Steering & Control

- Flexible and save Settings -

Requirements

- Individual Processes / modules needs to define options

- Documentation, Reasonable defaults, no hidden settings, maybe expert settings?

- Independent of library's through abstraction

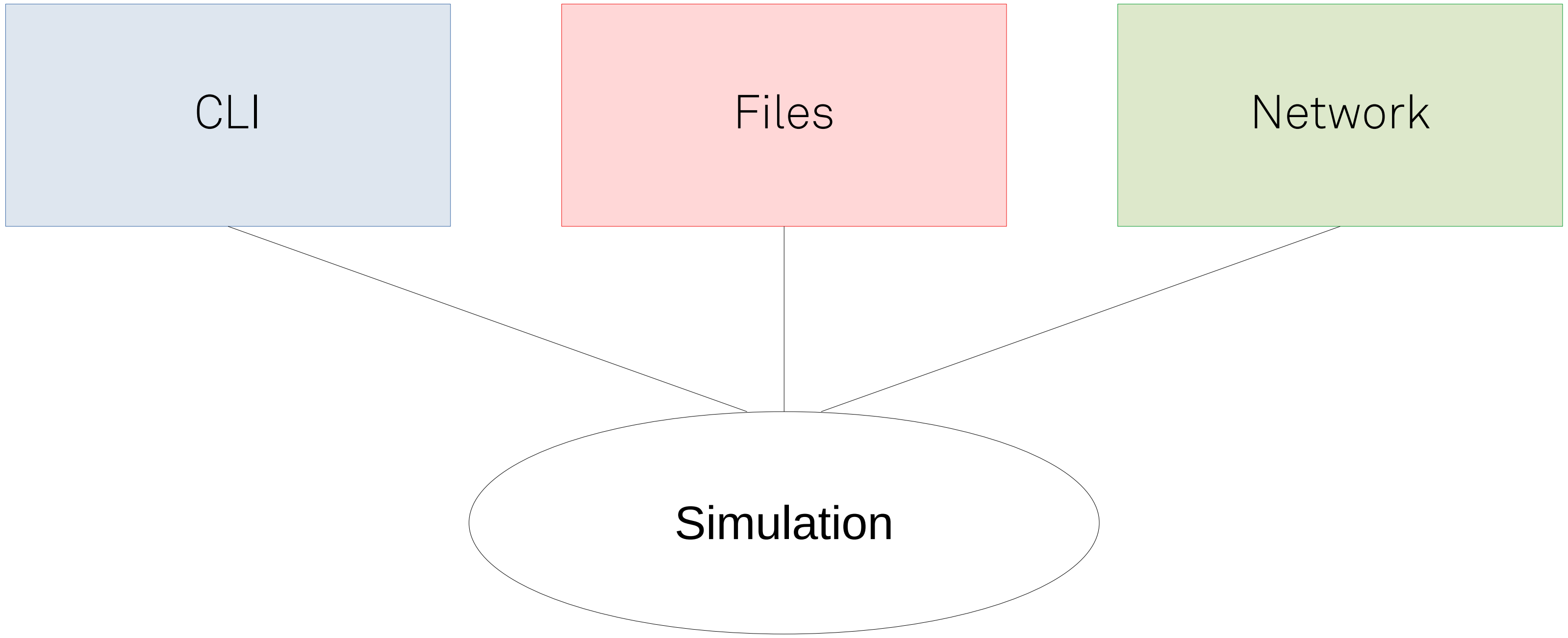
- Tree structure, but allow aliases for important settings

