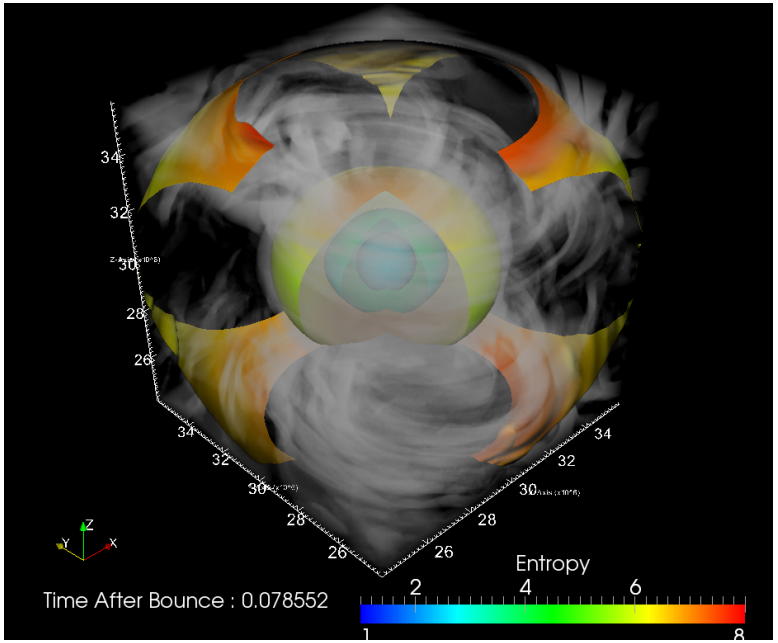
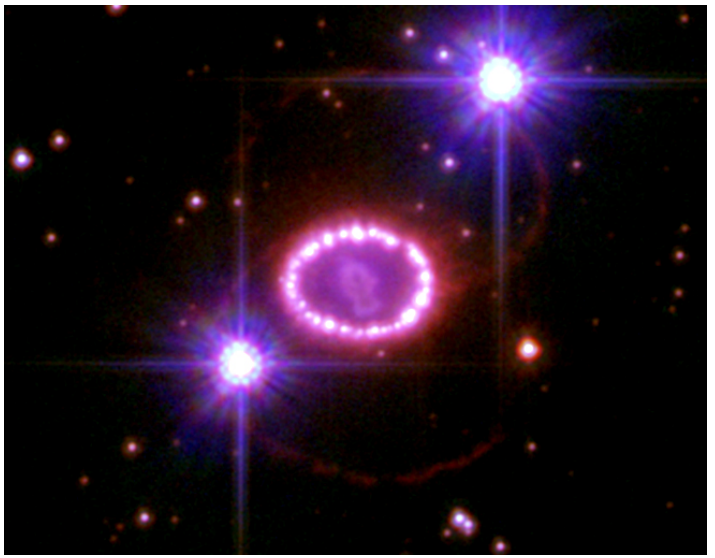
The FISH/ELEPHANT astrophysics codes for modelling stellar explosions

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Roger Käppeli

Seminar for Applied Mathematics





Collaborators:
Ruben Cabezon
Matthias Liebendörfer
Albino Perego
F.-K. Thielemann
Nicolas Vasset
Christian Winteler
John Biddiscombe (CSCS)
Sadaf Alam (CSCS)

CSCS
Swiss National Supercomputing Centre

HP2C High Performance and High Productivity Computing

Outline

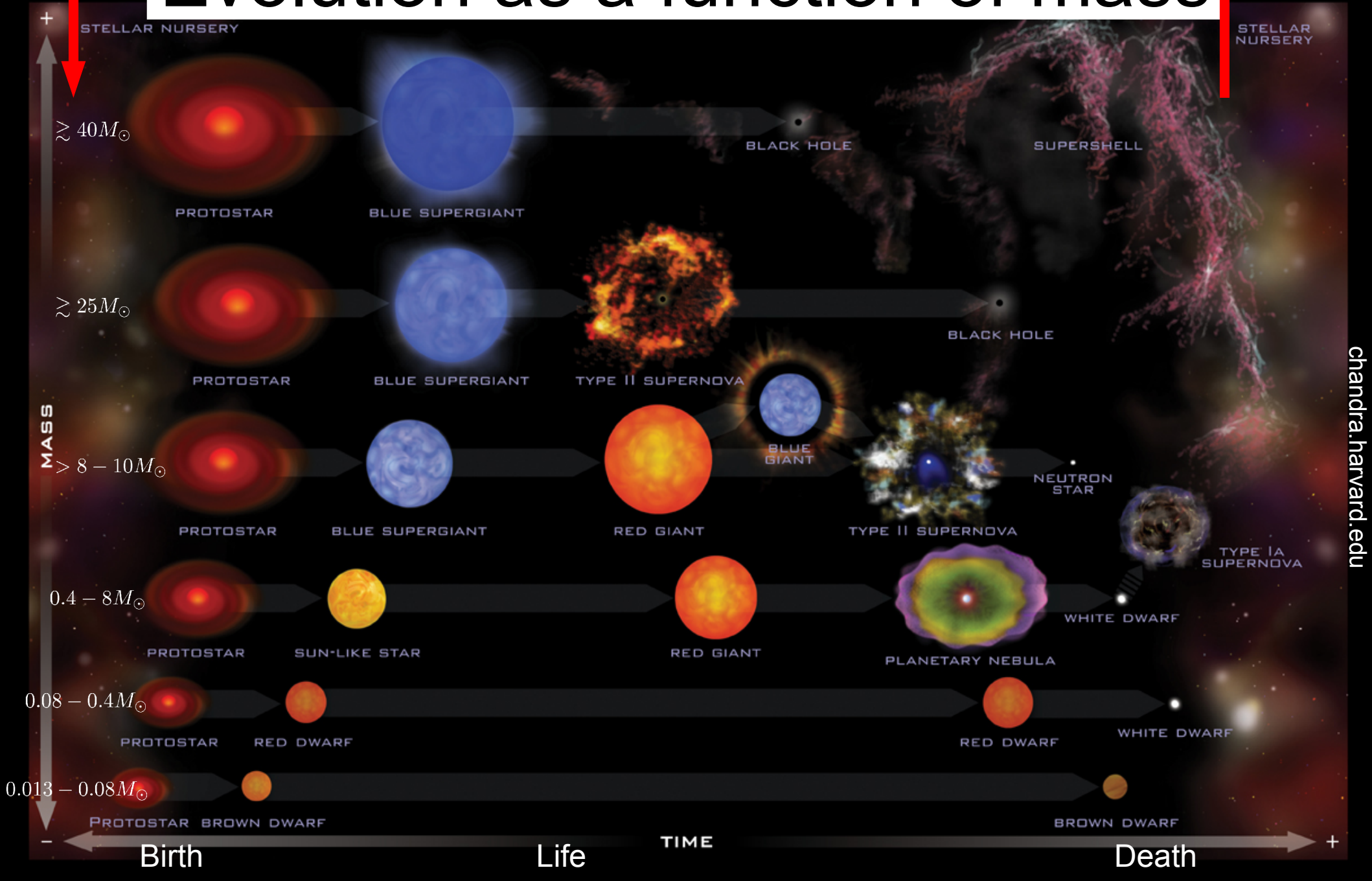
i. Stellar explosions
   - A (very) brief introduction to what we do and why we are doing it

ii. Methods & Algorithms
   - A quick overview of the origin of the computational work

iii. Implementation & Parallelisation
   - How the work is distributed to computational units

Stellar life cycle

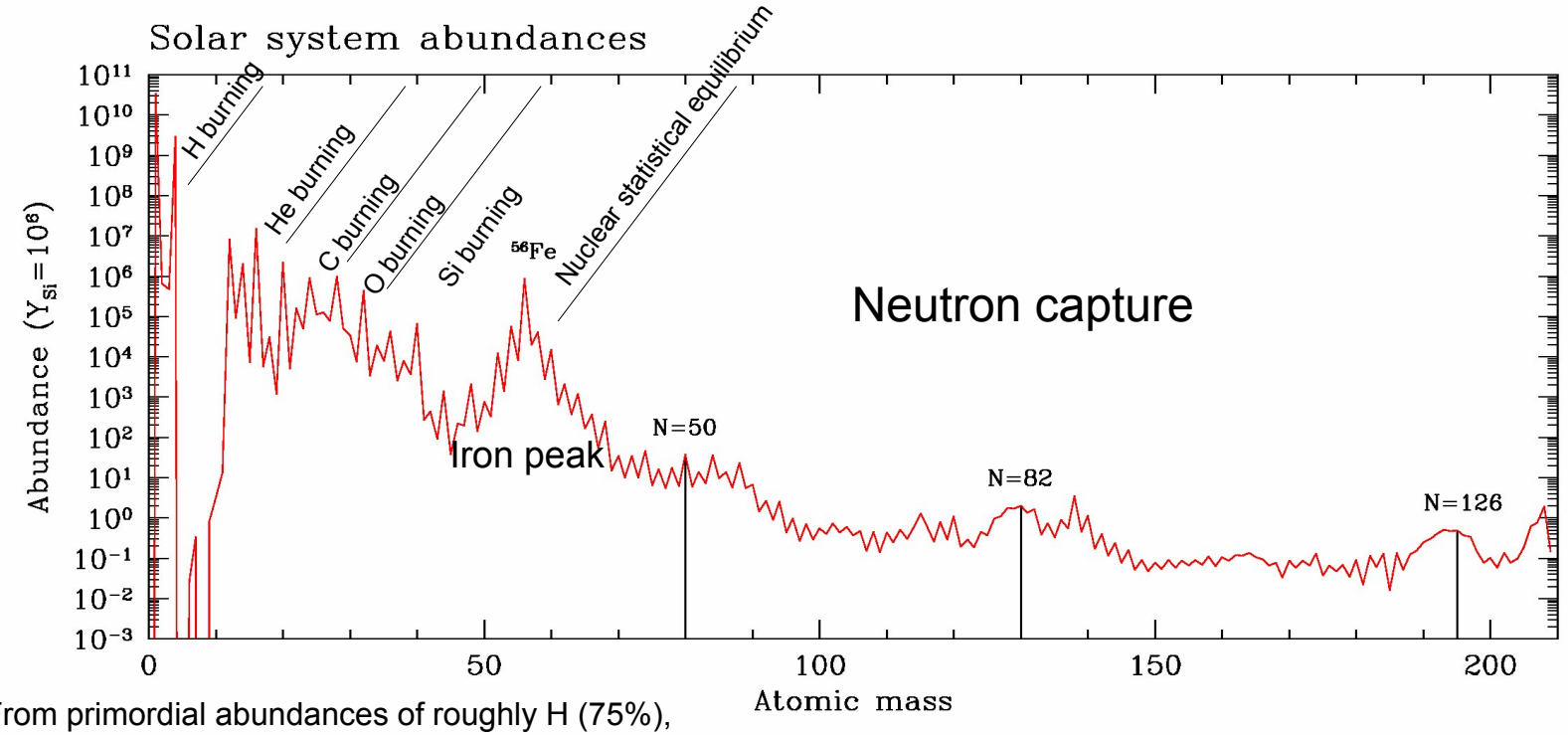
Evolution as a function of mass



Stellar life cycle

Evolution as a function of mass

Where do the elements come from?



From primordial abundances of roughly H (75%), He (25%), (very) small amount of Li

