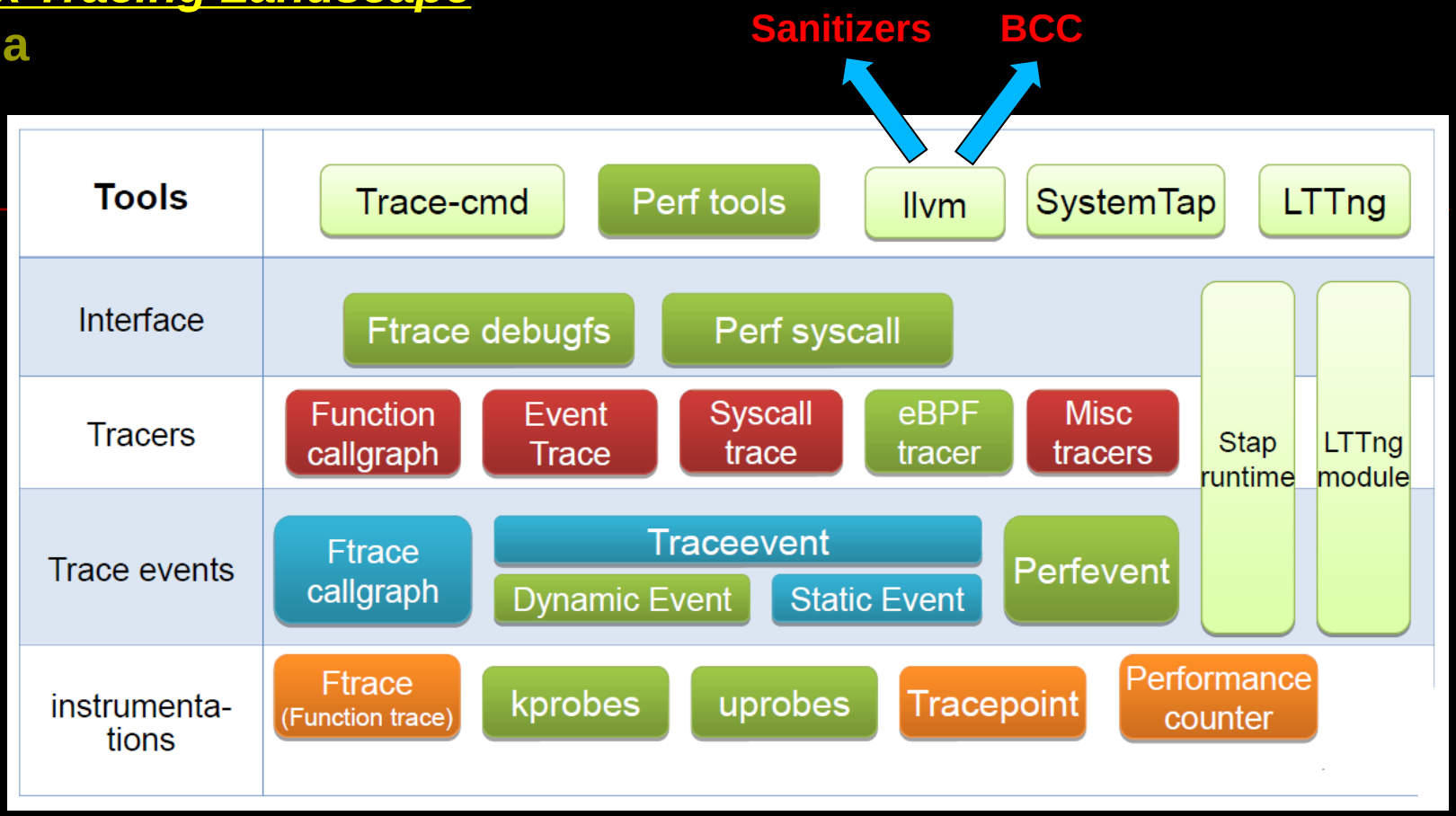
…

- http://www.brendangregg.com/dtrace.html



Linux Tracing Landscape

- a

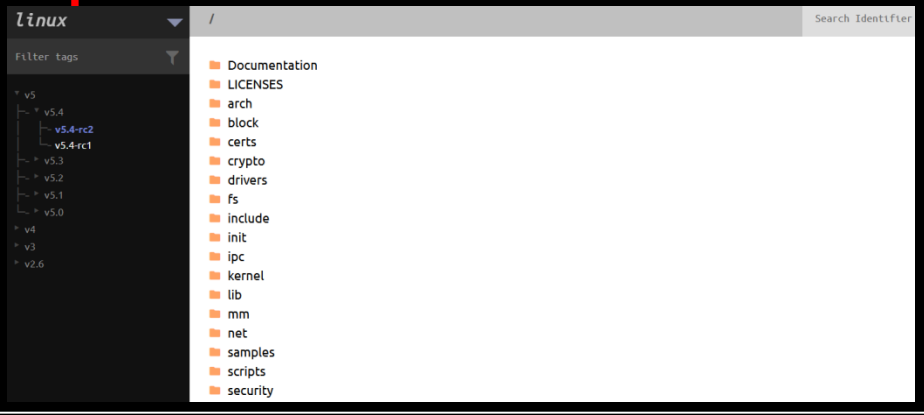


Source: “Dynamic Probes for Linux”, Masami Hiramatsu, Tracing Summit 2015

- $KERNEL_SRC/Documentation/kbuild/modules.rst
- $KERNEL_SRC/Documentation/trace/events.rst
- $KERNEL_SRC/Documentation/trace/ftrace*.rst
- $KERNEL_SRC/Documentation/kprobes
- $KERNEL_SRC/Documentation/trace/kprobetrace.rst
- $KERNEL_SRC/Documentation/trace/uprobetracer.rst
- $KERNEL_SRC/Documentation/trace/tracepoints*.rst
- $KERNEL_SRC/Documentation/trace/features/*.*

Browsing Kernel Code Online

- https://www.kernel.org/
- https://github.com/torvalds/linux
- https://elixir.bootlin.com/linux/latest/source



2) eBPF

2.1 BPF (Berkeley Packet Filter, aka cBPF)

- https://en.wikipedia.org/wiki/Berkeley_Packet_Filter
- http://www.tcpdump.org/papers/bpf-usenix93.pdf
- History
  - Before BPF, each OS (Sun, DEC, SGI etc) had its own packet filtering API
  - In 1993: Steven McCanne & Van Jacobsen released a paper titled the *BSD Packet Filter (BPF)*
  - Implemented as “Linux Socket Filter” in kernel 2.2
  - While maintaining the BPF language (for describing filters), uses a different internal architecture

Source: ebpfbasics-190611051559.pdf

What is it

- Network packet filtering, Seccomp
- Filter Expressions → Bytecode → Interpret
- Small, in-kernel VM, Register based, switch dispatch interpreter, few instructions
- BPF uses a simple, non-shared buffer model made possible by today's larger address space

Source: ebpfbasics-190611051559.pdf

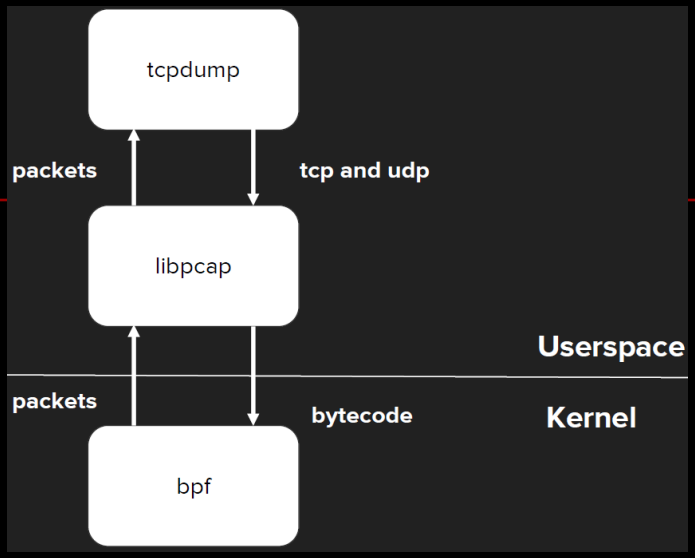
- Bytecode, register based VM, with a limited instruction set
- Runs in-kernel, designed for fast packet filtering
- 32-bit instructions (LOAD, STORE, ALU, BRANCH, RETURN)
- 2, 32-bit registers (A, X), hidden frame pointer

Source: understandingebpfinahurry-190611040804.pdf

https://blog.cloudflare.com/bpf-the-forgotten-bytecode/

...

tcpdump



Source: understandingebpfinahurry-190611040804.pdf

```
# tcpdump host 127.0.0.1 and port 22 -d
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 18
(002) ld       [26]
(003) jeq      #0x7f000001      jt 6    jf 4
(004) ld       [30]
(005) jeq      #0x7f000001      jt 6    jf 18
(006) ldb      [23]
(007) jeq      #0x84            jt 10   jf 8
(008) jeq      #0x6             jt 10   jf 9
(009) jeq      #0x11            jt 10   jf 18
(010) ldh      [20]
(011) jset     #0x1fff          jt 18   jf 12
(012) ldxb     4*([14]&0xf)
(013) ldh      [x + 14]
[...]
```

Optimizes packet filter performance

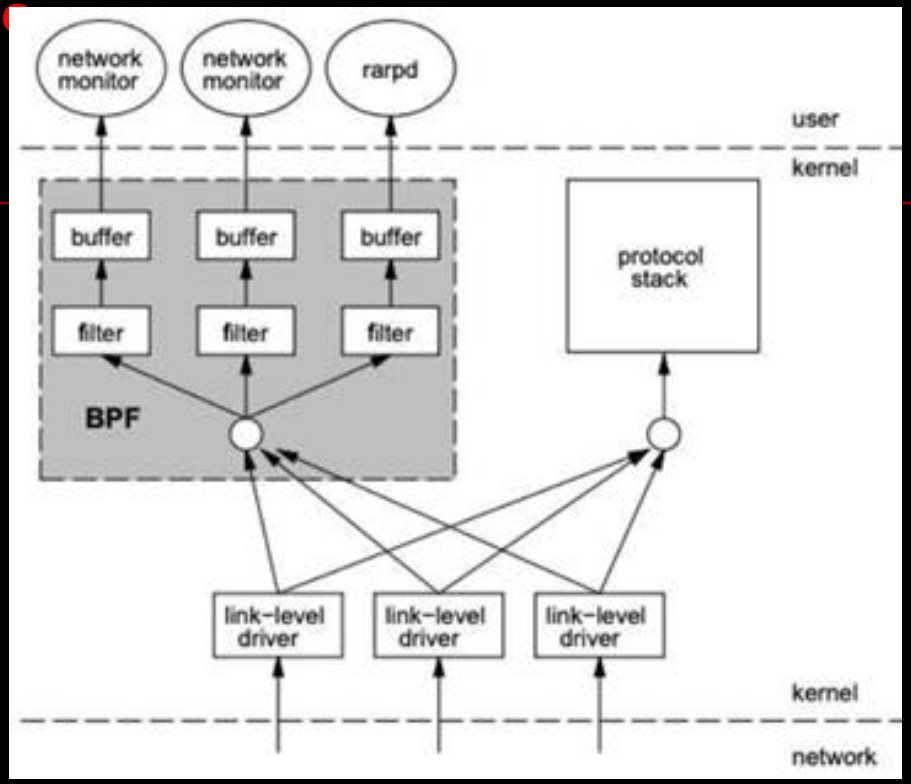
2 x 32-bit registers & scratch memory

User-defined bytecode executed by an in-kernel sandboxed virtual machine

Steven McCanne and Van Jacobson, 1993

Source: https://www.slideshare.net/brendangregg/kernel-recipes-2017-performance-analysis-with-bpf

Workflow



- Utilizes sockets for passing/receiving packets to/from the kernel-space
- Filters are attached with the setsockopt(2) system call
- Create a special-purpose socket (i.e., PF_PACKET) 2
- Attach a BPF program to the socket using the setsockopt(2) system call
- Set the network interface to promiscuous mode with ioctl(2) (optionally)
- Set various BPF parameters, (e.g. buffer size, attach some BPF filters) This is done using the ioctl(2) system call
- Read packets from the kernel, or send raw packets, by reading/writing to the corresponding file descriptor of /dev/bpf using read(2)/write(2) system calls
- Read packets from the kernel, or send raw packets, by reading/writing to the file descriptor of the socket using recvfrom(2)/sendto(2) system calls

Source: ebpfbasics-190611051559.pdf

2.2 eBPF (extended BPF)

- since Linux Kernel v3.15 and ongoing
- aims at being a universal in-kernel virtual machine
- a simple way to extend the functionality of Kernel at runtime
- “dtrace for Linux”

BPF Features by Linux Kernel Version

- https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md

Instruction Set

- https://github.com/iovisor/bpf-docs/blob/master/eBPF.md

Projects using eBPF

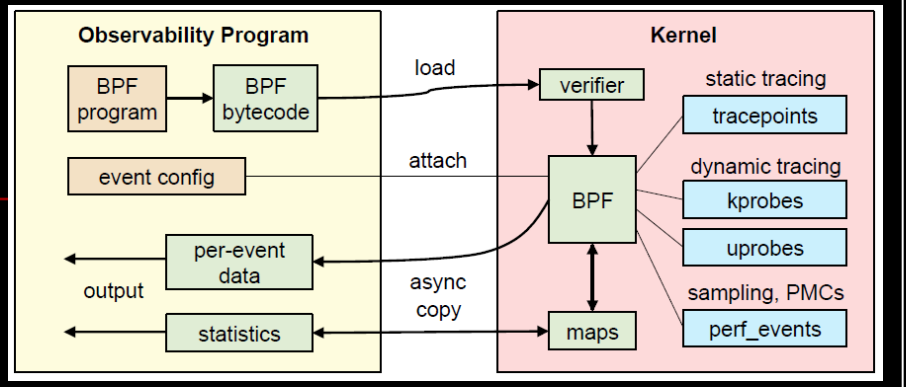
- http://cilium.readthedocs.io/en/latest/bpf/#projects-using-bpf

Good Resource

- https://github.com/zoidbergwill/awesome-ebpf

What is it

- User-defined, sandboxed bytecode executed by the kernel
- VM that implements a RISC-like assembly language in kernel space



- Similar to LSF, but with the following improvements:
  - More registers, JIT compiler (flexible/ faster), verifier
  - Attach on Tracepoint, Kprobe, Uprobe, USDT
  - In-kernel trace aggregation & filtering
  - Control via bpf()
  - Designed for general event processing within the kernel
  - All interactions between kernel/ user space are done through eBPF “maps”

Source: ebpfbasics-190611051559.pdf
Source: https://kernel-recipes.org/en/2017/talks/performance-analysis-with-bpf/

bpf() system call

- http://www.man7.org/linux/man-pages/man2/bpf.2.html

NAME top

bpf - perform a command on an extended BPF map or program

SYNOPSIS top

```
#include <linux/bpf.h>

int bpf(int cmd, union bpf_attr *attr, unsigned int size);
```