- Runtime structure, compile time will be overkill

- Callbacks should be possible, but polling the default

- Support of reflection

The Situation



Overwrites file input

Basic steering with default values

Sharing / distribution of settings

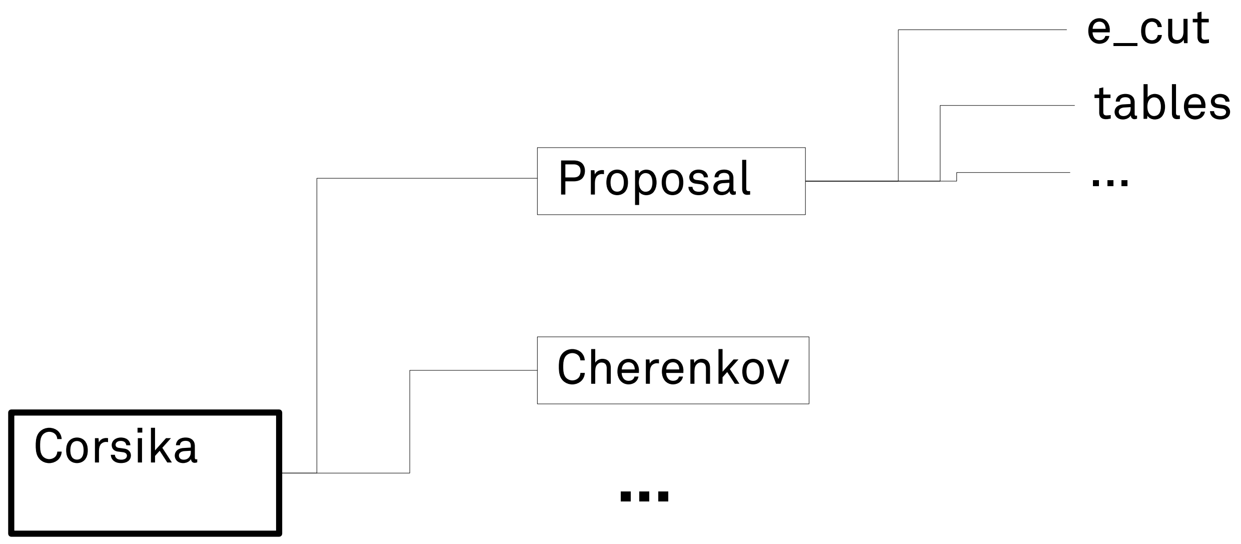
Exkurs: ?? Network ??

- Large scale parallel execution of very high energy shower → controller and worker

- Easier deployment on cluster infrastructure

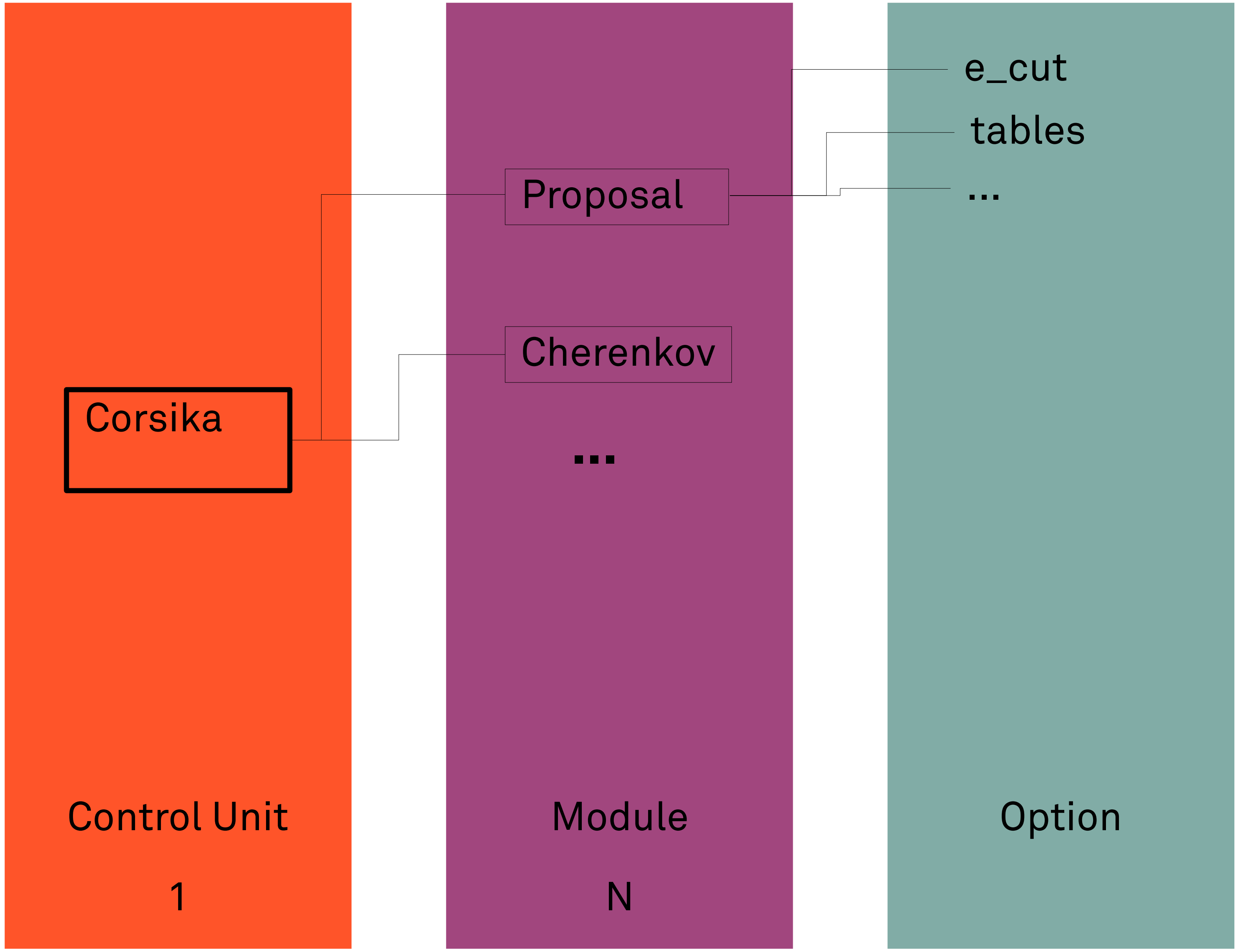
- Dynamic processing on queue based infrastructure with appearing / disappearing worker

