BY CHARLES ESCOTT

MENTORS: JIM PIVARSKI AND DAVID LANGE

THE PROBLEM

- Particle physics data is too complicated for normal coding libraries
- Awkward Arrays supports less conventional types/functions using Python
  - Easier to work with
  - JIT (just in time) compilation => slow to run

Columns

Rows	0	1	2	3
0	0	1	2	3
1	4	5		
2	6	7	8	

Jagged Array Structure

oreilly.com

THE SOLUTION

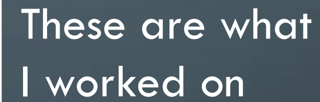
pybind11

pybind11.readthedocs.io

- Implement Awkward Arrays in C++!
  - Pre-compiled => usually much faster speeds
- Pybind11 allows C++ code to compile into Python binaries
- End product: a Python library that's pre-compiled, written in C++

AWKWARD ARRAYS

- Jagged Arrays
- Tables
- Chunked Arrays
- Masked Arrays
- Indexed Arrays
- Sparse Arrays
- Appendable Arrays
- Etc.



INSIGHT INTO PYBIND11

BINDING C++ TO PYTHON

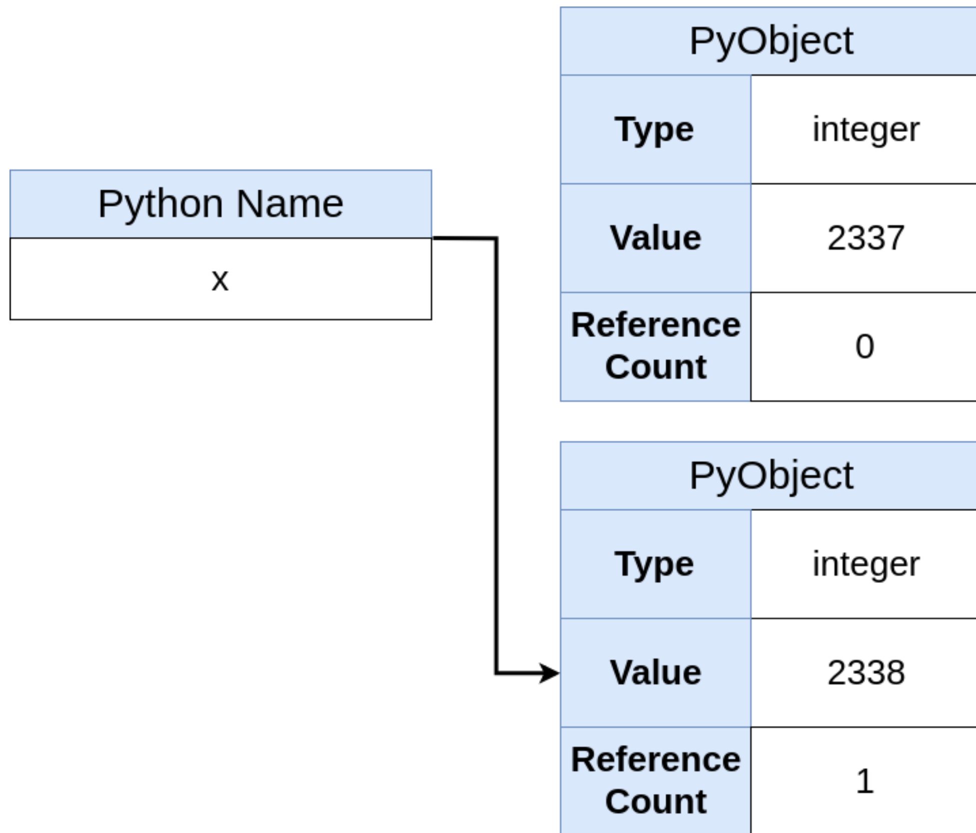
- Pybind11 is a very versatile library
- What you can do with it:
  - Expose properties, static methods (class methods), classes, constructors, iteration, overloaded functions, default parameters, and more to Python
- What you can't do with it:
  - Have C++ classes inherit from Python classes
  - Find well-written documentation

```
YBIND11_MODULE(array_impl, m) {
    py::class_<JaggedArray>(m, "JaggedArray")
        .def(py::init<py::object, py::object, py::objec
        .def_static("slice_test", &JaggedArray::slice_t
        .def_static("test_make", &JaggedArray::test_mak
        .def_property("starts", &JaggedArray::get_start
        .def_property("stops", &JaggedArray::get_stops,
        .def_property("content", &JaggedArray::python_g
        .def_static("offsets2parents", &JaggedArray::py
        .def_static("counts2offsets", &JaggedArray::pyt
        .def_static("startsstops2parents", &JaggedArray
        .def_static("parents2startsstops", &JaggedArray
            py::arg("parents"), py::arg("length") = -1)
```

CARRYING OVER FROM PYTHON TO C++

(I CAN'T BELIEVE IT'S NOT PYTHON!)

