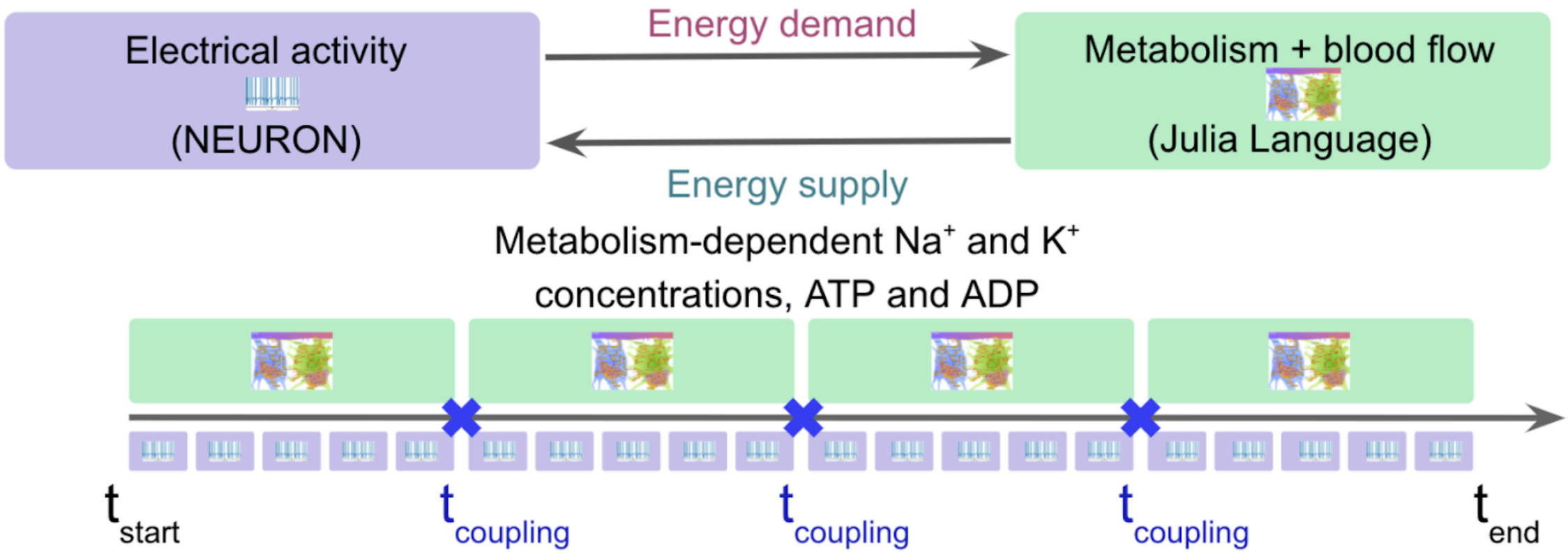
Multiscale Concept

V, synaptic glutamate and GABA,
Na^+ and K^+ concentrations and currents,
Ca^{2+} concentrations, ATP and ADP



MULTISCALE RUN SOLVER

EPFL Blue Brain Project

Neurodamus (NEURON)

Neurons

Neuronal activity

Astrocytes

Ca^{++}

Vasoactive substances

STEPS

Extracellular space

Track concentration of ions

AstroVascPy

Vasculature

Modulation of vessels' diameter

Blood Flow alteration

Metabolism (Julia)

Concentrations of metabolites and ions

NEURON

Backend for the simulation of neurons:

- The good-old de-facto software for detailed simulations of neurons
- Relatively low-level API
    - Create somas, sections, individual synapses with its properties
    - Networks of neurons was a step further (and we use thousands of them)
- Highly customisable (via .mod files, translated to C)
- Has it's own scripting language - HOC. Now also features Python

Neurodamus

Provide a high-level tool to setup large networks of neurons:

- Read main config from file (BlueConfig, sonata_config.json)
- Instantiate neurons (nodes) from designated files (mvd[2,3], sonata)
    - Including morphologies (in ascii or hdf5)
- Instantiate synapses from our connectivity files (nrn.h5, syn2, sonata edges)
- Define additional behaviour of the connections
    - spontaneous firing rate, weights, connection params
    - Setup Reporting
    - …
    - **Orchestrator of Multiscale Run Project**

For more details on the circuit, check:

1. multiscale_run_dimage/notebooks/multiscale_run_rat_sscx_S1HL_V7/rat_sscx_S1HL_V7 (circuit)
2. Circuit configuration files, i.e. circuit_config.json, node_sets.json, simulation_config.json [found in multiscale_run_dimage/notebooks/multiscale_run_rat_sscx_S1HL_V7]

Sonata guide

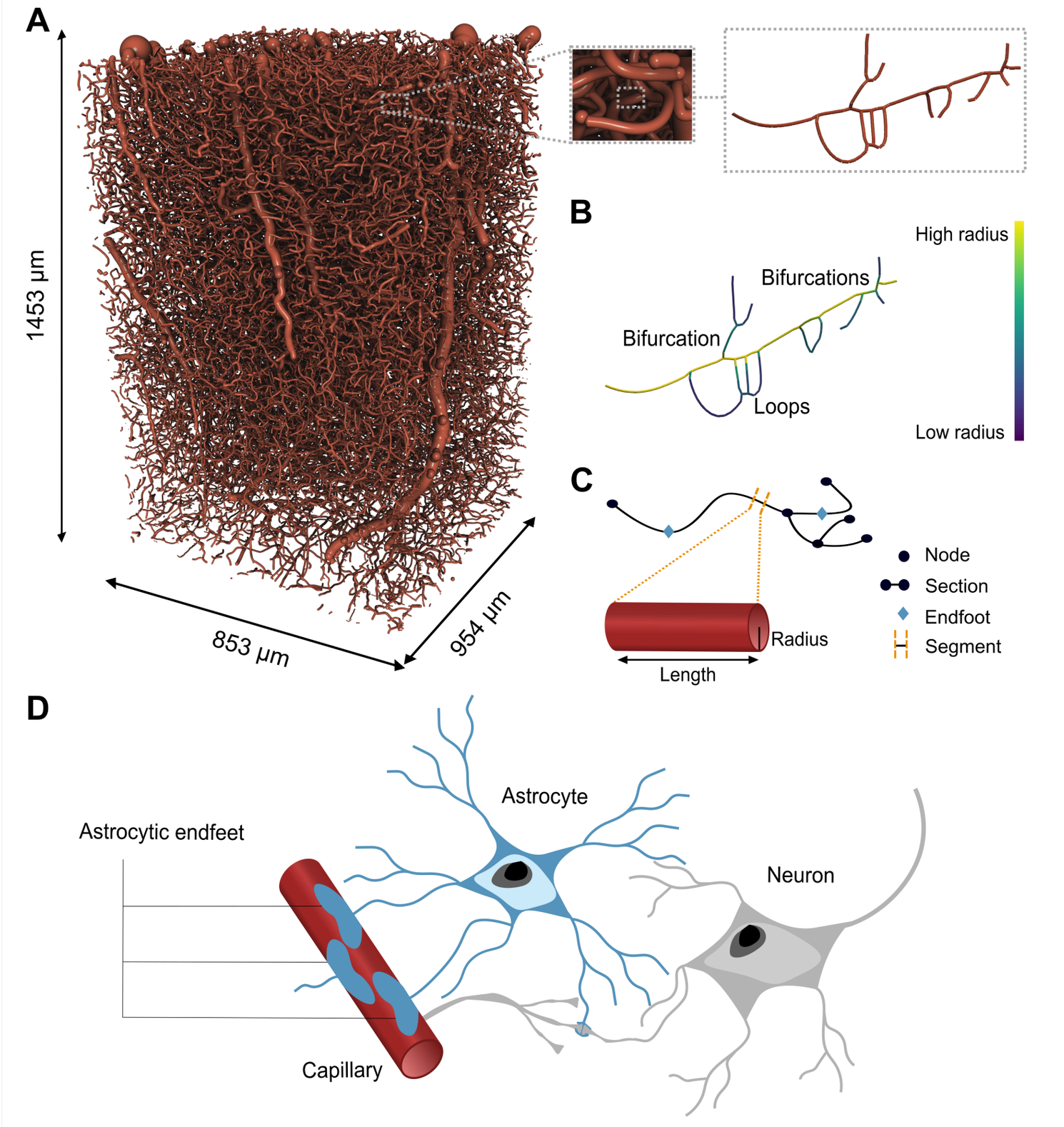
AstroVascPy

The Blood Flow model is based on:

- A coupling between the rat cerebral vasculature (undirected graph), and a circuit of synthesized astrocytic morphologies, where every endfoot is linked to a graph edge.

