Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology MeteoSwiss

Semantic data access to gridded weather data based on zarr

7th ENES HPC Workshop

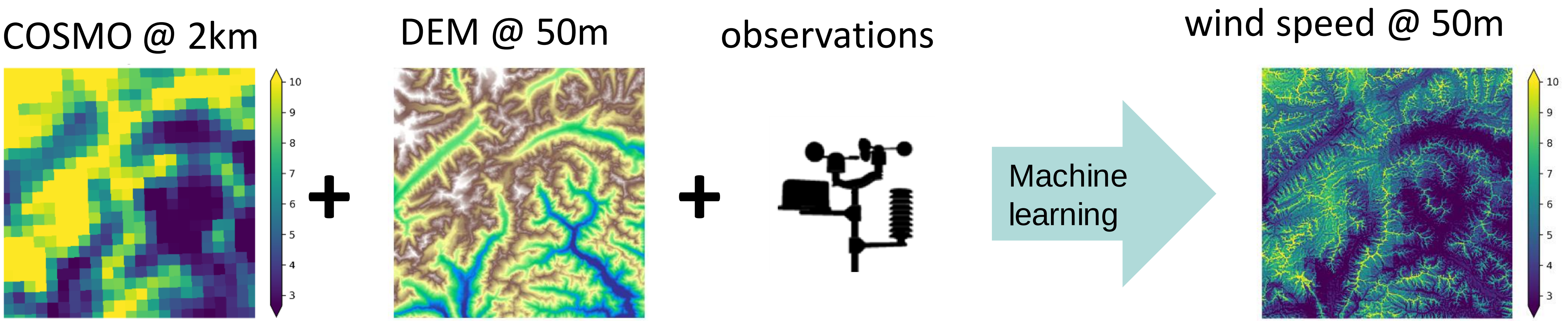
May 11 2022

Gabriela Aznar Siguan, Néstor Tarín, Marcus Schulte, Manuel Moser, Philipp Falke, Mathieu Bernard, Christoph Spirig, Mathieu Schaer, Mark Liniger

MOTIVATION

Postprocessing and downscaling as a typical data science application

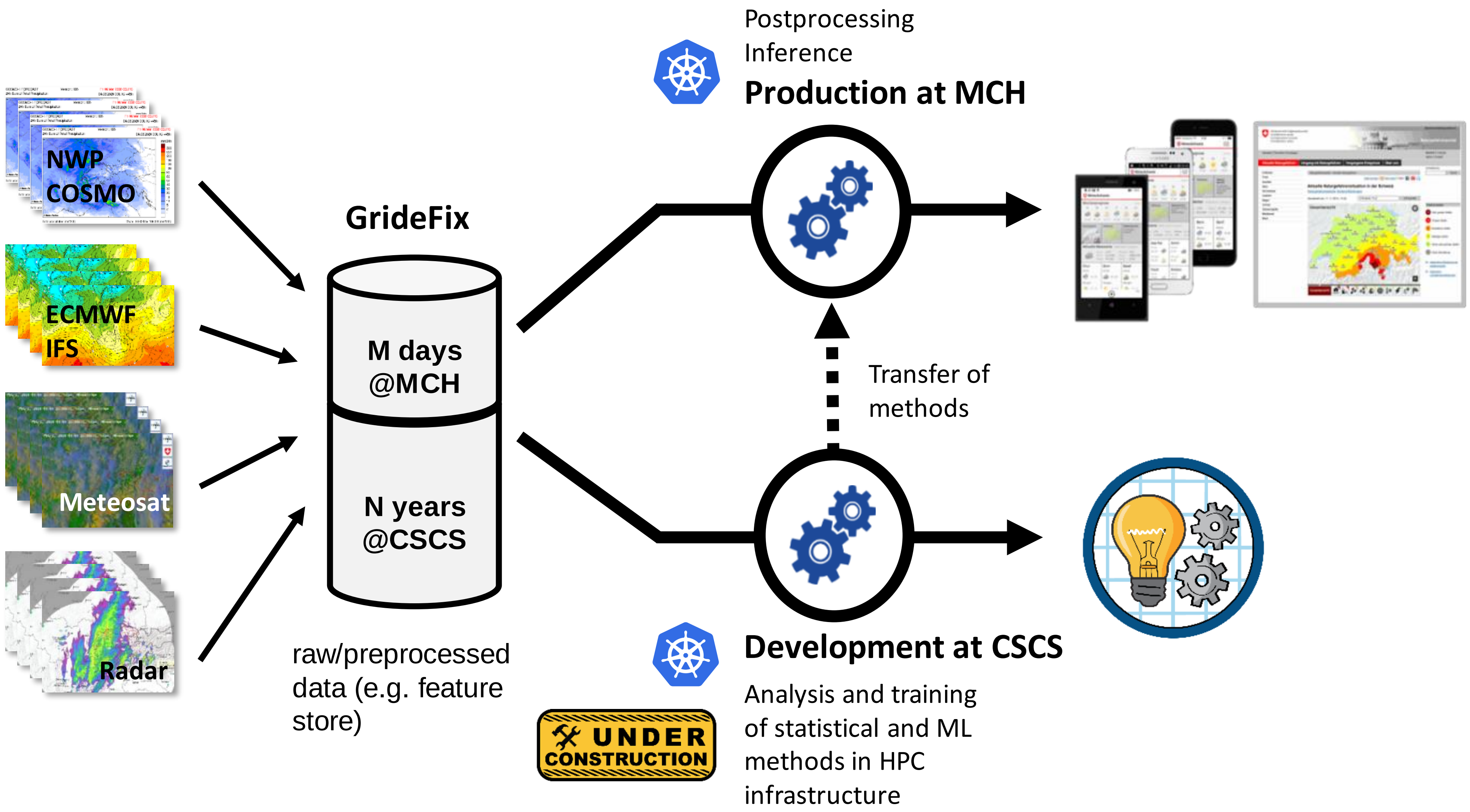
refining numerical weather predictions and climate simulations
using large archives of past data and statistics/machine learning



Daniele Nerini et al. EMS 2021

manipulating **~ 100 TB + 35 TB/yr** heterogeneous data

Unified access (API) to feed data pipelines



GrideFix database in a nutshell

- GrideFix is a **cloud-native** database: it can be deployed close to the data and accessed from everywhere.

