



# A Python Roadmap

for Accessing and Interfacing  
with Big Environmental Data in  
Cloud Storage Providers

**Ryan Paul Lafler**

**Premier Analytics Consulting, LLC**



**Premier Analytics  
Consulting, LLC**

**Copyright © 2025 by Ryan Paul Lafler &  
Premier Analytics Consulting, LLC.  
All Rights Reserved.**

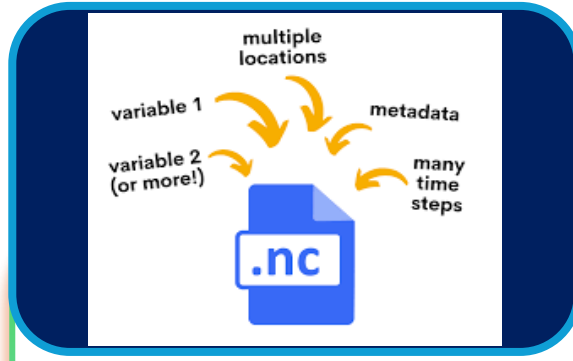
No part of this presentation may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the copyright holder, except in the case of brief quotations or references for educational purposes.

[www.Premier-Analytics.com](http://www.Premier-Analytics.com)

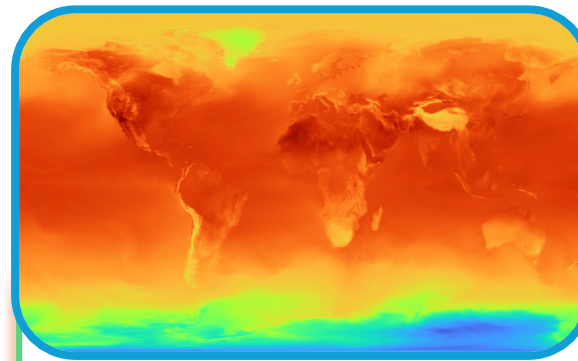
# Objectives and Main Concepts



**Understand  
Properties of  
Optimized  
Cloud  
Formats for  
Big Data**



**Investigate  
Metadata for  
Spatial &  
Temporal File  
Formats**



**Connect to  
CMIP6 Big  
Data in Google  
Cloud Storage  
with Python**



**Python  
Frameworks,  
Libraries,  
and Data  
Pipelines**





# 1. Optimized Cloud Formats for Big Climate Datasets

# Climate Data Fundamentals

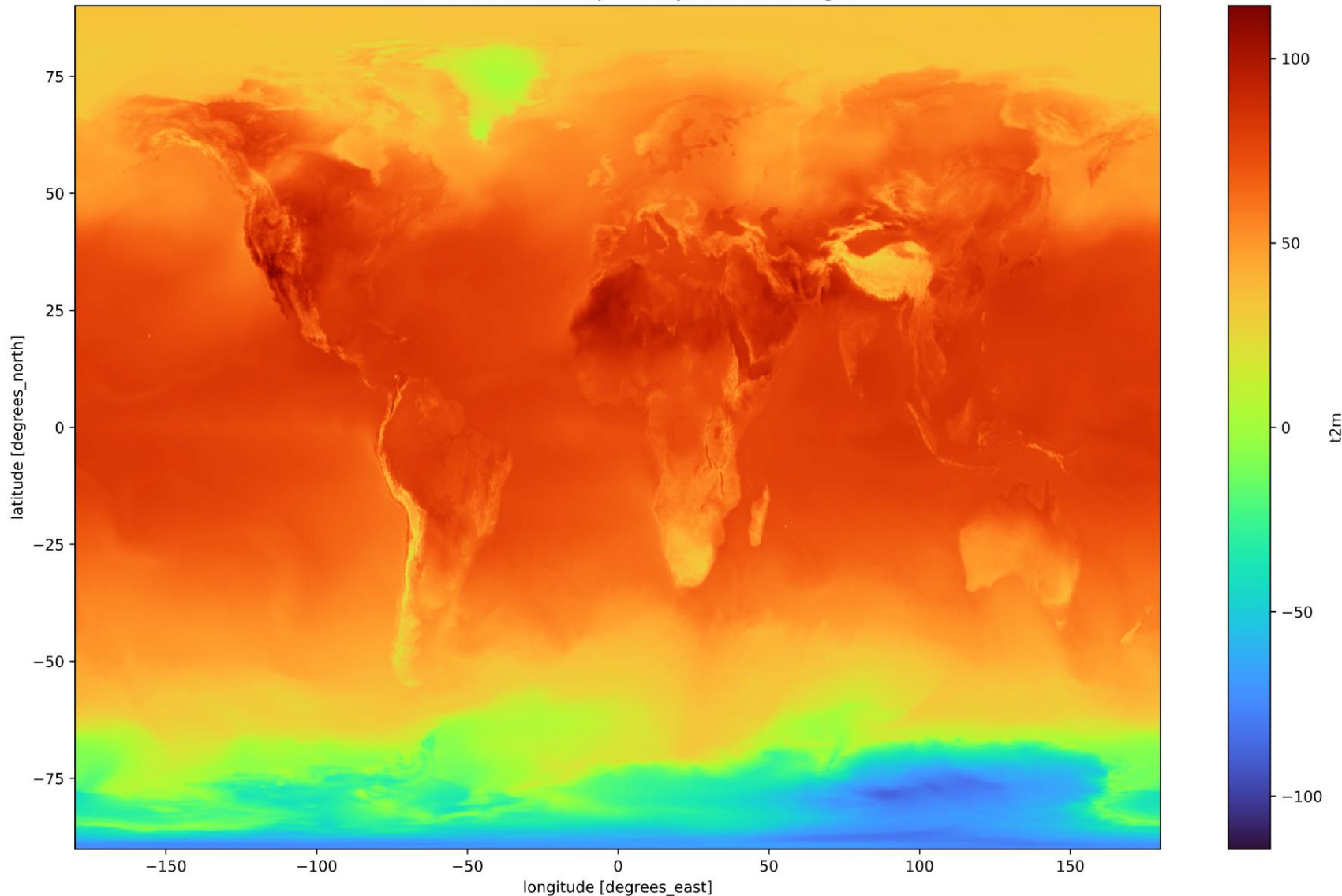
- Climate data divided into 2-major domains: **gridded** and **discrete** datasets
  - Variables recorded over a continuous extent are defined as **gridded datasets**
  - Variables recorded at specific locations or regions (polygons) are **discrete datasets**
- **Gridded climate data** represented by arrays made up of individual pixels:
  1. **Spatial resolution** defines how much detail a pixel captures (i.e., 100 m x 100 m)
  2. **Coordinate system** defines data's spatial extent (i.e., lon & lat)
  3. **Pixels** denote a values for the climate variable over its spatial extent

# Climate Data Fundamentals

- **Discrete climate data** represented by flat-file data structures for locations & areas:
  1. **Mixtures of defined geometries** (i.e., points, polygons, lines)
  2. **Coordinate system** defines exact locations of geometries

# ECMWF 2-Meter Global Temperature Gridded Climate Data

time = 2024-07-19, step = 8 days 00:00:00, heig...