Adapted from Asplund 2005

MASS

$\gtrsim 40M_\odot$

$\gtrsim 25M_\odot$

$> 8-10M$

$0.4-8M_\odot$

$0.08-0.4M_\odot$

$0.013-0.08M_\odot$

STELLAR NURSERY

PROTOSTAR

RED DWARF

PROTOSTAR BROWN DWARF

RED DWARF

WHITE DWARF

BROWN DWARF

TYPE Ia SUPERNOVA

TIME

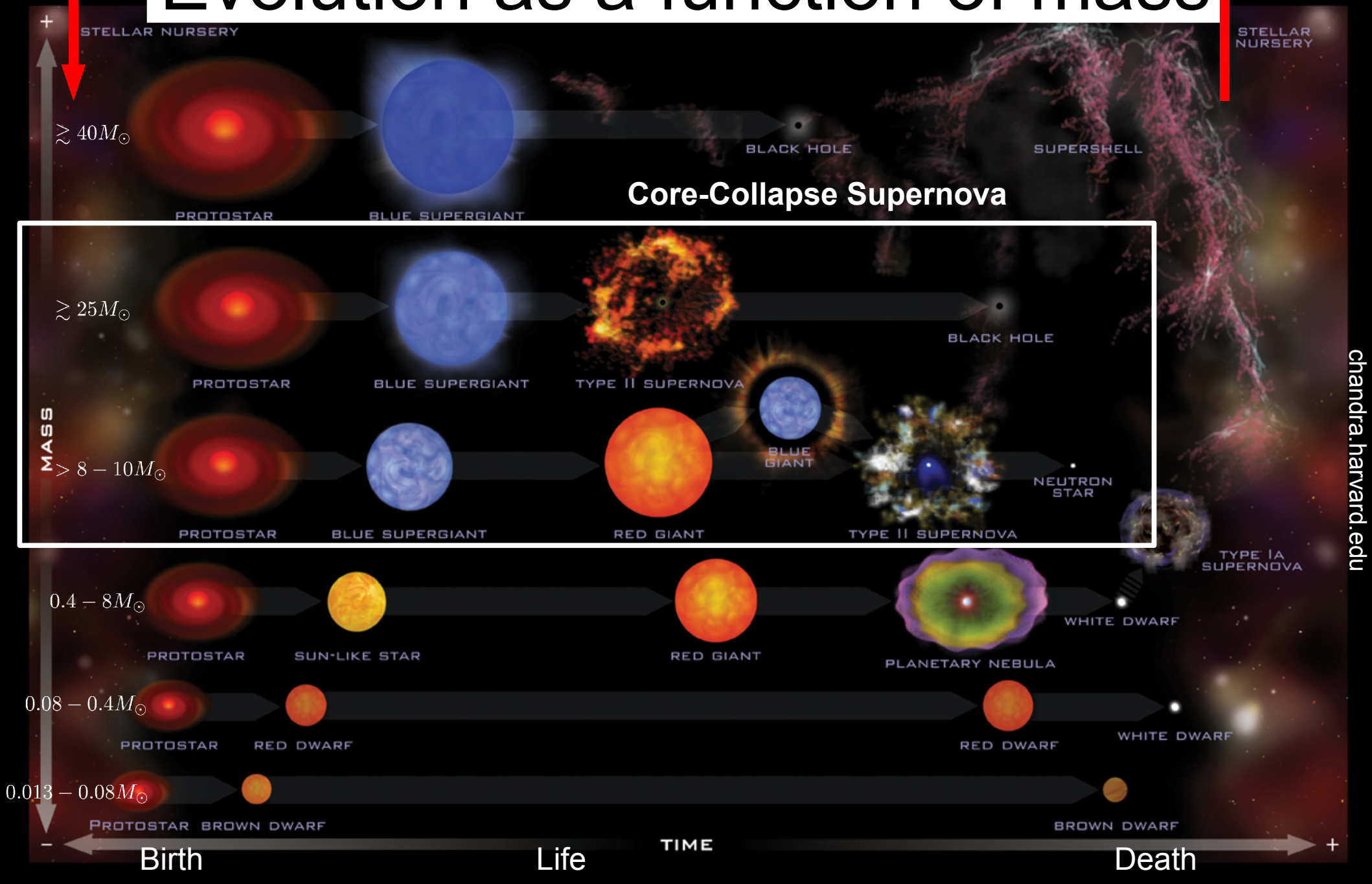
Birth

Life

Death

Stellar life cycle

Evolution as a function of mass



Core-collapse supernova

- General idea:
  - Implosion of iron core of massive $M \gtrsim 8 M_\odot$ at the end of thermonuclear evolution
  - Explosion powered by gravitational binding energy of forming compact remnant:

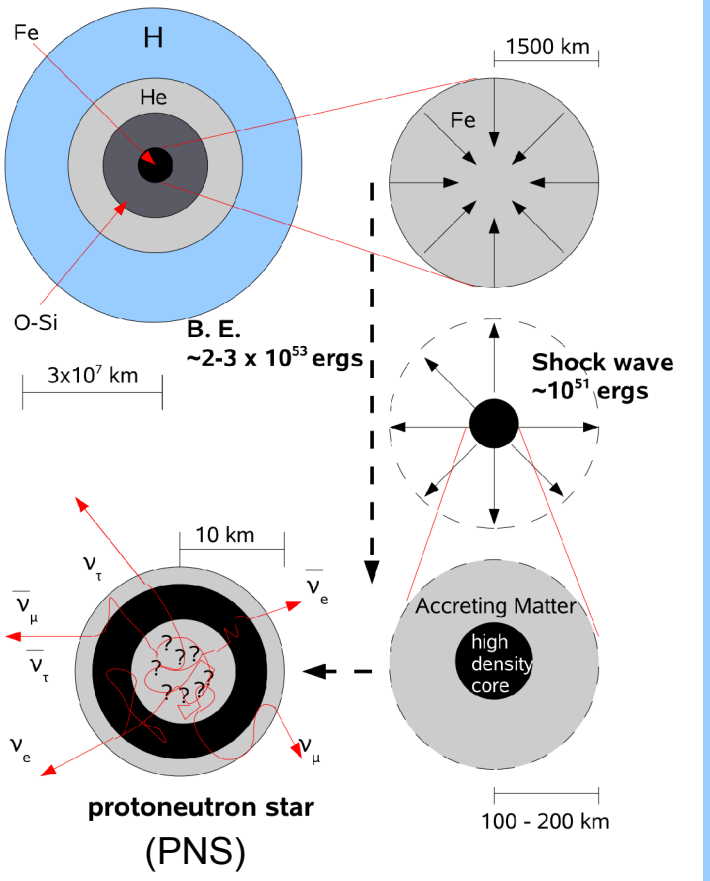
$$E_b \approx 3 \times 10^{53} \left(\frac{M}{M_\odot}\right)^2 \left(\frac{R}{10\mathrm{km}}\right)^{-1} \mathrm{erg}$$

GRAVITY BOMB!

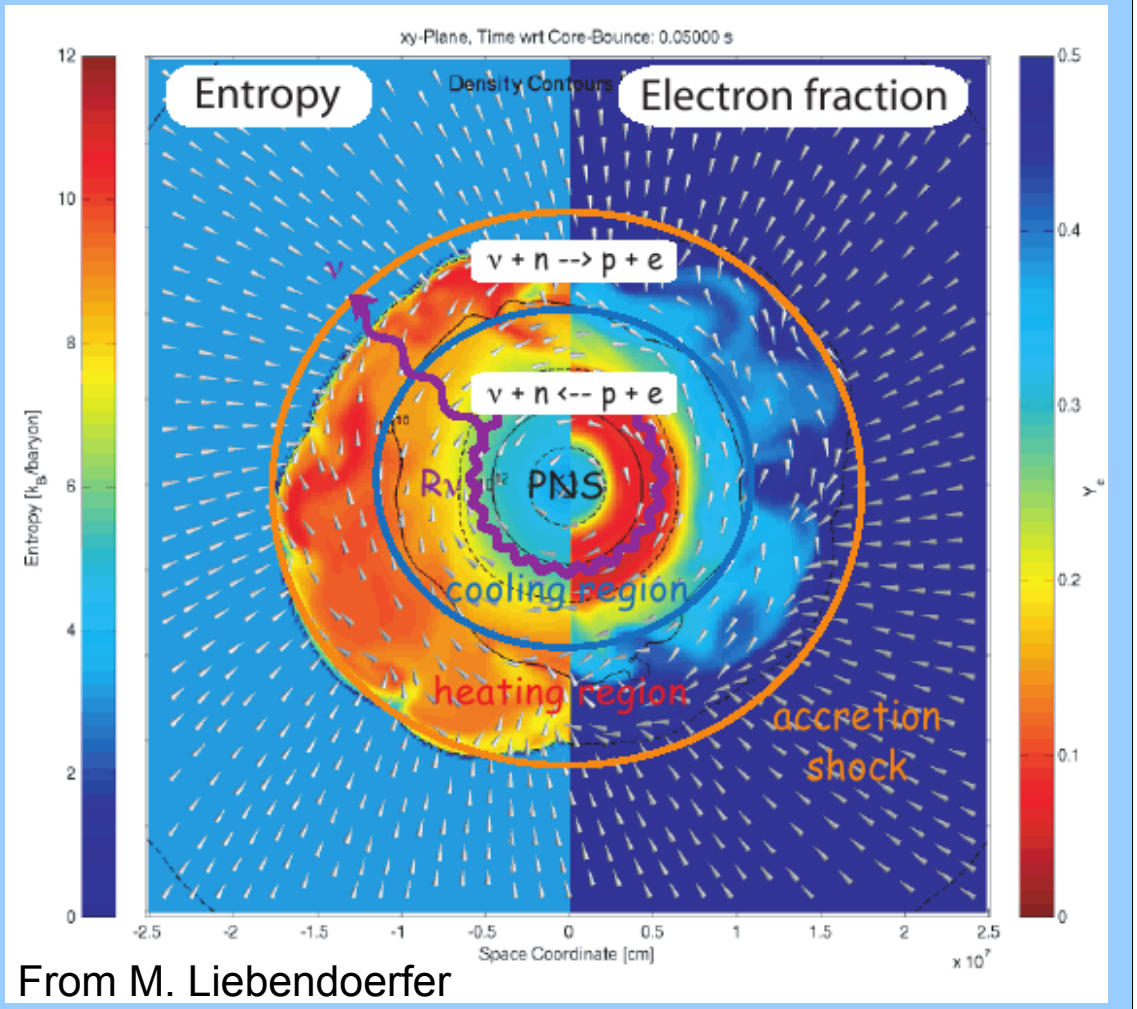
M Mass of remnant

R Radius of remnant

Core-collapse supernova



From S. Scheidegger



From M. Liebendoerfer

CCSN Explosion Mechanism?

- Discussed explosion mechanisms:
  - "Enhanced" neutrino-driven explosion mechanism

    Hydro. instabilities: convection, Standing Accretion Shock Instabilities (SASI) e.g. Blondin et al. 2003, Blondin & Shaw 2007, Foglizzo et al. 2008, Iwakami et al. 2008, Marek & Janka 2009, ...
  - MHD mechanism

    Rapid rotation + Magnetic field amplification (Flux compression, winding, MRI, dynamos) e.g. Akiyama et al. 2003, Wilson et al. 2005, Kotake et al. 2006, Burrows et al. 2007 Winteler et al. 2012...
  - Acoustic mechanism

    Excitation of ProtoNeutron Star (PNS) oscillations by accretion/SASI generating acoustic power to reheat the stalled shock Burrows et al. 2006,2007
  - Phase transition induced explosion mechanism

    Additional compactification of PNS due to phase transition from hadronic matter to quark matter Migdal et al. 1971, … Sagert, Fischer et al. 2009

Outline

i. Stellar explosions

- A (very) brief introduction to what we do and why we are doing it

ii. Methods & Algorithms

- A quick overview of the origin of the computational work

iii. Implementation & Parallelisation

- How the work is distributed to computational units

CCSN model

Model's ingredients wish list:

1) Multi-D hydro. — (no explosions generally in 1D, e.g. Thompson et al. (2003), Rampp & Janka (2002),Liebendoerfer et al. (2002/2005))

2) Plasma physics — Stars have magnetic fields, e.g. Sun!

3) Weak interactions

4) Neutrino transport — Most of the released gravitational binding energy “available” in form of neutrinos!

5) Nuclear physics — Equation of state describing matter at extreme conditions

6) General relativity — Very compact and very massive objects!

7) “Accurate” initial conditions

CCSN model

Actual model's ingredients list:

1) Multi-D hydro. →
2) Plasma physics →

} Parallel 3D ideal MHD code (Assume infinite conductivity)

3) Weak interactions →
4) Neutrino transport →

} Spectral leakage scheme
developed by A. Perego
Rosswog & Liebendörfer 2003
"Not so bad"... 2D simulations shown that ν contribute only 10-25% to explosion energy

5) Nuclear physics → EoS e.g. Lattimer & Swesty 1991, Shen et al. 1998, Hempel et al. 2011

6) General relativity → Spherical effective GR potential Marek et al. 2006

7) "Accurate" initial conditions → + 2D axisymmetric Newton potential

The Radiation-MHD equations

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho v_i}{\partial x_i} = 0$$ Mass

$$\frac{\partial (\rho v_i)}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho v_i v_j + P_* I_{ij} - b_i b_j\right) = -\rho \frac{\partial \phi}{\partial x_i} + (\dot{\rho v_i})_\nu$$ Momentum

$$\frac{\partial E}{\partial t} + \frac{\partial}{\partial x_j}\left[\left(E + P_*\right) v_j - v_i b_i b_j\right] = -\rho v_i \frac{\partial \phi}{\partial x_i} + (\dot{\rho e})_\nu$$ Energy

$$\frac{\partial \rho Y_e}{\partial t} + \frac{\partial Y_e \rho v_i}{\partial x_i} = (\rho \dot{Y}_e)_\nu$$ Electron #

$\left(\boldsymbol{b} = \frac{\boldsymbol{B}}{\sqrt{4\pi}}\right)$ Magnetic flux

$$\frac{\partial b_i}{\partial t} = \frac{\partial}{\partial x_j}\left(v_i b_j - v_j b_i\right)$$

No monopoles

$$\boldsymbol{\nabla} \cdot \boldsymbol{b} = 0$$

$$E = \frac{1}{2}\rho v^2 + \rho e + \frac{b^2}{2}$$

$$P_* = p + \frac{b^2}{2}$$

EoS: $p = p(\rho, e, ...)$

The Radiation-MHD equations (2)

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho v_i}{\partial x_i} = 0$$ Mass

$$\frac{\partial (\rho v_i)}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho v_i v_j + P_* I_{ij} - b_i b_j\right) = -\rho \frac{\partial \phi}{\partial x_i} + \left(\dot{\rho v_i}\right)_\nu$$ Momentum

$$\frac{\partial E}{\partial t} + \frac{\partial}{\partial x_j}\left[\left(E + P_*\right) v_j - v_i b_i b_j\right] = -\rho v_i \frac{\partial \phi}{\partial x_i} + \left(\dot{\rho e}\right)_\nu$$ Energy

$$\frac{\partial \rho Y_e}{\partial t} + \frac{\partial Y_e \rho v_i}{\partial x_i} = \left(\rho \dot{Y}_e\right)_\nu$$ Electron #

$\left(\boldsymbol{b} = \frac{\boldsymbol{B}}{\sqrt{4\pi}}\right)$ Magnetic flux

$$\frac{\partial b_i}{\partial t} = \frac{\partial}{\partial x_j}\left(v_i b_j - v_j b_i\right)$$

No monopoles

$$\boldsymbol{\nabla} \cdot \boldsymbol{b} = 0$$

$$E = \frac{1}{2}\rho v^2 + \rho e + \frac{b^2}{2}$$

$$P_* = p + \frac{b^2}{2}$$

EoS: $p = p(\rho, e, ...)$

The Radiation-MHD equations (2)

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho v_i}{\partial x_i} = 0$$

Mass

$$\nabla^2 \phi = 4\pi G \rho$$

Poisson eq.