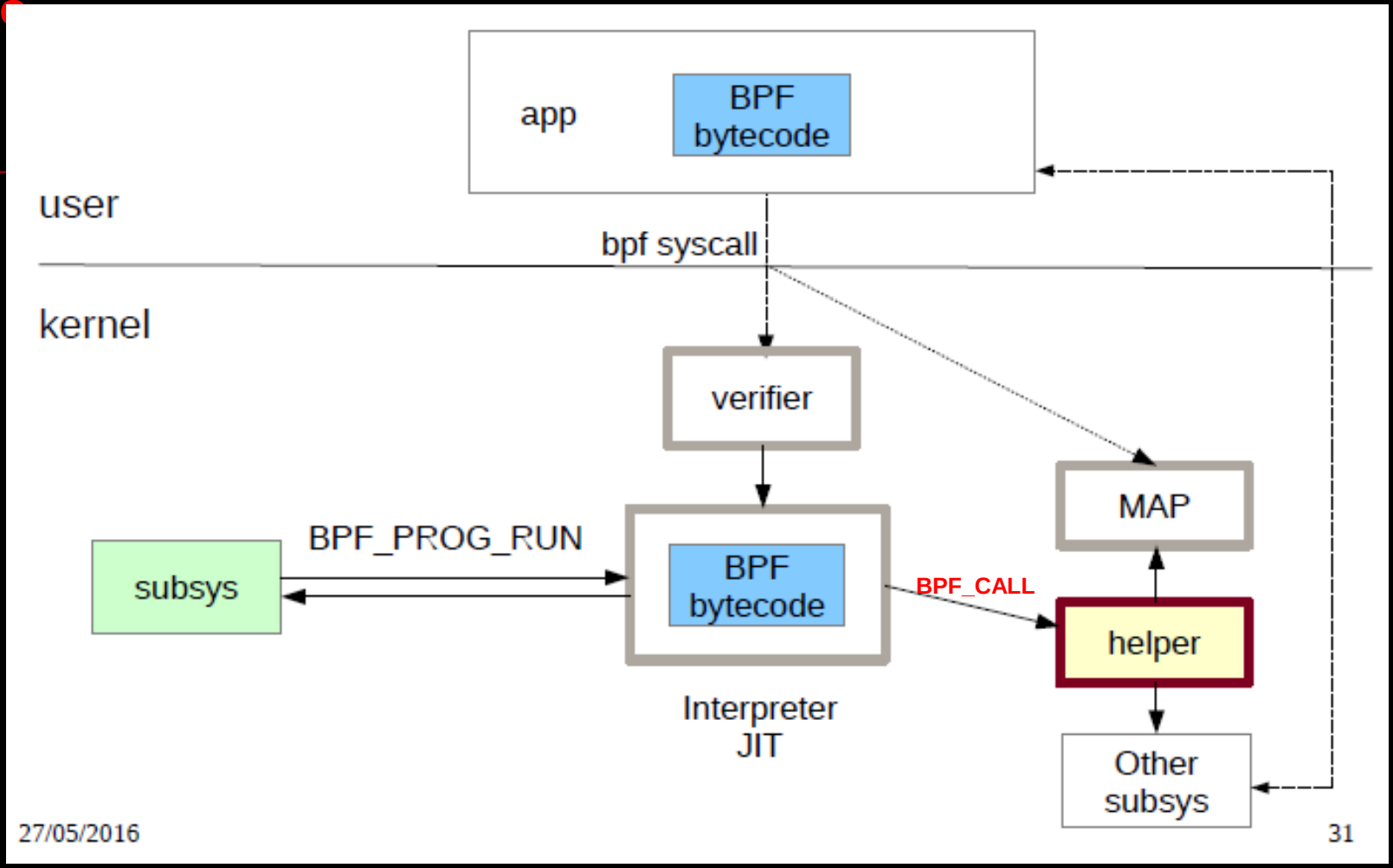
DESCRIPTION top

The **bpf**() system call performs a range of operations related to extended Berkeley Packet Filters. Extended BPF (or eBPF) is similar to the original ("classic") BPF (cBPF) used to filter network packets. For both cBPF and eBPF programs, the kernel statically analyzes the programs before loading them, in order to ensure that they cannot harm the running system.

eBPF extends cBPF in multiple ways, including the ability to call a fixed set of in-kernel helper functions (via the **BPF_CALL** opcode extension provided by eBPF) and access shared data structures such as eBPF maps.

...

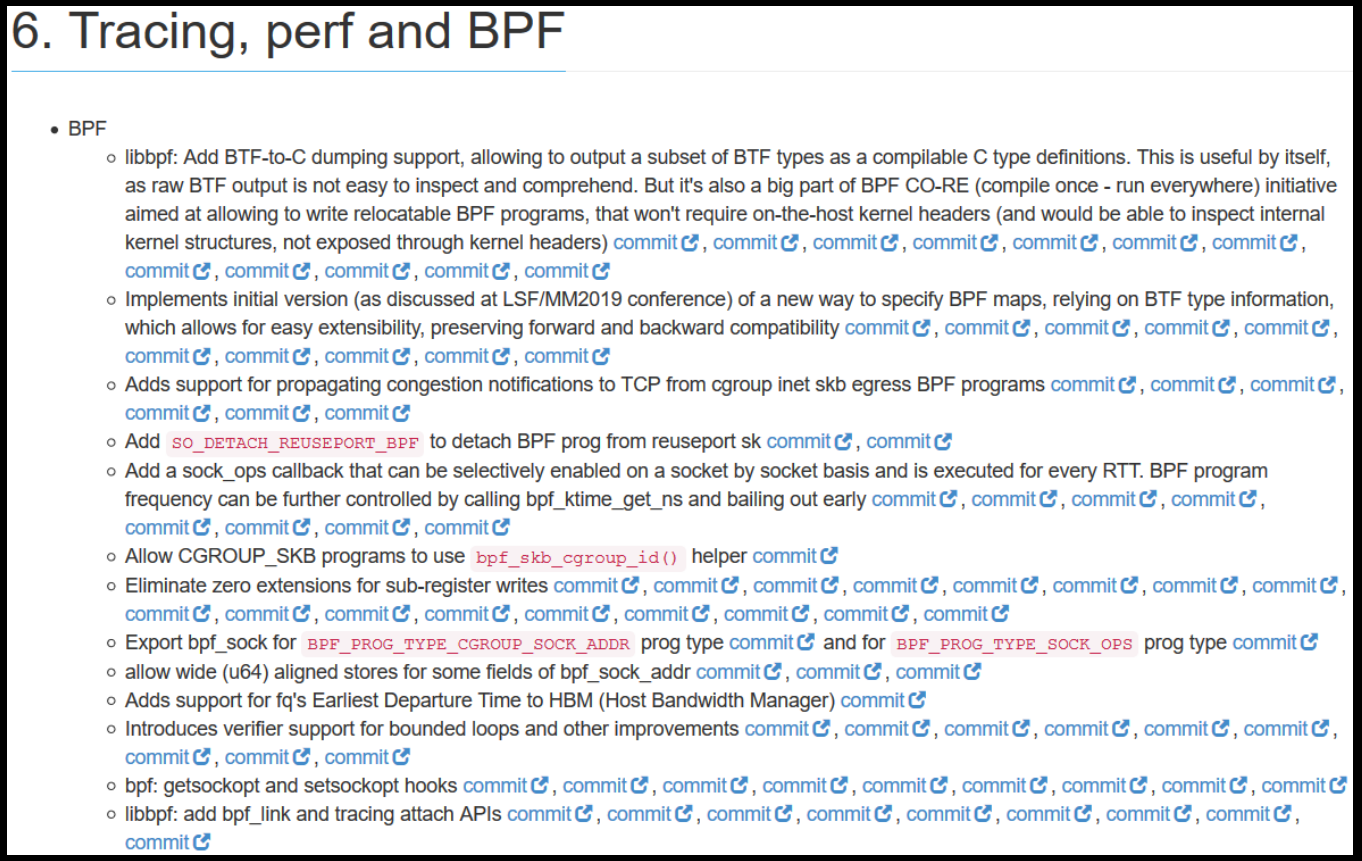
Workflow



Source: http://www.slideshare.net/vh21/meet-cutebetweenebpfandtracing

Internals

- **Pls refer to my presentation "eBPF in Action" at LC3 Beijing (on Jun 25, 2018)**
- **$KERNEL_SRC/Documentation/networking/filter.txt**
- **https://kernelnewbies.org/Linux_5.3**

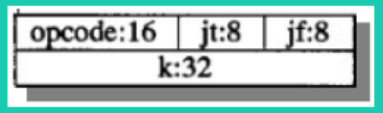
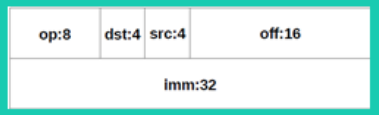


6. Tracing, perf and BPF

- BPF
  - libbpf: Add BTF-to-C dumping support, allowing to output a subset of BTF types as a compilable C type definitions. This is useful by itself, as raw BTF output is not easy to inspect and comprehend. But it's also a big part of BPF CO-RE (compile once - run everywhere) initiative aimed at allowing to write relocatable BPF programs, that won't require on-the-host kernel headers (and would be able to inspect internal kernel structures, not exposed through kernel headers) commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit
  - Implements initial version (as discussed at LSF/MM2019 conference) of a new way to specify BPF maps, relying on BTF type information, which allows for easy extensibility, preserving forward and backward compatibility commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit
  - Adds support for propagating congestion notifications to TCP from cgroup inet skb egress BPF programs commit, commit, commit, commit, commit, commit
  - Add `SO_DETACH_REUSEPORT_BPF` to detach BPF prog from reuseport sk commit, commit
  - Add a sock_ops callback that can be selectively enabled on a socket by socket basis and is executed for every RTT. BPF program frequency can be further controlled by calling bpf_ktime_get_ns and bailing out early commit, commit, commit, commit, commit, commit, commit, commit, commit
  - Allow CGROUP_SKB programs to use `bpf_skb_cgroup_id()` helper commit
  - Eliminate zero extensions for sub-register writes commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit
  - Export bpf_sock for `BPF_PROG_TYPE_CGROUP_SOCK_ADDR` prog type commit and for `BPF_PROG_TYPE_SOCK_OPS` prog type commit
  - allow wide (u64) aligned stores for some fields of bpf_sock_addr commit, commit, commit
  - Adds support for fq's Earliest Departure Time to HBM (Host Bandwidth Manager) commit
  - Introduces verifier support for bounded loops and other improvements commit, commit, commit, commit, commit, commit, commit, commit, commit, commit
  - bpf: getsockopt and setsockopt hooks commit, commit, commit, commit, commit, commit, commit, commit, commit, commit
  - libbpf: add bpf_link and tracing attach APIs commit, commit, commit, commit, commit, commit, commit, commit, commit, commit

- **...**

Comparison

	cBPF	eBPF
Register	**Two 32 bit registers:** A: accumulator X: indexing	**Eleven 64 bit registers:** R0: return value/exit value R1-R5: arguments R6-R9: callee saved registers R10: read-only frame pointer
Instruction	~30 	~90 
JIT	Support	Support (better mapping with newer architectures for JITing)
Toolchain	GCC, tools/net	LLVM eBPF backend
Platform	x86_64, ARM, ARM64, SPARC, PowerPC, MIPS and s390	x86-64, aarch64, s390x...

System Call		`#include <linux/bpf.h>`
		`int bpf(int cmd, union bpf_attr *attr, unsigned int size);`
		(CALL, MAP, LOAD...)

Application	tcpdump... apply for seccomp filters, traffic control…	DDoS Mitigation, Intrusion Detection, Container Security, SDN Configuration, Observability...

3) Development
3.1 Toolchain
LLVM

- eBPF backend firstly introduced in LLVM 3.7 release
- http://llvm.org/docs/CodeGenerator.html#the-extended-berkeley-packet-filter-ebpf-backend
- (
  - Enabled by default with all major distributions
    - Registered targets: `llc --version`
    - `llc`'s BPF `-march` options: bpf, bpfeb, bpfel
    - `llc`'s BPF `-mcpu` options: generic, v1, v2, probe
  - Assembler output through `-S` supported
  - `llvm-objdump` for disassembler and code annotations (via DWARF)
  - Annotations correlate directly with kernel verifier log
  - Outputs ELF file with maps as relocation entries
    - Processed by BPF loaders (e.g. iproute2) and pushed into kernel

  Source: https://ossna2017.sched.com/event/BCsg/making-the-kernels-networking-data-path-programmable-with-bpf-and-xdp-daniel-borkmann-covalent
- $LLVM_SRC/lib/Target/BPF
- http://cilium.readthedocs.io/en/latest/bpf/
- LLVM 9.0 Released With Ability To Build The Linux x86_64

GCC (GCC support for eBPF is on the way)

- https://www.phoronix.com/scan.php?page=news_item&px=GNU-Binutils-eBPF-Support
  //GNU Binutils Begins Landing eBPF Support
- https://www.phoronix.com/scan.php?page=news_item&px=GCC-10-eBPF-Backend-Plans
  //Oracle Is Aiming To Contribute An eBPF Backend To The GCC 10 Compiler
- https://www.phoronix.com/scan.php?page=news_item&px=Oracle-More-DTrace-Linux-eBPF
  //Oracle Is Working To Upstream More Of DTrace To The Linux Kernel & eBPF Implementation
- https://www.phoronix.com/scan.php?page=news_item&px=Oracle-GCC-10-eBPF-V2
  //2019-8-17::Oracle Continues Working On eBPF Support For GCC 10
- https://www.phoronix.com/scan.php?page=news_item&px=GCC-10-eBPF-Port-Lands
  //GCC 10 Lands The eBPF Port For Targeting The Linux In-Kernel VM

Status

- **Phase 1:** add eBPF target to the toolchain
  - `bpf-unknown-none`
  - binutils (upstream since May 2019)
  - GCC (upstream since September 2019)
- **Phase 2:** make the generated programs palatable for the kernel loaders and verifier, and **keep it that way**.
- **Phase 3:** provide development goodies for eBPF developers
  - GNU simulator
  - GDB
  - ...

Source: "eBPF support in the GNU Toolchain", Jose E. Marchesi (Oracle), Linux Plumbers Conference 2019

3.2 Debugging

- https://www.netronome.com/documents/143/UG_Getting_Started_with_eBPF_Offload.pdf
- int bpf(int cmd, union bpf_attr *attr, unsigned int size);

log_level

- 0: No debug output.
- 1: Debug information from the verifier (all instructions).
- 2: More information: add all register states after each instruction.

