FLOX

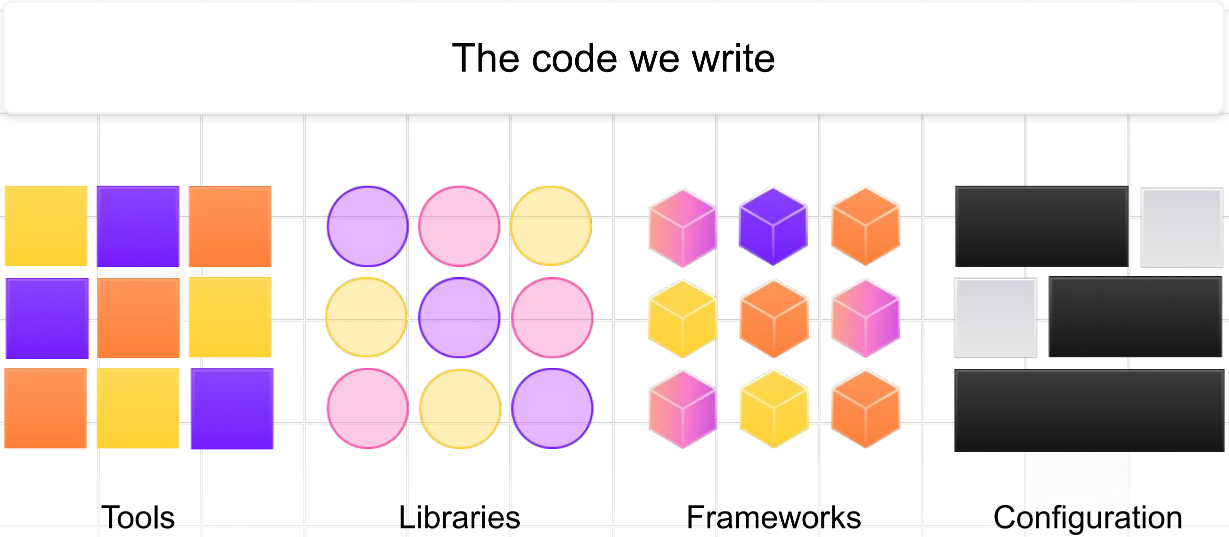
Reproducible dev environments w/Flox

Ross Turk

flox.dev

A few abstract thoughts about software environments…

Software is built on top of a lot of stuff



The code we write

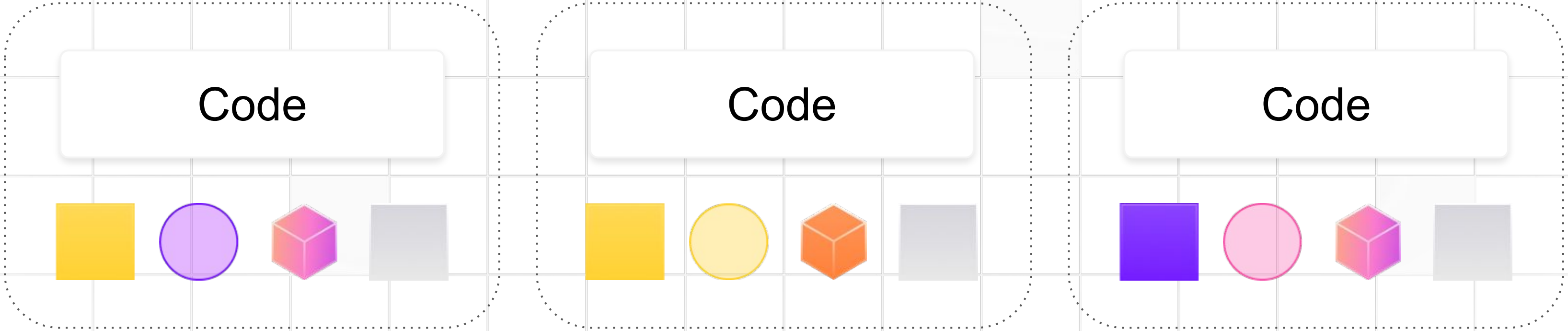
Tools

Libraries

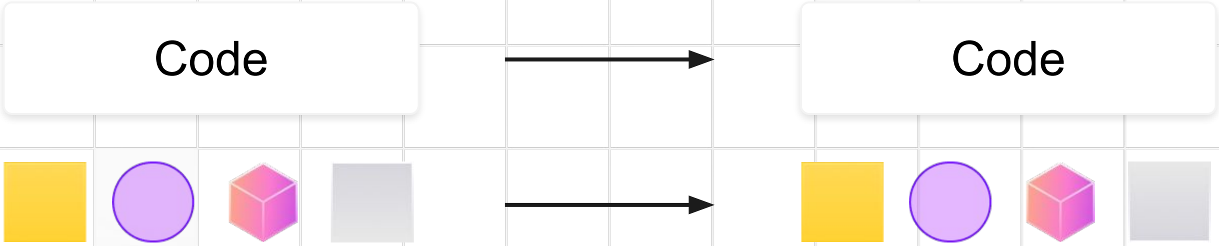
