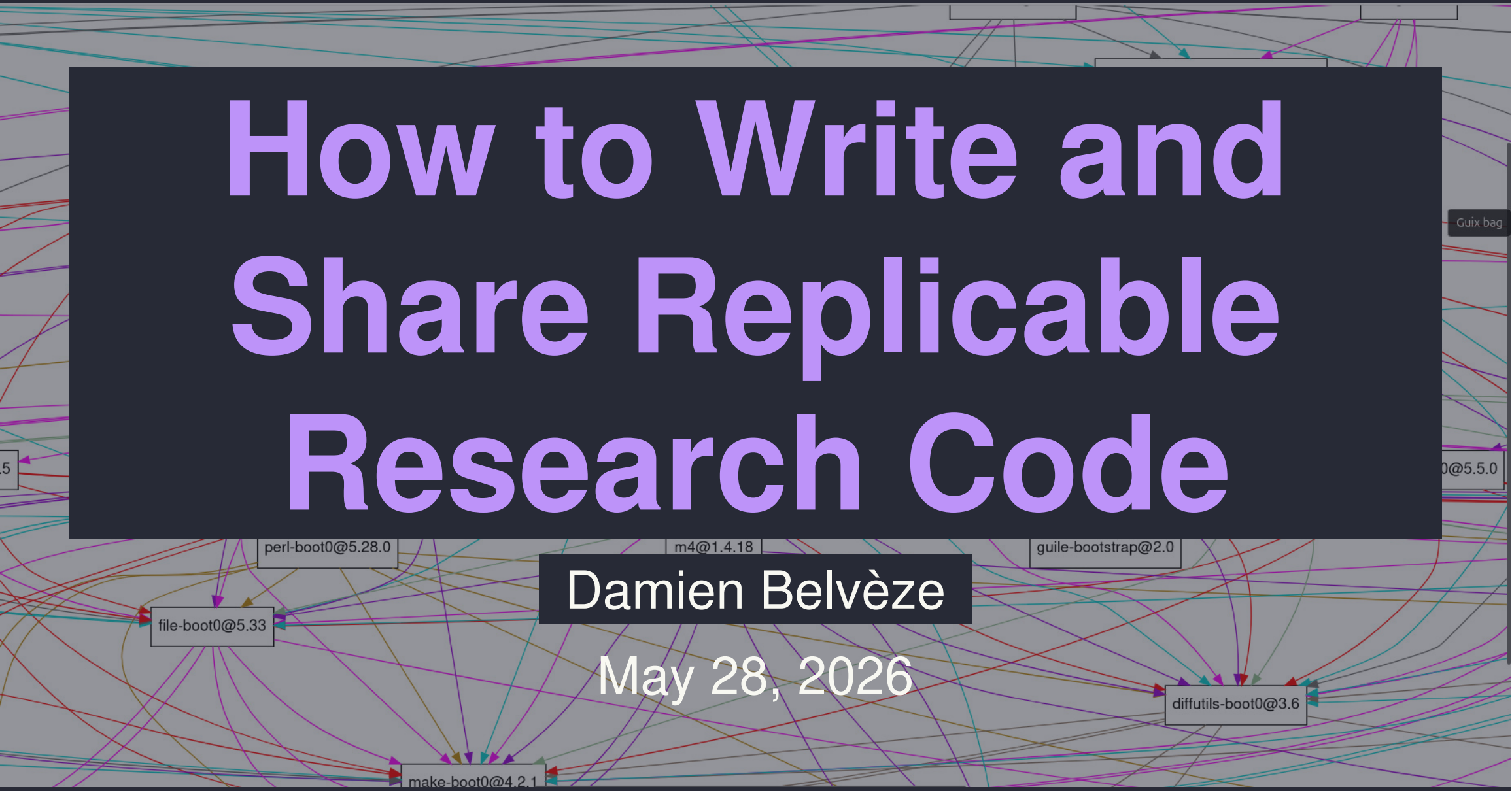
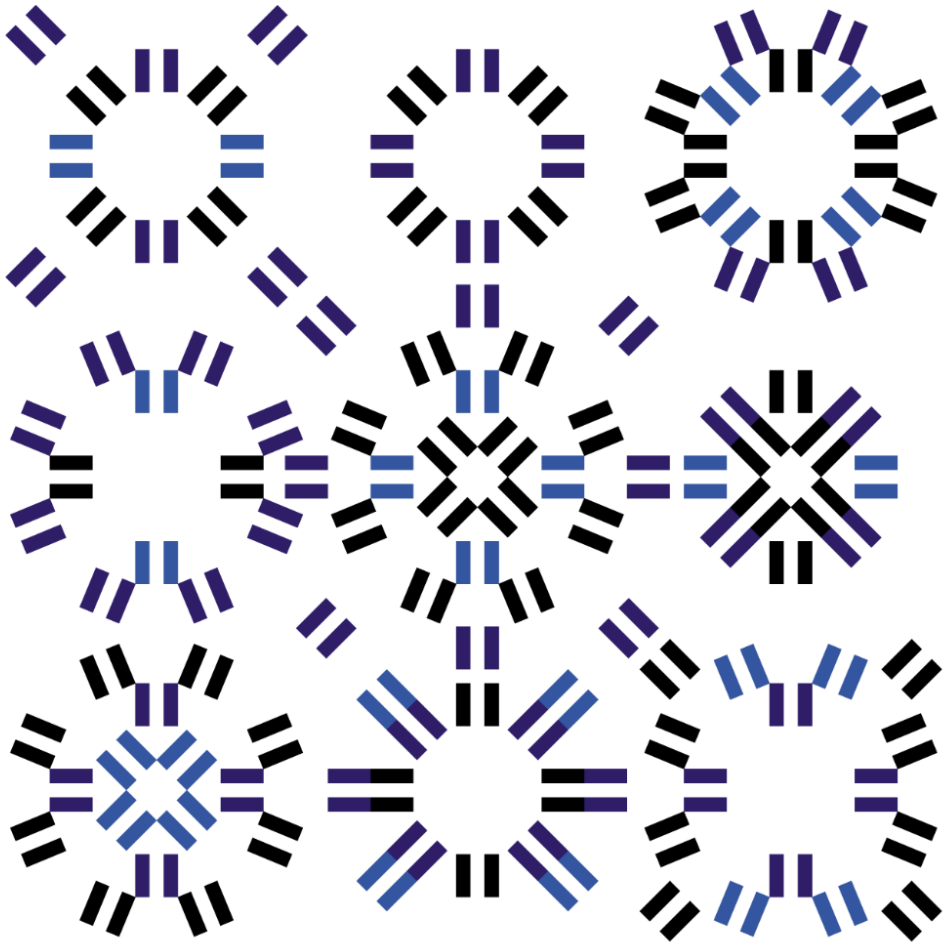


How to Write and Share Replicable Research Code

Damien Belvèze

May 28, 2026

Université de Rennes



How to Write and Share Replicable Research Code

press S key to access slide notes, F : back to presentation view

European Digital UniverCity

Université de Rennes

UNIVERSITAT JAUME·I

what is actualy doing Open Science

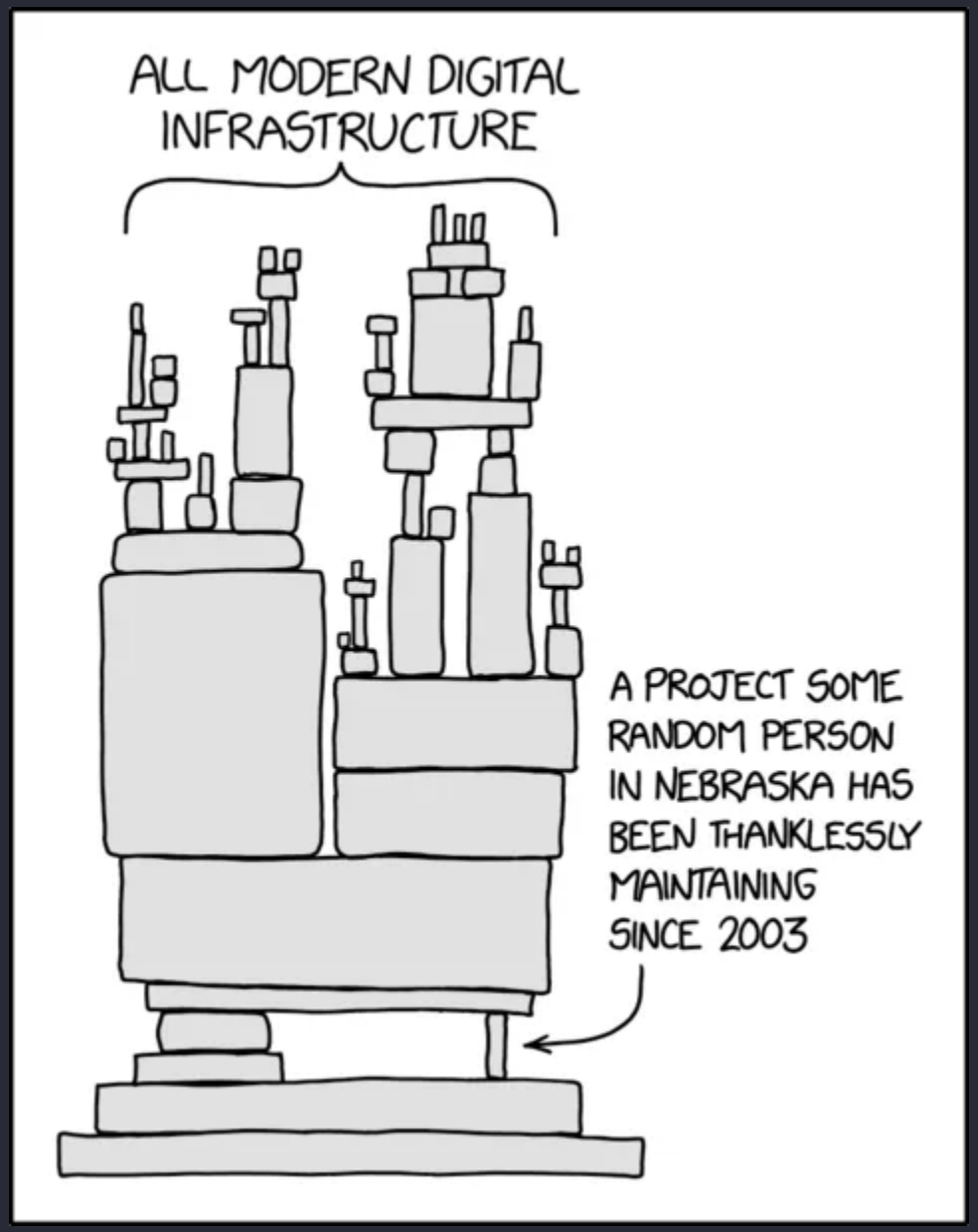
Open Science is about :

