5TH HBP STUDENTS WORKSHOP

A morphologically-detailed neuronal network simulation library for heterogeneous high performance computing architectures

2ND FEBRUARY 2021 I BRENT HUISMAN

Human Brain Project

JÜLICH Forschungszentrum

FET FLAGSHIPS

Co-funded by the European Union

CORE ARBOR TEAM

From different institutions

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

- Ben Cumming
- Stuart Yates
- Nora Abi Akar

JÜLICH
FORSCHUNGSZENTRUM

- Anne Küsters
- Thorsten Hater
- Brent Huisman

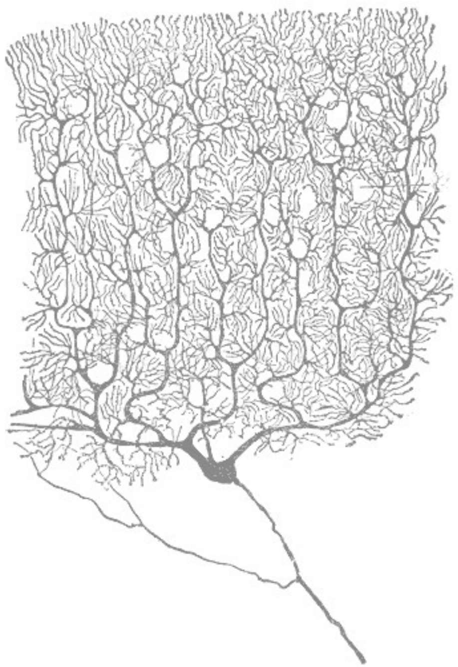
openly available @ github.com/arbor-sim/arbor

WHAT IS ARBOR?

A morphologically-detailed neuronal network simulation library for heterogeneous HPC architectures

A **library** for the simulation

- of **large networks** of morphologically-**detailed, spiking neurons**
- for all **HPC** systems in the HBP



Purkinje cell by Santiago Ramón y Cajal

Runs on multiple architectures

- **GPU** systems,
- **vectorized** multicore,
- Intel **AVX** and **laptops**

Modular design for **extensibility** to new computer architectures

WHY ARBOR?

To solve multi-compartment simulations with large networks on new HPC architectures

Problems and models that are challenging to explore with current software and systems, e.g.

- Near real-time multi-compartment simulations
- Large networks with long simulations, parameter search, statistical validation
- Field potential calculations of large networks with volume visualization

Adapting existing simulators to **new HPC architectures** is hard, e.g. for

- Highly parallel architectures such as Intel Xeon and Intel KNL
- Wider vector operations such as AVX, AVX2, AVX512
- Specialized accelerator hardware as GPUs

