GODDARD EARTH SCIENCES

NASA

EXPLORE EARTH

YOUR HOME, OUR MISSION

Challenges and opportunities for effective storage and dissemination of Earth System model outputs

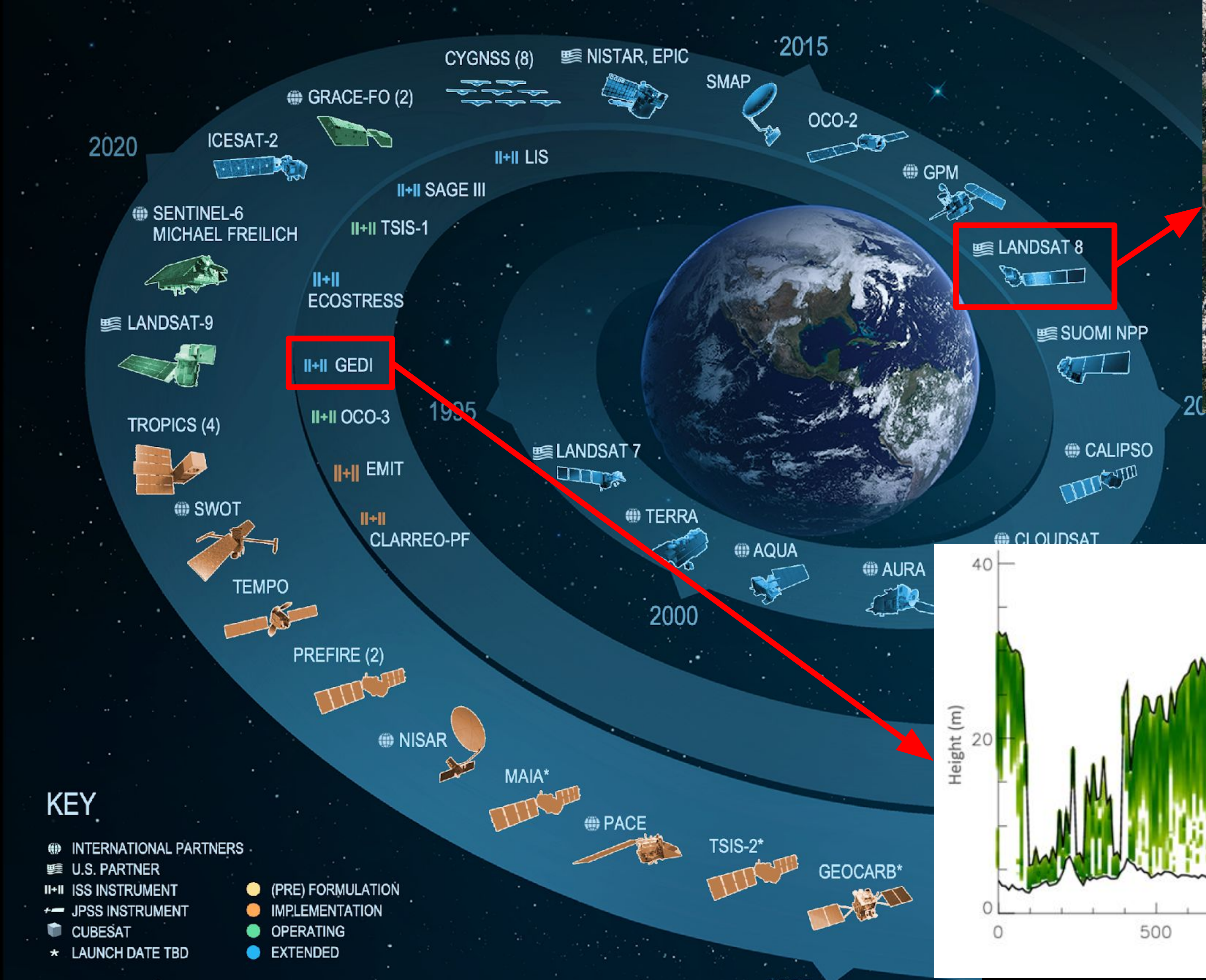
Alexey N. Shiklomanov
NASA Goddard Space Flight Center

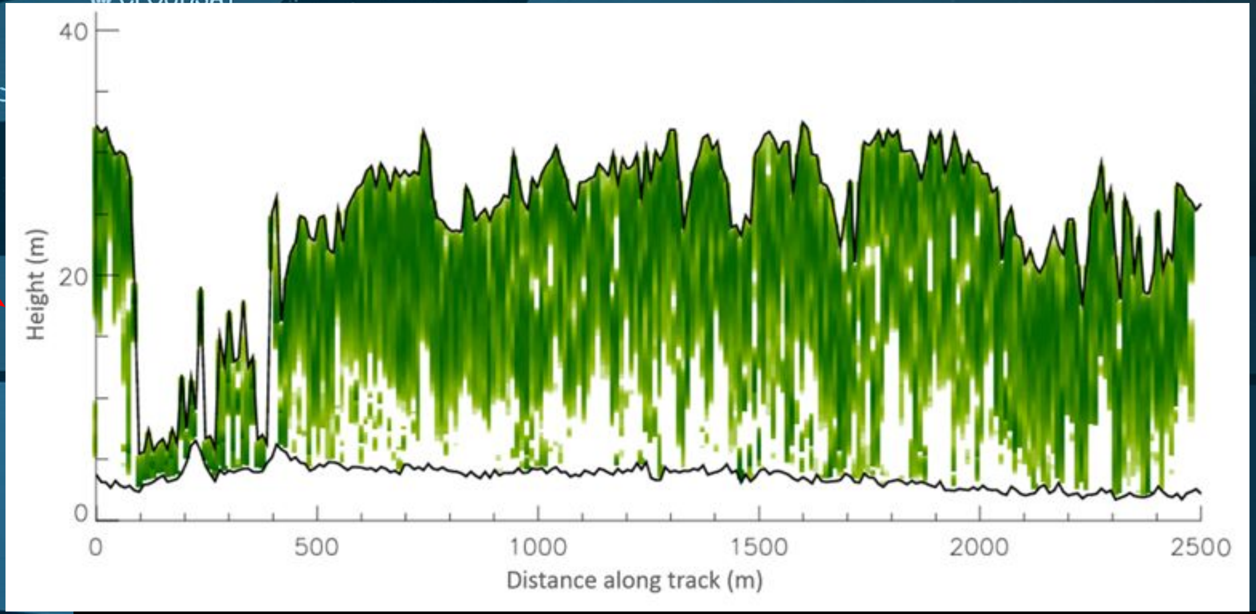
Outline

(1) NASA is moving its Earth Science data into the commercial cloud. Why? What does mean for you?