- llvm-objdump, llvm-mc…

bpftool

- $KERNEL_SRC/tools/bpf /bpftool

```
bpftool
├── bash-completion
├── btf.c
├── btf_dumper.c
├── cfg.c
├── cfg.h
├── cgroup.c
├── common.c
├── Documentation
├── feature.c
├── jit_disasm.c
├── json_writer.c
├── json_writer.h
├── main.c
├── main.h
├── Makefile
├── map.c
├── map_perf_ring.c
├── net.c
├── netlink_dumper.c
├── netlink_dumper.h
├── perf.c
├── prog.c
├── tracelog.c
├── xlated_dumper.c
└── xlated_dumper.h
```

```
# bpftool prog show
1337: sched_cls  name cls_entry  tag e202124da7c84e89
      loaded_at Mar 08/19:53  uid 0
      xlated 304B  not jited  memlock 4096B
```

```
# bpftool prog dump xlated id 1337
   0: (71) r6 = *(u8 *)(r1 +142)
   1: (54) (u32) r6 &= (u32) 1
   2: (15) if r6 == 0x0 goto pc+7
   3: (bf) r6 = r1
   [...]
  37: (95) exit
```

```
# bpftool map
1234: array  name ch_rings  flags 0x0
      key 4B  value 4B  max_entries 7860  memlock 65536B

# bpftool map dump id 1234
key: 00 00 00 00  value: 00 00 00 00
key: 01 00 00 00  value: 00 00 00 00
key: 02 00 00 00  value: 00 00 00 00
key: 03 00 00 00  value: 00 00 00 00
[...]
Found 7860 elements
```

4) Summary

- **Evaluation**

"eBPF is Linux's new superpower"

Gaurav Gupta

"Run code in the kernel without having to write a kernel module"

Liz Rice

"eBPF does to Linux what JavaScript does to HTML"

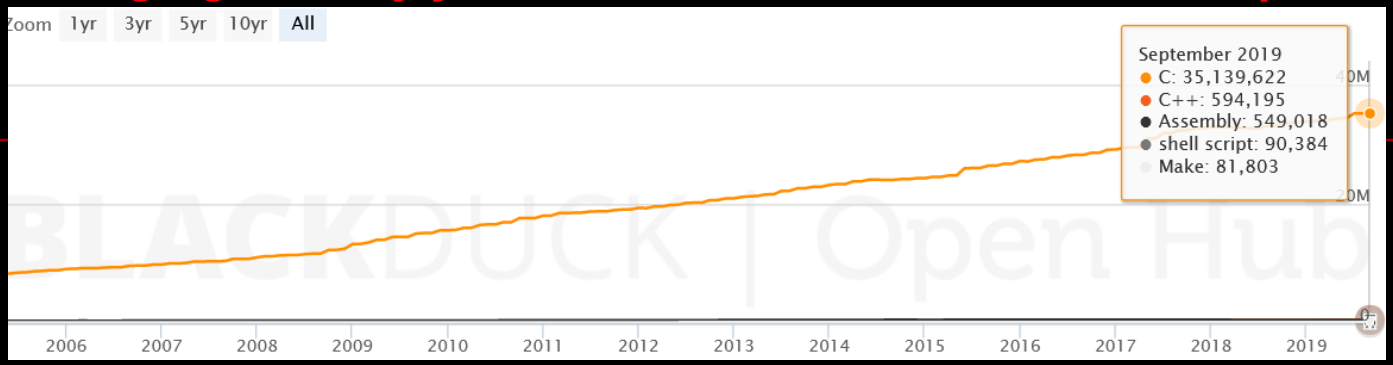
Brendan Gregg

Source: ebpfbasics-190611051559.pdf

- **Kernel Space & User Space Instrumentation**

- **Polyglot VM**

  Changing the way you think about Linux Kernel development:



Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C	35,139,622	6,241,390	15.1%	6,333,153	47,714,165	95.7%
C++	594,195	262,892	30.7%	116,246	973,333	2.0%
Assembly	549,018	124,794	18.5%	96,032	769,844	1.5%

Source: https://www.openhub.net/p/linux/analyses/latest/languages_summary

- **The Next Linux Superpower:**

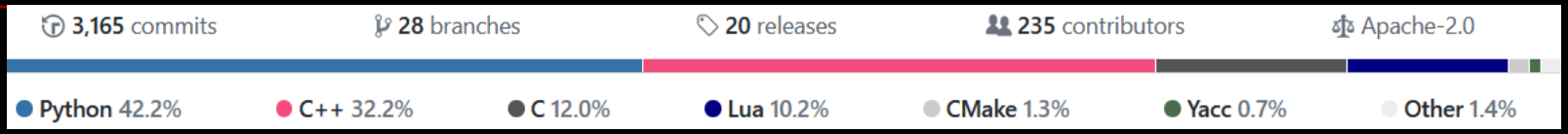
  User Space/Kernel Space Repartition & Unifying

  Reconstructing nearly every aspect of Linux Networking and Security subsystem

II. BCC(BPF Compiler Collection)

1) Overview

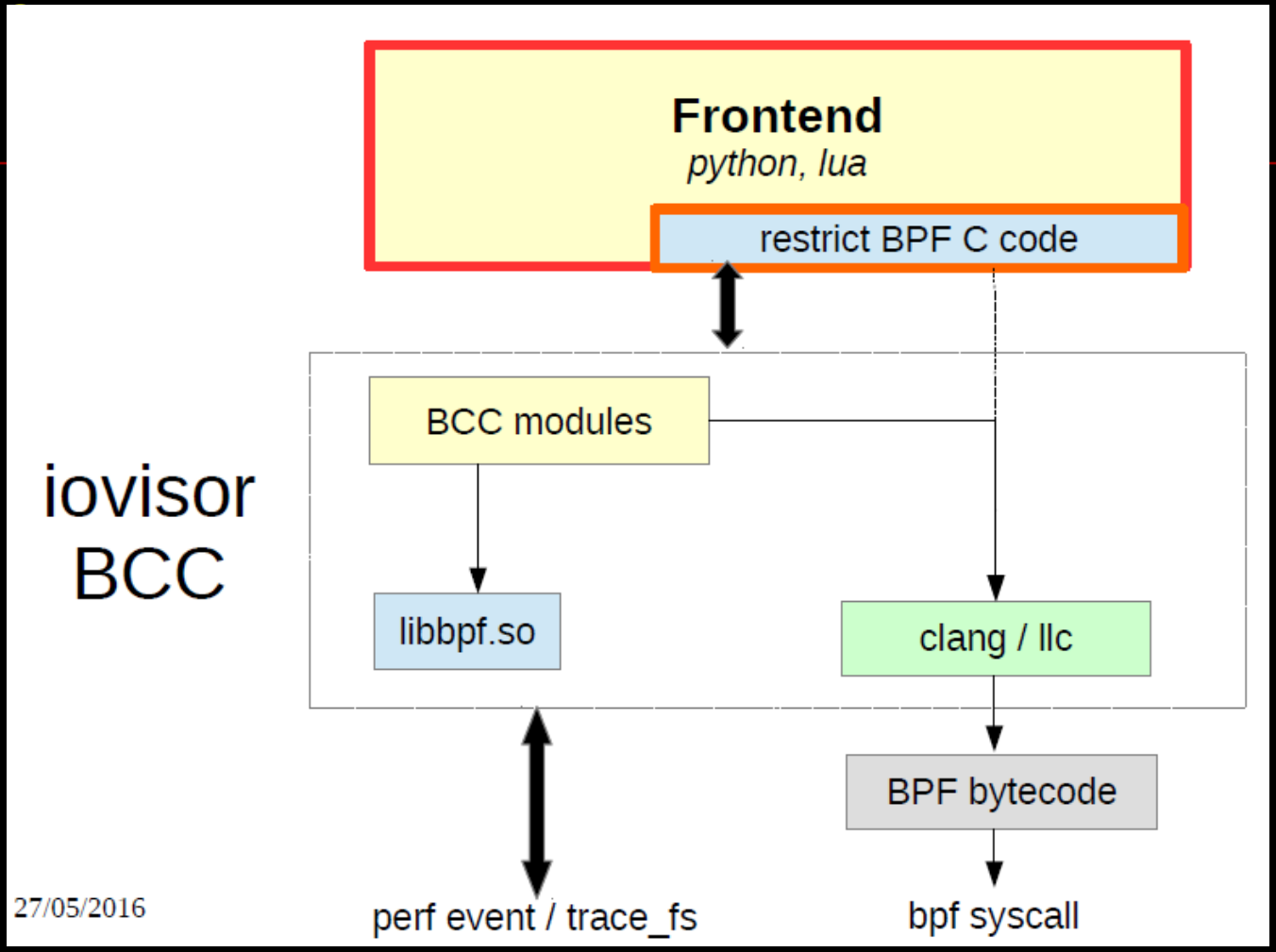
- https://github.com/iovisor/bcc/



A toolkit with Python/Lua frontend for compiling, loading, and executing BPF programs, which allows user-defined instrumentation on a live kernel image:

- compile BPF program from C source
- attach BPF program to kprobe/uprobe/tracepoint/USDT/socket
- poll data from BPF program
- framework for building new tools or one-off scripts
- additional projects to support Go, Rust, and DTrace-style frontend
- …

Architecture

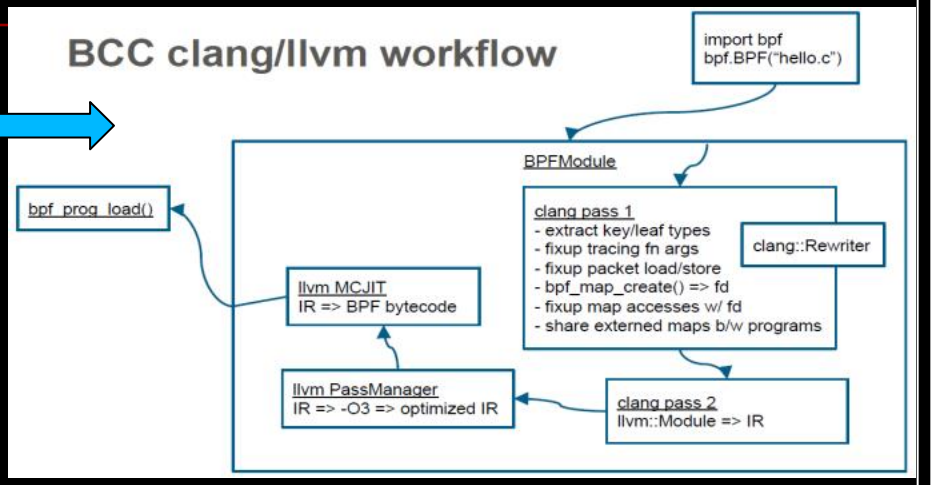


Source: http://www.slideshare.net/vh21/meet-cutebetweenebpfandtracing

A Sample

- https://lwn.net/Articles/747640/ //Some advanced BCC topics

```
#!/usr/bin/env python

from bcc import BPF
from time import sleep

program = """
    BPF_HASH(callers, u64, unsigned long);

    TRACEPOINT_PROBE(kmem, kmalloc) {
        u64 ip = args->call_site;
        unsigned long *count;
        unsigned long c = 1;

        count = callers.lookup((u64 *)&ip);
        if (count != 0)
            c += *count;

        callers.update(&ip, &c);

        return 0;
    }
"""
b = BPF(text=program)

while True:
    try:
        sleep(1)
        for k,v in sorted(b["callers"].items()):
            print ("%s %u" % (b.ksym(k.value), v.value))
        print
    except KeyboardInterrupt:
        exit()
```



Source: http://linuxplumbersconf.org/2015/ocw//system/presentations/3249/original/bpf_llvm_2015aug19.pdf

The output from this little program looks like:

```
# ./example.py
i915_sw_fence_await_dma_fence 4
intel_crtc_duplicate_state 4
SyS_memfd_create 1
drm_atomic_state_init 4
sg_kmalloc 7
intel_atomic_state_alloc 4
seq_open 504
SyS_bpf 22
```

2) Development

- https://github.com/iovisor/bcc/blob/master/docs/tutorial.md
- https://github.com/iovisor/bcc/blob/master/docs/reference_guide.md
- https://github.com/iovisor/bcc/blob/master/docs/tutorial_bcc_python_developer.md
- …

2.1 New Ideas

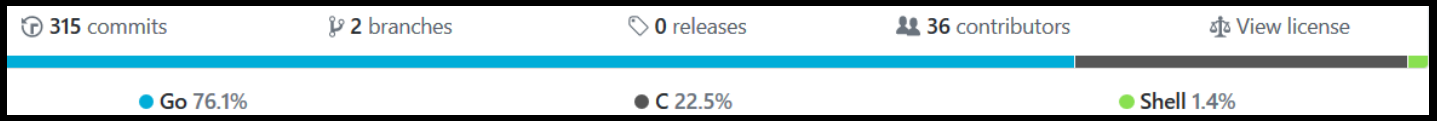
py2bpf

- https://github.com/facebookresearch/py2bpf
- translates functions from Python to BPF

```
q = py2bpf.datastructures.BpfQueue(ctypes.c_int)

@py2bpf.kprobe.probe('sys_close')
def on_sys_close(pt_regs):
    pid = py2bpf.funcs.get_current_pid_tgid() & 0xffffffff
    ptr = py2bpf.funcs.addrof(pid)
    cpuid = py2bpf.funcs.get_smp_processor_id()
    py2bpf.funcs.perf_event_output(pt_regs, q, cpuid, ptr)
    return 0


with on_sys_close():
    for pid in q:
        print('pid={}'.format(pid))
```

gobpf

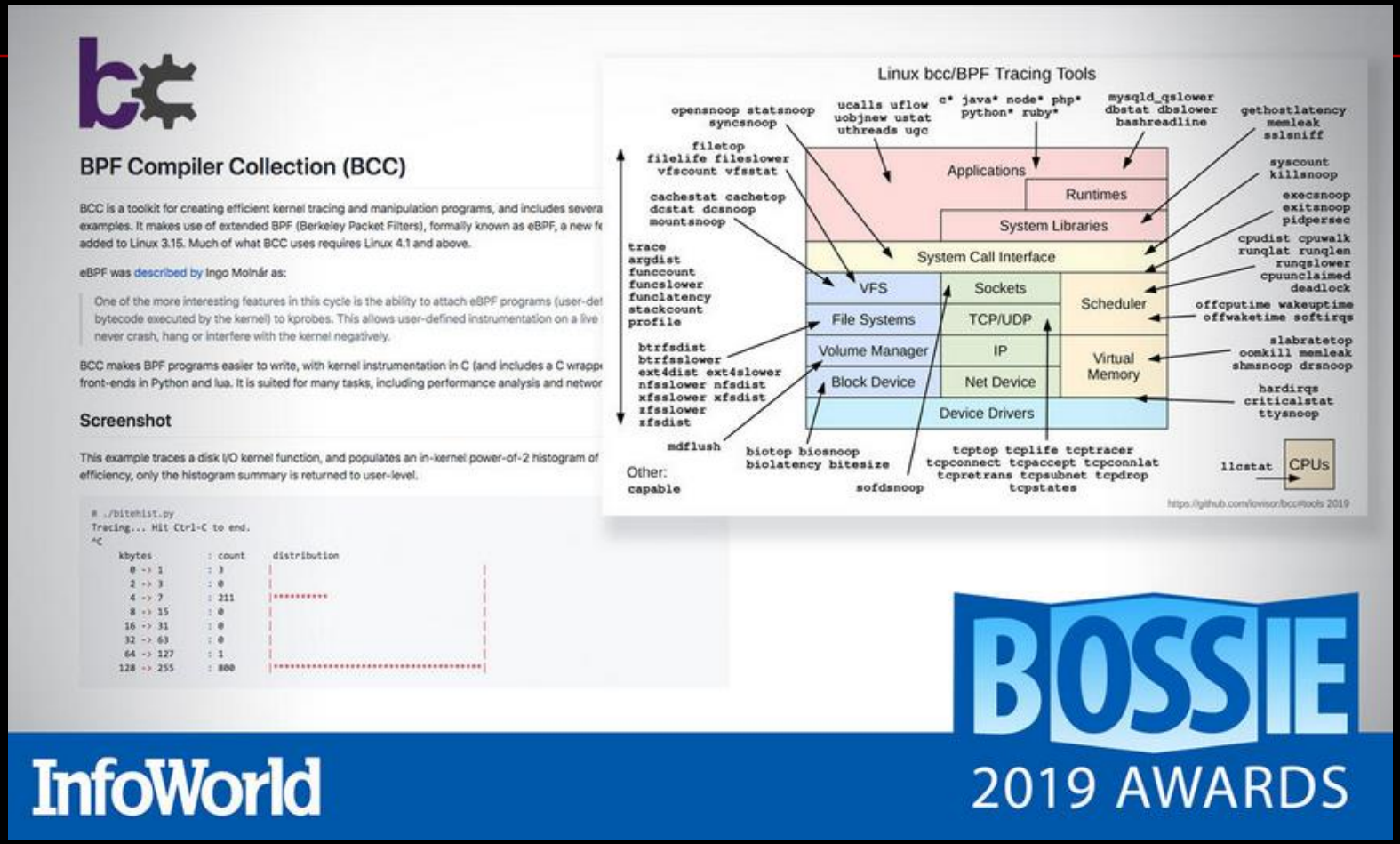
- https://github.com/iovisor/gobpf
- provides go bindings for the BCC framework as well as low-level routines to load and use eBPF programs from .elf files



