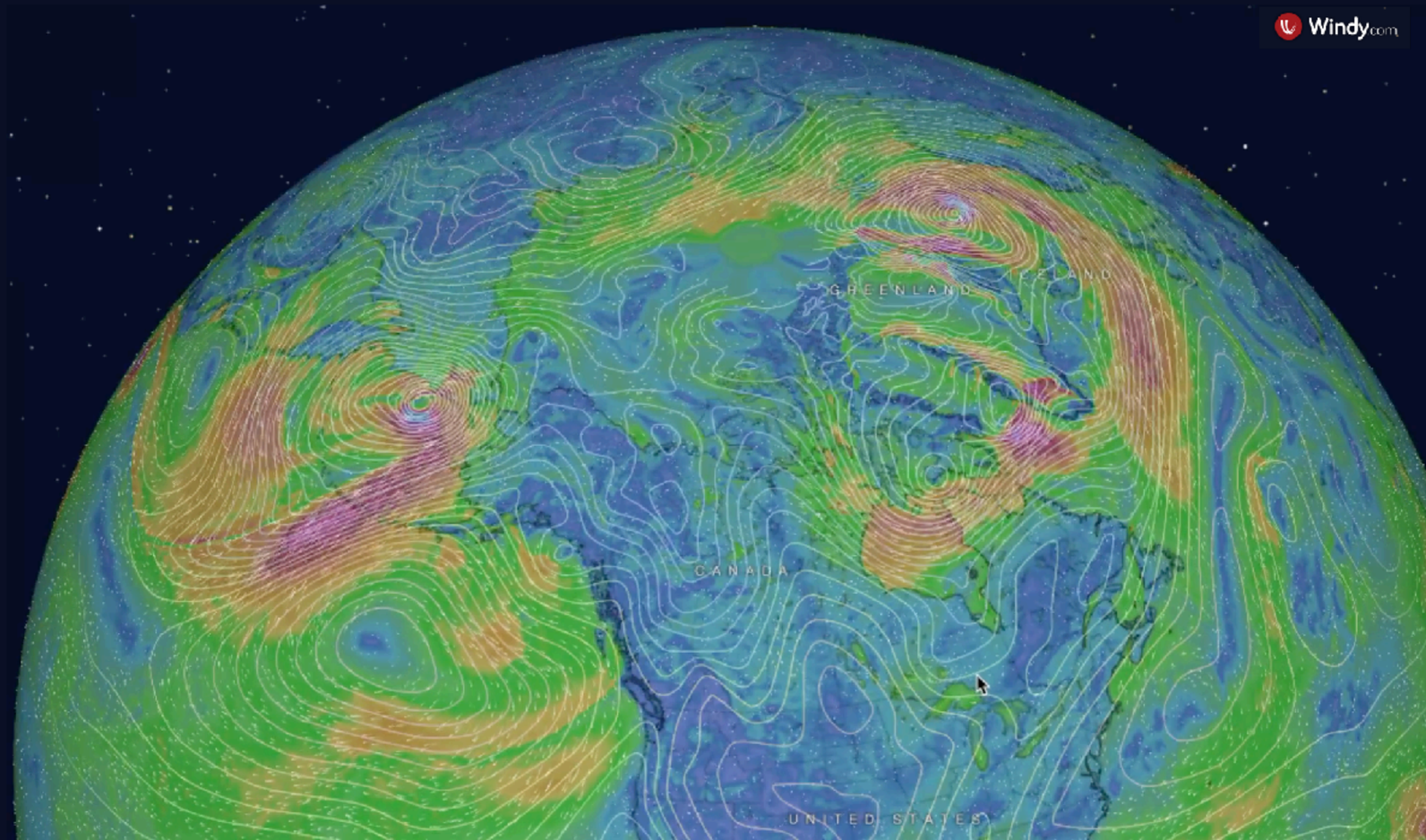
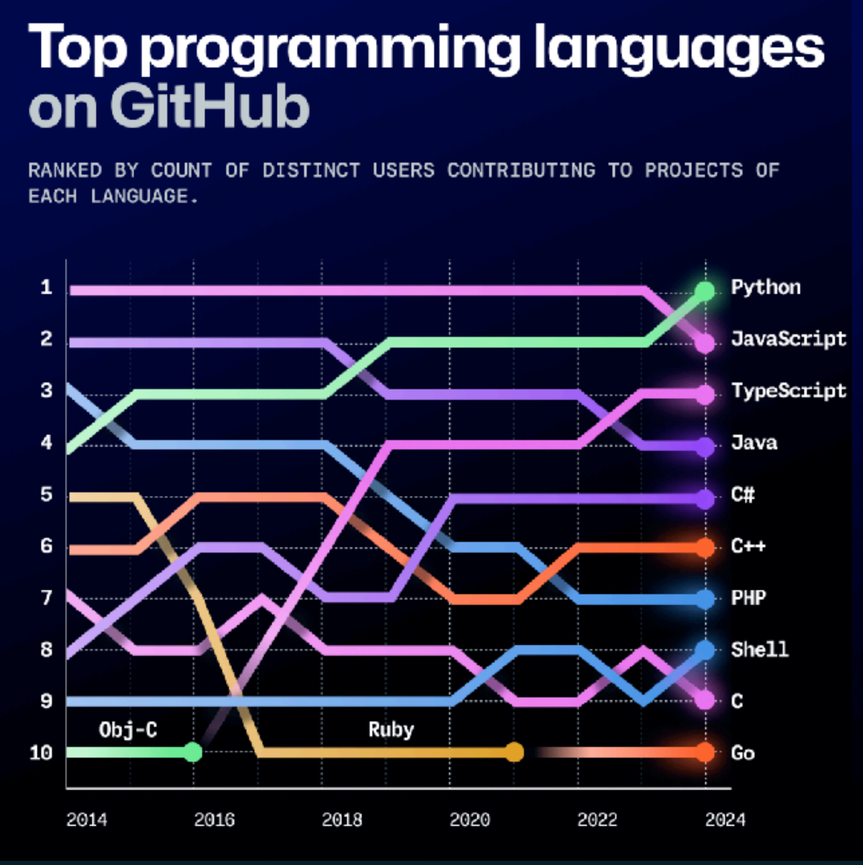
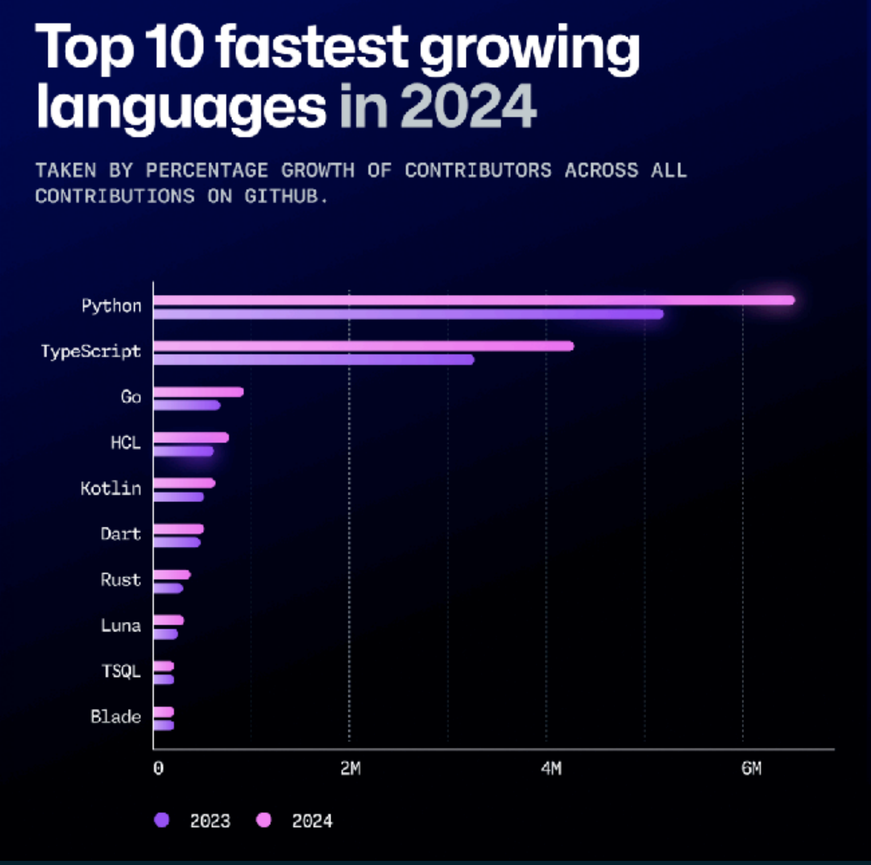
Interactive climate modelling

