INTERACTIVE HPC WITH JUPYTERLAB

Training Course – custom Jupyter kernel

2024-04-22..23 I JENS HENRIK GÖBBERT (J.GOEBBERT@FZ-JUELICH.DE)
HERWIG ZILKEN (H.ZILKEN@FZ-JUELICH.DE)

JÜLICH Forschungszentrum

CUSTOM
JUPYTER KERNEL

Jülich Forschungszentrum

TERMINOLOGY

What is a Jupyter Kernel?

Jupyter Kernel
A “kernel” refers to the separate process which executes code cells within a Jupyter notebook.

Jupyter Kernel

- **run code** in different programming languages and environments.
- can be **connected to** a notebook (one at a time).
- **communicates** via ZeroMQ with the JupyterLab.

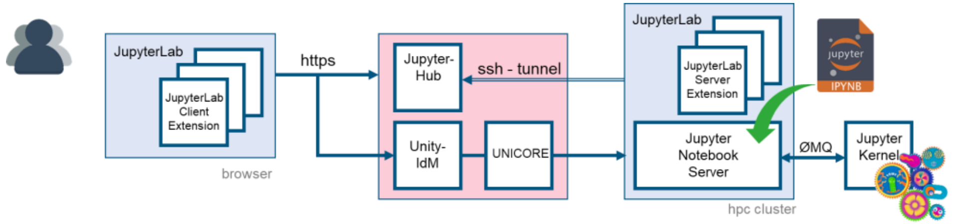
- Multiple **preinstalled** Jupyter Kernels can be found on our clusters
  - Python, R, Julia, Bash, C++, Ruby, JavaScript
  - Specialized kernels for visualization, quantum-computing

- You can easily **create your own kernel** which for example runs your specialized virtual Python environment.

https://jupyter-notebook.readthedocs.io/
https://github.com/jupyter/jupyter/wiki/Jupyter-kernels
https://zeromq.org

