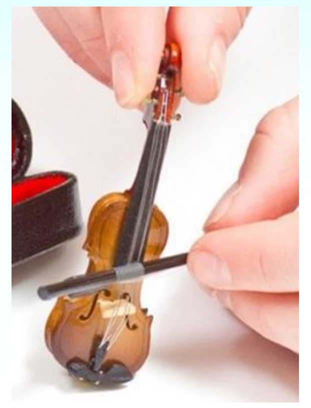
Extending libbpf for Kubernetes

Vinay Kulkarni

@iSkibum

Agenda

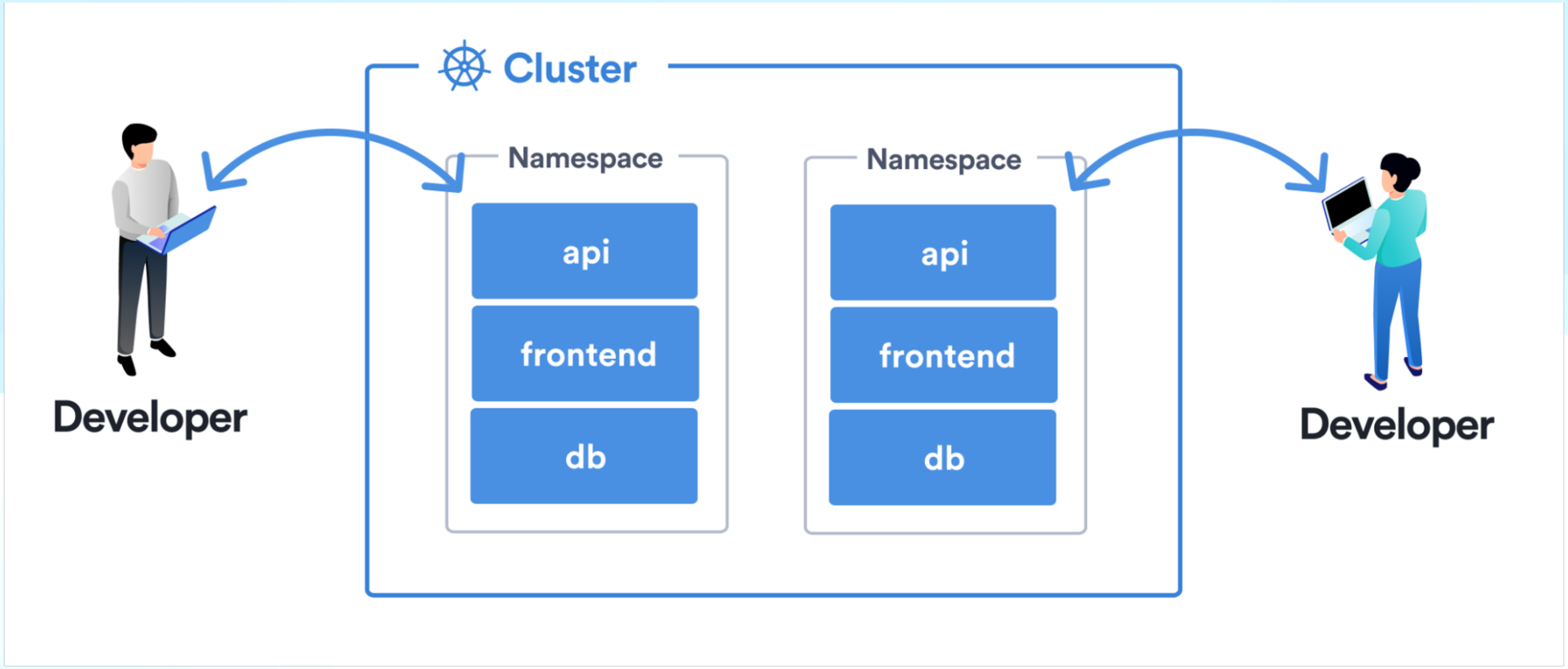
- eBPF to Proactively Do Stuff
  - Use Cases
- Problem at Hand …
- … How eBPF Came to the Rescue
- Pain Points
- Potential Solution
- Summary & Discussion



