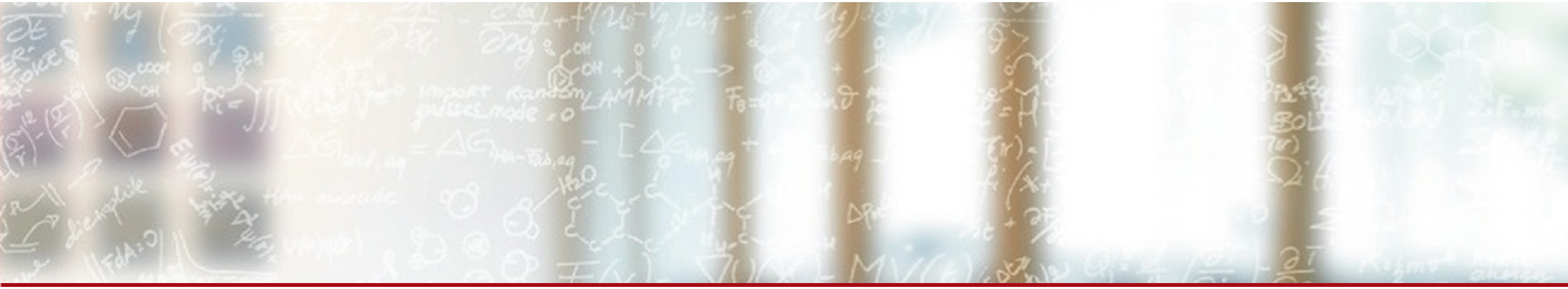
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

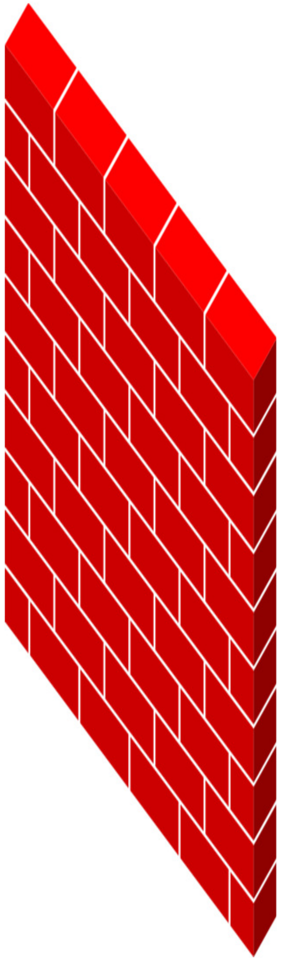


Interactive Computing with Julia in JupyterLab

CSCS User Lab Day – Meet the Swiss National Supercomputing Centre
Samuel Omlin
September 1st 2020

