Fuchsia Components and Linux Containers

By Claire Gonyeo

Who am I?

Name: Claire

Role: Software Engineer

Team: Component Framework

Company: Google

What am I talking about?

Fuchsia!

First: a story about me

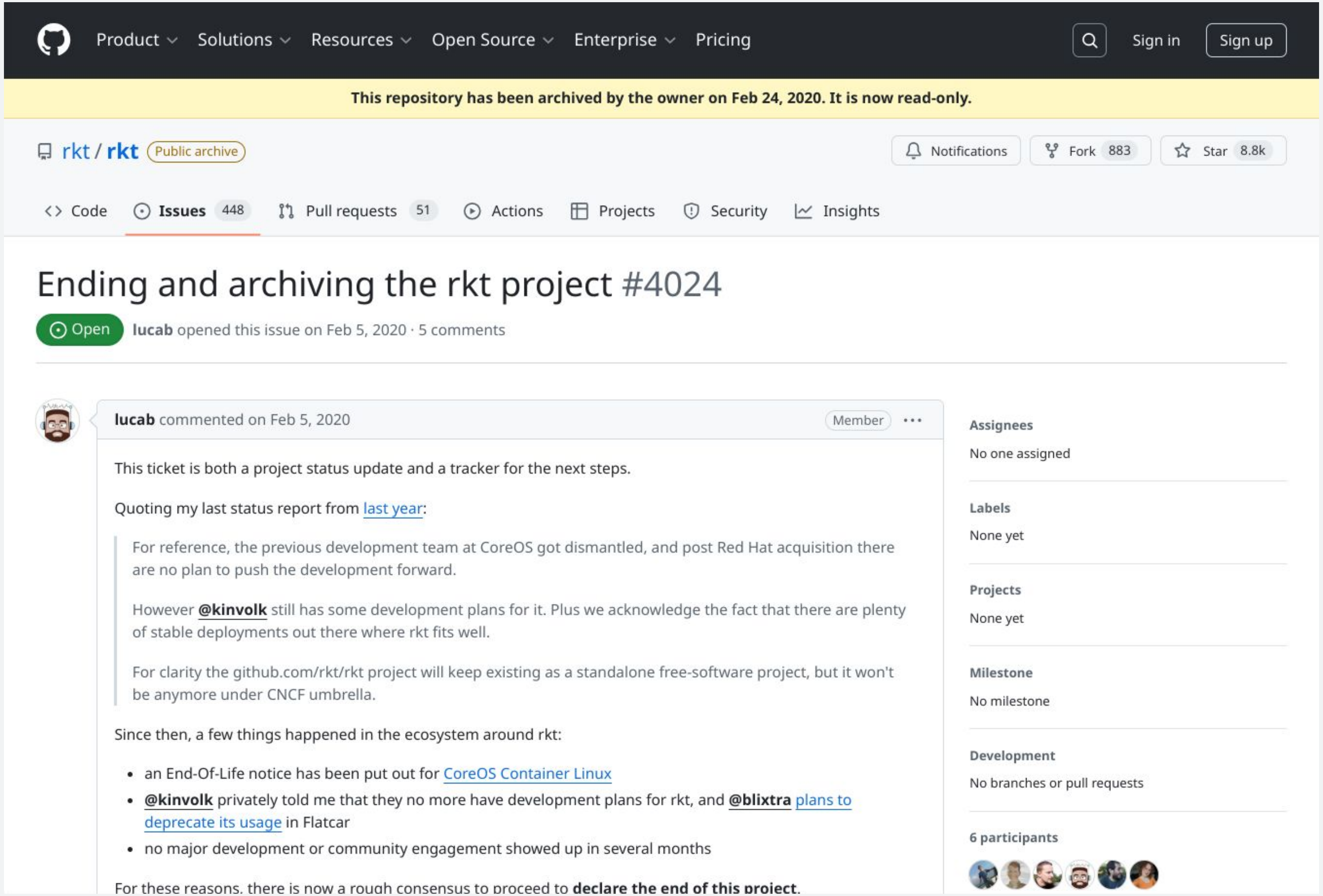
2015-2018: CoreOS

https://github.com/rkt/rkt

- I worked at CoreOS from 2015 to 2018
- I started on the rkt team
- Rkt aimed to be an alternative to Docker
- Objective was for it to be usable in the same ways as Docker

- Build software packages holding all dependencies for an executable
- Distribute software packages using The Update Framework
- Store software packages in content-addressed storage, deduplicating blobs across packages
- Reassemble content-addressed blobs into directory structure
- Launch namespaced executables with directory from last step as root

rkt didn't make it



This repository has been archived by the owner on Feb 24, 2020. It is now read-only.

Ending and archiving the rkt project #4024

lucab opened this issue on Feb 5, 2020 · 5 comments

lucab commented on Feb 5, 2020

This ticket is both a project status update and a tracker for the next steps.

Quoting my last status report from last year:

> For reference, the previous development team at CoreOS got dismantled, and post Red Hat acquisition there are no plan to push the development forward.
>
> However **@kinvolk** still has some development plans for it. Plus we acknowledge the fact that there are plenty of stable deployments out there where rkt fits well.
>
> For clarity the github.com/rkt/rkt project will keep existing as a standalone free-software project, but it won't be anymore under CNCF umbrella.

Since then, a few things happened in the ecosystem around rkt:

- an End-Of-Life notice has been put out for CoreOS Container Linux
- **@kinvolk** privately told me that they no more have development plans for rkt, and **@blixtra** plans to deprecate its usage in Flatcar
- no major development or community engagement showed up in several months

For these reasons, there is now a rough consensus to proceed to **declare the end of this project**.

