Zero-Touch OS Infrastructure for Container and Kubernetes Workloads

Open Source Camp Nürnberg

18. April 2024

Hello, I'm

Thilo

Thilo Fromm

Flatcar Maintainer, Eng. Mgr @ Microsoft

Github: t-lo
Mastodon: @thilo@fromm.social
Email: thilofromm@microsoft.com

Outline

Foundational Concepts

Fresh & Stable: Staying up to Date, safely

Composability

Community

Container Optimised Linux

Container Optimised Linux

Rethink the OS as a cloud-native replaceable commodity

- Provision like a container app or pod:

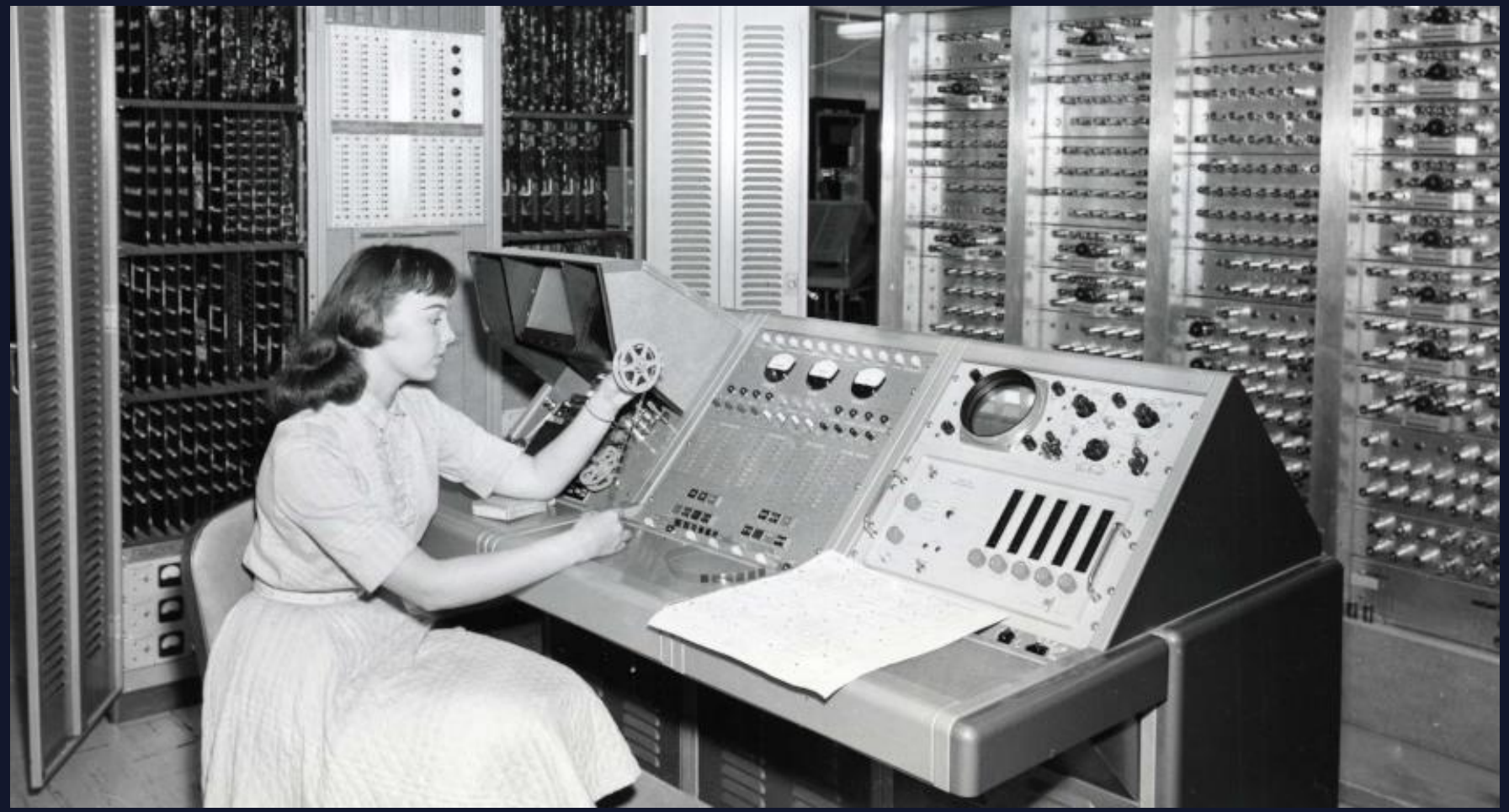
  Nodes are instances of immutable images