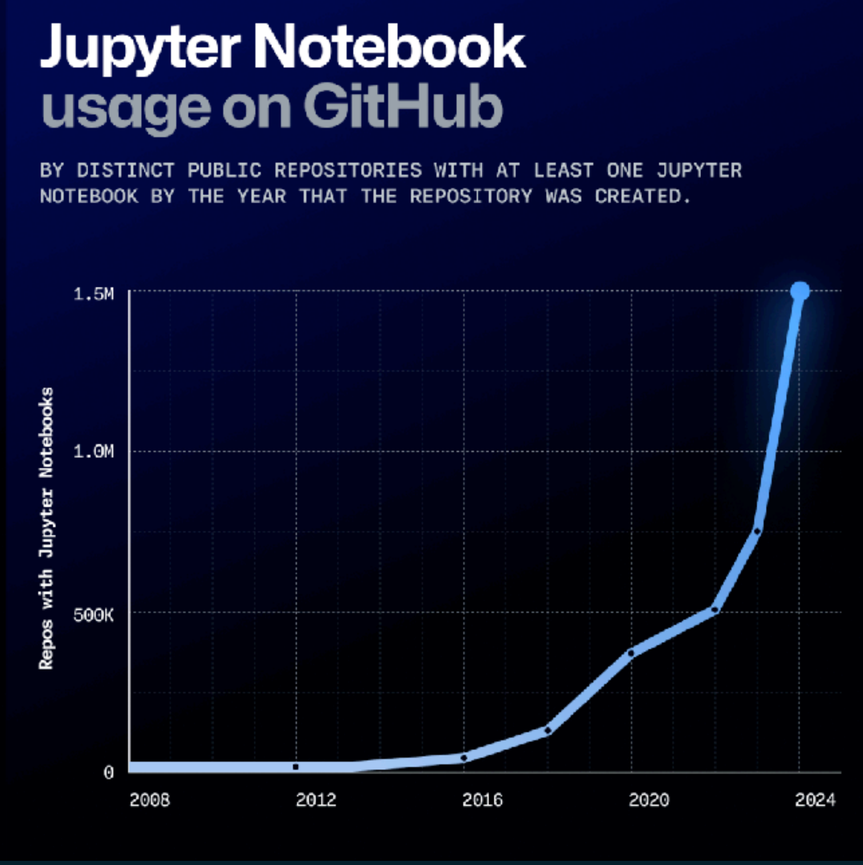


JuliaEO 2026, Terceira

Milan Klöwer and **many colleagues**
University of Oxford







From Octoverse 2024

Why is Python successful?

- ~~Speed~~ but using C
- Easy, interactive
- Cross-platform
- Extensions and software stack
- Used for open-source

Previous work on 0-dimensional climate models by, e.g.

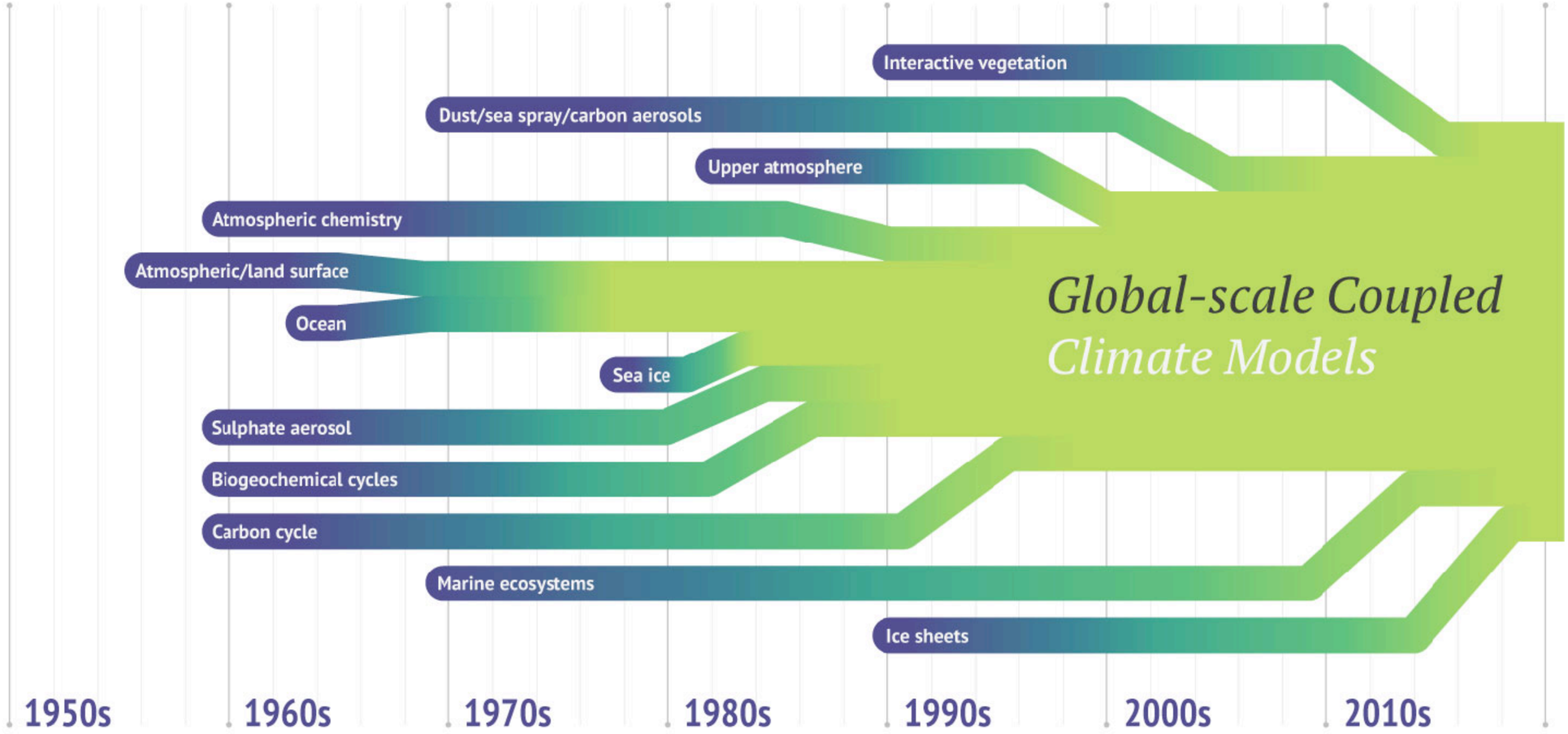
Arrhenius, 1896

On the Influence of Carbonic Acid in the Air upon the Temperature of the Ground

$$\Delta F = \alpha \ln(C/C_0)$$

Climate models

For decades scientists have been using mathematical models to help us learn more about the Earth's climate. Known as climate models, they are driven by the fundamental physics of the atmosphere and oceans, and the cycling of chemicals between living things and their environment. Over time they have increased in complexity, as separate components have merged to form coupled systems.



Note: There were some very simplified models before the dates mentioned.

CarbonBrief
CLEAR ON CLIMATE

The equations: It's complicated!

Gravity

Pressure gradient

Momentum advection

Coriolis

Viscous forces

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} + 2\boldsymbol{\Omega}\times\mathbf{u}\right) = \rho\mathbf{g} - \nabla p + \mu\nabla^2\mathbf{u}$$

Conservation of mass (compressible)

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0$$

Ideal gas law, equation of state

$$p = \rho R T$$

$$\frac{\partial T}{\partial t} + (\mathbf{u}\cdot\nabla)T - \frac{RT}{c_p}\frac{D\ln p}{Dt} = \frac{Q}{c_p}$$

