Data Management

May 17, 2023 @ STEADY Workshop

Shiny Brar

he/him/il

Technology Lead &
Software Manager

CHIME/FRB Telescopes

Data Governance

Industry Lingo

- Architecture
- Storage & Operations
- Modelling & Design
- Integration
- Metadata
- Quality
- Documentation
- Security

Data Management

Academic Lingo

You have to do everything →

I will do everything!

I can do somethings…

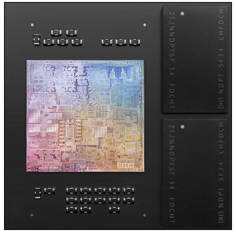
What do I need to do to graduate?

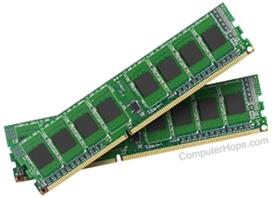
Data

Overview

- Data
  - Storage Mediums
  - File Systems
  - Archetypes
- Management
  - Why?
  - How?
  - Best Practices
- Case Study
  - CHIME/FRB Telescopes

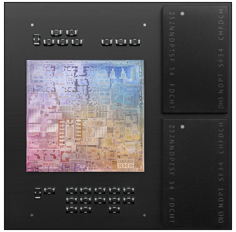
Storage Mediums

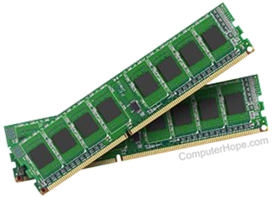




L1/L2/L3 Cache	Random Access Memory	Solid State Drives	Hard Disk Drives	Magnetic Tape
32→256 KB **256KB → 8MB**	**4GB → 128GB**	**120GB → 4TB**	**500GB → 16TB**	**~500TB → 1PB**
instant / ~ps ~100x / 25x > RAM $$$$$	~1 → 100 ns ~10's GB/s $$$$	~1 → 10 us ~0.5 → 3 GB/s $$$	1 → 10ms 100 → 500 MB/s $$	1 → 100 MB/s secs → minutes $
Volatile	Volatile	Non Volatile No Moving Part	Non Volatile Spinning Platter	Non Volatile Basically a Walkman!

Storage Systems





L1/L2/L3 Cache

Lines of data!

——————

□

Hit or Miss

Random Access Memory

Matrix Structure
Rows x **Columns**
Addressable Grid

Random Access
Bit/Byte Level Address

Solid State Drives

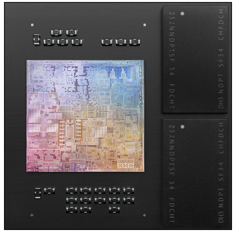
Hierarchical Structure

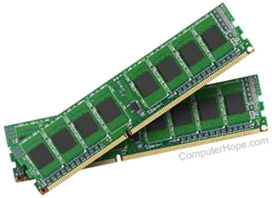
- **Boot Sector**
- **File System Metadata**
- **File Allocation Structure (File Systems)**
- **Directories**
- **File Data**

Hard Disk Drives

Magnetic Tape

Storage File Systems





L1/L2/L3 Cache

Lines of data!
——————
□

Hit or Miss

Random Access Memory

Matrix Structure
Rows x Columns
Addressable Grid

Random Access
Bit/Byte Level Address

Solid State Drives

Hard Disk Drives

Magnetic Tape

Hierarchical Structure

- **Boot Sector**
- **File System Metadata**
- **File Allocation Structure (File Systems)**

Directories
Files
Pirated Gaming Data etc.

Storage File Systems

- Hierarchy
  - Fiction / Non-Fiction / Topic ➔ Folders / Sub-Folders
- Metadata
  - Spine / Book Cover ➔ Name, Size, Type of File
- Allocation
  - Books on a Shelf ➔ Files on storage medium
- Access
  - Locate & Borrow ➔ Open, Read & Write
- Permissions
  - Dark Art Sections!