Assignees	No one assigned
Labels	None yet
Projects	None yet
Milestone	No milestone
Development	No branches or pull requests

6 participants

2018: I joined Google

The Component Framework

The Component Framework

- Build software packages holding all dependencies for an executable
- Distribute software packages using The Update Framework
- Store software packages in content-addressed storage, deduplicating blobs across packages
- Reassemble content-addressed blobs into directory structure
- Launch namespaced executables with directory from last step as root

The Component Framework

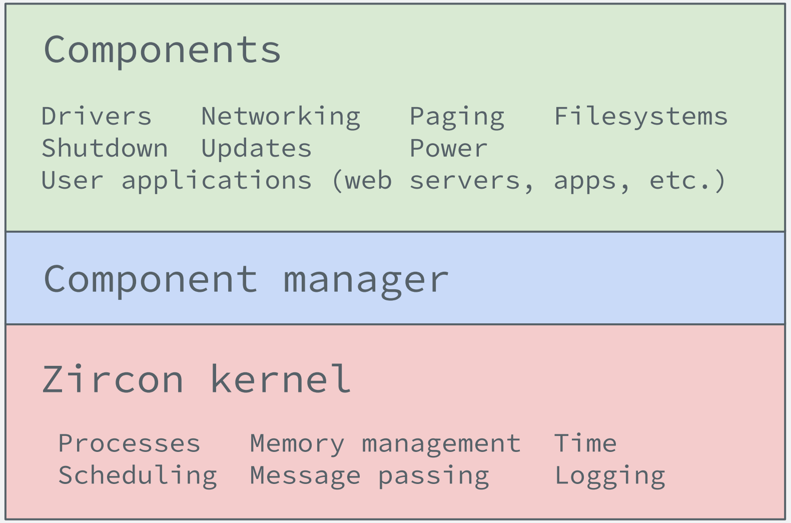
- Launch namespaced executables with directory from last step as root

Sources of differences

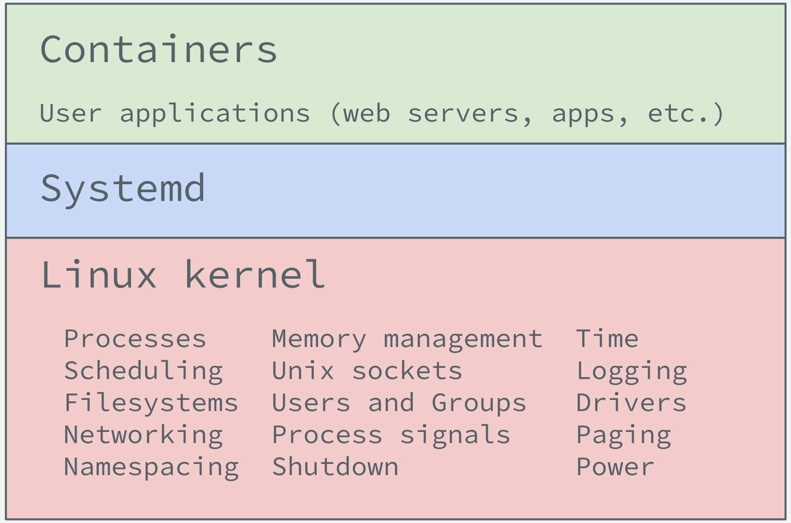
- Fuchsia is not Linux
- One vs many hosts
- Different objectives

Fuchsia != Linux

Fuchsia


Components
Drivers Networking Paging Filesystems
Shutdown Updates Power
User applications (web servers, apps, etc.)
Component manager
Zircon kernel
Processes Memory management Time
Scheduling Message passing Logging

Linux


Containers
User applications (web servers, apps, etc.)
Systemd
Linux kernel
Processes Memory management Time
Scheduling Unix sockets Logging
Filesystems Users and Groups Drivers
Networking Process signals Paging
Namespacing Shutdown Power

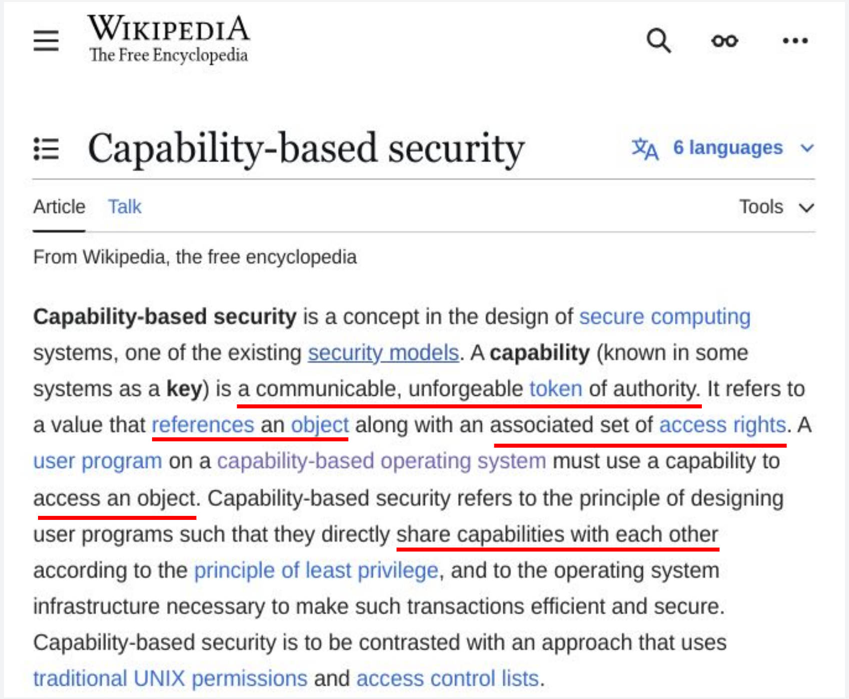
So what *does* Zircon do?

Capabilities!

Capability

An unforgeable token

- … that references an object,
- … that has access rights,
- … that can be used to access its object,
- … that can be shared with other programs.



WIKIPEDIA
The Free Encyclopedia

Capability-based security

6 languages

Article Talk Tools

From Wikipedia, the free encyclopedia

Capability-based security is a concept in the design of secure computing systems, one of the existing security models. A **capability** (known in some systems as a **key**) is a communicable, unforgeable token of authority. It refers to a value that references an object along with an associated set of access rights. A user program on a capability-based operating system must use a capability to access an object. Capability-based security refers to the principle of designing user programs such that they directly share capabilities with each other according to the principle of least privilege, and to the operating system infrastructure necessary to make such transactions efficient and secure. Capability-based security is to be contrasted with an approach that uses traditional UNIX permissions and access control lists.

