Pipelines & Workflow in LSST-DESC

Joe Zuntz

Overview

- LSST & DESC
- 3x2pt
- Pipelines

LSST

LSST

- Large Synoptic Survey Telescope

- 8.4m mirror
  3.2 Gpix camera
  10 year main survey
  37B sources
  0.5 Exabytes total
  3.5 degree field of view
  6 colours
  18,000 sq. deg survey

- Under construction in Cerro Panchon, Chile

- Can image the entire sky every 3 days

- First science light 2021

LSST Science

- Transients
  - 90% of solar system objects >140m
  - 10^6 asteroids, 10^4 objects beyond Neptune
  - 10M alerts/night within 1 minute of a changed object
- Milky Way
  - 10^{10} Main Sequence stars to 100 kpc
  - Milky Way Survey volume ~1000 greater than previous surveys
- Cosmology
  - ~10^9 **weak lensed** galaxies, ~10^6 Supernovae,
    10^4 galaxy-galaxy strong lenses,

LSST Construction Status

Parts of LSST:
The Project

- Builds & operates the telescope!

- Data Management group (DM)
  Runs Level 1 and Level 2 processing
    - Level 1: prompt data products, such as transients and difference images
    - Level 2: Annual releases of calibrated exposure and co-added images, catalogs of sources

- Building *DM Stack* software framework to run analyses at scale
    - See https://pipelines.lsst.io/

Parts of LSST: Collaborations

- Science analyses done in the collaborations

- Largest is cosmology group, the Dark Energy Science Collaboration, *DESC*

- DESC has some of the biggest workflow issues, since it needs to correlate the sky on large scales

- Current DESC activity focused on *Data Challenges* - simulations of images and catalogs.

Project/Collaboration Interface

- Collaborations must test DM code
- DM implements and absorbs algorithms from collaborations as needed
    - e.g. lensing science requires galaxy shapes - DM is ingesting algorithms to do this from DESC

LSST Pipelines

- Different parts of LSST use pipelines differently
- Project
    - Runs a single unified large pipeline
    - Runs on nightly data and for annual releases
    - Mainly image analysis, in parallel
- DESC
    - Run more varied science pipelines
    - Repeated more often, for different science analyses

3x2pt

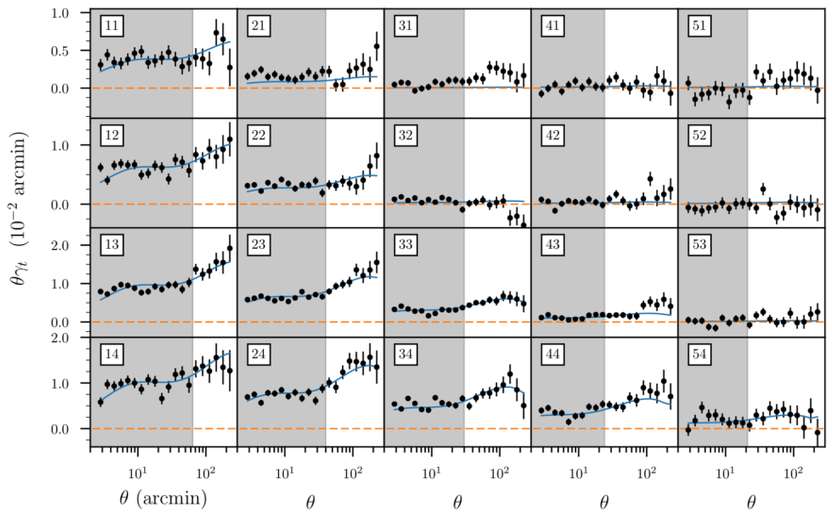
Case Study: 3x2pt Cosmology

- 3x2pt is a key science goal of LSST - most powerful dark energy constraints

- Combined lensing and galaxy clustering measurements

- Successfully measured by current DES and KiDS surveys

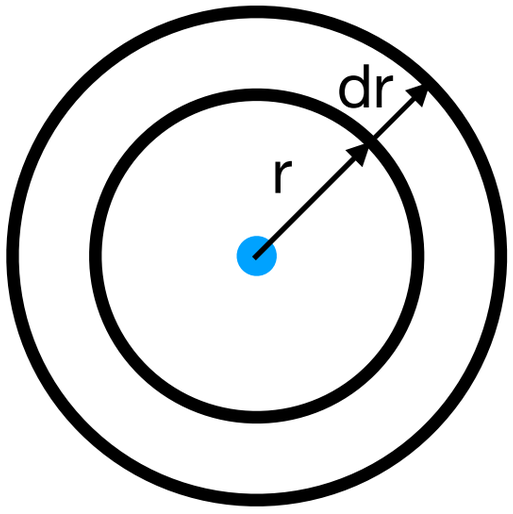
- A prototype case for our pipeline development



