Omnetpy: using Python to write OMNeT++ simulations

Marcos Modenesi

Final project for Computer Science degree

Facultad de Matemática, Astronomía y Física
Directors: **Dr. Juan A. Fraire, Lic. Pablo G. Ventura**

UNC Universidad Nacional de Córdoba

Structure of the presentation

1. What?
2. Why?
3. How?
4. Results

1 / 4 - What?

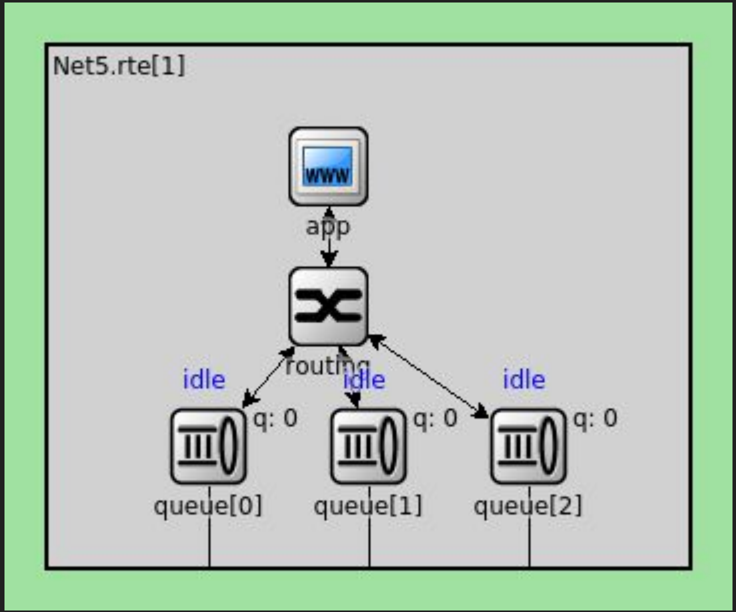
How do we use OMNeT++?

1. Network Topology specification
   - modules, submodules
   - connections
   - channel properties
2. Simulation configuration
   - actual values to parameters
3. Network behavior specification
   - what modules actually *do*

How do we use OMNeT++?

1. Network Topology specification

```
package node;

module Node
{
    parameters:
        int address;
        string appType;
        @display("i=misc/node_vs,gold");
    gates:
        inout port[];
        submodules:
    app: <appType> like IApp {
        parameters:
            address = address;
    }
    routing: Routing {
        parameters:
        gates:
            in[sizeof(port)];
            out[sizeof(port)];
        }
    queue[sizeof(port)]: L2Queue {
        parameters:
            @display("p=80,200,row");
    }
    connections:
        routing.localOut --> app.in;
        routing.localIn <-- app.out;
        for i=0..sizeof(port)-1 {
            routing.out[i] --> queue[i].in;
            routing.in[i] <-- queue[i].out;
            queue[i].line <--> port[i];
       }
}
```



How do we use OMNeT++?

2. Simulation configuration

```
[Config Net5]
network = networks.Net5
**.destAddresses = "1 3"
**.sendIaTime = uniform(500ms, 1500ms)  # high traffic
```

How do we use OMNeT++?

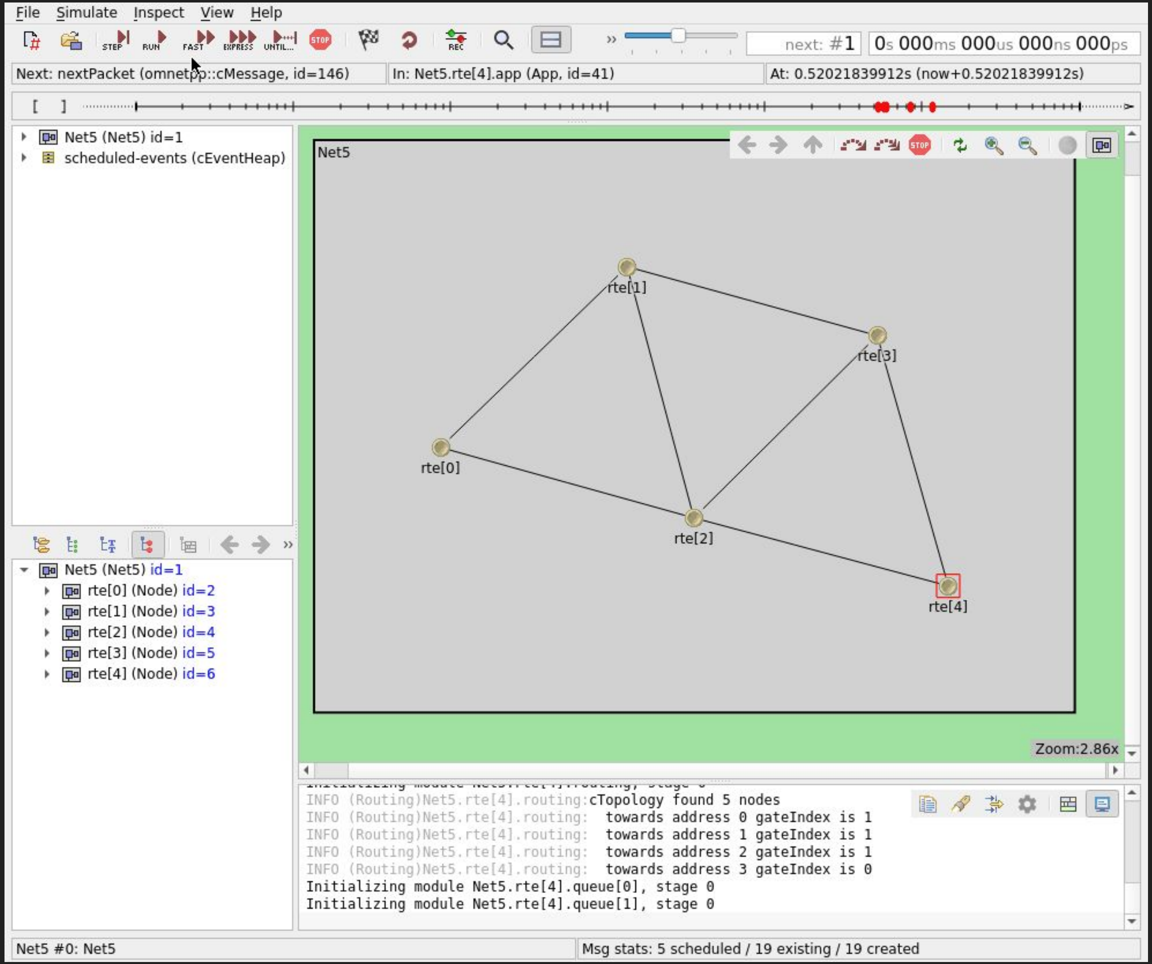
3. Network behavior specification

```
class Routing : public cSimpleModule
{
  private:
    int myAddress;

    typedef std::map<int, int> RoutingTable;  // destaddr -> gateindex
    RoutingTable rtable;

    simsignal_t dropSignal;
    simsignal_t outputIfSignal;

  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(Routing);
```