4-vector — 4-momentum — Distribution function

$$\left(\frac{\mathrm{d}x^\alpha}{\mathrm{d}\tau}\right)\frac{\partial f}{\partial x^\alpha} + \left(\frac{\mathrm{d}p^\alpha}{\mathrm{d}\tau}\right)\frac{\partial f}{\partial p^\alpha} = \left(\frac{\delta f}{\delta \tau}\right)_{\mathrm{coll}}$$

Proper time — Absorption, emission, scattering in medium

Relativistic Boltzmann eq. For each ν species

(Neutrinos massless! Propagate at speed of light c.)

Solution Algorithm: An Overview

i. MHD (FISH) Käppeli et al. 2011
   - Split hydro. and magnetic variables update
   - Dimensional splitting: solves eqs in 1D
   - Uses dim.-split constrained transport for $\nabla \cdot \boldsymbol{b} = 0$

ii. Radiative transfer (ELEPHANT) Liebendörfer et al. 2009
   - Full transfer NOT feasible in 3D (3+3+1=7 dim. problem!)
   - Approximation: Isotropic Diffusion Source Approx. (IDSA)

iii. Coupling: Radiation-MHD (FISH+ELEPHANT)

Outline

i. Stellar explosions
   - A (very) brief introduction to what we do and why we are doing it

ii. Methods & Algorithms
   - A quick overview of the origin of the computational work

iii. Implementation & Parallelisation
   - How the work is distributed to computational units

Implementation details

i. Directional operator splitting

$\Longrightarrow$ Write only 1D update routines

ii. Data rotated so that stencil OPs along contiguous memory direction

iii. Distributed memory parallelisation with MPI

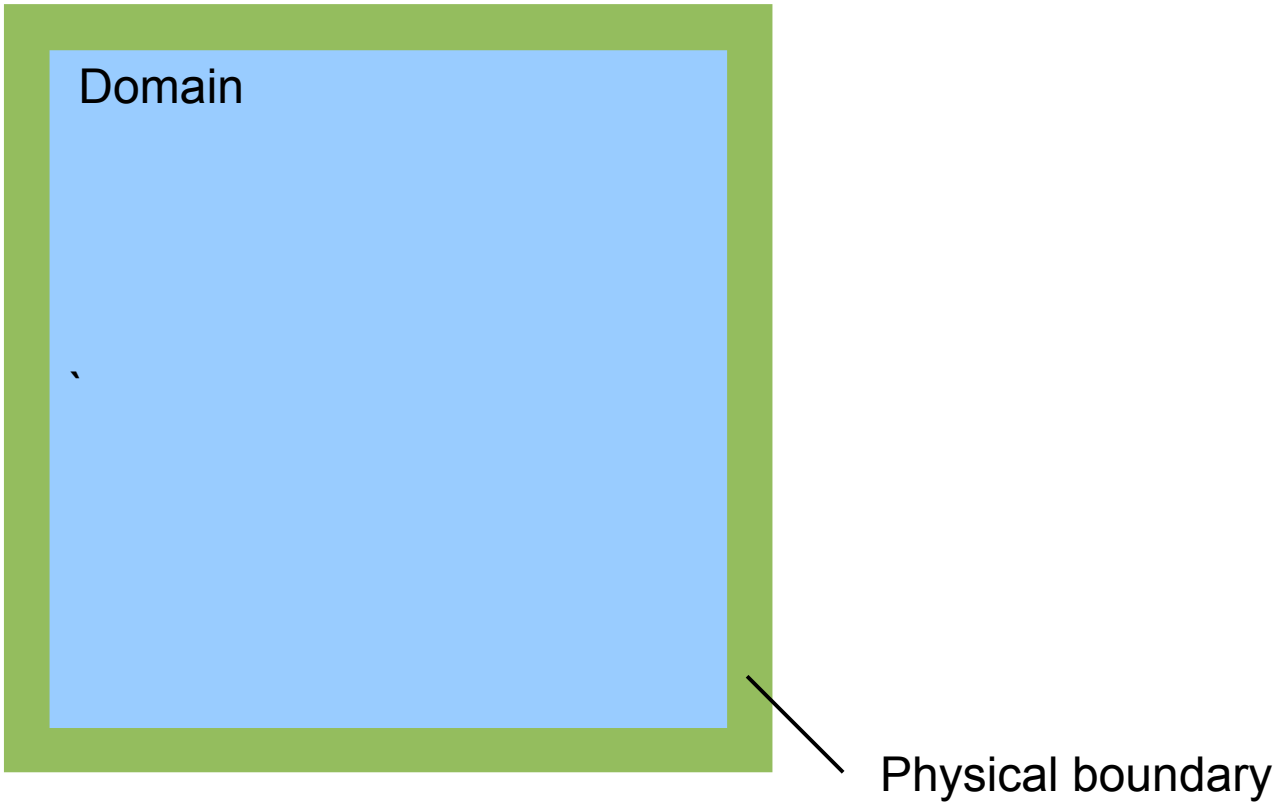
iv. Overlap communication/computation with non-blocking communication: persistent communications