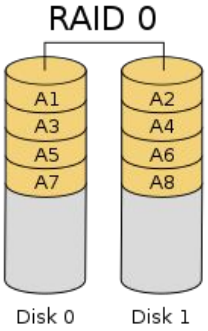
Storage File Systems

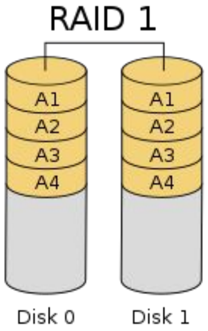
FAT	File Allocation Table – FAT32 / exFAT Windows, macOS, Linux	• Simple & Lightweight • Max file size (4GB) & Max partition (2TB) • Lacks permissions
NTFS	**New Technology File System** **Mainly Windows**	• Provides permissions / encryption / compression • Improved reliability • Read only access on other OS
APFS	**Apple File System** **macOS & iOS**	• Designed for modern storage, e.g. SSD's & Flash • NTFS + Snapshotting & Shared Spaces • Read only access on other OS
ext4	**Fourth Extended File System** **Linux**	• Performance & Scalability • All the above features • Limited Compatibility
ZFS	**Zettabyte File System** **Advanced**	• Data Integrity & RAID like capability • Large Scale Deployments • Native support for macOS & Windows

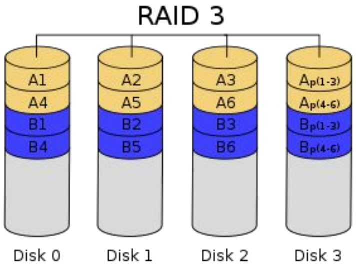
Storage Archetypes: RAID

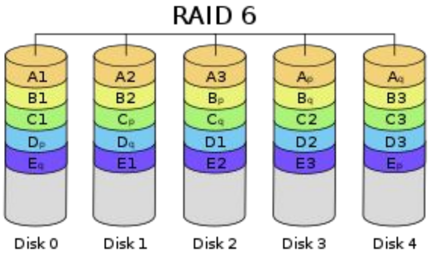
- Redundant Array of Independent Disks
- Combines multiple physical storage systems into a single logical unit
- Improves redundancy (parity), performance (stripe) or both.

RAID 0 | RAID 1 | RAID 3 | RAID 6









Storage Archetypes – RAID

Redundant Array of Independent Disks

- Combines multiple physical storage systems into a single logical unit
- Improves redundancy (parity), performance (stripe) or both.

RAID 0 | RAID 1 | RAID 3 | RAID 6

```
sudo apt-get install mdadm
sudo mdadm --create /dev/md0 --level=0 --raid-devices=2 /dev/sdX1 /dev/sdY1
sudo mkfs.ext4 /dev/md0
```

