LINUX KERNEL E-BPF: CONCEPTS AND USE CASES

Kiran Divekar

25th August 2024

ABOUT ME

- BE Computer from COEP Pune.
- 20 + years of IT industry experience working mainly on Linux Kernel,

  System programming in various domains

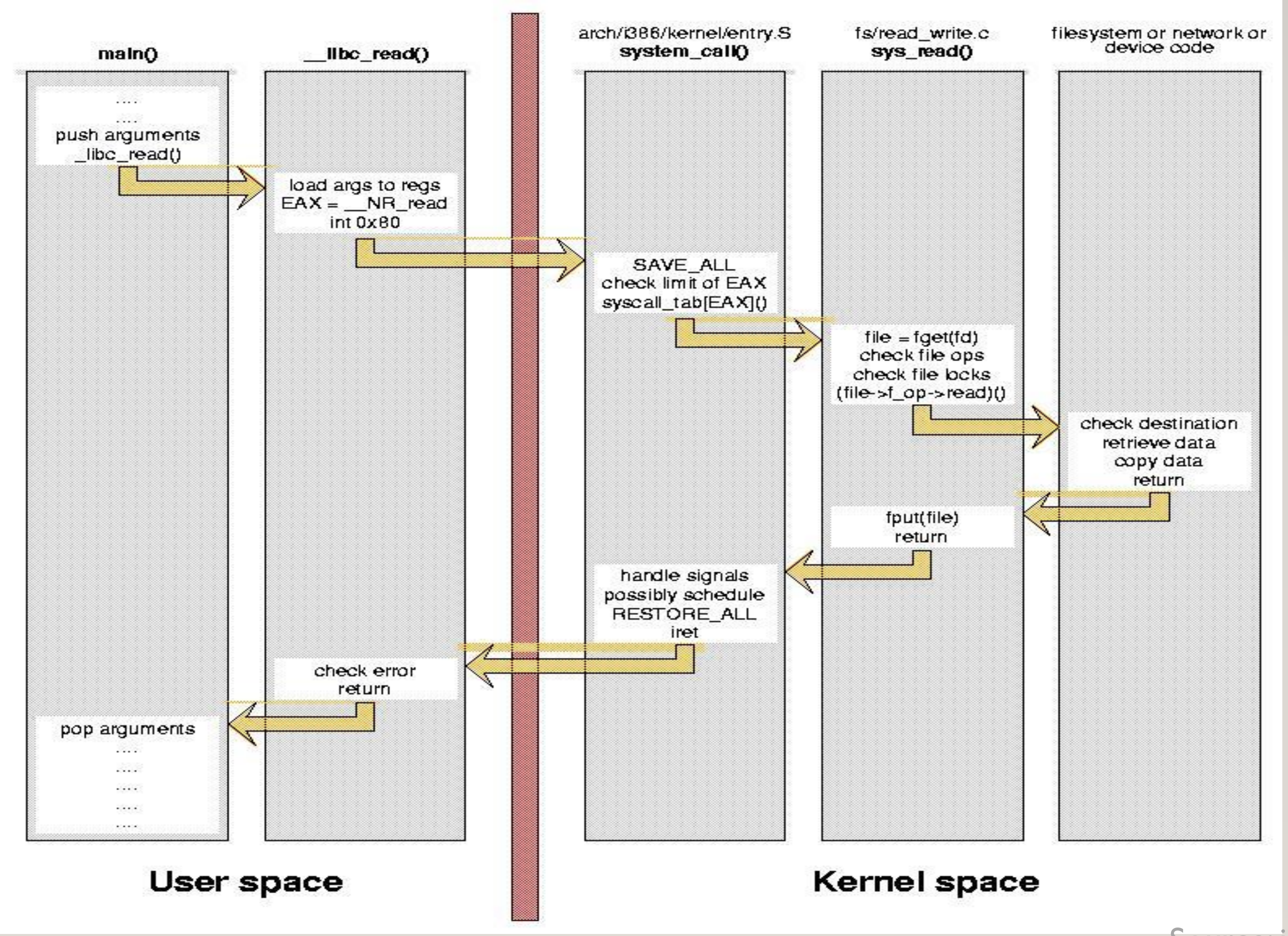
- Core Contributor in Linux kernel 2.6.36 development cycle (Year: 2010)
- Delivered sessions in many open source events, meetups, Foss.in, Nasscom webinar.

- Many open source events OpenStack Boston (2017), DockerCon San Franciso (2018), VMWorld, Cisco Live, Kubernetes Forum Banglore 2020

