Mud, energy, and containers

Gratitude

Thank you for listening, asking questions,
and helping make this conversation better!

Outline

1. 🟤 Mud
2. ⚡ Energy
3. 🖼️ Scenario

We need to talk about mud

Mud can be friend or foe.

Image: Ambroży Sabatowski, Mud in the forest, 1922 (Wikimedia Commons)

Mud as friend

- Adobe (Spanish for *mudbrick*) is an over 7,000 year old technology.
- Innovations like Superadobe continue to enhance the use of mud.
- *Mud Futures* demos 3D printed mud structures (History Colorado Center, Denver)

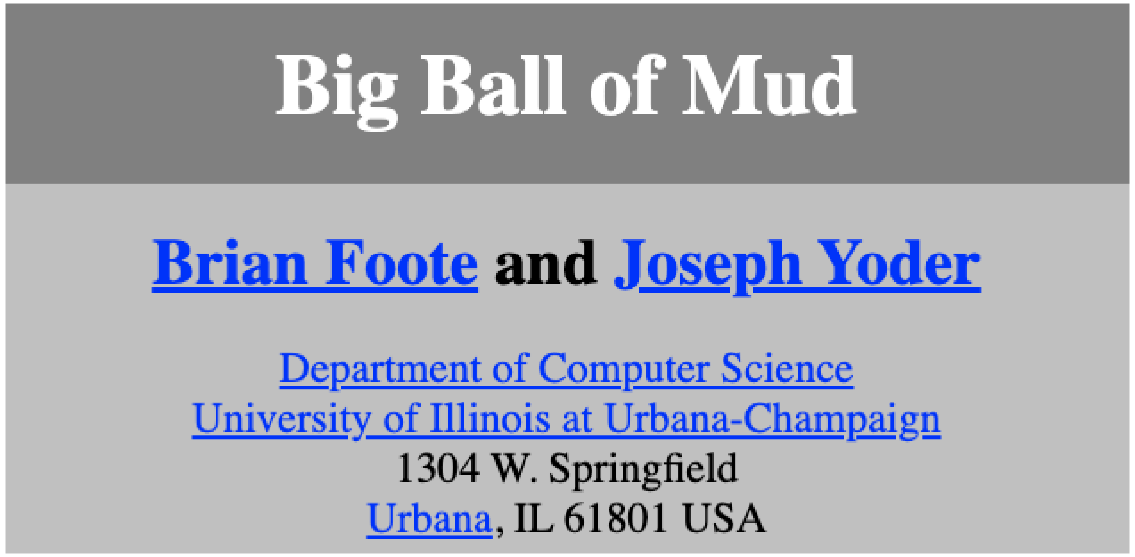
Image: DVIDSHUB, 2012 (Wikimedia Commons)

Mud as foe

Mud can also cause damage if left untended.

Image: King of Hearts, 2023 (Wikimedia Commons)

Software mud



Big Ball of Mud

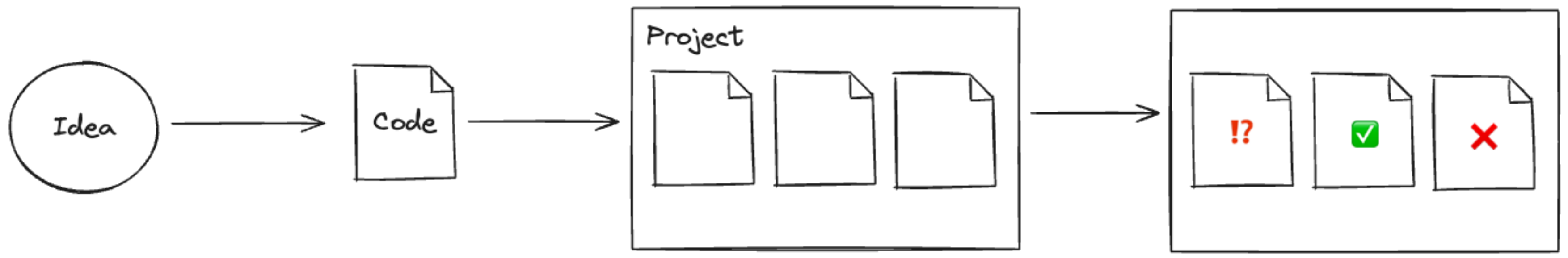
Brian Foote and Joseph Yoder

Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield
Urbana, IL 61801 USA

Foote, Brian, and Joseph Yoder. "Big ball of mud." *Pattern languages of program design 4* (1997)

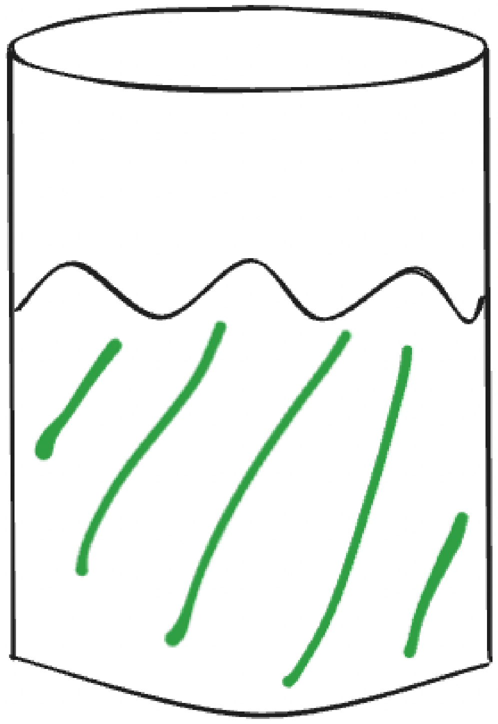
"A BIG BALL OF MUD is haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle."

Software mud as necessary



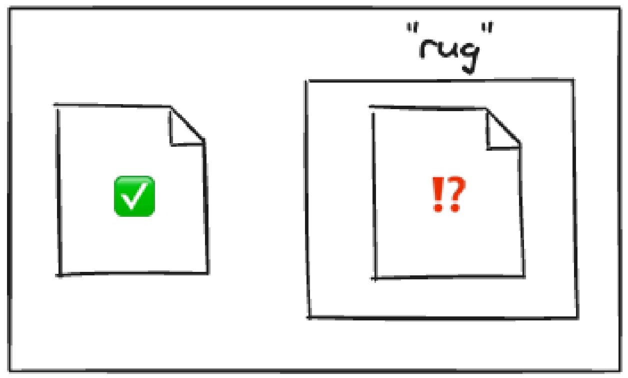
Software projects often move from ***“throwaway code”*** (little balls of mud) to ***“piecemeal growth”*** and ***“keep it working”*** which can emerge as ***“big balls of mud”*** along the journey.

Software mud as necessary



A perspective: building a vessel for the mud.

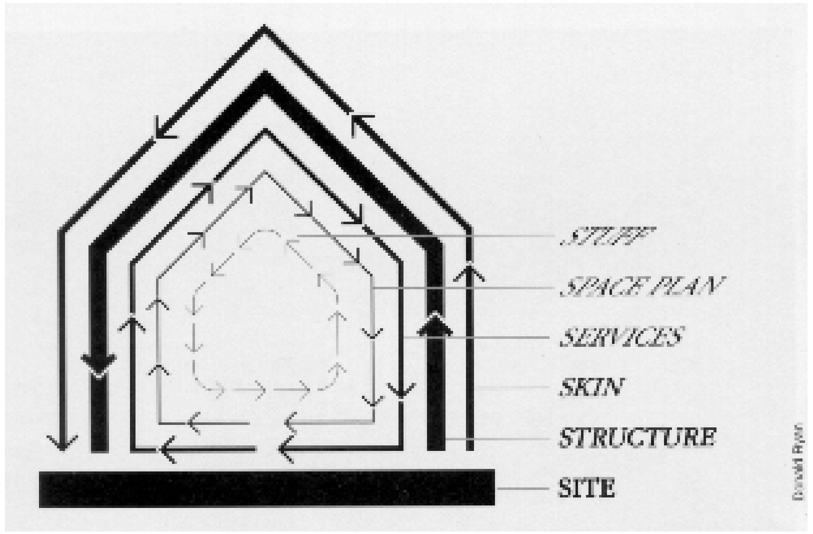
Defining a space for challenges



Sometimes we need to ***"sweep it under the rug"***.

