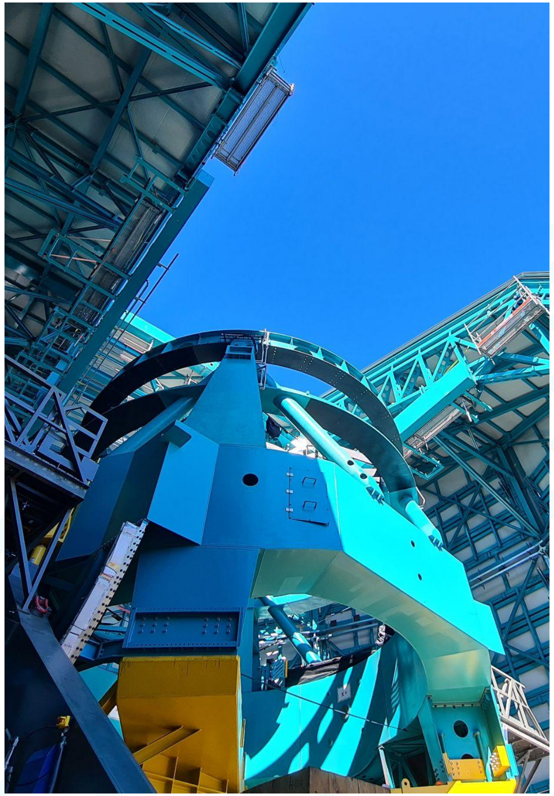
Rubin Observatory Data Processing Workflows

Colin Slater
Rubin Data Management,
Deputy Subsystem Scientist

Rubin Background

- The observatory collects 3.2 gigapixel images, 1000 new images per night, estimated ~200 nights per year, for 10 years.
- 20 TB raw per night, turns into *larger* volume once processed.
- We process every image within 60 seconds to detect transient and moving objects.
- Once per year, we process all images to date into calibrated images, coadded images, difference images, and measurements of all the sources on all of the images and coadds.



Software Stack

- Pipelines are in Python, with C++ for critical “inner loop” components/image manipulation
- “Tasks” are the basic building block, e.g. CalibrateTask, AssembleCoaddTask, AstrometryTask
- Tasks declare the types of data they need as input and produce as output. Forbidden from doing their own IO.
- The [data] Butler provides abstraction for accessing the data, based on rigidly-defined key-value dictionary (e.g. {visit=1234, detector=1})
- Tasks are called by an executor. The executor gets data from the butler, passes it to the task, then stores the outputs via the butler.