eBPF + Proactive: Use Cases

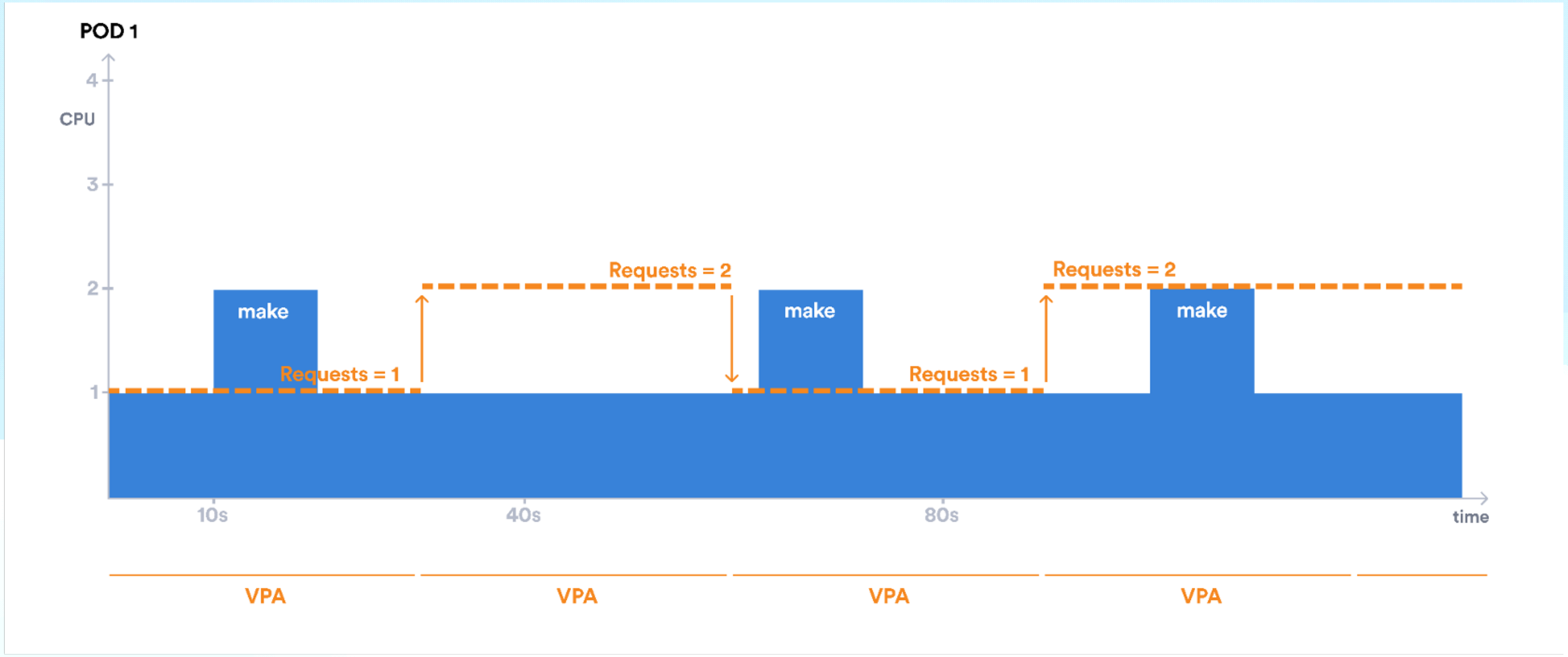
- Cloud-based Development Environment
    - Developers / dev teams write code with local IDE
    - Build code or run a battery of tests is Kubernetes pods
    - <u>Need</u>: Resize up build/test pod quickly
- Containerized Java application with high startup CPU requirements
    - Running time CPU usage is 1/10th the startup time CPU needs
        - Allocating too little CPU -> long startup time
        - Allocating startup requirements -> underutilized cluster
        - <u>Need</u>: Resize down pod quickly after startup
- eBPF network stats program attachment to pod veth