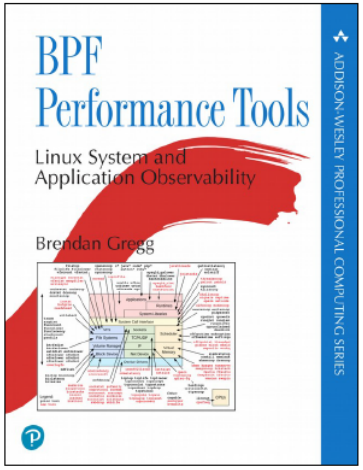
- requirement: install the latest released version of libbcc
- still is in early stage, but usable

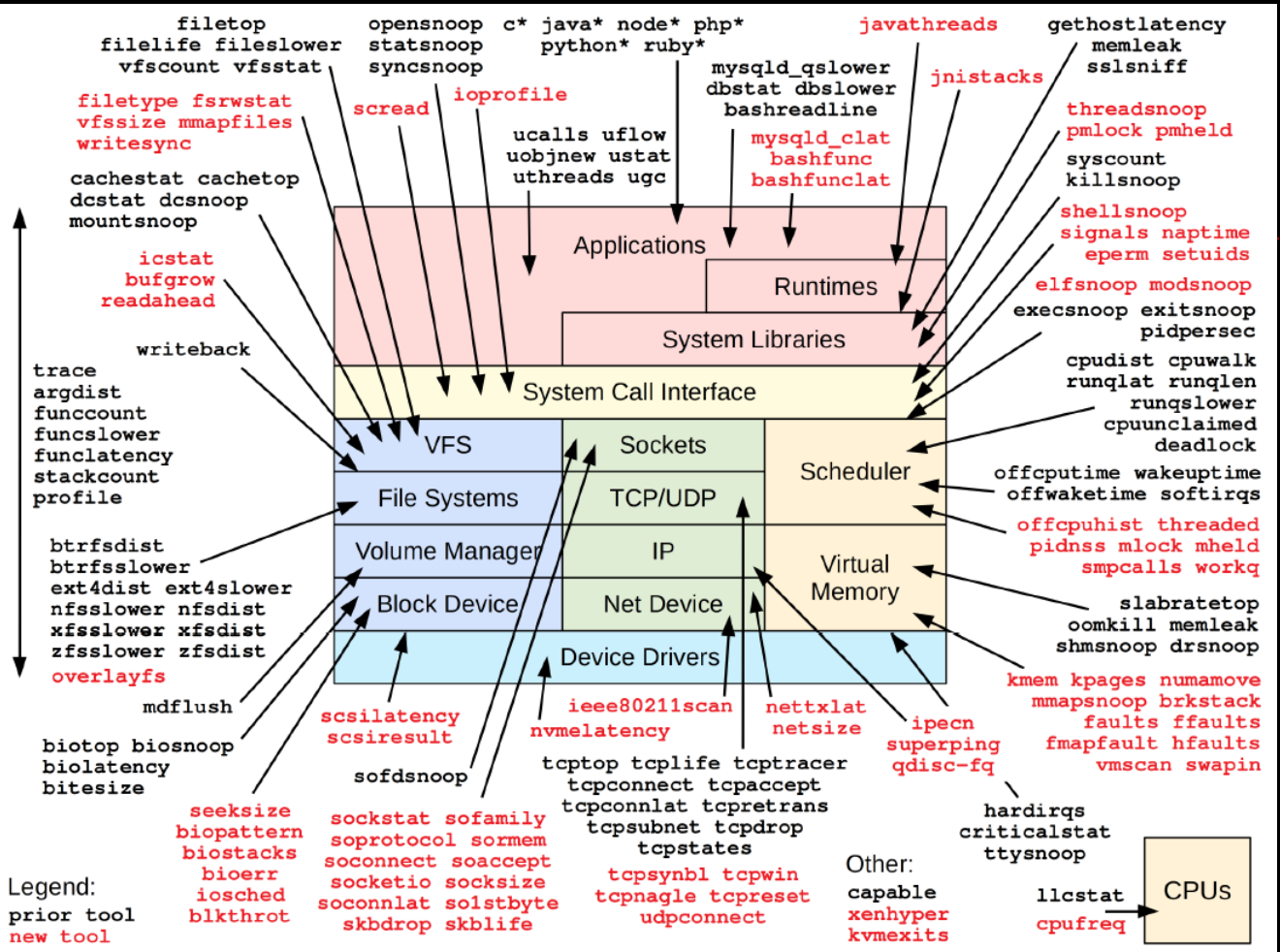
3) Summary

- https://www.infoworld.com/article/3444198/the-best-open-source-software-of-2019.html



BPF Perf Tools





Source: "BPF Tracing Tools", Brendan Gregg, Linux Plumbers Conference 2019

III. LISA

1) Overview

- Linux Integrated System Analysis
- https://github.com/ARM-software/lisa
-

The LISA project provides a toolkit that supports regression testing and interactive analysis of Linux kernel behavior. LISA stands for Linux Integrated/Interactive System Analysis. LISA's goal is to help Linux kernel developers to measure the impact of modifications in core parts of the kernel. The focus is on the scheduler (e.g. EAS), power management and thermal frameworks. However LISA is generic and can be used for other purposes too.

LISA has a *host/target* model. LISA itself runs on a *host* machine, and uses the devlib toolkit to interact with the *target* via SSH, ADB or telnet. LISA is flexible with regard to the target OS; its only expectation is a Linux kernel-based system. Android, GNU/Linux and busybox style systems have all been used.

LISA provides features to describe workloads (notably using rt-app) and run them on targets. It can collect trace files from the target OS (e.g. systrace and ftrace traces), parse them via the TRAPpy framework. These traces can then be parsed and analysed in order to examine detailed target behaviour during the workload's execution.

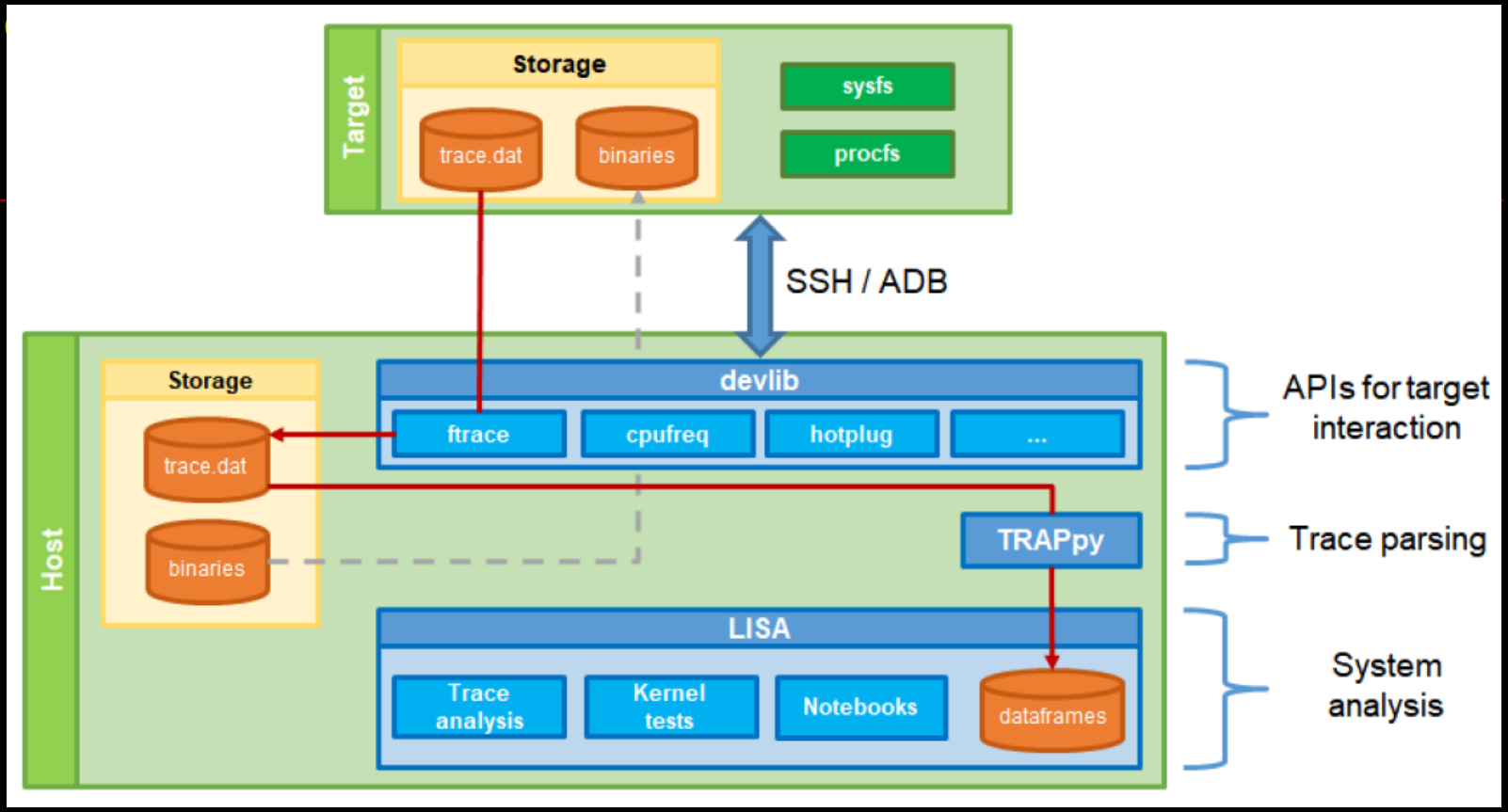
Some LISA features may require modifying the target OS. For example, in order to collect ftrace files the target kernel must have CONFIG_DYNAMIC_FTRACE enabled.

There are two "entry points" for running LISA:

- Via the Jupyter/IPython notebook framework. This allows LISA to be used interactively and supports visualisation of trace data. Some notebooks are provided with example and ready-made LISA use-cases.
- Via the automated test framework. This framework allows the development of automated pass/fail regression tests for kernel behaviour. The BART toolkit provides additional domain-specific test assertions for this use-case. LISA provides some ready-made automated tests under the `lisa/tests/` directory.

- Unlike devlib or trappy, LISA is now Python 3 only.
- A big refactoring effort was started in mid 2018, which produced a lot of (much needed) changes.

Architecture

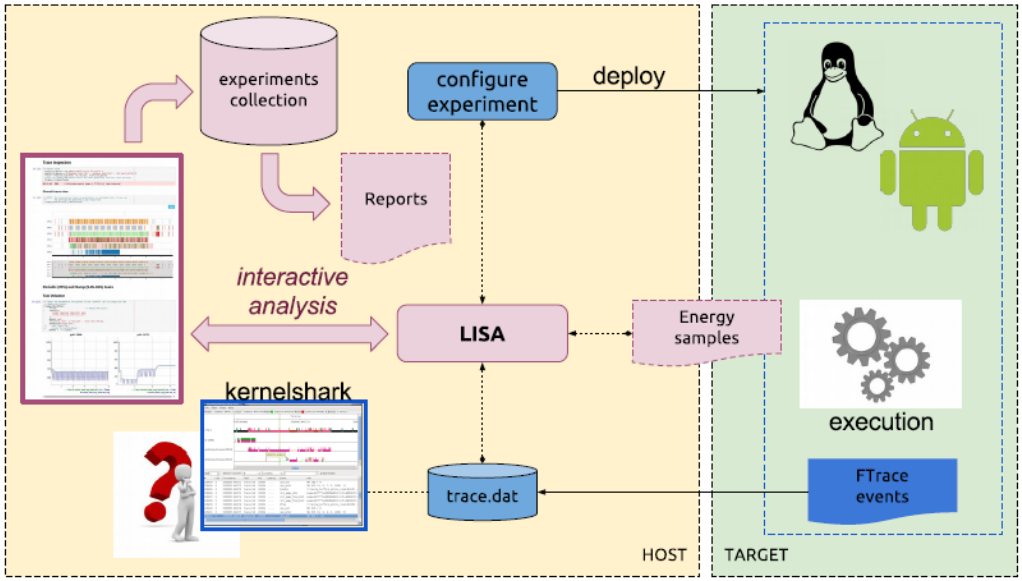


Source: https://lisa-linux-integrated-system-analysis.readthedocs.io/en/master/

2) Internals

Workflow

- Experimenting using an “interactive environment”
- Data analysis and post-processing
- Tests definitions to support regression analysis