- sharing things by making them findable and accessible
- adapting them to existing artifacts (interoperability)
- preparing and documenting them so they could be reused (reusability)
- opening them to contribution (specific for source code)

a source code is static…

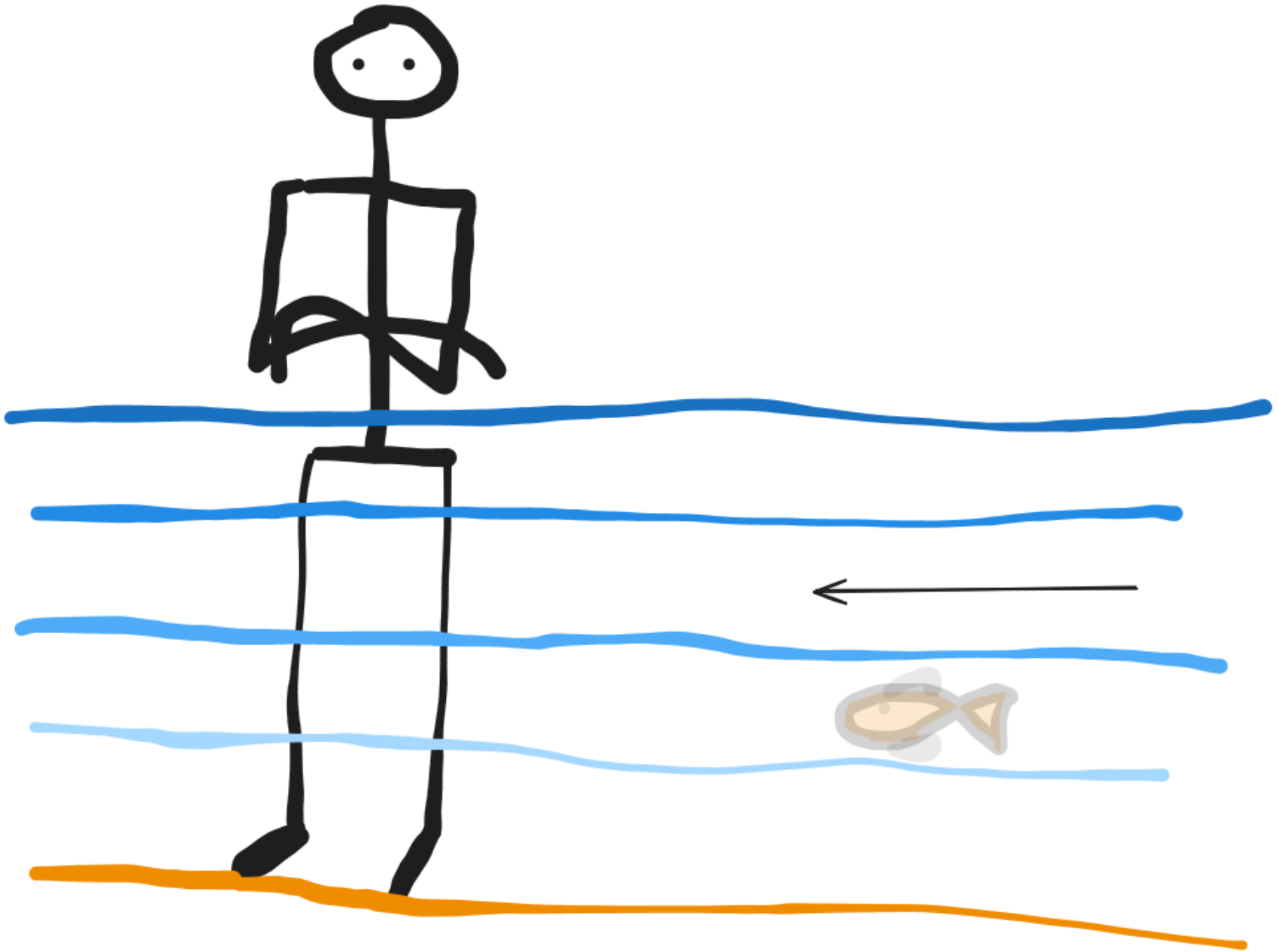
…but the environment which it’s designed for is moving quickly…

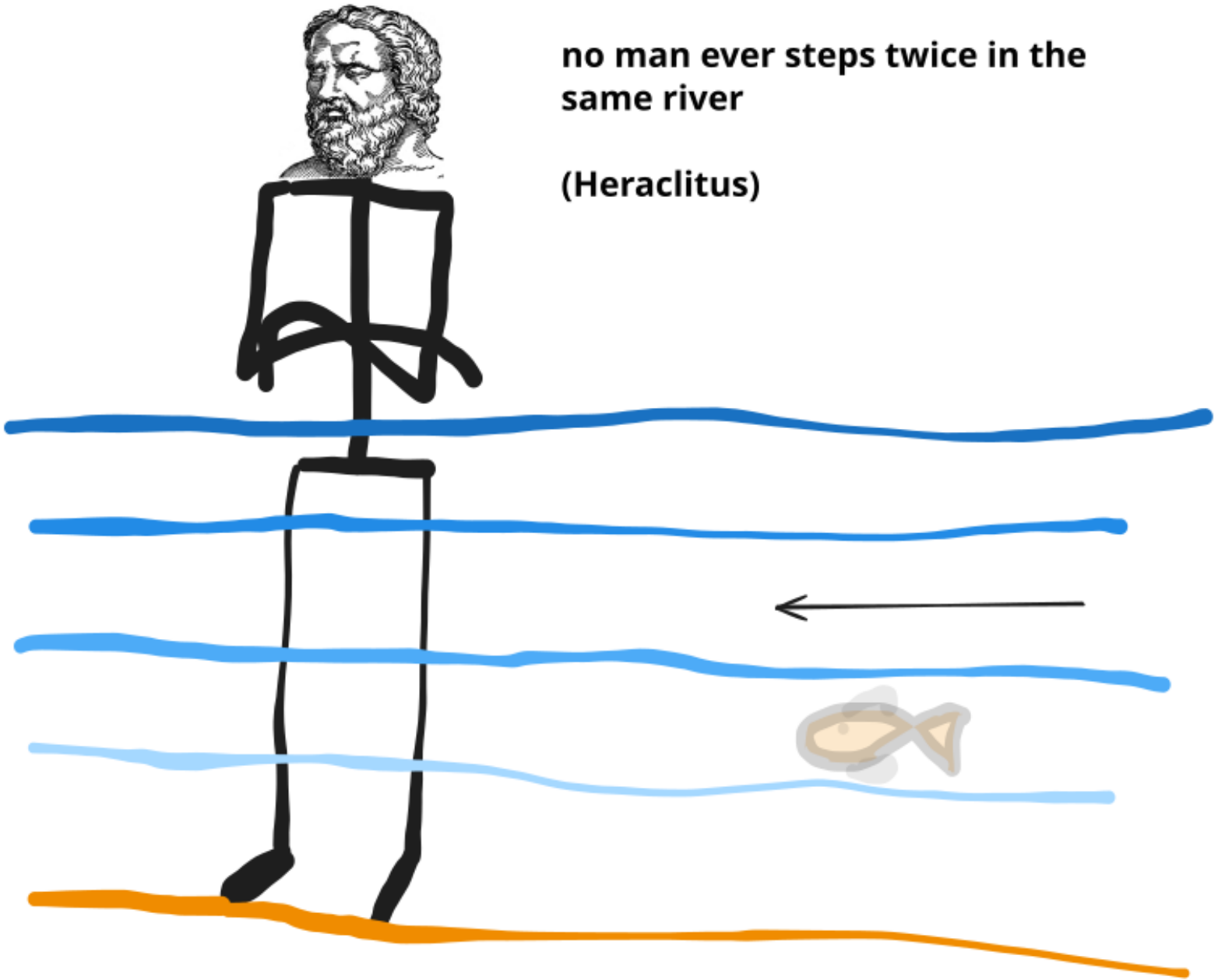
…creating code is like building house of cards grounded on sand