Amazon S3

GrideFix database in a nutshell

- GrideFix is a **cloud-native** database: it can be deployed close to the data and accessed from everywhere.
- It handles **multi-dimensional arrays** using compressed binary chunks following the **Zarr** specification.

Amazon S3

Zarr

GrideFix database in a nutshell

- GrideFix is a **cloud-native** database: it can be deployed close to the data and accessed from everywhere.
- It handles **multi-dimensional arrays** using compressed binary chunks following the **Zarr** specification.
- GrideFix offers a **unified** and **semantic data access**. Users can query data by tags and standard variable names, independently of the data source and original file structure.

Amazon S3

Zarr

GrideFix database in a nutshell

- GrideFix is a **cloud-native** database: it can be deployed close to the data and accessed from everywhere.
- It handles **multi-dimensional arrays** using compressed binary chunks following the **Zarr** specification.
- GrideFix offers a **unified** and **semantic data access**. Users can query data by tags and standard variable names, independently of the data source and original file structure.
- Timestamped data can be imported and deleted efficiently, it is optimized to implement a **rolling archive**.

GrideFix database in a nutshell

- GrideFix is a **cloud-native** database: it can be deployed close to the data and accessed from everywhere.
- It handles **multi-dimensional arrays** using compressed binary chunks following the **Zarr** specification.
- GrideFix offers a **unified** and **semantic data access.** Users can query data by tags and standard variable names, independently of the data source and original file structure.
- Timestamped data can be imported and deleted efficiently, it is optimized to implement a **rolling archive**.
- A **Python API** provides the requested data as **dask** arrays, allowing to slice and manipulate the data along any arbitrary dimension with **parallel** computations.