subclass cSimpleModule

implement abstract methods

register the module


File Simulate Inspect View Help
next: #1
0s 000ms 000us 000ns 000ps
Next: nextPacket (omnetpp::cMessage, id=146)
In: Net5.rte[4].app (App, id=41)
At: 0.52021839912s (now+0.52021839912s)
Net5 (Net5) id=1
scheduled-events (cEventHeap)
Net5
rte[0]
rte[1]
rte[2]
rte[3]
rte[4]
Zoom:2.86x
Net5 (Net5) id=1
rte[0] (Node) id=2
rte[1] (Node) id=3
rte[2] (Node) id=4
rte[3] (Node) id=5
rte[4] (Node) id=6
INFO (Routing)Net5.rte[4].routing:cTopology found 5 nodes
INFO (Routing)Net5.rte[4].routing: towards address 0 gateIndex is 1
INFO (Routing)Net5.rte[4].routing: towards address 1 gateIndex is 1
INFO (Routing)Net5.rte[4].routing: towards address 2 gateIndex is 1
INFO (Routing)Net5.rte[4].routing: towards address 3 gateIndex is 0
Initializing module Net5.rte[4].queue[0], stage 0
Initializing module Net5.rte[4].queue[1], stage 0
Net5 #0: Net5
Msg stats: 5 scheduled / 19 existing / 19 created

1 / 4 - What?

How do we use OMNeT++?

1. Network Topology specification
   - modules, submodules
   - connections
   - channel properties
2. Simulation configuration
   - actual values to parameters
3. Network behavior specification
   - what modules actually *do*

in C++

How do we use OMNeT++?

1. Network Topology specification
   - modules, submodules
   - connections
   - channel properties
2. Simulation configuration
   - actual values to parameters
3. Network behavior specification
   - what modules actually *do*

in C++

…but also in Python

2 / 4 - Why?

C++	Python
focused on performance and resource efficiency	focused on readability
C-like syntax	simpler syntax
compiled	interpreted
low level	high level
static and explicit types	dynamic and implicit types
manual memory management	automatic memory management

Performance vs. productivity.

Focusing on the smallest details vs. focusing on the problem you’re trying to solve.

Knowledge and familiarity with Python is a given amongst most 3rd year students (not true for C++).

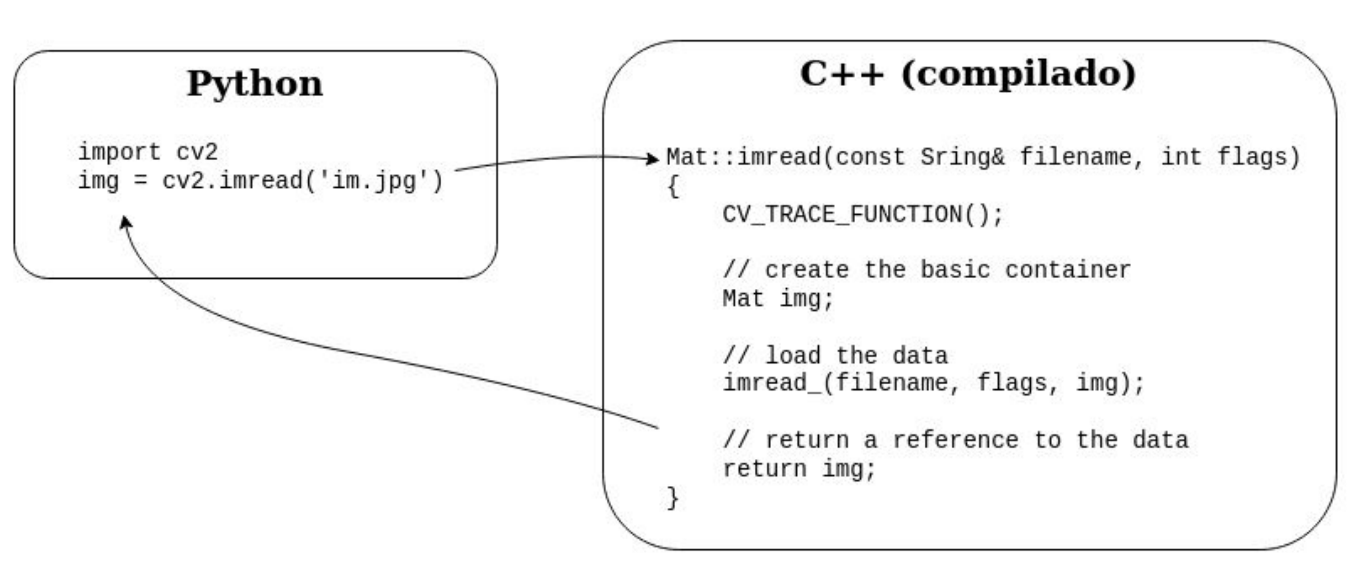
3 / 4 - How?