Protoype

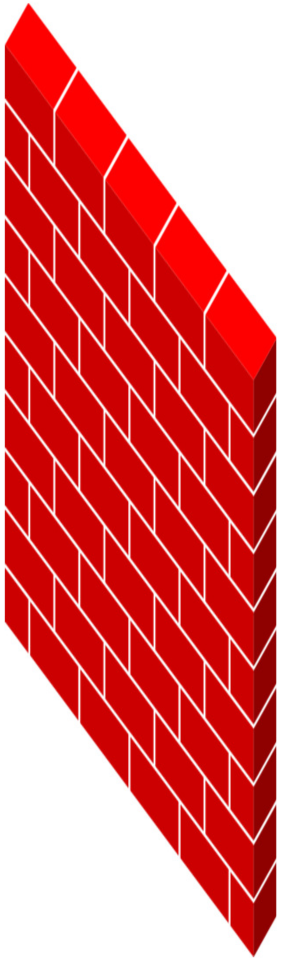
```
P = rand(4,3)
```



Production code

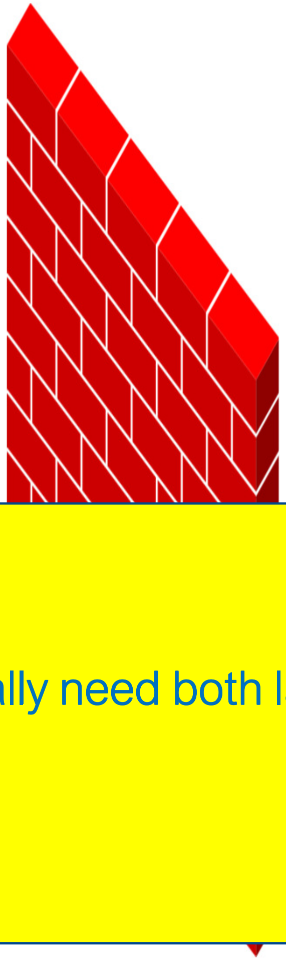
```
float* P;
P = malloc(…);
rand(P,…);
```

The two language problem



Protoype (MATLAB / Python /...)	Production code (C / C++ / Fortran / ...)
simple & high-level	complex & low-level
interactive	not interactive
low development cost	high development cost
slow	fast

