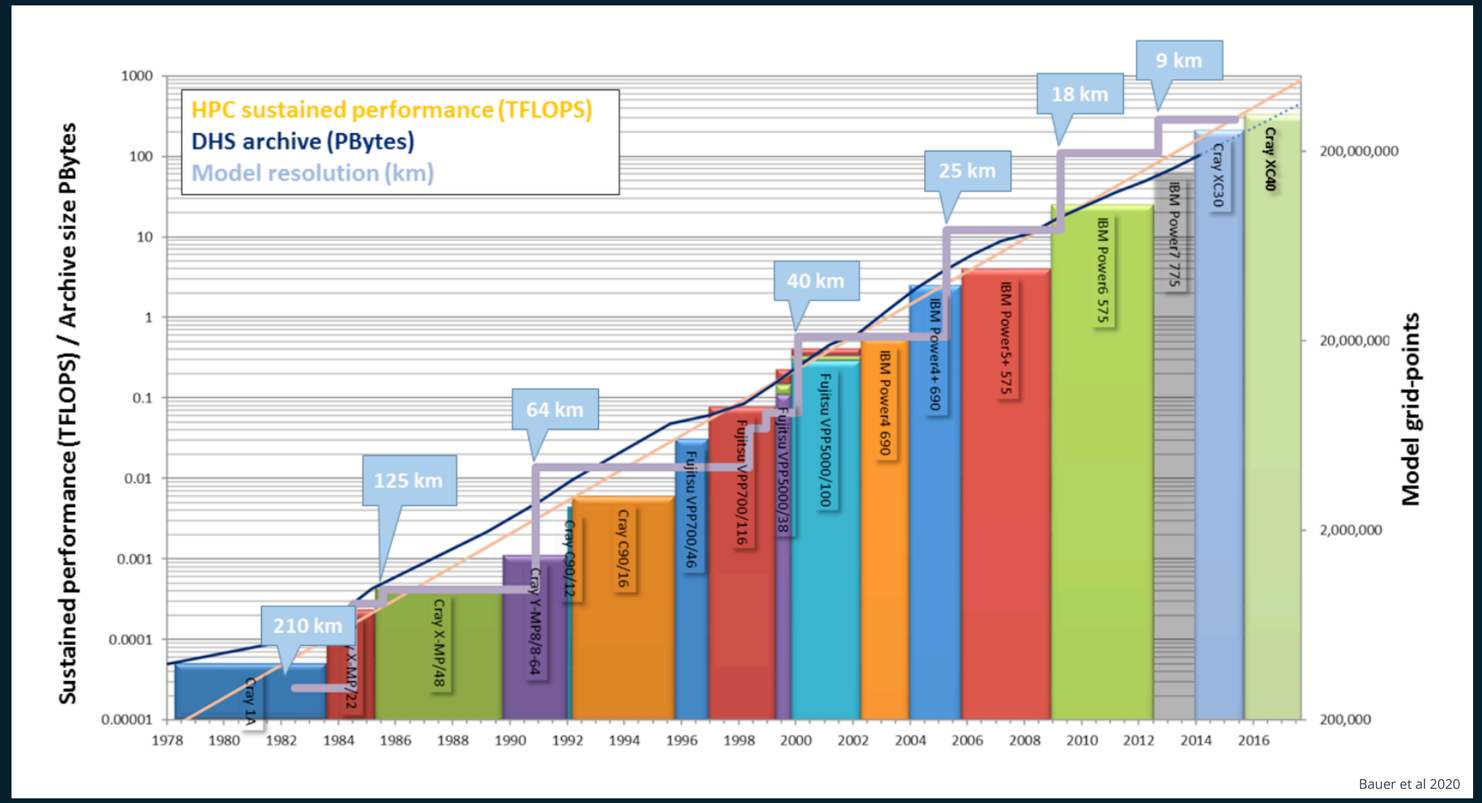
How much bitwise information is in weather and climate data?

Milan Klöwer[1], Miha Razinger[2], Juanjo Dominguez[2]
Aaron Spring[3], Hauke Schulz[3], Peter Düben[2], Tim Palmer[1]

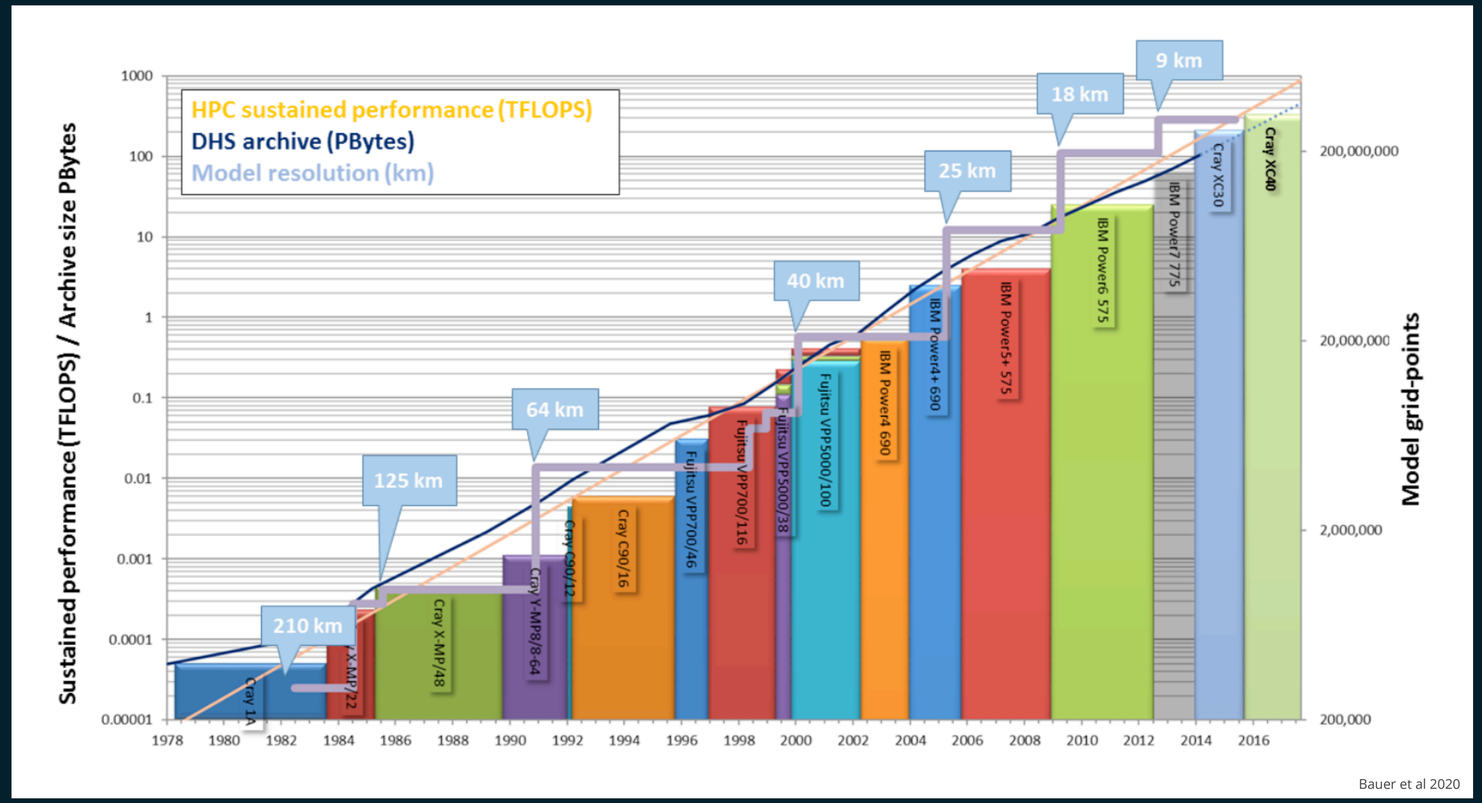
[1]University of Oxford, UK
[2]European Centre for Medium-Range Weather Forecasts, UK
[3]Max Planck Institute for Meteorology, Hamburg, Germany



Bauer et al 2020

40+ years of numerical weather prediction

- x10^6 increase in performance and data
- From 200km resolution to 9km
- Ensemble forecasts
- Towards “storm-resolving” models
- Often reliable 7-10 day forecasts

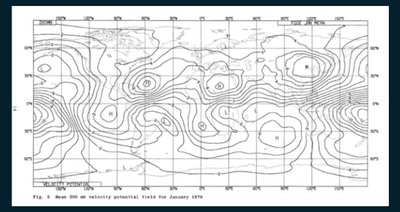


Bauer et al 2020

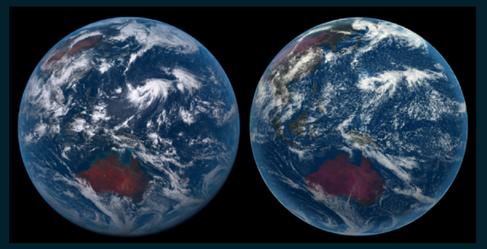
40+ years of numerical weather prediction

- x10^6 increase in performance and data
- From 200km resolution to 9km
- Ensemble forecasts
- Towards “storm-resolving” models
- Often reliable 7-10 day forecasts

ECMWF 1979



“Digital twins”, Stevens 2019



How much have we questioned the bitwise representation of our data?

