love DATA week

9 -13 February

# Join for the Lib4RI coffee lectures

Monday, 9 Feb. @ 13:00-13:30
**Data Management Plans made easy: live demo**

Tuesday, 10 Feb. @ 13:00-13:30
**Accelerating battery materials research with linked data and automated workflows**

Wednesday, 11 Feb. @ 13:00-13:30
**From raw to reproducible: Versioning research data for the future**

Thursday, 12 Feb. @ 13:00-13:30
**Galaxy Swiss: Enabling reproducible, interdisciplinary data analysis for Swiss researchers**

Friday, 13 Feb. @ 13:00-13:30
**Containers, Docker, Environments, oh my! How to share your code with less chaos**

*lib4ri.ch/love-data-week-2026*

LIB4RI DATA

***Stop by any Lib4RI location for Lib4RI or LoveData sticker***

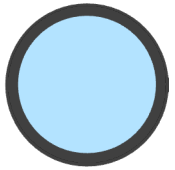
Lib4RI

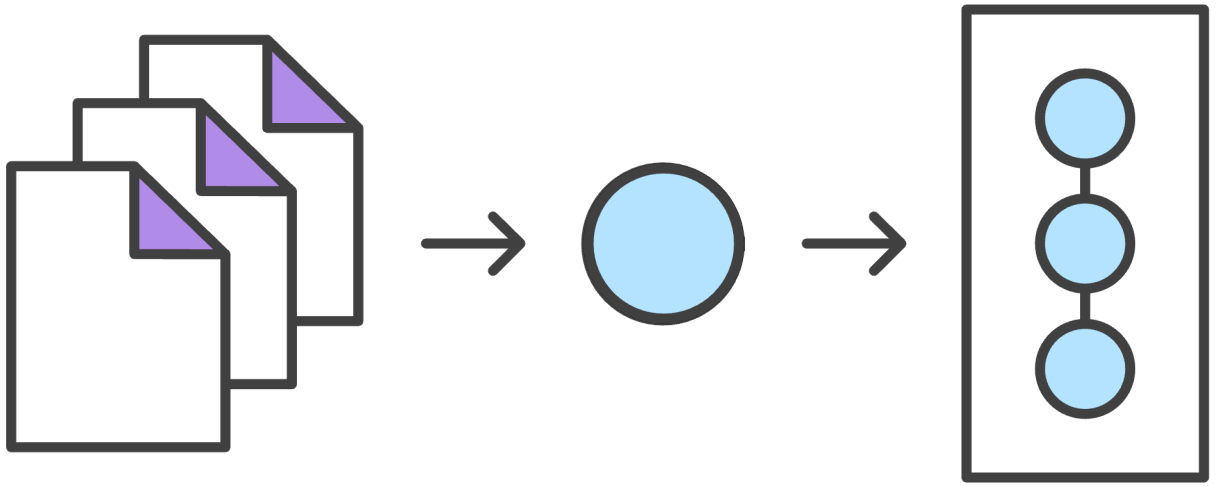
February 11, 2025

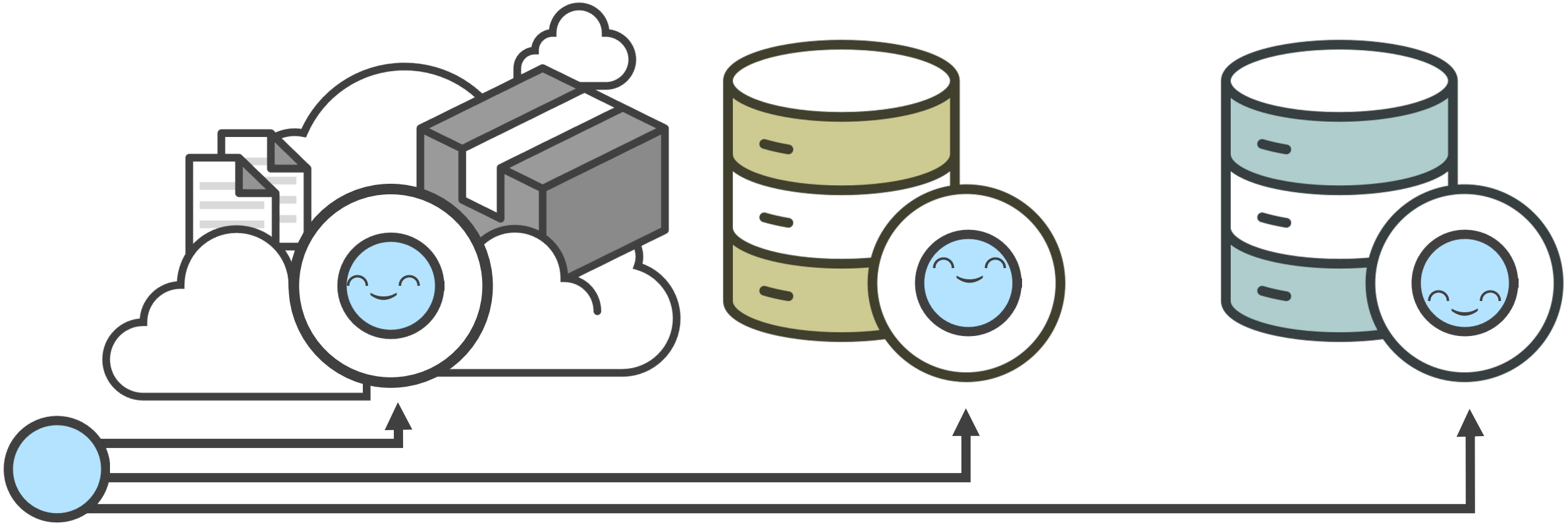
# From Raw to Reproducible:

## Versioning Research Data for the Future

Dr. Chase Núñez







# keep a true raw data file:

**3** copies on **2** different medium, and **1** off-site

## ERA 1: MECHANICAL & EARLY MAGNETIC (Pre-1970s)

**PUNCH CARDS (1800s-1950s):**
Data as holes in paper. Limited capacity, slow.

**MAGNETIC TAPE REEL**
**Magnetic Tape (1950s+):**
Sequential access, high capacity for archival. Still in use.

**HARD DISK DRIVE**
**IBM 350 RAMAC (1956):**
First HDD. Massive size, 5MB capacity. Random access breakthrough.

## ERA 2: THE MAGNETIC DISK EVOLUTION (1970s-1990s)

**8" INCH FLOPPY DISK**
**8" Floppy (1971):**
Portable magnetic storage. Became smaller (5.25", 3.5").

**3.5-INCH FLOPPY DISK**
**3.5" Fleppy (1980s):**
Dominant portable media, ubiqultous. ~1.44MB.

**COMPACT DISC**
**CD-ROM (1980s):**
Optical storage revolution. Digital audio & data, ~700MB. Read-only initially.

## ERA 3: OPTICAL & EARLY FLASH (Late 1990s-2000s)

**DVD**
**(Mid-1990s):**
Higher density optical. Video & data standard, ~4.7GB+.

**USB DRISH**
**(2000):**
**(2000):** Portable, durable flash memory. Replaced floppies. Capacities grew rapidly.

**BLU-RAY DISC**
**Blu-ray (2006):**
HD video & large data. Up to 100GB+.

## ERA 4: SOLID STATE & BEYOND (2010s-Present)

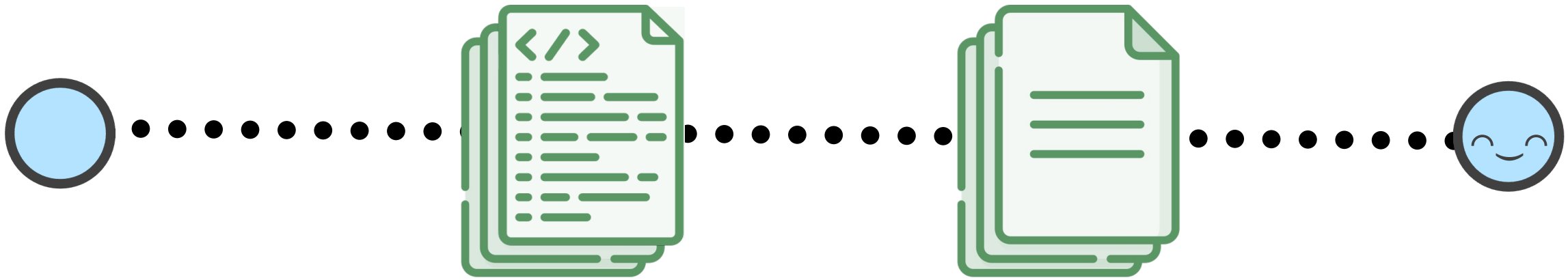
**SOLID STATE DRIVE (SSD)**
**SSD (2010s+):**
No moving parts. Faster, reliable, energy-efficient. Mainstream PC storage.

**NVMe M.2**
**(Mid-2010s+):**
**(Mid-2010s+):**
Direct PCIe connection. Extremely high speeds, compact form factor.

**CLOUD STORAGE**
**(Ongoing):**
Remote server data via internet. Scalable, accessible anywhere, service-based.

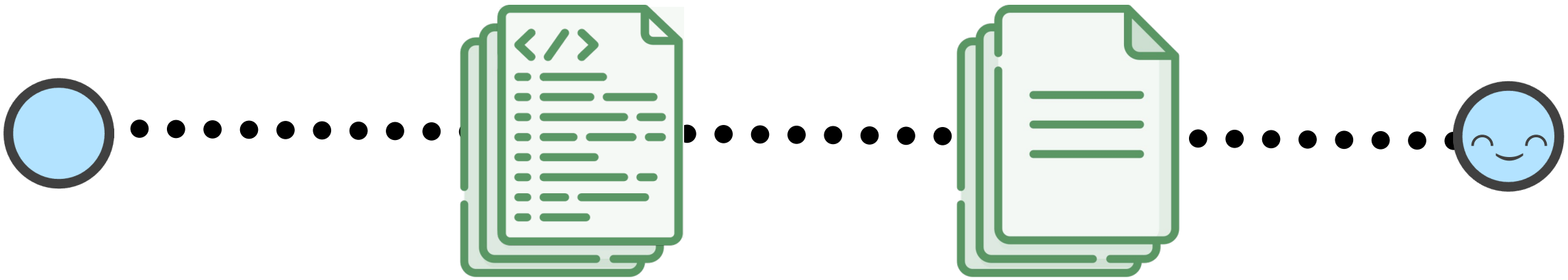
**FUTURE**
**(DNA STORAGE, ETC.):**
Emerging concepts for ultra-high density, long-term archival.

**3** copies on **2** **different medium**, and **1** off-site



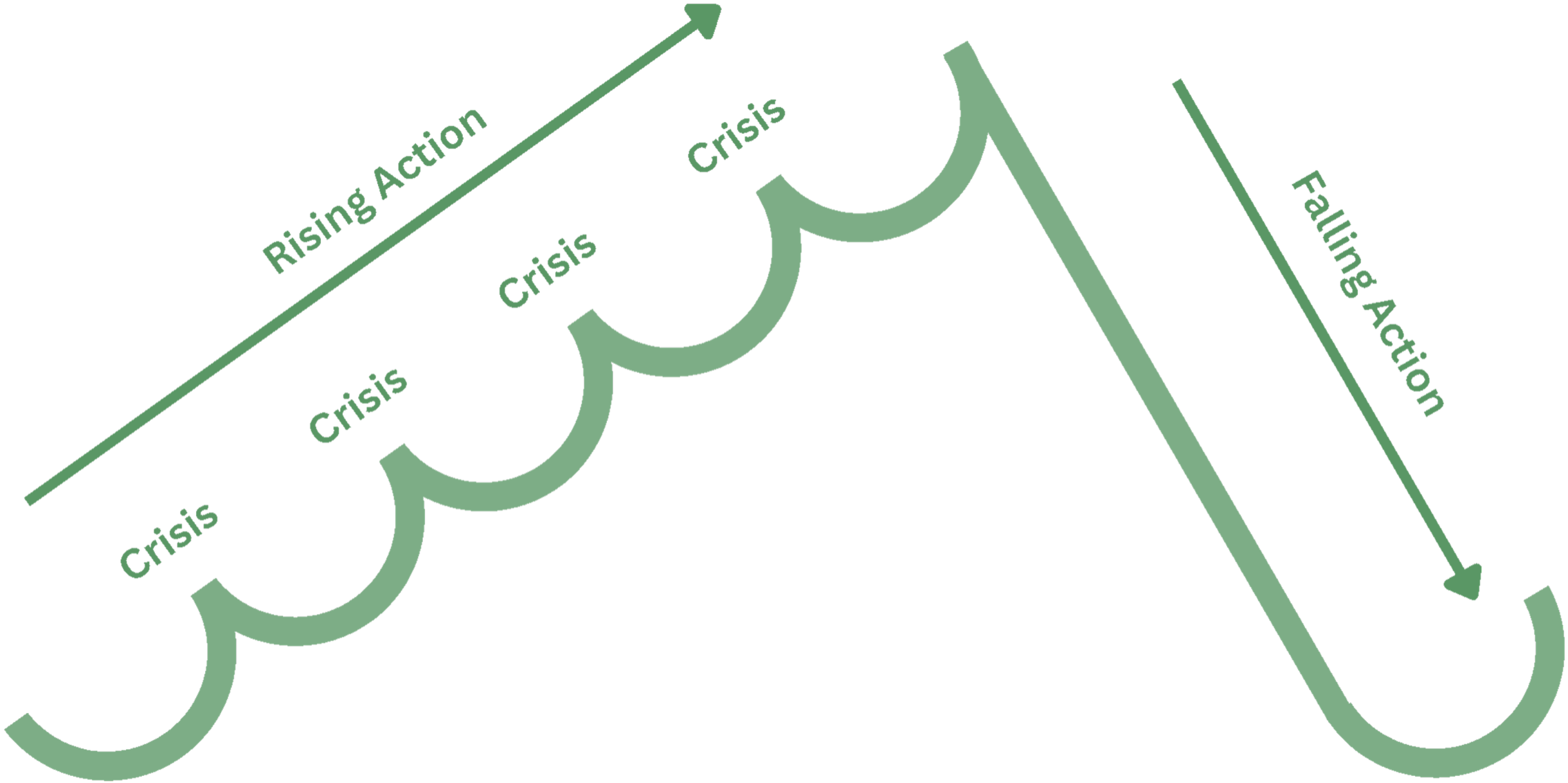
# record every change:

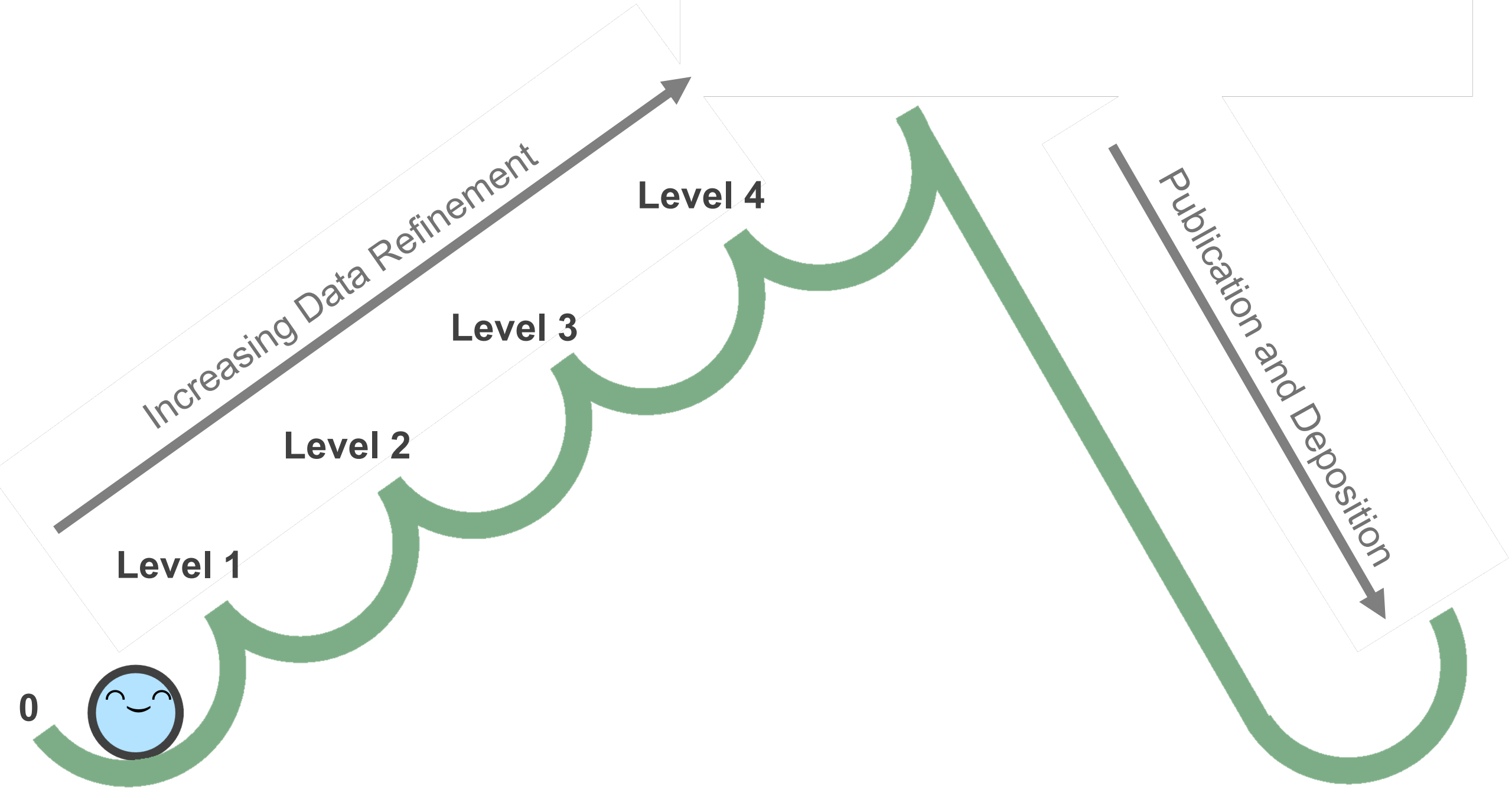
**what** changed; **why** it changed; and **how** it changed



# build a narrative file structure:

***a complete story. no sequels.***


Rising Action
Crisis
Crisis
Crisis
Crisis
Falling Action


Increasing Data Refinement
Level 4
Level 3
Level 2
Level 1
0
Publication and Deposition

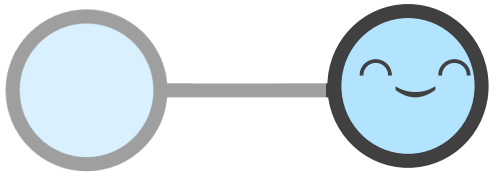
# level 0 data

## Raw/Untouched

- **all original data files and metadata, *exactly as collected***
    - raw tables
    - original images/videos
    - instrument logs
- do not apply any corrections, filters, or formatting.
- do not discard apparent errors or outliers – keep the true raw readings.

# level 1 data

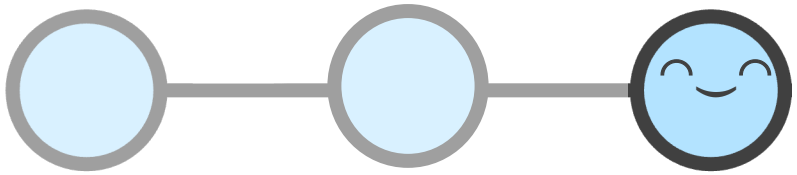
## Cleaned/Calibrated



- **the same raw data as L0, but truer**
  - removing obvious sensor artifacts
  - applying known corrections
  - unit conversions
  - Timestamps
  - geolocation tags
- **do not aggregate or summarize**
  - keep one-to-one correspondence with each raw sample.

# level ② data

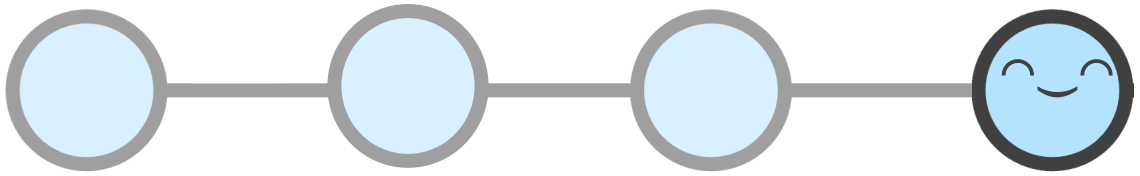
## Derived Variables



- **computed columns or variables derived from L1**
  - spectral indices
  - biomass estimates
  - interpolated values
- these should **still align with the original** measurement points or pixels.
- **exclude** large-scale averages or joined data from other sources.

# level ③ data

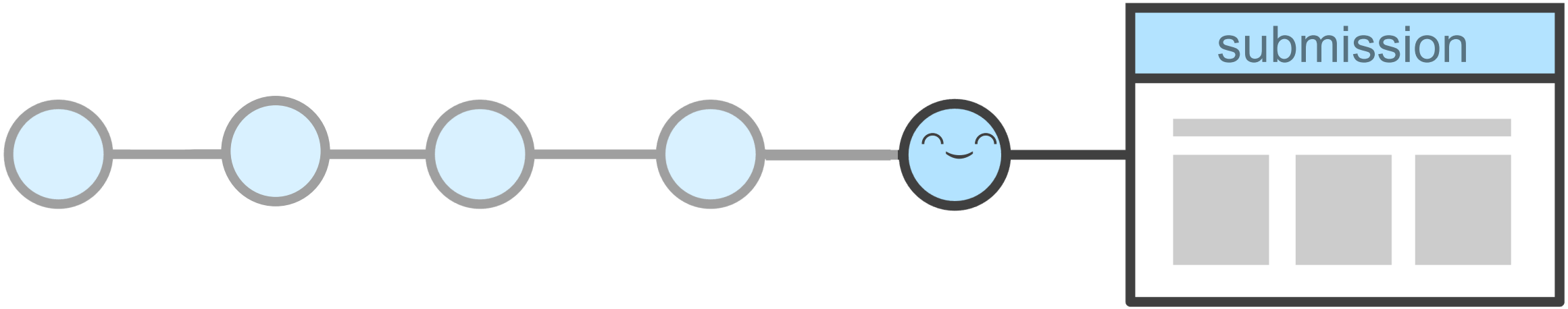
## Aggregated/Gridded



- data **re-mapped or summarized** over space/time
  - hourly or spatial means
  - gridded maps
  - joined sensor networks
- these products should be **internally consistent**
  - a complete time-series or uniform grid
- **exclude** original sample-level detail and raw logs.
- level 3 data often have **reduced resolution or frequency**

# level 4 data

## Model/Analysis Outputs




- results from **models or analyses** that **combine L0–L3 products**
  - climate model runs
  - forecast maps
  - statistical summaries over many measurements
- **exclude** any new raw observations or low-level variables.
- level 4 is the **end-of-pipeline**
  - a forest biomass map predicted by a model using L3 data


Reviewer two wants
to fight !


De-aggregate your
covariates !

