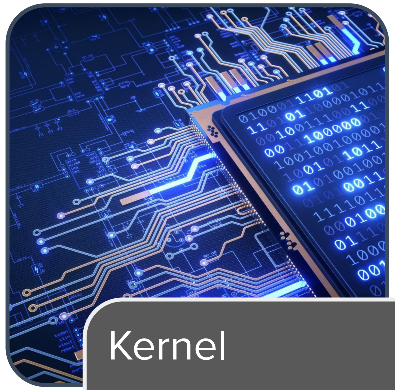
Systems Advanced II Linux

eBPF

DE HOGESCHOOL
MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

Systems Advanced II - Linux



Kernel

- virtual memory
- user space and kernel space
- processes and threads
- sockets & pipes
- systemd
- Docker

Networking

- routing
- firewall
- eBPF
- certificates
- ssh
- latency
- bandwidth

Services

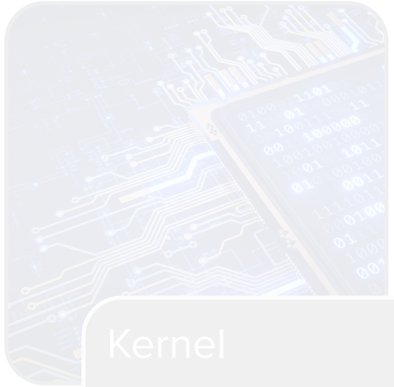
- nfs
- load balancers and reverse proxies
- databases
- logging

Clusters

- Storage
- Kubernetes

Licensing & open-source software

Systems Advanced II - Linux



Kernel

- virtual memory
- user space and kernel space
- processes and threads
- sockets & pipes
- systemd
- Docker

Networking

- routing
- firewall
- eBPF
- certificates
- ssh
- latency
- bandwidth

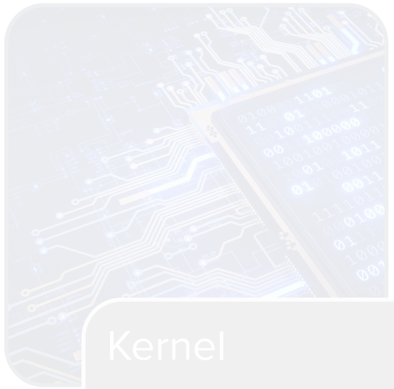
Services

- nfs (rook)
- load balancers and reverse proxies
- databases
- logging

Clusters

- Storage
- Kubernetes

Systems Advanced II - Linux



Kernel

- virtual memory
- user space and kernel space
- processes and threads
- sockets & pipes
- systemd
- Docker