“In modern software there are always dependencies. No one spells out the ones and zeros for the machine by hand, so we write towers of software built on towers of software built on towers of software. And these dependencies change”

Fobbe (2024)



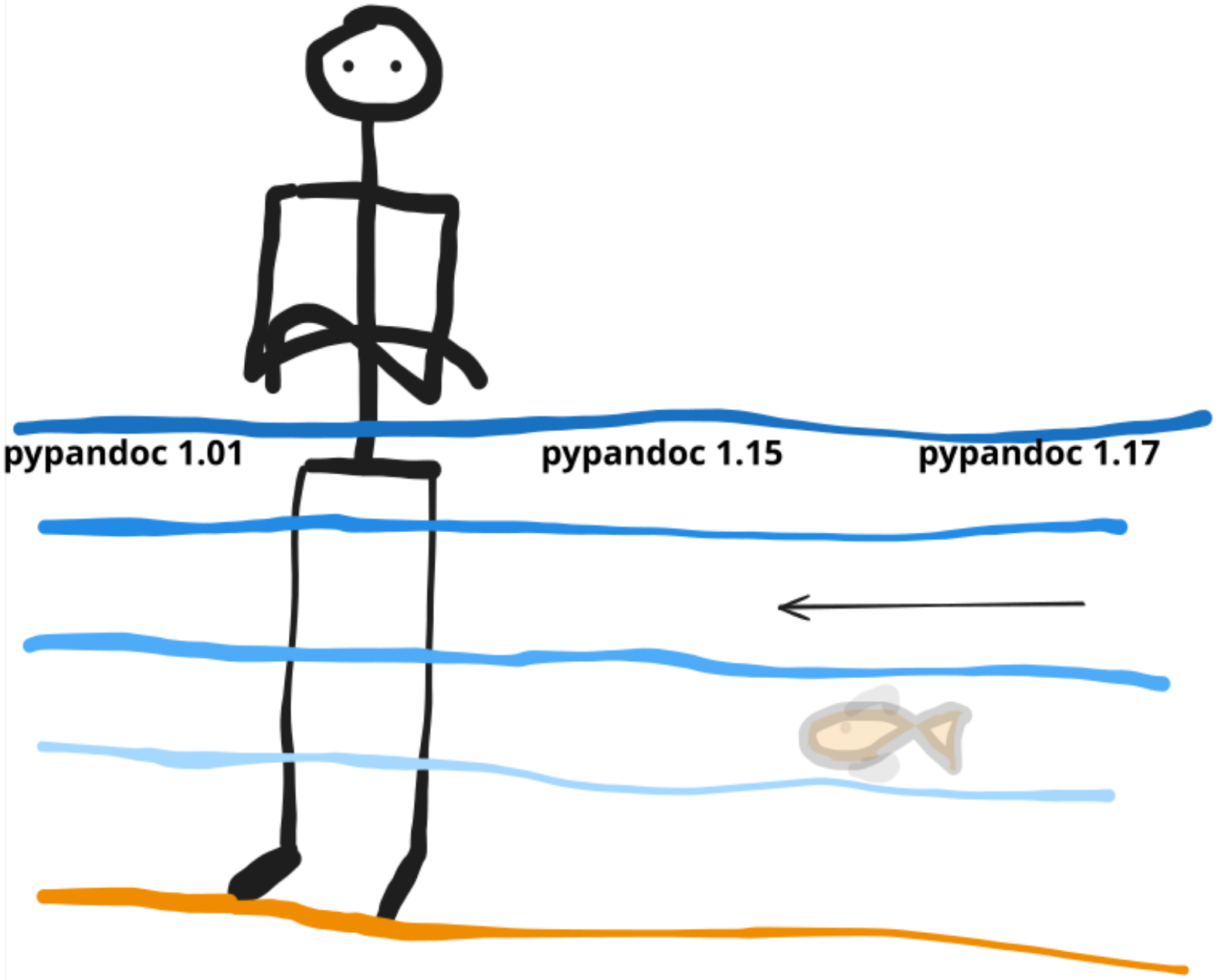


no man ever steps twice in the same river

(Heraclitus)

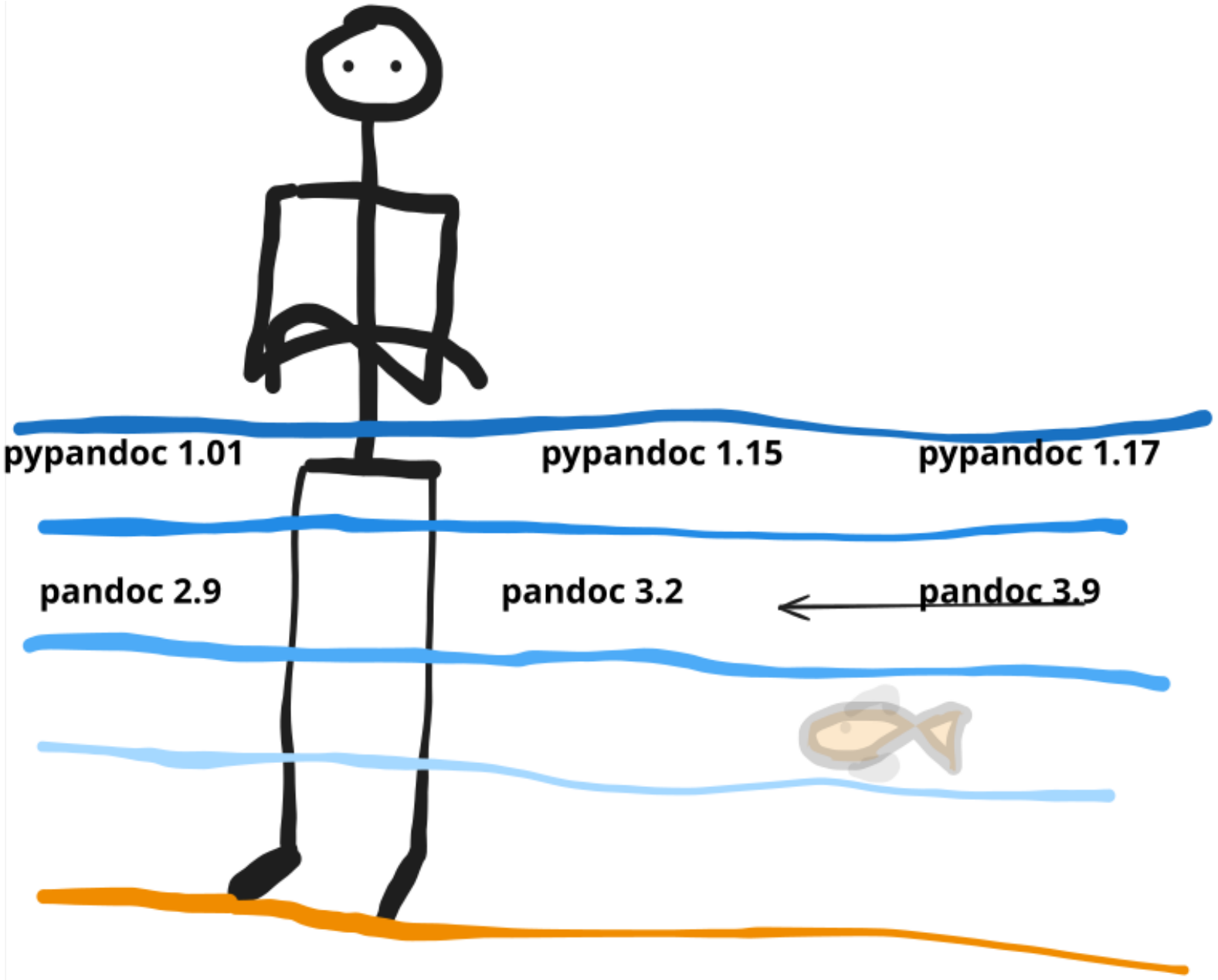
Speaker notes

Heraclitus used to say that the man also was not the same. But source code does not change by itself, and little care will be provided to this artifact after the publications is done. While it remains stable, its environment is continuously changing. the stream of the river (and of the time) goes from right to left. The libraries which are called are constantly updated with bug fixes and new functions added.


