Pegasus Users Group

MEETING

Running Rubin LSST Science Pipelines on AWS

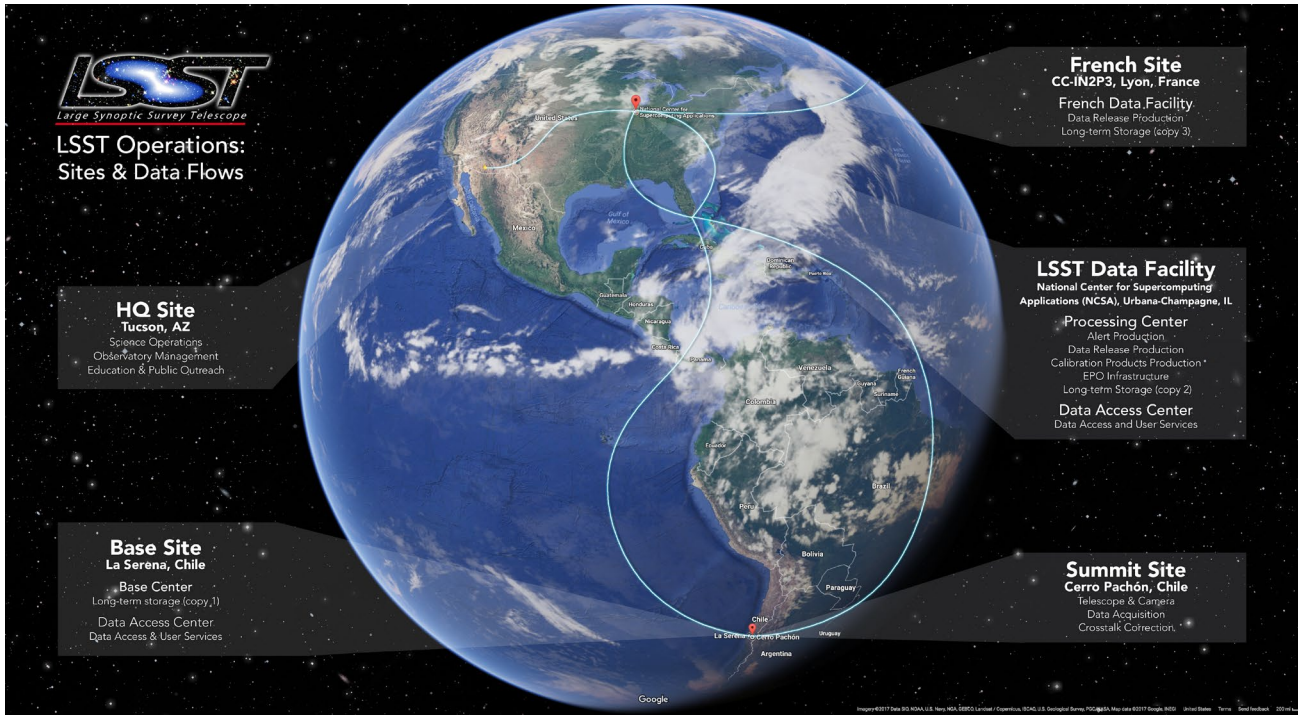
Dino Bektesevic

Department of Astronomy, University of Washington

February 23 , 2021

Vera C. Rubin Observatory
Legacy Survey of Space and Time

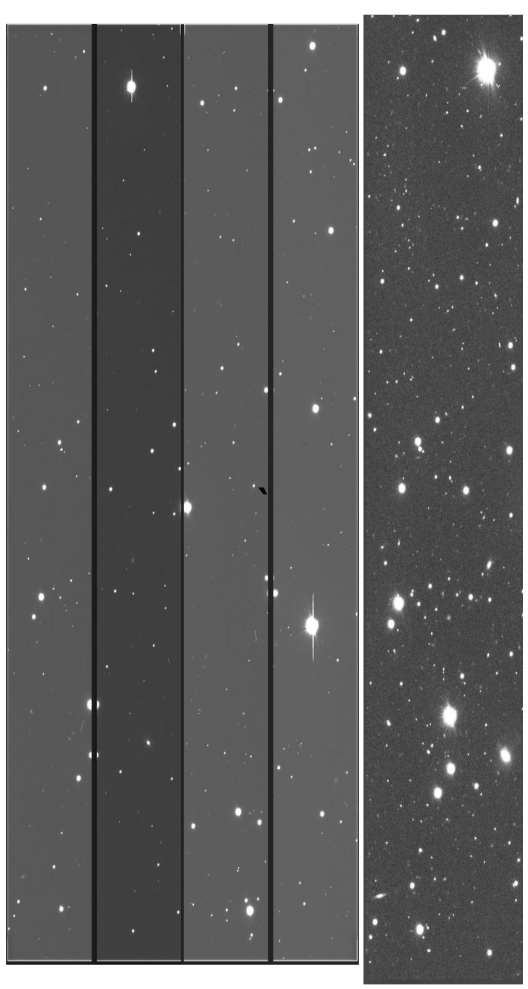
- 8m telescope under construction in Chile
- 10-year survey of the sky
  - visible southern hemisphere imaged every 3 nights
  - 20 TB of data nightly, 200 petabytes total

