Presentation on

DataLad

Distributed Data Management System

Presented by:
Mayra Elwes, Bhanu Prasanna Koppolu

Institute for Biomedical Informatics (BI-K)
Uniklinik Köln

Slides & Code on: ***GitHub***

Objectives:

1. DataLad Introduction
2. Version Control - Git, Git-Annex, and Why?
3. YODA Framework
4. DataLad Datasets
5. Transitioning into DataLad
6. Live Code Demo
7. Reproducibility

Acknowledgments

1. Joey Hess (Git-Annex)
2. The DataLad Team

SPONSORED BY THE

Federal Ministry of Education and Research

01GQ1112
01GQ1411

germany-usa

NSF
1129855
1429999

Human Brain Project

EUROPEAN UNION

VirtualBrainCloud

JÜLICH Forschungszentrum

hhu Heinrich Heine Universität Düsseldorf

DARTMOUTH
VOX CLAMANTIS IN DESERTO
1769

NIH

cbbs
center for behavioral brain sciences

SACHSEN-ANHALT
Ministerium für Wissenschaft und Wirtschaft

OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG

DataLad Introduction

DataLad is a free and open-source Distributed Data Management System.

It manages both the code and data simultaneously.

DataLad Dataset is just a Git repository. It is completely Domain Agnostic.

Uses both git and git-annex for version control of very large datasets and files.

Tracks and helps produce Reproducible Data Analysis.

There is a DataLad Handbook available in Web Format on: Handbook, Documentation.

Why DataLad?

Common Challenges in Scientific Research 🧐

Forgetting which parameters yielded the best results.

Losing work or data due to inadequate backup and versioning.

Inability to quickly reproduce results with new or updated data.

Challenges in coordinating work among team members.

Lack of detailed records of data processing steps and analysis history.

How DataLad helps solve them 😀

Combines Git and Git-annex to track code and large datasets.

Keeps a complete history of data transformations and parameters used.

Manages large files seamlessly without bloating repositories.

Enables easy rerunning of analyses to reproduce results.

Simplifies sharing and integrating work from multiple contributors.

Local Version Control

- Ben's project requires automated tracking of:
  - **When** a file was last changed
  - **Where** a file came from
  - **What** input files were used to generate outputs
  - **Why** certain actions were taken
- Ben makes frequent changes to his analysis scripts.
- He values the ability to return to a previously recorded state for future reference.

Distributed Version Control

- Ben's work is spread across multiple devices (laptop, desktop, remote server).
- He wants automatic and efficient synchronisation.

Collaboration requires:

- **Distributed synchronisation** with centralised storage
- **Preservation of origin and authorship**
- **Combining simultaneous contributions**

Data Management with DataLad 😀

Ben uses **local version control** for his own work and **distributed version control** for collaboration.

He works on subsets of data at any time:

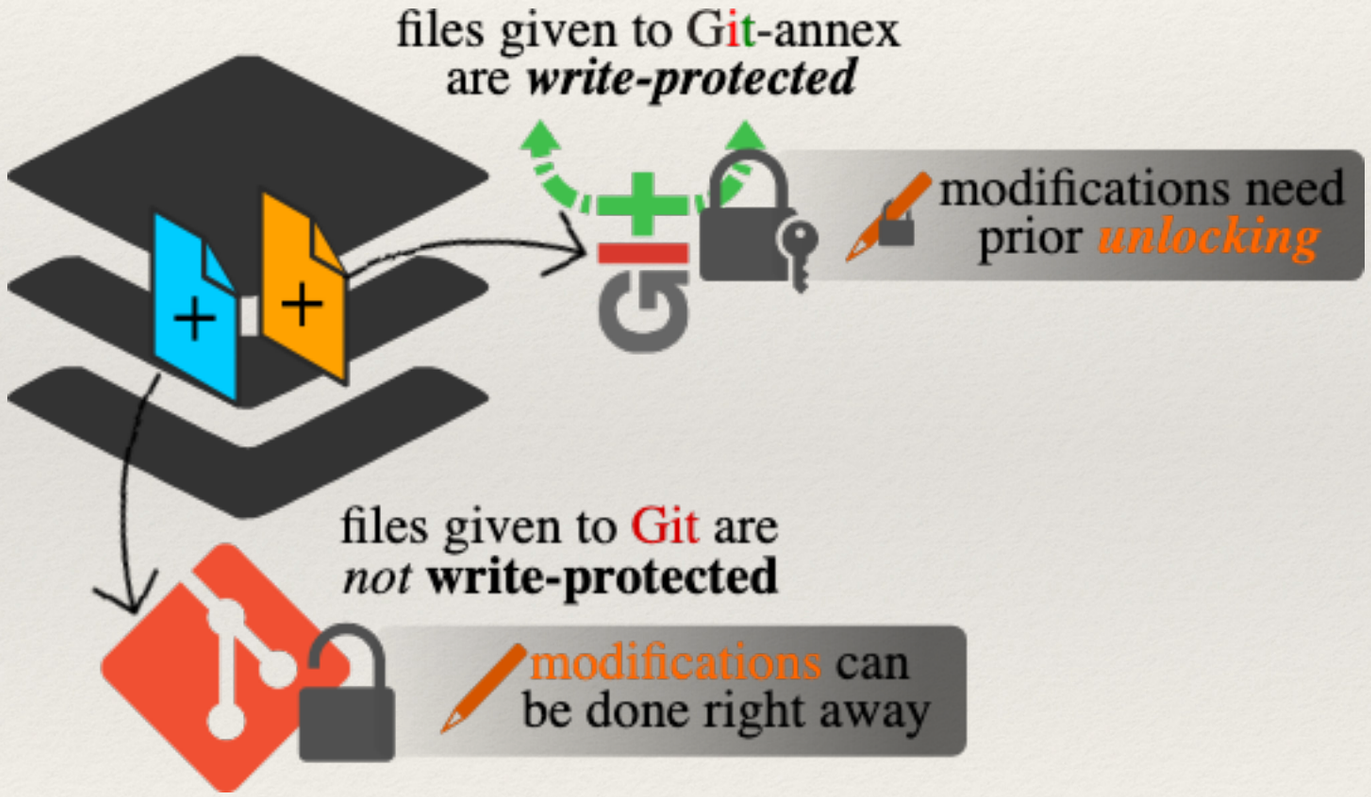
- All files stored on a server.
- A few files rotated in/out of his laptop.

