hiopy

Optimizing Model Output for Analysis

Tobias Kölling

Parallel IO NHR Workshop on 07.05.2024

MAX-PLANCK-INSTITUT
FÜR METEOROLOGIE

example: NextGEMS Simulations

- Model resolution: 5km
- Typical output size: **1 PB**
- Typical simulation time: **3 Weeks**

600 MB / s


1TB
DataTraveler® Max
1000MB/s Read
900MB/s Write
1000Mo/s en lecture
900Mo/s en écriture
1000MB/s in lettura
900MB/s in scrittura
USB 3.2 GEN 2
USB TYPE-C
Kingston
Kingston TECHNOLOGY

high-resolution model output can be slow

the time it takes until the analysis plot is ready

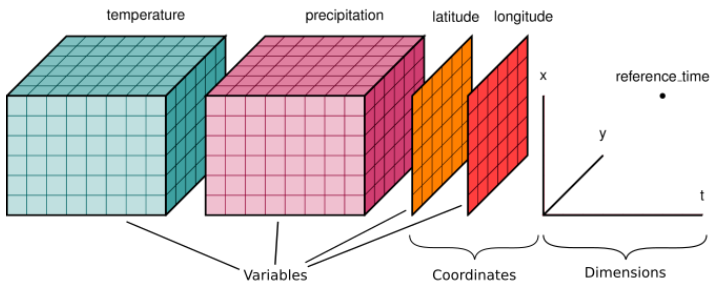
- **understanding** the data
- **coding** the analysis
- **getting** the data

Useful output is written once and read **at least** once.

*optimize output for **analysis***

(not write throughput)

(for this talk)



- n-dimensional variables
- shared dimensions
- coordinates
- attributes for metadata

- a single file
- a storage format
- shaped by storage & handling

```
$ ls *.nc
ngc2009_atm_mon_20200329T000000Z.nc
ngc2009_oce_2d_1h_inst_20200329T000000Z.nc
ngc2009_atm_pl_6h_inst_20200329T000000Z.nc
ngc2009_lnd_tl_6h_inst_20200329T000000Z.nc
ngc2009_lnd_2d_30min_inst_20200329T000000Z.nc
ngc2009_atm_2d_30min_inst_20200329T000000Z.nc
ngc2009_oce_0-200m_3h_inst_1_20210329T000000Z.nc
ngc2009_oce_0-200m_3h_inst_2_20210329T000000Z.nc
ngc2009_oce_moc_1d_mean_20210329T000000Z.nc
ngc2009_oce_2d_1d_mean_20210329T000000Z.nc
ngc2009_oce_ml_1d_mean_20210329T000000Z.nc
ngc2009_oce_2d_1h_mean_20210329T000000Z.nc
...
$ ls *.nc | wc -l
  12695
```

- provides an easy-to-understand overview
- forces consistency across output
- cutting things is easier than glueing things

```
1 ds = cat.ICON.ngc4008.to_dask()
```

xarray.Dataset

▸ Dimensions: (**time**: 10958, **depth_half**: 73, cell: 49152, **level_full**: 90, **crs**: 1, **depth_full**: 72, **soil_depth_water_level**: 5, **level_half**: 91, **soil_depth_energy_level**: 5)

▾ Coordinates:

crs	(crs)	float32	nan
depth_full	(depth_full)	float32	1.0 3.1 ... 5.546e+03 ...
depth_half	(depth_half)	float32	0.0 2.0 4.2 ... 5.681e...
level_full	(level_full)	int32	1 2 3 4 5 6 7 ... 85 8...
level_half	(level_half)	int32	1 2 3 4 5 6 7 ... 86 8...
soil_depth_en...	(soil_depth_energy_level)	float32	0.0325 0.192 0.7755...
soil_depth_wa...	(soil_depth_water_level)	float32	0.0325 0.192 0.7755...
time	(time)	datetime64[ns]	2020-01-02 ... 2050...

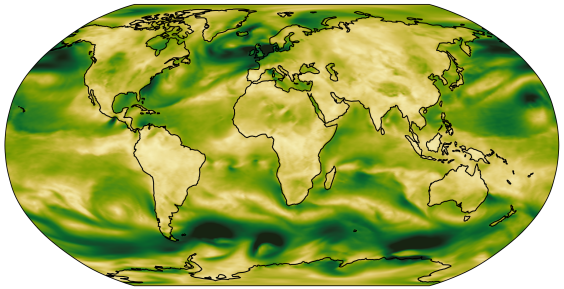
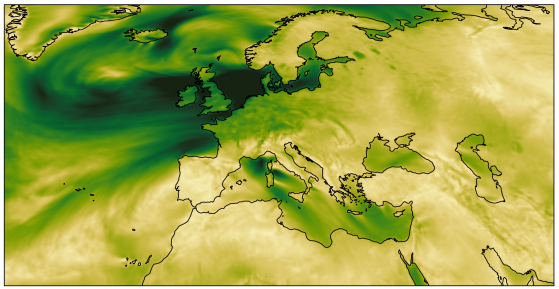
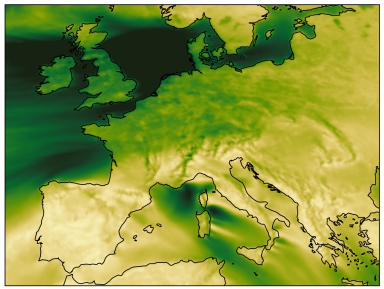
▾ Data variables:

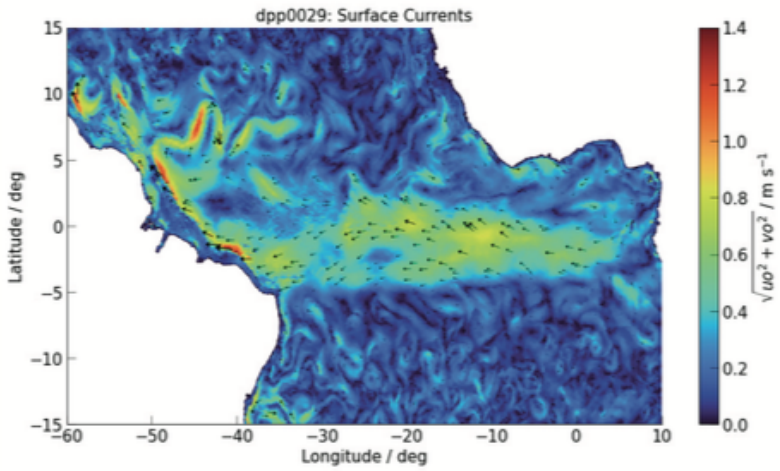
A_tracer_v_to	(time, depth_half, cell)	float32	...
FrshFlux_IceSalt	(time, cell)	float32	...
FrshFlux_TotalIce	(time, cell)	float32	...
Qbot	(time, cell)	float32	...
Qtop	(time, cell)	float32	...
Wind_Speed_1...	(time, cell)	float32	...
atmos_fluxes_F...	(time, cell)	float32	...
atmos_fluxes_F...	(time, cell)	float32	...
atmos_fluxes_F...	(time, cell)	float32	...
atmos_fluxes_F...	(time, cell)	float32	...
atmos_fluxes_...	(time, cell)	float32	...
atmos_fluxes_...	(time, cell)	float32	...
atmos_fluxes_...	(time, cell)	float32	...

