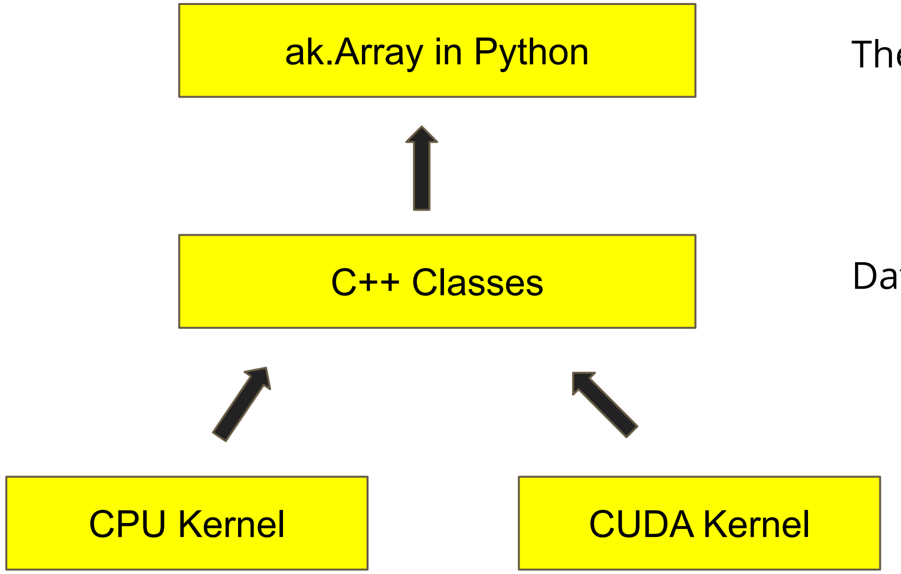
Automating Awkward Array Testing

Santam Roy Choudhury
(National Institute of Technology, Durgapur)

