Numerical Simulation of Elastic Waves in 3D using FFT algorithm and MPI protocol

Peter Klin
Centro Ricerche Sismologiche (CRS)
OGS – Trieste, ITALY
pklin@inogs.it

Outline

Introduction

Case study: simulation of earthquake motion in 3D

Numerical solution of the wave equation

Fourier method

FFT algorithm

Parallel implementation with MPI

Domain decomposition I

Domain decomposition II

Introduction to the Lab Session (this afternoon)

Elastic waves propagation in 2D (with dom. dec. I)

Simulation of earthquake ground motion

Purpose:

Improve seismic hazard assessment in sites with

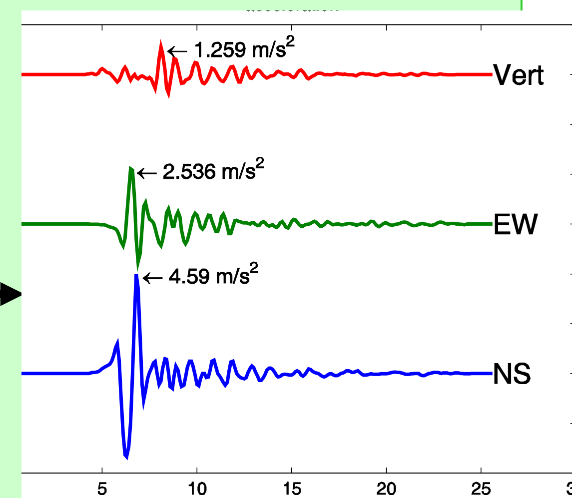
- lacking empirical ground motion data (past earthquakes)
- complex geology

Simulation of earthquake ground motion

Method:

Numerical simulation based on:

- seismic source model (wave generation)
- geological structure model (*wave propagation*)



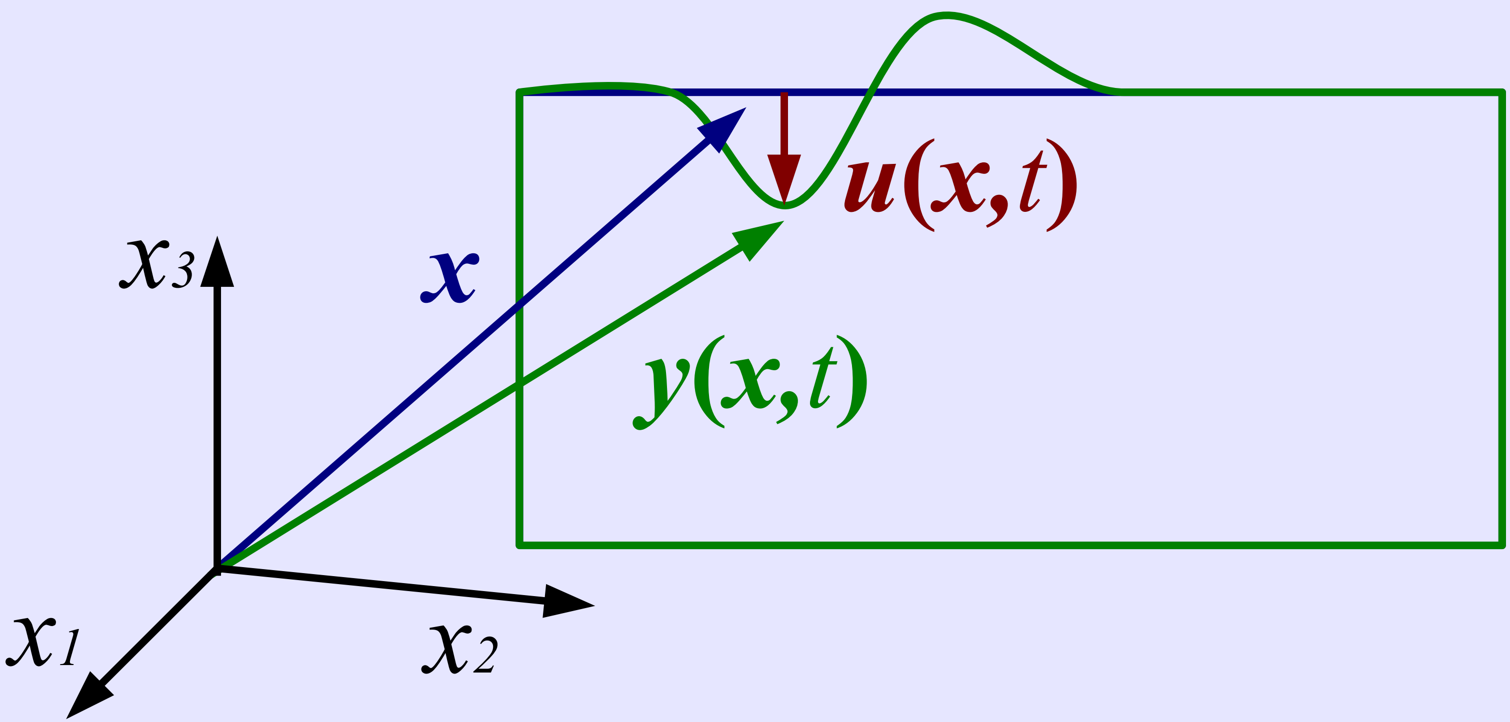
$$u(x,t)=s(t)*G(x,t)$$

Green's function

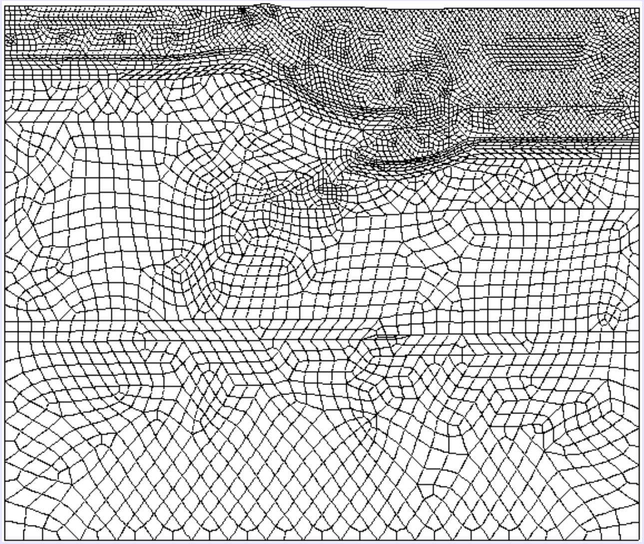
Simulation of earthquake ground motion

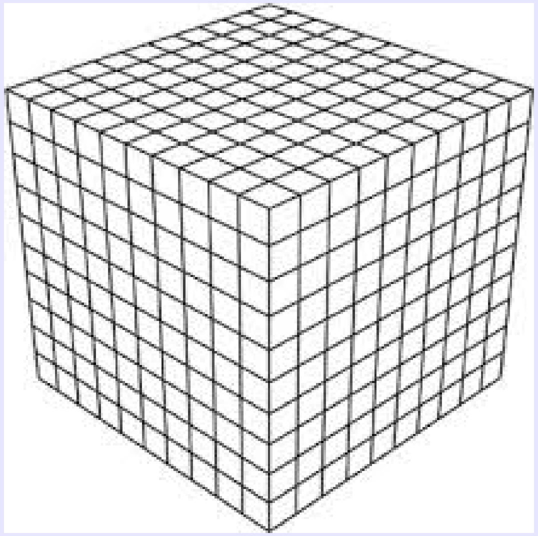
Physical quantity:

Displacement: $\boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{y}(\boldsymbol{x},t) - \boldsymbol{x}$ continuous and differentiable!



Spatial discretization





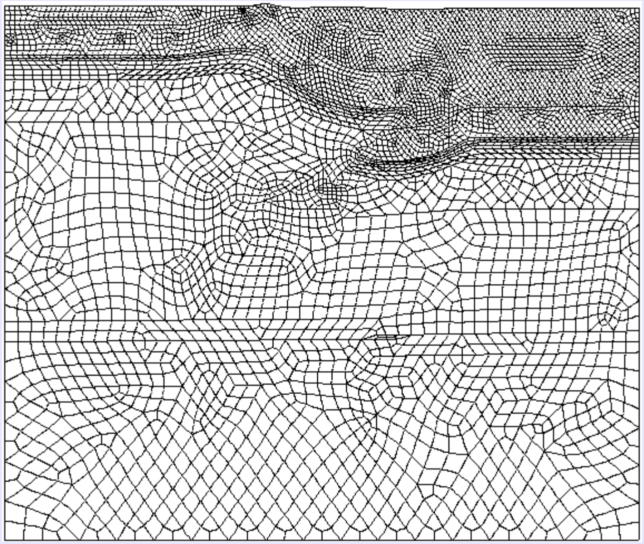
Unstructured grid

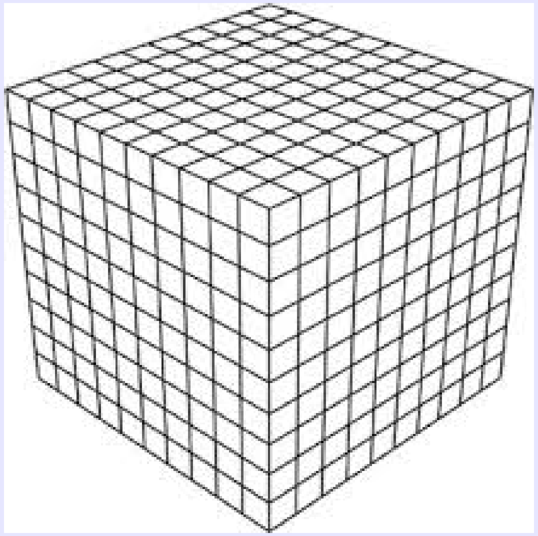
variational formulation
(spatial integrals)

Structured grid

variational formulation
&
differential formulation
(spatial derivatives)

Spatial discretization





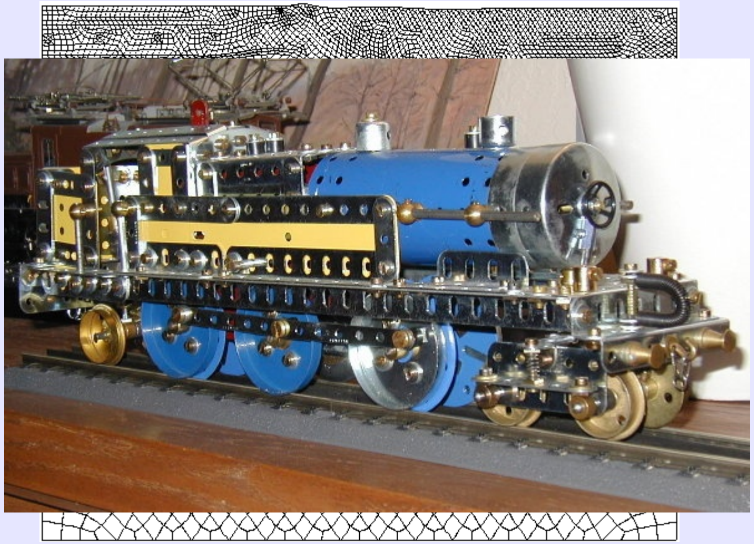
Unstructured grid | **Structured grid**

very flexible ⟷ less flexible

sophisticated for 3D ⟷ simple

Spatial discretization

Our choice

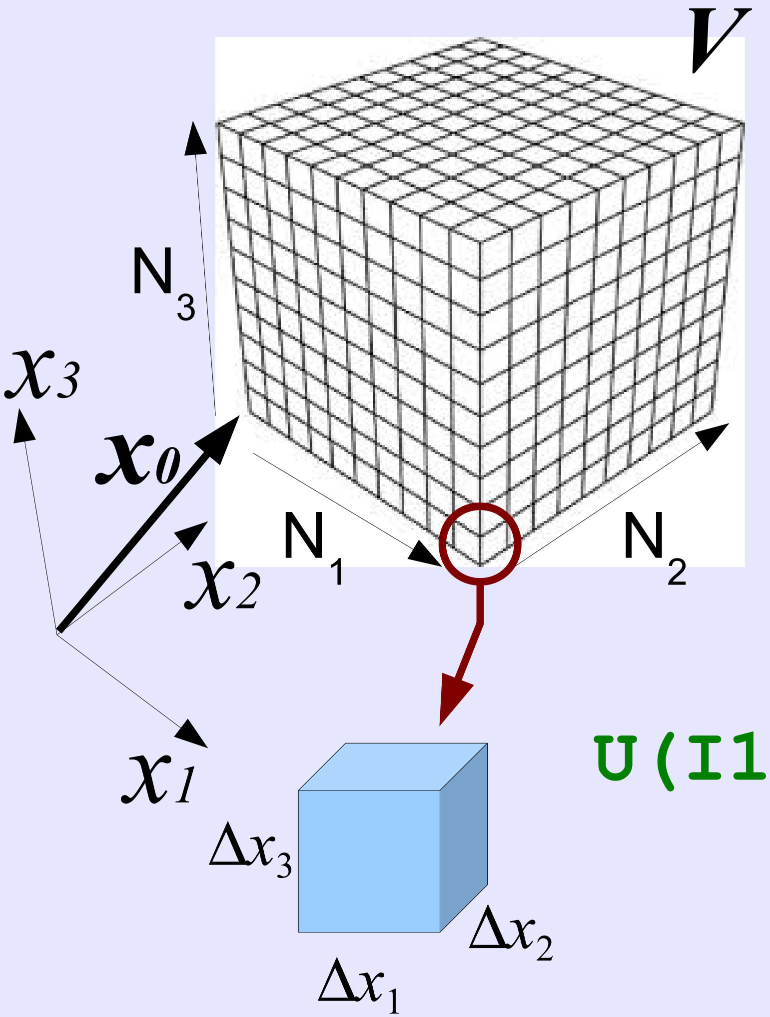


Unstructured grid | **Structured grid**

very flexible ⟷ less flexible

sophisticated for 3D ⟷ simple

Spatial discretization



Physical variable:

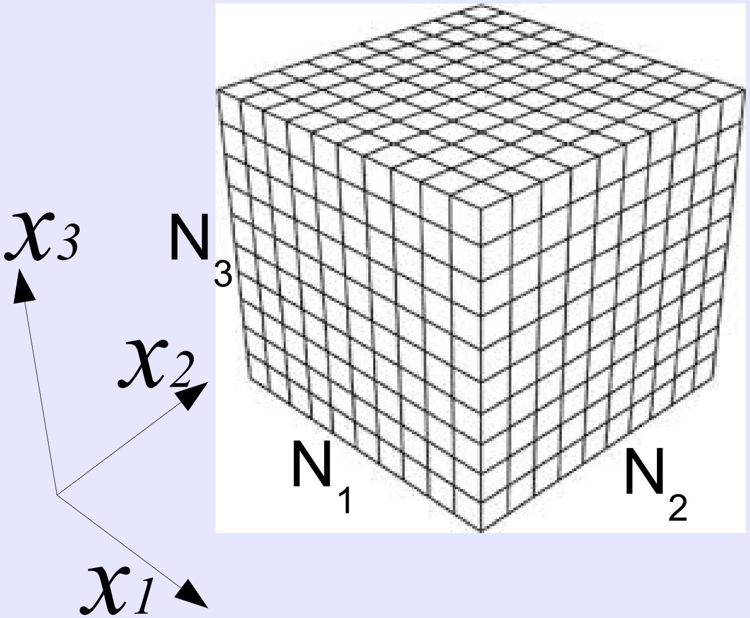
$$\boldsymbol{u}(\boldsymbol{x}) : V \subset \mathbb{R}^3 \rightarrow \mathbb{R}^3$$

Numerical variable (FORTRAN):

```
REAL,DIMENSION(N1,N2,N3,3) :: U
```

`U(I1,I2,I3,IC)` $= u_{ic}(\boldsymbol{x_0} + (i_1 \Delta x_1,\ i_2 \Delta x_2,\ i_3 \Delta x_3))$

Spatial derivatives in structured grid



`U(N1,N2,N3)` → $u(x_1, x_2, x_3)$

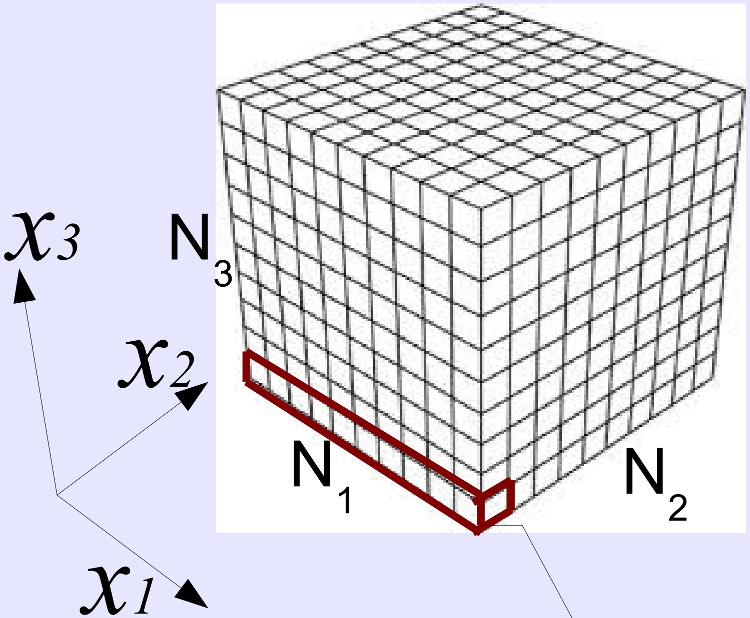
?

