A Graph-Based, In-Memory Workflow Library for Brain/MINDS 2.0

NEST Conference 2025

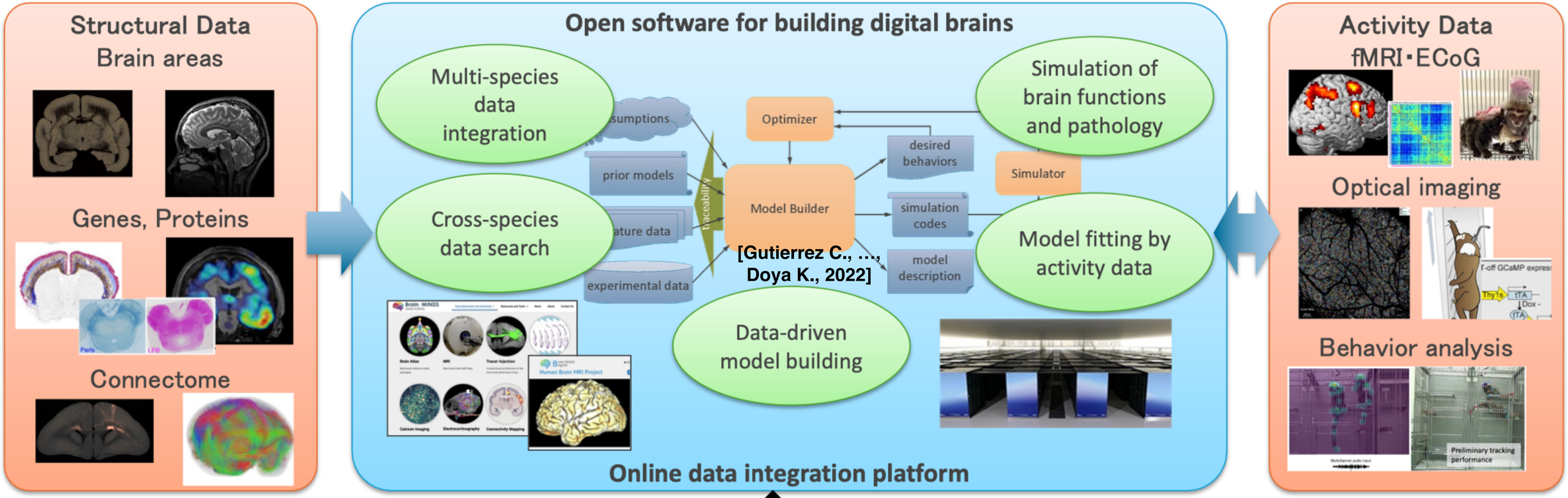
Carlos Gutierrez (1), Henrik Skibbe (2), Yukako Yamane (1), Kenji Doya (1)

(1) Okinawa Institute of Science and Technology Graduate University
(2) RIKEN CBS

Brain/MINDS 2.0 - The Digital Brain Project

OIST

Goal: open software for building digital brains
-online platform for data integration, model building and simulations-



AI  LLMs

The Digital Brain as a graph-based framework

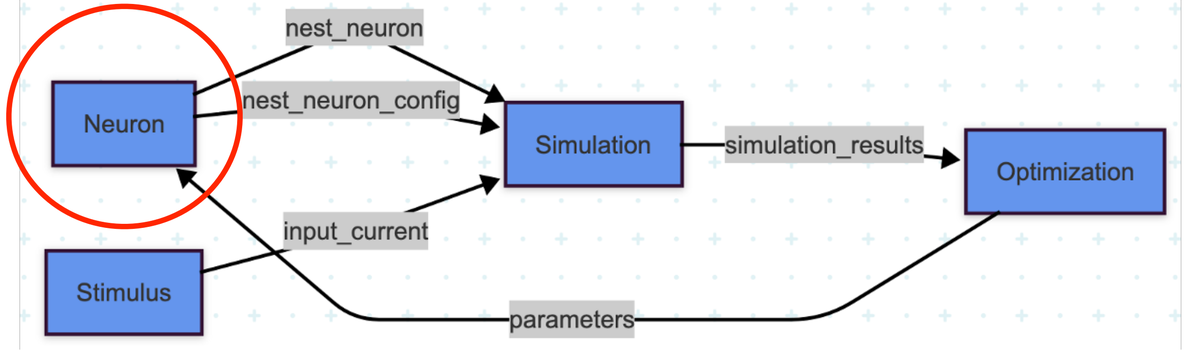
- The BM2 aims to create a framework that supports **interoperability with open data and popular neuroscience tools** by leveraging their Python interfaces.

- Modern brain modeling and simulation require complex sequences of data processing, model configuration, and analysis steps. While **powerful tools** exist, they frequently **demand advanced programming knowledge, limiting accessibility and collaboration potential**. **Shared models are frequently difficult to understand, execute and expand**.

- We present a python-based graph framework that **transforms brain modeling steps into modular, interpretable, reusable components through a node-edge** architecture.

Neuro-WorkFlow

- A NODE defines
  - **Parameters**
  - **Input port**
  - **Output port**
  - **Methods**
- NODES are connected by PORTS
- A NODE is a python class with a defined schema (structure)
- Example of a neuro-workflow with 4 NODES



Node Schema i.e: Neuron

Component	Details
Node Type	`nest_neuron_setup`
Description	Sets up NEST neurons for neural simulations

Parameters

Parameter Name	Type	Description	Default Value
`nest_model`	string	NEST neuron model type	'iaf_psc_alpha'
`threshold`	float	Firing threshold (mV)	-55.0
`resting_potential`	float	Resting membrane potential (mV)	-70.0
`time_constant`	float	Membrane time constant (ms)	20.0
`refractory_period`	float	Refractory period after spiking (ms)	2.0

Input Ports

Port Name	Type	Description	Optional
`parameters`	Dict	Parameter updates from optimization	Yes

Output Ports

Port Name	Type	Description
`nest_neuron`	Object	NEST neuron object
`nest_neuron_config`	Dict	Configuration of the NEST neuron

Methods

Method Name	Description	Inputs	Outputs
`configure`	Sets up the node with initial parameters	nest_model, threshold, resting_potential, time_constant, refractory_period	None
`create_neuron`	Creates a NEST neuron with the specified parameters	parameters (from input port)	nest_neuron, nest_neuron_config

