# acts Getting Started

supported by

EP-R&D
Programme on Technologies for Future Experiments

iris hep

Bundesministerium für Bildung und Forschung

cooperations

HSF
HEP Software Foundation

intel

CERN openlab

A. Salzburger (CERN) for the ACTS project

@SaltyBurger

# Getting started with the Core library

**Core**

acts-developers@cern.ch

CPU multi-threaded library of
tracking reconstruction components

**R&D1**

acts-parallelization@cern.ch

CPU/GPU “single source” demonstrator
re-implementing the main Core chain

**R&D2**

acts-machinelearning@cern.ch

Machine learning and ML assisted
modules for track reconstruction

# Getting started with the Core library

**Core**

acts-developers@cern.ch

CPU multi-threaded library of
tracking reconstruction components

| Core | Fatras | Plugins | Examples |
|---|---|---|---|
| Library code | Fast simulation | Plugin code | Examples |

# Getting started with the Core library

**Core**

acts-developers@cern.ch

CPU multi-threaded library of
tracking reconstruction components

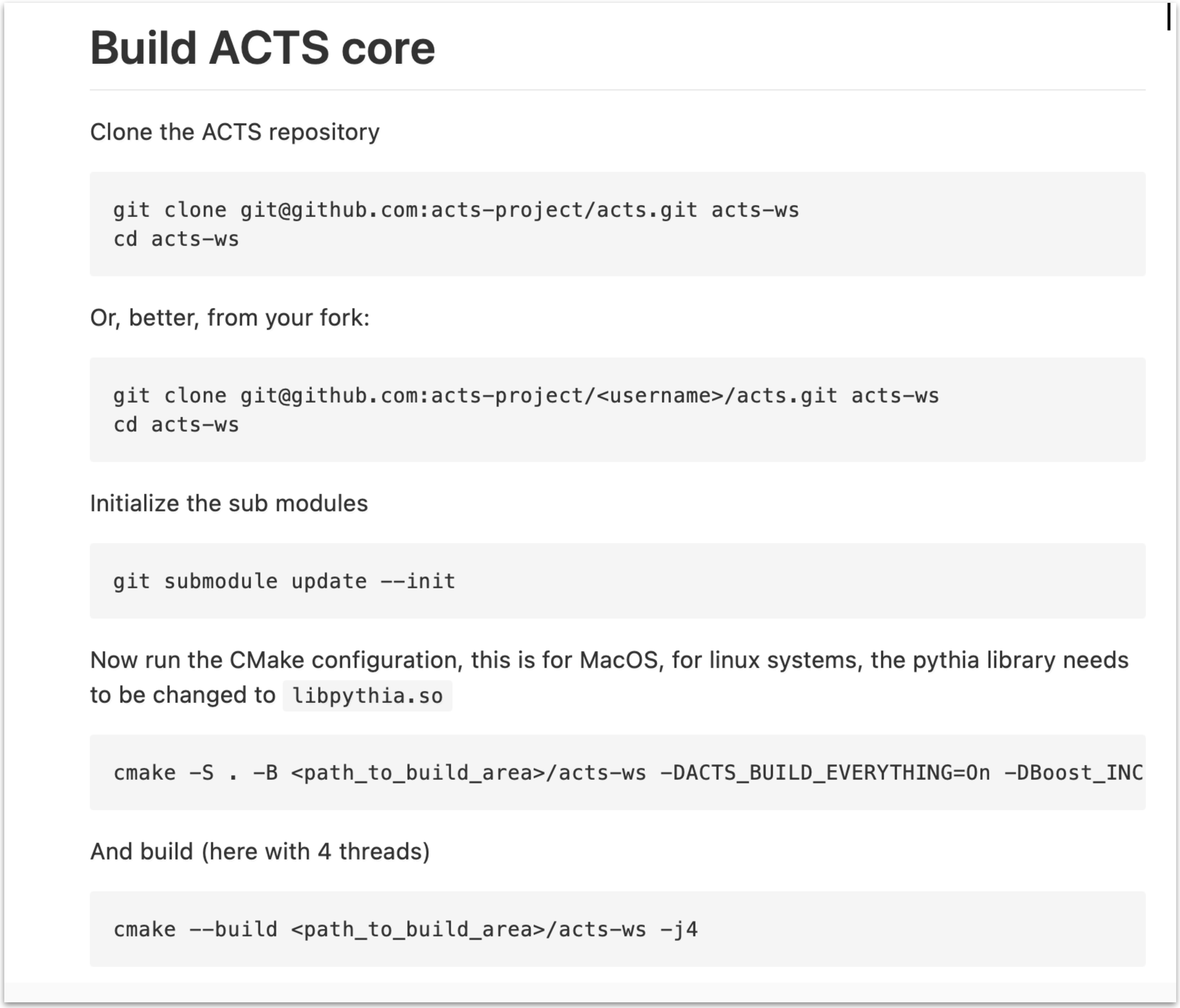
| | Core | Fatras | Plugins | Examples |
|---|---|---|---|---|
| | Library code | Fast simulation | Plugin code | Examples |
| Dependencies | Eigen<br>BOOST | Geant4 | | ROOT<br>Pythia8, HepMC<br>Geant4<br>DD4hep, EDM4hep, PODIO |

# Getting started with the Core library: externals & building

Several options to work with ACTS

- build from externals and ACTS from scratch
    - Build externals
    - Build ACTS

- Use LCG or User container https://acts.readthedocs.io/en/latest/getting_started.html



## Build ACTS core

Clone the ACTS repository

```
git clone git@github.com:acts-project/acts.git acts-ws
cd acts-ws
```

Or, better, from your fork:

```
git clone git@github.com:acts-project/<username>/acts.git acts-ws
cd acts-ws
```

Initialize the sub modules

```
git submodule update --init
```

Now run the CMake configuration, this is for MacOS, for linux systems, the pythia library needs to be changed to `libpythia.so`

```
cmake -S . -B <path_to_build_area>/acts-ws -DACTS_BUILD_EVERYTHING=On -DBoost_INC
```

And build (here with 4 threads)

```
cmake --build <path_to_build_area>/acts-ws -j4
```

