earthmover

VirtualiZarr & Icechunk:

Build a cloud-optimised datacube in 3 lines

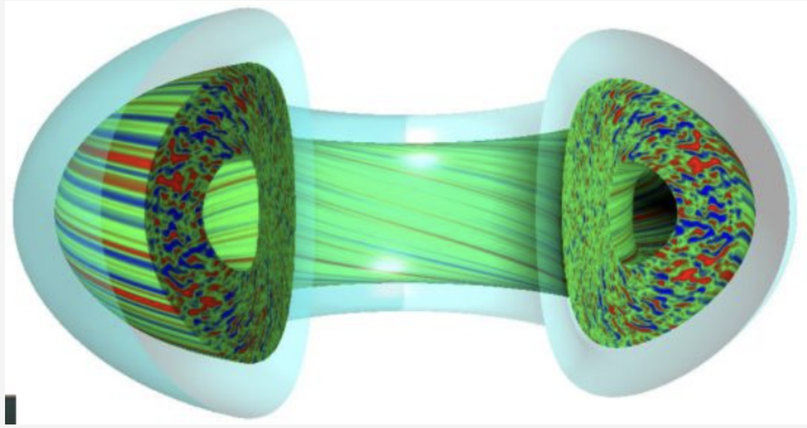
Tom Nicholas - SciPy - July 2025

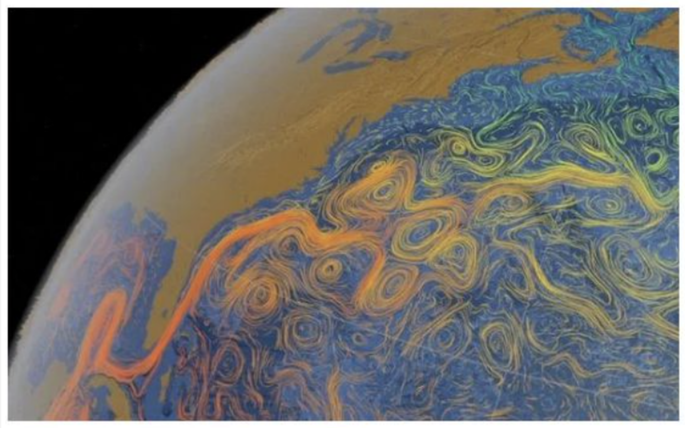
about me

Tom Nicholas, PhD
FORWARD ENGINEER

- Plasma Physicist
- Pivoted to geoscience data
- Open source maintainer

PRIOR EXPERIENCE





OPEN SOURCE CONTRIBUTIONS

xarray PANGEO Zarr

earthmover
the cloud platform for scientific data teams

MISSION STATEMENT

To empower people to use scientific data to solve humanity's greatest challenges

story

- ✓ 18 months ago I was asked to cloud-optimize some data - I'm finally done 😅
- ✓ Such a pain I wrote a new package - now it's 3 lines of Python 💪
- ✓ This approach (VirtualiZarr + Icechunk) should work for many datasets 🌐
- ✓ Avoids copying the data - a cloud-native bridge for archival data 📜☁️
- ✓ Icechunk's "multiplayer mode" means users can read data during updates!

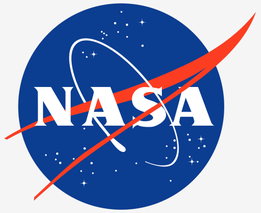
Collaborators:

VirtualiZarr: Max Jones, Sean Harkins, Aimee Barciaukas & Kyle Barron (DevSeed), Raphael Hagen (CarbonPlan), Julia Signell (Element84),

Icechunk: Sebastian Galkin & Deepak Cherian (Earthmover)

OAE Efficiency Map: Shane Loeffler & Kata Martin (CarbonPlan), Matt Long ([C]Worthy)

development SEED



earthmover

(carbon)plan

[C]Worthy

EARTHMOVER.IO

dataset: [C]Worthy OAE Efficiency Atlas

- Ensemble of climate model simulations
- Arranged in a grid (“tiles”)
- On S3 (in Source Cooperative’s bucket)

🙄 NetCDF

😬 50TB (200TB uncompressed)

😱 500,000 netCDF4 files!

😵‍💫 Logically 6-dimensional datacube

☝️ (3D + time + 2 ensemble dimensions)

