ECMWF – DESTINATION EARTH

SPECTRAL TRANSFORM

Andreas Müller, ECMWF

Funded by the European Union
Destination Earth implemented by ECMWF esa EUMETSAT

Overview

10 minutes	• Fourier transform • Spectral transform
60 minutes 30 minutes	• hands-on exercises with Python • 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

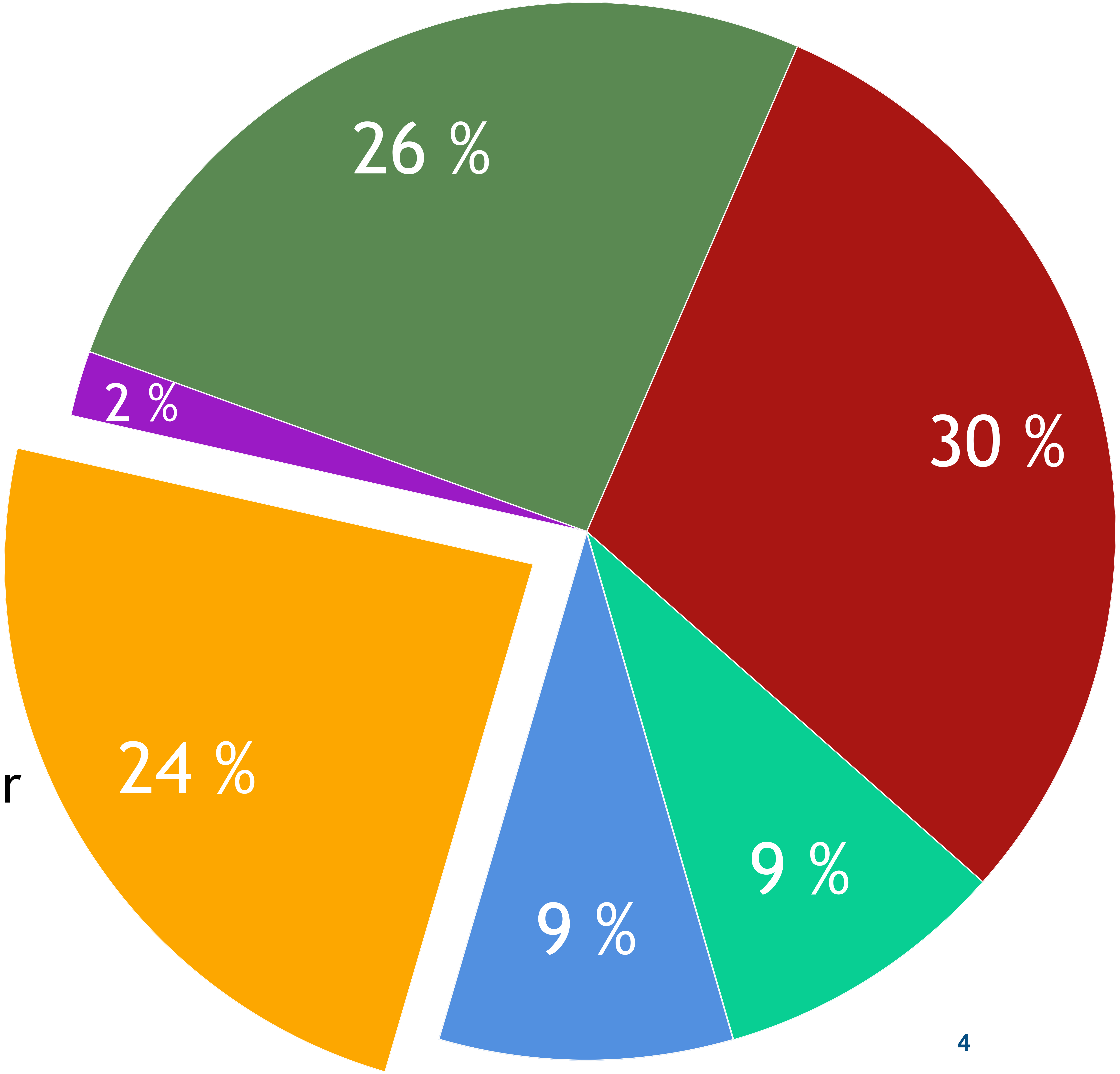
- spectral transform
- semi-Lagrangian
- semi-implicit

IFS (Integrated Forecast System)

technology applied at ECMWF for the last 30 years

- spectral transform
- semi-Lagrangian
- semi-implicit

pie chart: % of runtime in 9km operational forecast



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

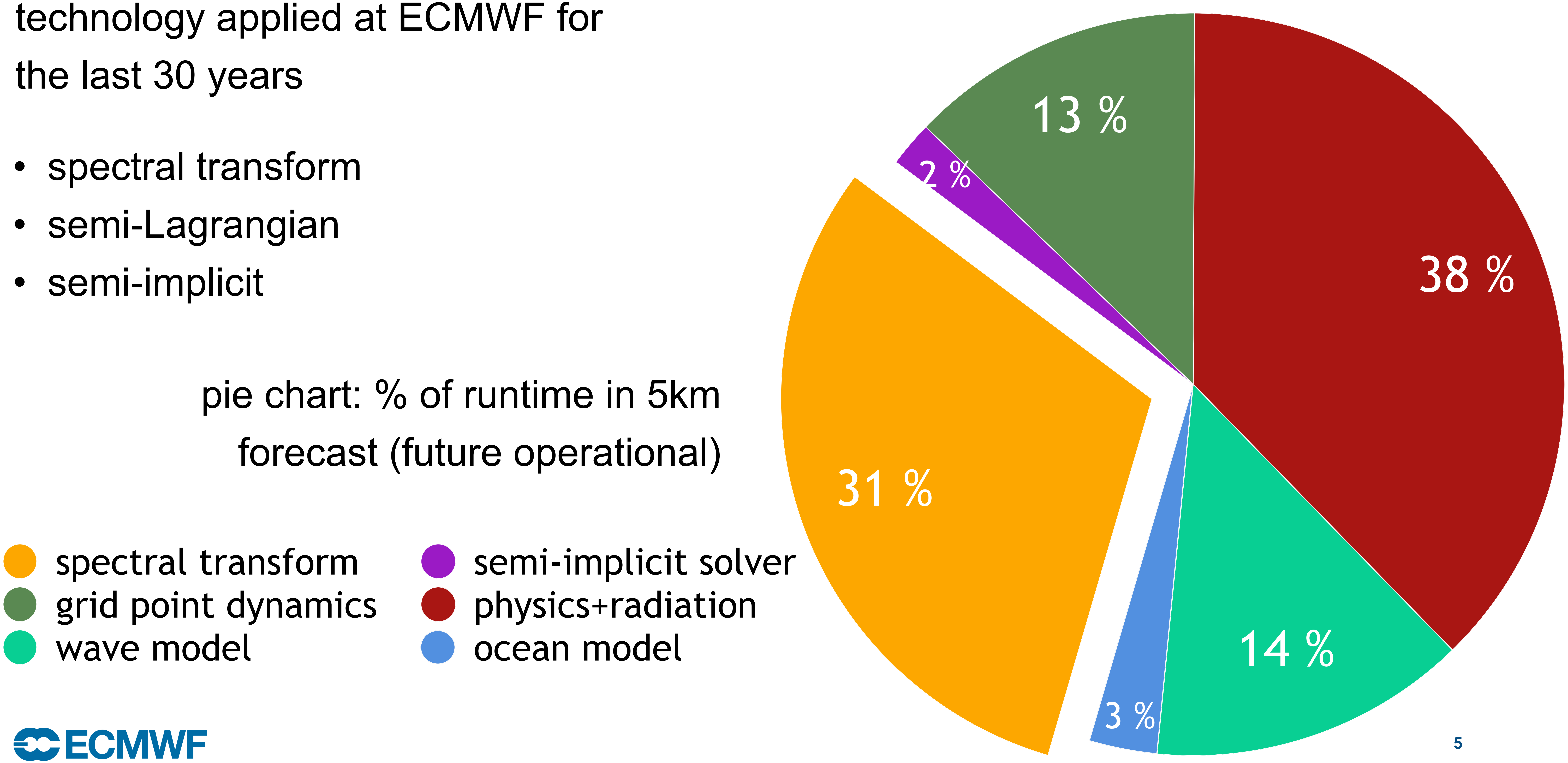
ECMWF

IFS (Integrated Forecast System)

technology applied at ECMWF for the last 30 years

- spectral transform
- semi-Lagrangian
- semi-implicit

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



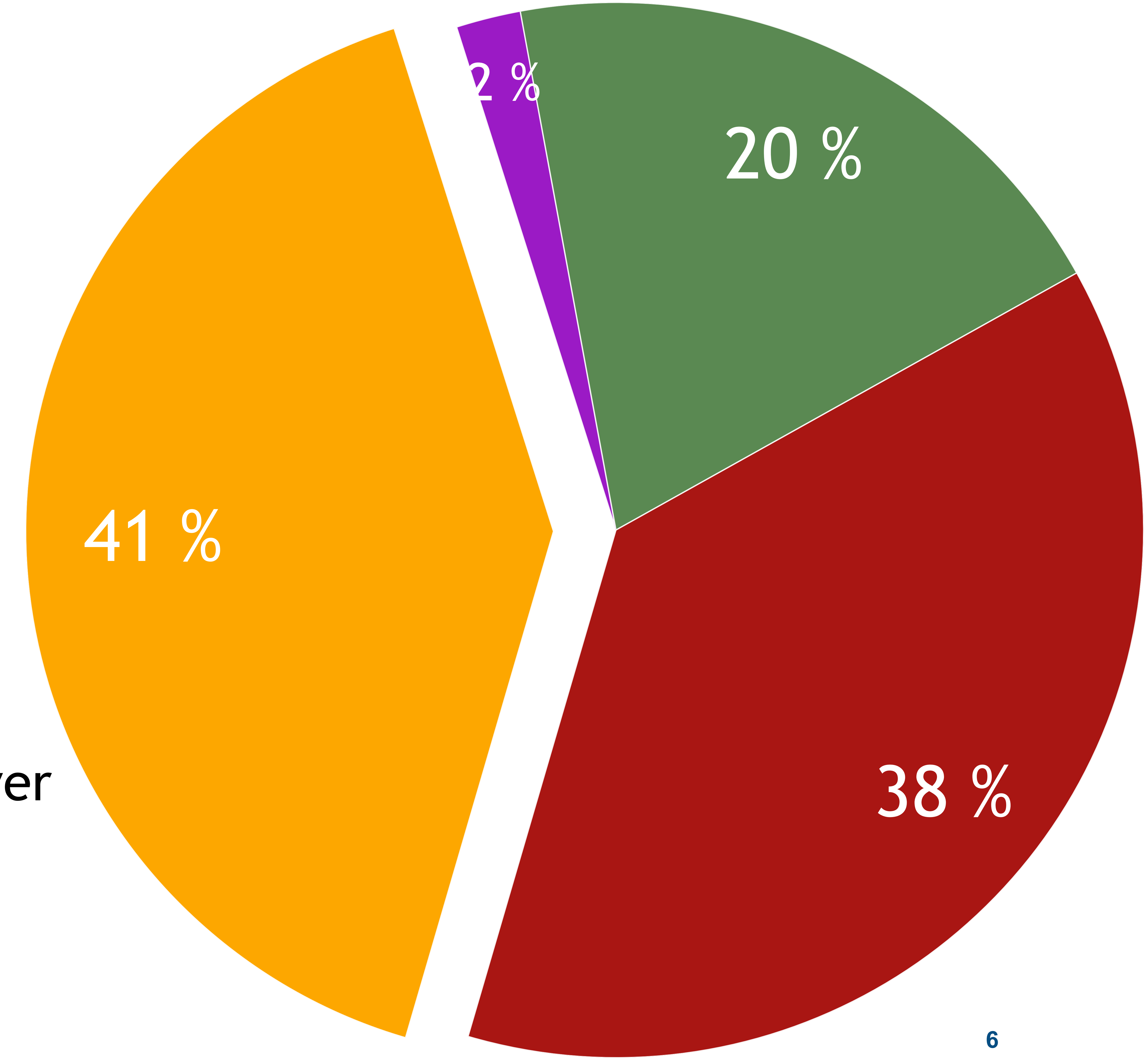
ECMWF

IFS (Integrated Forecast System)