20 JULY 2024

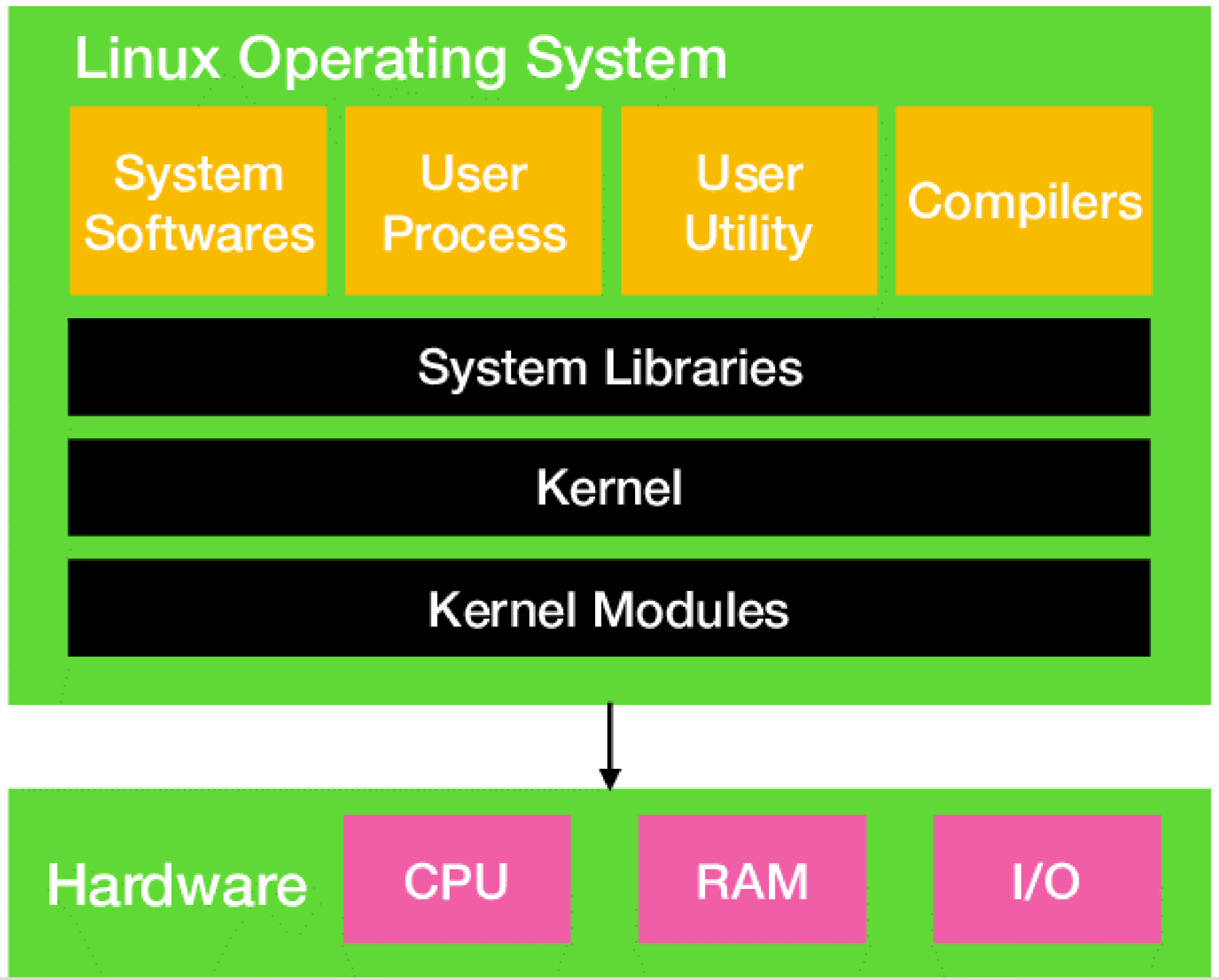
Source: internet

BASICS: LINUX SYSTEM CALLS



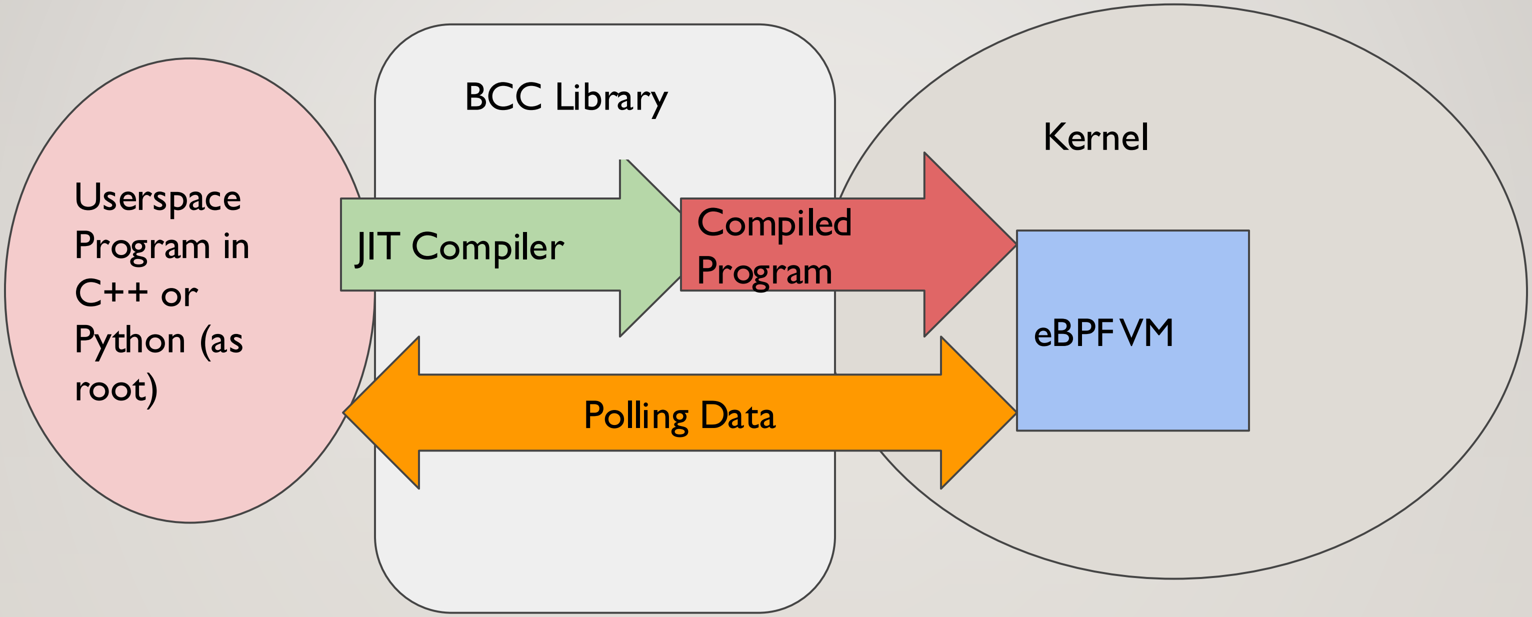
Source: internet

LINUX KERNEL MODULES



Source: internet

EBPF PROGRAMS



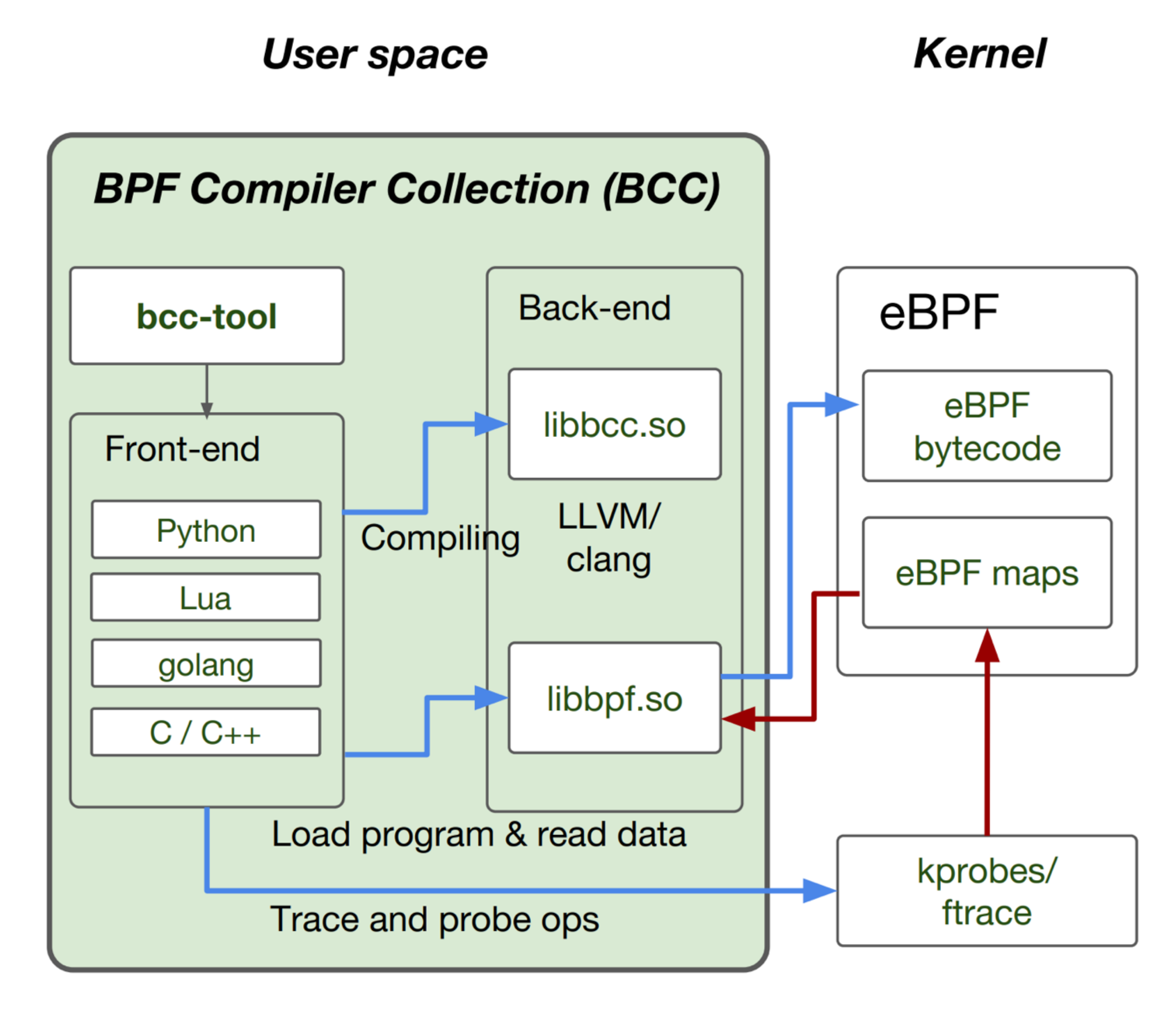
Source: internet

EBPF: IN-KERNEL VM

- BPF: Original network packet filtering, but extended ability to call kernel functions (eBPF) - 3.15 kernel
- eBPF : subset of C and compiled into bytecode
- eBPF sandbox. Copy data into sandbox and to perf ring buffer (user space reads from the buffers)
- BCC - The BPF Compiler Collection, built with LLVM and Clang
- bpftrace - “a high-level tracing language” for eBPF, similar to awk, can be utilized from command line
- Kernel Code - eBPF’s VM lives in the Linux kernel.