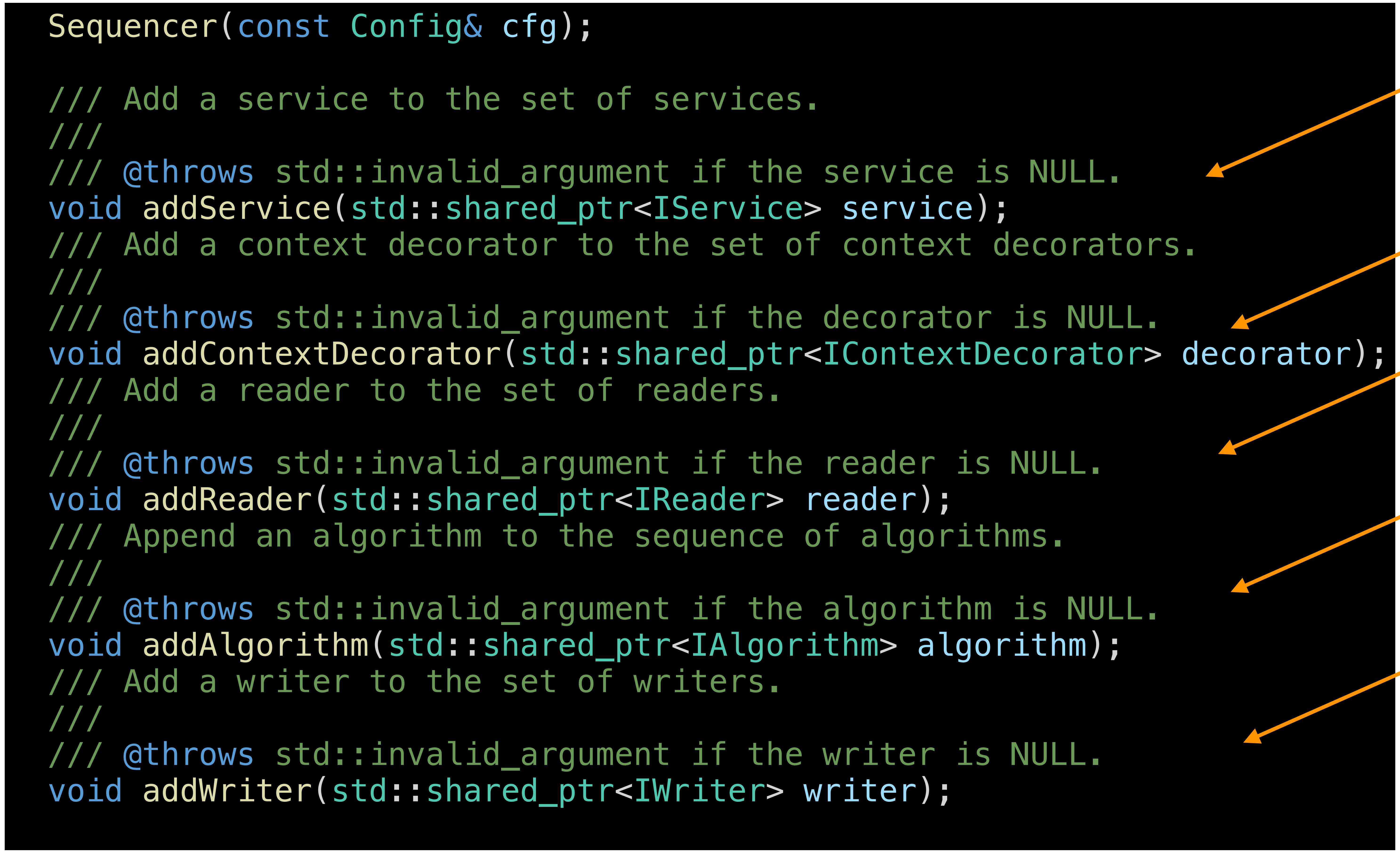
# Example framework: a playground - and not more

The example framework shipped with ACTS provides a showroom

- Event Data Model
  - A convenient container class (a candidate for being promoted to Core)
  - Simulated Hits (borrowed from ActsFatras)
  - Particles (same)
  - Measurement & track representation (from ActsCore)
- Framework
  - Sequencer & Algorithm interface
  - WhiteBoard for transient data transfer
  - I/O Functionality
  - Services
- Algorithms
  - Some sample algorithms
- Python
  - Python bindings
- Detectors
  - several detector examples

# Example framework: the Sequencer

Event parallel (using tbb) algorithm chain  executor:



```
Sequencer(const Config& cfg);

/// Add a service to the set of services.
///
/// @throws std::invalid_argument if the service is NULL.
void addService(std::shared_ptr<IService> service);
/// Add a context decorator to the set of context decorators.
///
/// @throws std::invalid_argument if the decorator is NULL.
void addContextDecorator(std::shared_ptr<IContextDecorator> decorator);
/// Add a reader to the set of readers.
///
/// @throws std::invalid_argument if the reader is NULL.
void addReader(std::shared_ptr<IReader> reader);
/// Append an algorithm to the sequence of algorithms.
///
/// @throws std::invalid_argument if the algorithm is NULL.
void addAlgorithm(std::shared_ptr<IAlgorithm> algorithm);
/// Add a writer to the set of writers.
///
/// @throws std::invalid_argument if the writer is NULL.
void addWriter(std::shared_ptr<IWriter> writer);
```

Job services, e.g. random numbers

Per event context attaching
(e.g. alignment)

