"Day 2 Ops"

Linux for Kubernetes and Container Workloads

FrOSCon 2024

Hello, I'm

Thilo

Thilo Fromm

Flatcar Maintainer

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

"Day 2"?

Day 0

Visionary Ideas / Brainstorming Phase

Large parts don't exist yet, focus on Key Feature(s)

Very limited exposure to users ("closed Alpha")

Code has "Proof of Concept" quality

Operations are manual, no time for automation

Things constantly break

Day 1

Project goes live / emerges from "stealth mode"

Main features are implemented

Service meets reality as users on-board

Spotty, gappy Automation

Much firefighting

"Test in Production", but not in a good way

Day 2

Project is Mature, in steady state

Deterministic feature lifecycle

No user-visible service degradations / interruptions

Operations are well-defined, extensively automated

Service impacts are well managed

"Boring", in a good way


"Day 2" OS

Day 2 OS

Deployments are fully automated and reproducible

Reliable update / lifecycle management

Robust and safe, proven support of your target environment

Solid configuration management / inventory management (SBOM)

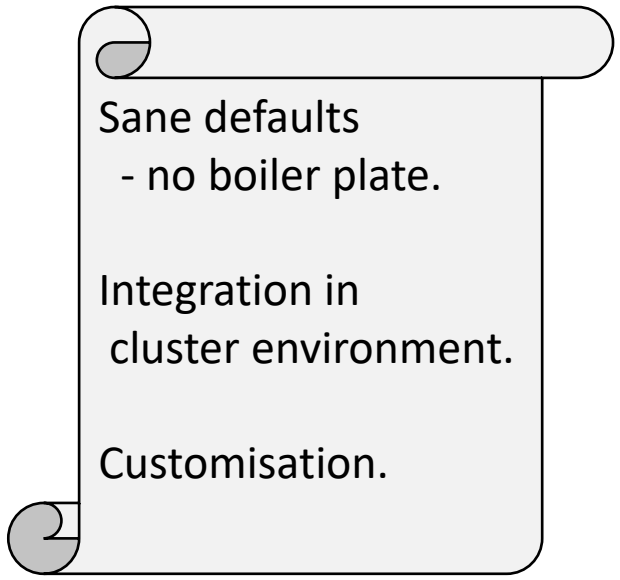
Integrates well into operational processes

On Kubernetes, we operate workloads as interchangeable, replaceable commodities ("cattle, not pets").
Can we translate this philosophy to operating systems?