**DATA**

data_level_0.csv

data_level_1.csv

data_level_2.csv

data_level_3.csv

data_level_4.csv

README.md

**SCRIPTS**

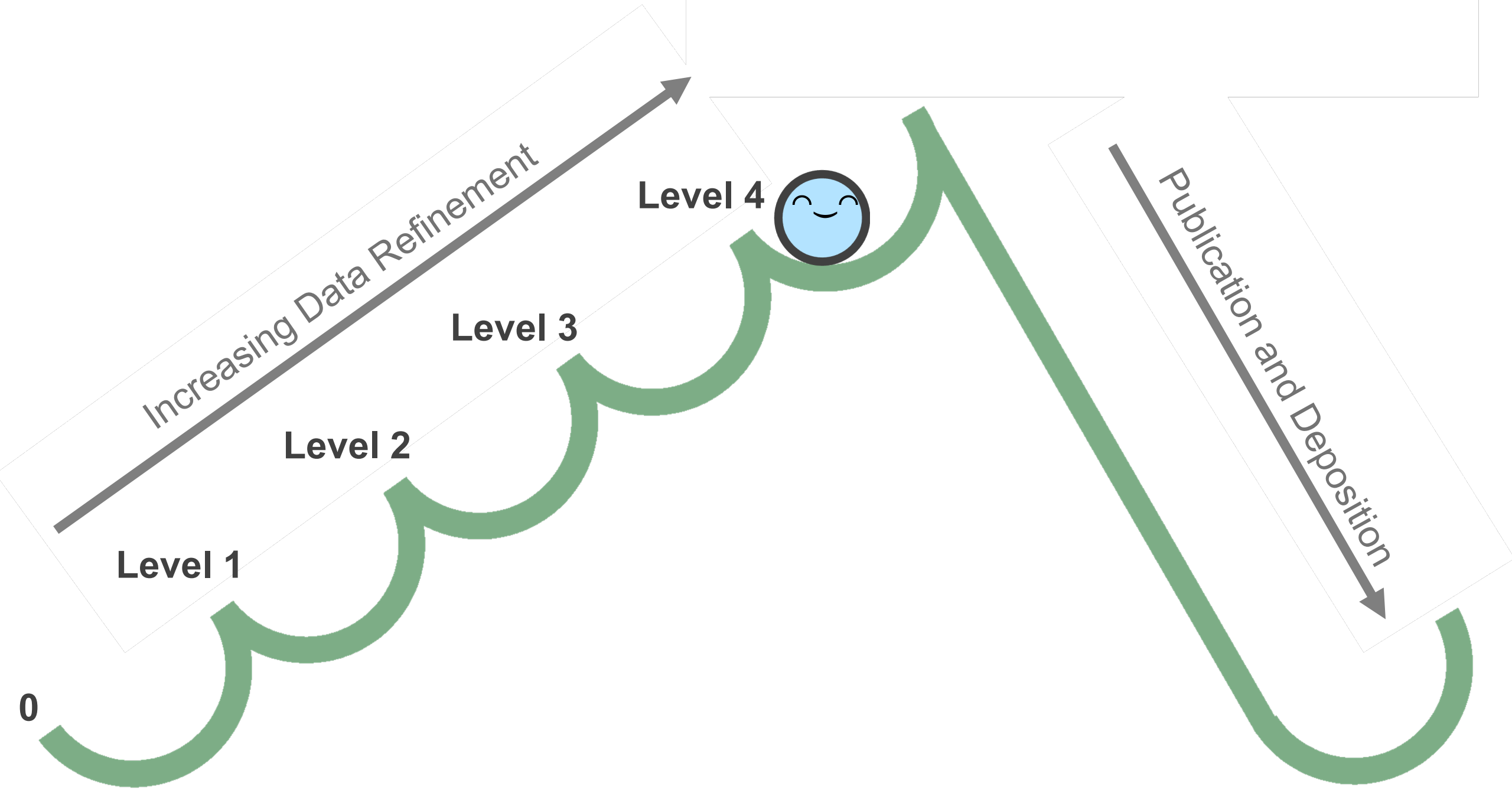
Level_0_to_1.R

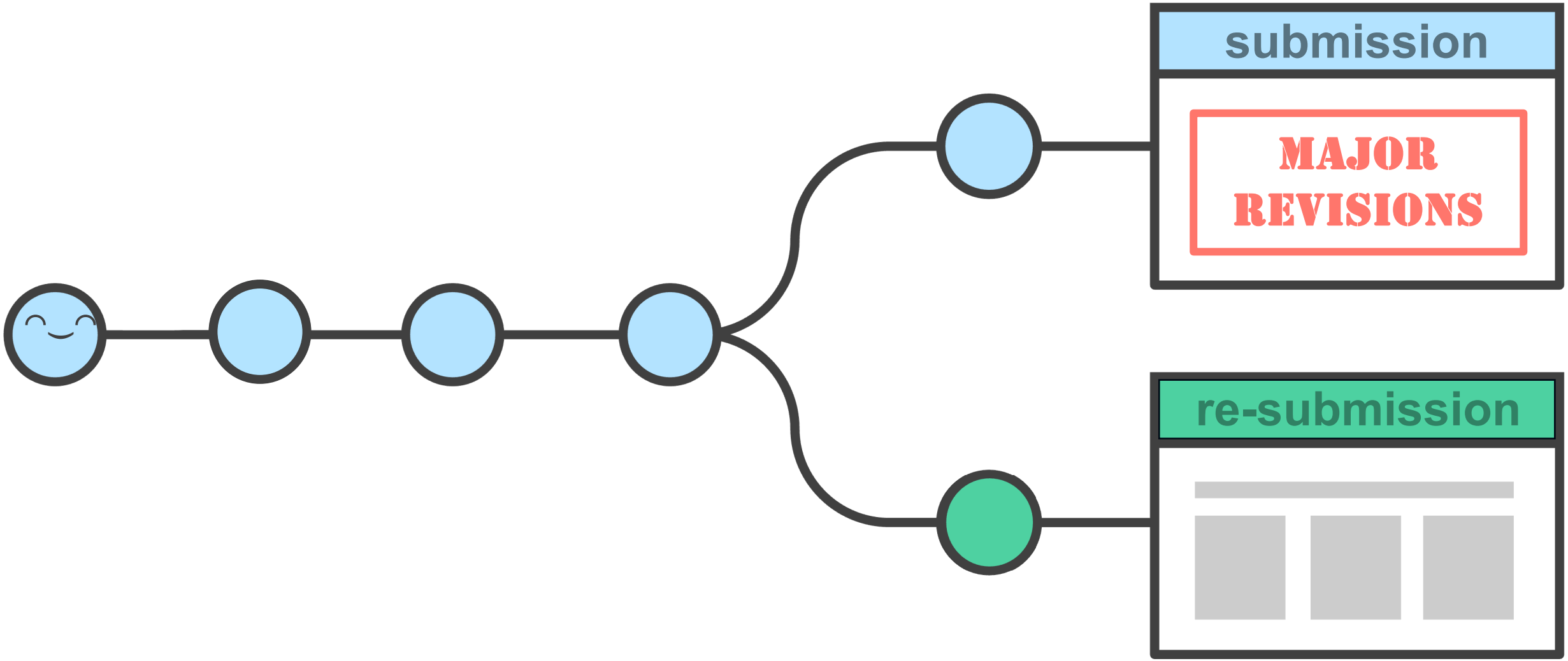
Level_1_to_2.R

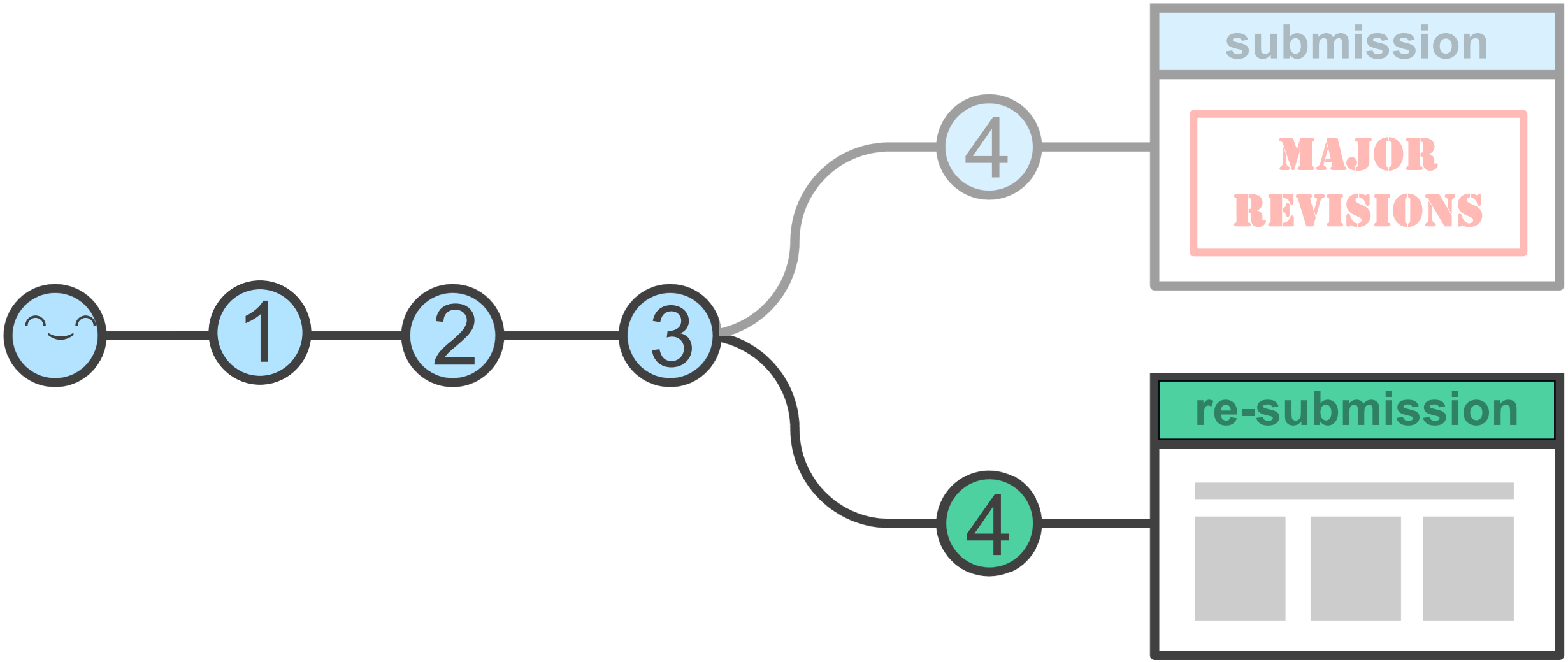
Level_2_to_3.R

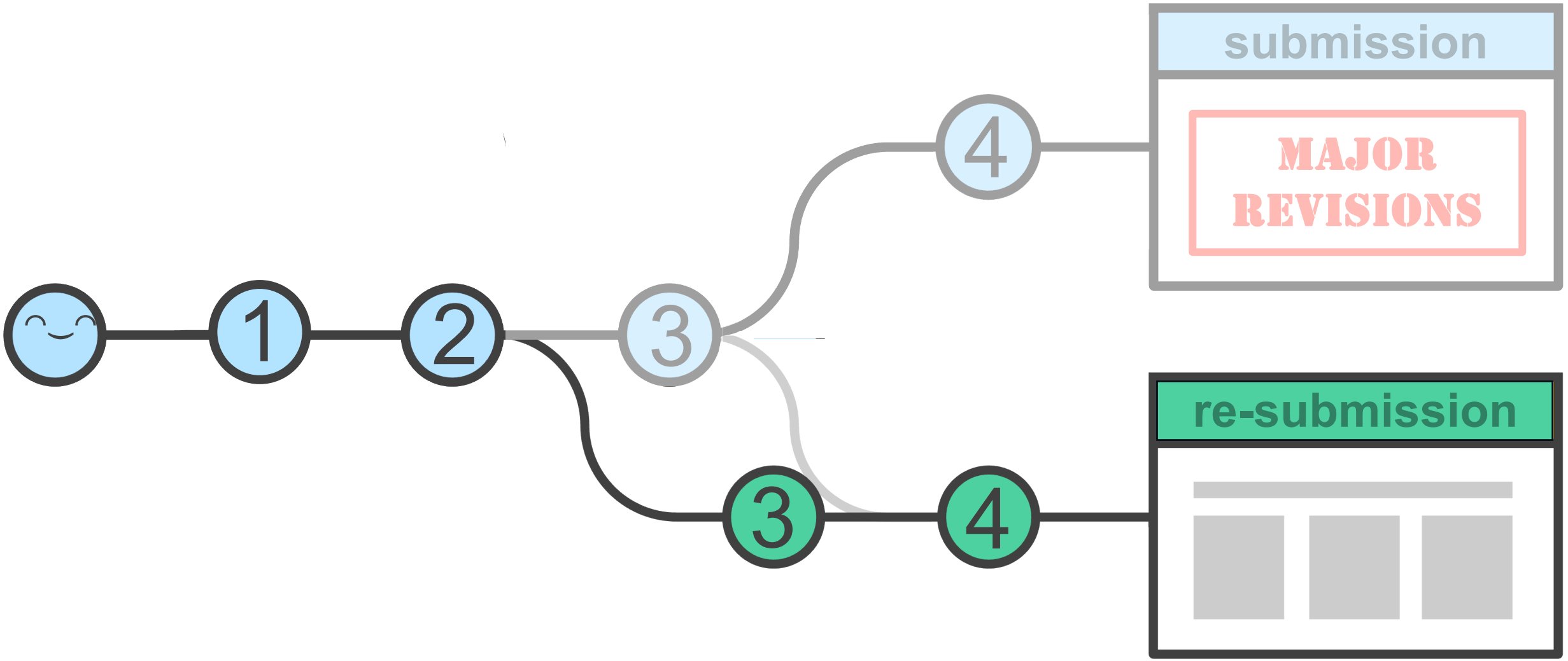
Level_3_to_4.R

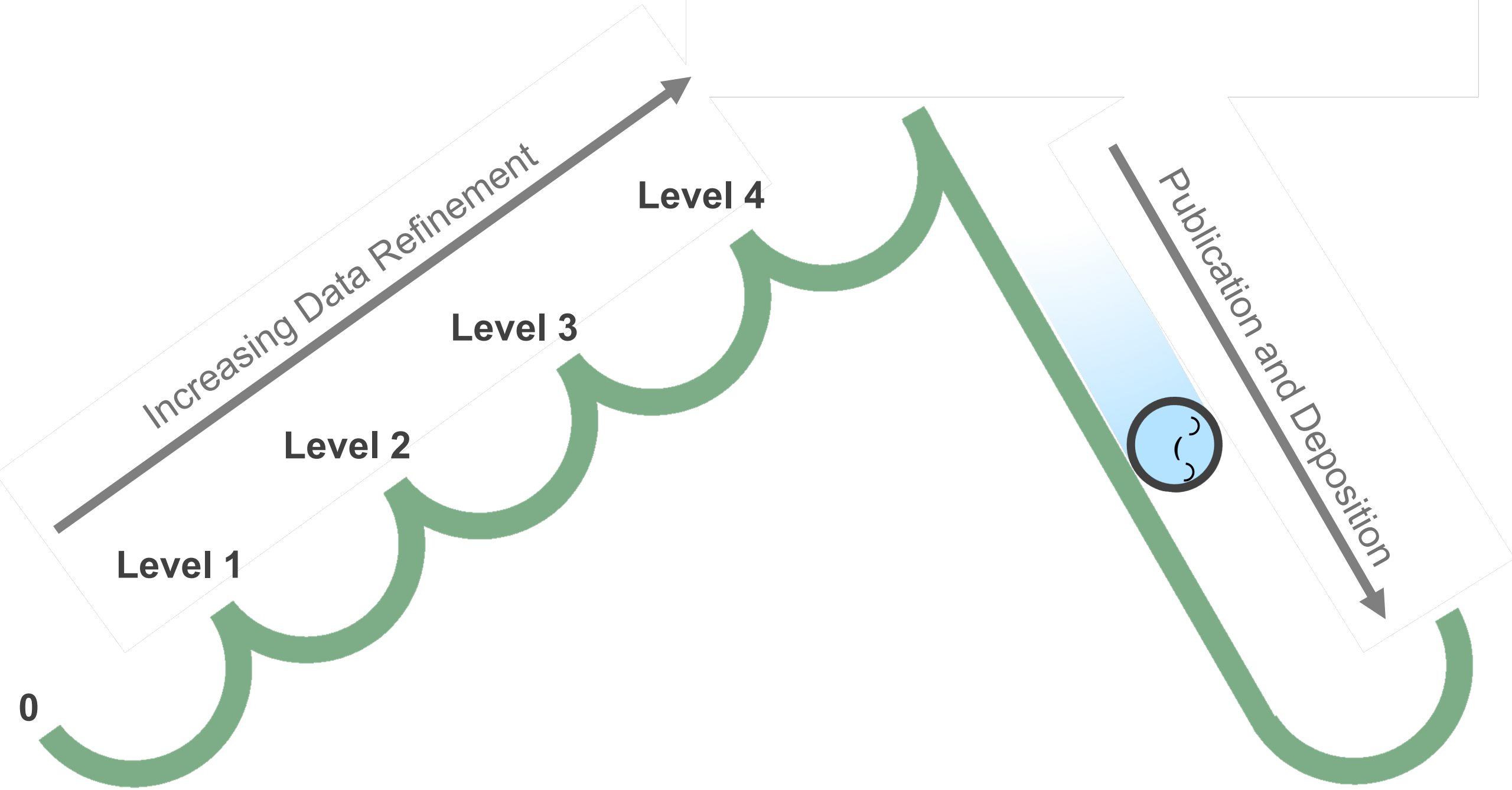
README.md

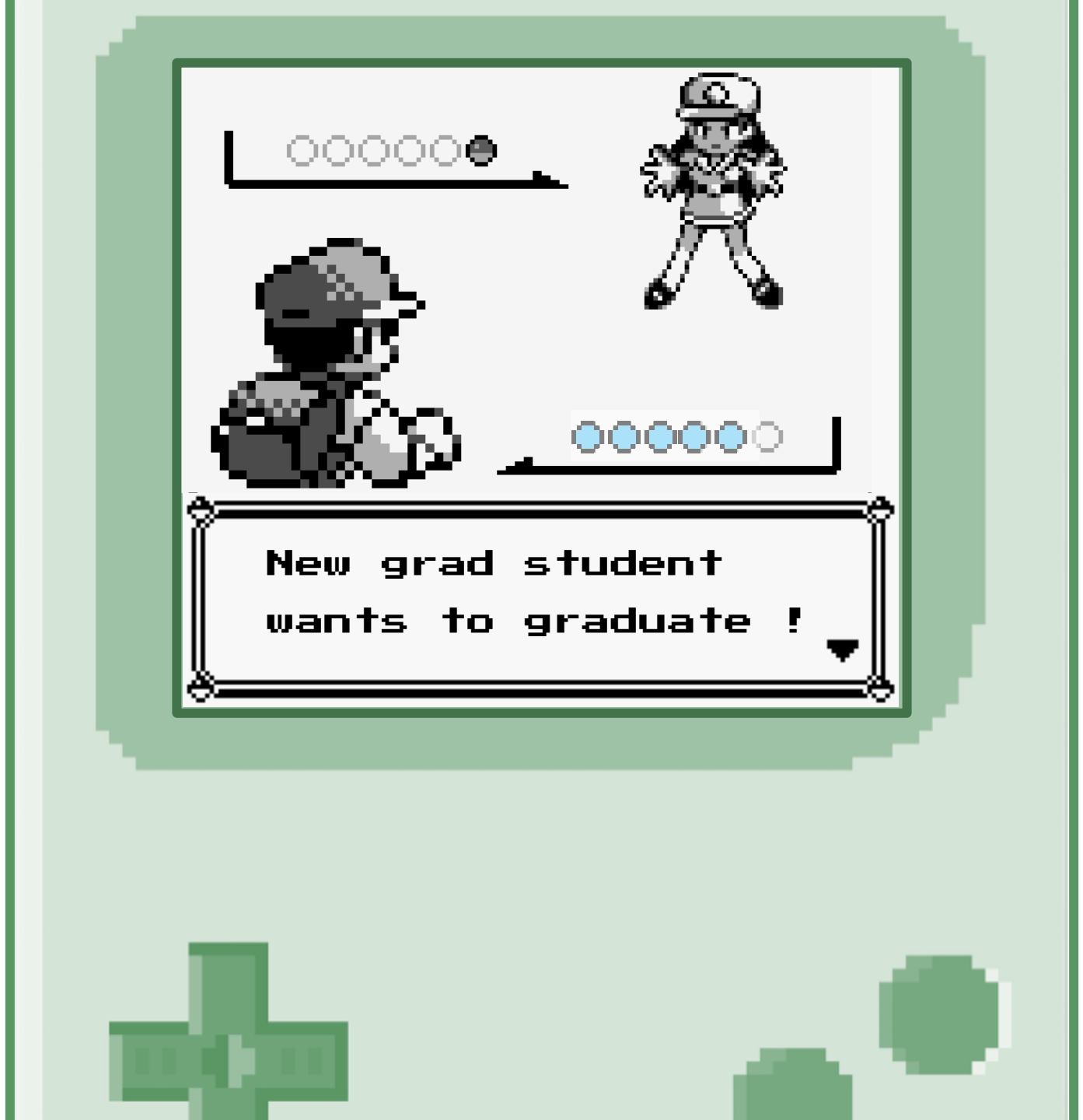

Increasing Data Refinement
Level 4
Level 3
Level 2
Level 1
0
Publication and Deposition