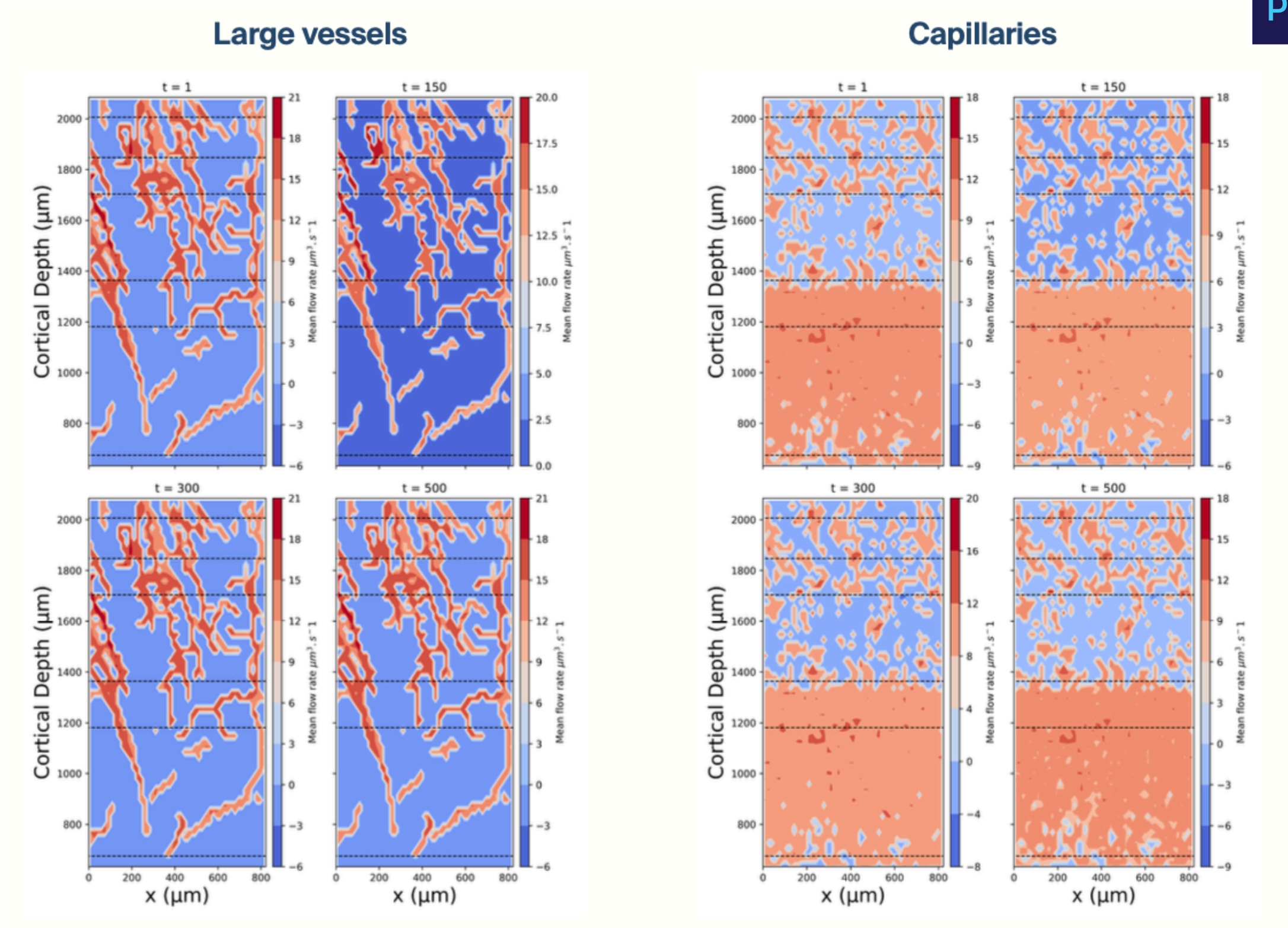
- A fluid dynamics model for computing the flow on each vessel of the vasculature:
  we solve the linearized Navier-Stokes equations.

- Radius dynamics driven by endfeet activity
  (obtained from *NEURON/Neurodamus* or from a stochastic simulation -standalone use-)



Results: Flow evolution

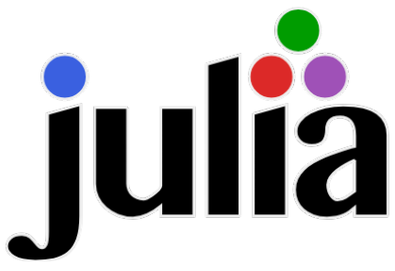
EPFL Blue Brain Project



Metabolism

- Written in Julia Scientific Computing Language
- Relies on SciML - DifferentialEquations.jl
- Python interface calling Julia code to interface it with the other simulators (STEPS, NEURON, and AstroVascPy)

```
prob = de.ODEProblem(self.model, self.vm, self.tspan_m, list(param))
try:
    sol = de.solve(
        prob,
        de.Rosenbrock23(autodiff=False),
        reltol=1e-8,
        abstol=1e-8,
        saveat=1,
        maxiters=1e6,
    )
```



[SciML: Differentiable Modeling and Simulation Combined with Machine Learning]

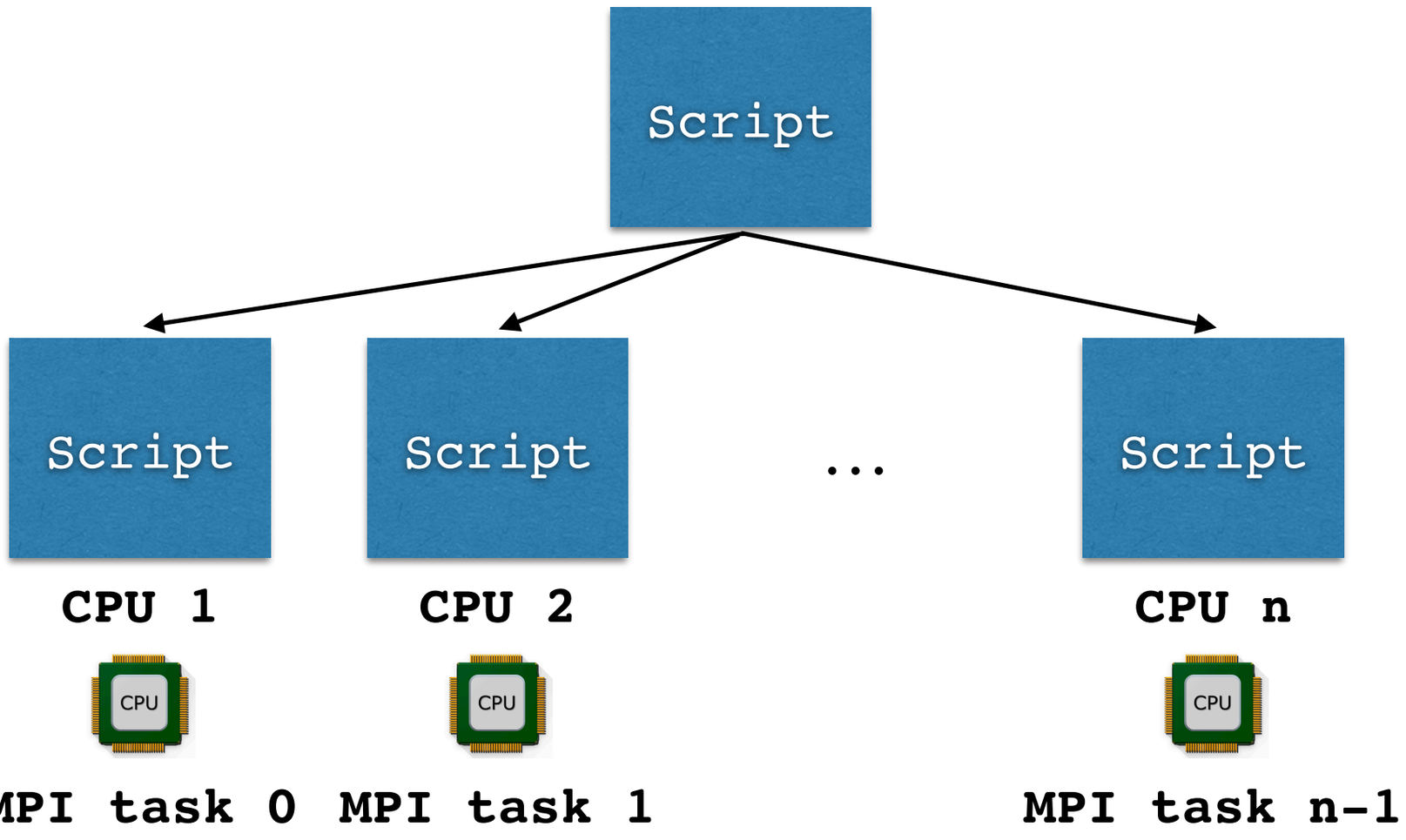
Any biologically interesting system involves a huge amount of neurons, astrocytes & blood vessels

These systems cannot be simulated in a personal computer but require a supercomputer

Distributed computing

Parallel Simulations: Message Passing Interface (MPI)

C++ (MPI library) / Python (mpi4py)



Each MPI process/task operates on the same code but on different data. If/Else If with MPI task ID.

