Ocean simulations with ClimaOcean.jl

Simone Silvestri and the Clima Ocean team

COMMODORE, Boulder, September 12th, 2024

ClimaOcean.jl and Oceananigans.jl

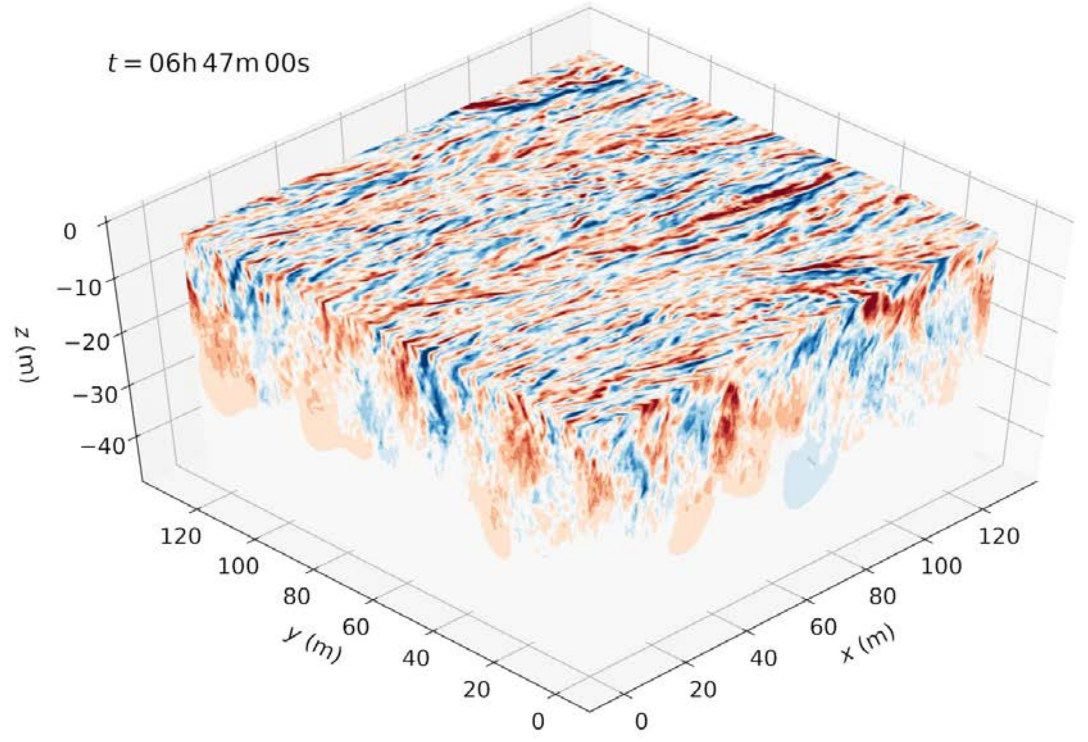
Oceananigans.jl

Finite volume engine

- Grids
- Fields
- Operators

Utilities for numerical experiments

- OutputWriters
- Diagnostics
- Callbacks



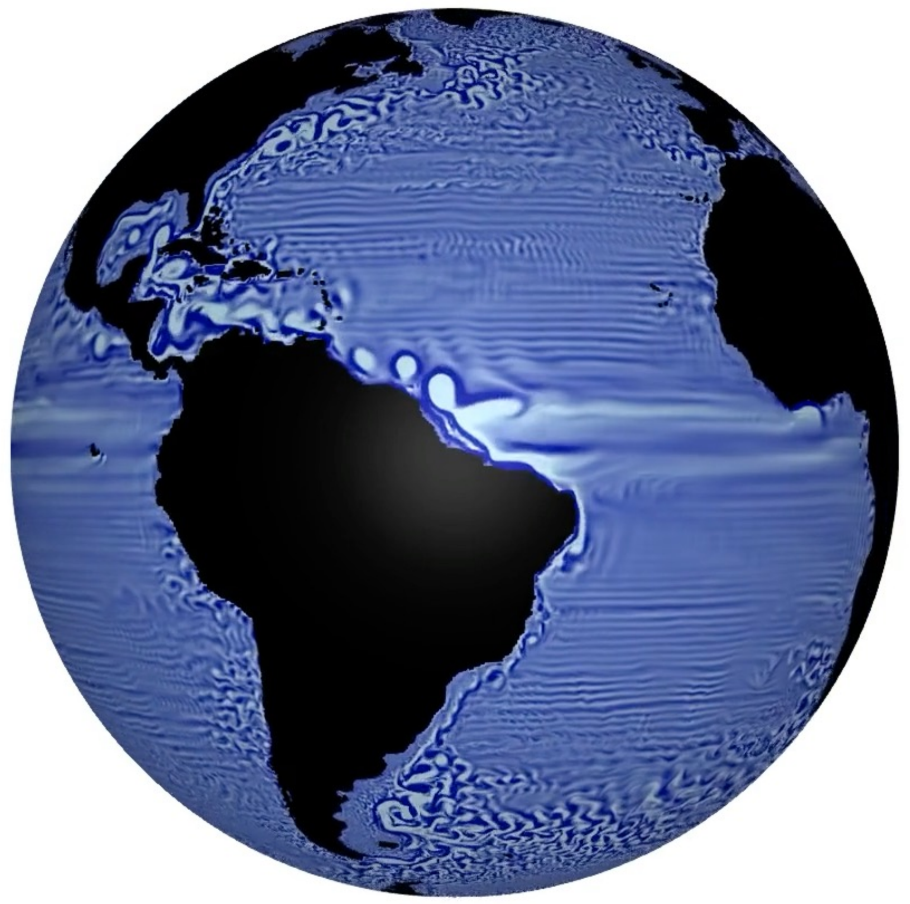
Domain-Specific numerics and physics

- Coriolis, Equation of State, Parameterizations…
- Pressure / free surface solvers…
- Time stepping schemes

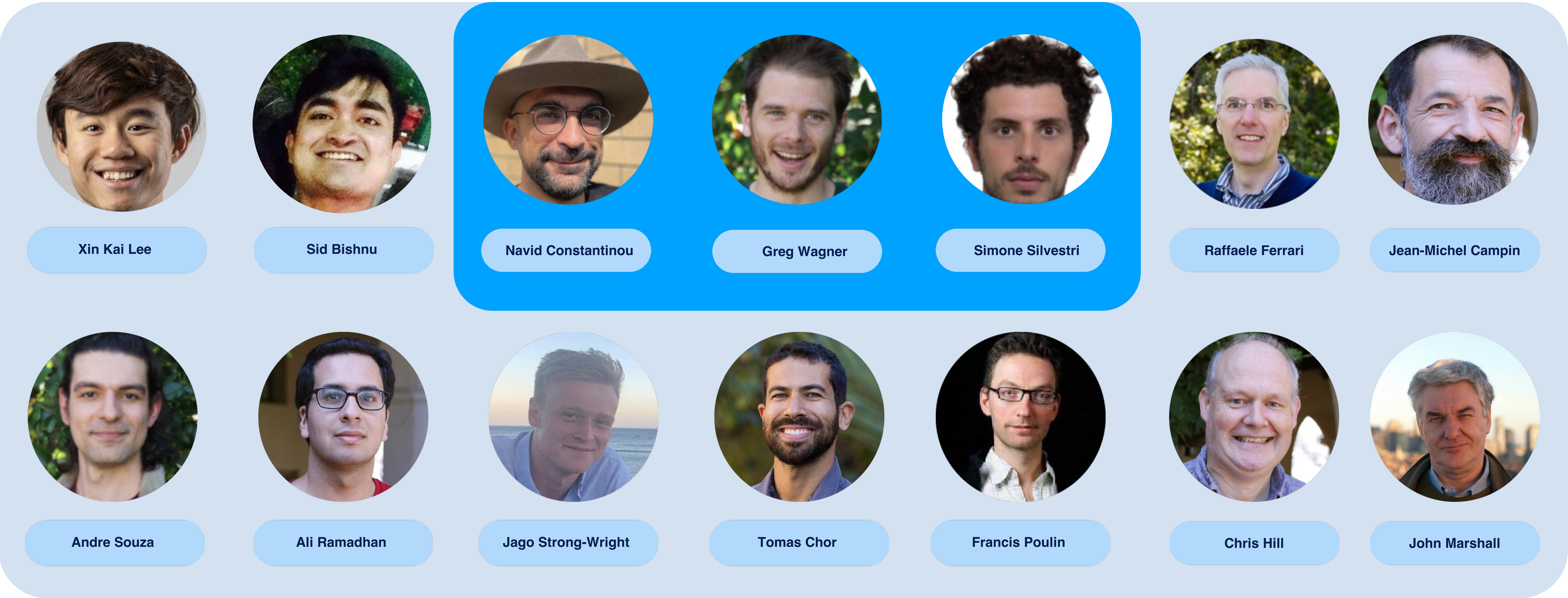
ClimaOcean.jl

Package for ocean-sea-ice simulations

- Bathymetry interpolation
- Surface flux computation
- Ocean-specific Diagnostics



ClimaOcean / Oceananigans developers and advisors



Rewriting a new ocean model

Flexibility and ease of use

+

Computational efficiency

Simulate physics from meter- to global-scale

Support rapid prototyping of parameterizations

Easy to use for process studies

Possibility of high-resolution

Necessary for global calibration

“A fast model can be a good model, but a good model must be a fast model! ***Computational efficiency is crucial….”***

(https://www.gfdl.noaa.gov/fv3/)

Oceananigans: Easy to use and Accessible

all written in julia

Faster than interpreted languages (Python, Matlab)

More flexible than compiled languages (C, Fortran)

Easy portability to virtually any architecture/systems

```julia
using Oceananigans
using GLMakie

grid = RectilinearGrid(CPU(),
                       size = (64, 64),
                       x = (-5, 5),
                       y = (-5, 5),
                       topology = (Bounded, Bounded, Flat))

model = NonhydrostaticModel(grid=grid, tracers=:c, advection=WENO())

gaussian(x, y) = exp(-(x^2 + y^2))
set!(model, c=gaussian)

c = model.tracers.c

∇c² = ∂x(c)^2 + ∂y(c)^2
∇c² = Field(∇c²)
compute!(∇c²)
```

Try changing CPU() to GPU()

Initial conditions

Diagnostics

Starred 923

Used in more that 20 scientific papers
10 from the MIT group

55+ contributors to the codebase

"...I have never experienced getting a useful calculation done as easily as I was able to do with Oceananigans. It not only has a sophisticated interface, but it is remarkably fast...".

Linux magazine

User interface:

- Programmatic vs namelist
- Designed so code "reads like a paper"