[C]Worthy

SOURCE COOPERATIVE

amazon S3

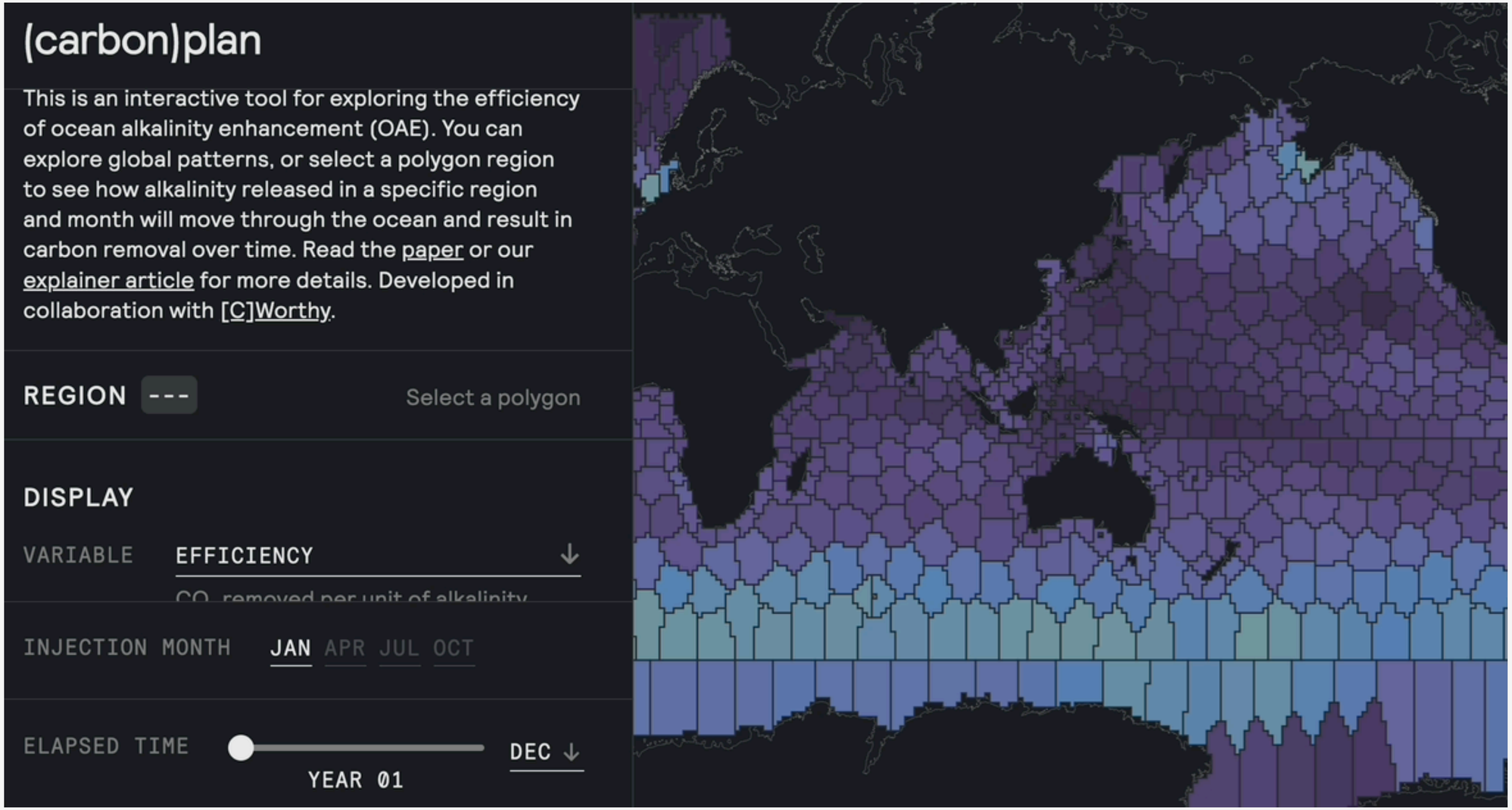
EARTHMOVER.IO

3 lines of code (almost)

```python
import virtualizarr as vz
import icechunk as ic

repo = ic.Repository.open_or_create(<path>)
session = repo.writable_session("main")

# (1) create chunk references
vds = vz.open_virtual_mfdataset('s3://bucket/**')

# (2) write references into icechunk
vds.virtualize.to_icechunk(session.store)

# (3) immutable commit
session.commit("wrote chunk references")
```

beautiful geospatial visualization

(carbon)plan



Shane Loeffler

HTTPS://CARBONPLAN.ORG/RESEARCH/OAE-EFFICIENCY/

EARTHMOVER.IO

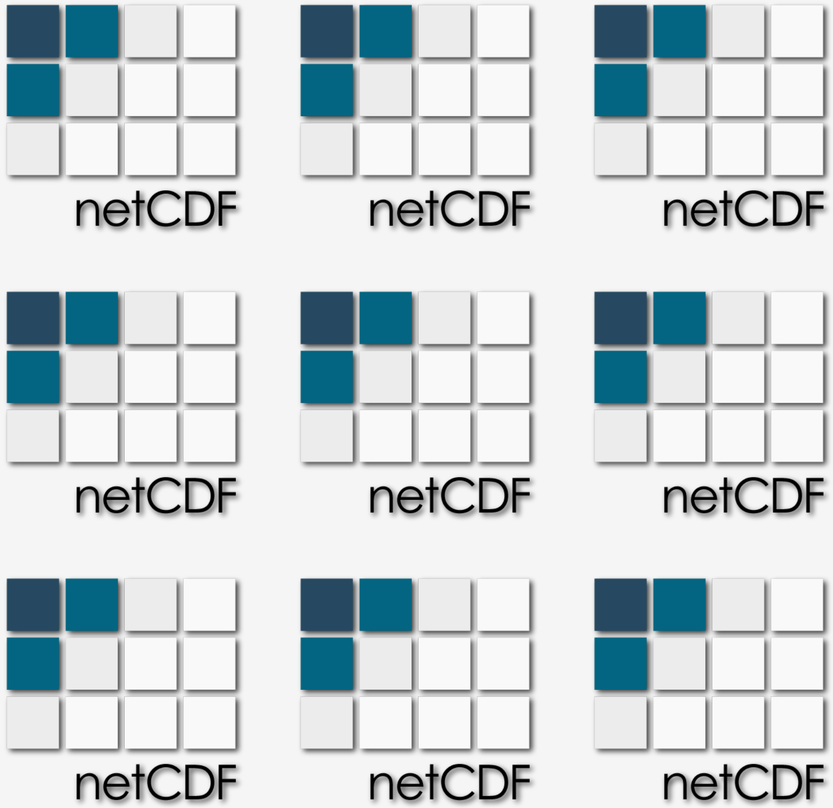
ugly distribution: “Pile of files / tiles”

