Accelerate research and reduce costs with cloud-optimised data formats

eResearch 2025

Steve Gillard

Senior Technologist
Amazon Web Services

Agenda

The problem statement

Defining “cloud optimised”

Examples of cloud optimised formats

Introducing Zarr for multi-dimensional scientific data

Some new tools to aid Zarr adoption

A benchmark comparison

Problem Statement

How can we analyse petabyte scale data with:

1. Greater accuracy / higher resolution
2. Faster performance
3. Lower cost
4. Better energy efficiency

As well as aligning with FAIR & CARE principles

Defining “Cloud Optimised”

1. The metadata describing structure and content can be read quickly

2. Data can be read using the HTTP protocol without the assumption of file system paths (ie, object storage)

3. Data can be organised in groupings that allow for efficient subsetting and distributed processing (eg, chunked)

R. P. Abernathey *et al.*, "Cloud-Native Repositories for Big Scientific Data," in *Computing in Science & Engineering*, vol. 23, no. 2, pp. 26-35, 1 March-April 2021, doi: 10.1109/MCSE.2021.3059437. https://ieeexplore.ieee.org/document/9354557

Types of Storage

	Block	File	Object
Example use	Local Volumes	Shared Filesystem	Data Lake
Protocols	iSCSI / FC	NFS / SMB / Lustre	HTTP / REST
# of Clients	Few	Thousands	Millions
Request Latency	Microseconds	<1 - 2 Milliseconds	10's Milliseconds
Throughput	250 – 4,000 MiB/s	3 – 10 GiB/s	10's GiB/s is achievable
Capacity	TBs	TBs → PBs	Scales to EBs per bucket
Durability	Low (3 9's)	Configuration dependent	High (11 9's)

Examples of Cloud Optimised Data Formats

Tabular data: Parquet, ORC

Imagery: Cloud Optimized GeoTIFF (COG)

Multi-dimensional arrays: Zarr

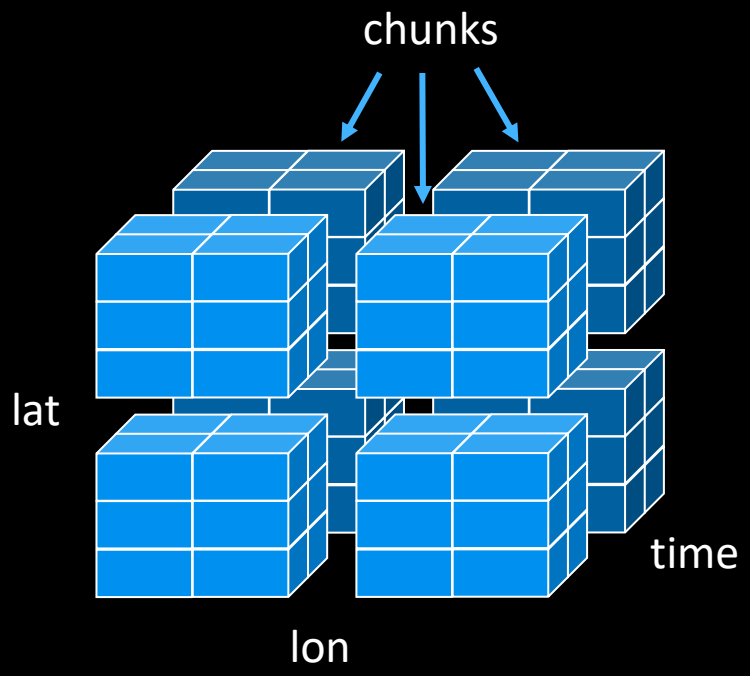
LiDAR data: Cloud Optimized Point Cloud (COPC)

Introducing Zarr

- A specification for storage of large N-dimensional arrays with chunking and compression
- Supports high-throughput distributed I/O on different storage systems
- Implementations in Python, R, C, C++, Rust, Javascript and Java
- Adoption increasing across Earth sciences and in medical imagery

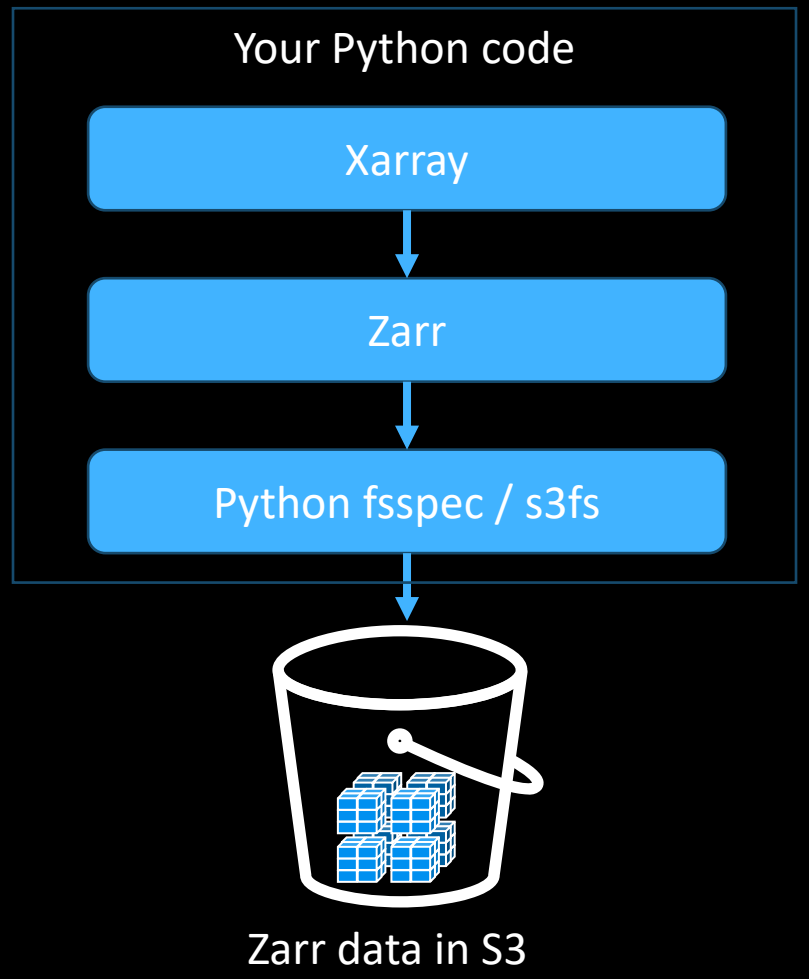
Introducing Zarr

https://zarr.dev/



- Hierarchical structure with stores, groups and arrays
- Metadata at each level in JSON
- Each chunk is stored in a separate object or file allowing:
  - Selective retrieval of only what you need
  - Parallel reading / writing
- Choosing the optimal chunk size and shape is an important consideration!
  - I recommend >5MB per chunk (compressed) when using cloud object storage

