UNIVERSITÄT GRAZ
UNIVERSITY OF GRAZ

Basics of Pencil Code & some philosophy

(Pencil Code School, Geneva/CERN, 20th of September 2025)

Philippe-A. Bourdin
Philippe.Bourdin@uni-graz.at

Overview:

* Magneto-Hydrodynamics (MHD)
* Requirements for PC
* Philosophy of the PC
* Getting and Installing PC
* OpenMP and MPI
* Choosing a sample
* Selecting modules and processor layout
* Compiling PC
* Efficient computing strategies
* Changing parameters
* Setting up initial condition (“start”)
* Propagate an existing state (“run”)
* Boundary conditions

Introduction to MHD

Basic equations:

$$c\vec{\nabla}\times\vec{B}=4\pi\vec{j} \qquad c\vec{\nabla}\times\vec{E}=-\frac{\partial\vec{B}}{\partial t} \qquad \vec{\nabla}\cdot\vec{B}=0$$ (Maxwell's equations)

Basic equations:

$c\vec{\nabla}\times\vec{B}=4\pi\vec{j}$ $\qquad$ $c\vec{\nabla}\times\vec{E}=-\frac{\partial\vec{B}}{\partial t}$ $\qquad$ $\vec{\nabla}\cdot\vec{B}=0$ (Maxwell's equations)

$\lambda\vec{j}=\vec{E}+\frac{1}{c}\vec{v}\times\vec{B}$ $\qquad$ λ : elect. resistivity, $\frac{1}{\lambda}=\sigma$: conductivity (Ohm's law)

Basic equations:

$$c\vec{\nabla}\times\vec{B}=4\pi\vec{j} \qquad c\vec{\nabla}\times\vec{E}=-\frac{\partial\vec{B}}{\partial t} \qquad \vec{\nabla}\cdot\vec{B}=0$$ (Maxwell's equations)

$$\lambda\vec{j}=\vec{E}+\frac{1}{c}\vec{v}\times\vec{B}$$ λ : elect. resistivity, $\frac{1}{\lambda}=\sigma$: conductivity (Ohm's law)

$$\rho\frac{\partial\vec{v}}{\partial t}+\rho\left(\vec{v}\cdot\vec{\nabla}\right)\vec{v}=-\vec{\nabla}p+\frac{1}{c}\vec{j}\times\vec{B}+\vec{F}_{ext}$$ (equation of motion)

Basic equations:

$c\vec{\nabla}\times\vec{B}=4\pi\vec{j}$ $\qquad$ $c\vec{\nabla}\times\vec{E}=-\frac{\partial\vec{B}}{\partial t}$ $\qquad$ $\vec{\nabla}\cdot\vec{B}=0$ (Maxwell's equations)

$\lambda\vec{j}=\vec{E}+\frac{1}{c}\vec{v}\times\vec{B}$ $\qquad$ λ : elect. resistivity, $\frac{1}{\lambda}=\sigma$: conductivity (Ohm's law)

$\rho\frac{\partial\vec{v}}{\partial t}+\rho\left(\vec{v}\cdot\vec{\nabla}\right)\vec{v}=-\vec{\nabla}p+\frac{1}{c}\vec{j}\times\vec{B}+\vec{F}_{ext}$ (equation of motion)

$\frac{\partial\rho}{\partial t}+\vec{\nabla}\cdot\left(\rho\vec{v}\right)=0$ (continuity equation)

Basic equations:

$c\vec{\nabla}\times\vec{B}=4\pi\vec{j}$ $\qquad$ $c\vec{\nabla}\times\vec{E}=-\frac{\partial\vec{B}}{\partial t}$ $\qquad$ $\vec{\nabla}\cdot\vec{B}=0$ (Maxwell's equations)

$\lambda\vec{j}=\vec{E}+\frac{1}{c}\vec{v}\times\vec{B}$ $\qquad$ λ : elect. resistivity, $\frac{1}{\lambda}=\sigma$: conductivity (Ohm's law)

$\rho\frac{\partial\vec{v}}{\partial t}+\rho\left(\vec{v}\cdot\vec{\nabla}\right)\vec{v}=-\vec{\nabla}p+\frac{1}{c}\vec{j}\times\vec{B}+\vec{F}_{ext}$ (equation of motion)

$\frac{\partial\rho}{\partial t}+\vec{\nabla}\cdot\left(\rho\vec{v}\right)=0$ (continuity equation)

$p=f\left(\rho,T\right)\ \Rightarrow\ p_{\text{ideal gas}}=R\rho T$ (equation of state)

Basic equations:

$c\vec{\nabla}\times\vec{B}=4\pi\vec{j}$ $\qquad$ $c\vec{\nabla}\times\vec{E}=-\frac{\partial\vec{B}}{\partial t}$ $\qquad$ $\vec{\nabla}\cdot\vec{B}=0$ (Maxwell's equations)

$\lambda\vec{j}=\vec{E}+\frac{1}{c}\vec{v}\times\vec{B}$ $\qquad$ λ : elect. resistivity, $\frac{1}{\lambda}=\sigma$: conductivity (Ohm's law)

$\rho\frac{\partial\vec{v}}{\partial t}+\rho\left(\vec{v}\cdot\vec{\nabla}\right)\vec{v}=-\vec{\nabla}p+\frac{1}{c}\vec{j}\times\vec{B}+\vec{F}_{ext}$ (equation of motion)

$\frac{\partial\rho}{\partial t}+\vec{\nabla}\cdot\left(\rho\vec{v}\right)=0$ (continuity equation)

$p=f\left(\rho,T\right)\ \Rightarrow\ p_{\text{ideal gas}}=R\rho T$ (equation of state)

Basic MHD assumptions:

* $v\ll c$: non-relativistic, quasi-stationary state change, no EM waves
* $\sigma\gg 1$: high conductivity, quasi-neutral plasma, E determined by $\partial B/\partial t$

What follows from MHD assumptions?:

$$c\nabla\times E=-\frac{\partial B}{\partial t}\ \Rightarrow\ c\frac{E}{L}\approx\frac{B}{T}\Rightarrow\frac{E}{B}=\frac{1}{c}\frac{L}{T}\approx\frac{v}{c}\ll 1$$ (*E* less important than B)

$$\frac{\partial E/\partial t}{c\nabla\times B}\approx\frac{E/T}{cB/L}\approx\frac{E\,v}{B\,c}\approx\frac{v^2}{c^2}\ll 1$$ (no displacement current)

$$\nabla\cdot E=4\,\pi\rho_e$$ (charge density)

What follows from MHD assumptions?:

$$c\nabla\times E=-\frac{\partial B}{\partial t}\ \Rightarrow\ c\frac{E}{L}\approx\frac{B}{T}\Rightarrow\frac{E}{B}=\frac{1}{c}\frac{L}{T}\approx\frac{v}{c}\ll 1$$ (*E* less important than B)

$$\frac{\partial E/\partial t}{c\nabla\times B}\approx\frac{E/T}{cB/L}\approx\frac{E\,v}{B\,c}\approx\frac{v^2}{c^2}\ll 1$$ (no displacement current)

$$\nabla\cdot E=4\,\pi\rho_e$$ (charge density)

Simplifications of MHD:

$\vec{v}$ given => magneto-kinematics

What follows from MHD assumptions?:

$$c\nabla\times E=-\frac{\partial B}{\partial t}\ \Rightarrow\ c\frac{E}{L}\approx\frac{B}{T}\Rightarrow\frac{E}{B}=\frac{1}{c}\frac{L}{T}\approx\frac{v}{c}\ll 1$$ (*E* less important than B)

$$\frac{\partial E/\partial t}{c\nabla\times B}\approx\frac{E/T}{cB/L}\approx\frac{E\,v}{B\,c}\approx\frac{v^2}{c^2}\ll 1$$ (no displacement current)

$$\nabla\cdot E=4\,\pi\rho_e$$ (charge density)

Simplifications of MHD:

$\vec{v}$ given => magneto-kinematics

$\vec{v}=0$ => magneto-hydrostatics

What follows from MHD assumptions?:

$$c\nabla\times E=-\frac{\partial B}{\partial t}\ \Rightarrow\ c\frac{E}{L}\approx\frac{B}{T}\Rightarrow\frac{E}{B}=\frac{1}{c}\frac{L}{T}\approx\frac{v}{c}\ll 1$$ (*E* less important than B)

$$\frac{\partial E/\partial t}{c\nabla\times B}\approx\frac{E/T}{cB/L}\approx\frac{E\,v}{B\,c}\approx\frac{v^2}{c^2}\ll 1$$ (no displacement current)

$$\nabla\cdot E=4\pi\rho_e$$ (charge density)

Simplifications of MHD:

$\vec{v}$ given => magneto-kinematics

$\vec{v}=0$ => magneto-hydrostatics

$\partial/\partial t=0$ => stationarity

What follows from MHD assumptions?:

$$c\nabla\times E=-\frac{\partial B}{\partial t}\ \Rightarrow\ c\frac{E}{L}\approx\frac{B}{T}\Rightarrow\frac{E}{B}=\frac{1}{c}\frac{L}{T}\approx\frac{v}{c}\ll 1$$ (*E* less important than B)

$$\frac{\partial E/\partial t}{c\nabla\times B}\approx\frac{E/T}{cB/L}\approx\frac{E\,v}{B\,c}\approx\frac{v^2}{c^2}\ll 1$$ (no displacement current)

$$\nabla\cdot E=4\pi\rho_e$$ (charge density)

Simplifications of MHD:

$\vec{v}$ given => magneto-kinematics

$\vec{v}=0$ => magneto-hydrostatics

$\partial/\partial t=0$ => stationarity

linearized equations => wave propagation, stability

Induction Equation

Example – evolution of ***B*** under influence of given ***v*:**

$$\frac{\partial \vec{B}}{\partial t} = c\vec{\nabla}\times\vec{E} = -c\vec{\nabla}\times\left(\lambda\vec{j} - \frac{1}{c}\vec{v}\times\vec{B}\right) = -c\vec{\nabla}\times\left(\frac{\lambda c}{4\pi}\vec{\nabla}\times\vec{B} - \frac{1}{c}\vec{v}\times\vec{B}\right)$$

$$\Rightarrow \frac{\partial \vec{B}}{\partial t} = \vec{\nabla}\times\left(\vec{v}\times\vec{B}\right) - \vec{\nabla}\times\left(\frac{\lambda c^2}{4\pi}\vec{\nabla}\times\vec{B}\right) = \vec{\nabla}\times\left(\vec{v}\times\vec{B}\right) - \eta\vec{\nabla}\times\vec{\nabla}\times\vec{B}$$

with $\eta = \frac{\lambda c^2}{4\pi} = \frac{1}{\mu_0 \sigma} = \text{const}.$ (magnetic diffusivity)