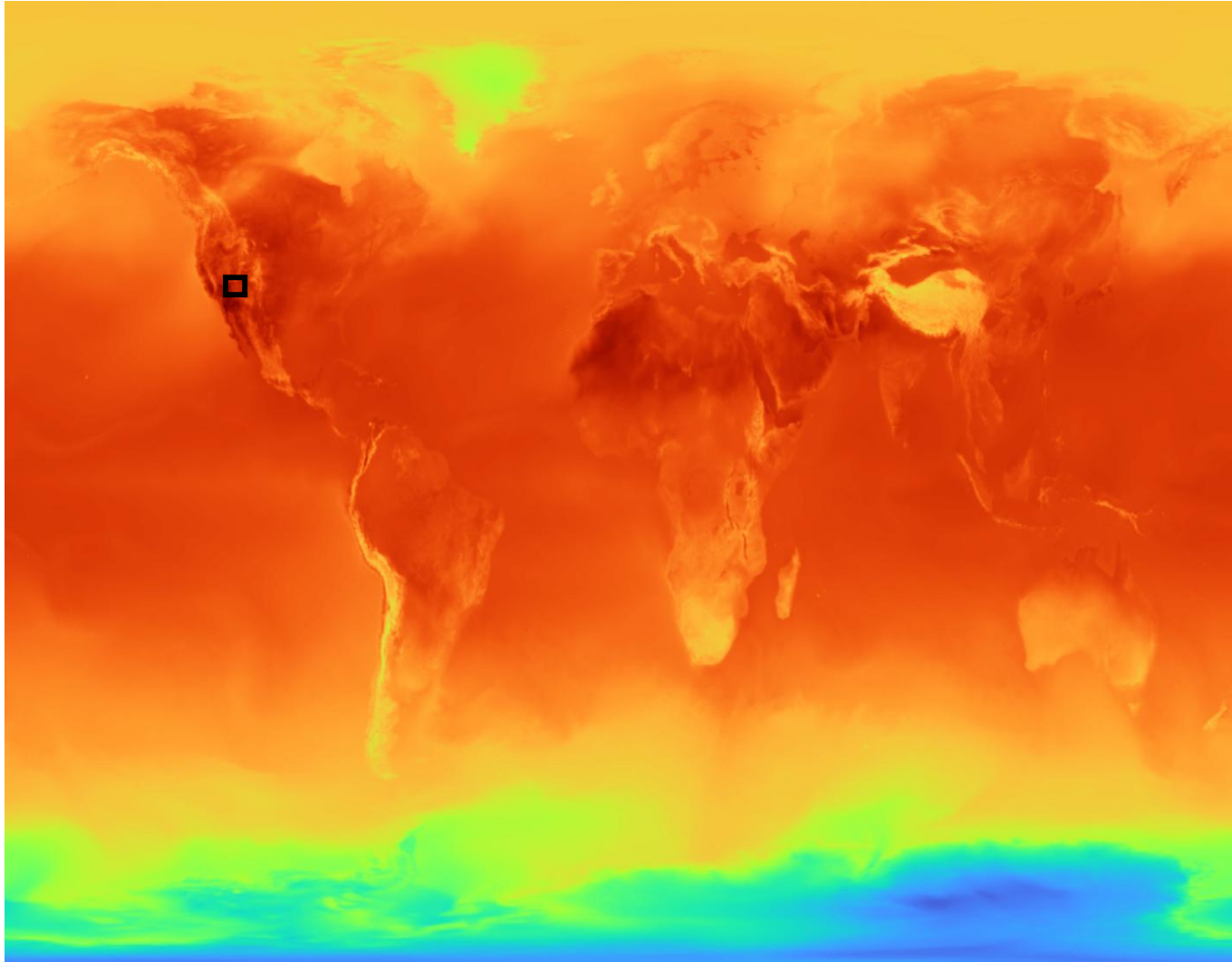


Continuous climate dataset spanning the entire Earth's extent measuring temperature (in Fahrenheit) 2 meters above the surface.

Forecasted data comes from the European Center for Medium-Range Weather Forecasts (ECMWF).



# Examining a Particular Region of Gridded Data



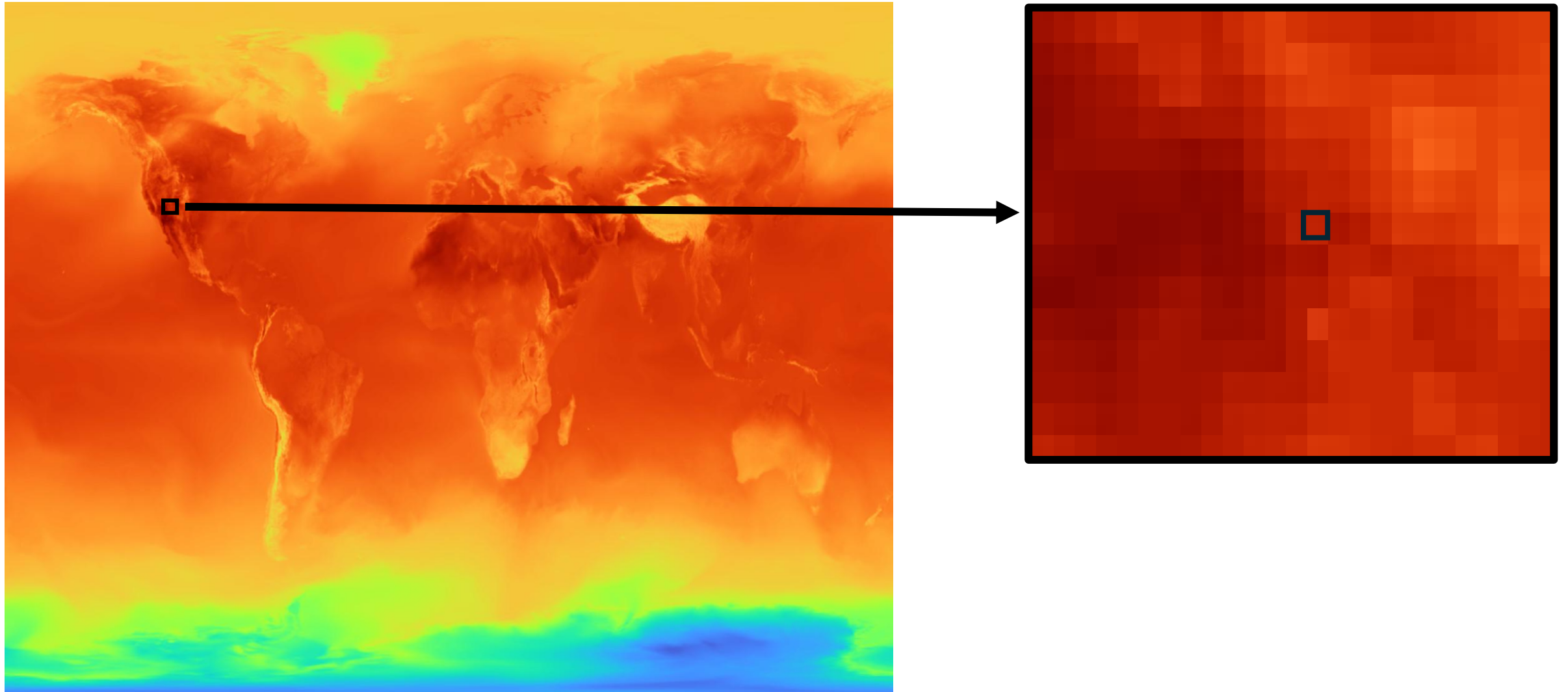
Gridded climate data are equivalent to images with defined coordinate systems.

Images contain pixels; gridded climate data contain pixels representing climate values.

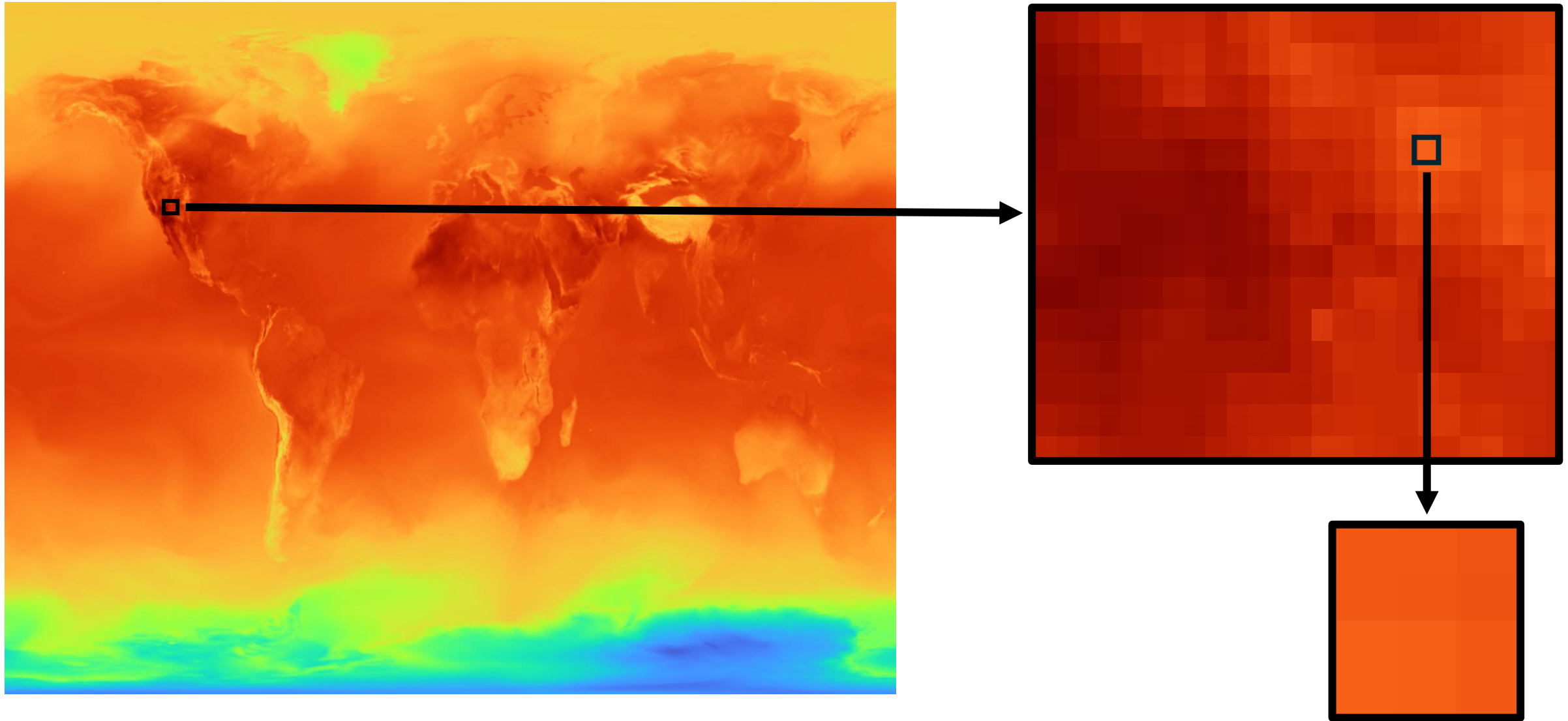
Zooming in on gridded data will make the pixels appear *blocky* → going beyond the spatial resolution of the dataset.