He aims to **publish** the project's data (raw data, outputs, or both), either **completely or selectively** at the project's end.

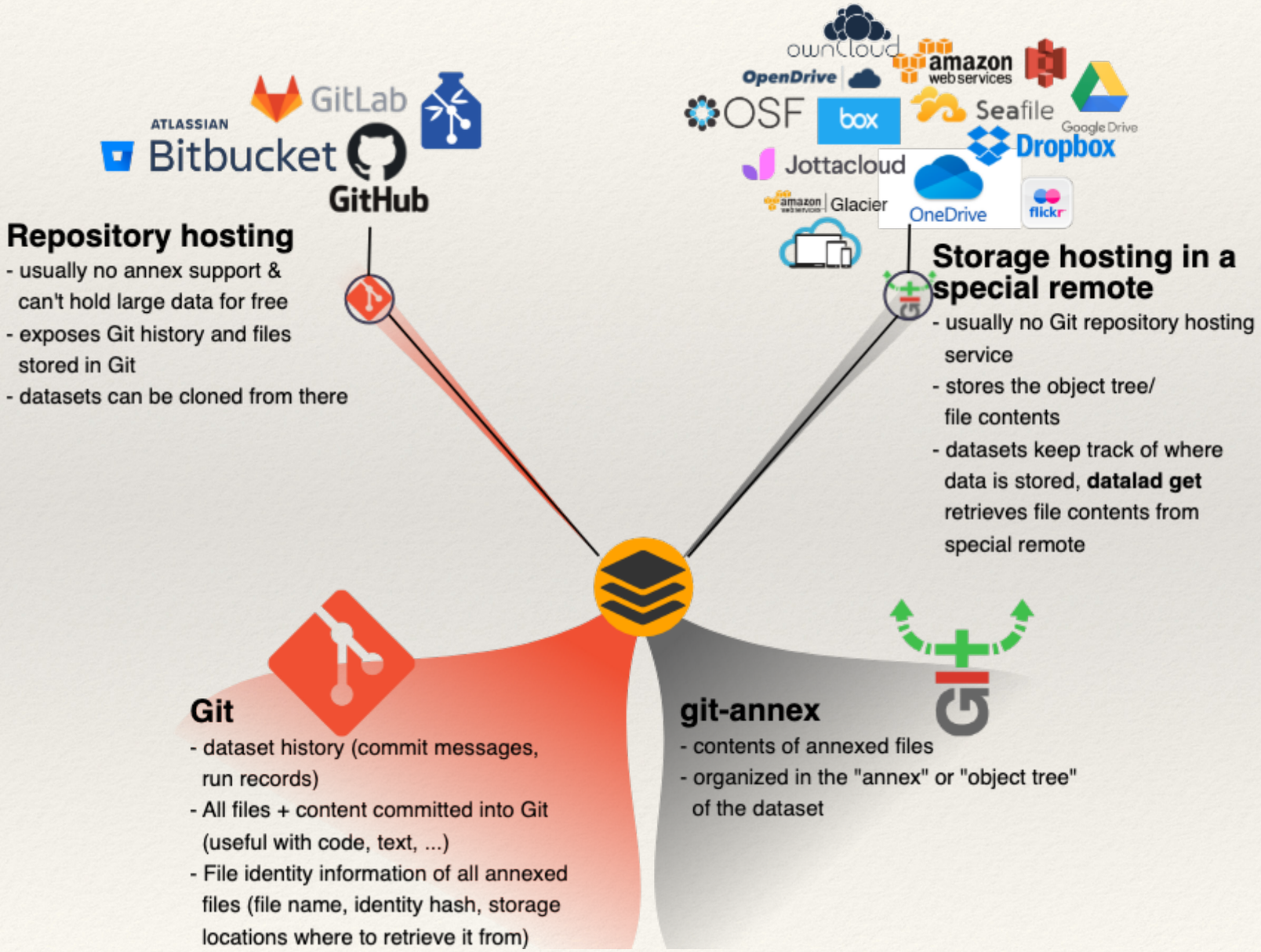
Git and Git-Annex

Git	Git-Annex
Handles Version control for code and small files.	Handles all large files and data efficiently.
File content are in the Git history and will be shared git/datalad push.	File contents are in annex. Not necessarily shared.