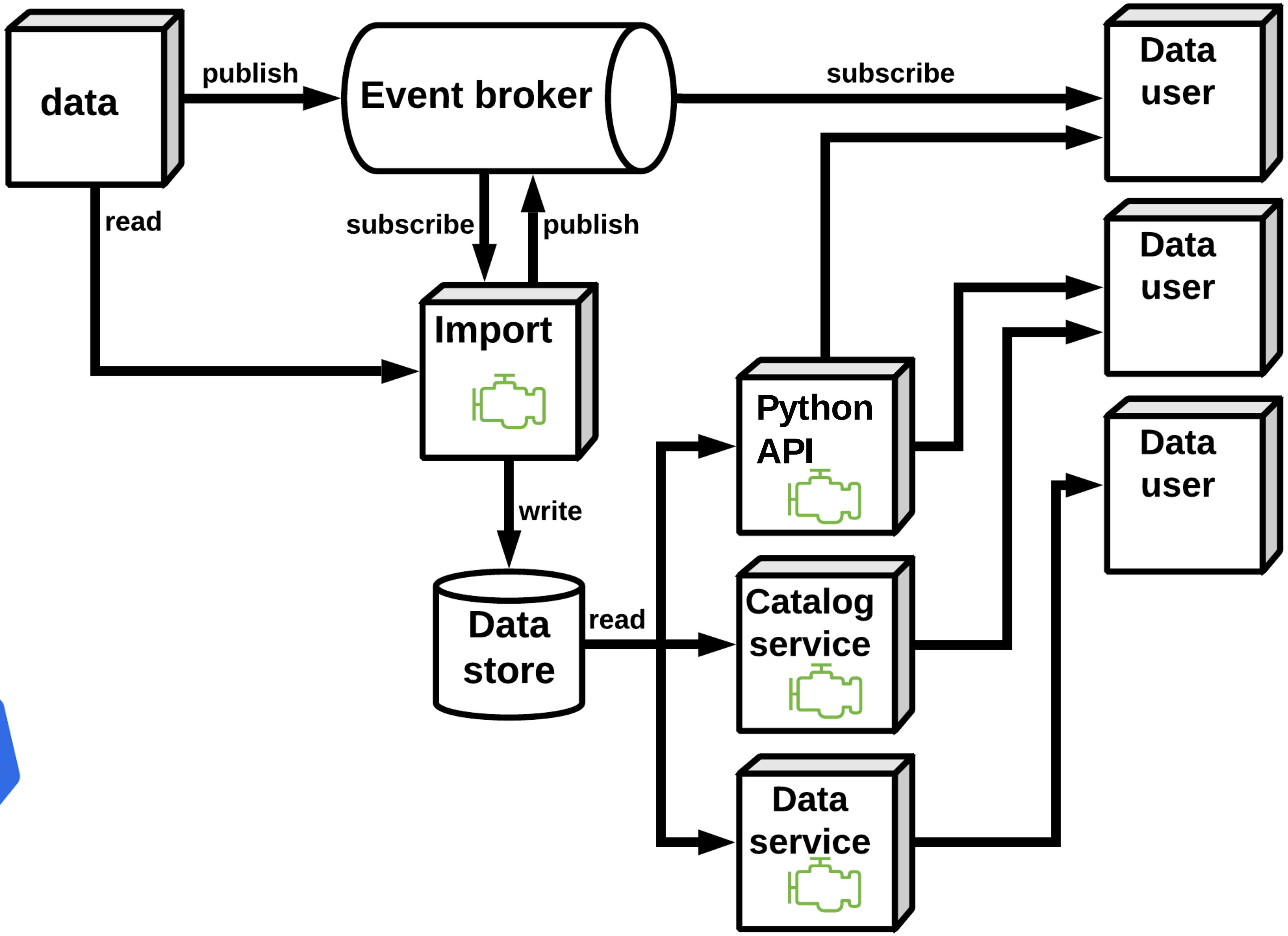
Amazon S3 Zarr xarray DASK

GrideFix database in a nutshell

- GrideFix is a **cloud-native** database: it can be deployed close to the data and accessed from everywhere.
- It handles **multi-dimensional arrays** using compressed binary chunks following the **Zarr** specification.
- GrideFix offers a **unified** and **semantic data access.** Users can query data by tags and standard variable names, independently of the data source and original file structure.
- Timestamped data can be imported and deleted efficiently, it is optimized to implement a **rolling archive**.
- A **Python API** provides the requested data as **dask** arrays, allowing to slice and manipulate the data along any arbitrary dimension with **parallel** computations.
- **Web services** exposing the API provide cloud-native **import**, **export** (via OPeNDAP) and **catalogue** services to facilitate language independent event-driven architectures.

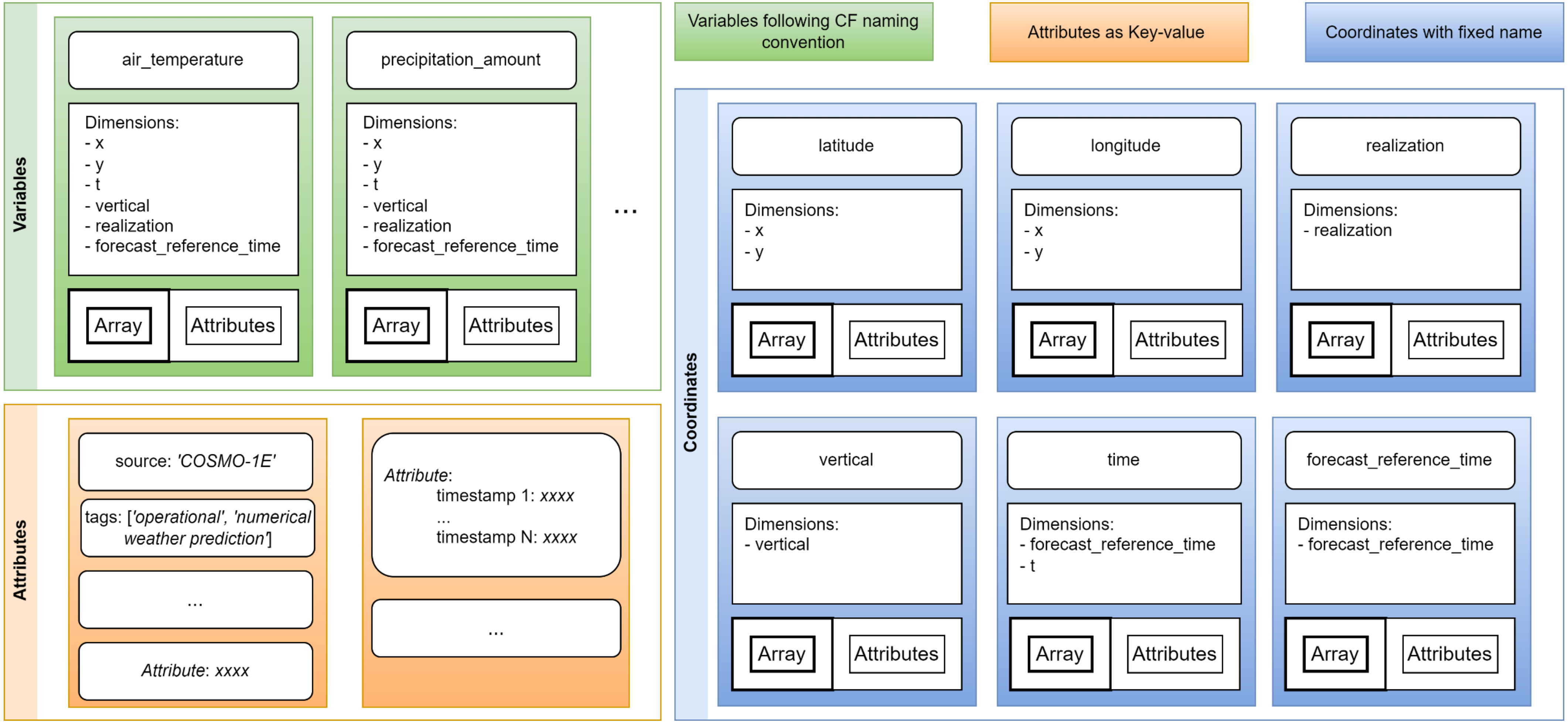
Amazon S3 Zarr xarray DASK FastAPI OPENAPI INITIATIVE