## Zooming Into a Specific Region of Gridded Data

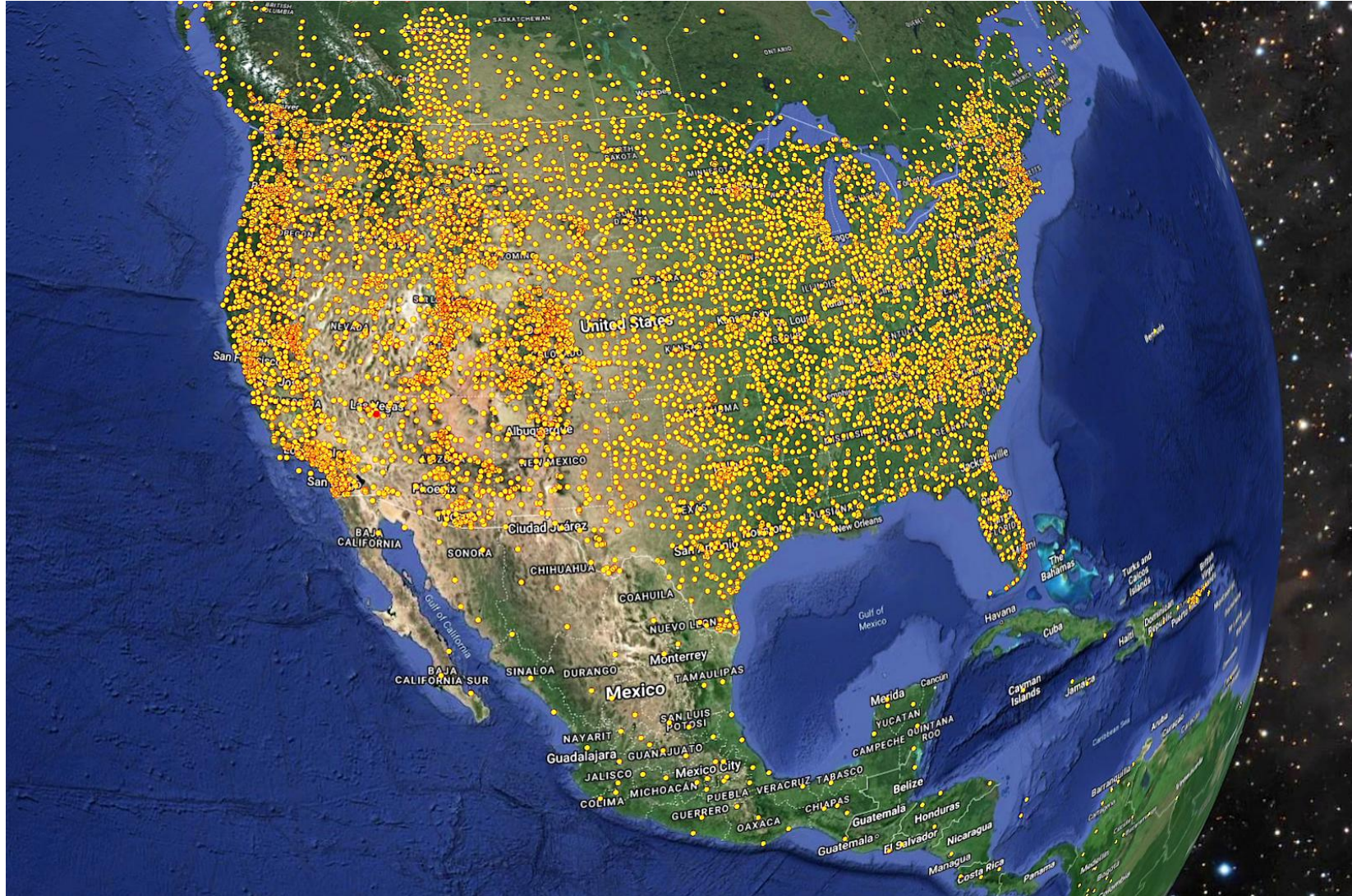


# Extracting a Single Pixel (Temperature Value) from Gridded Data





# NOAA-GHCN Monitoring Stations Discrete Climate Data



NOAA Global Historical Climatology Network (GHCN) uses fixed monitoring stations measuring hourly and daily climatology values at their locations.

Gives precise measurements at locations, requires interpolation to measure in-between distances.

# What Makes Climate Data Unique?

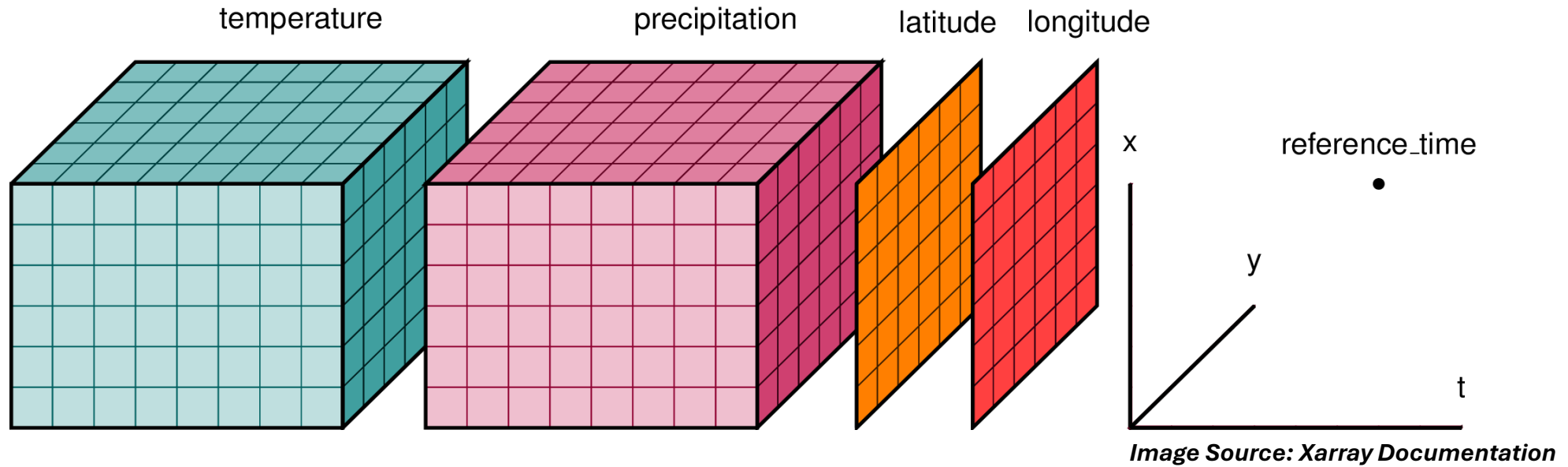
- Climate data adds an additional dimension to spatial data: **time**
- **Spatiotemporal data** at the minimum contains **3-dimensions** (longitude, latitude, time)
- This transforms gridded datasets into **multi-dimensional arrays** where pixels (climate variable values) are measured at different time intervals
- **Temporal resolution** is like **spatial resolution**, but with respect to time → it denotes the frequency that climate values are measured / recorded / forecasted
  - **High temporal resolution** data are measured every 30-min, 1-hour, 6-hours, or daily
  - **Low temporal resolution** data are measured every week, month, or year
  - A higher temporal resolution provides finer details about how climate values **change over time** → more information leads to better insights



# How Large Can Gridded Climate Data Grow?

- Climate datasets measure many different variables in some data structure
- Gridded climate datasets often **share consistent spatial and temporal dimensions** when measuring different **variables**
  - Sharing dimensions keeps the file's metadata cohesive & variables measured along consistent scales (dimensions)
- Example: NOAA'S Real-Time Mesoscale Analysis (RTMA) data
  - Provides **hourly forecasted** data at **2.5 km x 2.5 km** spatial resolution
  - Extent covers the **continental United States (CONUS)** region with AK\*, HI\*, & PR\*
  - Measures hourly climate attributes such as: **surface temp, wind speed, zonal (U) and meridional (V) wind components, surface pressure, dew point temp, cloud cover, precipitation, visibility, and relative humidity**

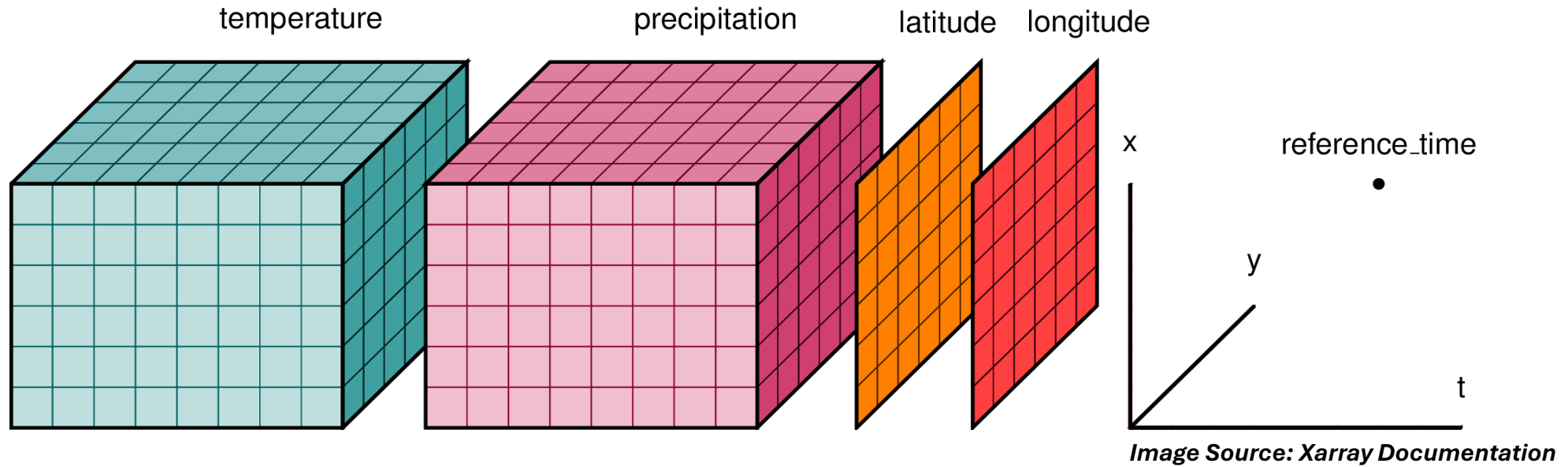
# Visualizing Multi-Dimensional Gridded Climate Data