Tree Patching

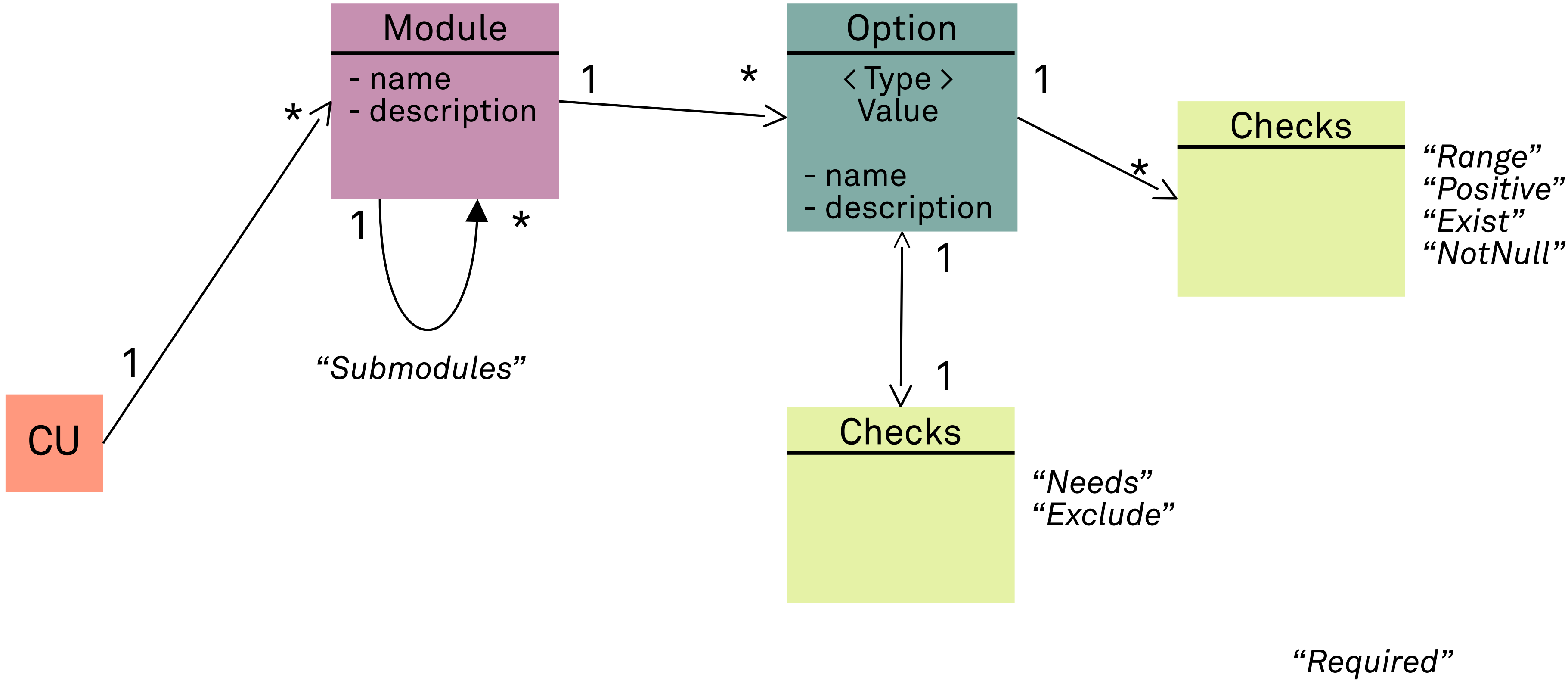


e.g.: corsika.proposal.e_cut

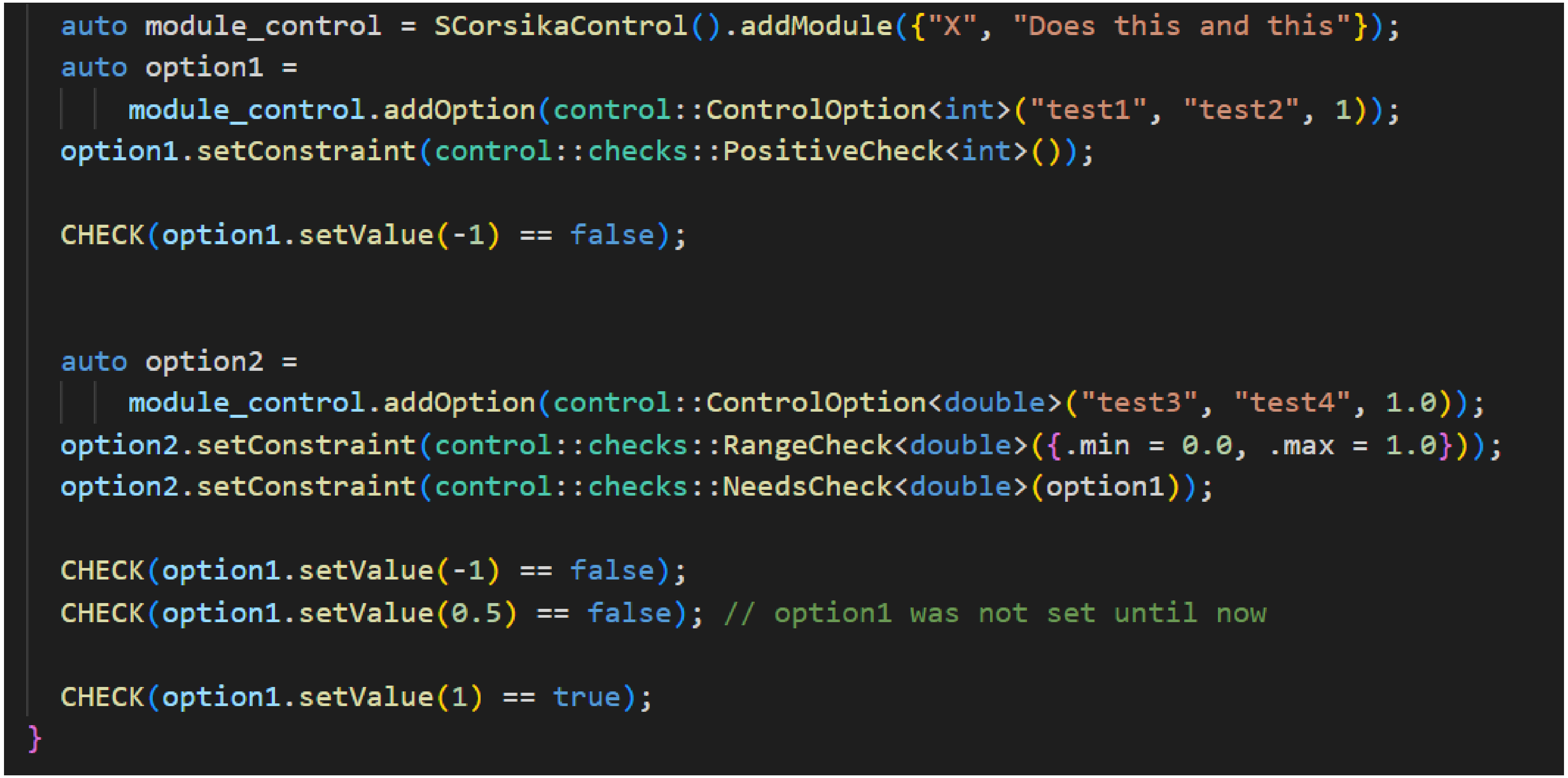
Nomenclature



Fundamental Implementation

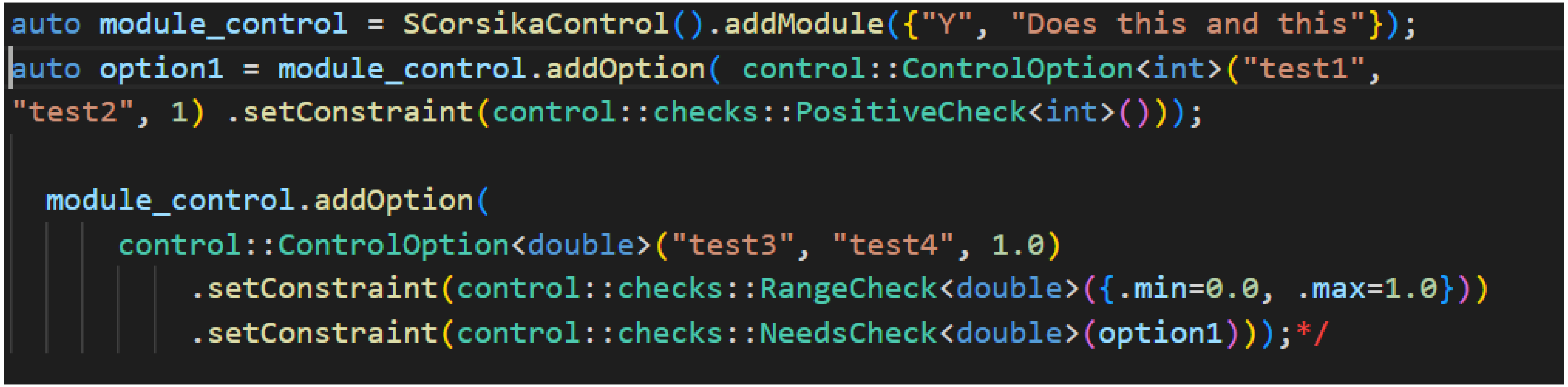


Syntax 1



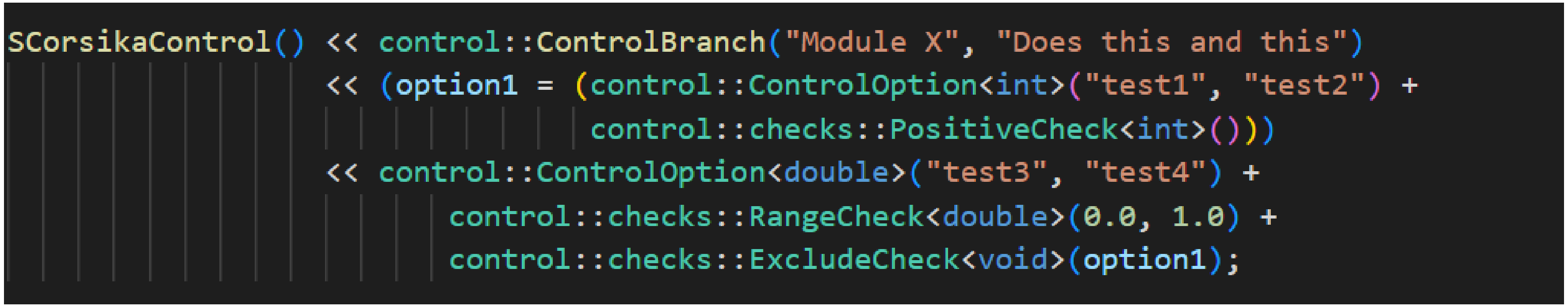
```
auto module_control = SCorsikaControl().addModule({"X", "Does this and this"});
auto option1 =
        module_control.addOption(control::ControlOption<int>("test1", "test2", 1));
option1.setConstraint(control::checks::PositiveCheck<int>());

CHECK(option1.setValue(-1) == false);


auto option2 =
        module_control.addOption(control::ControlOption<double>("test3", "test4", 1.0));
