Supporting Long-term Reproducible Software Execution

Luís Oliveira, **David Wilkinson**, Daniel Mossé, Bruce Childers
Computer Science Department – University of Pittsburgh
{loliveira,dwilk}@cs.pitt.edu

Motivation

Reproducibility crisis

- Scientists have difficulties in reproducing results

Current solutions aim to fix replicability

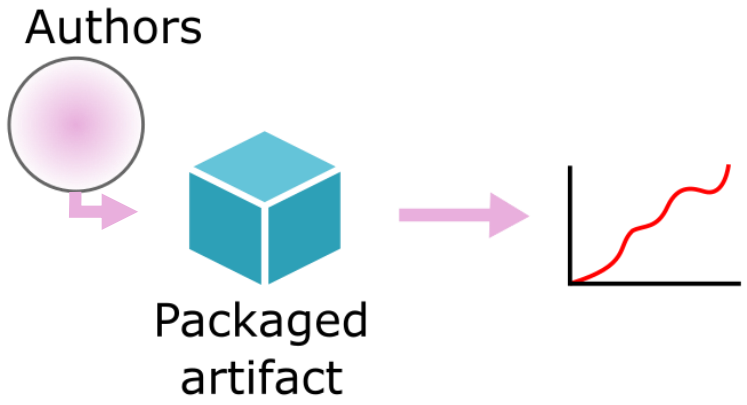
- What can be done to tackle reproducibility in the long-run?

Repeatable, Replicable and Reproducible

There is some ambiguity in these terms

- Repeatable – Same team, using own software
- Replicable – Different team, using original software
- Reproducible – Different team using independent implementation

Entry-level: Repeatability

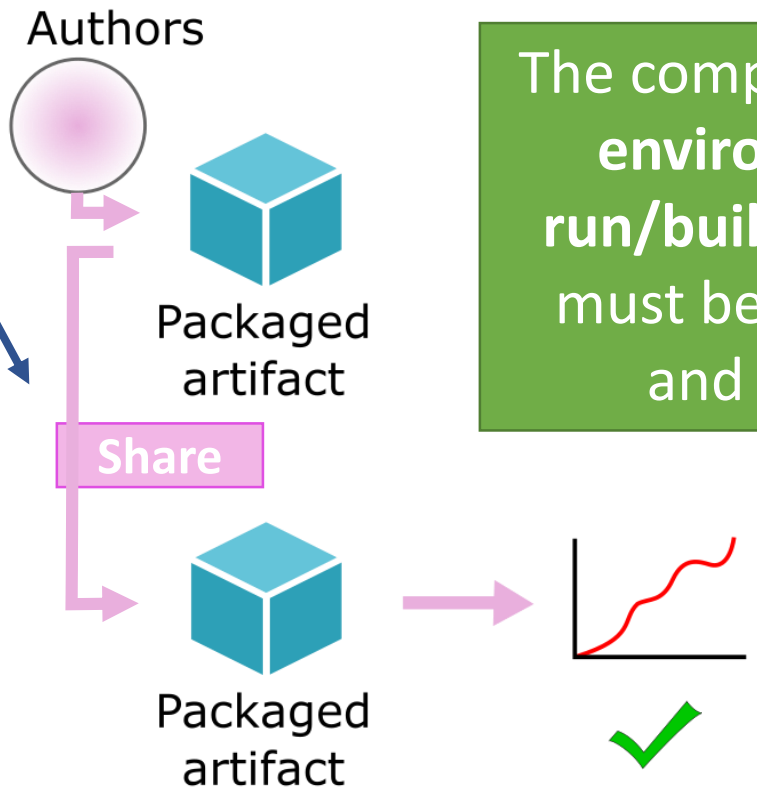


Packaging the software together with run/build instructions suffices

As long as the **execution environment** is not modified!