Capability

An unforgeable token

… that references an object,

… that has access rights,

… that can be used to access its object,

… that can be shared with other programs.

File descriptor

An unforgeable token

… that references an object,

… that has access rights,

… that can be used to access its object,

… that can be shared with other programs.

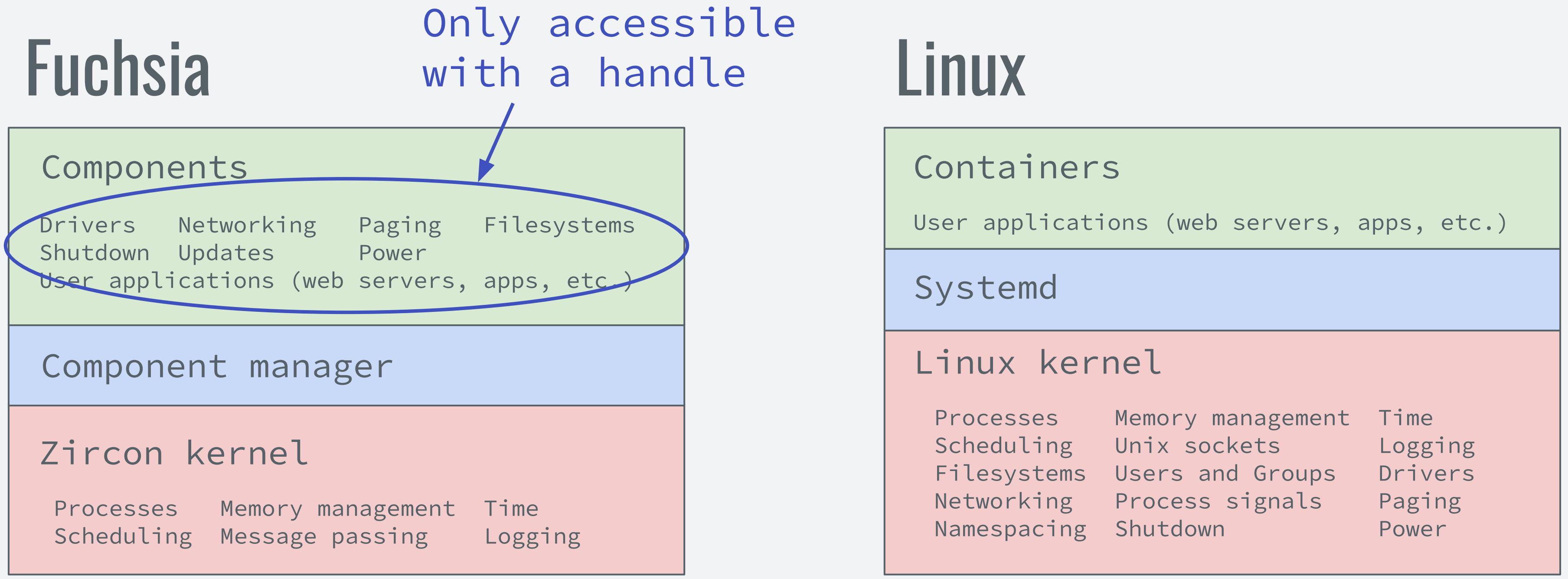
What if a process could *only* use file descriptors?

🙅‍♀️

```
int hello_fd = open("hello.txt", O_RDONLY);
```

👍

```
int hello_fd = openat(root_fd, "hello.txt", O_RDONLY);
```


Fuchsia
Only accessible with a handle
Components
Drivers Networking Paging Filesystems
Shutdown Updates Power
User applications (web servers, apps, etc.)
Component manager
Zircon kernel
Processes Memory management Time
Scheduling Message passing Logging
Linux
Containers
User applications (web servers, apps, etc.)
Systemd
Linux kernel
Processes Memory management Time
Scheduling Unix sockets Logging
Filesystems Users and Groups Drivers
Networking Process signals Paging
Namespacing Shutdown Power

A default component

Has:

- A handle to its own process
- A handle to its own job
- A handle to its package directory

Does not have:

- Access to mutable storage
- Access to the network
- The ability to launch other processes
- The ability to emit logs
- The ability to interact with other components (aside from its package provider)

Most Component configuration knobs are about capability handles

Component manifest

```
{
    program: {
        runner: "elf",
        binary: "bin/app",
    },
    use: [ {
        protocol: "fuchsia.logger.LogSink",
    } ],
    capabilities: [ {
        protocol: "fuchsia.examples.Echo",
    } ],
    expose: [ {
        protocol: "fuchsia.examples.Echo",
        from: "self",
    } ],
}
```