- Leverage container isolation from the OS side

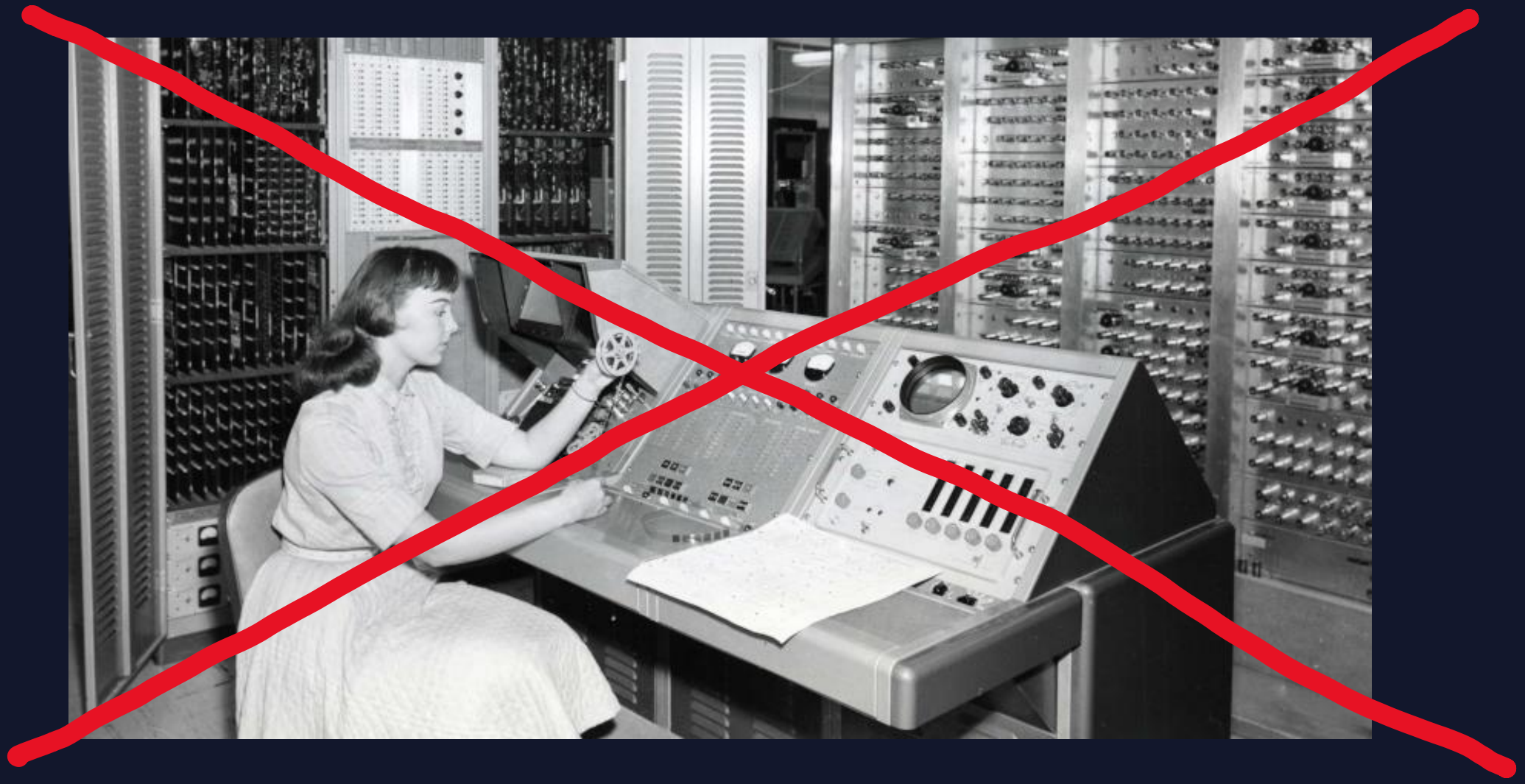
- Zero touch: *safe* auto-updates

==> A "Day #2" operating system

UX Philosophy

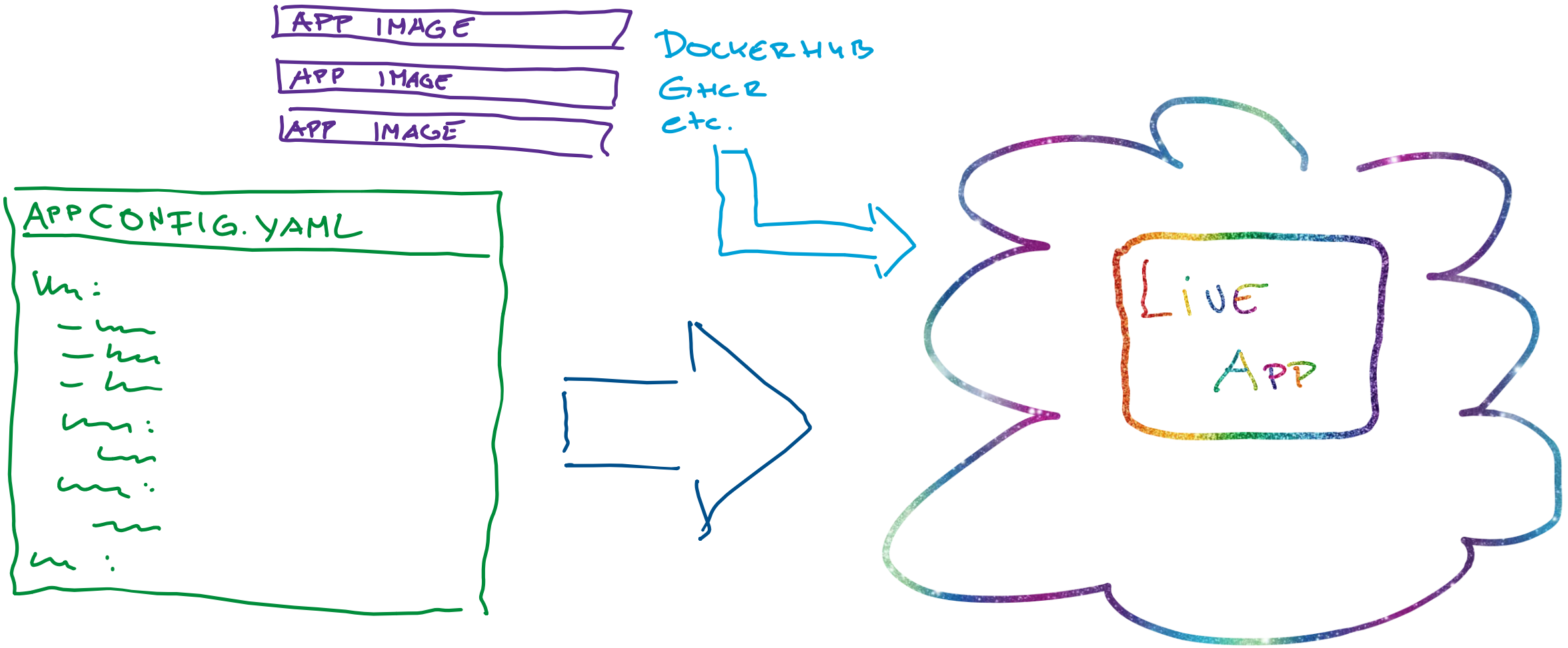


UX Philosophy

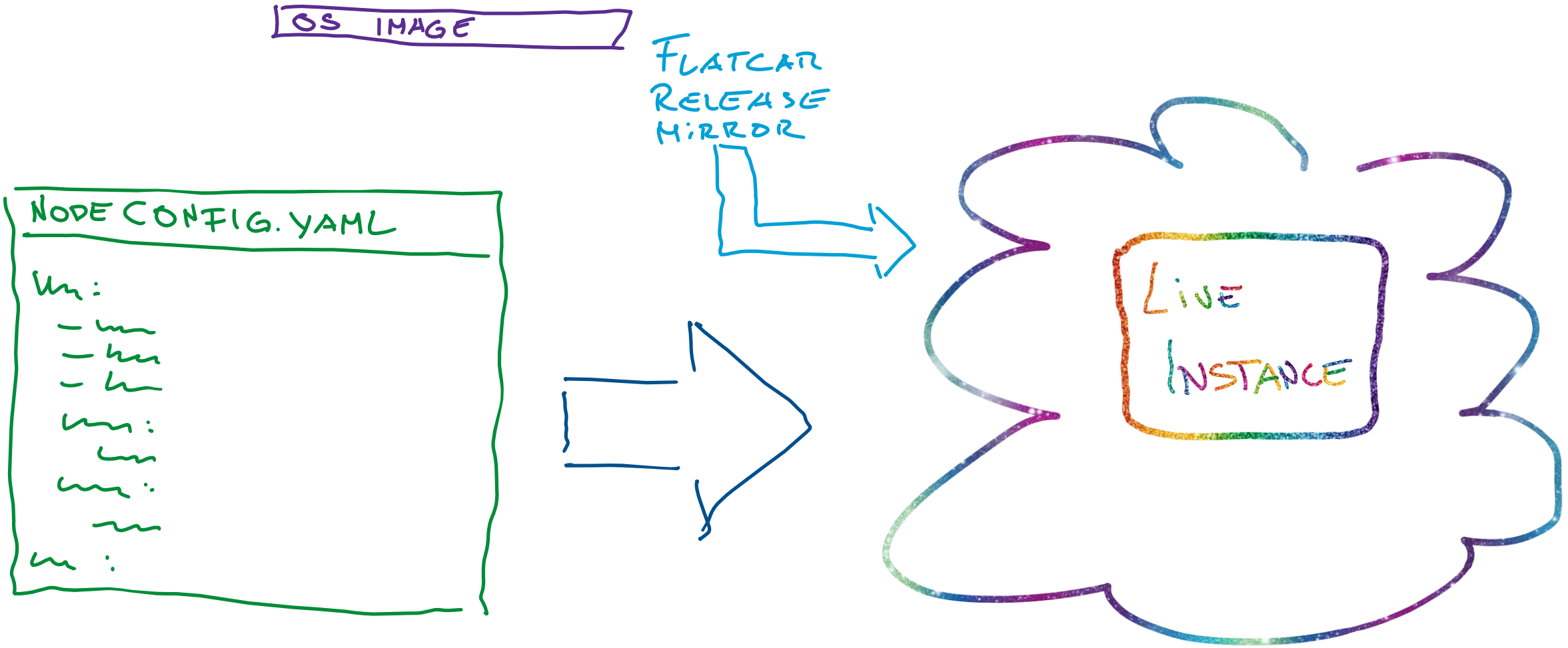


UX Philosophy

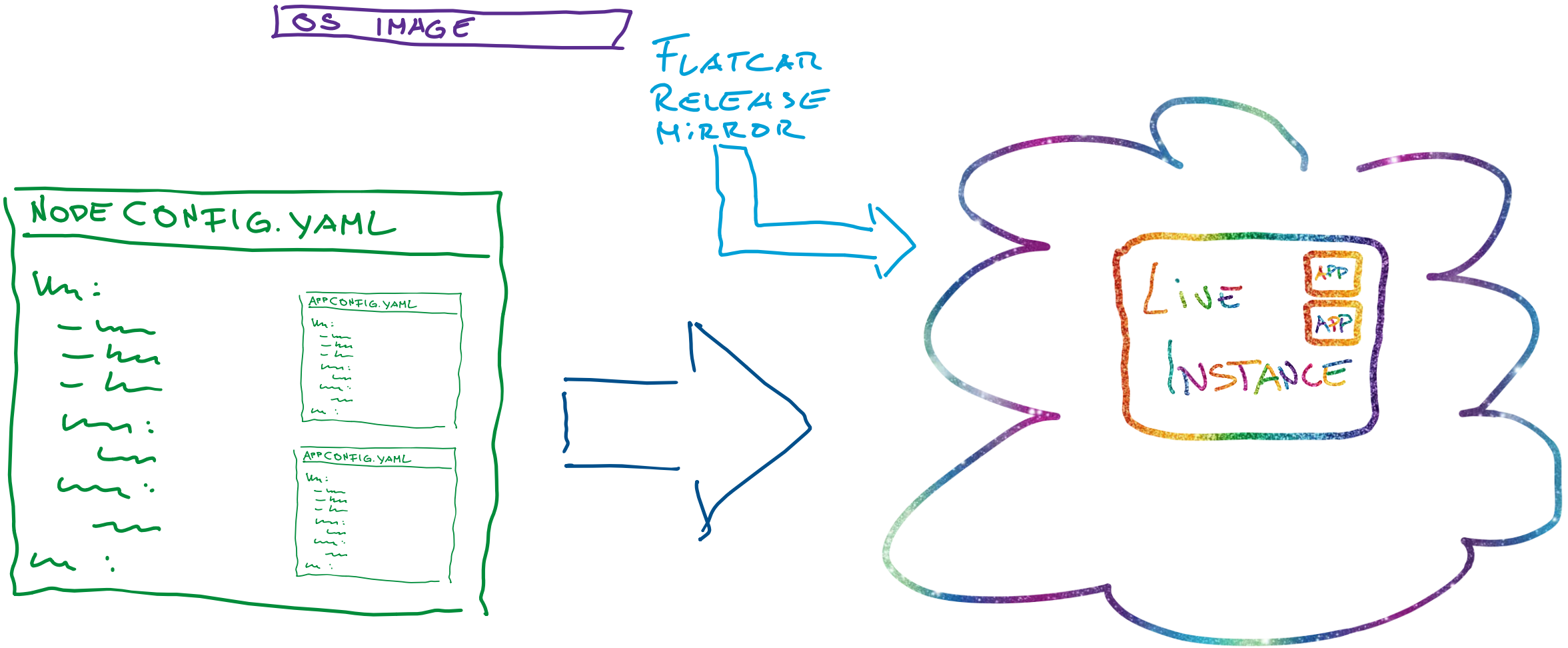
Container / Kubernetes App Provisioning



OS Provisions like a Container App



Bootstrap Initial Apps when Provisioning



Declarative configuration

```
passwd:
  users:
    - name: caddy
      no_create_home: true
      groups: [ docker ]

storage:
  files:
    - path: /srv/www/html/index.html
      mode: 0644
      user:
        name: caddy
      group:
        name: caddy
      contents:
        inline: |
          <html><body align="center">
          <h1>Hello KCD!</h1>
          <img src="kcd.png" alt="KCD logo" />
          </body></html>
    - path: /srv/www/html/kcd.png
      mode: 0644
      user:
        name: caddy
      group:
        name: caddy
      contents:
        local: kcd.png
```

```
systemd:
  units:
    - name: kcd-demo-webserver.service
      enabled: true
      contents: |
        [Unit]
        Description=KCD example static web server
        After=docker.service
        Requires=docker.service
        [Service]
        User=caddy
        TimeoutStartSec=0
        ExecStartPre=-/usr/bin/docker rm --force caddy
        ExecStart=docker run -i -p 80:80 \
                  -v /srv/www/html:/usr/share/caddy \
                  docker.io/caddy caddy file-server \
                  --root /usr/share/caddy --access-log
        ExecStop=/usr/bin/docker stop nginx1
        Restart=always
        RestartSec=5s
        [Install]
        WantedBy=multi-user.target
```

