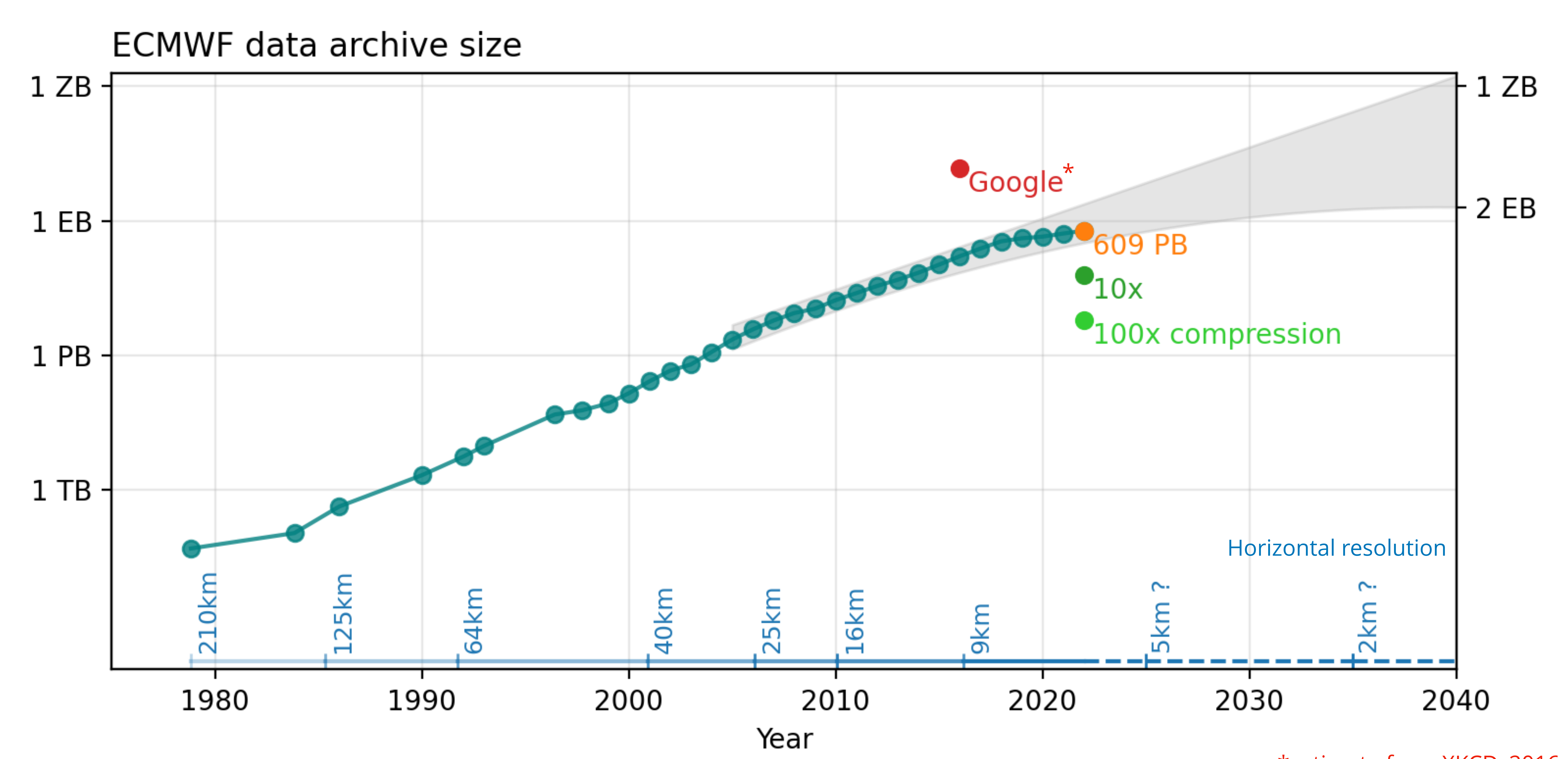
Data challenges in Earth Sciences

What is the real information in weather and climate data?

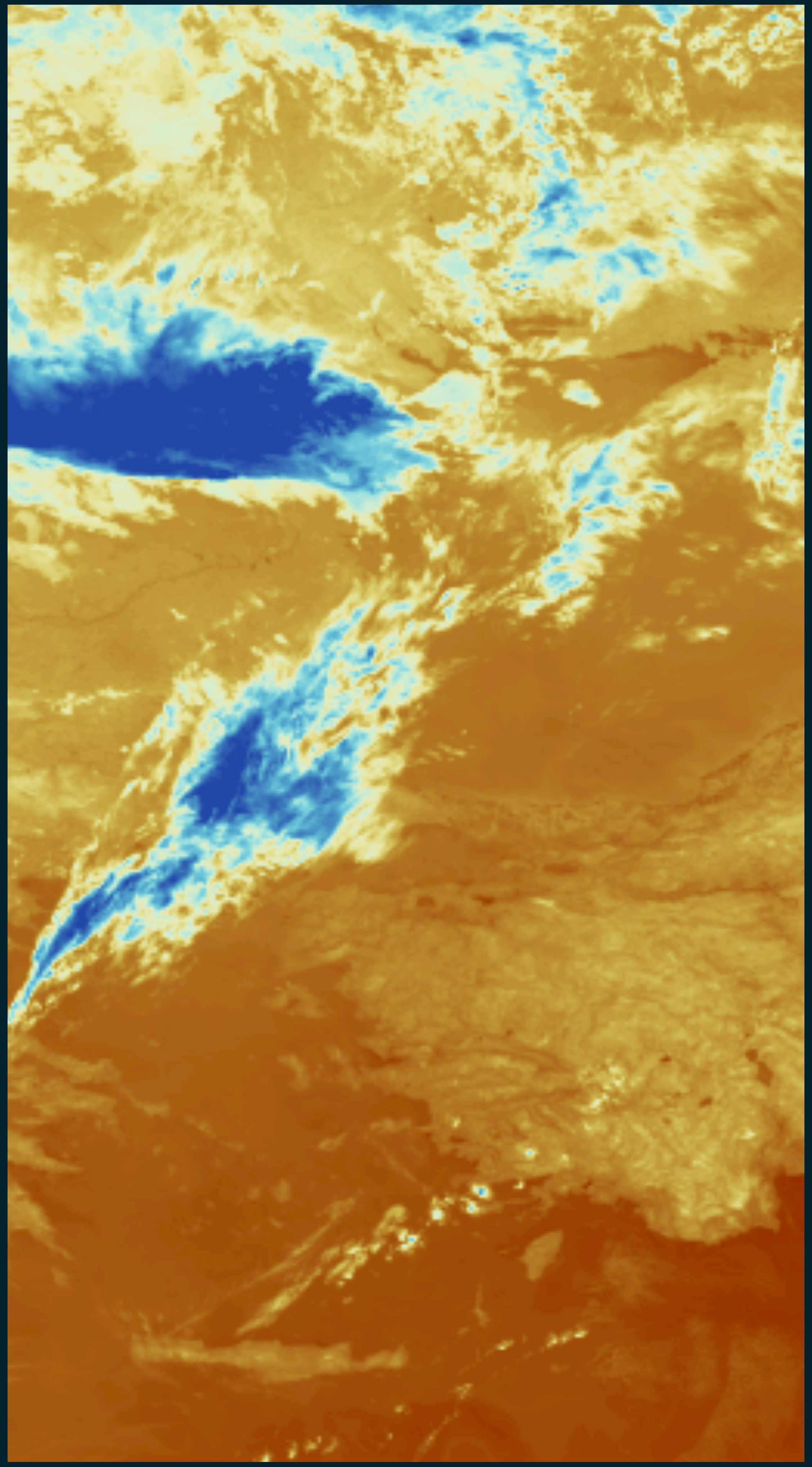
Milan Klöwer
Massachusetts Institute of Technology

Earth system data: Will we enter the *Google regime*?