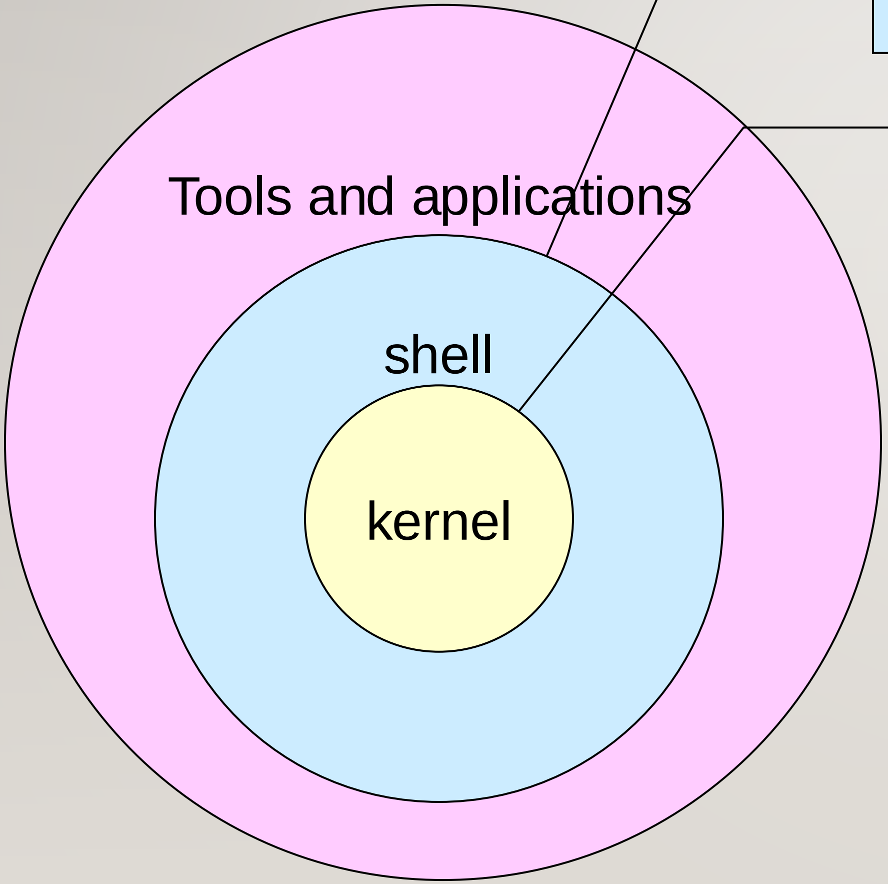
USING IOVISOR BCC

- Load and execute in kernel
- Verifier ensures safety, crash-free.
- JIT compiler: bytecode -> machine code.
- Maps:
  - Sharing of data
  - Ring buffers
  - Hash tables, arrays



Source: internet

Apps vs kmods vs eBPF



- Applications
- Openoffice, chrome etc.

- Command language interpreter
- e.g. Bash shell

Kernel : main OS
- Add kernel module functionality using insmod
-Device drivers, interrupt handling etc.