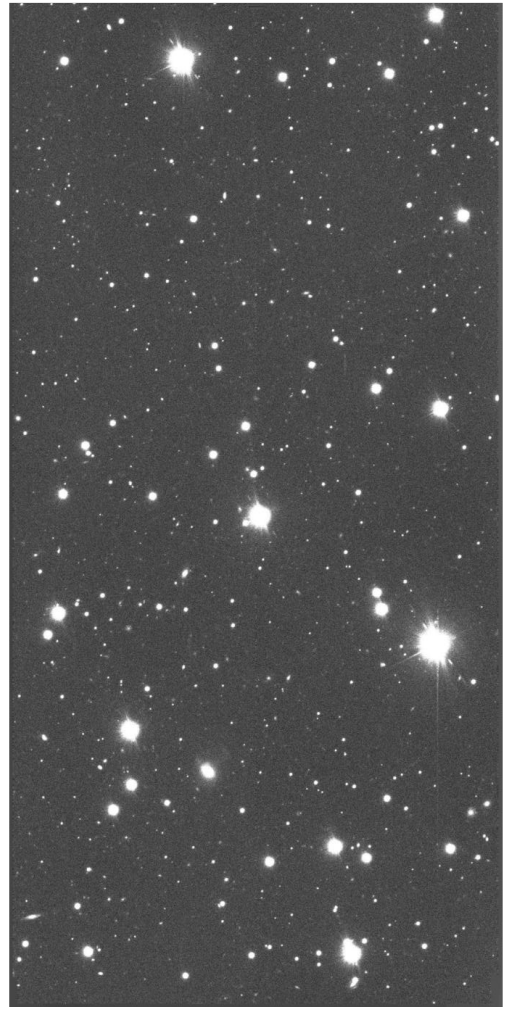


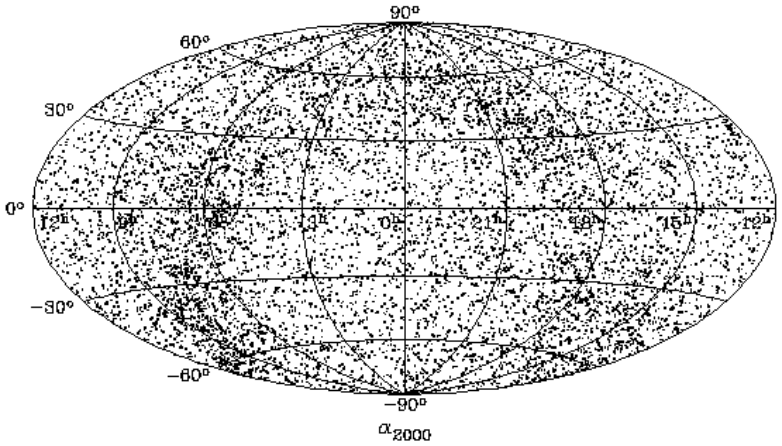
Rubin on AWS

- Can we build a system to analyze data at the scale of Rubin?
  - Can this be used to speed up science?
  - Would it be affordable?
  - Easy to use?
  - Would it support doing an analysis not provided by Rubin
- Start smaller
  - A nights worth of data
  - At a reasonable cost

