Awkward Array

*An array library for **nested**, **variable-sized data**, including arbitrary-length lists, records, mixed types, and missing data, using **NumPy-like idioms**.*

*Arrays are **dynamically typed**, but operations on them are **compiled and fast**.*
Coincides with NumPy when arrays are regular; generalizes when they're not.

Ianna Osborne, Princeton University

NSF
PHY-2323298
iris hep
Institute for Research & Innovation in Software for High Energy Physics

Example of an “awkward” array

Record structures with differently typed fields

Array of variable-length lists (“ragged” or “jagged” arrays)

```
array = ak.Array( [

    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],

    [ ],

    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]

 ] )
```

Heterogeneous data (union/variant types)

Nested variable-length lists

Missing data

Numpy-like expression

```
output = np.square(array["y", ..., 1:])
```

Result

```
output.to_list()
[
    [[], [4], [4, 9]],
    [],
    [[4, 9, 16], [4, 9, 16, 25]]
]
```

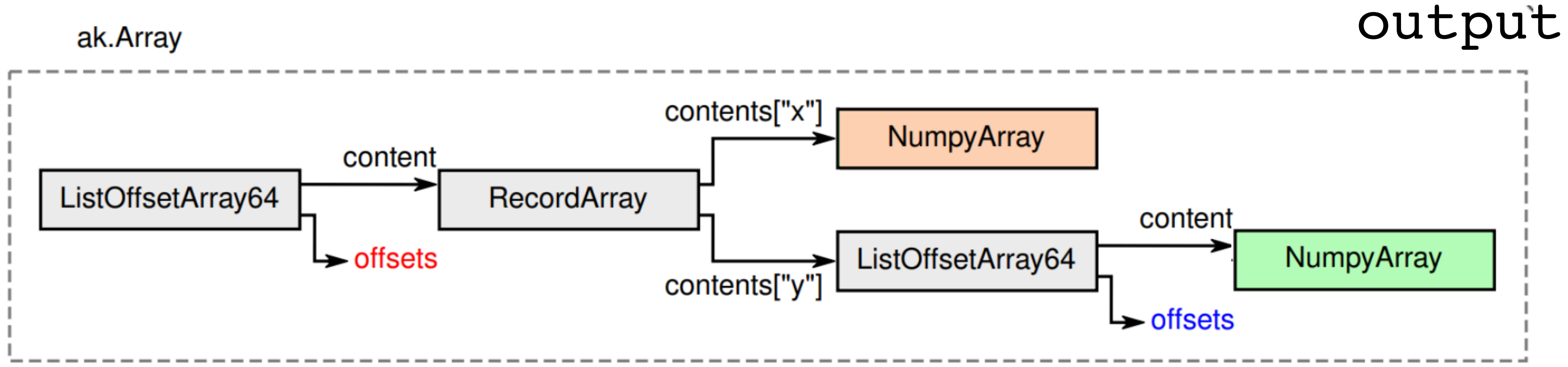
Equivalent Python

```
output = []
for sublist in python_objects:
    tmp1 = []
    for record in sublist:
        tmp2 = []
        for number in record["y"][1:]:
            tmp2.append(number)
        tmp1.append(tmp2)
    output.append(tmp1)
```

(for a similar calculation 10 million times larger, single threaded)

