blue yonder

Forward looking. Forward thinking.

Embedding Python

Charming the Snake with C++

July 23, 2014, Michael König

Outline

- Why meddle with snakes?
- Don't innovate, integrate!
- Snake charming 101
- Including batteries
- Summary

blue yonder

blue yonder

Forward looking. Forward thinking.

Why Meddle With Snakes?

How C++ Benefits from Python

Data Science at Blue Yonder

- Prediction platform
  - Distributed task scheduling system
  - Handles data source access
  - Designed for reliable 24/7 operations
  - Written in C++
- Simplifies / reduces chores
- No real help for prediction model development

Beyond C++

- Python
  - Scikit-learn: Machine learning algorithms
  - Numpy: Efficiency
  - Core language: Quick feedback loop
- Solution: Embedding Python
  - Safe, reliable operations
  - Fast model development
  - Quick to production (no redevelopment)

blue yonder

blue yonder

Forward looking. Forward thinking.

Don't Innovate, Integrate!

Embedding the Python Interpreter

Parental Advisory

blue yonder

Interpreter Basics

- Essential CPython calls

```
struct python_interpreter {     // context manager
    python_interpreter() {      // __enter__()
        auto const signals_handled = false;
        Py_InitializeEx(signals_handled);
    }

    ~python_interpreter() {     // __exit__()
        Py_Finalize();
    }
};
```

C++

blue yonder

Interpreter Basics

- Usage pattern:

```
int main() {
    python_interpreter interpreter; // "with"

    // do things with the python interpreter

    return 0;
}
```

C++

- Interpreter reinitialization can be problematic

blue yonder

CPython in Threaded Environments

- Global interpreter lock (GIL)
- Main thread:
  - Make interpreter thread-aware
  - Release GIL for other threads
- Worker threads:
  - Acquire / release GIL

blue yonder

Thread-aware Interpreter

```
struct python_interpreter {     // context manager
    python_interpreter() {      // __enter__()
        Py_InitializeEx(false);
        PyEval_InitThreads();
        thread_state = PyEval_SaveThread();
    }
    ~python_interpreter() {     // __exit__()
        PyEval_RestoreThread(thread_state);
        Py_Finalize();
    }
    PyThreadState * thread_state;
};
```

C++

blue yonder

Global Interpreter Lock

```
struct global_interpreter_lock { // context manager
    global_interpreter_lock() {  // __enter__()
        gil_state = PyGILState_Ensure();
    }

    ~global_interpreter_lock() { // __exit__()
        PyGILState_Release(gil_state);
    }

    PyGILState_STATE gil_state;
};
```

C++

blue yonder

Worker Threads

- Usage pattern:

```
void worker_thread() {
    global_interpreter_lock lock; // "with"

    // do things with the python interpreter
}
```

C++

blue yonder

blue yonder

Forward looking. Forward thinking.

Snake Charming 101

Interacting with the Python Interpreter

Interacting With Python

- boost::python C++ library
  - www.boost.org
  - Mature, commercially usable open source software
- Features
  - CPython API wrappers
  - Python object life time management
  - C++ ↔ Python type conversion
  - Rudimentary exception handling
  - Expose C++ code to Python

blue yonder

Evaluating Python Code with C++

C++

```
namespace bp = boost::python;
std::string const python_code = "2 * 21";


try {
    auto py_result = bp::eval(python_code);
    auto cpp_result = bp::extract<int>(py_result);
} catch (bp::error_already_set) {
    // improved error handling with stack trace etc.
}
```

- Wrap boost::python features for better error handling

blue yonder

blue yonder

Forward looking. Forward thinking.

Including Batteries

Make Python Users Feel at Home

Embrace Python

- Exchange cool C++ data structures as
  - Lists!
  - Dictionaries!
- More than simple data?
  - Check for standards (iterators, Python DB API, etc.)
- Python code should never know it is embedded

blue yonder

Logging Out-Of-The-Box

- Python logging facility

```
import logging
logging.warning('Out of dictionaries')
```