`D1U(N1,N2,N3)` → $\partial_1 u(x_1, x_2, x_3)$

`D2U(N1,N2,N3)` → $\partial_2 u(x_1, x_2, x_3)$

`D3U(N1,N2,N3)` → $\partial_3 u(x_1, x_2, x_3)$

Spatial derivatives in structured grid



`U(N1,N2,N3)` → $u(x_1, x_2, x_3)$

?

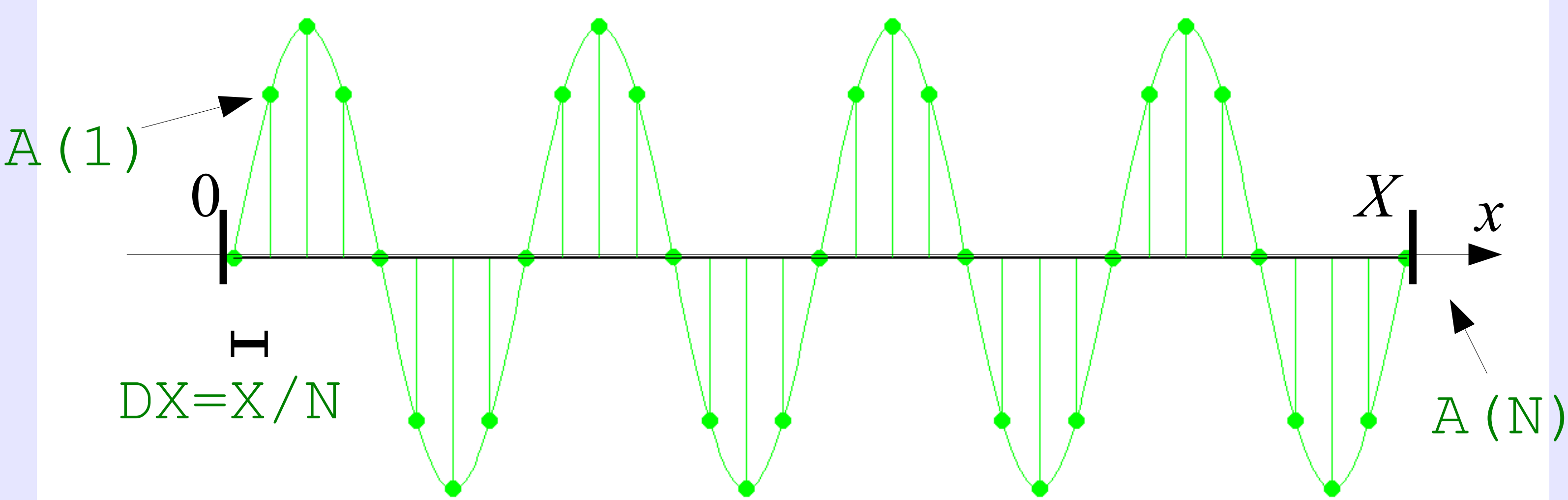
`D1U(N1,N2,N3)` → $\partial_1 u(x_1, x_2, x_3)$

`D2U(N1,N2,N3)` → $\partial_2 u(x_1, x_2, x_3)$

`D3U(N1,N2,N3)` → $\partial_3 u(x_1, x_2, x_3)$

`A = U(1:N1,1,1)` ? → `DA(1:N1)`

Spatial derivatives with the Fourier method

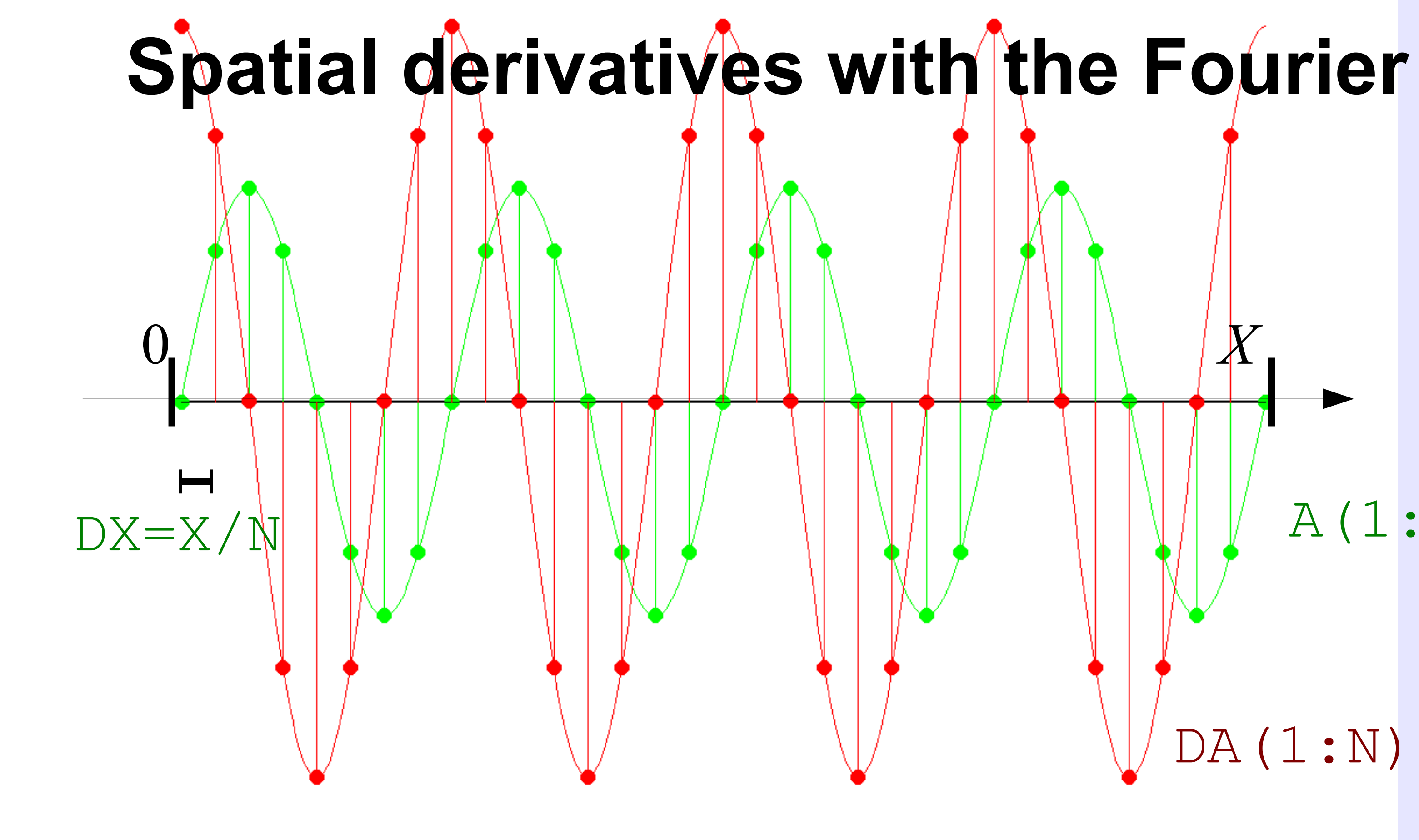



```
A(I) = C*SIN(W*I); I=1,2,...N
```

```
REAL :: C
REAL :: W=J*2*PI/N !        J.GE.0 & J.LT.N
```

Nyquist criterion

Spatial derivatives with the Fourier method



A(I) = C*SIN(W*I); I=1,2,...N

DA(I) = W*C*COS(W*I)/DX; I=1,2,...N

EXACT!

Spatial derivatives with the Fourier method

```
…
REAL, DIMENSION(N) :: A, DA
REAL, PARAMETER :: DX=X/REAL(N)
REAL, PARAMETER :: W= J*2*PI/N !J.GE.0 & J.LT.N
REAL :: C
```

```
A(I) = C*SIN(W*I)  ⟶  DA(I) = W*C*COS(W*I)/DX

A(I) = C*COS(W*I)  ⟶  DA(I) = -W*C*SIN(W*I)/DX
```

I=1,2,...N

Spatial derivatives with the Fourier method

```
…
COMPLEX, DIMENSION (N) :: A,DA
COMPLEX, PARAMETER :: AI = CMPLX(0.0,1.0)
REAL, PARAMETER :: DX=X/REAL(N)
REAL, PARAMETER :: W= J*2*PI/N
COMPLEX :: C
```

```
A(I)=C*EXP(AI*W*I) → DA(I)=C*AI*W*EXP(AI*W*I)/DX
```

I=1,2,...N

$$\exp(i\phi) = \sin(\phi) + i\cos(\phi)$$

Spatial derivatives with the Fourier method

```
COMPLEX, DIMENSION (N) :: A,DA
COMPLEX :: AI = CMPLX(0.0,1.0)
REAL, PARAMETER :: DX=X/REAL(N)
REAL, DIMENSION(N), PARAMETER:: W=(J*2*PI/N,J=0,N-1)
COMPLEX, DIMENSION(N):: C !DEFINED SOMEHOW
```

```
A(I) = SUM(C(:)*EXP(AI*W(:)*I))

DA(I) = SUM( AI*W(:)*C(:)*EXP(AI*W(:)*I))/DX
```

I=1,2,...N

$$\partial_x [\sum_n c_n \exp(i k_n x)] = \sum_n i k_n c_n \exp(i k_n x)$$

Spatial derivatives with the Fourier method

```
COMPLEX, DIMENSION (N) :: A,DA
COMPLEX :: AI = CMPLX(0.0,1.0)
REAL, PARAMETER :: DX=X/REAL(N)
REAL, DIMENSION(N), PARAMETER:: W=(J*2*PI/N,J=0,N-1)
COMPLEX, DIMENSION(N):: C
```

```
A(I) = SUM(C(:)*EXP(AI*W(:)*I))

DA(I) = SUM( AI*W(:)*C(:)*EXP(AI*W(:)*I))/DX
```

I=1,2,...N

A(:) → ? → C(:)

Spatial derivatives with the Fourier method

```
COMPLEX, DIMENSION (N) :: A,DA
COMPLEX :: AI = CMPLX(0.0,1.0)
REAL, PARAMETER :: DX=X/REAL(N)
REAL, DIMENSION(N), PARAMETER:: W=(J*2*PI/N,J=0,N-1))
COMPLEX, DIMENSION(N):: C
```

Discrete Fourier Transform

```
C(I) = SUM(A(:)*EXP(-AI*W(:)*I))
```

I=1,2,...N

Inverse Discrete Fourier Transform

```
A(I) = SUM(C(:)*EXP(AI*W(:)*I))/N
```

I=1,2,...N

Spatial derivatives with the Fourier method

```
COMPLEX, DIMENSION(N):: A,DA
COMPLEX, DIMENSION(N):: AIW
AIW=CMPLX(0.0,(J*2*PI/N,J=0,N-1))
DA=IDFT(AIW*DFT(A))/(N*DX)
```

Dynamic meteorology and oceanography (Kreiss & Oliger,1973)

2D acoustic waves (Gazdag 1981)
2D elastic waves (Kosloff et al. 1984)

Spatial derivatives with the Fourier method

```
COMPLEX, DIMENSION(N):: A,DA
COMPLEX, DIMENSION(N):: AIW

AIW=CMPLX(0.0,(J*2*PI/N,J=0,N-1))

DA=IDFT(AIW*DFT(A))/(N*DX)
```

```
 FUNCTION IDFT(C)
DO I=1,N
 IDFT(I)=SUM( C(:) * EXP(AIW(:)*I))
ENDDO
```

```
 FUNCTION DFT(A)
DO I=1,N
 DFT(I)=SUM( A(:) * EXP(-AIW(:)*I))
ENDDO
```

Spatial derivatives with the Fourier method

```
COMPLEX, DIMENSION(N):: A,DA
COMPLEX, DIMENSION(N):: AIW

AIW=CMPLX(0.0,(J*2*PI/N,J=0,N-1))
DA=IDFT(AIW*DFT(A))/(N*DX)
```

```
 FUNCTION IDFT(C)
DO I=1,N
 IDFT(I)=SUM( C(:) * EXP(AIW(:)*I
ENDDO
```

```
 FUNCTION DFT(A)
DO I=1,N
 DFT(I)=SUM( A(:) * EXP(-AIW(:)*I)
ENDDO
```

runtime = $O(N^2)$

NOT for HPC !

Fast Fourier Transform (FFT)

- many different algorithms for evaluating DFT
- runtime improvement from $\mathbf{O}(N^2)$ to $\mathbf{O}(N \log_2 N)$

OK for HPC!

- complex & real data
- several libraries and packages (FFTPACK, FFTW...)

Use always FFT, never implement DFT

For more details see last week Gavin Pringle presentation:
http://indico.ictp.it/event/a13229/session/8/contribution/35/material/0/1.pdf

Spatial derivatives using FFTW3

```
INCLUDE 'fftw3.f'
COMPLEX, DIMENSION(N):: A,DA
COMPLEX, DIMENSION(N):: AIW=&
                    CMPLX(0.0,(J*2*PI/N,J=0,N-1))/(N*DX)
COMPLEX, DIMENSION(N):: C
INTEGER*8, DIMENSION(2) :: I8PLAN
```

Setup FFT

```
CALL SFFTW_PLAN_DFT_1D(I8PLAN(1),&
                       N,A,C,FFTW_FORWARD,FFTW_ESTIMATE)

CALL SFFTW_PLAN_DFT_1D(I8PLAN(2),&
                       N,C,DA,FFTW_BACKWARD,FFTW_ESTIMATE)
```

Apply FFT

```
CALL SFFTW_EXECUTE_DFT(I8PLAN(1),A,C)
C=C*AIW
CALL SFFTW_EXECUTE_DFT(I8PLAN(2),C,DA)
```

Additional speed up with FFT

1) Buy one & get one free

Complex array → **FFT** → Complex array

```
REAL, DIMENSION(N,2) :: U
COMPLEX, DIMENSION(N):: A,DA

A = CMPLX(U(1:N,1),U(1:N,2))

DA=IFFT(AIW*FFT(A))

D1U(1:N,1)=REAL(DA)
D1U(1:N,2)=AIMAG(DA)
```

Additional speed up with FFT

2) From real to complex and back (N even)

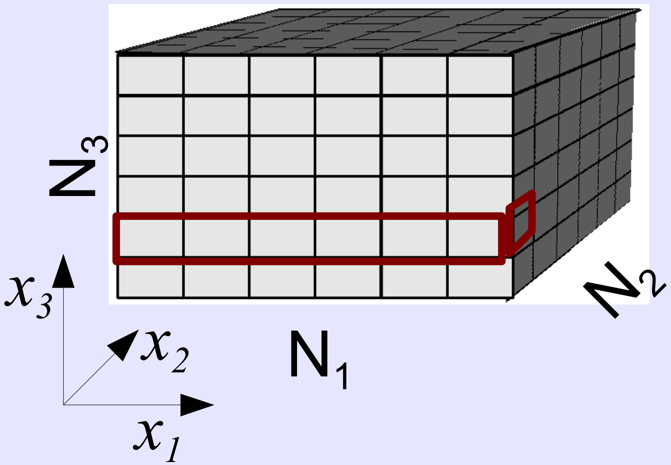
real (*N*) → **FFT_R2C** → complex (N/2+1)

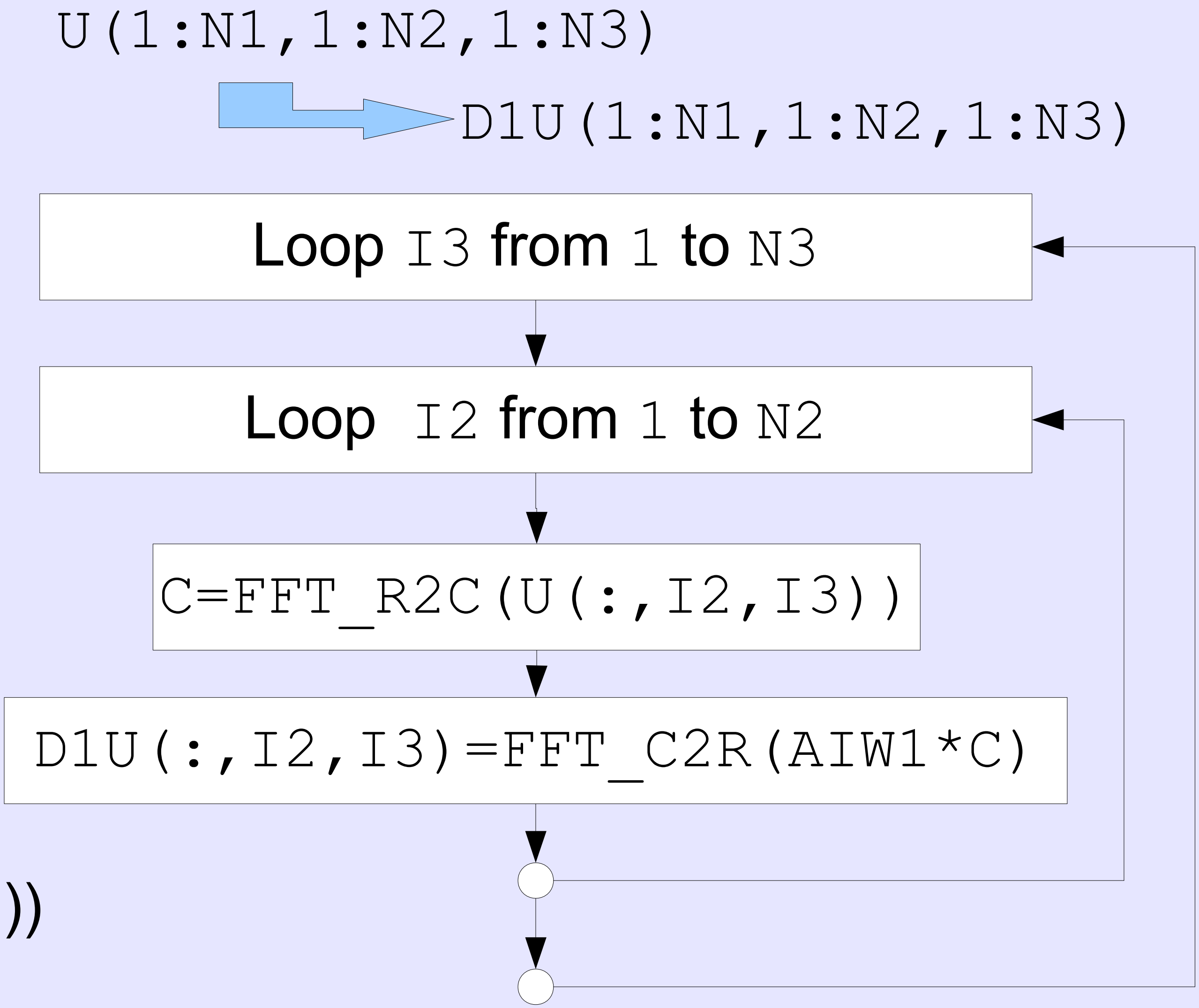
complex (N/2+1) → **FFT_C2R** → real (*N*)

```
REAL, DIMENSION(N) :: U, DU
COMPLEX, DIMENSION(N/2+1):: C,AIW

C=FFT_R2C(U)
DU=FFT_C2R(AIW*C)
```

