

What should be the anthem of BPF?

May 2023

Dictionary: Anthem

1. a rousing or uplifting song identified with a particular group, body, or cause.
2. a choral composition based on a biblical passage, for singing by a choir in a church service.

What is BPF?

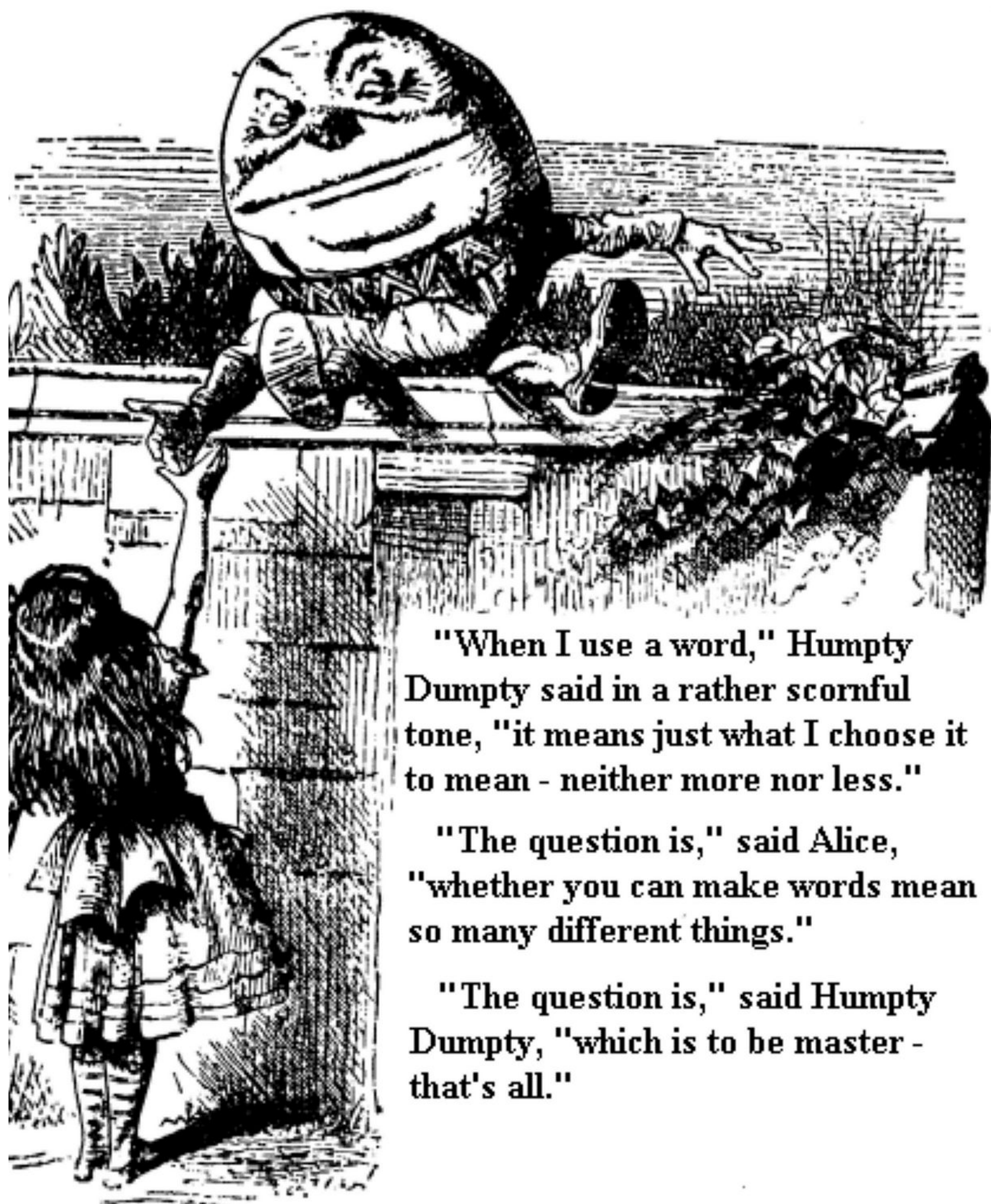
A. technology

B. philosophy

C. religion

D. cult

E. buzzword



"When I use a word," Humpty Dumpty said in a rather scornful tone, "it means just what I choose it to mean - neither more nor less."