technology applied at ECMWF for the last 30 years

- spectral transform
- semi-Lagrangian
- semi-implicit

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

ECMWF

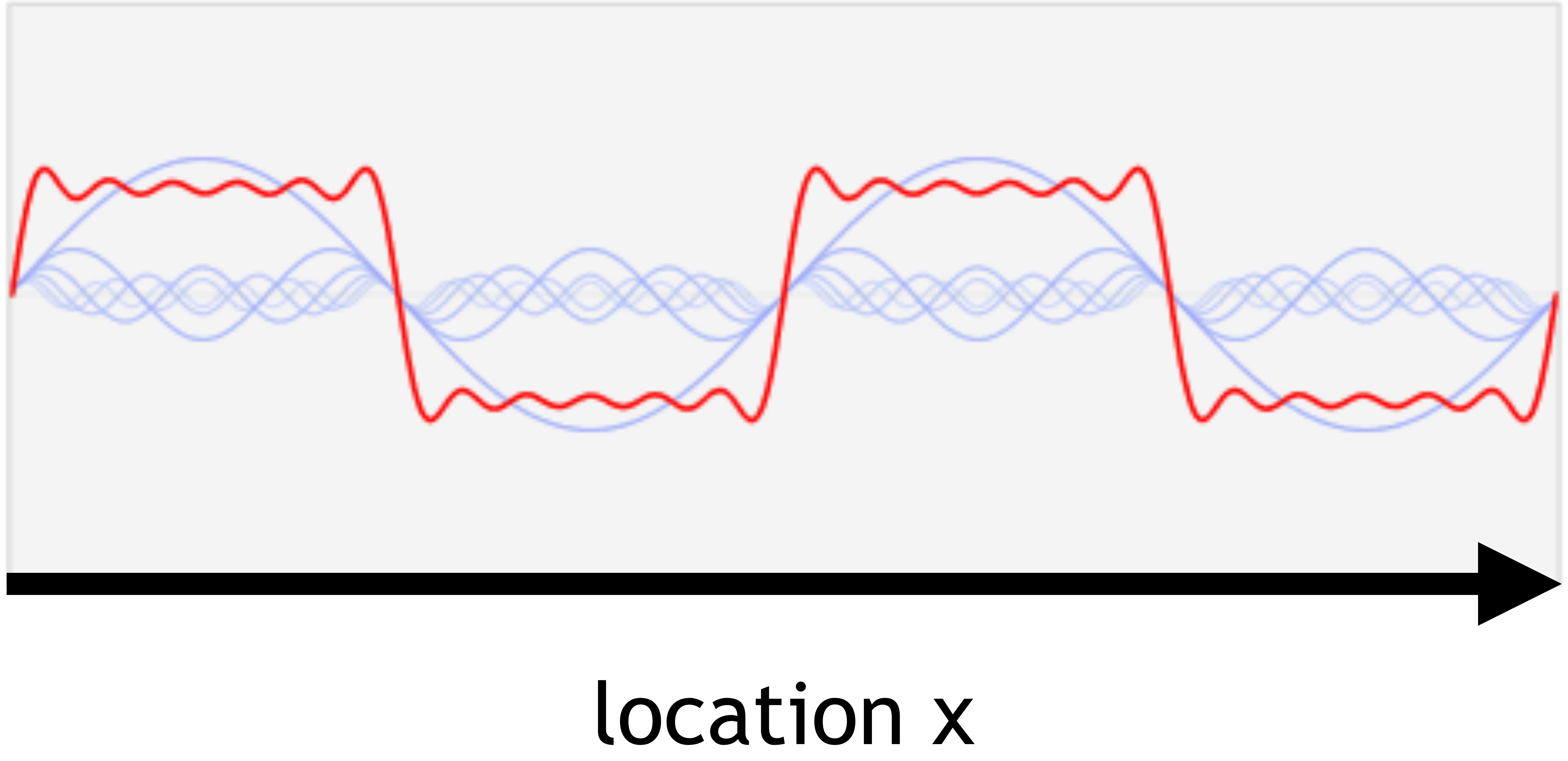
Fourier transform

Fourier transform = Spectral transform in 1D

location x

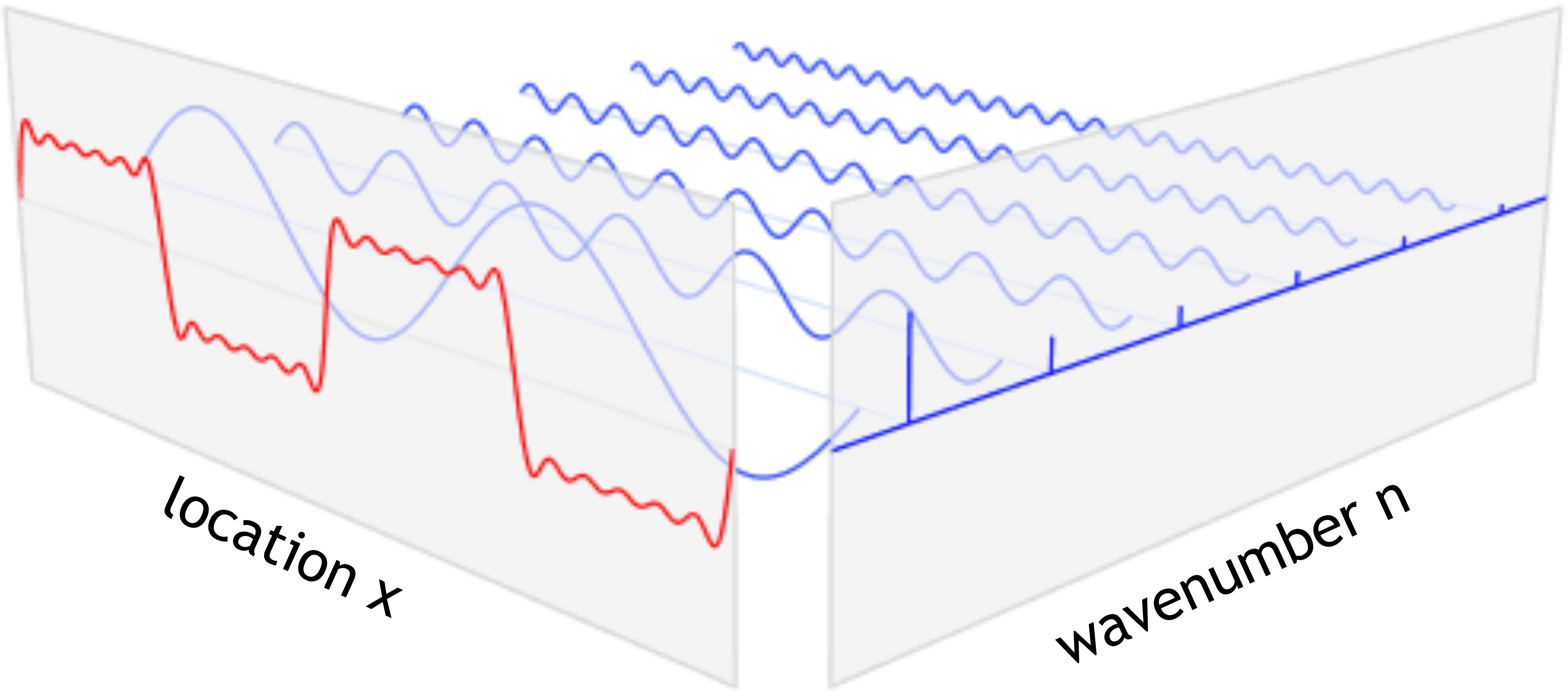
Fourier transform

Fourier transform = Spectral transform in 1D



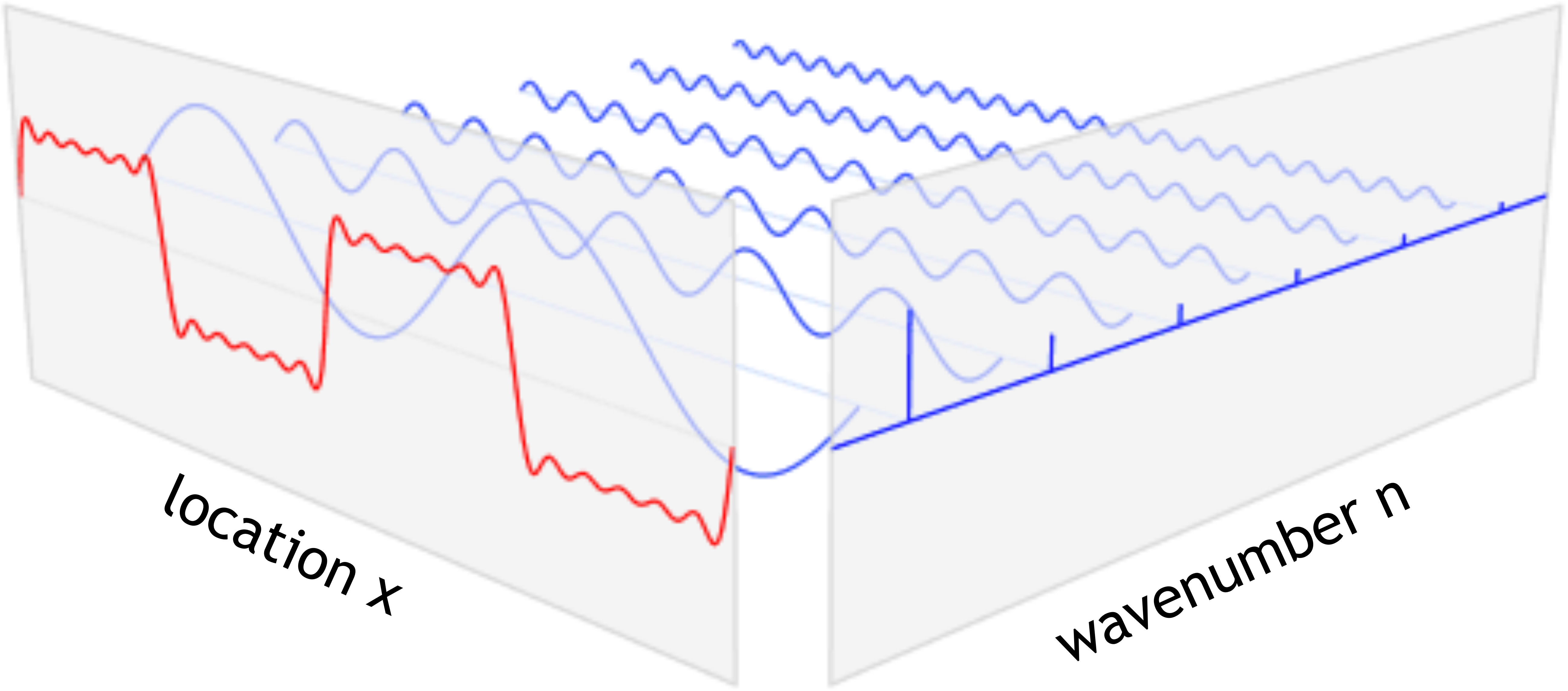
Fourier transform

Fourier transform = Spectral transform in 1D



grid point space **Fourier space**

Fourier transform



function in grid point space

Fourier coefficients

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

Fourier transform

function in grid point space

Fourier coefficients

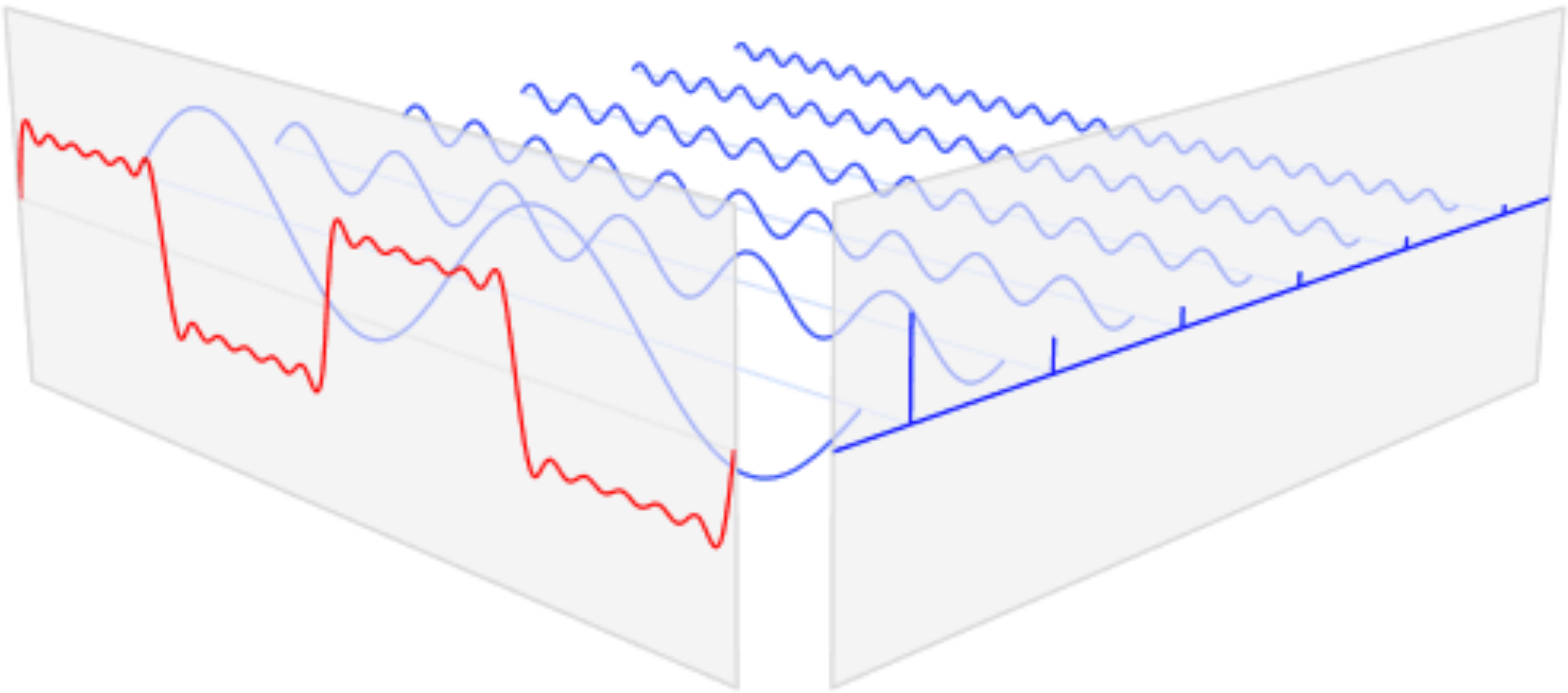
$$f(x) = \sum_n f_n \cdot \mathrm{e}^{-2\pi \mathrm{i}\, n\, x}$$

differentiation

simple multiplication

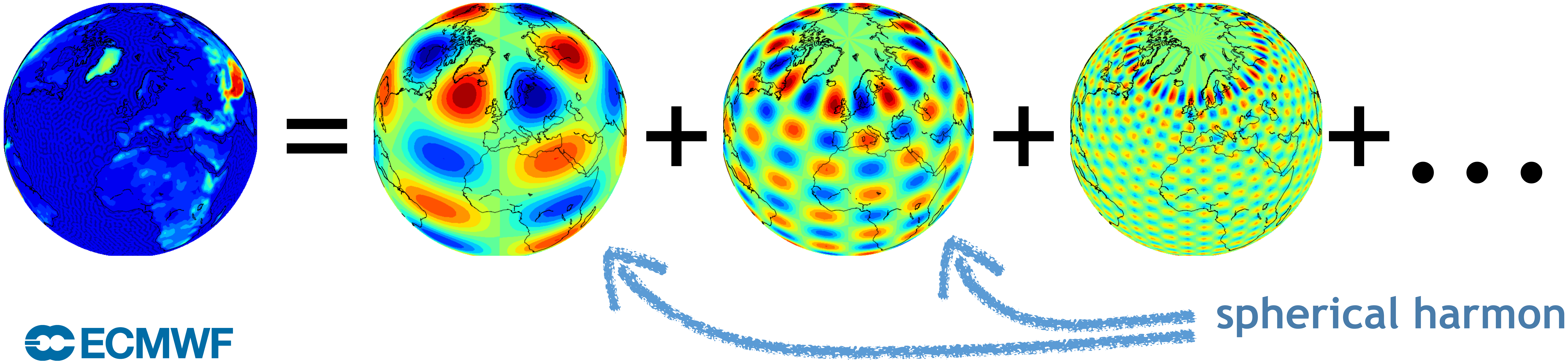
$$\frac{\mathrm{d}f(x)}{\mathrm{dx}} = \sum_n (-2\pi \mathrm{i}\, n\, f_n) \cdot \mathrm{e}^{-2\pi \mathrm{i}\, n\, x}$$