Heat sources

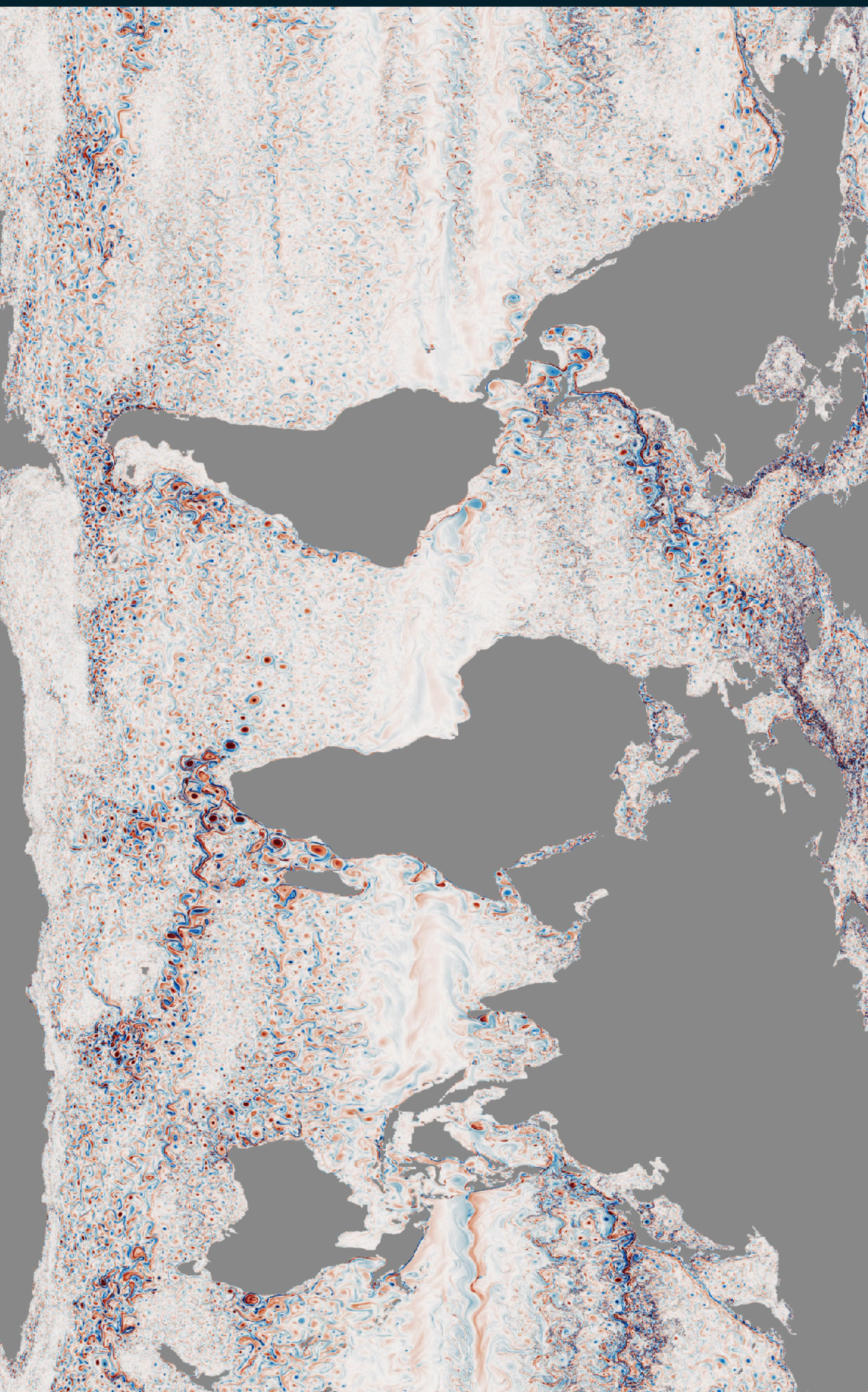
Temperature advection

Adiabatic conversion

What's missing? Humidity, clouds, precipitation, snow, radiation, surface fluxes, ocean, land, …

Global ocean simulated at 10km

Silvestri et al. 2023 *arXiv:2309.06662*



Parameterizations

Representing the unresolved processes

Atmosphere
- Radiation
- Clouds
- Precipitation
- Convection
- Surface fluxes
- Turbulence
- Gravity waves

Ocean
- Vertical diffusion and mixing
- Eddies and turbulence
- Cross-thermocline transport
- Interaction with bathymetry
- Surface fluxes (atmosphere and ice)

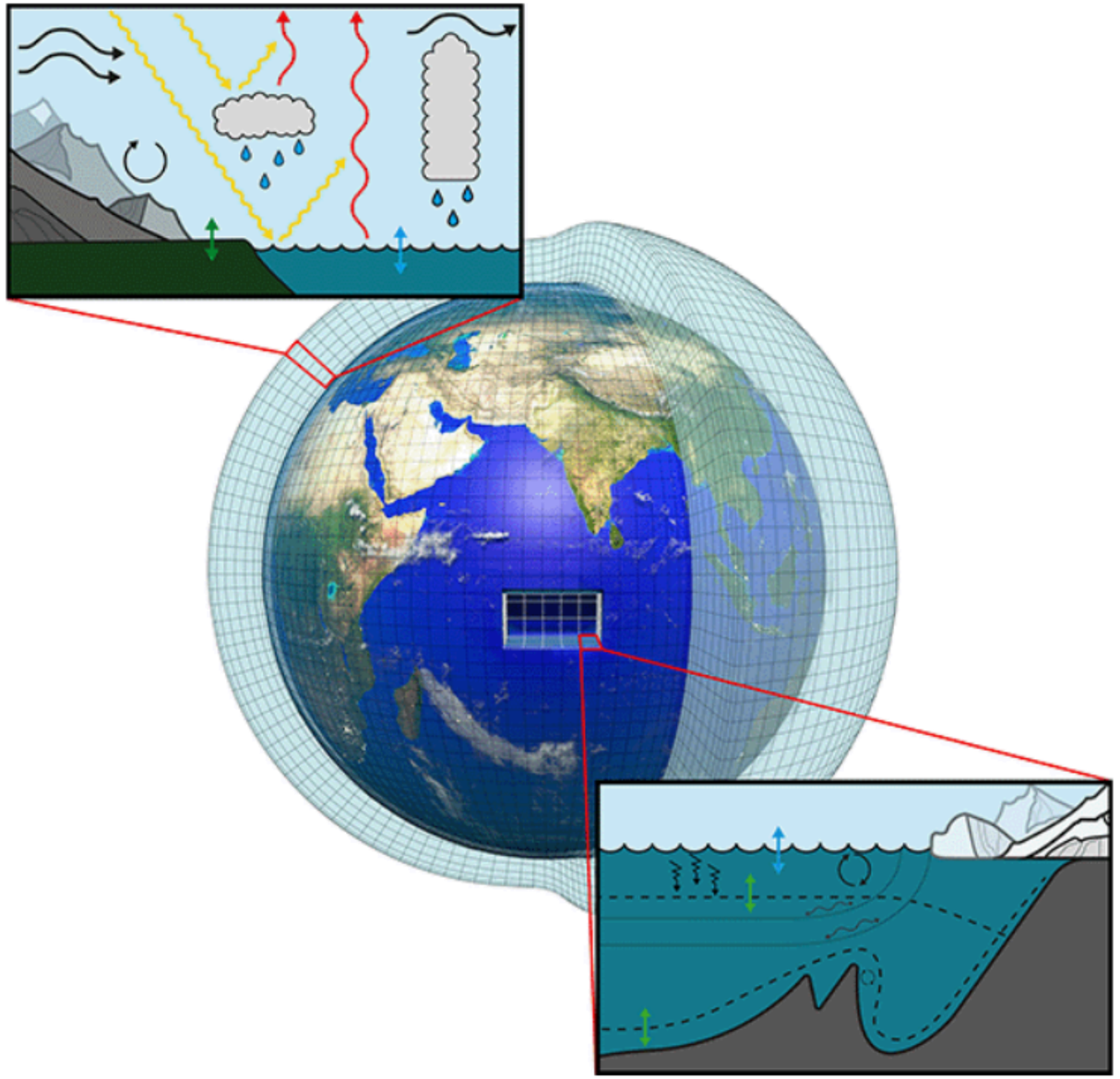


Figure 2. Atmospheric and oceanic parametrization schemes. Within an atmospheric column, typical parametrization schemes include moist convection, clouds and cloud microphysics, radiation, turbulence, turbulent exchange with the land and ocean, and both orographic and non-orographic gravity wave drag. Within an oceanic column, typical parametrization schemes include vertical diffusion, sub-mesoscale and mesoscale eddies, turbulent exchange with the atmosphere, and cross-thermocline transports.

Christensen & Zanna, 2022

High-performance computing is cool
but how to accelerate climate research?

Oldschool Fortran namelists: A *monolithic* interface?

You can only change what the developers wants you to change

The NAMELIST input data file specifies the following:

```
NFIRST=1          start from scratch
NLAST=42          stop after timestep 42
NNERGY=20         print fields every 20 timesteps
NTSI=1            print single line of information every timestep
NMIX=10           do a mixing timestep every 10 timesteps
AMF=1.E9          horizontal mixing of momentum
AHF=2.E7          horizontal mixing of heat, salinity, tracers
FKPMF=1.0         vertical mixing of momentum
FKPHF=1.0         vertical mixing of heat, salinity, tracers
DTTSF=172800.     length of timestep on temp., sal., tracers
DTUVF=7200.       length of timestep on internal mode velocities
DTSFF=7200.       length of timestep on stream function
MXSCAN=25         cut off relaxation at 25 scans if not converged
SORF=1.50         coefficient of over-relaxation
CRITF=4.E9        criterion for convergence of relaxation
ACORF=0.5         weight forward and backward timestep equally in
                    the semi-implicit Coriolis term
TINITF=...        initial values of temperature, salinity, tracers
                    (note..  for purposes of precision, salinity is
                     carried in the model in units of parts per part,
                     with .035 subtracted off, however TINITF is ex-
                     pected in parts per part; .035 is subtracted
                     after read-in)
ISIS=13,...       the island and its perimeter points are included
                    in a box between I=13-16 and J=9-12
```

- Speed ✓
- Cross-platform ✗
- Extensions ✗
- Open-source ✗
- Easy, interactive ✗
- Physical accuracy ✓