Cloud-based Development Environment



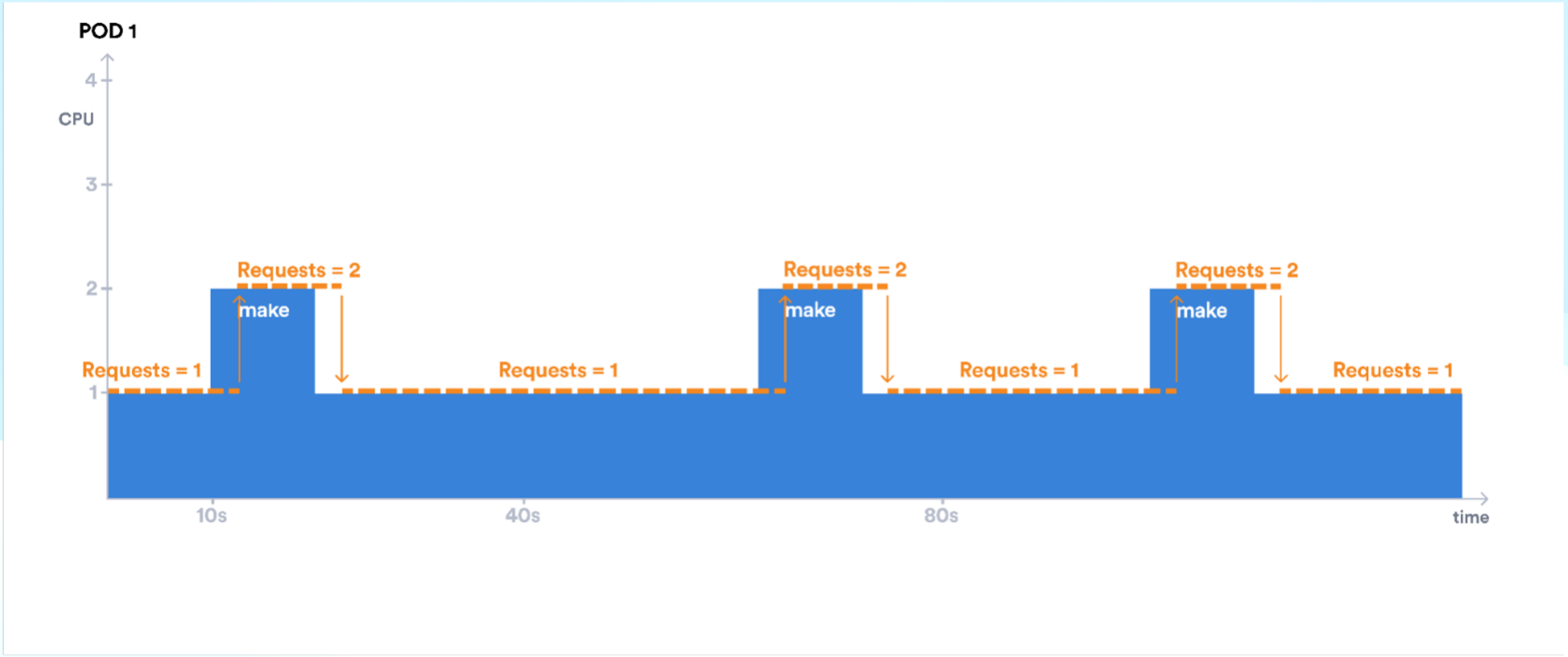
- Write code locally -> rsync -> build or run tests in K8s pod
- https://kccncna2022.sched.com/event/182HU