Synchronizes all files, including large ones.	Synchronizes only metadata; files are retrieved on-demand.
Files are always present locally.	Files may be stored remotely and retrieved on demand, saving local

Git and Git-Annex



Git and Git-Annex



Why Version Control?

DataLad controls both Datasets and files. DataLad creates a full log of changes made to the datasets by whom, when, and how.

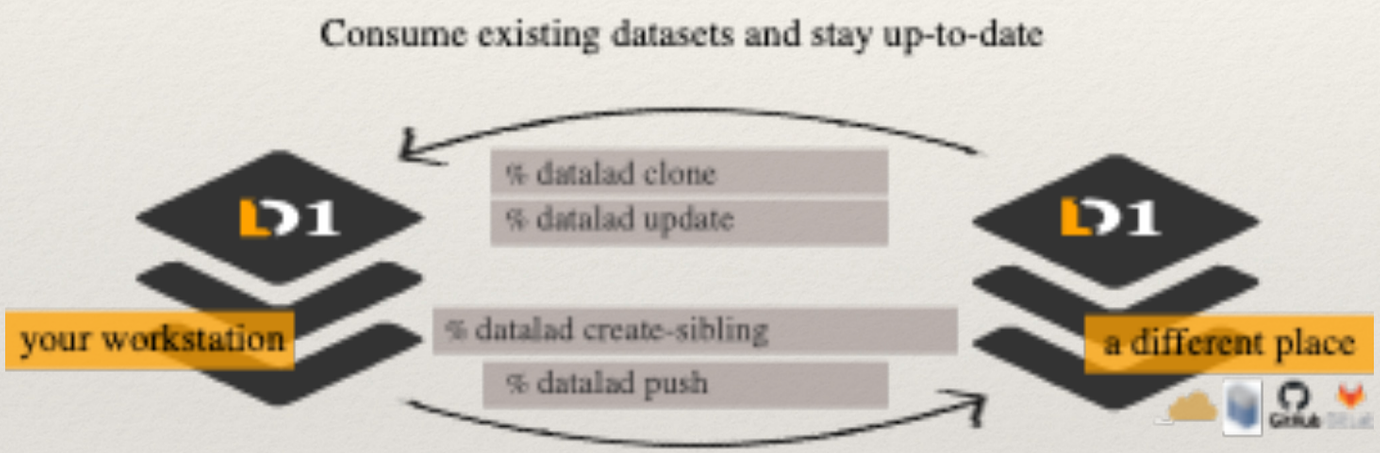
Regardless of Size, it can be version controlled using the same command:

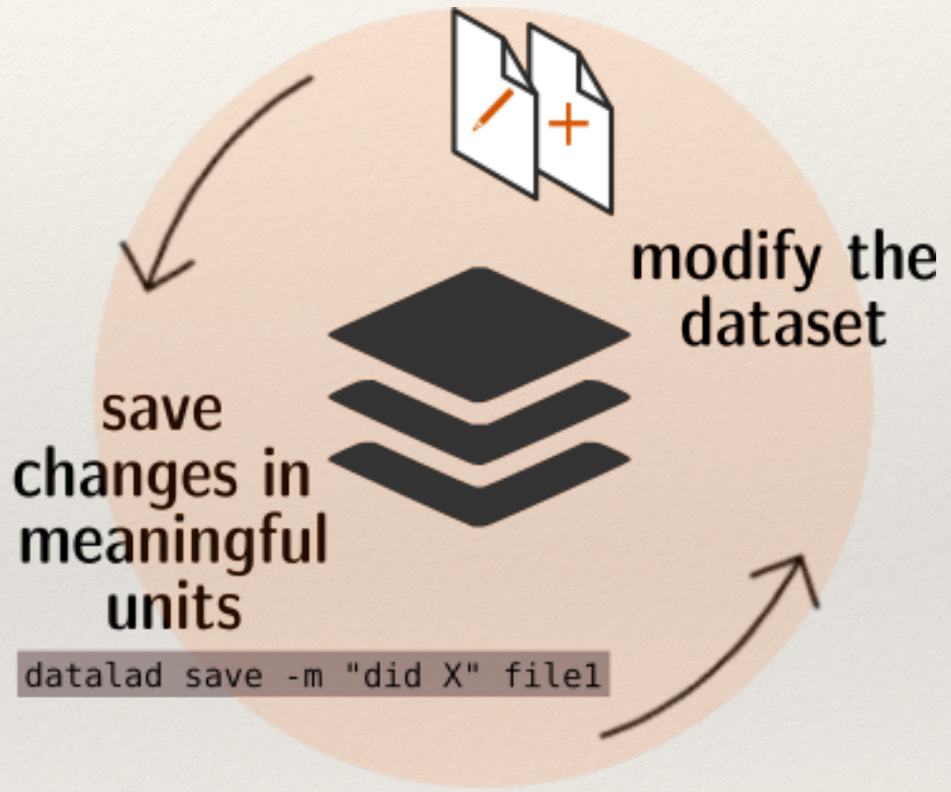
```
datalad save -m “save message” file-path
```

Version Controlling provides with the following benefits:

1. A proper structure for keeping files organised.
2. A proper way to keep track of changes.
3. Helps to look back at previous states and can also be reverted.

Why Version Control?





Why YODA is useful?

The core idea of YODA is to:

Organise projects in a modular and transparent way.

Separate raw data, code, and results.

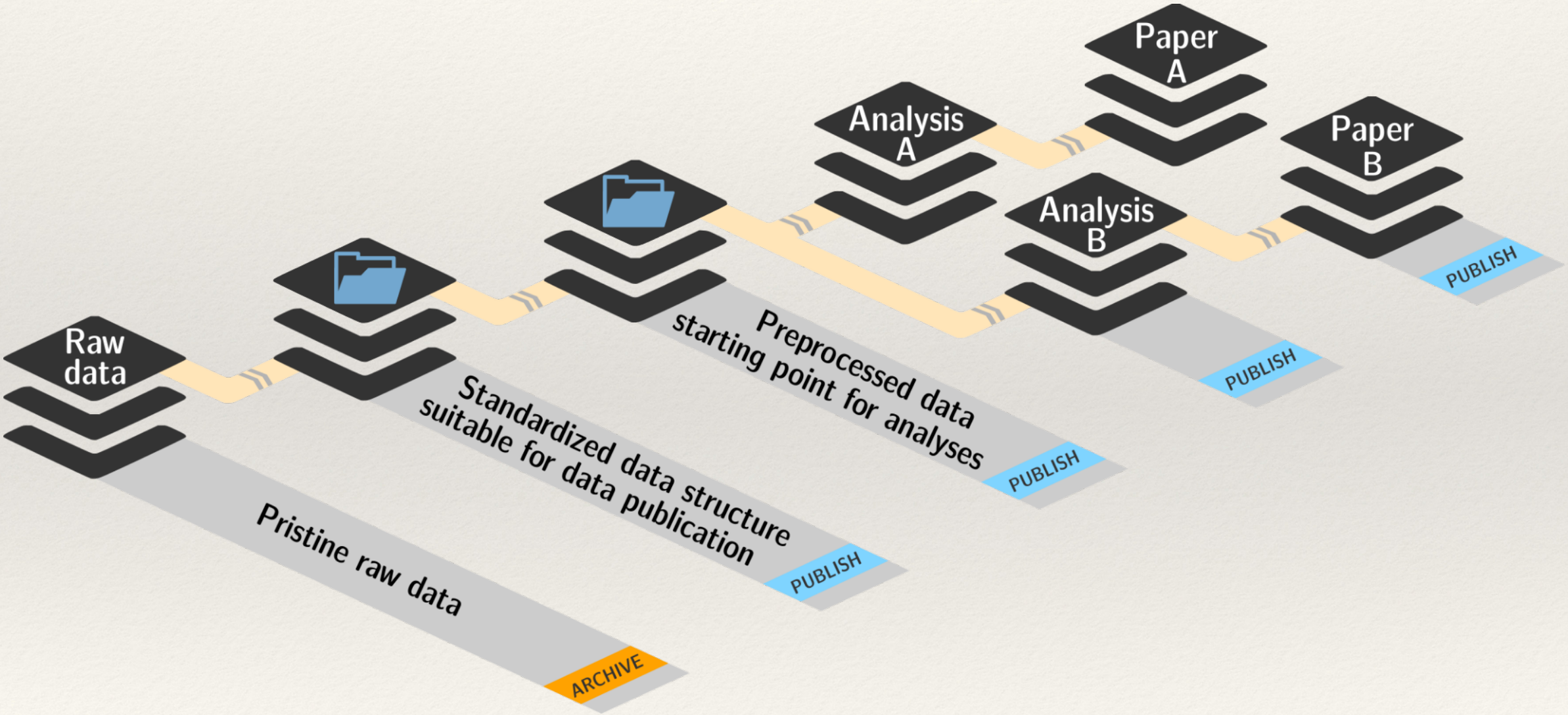
The Key components of YODA are:

1. **Modularity**: Use datasets and sub-datasets for different project components.
2. **Separation of Data and Code**: Keep raw data, scripts, and outputs in distinct directories.
3. **Provenance Tracking**: Record the history and origin of data and results.

This information was got and simplified from: Website.

```
(uni) bhanuprasanna@Bhanus-MacBook-Pro datalad-yoda-demo % tree
.
├── CHANGELOG.md
├── README.md
├── code
│   └── README.md
├── data
│   ├── processed
│   │   ├── test
│   │   └── train
│   └── raw
├── params
└── results
```

Dataset Linkage and Nesting



DataLad Datasets

Use `datalad create` command.

You can just add the following: (file_path, -c text2git file_path, -c yoda file_path) or use . to add in cwd/pwd.

```
(uni) bhanuprasanna@Bhanus-MacBook-Pro DataLad % datalad create -c yoda datalad-yoda-demo
[INFO   ] Running procedure cfg_yoda
[INFO   ] == Command start (output follows) =====
[INFO   ] == Command exit (modification check follows) =====
run(ok): /Users/bhanuprasanna/Documents/Uniklinik-Koln/DataLad/datalad-yoda-demo (dataset) [/Users/bhanuprasanna/anaconda3/envs/uni/...]
create(ok): /Users/bhanuprasanna/Documents/Uniklinik-Koln/DataLad/datalad-yoda-demo (dataset)
action summary:
  create (ok: 1)
  run (ok: 1)
(uni) bhanuprasanna@Bhanus-MacBook-Pro DataLad % cd datalad-yoda-demo
(uni) bhanuprasanna@Bhanus-MacBook-Pro datalad-yoda-demo % ls -a
.               ..              .datalad        .git            .gitattributes  CHANGELOG.md    README.md       code
(uni) bhanuprasanna@Bhanus-MacBook-Pro datalad-yoda-demo % tree
.
├── CHANGELOG.md
├── README.md
└── code
    └── README.md

2 directories, 3 files
```

DataLad Datasets

`datalad save` command for commit with messages.

```
datalad save -m “message” file_path
```

```
bhanuprasanna@Bhanus-MacBook-Pro data % datalad status
untracked: A12.csv (file)
untracked: A2.csv (file)
untracked: A21.csv (file)
untracked: A3.csv (file)
untracked: A4.csv (file)
bhanuprasanna@Bhanus-MacBook-Pro data % datalad save -m "Added MTL Datasets into data folder and now tracking it with DataLad"
add(ok): A12.csv (file)
add(ok): A2.csv (file)
add(ok): A21.csv (file)
add(ok): A3.csv (file)
add(ok): A4.csv (file)
save(ok): . (dataset)
action summary:
  add (ok: 5)
  save (ok: 1)
```

DataLad Datasets

Working with large data and have little disk-usage.

DataLad helps in this regard by providing us content on Demand using the get and the content can be dropped when not in use using drop.

```
datalad get file

datalad drop file
```

DataLad Datasets

`datalad run`: Tracks and records commands and their outputs in a version-controlled manner, ensuring full reproducibility of data processing and analysis steps.

```
datalad run --input <input> --output <output> <command>
```

`datalad rerun`: Re-executes previously recorded commands, making it easy to reproduce or update results exactly as before.

```
datalad rerun —-since=<commit_code>
```