Rubin LSST Science Pipelines





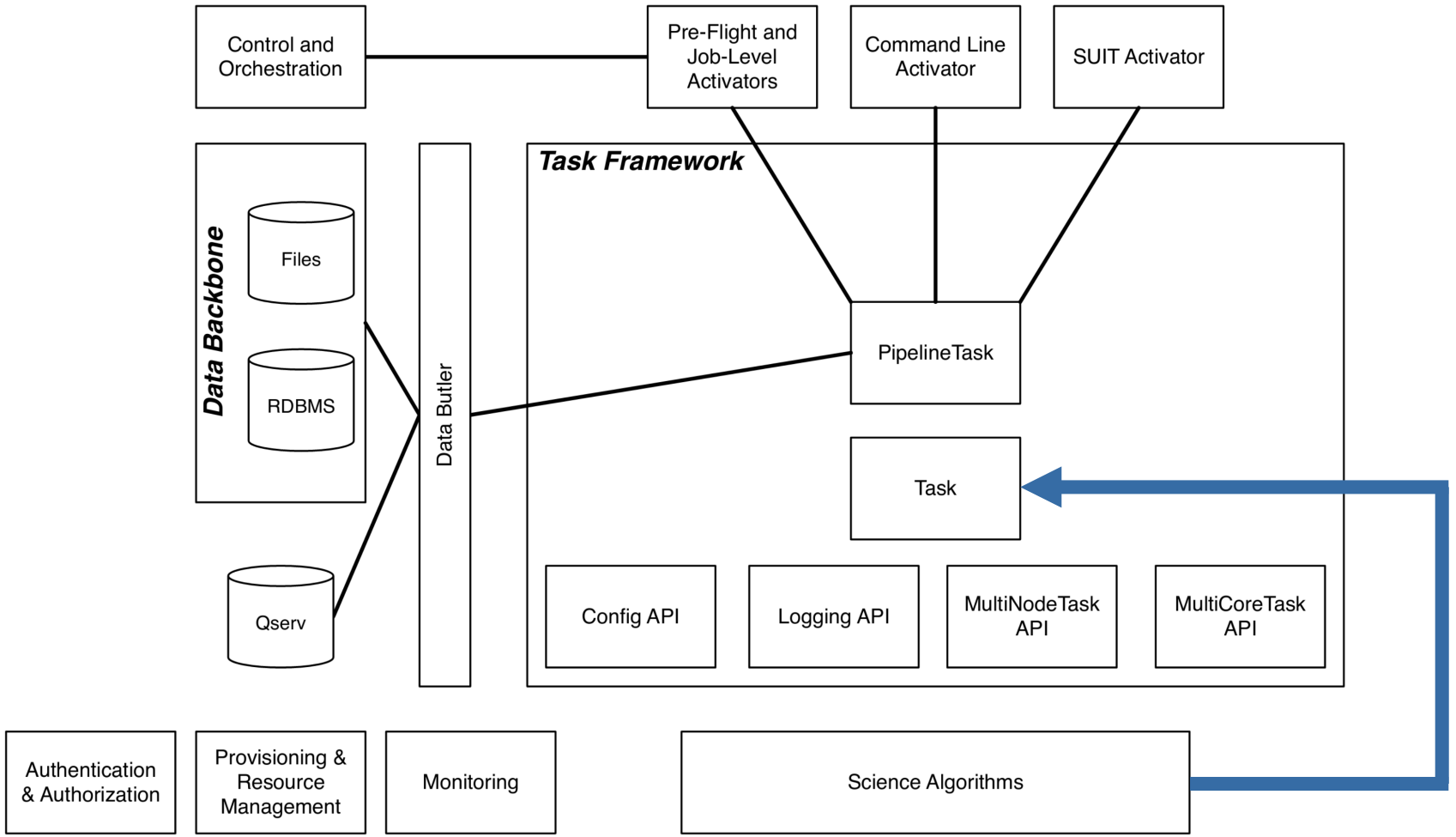


Raw exposure

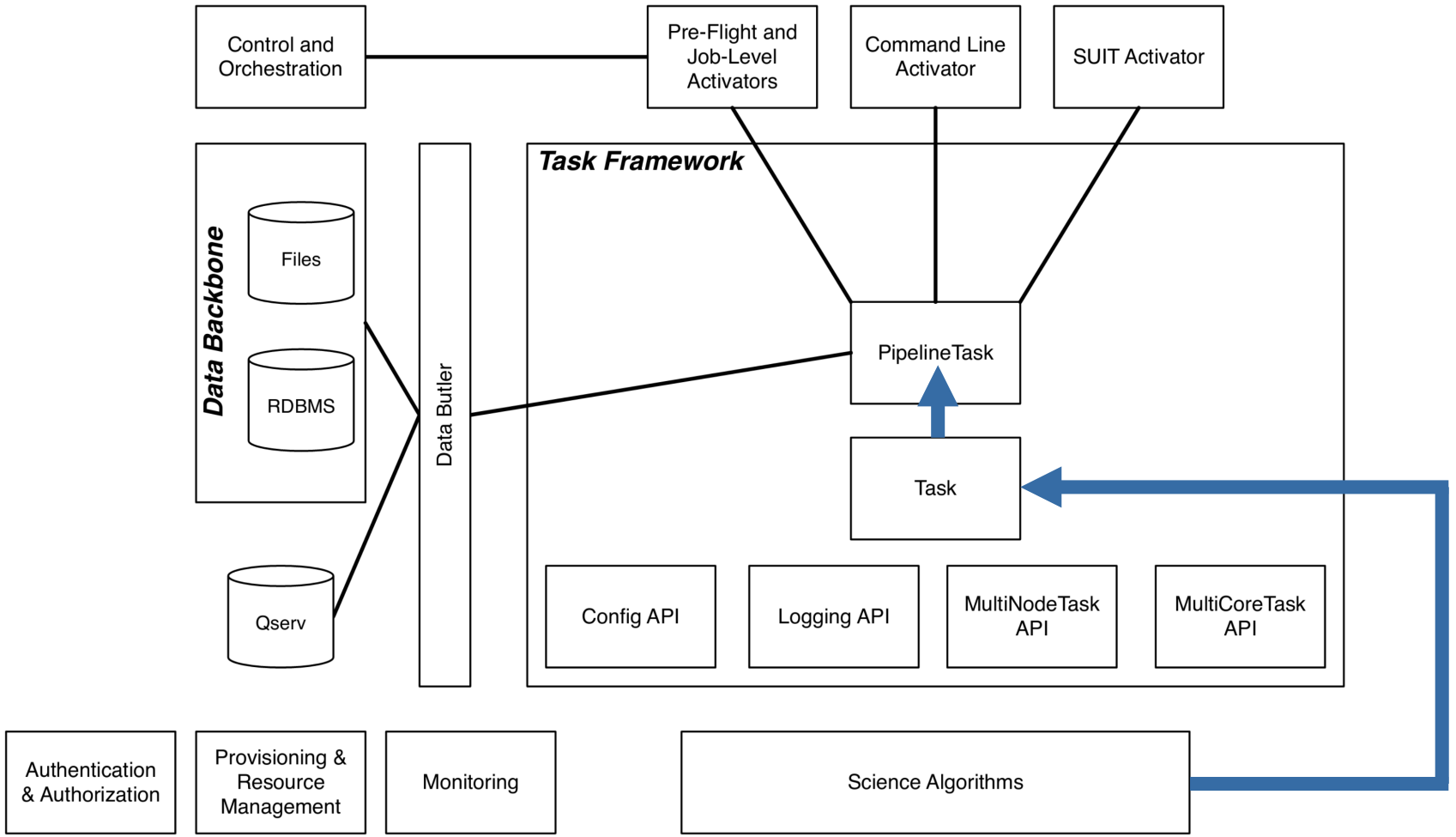
Processed exposure

Object catalog

