CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

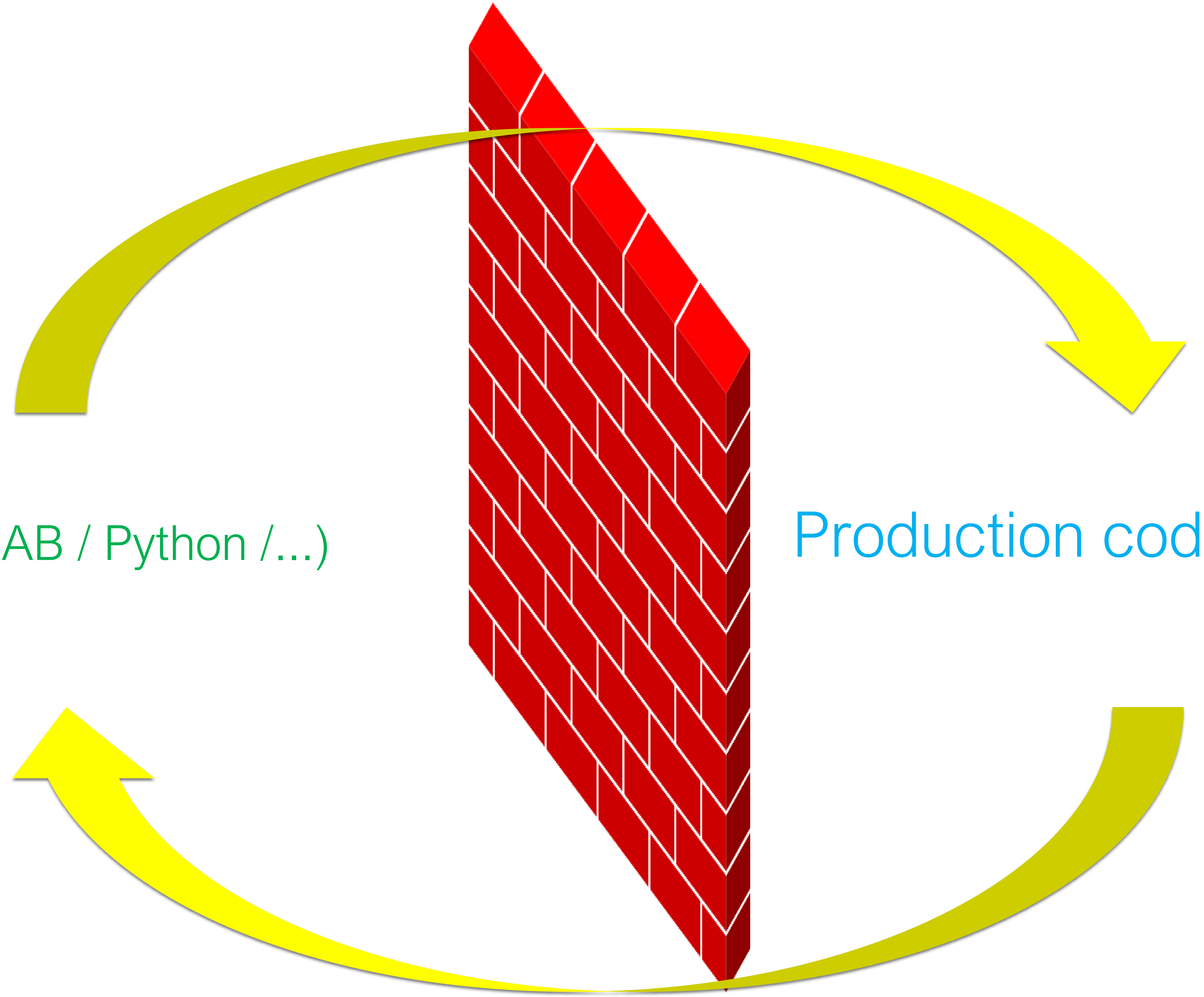
Interactive Supercomputing on Piz Daint: Using Julia with Jupyter Notebooks

CSCS User Lab Day 2022 – Meet the Swiss National Supercomputing Centre

Dr. Samuel Omlin

September 2nd 2022

The two language problem



Protoype (MATLAB / Python /...)