JUPYTER KERNEL

How to create your own Juypter Kernel

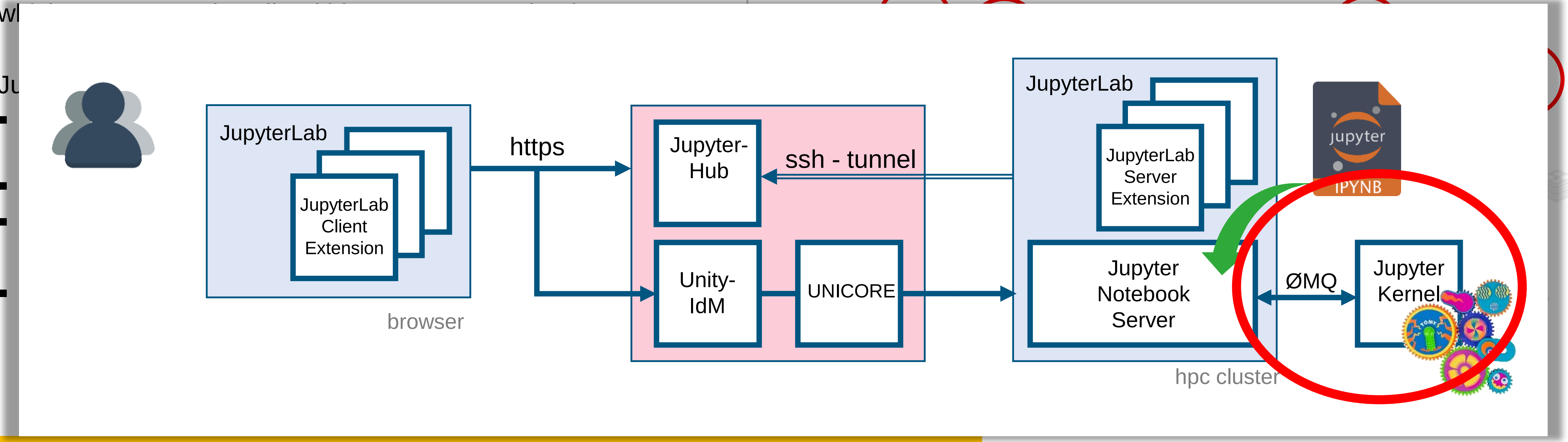


Jupyter Kernel

A “kernel” refers to the separate process

wh

Ju



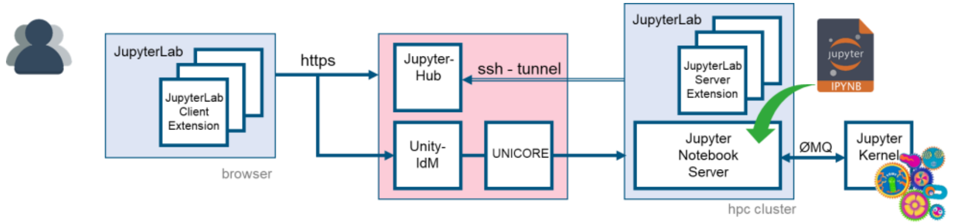
You can easily **create your own kernel** which for example
runs your specialized **virtual Python environment** including **modules of the system.**

https://github.com/jupyter/jupyter/wiki/Jupyter-kernels

JÜLICH Forschungszentrum

JUPYTER KERNEL

How to create your own Juypter Kernel



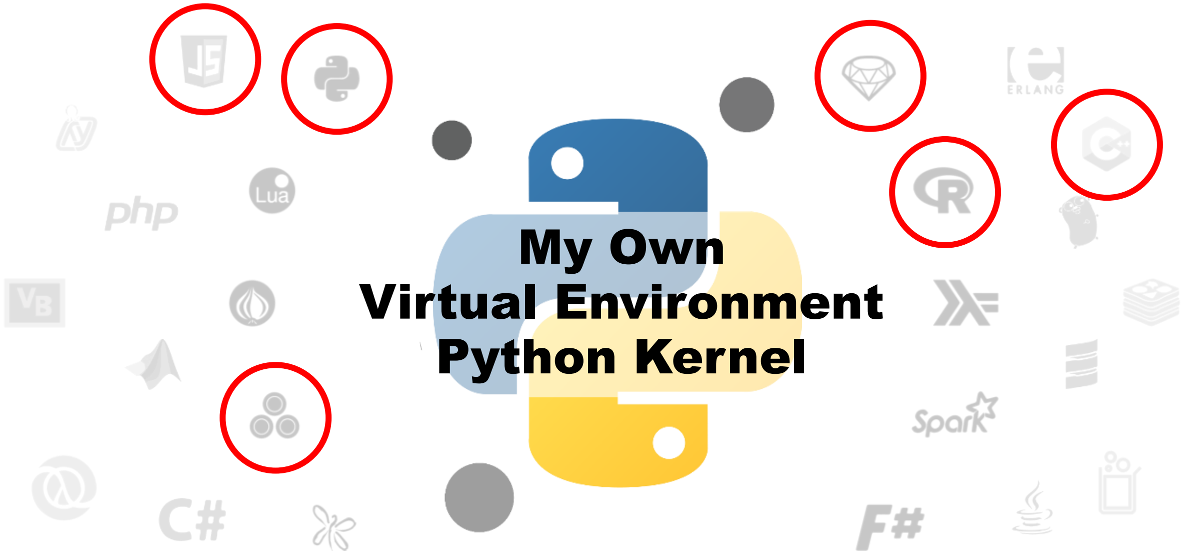
Jupyter Kernel

A “kernel” refers to the separate process
which executes code cells within a Jupyter notebook.

Jupyter Kernel

- run code in different programming languages **and environments**.
- can be connected to a notebook (one at a time).
- communicates via ZeroMQ with the JupyterLab.

