# COFFEA-CASA
## A USER’S PERSPECTIVE

**Presented By:** Mat Adamec *(Nebraska)*

**Contributors:** Garhan Attebury, Ken Bloom, Carl Lundstedt, Oksana Shadura, John Thiltges, Andrew Wightman (*Nebraska*); Brian Bockelman (*Morgridge Institute*)

## What is Coffea?

- Coffea is designed for columnar analysis – think NumPy, not loops.
- Awkward arrays serve as the foundation of Coffea and handle jagged data structures.

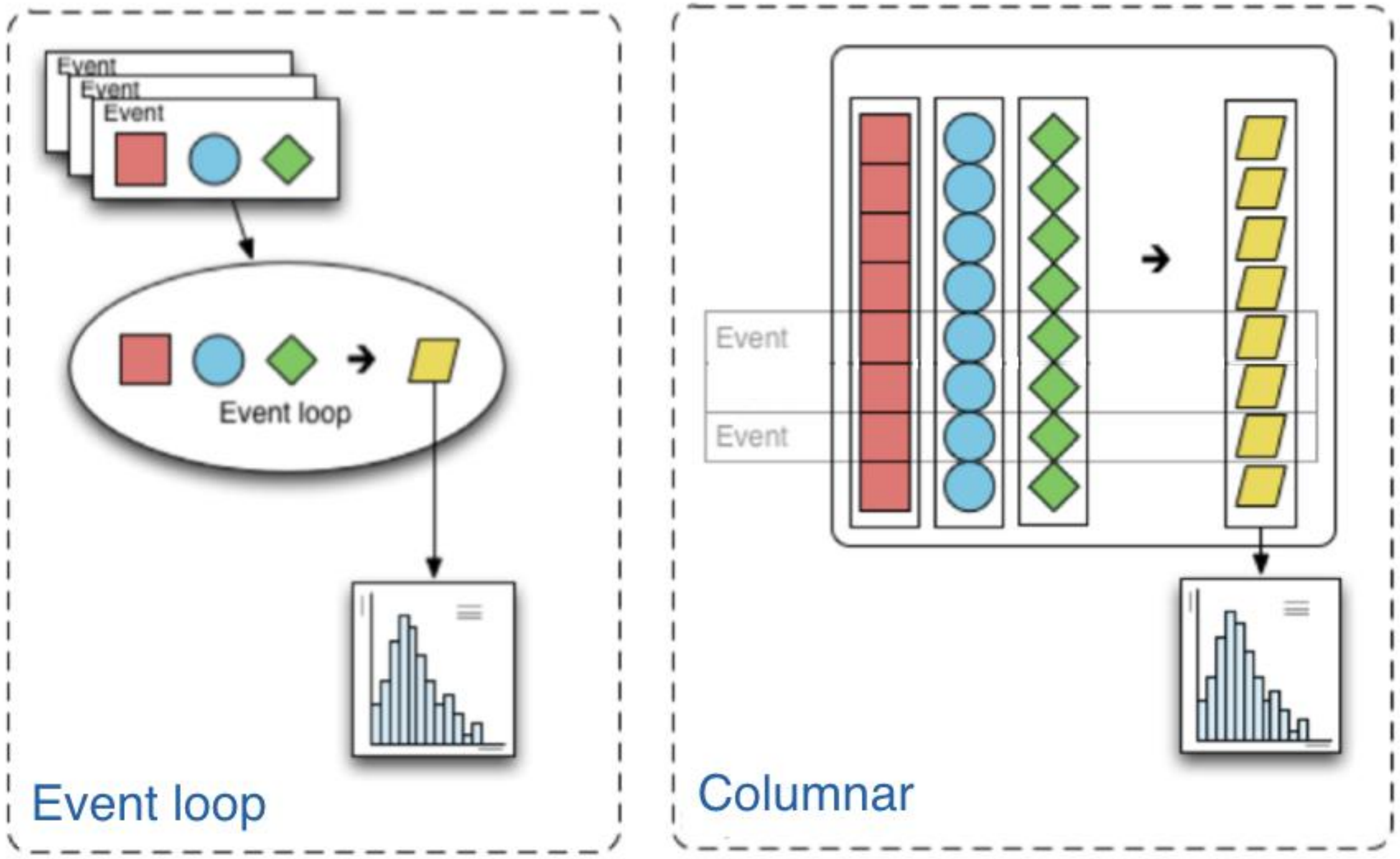
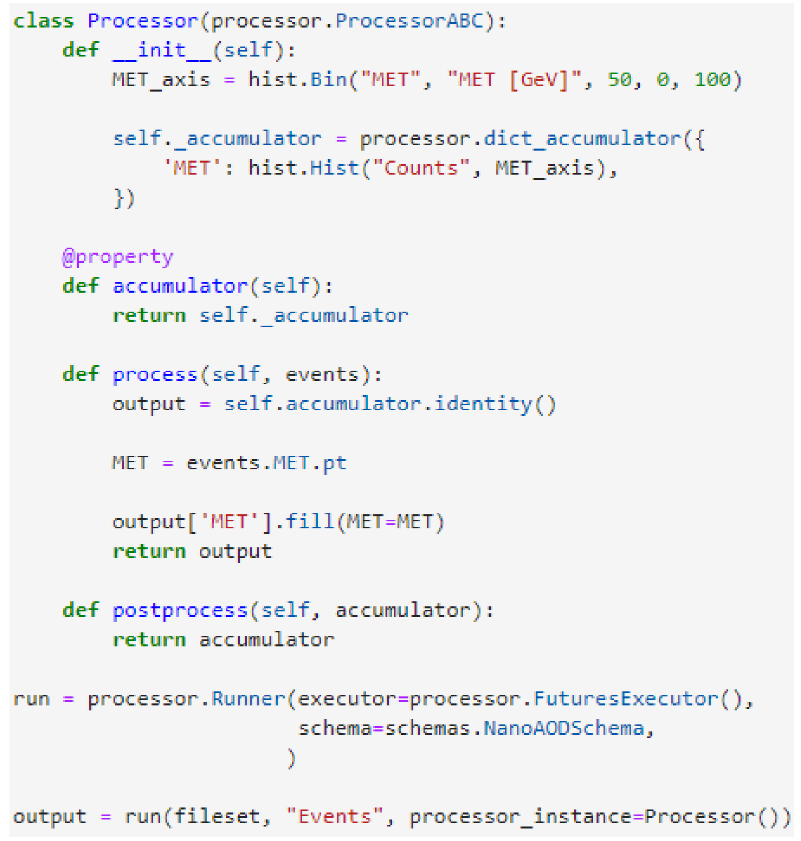