Grid	Cells
1° by 1°	0.06M
10 km	5.1M
5 km	20M
1 km	510M
200 m	12750M

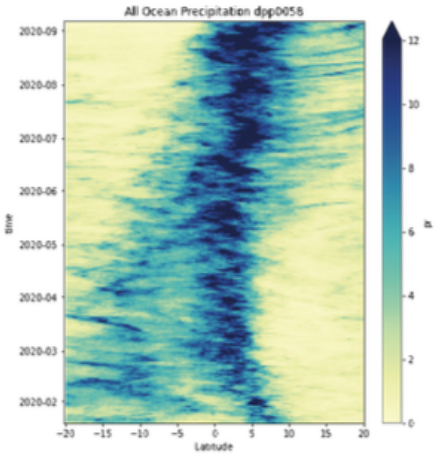
Screen	Pixels
VGA	0.3M
Full HD	2.1M
MacBook 13’	4.1M
4K	8.8M
8K	35.4M

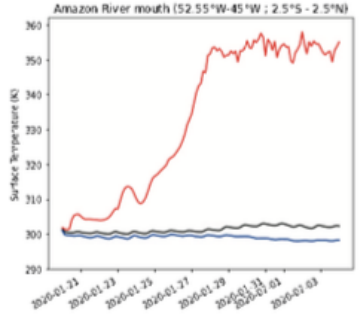
It’s **impossible** to look at the entire globe in full resolution.



5%, $1/4^2$ -> **0.3%**





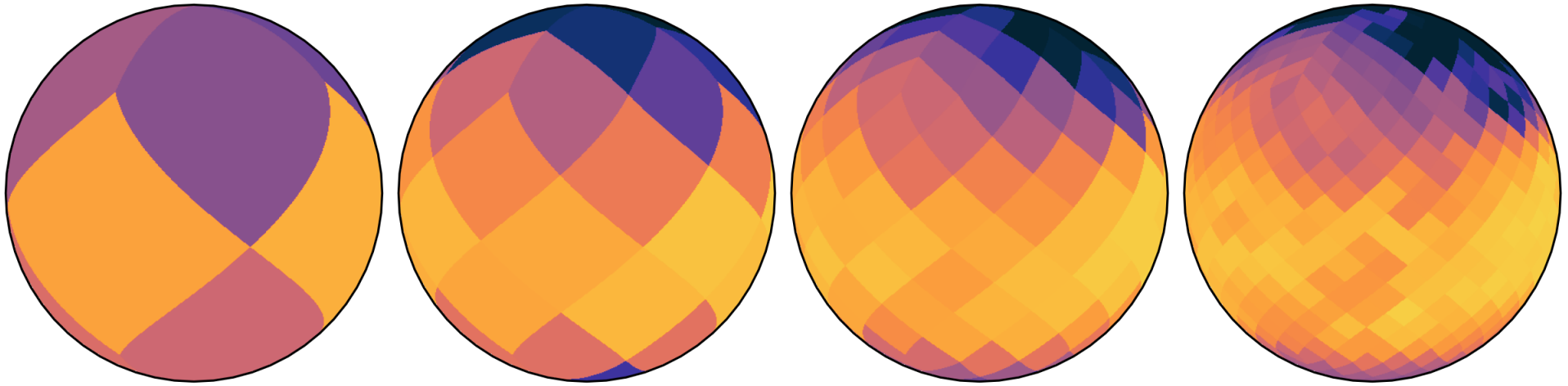
0.1%, 1^2 -> **0.1%**

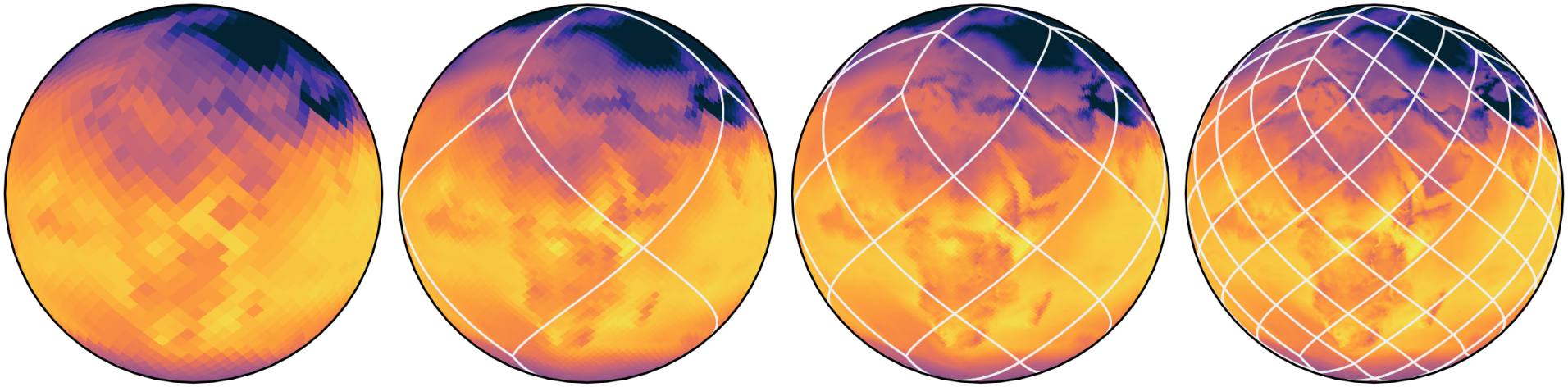
34%, $1/4^2$ -> **2%**

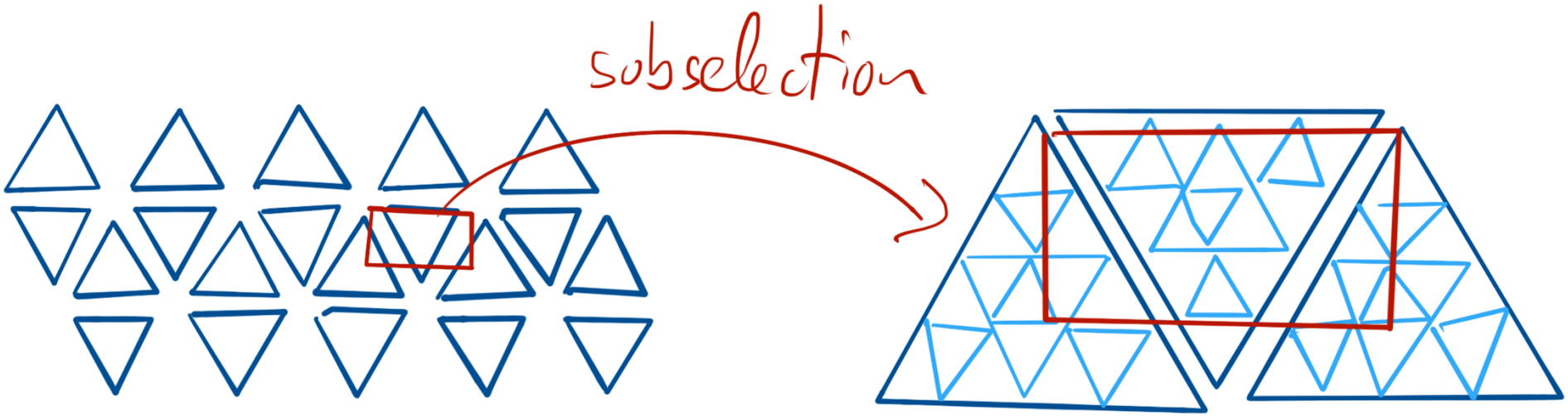
useful data
required resolution
fraction of globe covered