The Different Layers in Awkward Array



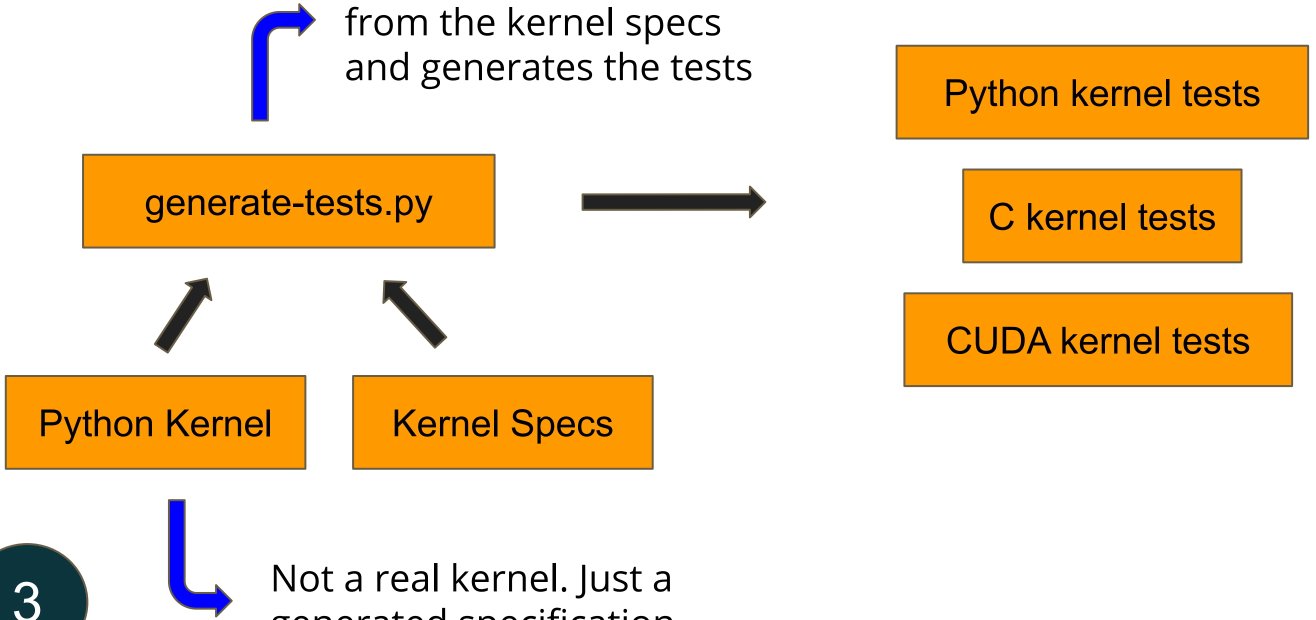
The one we interact with

Data Navigation

Operates on CPU Pointers

Operates on GPU Pointers

The Current Testing Infrastructure

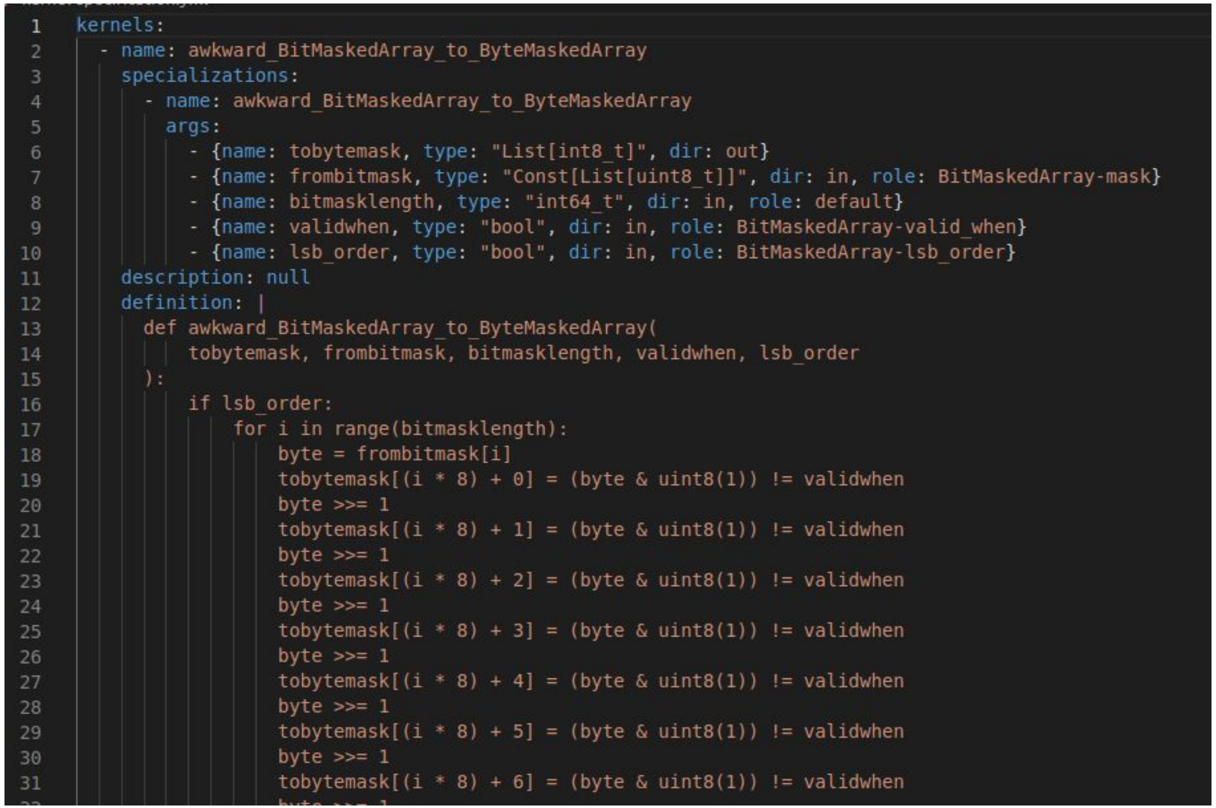


Creates a python kernel from the kernel specs and generates the tests

Auto generated tests

Not a real kernel. Just a generated specification.