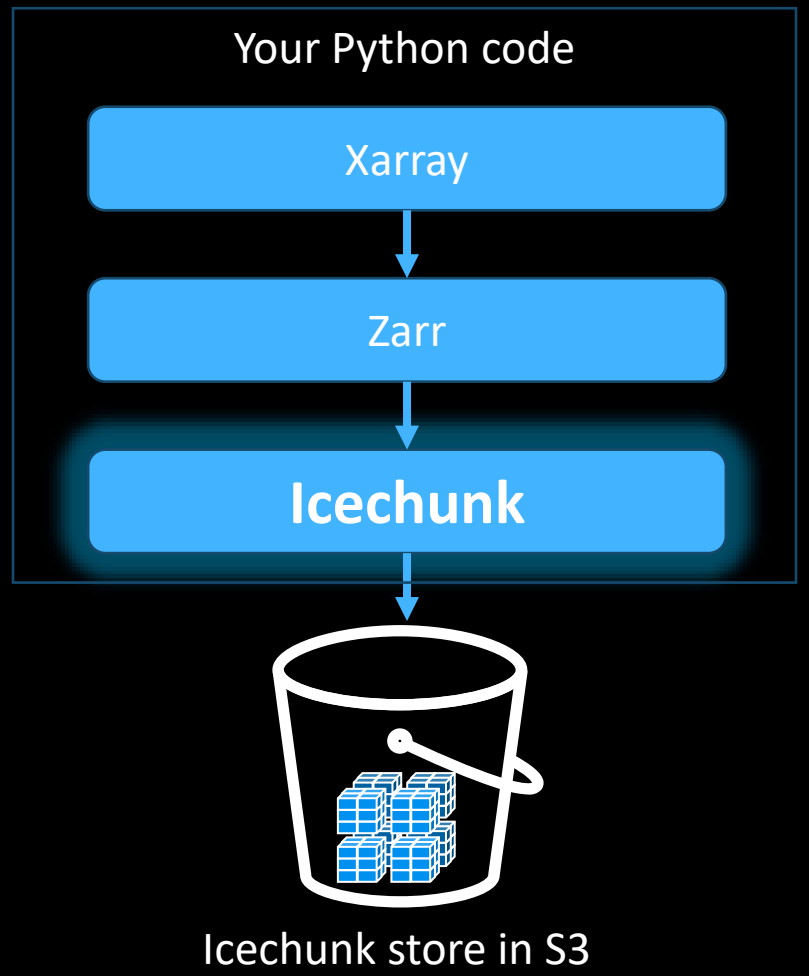
Zarr-Python Storage Integration



- Uses Python's fsspec for I/O
- Access data directly in S3 with no intermediate store!
- Quite fast for many use cases
- Can present challenges in:
  - Consistency while updates are in progress
  - Sychronisation of concurrent writers
  - Performance and scalability (limited in part by Python's GIL)