Problem is …



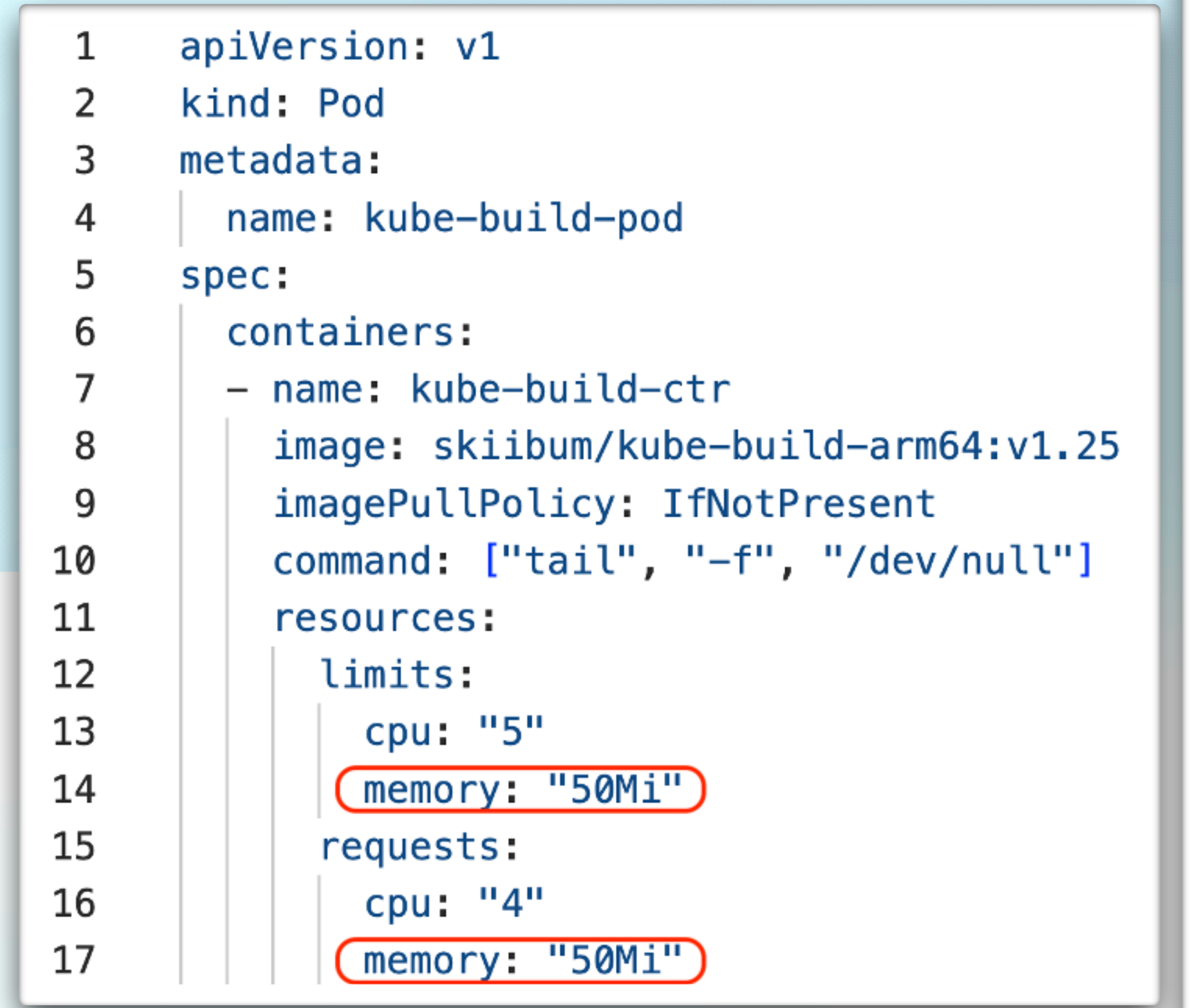
- Kubernetes v1.27 enables in-place restart-free resize of pod resources (CPU , memory)
- Vertical Pod Autoscaler can resize pod resources on seeing increased usage
  - Reactive - may not be good enough!