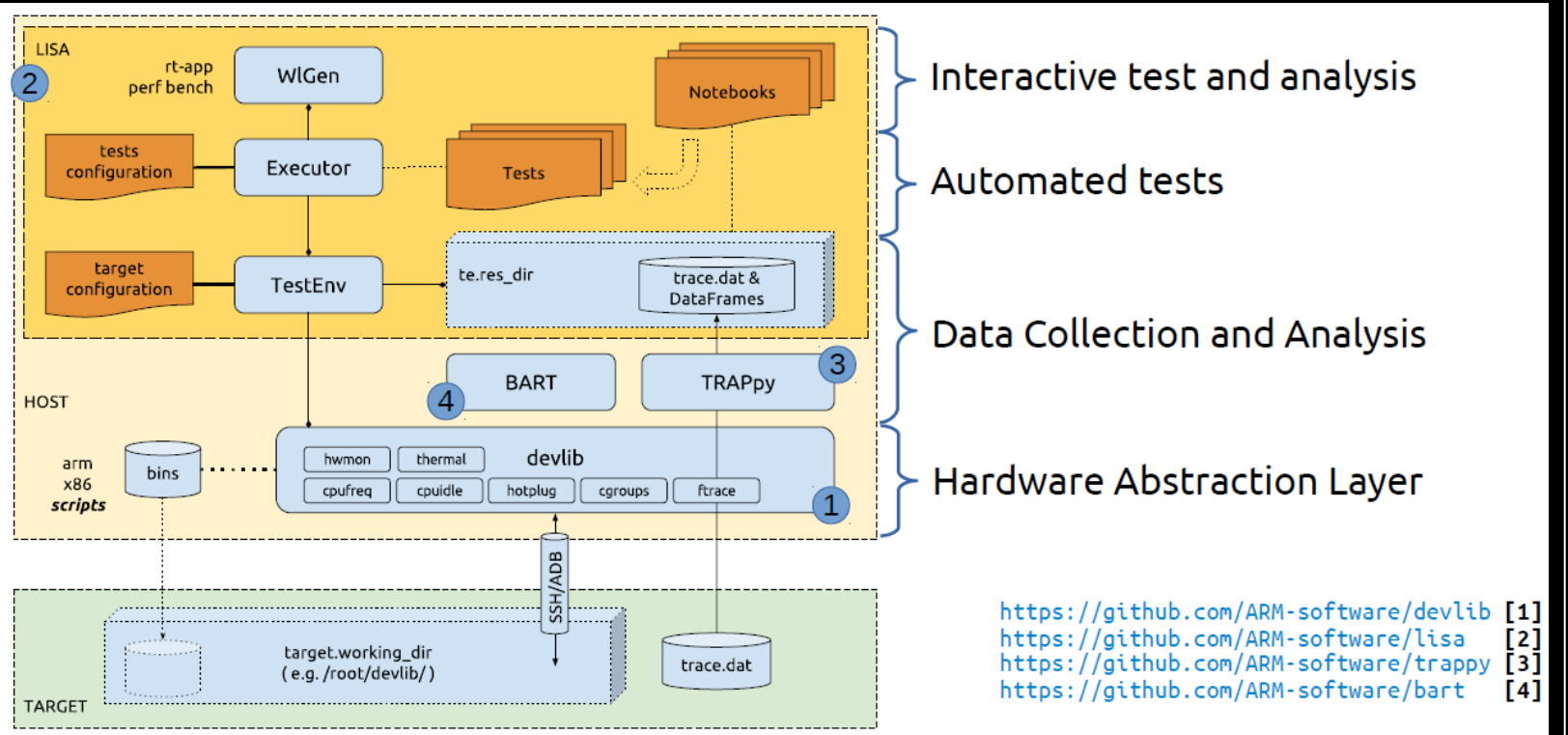


Evaluate trade-offs on Power/Performances

Classical flow vs LISA flow

Source: “Linux Integrated System Analysis (LISA) & Friends ”, Patrick Bellasi, ELC 2016

Toolkit



https://github.com/ARM-software/devlib [1]
https://github.com/ARM-software/lisa [2]
https://github.com/ARM-software/trappy [3]
https://github.com/ARM-software/bart [4]

Source: "Linux Integrated System Analysis (LISA) & Friends ", Patrick Bellasi, ELC 2016

Data Analysis

- LISA::Trace glues platform data with TRAPpy DataFrames

  more complete analysis dataset

- LISA::Filters

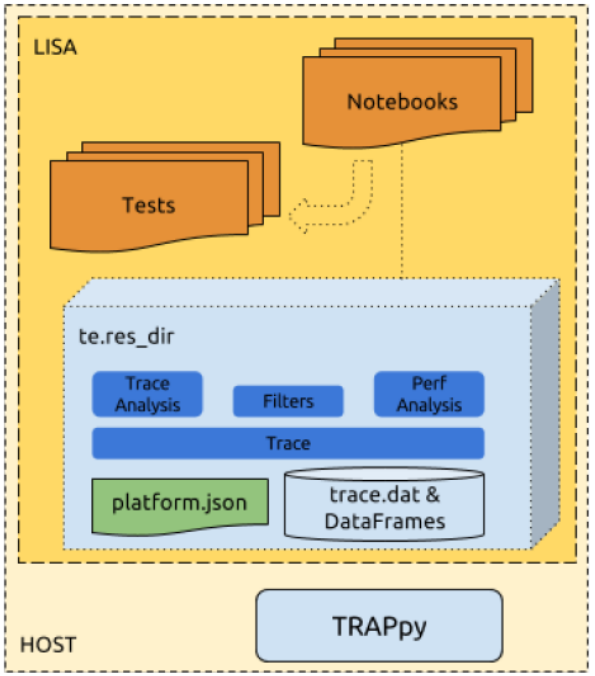
  commonly used events filtering functions

- LISA::TraceAnalysis

  commonly used trace events plots

- LISA::PerfAnalysis

  commonly used performance plots



Source: "Linux Integrated System Analysis (LISA) & Friends ", Patrick Bellasi, ELC 2016

trappy

- https://github.com/ARM-software/trappy
- Under the hood, it is used to convert the trace events into pandas.DataFrame which are suited to handling large data sets

---

rt-app

- https://github.com/scheduler-tools/rt-app
- emulates typical mobile and real-time systems use cases and gives runtime information

workload-automation

- https://github.com/ARM-software/workload-automation
- a framework for executing workloads and collecting measurements on Android and Linux devices. It includes automation for nearly 40 workloads and supports some common instrumentation (ftrace, hwmon) along with a number of output formats

2.1 Automated Testing

- **exekall** is the test runner of LISA. **lisa-test** command is a thin wrapper on top of exekall.
- use `exekall run *subcommand*` to start a test session
  e.g., `exekall run lisa/tests/ --conf target_conf.yml`
- **bisector** allows setting up the steps of a test iteration, repeating them an infinite number of times (by default).

e.g.,

```
# As a convenience, myreport.yml.gz.log will also be created, with a
# behaviour similar to: bisector run ... 2>&1 | tee myreport.yml.gz.log
bisector run --steps steps.yml --report myreport.yml.gz
```

- **check-test-fix.py** tool can be used to check that a fix to a test resolved errors or a regression.

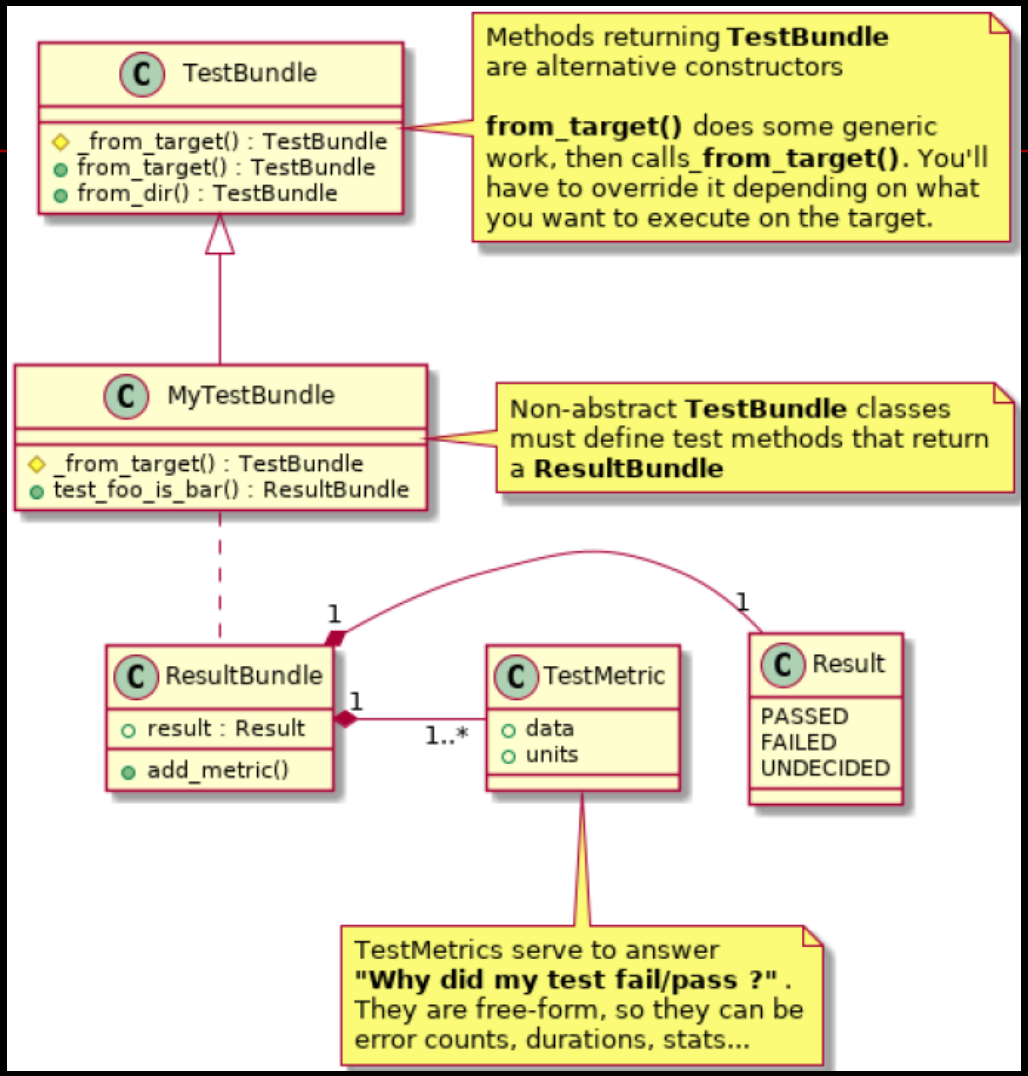
e.g.,

```
# The test to check is selected using --select in the same way as for `exekall run`.
# hikey960.report.yml.gz is a bisector report generated using `bisector run`
# All options coming after the report are passed to `bisector report` to
# control what artifacts are downloaded and what TestBundle are used.
check-test-fix.py --select 'OneSmallTask:test_task_placement' hikey960.report.yml.gz -
↪oiterations=1-20
```

- **Wltest** enables the comparison of power and performance impacts of kernel changes on Android devices.
- ...

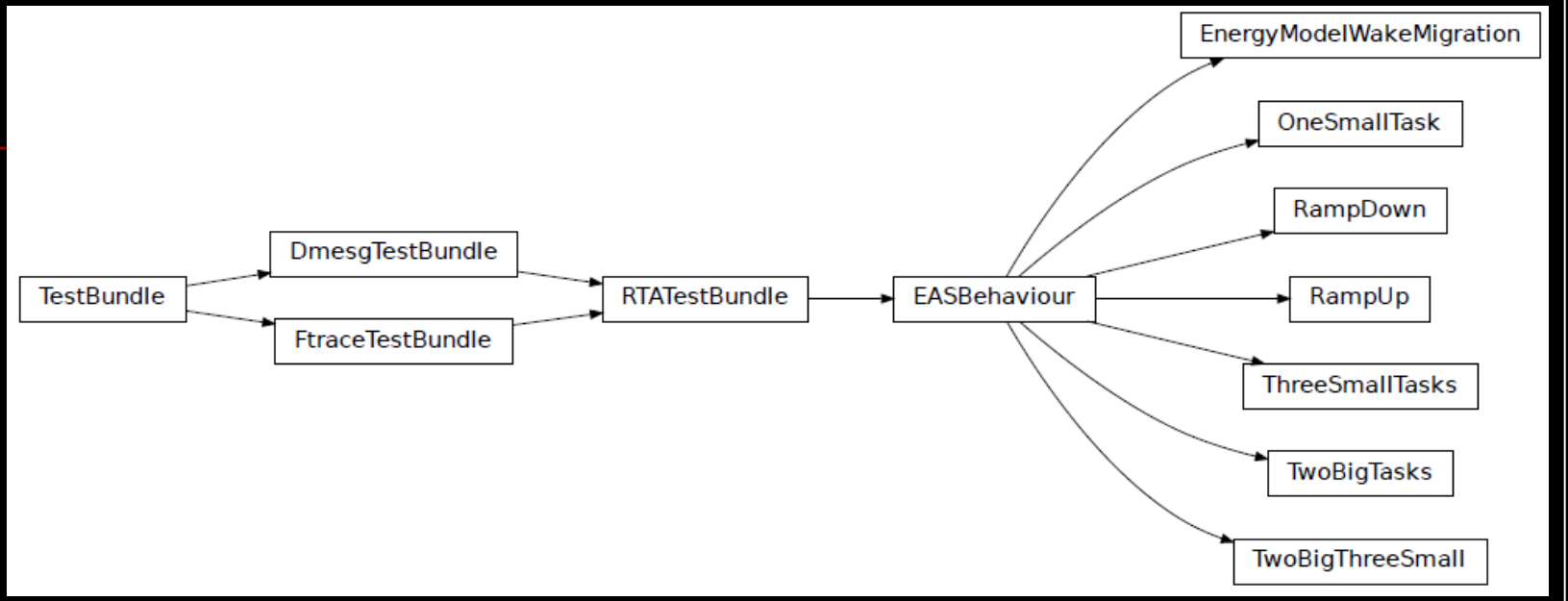
Writing Tests

- the main class of the kernel tests is **TestBundle**



Source: https://lisa-linux-integrated-system-analysis.readthedocs.io/en/master/

- **E.g., scheduler tests**
  EAS tests



Source: https://lisa-linux-integrated-system-analysis.readthedocs.io/en/master/

- **HWMON: ARM Energy Probe (AEP)**

  ARM Energy Probes are lightweight power measurement tools for software developers. They can monitor up to three voltage rails simultaneously.

Source: https://lisa-linux-integrated-system-analysis.readthedocs.io/en/master/

Caiman

https://github.com/ARM-software/caiman

Caiman can usually auto-detect the Energy Probe device. But on older versions of Linux which do not have `libudev.so.0`, such as Red Hat, it cannot be auto-detected. In that case the device name will need to be provided, usually `/dev/ttyACM0`.

setting up the ARM Energy Probe or National Instruments DAQ and running caiman…

3) Development

- https://github.com/ARM-software/lisa/tree/master/doc
- https://lisa-linux-integrated-system-analysis.readthedocs.io/en/master/

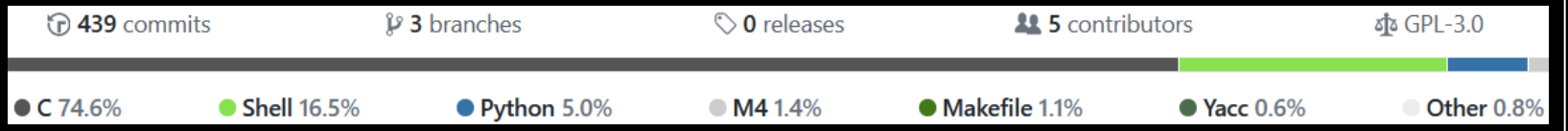
- …

IV. Drgn

1) Overview

- https://github.com/osandov/drgn

scriptable debugger library



- drgn (pronounced "dragon") is a debugger-as-a-library. In contrast to existing debuggers like GDB which focus on breakpoint-based debugging, drgn excels in live introspection. drgn exposes the types and variables in a program for easy, expressive scripting in Python.

drgn was developed for debugging the Linux kernel (as an alternative to the crash utility), but it can also debug userspace programs written in C. C++ support is planned.