DES Y1 3x2pt Data

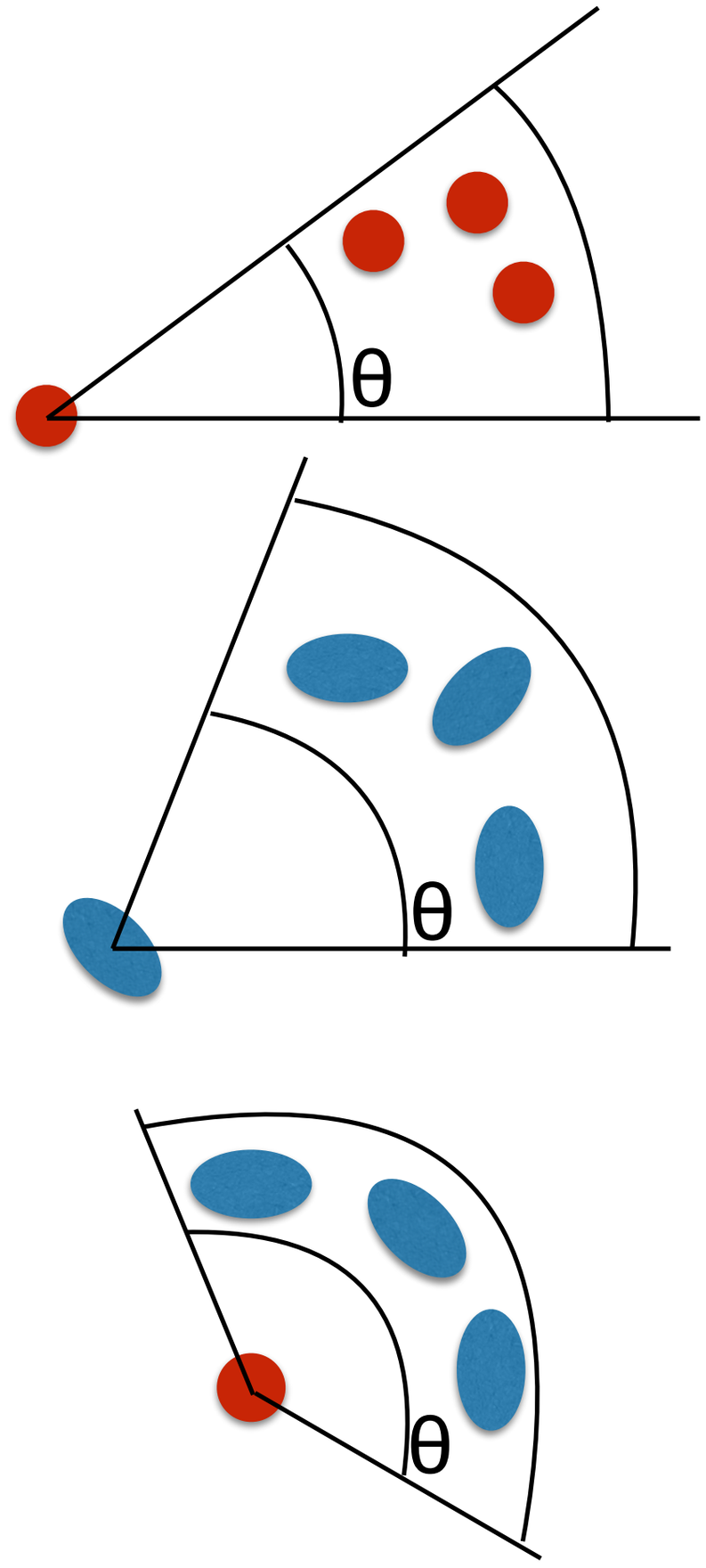
Two-Point Correlations

- Measures correlation in a field at different separations
    - $\langle f(\mathbf{x}) \cdot f(\mathbf{x}+\mathbf{r}) \rangle_{\mathbf{x}}$
    - Real space easy to understand
    - Fourier space *f* also very useful
    - In photometric surveys like LSST we mainly consider 2D fields
    - Remember that the sky is a sphere!