Ideally …

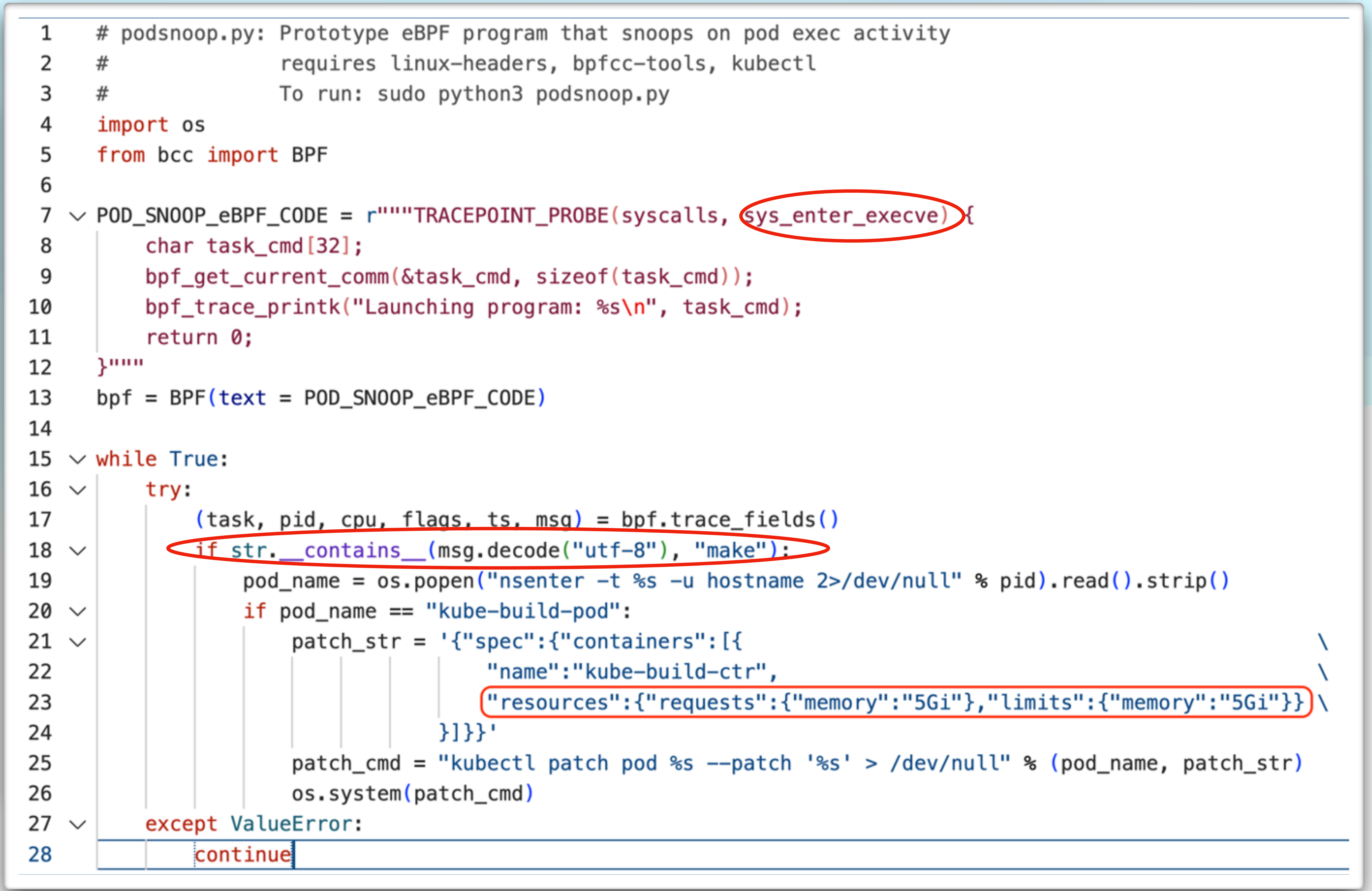


- Pod resources are resized before it becomes a problem
    - Proactive

eBPF makes it possible!