Dockerfile

```
# syntax=docker/dockerfile:1

FROM node:lts-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

How to build application

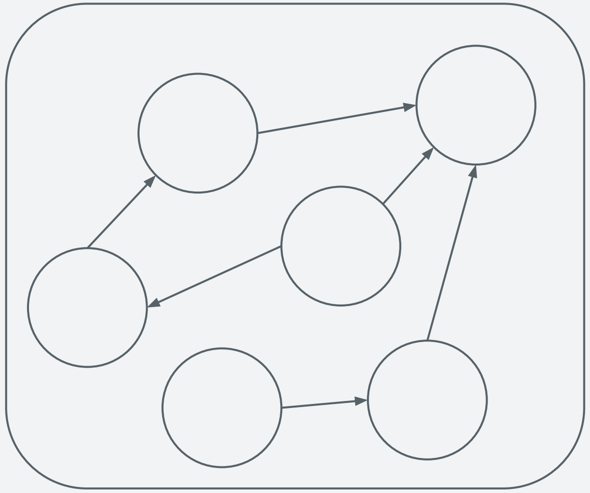
How to run application

Connects to log server

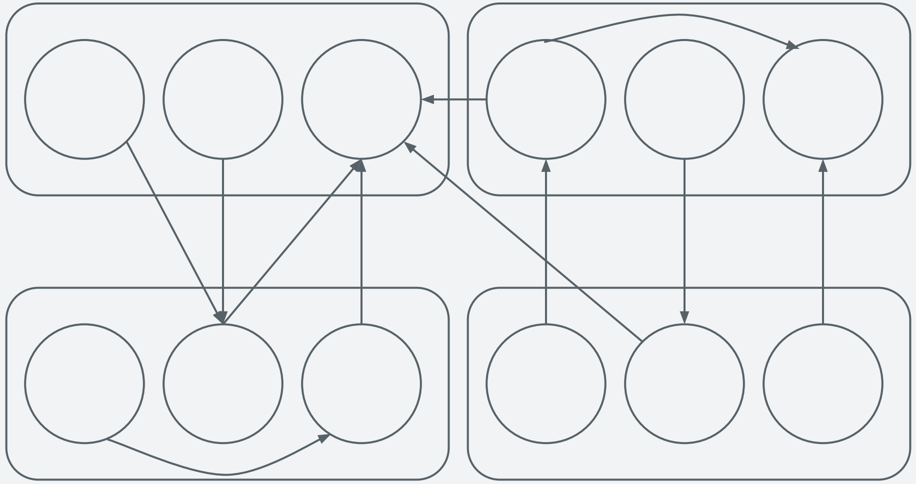
Provides echo IPC server

One host or many?

Components

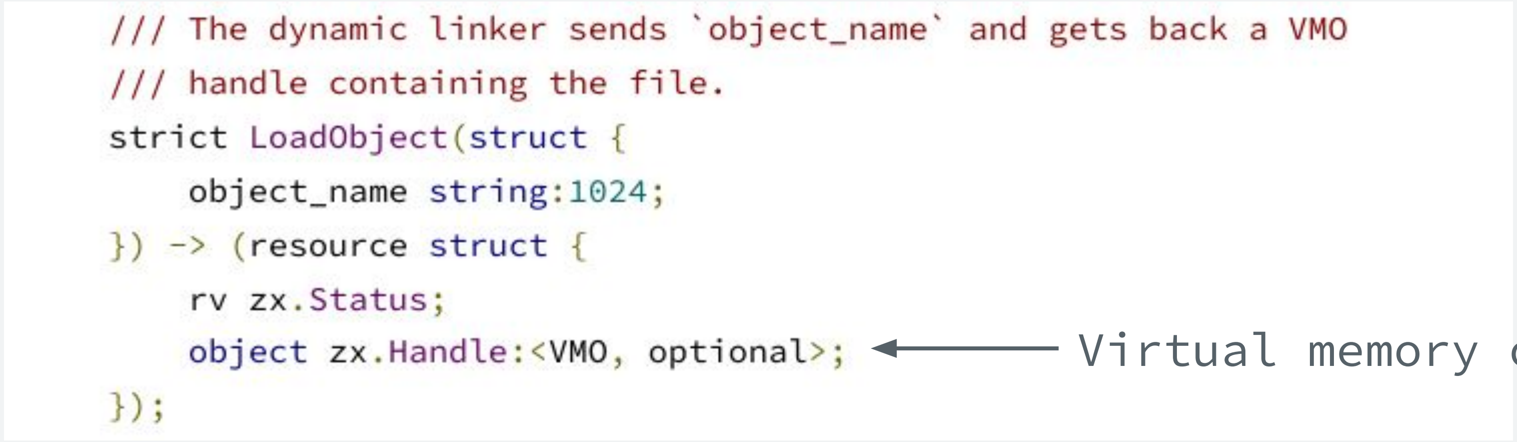


Containers



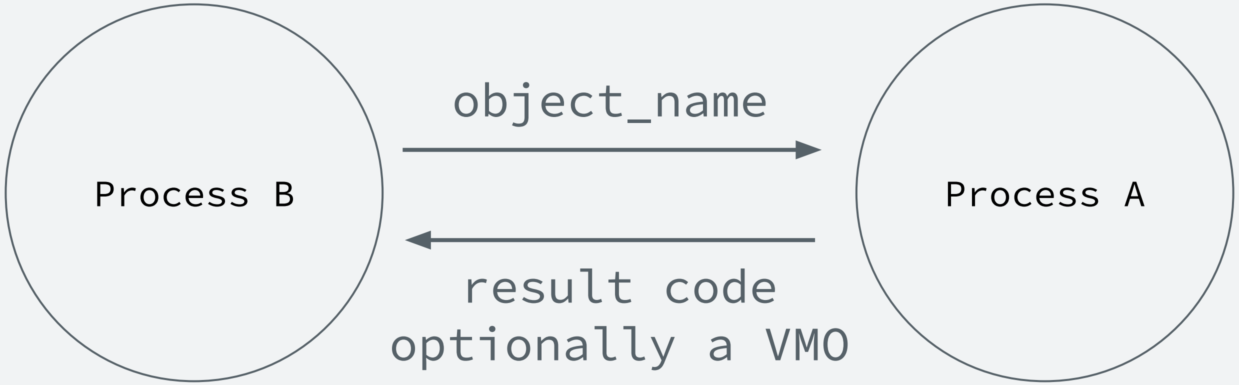
FIDL

Fuchsia Interface Definition Language