High level architecture



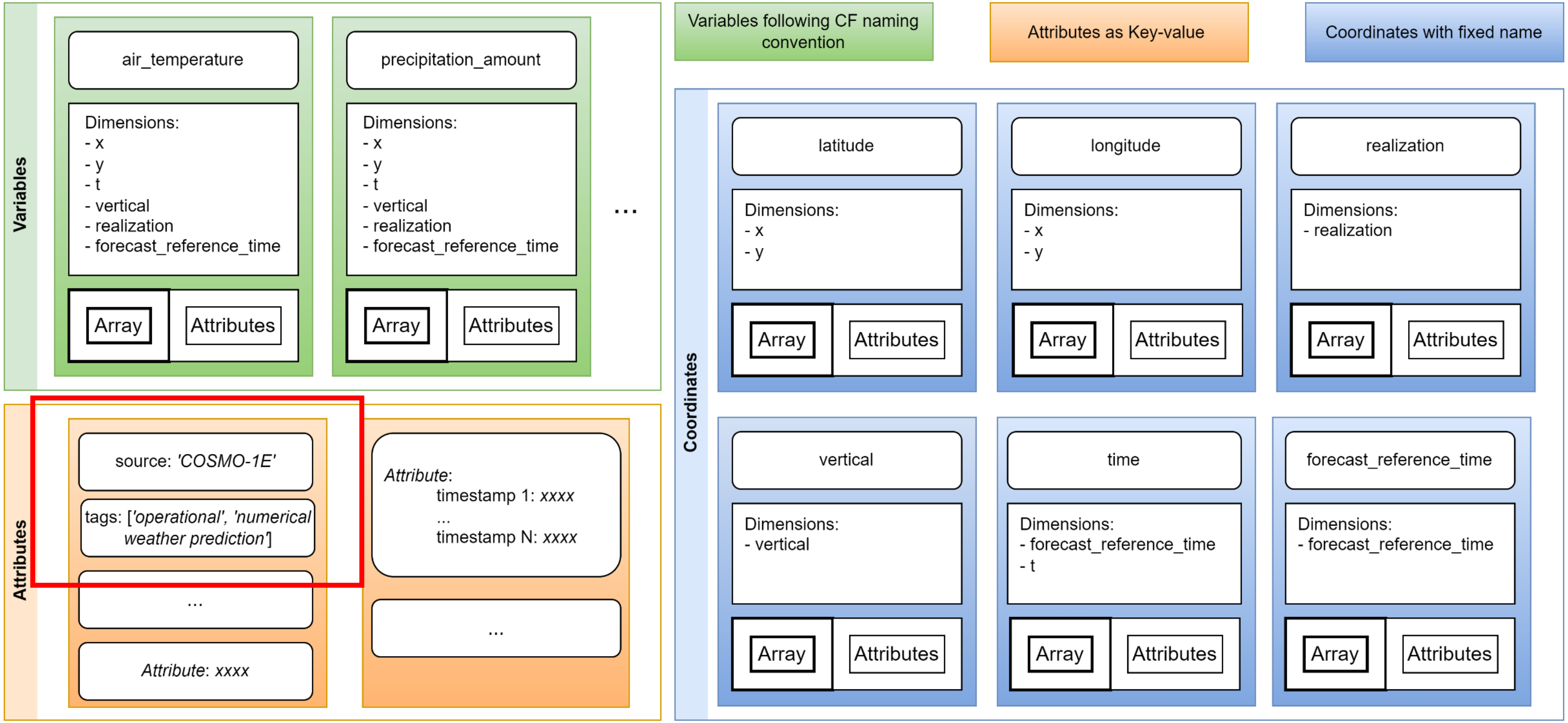
HOW DOES GRIDEFIX WORK?

Data model



Variables
air_temperature
Dimensions:
- x
- y
- t
- vertical
- realization
- forecast_reference_time
Array
Attributes
precipitation_amount
Dimensions:
- x
- y
- t
- vertical
- realization
- forecast_reference_time
Array
Attributes
…
Attributes
source: 'COSMO-1E'
tags: ['operational', 'numerical weather prediction']
...
Attribute: xxxx
Attribute:
timestamp 1: xxxx
...
timestamp N: xxxx
...
Variables following CF naming convention
Attributes as Key-value
Coordinates with fixed name
Coordinates
latitude
Dimensions:
- x
- y
Array
Attributes
longitude
Dimensions:
- x
- y
Array
Attributes
realization
Dimensions:
- realization
Array
Attributes
vertical
Dimensions:
- vertical
Array
Attributes
time
Dimensions:
- forecast_reference_time
- t
Array
Attributes
forecast_reference_time
Dimensions:
- forecast_reference_time
Array
Attributes

Labeled data to ease search



Data format defined by GrideFix arrays with Zarr

```
COSMO-1
├── 30a23917-11e7-4950-82b2-e8690184ac55
|   ├── .zattrs
|   ├── coordinates
|   |   ├── latitude
|   |   ├── longitude
|   |   ├── time
|   |   └── vertical
|   ├── 2020-01-01T00:00:00
|   |   ├── air_temperature
|   |   ├── eastward_wind
|   |   ├── mass_fraction_of_cloud_liquid_water_in_air
|   |   └── northward_wind
|   └── 2020-01-01T06:00:00
|       ├── air_temperature
|       ├── eastward_wind
├── 9ed433bb-ebfc-4f00-8f31-f0441d8c703f
|   ├── .zattrs
|   ├── coordinates
|   |   ├── latitude
|   |   ├── longitude
|   |   ├── time
|   |   └── vertical
|   └── air_temperature
|   └── eastward_wind
|   └── mass_fraction_of_cloud_liquid_water_in_air
|   └── northward_wind
├── 482505bf-f2c3-43bf-867c-e34f6156bb21
...
```

- COSMO-1 ← Source group
- 30a23917-11e7-4950-82b2-e8690184ac55 ← Array group identified by uuid
- .zattrs ← Tags go here
- coordinates ← Array coordinates group
- 2020-01-01T00:00:00 ← Time axis group
- air_temperature ← Variable array
- 9ed433bb-ebfc-4f00-8f31-f0441d8c703f ← Array uuid
- .zattrs ← Tags go here
- air_temperature ← Static data