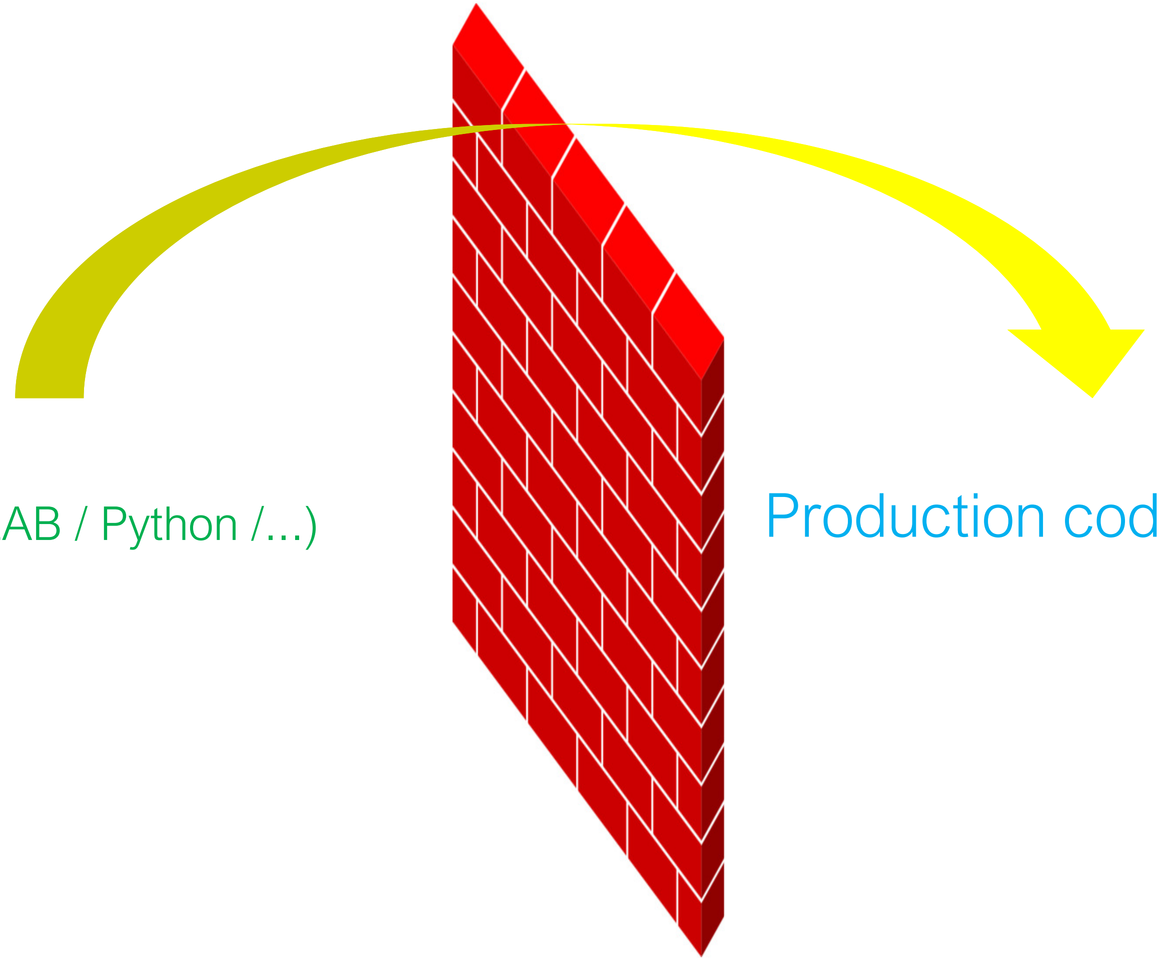
The two language problem



Protoype (MATLAB / Python /...)	Production code (C / C++ / Fortran / ...)
simple & high-level	complex & low-level
interactive	not interactive
low development cost	high development cost
slow	fast

We really need both languages!