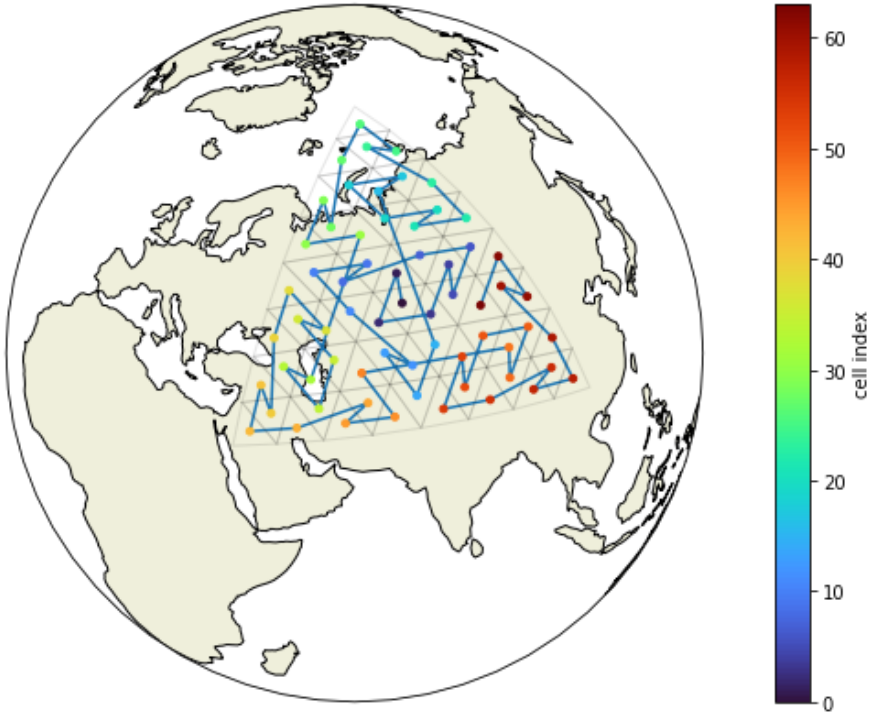
Plots by Marius Winkler

Analysis scripts are **forced** to load way too much data.