on the sphere: spectral transform



grid point space

spectral space



spherical harmonics

on the sphere: spectral transform

Spectral coefficients

Spherical harmonics

Latitude

Longitude

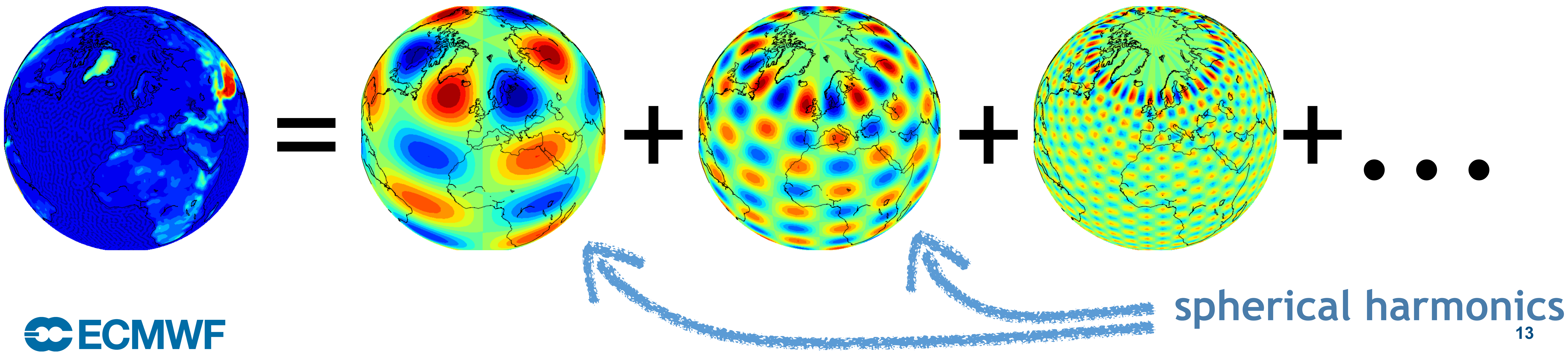
Grid point variable

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

m: zonal wavenumber
n: total wavenumber
M: truncation

grid point space

spectral space



ECMWF

on the sphere: spectral transform

Spectral coefficients

Latitude

Longitude

Spherical harmonics

Grid point variable

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

m: zonal wavenumber
n: total wavenumber
M: truncation

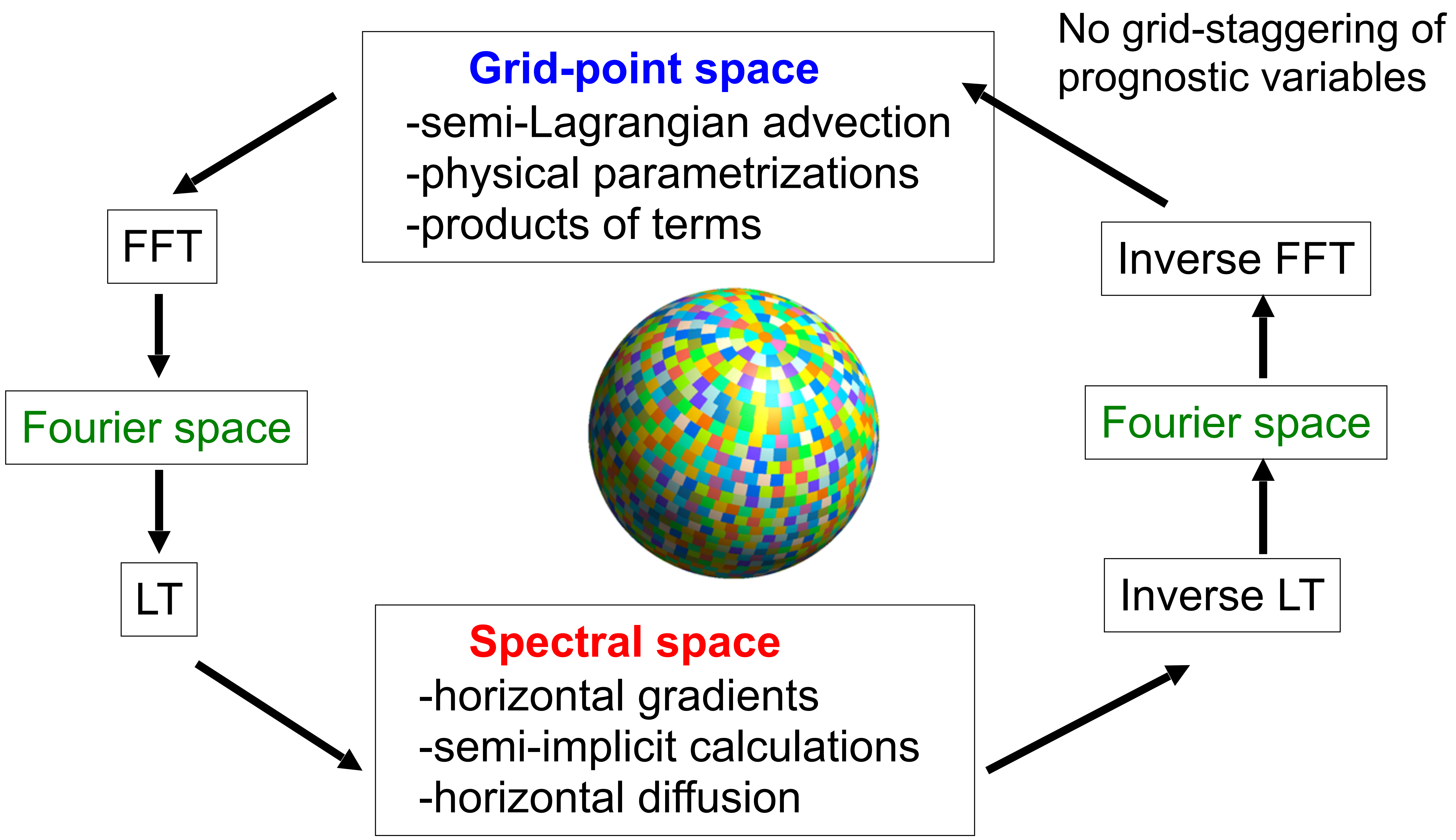
Legendre polynomials

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

Legendre transform

Fourier transform

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

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

aliasing

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

aliasing

wave generated in spectral space

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

aliasing

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

ECMWF

aliasing

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

aliasing

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

ECMWF

aliasing

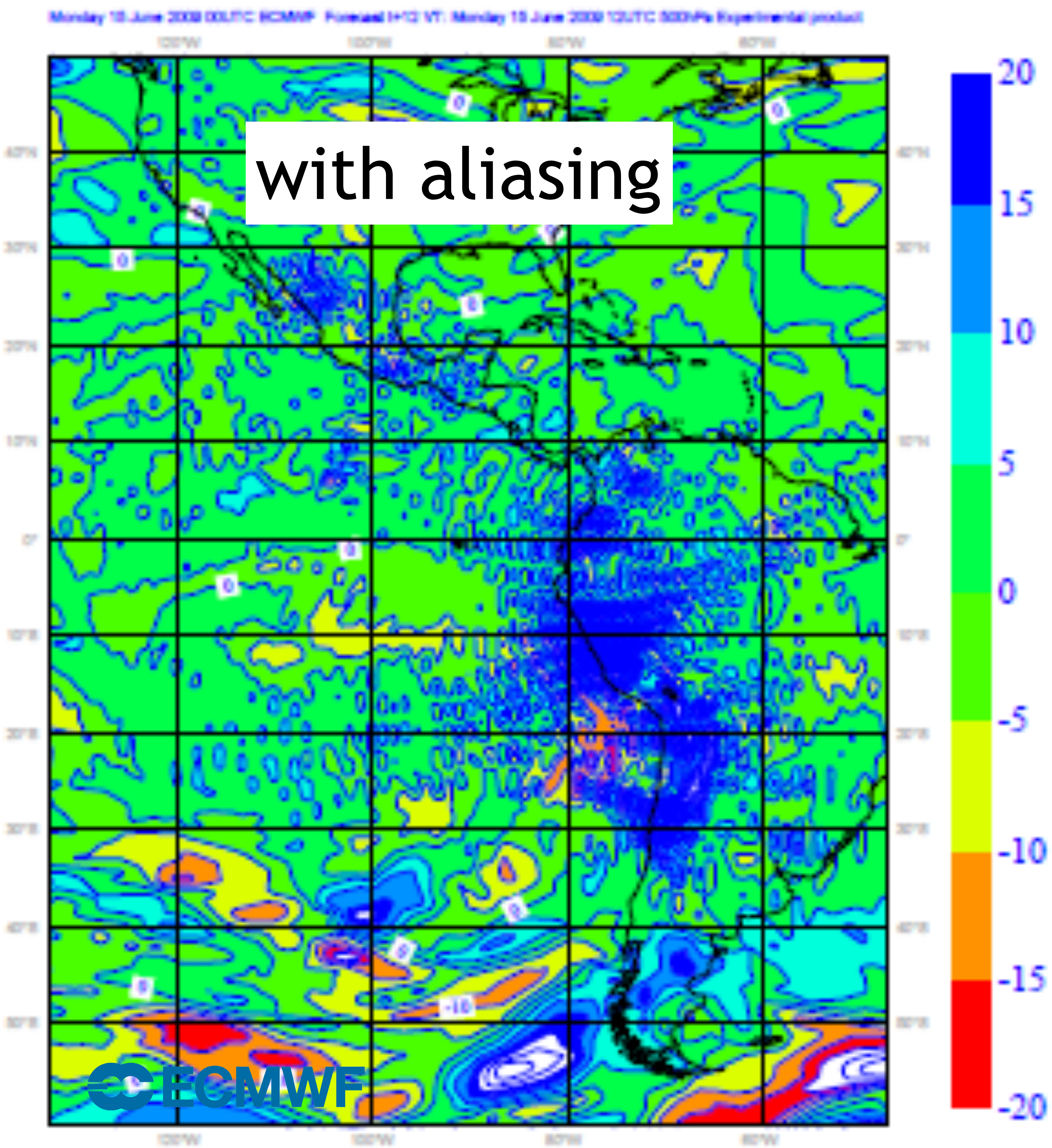
Issue: multiplication of two variables produces shorter waves than grid can handle

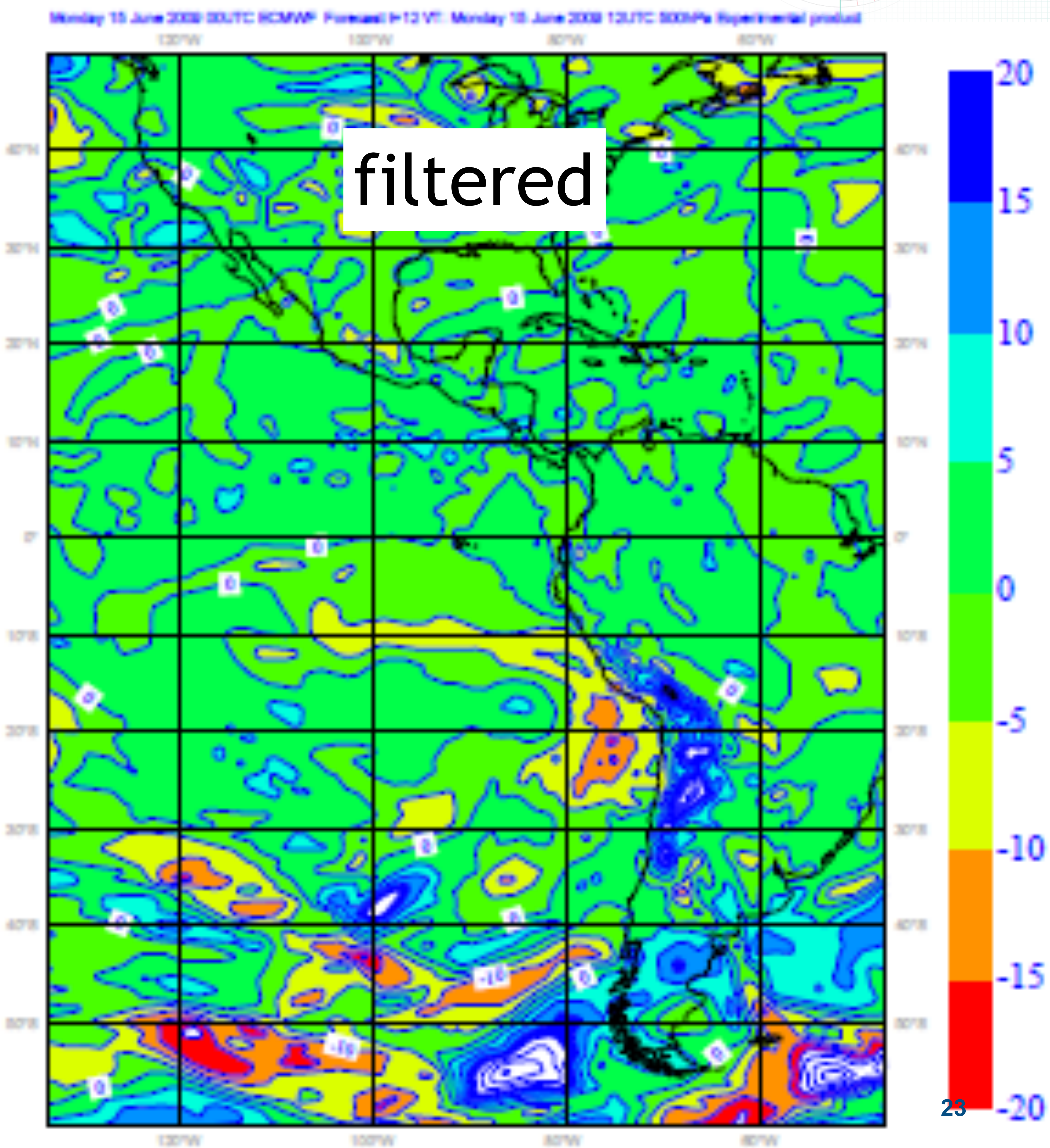
grid can handle $\lambda \geq 2\ \Delta x$

ECMWF

aliasing example

500hPa adiabatic zonal wind tendencies (T159)

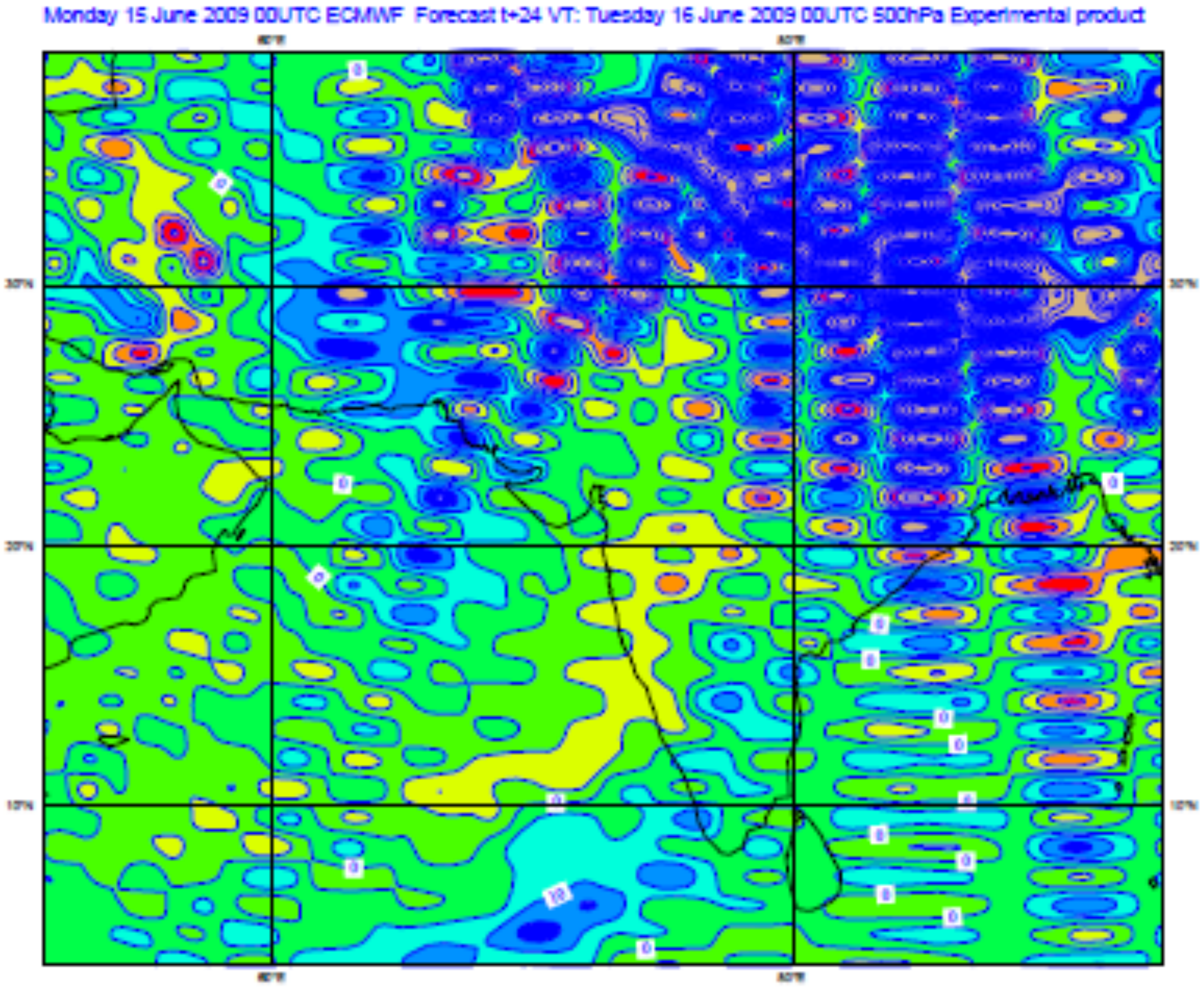




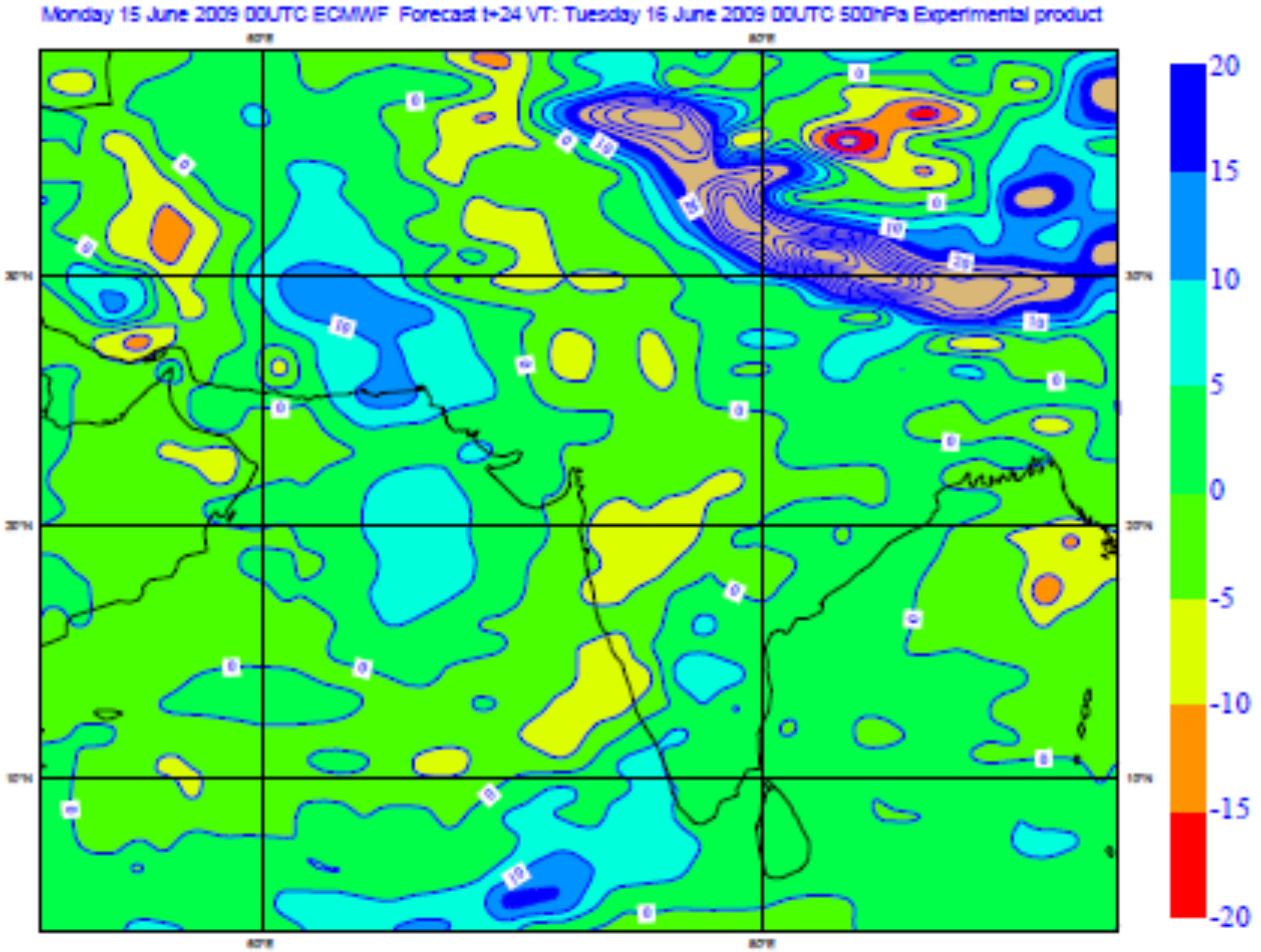
aliasing example

500hPa adiabatic meridional wind tendencies (T159)