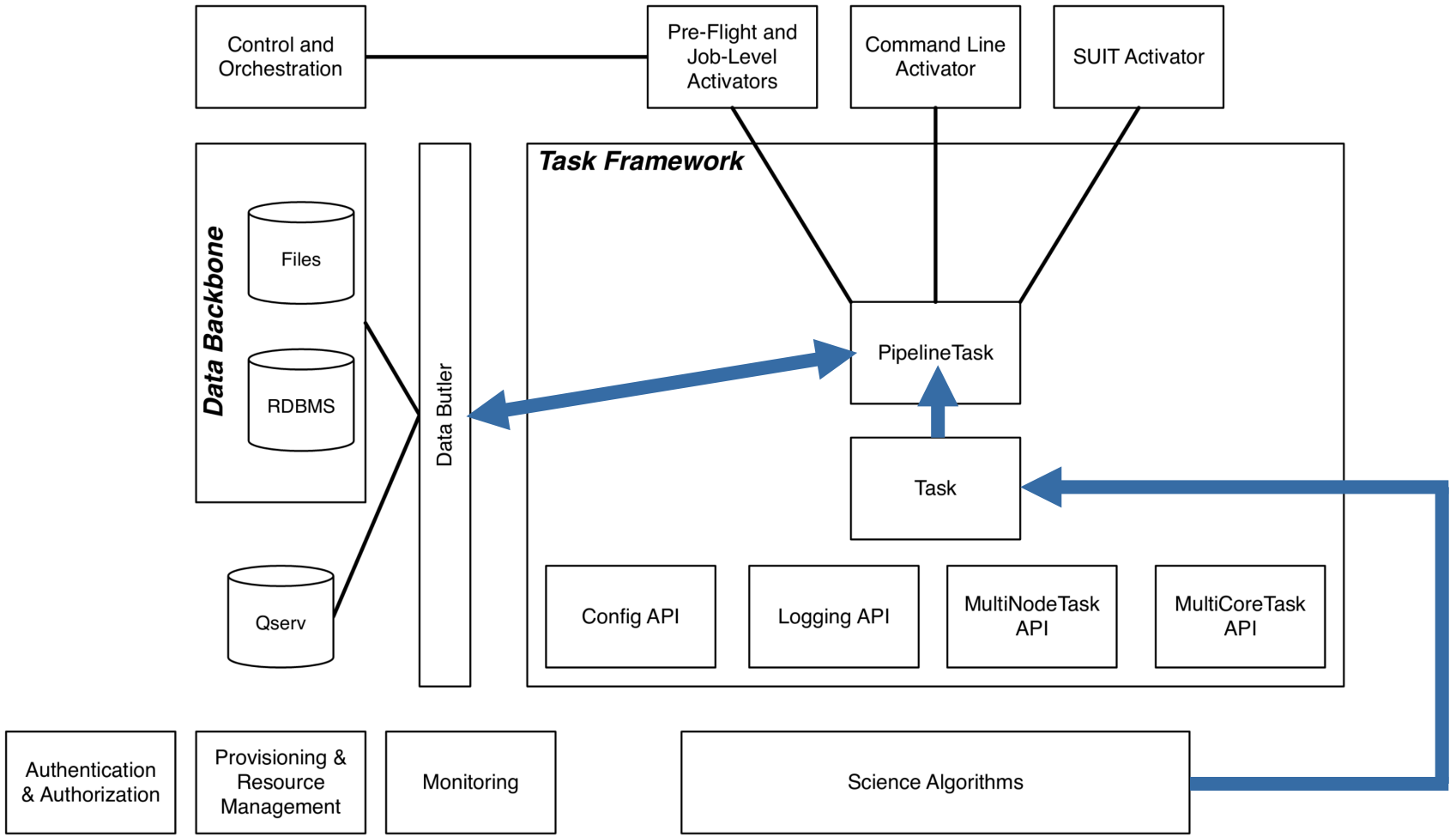
Rubin LSST Science Pipelines



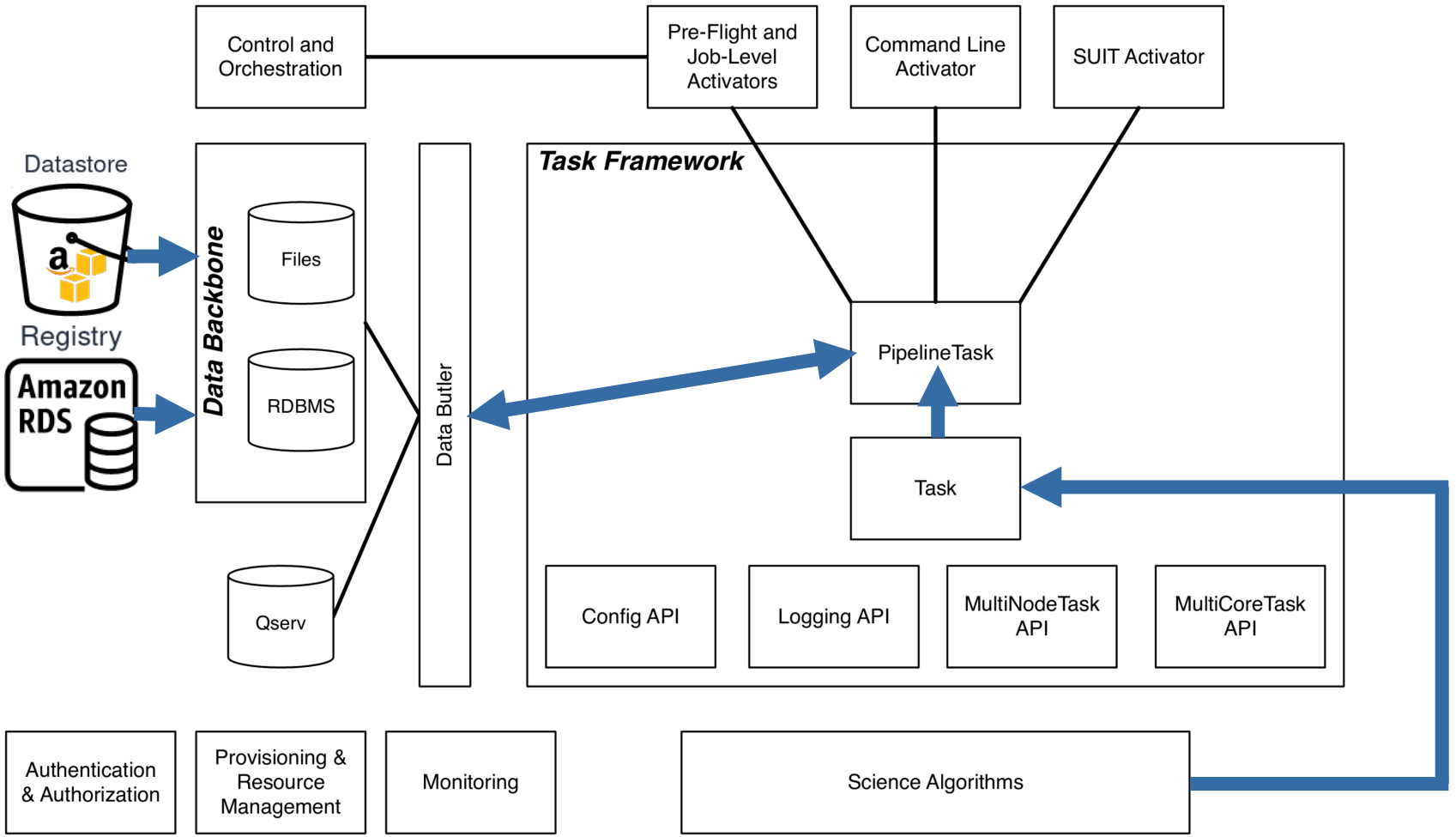
Rubin LSST Science Pipelines