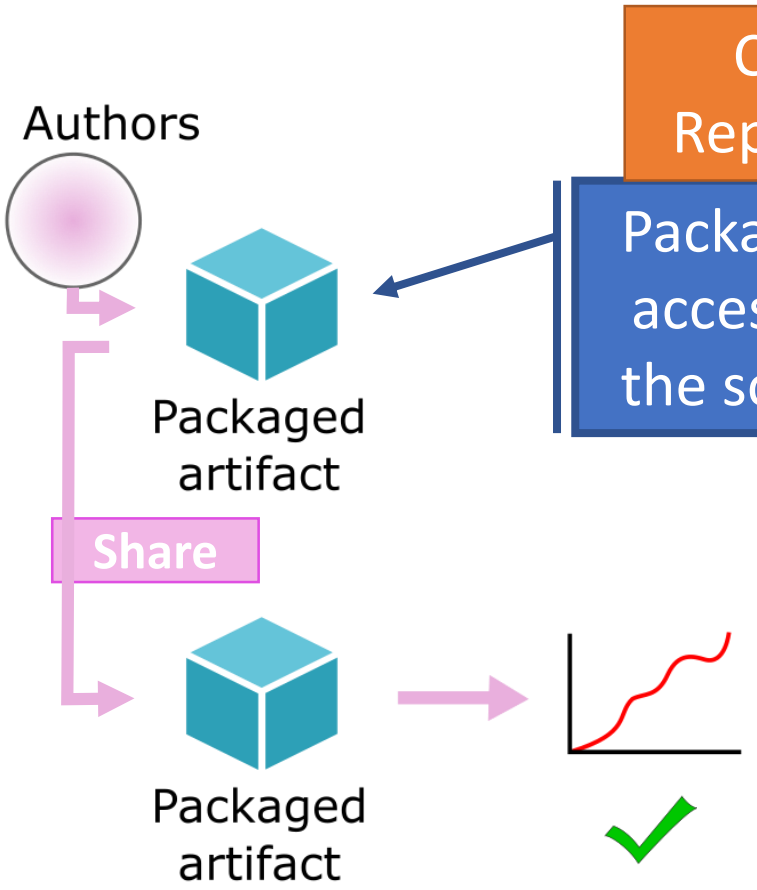
Sharing software: Replicability

Sharing the software is not enough



The complete **execution environment**, and **run/build instructions** must be documented and replicated

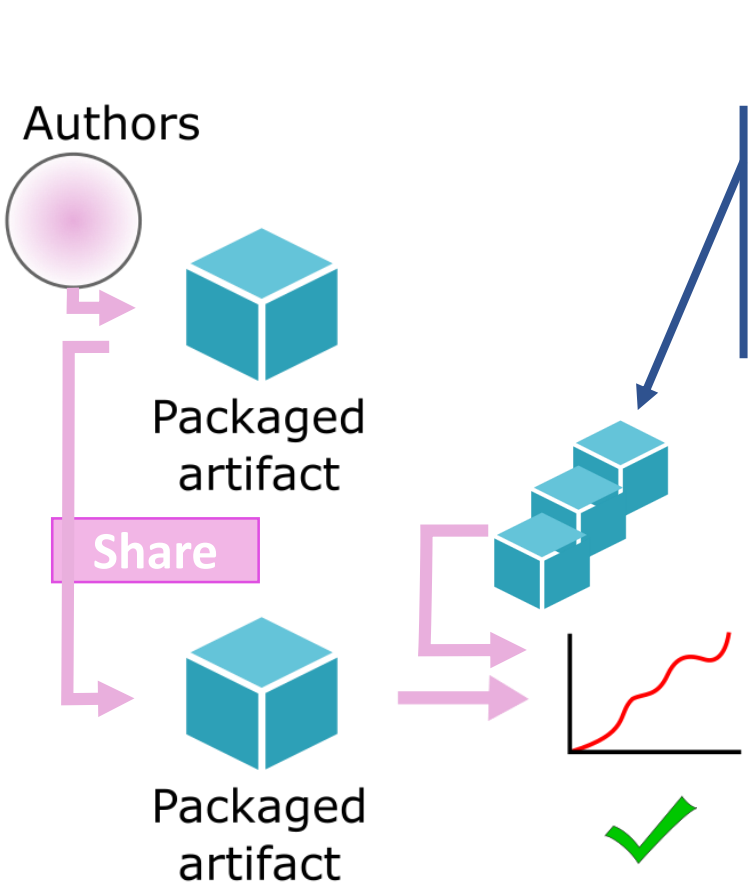
Sharing software: Replicability

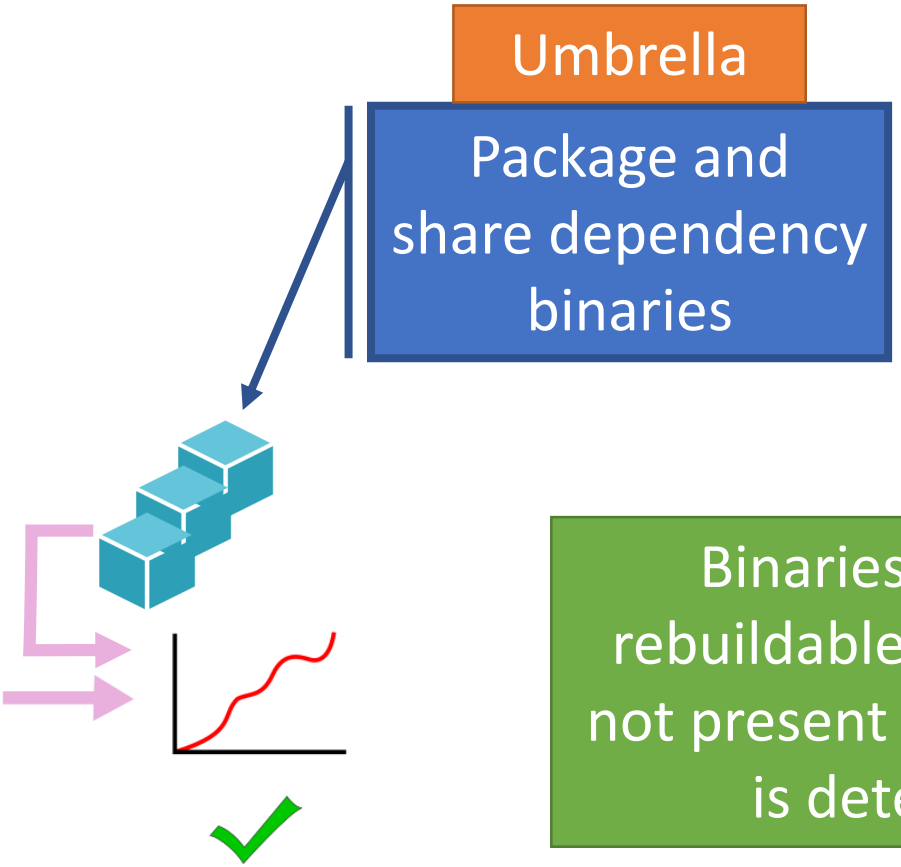


Environment is only partially packaged

Problematic to run different experiments (e.g. other datasets)

