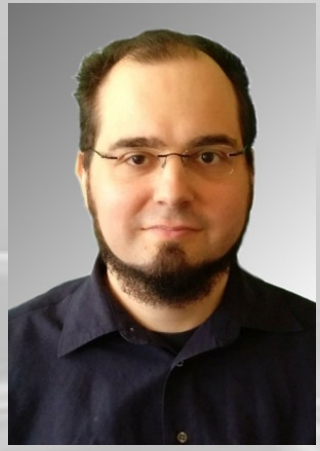
Miniymposium: Leveraging Data Lakes to Manage and Process Scientific Data

Hendrik Nolte, Julian Kunkel

Your Organizers



Julian Kunkel

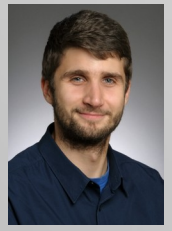
HPC-Team Lead @ GWDG

Prof. at University of Göttingen

Hendrik Nolte

HPC-Team @ GWDG

Agenda



- Hendrik Nolte: A FAIR Digital Object-based Data Lake Architecture to Support Various User Groups and Scientific Domains

- Rihan Hai: Data Integration in Data Lakes

- Pegdwendé Nicolas Sawadogo: Enabling Industrialized Analysis of Textual Documents in Data Lakes

- Mark Greiner: Utilizing Data Lakes for Managing Multidisciplinary Research Data

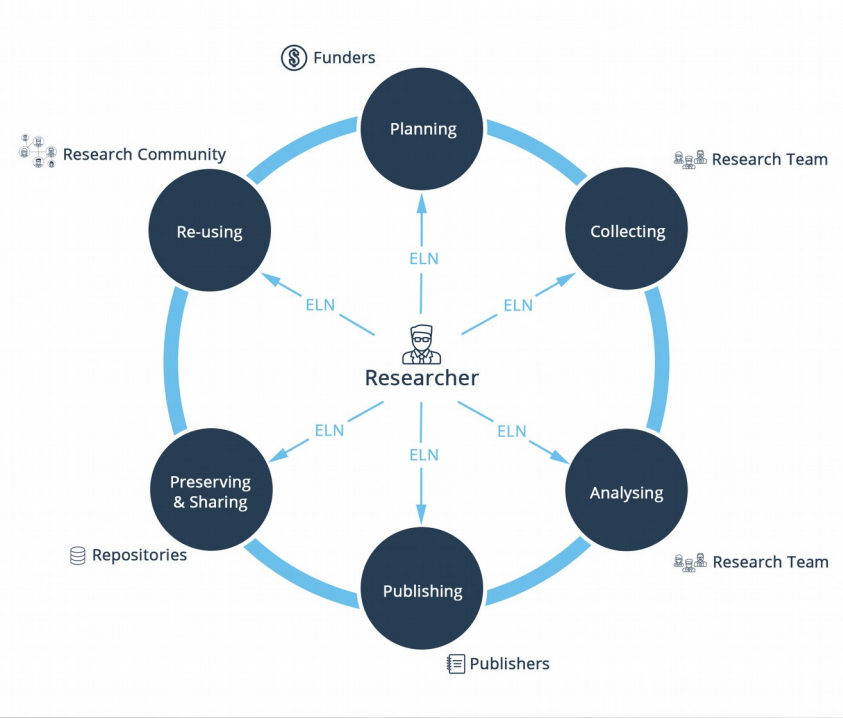
A FAIR Digital Object-Based Data Lake Architecture

Hendrik Nolte, Piotr Kasprzak, Julian Kunkel, Philipp Wieder, Ramin Yahyapour

Motivation Data Lake

Typical requirements from researchers

- Central data storage
- Search capabilities
- Enforced governance
- Data sharing
- Reproducible processing
- High productivity/usability



https://www.labfolder.com/guide-research-data-management/

Often research data management on production systems is far from this

Example Use Case

Consider MRI patient data (simplified) in the clinical environment

- Metadata
  - Patient information: name, age, gender
  - Measurements (e.g., MRI, blood tests): instrument, tracer, data, …,
- Data
  - Original data, e.g. MR images in DICOM format
  - Inferred data from expert systems for decision support
    - e.g., segmentation via freesurfer
  - Generated reports by doctors
- This data/metadata is normally managed via PACS

Use cases

- Creation of the diagnosis for a single patient
- Tracing the development for this patient over time
- How about utilizing data for additional purposes?

Exploiting the Data Treasure

Additional use cases in order to support patients optimally

- Interactive analysis of the patient's data using visual analytics
- Learning from the cohort of data (e.g. find similarities for best treatment)
  - Requires searching patients expressing similar features
- Researching new methods for decision support
  - Using Deep Learning (best: multiple algorithms)
- Integrating additional data sources (weather, geo-location, …)

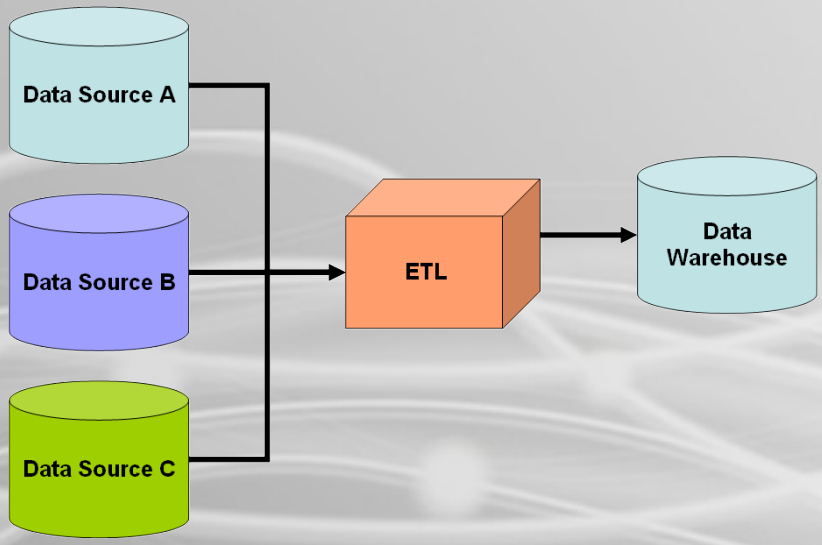
The UMG-MeDIC is a good approach for a research infrastructure

- It shares many characteristics of a data lake and follows FAIR principle
- The optimal satisfaction of the requirements is a moving target
  - e.g., efficient processing of analysis workflows
- Therefore, let's investigate challenges/approaches for these problems