Cox (1984), NOAA GFDL Tech Report No. 1

Monolithic interface

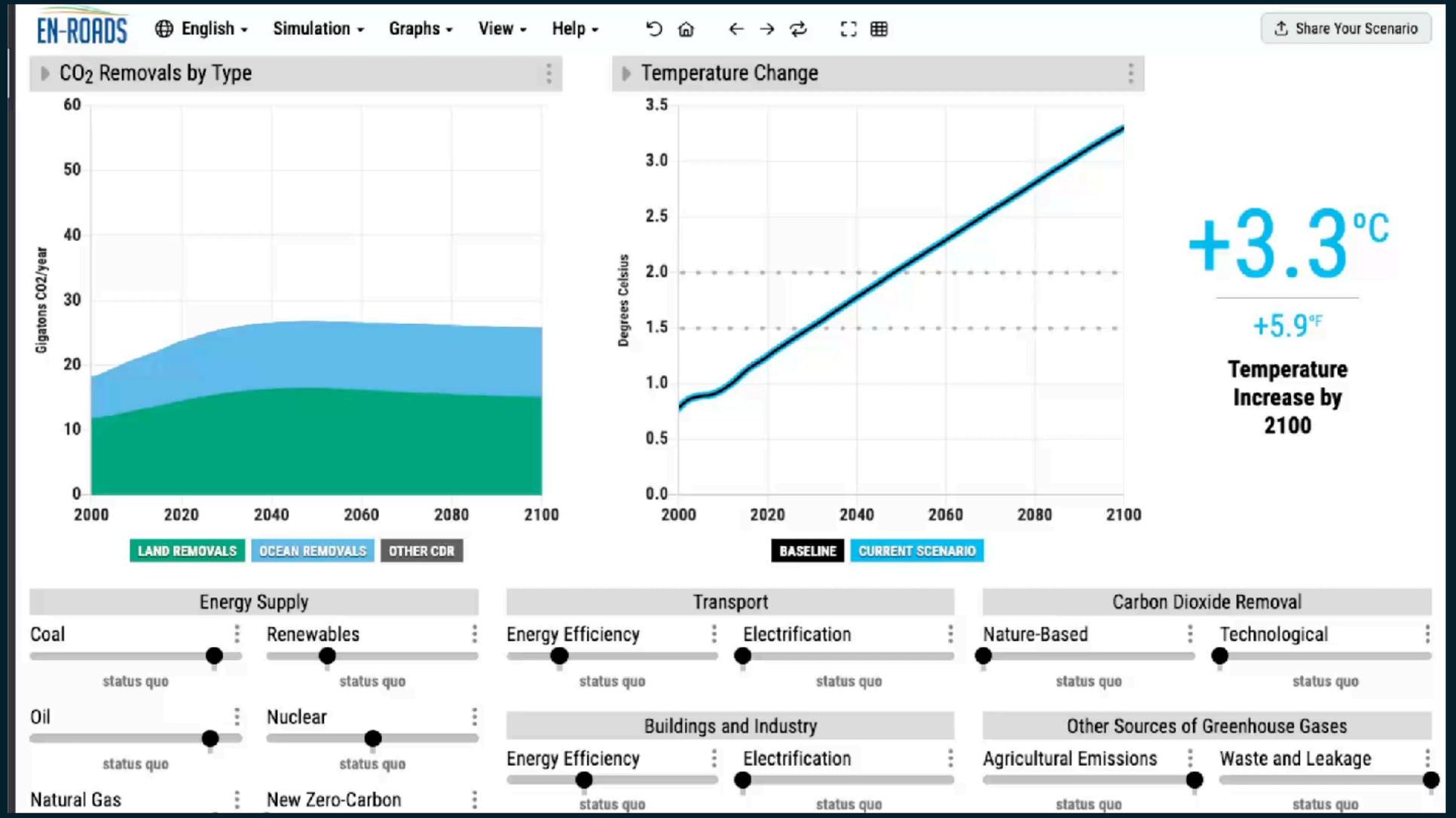
(Spend days or weeks
to compile the model)

You can steer but you can’t change the plane!

Interactive but also a monolithic interface

en-roads.climateinteractive.org

Cool, but you still can only change what the developers wants you to change



Climate models are used for

(*not* weather forecast models)

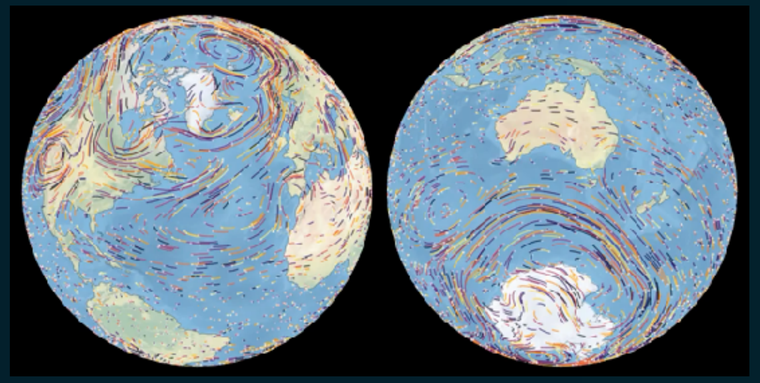
- Repeated pre-defined scenarios (CMIP etc)
- High-performance computing benchmarks
- Scientific one-off experiments
- Counterfactual worlds
- Software development
- Exoplanets
- Teaching (users become developers)
- …

Many are not standard/repeated workflows!

Advantages of a monolithic interface

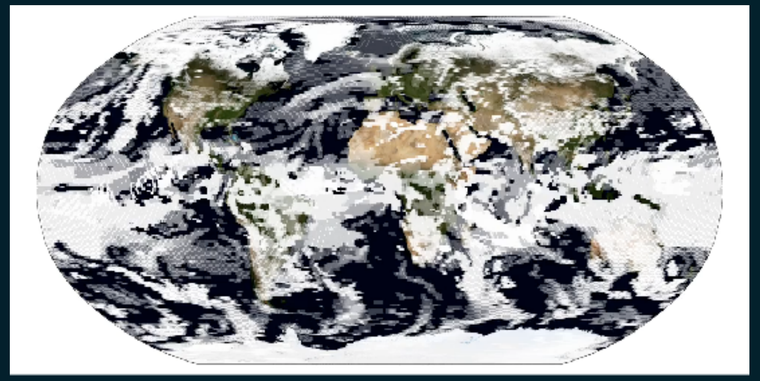
- Simpler interface
- Easier to design
- Easier to anticipate user experience
- Safety (harder to break things)
- Easier to maintain
- Developer/user separation

SpeedyWeather: Play climate modelling like it's LEGO

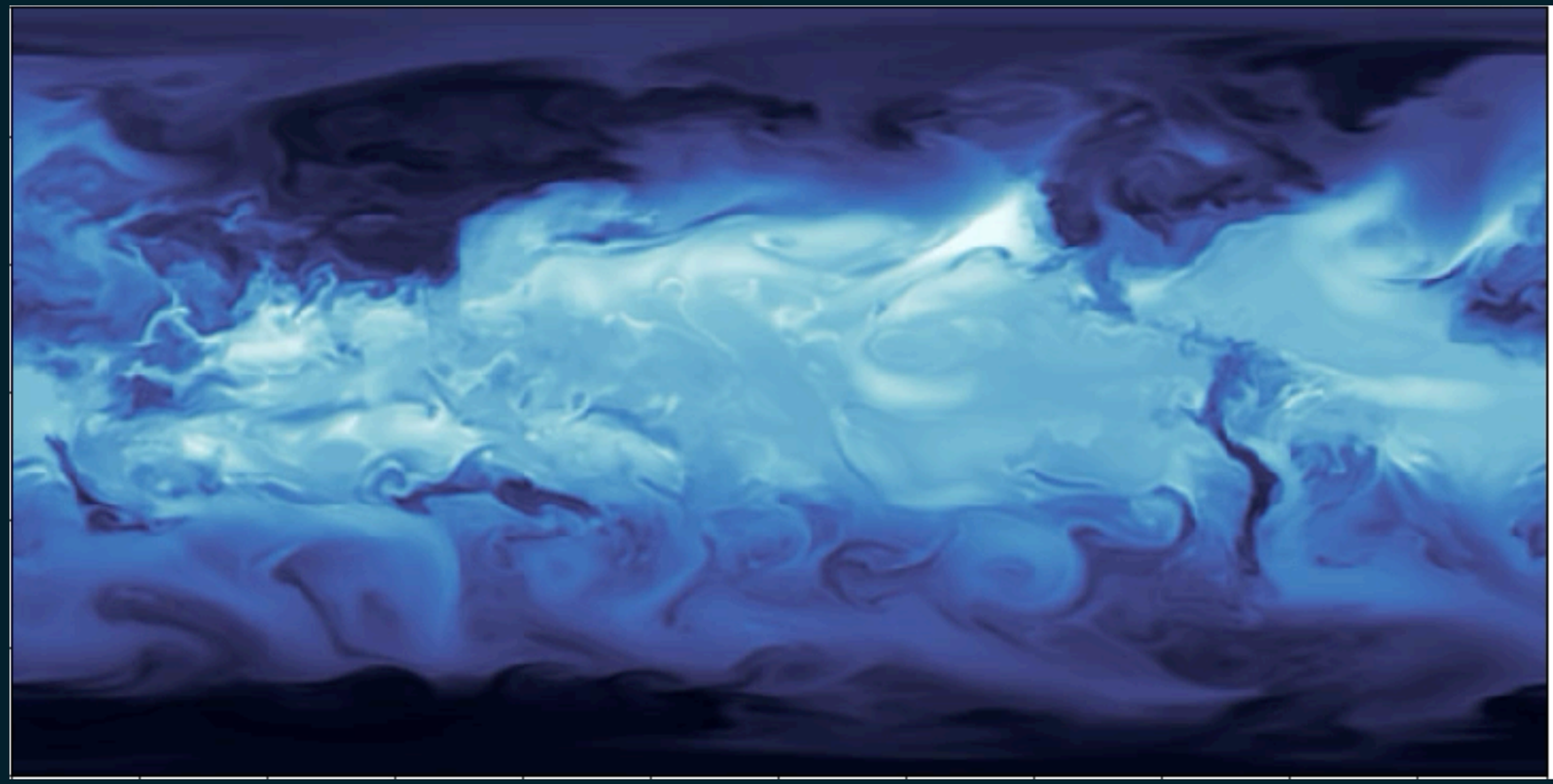


Particle advection

Cloud cover



Humidity at 50km resolution



Pier Luigi Vidale: "*unthinkable on such a small laptop … like playing a video game*"