343 ms ± 10.1 ms
576.0 MB of memory

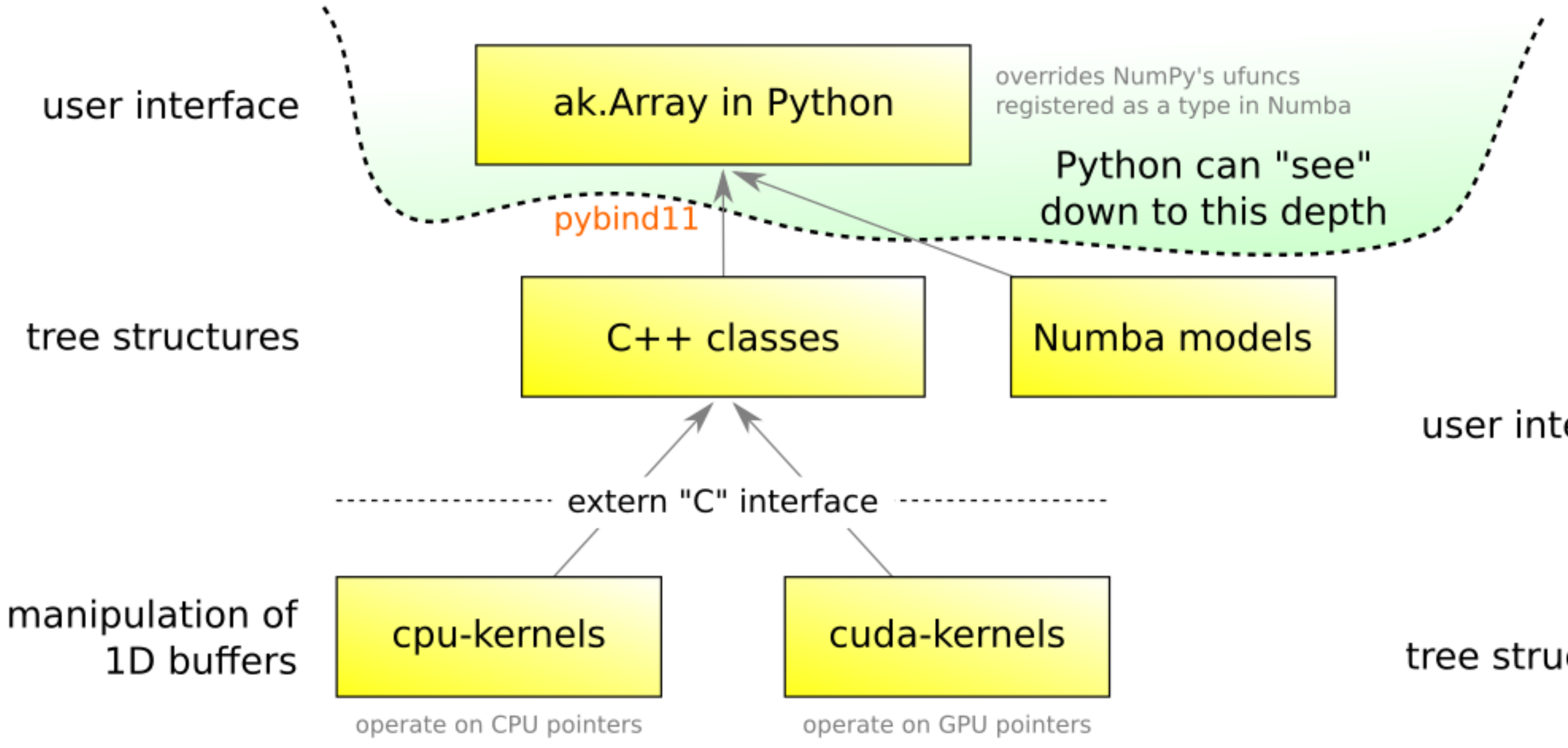
3.03 s ± 532 ms
4–5 GB of memory



PHY-2323298

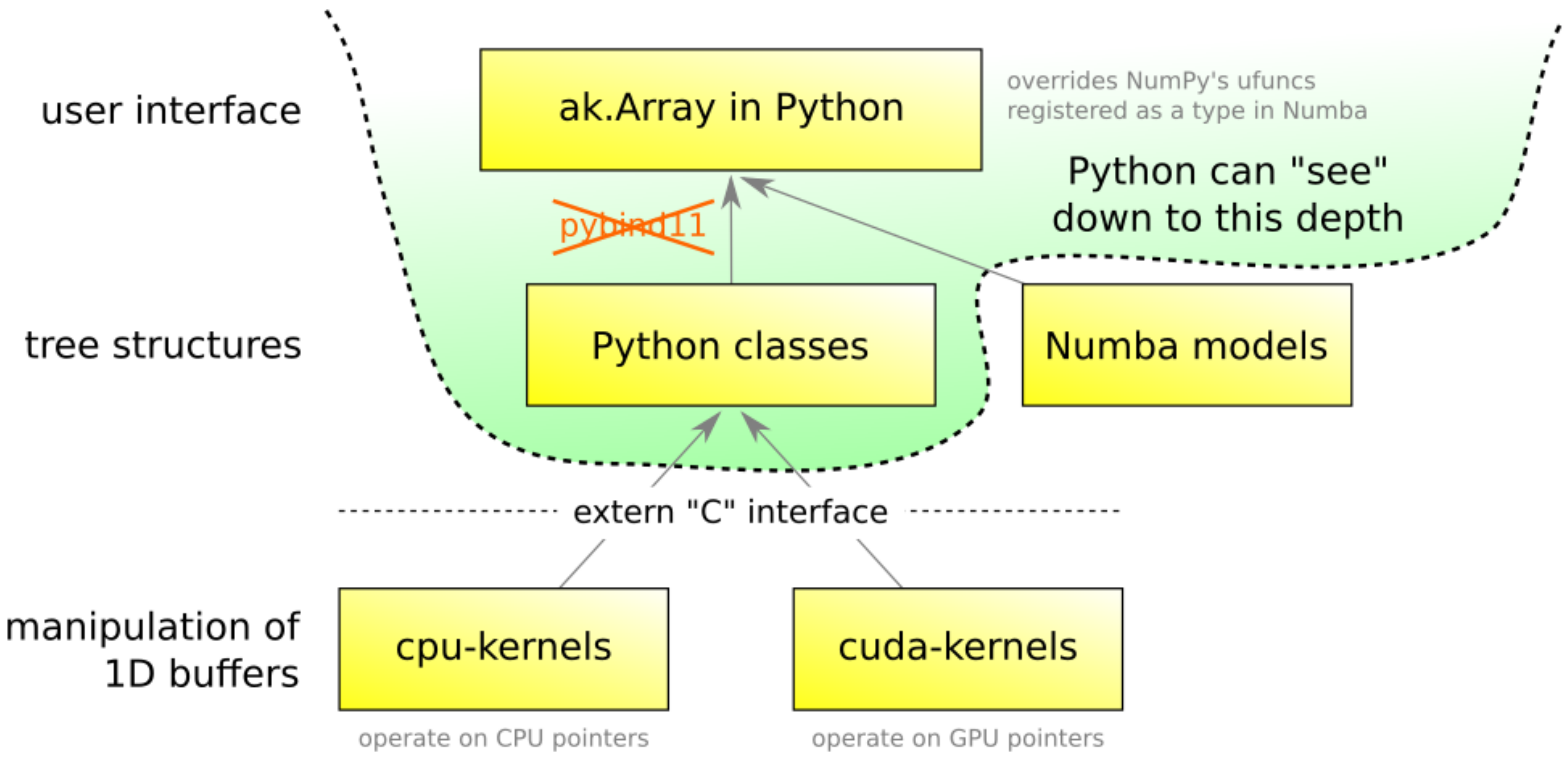
Institute for Research & Innovation in Software for High Energy Physics

Evolution of Architecture



[Awkward 1.x](#)

[Awkward 2.x](#)



[Lessons learned in Python-C++ integration](#)

Jim Pivarski, Princeton University, ACAT 2021

PRINCETON UNIVERSITY

iris hep

How it works

"node1-data" =>
```
[0x01, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
0x00  …  0x00,