"The question is," said Alice, "whether you can make words mean so many different things."

"The question is," said Humpty Dumpty, "which is to be master - that's all."

BPF is an universal assembly language

- strictly typed assembly language
- safe for kernel and for HW
- stable instruction set
- extensions are backwards compatible



BPF is a sequence of commands that can be understood



BPF use cases

- user space tells kernel what to do
- HW tells kernel what to do
- one server tells another server what to do

... the response could be: "yeah, I can do this" or "No, not right now".

... and since the intent is understood it's execution can be in a different form.
(CPU executes one instruction, BPF executes whole program)

BPF vs Sandboxing (wasm, ...)

- BPF program is understood before execution
- Sandbox restricts execution environment. It doesn't understand what's running in the sandbox.

BPF design goals

- verifiable ISA
- write programs in C and compile into BPF ISA with GCC/LLVM
- Just-In-Time convert to modern 64-bit CPU
- minimal performance overhead:
 - bpf vs native (C -> BPF ISA -> native ISA vs C -> native ISA)
 - transition from native to bpf (native code -> BPF code -> native code)
- BPF calling convention compatible with modern 64-bit ISAs

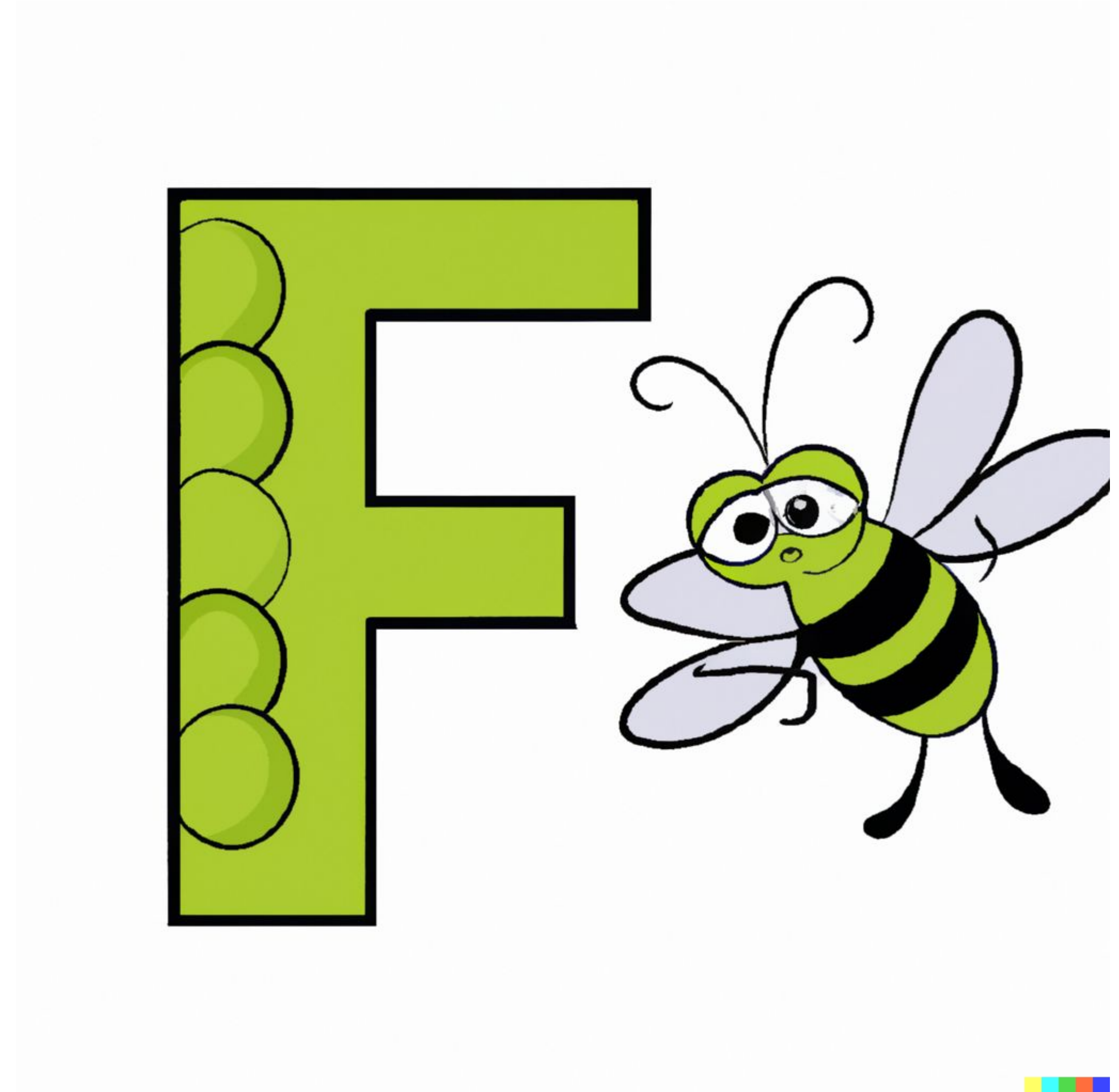
- Quiz:
 - What's faster the kernel in C or compiled through BPF ?
 - How sandboxed code calls native ? Hint: https://en.wikipedia.org/wiki/Foreign_function_interface