Controlled environment, repeatable processes

- OMNeT++ 5.5.1
- Linux 1902cc2cffcf 5.3.0-29-generic
- Ubuntu 18-10 (Cosmic)
- g++ (Ubuntu 8.3.0-6ubuntu1 18.10.1) 8.3.0
- GNU Make 4.2.1
- Python 3.6.8
- pybind11 2.4.3

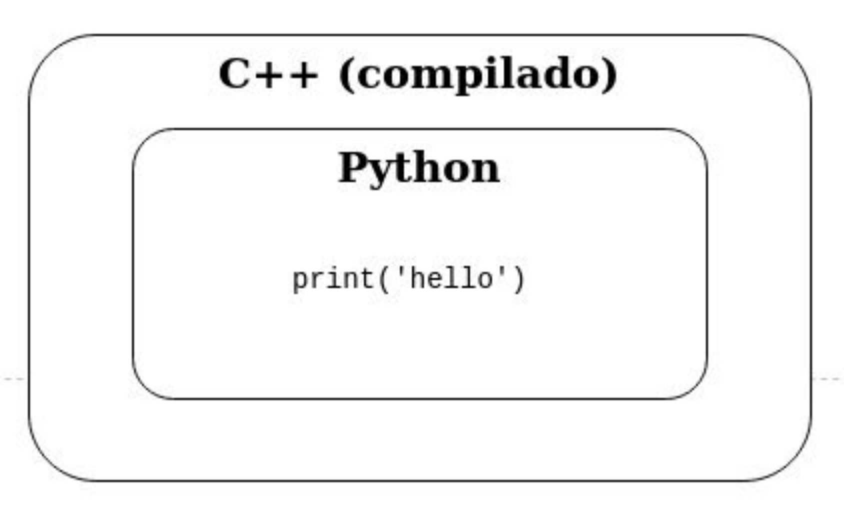
Understanding the two way interactions between C++ and Python