Frameworks

Configuration

Each project has its own annoying collection of stuff



Sharing these environments can be hard



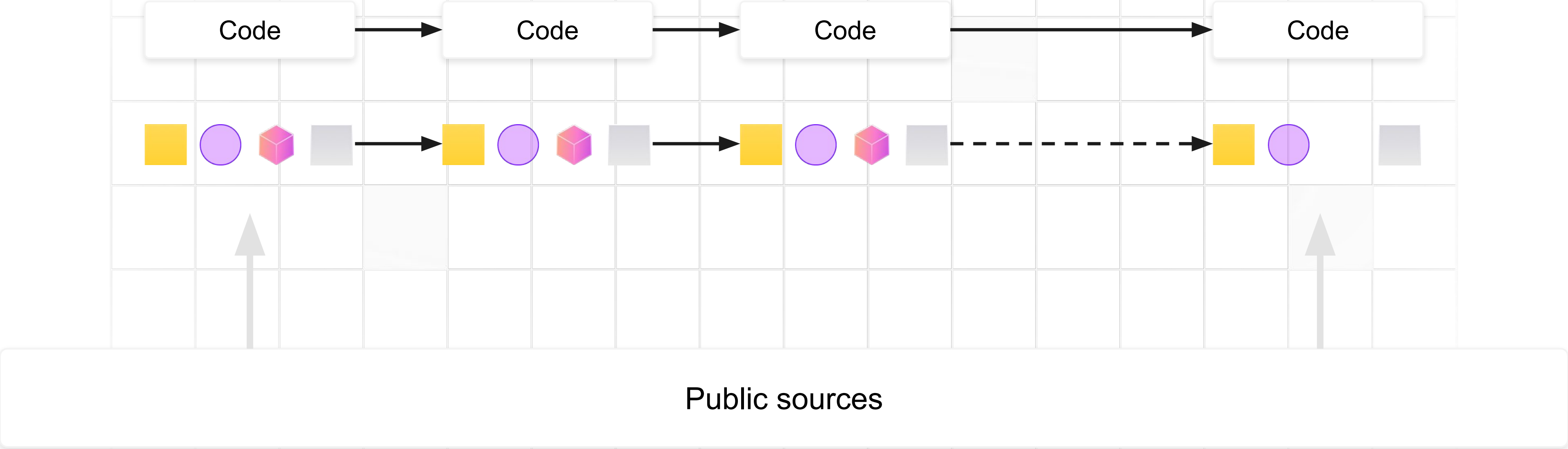
Code

Code

Your system

Your friend’s system

Recreating the past is even harder

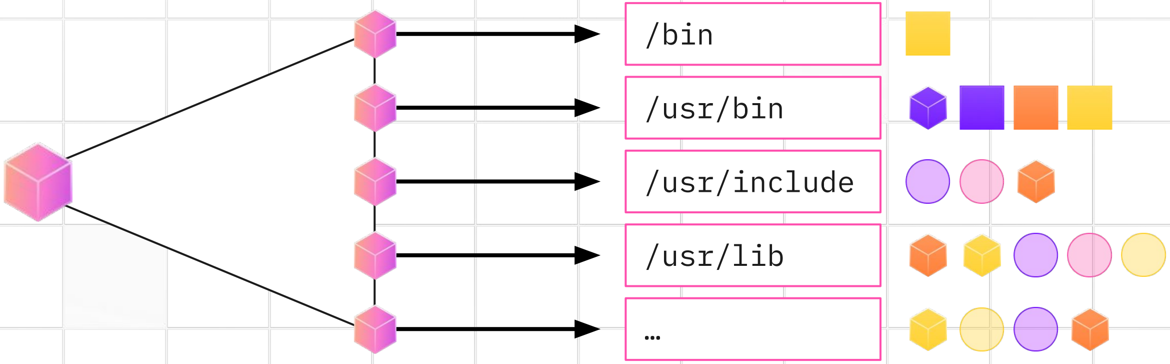


FLOX