The anthem of BPF

In the kernel, there is a tool
What it can do is pretty cool
It can trace, it can filter,
It can load and unload
It's time for you to get on board.

BPF, BPF, what can I say?
It's a sandboxed program in the kernel today.
It can do a lot of things, you see,
Like analyze and debug the kernel for me.

We're excited to create and innovate
In the booming BPF kernel space.
BPF is fast and secure. It's easy to use.
Start using today. There is no excuse.



Ask AI: Where BPF is used?

BPF is used in a variety of places, including:

- **Network monitoring and troubleshooting:** BPF can be used to monitor network traffic and troubleshoot network problems. For example, BPF can be used to identify and block malicious traffic, or to track the performance of a network.
- **Security auditing:** BPF can be used to audit the security of a system. For example, BPF can be used to track which processes are accessing sensitive data, or to identify potential security vulnerabilities.
- **Performance profiling:** BPF can be used to profile the performance of a system. For example, BPF can be used to identify bottlenecks in a system, or to track the performance of a specific application.
- **Tracing:** BPF can be used to trace the execution of a program. For example, BPF can be used to track the calls that a program makes to the kernel, or to track the flow of data through a program.
- **Load balancing:** BPF can be used to load balance traffic across a network of servers. For example, BPF can be used to route traffic to the server with the least load, or to route traffic based on the content of the request.
- **Network programmability:** BPF can be used to extend the functionality of the Linux kernel. For example, BPF can be used to create new network filters, or to create new network protocols.

BPF is a powerful tool that can be used to improve the performance, security, and functionality of Linux systems. It is a valuable tool for system administrators, developers, and security researchers.

Ask AI: write eBPF program that parses IPv4

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

SEC("xdp")
int xdp_prog(struct xdp_md *ctx) {
    void *data = (void *) (long) ctx->data;
    void *data_end = (void *) (long) ctx->data_end;

    // Check packet length
    if (data + sizeof(struct ethhdr) + sizeof(struct iphdr) > data_end) {
        return XDP_DROP;
    }

    // Parse Ethernet header
    struct ethhdr *eth = data;
    if (eth->h_proto != htons(ETH_P_IP)) {
        return XDP_PASS;
    }

    // Parse IPv4 header
    struct iphdr *ip = data + sizeof(struct ethhdr);
    if (ip->version != 4) {
        return XDP_PASS;
    }

    // Extract source and destination IP addresses
    __u32 src_ip = ip->saddr;
    __u32 dst_ip = ip->daddr;

    // Do something with the IP addresses
    // ...