Introducing Icechunk

https://icechunk.io/

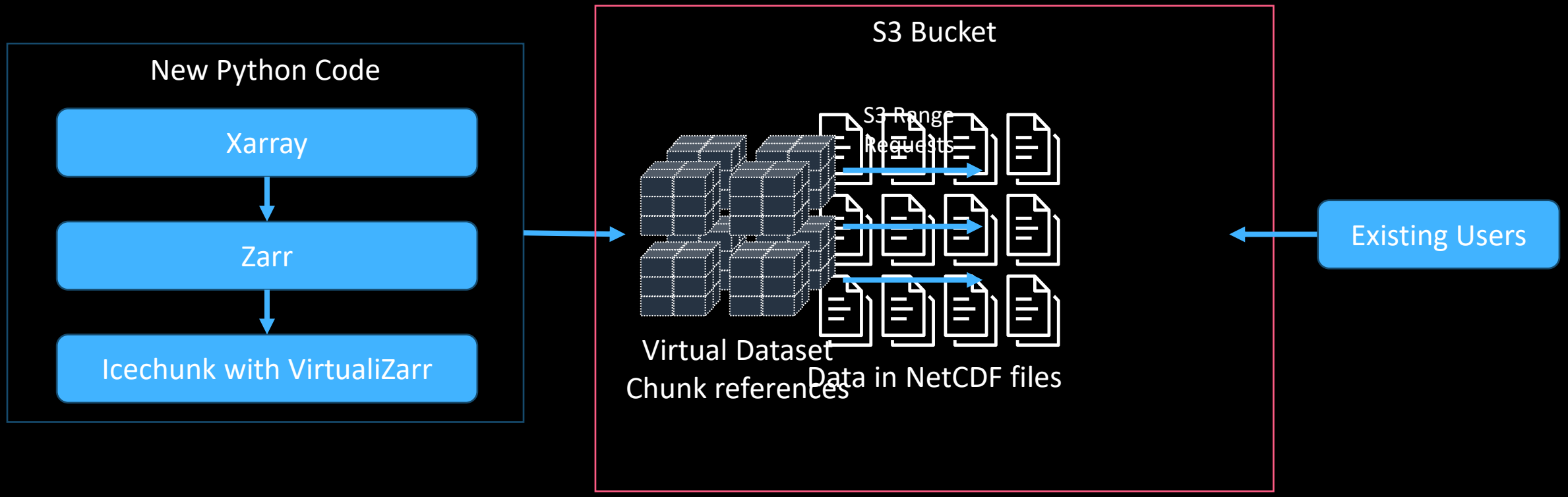


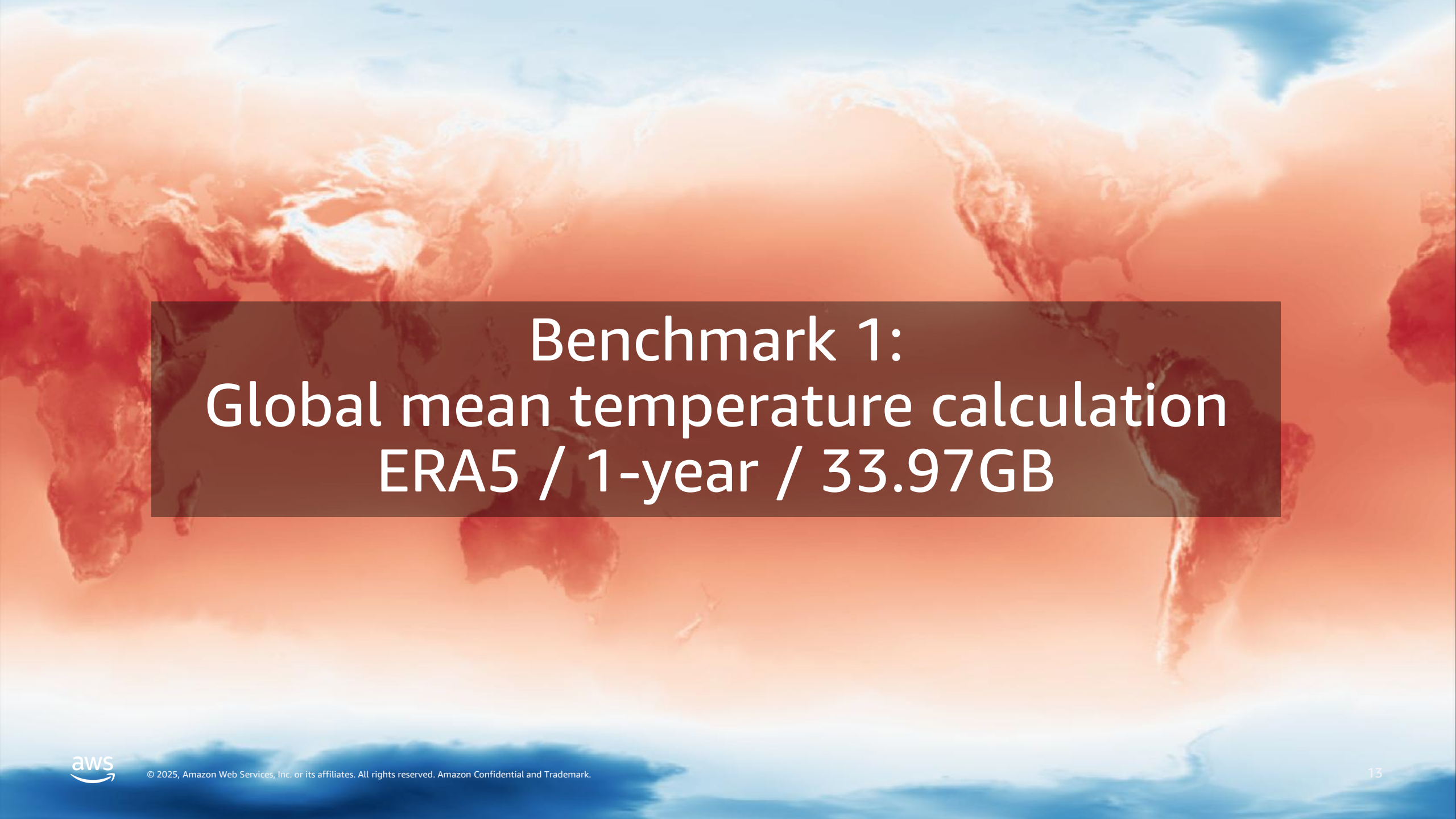
Icechunk store in S3

- Icechunk is an alternative tensor storage engine based on the Zarr data model
- Open source, developed by Earthmover
- Inspired by Apache Iceberg
- Provides:
  - Transactional isolation
  - Snapshots & time travel
  - Versioning & branching
- Written in Rust with a Python interface – its scalable and fast!

What if you must support existing users and tools and can’t convert to Zarr?