Production code (C / C++ / Fortran / ...)

Solution

A language that can be used for both

Protoype & Production code

julia

Solution

julia

simple & high-level

interactive

low development cost

fast

Solution

simple & high-level

interactive

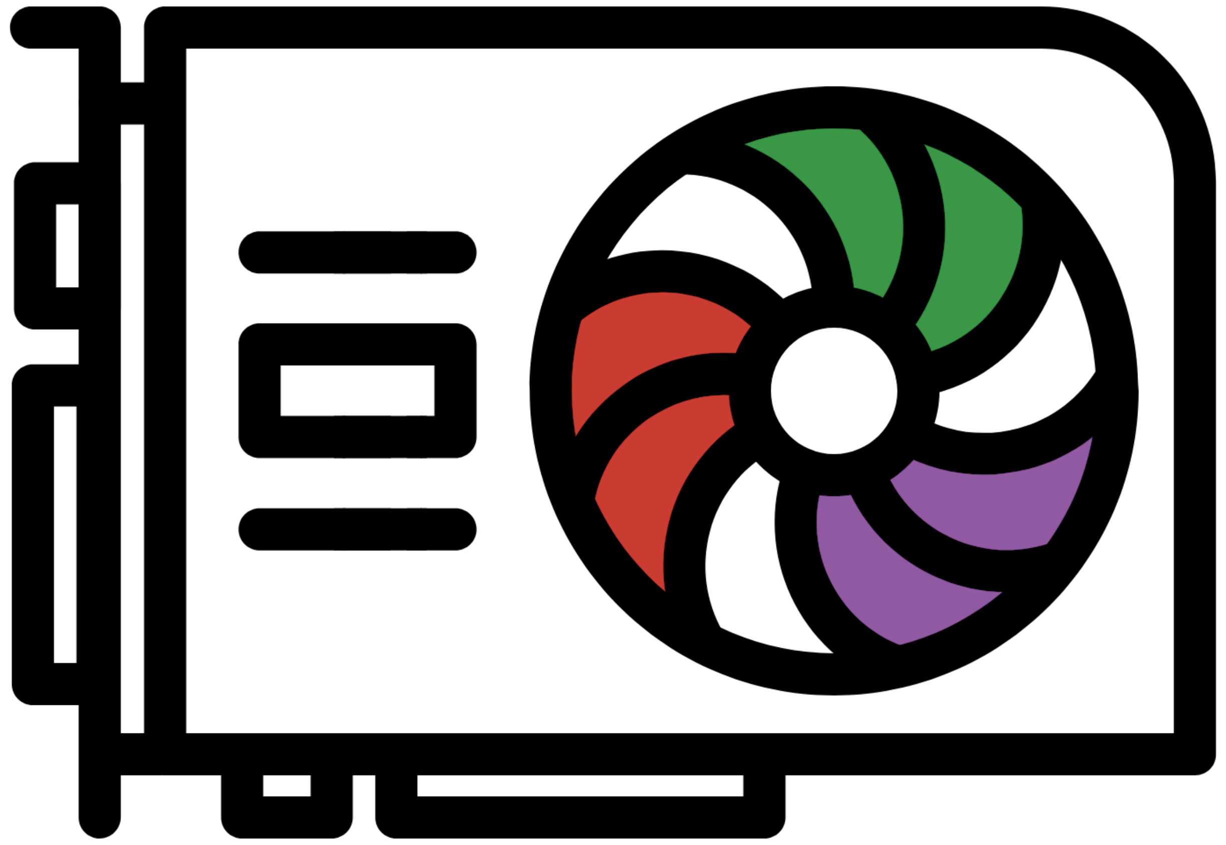
low development cost

fast

Fast and interactive???