Induction Equation

Example – evolution of ***B*** under influence of given ***v*:**

$$\frac{\partial \vec{B}}{\partial t} = c\,\vec{\nabla}\times\vec{E} = -c\,\vec{\nabla}\times\left(\lambda\,\vec{j} - \frac{1}{c}\,\vec{v}\times\vec{B}\right) = -c\,\vec{\nabla}\times\left(\frac{\lambda\,c}{4\,\pi}\,\vec{\nabla}\times\vec{B} - \frac{1}{c}\,\vec{v}\times\vec{B}\right)$$

$$\Rightarrow \frac{\partial \vec{B}}{\partial t} = \vec{\nabla}\times\left(\vec{v}\times\vec{B}\right) - \vec{\nabla}\times\left(\frac{\lambda\,c^2}{4\,\pi}\,\vec{\nabla}\times\vec{B}\right) = \vec{\nabla}\times\left(\vec{v}\times\vec{B}\right) - \eta\,\vec{\nabla}\times\vec{\nabla}\times\vec{B}$$

with $\eta = \frac{\lambda\,c^2}{4\,\pi} = \frac{1}{\mu_0\,\sigma} = \text{const}.$ (magnetic diffusivity)

$$\Rightarrow \frac{\partial \vec{B}}{\partial t} = \vec{v}\left(\vec{\nabla}\cdot\vec{B}\right) - \vec{B}\left(\vec{\nabla}\cdot\vec{v}\right) + \left(\vec{B}\cdot\vec{\nabla}\right)\vec{v} - \left(\vec{v}\cdot\vec{\nabla}\right)\vec{B} - \eta\,\vec{\nabla}\times\vec{\nabla}\times\vec{B}$$

induction = 0 – expansion/contraction + shear/stretching – advection – diffusion

Introduction to HD

Basic equations:

* Take MHD equation set and remove all electro-magnetic terms from it...

Basic equations:

* Take MHD equation set and remove all electro-magnetic terms from it...

=> It remains:

* momentum conservation

* continuity equation

* equation of state

Basic equations:

* Take MHD equation set and remove all electro-magnetic terms from it...

=> It remains:

* momentum conservation $\rho\frac{\partial \vec{v}}{\partial t}+\rho\left(\vec{v}\cdot\vec{\nabla}\right)\vec{v}=-\vec{\nabla}\, p+\vec{F}_{ext}$

* continuity equation $\frac{\partial \rho}{\partial t}+\vec{\nabla}\cdot\left(\rho\vec{v}\right)=0$

* equation of state $p=f\left(\rho,T\right)\ \Rightarrow\ p_{\text{ideal gas}}=R\rho T$

Basic equations:

* Take MHD equation set and remove all electro-magnetic terms from it...

=> It remains:

* momentum conservation $\rho\frac{\partial \vec{v}}{\partial t}+\rho\left(\vec{v}\cdot\vec{\nabla}\right)\vec{v}=-\vec{\nabla}\, p+\vec{F}_{ext}$

* continuity equation $\frac{\partial \rho}{\partial t}+\vec{\nabla}\cdot\left(\rho\vec{v}\right)=0$

* equation of state $p=f\left(\rho,T\right)\ \Rightarrow\ \ p_{\text{ideal gas}}=R\rho T$

* energy balance, if there is viscosity – or for compressible fluids...
  (see also Navier-Stokes equations for viscous media)

Requirements

Requirements

Technical requirements:

* Laptop
* Windows or MAC users: VirtualBox or VMware (or dual boot)
* ubuntu 20.04 LTS or newer (or other Linux-oid operating system)
* installed packages (ubuntu DEB package names):
  - mandatory: gcc, gfortran, make
    subversion or git
    libhdf5-openmpi-dev
  - optional: gnuplot, julia, libomp-dev
    jupyter, pluma

Philosophy of PC

Philosophy of PC

=> Philosophy: **What is your view?**

Philosophy of PC

=> Philosophy: **What is your view?**

permissive

restrictive

Philosophy of PC

=> Philosophy: **What is your view?**

permissive

progressive

restrictive

conservative

Philosophy of PC

=> Philosophy: **What is your view?**

permissive

progressive

lean

restrictive

conservative

complete

Philosophy of PC

=> Philosophy: **What is your view?**

permissive		restrictive
progressive		conservative
lean		complete
efficient		powerful

Philosophy of PC

=> Philosophy: **What is your view?**

permissive		restrictive
progressive		conservative
lean		complete
efficient		powerful
up-to-date		well tested

Philosophy of PC

=> Philosophy: **What is your view?**

permissive		restrictive
progressive		conservative
lean		complete
efficient		powerful
up-to-date		well tested

=> Pencil Code Steering Committee

5 community representatives (Jenny, Piyali, Axel, Matthias, Philippe)

https://pencil-code.org/contact.php

Philosophy of PC

=> Philosophy: **What is your view?**

permissive		restrictive
progressive		conservative
lean		complete
efficient		powerful
up-to-date		well tested

=> Pencil Code Steering Committee

5 community representatives (Jenny, Piyali, Axel, Matthias, Philippe)

https://pencil-code.org/contact.php

=> Code of Conduct

pencil-code/license/CODE_OF_CONDUCT.md

Getting and Installing PC

Getting and Installing PC

Download:

https://pencil-code.org/ => “Download“ => subversion or git

Getting and Installing PC

Download:

https://pencil-code.org/ => “Download“ => subversion or git.

Getting write access:

https://account.pencil-code.org/ => “register“ as new user.

Important: use the same username and email address as on GitHub!

Getting and Installing PC

Download:

https://pencil-code.org/ => “Download“ => subversion or git.

Getting write access:

https://account.pencil-code.org/ => “register“ as new user.

Important: use the same username and email address as on GitHub!

Once registered, apply for write access to “main” repository.

Inform your mentor or a core developer to grant your request.

(Requests get deleted if unanswered for longer time...)

Getting and Installing PC

Download:

https://pencil-code.org/ => “Download“ => subversion or git.

Getting write access:

https://account.pencil-code.org/ => “register“ as new user.

Important: use the same username and email address as on GitHub!

Once registered, apply for write access to “main” repository.

Inform your mentor or a core developer to grant your request.

(Requests get deleted if unanswered for longer time...)

Installing:

https://pencil-code.org/ => “Documentation“ => “Quick Start Guide“.

OpenMP parallelization

How does OpenMP work?

1) Parallelization by independent threads
2) Parallelization of loops
3) Parallelization by separation of the input parameter space

How does OpenMP work?

1) Parallelization by independent threads
2) Parallelization of loops
3) Parallelization by separation of the input parameter space

* clear begin and end of parallelized code segments
* data models are typically “private” versus “shared” memory

How does OpenMP work?

1) Parallelization by independent threads
2) Parallelization of loops
3) Parallelization by separation of the input parameter space

* clear begin and end of parallelized code segments
* data models are typically “private” versus “shared” memory

=> Short overview:
https://cyber.dabamos.de/programming/modernfortran/openmp.html

=> Documentation of version 5, including examples:
https://www.openmp.org/wp-content/uploads/openmp-examples-5.0.0.pdf

OpenMP parallelization

Implicit parallelization within the code:

```
#define N 100000

int main() {
    int a[N];

#pragma omp parallel for
    for (int i = 0; i < N; ++i)
        a[i] = 2 * i;

    return 0;
}
```

1) loop is usually executed sequentially (on one processor)

OpenMP parallelization

Implicit parallelization within the code:

```
#define N 100000

int main() {
    int a[N];

#pragma omp parallel for
    for (int i = 0; i < N; ++i)
        a[i] = 2 * i;

    return 0;
}
```

1) loop is usually executed sequentially (on one processor)
2) on multiple processors, loop can be split in equal independent chunks

Implicit parallelization within the code:

```
#define N 100000

int main() {
    int a[N];

#pragma omp parallel for
    for (int i = 0; i < N; ++i)
        a[i] = 2 * i;

    return 0;
}
```

1) loop is usually executed sequentially (on one processor)
2) on multiple processors, loop can be split in equal independent chunks
3) distribution of computation on many processors (#proc)

=> up to #proc times faster!

Implicit parallelization within the code:

```
#define N 100000

int main() {
    int a[N];

#pragma omp parallel for
    for (int i = 0; i < N; ++i)
        a[i] = 2 * i;

    return 0;
}
```

1) loop is usually executed sequentially (on one processor)
2) on multiple processors, loop can be split in equal independent chunks
3) distribution of computation on many processors (#proc)

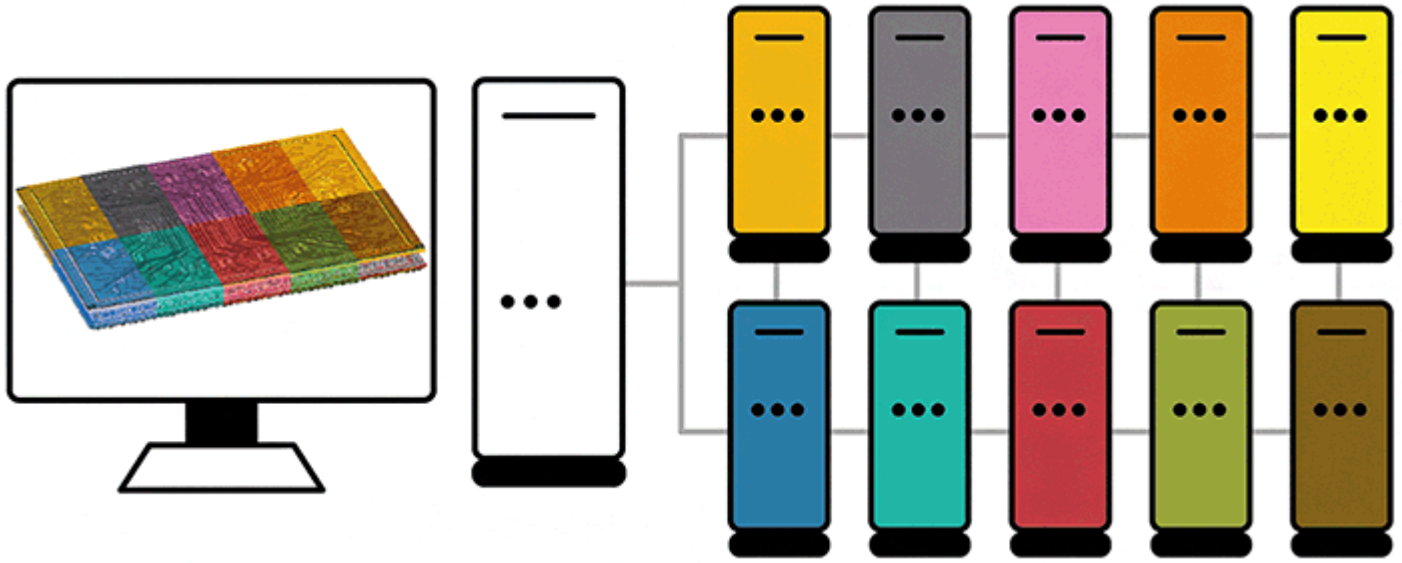
=> up to #proc times faster!

(if code permits parallelization!!!)

MPI parallelization

MPI parallelization

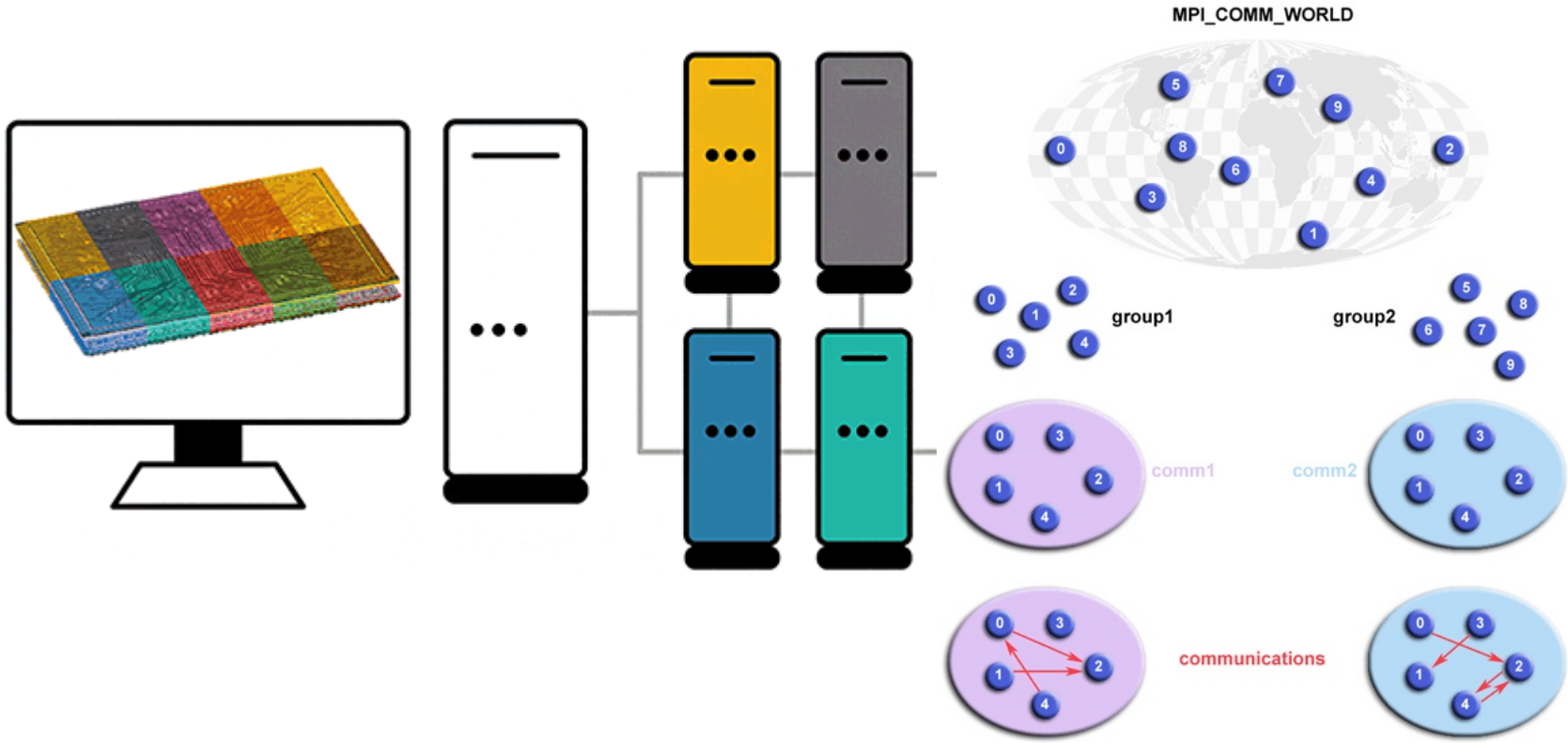
* **Message Passing Interface (MPI) – how it works:**



=> typically identical compute nodes

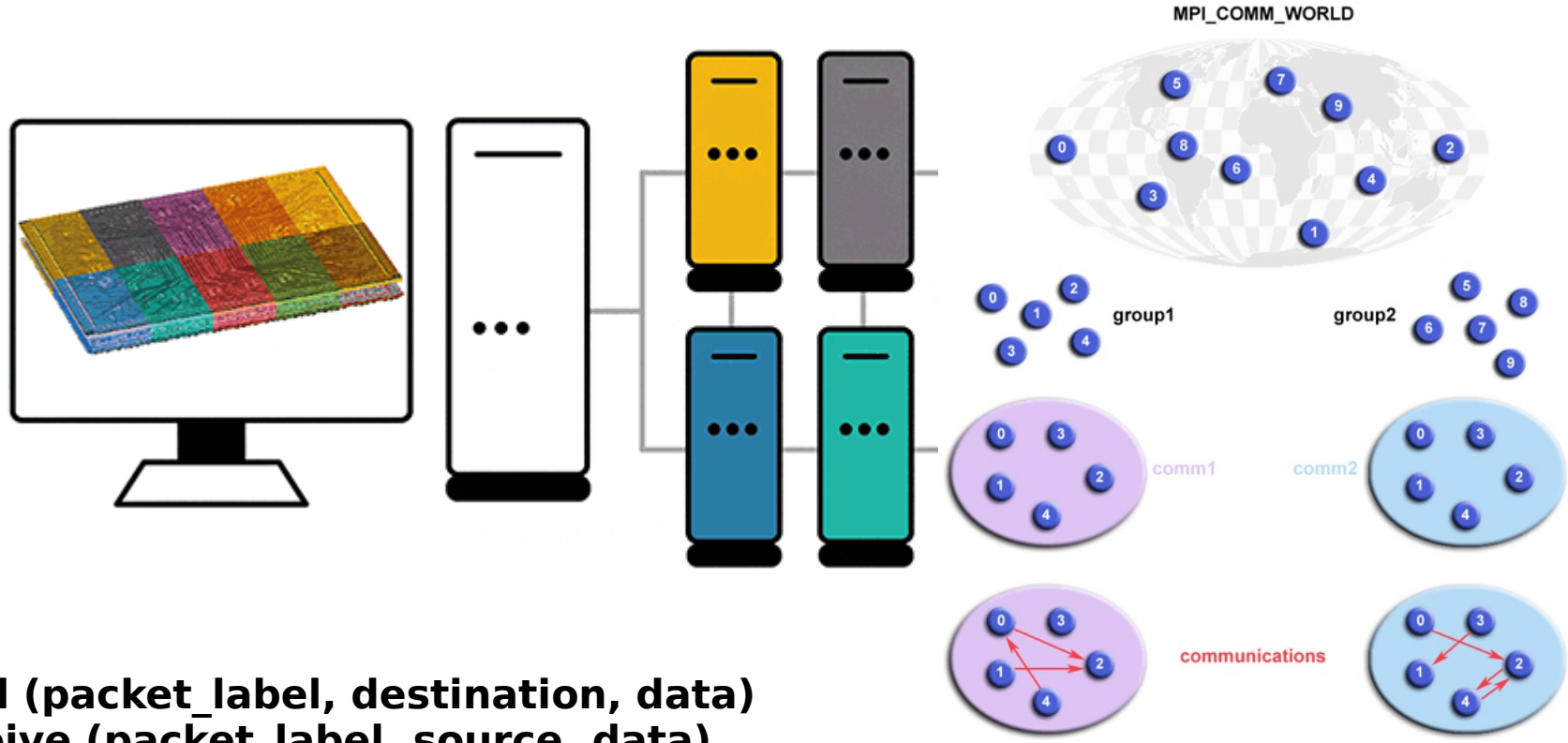
MPI parallelization

* **Message Passing Interface (MPI) – how it works:**



MPI parallelization

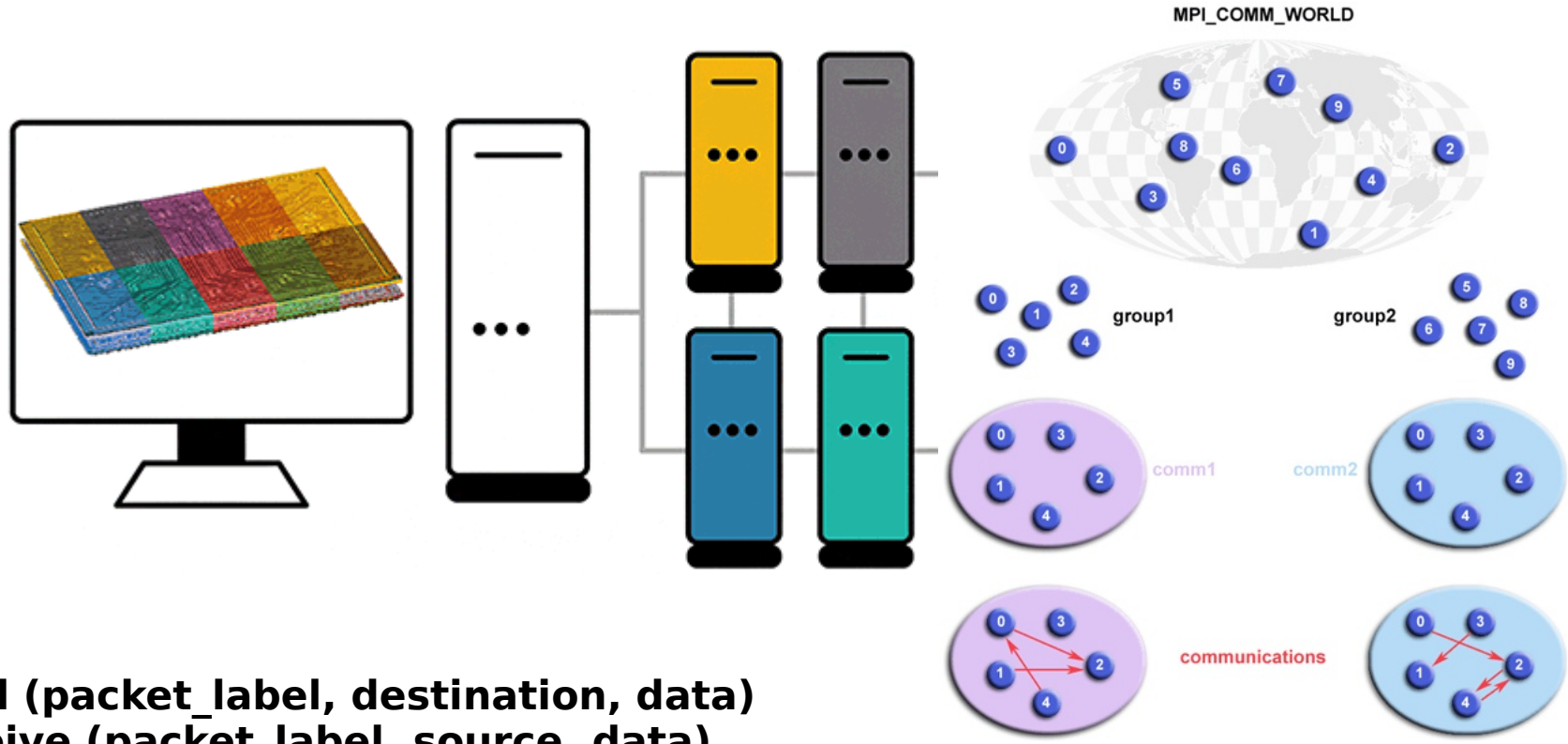
* **Message Passing Interface (MPI) – how it works:**



MPI_Send (packet_label, destination, data)
MPI_Receive (packet_label, source, data)

MPI parallelization

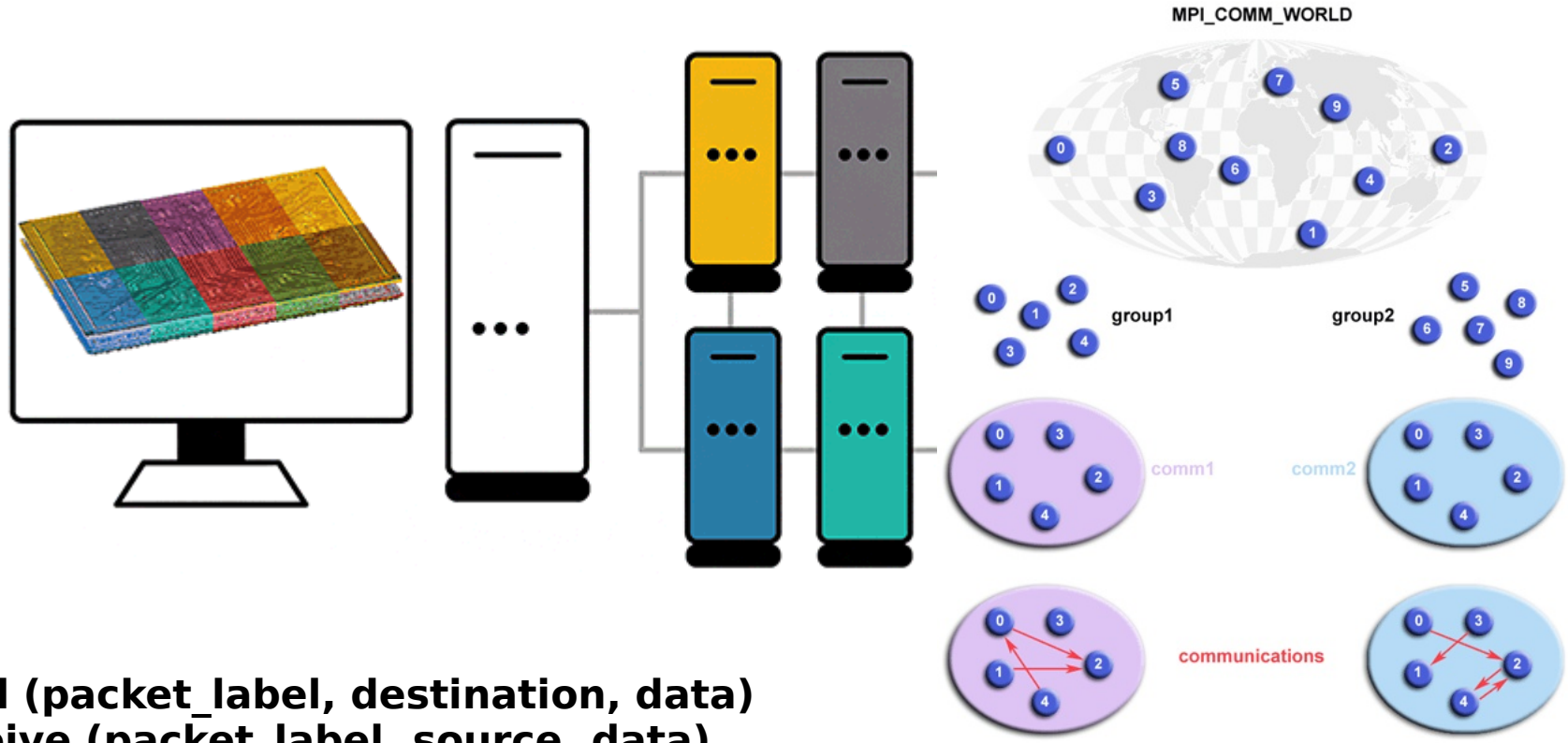
* **Message Passing Interface (MPI) – how it works:**



MPI_Send (packet_label, destination, data)
MPI_Receive (packet_label, source, data)
...
MPI_ISend (packet_label, destination, data)
MPI_IReceive (packet_label, source, data)
MPI_IWait (packet_label, source)

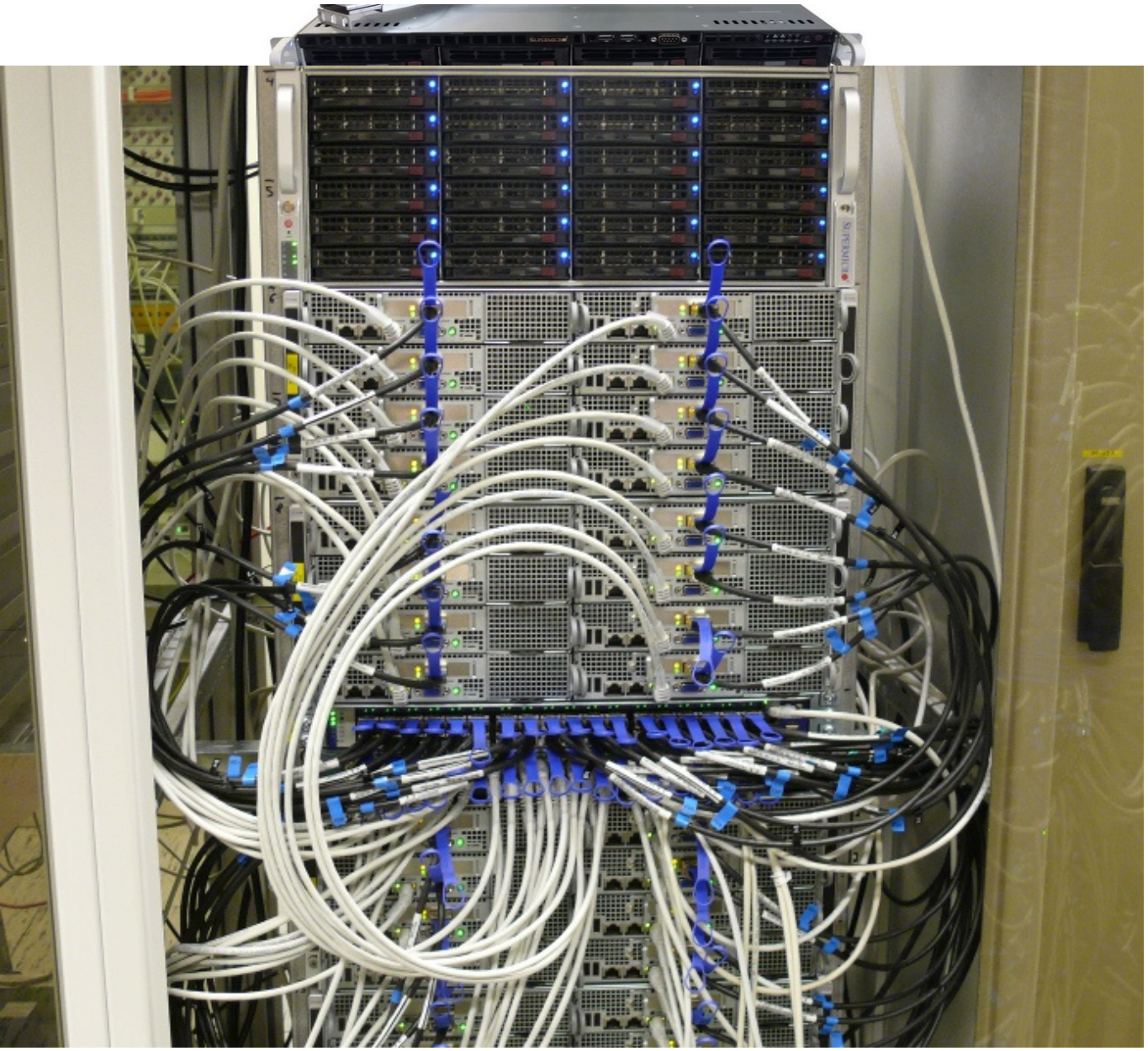
MPI parallelization

* **Message Passing Interface (MPI) – how it works:**

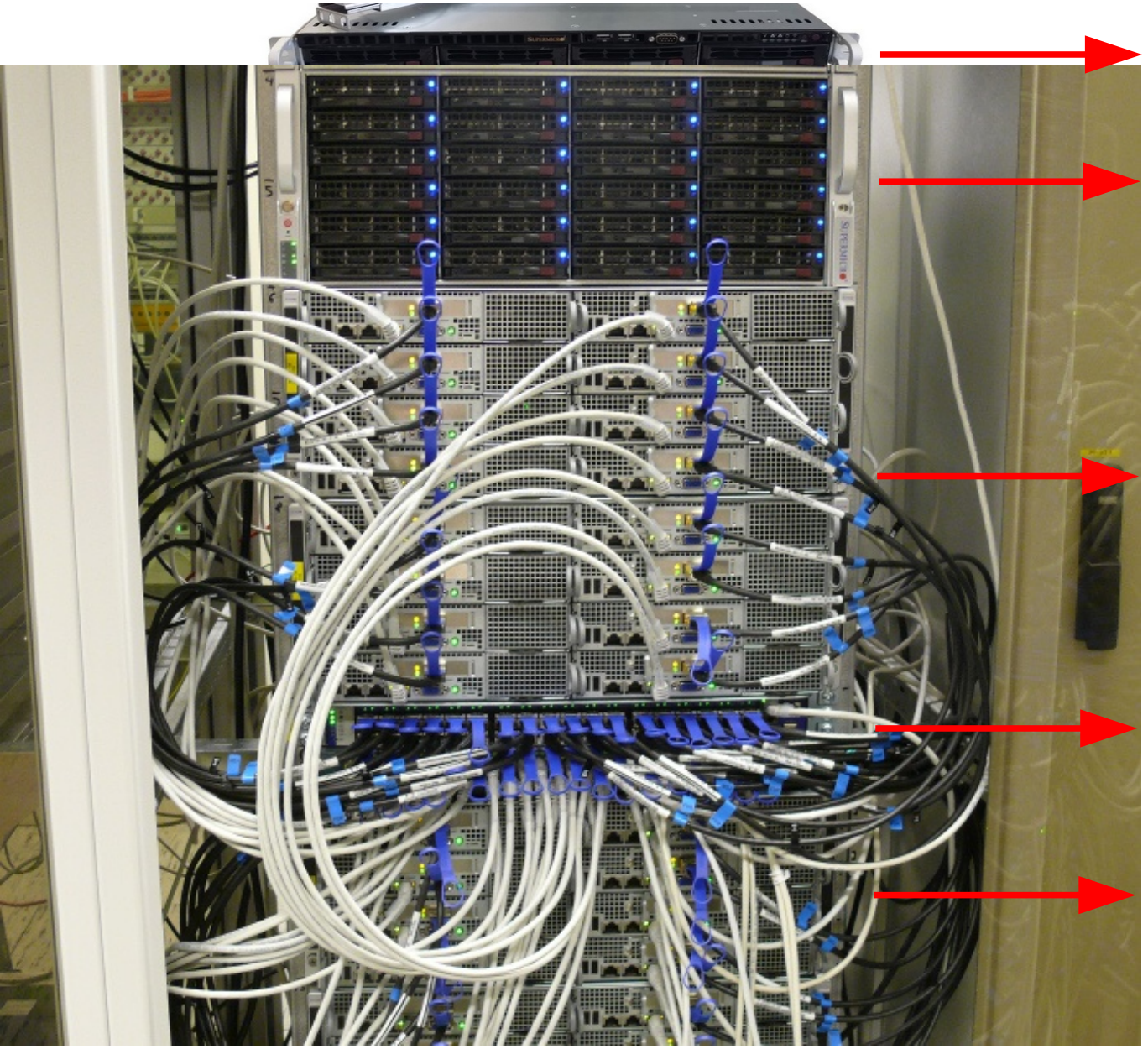


MPI_Send (packet_label, destination, data)
MPI_Receive (packet_label, source, data)
...
MPI_ISend (packet_label, destination, data)
MPI_IReceive (packet_label, source, data)
MPI_IWait (packet_label, source)
...
MPI_Broadcast (packet_label, source, data[, target group])
MPI_Barrier ([target group])

LEO high-performance computer:



LEO high-performance computer:



frontend (LEO1 + LEO2)

harddisks (LEONAS1 + 2)

compute nodes (LEONs)

network switches

compute nodes (LEONs)

LEO high-performance computer:

* **LEO1 frontend server:**
  40 processors, 384 GB RAM, 56 GBit/s Infiniband network, 64 bit CentOS 7

LEO high-performance computer:

* **LEO1 frontend server:**
  40 processors, 384 GB RAM, 56 GBit/s Infiniband network, 64 bit CentOS 7

* **LEON1-32 compute nodes:**
  40 processors, 256 GB RAM, 56 GBit/s Infiniband network, 64 bit CentOS 7
  => 1280 processors
  => 8 TB RAM
  => 128 TB SSD for parallel file IO

LEO high-performance computer:

* **LEO1 frontend server:**
  40 processors, 384 GB RAM, 56 GBit/s Infiniband network, 64 bit CentOS 7

* **LEON1-32 compute nodes:**
  40 processors, 256 GB RAM, 56 GBit/s Infiniband network, 64 bit CentOS 7
  => 1280 processors
  => 8 TB RAM
  => 128 TB SSD for parallel file IO

* **scientific application areas:**
  MHD, PIC, equation inversions, atmospheric models, MMS data processing

LEO high-performance computer:

* **LEO1 frontend server:**
  40 processors, 384 GB RAM, 56 GBit/s Infiniband network, 64 bit CentOS 7

* **LEON1-32 compute nodes:**
  40 processors, 256 GB RAM, 56 GBit/s Infiniband network, 64 bit CentOS 7
  => 1280 processors
  => 8 TB RAM
  => 128 TB SSD for parallel file IO

* **scientific application areas:**
  MHD, PIC, equation inversions, atmospheric models, MMS data processing

* **computation time:**
  > 10 million core-hours per year (~ 250 k€ per year)

LEO high-performance computer:

* **software modules library: “module avail”**

```
---------------------- /software/modules/COMPILER ----------------------
gnu/RHEL-7(default)

------------------------ /software/modules/DATA ------------------------
idl/7.06/32_bit              python/2.7.9+3.4.2(default)
idl/7.06/64_bit              qsas/3.0.1(default)
idl/8.2.3                    spedas/1.00(default)
idl/8.4(default)             spedas/bleeding
julia/0.3.9(default)         spedas/yesterday
julia/0.4.0-dev

------------------------ /software/modules/LIBS ------------------------
fftw/3.3.4(default)              phdf5/1.8.17/mvapich2/2.1/gnu
nasa-cdf/3.6.0.4(default)        phdf5/1.8.17/openmpi/1.10.3/gnu
phdf5/1.8.17/mpich/3.1.4/gnu     solarsoft/2014-12-09(default)

------------------------ /software/modules/MPI -------------------------
mpich/3.1.4/gnu    mvapich2/2.1/gnu   openmpi/1.10.3/gnu

---------------------- /software/modules/TOOLS -----------------------
dislin/10.5.3(default)              valgrind/3.11.0/openmpi/1.10.3/gnu
filezilla/3.10.0.2(default)
```

LEO high-performance computer:

* **MPI job queue:**

Name	maximum Runtime per job	at least Nodes per job	maxium Nodes per job	maximum simultaneously used Nodes by this queue	maximum running Jobs per User	Priority	Default
`short`	20 min	1	2	2	2	20	
`normal`	24 h	1	13	32	4	10	X
`long`	96 h	1	4	4	1	5	
`large`	12 h	17	32	32	1	15	

Compiling PC

Compiling PC

1) Choosing a sample

   => **cd pencil-code/samples/** && ls -la

Compiling PC

1) Choosing a sample

   => **cd pencil-code/samples/** && ls -la

2) Selecting modules

   => edit **src/Makefile.local**

Compiling PC

1) Choosing a sample

   => **cd pencil-code/samples/** && ls -la

2) Selecting modules

   => edit **src/Makefile.local**

3) Selecting processor layout

   => edit **src/cparam.local**

Compiling PC

1) Choosing a sample

   => **cd pencil-code/samples/** && ls -la

2) Selecting modules

   => edit **src/Makefile.local**

3) Selecting processor layout

   => edit **src/cparam.local**

4) Compiling PC

   => **pc_build**

Memory alignment and cache efficiency

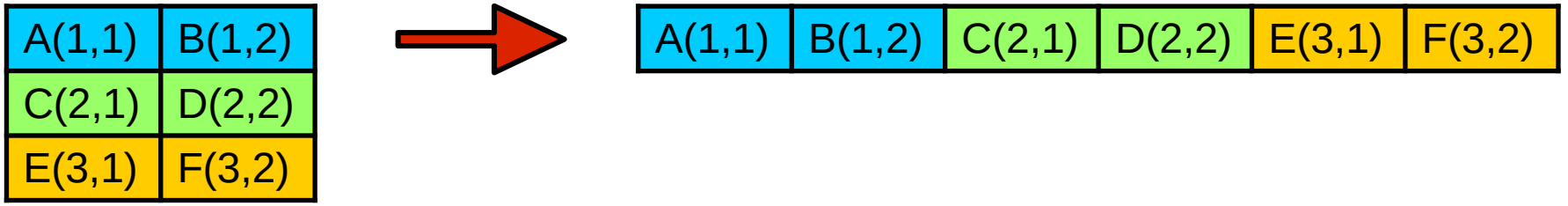
Memory alignment and cache efficiency

Alignment of data in memory:

A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)

Memory alignment and cache efficiency

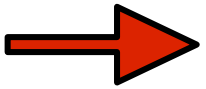
Alignment of data in memory:



Memory alignment and cache efficiency

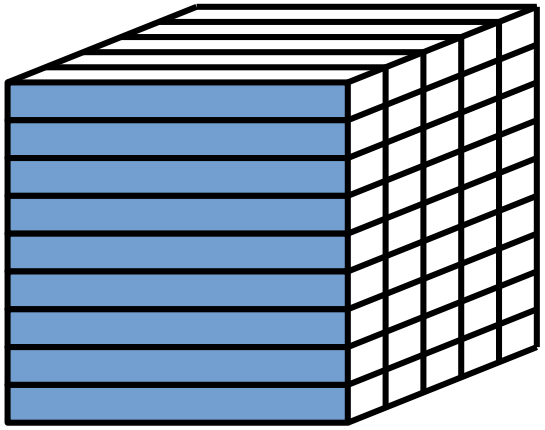
Alignment of data in memory:

A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)



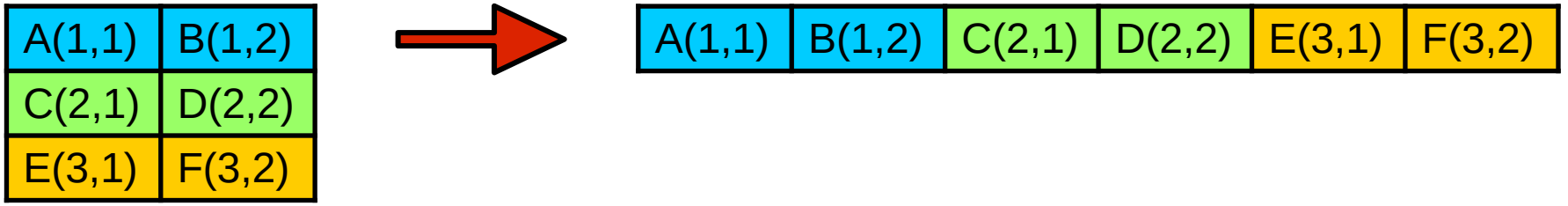
A(1,1)	B(1,2)	C(2,1)	D(2,2)	E(3,1)	F(3,2)

Cache efficiency:

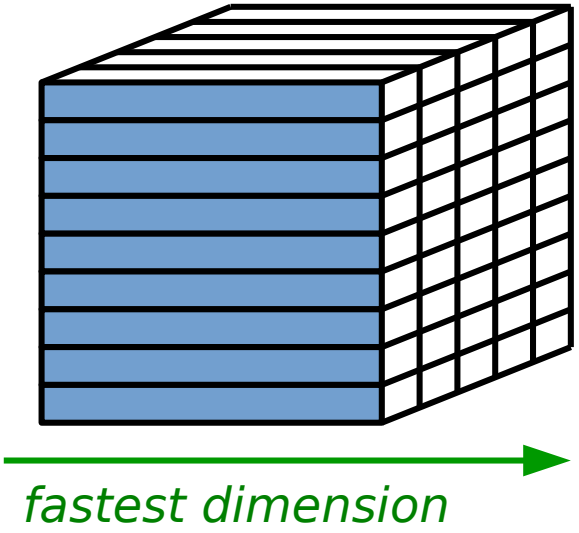


Memory alignment and cache efficiency

Alignment of data in memory:



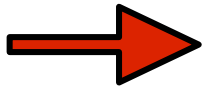
Cache efficiency:



Memory alignment and cache efficiency

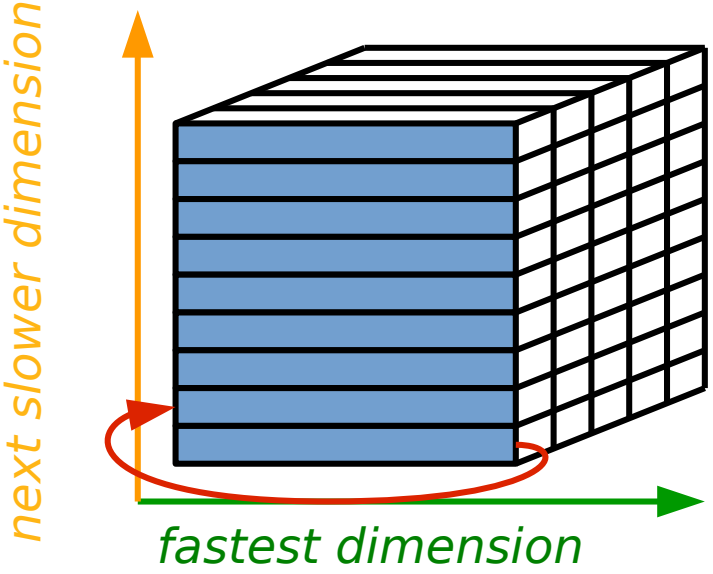
Alignment of data in memory:

A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)



A(1,1)	B(1,2)	C(2,1)	D(2,2)	E(3,1)	F(3,2)

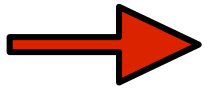
Cache efficiency:



Memory alignment and cache efficiency

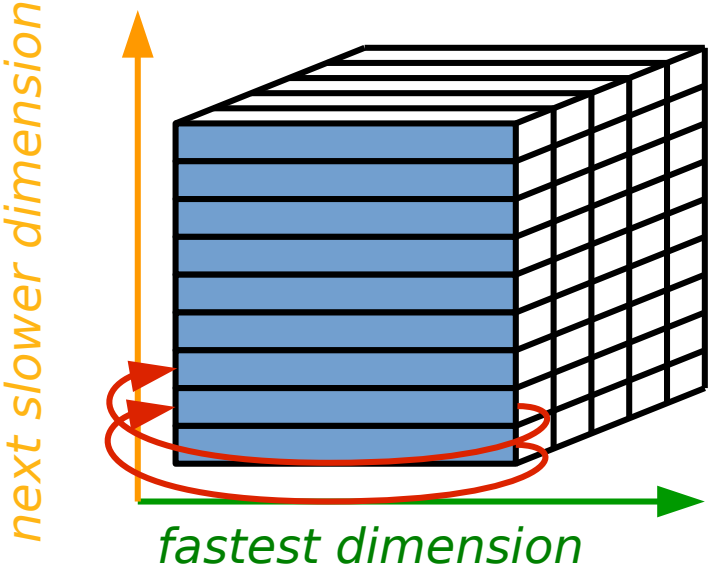
Alignment of data in memory:

A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)



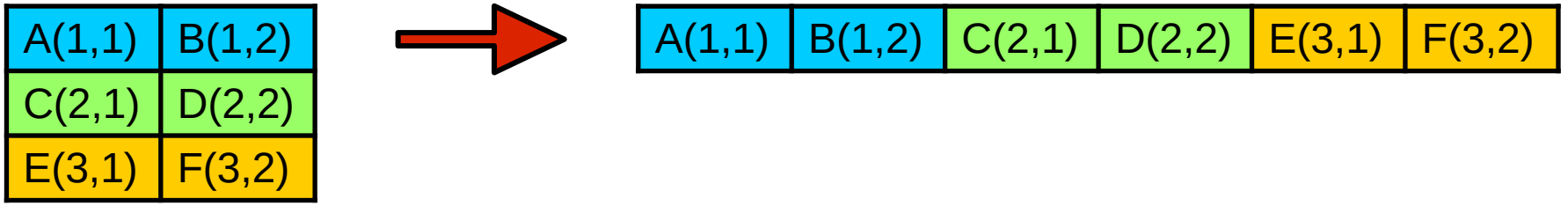
A(1,1)	B(1,2)	C(2,1)	D(2,2)	E(3,1)	F(3,2)

Cache efficiency:

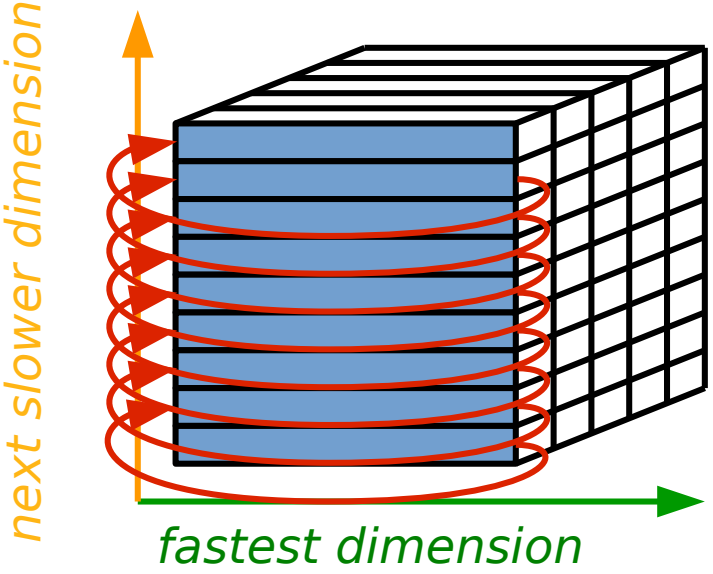


Memory alignment and cache efficiency

Alignment of data in memory:



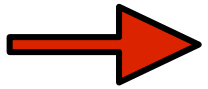
Cache efficiency:



Memory alignment and cache efficiency

Alignment of data in memory:

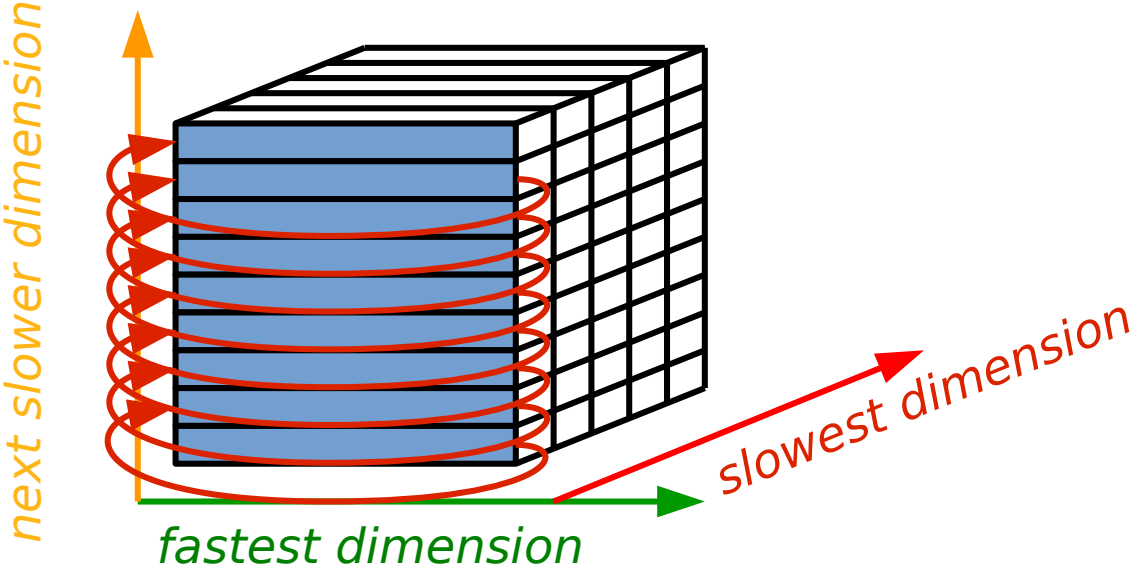
A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)



A(1,1)	B(1,2)	C(2,1)	D(2,2)	E(3,1)	F(3,2)

Cache efficiency:

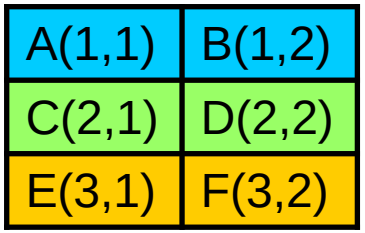
=> both is achieved by elongated subdomains and optimal data access



Example: Loop with data access (read/write)

Example: Loop with data access (read/write)

```
for iy = 1...3 {
      for ix = 1...2 {
            f[ix,iy] = ix+iy*2-2
      }
}
```



Example: Loop with data access (read/write)

C / C++ / NumPy:

```
for iy = 1...3 {
      for ix = 1...2 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)

=> “Row Major”

Example: Loop with data access (read/write)

C / C++ / NumPy:

```
for iy = 1...3 {
      for ix = 1...2 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)

=> “Row Major”

Fortran / IDL / Julia / Matlab / R:

```
for ix = 1...2 {
      for iy = 1...3 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

=> “Column Major”

Example: Loop with data access (read/write)

C / C++ / NumPy:

```
for iy = 1...3 {
      for ix = 1...2 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)

=> “Row Major”

A(1,1)	B(1,2)	C(2,1)	D(2,2)	E(3,1)	F(3,2)
1	2	3	4	5	6

Fortran / IDL / Julia / Matlab / R:

```
for ix = 1...2 {
      for iy = 1...3 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

=> “Column Major”

Example: Loop with data access (read/write)

Unaligned access:

C / C++ / NumPy:

```
for iy = 1...3 {
      for ix = 1...2 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)

=> “Row Major”