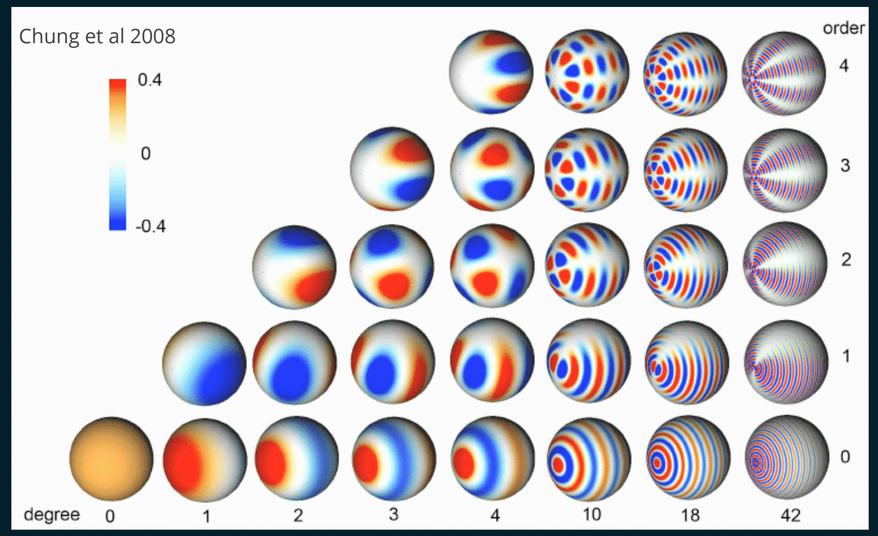
(computation and archive)

Schools of thoughts: Data compression

School 1: Transformations
(the physical perspective)

- Spectral
- EOF
- Tensor trains
- ...

- Spatial structure
- Expensive
- Error bounds difficult
- No random access

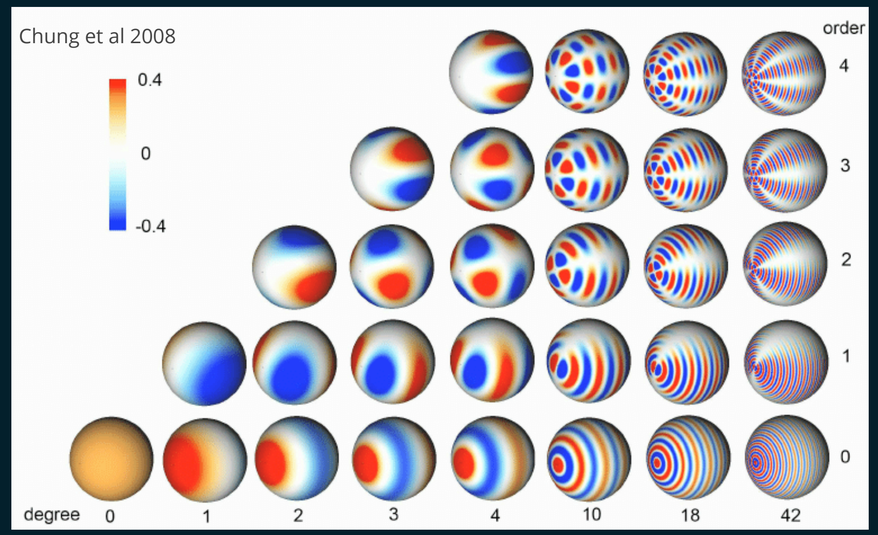


Schools of thoughts: Data compression

School 1: Transformations
(the physical perspective)

- Spectral
- EOF
- Tensor trains
- ...

- Spatial structure
- Expensive
- Error bounds difficult
- No random access



School 2: Precision and information theory
(the binary perspective)

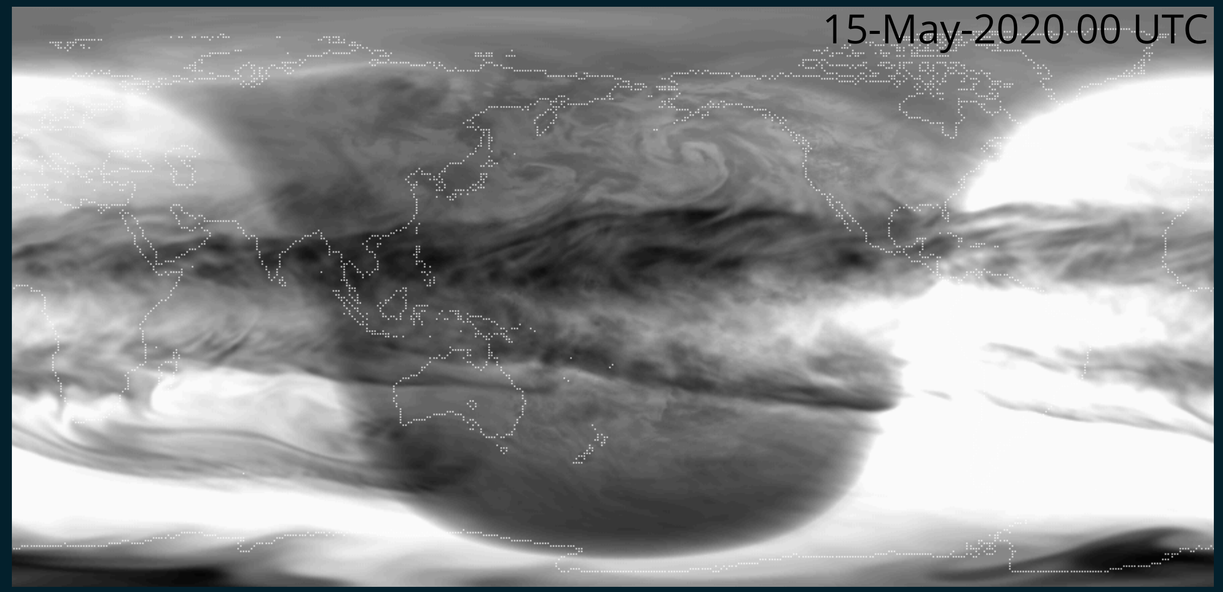
- Bitwise encoding
- Floats; linear or logarithmic quantisation
- Entropy coding
- Lossless compression

- Spatial structure
- Cheap transforms
- Rigid error bounds
- Random access

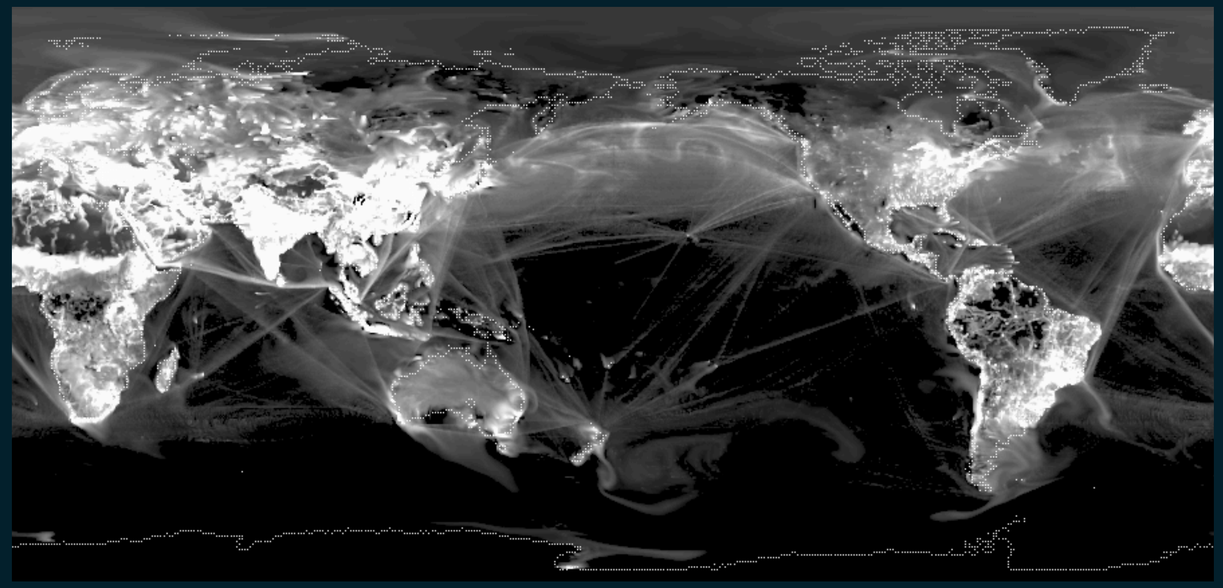
Copernicus Atmospheric Monitoring Service (CAMS)

60+ atmospheric variables,
including NO_2, O_3, SO_2, aerosols, ...

