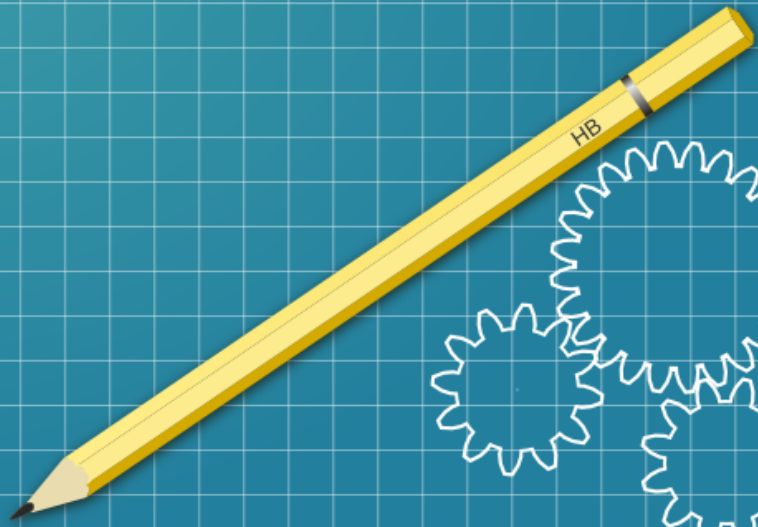
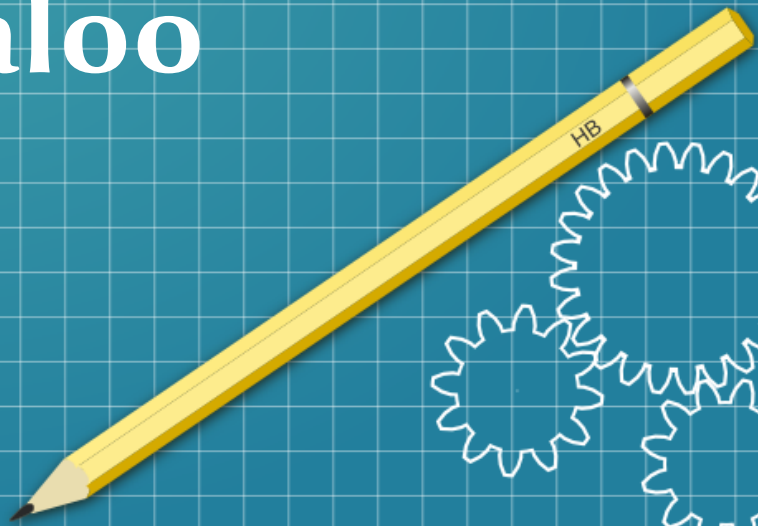
IDEFIX USER DAYS
TUTORIAL: USAGE II



Marc Van den Bossche

IDEFIX USER DAYS TUTORIAL: USAGE II

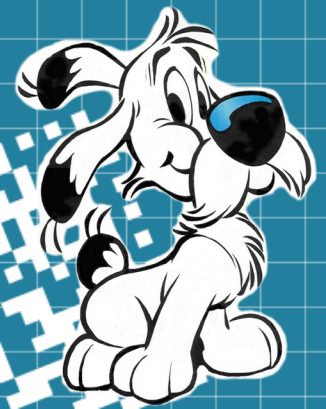
Electric boogaloo

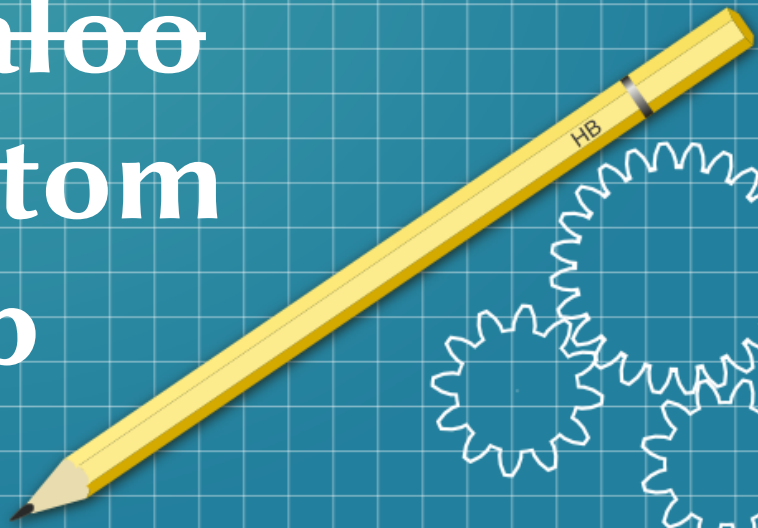


Marc Van den Bossche

IDEFIX USER DAYS
TUTORIAL: USAGE II

~~Electric boogaloo~~
Creating a custom Idefix setup





Marc Van den Bossche

Custom setup

Required files in your “problem directory”

- `definitions.hpp`
- `idefix.ini`
- `setup.cpp`

Custom setup

Required files in your "problem directory"

- `definitions.hpp`
- `idefix.ini`
- `setup.cpp`

setup.cpp file

What is its purpose ?

setup.cpp file

What is its purpose ?

- Initial state of the simulation
- Problem-specific boundary conditions
- “Internal boundaries”: density floor, velocity limiters…
- Custom gravitational potential
- “Source terms”: local cooling function, wave killing zone…
- Local viscosity and diffusivities
- Custom outputs

setup.cpp file

How to do it ?

→ two default class methods

```
// from setup.hpp (doesn't need to be changed)
class Setup {
  public:
    Setup(Input &, Grid &, DataBlock &, Output &);
    void InitFlow(DataBlock &);
  };
```

setup.cpp file

How to do it ?