- Multiple **preinstalled** Jupyter Kernels can be found on our clusters
    - Python, R, Julia, Bash, C++, Ruby, JavaScript
    - Specialized kernels for visualization, quantum computing



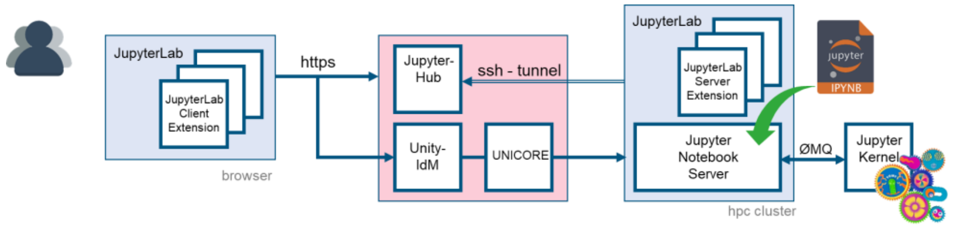
You can easily **create your own kernel** which for example
runs your specialized **virtual Python environment** including **modules of the system.**

https://github.com/jupyter/jupyter/wiki/Jupyter-kernels

JÜLICH Forschungszentrum

JUPYTER KERNEL

How to create your own Juypter Kernel



Jupyter Kernel

A “kernel” refers to the separate process
which executes code cells within a Jupyter notebook.

Jupyter Kernel

- run code in different programming languages **and environments**.
- can be connected to a notebook (one at a time).
- communicates via ZeroMQ with the JupyterLab.

- Multiple **preinstalled** Jupyter Kernels can be found on our clusters
  - Python, R, Julia, Bash, C++, Ruby, JavaScript
  - Specialized kernels for visualization, quantumcomputing

Building your own Jupyter kernel is a three step process

1. Create/Pimp new **virtual Python environment**
   `venv`
2. Create/Edit **launch script** for the Jupyter kernel
   `kernel.sh`
3. Create/Edit Jupyter **kernel configuration**
   `kernel.json`

You can easily **create your own kernel** which for example
runs your specialized **virtual Python environment** including **modules of the system.**

https://github.com/jupyter/jupyter/wiki/Jupyter-kernels

JÜLICH Forschungszentrum

SHORT DIGRESSION:

Lmod (Lua-based Modules) for managing environment modules

What is the problem Lmod solves?

- On a “normal” workstation software is provided in general on system level once.
  It is not required that any distinct shell can change fundamental settings.
- HPC systems need to support **multiple versions software packages**
  - Compilers (e.g. gcc, icc, clang), libraries (e.g. MPI, HDF5), software (e.g. Python)
    ➔ Lmod calls each a **module**

How does Lmod allow to switch between modules?

- Switching between modules is done by
  - Change **environment variables** (most prominent PATH and LD_LIBRARY_PATH)
  - Ensure that **dependencies** to other modules are fulfilled.
    ➔ unload/load modules which conflict/required

