What happens when you `kubectl apply`?

A live trace of one Pod, from keystroke to running process.

$ finger stone

```
Login: stone                            Name: Fredrik Steen <stone@varnish-software.com>
Directory: /home/stone                  Shell: /bin/zsh
Logged in.
A lot of mail.
Plan:

Director of Software Engineering @ Varnish Software https://www.varnish-software.com/

LinkedIn: fredriksteen
GitHUb: stone
Home: https://tty.se/

Linux and Open Source DevOps/Platform Engineer/developer since the mid 90s,
PsyTrance-DJ, Beekeeper, Maker-Tinkerer, Yeast-wrangler and Beer enthusiast.
```

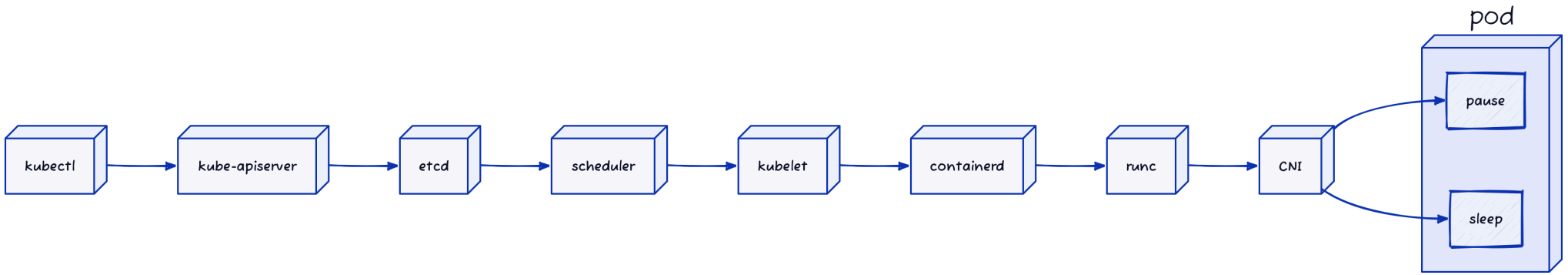
The whole demo, in one file

```yaml
# pod.yaml
apiVersion: v1
kind: Pod
metadata: { name: demo, labels: { app: demo } }
spec:
  containers:
  - name: demo
    image: busybox:latest
    command: ["sleep", "3600"]
```

One pod. One container. Sleeps for an hour.

What happens when you `kubectl apply -f pod.yaml`?