- In a purely Gaussian field, 2pt correlations describe all the available information

3 x 2pt Correlations



- w(θ)
  Galaxy density correlation function

- $\xi_+(\theta)$, $\xi_-(\theta)$
  Shear correlation functions

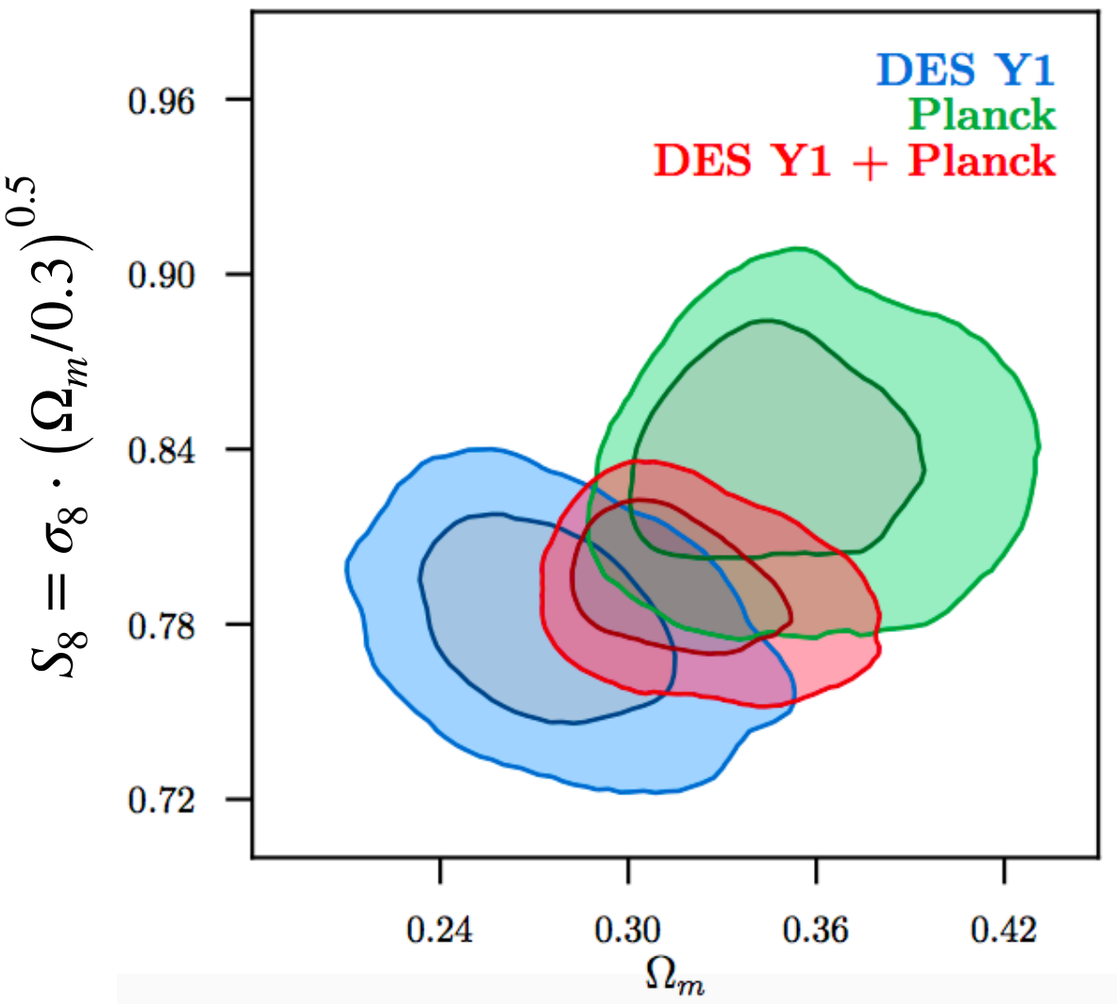
- $\boldsymbol{\gamma}_t(\theta)$
  Shear around lens galaxies

Systematics & Complementarity

- Shear - any galaxies
  - Sensitive to intrinsic alignments of galaxies & photometric redshift errors
- Density - luminous red galaxies
  - Amplitude has unknown galaxy bias
  - Smaller redshift errors
- Cross
  - Sensitive to both bias and IA

3x2pt Science

- 3x2pt measures:
  - cosmic structure amplitude
  - growth
  - redshift-distance relation
- Cosmological parameters $\Omega_m, \sigma_8, w(z)$