Fully Automated Deployments

Kubernetes automates service deployments

Configure -> Deploy -> Operate



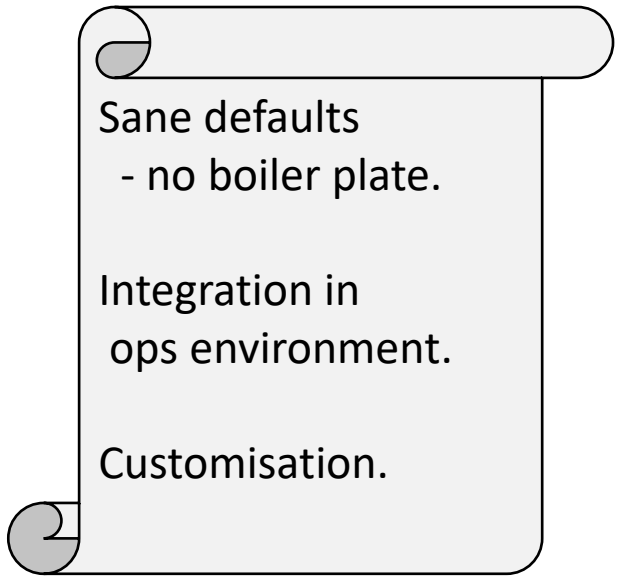
my-service.yaml

`kubectl apply`

Automated
- Config management
- Version management
- Lifecycle management

Flatcar automates Node deployments

Configure -> Deploy -> Operate



Butane-config.yaml

Custom data,
User data,
Http, or
[i]PXE

ignition.json

Automated
- Self-configuration
- Unattended updates

Declarative configuration, before provisioning


```yaml
variant: flatcar
version: 1.0.0

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

storage:
  files:
    - path: /srv/www/html/index.html
      mode: 0644
      user:
        name: caddy
      contents:
        inline: |
          <html><body align="center">
          <h1>Hallo FrOSCon 2024!</h1>
          <img src="froscon_logo_print_color.png" alt="FrOSCon logo"
          </body></html>
    - path: /srv/www/html/froscon_logo_print_color.png
      mode: 0644
      user:
        name: caddy
      group:
        name: caddy
      contents:
        local: froscon_logo_print_color.png

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

Provisioning Demo

Provisioning

Reproducible, Cloud Agnostic

Zero touch, Automatable

Supports complex deployments (See e.g. the Flatcar Jitsi server)

No config drift

Integrates into existing automation (Go, Terraform/OpenTOFU, ClusterAPI)

Day 2 OS

- Fully automated, easy to deploy
- Reliable update / lifecycle management
- Robust and safe, proven support of your target environment
- Solid configuration management / inventory management (SBOM)