SpeedyWeather: Play climate modelling like it's LEGO

Honest (?) self assessment

- Speed (incl GPUs) ✓
- Cross-platform ✓
- Extensions ✓
- Open-source ✓
- Easy, interactive ✓
- Physical accuracy ?

using Pluto.jl notebooks

SpeedyWeather demo

```
using SpeedyWeather ✓, PlutoUI ✓
```

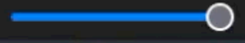
```
import GLMakie ✓
```

Choose resolution

```
@bind resolution Slider([21, 31, 42, 63, 85])
```

33.4 ms

Choose a forcing parameter



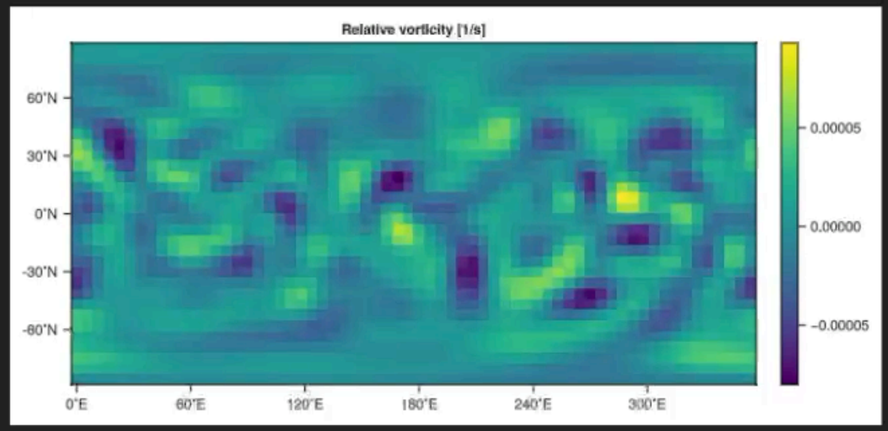
```
@bind m Slider(2:2:8)
```

33.9 ms

Hack a quick extension to SpeedyWeather

```
begin
    struct MyKolmogorovFlow <: SpeedyWeather.AbstractForcing end
    function SpeedyWeather.forcing!(d, p, ::MyKolmogorovFlow, lf, model)
        Fu = d.tendencies.u_tend_grid
        set!(Fu, (λ, φ, σ) -> 3e-12*sind(m*φ), model.geometry)
    end
end
```

```
run!(simulation, period=Day(3));
```