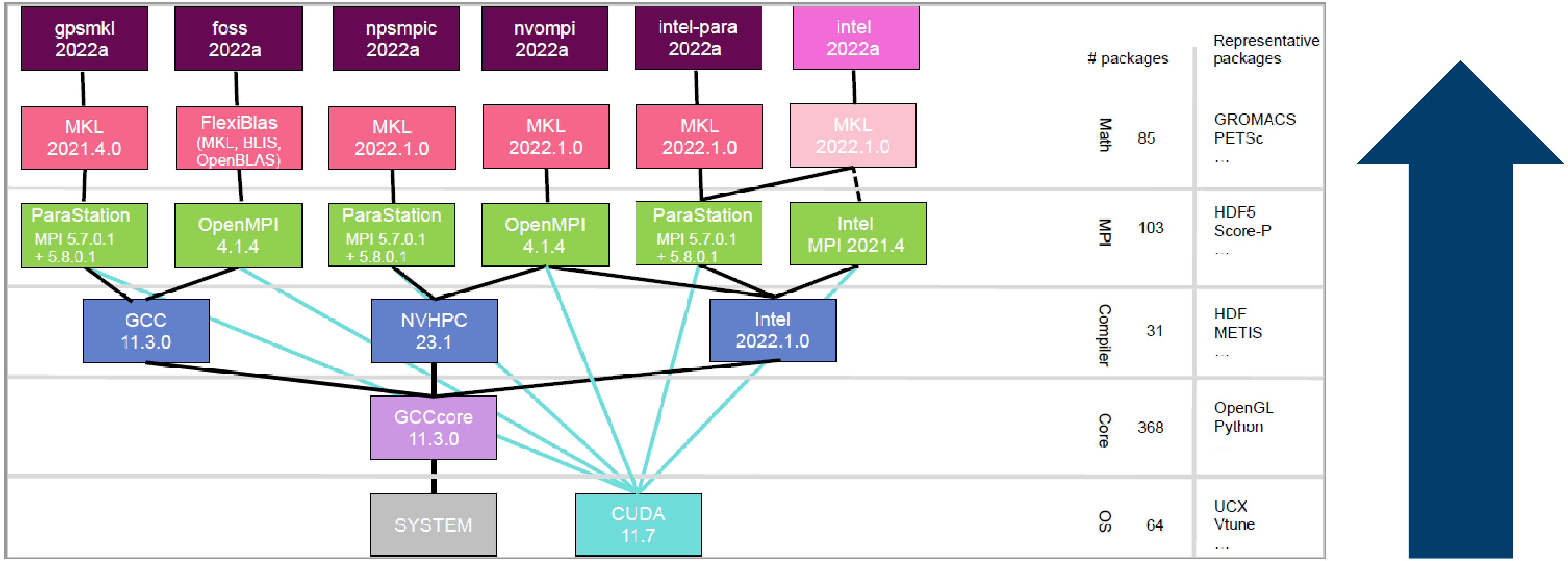
https://lmod.readthedocs.io

JÜLICH
Forschungszentrum

SHORT DIGRESSION:

Lmod (Lua-based Modules) for managing environment modules

The module dependencies are organized a dependency tree (one tree per stage)



Toolchain dependency tree used at Jülich Supercomputing Centre

JÜLICH Forschungszentrum

SHORT DIGRESSION:

Lmod (Lua-based Modules) for managing environment modules

How does Lmod knows how to load a module?

- Lua files in $MODULEPATH
  - Exercise 1: echo $MODULEPATH

Where is the software installed then?

  - /p/software/${SYSTEMNAME}/stages**/<STAGE>/**software/
  - Exercise 2: check the Lua file for the OpenCV module
  - Exercise 3: check the content of this Lua file

SHORT DIGRESSION:

Package manager for high-performance environments

Spack

- “Spack is a multi platform package manager that builds and installs multiple versions and configurations of software”
- https://github.com/spack/spack

Easybuild

- “EasyBuild is a software build and installation framework
  that allows you to manage (scientific) software on High Performance Computing (HPC) an efficient way.”
- https://github.com/easybuilders/easybuild

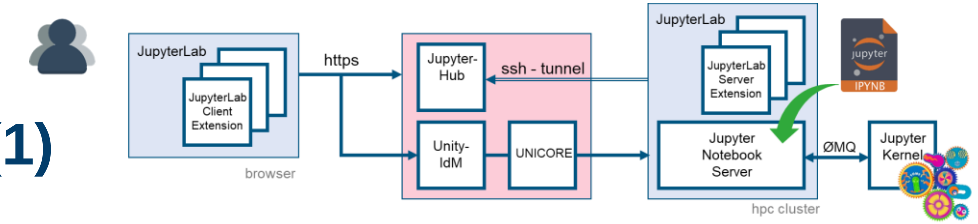
SHORT DIGRESSION:

Virtual Python Environment

- **Isolation:**
  - Self-contained and isolated environment for Python projects
  - Allows to install and manage different versions of Python, libraries, and packages without interfering with other Python
- **Reproducibility:**
  - Recreate the environment in which your code was developed and tested, even on a different machine.
- **Consistency:**
  - Ensures that same versions of Python and packages are used.
  - Reduces the likelihood of compatibility issues and makes it easier to collaborate on a project.
- **Flexibility:**
  - Easily switch between different versions of Python and packages.