Operate the OS like a Container App or Pod

Sensible defaults, no boilerplate

- Storage networking, users, ssh, systemd units – only if you need these
- Inline / download custom directories and files

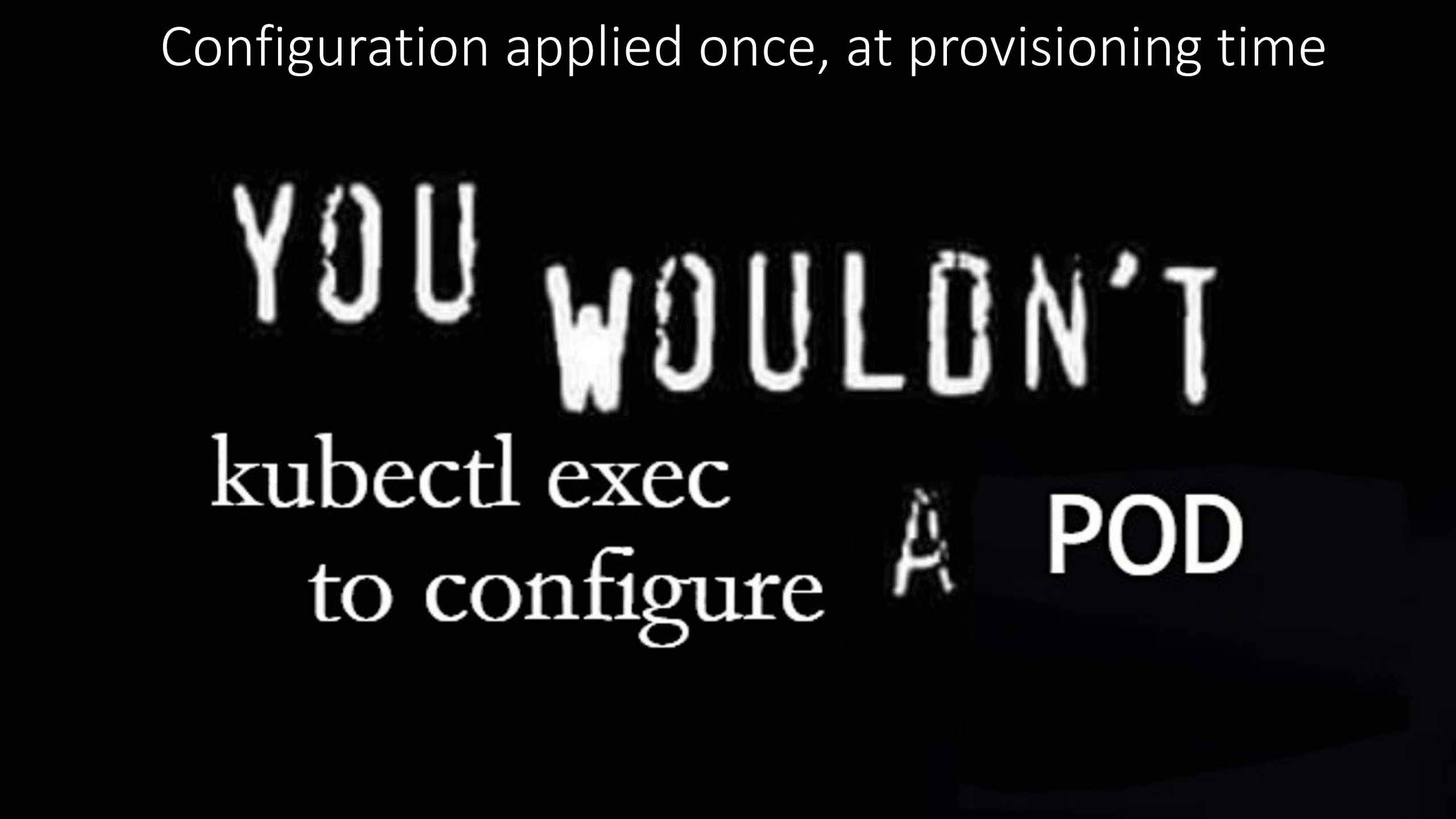
No config drift

- Configured at first boot / during provisioning
- New and existing (updated) node configs do not differ

Extensive Automation

- OS supports many cloud providers and private clouds, support is growing
- Terraform integration, Go bindings
- ClusterAPI integration

Configuration applied once, at provisioning time



Large-Scale deployments? ClusterAPI!

Supported out-of-the box by Core CAPI and image-builder

Multiple large vendors are supported

- AWS
- Azure
- VSphere
- OpenStack
- Tinkerbell (via Sysexts)

GCP support is work-in-progress.

Piloting sysext CAPI deployments (composed at provisioning, updatable)

Image-Based OS

Provisioning and updates are immutable images

- Always built from scratch, always fully tested. Self-contained, all bits included.
- No version drift: releases are frozen version sets
- No difference between new and existing (updated) nodes

All OS binaries on a separate, immutable partition

- Everything is in /usr, read-only and dm-verity protected

In-place updates via A/B partitions

- Retains node state - DB node operators rejoice!
- Updates are atomic, roll-backs are easy

Provisioning Demo

Leverage Container Isolation

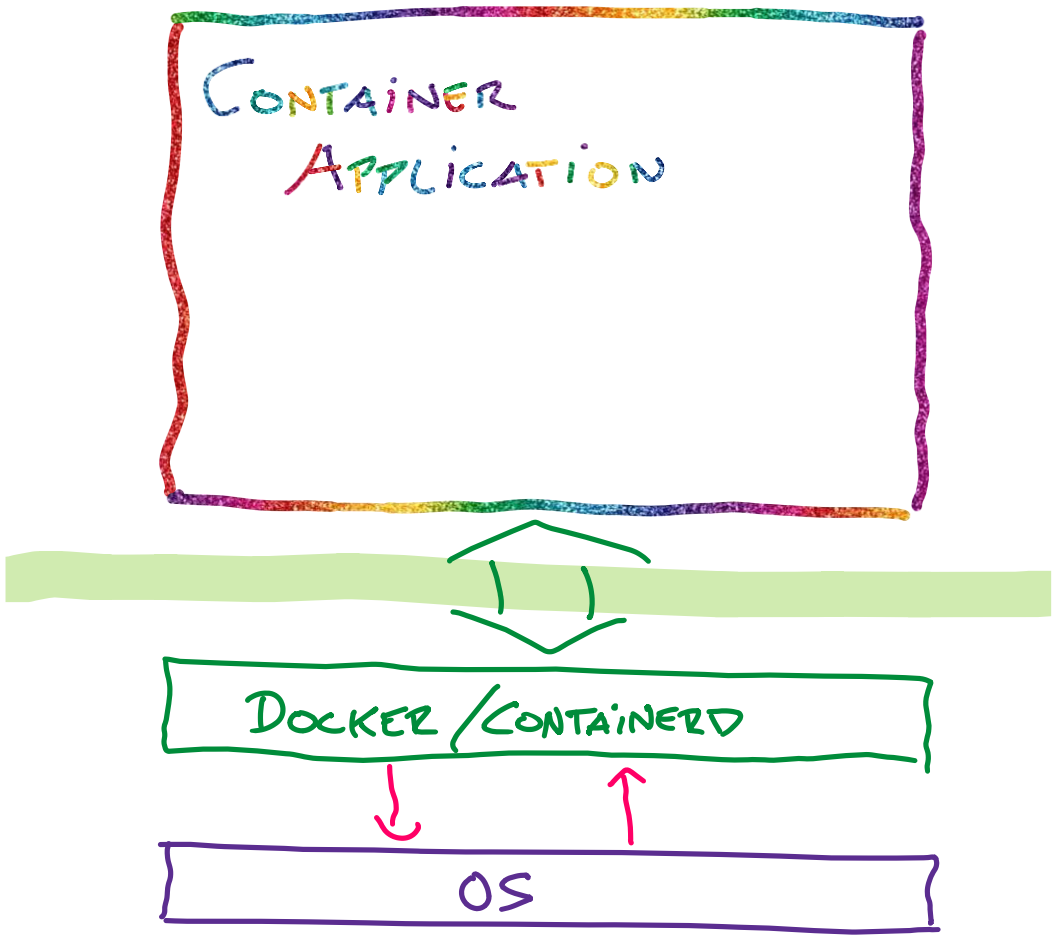
Container apps are self-contained and run isolated

- From each other, but also from the OS
- Few and well-defined dependencies on the OS