Image courtesy of Nick Smith, ACAT 2021.

# What is Coffea?

- Coffea analyses are written in a “Processor” class. This is where analysis is done.
- The Processor class gets deployed on an executor, which chunks up input data and feeds it in.
- Coffea has several executors. Coffea-Casa uses Dask.



```
class Processor(processor.ProcessorABC):
    def __init__(self):
        MET_axis = hist.Bin("MET", "MET [GeV]", 50, 0, 100)

        self._accumulator = processor.dict_accumulator({
            'MET': hist.Hist("Counts", MET_axis),
        })

    @property
    def accumulator(self):
        return self._accumulator

    def process(self, events):
        output = self.accumulator.identity()

        MET = events.MET.pt

        output['MET'].fill(MET=MET)
        return output

    def postprocess(self, accumulator):
        return accumulator

run = processor.Runner(executor=processor.FuturesExecutor(),
                       schema=schemas.NanoAODSchema,
                      )

output = run(fileset, "Events", processor_instance=Processor())
```

define histograms

process() runs per-chunk

columnar selection of relevant data

fill histograms

define an executor; Futures is for local runs!

run the processor, results go to output

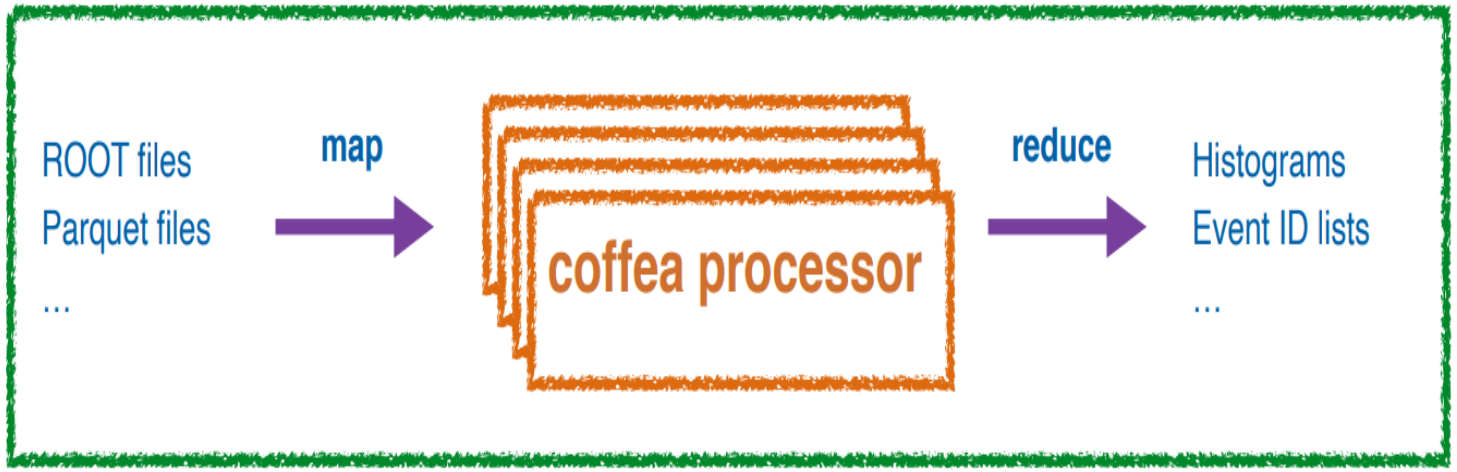


Image courtesy of Nick Smith, ACAT 2021.