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: kube-build-pod
spec:
  containers:
  - name: kube-build-ctr
    image: skiibum/kube-build-arm64:v1.25
    imagePullPolicy: IfNotPresent
    command: ["tail", "-f", "/dev/null"]
    resources:
      limits:
        cpu: "5"
        memory: "50Mi"
      requests:
        cpu: "4"
        memory: "50Mi"
```

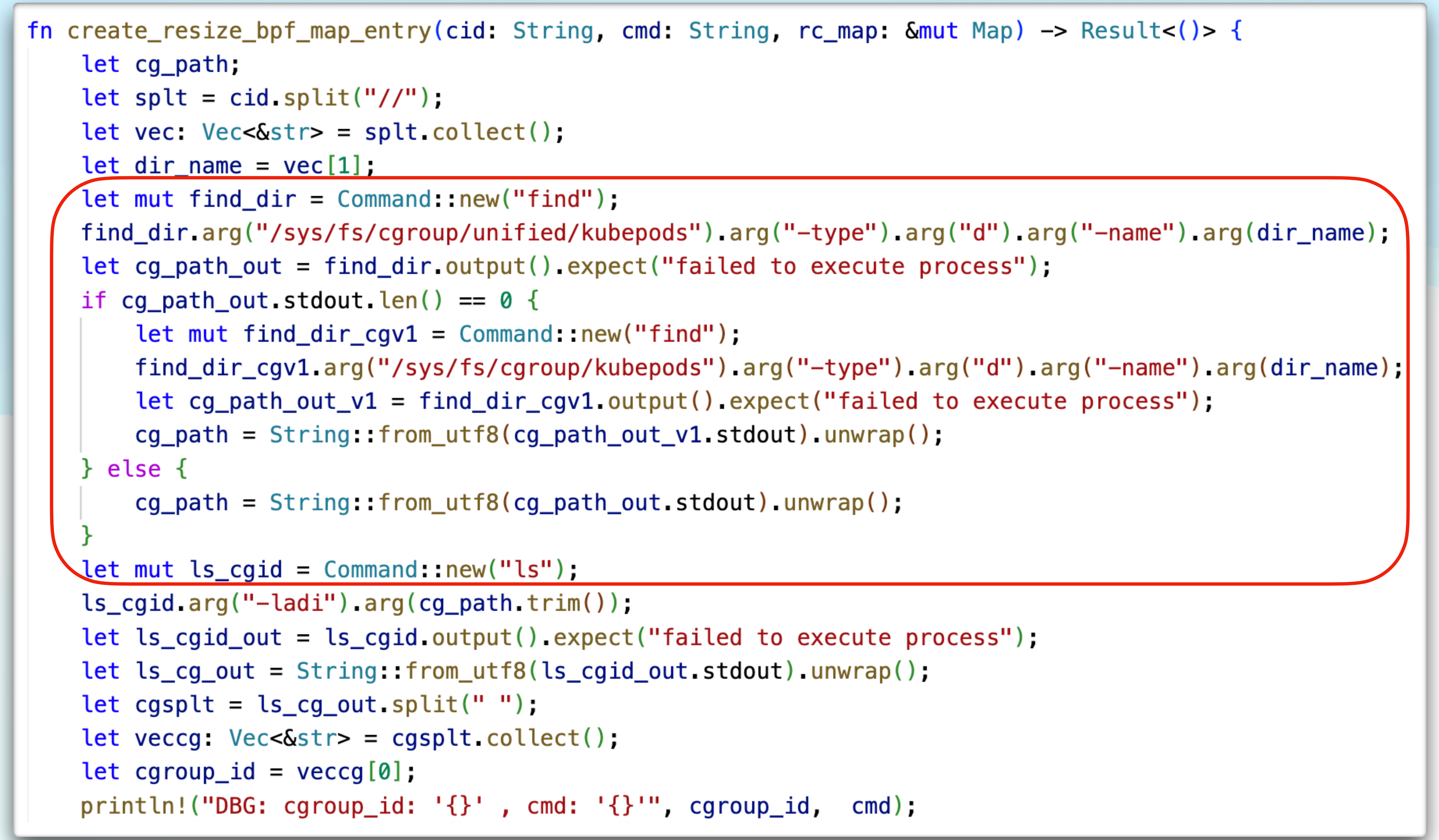


```python
# podsnoop.py: Prototype eBPF program that snoops on pod exec activity
#              requires linux-headers, bpfcc-tools, kubectl
#              To run: sudo python3 podsnoop.py
import os
from bcc import BPF

POD_SNOOP_eBPF_CODE = r"""TRACEPOINT_PROBE(syscalls, sys_enter_execve) {
    char task_cmd[32];
    bpf_get_current_comm(&task_cmd, sizeof(task_cmd));
    bpf_trace_printk("Launching program: %s\n", task_cmd);
    return 0;
}"""
bpf = BPF(text = POD_SNOOP_eBPF_CODE)