```
/// The dynamic linker sends `object_name` and gets back a VMO
/// handle containing the file.
strict LoadObject(struct {
    object_name string:1024;
}) -> (resource struct {
    rv zx.Status;
    object zx.Handle:<VMO, optional>;
});
```

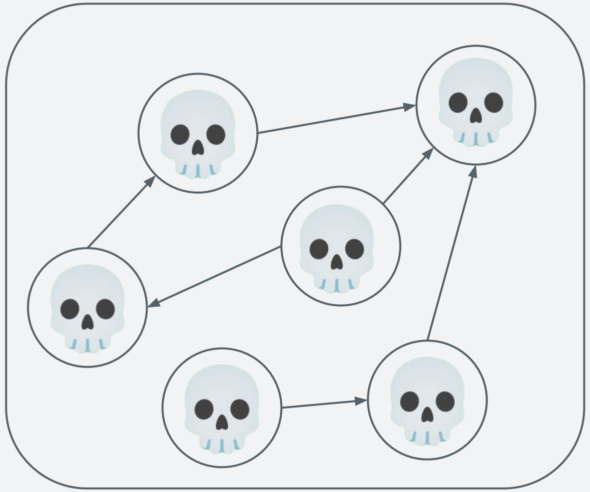
← Virtual memory object



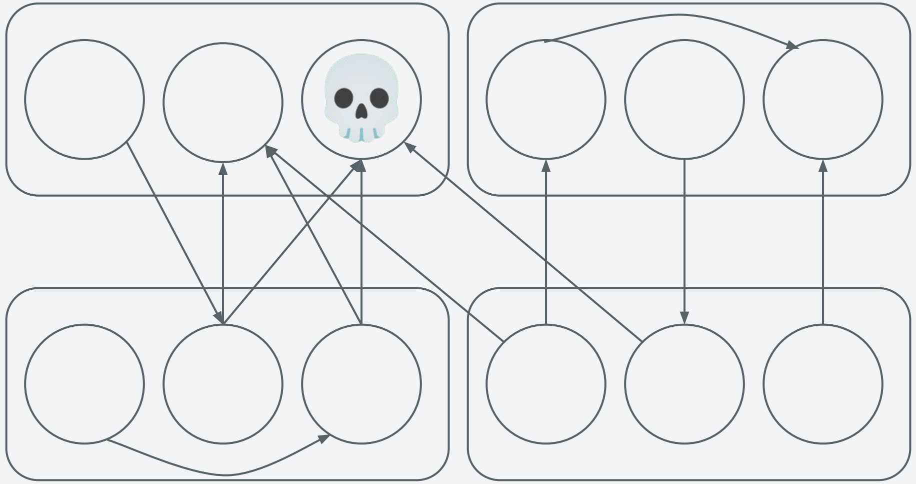
Because components are all on the same machine, they can rely on sharing machine-local resources

Fault tolerance

Components



Containers



Different goals mean different solutions