## 3D Gridded Data Structure

- Gridded data organized as a 3D array with dimensions (**longitude, latitude, time**)
- Each climate variable stored inside of its own 3D array, sharing consistent **spatial** (longitude & latitude) and **temporal** (time) resolutions
- Consistent dimensions makes subsetting, merging, and concatenating arrays efficient

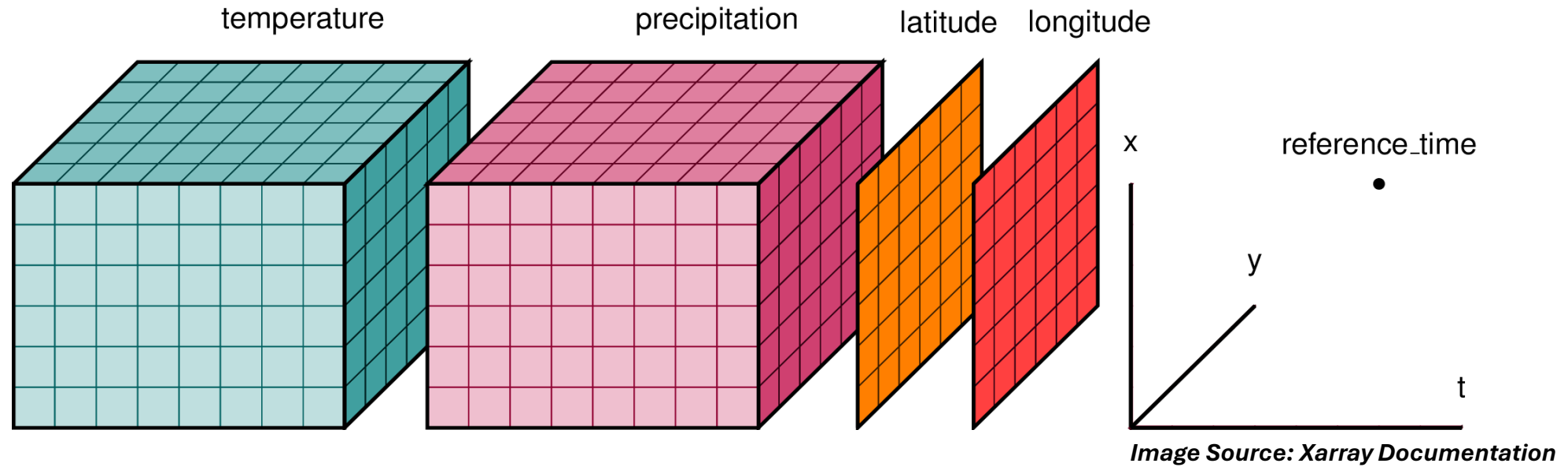
# Visualizing Multi-Dimensional Gridded Climate Data



## 2D Spatial Grids at Each Time (Slice) Step

- Time (slices) steps are represented by 2D arrays indexed by **(longitude, latitude)** dimensions
- Each 2D array represents a step in time that is consistent for all climate variables
- Uniform grid cells and time steps allow efficient filtering, indexing, and array subsetting to occur across all time step arrays

# Visualizing Multi-Dimensional Gridded Climate Data



## Extracting Time Series for a Specific Location

- Assuming spatial dimensions are consistent across all variables, each pixel represents the same unique location across 2D time slices at some spatial resolution (i.e., 2.5 km x 2.5 km)
- Nearest neighbor lookups used to index desired pixel (location) and retrieve the time series for that pixel across all 2D time slices



# How is Gridded Data Organized into Cohesive Data Structures?

- Different philosophies for storing, indexing, and managing climate data
- Usage depends on application → is focus on real-time data streaming? Saving storage space? Fast indexing for historical lookups? Programming language interoperability? Simplicity?
- Climate data can be organized into different structures:
  - **Flat-files (CSV, tab-delimited, spreadsheet):** Longitude denoted by columns and latitude by rows that is inefficient to handle complex data (simple to import, however)
  - **Multidimensional Cube (Explicit Arrays):** Organizes data into explicit arrays sharing *consistent dimensions* (longitude, latitude, time, pressure level, etc.) that can be easily sliced, filtered, and indexed
  - **Sequential (Implicit) Arrays:** Relies heavily on metadata to *reconstruct* relationships between layers of data → linear stack of layers stored sequentially in data structure

# Flat-Files for Climate Data

- Flat-files are easy to access, readable on any device, and interoperable with any language
- Information and data are readable by humans; no standards for data quality:
  - Longitude and latitude grids may become irregular → problems indexing & filtering
  - No coordinate reference system attached to the dataset (lack of metadata)
  - No built-in compression schemes to reduce the footprint (file size) of datasets
  - Rudimentary and slow indexing; not scalable to multidimensional climate data
  - Lack of support for metadata & data standardization protocols
  - Information redundancy leads to larger-than-necessary file sizes
- Flat-files are inefficient vectors for storing and processing big climate data

# Cube Data (Explicit Array) Structures for Climate Data

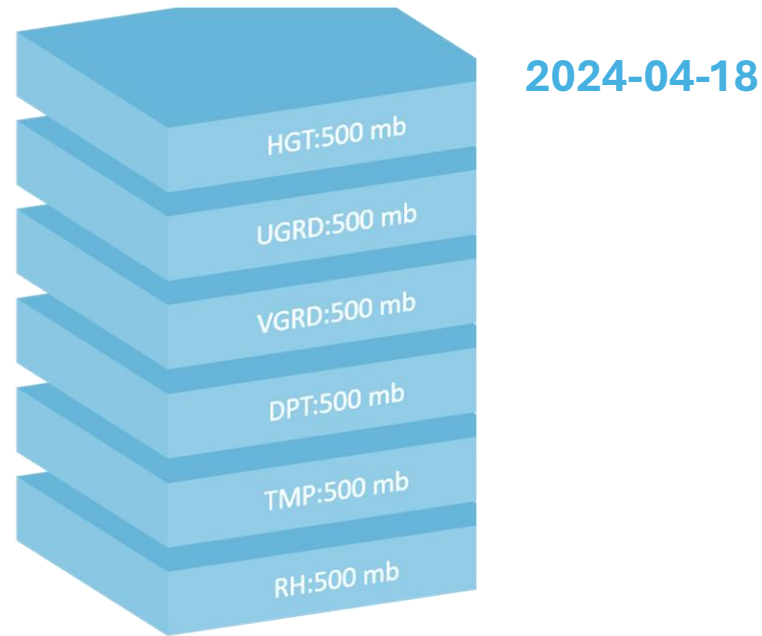
- Cube data structures organize climate data into explicit arrays by their dimensions
- Cube data requires specialized libraries & languages to interface with data inside of cubes:
  - Enables efficient access of big gridded datasets using explicit calls to arrays
  - Automatic indexing allows for quicker look-ups and subsetting than flat-files
  - Comes with metadata & data standardization protocols; reduces data redundancy
  - Built-in compression schemes to reduce larger file sizes & save space in storage
  - Supported across programming languages → interoperable data format
  - Inefficient for streaming real-time data or forecasted data
- Cube formats embrace **random access** which allows user to query from any array in the cube without having to read entire file in *sequentially* → can query data anywhere in the cube

# Sequential (Implicit) Array Structures for Climate Data

- Optimal for real-time (or forecasted) data streaming → new data arrays added sequentially
  - Arrays (called **records**) are independent of each other
    - Relationships between dimensions must be re-constructed from the metadata
  - Metadata is encoded in the **header** of each record → records are identifiable by header
  - Offers significant compression compared to data cubes and flat files
  - Keeping arrays independent (isolated) allows for quick access to updated forecasts
    - *However*, this means multidimensional queries will be computationally expensive
      - Must read all records sequentially and re-construct time and space dimensions between records to generate time series or compare records to one another
- Optimal when extracting *one* climate variable, or time step, or pressure level at-a-time