No inter-dependencies OS <-> App

- No shared libraries / binaries
- No shared configuration

➔ Portable Applications



Leverage Container Isolation from the OS side

Well-defined interfaces OS <-> App

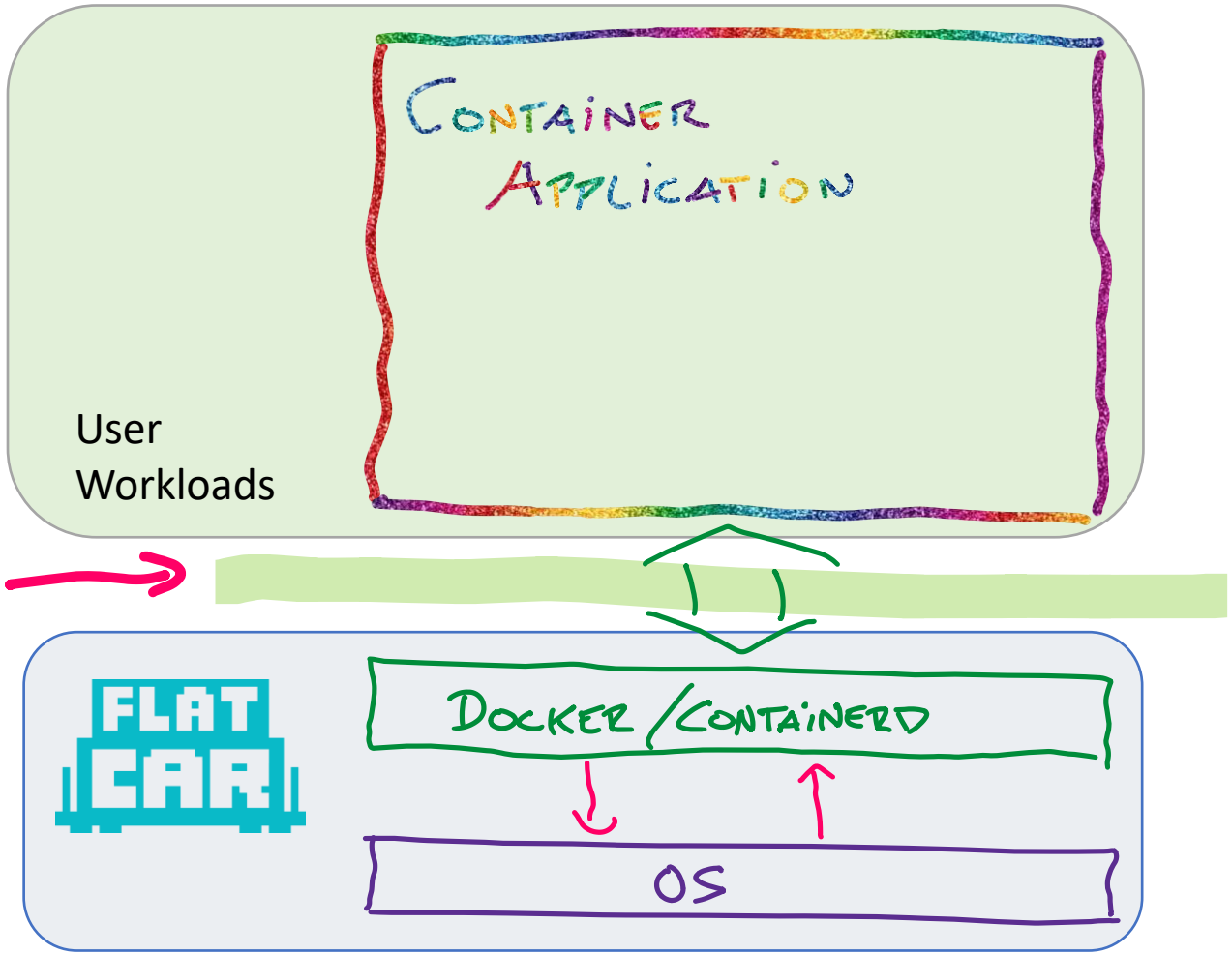
- Very few components, easy to test thoroughly
- No other inter-dependencies
- Container apps isolate from the OS

Runtime + Kernel is a Contract

- App relies on contract and nothing else
- OS guarantees and fulfils contract

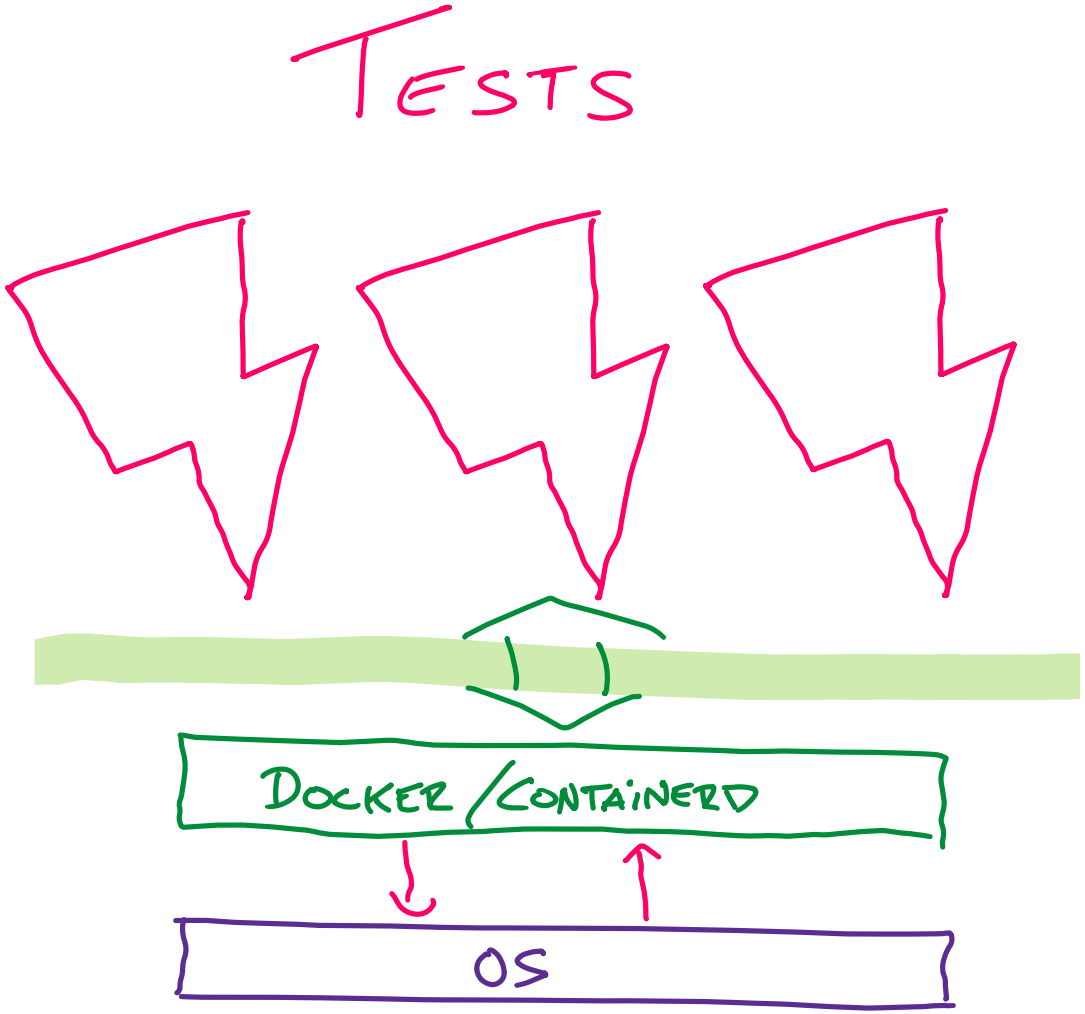
➔ Interchangeable OS

- Main Focus on upholding runtime contract



Interchangeable OS

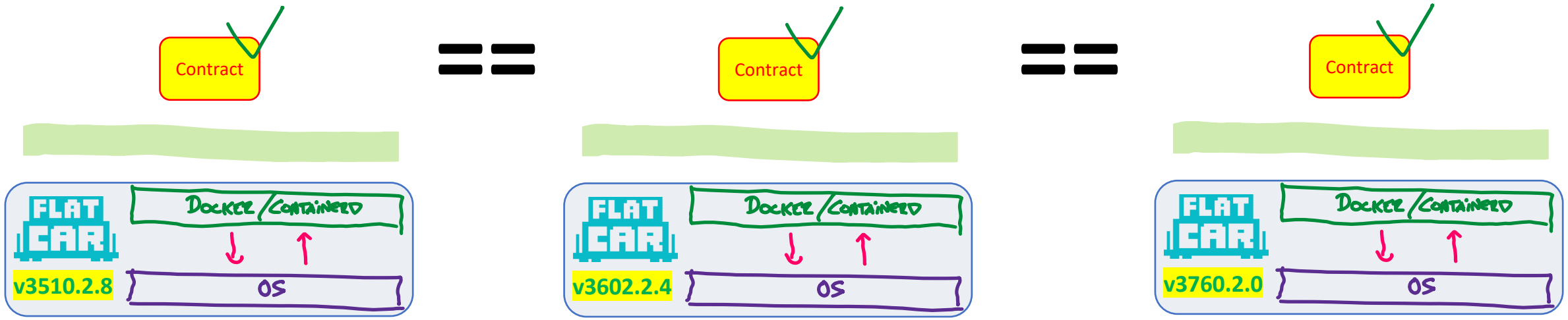
Contract is well-testable (and rigorously tested)