In theory... see e.g. Schubert et al. 2011

v. Shared memory parallelisation with OpenMP

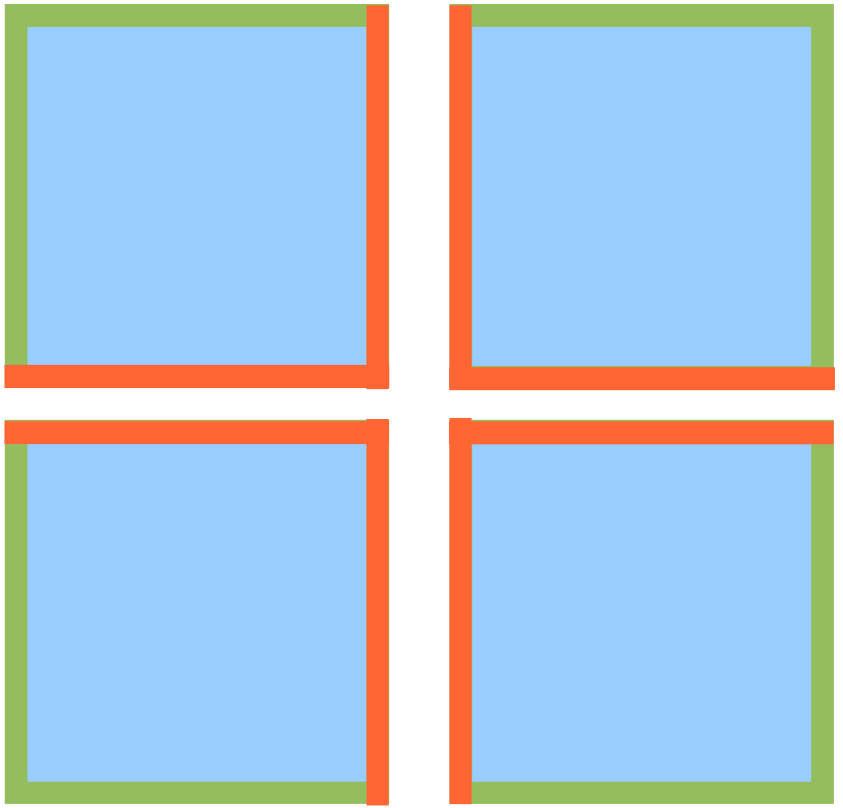
Parallelisation: MPI

i. Domain decomposition



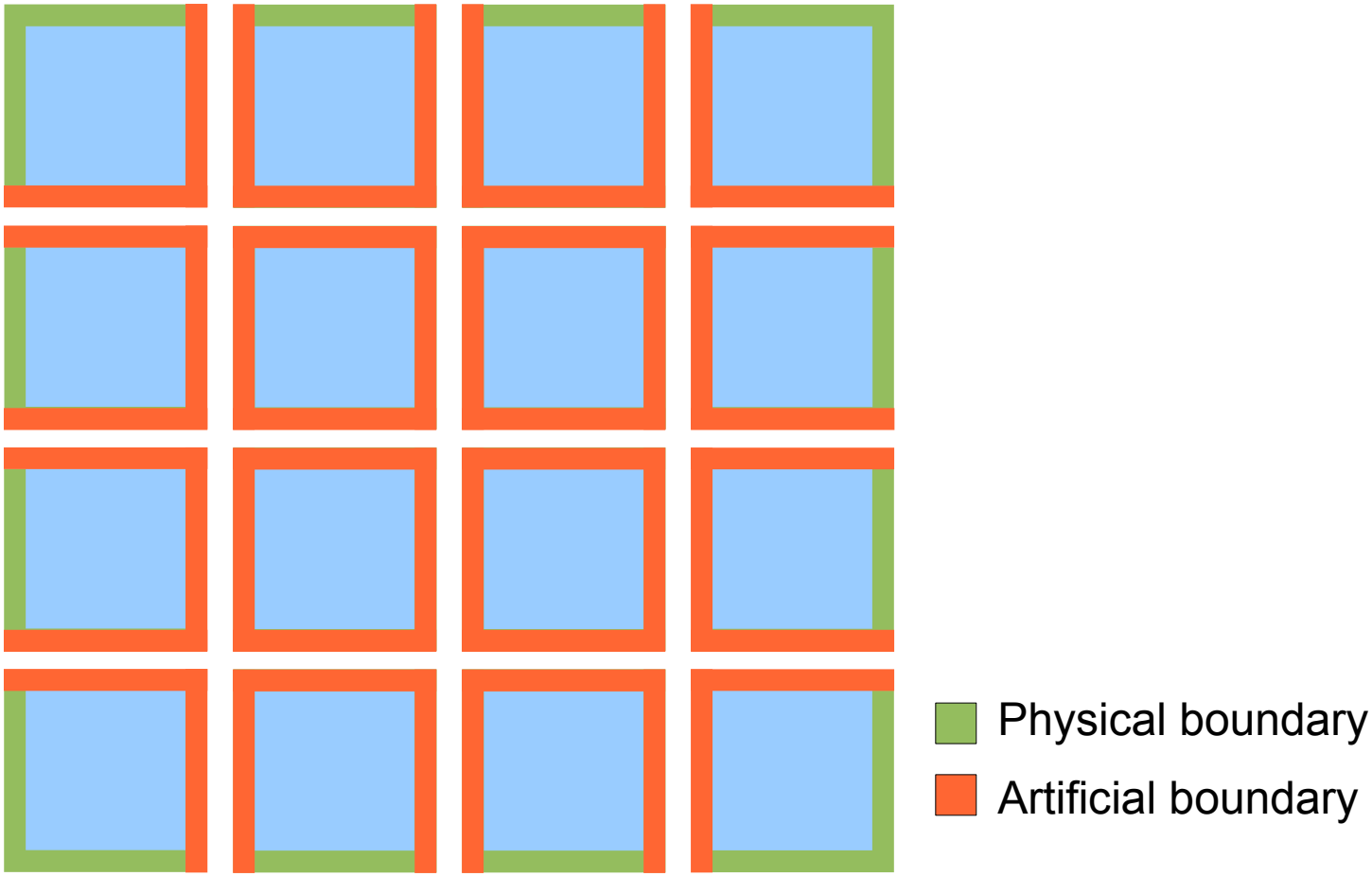
Parallelisation: MPI

i. Domain decomposition



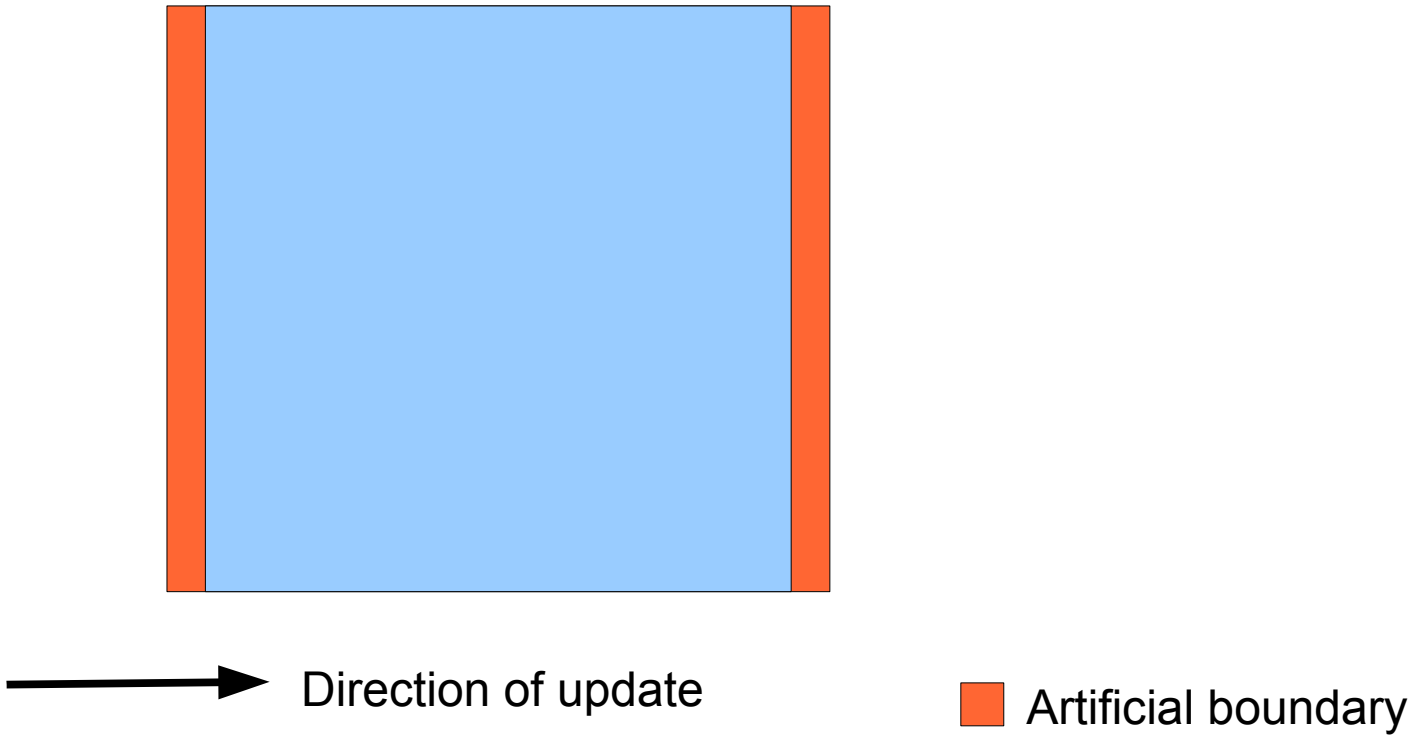
Parallelisation: MPI

i. Domain decomposition



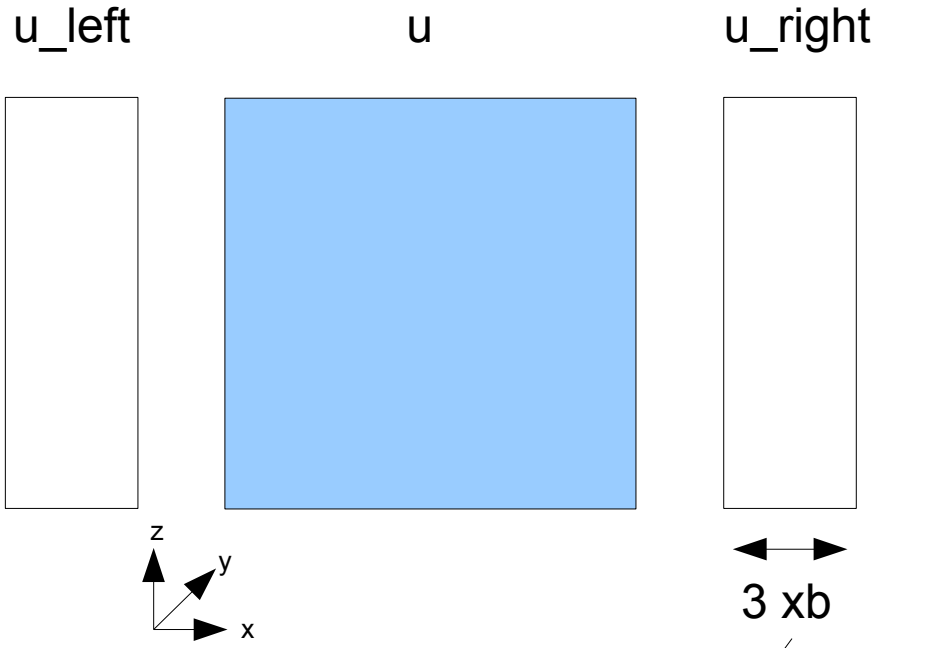
Parallelisation: MPI

i. Domain decomposition



Parallelisation: MPI

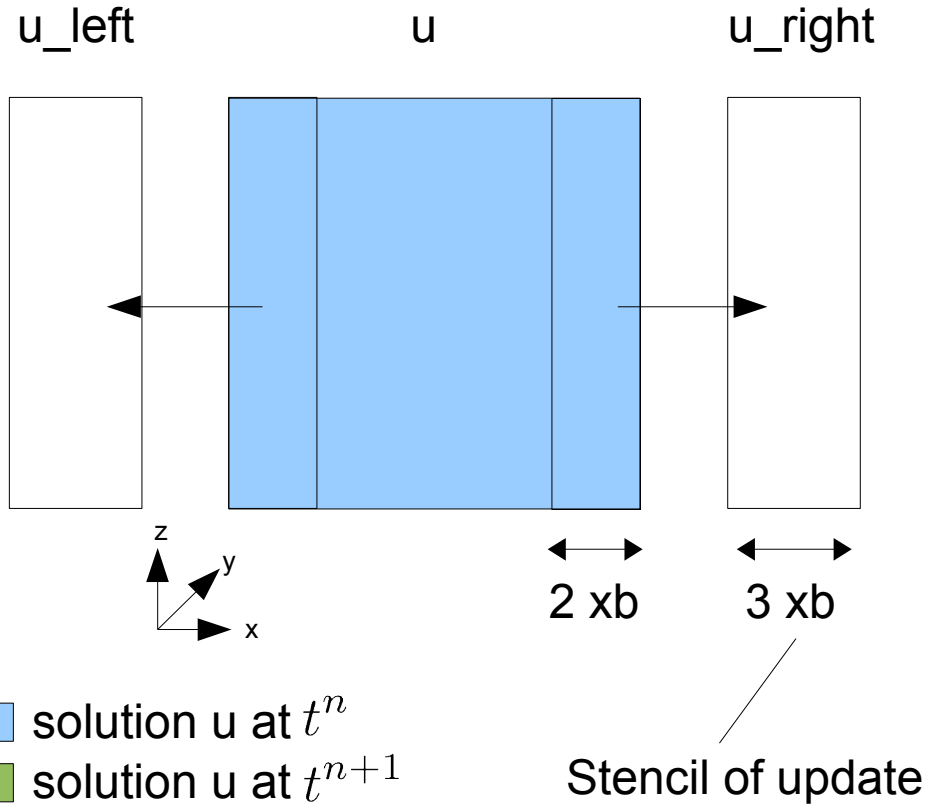
i. Overlapping communication/computation




```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

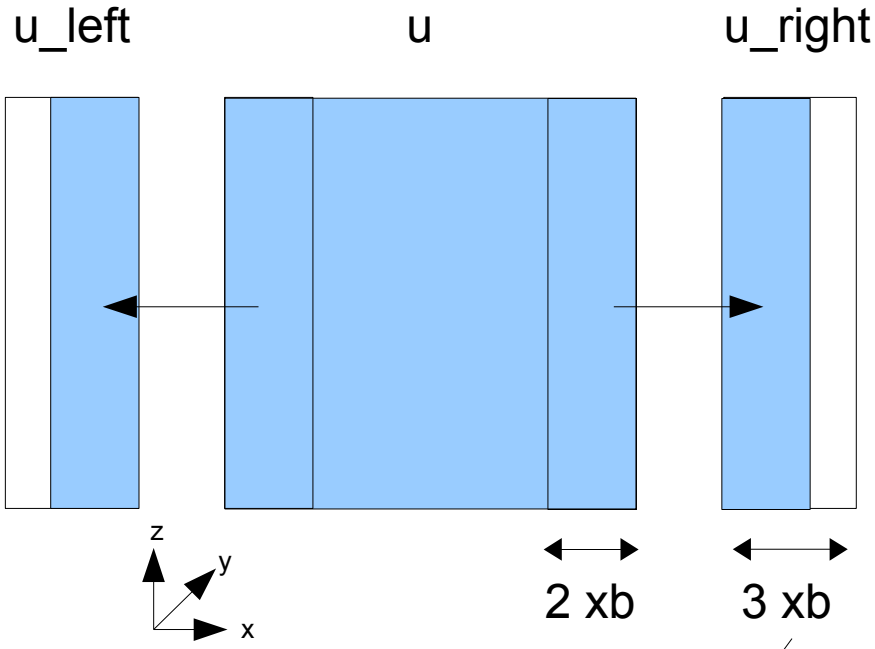
i. Overlapping communication/computation




```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

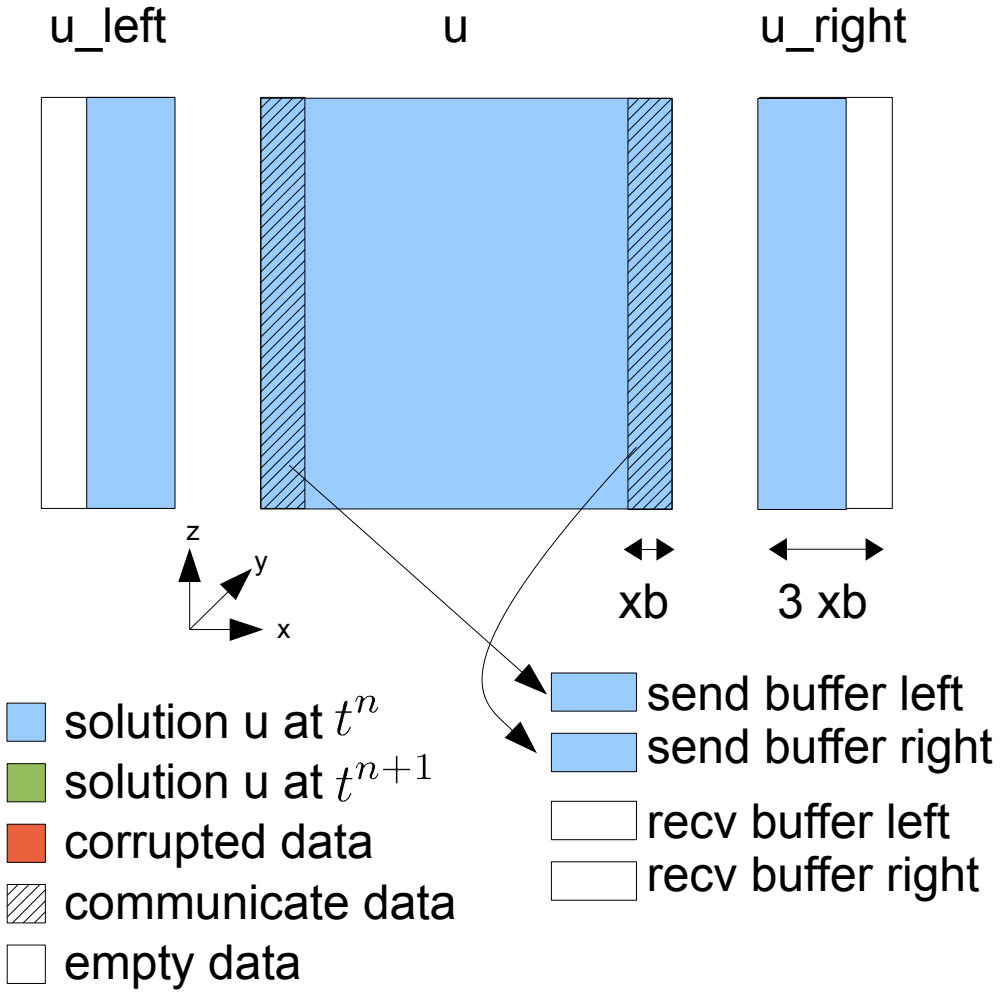
i. Overlapping communication/computation




```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

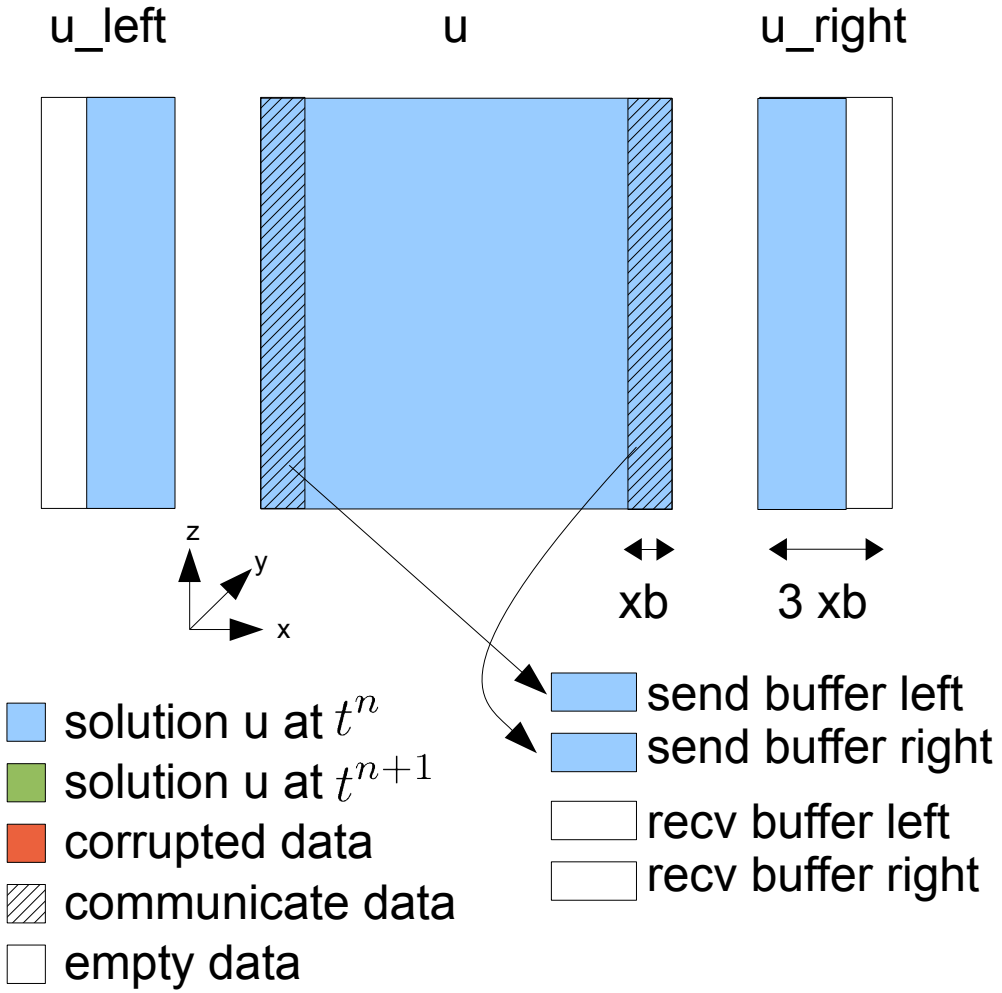
i. Overlapping communication/computation