SpeedyWeather's monorepo

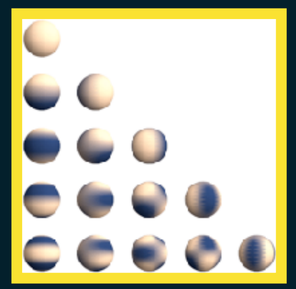
SpeedyWeather.jl

SpeedyTransforms.jl

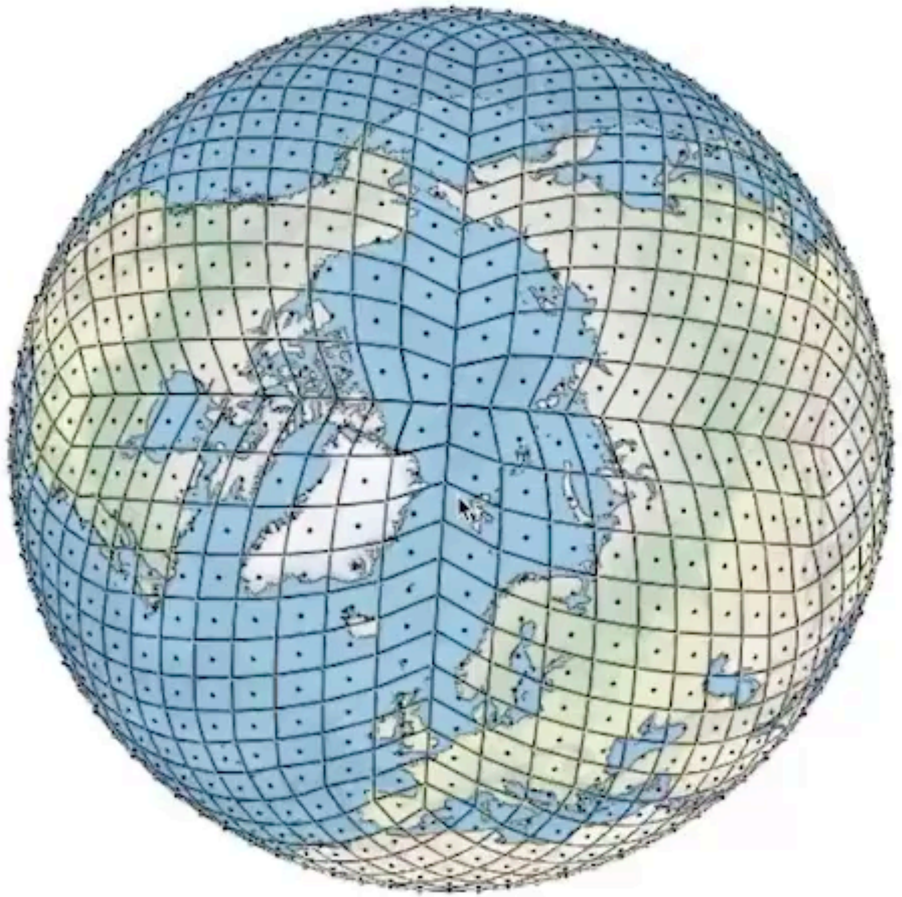
LowerTriangularArrays.jl

RingGrids.jl

SpeedyWeatherInternals.jl



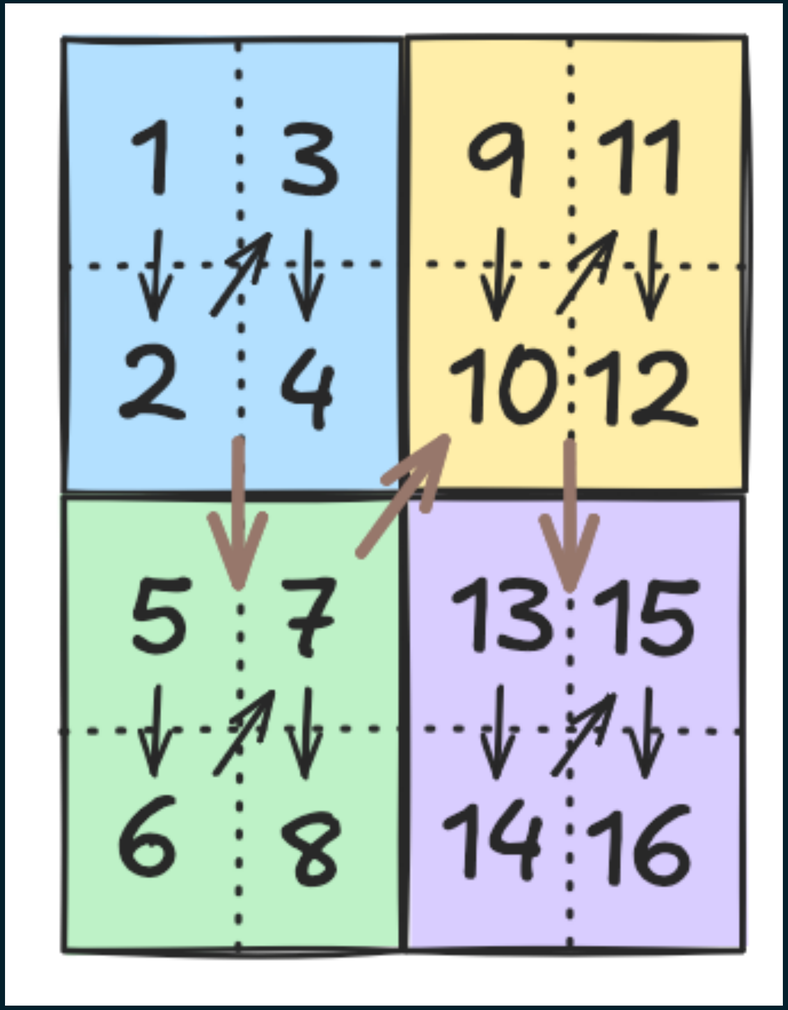
RingGrids.jl



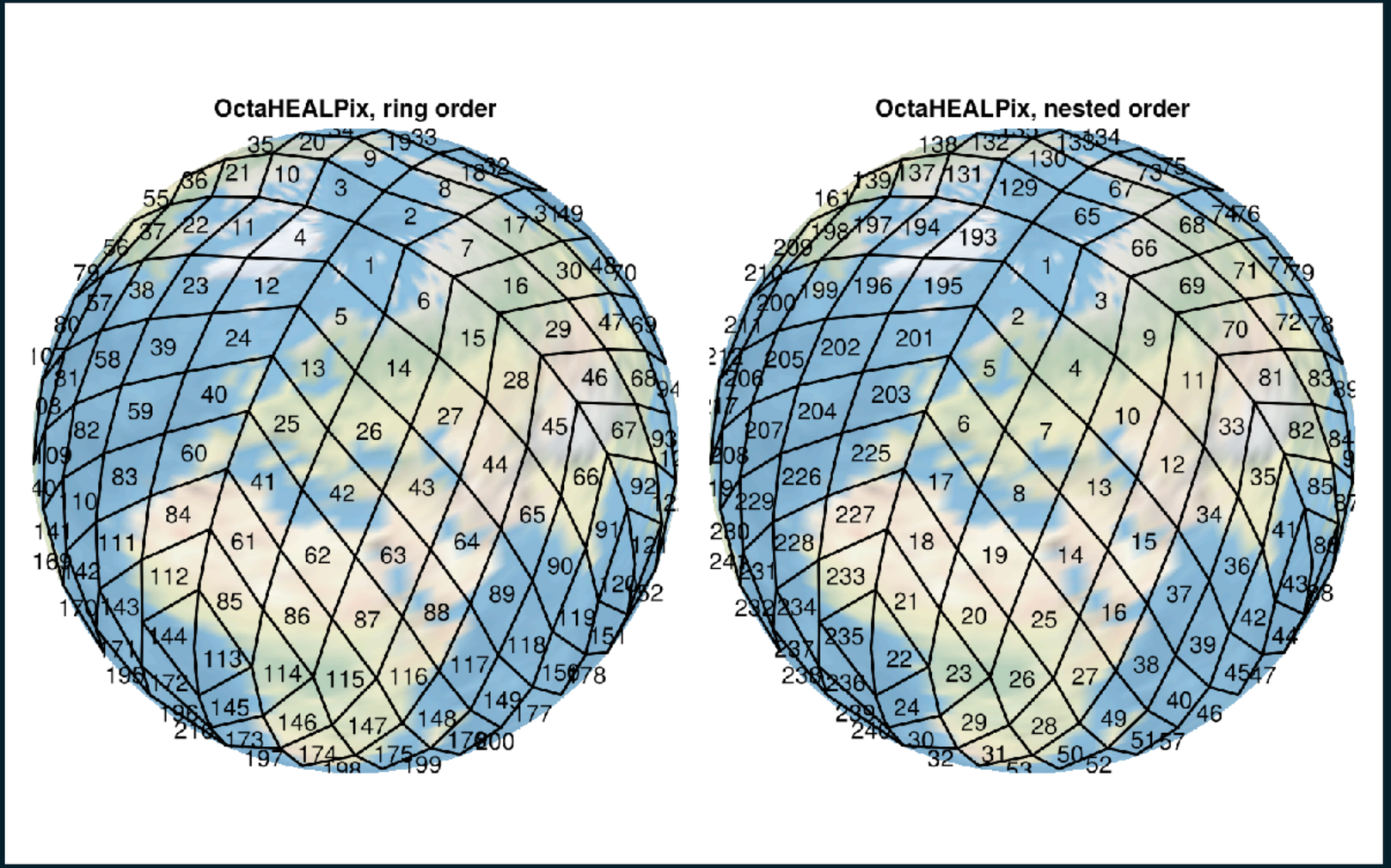
HEALPix grid visualised with GLMakie+GeoMakie

- Gaussian and octahedral grids
- HEALPix
- OctaHEALPix (new!)
- N-dimensional fields
- Array-agnostic
- GPU ready

RingGrids.jl — hierarchical HEALPix grids



Hierarchial yields trivial resolution changes



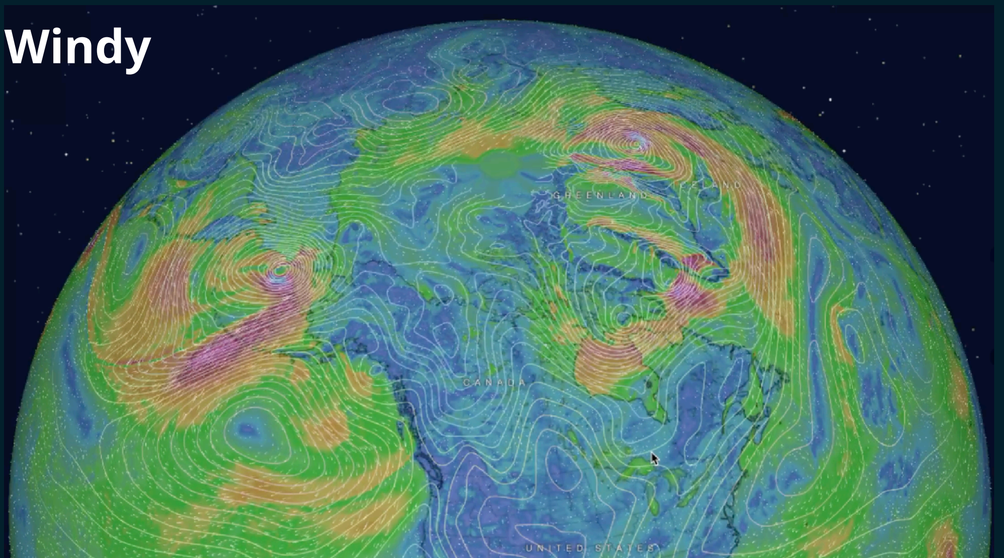
OctaHEALPix: Equal-area, efficient spectral transform, hierarchical, and just one big square matrix

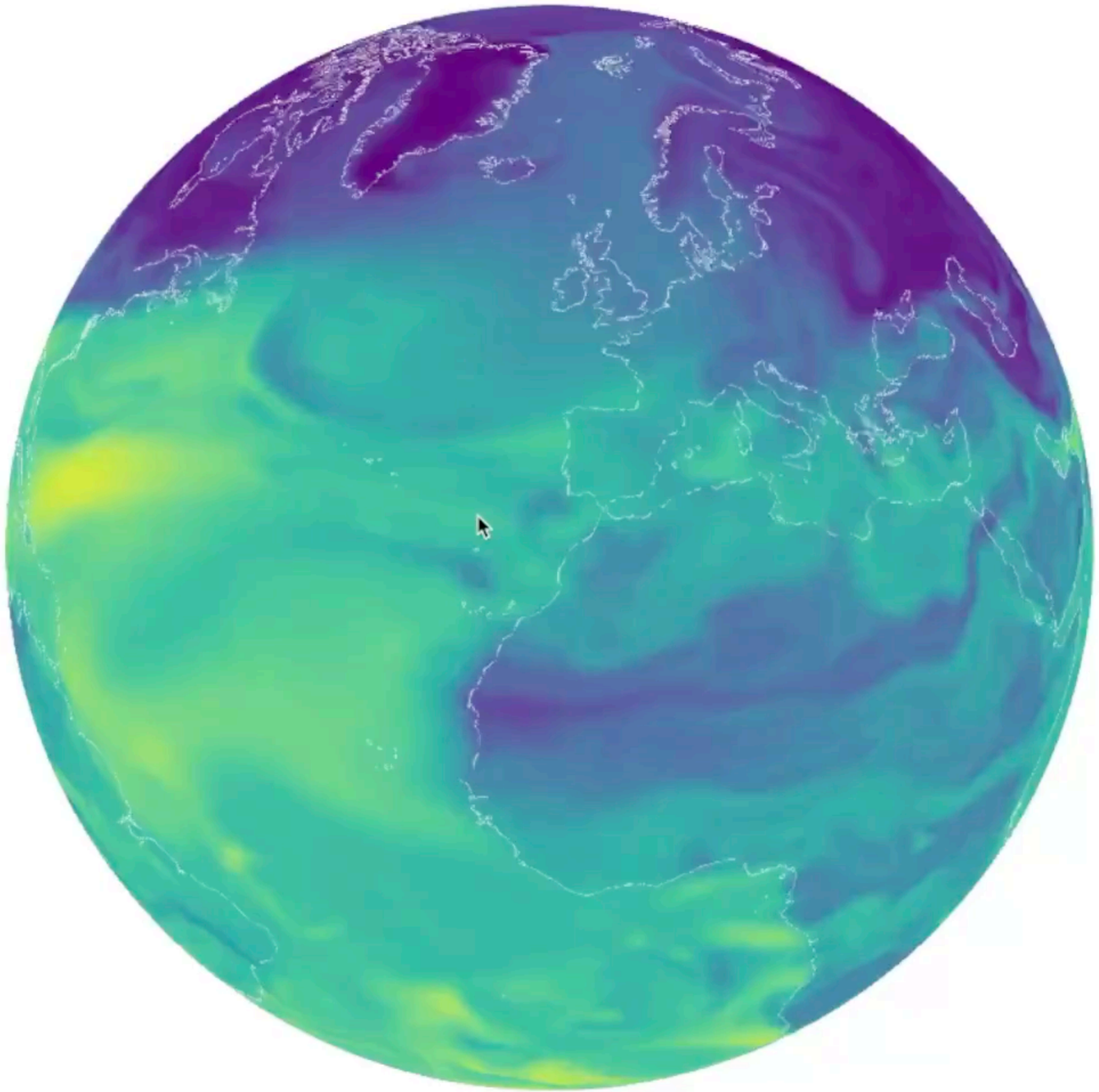
Climate modelling like playing with windy?