- https://lwn.net/Articles/789641/ //A kernel debugger in Python
- https://drgn.readthedocs.io/en/latest/

drgn debugs the running kernel by default; run `sudo drgn`. To debug a running program, run `sudo drgn -p $PID`. To debug a core dump (either a kernel vmcore or a userspace core dump), run `drgn -c $PATH`. The program must have debugging symbols available.

Script Mode

Script mode is useful for reusable scripts. Simply pass the path to the script along with any arguments:

```
$ cat script.py
import sys
from drgn.helpers.linux import find_task

pid = int(sys.argv[1])
uid = find_task(prog, pid).cred.uid.val.value_()
print(f'PID {pid} is being run by UID {uid}')
$ sudo drgn script.py 601
PID 601 is being run by UID 1000
```

It's even possible to run drgn scripts directly with the proper shebang:

```
$ cat script2.py
#!/usr/bin/env drgn

mounts = prog['init_task'].nsproxy.mnt_ns.mounts.value_()
print(f'You have {mounts} filesystems mounted')
$ sudo ./script2.py
You have 36 filesystems mounted
```

Future

- Combining with BPF tracing
- Better debug information
- Better vmcores

Source: "drgn: Programmable Debugging", Omar Sandoval (Facebook), Linux Plumbers Conference 2019

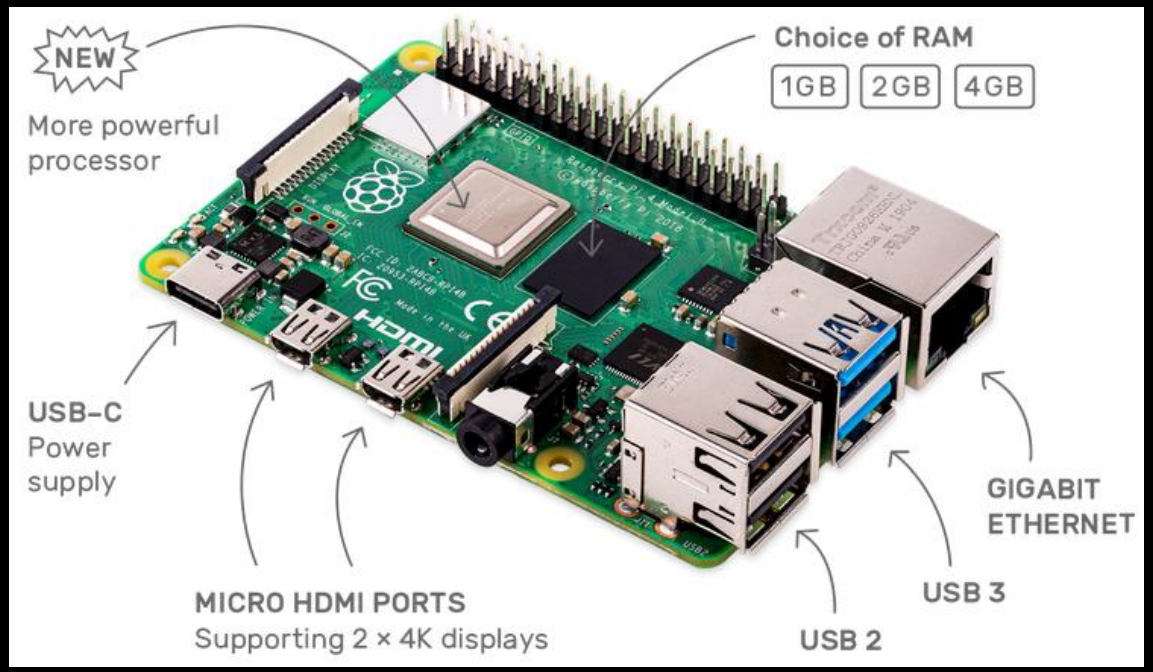
V. Practice on ARM

1) Development Environment

1.1 HW/SW Environment

RPi 4B

- https://www.raspberrypi.org/products/raspberry-pi-4-model-b/



Model 4B vs 3B+

Features/Specs	**Raspberry Pi 4B**	**Raspberry Pi 3 B+**
Release date	24th June 2019	14th March 2018
SoC	Broadcom BCM2711 quad-core Cortex-A72 @ 1.5 GHz	Broadcom BCM2837B0 quad-core Cortex-A53 @ 1.4 GHz
GPU	VideoCore VI with OpenGL ES 1.1, 2.0, 3.0	VideoCore IV with OpenGL ES 1.1, 2.0
Video Decode	H.265 4Kp60, H.264 1080p60	H.264 & MPEG-4 1080p30
Video Encode	H.264 1080p30	
Memory	1GB, 2GB, or 4GB LPDDR4	1GB LPDDR2
Storage	microSD card	
Video & Audio Output	2x micro HDMI ports up to 4Kp60 3.5mm AV port (composite + audio) MIPI DSI connector	1x HDMI 1.4 port up to 1080p60 3.5mm AV port (composite + audio) MIPI DSI connector
Camera	MIPI CSI connector	
Ethernet	Native Gigabit Ethernet	Gigabit Ethernet over USB (300 Mbps max.)
WiFi	Dual band 802.11 b/g/n/ac	
Bluetooth	Bluetooth 5.0 + BLE	Bluetooth 4.2 + BLE
USB	2x USB 3.0 + 2x USB 2.0	4x USB 2.0
Expansion	40-pin GPIO header	
Power Supply	5V via USB type-C up to 3A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT	5V via micro USB up to 2.5A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT
Dimensions	85×56 mm	
Default OS	Raspbian (after June 24, 2019)	Raspbian (after March 2018)
Price	$35 (1GB RAM), $45 (2GB RAM), $55 (4GB RAM)	$35 (1GB RAM)

Source: https://www.cnx-software.com/2019/06/24/raspberry-pi-4-vs-pi-3-what-are-the-differences/

Manjaro

- an open-source Linux distribution based on the Arch Linux operating system
- https://en.wikipedia.org/wiki/Manjaro_Linux
- https://manjaro.org
-

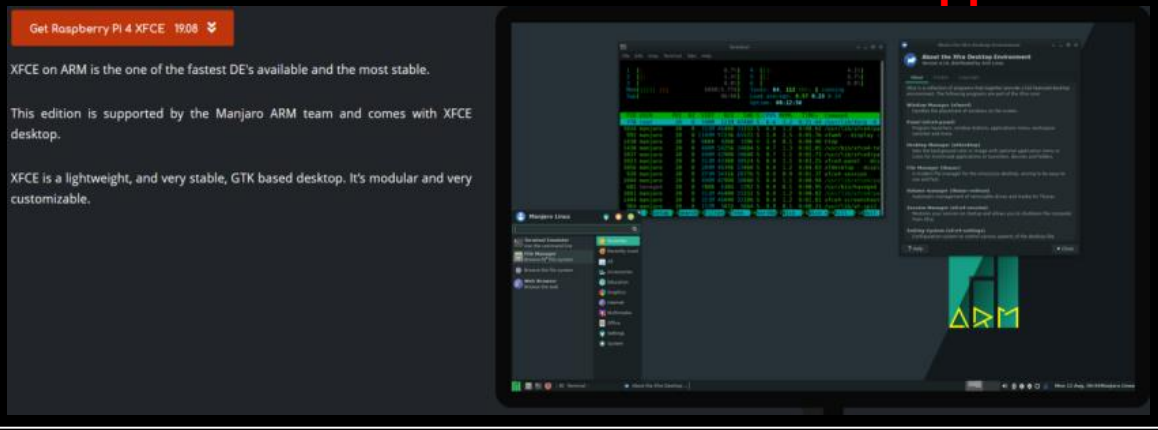
Rank	Distribution	H.P.D*
1	Mint	2880▼
2	Debian	1668▬
3	Ubuntu	1347▬
4	openSUSE	1234▲
5	elementary	1085▬
6	Manjaro	1039▼
7	Fedora	982▼
8	Zorin	914▬
9	CentOS	773▬
10	deepin	736▲

2016-->2019

Rank	Distribution	HPD*
1	MX Linux	4895
2	Manjaro	2597
3	Mint	2048
4	Debian	1543
5	Ubuntu	1398
6	elementary	1302
7	Solus	1087
8	Fedora	992
9	deepin	847
10	Zorin	830

https://www.distrowatch.com

- the first ARM64 Linux distribution that support RPi 4



- Advantages (from my point of view)
  - update packages quickly
  - group install
    https://www.archlinux.org/groups/
  - Python 3 is the default Python version now
    https://hg.python.org/peps/rev/76d43e52d978
  - much more stable than expected
- Preparation

```
sudo pacman -Syu                                                    //update system
pacman -Ss packagename                                              //search package


sudo pacman -S --needed base base-devel
sudo pacman -S --needed linux-tools

sudo pacman -S --needed vim colordiff axel                          //utilities
sudo pacman -S --needed cmake ninja maven meson scons               //Build tools
sudo pacman -S --needed valgrind c-ares protobuf boost              //Dev tools

sudo pacman -S --needed llvm clang lld lldb                         //LLVM
sudo pacman -S --needed libxml2 doxygen swig subversion             //for build LLVM
sudo pacman -S --needed python-pip cython python-netaddr python-pyroute2     //Python packages
sudo pacman -S --needed docker qemu libvirt podman skopeo buildah            //Virtualization

sudo pacman -S --needed iperf  ethtool                              //Network tools
sudo pacman -S --needed bc rsync                                    //for build Kernel

sudo pacman -S --needed rust                                        //Rust
sudo pacman -S --needed lua luajit luarocks                         //Lua
```

https://wiki.archlinux.org/index.php/Pacman

- https://www.howtogeek.com/430556/why-i-switched-from-ubuntu-to-manjaro-linux/

1.2 rebuild the Linux Kernel with desired configurations

get Kernel configuration

- sudo modprobe configs
  zcat /proc/config.gz > old.config //currently v4.19.79-2

edit new Kernel configuration

- e.g. edit "rpi4kernel-5.3_defconfig" (use *old.config* as the base) by adding following configures (remove the old one if it was configured twice)

```
#
# eBPF
#
CONFIG_CGROUP_BPF=y
CONFIG_BPF=y
CONFIG_BPF_SYSCALL=y
CONFIG_BPF_JIT_ALWAYS_ON=y
CONFIG_IPV6_SEG6_BPF=y
# CONFIG_BPFILTER is not set
CONFIG_NET_CLS_BPF=m
CONFIG_NET_ACT_BPF=m
CONFIG_BPF_JIT=y
CONFIG_BPF_STREAM_PARSER=y
CONFIG_LWTUNNEL_BPF=y
CONFIG_HAVE_EBPF_JIT=y
CONFIG_BPF_LIRC_MODE2=y
CONFIG_BPF_EVENTS=y
# CONFIG_TEST_BPF is not set
# XDP
CONFIG_XDP_SOCKETS=y
CONFIG_XDP_SOCKETS_DIAG=m
```

```
#
# Miscs
#
CONFIG_KSM=y
CONFIG_PROC_EVENTS=y
#CONFIG_IKHEADERS=y
CONFIG_REMOTEPROC=m
CONFIG_REISERFS_PROC_INFO=y
CONFIG_PROC_CHILDREN=y
```

...

```
#
# Debug
#
CONFIG_ARCH_HAS_DEBUG_VIRTUAL=y
CONFIG_STRIP_ASM_SYMS=y
CONFIG_UNUSED_SYMBOLS=y
CONFIG_PM_DEBUG=y
CONFIG_CMA_DEBUGFS=y
CONFIG_L2TP_DEBUGFS=m
CONFIG_6LOWPAN_DEBUGFS=y
CONFIG_CFG80211_DEBUGFS=y
CONFIG_MAC80211_DEBUGFS=y
CONFIG_DEBUG_DEVRES=y
CONFIG_SCSI_DEBUG=m
CONFIG_DM_DEBUG=y
CONFIG_DM_DEBUG_BLOCK_MANAGER_LOCKING=y
CONFIG_ATH9K_DEBUGFS=y
CONFIG_ATH6KL_DEBUG=y
CONFIG_SUNRPC_DEBUG=y
CONFIG_DLM_DEBUG=y
CONFIG_DYNAMIC_DEBUG=y
CONFIG_DEBUG_INFO=y
#CONFIG_DEBUG_INFO_DWARF4=y
CONFIG_DEBUG_SECTION_MISMATCH=y
CONFIG_DEBUG_RODATA_TEST=y
CONFIG_DEBUG_VM=y
CONFIG_DEBUG_SHIRQ=y
CONFIG_DEBUG_LIST=y
CONFIG_ARM64_PTDUMP_DEBUGFS=y
CONFIG_DEBUG_WX=y
CONFIG_PROC_KCORE=y
CONFIG_PROC_VMCORE=y
CONFIG_PROC_VMCORE_DEVICE_DUMP=y
CONFIG_FTRACE_SYSCALLS=y
CONFIG_HWLAT_TRACER=y
```

setup

- git clone https://github.com/raspberrypi/linux
  //currently, master is branch rpi-4.19.y
  git checkout rpi-5.3.y
  cp rpi4kernel-5.3_defconfig $RPILINUX_SRC/arch/arm64/configs

build and install the new Kernel

- cd $RPILINUX_SRC
  make ARCH=arm64 rpi4kernel-5.3_defconfig   //generate .config
  make -j4
  //on **RPi 4B** with Manjaro-ARM-xfce-rpi4-19.08 + GCC 9.1.0 + gnu ld 2.32 + 4GB LPDDR4 + 

, and **JOBS=4:**
  ~1h50m for a full build
  sudo make headers_install modules_install install
- cd /boot
  sudo cp vmlinux-5.3.5-MANJARO-ARM+ kernel8.img
  //same as $RPILINUX_SRC/arch/arm64/boot/Image
  sudo reboot
-
```
[myrpi4@myrpi4h1 ~]$ uname -a
Linux myrpi4h1 5.3.5-MANJARO-ARM+ #1 SMP PREEMPT Sat Oct 12 14:08:59 UTC 2019 aarch64 GNU/Linux
```

2) Experiments
2.1 BCC
build

- https://github.com/iovisor/bcc/blob/master/INSTALL.md

Required

```
CONFIG_BPF=y
CONFIG_BPF_SYSCALL=y
# [optional, for tc filters]
CONFIG_NET_CLS_BPF=m
# [optional, for tc actions]
CONFIG_NET_ACT_BPF=m
CONFIG_BPF_JIT=y
# [for Linux kernel versions 4.1 through 4.6]
CONFIG_HAVE_BPF_JIT=y
# [for Linux kernel versions 4.7 and later]
CONFIG_HAVE_EBPF_JIT=y
# [optional, for kprobes]
CONFIG_BPF_EVENTS=y
```