Julia code is compiled, yet only shortly before you use it the first time.

Solution



CUDA.jl

Native Julia Code for GPUs!

simple & high-level

interactive

low development cost

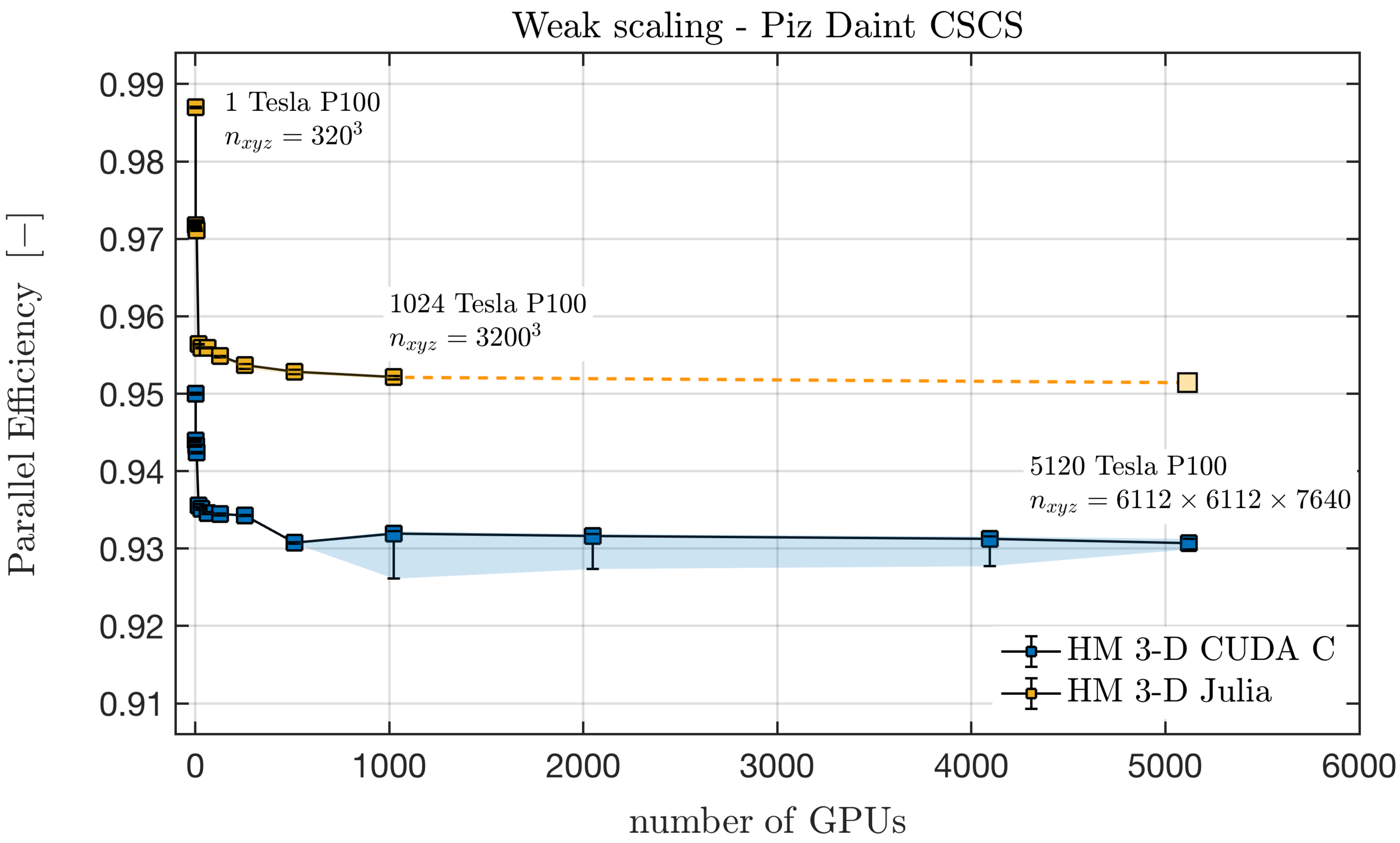
fast

MPI.jl

Julia suitable for GPU supercomputing

Single GPU performance:

93% of the the CUDA C code



Agenda

- Introduction ✓
- Julia on Piz Daint
- Julia in JupyterLab at CSCS
- Julia Notebook examples
- Conclusions & Outlook

Julia on Piz Daint

Default Julia modules (long-term support version):

```
$> module load daint-gpu  # or daint-mc
$> module load Julia                      <- includes MPI + CUDA packages
$> module load JuliaExtensions            <- Plots, PyCall & HDF5 packages…
```

Available packages:

```
julia> versioninfo()
```

Note on the Julia package manager:

`julia> Pkg.status()` shows only the packages installed by the user by default, but you can load the above packages normally, e.g.:

```
julia> using MPI
```

Start an interactive Julia session with GPU:

```
$> srun -C gpu --time=04:00:00 --pty bash
$> julia
```

Julia on Piz Daint

Default Julia modules (long-term support version):

```
$> module load daint-gpu  # or daint-mc
$> module load Julia
$> module load JuliaExtensions
```

stacked environment:

- user installed packages have precedence!
- user package installation on home

Available packages:

```
julia> versioninfo()
```

Note on the Julia package manager:

`julia> Pkg.status()` shows only the packages installed by the user by default, but you can load the above packages normally, e.g.:

```
julia> using MPI
```

Start an interactive Julia session with GPU:

```
$> srun -C gpu --time=04:00:00 --pty bash
$> julia
```

Julia on Piz Daint

Default Julia modules (long-term support version):

```
$> module load daint-gpu  # or daint-mc
$> module load Julia
$> module load JuliaExtensions
```

≠ latest Julia modules (non-default):
- no more stacked environment
- package installation on scratch

Available packages:

```
julia> versioninfo()
```

Note on the Julia package manager:

```
julia> Pkg.status()
```

shows only the packages installed by the user by default, but you can load the above packages normally, e.g.:

```
julia> using MPI
```

Start an interactive Julia session with GPU:

```
$> srun -C gpu --time=04:00:00 --pty bash
$> julia
```

Julia on Piz Daint

Default Julia modules (long-term support version):

```
$> module load daint-gpu  # or daint-mc
$> module load Julia                           <- includes MPI + CUDA packages
$> module load JuliaExtensions                 <- Plots, PyCall & HDF5 packages…
```

Available packages:

```
julia> versioninfo()
```

Note on the Julia package manager:

`julia> Pkg.status()` shows only the packages installed by the user by default, but you can load the above packages normally, e.g.:

```
julia> using MPI
```

More information: https://user.cscs.ch/tools/interactive/julia/

Agenda

- Introduction ✓
- Julia on Piz Daint ✓
- Julia in JupyterLab at CSCS
- Julia Notebook examples
- Conclusions & Outlook

Julia in JuperLab at CSCS

- Uses Julia default modules: `Julia` and `JuliaExtensions` are **automatically loaded**.
- Accesses the **same stacked environment**
- Currently not set up for usage with MPI (not yet straigtforward and well supported): **use a single node**.

Installing a package from the command line or from JupyterLab gives the exact same result!

Agenda

- Introduction ✓
- Julia on Piz Daint ✓
- Julia in JupyterLab at CSCS ✓
- Julia Notebook examples
- Conclusions & Outlook

Notebook 1: using the stacked environment

https://user.cscs.ch/tools/interactive/jupyterlab/#ijulia

Side note:
run the notebook to see
what versions are now
available etc!

