

# ECMWF – DESTINATION EARTH

## SPECTRAL TRANSFORM

Andreas Müller, ECMWF

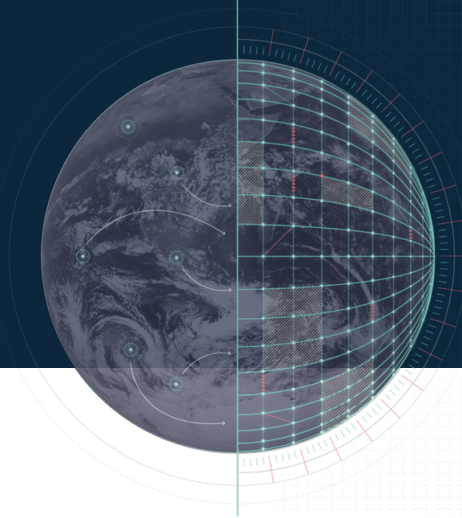


Funded by  
the European Union

**Destination Earth**

implemented by





10 minutes

- Fourier transform
- Spectral transform

60 minutes

- hands-on exercises with Python

30 minutes

- break in between

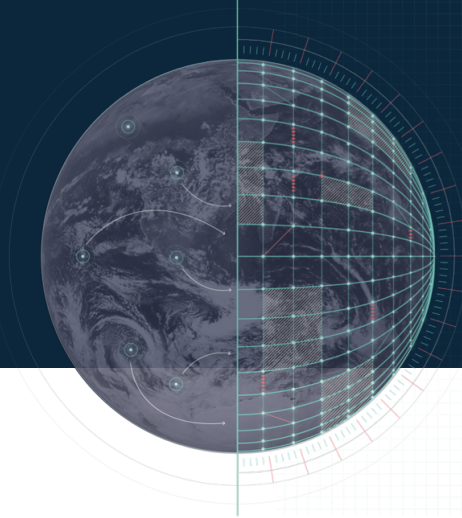
30 minutes

- aliasing
- parallelization
- performance
- Fast Legendre Transform

rest

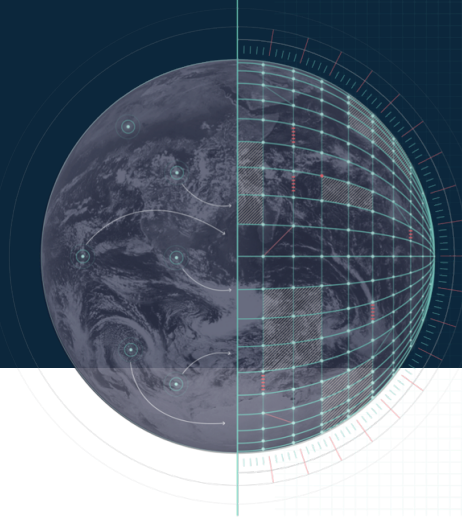
- time for questions

# IFS (Integrated Forecast System)



technology applied at ECMWF for  
the last 30 years

- spectral transform (since April 1983)
- semi-implicit
- semi-Lagrangian

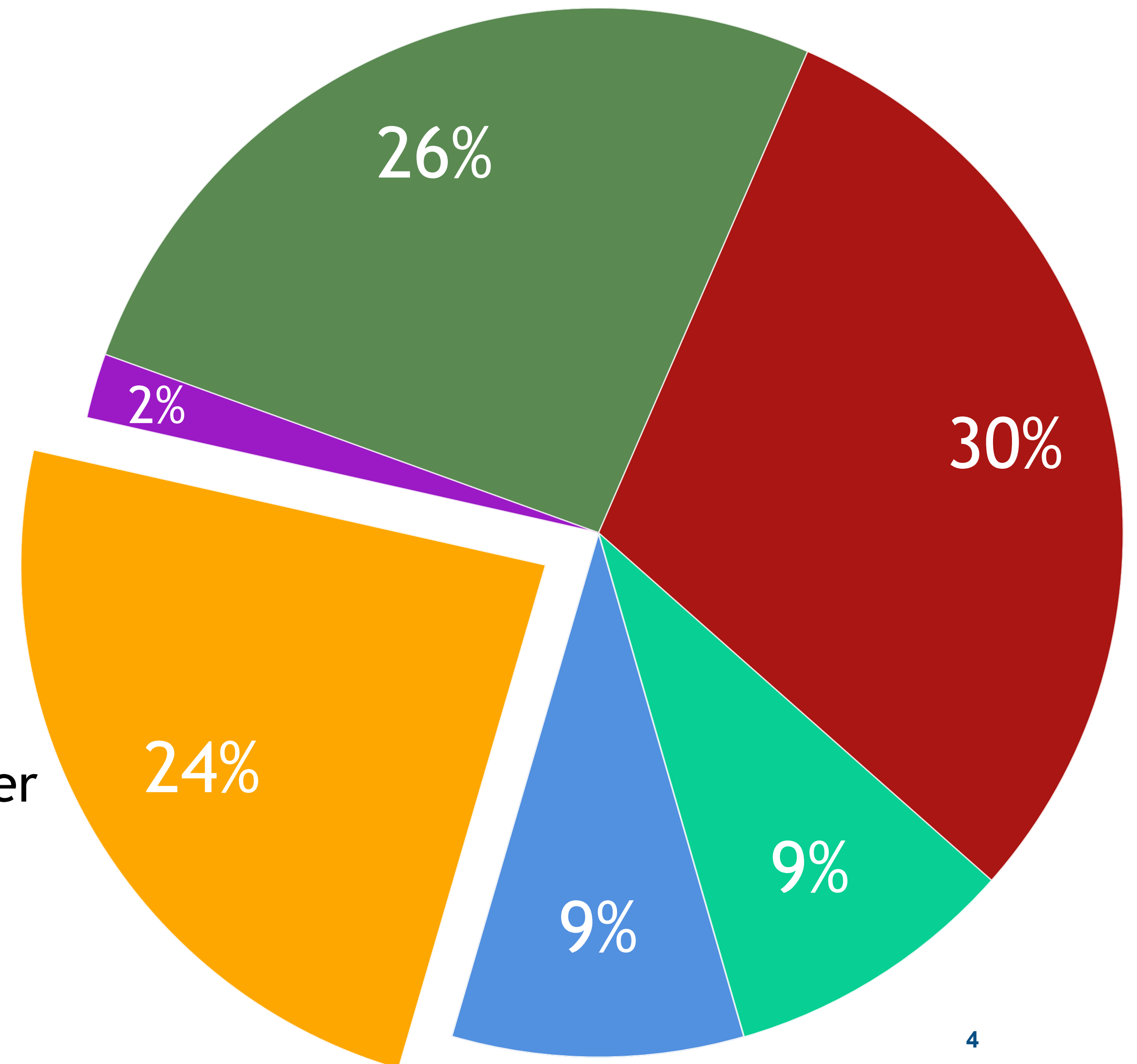


# IFS (Integrated Forecast System)

technology applied at ECMWF for the last 30 years

- spectral transform (since April 1983)
- semi-implicit
- semi-Lagrangian

pie chart: % of runtime in 9km operational forecast



- spectral transform
- grid point dynamics
- wave model
- semi-implicit solver
- physics+radiation
- ocean model

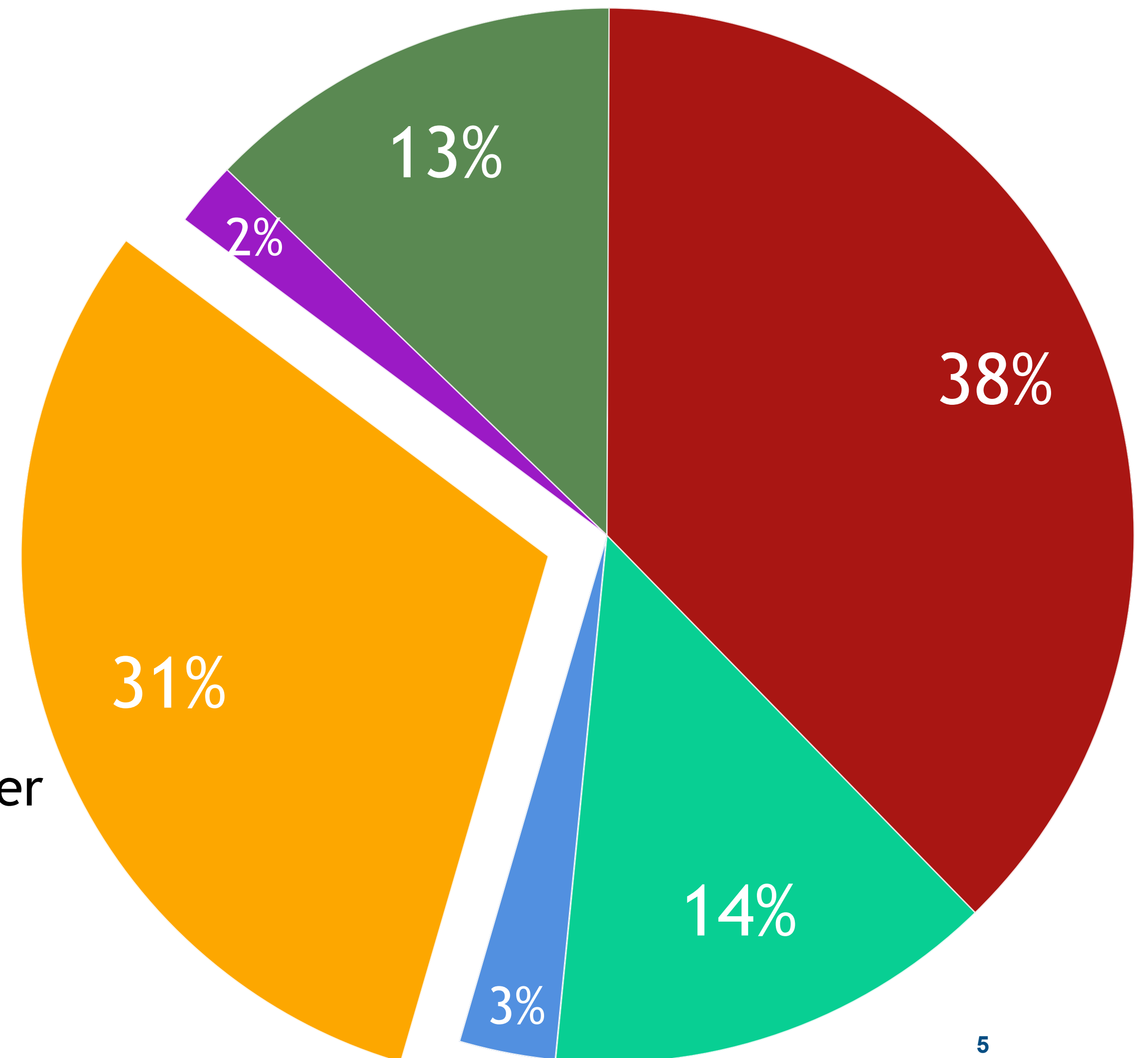


# IFS (Integrated Forecast System)

technology applied at ECMWF for the last 30 years

- spectral transform (since April 1983)
- semi-implicit
- semi-Lagrangian

pie chart: % of runtime in 5km forecast (future operational)



- spectral transform
- grid point dynamics
- wave model
- semi-implicit solver
- physics+radiation
- ocean model

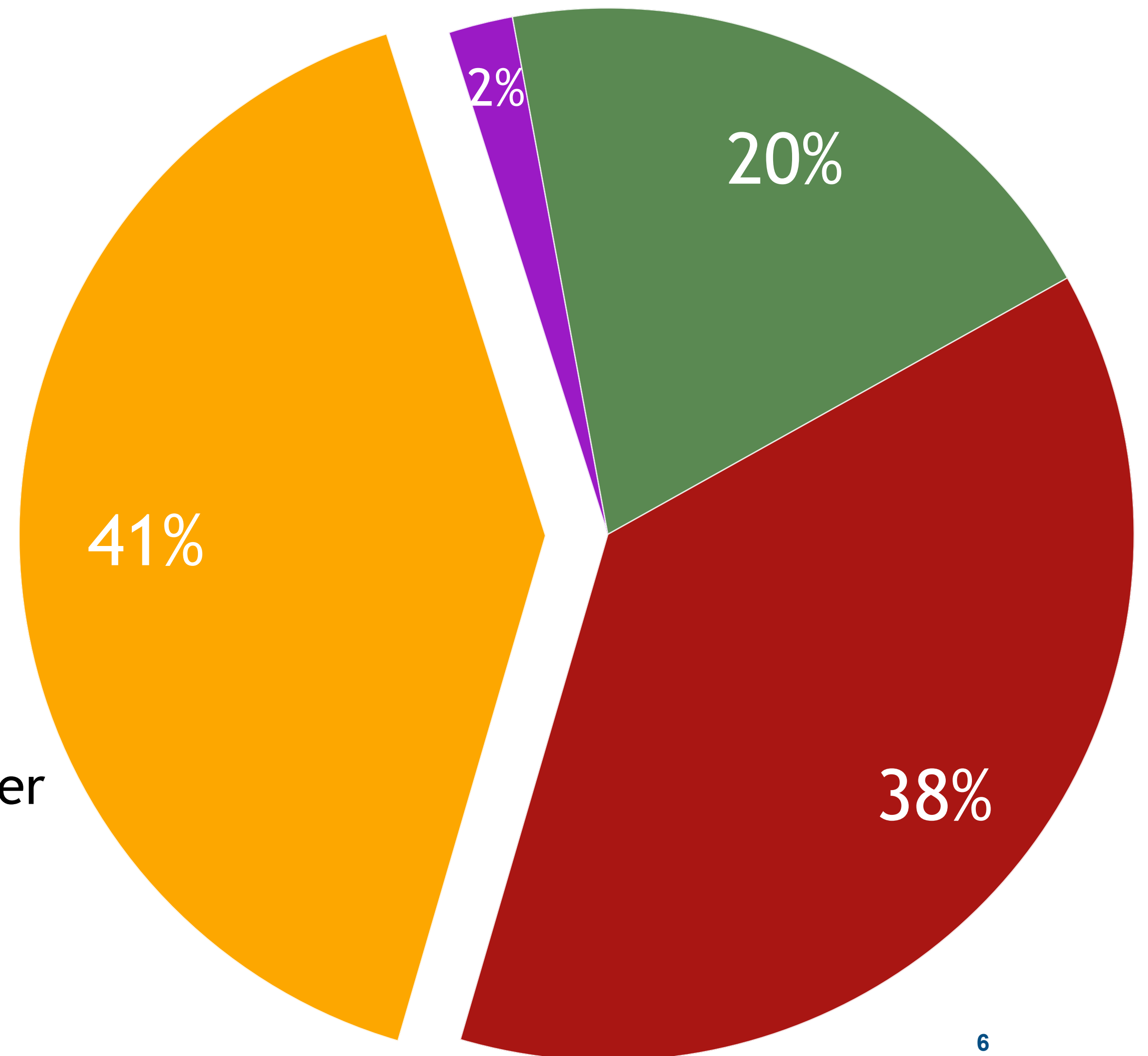


# IFS (Integrated Forecast System)

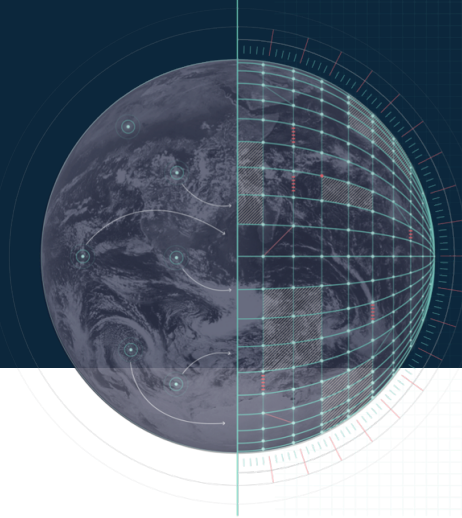
technology applied at ECMWF for the last 30 years

- spectral transform (since April 1983)
- semi-implicit
- semi-Lagrangian

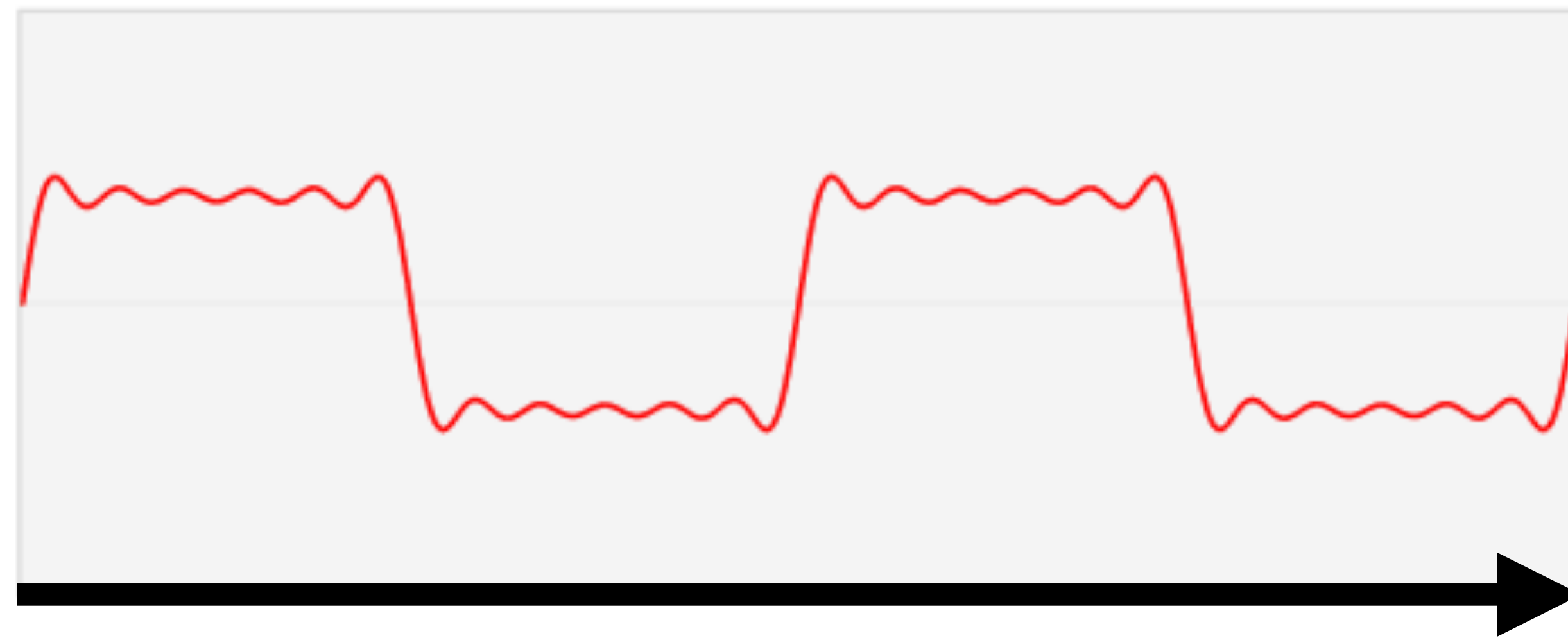
pie chart: % of runtime in 1.25km forecast (experiment, no ocean)



- spectral transform
- grid point dynamics
- wave model
- semi-implicit solver
- physics+radiation
- ocean model



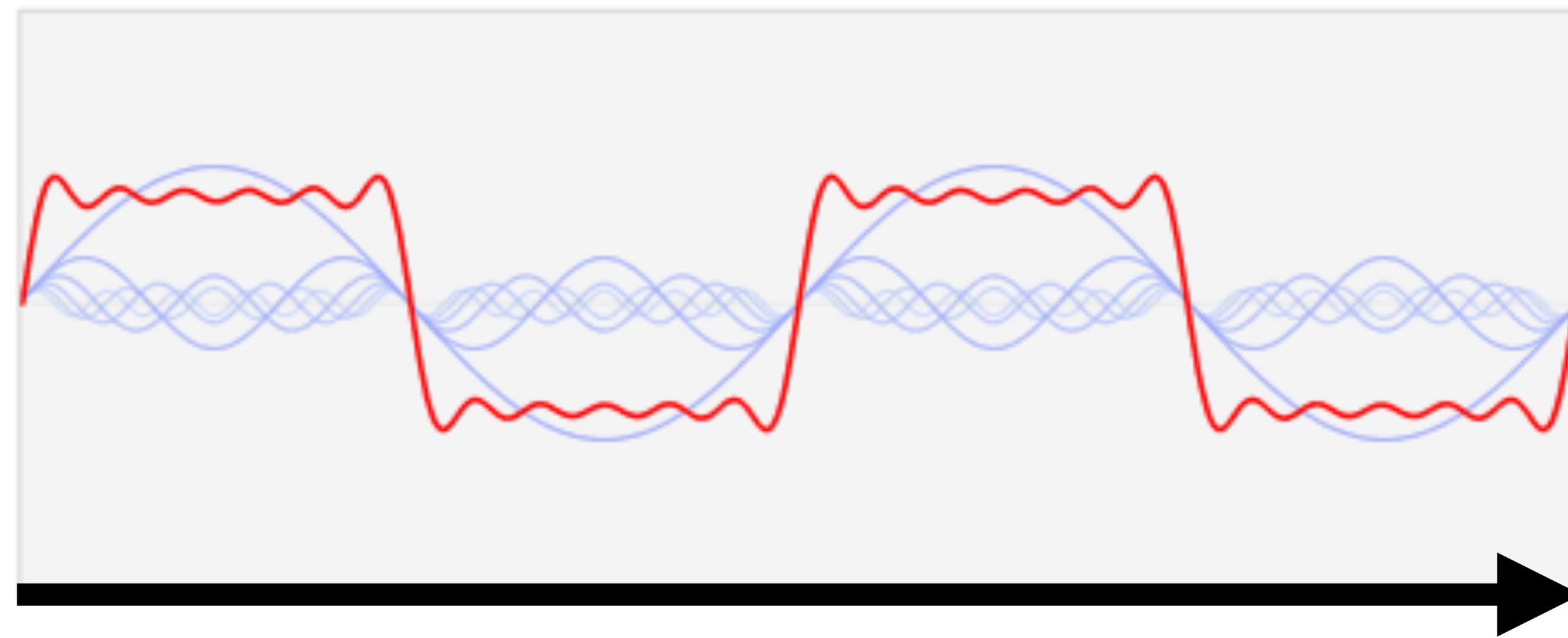
Fourier transform = Spectral transform in 1D



location  $x$



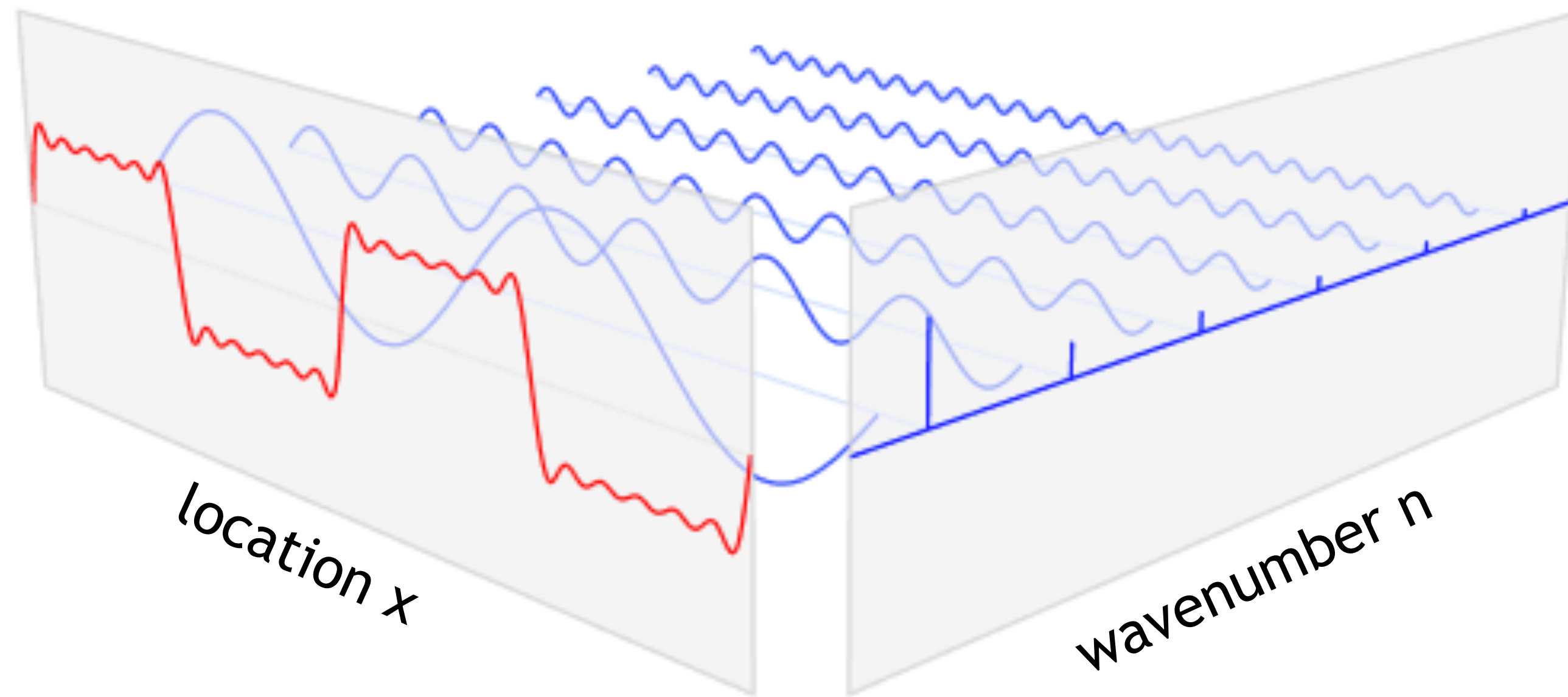
Fourier transform = Spectral transform in 1D