Every MPI process has its own memory space. Processes are ~completely~ isolated!

MPI tutorials: (1), (2), (3)

Parallel Simulations: Message Passing Interface (MPI)

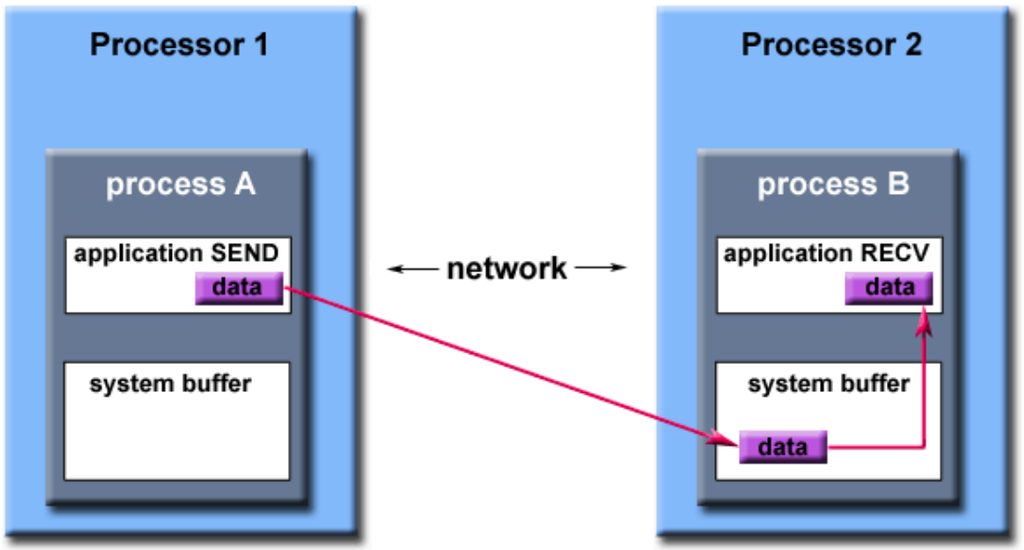
Usual command:

```
mpirun -n 2048 python script.py args
   mpirun -n 4096 script.exe args
```

MPI tutorials: (1), (2), (3)

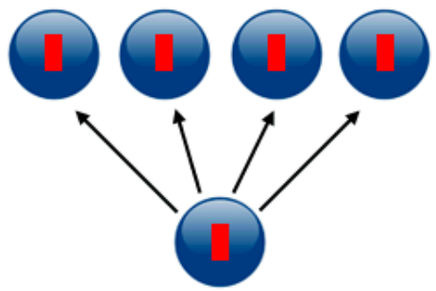
Parallel Simulations: Message Passing Interface (MPI)

Point-to-Point Operations

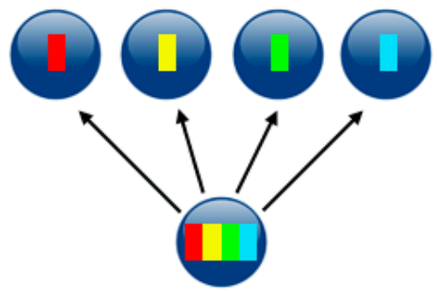


Every MPI process has its own memory space. Processes are ~completely~ isolated!