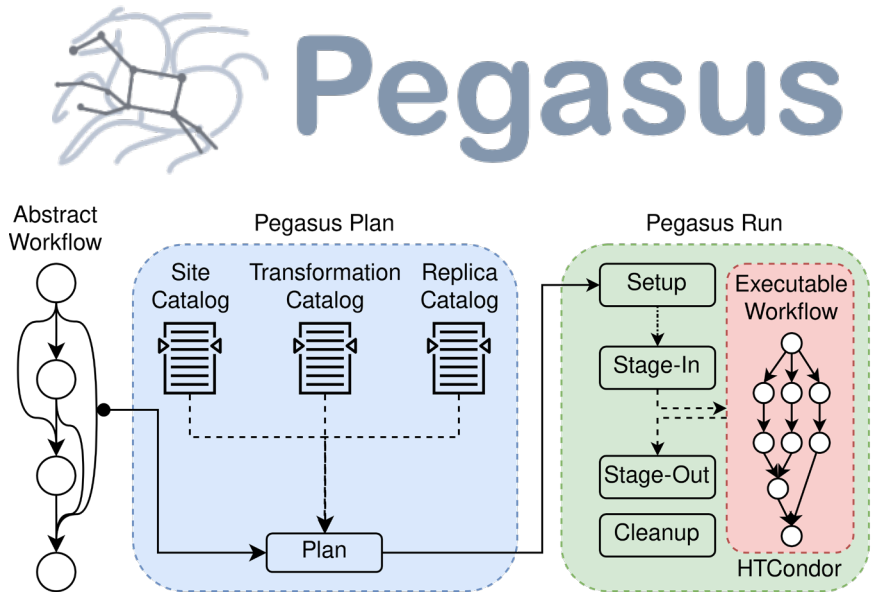
Rubin LSST Science Pipelines

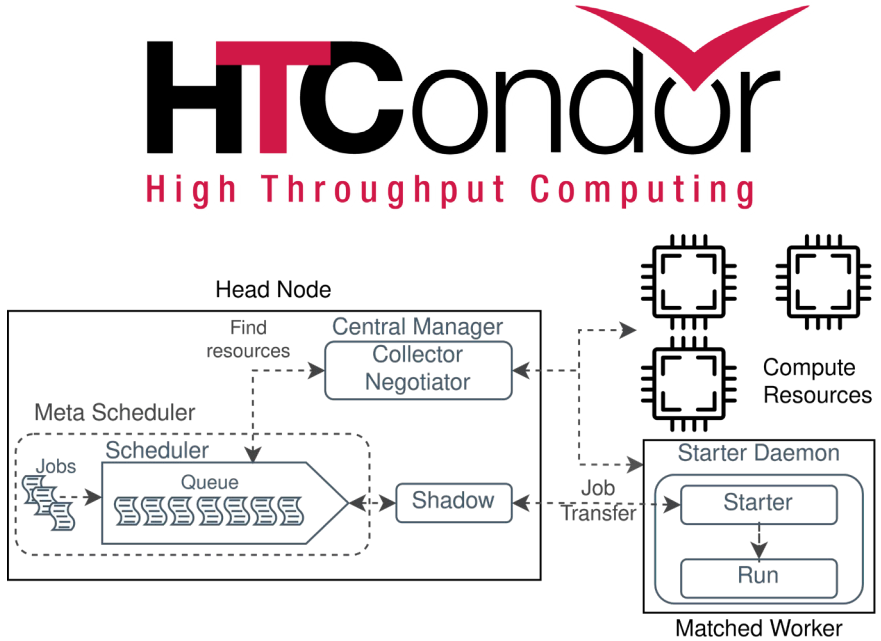


Rubin LSST Science Pipelines

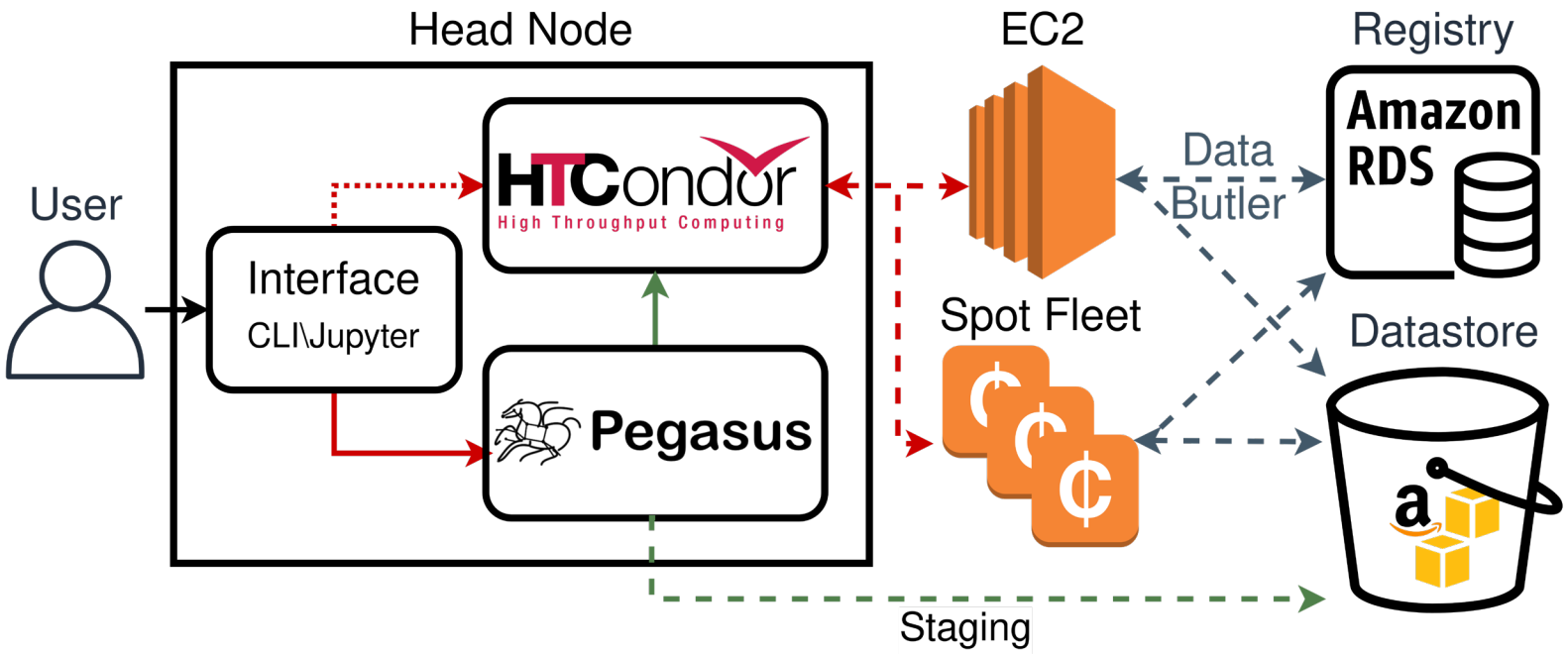