Ramadhan et al, JOSS, 2020

Dynamical core algorithmic implementation 🚀

fast compute

Make sure the computations take advantage
of the parallel power of GPUs

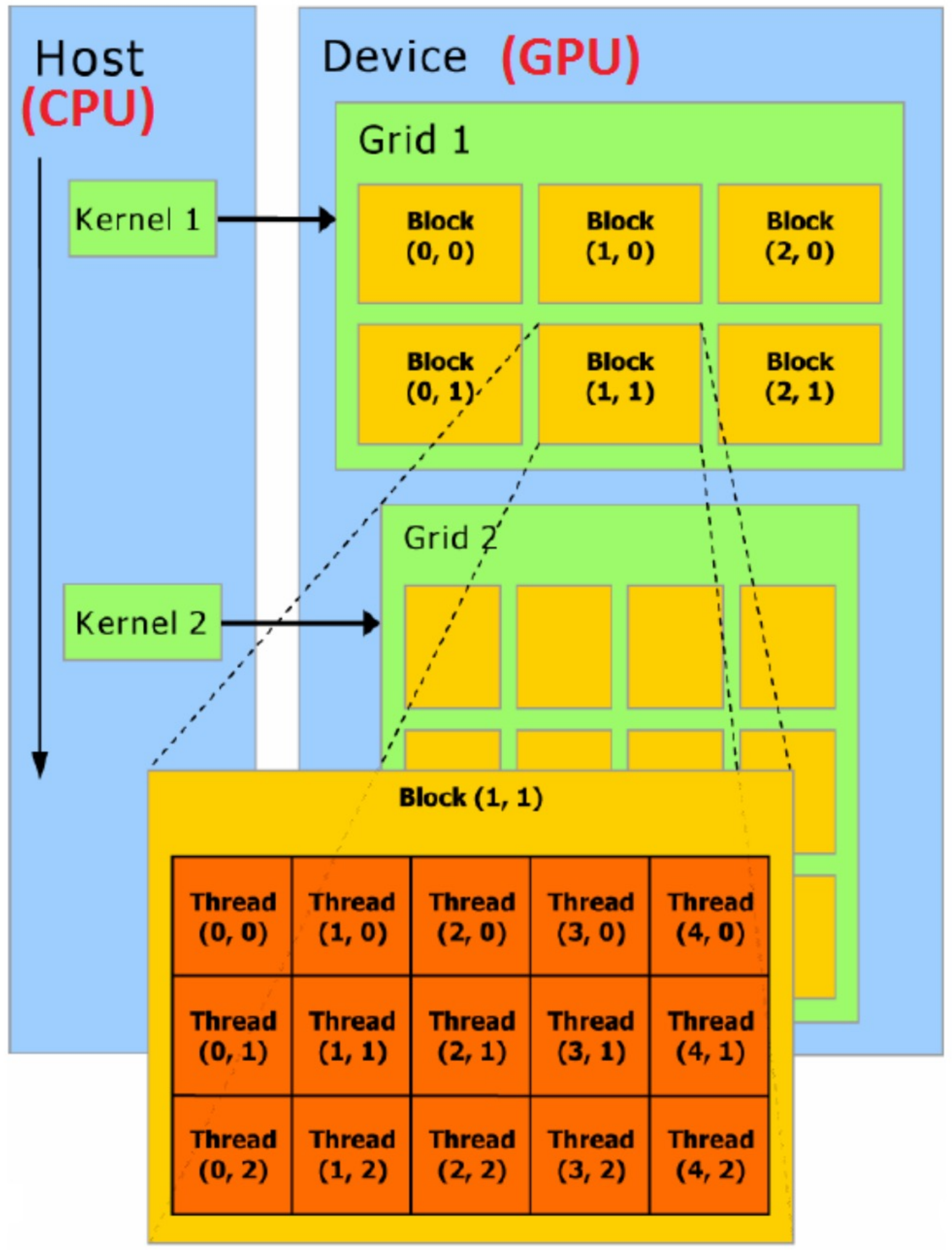
memory leanness

minimise temporary array creations
loop over the domain as few times as possible

scalable solvers

compute is fast, communication is expensive,
overlap communication with computation

GPU Parallelization



GPU execution model:
expose parallelization!

$$\frac{\partial \boldsymbol{u}_h}{\partial t} = G_{\boldsymbol{u}} - g\nabla\eta + \frac{\partial}{\partial z}\kappa_u\frac{\partial \boldsymbol{u}_h}{\partial z}$$

$$\frac{\partial w}{\partial z} = -\nabla\cdot\boldsymbol{u}_h$$

$$\frac{\partial p}{\partial z} = b$$

$$\frac{\partial \boldsymbol{U}_b}{\partial t} = \dots \qquad \frac{\partial \eta}{\partial t} = \dots$$

$$\frac{\partial \theta}{\partial t} = G_\theta + \frac{\partial}{\partial z}\kappa_c\frac{\partial \theta}{\partial z}$$

$$\frac{\partial S}{\partial t} = G_S + \frac{\partial}{\partial z}\kappa_c\frac{\partial S}{\partial z}$$

3D computation of tendencies

3D kernel: each thread holds a computational cell

Implicit vertical diffusion

2D kernel: each thread holds a computational column

W and P integral computation

2D kernel: each thread holds a computational column

2D barotropic solver

2D kernel: each thread holds a computational cell

Memory Leanness

advection of temperature

$$\partial_t T = -\frac{\partial uT}{\partial x} - \frac{\partial vT}{\partial y} - \frac{\partial wT}{\partial z} + \ldots$$

"classic" Fortran-style temp arrays on CPU are cheap

temporary array
temporary array
temporary array

```
uT = compute_x_fluxes(u, T)
vT = compute_y_fluxes(v, T)
wT = compute_z_fluxes(w, T)
rhs_T = flux_divergence(uT, vT, wT)
```

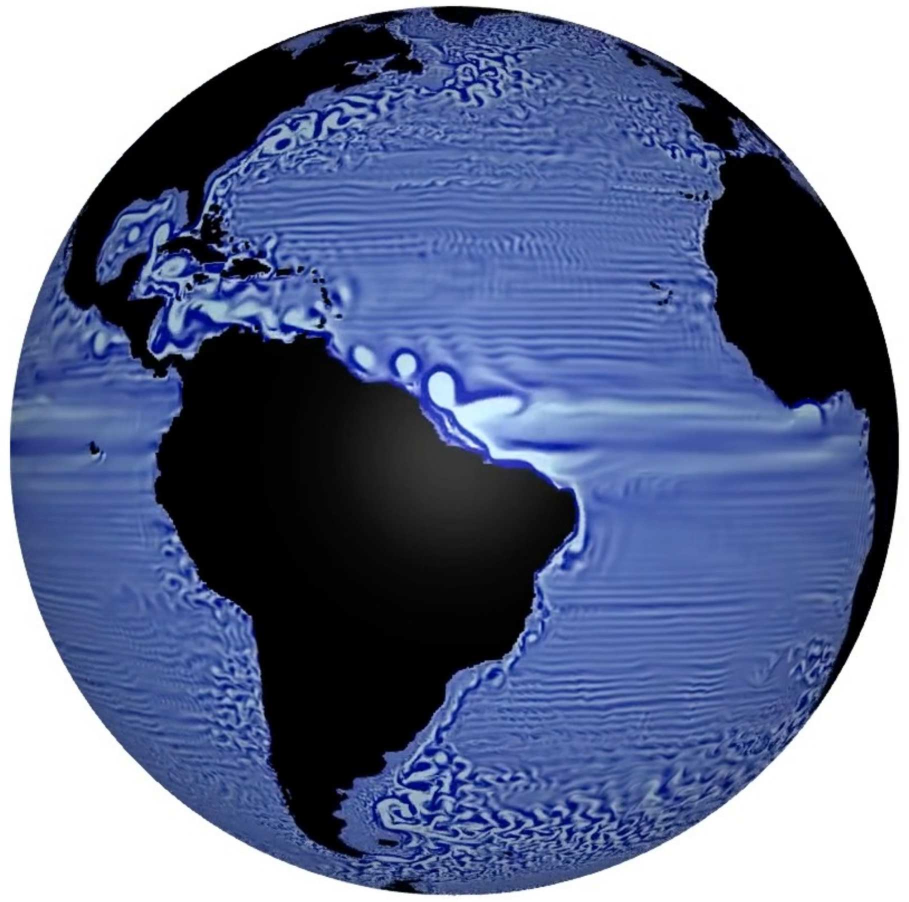
Oceananigans
GPU-friendly kernel fusion
no memory allocation

```
tendency_T(i, j, k) = - div_uT(i, j, k) - div_vT(i, j, k) - div_wT(i, j, k)

@kernel function calculate_tendency_T!(rhs_T)
    i, j, k = @index(Global, NTuple)
    rhs_T[i, j, k] = tendency_T(i, j, k)
end
```

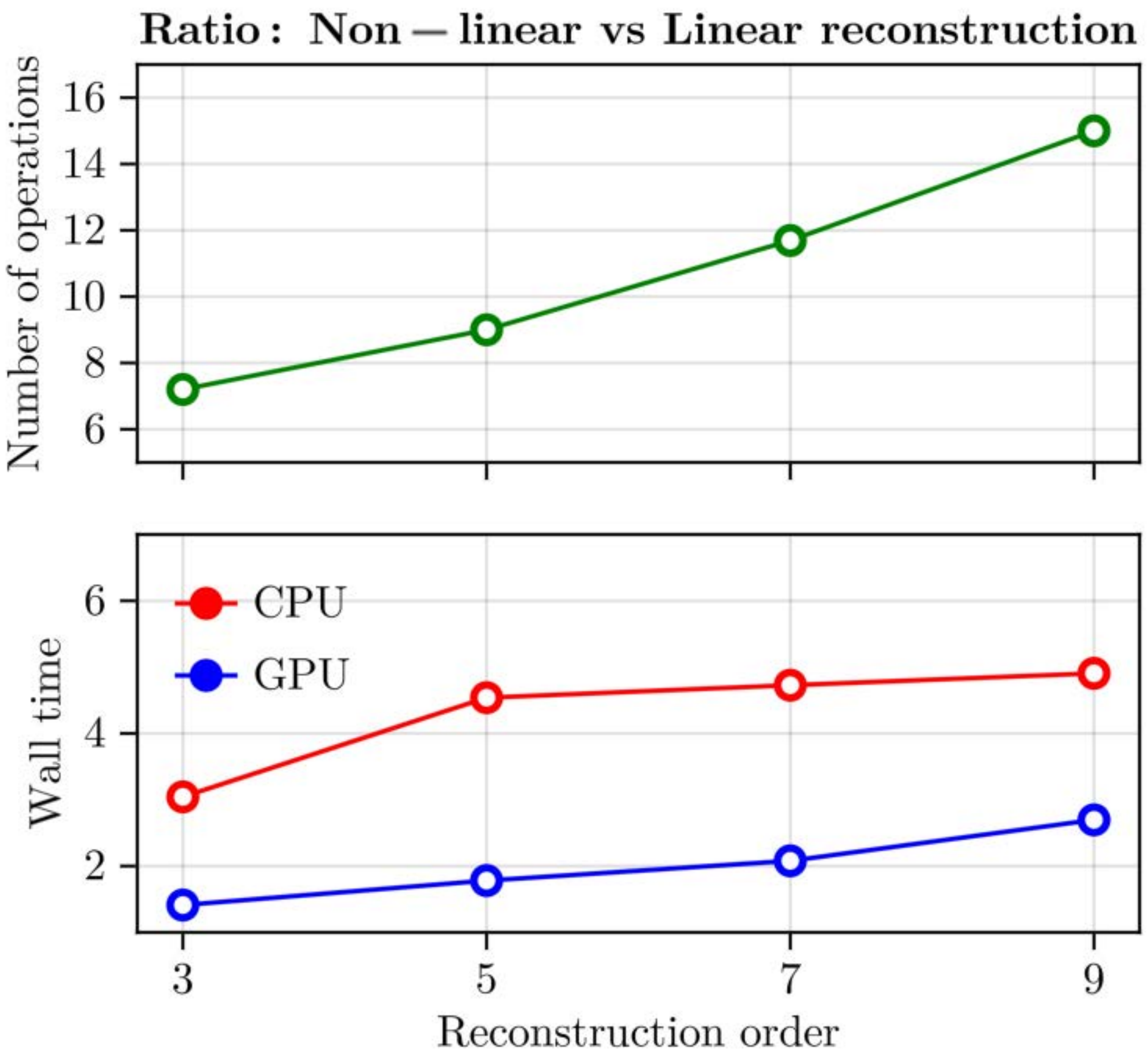
we launch as few kernels as possible:
only one for the tendency of each prognostic quantity

1/4° horizontal resolution
50 vertical levels
15 GB memory footprint



fits ***easily*** on 1 Nvidia V100 GPU

Compute bound numerical schemes?