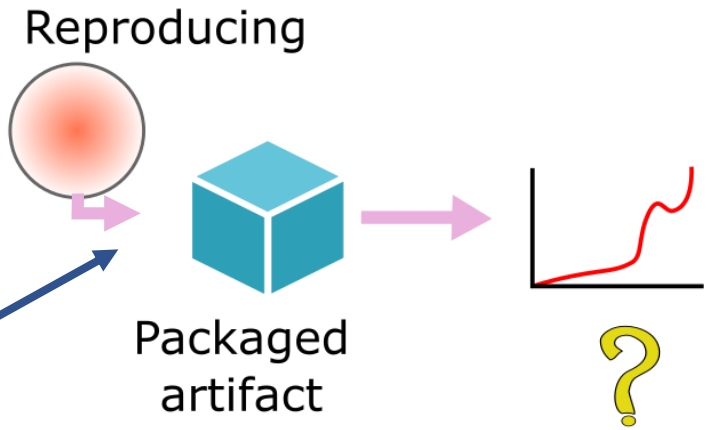
Sharing software: Replicability





Binaries are not rebuildable if source is not present (e.g. if a bug is detected)

Final objective: Reproducibility



Independently developed

Reproducing the **execution environment**, and **run/build instructions**

But what if (when) the results are not reproduced?

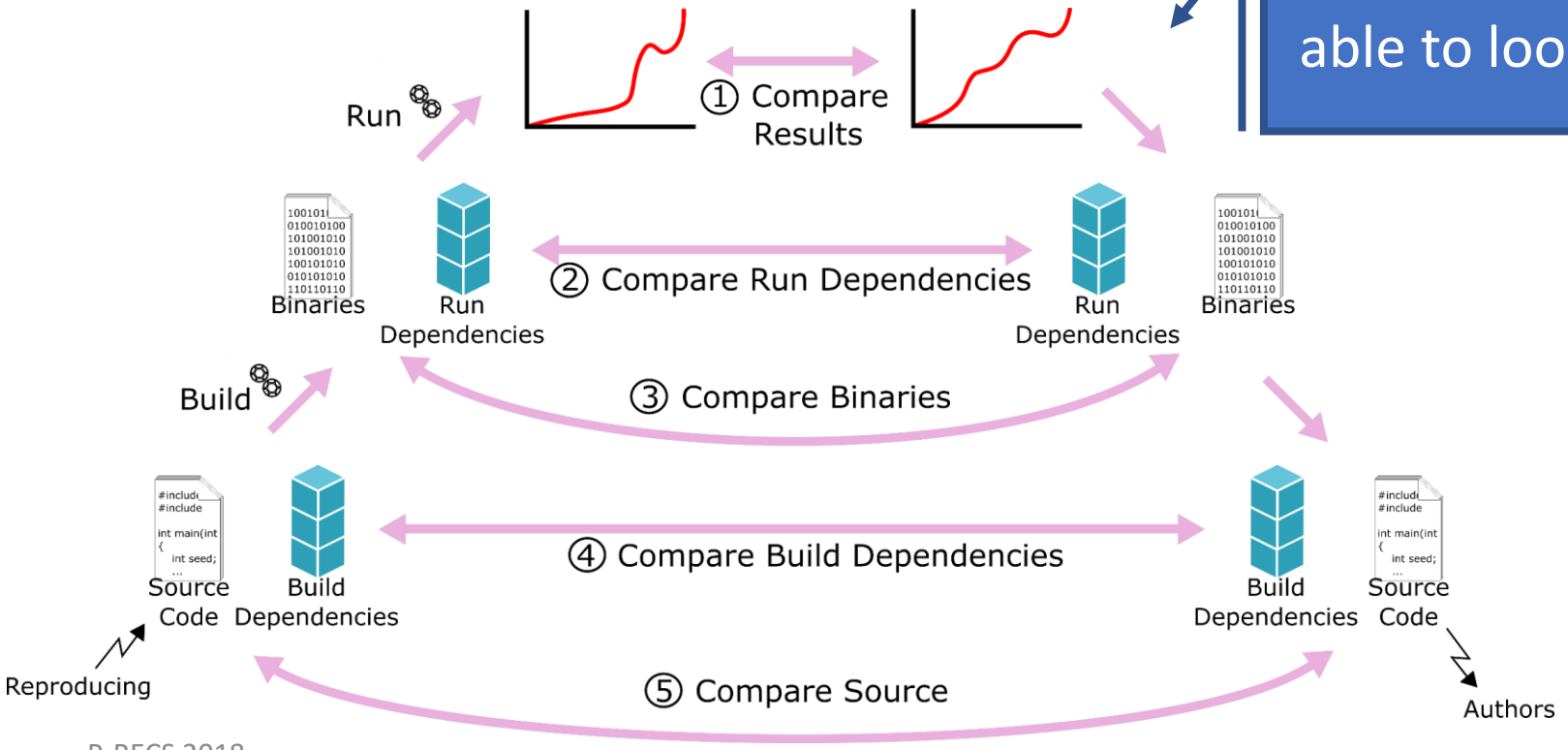
Where is the error?