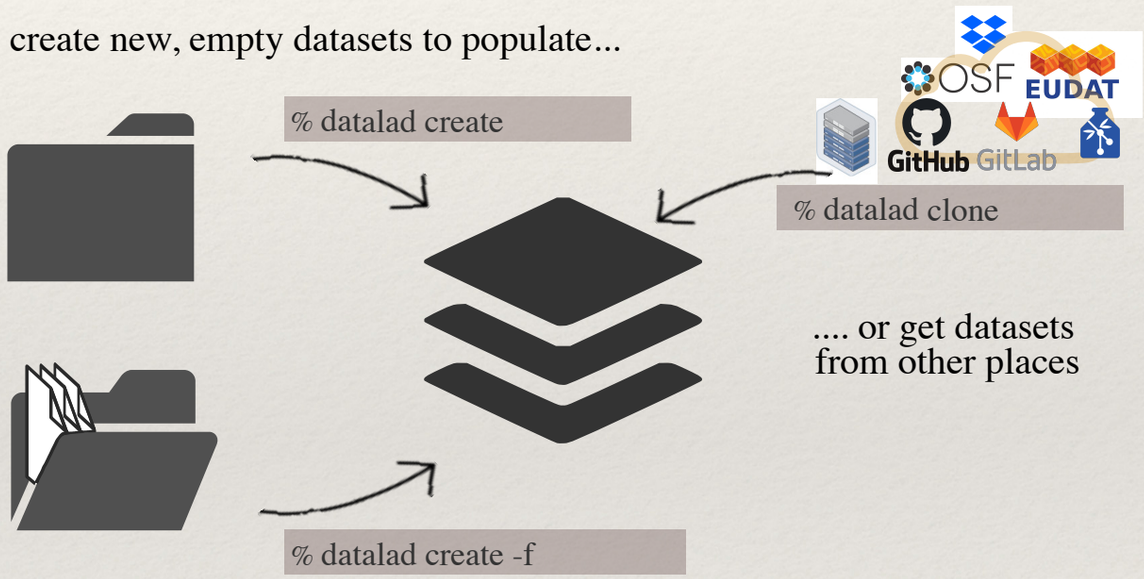
Transitioning into DataLad

Transitioning Existing Projects into DataLad

Unlock DataLad Features: Transform your projects to leverage DataLad's powerful capabilities.

Preparation Is Key: Familiarise yourself with DataLad basics before proceeding.

Backup First: Create copies of your data to prevent any unintended data loss.



Step 1 – Planning

Determine Dataset Structure:

- Single dataset or multiple nested datasets?

Considerations:

- Current project size and expected growth.
- Nature of the content (data types, reuse potential).

Guidelines:

- Follow YODA principles for best practices.

Step 2 – Dataset Creation

Initialise Datasets:

- Use `datalad create --force` in non-empty directories.

Nested Datasets:

- Create sub-datasets for nested directories as needed.

Automation:

- Utilise bash loops to automate dataset creation for multiple directories.

Step 3 – Saving Dataset Contents

Save Changes:

- Use `datalad save` to commit dataset contents.

Git vs. git-annex:

- **Git**: Small, frequently modified files (e.g., code, text).
- **git-annex**: Large or binary files (e.g., datasets, media).

Hosting Considerations:

- Be mindful of file size limits on platforms like GitHub.
- Consider services like GIN or OSF for annexed content.

Step 4 – Reproducing Analyses & Summary

Reproduce Analyses:

- Use `datalad run` or `datalad containers-run` to rerun computations.
- Specify outputs to ensure previous results are updated.

Key Takeaways:

- Plan your dataset structure thoughtfully.
- Backup your data before making changes.
- Understanding DataLad enhances your data management efficiency.

Live Code Demo

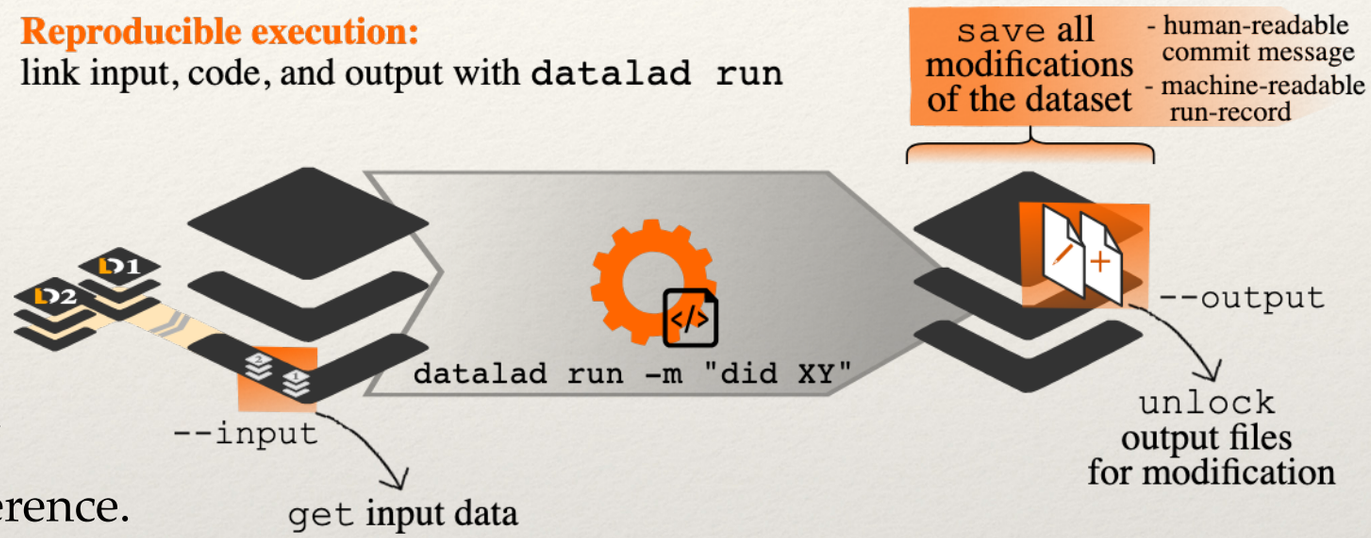
Code to follow along:

Link

Reproducibility

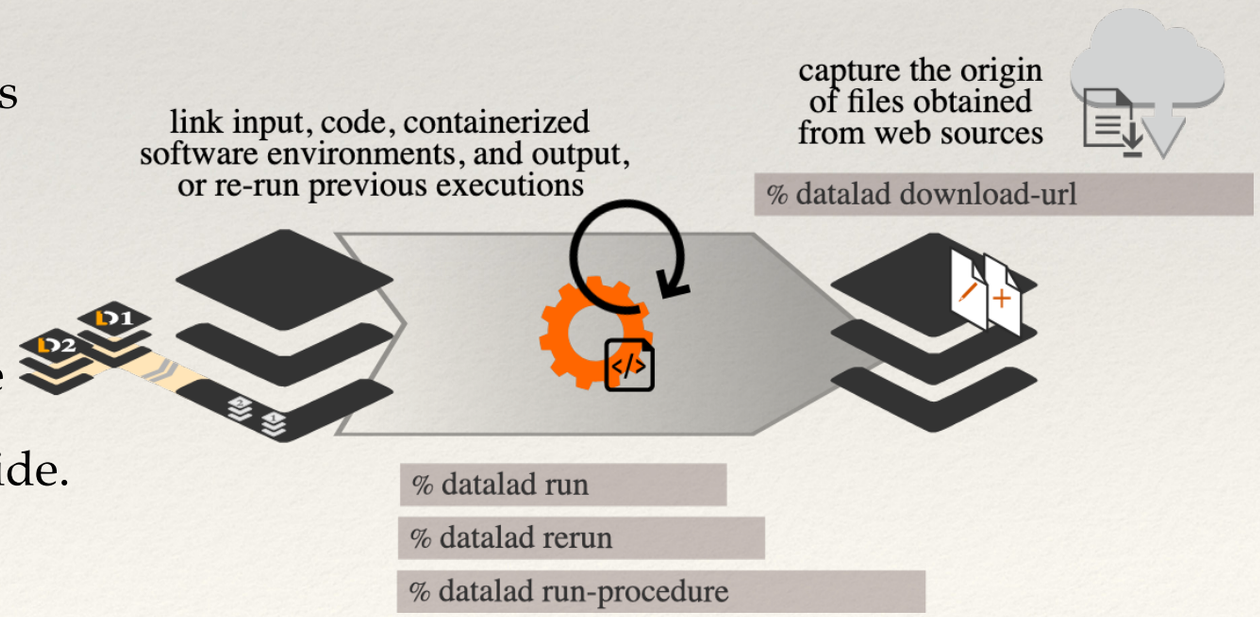
Reproducible Analyses with DataLad

DataLad helps create datasets that can easily and automatically **recompute analyses** for future reference.



Captures the provenance of your results, and enables automatic recomputation with datalad run.

This misses a very crucial part of Digital Provenance which is software. It is addressed in the following slide.



Ensuring Reproducibility with Containers

Reproducibility Challenge: Ensuring analyses can be replicated across different environments is difficult due to variations in operating systems, software versions, and configurations.

DataLad and Reproducibility:

- DataLad manages datasets, code, and human-readable documentation to enhance reproducibility.
- However, it's insufficient without considering the **software environment** (e.g., OS, libraries).

Software Containers: Offer a solution by bundling the necessary software environment, libraries, and dependencies in a portable format, ensuring analyses run consistently across different systems.

Ensuring Reproducibility with Containers

What are Containers?

- Lightweight, portable environments that bundle **software libraries** and **dependencies** in the exact versions needed for your analysis.
- Provide a **secluded software environment** that doesn't affect the host system.