Reproducible, consistent environments that span platforms, projects & time

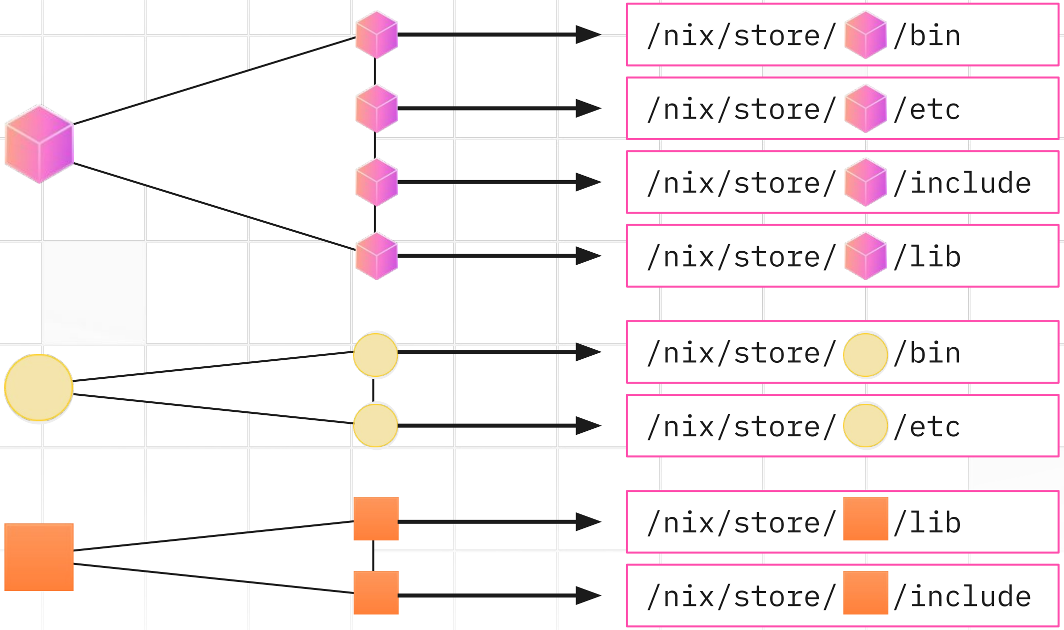
Okay, now a few quick thoughts about package management

Most package managers operate at the system level



which means that stuff is either “installed” or it’s not,
and uninstalling means **finding** and **removing** stuff.