Speed up archive data with VirtualiZarr

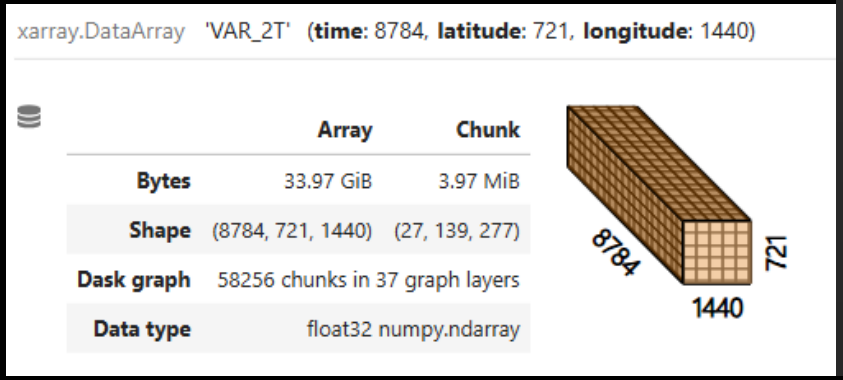


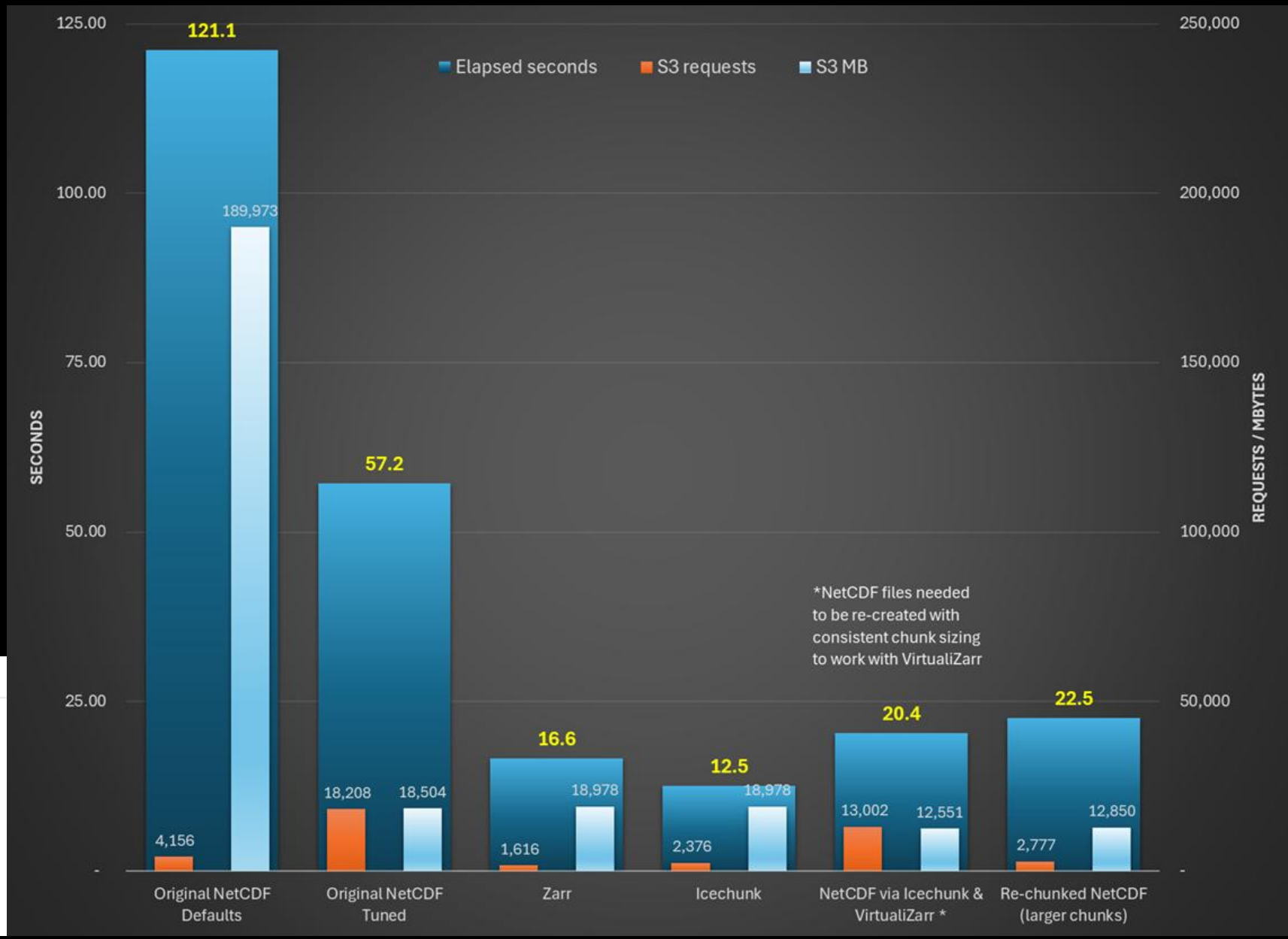


Benchmark 1:
Global mean temperature calculation
ERA5 / 1-year / 33.97GB

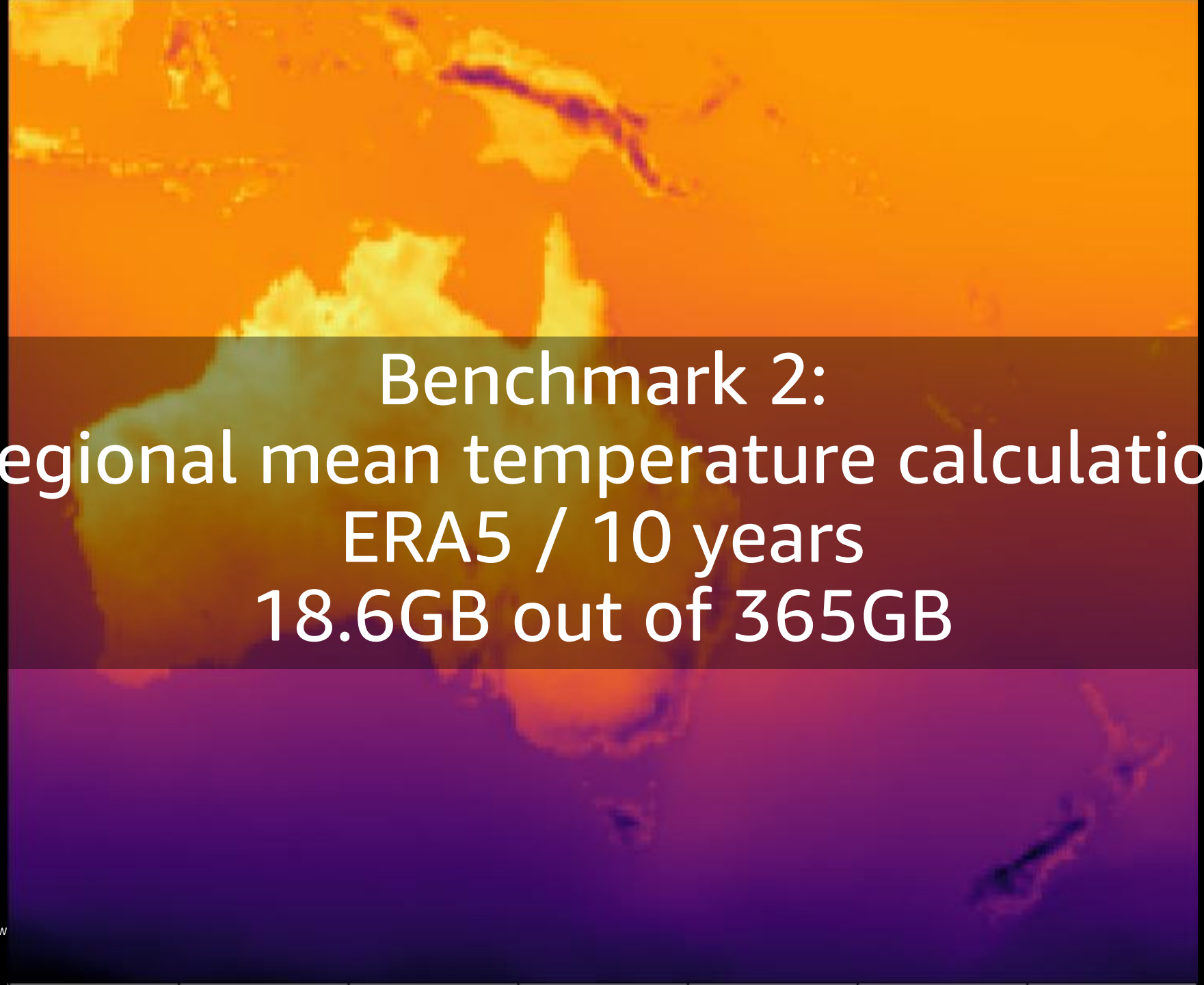
Benchmark 1

- Mean calculation across a full year of ERA5 temperature data (34GB uncompressed)
- Processed using Xarray and Dask with 24 workers
- Measuring:
  - Elapsed time in seconds
  - Number of S3 requests
  - Amount of data retrieved





Benchmark machine: AWS r7a.12xlarge, 48 vCPUs, 384GB RAM, 18.75 Gbps Bandwidth