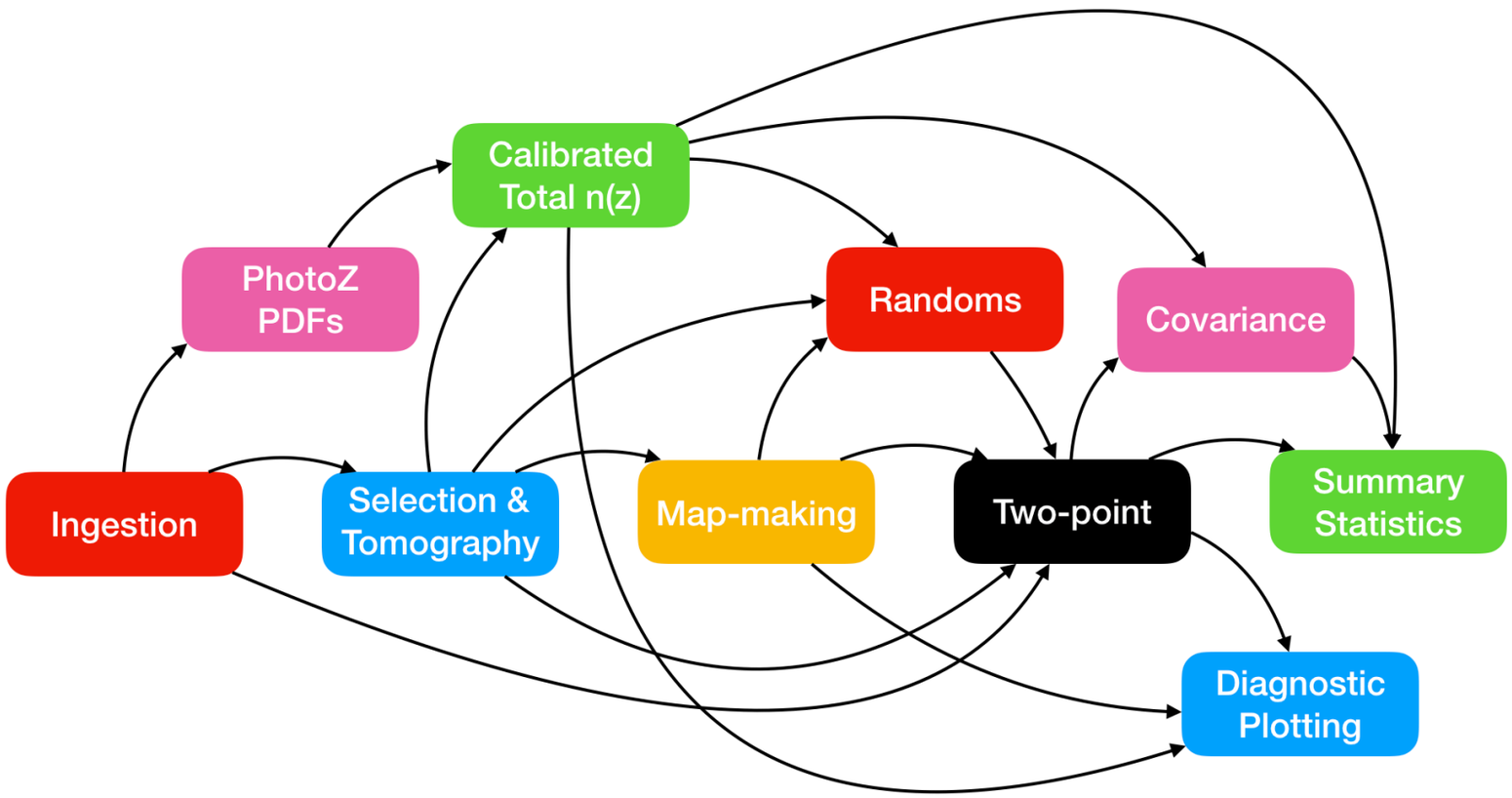


DES Y1 3x2pt Constraints

Pipelines

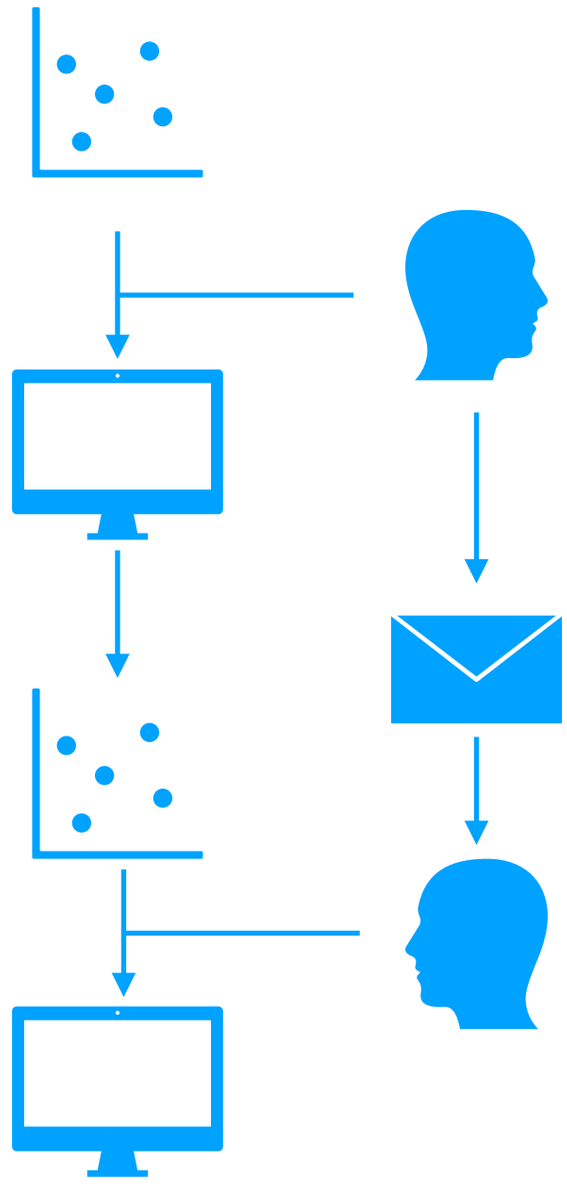
3x2pt Pipelines

- Many different analysis stages between catalog and cosmology
- Each stage is research problem in itself
- Most are also HPC problems
- Collecting & combining into a coherent pipeline is a key infrastructure challenge
    - Traditional approach is somewhat incoherent