```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

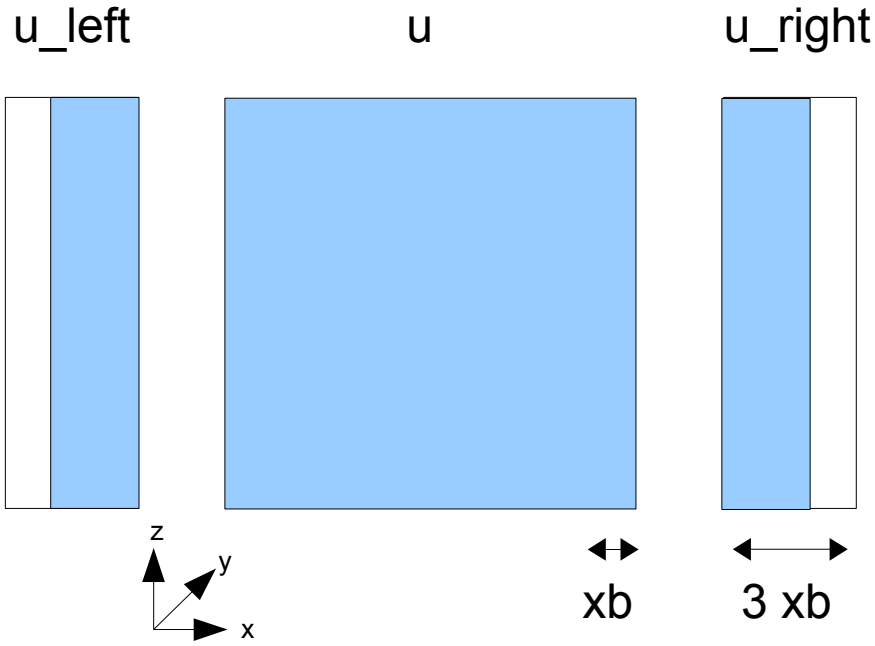
i. Overlapping communication/computation




```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

i. Overlapping communication/computation



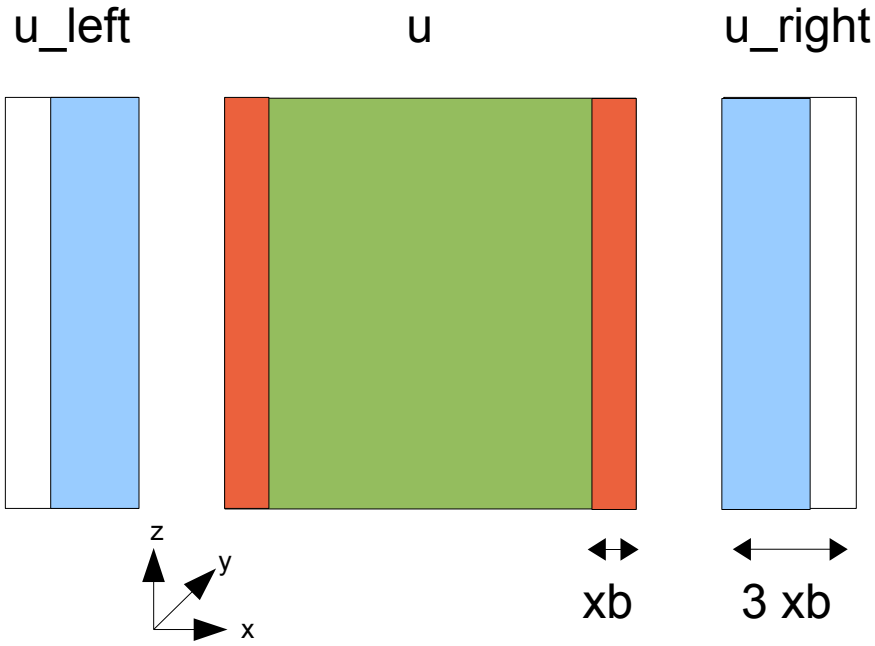
- solution u at t^n
- solution u at t^{n+1}
- corrupted data
- communicate data
- empty data

- send buffer left
- send buffer right
- recv buffer left
- recv buffer right

```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

i. Overlapping communication/computation



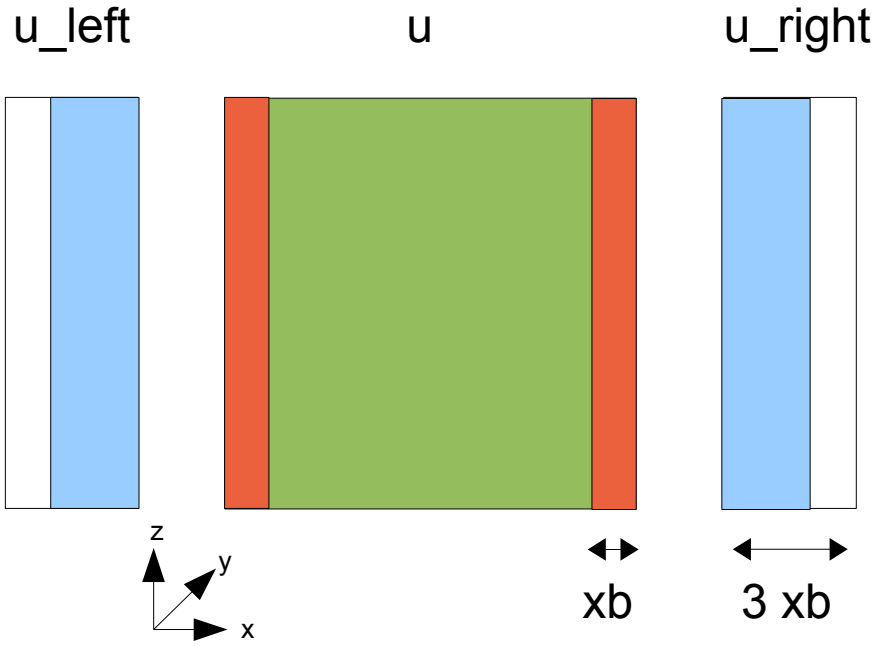
- solution u at t^n
- solution u at t^{n+1}
- corrupted data
- communicate data
- empty data

- send buffer left
- send buffer right
- recv buffer left
- recv buffer right

```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

i. Overlapping communication/computation



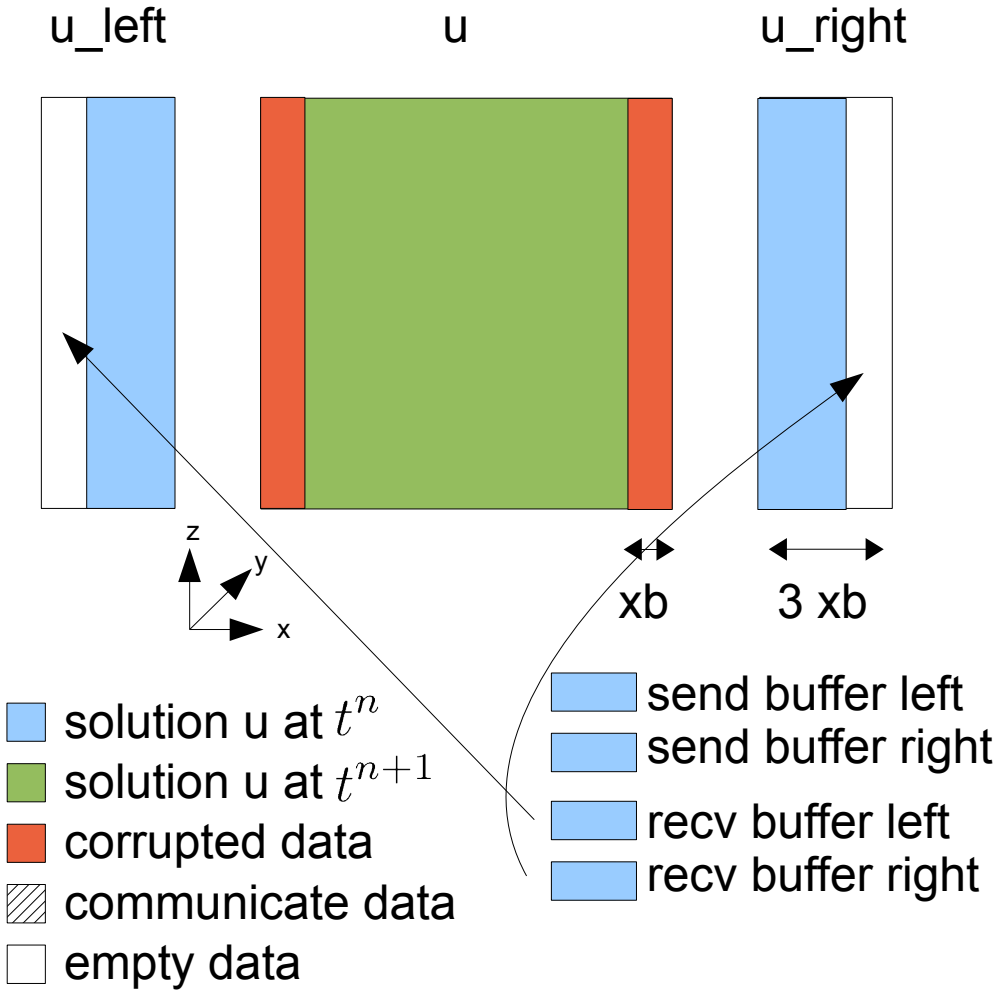
- solution u at t^n
- solution u at t^{n+1}
- corrupted data
- communicate data
- empty data

- send buffer left
- send buffer right
- recv buffer left
- recv buffer right