Interchangeable OS

Contract is well-tested

Always upheld across releases

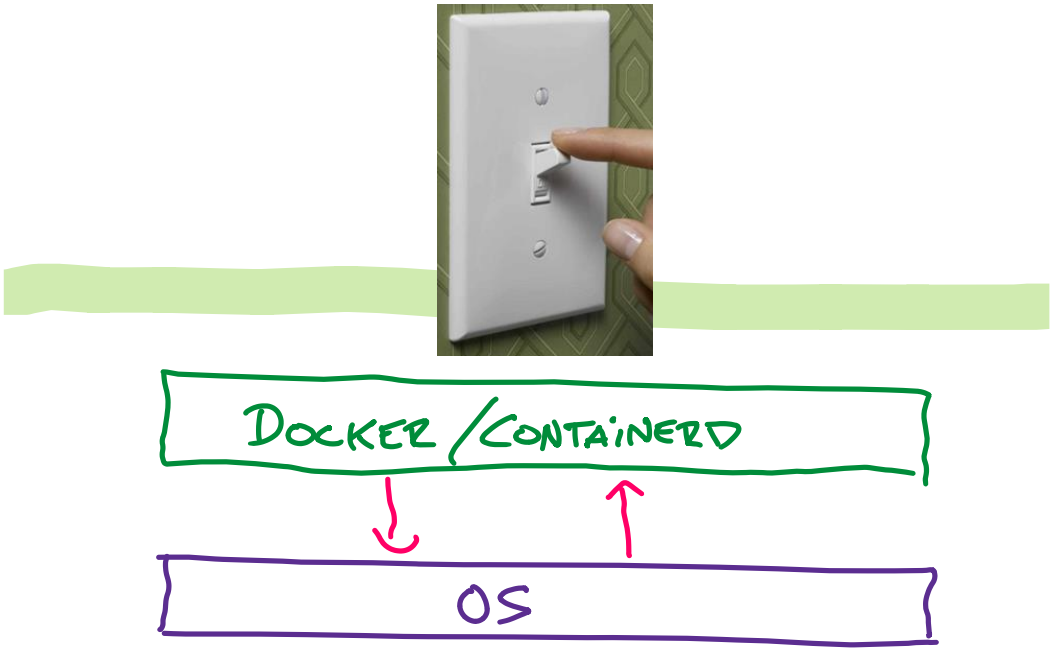


Interchangeable OS

Contract is well-tested

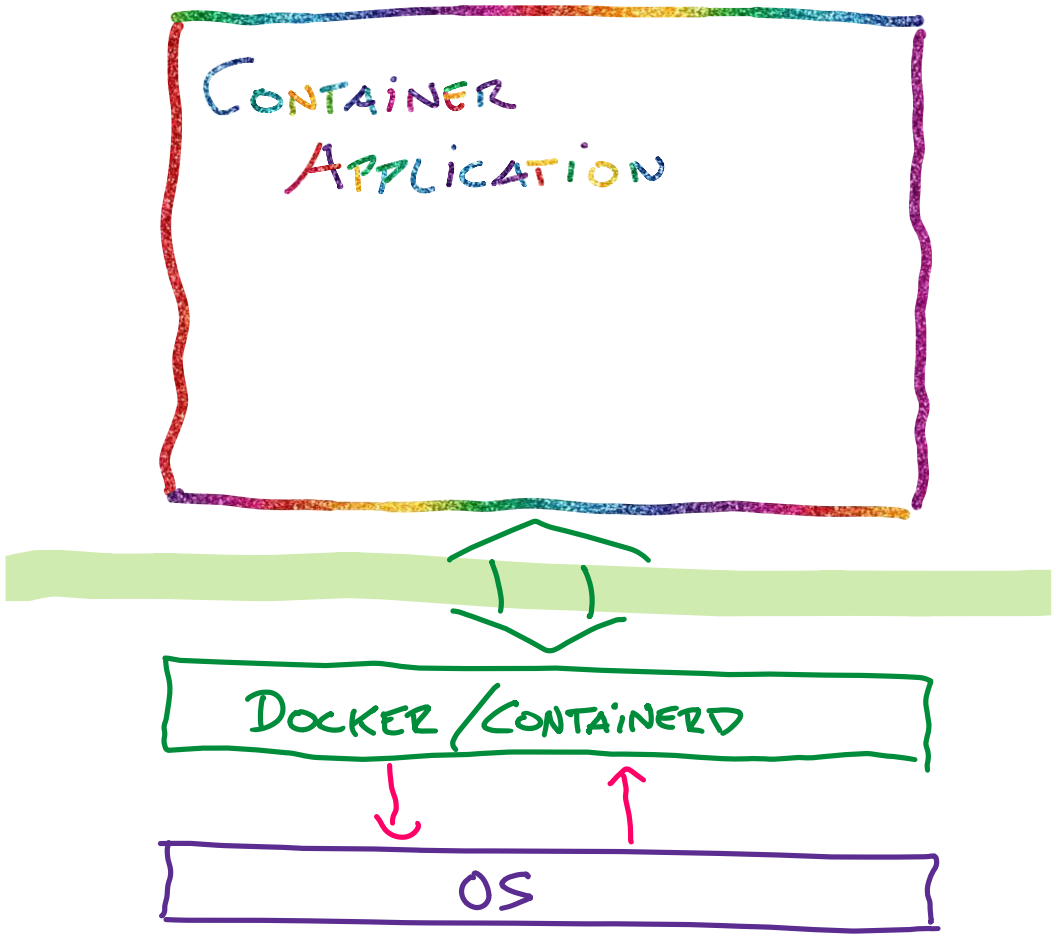
Always upheld across releases

Contract is our “light switch”