Per event (pre-algorithms)
input reading*

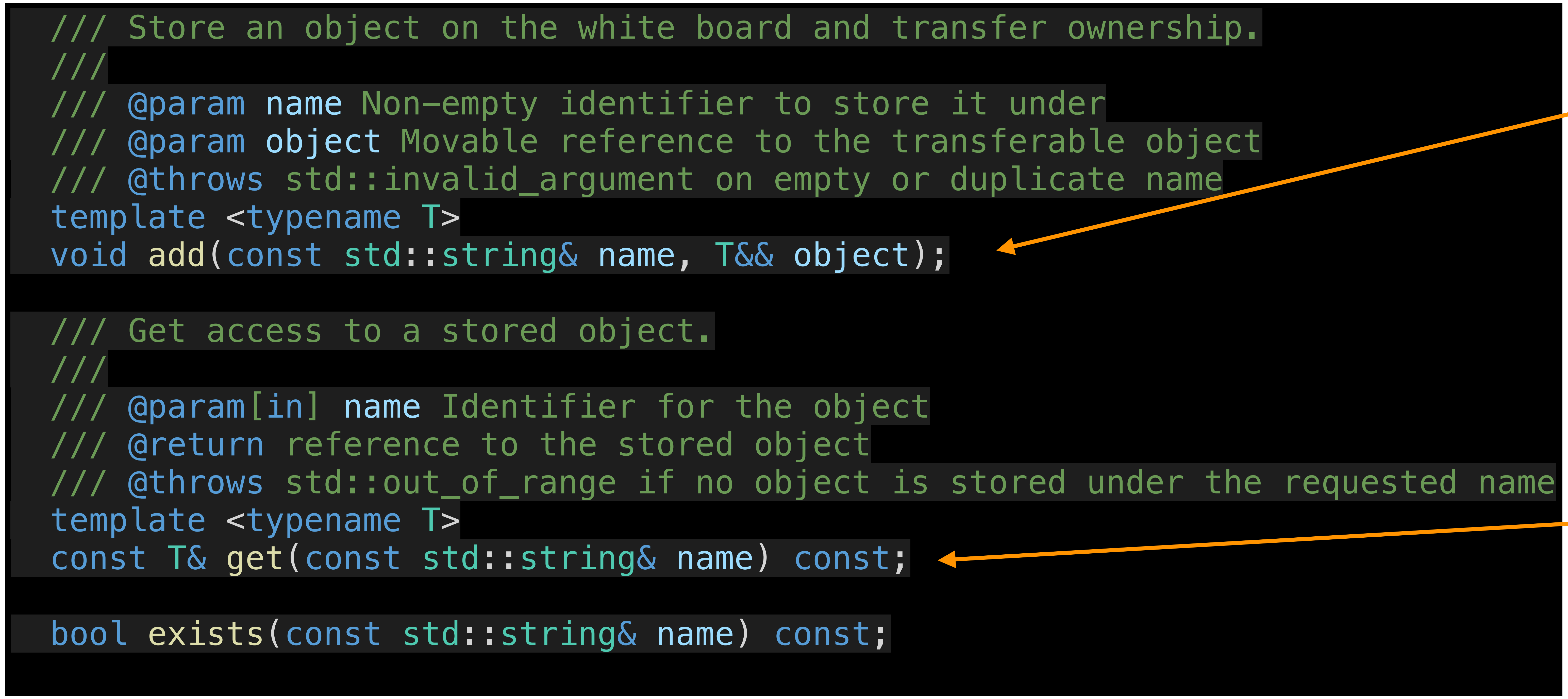
Per event algorithm chain*

Per event (post-algorithms)*
output writing

*readers, algorithms, writers are executed in the order in which they are added

# Example framework: the WhiteBoard

Per-event store for reading writing event data



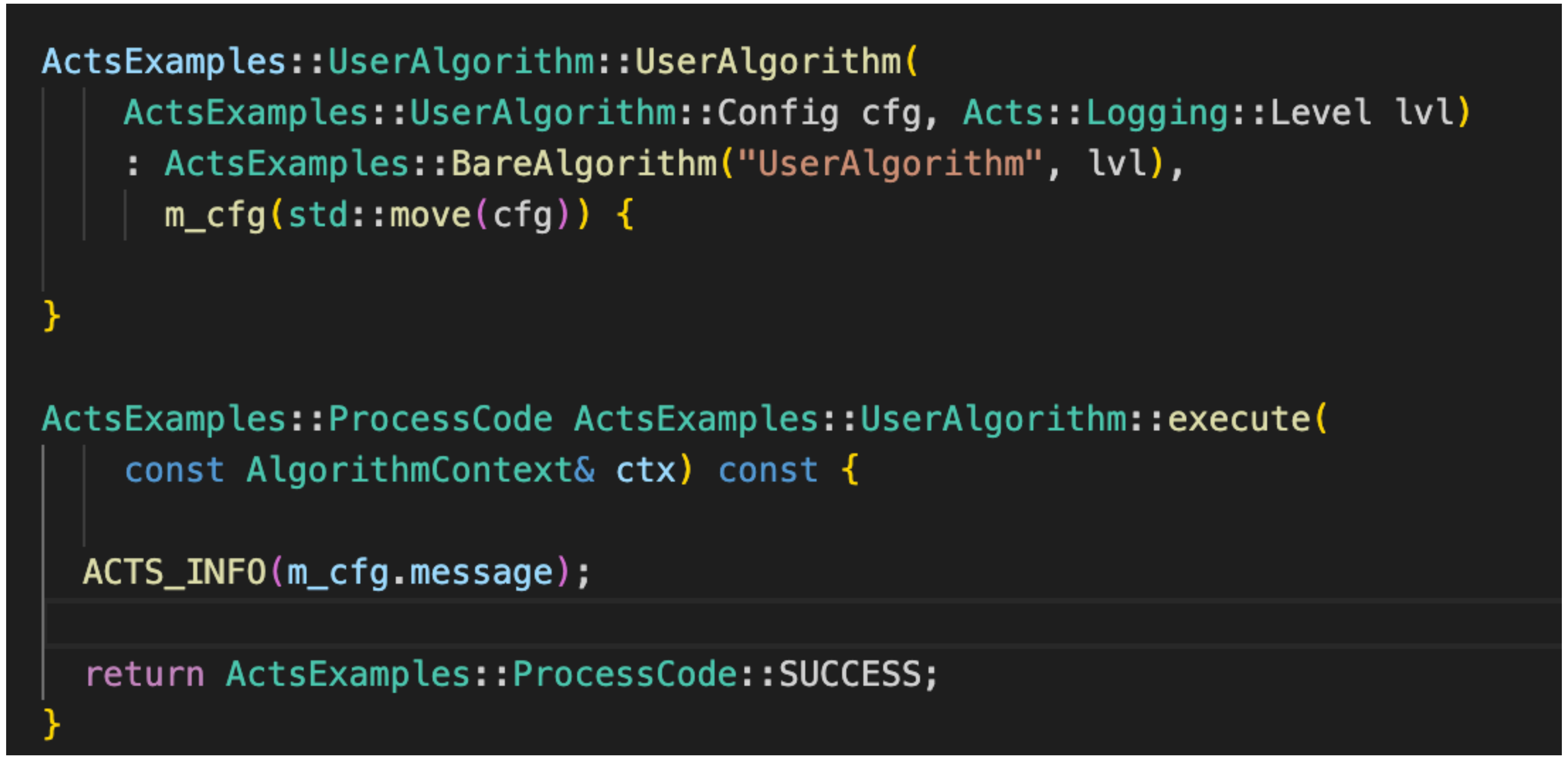
```
  /// Store an object on the white board and transfer ownership.
  ///
  /// @param name Non-empty identifier to store it under
  /// @param object Movable reference to the transferable object
  /// @throws std::invalid_argument on empty or duplicate name
  template <typename T>
  void add(const std::string& name, T&& object);

  /// Get access to a stored object.
  ///
  /// @param[in] name Identifier for the object
  /// @return reference to the stored object
  /// @throws std::out_of_range if no object is stored under the requested name
  template <typename T>
  const T& get(const std::string& name) const;

  bool exists(const std::string& name) const;
```

Adding data to the event store (readers e.g. fetch data from I/O and add them to the event store)