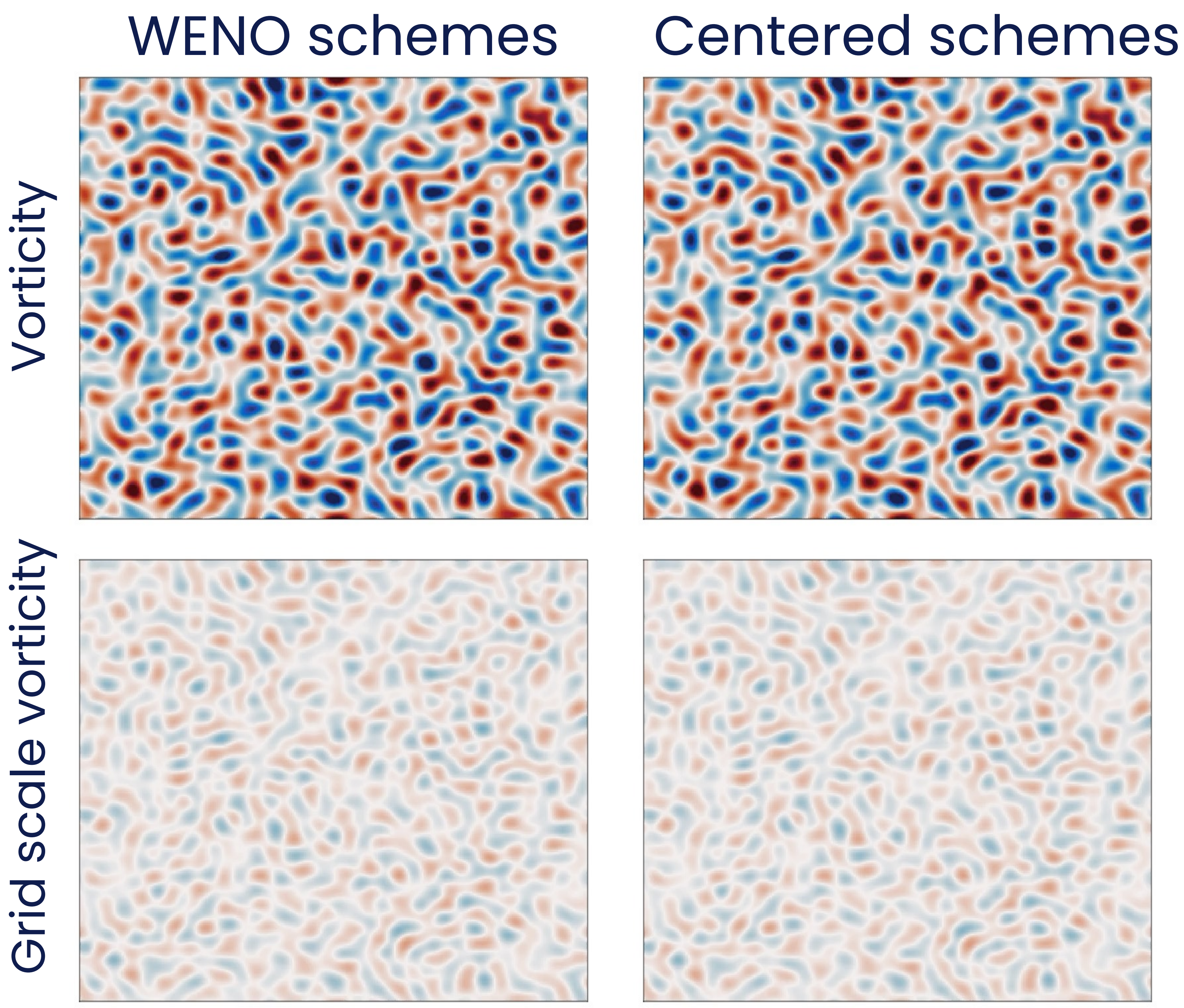
1. Memory load are costly
2. Computations are cheap

→ Once memory is loaded in registers flops (add, multiply especially) are not too costly.

→ Maximize Flops per retrieved byte

→ Optimize for register pressure and memory loads

WENO reconstruction schemes



Non-linear numerical reconstruction

$$\tilde{c}_{i+1/2} = \sum_S \omega^S \tilde{c}^S_{i+1/2}$$

$$\omega^S = f(c_{i-N} \ldots c_{i+N})$$

$$\tilde{c}^S_{i+1/2} = \textit{linear reconstruction within stencil } S\ldots$$

Why they are appealing:

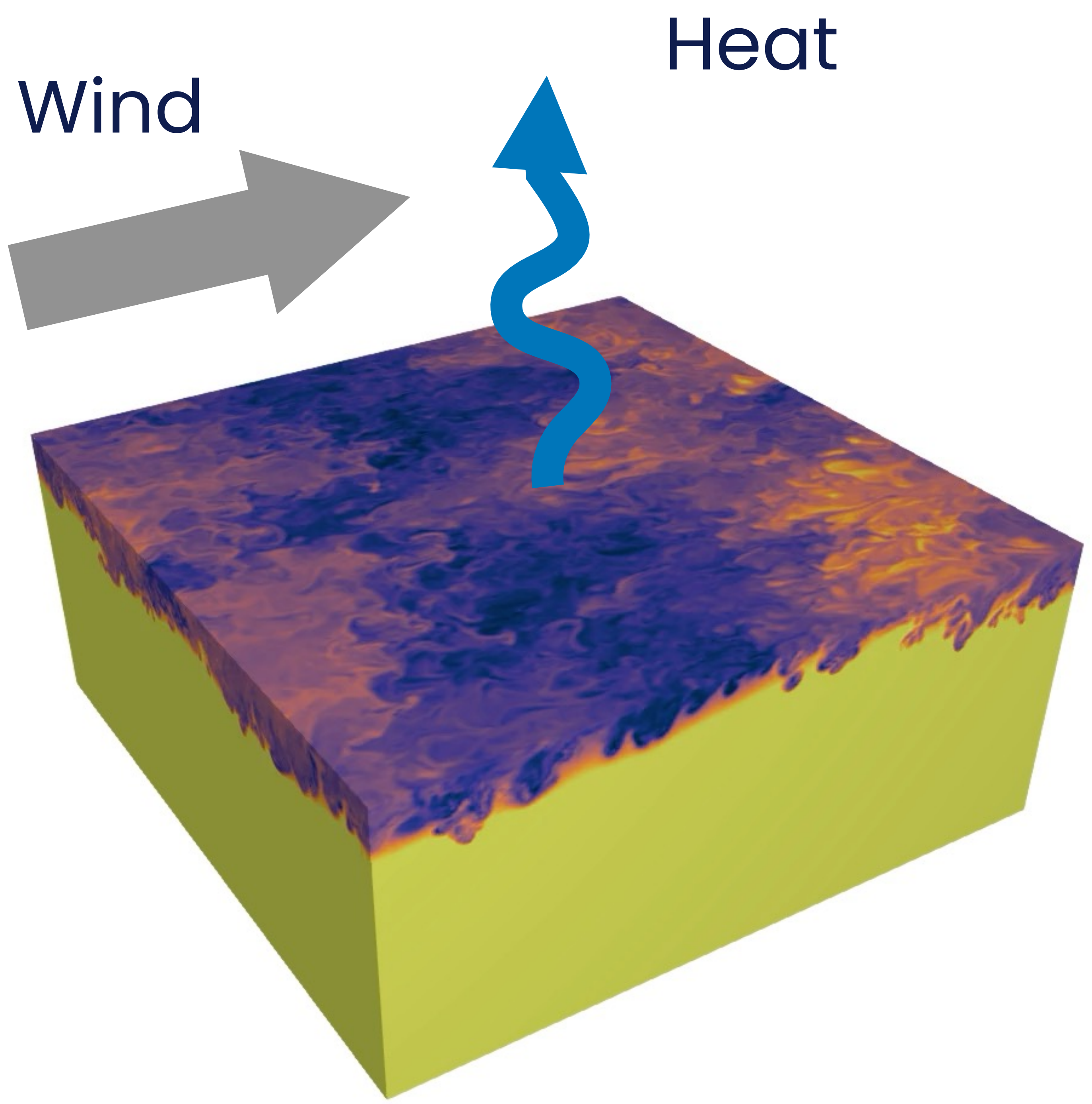
- avoid explicit diffusion
- preserve gradients
- minimal diffusion with minimal noise?

Important for mesoscale resolving simulations?

What is the downside:

- low control on the dissipation
- diapycnal diffusivity?

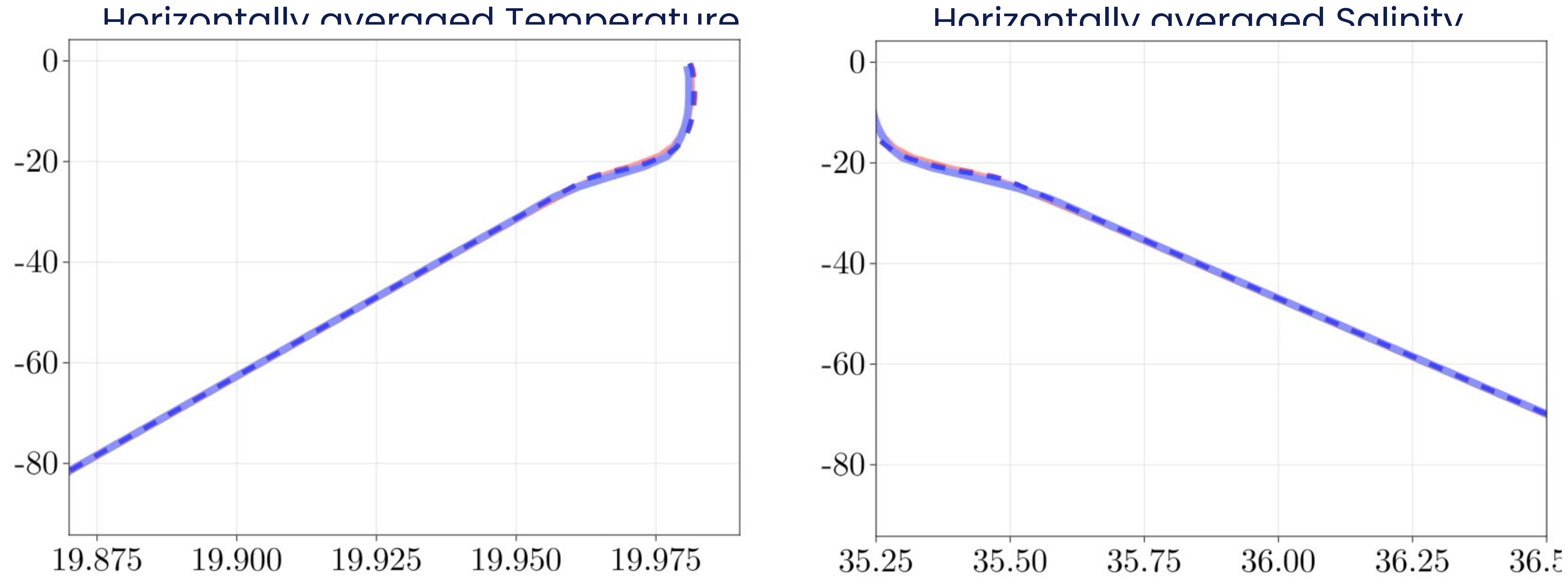
Upper ocean mixing: WENO vs SGS



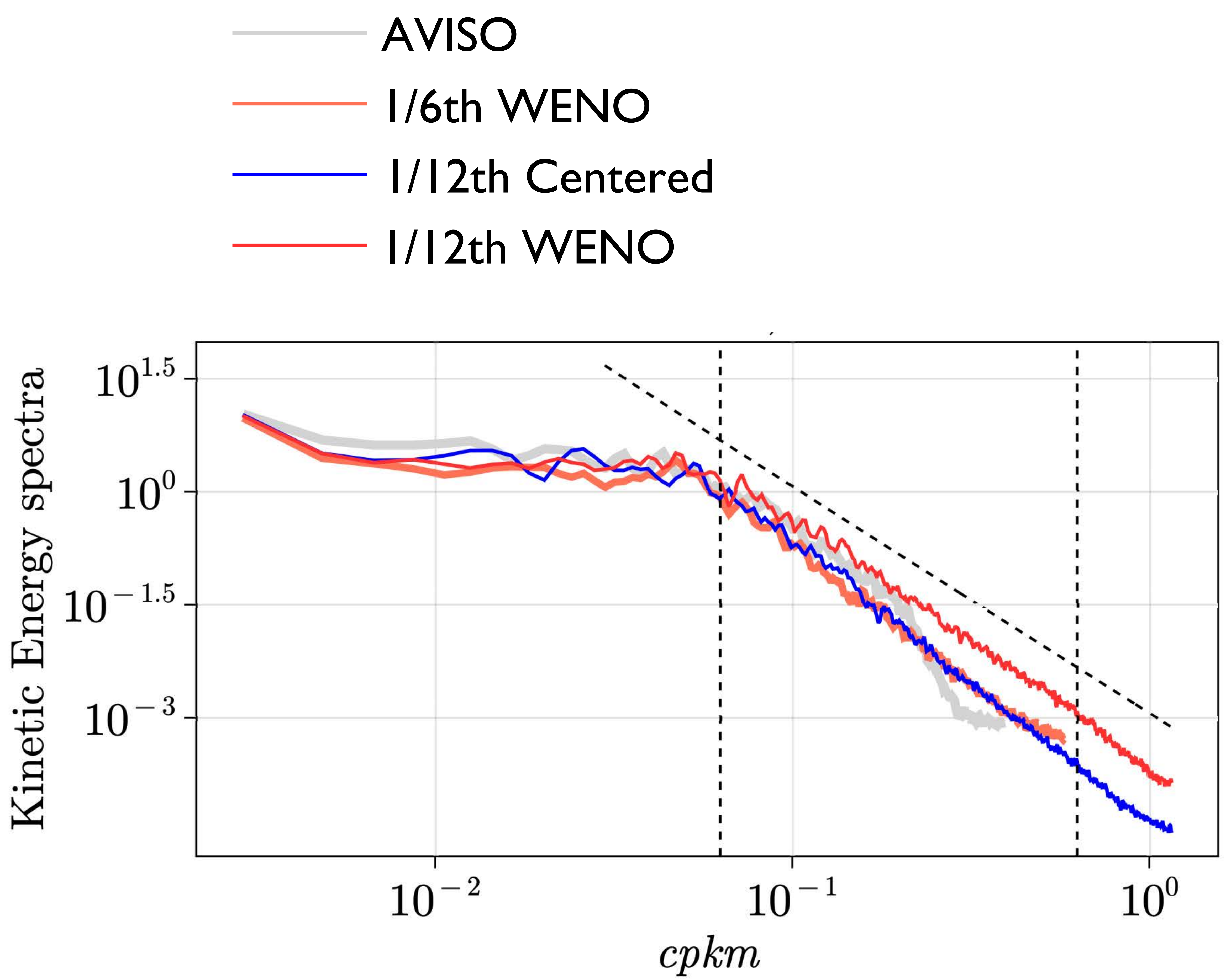
Instantaneous temperature in the 0.5 meter resolution case