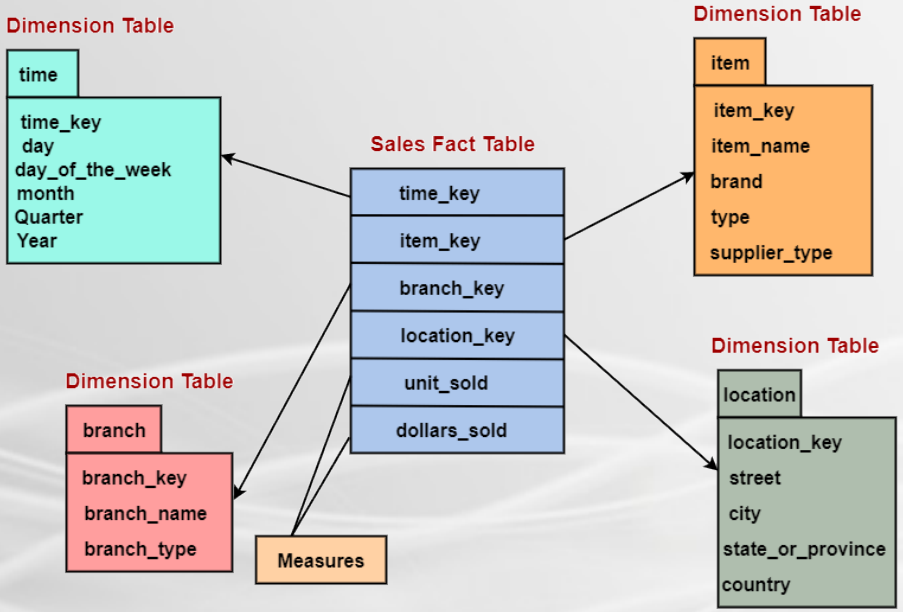
Motivation
Data Silos

https://global.hitachi-solutions.com/blog/break-down-data-silos

Motivation
Data Warehouse



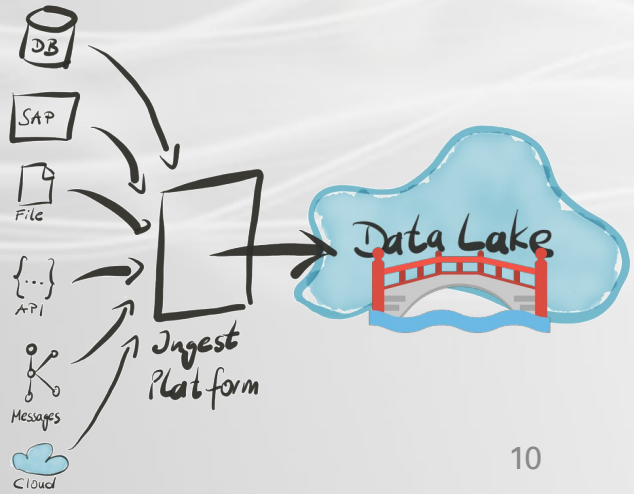
https://en.wikipedia.org/wiki/Data_integration#/media/File:Datawarehouse.png



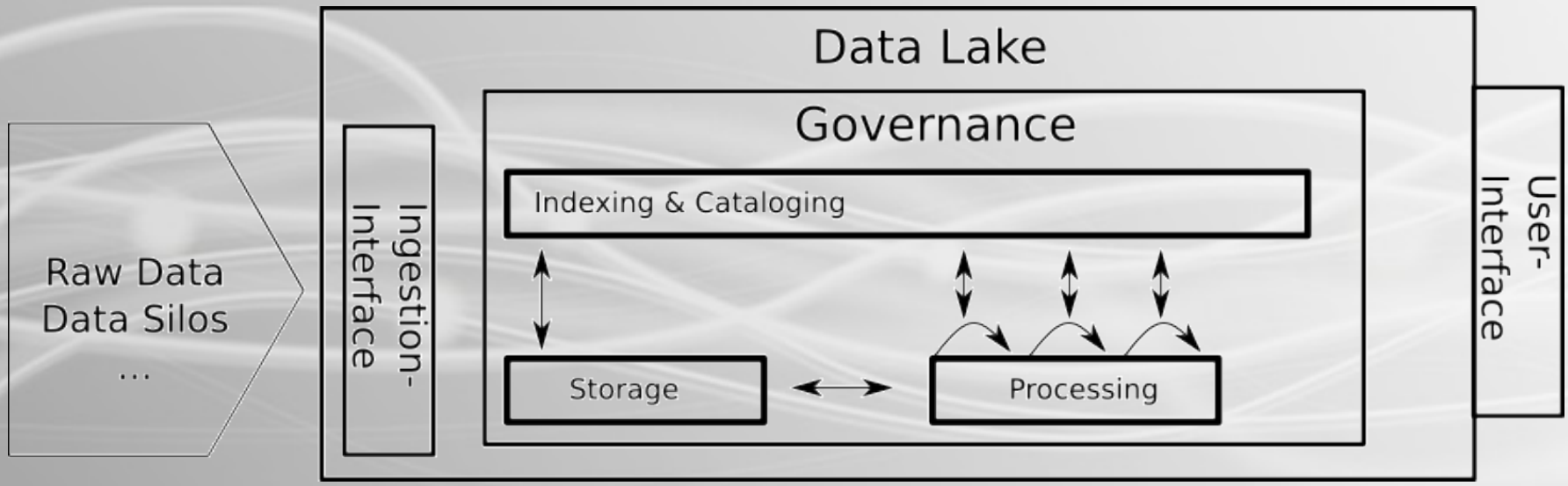
https://www.javatpoint.com/data-warehouse-what-is-star-schema