Process Steps

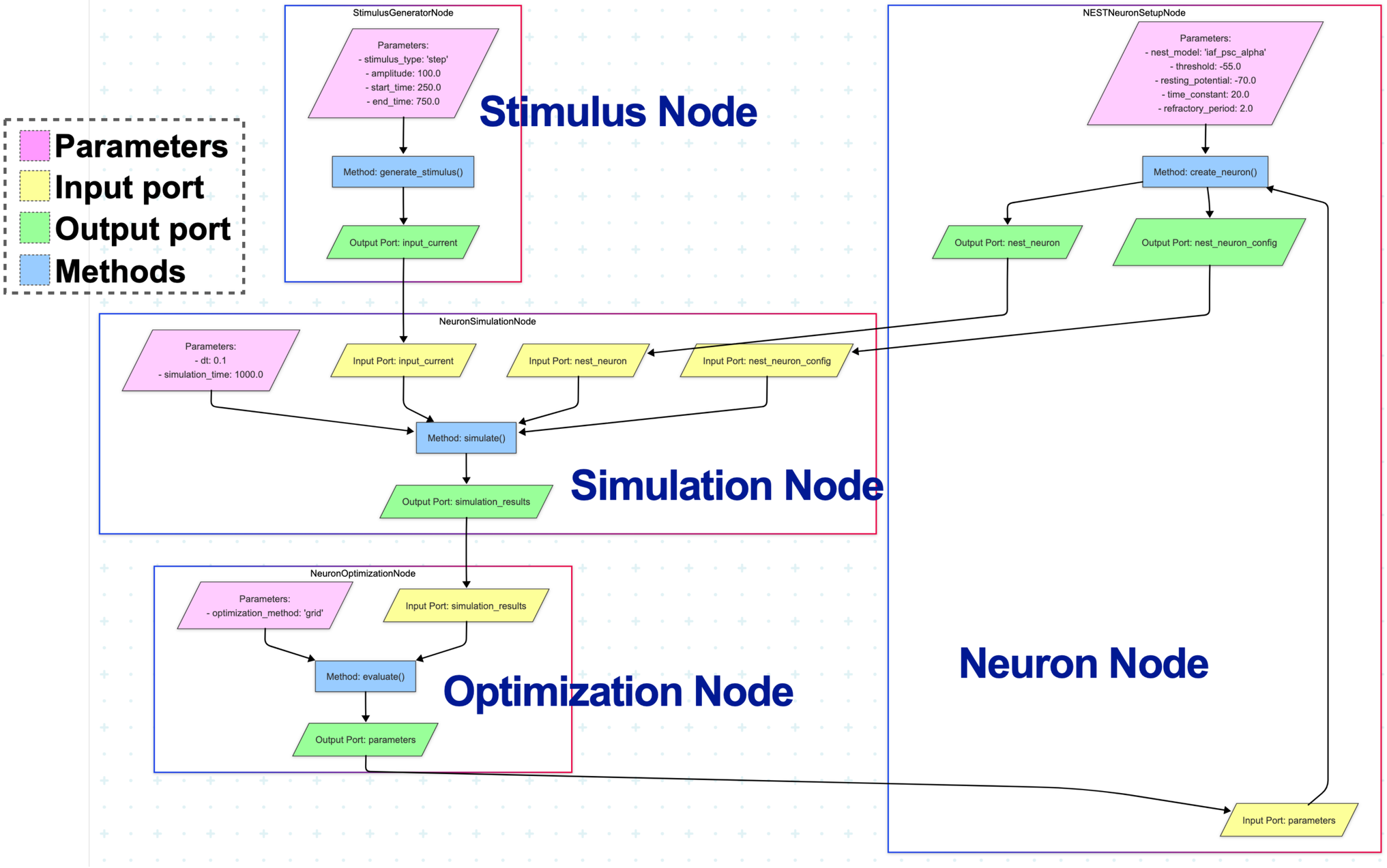
1. Receive parameters (if any) from optimization node
2. Execute `create_neuron` method
3. Output the NEST neuron object and configuration

In Python a Node is defined as a Class:

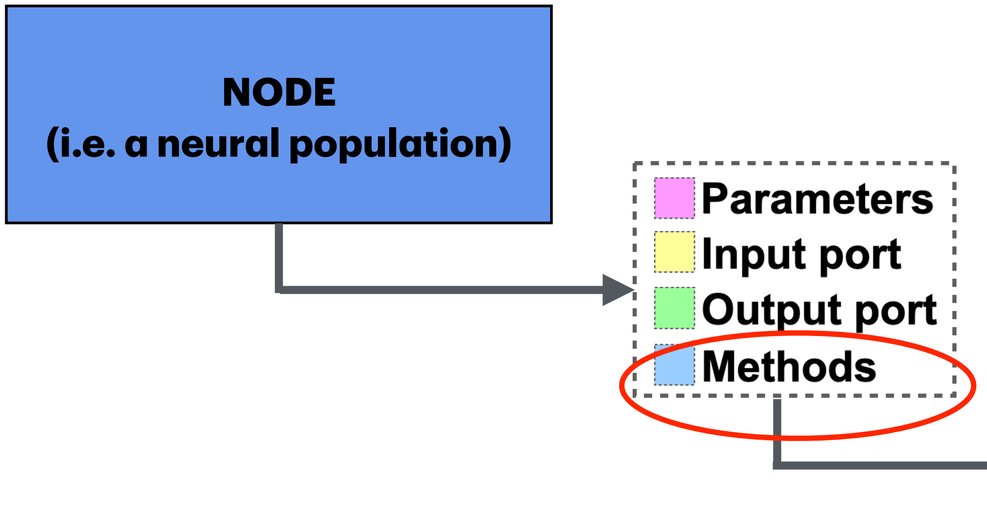
```python
class NESTNeuronSetupNode(Node):
    """Node for creating and configuring neuron models.
    This class represents a neuron model with configurable parameters.
    In a real implementation, this could be a more complex model or
    interface with external neural simulators."""

    NODE_DEFINITION = NodeDefinitionSchema(
        type='neuron_setup',
        description='Creates and configures a neuron model in NEST',
        parameters={
            'nest_model': ParameterDefinition(
                default_value='iaf_psc_alpha',
                description='neuron model name in NEST'
            ),
            'threshold': ParameterDefinition(
                default_value=-55.0,
                description='Firing threshold (mV)',
                constraints={'min': -70.0, 'max': -40.0},
                optimizable=True,
                optimization_range=[-65.0, -45.0]
            ),
            'resting_potential': ParameterDefinition(
                default_value=-70.0,
                description='Resting membrane potential (mV)',
                constraints={'min': -80.0, 'max': -60.0},
            ),
            'time_constant': ParameterDefinition(
                default_value=20.0,
                description='Membrane time constant (ms)',
                constraints={'min': 5.0, 'max': 50.0},
                optimizable=True,
                optimization_range=[10.0, 30.0]
            ),
            'refractory_period': ParameterDefinition(
                default_value=2.0,
                description='Refractory period (ms)',
                constraints={'min': 1.0, 'max': 5.0}
            )
        },
        inputs={
            'parameters': PortDefinition(
                type=PortType.DICT,
                description='Parameters to configure the neuron',
```

```python
def create_neuron(self, parameters: Optional[Dict[str, Any]] = None) -
    """Create and configure a neuron model.

    Args:
        parameters: Optional parameters to override defaults

    Returns:
        Dictionary with neuron model and configuration
    """
    # If parameters are provided, configure the node
    if parameters:
        self.configure(**parameters)

    # Create nest neuron object
    neuro_params = {
        "V_th": self._parameters['threshold'],
        "E_L": self._parameters['resting_potential'],
        "tau_m": self._parameters['time_constant'],
        "t_ref": self._parameters['refractory_period'],
    }

    nest.set_verbosity("M_ERROR")
    nest.ResetKernel()
    neuron = nest.Create(self._parameters['nest_model'], params=neuro_
```

Example of a neuro-workflow with 4 NODES



A NODE is a python class



Python interfaces
(Libraries, methods, etc.)

Interoperability

Popular neuroscience tools

NEST,
TVB,
NEURON,
BMTK,
Custom python code,
etc.