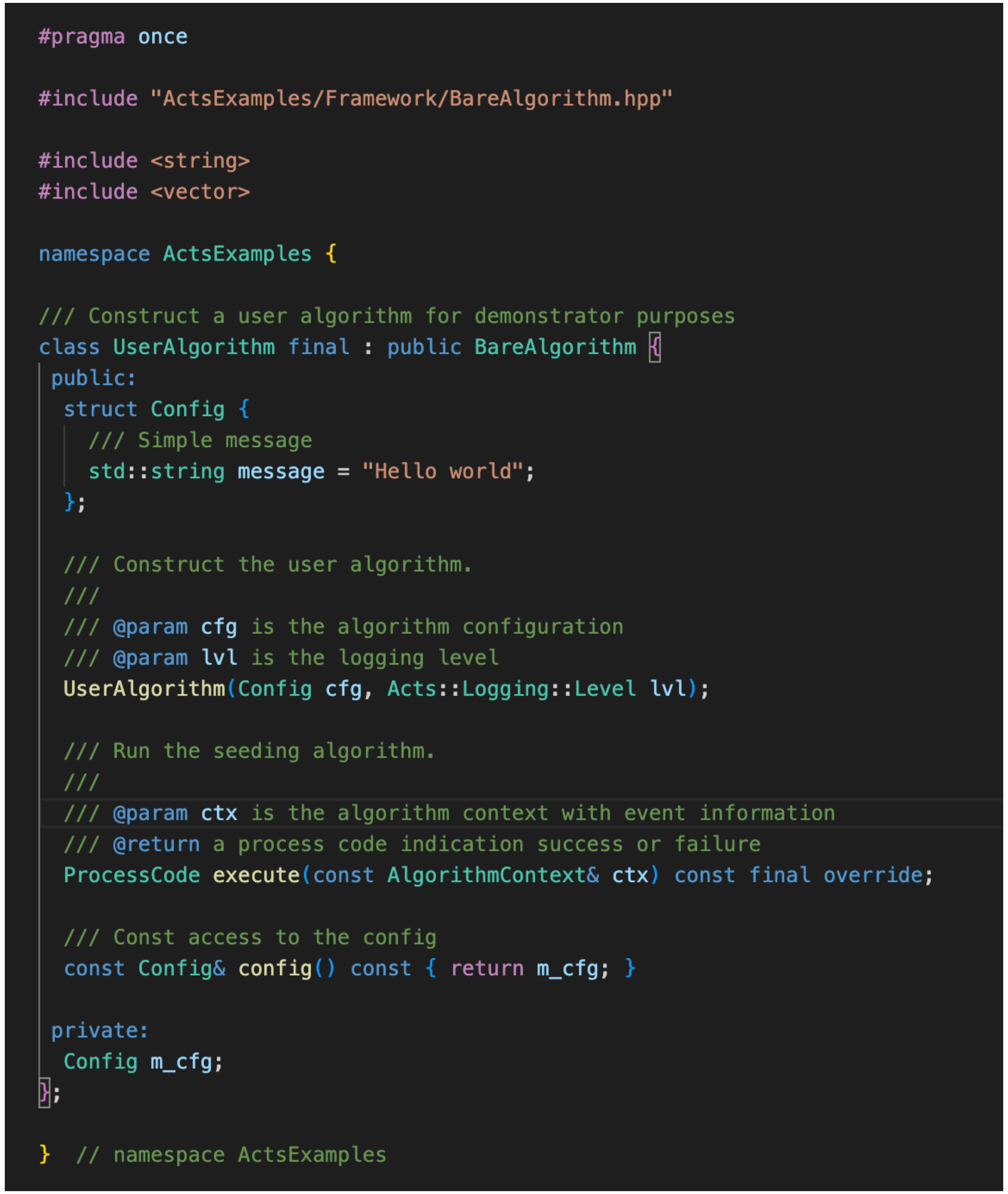
Getting data from event store

# Step1: adding a user algorithm

This adds a simple user algorithm which prints out some chosen/configured message



```
ActsExamples::UserAlgorithm::UserAlgorithm(
    ActsExamples::UserAlgorithm::Config cfg, Acts::Logging::Level lvl)
    : ActsExamples::BareAlgorithm("UserAlgorithm", lvl),
      m_cfg(std::move(cfg)) {

}

ActsExamples::ProcessCode ActsExamples::UserAlgorithm::execute(
    const AlgorithmContext& ctx) const {

  ACTS_INFO(m_cfg.message);

  return ActsExamples::ProcessCode::SUCCESS;
}
```

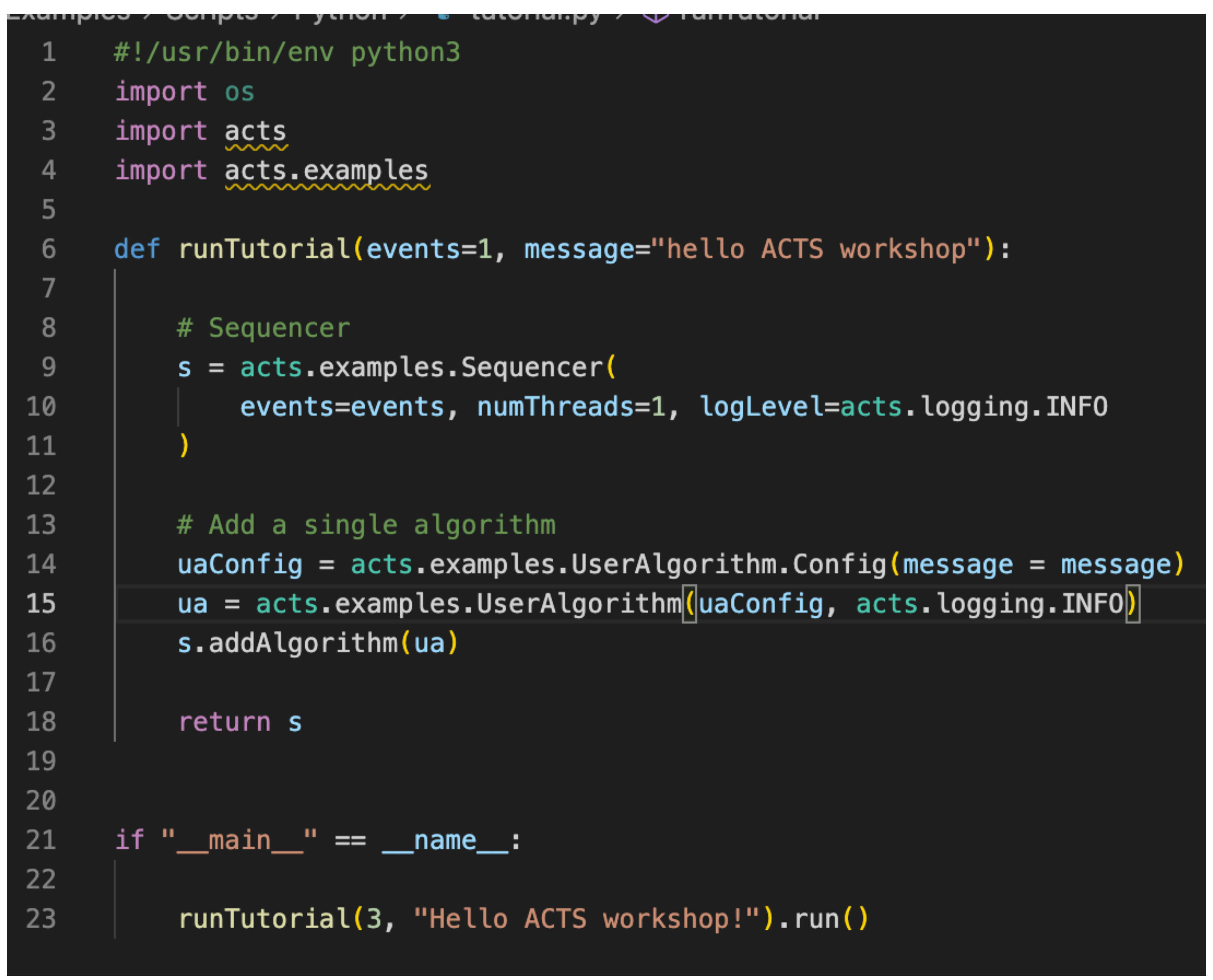


```
#pragma once

#include "ActsExamples/Framework/BareAlgorithm.hpp"

#include <string>
#include <vector>

namespace ActsExamples {

/// Construct a user algorithm for demonstrator purposes
class UserAlgorithm final : public BareAlgorithm {
 public:
  struct Config {
    /// Simple message
    std::string message = "Hello world";
  };

  /// Construct the user algorithm.
  ///
  /// @param cfg is the algorithm configuration
  /// @param lvl is the logging level
  UserAlgorithm(Config cfg, Acts::Logging::Level lvl);

  /// Run the seeding algorithm.
  ///
  /// @param ctx is the algorithm context with event information
  /// @return a process code indication success or failure
  ProcessCode execute(const AlgorithmContext& ctx) const final override;

  /// Const access to the config
  const Config& config() const { return m_cfg; }

 private:
  Config m_cfg;
};

}  // namespace ActsExamples
```