Fields plotted as polygons of the underlying grid

- *Add a play button?*
- *Switch fields of a simulation?*

Windy





SpeedyWeather+GeoMakie+GLMakie

SpeedyWeather: Play climate modelling like it's LEGO

- Modularity
- Flexible
- Hackable
- Easy to use
- Visualisation support
- GPU (almost) ready
- Differentiable via Enzyme

```
using SpeedyWeather ✓ , GLMakie ✓
```

```
spectral_grid =
SpectralGrid{Spectrum{...}, RingGrids.OctahedralGaussianGrid{...}}
├ Number format: Float32
├ Spectral:      T21 LowerTriangularMatrix
├ Grid:          32-ring RingGrids.OctahedralGaussianGrid, 1600 grid points
├ Resolution:    5.08°, 565km (at 6371km radius)
├ Vertical:      8-layer atmosphere, 2-layer land
└ Architecture:  CPU using Array
```

```
spectral_grid = SpectralGrid(trunc=21, nlayers=8)
```

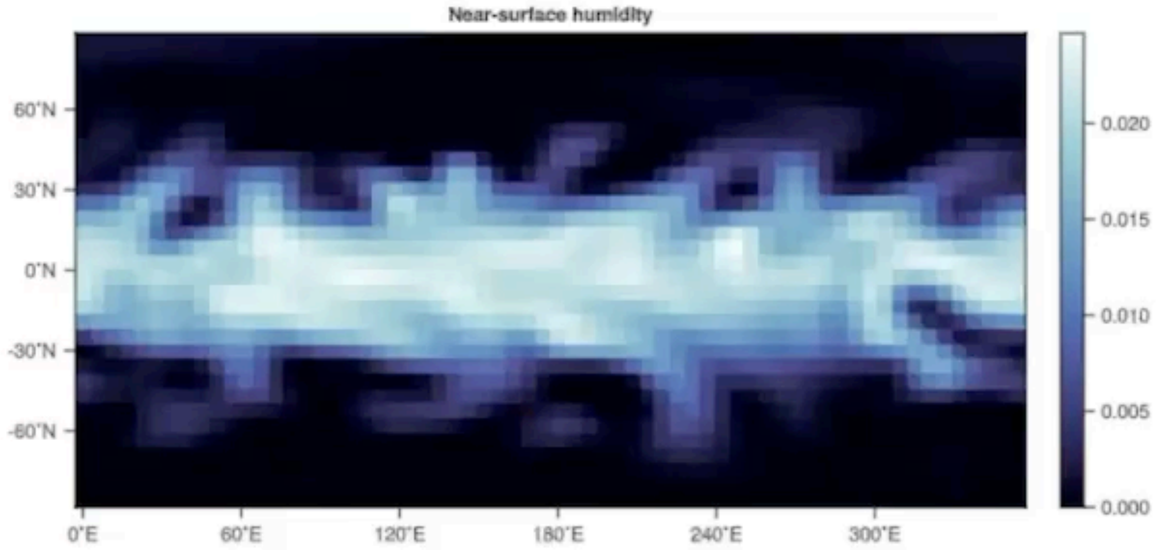
1.1 ms

```
model = PrimitiveWetModel(spectral_grid);
```

12.0 ms

```
simulation = initialize!(model);
```

```
run!(simulation, period=Day(10));
```




```
begin
    q = simulation.diagnostic_variables.grid.humid_grid[:, end]
    heatmap(q, title="Near-surface humidity", colormap=:ice)
end
```

Learning surface roughness

Roughness length =
f(
vegetation,
sea ice,
snow,
orography,
surface wind
)

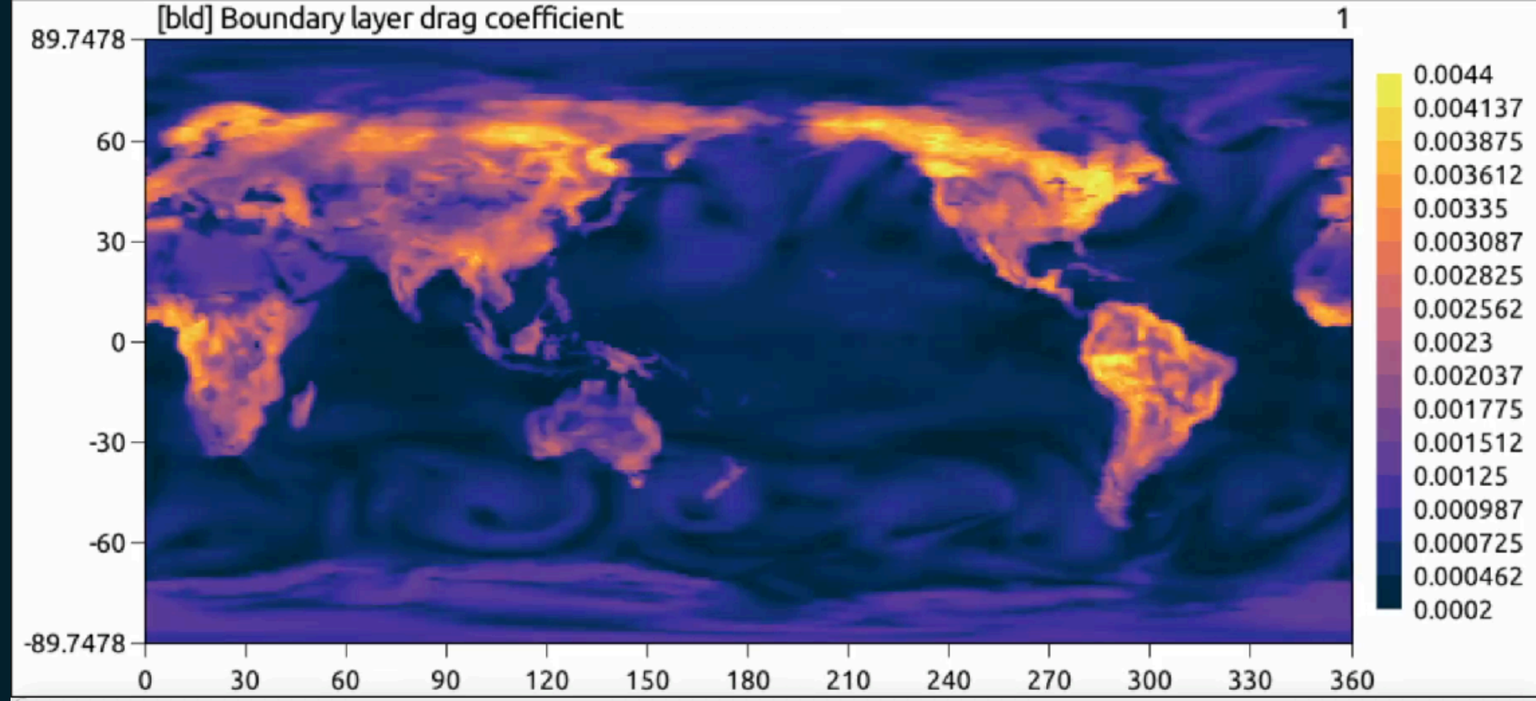
Learned offline in pytorch,
Inference done on every time step in every cell