In the original artifact? In one of its dependencies? In the reproduction?

Where is the error?

In the original artifact? In one of its dependencies? In the reproduction?

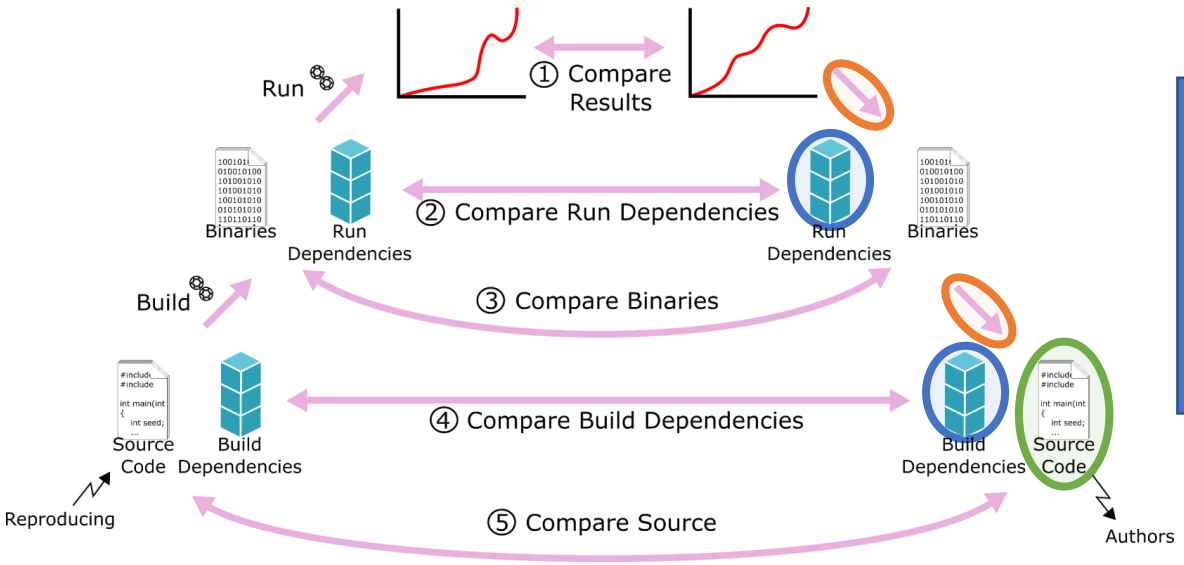
We need to be able to look for it



Where is the error?

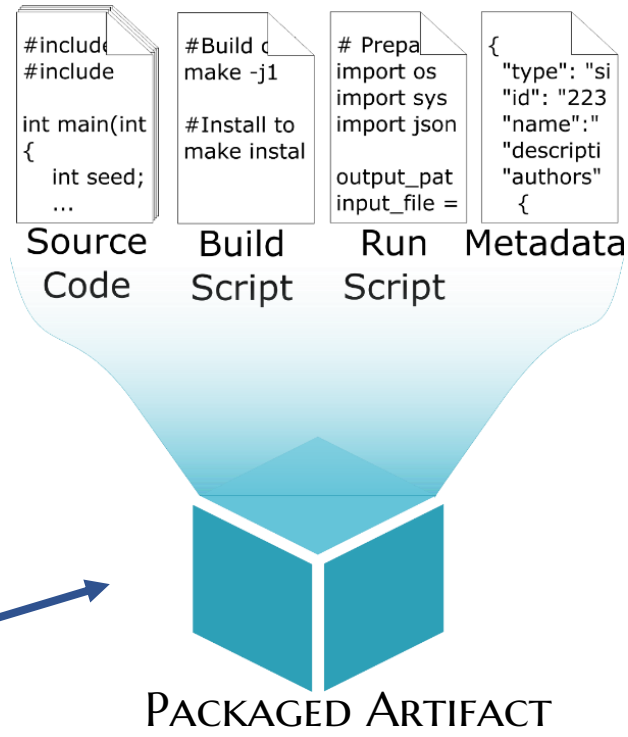
Transparency:
Track, preserve, and inspect **all** software

Source code required to find and correct errors. Making the software useful again.