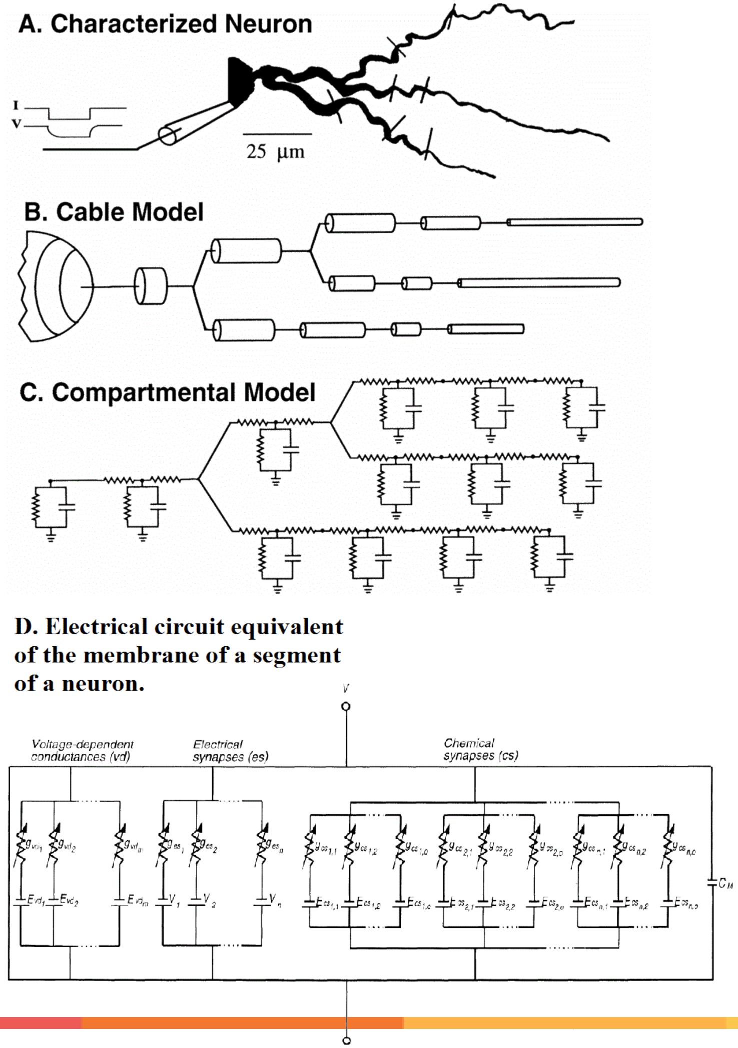
Source of picture: flaticon.com

QUESTIONS: PROGRAMMING EXPERIENCE?

- Languages
  - Python, C++, Matlab, ...
- Operating systems
  - MacOS, Linux, Windows
- Tools
  - Bash, Git, ...
- Hardware
  - GPU, HPC, ...

ARBOR'S NEURON MODEL

Arbor simulates networks of multi-compartment neurons



- **Neurons:** approximated by axonal delay, synaptic functions and a set of cables (for dendrites + soma) connected in a tree.
- **Cables:** characterized as 1D electrical compartments (of variable diameter) composed of ion channels, cable resistance and capacitance.
- Neurons represented as sparse, close-to-band matrices to be solved (e.g. by Hines solver) against known current states due to synaptic conductance.
- **Network** and spike exchange between neurons at synapses are represented by concatenations of matrices.

Source: Koch, *Methods in Neuronal Modeling: From Ions to Networks*

CABLE EQUATION

A cell is modelled as a branching, one-dimensional electrical system

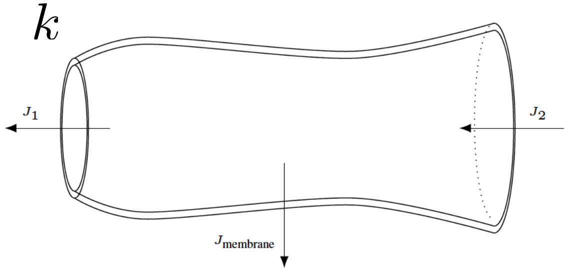
$$\frac{\partial}{\partial x}\left(\sigma \frac{\partial v}{\partial x}\right) = \left(c_m \cdot \frac{\partial v}{\partial t} + \sum_{\text{channels } k} g_k(\underline{s}_k(x,t))(v - e_k^{\text{rev}})\right) \cdot \frac{\partial S}{\partial x}$$
$$+ \sum_{\text{synapses k}} I_i^{\text{syn}}(\underline{s}_k^{\text{syn}}(t), v(x_k^{\text{syn}}))\, \delta x_k^{\text{syn}}$$
$$+ \sum_{\text{injections k}} I_k^{\text{inj}}(t)\, \delta x_k^{\text{inj}},$$

, where

$$\frac{d}{dt}\underline{s}_k(x,t) = f_k(\underline{s}_k, v(x,t)),$$
$$\frac{d}{dt}\underline{s}_k^{\text{syn}}(t) = f_k^{\text{syn}}(\underline{s}_k^{\text{syn}}, v(x_k^{\text{syn}}, t), t),$$

with

- Axial conductivity σ of the intracellular medium
- Membrane areal capacitance c_m, areal conductance g_k for an ion channel of ty k as a function of channel's state $\underline{s}_k$
- Corresponding reversal potential e_k^{ref}
- Membrane surface area $S(x)$ as a function of axial distance x
- Current I_k^{syn} produced by a synapse at position x_k^{syn} as a function of the synaptic state $\underline{s}_k^{\text{syn}}$ and local voltage
- Injected current $I_k^{\text{inj}}(t)$ at position x_k^{inj}