- 60+ atmospheric variables
- including NO_2, O_3, SO_2, aerosols, ...
- Varying spatial structures (physics, chemistry)

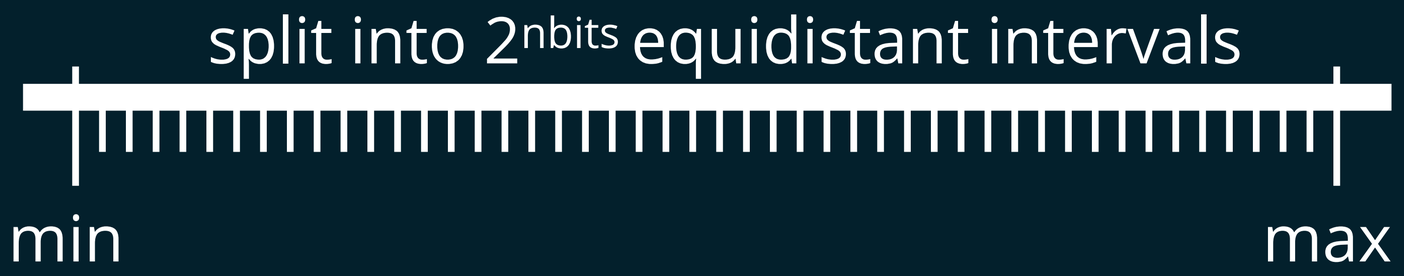


NO_2 in the stratosphere

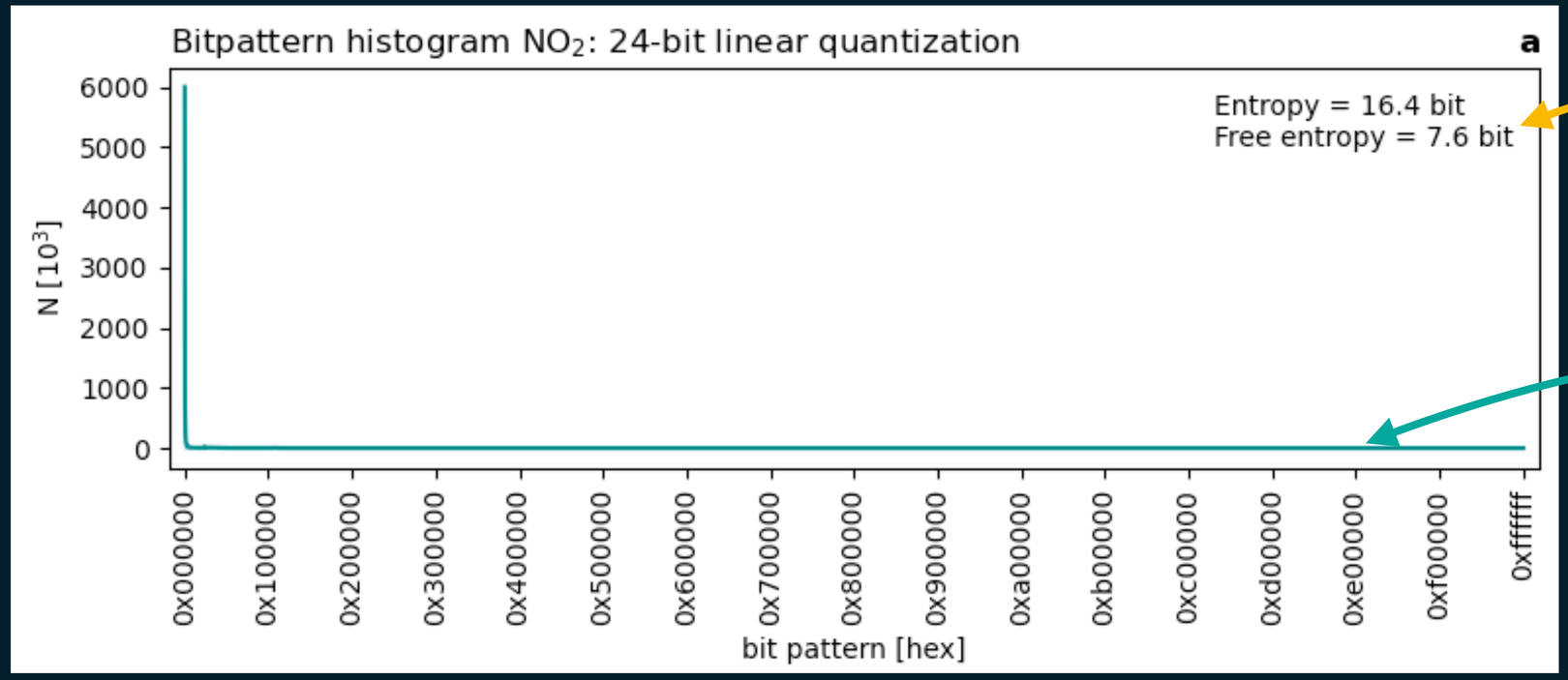


NO_2 at the surface

Current compression method: 24-bit linear quantization



Use nbits = 24 instead of 64 bit per number

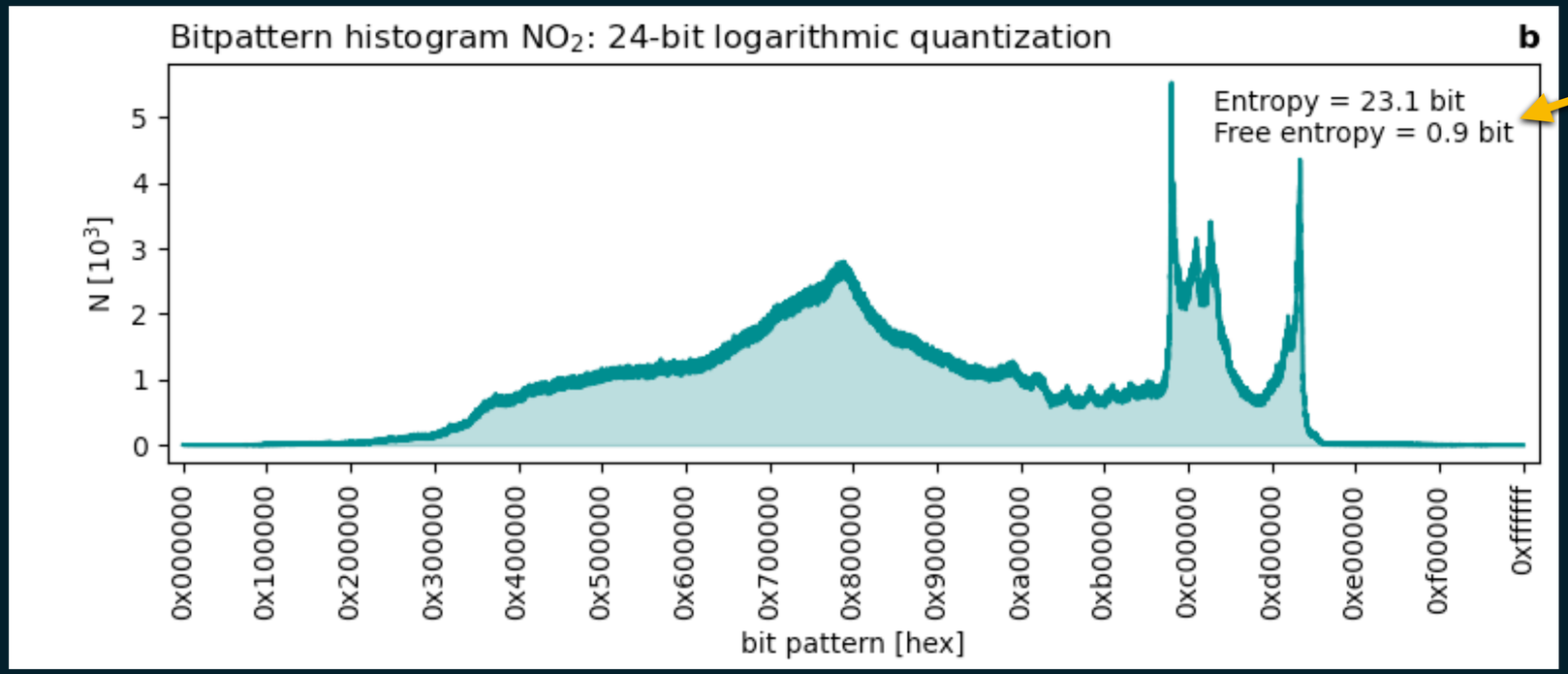


>7 bits are effectively unused

Most bit patterns very rarely used

How much information is in the first bit?
~0 bit. Information <= Entropy.

Alternative: Logarithmic quantization

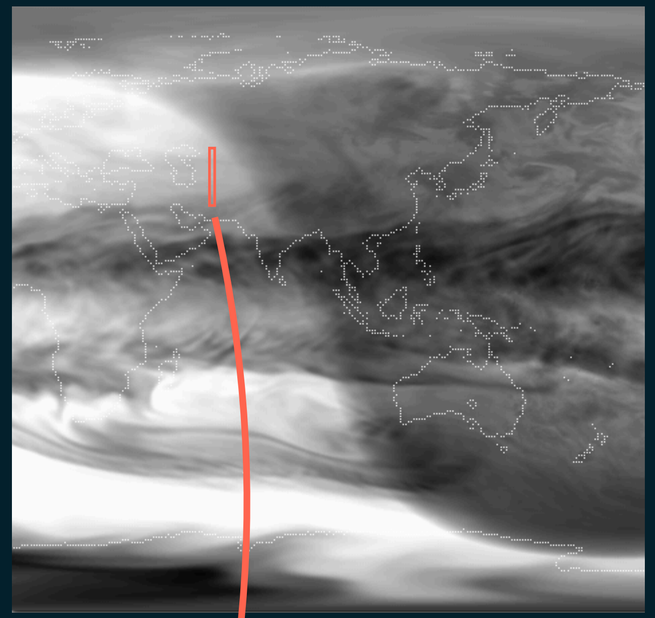


Most bits are effectively used

Floats are also approximately log-distributed

How much entropy is in the first bit?