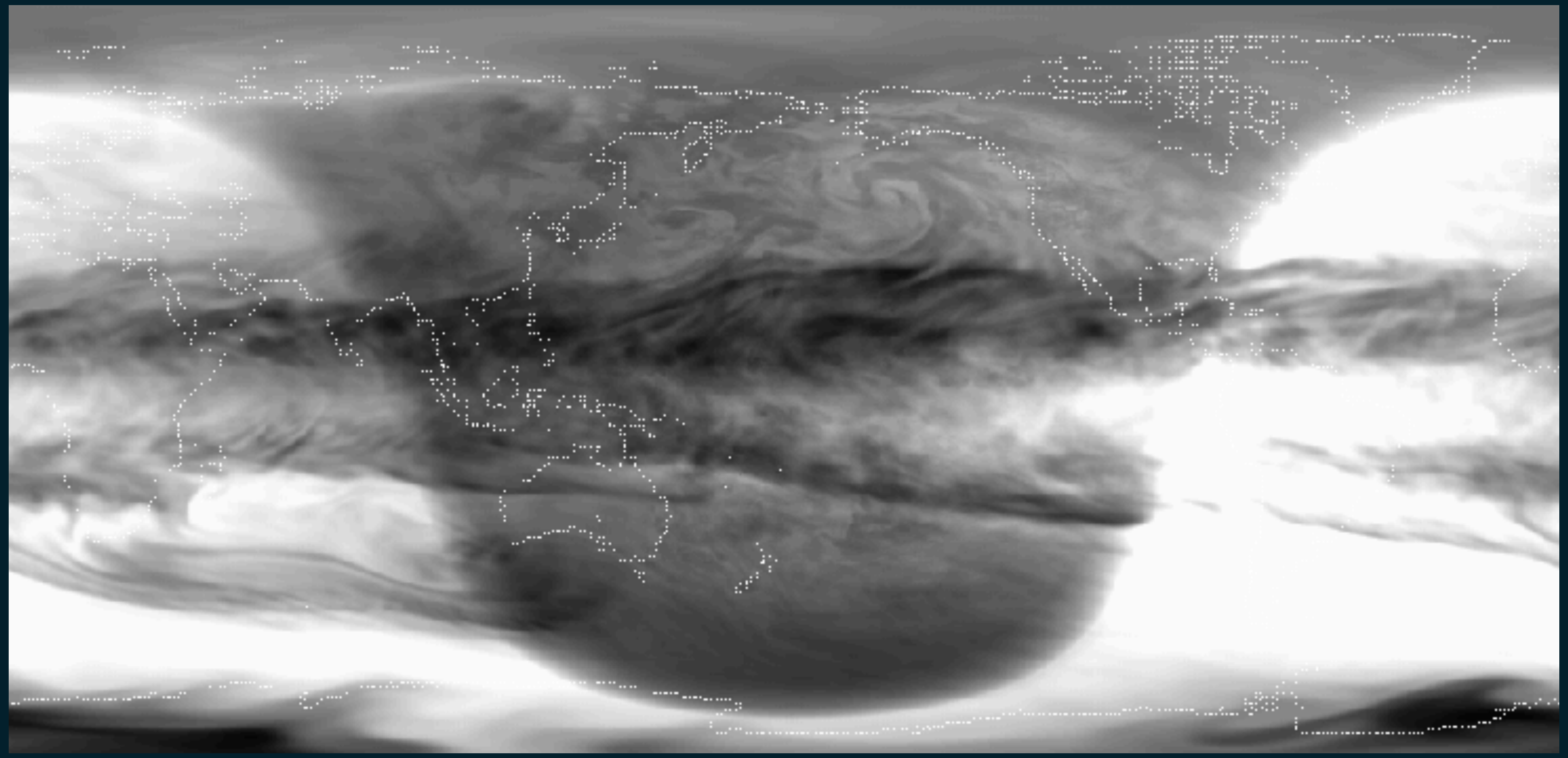


*estimate from XKCD, 2016

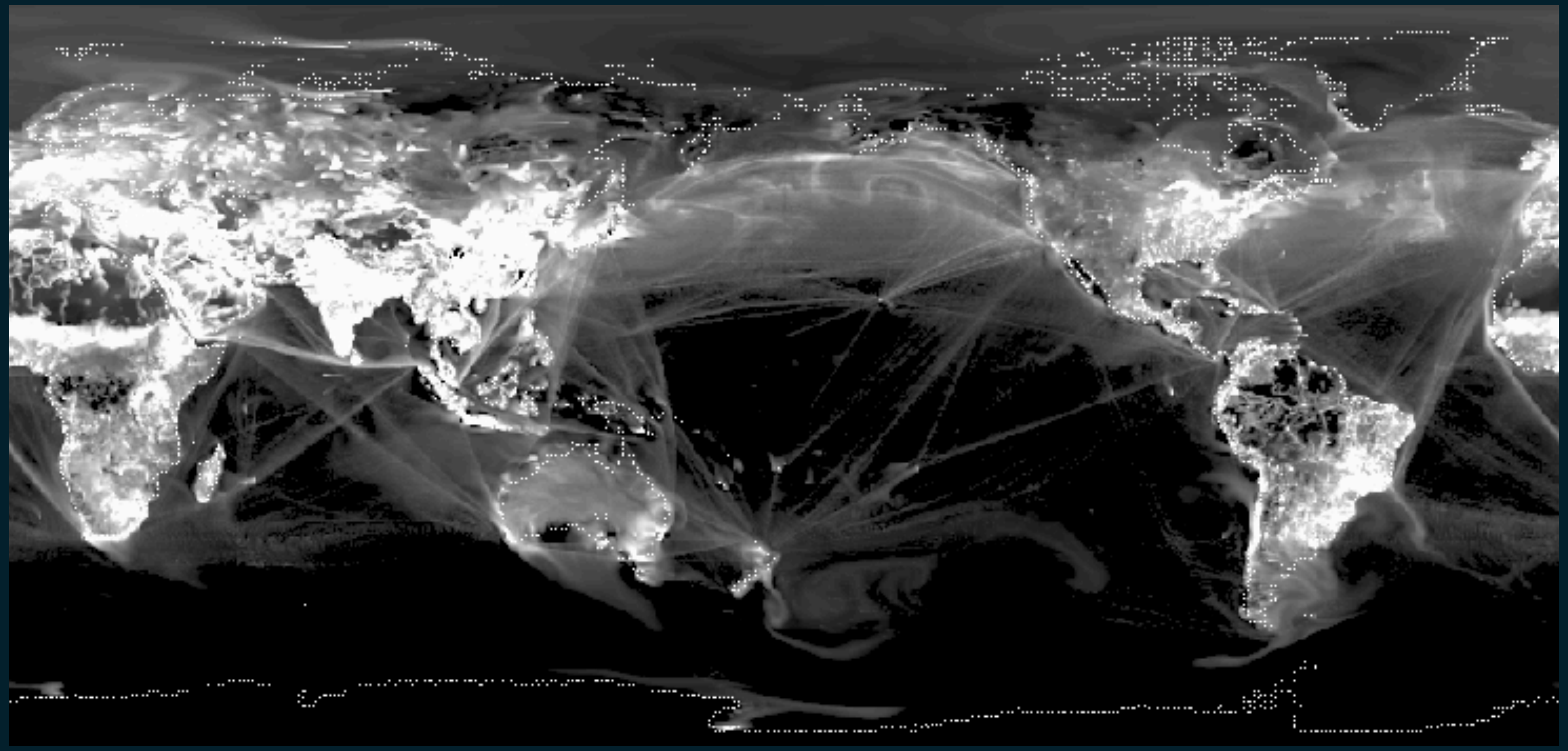
What data are we dealing with?