→ two default class methods

```
// from setup.hpp (doesn't need to be changed)
class Setup {
  public:
    Setup(Input &, Grid &, DataBlock &, Output &);
    void InitFlow(DataBlock &);
  };
```

Constructor: where custom function will be enrolled: for custom boundaries, sources, potential...

setup.cpp file

How to do it ?

→ two default class methods

```
// from setup.hpp (doesn't need to be changed)
class Setup {
  public:
    Setup(Input &, Grid &, DataBlock &, Output &);
    void InitFlow(DataBlock &);
  };
```

Constructor: where custom function will be enrolled: for custom boundaries, sources, potential...

Initial condition of the simulation

How to use a custom parameters ?

`idefix.ini`: set boundary to `userdef`

`setup.cpp`: Function needs to be “enrolled” inside the constructor

How to use a custom parameters ?

`idefix.ini`: set boundary to `userdef`

`setup.cpp`: Function needs to be "enrolled" inside the constructor

```
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  // Set the function for userdefboundary
  data.hydro->EnrollUserDefBoundary(&MyBoundary);
  data.hydro->EnrollInternalBoundary(&MyInternalBoundary);

  output.EnrollUserDefVariables(&ComputeUserVars);

  gammaGlob=data.hydro->GetGamma();
  epsilonGlob = input.Get<real>("Setup","epsilon",0);
  countGlob = input.Get<int>("Setup","count",0);
}
```

How to use a custom parameters ?

`idefix.ini`: set boundary to `userdef`

`setup.cpp`: Function needs to be "enrolled" inside the constructor

```
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  // Set the function for userdefboundary
  data.hydro->EnrollUserDefBoundary(&MyBoundary);
  data.hydro->EnrollInternalBoundary(&MyInternalBoundary);

  output.EnrollUserDefVariables(&ComputeUserVars);

  gammaGlob=data.hydro->GetGamma();
  epsilonGlob = input.Get<real>("Setup","epsilon",0);
  countGlob = input.Get<int>("Setup","count",0);
}
```

Custom functions

How to use a custom parameters ?

`idefix.ini`: set boundary to `userdef`

`setup.cpp`: Function needs to be "enrolled" inside the constructor

```
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  // Set the function for userdefboundary
  data.hydro->EnrollUserDefBoundary(&MyBoundary);
  data.hydro->EnrollInternalBoundary(&MyInternalBoundary);

  output.EnrollUserDefVariables(&ComputeUserVars);

  gammaGlob=data.hydro->GetGamma();
  epsilonGlob = input.Get<real>("Setup","epsilon",0);
  countGlob = input.Get<int>("Setup","count",0);
}
```

Custom functions

Fetching parameters from the `idefix.ini`

```
[Setup]
epsilon         0.1
count           10
```

How to use a custom parameters ?

`idefix.ini`: set boundary to `userdef`

`setup.cpp`: Function needs to be "enrolled" inside the constructor

```
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  // Set the function for userdefboundary
  data.hydro->EnrollUserDefBoundary(&MyBoundary);
  data.hydro->EnrollInternalBoundary(&MyInternalBoundary);

  output.EnrollUserDefVariables(&ComputeUserVars);

  gammaGlob=data.hydro->GetGamma();
  epsilonGlob = input.Get<real>("Setup","epsilon",0);
  countGlob = input.Get<int>("Setup","count",0);
}
```

Run-time linking

Custom functions

Fetching parameters from the `idefix.ini`

```
[Setup]
epsilon         0.1
count           10
```

Specifying the initial state

```
void Setup::InitFlow(DataBlock &data) {
    // Create a host copy of the dataBlock
    for(/* loop on last dimension */) {
        for(/* loop on second dimension */) {
            for(/* loop on first dimension */) {
         // density = ...
         // velocity_1 = ...
         // velocity_2 = ...
         // velocity_3 = ...
         // pressure = ...
            }
        }
    }
    // Send the dataBlock to device
}
```

Specifying the initial state

```
void Setup::InitFlow(DataBlock &data) {
    // Create a host copy of the dataBlock
    DataBlockHost d(data);
    for(/* loop on last dimension */) {
        for(/* loop on second dimension */) {
            for(/* loop on first dimension */) {
         // density = ...
         // velocity_1 = ...
         // velocity_2 = ...
         // velocity_3 = ...
         // pressure = ...
            }
        }
    }
    // Send the dataBlock to device
}
```

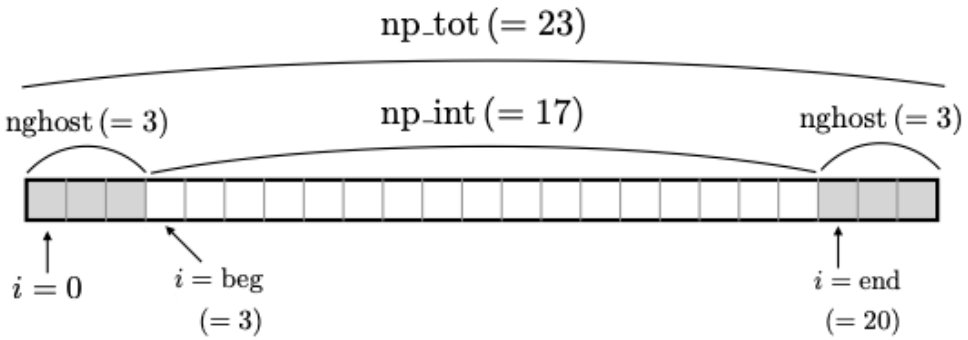
Specifying the initial state

```
void Setup::InitFlow(DataBlock &data) {
    // Create a host copy of the dataBlock
    DataBlockHost d(data);
    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
        // density = ...
        // velocity_1 = ...
        // velocity_2 = ...
        // velocity_3 = ...
        // pressure = ...
            }
        }
    }
    // Send the dataBlock to device
}
```