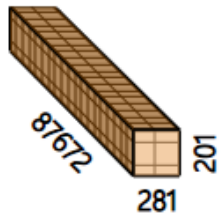
Benchmark 2:
Regional mean temperature calculation
ERA5 / 10 years
18.6GB out of 365GB

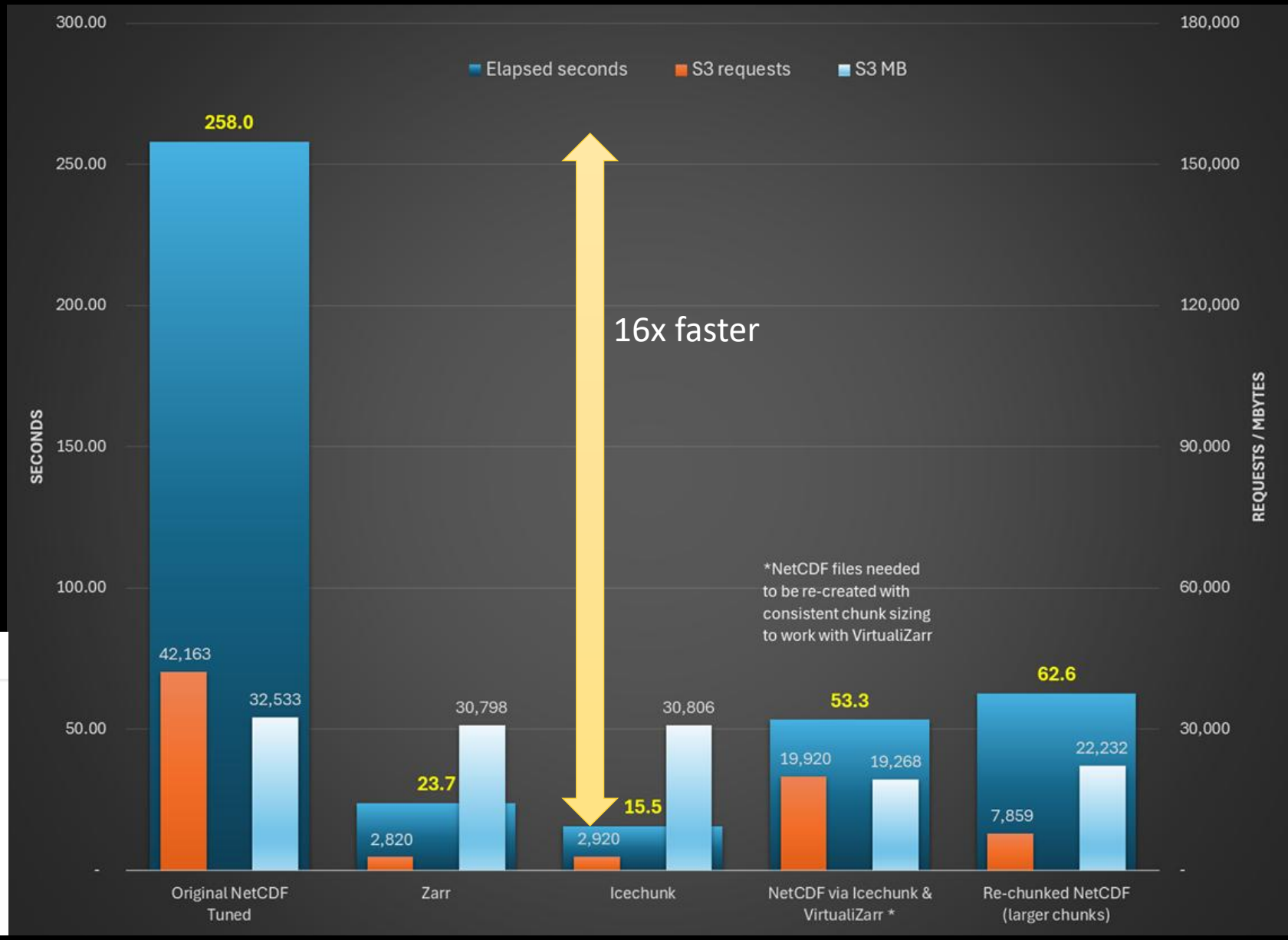
Benchmark 2

- ANZ region temperature mean using ERA5
- Across a 10-year dataset (18.45GB out of 339GB)
- Processed using Xarray and Dask with 24 workers
- Measuring:
  - Elapsed time in seconds
  - Number of S3 requests
  - Amount of data retrieved

xarray.DataArray 'VAR_2T' (**time**: 87672, **latitude**: 201, **longitude**: 281)