Notebook 2: glacier flow using GPU

2-D Shallow ice equations

$$\frac{\partial H}{\partial t} = -\nabla_i (qH_i)$$

$$qH_i = -\frac{H^3 g}{3\mu} \nabla_i (H + B)$$

Notebook 2: glacier flow using GPU

2-D Shallow ice equations

$$\frac{\partial H}{\partial t} = -\nabla_i (qH_i)$$

$$qH_i = -\frac{\boldsymbol{H^3} g}{3\mu} \nabla_i (H + B)$$

Nonlinear diffusion!

Notebook 2: glacier flow using GPU

Numerics

- Iterative algorithm with implicit time stepping
- Pseudo-transient method
- Numerical damping for convergence acceleration

Notebook 2: glacier flow using CPU

Demo…

JupyterHub
https://omlins.jupyter.cscs.ch/hub/spawn/omlins
160%
CSCS
ETH zürich
jupyterhub
Home
Token
Services
User: omlins

Node Type: GPU

Nodes: 1

Duration (hr): 1

-
+
-
+
Advanced options
Launch JupyterLab
Help | Privacy | Terms
2019 © CSCS | www.cscs.ch

Notebook 2: glacier flow using GPU

Demo…

```
┌ Info: Saved animation to
│   fn = /users/omlins/jupyterlab_test/glacier/tmp.gif
└ @ Plots /apps/daint/UES/jenkins/7.0.UP03/21.09/daint-gpu/software/JuliaExtensions/1.6.3-CrayGNU-21.09-cuda/extensions/packages/Plots/5kcBO/src/animation.jl:114
```

shallow_ice_... - JupyterL
https://omlins.jupyter.cscs.ch/user/omlins/lab?
160%
File
Edit
View
Run
Kernel
Tabs
Settings
Help
Mem:
1.03 / 59.31 GB
shallow_ice_CuArrays.ipynb
Code
Julia 1.6.3
IP
[7]:
H
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
0.6
0.5
0.4
0.3
0.2
Simple
0
1
Julia 1.6.3 | Idle
Mem: 1.03 / 59.31 GB
Mode: Command
Ln 1, Col 1
shallow_ice_CuArrays.ipynb

Notebook 2: glacier flow using GPU

Demo…

shallow_ice_... - JupyterL
https://omlins.jupyter.cscs.ch/user/omlins/lab?
110%
File
Edit
View
Run
Kernel
Tabs
Settings
Help
Mem:
1.15 / 59.31 GB
shallow_ice_CuArrays.ipynb
Code
Julia 1.6.3
IP