Open data

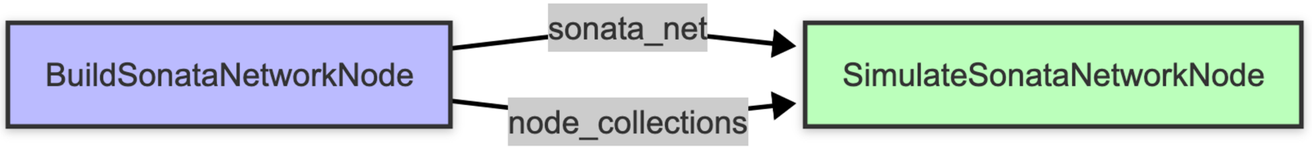
s3://brainminds-marmoset-connectivity

s3://aind-open-data
s3://allencell/
s3://allen-mouse-brain-atlas

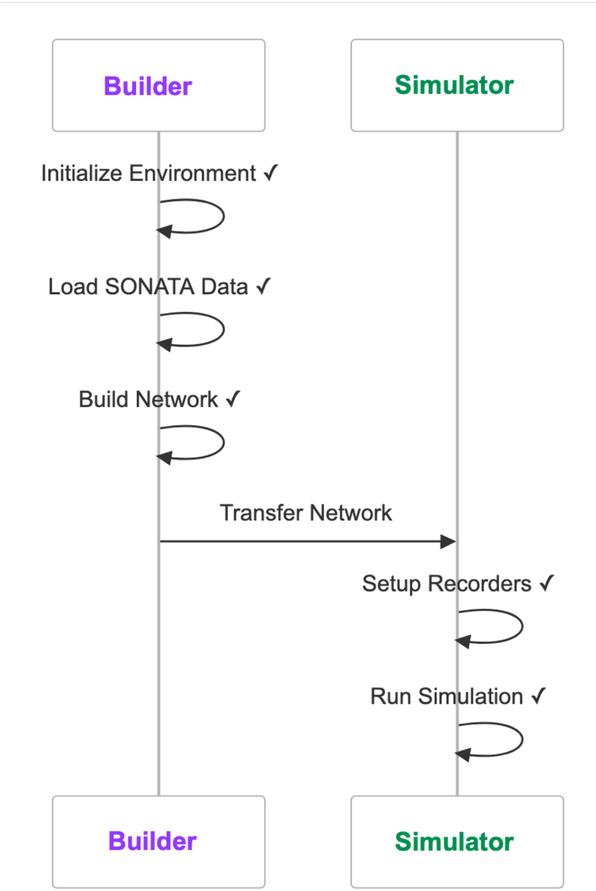
BMCR,
Allen Institute resources,
EBRAINS resources,
Connectomes,
RIKEN CBS data sharing,
etc.

OIST

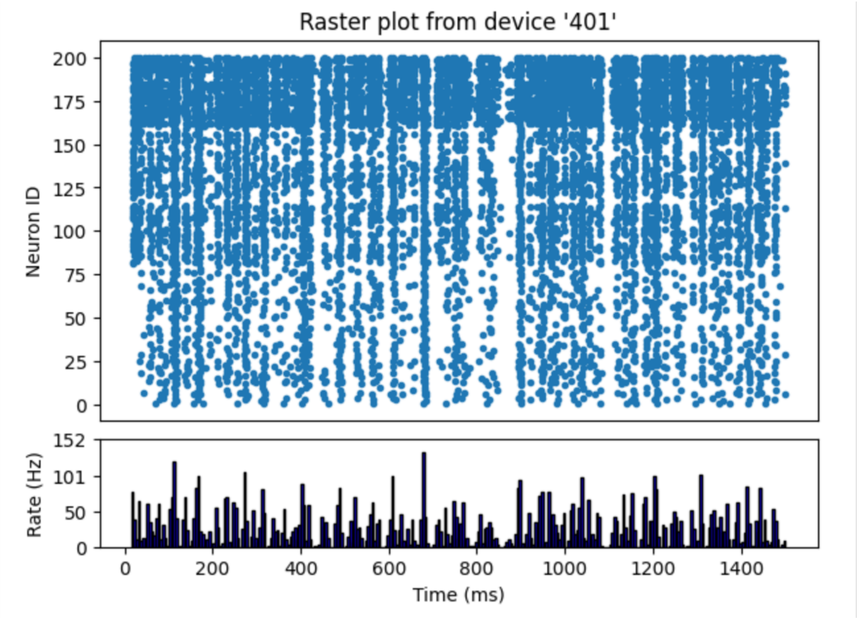
This example shows a **SONATA-NEST** implementation



BuildSonataNetworkNode: Network construction from SONATA files.

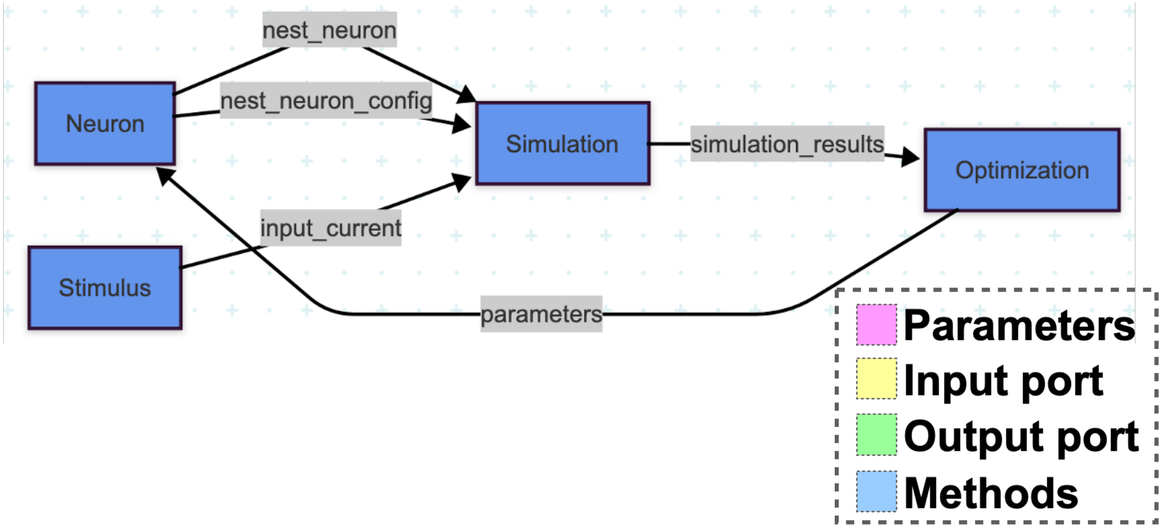


SimulateSonataNetworkNode: Network simulation.



Simplicity in the creation of workflows

```python
# Create nodes
neuron_node = NESTNeuronSetupNode("neuron_node")
stimulus_node = StimulusGeneratorNode("stimulus_node")
simulation_node = NeuronSimulationNode("simulation_node")
optimization_node = JointOptimizationNode("optimization_node")

# Configure nodes with initial parameters
neuron_node.configure(
    nest_model='iaf_psc_alpha',
    threshold=-55.0,
    resting_potential=-70.0,
    time_constant=20.0,
    refractory_period=2.0
)

stimulus_node.configure(
    stimulus_type='step',
    amplitude=100.0,
    start_time=250.0,
    end_time=750.0
)
```




```python
workflow = (
    WorkflowBuilder("encapsulated_optimization_workflow")
    .add_node(neuron_node)
    .add_node(stimulus_node)
    .add_node(simulation_node)
    .add_node(optimization_node)

    # Connect neuron and stimulus nodes to simulation node
    .connect("neuron_node", "nest_neuron", "simulation_node", "nest_neuron")
    .connect("neuron_node", "nest_neuron_config", "simulation_node", "nest_neuron_config")
    .connect("stimulus_node", "input_current", "simulation_node", "input_current")

    # Connect simulation results to optimization node
    .connect("simulation_node", "simulation_results", "optimization_node", "simulation_results")

    # Connect parameter metadata to optimization node
    .connect("neuron_node", "parameter_metadata", "optimization_node", "parameter_metadata")
    .connect("stimulus_node", "parameter_metadata", "optimization_node", "parameter_metadata")
```

It is possible to interact with nodes and ports using methods like:

NODES

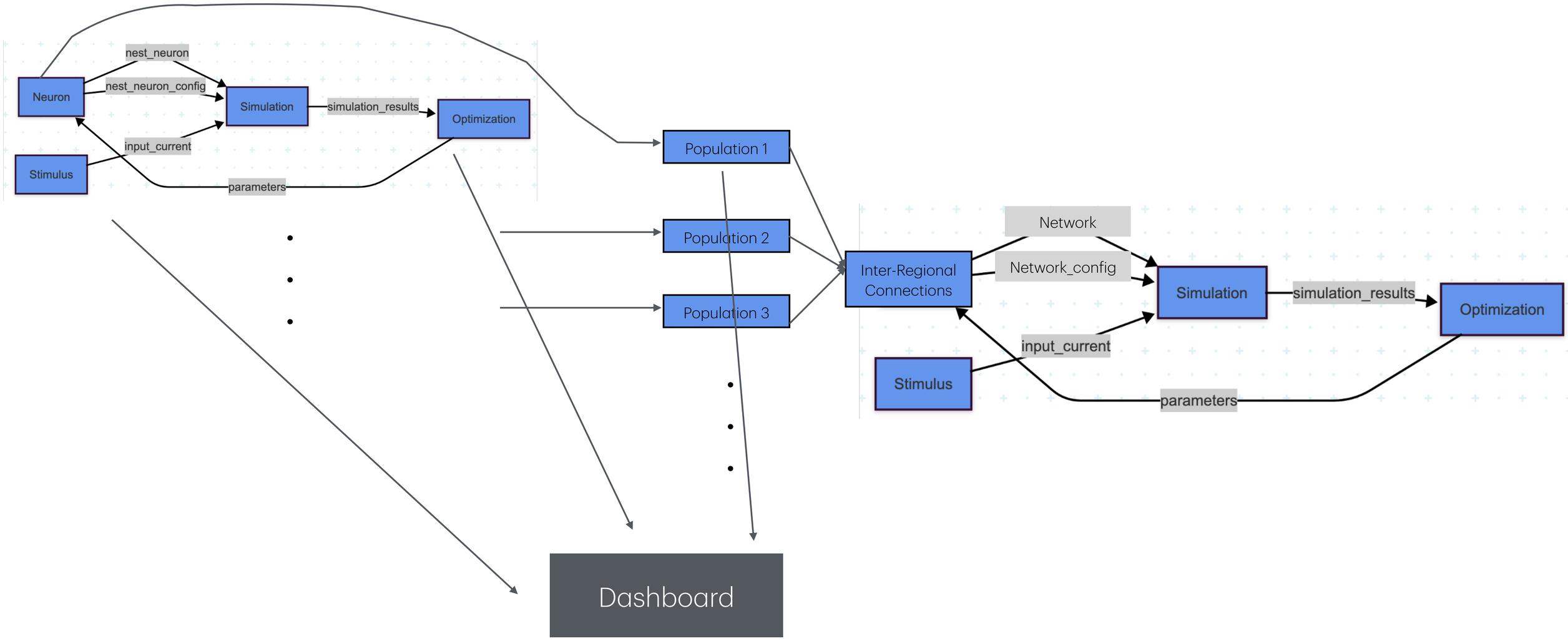
```
# Get an input port object
input_port = node.get_input_port("input_name")

# Get an output port object
output_port = node.get_output_port("output_name")

# Get the value of an output port
output_value = node.get_output("output_name")

# Get all current parameter values
parameters = node.get_parameters()
print(f"Current parameters: {parameters}")

# Get optimizable parameters
optimizable_params = node.get_optimizable_parameters()
print(f"Optimizable parameters: {optimizable_params}")

# Get process steps
process_steps = node.get_process_steps()
print(f"Process steps: {process_steps}”)
```

WORKFLOW

```
# Get a node from a workflow
node = workflow.get_node("node_name")

# Get all nodes in a workflow
nodes = workflow.get_nodes()
for name, node in nodes.items():
    print(f"Node: {name}")

# Get all connections in a workflow
connections = workflow.get_connections()
for conn in connections:
    print(f"Connection: {conn}")
```

OIST

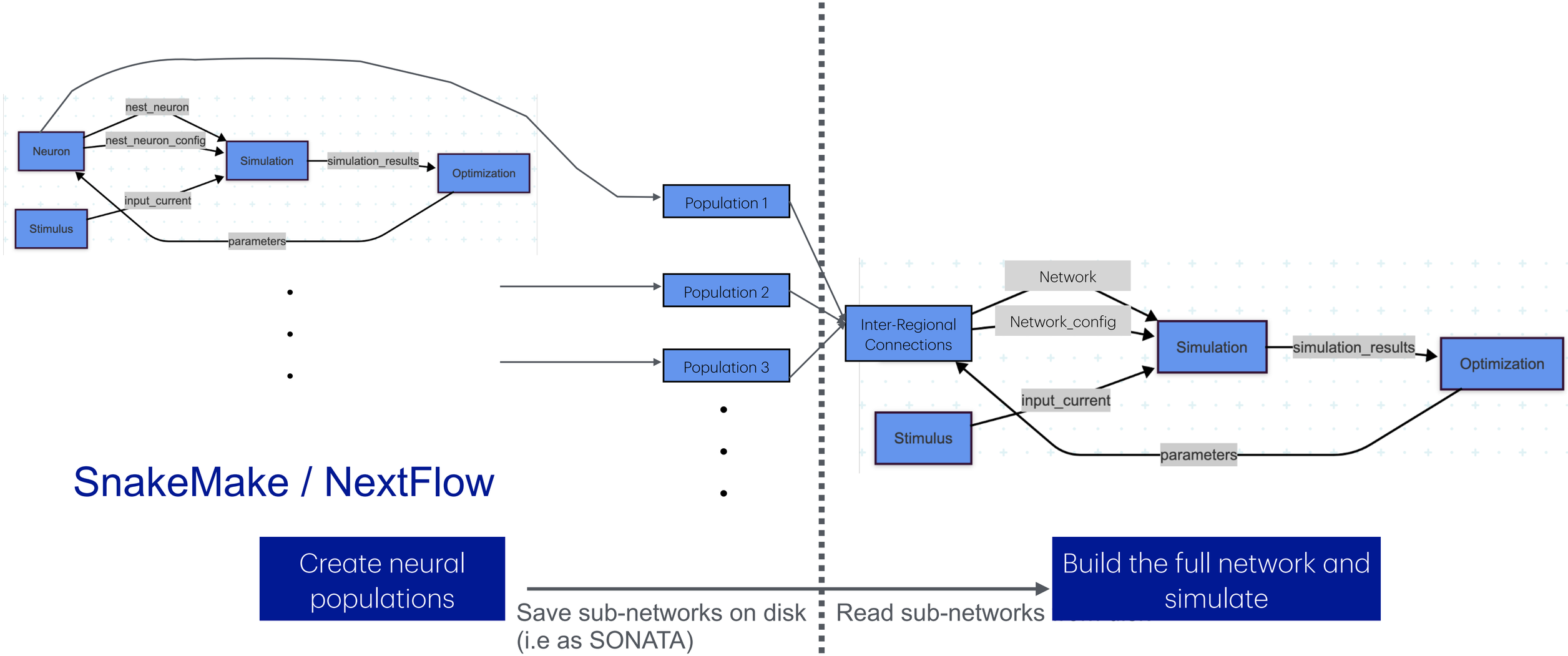
Example workflow for brain modeling:



Mapping Neuro-Workflow to SnakeMake/Nextflow

OIST

Neuro-Workflow



SnakeMake / NextFlow

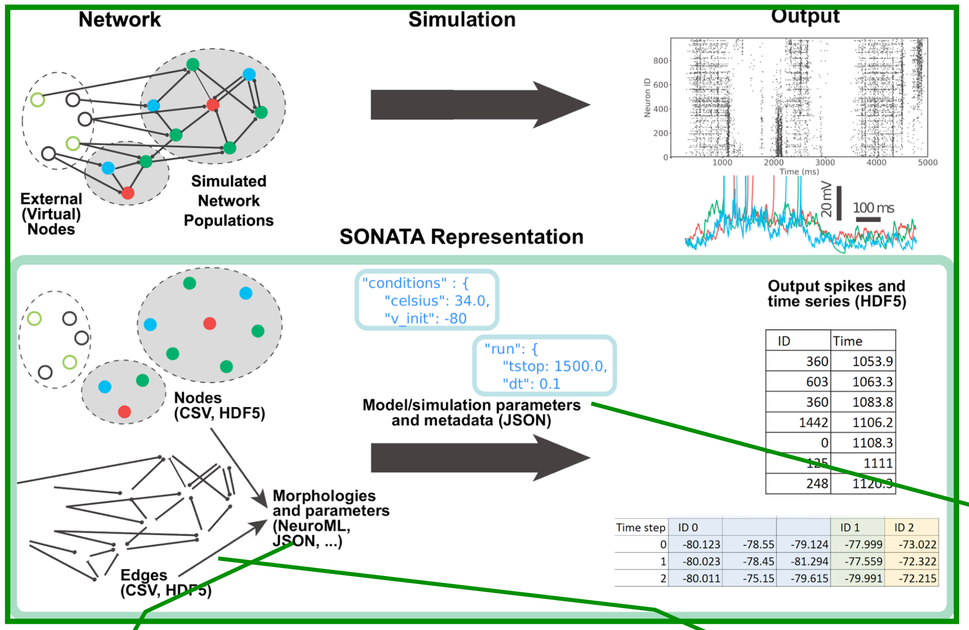
Create neural populations

Save sub-networks on disk (i.e as SONATA)

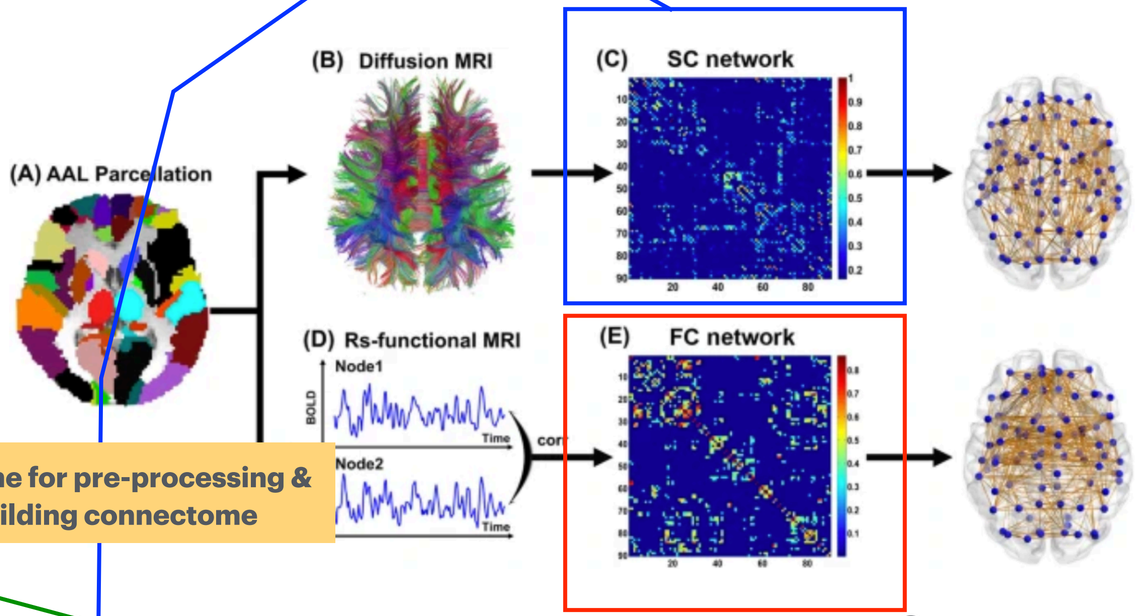
Read sub-networks

Build the full network and simulate

Example of Data-to-Model Mapping



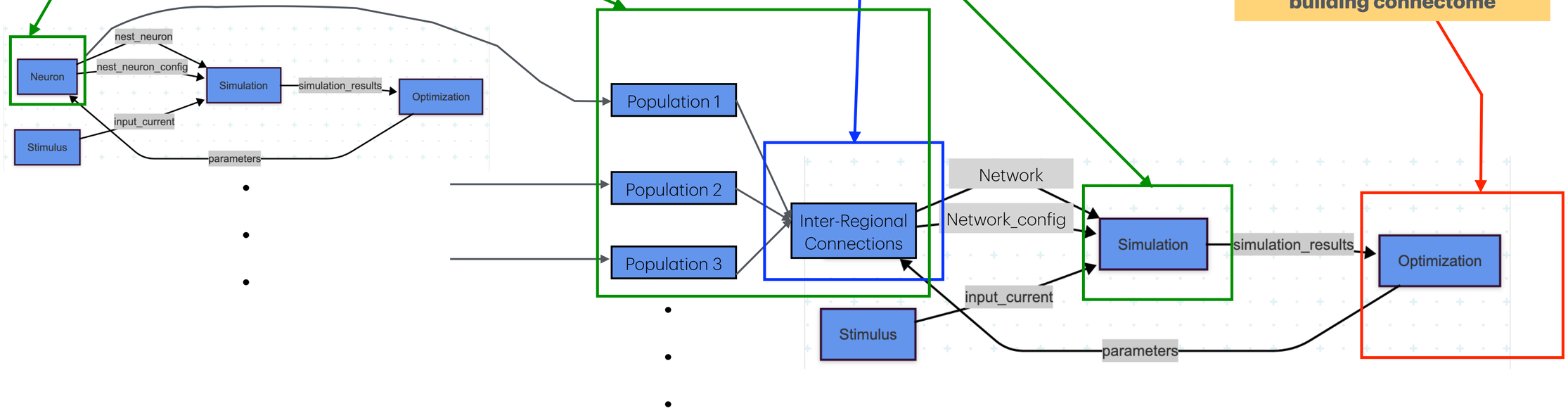
Data in SONATA



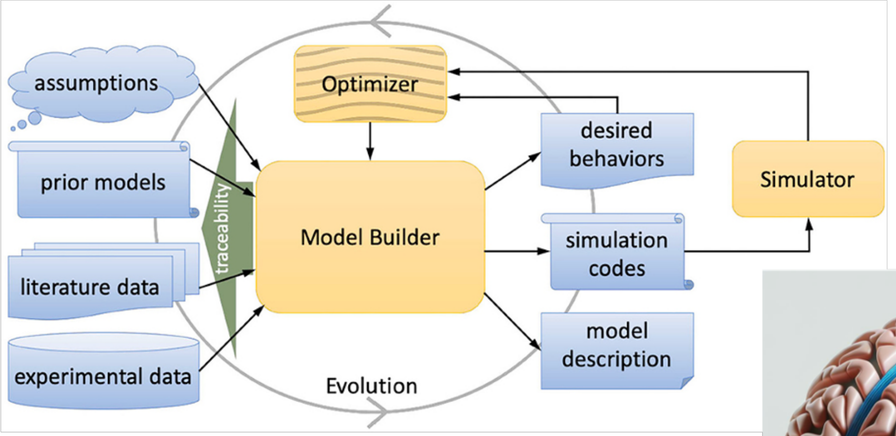
Pipeline for pre-processing & building connectome

Connectomme

Pipeline for pre-processing & building connectome



Community-based approach for building digital brains



- The model builder aims to be **composed by several custom nodes built by BM2** researchers and collaborators.
- Several **easy-to-understand complex brain models** and workflows may be built and **deployed easily**.
- Data integration and data analysis workflows can be deployed as well.
- Machine learning / AI dedicated nodes can also be created and integrated.