- If done correctly, can be implemented to appear just like a Python library
- Class functions like __getitem__, __str__, __len__, etc. can be used
  - Works both for exposing C++ to Python and for built-in Python objects in the C++
- Object, slice, list, tuple, array (from NumPy), and many more are available for use in C++, with type casting support for iterables
- Automatically translates exceptions, strings, and integer types to their C++ counterparts



realpython.com

MEMORY OWNERSHIP

- Python reference counting in C++, too
- Pybind11 does it for you, again
- (mostly) No issues with memory
- `std::unique_ptr` and `std::shared_ptr` are supported, but usually unnecessary

NUMPY ARRAYS

```
struct buffer_info {
    void *ptr = nullptr;
    ssize_t itemsize = 0;
    ssize_t size = 0;
    std::string format;
    ssize_t ndim = 0;
    std::vector<ssize_t> shape;
    std::vector<ssize_t> strides;
```

- General array class and `array_t<>` templated class
- Buffer info struct
  - Pro: easy to work with in C without worrying about type (only itemsize)
  - Con: lots of code. Lots.

EXAMPLE OF ≥1DIM. ARRAY TRAVERSAL

(INCLUDING UNCONVENTIONAL STRIDES)

```
int getMax_32bit(struct c_array *input, ssize_t dim, ssize_t index, int32_t *max) {
    // to be initially called with getMax_32bit(input, 0, 0, *0)
    if (dim > input->ndim - 1)
        return 0;
    if (dim == 0) {
        if (input->shape[0] == 0) {
            *max = 0;
            return 1;
        }
        *max = ((int32_t*)input->ptr)[0];
    }
    ssize_t N = input->strides[dim] / input->itemsize;
    if (dim == input->ndim - 1) {
        for (ssize_t i = 0; i < input->shape[dim]; i++)
            if (((int32_t*)input->ptr)[index + i * N] > *max)
                *max = ((int32_t*)input->ptr)[index + i * N];
        return 1;
    }
    for (ssize_t i = 0; i < input->shape[dim]; i++)
        getMax_32bit(input, dim + 1, index + i * N, max);
    return 1;
}
```

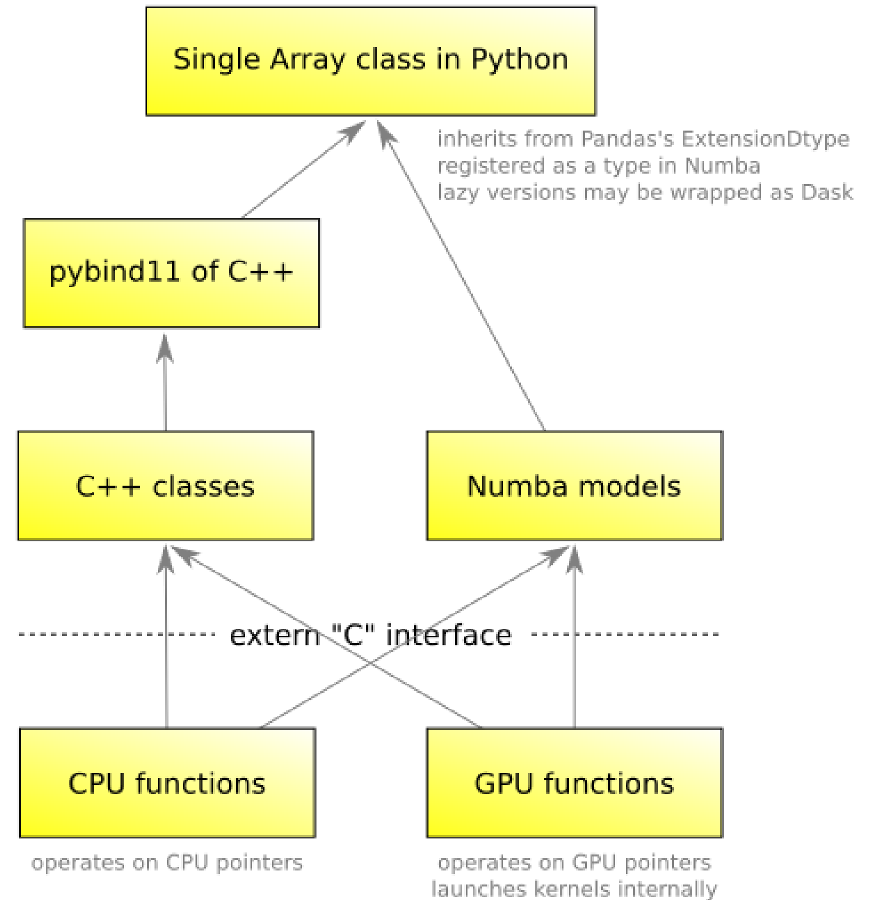
getitem

step

Recursively nested "for" loops over a flat array with > 0 dimensions and any valid shape/strides

CHANGING DIRECTION TO C

NEW PROJECT ARCHITECTURE



Made by Jim Pivarski (thanks Jim, I'm stealing your diagram)

IMPLEMENTING C >> C++ >> PYTHON

- C is a bit more difficult than C++
  - No templates
  - No overloaded functions
  - No classes
  - No exceptions
  - No booleans
- Keeping C methods clean of memory management
- Ultimately, not too bad

“FINAL” PRODUCT

- A solid basis for Awkward 1.0 (the new architecture)
- Many hours and 5000+ lines of code worth of troubleshooting, research, and experience
- A JaggedArray class in C++/C which functions independently from awkward-array and contain ZERO Python code

THANKS FOR LISTENING :)