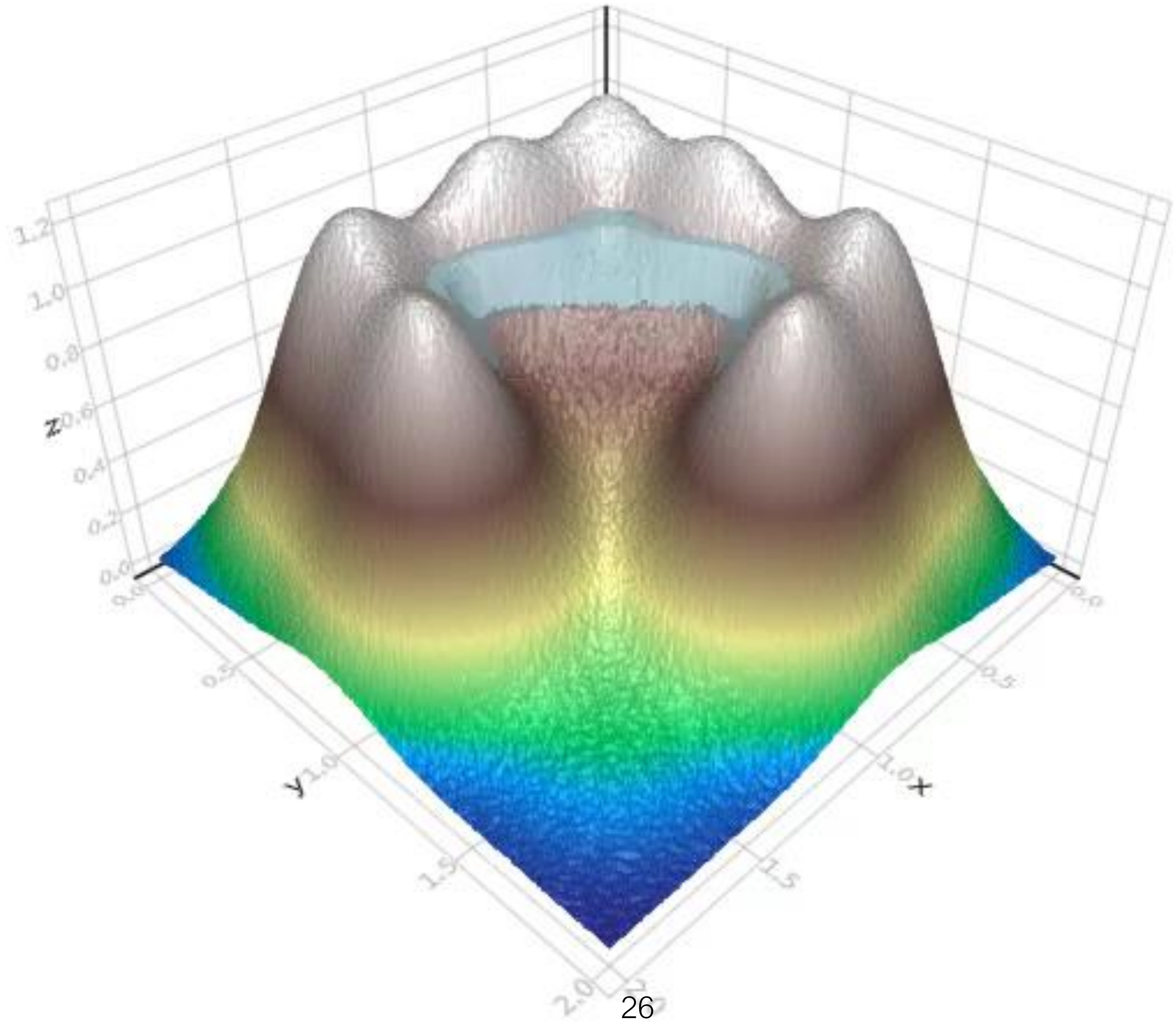
```
    end
end
```

```
  iter = 10, err = 3.116e-05
  iter = 20, err = 5.049e-06
  iter = 30, err = 2.096e-06
  iter = 40, err = 4.015e-07
  iter = 50, err = 1.437e-07
  iter = 60, err = 3.555e-08
  iter = 70, err = 1.362e-08
  iter = 80, err = 8.628e-09
┌ Info: Saved animation to
│   fn = /users/omlins/jupyterlab_test/glacier/tmp.gif
└ @ Plots /apps/daint/UES/jenkins/7.0.UP03/21.09/daint-gpu/software/JuliaExtensions/1.6.3-CrayGNU-21.09-cuda/extensions/packages/Plots/5kcBO/src/animation.jl:114
```