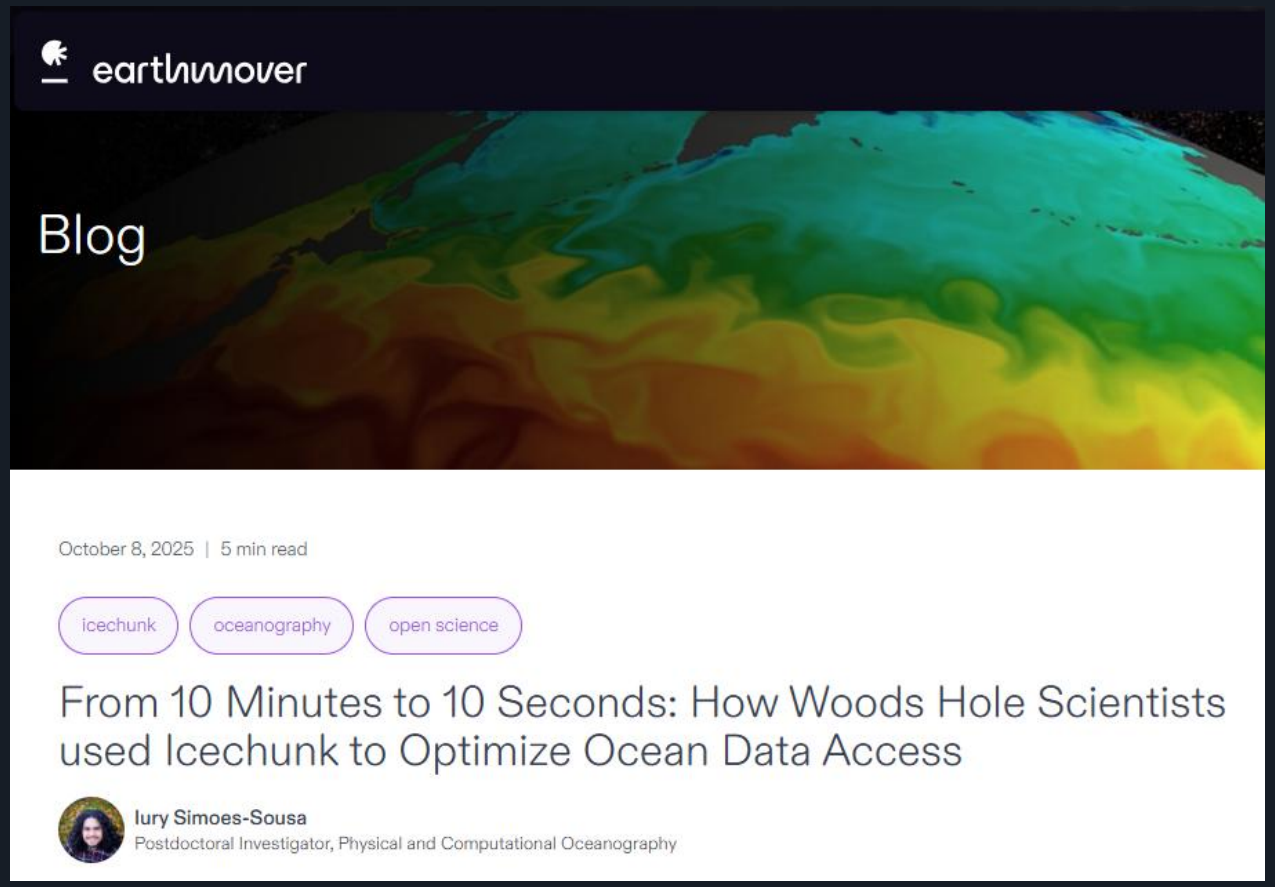
	Array	Chunk
Bytes	18.45 GiB	10.94 MiB
Shape	(87672, 201, 281)	(192, 103, 145)
Dask graph	2760 chunks in 22 graph layers	
Data type	float32 numpy.ndarray	