- Integrates well into operational processes

On Kubernetes, we operate workloads as interchangeable, replaceable commodities ("cattle, not pets").
Can we translate this philosophy to operating systems?

Reliable Update / Lifecycle Management

Container Apps are isolated

No Dependencies to OS Libraries

Container images are self-sufficient

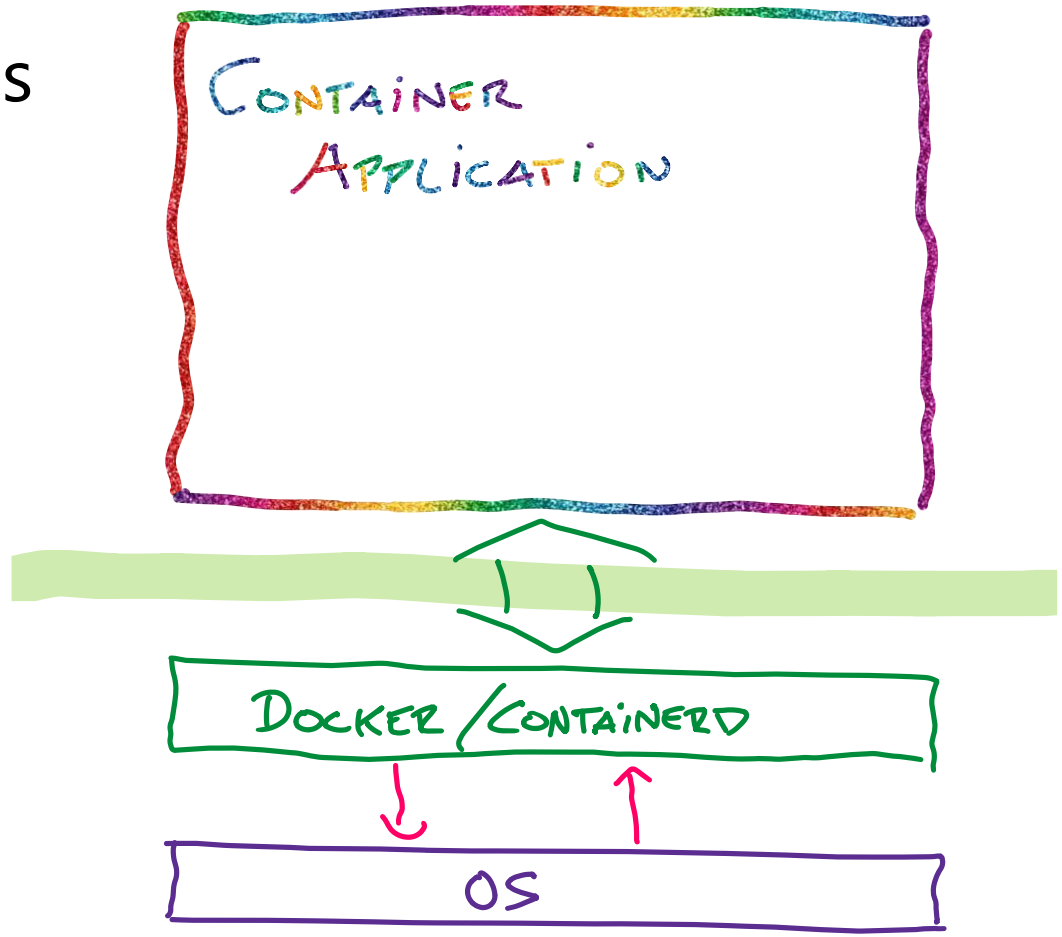
No Dependencies to OS Tools and Services

Only require Container Runtime

No Dependencies to OS Configuration

App config cleanly separated from OS

➔ Portable Applications



And so is the OS!

Well-defined interfaces OS <-> App

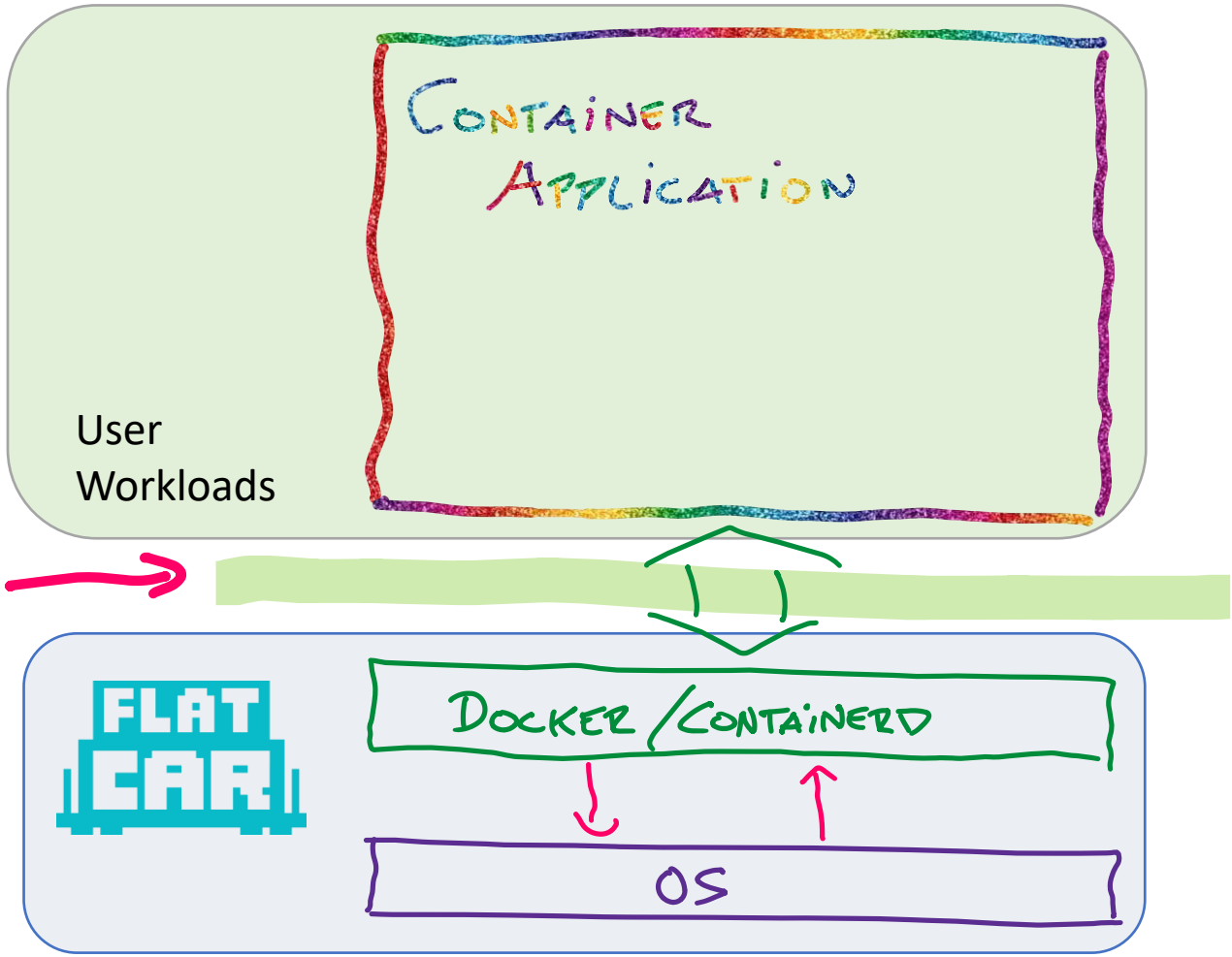
- Very few components, easy to test thoroughly
- No other inter-dependencies
- Container apps isolate from the OS

Runtime + Kernel == Contract

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

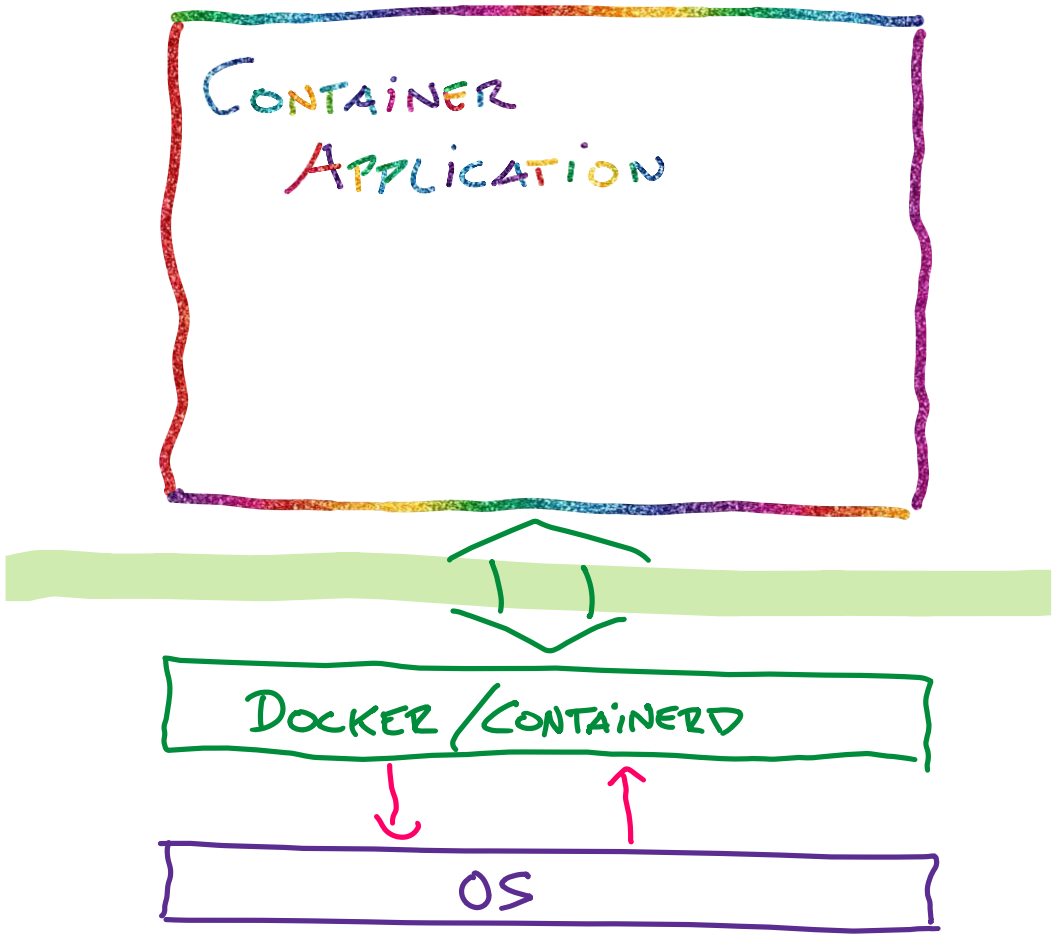
➔ Interchangeable OS

- Just uphold the contract!



Staying up to date

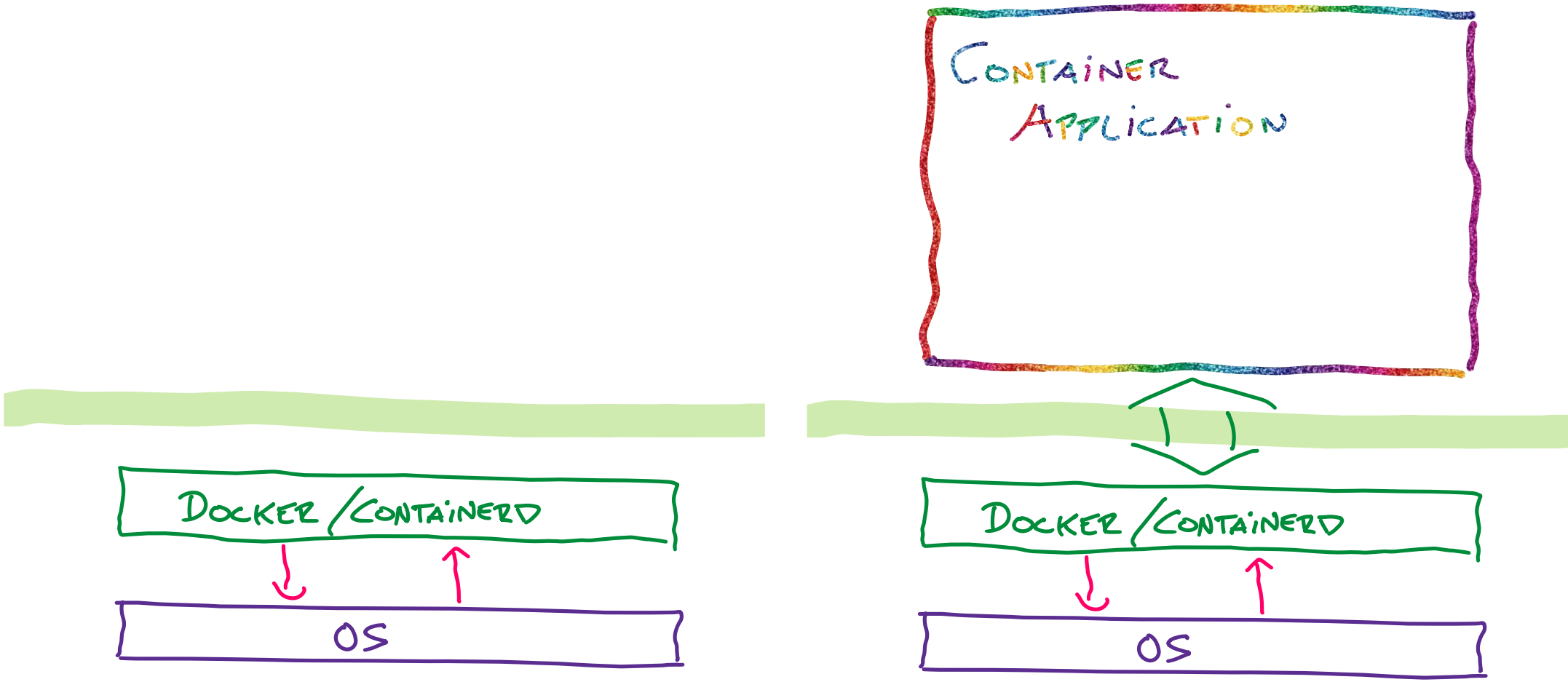
Atomic In-Place Updates



Staying up to date

Atomic In-Place Updates

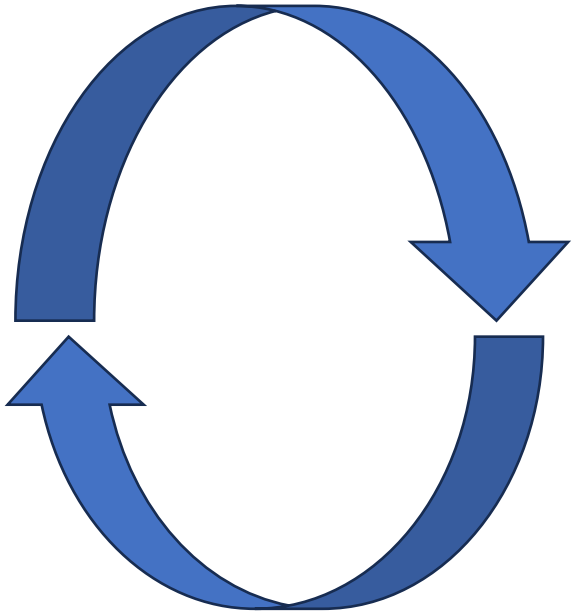
1. Stage

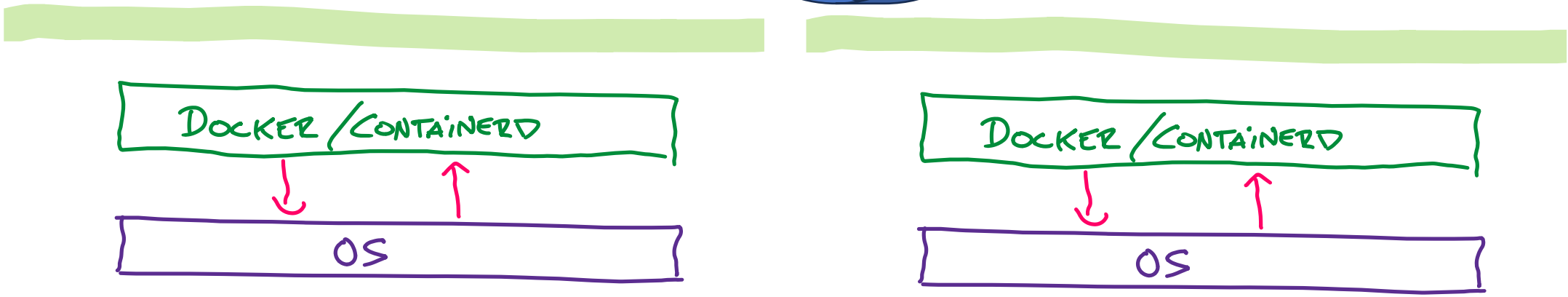


Staying up to date

Atomic In-Place Updates

1. Stage
2. Activate (Reboot)

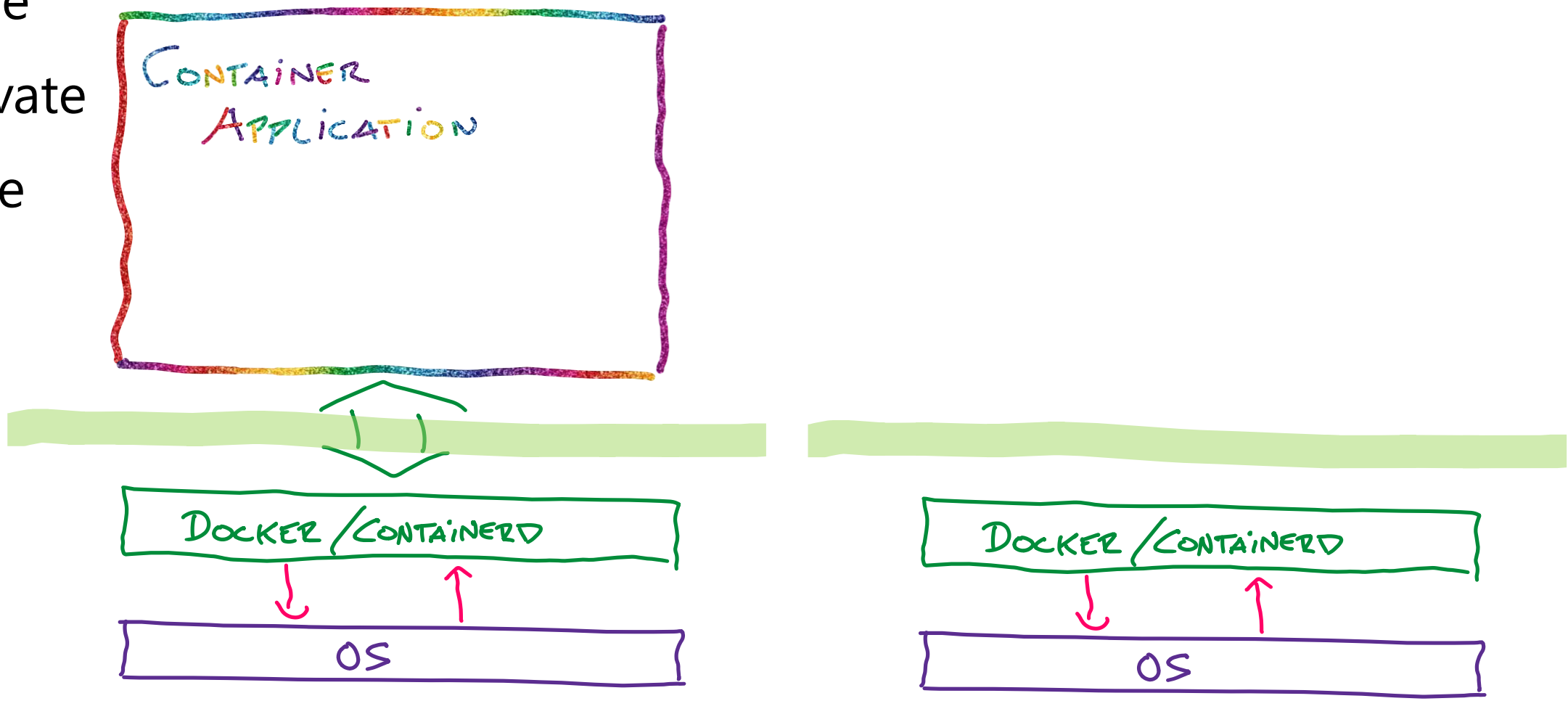




Staying up to date

Atomic In-Place Updates

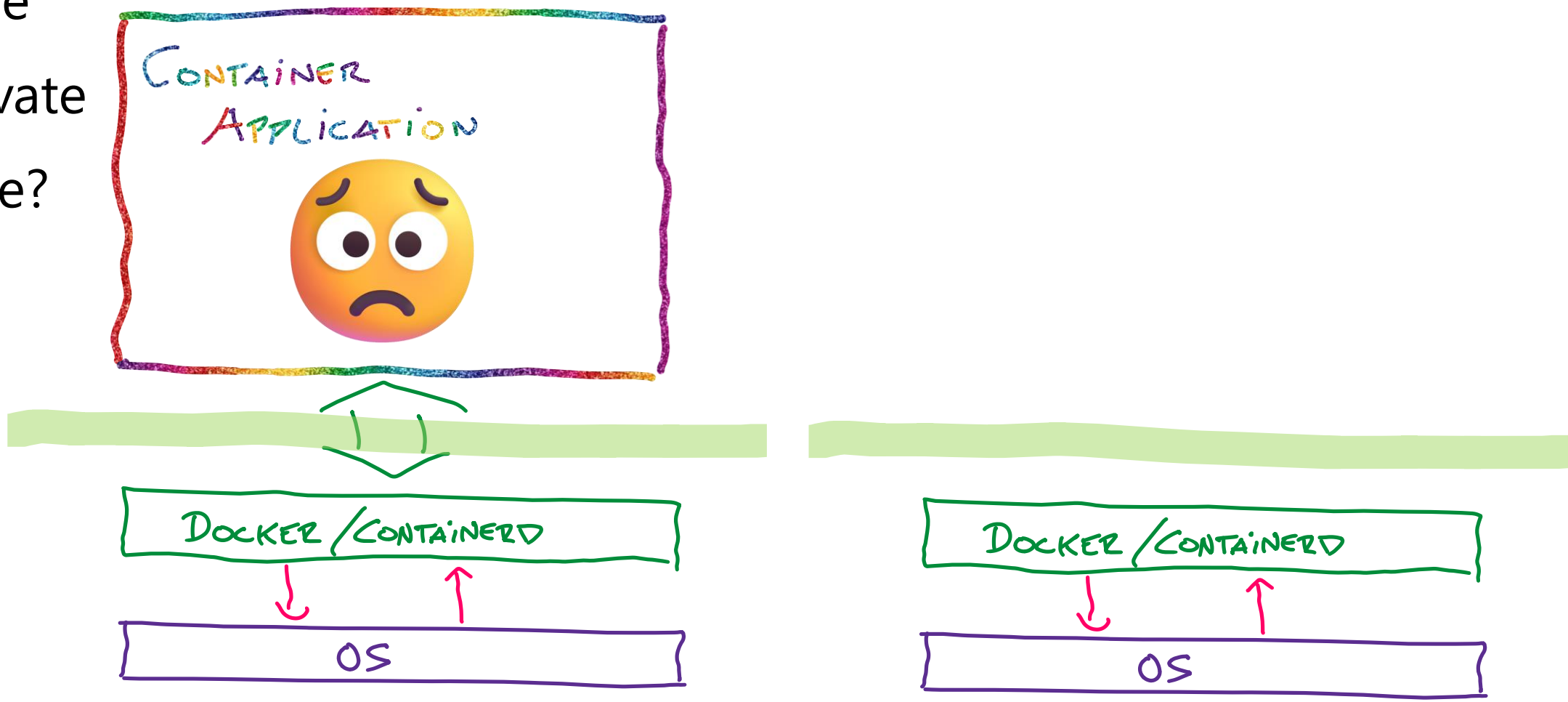
1. Stage
2. Activate
3. Done



Staying up to date

Atomic In-Place Updates

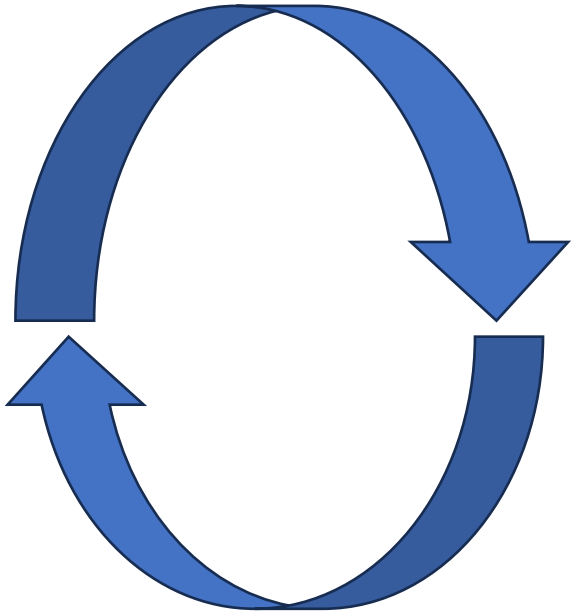
1. Stage
2. Activate
3. Done?

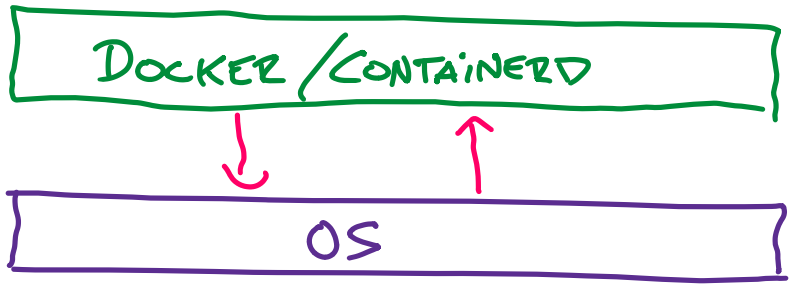