NUMERICAL MODEL

Cell state evolution is numerically solved with first order methods

- **Space discretization:** Vertex-centered 1D finite volume method using first-order approximation for axial current flux

$$c_i \frac{dV_i}{dt} = \sum_{j:\ X_j \cap X_i \neq \emptyset} \sigma_{i,j}(V_j - V_i) - \sum_{k:\ x_k^{\text{inj}} \in X_i} I_k^{\text{inj}}(t)$$

$$- \sum_{k:\ x_k^{\text{syn}} \in X_i} I_k^{\text{syn}}(\underline{s}_k^{\text{syn}}, V_i)$$

$$- \sum_{\text{channels } k} S_i \cdot g_k(\underline{s}_{k,i})(V_i - e_k^{\text{rev}}),$$

with

$$\frac{d\underline{s}_{k,i}}{dt} = f_k(\underline{s}_{k,i}, V_i),$$

$$\frac{d\underline{s}_k^{\text{syn}}}{dt} = f_k^{\text{syn}}(\underline{s}_k^{\text{syn}}, V_i, t),$$

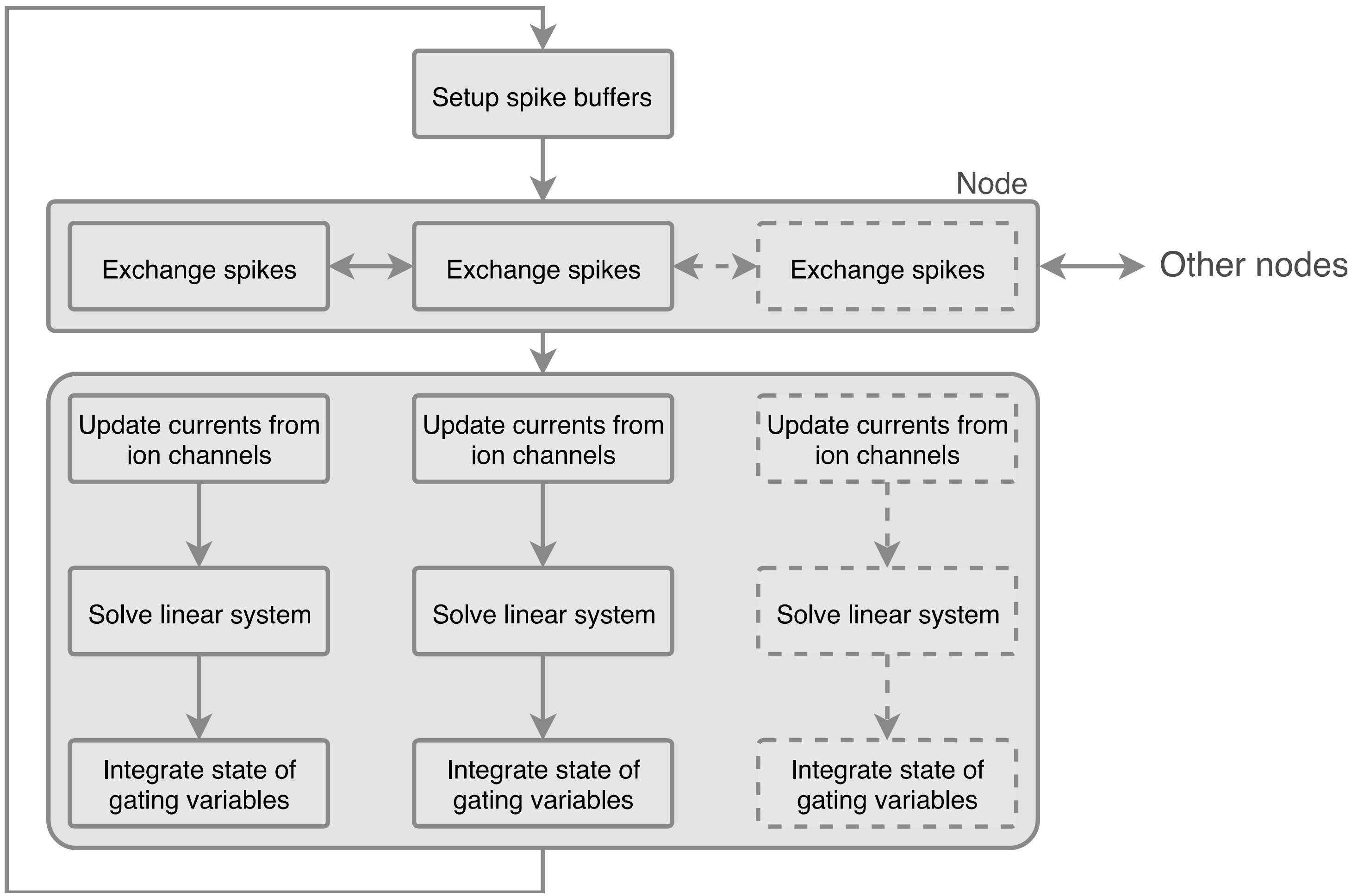
- Voltage and channel state **time evolution split:** Lie-Trotter
  - **Time discretization:** First-order implicit Euler integration