while True:
    try:
        (task, pid, cpu, flags, ts, msg) = bpf.trace_fields()
        if str.__contains__(msg.decode("utf-8"), "make"):
            pod_name = os.popen("nsenter -t %s -u hostname 2>/dev/null" % pid).read().strip()
            if pod_name == "kube-build-pod":
                patch_str = '{"spec":{"containers":[{                                             \
                        "name":"kube-build-ctr",                                              \
                        "resources":{"requests":{"memory":"5Gi"},"limits":{"memory":"5Gi"}}\
                      }]}}'
                patch_cmd = "kubectl patch pod %s --patch '%s' > /dev/null" % (pod_name, patch_str)
                os.system(patch_cmd)
    except ValueError:
        continue
```

- action = (command == ‘make’) ? resize pod : have a beer ;)

Some Rough Edges

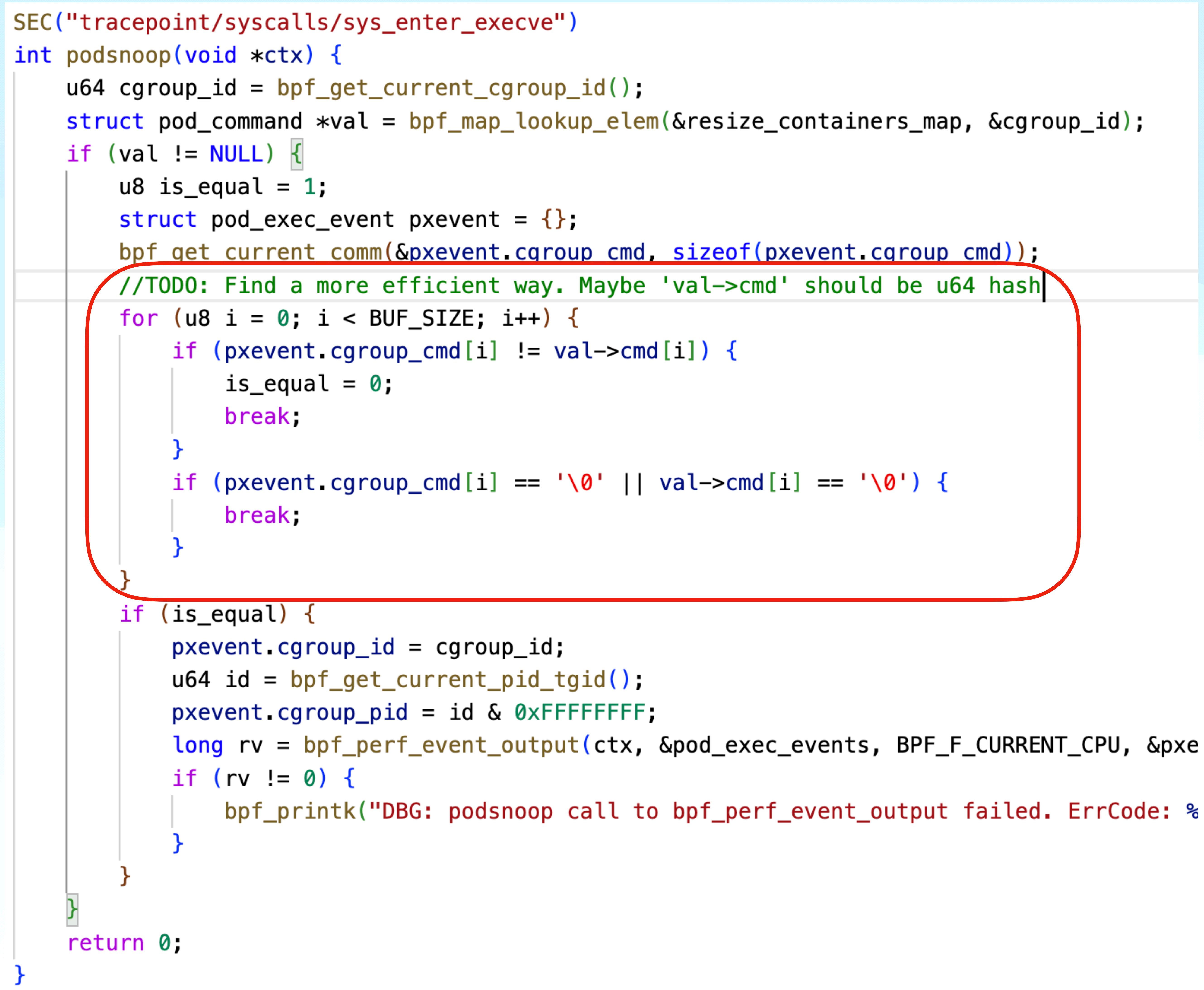
- Ugly demo code: https://github.com/vinaykul/ebpf-playground/tree/main/ebpf-pod-resize-libbpf-rs-CO-RE