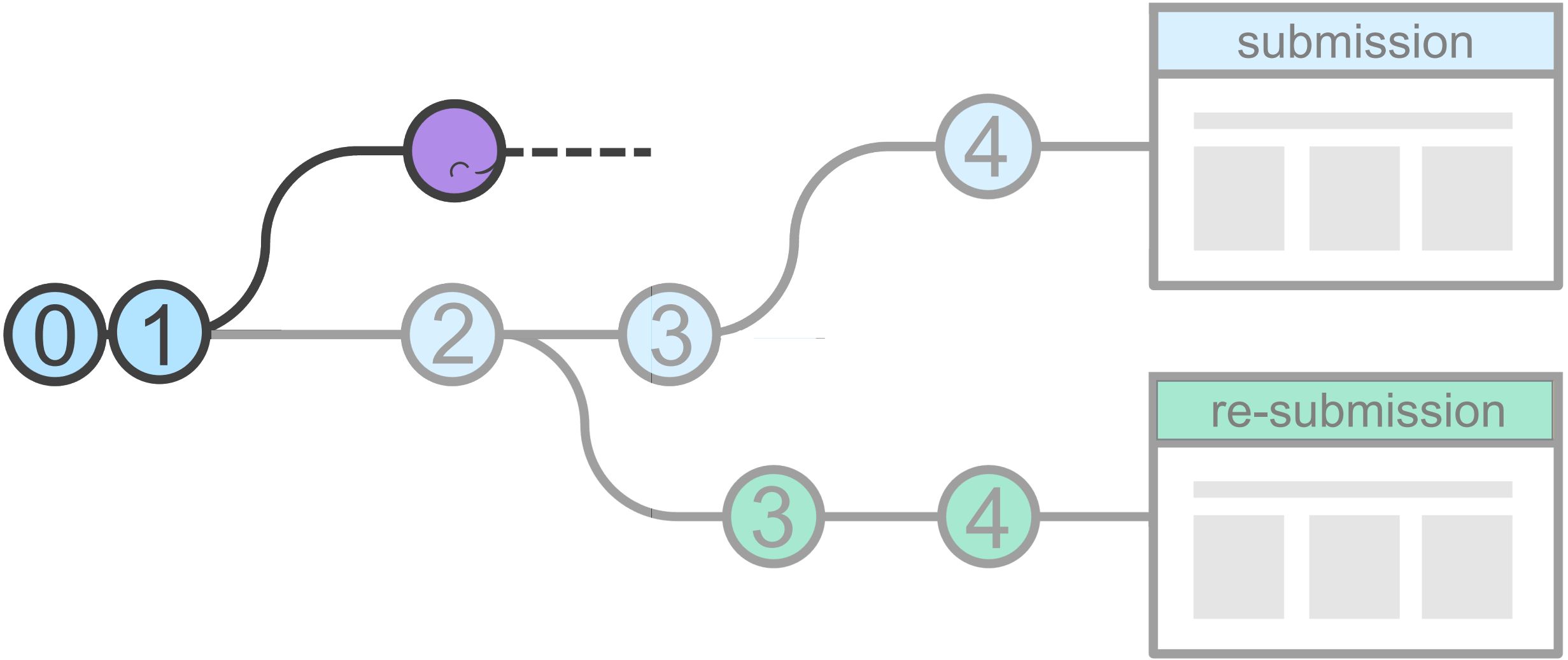

submission
MAJOR REVISIONS
re-submission

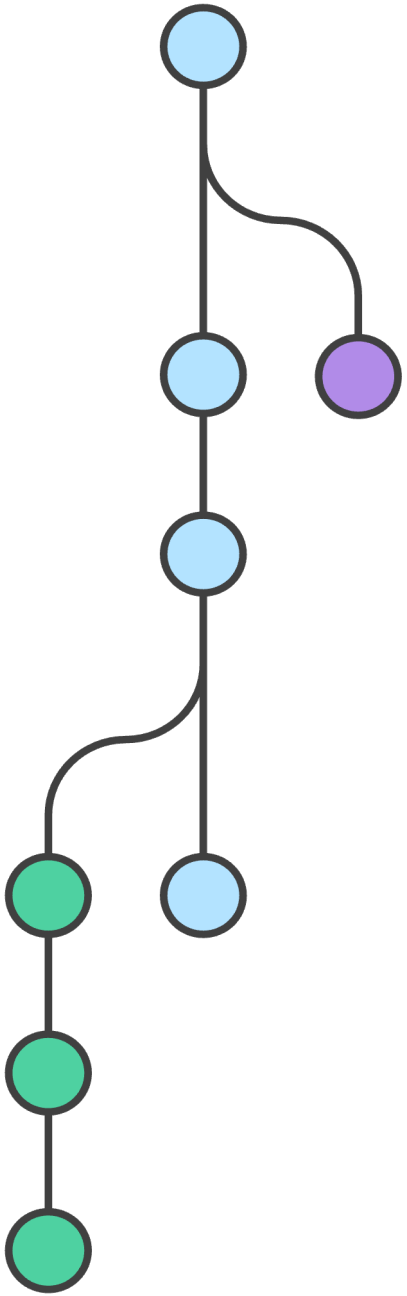

1
2
3
4
submission
MAJOR REVISIONS
4
re-submission


submission
MAJOR REVISIONS
re-submission
1
2
3
4
3
4

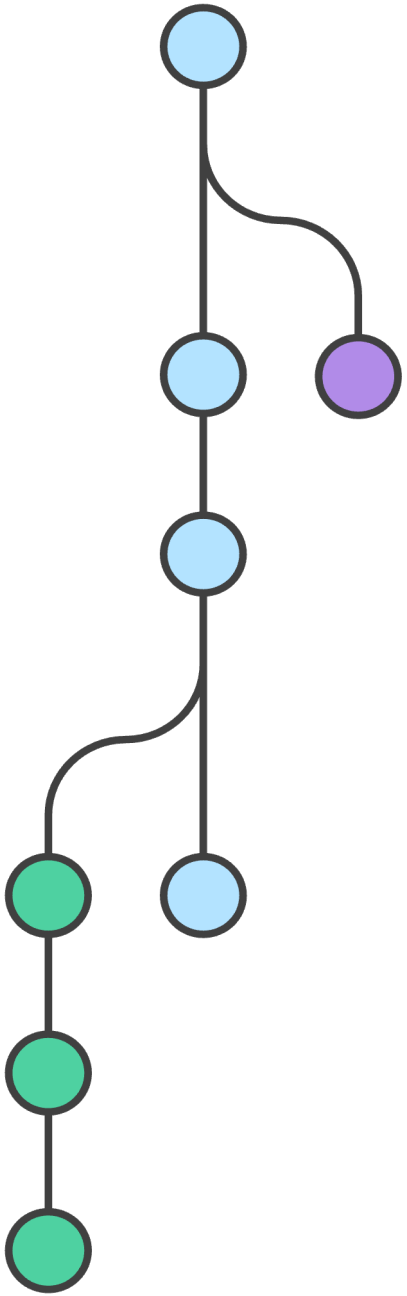

Increasing Data Refinement
Level 4
Level 3
Level 2
Level 1
0
Publication and Deposition


New grad student
wants to graduate !

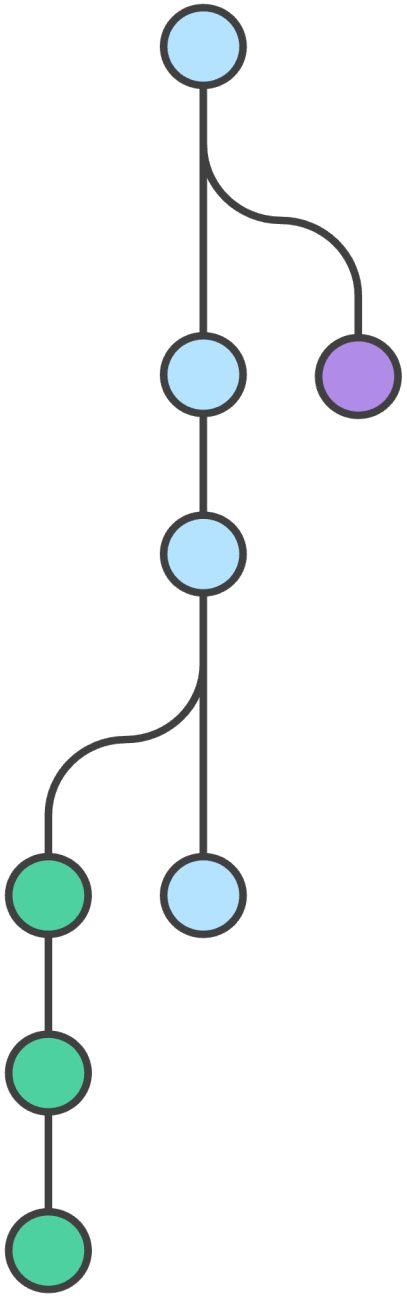

submission
re-submission
0
1
2
3
4
3
4



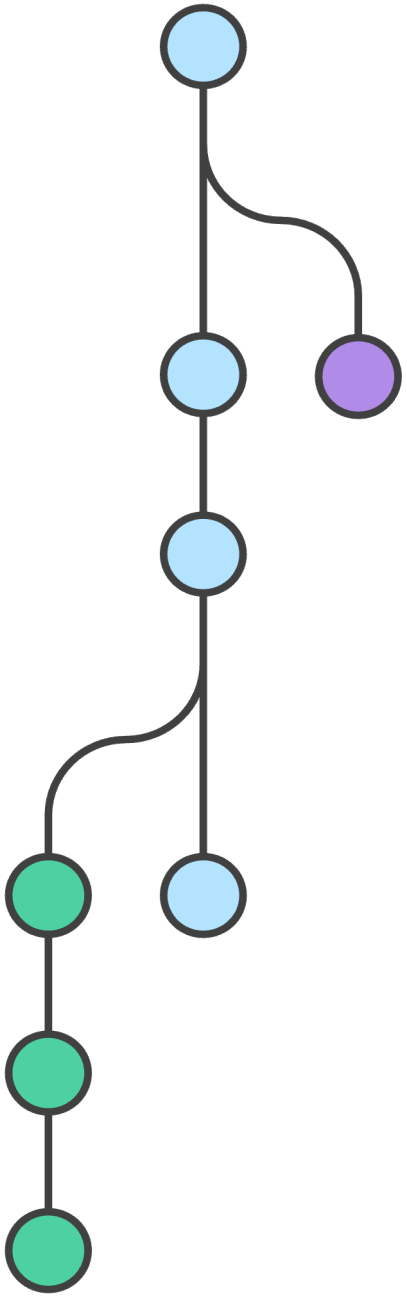
- We advocate for **working directly from your raw data**, and version control your **code.**



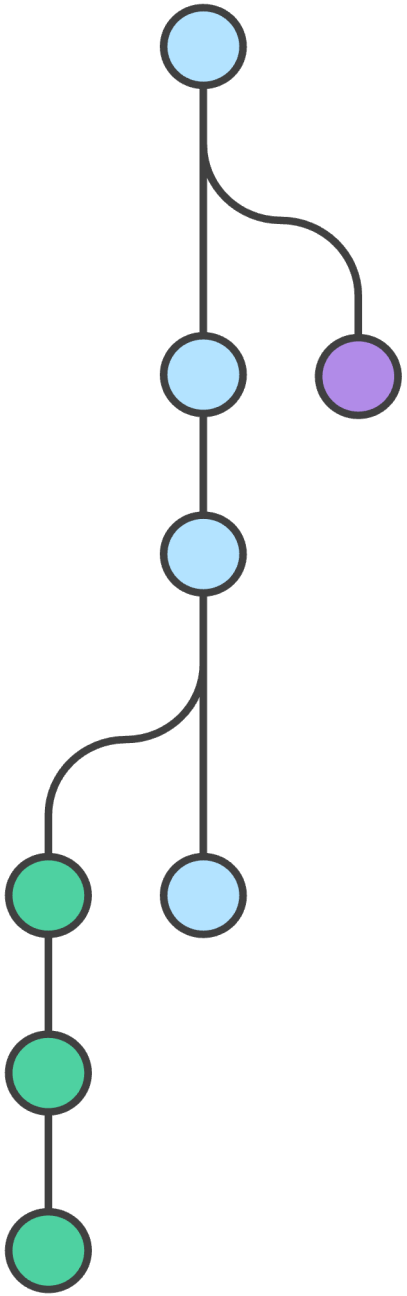
- We advocate for **working directly from your raw data**, and version control your **code.**
  - Code (should be) a clear record of each decision you have made, written for those that come later.
  - You are ensuring reproducibility every time you rerun your analysis.
  - Future revisions, collaborations, and experimentation become easier.