Introduction: Data Lake

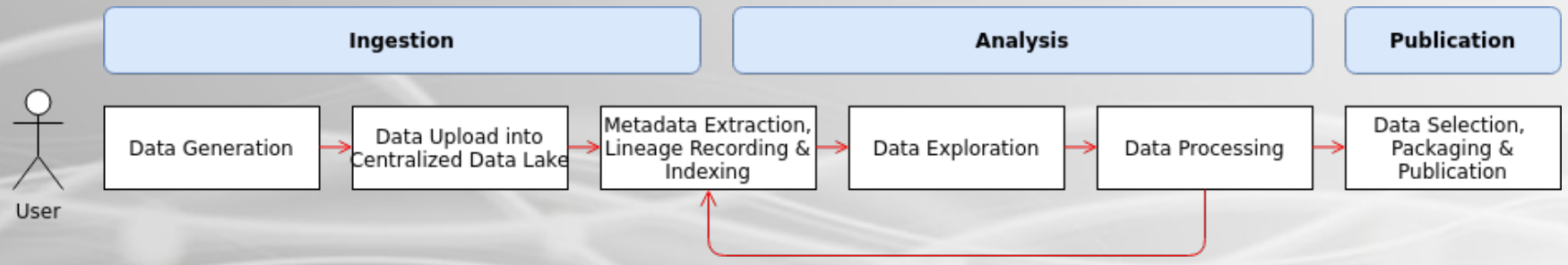
- With cheap storage costs, people promote the concept of data lake
  - Combines data from many sources and of any type
  - Allows for conducting future analysis and not miss any opportunity
- Attributes of the data lake
  - **Dump** everything: all time all data: raw sources and processed data
    - Decide during analysis which data is important, e.g., no “schema“ until read
  - **Dive** in anywhere: enable users across multiple business units to
    - Refine, explore and enrich data on their terms
  - **Fishing** for data, i.e., flexible access
    - Shared infrastructure supports various patterns
    - Usage: Batch, interactive, online, search



Concept of a Data Lake



General User Workflow on a Data Lake



Data Lake Functionalities

- Data repository for all raw, processed and associated data
- Start remote compute jobs, i.e. perform data analytic tasks
  - Automatic provenance auditing
  - Automatic artifact management
- Management of project/experiment/task metadata
- Search and inspection of data
- Sharing of data and developed methods
  - Portable packages of performed workflows
- Enforced governance
- Guided workflows
  - Python-Notebooks
  - Web-UI’s

Reference Architecture: Zone Architecture I

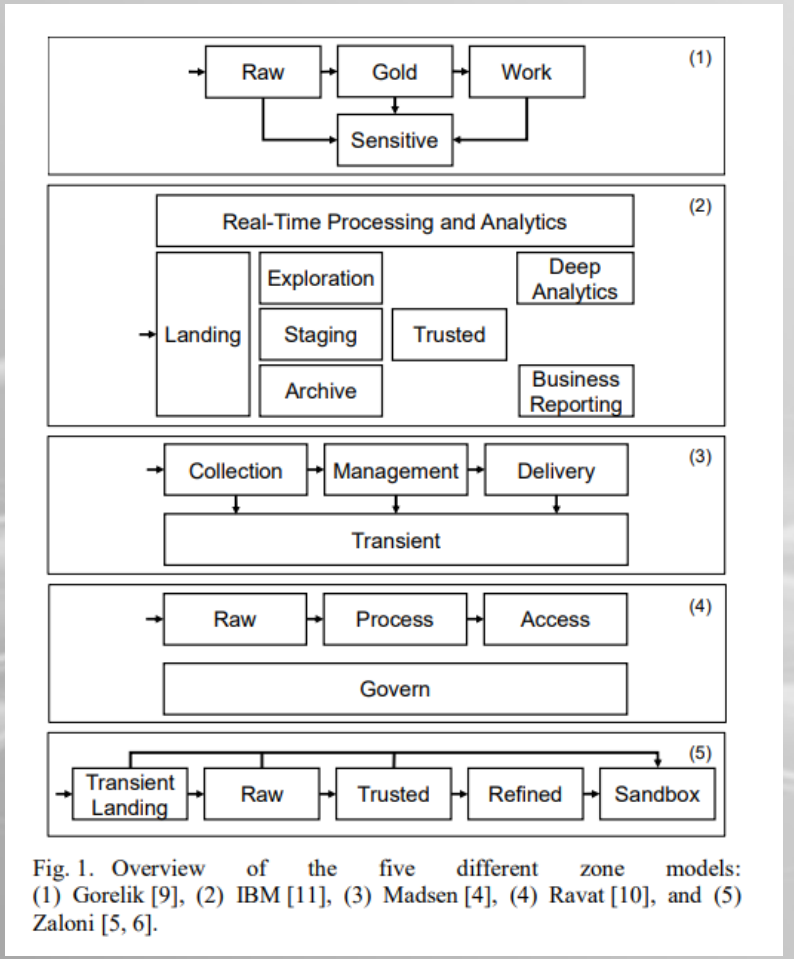


Fig. 1. Overview of the five different zone models: (1) Gorelik [9], (2) IBM [11], (3) Madsen [4], (4) Ravat [10], and (5) Zaloni [5, 6].

- Zone Architecture most common, but elusive
- Differences:
  - User Groups
  - Number and focus of Zones
- Fundamental idea:
  - Different Zones contain data in different degrees of processing

Giebler et al. “A Zone Reference Model for Enterprise-Grade Data Lake Management” (2020) IEEE Xplore

Reference Architecture: Zone Architecture II

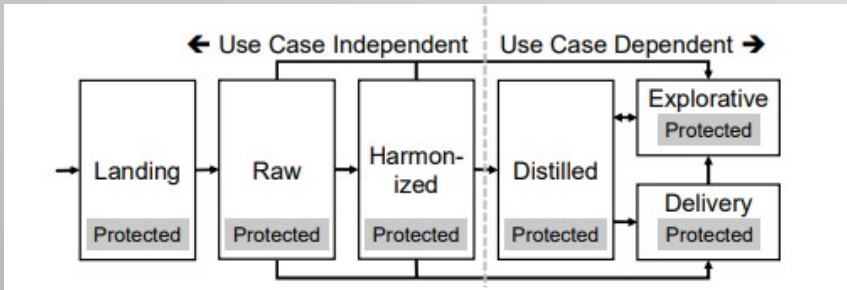


Fig. 3. The developed zone reference model comprises six zones. From the left to the right, data are processed more and more for specific usage.