~1bit



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

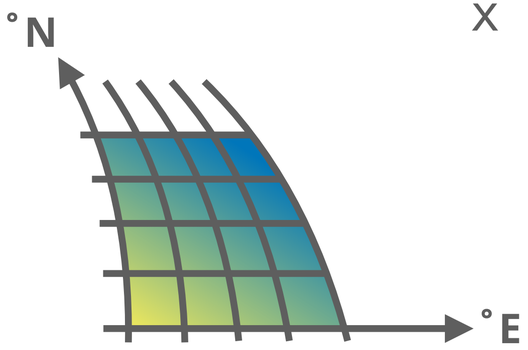
Real!? False!?

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

Bitwise real information content

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

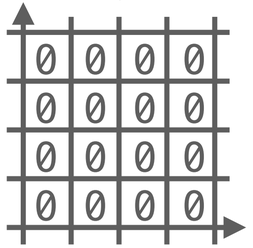
1 **Gridded data**



2 **Data as binary floating-point numbers**

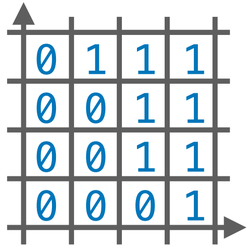
sign exponent mantissa bits

`x = 1.23… = 0 01111111 00110100100011001010001`



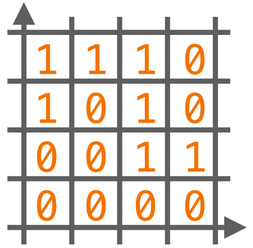
sign bits

…



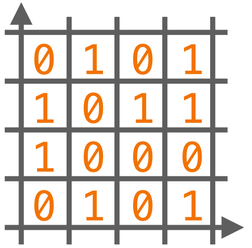
5th exponent bits

…



8th mantissa bits

…



last mantissa bits

3 **Analyse bits in adjacent grid points for every bit position**

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

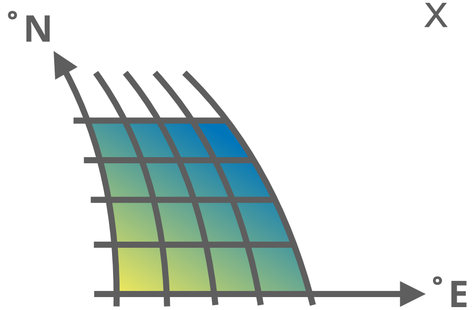
M > 0 bit

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

M = 0 bit

If adjacent bits are **independent**

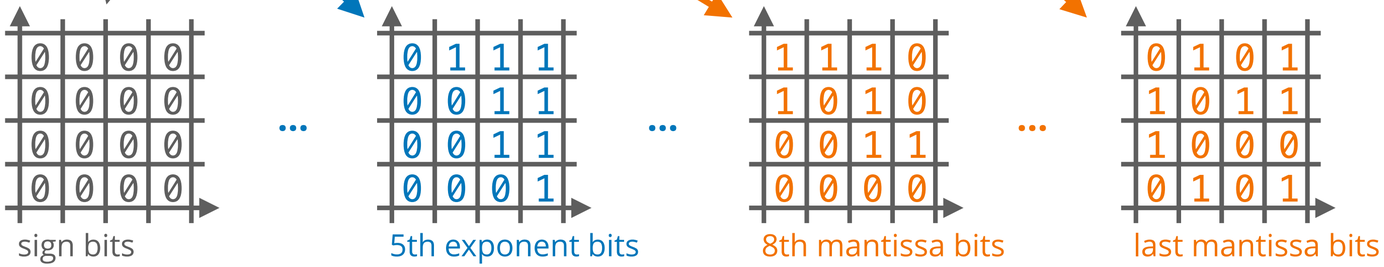
1 Gridded data



2 Data as binary floating-point numbers

sign exponent mantissa bits

x = 1.23… = 0 01111111 00110100100011001010001



sign bits | 5th exponent bits | 8th mantissa bits | last mantissa bits

3 Analyse bits in adjacent grid points for every bit position

4 The mutual information M between bits in adjacent grid points

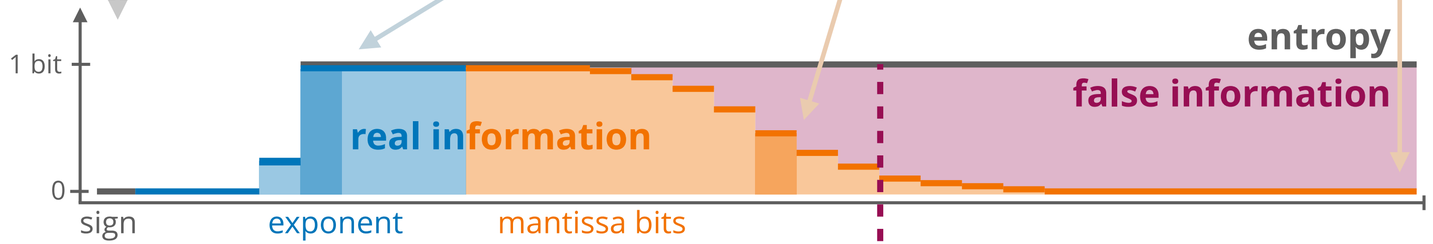
M = 0 bits
If all bits are **identical**