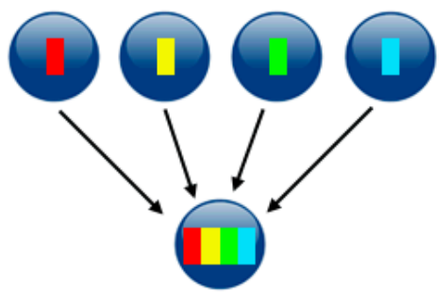
Collective Operations



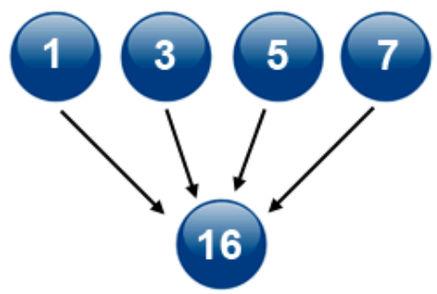
broadcast



scatter

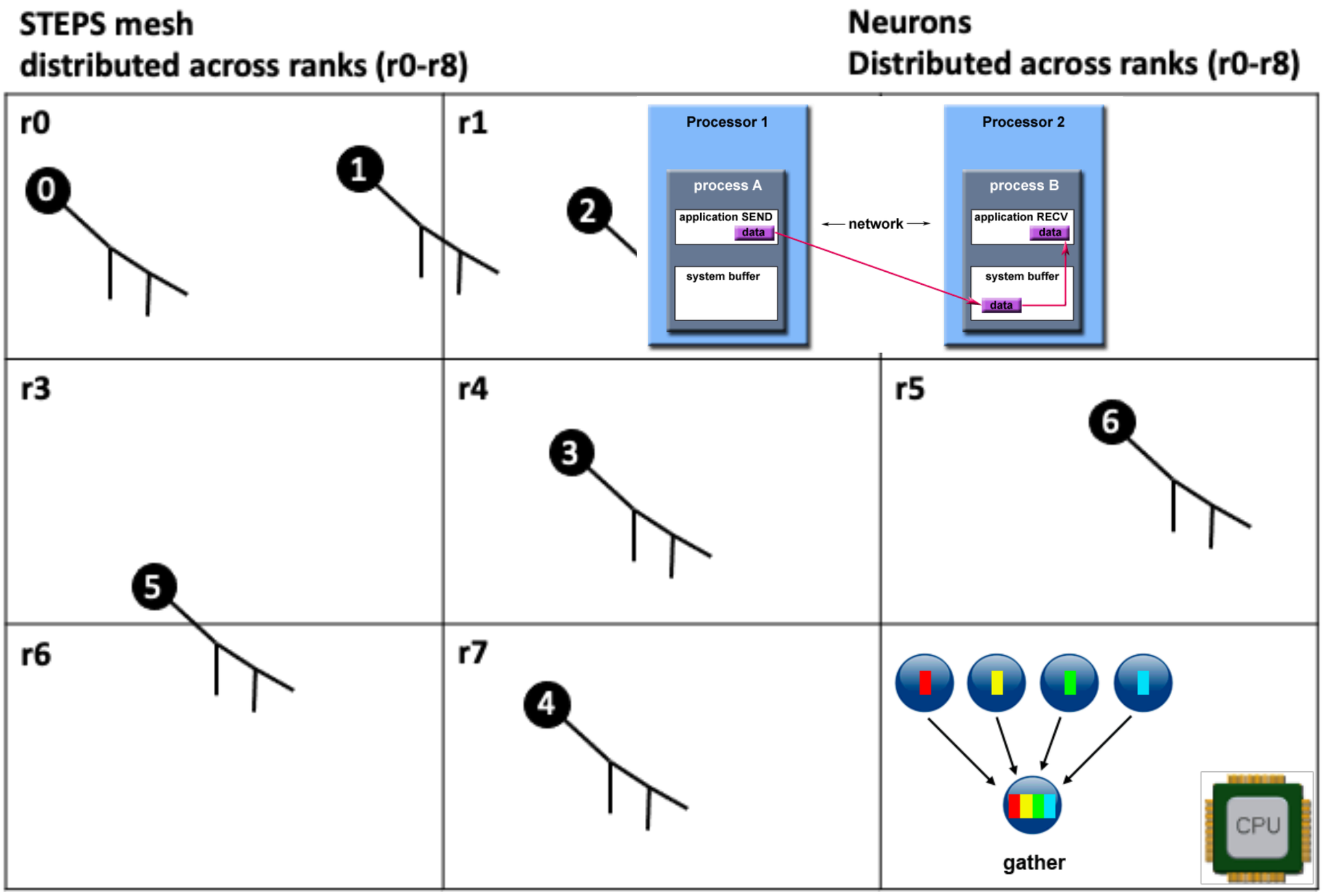


gather



reduction

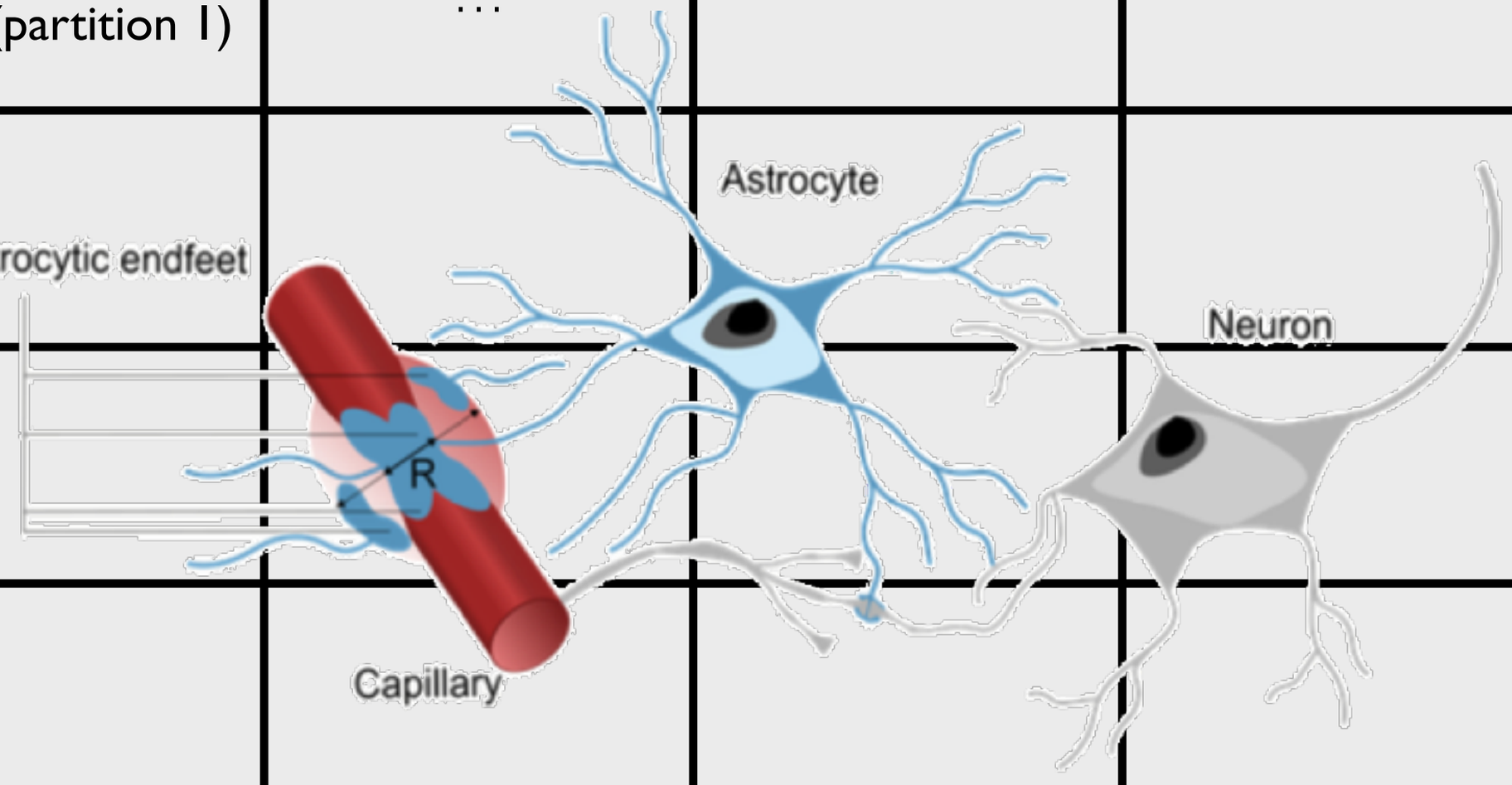
MULTISCALE RUN SOLVER



Problem of independent solvers: Objects in the same CPU core may not spatially overlap

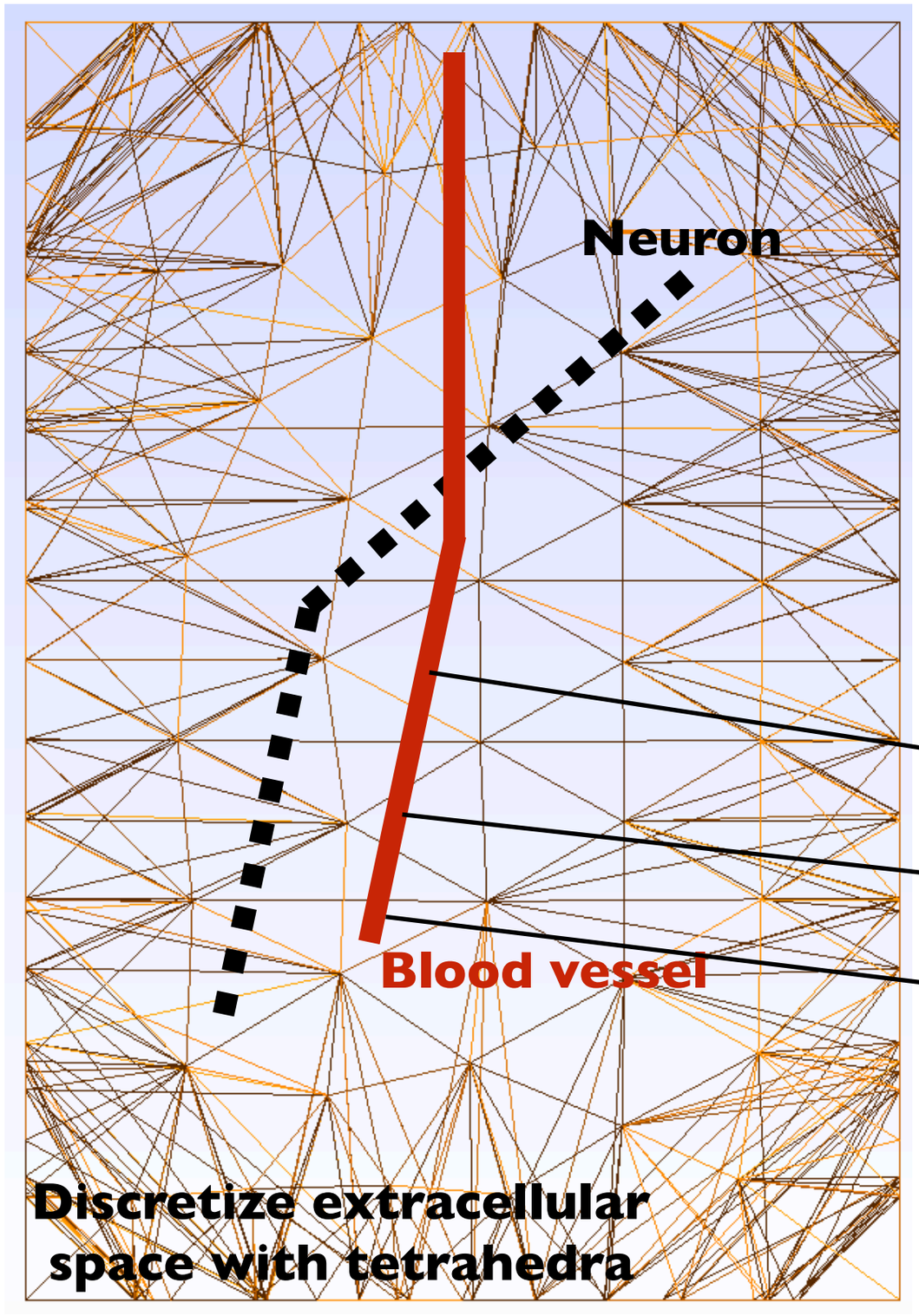
MULTISCALE RUN SOLVER

STEPS distributed mesh (partition 1)	…		
Which entities are in this subdomain (?)		…	STEPS distributed mesh (partition n)



Associate Tetrahedra/ Neurons/ Vasculature with the help of STEPS intersect method