Package size

Minimal

Manifest
Executable
Required libraries

Convenient

Manifest
Executable
Required libraries
Other libraries
Package manager
Shell + tools

Deployment ease

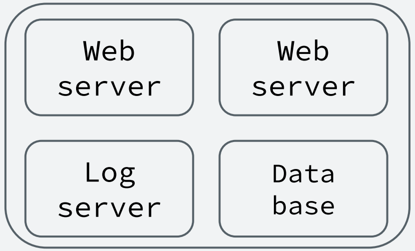
Set of components
to run mostly fixed
at OS build time

docker run ...

Deployment objective

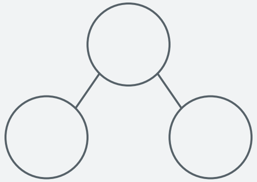
Single consumer device

Heterogenous workload

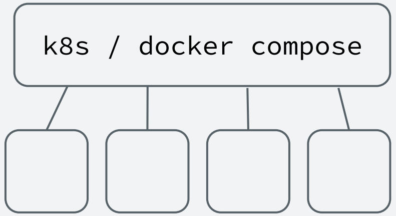


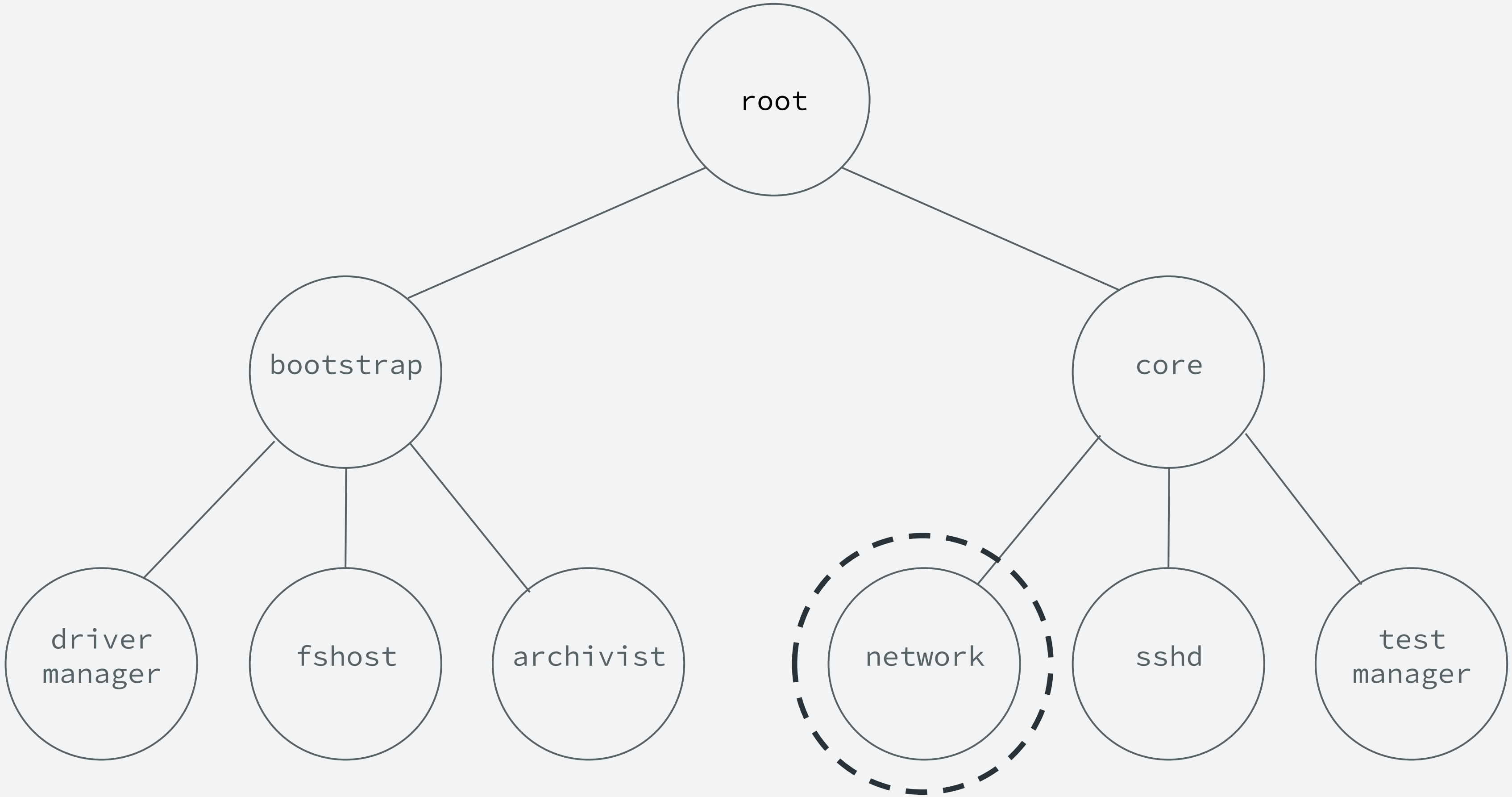
Cross-sandbox orchestration