The Kernel Specification

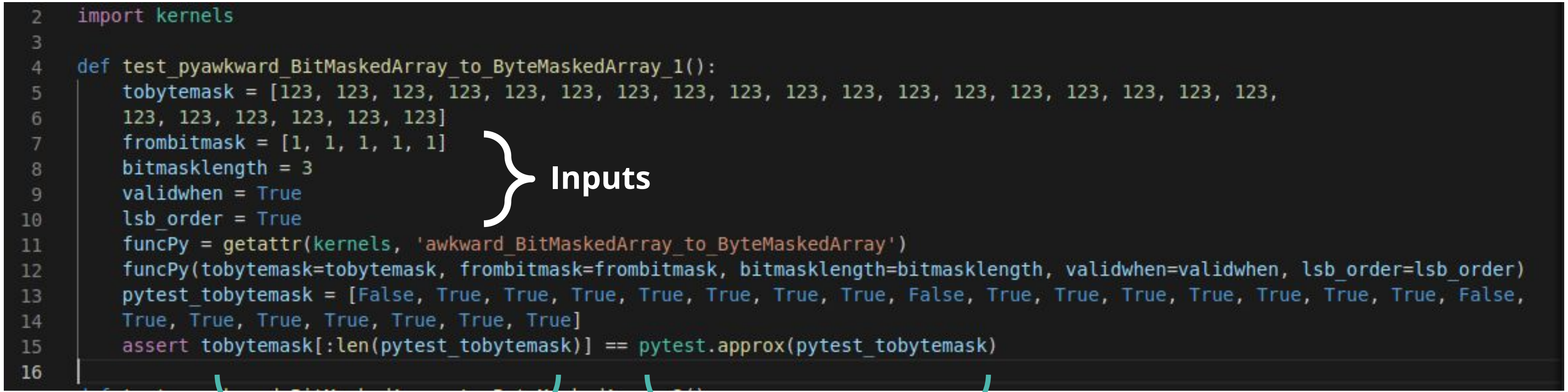


```
kernels:
  - name: awkward_BitMaskedArray_to_ByteMaskedArray
    specializations:
      - name: awkward_BitMaskedArray_to_ByteMaskedArray
        args:
          - {name: tobytemask, type: "List[int8_t]", dir: out}
          - {name: frombitmask, type: "Const[List[uint8_t]]", dir: in, role: BitMaskedArray-mask}
          - {name: bitmasklength, type: "int64_t", dir: in, role: default}
          - {name: validwhen, type: "bool", dir: in, role: BitMaskedArray-valid_when}
          - {name: lsb_order, type: "bool", dir: in, role: BitMaskedArray-lsb_order}
    description: null
    definition: |
      def awkward_BitMaskedArray_to_ByteMaskedArray(
          tobytemask, frombitmask, bitmasklength, validwhen, lsb_order
      ):
          if lsb_order:
              for i in range(bitmasklength):
                  byte = frombitmask[i]
                  tobytemask[(i * 8) + 0] = (byte & uint8(1)) != validwhen
                  byte >>= 1
                  tobytemask[(i * 8) + 1] = (byte & uint8(1)) != validwhen
                  byte >>= 1
                  tobytemask[(i * 8) + 2] = (byte & uint8(1)) != validwhen
                  byte >>= 1
                  tobytemask[(i * 8) + 3] = (byte & uint8(1)) != validwhen
                  byte >>= 1
                  tobytemask[(i * 8) + 4] = (byte & uint8(1)) != validwhen
                  byte >>= 1
                  tobytemask[(i * 8) + 5] = (byte & uint8(1)) != validwhen
                  byte >>= 1
                  tobytemask[(i * 8) + 6] = (byte & uint8(1)) != validwhen
```

Arguments

Function Definition

A Python Kernel Test



```python
import kernels

def test_pyawkward_BitMaskedArray_to_ByteMaskedArray_1():
    tobytemask = [123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123,
    123, 123, 123, 123, 123, 123]
    frombitmask = [1, 1, 1, 1, 1]
    bitmasklength = 3
    validwhen = True
    lsb_order = True
    funcPy = getattr(kernels, 'awkward_BitMaskedArray_to_ByteMaskedArray')
    funcPy(tobytemask=tobytemask, frombitmask=frombitmask, bitmasklength=bitmasklength, validwhen=validwhen, lsb_order=lsb_order)
    pytest_tobytemask = [False, True, True, True, True, True, True, True, False, True, True, True, True, True, True, True, False,
    True, True, True, True, True, True, True]
    assert tobytemask[:len(pytest_tobytemask)] == pytest.approx(pytest_tobytemask)
```

Inputs

Result got from kernel

Result we are expecting

Does it cover all of the test cases?

NO

What are some of the loopholes here?