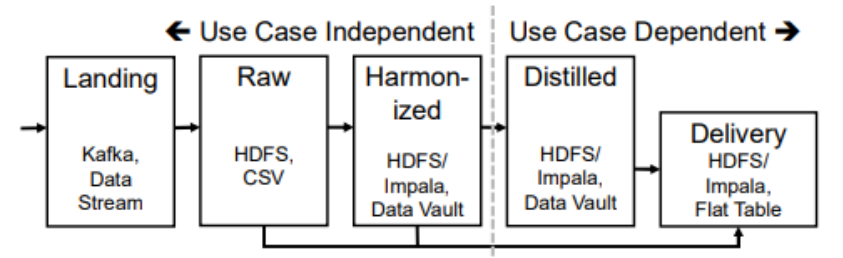
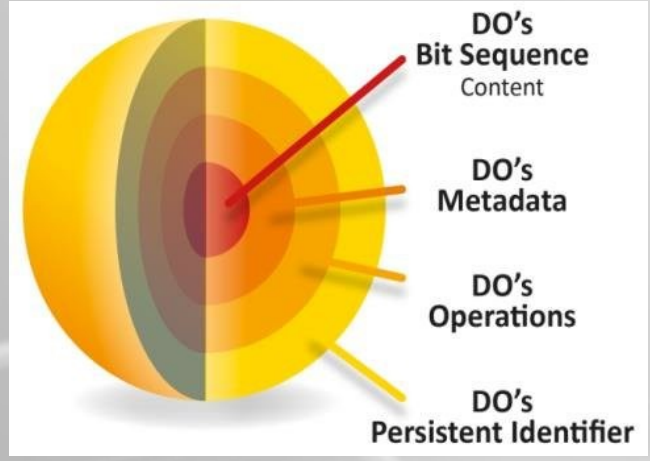


Fig. 4. The prototypical implementation of the zone reference model uses five of the six zones. Data are stored in different systems and schemata to realize the defined characteristics of the zones.

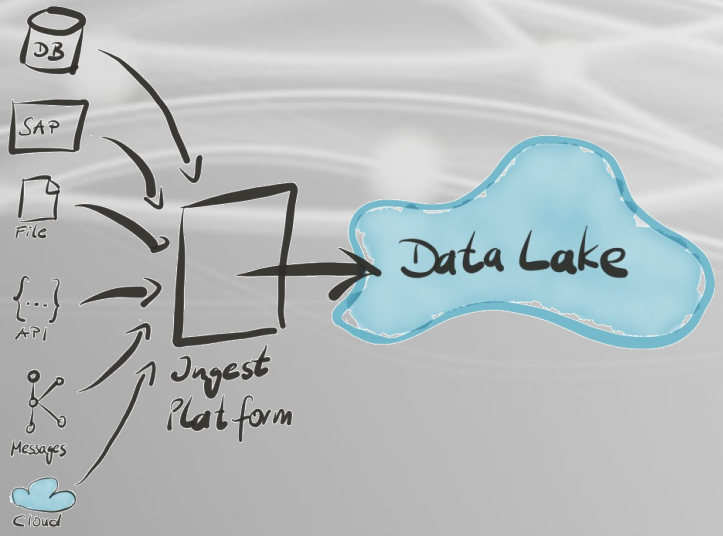
- Each zone has a different backend, i.e. storage, metadata and processing
- Not consistency safe
- Complete pipeline not possible in self service for users
- No overarching data lineage
- And no reproducibility
- Hard to scale access rights
- No global view

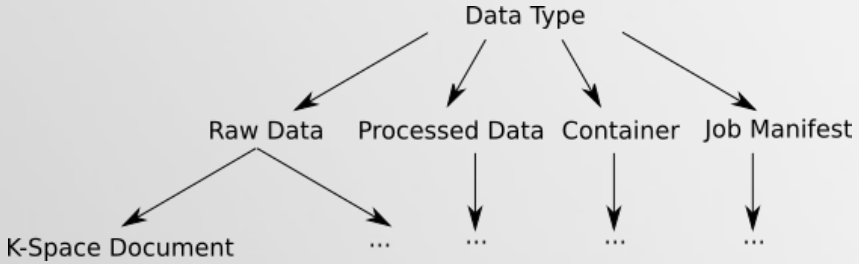
Giebler et al. “A Zone Reference Model for Enterprise-Grade Data Lake Management” (2020) IEEE Xplore

Data Lake Based on FAIR Digital Objects



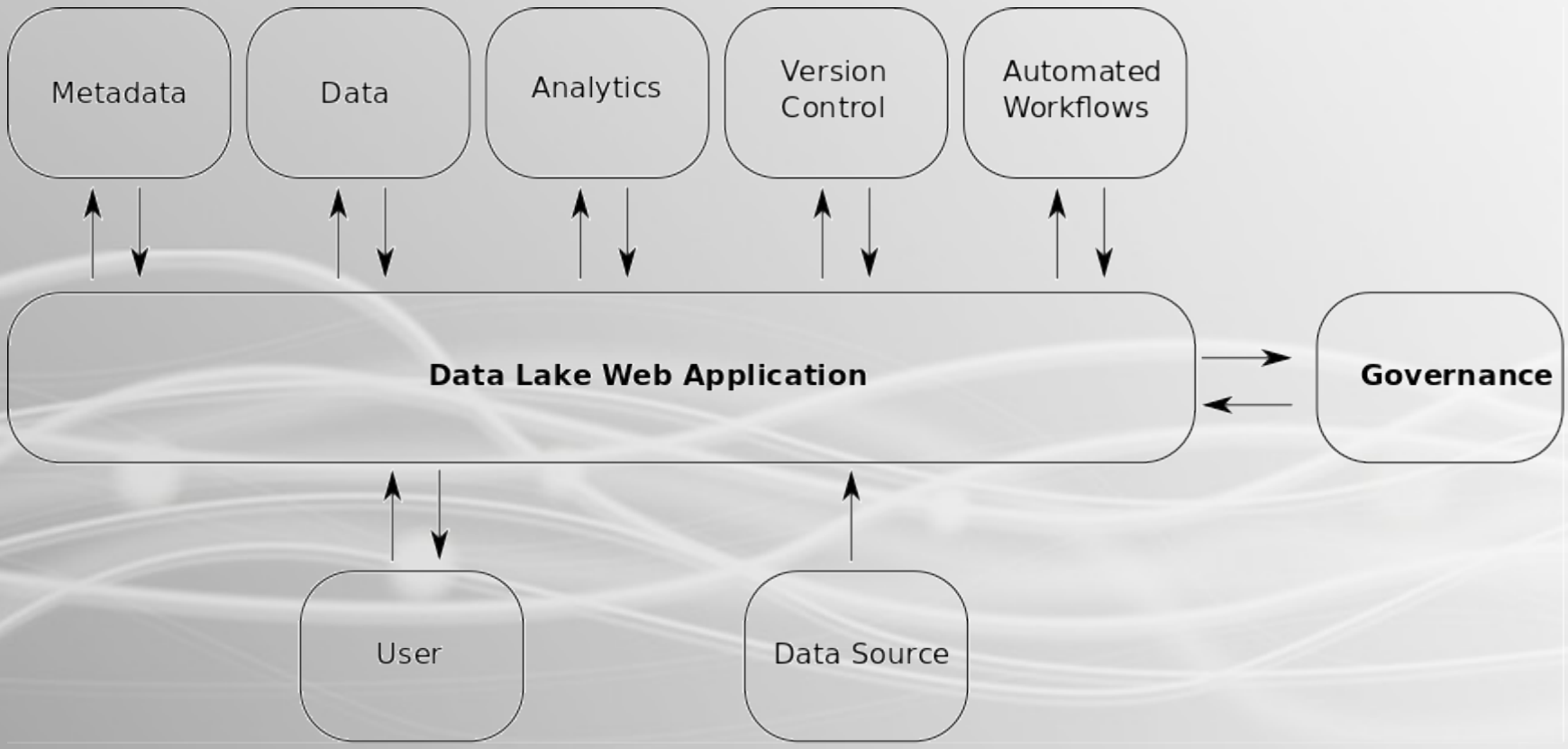
Koenraad De Smedt et al. "FAIR Digital Objects for Science: From Data Pieces to Actionable Units"(2020) Publications