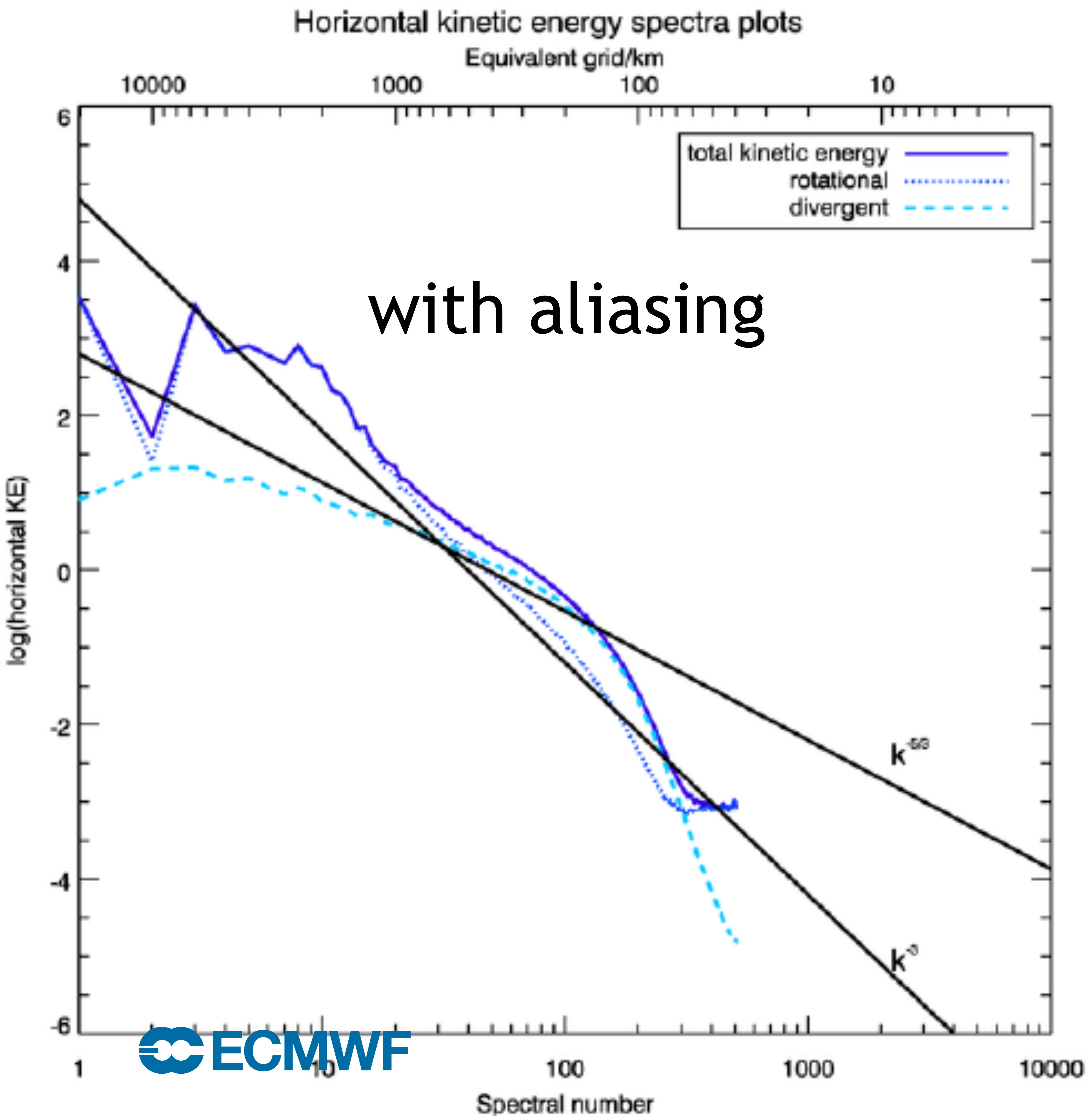
with aliasing

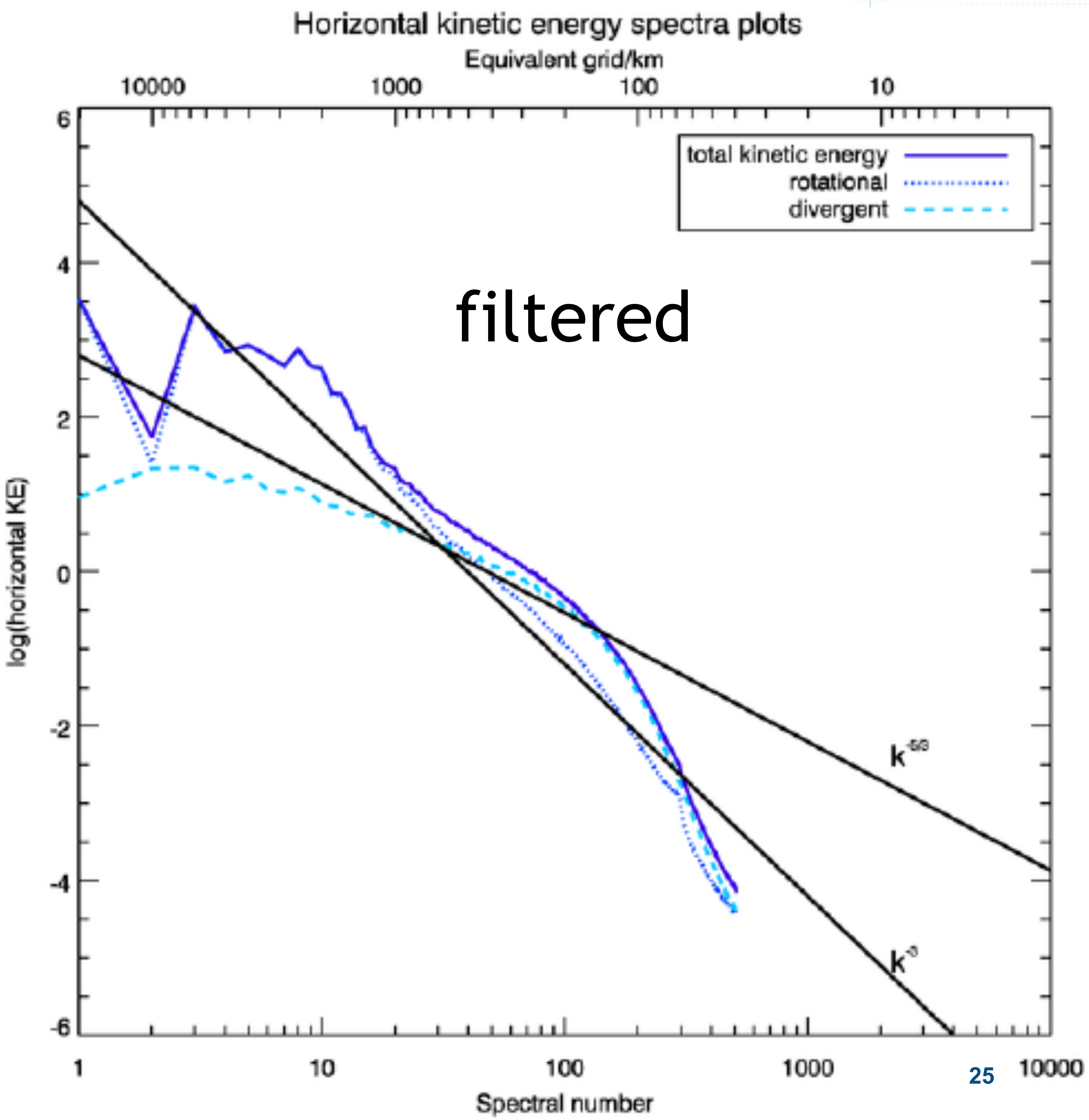


filtered



aliasing example
kinetic energy spectra, 100 hPa





ECMWF

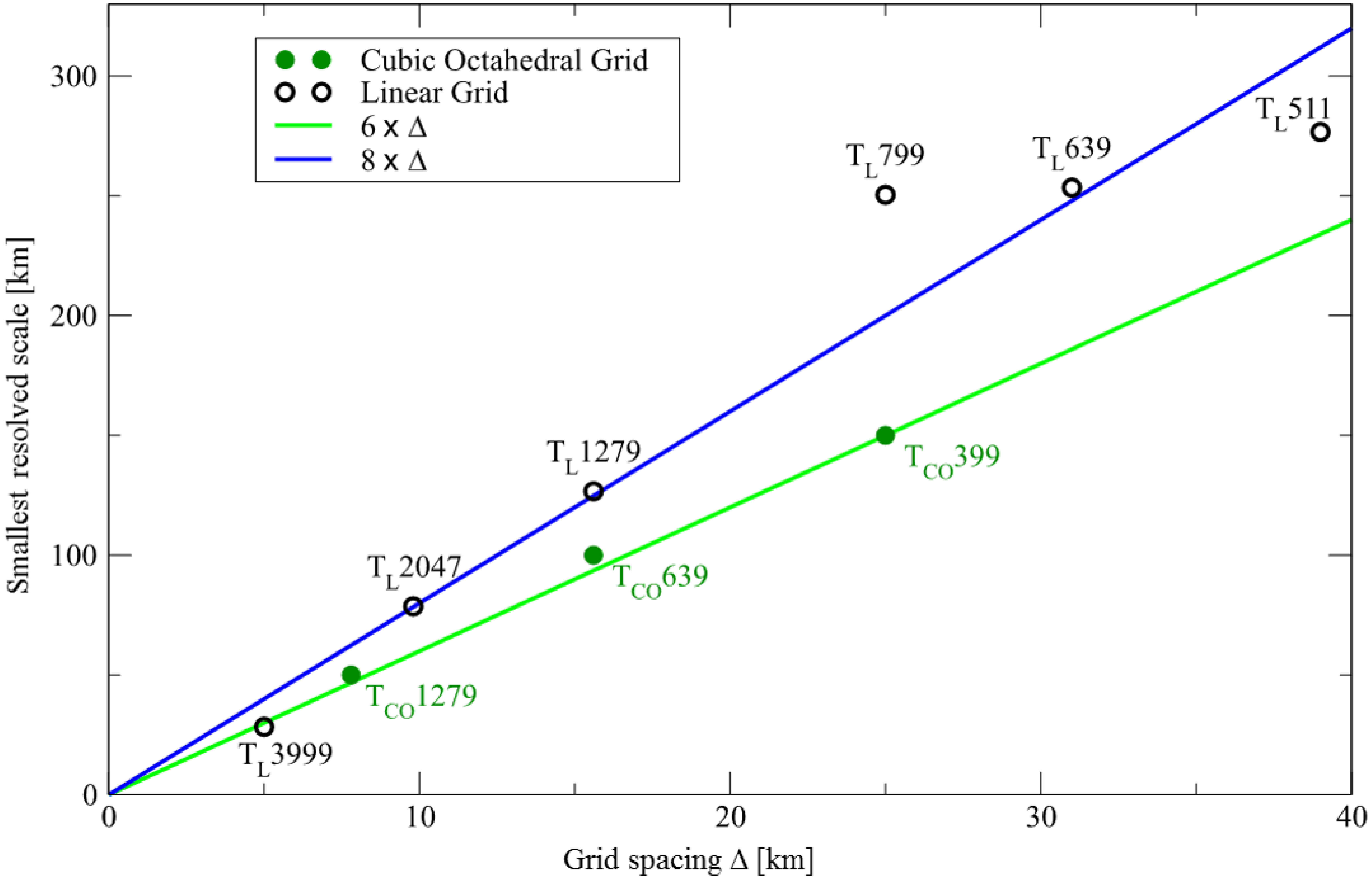
alternatives to using a filter

Idea: use more grid points than spectral coefficients

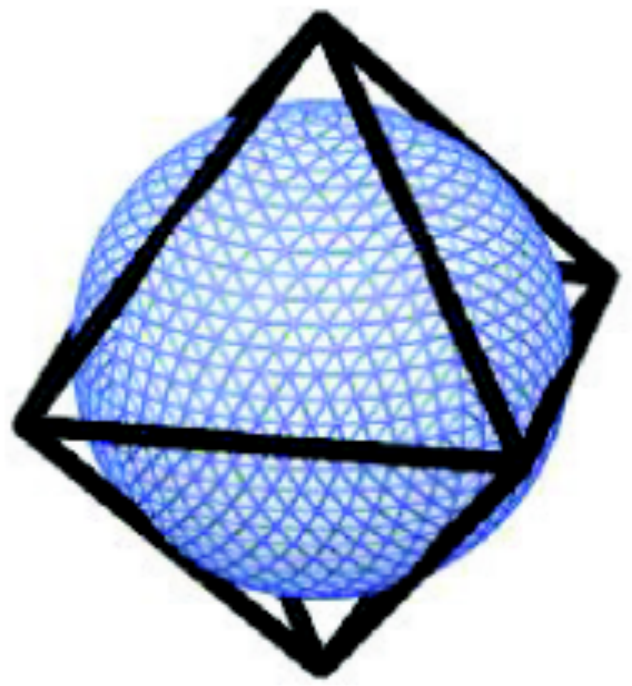
Orszag, 1971:

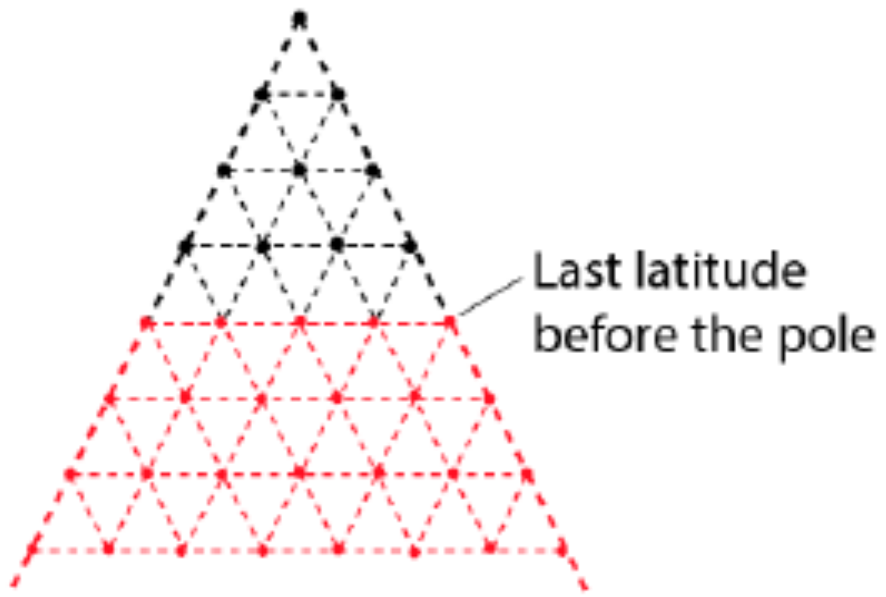
2N+1 gridpoints to N waves : linear grid	~ 1-2 Δ
3N+1 gridpoints to N waves : quadratic grid	~ 2-3 Δ
4N+1 gridpoints to N waves : cubic grid	~ 3-4 Δ *(Wedi, 2014)*
	Spatial filter range

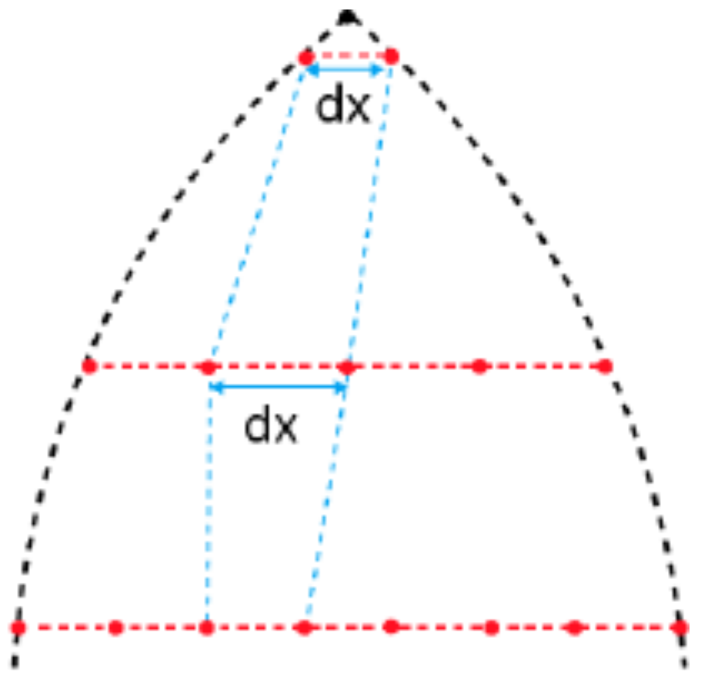
effective resolution
of linear and cubic grids (Abdalla et al. 2013)



Cubic octahedral (Gaussian) grid of IFS



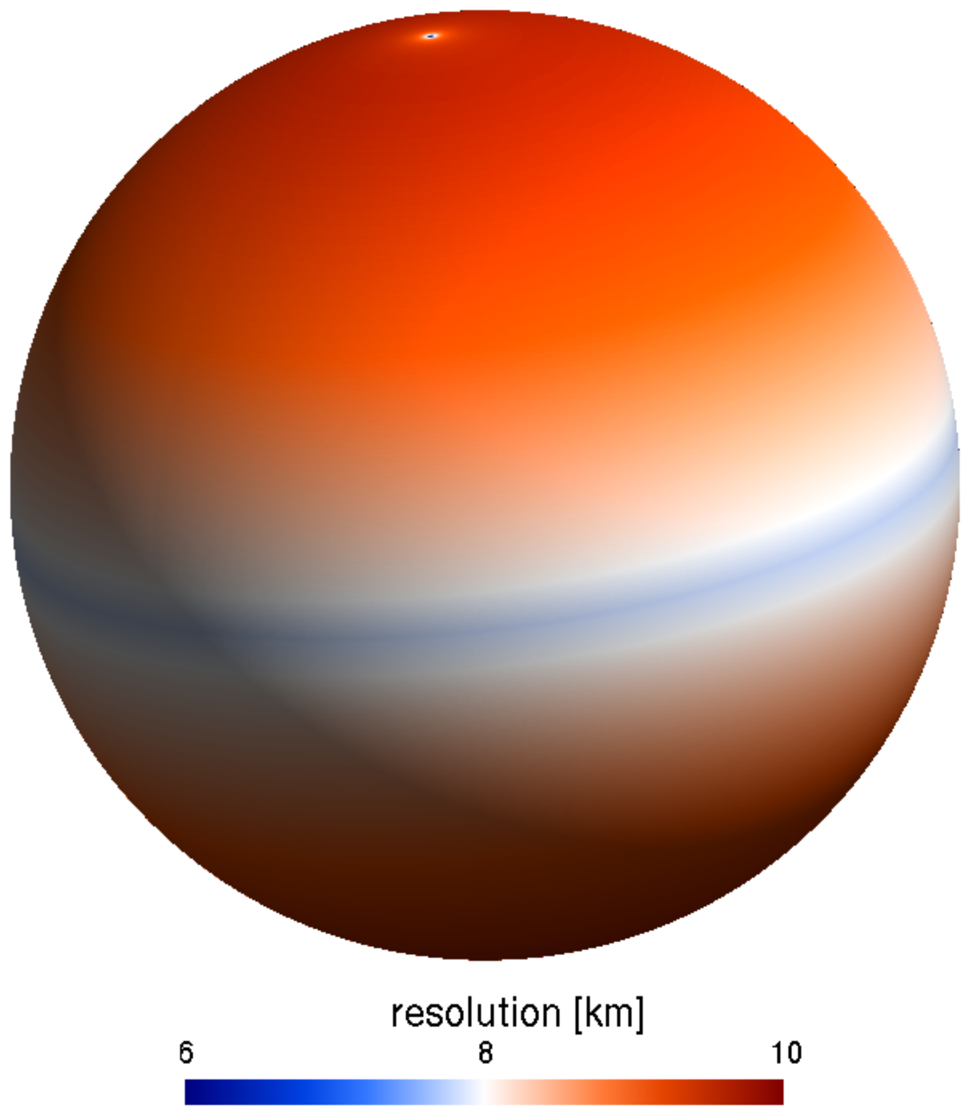




- No aliasing in nonlinear products
- Improved accuracy and mass conservation compared with linear grid
- Efficiency and scalability for large size problems: high grid-point resolution for a given spectral truncation i.e. expensive transforms become a smaller fraction of total computations

Collignon projection on the sphere: Number of points at latitude line i = 4 × i + 16, i = 1, . . . ,2N

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

ECMWF

inverse spectral transform

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

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

fields (variables, height levels)

real and imaginary part

wave numbers
m=0,…,N;   n=0,…,N-m
(N: truncation)