- Not too many specific test cases
- Not testing for specific errors
- The roles of the arguments are not well defined

What is a good solution to fill up the gap?

Property based Testing

What is property based testing?

A type of test in which we define the properties of the input and the output that we are expecting

What is the advantage of having property based tests?

- More hard coded input datas to test with
- Test a larger section of the codebase
- Very little code
- More efficient
- Flexible

Hypothesis Library

- *A boon to property based testing*

- Various strategies to get constraints based data
- Get a more elaborative test result
- Regenerate failing test inputs
- Shrinking

And much more..

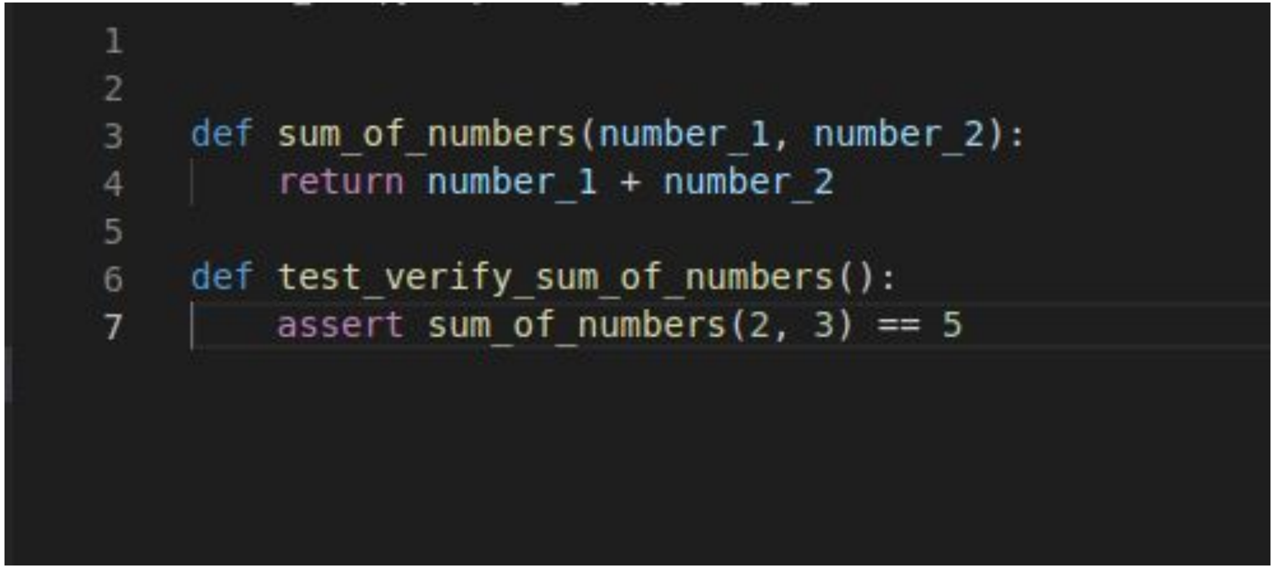
How a unit test runs

$$\begin{tikzcd}\text{Input Data} \arrow[r] & \text{Perform an operation} \arrow[r] & \text{Assert the result}\end{tikzcd}$$