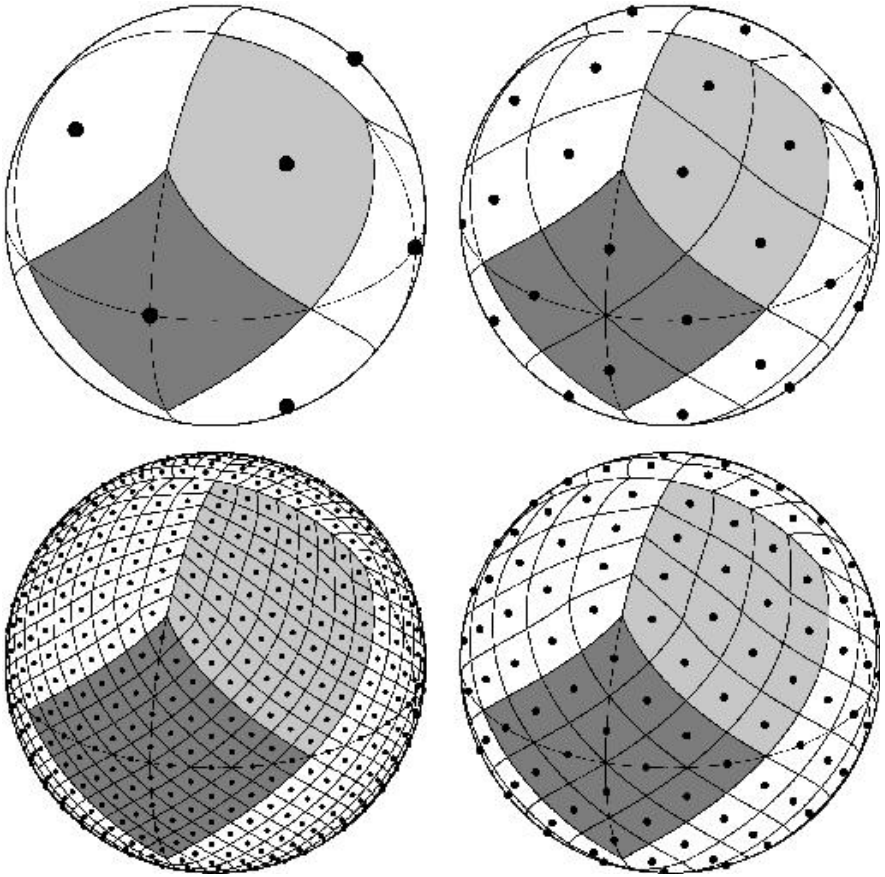


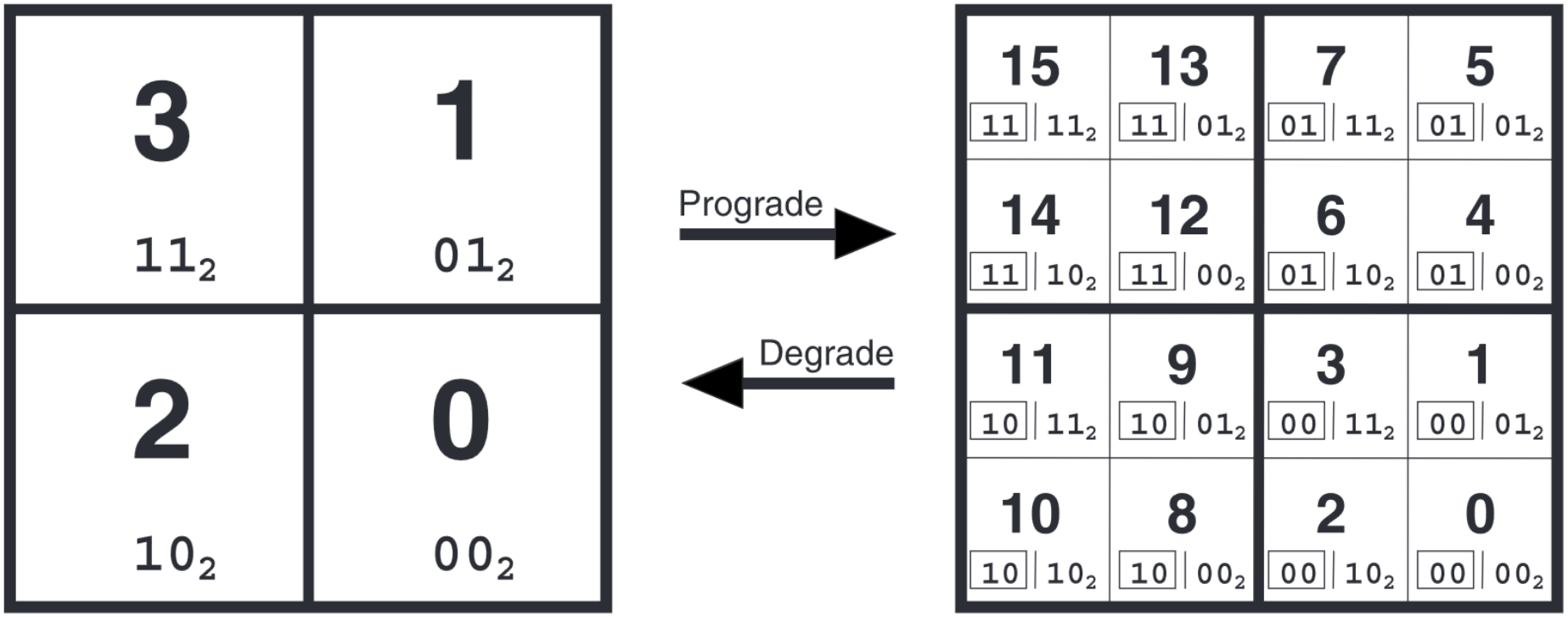

subselection



- spatially close ≈ close in index space
- recursive pattern (repeats every 4^n)

- hierarchical
- equal area
- iso latitude





aggregation can be exact and is easy to implement

scale analysis with screen size

(instead of with model size)

Select ICON model output at all dropsonde locations during EUREC4A field campaign:

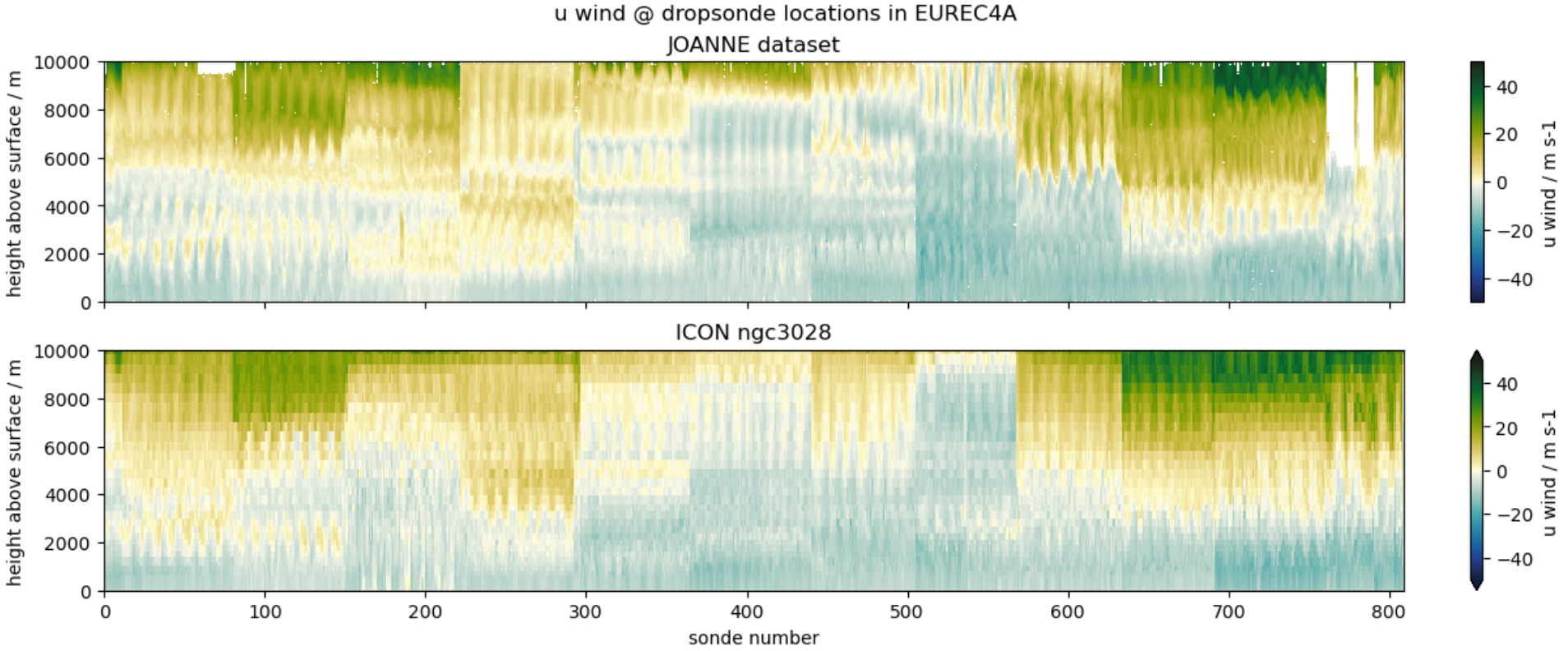
```
sonde_pix = healpy.ang2pix(
    icon.crs.healpix_nside, joanne.flight_lon, joanne.flight_lat,
    lonlat=True, nest=True
)

icon_sondes = (
    icon[["ua", "va", "ta", "hus"]]
    .sel(time=joanne.launch_time, method="nearest")
    .isel(cell=sonde_pix)
    .compute()
)
```

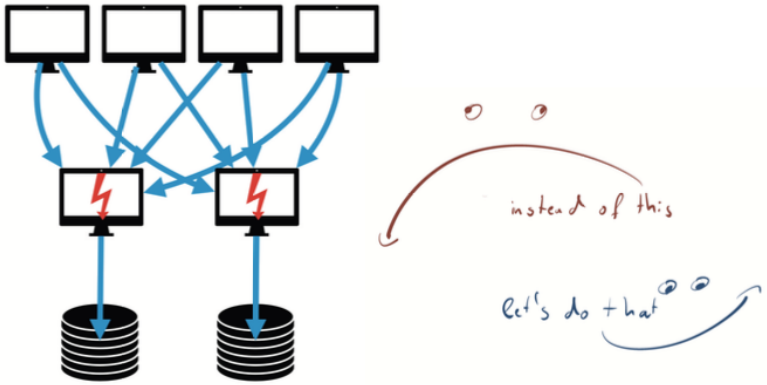
(55 sec, 1GB, single thread, full code at easy.gems)