location  $x$



Fourier transform = Spectral transform in 1D

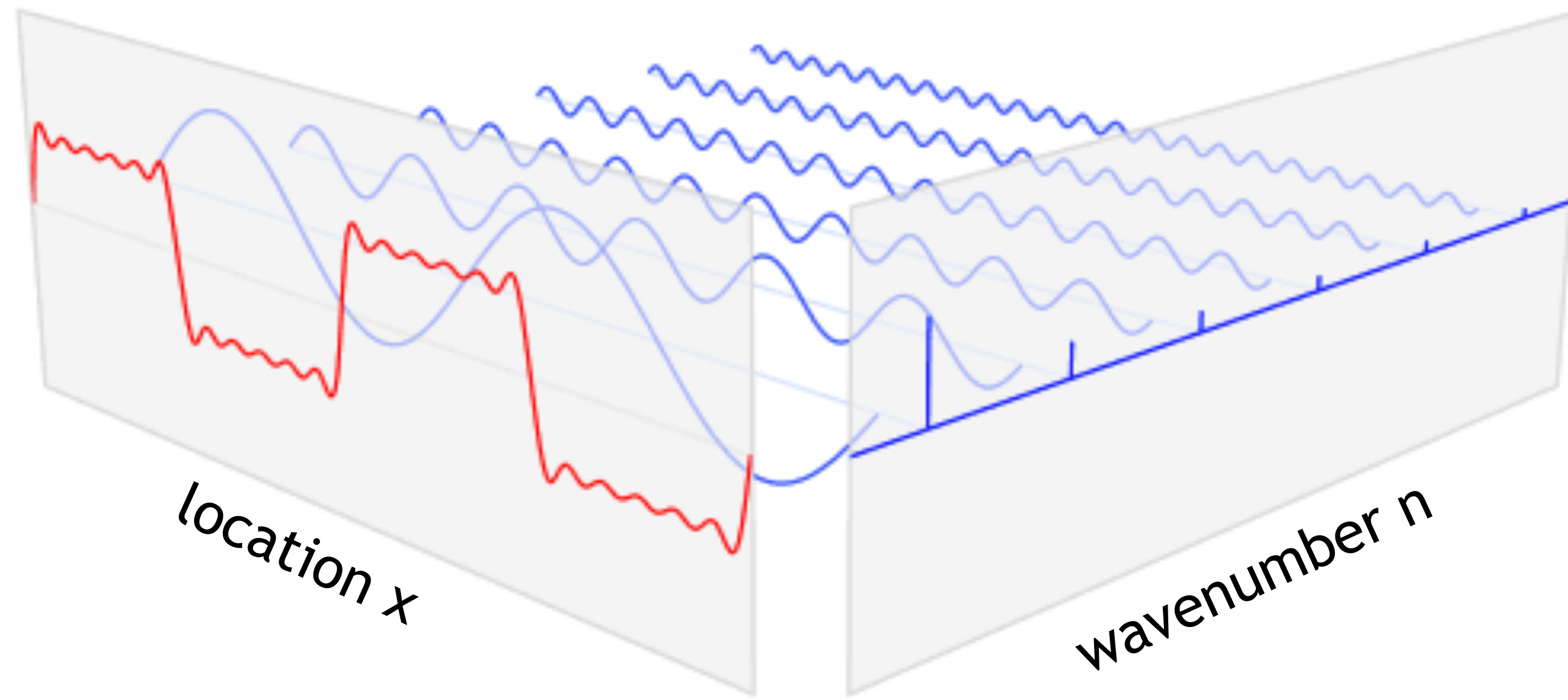


grid point space

Fourier space



# Fourier transform



function in grid  
point space

$$f(x) = \sum_n f_n \cdot e^{-2\pi i n x}$$

Fourier  
coefficients

# Fourier transform



function in grid  
point space

$$f(x) = \sum_n f_n \cdot e^{-2\pi i n x}$$

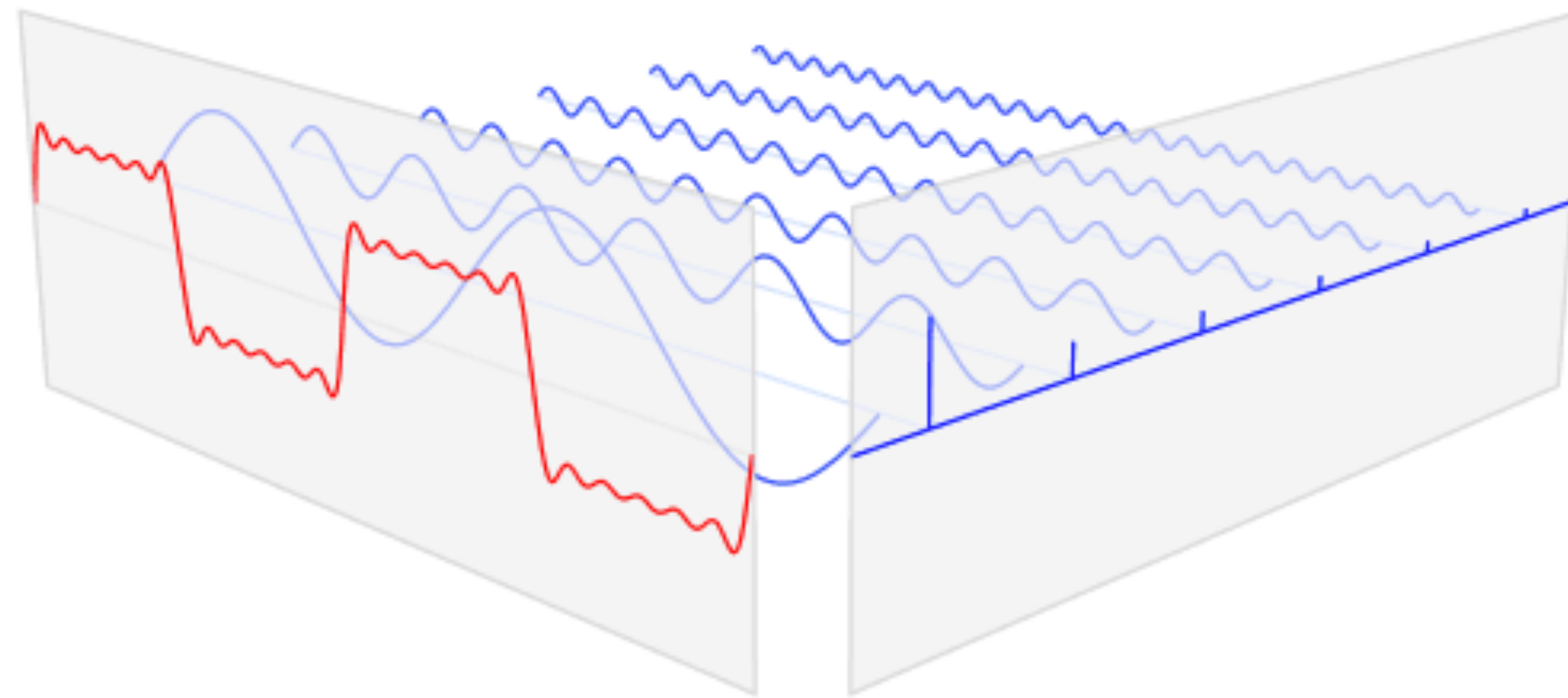
Fourier  
coefficients

differentiation

$$\frac{df(x)}{dx} = \sum_n (-2\pi i n f_n) \cdot e^{-2\pi i n x}$$

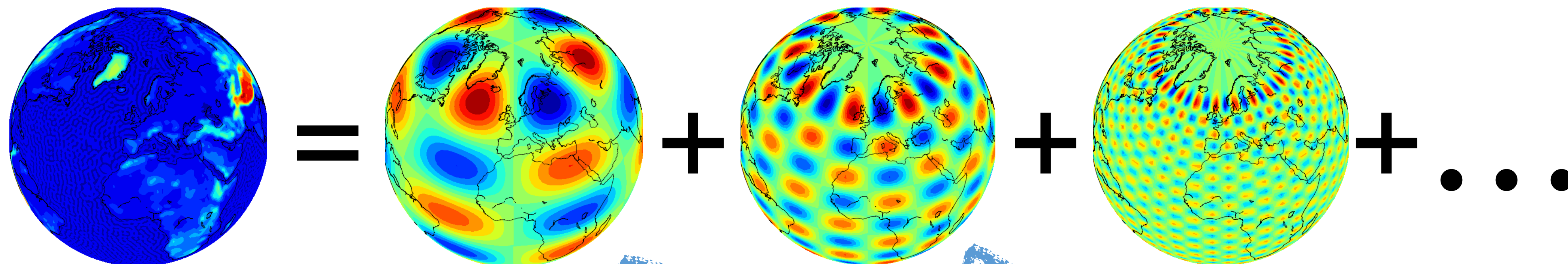
simple  
multiplication

# on the sphere: spectral transform



grid point space

spectral space



spherical harmonics



# on the sphere: spectral transform

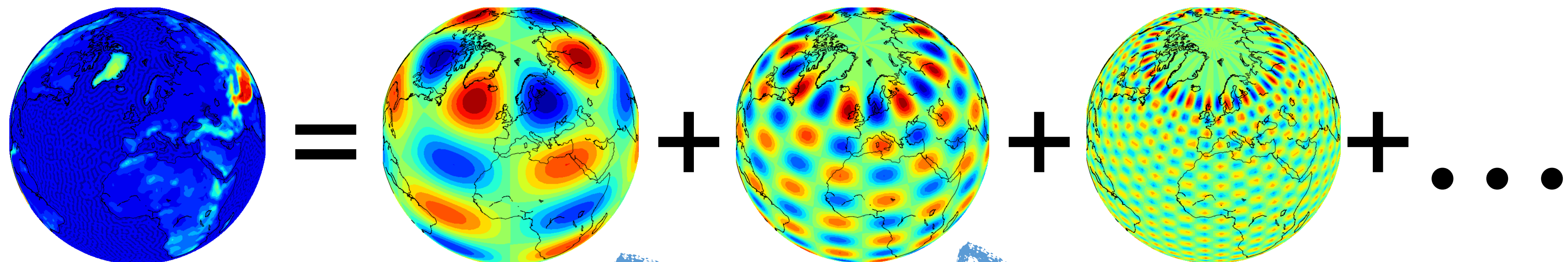
Grid point variable      Latitude      Longitude      Spectral coefficients      Spherical harmonics

$$f(\phi, \lambda) = \Re \left( \sum_{m=0}^M \sum_{n=m}^M f_{m,n} Y_n^m(\phi, \lambda) \right)$$

m: zonal wavenumber  
n: total wavenumber  
M: truncation

grid point space

spectral space



spherical harmonics



# on the sphere: spectral transform

Spectral coefficients

Grid point variable      Latitude      Longitude      Spherical harmonics

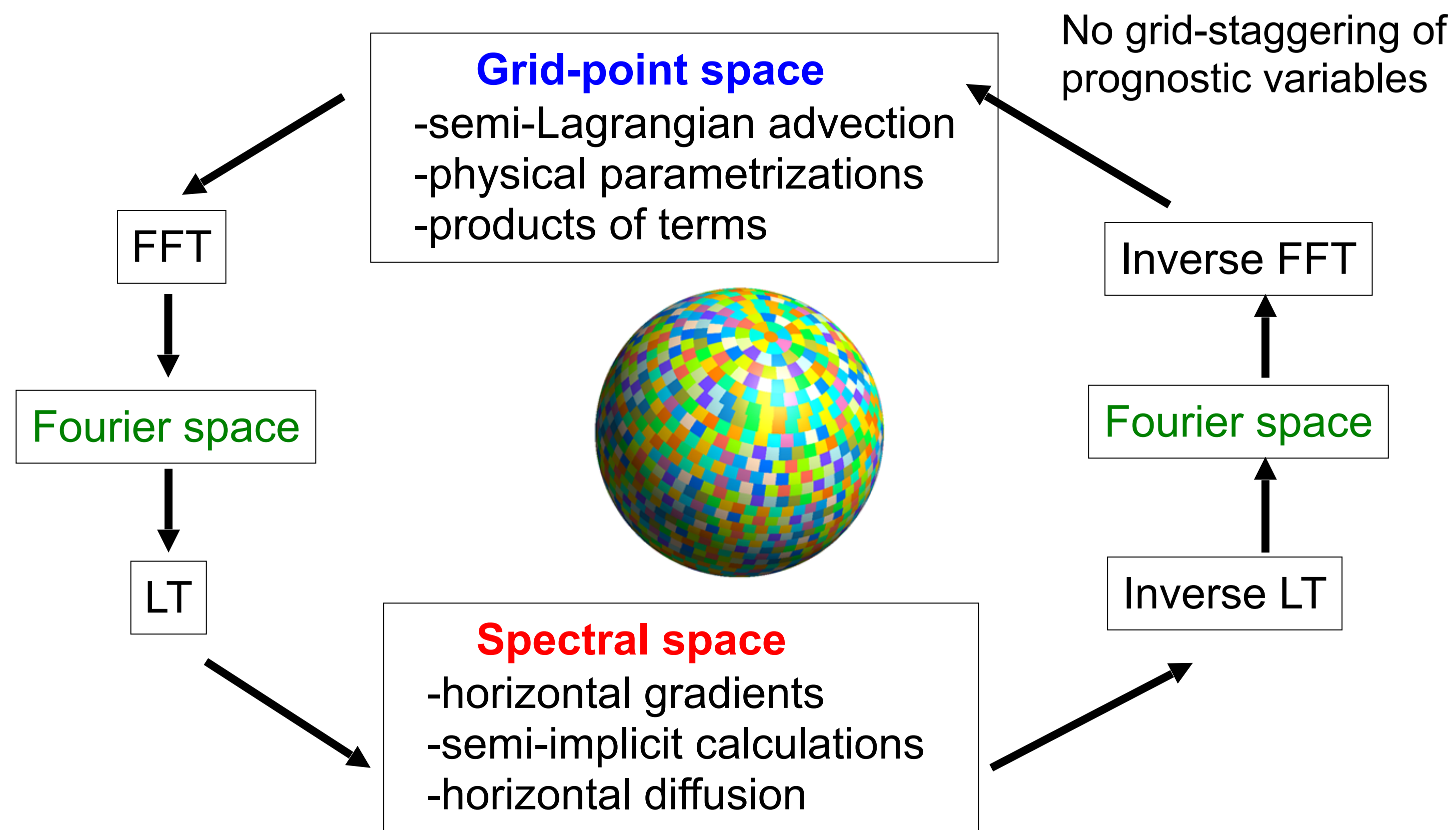
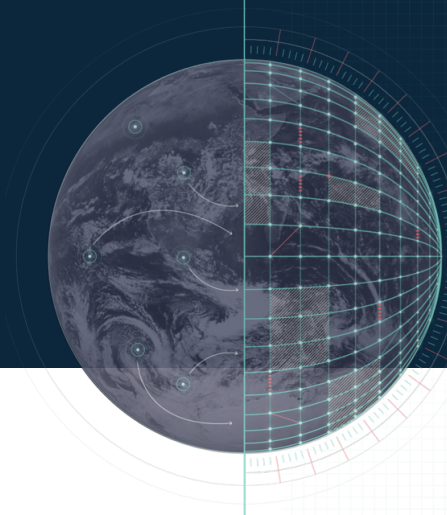
$$f(\phi, \lambda) = \Re \left( \sum_{m=0}^M \sum_{n=m}^M f_{m,n} Y_n^m(\phi, \lambda) \right)$$

m: zonal wavenumber  
n: total wavenumber  
M: truncation

Legendre polynomials

$$f(\phi, \lambda) = \Re \left( \underbrace{\sum_{m=0}^M e^{im\lambda}}_{\text{Fourier transform}} \underbrace{\sum_{n=m}^M f_{m,n} P_n^m(\phi)}_{\text{Legendre transform}} \right)$$

# time step in IFS



FFT: Fast Fourier Transform, LT: Legendre Transform



**for everyone: interactive web-app about spectral transform**

open in a browser: [anmrde.github.io/spectral](https://anmrde.github.io/spectral)

**optional: Python course**

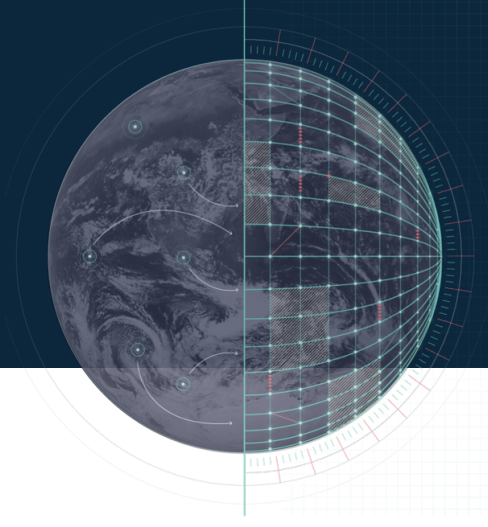
open in Jupyterlab in your browser: `/NMcourse/spectral/solution.ipynb`

Exercises are getting more difficult. Feel free to skip exercises as you want. The full Python course is designed to fill 20 hours.

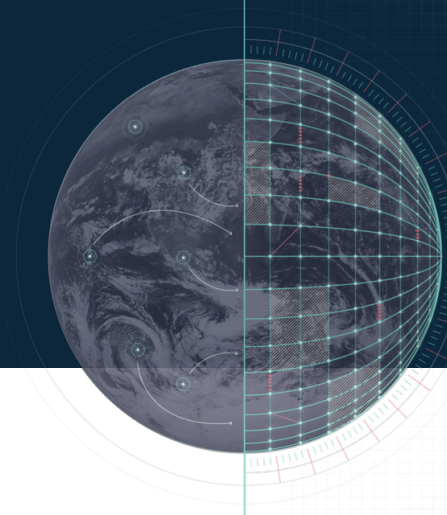
**files:**     `exercises.ipynb`: Python notebook with exercises  
              `solution.ipynb`: notebook including sample solutions

**ECMWF Jupyterhub (16GB of RAM) or personal Linux computer:**

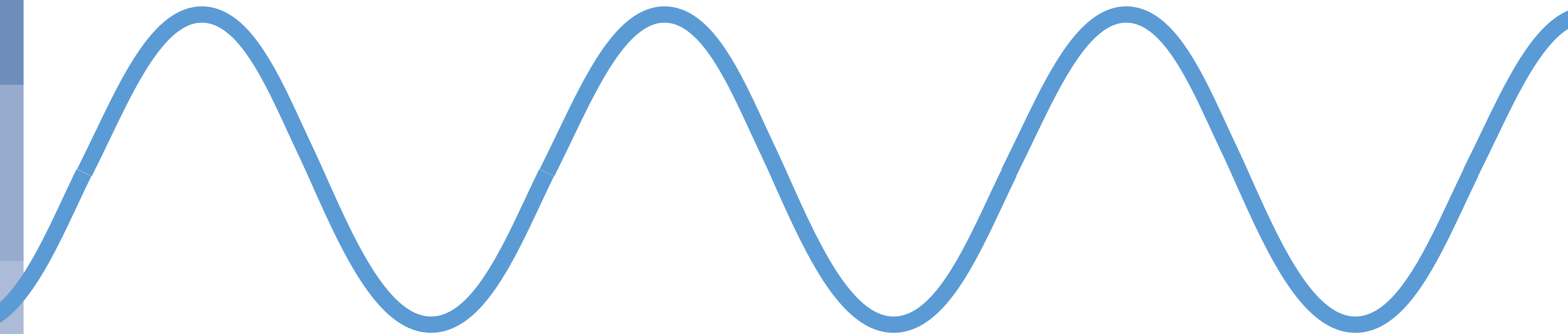
<https://github.com/anmrde/spectral/tree/master/jupyter>