Storage Archetypes – Zettabyte File System

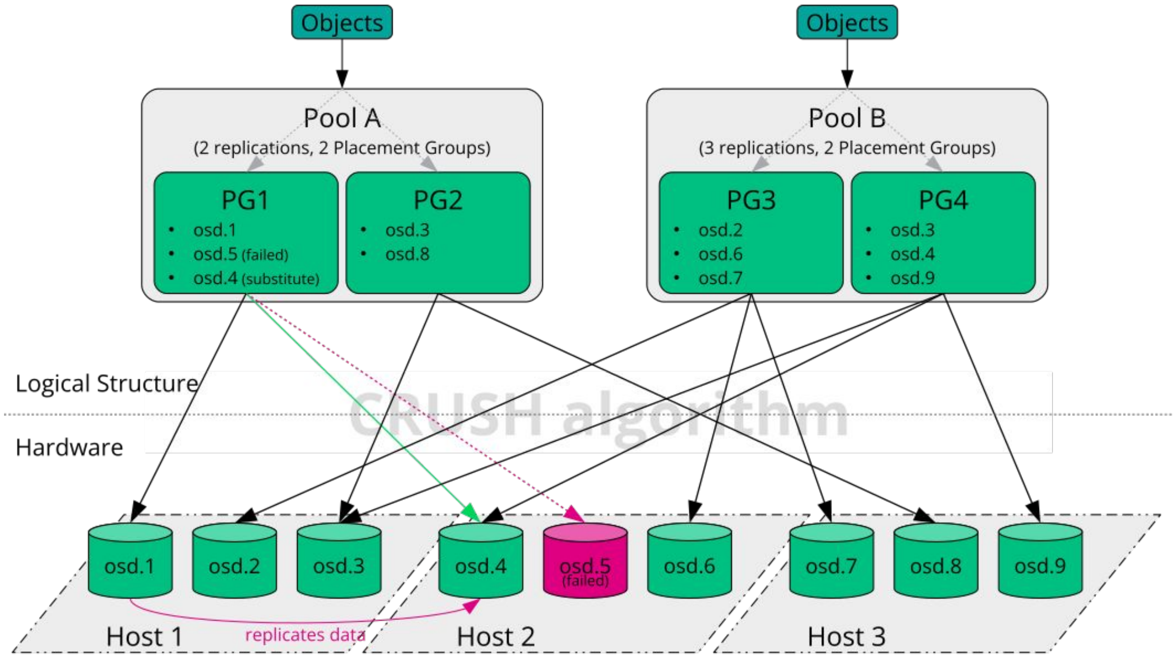
- The Librarian who manages with remarkable efficiency & intelligence.
- Comprehensive: File System + Volume Manager
- Operates at higher levels, sitting on top of storage media or RAID
- Integrity checks via checksums
- Snapshots, clones, deduplication, compression, scalable
- Copy-on-Write!
- Not simple but simpler than RAID

Storage Archetypes – Limitations of File Systems

- File Size & Count
- Total Storage Capacity
- Interoperability
- Metadata Overhead
- Limited to 1 node/server/computer
- Practical limit of ~1-2PB

Storage Archetypes: Massively Scalable Storage

- Infinitely Scalable
- Store objects not files
- Objects served as URLs
- CRUSH Algorithm
  - Deterministic Mapping
  - Failure Aware
  - Load Balancing
- E.g. Google Drive.



Management

Why manage data?

Why manage data?

- Ensuring Accuracy
  - By organizing and documenting your data you can easily track changes
  - Ensures integrity of research
- Completeness
  - Standardized protocols of collecting and storage of data
  - Reduces risk of missing critical information
- Accessibility over time
  - Access and retrieval even after significant amount of time
  - Longevity of research!
- Sharing & Collaboration
  - Fosters transparency & reproducibility
  - Validation of research!
- Avoid Data Loss
  - It's not important, unless it was.
  - All your research should not be on your laptop!

Data Accuracy

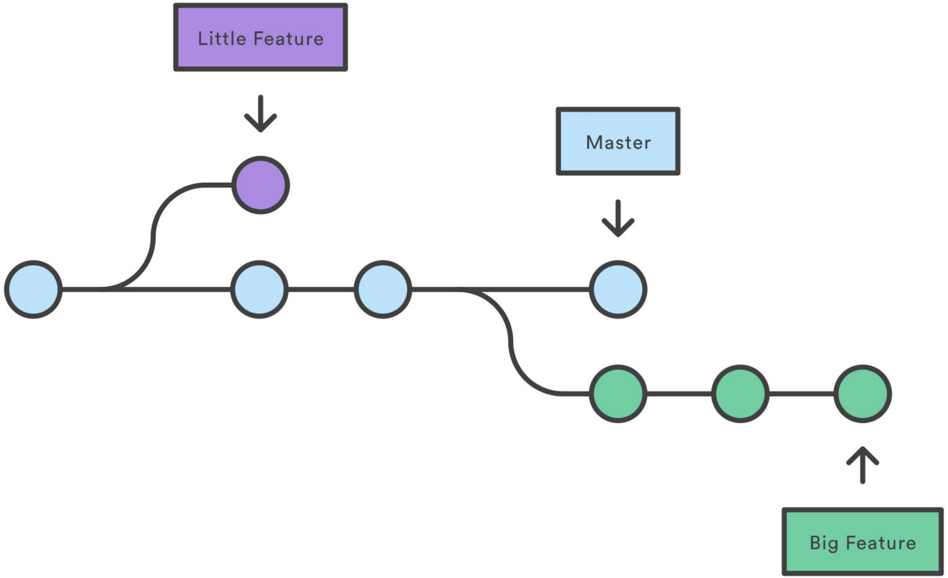
- Use Version Control to Tag data

```python
import subprocess
import h5py
import pkg_resources

# Get the Git commit hash
git_hash = subprocess.check_output(['git', 'rev-parse', 'HEAD']).decode().strip()
# Get the version of your Python package
package_version = pkg_resources.get_distribution('mypackage').version
# Open the HDF5 file in write mode
with h5py.File('data.hdf5', 'a') as f:
    # Create or update a dataset in the header group to store the metadata
    header_group = f.require_group('/header')
    header_group.attrs['git_hash'] = git_hash
    header_group.attrs['package_version'] = package_version
```