(2) What do we need to do differently with data in the cloud? What does it really mean for data to be “cloud-optimized”?

(3) How do we get more people to use Earth System model data?

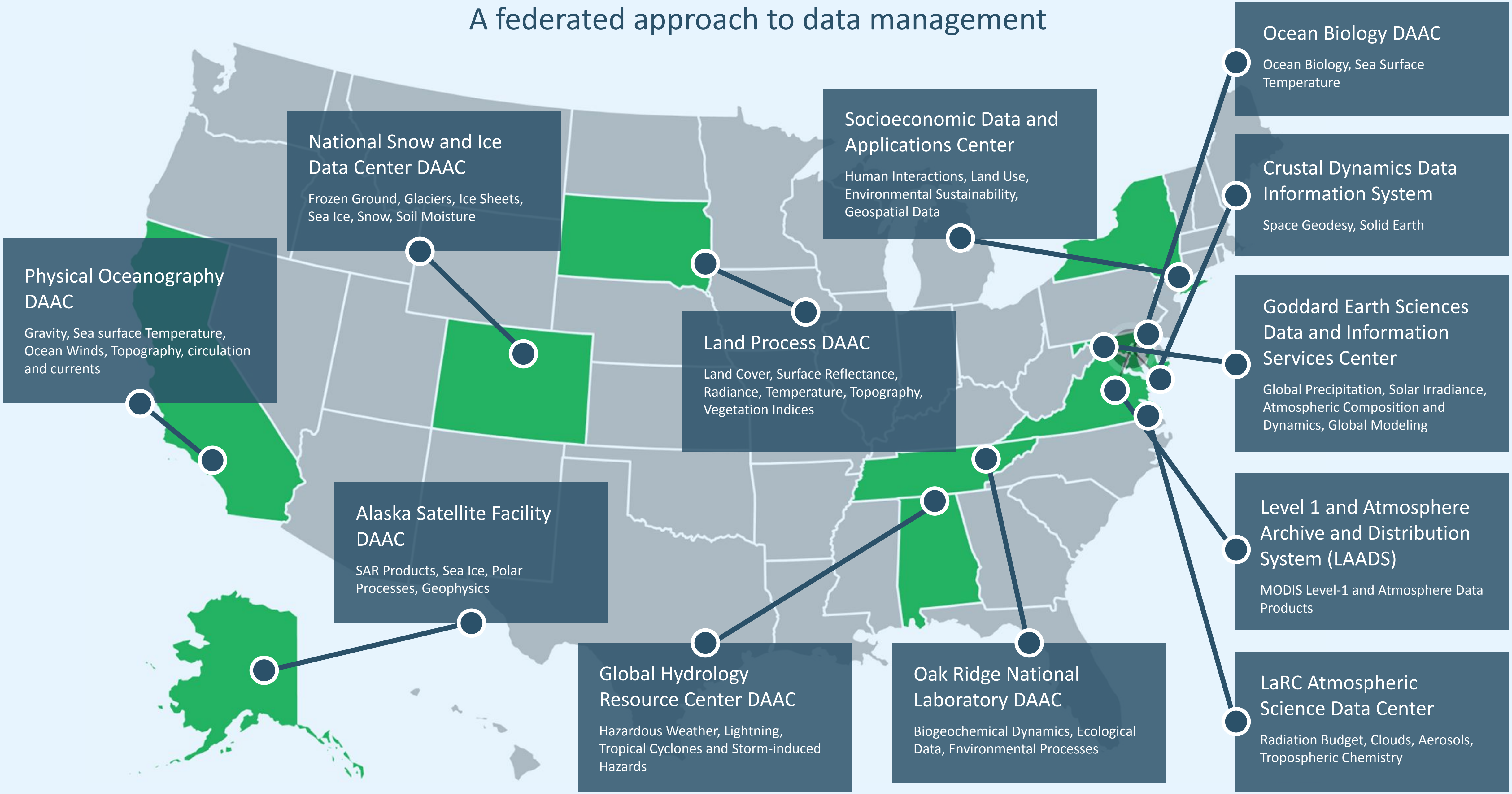




~11,000 datasets from **~1000 instruments** totaling **~62 PB of data**

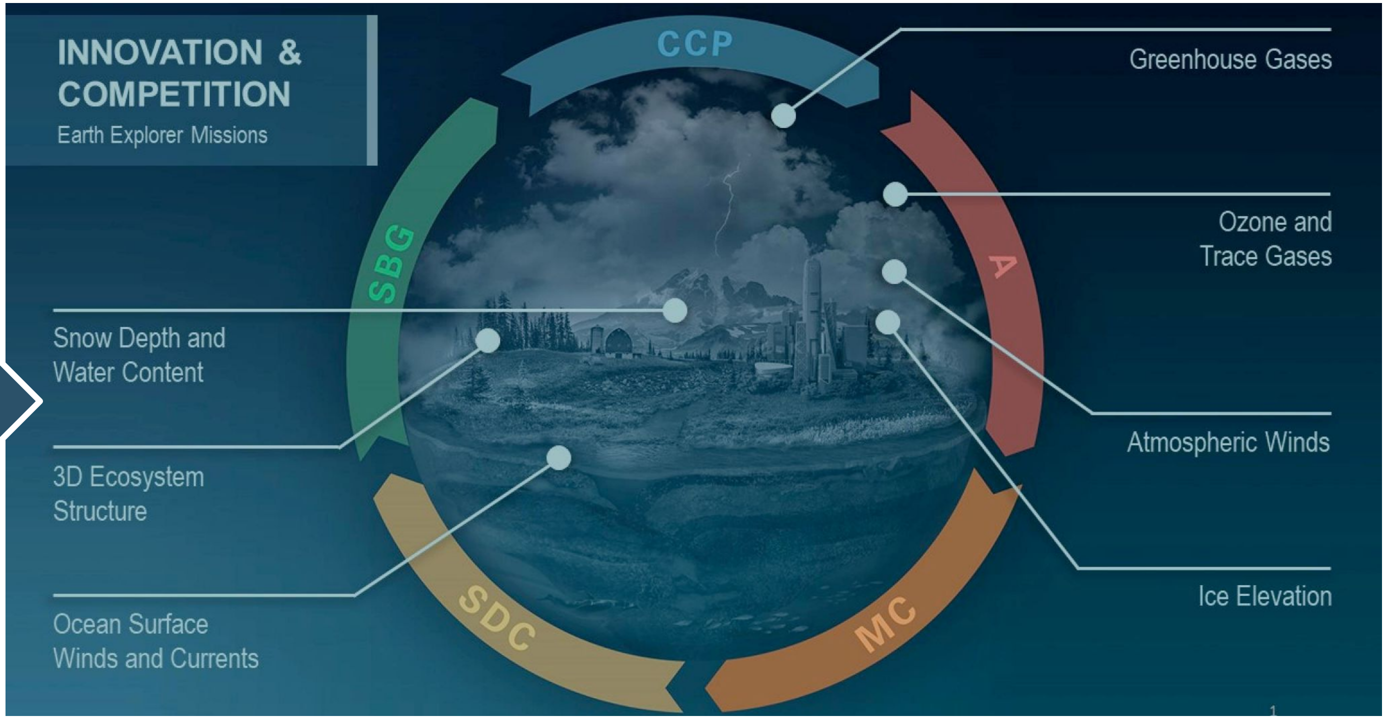
NASA's Distributed Active Archive Centers (DAACs):