**Issue:** multiplication of two variables produces shorter waves than grid can handle



wave generated in spectral space

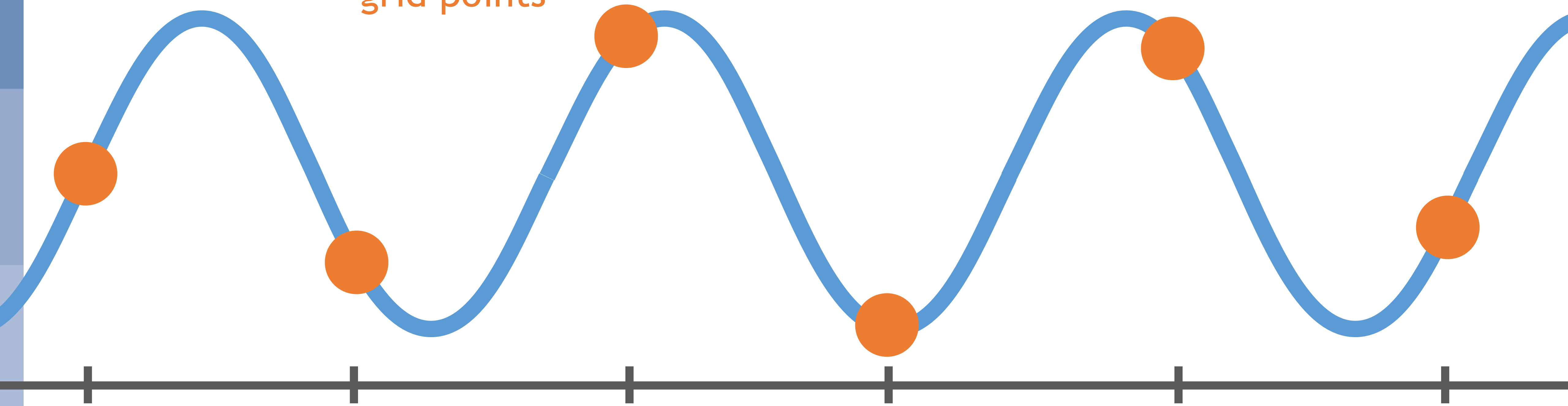


**Issue:** multiplication of two variables produces shorter waves than grid can handle



wave generated in spectral space

grid points

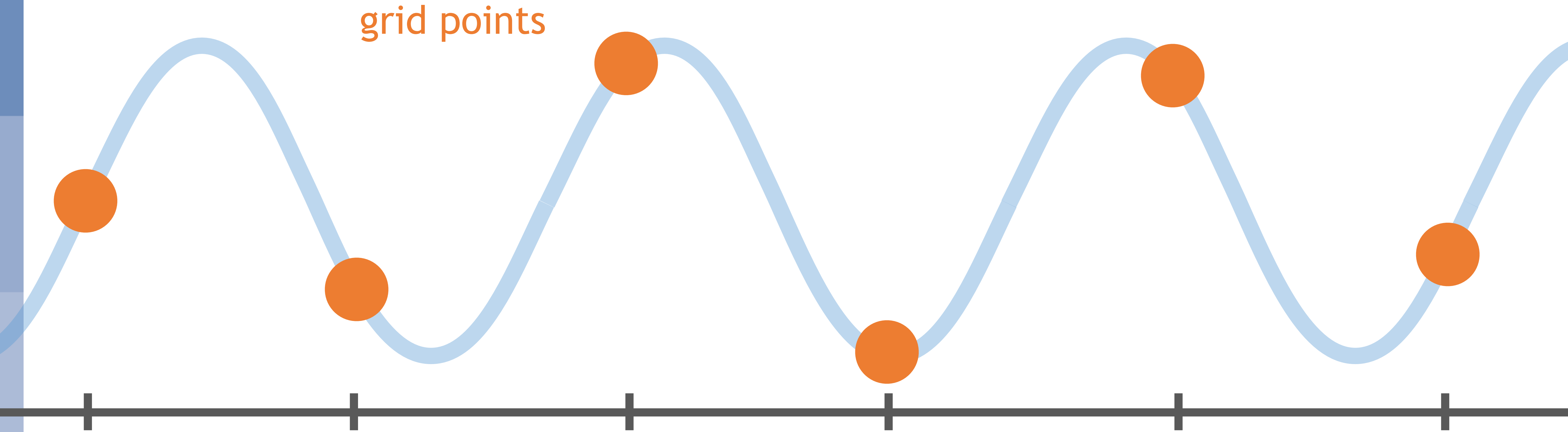


**Issue:** multiplication of two variables produces shorter waves than grid can handle



wave generated in spectral space

grid points

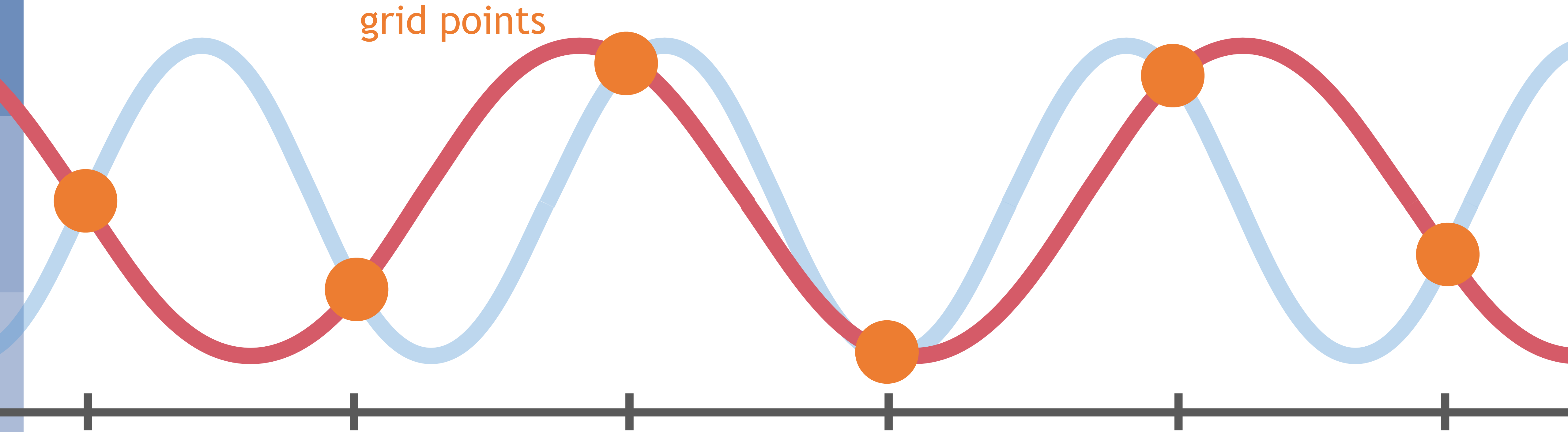


**Issue:** multiplication of two variables produces shorter waves than grid can handle



wave generated in spectral space

grid points

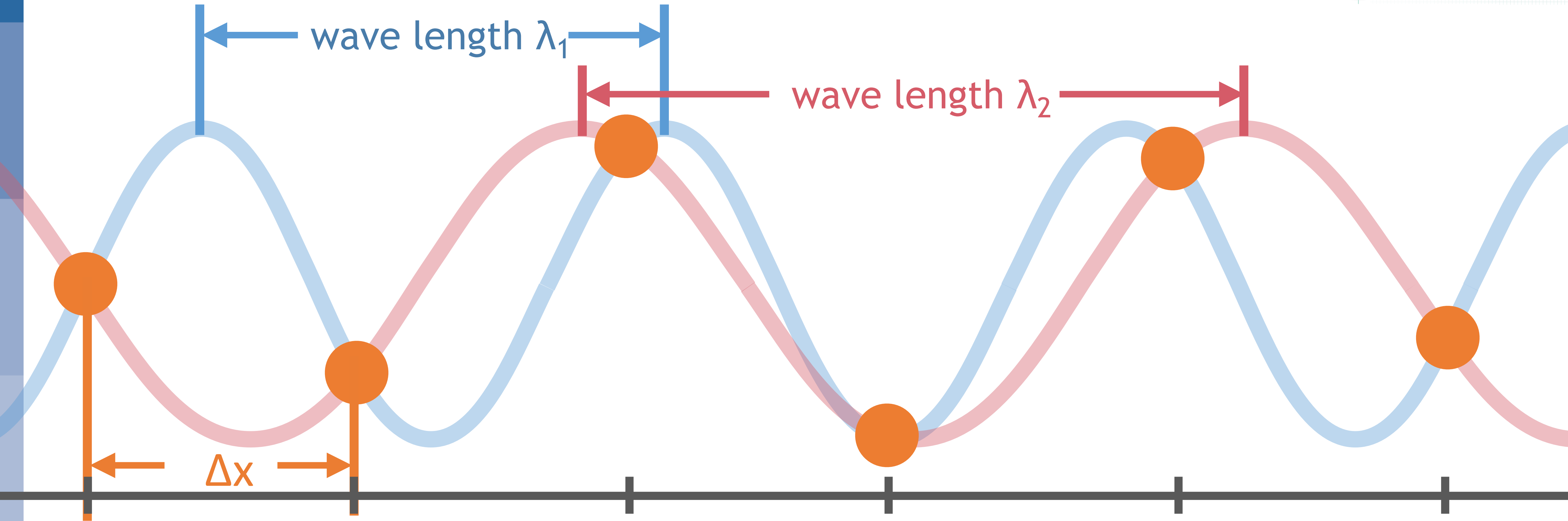


**Issue:** multiplication of two variables produces shorter waves than grid can handle

wave in grid point space



# aliasing

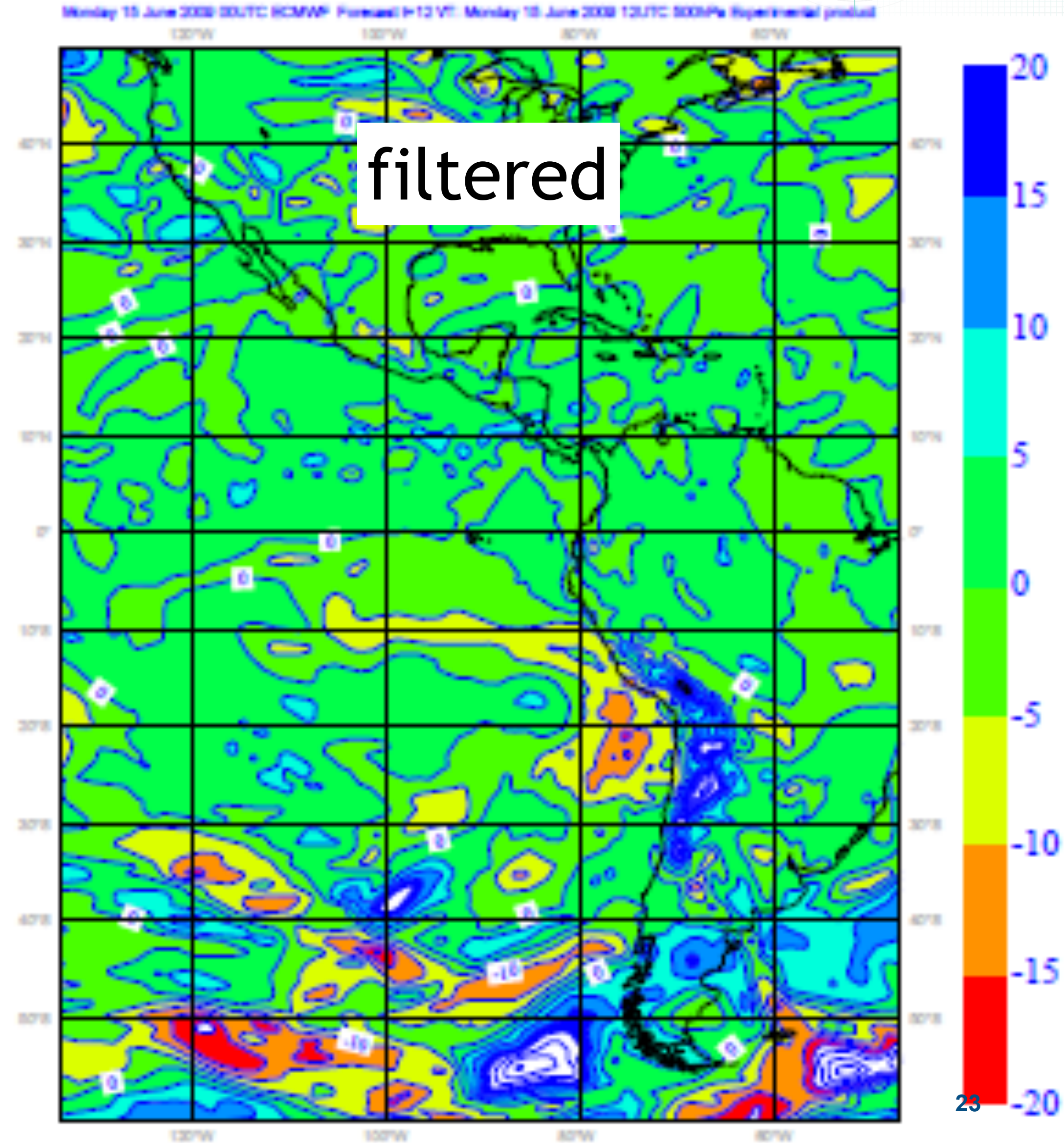
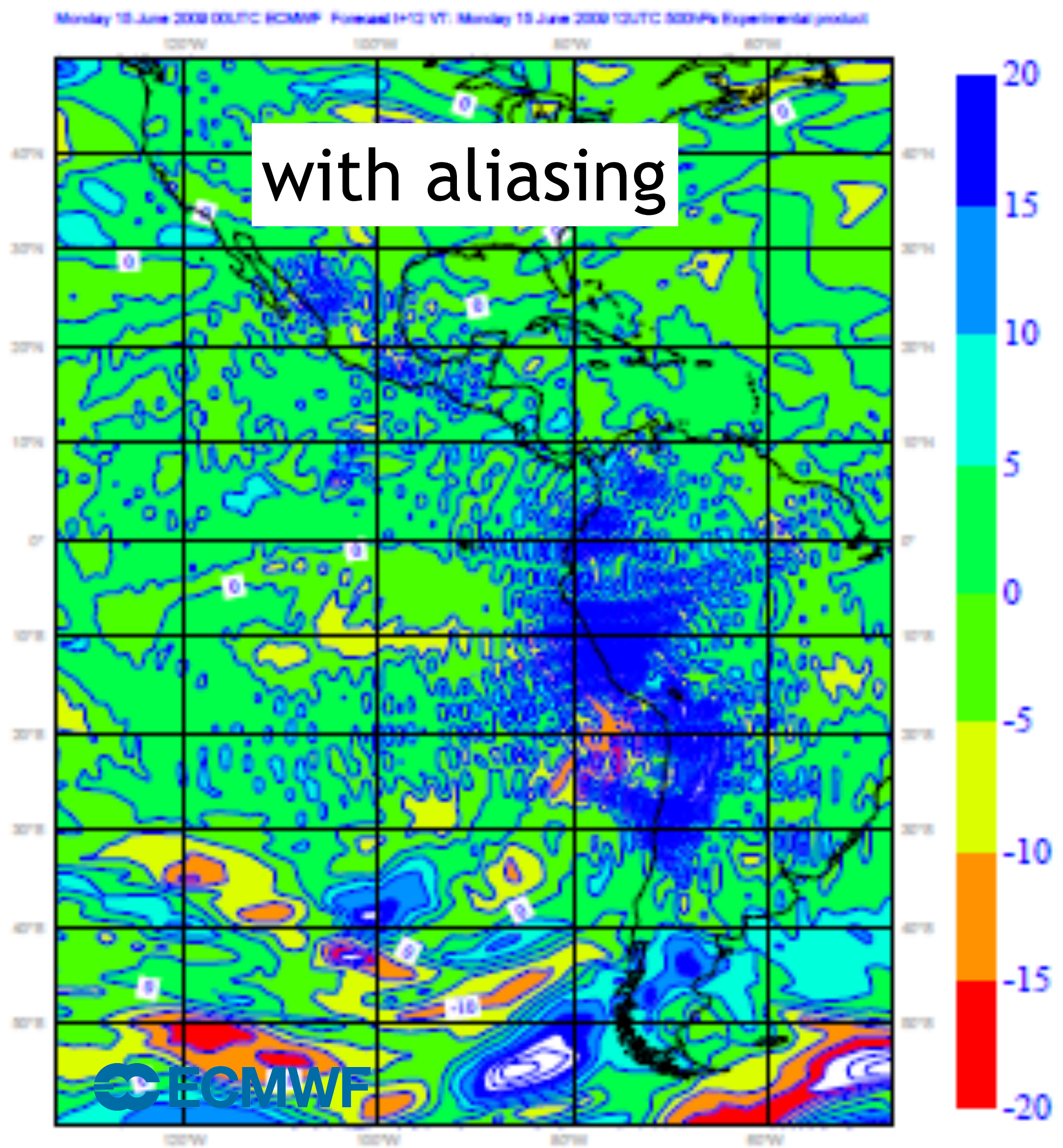
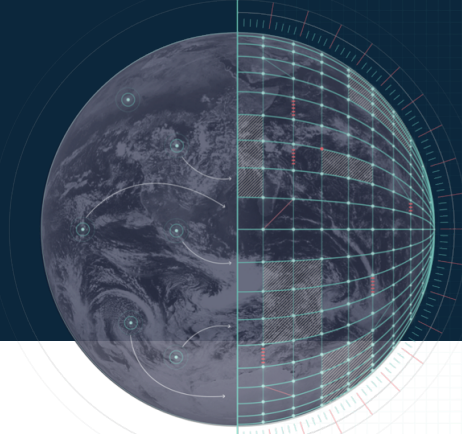


**Issue:** multiplication of two variables produces shorter waves than grid can handle

grid can handle  $\lambda \geq 2 \Delta x$

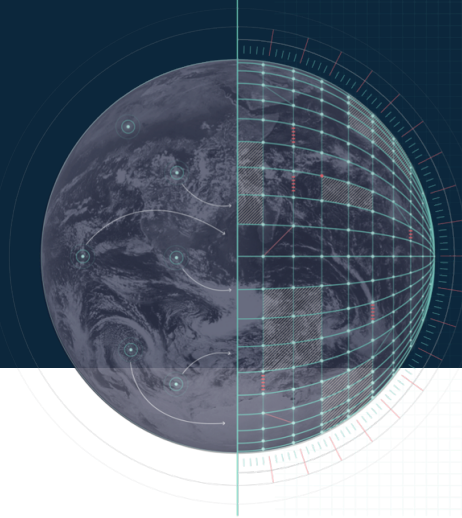
# aliasing example

## 500hPa adiabatic zonal wind tendencies (T159)



# aliasing example

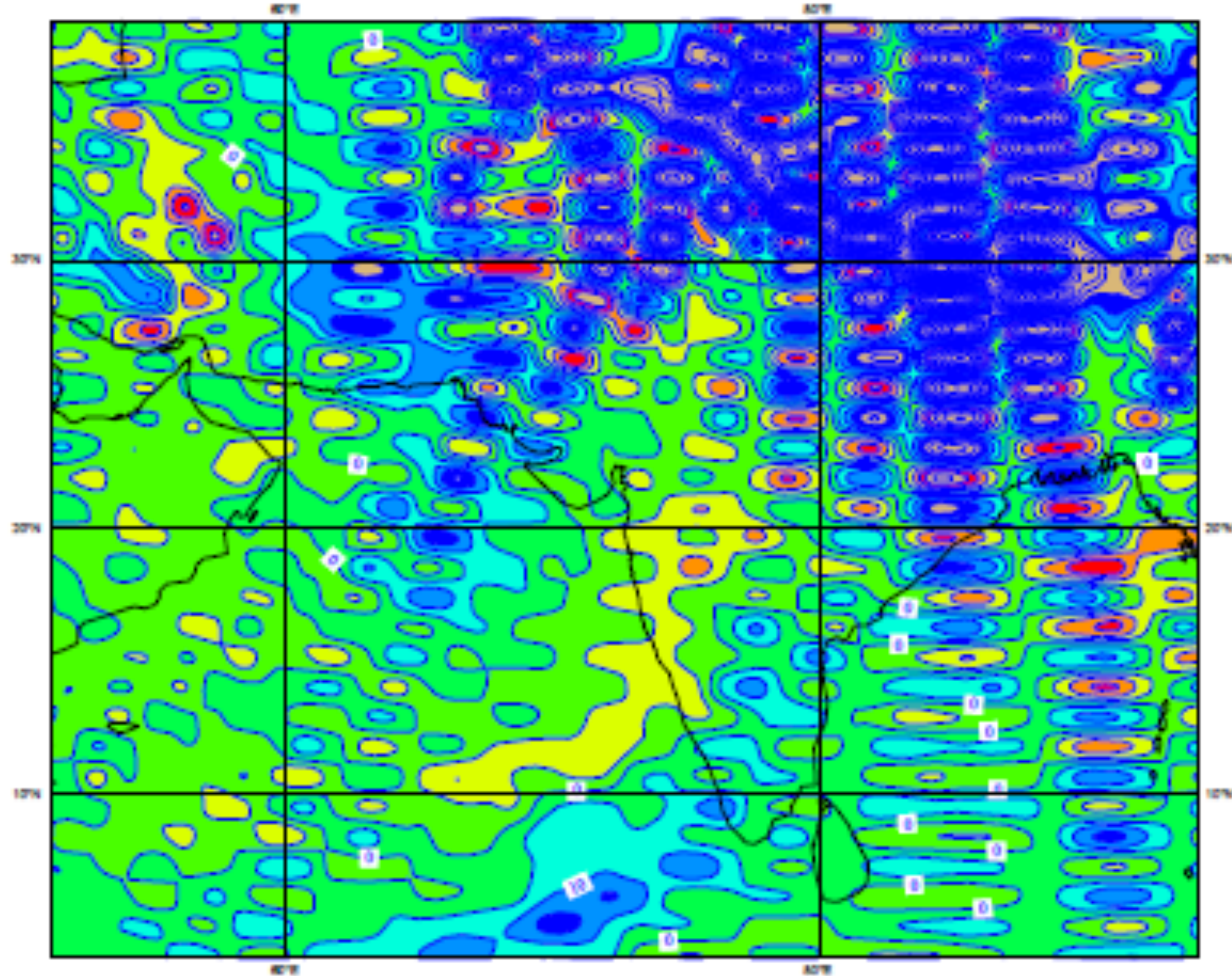
## 500hPa adiabatic meridional wind tendencies (T159)



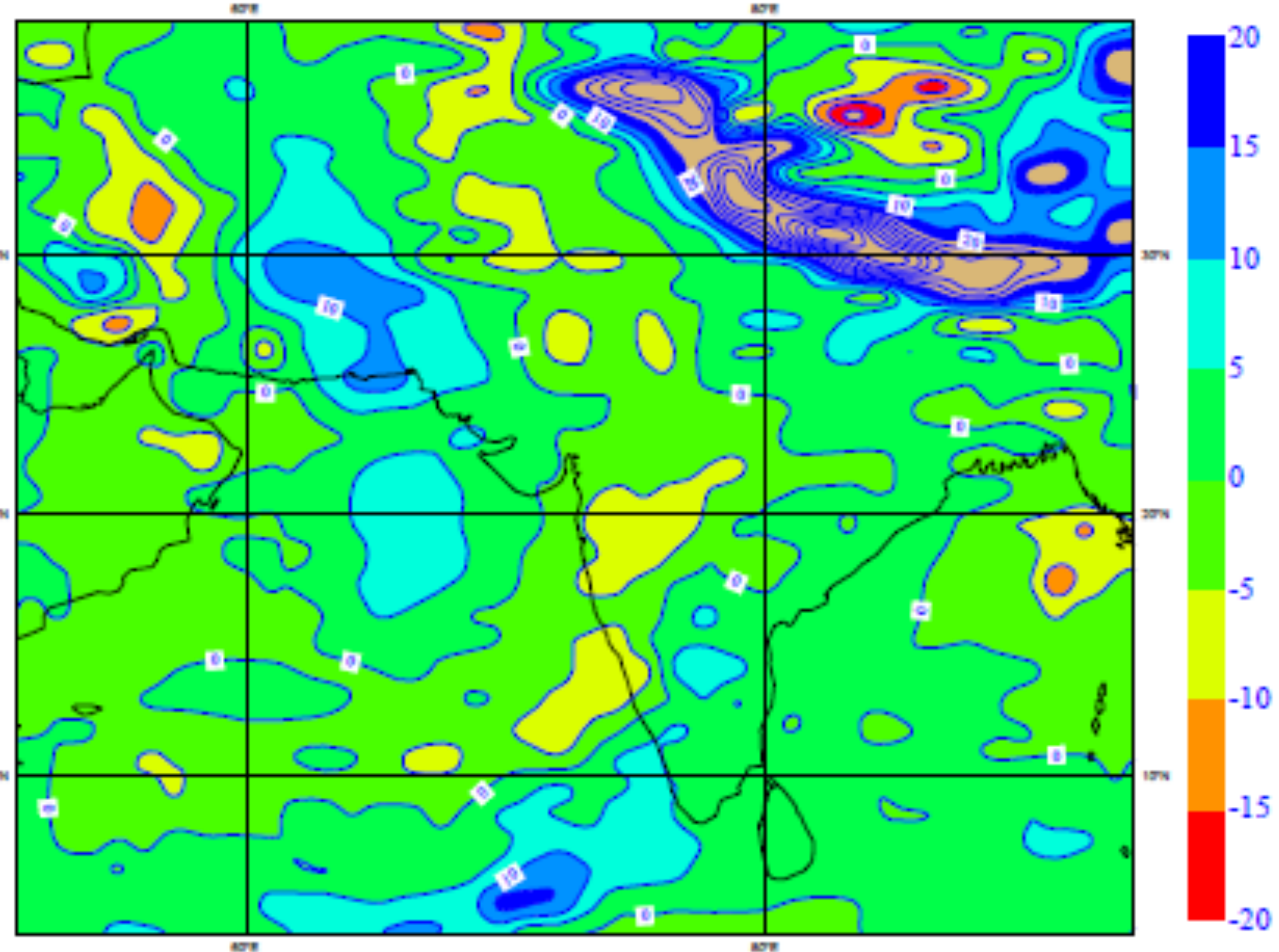
with aliasing

filtered

Monday 15 June 2009 00UTC ECMWF Forecast t+24 VT: Tuesday 16 June 2009 00UTC 500hPa Experimental product