pypandoc 1.01
pypandoc 1.15
pypandoc 1.17

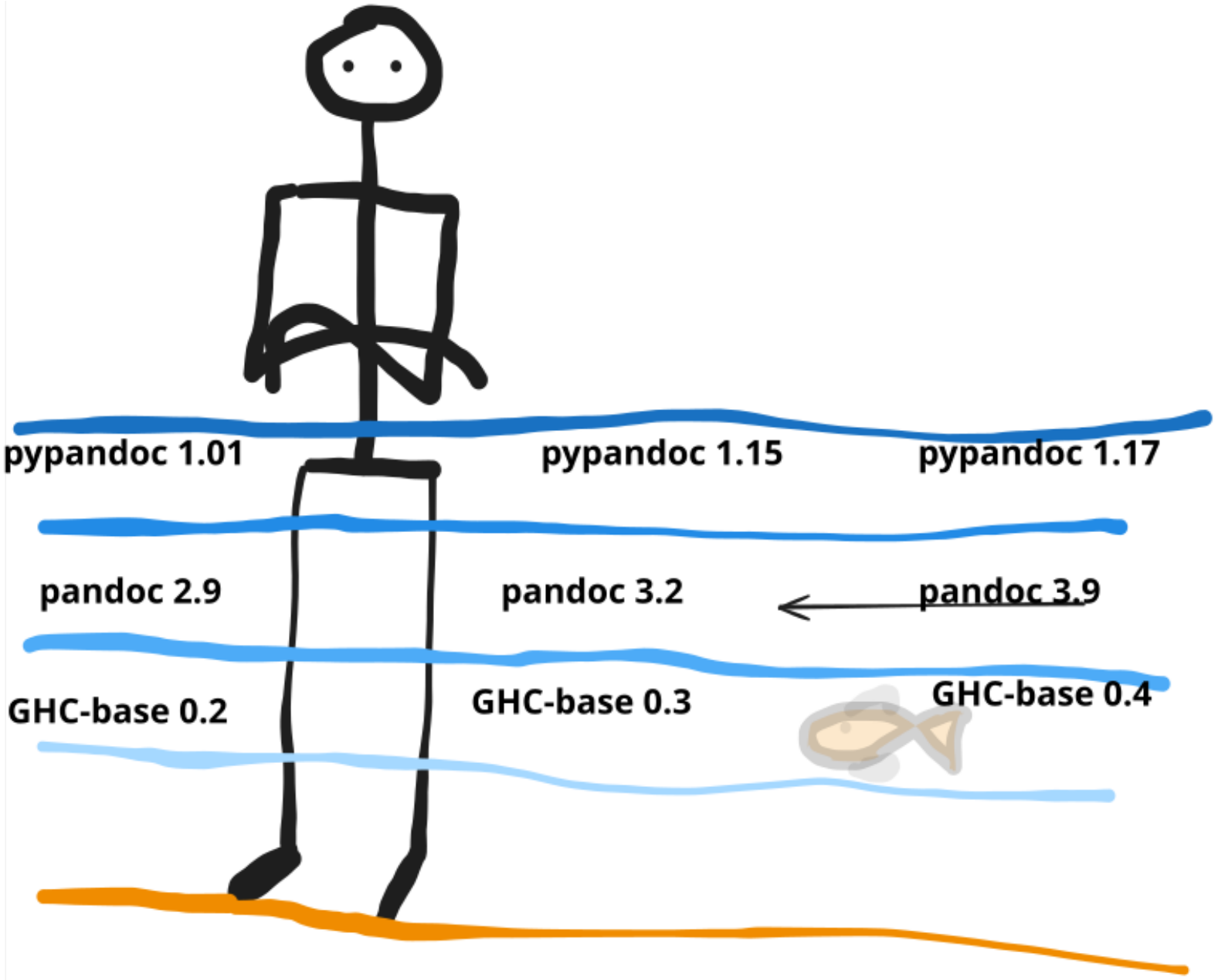
Speaker notes

let’s take a simple scritp that converts with Python a bunch of files from a format to another. this Python script needs the pypandoc library. Several versions of it can still be found online. the latest ones are on the right side of the image.


pypandoc 1.01
pypandoc 1.15
pypandoc 1.17
pandoc 2.9
pandoc 3.2
pandoc 3.9

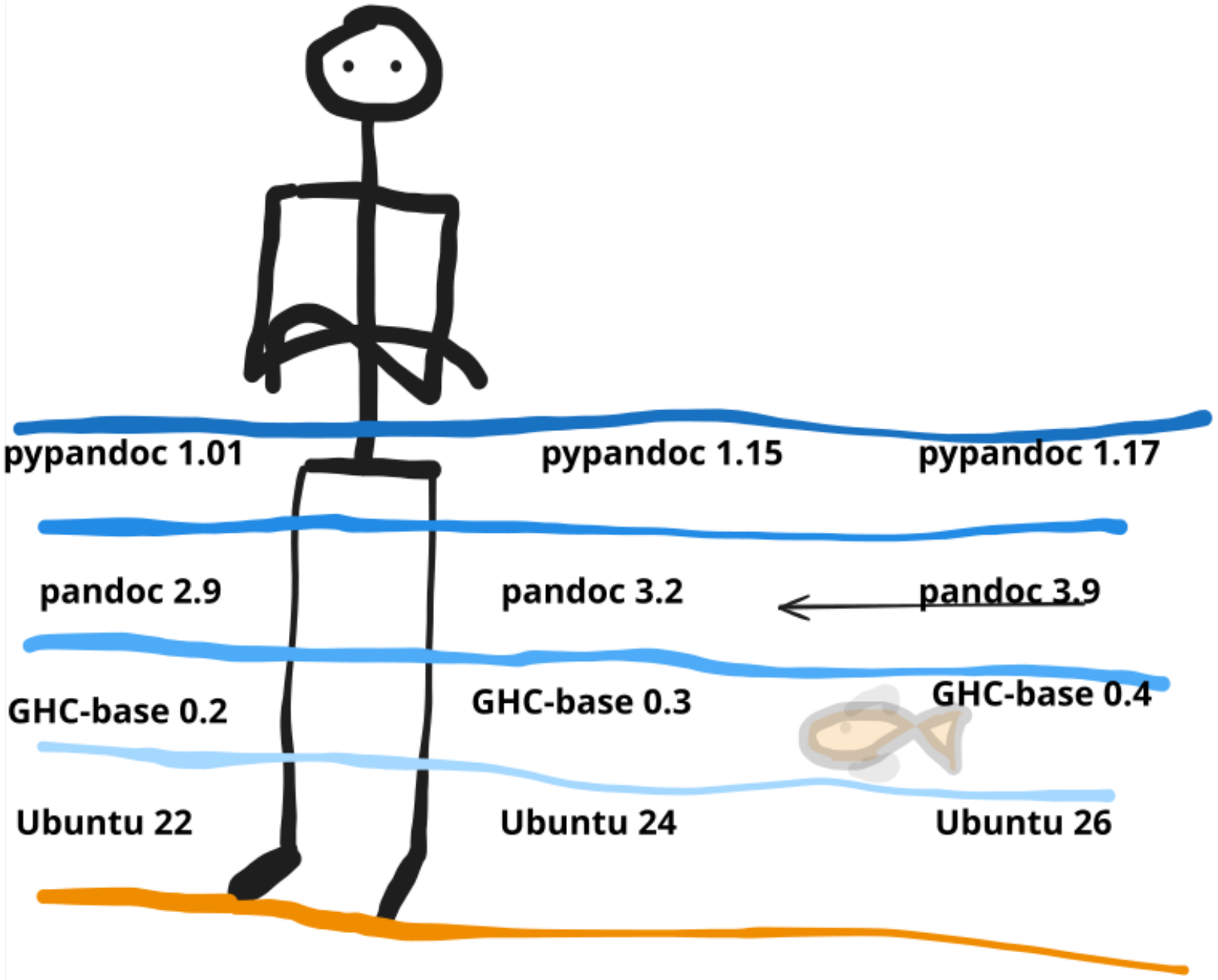
Speaker notes

as a package dependency, pypandoc needs a version of pandoc installed on the system to work. the pandoc version should be adapted to the pypandoc python library let’s say pandoc 3.2 will fit.


pypandoc 1.01
pypandoc 1.15
pypandoc 1.17
pandoc 2.9
pandoc 3.2
pandoc 3.9
GHC-base 0.2
GHC-base 0.3
GHC-base 0.4