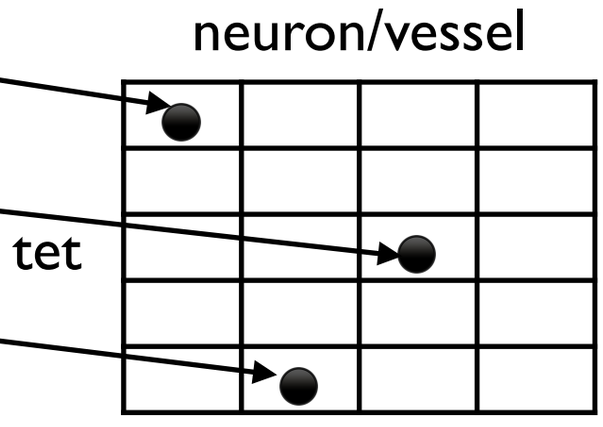
MULTISCALE RUN SOLVER



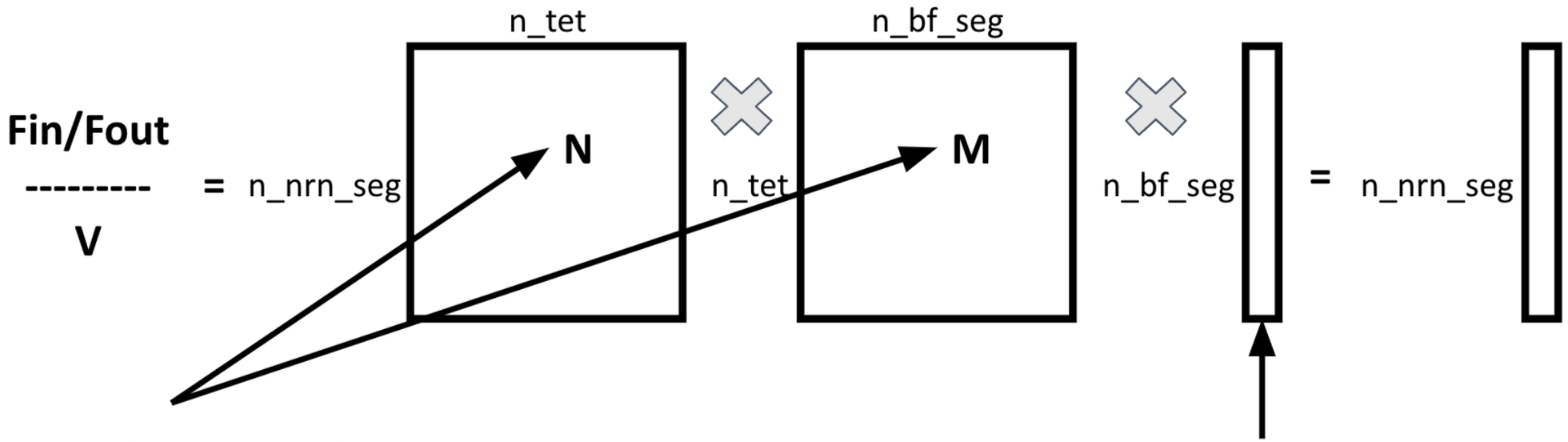
STEPS intersect method returns a list of tuples per straight segment:
[(tet_id, ratio), …]

- Associate Tetrahedra with Neurons
- Associate Tetrahedra with Blood Vessels

mapping between Tets & Neurons/Vessels
sparse matrices



MULTISCALE RUN SOLVER



Sparse matrices that associate neurons/vessels with tets

Additionally we have the fractions of neurons/vessels crossing the corresponding tets

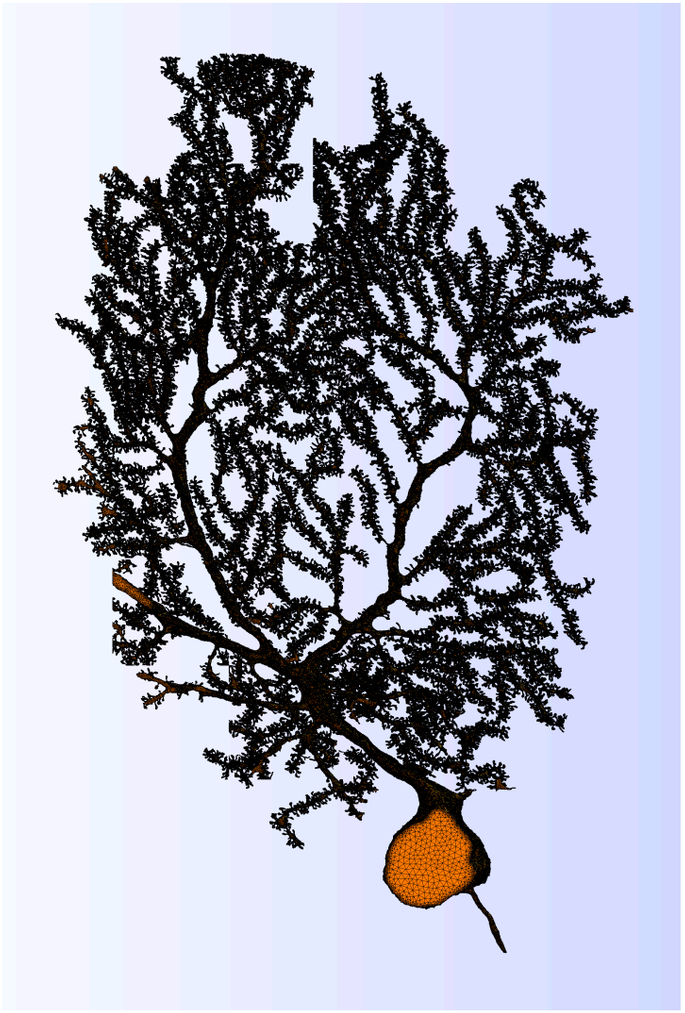
graph.edge_properties["flow"]
---
graph.edge_properties["volume"]

Once we build the sparse matrices (just at the beginning + possibility to store them in the disk for future runs), we can transform quantities from one network to the other

STEPS
Latest HPC-proof version

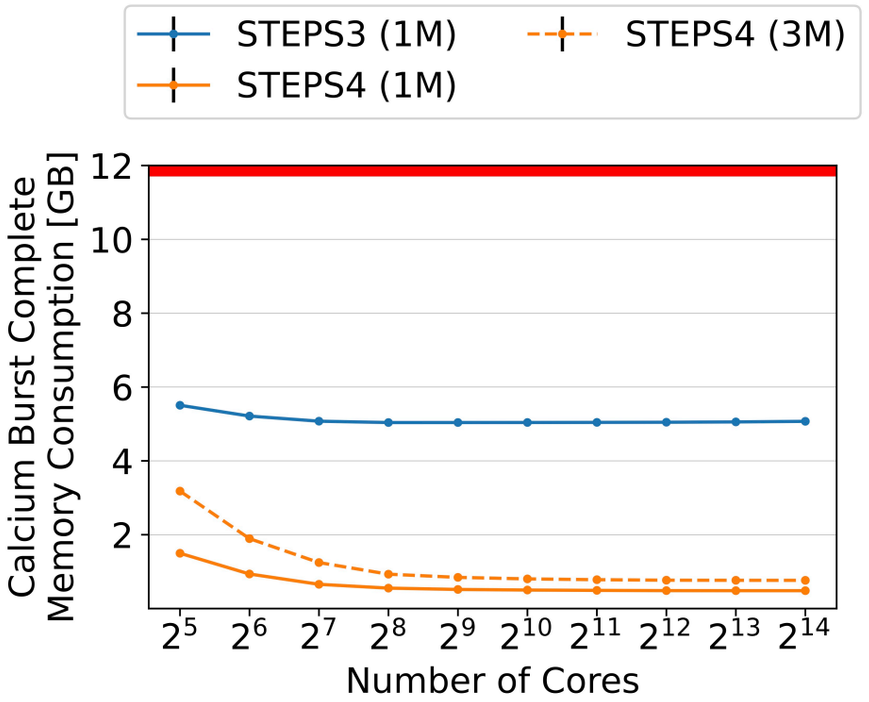
- **STEPS 3** (TetOpSplit) vs **STEPS 4** (DistTetOpSplit)
  - **STEPS 3** parallelizes the computation but the full mesh is loaded across processors
  - **STEPS 4** parallelizes both computations and mesh, i.e. mesh is partitioned and shared across CPU cores
- For **STEPS 4** we need a preprocessing step (mesh partitioning)
  - Gmsh is a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities.