The path



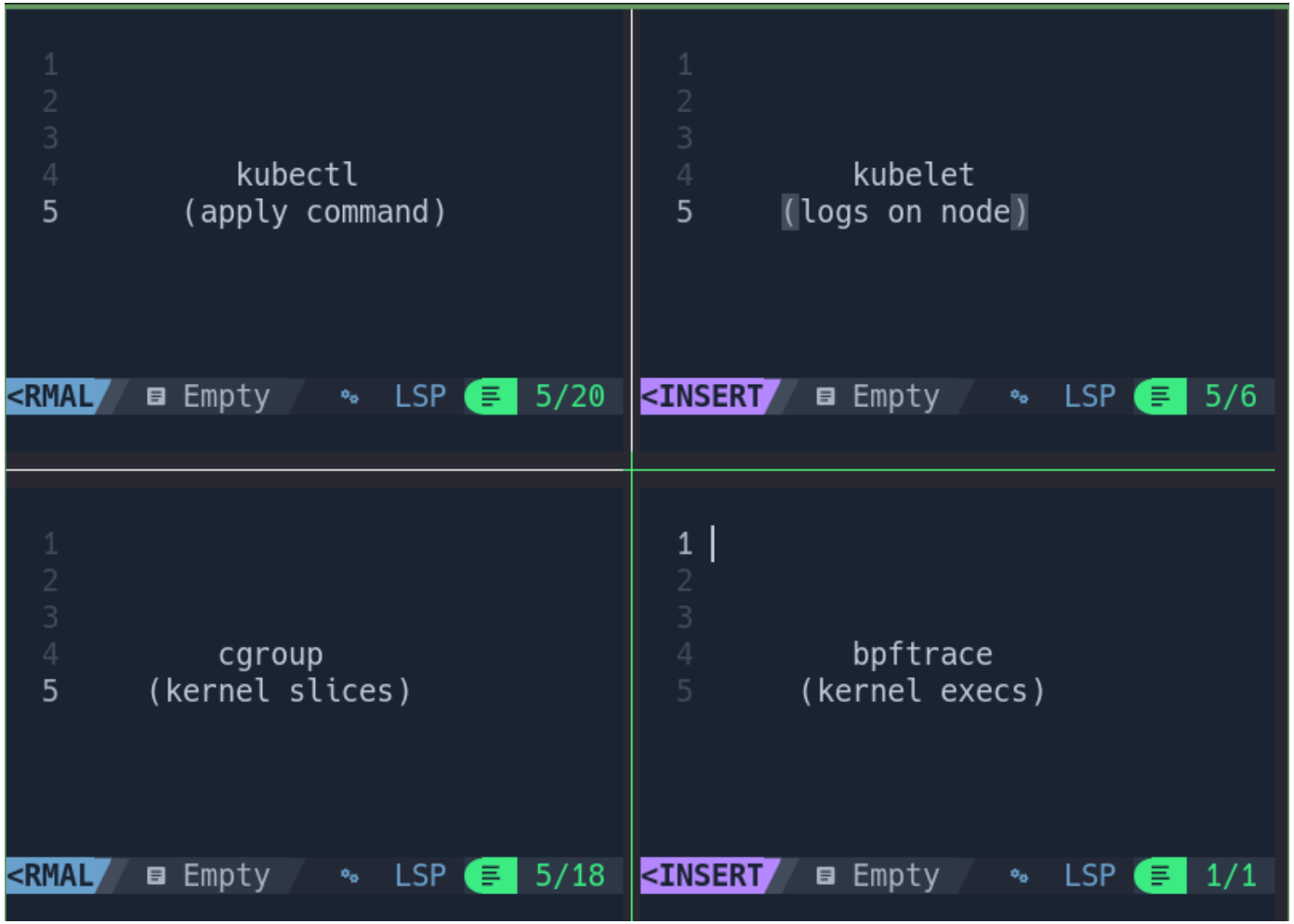
- Six processes you didn't write.
- Three Kubernetes components. One container runtime.
- One cgroup. One network namespace.

All so a `sleep` can happen.

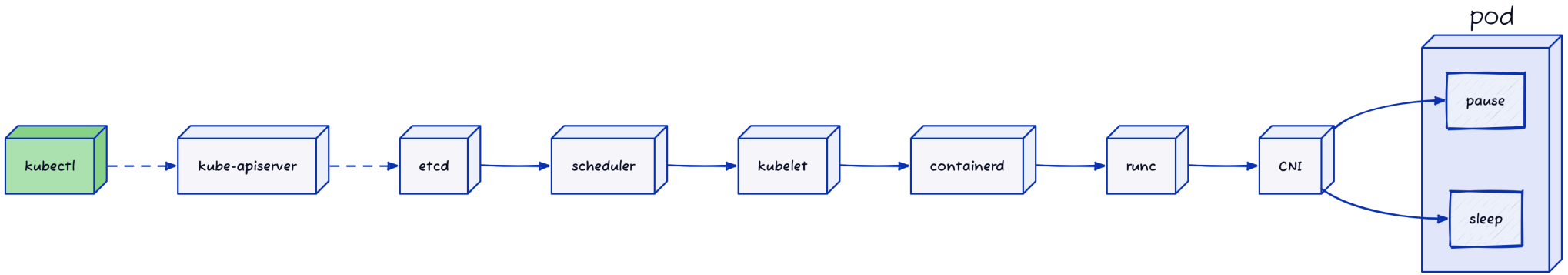
Four panes, four viewpoints

Top row = **Kubernetes view**.

Bottom row = **Linux view**.