Networking

- routing
- firewall
- eBPF
- certificates
- ssh
- latency
- bandwidth

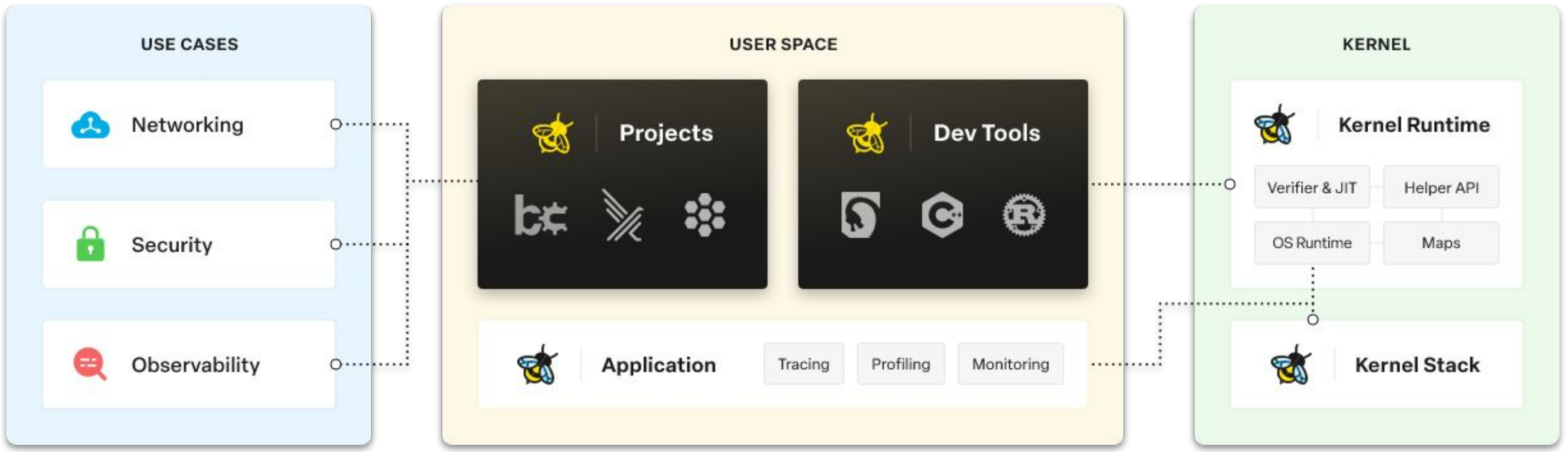
Services

- nfs (rook)
- load balancers and reverse proxies
- databases
- logging

Clusters

- Storage
- Kubernetes

What is eBPF?



- "extended Berkeley Packet Filter"
- Technology to write custom code that can dynamically change the way the Linux kernel behaves.
- platform for building a whole new generation of security, observability, and networking tools.
- Evolved from the Berkeley Packet Filter (BPF), which was originally designed for efficient packet filtering.
- Can now be used to instrument almost any part of the Linux kernel and user space programs.

Kernel Development 101

Option 1
Native Support

- Change kernel source code
- Expose configuration API
- Wait 5 years for your users to upgrade

Cons:

Option 2
Kernel Module

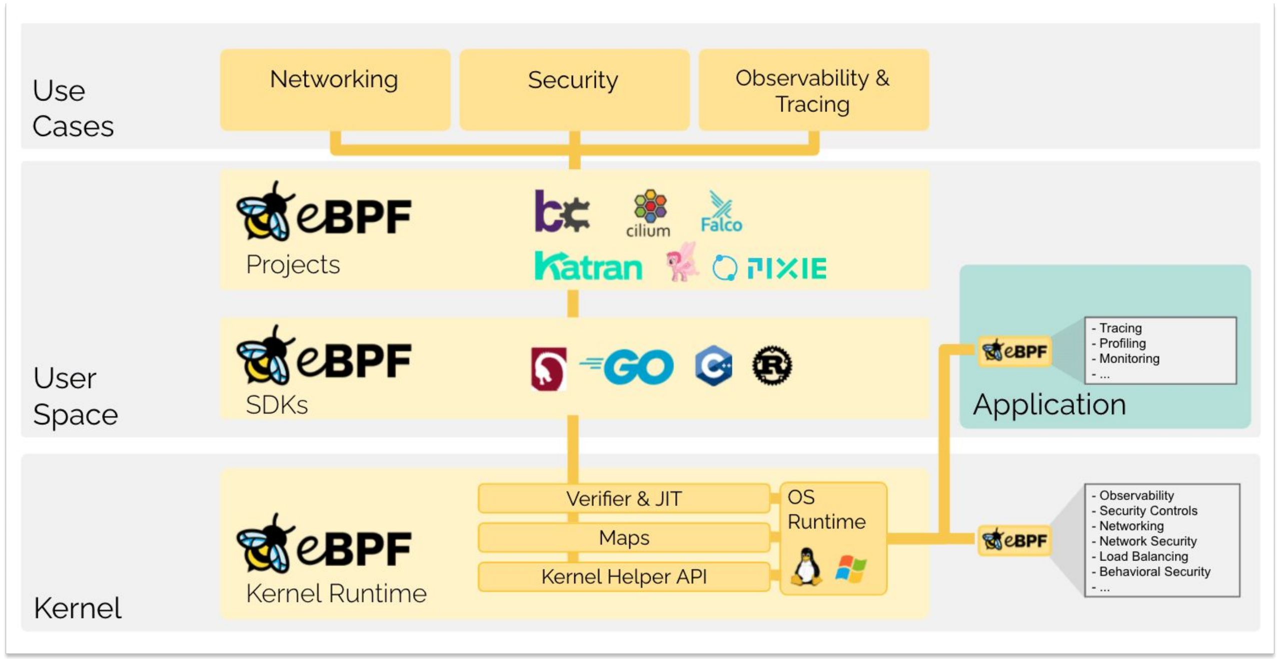
- Write kernel module
- Every kernel release will break it

Cons:

- You likely need to ship a different module for each kernel version
- Might crash your kernel

Why is eBPF important?