-eBPF
- Re-Programming the kernel without source code changes or loading modules

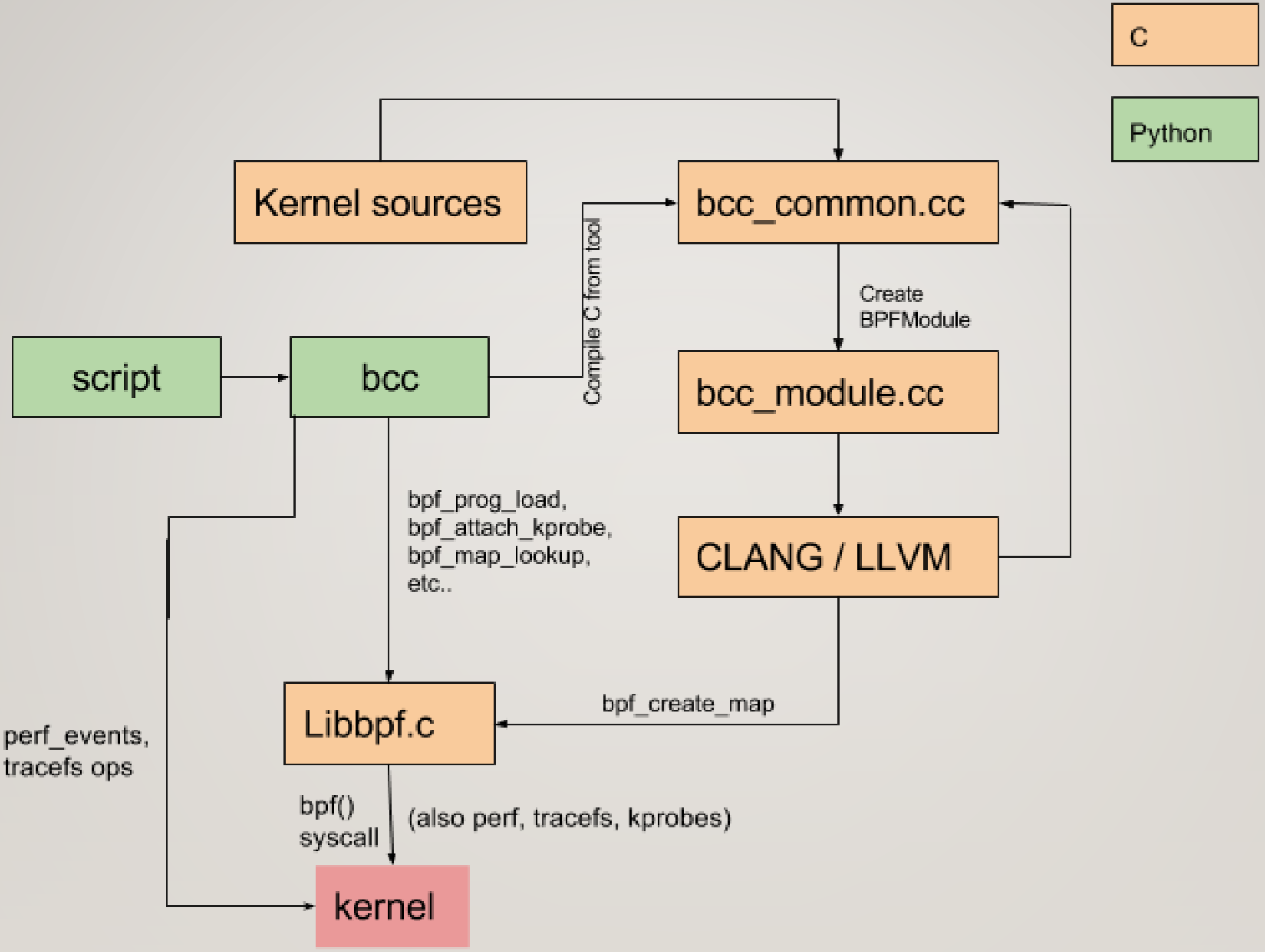
- Observability, traceability

BCC VS LIBBPF

	BCC	Libbpf + CO-RE
1	Clang front-end to modify user-written BPF programs. Difficult to find the problem and figure out a solution	directly use the libbpf library provided by kernel developers to develop BPF programs.
2	Need to remember naming conventions and automatically generated tracepoint structs.	Libbpf acts like a BPF program loader and relocates, loads, and checks BPF programs. BPF developers only need to focus on the BPF programs' correctness and performance.
3	When a tool starts, it takes many CPU and memory resources to compile the BPF program. Complete libraries need to be available and run at compile time.	No need of system-wide dependencies to be present on the target machine for running. It reduces the overall application size as well as resource consumption on runtime.
4	BCC depends on kernel header packages, which you must install on each target host.	Libbpf enables you to generate binaries that are compiled once and can be run anywhere.