0.5m

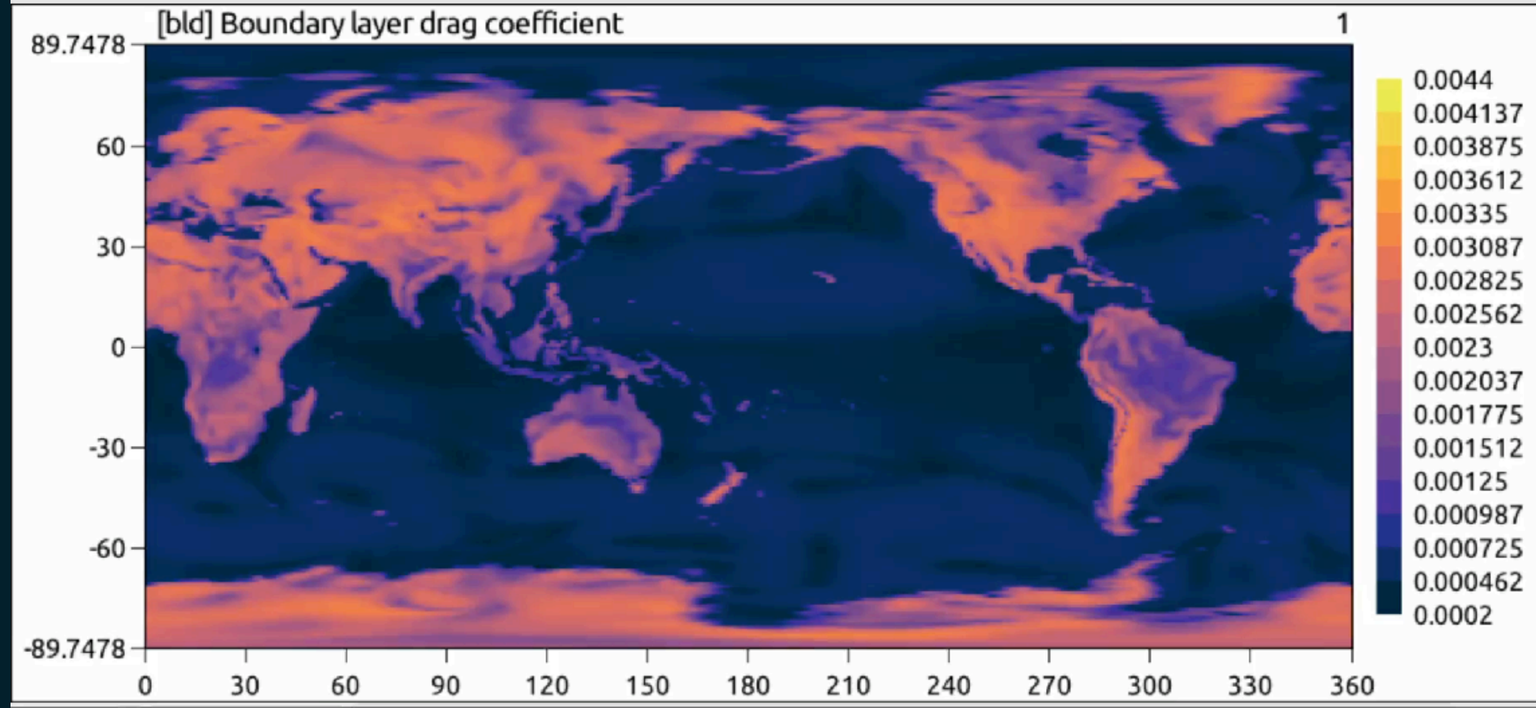
1cm

Constant over land / ocean

with Greg Munday

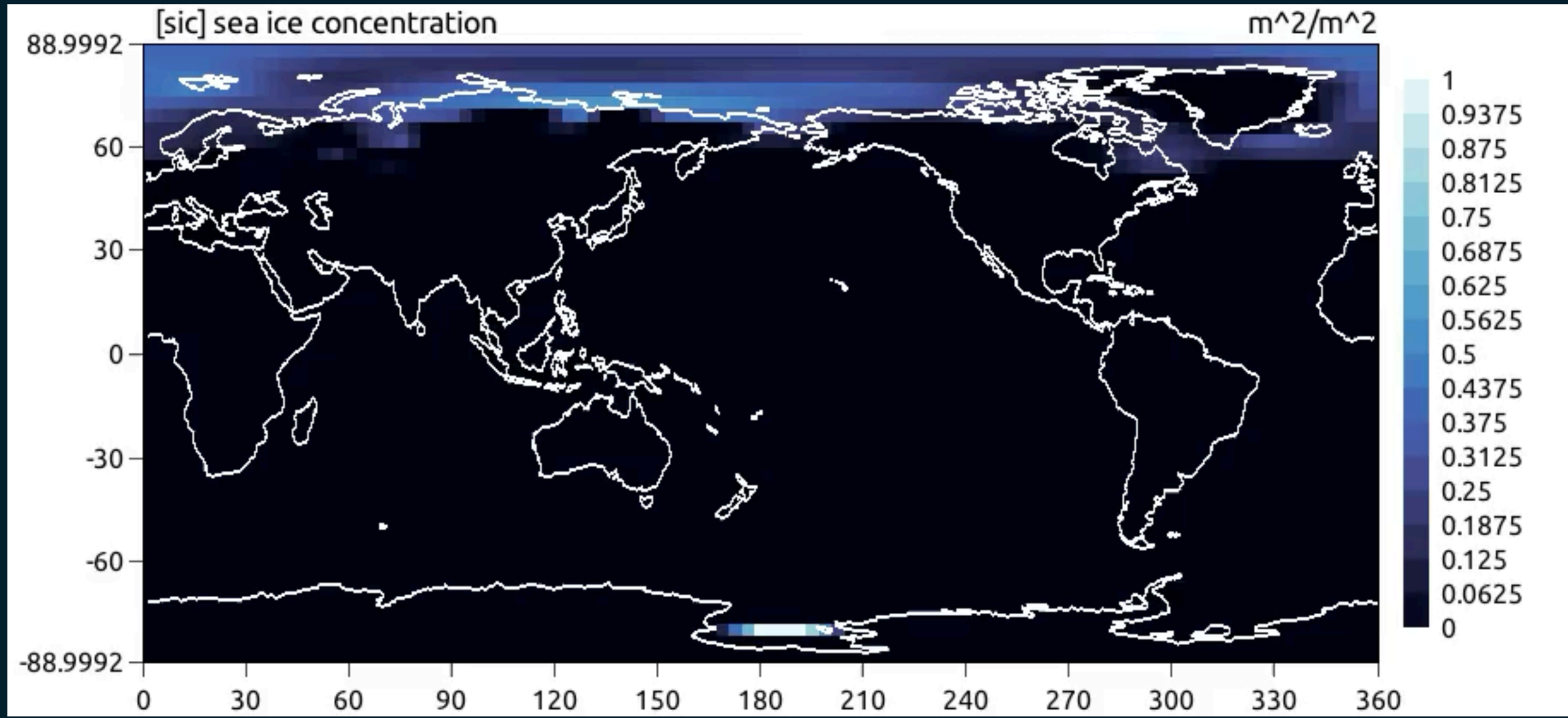


NcVis



Speedy…Climate?

- Slab ocean model (zero velocity)
- Thermodynamic sea ice model
- Land bucket model (temperature, humidity, snow)



SpeedyWeather on the GPU

```julia
## OceanSeaIceAlbedo
export OceanSeaIceAlbedo

"""Albedo that scales linearly between ocean and ice albedo depending on sea ice concentration.
Fields are $(TYPEDFIELDS)"""
@kwdef struct OceanSeaIceAlbedo{NF} <: AbstractAlbedo
    "[OPTION] Albedo over open ocean [1]"
    albedo_ocean::NF = 0.06

    "[OPTION] Albedo over sea ice at concentration=1 [1]"
    albedo_ice::NF = 0.6
end

Adapt.@adapt_structure OceanSeaIceAlbedo
OceanSeaIceAlbedo(SG::SpectralGrid; kwargs...) = OceanSeaIceAlbedo{SG.NF}(;kwargs...)
initialize!(::OceanSeaIceAlbedo, ::PrimitiveEquation) = nothing
@propagate_inbounds function albedo!(ij, diagn, progn, albedo::OceanSeaIceAlbedo, model)
    (; albedo_ocean, albedo_ice) = albedo
    κ = haskey(progn.ocean, :sea_ice_concentration) ? progn.ocean.sea_ice_concentration[ij] : zero(eltype(diagn.albedo))

    # set ocean albedo linearly between ocean and ice depending on sea ice concentration
    diagn.albedo[ij] = albedo_ocean + κ * (albedo_ice - albedo_ocean)
end
```

1 line of “GPU code”

In contrast to: Gordon Bell Prize for climate modelling 2025

Not our idea of "separation of concerns"?

5.2 Separation of Concerns: From Fortran to GPU

To achieve performance and scalability on today's heterogeneous systems, ICON's code base has evolved to support a wide range of configurations: from pragmas enabling OpenMP and OpenACC parallelization, to Intel, Cray, GCC, and NEC vector annotations

dynamical core, the most compute-intensive part of the most expensive atmosphere component, contains 2728 non-empty Fortran source lines of code, of which less than 50 % are actually describing the computation. The remaining lines of code are optimizations

```
      !$ACC PARALLEL DEFAULT(PRESENT) ASYNC(1)          <- OpenACC GPU pragmas
      !$ACC LOOP GANG VECTOR TILE(32, 4)
#ifdef __LOOP_EXCHANGE                                  <- CPU vectorization pragma
      DO jc = i_startidx, i_endidx                       <- duplicate loop header code
!DIR$ IVDEP
        DO jk = 1, nlev
        z_ekinh(jk,jc,jb) =  &
#else
!$NEC outerloop_unroll(4)                               <- NEC-specific pragma
      DO jk = 1, nlev                                    <- duplicate loop header code
        DO jc = i_startidx, i_endidx
        z_ekinh(jc,jk,jb) =  &
#endif
          p_int%e_bln_c_s(jc,1,jb)*z_kin_hor_e(ieidx(jc,jb,1),jk,ieblk(jc,jb,1)) + &
          p_int%e_bln_c_s(jc,2,jb)*z_kin_hor_e(ieidx(jc,jb,2),jk,ieblk(jc,jb,2)) + &
          p_int%e_bln_c_s(jc,3,jb)*z_kin_hor_e(ieidx(jc,jb,3),jk,ieblk(jc,jb,3))

        ENDDO                                            <- some lines down, more OpenMP pragmas
      ENDDO
      !$ACC END PARALLEL
```

Intertwined lines for scientist and "performance engineer"

TravellingSailorProblem.jl

Fly Christmas presents (=particles) to children (=destinations) around the globe following the wind. More points for longer distances particles fly and negative points if they miss it.

- SpeedyWeather+GLMakie+GeoMakie
- Challenge evaluated with GitHub actions
- Used for PhD-fresher teaching
- Optimisation problem
- Wither interactive Makie visualisation

Live demo on Friday!

with Anshul