FFT_R2C & FFT_C2R (almost) half less expensive than FFT

Spatial derivatives with FFT in a volume

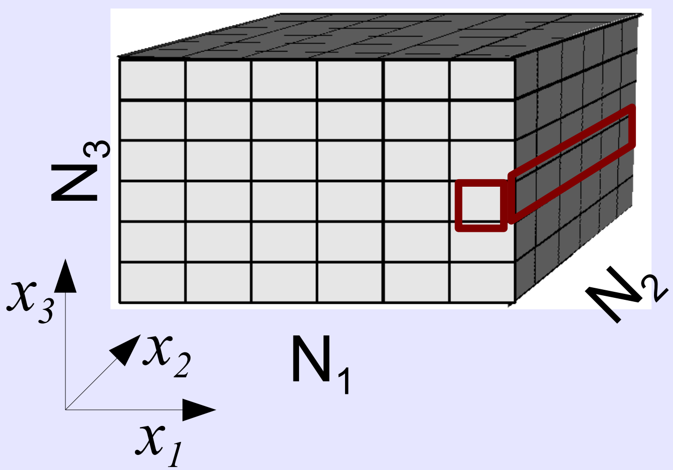




Runtime

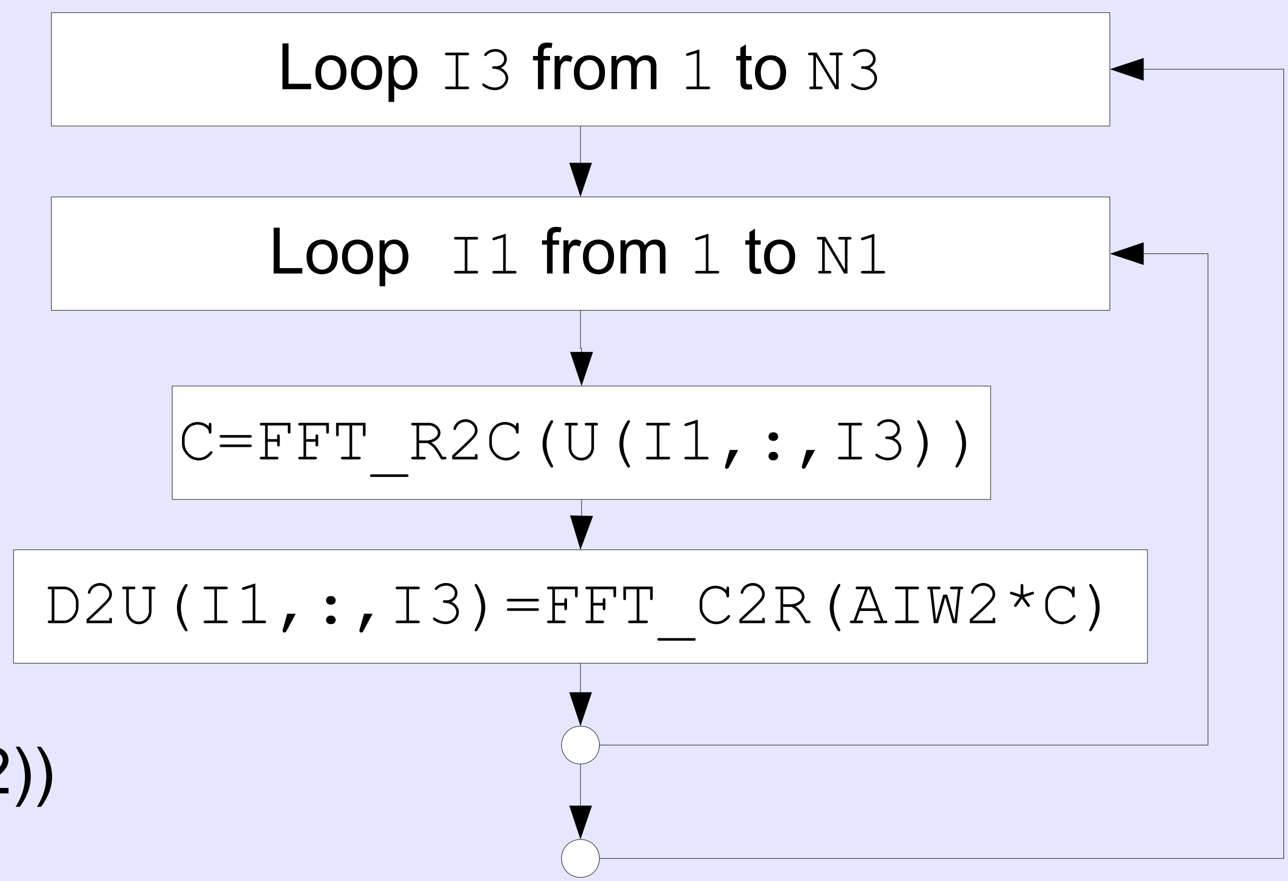
N3*N2*N1*(1+2*log2(N1))

Spatial derivatives with FFT in a volume



U(1:N1,1:1:N2,1:N3)

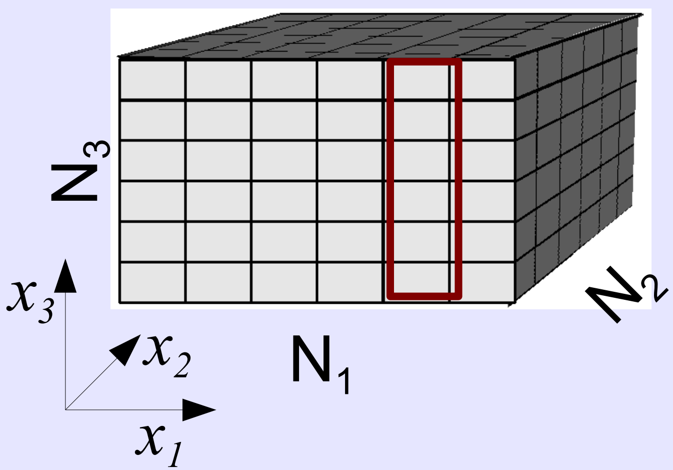
D2U(1:N1,1:N2,1:N3)

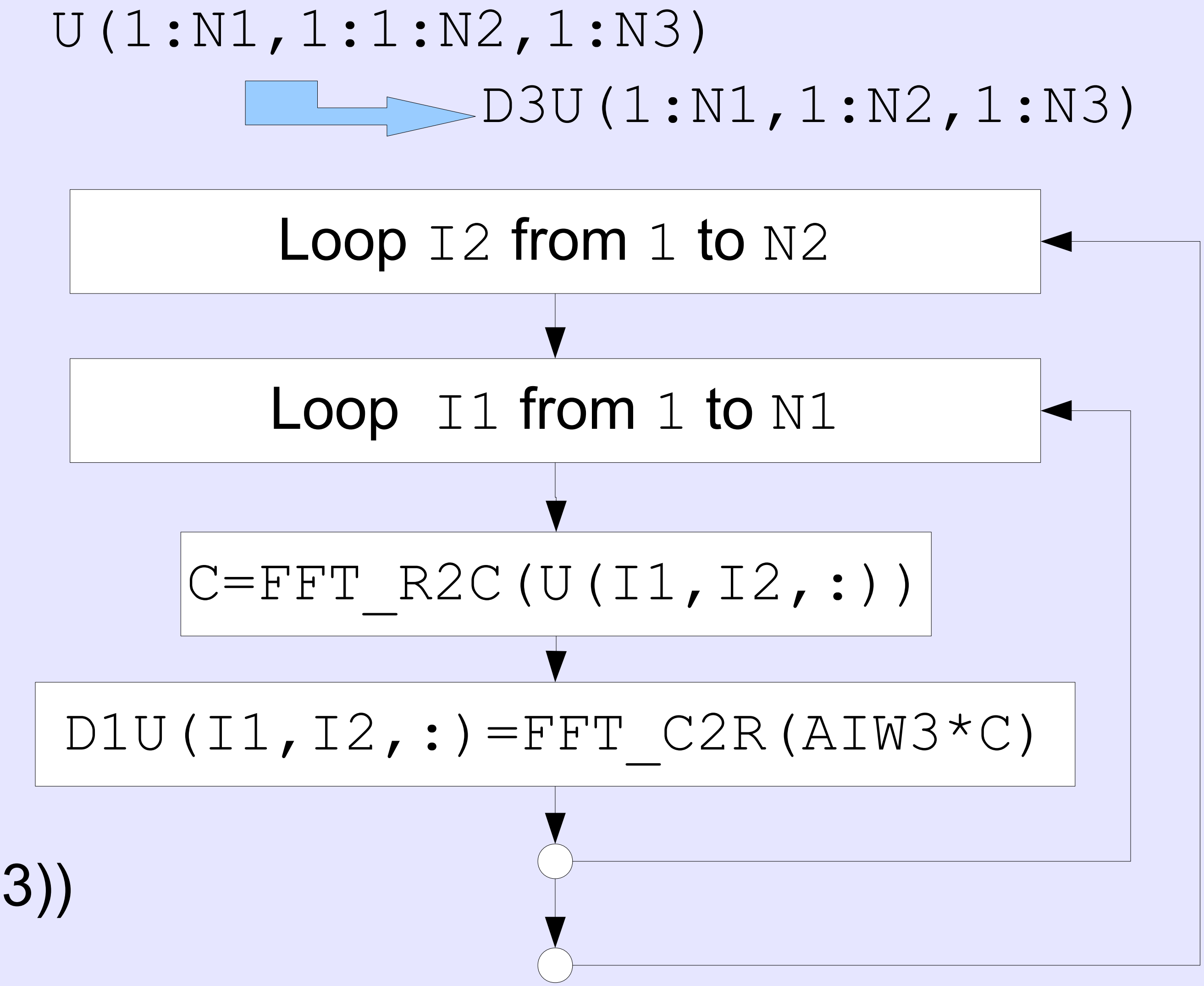


<u>Runtime</u>

N3*N2*N1*(1+2*log2(N2))

Spatial derivatives with FFT in a volume





<u>Runtime</u>

N3*N2*N1*(1+2*log2(N3))

Elastic waves

$$\partial_{tt}\boldsymbol{u} = \rho^{-1}\left(\nabla\left[\boldsymbol{C}\nabla^T\boldsymbol{u}\right]+\boldsymbol{f}\right)$$ Equation of motion

Displacement vector

$$\boldsymbol{u} = \left(u_{1,}u_{2,}u_3\right)^T$$

Elasticity matrix:

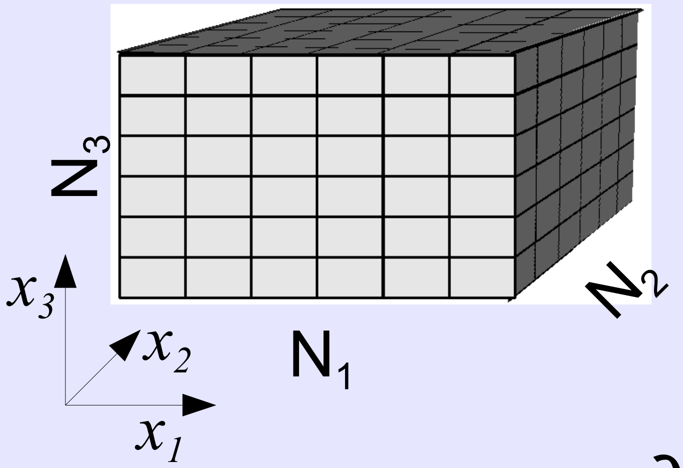
$$\boldsymbol{C} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} \\ c_{12} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} \\ c_{13} & c_{23} & c_{33} & c_{34} & c_{35} & c_{36} \\ c_{14} & c_{24} & c_{34} & c_{44} & c_{45} & c_{46} \\ c_{15} & c_{25} & c_{35} & c_{45} & c_{55} & c_{56} \\ c_{16} & c_{26} & c_{36} & c_{46} & c_{56} & c_{66} \end{pmatrix}$$

Auld's differential operator

$$\nabla = \begin{pmatrix} \partial_1 & 0 & 0 & 0 & \partial_3 & \partial_2 \\ 0 & \partial_2 & 0 & \partial_3 & 0 & \partial_1 \\ 0 & 0 & \partial_3 & \partial_2 & \partial_1 & 0 \end{pmatrix}$$

$\partial_1, \partial_2, \partial_3$:
partial derivatives
in the 3 spatial directions

Computational cost



$$\partial_{tt}\boldsymbol{u} = \rho^{-1}\left(\nabla\left[\boldsymbol{C}\nabla^{T}\boldsymbol{u}\right] + \boldsymbol{f}\right)$$

∂_1	6 * N3*N2*N1*(1+2*log2(N1)) +
∂_2	6 * N3*N2*N1*(1+2*log2(N2)) +
∂_3	6 * N3*N2*N1*(1+2*log2(N3)) +
Constants multiplication	6 * N3*N2*N1*12 =

Runtime: 6*N*(15+2*log2(N))

N=N3*N2*N1

Recasting 2nd order as 1st order

$$\begin{cases} \partial_{tt} \boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{F}(t,\boldsymbol{u}(\boldsymbol{x},t)) \\ \partial_{t} \boldsymbol{u}(\boldsymbol{x},0) = \dot{\boldsymbol{u}}_{\boldsymbol{0}} \\ \boldsymbol{u}(\boldsymbol{x},0) = \boldsymbol{u}_{\boldsymbol{0}} \end{cases}$$

Recasting 2nd order as 1st order

$$\begin{cases} \partial_{tt}\boldsymbol{u}(\boldsymbol{x},t)=\boldsymbol{F}(t,\boldsymbol{u}(\boldsymbol{x},t)) \\ \partial_{t}\boldsymbol{u}(\boldsymbol{x},0)=\dot{\boldsymbol{u}}_{\boldsymbol{0}} \\ \boldsymbol{u}(\boldsymbol{x},0)=\boldsymbol{u}_{\boldsymbol{0}} \end{cases}$$

$$\begin{cases} \partial_{t}\boldsymbol{u}(\boldsymbol{x},t)=\boldsymbol{v}(\boldsymbol{x},t) \\ \partial_{t}\boldsymbol{v}(\boldsymbol{x},t)=\boldsymbol{F}(t,\boldsymbol{u}(\boldsymbol{x},t)) \\ \boldsymbol{v}(\boldsymbol{x},0)=\dot{\boldsymbol{u}}_{\boldsymbol{0}} \\ \boldsymbol{u}(\boldsymbol{x},0)=\boldsymbol{u}_{\boldsymbol{0}} \end{cases}$$

Recasting 2nd order as 1st order

$$\begin{cases} \partial_{tt} \boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{F}(t,\boldsymbol{u}(\boldsymbol{x},t)) \\ \partial_t \boldsymbol{u}(\boldsymbol{x},0) = \dot{\boldsymbol{u}}_0 \\ \boldsymbol{u}(\boldsymbol{x},0) = \boldsymbol{u}_0 \end{cases}$$

$$\begin{cases} \partial_t \boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{v}(\boldsymbol{x},t) \\ \partial_t \boldsymbol{v}(\boldsymbol{x},t) = \boldsymbol{F}(t,\boldsymbol{u}(\boldsymbol{x},t)) \\ \boldsymbol{v}(\boldsymbol{x},0) = \dot{\boldsymbol{u}}_0 \\ \boldsymbol{u}(\boldsymbol{x},0) = \boldsymbol{u}_0 \end{cases}$$

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} \quad ; \quad \tilde{\boldsymbol{F}} = \begin{bmatrix} 1 \\ \boldsymbol{F} \end{bmatrix} \quad \rightarrow \quad \begin{cases} \partial_t \boldsymbol{Y}(\boldsymbol{x},t) = \tilde{\boldsymbol{F}}(\boldsymbol{x},t) \\ \boldsymbol{Y}(\boldsymbol{x},0) = \boldsymbol{Y}_0 \end{cases}$$