Staying up to date

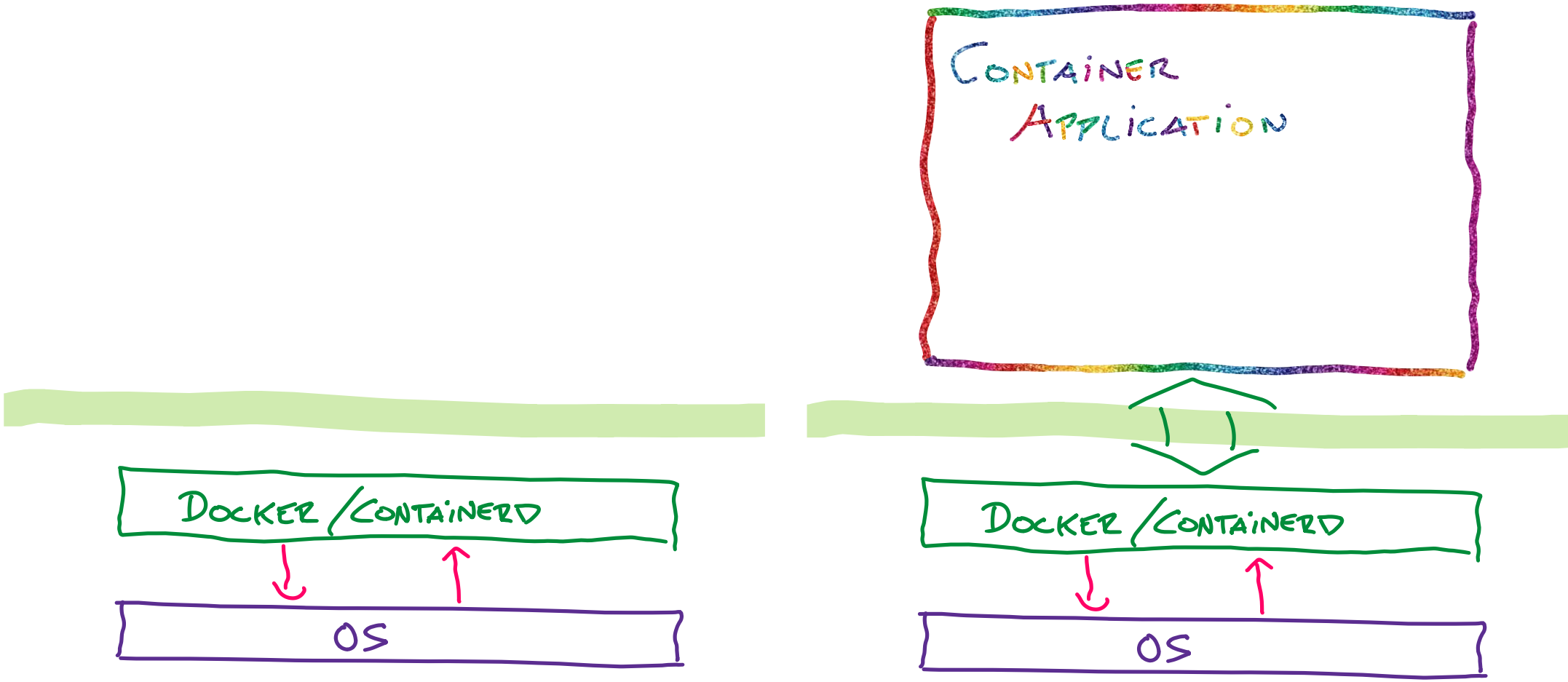
Atomic In-Place Updates



Staying up to date

Atomic In-Place Updates

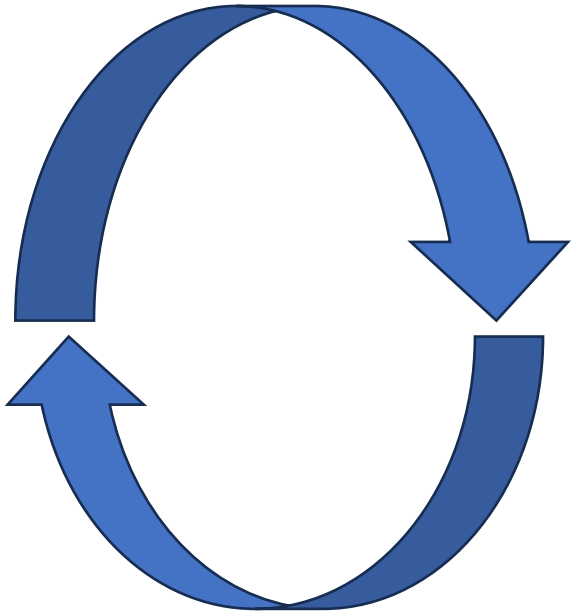
1. Stage

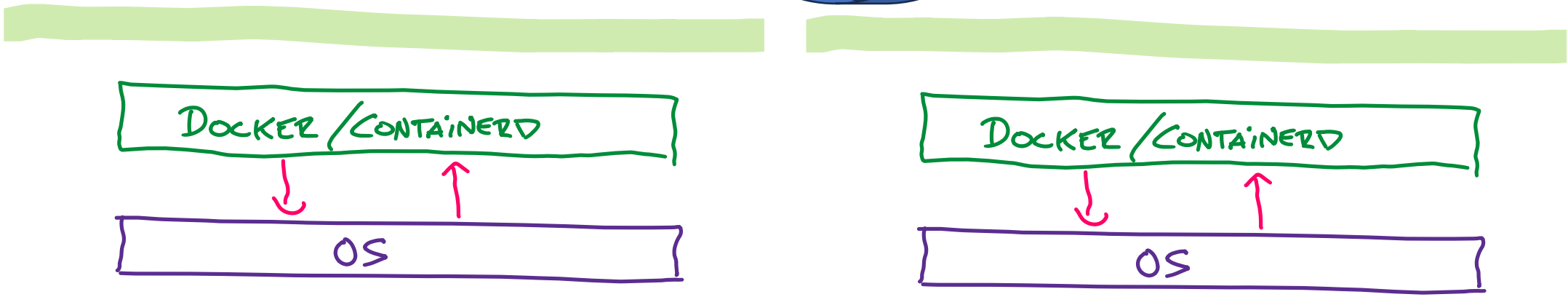


Staying up to date

Atomic In-Place Updates

1. Stage
2. Activate (Reboot)

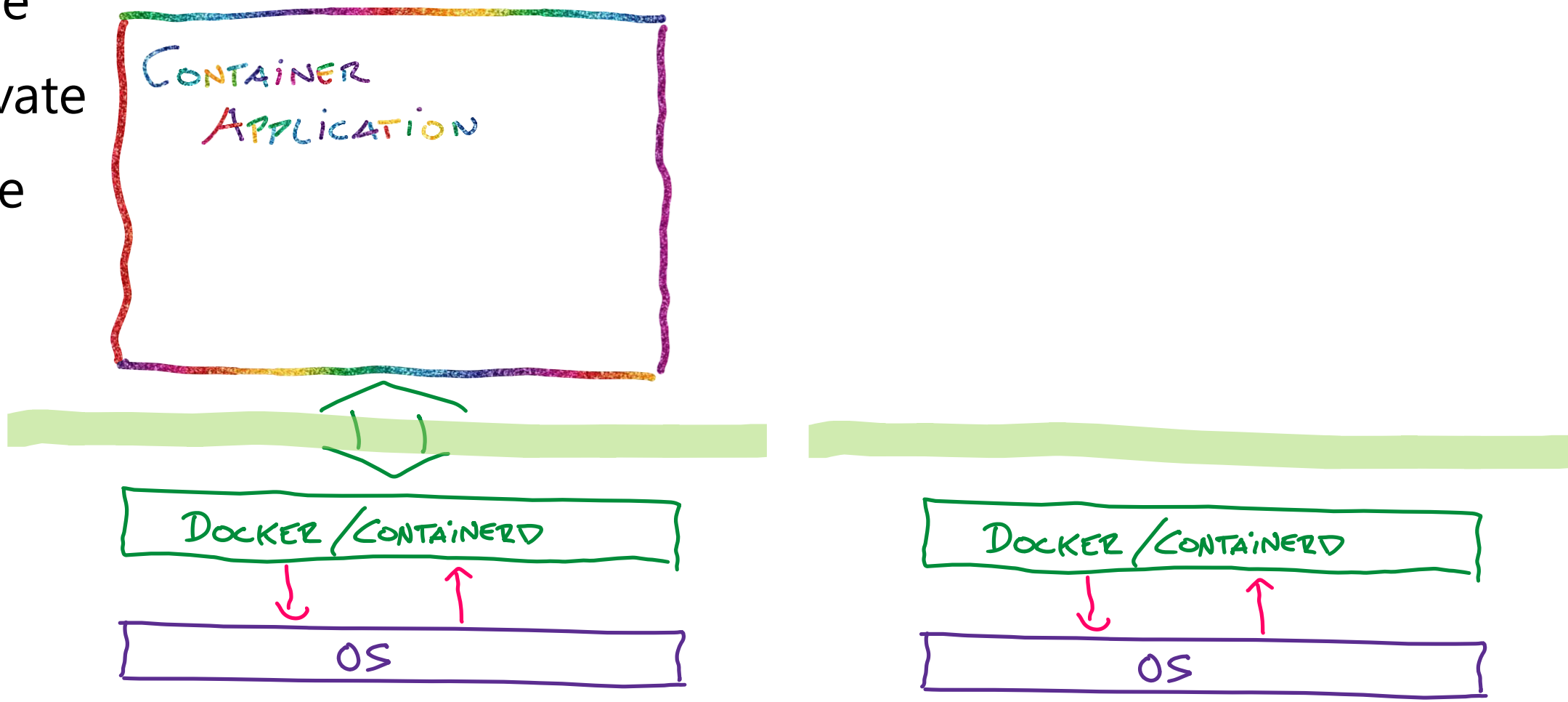




Staying up to date

Atomic In-Place Updates

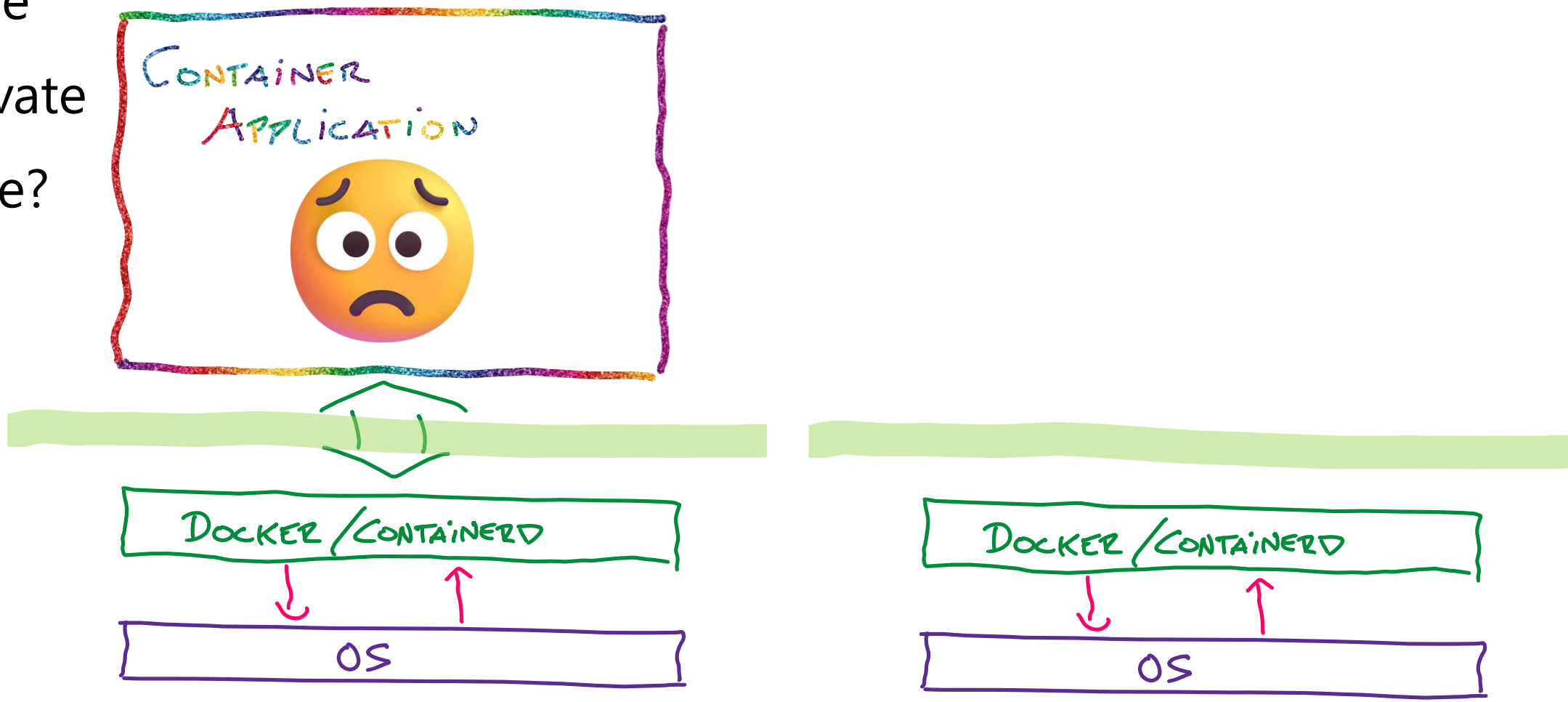
1. Stage
2. Activate
3. Done



Staying up to date

Atomic In-Place Updates

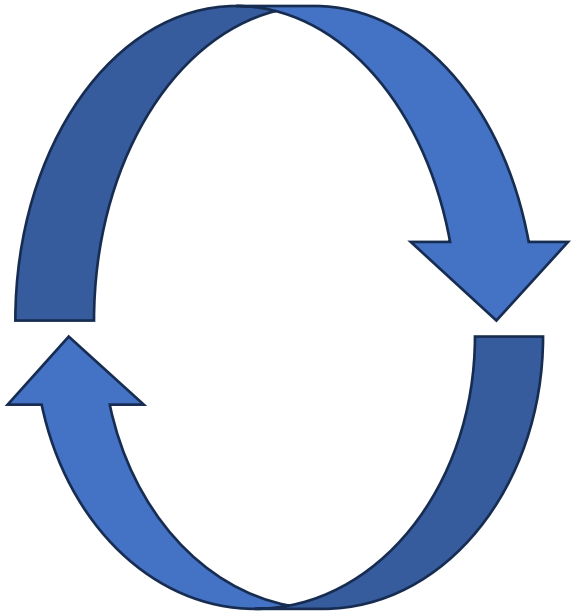
1. Stage
2. Activate
3. Done?