Extend the interpreter

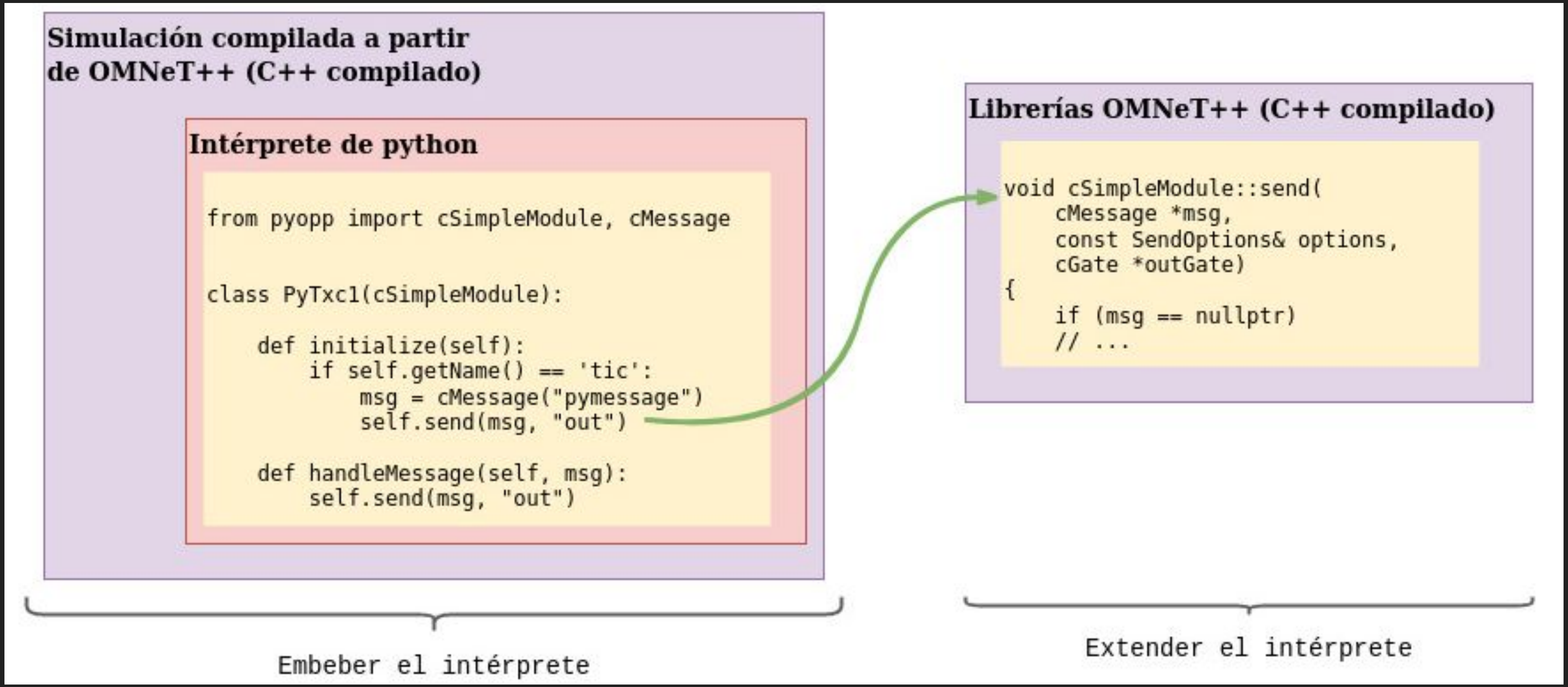


Understanding the two way interactions between C++ and Python

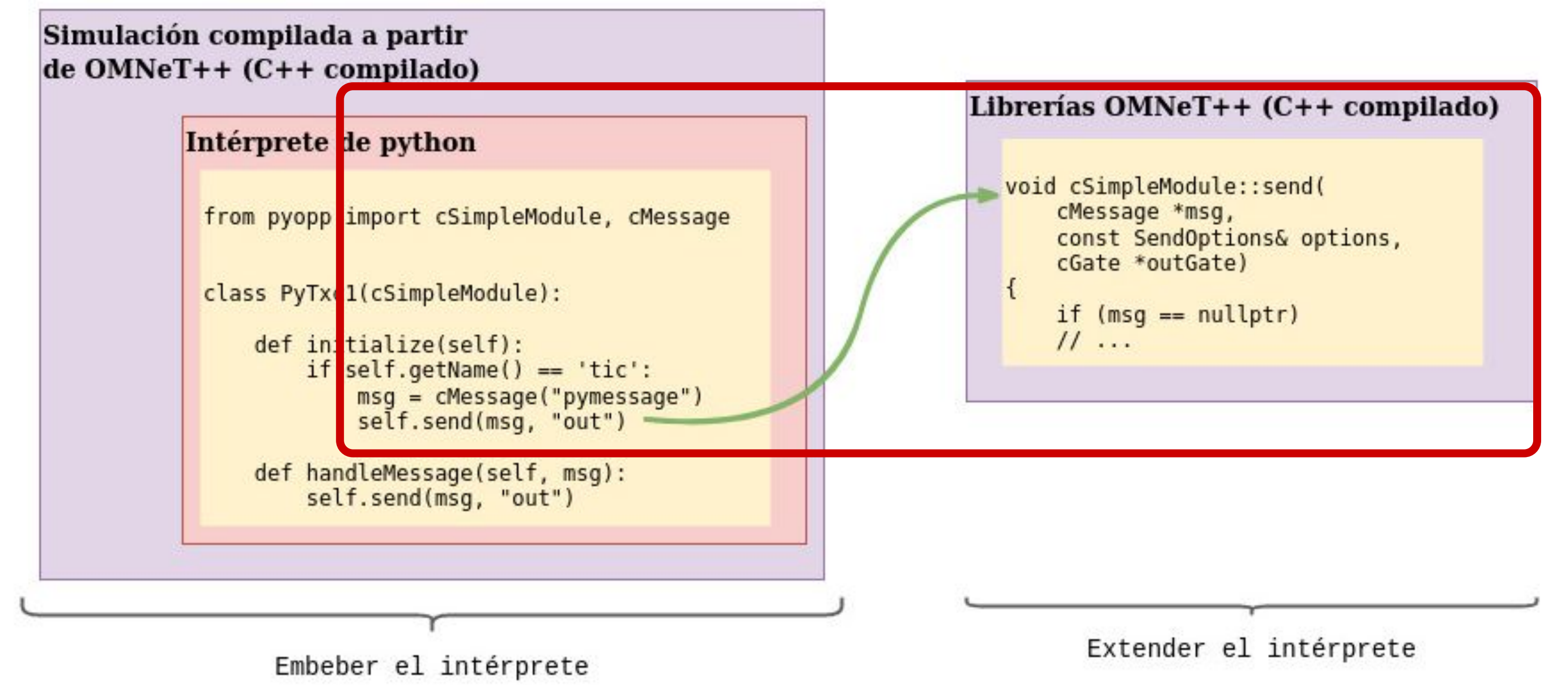
Embed the interpreter



We need both (extend and embed)



Subgoal number 1: extend the interpreter



Subgoal number 1: extend the interpreter

```c
int square(int x)
{
    return x * x;
}
```

```c
#define PY_SSIZE_T_CLEAN
#include <Python.h>

static PyObject *square(PyObject *self, PyObject *args) {
  int input;
  if (!PyArg_ParseTuple(args, "i", &input)) {
        return NULL;
  }

  return PyLong_FromLong((long)input * (long)input);
}

static PyMethodDef example_methods[] = {
        {"square", square, METH_VARARGS, "Returns a square of an integer"},
        {NULL, NULL, 0, NULL},
};

static struct PyModuleDef example_definition = {
        ,
        "example",
        "example module containing square() function",
        -1,
        example_methods,
};

PyMODINIT_FUNC PyInit_example(void) {
  PyObject *m = PyModule_Create(&example_definition);
  return m;
}
```

Subgoal number 1: extend the interpreter

pybind11

Creating bindings for a custom type

Let's now look at a more complex example where we'll create bindings for a custom C++ data structure named `Pet`. Its definition is given below:

```
struct Pet {
    Pet(const std::string &name) : name(name) { }
    void setName(const std::string &name_) { name = name_; }
    const std::string &getName() const { return name; }

    std::string name;
};
```

The binding code for `Pet` looks as follows:

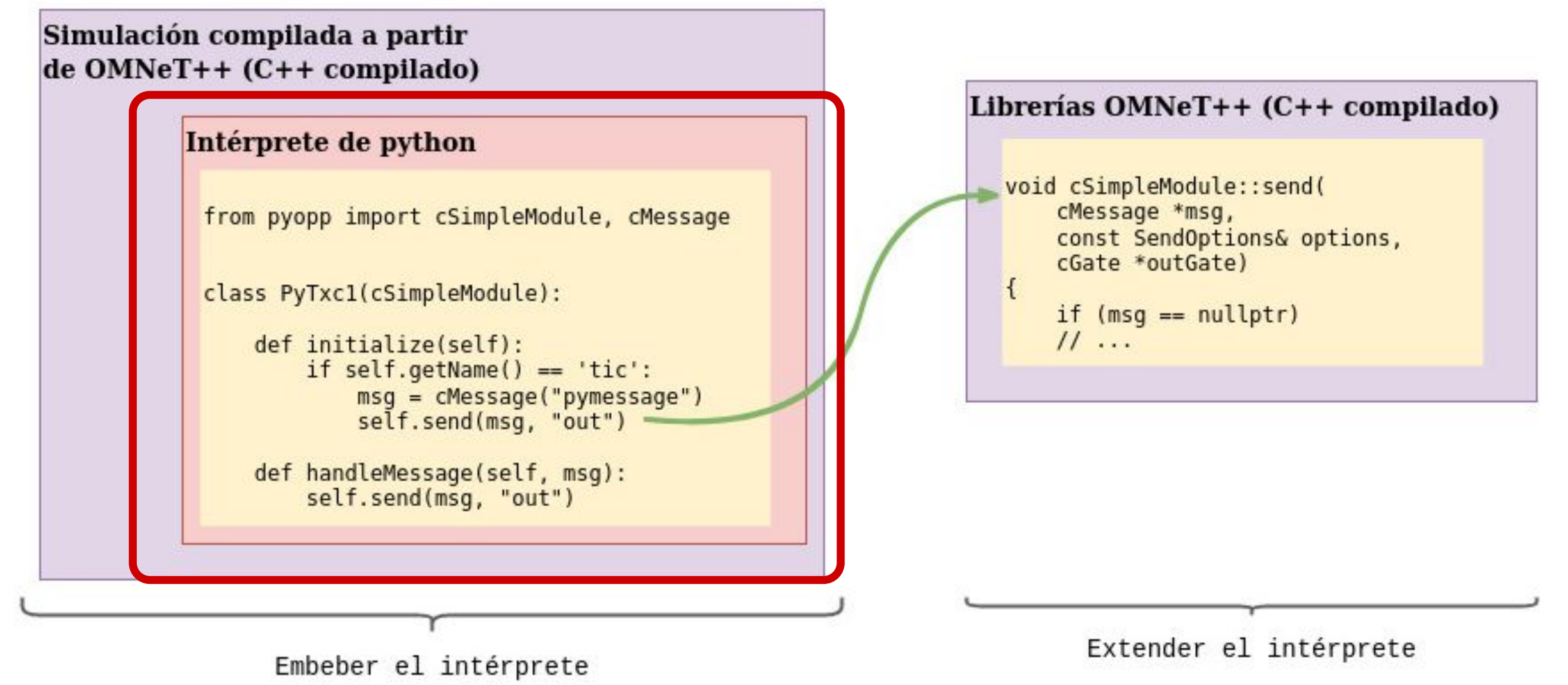
```
#include <pybind11/pybind11.h>

namespace py = pybind11;

PYBIND11_MODULE(example, m) {
    py::class_<Pet>(m, "Pet")
        .def(py::init<const std::string &>())
        .def("setName", &Pet::setName)
        .def("getName", &Pet::getName);
}
```