"Overgrown, tangled, haphazard spaghetti code is hard to comprehend, repair, or extend, and tends to grow even worse if it is not somehow brought under control."

Defining a space for challenges



Software exhibits ***"shearing layers"***. *"Our basic argument is that there isn't any such thing as a building. A building properly conceived is several layers of longevity of built components"* (How Buildings Learn, Brand 1994).

Organize artifacts together by rate of change.

Software design as continuous

> *“Things that are good have a certain kind of structure. You can’t get that structure except dynamically. Period.* ***In nature you’ve got continuous very-small-feedback-loop adaptation going on, which is why things get to be harmonious****. That’s why they have the qualities we value. If it wasn’t for the time dimension, it wouldn’t happen.”*
>
> \- Christopher Alexander (Brand, Stewart. *How buildings learn: What happens after they’re built.* 1995)

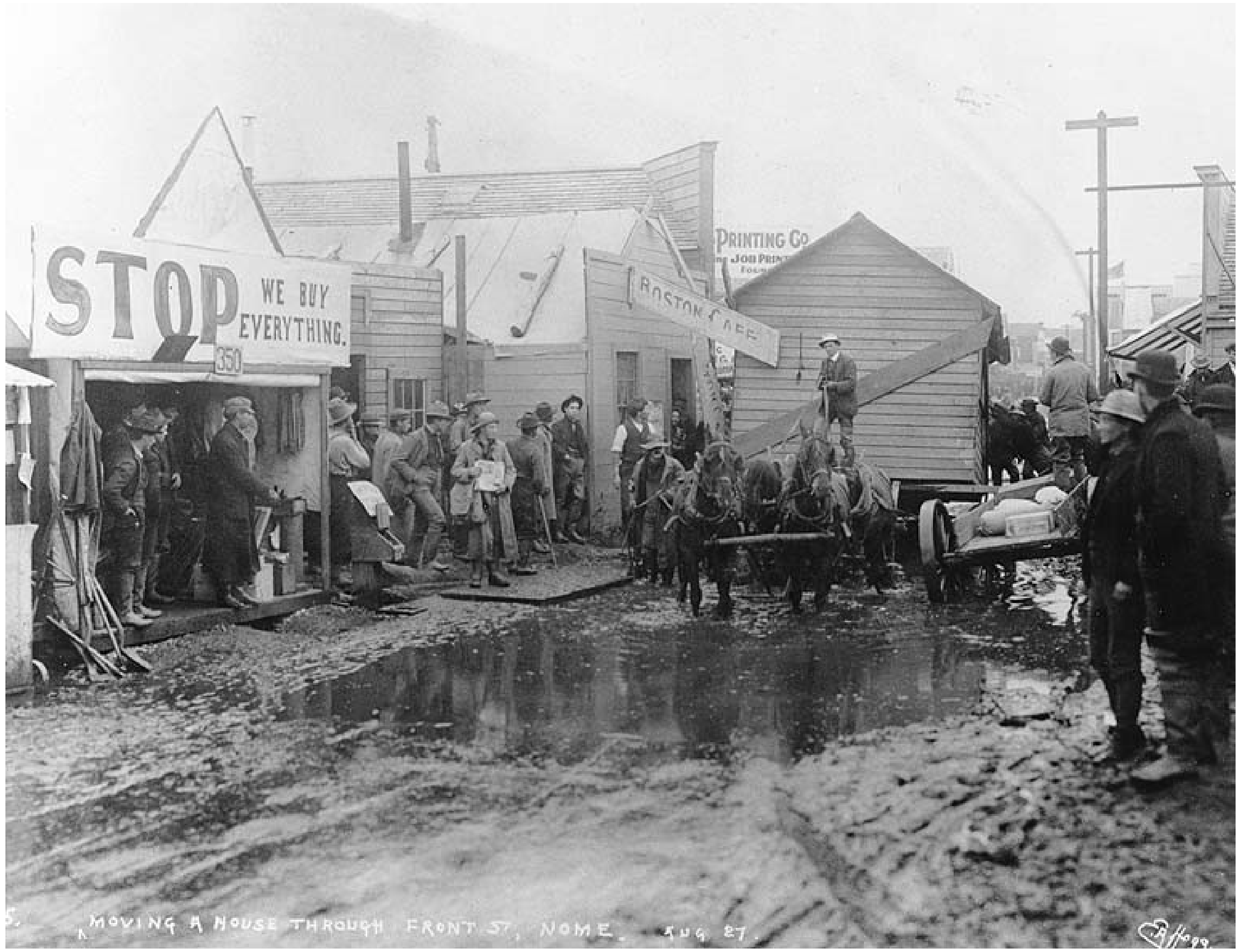
🪴🪴🪴

Software design as continuous

A good summary of general approach:

> "Make it work. Make it right. Make it fast."
>
> - Kent Beck (one reference)

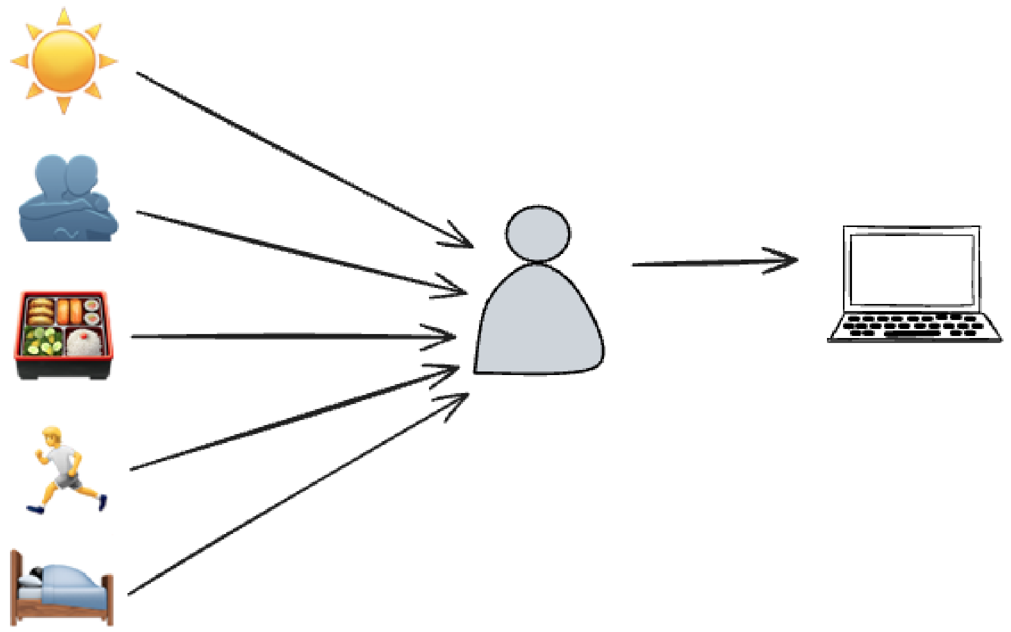
Movement in mud



Movement through mud takes energy!

Image: Eric A. Hegg, 1900 (Wikimedia Commons)

Knowledge work



- Software work is knowledge work.
- Knowledge work is afforded through a complex network of energy sources.

Software knowledge work fatigue

> “66% of developers rated the severity of fatigue during programming tasks as high or very high. 59% of developers rated the frequency of their fatigue during programming tasks as often or very often. Stress and sleepiness were the most voted causes.”

- S. Sarkar and C. Parnin, “Characterizing and Predicting Mental Fatigue during Programming Tasks,” 2017 (link)

Software work can be toiling!

Software fatigue - an estimate

🕒

How much mud can we cultivate (an estimate)?

- 6 hours a day to work (minus chats, restroom, food, travel)
- 3 hours a day for “deep work” (minus interruptions)
- 5 days a week * 3 hours = 15 hours a week for deep software work

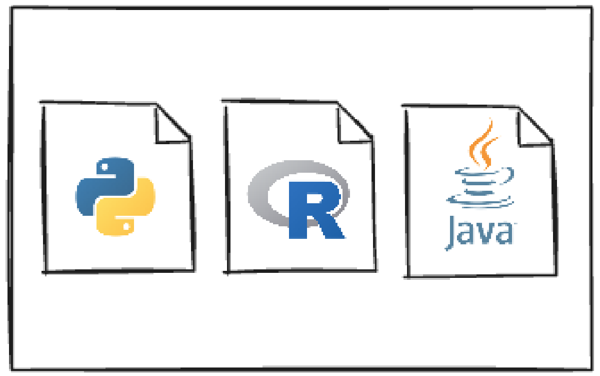
Software fatigue and piecemeal growth

We’re up against time on three fronts:

- Piecemeal growth entails (somewhat slow) incremental change.
- Development can only take place through a small time window.
- We have due dates from people expecting us to do something.

It’s crucial to be selective about what software development we do.

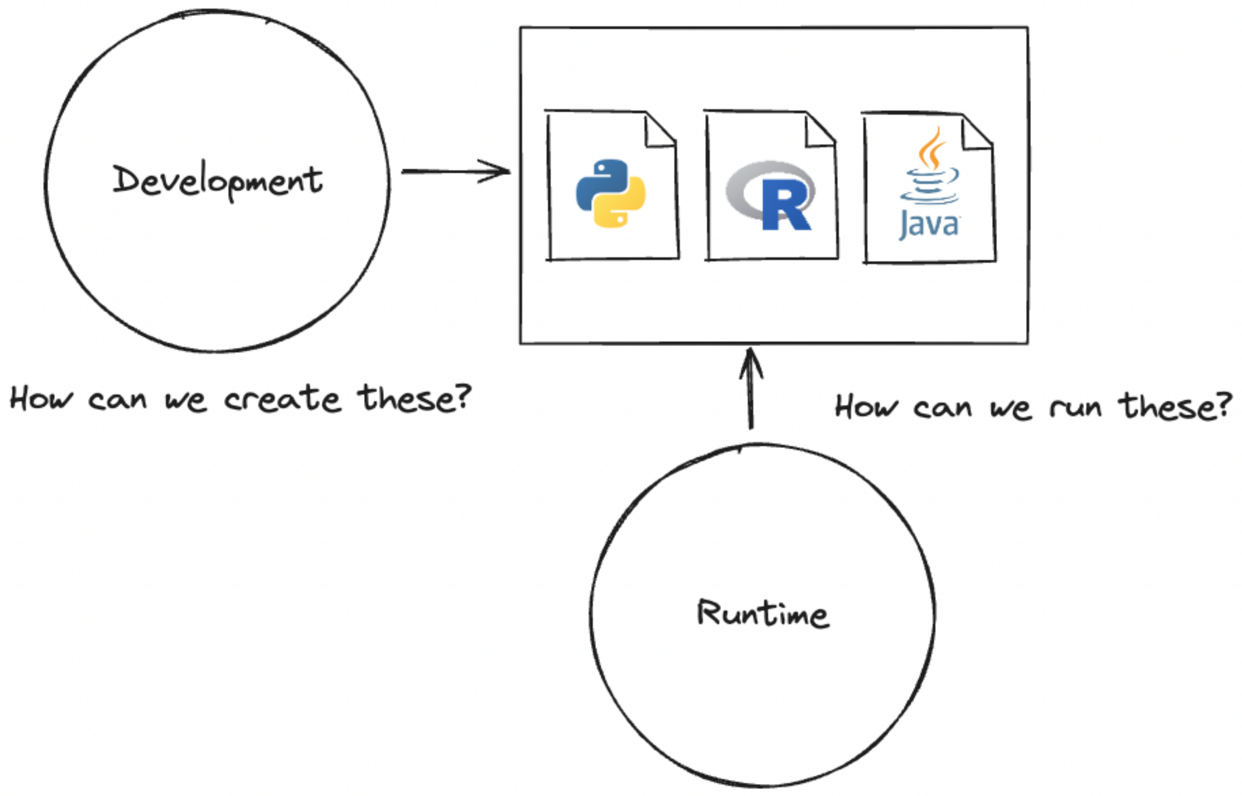
A Scenario



Accomplish a goal using “good enough” tools for a project.

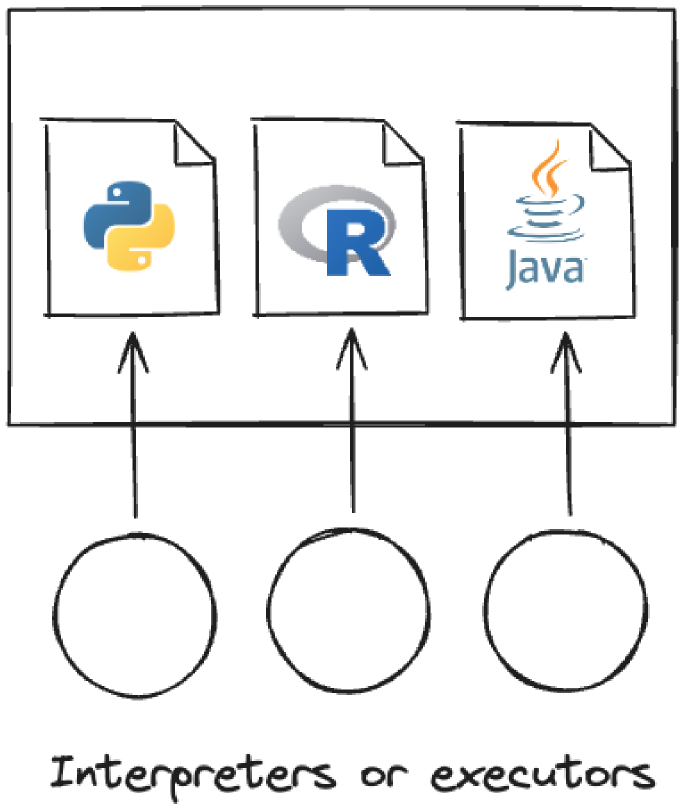
- We don’t have time to redevelop existing tools.
- We need to balance our time to reach due dates without burnout.

Environments



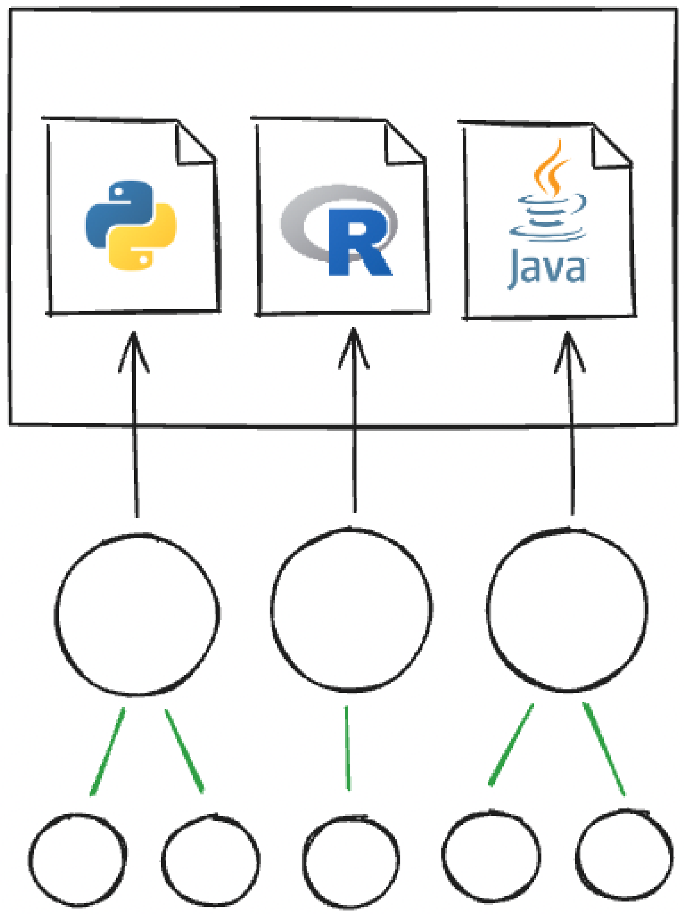
We can distinguish between development and runtime environments (not all are used for the same things).

Runtime environments



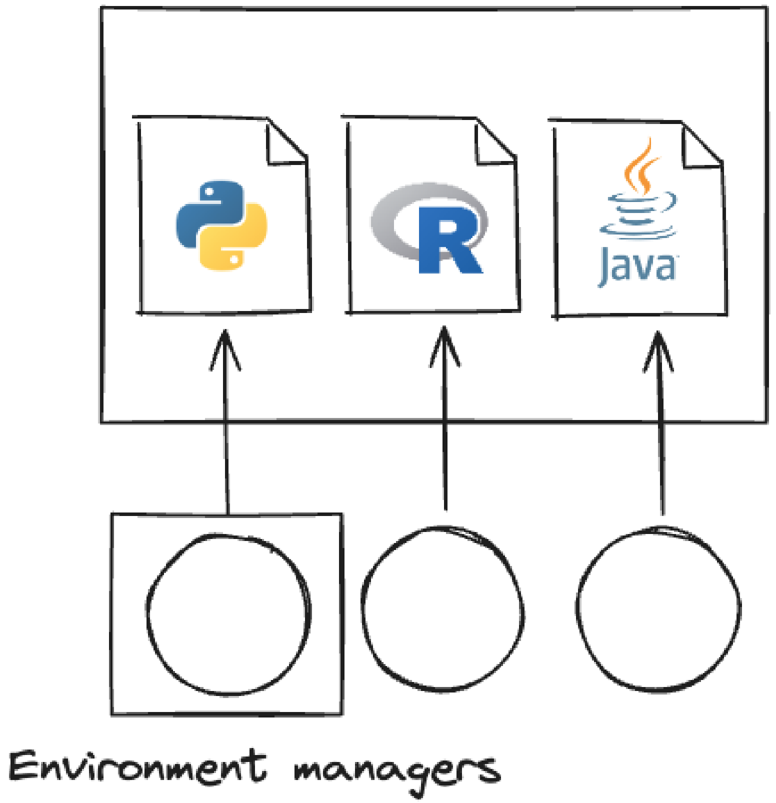
Runtime environments: each language entails a unique journey to execution or processing (compilers, interpreters, executors).

Runtime environments



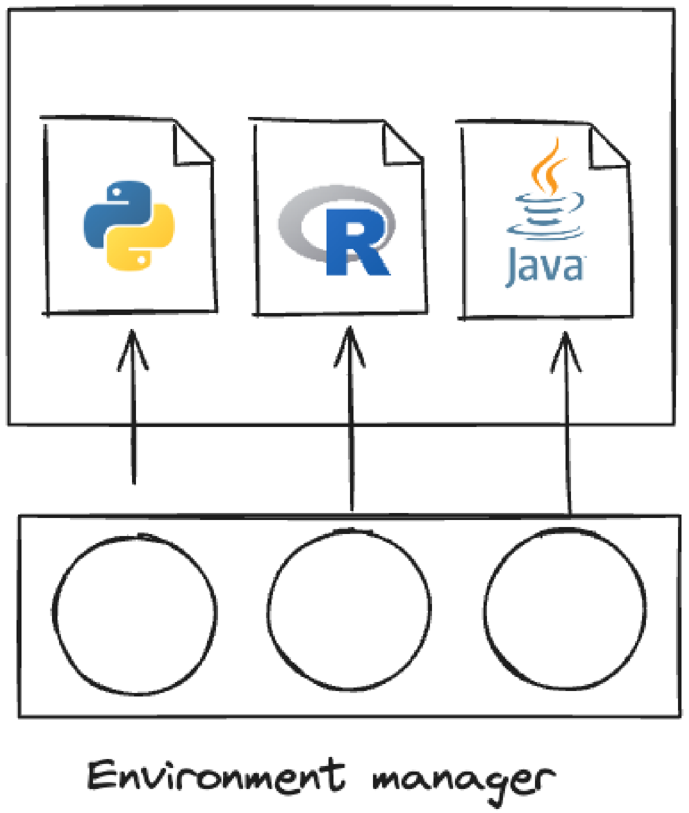
These artifacts connect to others as a complex system.

Runtime environments



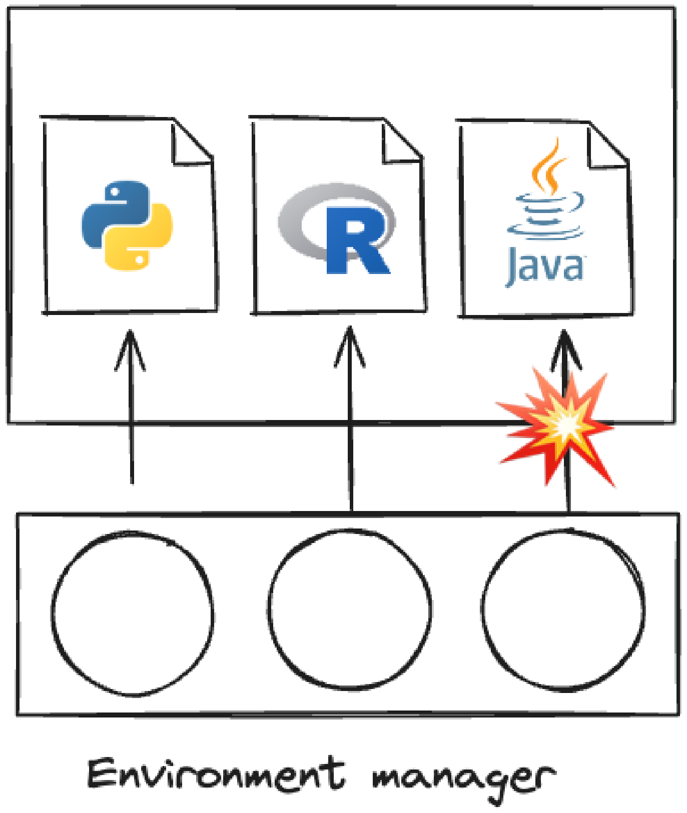
It’s healthy to wrap these in environment managers for reproducible results.

Runtime environments



Sometimes we can use one environment manager for all.

Runtime environments



Other times we can’t do this.

Runtime environments

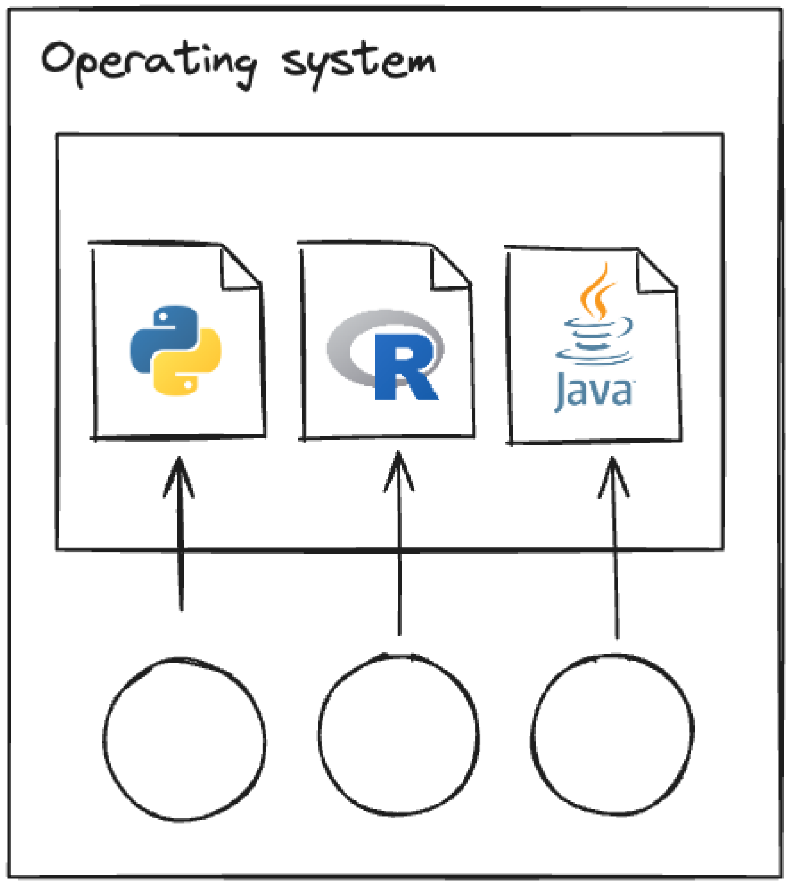
How do we “make it work” without burning too much time learning another language, execution tool, or other aspects?

Runtime environments

How do we “make it work” without burning too much time learning another language, execution tool, or other aspects?

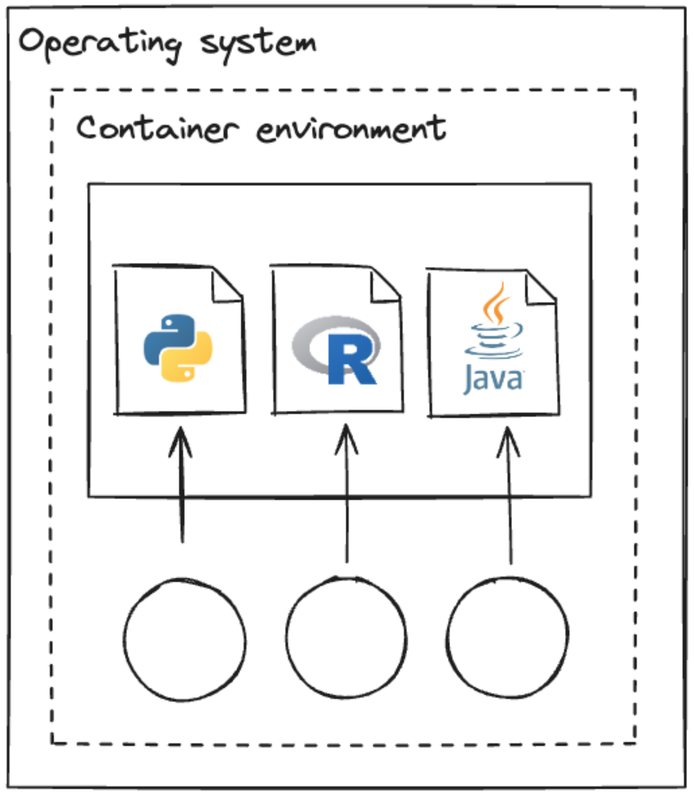
- Sense and acknowledge ***“shearing layers”***
    - Perhaps as changes of different time rates
- Consider ***“sweep it under the [container]”*** to ***“keep it working”***

Environments within a "world"



Environments exist within a "world" (an operating system).

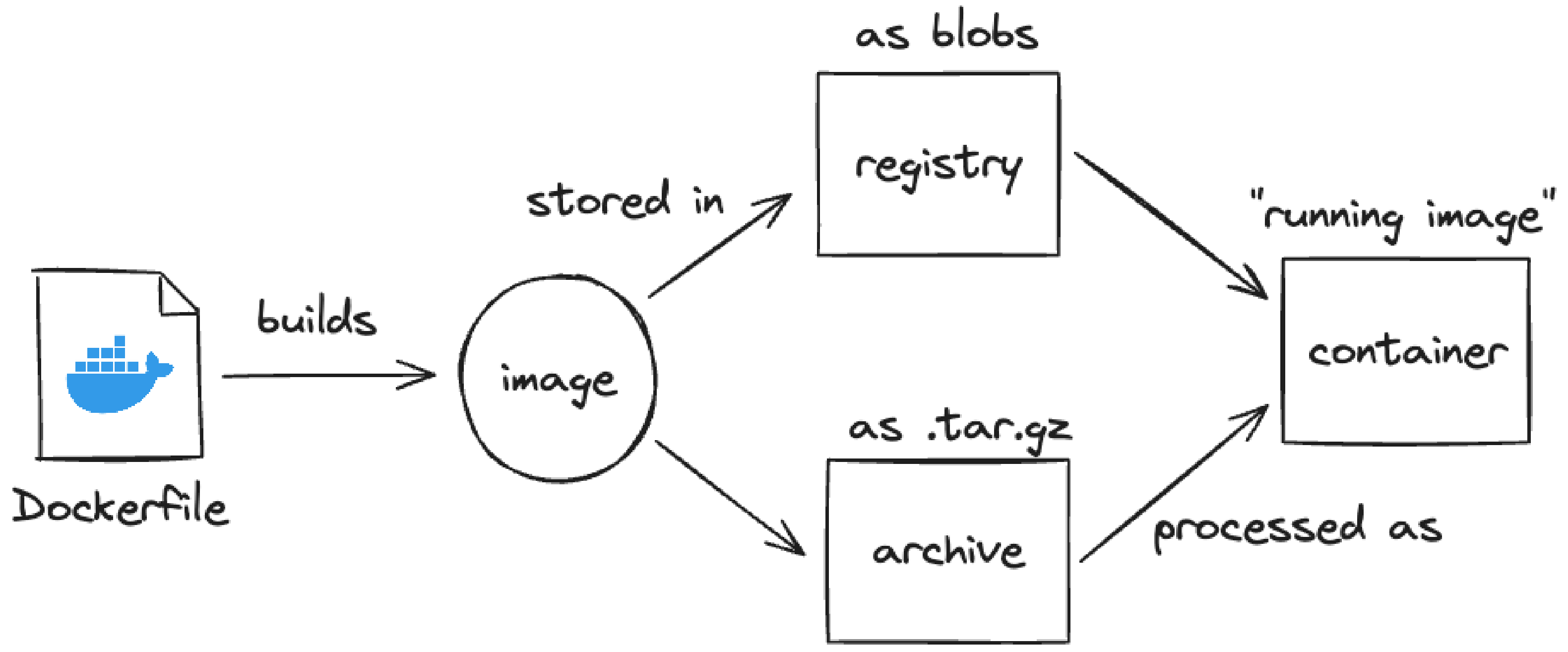
Managing an OS



We can manage operating system aspects with container environments (a kind of “virtual OS”).

What in the “world” is a container?

What is a container?



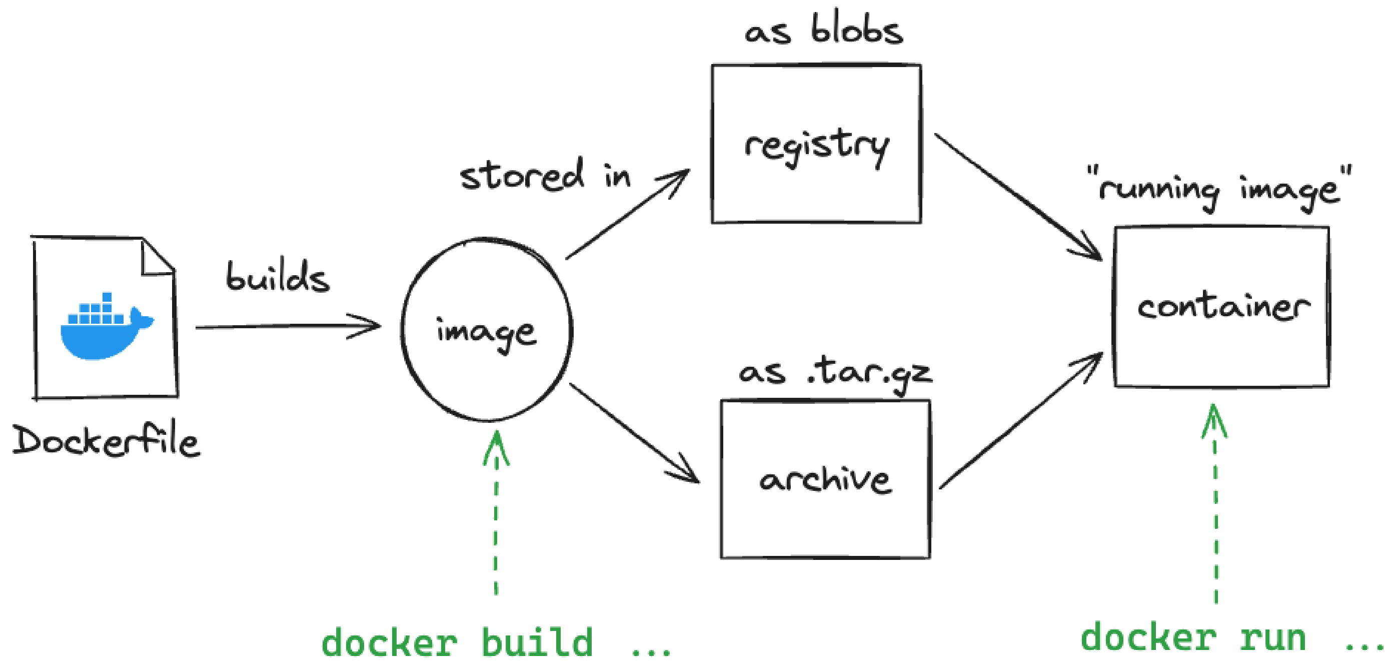
Dockerfiles are used to build images which can be processed as containers.

What does a Dockerfile look like?

```dockerfile
# base image for java
FROM openjdk:22-slim

# provide a version argument
ARG version=x.x.x

# set the workdir to /app
WORKDIR /app

# copy local data to image
COPY file.txt .

# install required packages
RUN apt-get update \
    && apt-get install --no-install-recommends -y wget

# Set the entrypoint for app
ENTRYPOINT ["/app/project"]
```

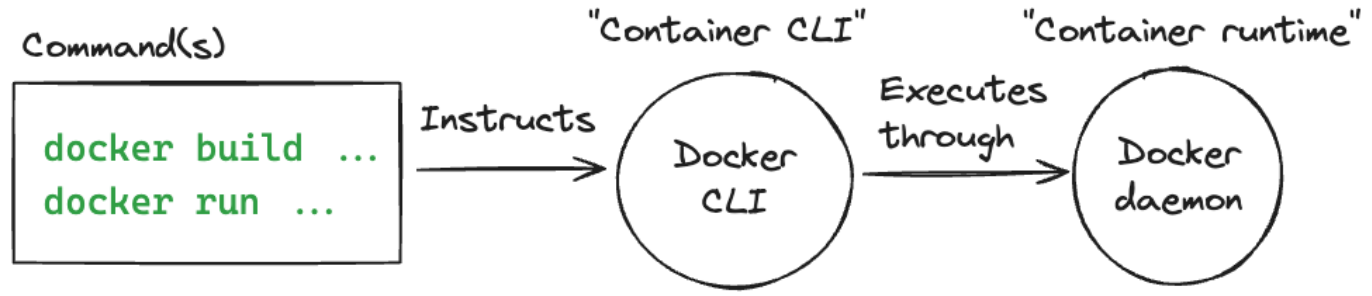
Dockerfiles are like a `bash` script to declare an environment.

How complicated are containers?



This process can be triggered using the `docker` CLI.

How complicated are containers?

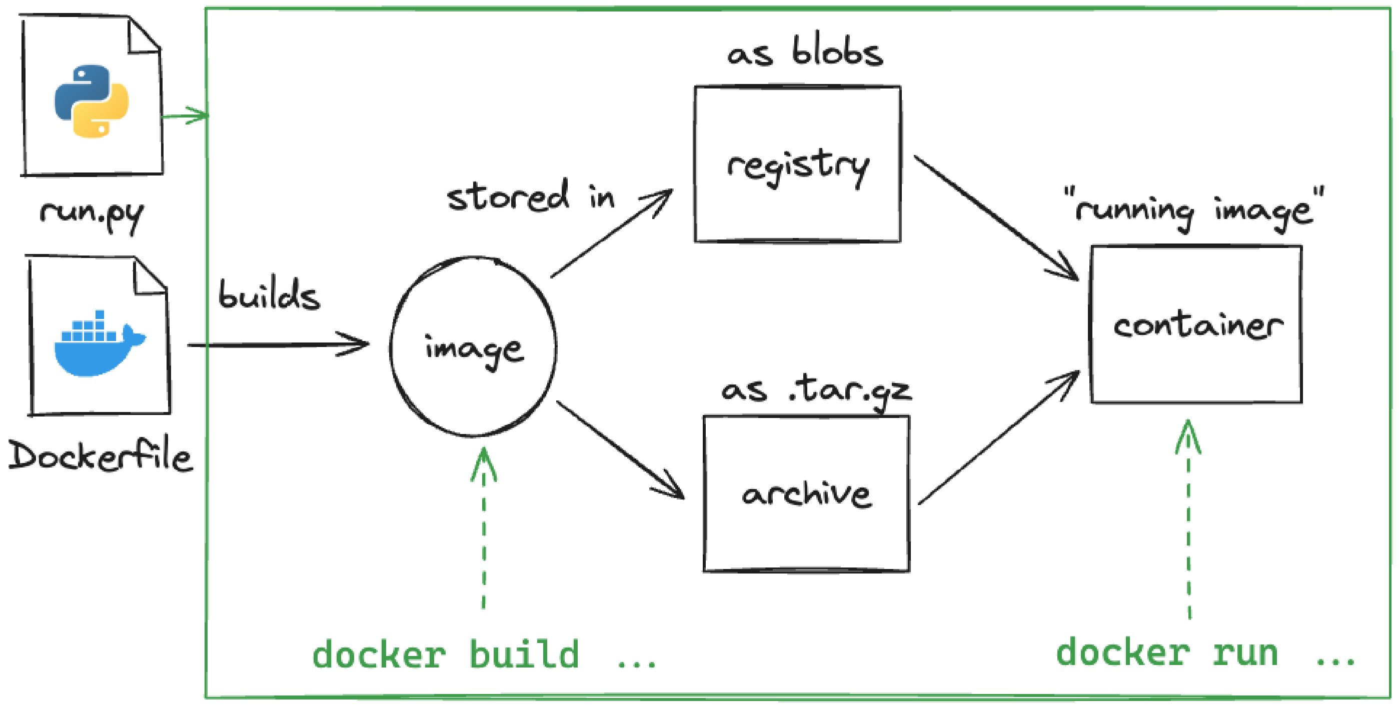


```
docker build  ...
docker run  ...
```

Commands are sent to a container CLI which then asks a container runtime to do something.

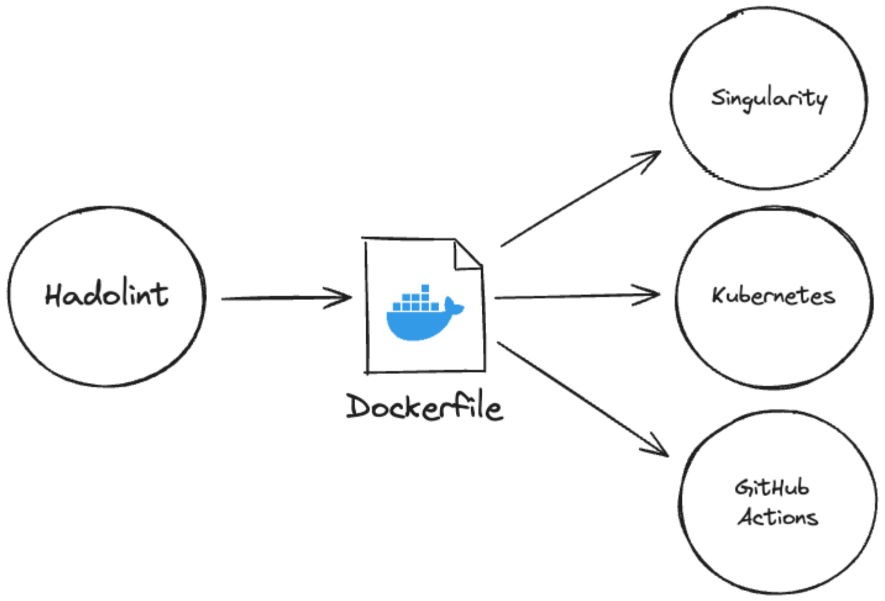
- Docker is one container runtime of many.

How complicated are containers?



Containers can also be used through source development kits (SDK’s) (like Python).

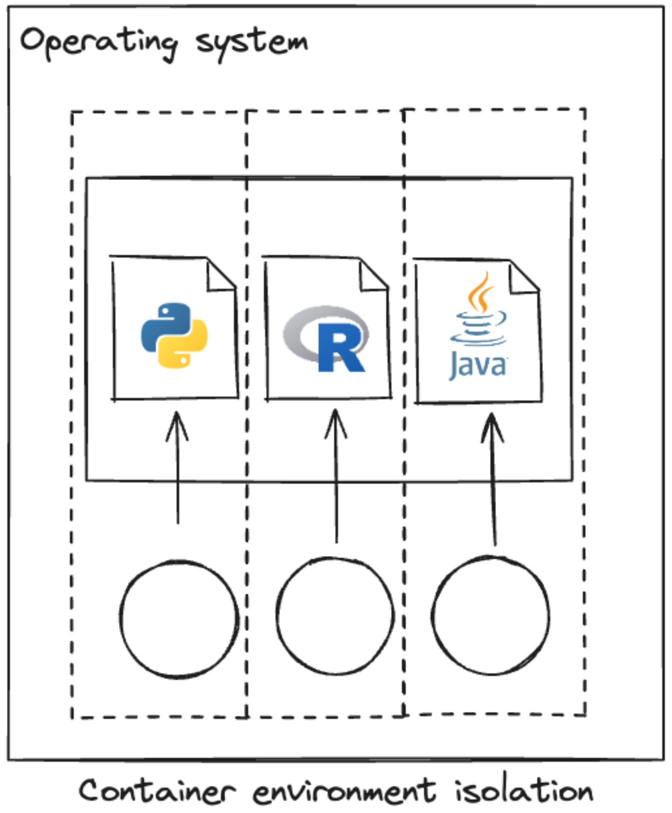
Why are you telling me about Docker?



- Dockerfiles and Docker are common (+ *collaboration*)
- Well integrated (+ *toolbelt*)
- Documented, versionable, and lintable (+ *DevEx*)
- Reduce impacts of ***time*** (+ *layer control*)

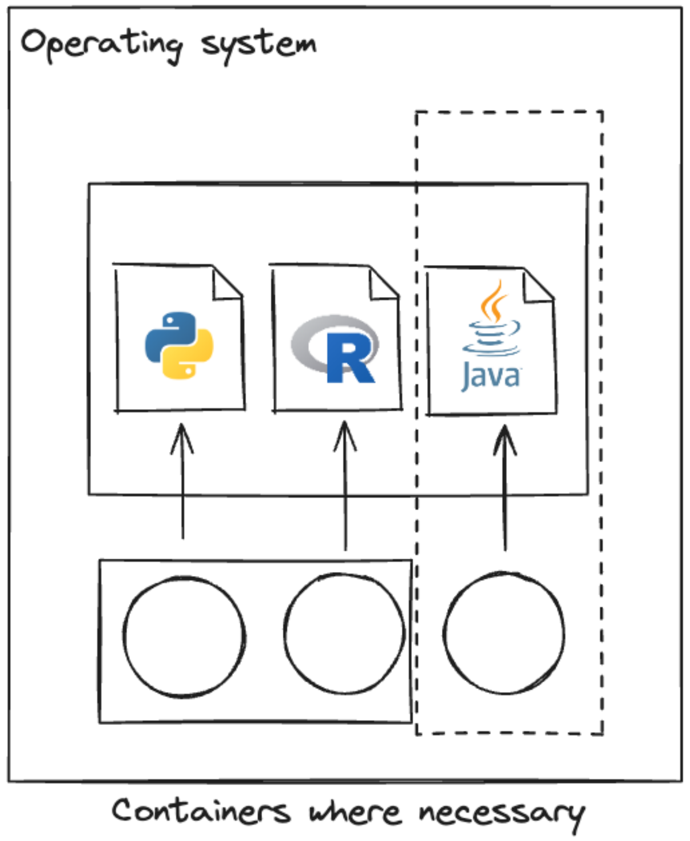
Back to usage!

Isolated container environments



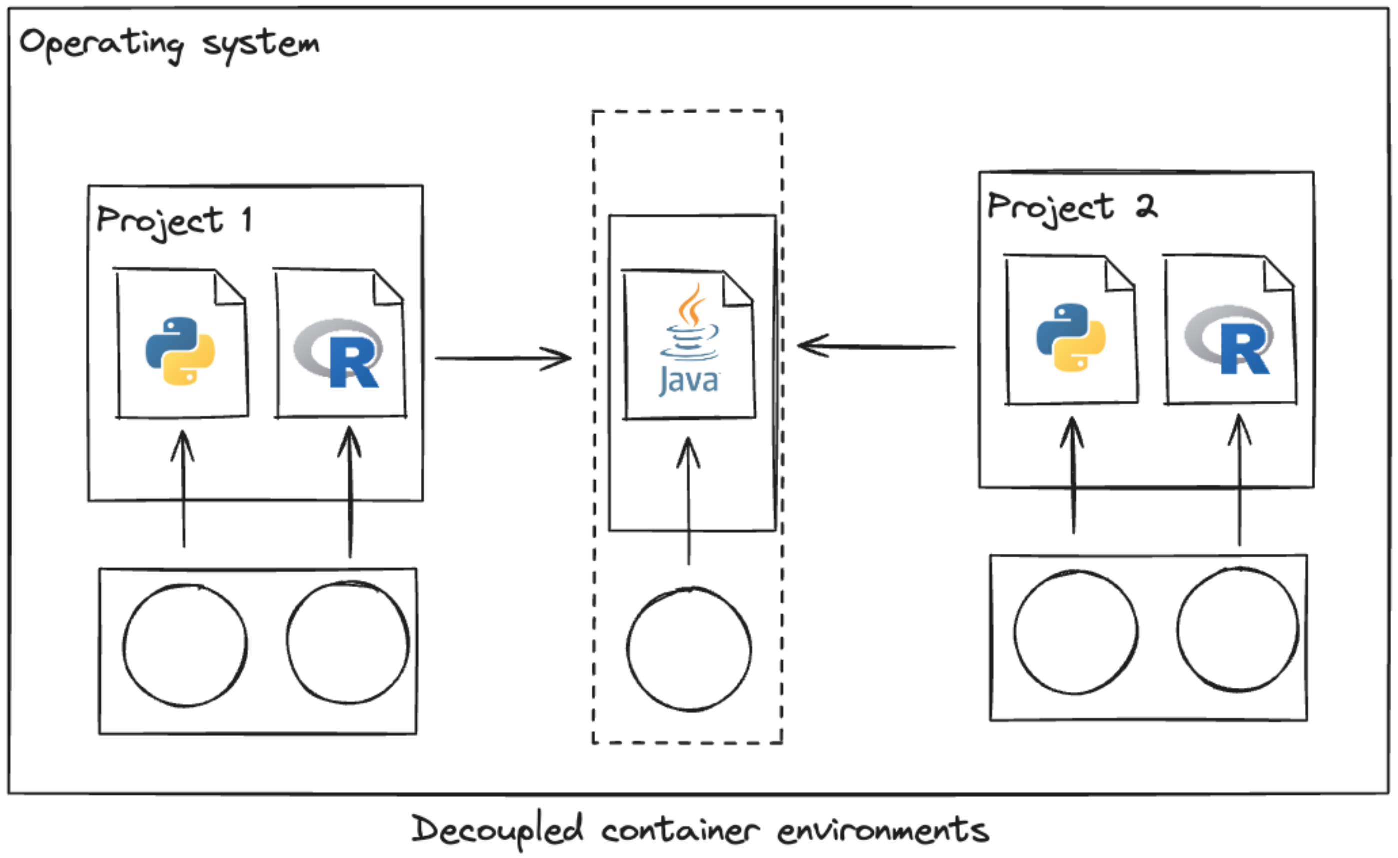
Container environments can be handled independently.

Multiple environments



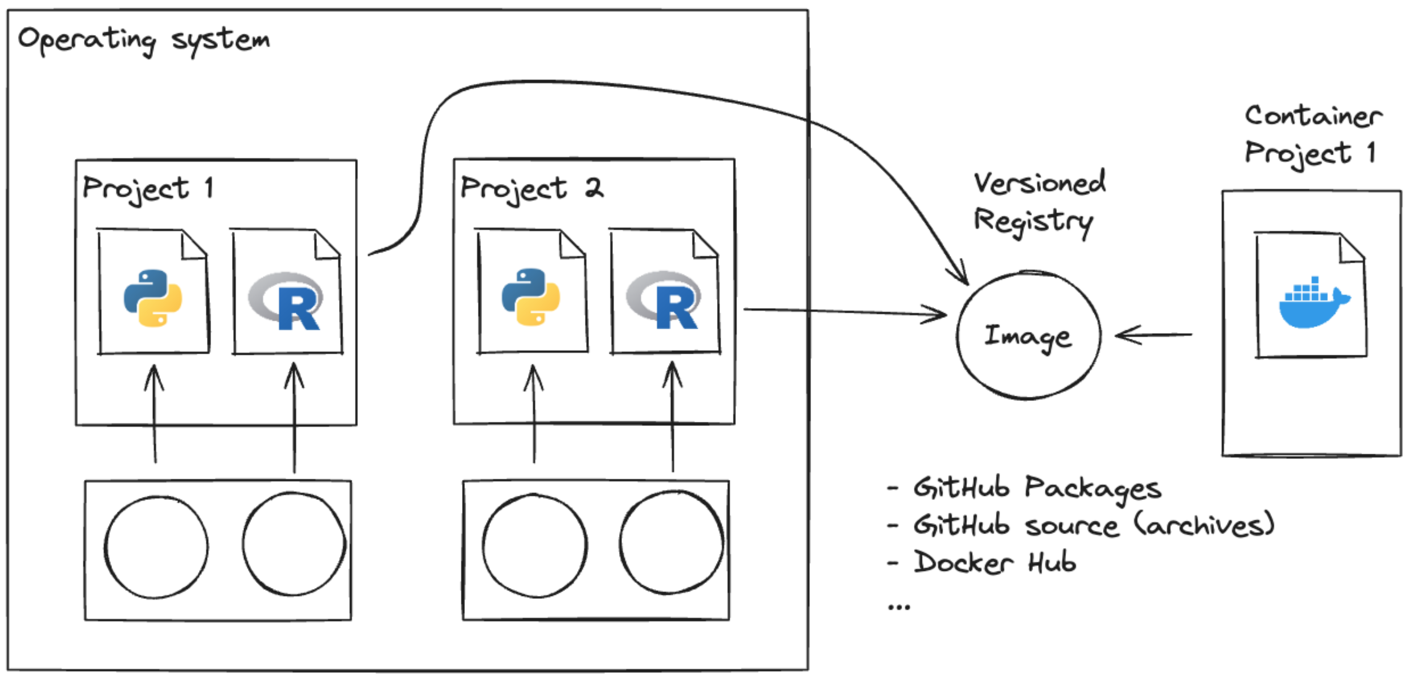
We can balance container environments alongside other environments.

Loosely coupled container environments



We can loosely couple with projects for reusability.

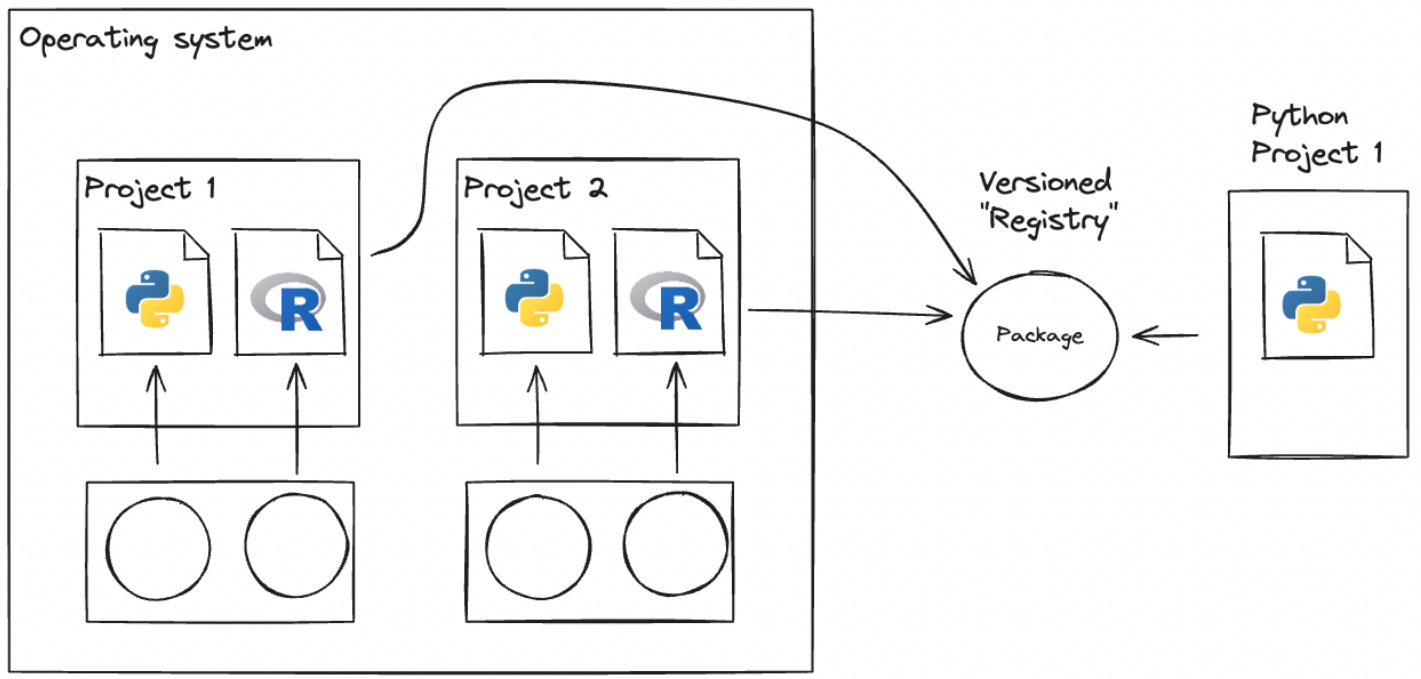
Registry decoupling



Further decoupling could mean greater reuse.

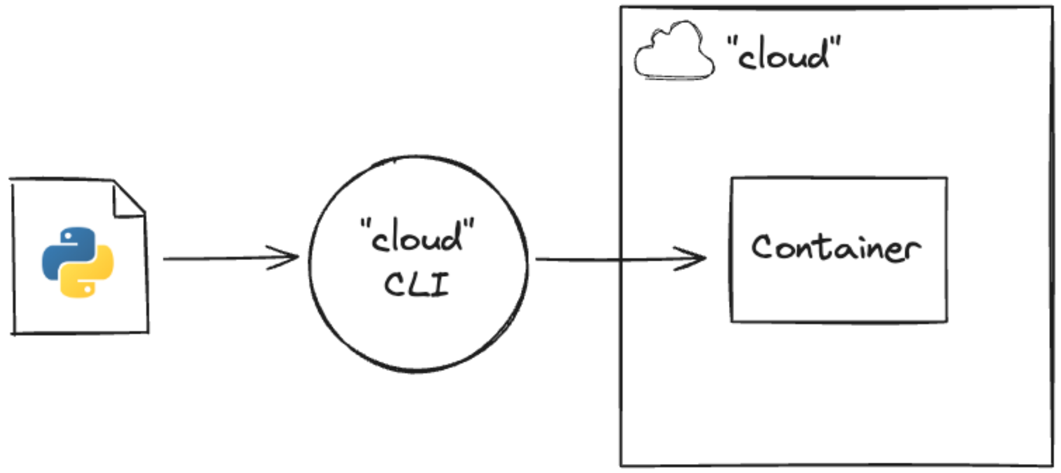
It could also be too much form before function.

Python decoupling



The same pattern works with Python code.

Additional notes: FaaS and "Serverless"



- Container technologies are for many purposes (including sophisticated engineering).
- They form the basis of Function-as-a-Service ("Serverless").

Recap

- *“Make it work. Make it right. Make it fast.”*
- Balancing our time and energy.
- Does this effectively manage the “shearing layers?” (or otherwise?)

Thank you!

Thank you for attending! Questions / comments?

Please don’t hesitate to reach out!