```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

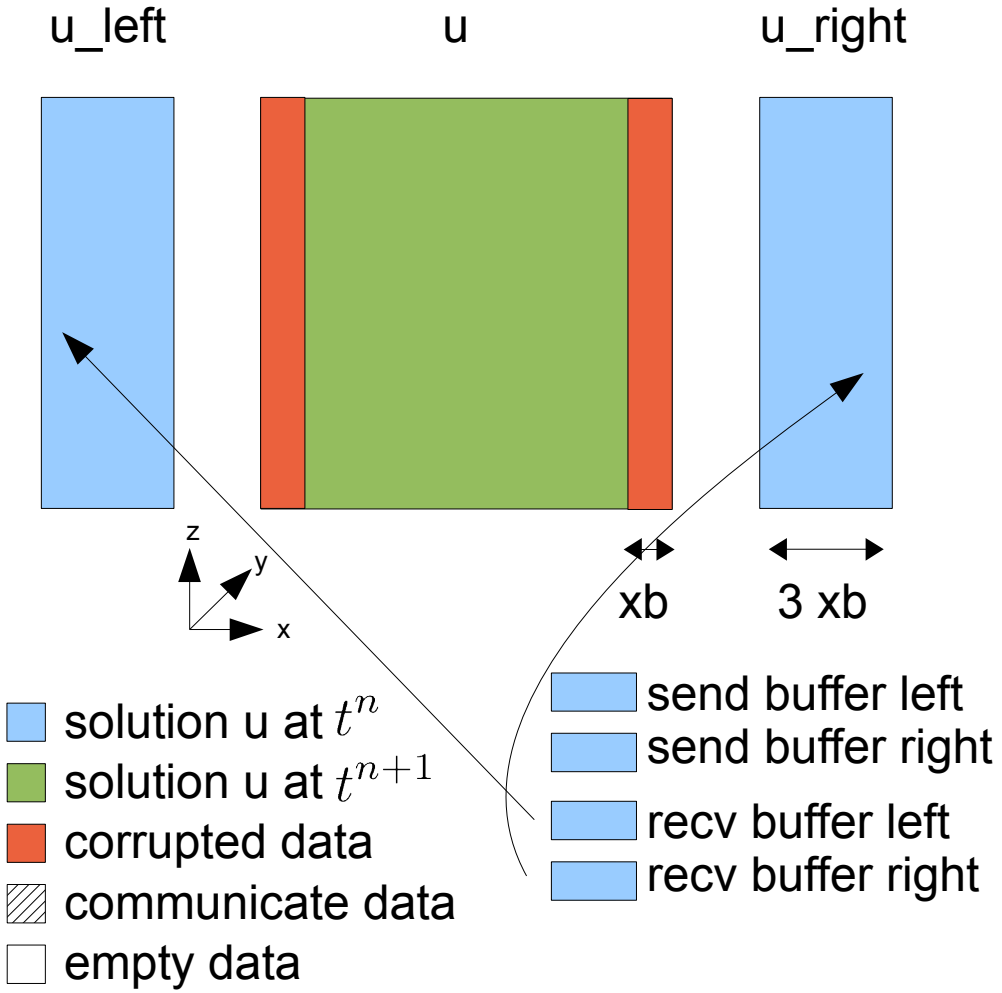
i. Overlapping communication/computation




```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

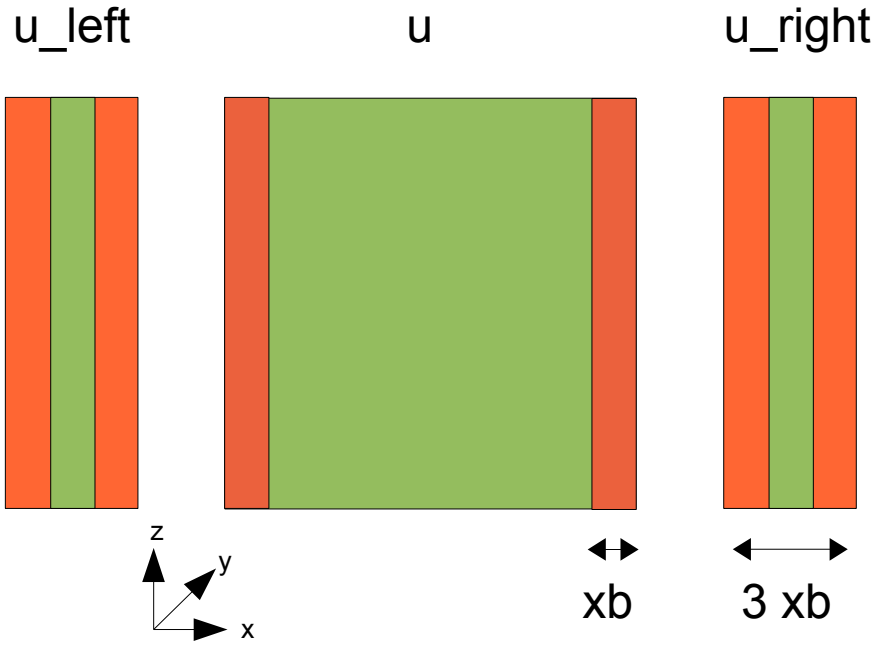
i. Overlapping communication/computation




```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

i. Overlapping communication/computation



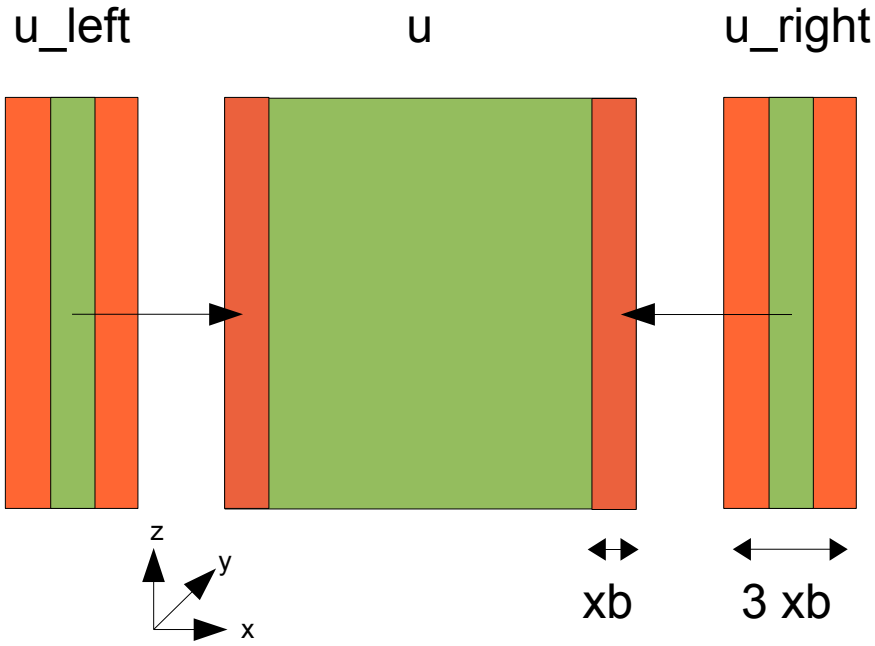
- solution u at t^n
- solution u at t^{n+1}
- corrupted data
- communicate data
- empty data

- send buffer left
- send buffer right
- recv buffer left
- recv buffer right

```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

i. Overlapping communication/computation



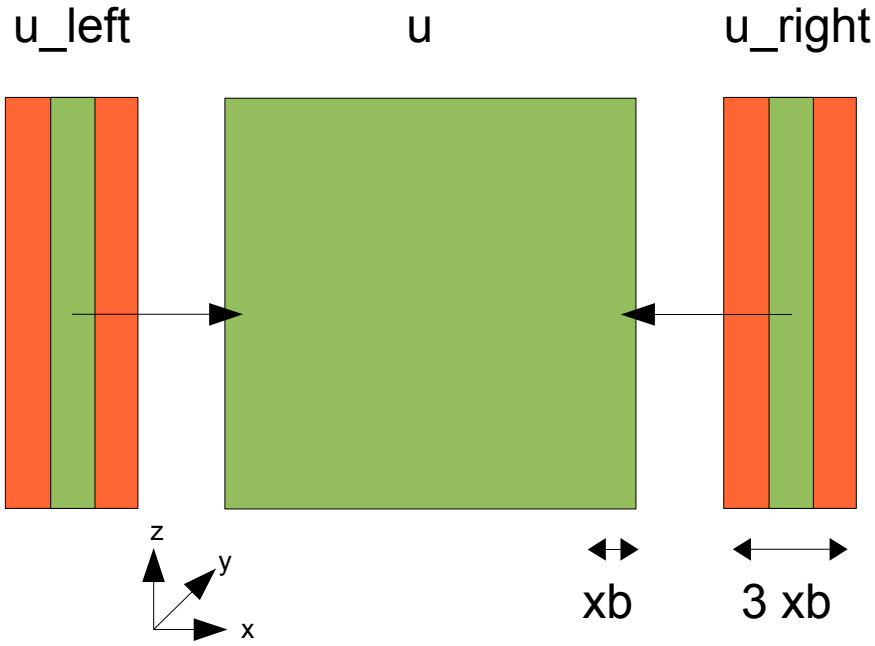
- solution u at t^n
- solution u at t^{n+1}
- corrupted data
- communicate data
- empty data

- send buffer left
- send buffer right
- recv buffer left
- recv buffer right

```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

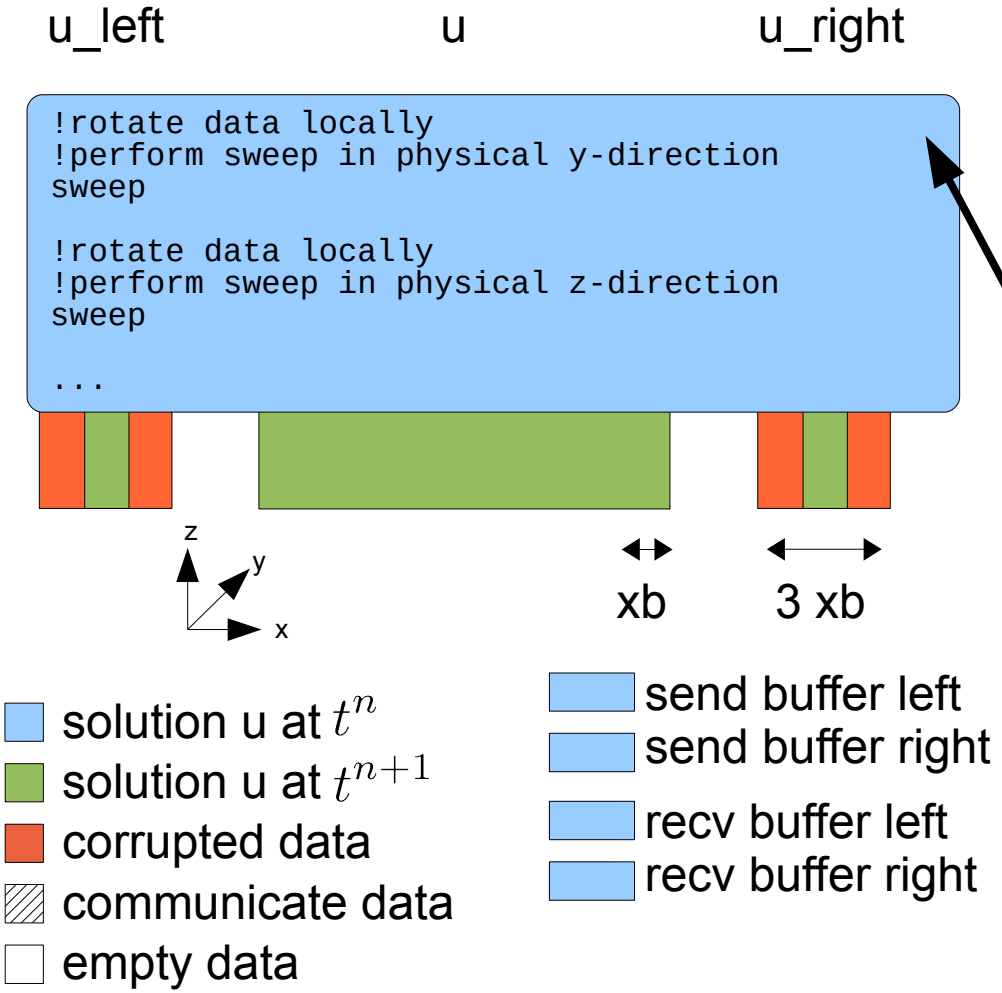
i. Overlapping communication/computation




```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

Parallelisation: MPI

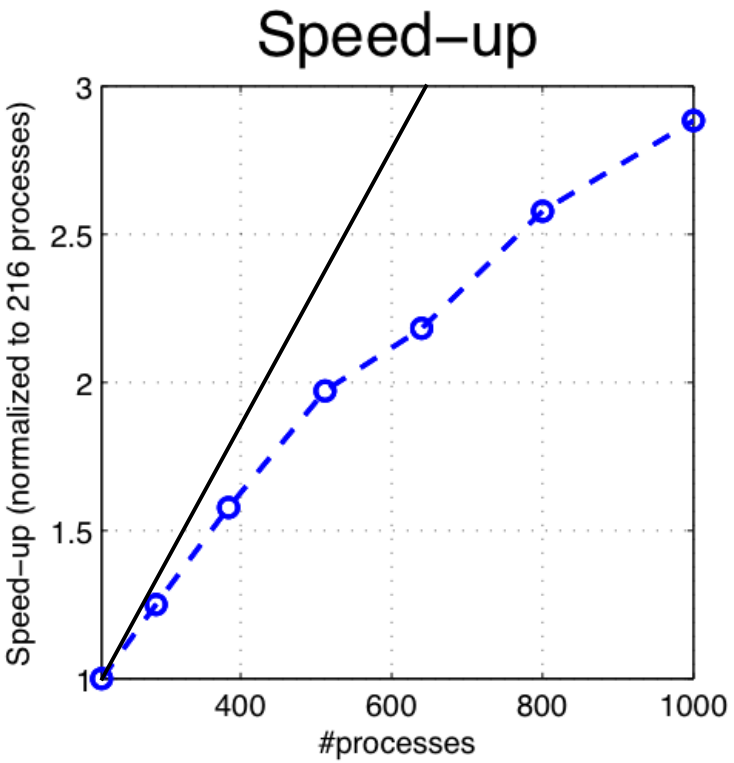
i. Overlapping communication/computation

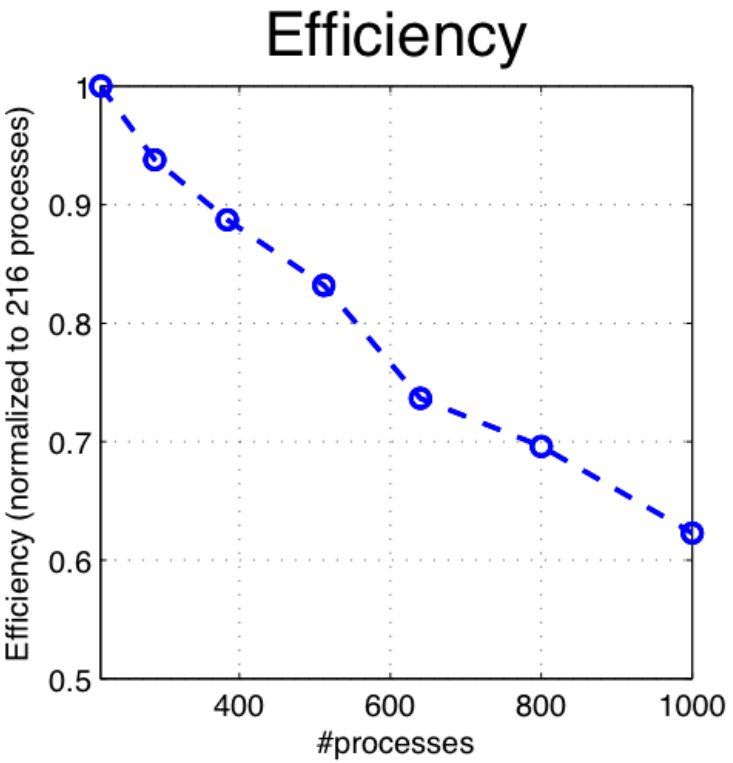