    return XDP_PASS;
}
```

Everything on this slide is AI generated

AI and search engines can answer:

- What is the anthem of BPF ?
- Where BPF is used ?
- What can I do with BPF ?
- Write eBPF program that does ...

BPF mission

- Accelerate the speed of innovation and creativity
- Provide the technology which enables programmers to deploy with unprecedented speed and scale

BPF values

- Innovate
- Enable others to innovate
- Communicate with empathy
- Challenge what's possible

What does it mean for...

For maintainers and developers:

- Improving the quality and speed of making decisions
- Code reviews are done with better efficiency
- Fewer arguments that side-track folks from their best work
- Clearer communication to ensure sound decisions

What does it mean for...

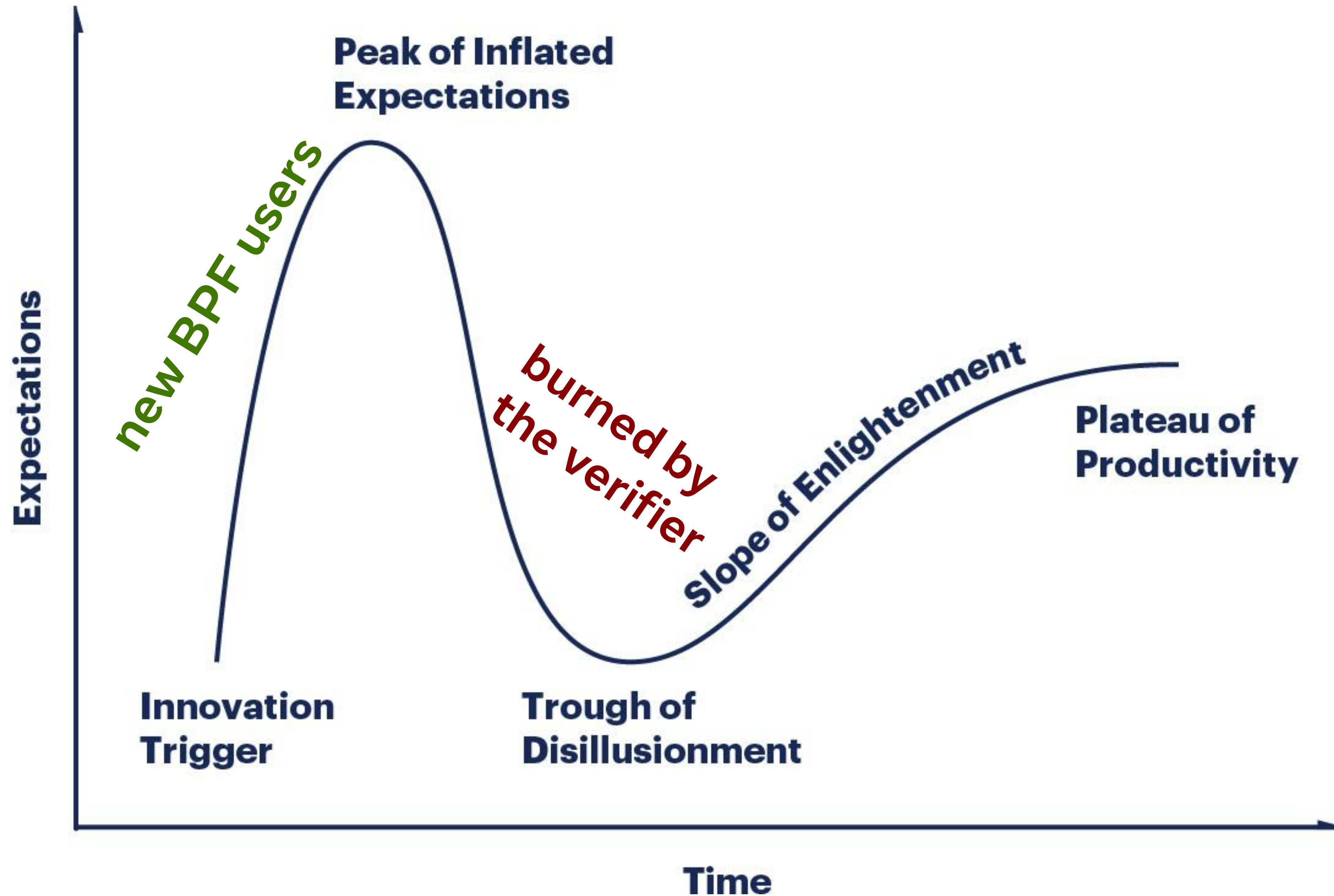