$$\frac{c_i}{\delta t} V_i' + \sum_j \sigma_{i,j} V_i' - \sum_j \sigma_{i,j} V_j' = -I_i^{\text{memb}} + \frac{c_i}{\delta t} V_i$$

  - **Channel state ODEs:** Integration with updated voltages depending on set of ODEs

CELL SIMULATION

Most time consuming parts on a CPU are updating currents and integrating gating variables



QUESTIONS: MODELLING EXPERIENCE?

- Math
  - (Systems of) (partial) differential equations
- Methods
  - Integration
  - Discretization
  - Solvers

ARBOR'S DESIGN MODEL

Describe the neuroscience first ...

Cells

- A "cell" **represents the smallest model** to be simulated
- A "cell" forms the **smallest unit of work** distributed across processes
- Types:
  - Specialized **leaky integrate-and-fire** cells
  - **Artificial spike** sources
  - **Multi-compartment** cells

Recipes

- A "recipe" **describes models** in a cell-oriented manner and supplies methods to
  - Map **global cell identifier** `gid` to cell type
  - Describe cells
  - List all **connections** from other cells that terminate on a cell
- Advantage: parallel **instantiation** of cell data

ARBOR’S DESIGN MODEL

… then translate it into execution.

Cell groups

- A “cell group” represents a **collection of cells of the same type** together with implementation of their simulation
- **Partitioning** into cell groups provided by decomposition
- A “simulation” manages **instantiation** of model and **scheduling** of spike exchange as well as **integration** for each cell group

Mechanisms

- In a recipe, mechanisms are specifications of **ion channel** and **synapse dynamics**
- Implementations of mechanisms:
  - Hand-coded for CPU/ GPU execution

    or
  - A translator (`modcc`) is used to compile a subset of NEURONs mechanism specification language NMODL to architecture-optimized vectorized `C++` or CUDA source

ARBOR’S DESIGN MODEL

… then translate it into execution.

Cell groups

- A “cell group” represents a **collection of cells of the same type** together with implementation of their simulation
- **Partitioning** into cell groups provided by decomposition
- A “simulation” manages **ins**… model and **schedulin**… …ge as well as **integr**… …roup

Me…

- In a recip… …specifications of **io**… …**pse dynamics** …s of mechanisms:
  - …-coded for CPU/ GPU execution

  or

  - A translator (`modcc`) is used to compile a subset of NEURONs mechanism specification language NMODL to architecture-optimized vectorized `C++` or CUDA source

Handled for you (mostly)

MODEL

Summary

- Arbor models:
  - Multicompartment neurons using a cable model transformed into a sparse matrix
  - Neurons characterized by axonal delays, synaptic functions and cables connected in a tree
  - Spike exchanges are global across computer nodes, functionally concatenating matrices
- Numerical solutions are discretized in time and space, and channel states are discretized ODEs
- Accelerator (GPU) optimization is focused on updating currents and integrating gating variables
- Models are composed of:
  - Cells representing the small unit of computation (LIF, Artificial sources, Multicompartment cells)
  - Recipes representing a parallelizable set of neuron construction and connections
  - Cell groups computed together on the GPU or CPU
  - Mechanism representing ion channel and synapse dynamics

QUESTIONS

EXERCISES

- Construct a small ring network
- 1: make a cell!
- 2: make the network
- 3: make the simulation
- 4: show some results