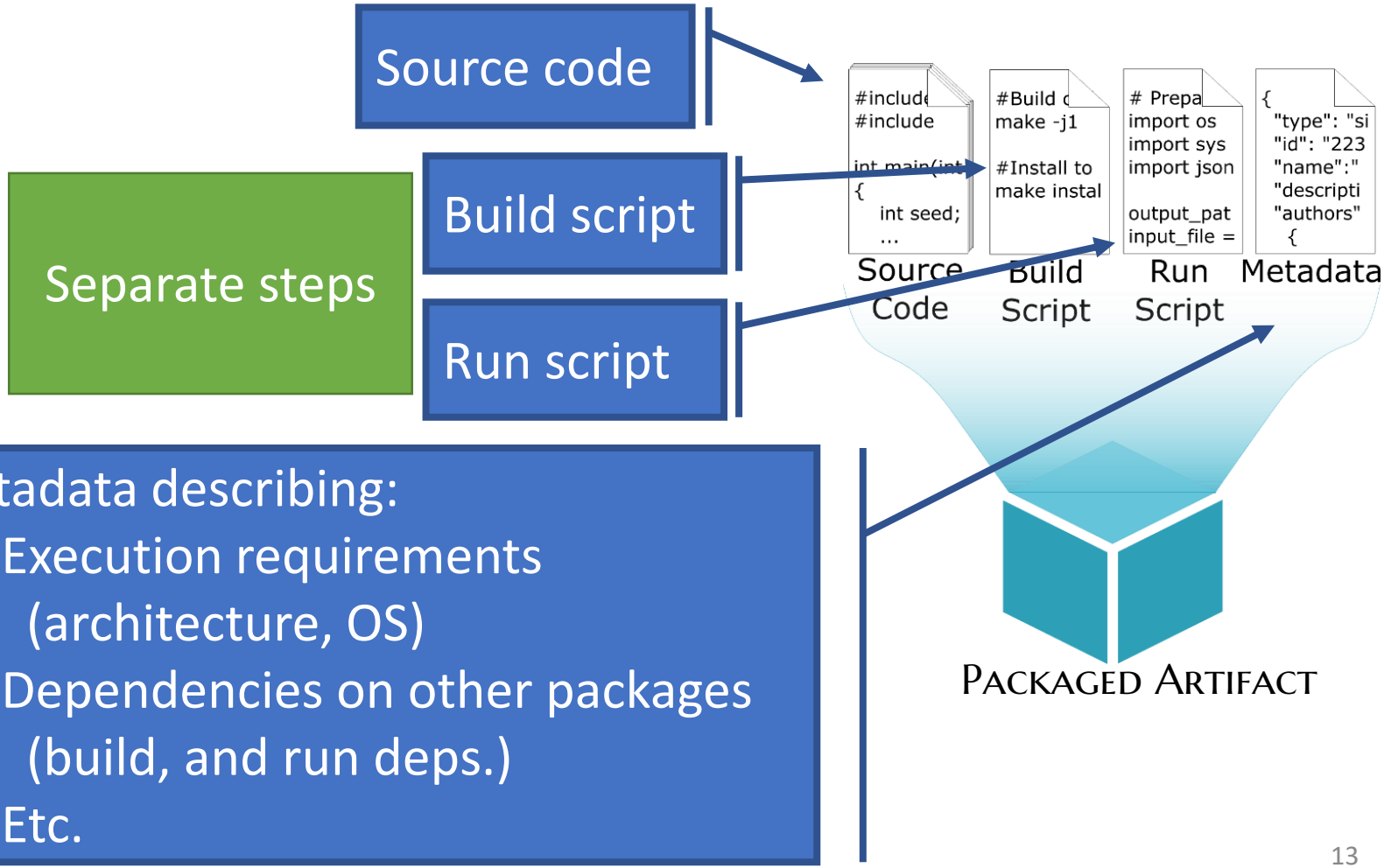
All dependencies and respective **source code** accessible