A federated approach to data management

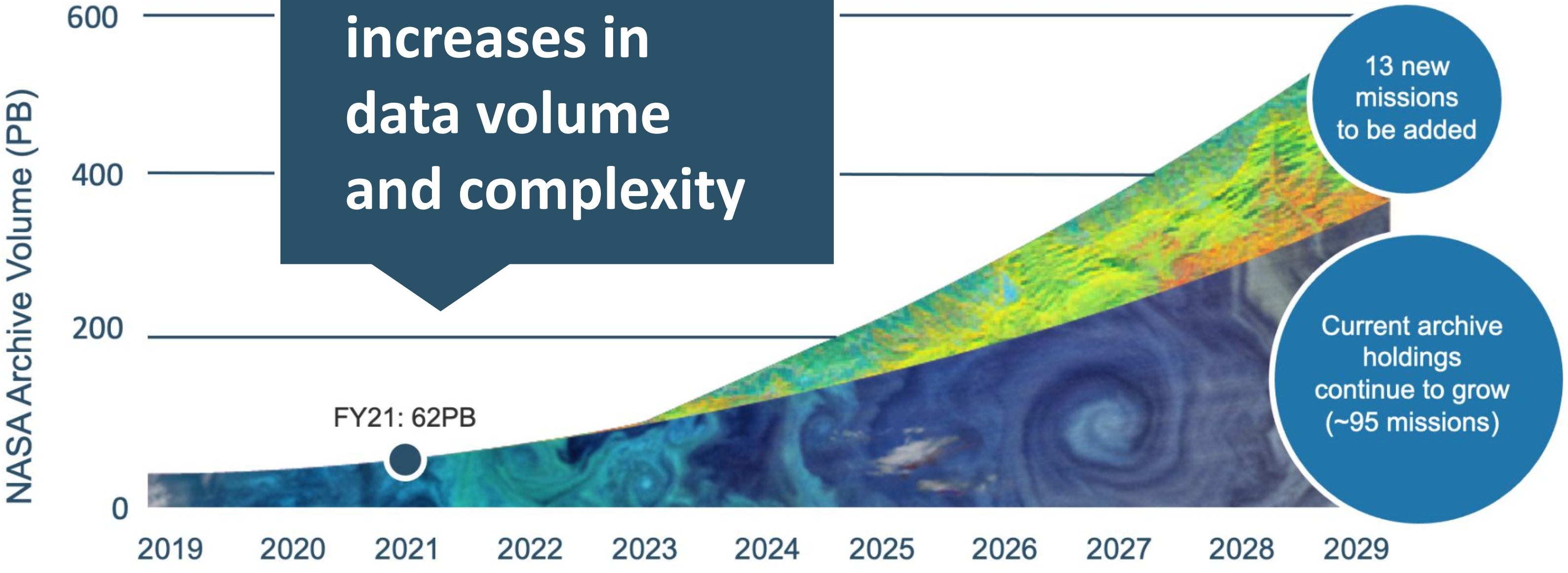


Challenges facing NASA's Earth Data ecosystem

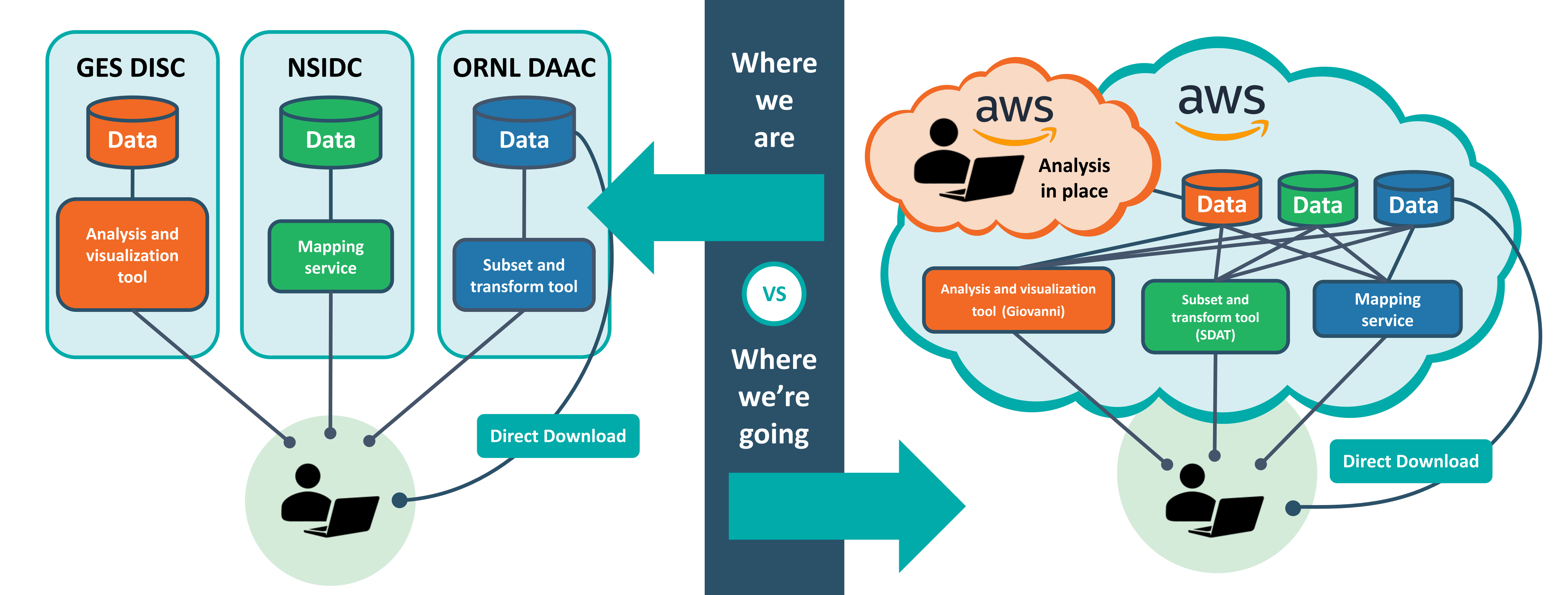
Push for transdisciplinary, multi-mission science



Dramatic increases in data volume and complexity



Increased emphasis on open, accessible, and reproducible science



What will stay the same?