JUPYTER KERNEL

1. Create/Pimp new virtual Python environment (1)



1. **Login to JupyterLab and open terminal**

2. **Load required modules**

```
Lnode:> module purge
Lnode:> module load Stages/2024
Lnode:> module load GCC
Lnode:> module load Python
```

3. **Load extra modules you need for your kernel**

```
Lnode:> module load <module you need>
```

1. **Create a virtual environment named <venv_name> at a path of your choice:**

```
Lnode:> python -m venv --system-site-packages <your_path>/<venv_name>
```

2. **Activate your environment**

```
Lnode:> source <your_path>/<venv_name>/bin/activate
```

Building your own Jupyter kernel is a three step process

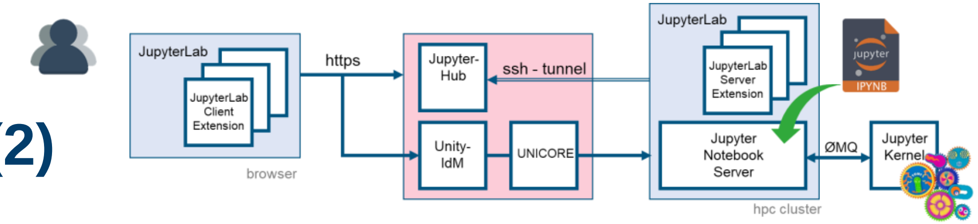
1. Create/Pimp new **virtual Python environment**
   `venv`
2. Create/Edit **launch script** for the Jupyter kernel
   `kernel.sh`
3. Create/Edit Jupyter **kernel configuration**
   `kernel.json`

https://gitlab.version.fz-juelich.de/jupyter4jsc/j4j_notebooks/-/blob/master/001-Jupyter/Create_JupyterKernel_general.ipynb

JÜLICH Forschungszentrum

JUPYTER KERNEL

1. Create/Pimp new virtual Python environment (2)



1. **Ensure python packages installed in the virtual environment are always prefered**

```
(<venv_name>) Lnode:> export PYTHONPATH=\
${VIRTUAL_ENV}/lib/python3.11/site-packages:${PYTHONPATH}
```

2. **Install Python libraries required for communication with Jupyter**

```
(<venv_name>) Lnode:>
       pip install --ignore-installed ipykernel
```

3. **Install whatever else you need in your Python virtual environment (using pip)**

```
(<venv_name>) Lnode:>
       pip install <python-package you need>
```

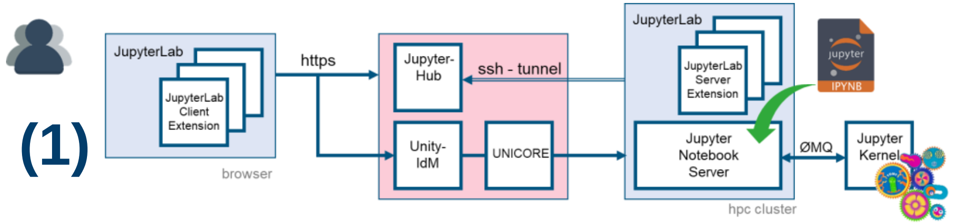
Building your own Jupyter kernel is a three step process

1. Create/Pimp new **virtual Python environment**
   `venv`
2. Create/Edit **launch script** for the Jupyter kernel
   `kernel.sh`
3. Create/Edit Jupyter **kernel configuration**
   `kernel.json`

https://gitlab.version.fz-juelich.de/jupyter4jsc/j4j_notebooks/-/blob/master/001-Jupyter/Create_JupyterKernel_general.ipynb