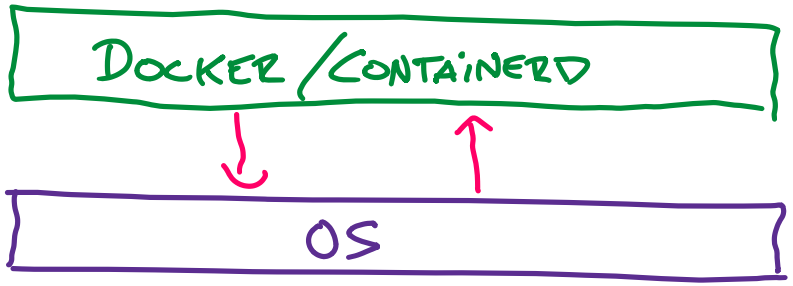
Staying up to date

Atomic Roll-Backs

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



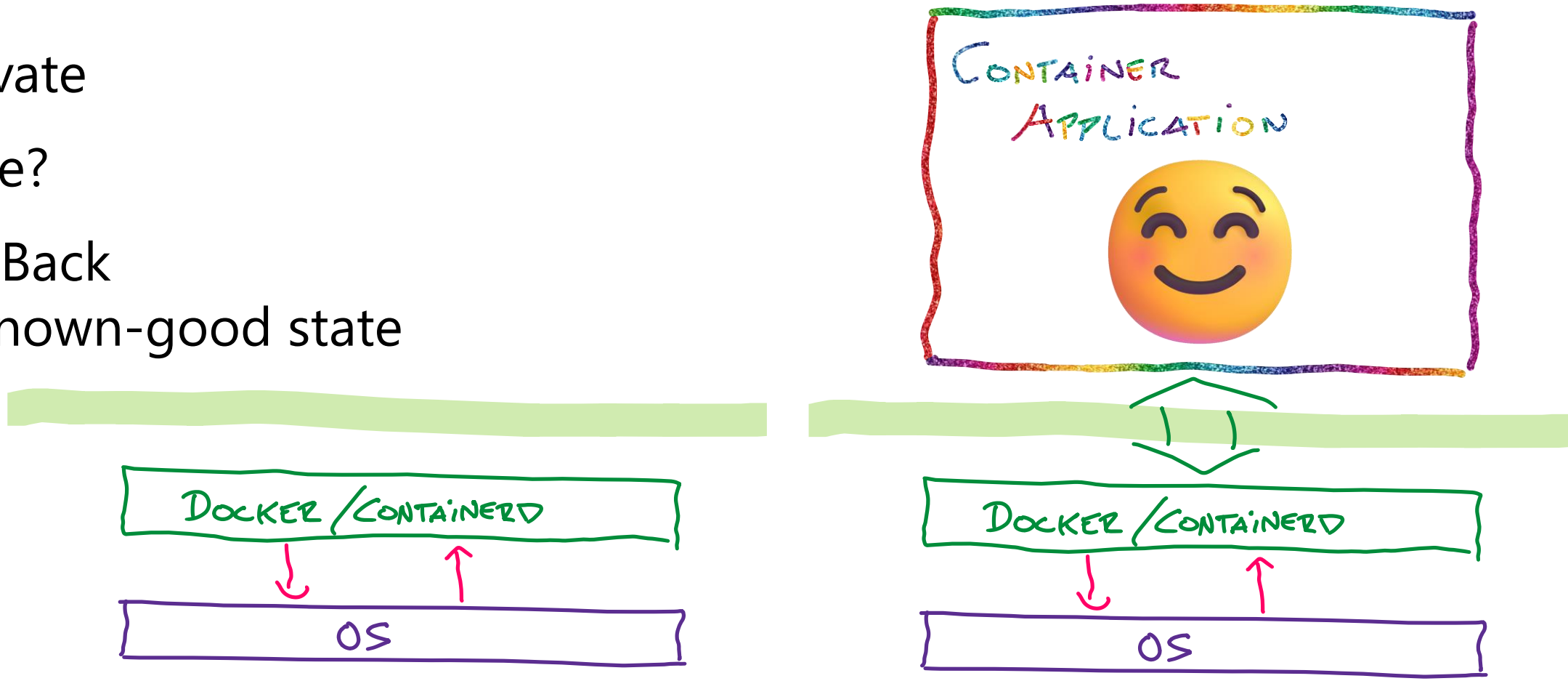




Staying up to date

Atomic Roll-Backs

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



Update Demo

Rolling out Updates

Defined Maintenance Windows

Update Operator for larger Clusters

Use Beta Canaries

Host-your-own stateful FOSS update server (Nebraska sub-project)

Day 2 OS

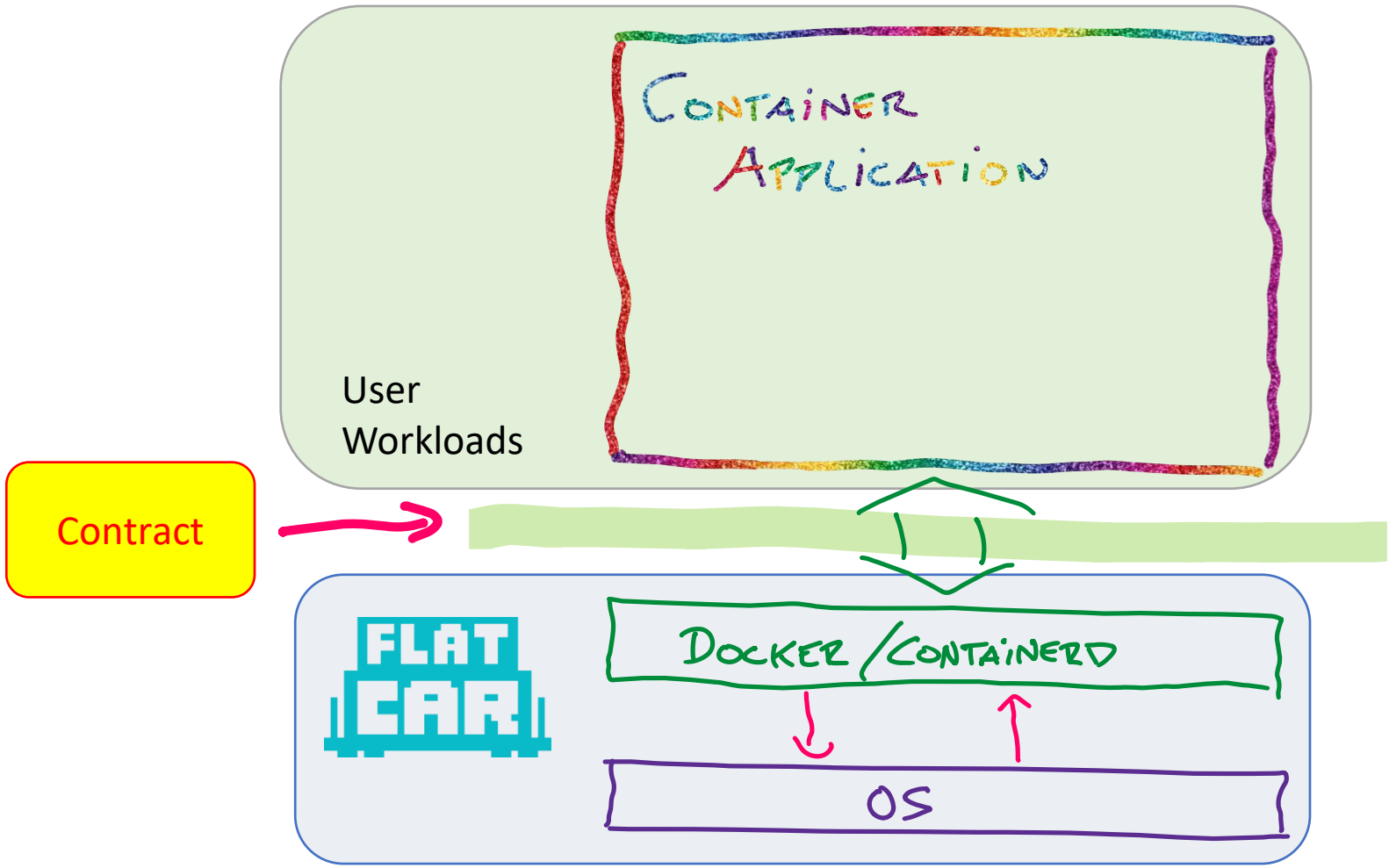
- Fully automated, easy to deploy
- Reliable update / lifecycle management
- Robust and safe, proven support of your target environment
- Solid configuration management / inventory management (SBOM)
- Integrates well into operational processes

On Kubernetes, we operate workloads as interchangeable, replaceable commodities ("cattle, not pets").
Can we translate this philosophy to operating systems?

Robust and Safe

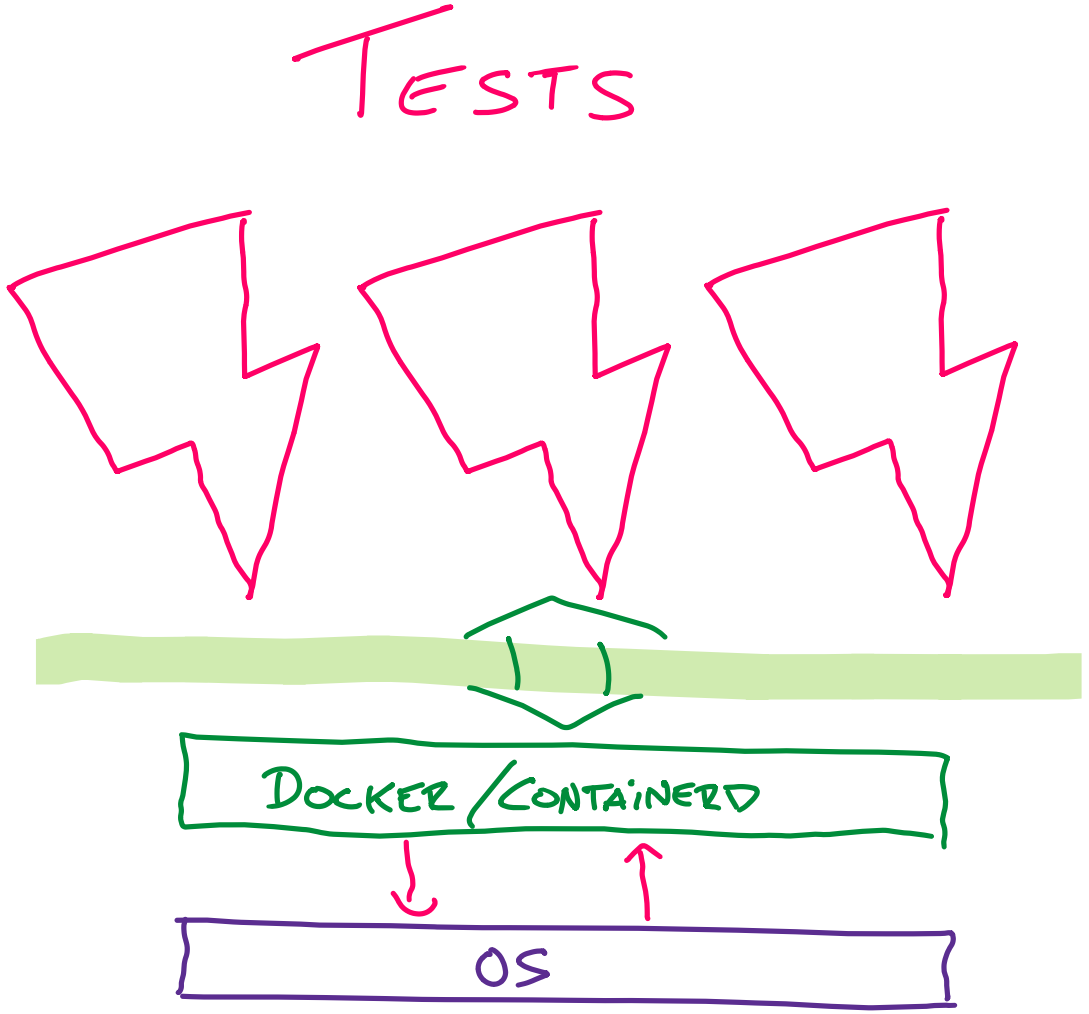
Remember the Contract?

Container Runtime + Kernel == Contract



Use the same contract for tests

Contract is well-testable (and rigorously tested)



What's in a test run?

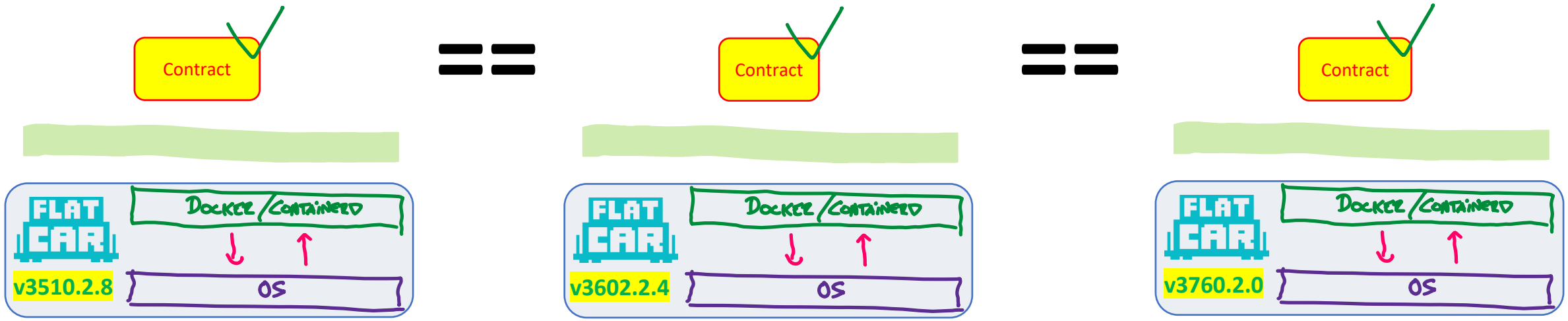
More than 100 tests, including complex Kubernetes scenarios like CNIs

Run on every release (all vendors), every CI build (QEMu) for almost every commit to the OS repo.

Contract is upheld across releases

Contract is well-tested

Always upheld across releases



Day 2 OS

- ✓ Fully automated, easy to deploy
- ✓ Reliable update / lifecycle management
- ? Robust and safe, proven support of your target environment