Scaling in the cloud

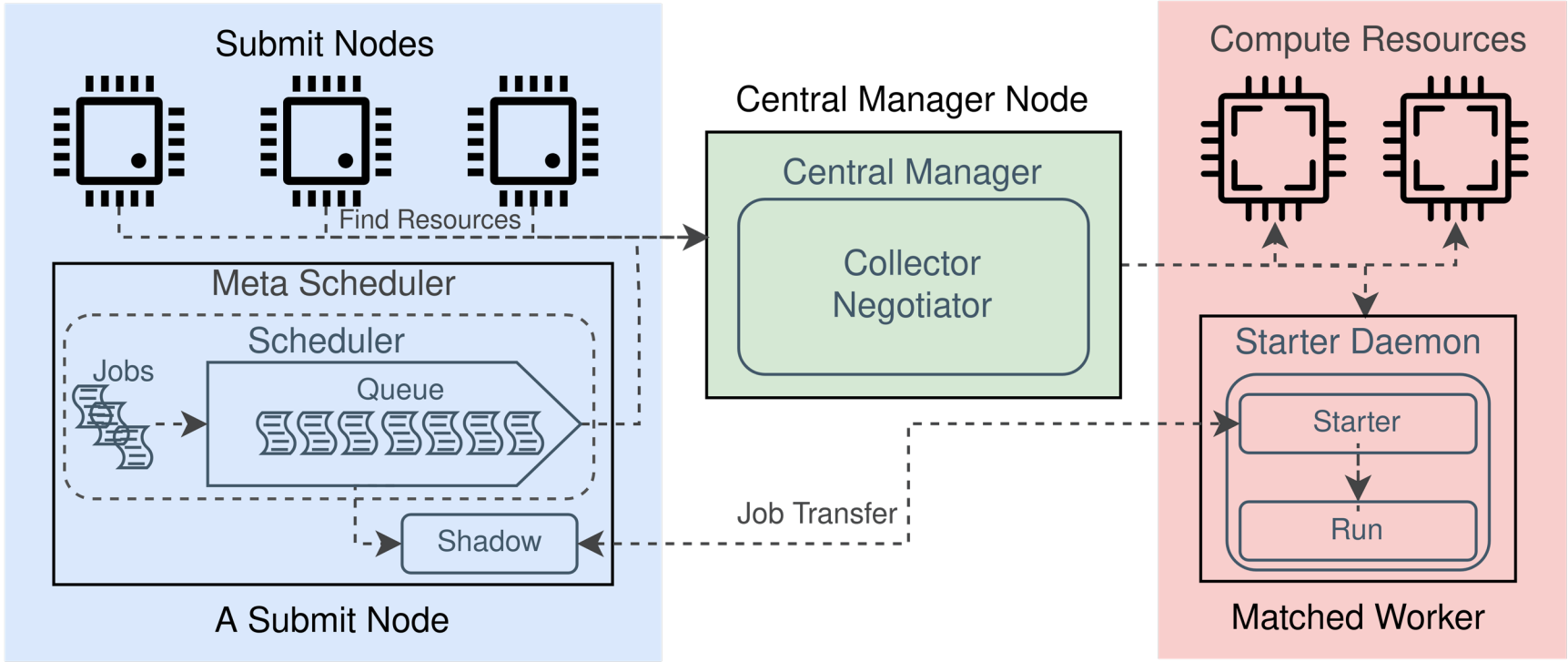




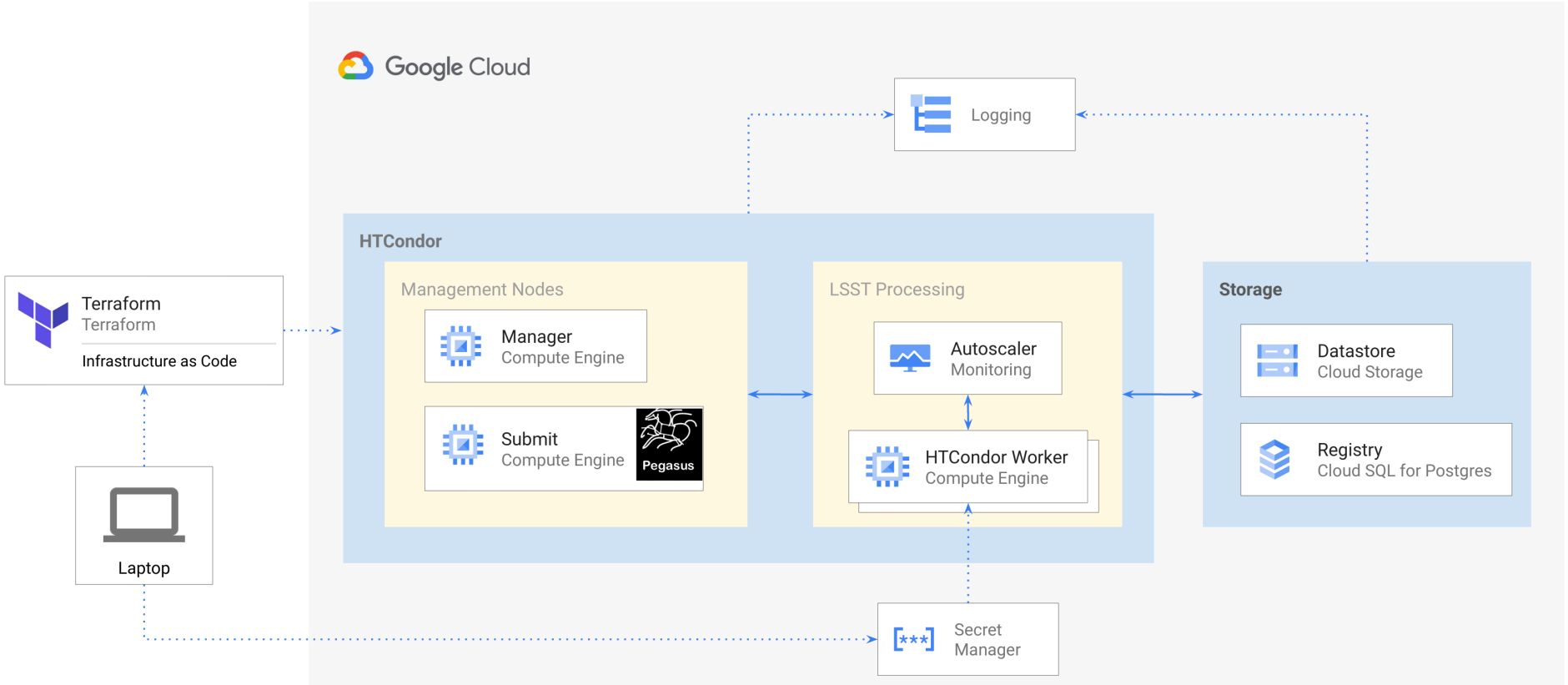
Basic infrastructure



Centralized architecture?