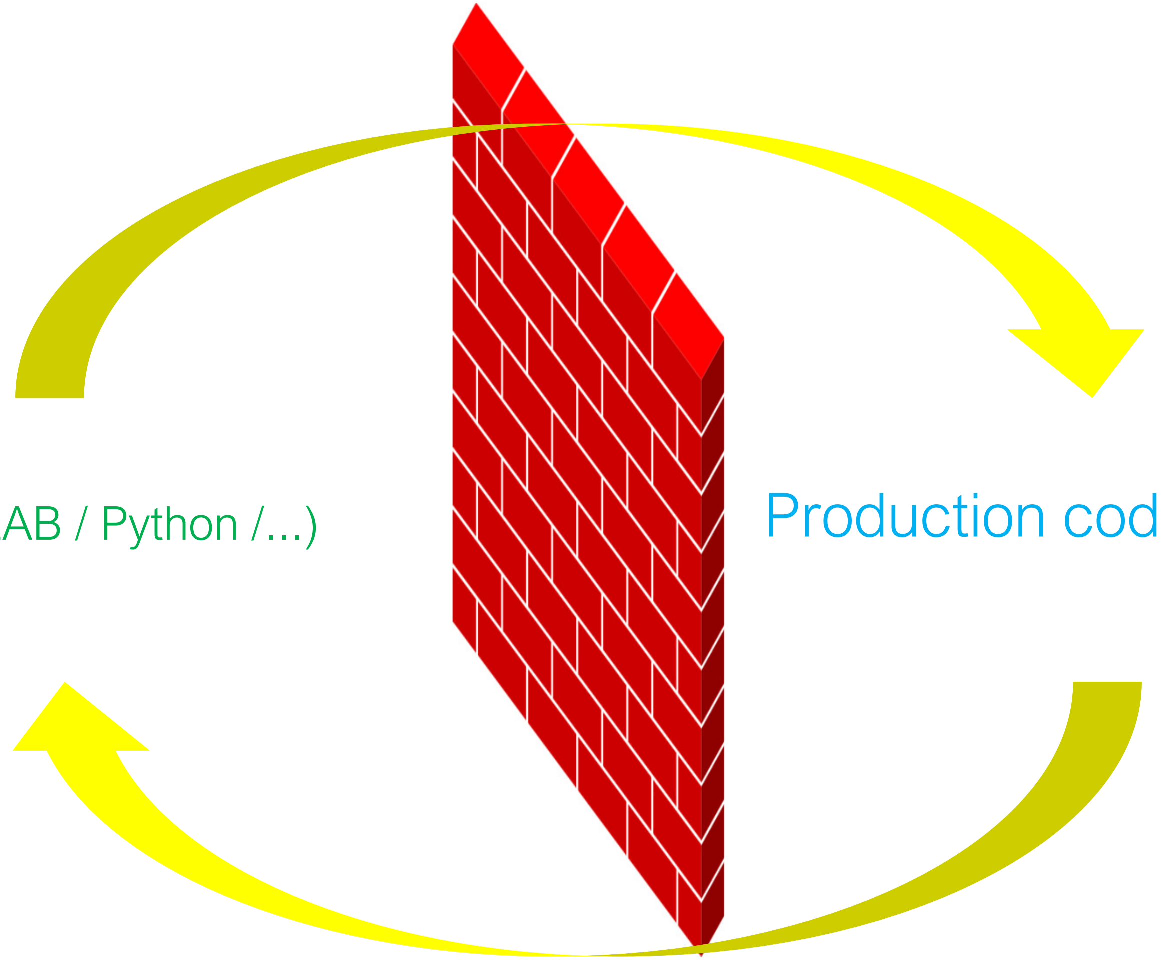
The two language problem



Protoype (MATLAB / Python /...)

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

The two language problem



Protoype (MATLAB / Python /...)