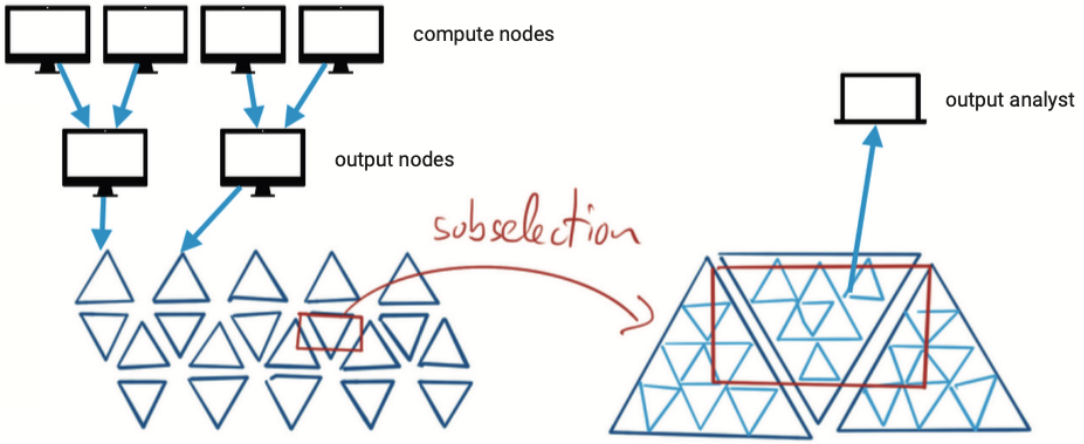
Select ICON model output at all dropsonde locations during EUREC4A field campaign:

```
icon_sondes = (
    icon[["ua", "va", "ta", "hus"]]
    .sel(time=joanne.launch_time, method="nearest")
    .dggs.sel_latlon(joanne.flight_lat, joanne.flight_lon)
    .compute()
)
```

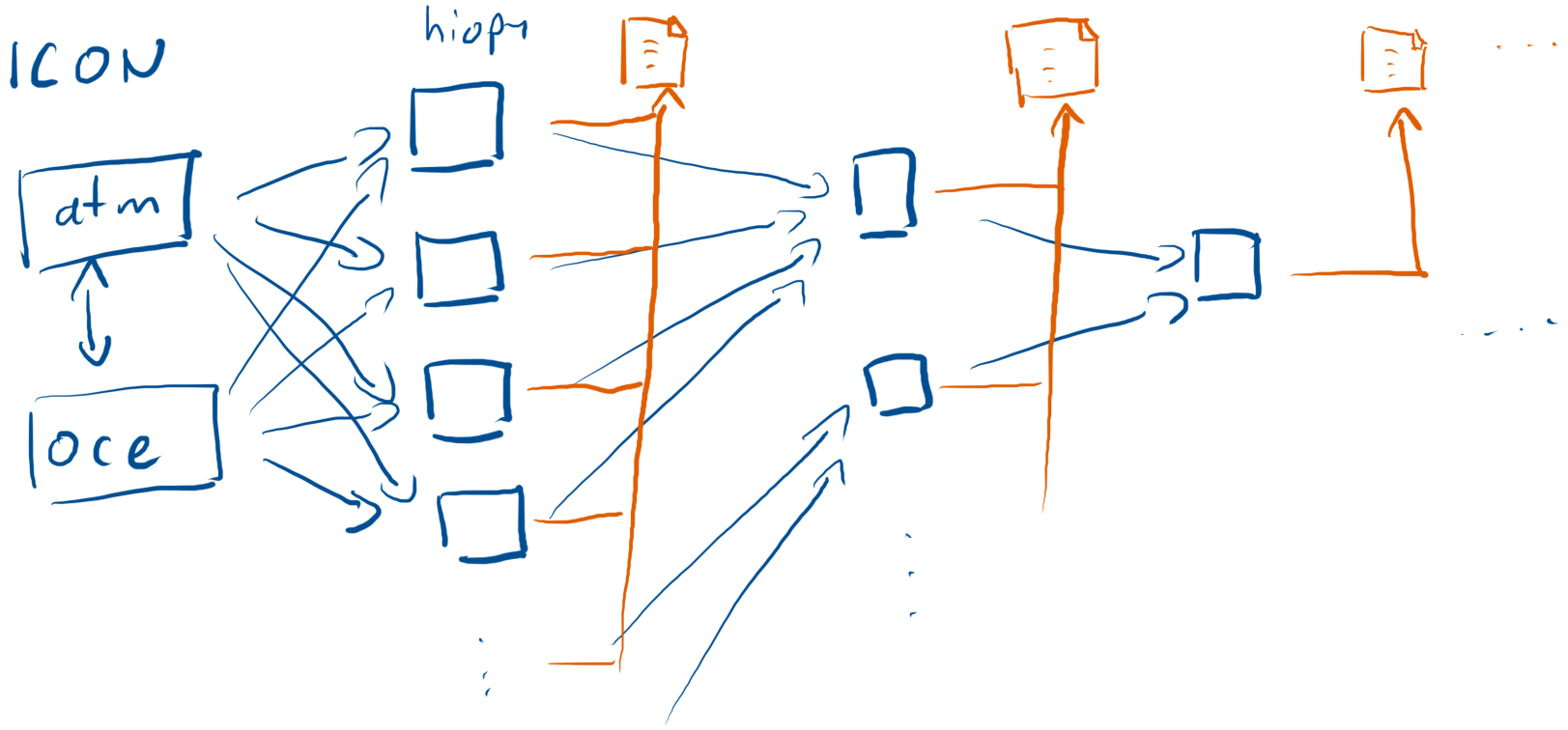
with XDGGS (?)