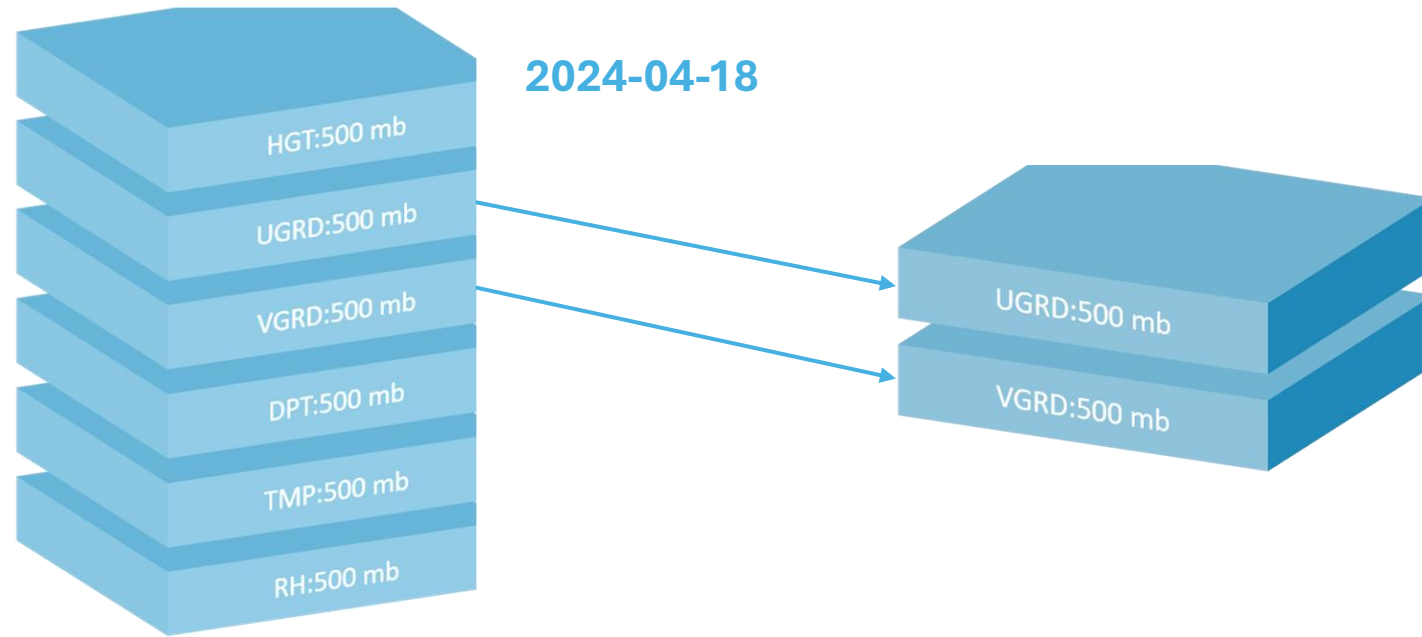
# Visualizing Sequential Climate Data Structures



## Accessing Sequential Climate Data Structures

- Shows stack of daily climate variable layers at air pressure level (500 mb) for April 18, 2024
- Each climate variable is stored as its own isolated array of data, with headers (metadata) describing the variable for each layer
- Efficient for querying climate variable layers given specific air pressure level & time slice

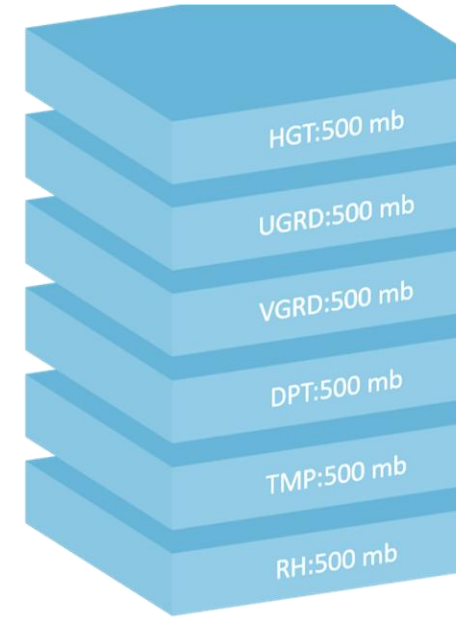
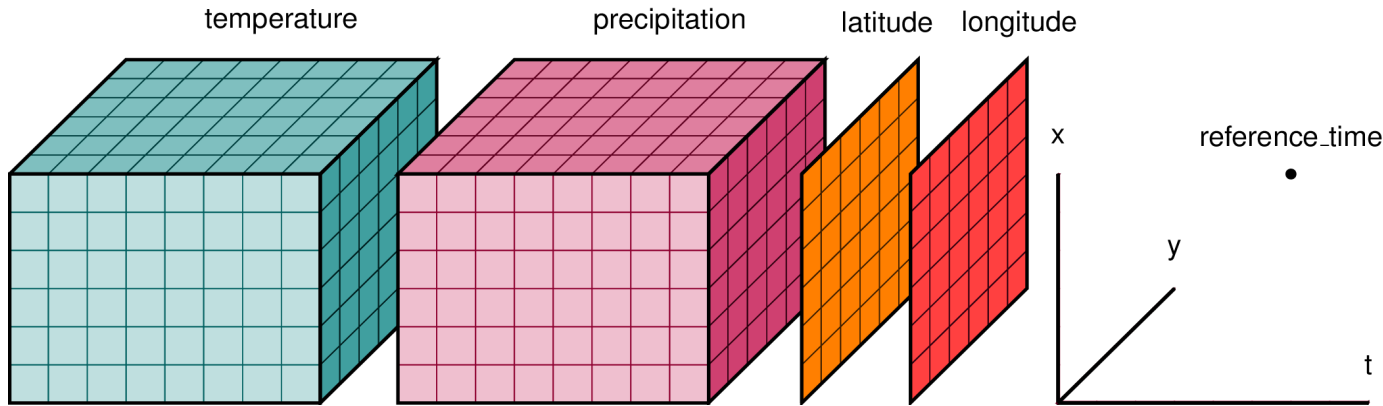
# Visualizing Sequential Climate Data Structures



## Extracting (Subsetting) Layers from the Larger Stack

- Instead of reading all layers in the stack, this example uses a *filter by keys* approach to reading in only the Zonal and Meridional wind components at the specified pressure level
- Particularly useful for big climate records in the cloud where latency (requests to the server) affect performance

# Comparing Cube and Sequential Climate Data Structures



2024-04-18

- Multidimensional slicing by dimensions important (Reanalysis)? **Cube.**
- Immediate access to one climate variable at-a-time (Realtime workflows)? **Sequential.**
- Querying time series across climate arrays or specific points? **Cube.**
- Use in realtime forecasts for operational meteorology? **Sequential.**

# Filetypes for Gridded Climate Data

- Diverse sets of filetypes are used to accommodate gridded data structures
- **Flat-file data structures include:**
  - CSV (.csv), Tab-delimited (.tsv), JSON (.json), and Excel Spreadsheets (.xlsx)
- **Cube data structures include:**
  - **NetCDF4 (.nc)** → Compression & chunking; unlimited dimensions; embedded metadata
  - **HDF5** → Hierarchies of climate attributes; file system-like structure; parallel I/O; suitable for big climate datasets; regular & irregular climate datasets; embedded metadata
  - **Zarr** → Stores data in hierarchies (file collections); creates chunked arrays for efficient retrieval from cloud storage bucket; parallel I/O; metadata contained in JSON in folder directory



# Filetypes for Gridded Climate Data

- **Sequential data structures include:**
  - **GRIB / GRIB2** → Stores arrays with message (header) as independent records; strong file compression; flexible grid structures; embeds metadata in each record
    - Optimal for quick single-record retrieval / updates
    - Inefficient when querying subsets of records (must read in entirety of called records)
- **Deciding which data structure and filetypes to use depends on your use case!**
- Zarr automatically performs chunking and explicitly defines dimension relationships in its data cube
  - GRIB2 does not chunk data and instead stores *entire* data arrays → must read through all records & ingest entire arrays to extract time series
  - Zarr is good for analysis-intensive datasets, while GRIB2 is good for realtime updates

# Delving into Zarr Archives

- Zarr archives can be created to act similarly to a file directory system
  - Climate attributes can contain **hierarchies** → think of subfolders within a parent folder
    - Hierarchies allow metadata to be shared between Zarr group parent (root) folders and its children
    - Metadata are contained within several **JSON files** created by the Zarr archive:
      - **.zattrs** → Metadata describing attributes on a group or array
      - **.zgroup** → Indicates data is a Zarr group
      - **.zarray** → Describes structure, dimensions, & chunking of climate data array
      - **.zmetadata** → Contains all consolidated metadata for *entire* Zarr archive
        - Preferable to use consolidated file when scanning structure of Zarr Archive

# Investigating a Typical Zarr Archive

Buckets > cmip6 > CMIP6 > HighResMIP > ECMWF > ECMWF-IFS-HR > highresSST-present > r1i1p1f1 > day > pr > gr > v20170915

CREATE FOLDER UPLOAD TRANSFER DATA OTHER SERVICES

Filter by name prefix only Filter Filter objects and folders Show Live objects only

<input type="checkbox"/>	Name	Size	Type	Created	Storage class	Last modified	Public access	Version h	
<input type="checkbox"/>	<a href="#">.zattrs</a>	7 KB	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">.zgroup</a>	24 B	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">.zmetadata</a>	16.6 KB	application/octet-stream	Feb 6, 2021, 6:06:33 AM	Standard	Feb 6, 2021, 6:06:33 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">lat/</a>	—	Folder	—	—	—	—	—	
<input type="checkbox"/>	<a href="#">lat_bnds/</a>	—	Folder	—	—	—	—	—	
<input type="checkbox"/>	<a href="#">lon/</a>	—	Folder	—	—	—	—	—	
<input type="checkbox"/>	<a href="#">lon_bnds/</a>	—	Folder	—	—	—	—	—	
<input type="checkbox"/>	<a href="#">pr/</a>	—	Folder	—	—	—	—	—	
<input type="checkbox"/>	<a href="#">time/</a>	—	Folder	—	—	—	—	—	
<input type="checkbox"/>	<a href="#">time_bnds/</a>	—	Folder	—	—	—	—	—	