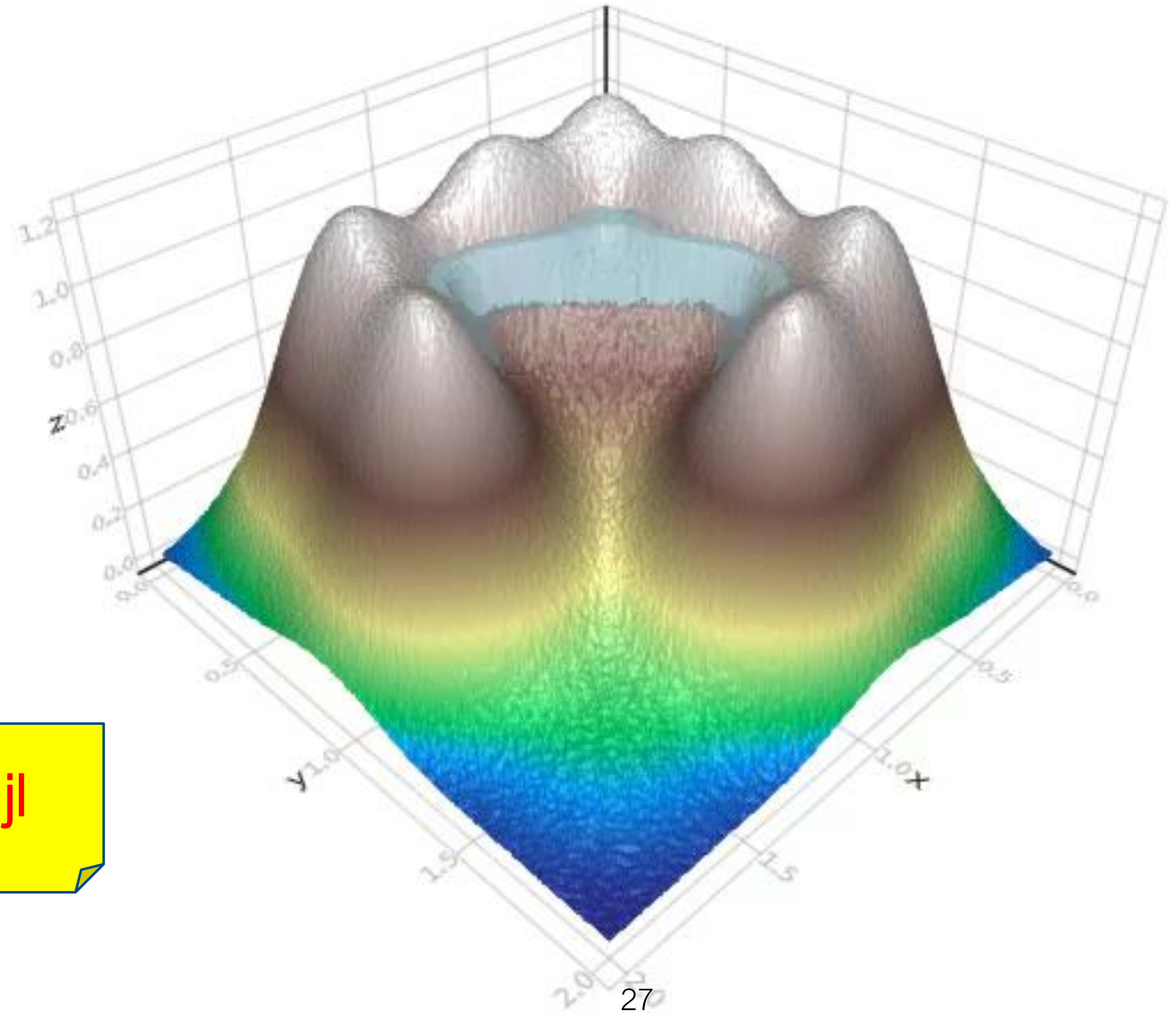
[7]:

[ ]:

3-D OpenGL visualization in Julia (different topography)



3-D OpenGL visualization in Julia (different topography)



Done with Makie.jl

Agenda

- Introduction ✓
- Julia on Piz Daint ✓
- Julia in JupyterLab at CSCS ✓
- Julia Notebook examples ✓
- Conclusions & Outlook

Conclusions & outlook

- same stacked environment in JupyterLab as when using Julia from command line
- `CUDA.jl` enables writing native Julia code for GPUs

Conclusions & outlook

- same stacked environment in JupyterLab as when using Julia from command line
- `CUDA.jl` enables writing native Julia code for GPUs

Questions / advice / feedback / …

I am the responsible for Julia computing – get in touch with me!

help@cscs.ch | Samuel.Omlin@cscs.ch

Thank you for your kind attention