- Solid configuration management / inventory management (SBOM)
- Integrates well into operational processes

On Kubernetes, we operate workloads as interchangeable, replaceable commodities ("cattle, not pets").
Can we translate this philosophy to operating systems?

Day 2 OS

- ✓ Fully automated, easy to deploy
- ✓ Reliable update / lifecycle management
- ? Robust and safe, proven support of your target environment
- Solid configuration management / inventory management (SBOM)
- Integrates well into operational processes

On Kubernetes, we operate workloads as interchangeable, replaceable commodities ("cattle, not pets").
Can we translate this philosophy to operating systems?

Major updates: Channel Progression

Release Channels

Stable 3760.2.0

amd64 arm64

Release Date: Jan 18, 2024

The Stable channel is intended for use in production clusters. Versions of Flatcar Container Linux have been tested as they move through Alpha and Beta channels before being promoted to stable.

containerd - 1.7.7
docker - 20.10.24
ignition - 2.15.0
kernel - 6.1.73
systemd - 252

Beta 3815.1.0

amd64 arm64

Release Date: Jan 18, 2024

The Beta channel is where Flatcar Container Linux stability is solidified. We encourage including some beta machines in production clusters in order to catch any issues that may arise with your setup.

containerd - 1.7.10
docker - 24.0.6
ignition - 2.15.0
kernel - 6.1.73
systemd - 252

Alpha 3850.0.0

amd64 arm64

Release Date: Jan 18, 2024

The Alpha channel follows a more frequent release cadence and is where new updates are introduced. Users can try the new versions of the Linux kernel, systemd and other core packages.