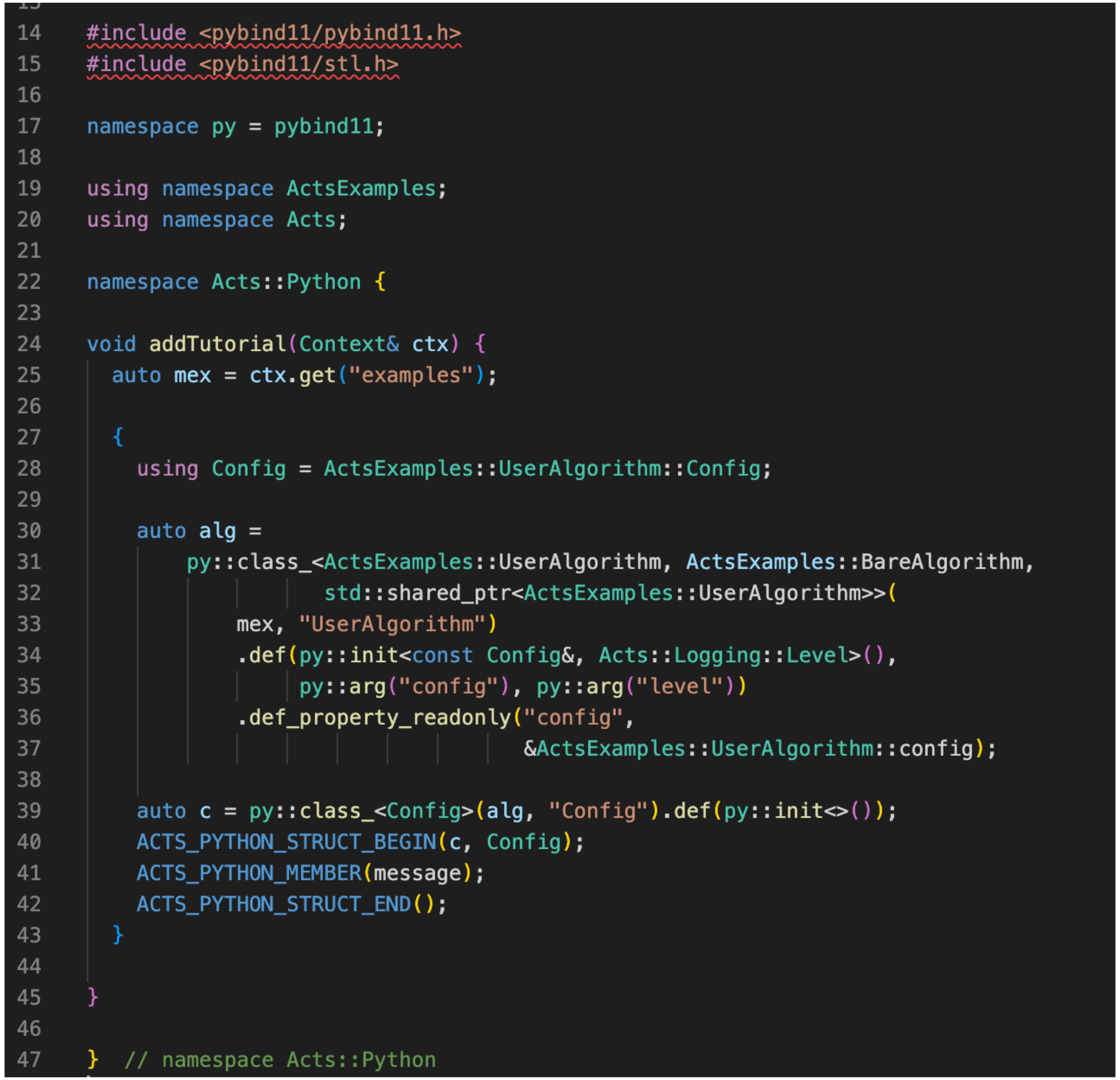
https://github.com/asalzburger/acts/tree/ws-add-user-algorithm

# Step 2: create python binding (and runnable script)

This adds python bindings to for the user algorithm and a runnable script



```python
#!/usr/bin/env python3
import os
import acts
import acts.examples

def runTutorial(events=1, message="hello ACTS workshop"):

    # Sequencer
    s = acts.examples.Sequencer(
        events=events, numThreads=1, logLevel=acts.logging.INFO
    )

    # Add a single algorithm
    uaConfig = acts.examples.UserAlgorithm.Config(message = message)
    ua = acts.examples.UserAlgorithm(uaConfig, acts.logging.INFO)
    s.addAlgorithm(ua)

    return s


if "__main__" == __name__:

    runTutorial(3, "Hello ACTS workshop!").run()
```



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

namespace py = pybind11;

using namespace ActsExamples;
using namespace Acts;

namespace Acts::Python {

void addTutorial(Context& ctx) {
  auto mex = ctx.get("examples");

  {
    using Config = ActsExamples::UserAlgorithm::Config;

    auto alg =
        py::class_<ActsExamples::UserAlgorithm, ActsExamples::BareAlgorithm,
                   std::shared_ptr<ActsExamples::UserAlgorithm>>(
            mex, "UserAlgorithm")
            .def(py::init<const Config&, Acts::Logging::Level>(),
                 py::arg("config"), py::arg("level"))
            .def_property_readonly("config",
                                   &ActsExamples::UserAlgorithm::config);

    auto c = py::class_<Config>(alg, "Config").def(py::init<>());
    ACTS_PYTHON_STRUCT_BEGIN(c, Config);
    ACTS_PYTHON_MEMBER(message);
    ACTS_PYTHON_STRUCT_END();
  }

}

}  // namespace Acts::Python
```

https://github.com/asalzburger/acts/tree/ws-add-user-algorithm-python-bindings

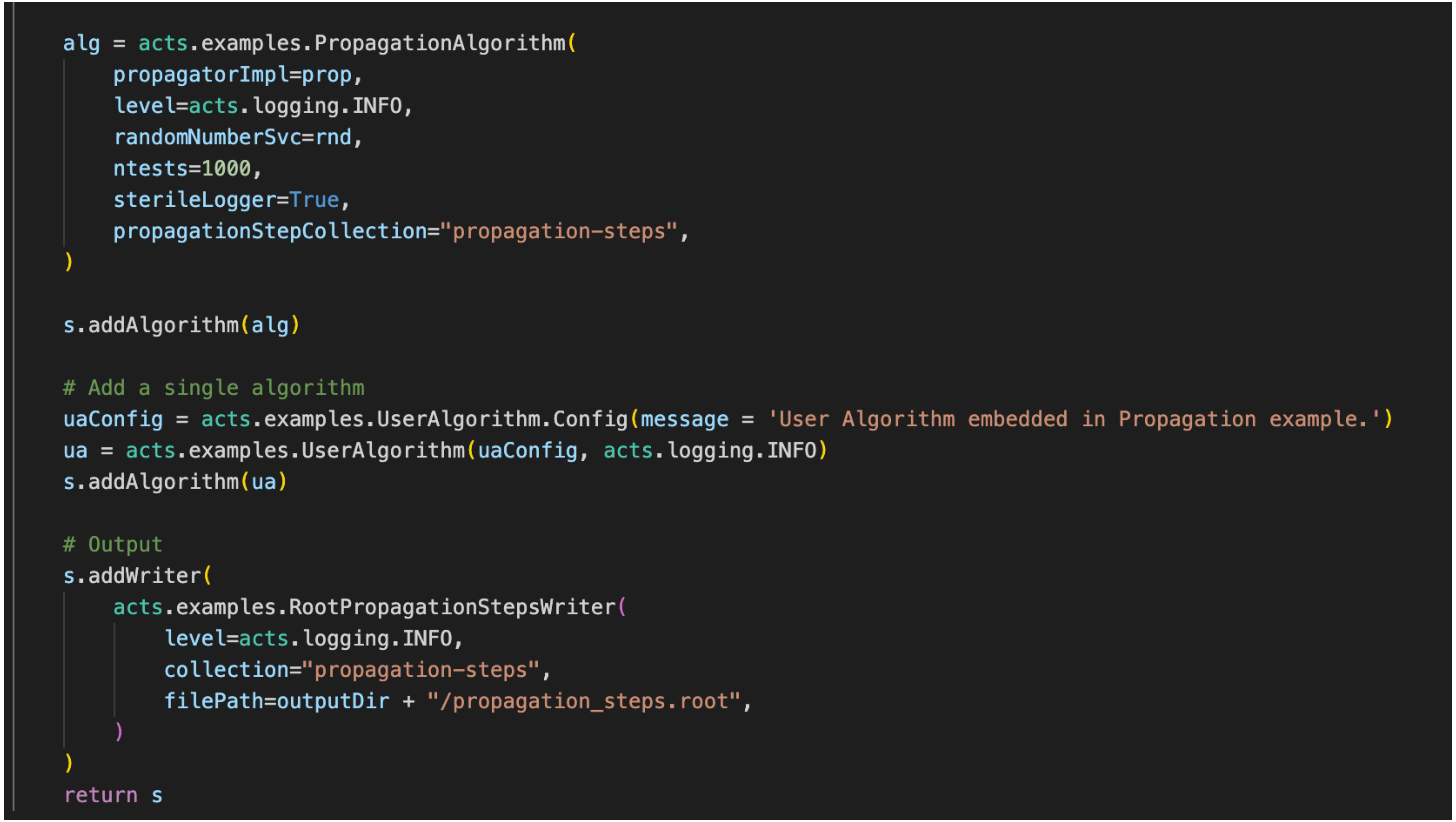
# Step 2: create python binding (and runnable script)

This adds python bindings to for the user algorithm and a runnable script