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

Solution

A language that can be used for both

Protoype & Production code

Solution

A language that can be used for both

Protoype & Production code

Solution

simple & high-level

interactive

low development cost

fast

Solution

julia

simple & high-level

interactive

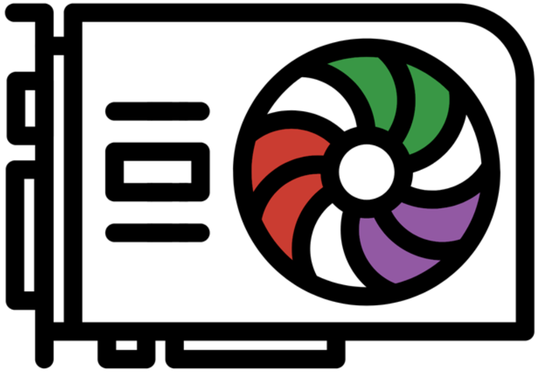
low development cost

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

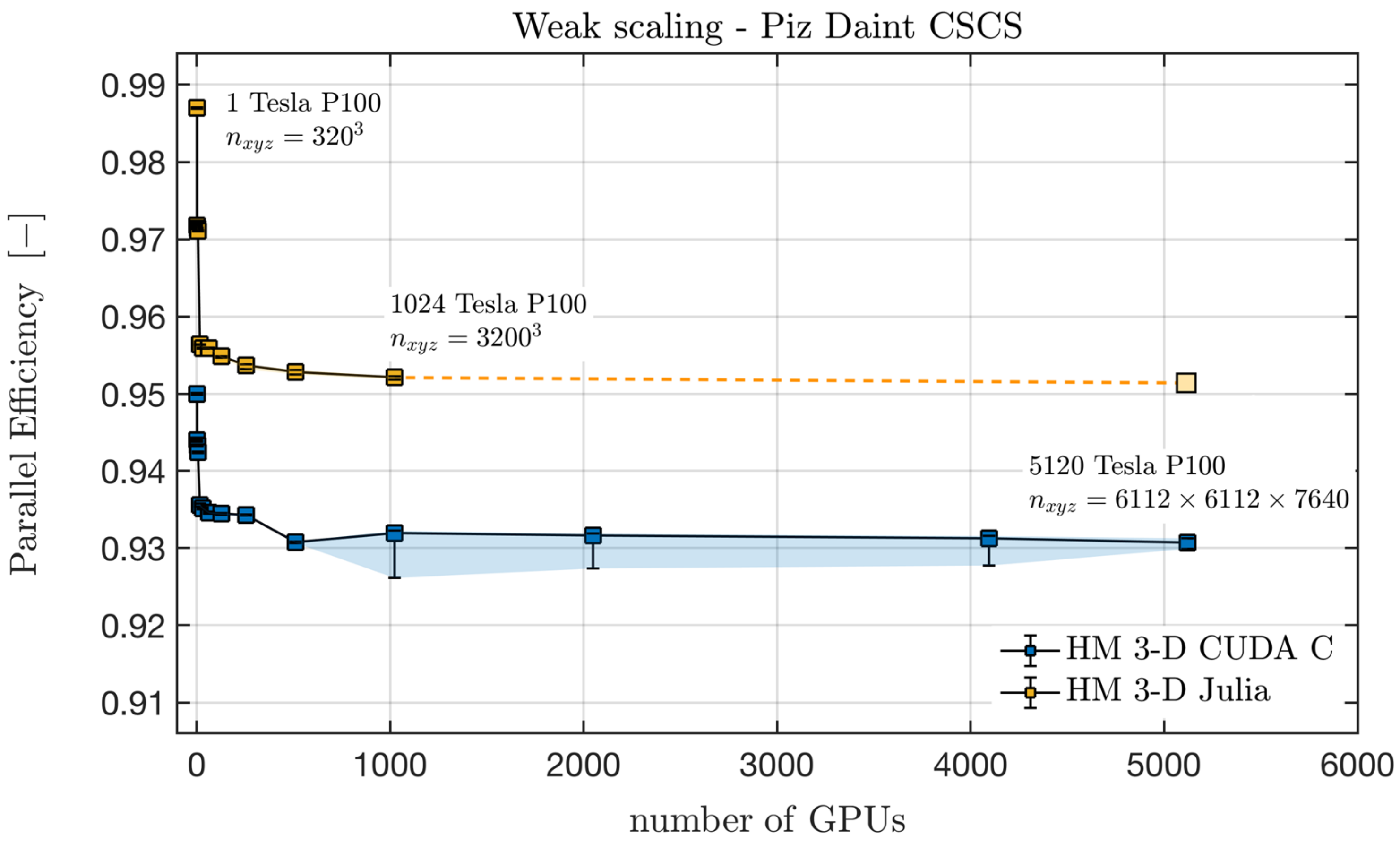
fast

MPI.jl

Julia suitable for GPU supercomputing

Single GPU performance:

93% of the the CUDA C code



Agenda

- Introduction: the two language problem ✓
- Julia on Piz Daint
- Julia in JupyterLab at CSCS
- Julia Notebook examples
- Conclusions & Outlook

Julia on Piz Daint

Julia modules:

```
$> 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 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

stacked environment:
user installed packages have precedence!

Julia modules:

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

Available packages:

```
julia> versioninfo()
```

Note on the Julia package manager 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

Julia modules:

```
$> 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 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: the two language problem ✓
- Julia on Piz Daint ✓
- Julia in JupyterLab at CSCS
- Julia Notebook examples
- Conclusions & Outlook

Julia in JuperLab at CSCS

- Accesses the same stacked environment
- The modules `Julia` and `JuliaExtensions` are automatically loaded.
- 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: the two language problem ✓
- 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

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!