Grid size is stored in the `DataBlockHost` object

- `d.np_tot` is a `std::vector<int>`
- `IDIR`, `JDIR` and `KDIR` are macros equal to 0,1,2

DataBlock & DataBlockHost class representation



Specifying the initial state

```
void Setup::InitFlow(DataBlock &data) {
    // Create a host copy of the dataBlock
    DataBlockHost d(data);
    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
             d.Vc(RHO,k,j,i) = 1.0;
             d.Vc(VX1,k,j,i) = 0.0;
             d.Vc(VX2,k,j,i) = 0.0;
             d.Vc(VX3,k,j,i) = example_function(i);
             d.Vc(PRS,k,j,i) = 1.0;
            }
        }
    }
    // Send the dataBlock to device
}
```

Grid size is stored in the DataBlockHost object

- d.Vc is a 4 dimension array where the **cell-centred fields** are stored
- Note the `()` to access data
- RHO, VX1, VX2, VX3, PRS, BX1, BX2, BX3, AX1, AX2, AX3 are macro for the id of the corresponding field in the array (1st dimension)
- Note the order `k,j,i`

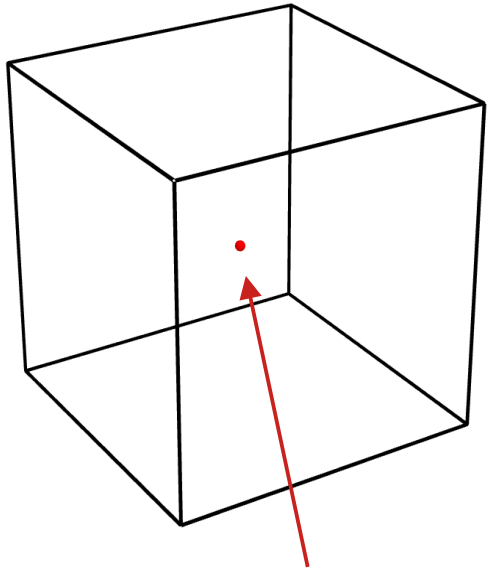
Specifying the initial state

```
void Setup::InitFlow(DataBlock &data) {
    // Create a host copy of the dataBlock
    DataBlockHost d(data);
    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
             d.Vc(RHO,k,j,i) = 1.0;
             d.Vc(VX1,k,j,i) = 0.0;
             d.Vc(VX2,k,j,i) = 0.0;
             d.Vc(VX3,k,j,i) = example_function(i);
             d.Vc(PRS,k,j,i) = 1.0;
            }
        }
    }
    // Send the dataBlock to device
    d.SyncToDevice();
}
```

Fields are evolved on the device (GPU) not the host (CPU)

Positions of the fields

Positions of the fields

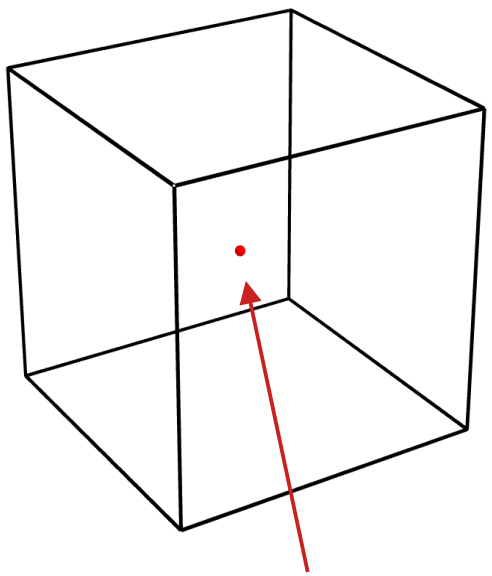


Cell-centred quantities

`RHO, VX*, PRS` ...

Are stored in `dataBlock.Vc`

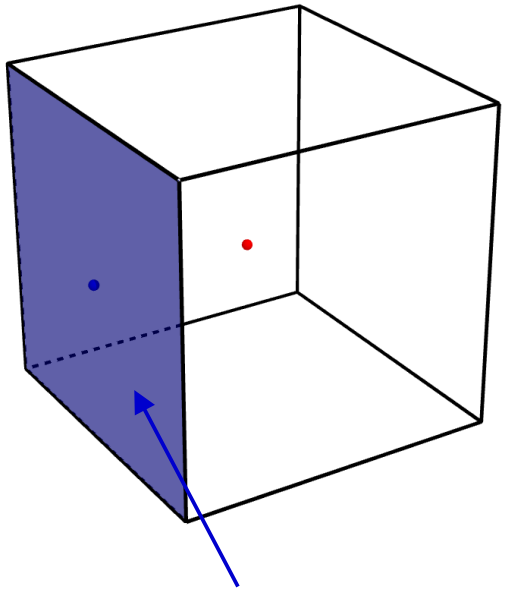
Positions of the fields



Cell-centred quantities

`RHO, VX*, PRS` ...

Are stored in `dataBlock.Vc`

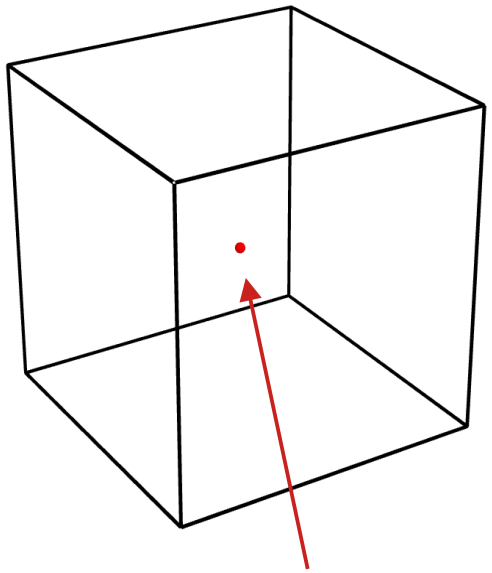


Surfaces quantities

`BX1s, BX2s, BX3s` ...