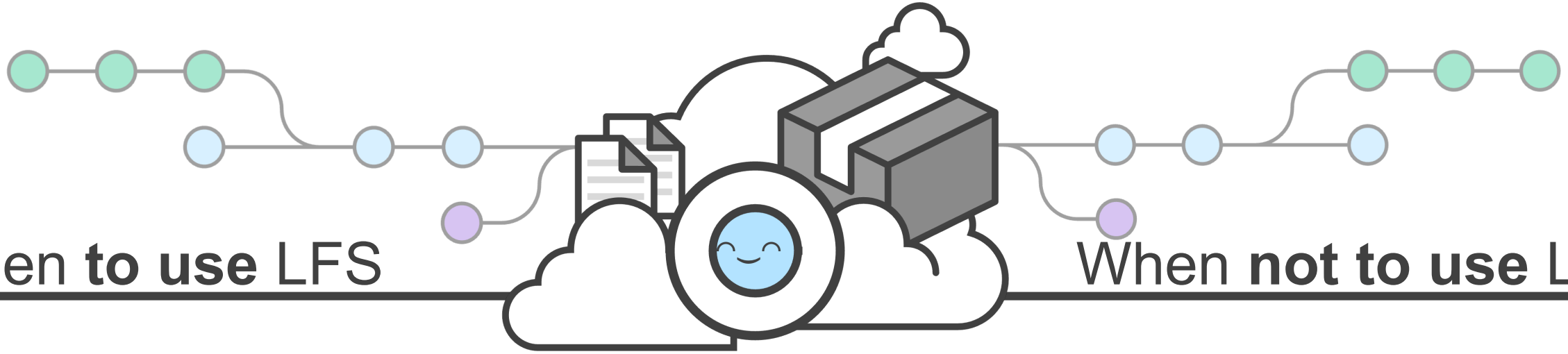
- If you must make manual changes, **a README.md is essential**



- If you must make manual changes, **a README.md is essential**
  - **what** changed; **why** it changed; and **how** it changed
  - A new grad student should be able to recreate the data and analysis with no guidance.



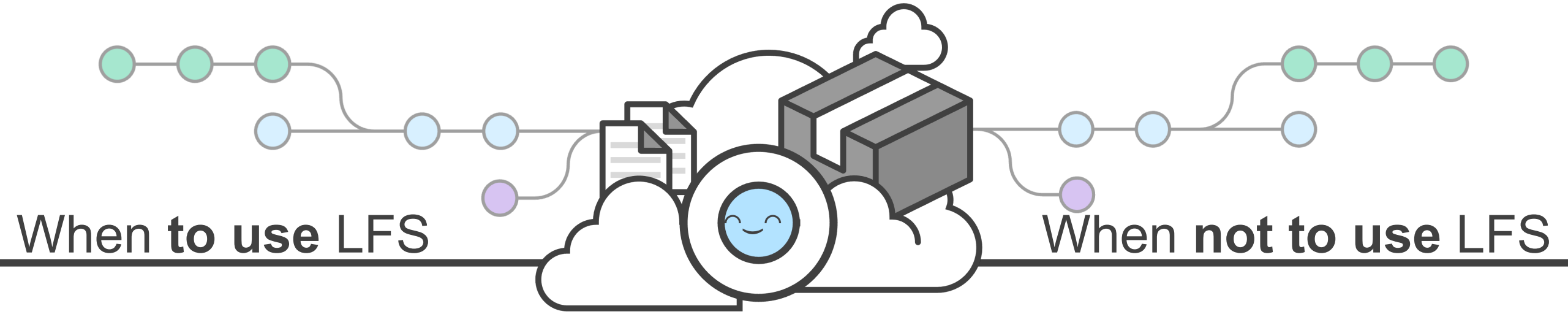
- If you must make manual changes, **a README.md is essential**
  - **what** changed; **why** it changed; and **how** it changed
  - A new grad student should be able to recreate the data and analysis with no guidance.
- If you **have stable data** that you would like to **share coupled** with your code (*e.g.* analysis outputs), you may consider version controlling your data along with your code.



# When **to use** LFS

- **Stable or infrequently changed data**
  - If your data files don’t change very often (e.g. final analysis outputs, design files, benchmarks), LFS can version each new release without forcing every clone to fetch all historic data.

# When **not to use** LFS

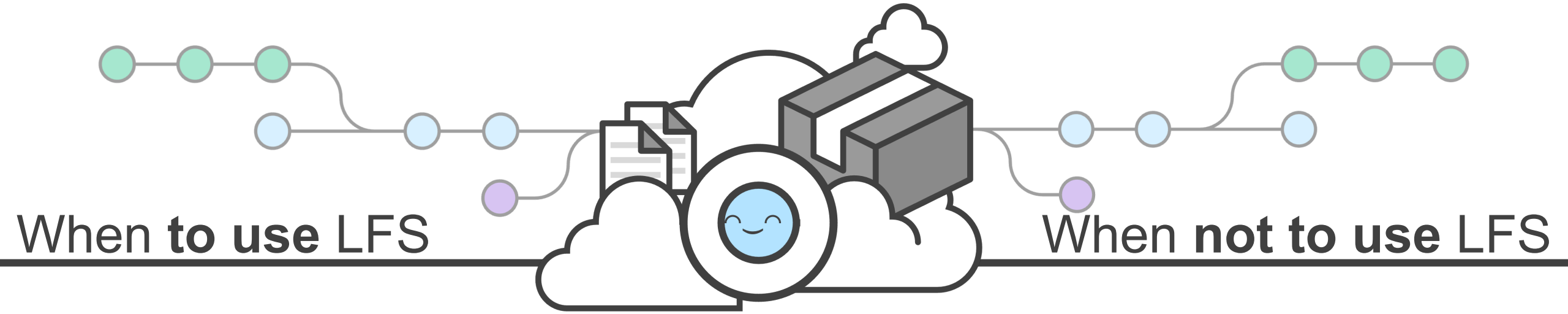


# When **to use** LFS

- **Stable or infrequently changed data**
  - If your data files don’t change very often (e.g. final analysis outputs, design files, benchmarks), LFS can version each new release without forcing every clone to fetch all historic data.

- **Coupling code and medium-size data**
  - When you want to keep code and its data together and share them via Git hosting with your team or your collaborators (Access controls remain unified with the repository).

# When **not to use** LFS



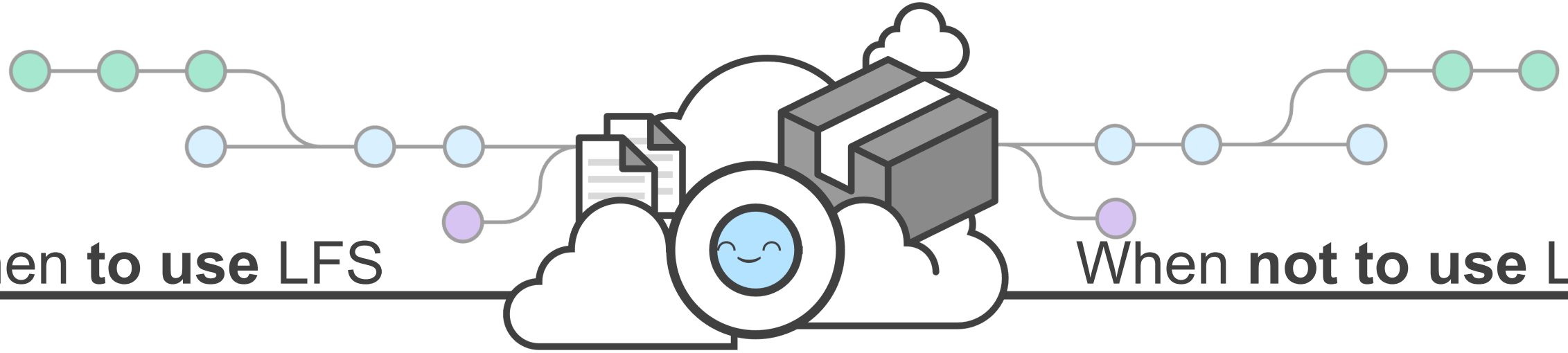
# When **to use** LFS

- **Stable or infrequently changed data**
  - If your data files don’t change very often (e.g. final analysis outputs, design files, benchmarks), LFS can version each new release without forcing every clone to fetch all historic data.

- **Coupling code and medium-size data**
  - When you want to keep code and its data together and share them via Git hosting with your team or your collaborators (Access controls remain unified with the repository).

- **Improving repo performance**
  - If a Git repo has become slow or large due to binary files, enabling LFS can “off-load” them and yield a “significant speedup” on clones/fetches.

# When **not to use** LFS

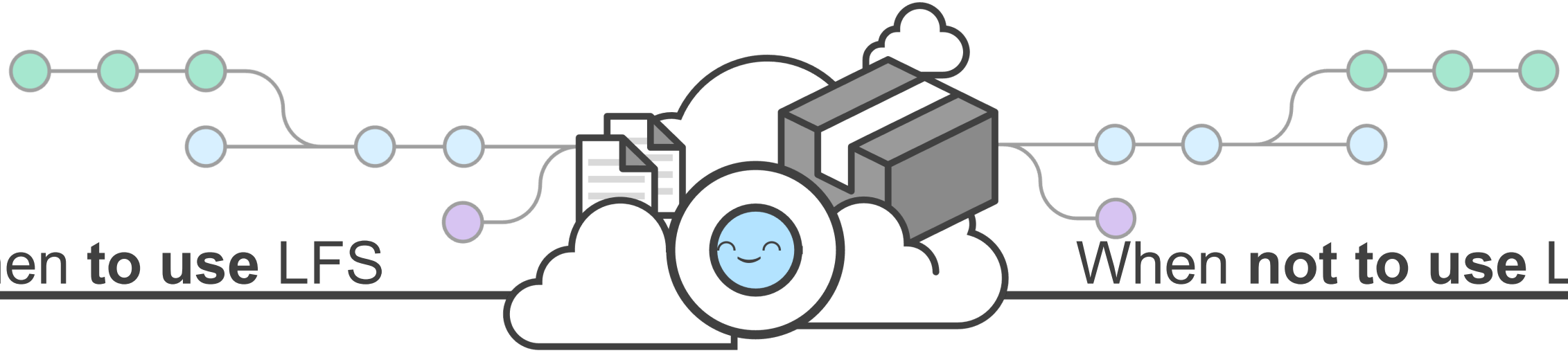


# When **to use** LFS

- **Stable or infrequently changed data**
  - If your data files don't change very often (e.g. final analysis outputs, design files, benchmarks), LFS can version each new release without forcing every clone to fetch all historic data.
- **Coupling code and medium-size data**
  - When you want to keep code and its data together and share them via Git hosting with your team or your collaborators (Access controls remain unified with the repository).
- **Improving repo performance**
  - If a Git repo has become slow or large due to binary files, enabling LFS can "off-load" them and yield a significant speedup on clones/fetches.

# When **not to use** LFS

- **Extremely large or rapidly changing data**
  - Git LFS is only feasible for managing small to medium-sized datasets (e.g. GitHub's LFS limits files to 1 GB by default), and LFS provides no special tooling for big-data workflows

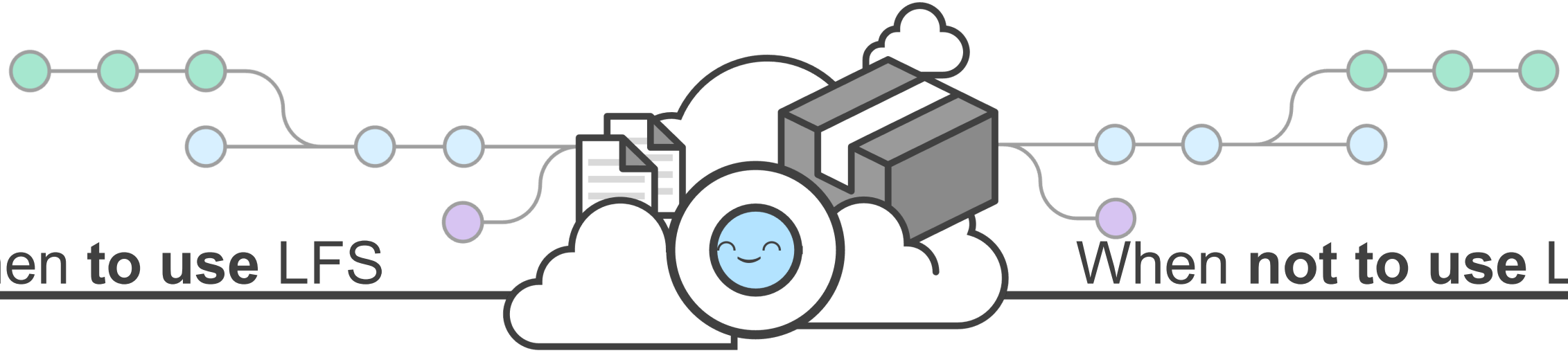


# When **to use** LFS

- **Stable or infrequently changed data**
  - If your data files don't change very often (e.g. final analysis outputs, design files, benchmarks), LFS can version each new release without forcing every clone to fetch all historic data.
- **Coupling code and medium-size data**
  - When you want to keep code and its data together and share them via Git hosting with your team or your collaborators (Access controls remain unified with the repository).
- **Improving repo performance**
  - If a Git repo has become slow or large due to binary files, enabling LFS can "off-load" them and yield a "significant speedup" on clones/fetches.