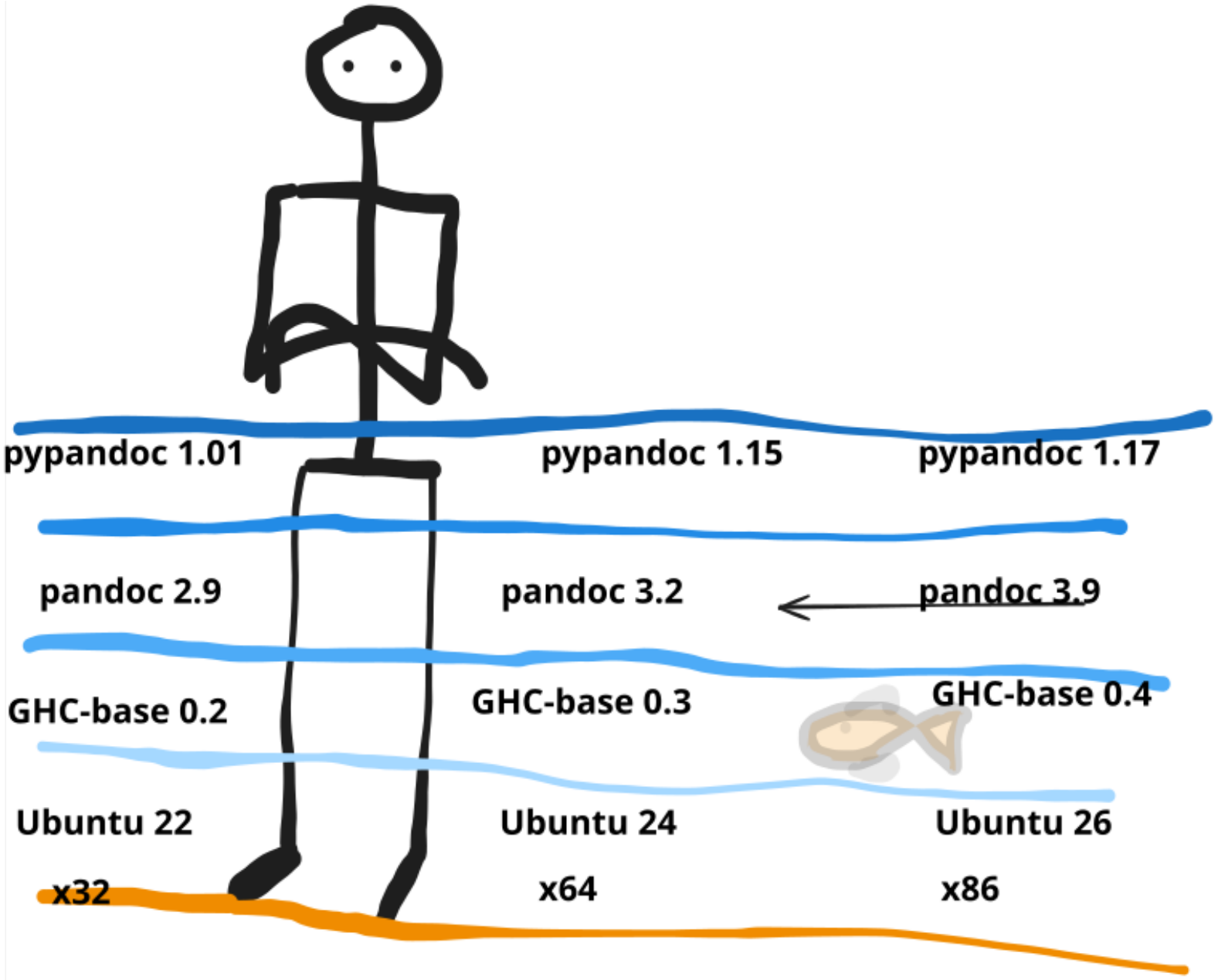
Speaker notes

pandoc can be used as a standalone programme or a bundle (python pandoc version) as a standalone programme, since it’s written in Haskell, in order to work, pandoc needs a Haskell compiler installed on the system GHC compiles Haskell programmes to binaries


pypandoc 1.01
pypandoc 1.15
pypandoc 1.17
pandoc 2.9
pandoc 3.2
pandoc 3.9
GHC-base 0.2
GHC-base 0.3
GHC-base 0.4
Ubuntu 22
Ubuntu 24
Ubuntu 26

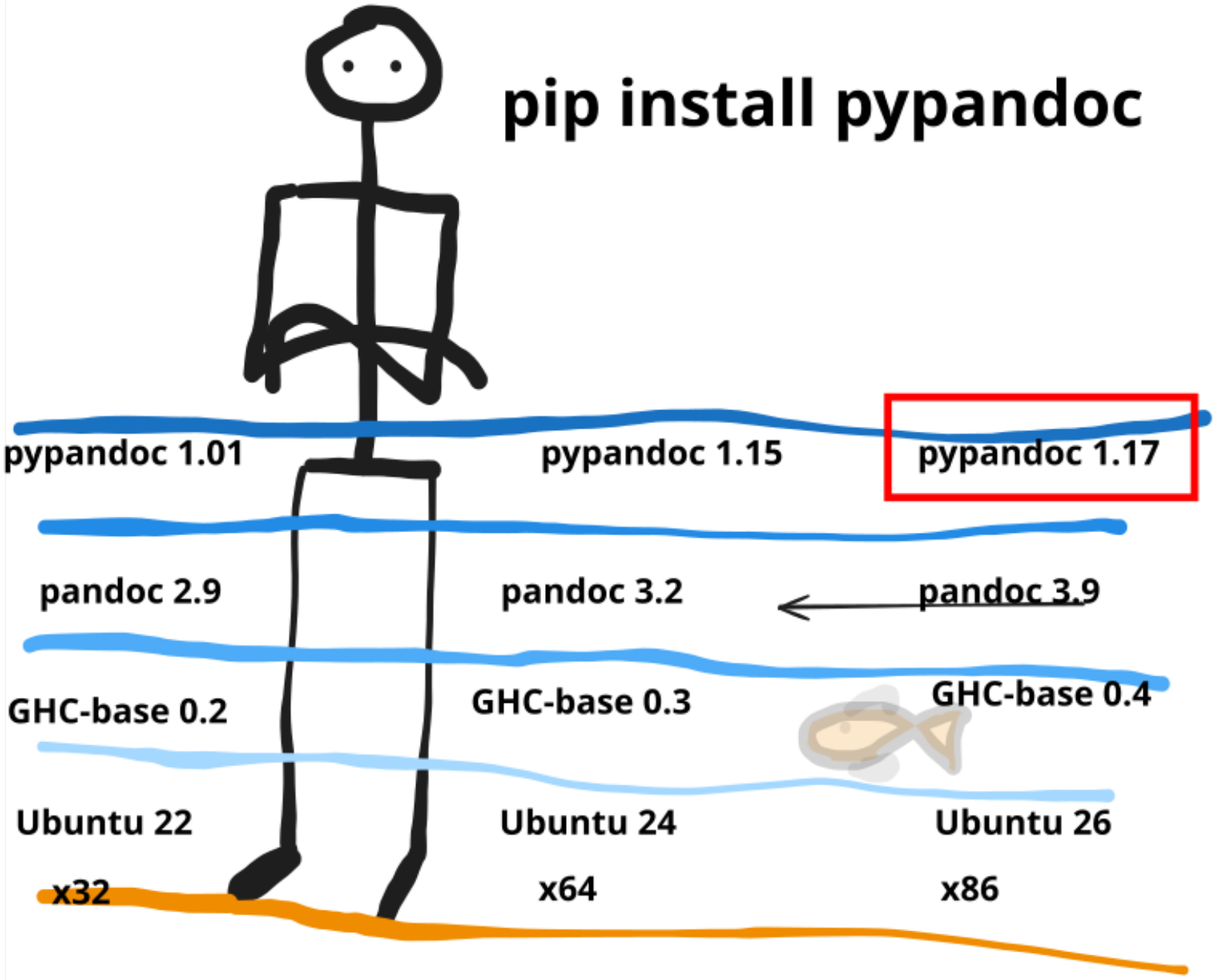
Speaker notes

a compiler is related to an operating system ; compiler needs to be adapted to the OS. Let’s say our OS is a GNU/Linux distribution : Ubuntu 24


pypandoc 1.01
pypandoc 1.15
pypandoc 1.17
pandoc 2.9
pandoc 3.2
pandoc 3.9
GHC-base 0.2
GHC-base 0.3
GHC-base 0.4
Ubuntu 22
Ubuntu 24
Ubuntu 26
x32
x64
x86

Speaker notes

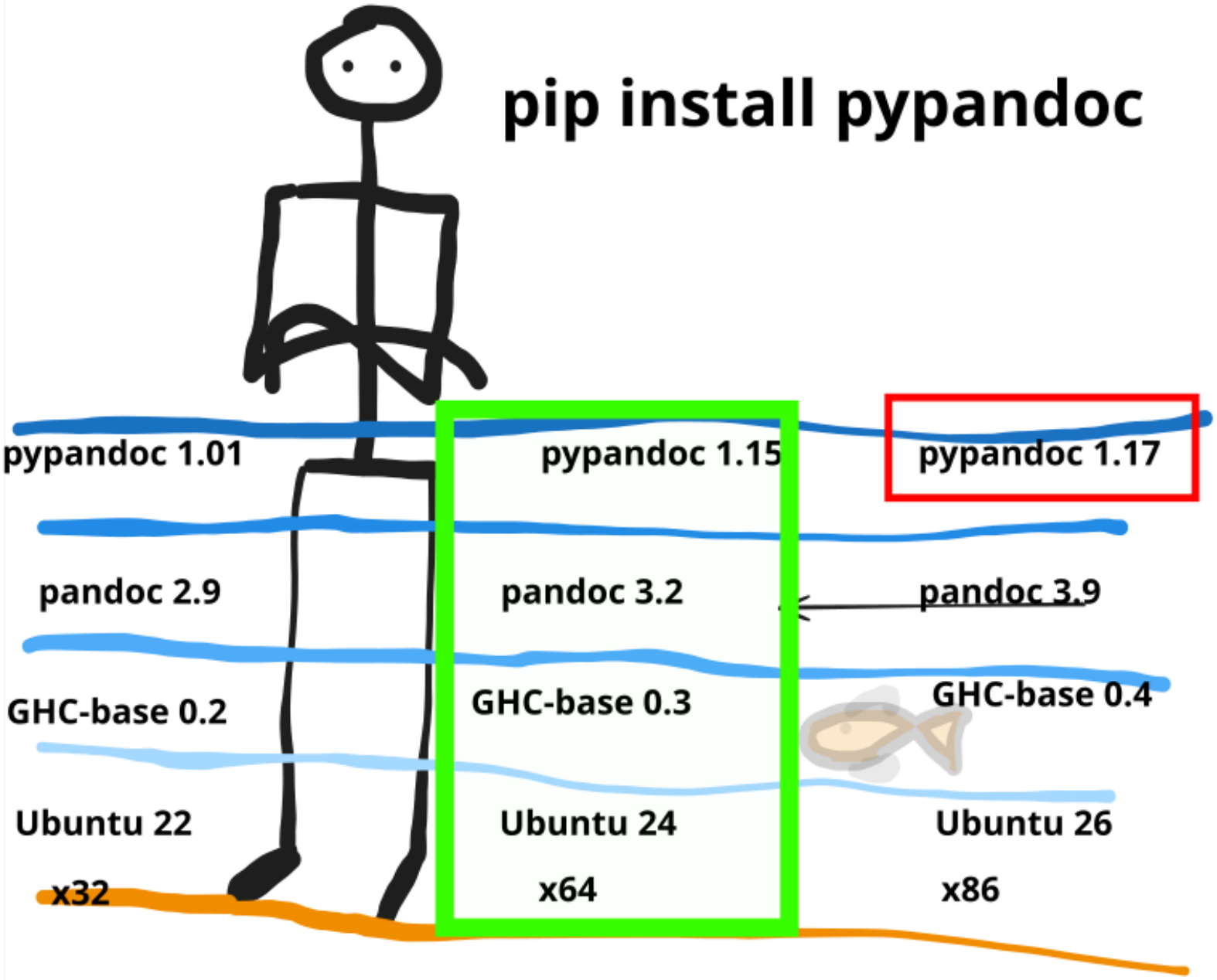
OS should be adapted to the machines capacities, especially to the processor. Not all GNU/Linux distros work with x86 or x32 bit processors.