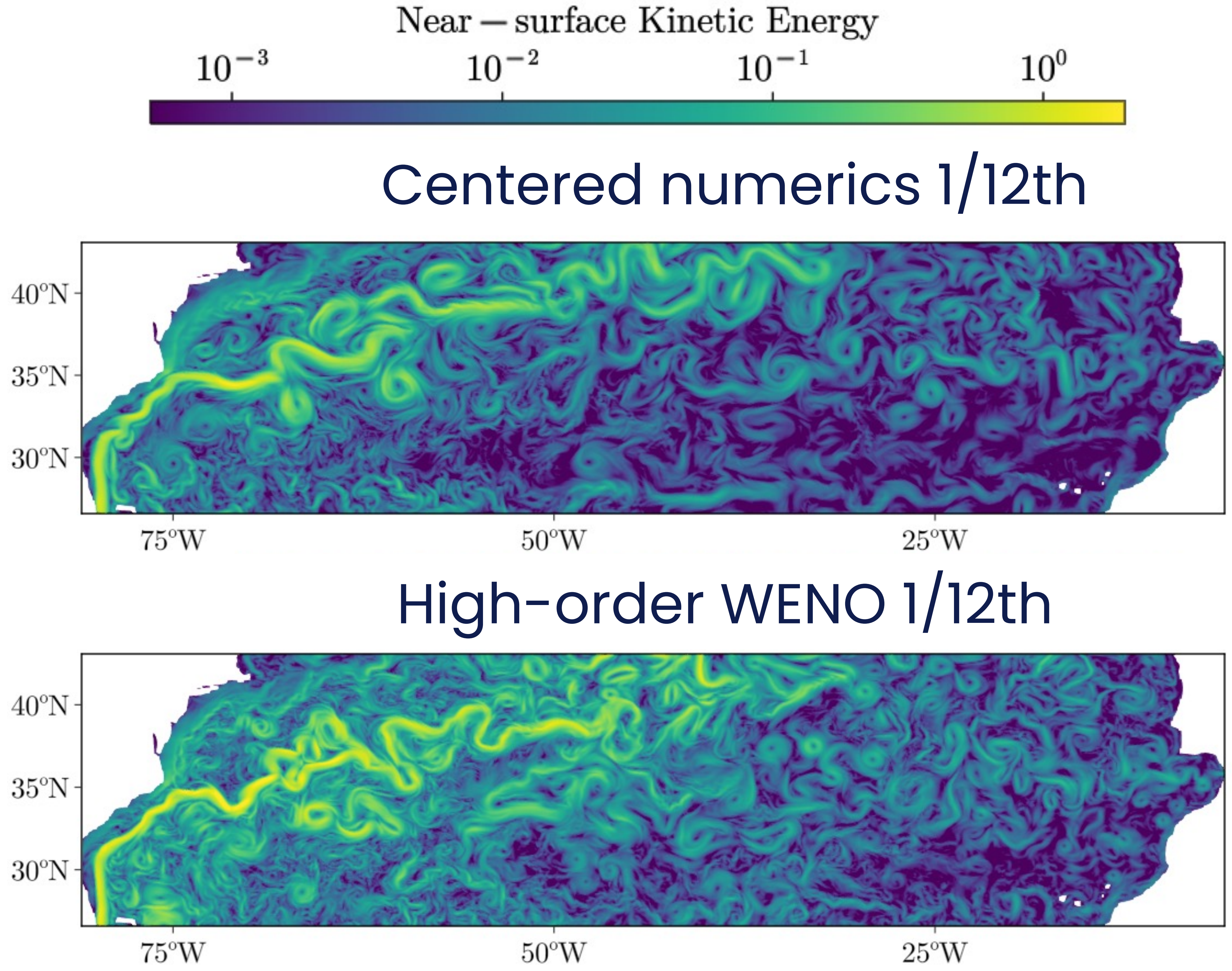
Coarse grid: **2** meters
Fine grid: **0.5** meters

Coarse grid: **SGS**
Coarse grid: **WENO9**
Fine grid: **SGS**
Fine grid: **WENO9**



WENO reconstruction schemes



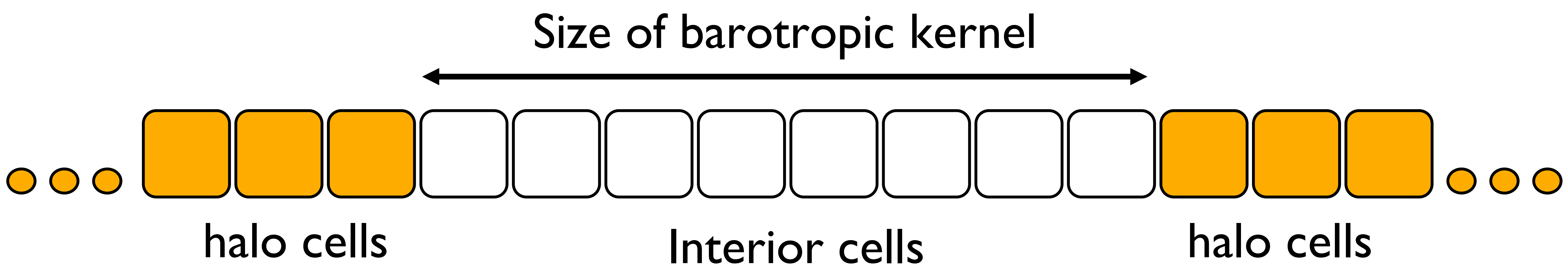


Multi-GPU parallel implementation

Hide GPU-GPU communication!

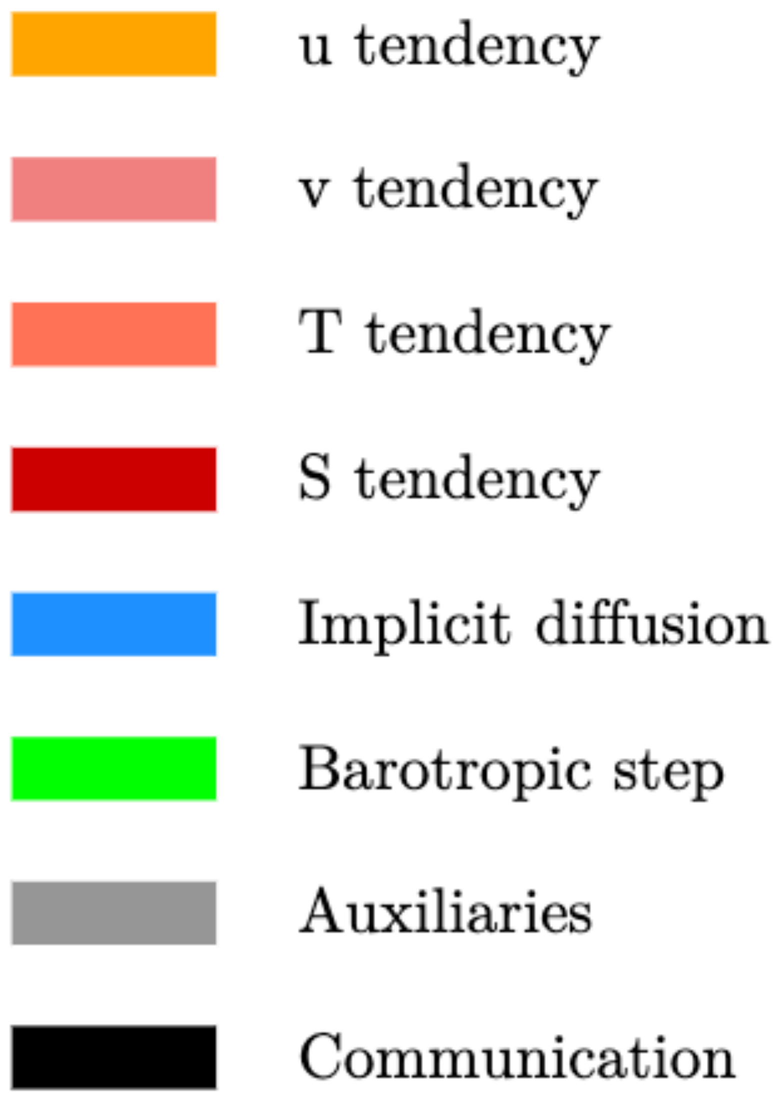
Easy for 3D baroclinic variables

Take advantage of memory leanness for the 2D barotropic solver

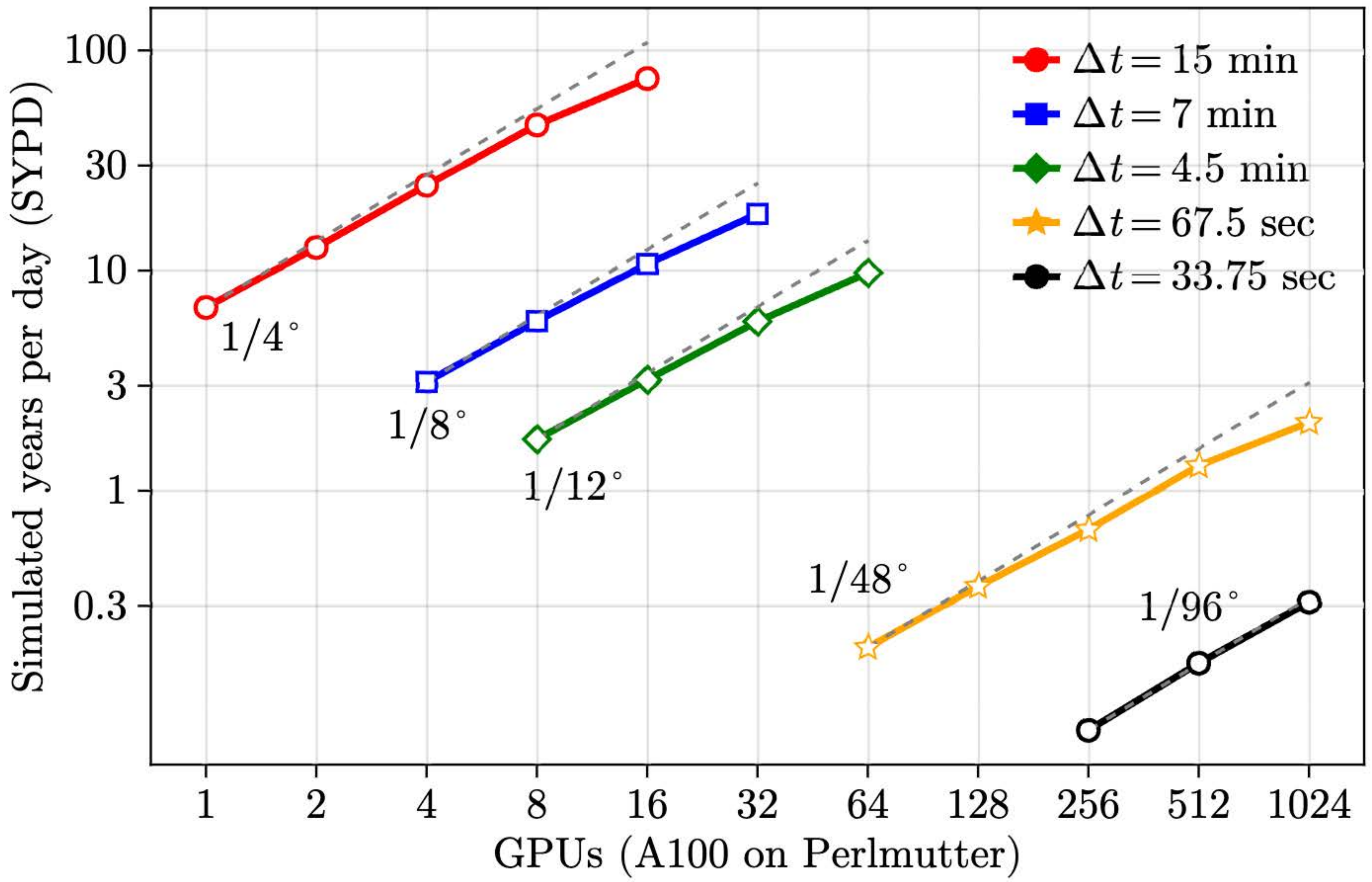


$1/4^{th}$ resolution, 4 GPUs

$1/48^{th}$ resolution, 256 GPUs



Scaling performance (dynamical core)