For sponsors and users (in addition to the earlier points):

- better engagement in the work that matters
- a sense of making a difference
- a way to give back
- be part of inside track of the latest developments
- a better view of the obstacles ahead (save time, money)
- good for the brand (successful recruiting)
- understand what competitors and strategic partners are concerned about (better long term view)

What is the biggest challenge for BPF ?

AI answers: complexity, security, verification.

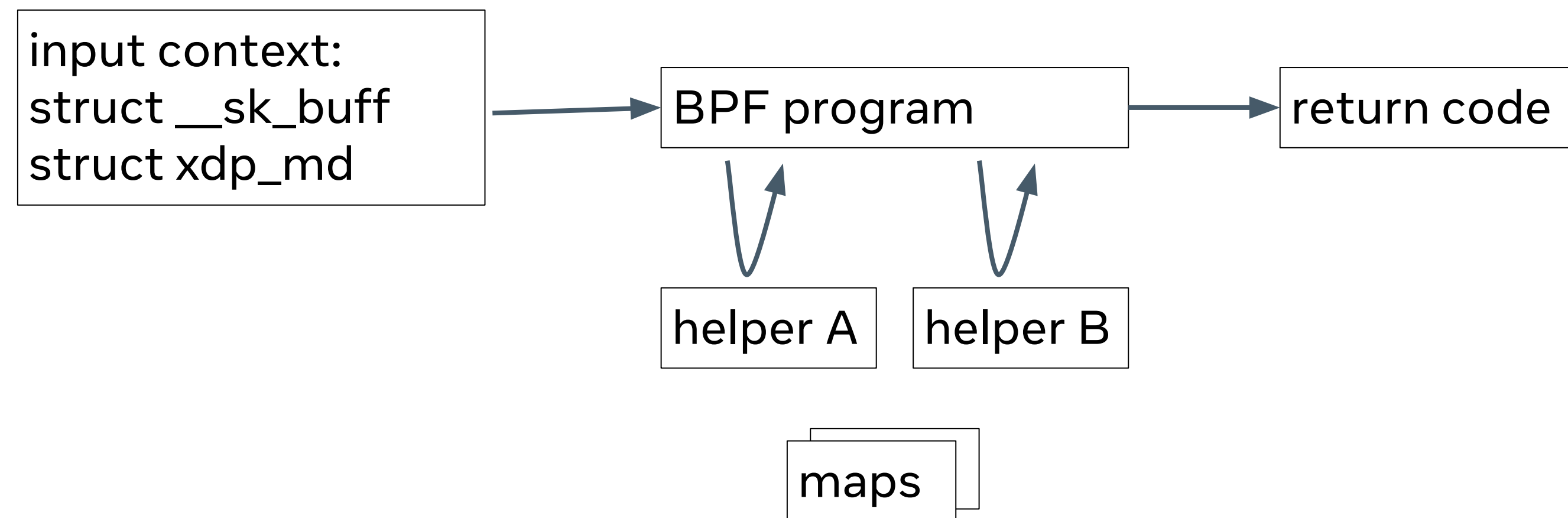
My answer: ease of use.



Major shift in BPF architecture happened in 2022

BPF and the kernel

- all BPF programs before 2022

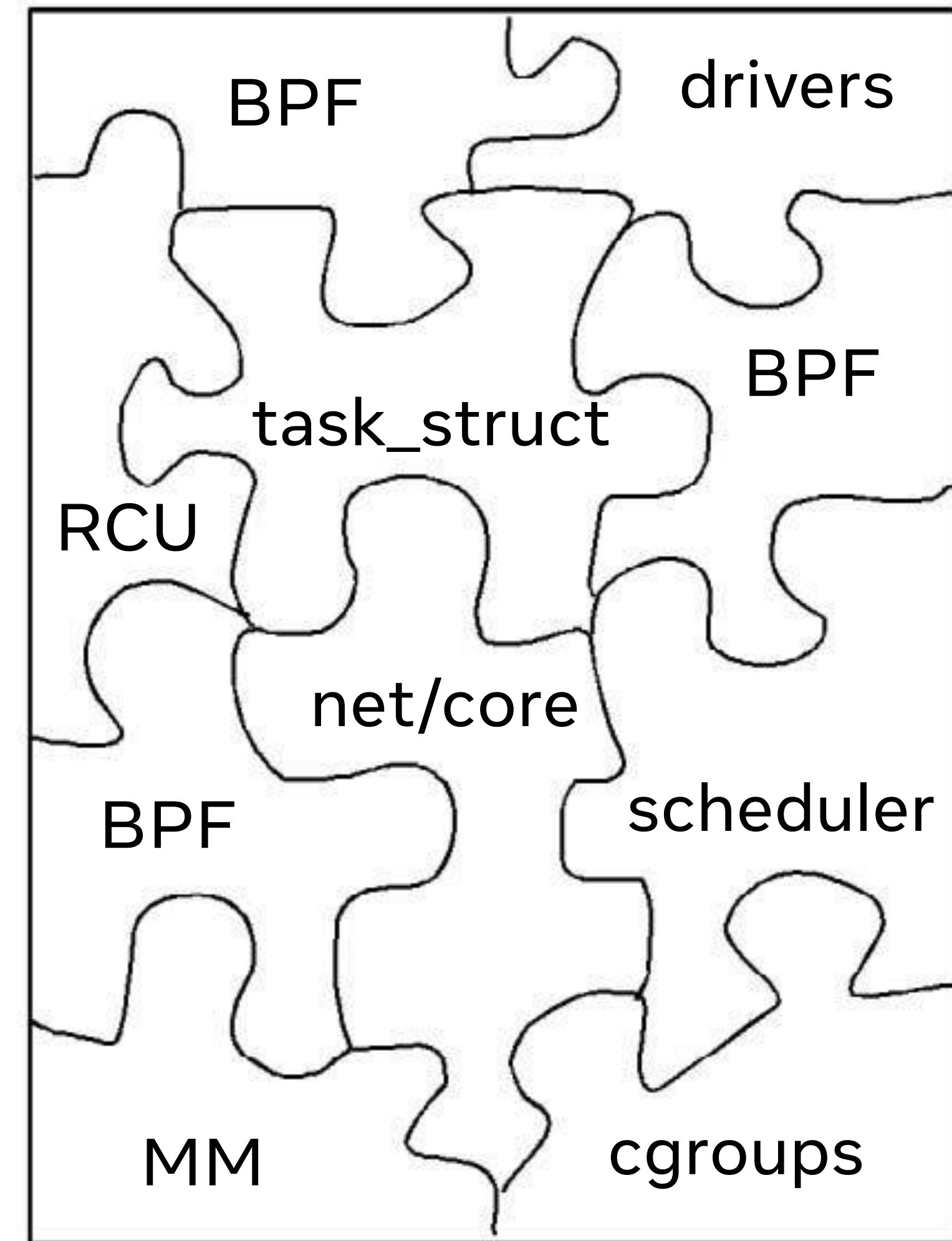


stable hook:

- kernel prepares data it wants BPF program to see
- kernel interprets return code

BPF in the kernel

- hid-bpf, sched-ext, netfilter, struct-ops
- "new tracing"
- Native calls: kernel ↔ BPF ↔ kernel
- BPF can refcnt ++, -- and stash kernel objects
- explicit rcu_read_lock/unlock
- NO stable API



BPF extended C is a safer C

```
int err_cast(struct task_struct *tsk)
{
    return((struct sk_buff *)tsk)->len;
}
```

OK in C.
NOT OK in BPF C.

```
int err_release_twice(struct __sk_buff *skb)
{
    struct bpf_sock_tuple tuple = {};

    struct bpf_sock *sk = bpf_sk_lookup_tcp(skb, &tuple, sizeof(tuple), 0, 0);
    bpf_sk_release(sk);
    bpf_sk_release(sk);
    return 0;
}
```

Call to action

- How would you see this vision apply to you?
- Can we do this together?
- Please help us spread the word
- Our door is open for feedback to make this better
- We want to hear your “wild ideas” for this technology
- Can we count on you to support our vision?
- Send patches