Are stored in `dataBlock.Vs`

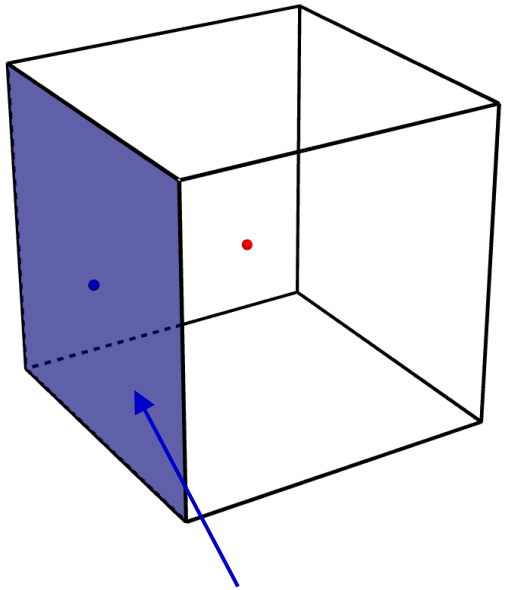
Positions of the fields



Cell-centred quantities

`RHO, VX*, PRS` ...

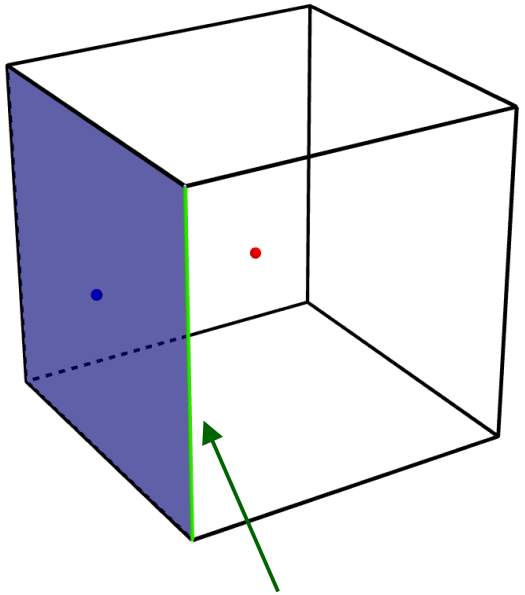
Are stored in `dataBlock.Vc`



Surfaces quantities

`BX1s, BX2s, BX3s` ...

Are stored in `dataBlock.Vs`



Edge quantities

`AX1e, AX2e, AX3e` ...

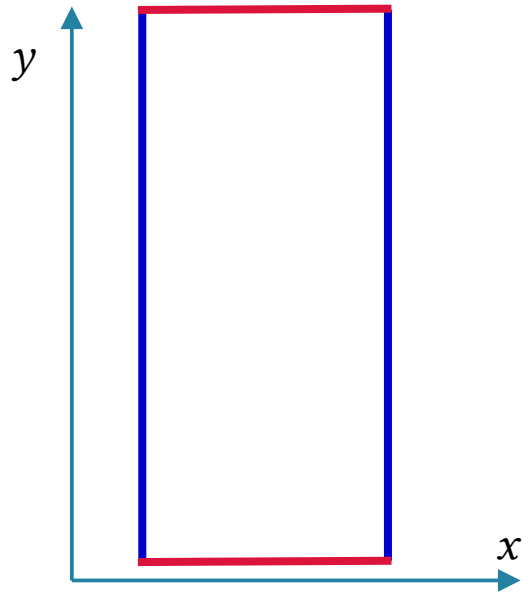
Are stored in `dataBlock.Ve`

Initial Condition: Exercise

Use the `idefix.ini` and `definitions.hpp` in `tutorials/usage/JetBox/` to create a setup of the following system:

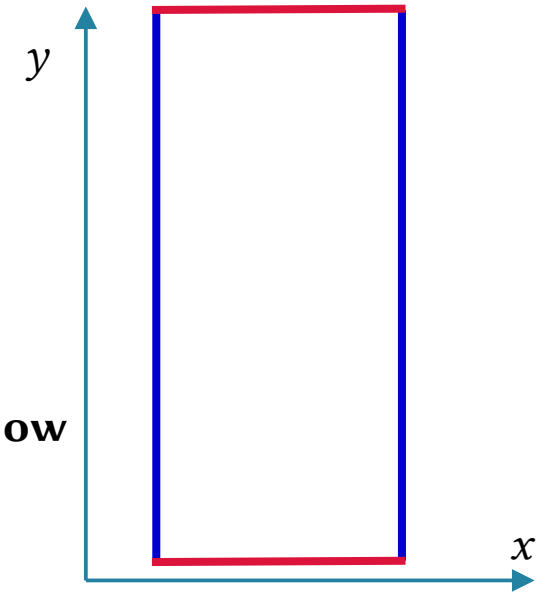
First look at the idefix.ini and definitions.hpp:

- 2D problem
- Isothermal
- Periodic in x, outflow in y



Initial Condition: Exercise

Use the `idefix.ini` and `definitions.hpp` in `tutorials/usage/JetBox/` to create a setup of the following system:



First look at the idefix.ini and definitions.hpp:

- 2D problem
- Isothermal
- Periodic in x, outflow in y

Then modify the `setup.cpp` to create the following initial flow

- Uniform density = 1
- Random fluid velocity ±10% of the sound speed

Initial Condition: Exercise

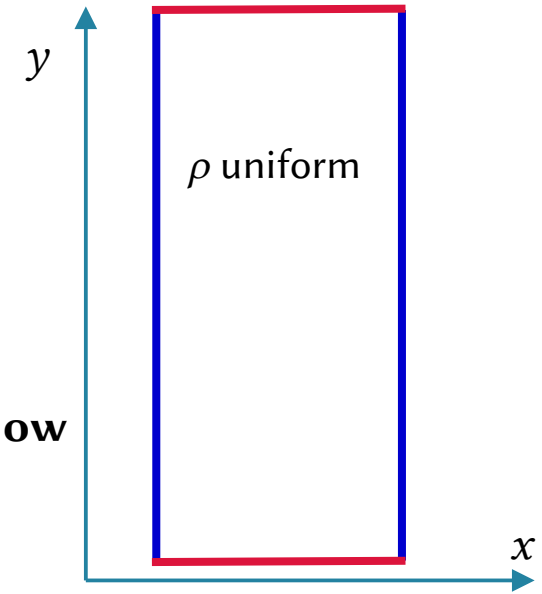
Use the `idefix.ini` and `definitions.hpp` in `tutorials/usage/JetBox/` to create a setup of the following system:

First look at the idefix.ini and definitions.hpp:

- 2D problem
- Isothermal
- Periodic in x, outflow in y

Then modify the `setup.cpp` to create the following initial flow

- Uniform density = 1
- Random fluid velocity ±10% of the sound speed



Initial Condition: Exercise

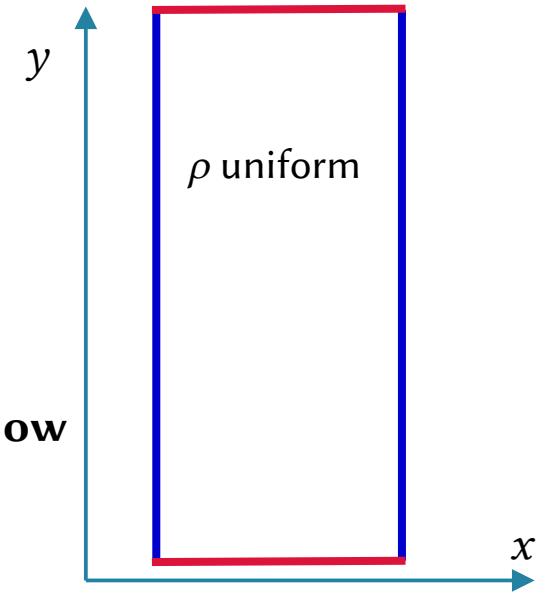
Use the `idefix.ini` and `definitions.hpp` in `tutorials/usage/JetBox/` to create a setup of the following system:

First look at the idefix.ini and definitions.hpp:

- 2D problem
- Isothermal
- Periodic in x, outflow in y

Then modify the `setup.cpp` to create the following initial flow

- Uniform density = 1
- Random fluid velocity ±10% of the sound speed



Pseudo-random in idefix with `idfx::randm()`

Initial Condition: Exercise

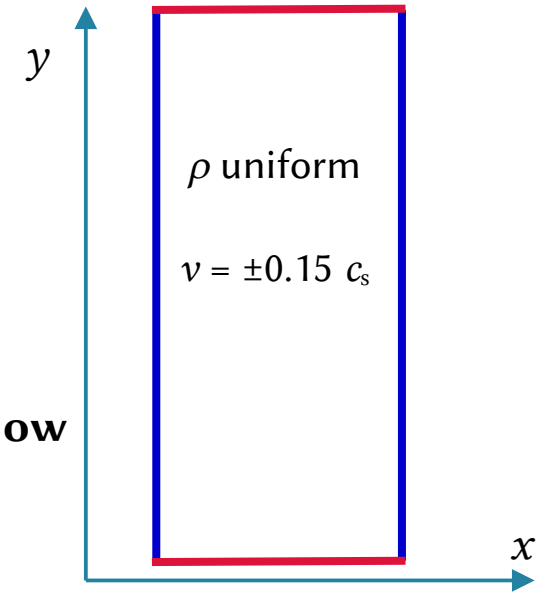
Use the `idefix.ini` and `definitions.hpp` in `tutorials/usage/JetBox/` to create a setup of the following system:

First look at the idefix.ini and definitions.hpp:

- 2D problem
- Isothermal
- Periodic in x, outflow in y

Then modify the `setup.cpp` to create the following initial flow

- Uniform density = 1
- Random fluid velocity ±10% of the sound speed



Pseudo-random in idefix with `idfx::randm()`

Boundary conditions

Boundary conditions

```
void MyBoundary(Hydro *hydro,int dir, BoundarySide side, real t) {
    if( /* wanted direction and wanted side */) {
        // declaring needed variable for kokkos
        loop_funtion(/* all ghost cells */) {
                        // density = ...
                        // pressure = ...
     }
    if( /* other wanted direction and wanted side */) {
        // declaring needed variable for kokkos
        loop_funtion(/* all ghost cells */) {
                        // density = ...
                        // pressure = ...
     }
}
```

Boundary conditions

Current direction: `IDIR`, `JDIR` or `KDIR`

```
void MyBoundary(Hydro *hydro,int dir, BoundarySide side, real t) {
    if( /* wanted direction and wanted side */) {
        // declaring needed variable for kokkos
        loop_funtion(/* all ghost cells */) {
                        // density = ...
                        // pressure = ...
     }
    if( /* other wanted direction and wanted side */) {
        // declaring needed variable for kokkos
        loop_funtion(/* all ghost cells */) {
                        // density = ...
                        // pressure = ...
     }
}
```