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**

5 Bitwise real information is the mutual information between bits



6 Rounding to remove the false information

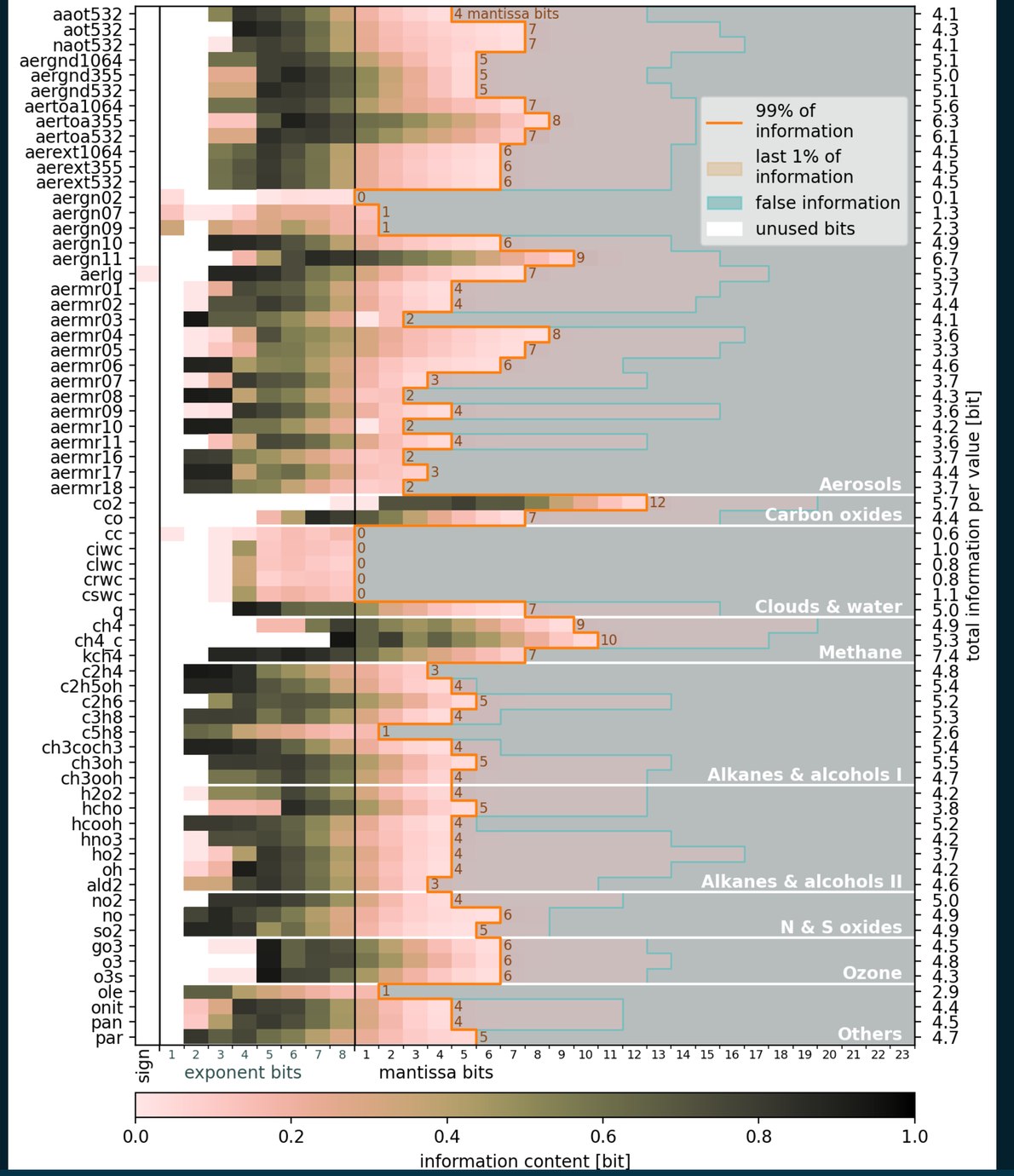
x ≈ 0 01111111 00110100100000000000000

Retain bits with **>99% of real information** in total

Remove false information by rounding trailing bits to 0

Bitwise real information content

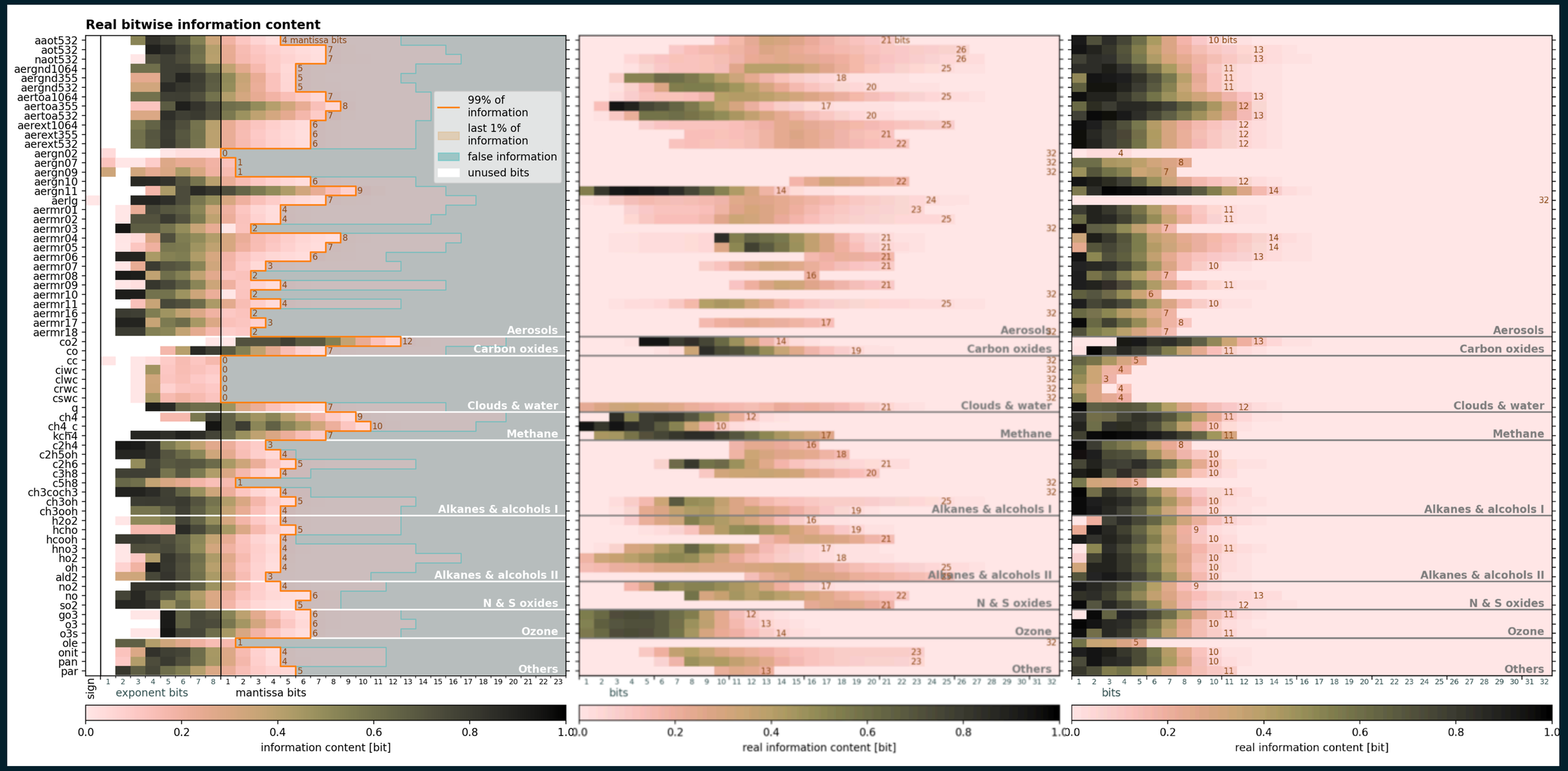
- Every variable requires a different precision
- Many bits do not contain real information
- Preserve only the bits with real information



Encoding: Floats

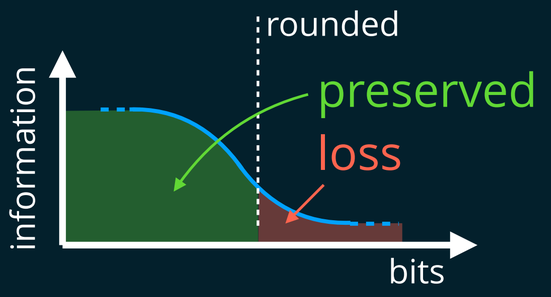
Linear quantisation

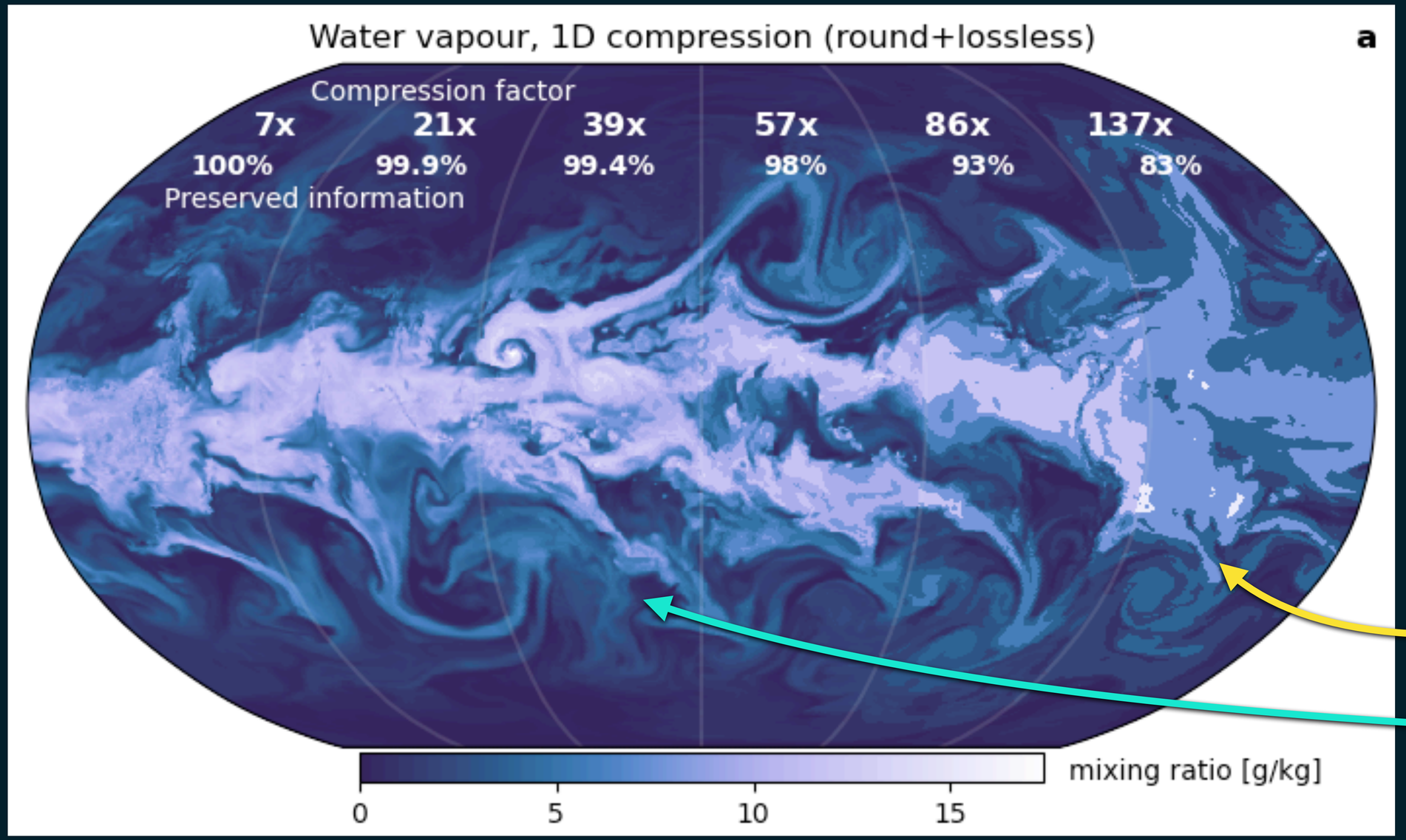
Logarithmic quantisation



- Many unused exponent bits, lossless compression essential
- Information is often spread across many bit positions
- Information is densely packed in the first bits