Optional

```
CONFIG_NET_SCH_SFQ=m
CONFIG_NET_ACT_POLICE=m
CONFIG_NET_ACT_GACT=m
CONFIG_DUMMY=m
CONFIG_VXLAN=m
```

- git clone https://github.com/iovisor/bcc
  cd $BCC_SRC
  git submodule update --init --recursive
  //https://github.com/iovisor/bcc/tree/master/src/cc

```
The libbpf directory is a git submodule for repository
  https://github.com/libbpf/libbpf

If you have any change in libbpf directory, please upstream to linux
first as libbpf repo is a mirror of linux/tools/lib/bpf directory.
```

- mkdir $BCC_SRC/build; cd $BCC_SRC/build
  cmake .. -DCMAKE_INSTALL_PREFIX=/usr
  make -j4
  sudo make install
  //on RPi 4B with Manjaro-ARM-xfce-rpi4-19.08 + Kernel 5.3.5 + GCC 9.1.0 + gnu ld 2.32 + 4GB LPDDR4 + 

, and JOBS=4: ~11m for a full build
- run an example
  cd /usr/share/bcc/examples/tracing

```
[myrpi4@myrpi4h1 tracing]$ sudo ./bitehist.py
[sudo] password for myrpi4:
Tracing... Hit Ctrl-C to end.
^C
log2 histogram
~~~~~~~~~~~~~~
     kbytes              : count     distribution
         0 -> 1          : 0        |                                        |
         2 -> 3          : 0        |                                        |
         4 -> 7          : 35       |****************************************|
         8 -> 15         : 16       |******************                      |
        16 -> 31         : 0        |                                        |
        32 -> 63         : 2        |**                                      |

linear histogram
~~~~~~~~~~~~~~~~
     kbytes        : count     distribution
        0          : 0        |                                        |
        1          : 0        |                                        |
        2          : 0        |                                        |
        3          : 0        |                                        |
        4          : 35       |****************************************|
        5          : 0        |                                        |
        6          : 0        |                                        |
        7          : 0        |                                        |
        8          : 11       |************                            |
        9          : 0        |                                        |
        10         : 0        |                                        |
        11         : 0        |                                        |
        12         : 5        |*****                                   |
        13         : 0        |                                        |
        14         : 0        |                                        |
        15         : 0        |                                        |
        16         : 0        |                                        |
```

2.2 LISA

overview

- $LISA_SRC

```
-rw-r--r--  1 myrpi4 myrpi4  2275 Sep 15 16:21 CONTRIBUTING.md
-rw-r--r--  1 myrpi4 myrpi4   407 Oct  3 12:41 devmode_requirements.txt
drwxr-xr-x  7 myrpi4 myrpi4  4096 Oct  9 10:22 doc/
drwxr-xr-x  5 myrpi4 myrpi4  4096 Oct  3 12:41 external/
drwxr-xr-x  8 myrpi4 myrpi4  4096 Oct 16 13:22 .git/
drwxr-xr-x  3 myrpi4 myrpi4  4096 Sep 15 16:21 .github/
-rw-r--r--  1 myrpi4 myrpi4   221 Sep 15 16:21 .gitignore
-rw-r--r--  1 myrpi4 myrpi4  2114 Sep 15 16:21 init_env
-rwxr-xr-x  1 myrpi4 myrpi4 11245 Oct  9 10:22 install_base.sh*
lrwxrwxrwx  1 myrpi4 myrpi4    15 Sep 15 16:21 install_base_ubuntu.sh -> install_base.sh*
drwxr-xr-x  8 myrpi4 myrpi4  4096 Sep 15 16:21 ipynb/
-rw-r--r--  1 myrpi4 myrpi4 11358 Sep 15 16:21 LICENSE.txt
drwxr-xr-x  8 myrpi4 myrpi4  4096 Oct  9 10:22 lisa/
-rw-r--r--  1 myrpi4 myrpi4  3063 Sep 15 16:21 logging.conf
-rw-r--r--  1 myrpi4 myrpi4 17727 Sep 15 16:21 .pylintrc
-rw-r--r--  1 myrpi4 myrpi4  4951 Sep 15 16:21 README.rst
-rw-r--r--  1 myrpi4 myrpi4   577 Oct  3 12:41 .readthedocs.yml
-rw-r--r--  1 myrpi4 myrpi4    55 Sep 15 16:21 requirements.txt
-rw-r--r--  1 myrpi4 myrpi4   111 Sep 15 16:21 setup.cfg
-rwxr-xr-x  1 myrpi4 myrpi4  3763 Oct  9 10:22 setup.py*
drwxr-xr-x  2 myrpi4 myrpi4  4096 Sep 15 16:21 shell/
-rw-r--r--  1 myrpi4 myrpi4  2258 Sep 15 16:21 target_conf.yml
drwxr-xr-x  3 myrpi4 myrpi4  4096 Oct  2 12:11 tests/
drwxr-xr-x  7 myrpi4 myrpi4  4096 Oct 14 13:53 tools/
-rw-r--r--  1 myrpi4 myrpi4  1265 Sep 15 16:21 .travis.yml
-rw-r--r--  1 myrpi4 myrpi4  2440 Sep 15 16:21 Vagrantfile
```

patching and build

- https://lisa-linux-integrated-system-analysis.readthedocs.io/en/master/setup.html
  sudo ./install_base.sh --install-all
  //on RPi 4B with Manjaro-ARM-xfce-rpi4-19.08 + Kernel 5.3.5 + GCC 9.1.0 + gnu ld 2.32 + 4GB LPDDR4 + 

  ,
  ~4.5h for our patched build

disable Android & remove limitations for some Python packages

```
diff --git a/install_base.sh b/install_base.sh
index fc4b8478..127e7af2 100755
--- a/install_base.sh
+++ b/install_base.sh
@@ -153,15 +153,15 @@ apt_packages=(

 # pacman-based distributions like Archlinux or its derivatives
 pacman_packages=(
-    coreutils
-    git
-    openssh
-    base-devel
-    wget
-    unzip
-    python
-    python-pip
-    python-setuptools
+    #coreutils
+    #git
+    #openssh
+    #base-devel
+    #wget
+    #unzip
+    #python
+    #python-pip
+    #python-setuptools

     # These packages can be installed from AUR
     # kernelshark
@@ -233,13 +233,13 @@ for arg in "$@"; do

     # TODO: remove --install-android-sdk, since it is only temporarily there to
     # give some time to migrate CI scripts
-    "--install-android-sdk" | "--install-android-tools" | "--install-all")
+    "--install-android-sdk" | "--install-android-tools")
         install_functions+=(
             install_android_sdk_manager # Needed by install_android_build_tools
             install_android_tools
         )
         apt_packages+=(openjdk-$ANDROID_SDK_JAVA_VERSION-jre openjdk-$ANDROID_SDK_JAVA_VERSION-jdk)
-        pacman_packages+=(jre$ANDROID_SDK_JAVA_VERSION-openjdk jdk$ANDROID_SDK_JAVA_VERSION-openjdk)
+        #pacman_packages+=(jre$ANDROID_SDK_JAVA_VERSION-openjdk jdk$ANDROID_SDK_JAVA_VERSION-openjdk)
```

```
diff --git a/external/devlib/setup.py b/external/devlib/setup.py
index df3377d0..b6af03df 100644
--- a/external/devlib/setup.py
+++ b/external/devlib/setup.py
@@ -84,11 +84,9 @@ params = dict(
         'pyserial',  # Serial port interface
         'wrapt',  # Basic for construction of decorator functions
         'future', # Python 2-3 compatibility
-        'enum34;python_version<"3.4"', # Enums for Python < 3.4
-        'contextlib2;python_version<"3.0"', # Python 3 contextlib backport for Python 2
-        'numpy<=1.16.4; python_version<"3"',
+        'enum34', # Enums for Python < 3.4
+        'contextlib2', # Python 3 contextlib backport for Python 2
         'numpy; python_version>="3"',
-        'pandas<=0.24.2; python_version<"3"',
         'pandas; python_version>"3"',
     ],
     extras_require={
```

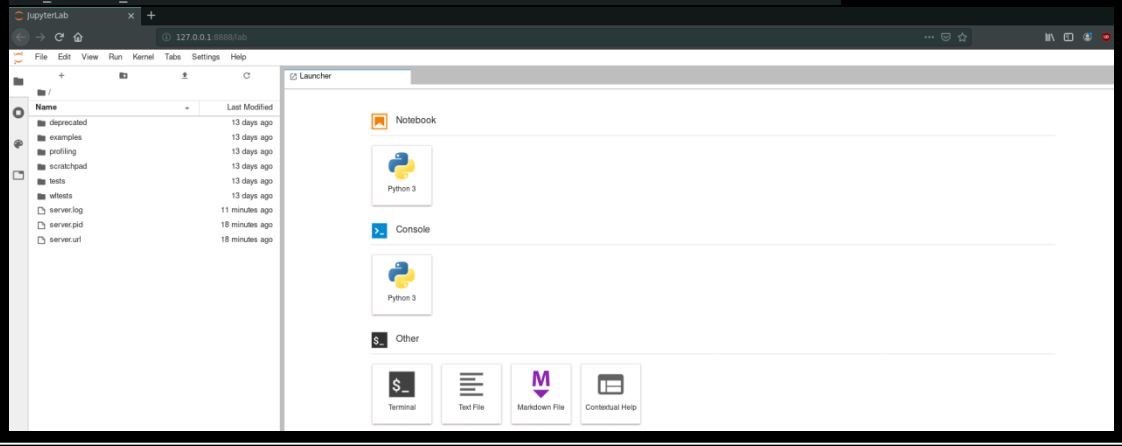
```
diff --git a/external/trappy/setup.py b/external/trappy/setup.py
index b750aae4..bf0566a1 100644
--- a/external/trappy/setup.py
+++ b/external/trappy/setup.py
@@ -66,7 +66,6 @@ setup(name='TRAPpy',
           "Environment :: Console",
           "License :: OSI Approved :: Apache Software License",
           "Operating System :: POSIX :: Linux",
-          "Programming Language :: Python :: 2.7",
           "Programming Language :: Python :: 3",
           # As we depend on trace data from the Linux Kernel/FTrace
           "Topic :: System :: Operating System Kernels :: Linux",
```

```
diff --git a/external/workload-automation/setup.py b/external/workload-automation/setup.py
index 36c697c6..45380032 100755
--- a/external/workload-automation/setup.py
+++ b/external/workload-automation/setup.py
@@ -80,7 +80,6 @@ params = dict(
     maintainer='ARM Architecture & Technology Device Lab',
     maintainer_email='workload-automation@arm.com',
     setup_requires=[
-        'numpy<=1.16.4; python_version<"3"',
         'numpy; python_version>="3"',
     ],
     install_requires=[
@@ -93,7 +92,6 @@ params = dict(
         'devlib>={}'.format(devlib_version),  # Interacting with devices
         'louie-latest',  # callbacks dispatch
         'wrapt',  # better decorators
-        'pandas>=0.23.0,<=0.24.2; python_version<"3.5.3"',  # Data analysis and manipulation
         'pandas>=0.23.0; python_version>="3.5.3"',  # Data analysis and manipulation
         'future',  # Python 2-3 compatiblity
     ],
```

start the Jupyter Notebook server
cd $LISA_SRC; source init_env

```
                    .:: LISA Shell ::.

Activating LISA Python 3.7 venv (/opt/MyWorkSpace/MyProjs/ARM/LISA/lisa-master/.lisa-venv-3.7) ...

Welcome to the Linux Integrated System Analysis SHELL!

LISA version:
    /opt/MyWorkSpace/MyProjs/ARM/LISA/lisa-master
    branch: heads/master
    commit: 25b7d70d339

Python version: 3.7

   app dir: /opt/MyWorkSpace/MyProjs/ARM/LISA/lisa-master/.lisa-venv-3.7/share/jupyter/lab
        @jupyter-widgets/jupyterlab-manager v1.0.2  enabled  OK
        jupyter-matplotlib v0.4.2  enabled  OK
JupyterLab v1.1.4
Known labextensions:

Type "man lisa" for help on available commands

Environment variables:
# Default root for exekall's artifacts
EXEKALL_ARTIFACT_ROOT=/opt/MyWorkSpace/MyProjs/ARM/LISA/lisa-master/results

# Default configuration file for exekall
LISA_CONF=/opt/MyWorkSpace/MyProjs/ARM/LISA/lisa-master/target_conf.yml

# By default use internal libraries
LISA_DEVMODE=1

# Base directory of LISA environment
LISA_HOME=/opt/MyWorkSpace/MyProjs/ARM/LISA/lisa-master

# Add some shell utilities to the PATH, based on the host ABI. Priority
# is determined by LISA_USE_SYSTEM_BIN
LISA_HOST_ABI=arm64

# By default use Lisa's PS1 and colorscheme for the shell
LISA_PRESERVE_SHELL=0
```

```
# Python binary to use. This allows users to install multiple versions
# in parallel, and ease testing
LISA_PYTHON=python3

# By default, store the results under that folder
LISA_RESULT_ROOT=/opt/MyWorkSpace/MyProjs/ARM/LISA/lisa-master/results

# Use the system binaries if 1, will use the ones shipped with LISA if 0
LISA_USE_SYSTEM_BIN=0

# 1 to make lisa-install use a venv specified in LISA_VENV_PATH, 0
# otherwise
LISA_USE_VENV=1

# Path to venv to be used by lisa-install
LISA_VENV_PATH=/opt/MyWorkSpace/MyProjs/ARM/LISA/lisa-master/.lisa-venv-3.7

[LISAShell lisa-master] \> lisa-jupyter

Starting Jupyter Notebooks...
Server already running:
   http://127.0.0.1:8888/?token=E5526ef0FFDFF765365d7DD6Cb4573CFfCEd8fe4FcEa4238
```



2.3 Drgn

- https://drgn.readthedocs.io/en/latest/installation.html

build & install

- build **libkdumpfile** if you want support for kdump-compressed kernel core dumps //Optional

  cd $DRGN_SRC

  python setup.py build

  sudo python setup.py install

run an example

-

```
[myrpi4@myrpi4h1 Drgn]$ sudo drgn
could not get debugging information for:
/boot/vmlinux-5.3.5-MANJARO-ARM+: not an ELF file
drgn 0.0.1 (using Python 3.7.4, with libkdumpfile)
For help, type help(drgn).
>>> import drgn
>>> from drgn import cast, container_of, execscript, NULL, Object, reinterpret
>>> from drgn.helpers.linux import *
>>> prog['init_task'].comm
Traceback (most recent call last):
  File "<console>", line 1, in <module>
KeyError: 'init_task'
>>>
```