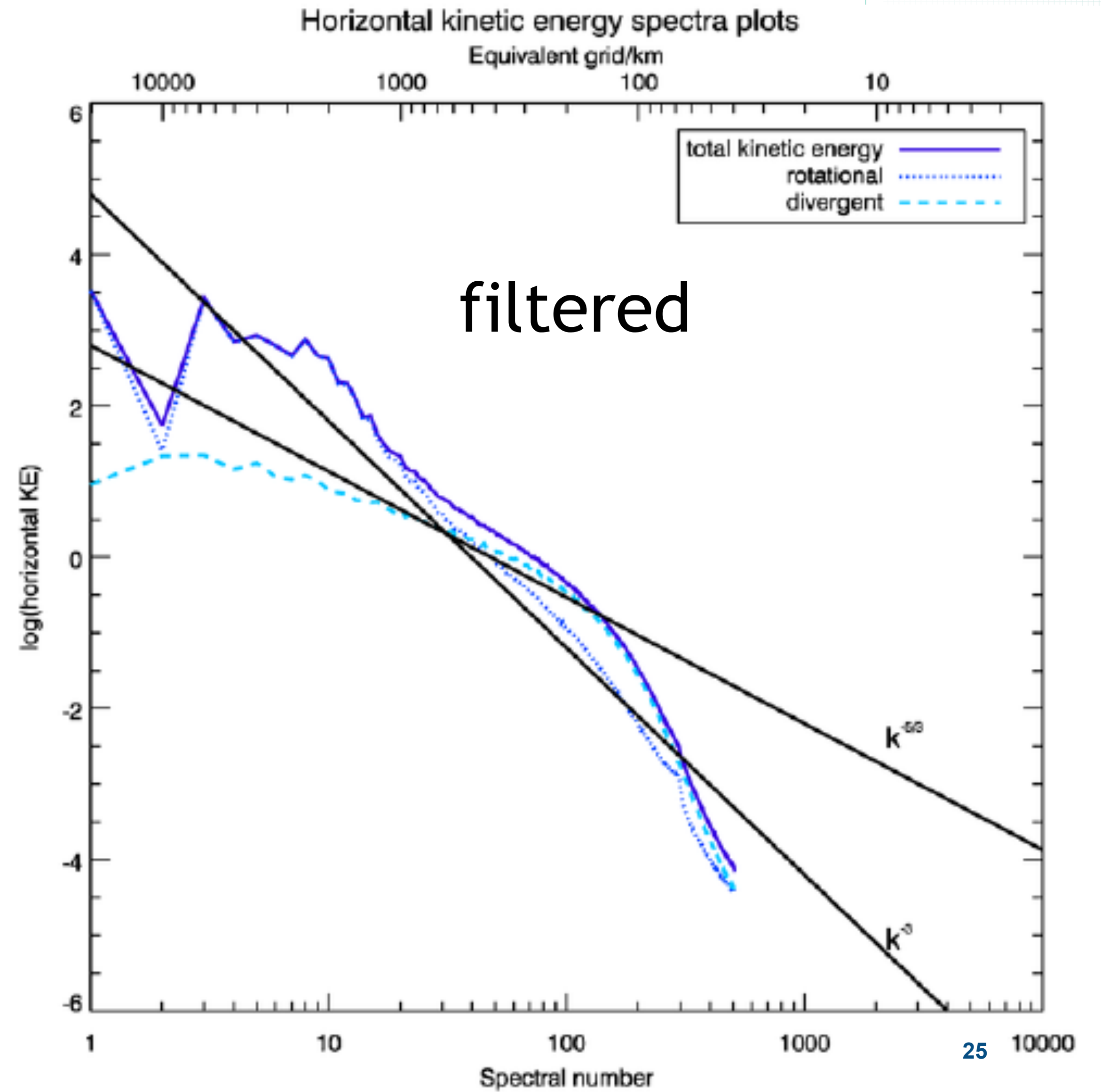
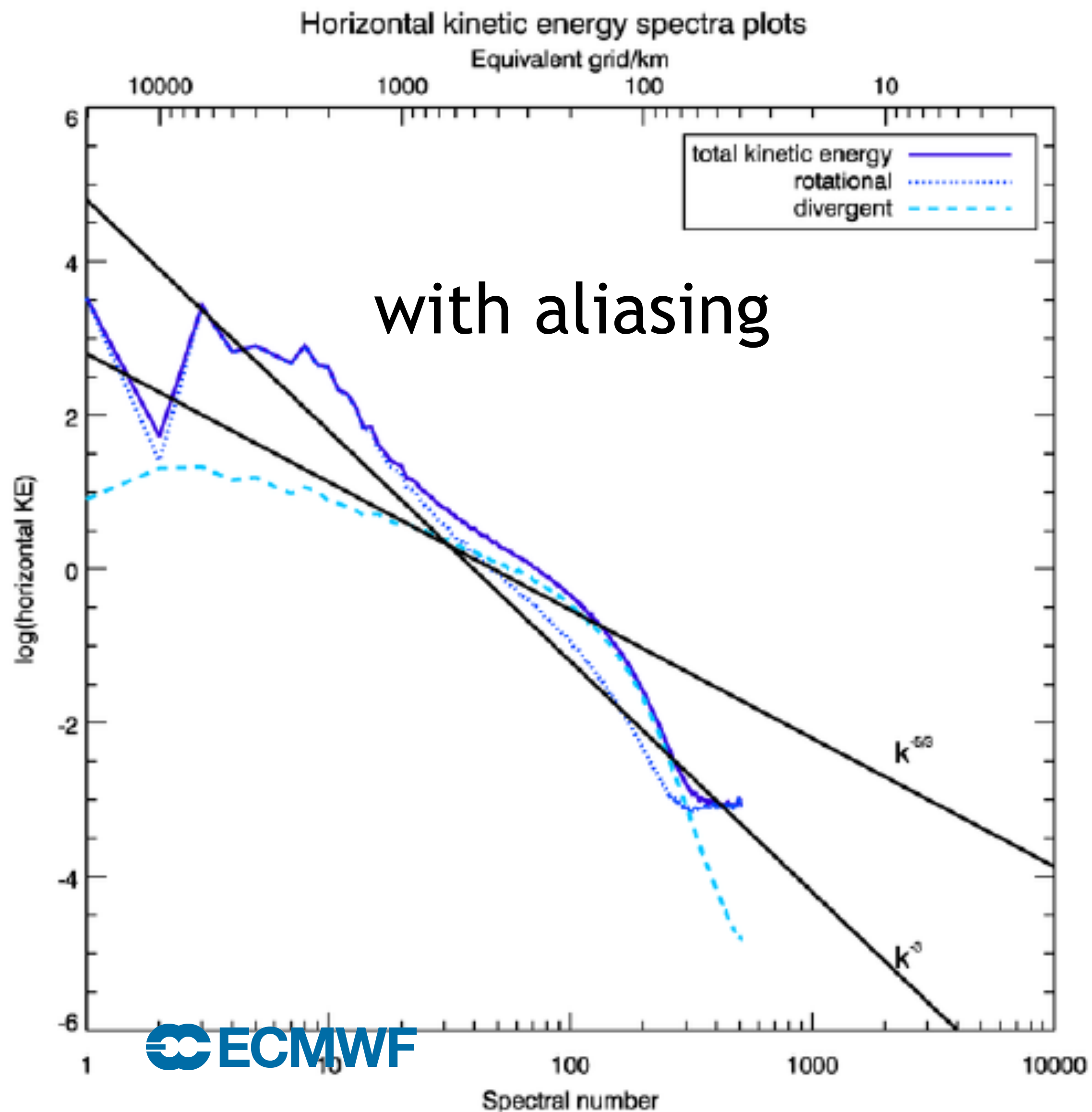


Monday 15 June 2009 00UTC ECMWF Forecast t+24 VT: Tuesday 16 June 2009 00UTC 500hPa Experimental product



# aliasing example

## kinetic energy spectra, 100 hPa





# alternatives to using a filter

**Idea:** use more grid points than spectral coefficients

Orszag, 1971:

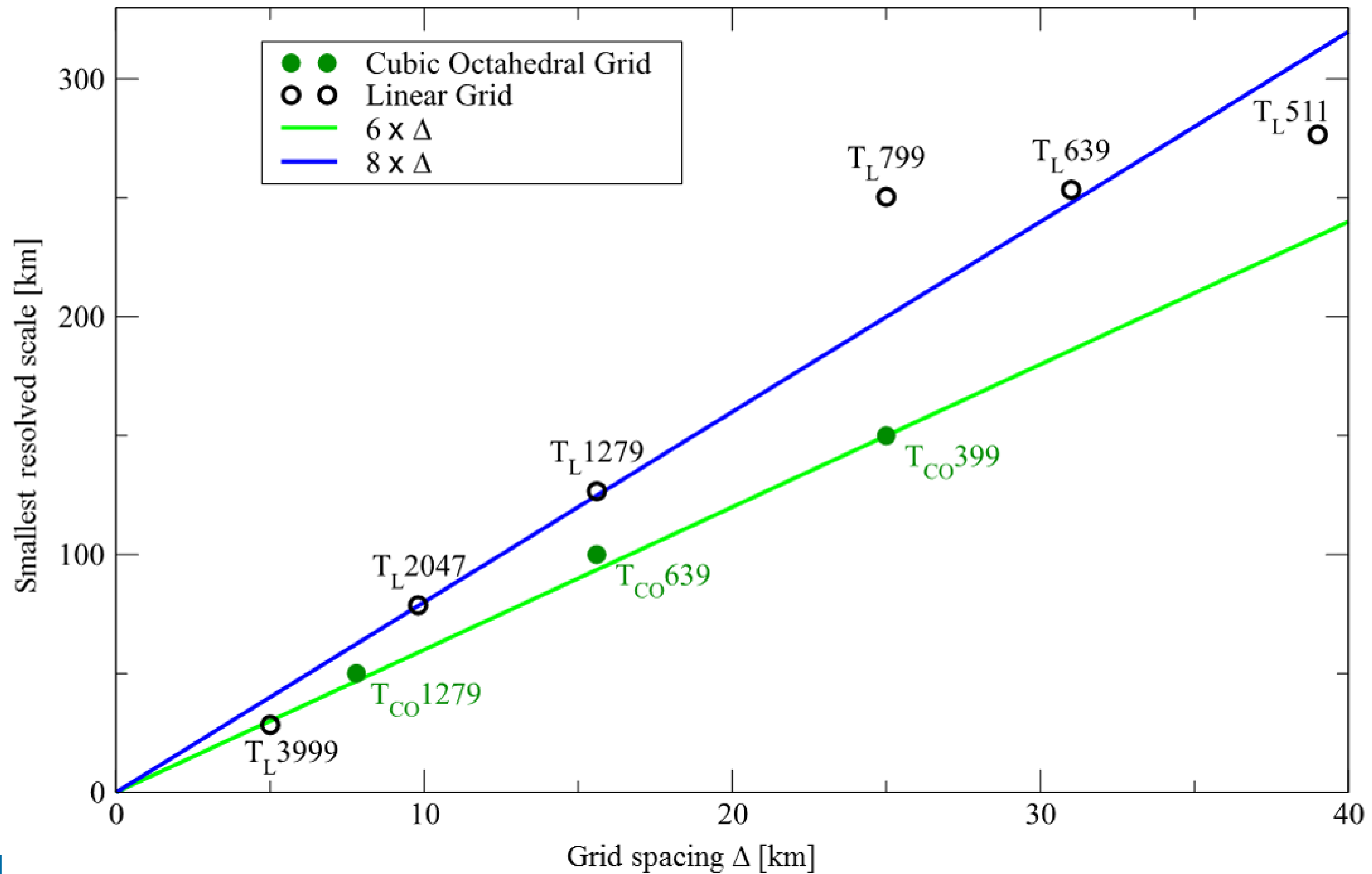
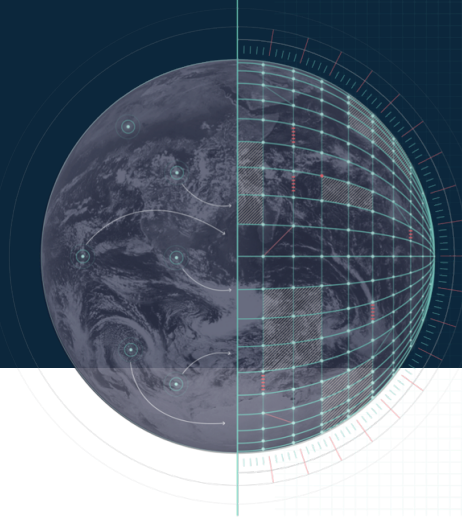
2N+1 gridpoints to N waves : linear grid ~ 1-2  $\Delta$

3N+1 gridpoints to N waves : quadratic grid ~ 2-3  $\Delta$

4N+1 gridpoints to N waves : cubic grid ~ 3-4  $\Delta$  (*Wedi, 2014*)

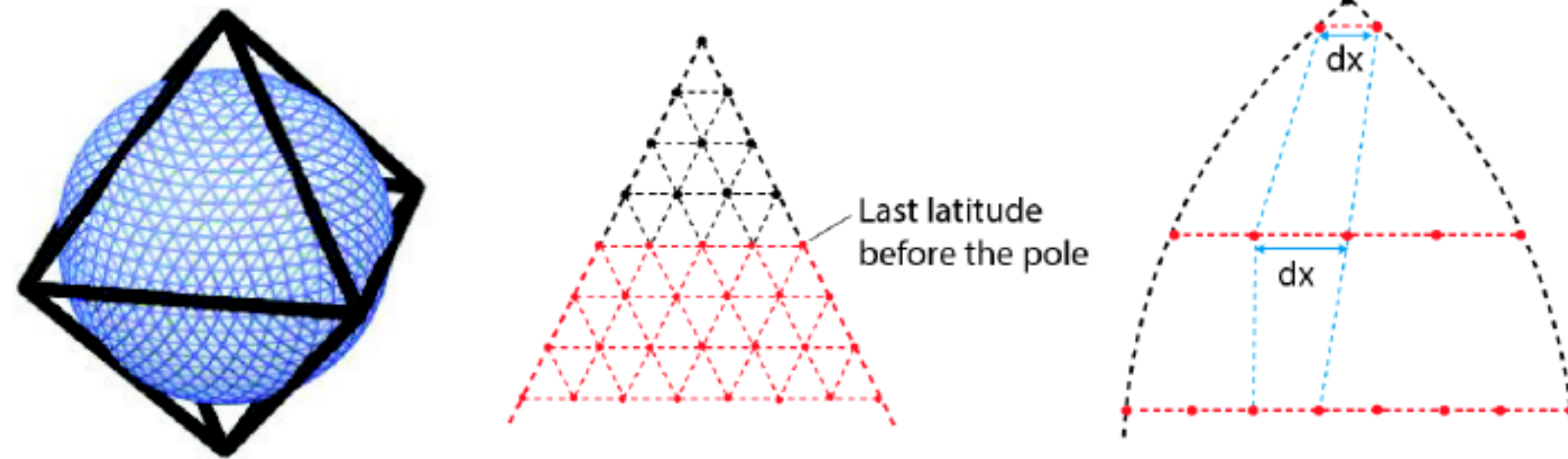
Spatial filter range

# effective resolution of linear and cubic grids (Abdalla et al. 2013)





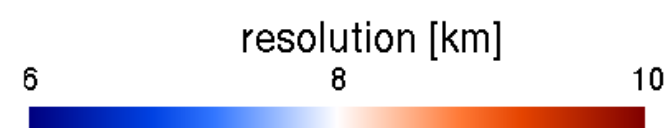
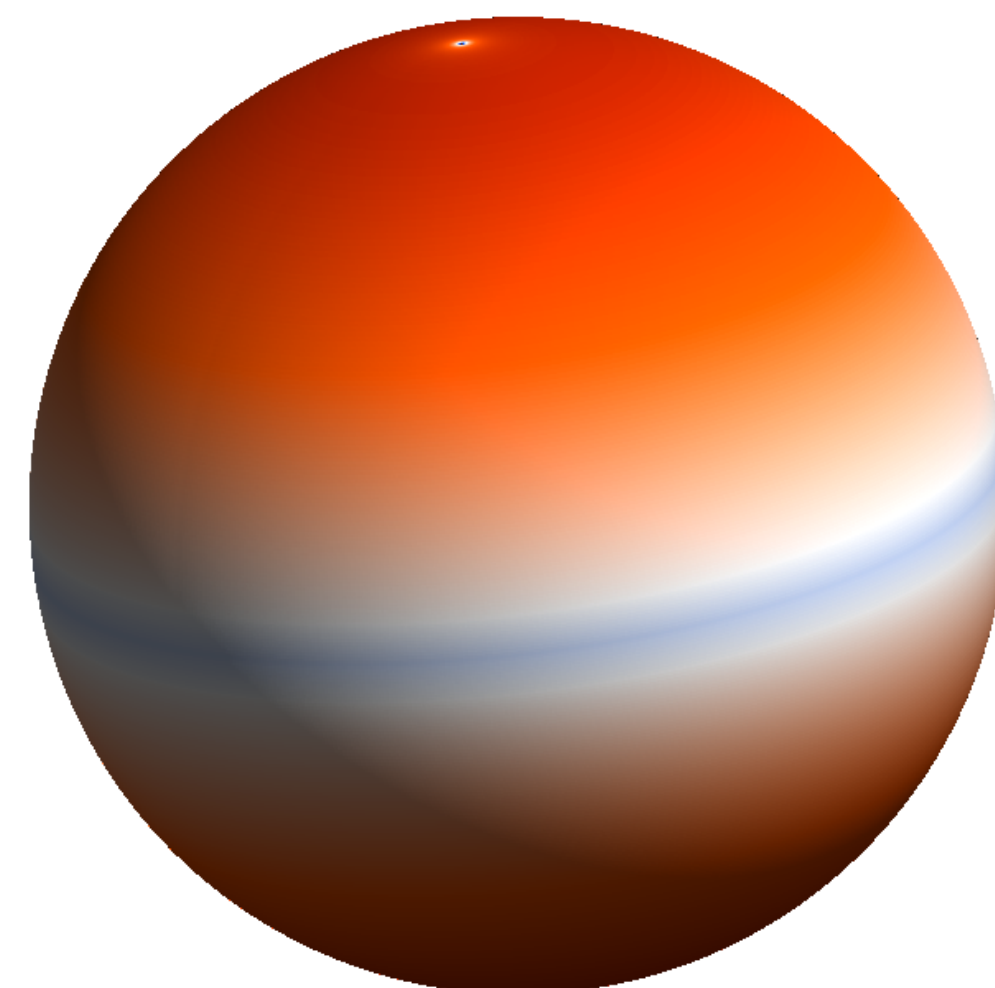
# Cubic octahedral (Gaussian) grid of IFS



- No aliasing in nonlinear products
- Improved accuracy and mass conservation compared with linear grid
- Efficiency and scalability for large size problems: high grid-point resolution for a given spectral truncation i.e. expensive transforms become a smaller fraction of total computations

Number of points at latitude line  $i = 4 \times i + 16$ ,  $i = 1, \dots, 2N$  (inspired by Collignon projection)

Variation of grid-point resolution with latitude



For a given spectral triangular truncation  $N$  the cubic reduced octahedral Gaussian grid has:

- $2N$  points between pole and equator which coincide with Gaussian latitudes
- $4N+16$  east-west points along the equator
- $4N(N+9)$  points in total



# inverse spectral transform

spectral data:  $D(f, i, n, m)$

fastest index left (column-major order like in Fortran)

fields (variables, height levels)

wave numbers  
 $m=0, \dots, N$ ;  $n=0, \dots, N-m$   
( $N$ : truncation)

real and imaginary part

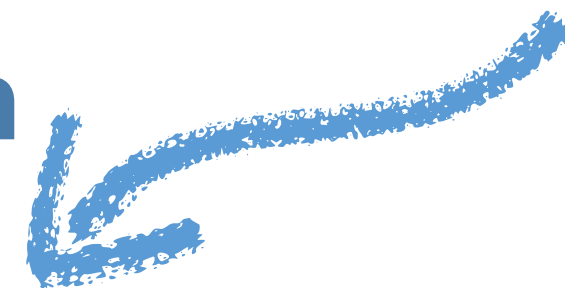


# inverse spectral transform

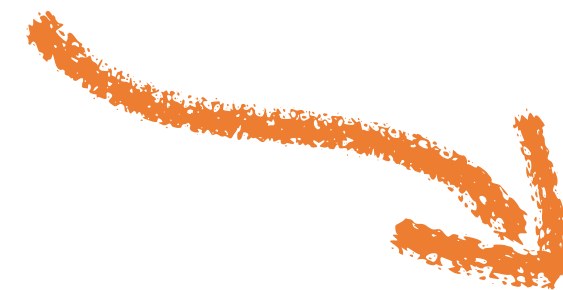
spectral data:  $D(f, i, n, m)$

$m=0, \dots, N$ ;  $n=0, \dots, N-m$

even  $n$



odd  $n$



for each  $m$ :

$$D_{e,m}(f, i, n)$$

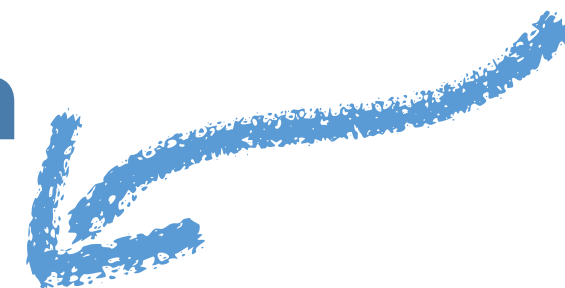
$$D_{o,m}(f, i, n)$$



# inverse spectral transform

spectral data:  $\mathbf{D}(f, i, n, m)$

even  $n$



odd  $n$



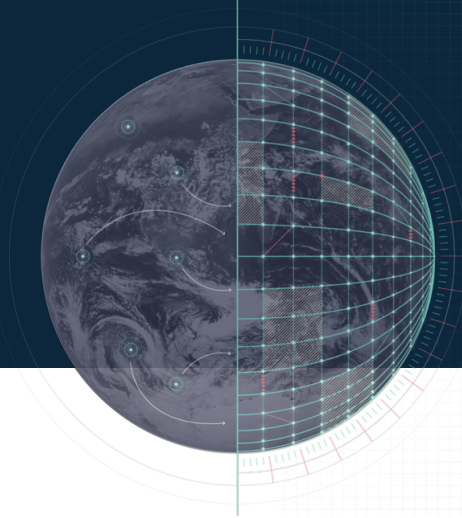
$m=0, \dots, N$ ;  $n=0, \dots, N-m$

$\mathbf{P}$ : precomputed Legendre polynomials

for each  $m$ :

$$\mathbf{S}_m(f, i, \phi) = \sum_n \mathbf{D}_{e,m}(f, i, n) \cdot \mathbf{P}_{e,m}(n, \phi), \quad \mathbf{A}_m(f, i, \phi) = \sum_n \mathbf{D}_{o,m}(f, i, n) \cdot \mathbf{P}_{o,m}(n, \phi)$$

matrix  
multiplications



# inverse spectral transform

spectral data:  $\mathbf{D}(f, i, n, m)$

even  $n$



odd  $n$



$m=0, \dots, N$ ;  $n=0, \dots, N-m$

$\mathbf{P}$ : precomputed Legendre polynomials

for each  $m$ :