pip install pypandoc
pypandoc 1.01
pypandoc 1.15
pypandoc 1.17
pandoc 2.9
pandoc 3.2
pandoc 3.9
GHC-base 0.2
GHC-base 0.3
GHC-base 0.4
Ubuntu 22
Ubuntu 24
Ubuntu 26
x32
x64
x86

Speaker notes

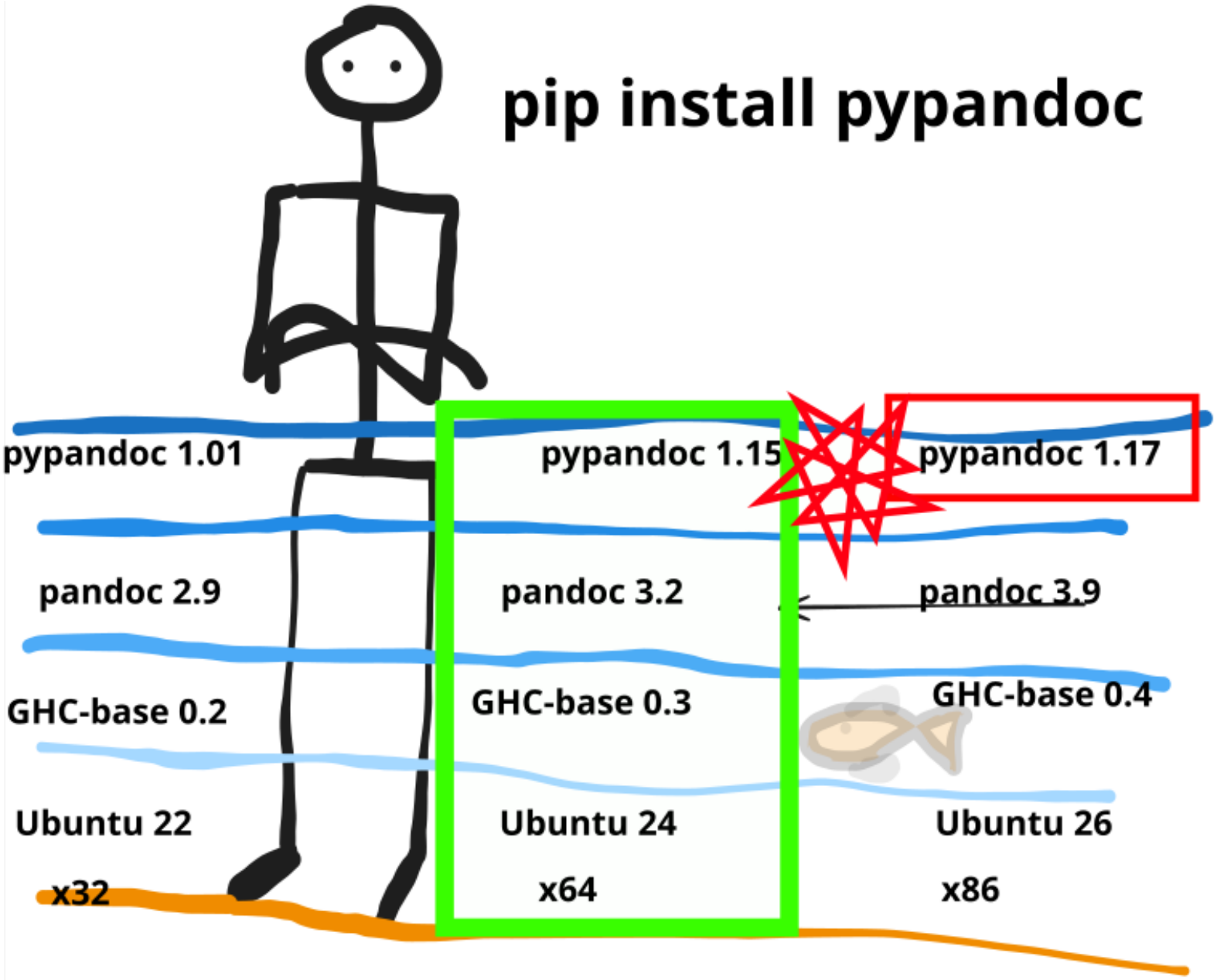
when you run your script, the command `pip install pypandoc` will install the latest version of Pypandoc ; in our example it's pypandoc 1.17

as Konrad Hinsen uses to say, there's what we call the **program** : **the code we care about**, the one that says "pip install pandoc" and the **environment**, which means the code we do not care about : all the software stack and the runtime. However this code is crucial to reproducibilty


pip install pypandoc
pypandoc 1.01
pypandoc 1.15
pypandoc 1.17
pandoc 2.9
pandoc 3.2
pandoc 3.9
GHC-base 0.2
GHC-base 0.3
GHC-base 0.4
Ubuntu 22
Ubuntu 24
Ubuntu 26
x32
x64
x86

Speaker notes

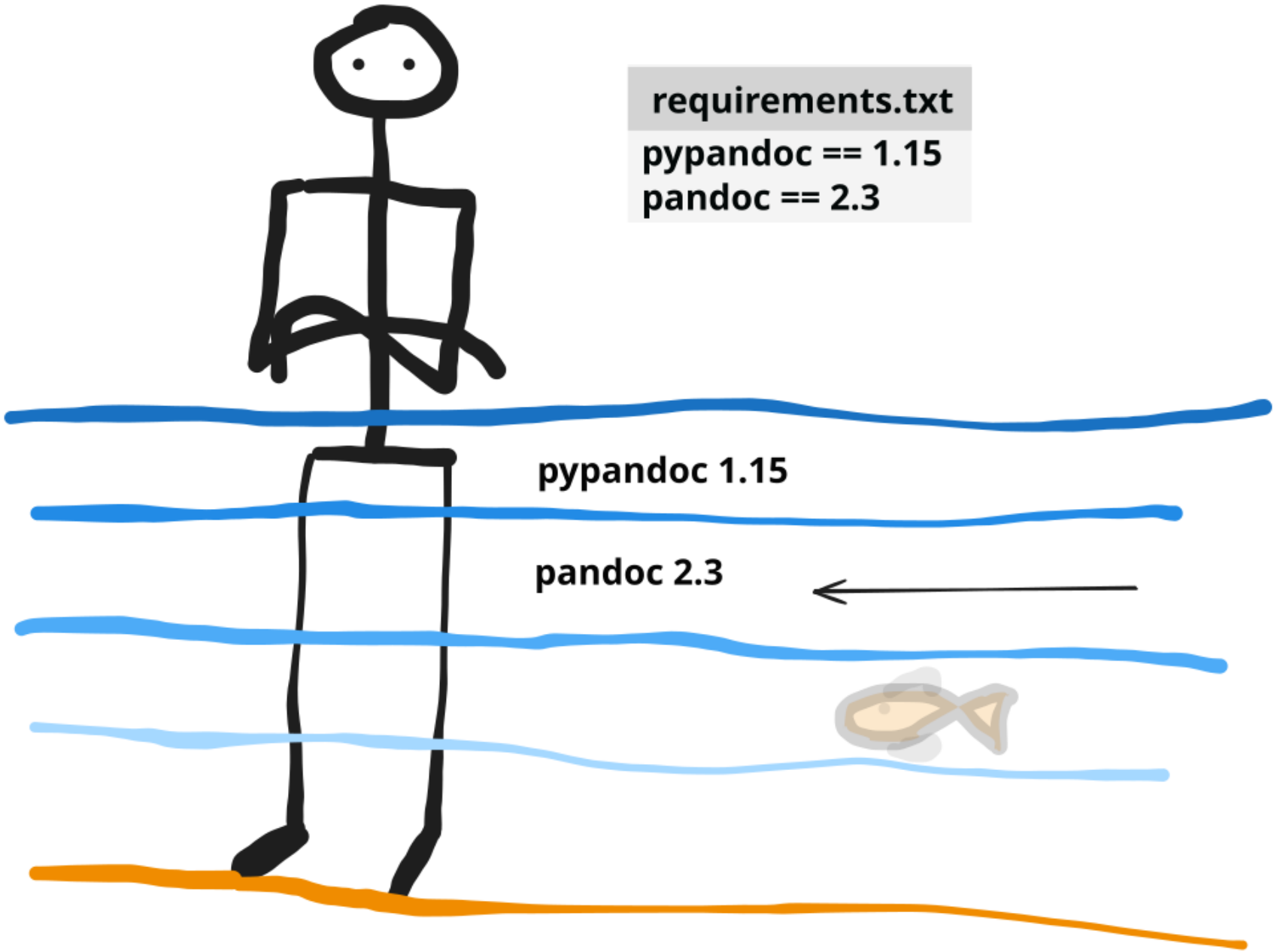
you have noticed that your script worked fine on your ubuntu 24 version, with the pypandoc version 1.15 will it still be true when we are going to run the code a few months later? with pip install pypandoc with no version pinned, the latest version (here the 1.17) will be installed automatically


pip install pypandoc
pypandoc 1.01
pypandoc 1.15
pypandoc 1.17
pandoc 2.9
pandoc 3.2
pandoc 3.9
GHC-base 0.2
GHC-base 0.3
GHC-base 0.4
Ubuntu 22
Ubuntu 24
Ubuntu 26
x32
x64
x86

Speaker notes

alas, quite often, the latest version of a Pypi library is nos operable with the underlying layers of the computational environments or with other python dependencies. there’s a bug ; you will need to uninstall 1.17, reinstall 1.15 how could you transfer the information to people interested in your software?