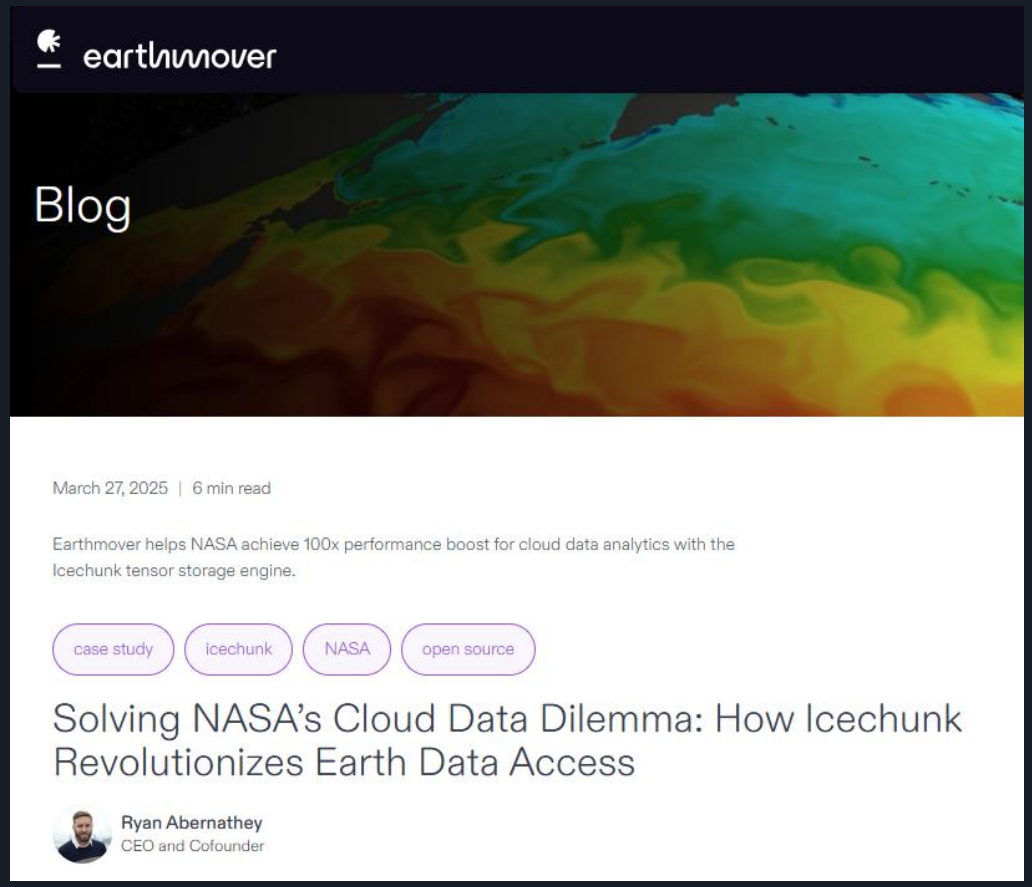




Benchmark machine: AWS r7a.12xlarge, 48 vCPUs, 384GB RAM, 18.75 Gbps Bandwidth

Success stories: Icechunk + Amazon S3





https://earthmover.io/blog/

Take aways

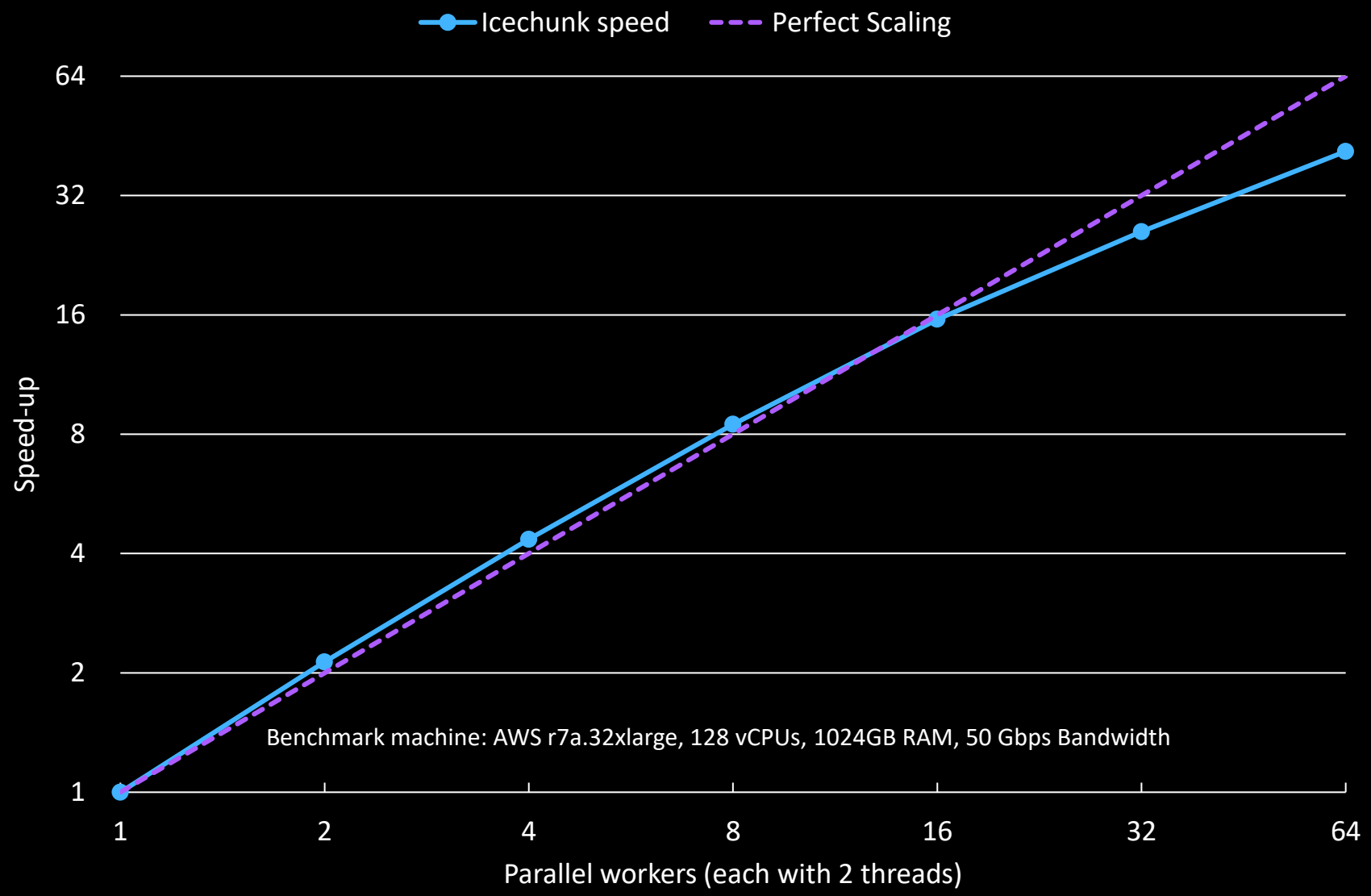
- Strongly consider using Zarr and Icechunk for storage of large multi-dimensional scientific datasets
- If legacy tools and uses must be supported, consider providing VirtualiZarr (or kerchunk) indexes
  - (You may still need to re-chunk the data)
- Choose your chunk size and shape carefully!

- Further information:
  - Some most excellent articles on Earthmover's blog on Zarr and Icechunk: https://earthmover.io/blog/
  - Participate in the Pangeo open source community: https://pangeo.io/
  - Benchmark calculations and Zarr conversion code is available in this workshop (Icechunk coming soon!): https://catalog.workshops.aws/climatedata/

Thank you!

Steve Gillard
gillards@amazon.com

Icechunk scaling – regional mean benchmark



6.87 seconds
35.9 Gbps
4.48 GB/s

Open Icechunk Store

[2]:
```
zarrbucketname =
```

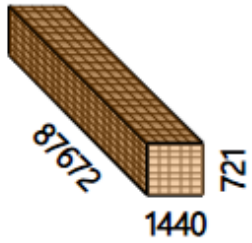
[3]:
```
import icechunk
import xarray as xr

storage = icechunk.s3_storage(bucket=zarrbucketname, prefix="icechunk-xl", from_env=True)
repo = icechunk.Repository.open(storage)
session = repo.readonly_session(branch="main")
ds = xr.open_zarr(session.store, zarr_format=3, consolidated=False)
```

[4]:
```
ds.VAR_2T
```

[4]: xarray.DataArray 'VAR_2T' (**time**: 87672, **latitude**: 721, **longitude**: 1440)

	Array	Chunk
Bytes	339.09 GiB	21.73 MiB
Shape	(87672, 721, 1440)	(192, 103, 288)
Dask graph	15995 chunks in 2 graph layers	
Data type	float32 numpy.ndarray	