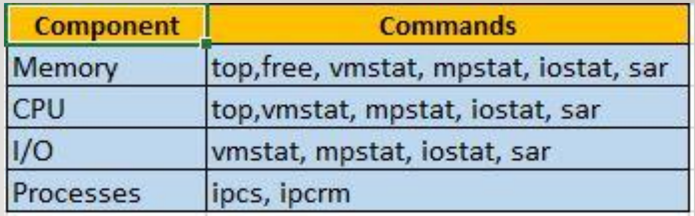
Source :Internet

BCC INTERNALS

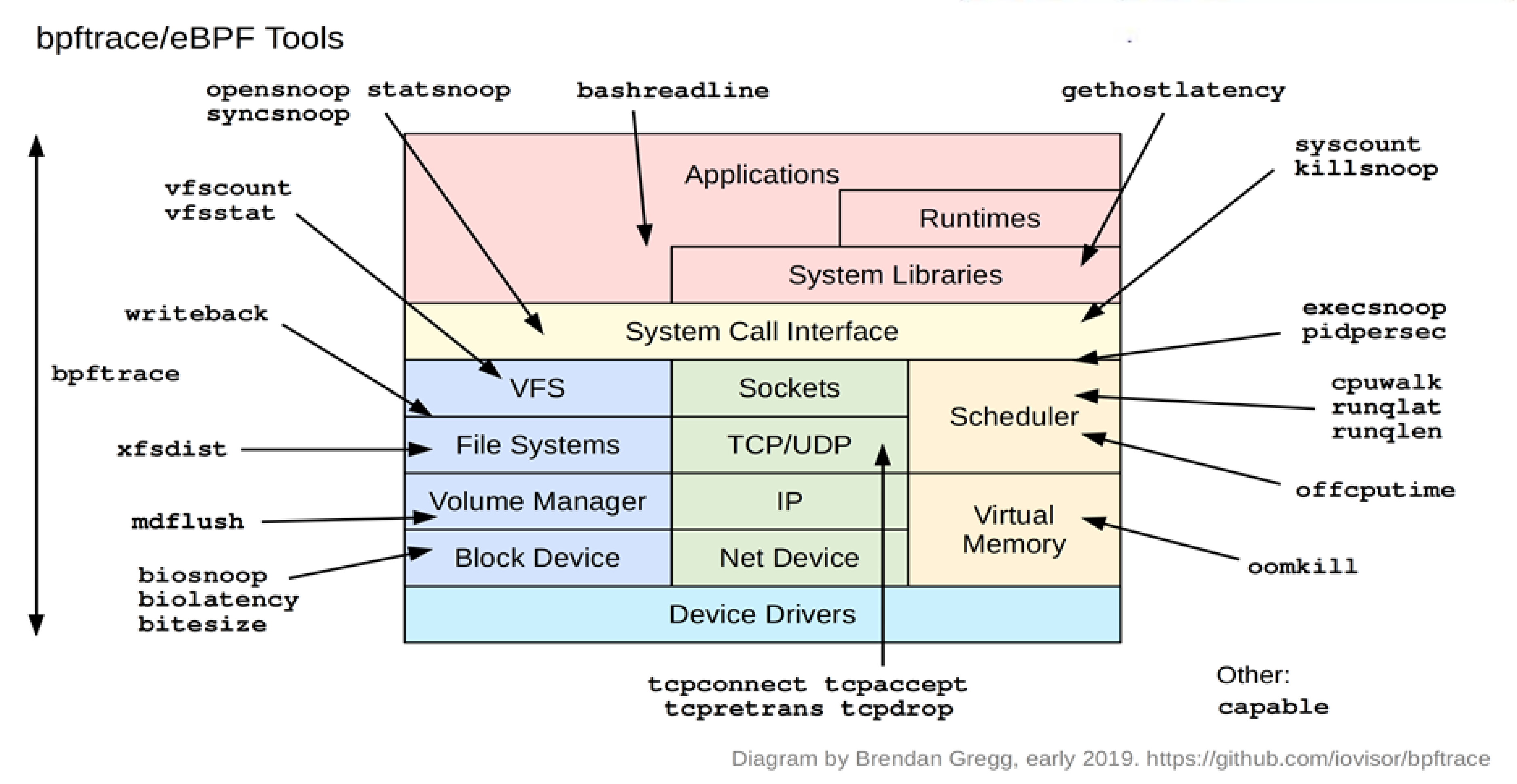


Source :Internet

BPFTRACE TOOLS



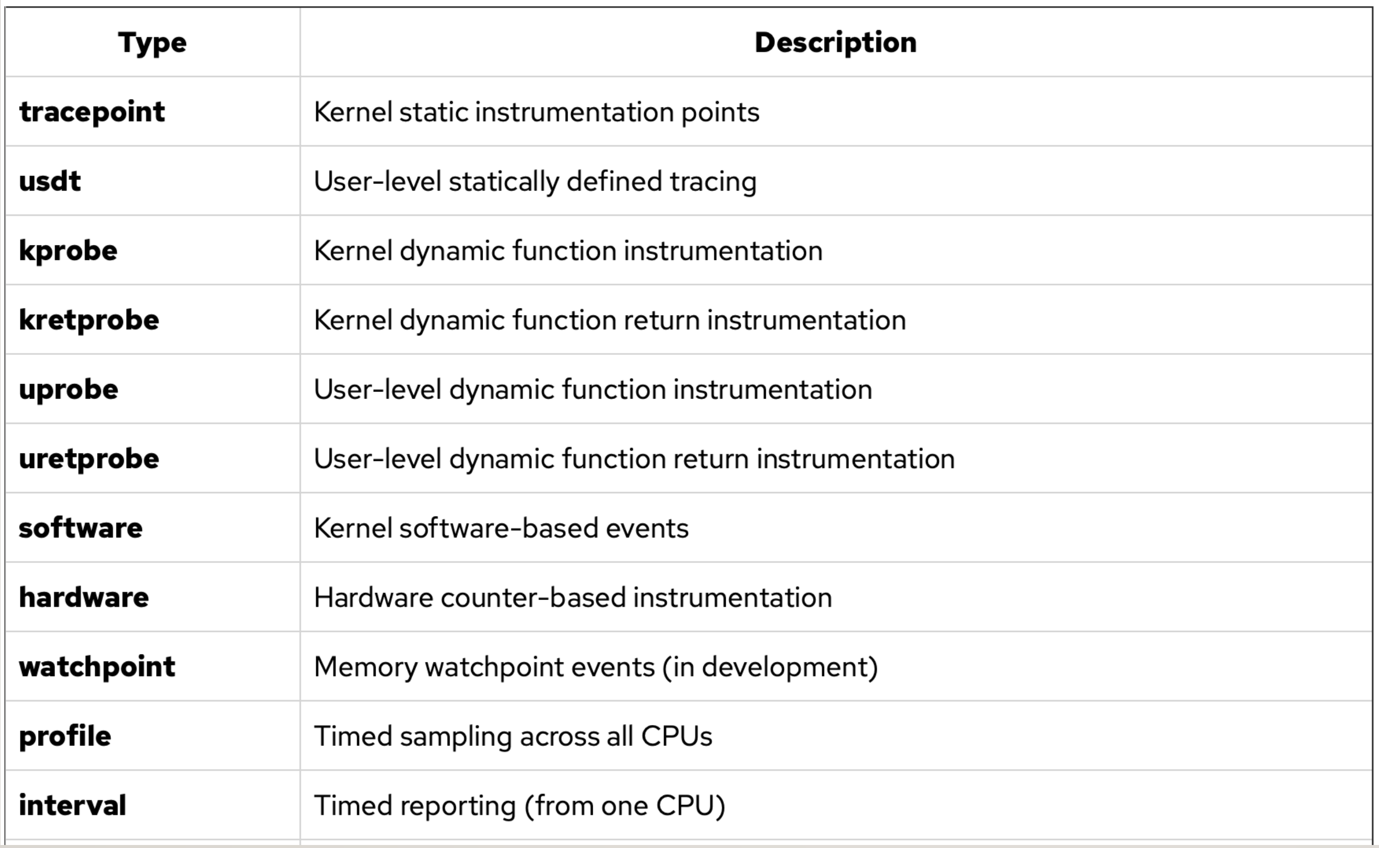
Component	Commands
Memory	top,free, vmstat, mpstat, iostat, sar
CPU	top,vmstat, mpstat, iostat, sar
I/O	vmstat, mpstat, iostat, sar
Processes	ipcs, ipcrm