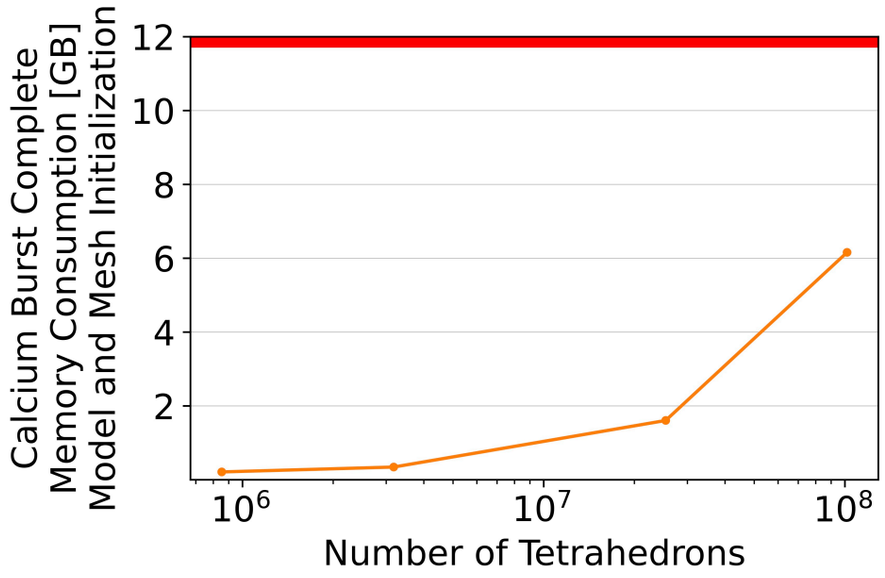
Parallel Simulations
STEPS 4.0

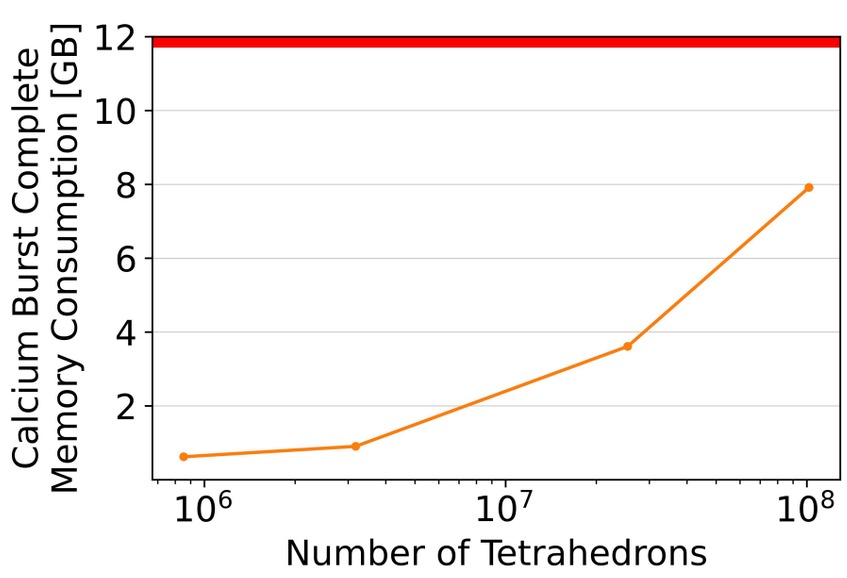


Realistic Purkinje cell reconstruction (**~17 million tetrahedra**)

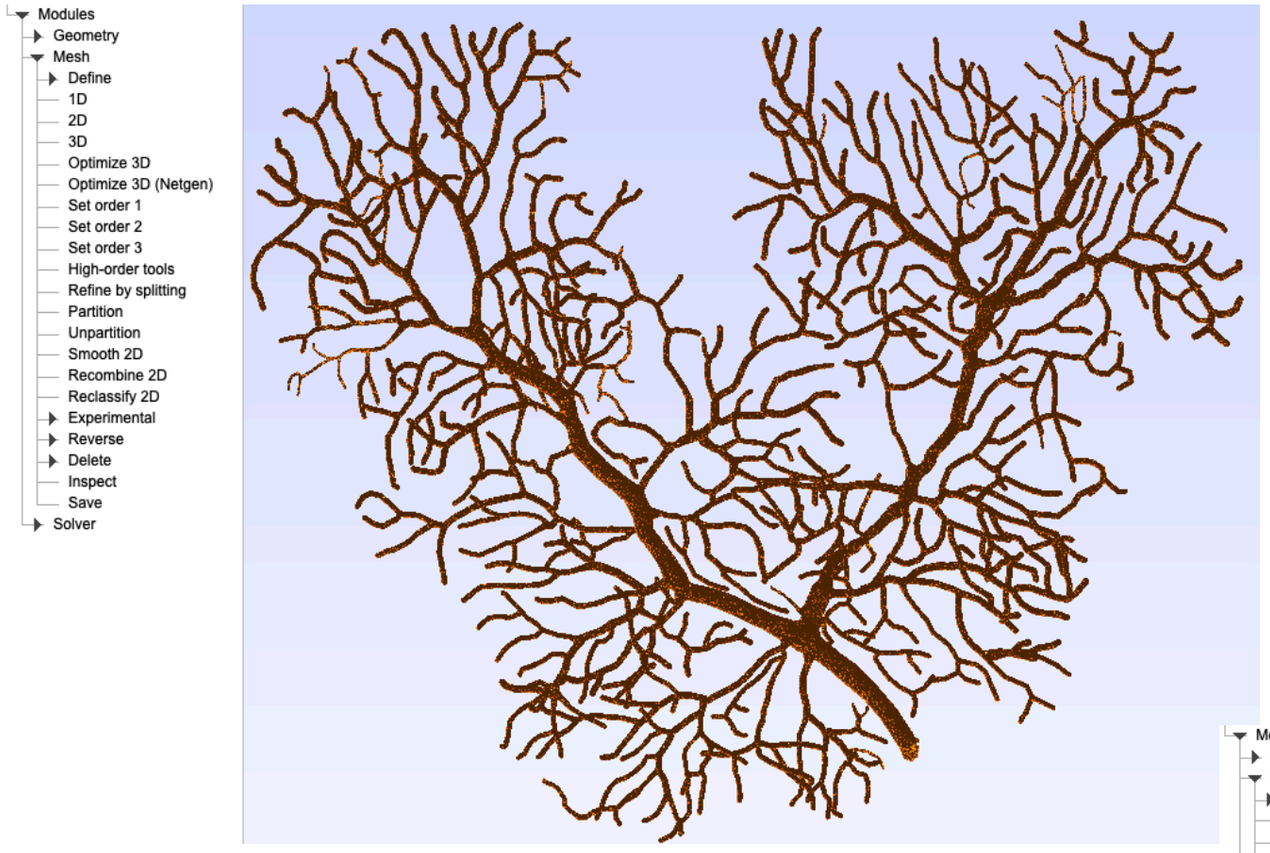
Study of Calcium concentration in the cell