We achieved the “extending” part

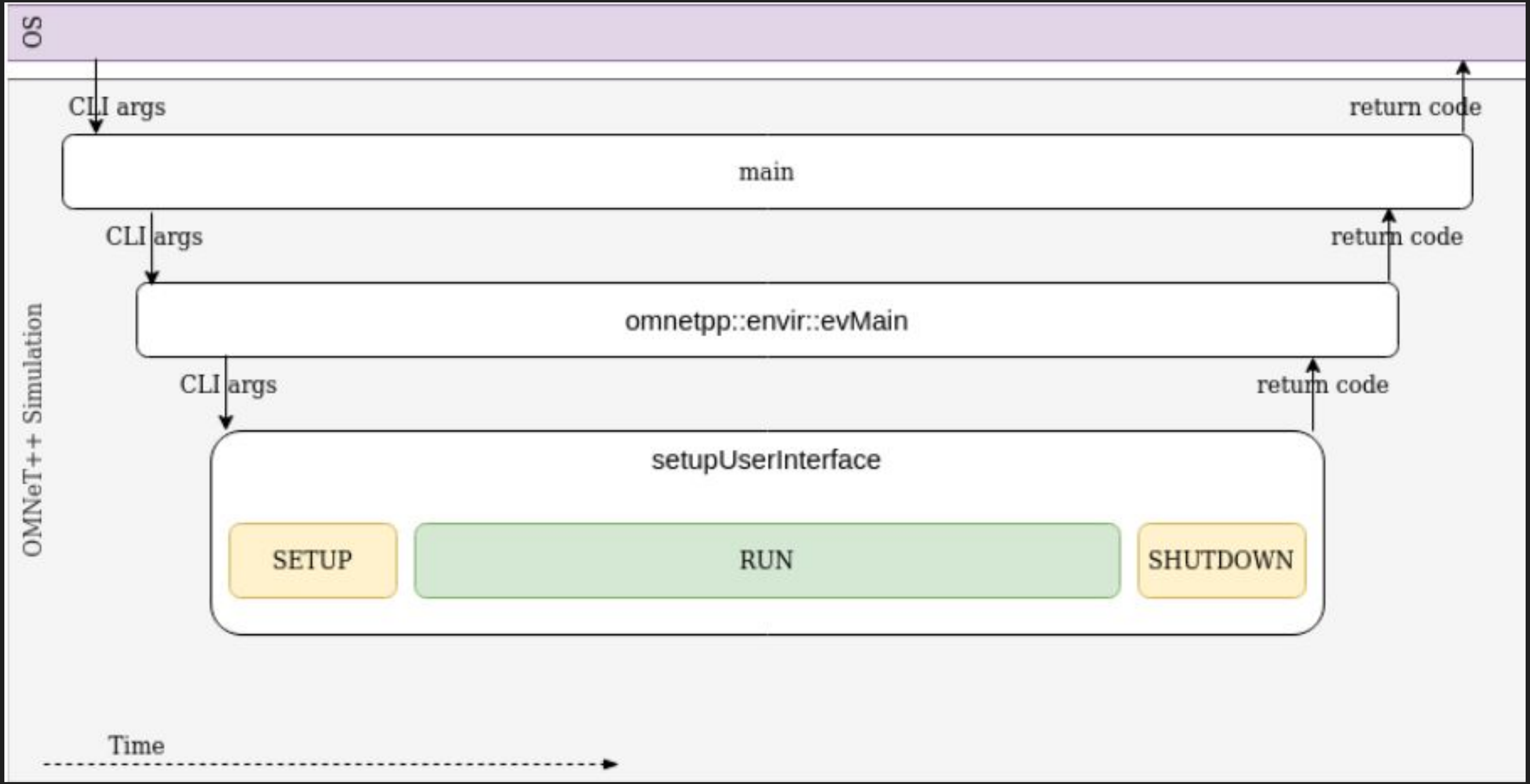
```
# python3
>>> from pyopp import cSimpleModule, cMessage
>>> sm = cSimpleModule()
>>> sm.handleMessage(cMessage('hello'))
<!> Error during startup/shutdown: Global simtime_t variable found, with value 0. Global
↪   simtime_t variables are forbidden, because scale exponent is not yet known at the time they
↪   are initialized. Please use double or const_simtime_t instead. Aborting.
```

[Source]

Subgoal number 2: embed the interpreter



Lifecycle of an OMNeT++ simulation binary



CodeFragments

A linked list of function pointers to be executed either at STARTUP or SHUTDOWN stages

```
/**
 * @brief Allows code fragments to be collected in global scope which will
 * then be executed from main() right after program startup. This is
 * used by in \opp for building global registration lists of
 * module types, network types, etc. Registration lists in fact
 * are a simple substitute for Java's Class.forName() method...
 *
 * @hideinitializer
 */
#define EXECUTE_ON_STARTUP(CODE)  \
  namespace { \
    void __ONSTARTUP_FUNC() {CODE;} \
```

cGlobalRegistrationList classes

List of all classes that can be instantiated. Given a module name the framework can lookup a cObjectFactory object which has functions for creating objects of that type of module.