1/4th degree: **> 25 SYPD** on **4 GPUs**

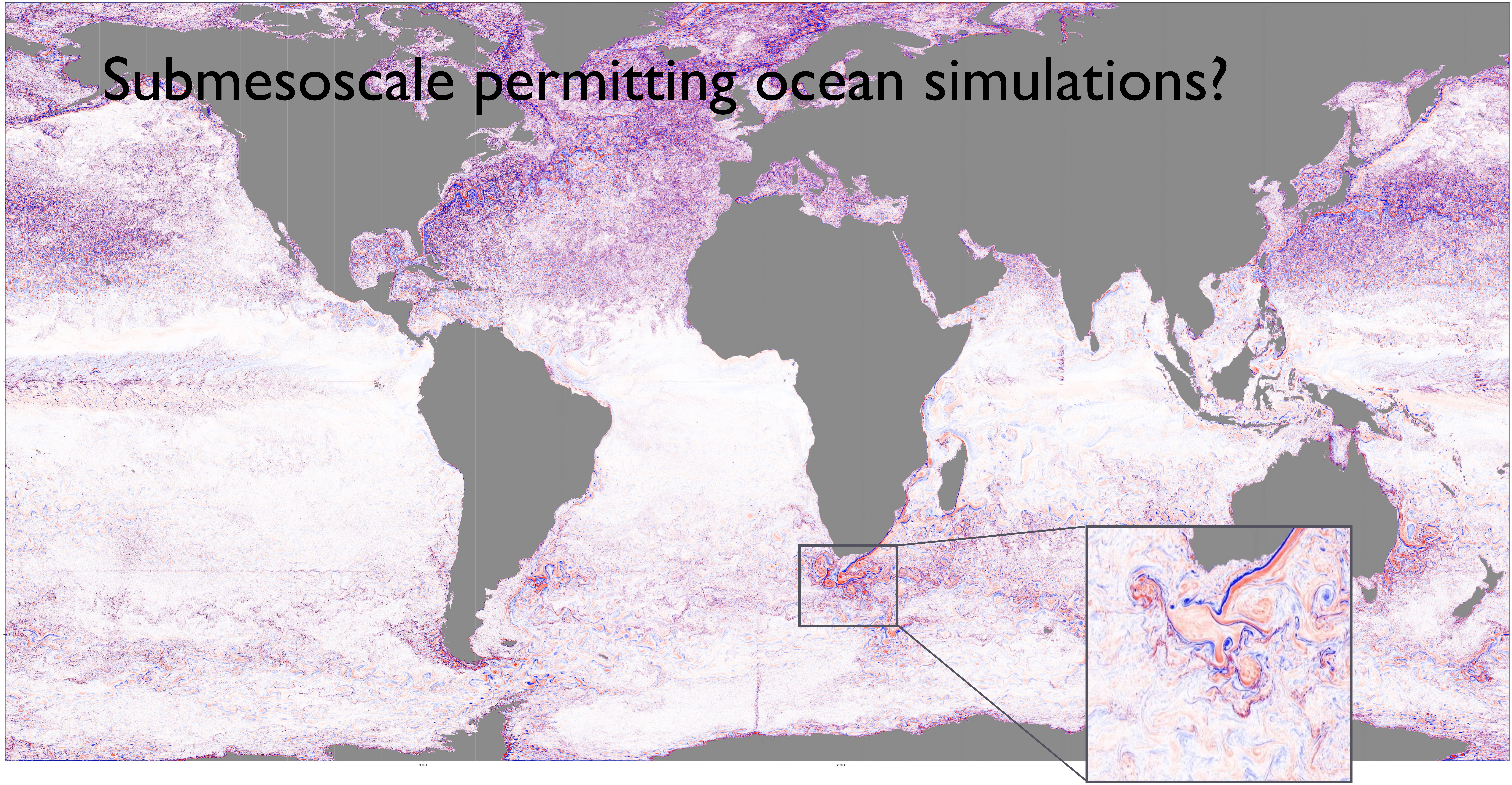
1/12th degree: **10 SYPD** on **64 GPUs**

1/48th degree: **> 1 SYPD** on **512 GPUs**

Possible bottlenecks to optimize in ClimaOcean

- Asynchronous I / O : passing memory between GPU and CPU is heavy!
- Surface fluxes computation
- Sea ice?

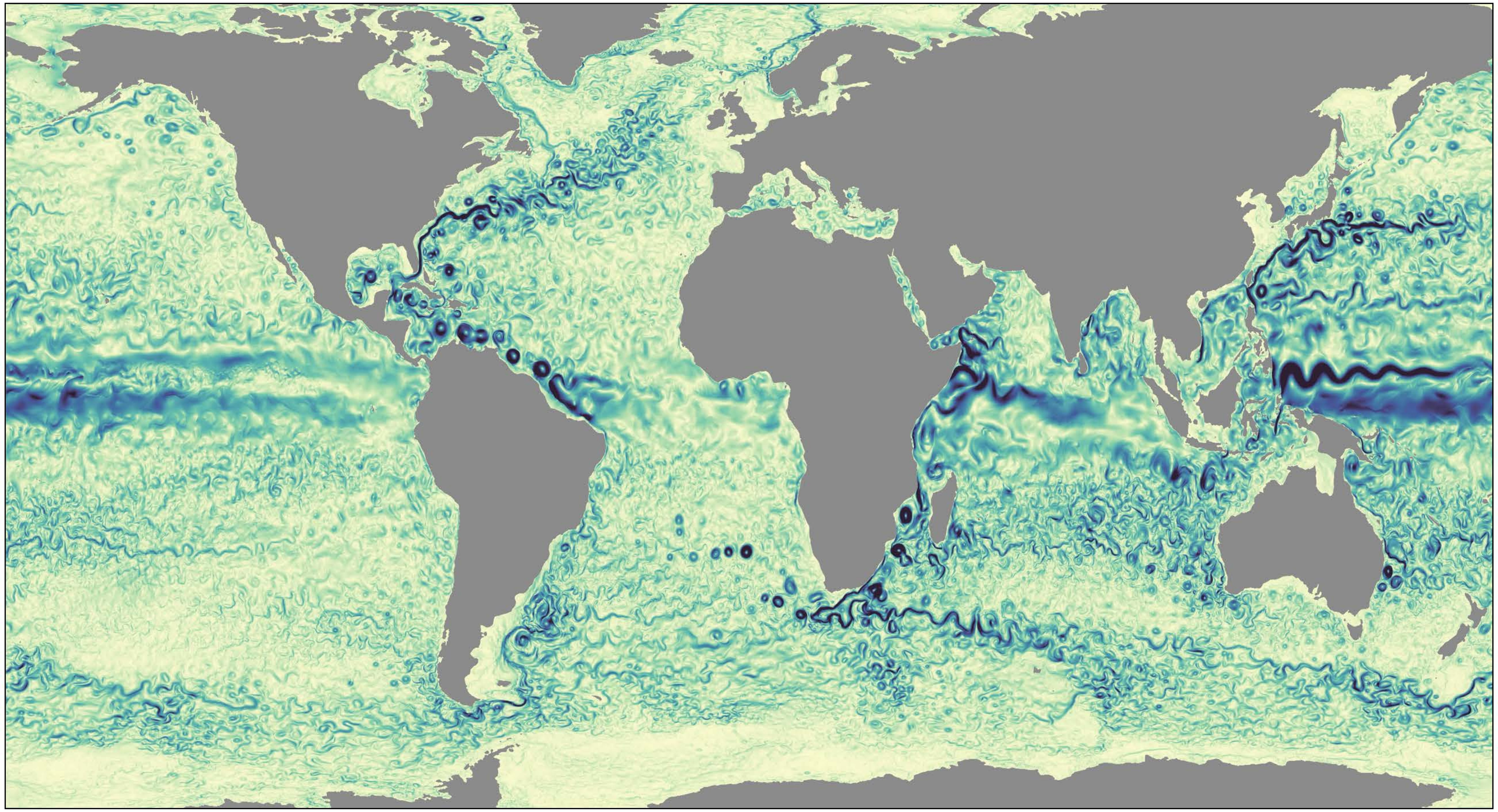
Submesoscale permitting ocean simulations?



1/48° horizontal resolution + 100 vertical levels ~ 2×10^{10} points

Run on 32 Nvidia A100 GPUs

Mesoscale resolving ocean simulations

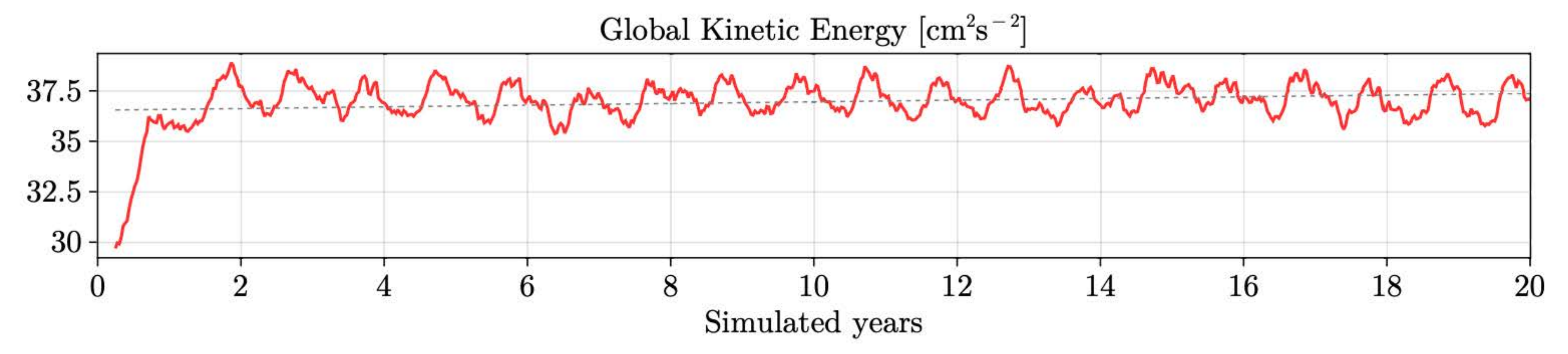


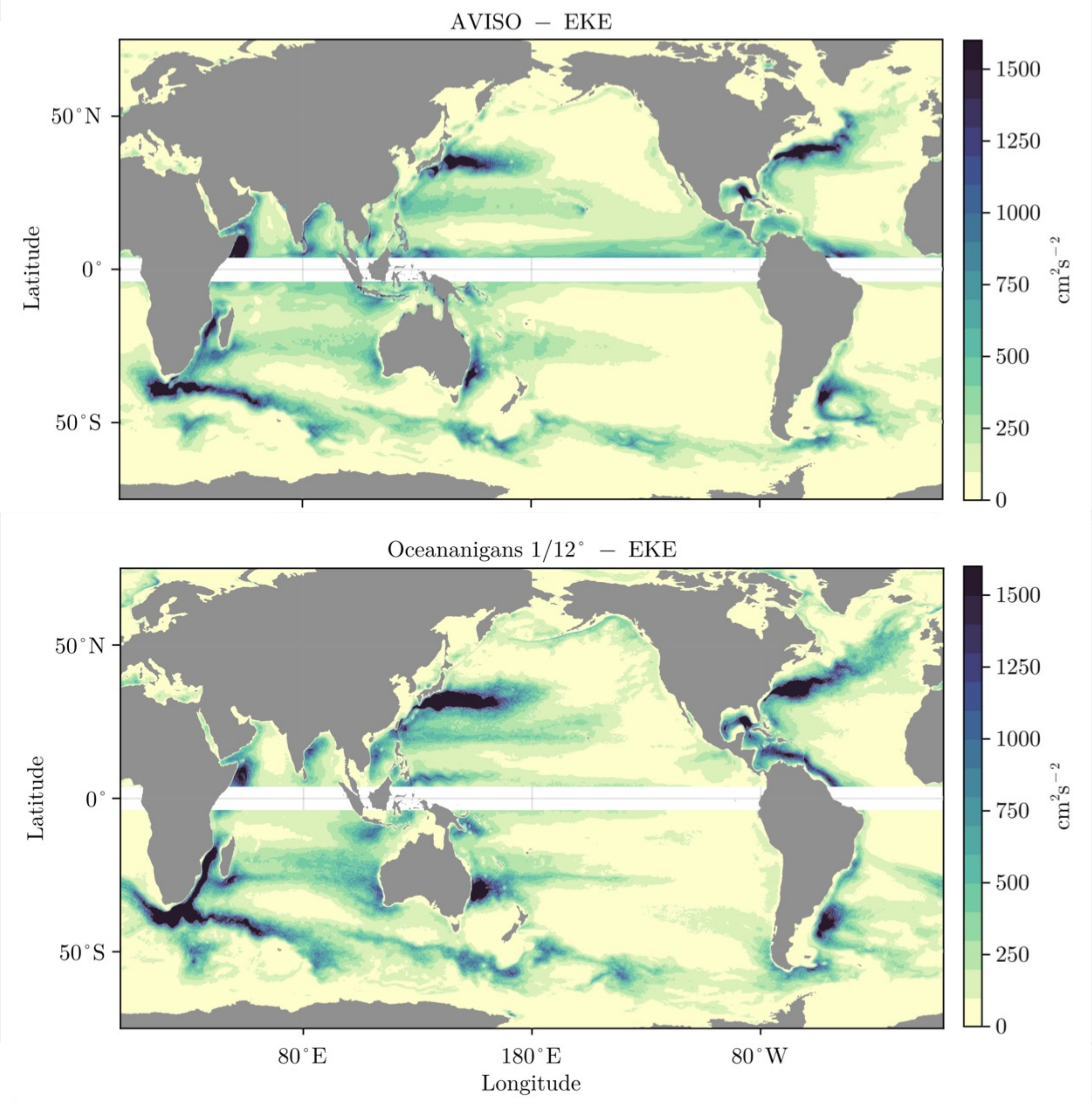
Near-global ocean simulation at 1/12º with
100 vertical levels on **2 GPU nodes** (~ 1.5 SYPD) - 8 GPUs