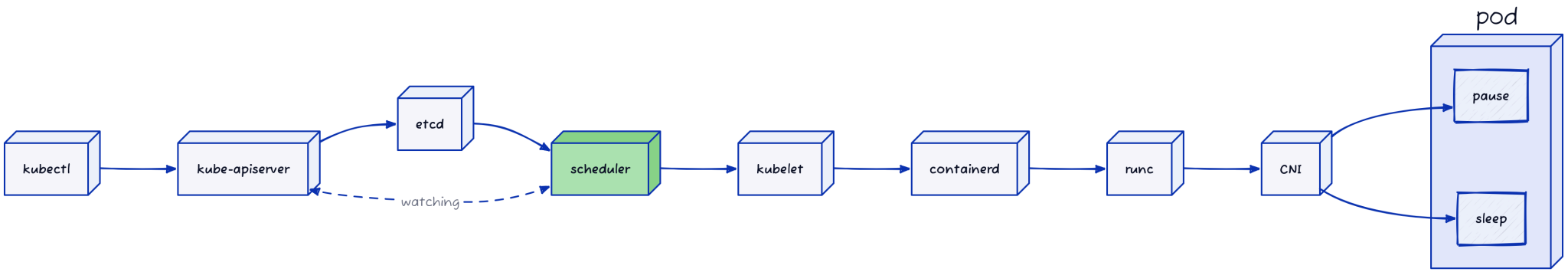


`kubectl apply`



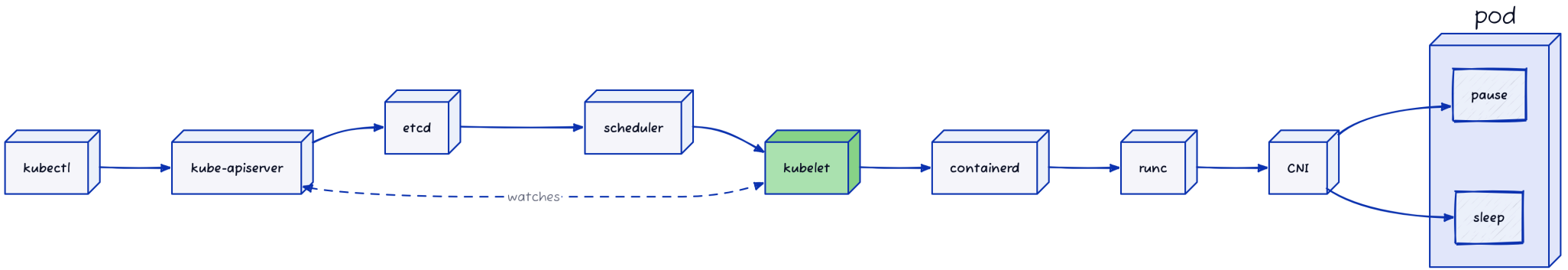
- TLS connection to the apiserver
- AuthN -> admission -> validation
- Object written to etcd
- apiserver fans the change out to its watchers

Scheduler binds the pod



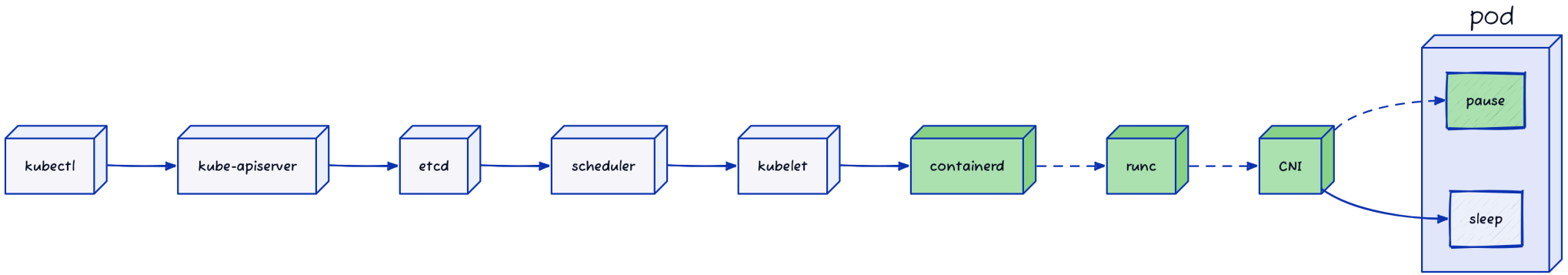
- Scheduler is its own process, watching the apiserver
- Filters nodes, scores nodes (priorities)
- Only one node in demo, but the loop still runs

Kubelet wakes up



- Kubelet watches the apiserver for pods bound to it self
- Starts a pod sync — its reconciliation loop
- From here, everything happens on the node
- Talks to containerd over the CRI (Container Runtime Interface)

Sandbox + cgroup (the meat)