How a tests written using hypothesis runs

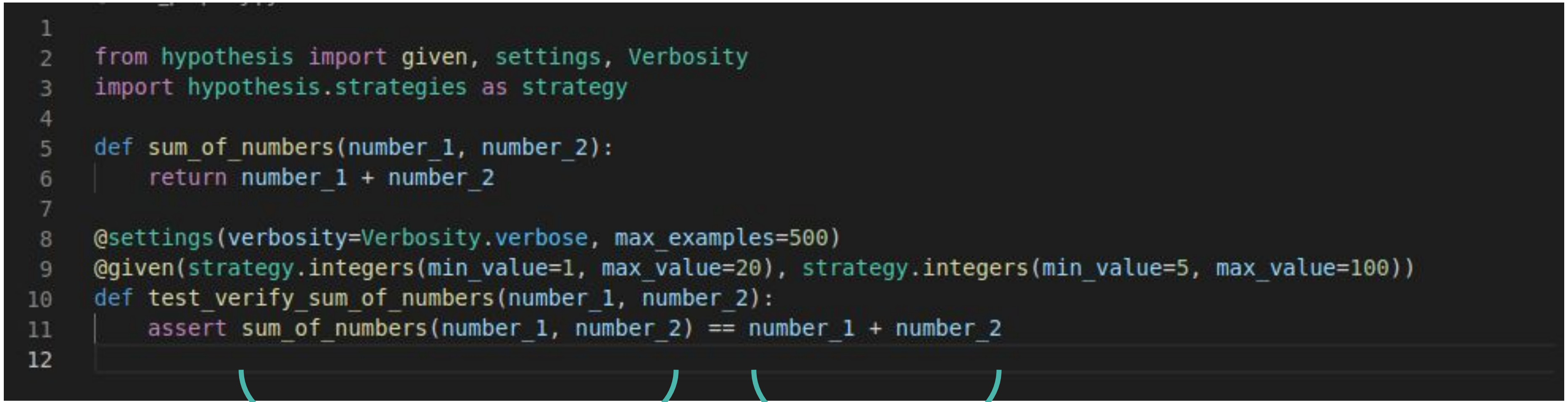
$$\begin{tikzcd}\text{Input Data based on some constraints} \arrow[r] & \text{Perform an operation} \arrow[r] & \text{Assert the result}\end{tikzcd}$$

A sample unit test



```python


def sum_of_numbers(number_1, number_2):
    return number_1 + number_2

def test_verify_sum_of_numbers():
    assert sum_of_numbers(2, 3) == 5
```

A similar property based test



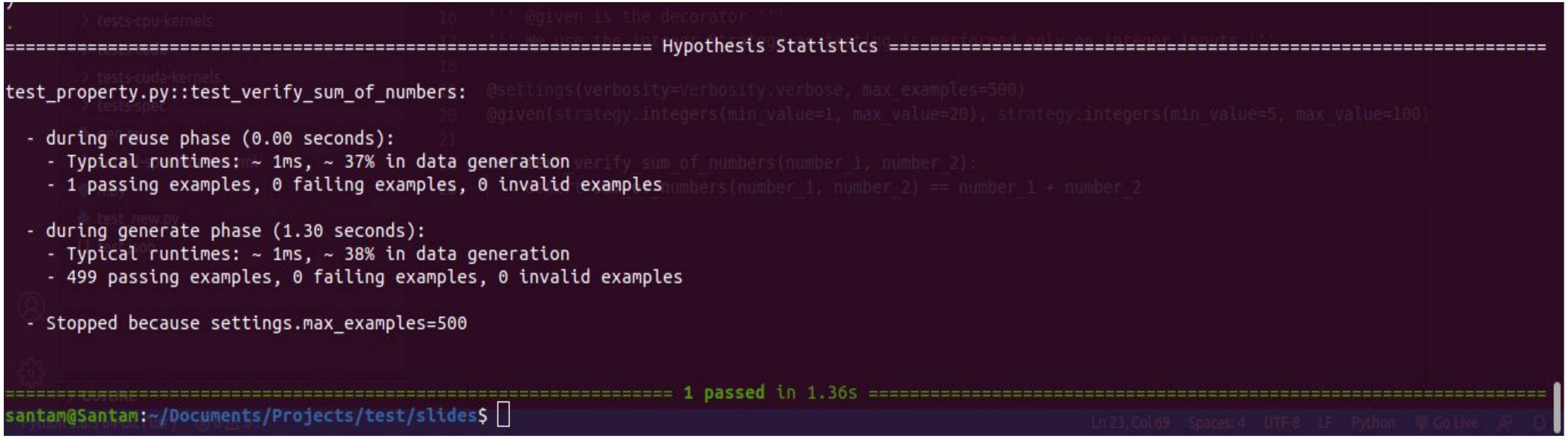
```python

from hypothesis import given, settings, Verbosity
import hypothesis.strategies as strategy

def sum_of_numbers(number_1, number_2):
    return number_1 + number_2

@settings(verbosity=Verbosity.verbose, max_examples=500)
@given(strategy.integers(min_value=1, max_value=20), strategy.integers(min_value=5, max_value=100))
def test_verify_sum_of_numbers(number_1, number_2):
    assert sum_of_numbers(number_1, number_2) == number_1 + number_2