Boundary conditions

Current direction: `IDIR`, `JDIR` or `KDIR`

Current side: `left` or `right`

```
void MyBoundary(Hydro *hydro,int dir, BoundarySide side, real t) {
    if( /* wanted direction and wanted side */) {
        // declaring needed variable for kokkos
        loop_funtion(/* all ghost cells */) {
                        // density = ...
                        // pressure = ...
     }
    if( /* other wanted direction and wanted side */) {
        // declaring needed variable for kokkos
        loop_funtion(/* all ghost cells */) {
                        // density = ...
                        // pressure = ...
     }
}
```

Boundary conditions

Current direction: `IDIR`, `JDIR` or `KDIR`

Current side: `left` or `right`

Current time

```
void MyBoundary(Hydro *hydro,int dir, BoundarySide side, real t) {
    if( /* wanted direction and wanted side */) {
        // declaring needed variable for kokkos
        loop_funtion(/* all ghost cells */) {
                        // density = ...
                        // pressure = ...
     }
    if( /* other wanted direction and wanted side */) {
        // declaring needed variable for kokkos
        loop_funtion(/* all ghost cells */) {
                        // density = ...
                        // pressure = ...
     }
}
```

Boundary conditions

Current direction: `IDIR`, `JDIR` or `KDIR`

Current side: `left` or `right`

Current time

Returns nothing: Modifies the values "in-place"

```
void MyBoundary(Hydro *hydro,int dir, BoundarySide side, real t) {
    if( /* wanted direction and wanted side */) {
        // declaring needed variable for kokkos
        loop_funtion(/* all ghost cells */) {
                        // density = ...
                        // pressure = ...
        }
    if( /* other wanted direction and wanted side */) {
        // declaring needed variable for kokkos
        loop_funtion(/* all ghost cells */) {
                        // density = ...
                        // pressure = ...
        }
}
```

Boundary conditions: Example

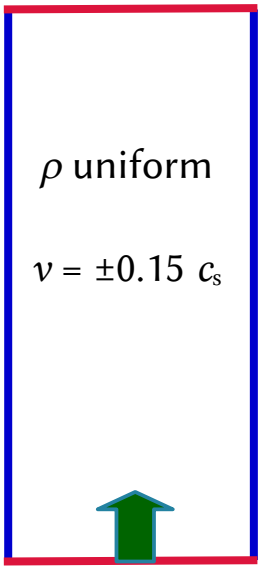
```
void MyBoundary(Hydro *hydro,int dir, BoundarySide side, real t) {
    DataBlock *data = hydro->data;
    if((dir==IDIR) && (side == left)) {

        // declaring needed variable for kokkos
        IdefixArray4D<real> Vc = hydro->Vc;
        int ighost = data->nghost[IDIR];

        hydro->boundary->BoundaryFor("UserDefBoundaryBeg", dir, side,
                        KOKKOS_LAMBDA (int k, int j, int i) {



                    Vc(RHO,k,j,i) = Vc(RHO,k,j,ighost);
                    // other fields ...
        });
     }
    // other sides ...
}
```

Variables used in the **BoundaryFor**

Boundary Condition Exercise: Jet in a Box

Modify the previous setup to include a custom boundary condition at $y = 0$

- For $0.45 \leq x \leq 0.55$, a jet with :
  - $\rho_{jet} = 10\ \rho_{ini}$
  - $\partial v_x/\partial y = 0$
  - $v_y = 10\ c_s$
- Outside the jet
  - $\partial \rho/\partial y = 0$
  - $\partial v_x/\partial y = 0$
  - $\partial v_y/\partial y = 0$



Internal boundary condition

```
void InternalBoundary(Hydro *hydro, const real t) {
  //declare needed variables
  loop_function(/* ... */
                if(/* density < floor*/) {
                  // density = floor
                }
               );
}
```

Internal boundary condition

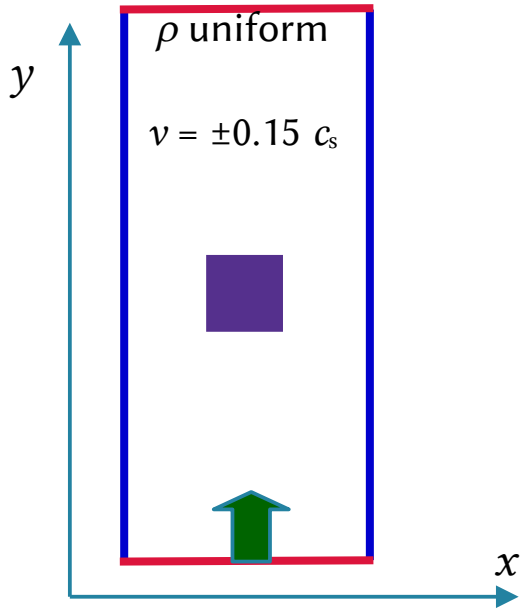
```
void InternalBoundary(Hydro *hydro, const real t) {
  DataBlock *data = hydro->data;
  IdefixArray4D<real> Vc = hydro->Vc;
  real densityFloor = densityFloorGlob;
  idefix_for("InternalBoundary", 0, data->np_tot[KDIR], 0, data->np_tot[JDIR],
                                 0, data->np_tot[IDIR],
             KOKKOS_LAMBDA (int k, int j, int i) {
               if(Vc(RHO,k,j,i) < densityFloor) {
                 Vc(RHO,k,j,i)=densityFloor;
               }
             });

}
```