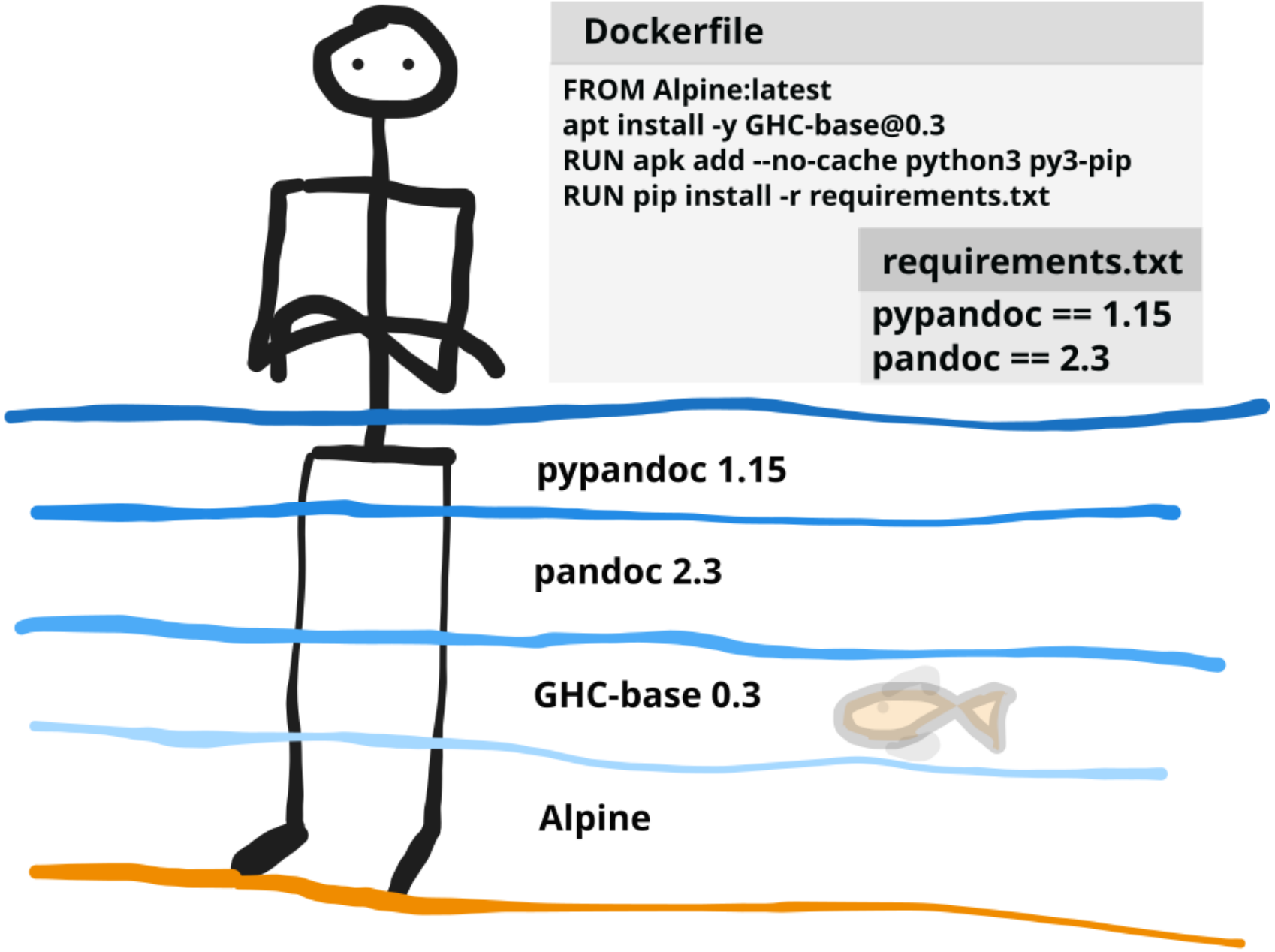
that’s why pinning your version could be usefull sometime while it can cause issues if your system itself has been updated but not the software stack.


requirements.txt
pypandoc == 1.15
pandoc == 2.3
pypandoc 1.15
pandoc 2.3

Speaker notes

you may use virtual environments to make snapshots of the libraries you use and then add this information to your code ; you need to generate a config file named requirements.txt this file will be made public along with the source code so that other people may use it to install the requested libraries in the proper versions.

this will do for the machines who have already GHC installed, but some system-dependencies may be requested that cannot be listed in the requirements text file because they are not python libraries


Dockerfile
FROM Alpine:latest
apt install -y GHC-base@0.3
RUN apk add --no-cache python3 py3-pip
RUN pip install -r requirements.txt
requirements.txt
pypandoc == 1.15
pandoc == 2.3
pypandoc 1.15
pandoc 2.3
GHC-base 0.3
Alpine

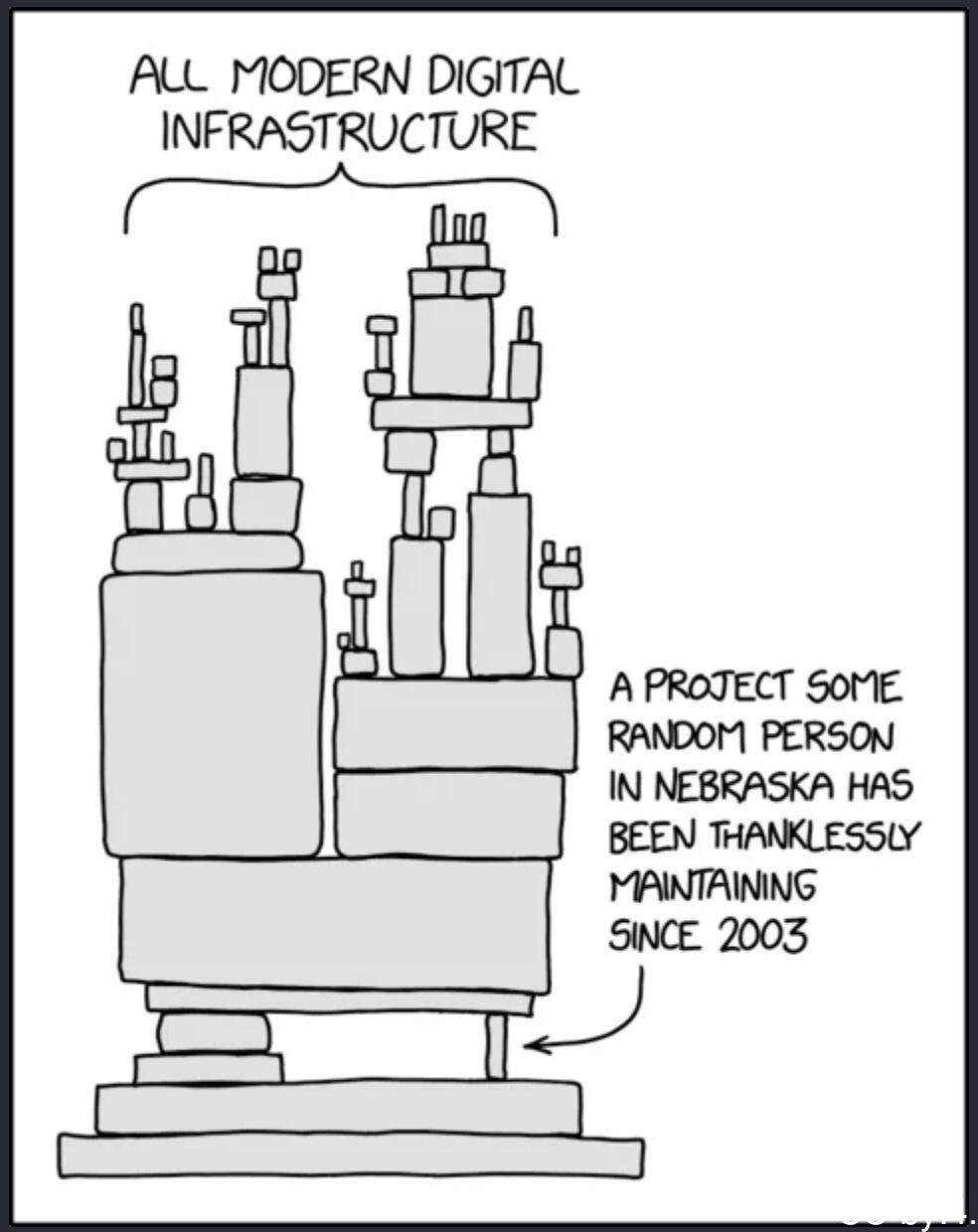
Speaker notes

a more comprehensive way of doing things would be to make a complete snapshot of all the needed Python dependencies along with the system dependencies. this new config file, known as a Dockerfile, will make it possible to run an image of the script based on a complete environment within a container. Everyone can download the container and run it with docker installed on their machine. Or if the dockerfile only was shared (because a docker image is heavy), you might use it to build your own image of the code.

below are described the main parts of a Dockerfile:

you will need an OS image to run this code ; Ubuntu is two heavy, Alpine weight much less and will drastically reduce the compilation time. So once an Alpine OS is pulled, you will add the different layers : first the compiler for Haskell (line 2) then Python itself, and the pip package manager(line 3) finally you will ask the system to install the python dependencies called by the script (line 4) this config file is oversimplified to show how it works basically I this machinery is quite complex and requires a deep understanding of how Docker builds images. besides the image pulled itself is not reproducible. Outputs may vary on the long run. Docker is better to deploy a code on different machines at the same time than to build reproducible source code on a long period of time. There must be something simpler to solve the reproducibility challenge

when older versions of dependencies disappear



- they might disappear from the language repositories (Pypi)
- they can also disappear from the forges where they were stored
- and new versions appear to be non compatible with other components (such as in our example)

Guix as a package manager offers solutions to those prooblems

- Guix provides fully consistent *computational environments* in only two config files (*manifest* and *channels*)
- these config files track all packages used with their own version and the dependencies to these packages
- if a dependency cannot be found online anymore, a fall-back system links missing components to their archived counterparts in Software Heritage (1)
- Guix can be used on Windows through WSL2
- guix allows to run on a single computer a script with different versions of the same package without any conflict
- for non-guix users, guix easily packages the software into a Docker container (2)

1. Courtès et al. (2024)
2. Tournier (2024)

example

```
(define-module (r-gristapi)
  #:use-module ((guix licenses) #:prefix license:)
  #:use-module (guix packages)
  #:use-module (guix build-system gnu) ;; pertinent ici ?
  #:use-module (gnu packages cran)  ;; pour installer les dépendances indiquées dans propagated-inputs
  #:use-module (gnu packages statistics)
  #:use-module (gnu packages geo)  ;; spécifique pour sf ?
  #:use-module (guix download) ;; pour télécharger un fichier depuis github
  #:use-module (guix build-system r)) ;; pour compiler des packages de R ?

(define-public r-gristapi
 (package
  (name "r-gristapi")
  (version "0.1.0")
  (source (origin
        (method url-fetch)
        (uri (string-append "https://github.com/spyrales/gristapi/archive/refs/tags/v0.1.0.tar.gz"))
        (sha256
         (base32
          "01jljdsj4qn7rjaxw3pvyyrw10r3zydf6v904lzvw8dbkzlj26i1"))))
  (build-system r-build-system)
  (propagated-inputs
   (list r-assertthat r-dplyr r-httr2 r-jsonlite r-magrittr r-r-methodss3 r-purrr r-r6 r-rlang r-sf r-tibble r-tidyr r-tidyselect))
  (synopsis "gristapi communicates from R with a Grist document using Grist api ")
  (description
   "The package provides an R6 class to connect to a Grist document (which contains multiple tables), and a set of functions to interact with the document.")
  (home-page "https://github.com/spyrales/gristapi")
  (license license:eupl1.2)))

r-gristapi
```

how guix should build it

under which name this package will be found in guix package manager

where are the sources that should be built, which unique version

which dependencies should be used to successfully build the source code

Speaker notes

Here is the description we have written for an Rpackage which communicates with the collaborative spreadsheets manager Grist. This description is written in scheme, the language used by guix. it contains all the information required to build from sources the package with R.

a package definition is written is scheme and contains details about how to compile a package, from which sources, which versions of the sources should be compiled, and which dependencies are needed in the compilation.

This is how Guix allows us to pin packages versions in a given source code

Guix users need only two config files to reproduce the environment or a past experience :

- **manifest.scm** : enumerate all the dependencies (system and software)
- **channels.scm** : defines their exact version

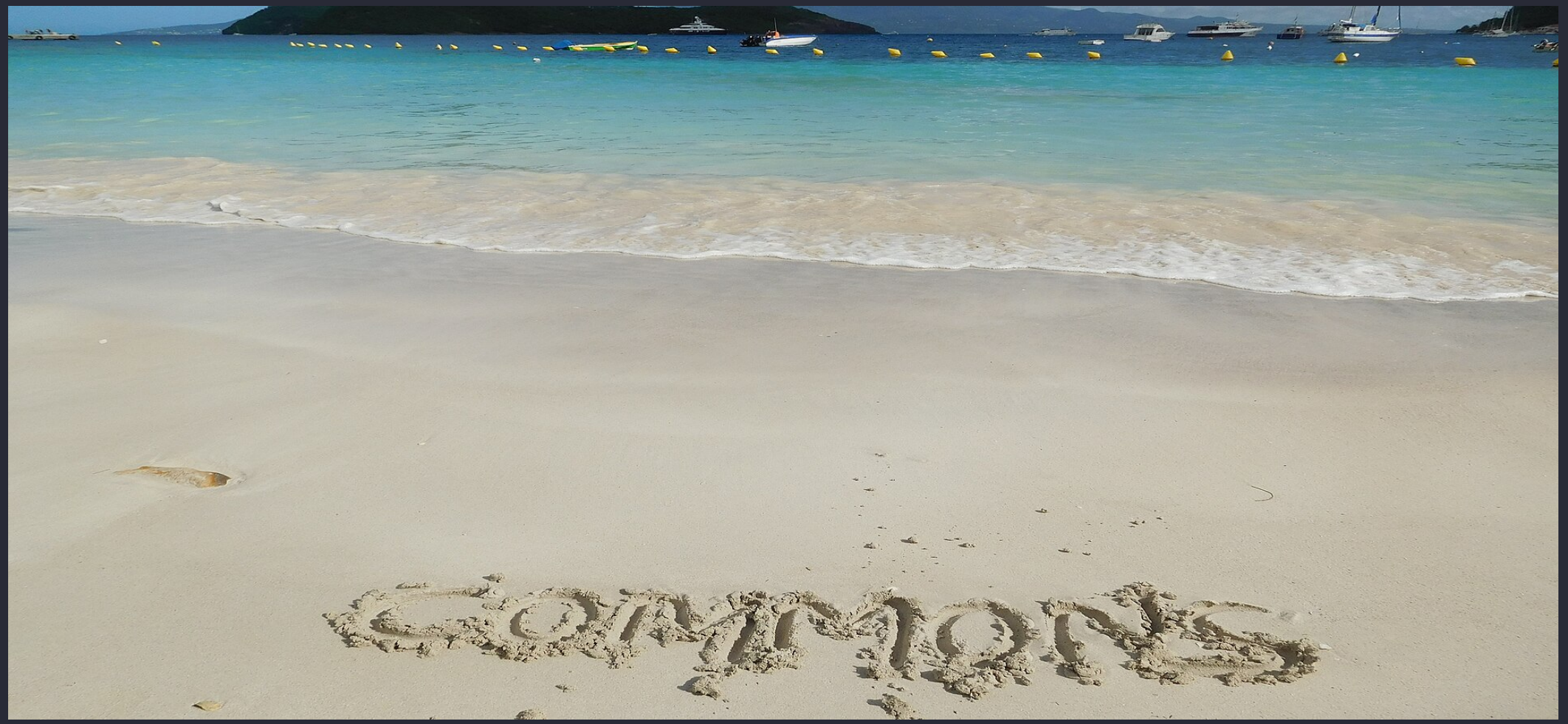
```
1 guix time-machine --channels=channels.scm -- shell --manifest=manifest.scm
2 -- Rscript myscript.R
```

Speaker notes

The command on this slide recreates a full environment from a list of **guix channels** (=git repositories containing guix definitions of usefull packages) and a **manifest** (i.e a list of packages used by the myscript.R file ; guix opens an isolated shell and runs the script in that exact environment.

written in scheme language as well, these two files may be generated from guix commands such as `guix package --export-manifest > manifest.scm`

Thanks to Guix maintainers, writing code feels a bit less like “writing in the sand”



references

Courtès, L., Sample, T., Tournier, S., & Zacchiroli, S. (2024, June). Source code archiving to the rescue of reproducible deployment. *Proceedings of the 2024 ACM Conference on Reproducibility and Replicability, Proceedings of the 2024 ACM Conference on Reproducibility and Replicability*. https://doi.org/10.1145/3641525.3663622

Fobbe, S. (2024). *The Limits of Reproducibility with Rocker Docker Images for R* (posts). https://seanfobbe.com/posts/2024-12-16_reproducibility-limits-rocker-docker-images-r/.

Tournier, S. (2024, December). (Re)déploiement de conteneurs et machines virtuelles avec guix. *JRES (Journées Réseaux de l'enseignement Et de La Recherche) 2024*.