→ What passes for a “cloud data archive”

- Result of a naive “lift and shift”

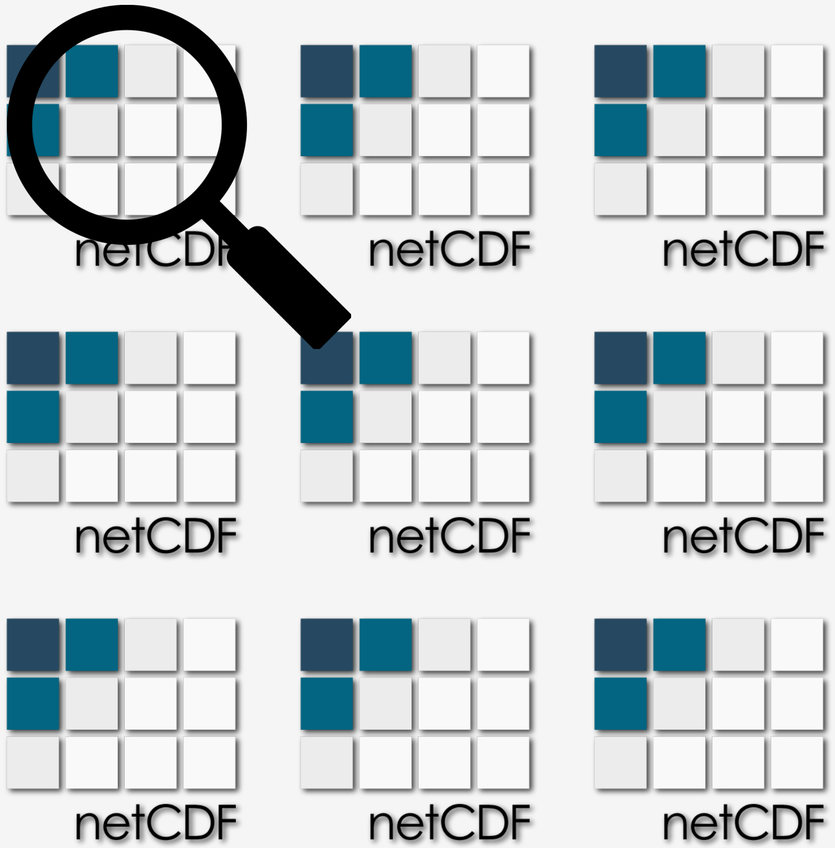
data has structure

- “Level 3 data” in geospatial jargon



problem: not “cloud-optimized”

- Client code has to look at metadata spread throughout each file
- Do this for every file, then check they can be combined
- If you called `xr.open_mfdataset` that’s what it would do



April 17, 2025 | 16 min read

Tom Nicholas
Forward Engineer

cloud native | fundamentals | icechunk | netCDF | open source | zarr

Fundamentals: What is Cloud-Optimized Scientific Data?

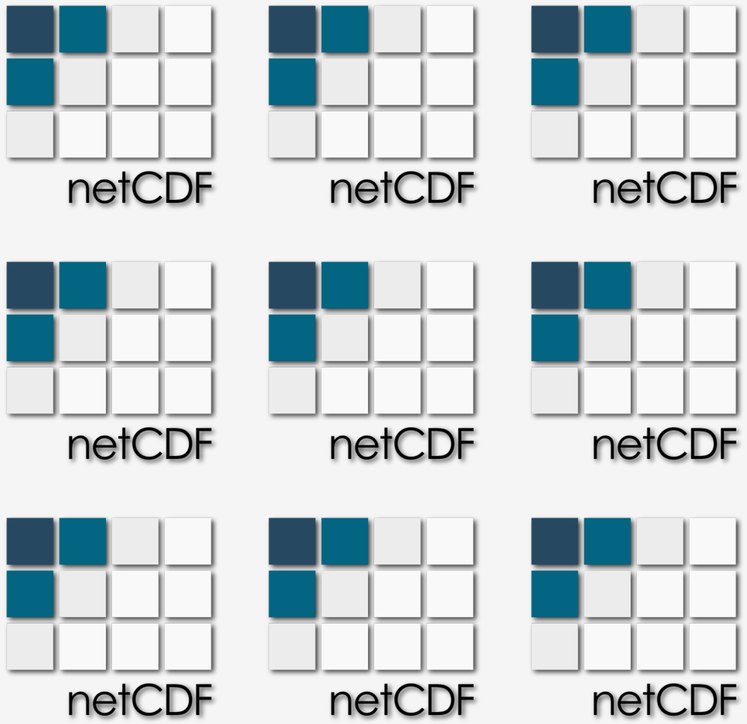
Why naively lifting scientific data to the cloud falls flat.