cGlobalRegistrationList classes

List of all classes that can be instantiated. Given a module name the framework can lookup a cObjectFactory object which has functions for creating objects of that type of module.

```
network Tictoc1
{
    submodules:
        tic: Txc1
        toc: Txc1
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

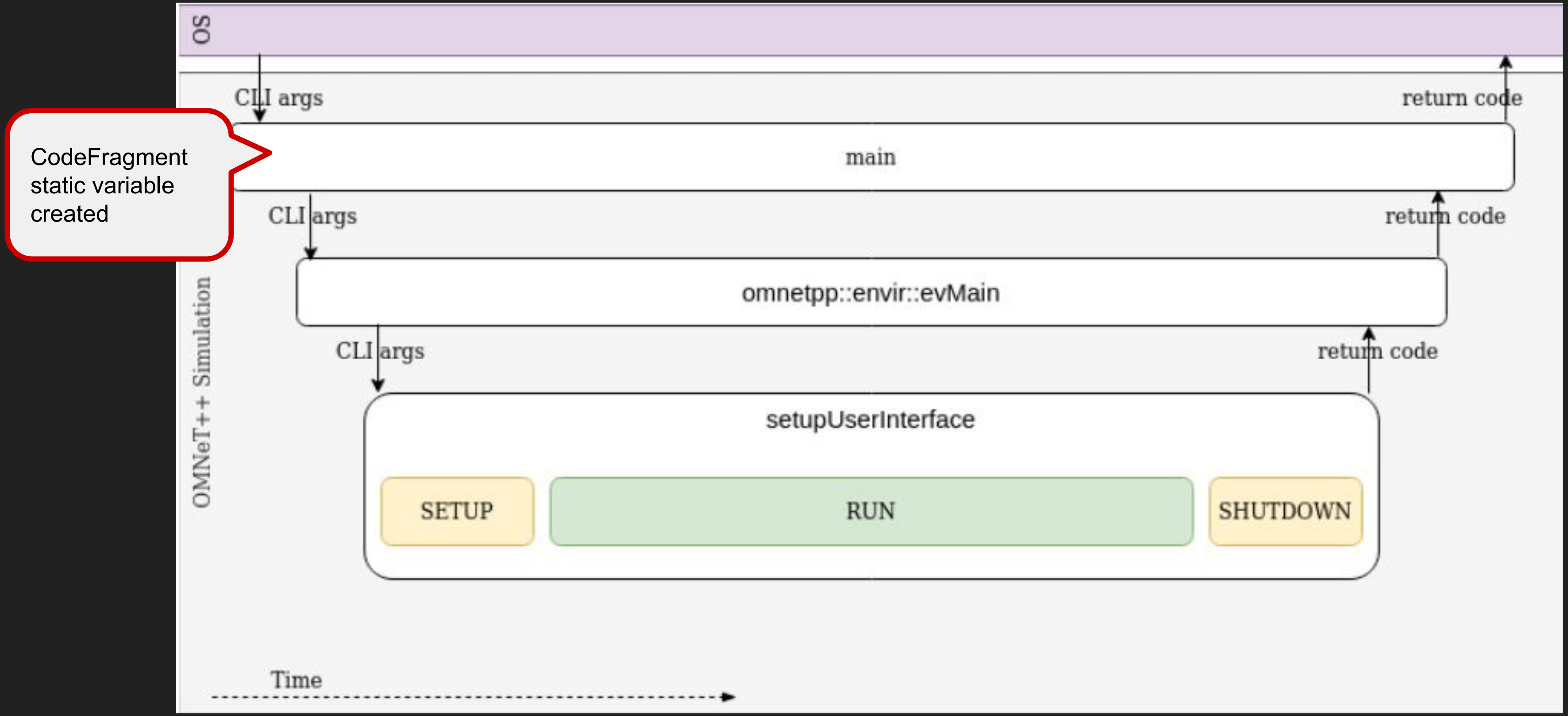
Define_Module macro

```
static void *__castfunc_13(omnetpp::cObject *obj)
{
    return (void*)dynamic_cast<Txc1*>(obj);
}
```

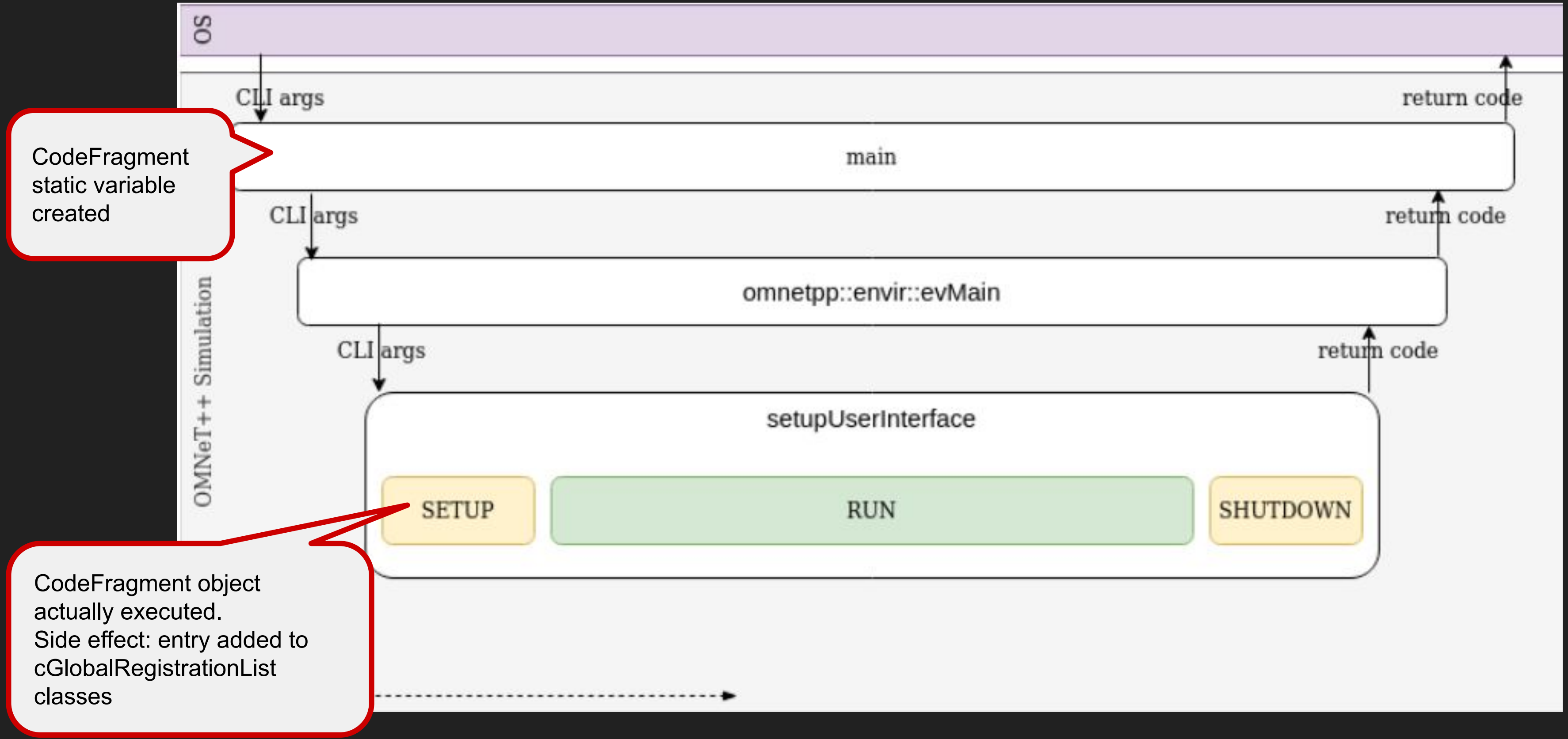
```
static omnetpp::cObject *__factoryfunc_13()
{
    omnetpp::cModule *ret = new Txc1;
    return ret;
}
```