JUPYTER KERNEL

2. Create/Edit launch script for the Jupyter kernel (1)



1. **Create launch script, which loads your Python virtual environment and starts the ipykernel process inside:**

```
(<venv_name>) Lnode:> touch ${VIRTUAL_ENV}/kernel.sh
```

2. **Make launch script executable**

```
(<venv_name>) Lnode:> chmod +x ${VIRTUAL_ENV}/kernel.sh
```

3. **Edit the launch script for your new Jupyter kernel**

```
(<venv_name>) Lnode:> vi ${VIRTUAL_ENV}/kernel.sh
```

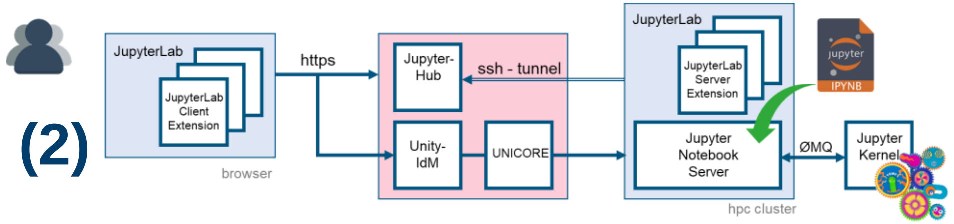
Building your own Jupyter kernel is a three step process

1. Create/Pimp new **virtual Python environment**
   `venv`
2. Create/Edit **launch script** for the Jupyter kernel
   `kernel.sh`
3. Create/Edit Jupyter **kernel configuration**
   `kernel.json`

https://gitlab.version.fz-juelich.de/jupyter4jsc/j4j_notebooks/-/blob/master/001-Jupyter/Create_JupyterKernel_general.ipynb

JUPYTER KERNEL

2. Create/Edit launch script for the Jupyter kernel (2)




```
#!/bin/bash

# Load required modules
module purge
module load Stages/2024
module load GCC
module load Python

# Load extra modules you need for your kernel
#module load <module you need>

# Activate your Python virtual environment
source <your_path>/<venv_name>/bin/activate

# Ensure python packages installed in the virtual environment are always prefered
export PYTHONPATH=${VIRTUAL_ENV}/lib/python3.11/site-packages:${PYTHONPATH}

exec python -m ipykernel $@
```

Building your own Jupyter kernel is a three step process

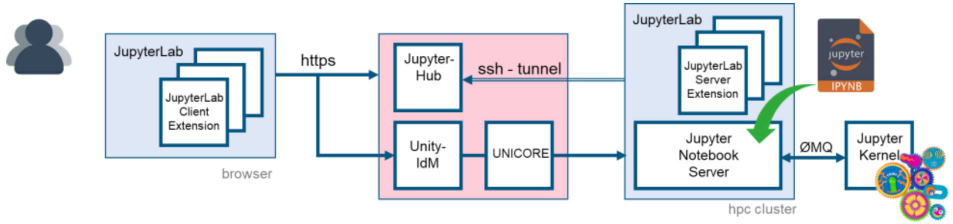
1. Create/Pimp new **virtual Python environment**
   `venv`
2. Create/Edit **launch script** for the Jupyter kernel
   `kernel.sh`
3. Create/Edit Jupyter **kernel configuration**
   `kernel.json`

https://gitlab.version.fz-juelich.de/jupyter4jsc/j4j_notebooks/-/blob/master/001-Jupyter/Create_JupyterKernel_general.ipynb

JÜLICH Forschungszentrum

JUPYTER KERNEL

3. Create/Edit Jupyter kernel configuration (1)



1. **Create your Jupyter kernel configuration files**

```
(<venv_name>) Lnode:>
python -m ipykernel install --user --name=<my-kernel-name>
```