ECMWF

inverse spectral transform

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

m=0,…,N; n=0,…,N-m

even n

odd n

for each m:

$\mathbf{D}_{e,m}(f, \mathrm{i}, n)$

$\mathbf{D}_{o,m}(f, \mathrm{i}, n)$

inverse spectral transform

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

m=0,…,N; n=0,…,N-m

P: precomputed Legendre polynomials

even n

odd n

for each m:

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

matrix multiplications

inverse spectral transform

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

m=0,…,N; n=0,…,N-m

P: precomputed Legendre polynomials

even n

odd n

for each m:

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

matrix multiplications

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

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

inverse spectral transform

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

m=0,…,N; n=0,…,N-m

P: precomputed Legendre polynomials

even n

odd n

for each m:

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

matrix multiplications

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

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

for each ϕ,f:

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

FFT: Fast Fourier Transform

ECMWF

inverse spectral transform

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

m=0,…,N; n=0,…,N-m

P: precomputed Legendre polynomials

even n

odd n

for each m:

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

matrix multiplications

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

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

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

FFT: Fast Fourier Transform

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

inverse spectral transform

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

spectral space

even n

odd n

for each m:

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

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

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

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

inverse Legendre transform

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

inverse Fourier transform

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

grid point space

inverse spectral transform

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

even n — odd n

for each m:

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

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

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

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

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

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

spectral space	m,n
inverse Legendre transform	m,f
inverse Fourier transform	ϕ,f
grid point space	ϕ,λ

parallelisation over these indices

inverse spectral transform

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

even n — odd n

for each m:

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

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

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

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

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

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

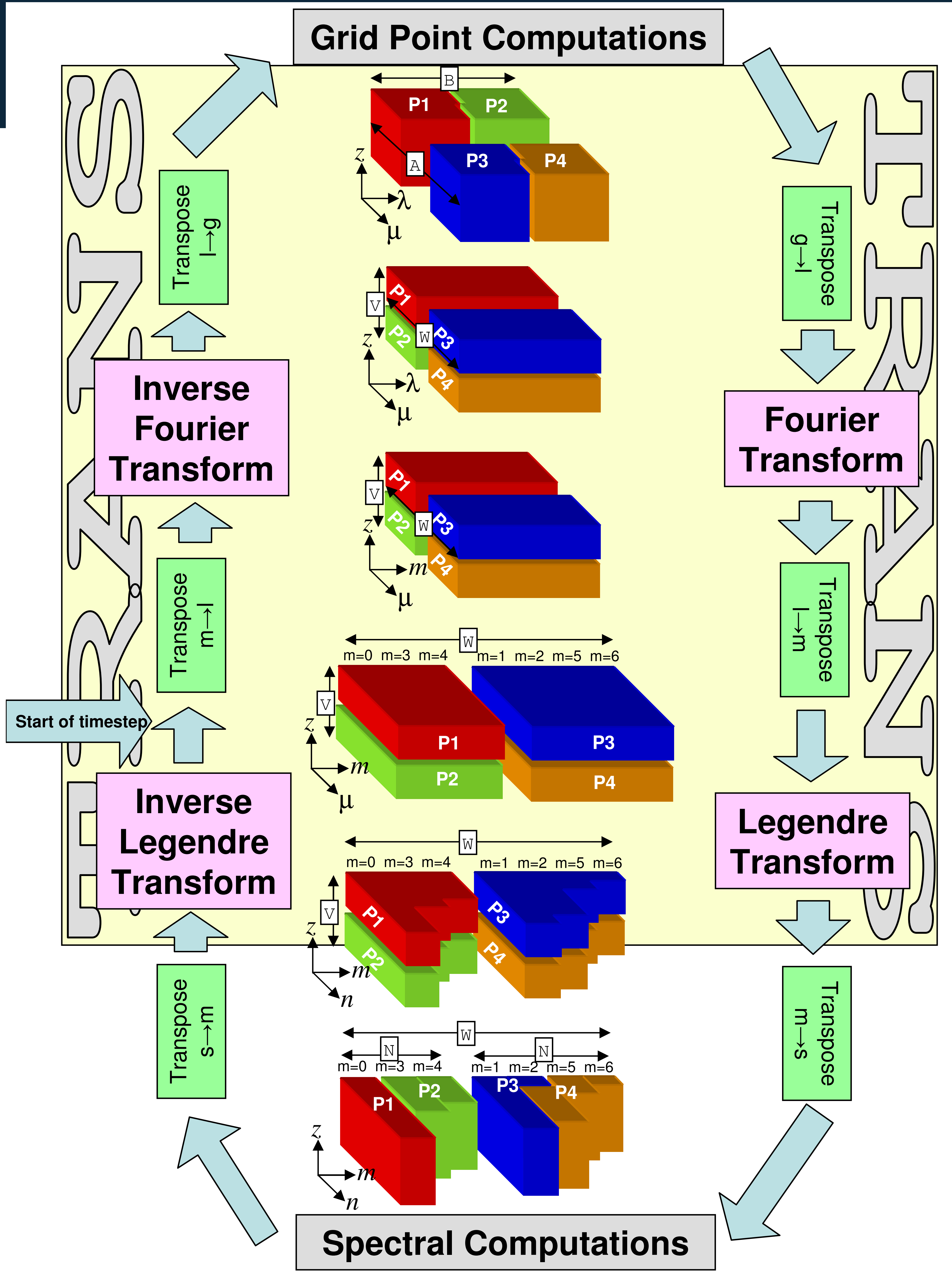
spectral space	m,n
inverse Legendre transform	m,f
inverse Fourier transform	ϕ,f
grid point space	ϕ,λ

parallelisation over these indices

lots of MPI communication

direct spectral transform

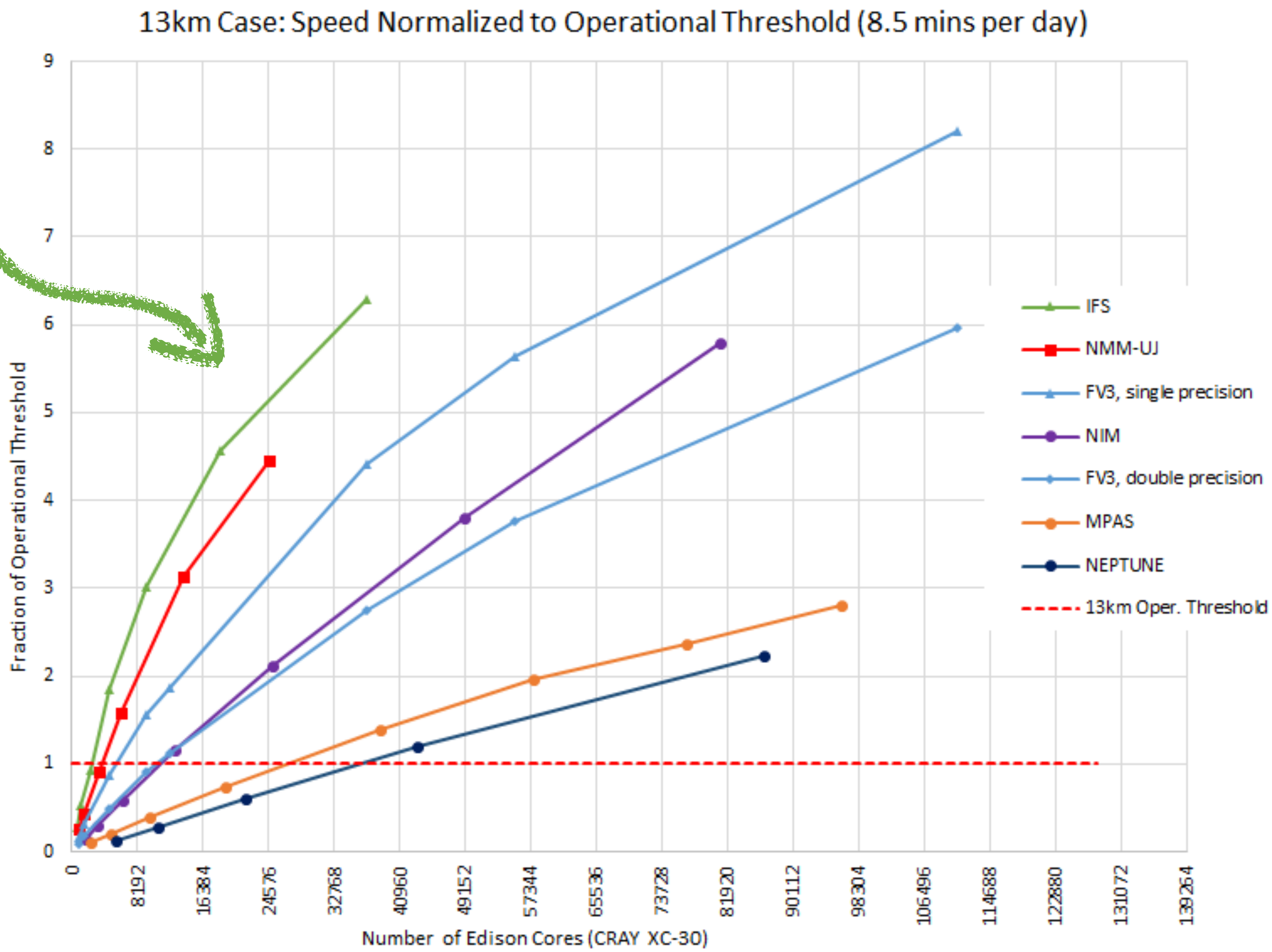
- same like inverse spectral transform
- reverse order
- multiply data with Gaussian quadrature weights before Legendre transform



ECMWF

performance comparison of IFS with other models

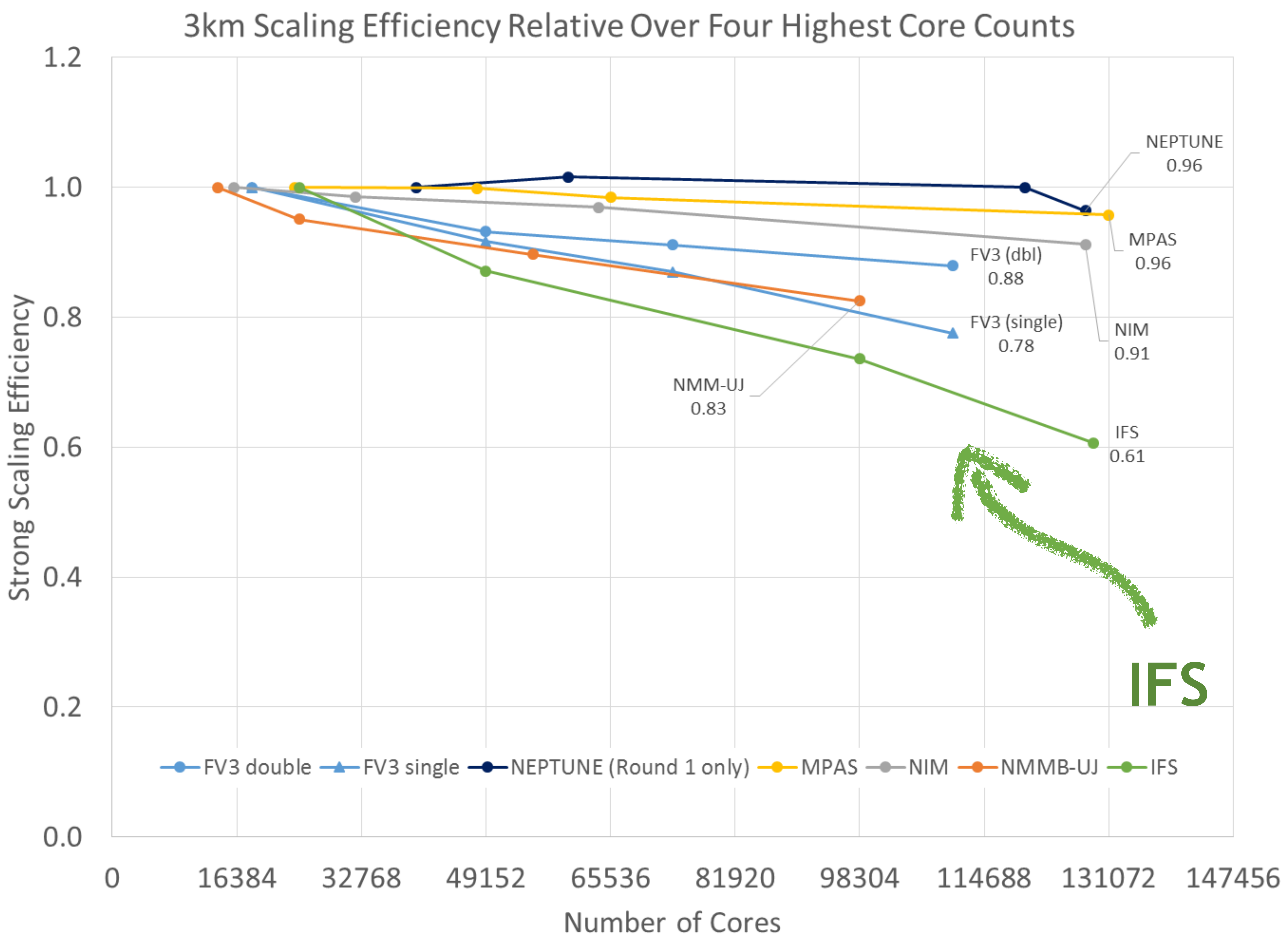
IFS



(Michalakes et al, NGGPS AVEC report, 2015)

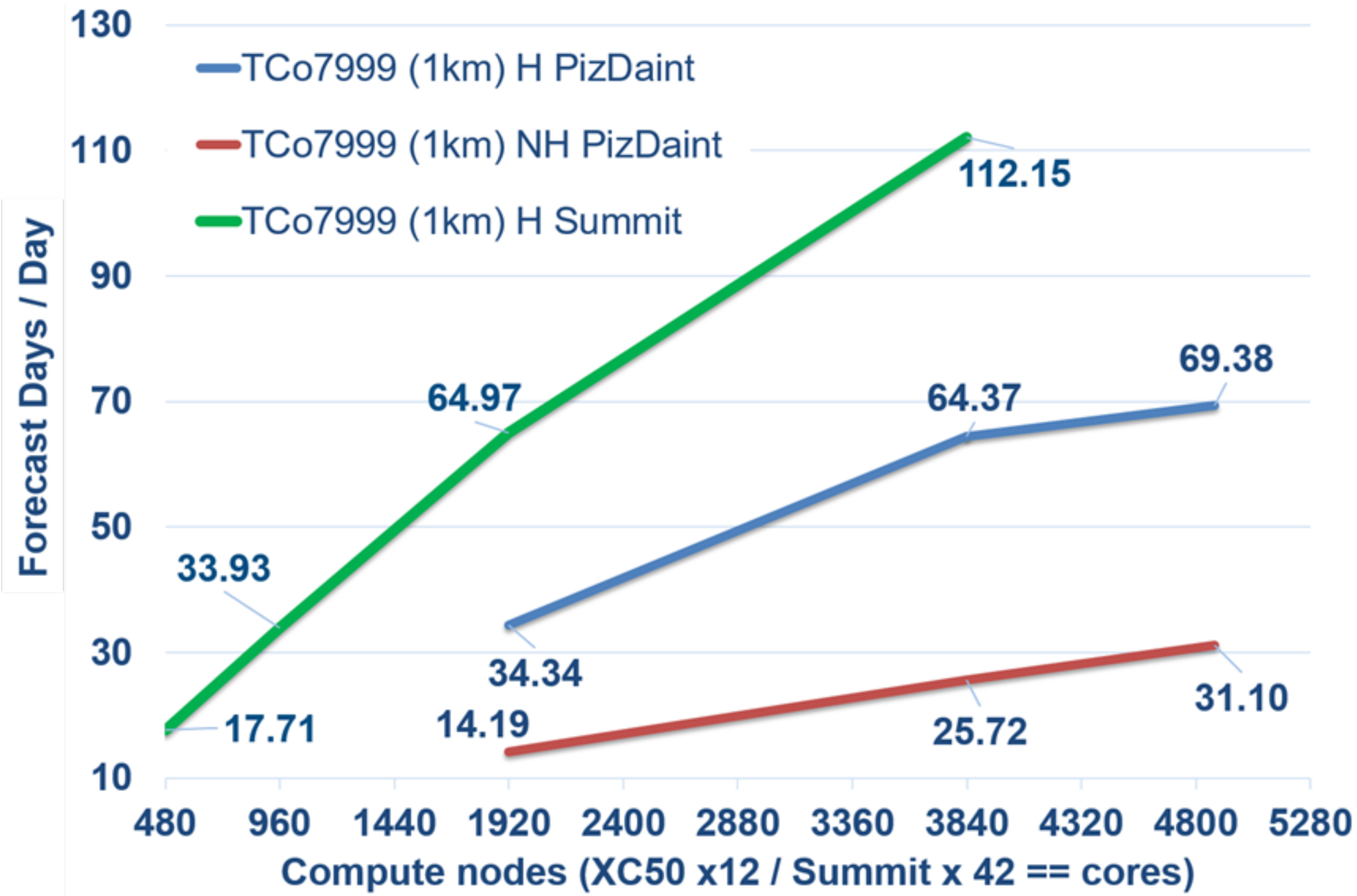
ECMWF

scalability comparison of IFS with other models



(Michalakes et al, NGGPS AVEC report, 2015)

IFS scaling on Summit and PizDaint (CPU only)



ECMWF

spectral transform vs discontinuous Galerkin

projected for 5km 2-day forecast

	DG, horizontally explicit => 4s time-step, almost no communication
communication volume:	**34 TB on 2880 MPI procs**
time to solution:	**4 hours**

ECMWF

spectral transform vs discontinuous Galerkin

projected for 5km 2-day forecast

	DG, horizontally explicit => 4s time-step, almost no communication	IFS (spectral transform): 240s time-step, lots of communication
communication volume:	**34 TB on 2880 MPI procs**	**427 TB on 2880 MPI procs**
time to solution:	**4 hours**	