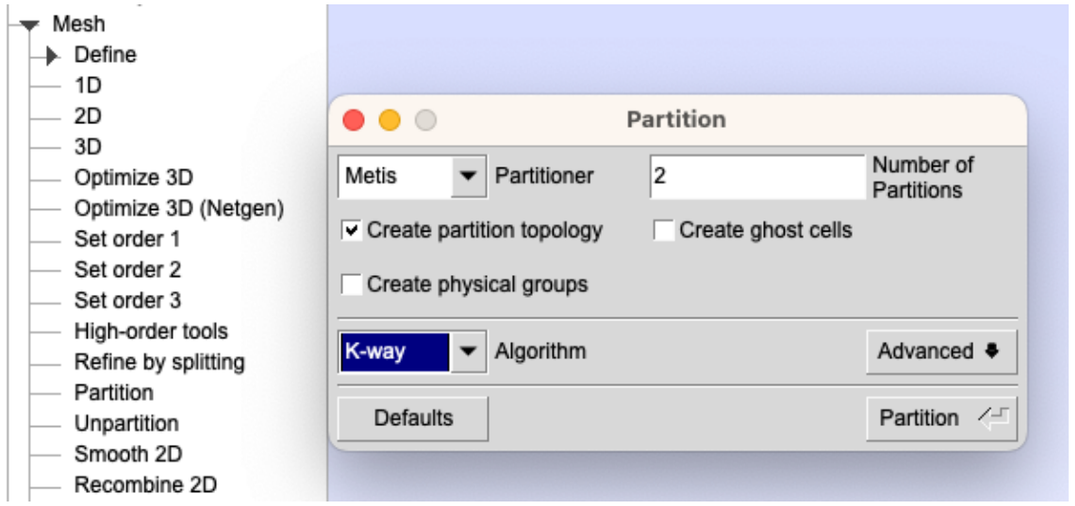


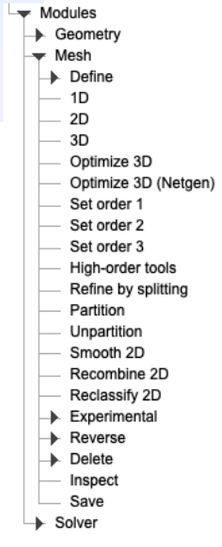


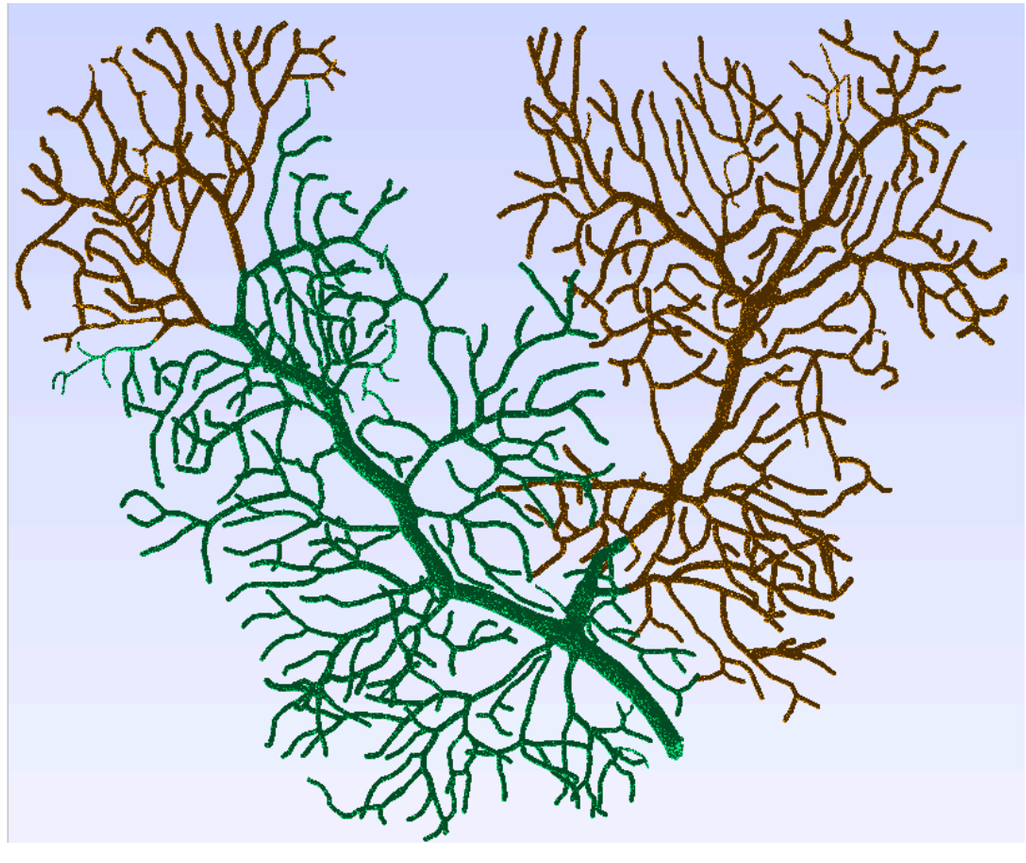


Gmsh





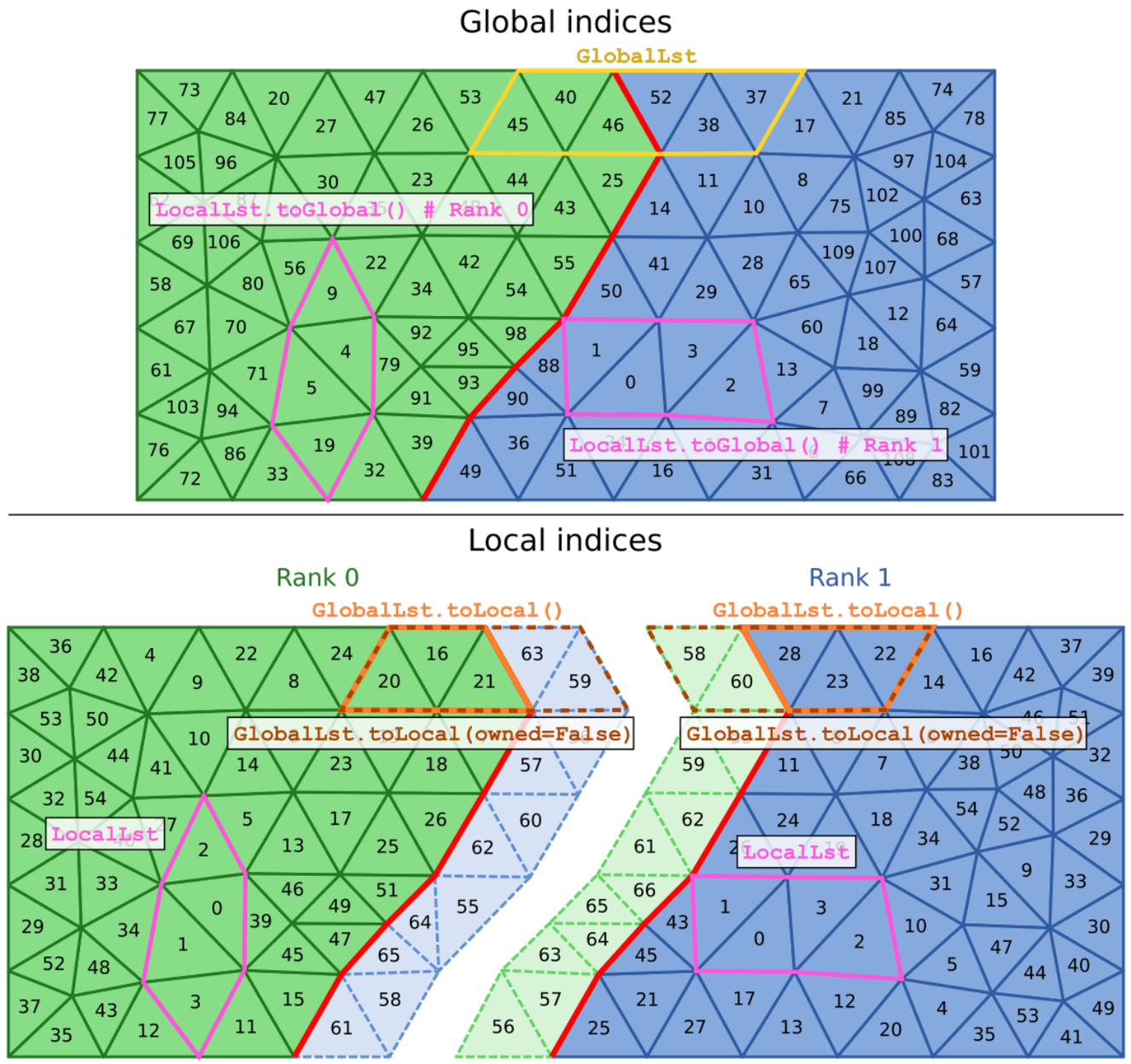




Complete instructions

Local/Global Elements

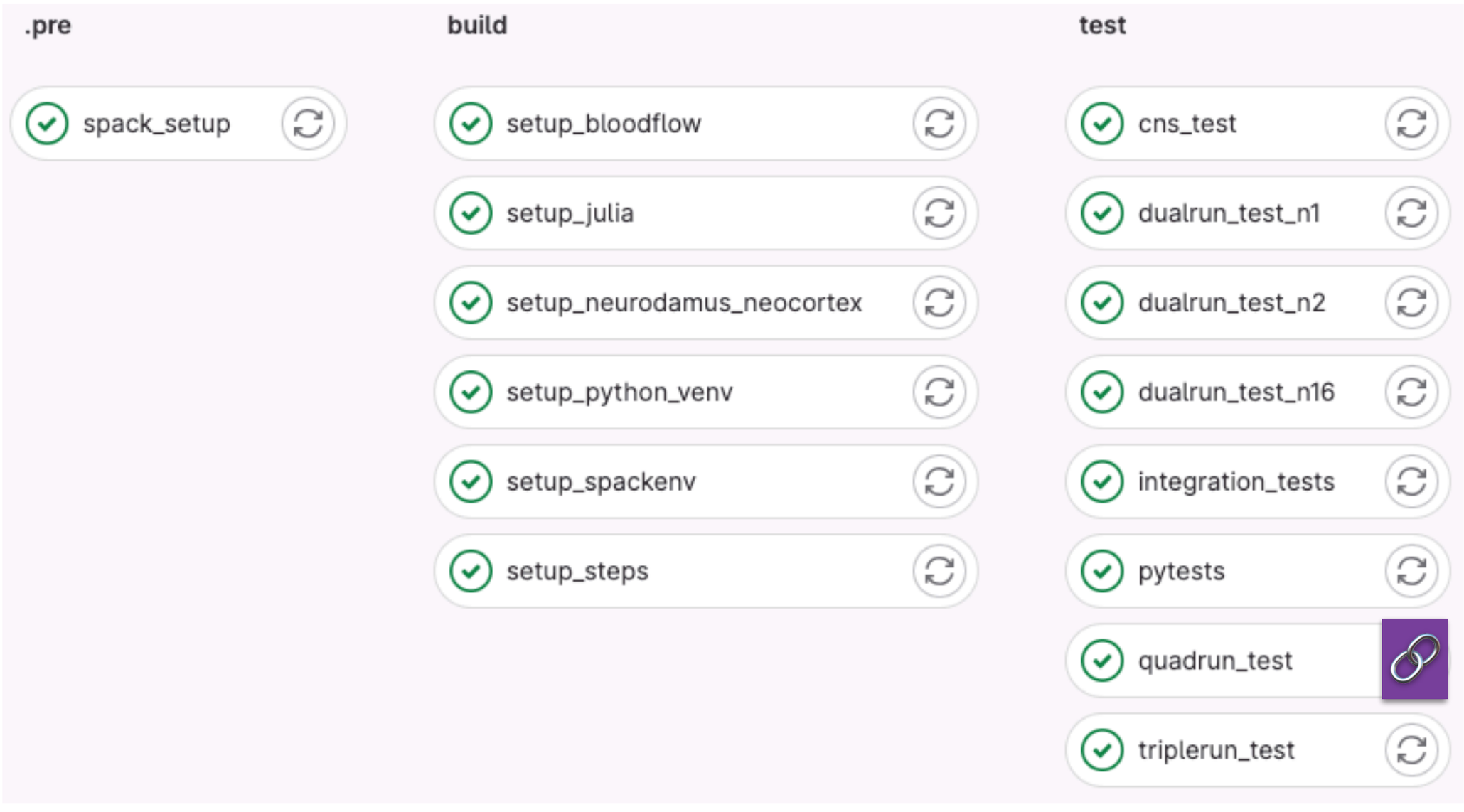
Each MPI process loads the part of the mesh that corresponds to its partition. The elements (tetrahedrons, triangles, vertices) can then be referred to by two types of indices: a **global index** that uniquely identifies the element across all MPI ranks; and a **local index** that is specific to each MPI rank.