A(1,1)	B(1,2)	C(2,1)	D(2,2)	E(3,1)	F(3,2)
1	2	3	4	5	6

Fortran / IDL / Julia / Matlab / R:

```
for ix = 1...2 {
      for iy = 1...3 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

=> “Column Major”

Example: Loop with data access (read/write)

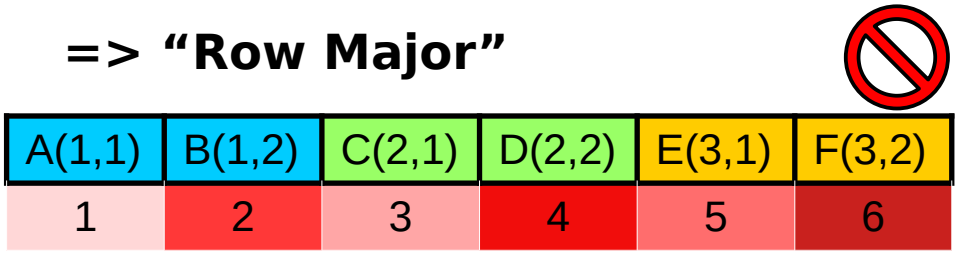
Unaligned access:

C / C++ / NumPy:

```
for iy = 1...3 {
      for ix = 1...2 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)

=> “Row Major”



Fortran / IDL / Julia / Matlab / R:

```
for ix = 1...2 {
      for iy = 1...3 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

=> “Column Major”

Aligned access:

C / C++ / NumPy:

```
for ix = 1...2 {
      for iy = 1...3 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

Fortran / IDL / Julia / Matlab / R:

```
for iy = 1...3 {
      for ix = 1...2 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

Example: Loop with data access (read/write)

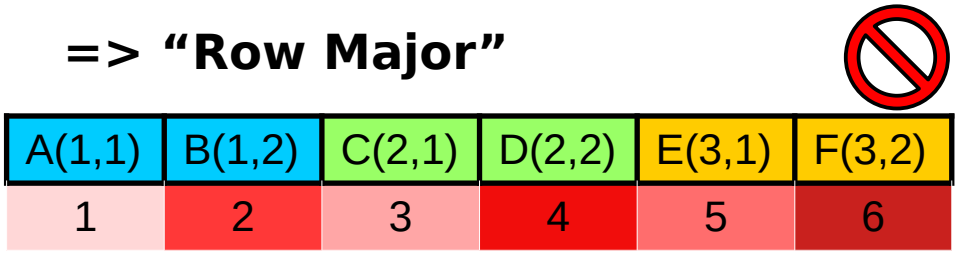
Unaligned access:

C / C++ / NumPy:

```
for iy = 1...3 {
      for ix = 1...2 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)

=> “Row Major”



Fortran / IDL / Julia / Matlab / R:

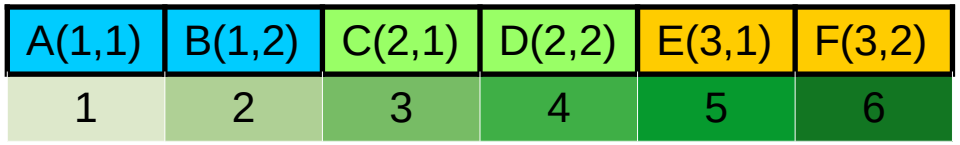
```
for ix = 1...2 {
      for iy = 1...3 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

=> “Column Major”

Aligned access:

C / C++ / NumPy:

```
for ix = 1...2 {
      for iy = 1...3 {
            f[ix,iy] = ix+iy*2-2
      }
}
```



Fortran / IDL / Julia / Matlab / R:

```
for iy = 1...3 {
      for ix = 1...2 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

Example: Loop with data access (read/write)

Unaligned access:

C / C++ / NumPy:

```
for iy = 1...3 {
      for ix = 1...2 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)

=> “Row Major”

A(1,1)	B(1,2)	C(2,1)	D(2,2)	E(3,1)	F(3,2)
1	2	3	4	5	6

Fortran / IDL / Julia / Matlab / R:

```
for ix = 1...2 {
      for iy = 1...3 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

=> “Column Major”

Aligned access:

C / C++ / NumPy:

```
for ix = 1...2 {
      for iy = 1...3 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

A(1,1)	B(1,2)	C(2,1)	D(2,2)	E(3,1)	F(3,2)
1	2	3	4	5	6

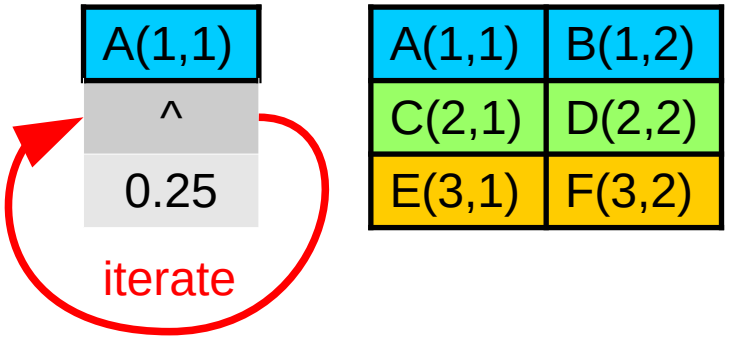
Fortran / IDL / Julia / Matlab / R:

```
for iy = 1...3 {
      for ix = 1...2 {
            f[ix,iy] = ix+iy*2-2
      }
}
```

Vectorization and SIMD

Vectorization and SIMD

Explicit iteration:



A(1,1)	B(1,2)
C(2,1)	D(2,2)
E(3,1)	F(3,2)

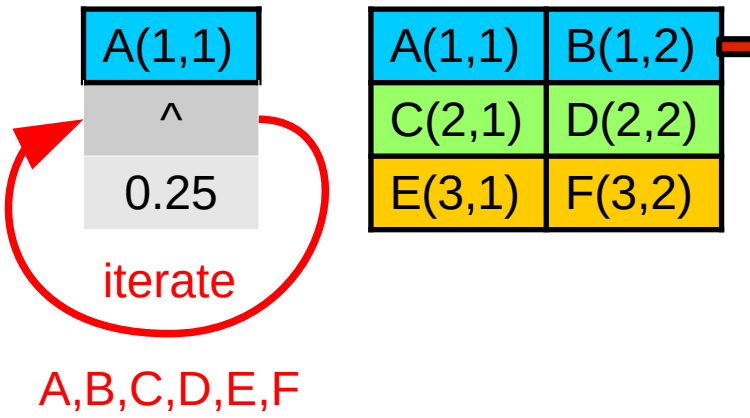
A,B,C,D,E,F

needed CPU instructions:

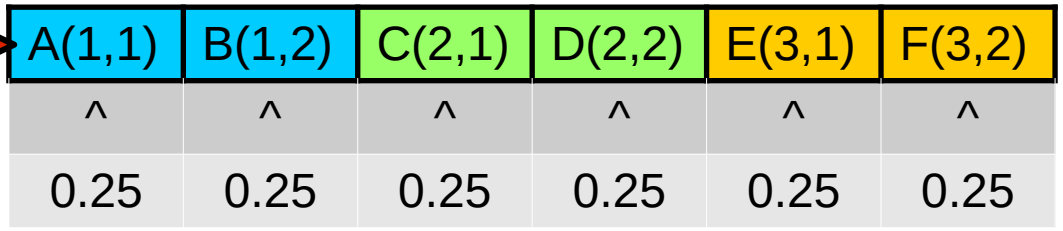
* 6 times loop overhead
* up to 6 times memory load
* 6 times mathematics
* 6 times memory store

Vectorization and SIMD

Explicit iteration:



Implicit iteration (vectorized):



needed CPU instructions:

* 6 times loop overhead
* up to 6 times memory load
* 6 times mathematics
* 6 times memory store

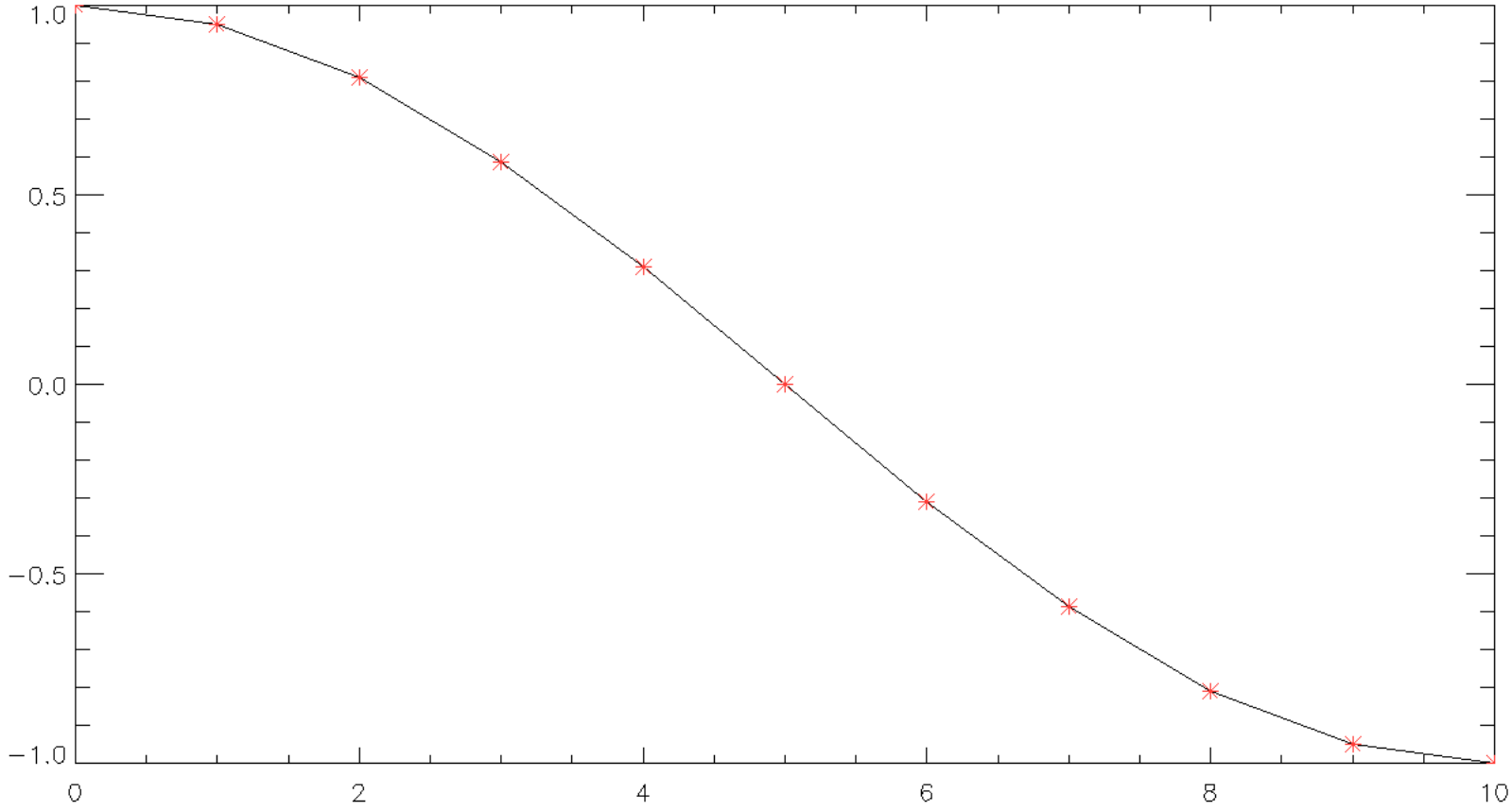
SIMD case:

* no loop overhead
* 1 memory load (L2-cache line!)
* 1 mathematics
* 1 memory store

  => up to 1000 times faster!

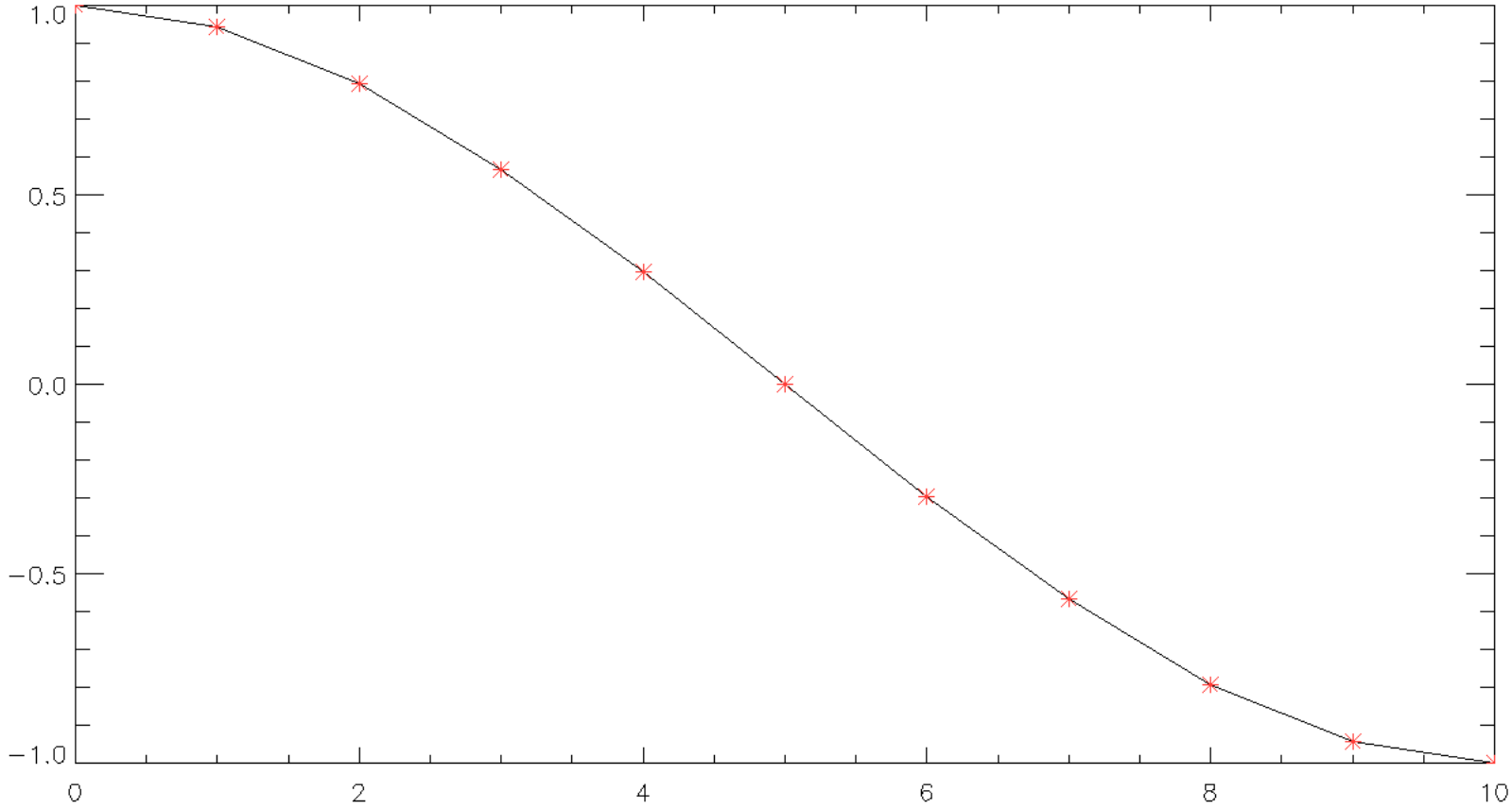
Abuse of mathematics

Abuse of mathematics



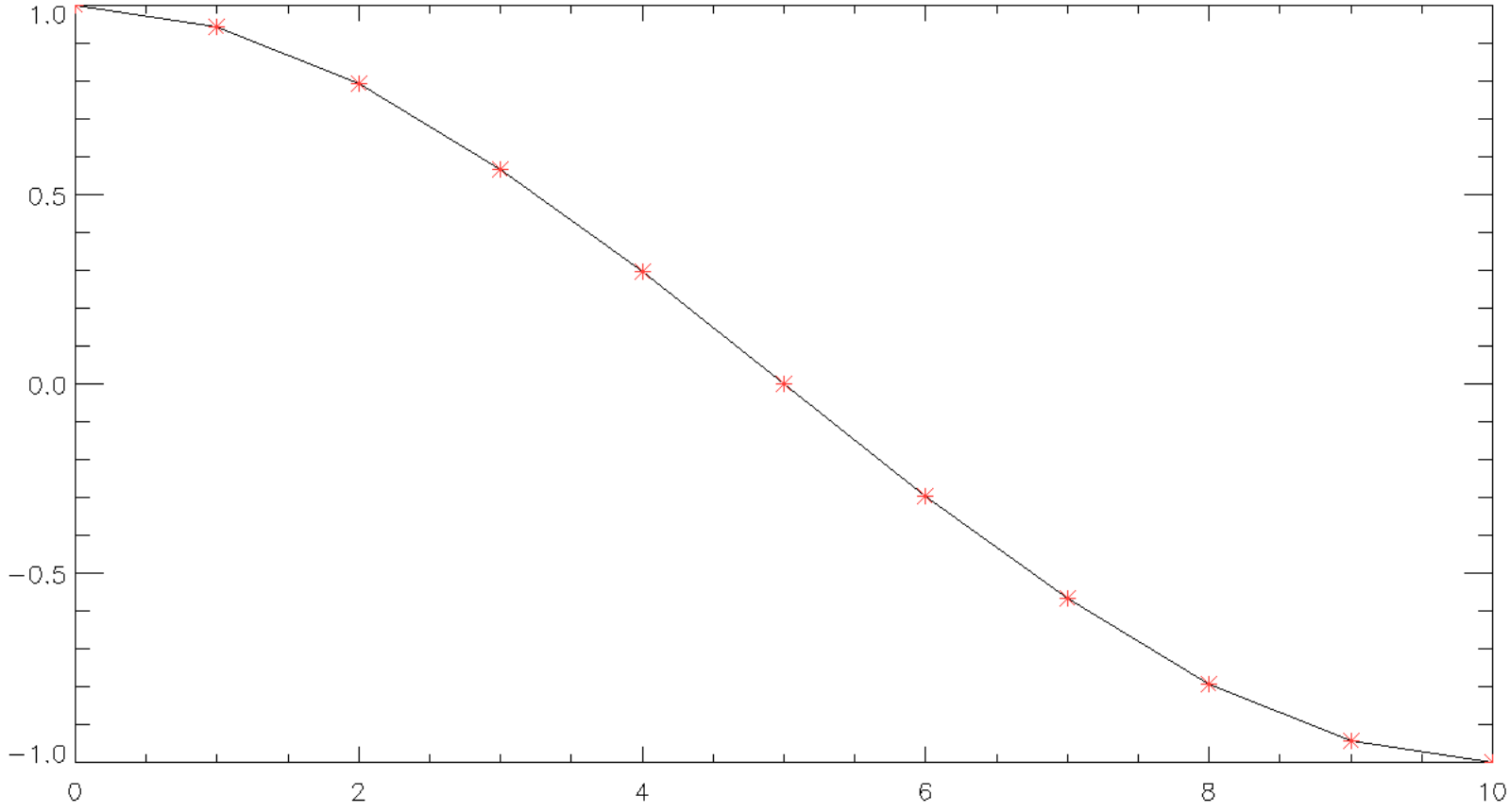
$$f(x)=\cos\left(\frac{\pi}{10}x\right)$$

Abuse of mathematics



$$f(x) = \ldots ?$$

Abuse of mathematics



$$\xi = \left(x - x_0 \right) / \Delta$$

$$f\left(\xi\right) = \frac{1}{2} + \xi\left(\frac{3}{4} - \frac{1}{4}\xi^2\right)$$

Running PC

Running PC

1) Changing parameters

   => edit **start.in**, **run.in**, **print.in**, …

Running PC

1) Changing parameters

   => edit **start.in**, **run.in**, **print.in**, …

2) Setting up initial condition (“start”)

   => **pc_run start**

Running PC

1) Changing parameters

   => edit **start.in**, **run.in**, **print.in**, …

2) Setting up initial condition (“start”)

   => **pc_run start**

3) Propagate an existing state (“run”)

   => **pc_run**

Boundary conditions

Boundary conditions

Problem: boundary condition is an assumption on the outside area

* ideas?

Boundary conditions

Problem: boundary condition is an assumption on the outside area

* Mirror/Symmetric $f(x_{B-j}) = f(x_{B+j})$
* Fixed Boundary $f(x_{B-j}) = f(x_B)$ or $f(x_{B-j}) = const.$
* Ignore Outside (= Mirroring)
* Extrapolation $f(x_{B-j}) = g(x_B, x_{B+j}, \ldots, f(x_B), f(x_{B+j}), \ldots)$
* Periodic $f(x_{B-j}) = f(x_{B+(N-1)-j})$
* Antisymmetric $f(x_{B-j}) = -f(x_{B+j})$ or $f(x_{B-j}) = f(x_B) - f(x_{B+j})$
* Wavelet (= extrapolation with set of pre-defined wave functions)
* and many other possibilities...

Boundary conditions

Problem: boundary condition is an assumption on the outside area

* Mirror/Symmetric $f(x_{B-j}) = f(x_{B+j})$
* Fixed Boundary $f(x_{B-j}) = f(x_B)$ or $f(x_{B-j}) = const.$
* Ignore Outside (= Mirroring)
* Extrapolation $f(x_{B-j}) = g(x_B, x_{B+j}, \ldots, f(x_B), f(x_{B+j}), \ldots)$
* Periodic $f(x_{B-j}) = f(x_{B+(N-1)-j})$
* Antisymmetric $f(x_{B-j}) = -f(x_{B+j})$ or $f(x_{B-j}) = f(x_B) - f(x_{B+j})$
* Wavelet (= extrapolation with set of pre-defined wave functions)
* and many other possibilities...

=> Be aware of the physical meaning of a boundary condition!