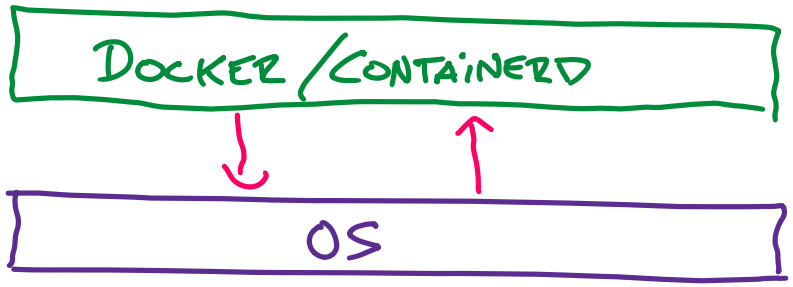


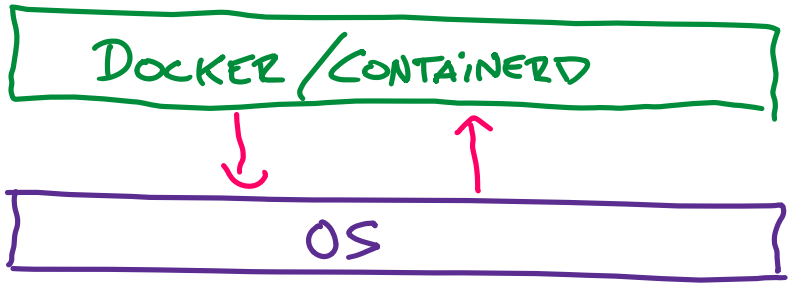
Staying up to date

Atomic Roll-Backs

1. Stage
2. Activate
3. Done?
4. Roll Back



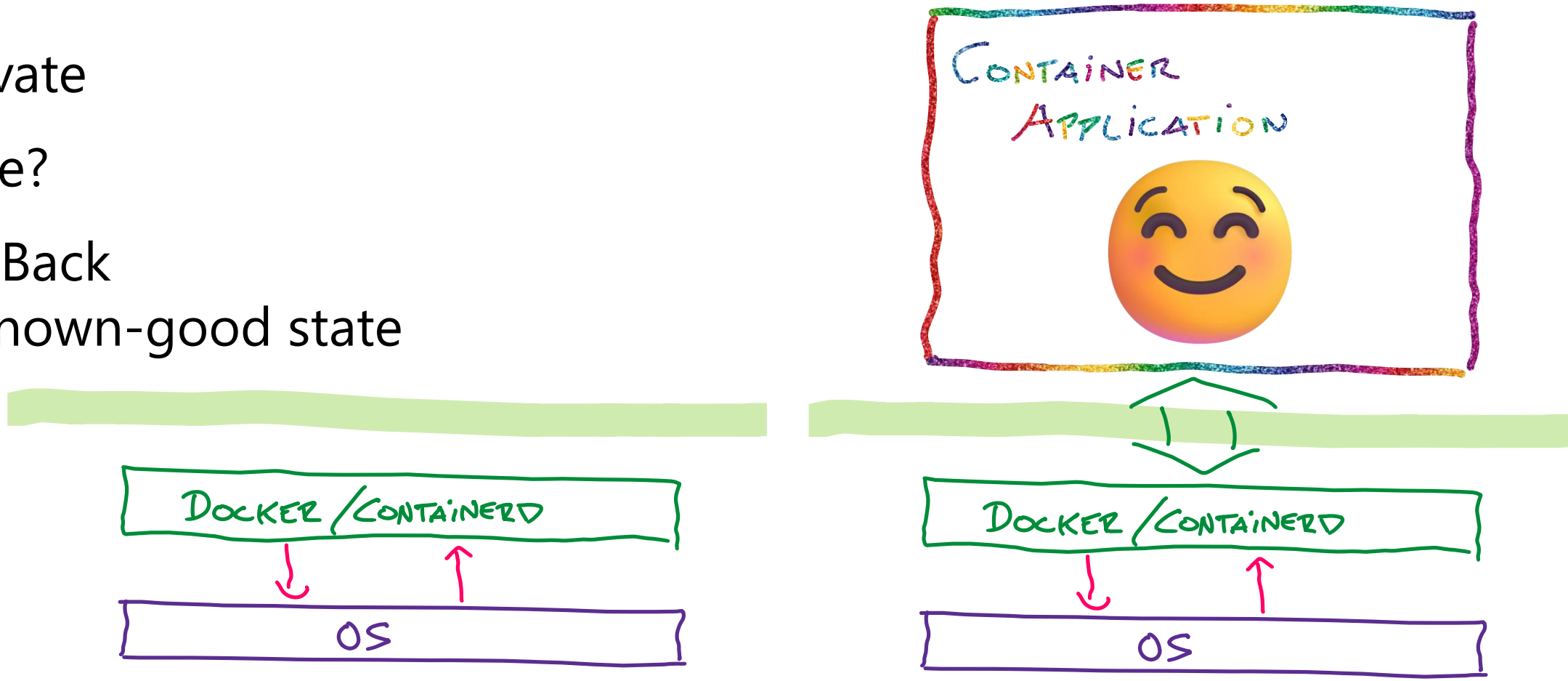




Staying up to date

Atomic Roll-Backs

1. Stage
2. Activate
3. Done?
4. Roll Back
   to known-good state



Update
Demo

(Usually automated.
Manual ONLY
for demo purposes)

Rolling out Updates

Updates need Reboots

- Maintenance windows (date / time)
- Sync via custom etcd lock (max number of nodes to reboot)
- Kubernetes: update operator (FLUO, KureD) w/ node draining, reboot, un-cordoning

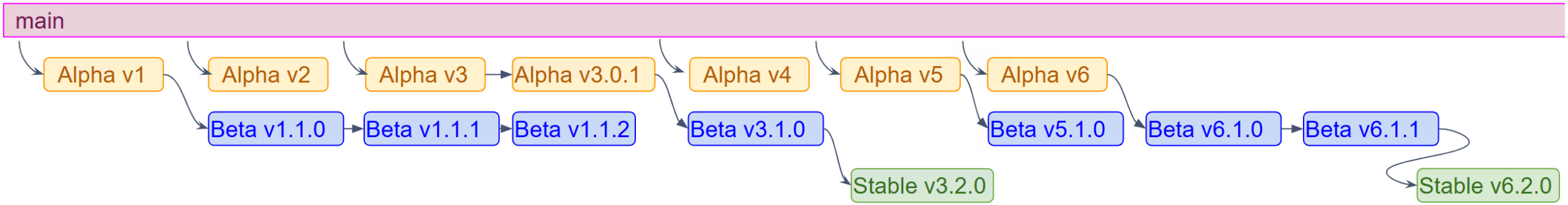
Stateful, FOSS update server

- Nebraska project - “Omaha” protocol used by chromium
- Easy to self-host. For large fleets – custom grouping, staggered roll-out, version overview, etc.

Users part of the stabilization process

- Run canaries and keep your workloads safe

Stabilisation Process: Ensuring *safe* updates



Major OS release stabilisation milestones:

"Alpha" Fully tested but may contain incomplete features. For developers.

"Beta" Fully tested for production use. Recommended for canaries

"Stable" For widespread production use.
Additional stabilisation through user feedback from Beta canaries.

Deployments defaults to "stable" but can be customised to any channel.

Stabilisation Process: Ensuring *safe* updates

Use stable for most workloads, and run a few Beta canaries

- Each Beta is fully tested
- Canaries smoke-test incoming changes and detect issues with your workload
  - (And roll-back is easy!)