```
salzburg@andimacbookprom1 python % python3 <path_to_source>/acts-ws/Examples/Scripts/Python/tutorial.py
22:37:14    Sequencer      INFO      Create Sequencer (single-threaded)
22:37:14    Sequencer      INFO      Added algorithm 'UserAlgorithm'
22:37:14    Sequencer      INFO      Processing events [0, 3)
22:37:14    Sequencer      INFO      Starting event loop with 1 threads
22:37:14    Sequencer      INFO        0 services
22:37:14    Sequencer      INFO        0 context decorators
22:37:14    Sequencer      INFO        0 readers
22:37:14    Sequencer      INFO        1 algorithms
22:37:14    Sequencer      INFO        0 writers
22:37:14    UserAlgorith   INFO      Hello ACTS workshop!
22:37:14    Sequencer      INFO      finished event 0
22:37:14    UserAlgorith   INFO      Hello ACTS workshop!
22:37:14    Sequencer      INFO      finished event 1
22:37:14    UserAlgorith   INFO      Hello ACTS workshop!
22:37:14    Sequencer      INFO      finished event 2
22:37:14    Sequencer      INFO      Processed 3 events in 33.541000 us (wall clock)
22:37:14    Sequencer      INFO      Average time per event: 2.000000 us/event
```

**https://github.com/asalzburger/acts/tree/ws-add-user-algorithm-python-bindings**

# Step 3: embed the algorithm in another example



```python
alg = acts.examples.PropagationAlgorithm(
    propagatorImpl=prop,
    level=acts.logging.INFO,
    randomNumberSvc=rnd,
    ntests=1000,
    sterileLogger=True,
    propagationStepCollection="propagation-steps",
)

s.addAlgorithm(alg)

# Add a single algorithm
uaConfig = acts.examples.UserAlgorithm.Config(message = 'User Algorithm embedded in Propagation example.')
ua = acts.examples.UserAlgorithm(uaConfig, acts.logging.INFO)
s.addAlgorithm(ua)