Data Accuracy

- Use Version Control to tag data
  - Traceability
  - Enables Collaboration
  - Pinpoint introduction of bugs
  - Safe place to experiment



Data Accuracy

- Use Version Control to tag data
- Hierarchical Data Structure
  - Store data in folders & subfolders …

    ➜ `<project>/<datasource>/<datatype>/<instrument>/…`

Data Accuracy

- Use Version Control to tag data
- Hierarchical Data Structure
  - Store data in folders & subfolders …
  - Use meaningful naming conventions
    - ➜ Consistent naming conventions
    - ➜ Include metadata in the filename e.g. `analysis_success.h5`
  - Parametrize and automated file paths
    - ➜ Where ever possible, automate file path generation

Data Accuracy

```
import os
import datetime
import subprocess

# Parameters
dataset_name = "steady"
subset_name = "data_management"
file_extension = ".csv"

# Get the current date in YYMMDD format
current_date = datetime.datetime.now().strftime("%y%m%d")
# Get the Git commit hash
git_sha = subprocess.check_output(['git', 'rev-parse', 'HEAD']).decode().strip()
# Define the base directory where the files will be stored
base_directory = "/path/to/data"
# Generate the file path using placeholders or variables
file_path = os.path.join(
    base_directory,
    dataset_name,
    current_date,
    f"{subset_name}_{current_date}_{git_sha[:7]}{file_extension}"
)
```

Data Accuracy

- Use Version Control to tag data
- Hierarchical Data Structure
  - Store data in folders & subfolders ...
  - Use meaningful naming conventions
  - Parametrize and automated file paths
  - Consider file system limitations
    - ➔ Character Restrictions: Windows does not allow : `<, >, :, ", /, \, |, ?, and *.`
    - ➔ Case Sensitivity & Path Separator: `/ vs. \`
    - ➔ Max File Length: `Windows 256 // Linux 4096`
  - Python Note: Create virtualenv in folders

Data Accuracy

- Use Version Control to tag data
- Hierarchical Data Structure
  - Store data in folders & subfolders …
  - Use meaningful naming conventions
  - Parametrize and automated file paths
  - Consider file system limitations
  - Create the highest bisecting hierarchy

  ➔ You should be able to go from the biggest to smallest data product intuitively

Data Completeness

- Use data models where possible
  - ➡ For analysis parameters
  - ➡ When saving data
  - ➡ When loading data
  - ➡ Especially when transforming!

```
from pydantic import BaseModel, Field, FilePath
import yaml

# Define the data model using pydantic
class DataModel(BaseModel):
    datapath: FilePath(exists=True)
    iterations: int = Field(..., gt=0, lt=10)
    threshold: float = Field(..., ge=0, le=1)

# Load YAML data into the data model
def load_parameters(file_path):
    with open(file_path, 'r') as f:
        config_data = yaml.safe_load(f)
    parameters = DataModel(**config_data)
    return parameters

# Example usage
file_path = 'data.yaml'
data_model = load_data_model_from_yaml(file_path)
print(data_model)
```