▼ Coordinates:

latitude	(latitude)	float64	90.0 89.75 89.5 ... -89.75 -90.0
longitude	(longitude)	float64	0.0 0.25 0.5 ... 359.2 359.5 359.8
time	(time)	datetime64[ns]	2015-01-01 ... 2024-12-31T23:00:00

► Indexes: (3)

► Attributes: (14)

Opening and reading the Icechunk virtual repo

[1]:
```
import xarray as xr
import icechunk
from virtualizarr import open_virtual_dataset, open_virtual_mfdataset
from virtualizarr.parsers import HDFParser
from virtualizarr.registry import ObjectStoreRegistry
```

[2]:
```
icechunk_store = icechunk.s3_storage(bucket=                    prefix="icechunk-virtual10y", from_env=True)
config = icechunk.RepositoryConfig.default()
credentials = icechunk.containers_credentials({"s3://sgtest-climate-data-us-west-2/": icechunk.s3_credentials()})
repo = icechunk.Repository.open(icechunk_store, config, credentials)
session = repo.readonly_session("main")
```

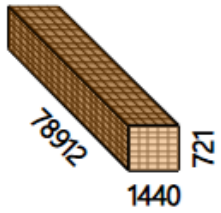
[3]:
```
ds = xr.open_zarr(session.store, zarr_format=3, consolidated=False, chunks={})
```

```
/home/ec2-user/SageMaker/.micromamba/envs/earthmover2/lib/python3.12/site-packages/numcodecs/zarr3.py:164: UserWarning: Numcodecs
  super().__init__(**codec_config)
```

[4]:
```
ds.VAR_2T
```

[4]: xarray.DataArray 'VAR_2T' (**time**: 78912, **latitude**: 721, **longitude**: 1440)

	Array	Chunk
Bytes	305.21 GiB	2.72 MiB
Shape	(78912, 721, 1440)	(24, 103, 288)
Dask graph	115080 chunks in 2 graph layers	
Data type	float32 numpy.ndarray	



▼ Coordinates:

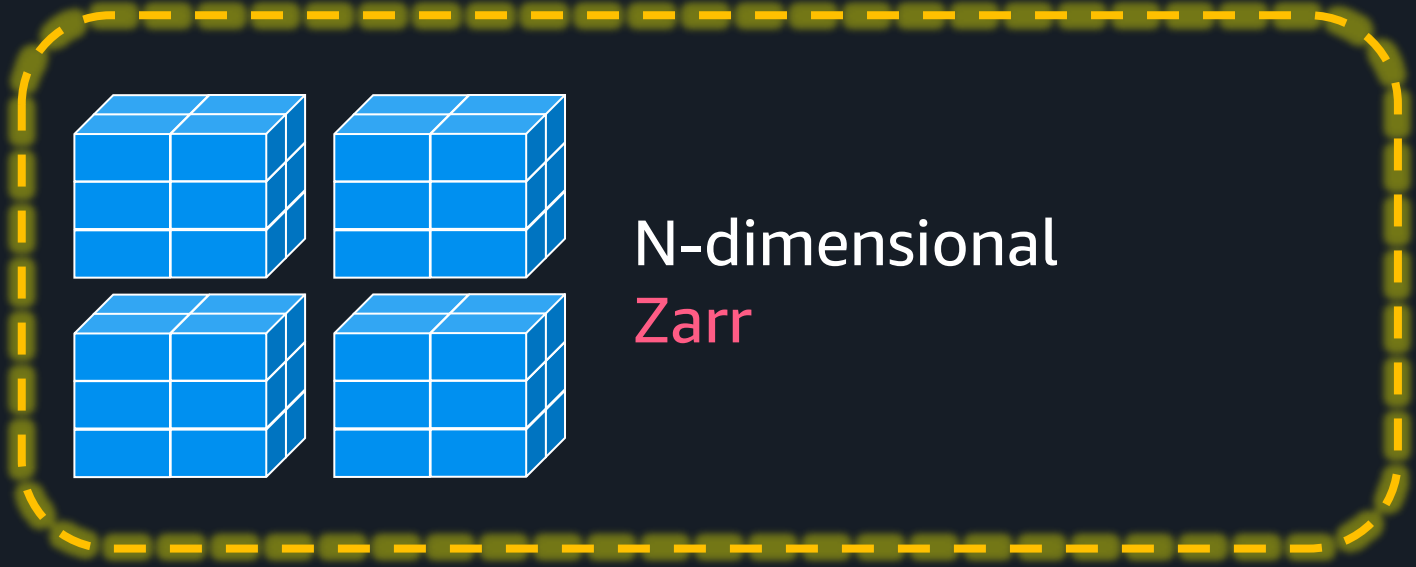
latitude	(latitude)	float64	90.0 89.75 89.5 ... -89.75 -90.0
longitude	(longitude)	float64	0.0 0.25 0.5 ... 359.2 359.5 359.8
time	(time)	datetime64[ns]	2016-01-01 ... 2024-12-31T23:00:00

► Indexes: (3)

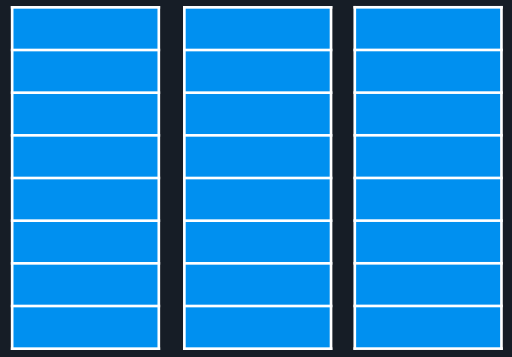
► Attributes: (14)

Examples of Cloud Optimised Data Formats

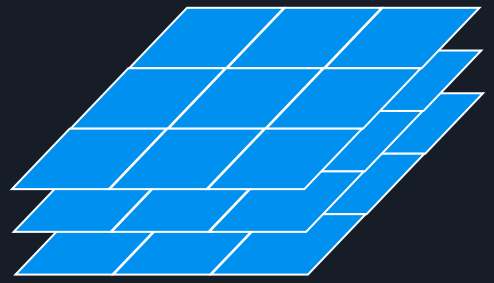
Row oriented
(Databases)



N-dimensional
Zarr



Column oriented
Parquet



Imagery
Cloud-optimised GeoTIFF