## Go to Zarr Archive in Google Cloud Storage (GCS)

- CMIP6 daily high-resolution ECMWF precipitation data from a particular model simulation
- Notice JSON metadata files and folders for each of the gridded dataset's dimensions & climate variables
- Let's open the consolidated metadata JSON file (**.zmetadata**)

# Consolidated Metadata JSON File Snapshot

```
{
  "metadata": {
    ".zattrs": {
      "Conventions": "CF-1.7 CMIP-6.0",
      "activity_id": "HighResMIP",
      "branch_method": "no parent",
      "branch_time_in_child": 0.0,
      "branch_time_in_parent": 0.0,
      "cmor_version": "3.2.4",
      "contact": "chris.roberts@ecmwf.int",
      "coordinates": "lat bnds time bnds lon bnds",
      "creation_date": "2017-08-25T09:50:40Z",
      "data_specs_version": "01.00.23",
      "end_year": "2014",
      "experiment": "forced atmosphere experiment for 1950-2014",
      "experiment_id": "highresSST-present",
      "external_variables": "areacella",
      "forcing_index": 1,
      "frequency": "day",
      "further_info_url": "https://furtherinfo.es-doc.org/CMIP6.ECMWF.ECMWF-IFS-HR.highresSST-present.none.r1i1p1f1",
      "grid": "Data interpolated onto 0.5x0.5 regular grid from native Tco399 cubic octahedral reduced Gaussian grid; 91 levels; top level 1 hPa",
      "grid_label": "gr",
```

- Top level key provides high-level metadata describing the overall dataset
- Lists dataset name, publisher's contact information, & creation date
- Provides information on start & end dates, grid size (0.5° x 0.5°), temporal frequency (daily)

# Decoding the Data Chunking Strategy

```
"pr/.zarray": {  
  "chunks": [  
    186,  
    361,  
    720  
  ],  
  "compressor": {  
    "blocksize": 0,  
    "clevel": 5,  
    "cname": "lz4",  
    "id": "blosc",  
    "shuffle": 1  
  },  
  "dtype": "<f4",  
  "fill_value": 1.0000000200408773e+20,  
  "filters": null,  
  "order": "C",  
  "shape": [  
    23741,  
    361,  
    720  
  ],  
  "zarr_format": 2  
},
```

```
"pr/.zattrs": {  
  "_ARRAY_DIMENSIONS": [  
    "time",  
    "lat",  
    "lon"  
  ],  
  "cell_measures": "area: areacella",  
  "cell_methods": "area: time: mean",  
  "comment": "includes both liquid and solid phases",  
  "long_name": "Precipitation",  
  "original_name": "tp",  
  "original_units": "kg*m-2*s-1",  
  "standard_name": "precipitation_flux",  
  "units": "kg m-2 s-1"  
},
```

- Climate attribute **pr/** is **precipitation flux**
- Overall shape of the precipitation flux climate dataset is **(23741 x 361 x 720)**
  - Contains a total of 23,741 2D-arrays of size **(720 x 361)** covering the entire globe
- Dataset is **chunked** into partitions of **186-time steps** with full longitude & latitude coverage



# Decoding the Data Chunking Strategy

```
"pr/.zarray": {  
  "chunks": [  
    186,  
    361,  
    720  
  ],  
  "compressor": {  
    "blocksize": 0,  
    "clevel": 5,  
    "cname": "lz4",  
    "id": "blosc",  
    "shuffle": 1  
  },  
  "dtype": "<f4",  
  "fill_value": 1.0000000200408773e+20,  
  "filters": null,  
  "order": "C",  
  "shape": [  
    23741,  
    361,  
    720  
  ],  
  "zarr_format": 2  
},
```

```
"pr/.zattrs": {  
  "_ARRAY_DIMENSIONS": [  
    "time",  
    "lat",  
    "lon"  
  ],  
  "cell_measures": "area: areacella",  
  "cell_methods": "area: time: mean",  
  "comment": "includes both liquid and solid phases",  
  "long_name": "Precipitation",  
  "original_name": "tp",  
  "original_units": "kg*m-2*s-1",  
  "standard_name": "precipitation_flux",  
  "units": "kg m-2 s-1"  
},
```

- Instead of loading in 23,741 gridded arrays, only 186 arrays are loaded into memory at any time
- This data is *only* chunked by its **time dimensions** → enables chunks to be read, processed, and transmitted in parallel without having to load entire dataset
- Zarr's native chunking saves memory, increases processing, & enables parallel performance

# Investigating the Chunks for Precipitation Data

Buckets

>

cmip6

>

CMIP6

>

HighResMIP

>

ECMWF

>

ECMWF-IFS-HR

>

highresSST-present

>

r1i1p1f1

>

day

>

pr

>

gr

>

v20170915

>

pr

CREATE FOLDER

UPLOAD

TRANSFER DATA

OTHER SERVICES

Filter by name prefix only

Filter

Filter objects and folders

Show **Live objects only**

<input type="checkbox"/>	Name	Size	Type	Created	Storage class	Last modified	Public access	Vers	
<input type="checkbox"/>	<a href="#">.zarray</a>	388 B	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">.zattrs</a>	384 B	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">0.0.0</a>	122.9 MB	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">1.0.0</a>	123.6 MB	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">10.0.0</a>	122.9 MB	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">100.0.0</a>	119.8 MB	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">101.0.0</a>	124.6 MB	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">102.0.0</a>	120 MB	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">103.0.0</a>	124.3 MB	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">104.0.0</a>	121 MB	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	
<input type="checkbox"/>	<a href="#">105.0.0</a>	124 MB	application/octet-stream	Jan 31, 2021, 10:15:56 AM	Standard	Jan 31, 2021, 10:15:56 AM	Value hidden <a href="#">Copy URL</a>	—	

**Go to pr/ Chunked Array in GCS**

- Notice data chunks labeled by their ID → each chunk contains no more than 186 time slices
- Chunk containing the first 186 time slices (closest to 1950) is denoted as **0.0.0**
- Each chunk contains an average 121 MB worth of precipitation data; instead of handling entire dataset, we're working with manageable chunks that fit into memory for quick querying!

# Zarr Data Fetching Strategy

- Rather than retrieve and process over the entire gridded dataset, Zarr creates **chunks** (partitions) in the dataset along the data's dimensions
  - Data can be chunked by *any* dimension, not only time
  - Finding optimal chunking strategy requires balancing **requests to the server with the amount of data that can be stored in memory** (this is a balancing act)
- Zarr chunk ID's include information in their labeling:
  - **0.0.0** → First chunk of **time**, first chunk of **latitude**, first chunk of **longitude**
  - **1.0.0** → Second chunk of **time**, first chunk of **latitude**, first chunk of **longitude**
  - **2.0.0** → Third chunk of **time**, first chunk of **latitude**, first chunk of **longitude**
- This allows for efficient querying by only focusing on desired chunks AND processing chunks in parallel for intensive analytical tasks

# Delving into GRIB / GRIB2 (GRIdded Binary) Filetypes

- **GRIdded Binary (GRIB)** filetypes pack climate data into isolated records with sections
  - GRIB files are often separated by time step and contain multiple climate variables for that specific time step
    - Separating files by time step keeps file sizes small and data moderate in size
  - Each climate variable within GRIB file is stored as its own independent record
- GRIB files come with metadata explaining each **climate variable record** for quick retrieval
  - Record metadata is *not shared* → each record possesses its own climate data
  - Metadata includes information about grid resolution; data scaling; data encoding; image compression algorithm
- GRIB filetypes allow for efficient data querying of climate attributes (records); supports advanced compression & irregular spatial grids; good for dissemination in cloud storage

# Investigating a Typical GRIB / GRIB2 Cloud Archive

AWS S3 Explorer **noaa-rtma-pds** ☐ Hide folders? Folder Bucket 24121

25 entries per page Search: rtma2p5.2025

Object	Last Modified	Timestamp	Size
<a href="#">rtma2p5.20250101/</a>			
<a href="#">rtma2p5.20250102/</a>			
<a href="#">rtma2p5.20250103/</a>			
<a href="#">rtma2p5.20250104/</a>			

Showing 1 to 4 of 4 entries (filtered from 24,121 total entries)

« < 1 > »

## [Go to NOAA-RTMA Amazon S3 Cloud Storage](#)

- NOAA-RTMA Amazon S3 storage bucket containing near-realtime updates of hourly meteorological data
- GRIB2 files organized in folders specified by { dataset name } and { date of analysis }
- Folder structure creates hierarchy for moderate-sized hourly GRIB2 files to exist within