Data Accessibility / Sharing & Collaboration

- Document common practices
  - ➜ Include a README.md with the code that analyzes data
  - ➜ Define the expected input / output data structures
- Use common and stable data formats from your field
  - ➜ `csv`, `json`, `par`, `hdf5`, `fits` etc.
  - ➜ e.g. even though commonly used, would advise against `npy` format.
- Use packages and formats with the largest community support
-

Avoid Data Loss

- Redundant Storage Archetypes
- Regular Data Backups
  - ➔ Local Backups > Cloud Backups
  - ➔ Cloud syncs can cause 10-100x slow down in access time
- Uninterrupted Power Supply (UPS)
- Robust Security Options
  - ➔ When in doubt do not give write permissions
  - ➔ Generally advise against password access, instead always use ssh keys
- Training & Awareness
  - ➔ Mean Time Before Failures (MTBF)

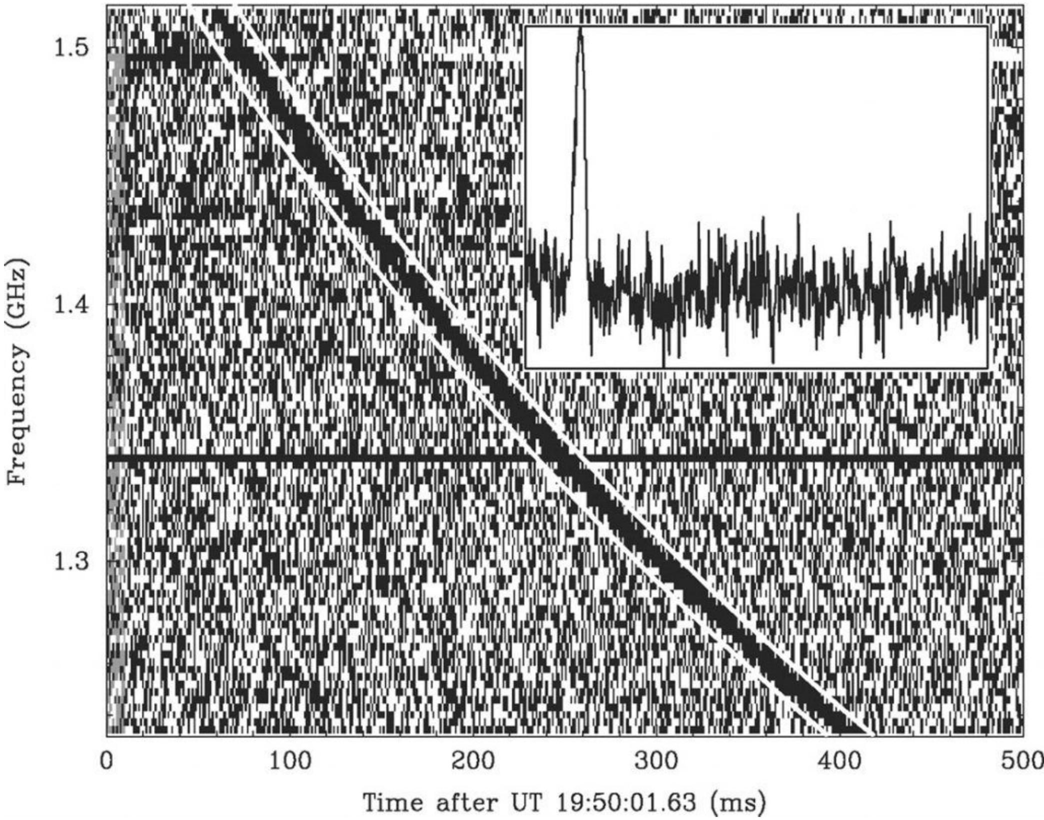
Best Practices

- Time Rule: I/O should take no more than 10% of compute time
  - ➜ Faster storage medium
    - ➜ Faster storage archetype
      - ➜ Split your saved data products
- Size Rule: Depends on your budget, but…
  - ➜ ~10GB ➜ SSD
  - ➜ ~10TB ➜ HDD
  - ➜ ~10PB ➜ Tape
- More I/O Speed
  - ➜ Read the Time Rule
- Always be redundant or backed up!
- Documentations

Case Study: CHIME/FRB Telescopes

Fast Radio Bursts

- Transient, ms-scale
- Astrophysical phenomenon
- Intense pulse of radio waves
- Extragalactic in origin.
- Enormous energy.