0x00, 0x07, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00]
```

"node0-offsets" =>
```
[0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x03, 0x00  …
0x00, 0x00, 0x07,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00]
```

buffers

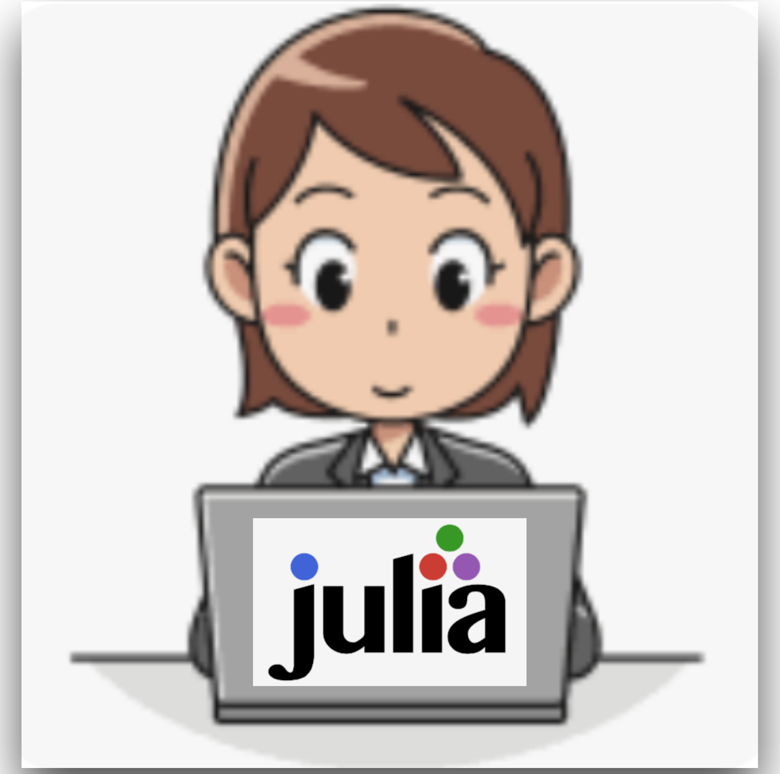

```
'{"offsets":"i64",
  "content":{"parameters":{},
             "inner_shape":[],
             "class":"NumpyArray",
             "form_key":"node1",
             "primitive":"int64"},
 "parameters":{},
 "class":"ListOffsetArray",
 "form_key":"node0"}'
```