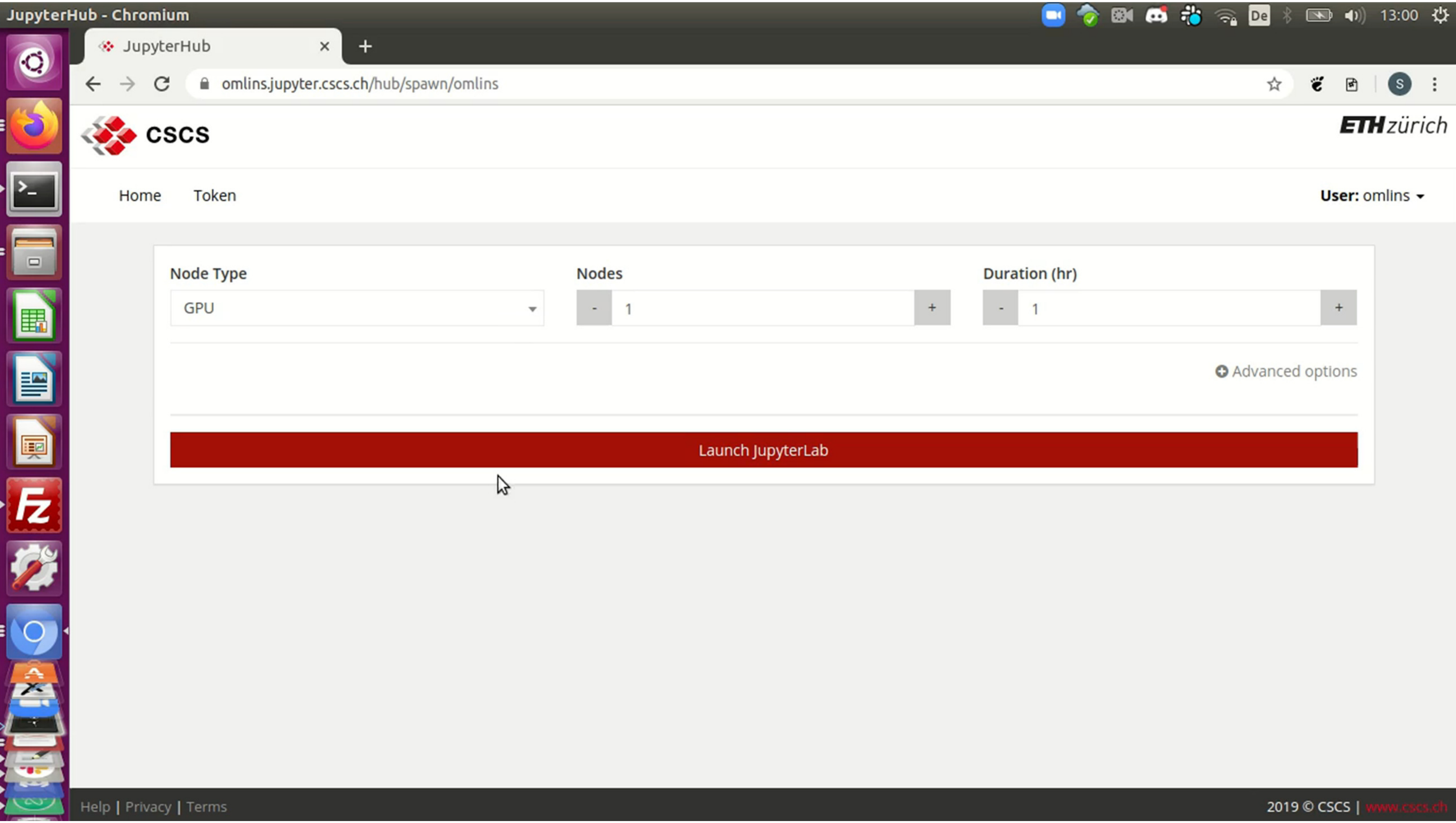
Notebook 2: glacier flow using GPU

Numerics

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

Notebook 2: glacier flow using GPU

Demo…

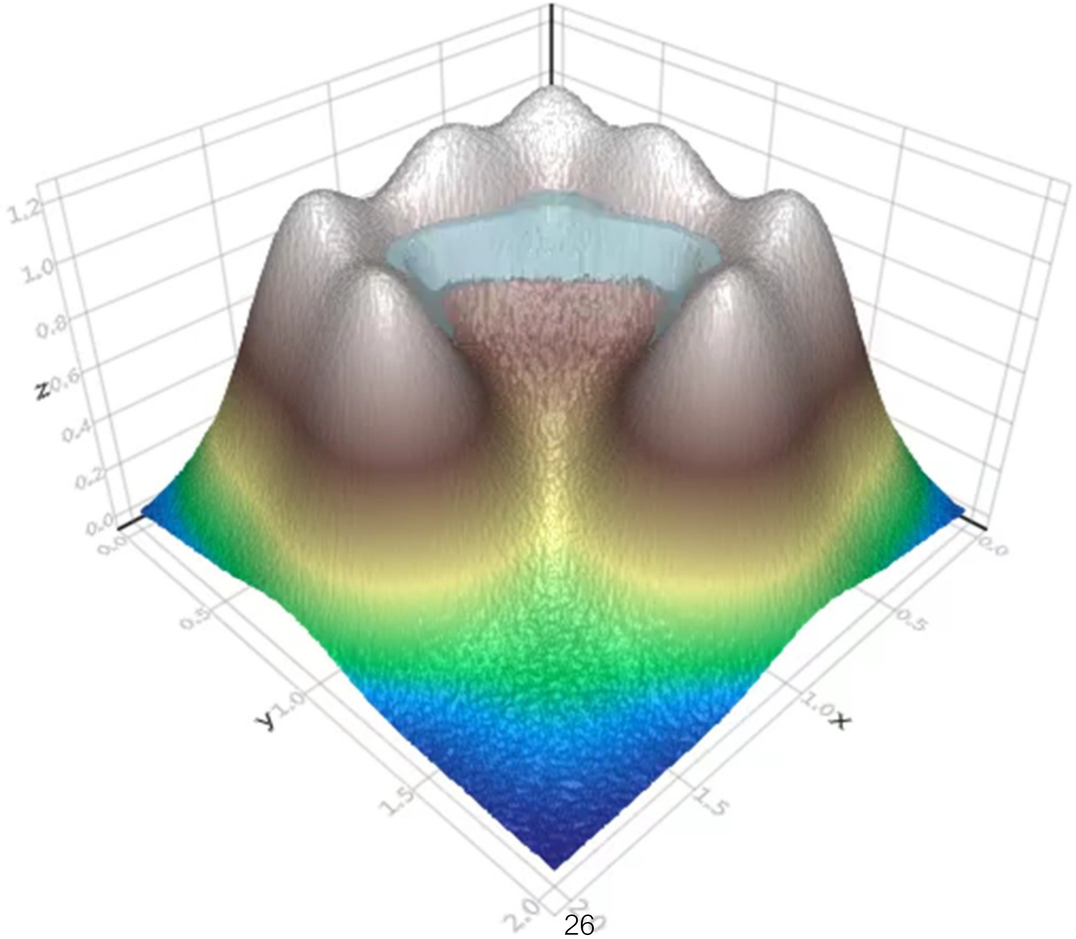


Node Type: GPU

Nodes: 1

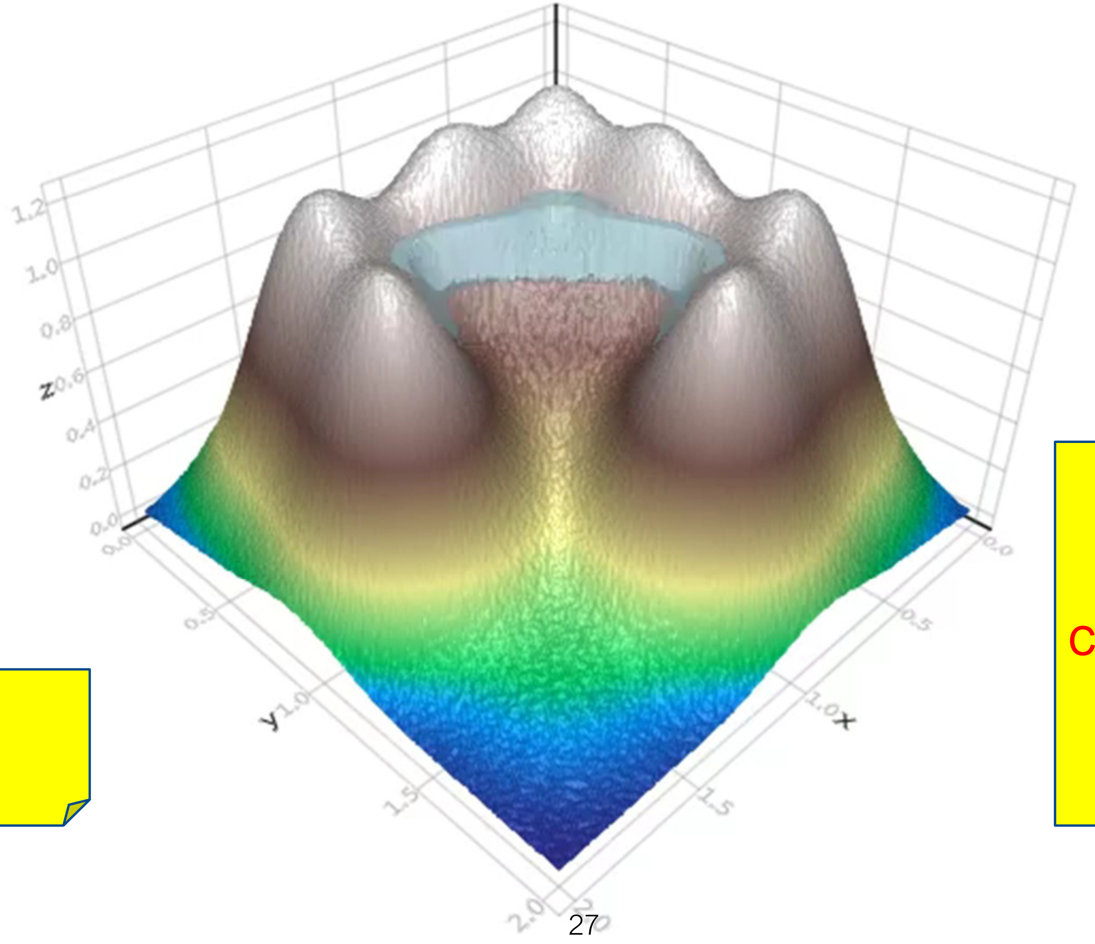
Duration (hr): 1

3-D OpenGL visualization in Julia (different topography)



CSCS ETHzürich

3-D OpenGL visualization in Julia (different topography)



Uses Makie.jl