Information-preserving compression





Visual artefacts

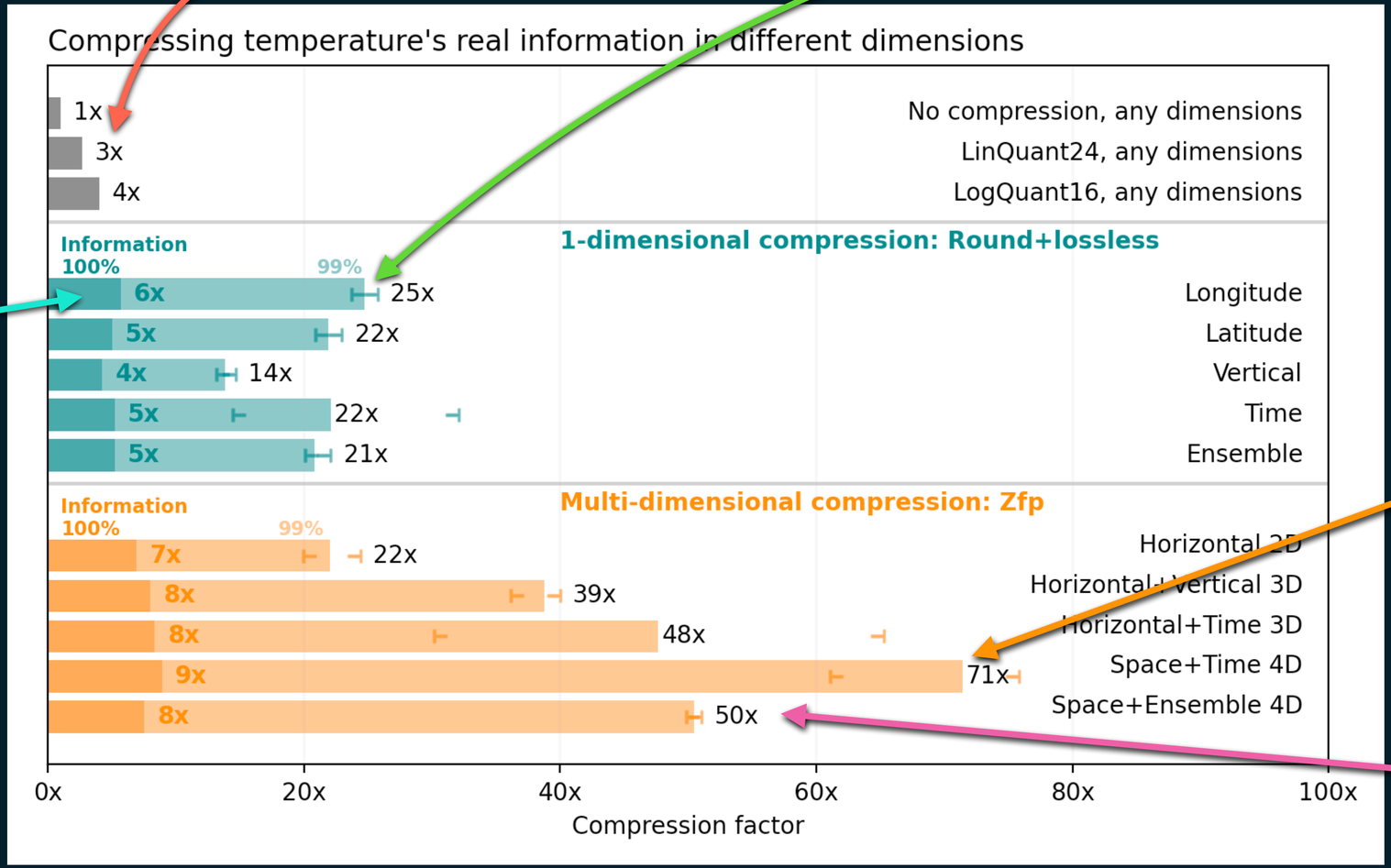
99% preserved information suggested as sweet spot