Report Issues detected by canaries

- The issue will be fixed in the next Beta, before changes go stable

==> Your clusters will receive stable versions that are proven to work

Composability

OS-level extensibility via Systemd Sysext

OS is immutable

- Nice set of tools, but I need podman/Kubernetes/WASM/…

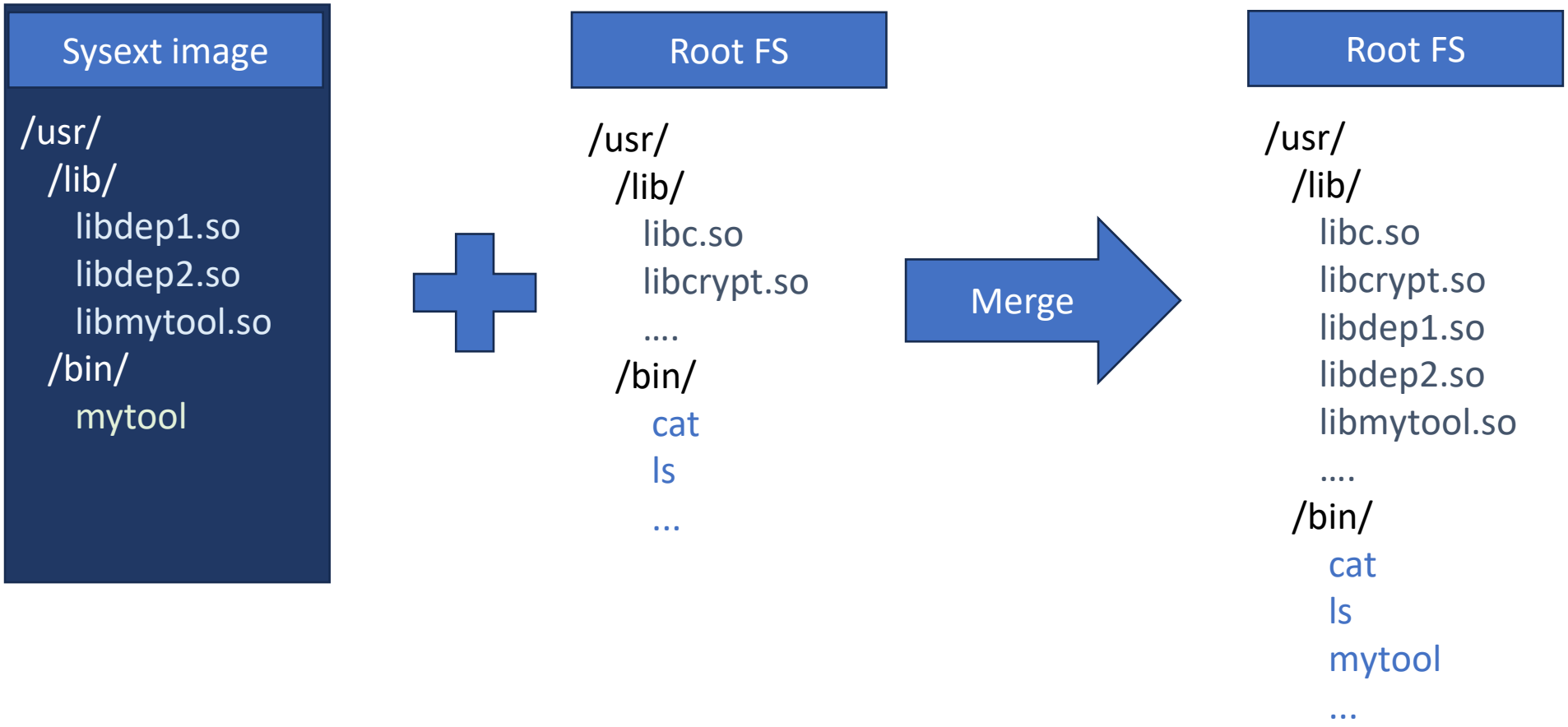
Extensible via systemd-Sysexts

- Immutable filesystem images that ship custom libraries / binaries as full root FS tree
  (only /usr and /opt subtrees supported)
- A/B updates independent from OS via systemd-sysupdate (via HTTPS server, e.g. Github Release)

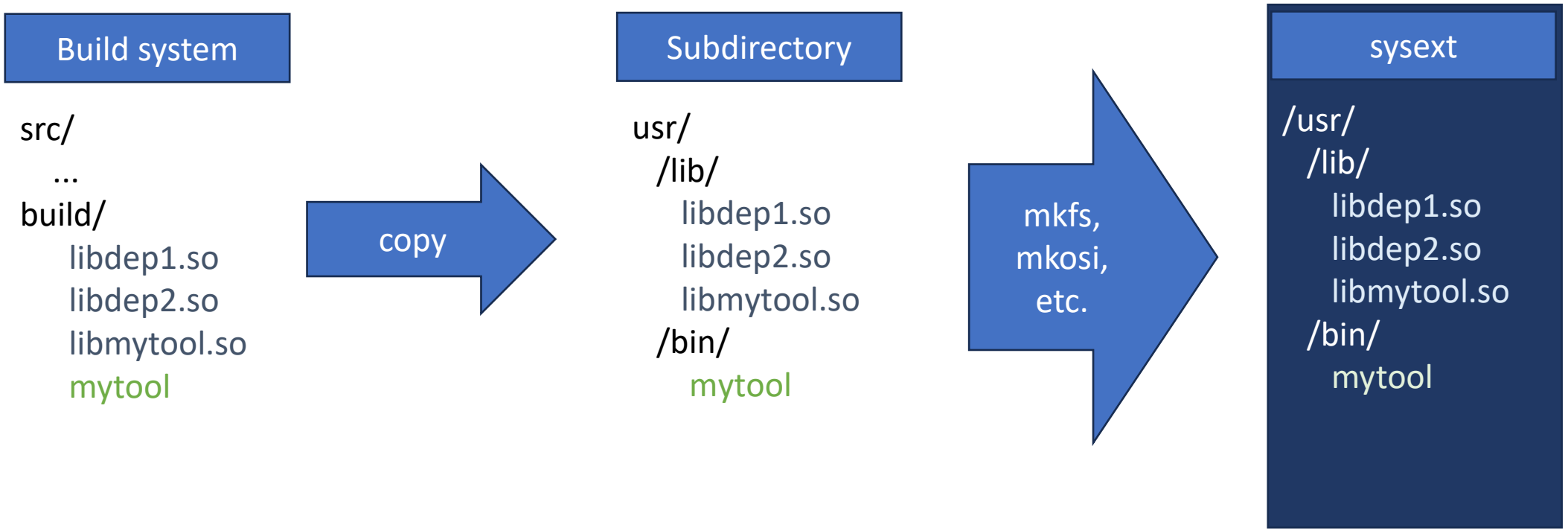
Flatcar makes extensive use of sysexts

- Bundled with the base OS and updated in lock-step, e.g. OEM / guest tools
- Independent of the base OS with custom update cycle, e.g. Kubernetes sysext for CAPI, WASM, …

Using Sysexts



Building Sysexts



Sysexts Day #2

Solid update story

- HTTPS endpoint + index file ("SHA256SUMS")
- Complimentary systemd-sysupdate service for staging + updating atomically
- Can be updated independently of the OS, but also support OS version requirements

Simple, easy development / publish process

- Straightforward packaging as filesystem image
- Lightweight metadata requirements

Clean separation between OS + customisations

- Independent of the base OS with custom update cycle, e.g. Kubernetes syst for CAPI, WASM, ...

Image composability in Flatcar

Pre-bake images

- Add custom sysexts + configuration to stock Flatcar release image
- Update via self-hosted sysexts (e.g. gitops via github actions)

Compose at provisioning time

- Use declarative configuration to download & configure, sysupdate to update

CAPI pilot

- Proof-of-concept Kubernetes sysext composed into stock image during provisioning
- CAPO, Tinkerbell are supported, CAPA, CAPZ, and CAPV work in progress.

Sysext Demo

Community

Flatcar Community

Community-driven FOSS project

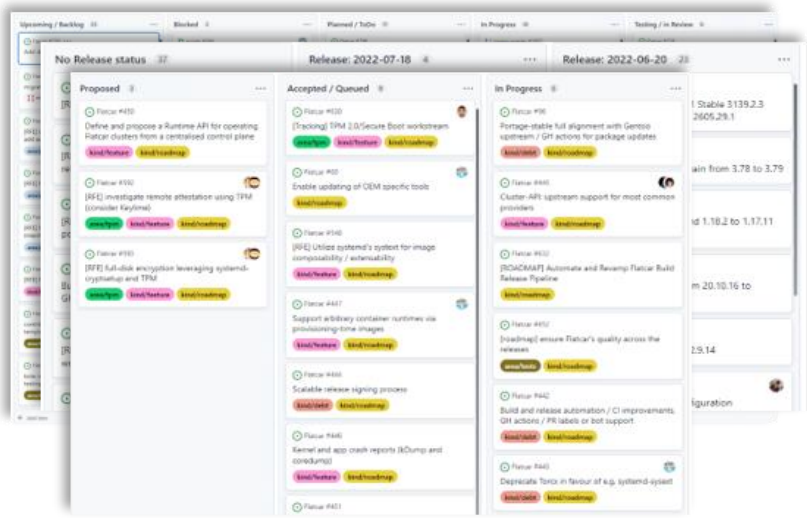
- No single vendor, full community stewardship
- Submitted to the CNCF as incubation project (ongoing)

Matrix, Slack - Our day-to-day comms

Office hours - Every 2nd Tuesday, 3:30pm UTC

Dev Sync - Every 4th Tuesday, 3:30pm UTC

Roadmap, Implementation, Releases



Portable, Easy to use SDK

Focus on low entry bar to OS Development

(Some Gentoo knowledge is useful though)

Used by Maintainers and in our automation

Includes easy-to-run, full test suite

```
git clone https://github.com/flatcar/scripts.git
cd scripts
git checkout alpha-3794.0.0

./run_sdk_container –t ./build_packages
./run_sdk_container –t ./build_image
./image_to_vm.sh --from=../build/images/amd64-usr/latest/ \
                 --format=qemu_uefi --image_compression_formats none

./run_local_tests.sh
```

Wrap Up

Leverage Isolation of OS and Apps

Declarative Configuration at Provisioning

Atomic, Automated Updates

Composable images with Sysext

Community driven, submitted to CNCF

Thank you

The Community's
Container Linux