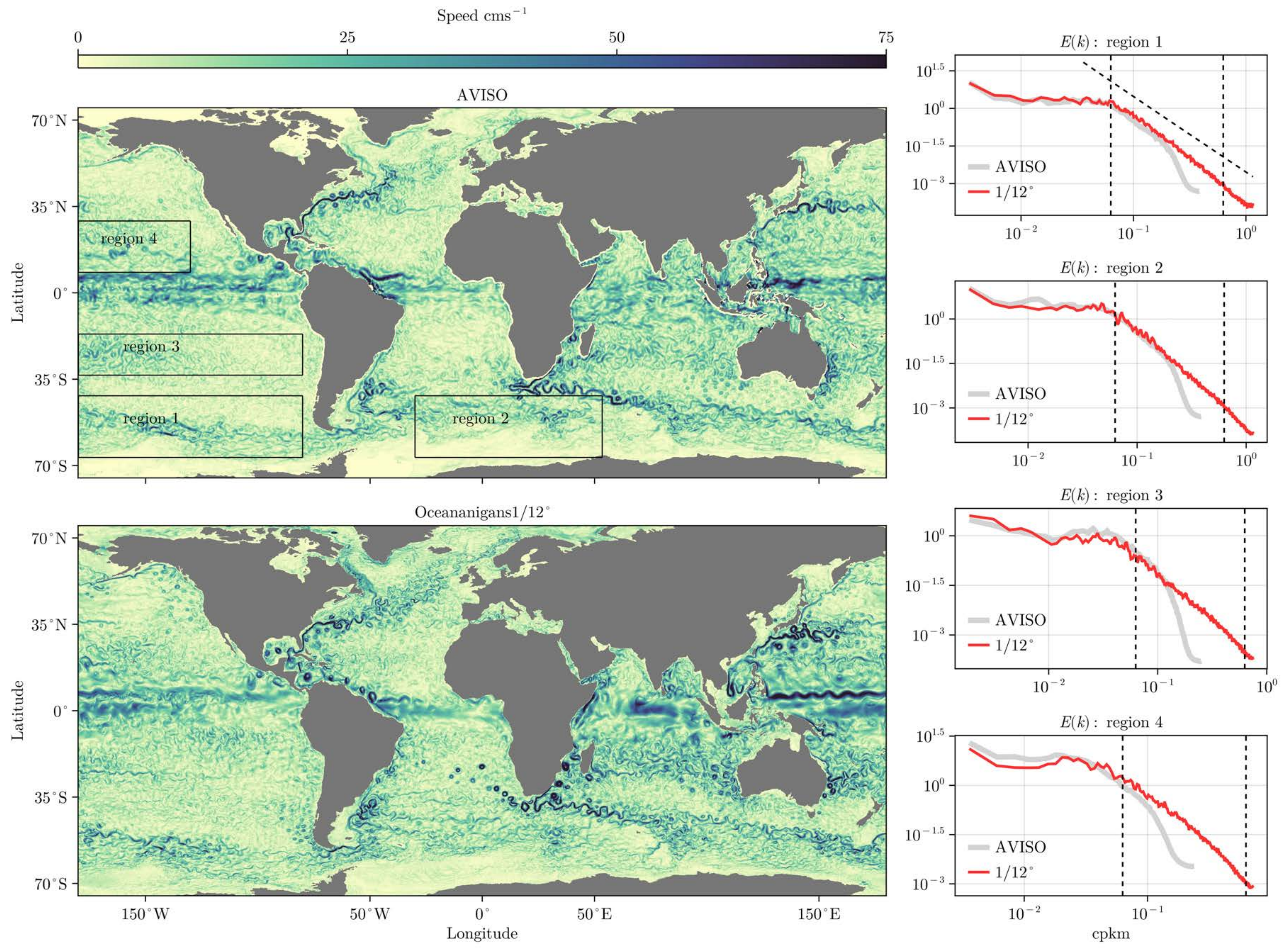
Testbed for performance and stability

- Surface Forcing:
  - Prescribed fluxes
  - Restoring (T, S)
- 20 years run
- Semi-idealized

Kinetic energy in the model

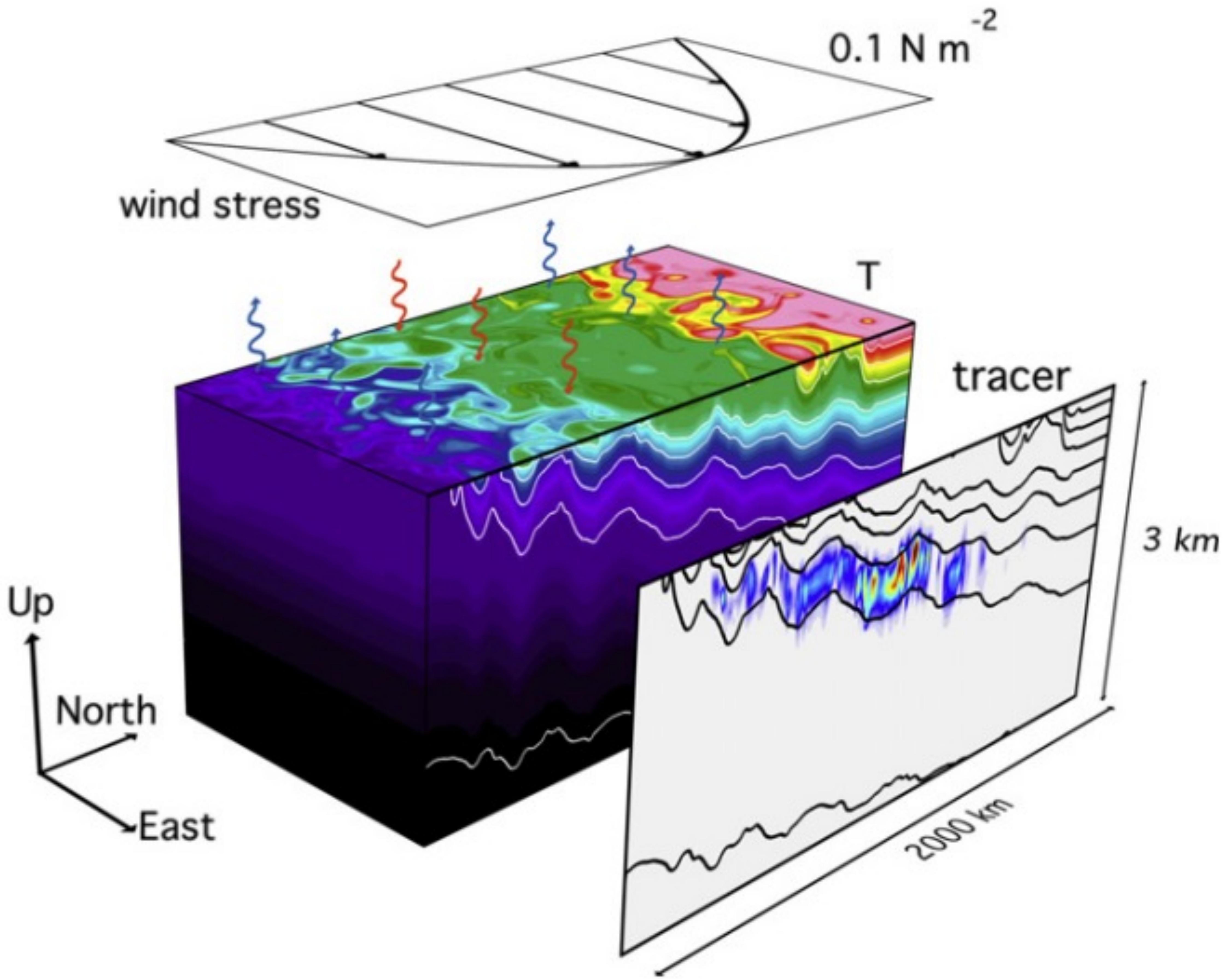






- Captures well mesoscale turbulence in turbulent regions
- Bias in large scale pattern of EKE

Diapycnal mixing in re-entrant channel



Hill et al. 2012, Ocean Modelling

Dissipation due to diffusive flux

$$P_x^n = \mathcal{D}_x \delta_x (T^{n+1} - T^n)$$

Dissipation due to advective flux

$$P_x^n = \mathcal{A}_x \delta_x (T^{n+1} - T^n) - U \delta_x (T^{n+1} T^n)$$

Allows calculating pointwise dissipation caused by implicit numerical schemes.

Configuration:

- Restoring at the north
- Differential heating and cooling
- Parabolic zonal wind stress
- No background diffusivity

Diapycnal mixing in re-entrant channel

Case 1

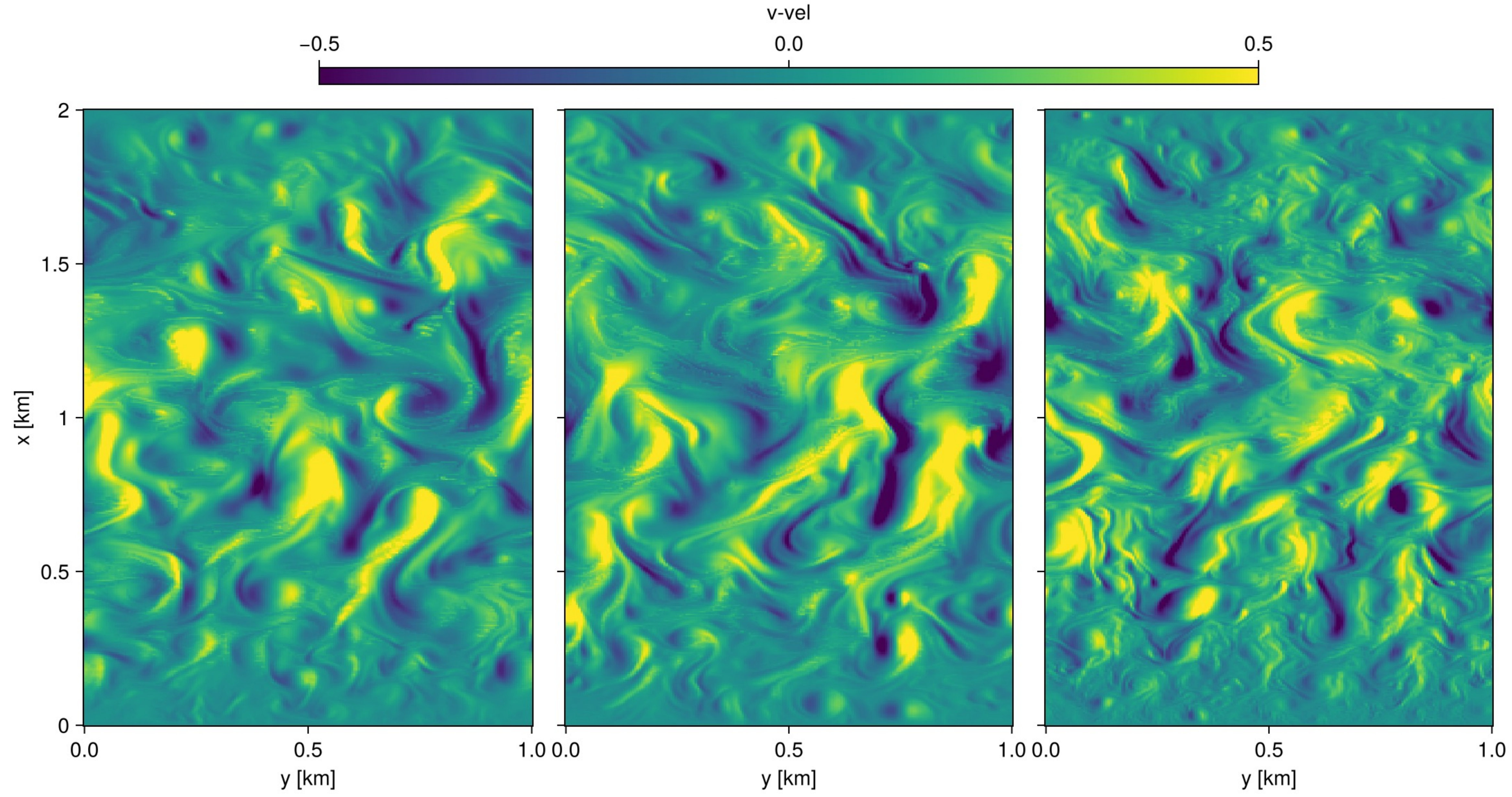
- Momentum:
  - 2nd Order
- Tracer:
  - Linear 3rd Order