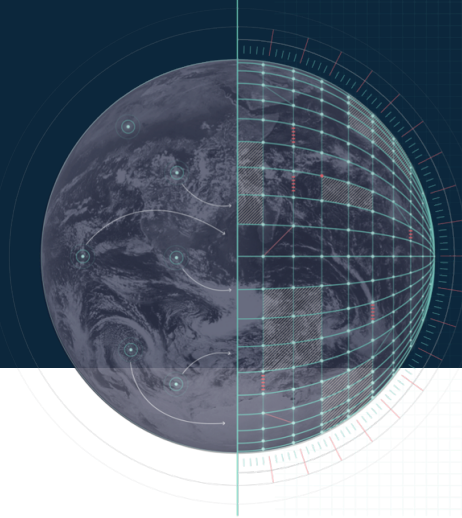
$$\mathbf{S}_m(f, i, \phi) = \sum_n \mathbf{D}_{e,m}(f, i, n) \cdot \mathbf{P}_{e,m}(n, \phi), \quad \mathbf{A}_m(f, i, \phi) = \sum_n \mathbf{D}_{o,m}(f, i, n) \cdot \mathbf{P}_{o,m}(n, \phi)$$



$$\phi > 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, \phi) + \mathbf{A}_m(f, i, \phi)$$

$$\phi < 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, -\phi) - \mathbf{A}_m(f, i, -\phi)$$

matrix  
multiplications



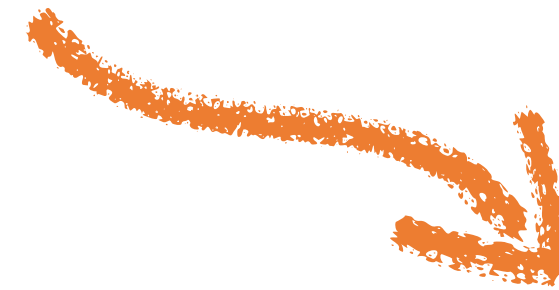
# inverse spectral transform

spectral data:  $\mathbf{D}(f, i, n, m)$

even  $n$



odd  $n$



$m=0, \dots, N$ ;  $n=0, \dots, N-m$

$\mathbf{P}$ : precomputed Legendre polynomials

for each  $m$ :

$$\mathbf{S}_m(f, i, \phi) = \sum_n \mathbf{D}_{e,m}(f, i, n) \cdot \mathbf{P}_{e,m}(n, \phi), \quad \mathbf{A}_m(f, i, \phi) = \sum_n \mathbf{D}_{o,m}(f, i, n) \cdot \mathbf{P}_{o,m}(n, \phi)$$



$$\phi > 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, \phi) + \mathbf{A}_m(f, i, \phi)$$

$$\phi < 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, -\phi) - \mathbf{A}_m(f, i, -\phi)$$

matrix multiplications

for each  $\phi, f$ :

$$\mathbf{G}_{\phi, f}(\lambda) = \text{FFT}(\mathbf{F}_{\phi, f}(i, m))$$

FFT: Fast Fourier Transform



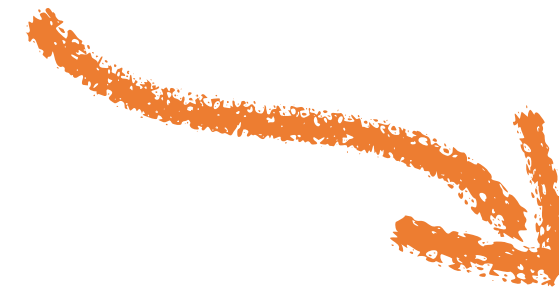
# inverse spectral transform

spectral data:  $\mathbf{D}(f, i, n, m)$

even  $n$



odd  $n$



$m=0, \dots, N$ ;  $n=0, \dots, N-m$

$\mathbf{P}$ : precomputed Legendre polynomials

for each  $m$ :

$$\mathbf{S}_m(f, i, \phi) = \sum_n \mathbf{D}_{e,m}(f, i, n) \cdot \mathbf{P}_{e,m}(n, \phi), \quad \mathbf{A}_m(f, i, \phi) = \sum_n \mathbf{D}_{o,m}(f, i, n) \cdot \mathbf{P}_{o,m}(n, \phi)$$



$$\phi > 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, \phi) + \mathbf{A}_m(f, i, \phi)$$

$$\phi < 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, -\phi) - \mathbf{A}_m(f, i, -\phi)$$

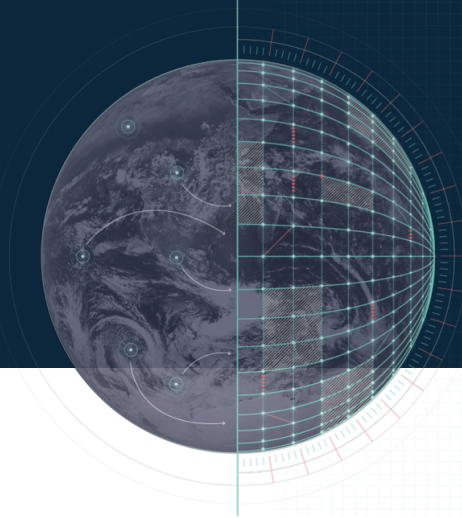
matrix multiplications

for each  $\phi, f$ :

$$\mathbf{G}_{\phi, f}(\lambda) = \text{FFT}(\mathbf{F}_{\phi, f}(i, m))$$

FFT: Fast Fourier Transform

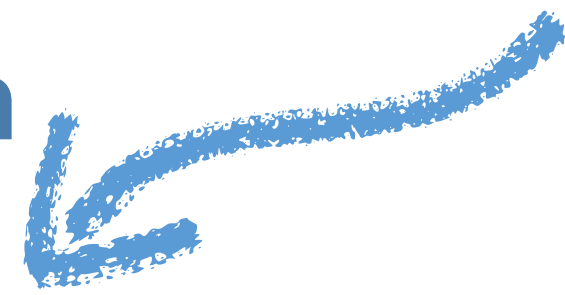
grid point data:  $\mathbf{G}(f, \lambda, \phi)$



# inverse spectral transform

spectral data:  $\mathbf{D}(f, i, n, m)$

even  $n$



odd  $n$



for each  $m$ :

$$\mathbf{S}_m(f, i, \phi) = \sum_n \mathbf{D}_{e,m}(f, i, n) \cdot \mathbf{P}_{e,m}(n, \phi),$$

$$\mathbf{A}_m(f, i, \phi) = \sum_n \mathbf{D}_{o,m}(f, i, n) \cdot \mathbf{P}_{o,m}(n, \phi)$$

$$\phi > 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, \phi) + \mathbf{A}_m(f, i, \phi)$$

$$\phi < 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, -\phi) - \mathbf{A}_m(f, i, -\phi)$$

spectral space

inverse Legendre transform

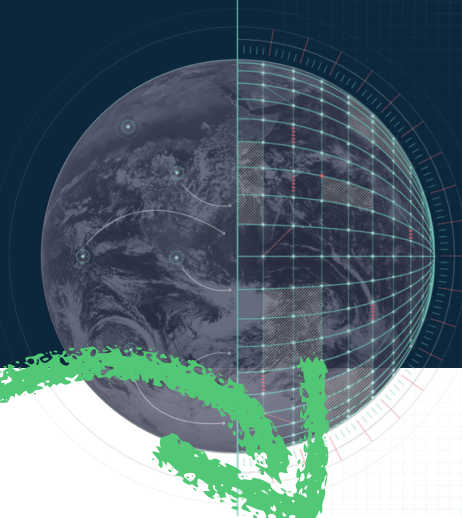
inverse Fourier transform

grid point space

for each  $\phi, f$ :  $\mathbf{G}_{\phi, f}(\lambda) = \text{FFT}(\mathbf{F}_{\phi, f}(i, m))$

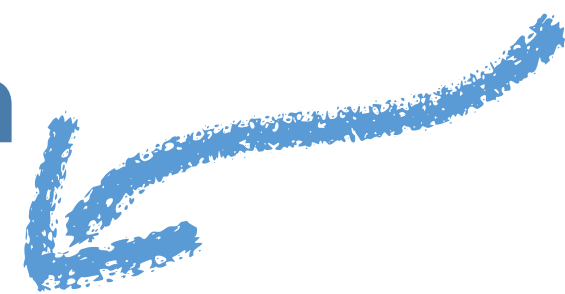
grid point data:  $\mathbf{G}(f, \lambda, \phi)$

# inverse spectral transform



spectral data:  $\mathbf{D}(f, i, n, m)$

even  $n$



odd  $n$



for each  $m$ :

$$\mathbf{S}_m(f, i, \phi) = \sum_n \mathbf{D}_{e,m}(f, i, n) \cdot \mathbf{P}_{e,m}(n, \phi),$$

$$\mathbf{A}_m(f, i, \phi) = \sum_n \mathbf{D}_{o,m}(f, i, n) \cdot \mathbf{P}_{o,m}(n, \phi)$$

$$\phi > 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, \phi) + \mathbf{A}_m(f, i, \phi)$$

$$\phi < 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, -\phi) - \mathbf{A}_m(f, i, -\phi)$$

for each  $\phi, f$ :  $\mathbf{G}_{\phi, f}(\lambda) = \text{FFT}(\mathbf{F}_{\phi, f}(i, m))$

grid point data:  $\mathbf{G}(f, \lambda, \phi)$

spectral space

$m, n$

parallelisation  
over these  
indices

inverse Legendre transform

$m, f$

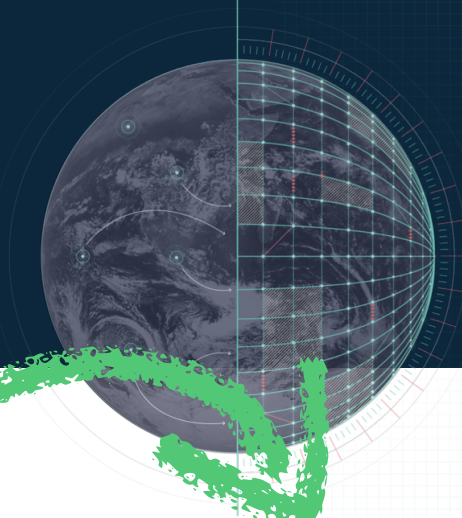
inverse Fourier transform

$\phi, f$

grid point space

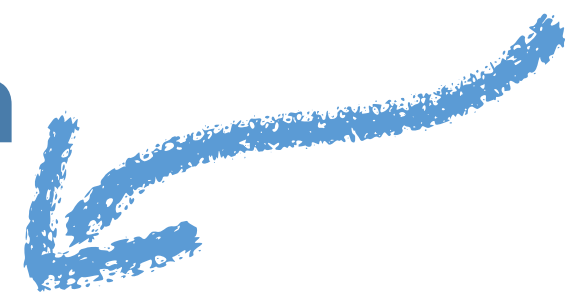
$\phi, \lambda$

# inverse spectral transform

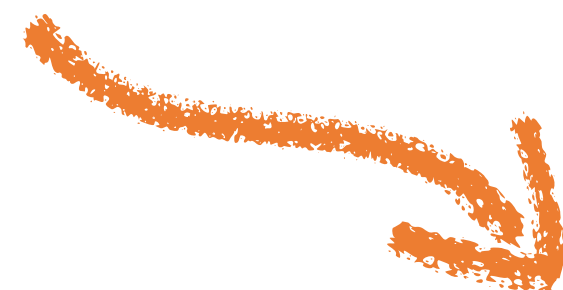


spectral data:  $\mathbf{D}(f, i, n, m)$

even  $n$



odd  $n$



for each  $m$ :

$$\mathbf{S}_m(f, i, \phi) = \sum_n \mathbf{D}_{e,m}(f, i, n) \cdot \mathbf{P}_{e,m}(n, \phi),$$

$$\mathbf{A}_m(f, i, \phi) = \sum_n \mathbf{D}_{o,m}(f, i, n) \cdot \mathbf{P}_{o,m}(n, \phi)$$

$$\phi > 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, \phi) + \mathbf{A}_m(f, i, \phi)$$

$$\phi < 0 : \mathbf{F}(i, m, \phi, f) = \mathbf{S}_m(f, i, -\phi) - \mathbf{A}_m(f, i, -\phi)$$

for each  $\phi, f$ :  $\mathbf{G}_{\phi, f}(\lambda) = \text{FFT}(\mathbf{F}_{\phi, f}(i, m))$

grid point data:  $\mathbf{G}(f, \lambda, \phi)$



spectral space

parallelisation  
over these  
indices

lots of MPI  
communication

inverse Legendre transform

inverse Fourier transform

grid point space

$m, n$



$m, f$



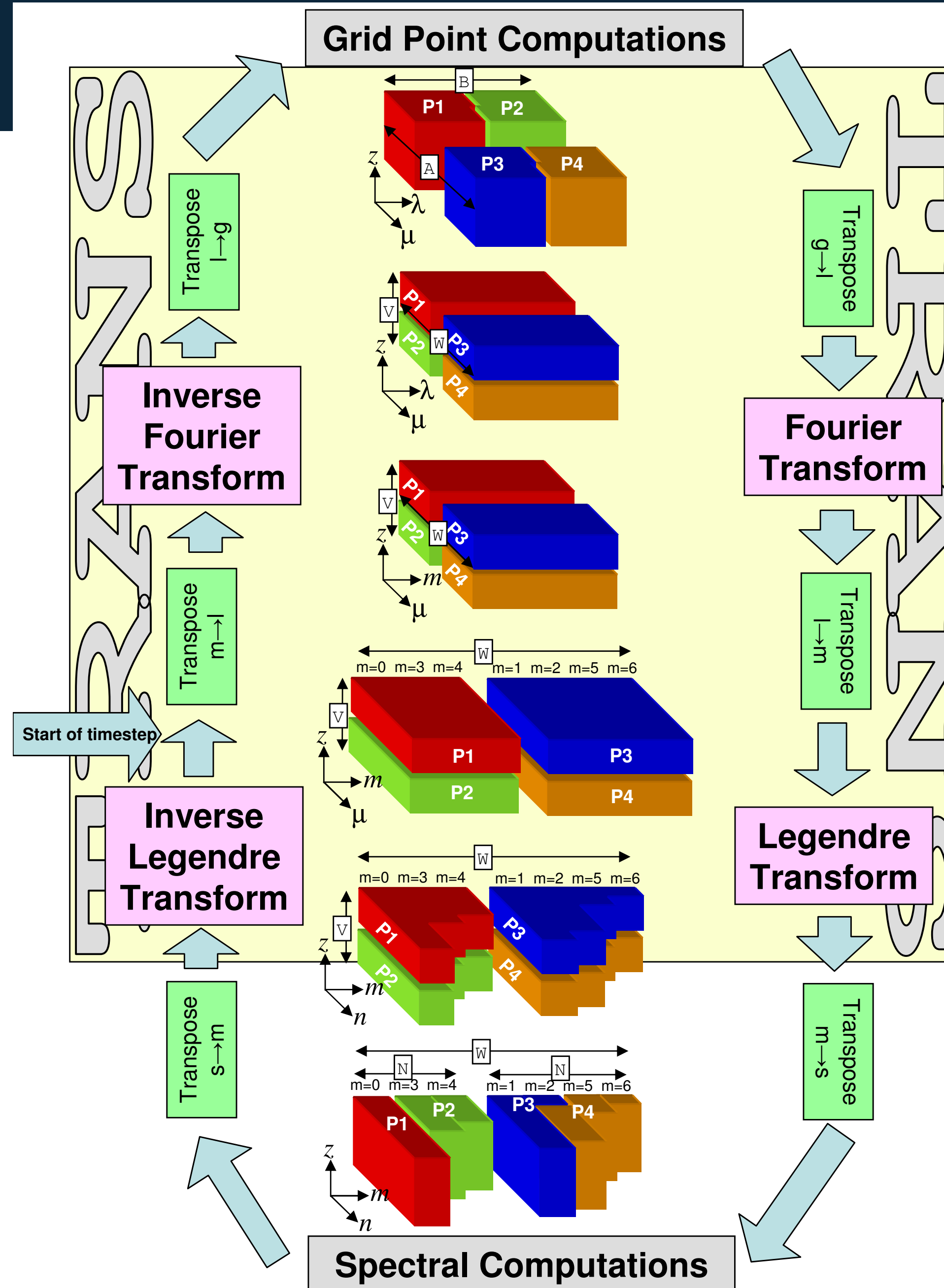
$\phi, f$



$\phi, \lambda$

# direct spectral transform

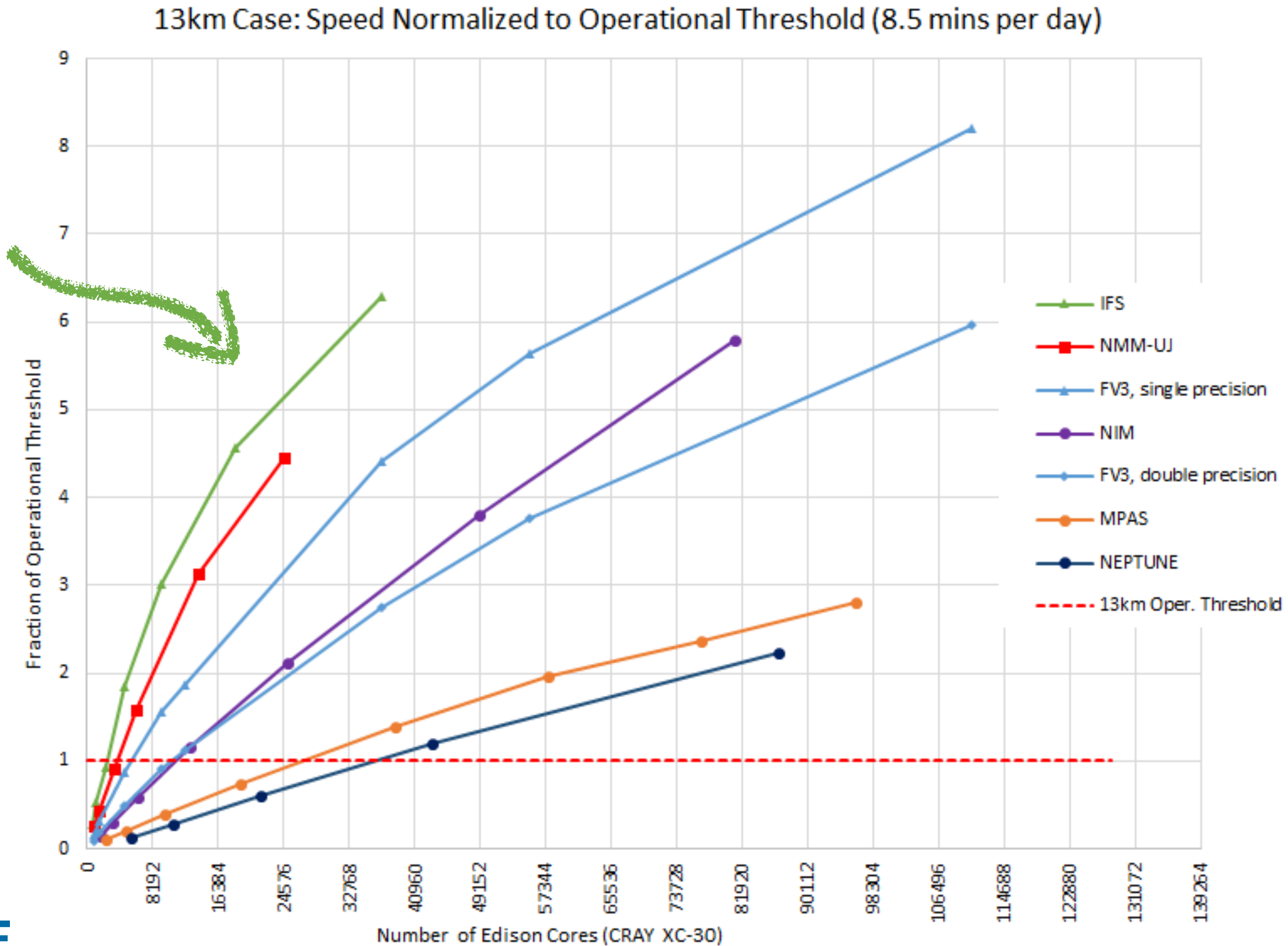
- same like inverse spectral transform
- reverse order
- multiply data with Gaussian quadrature weights before Legendre transform



# performance comparison of IFS with other models



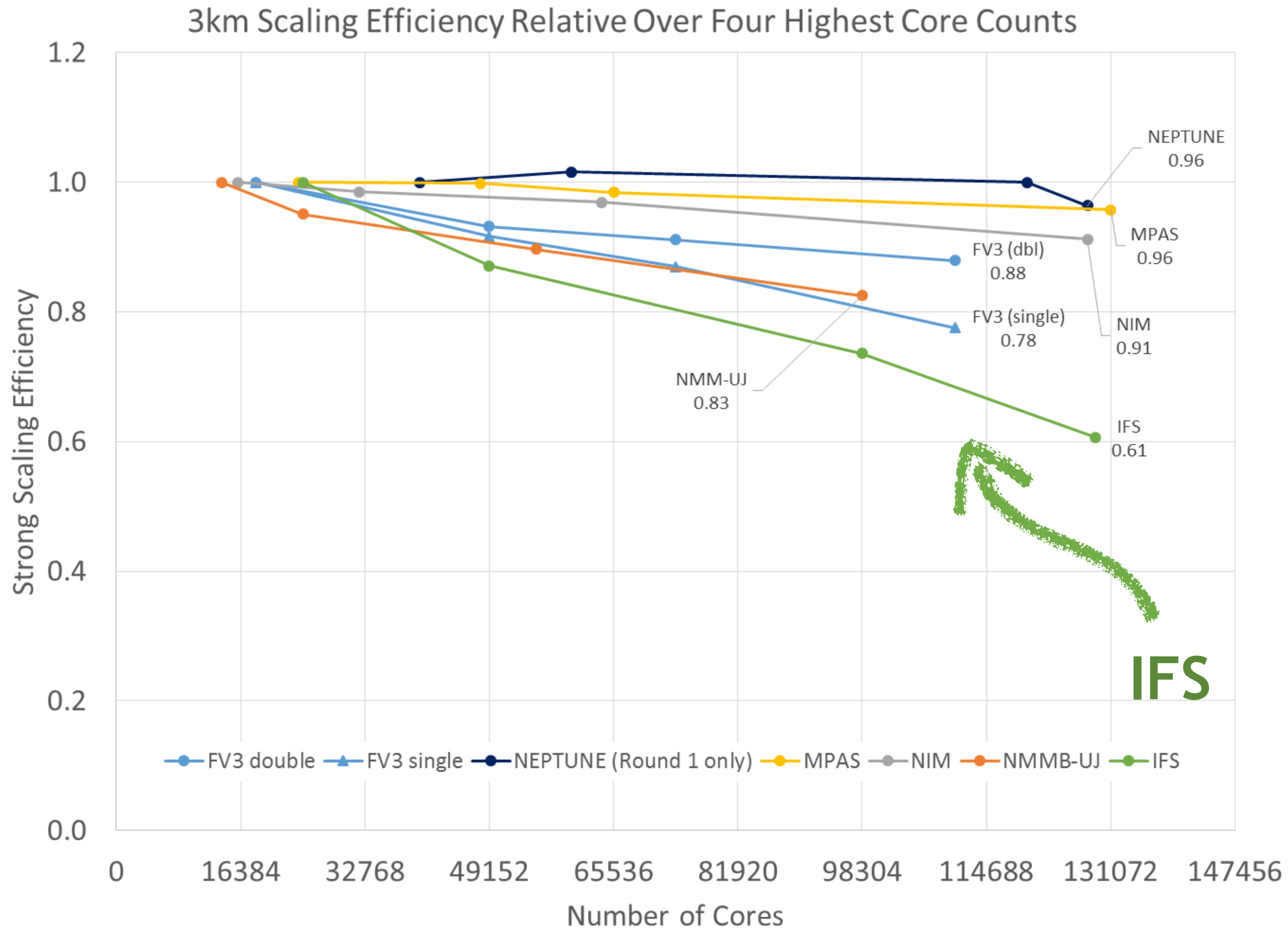
IFS



(Michalakes et al, NGGPS AVEC report, 2015)