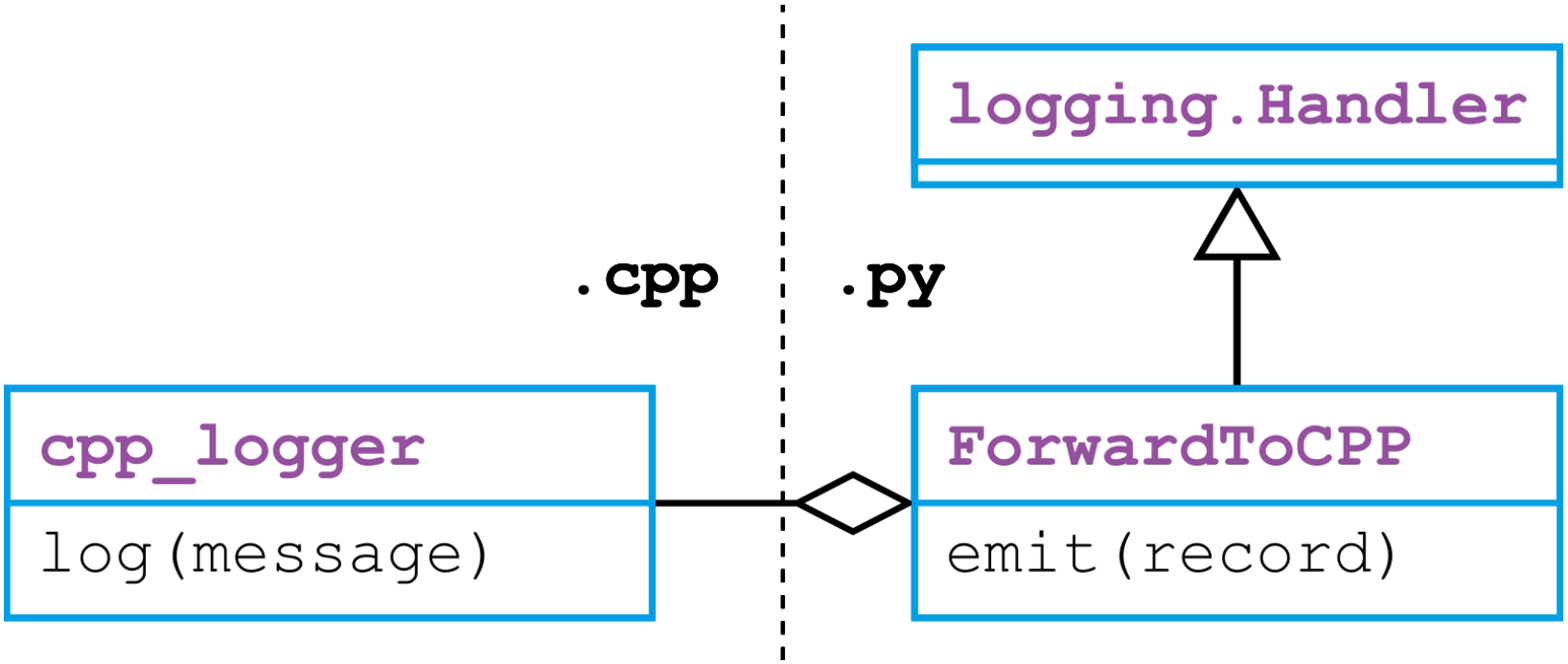
Py

- Goal: Forward log messages to C++

- Idea: Register custom `logging.Handler`

Logging Out-Of-The-Box

- Problem: C++ class implements Python concept?



- Solution: One layer of indirection

blue yonder

C++ to Python

C++

```
struct cpp_logger {
    void log(std::string const & message) {
        // really simple logging
        std::cout << message << std::endl;
    }
}

// make cpp_logger a python callable
bp::class_<cpp_logger>{"cpp_logger", bp::no_init}
    .def("__call__", &cpp_logger::log);
```


blue yonder

Python to C++

Py

```
import logging

class ForwardToCPP(logging.Handler):
    def __init__(self, receiver):
        self.receiver = receiver
        logging.Handler.__init__(self)

    # satisfy logging.Handler concept
    def emit(self, record):
        self.receiver(record.getMessage())
```


blue yonder

Initializing the Forwarding Log Handler

- Pass instance of exposed C++ class to Python class

```
cpp_logger logger;

auto main_module = bp::import("__main__");
auto main_dict = main_module.attr("__dict__");
auto create_handler = main_dict["ForwardToCPP"];

auto handler = create_handler(logger);
```

C++

blue yonder

blue yonder

Forward looking. Forward thinking.

Summary

Summary

- Embedding Python is not that hard
- boost::python helps
  - A little clumsy to use
- Let users write pythonic code
  - Adhere to Python standards / conventions
  - Maintain unit testability

blue yonder

Blue Yonder is hiring!

Please visit our booth at EuroPython 2014