```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

(Strong) Scaling: MPI

FISH (MHD): 600x600x600 zones

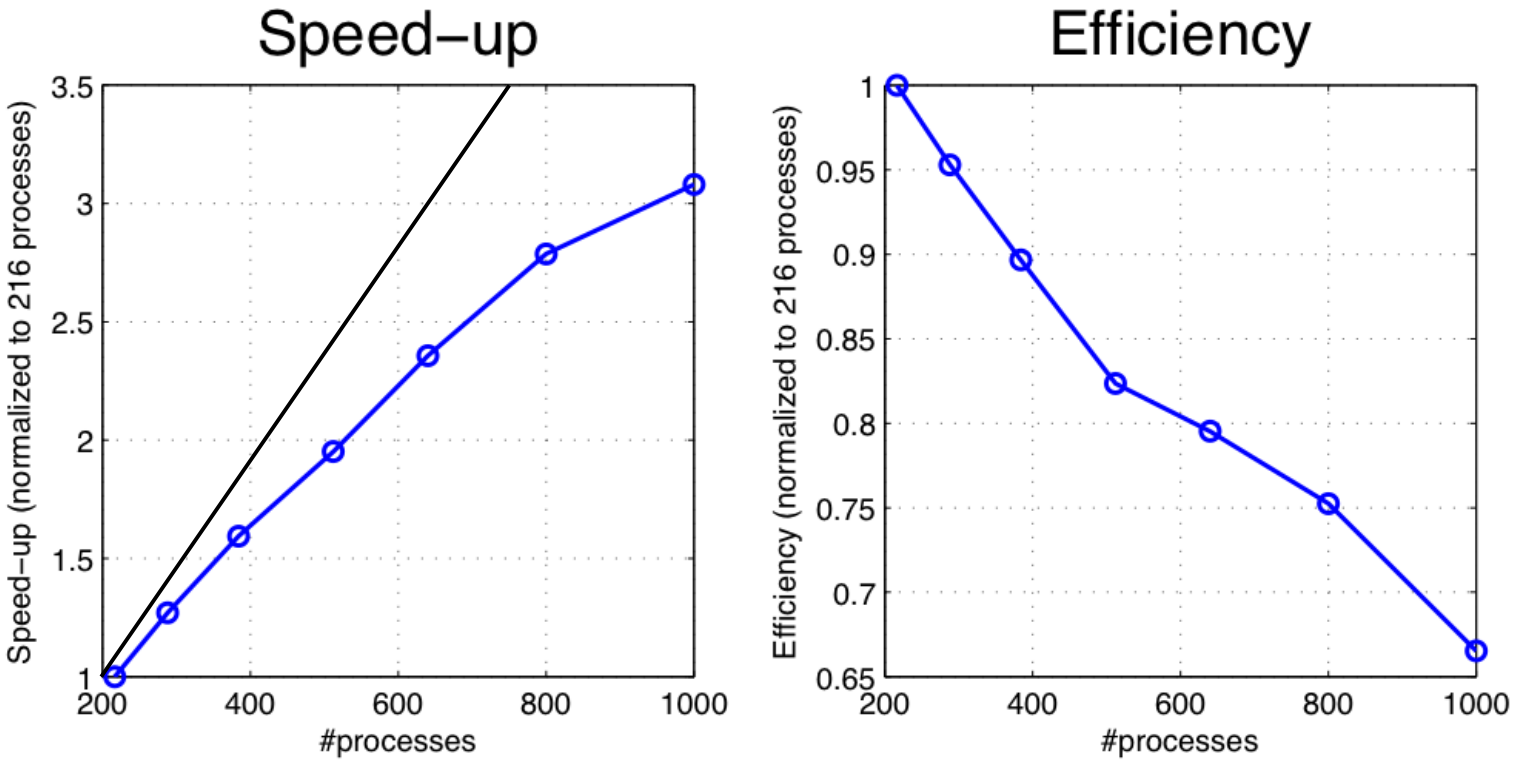




Performed on Cray XT5 @ CSCS

(Strong) Scaling: MPI (2)

FISH+ELEPHANT (Radiation+MHD): 600x600x600 zones



Performed on Cray XT5 @ CSCS

Performance?

i. Decrease in efficiency mainly due to:

- Large stencil

  $\Rightarrow$ Modify algorithm? Yes! BUT the decrease is (just) postponed...

- Overhead from overlapping comm./computation finally dominates

  $\Rightarrow$ Use OpenMP on nodes and MPI for inter-node communication?

Parallelisation: MPI+OpenMP

i. Overlapping communication/computation

```
!update

!series of loops



do k=1,nz
  do j=1,ny
    call update_1(u(:,j,k),nx)
  enddo
enddo




do k=1,nz
  do j=1,ny
    call update_2(u(:,j,k),nx)
  enddo
enddo


...
```

u_lef … _right

xb

update

solu

solu

corrupted data

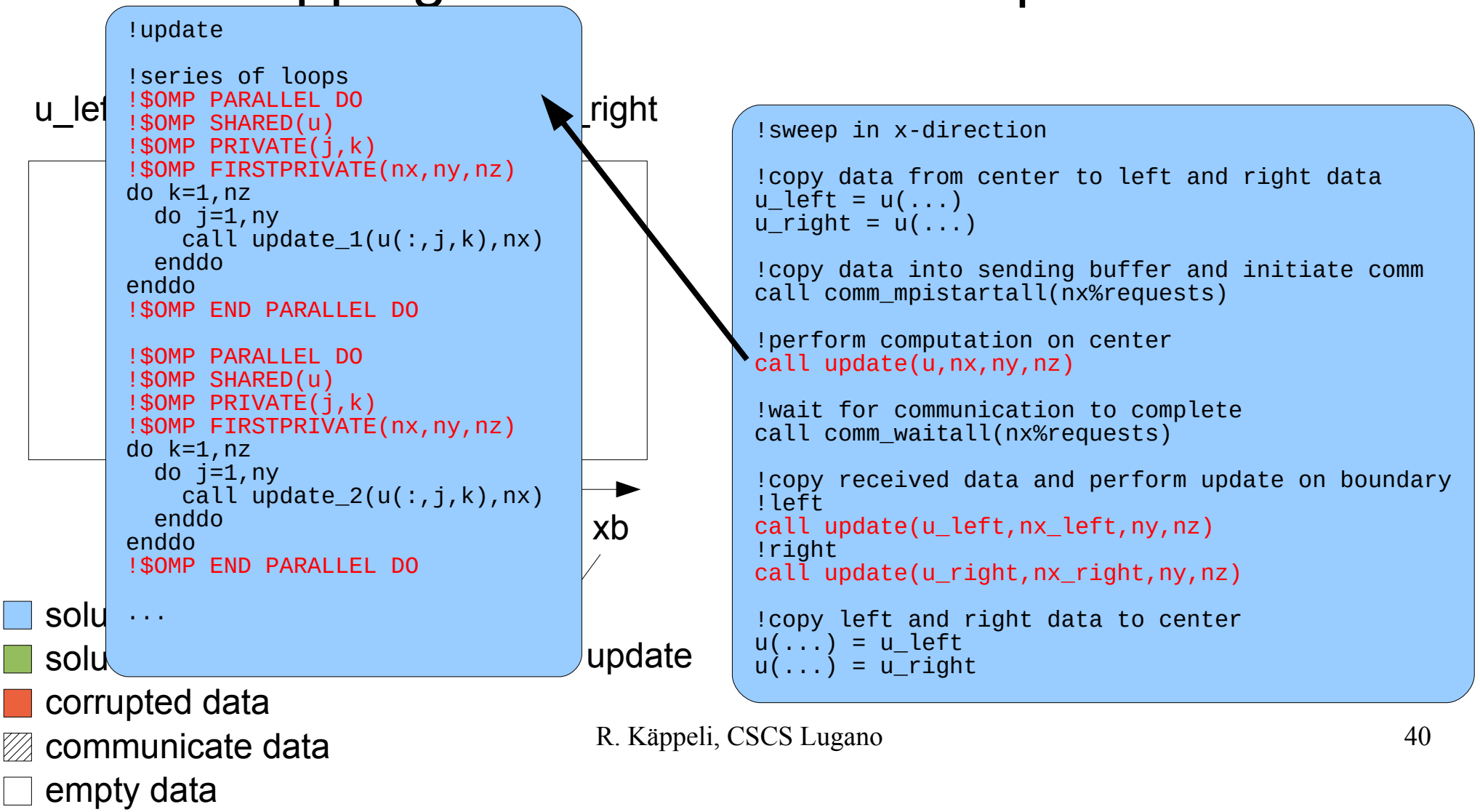
communicate data

empty data

```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)

!copy left and right data to center
u(...) = u_left
u(...) = u_right
```