- Adds new functionality to Linux kernel without kernel modules and can be dynamically loaded without rebooting or recompiling kernel.
- Can hook anywhere in the kernel to modify functionality
- eBPF programs are **sandboxed** and **verified** for safety.
- eBPF programs are compiled into efficient machine code using JIT compiler, making them high-performance for network packet processing and other performance-critical applications.
- Widely used in cloud native environments, e.g. for advanced network routing and load balancing in Kubernetes clusters.



Organizations in every industry use eBPF in production

Google uses eBPF for security auditing, packet processing, and performance monitoring.

Netflix uses eBPF at scale for network insights.

Android uses eBPF to monitor network usage, power, and memory profiling.

S&P Global uses eBPF through Cilium for networking across multiple clouds and on-prem.

Shopify uses eBPF through Falco for intrusion detection.

Cloudflare uses eBPF for network security, performance monitoring, and network observability.

Ikea uses eBPF through Cilium for networking and load balancing in their private cloud.

Apple uses eBPF through Falco for kernel security monitoring.

Meta uses eBPF to process and load balance every packet coming into their data centers

Datadog uses eBPF for networking and security in their SaaS product

Alibaba uses eBPF through Cilium to provide networking in their cloud

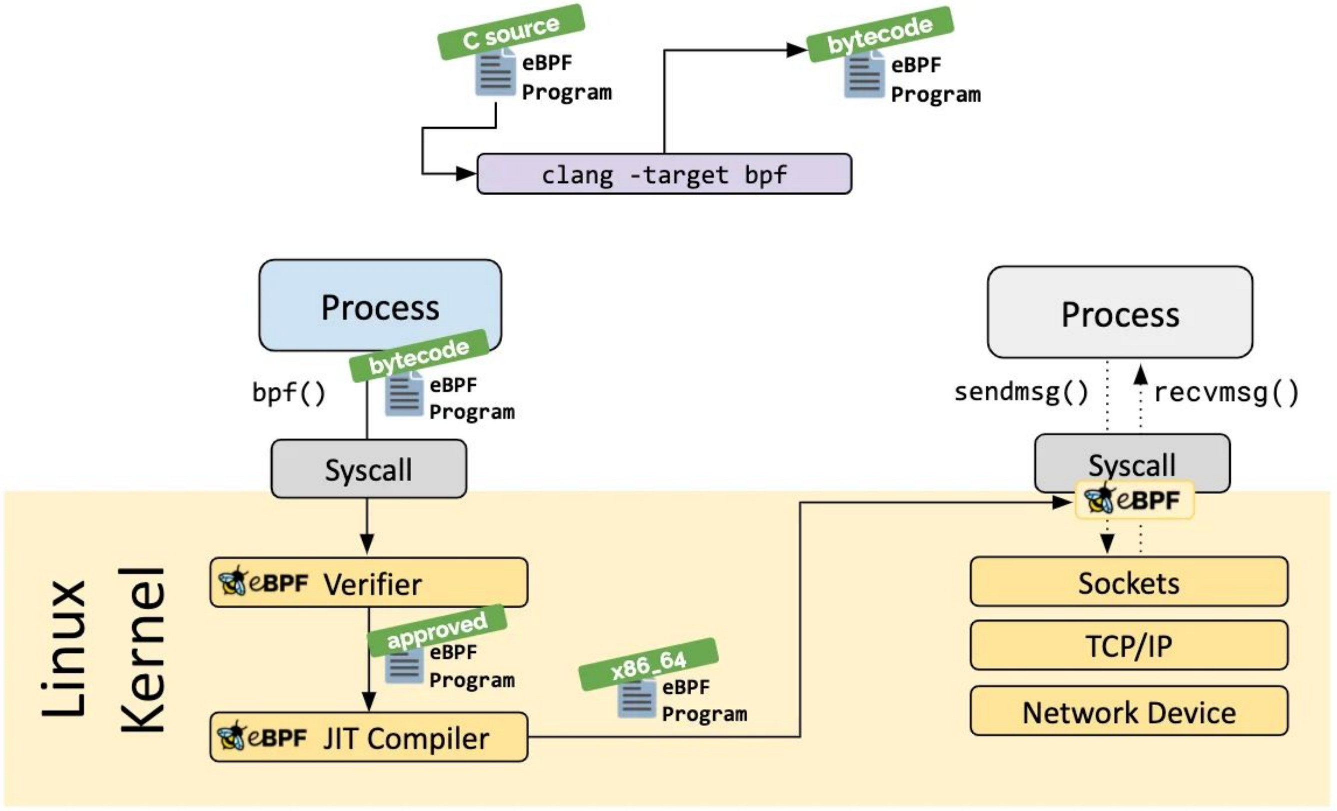
Seznam uses eBPF for load balancing

Examples

- NetFlix
  - https://netflixtechblog.com/how-netflix-uses-ebpf-flow-logs-at-scale-for-network-insight-e3ea997dca96
- Android
  - https://source.android.com/docs/core/architecture/kernel/bpf
- Cloudflare
  - https://blog.cloudflare.com/programmable-packet-filtering-with-magic-firewall/
- Apple
  - https://www.youtube.com/watch?v=ZBlJSr6XkN8
- Cilium
  - https://cilium.io/

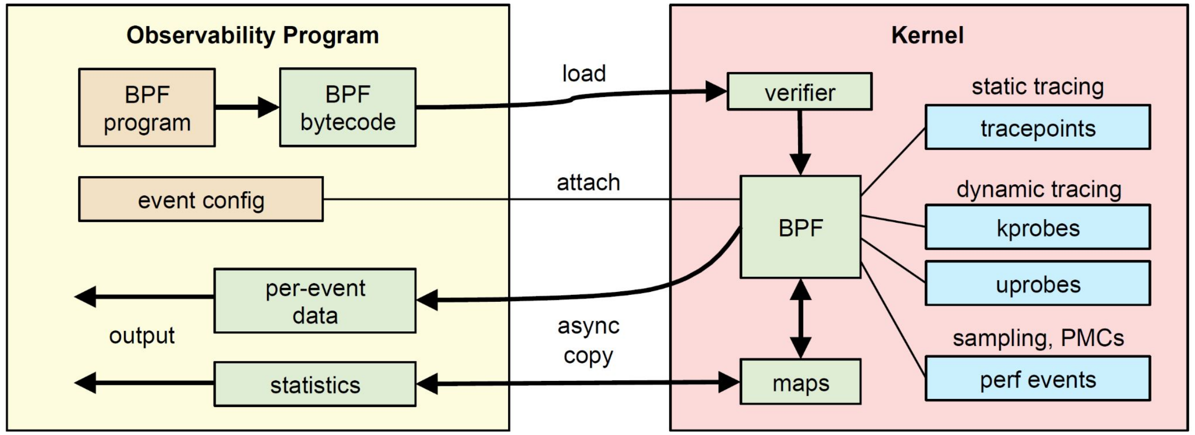
eBPF Components

- Technology that enables custom code to run directly in the Linux kernel.
- eBPF programs are written in a restricted C-like language and compiled into bytecode that runs in the eBPF virtual machine *(see compiler tools slide).*
- Bytecode is loaded into the kernel and attached to various hooks, such as tracepoints, kprobes, and perf events.
- eBPF programs are verified by the kernel 'verifier' before loading to ensure they are safe to run.
- eBPF programs are run whenever the associated hook is triggered.
- eBPF programs can interact with **eBPF maps**, which are key-value data structures for storing and retrieving data.