ECMWF

spectral transform vs discontinuous Galerkin

projected for 5km 2-day forecast

	DG, horizontally explicit => 4s time-step, almost no communication	IFS (spectral transform): 240s time-step, lots of communication
communication volume:	**34 TB on 2880 MPI procs**	**427 TB on 2880 MPI procs**
time to solution:	**4 hours**	**12 minutes**

ECMWF

spectral transform vs discontinuous Galerkin

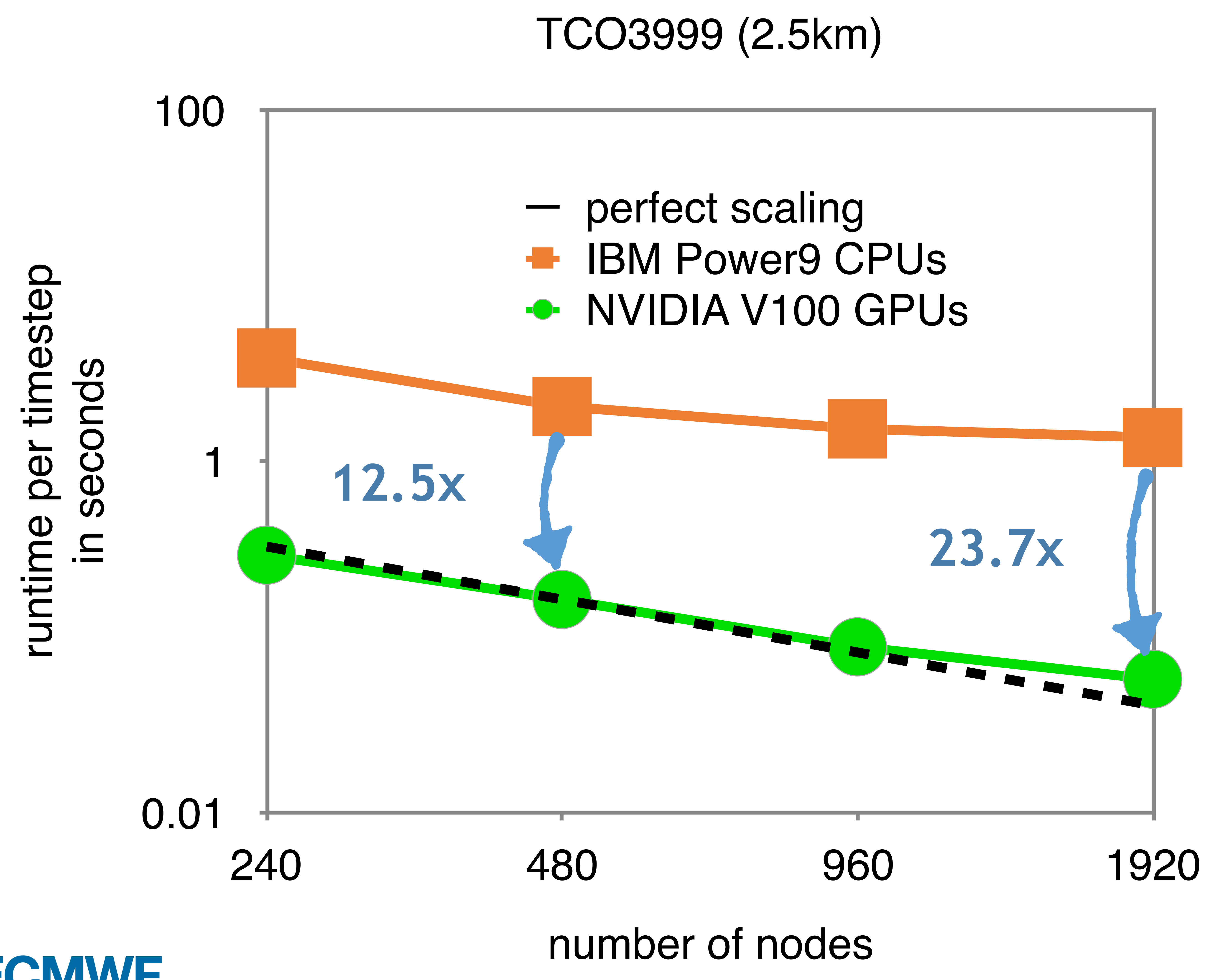
projected for 5km 2-day forecast

	DG, horizontally explicit => 4s time-step, almost no communication	IFS (spectral transform): 240s time-step, lots of communication	DG (like on the left)
communication volume:	**34 TB on 2880 MPI procs**	**427 TB on 2880 MPI procs**	**689 TB on 57600 MPI procs**
time to solution:	**4 hours**	**12 minutes**	**12 minutes**

ECMWF

GPUs vs CPUs on Summit

spectral transform only

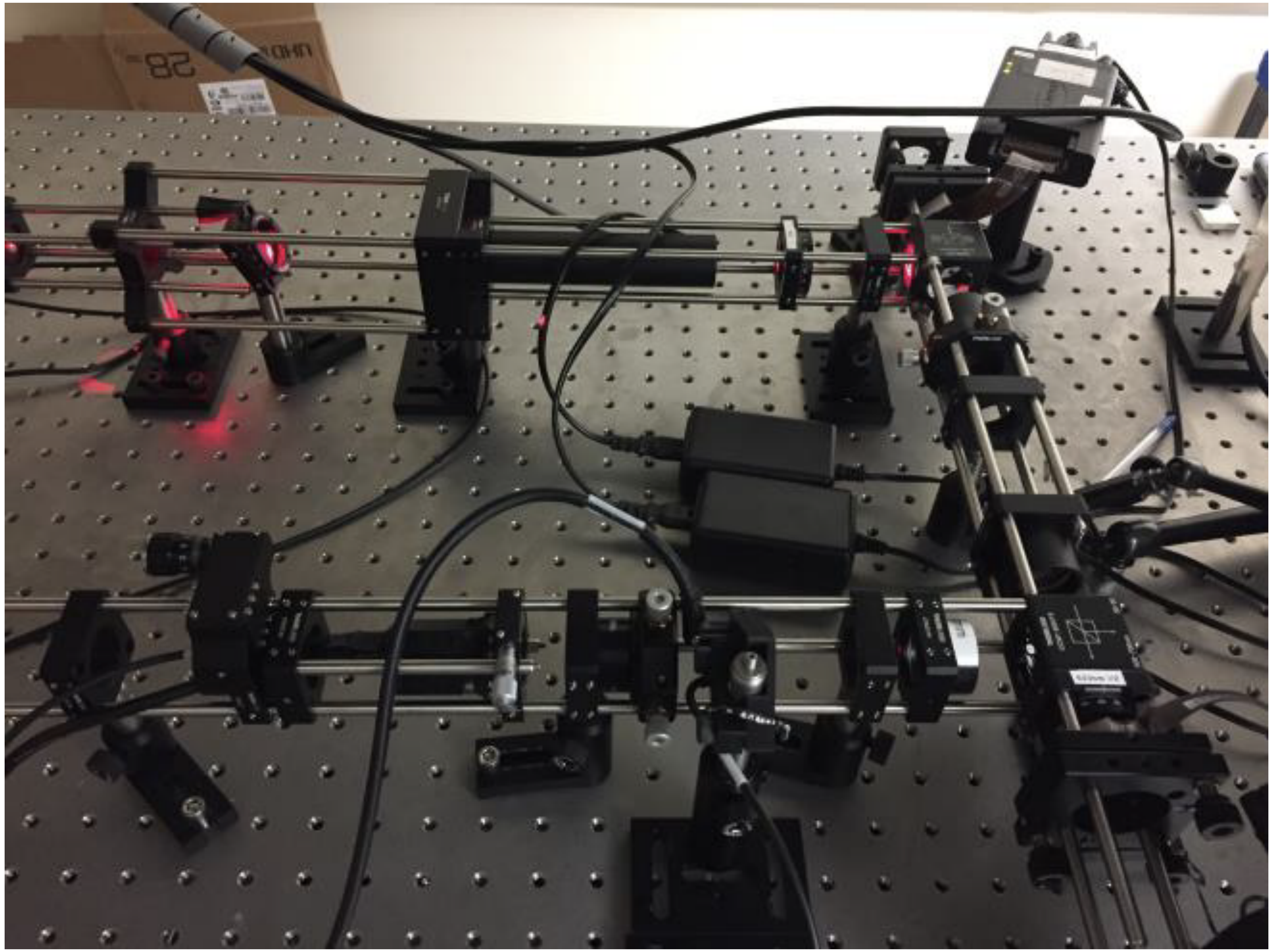


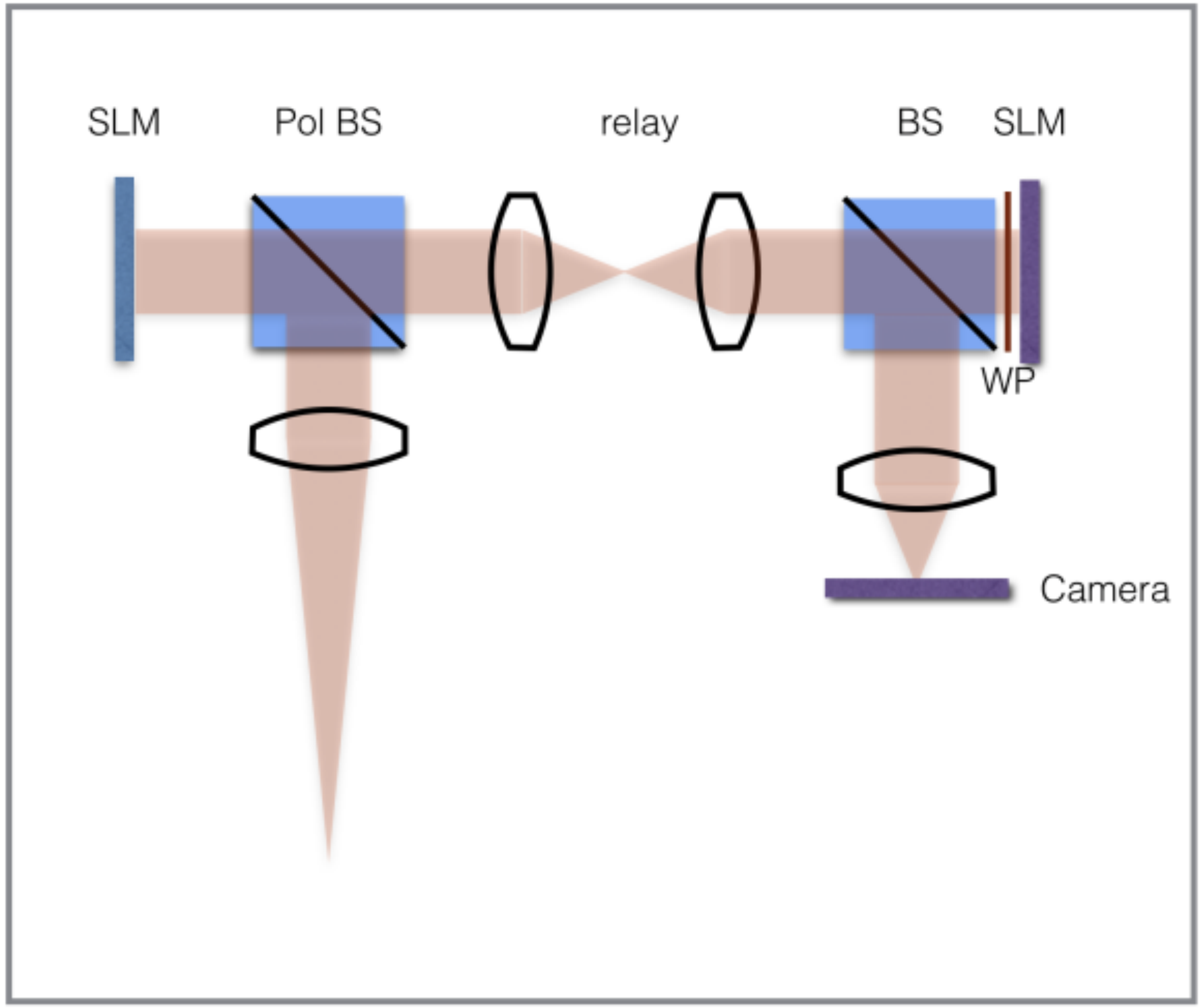
At 2.5km resolution, less than 1s per time-step fits operational needs.

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.

ECMWF

Optalysys: optical processor for spectral transform





Figures used with permission from Optalysys, 2017

Fast Legendre Transform

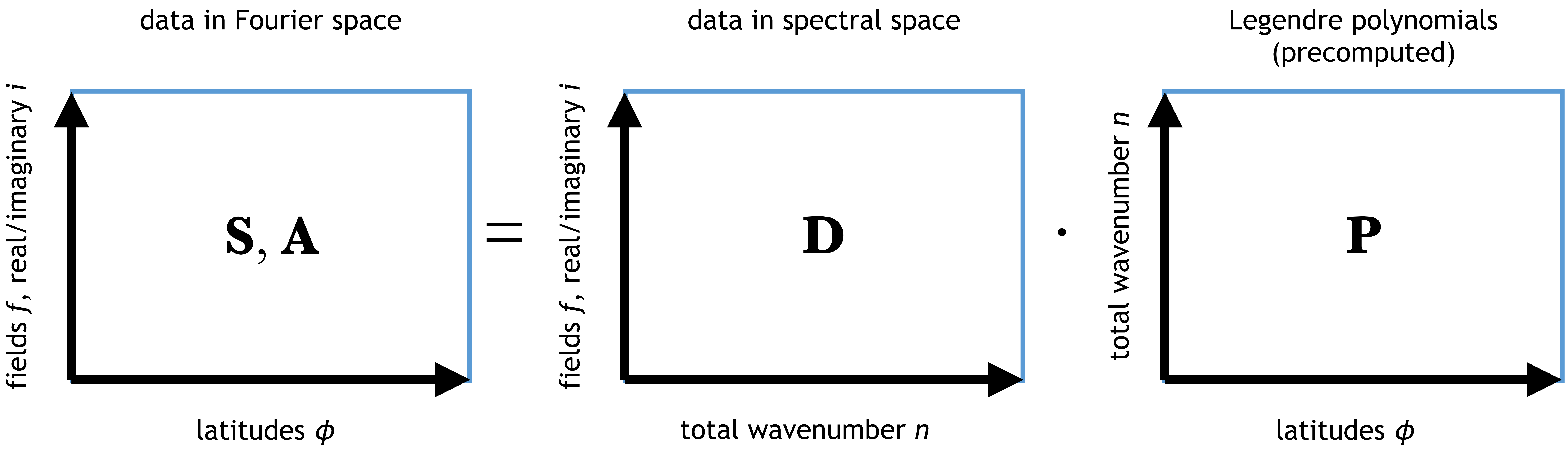
- recap: Legendre transform:

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

Fast Legendre Transform

- recap: Legendre transform:

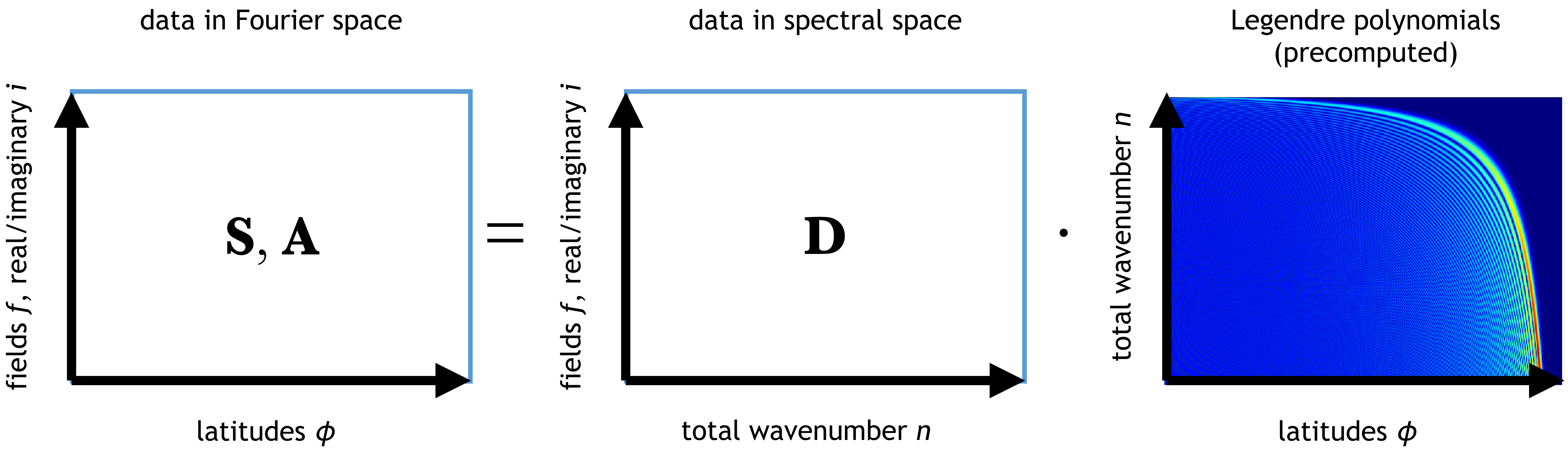
$$\mathbf{S}_m(f,\mathrm{i},\phi) = \sum_n \mathbf{D}_{e,m}(f,\mathrm{i},n) \cdot \mathbf{P}_{e,m}(n,\phi), \quad \mathbf{A}_m(f,\mathrm{i},\phi) = \sum_n \mathbf{D}_{o,m}(f,\mathrm{i},n) \cdot \mathbf{P}_{o,m}(n,\phi)$$