# Investigating a Typical GRIB / GRIB2 Cloud Archive

AWS S3 Explorer

noaa-rtma-pds / rtma2p5.20250103

Hide folders? Folder Bucket

25 entries per page

Search: rtma2p5.t

Object	Last Modified	Timestamp	Size
<a href="#">rtma2p5.t00z.2dvaranl_ndfd.grb2_wexp</a>	a day ago	2025-01-02 16:52:38	80 MB
<a href="#">rtma2p5.t00z.2dvaranl_ndfd.grb2_wexp.idx</a>	a day ago	2025-01-02 16:52:41	662 Bytes
<a href="#">rtma2p5.t00z.2dvarerr_ndfd.grb2_wexp</a>	a day ago	2025-01-02 16:52:34	58 MB
<a href="#">rtma2p5.t00z.2dvarerr_ndfd.grb2_wexp.idx</a>	a day ago	2025-01-02 16:52:37	961 Bytes
<a href="#">rtma2p5.t00z.2dvarges_ndfd.grb2_wexp</a>	a day ago	2025-01-02 16:52:36	80 MB

**Go to NOAA-RTMA Records for 2025-01-03**

- Looking at hourly data for January 3, 2025 → GRIB2 files labeled by the hour (UTC) time they contain meteorological data for
- High resolution of RTMA CONUS data (2.5 km x 2.5 km) and many climate attributes (i.e., temp, cloud cover, wind speed & direction, dewpoint temp) create moderate sized 80 MB files

# Looking at the Associated Metadata (Index) File

```
1:0:d=2025010300:HGT:surface:anl:
2:7490118:d=2025010300:PRES:surface:anl:
3:14980236:d=2025010300:TMP:2 m above ground:anl:
4:21065993:d=2025010300:DPT:2 m above ground:anl:
5:27151750:d=2025010300:UGRD:10 m above ground:anl:
6:32769386:d=2025010300:VGRD:10 m above ground:anl:
7:38387022:d=2025010300:SPFH:2 m above ground:anl:
8:45409020:d=2025010300:WDIR:10 m above ground:anl:
9:51026656:d=2025010300:WIND:10 m above ground:anl:
10:56644292:d=2025010300:GUST:10 m above ground:anl:
11:62261928:d=2025010300:VIS:surface:anl:
12:68815805:d=2025010300:CEIL:cloud ceiling:anl:
13:77242165:d=2025010300:TCDC:entire atmosphere (considered as a single layer):anl:
```

- Index file accompanies each GRIB2 file → provides rapid index to each **record** contained within GRIB2 file
- Includes information that data is **analysis (anl)** data, not forecasts:
  - **Byte offset** → Exact location in bytes for the start of each independent record
  - **Timestamp** → Exact date & time for data within GRIB2 file (2025-01-03 00:00 UTC)
  - **Variable name** → Shorthand abbreviation (identifier) for each climate variable
  - **Vertical layer** → How far above (or at) surface level data was collected at

# Overview of GRIB / GRIB2 Filetypes

- GRIB / GRIB2 files support independent records of climate variables
  - Each variable (record) can be individually queried without reading the entire GRIB2 file
- Cannot chunk GRIB / GRIB2 files → must read entire record(s) when subsetting
  - Lack of chunking makes time series queries at specific locations or sub-regions computationally expensive
  - GRIB / GRIB2 is optimal for selecting entire climate variable records at-a-time
- Allows for random access of entire records within GRIB / GRIB2 files
- GRIB / GRIB2 file does not create a data cube with explicit spatial indices
  - Each record must parse its metadata to search for a point (using nearest neighbor lookup)
  - And individual scan records across time to search for point and retrieve data
  - Each record must also be *decompressed* into original climate data values

A satellite view of Earth is shown on the left side of the slide, showing the Americas and surrounding oceans.

## 2. Connecting to CMIP6 in Google Cloud Storage with Python

# What is Object Storage? How are Data Stored in the Cloud?

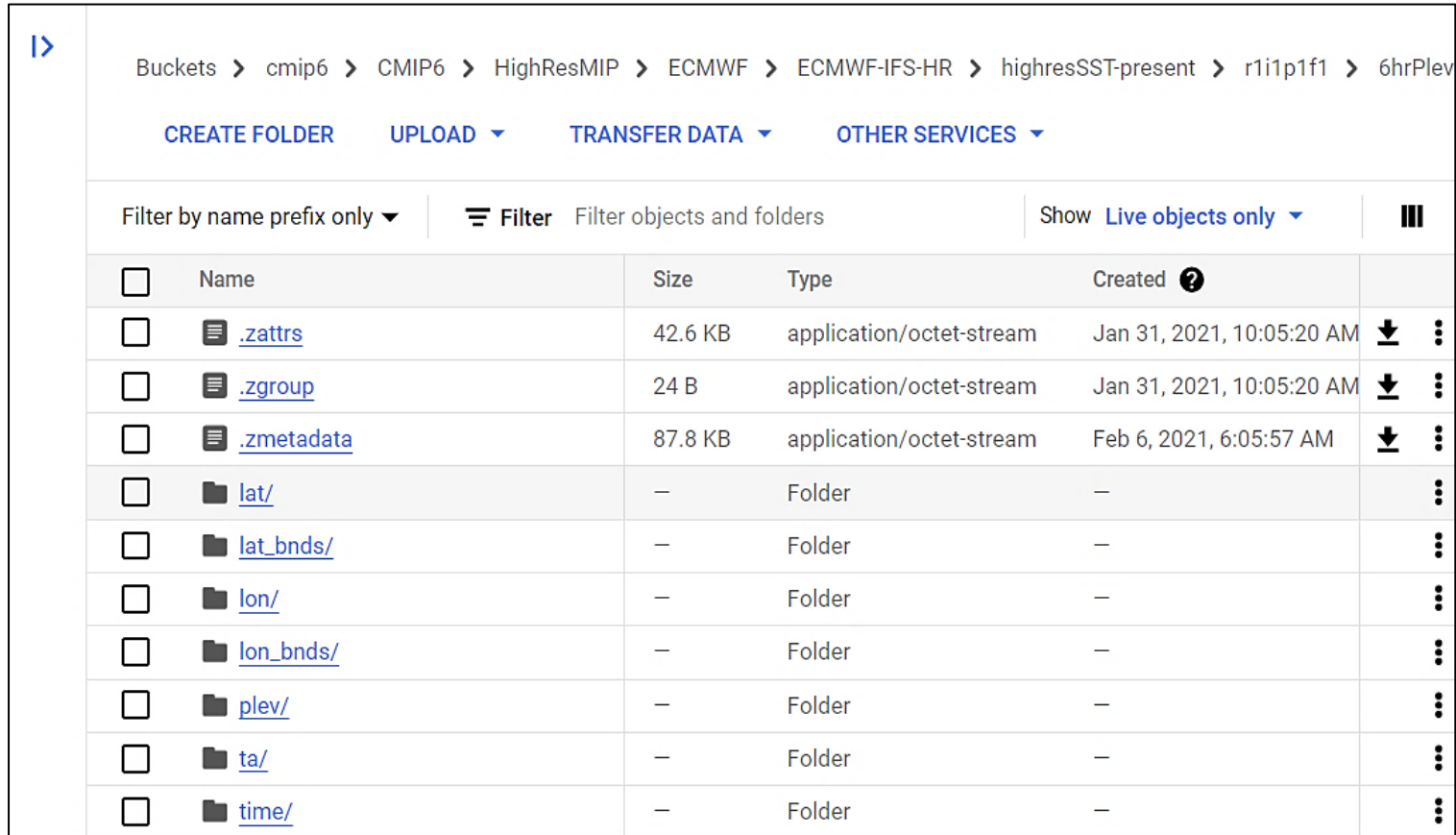
- Python contains a growing number of community-supported libraries for handling data structures in cloud object storage
- Cloud Object Storage services allow developers to create flexible file system hierarchies holding *petabytes* of optimized data structures entirely in the cloud
  - **Google Cloud Storage (GCS)**
  - **Amazon S3**
  - **Microsoft Azure**
- Cloud Object Storage writes files into **BLOBs** (**B**inary **L**arge **OB**ject) stored in folder hierarchies
  - Maintainers should employ proper file naming conventions to navigate directories
- Public climate data like CMIP6 are stored as BLOBs inside of *buckets*
- Python APIs exist to access cloud storage providers entirely *within* Python



# Python Libraries for Working with Cloud Object Storage

- Python community and official developers release open-source API tools for cloud storage
  - **GCSFS:** Google Cloud Storage File System API that integrates a *Pythonic* interface for streaming, writing data chunks, & reading data files directly from GCS
  - **Boto3:** Low-level Software Development Kit (SDK) for managing data in Amazon S3 that integrates a *Pythonic* interface for streaming, writing, & reading files directly from S3
  - **S3FS:** Simplified Amazon S3 File System API that allows manipulation & retrieval of data blobs in S3 buckets using folder pathways → acts as a wrapper around Boto3
  - **FSSPEC:** Filesystem Spec that supports wide-versatility for different cloud & local file storage systems → community-supported and generalized for any cloud provider
  - **Azure-Storage-Blob:** Microsoft Azure Cloud Storage library for streaming, downloading, writing, & navigating buckets & blobs for Azure-specific storage

# Visualizing CMIP6 Zarr Archive in Google Cloud Storage (GCS)



The screenshot displays the Google Cloud Storage interface for a bucket named 'cmip6'. The breadcrumb path is: Buckets > cmip6 > CMIP6 > HighResMIP > ECMWF > ECMWF-IFS-HR > highresSST-present > r1i1p1f1 > 6hrPlev. The interface includes buttons for 'CREATE FOLDER', 'UPLOAD', 'TRANSFER DATA', and 'OTHER SERVICES'. A filter bar shows 'Filter by name prefix only' and 'Filter objects and folders'. The table below lists the objects and folders in the bucket.