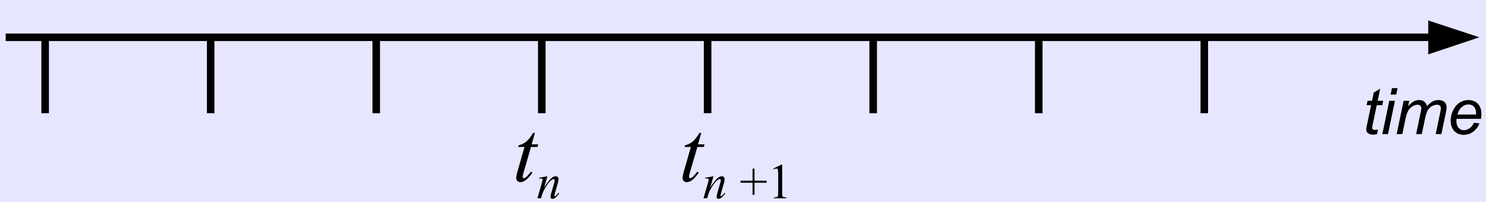
Explicit one step time integration

$$\begin{cases} \partial_t \boldsymbol{Y}(\boldsymbol{x}, t) = \boldsymbol{F}(\boldsymbol{x}, t) \\ \boldsymbol{Y}(\boldsymbol{x}, 0) = \boldsymbol{Y}_{\mathbf{0}} \end{cases}$$

Discretization of the time axis

$$t_{n+1} = t_n + \Delta t \qquad \forall n \in \{0, 1, \ldots \mathrm{N}\}$$

Δt respects some crieteria (not dicussed here)



Explicit one step time integration

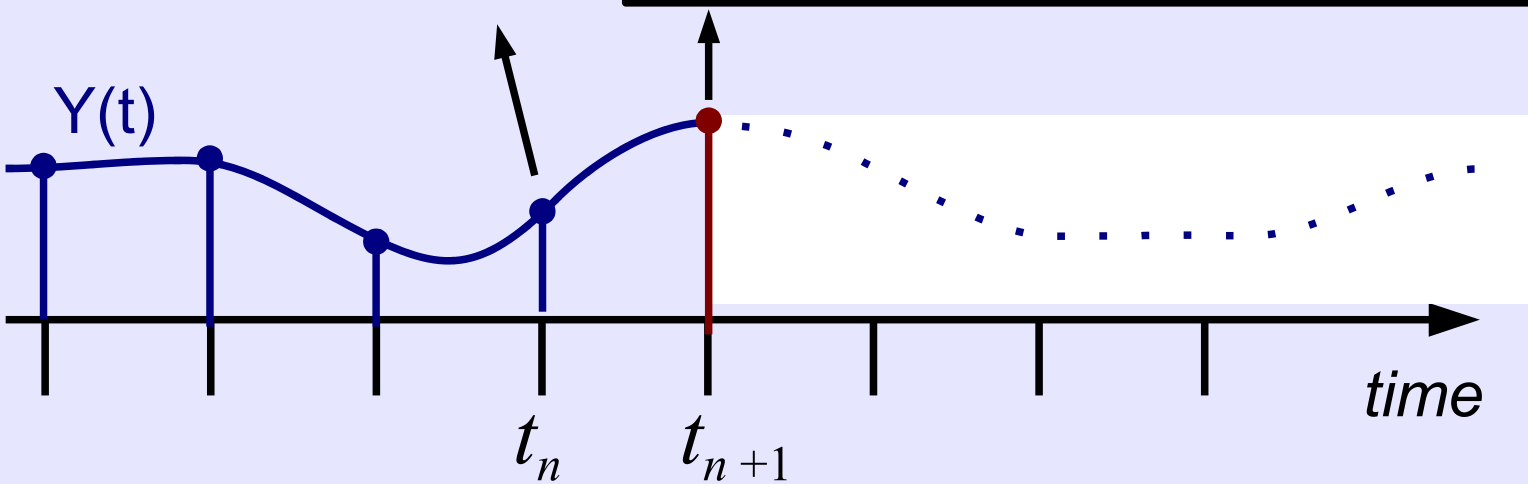
$$\begin{cases} \partial_t \boldsymbol{Y}(\boldsymbol{x}, t) = \boldsymbol{F}(\boldsymbol{x}, t) \\ \boldsymbol{Y}(\boldsymbol{x}, 0) = \boldsymbol{Y}_0 \end{cases}$$

Discretization of the time axis

$$t_{n+1} = t_n + \Delta t \qquad \forall n \in \{0, 1, \ldots N\}$$

Y_n

$$Y_{n+1} = Y_n + \Delta t \times \phi(t_n, Y_n; \Delta t, F)$$



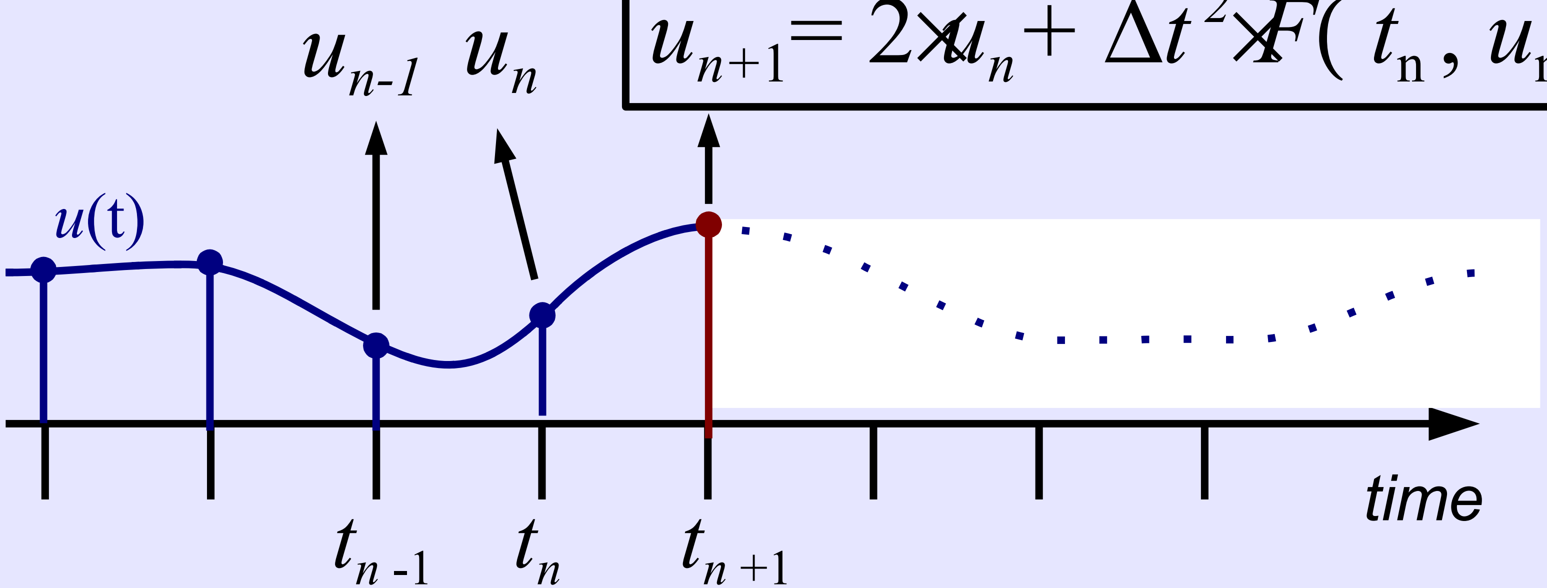
Explicit two step time integration

$$\begin{cases} \partial_{tt} \boldsymbol{u}(\boldsymbol{x}, t) = F(t, \boldsymbol{u}(\boldsymbol{x}, t)) \\ \boldsymbol{u}(\boldsymbol{x}, 0) = \boldsymbol{u}_{\boldsymbol{0}} \\ \boldsymbol{u}(\boldsymbol{x}, -\Delta t) = \boldsymbol{u}_{-\boldsymbol{1}} \end{cases}$$

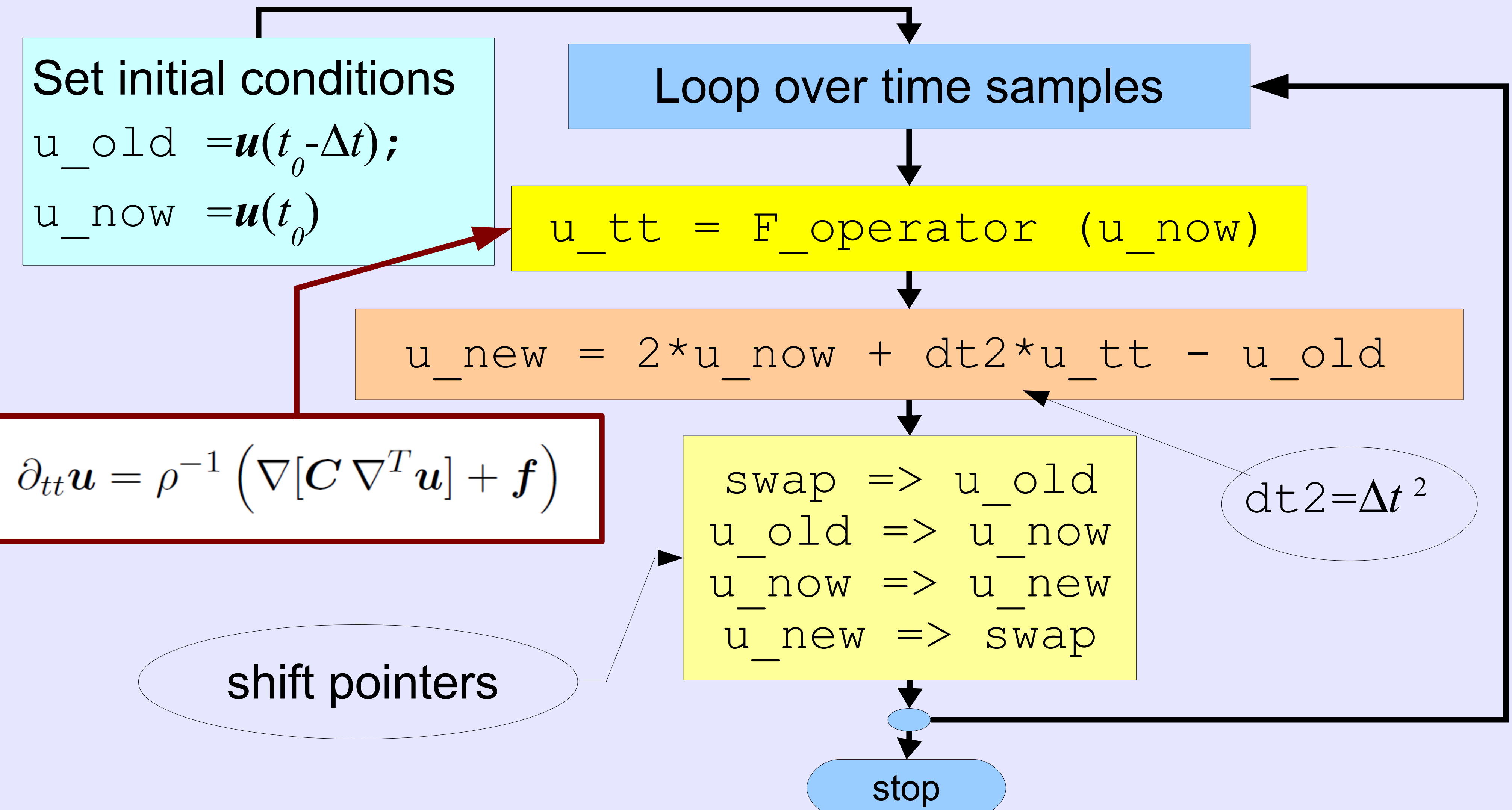
Discretization of the time axis

$$t_{n+1} = t_n + \Delta t \qquad \forall n \in \{0, 1, \ldots \mathrm{N}\}$$

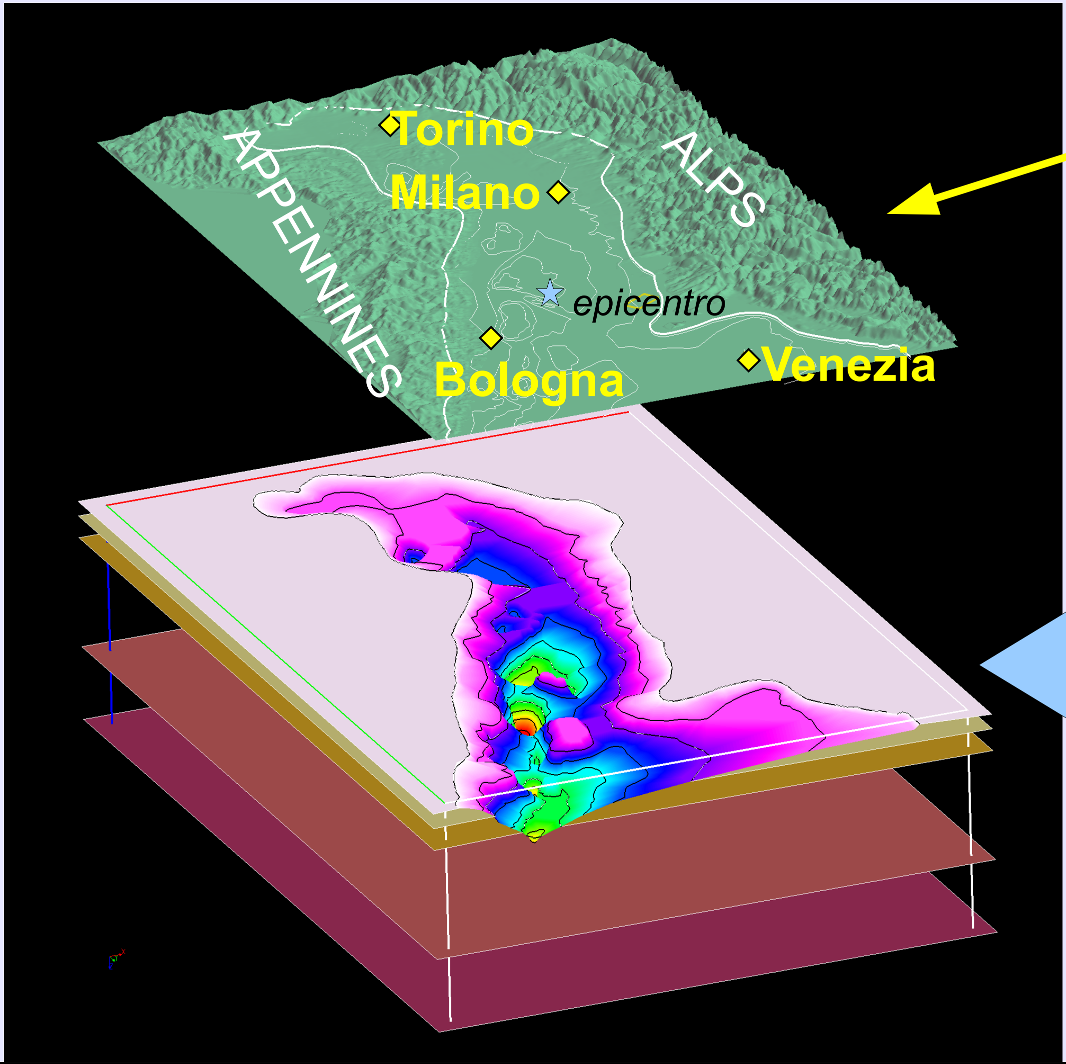
$$u_{n+1} = 2 \times u_n + \Delta t^2 \times F(t_\mathrm{n}, u_\mathrm{n}) - u_{n-1}$$