Fast Legendre Transform

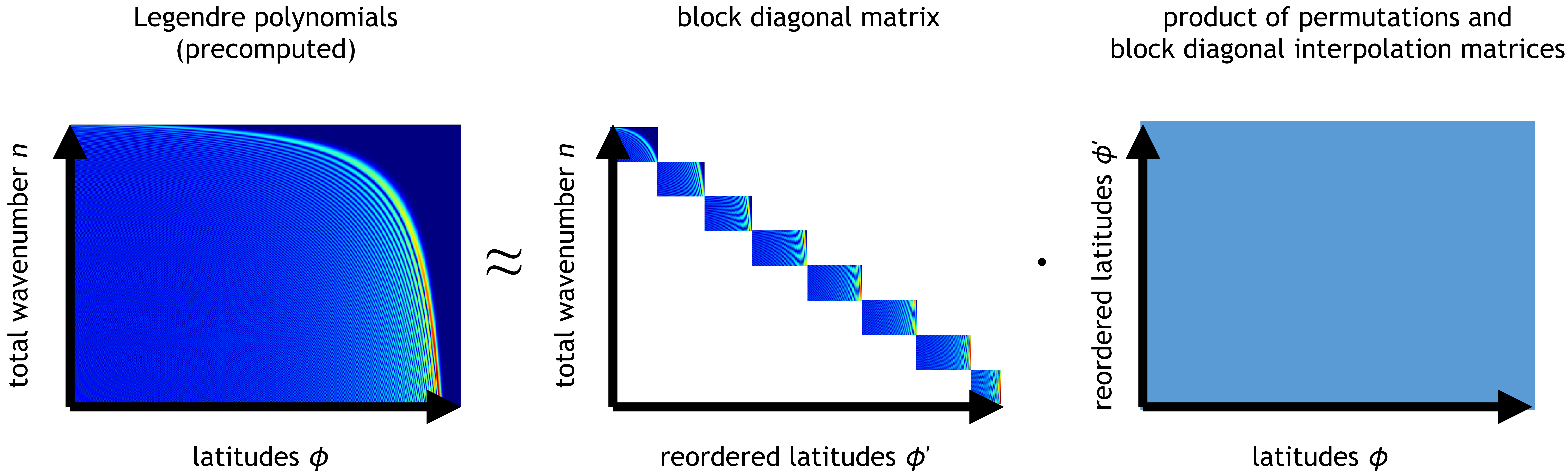
- recap: Legendre transform:

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



Fast Legendre Transform

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



Fast Legendre Transform

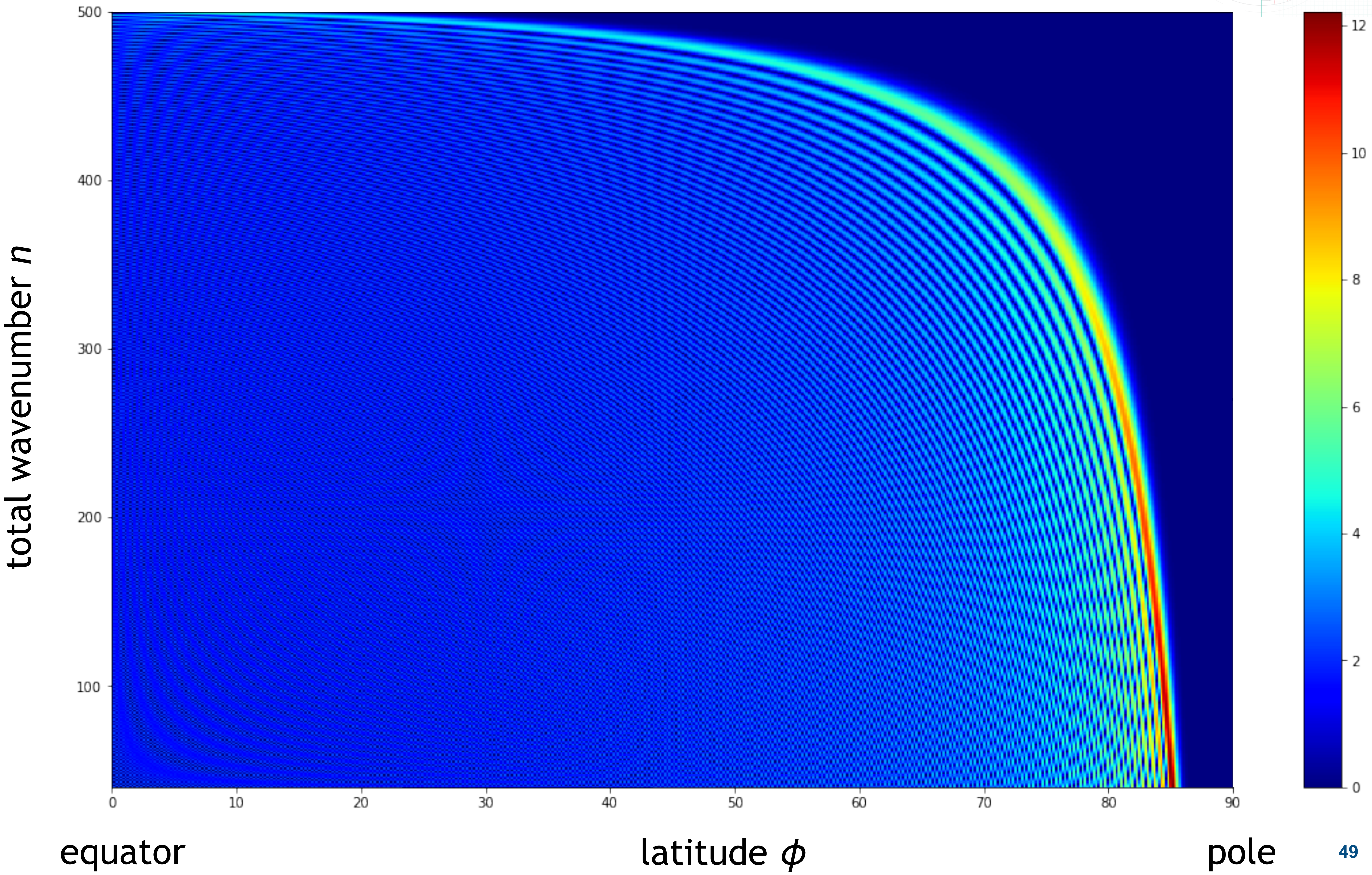
matrix of
Legendre polynomials

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



ECMWF

Fast Legendre Transform

matrix of
Legendre polynomials

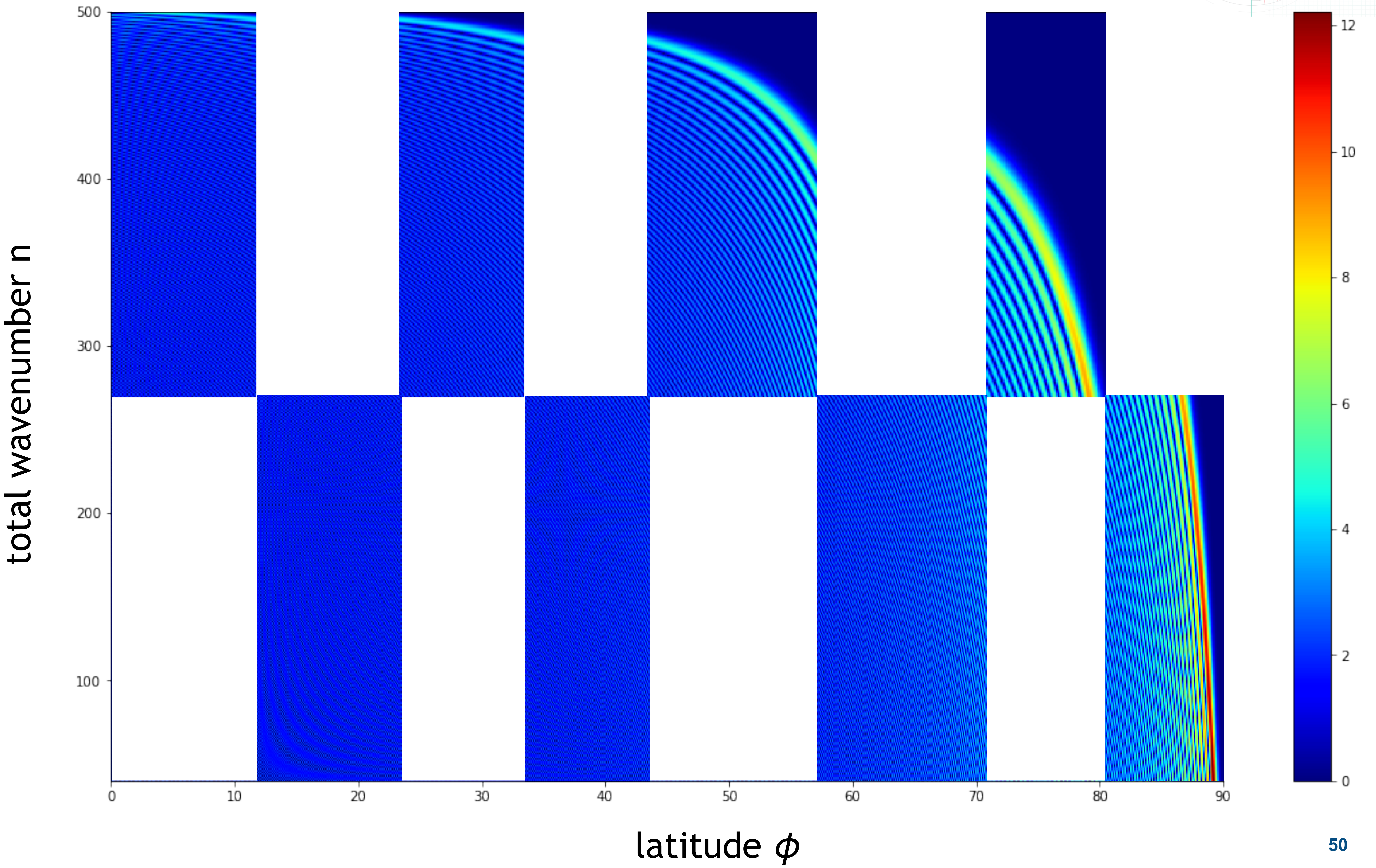
truncation N=500,
zonal wavenumber
m=40

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

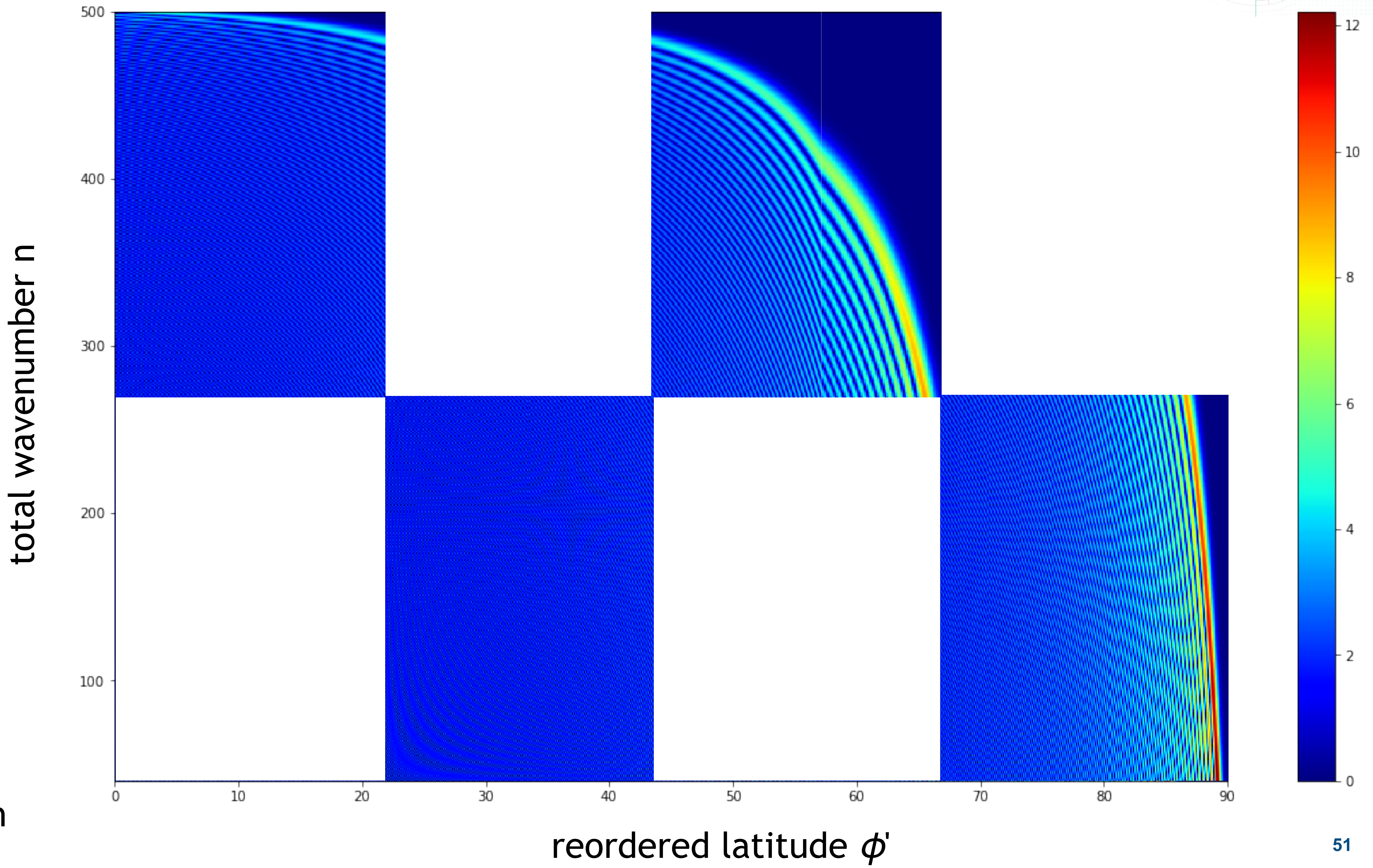
FLT:

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

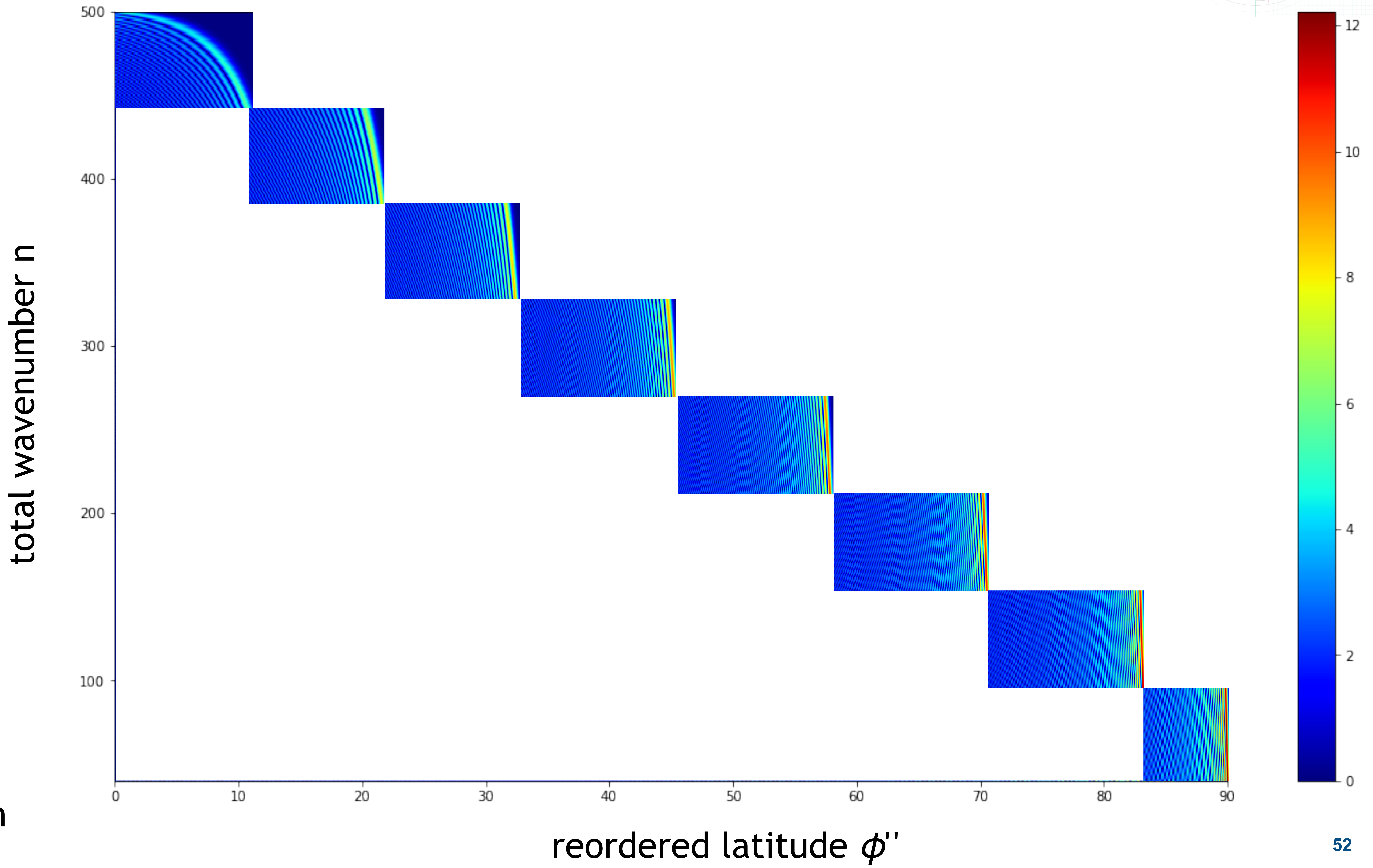
FLT:

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

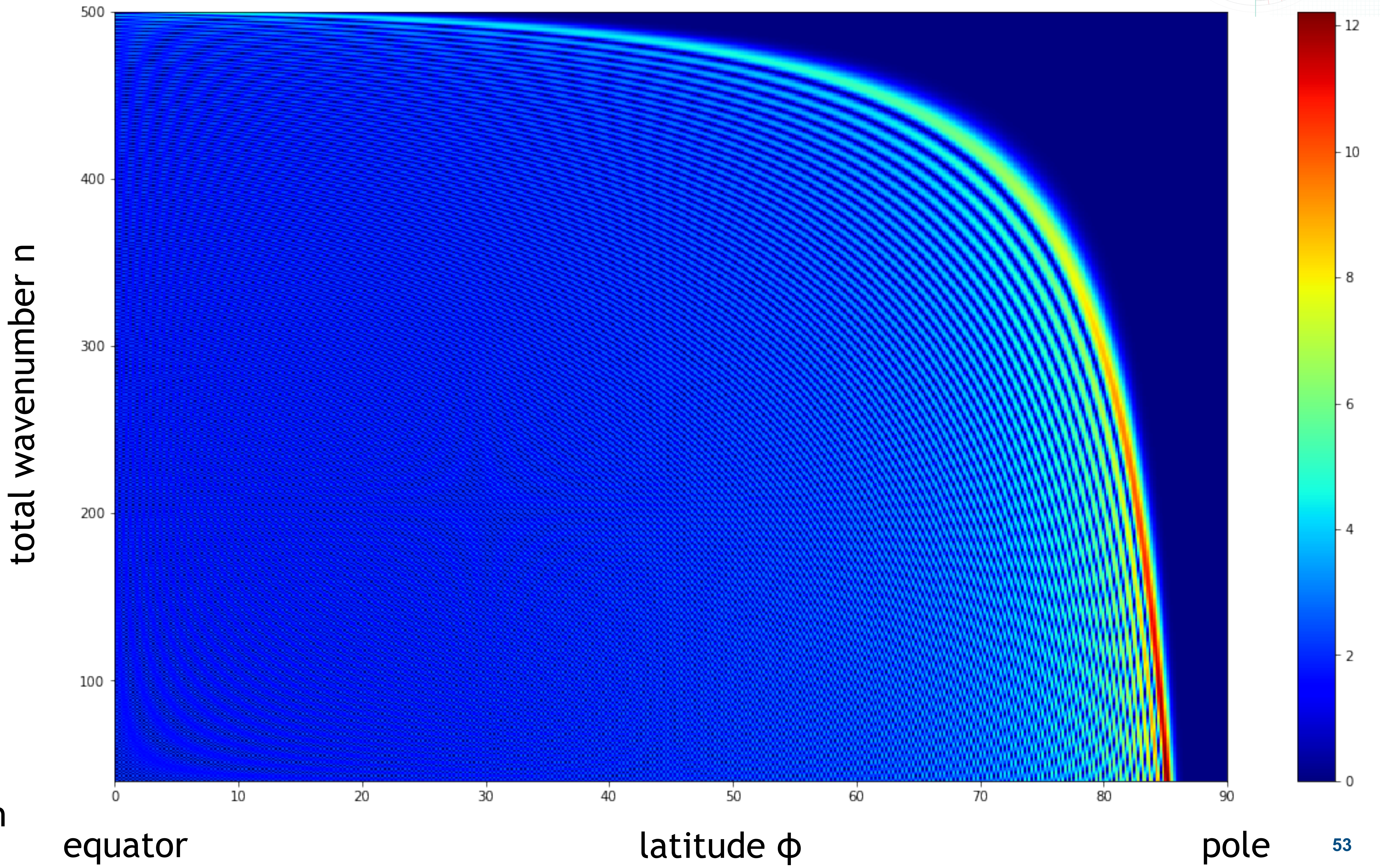
FLT:

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

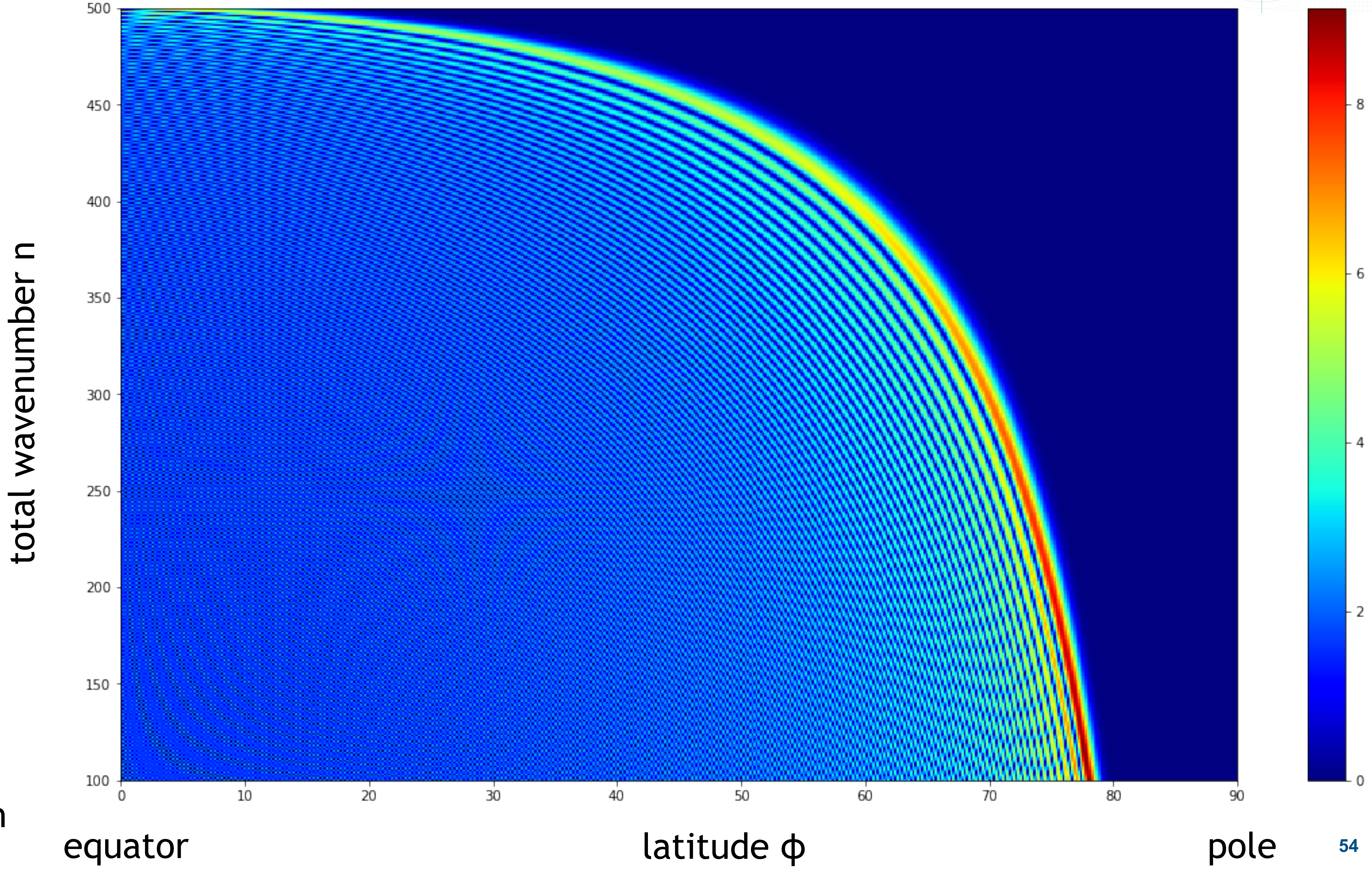
FLT:

step 1: split matrix into two rows

step 2: use interpolation to empty half of the columns

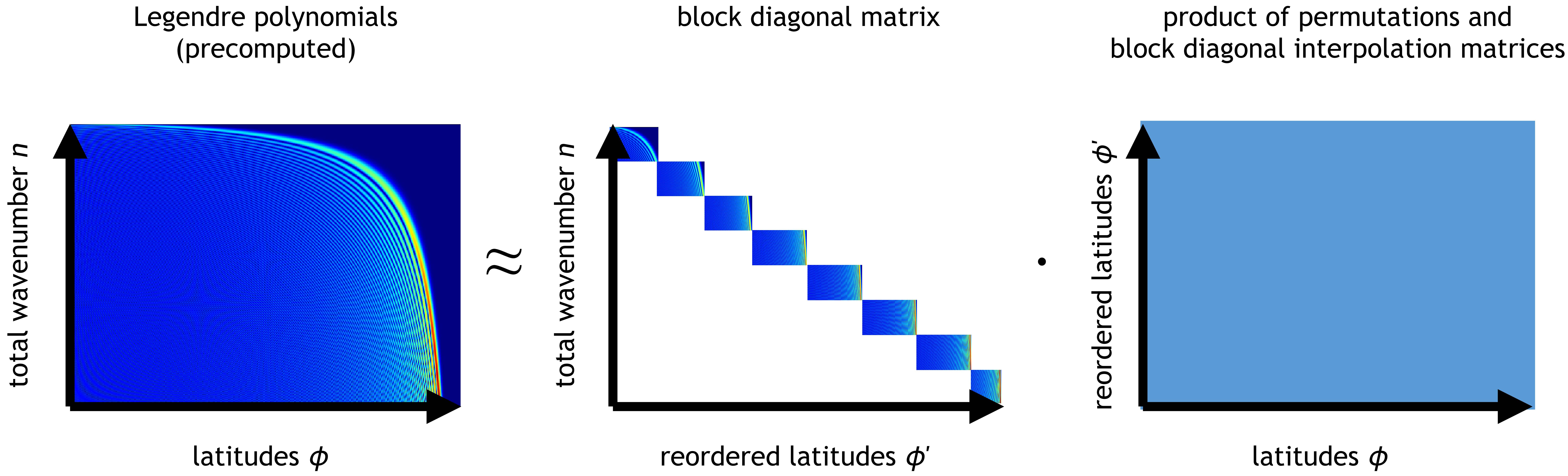
step 3: reorder columns

step 4: apply to each block recursively



Why should Fast Legendre Transform improve performance?

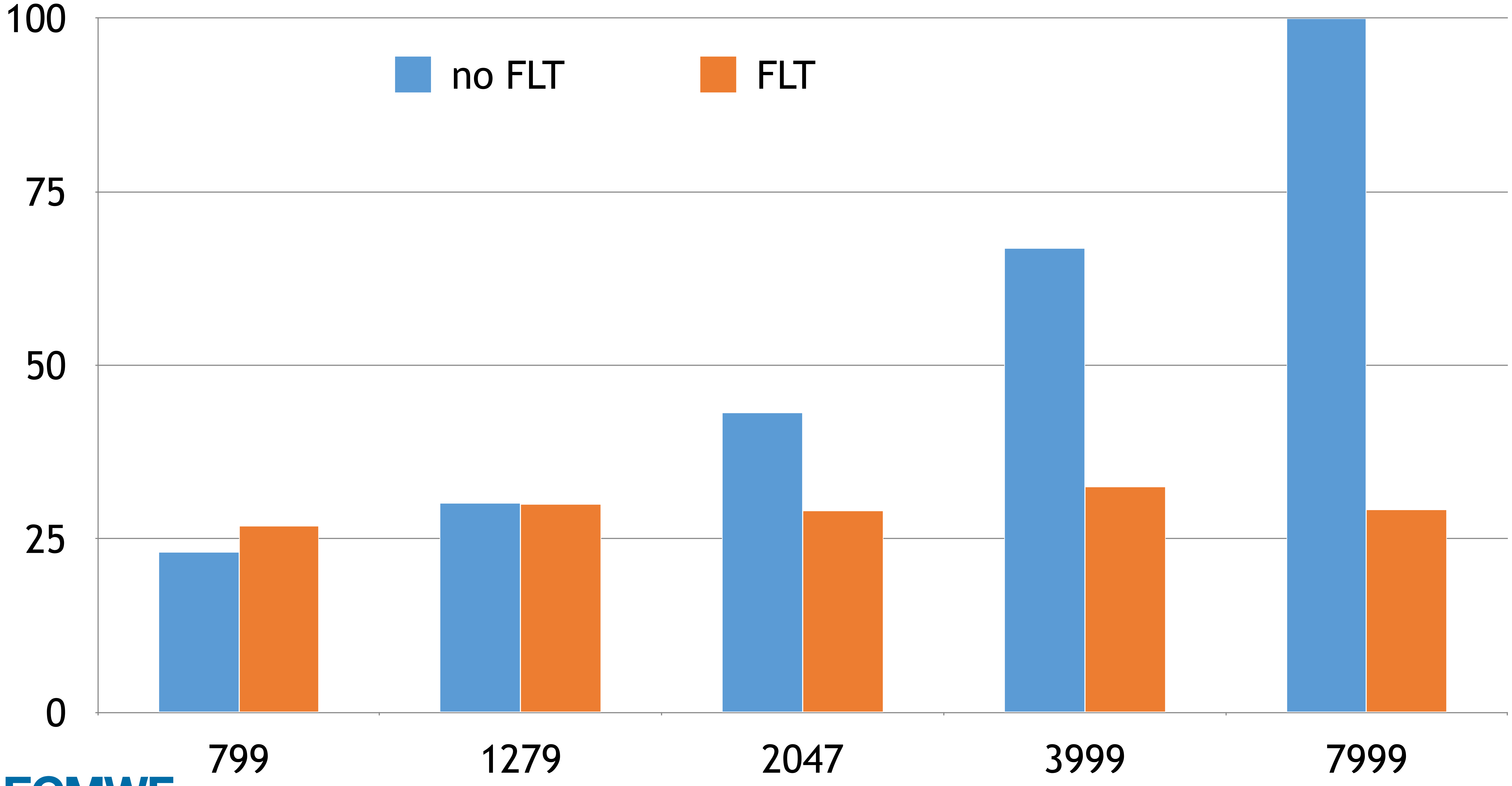
- recap:



- 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

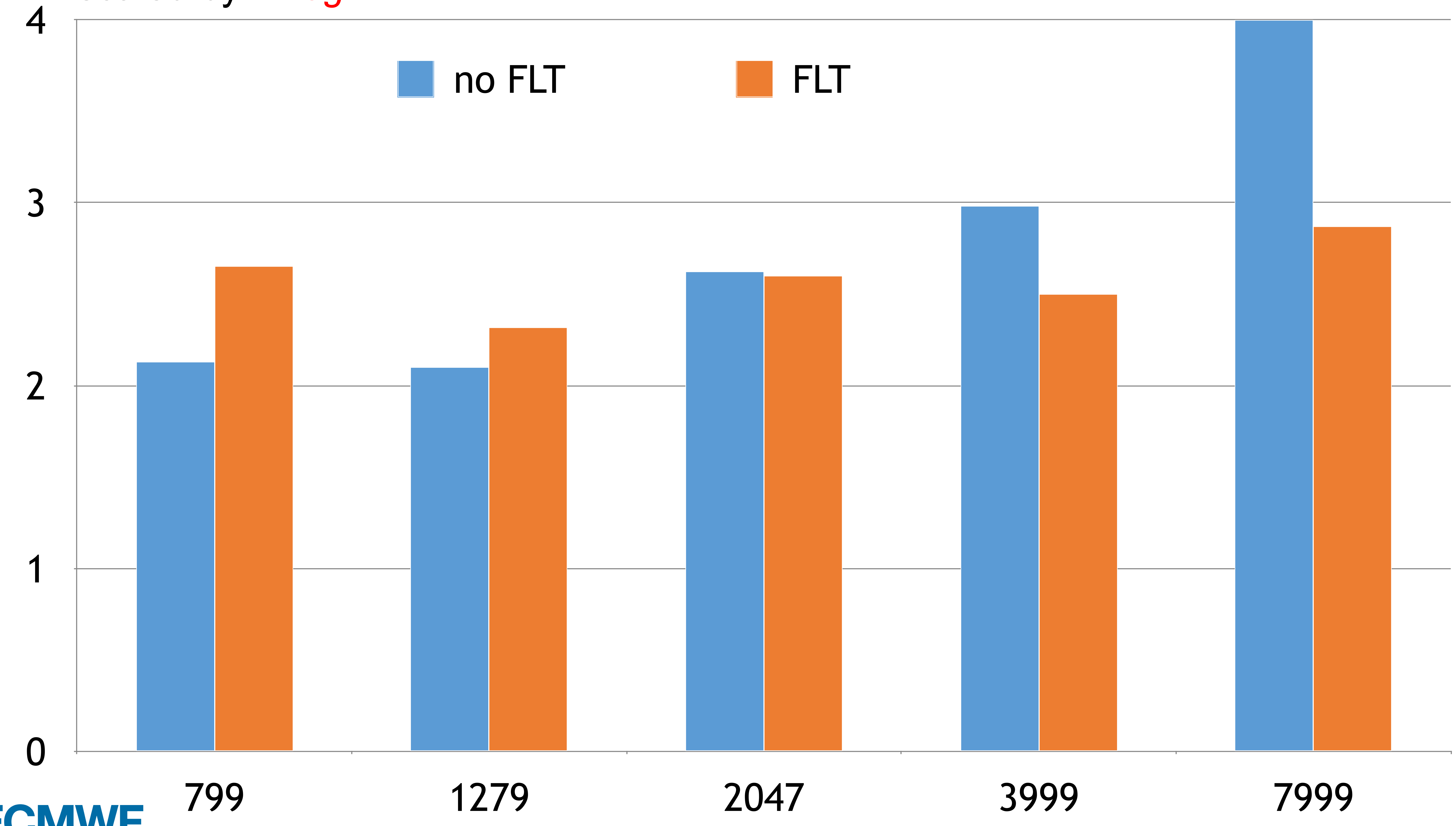
Number of floating point operations for direct or inverse spectral transforms of a single field, scaled by $N^2 log^3 N$



ECMWF

Fast Legendre Transform
wallclock time

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



ECMWF

Questions?