Time just to open:

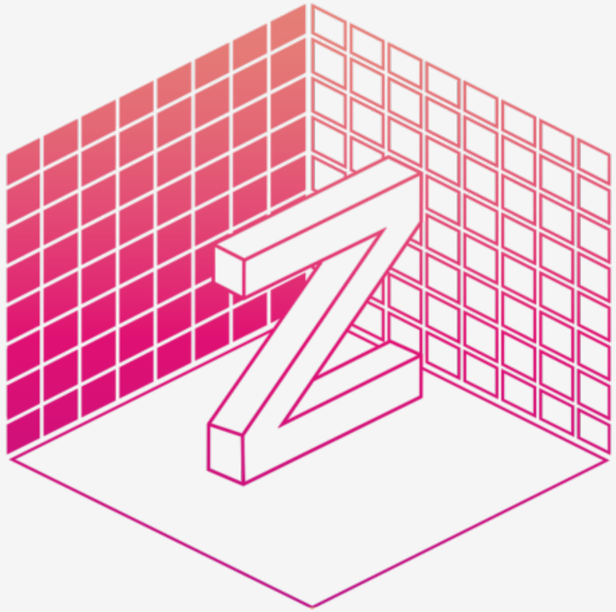
1 minute per file x 500,000 files = **an entire year!!!** 🤯

rewrite as Zarr!