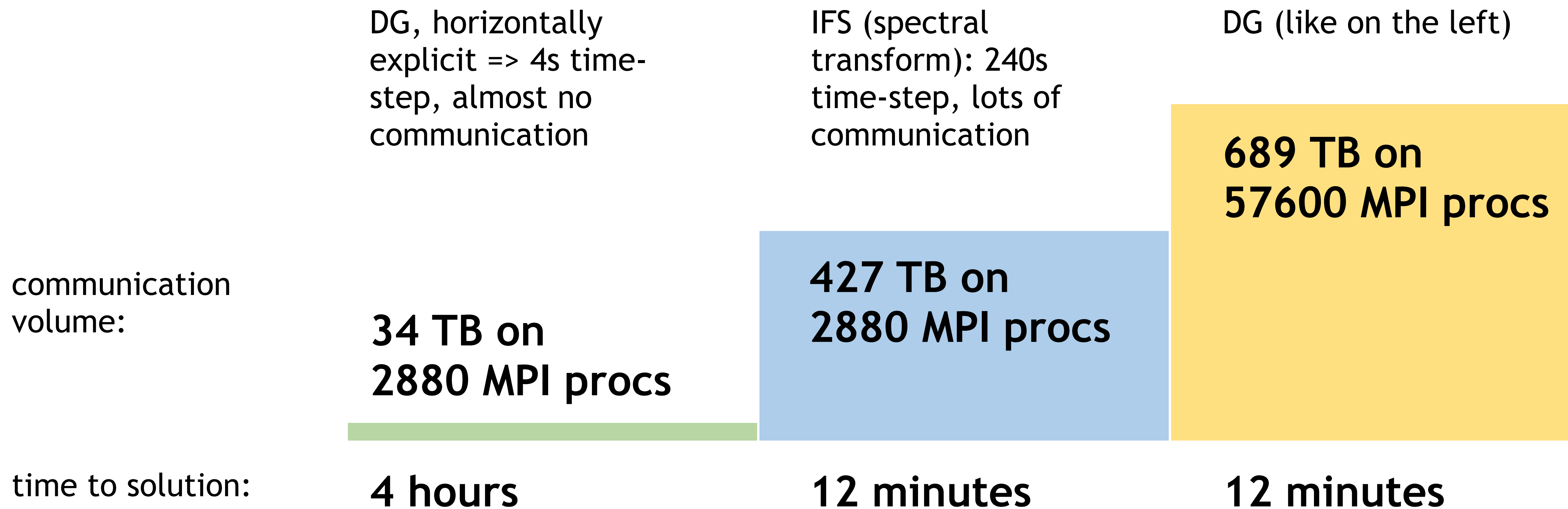
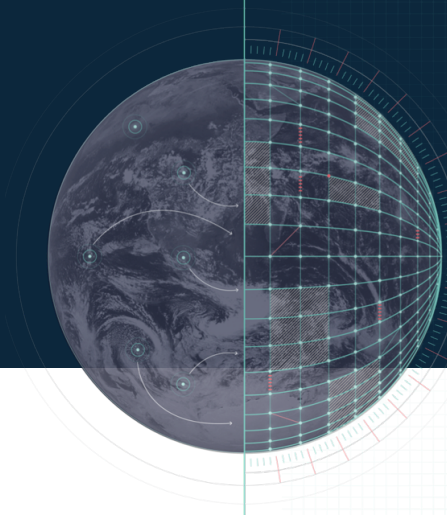
# scalability comparison of IFS with other models



(Michalakes et al, NGGPS AVEC report, 2015)

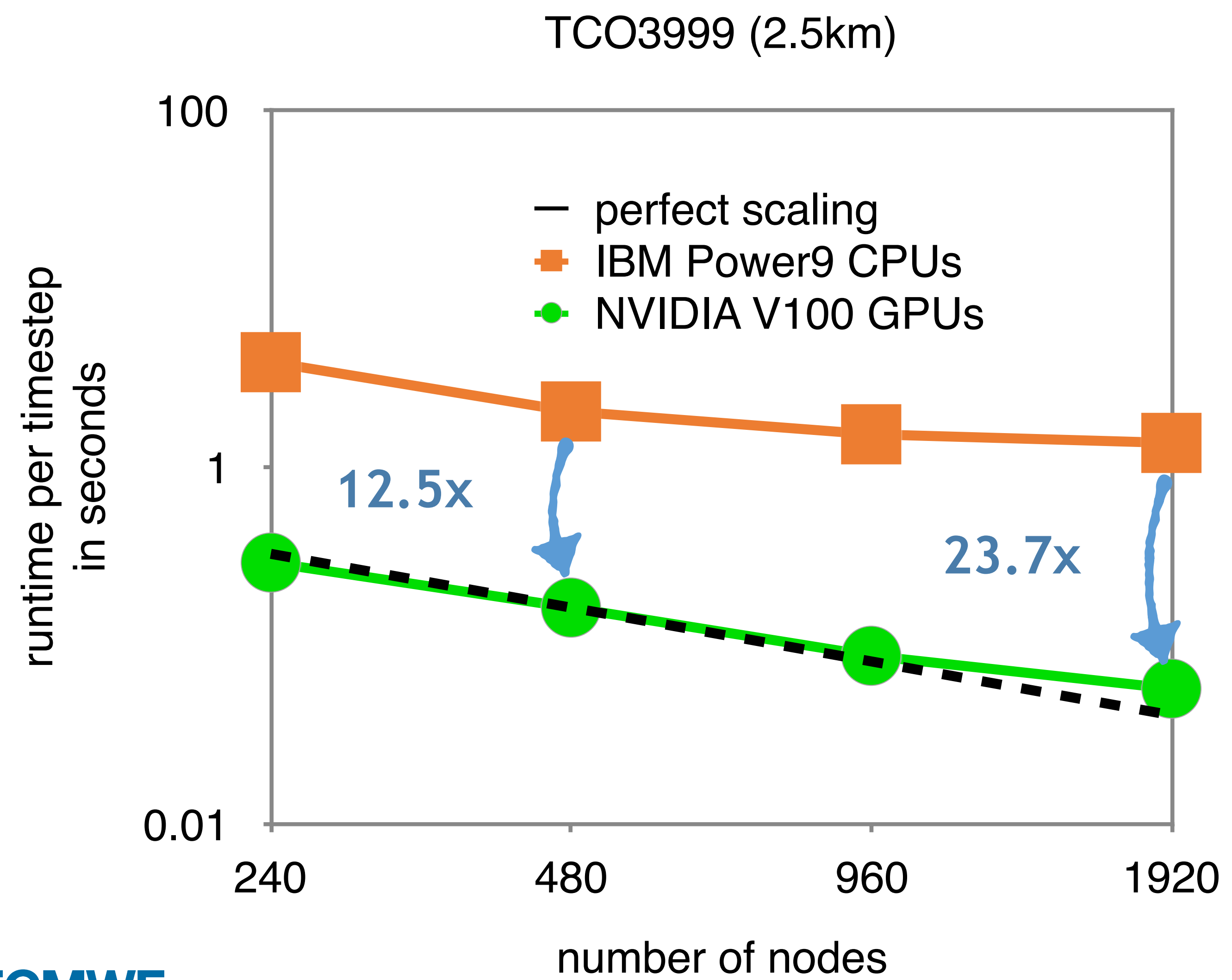
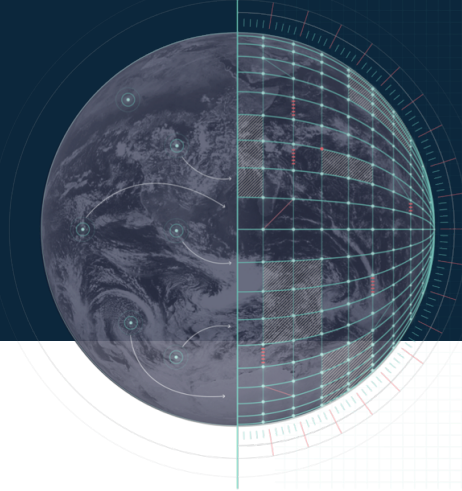
# spectral transform vs discontinuous Galerkin

projected for 5km 2-day forecast



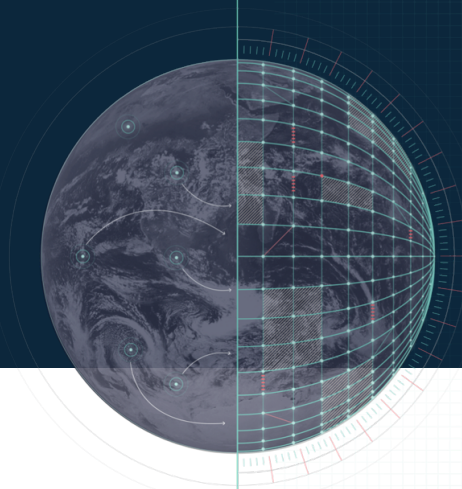
# GPUs vs CPUs on Summit

spectral transform only

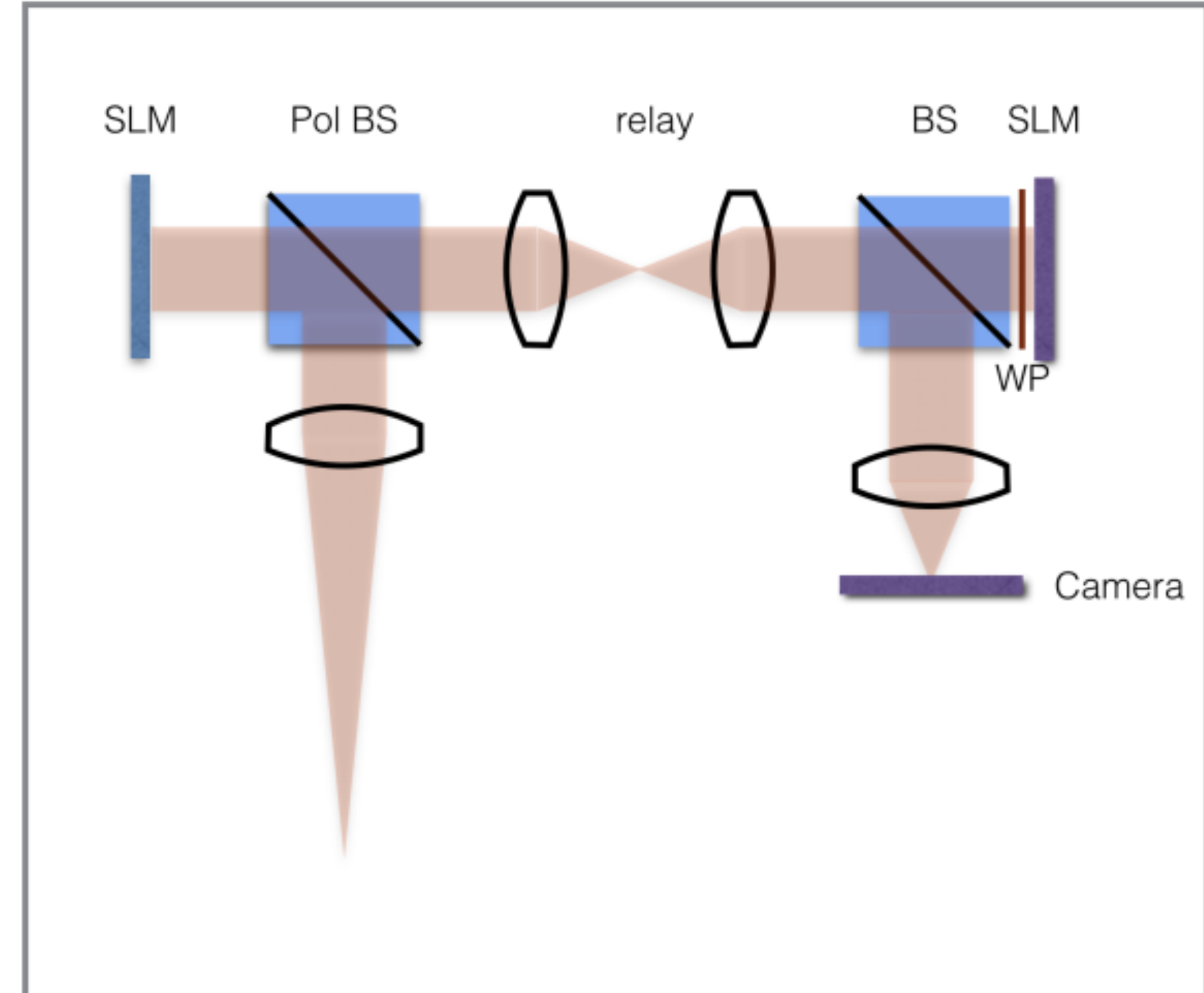
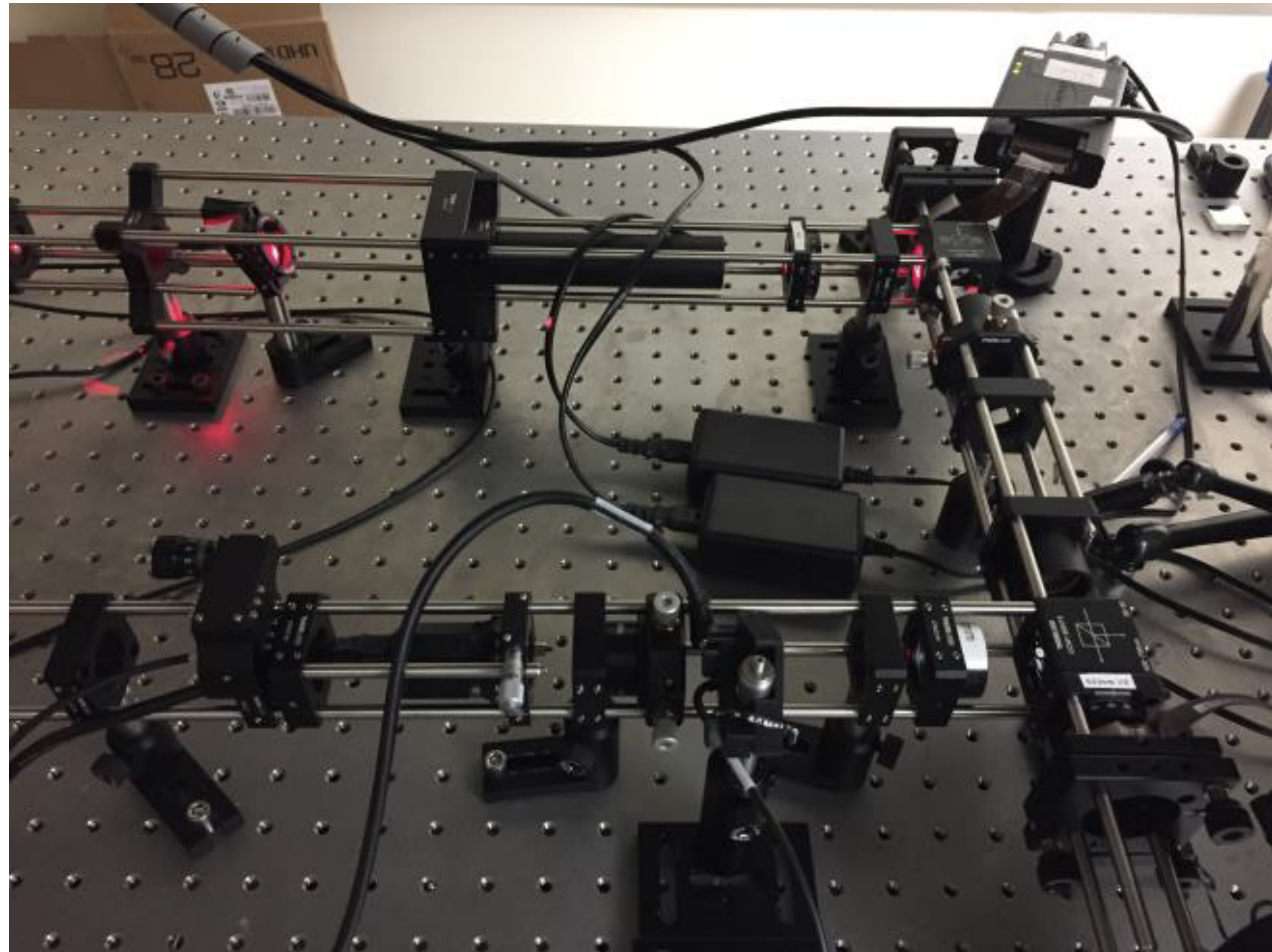


At 2.5km resolution, less than 1s per time-step fits operational needs.

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE office of Science User Facility supported under contract DE-AC05-00OR22725.



# Optalysys: optical processor for spectral transform



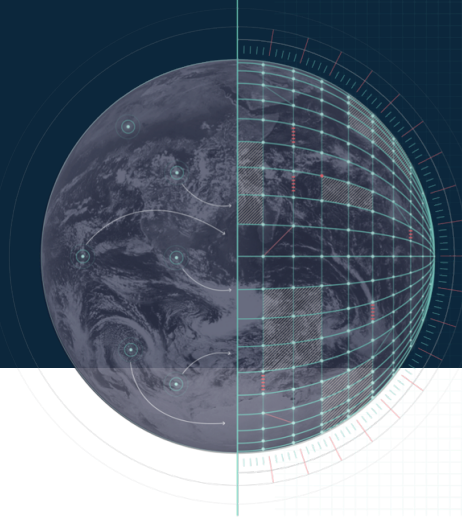
Figures used with permission from Optalysys, 2017

# Fast Legendre Transform



- recap: Legendre transform:

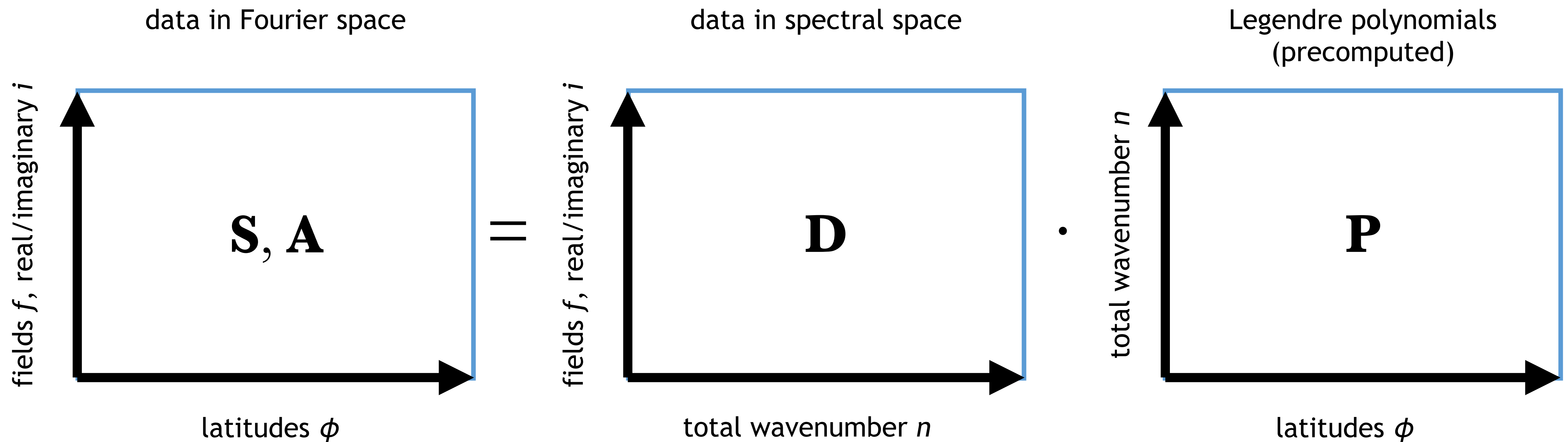
$$\mathbf{S}_m(f, i, \phi) = \sum_n \mathbf{D}_{e,m}(f, i, n) \cdot \mathbf{P}_{e,m}(n, \phi), \quad \mathbf{A}_m(f, i, \phi) = \sum_n \mathbf{D}_{o,m}(f, i, n) \cdot \mathbf{P}_{o,m}(n, \phi)$$



# Fast Legendre Transform

- recap: Legendre transform:

$$\mathbf{S}_m(f, i, \phi) = \sum_n \mathbf{D}_{e,m}(f, i, n) \cdot \mathbf{P}_{e,m}(n, \phi), \quad \mathbf{A}_m(f, i, \phi) = \sum_n \mathbf{D}_{o,m}(f, i, n) \cdot \mathbf{P}_{o,m}(n, \phi)$$

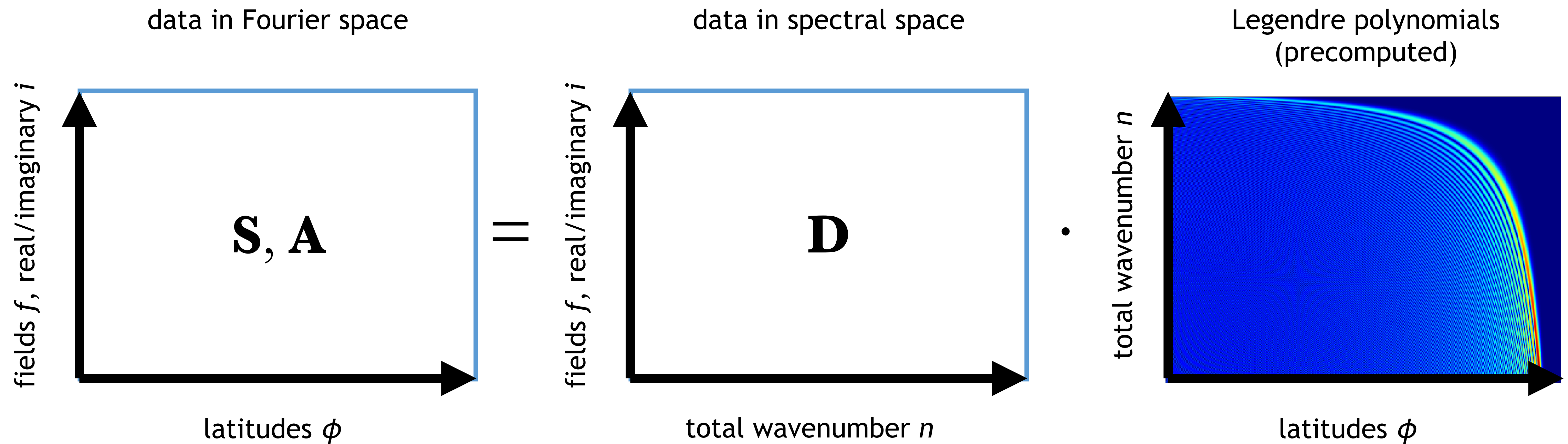


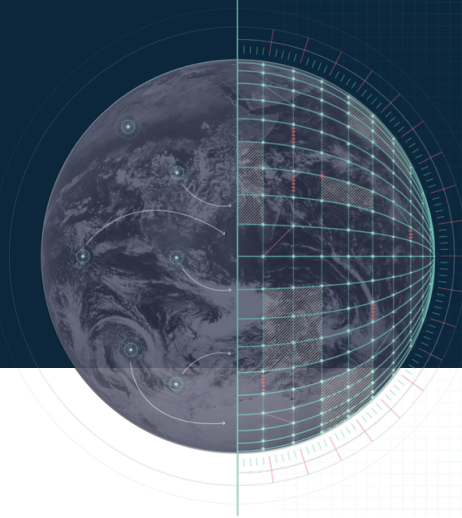


# Fast Legendre Transform

- recap: Legendre transform:

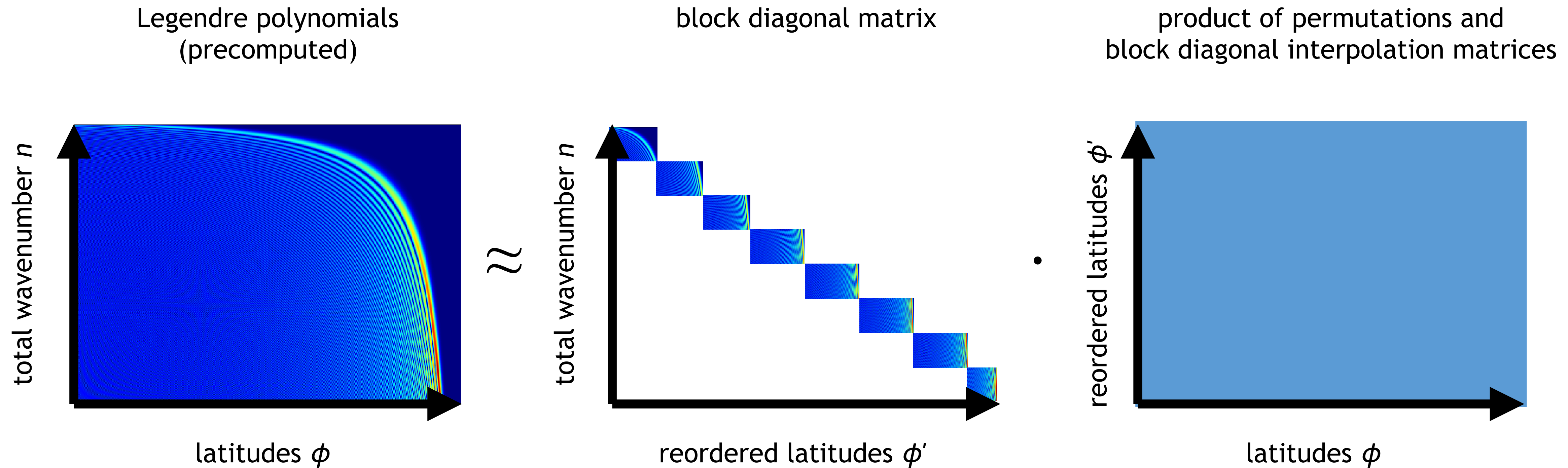
$$\mathbf{S}_m(f, i, \phi) = \sum_n \mathbf{D}_{e,m}(f, i, n) \cdot \mathbf{P}_{e,m}(n, \phi), \quad \mathbf{A}_m(f, i, \phi) = \sum_n \mathbf{D}_{o,m}(f, i, n) \cdot \mathbf{P}_{o,m}(n, \phi)$$