```
namespace
{
    void __onstartup_func_13()
    {
        omnetpp::classes.getInstance()->add(
            new omnetpp::cObjectFactory(
                omnetpp::opp_typename(typeid(Txc1)),
                __factoryfunc_13,
                __castfunc_13,
                "module"));
    }

    static omnetpp::CodeFragments __onstartup_obj_13(
        __onstartup_func_13,
        omnetpp::CodeFragments::STARTUP);
 }
```

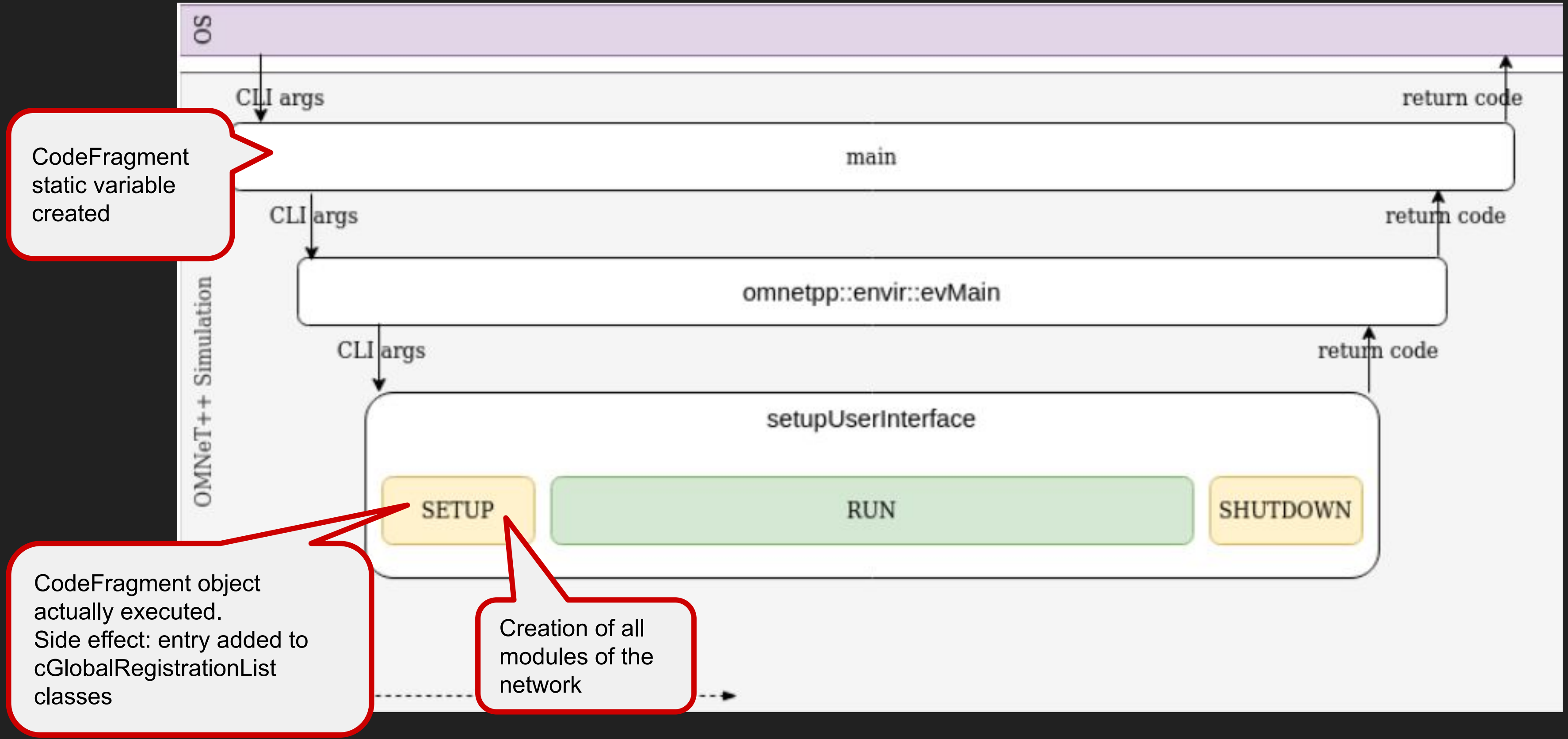
Lifecycle of an OMNeT++ simulation binary



Lifecycle of an OMNeT++ simulation binary



Lifecycle of an OMNeT++ simulation binary



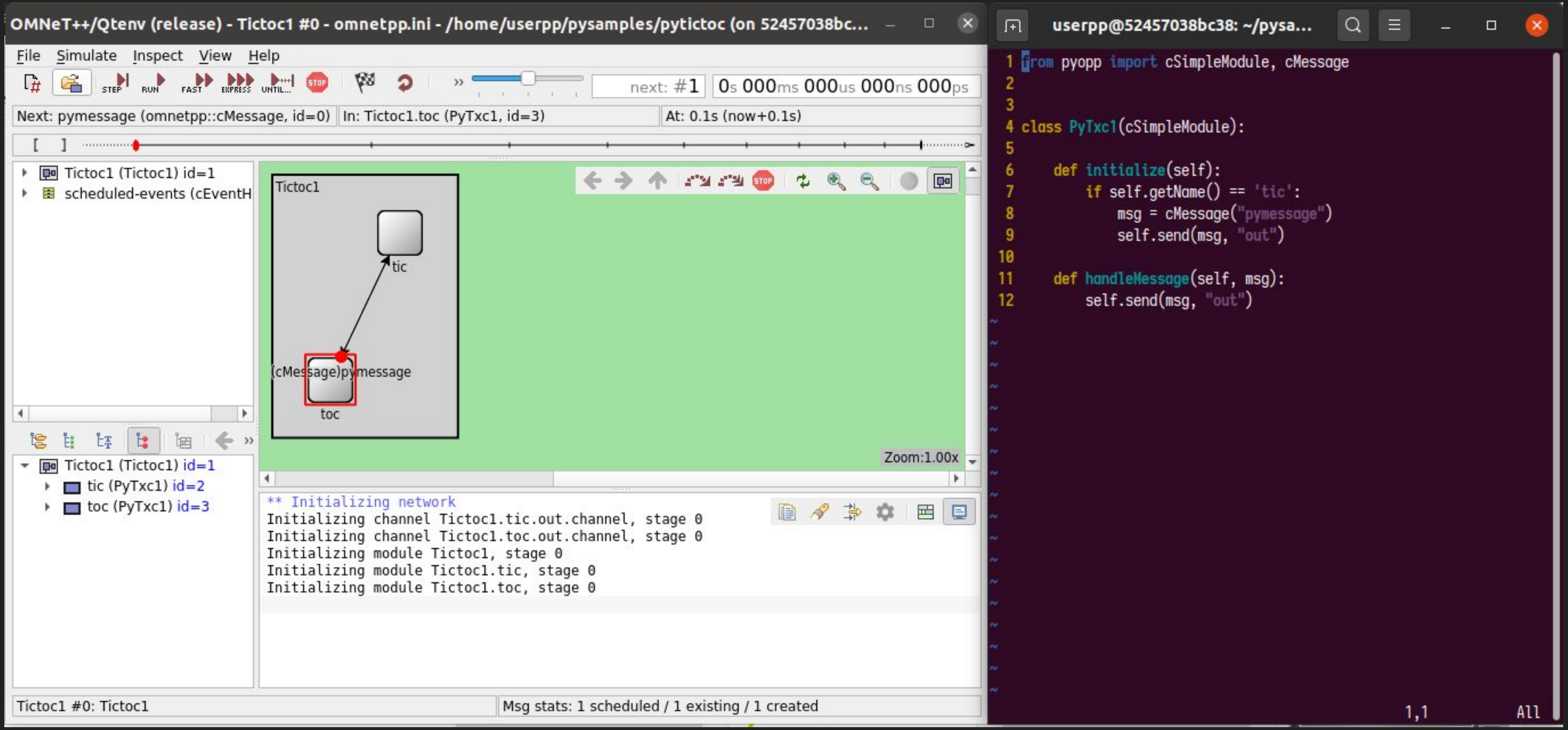
Define_Python_Module macro