# When **not to use** LFS

- **Extremely large or rapidly changing data**
  - Git LFS is only feasible for managing small to medium-sized datasets (e.g. GitHub's LFS limits files to 1 GB by default), and LFS provides no special tooling for big-data workflows
- **Need for full history/offline clones.**
  - LFS stores only the **latest** version of each tracked file in Git. The Git history contains only pointers. To retrieve older data versions, users must explicitly fetch them

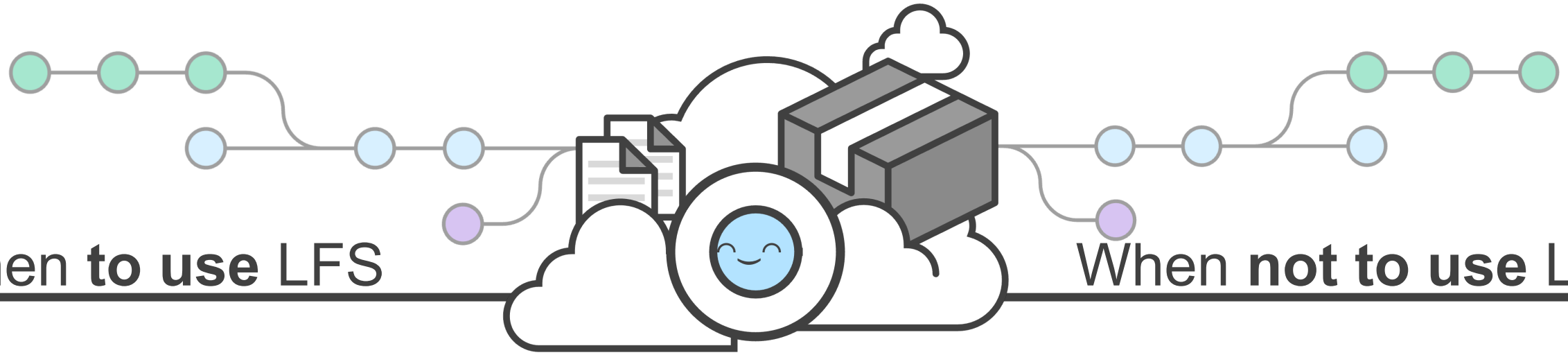


# When **to use** LFS

- **Stable or infrequently changed data**
  - If your data files don't change very often (e.g. final analysis outputs, design files, benchmarks), LFS can version each new release without forcing every clone to fetch all historic data.
- **Coupling code and medium-size data**
  - When you want to keep code and its data together and share them via Git hosting with your team or your collaborators (Access controls remain unified with the repository).
- **Improving repo performance**
  - If a Git repo has become slow or large due to binary files, enabling LFS can "off-load" them and yield a "significant speedup" on clones/fetches.

# When **not to use** LFS

- **Extremely large or rapidly changing data**
  - Git LFS is only feasible for managing small to medium-sized datasets (e.g. GitHub's LFS limits files to 1 GB by default), and LFS provides no special tooling for big-data workflows
- **Need for full history/offline clones.**
  - LFS stores only the **latest** version of each tracked file in Git. The Git history contains only pointers. To retrieve older data versions, users must explicitly fetch them
- **Fine-grained provenance needs**
  - LFS tracks file snapshots but not *how* files were generated. If you have complex data-cleaning pipelines or manual steps, simply dumping the results into LFS won't capture that process.



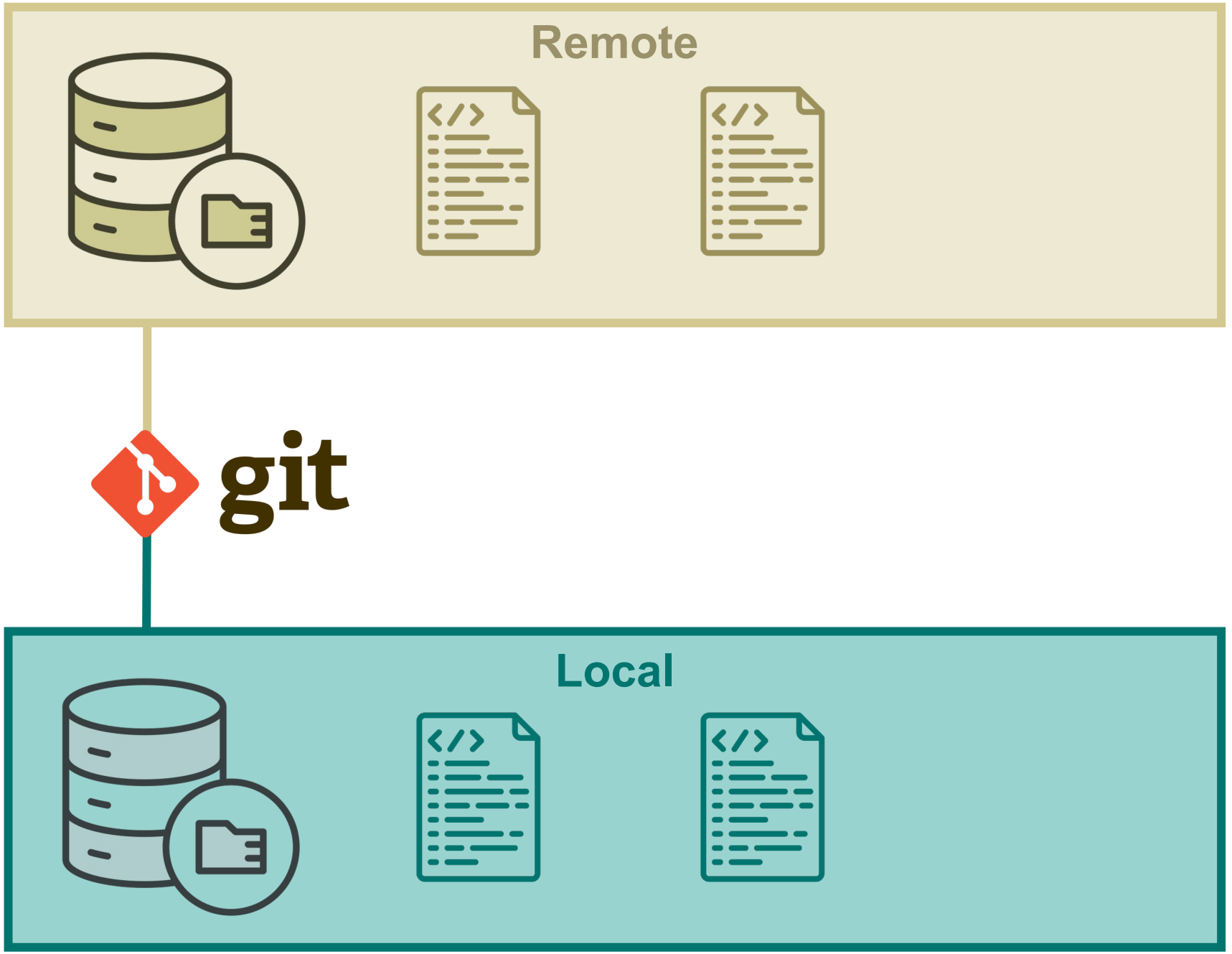
# When **to use** LFS

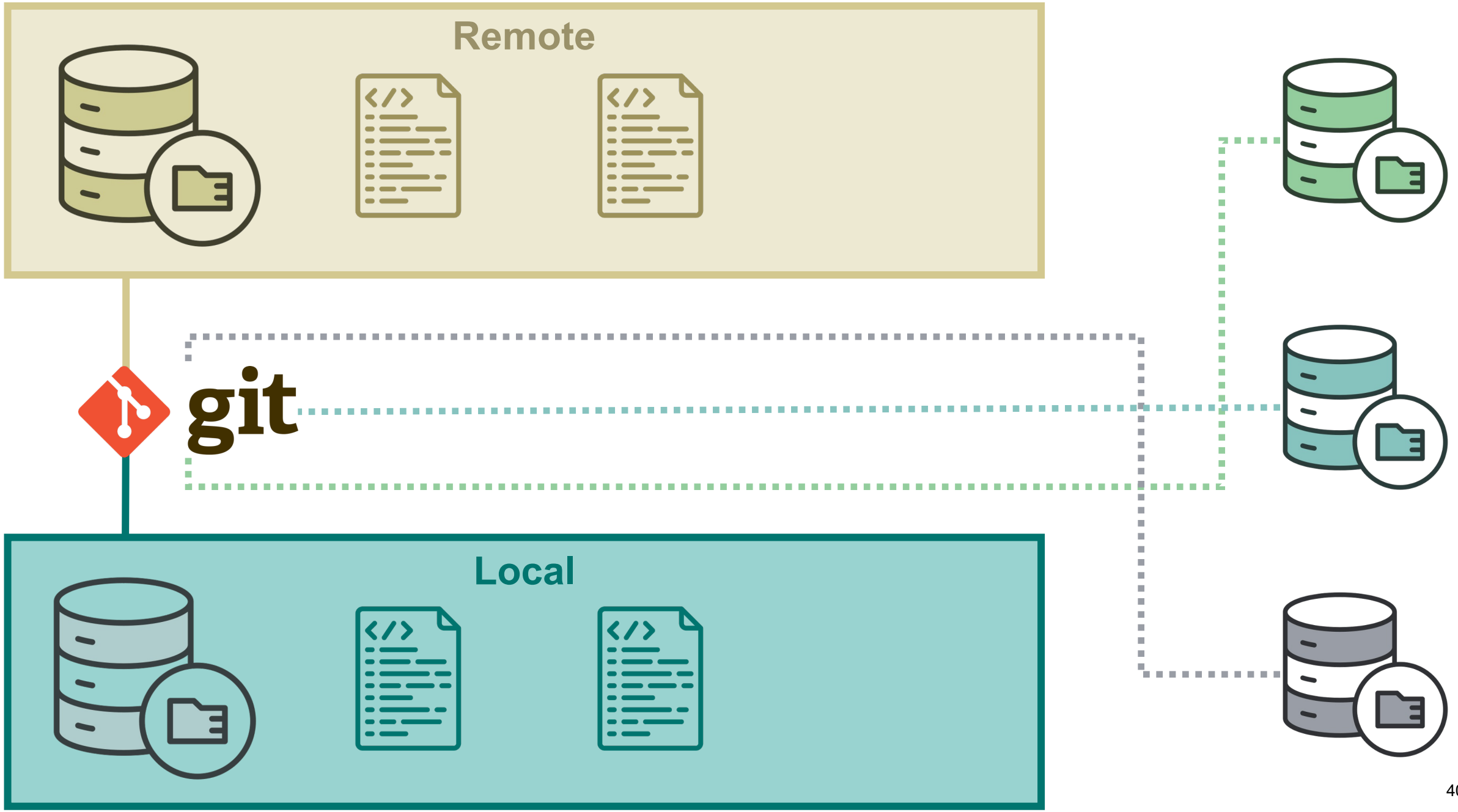
- **Stable or infrequently changed data**
  - If your data files don't change very often (e.g. final analysis outputs, design files, benchmarks), LFS can version each new release without forcing every clone to fetch all historic data.
- **Coupling code and medium-size data**
  - When you want to keep code and its data together and share them via Git hosting with your team or your collaborators (Access controls remain unified with the repository).
- **Improving repo performance**
  - If a Git repo has become slow or large due to binary files, enabling LFS can "off-load" them and yield a "significant speedup" on clones/fetches.