Thank you

Annex

2S05m The Digital Brain: Frameworks for Current and Future Innovation

Date： July 25 8:45-10:45

Venue：Room 5 (301B)

Organizers： Carlos Enrique Gutierrez (Okinawa Institute of Science and Technology Graduate University, Neural Computation Unit)
Viktor Jirsa (Institut de Neurosciences des Systemes)

Speakers： Viktor Jirsa (Institut de Neurosciences des Systemes)
Salvador Dura-Bernal (State University of New York (SUNY) Downstate Health Sciences University)
Kenji Doya (Neural Computation Unit, Okinawa Institute of Science and Technology Graduate University)
Sacha van Albada (Institute for Advanced Simulation (IAS-6). Computational and Systems Neuroscience. Jülich Research Centre)

2S05a Advances in Brain Modeling and AI applications

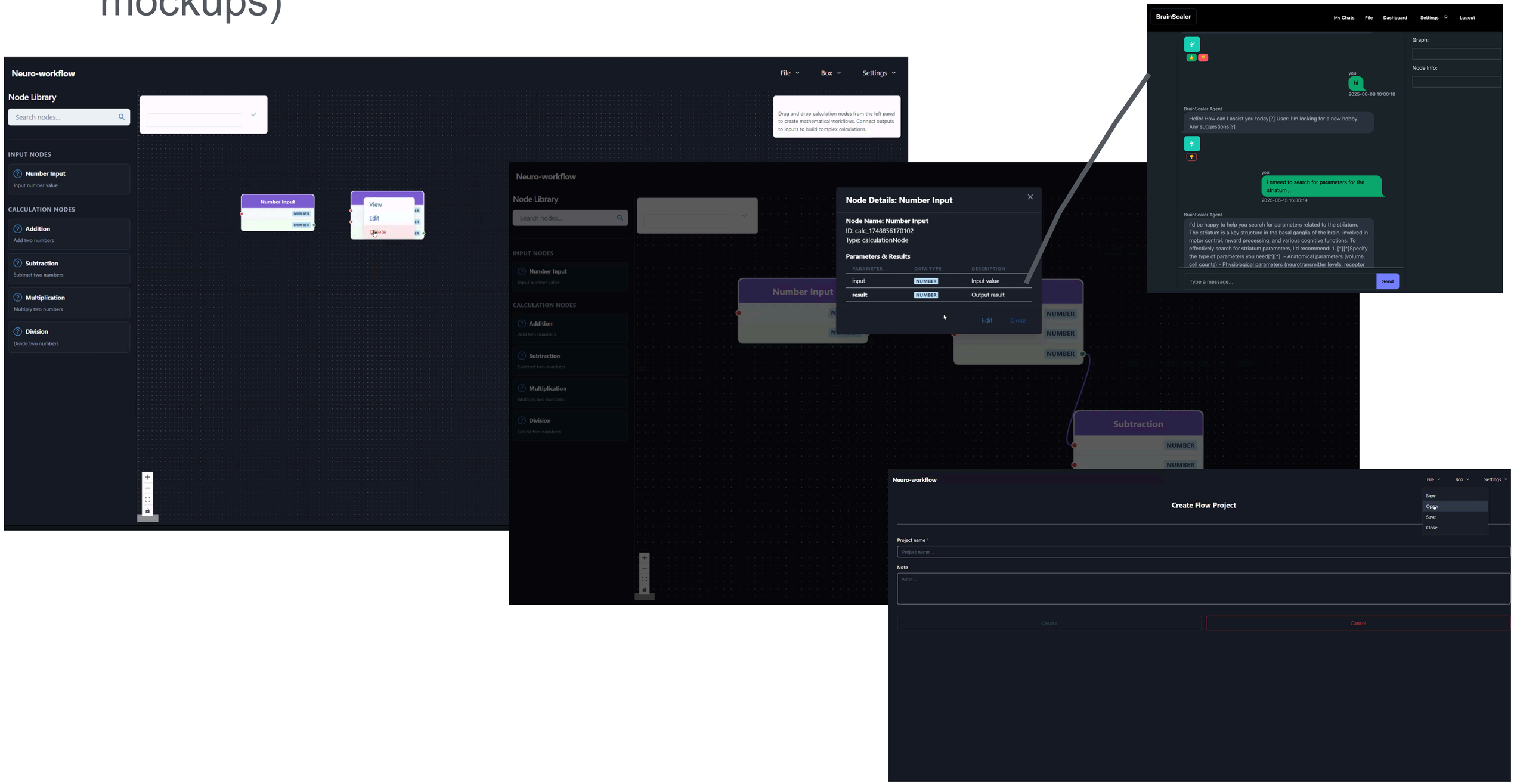
Date： July 25 14:10-16:10

Venue：Room 5 (301B)

Organizers： Ken Nakae (NINS)
Kenji Doya (OIST)

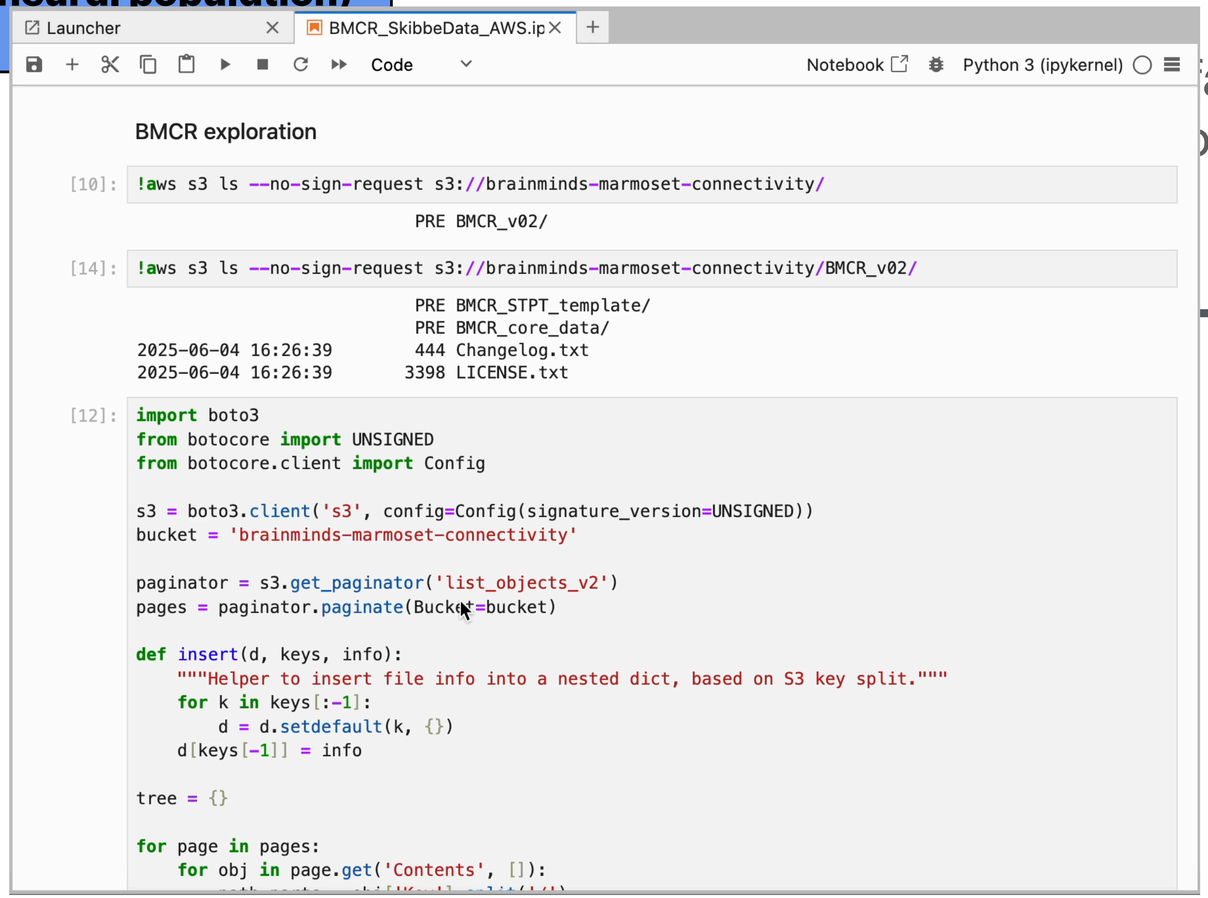
Speakers： Yi Zeng (Brain-inspired Cognitive Intelligence Lab, Institute of Automation, Chinese Academy of Sciences)
Shinya Ito (Allen Institute, Modeling, Analysis & Theory (MAT) team)
Takuya Isomura (RIKEN Center for Brain Science)
Charissa Poon (RIKEN Center for Brain Science)

Development of a graphical interface for neuron-workflow (preliminary mockups)



A NODE is a python class

NODE
(i.e. a neural population)



Interoperability

Popular neuroscience tools

TVB,
NEST,
NEURON,
BMTK,
Custom python code,
etc.