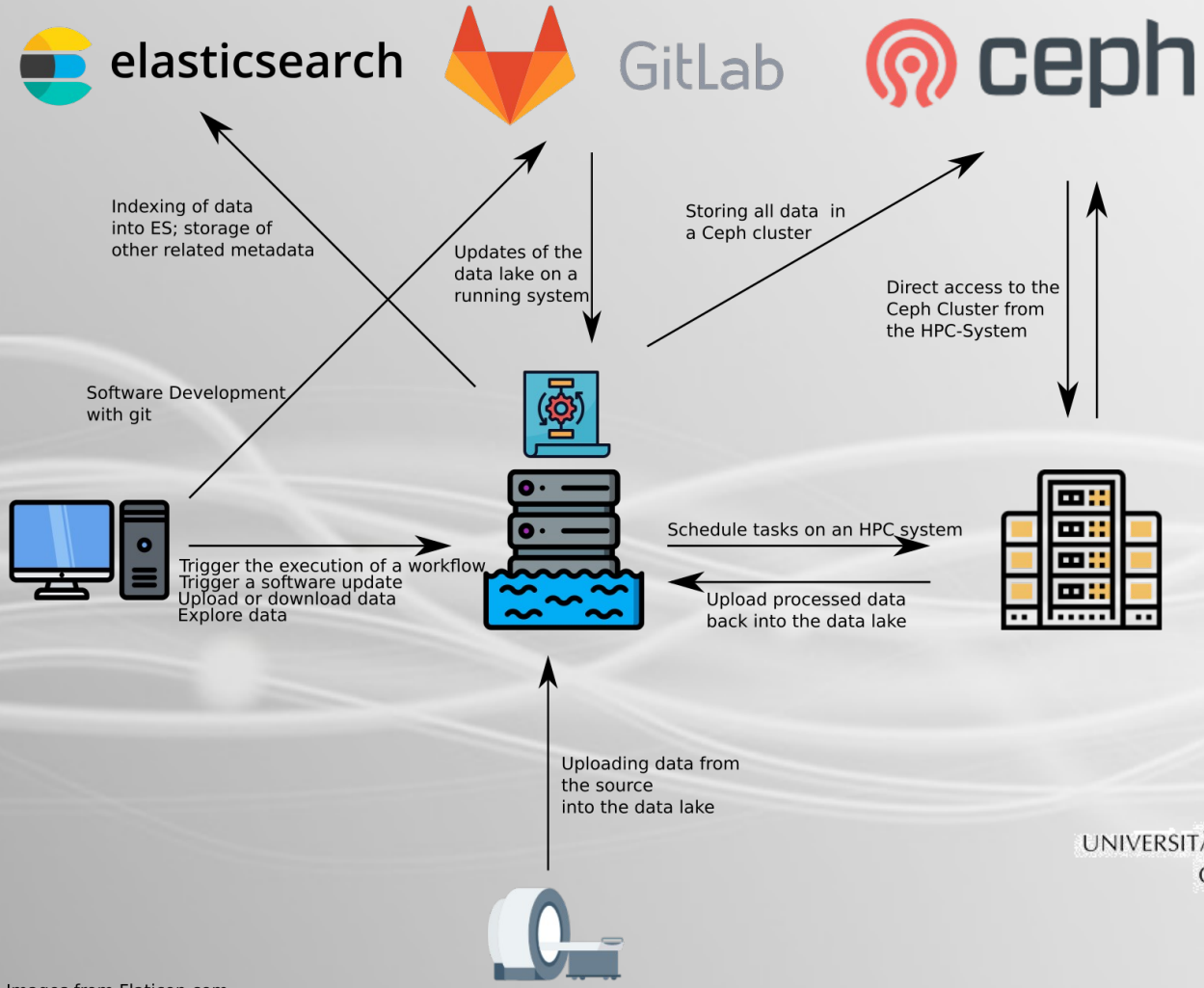


- Unified and flat space
- Data refinement based on the type
  - Raw Date or Processed Date
- Data type needs to be registered
  - Define metadata attributes
- Governance based on types and their typed attributes
- User does not interact with backend services directly
- Instead user call functions of these DO's
  - Guarantees a global consistent state