Different cloud providers?



https://www.youtube.com/watch?v=CQgLEWZ_E1c

Workflow?

```
description: cp_pipe DARK calibration construction
tasks:
  isr:
    class: lsst.ip.isr.isrTask.IsrTask
    config:
      connections.ccdExposure: 'raw'
      connections.outputExposure: 'cpDarkIsr'
      doBias: True
      doVariance: True
      doLinearize: True
      doCrosstalk: True
      doDefect: True
      doNanMasking: True
      doInterpolate: True
      doBrighterFatter: False
      doDark: False
      doFlat: False
      doApplyGains: False
      doFringe: False
  cpDark:
    class: lsst.cp.pipe.cpDarkTask.CpDarkTask
    config:
      connections.inputExp: 'cpDarkIsr'
      connections.outputExp: 'cpDarkProc'
  cpCombine:
    class: lsst.cp.pipe.cpCombine.CalibCombineTask
    config:
      connections.inputExps: 'cpDarkProc'
      connections.outputData: 'dark'
      calibrationType: 'dark'
      exposureScaling: "DarkTime"
      python: config.mask.append("CR")
contracts:
  - isr.doDark == False
  - cpCombine.calibrationType == "dark"
  - cpCombine.exposureScaling == "DarkTime"
```

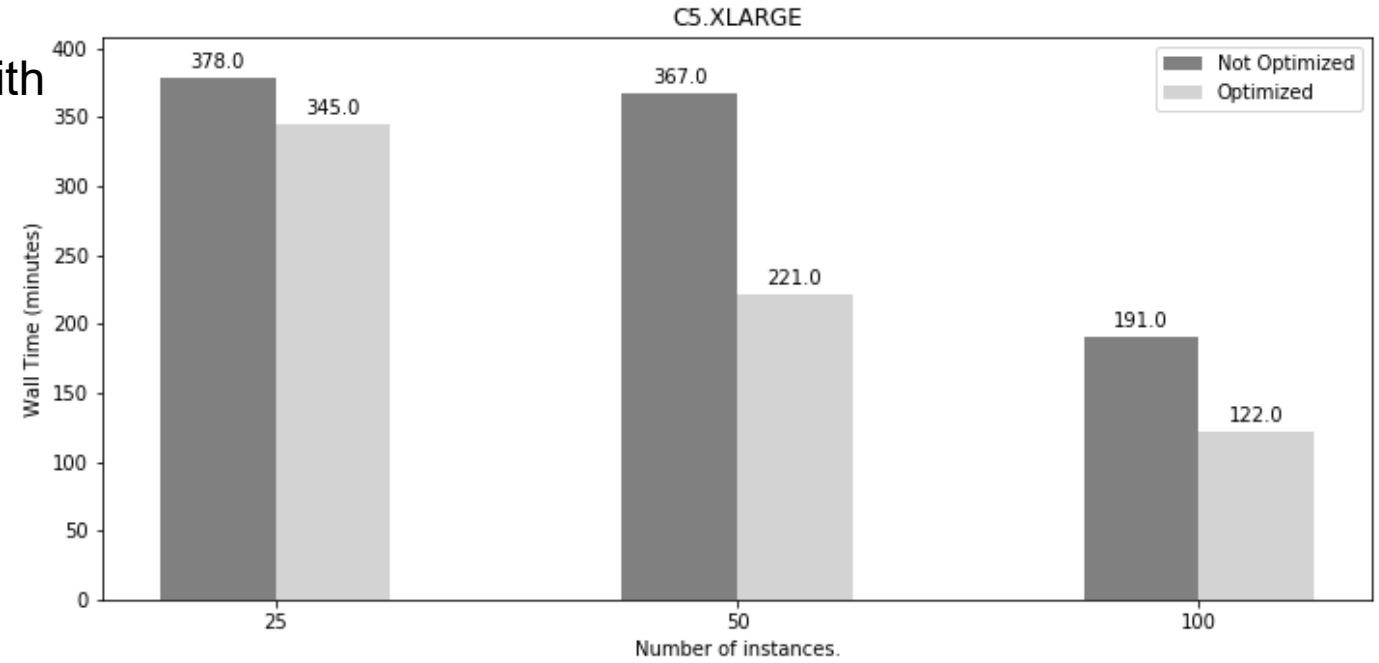
pipetask build
--pipeline …
--qgraph ...

```
def generateDax(f, name="dax", noInitJob=False, initPickle=None):
    """Generate a Pegasus DAX abstract workflow"""
    dax = peg.ADAG(name)

                    •       •       •

    for line in f:

                    •       •       •

            job.addProfile(peg.Profile(peg.Namespace.CONDOR, "request_cpus", "1"))
            if taskname in demandingTasks:
                job.addProfile(peg.Profile(peg.Namespace.CONDOR, "request_memory", "28GB"))
            else:
                job.addProfile(peg.Profile(peg.Namespace.CONDOR, "request_memory", "2GB"))

                    •       •       •

            logfile = peg.File("log.%s.%06d.out" % (taskname, iq) )
            dax.addFile(logfile)
            job.setStderr(logfile)
            job.uses(logfile, link=peg.Link.OUTPUT)

            dax.addJob(job)

                    •       •       •
```