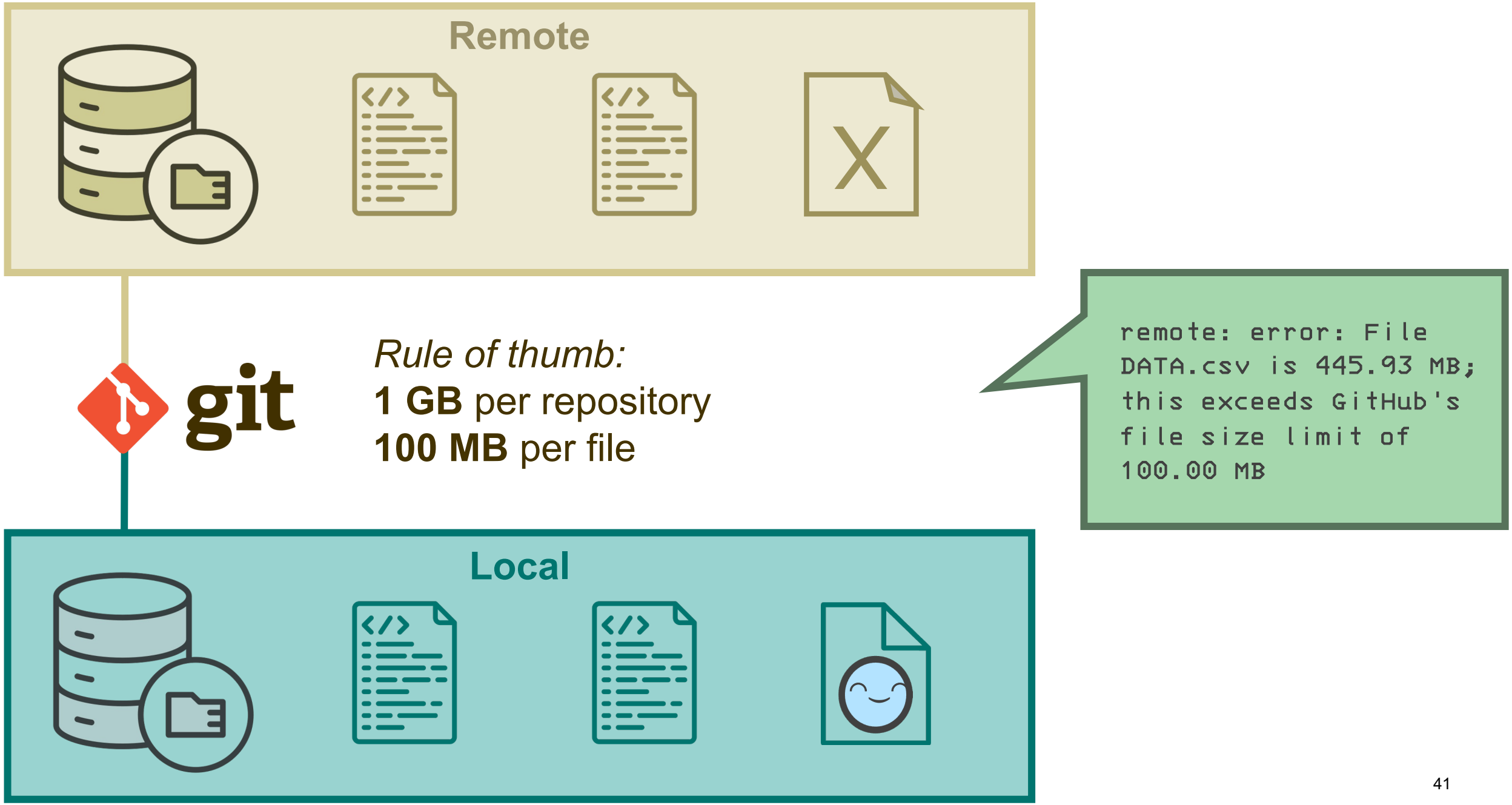
# When **not to use** LFS

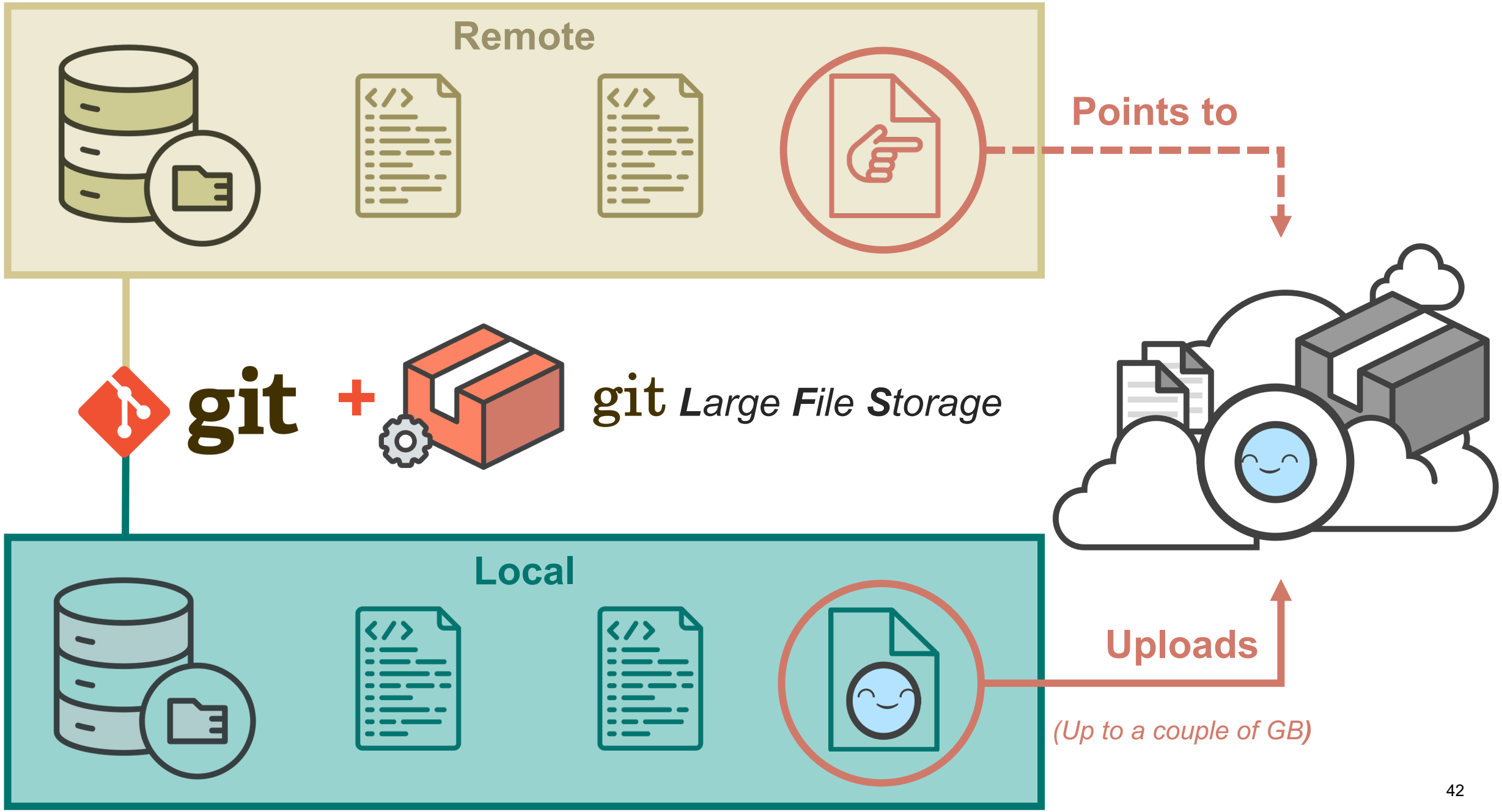
- **Extremely large or rapidly changing data**
  - Git LFS is only feasible for managing small to medium-sized datasets (e.g. GitHub's LFS limits files to 1 GB by default), and LFS provides no special tooling for big-data workflows
- **Need for full history/offline clones.**
  - LFS stores only the **latest** version of each tracked file in Git. The Git history contains only pointers. To retrieve older data versions, users must explicitly fetch them
- **Fine-grained provenance needs**
  - LFS tracks file snapshots but not *how* files were generated. If you have complex data-cleaning pipelines or manual steps, simply dumping the results into LFS won't capture that process.
- **Sensitive or closed data**
  - If your data must stay private (e.g. patient data or proprietary info), you may not be allowed to put it on a Git LFS server, even a private one.


Local


Remote
git
Local


Remote
git
Local


Remote
git
Rule of thumb:
1 GB per repository
100 MB per file
remote: error: File DATA.csv is 445.93 MB; this exceeds GitHub's file size limit of 100.00 MB
Local


Remote
Points to
git
+
git Large File Storage
Local
Uploads
(Up to a couple of GB)

```
git lfs install
```

Download and install the Git command line extension.

Running this will prompt you to set up Git LFS with your user account.

```
git lfs install
```

```
git lfs track "*.csv"
```

Select the file types to manage in each Git repository where you want to use Git LFS.

You can configure additional fizle extensions at anytime.

```
git lfs install
```

```
git lfs track "*.csv"
```

```
git add .gitattributes
```

Now make sure .gitattributes is tracked, which contains these new file type trackers.

```
git lfs install
```

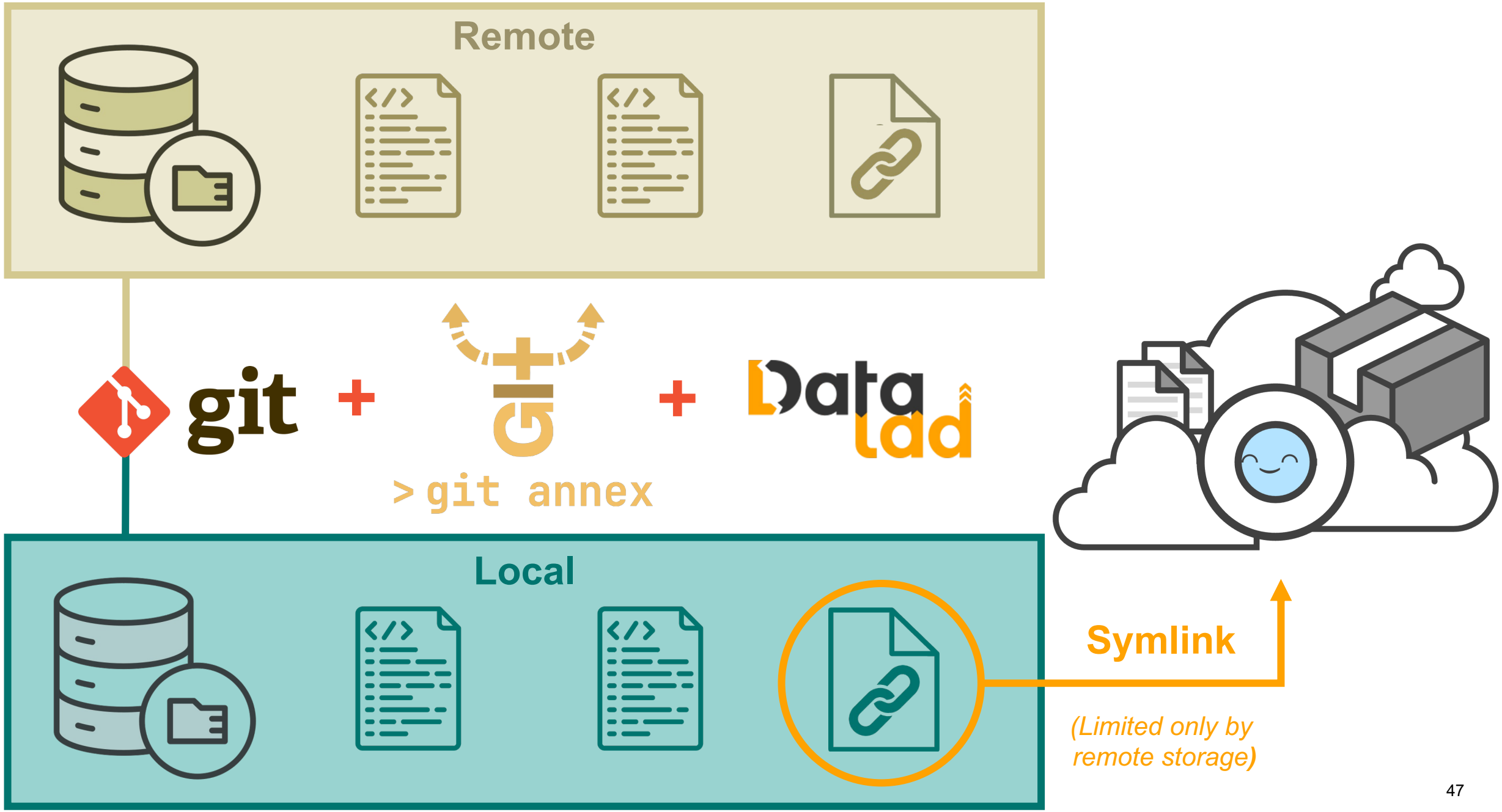
```
git lfs track "*.csv"
```

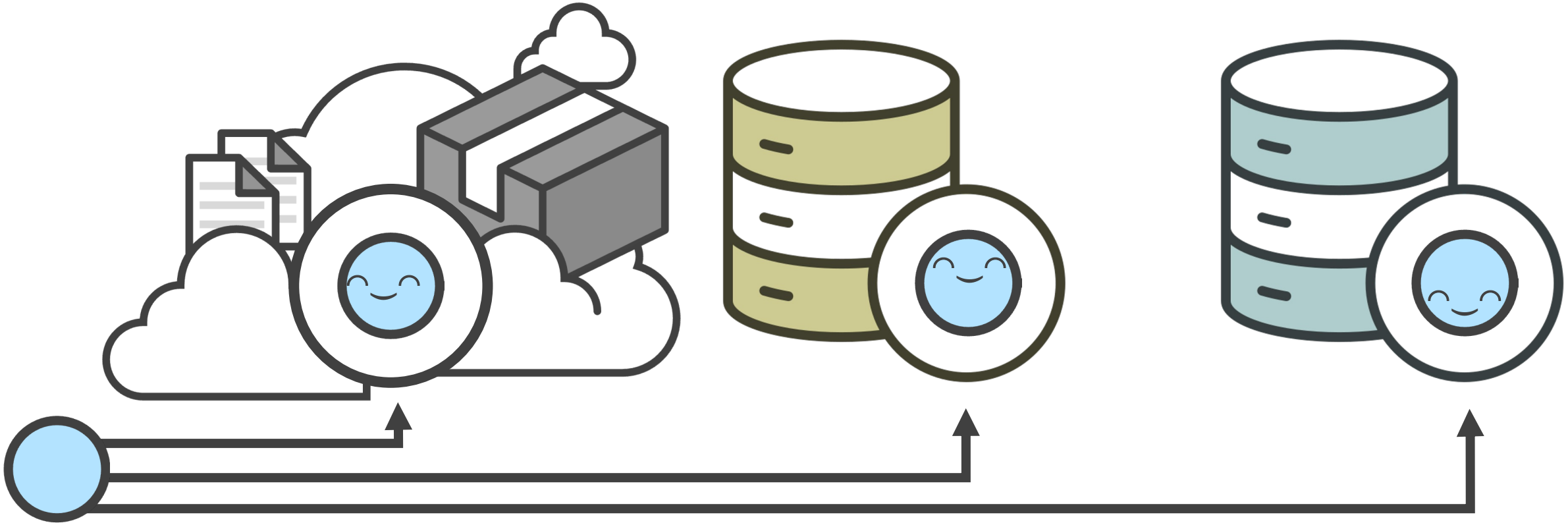
```
git add .gitattributes
```

```
git add DATA.csv
git commit -m "Add DATA file"
git push origin main
```

Now you can just commit and push as you normally would.

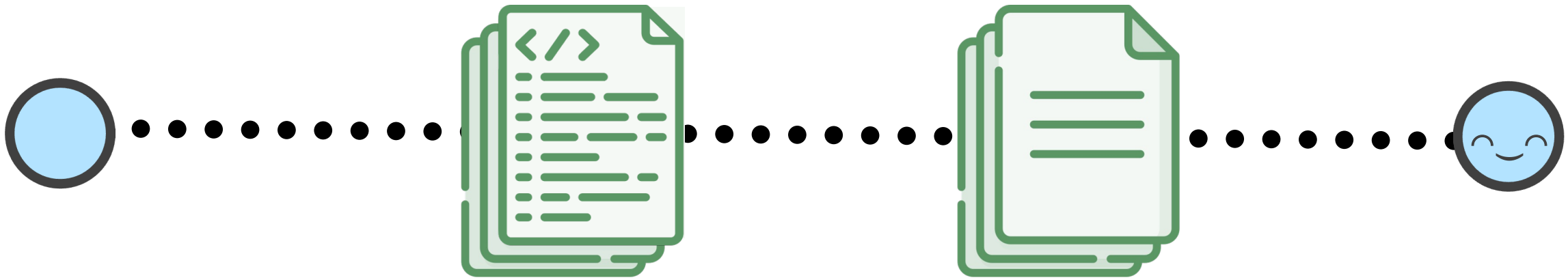
Add your data file, and push to your current branch.