- All NASA Earth Science data will continue to be 100% free and open to public.
- Existing data services (including direct download) will continue to work without disruption
- On-premise HPC will continue to play an important role in the NASA computing ecosystem

What will change?

- It will be easier for DAACs to collaborate and develop tools that work with more datasets, now that they always have direct access to each other's data.
- *New* options for analyzing data and developing tools "in place" in the cloud, without needing to download data.

Challenges

1. Many tools and workflows that rely on local hardware do not work (well) in the cloud
2. Planning and managing commercial cloud costs
3. Working across different commercial cloud providers (e.g., egress costs)
4. Integrating commercial cloud and on-prem (HPC) compute in a secure and cost-effective way
5. Diversity of NASA's Earth Science data and users makes it difficult to standardize data catalogs and tools

Amazon Web Services (AWS) regions

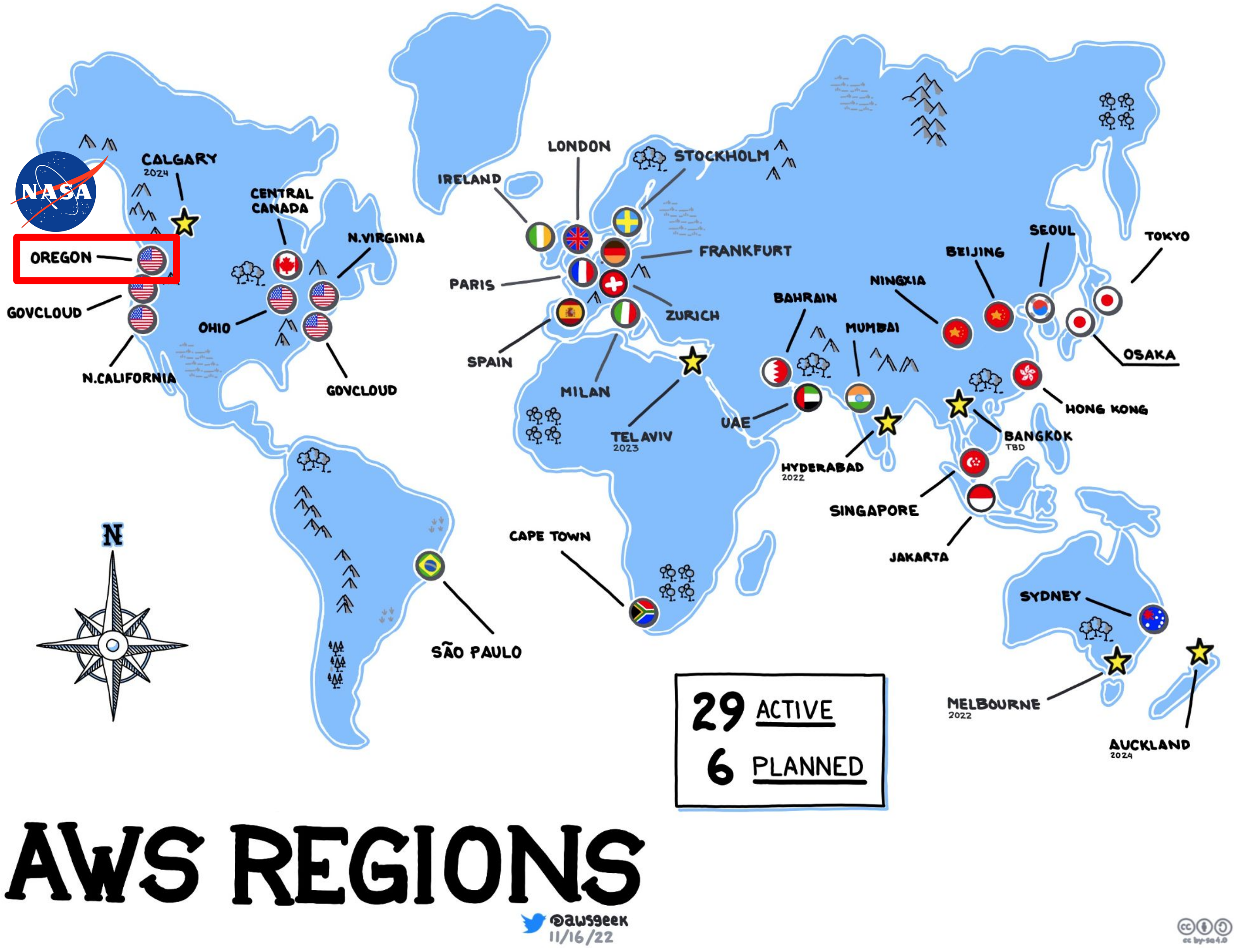
Anyone can launch an AWS service **in any region, from anywhere**, regardless of where they are physically located.

Moving and accessing data ***within*** a region is **free and fast**. Moving data ***out*** of a region is **slower and costs money**.

- <u>Requester pays</u> — Data *user* pays out of their AWS account
- <u>Provider pays</u> — Data *provider* pays out of their AWS account
- Under certain conditions (e.g., AWS Open Data Registry), AWS will pay storage + egress costs

Putting **as much compute and storage as possible in the same region** is essential to minimizing cost and maximizing performance.

NASA's Earth Science data archive is in **us-west-2** (Oregon). NOAA's data are currently in **us-east-1** (Virginia).



File system vs. network storage

Total read time [s] = Latency [s] + (Volume [MB] ÷ Bandwidth [MB/s])

"Time to first byte"

Filesystem storage (HDD, SSD)

Latency: Low (good)

- <0.1 ms for SSD; 1-10 ms for HDD

Bandwidth: Slow (bad)

- <100 MB/s for HDD; 200-500 MB/s for typical SSD; 5000-7000 MB/s for top-line SSD

Prefer **small chunk sizes** that can be read using **many small read operations.**

Uses **file system calls** built into operating system (and every programming language).