Data format defined by GrideFix arrays with Zarr

```
COSMO-1
├── 30a23917-11e7-4950-82b2-e8690184ac55
|   ├── .zattrs
|   ├── coordinates
|   |   ├── latitude
|   |   ├── longitude
|   |   ├── time
|   |   └── vertical
|   ├── 2020-01-01T00:00:00
|   |   ├── air_temperature
|   |   ├── eastward_wind
|   |   ├── mass_fraction_of_cloud_liquid_water_in_air
|   |   └── northward_wind
|   └── 2020-01-01T06:00:00
|       ├── air_temperature
|       ├── eastward_wind
├── 9ed433bb-ebfc-4f00-8f31-f0441d8c703f
|   ├── .zattrs
|   ├── coordinates
|   |   ├── latitude
|   |   ├── longitude
|   |   ├── time
|   |   └── vertical
|   └── air_temperature
|   └── eastward_wind
|   └── mass_fraction_of_cloud_liquid_water_in_air
|   └── northward_wind
├── 482505bf-f2c3-43bf-867c-e34f6156bb21
...
```

Source group
Array group identified by uuid
Tags go here
Array coordinates group

Time axis group
Variable array

Advantages:

- rolling archive efficient
- time-dependent attributes

Array uuid
Tags go here

Static data

Data not changing often.
Advantages:

- more chunking freedom

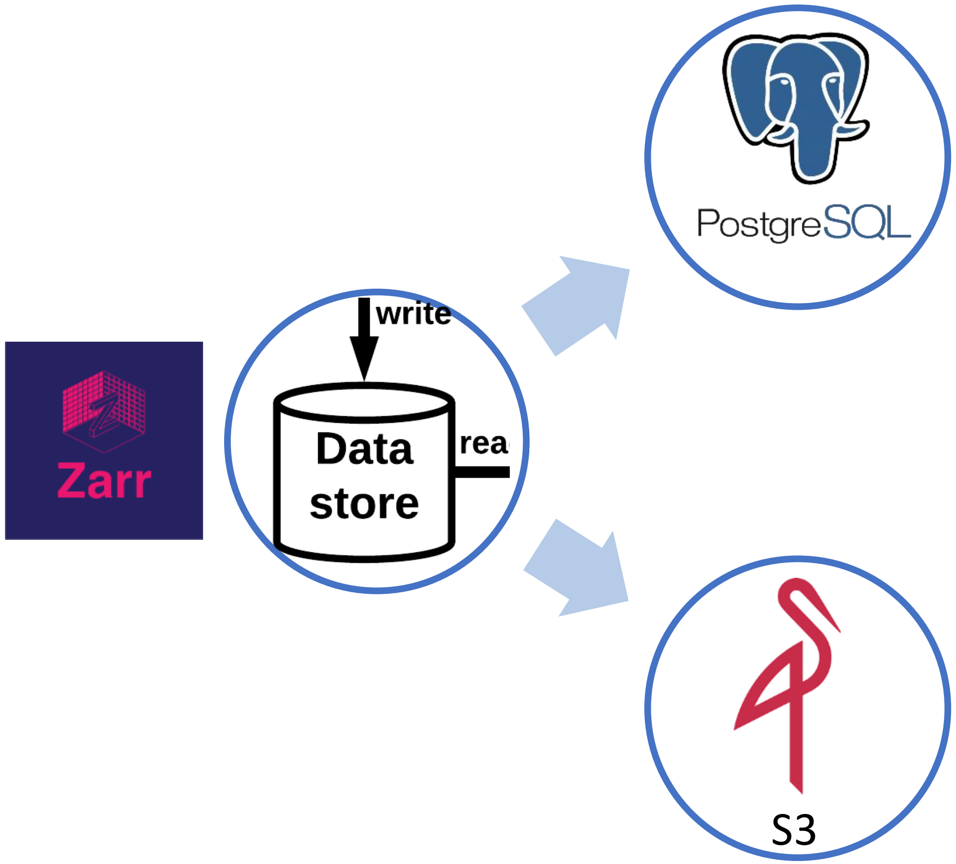
GrideFix arrays are combined into datasets

```
COSMO-1
├── 30a23917-11e7-4950-82b2-e8690184ac55
|   ├── .zattrs
|   ├── coordinates
|   |   ├── latitude
|   |   ├── longitude
|   |   ├── time
|   |   └── vertical
|   ├── 2020-01-01T00:00:00
|   |   ├── air_temperature
|   |   ├── eastward_wind
|   |   ├── mass_fraction_of_cloud_liquid_water_in_air
|   |   └── northward_wind
|   └── 2020-01-01T06:00:00
|       ├── air_temperature
|       ├── eastward_wind
├── 9ed433bb-ebfc-4f00-8f31-f0441d8c703f
|   ├── .zattrs
|   ├── coordinates
|   |   ├── latitude
|   |   ├── longitude
|   |   ├── time
|   |   └── vertical
|   └── air_temperature
|   └── eastward_wind
|   └── mass_fraction_of_cloud_liquid_water_in_air
|   └── northward_wind
├── 482505bf-f2c3-43bf-867c-e34f6156bb21
...
```

Source
Array uuid
Tags go here
Array coordinates
Time axis group
Variable array
Array uuid
Tags go here
Static data

Arrays with **same source and tags** are concatenated along the major time axis into a **dataset**.