bpftrace/eBPF Tools

Diagram by Brendan Gregg, early 2019. https://github.com/iovisor/bpftrace

KERNEL INSTRUMENTATION



Type	Description
tracepoint	Kernel static instrumentation points
usdt	User-level statically defined tracing
kprobe	Kernel dynamic function instrumentation
kretprobe	Kernel dynamic function return instrumentation
uprobe	User-level dynamic function instrumentation
uretprobe	User-level dynamic function return instrumentation
software	Kernel software-based events
hardware	Hardware counter-based instrumentation
watchpoint	Memory watchpoint events (in development)
profile	Timed sampling across all CPUs
interval	Timed reporting (from one CPU)

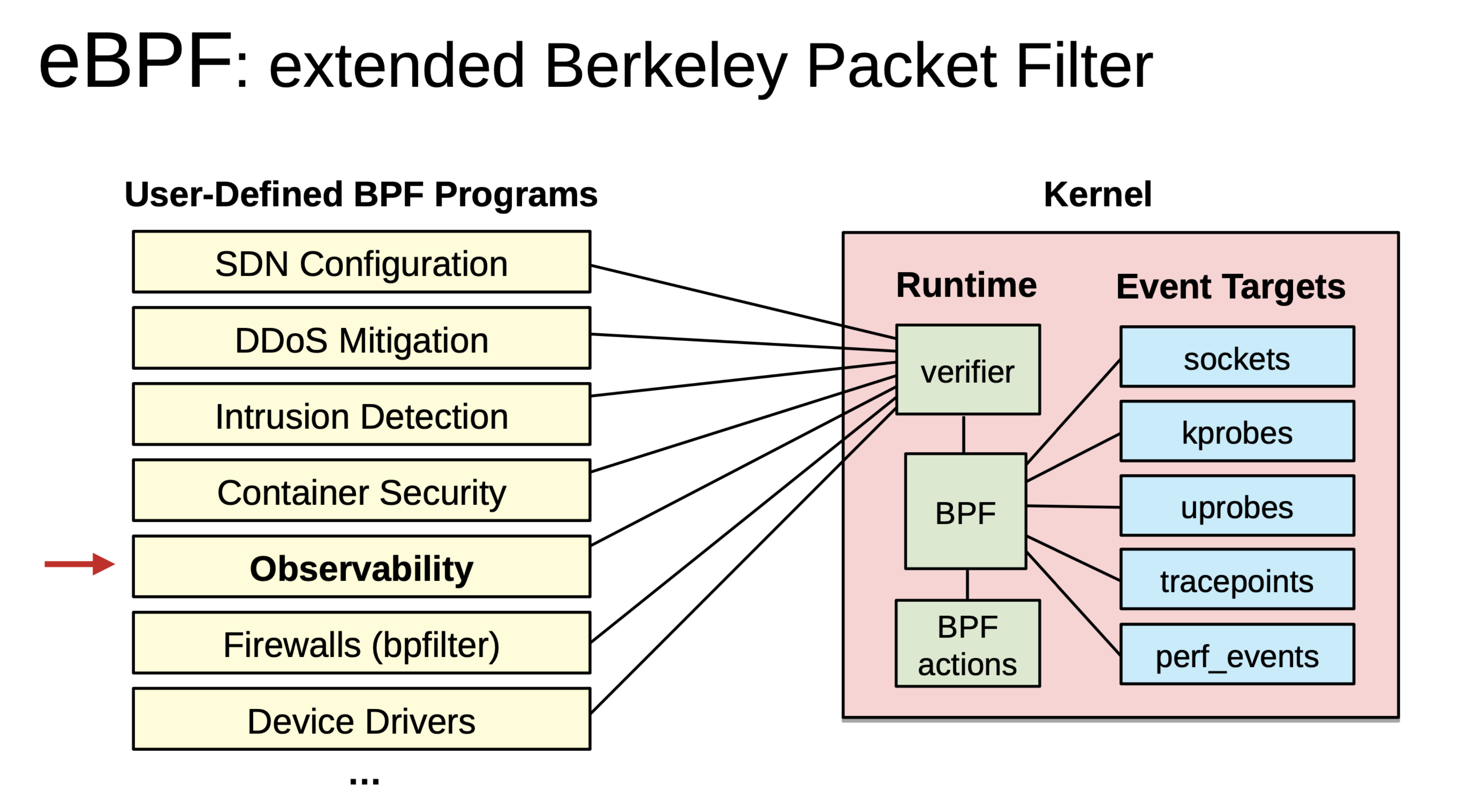
Dynamic

- Kernel space: kprobes
- User space: uprobes

Static:

- Kernel trace points
- USDT (statically defined)

USE CASES



eBPF: extended Berkeley Packet Filter

User-Defined BPF Programs

- SDN Configuration
- DDoS Mitigation
- Intrusion Detection
- Container Security
- **Observability**
- Firewalls (bpfilter)
- Device Drivers
- ...