Multi-dimensional compression

Fixed bitrate

Compress along longitudes

Discarding 1% of information allows much higher compression

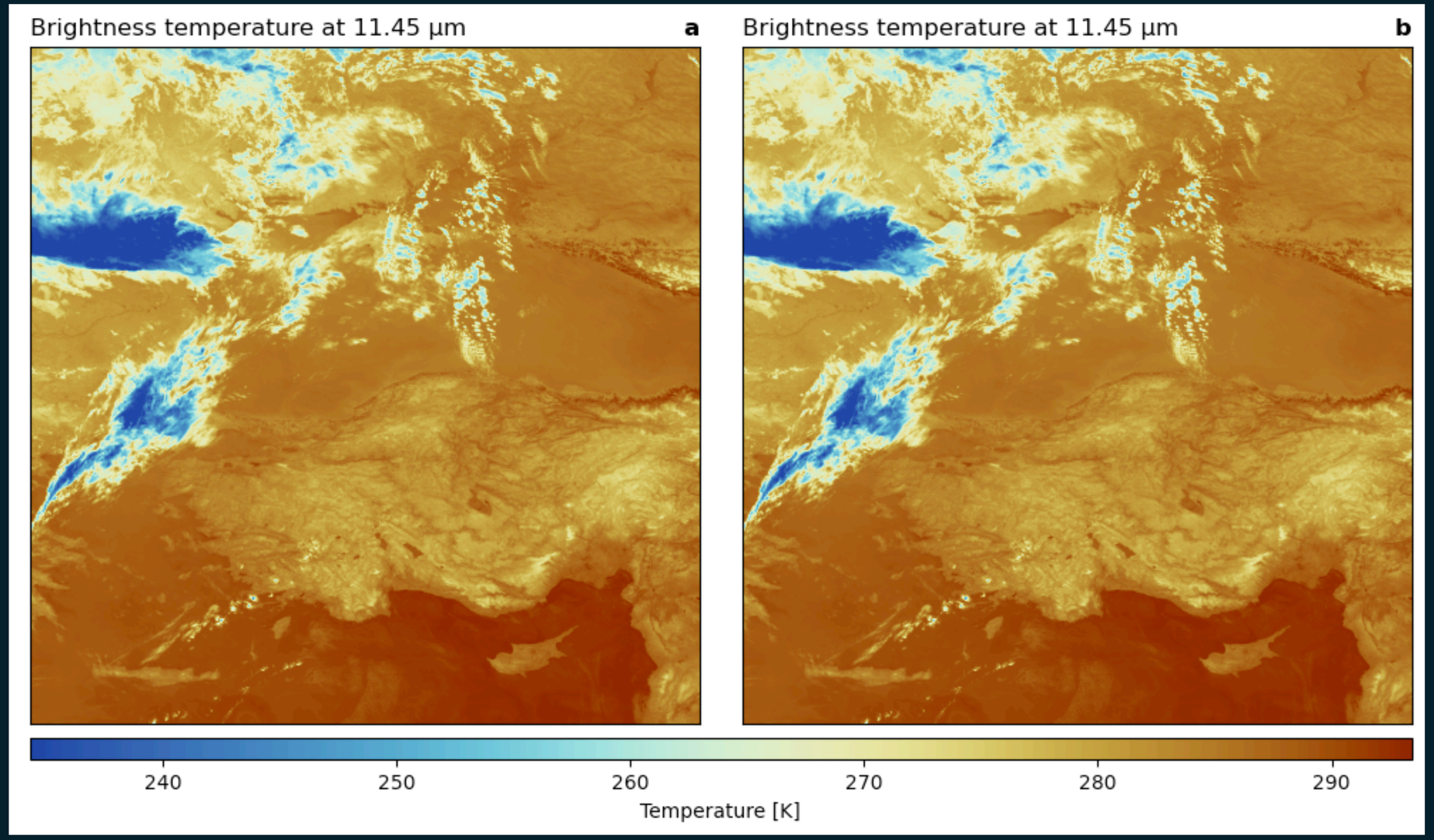


Space-time compression yields highest compression factors

Ensemble compression suffers from unsorted ensemble

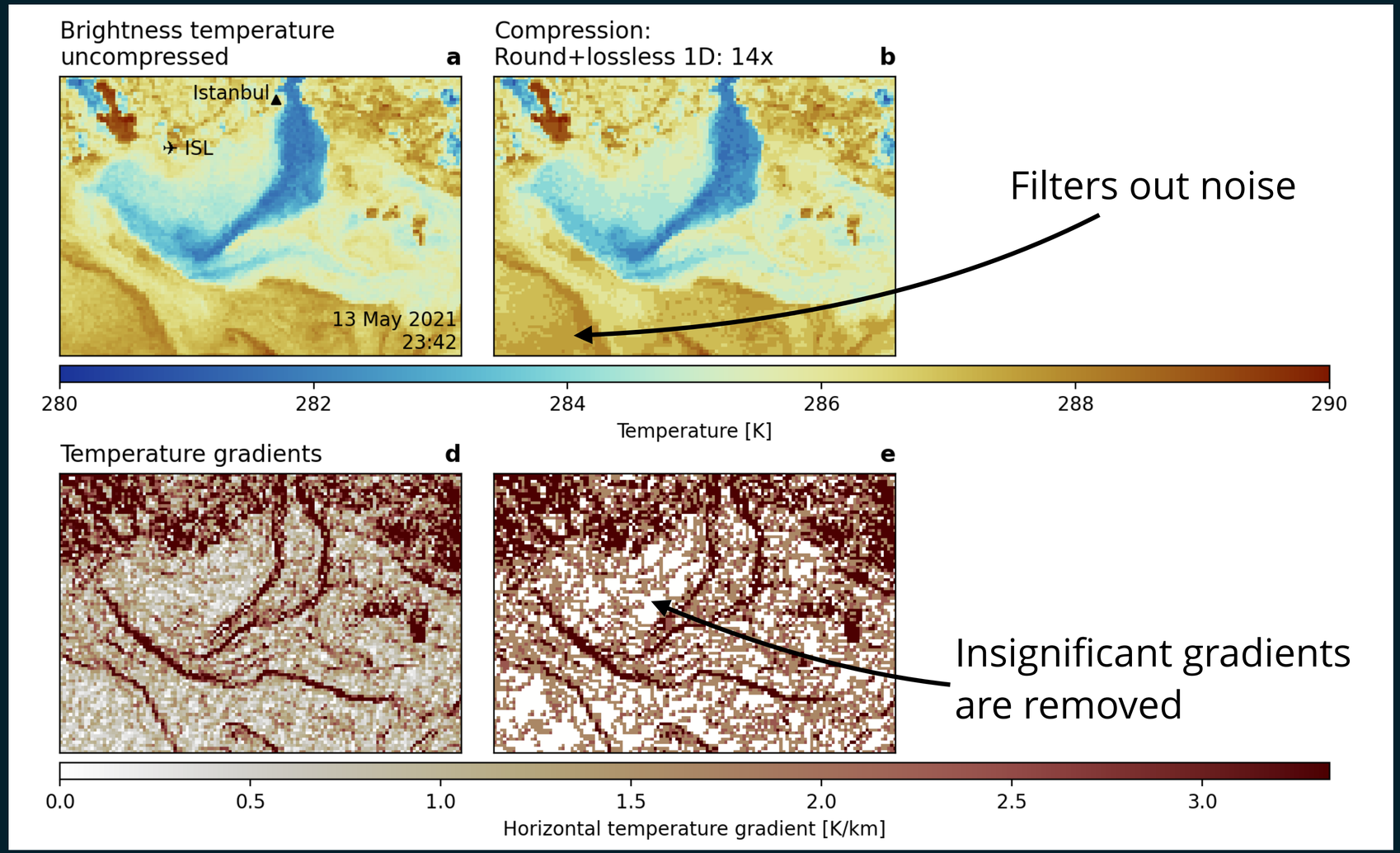
Information-preserving compression

One is 15-20x smaller than the other, which has been compressed?



Information-preserving compression

Filtering out insignificant gradients



Workflow

1. Determine the real bitwise information content

Round+lossless compression

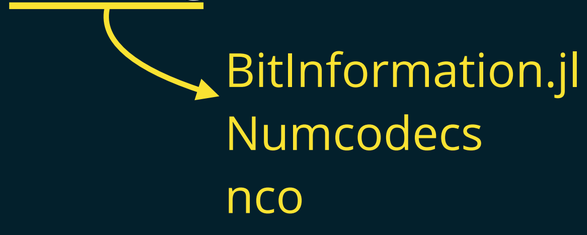
2. Remove insignificant bits via rounding
3. Lossless compression

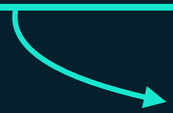
Workflow

1. Determine the real bitwise information content

Round+lossless compression

2. Remove insignificant bits via rounding
3. Lossless compression





Workflow

1. Determine the real bitwise information content
   - BitInformation.jl v0.5.1

Round+lossless compression

2. Remove insignificant bits via rounding
   - BitInformation.jl
   - Numcodecs
   - nco
3. Lossless compression
   - Zstandard, ...

Advantages

- Information analysis only once offline
- Rounding is very fast
- Any lossless codec can be used
- Flexibility size/performance
- Works with any bit encoding
- Error bounds from floats
- Flexible chunking/granularity possible

Workflow

1. Determine the real bitwise information content
   - BitInformation.jl v0.5.1

Round+lossless compression

2. Remove insignificant bits via rounding
   - BitInformation.jl
   - Numcodecs
   - nco
3. Lossless compression
   - Zstandard, ...

Advantages

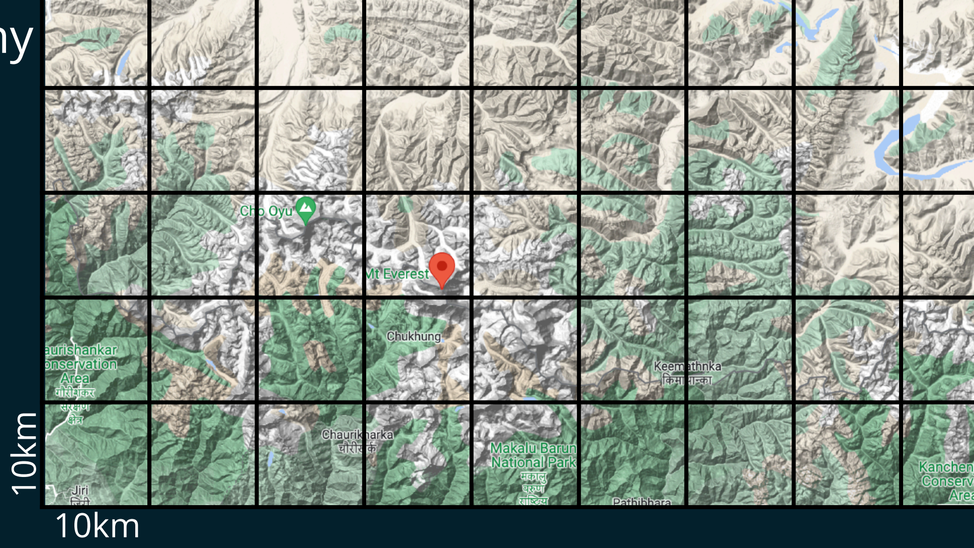
- Information analysis only once offline
- Rounding is very fast
- Any lossless codec can be used
- Flexibility size/performance
- Works with any bit encoding
- Error bounds from floats
- Flexible chunking/granularity possible

Disadvantages

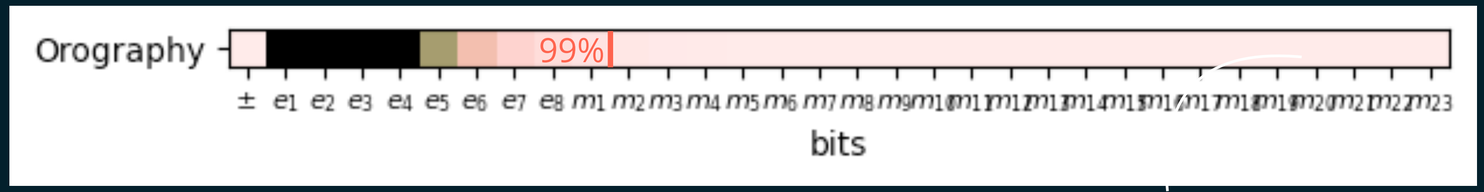
- Is real information well defined for any data?
- Mostly only 1D lossless compression

Julia workflow example

Orography

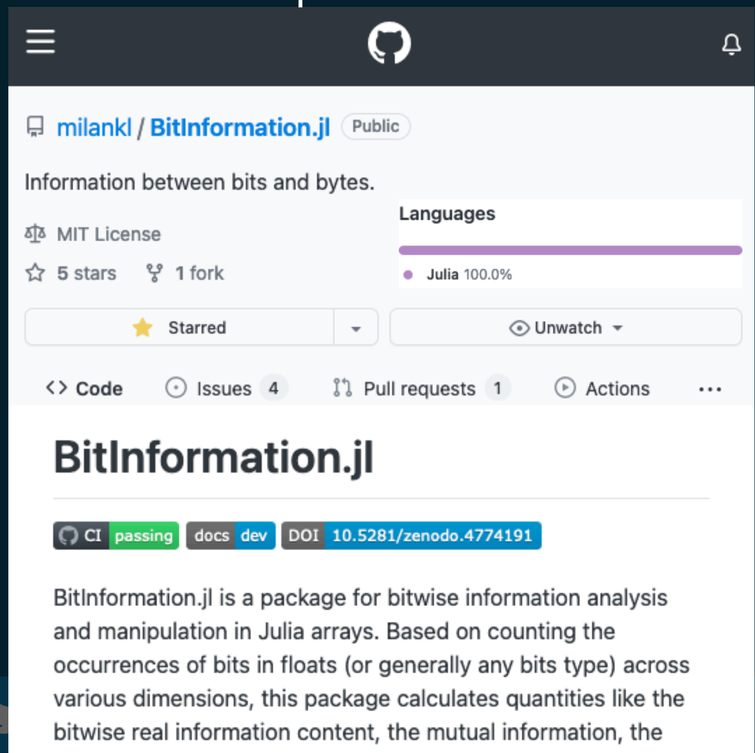



```
julia> using NetCDF, BitInformation

julia> ncfile = NetCDF.open("orography.nc")

julia> orog = ncfile.vars["orog"][:,:]

julia> bi = bitinformation(orog)
```




```
julia> round(8848.86,1)
8192.0

julia> round(60.0,1)
64.0
```