Brightness temperature



NO_2 in the stratosphere



NO_2 at the surface

- **Many** different variables
- Varying uncertainties
- Linear or log-distributed
- Possibly many zeros

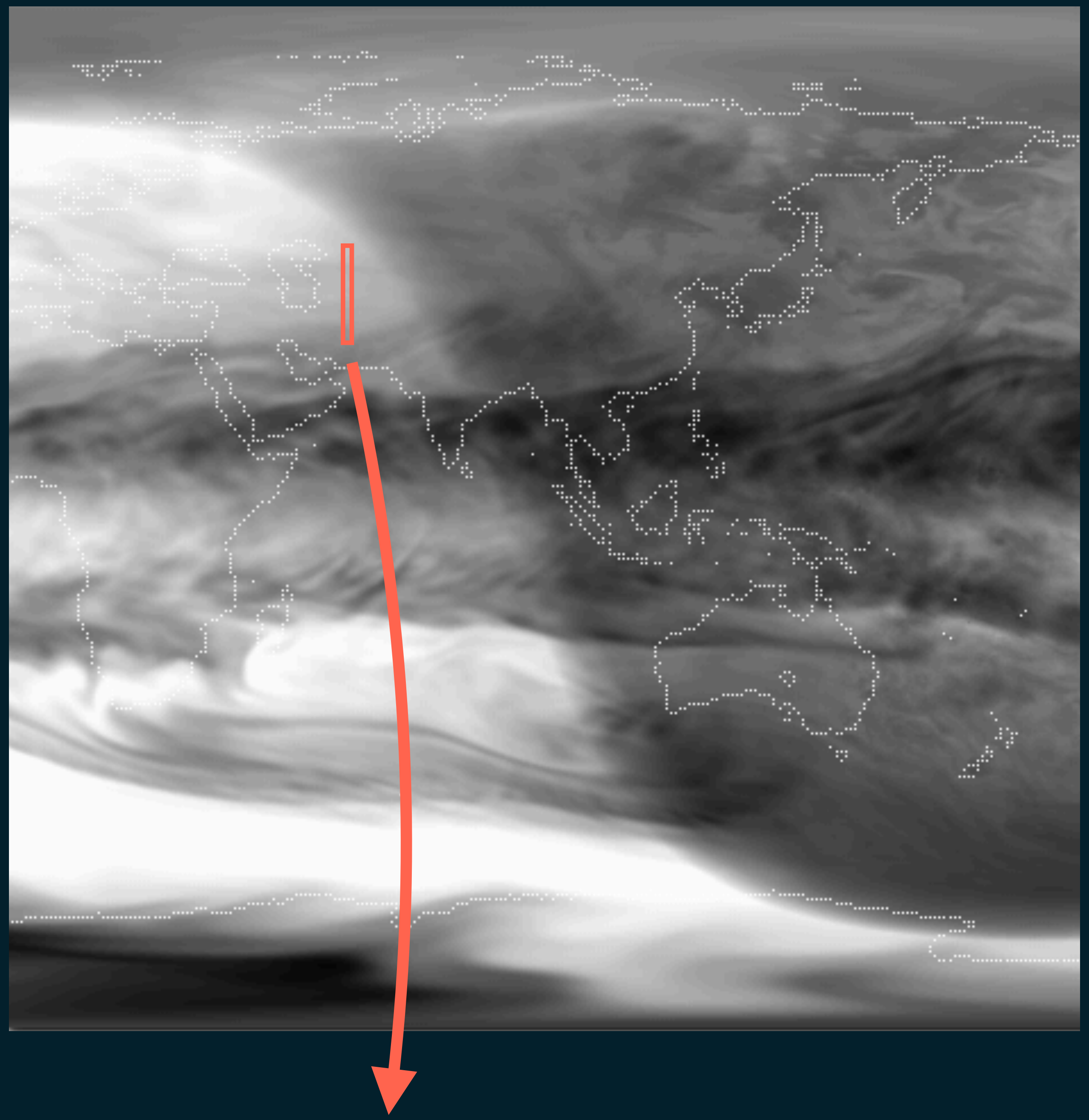
- Smoothness varies spatially
- Strong gradients
- Point sources

- Unstructured grids
- Spectral coefficients
- Masked data

What is real and false information in data?

Problem: What is the uncertainty in data and how can it be estimated if unknown?

Possible solution:
Find an objective way to separate real and false information!



```
0.050386034
0.050390966
0.05040059
0.050441727
0.05046302
0.05046855
0.050488267
0.05050127
0.050520953
0.05052939
0.050532646
```

Encoded in bits

Real!? False!?