Case 2

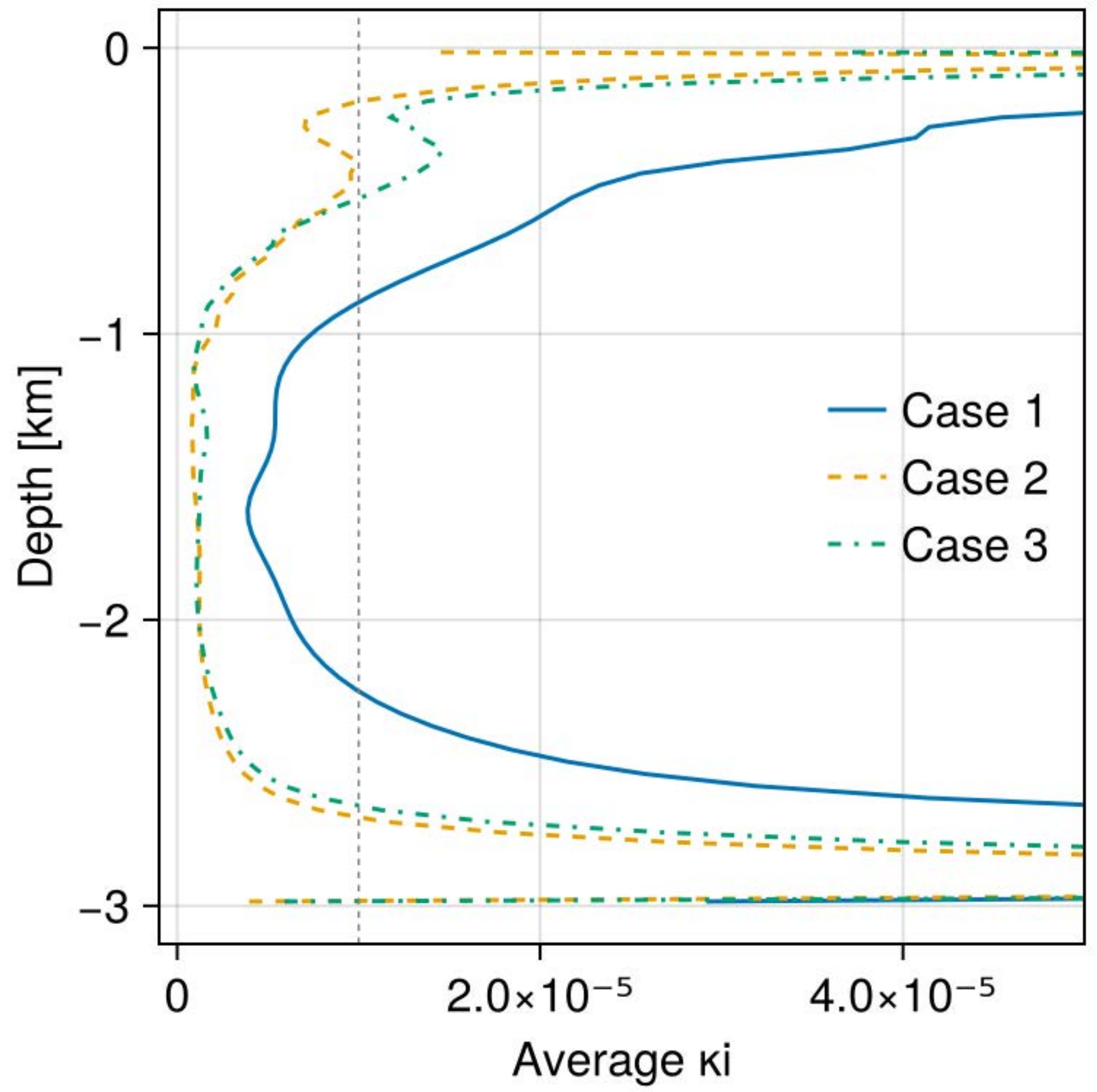
- Momentum:
  - 2nd Order
- Horizontal Tracer:
  - WENO 9th order

Case 3

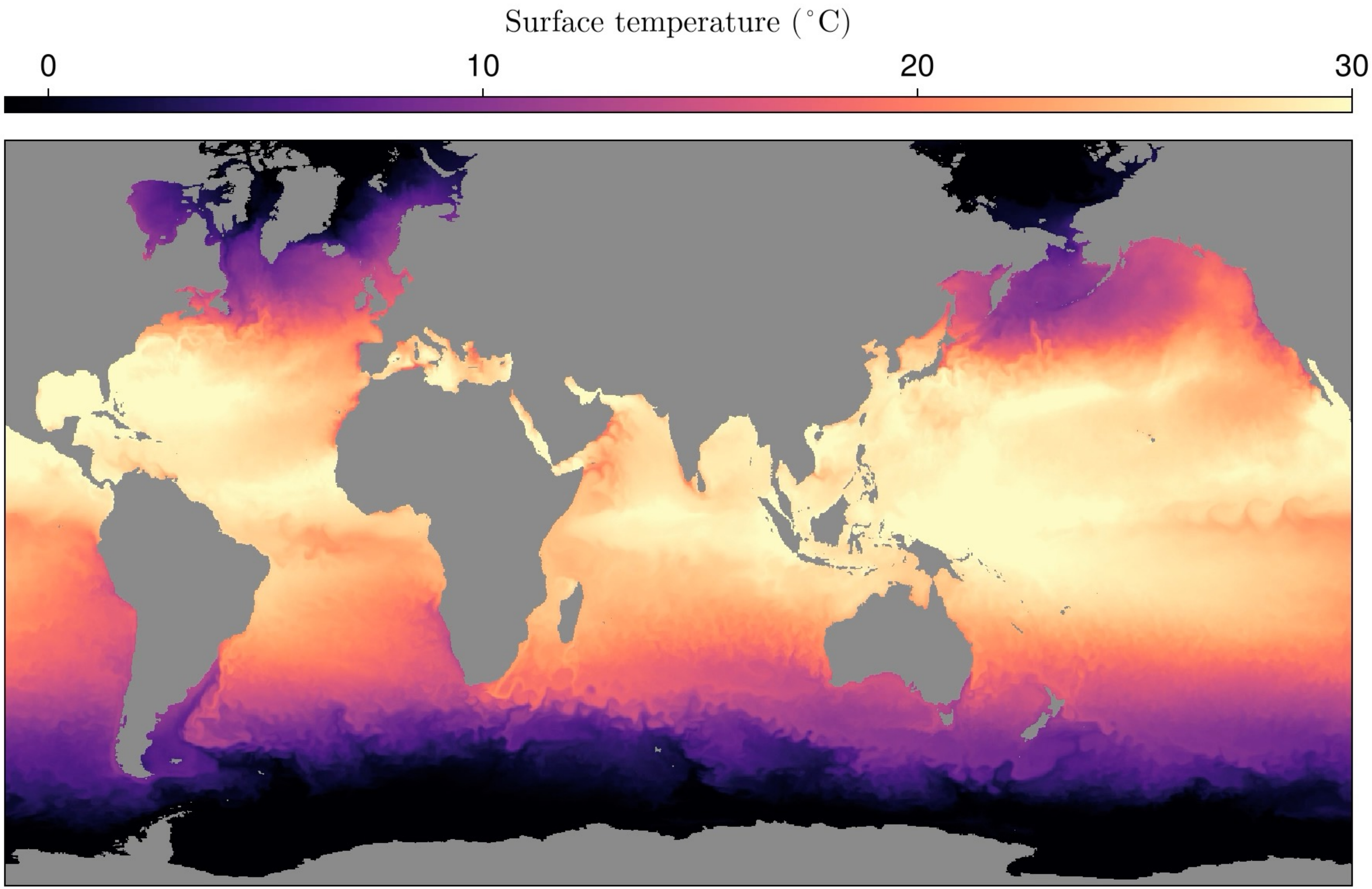
- Momentum:
  - WENO (vector invariant)
- Horizontal Tracer:
  - WENO 9th order



Case 1 Case 2 Case 3



Moving forward?