- Time to open now <1 second! 🥳
- Uses 2x the storage 👎

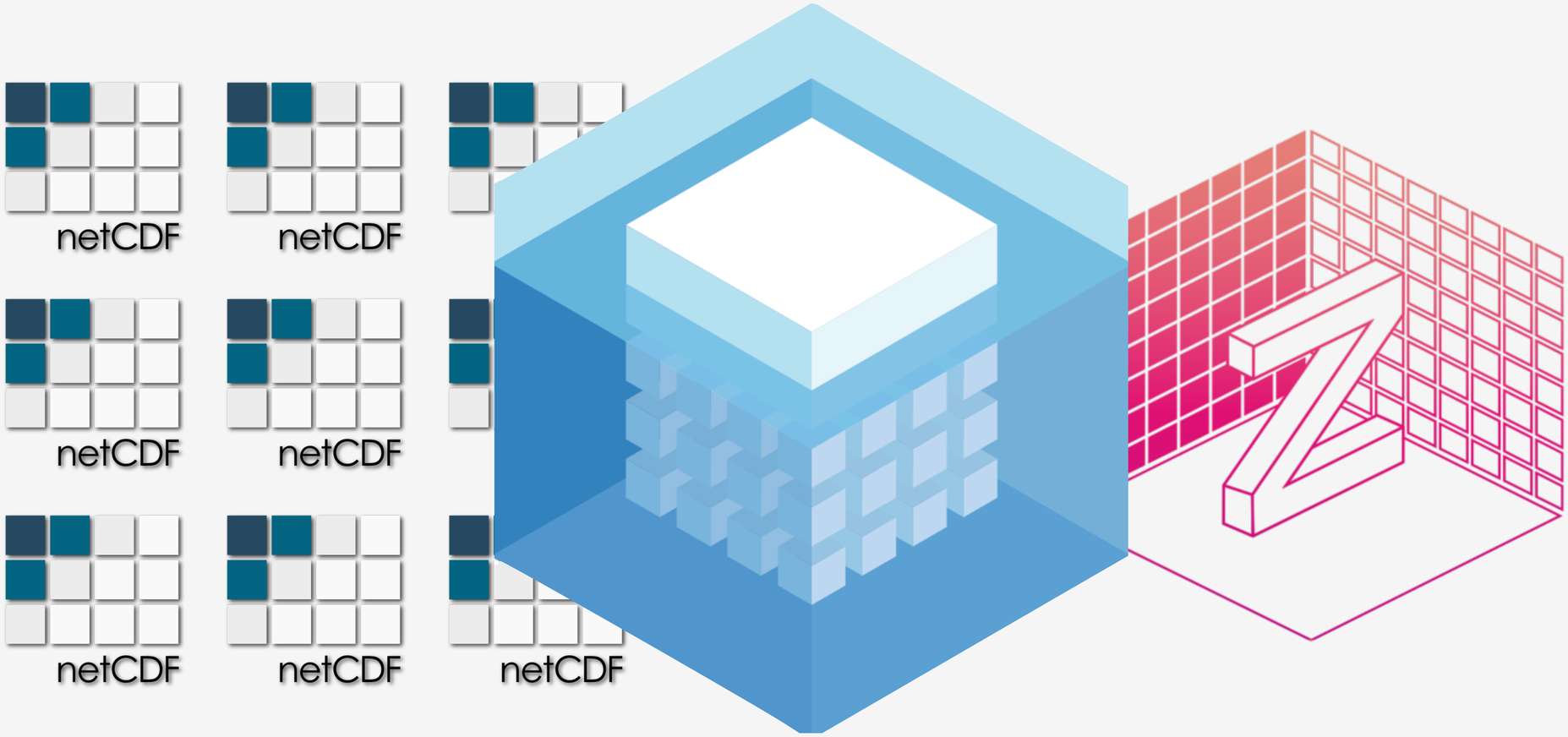


50TB



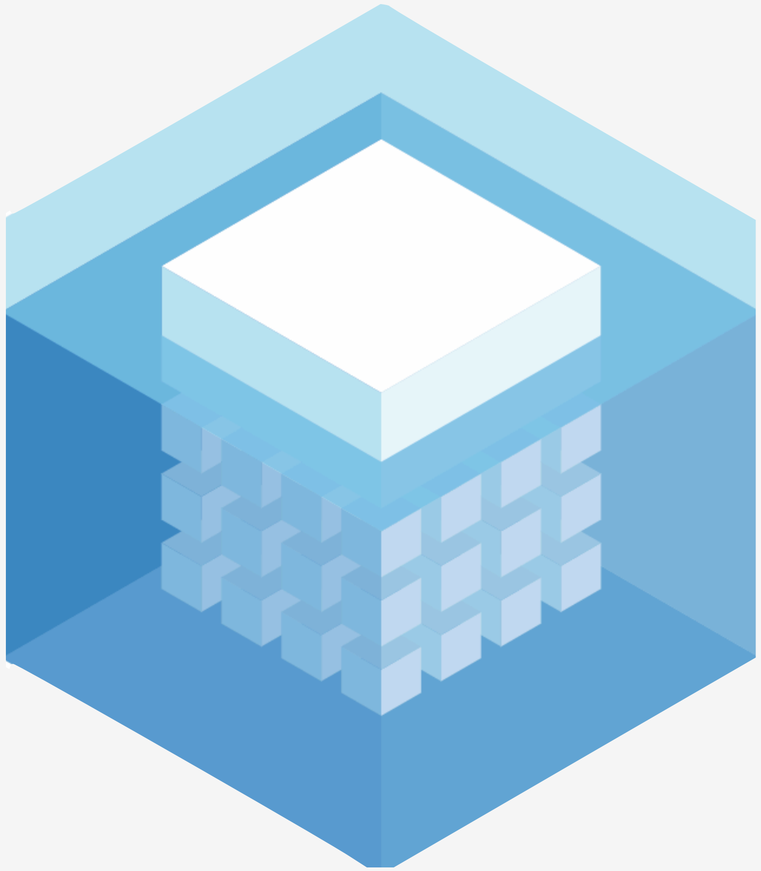
+50TB

but what if it just looked like Zarr... 🤔



Icechunk stores “virtual chunks”

- Transactional, cloud-native storage engine for Zarr
- Works together with Zarr Python 3 and Xarray
- Supports virtual chunks



ICECHUNK

- Core implemented in Rust; thin Python wrapper
- 100% open source (Apache 2.0)

https://icechunk.io

https://github.com/earth-mover/icechunk/

VirtualiZarr extracts virtual chunks

```
import virtualizarr as vz

vz.open_virtual_dataset(
```

)

“chunk manifests”