# Output
s.addWriter(
    acts.examples.RootPropagationStepsWriter(
        level=acts.logging.INFO,
        collection="propagation-steps",
        filePath=outputDir + "/propagation_steps.root",
    )
)
return s
```

**https://github.com/asalzburger/acts/tree/ws-add-user-algorithm-python-bindings-embedded**

# Step 3: embed the algorithm in another example

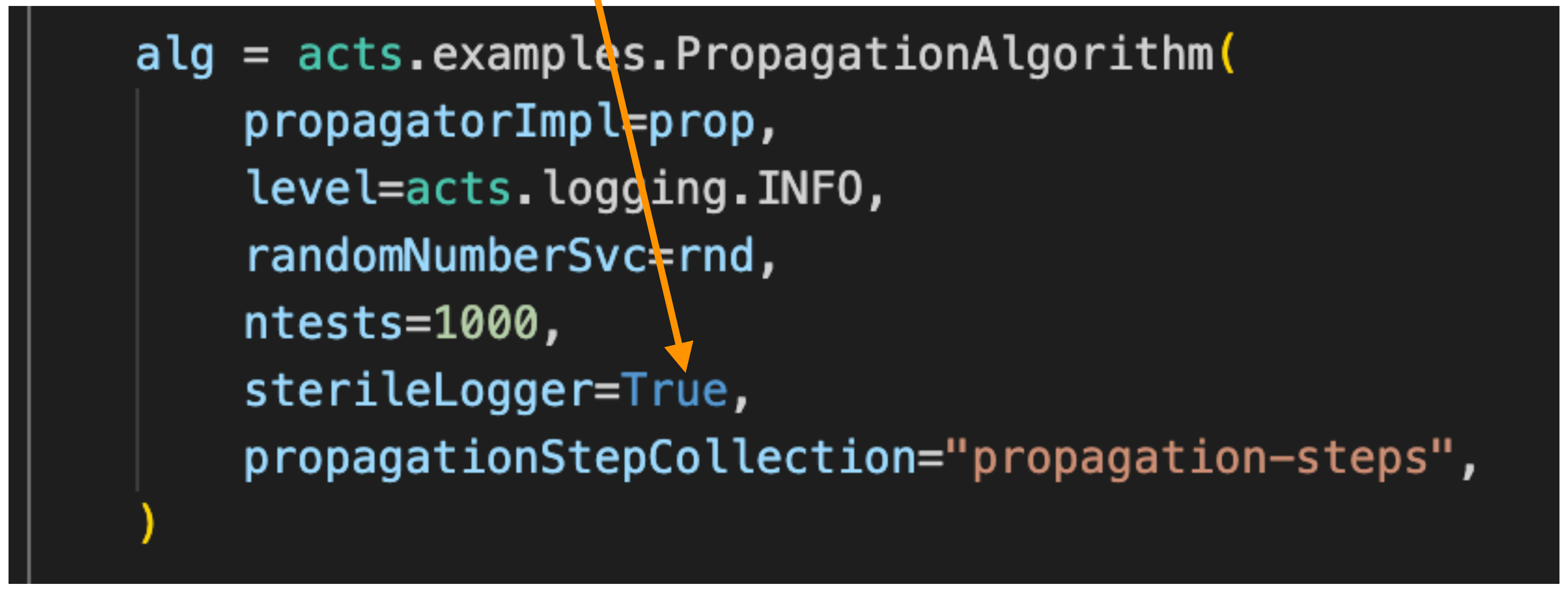
```
22:47:32    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
22:47:32    Sequencer      INFO      finished event 89
22:47:32    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
22:47:32    Sequencer      INFO      finished event 90
22:47:32    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
22:47:32    Sequencer      INFO      finished event 91
22:47:32    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
22:47:32    Sequencer      INFO      finished event 92
22:47:32    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
22:47:32    Sequencer      INFO      finished event 93
22:47:32    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
22:47:32    Sequencer      INFO      finished event 94
22:47:32    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
22:47:32    Sequencer      INFO      finished event 95
22:47:32    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
22:47:32    Sequencer      INFO      finished event 96
22:47:32    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
22:47:32    Sequencer      INFO      finished event 97
22:47:32    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
22:47:32    Sequencer      INFO      finished event 98
22:47:32    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
22:47:32    Sequencer      INFO      finished event 99
22:47:32    Sequencer      INFO      Processed 100 events in 2.356004 s (wall clock)
22:47:32    Sequencer      INFO      Average time per event: 23.429318 ms/event
```

**https://github.com/asalzburger/acts/tree/ws-add-user-algorithm-python-bindings-embedded**

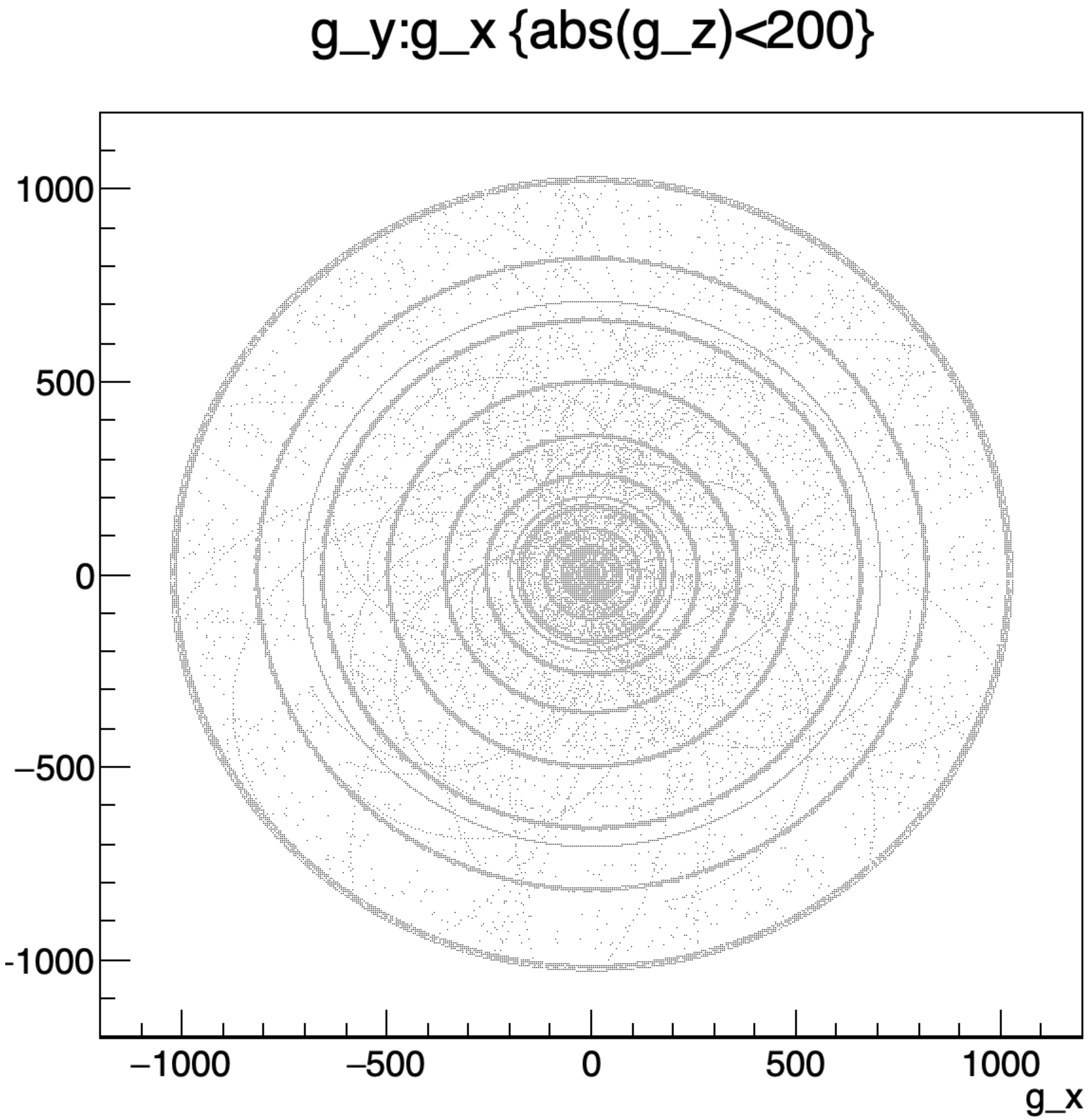
# Step 3: embed the algorithm in another example

Creates an output root file

Change to **False** here for a non-empty root file

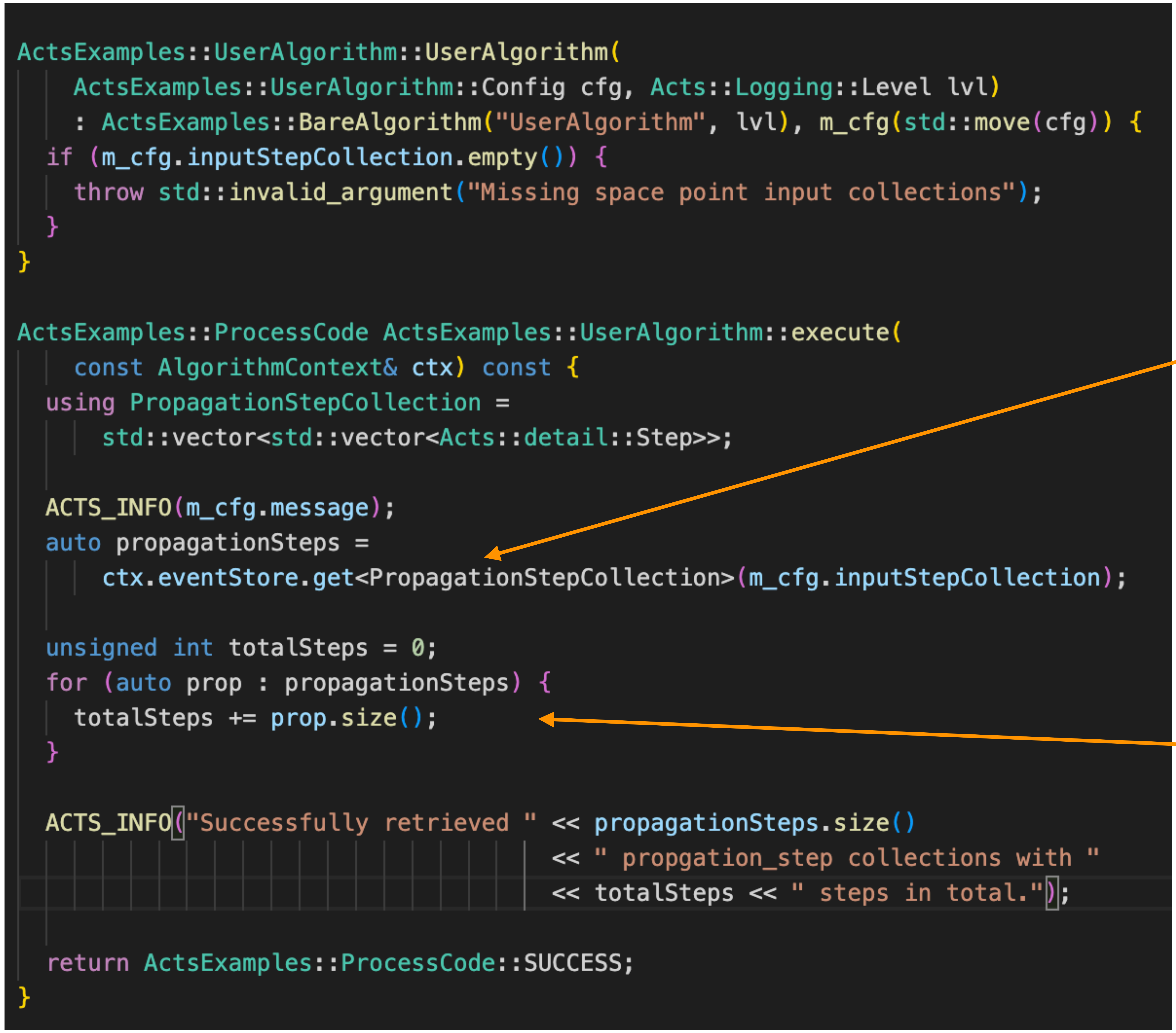