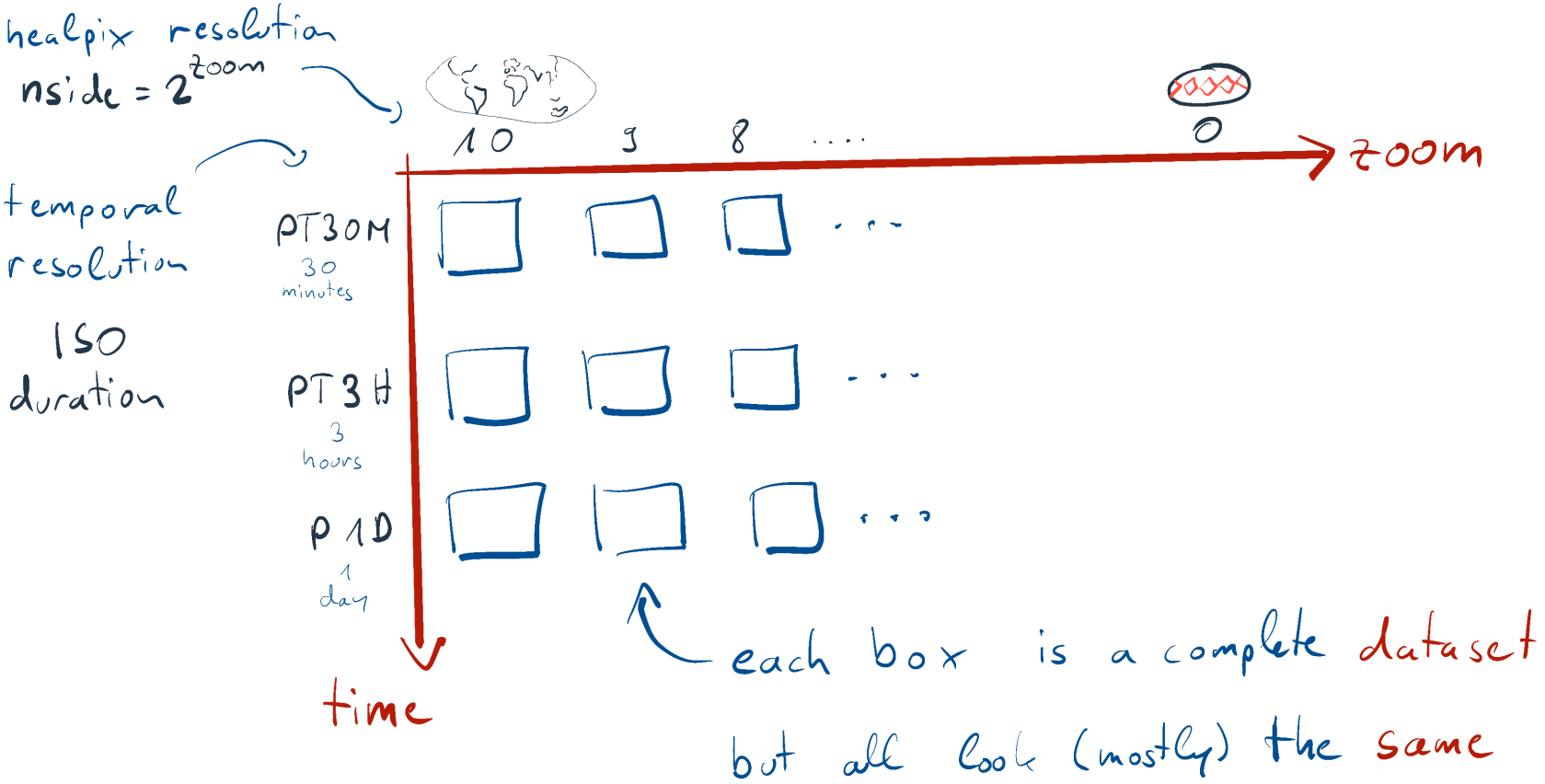

u wind @ dropsonde locations in EUREC4A
JOANNE dataset
ICON ngc3028
height above surface / m
10000
8000
6000
4000
2000
0
40
20
0
−20
−40
u wind / m s-1
0
100
200
300
400
500
600
700
800
sonde number