Let’s run an example!

## Barriers to Coffea

- Learning a new analysis syntax.
  - Coffea can make analysis code simpler and more readable – but it *is* different!
- Environment setup – especially configuring your own Jupyter Notebook
  - Even when set up, locally-hosted Jupyter Notebooks can run into disconnection issues. This isn't great for long analyses.
- Working out deployment on various executors
  - Running locally works, but it's the least efficient solution.

# Barriers to Coffea

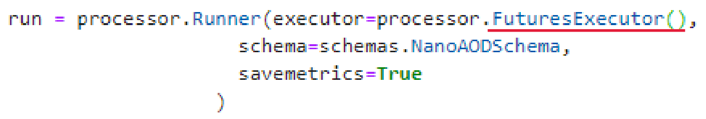
- Learning a new analysis syntax.
  - Coffea can make analysis code simpler and more readable – but it *is* different!
- Environment setup – especially configuring your own Jupyter Notebook
  - Even when set up, locally-hosted Jupyter Notebooks can run into disconnection issues. This isn't great for long analyses.
- Working out deployment on various executors
  - Running locally works, but it's the least efficient solution.

Coffea-Casa addresses these two barriers!

(…but you still have to learn coffea)

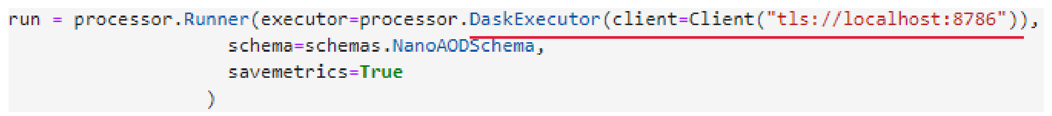
# Core Features of Coffea-Casa

- Coffea-Casa uses JupyterLab to bring Jupyter Notebooks to the cloud
  - Fewer worries about network stability while running long analyses!
- Tokens give access to CMS data without certificate set-up
  - Just replace the remote root filepath's redirector with xcache:
    - **root://xrootd.unl.edu//…** $\longrightarrow$ **root://xcache//…**
  - There is an [opendata instance]() for those outside of CMS; UChicago has an [ATLAS instance]().
- Dask executor runs out of the box
  - Just point to the scheduler that lives in every coffea-casa instance!



```
run = processor.Runner(executor=processor.FuturesExecutor(),
                       schema=schemas.NanoAODSchema,
                       savemetrics=True
                      )
```

$\longrightarrow$



```
run = processor.Runner(executor=processor.DaskExecutor(client=Client("tls://localhost:8786")),
                       schema=schemas.NanoAODSchema,
                       savemetrics=True
                      )
```

## QOL Features of Coffea-Casa

- xcache integration means your files will be cached and future analysis runs retrieve data faster
- Git integration in the UI
  - Your analysis isn’t “tied” to coffea-casa. Bring an existing coffea analysis over with minimal changes!
  - You can still access Git from the terminal if you want.
- ServiceX can be used on Coffea-Casa to deliver only the columns of data you’re interested in

## Let’s take a look!

## Ways to Improve

- Still in early stages; unexpected issues can arise
  - In my experience, the coffea-casa dev team has been great about resolving these in the past.
- Dependency management is a mess, mainly because solutions vary by use case
  - The solution for a pip-able dependency is *not* the solution for a local dependency in the analysis directory is *not* the solution for a local dependency in a different directory.
  - We’ve wrestled with this for a while. It’s uncertain when or how this will all be unified.
- There’s always a need for more workers and resources

## Want to Jump Right In?

- Documentation
  - Awkward (awkward-array.readthedocs.io/)
  - Coffea (https://coffeateam.github.io/coffea/)
  - Coffea-Casa (https://coffea-casa.readthedocs.io/en/latest/cc_user.html)
  - ServiceX (https://iris-hep.org/projects/servicex)

- Tutorials
  - Open either https://coffea.casa (for CMS) or https://coffea-opendata.casa (for opendata) to sign in.
  - Clone the coffea-casa-tutorials repository. (https://github.com/CoffeaTeam/coffea-casa-tutorials.git)
  - See the readme for how to navigate its examples!