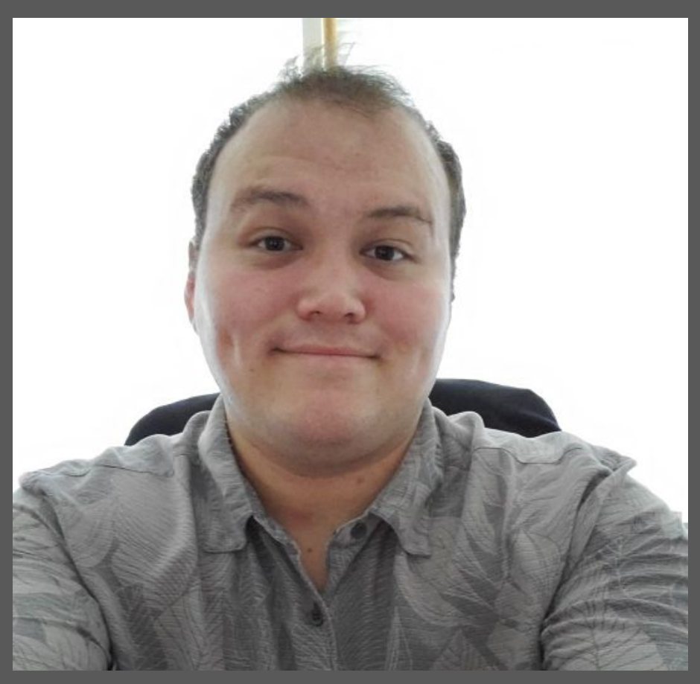
Keeping Things Consistent:

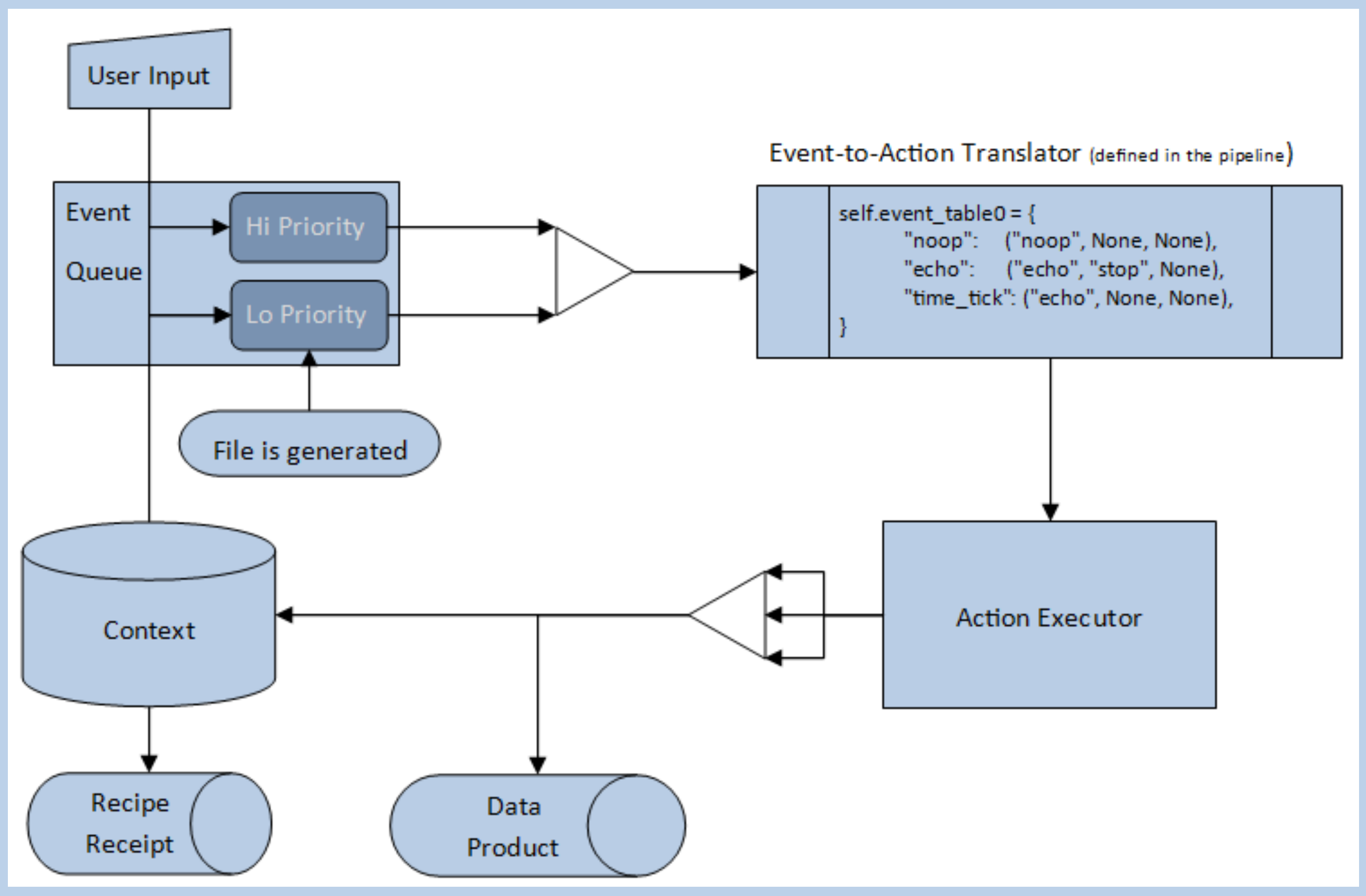
Developing a Framework to Unify Future W. M. Keck Observatory Data Reduction Pipelines

M. K. Brown, J. A. Mader, L. Rizzi, S. H. Kwok, J. Walawender (WMKO)



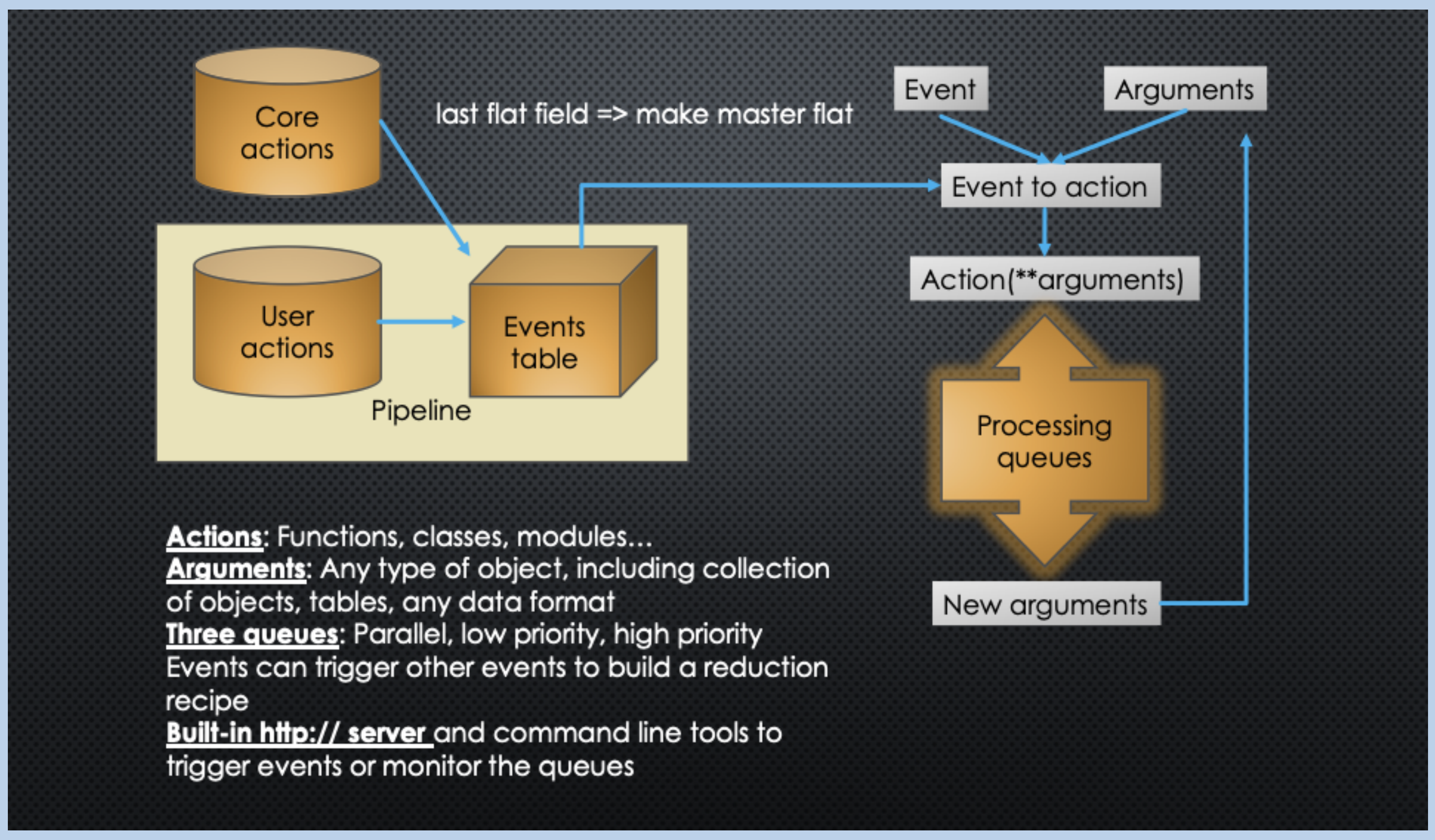
There are many wonderful instruments producing exciting and varied science that each have their own ways of reducing data even though the underlying processes are based on the same principles. This leads to effort wasted on rewriting code that has already been written, inefficient processes using archaic infrastructure, and increased complexity in trying to reproduce refereed work. This poster will analyze the effort to develop a generalized pipeline framework to facilitate the reduction of W. M. Keck Observatory data with a unified approach while allowing each instrument's unique curiosities to be worked into the reduction process. The modular approach will allow pipeline developers to reuse code for specific processes that are common among data reductions. The Framework will utilize many astronomy tools in Python in order to reduce the amount of code required to be written. Finally, the poster will examine the approach being taken to record the process history for scientific reproducibility.

Framework Process Workflow



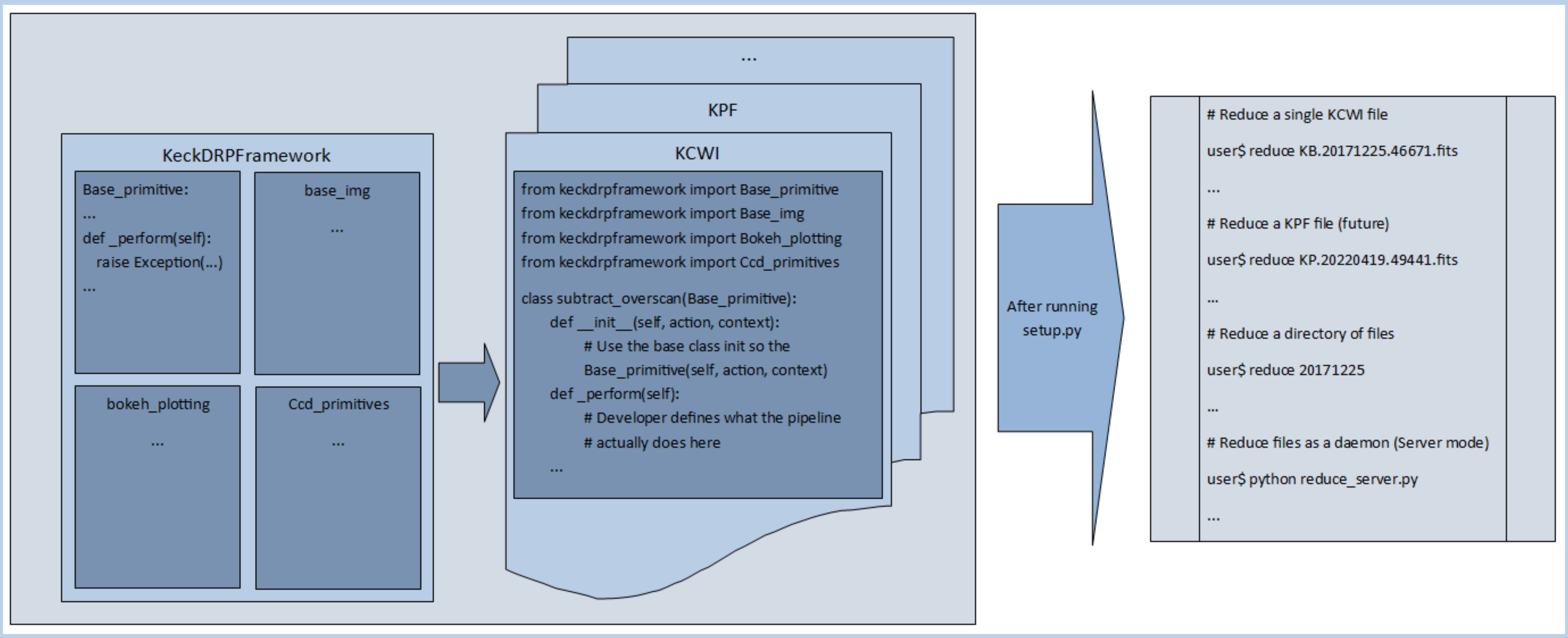
```
self.event_table0 = {
        "noop":    ("noop", None, None),
        "echo":     ("echo", "stop", None),
        "time_tick": ("echo", None, None),
}
```

Example of Recipe Transition



Actions: Functions, classes, modules…
Arguments: Any type of object, including collection of objects, tables, any data format
Three queues: Parallel, low priority, high priority
Events can trigger other events to build a reduction recipe
Built-in http:// server and command line tools to trigger events or monitor the queues

Framework Codebase Ecosystem



```
Base_primitive:
…
def _perform(self):
    raise Exception(…)
…
```

```
from keckdrpframework import Base_primitive
from keckdrpframework import Base_img
from keckdrpframework import Bokeh_plotting
from keckdrpframework import Ccd_primitives

class subtract_overscan(Base_primitive):
    def __init__(self, action, context):
        # Use the base class init so the
        Base_primitive(self, action, context)
    def _perform(self):
        # Developer defines what the pipeline
        # actually does here
    …
```

```
# Reduce a single KCWI file
user$ reduce KB.20171225.46671.fits
…
# Reduce a KPF file (future)
user$ reduce KP.20220419.49441.fits
…
# Reduce a directory of files
user$ reduce 20171225
…
# Reduce files as a daemon (Server mode)
user$ python reduce_server.py
…
```

The main goal of the pipeline framework is to reduce the effort needed from the organization to maintain pipeline support for users who may not be regular users of WMKO data. By having a single template for what a pipeline should look like, debugging the application becomes much easier for WMKO staff who might need to support it. It also makes execution of pipelines easier for users as well. With a single entry point for reduction with multiple pipelines, end users only need to learn a single tool for all of WMKO data reduction! In order to facilitate this, a common set of base classes were created that allow developers to write new modules for new instruments that can easily be slotted into the pipeline. Since the framework only calls these predefined functions, if the pipeline is written properly it will always be called properly. However, since only the handle is pre-defined, the pipeline developers have ultimate control over what happens in the execution of the function. Moreover, since the Framework is constantly updating a global state of the reduction in the context, it makes it easy to trace the steps taken in order to verify the results.

If you want to learn more, you can check out the code at https://github.com/Keck-DataReductionPipelines/KeckDRPFramework
For an example implementation, check out https://github.com/Keck-DataReductionPipelines/KCWI_DRP