instead of this
let's do that

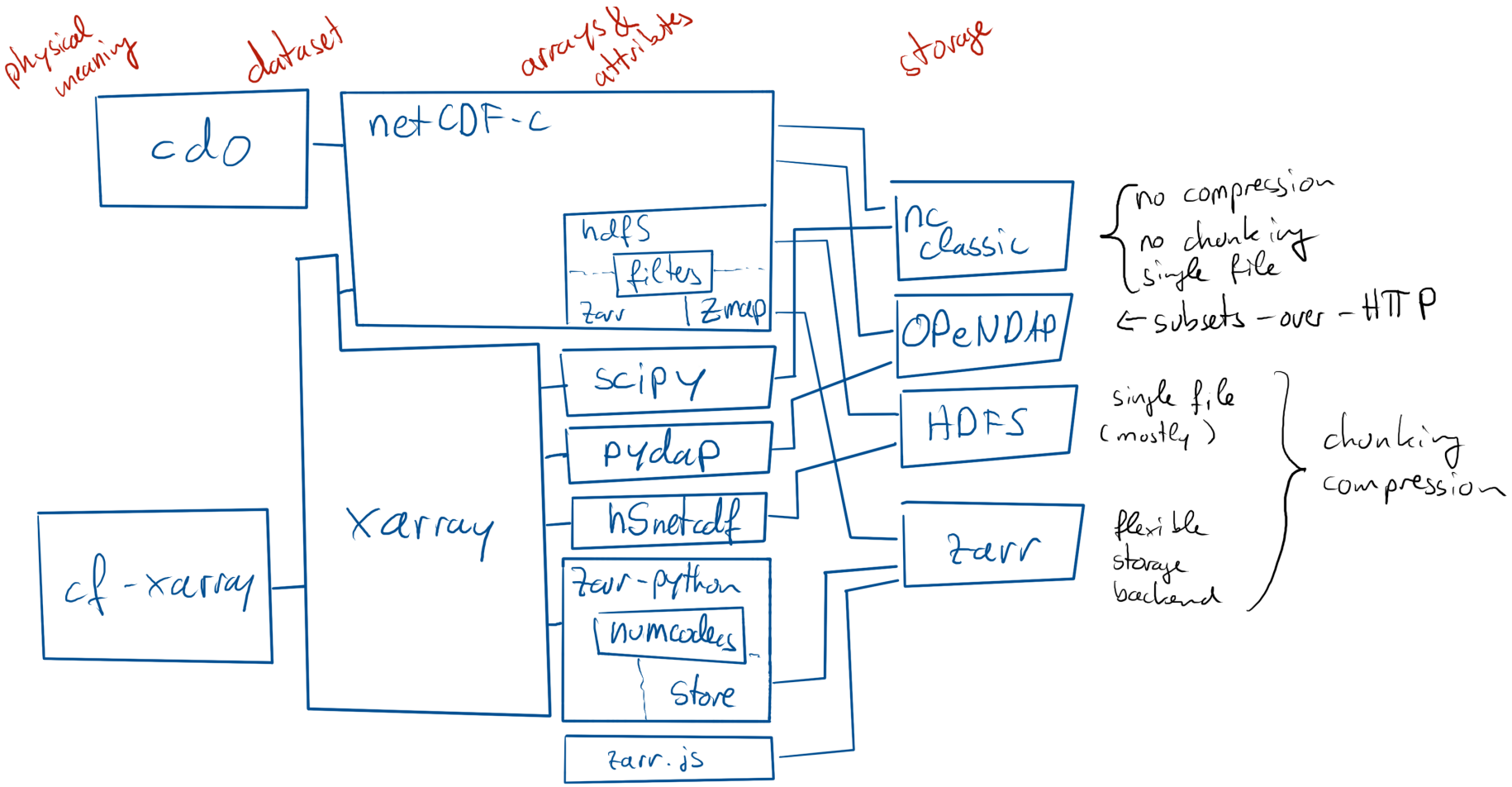

compute nodes
output nodes
output analyst
subselection

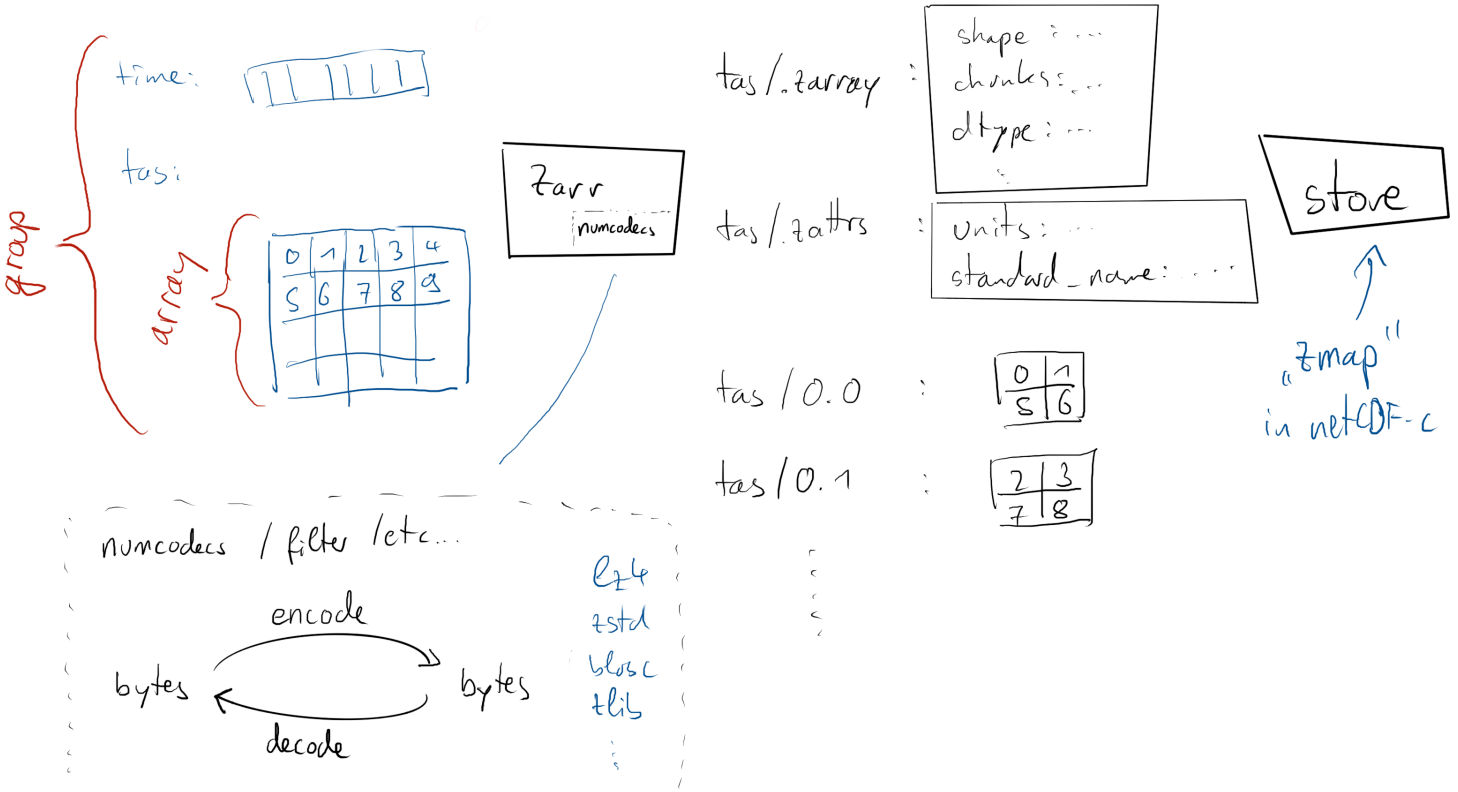
```
yac = YAC(xml, xsd)
sgd = HealPixSubgridDefinition(2**10, nchunks, ichunk)
dataset = zarr.open_consolidated(output_folder, mode="r+")
comp_id = yac.def_comp("healpix_io")
point_id, grid = make_yac_grid(sgd)
fields = {varname: Field.create(varname, comp_id, point_id)
          for varname in varnames}
put_fields = ...
yac.search()
steps = compute_nsteps(yac, fields)
for i in range(steps):
    for varname, field in fields.items():
        buffer = field.get()
        put_fields[varname].put(coarsen(buffer))
        dataset[varname][i,sgd.cell_chunk_slice] = buffer
```


ICON
hiopy
atm
oce


healpix resolution
nside = 2^zoom
temporal resolution
ISO duration
10
9
8
....
0
zoom
PT30M
30 minutes
PT3H
3 hours
P1D
1 day
time
each box is a complete dataset
but all look (mostly) the same

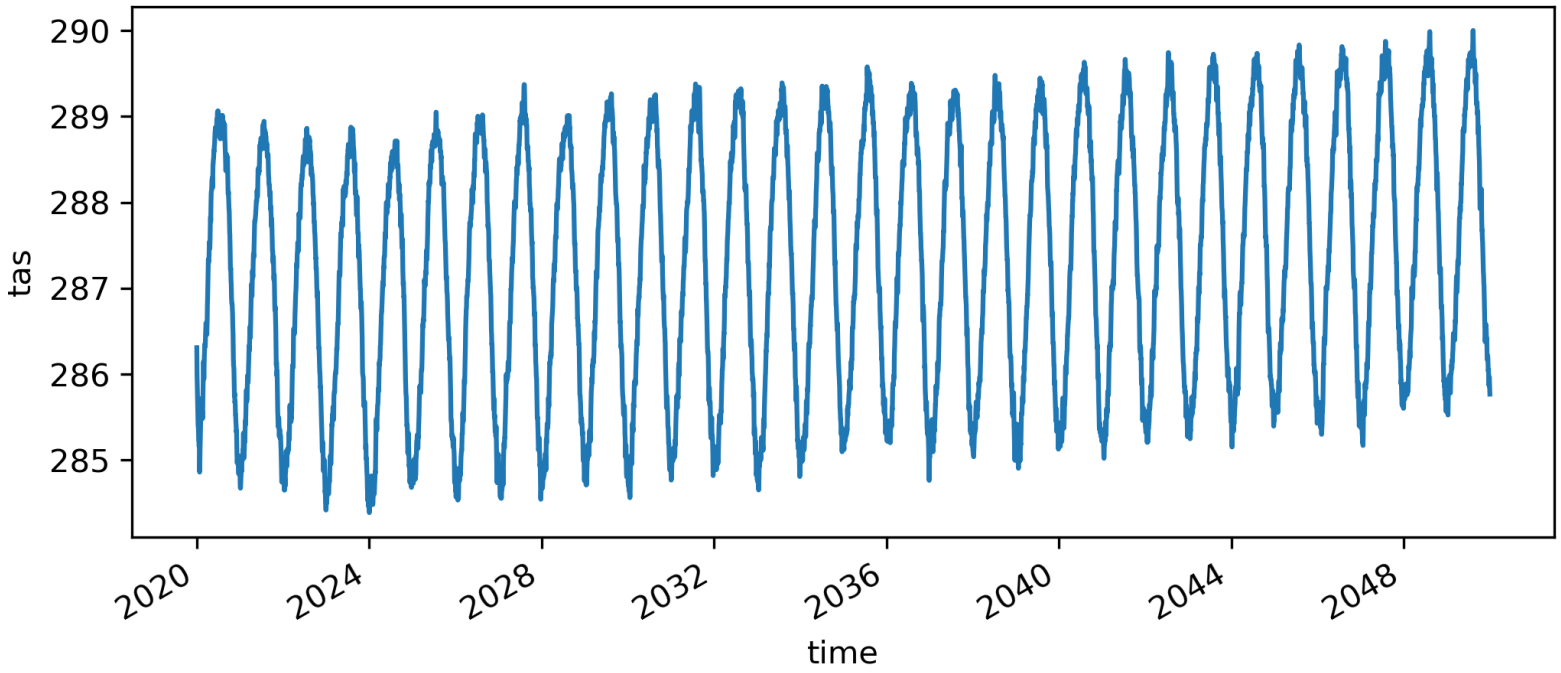
- output process is coupled to the running model
- writes entire hierarchy at once
- dataset is accessible as soon as the model **starts**


physical meaning
dataset
arrays & attributes
storage
cdo
netCDF-c
hdf5
filters
zarr
Zmap
scipy
pydap
h5netcdf
zarr-python
numcodecs
Store
zarr.js
xarray
cf-xarray
nc classic
OPeNDAP
HDF5
zarr
no compression
no chunking
single file
← subsets-over-HTTP
single file (mostly)
flexible storage backend
chunking
compression