Every pod runs two containers minimum:

1. **pause** — owns the network namespace, keeps it alive
2. Our actual container(s)

a new `kubelet-kubepods-besteffort-pod<UID>.slice/` appears, then `cri-containerd-<id>.scope` inside it. `[exec] containerd-shim`, then `[exec] pause`

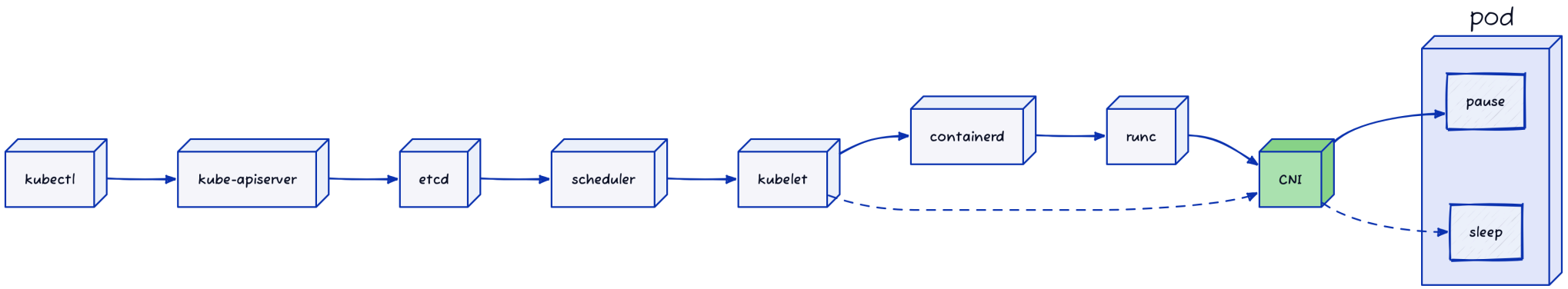
What each layer is doing

Tool	Job
containerd	the CRI runtime kubelet talks to
containerd-shim	per-container supervisor (outlives runc)
runc	creates the namespaces, sets up cgroups, pivots root
pause	tiny C program that holds onto the netns

`cgroup` = resource accounting/limits.

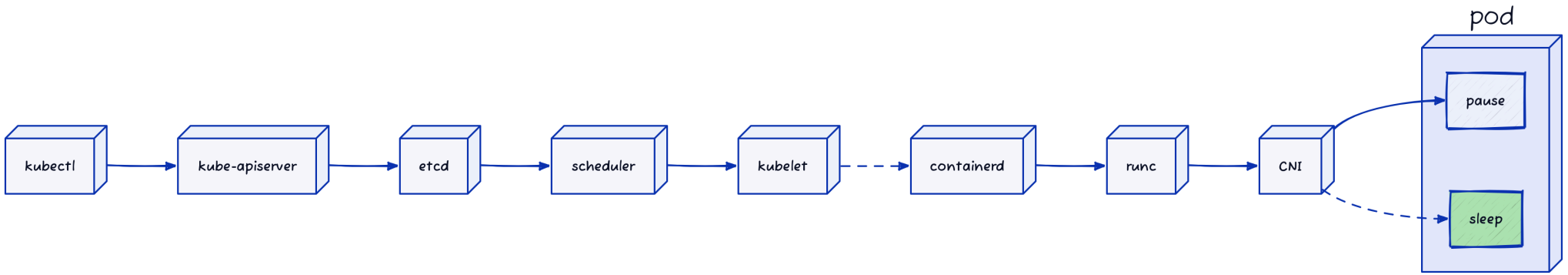
`namespace` = isolation (PID, net, mount, …).

CNI assigns the IP



- Sandbox has a netns but no interface yet
- Kubelet calls a CNI plugin
- Plugin creates a veth pair host <-> netns
- Assigns IP, adds route

App container starts



- Image already in containerd's content store (we pre-pulled)
- kubelet asks containerd to create + start the app container
- `runc` runs again, in the existing sandbox
- Our command execs

Questions?

Take away

Before the actual application container (`busybox`) can run, the foundational isolation environment the sandbox needs to run

pause - A tiny, practically empty C program (`sleep` indefinitely). It is the first container started in the Pod. Its sole purpose is to hold the network namespace and IPC namespace open so other containers can join them.