Network-based object storage (e.g., S3)

Latency: High (bad)

- 100-200 ms; each API call costs (a little) money ($0.0004/1000 requests)

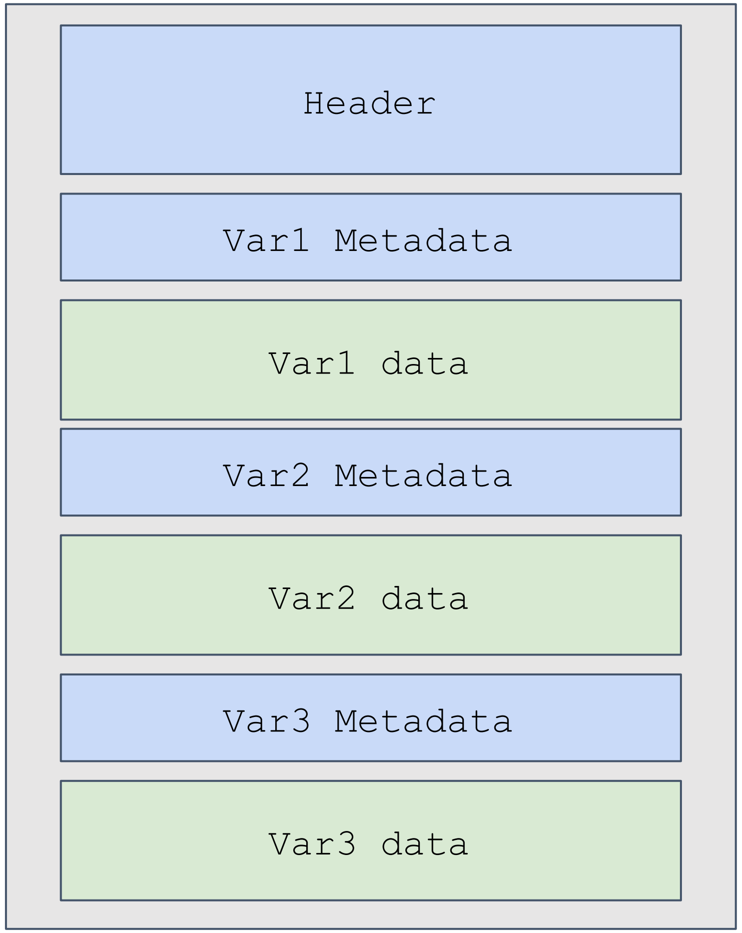
Bandwidth: Fast (good)

- Up to 100,000 MB/s (within-region; out of region, depends on distance and internet bandwidth)

Prefer **larger chunk sizes** that **minimize the number of requests**. "Cloud-optimized" data formats are ones that **minimize the number of API calls** required to read a dataset through a combination of **consolidated, predictably sized metadata** headers and **well-thought-out chunking strategy.**

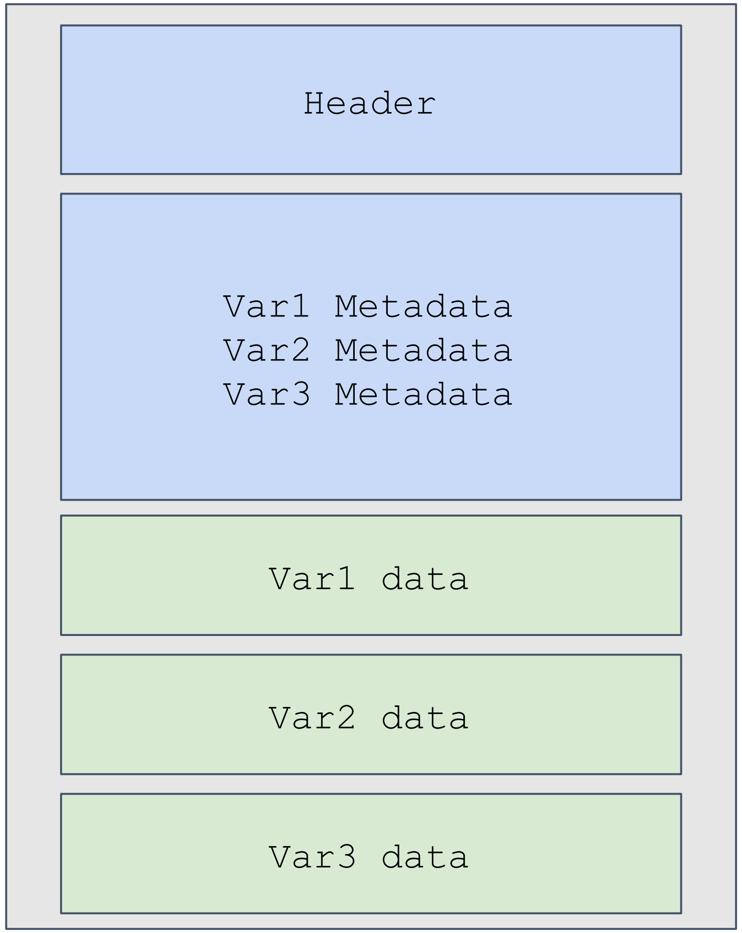
Uses **HTTP requests** (e.g., curl); requires **special libraries or programming language abstractions** (e.g., Python fsspec)

“Cloud-optimized” metadata



Bad

1 API call to open the file + **2 API calls per variable** (1 to read the metadata, 1 to read the data) to read the data.