Schematic Architecture

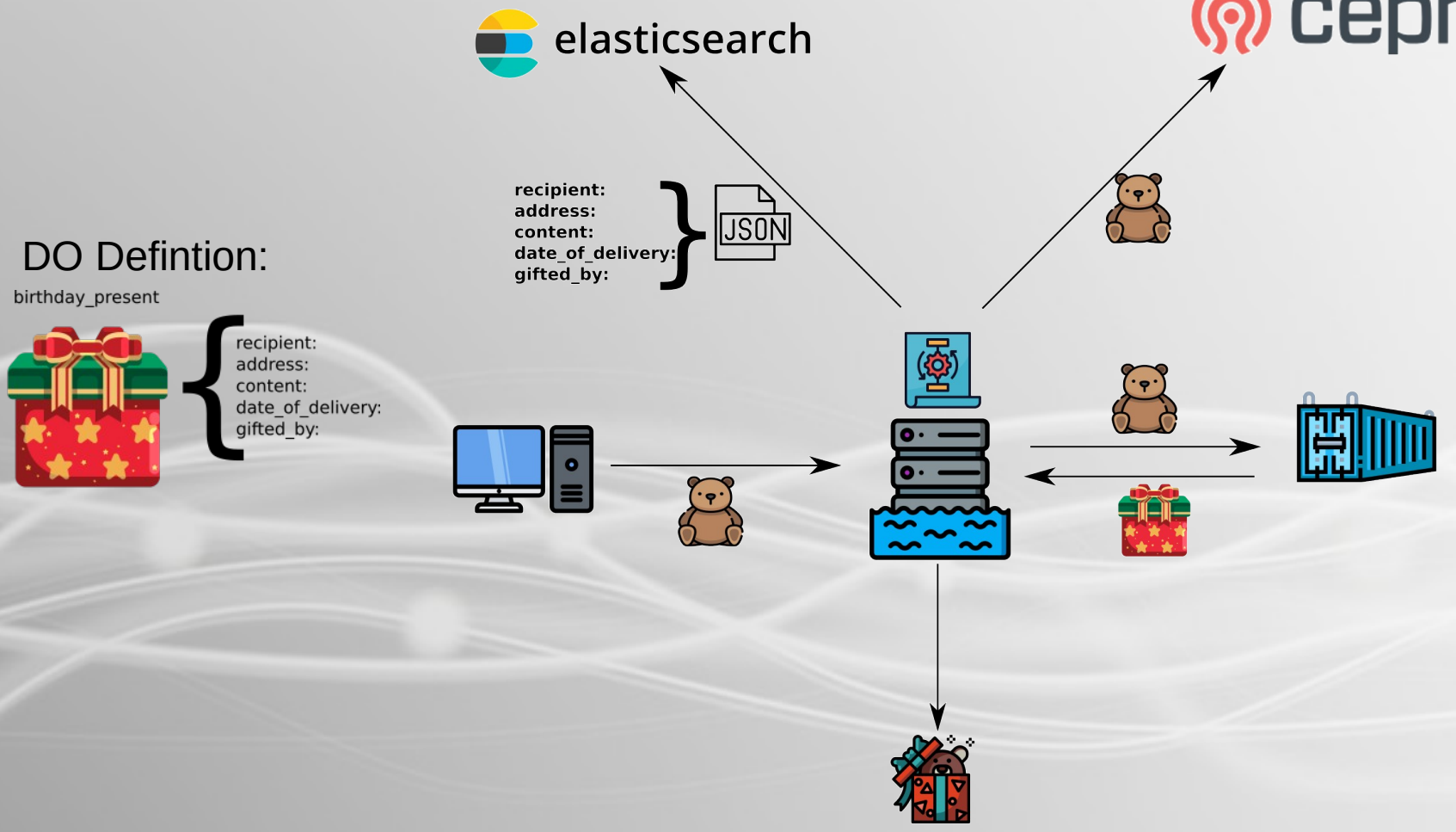


Overview



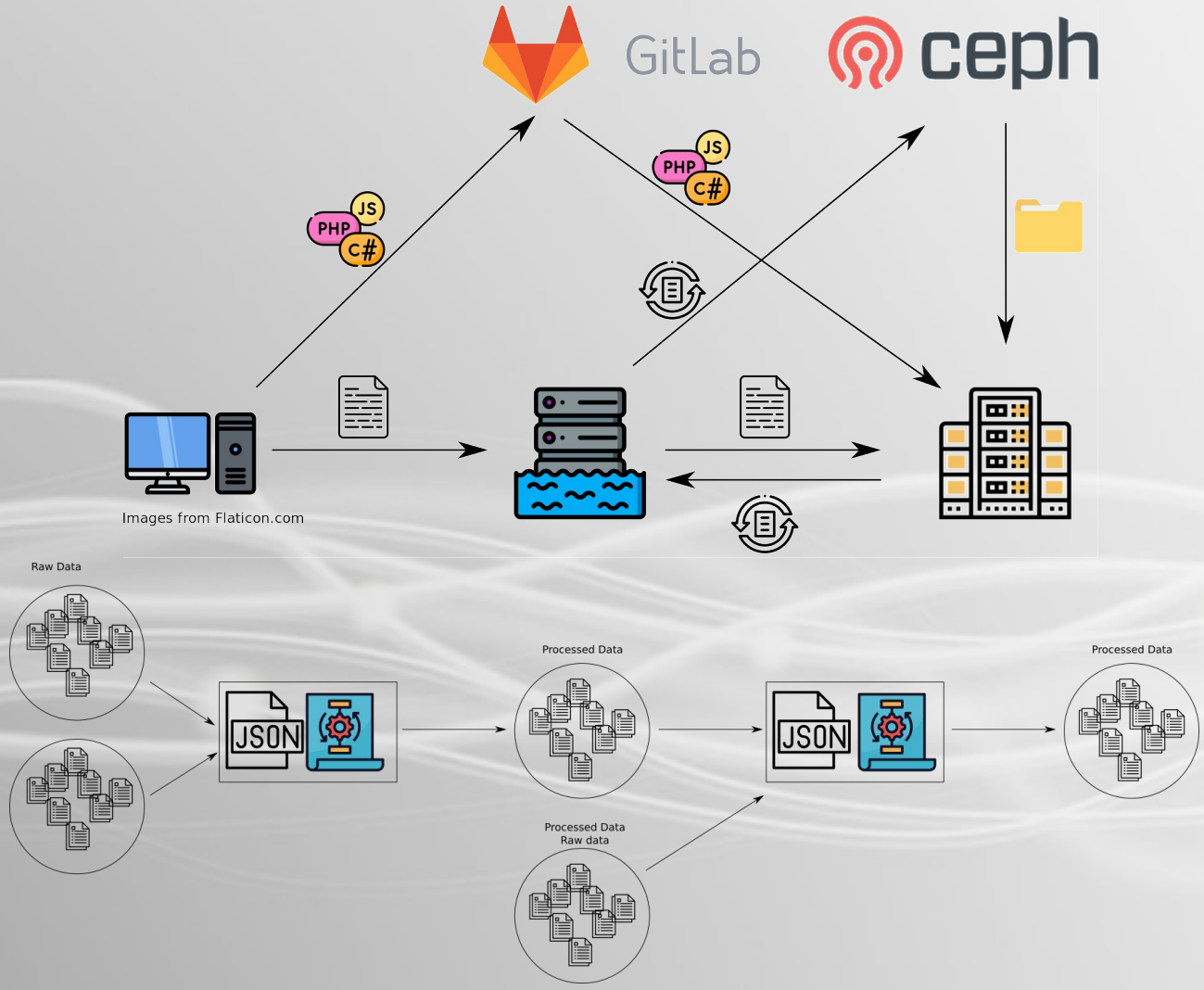
Images from Flaticon.com

Ingestion



Images from Flaticon.com

Processing


GitLab
ceph
JS
PHP
C#
Images from Flaticon.com
Raw Data
JSON
Processed Data
Processed Data
Raw data
JSON
Processed Data