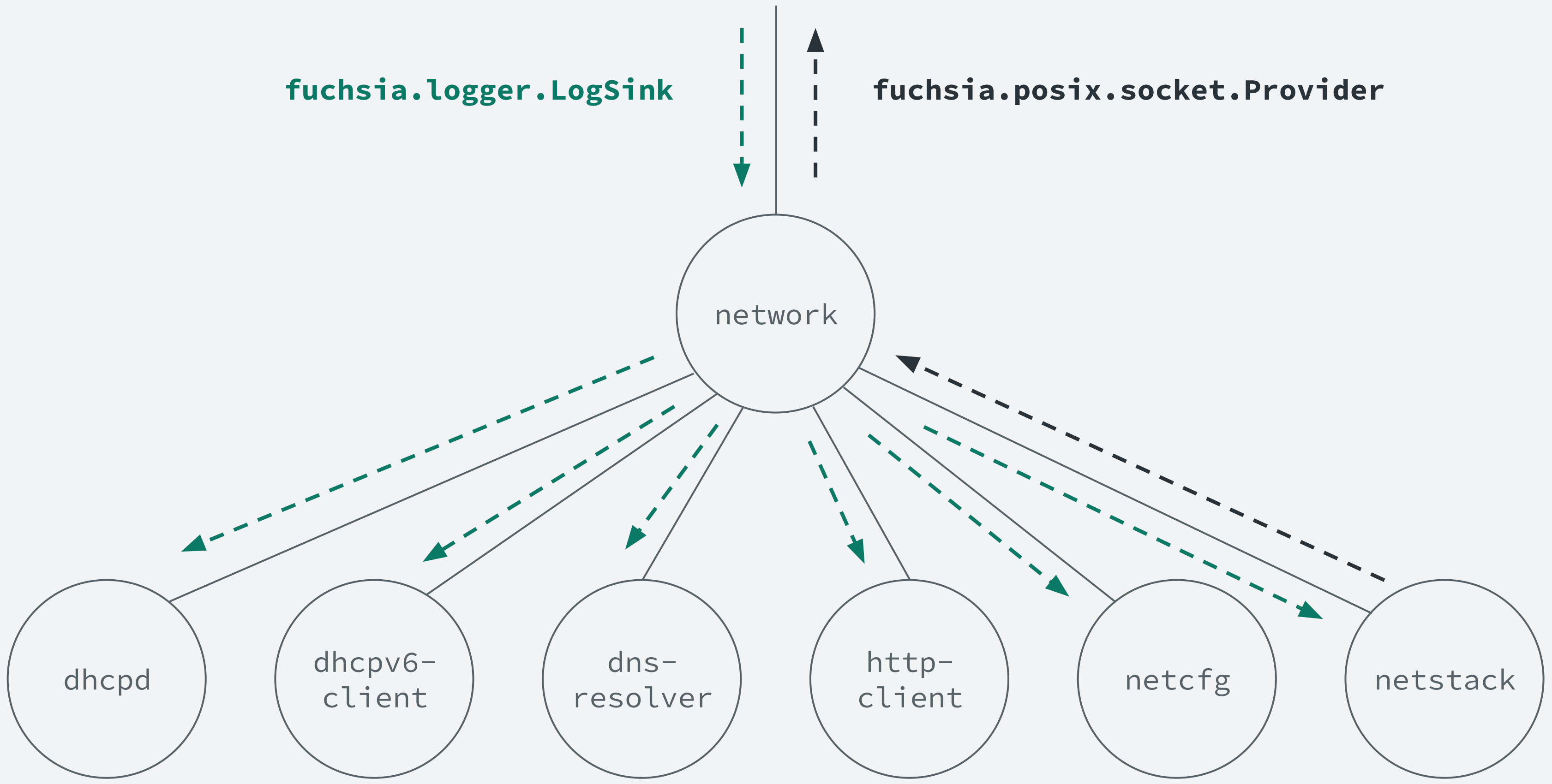
Part of component framework

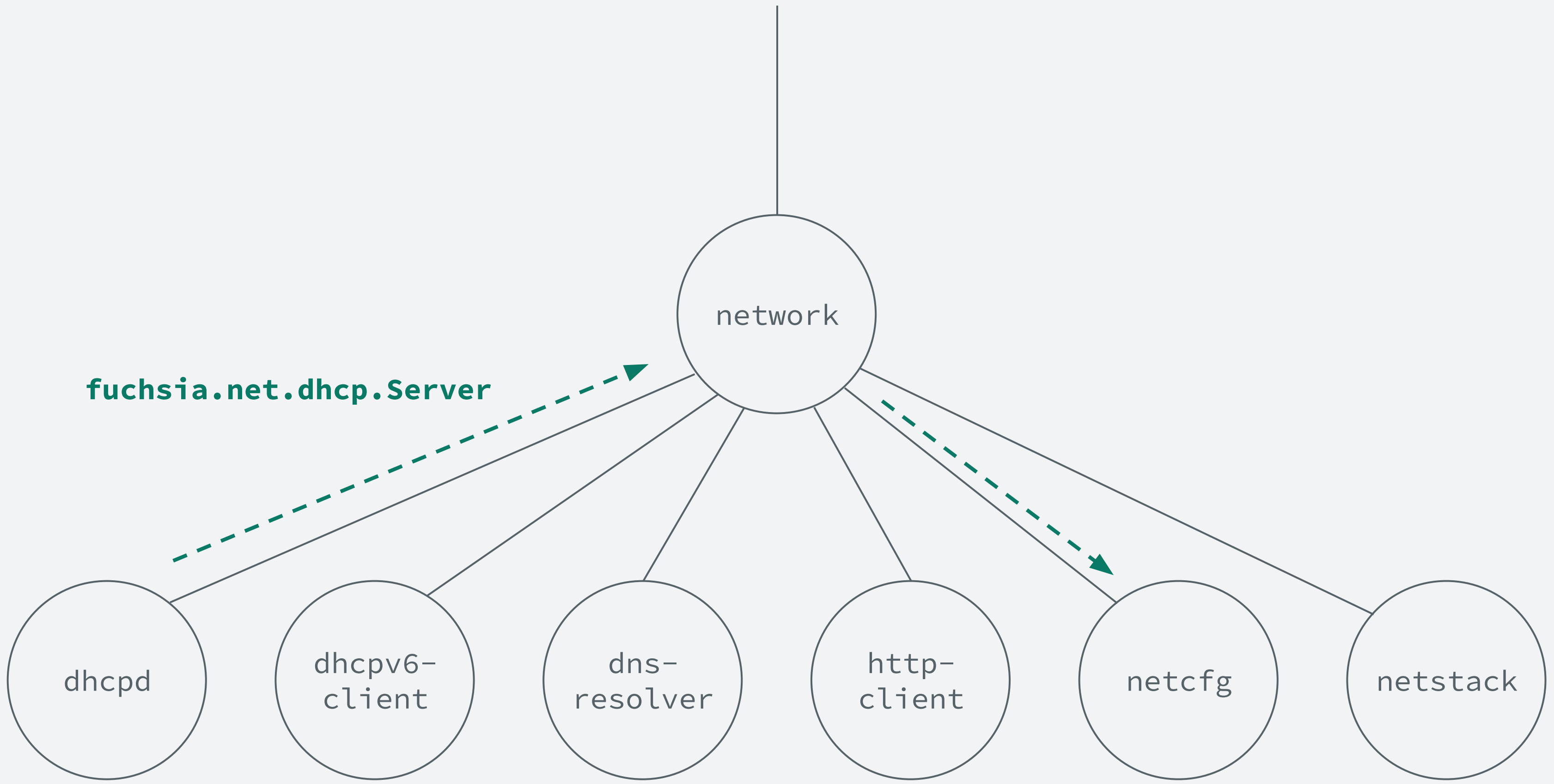


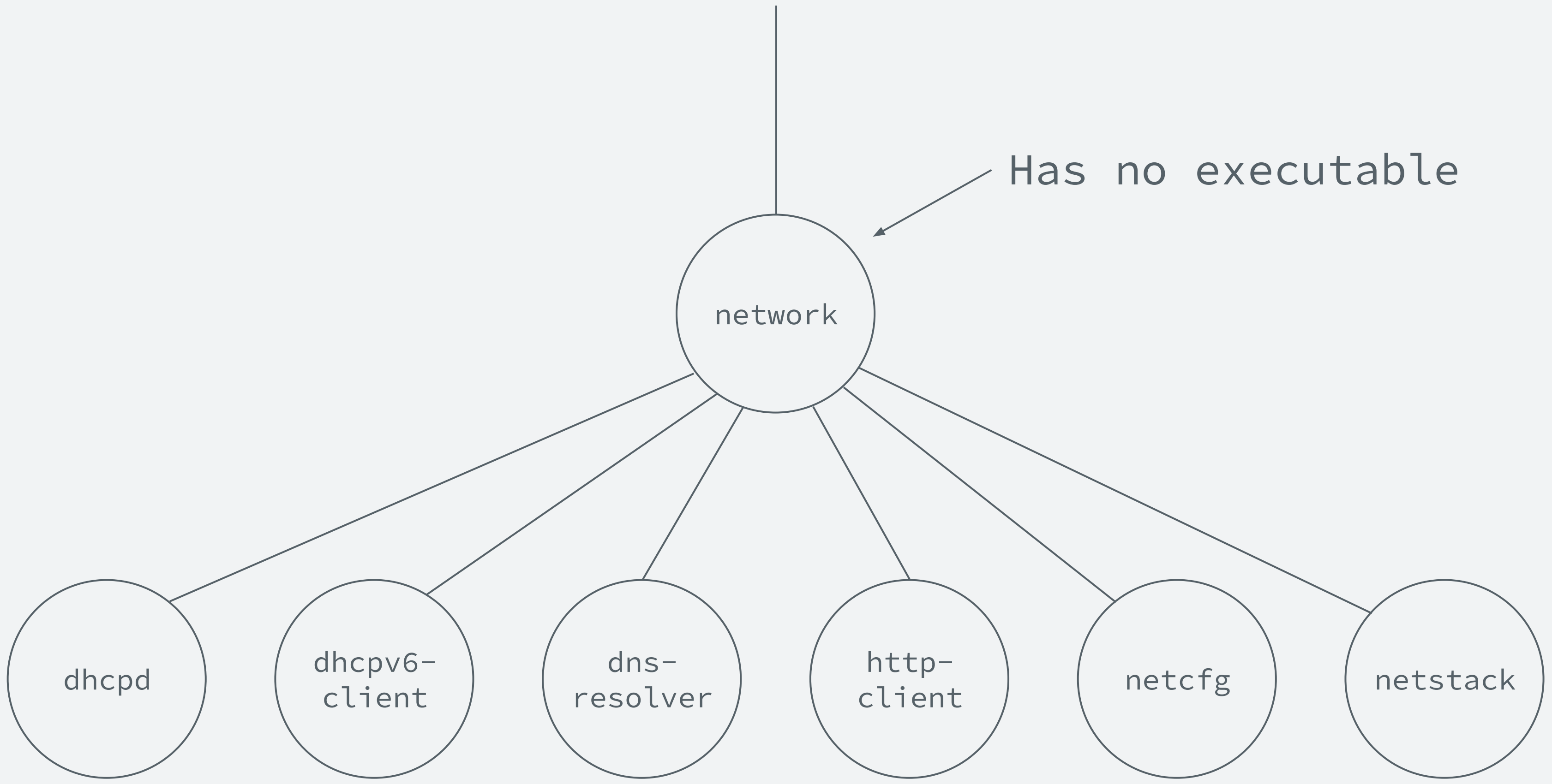
Handled at higher layer

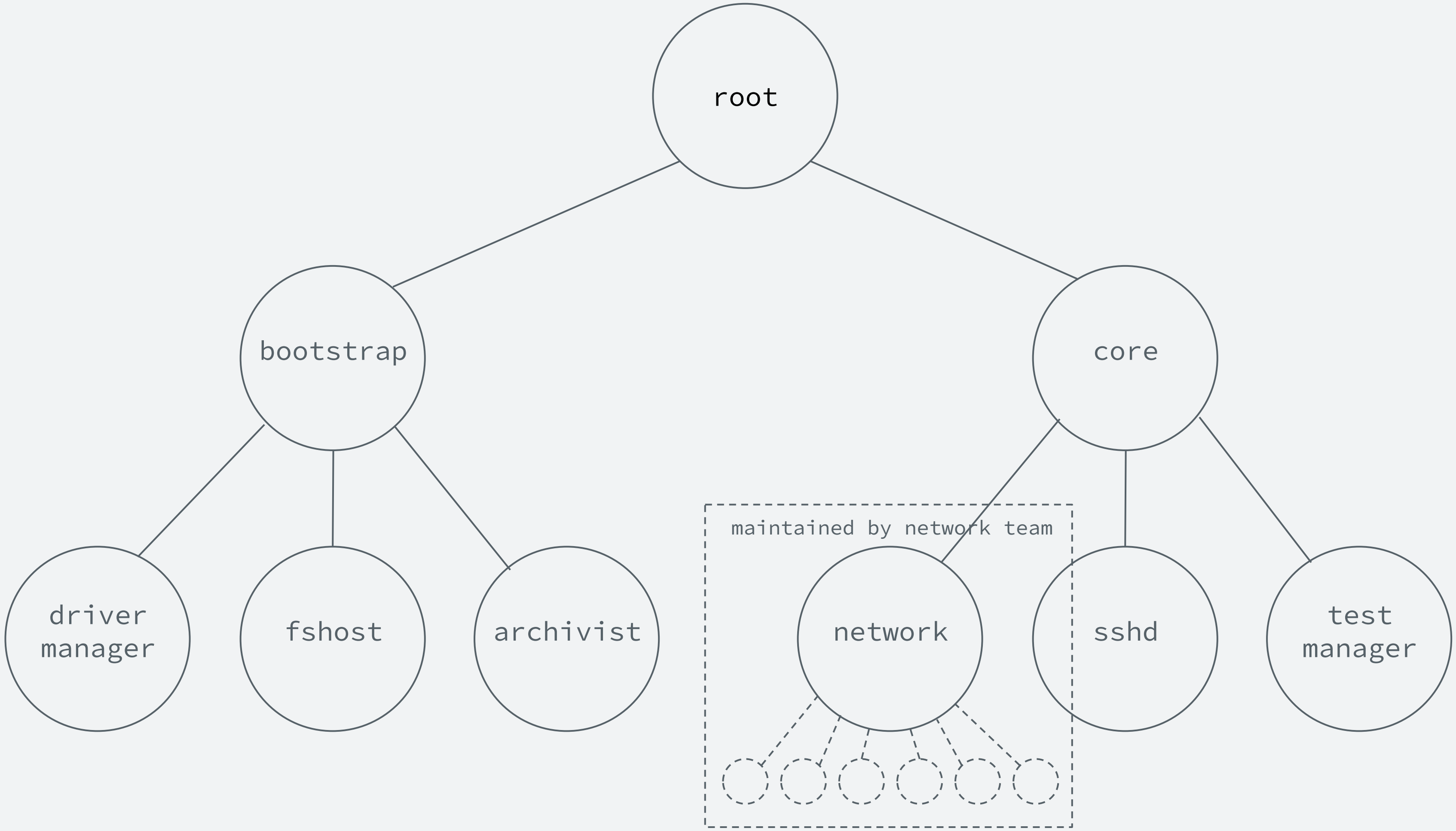



root
bootstrap
core
driver manager
fshost
archivist
network
sshd
test manager


fuchsia.logger.LogSink
fuchsia.posix.socket.Provider
network
dhcpd
dhcpv6-client
dns-resolver
http-client
netcfg
netstack


network
fuchsia.net.dhcp.Server
dhcpd
dhcpv6-client
dns-resolver
http-client
netcfg
netstack


Has no executable
network
dhcpd
dhcpv6-client
dns-resolver
http-client
netcfg
netstack


root
bootstrap
core
driver manager
fshost
archivist
maintained by network team
network
sshd
test manager

Capability-centric design

Standardized IPC system

Single machine scope

Tree of sandboxes

Model powers low-level OS features

More detailed inputs/outputs from sandbox

Weaker inter-sandbox fault tolerance

Configuration and building in separate files

Sandboxes can encapsulate other sandboxes

Thank you!

https://fuchsia.dev