option2.setConstraint(control::checks::RangeCheck<double>({.min = 0.0, .max = 1.0}));
option2.setConstraint(control::checks::NeedsCheck<double>(option1));

CHECK(option1.setValue(-1) == false);
CHECK(option1.setValue(0.5) == false); // option1 was not set until now

CHECK(option1.setValue(1) == true);
}
```

Syntax 2



```
auto module_control = SCorsikaControl().addModule({"Y", "Does this and this"});
auto option1 = module_control.addOption( control::ControlOption<int>("test1",
"test2", 1) .setConstraint(control::checks::PositiveCheck<int>()));

  module_control.addOption(
      control::ControlOption<double>("test3", "test4", 1.0)
          .setConstraint(control::checks::RangeCheck<double>({.min=0.0, .max=1.0}))
          .setConstraint(control::checks::NeedsCheck<double>(option1)));*/
```

Syntax 3



```
SCorsikaControl() << control::ControlBranch("Module X", "Does this and this")
                  << (option1 = (control::ControlOption<int>("test1", "test2") +
                                control::checks::PositiveCheck<int>()))
                  << control::ControlOption<double>("test3", "test4") +
                        control::checks::RangeCheck<double>(0.0, 1.0) +
                        control::checks::ExcludeCheck<void>(option1);
```

Go all the way!

Not implemented as of now

Overload operator,() and operator[] to create a syntax similar to python