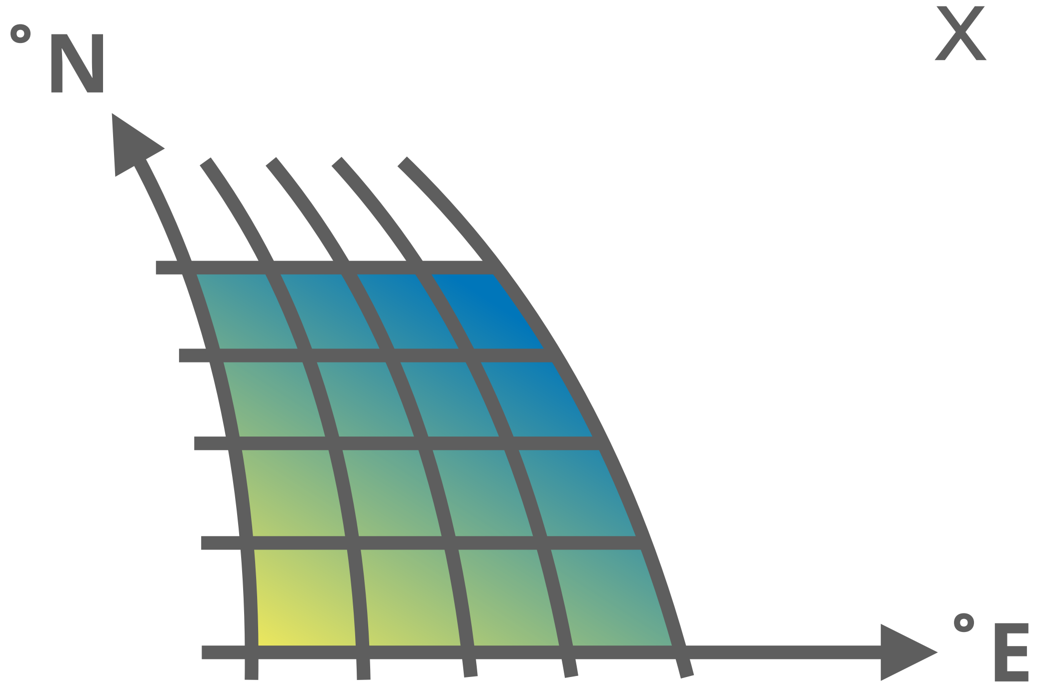
```
00111101010011100110000110010110
00111101010011100110011011000010
00111101010011100111000011011001
00111101010011101001101111111100
00111101010011101011001001010000
00111101010011101011100000011100
00111101010011101100110011001001
00111101010011101101101001101011
```

Compressible Incompressible

Bitwise real information content

defined here as the **mutual bitwise information in adjacent grid points**

1 Gridded data

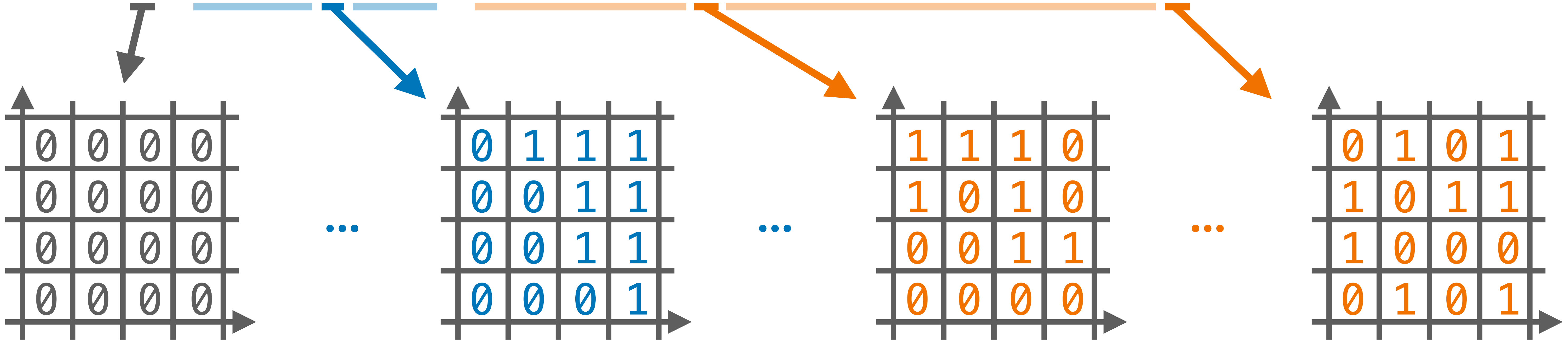


2 Data as binary floating-point numbers

sign exponent mantissa bits

`x = 1.23… = 0 01111111 00110100100011001010001`

3 Analyse bits in adjacent grid points for every bit position



sign bits

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

…

5th exponent bits

0	1	1	1
0	0	1	1
0	0	1	1
0	0	0	1

…

8th mantissa bits

1	1	1	0
1	0	1	0
0	0	1	1
0	0	0	0

…

last mantissa bits

0	1	0	1
1	0	1	1
1	0	0	0
0	1	0	1

4 The mutual information M between bits in adjacent grid points

M = 0 bits

If all bits are **identical**

M ≈ 1 bit

If `0` is **certainly adjacent** to `0`; `0` and 1 occur equally frequent

M > 0 bit

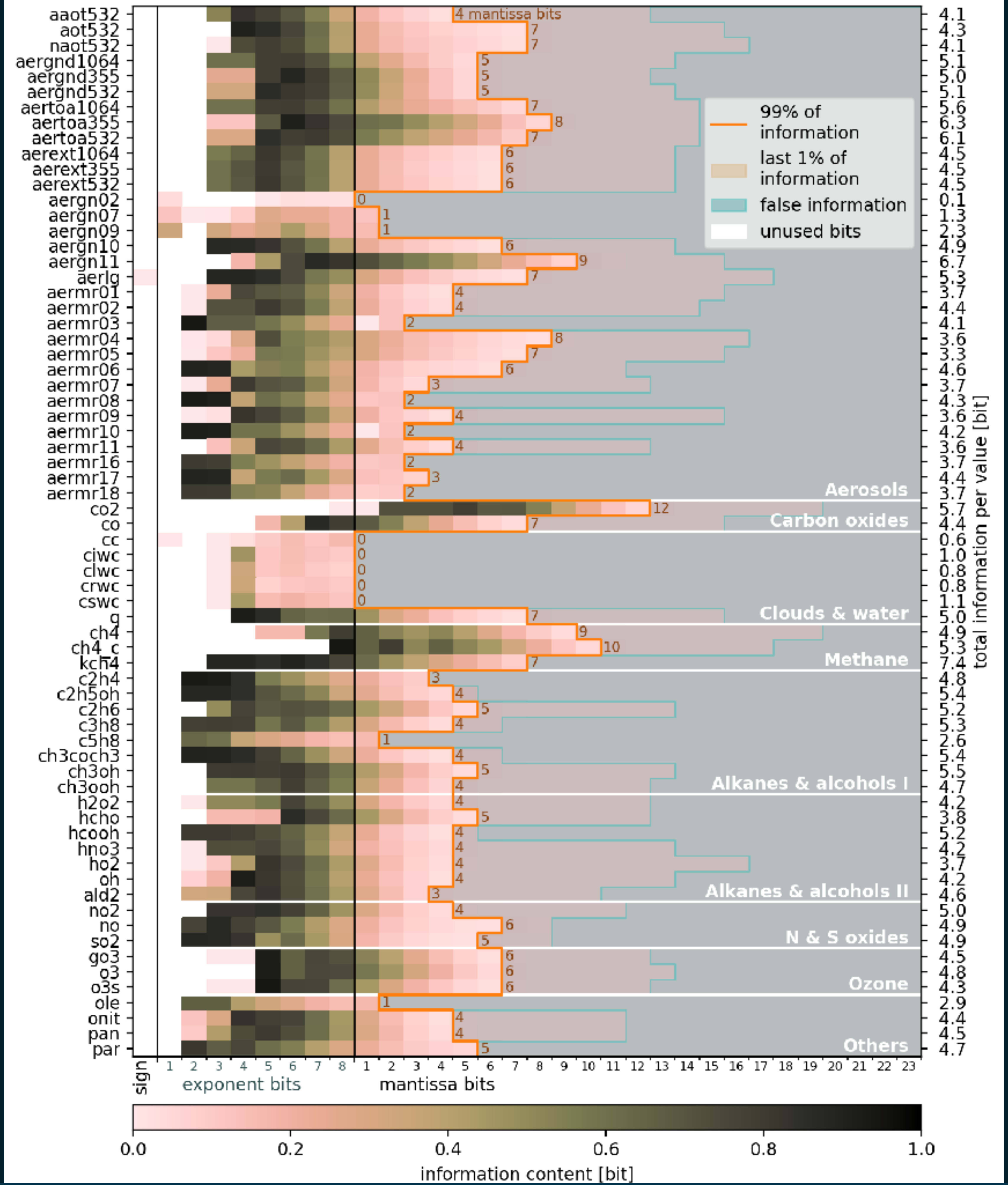
If `0` is **likely adjacent** to `0` and 1 is likely adjacent to 1