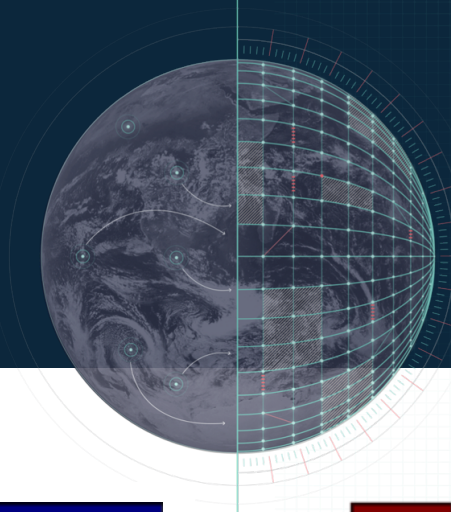


# Fast Legendre Transform

- idea behind the Fast Legendre Transform (explanation based on Figure 1 of Seljebotn, 2012):



# Fast Legendre Transform



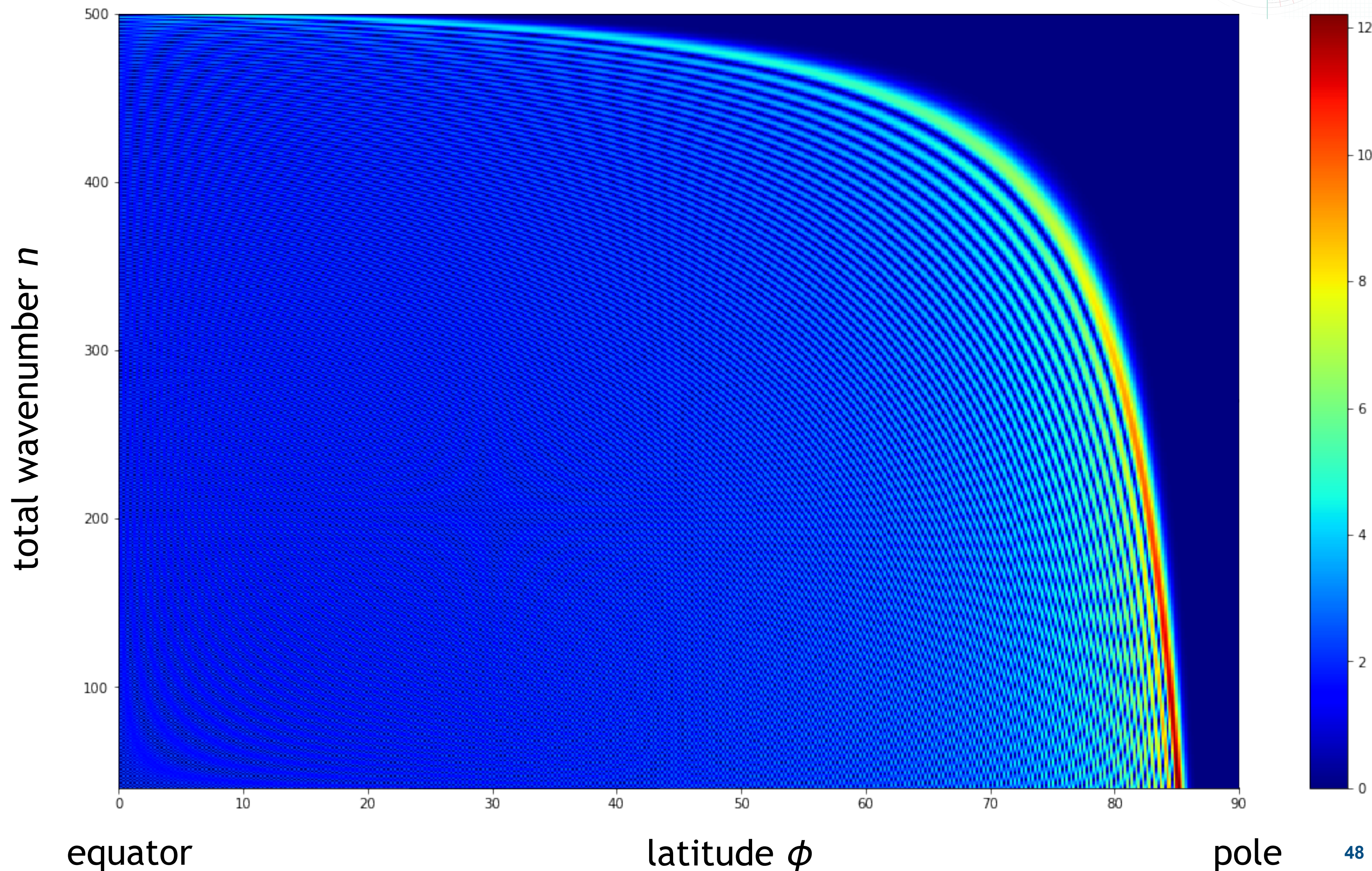
matrix of  
Legendre polynomials

truncation  $N=500$ ,  
zonal wavenumber  
 $m=40$

**FLT:**

**step 1:** split matrix  
into two rows

**step 2:** use  
interpolation to  
empty half of the  
columns





# Fast Legendre Transform

matrix of Legendre polynomials

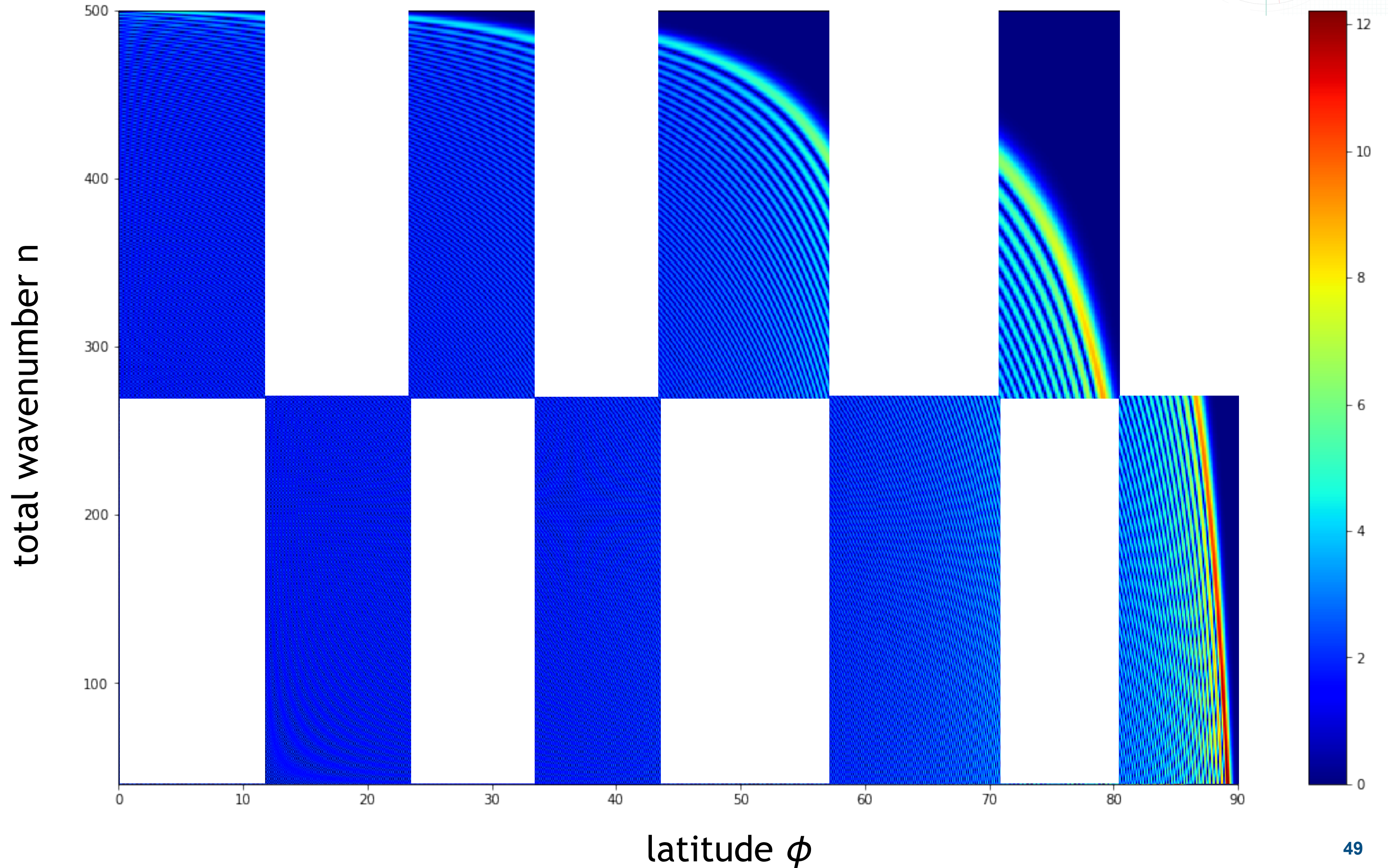
truncation  $N=500$ ,  
zonal wavenumber  
 $m=40$

**FLT:**

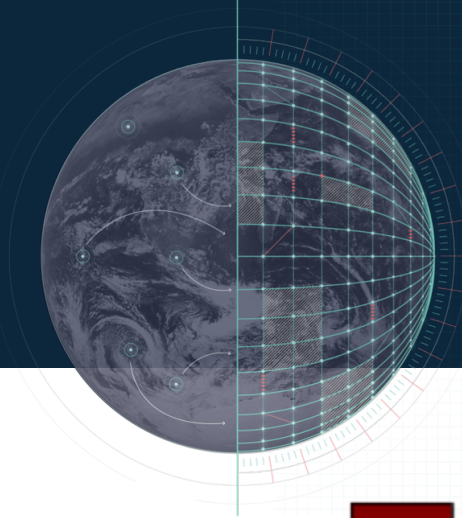
**step 1:** split matrix  
into two rows

**step 2:** use  
interpolation to  
empty half of the  
columns

**step 3:** reorder  
columns



# Fast Legendre Transform



matrix of  
Legendre polynomials

truncation  $N=500$ ,  
zonal wavenumber  
 $m=40$

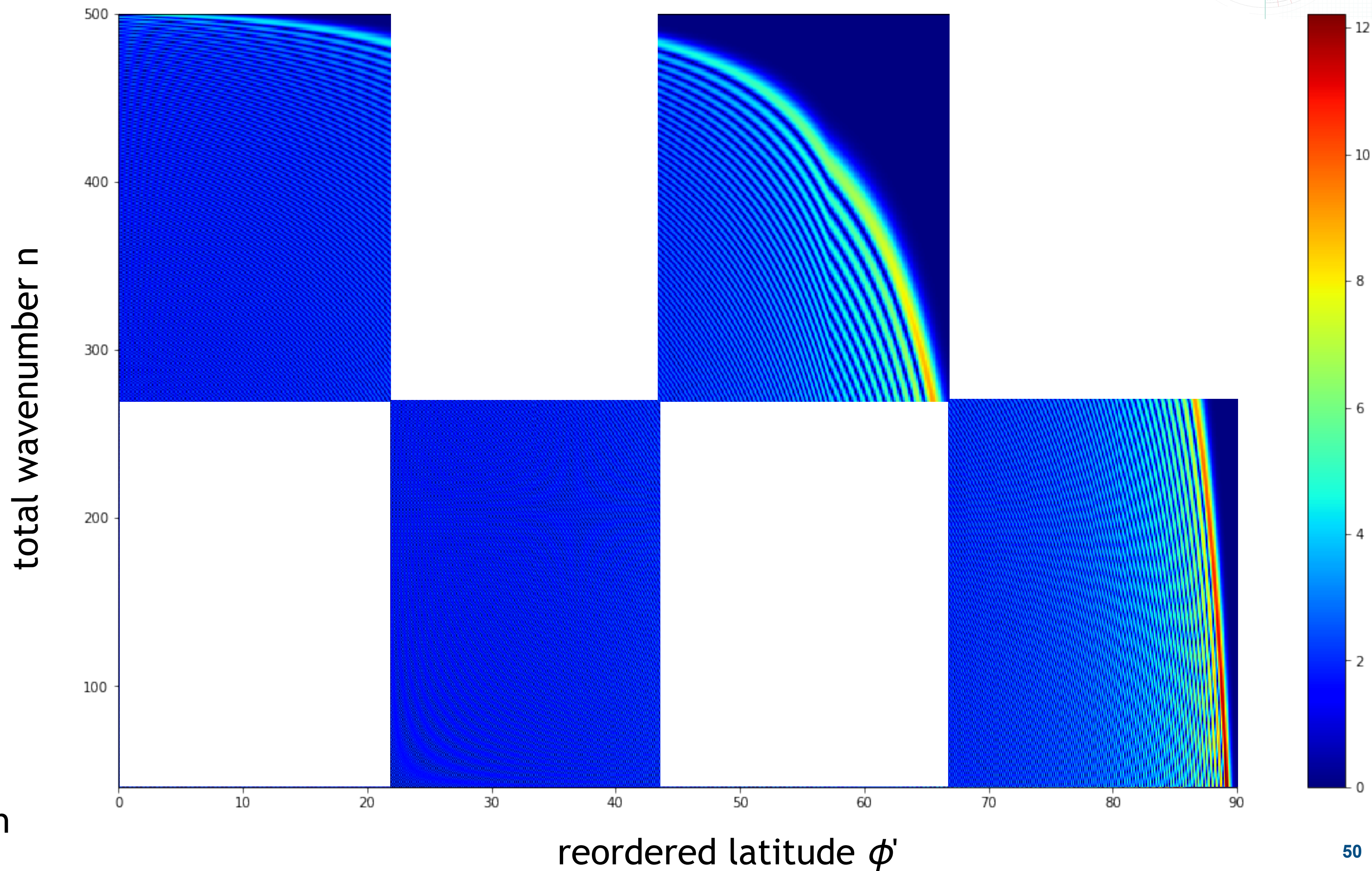
**FLT:**

**step 1:** split matrix  
into two rows

**step 2:** use  
interpolation to  
empty half of the  
columns

**step 3:** reorder  
columns

**step 4:** apply to each  
block recursively



# Fast Legendre Transform



matrix of Legendre polynomials

truncation  $N=500$ ,  
zonal wavenumber  
 $m=40$

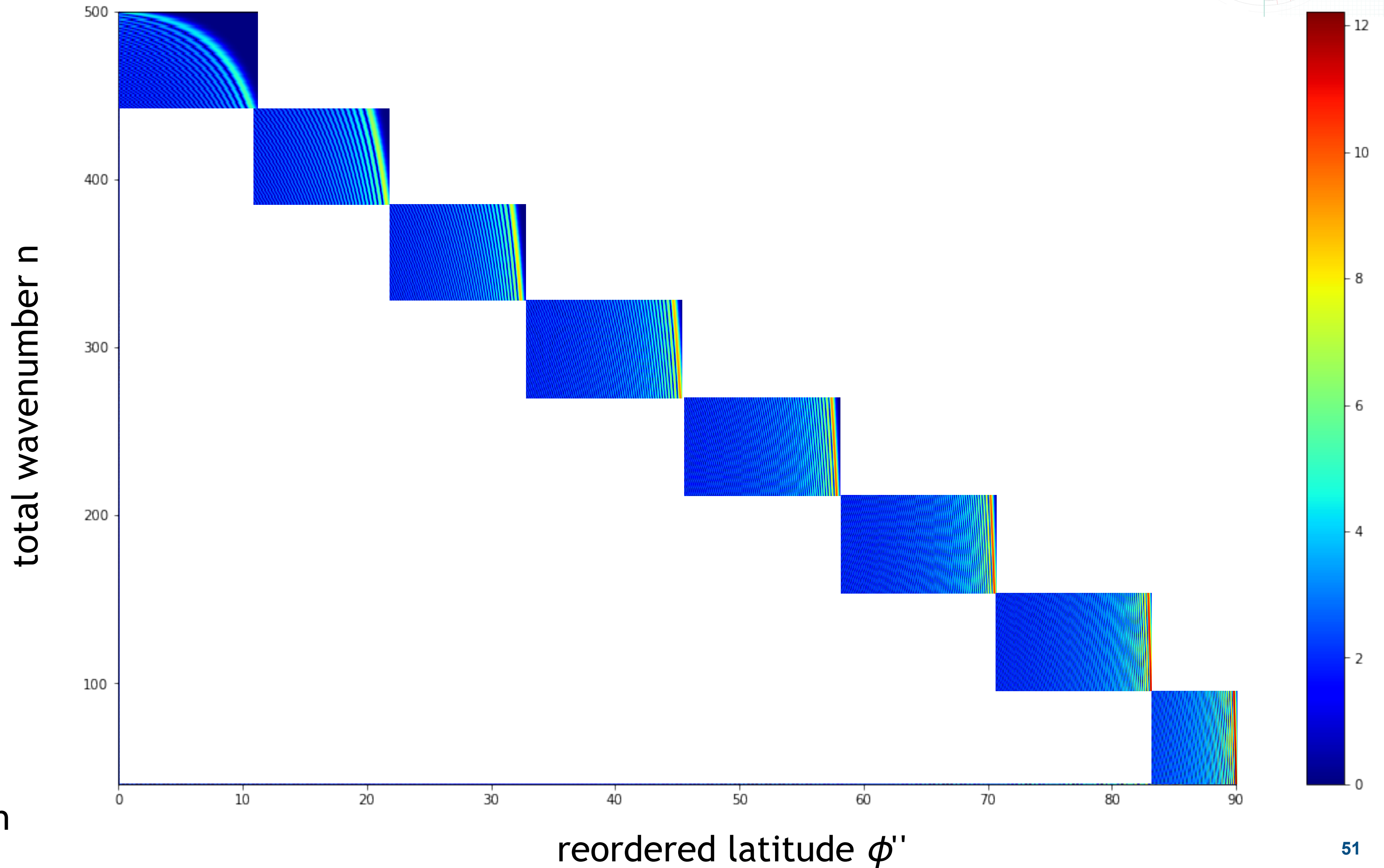
**FLT:**

**step 1:** split matrix  
into two rows

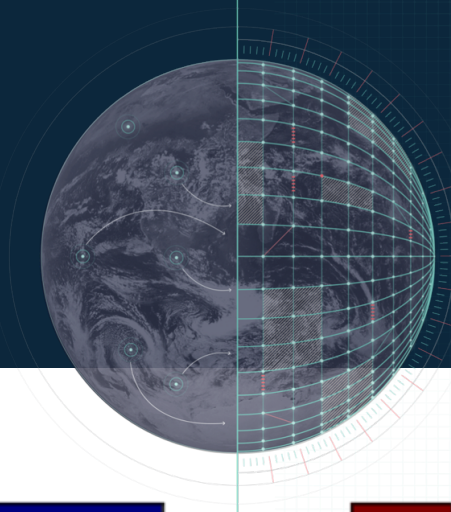
**step 2:** use  
interpolation to  
empty half of the  
columns