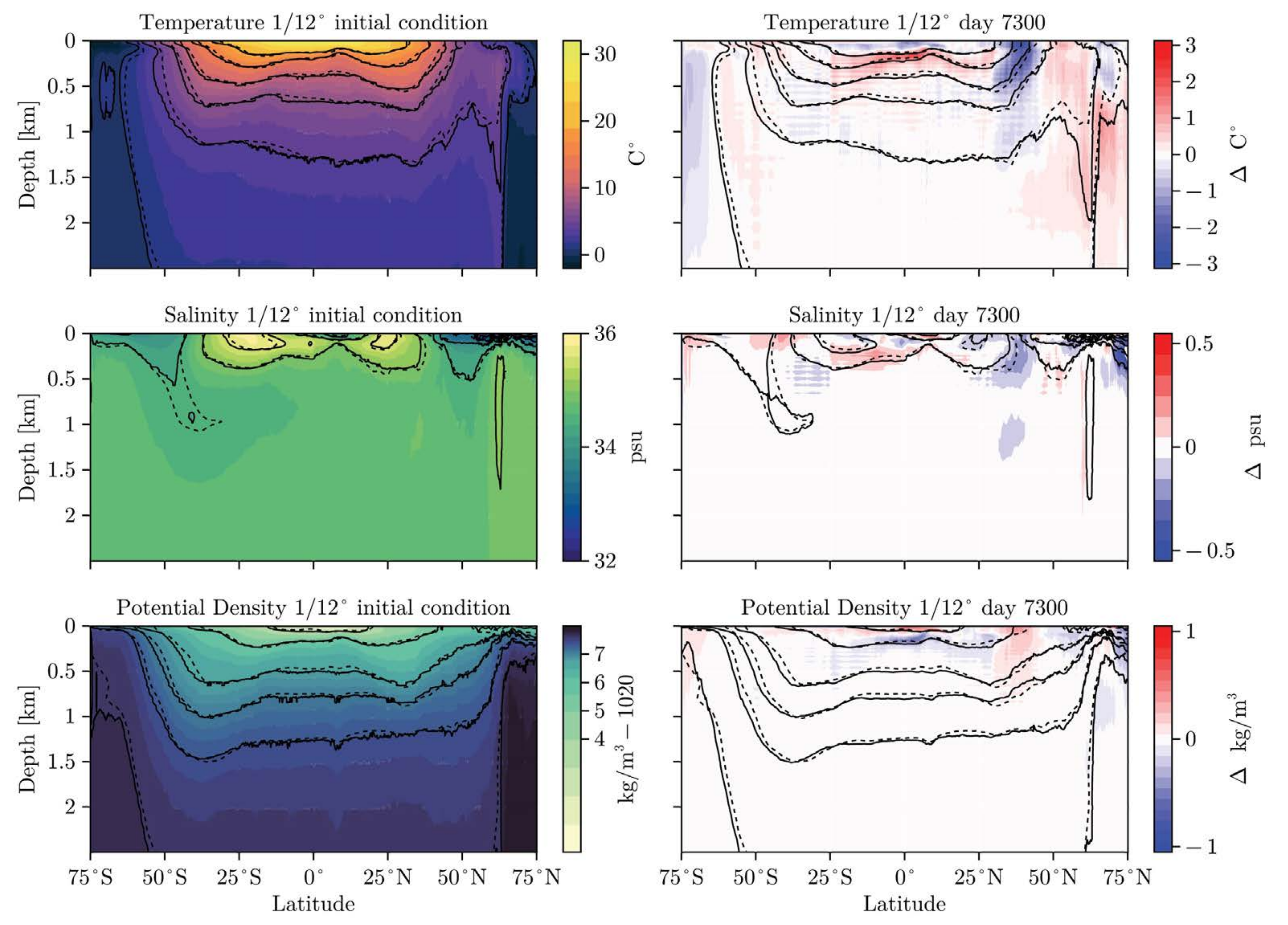


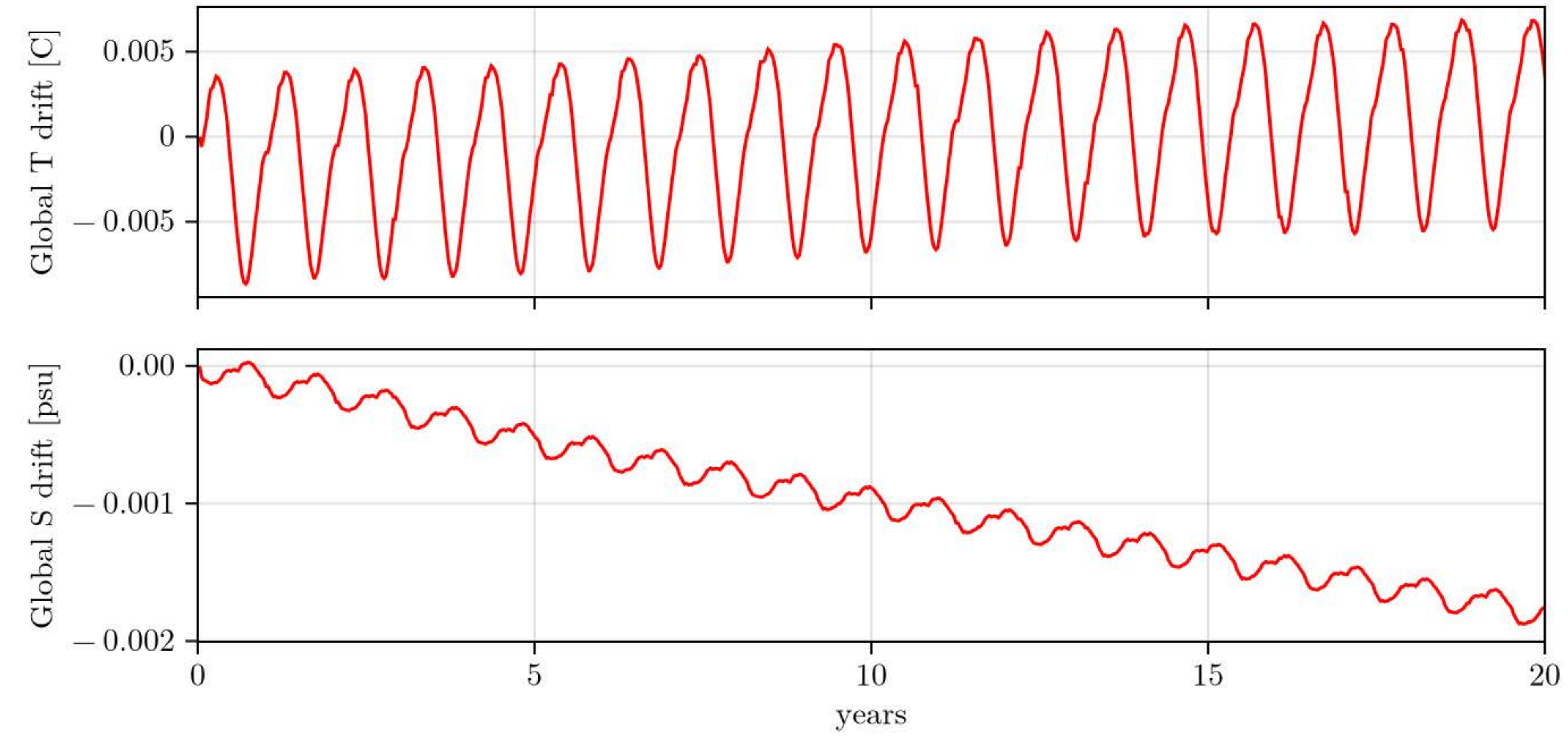
Summary

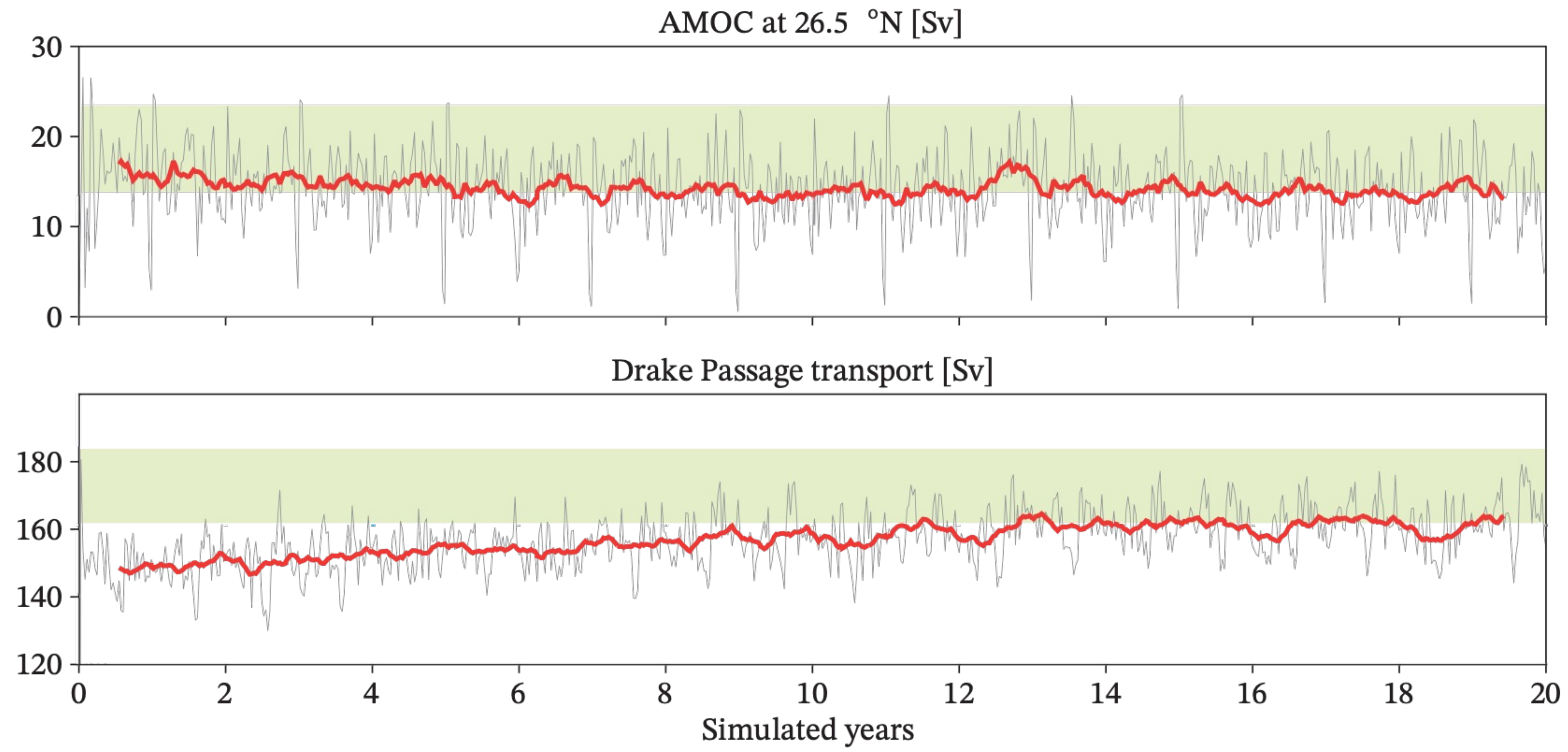
- CliMA is writing a new ocean model called ClimaOcean
- We are leveraging modern programming languages and architectures
- Targeting high-resolution eddying configuration

Thank you!

Evolution over 20 years

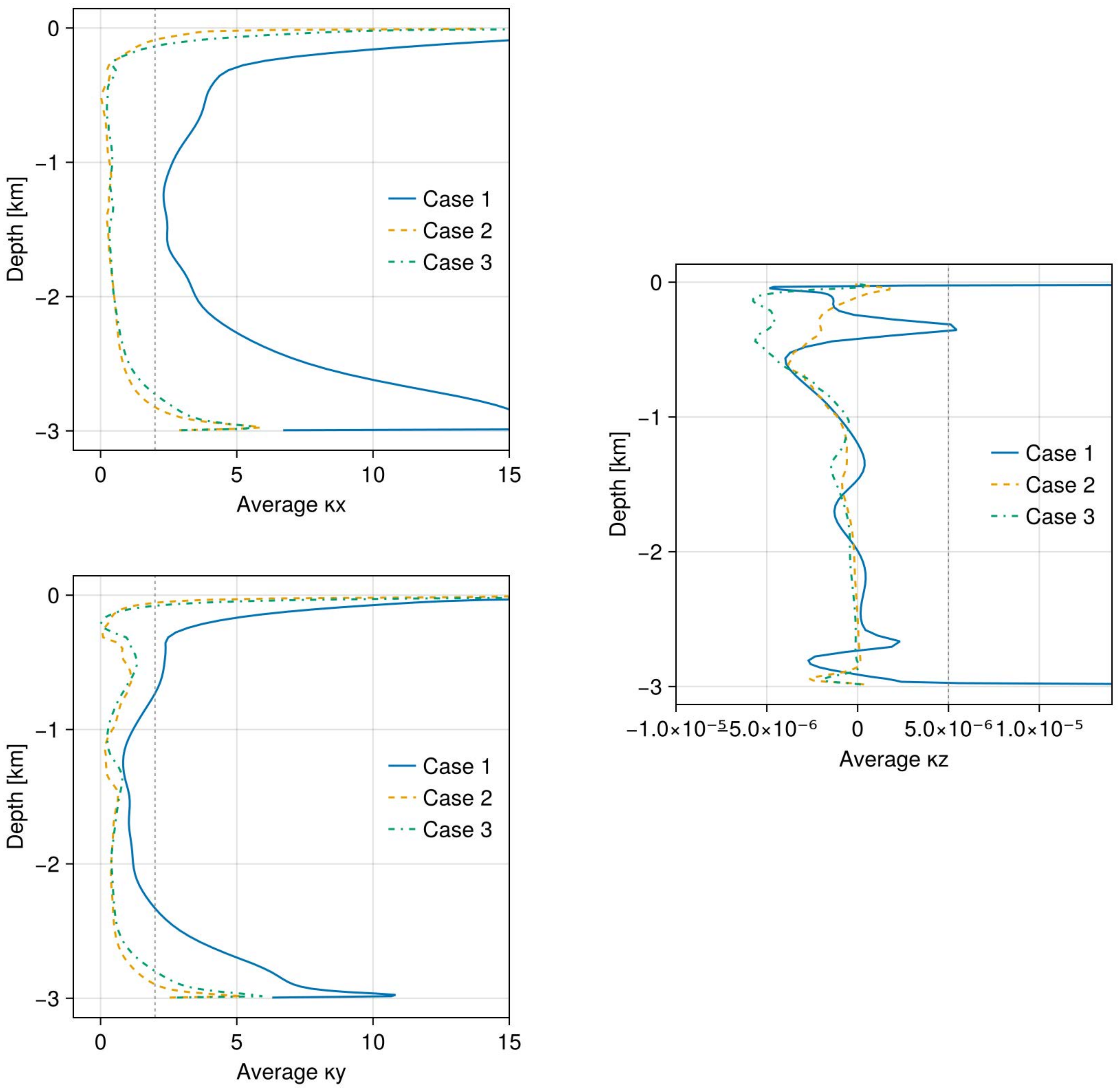






- Drift in density structure comparable to other z-coordinate models
- Reasonable transport

Diapycnal mixing in re-entrant channel



$$\kappa_d = -\frac{\langle P_d \rangle}{2\langle \partial_d T^2 \rangle} \qquad d = x, y, z$$

$$\kappa_i = -\frac{\langle P_x \rangle + \langle P_y \rangle + \langle P_z \rangle}{2\langle \partial_z T^2 \rangle}$$

<u>Conclusion</u>

- We need high-order schemes in the horizontal
- Vertical advection plays little role in diapycnal mixing
- Improving the momentum scheme affects spurious mixing very little

<u>Points of concern</u>

- Top boundary?
- Bottom region