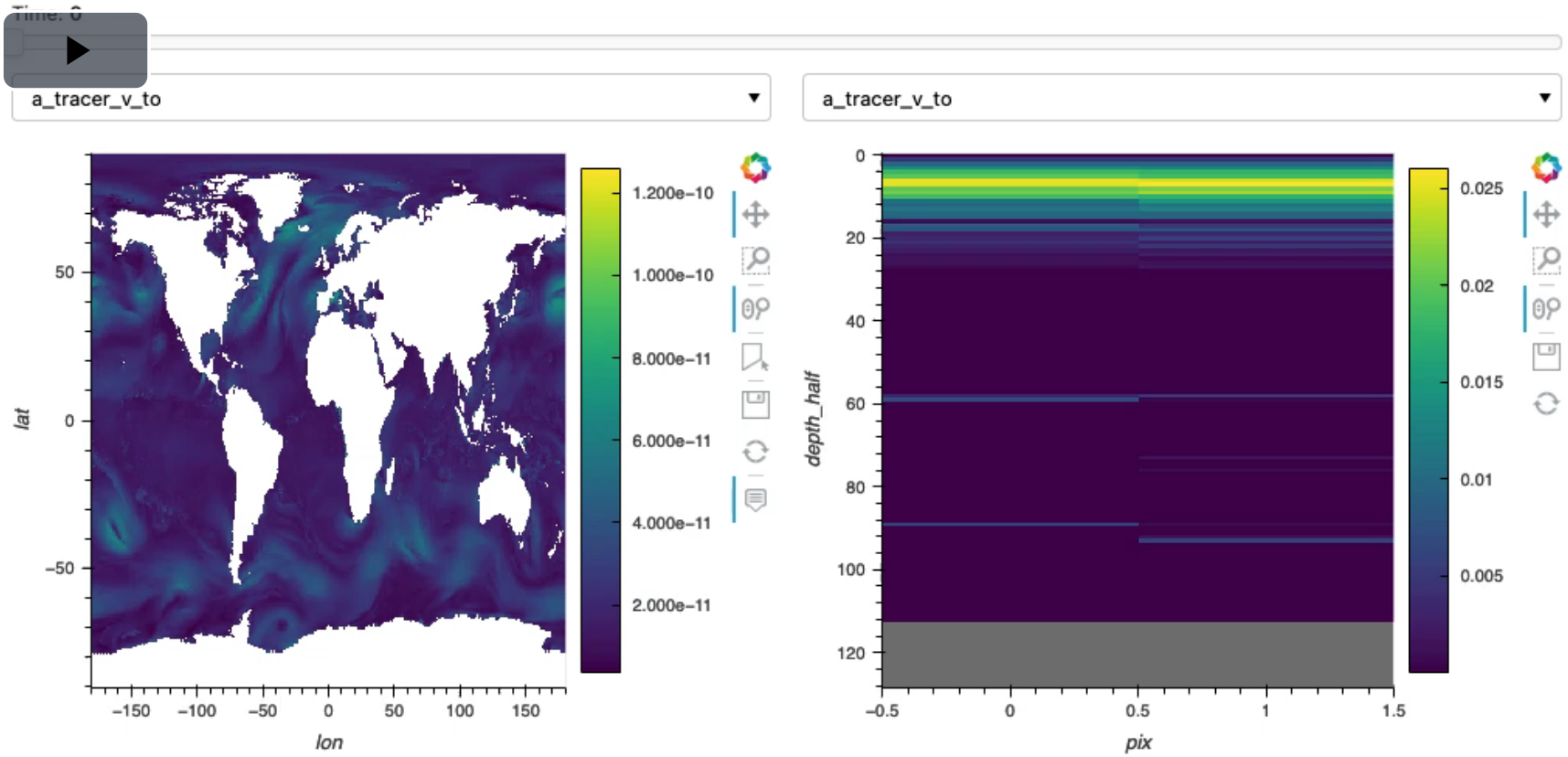

group
time:
tas:
array
0 1 2 3 4
5 6 7 8 9
zarr
numcodecs
tas/.zarray :
shape : ...
chunks : ...
dtype : ...
tas/.zattrs :
units : ...
standard_name : ...
store
"zmap"
in netCDF-C
tas/0.0 :
0 1
5 6
tas/0.1 :
2 3
7 8
numcodecs / filter / etc...
encode
bytes
bytes
decode
lz4
zstd
blosc
tiff
Filesystem
GCS, S3, SWIFT
MongoDB
Sqlite
Zipfile
reference
Memory
SLK
IPFS / IPLD
Sharded

- can **improve** performance (less I/O)
- **reduces** storage cost
- requires good chunking

```
1 cat.ICON.ngc4008(time="P1D", zoom="0").to_dask().tas.mean("cell").p
```



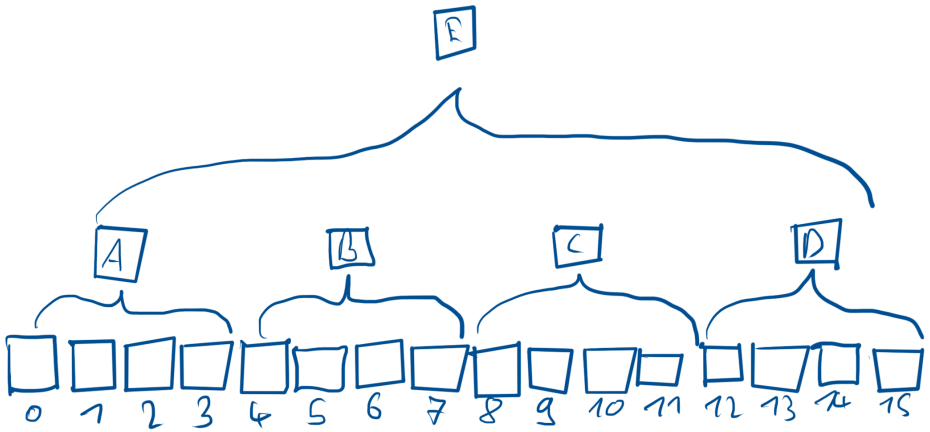
(100ms, 250MB, single thread)


a_tracer_v_to
a_tracer_v_to
1.200e−10
1.000e−10
8.000e−11
6.000e−11
4.000e−11
2.000e−11
lat
lon
50
0
−50
−150
−100
−50
0
50
100
150
0.025
0.02
0.015
0.01
0.005
depth_half
pix
0
20
40
60
80
100
120
−0.5
0
0.5
1
1.5

Output tested on multiple $\mathcal{O}(\mathrm{PB})$-scale model runs, 100+ users:

- remarkably little issues raised
- very positive general feedback
- enabled diagnostics which seemd impossible before

- Large datasets simplify access
- Chunking, hierarchies and compression play best together
- Healpix supports this ↗
- `hiopy` can build such datasets
- It’s about data structures, not so much the tooling



1 point

4 points

16 points

Σ 21 points

31% increase

mean = (E − "0" − D + "12") / 12

4 loads instead of 12 ⟶ 3 times less reads