“Virtual” `xr.Dataset`(

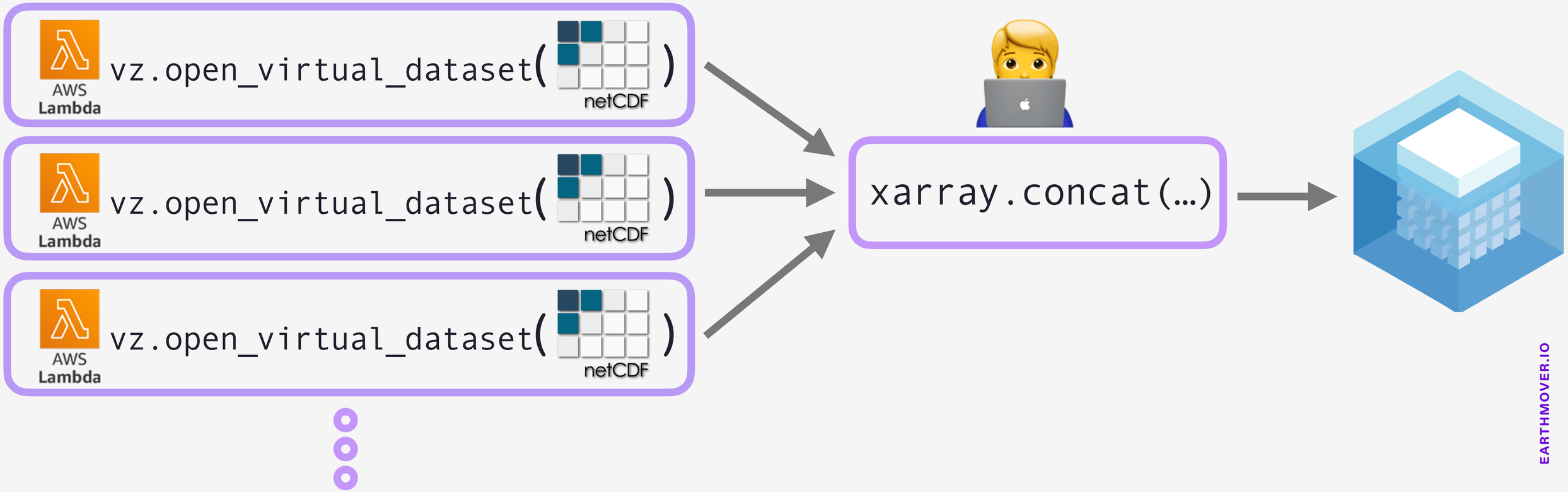
```
{
    "0.0.0": {"path": "s3://bucket/foo.nc", "offset": 100, "length": 100},
    "0.0.1": {"path": "s3://bucket/foo.nc", "offset": 200, "length": 100},
    "0.1.0": {"path": "s3://bucket/foo.nc", "offset": 300, "length": 100},
    "0.1.1": {"path": "s3://bucket/foo.nc", "offset": 400, "length": 100},
}
```

)

- Combining chunk references from different files == **array concatenation**
  - So use `xarray.concat()`! (By wrapping `ManifestArrays`)
- Can write references to Icechunk stores
  - (Or to the Kerchunk format)

Scaling to 500,000 files

- Map-reduce problem, called via `vz.open_virtual_mfdataset(<filepaths>)`
  - Serverlessly map `vz.open_virtual_dataset` over files as AWS Lambdas
    - (Max 1000 lambdas at once, so batch this)
- Demo...



Example

- 10GB / s
- From laptop
- Batched
- Resumable
- General
- 3 lines (ish...)

```python
loops over batches of 720 files
- polygon_id in polygon_ids[1:]:
  session = repo.writable_session("main")

  # parse and combine virtual references in parallel
  combined_vds_single_polygon = vz.open_virtual_mfdataset(
      list_objects_nested(polygon_id),
      combine="nested", concat_dim=["injection_date", "elapsed_time"],
      coords="minimal", data_vars="minimal", compat="override", join='override',
      decode_times=True,
      backend=HDFVirtualBackend,
      virtual_backend_kwargs={'cache': True},
      parallel='lithops',
  ).virtualize.rename_paths(http_to_s3).drop_vars(LOW_DIMENSIONAL_VARS, errors='ignore')

  # write virtual references to icechunk
  combined_vds_single_polygon.virtualize.to_icechunk(session.store, append_dim='polygon_id')

  # commit them
  msg = f"wrote chunk refs for {polygon_id}"
  print(msg, session.commit(msg))
```

14.9s Python

```
2025-04-26 19:31:11,685 [INFO] config.py:139 -- Lithops v3.6.1.dev0 - Python3.12
2025-04-26 19:31:11,762 [INFO] aws_s3.py:59 -- S3 client created - Region: us-west-2
2025-04-26 19:31:12,484 [INFO] aws_lambda.py:97 -- AWS Lambda client created - Region: us-west-2
2025-04-26 19:31:12,486 [INFO] config.py:139 -- Lithops v3.6.1.dev0 - Python3.12
```

Watch out for releases! ✨

→ Replicate demo yourself with upcoming VirtualiZarr 2.0 release

- Scaling demo some unreleased features in VirtualiZarr 2.0
  - But VirtualiZarr and Icechunk work together today!
- Icechunk 1.0 was also released today!
  - Production-ready
  - Stable, backwards-compatible on-disk format! (With open specification)