Supporting long term reproducibility with Occam



Authors put their software in an Occam package

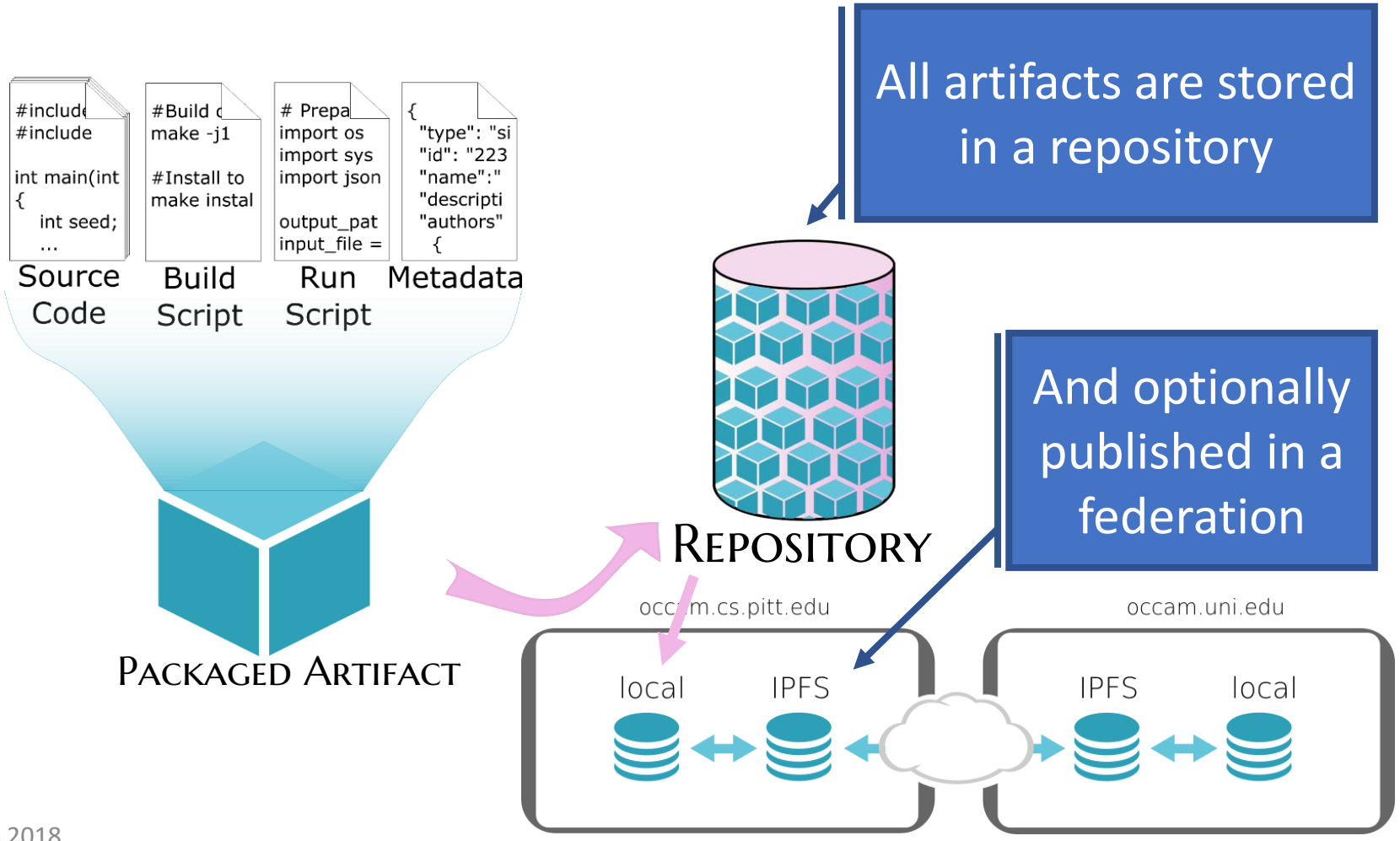
Supporting long term reproducibility with Occam



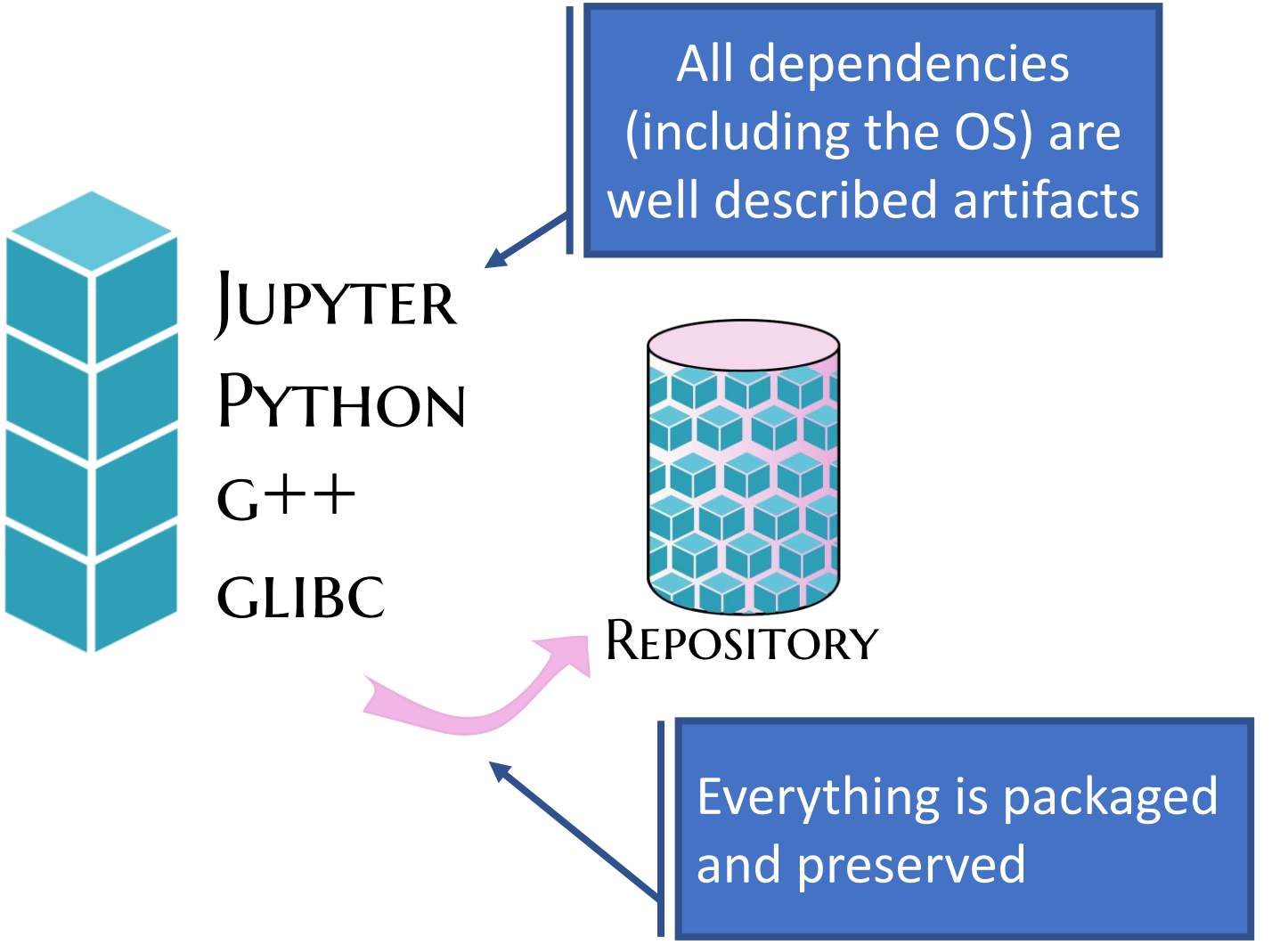
Metadata describing:

- Execution requirements (architecture, OS)
- Dependencies on other packages (build, and run deps.)
- Etc.

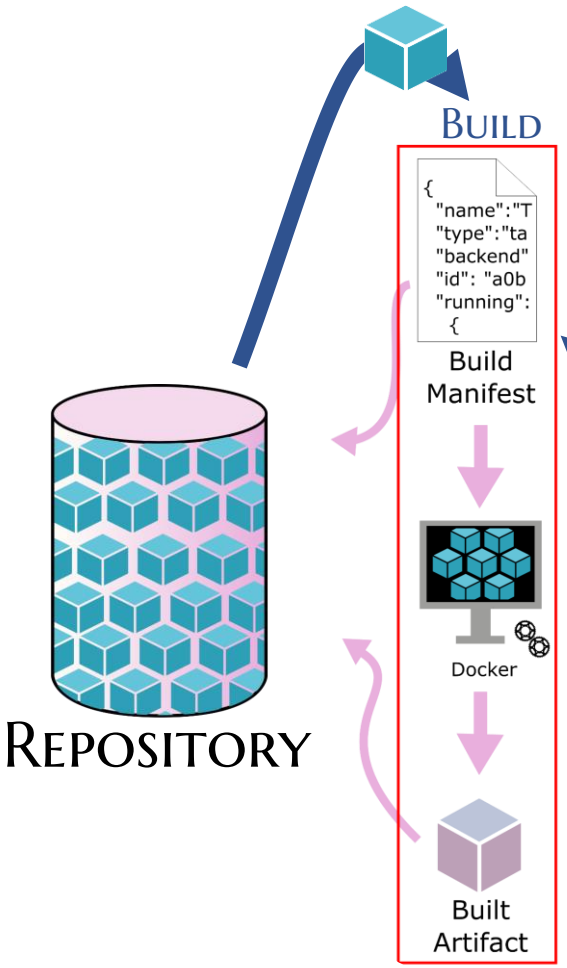
Supporting long term reproducibility with Occam