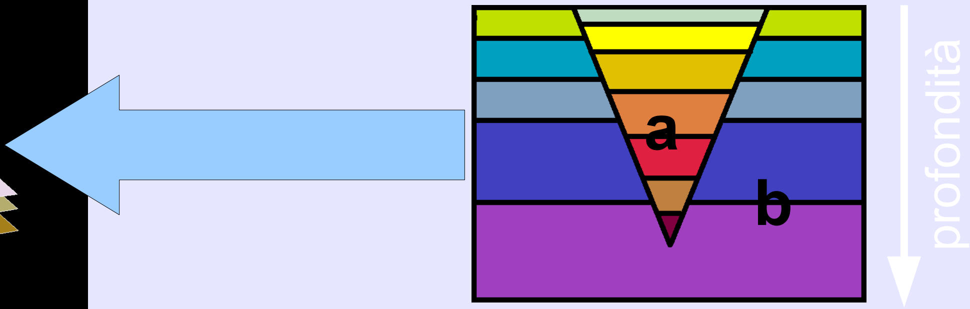


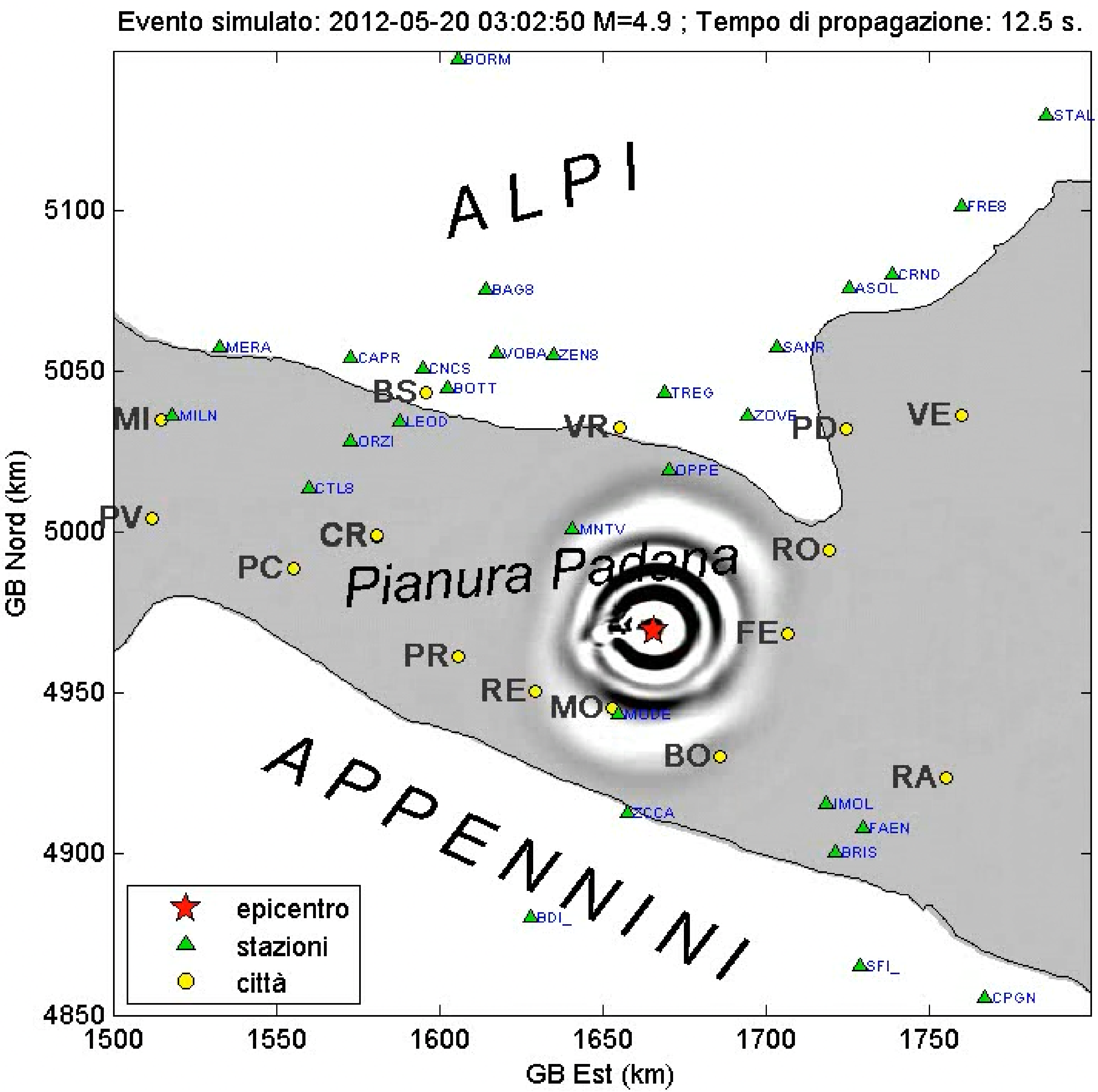
Explicit two step time integration



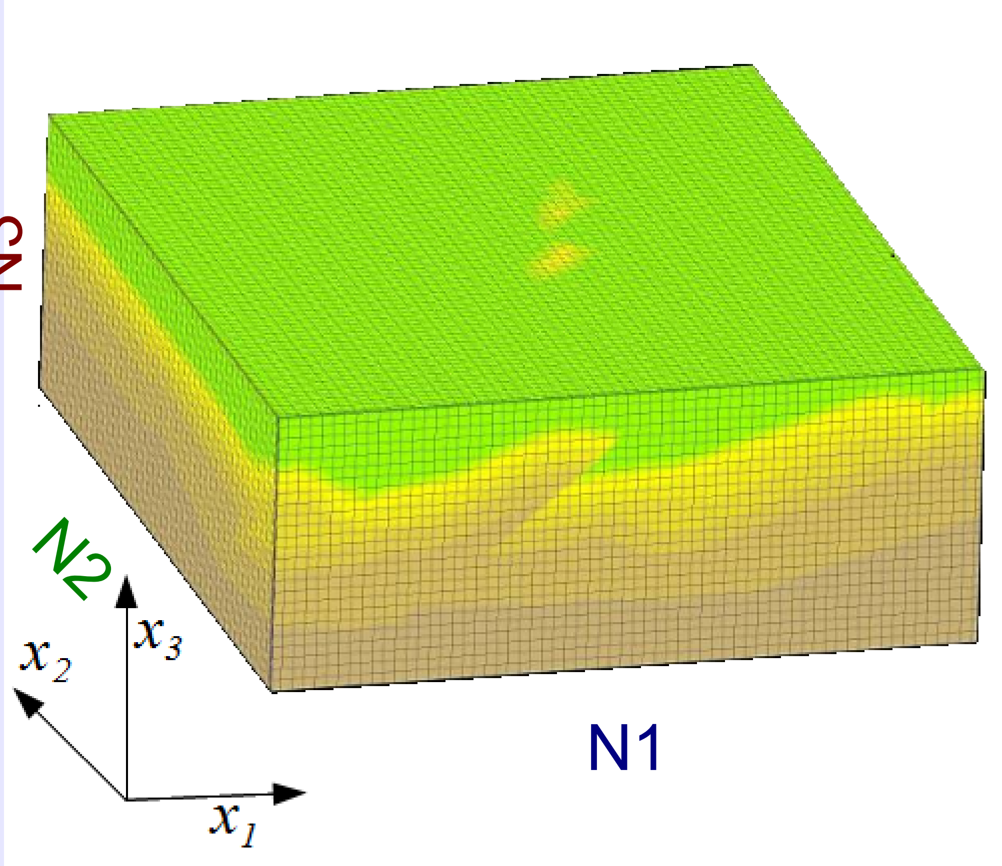
Simulation of earthquake ground motion






Evento simulato: 2012-05-20 03:02:50 M=4.9 ; Tempo di propagazione: 12.5 s.
ALPI
Pianura Padana
APPENNINI
MI
PV
BS
CR
PC
VR
PD
VE
RO
FE
PR
RE
MO
BO
RA
BORM
STAL
FRES
CRND
ASOL
BAGS
MERA
CAPR
CNCS
VOBA
ZENS
SANR
BOTT
TREG
MILN
LEOD
ZOVE
ORZI
OPPE
CTLS
MNTV
MODE
ZCCA
IMOL
FAEN
BRIS
BDI_
SFI_
CPGN
epicentro
stazioni
città
GB Nord (km)
GB Est (km)
5100
5050
5000
4950
4900
4850
1500
1550
1600
1650
1700
1750

Simulation of earthquake ground motion



Dimension of the problem (Po Plain):

N1=768; N2=768; N3=192;

N=N1*N2*N3=1132462082

$N \sim 2^{27}$

$N \sim 10^{9}$

$NT=60000=6*10^{4}$

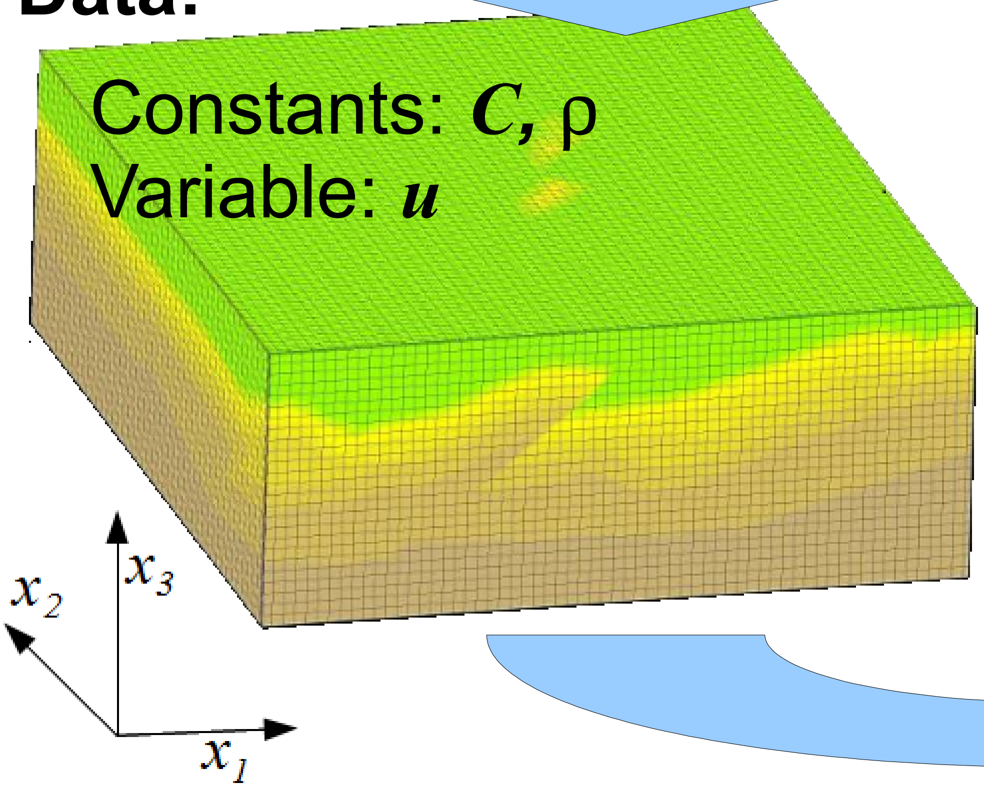
Runtime: NT*6*N*(15+2*log2(N))

$\sim 2.5 * 10^{17}$

Parallelization of the code

Data:

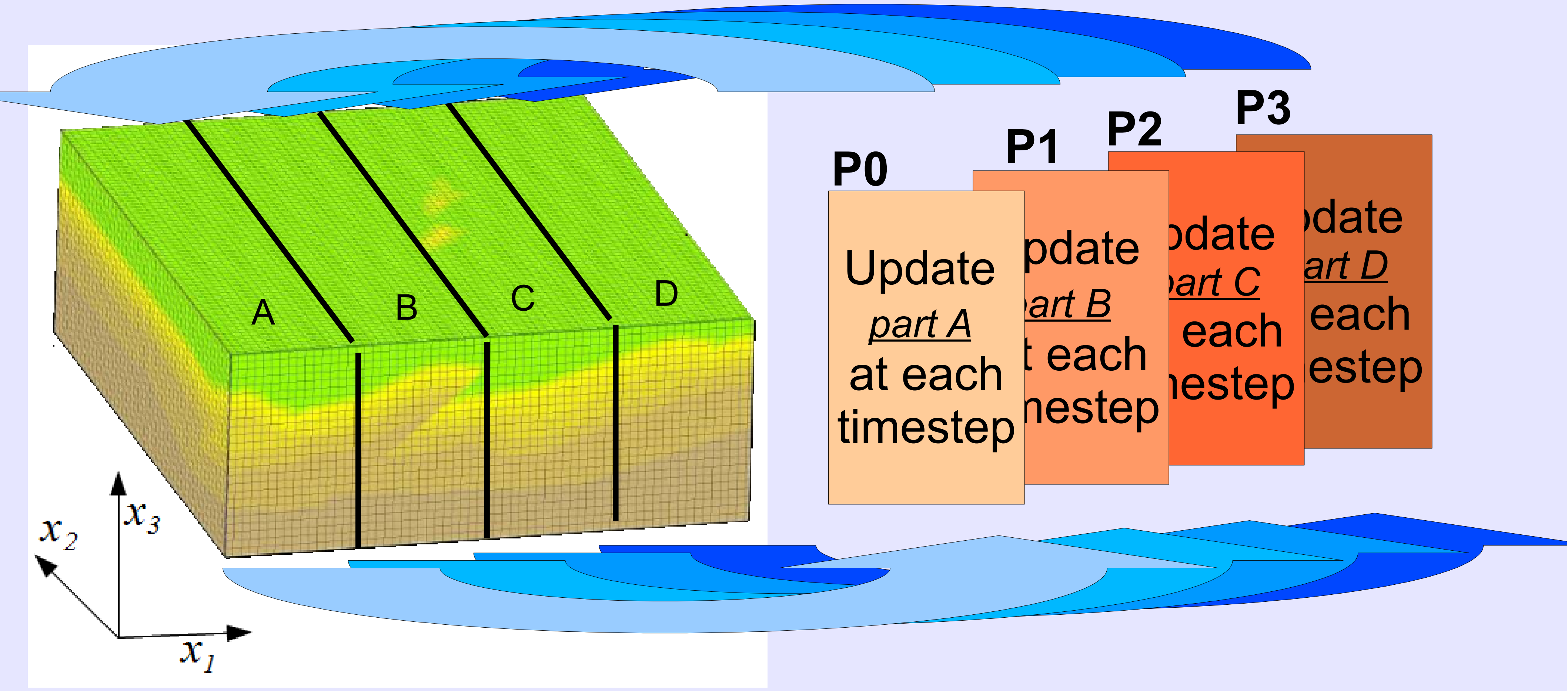
Constants: C, ρ
Variable: u



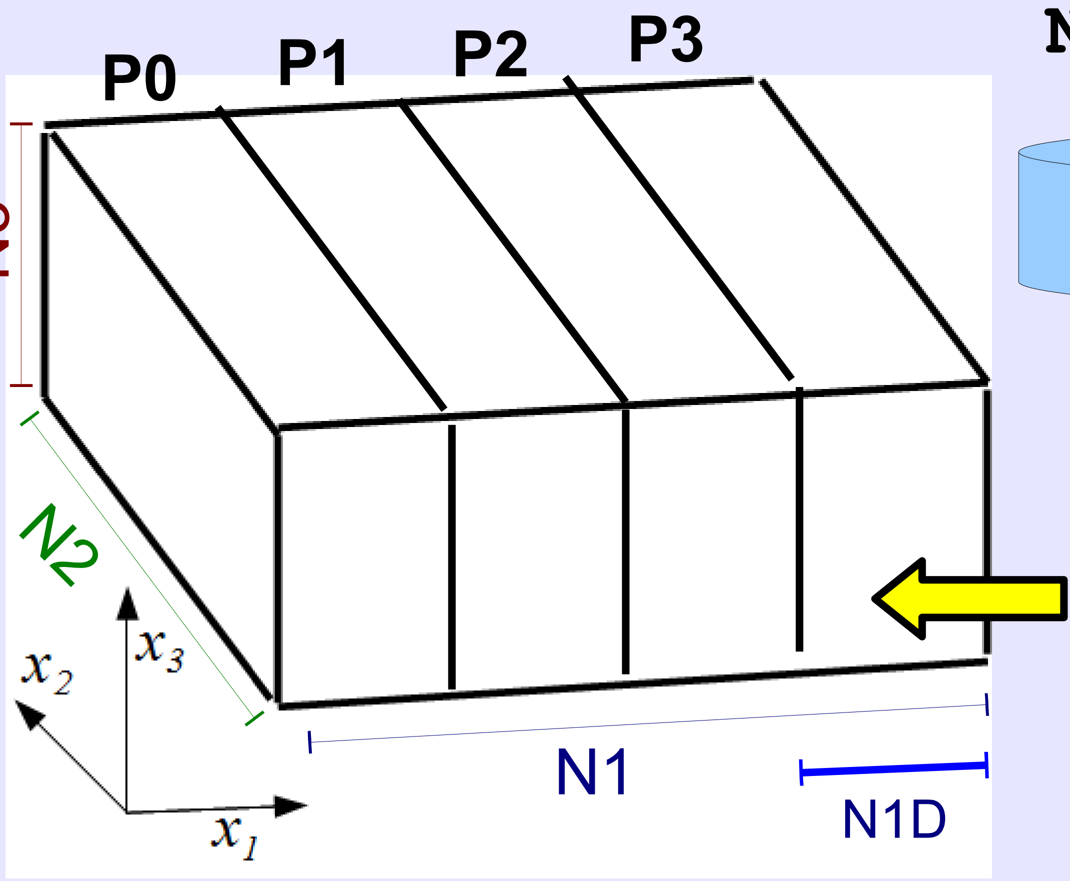
Task:

Update u
at each timestep

Parallelization of the code

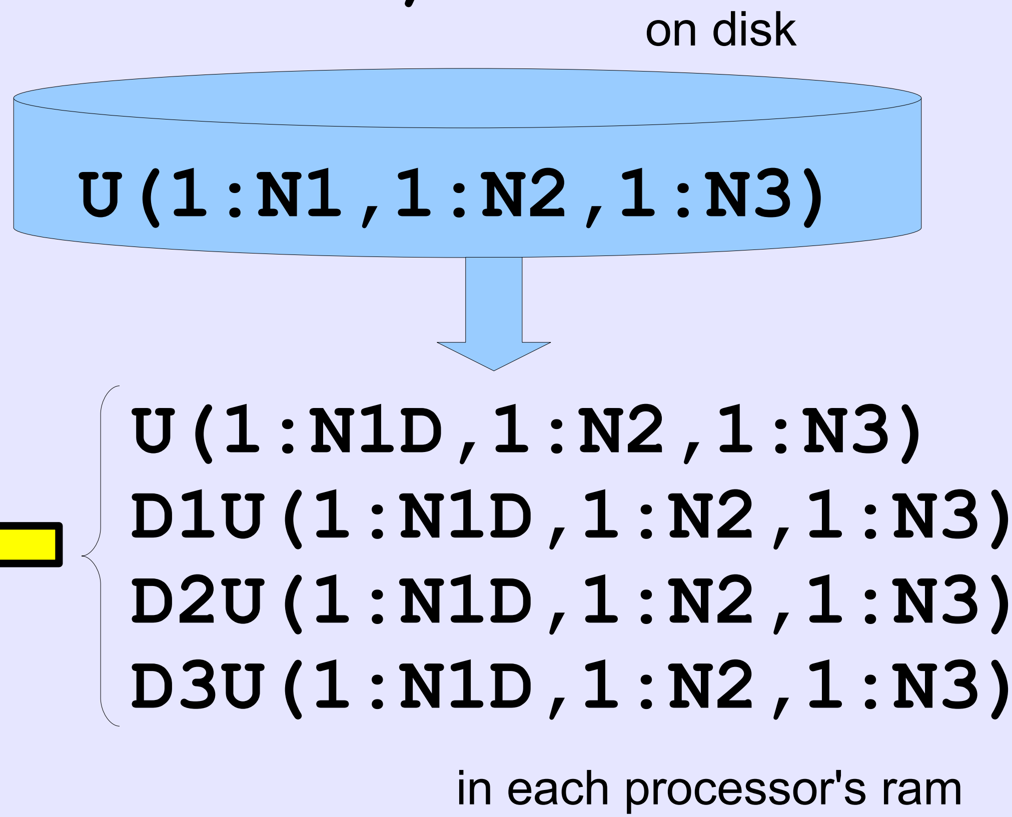


1D domain decomposition

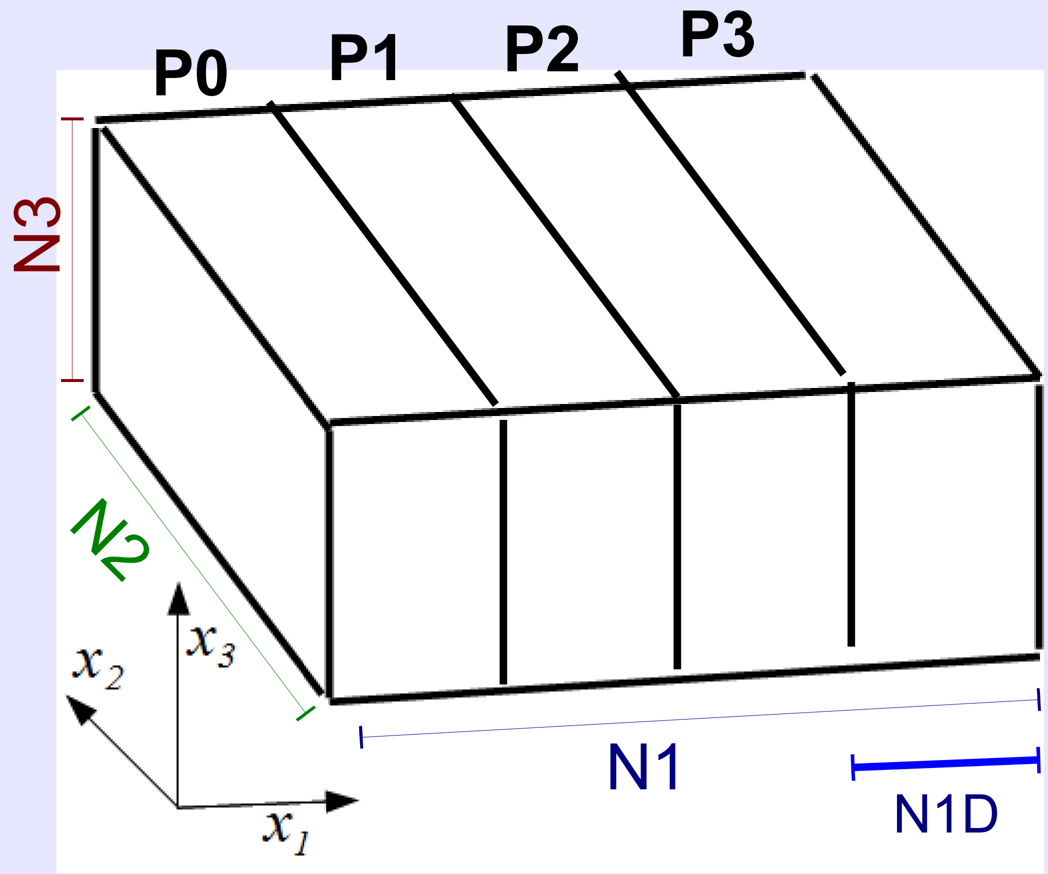


NP = number of processes

N1D = N1/NP



1D domain decomposition



NP = number of processes

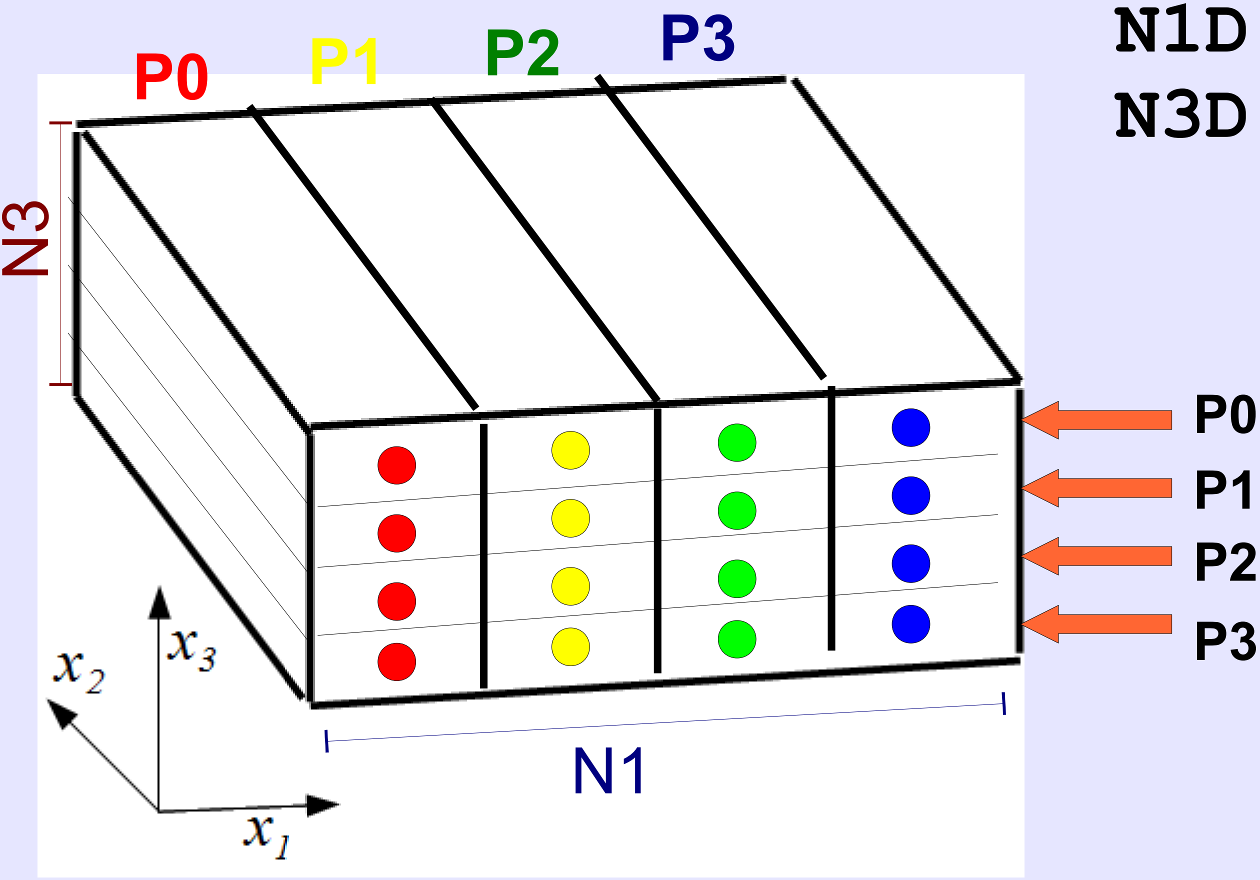
N1D = N1/NP

U(1:N1D,1:N2,1:N3)

∂_3 & ∂_2 embarrassingly parallel

∂_1 needs communication!

1D domain decomposition




```
NP   = number of processes
N1D  = N1/NP
N3D  = N3/NP
```

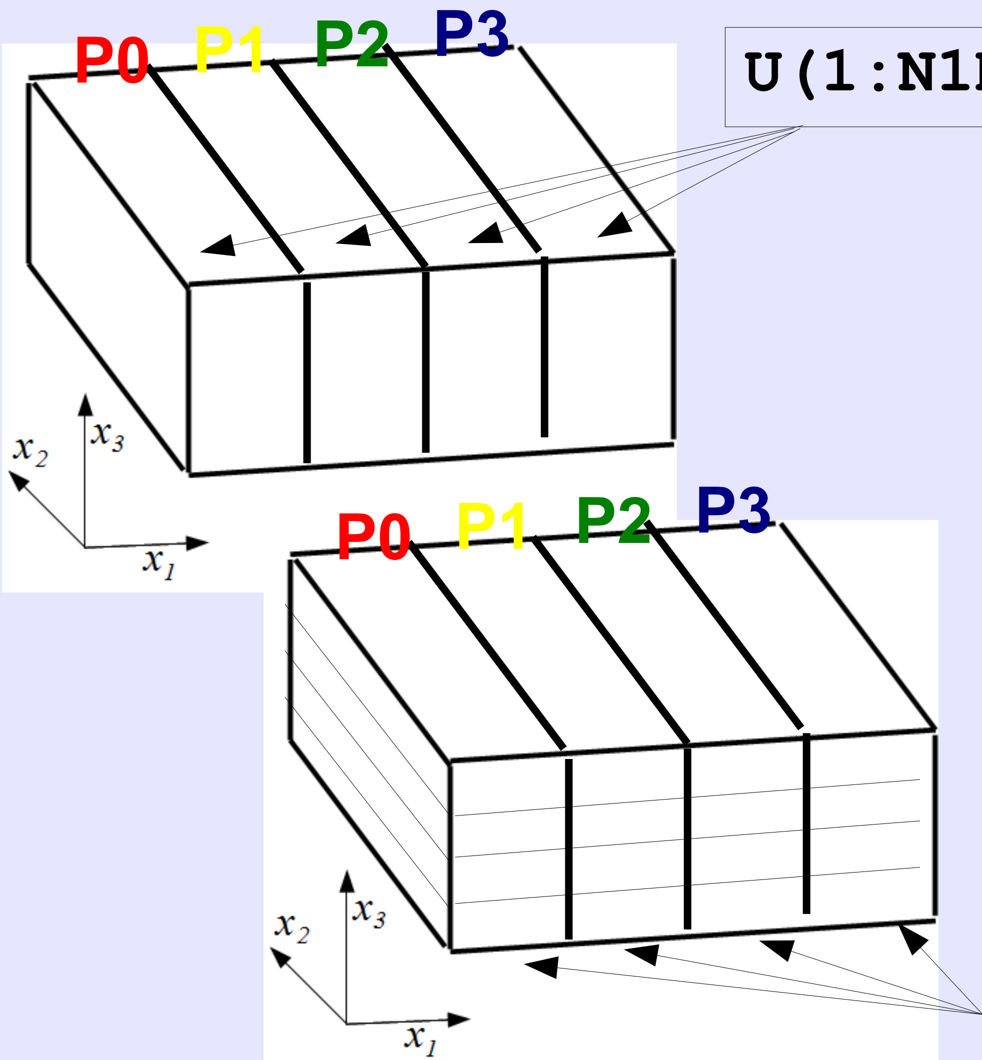
`U(1:N1D,1:N2,1:N3)`

suitable for ∂_3 & ∂_2

`U(1:N1,1:N2,1:N3D)`

suitable for ∂_1 (& ∂_2)

1D domain decomposition



`U(1:N1D,1:N2,1:N3)`

NP = number of processes

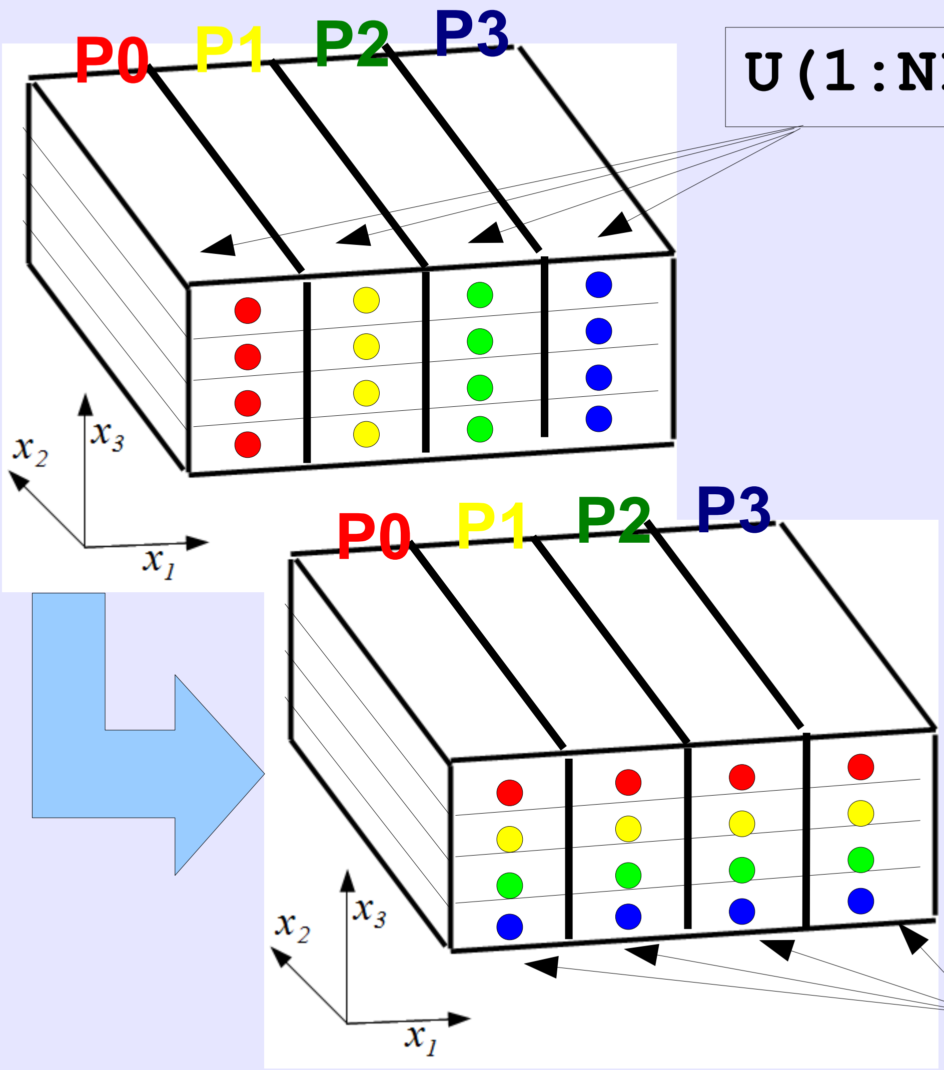
N1D = N1/NP

N3D = N3/NP

Container for data from other processes

`B(1:N1D,1:N2,1:N3D,1:NP)`

1D domain decomposition




```
U(1:N1D,1:N2,1:N3)
```

NP = number of processes

N1D = N1/NP

N3D = N3/NP

```
INTEGER :: NB=N1D*N2*N3D
!FROM U TO B
CALL MPI_ALLTOALL&
        (U,NB,MPI_REAL,&
         B,NB,MPI_REAL,&
        MPI_COMM_WORLD,IERR)
```