containerd - 1.7.11
docker - 24.0.6
ignition - 2.15.0
kernel - 6.6.12
systemd - 252

LTS 3510.3.1

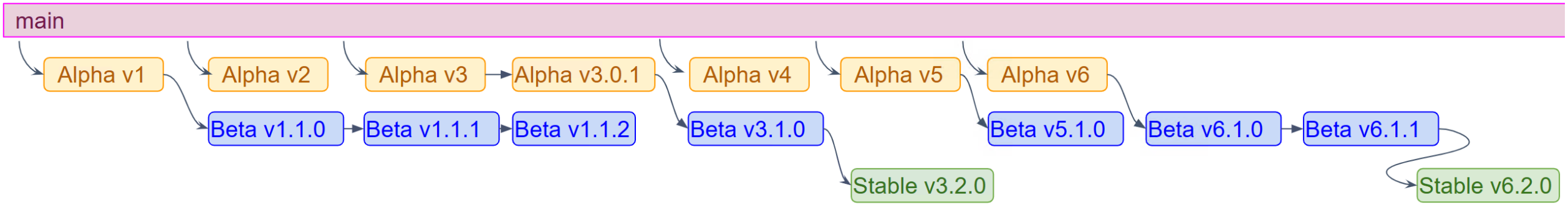
amd64 arm64

Release Date: Oct 25, 2023

LTS release streams will be maintained for an extended lifetime of 18 months. The yearly LTS streams have an overlap of 6 months.

containerd - 1.6.16
docker - 20.10.23
ignition - 2.14.0
kernel - 5.15.136
systemd - 252

Ensuring *safe* updates through channel progression



Major OS release stabilisation milestones:

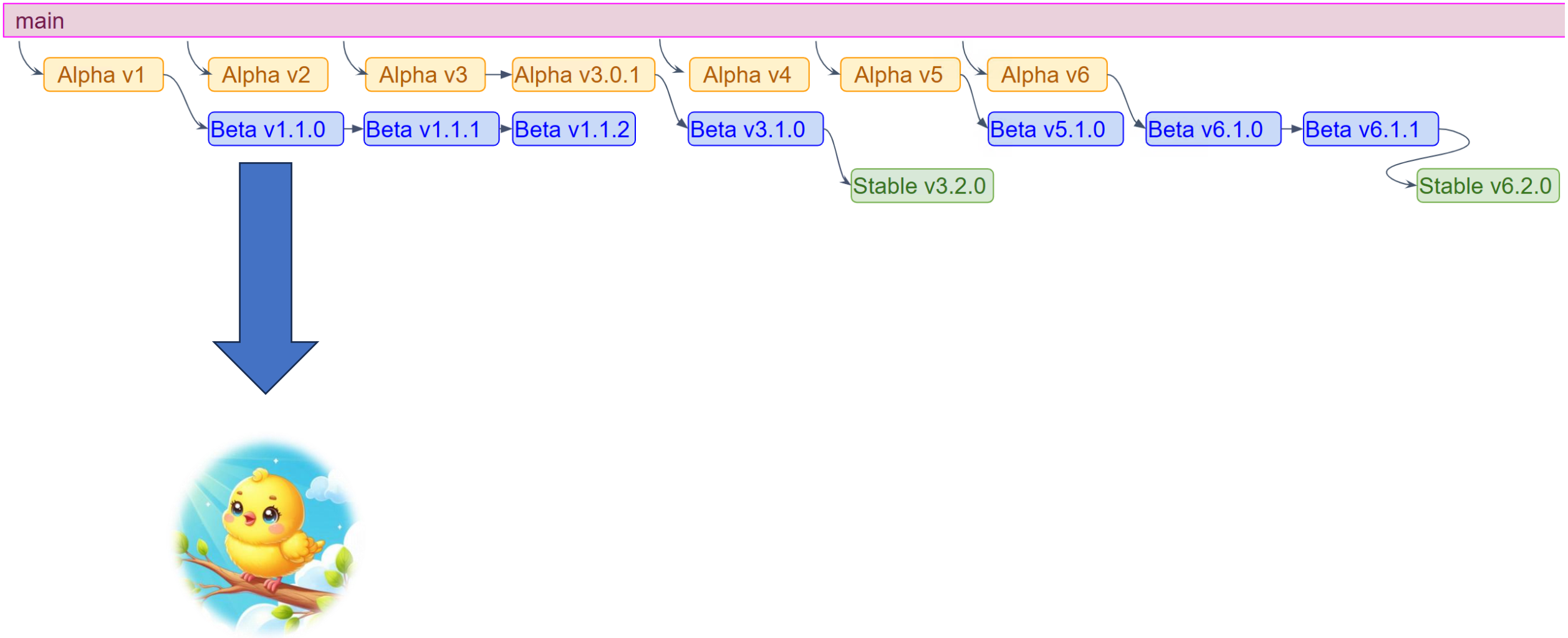
"Alpha" Fully tested but may contain incomplete features. For developers.

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

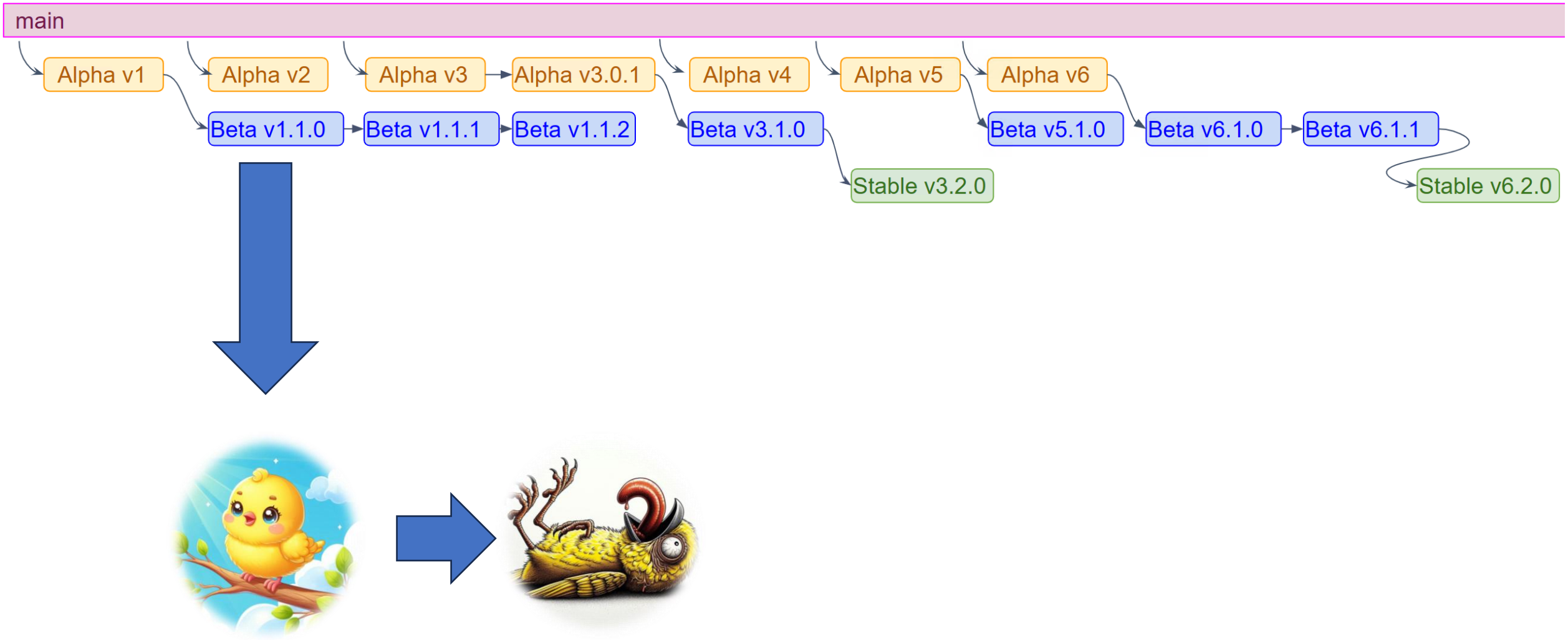
"Stable" For widespread production use.
Additional stabilisation through user feedback from Beta canaries.

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

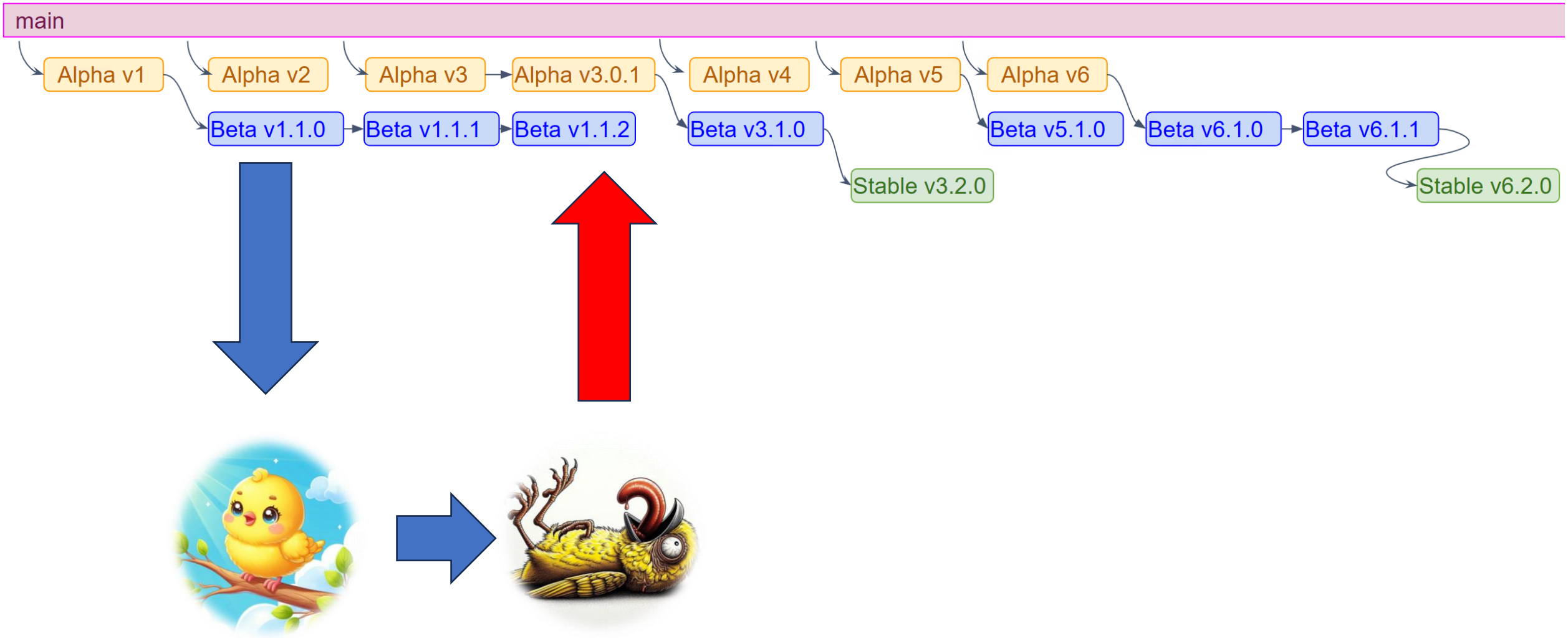
Ensuring *safe* updates through Beta canaries



Ensuring *safe* updates through Beta canaries



Ensuring *safe* updates through Beta canaries



Day 2 OS

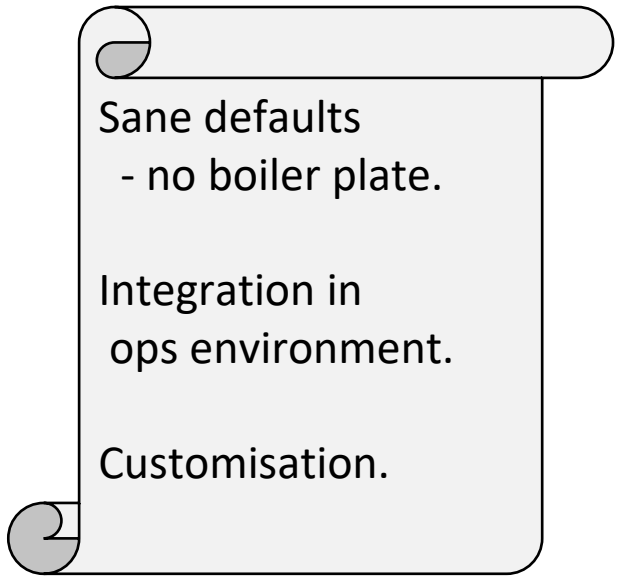
- Fully automated, easy to deploy
- Reliable update / lifecycle management
- Robust and safe, proven support of your target environment
- Solid configuration management / inventory management (SBOM)
- Integrates well into operational processes

On Kubernetes, we operate workloads as interchangeable, replaceable commodities ("cattle, not pets").
Can we translate this philosophy to operating systems?

Configuration and Inventory management