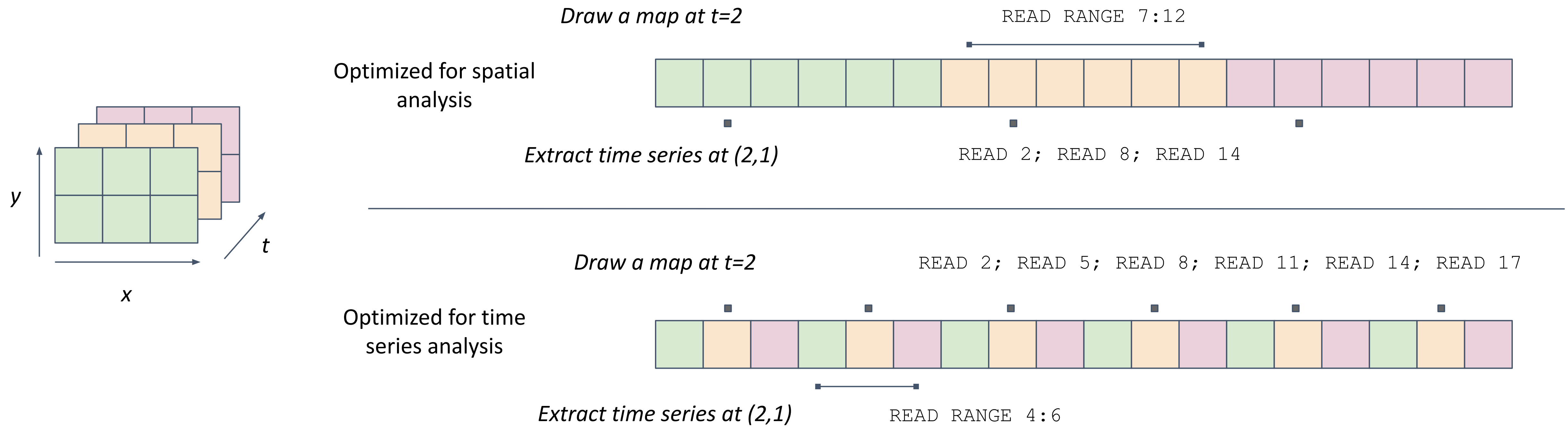
Good

1 API call to open the file and get metadata for all variables. Then, **only 1 API call per variable** to read the data.

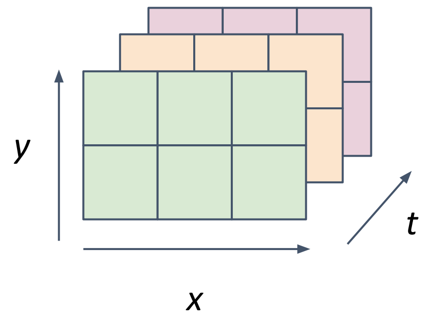
Array storage

Sequential reads are *much* faster than random reads.

For multi-dimensional datasets, there is a fundamental performance tradeoff in array storage for slicing along different dimensions.



Chunk size and shape



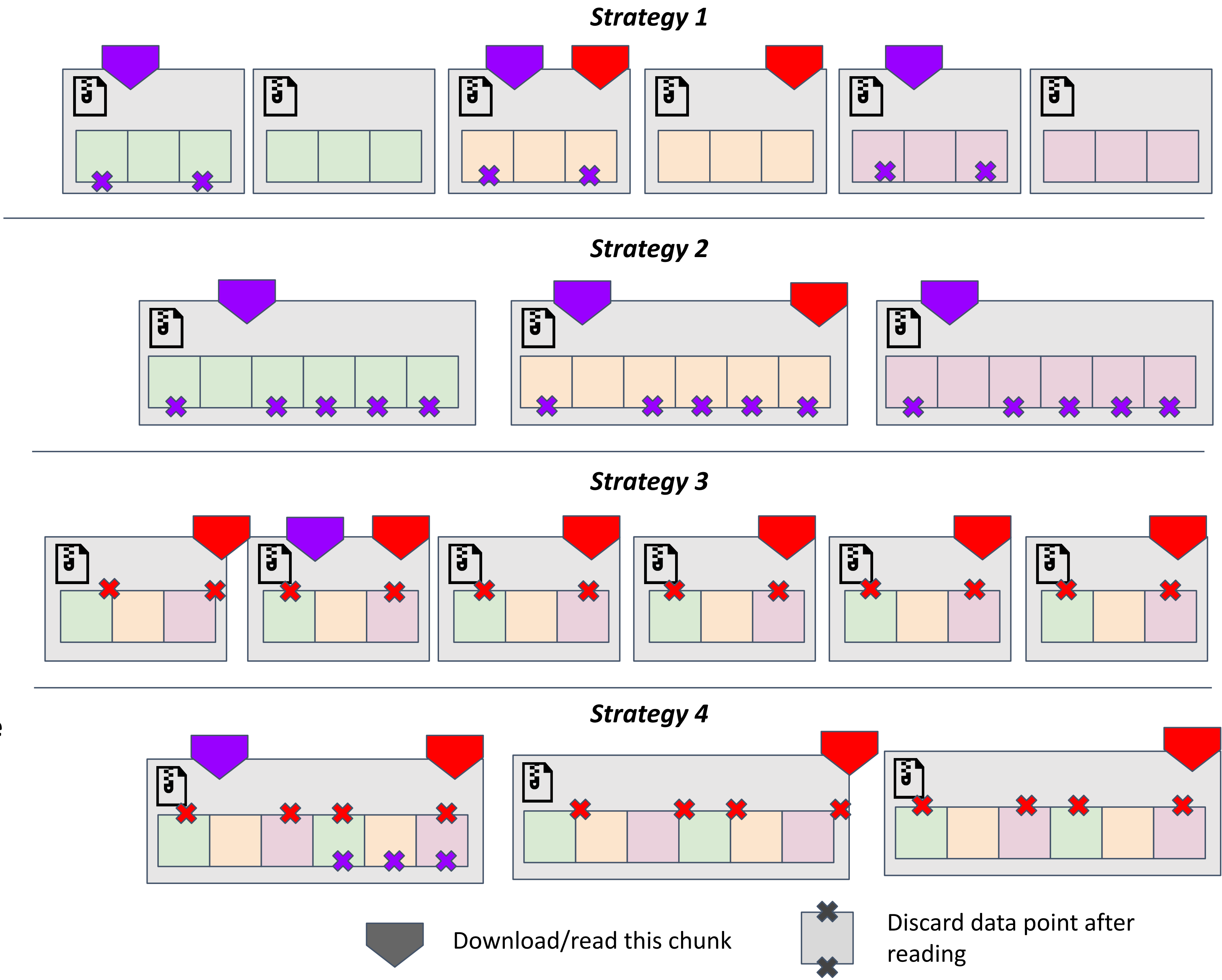
Draw a map at t=2

Extract time series at (2,1)

In binary formats like NetCDF-4 (HDF5) and Zarr, chunks are **compressed**, so it is **impossible to read part of a chunk**.

For a given use case, we want to **minimize the number of API calls** and **minimize the read volume of unused data**.

Again, there is a **fundamental trade-off** in chunking strategies for different access patterns.



Geospatial data models

Unstructured ⟷ Structured

Vector-based / tabular

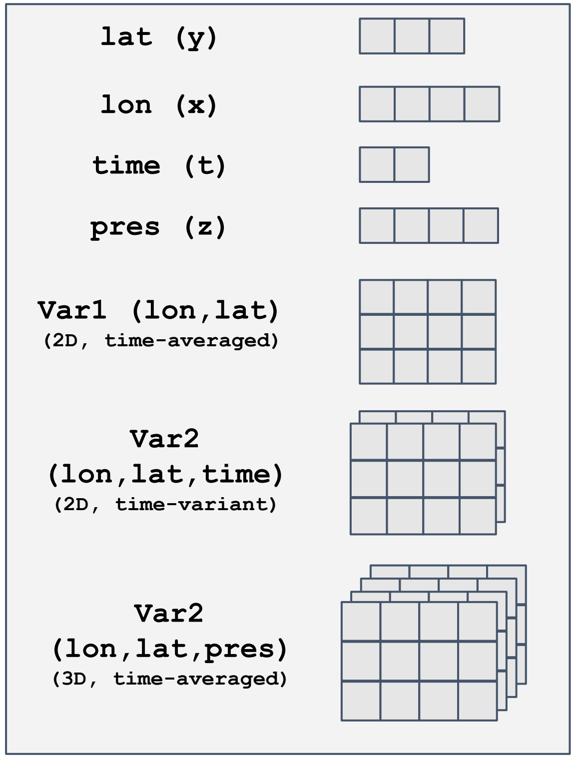
X	Y	T	Var1	Var2
1	1	1	…	…
1	2	1	…	…
2	1	1	…	…
2	2	1	…	…
1	1	2	…	…
1	2	2	…	…
2	1	2	…	…
2	2	2	…	…

Examples: Row-based plain-text (e.g., CSV); columnar binary (e.g., Parquet, Feather, FST); relational database (SQL); point tile

Maximum flexibility, but minimal efficiency for storage and access.