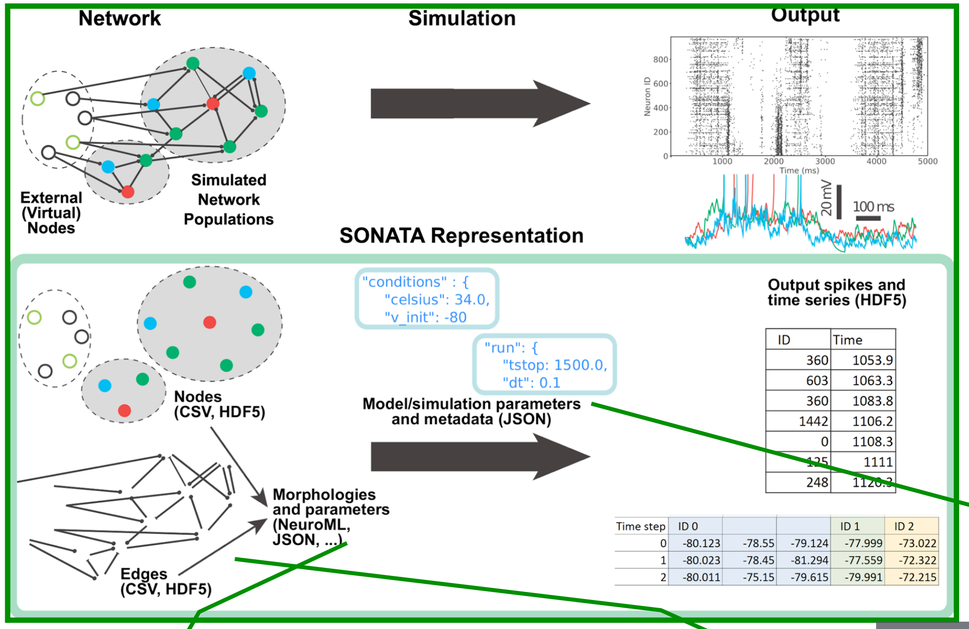
`s3://brainminds-marmoset-connectivity`

Open data

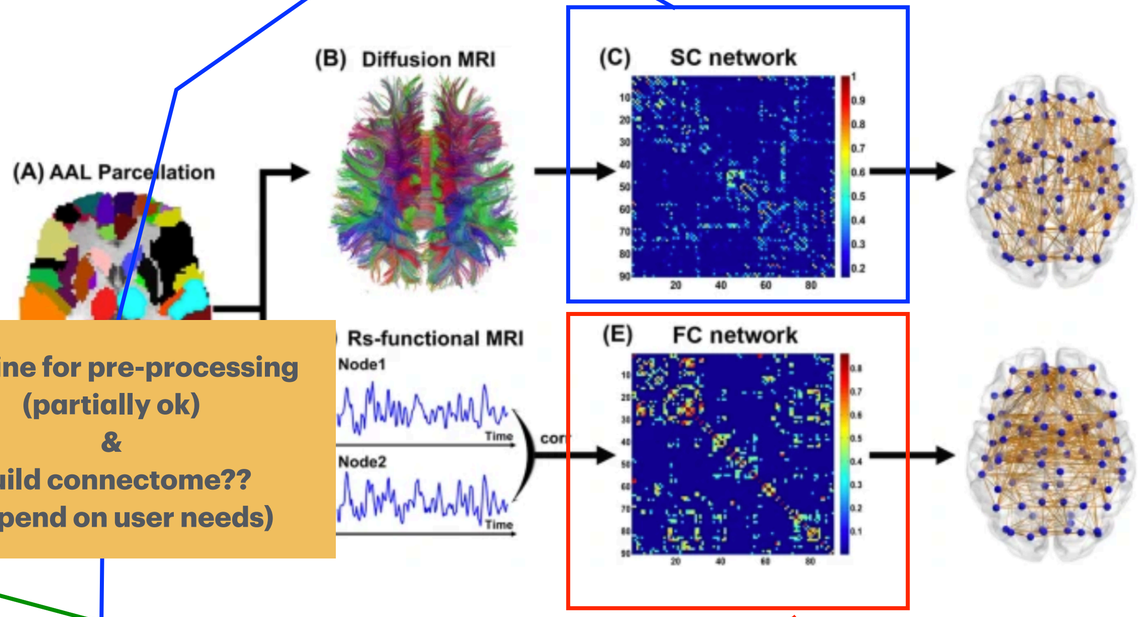
BMCR,
Allen Institute resources,
EBRAINS
Connectomes,
RIKEN CBS data sharing,
etc.

`s3://aind-open-data`
`s3://allencell/`
`s3://allen-mouse-brain-atlas`

Data in SONATA



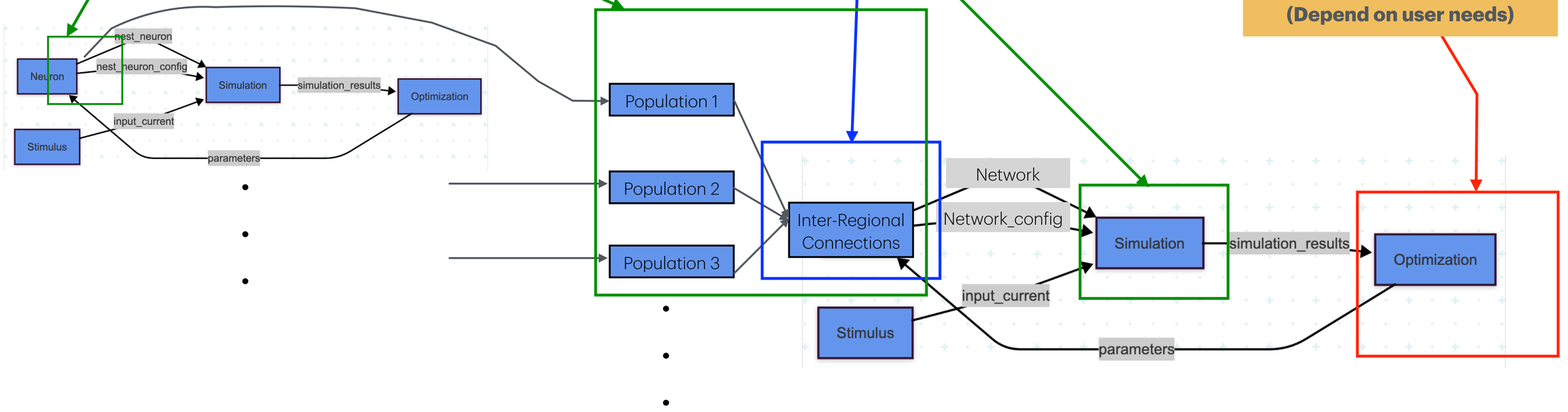
Connectomme



Pipeline for pre-processing (partially ok) & build connectome?? (Depend on user needs)

Pipeline for pre-processing & build connectome?? (Depend on user needs)

WorkFlow - Digital Brains



WorkFlow - Digital Brains - Scheme

A NODE defines **inputs, outputs, parameters, and processing steps** (functions).