Nix places packages into a special store



/nix/store/⬢/bin

/nix/store/⬢/etc

/nix/store/⬢/include

/nix/store/⬢/lib

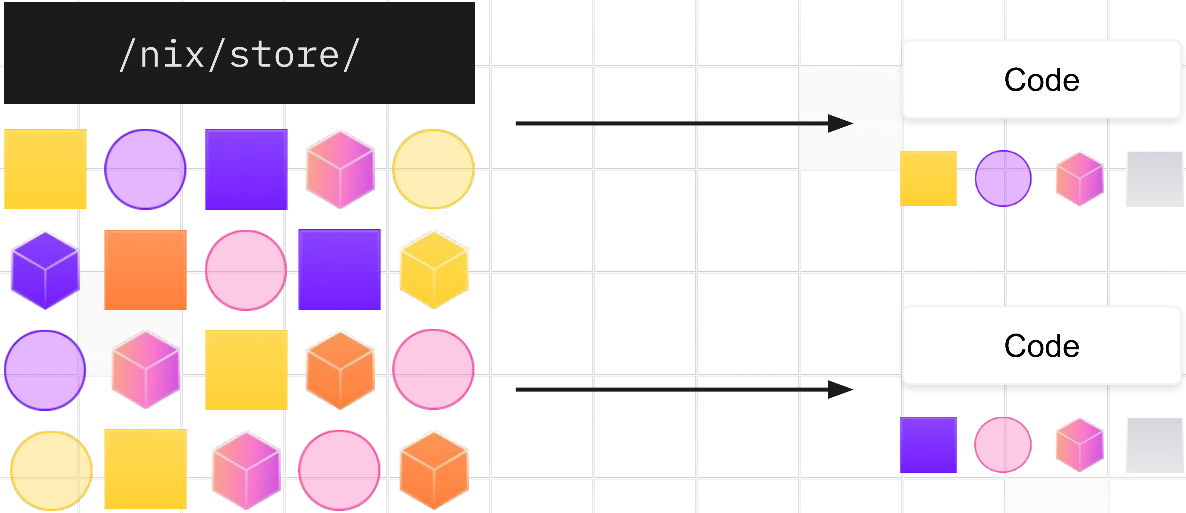
/nix/store/●/bin

/nix/store/●/etc

/nix/store/■/lib

/nix/store/■/include

And weaves them together dynamically



using a mad clever web of symlinks, hook scripts, and environment variables.

Nix does more than just manage packages

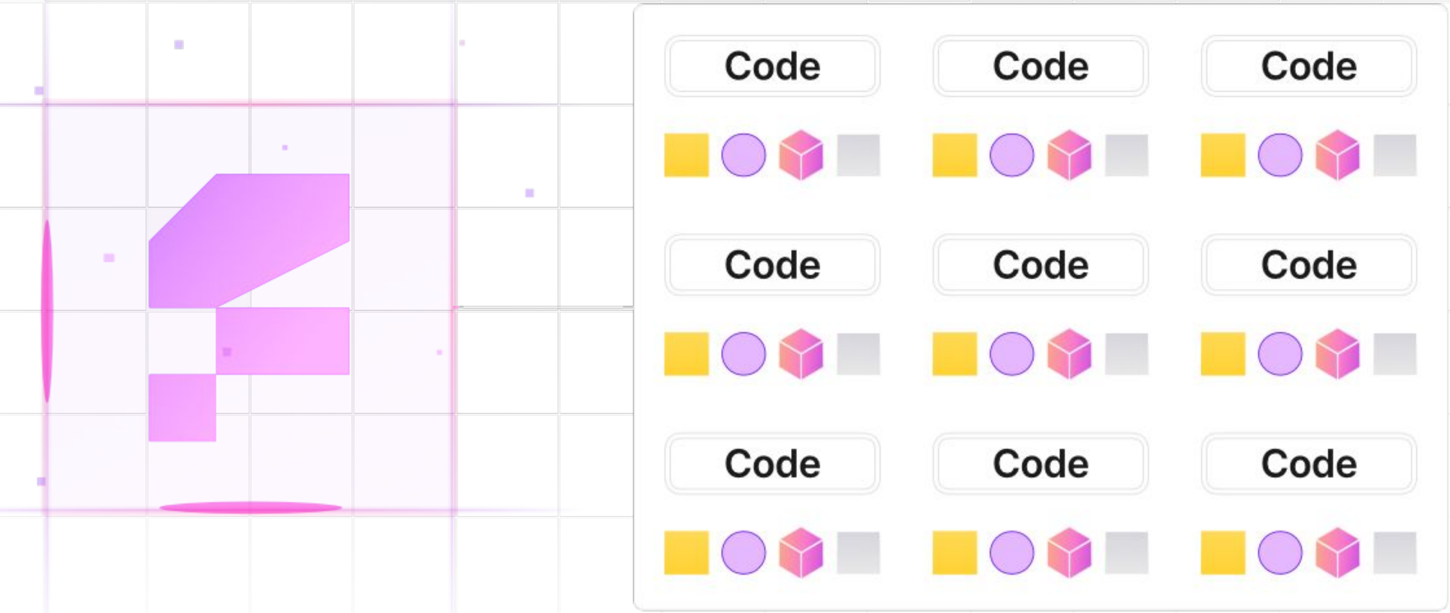
- Builds software deterministically using a declarative language

- Allows for sophisticated overrides and complex integrations

...it also acts *nothing like* a traditional package manager and it has a steep learning curve.

What is Flox?