Metadata store to improve performance



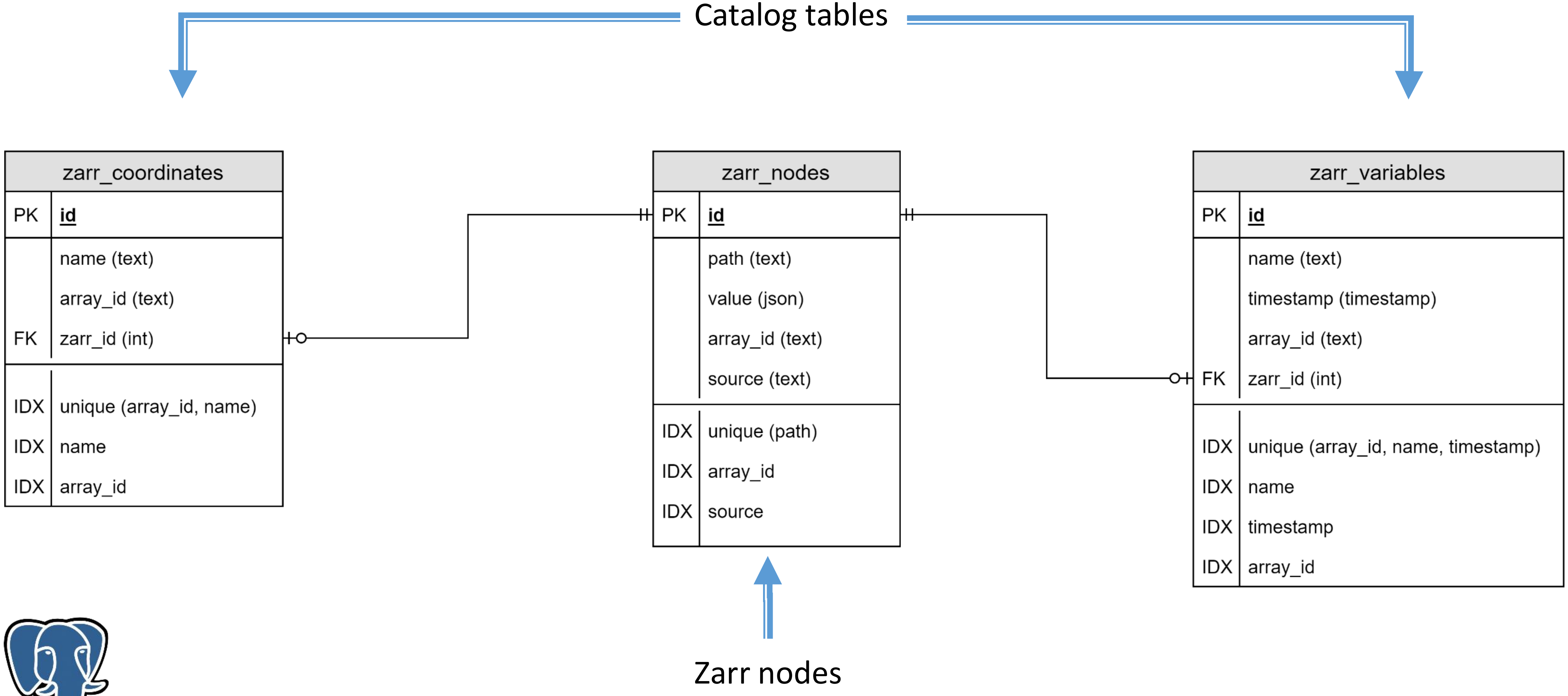
Metadata store:

- Fast catalog queries
- Fast identification of queried data to retrieve
- Data consistency using transactional operations
- No need to consolidate metadata

Chunk store:

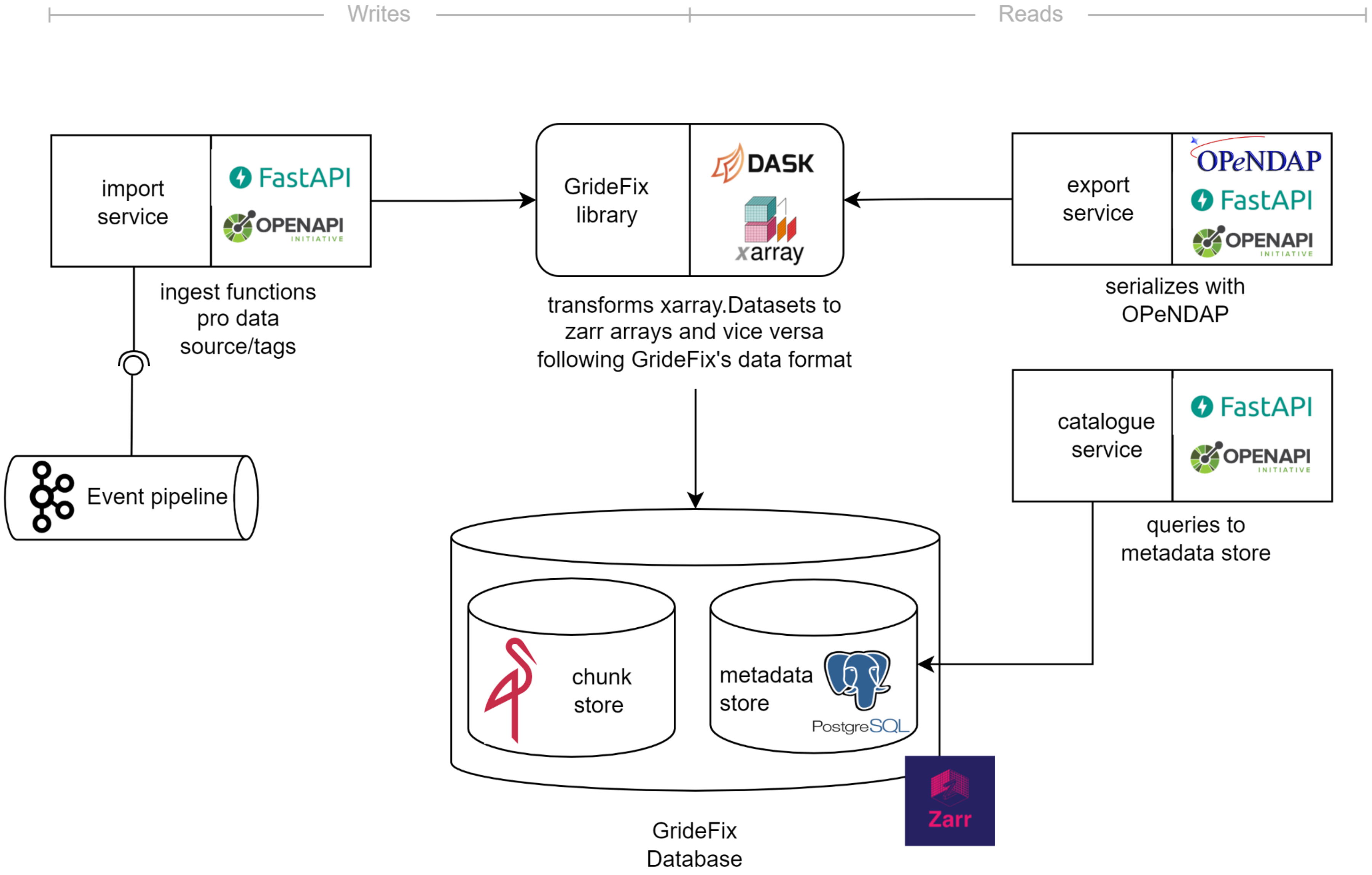
- Store available from everywhere

Zarr metadata store data model



PostgreSQL

APIs



Writes
Reads
import service
FastAPI
OPENAPI INITIATIVE
ingest functions pro data source/tags
Event pipeline
GrideFix library
DASK
xarray
transforms xarray.Datasets to zarr arrays and vice versa following GrideFix's data format
export service
OPeNDAP
FastAPI
OPENAPI INITIATIVE
serializes with OPeNDAP
catalogue service
FastAPI
OPENAPI INITIATIVE
queries to metadata store
chunk store
metadata store
PostgreSQL
Zarr
GrideFix Database

Example: start using the catalogue web service

language independent access

cloud-native → scalable

get fast information about available data

```
In [1]: import requests
        DISCOVER_URL = 'http://discover.meteoswiss.ch/gridefix/api'

In [3]: # 1. get sources which contain variables of interest for surface values
        variables = 'air_temperature,eastward_wind,northward_wind,wind_speed_of_gust'
        tags = 'surface'

        sources = requests.get(f'{DISCOVER_URL}/sources?tags={tags}')
        sources.json()

Out[3]: ['COSMO-2E', 'COSMO-1E', 'ECMWF_IFS']

In [4]: # 2. choose source and see details
        source = 'COSMO-2E'

        source_details = requests.get(f'{DISCOVER_URL}/sources/{source}')
        source_details.json()

Out[4]: [{'bounding_box': [5.304744720458984,
           45.49021530151367,
           10.898223876953125,
           48.099998474121094],
          'crs': 'EPSG:4326',
          'description': 'MeteoSwiss ensemble forecasting system',
          'level_type': 'depth',
          'name': 'COSMO-2E',
          'source_crs': 'EPSG:4326',
          'tags': ['numerical weather prediction',
           'operational',
           'forecast',
           'ensemble',
           'depth'],
          'timestamps': ['2022-03-09T12:00:00',
           '2022-03-09T18:00:00',
           '2022-03-10T00:00:00',
           '2022-03-10T06:00:00',
           '2022-03-10T12:00:00',

In [5]: # 3. get available timestamps for chosen source and tags
        tags = 'operational,numerical weather prediction,ensemble,surface,forecast'

        timestamps = requests.get(f'{DISCOVER_URL}/sources/{source}/timestamps?variable={variables}&tags={tags}')
        timestamps = timestamps.json()

        print(f'\n Timestamps available for variables {variables} in dataset ({source},{tags}):')
        print(f'{timestamps[0]}, {timestamps[1]}, {timestamps[2]} ... {timestamps[-1]}')

 Timestamps available for variables air_temperature,eastward_wind,northward_wind,wind_speed_of_gust in dataset (COS
MO-2E,operational,numerical weather prediction,ensemble,surface,forecast):
2022-03-09T12:00:00, 2022-03-09T18:00:00, 2022-03-10T00:00:00 ... 2022-04-28T00:00:00
```

Example: read desired dataset using the Python API

Zarr

Chunked, compressed multi-dimensional arrays in flexible data stores

DASK

Flexible, general-purpose parallel computing framework.

xarray

High-level API for analysis of multidimensional labelled arrays.

```
In [1]: database = GridefixDatabase(metadata_store, chunk_store)

In [2]: source = 'COSMO-1E'
        tags = ['operational', 'numerical weather prediction', 'ensemble', 'surface', 'forecast']
        timestamps = ('2022-01-01', '2022-04-27') # range of timestamps when the data was generated
        variables =  ['eastward_wind', 'northward_wind', 'wind_speed_of_gust'] # variables of interest

In [3]: dataset = database.get_xarray_dataset(source, tags, timestamps, variables)

In [4]: dataset
```

Out[4]:

xarray.Dataset

▸ Dimensions: (**forecast_reference_time**: 395, **realization**: 11, t: 46, y: 252, x: 376)

▾ Coordinates:

forecast_refer...	(forecast_reference_time)	datetime64[ns]	2022-03-08T15:...
realization	(realization)	float64	0.0 1.0 2.0 3.0 .....
time	(forecast_reference_time, t)	datetime64[ns]	...
longitude	(y, x)	float64	...
latitude	(y, x)	float64	...