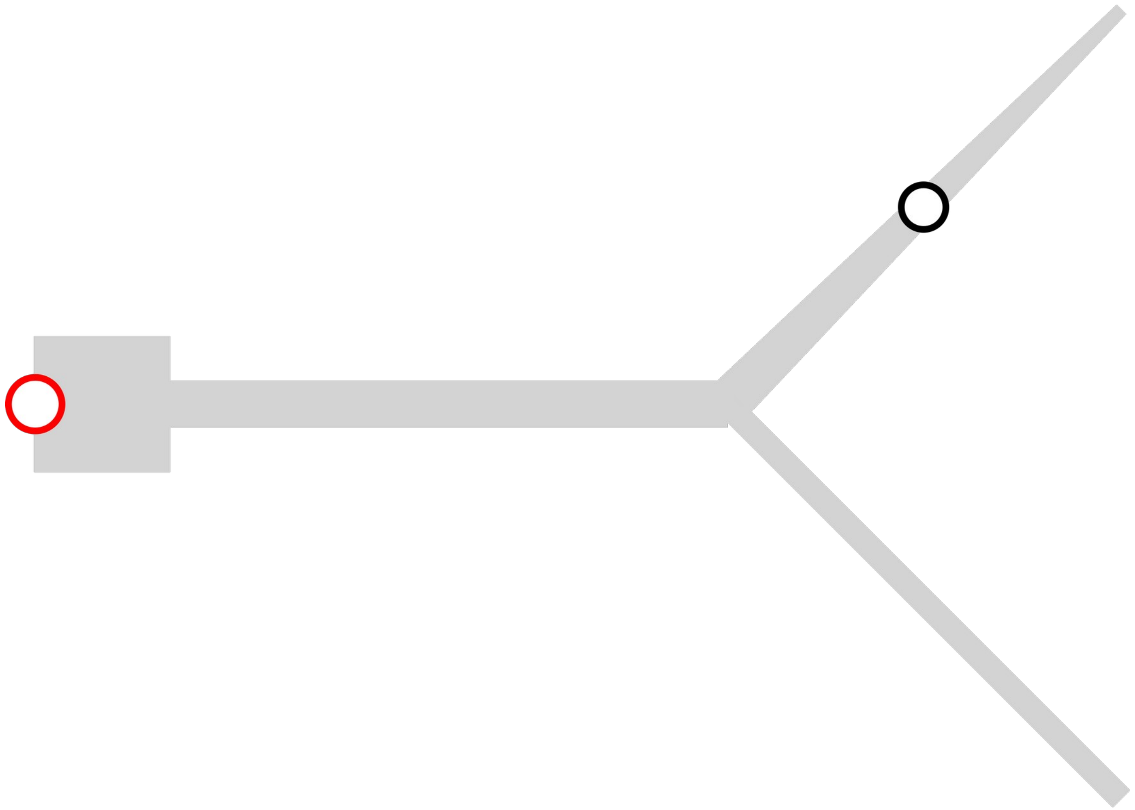
STEP 0

- Go to https://jupyter-jsc.fz-juelich.de and login
  - Add new Jupyterlab
    - System: Jusuf
    - Project: training2103
- Open Terminal
  - Bottom of screen or File > New > Terminal
  - source /p/project/training2103/arbor/activate
- Start ‘python’
  - import arbor

STEP 0

- Documentation at https://arbor.readthedocs.io
  - ‘Concepts’ for conceptual explanation
  - Python and C++ pages detail the API for the matching (roughly) conceptual page.
- Today: Python section
  - ‘Cable Cells’ page
  - Various ‘Cell *’ pages

STEP 1: MAKE A CELL



- Cell with segments (grey), junction site (black)

- Use segment_tree
- HH dynamics on soma
- Passive dendrites
- spike_detector at “root”

STEP 1: MAKE THE NETWORK

- Cells are the basic building blocks in Arbor
- Recipes tie them together
  - Therefore, the network is in the recipe
- You’ll make your own recipe by inheriting from arbor.recipe
- Some member functions need to be overridden

- TODO:
- Override all functions
- Connect each cell with the previous
- Place an event generator on the first cell.
- Probe the membrane voltage at "root"