CHIME Telescope

CHIME/FRB Telescopes: Data

- 4 Telescope Sites
- 2.2 TB/sec ➔ 190 PB/day ➔ 69 EB/year
- ~ Every computer / phone / video stream for entirety of Canada

- 69 EB/year reduced to ~500PB/year
- Search this data to find FRB's
- All FRB data from previous 5 years ➔ 750 TB
- ~0.002% of all data ➔ ~5 minutes of data.
- Everything else is discarded.

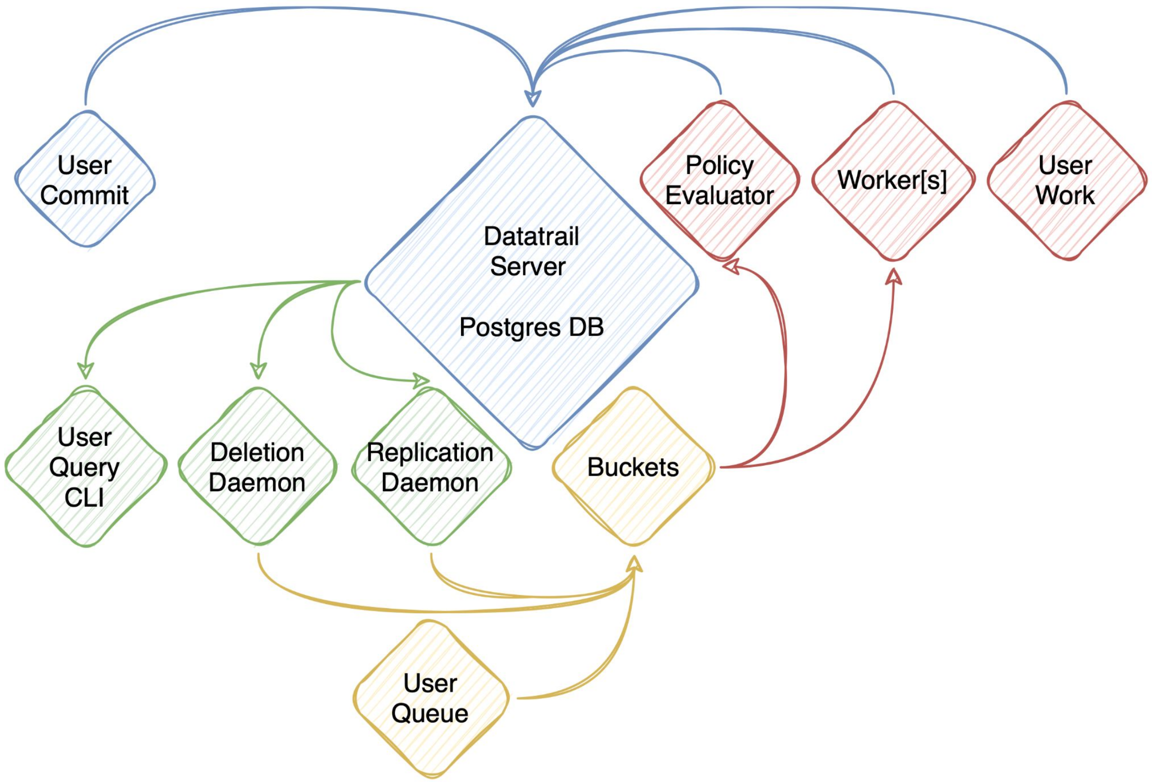
CHIME/FRB Telescopes:
Storage Medium / File System / Archetype

- ~23 TB RAM spread over ~400 nodes
- ~40TB High Speed Flash Storage
- (60+40) x 10TB HDDs = 1 PB Raw HDD Storage
- Configured ZFS File System
- RAID z5 with 10% Hot Spares
- Defaults User Permissions: Read Only
- Parameterized Path Structure
  `/data/<telescope-site>/<data-product>/[raw|processed]/YYYY/MM/DD/astro_[event]/*.h5`
- Data backed up in ~realtime to CANFAR (Compute Canada)

What is Datatrail?