Dependencies are first-class



Repeatable execution environments

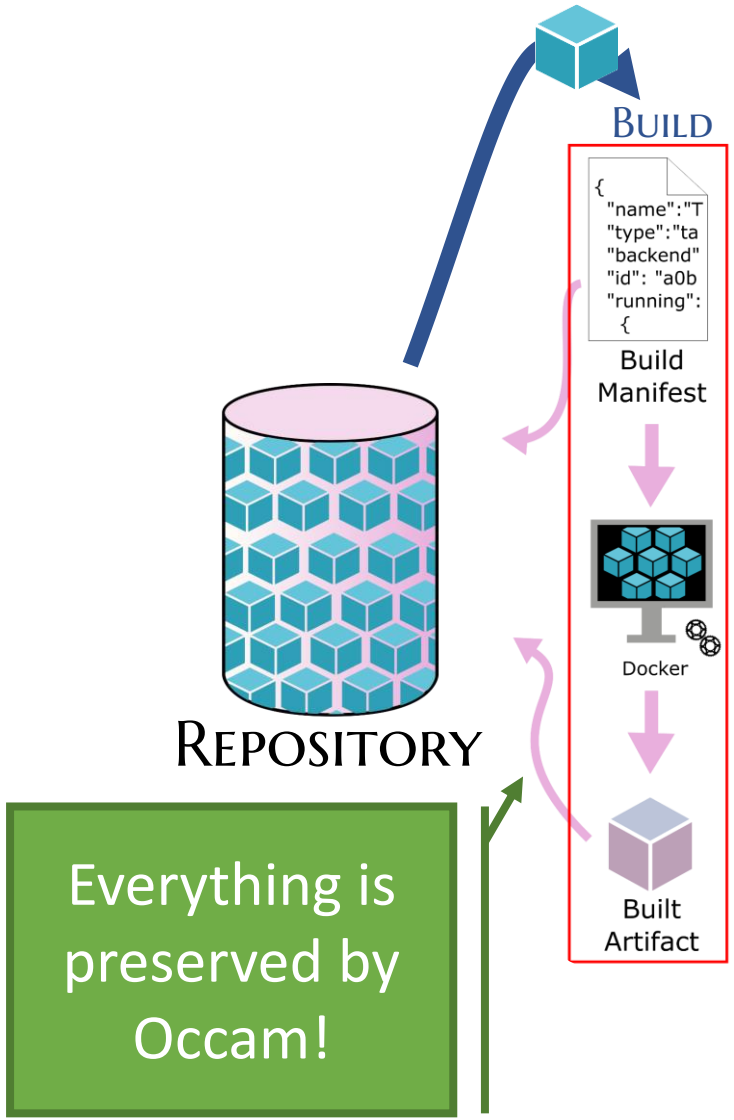


Software can be **automatically built** using the instructions on the metadata.

Manifests contain a digest of the software (main artifact and its dependencies).

Automatically generated!

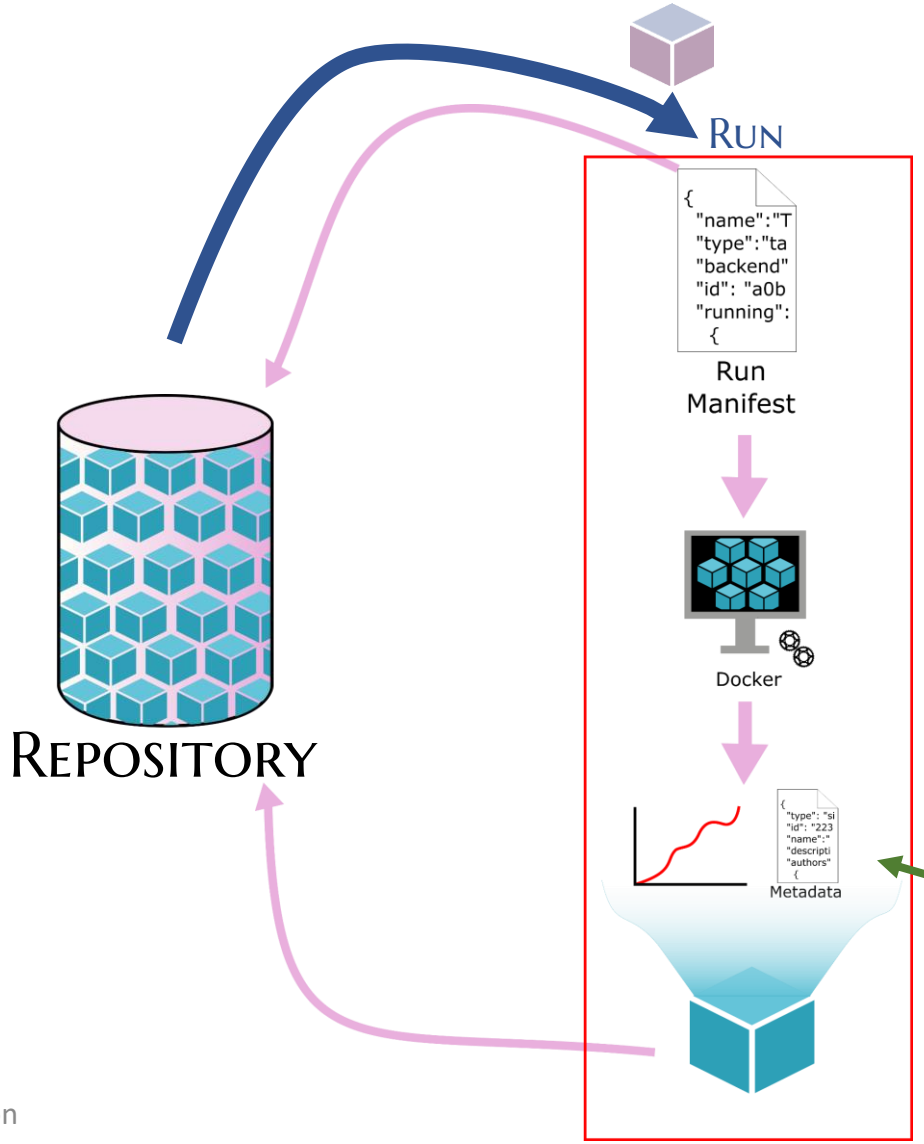
Repeatable execution environments



Replicable Execution Environments are **created on-demand** (e.g. on Docker).

Everything is preserved by Occam!

Repeatable execution environments



Results are packed with metadata that track their **lineage and provenance**.

Final thoughts

Need feedback from the community:

{loliveira,dwilk}@cs.pitt.edu

Current Occam implementation:

- Preserve as much as possible
  - Prevent silent loss of fidelity
  - Improve the longevity of software
- Preserving source code is vital
  - And the ability to build/run it.
- Dependencies are important
  - They may be the source of errors