Flox makes it easy to build virtual environments



FLOX

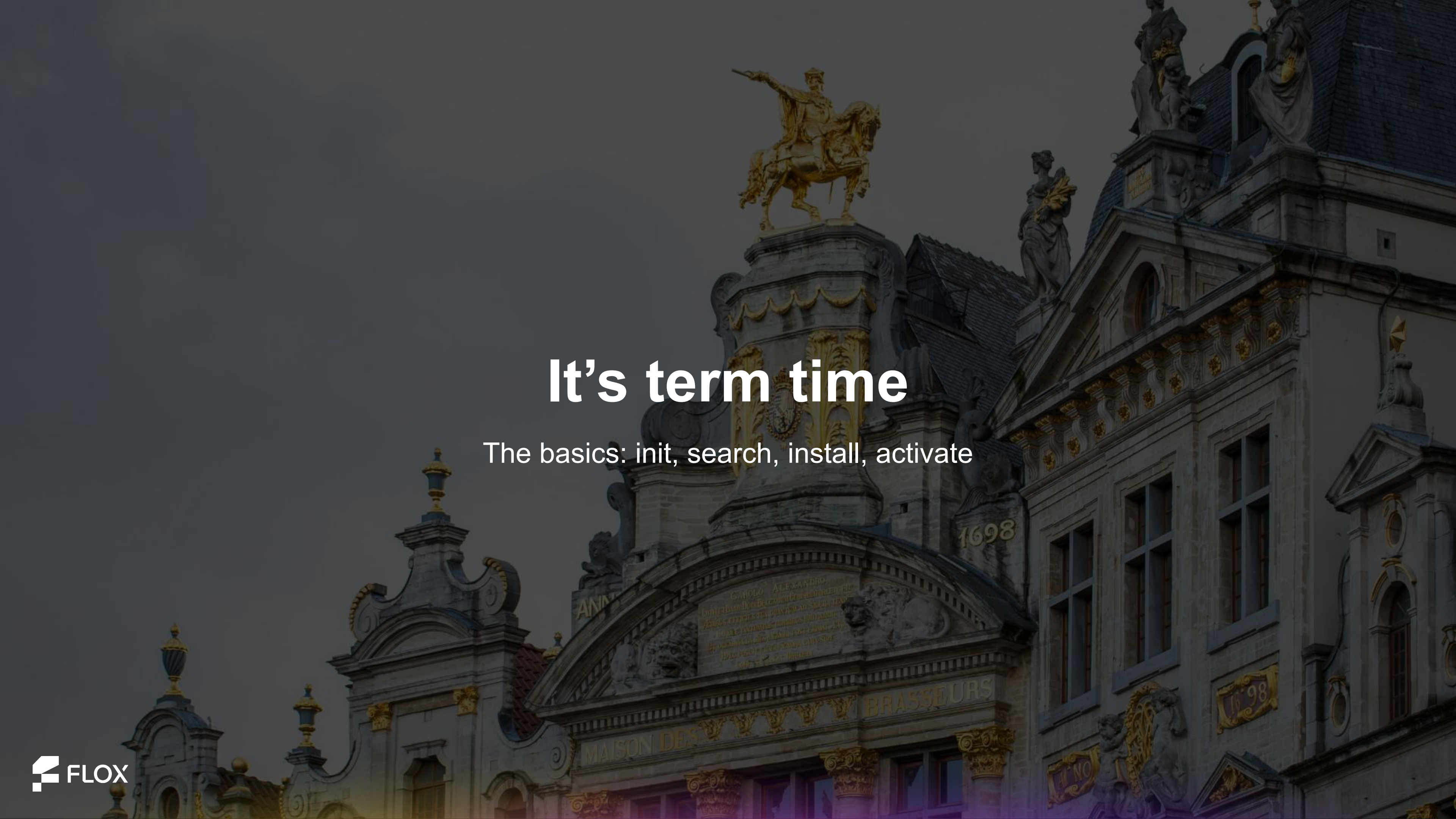
Flox still behaves a lot like a package manager, but adds a few new subcommands.

```
+flox init

 flox search
 flox install
 flox uninstall

+flox activate

+flox push
+flox pull
```



It’s term time

The basics: init, search, install, activate

What else can it do?

Three ways to use Flox environments

```
# Manage alongside code
cd myproject
flox init
flox install nodejs
git add .flox

# FloxHub remote activation
mkdir funtools
flox init
flox install lolcat charasay
flox push
ssh me@remote
flox activate -r [username]/funtools

# Default environment
cd ~
flox init
flox install inetutils bat
echo eval '"$(flox activate)"' >> .zshrc
```

What’s in the manifest?