Subsetting requires exhaustive search along all dimensions.

Multi-dimensional, coordinate-based

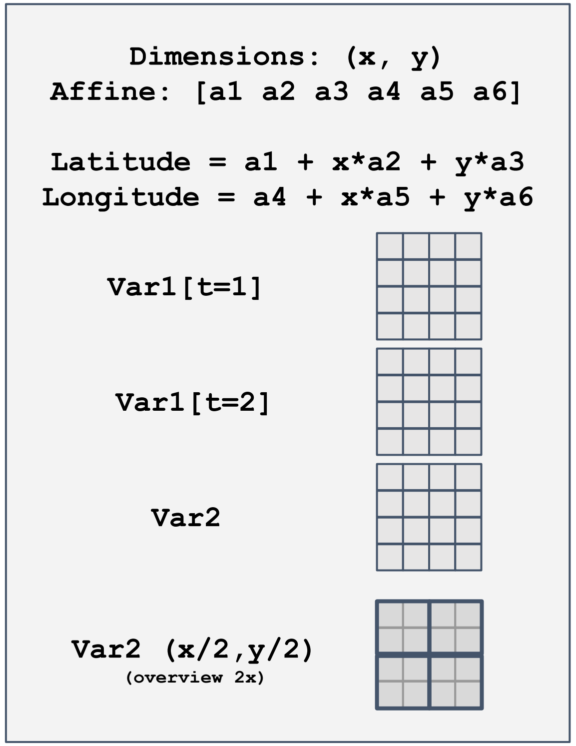


Examples: NetCDF, Zarr, GRIB(2)

Sacrifice some flexibility for some efficiency.

Subsetting requires exhaustive search, but only once along *each* dimension.

Raster



Examples: GeoTiff, JPEG

Least flexibility, but most efficiency.

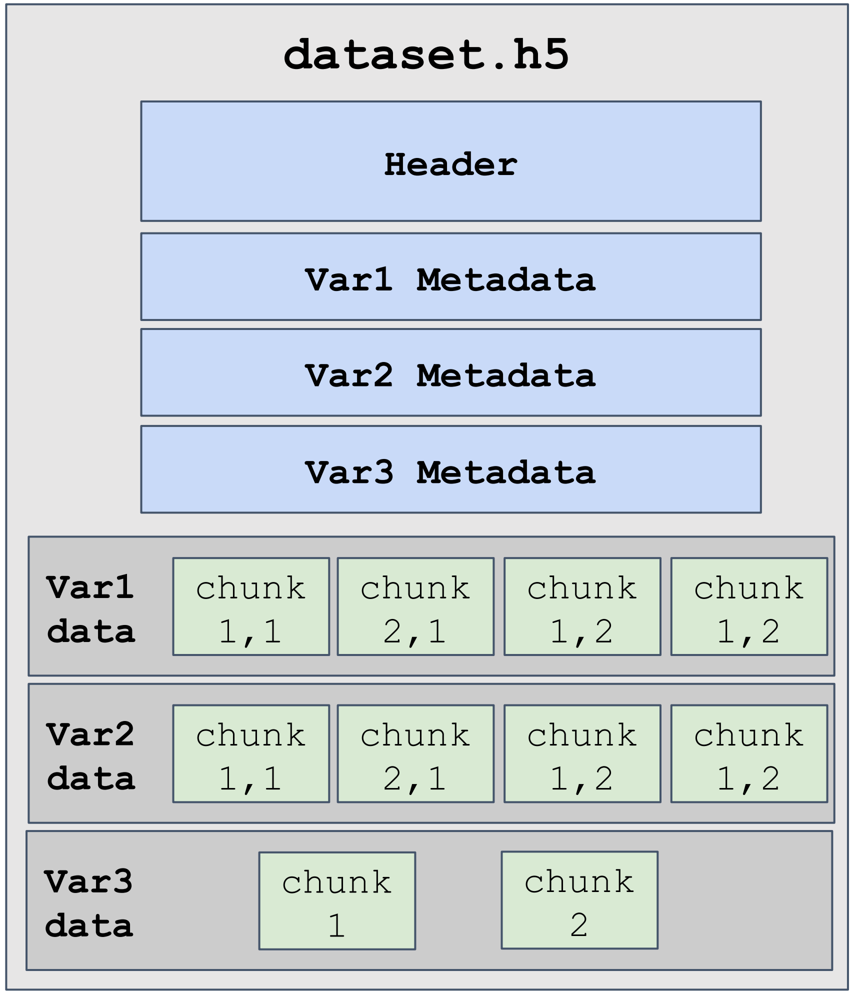
Subsetting can be done analytically using affine transform.

Overviews (data stored at different zoom levels) are very useful for dynamic visualization.