▾ Data variables:

northward_wind	(forecast_reference_time, t, realization, y, x)	float32	...
wind_speed_of...	(forecast_reference_time, t, realization, y, x)	float32	...
eastward_wind	(forecast_reference_time, t, realization, y, x)	float32	...

dask arrays: - convenient for manipulating big data → development/training use case.
- currently with single machine: local resources used.

Example: read desired dataset using export web service

language independent access with OPeNDAP

cloud-native → scalable

xarray dataset:

- lazy loading
- dask.compute() in service pod → not for manipulating big data → use case inference

slice across any dimension

In [7]:
```python
import xarray as xr
RETRIEVE_URL = 'http://retrieve-gridefix-main-prod.apps.cp.meteoswiss.ch/gridefix/api'
```

In [8]:
```python
# 4. get variables for last available timestamp containing all of them
dataset = xr.open_dataset(f'{RETRIEVE_URL}/sources/{source}/tags/{tags}/timestamps/{timestamps[-1]}/variables/{varia
dataset
```

Out[8]:

xarray.Dataset

▸ Dimensions: (**forecast_reference_time**: 1, **realization**: 21, t: 121, y: 127, x: 188)

▾ Coordinates:

forecast_refer...	(forecast_reference_time)	datetime64[ns]	2022-04-28
realization	(realization)	float64	0.0 1.0 2.0 3.0 .....
time	(forecast_reference_time, t)	datetime64[ns]	...
longitude	(y, x)	float64	...
latitude	(y, x)	float64	...

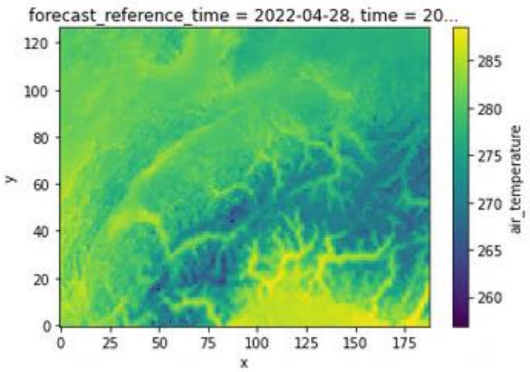
▾ Data variables:

air_temperature	(forecast_reference_time, t, realization, y, x)	float32	...
northward_wind	(forecast_reference_time, t, realization, y, x)	float32	...
wind_speed_of...	(forecast_reference_time, t, realization, y, x)	float32	...

In [9]:
```python
# 5. load only what you need
data_load = dataset.air_temperature.isel(forecast_reference_time=0, t=0).mean(dim='realization').load()
```

In [10]:
```python
data_load.plot()
```

Out[10]: <matplotlib.collections.QuadMesh at 0x7f0e8db7e350>



Summary and Future work

- GrideFix is a **data store** and **service** which offers unified access to multi-dimensional data.

- The cloud-native services allow to define **event-driven architectures**.

- Computations with large amount of data (e.g. training) possible through the Python API with **parallel computations** in dask.

Work in progress and future work:

- Async services: calls to PostgreSQL database not involving zarr.

- Use dask distributed on several machines.

- Quantify thoughtfully performance to quantify tradeoff: less performance at expense of handling metadata and rolling archive.

- Configurations for big archives
  - chunk store options: use several buckets in the cloud object store or local object store.
  - chunks: adapt chunking by use case of dataset in import functions

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology MeteoSwiss

MeteoSwiss
Operation Center 1
CH-8058 Zurich-Airport
T +41 58 460 91 11 www.meteoswiss.ch

MeteoSvizzera
Via ai Monti 146
CH-6605 Locarno-Monti
T +41 58 460 92 22
www.meteosvizzera.ch

MétéoSuisse
7bis, av. de la Paix
CH-1211 Genève 2
T +41 58 460 98 88
www.meteosuisse.ch

MétéoSuisse
Chemin de l‘Aérologie
CH-1530 Payerne
T +41 58 460 94 44
www.meteosuisse.ch

MeteoSwiss

Metadata store glue code

```
class PostgreSQLStore(MutableMapping):

    def __getitem__(self, key: str) -> str:
        with self.con().cursor() as c:
            c.execute('SELECT value::text FROM zarr_nodes WHERE path = %s', (key,))
            r = c.fetchone()

        if r:
            return r[0]
        else:
            raise KeyError(key)

    def __setitem__(self, key: str, value: bytes) -> None:
        # e.g.
        #   key = /COSMO-1E/98e81709-57dd-4bed-b318-ed261188056c/2022-03-30T12:00:00/northward_wind/.zarray
        #   value = {"dtype": "<f4", ...}
        #   -> will insert the zarr_node and the zarr_variable northward_wind with the timestamp 2022-03-30T12:00:00

        self.upsert({key: value})
    ...

class GridefixDatabase:
    metadata_store: PostgreSQLStore

    def __init__(self, metadata_store, chunk_store=None):

        self.metadata_store = metadata_store
        self.chunk_store = chunk_store

        self.root = zarr.group(store=self.metadata_store,
                               chunk_store=self.chunk_store)
    ...
```

GrideFix: Cloud-native database