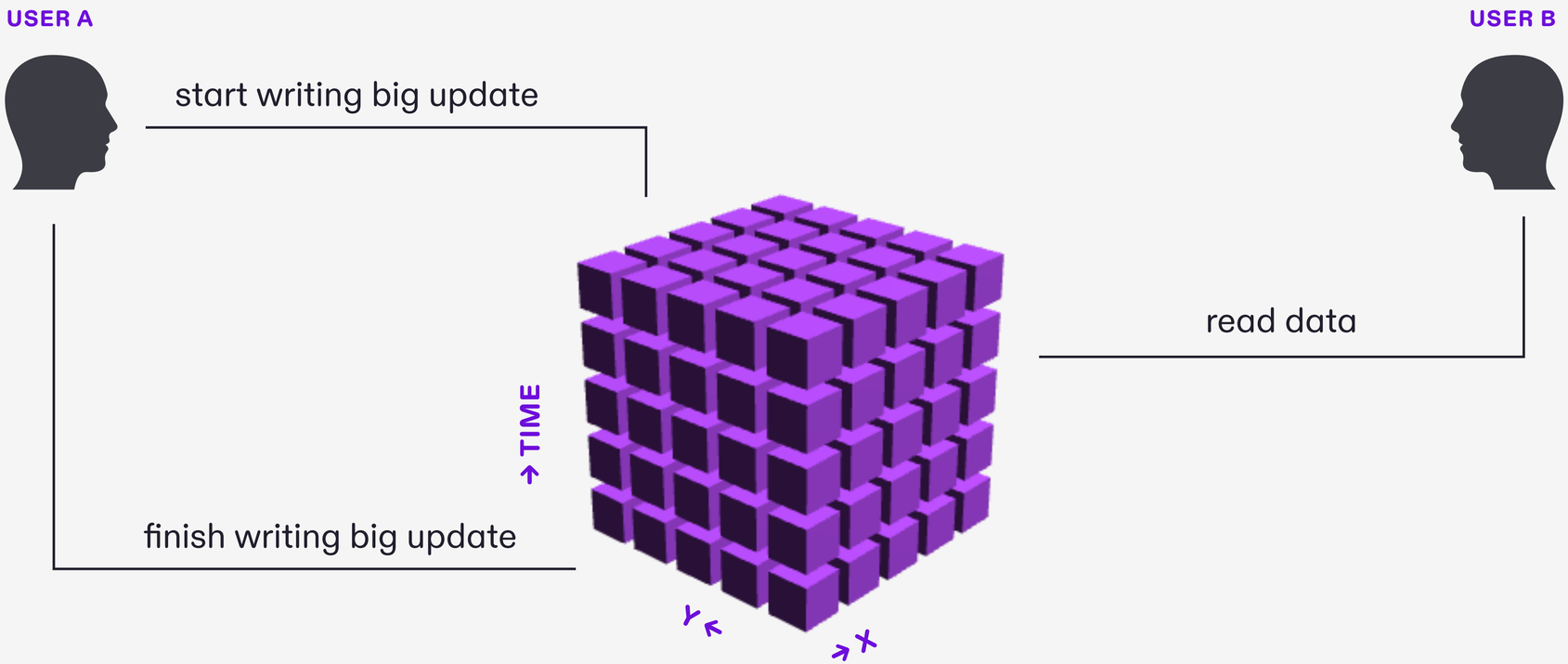
Reading whilst writing?

- Each loop iteration adds more data to the store
- Real-world data often needs to do this periodically on ongoing basis
  - e.g. each time a NASA imaging satellite transmits another image
- Want a full history of changes

- Also want users to be able to safely read previous data *whilst new data is being ingested*...

Zarr without Icechunk

→ Zarr is not designed for “multiplayer mode”



These issues stem from the fact that Zarr is *not* a monolithic file format.
Zarr data is spread over many files. (More like a database.)

Zarr with Icechunk: Multiplayer mode

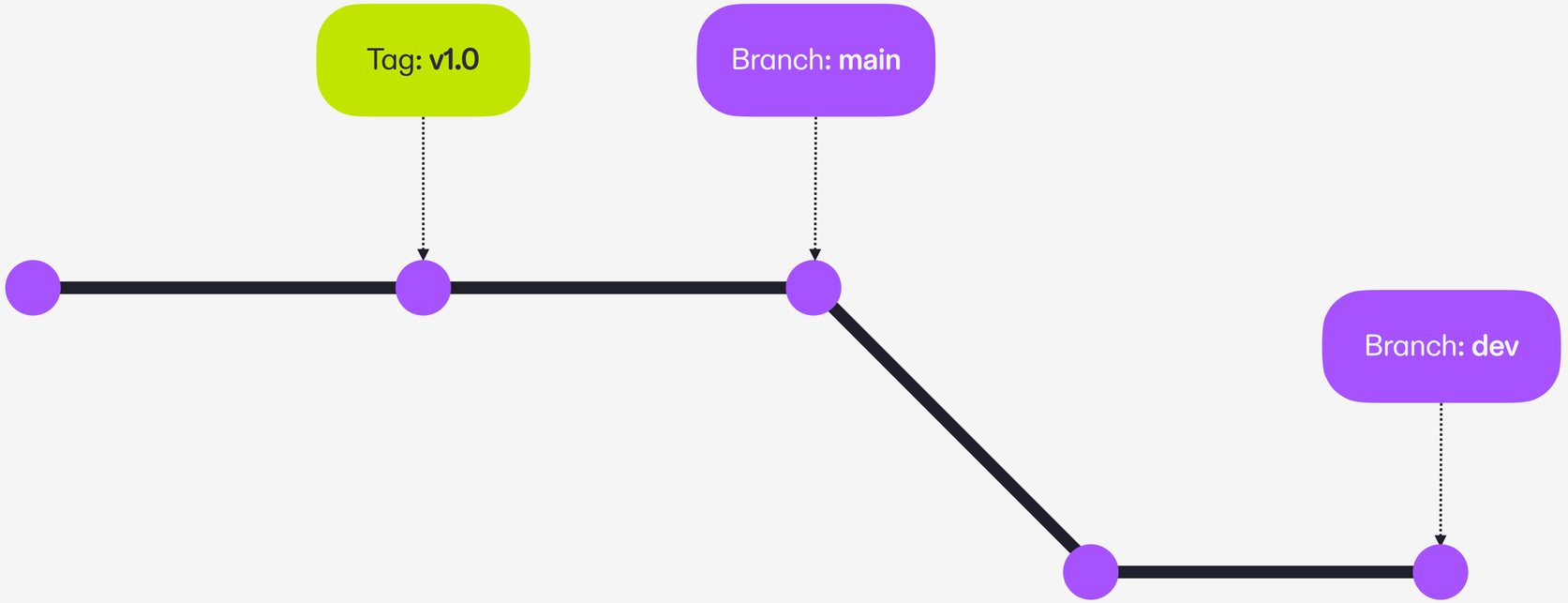
→ transactions enable safe collaboration