- Packages
- Environment variables
- Shell hooks
- Supported architectures

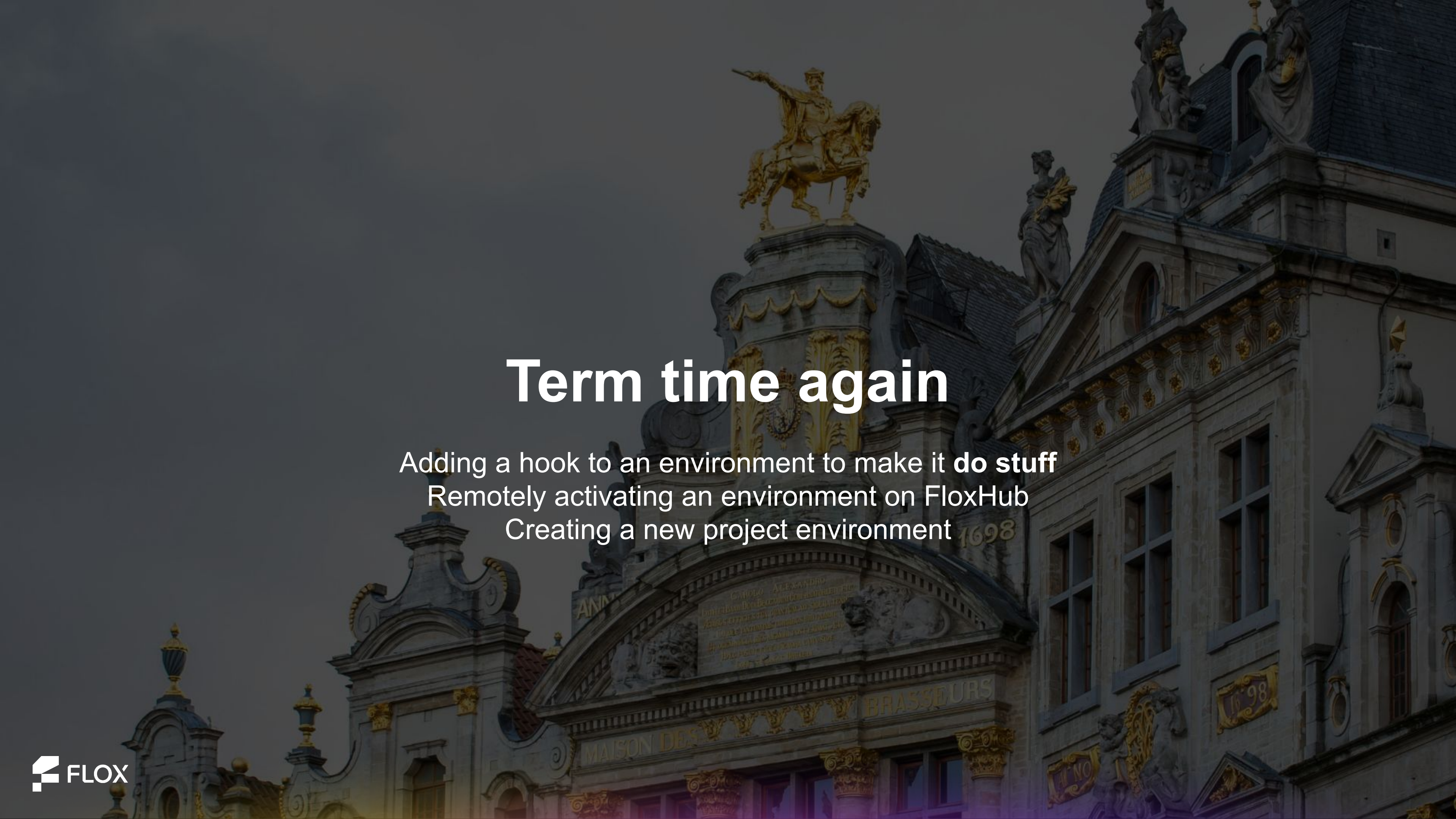
What’s in the manifest?

```toml
[install]
podman.pkg-path = "podman"
buildah.pkg-path = "buildah"
qemu = { systems = ["aarch64-darwin"], pkg-path = "qemu" }

[vars]
BUILDAH_CPPFLAGS="-DDEBUG"

[hook]
script = """
  if [[ $(uname -m) == 'arm64' ]]; then
    podman machine start
  fi
"""

[options]
systems = [
  "x86_64-linux",
  "aarch64-darwin"
]
```