Done on Laptop.
We will see if Makie.jl can be installed after Piz Daint upgrade.

Agenda

- Introduction: the two language problem ✓
- 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
- We will see if **`Makie.jl`** can be installed after Piz Daint upgrade.

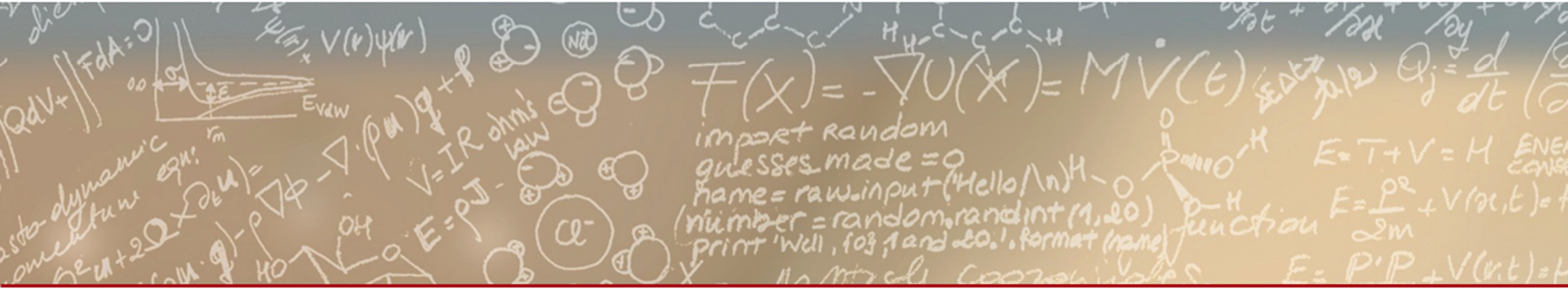
Conclusions & outlook

- same stacked environment in JupyterLab as when using Julia from command line
- `CUDA.jl` enables writing native Julia code for GPUs
- We will see if **`Makie.jl`** can be installed after Piz Daint upgrade.

Questions / advice / feedback / …

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

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


F(X) = -∇U(X) = MV(t)
import random
guesses_made = 0
name = raw_input('Hello\n')
number = random.randint(1, 20)
print 'Well, {0}, 1 and 20.'.format(name)
E = T + V = H
V = IR ohm's law
E = ρJ
function

Thank you for your kind attention