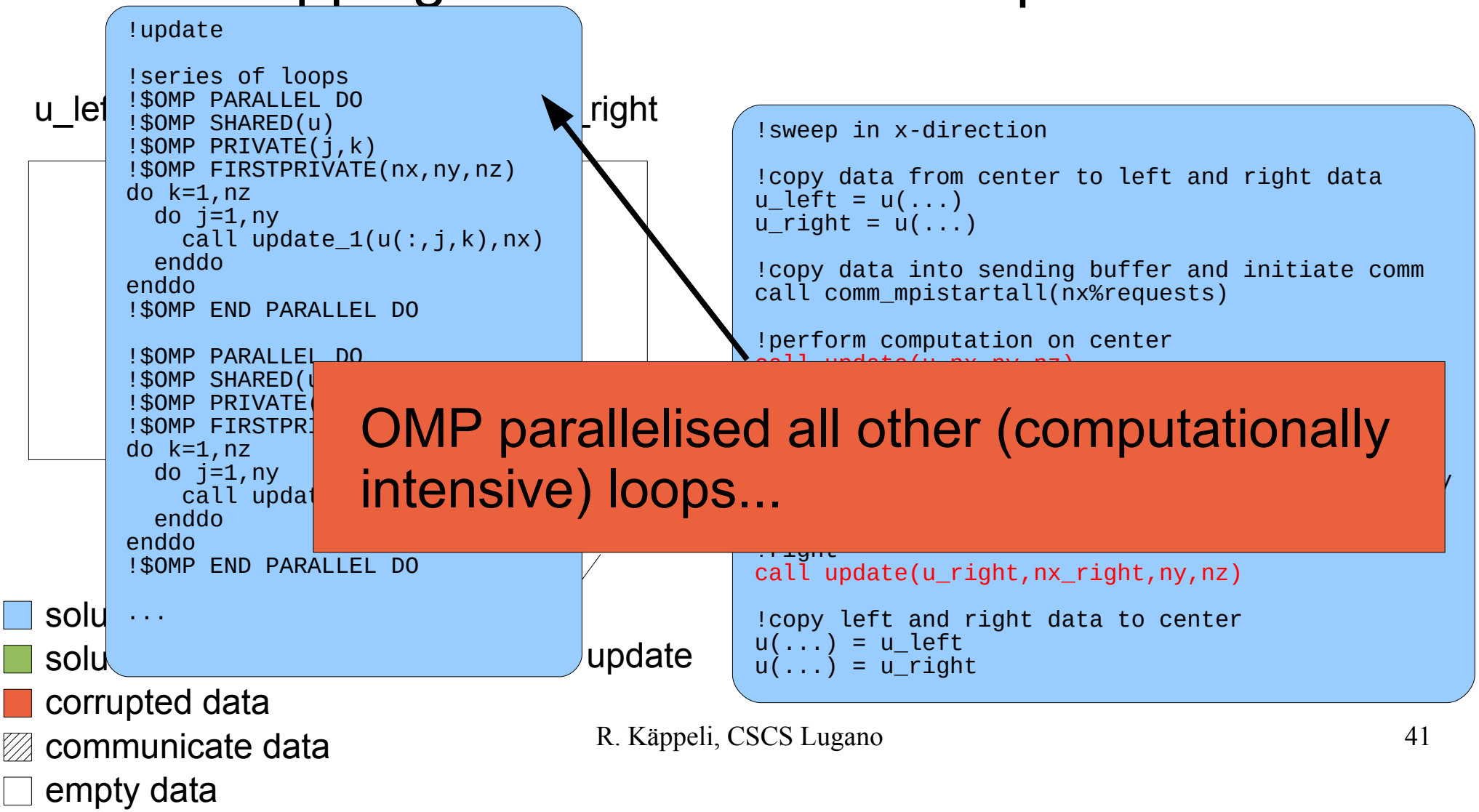
Parallelisation: MPI+OpenMP

i. Overlapping communication/computation



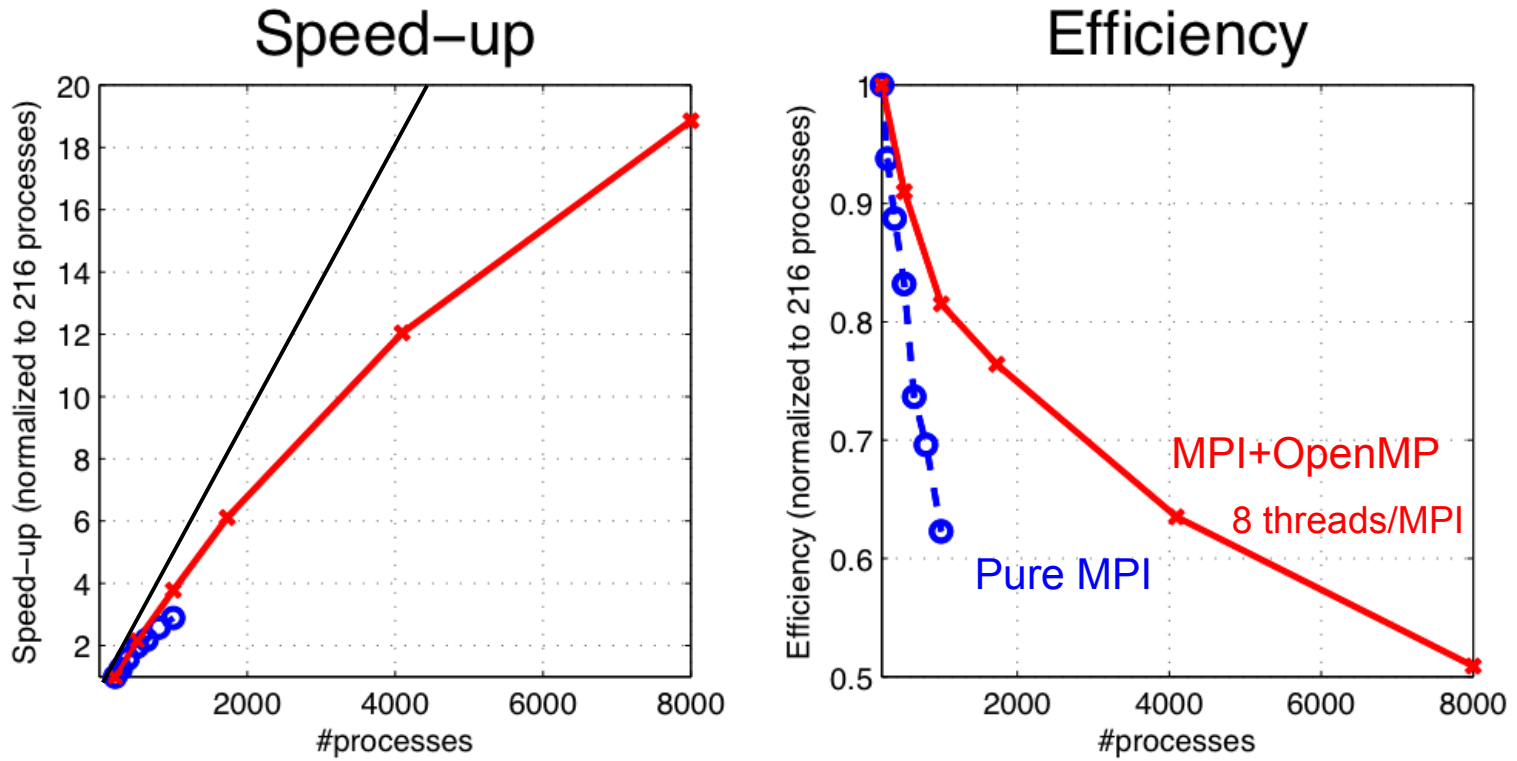
Parallelisation: MPI+OpenMP

i. Overlapping communication/computation



(Strong) Scaling: MPI+OpenMP

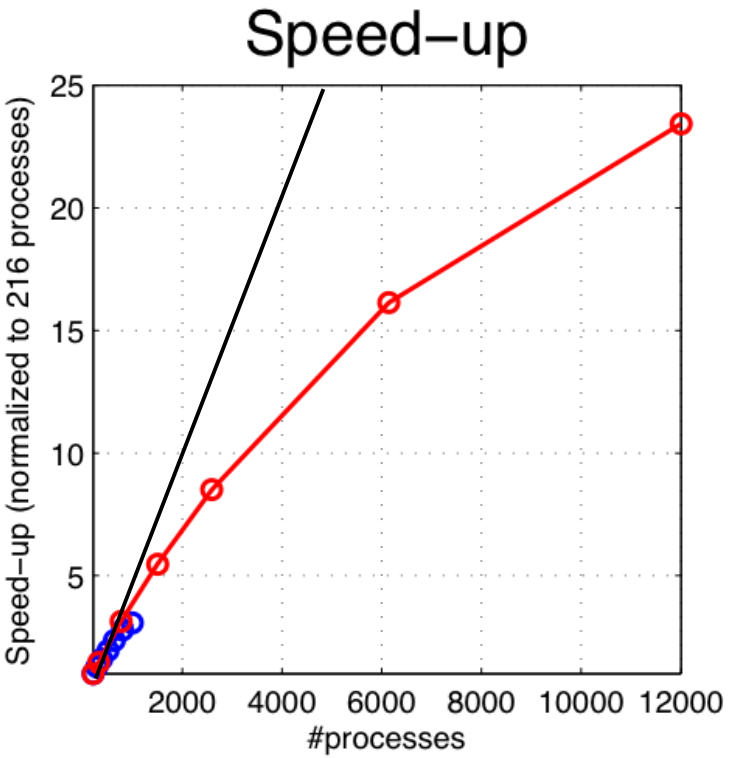
FISH (MHD): 600x600x600 zones

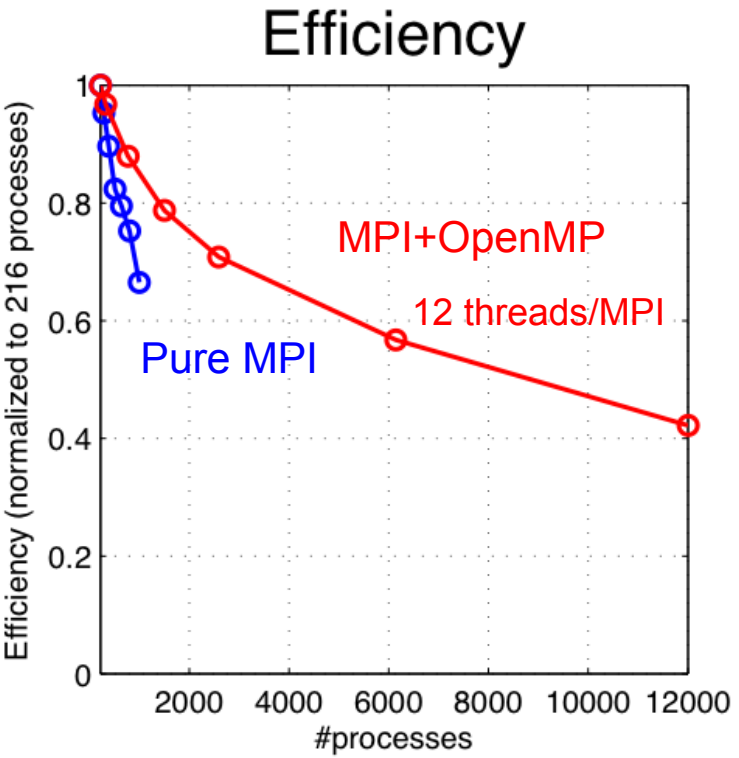


Performed on Cray XT5 @ CSCS

(Strong) Scaling: MPI+OpenMP (2)

FISH+ELEPHANT (Radiation+MHD): 600x600x600 zones





Performed on Cray XT5 @ CSCS

OpenMP

i. May introduce additional problems (race conditions...)
   - Compare results carefully **and**/or use software tools

ii. Performance issues
   - (False) sharing bad
   - Overhead (parallel region initialisation)

iii. Minimize serial part(s)
   - Running at (1/#cores) efficiency in serial part!

OpenMP

i. May introduce additional problems (race conditions...)
   - Compare results carefully **and**/or use software tools

ii. Performance issues
   - (False) sharing bad
   - Overhead (parallel region initialisation)

iii. Minimize serial part(s)
   - Running at (1/#cores) efficiency in serial part!

OpenMP

i. May introduce additional problems (race conditions...)

- efully **and**/or use software tools

ii. P

- egior

iii. M s)

- s) eff

```
!update

!series of loops
!$OMP PARALLEL DO IF (.not. small)
!$OMP SHARED(u)
!$OMP PRIVATE(j,k)
!$OMP FIRSTPRIVATE(nx,ny,nz)
do k=1,nz
  do j=1,ny
    call update_1(u(:,j,k),nx)
  enddo
enddo
!$OMP END PARALLEL DO

!$OMP PARALLEL DO IF (.not. small)
!$OMP SHARED(u)
!$OMP PRIVATE(j,k)
!$OMP FIRSTPRIVATE(nx,ny,nz)
do k=1,nz
  do j=1,ny
    call update_2(u(:,j,k),nx)
  enddo
enddo
!$OMP END PARALLEL DO

...
```

```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)
```

Overhead dominates!

OpenMP

i. May introduce additional problems (race conditions...)
   - Compare results carefully **and**/or use software tools

ii. Performance issues
   - (False) sharing bad
   - Overhead (parallel region initialisation)

iii. Minimize serial part(s)
   - Running at (1/#cores) efficiency in serial part!

OpenMP

i. May introduce additional problems (race conditions...)

- efully **and**/or use software tools

ii. P

- egior

iii. M s)

- s) eff

```
!update

!series of loops
!$OMP PARALLEL DO IF (.not. small)
!$OMP SHARED(u)
!$OMP PRIVATE(j,k)
!$OMP FIRSTPRIVATE(nx,ny,nz)
do k=1,nz
  do j=1,ny
    call update_1(u(:,j,k),nx)
  enddo
enddo
!$OMP END PARALLEL DO

!$OMP PARALLEL DO IF (.not. small)
!$OMP SHARED(u)
!$OMP PRIVATE(j,k)
!$OMP FIRSTPRIVATE(nx,ny,nz)
do k=1,nz
  do j=1,ny
    call update_2(u(:,j,k),nx)
  enddo
enddo
!$OMP END PARALLEL DO

...
```

```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)
```

Overhead dominates!
- Yes, but SERIAL!

OpenMP

i. May introduce additional problems (race conditions...)

- efully **and**/or use software tools

ii. P

- egior

iii. M s)

- s) eff

```
!update

!series of loops
!$OMP PARALLEL DO IF (.not. small)
!$OMP SHARED(u)
!$OMP PRIVATE(j,k)
!$OMP FIRSTPRIVATE(nx,ny,nz)
do k=1,nz
  do j=1,ny
    call update_1(u(:,j,k),nx)
  enddo
enddo
!$OMP END PARALLEL DO

!$OMP PARALLEL DO IF (.not. small)
!$OMP SHARED(u)
!$OMP PRIVATE(j,k)
!$OMP FIRSTPRIVATE(nx,ny,nz)
do k=1,nz
  do j=1,ny
    call update_2(u(:,j,k),nx)
  enddo
enddo
!$OMP END PARALLEL DO

...
```

```
!sweep in x-direction

!copy data from center to left and right data
u_left = u(...)
u_right = u(...)

!copy data into sending buffer and initiate comm
call comm_mpistartall(nx%requests)

!perform computation on center
call update(u,nx,ny,nz)

!wait for communication to complete
call comm_waitall(nx%requests)

!copy received data and perform update on boundary
!left
call update(u_left,nx_left,ny,nz)
!right
call update(u_right,nx_right,ny,nz)
```

Overhead dominates!
- Yes, but SERIAL!
- However, it's better!!!

OpenMP

i. May introduce additional problems (race conditions...)
   - Compare results carefully **and**/or use software tools

ii. Performance issues
   - (False) sharing bad
   - Overhead (parallel region initialisation)

iii. Minimize serial part(s)

Competition!

   - Running at (1/#cores) efficiency in serial part!

OpenMP

i. May introduce additional problems (race conditions...)
   - Compare results carefully **and**/or use software tools

ii. Performance issues
   - (False) sharing bad
   - Overhead (parallel region initialisation)

iii. Minimize serial part(s)

Competition!

   - Running at (1/#cores) efficiency in serial part!

The End, Thanks!