```
#include <omnetpp/omnetpy.h>

Define_Python_Module("txc", "PyTxc");
```

- Mimics define_Module
    - Defines a factory function
    - Defines a cast function
    - Creates a function that registers these to OMNeT++ as the factories for “PyTxc” modules
    - Creates static instance of CodeFragments to be run during STARTUP
- Makes sure a Python interpreter is alive inside the simulation

[Source]

Mission accomplished!

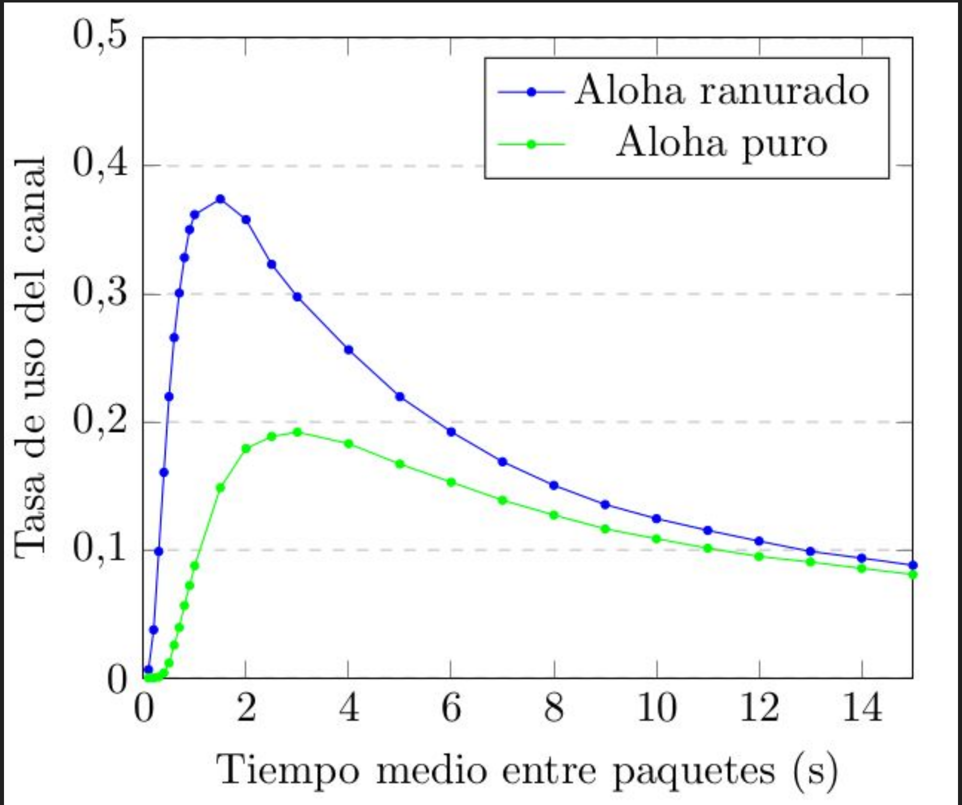


4 / 4 - Results

Almost all examples from OMNeT++ rewritten in Python

- pyaloha
- pycanvas
- pycqn
- pydyna **XX (delete module)**
- pyfifo
- pyhistograms
- pyhypercube
- pyrouting
- pytictoc

Model comparison



- omnetpy and OMNeT++ models throw exactly the same results
- omnetpy models are less performant

Strengths:

- the approach works (with some gotchas, that may or may not be overcome)
- absolutely no intervention of the OMNeT++ source code
- suddenly, models have access the vast ecosystem of Python libraries

Weaknesses:

- memory management subtleties
- project is “stuck” on not so new versions
- binding generation by hand, as needed

Open challenges:

- Automate the binding generation
- Thorough testing
- Solve memory management during module deletion
- Enable python debugging

Wanna try it out?

https://github.com/mmodenesi/omnetpy/

Thanks!