```

The CPU Kernel Function will come here

The Python Kernel Function will come here

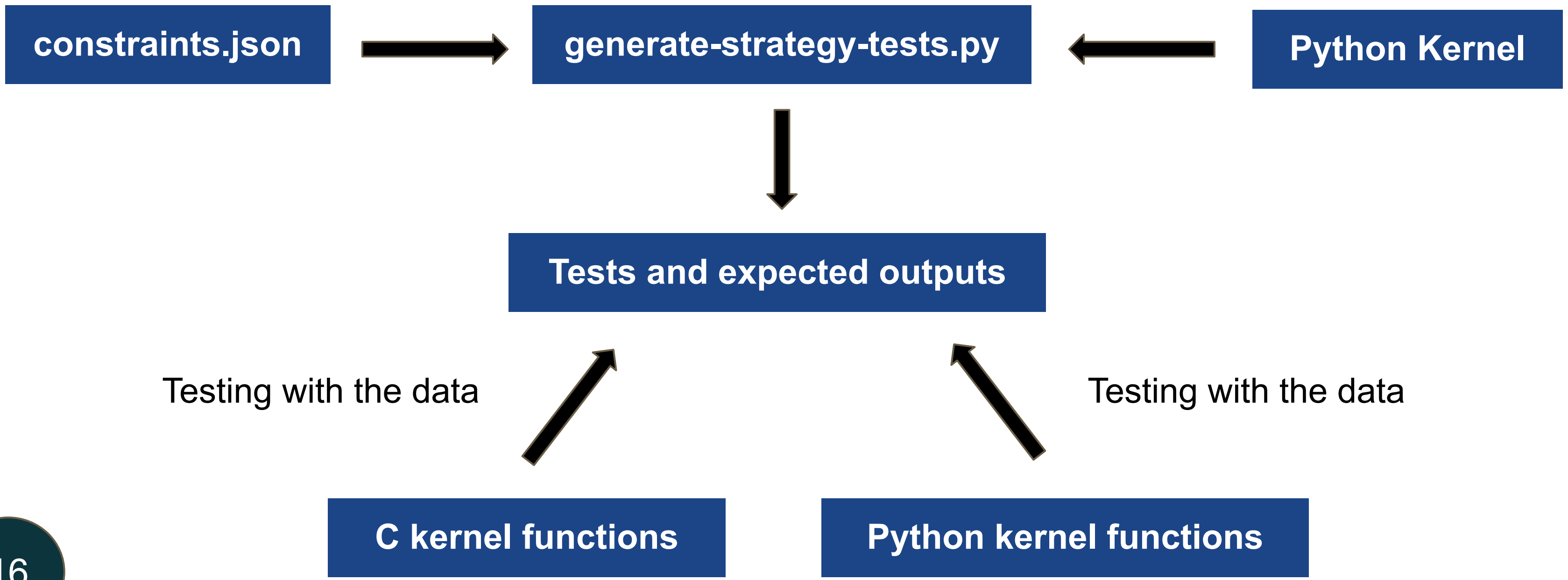
Let’s Run It

A well organized statistical result!!



```
============================================================== Hypothesis Statistics ==============================================================

test_property.py::test_verify_sum_of_numbers:

  - during reuse phase (0.00 seconds):
    - Typical runtimes: ~ 1ms, ~ 37% in data generation
    - 1 passing examples, 0 failing examples, 0 invalid examples

  - during generate phase (1.30 seconds):
    - Typical runtimes: ~ 1ms, ~ 38% in data generation
    - 499 passing examples, 0 failing examples, 0 invalid examples

  - Stopped because settings.max_examples=500


================================================================ 1 passed in 1.36s ================================================================
santam@Santam:~/Documents/Projects/test/slides$ 
```

The Approach



An overview

- Unit tests may leave some corner cases untested which can be found out using property based test.
- The awkward array creates a python kernel which is a specification used to auto generating tests.
- The hypothesis library can be used to get a well documented result of test cases.

Some resources

Hypothesis- https://hypothesis.readthedocs.io/en/latest/index.html#

Awkward Array- https://github.com/scikit-hep/awkward-1.0

The Kernel Specification-
https://github.com/scikit-hep/awkward-1.0/blob/main/kernel-specification.yml

The generate tests script-
https://github.com/SantamRC/awkward-1.0/blob/main/dev/generate-tests.py

A Special Mention to Jim Pivarski and Ianna Osborne

Thank You!!

https://github.com/SantamRC

santamdev404@gmail.com