It handles data flow and type validation and it can be **extended to create specialized/custom nodes**.

Parameters
Input port
Output port
Methods

NESTNeuronSetupNode Schema

Node Structure Table

Component	Details
Node Type	`nest_neuron_setup`
Description	Sets up NEST neurons for neural simulations

Parameters

Parameter Name	Type	Description	Default Value
`nest_model`	string	NEST neuron model type	'iaf_psc_alpha'
`threshold`	float	Firing threshold (mV)	-55.0
`resting_potential`	float	Resting membrane potential (mV)	-70.0
`time_constant`	float	Membrane time constant (ms)	20.0
`refractory_period`	float	Refractory period after spiking (ms)	2.0

Input Ports

Port Name	Type	Description	Optional
`parameters`	Dict	Parameter updates from optimization	Yes

Output Ports

Port Name	Type	Description
`nest_neuron`	Object	NEST neuron object
`nest_neuron_config`	Dict	Configuration of the NEST neuron

Methods

Method Name	Description	Inputs	Outputs
`configure`	Sets up the node with initial parameters	nest_model, threshold, resting_potential, time_constant, refractory_period	None
`create_neuron`	Creates a NEST neuron with the specified parameters	parameters (from input port)	nest_neuron, nest_neuron_config

Process Steps

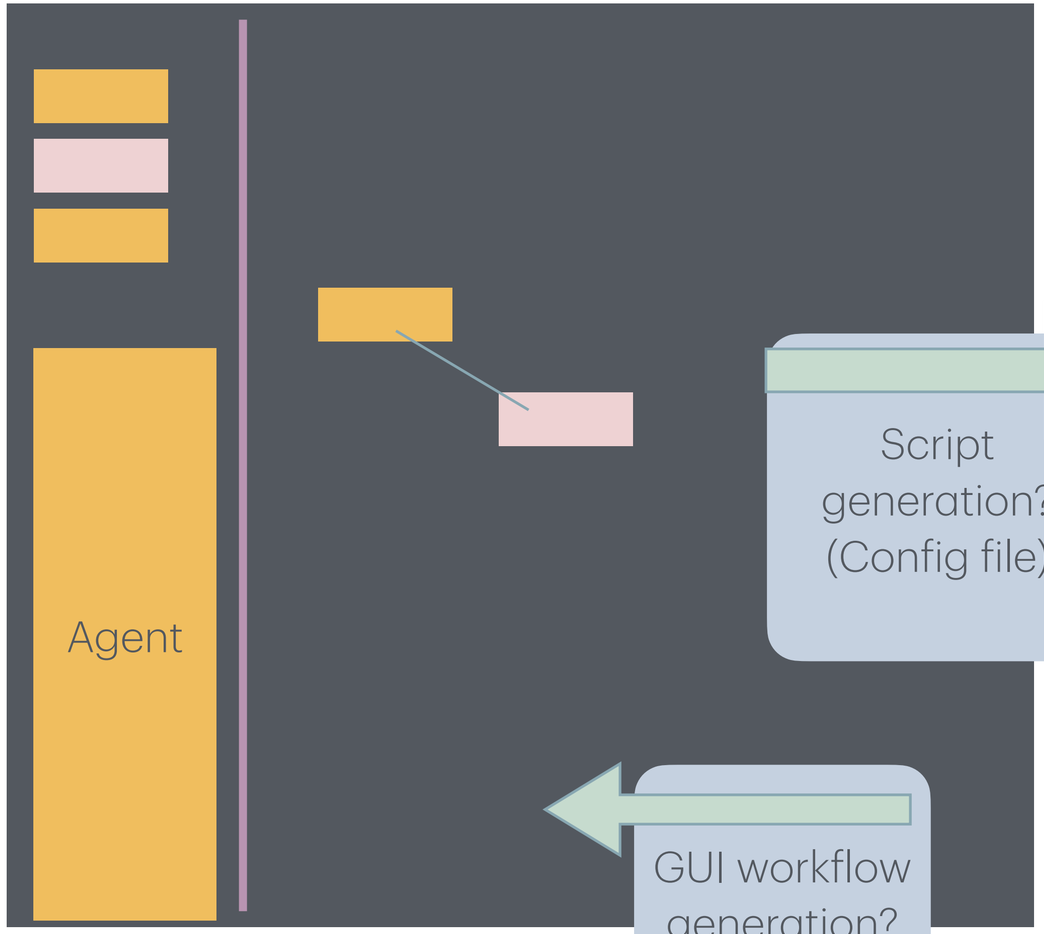
1. Receive parameters (if any) from optimization node
2. Execute `create_neuron` method
3. Output the NEST neuron object and configuration

In Python a Node is defined as a Class:

```python
class NESTNeuronSetupNode(Node):
    """Node for creating and configuring neuron models.
    This class represents a neuron model with configurable parameters.
    In a real implementation, this could be a more complex model or
    interface with external neural simulators."""

    NODE_DEFINITION = NodeDefinitionSchema(
        type='neuron_setup',
        description='Creates and configures a neuron model in NEST',
        parameters={
            'nest_model': ParameterDefinition(
                default_value='iaf_psc_alpha',
                description='neuron model name in NEST'
            ),
            'threshold': ParameterDefinition(
                default_value=-55.0,
                description='Firing threshold (mV)',
                constraints={'min': -70.0, 'max': -40.0},
                optimizable=True,
                optimization_range=[-65.0, -45.0]
            ),
            'resting_potential': ParameterDefinition(
                default_value=-70.0,
                description='Resting membrane potential (mV)',
                constraints={'min': -80.0, 'max': -60.0},
            ),
            'time_constant': ParameterDefinition(
                default_value=20.0,
                description='Membrane time constant (ms)',
                constraints={'min': 5.0, 'max': 50.0},
                optimizable=True,
                optimization_range=[10.0, 30.0]
            ),
            'refractory_period': ParameterDefinition(
                default_value=2.0,
                description='Refractory period (ms)',
                constraints={'min': 1.0, 'max': 5.0}
            )
        },
        inputs={
            'parameters': PortDefinition(
                type=PortType.DICT,
                description='Parameters to configure the neuron',
```

```python
def create_neuron(self, parameters: Optional[Dict[str, Any]] = None) -> Dict[str, Any]:
    """Create and configure a neuron model.

    Args:
        parameters: Optional parameters to override defaults

    Returns:
        Dictionary with neuron model and configuration
    """
    # If parameters are provided, configure the node
    if parameters:
        self.configure(**parameters)

    # Create nest neuron object
    neuro_params = {
        "V_th": self._parameters['threshold'],
        "E_L": self._parameters['resting_potential'],
        "tau_m": self._parameters['time_constant'],
        "t_ref": self._parameters['refractory_period'],
    }

    nest.set_verbosity("M_ERROR")
    nest.ResetKernel()
    neuron = nest.Create(self._parameters['nest_model'], params=neuro_params)
```

Python




```
# Create a network builder node
build_network = BuildSonataNetworkNode("SonataNetworkBuilder")
```

```
# Configure the network builder
build_network.configure(
    sonata_path="../data/300_pointneurons",  # Path to our
SONATA configuration
    net_config_file="circuit_config.json",
    sim_config_file="simulation_config.json",
    hdf5_hyperslab_size=1024
)
```

```
# Create a simulation node
simulate_network =
SimulateSonataNetworkNode("SonataNetworkSi
mulation")
```

```
workflow_builder.connect(
    "SonataNetworkBuilder", "sonata_net",
    "SonataNetworkSimulation", "sonata_net"
)
```

```
# Configure the network builder
build_network.configure(
    sonata_path="../data/300_pointneurons",  # Path to our
SONATA configuration
    net_config_file="circuit_config.json",
    sim_config_file="simulation_config.json",
    hdf5_hyperslab_size=3000
)
```

EOF