2. **Update your kernel file to use the lauch script**

```
(<venv_name>) Lnode:>
vi ~/.local/share/jupyter/kernels/<my-kernel-name>/kernel.json
{
 "argv": [
  "<your_path>/<venv_name>/kernel.sh",
  "-m",
  "ipykernel_launcher",
  "-f",
  "{connection_file}"
 ],
 "display_name": "<my-kernel-name>",
 "language": "python"
}
```

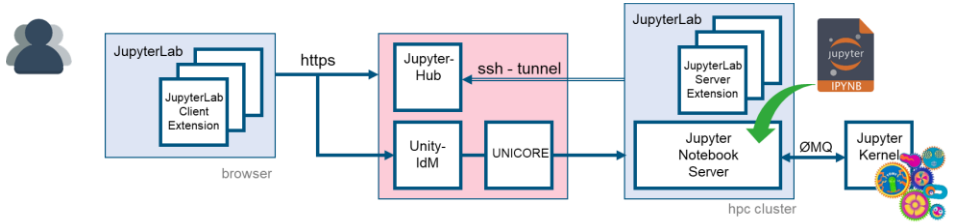
Building your own Jupyter kernel is a three step process

1. Create/Pimp new **virtual Python environment**
   `venv`
2. Create/Edit **launch script** for the Jupyter kernel
   `kernel.sh`
3. Create/Edit Jupyter **kernel configuration**
   `kernel.json`

https://gitlab.version.fz-juelich.de/jupyter4jsc/j4j_notebooks/-/blob/master/001-Jupyter/Create_JupyterKernel_general.ipynb

JUPYTER KERNEL

Run your Jupyter kernel configuration

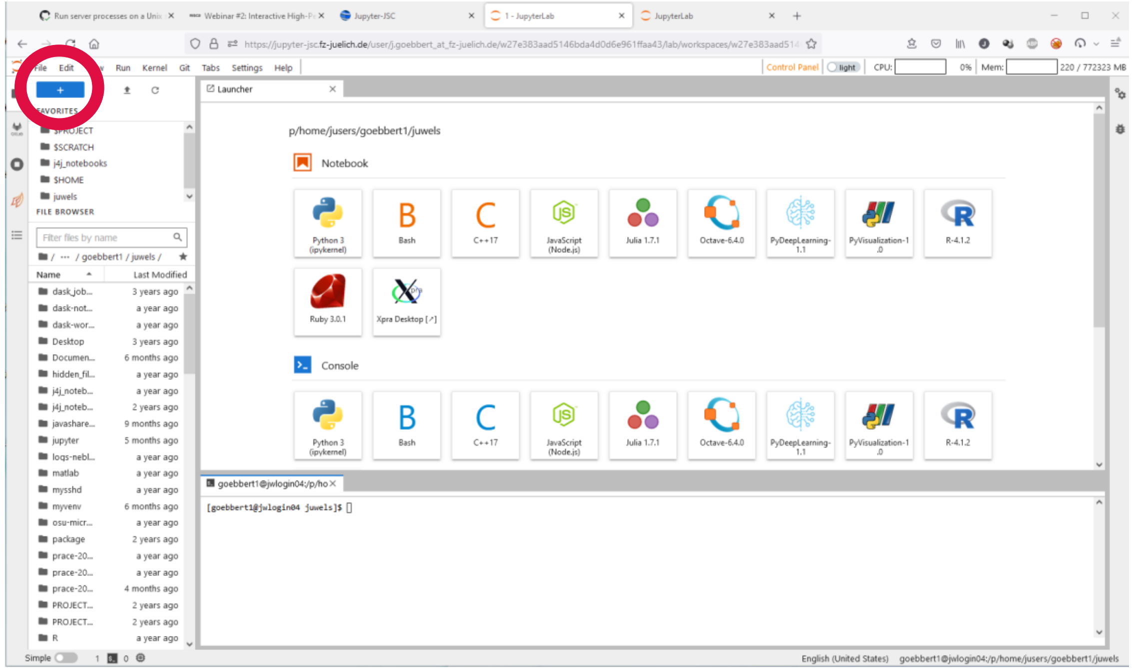


Run your Jupyter Kernel

1. https://jupyter-jsc.fz-juelich.de
2. Choose system where your Jupyter kernel is installed in `~/.local/share/jupyter/kernels`
3. Select your kernel in the launch pad or click the kernel name.