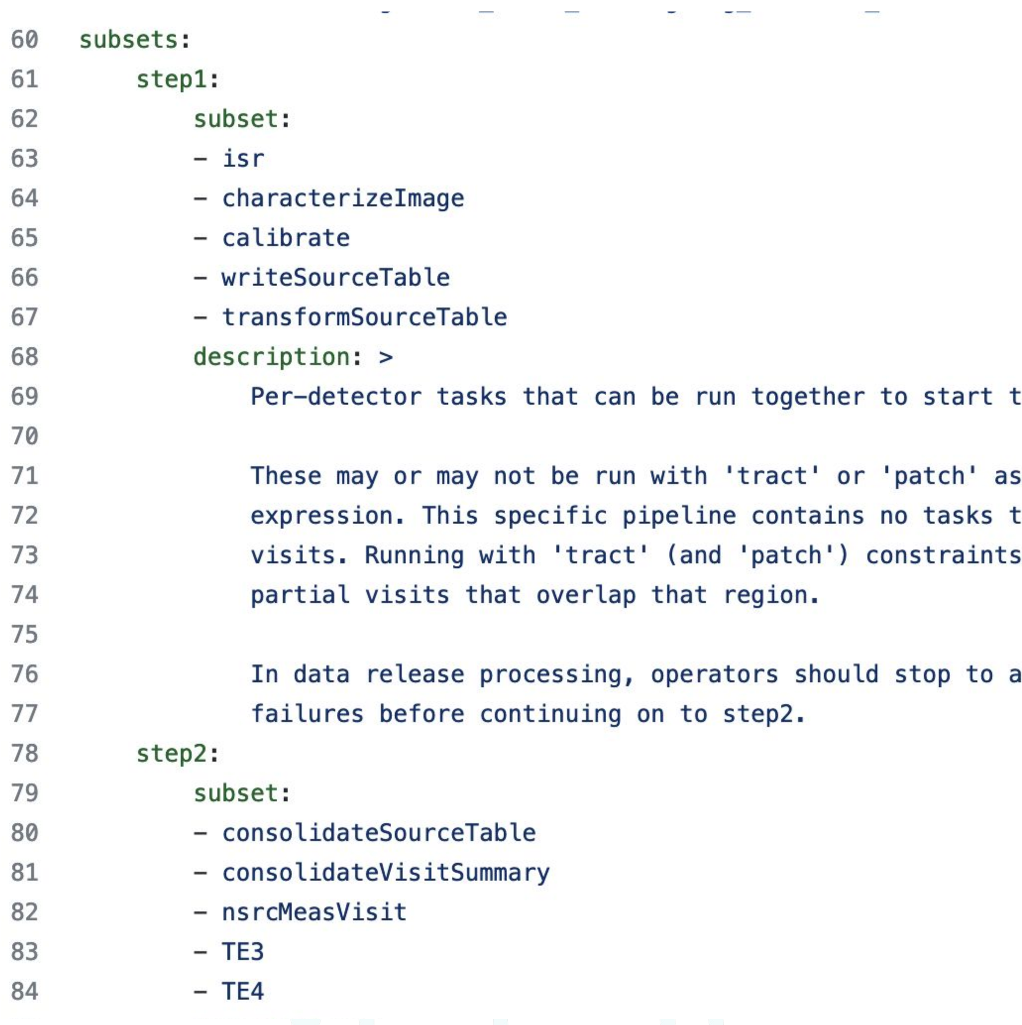
Software Stack

- Because tasks declare their inputs and outputs, the definition a “pipeline” of multiple tasks is just a list of task names.
- “Quantum Graph Builder” takes the list of tasks, plus a constraint on the input data (e.g. visit number between 1000 and 2000), creates a directed acyclic graph of the all of the quanta to be executed.
  - “Quantum” = an instance of a specific task, to be run on a specific input dataset)
- The graph builder uses the Butler’s database of what data already exist to know what work is required and what is already done.
- Resulting DAG is designed to be executor-agnostic; just need an interface layer to take care of IO. Have used HTCondor, Parsl, and Panda all successfully.

VERA C. RUBIN OBSERVATORY

YAML files define which tasks are invoked in each step

https://github.com/lsst/drp_pipe/tree/main/pipelines



```yaml
subsets:
    step1:
        subset:
        - isr
        - characterizeImage
        - calibrate
        - writeSourceTable
        - transformSourceTable
        description: >
            Per-detector tasks that can be run together to start t

            These may or may not be run with 'tract' or 'patch' as
            expression. This specific pipeline contains no tasks t
            visits. Running with 'tract' (and 'patch') constraints
            partial visits that overlap that region.

            In data release processing, operators should stop to a
            failures before continuing on to step2.
    step2:
        subset:
        - consolidateSourceTable
        - consolidateVisitSummary
        - nsrcMeasVisit
        - TE3
        - TE4
```

- So, just build a giant DAG for all of the data and all of the tasks in a data release, and then hit “go”?
- Not quite, we only make graphs of small subsets of the overall data release:
    - Same tasks, on different groups of data in parallel
    - Different types of tasks separated by global sequence points
- Even though we could theoretically make a huge graph, we humans need to be able to understand and manage the processing.
- Easier to partition the problem and create graphs in those partitions.

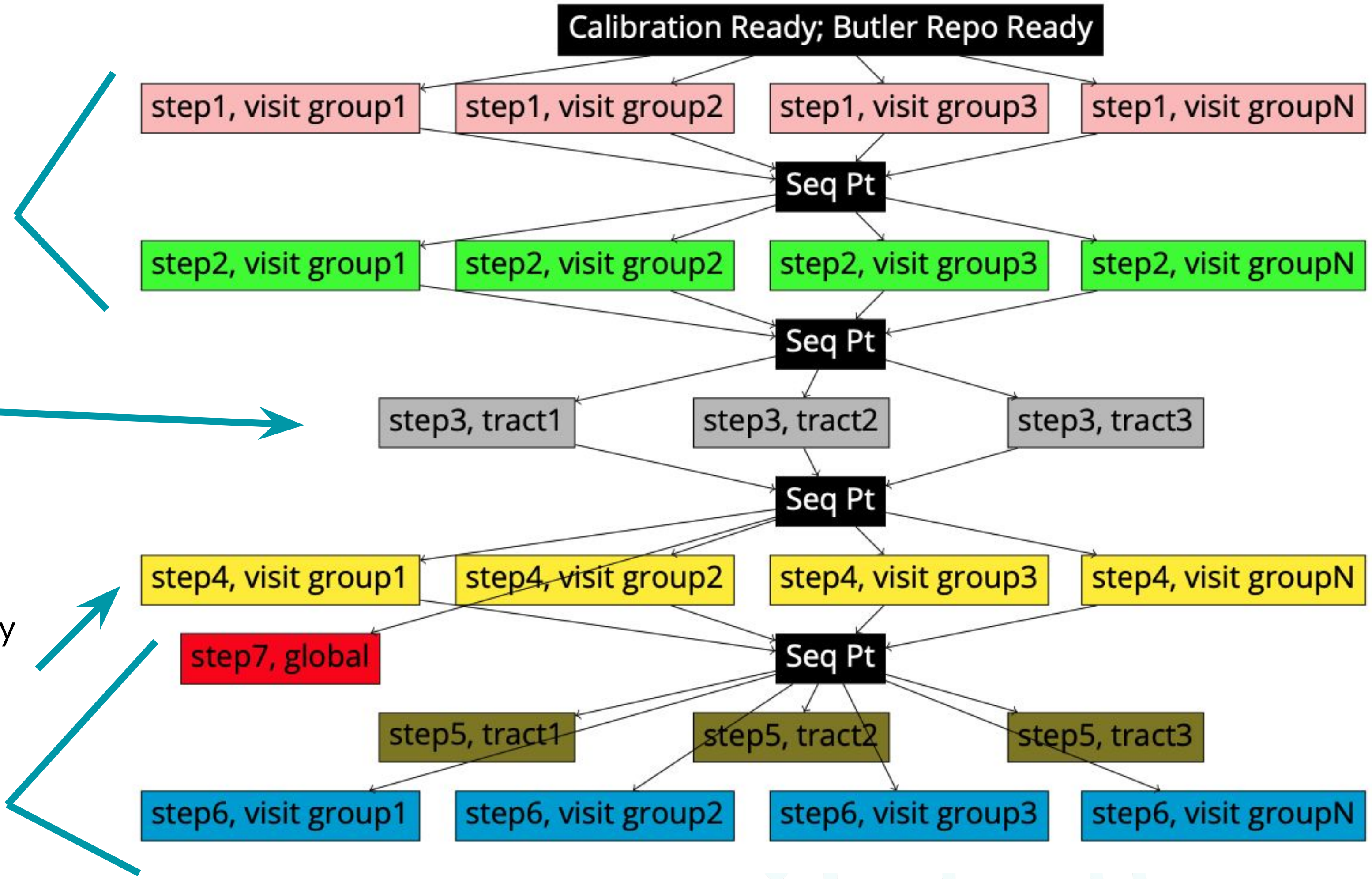
VERA C. RUBIN OBSERVATORY

Single Frame Processing
~100M Jobs, ~minutes each

Coadds
~100k jobs, ~hours each

Force Photometry & Image Differencing

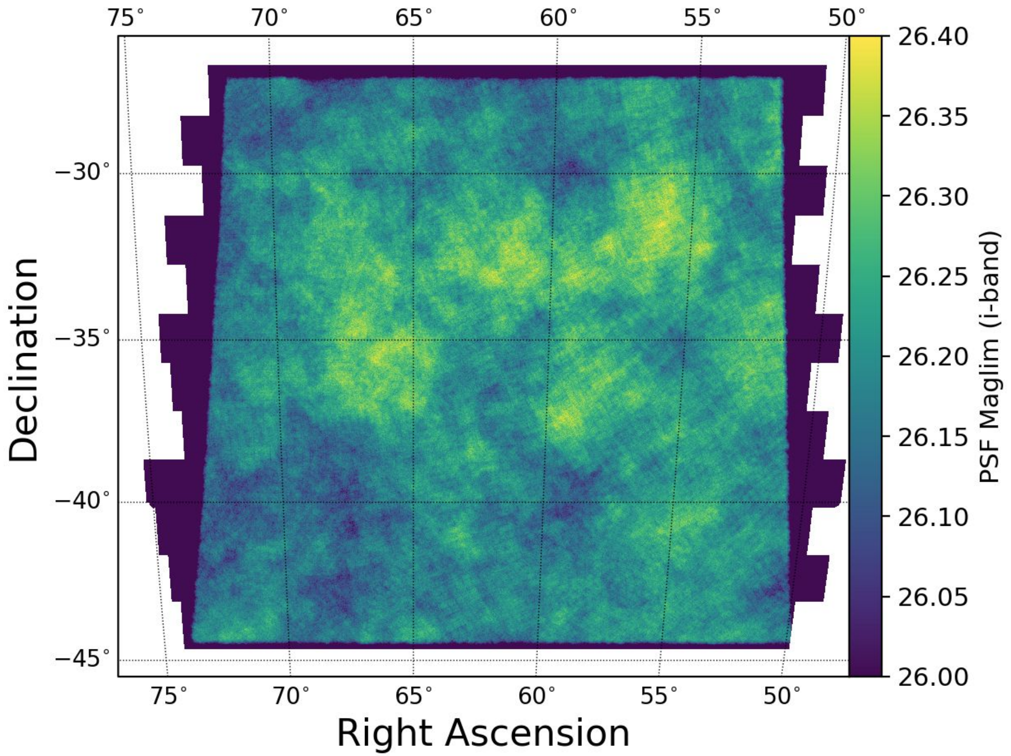
Aggregations



“In-production” on precursor data

- We’ve used a large dataset of simulated images as a testbed, prior to real on-sky data.
- 300 sq degrees, ~22k visits, ~10% of the data volume of year 1.
- Processed at Google, using Panda for executing individual workflows. Total of ~2.5M core hours.

Simulation reference for DESC DC2:
https://arxiv.org/abs/2010.05926



Future target is multi-site

- “Real” data releases will have processing split between SLAC (25%), France (CC-IN2P3, 50%), and the UK (RAL/Lancaster, 25%)
- Currently developing the plan to manage this, so some speculation follows:
    - We’d like each workflow to run in a single datacenter
    - Initial assumption is to roughly partition the sky 25%/50%/25%; each workflow can be compact on the sky.
    - Some data shuffling is required, but it can probably be done during global sequence points.

Multi-site is tricky

- Complex part is that we currently have a monolithic database that knows about every data product
- "Graph building" could all be done at a central site, then the resulting DAGs distributed to different sites, with some data URL translation to allow accessing site-specific
- Rucio used to manage transfers, both of data that gets replicated everywhere (e.g. calibration) and data that needs to be brought back to SLAC. Need to also transfer the database entries describing those datasets.

Details available at: https://dmtn-213.lsst.io/

Future wish-list

- We’re working on the workflow-of-workflows management software right now.
- We have some tricks for avoiding lots of very small batch jobs, but it’s a bit ad hoc
  - Could imagine more general solutions for equalizing allocation lengths
- Have not measured efficiency — Are these very big DAGs causing excessive overhead?
  - Our DAGs are very big when fully-enumerated with datasets, but structurally pretty simple and repetitious.
- Still have to scale by 10x for the real data — Always a worry!