Awkward Array

ak_julia.py module →

← PythonCall dependency

Awkward Array.jl




```
array = ak.from_buffers(form, len, containers)
form, len, containers = ak.to_buffers(array)
```

```
array = AwkwardArray.from_iter([[1, 2, 3], [4], [5, 6, 7]])
form, len, containers = AwkwardArray.to_buffers(array)
array = AwkwardArray.from_buffers(form, len, containers)
```

Awkward Array Library Integrations

Awkward Array is a part of larger scientific Python ecosystem. It works well with other libraries.

APACHE ARROW

Parquet

Lossless, bidirectional, zero-copy conversions between Awkward Array, Apache Arrow, Feather, and Parquet.

ROOT Data Analysis Framework

JIT-compiled as a C++ iterator in ROOT's RDataFrame workflow.

CuPy

Awkward Arrays on GPUs, backed by CuPy.

JIT-compile Awkward Arrays in C++ with cppyy. Header-only library to build Awkward Arrays in C++ and transfer them to Python.

Numba

Efficiently iterate over or build Awkward Arrays in code JIT-compiled with Numba, on CPUs and GPUs

pandas

polars

RAPIDS

Akimbo: work with Awkward Arrays as DataFrame columns

[ragged]

Manipulating ragged arrays as though they were **NumPy** or **CuPy** arrays, following the Array API specification.

TensorFlow

Bidirectional conversion between Awkward Array and TensorFlow RaggedTensor.

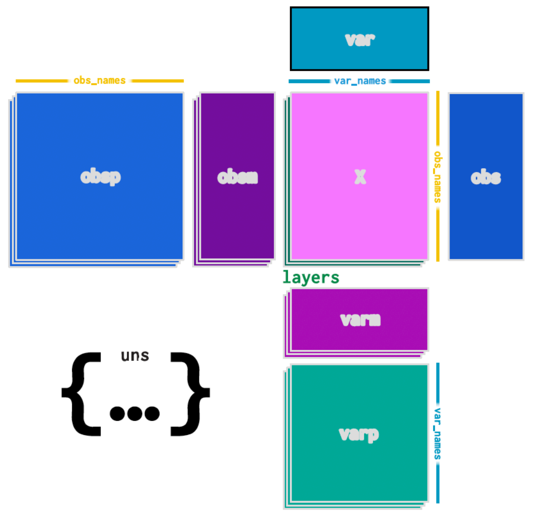
Auto-differentiate through expressions involving Awkward Arrays, using JAX

bluesky

Store Awkward Arrays in the Tiled database and slice them in the cloud.

DASK

dask-awkward is a high-level collection type for Dask, alongside Array and DataFrame.



Integrated into AnnData for single-cell genomics.

Read any Kaitai Struct format into Awkward Arrays.

AwkwardArray.jl

Exchange data with the Julia language through a reimplementation of the Awkward Array memory layouts.

NSF PHY-2323298

iris hep Institute for Research & Innovation in Software for High Energy Physics

Why Array-oriented Programming?

- **Conciseness**: You can express complex operations on large datasets with just a few lines of code.
- **Performance**: Operations on entire arrays can be optimized by the underlying libraries to run faster than equivalent loops.
- **Simplicity**: It simplifies code by eliminating the need for explicit loops and conditionals.

PHY-2323298

Institute for Research & Innovation in Software for High Energy Physics