One of the many alternatives: Conda

Base your Jupyter Kernel on a Conda environment.
(check *3-create_JupyterKernel_conda.ipynb*)



Jupyter kernel are **NOT limited** to Python at all!
The kernel-endpoint just needs to talk the Jupyter's kernel protocol (in general over ZeroMQ).
E.g.

- IRkernel for R (https://github.com/IRkernel/IRkernel)
- IJulia.jl (https://github.com/JuliaLang/IJulia.jl)

JÜLICH Forschungszentrum

JUPYTER KERNEL

Shortcut! – Do not use this approach – Just for educational purpose

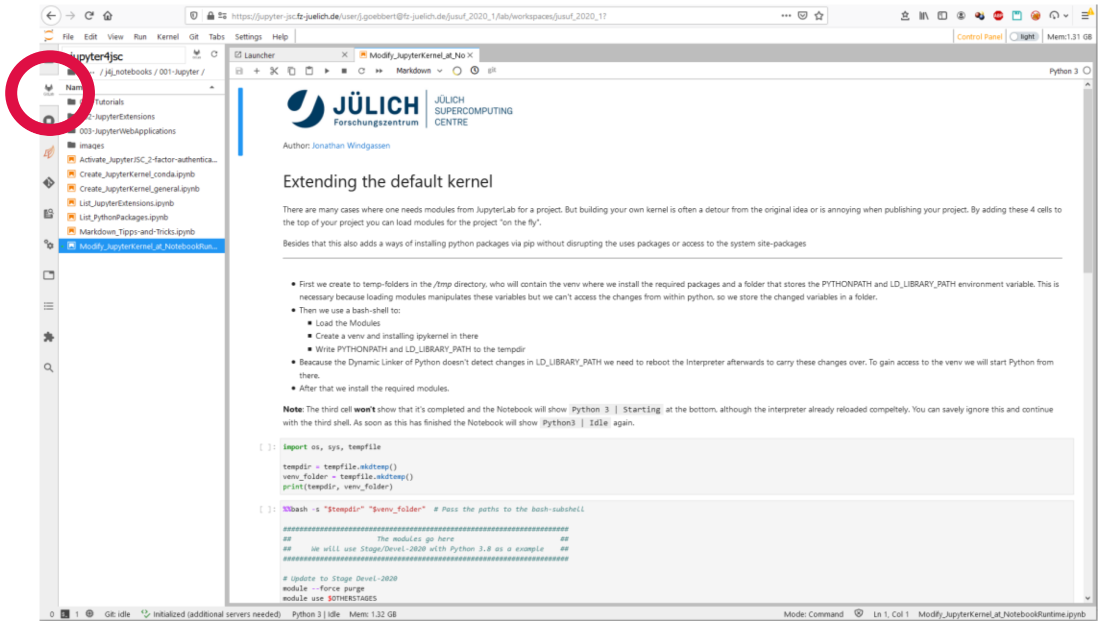
You do NOT want to build your own kernel,
every time you QUICKLY need a package or module.

Hack No. 1:

```
os.execve(f"{venv_folder}/bin/python", args, env)
```

1. **Create** a Python virtual environment at any location.
2. **WITHIN** the notebook
   - restart the kernel´s python interpreter
   - of that Python virtual environment
   - with the correct environment variables set.

Can stop the communication of the running ipykernel with the Jupyter server which will stop the kernel.



Hack No. 2:

```
import sys
sys.path.append('/home/.local/lib/python3.11/site-packages')
```

Dangerous: You easily can mess up with version requirements of Python packages installed at other places.

JÜLICH
Forschungszentrum