Solution

dwarf_index.c (libdrgn) | dwarf_index.h (libdrgn) | linux_kernel.c (libdrgn)

```
} « end report_kernel_modules »

static struct drgn_error *
report_default_vmlinux(struct drgn_program *prog,
                       struct drgn_dwarf_index *dindex,
                       bool *vmlinux_is_pending)
{
    static const char * const vmlinux_paths[] = {
        /*
         * The files under /usr/lib/debug should always have debug
         * information, so check for those first.
         */
        "/usr/lib/debug/boot/vmlinux-%s",
        "/usr/lib/debug/lib/modules/%s/vmlinux",
        "/boot/vmlinux-%s",
        "/lib/modules/%s/build/vmlinux",
        "/lib/modules/%s/vmlinux",
        NULL,
    };
```

$RPILINUX_SRC

```
drwxr-xr-x  26 myrpi4 myrpi4      4096 Oct 15 23:59 sound/
-rw-r--r--   1 myrpi4 myrpi4   3657102 Oct 16 01:12 System.map
-rw-r--r--   1 myrpi4 myrpi4   1458280 Oct 16 01:11 .tmp_kallsyms1.o
-rw-r--r--   1 myrpi4 myrpi4   8560443 Oct 16 01:11 .tmp_kallsyms1.S
-rw-r--r--   1 myrpi4 myrpi4   1458280 Oct 16 01:12 .tmp_kallsyms2.o
-rw-r--r--   1 myrpi4 myrpi4   8560443 Oct 16 01:12 .tmp_kallsyms2.S
-rw-r--r--   1 myrpi4 myrpi4   3657102 Oct 16 01:12 .tmp_System.map
-rwxr-xr-x   1 myrpi4 myrpi4 222986776 Oct 16 01:11 .tmp_vmlinux1*
-rwxr-xr-x   1 myrpi4 myrpi4 224428928 Oct 16 01:12 .tmp_vmlinux2*
drwxr-xr-x  36 myrpi4 myrpi4      4096 Oct 15 23:26 tools/
drwxr-xr-x   3 myrpi4 myrpi4      4096 Oct 15 23:34 usr/
-rw-r--r--   1 myrpi4 myrpi4         2 Oct 16 01:10 .version
drwxr-xr-x   4 myrpi4 myrpi4      4096 Oct 16 00:07 virt/
-rwxr-xr-x   1 myrpi4 myrpi4 224428928 Oct 16 01:12 vmlinux*
-rw-r--r--   1 myrpi4 myrpi4       134 Oct 16 01:12 .vmlinux.cmd
-rw-r--r--   1 myrpi4 myrpi4 582146608 Oct 16 01:11 vmlinux.o
```

sudo mkdir –p /usr/lib/debug/boot
sudo cp $RPILINUX_SRC/linux/vmlinux /usr/lib/debug/boot/vmlinux-5.3.5-MANJARO-ARM+

```
[myrpi4@myrpi4h1 /]$ sudo drgn
[sudo] password for myrpi4:
drgn 0.0.1 (using Python 3.7.4, with libkdumpfile)
For help, type help(drgn).
>>> import drgn
>>> from drgn import cast, container_of, execscript, NULL, Object, reinterpret
>>> from drgn.helpers.linux import *
>>> prog['init_task'].comm
(char [16])"swapper/0"
```

```
>>> from drgn.helpers.linux import list_for_each_entry
>>> for mod in list_for_each_entry('struct module',prog['modules'].address_of_(),'list'):
...     if mod.refcnt.counter > 10:
...         print(mod.name)
...
(char [56])"rfcomm"
(char [56])"bluetooth"
(char [56])"ipv6"
>>>
```

2.4 Rethink extending LISA

Limitations of LISA (from my point of view)

- Bias towards automated testing, which has overlap in functionality with LAVA(https://www.lavasoftware.org/) from Linaro in some extent
- Android-oriented, while it is required to adapt to a much more generalized Linux system
- Lack of plug-in mechanism to integrated existed Kernel analysis and test projects

…

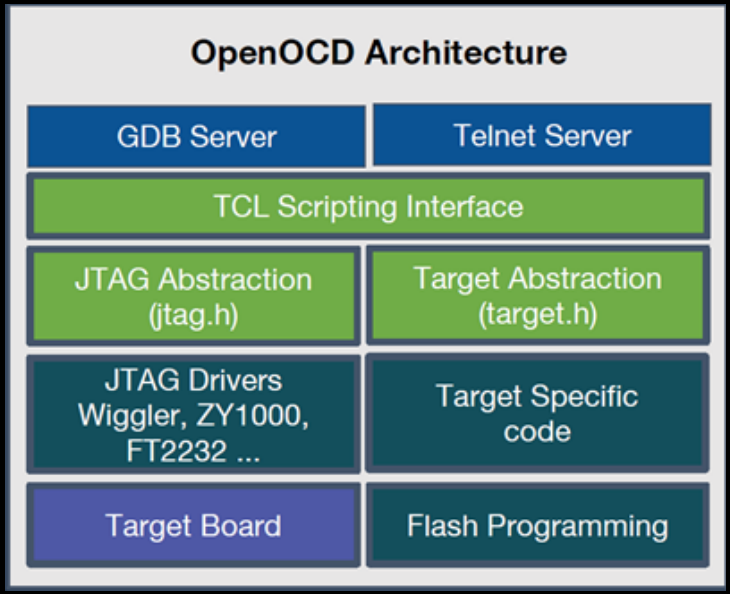
LISA has provided a good base framework and starting point for integrated Linux Kernel analysis.

Try to extend LISA in the following ways

- In addition to Ftrace, combining with eBPF support
- Architecture redesign (for better support existing Kernel analysis and test utilities)

  e.g. flexible plug-in mechanism and well-designed interfaces for system extension…

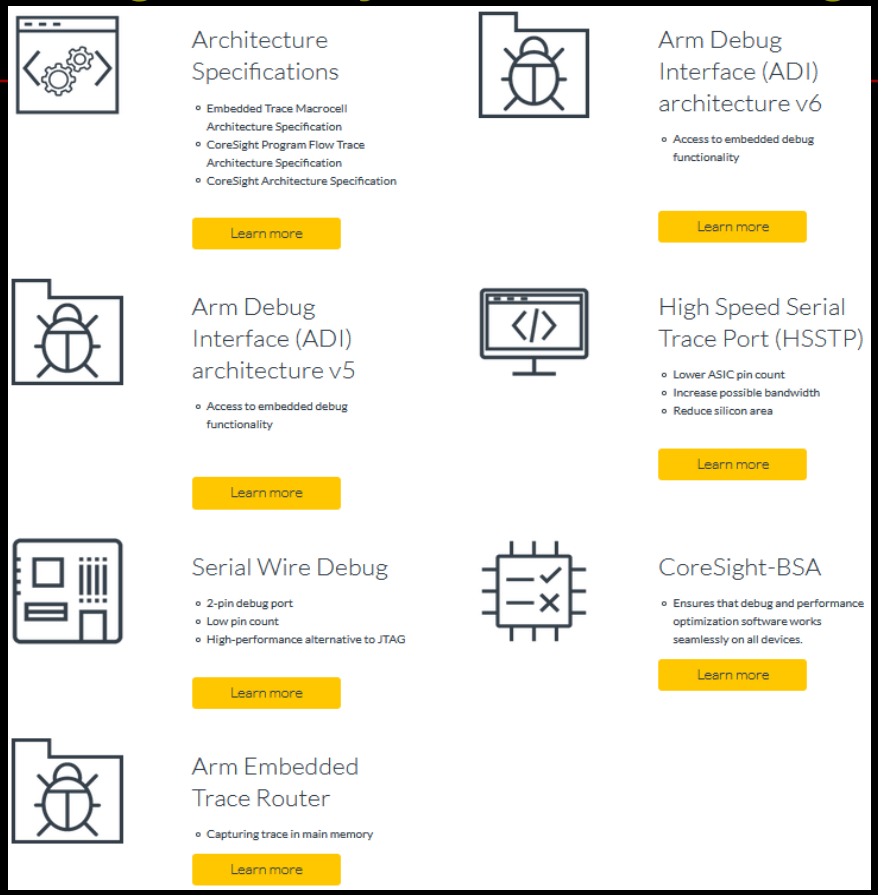
- **Hardware-assisted debugging**
  1. better support hardware debug interface like JTAG and SWD
     https://en.wikipedia.org/wiki/JTAG
     https://www.arm.com/files/pdf/Serial_Wire_Debug.pdf
  2. integrate good on-chip debugger like OpenOCD



Source: "OpenOCD support for AArch64 targets", Omair Javaid, Linaro Connect HK 2018

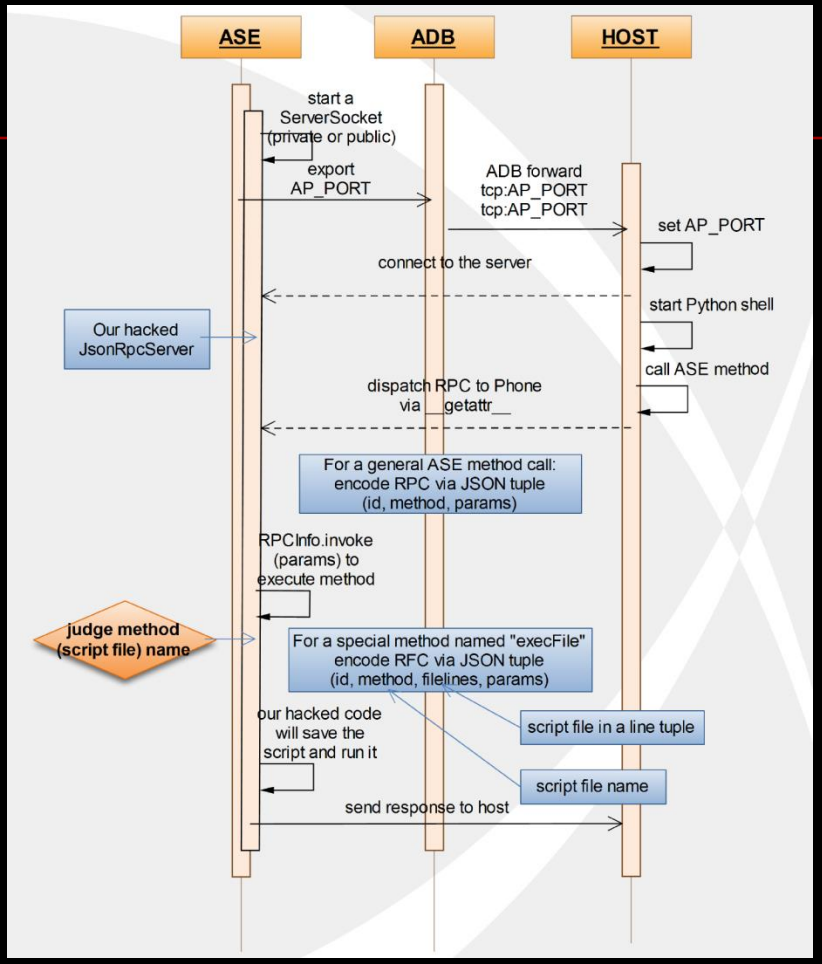
  3. support more mechanism like like Semihosting
     https://developer.arm.com/architectures/learn-the- architecture/debugger-usage/semihosting

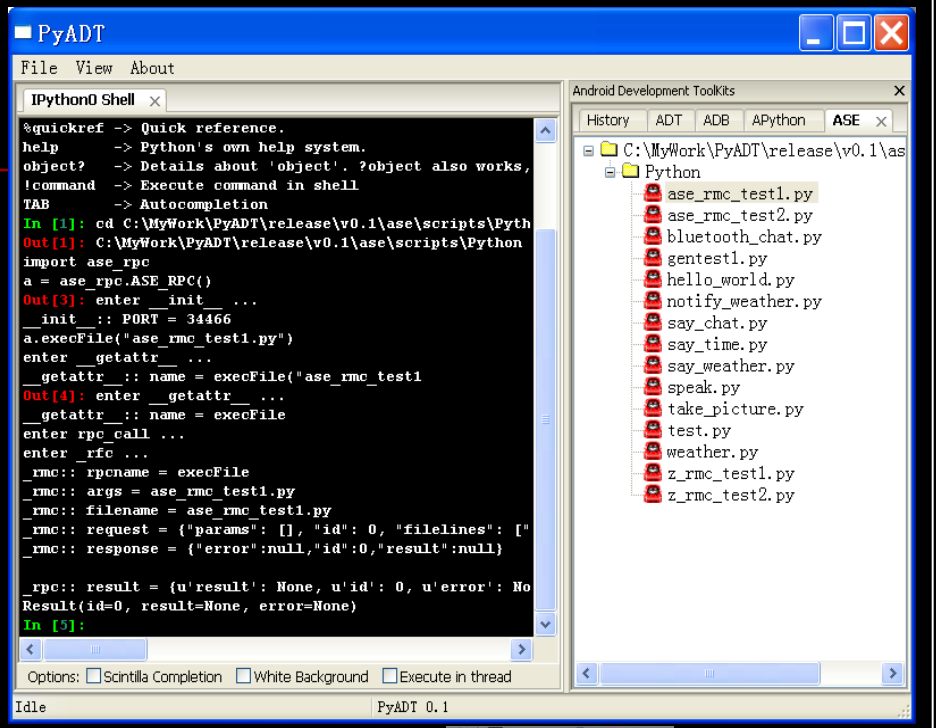
- **Full CoreSight support**
  https://developer.arm.com/architectures/cpu-architecture/debug-visibility-and-trace/coresight-architecture

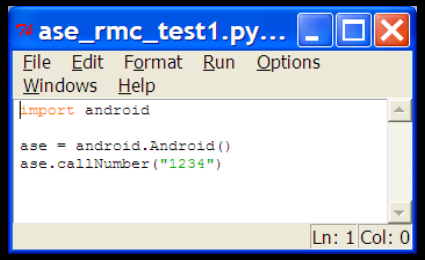


- **The new project LISA2 is hosted at:**
  https://github.com/Yuwenfeng2019/LISA2

- new interactive method for eBPF-based tracing with BCC
  my old toy “PyADT”



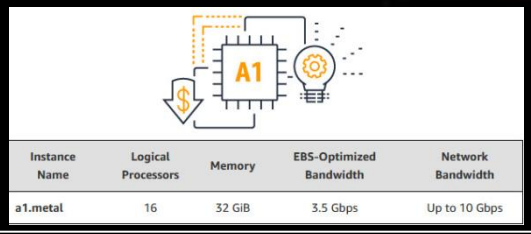


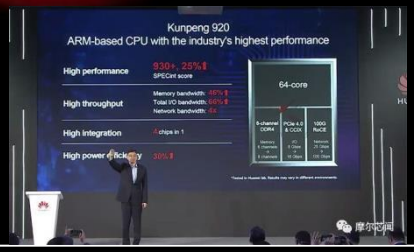


base on ASE, a.k.a SL4A (Scripting Layer for Android)
https://github.com/aosp-caf-upstream/platform_external_sl4a

V. Wrap-up

- Hacking everything with python™ and eBPF!





Q & A

Thanks!

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- https://www.python.org
- http://llvm.org
- http://www.brendangregg.com/
- https://en.wikipedia.org/wiki/Just-in-time_compilation
- http://dwarfstd.org/
- …