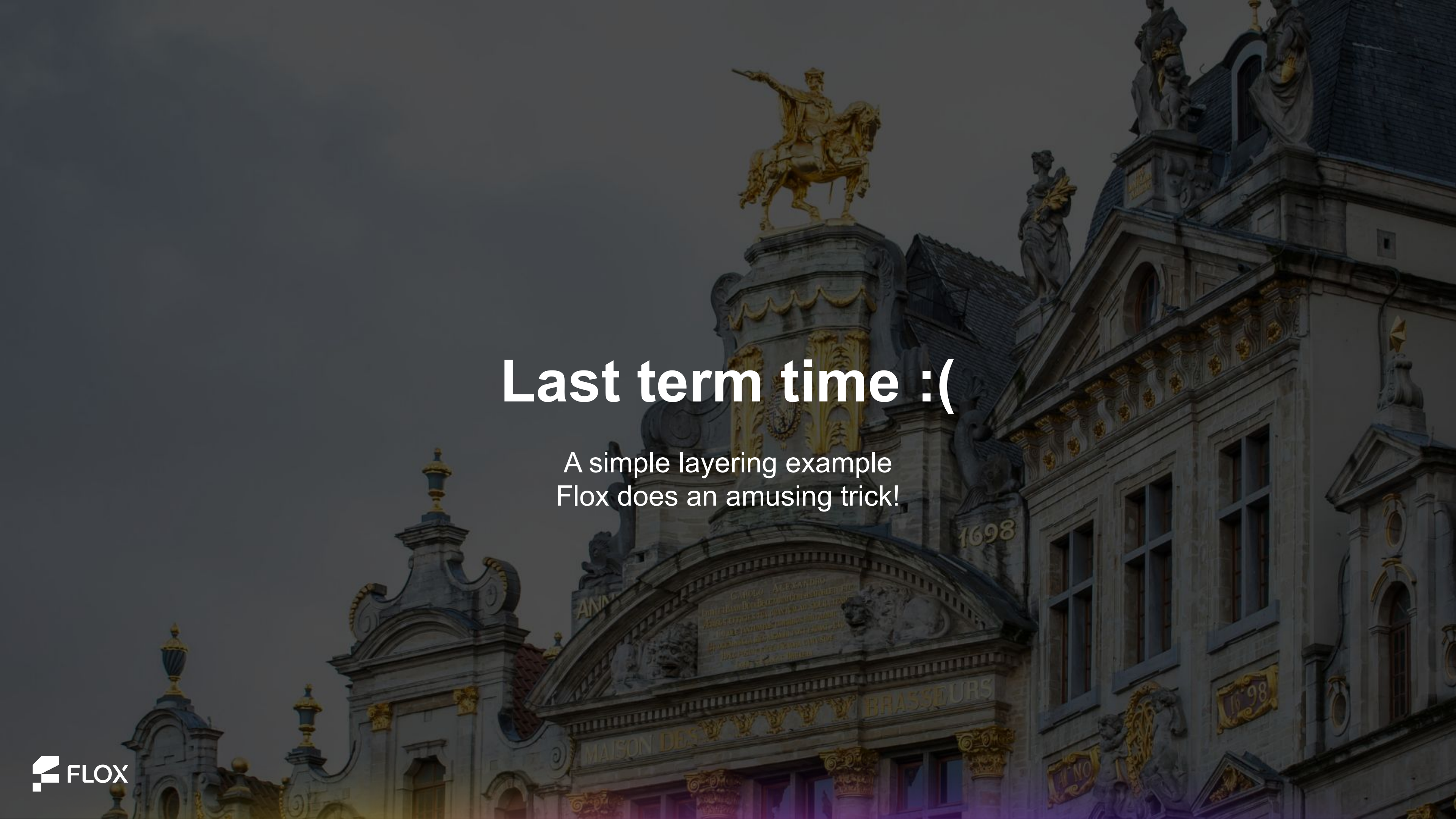


Term time again

Adding a hook to an environment to make it **do stuff**
Remotely activating an environment on FloxHub
Creating a new project environment

A word on isolation and layering

FLOX



Last term time :(

A simple layering example
Flox does an amusing trick!

What’s next for Flox?

What’s next for Flox?

- More robust public catalog: historical versions & curation
- Private catalogs
- “Escape hatches” Nix devs can use to build fancy environments

FLOX

flox.dev