Job Manifest

```
In [15]: import data_lake

         conn = data_lake.Connection(username='',password='', url='data-lake.gwdg.de')
```

```
In [8]: job = data_lake.Job()
        job.comment = 'Beispiel Job'
        job.container_name = 'bart-DeepDeepLearning'
        job.compute.append('cd /program/hpc-statustagung-rep/')
        job.compute.append('python3 example_python_script.py')
```

```
In [9]: job.git = [
            {
                "uri" : "git@gitlab.gwdg.de:hnolte1/hpc-statustagung-rep.git",
                "type" : "build",
                "bash" : "mkdir /program/output/"
            }
        ]
```

```
In [10]: job.job_name = "Output-Test"
         job.data_category = "kspace"
         job.env_var = {
             "OUTPUT_FOLDER" : "/program/output/",
             "NUM_REPETITIONS" : "2"
         }
```

```
In [11]: job.output_dir='/program/output'
```

```
In [ ]: res = job.submit(conn)
```

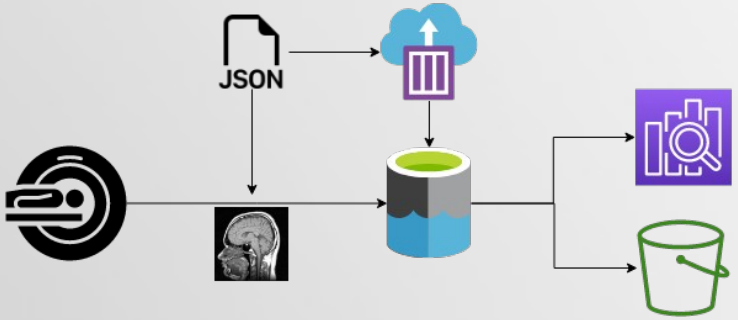
```
In [ ]: res = data_lake.show_jobs(conn, 'DataOutput','output_file_1.txt',format='GitRepositories,GitCommits,ContainerName,DataOutput,Comment,NUM_REPETITIONS,CreateDate')
```

```
In [17]: print(res.content.decode('utf-8'))
```

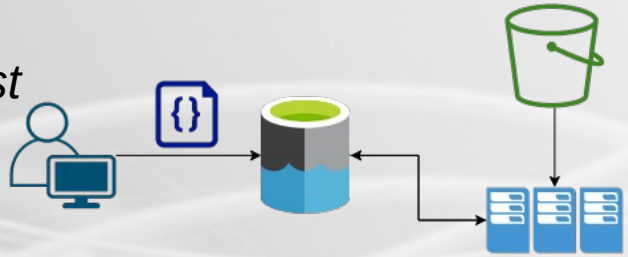
GitRepositories	GitCommits	ContainerName	DataOutput	Comment	NUM_REPETITIONS	CreateDate
hpc-statustagung-rep,	a1077ca42de7e8c4bc2f6d9265ddd07627588115,	bart-DeepDeepLearning.sif	output_file_1.txt output_file_0.txt	Beispiel Job	2	2021-04-16T15:09:24.563806
hpc-statustagung-rep,	a1077ca42de7e8c4bc2f6d9265ddd07627588115,	bart-ddl.sif	output_file_2.txt output_file_1.txt output_file_0.txt	Beispiel Job	3	2021-04-06T20:36:43.716102
hpc-statustagung-rep,	a1077ca42de7e8c4bc2f6d9265ddd07627588115,	bart-ddl.sif	output_file_1.txt output_file_0.txt	Beispiel Job 3	2	2021-04-07T13:11:48.496271

Workflow Example I
Data Science

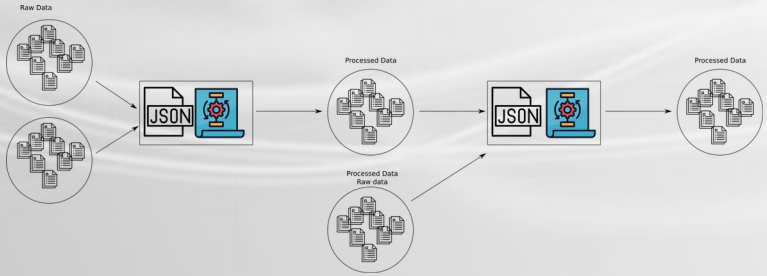
1) **Ingest Data**: Upload file(s), extract metadata online, or send it along



2) **Trigger Processing**: Send a *Job Manifest* to the data lake, to trigger reproducible processing on HPC-Resources

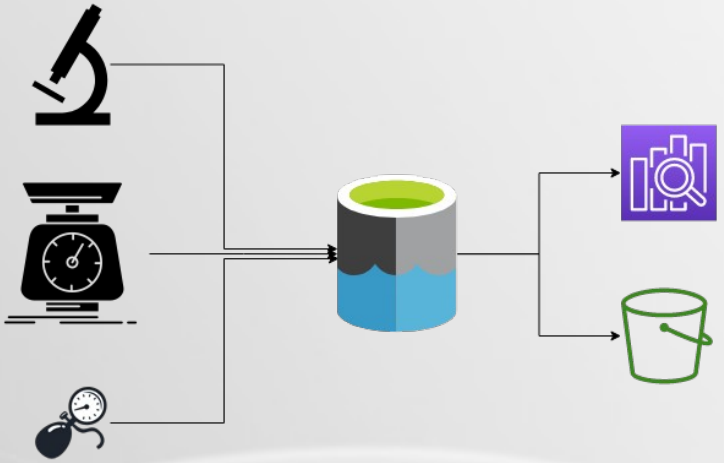


3) Inspect & Share Provenance Graph



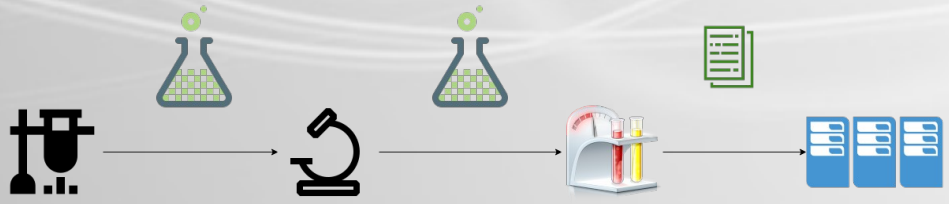
Workflow Example II
MPI-CEC

1) **Dataintegration:** Integrate data from different sources into a unified & homogeneous data management system