Internal Condition Exercise: Block in the jet

Modify the previous setup to include a custom internal condition

In the region $(x,y) \in [0.4,0.6] \times [1.9, 2.1]$, set

- $\rho = 1$
- $v_x = v_y = 0$

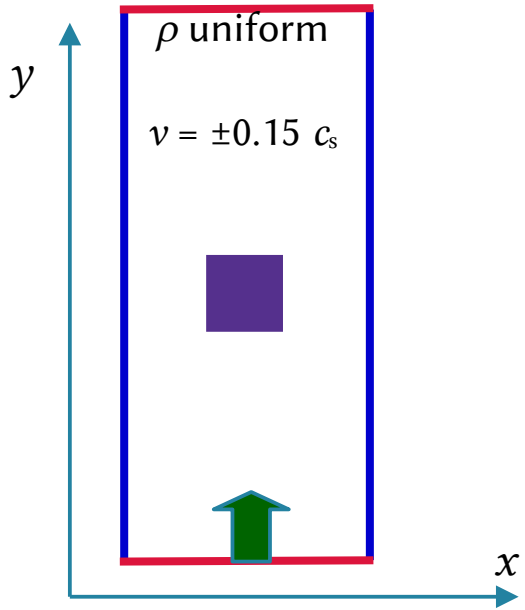


Internal Condition Exercise: Block in the jet

Modify the previous setup to include a custom internal condition

In the region $(x,y) \in [0.4,0.6] \times [1.9, 2.1]$, set

- $\rho = 1$
- $v_x = v_y = 0$



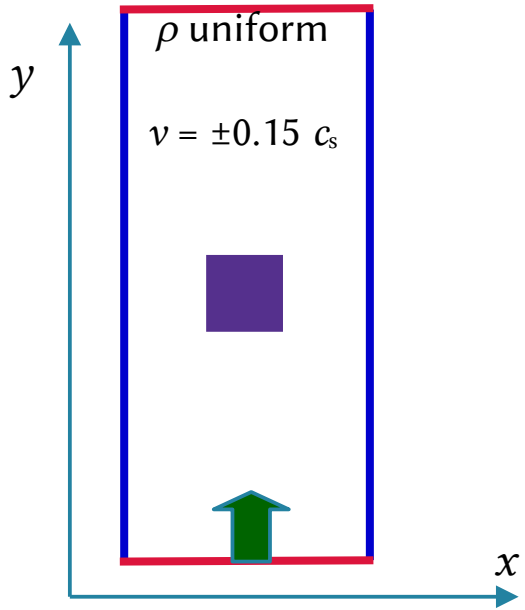
Coordinates of cells are stored in
`hydro→data→x[IDIR]`

Internal Condition Exercise: Block in the jet

Modify the previous setup to include a custom internal condition

In the region $(x,y) \in [0.4,0.6] \times [1.9, 2.1]$, set

- $\rho = 1$
- $v_x = v_y = 0$



Coordinates of cells are stored in

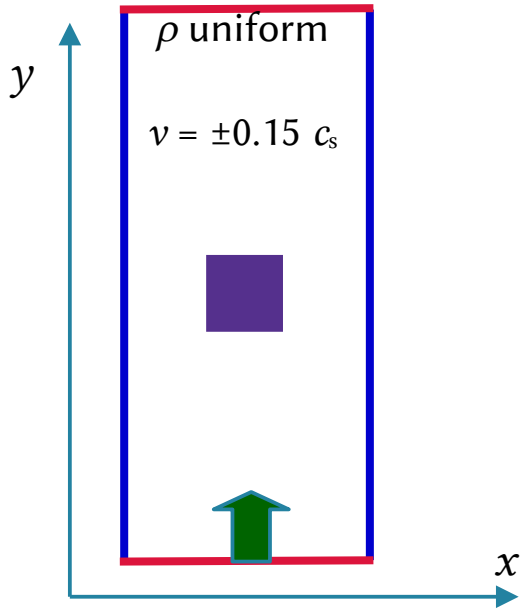
```
hydro→data→x[IDIR]
hydro→data→x[JDIR]
```

Internal Condition Exercise: Block in the jet

Modify the previous setup to include a custom internal condition

In the region $(x,y) \in [0.4,0.6] \times [1.9, 2.1]$, set

- $\rho = 1$
- $v_x = v_y = 0$



Coordinates of cells are stored in

```
hydro→data→x[IDIR]
hydro→data→x[JDIR]
hydro→data→x[KDIR]
```

Other custom functions

Always the same principle, you fill an array with modified values:

- Gravitational potential

Other custom functions

Always the same principle, you fill an array with modified values:

- Gravitational potential

```
void Potential(DataBlock &data, const real t, IdefixArray1D<real> &x1,
IdefixArray1D<real> &x2, IdefixArray1D<real> &x3, IdefixArray3D<real> &phi){
// ...
}


//enroll it in Setup::Setup
data.gravity->EnrollPotential(&Potential);
```

Other custom functions

Always the same principle, you fill an array with modified values:

- Gravitational potential

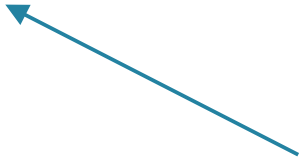
```
void Potential(DataBlock &data, const real t, IdefixArray1D<real> &x1,
IdefixArray1D<real> &x2, IdefixArray1D<real> &x3, IdefixArray3D<real> &phi){
// ...
}


//enroll it in Setup::Setup
data.gravity->EnrollPotential(&Potential);
```

You fill `phi`

Other custom functions

Always the same principle, you fill an array with modified values:

- Gravitational potential
- Viscosity



Other custom functions

Always the same principle, you fill an array with modified values:

- Gravitational potential
- Viscosity

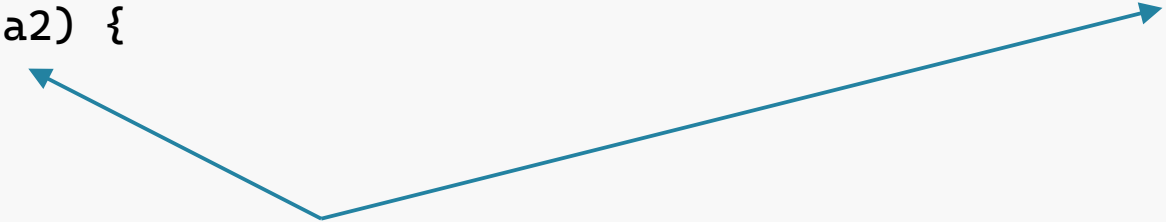
```
void MyViscosity(DataBlock &data, const real t, IdefixArray3D<real> &eta1,
IdefixArray3D<real> &eta2) {
// ...
}


//enroll it in Setup::Setup
data.hydro->viscosity->EnrollViscousDiffusivity(&MyViscosity);
```

Other custom functions

Always the same principle, you fill an array with modified values:

- Gravitational potential
- Viscosity

```
void MyViscosity(DataBlock &data, const real t, IdefixArray3D<real> &eta1,
IdefixArray3D<real> &eta2) {
// ...
}


//enroll it in Setup::Setup
data.hydro->viscosity->EnrollViscousDiffusivity(&MyViscosity);
```



Other custom functions

Always the same principle, you fill an array with modified values:

- Gravitational potential
- Viscosity
- It’s the same for all the other cases…

Custom outputs

```
void ComputeUserVars(DataBlock & data, UserDefVariablesContainer &variables) {


  // Make references to the user-defined arrays (variables is a container of
IdefixHostArray3D)
  // Note that the labels should match the variable names in the input file
  IdefixHostArray3D<real> Er  = variables["Er"];
  IdefixHostArray3D<real> Eth = variables["Eth"];


    Kokkos::deep_copy(Er,data.hydro.emf.Ex1);
    Kokkos::deep_copy(Eth,data.hydro.emf.Ex2);



}
```

Outputs are written by the host

Custom outputs

```
void ComputeUserVars(DataBlock & data, UserDefVariablesContainer &variables) {

  // Make references to the user-defined arrays (variables is a container of
IdefixHostArray3D)
  // Note that the labels should match the variable names in the input file
  IdefixHostArray3D<real> Er  = variables["Er"];
  IdefixHostArray3D<real> Eth = variables["Eth"];

    Kokkos::deep_copy(Er,data.hydro.emf.Ex1);
    Kokkos::deep_copy(Eth,data.hydro.emf.Ex2);


}
```

Outputs are written by the host

```
[Output]
uservar     Er      Eth
vtk         0.1
dmp         10.0
log         10
```

Names must match in the `idefix.ini`

Command line options

```
$ ./idefix [options]
```

Command line options

```
$ ./idefix [options]
```

- For MPI decomposition : `-dec n1 n2 n3`
  `n1*n2*n3` must be equal to #MPI processes, dimension-dependant

Command line options

```
$ ./idefix [options]
```

- For MPI decomposition : `-dec n1 n2 n3`
  `n1*n2*n3` must be equal to #MPI processes, dimension-dependant
- `-restart [n]` Restarts from dump file number n

Command line options

```
$ ./idefix [options]
```

- For MPI decomposition : `-dec n1 n2 n3`
  `n1*n2*n3` must be equal to #MPI processes, dimension-dependant
- `-restart [n]` Restarts from dump file number n
- `-i <inifile>` Uses a specific inifile

Command line options

```
$ ./idefix [options]
```

- For MPI decomposition : `-dec n1 n2 n3`
  `n1*n2*n3` must be equal to #MPI processes, dimension-dependant
- `-restart [n]` Restarts from dump file number n
- `-i <inifile>` Uses a specific inifile
- `-maxcycles <n>` stops integration after n cycles

Command line options

```
$ ./idefix [options]
```

- For MPI decomposition : `-dec n1 n2 n3`
  `n1*n2*n3` must be equal to #MPI processes, dimension-dependant
- `-restart [n]` Restarts from dump file number n
- `-i <inifile>` Uses a specific inifile
- `-maxcycles <n>` stops integration after n cycles
- `-nolog`

Command line options

```
$ ./idefix [options]
```

- For MPI decomposition : `-dec n1 n2 n3`
  `n1*n2*n3` must be equal to #MPI processes, dimension-dependant
- `-restart [n]` Restarts from dump file number n
- `-i <inifile>` Uses a specific inifile
- `-maxcycles <n>` stops integration after n cycles
- `-nolog`
- `-nowrite`

Command line options

```
$ ./idefix [options]
```

- For MPI decomposition : `-dec n1 n2 n3`
  `n1*n2*n3` must be equal to #MPI processes, dimension-dependant
- `-restart [n]` Restarts from dump file number n
- `-i <inifile>` Uses a specific inifile
- `-maxcycles <n>` stops integration after n cycles
- `-nolog`
- `-nowrite`
- `-Werror` warnings are treated as errors

Command line options

```
$ ./idefix [options]
```

- For MPI decomposition : `-dec n1 n2 n3`
  `n1*n2*n3` must be equal to #MPI processes, dimension-dependant
- `-restart [n]` Restarts from dump file number n
- `-i <inifile>` Uses a specific inifile
- `-maxcycles <n>` stops integration after n cycles
- `-nolog`
- `-nowrite`
- `-Werror` warnings are treated as errors
- `-kokkos-num-devices=x` tells kokkos the number of device on each node

Stopping the code

- Outside a job scheduler:

```
$ kill -s SIGUSR2  <idefix pid>
```

Idefix will send an abort signal to all its processes

- If this is not possible

```
$ touch stop
```

Has the same effect

⟹ When receiving the abort signal, Idefix will write a restart (dump) file and stop

Template credits: