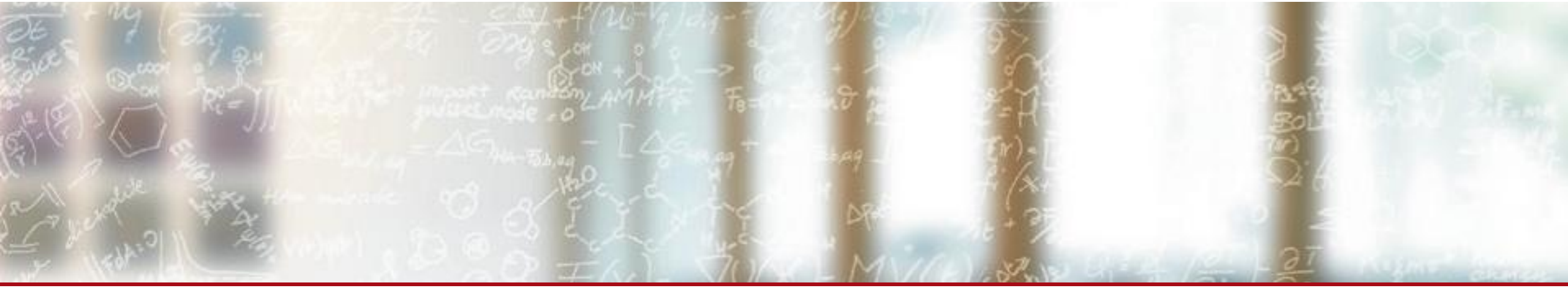
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Interactive computing on Alps

- **Jupyter** notebooks via JupyterHub
- **Julia** in JupyterHub, with GPU profiling
- Best practices for setting up **VSCode**

27 May 2025

Today’s topics

1. Introduction to **JupyterHub** on Alps vClusters
2. Using **Julia** with JupyterHub, including GPU profiling
3. Best practices for setting up **VSCode** on Alps vClusters

Tim Robinson

HPC Platform Service Manager

Samuel Omlin

Research Software Engineer

Prashanth Kanduri

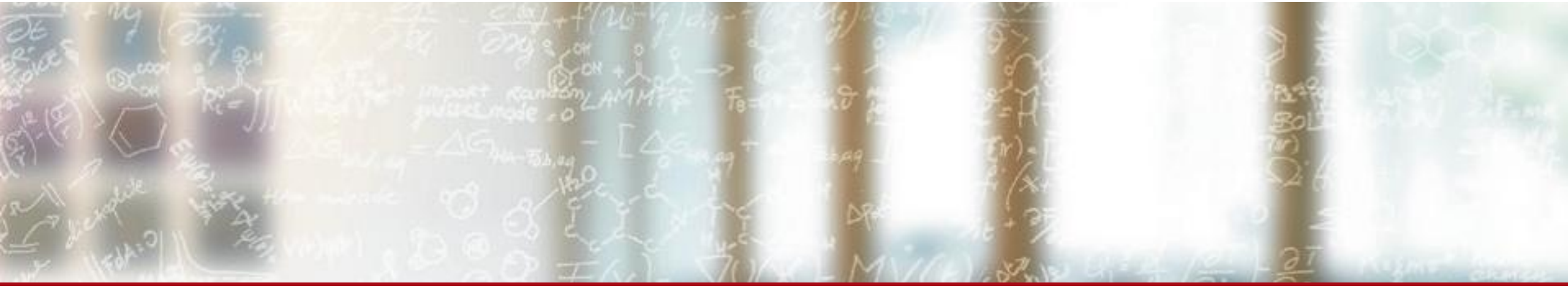
Research Software Engineer

JupyterHub

- Run notebooks on compute nodes on Alps
- Separate deployment per vCluster
  - https://jupyter-daint.cscs.ch
  - https://jupyter-santis.cscs.ch
  - https://jupyter-clariden.cscs.ch
- Documentation is available in CSCS Docs
- Software environment is provided by uenv
  - You can use your own virtual environment by creating a kernel
  - You can load your own uenv (⊕Advanced options)
- Initial support for container images
- Happy to receive your feedback and requests for enhancement

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Using Julia in JupyterHub

Webinar at the Swiss National Supercomputing Centre (CSCS), ETH Zurich

Dr. Samuel Omlin[1]

May 27 2025

[1]Swiss National Supercomputing Centre (CSCS), ETH Zurich

Agenda

- Introduction: Julia for HPC
- Setting up Julia in JupyterHub
- GPU profiling using Julia
- Documentation
- Conclusions

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Introduction: Julia for HPC

Julia for HPC: low level CUDA kernel

```
function cumsum_dim1!(B, A)
    iy = (blockIdx().y-1) * blockDim().y + threadIdx().y
    iz = (blockIdx().z-1) * blockDim().z + threadIdx().z
    tx = threadIdx().x
    pow2 = Int(log2(blockDim().x))
    cumsum_ix = 0.0
    for ix_offset = 0 : blockDim().x : size(A,1)-1
        ix = threadIdx().x + ix_offset
        @inbounds val = A[ix,iy,iz]              # Read the x-dimension chunk into thread-local registers.
        cumsum_tx = val
        pos   = 1
        width = 2
        for i = 1:pow2
            cumsum_lower = shfl_sync(CUDA.FULL_MASK, cumsum_tx, pos, width)
            if ((tx-1) % width) > pos-1
                cumsum_tx += cumsum_lower
            end
            pos   *= 2                           # pos   = 2^(pow2-1)
            width *= 2                           # width = 2^pow2
        end
        cumsum_tx += cumsum_ix
        @inbounds B[ix,iy,iz] = cumsum_tx        # Write the x-dimension chunk of results to main memory.
        cumsum_ix = shfl_sync(CUDA.FULL_MASK, cumsum_tx, blockDim().x)
    end
    return nothing
end
```

~ theoretical peak performance
(effective memory throughput)

Julia for HPC: high-level ParallelStencil kernel (architecture-agnostic)

math-close notation and in an architecture-agnostic fashion, using a single equation with second order derivatives:

```
@parallel function diffusion2D_step!(T2, T, Ci, lam, dt, dx, dy)
    @inn(T2) = @inn(T) + dt*(lam*@inn(Ci)*(@d2_xi(T)/dx^2 + @d2_yi(T)/dy^2));
    return
end
```

or using multiple equations with first order derivatives (note that `qx`, `qy` and `dTdt` will be computed on the fly...):

```
@parallel function diffusion3D_step!(T2, T, Ci, lam, dt, dx, dy)
    @all(qx)    = -lam*@d_xi(T)/dx                          # Fourier's law of heat conduction
    @all(qy)    = -lam*@d_yi(T)/dy                          # ...
    @all(dTdt) = @inn(Ci)*(-@d_xa(qx)/dx - @d_ya(qy)/dy)   # Conservation of energy
    @inn(T2)    = @inn(T) + dt*@all(dTdt)                   # Update of temperature
    return
end
```

Then the kernel can be called simply as:

```
@parallel diffusion2D_step!(T2, T, Ci, lam, dt, dx, dy)
```

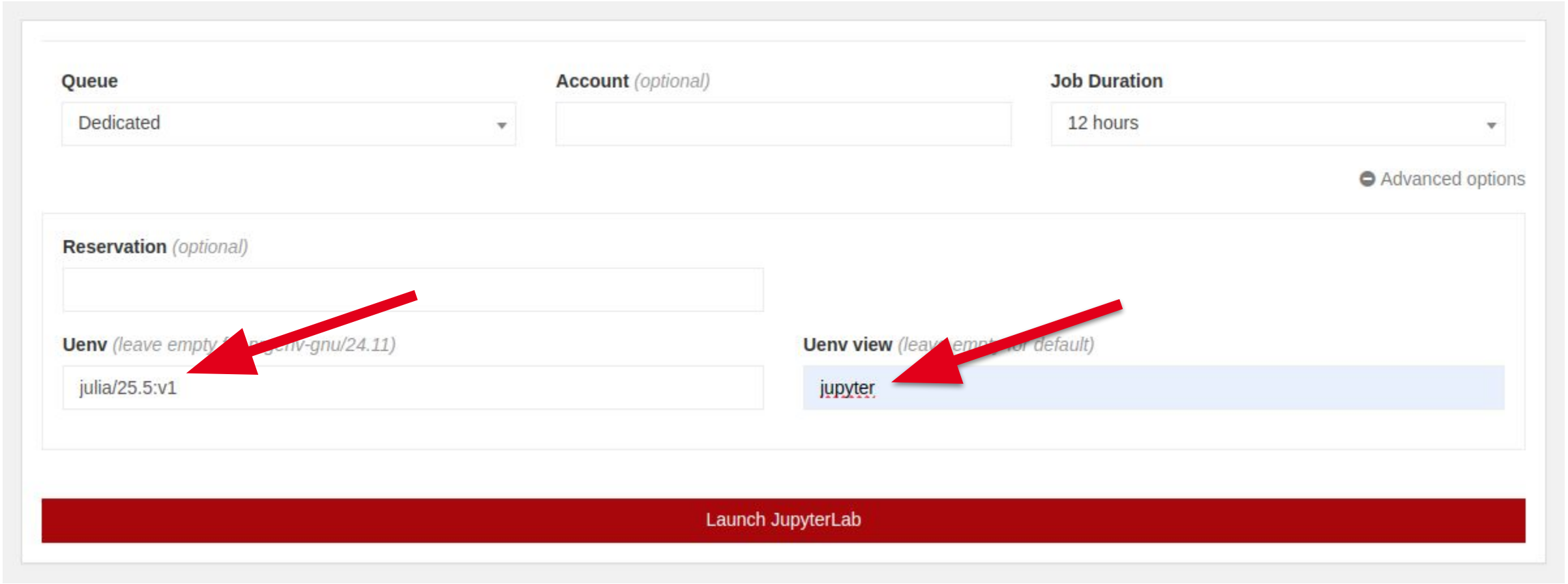
code generation & optimization
=>
near theoretical peak performance
(effective memory throughput)

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

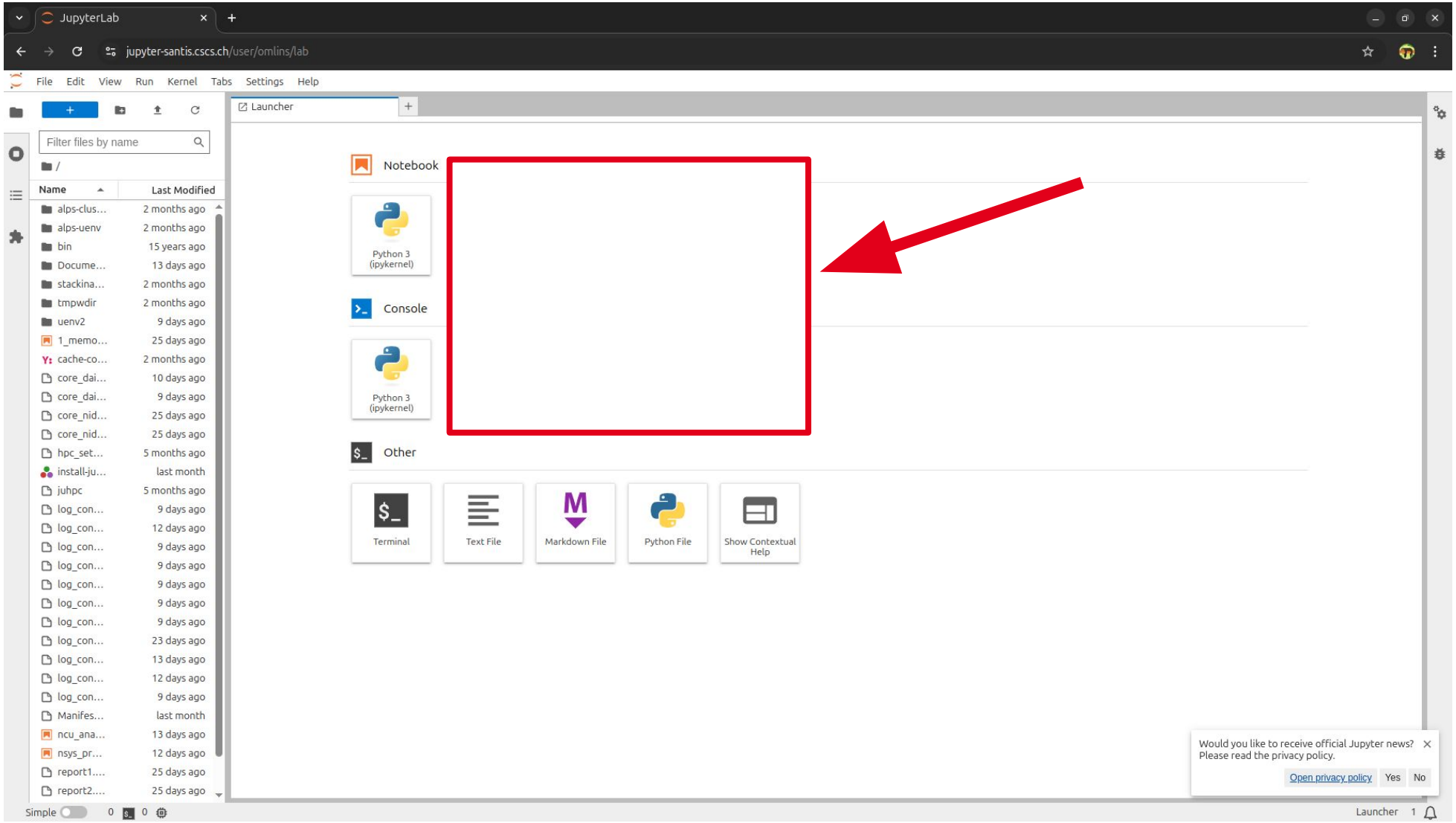
ETH zürich

Setting up Julia in JupyterHub

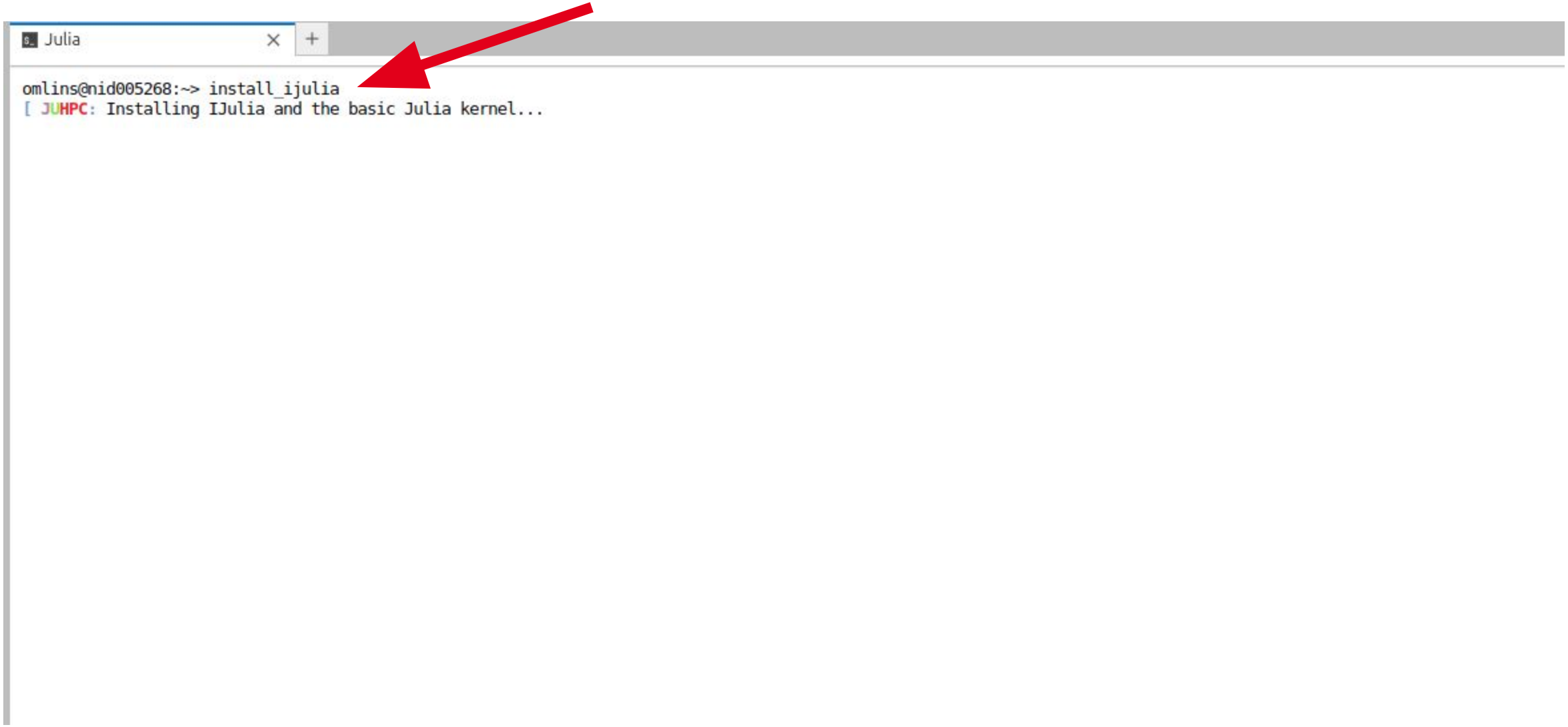
Start a Julia-enabled server



First time IJulia & kernel installation



First time IJulia & kernel installation

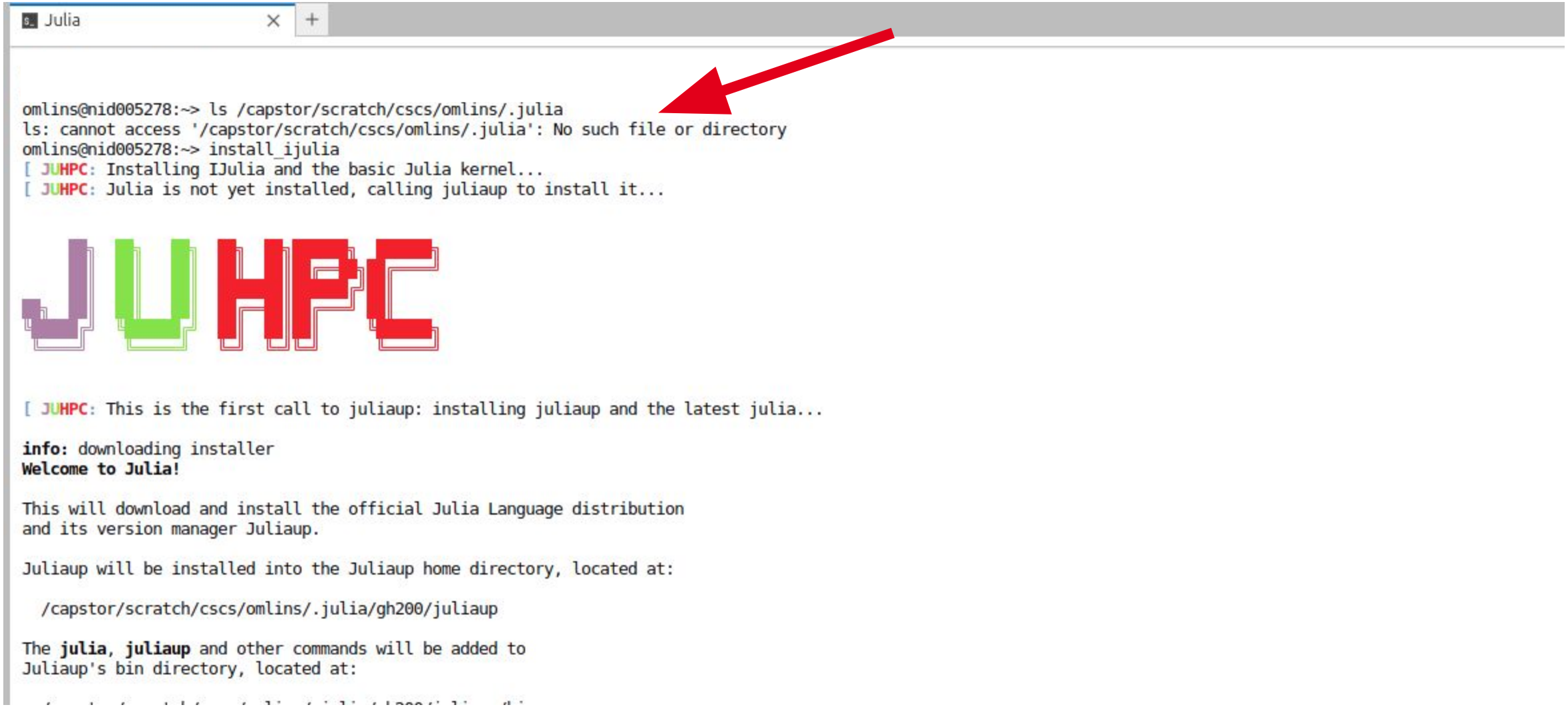


First time IJulia & kernel installation

Julia

```
  [a63ad114] + Mmap v1.11.0
  [ca575930] + NetworkOptions v1.2.0
  [44cfe95a] + Pkg v1.11.0
  [de0858da] + Printf v1.11.0
  [3fa0cd96] + REPL v1.11.0
  [9a3f8284] + Random v1.11.0
  [ea8e919c] + SHA v0.7.0
  [6462fe0b] + Sockets v1.11.0
  [f489334b] + StyledStrings v1.11.0
  [fa267f1f] + TOML v1.0.3
  [a4e569a6] + Tar v1.10.0
  [cf7118a7] + UUIDs v1.11.0
  [4ec0a83e] + Unicode v1.11.0
  [deac9b47] + LibCURL_jll v8.6.0+0
  [e37daf67] + LibGit2_jll v1.7.2+0
  [29816b5a] + LibSSH2_jll v1.11.0+1
  [c8ffd9c3] + MbedTLS_jll v2.28.6+0
  [14a3606d] + MozillaCACerts_jll v2023.12.12
  [83775a58] + Zlib_jll v1.2.13+1
  [8e850ede] + nghttp2_jll v1.59.0+0
  [3f19e933] + p7zip_jll v17.4.0+2
        Info Packages marked with ⌅ have new versions available but compatibility constraints restrict them from upgrading. To see why use `status --outdated -m`
    Building Conda → `/capstor/scratch/cscs/omlins/.julia/gh200/juliaup/depot/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/b19db3927f0db4151cb86d073689f2428e52
    Building IJulia → `/capstor/scratch/cscs/omlins/.julia/gh200/juliaup/depot/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/be30be76e25b0aff2c9a85930ed3ac34c5f1
Precompiling project...
  20 dependencies successfully precompiled in 35 seconds. 21 already precompiled.
[ Info: Installing Julia kernelspec in /users/omlins/.local/share/jupyter/kernels/julia-1.11
[ JUHPC: ... done: installation completed.
[ JUHPC: Installing a kernel for Julia under Nsight Systems...
[ Info: Installing Julia under Nsight Systems kernelspec in /users/omlins/.local/share/jupyter/kernels/julia-under-nsight-systems-1.11
[ JUHPC: ... done: installation completed.
omlins@nid005268:~>
```

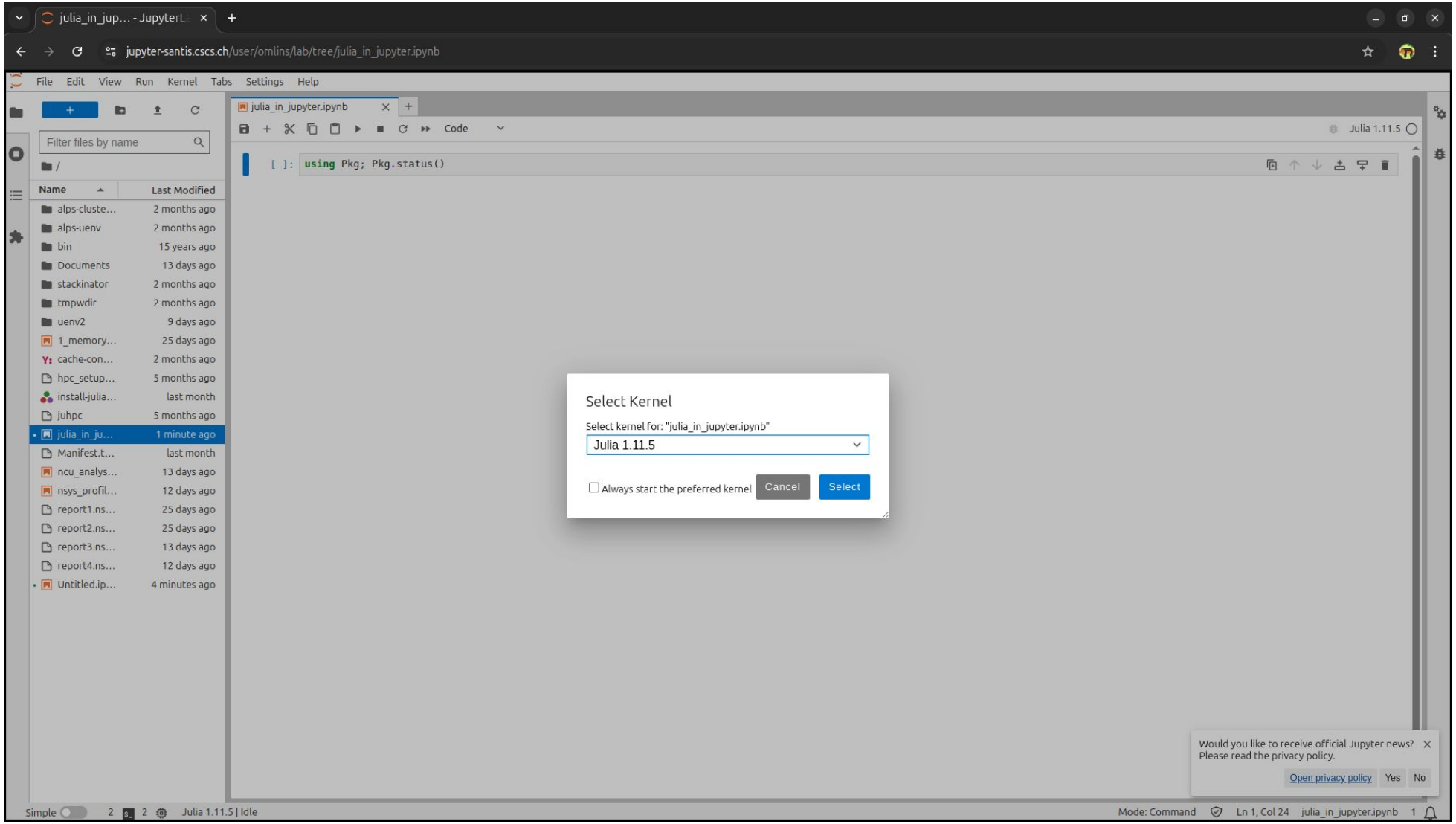
First time IJulia & kernel installation: if Julia missing…



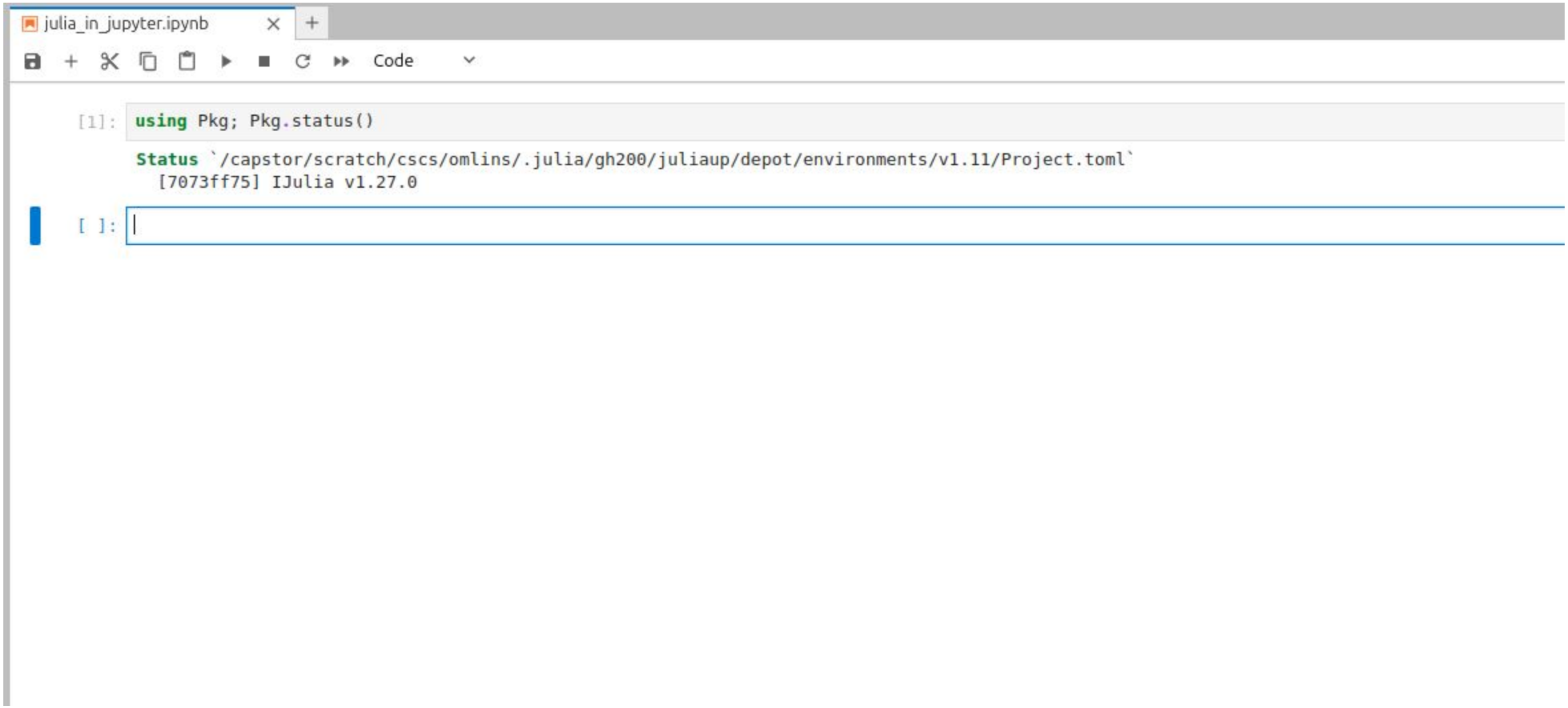
First time IJulia & kernel installation: if Julia missing…

First time IJulia & kernel installation: result

First time IJulia & kernel installation: result



First time IJulia & kernel installation: result



Package installation: predefined preferences and ENV

julia_in_jupyter.ipynb

Code

```
[1]: using Pkg; Pkg.status()
```

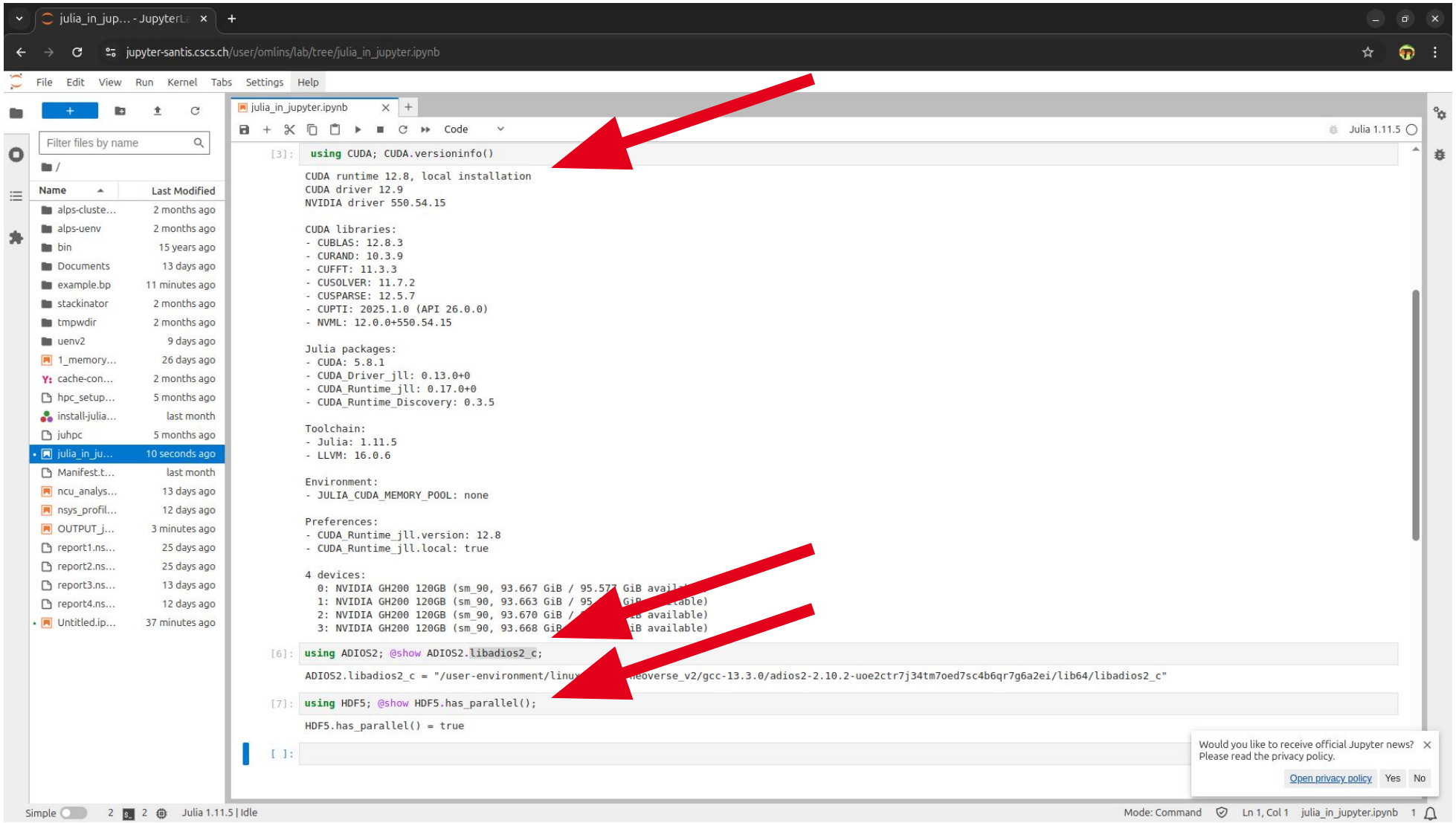
```
Status `/capstor/scratch/cscs/omlins/.julia/gh200/juliaup/depot/environments/v1.11/Project.toml`
  [7073ff75] IJulia v1.27.0
```

```
[2]: Pkg.add(["CUDA", "ADIOS2", "HDF5"]);
```

```
   Updating registry at `/capstor/scratch/cscs/omlins/.julia/gh200/juliaup/depot/registries/General.toml`
  Resolving package versions...
  Installed libaec_jll ──────────────── v1.1.3+0
  Installed GPUArraysCore ───────────── v0.2.0
  Installed MPICH_jll ───────────────── v4.3.0+1
  Installed Crayons ─────────────────── v4.1.1
  Installed Adapt ───────────────────── v4.3.0
  Installed ColorTypes ──────────────── v0.12.1
  Installed Scratch ─────────────────── v1.2.1
  Installed PrettyTables ────────────── v2.4.0
  Installed Zstd_jll ────────────────── v1.5.7+1
  Installed demumble_jll ────────────── v1.3.0+0
  Installed TableTraits ─────────────── v1.0.1
  Installed CUDA_Driver_jll ─────────── v0.13.0+0
  Installed nlohmann_json_jll ───────── v3.11.3+0
  Installed GPUCompiler ─────────────── v1.5.1
  Installed MPI ─────────────────────── v0.20.22
  Installed OpenSSL_jll ─────────────── v3.5.0+0
  Installed Hwloc_jll ───────────────── v2.12.1+0
  Installed LLVMLoopInfo ────────────── v1.0.0
```

For `MPI.jl`, `CUDA.jl`, `HDF5.jl`, and `ADIOS2.jl`, the julia uenv provides the libraries and presets package preferences and environment variables for automatic optimal installation and usage

Package installation: predefined preferences and ENV



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

GPU profiling using Julia

CUDA profiling - using CUDA.jl profiler

julia_in_jupyter.ipynb | CUDA_nsys_profiling.ipynb | CUDA_integrated_profiling.i

Code — Julia 1.11.5

```
[1]: using CUDA

[2]: CuArray([1]) .+ 1;

[3]: CUDA.@profile CuArray([1]) .+ 1
```

```
[3]: Profiler ran for 227.93 µs, capturing 71 events.

Host-side activity: calling CUDA APIs took 106.81 µs (46.86% of the trace)
```

Time (%)	Total time	Calls	Time distribution	Name
36.51%	83.21 µs	1		cuLaunchKernel
4.50%	10.25 µs	2	5.13 µs ± 3.2 (2.86 ‥ 7.39)	cuMemAlloc
1.78%	4.05 µs	1		cuMemcpyHtoDAsync

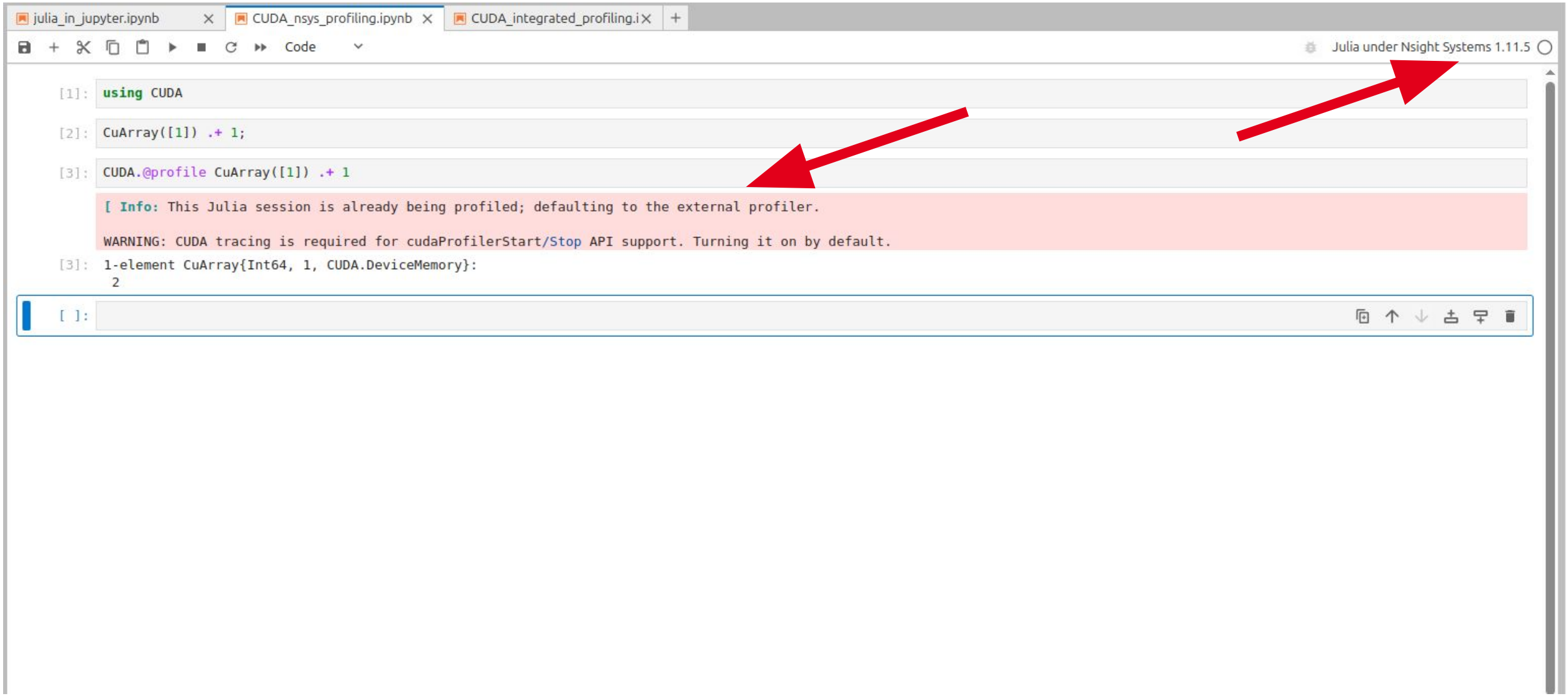
```
Device-side activity: GPU was busy for 2.38 µs (1.05% of the trace)
```

Time (%)	Total time	Calls	Name
0.73%	1.67 µs	1	gpu_broadcast_kernel_linear(CompilerMetadata<DynamicSize, DynamicCheck, void, CartesianIndices<1, Tuple<OneTo<Int64>>>, NDRange<1, DynamicSize, DynamicSize, CartesianIndices<1, Tuple<OneTo<Int64>>>, CartesianIndices<1, Tuple<OneTo<Int64>>>>>, CuDeviceArray<Int64, 1, 1>, Broadcasted<CuArrayStyle<1, DeviceMemory>, Tuple<OneTo<Int64>>, _, Tuple<Extruded<CuDeviceArray<Int64, 1, 1>, Tuple<Bool>, Tuple<Int64>>, Int64>>)
0.31%	715.26 ns	1	[copy pageable to device memory]

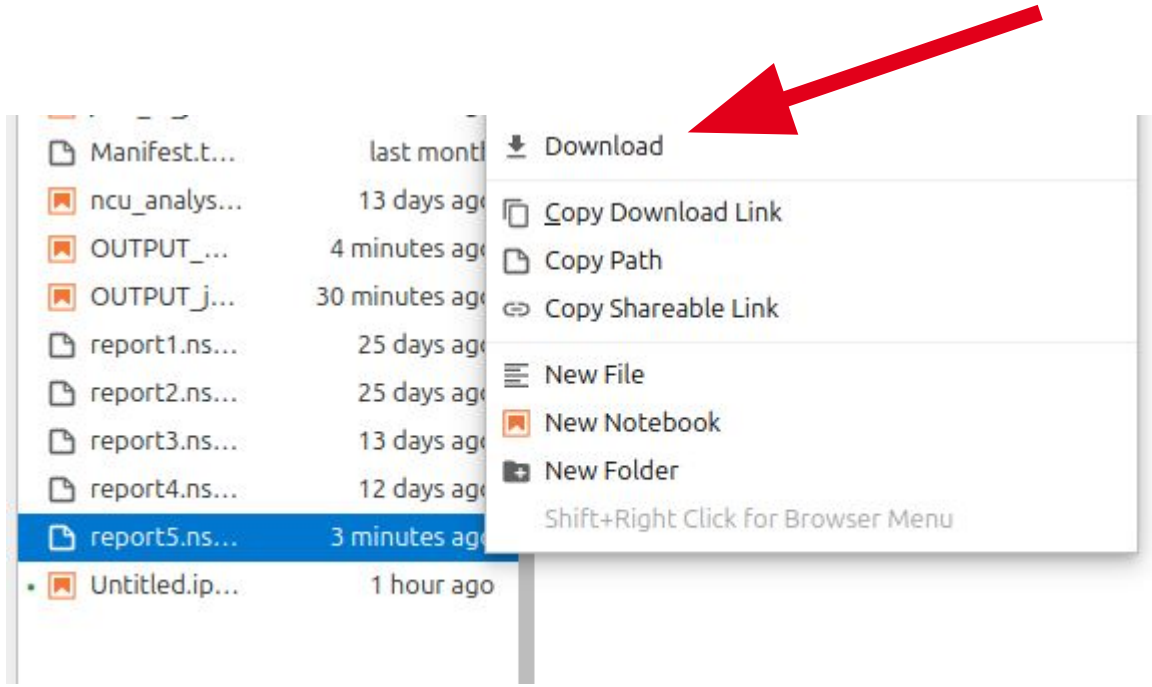
CSCS

ETH zürich

CUDA profiling - using Nvidia Nsight System

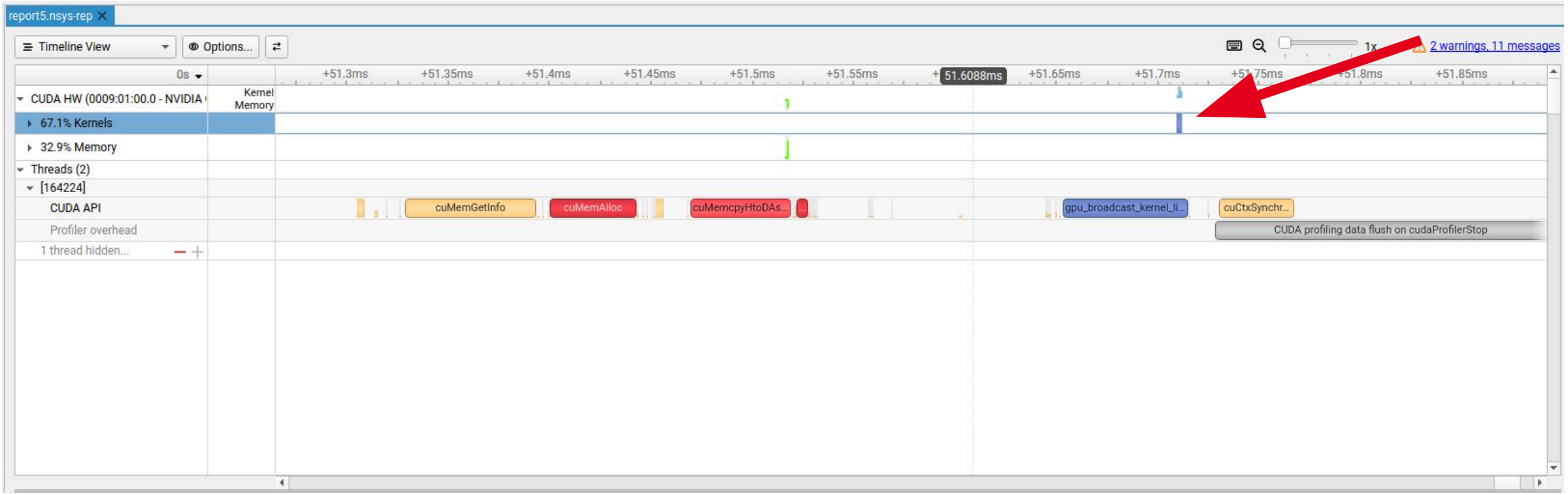


CUDA profiling - using Nvidia Nsight System

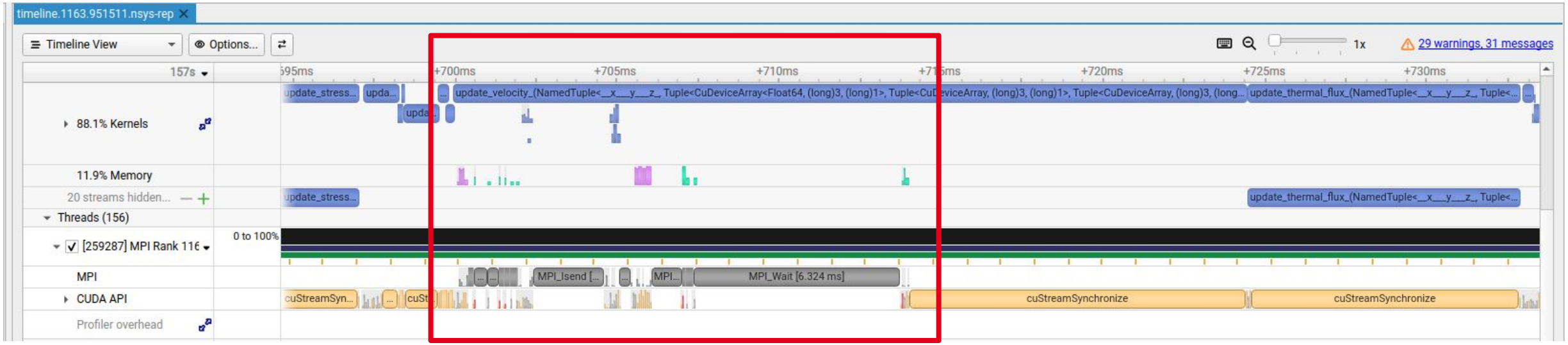



```
omlins@Omlin-Lenovo:~/tmpwdir$ nsys-ui ~/Downloads/report5.nsys-rep
```

CUDA profiling - using Nvidia Nsight System



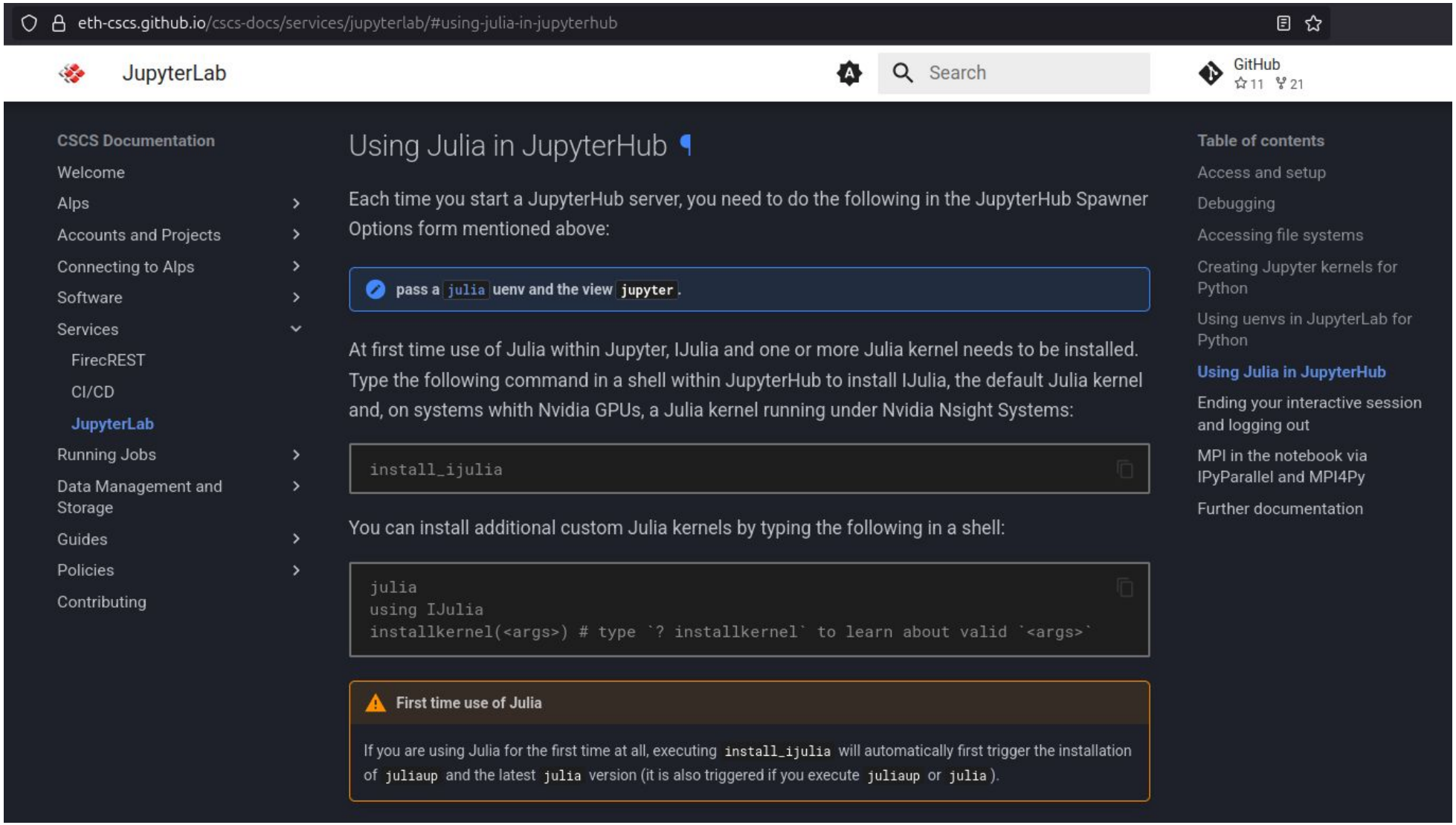
CUDA profiling - using Nvidia Nsight System (real-world example)



Verification that communication is overlapped with computation
(MPI not available in Jupyter)

Documentation

Documentation: docs.cscs.ch/



Documentation: docs.cscs.ch/

Conclusions

Conclusions

- Setting up Julia in JupyterHub is trivial
- Two ways for GPU profiling using Julia
- Julia is a great language for HPC

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

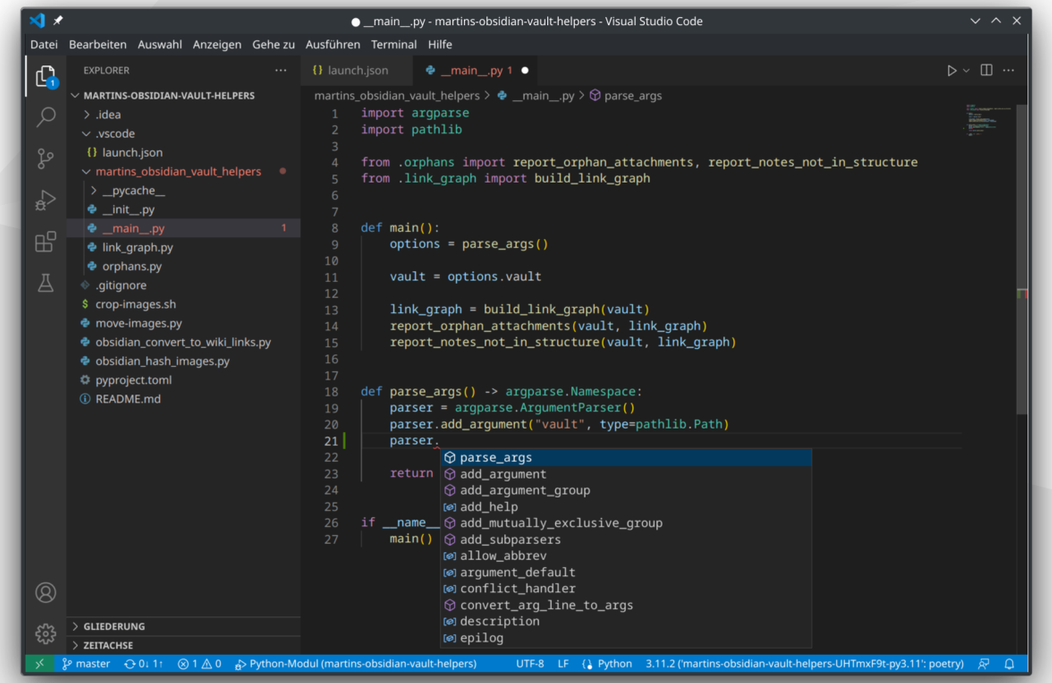
VS Code for Interactive Computing

Productivity and Development Tips

Prashanth Kanduri
& Lukas Drescher

Why use an IDE?

- Intuitive/familiar user experience
- Ease of managing files, visual previews
- Syntax highlighting
- Handy extensions for languages, frameworks and debugging
- Development aids like codebase awareness, code completion, etc
- Easy to setup!



Basic Setup

- Ensure that SSH config and 2FA works and SSH keys are refreshed
- `ssh <clustername>` on the local should get you to the shell of a cluster

SSH config

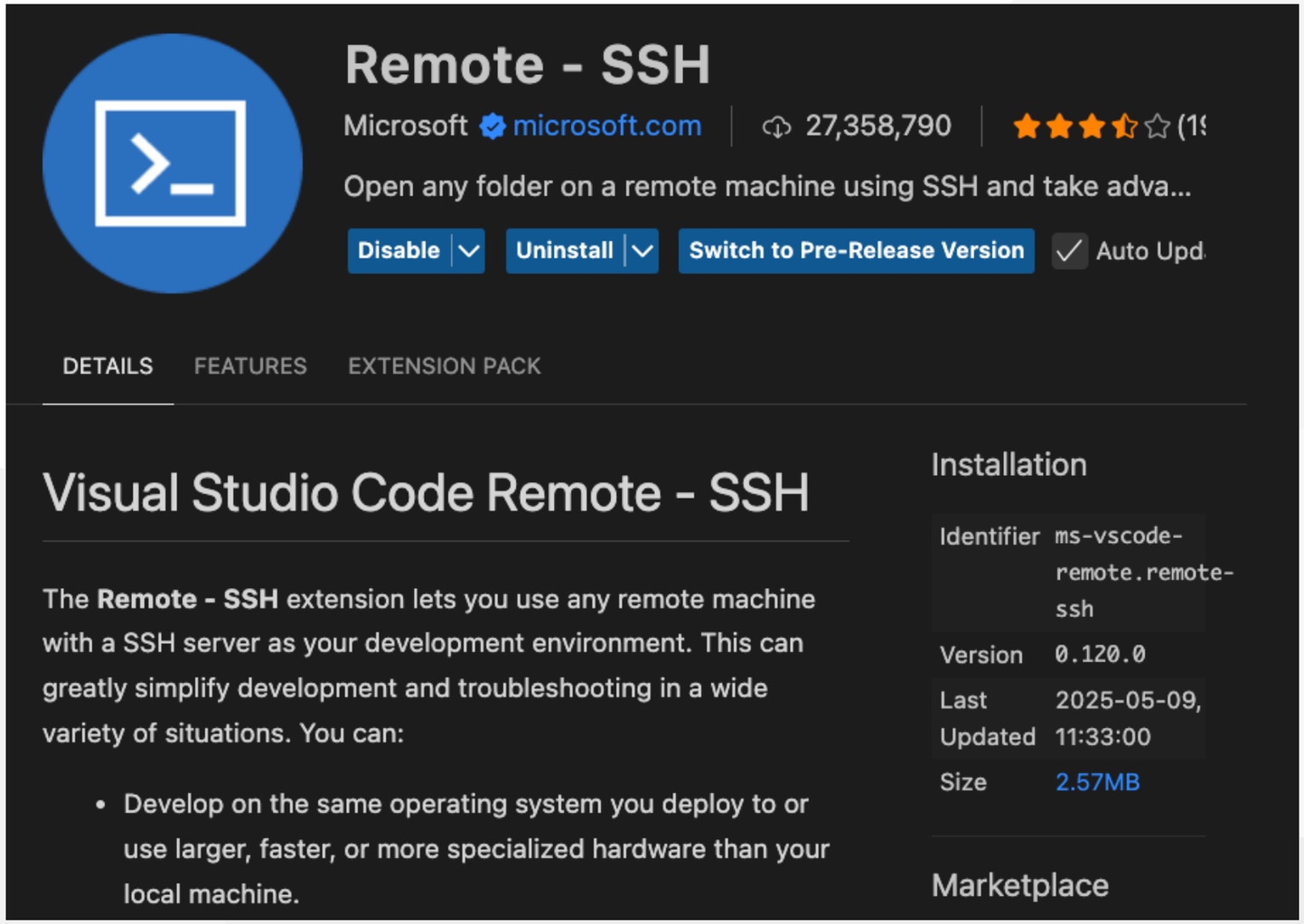
```
Host ela
    HostName ela.cscs.ch
    User <username>
    IdentitiesOnly yes
    IdentityFile ~/.ssh/cscs-key

Host santis
    User <username>
    IdentitiesOnly yes
    IdentityFile ~/.ssh/cscs-key
    ProxyJump ela
```

Password-Less Login

```
➜  ssh santis
Last login: Mon May 26 18:47:17 2025
from ela6.cscs.ch
================================================
   IMPORTANT NOTICE FOR USERS of Alps (Santis)

  Documentation: Alps User Guide -
  https://confluence.cscs.ch/x/mQ_KMg

  Request support: Service Desk at
  https://support.cscs.ch

================================================
```

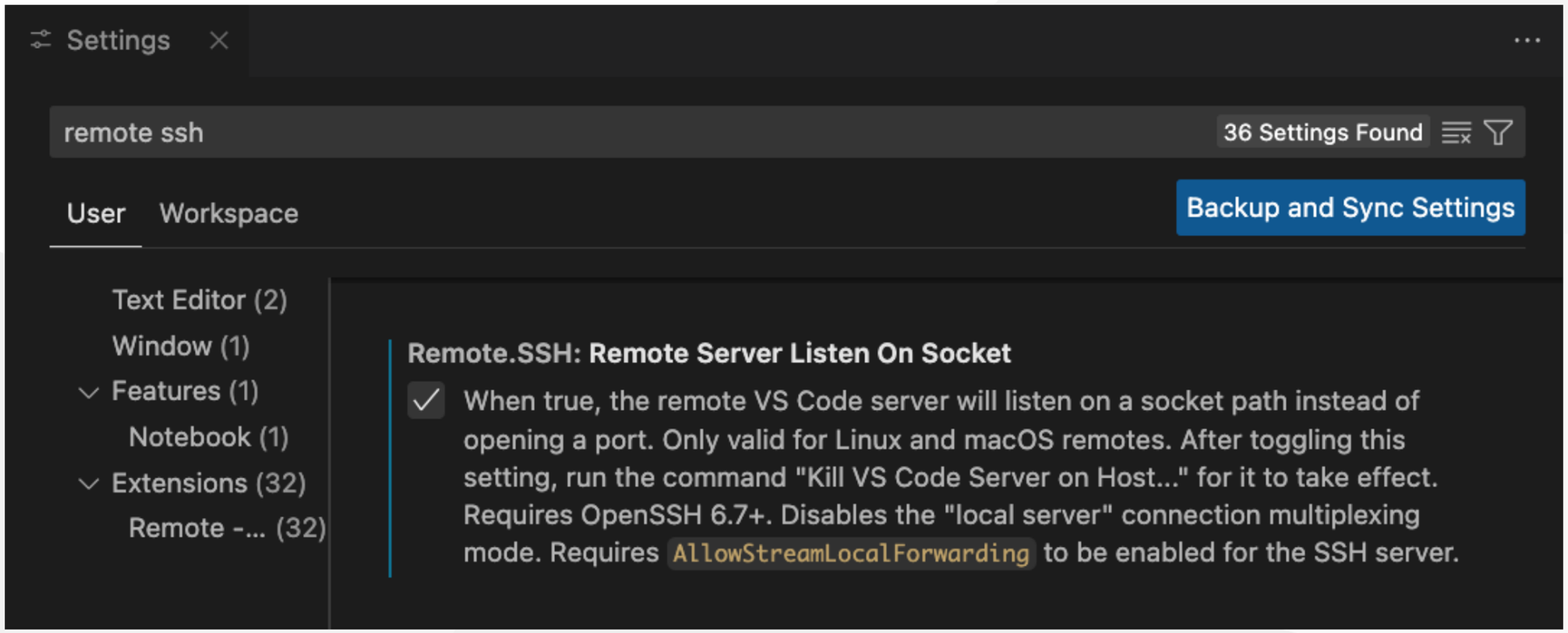
VS Code Setup

Install the Remote-SSH extension by Microsoft from the marketplace.



VS Code Setup cont...

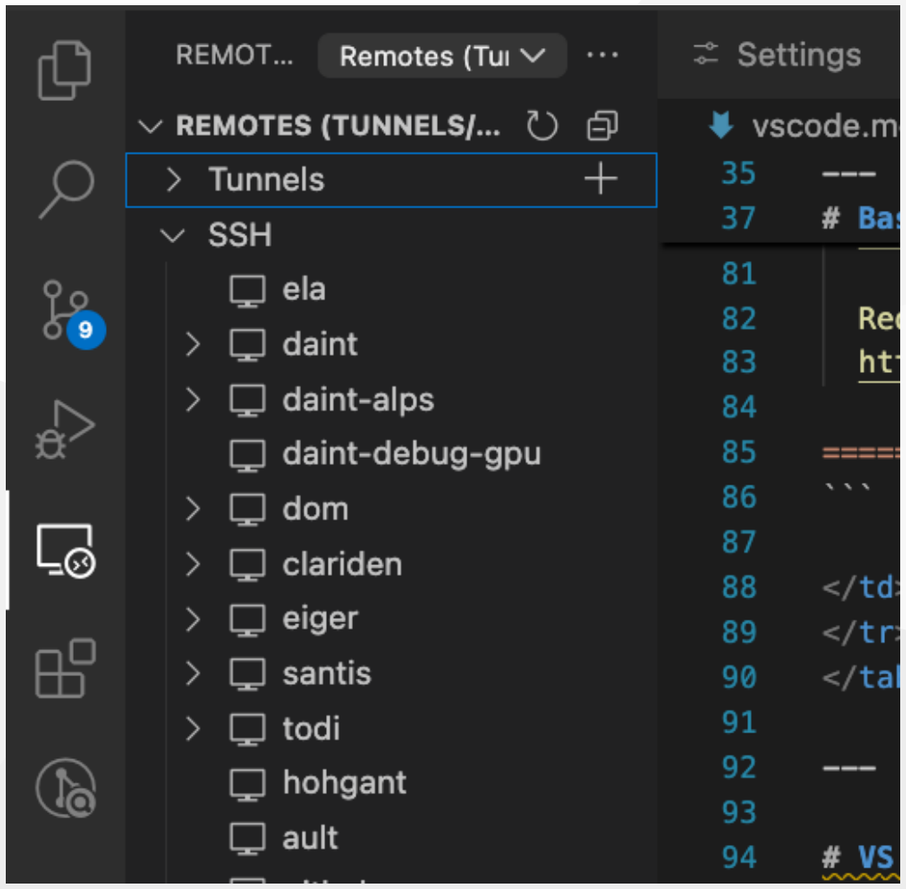
After installation, in the settings, scroll down to the extensions and find the section concerning the one above.
There activate the **Remote Server Listen On Socket** setting.



Connect to a Cluster

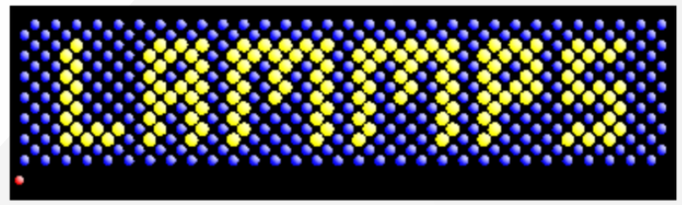
With these settings, restart VS Code and on the left-hand side, find the **Remote Explorer** with the hostnames setup in the `~/.ssh/config` file.

Select the cluster and the window will reload to login. One can select a folder to open there.



Loading Development Environments

PyTorch



NVIDIA CUDA

GCC

FAST. FLEXIBLE. FREE. GROMACS

python™

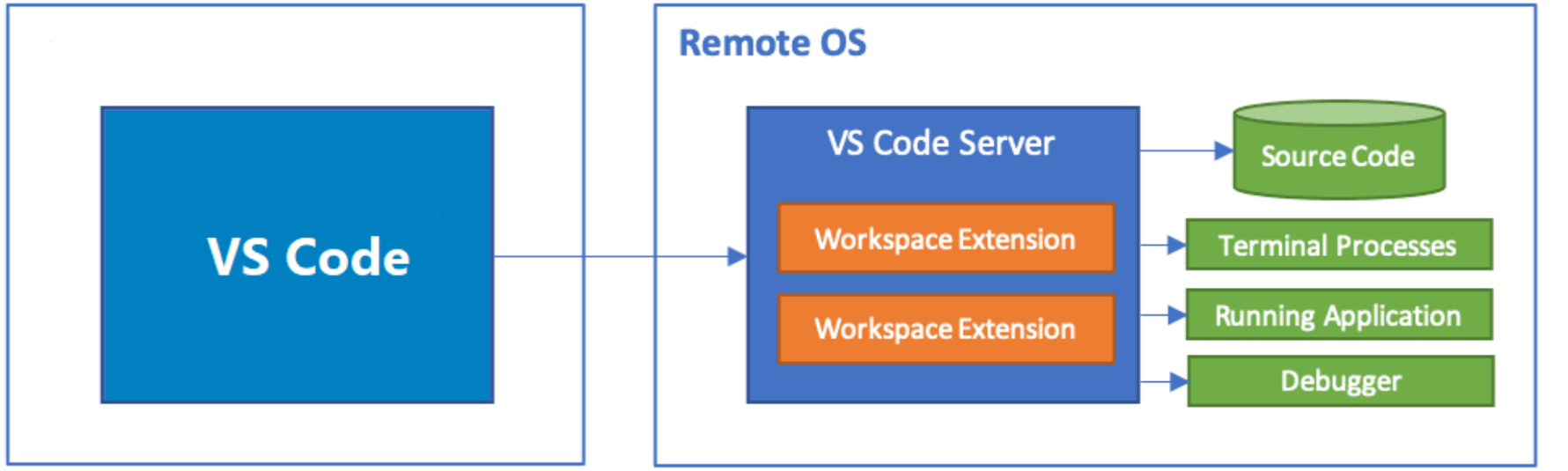
ICON

kokkos

GridTools 4python

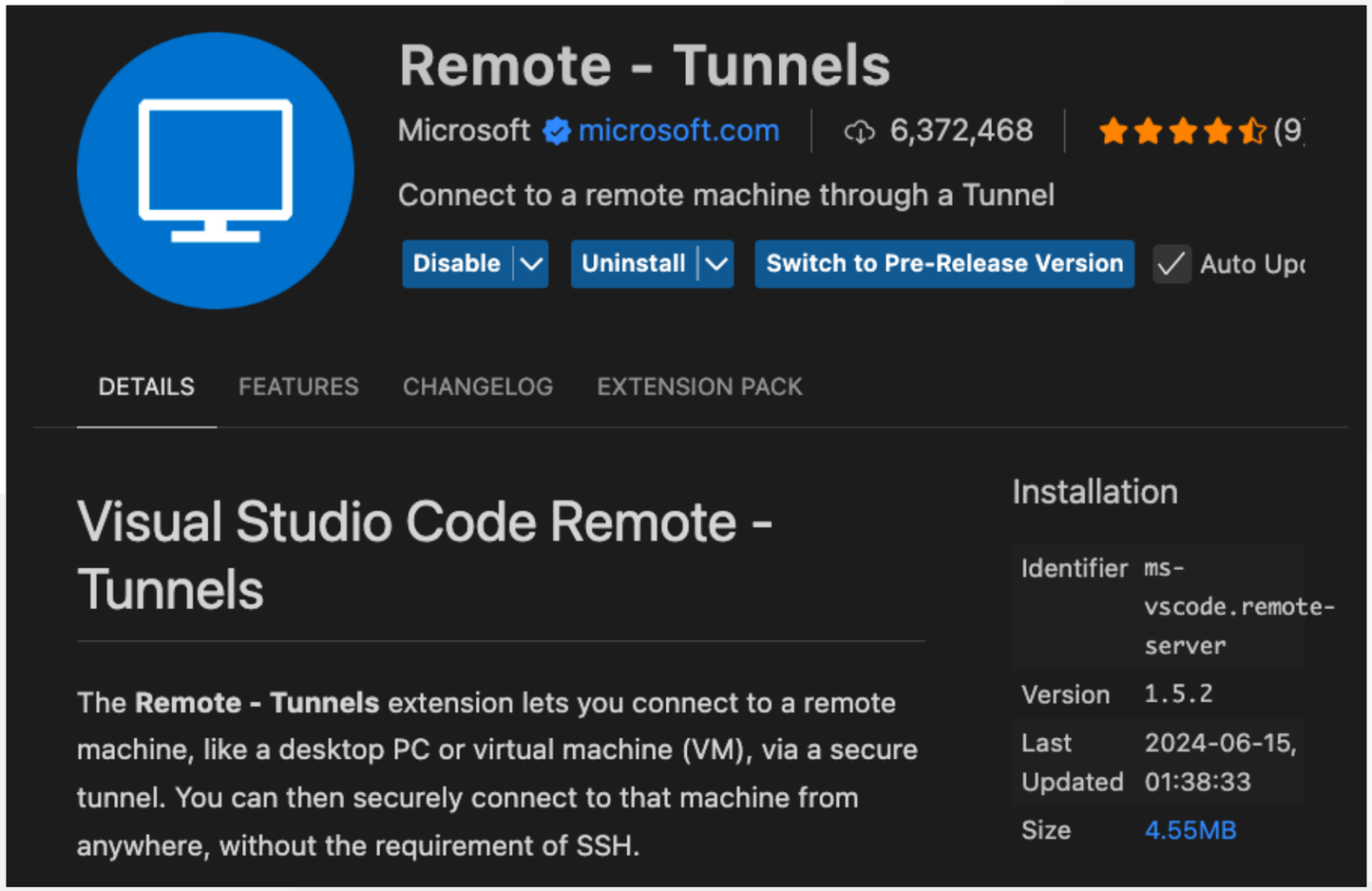
Working with Remote Tunnels

> The Visual Studio Code **Remote - Tunnels** extension lets you connect to a remote machine, like a desktop PC or virtual machine (VM), via a secure tunnel. You can connect to that machine from a VS Code client anywhere, without the requirement of SSH.



Setting up Locally

Install the **Remote Tunnels** extension from Microsoft on VS Code.



Setting up on Alps

Download the VS Code CLI tool `code`, which CSCS provides for easy download. There are two executables, one for using on systems with x86 or ARM CPUs respectively.

`ARM64` → `daint`, `clariden`, `santis`
`x86-64` → `eiger`, `bristen`

Download with the following commands (for ARM64) on the cluster:

```
wget https://jfrog.svc.cscs.ch/artifactory/uenv-sources/vscode/vscode_cli_alpine_arm64_cli.tar.gz
tar -xf vscode_cli_alpine_arm64_cli.tar.gz
```

We will now have an executable named `code` which we shall use. We can make it accessible from anywhere by adding its absolute path to the `PATH` environment variable in our `.bashrc` file.

```
export PATH=/location/of/code/executable:$PATH
```

Tunnel with UENV

On the shell of a cluster, start a uenv session and then a tunnel. For a `uenv` already loaded, one can use the following command.

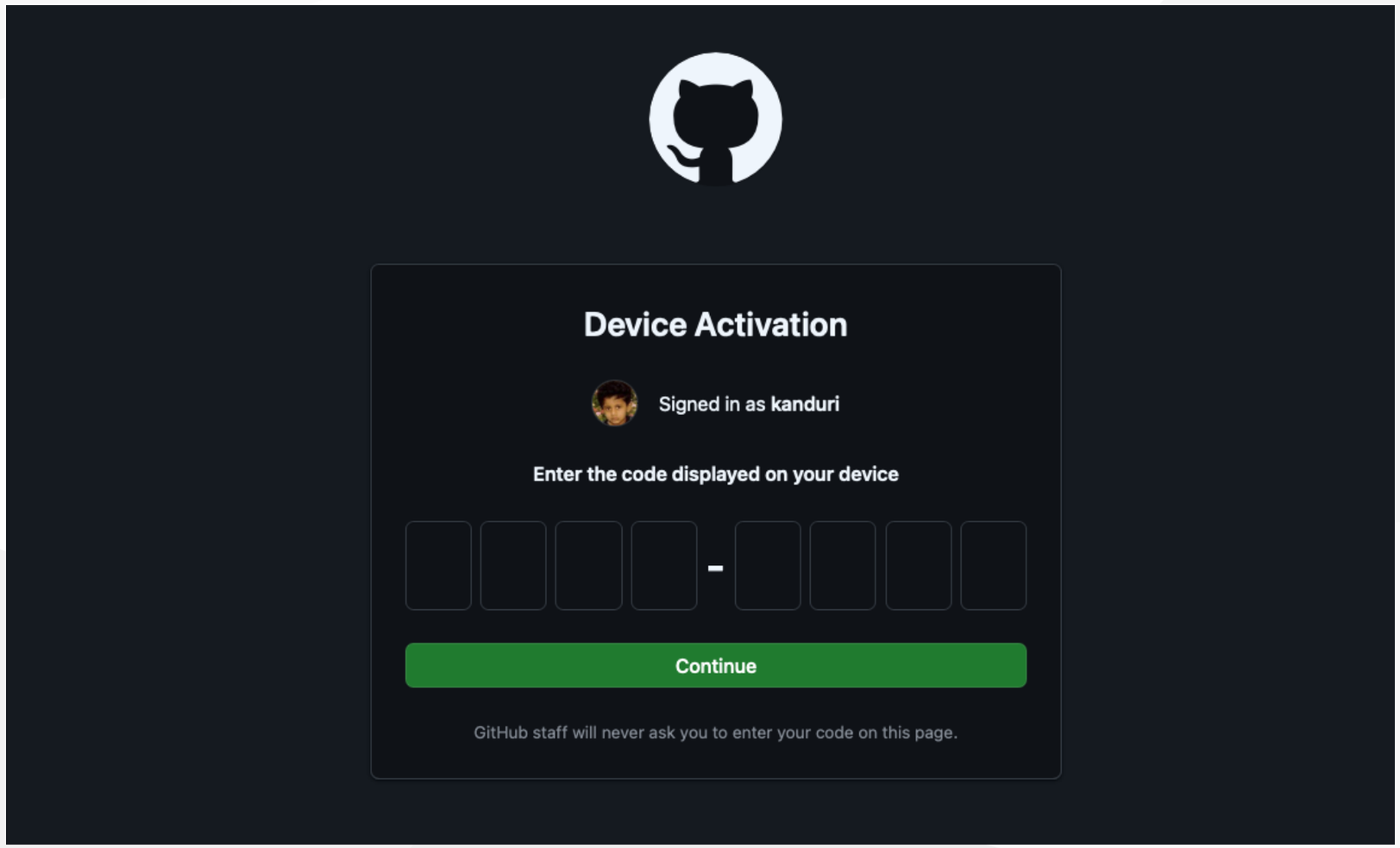
```
# start a uenv session on the login node
uenv start --view=default prgenv-gnu/24.11:v1
# then start the tunnel
code tunnel --name=$CLUSTER_NAME-tunnel
```

It will ask the user to sign in using Microsoft or GitHub account. We have reliably tested it with GitHub.

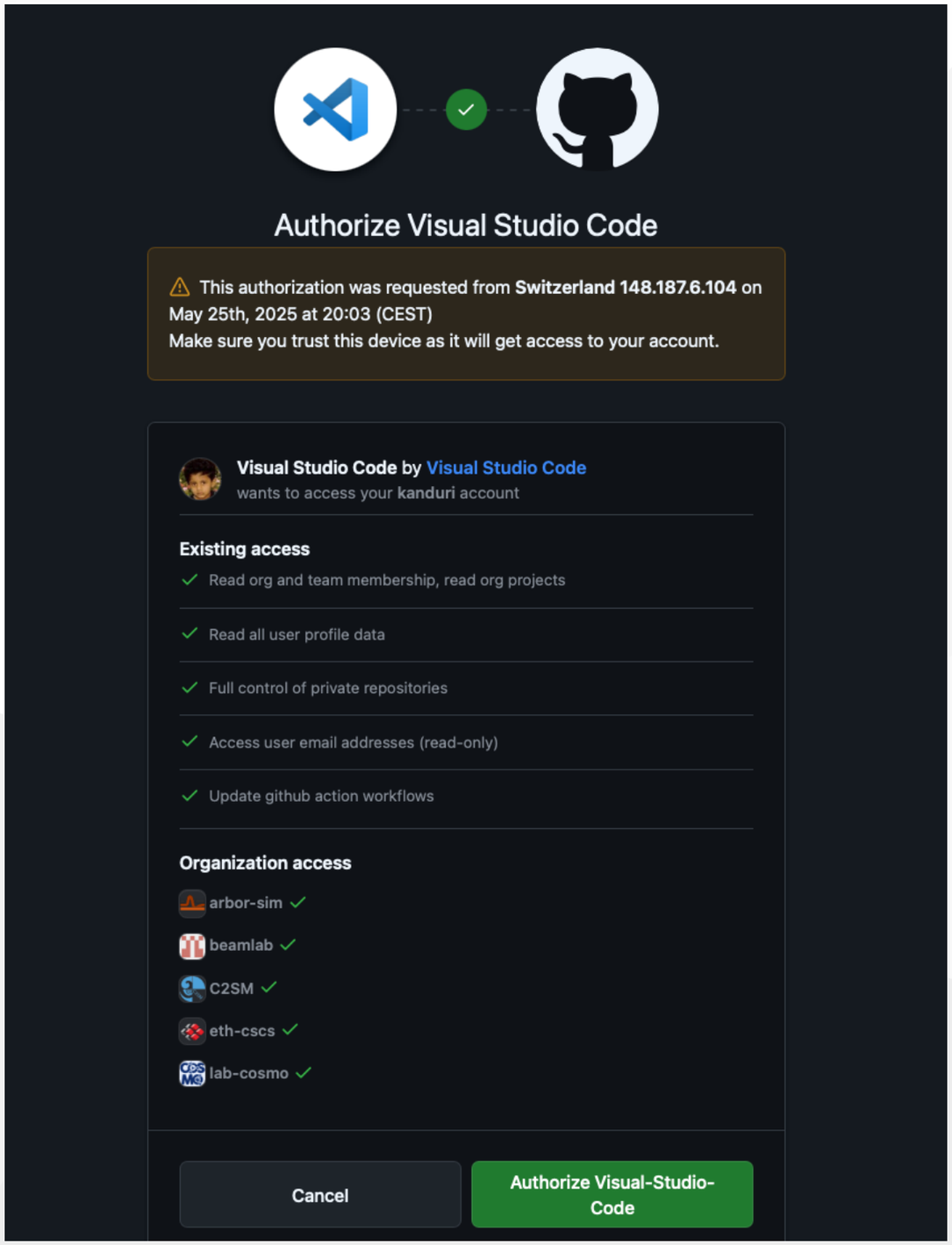
```
> code tunnel --name=$CLUSTER_NAME-tunnel
...
? How would you like to log in to Visual Studio Code? ›
  Microsoft Account
❯ GitHub Account
```

This will begin the first time setup process. It will ask to open a URL on the browser.

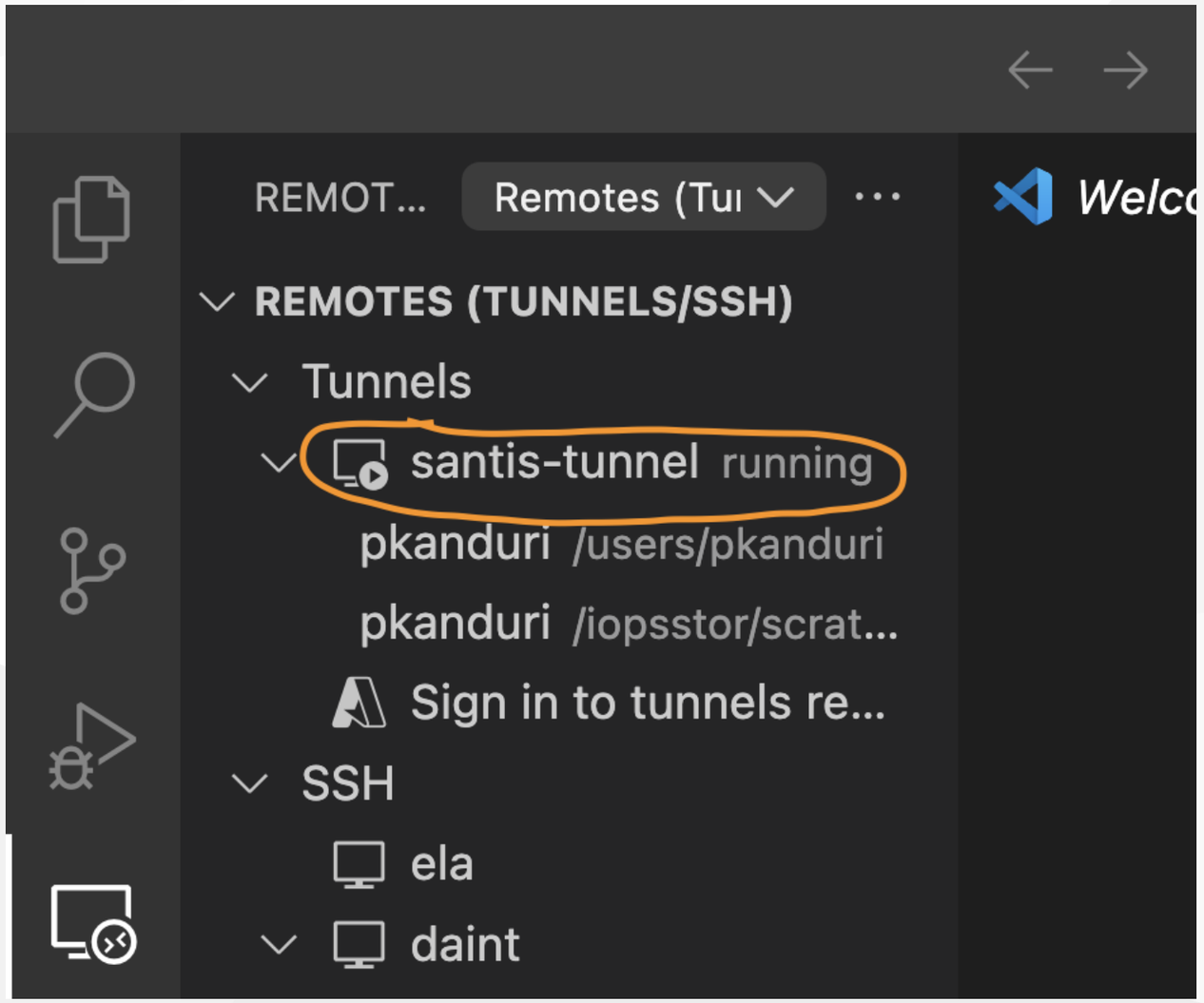
First-Time Setup



First-Time Setup cont...



Access from VS Code



Interactively Connect to a Compute Node

One can log into a compute node and use the same `code tunnel` command.

```
# log into daint
ssh daint

# start an interactive shell session
srun -t 120 -N 1 --pty bash

# set up the environment before starting the tunnel
uenv start prgenv-gnu/24.11:v1 --view=default
code tunnel --name=$CLUSTER_NAME-tunnel
```

- `-t 120` requests a 2 hour (120 minute) reservation
- `-N 1` requests a single rank - only one rank/process is required for VSCode
- `--pty` allows forwarding of terminal I/O, for `bash` to work interactively

Connect VS Code with a Container

This will use CSCS's custom **Container Engine** which can easily pull a container from a registry like DockerHub. Same setup process as earlier with GitHub.

TOML File with Image and Mount Paths

```
image = "nvcr.io#nvidia/pytorch:24.01-py3"
writable = true
mounts = ["/iopsstor/scratch/cscs/username",
      "/users/username/.local/aarch64/bin",
          "/users/username/.bash_history"]
workdir = "/iopsstor/scratch/cscs/username"
```

Launch Container & Tunnel

```
# launch container on compute node
srun -N 1 --environment=/path/to/vscode-pytorch.toml --pty bash
# start tunnel
cd /path/to/code/executable
./code tunnel --name=$CLUSTER_NAME-tunnel
```

Using `code` on Different Architectures

Given the `code` application is architecture specific, and the Home Directory is shared across clusters, we can use a trick to streamline our workflow.

On both `x86` and `ARM` clusters, we can download the respective clusters and store them in specific folders using the `uname -m` command.

```
mkdir -p $HOME/.local/$(uname -m)/bin
cp ./code $HOME/.local/$(uname -m)/bin
```

Then in the `.bashrc` file, we can add the following line which will ensure that the correct executable is available for the architecture.

```
export PATH=$HOME/.local/$(uname -m)/bin:$PATH
```