Reference implementation



Using NCO

```
> ncks -L 9 --baa=8 --ppc orog=1 orography.nc orography_round.nc
```

8MB 1.2MB

Python workflow example

github.com/observingClouds/xbitinfo
xarray-wrapper around BitInformation.jl

A Spring & H Schulz, MPI

How to use

```
import xarray as xr
import xbitinfo as xb
ds = xr.tutorial.load_dataset(inpath)
bitinfo = xb.get_bitinformation(ds, dim="lon")  # calling bitinformation.j
keepbits = xb.get_keepbits(bitinfo, inflevel=0.99)  # get number of mantis
ds_bitrounded = xb.xr_bitround(ds, keepbits)  # bitrounding keeping only k
ds_bitrounded.to_compressed_netcdf(outpath)  # save to netcdf with compres
```

observingClouds / xbitinfo Public

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

xbitinfo.readthedocs.io

MIT license

6 stars 0 forks

Languages

Python 97.9% Julia 2.1%

Starred Watch

Code Issues 7 Pull requests Discussions Actions Projects

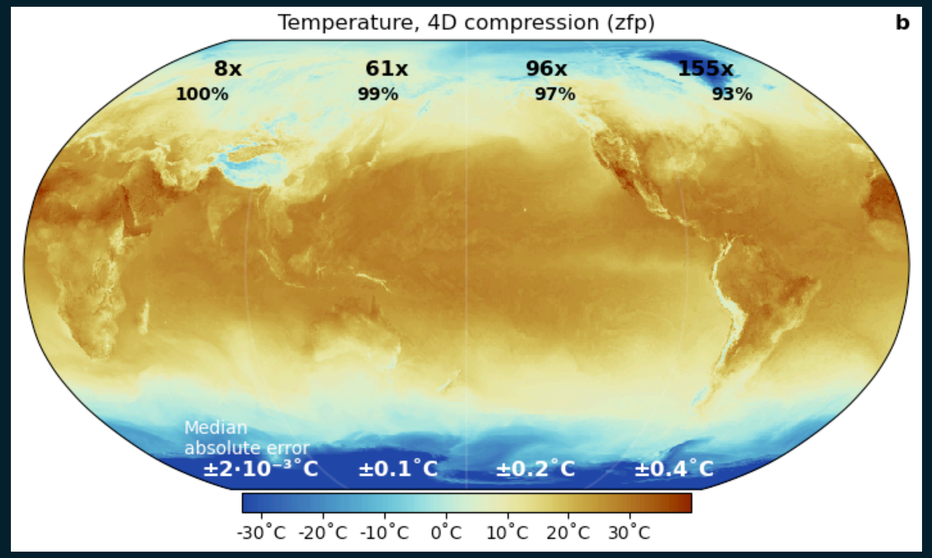
README.md

xbitinfo: Retrieve information content and compress accordingly

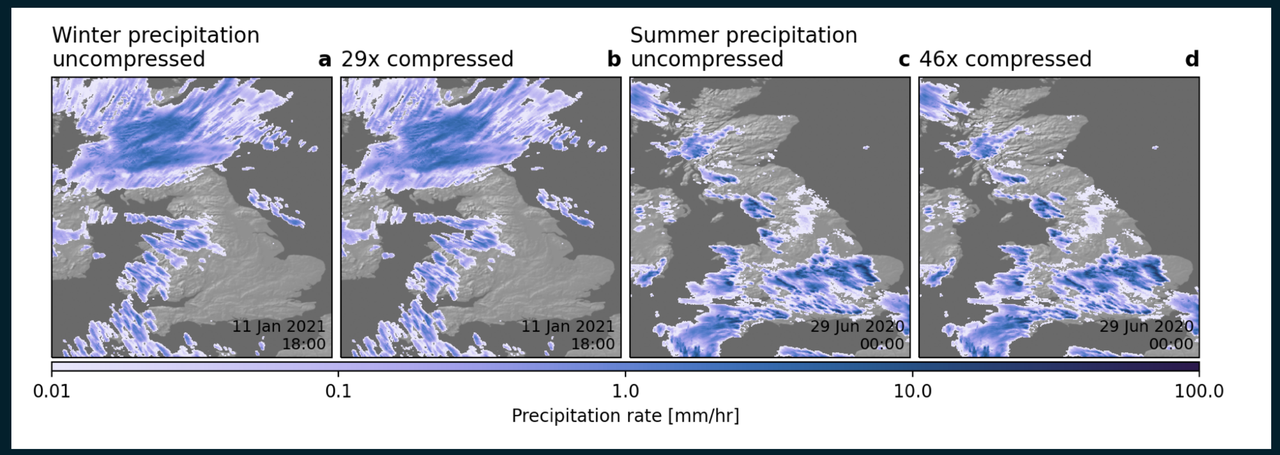
launch binder CI passing pre-commit.ci passed docs passing pypi package or version not fou

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

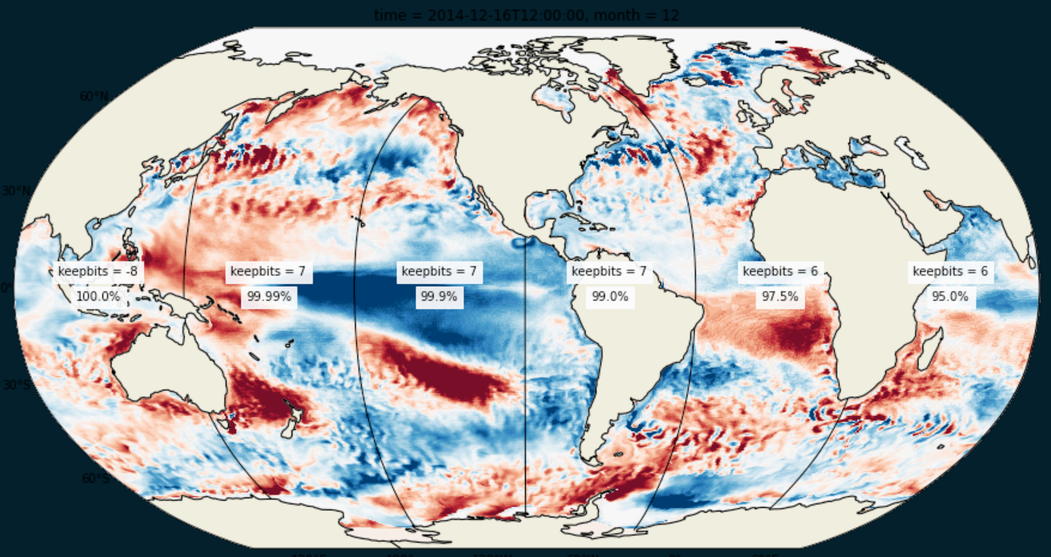
Application examples



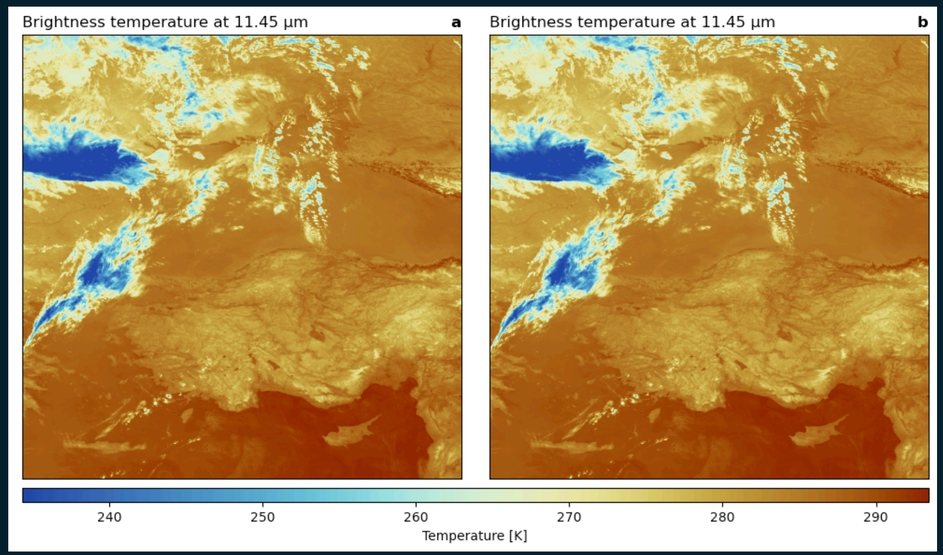
ECMWF's ensemble temperature forecast



Precipitation from radar data



CMPI6 sea surface temperatures
and ICON-Ocean model output



Brightness temperature from satellites