STEP 3: MAKE THE SIMULATION

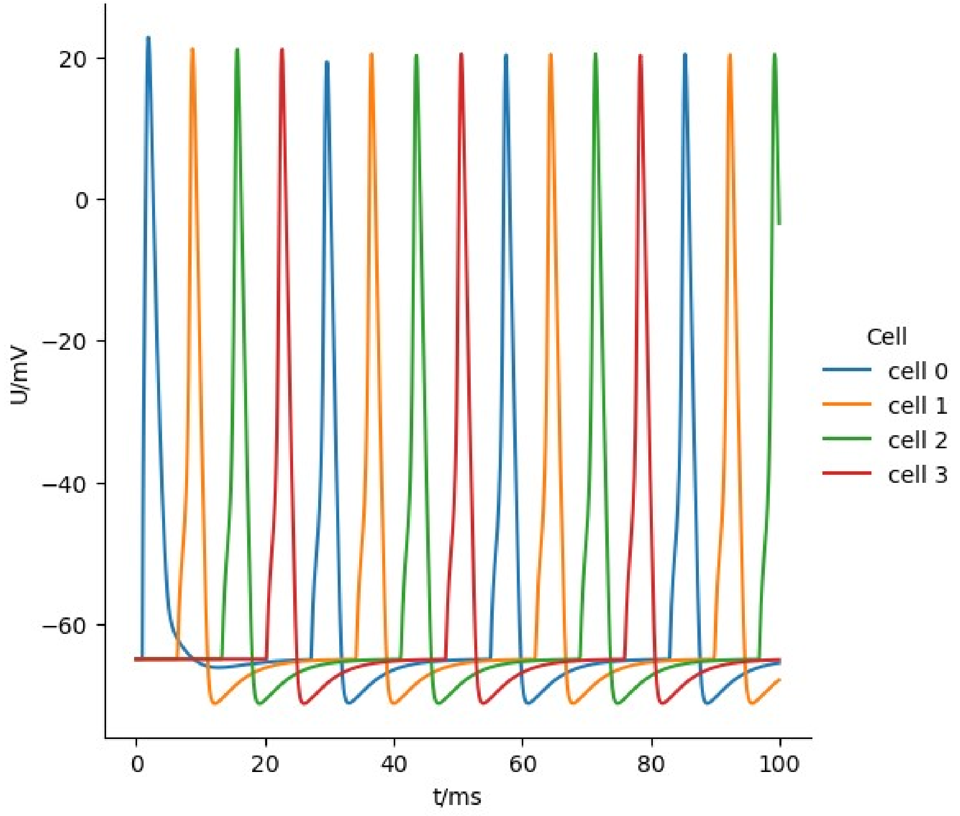
- Look through the Simulation page, Hardware page
  - Start with an arbor.simulation object, and see what you need for it.
    - (defaults are OK)
  - We need to store handles to the samplers, because later we'll use arbor.simulation.samples() to retrieve results.

STEP 4: SHOW YOUR RESULTS!

- We can extract results from
  - arbor.simulation.spikes
  - arbor.simulation.samples
- Print or plot.
  - Matplotlib, Seaborn (, Pandas?)

RESULTS

((0, 0), 1.42525561)
((1, 0), 8.32347893)
((2, 0), 15.22188202)
((3, 0), 22.12039976)
((0, 0), 29.08125267)
((1, 0), 36.00941203)
((2, 0), 42.93738402)
((3, 0), 49.86534444)
((0, 0), 56.79361867)
((1, 0), 63.72184675)
((2, 0), 70.65009778)
((3, 0), 77.57832833)
((0, 0), 84.50656887)
((1, 0), 91.43476941)
((2, 0), 98.3629114)



BONUS: ARBOR ON HPC

- source /p/project/training2103/arbor/bonus
- We're going big! MPI:
  https://en.wikipedia.org/wiki/Message_Passing_Interface
- Lookup arbor.mpi_comm and how to use it.
- An arbor.simulation will now run distributed
  - arbor.simulation.samples only knows about local results
  - You'll need to save results from the instances yourself!
- srun -A training2103 -n N_JOBS python network_ring_bonus.py

THE END

- Get the answers
  - source /p/project/training2103/arbor/hurray
- Leave some feedback
  - arbor-sim.github.io/feedback
- Questions?