```rust
fn create_resize_bpf_map_entry(cid: String, cmd: String, rc_map: &mut Map) -> Result<()> {
    let cg_path;
    let splt = cid.split("//");
    let vec: Vec<&str> = splt.collect();
    let dir_name = vec[1];
    let mut find_dir = Command::new("find");
    find_dir.arg("/sys/fs/cgroup/unified/kubepods").arg("-type").arg("d").arg("-name").arg(dir_name);
    let cg_path_out = find_dir.output().expect("failed to execute process");
    if cg_path_out.stdout.len() == 0 {
        let mut find_dir_cgv1 = Command::new("find");
        find_dir_cgv1.arg("/sys/fs/cgroup/kubepods").arg("-type").arg("d").arg("-name").arg(dir_name);
        let cg_path_out_v1 = find_dir_cgv1.output().expect("failed to execute process");
        cg_path = String::from_utf8(cg_path_out_v1.stdout).unwrap();
    } else {
        cg_path = String::from_utf8(cg_path_out.stdout).unwrap();
    }
    let mut ls_cgid = Command::new("ls");
    ls_cgid.arg("-ladi").arg(cg_path.trim());
    let ls_cgid_out = ls_cgid.output().expect("failed to execute process");
    let ls_cg_out = String::from_utf8(ls_cgid_out.stdout).unwrap();
    let cgsplt = ls_cg_out.split(" ");
    let veccg: Vec<&str> = cgsplt.collect();
    let cgroup_id = veccg[0];
    println!("DBG: cgroup_id: '{}' , cmd: '{}'", cgroup_id,  cmd);
```

- Hacky way to find containerID <> cgroup_id mapping, isn't it?

Some Rough Edges



```
SEC("tracepoint/syscalls/sys_enter_execve")
int podsnoop(void *ctx) {
    u64 cgroup_id = bpf_get_current_cgroup_id();
    struct pod_command *val = bpf_map_lookup_elem(&resize_containers_map, &cgroup_id);
    if (val != NULL) {
        u8 is_equal = 1;
        struct pod_exec_event pxevent = {};
        bpf_get_current_comm(&pxevent.cgroup_cmd, sizeof(pxevent.cgroup_cmd));
        //TODO: Find a more efficient way. Maybe 'val->cmd' should be u64 hash
        for (u8 i = 0; i < BUF_SIZE; i++) {
            if (pxevent.cgroup_cmd[i] != val->cmd[i]) {
                is_equal = 0;
                break;
            }
            if (pxevent.cgroup_cmd[i] == '\0' || val->cmd[i] == '\0') {
                break;
            }
        }
        if (is_equal) {
            pxevent.cgroup_id = cgroup_id;
            u64 id = bpf_get_current_pid_tgid();
            pxevent.cgroup_pid = id & 0xFFFFFFFF;
            long rv = bpf_perf_event_output(ctx, &pod_exec_events, BPF_F_CURRENT_CPU, &pxe
            if (rv != 0) {
                bpf_printk("DBG: podsnoop call to bpf_perf_event_output failed. ErrCode: %
            }
        }
    }
    return 0;
}
```

- ~~Maybe add bpf_strncmp(…)?~~ NVM: It has already been added in libbpf

Potential libbpf Extensions

- Add: `u64 bpf_get_container_cgroup_id(const char *container_id)`
    - How: Scan /sys/fs/cgroup for container_id (For cgroups v1, look under /sys/fs/cgroup/cpu)
        - If found, return its i-node number
        - If not, return 0
- Add: `int bpf_get_cgroup_id_container(u64 cgroup_id, const char *container_id)`
    - How: Native version of `find /sys/fs/cgroup -inum <cgroup_id>
- ~~Add: int bpf_strncmp(const char *s1, const char *s2, register size_t n)~~

Summary + Discussion + Q&A

- At least two use cases that could leverage simplified cgroup_id <> container_id helper.
- Is this the right way to do it?
  - If not, any alternative suggestions?