- Data management platform built for CHIME
- Scalable to run at multiple sites - CHIME, outriggers, future outriggers?
- Registers data into database
- Handles deletion and replication between sites
- Policy driven approach to products
    - Allows different rules to be applied to different types of data
        - Eg. Classified FRBs backed up and kept forever
        - RFI stays local until deleted a few weeks later

Datatrail: Overview



Interacting with Datatrail

- Interaction with Datatrail via CLI

- Users can:
  - List all datasets in Datatrail
  - Download datasets
  - View dataset policies
  - List where data is stored
  - See number of files and file size

```
>> datatrail --help
Usage: datatrail [OPTIONS] COMMAND [ARGS]...

  Datatrail Command Line Interface.

Options:
  --help  Show this message and exit.

Commands:
  config     Datatrail CLI Configuration.
  list (ls)  List scopes & datasets
  ps         Details of a dataset.
  pull       Download a dataset.
  version    Show versions.
```

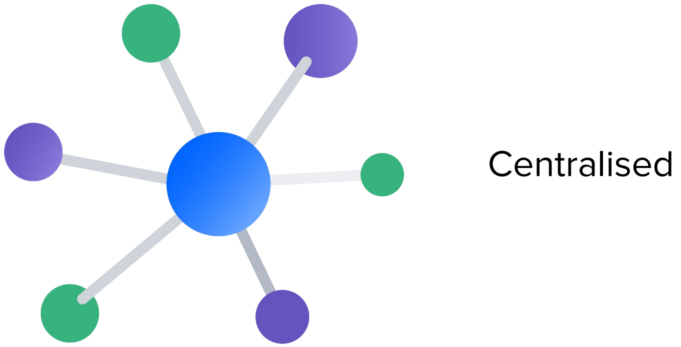
Version Control - Types

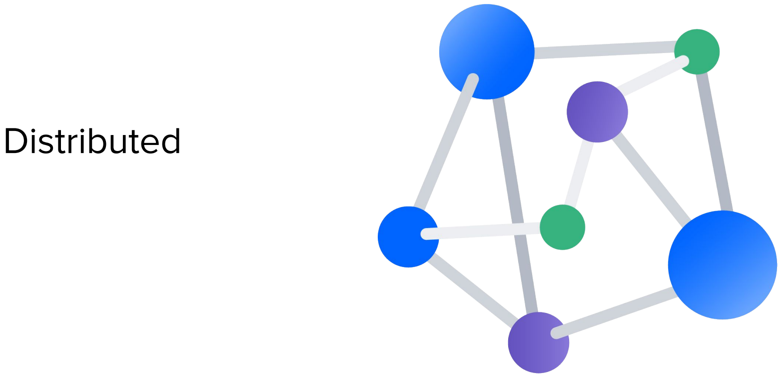
Centralised:

- Single complete copy
- Users write to main branch
- Unavailable while checked out

Pros/Cons:

- Works well with large files
- Easier to understand
- One point of failure
- Lack of stability
- Online





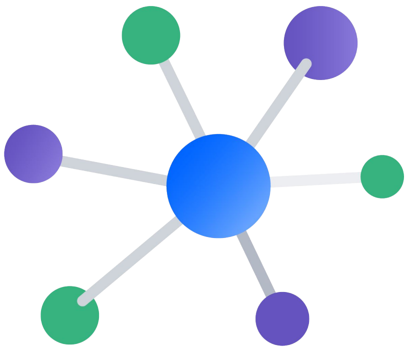
Version Control - Types

Distributed:

- Full copy of repository checked out
- Commit, branch, and merge locally
- Requires more storage space

Pros/Cons:

- Backups
- Faster workflow
- Offline
- Less intuitive
- More prone to conflicts



Centralised

Distributed