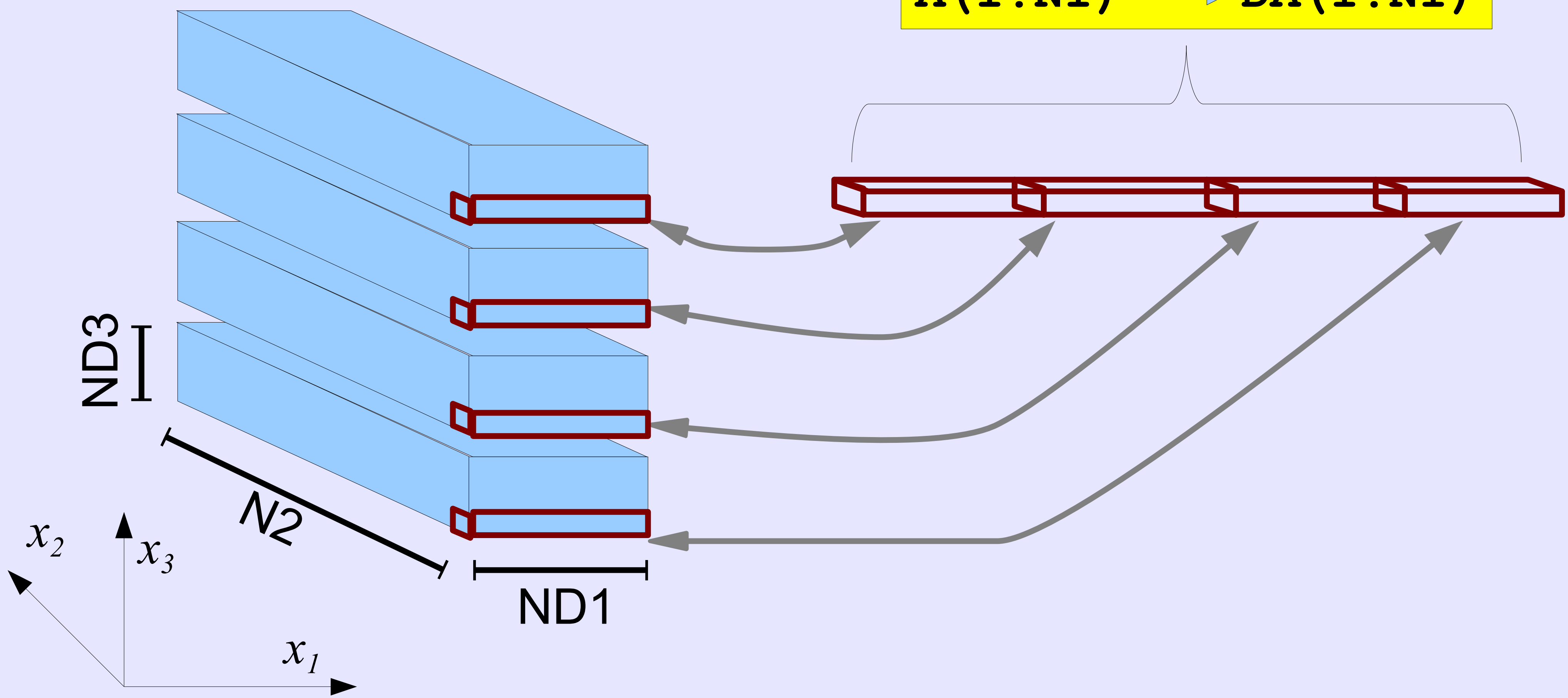
```
B(1:N1D,1:N2,1:N3D,1:NP)
```

1D domain decomposition

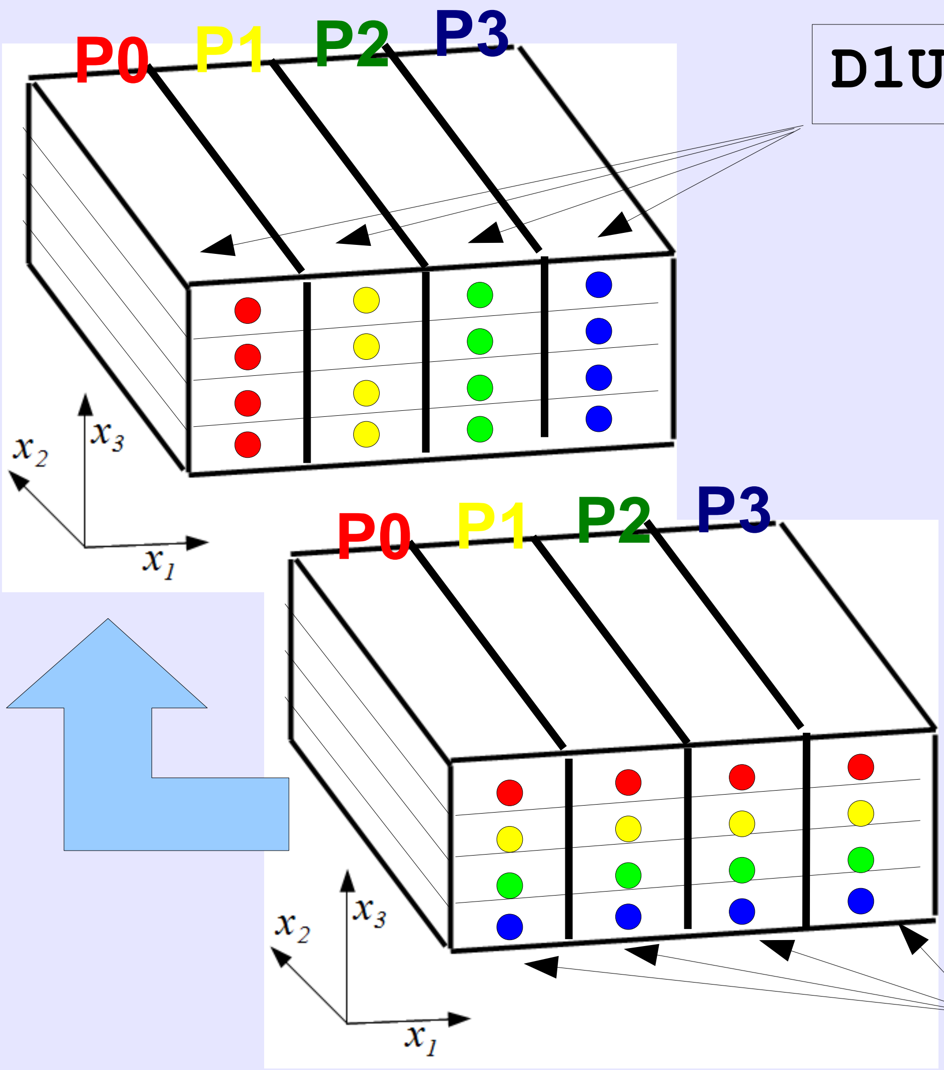
`B(1:ND1,1:N2,1:ND3,1:NP)`

Perform derivative “à la Fourier“

`A(1:N1)` → `DA(1:N1)`



1D domain decomposition



D1U(1:N1D,1:N2,1:N3)

NP = number of processes

N1D = N1/NP

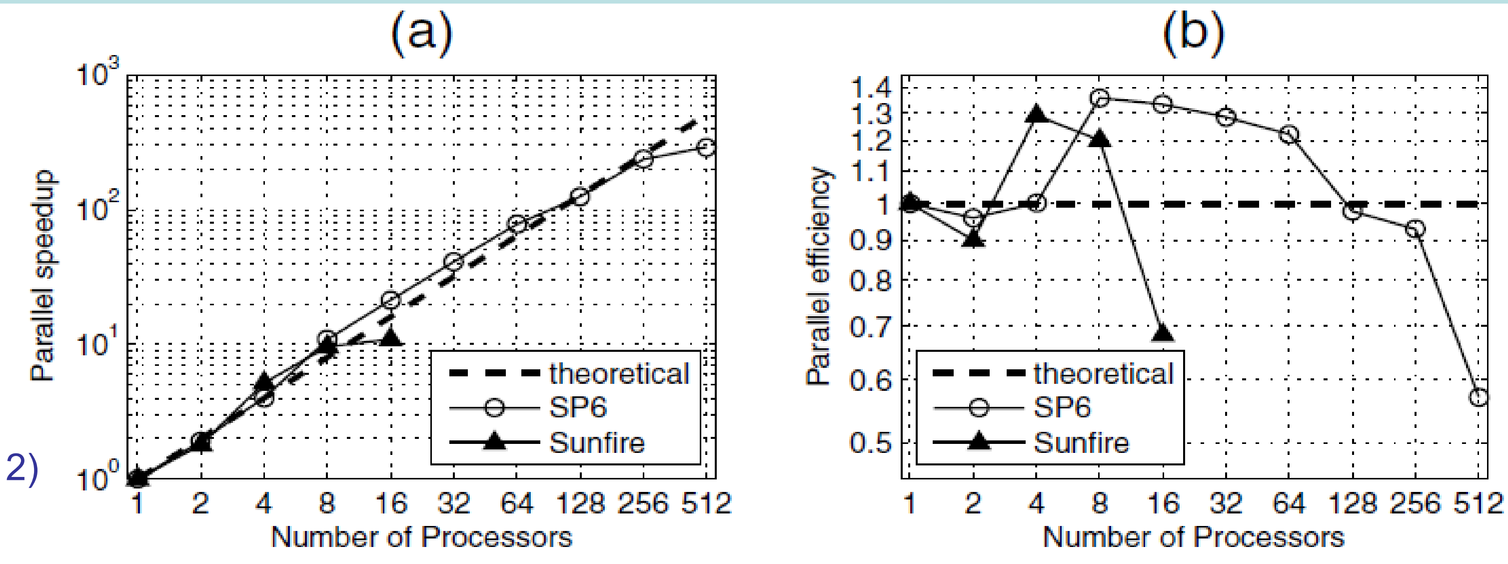
N3D = N3/NP

```
INTEGER :: NB=N1D*N2*N3D
!FROM U TO B
CALL MPI_ALLTOALL&
        (B,NB,MPI_REAL,&
         D1U,NB,MPI_REAL,&
        MPI_COMM_WORLD,IERR)
```

B(1:N1D,1:N2,1:N3D,1:NP)

Efficiency of 1D domain decomposition

strong scaling
constant problem size
(512 x 512 x 512)



Limitations of the 1D domain decomposition:

- All processes communicate with all processes
- Number of processes ≤ Number of samples along the shorter direction

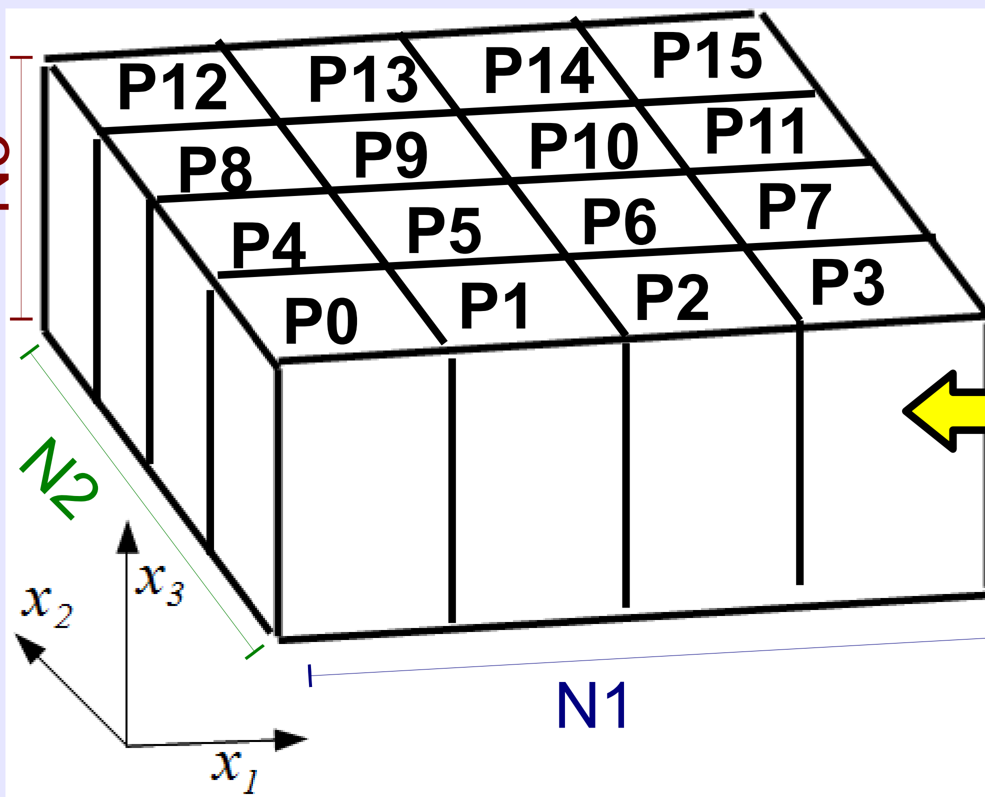
2D domain decomposition

number of processes:

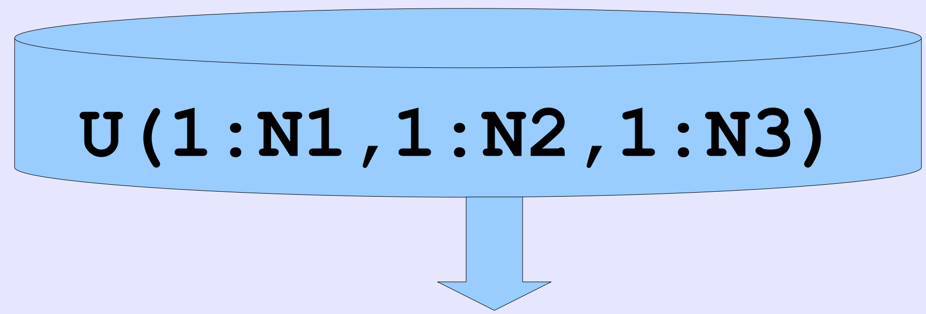
`NP = NPS*NPS   !NPS>0`

`N1D = N1/NPS`

`N2D = N2/NPS`



on disk



`U(1:N1D,1:N2D,1:N3)`

in each processor's ram

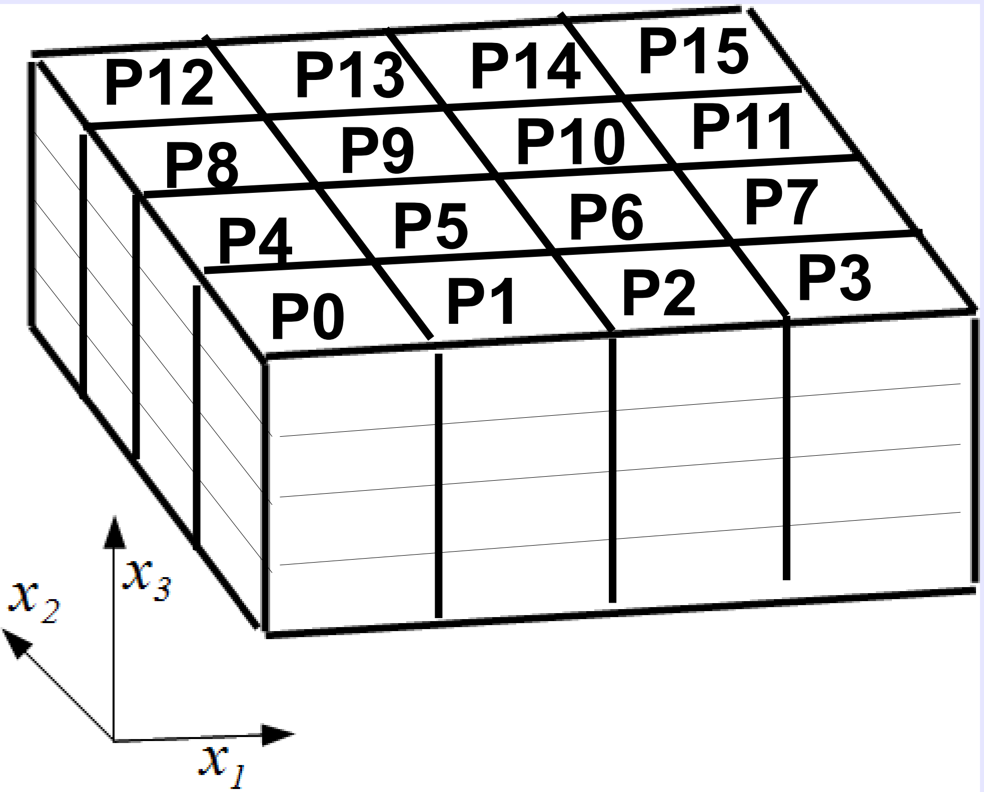
∂_3 embarrassingly parallel

∂_1 & ∂_2 need communication!

2D domain decomposition

number of processes:

```
NP = NPS*NPS   !NPS>0
```




```
N1D = N1/NPS
N2D = N2/NPS
N3D = N3/NPS
```

```
U(1:N1D,1:N2D,1:N3)
```

```
B(1:N1D,1:N2D,1:N3D,1:NPS)
```

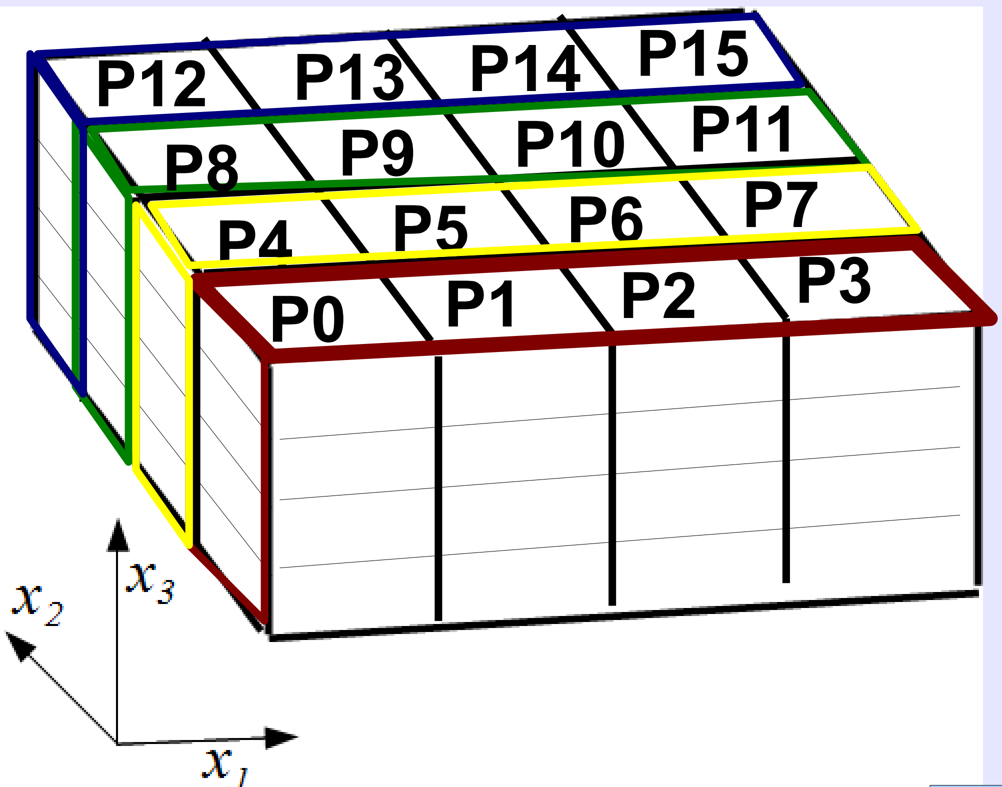
Container for data from other processes

2D domain decomposition

number of processes:

```
NP = NPS*NPS   !NPS>0
```

```
N1D = N1/NPS
N2D = N2/NPS
N3D = N3/NPS
```




```
U(1:N1D,1:N2D,1:N3)

B(1:N1D,1:N2D,1:N3D,1:NPS)

INTEGER :: NB=N1D*N2*N3D

!FROM U TO B ONLY FOR D1
CALL MPI_ALLTOALL&
        (U,NB,MPI_REAL,&
         B,NB,MPI_REAL,&
         COMM1,IERR)
```

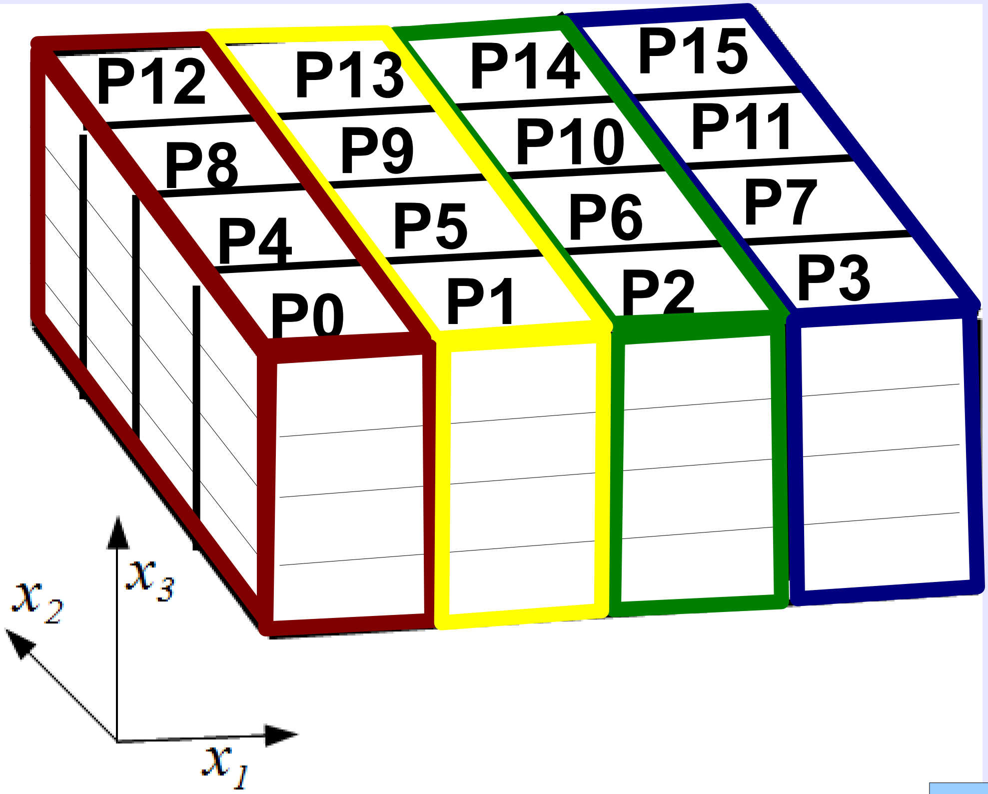
MPI_COMM_WORLD → DIFFERS! → COMM1

2D domain decomposition

number of processes:

```
NP = NPS*NPS   !NPS>0
```

```
N1D = N1/NPS
N2D = N2/NPS
N3D = N3/NPS
```

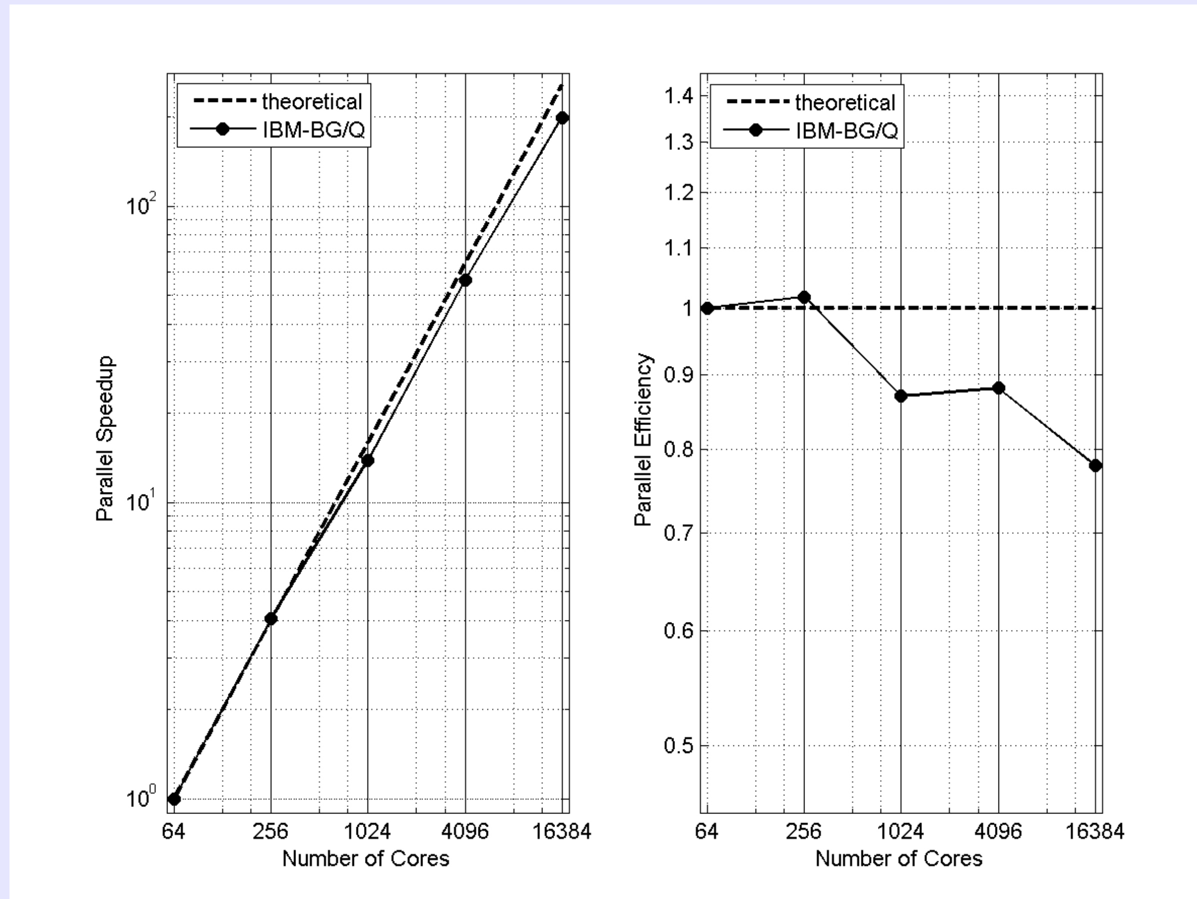



```
U(1:N1D,1:N2D,1:N3)

B(1:N1D,1:N2D,1:N3D,1:NPS)

INTEGER :: NB=N1D*N2*N3D

!FROM U TO B ONLY FOR D2
CALL MPI_ALLTOALL&
        (U,NB,MPI_REAL,&
         B,NB,MPI_REAL,&
         COMM2,IERR)
```

MPI_COMM_WORLD → DIFFERS! → COMM2

Efficiency of 2D domain decomposition

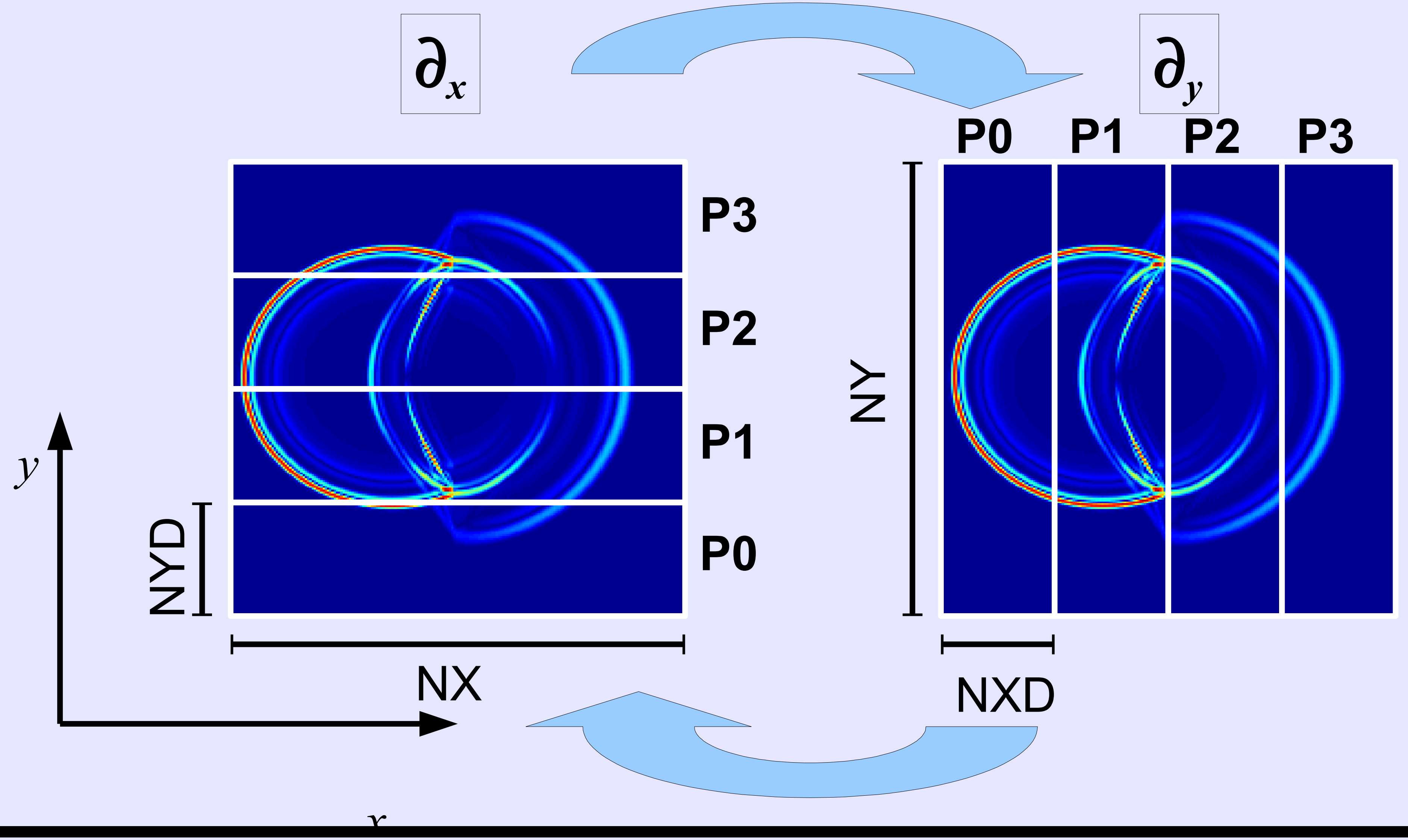
IBM BG/Q at CINECA



Strong scaling
constant
problem size
(2048 x 2048 x 512)

2D domain decomposition allows to run FPSM on massively parallel computers!

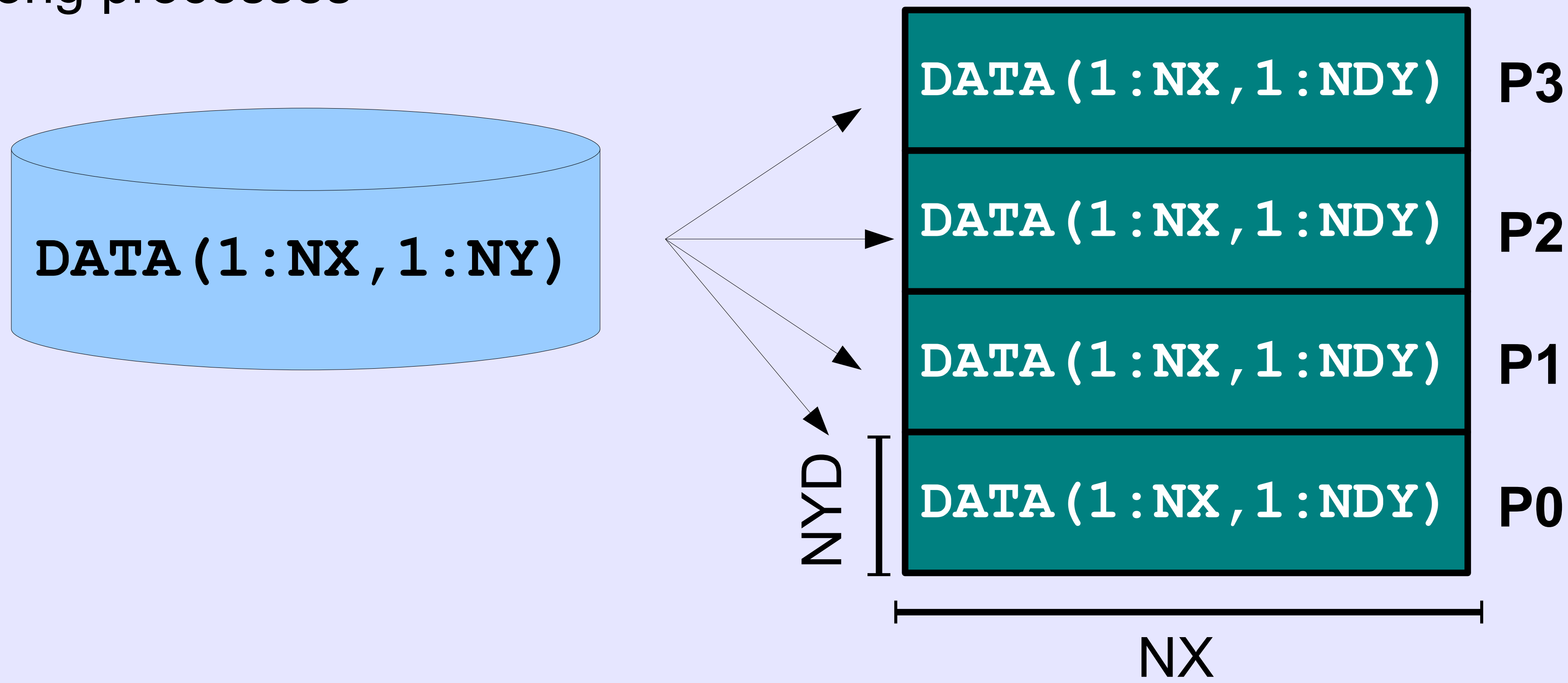
Lab-Session: implementing 2D wave simulation with MPI



Lab-Session: implementing 2D wave simulation with MPI

Exercise 1: Space domain partitioning

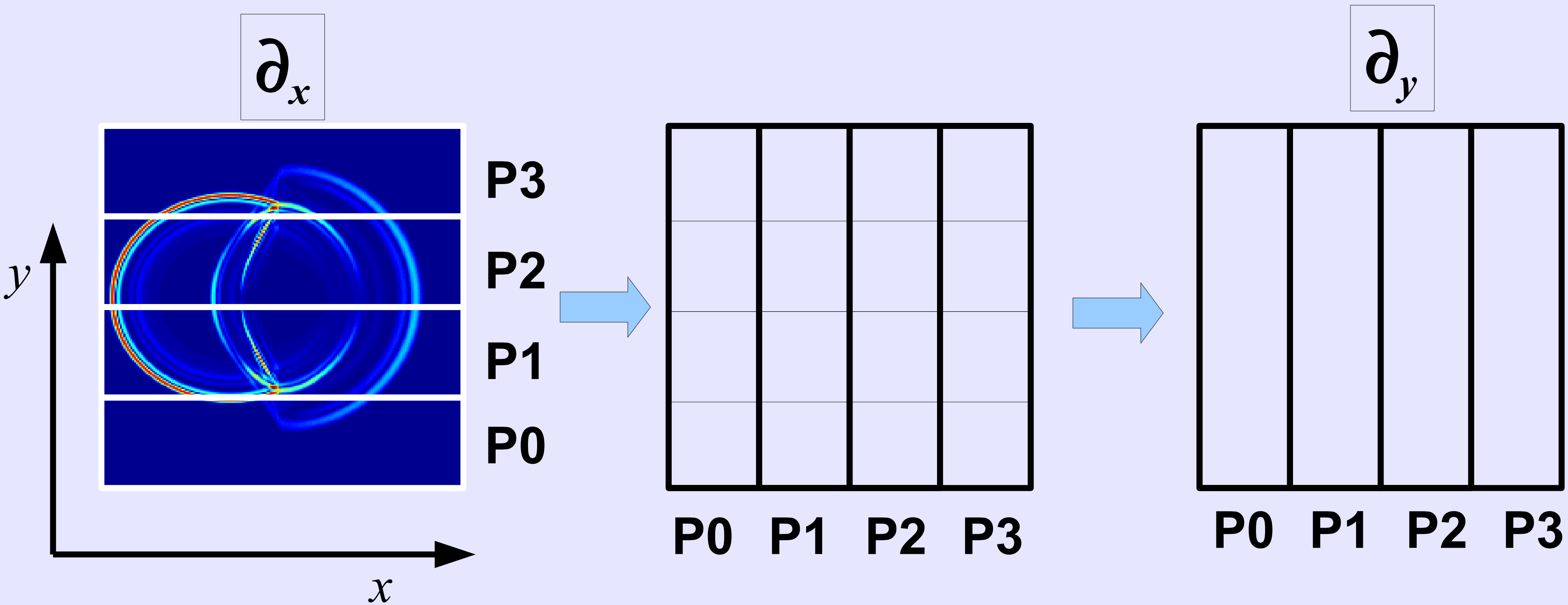
distribute input data (elastic parameters, initial values for u) among processes



Lab-Session: implementing 2D wave simulation with MPI

Exercise 2: Partitioning rearrangement

Rearrange partitioning from ∂_x to ∂_y configuration



Thank you!

pklin@inogs.it