Configuration Management

Configure -> Deploy -> Operate



Butane-config.yaml

Custom data,
User data,
Http, or
[i]PXE

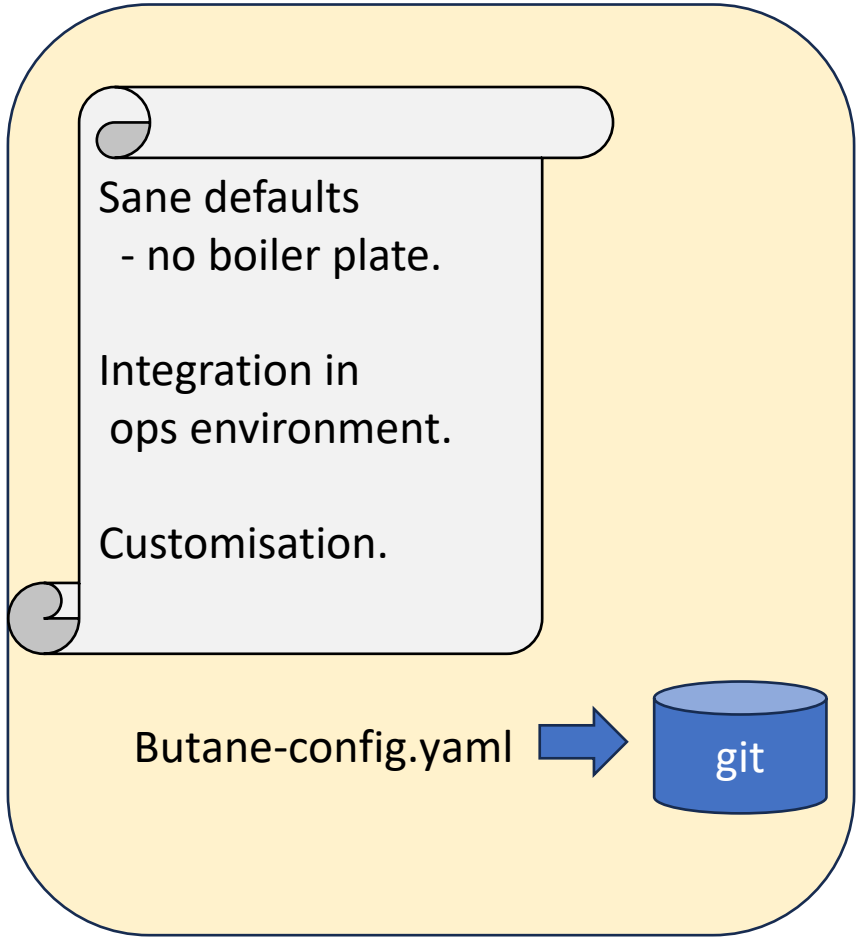
ignition.json

Applied ONCE.
At provisioning time.

Automated
- Self-configuration
- Unattended updates

Configuration Management

Configure -> Deploy -> Operate



Custom data,
User data,
Http, or
[i]PXE

ignition.json

Automated
- Self-configuration
- Unattended updates

Software Inventory Management

Software Inventory Management

Image-Based OS:

- Provisioned from a full disk image.
- Updated by a full OS partition image.
- Immutable: No package manager, no post-deplomentinstallations.

Software Inventory Management

Image-Based OS:
- Provisioned from a full disk image.
- Updated by a full OS partition image.
- Immutable: No package manager, no post-deplomentinstallations.

All packages and all files are documented for each release.
- No version drift.
- OS software inventory is pinned to OS release.

Day 2 OS

- Fully automated, easy to deploy
- Reliable update / lifecycle management
- Robust and safe, proven support of your target environment
- Solid configuration management / inventory management (SBOM)
- Integrates well into operational processes

On Kubernetes, we operate workloads as interchangeable, replaceable commodities ("cattle, not pets").
Can we translate this philosophy to operating systems?

Immutable OS Extensions

OS-level extensibility via Systemd Sysext

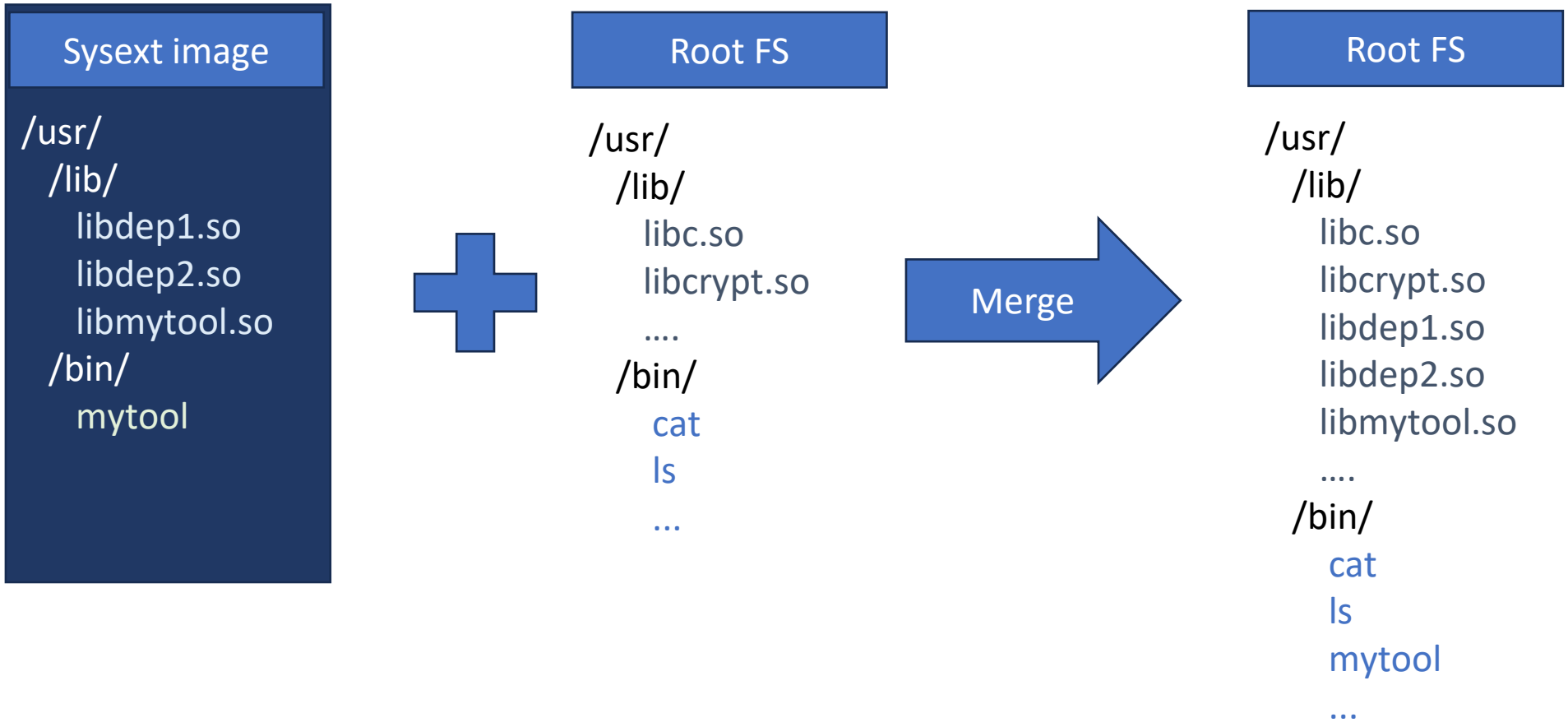
Nice set of tools, but I need podman/Kubernetes/WASM/...

Extensible via systemd-sysext:

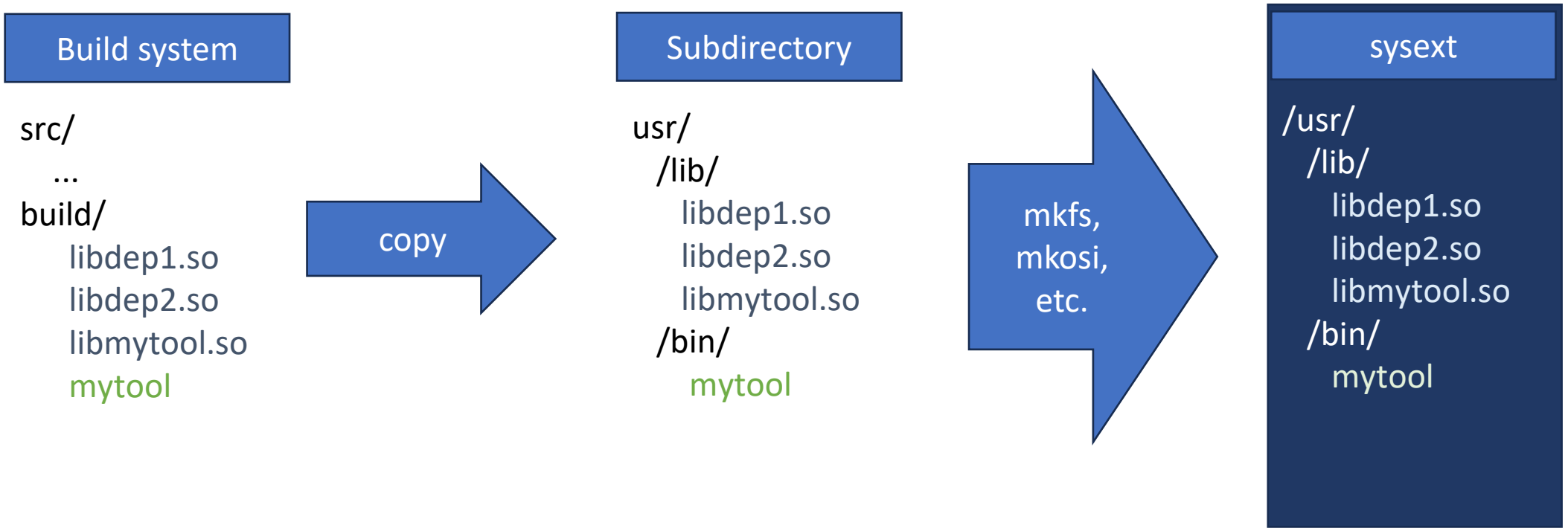
- Image-based, immutable extensions

Solid update story separate from OS updates

Using Sysexts (Merging)



Building Sysexts



Sysexts Day #2

Clean separation between OS + customisations

Solid provisioning story: supply update config via Ignition

Solid update story: update customisations independent of the OS

Straightforward packaging: build output to FS image

Easy publishing of updates: Serve via HTTPS, white-list in index file

Image composability in Flatcar

System dependent sysexts

- OEM / Vendor support (guest tools etc.)
- Optional OS features (python, zfs, podman, …)

System independent sysexsts

- Kubernetes (e.g. for ClusterAPI support), Docker compose
- Tailscale, Wasmcloud + Wasmtime
- Cri-o, k3s, Rke2 (next-gen Rancher)

Sysext Demo

Day 2 OS

- Fully automated, easy to deploy
- Reliable update / lifecycle management
- Robust and safe, proven support of your target environment
- Solid configuration management / inventory management (SBOM)
- Integrates well into operational processes

On Kubernetes, we operate workloads as interchangeable, replaceable commodities ("cattle, not pets").
Can we translate this philosophy to operating systems?

Flatcar Community

Community-driven FOSS project

No single vendor, full community stewardship

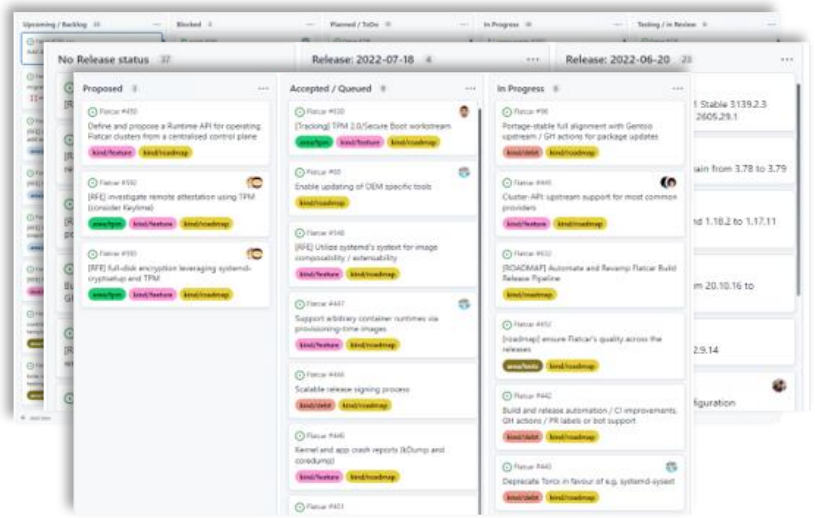
Submitted to the CNCF as incubation project (ongoing)

Matrix, Slack - Our day-to-day comms

Office hours - Every 2nd Wednesday, 2:30pm UTC

Dev Sync - Every 4th Wednesday, 2:30pm UTC

Roadmap, Implementation, Releases



Thank you

The Community's
Container Linux