Remote
git
+
> git annex
+
Datalad
Local
Symlink
(Limited only by
remote storage)



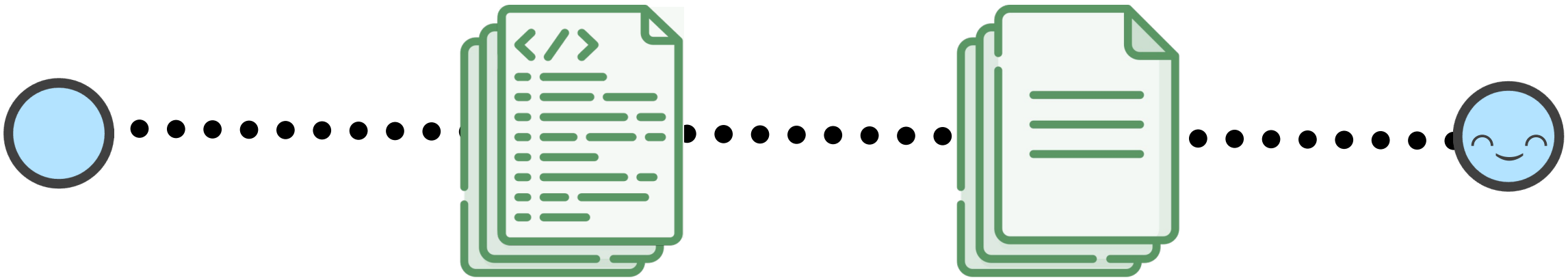
# keep a true raw data file:

**3** copies on **2** different medium, and **1** off-site



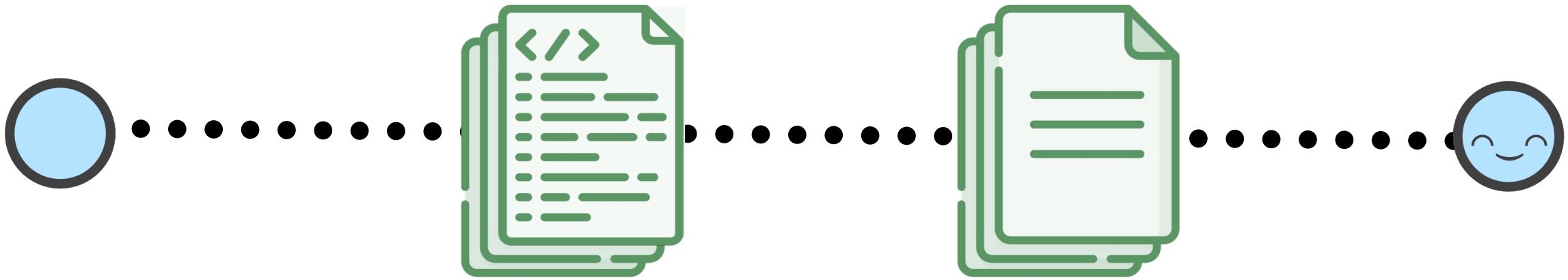
# record every change:

what changed; why it changed; and how it changed



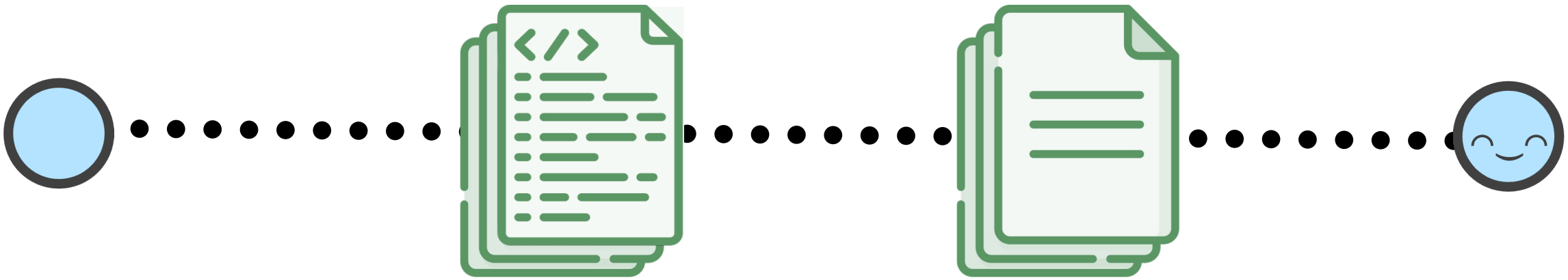
# That can (*ideally*) be

## *well documented code*



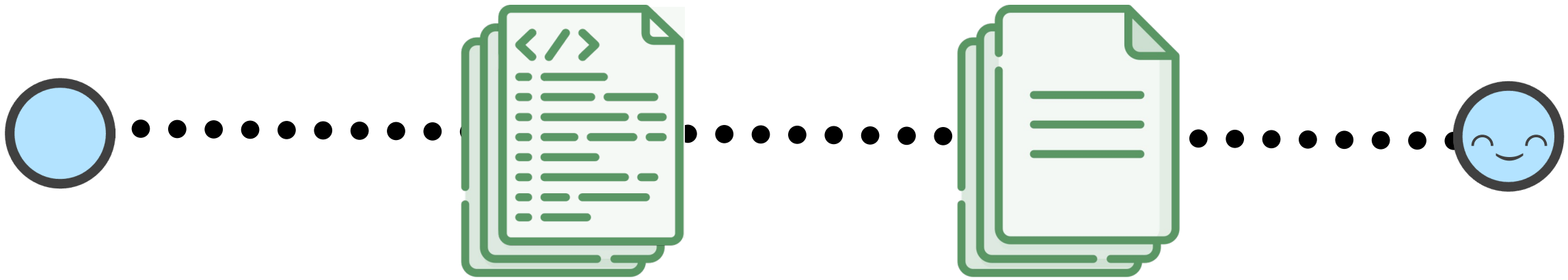
# or (*in limited circumstances*)

## *a detailed ReadMe document*



# build a narrative file structure:

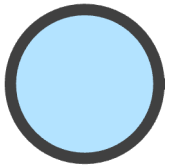
***a complete story. no sequels.***



**Git LFS can add to those narratives,**

***but it alone cannot replace them***

*“ how vulnerable the repository of all our potential once was,*
*how perilous our infancy, how humble our beginnings,*
*how many rivers we had to cross before we found our way. ”*



**Carl Sagan**

Lib4RI

February 11, 2025

# From Raw to Reproducible:

## Versioning Research Data for the Future

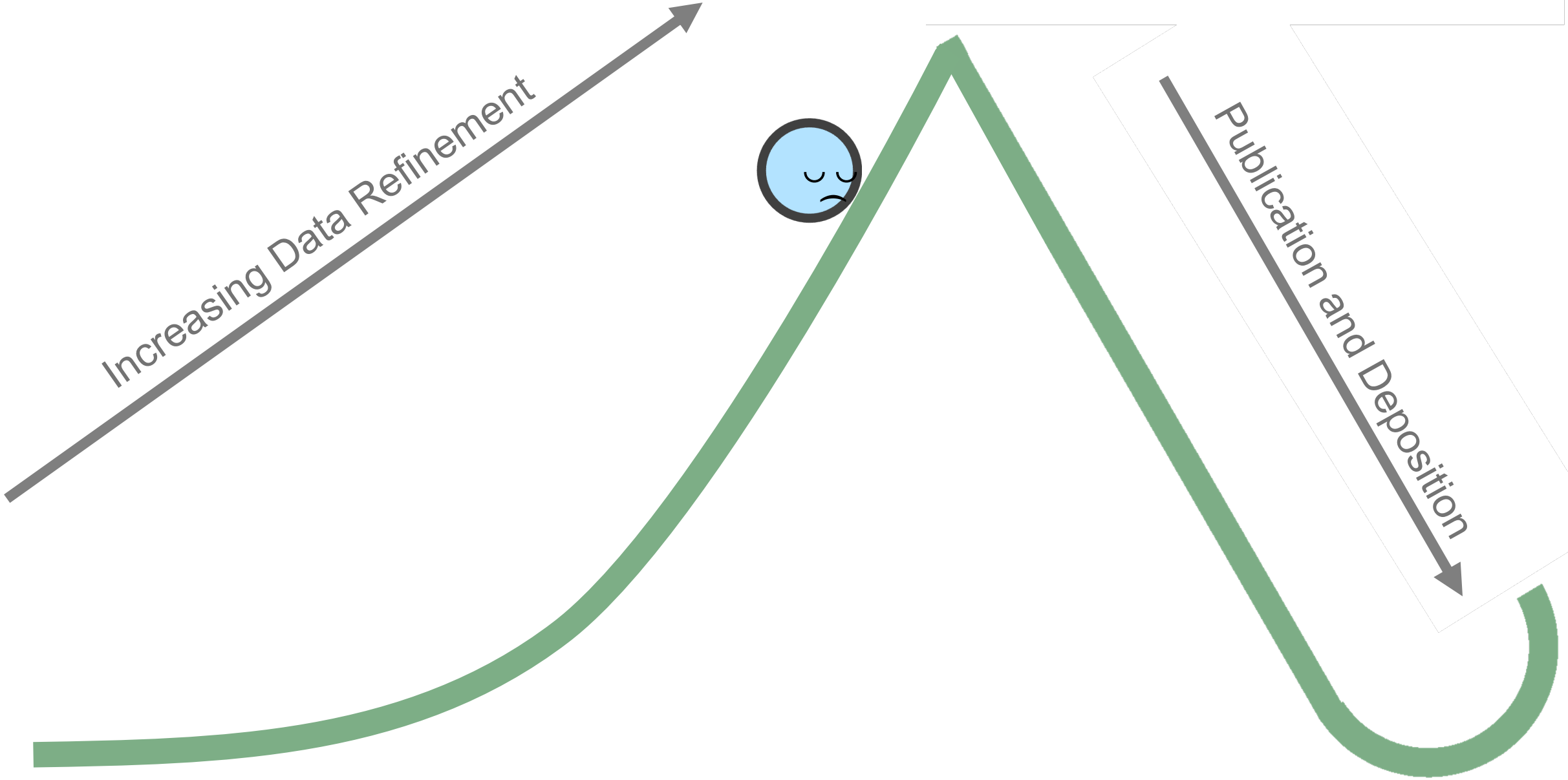
chase.nunez@lib4ri.ch data@lib4ri.ch

**DATA**

**data_CLEANED_FINAL_REVISED_V5.csv**

**SCRIPTS**

**analysis_V5.csv**


Increasing Data Refinement
Publication and Deposition

## Option 1: Eawag homedirectory

- **Advantages**
  - Standard provided for Eawag staff
  - Automatic snapshotting of stored data
  - Secure storage on Eawag/Empa sites
- **Disadvantages**
  - Not well suited for data sharing
  - Only suited for small data (standard storage capacity: 20 GB)
    - But: also Q drive can be used
  - Not very suited for longer term data storage (intermediate solution)

**Typical use case:** Backup of personal data. Project storage for own Eawag projects (with limited data volume).

Eawag home directory: Built-in for small data.

## Option 2: Sharepoint/OneDrive

- **Advantages**
  - Easy sharing of files/folders
  - Working tool suite included
- **Disadvantages**
  - Limited storage capacity (100GB)
  - Only sharing within Eawag

**Typical use case:** Cloud storage for shared project data with Eawag collaborators.

Sharepoint: Built in for medium storage and Eawag sharing

## Option 3: “Messdaten storage”

- **Advantages**
  - On Eawag NAS storage
  - Per project request: volume customizable, TB range
- **Disadvantages**
  - Only accessible on campus
  - Not very suited for collaboration

**Typical use case:** Shared project data within Eawag (larger data volumes).

Contact Eawag RDM

# Option 4: SwitchDrive

- **Advantages**
  - Easy setup and user management
  - Free storage capacity
  - Easily synchronized to physical device
  - Secure storage in Switzerland
  - Easy solution for sharing with Swiss academic collaborators
- **Disadvantages**
  - Limited storage capacity (100GB; but: Switch compute)
  - Data synchronization with external partners requires more setup
  - Not very well suited for large data

**Typical use case:** Cloud storage and sharing for projects with academic collaborators.

Switchdrive: Easiest for sharing with Swiss academics.

## Option 5: S3 bucket

- **Advantages**
  - Flexible storage volume
  - Fastest data transfer speeds
  - Anyone can be given access
  - Can be hosted on Eawag server/Virtual machine
- **Disadvantages**
  - Requires IT support (setting up S3 storage)
  - Limited access configuration (only full access/no access)

**Typical use case:** Cloud storage for very large volumes, with limited project sharing (no complex user management), and need for fast file transfer speeds.

S3 bucket: Best for very large data.

## Option 6: Owncloud/Nextcloud instance

- **Advantages**
  - Flexible storage volume
  - Flexible user access (similar to Switchdrive)
  - Can be hosted on Eawag server/Virtual machine
  - Supported at ETH
- **Disadvantages**
  - Requires IT support (setting up Nextcloud/Owncloud server)
  - More time data management/maintenance

**Typical use case:** Projects with shared data, and complex user management; recommended for longer-running projects only (more complex setup)

Nextcloud/Owncloud: Interesting for long term projects with complex user management.

# README.md

1. **Project Title and Description**
   What the code does, the scientific problem it solves, and the associated paper or research project.
   Version/Date: Latest version release date (YYYY-MM-DD).

2. **Getting Started & Installation**
   Prerequisites: List software, hardware, libraries, and version numbers (e.g., Python 3.9, MATLAB R2023a).
   Installation Steps: Step-by-step instructions, including pip install or conda commands.

3. **Usage & Examples**
   Running the Code: Command-line examples to run the main analysis.
   Parameters: Description of parameters, configurations, or input arguments.

4. **Data Provenance & Structure**
   File Structure: Brief overview of files, folders, and data files.
   Data Source: Where data was obtained, or how it was collected (laboratory, survey, web).

5. **Citations & License**
   License: Important for reuse (e.g., MIT, GPL, CC-BY).
   Citation: How to cite the software and the related scientific publication.

6. **Authors & Contact**
   Contact Information: Name/email of the principal investigator or developer.


# Pros

•Supports multiple remotes that you can store the binaries.

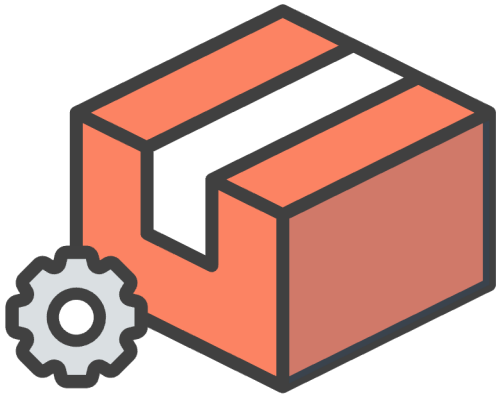
•Larger files can be hosted by multiple backends (S3 buckets, rsync, etc.)

# Cons

•Written for UNIX. Windows support “in beta” and has been for a long time.

•Users need to learn separate commands for day-to-day work

•not supported by github and bitbucket



Git LFS

•Supported by github, bitbucket and gitlab

•Most supported on all OS’s

•Easier  to use.

•automated based on filters

•Requires a custom server implementation to work. A simple ssh remote is not sufficient.