Kernel

Runtime

- verifier
- BPF
- BPF actions

Event Targets

- sockets
- kprobes
- uprobes
- tracepoints
- perf_events

Source :Internet

EBPF: USE CASES

- Security
  - Intercepting system call, providing process contexts
  - Incident response

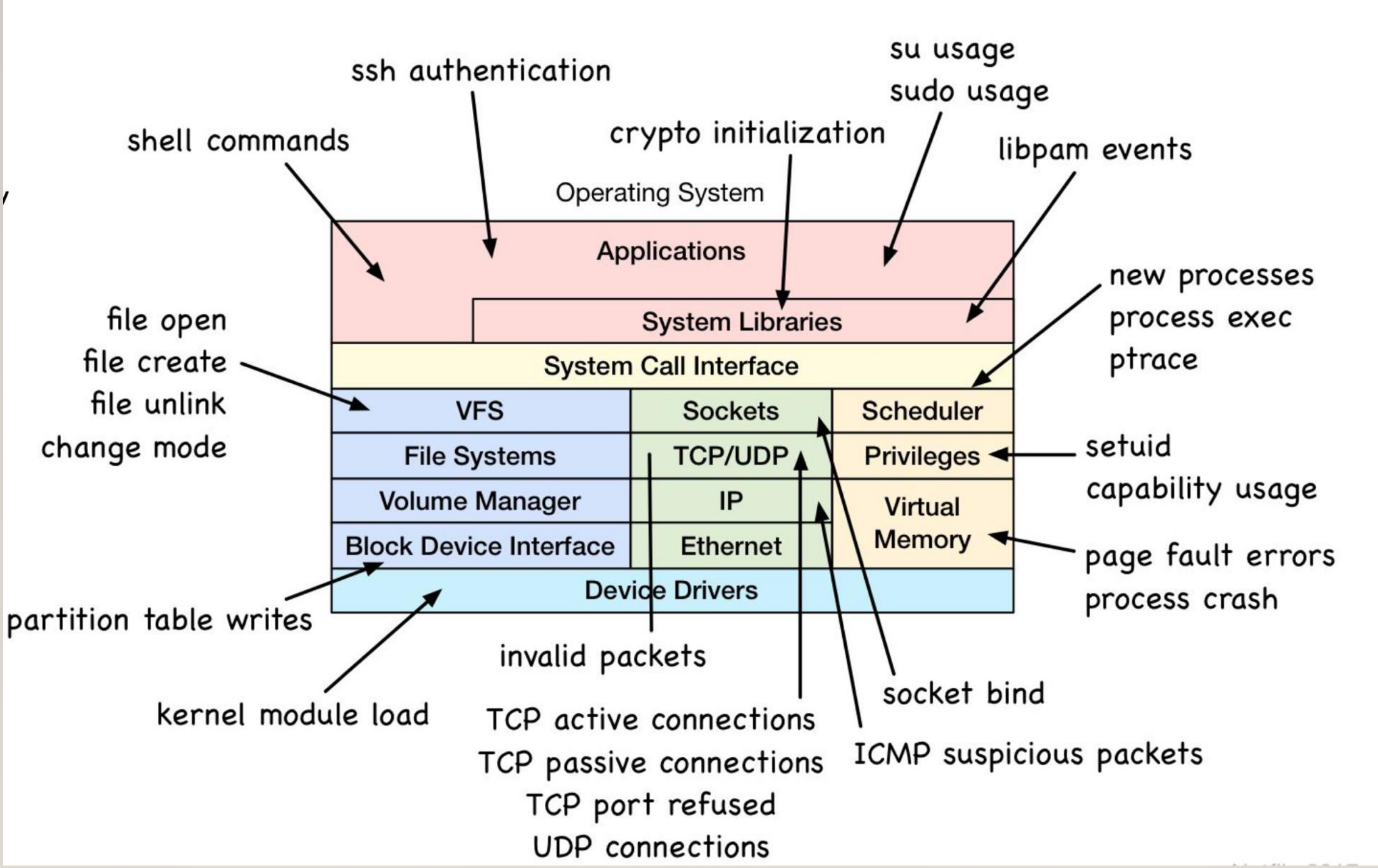
- Monitoring, Observability
  - Increased depth in visibility
  - Aggregation of custom metrics

- Networking
- IDS, IPS, Policy Enforcement
  - Fulfill packet processing requirements
  - Enforce security policy beyond tools like SELinux

- Tracing, Profiling
  - Collection of system data using tracepoints, kprobes etc.

DEMO TIME !!!

MONITORING



Operating System

Applications

System Libraries

System Call Interface

VFS

File Systems

Volume Manager

Block Device Interface

Sockets

TCP/UDP

IP

Ethernet

Scheduler

Privileges

Virtual Memory

Device Drivers

ssh authentication

su usage
sudo usage

shell commands

crypto initialization

libpam events

new processes
process exec
ptrace

file open
file create
file unlink
change mode

setuid
capability usage

page fault errors
process crash

partition table writes

kernel module load

invalid packets

TCP active connections
TCP passive connections
TCP port refused
UDP connections

socket bind

ICMP suspicious packets

Source :Internet

THANK YOU

- *ebpf.io*
- *Presentations from Brenden Greg*
- The Linux Kernel Internals.
- *And many more.*