**step 3:** reorder  
columns

**step 4:** apply to each  
block recursively



# Fast Legendre Transform



matrix of  
Legendre polynomials

truncation  $N=500$ ,  
zonal wavenumber  
 $m=40$

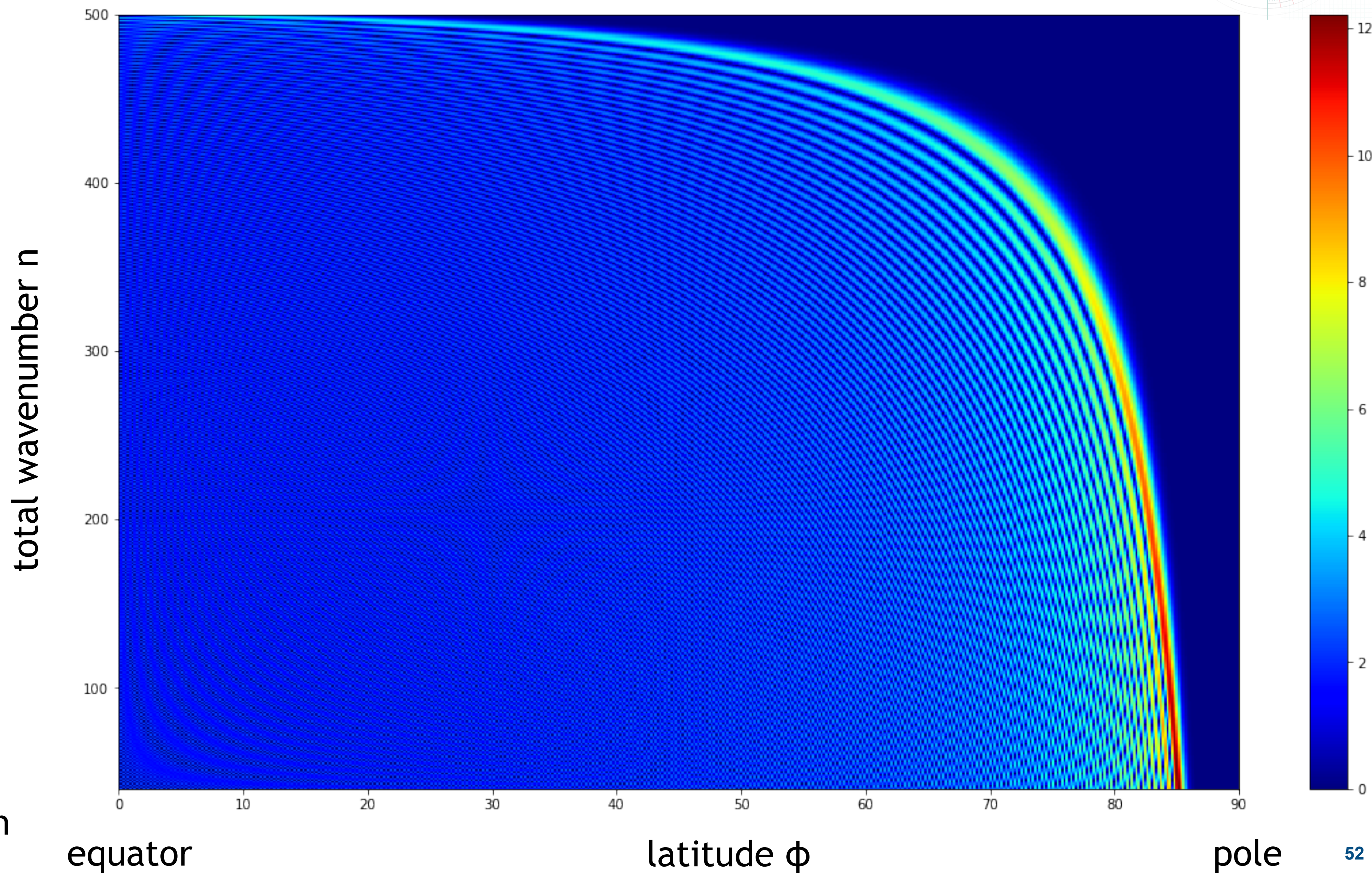
**FLT:**

**step 1:** split matrix  
into two rows

**step 2:** use  
interpolation to  
empty half of the  
columns

**step 3:** reorder  
columns

**step 4:** apply to each  
block recursively



# Fast Legendre Transform



matrix of Legendre polynomials

truncation  $N=500$ ,  
zonal wavenumber  
 $m=100$

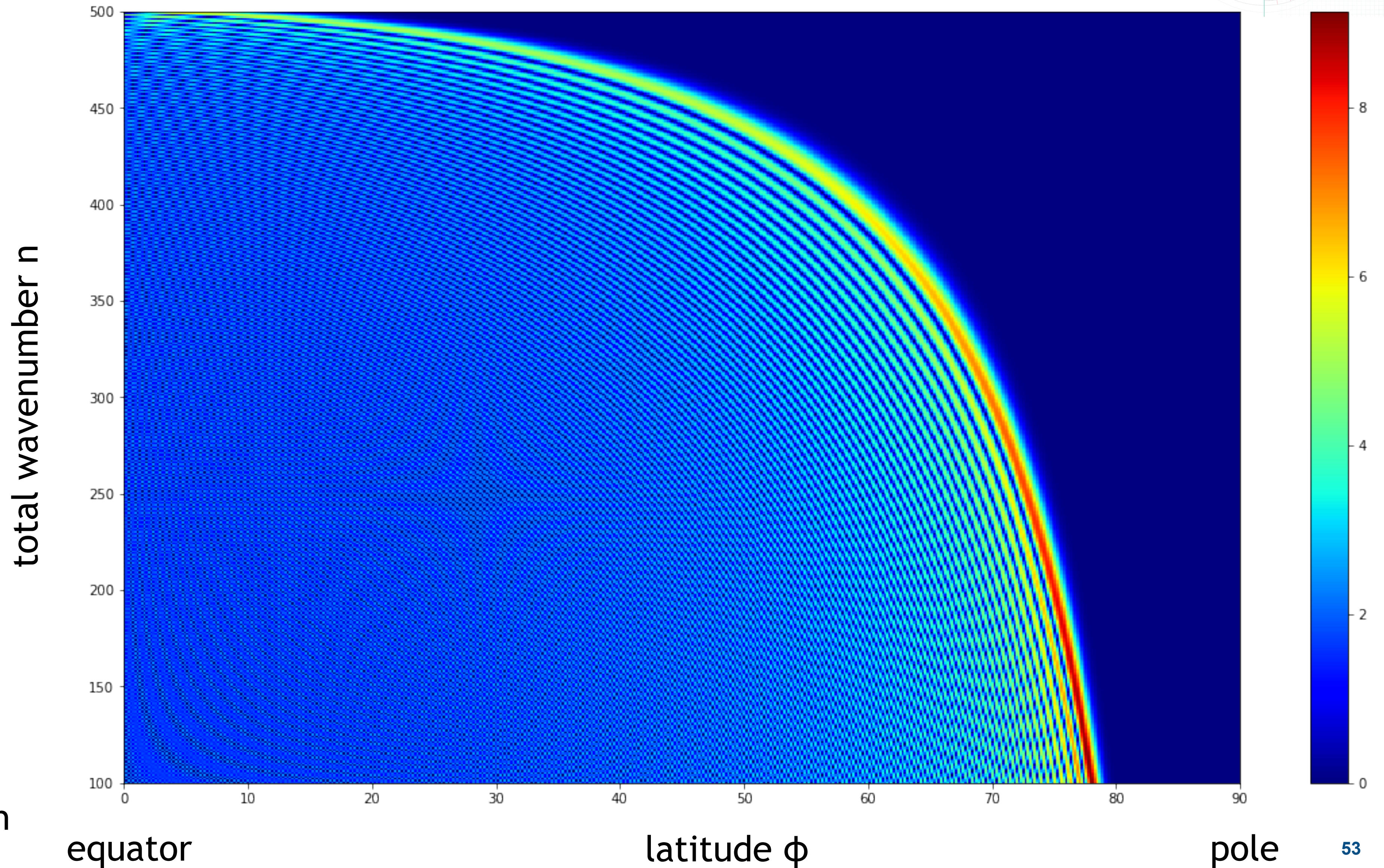
**FLT:**

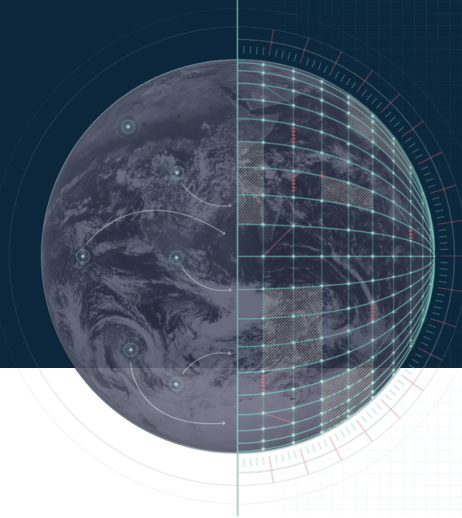
**step 1:** split matrix  
into two rows

**step 2:** use  
interpolation to  
empty half of the  
columns

**step 3:** reorder  
columns

**step 4:** apply to each  
block recursively

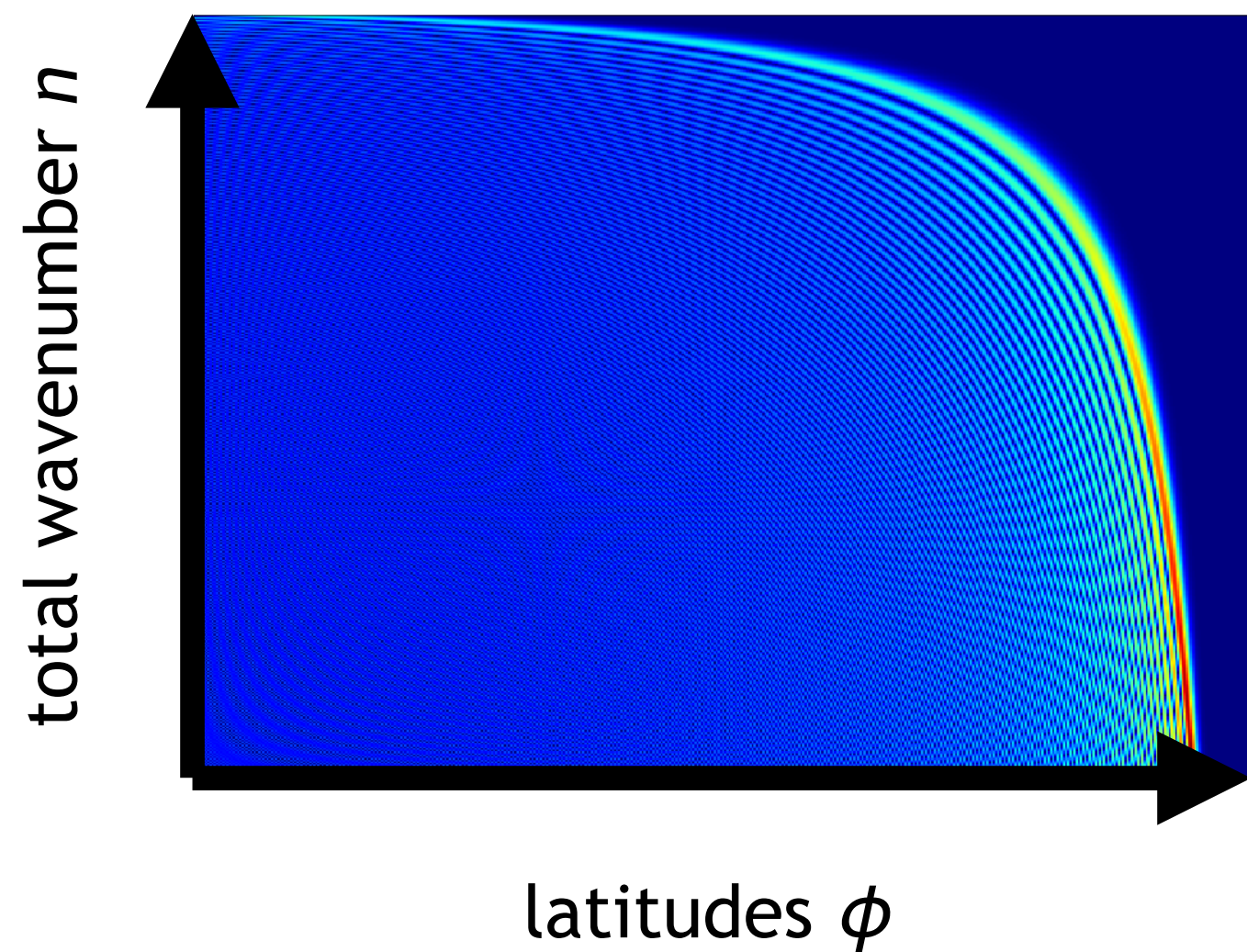




# Why should Fast Legendre Transform improve performance?

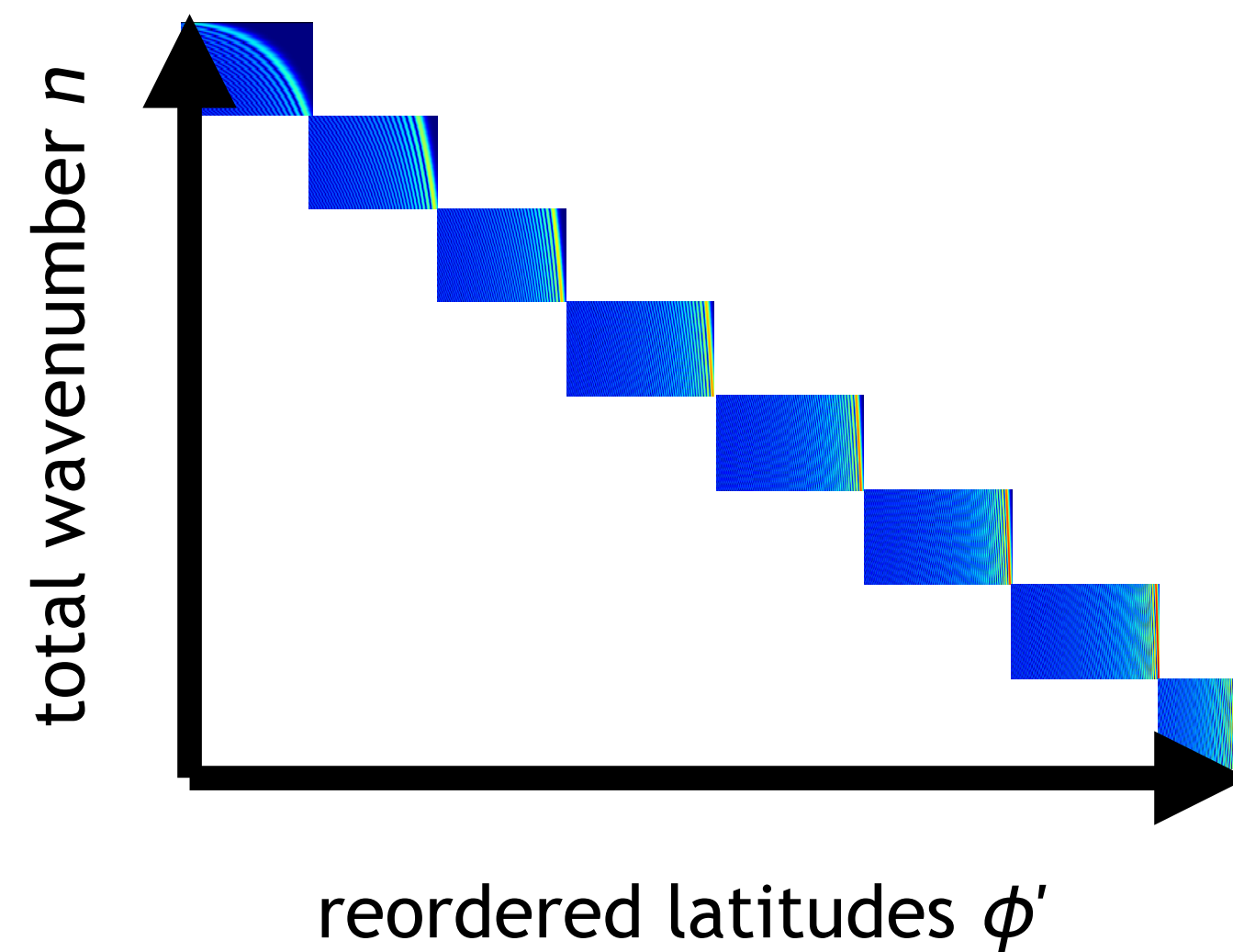
- recap:

Legendre polynomials  
(precomputed)



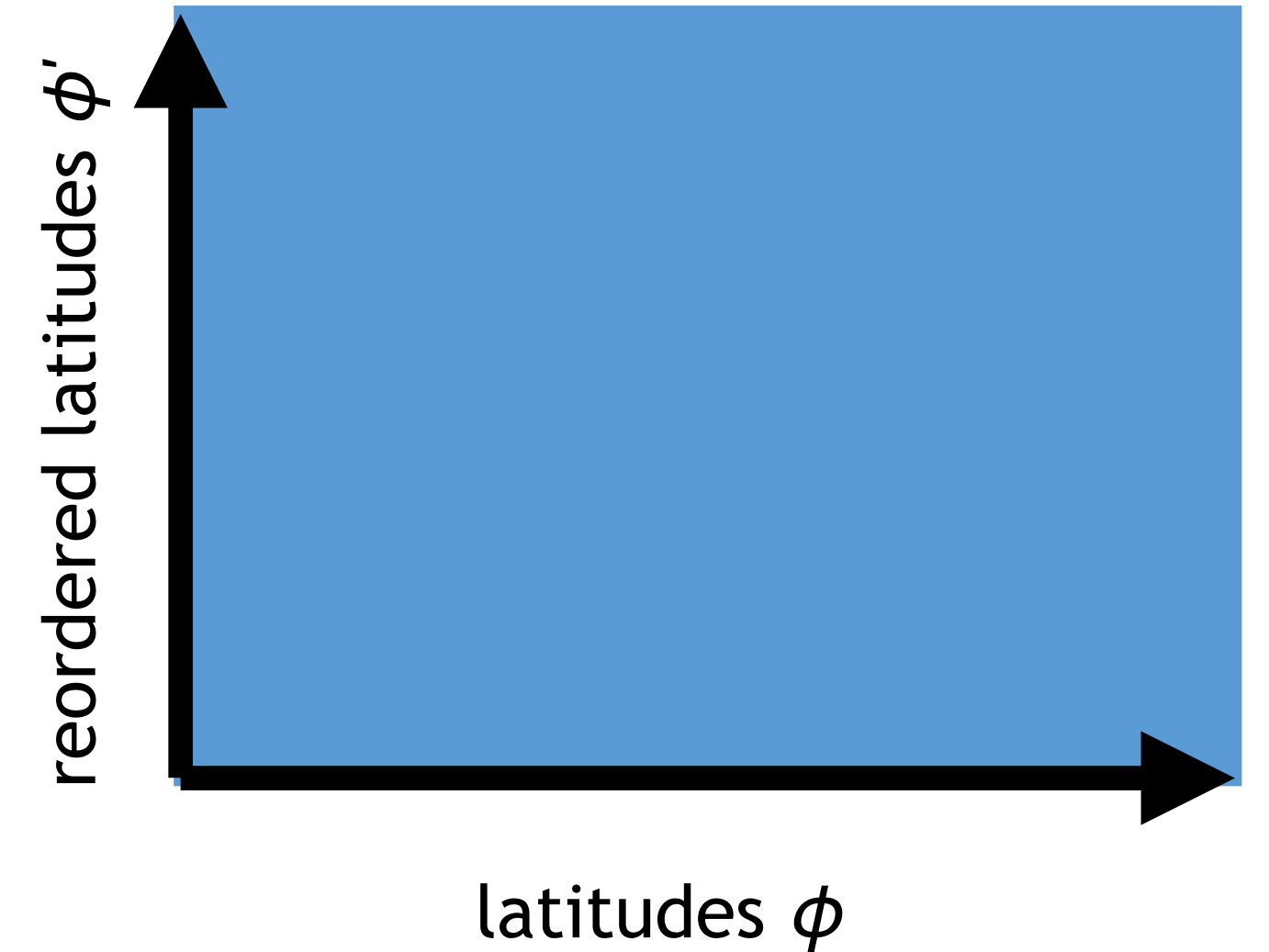
$\approx$

block diagonal matrix



$\cdot$

product of permutations and  
block diagonal interpolation matrices

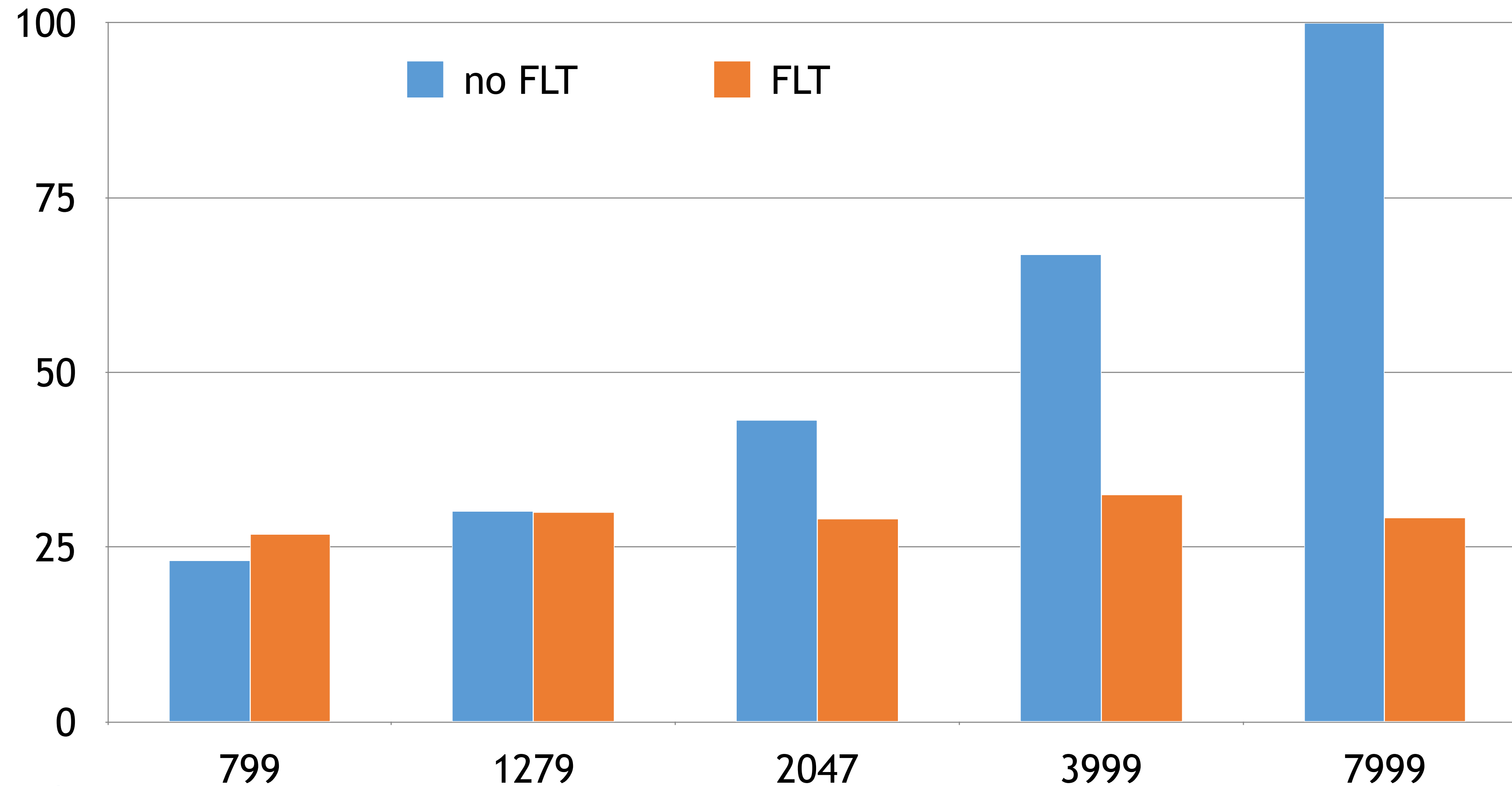


- these matrices are all fixed (independent of the current simulation!)
- only small blocks need to be multiplied with the current data, everything else is built into the algorithm

# Fast Legendre Transform floating point operations



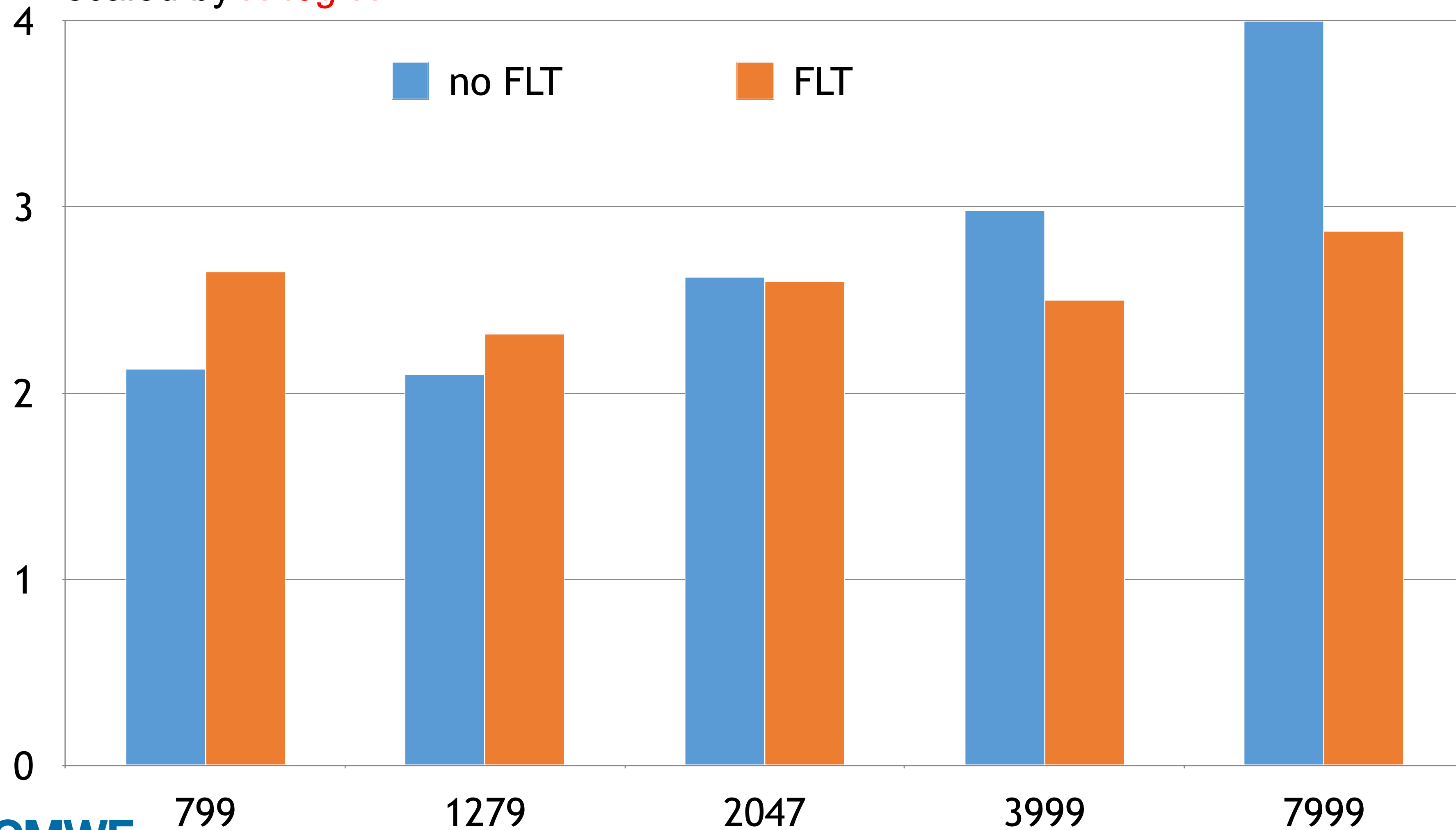
Number of floating point operations for direct or inverse spectral transforms of a single field, scaled by  $N^2 \log^3 N$



# Fast Legendre Transform wallclock time



Average wall-clock time compute cost of  $10^7$  spectral transforms  
scaled by  $N^2 \log^3 N$





# Questions?