<input type="checkbox"/>	Name	Size	Type	Created ?	
<input type="checkbox"/>	<a href="#">.zattrs</a>	42.6 KB	application/octet-stream	Jan 31, 2021, 10:05:20 AM	
<input type="checkbox"/>	<a href="#">.zgroup</a>	24 B	application/octet-stream	Jan 31, 2021, 10:05:20 AM	
<input type="checkbox"/>	<a href="#">.zmetadata</a>	87.8 KB	application/octet-stream	Feb 6, 2021, 6:05:57 AM	
<input type="checkbox"/>	<a href="#">lat/</a>	—	Folder	—	
<input type="checkbox"/>	<a href="#">lat_bnds/</a>	—	Folder	—	
<input type="checkbox"/>	<a href="#">lon/</a>	—	Folder	—	
<input type="checkbox"/>	<a href="#">lon_bnds/</a>	—	Folder	—	
<input type="checkbox"/>	<a href="#">plev/</a>	—	Folder	—	
<input type="checkbox"/>	<a href="#">ta/</a>	—	Folder	—	
<input type="checkbox"/>	<a href="#">time/</a>	—	Folder	—	

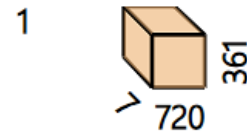
- CMIP6 high-resolution ECMWF historical 6-hour air temperature
- Gridded arrays are organized into cloud-friendly Zarr archive
  - Data cube allows efficient multi-dimensional queries
  - Supports chunking & distributed processing
- Each dimension receives its own folder & metadata

# Lazily-Loading CMIP6 Zarr Archive into Python with GCSFS

xarray.DataArray 'ta' (time: 94964, plev: 7, lat: 361, lon: 720)



	Array	Chunk
Bytes	643.66 GiB	249.86 MiB
Shape	(94964, 7, 361, 720)	(36, 7, 361, 720)
Dask graph	2638 chunks in 2 graph layers	
Data type	float32 numpy.ndarray	



## ▼ Coordinates:

lat	(lat)	float64	-90.0 -89.5 -89.0 ... 89.5 90.0
lon	(lon)	float64	0.0 0.5 1.0 ... 358.5 359.0 359.5
plev	(plev)	float64	9.25e+04 8.5e+04 ... 2.5e+04 5e+03
time	(time)	datetime64[ns]	1950-01-01 ... 2014-12-31T18:00:00

- Air temperature contains 643.66 GB worth of data (Jan. 1, 1950 – Dec. 31, 2014)
  - Variable partitioned into manageable chunks averaging 250 MB in size
  - Each chunk contains only 36 time slices, all 7 pressure levels, & lon / lat dimensions preserved
    - Entire archive partitioned into 2,638 total data chunks
- Data organized into multi-dimensional cube as **(time, pressure level, lat, lon)**
- Chunks allow efficient querying, filtering, & processing of large records in-parallel

# Connecting to a CMIP6 Dataset Zarr Archive in GCS



```
import gcsfs
# Google Cloud Storage (GCS) Bucket Name
bucket = "cmip6"

# GCS File System Client
client = gcsfs.GCSFileSystem(
    project = bucket
)
# GCS internal connection to ECMWF High-Resolution simulations folder
main_path = f"gs://{bucket}/CMIP6/HighResMIP/ECMWF/ECMWF-IFS-HR"
```

Import GCSFS library. Specify name of GCS bucket → create an instance of GCSFS client in Python runtime and define main path to reach high-resolution ECMWF simulations.

# Connecting to a CMIP6 Dataset Zarr Archive in GCS



```
# Simulation using fully-coupled model for ocean & atmosphere
```

```
experiment = "highresSST-present"
```

```
# Realization initialization conditions
```

```
simulation = "r1i1p1f1"
```

```
# Temporal resolution
```

```
time = "6hrPlevPt"
```

```
# Climate variable
```

```
var = "ta"
```

Parameters that modify folder pathway leading to desired Zarr archive → use simulations provided by fully-coupled model at every 6-hours for air temp at all pressure levels.



# Connecting to a CMIP6 Dataset Zarr Archive in GCS



# Type of Grid

```
gr = "gr"
```

# Simulation dataset version

```
simulation = "v20170915"
```

# Concatenate folder pathway to connect with desired Zarr Archive:

```
gcs_path=f"{main_path}/{experiment}/{simulation}/{time}/{var}/{gr}/{version}"  
print(gcs_path)
```

```
gs:// cmip6 / CMIP6 / HighResMIP / ECMWF / ECMWF-IFS-HR / highresSST-  
present / r1i1p1f1 / 6hrPlevPt / ta / gr / v20170915
```

# Lazily-Load and View the Connected Zarr Archive's Structure



```
# Connect GCS Client to Zarr Archive
```

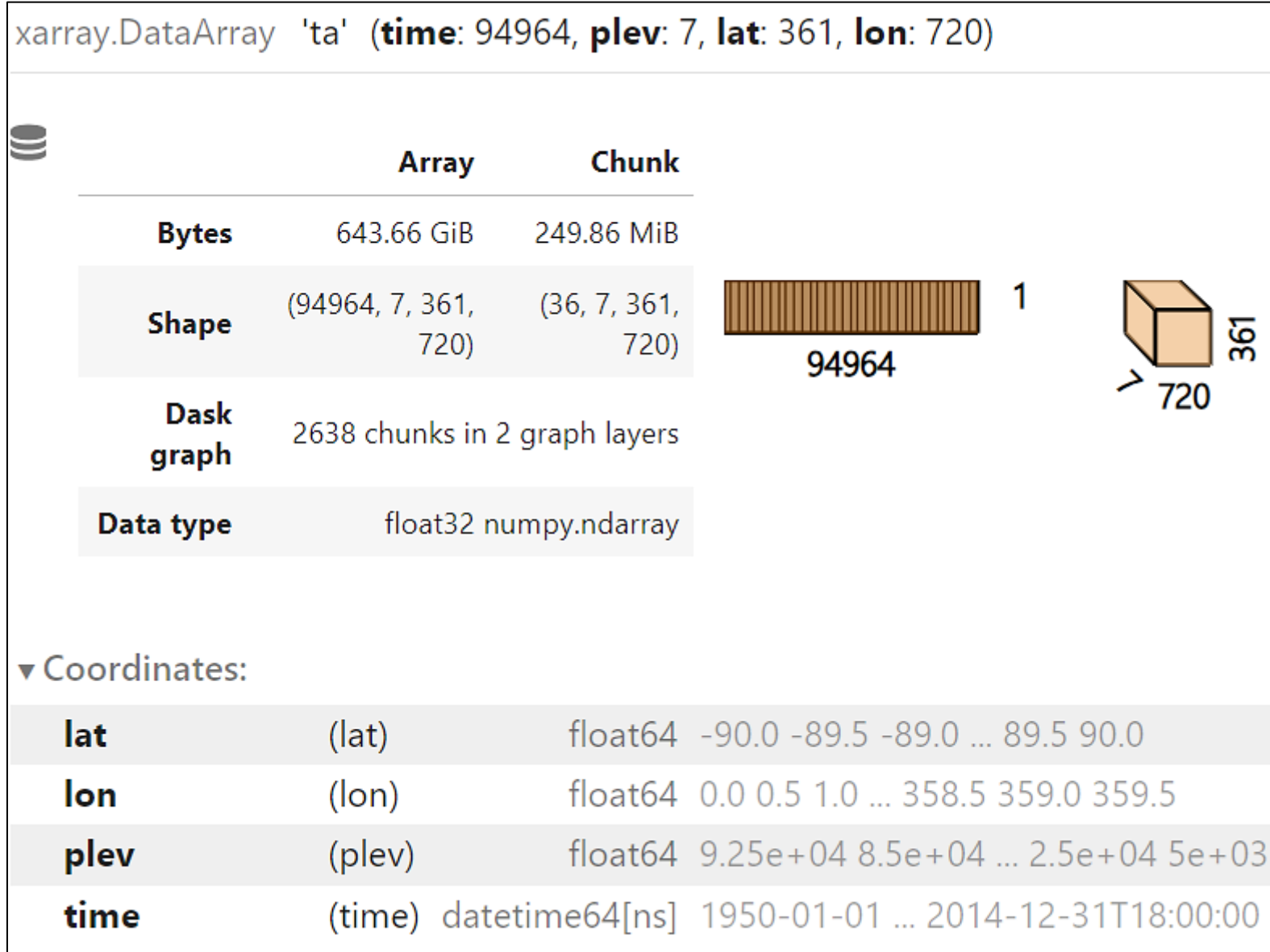
```
gcs_client = client.get_mapper(  
    gcs_path  
)
```

```
# Lazily load entire dataset into Python
```

```
ds = xr.open_zarr(  
    gcs_client ,  
    consolidated = True ,  
    chunks = "auto"  
)
```

```
ds # View lazily-loaded dataset
```

# Lazily-Loading CMIP6 Zarr Archive into Python with GCSFS



# A Python Roadmap

for Accessing and Interfacing with Big  
Environmental Data in Cloud Storage  
Providers

**Ryan Paul Lafler**

**[rplafler@premier-analytics.com](mailto:rplafler@premier-analytics.com)**



**Premier Analytics  
Consulting, LLC**

**[www.Premier-Analytics.com](http://www.Premier-Analytics.com)**