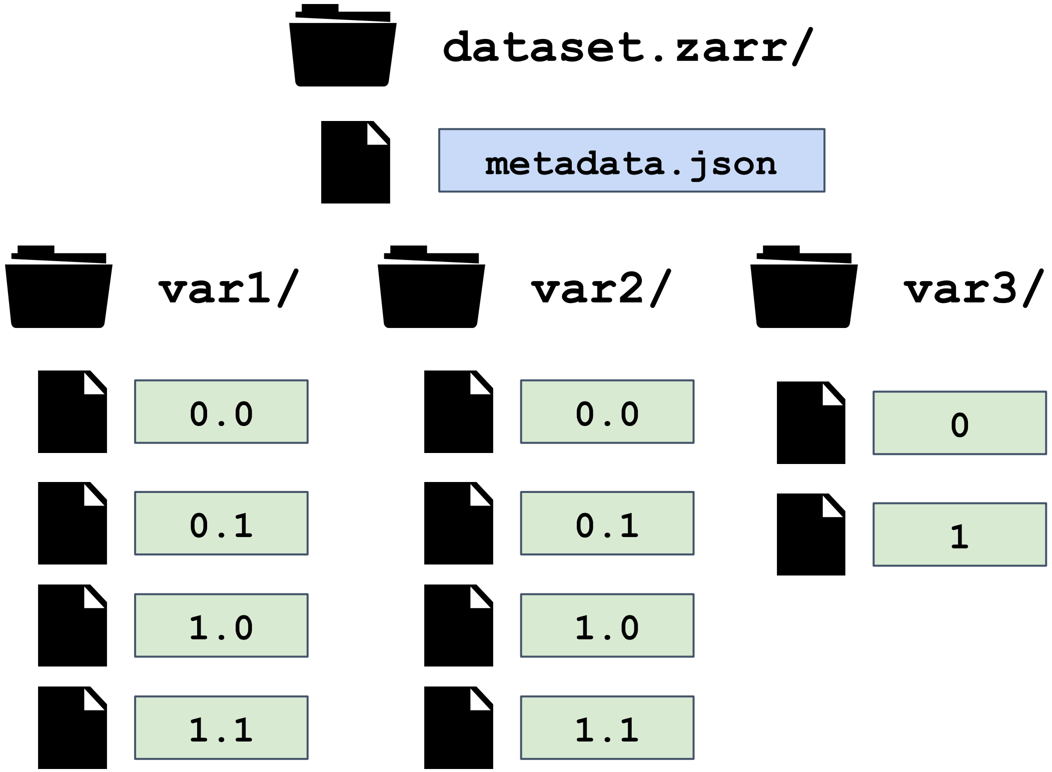
HDF5 vs. Zarr

Both formats fill the same niche: Binary, compressed, chunked storage of multiple arrays of arbitrary dimensionality.

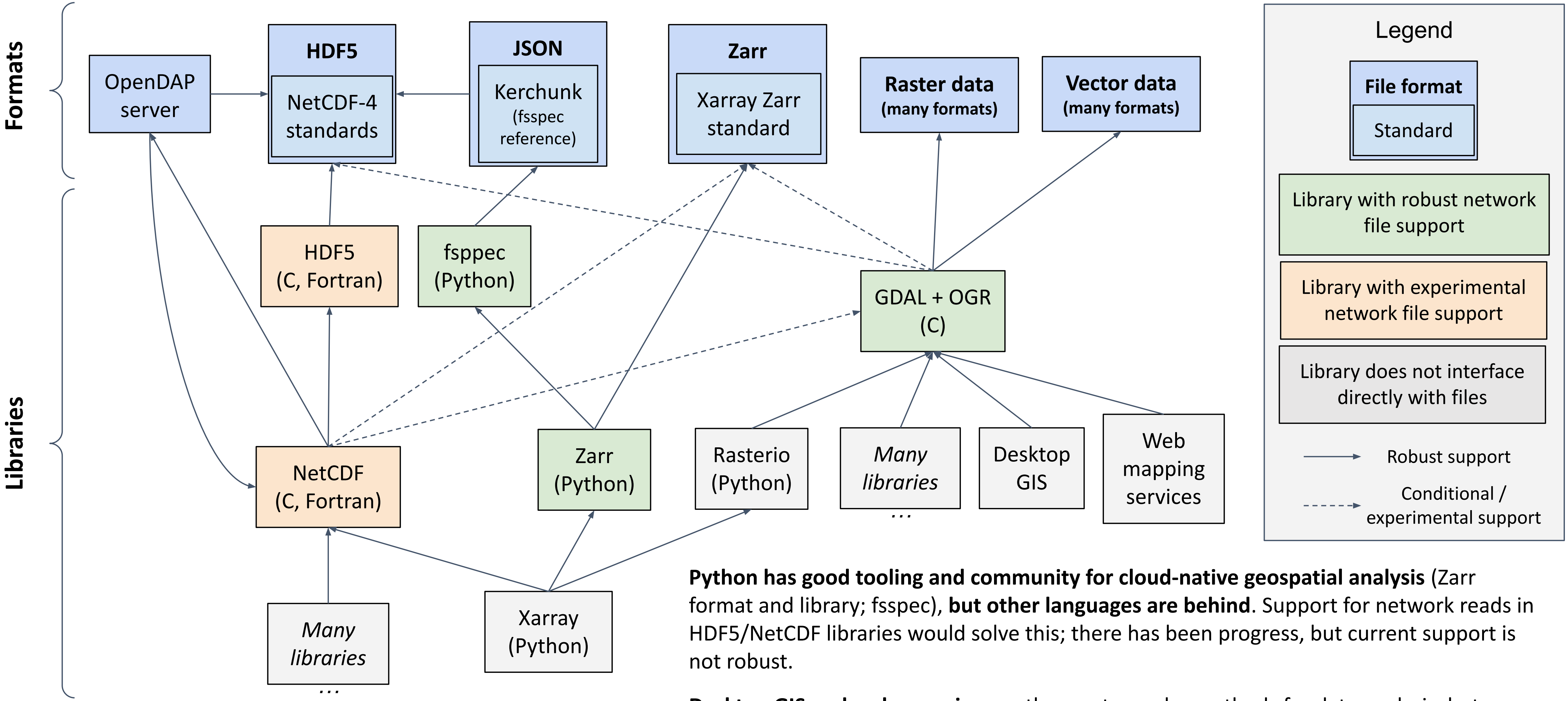
In HDF5, metadata and chunks are all stored within a single file.



In Zarr, metadata and chunks are stored as separate files.



Common geospatial data formats and libraries



Python has good tooling and community for cloud-native geospatial analysis (Zarr format and library; fsspec), **but other languages are behind**. Support for network reads in HDF5/NetCDF libraries would solve this; there has been progress, but current support is not robust.

Desktop GIS and web mapping are the most popular methods for data analysis, but **fragmentation between multi-dimensional (NetCDF/Zarr) and geospatial (GDAL/OGR)** standards, tools, and communities inhibits usage of model data in these tools.

Takeaways

(1) NASA, and other agencies and organizations, are moving their data into the commercial cloud.
   (a) Pros: Data centralization; greater interoperability between tools; analysis in place
   (b) Cons: Opaque cost model; need to adjust computing paradigms and workflows; dealing with egress

(2) Cloud-optimization of data generally means minimizing the number of API calls required to read a dataset, through a combination of consolidated metadata headers and well-thought-out chunking strategy.

(3) Getting Earth System model data into desktop GIS and web mapping tools is a great way to broaden their usage, but this is difficult due to differences in the underlying data models and fragmentation between modeling and geospatial communities.