1. Pipeline definition
2. Plan a Quantum Graph
3. Convert to DAX
4. Profit??

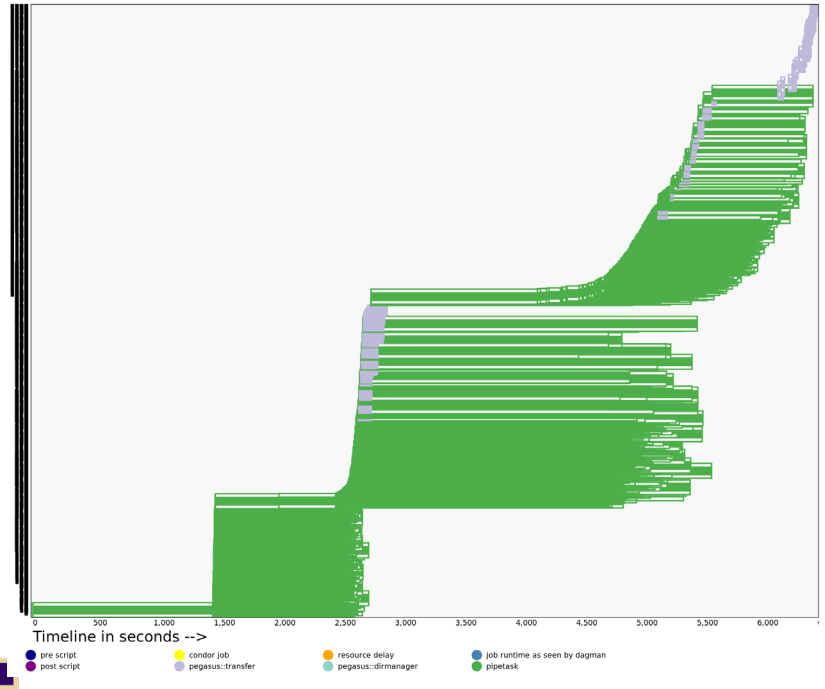
Optimize execution

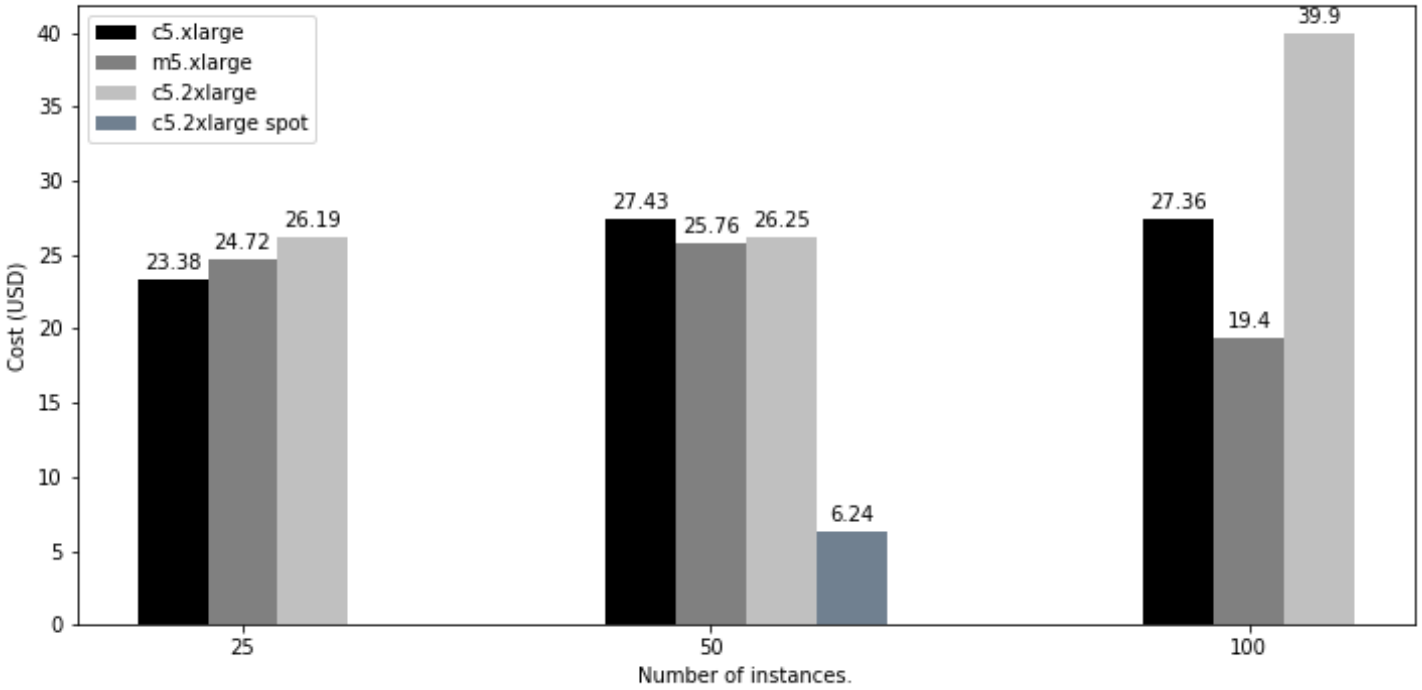
- Hyper-Suprime Cam dataset, ~2\% of nightly data volume.
  - 6787 images, 20 361 jobs
- At scale, PostgreSQL registry has to handle many simultaneous connections.
- Careful job requirements configuration during Quantum Graph to DAX conversion
- Avoid using the head node as a staging site
  - Add an S3 Bucket in Site-Catalog locations
  - Use a *nonsharedfs* file system in tandem with a Replica-Catalog
- Job clustering
  - reduces time lost on log transfers
  - Maximizes CPU usage
- Pick an appropriate instance type and size



Cost estimates

- Cost is ~constant
- Compute resources are competitive to on-premise resources
  - Especially so for small-to-mid workflows with short individual jobs
- Storage remains use-specific





What next?

- Infrastructure as code – Packer, Terraform scripts
  - Advanced deployements
  - https://github.com/astronomy-commons/RubinAWS
- Integration
  - with Jupyter Notebooks:
    - Demo notebooks with prepared data sources
    - Integration of cluster monitoring and workflow execution tools with the work environment (Workflow statistics via the API and not CLI)
  - With other OSs
  - Of Pegasus with Rubin Workflow Execution
  - Of Pegasus Dashboard with User Nodes
  - Pegasus Hooks are exciting (no idea what to use them for, yet)
- Registry and Datastore optimizations
  - Centralized Registry sets a maximum upper limit to scalability
  - Datastores never saw any love (no caching, no staging, no tiered storage)

Our Pegasus Feedback

- Better CWL support (why not native?)
- 5.0 Brings a Python API for many features that were previously CLI only – but their integration with Pegasus is odd and so is using them
  - StampedeStatistics vs StampedeWorkflowStatistics
    - No tutorial, or too hard to find one
    - Both have a code comment “main stats class”
    - behavior can be unexpected with default params (i.e. “expanded_workflow=True” but that doesn’t look like a very important parameter in the docstring directly below it or above, it’s mentioned but not explained)
    - Behavior can be unexpected because they’re so state-full (forgetting to turn filters on/off)
    - these packages are not standalone even when their dependence on main-body Pegasus is minimal
- Better/Different ways to handle clustered statistics? It seems near impossible to preserve job metadata after clustering jobs.
- Better log handling (perhaps Kafka, Apache DistributedLog, CloudWatch etc. APIs?) would make shared-nothing scaling much easier to maintain and debug.
- Ability to rewrite the DAG after a partial execution.

Thank You!