MULTISCALE RUN SOLVER

EPFL Blue Brain Project

CI/CD: Automated Validation Monitoring



Docker Image

https://github.com/kotsaloscv/multiscale_run_dimage

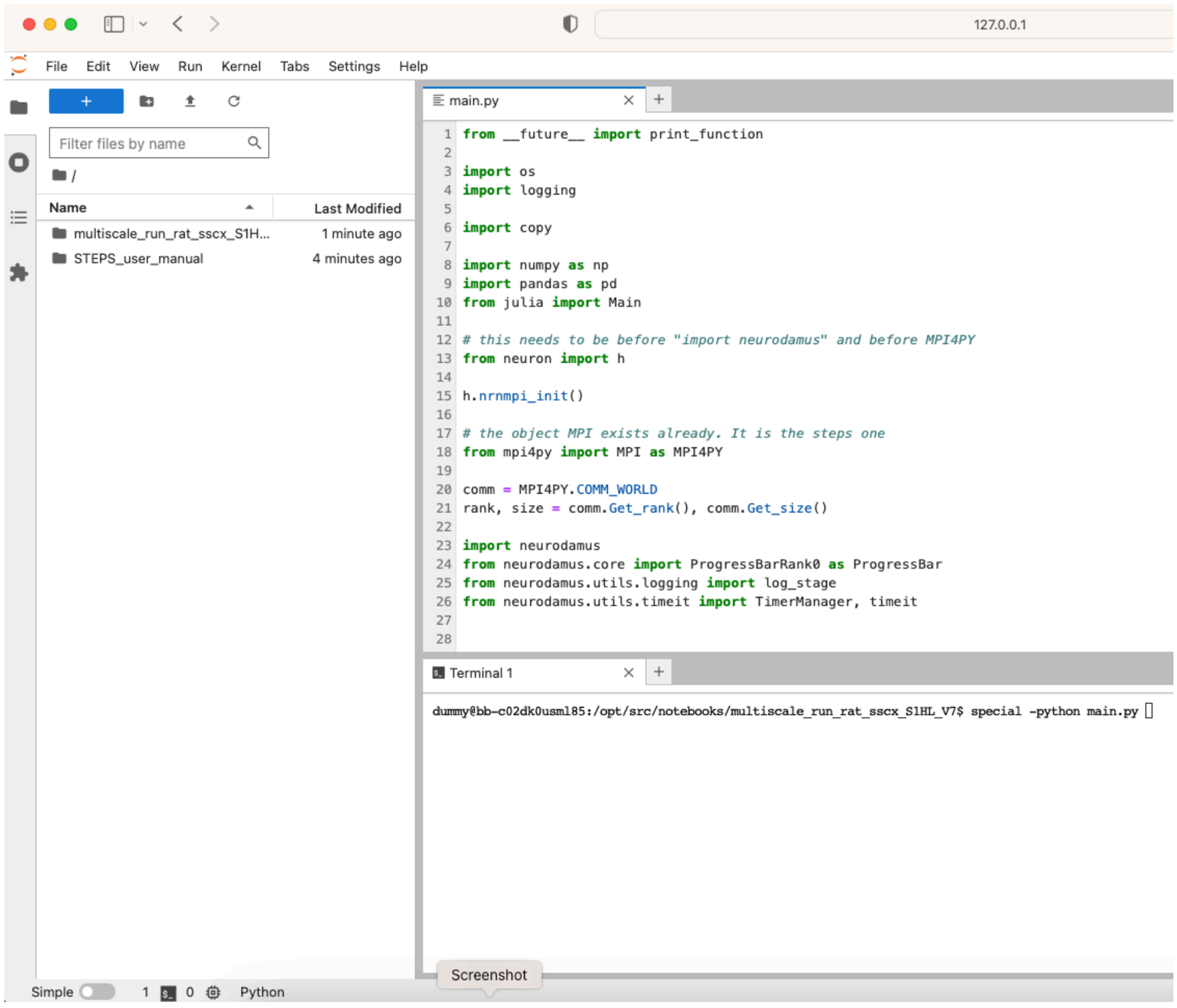
The docker image packages all the previously mentioned solvers.

Once deployed, just import the needed modules, write your script, and execute it.
Everything is baked in the environment.

Useful docker commands:

- docker build Dockerfile [aka recipe]
- docker pull kotsaloscv/multiscale_run:v0.0.0
- docker images
- docker ps -a
- docker rmi -f IMAGE_ID
- docker rm -f CONTAINER_ID
- docker run --rm -it kotsaloscv/multiscale_run:v0.0.0 bash
- docker exec -it NAME_OF_RUNNING_CONTAINER bash

Docker Image
Jupyter Lab



Docker Image
Jupyter Lab

docker-compose up [old version]
or
docker compose up [new version]

to deploy the Jupyter Lab

Orchestrator: **docker-compose.yml** [YAML file]

https://github.com/kotsaloscv/multiscale_run_dimage

Virtual Machines:
12 VMs & 30 users

1. **35.158.94.77** : user01 / user02
2. **3.70.127.45** : user03 / user04
3. **3.123.37.121** : user05 / user06
4. **18.184.166.80** : user07 / user08
5. **52.59.198.109** : user09 / user10
6. **18.159.196.130** : user11 / user12
7. **18.197.134.13** : user13 / user14 / user25
8. **3.77.233.89** : user15 / user16 / user26
9. **18.192.122.159** : user17 / user18 / user27
10. **3.122.95.19** : user19 / user20 / user28
11. **3.71.175.108** : user21 / user22 / user29
12. **3.75.216.204** : user23 / user24 / user30

VMs access & command line

Access it: ssh user[01-30]@ip_address
Example: ssh user01@ip_address

Password: CNSLeipzig[01-30]
Example: CNSLeipzig01 for user01

Home folder: /efs/home/user[01-30]
EFS stands for Amazon Elastic File System

1. Access it **[ssh]**
2. git clone https://github.com/kotsaloscv/multiscale_run_dimage.git **in your home folder**
3. cd multiscale_run_dimage
4. echo -e "DUID=$(($(id -u)+0))\nDGID=$(id -g)\nHOST=$(hostname)" > .env
5. docker run --rm -it kotsaloscv/multiscale_run:v0.0.0 bash
6. cd /opt/src
7. gdown 1ZgdF4R2UgL_s8TK4lnb8qnSmhxex81gJ -O - --quiet | tar zxf -
8. cd multiscale_run_rat_sscx_S1HL_V7
9. special -python main.py

VMs
Jupyter Lab

Access it: ssh user[01-30]@ip_address
Example: ssh user30@**3.75.216.204**

Password: CNSLeipzig[01-30]
Example: CNSLeipzig30 for user30

Home folder: /efs/home/user[01-30]
EFS stands for Amazon Elastic File System

1. Access it **[ssh]**
2. git clone https://github.com/kotsaloscv/multiscale_run_dimage.git **in your home folder**
3. cd multiscale_run_dimage
4. echo -e "DUID=$(($(id -u)+0))\nDGID=$(id -g)\nHOST=$(hostname)" > .env
5. docker compose up
6. Open a new terminal and type: ssh -N -f -L 8888:localhost:8888 user[01-30]@ip_address
7. Open the http address generated by the docker compose up
8. In the Jupyter Lab open a terminal
9. gdown 1ZgdF4R2UgL_s8TK4lnb8qnSmhxex81gJ -O - --quiet | tar zxf -
10. Ready to access and execute the code