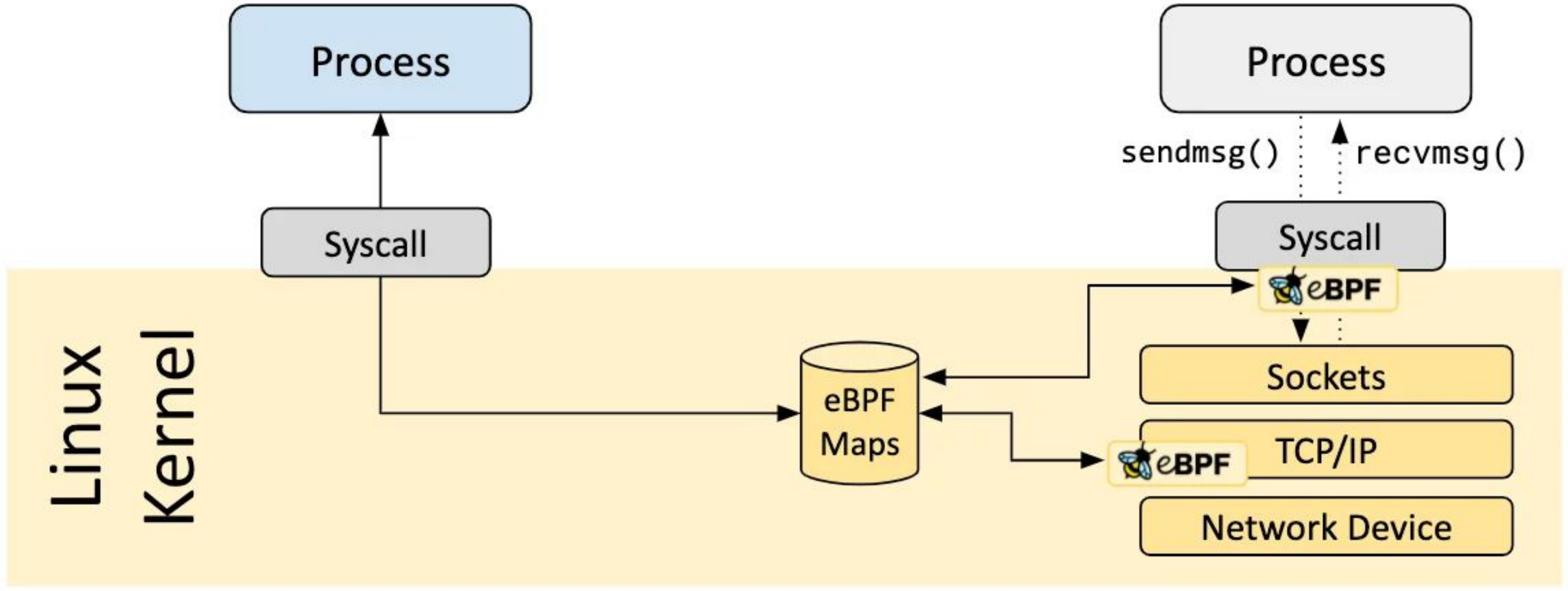


Enhanced Berkeley Packet Filter (BPF, aka eBPF)

– Safe, efficient, advanced, production tracing. Best on Linux 4.9+.



eBPF Maps



- Key-value data structures that allow eBPF programs to store and retrieve data in the kernel space.
- "in-kernel database" that can be accessed by eBPF programs.
- Can be used to share data between different eBPF programs or **between eBPF programs and user space applications**.

E.g. the use of a hash map in a DDoS mitigation system:
The system can use an eBPF program to track the number of requests coming from each IP address, and use an eBPF map to store the count for each IP address. By using a hash map, the system can quickly look up the count for a given IP address and determine if it is making too many requests.

eBPF map types

- **Hash maps**: hash table to store key-value pairs and provide fast access to data.
- **Array maps**: contiguous array of elements to store data and are useful for scenarios where the data is accessed sequentially.
- **Per-CPU maps**: similar to array maps, but provide a separate array for each CPU core in the system. Useful for scenarios where data needs to be accessed and updated concurrently by multiple CPU cores.
- **LRU maps**: maintain a cache of the most recently used items and can be used to implement caching functionality in eBPF programs.
- https://www.kernel.org/doc/html/v6.1-rc5/bpf/maps.html

```
struct bpf_map_def SEC("maps") my_map = {
  .type = BPF_MAP_TYPE_HASH,
  .key_size = sizeof(uint32_t),
  .value_size = sizeof(uint64_t),
  .max_entries = 1024
};

int fd = bpf_map_create(&my_map);
```

E.g. the use of a per-CPU map in a load balancer:
The system can use an eBPF program to track the number of requests being handled by each CPU core, and use a per-CPU map to store the count for each core. By using a per-CPU map, the system can update the count for each core independently and avoid lock contention (== waiting for a shared resource).

eBPF networking

```c
SEC("xdp_drop_all")
int xdp_drop_all(struct xdp_md *ctx) {
  return XDP_DROP;
}
```

eBPF can be used to create different types of networking programs that can intercept and modify network traffic:

- **XDP(eXpress Data Path)** is an eBPF-based high-performance data path used to send and receive network packets at high rates by bypassing most of the operating system networking stack.
  - merged in the Linux kernel and licensed under GPL.
  - Amazon, Google and Intel support its development.
  - Microsoft released XDP for Windows in 2022, licensed under MIT License.
- **TC programs**: These programs are used to filter or modify traffic at different stages in the Linux traffic control (TC) stack.
- **Socket filter programs**: These programs are used to filter incoming or outgoing traffic at the socket layer.

eBPF tracing

eBPF can be used to create different types of tracing programs that can capture and analyze system events:

- **Tracepoints**: These programs are used to trace events that are triggered by the kernel or user space applications.
- **Kprobes**: These programs are used to trace events that occur within the kernel, such as function calls or variable accesses.
- **Uprobes**: These programs are used to trace events that occur within user space applications, such as function calls or system calls.

```
 include <linux/ptrace.h>
 include <linux/sched.h>
 include <uapi/linux/ptrace.h>


int trace_syscall_entry(struct pt_regs *ctx) {
    char *syscall_name = (char *)
bpf_map_lookup_elem(&syscall_map, &ctx->ax);
    if (syscall_name != NULL) {
        bpf_trace_printk("Syscall %s entered\n", syscall_name);
    }
    return 0;
}


SEC("tracepoint/syscalls/sys_enter_execve")
int tracepoint_sys_enter_execve(struct pt_regs *ctx) {
    return trace_syscall_entry(ctx);
}


struct bpf_map_def SEC("maps") syscall_map = {
    .type = BPF_MAP_TYPE_HASH,
    .key_size = sizeof(int),
    .value_size = sizeof(char *),
    .max_entries = 256,
};


char *syscall_names[] = {
    [__NR_execve] = "execve",
    [__NR_exit] = "exit",
    [__NR_read] = "read",

};


__attribute__((constructor))
void syscall_map_init(void) {
    for (int i = 0; i < sizeof(syscall_names) /
sizeof(syscall_names[0]); i++) {
        bpf_map_update_elem(&syscall_map, &i, &syscall_names[i],
BPF_ANY);
    }
}
```

eBPF Tools & Libraries

- Linux Kernel (eBPF runtime)
- llvm compiler (eBPF backend)
- gcc compiler (eBPF backend) - similar to llvm

- rbpf (User Space eBPF runtime)

- eBPF for Windows - *not yet*

Outlook: Future of eBPF

eBPF is turning the Linux kernel into a microkernel.

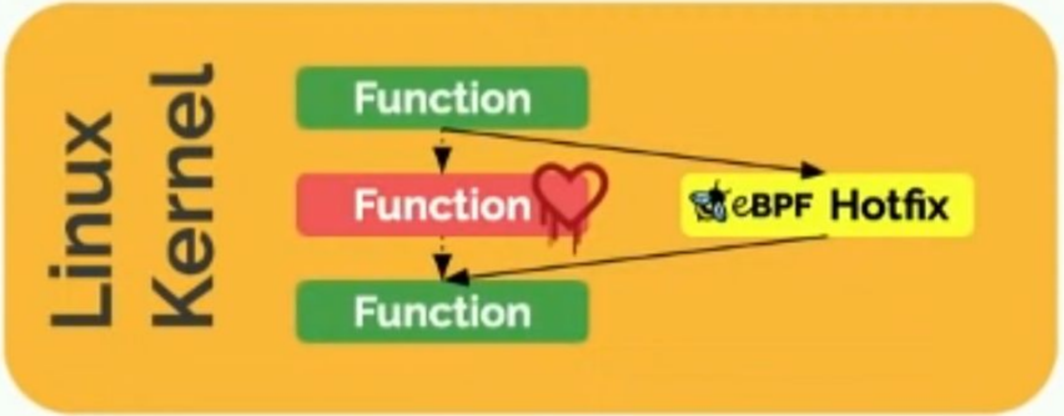
- An increasing amount of new kernel functionality is implemented with eBPF.
- 100% modular and composable.
- New additions can evolve at a rapid pace. Much quicker than normal kernel development.

Example: The linux kernel is not aware of containers and microservices (it only knows about namespaces). Cilium is making the Linux kernel container and Kubernetes aware.

eBPF could enable the Linux kernel hotpatching we always dreamed about.

Problem:

- Linux kernel vulnerability requires to patch kernel.
- Rebooting 20'000 servers takes a very long time without risking extensive downtime.



LAB: Hello, World!

LAB:

https://play.instruqt.com/embed/isovalent/tracks/ebpf-getting-started?token=em_9nxLzhIV41gb3rKM&show_challenges=true

Documentation:

https://www.kernel.org/doc/html/latest/bpf/index.html

Examples:

https://github.com/iovisor/bcc

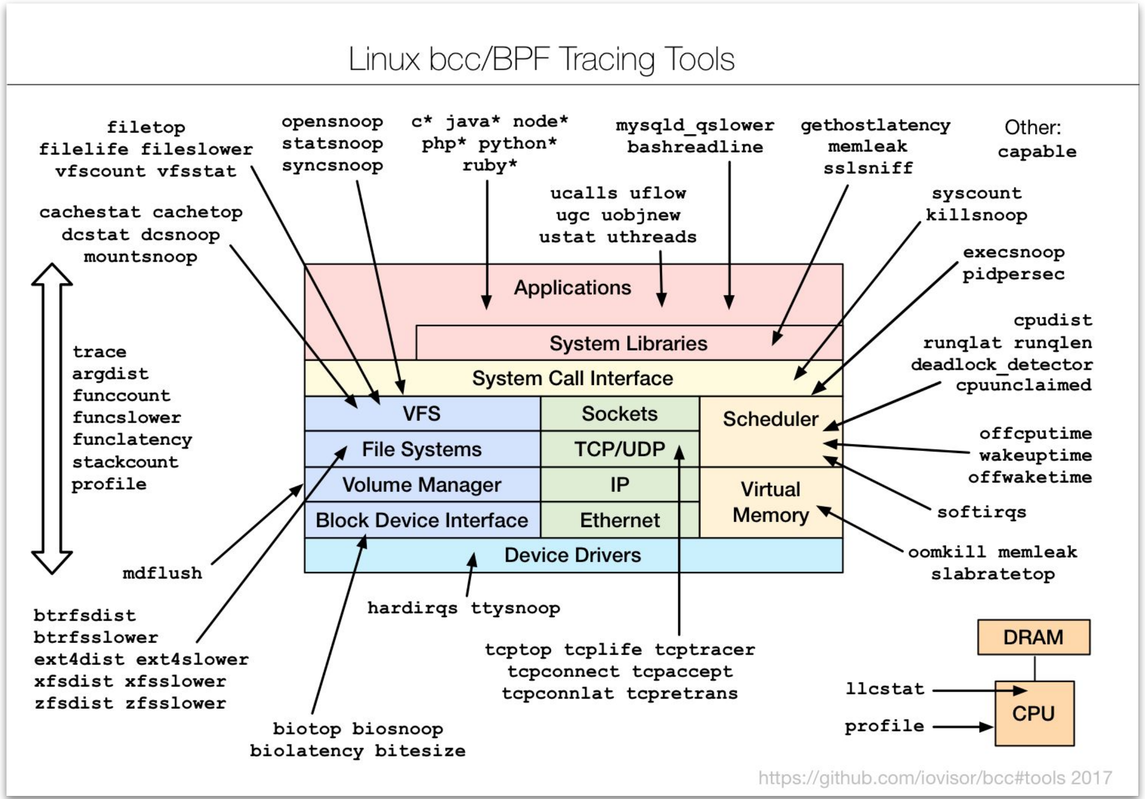
LAB: RHEL performance observability using bcc toolkit

bcc (BPF Compiler Collection) toolkit:

https://github.com/iovisor/bcc

LAB:

https://rhel-labs.instruqt.com/tracks/ebpf-tracing



end