- ✓ All updates occur with a **transaction** and create a new **snapshot** of the dataset
- ✓ **Serializable isolation** between transactions; readers only ever see a committed snapshot
- ✓ **No locks** required for reading or writing data
- ✓ **Optimistic concurrency control** for detecting and resolving write conflicts

These features make Zarr work **more like a database.**

Git-like version control for Array data

→ Snapshots, Branches, Tags



“Which file formats?”

- Currently can parse netCDF4, HDF5, netCDF3, “native” Zarr (v3), FITS
- TIFF, COG, GRIB coming soon
- In VirtualiZarr v2.0 you can write your own custom “Parser” for a more niche format
  - e.g. WIP parser for HuggingFace’s SafeTensors format for ML model weights

```python
@runtime_checkable
class Parser(Protocol):
    def __call__(
        self,
        file_url: str,
        object_store: ObjectStore,
    ) -> ManifestStore:
        ...
```

EARTHMOVER.IO

Aside: Parsers as a runtime Zarr translation layer

- Parser returns a “ManifestStore”
  - Zarr-python/Xarray can read this store directly
  - Uses obstore rust crate under the hood

```python
from virtualizarr.parsers import TiffParser

manifest_store = TiffParser("<file.tiff>")

ds = xr.open_zarr(manifest_store)
```

- So you don’t actually need to serialize to Kerchunk / Icechunk format to read data back…
  - i.e. use VirtualiZarr as a *runtime translation layer* to read any (parseable) format as Zarr!
- (This only makes sense for already-cloud-optimized data, or when running outside of the cloud)

Summary

- ✓ Level 3/4 datasets often in archival formats with Zarr-like (array) structure
- ✓ *Virtual* Icechunk stores point at files *without copying the data*
- ✓ Build virtual datacubes using Xarray syntax via VirtualiZarr
- ✓ Icechunk allows incremental updates as new data arrives

Format	**NetCDF4**	**“Native” Zarr**	**Icechunk**
# of URLs	500,000	1	1
Time to open	~1 year	< 1 sec	< 1 sec
Storage increase	0%	100%	<0.0004%
Convert using Xarray?	N/a	Yes	Yes
Version-controlled?	No	No	Yes
Update-safe?	No	No	Yes

"But you can’t change the chunks”

- Correct. That is the primary downside of this approach.
- You can choose to write native chunks alongside the virtual chunks though
  - Allows you to ensure small coordinate variables are all one chunk
  - Allows you to incrementally overwrite data with more suitable chunking after virtual ingestion

Bonus: “What datasets can you virtualize?”

- Currently some additional requirements imposed by Zarr data model
  - 1 chunk = 1 HTTP range request
  - Homogenous chunk shapes
  - Homogenous chunk codecs (e.g. compression)
  - No other per-chunk metadata, only per-array
- Some of these could be relaxed by future Zarr development...

Bonus: “What about Kerchunk?”

✓ Kerchunk is two things:

1. Python package
   - VirtualiZarr package replaces this
2. Format for storing references
   - Icechunk format is an alternative
   - (Though VirtualiZarr can write to both)

Format	NetCDF4	“Native” Zarr	Kerchunk	Icechunk
# of URLs	500,000	1	1	1
Time to open	~1 year	< 1 sec	< 1 sec	< 1 sec
Storage increase	0%	100%	0.0004%	0.0004%
Convert using Xarray?	N/a	Yes	No	Yes
Version-controlled?	No	No	No	Yes
Update-safe?	No	No	No	Yes

Bonus: “How does the parallelization work?”

- vz.open_virtual_mfdataset accepts an Executor
  - Follows concurrent.Futures interface (idea from Cubed)
  - Comes with a LithopsExecutor a DaskDelayedExecutor, and works with the python ThreadPoolExecutor
- Lithops
  - Open-source package abstracting over serverless APIs of various cloud providers
  - Had to build a runtime using Docker first
  - But then my python function just runs on AWS Lambdas