M = 0 bit

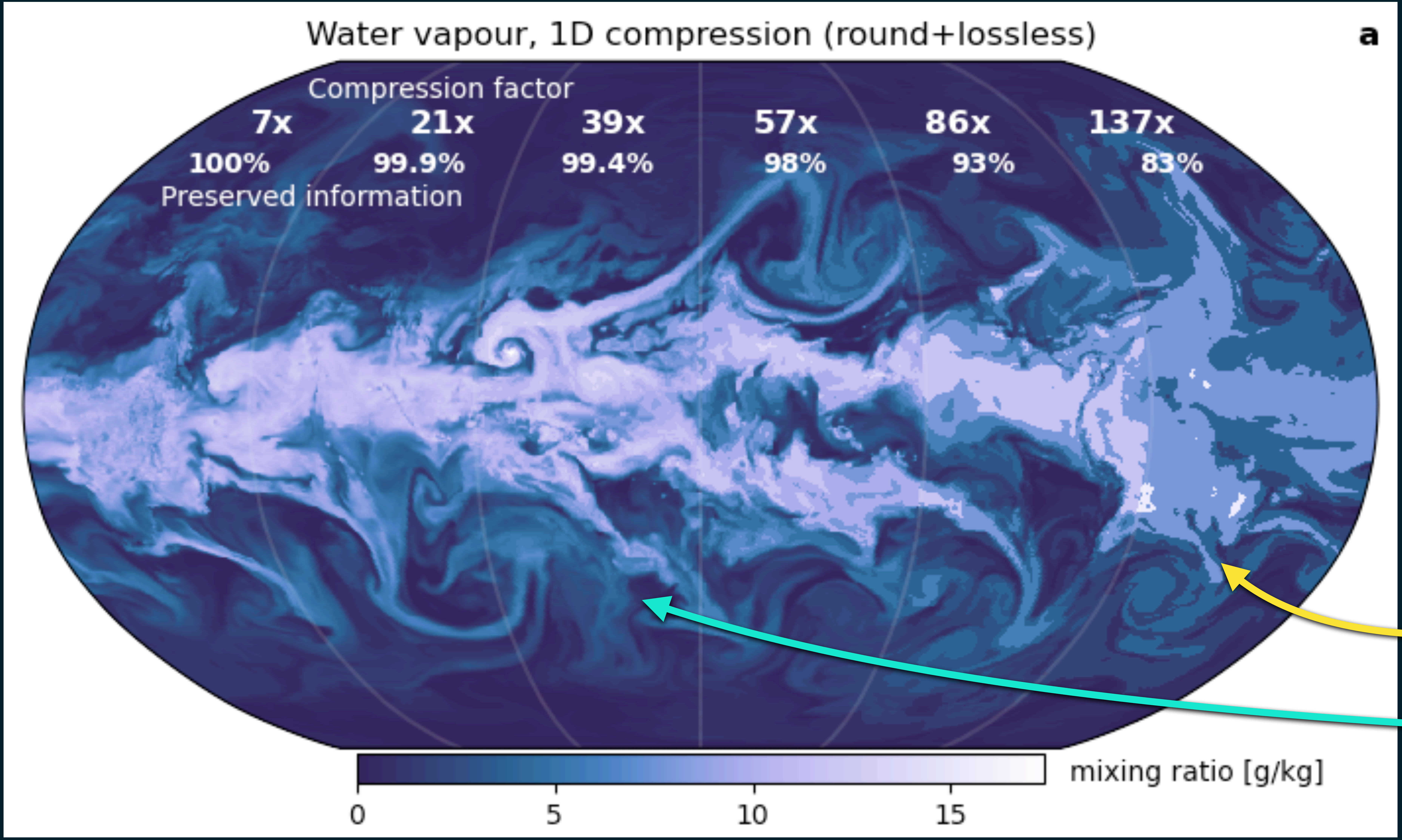
If adjacent bits are **independent**

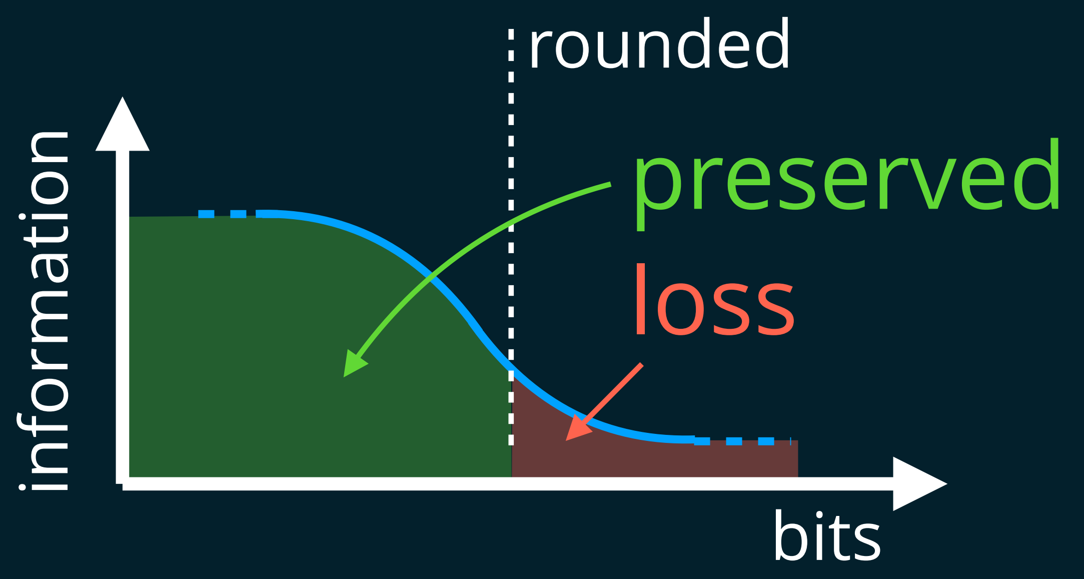
Bitwise
real information content

- Every variable requires a different precision
- Many bits do not contain real information
- Preserve only the bits with real information



Information-preserving compression





303.2519283

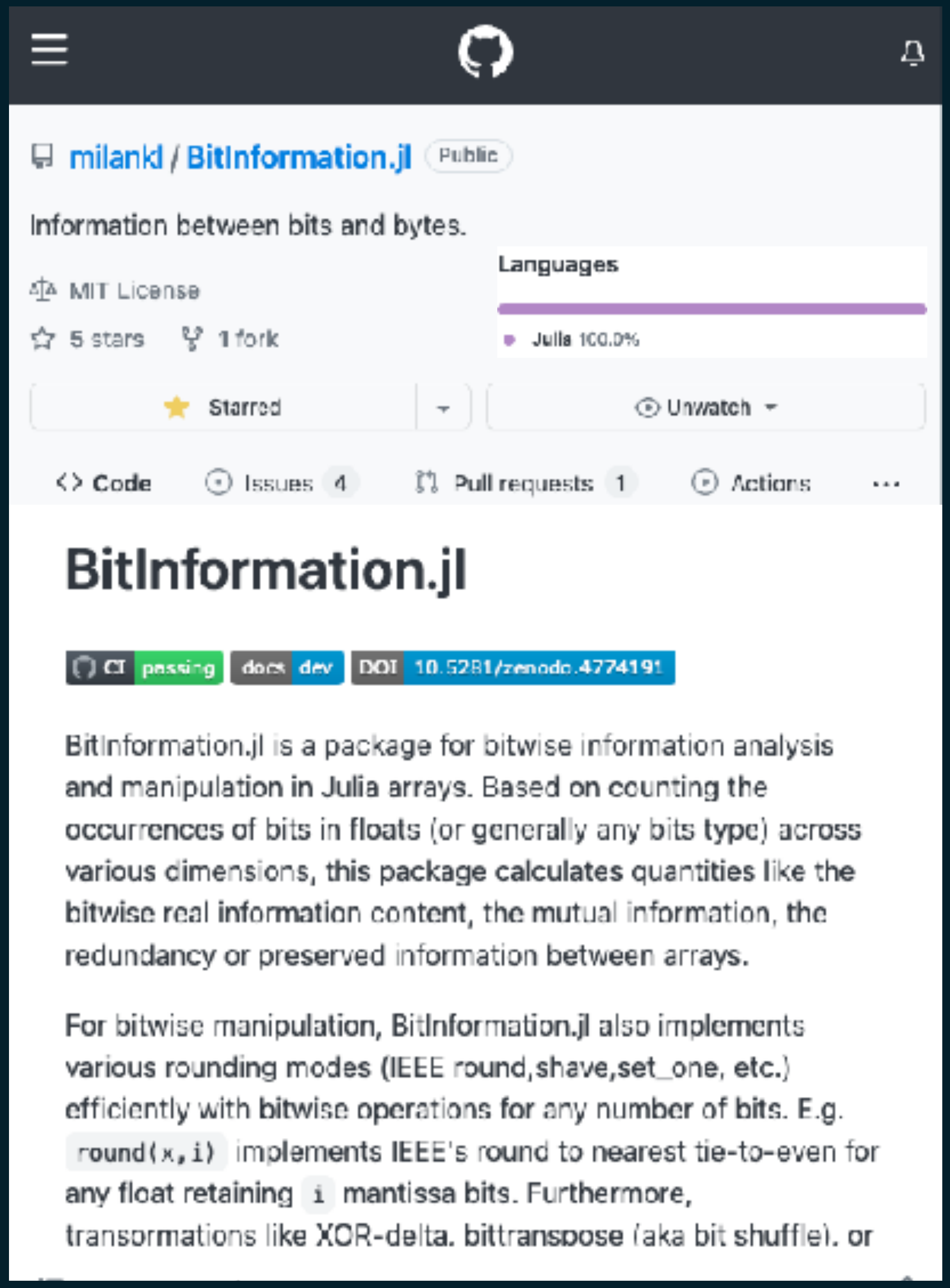
303.2500000

Visual artefacts

99% preserved information suggested as sweet spot

Implementations

Julia: BitInformation.jl



milankl / BitInformation.jl Public

Information between bits and bytes.

MIT License

5 stars 1 fork

Languages

Julia 100.0%

Starred

Unwatch

<> Code Issues 4 Pull requests 1 Actions ···

BitInformation.jl

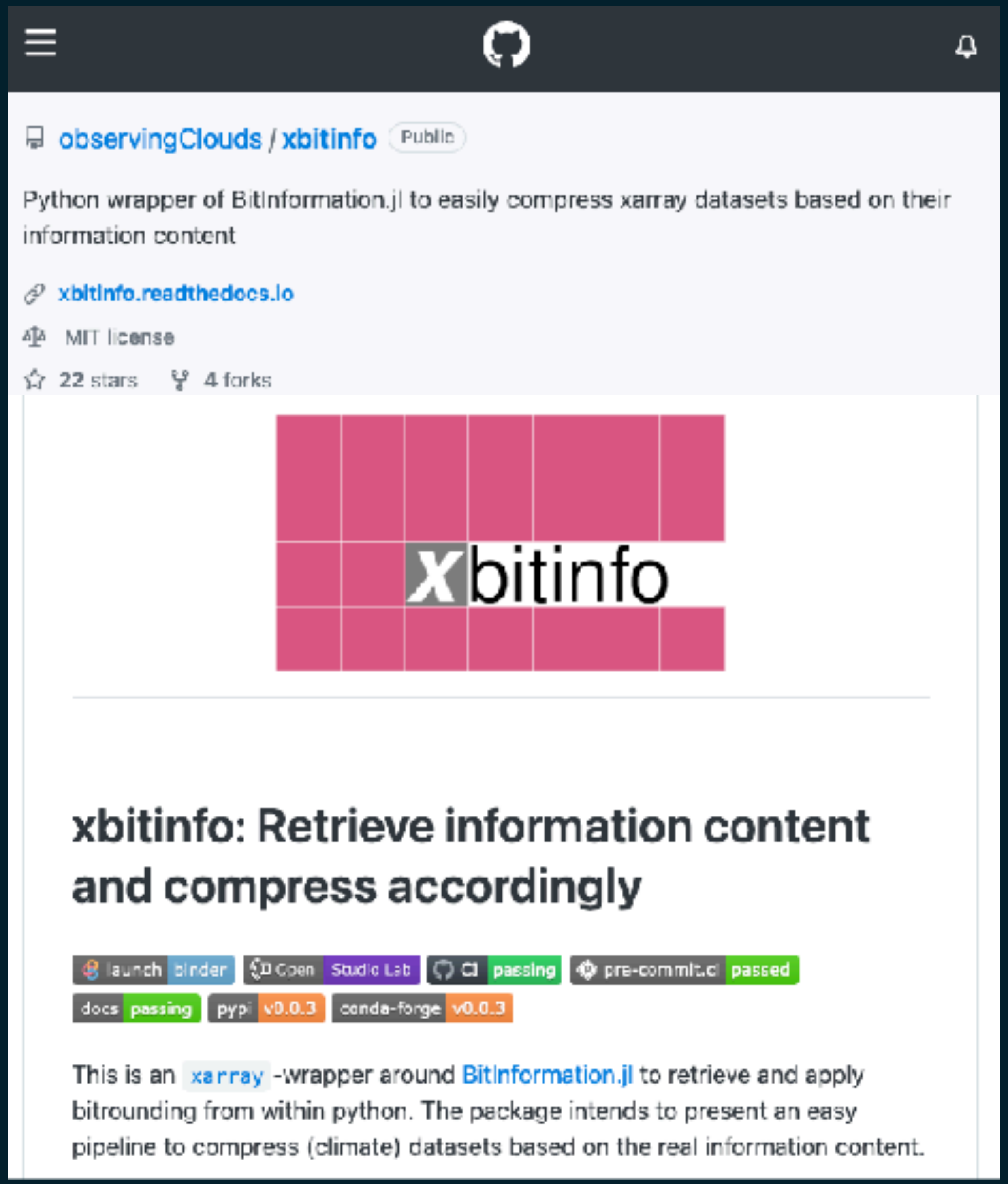
CI passing docs dev DOI 10.5281/zenodo.4774191

BitInformation.jl is a package for bitwise information analysis and manipulation in Julia arrays. Based on counting the occurrences of bits in floats (or generally any bits type) across various dimensions, this package calculates quantities like the bitwise real information content, the mutual information, the redundancy or preserved information between arrays.

For bitwise manipulation, BitInformation.jl also implements various rounding modes (IEEE round,shave,set_one, etc.) efficiently with bitwise operations for any number of bits. E.g. `round(x,i)` implements IEEE's round to nearest tie-to-even for any float retaining `i` mantissa bits. Furthermore, transormations like XOR-delta. bittranspose (aka bit shuffle). or

Milan Klöwer

Python & xarray: xbitinfo



observingClouds / xbitinfo Public

Python wrapper of BitInformation.jl to easily compress xarray datasets based on their information content

xbitinfo.readthedocs.io

MIT license

22 stars 4 forks

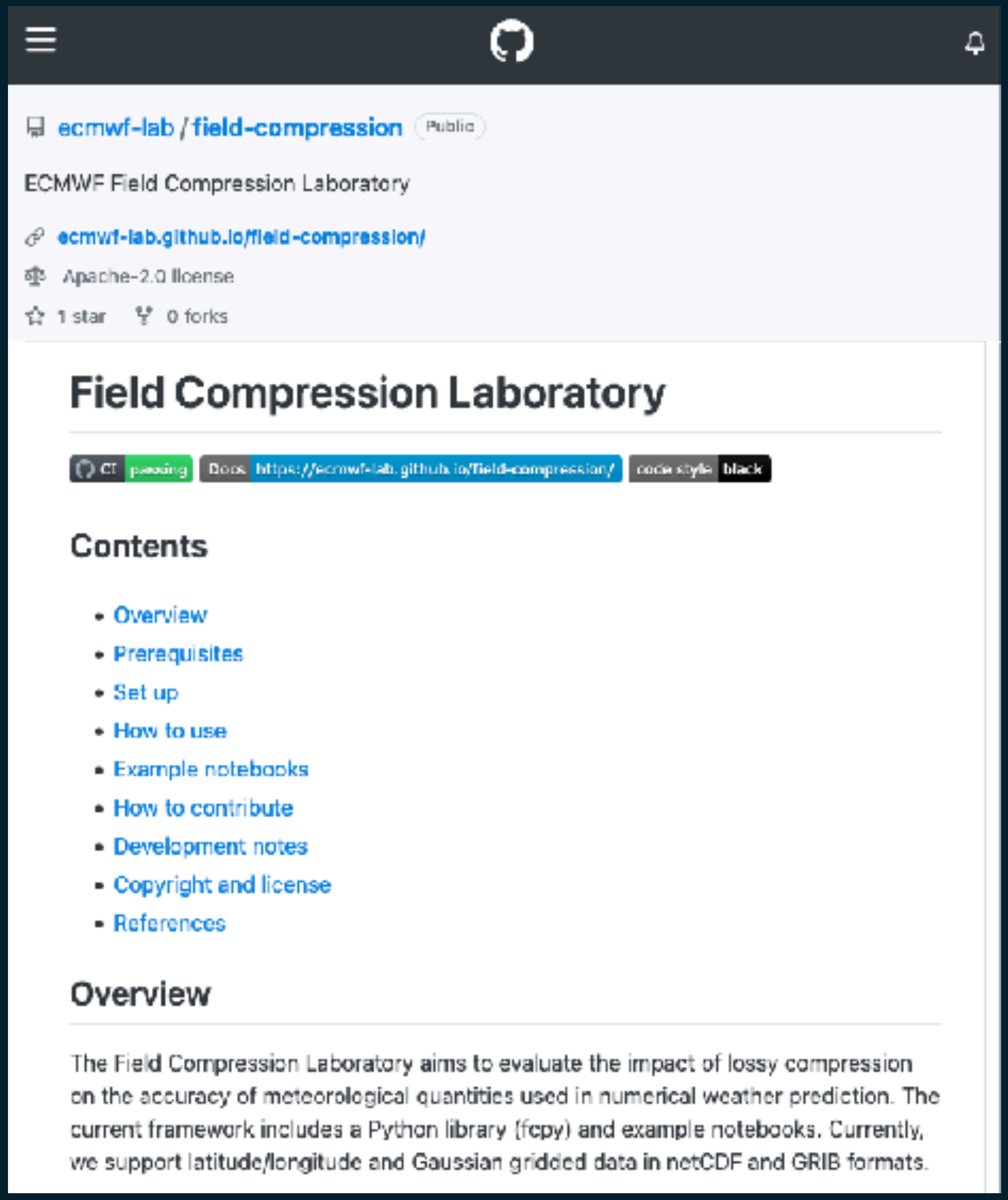
xbitinfo: Retrieve information content and compress accordingly

launch binder | Open Studio Lab | CI passing | pre-commit.ci passed
docs passing | pypi v0.0.3 | conda-forge v0.0.3

This is an `xarray` -wrapper around BitInformation.jl to retrieve and apply bitrounding from within python. The package intends to present an easy pipeline to compress (climate) datasets based on the real information content.

Schulz, Spring

Python, netCDF+GRIB



ecmwf-lab / field-compression Public

ECMWF Field Compression Laboratory

ecmwf-lab.github.io/field-compression/

Apache-2.0 license

1 star 0 forks

Field Compression Laboratory

CI passing | Docs https://ecmwf-lab.github.io/field-compression/ | code style black

Contents

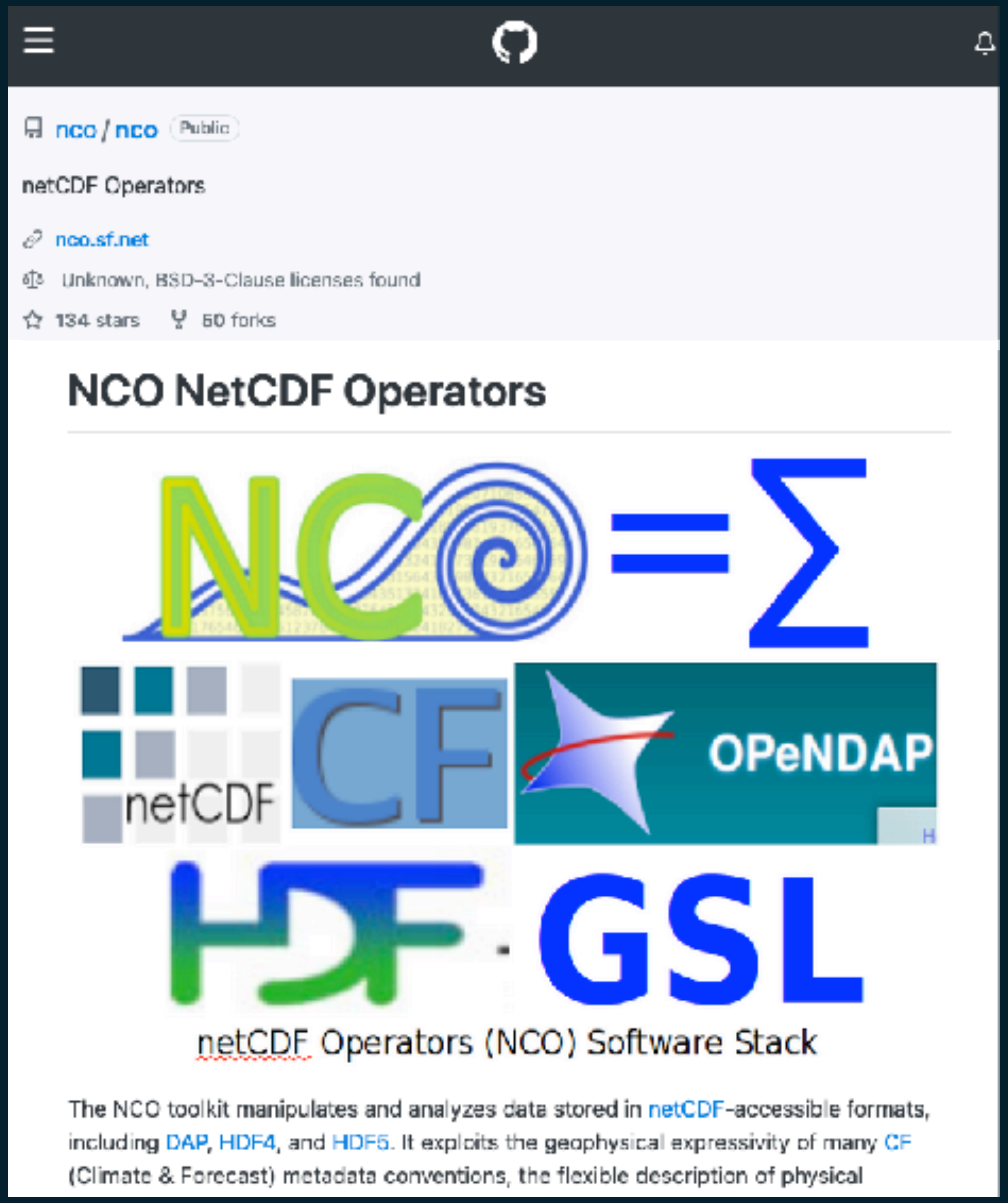
- Overview
- Prerequisites
- Set up
- How to use
- Example notebooks
- How to contribute
- Development notes
- Copyright and license
- References

Overview

The Field Compression Laboratory aims to evaluate the impact of lossy compression on the accuracy of meteorological quantities used in numerical weather prediction. The current framework includes a Python library (fcpy) and example notebooks. Currently, we support latitude/longitude and Gaussian gridded data in netCDF and GRIB formats.

David Meyer

netCDF+HDF: NCO



nco / nco Public

netCDF Operators

nco.sf.net

Unknown, BSD-3-Clause licenses found

134 stars 60 forks

NCO NetCDF Operators

netCDF Operators (NCO) Software Stack

The NCO toolkit manipulates and analyzes data stored in netCDF-accessible formats, including DAP, HDF4, and HDF5. It exploits the geophysical expressivity of many CF (Climate & Forecast) metadata conventions, the flexible description of physical

Zender et al

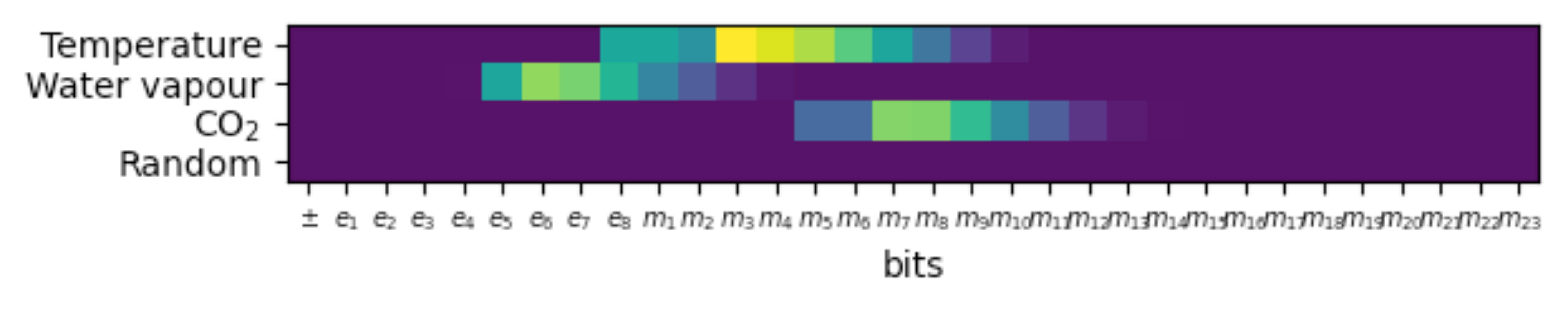
BitInformation.jl

1. Analyse bitinformation

```
julia> using BitInformation

julia> A = …

julia> bitinformation(A)
32-element Vector{Float64}:
 0.9999975605213736
 0.2667830274416382
 0.2934084155989065
 0.5817502292436738
 0.9331021403275142
 ⋮
```



2. Remove noise

```
julia> keepbits = 7

julia> A_round = round(A,keepbits)

julia> bitstring.(A_round)
10000000-element Vector{String}:
 "11000000101010110000000000000000"
 "11000000101000100000000000000000"
 "11000000100111110000000000000000"
 "11000000100110110000000000000000"
 "11000000100110110000000000000000"
 ⋮
```

3. Compress

```
julia> using JLD2

julia> jldopen("test.jld2","w"; compress=true) do f
           f["A"] = A_round
       end
```