2) User Interaction: Users can manage & inspect data/experiments/measurements etc. via web pages

3) Lineage Recording: Inspect data lineage of physical and digital analysis



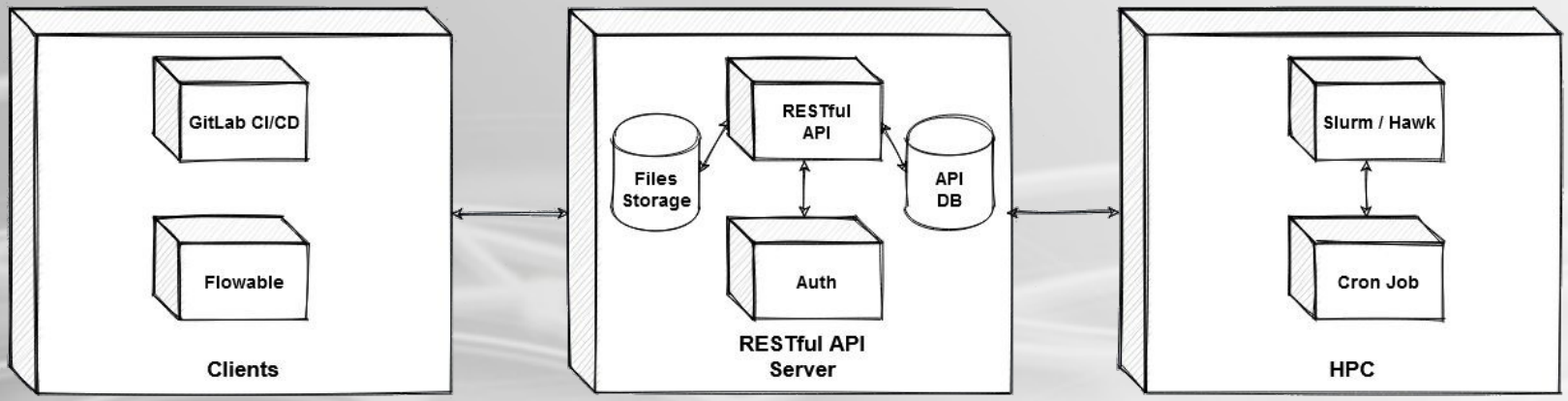
Conclusion

- Typical requirements from researchers within such a distributed institute are:
  - Central place to store all data
    - Particularly unstructured research data
  - Globally enforced governance
  - High-Level user interface
  - Easy way to share data and developed methods
  - Search capabilities over all collected data sets
  - Reproducible processing
    - Built-in provenance auditing

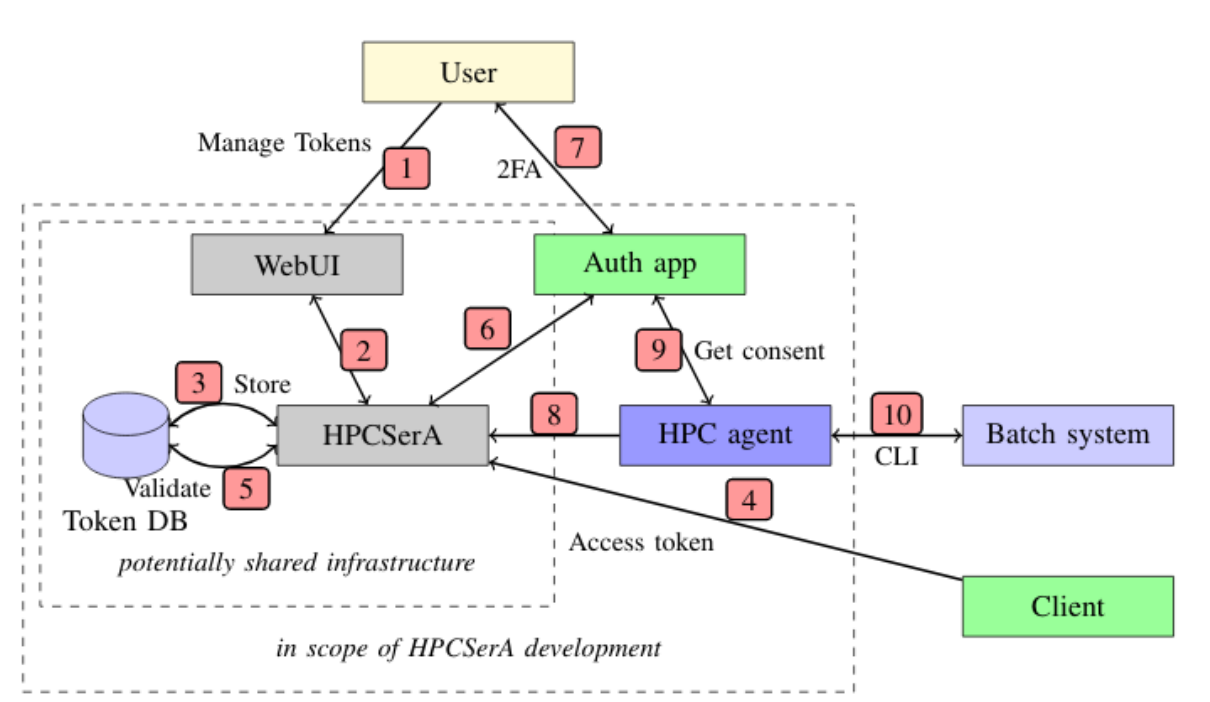
- Our Data Lake **can fulfill** these requirements with the following set of features:
  - WebApp offers this single central contact point
  - Data-Governances can be enforced on the data lake layer
  - Web-Interfaces or libraries
  - Access to data and methods can be managed by users
  - The data catalog offers search capabilities over all entities
  - Built-in reproducibility and provenance auditing using Job Manifests and HPCSerA

Our data lake abstracts the complicated storage architecture and combines a research data management system with an HPC cluster

Remote Job Execution HPCSerA



HPCSerA Security Aspects



Role Number	Role	Description
1	GET_JobStatus	Client can retrieve information about a submitted job
2	UPDATE_JobStatus	Used by client/agent to update the job status
3	GET_Job	Endpoint used by the agent to retrieve job information
4	POST_Code	Client to ingest new code to the HPC sytsem
5	GET_Code	Agent pulls new code. Might be necessary to run new job
6	POST_Job	Client triggers parameterized job
7	UPDATE_Job	Client updates already triggered job
8	DELETE_Job	Client deletes already triggered job