```
alg = acts.examples.PropagationAlgorithm(
    propagatorImpl=prop,
    level=acts.logging.INFO,
    randomNumberSvc=rnd,
    ntests=1000,
    sterileLogger=True,
    propagationStepCollection="propagation-steps",
)
```



https://github.com/asalzburger/acts/tree/ws-add-user-algorithm-python-bindings-embedded

# Step 4: connect the algorithms



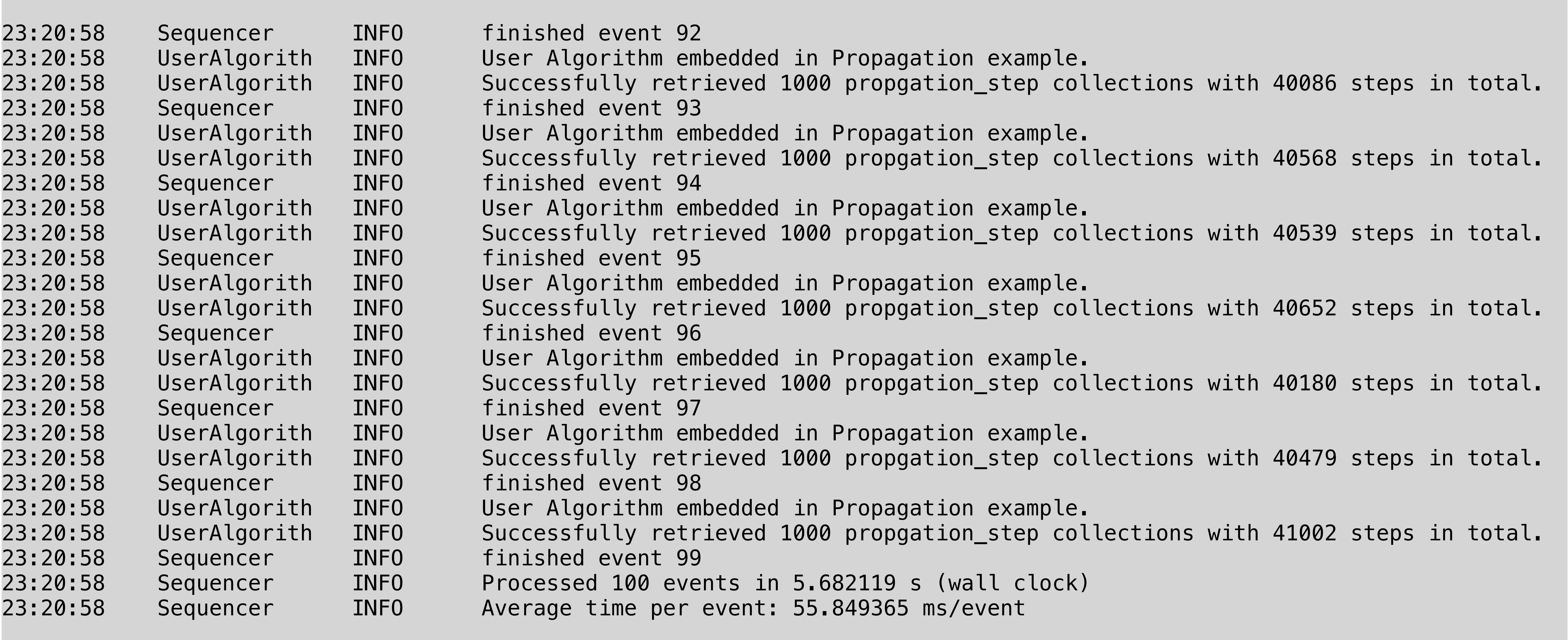
```
ActsExamples::UserAlgorithm::UserAlgorithm(
    ActsExamples::UserAlgorithm::Config cfg, Acts::Logging::Level lvl)
    : ActsExamples::BareAlgorithm("UserAlgorithm", lvl), m_cfg(std::move(cfg)) {
  if (m_cfg.inputStepCollection.empty()) {
    throw std::invalid_argument("Missing space point input collections");
  }
}

ActsExamples::ProcessCode ActsExamples::UserAlgorithm::execute(
    const AlgorithmContext& ctx) const {
  using PropagationStepCollection =
      std::vector<std::vector<Acts::detail::Step>>;

  ACTS_INFO(m_cfg.message);
  auto propagationSteps =
      ctx.eventStore.get<PropagationStepCollection>(m_cfg.inputStepCollection);

  unsigned int totalSteps = 0;
  for (auto prop : propagationSteps) {
    totalSteps += prop.size();
  }

  ACTS_INFO("Successfully retrieved " << propagationSteps.size()
                                      << " propgation_step collections with "
                                      << totalSteps << " steps in total.");

  return ActsExamples::ProcessCode::SUCCESS;
}
```

Retrieve the output of the previous algorithm from the event-contextual WhiteBoard.

Do some fancy stuff with it.

https://github.com/asalzburger/acts/tree/ws-add-user-algorithm-python-bindings-embedded-connected

# Step 4: connect the algorithms



```
23:20:58    Sequencer      INFO      finished event 92
23:20:58    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
23:20:58    UserAlgorith   INFO      Successfully retrieved 1000 propgation_step collections with 40086 steps in total.
23:20:58    Sequencer      INFO      finished event 93
23:20:58    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
23:20:58    UserAlgorith   INFO      Successfully retrieved 1000 propgation_step collections with 40568 steps in total.
23:20:58    Sequencer      INFO      finished event 94
23:20:58    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
23:20:58    UserAlgorith   INFO      Successfully retrieved 1000 propgation_step collections with 40539 steps in total.
23:20:58    Sequencer      INFO      finished event 95
23:20:58    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
23:20:58    UserAlgorith   INFO      Successfully retrieved 1000 propgation_step collections with 40652 steps in total.
23:20:58    Sequencer      INFO      finished event 96
23:20:58    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
23:20:58    UserAlgorith   INFO      Successfully retrieved 1000 propgation_step collections with 40180 steps in total.
23:20:58    Sequencer      INFO      finished event 97
23:20:58    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
23:20:58    UserAlgorith   INFO      Successfully retrieved 1000 propgation_step collections with 40479 steps in total.
23:20:58    Sequencer      INFO      finished event 98
23:20:58    UserAlgorith   INFO      User Algorithm embedded in Propagation example.
23:20:58    UserAlgorith   INFO      Successfully retrieved 1000 propgation_step collections with 41002 steps in total.
23:20:58    Sequencer      INFO      finished event 99
23:20:58    Sequencer      INFO      Processed 100 events in 5.682119 s (wall clock)
23:20:58    Sequencer      INFO      Average time per event: 55.849365 ms/event
```

**https://github.com/asalzburger/acts/tree/ws-add-user-algorithm-python-bindings-embedded-connected**