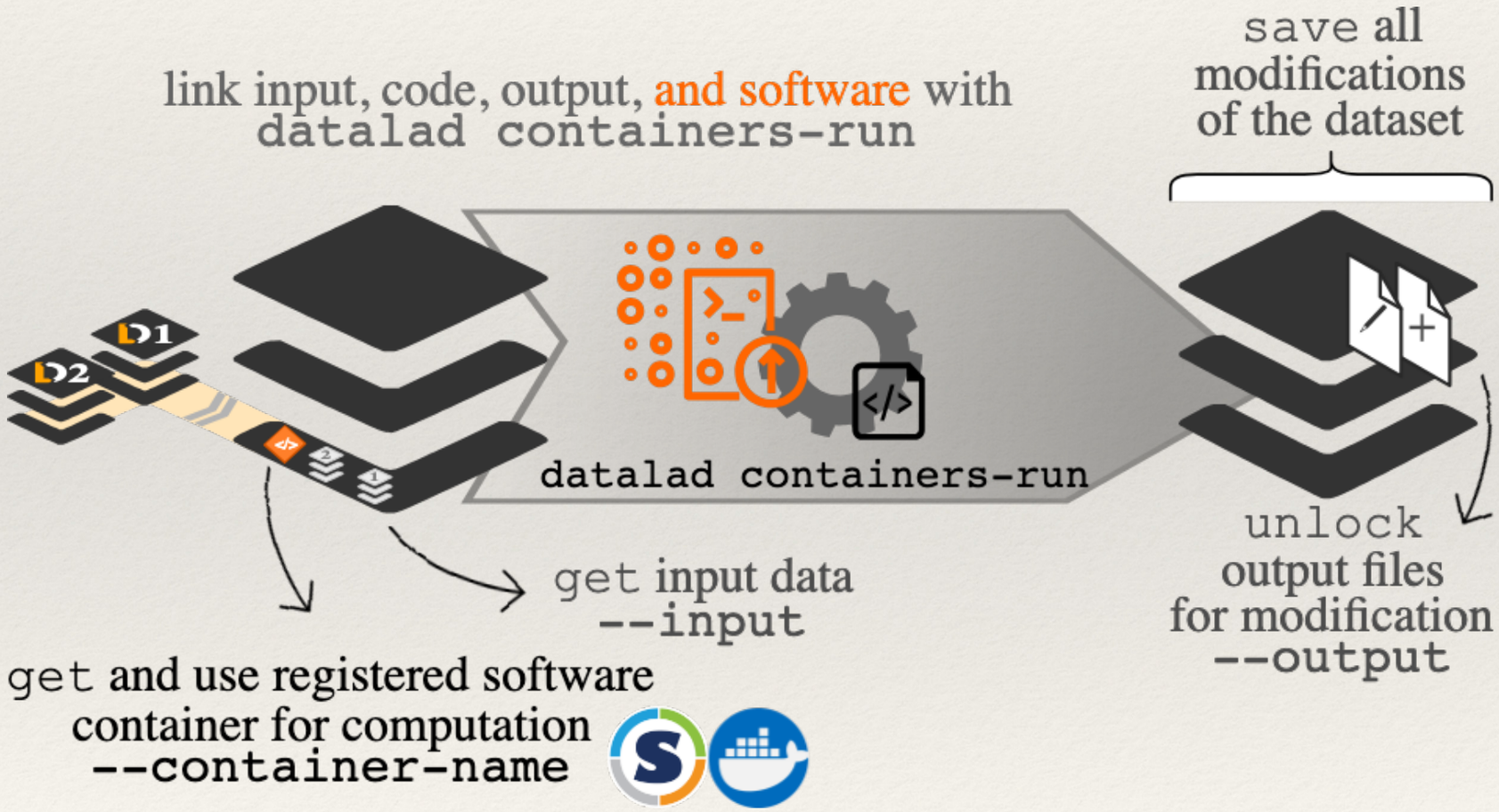
Why Use Containers?

- Guarantees **computational reproducibility** by sharing the same software environment across different machines.
- **Lightweight** compared to virtual machines and **easy to share**.
- Ideal for running analyses without modifying or installing software on other machines.

Key Tools:

- **Docker**: Popular for general use.
- **Singularity**: Often used on high-performance computing (HPC) systems.

Ensuring Reproducibility with Containers



Ensuring Reproducibility with Containers

DataLad-Container Extension: Adds functionality to manage software containers like **Singularity** and **Docker**.

- Allows registration of containers to datasets using commands like datalad containers-add.
- Supports executing commands in a precise software environment via datalad containers-run.

Benefits:

- Isolates the computational environment, ensuring reproducibility.
- Easy to share the complete software setup with others or rerun on different machines.
- Example: Running an analysis with Singularity can be done using a command like:

```
$ datalad containers-run --container-name <name> --input <input> --output
<output> <command>
```

Thank you for your Attention. 😀

References

- https://www.datalad.org/#install
- https://github.com/datalad/datalad
- https://yoda.yale.edu/about/roles-responsibilities/
- https://handbook.datalad.org/en/latest/basics/101-127-yoda.html
- https://handbook.datalad.org/en/latest/index.html
- https://www.youtube.com/@DataLad