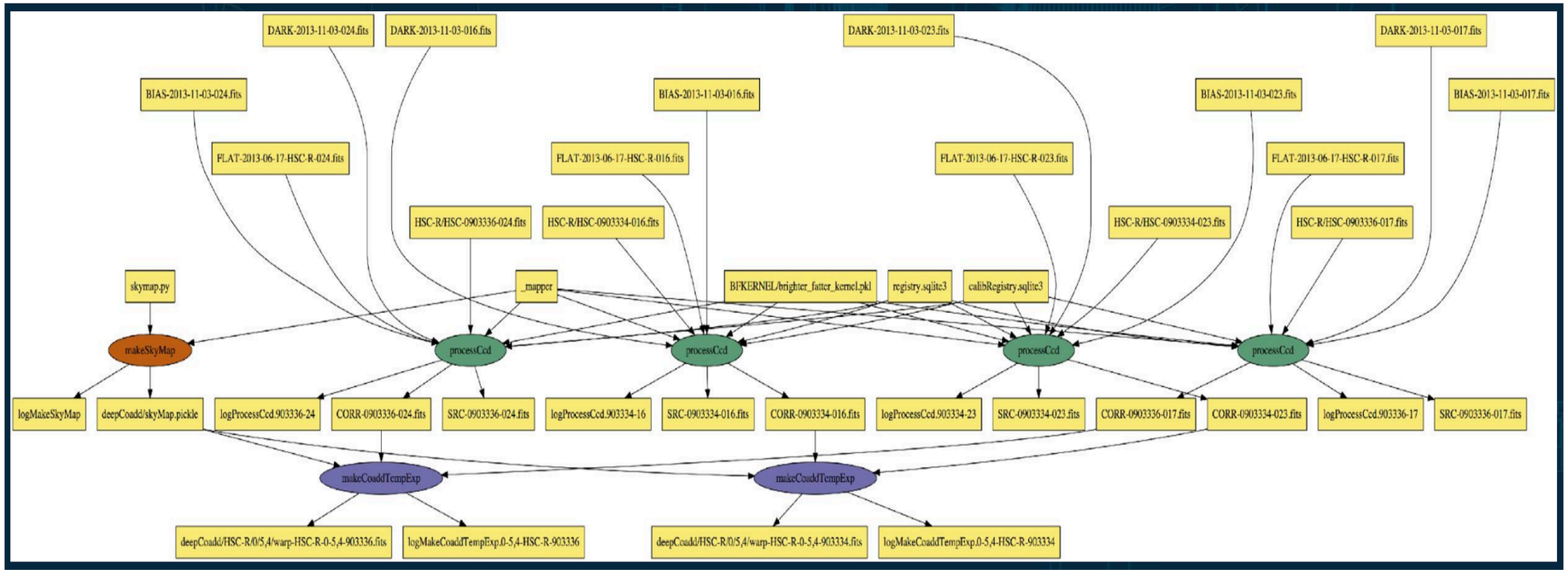
3x2pt Pipeline Goals

- Automation!
  - Traditional approach to workflow is humans + emails
- Collect together the processes that go from DM catalogs to MCMC samplers
- Run easily at DESC HPC Facilities
- Be testable at small scales on a laptop
- Use & provide streaming and parallel algorithms and tools
- Separate workflow from the scientific logic
- Easy debugging and development



Workflow Tools Overview

- Various Workflow Management Frameworks exist
- Manage launching jobs, dependency on previous jobs, data transfer,
- Especially important when multiple computational systems are used for different pieces of analysis, e.g. grid + cluster
- Though most of these are *not* designed for shared HPC!
  - Might be viable for cosmology on other types of system



Some Workflow Frameworks

CloudSlang

Parsl

Pegasus

RADICAL-Pilot

Pinball

Makeflow

PanDA

swift

Airflow

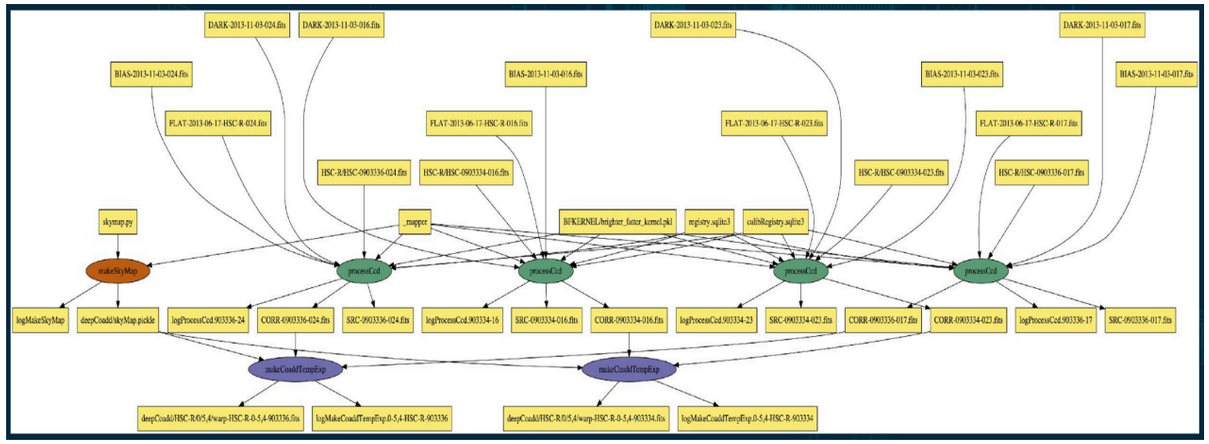
Project Workflow

- Project conducted a survey of workflow systems: https://dmtn-025.lsst.io/

- Planning on using Pegasus workflow
  - Experiments underway

- Building on existing task-running infrastructure

makeSkyMap
ProcessCcd
mosaic
makeCoaddTempExp
assembleCoadd
detectCoaddSources
mergeCoaddDetections
measureCoaddSources
mergeCoaddMeasurements
forcedPhotCoadd



https://project.lsst.org/meetings/lsst2017/sites/lsst.org.meetings.lsst2017/files/dm_stack_pegasus_chiang.pdf

Pegasus

- https://pegasus.isi.edu
- Experimented with this in DESC, + Project usage

- Powerful features!
  - Fault tolerance
  - Multi-site
  - Monitoring UI
  - Failure re-try
  - Remote file handling
  - Provenance tracking

Pegasus Description

- Text files describe components:

```
data1.txt  file:///home/joe/data1.txt    site="local"
data2.txt  http://example.org/data2.txt  site="example"
```

- Replica catalog (input files)

- Transformation catalog (executables)

```
tr task1 {
    site condorpool {
        pfn “/usr/bin/task1.exe"
        arch "x86_64"
        os "MACOSX"
        type "INSTALLED"
    }
}
```

- Site catalog (computers)

```
…
<site handle="local" arch="x86_64" os=“MACOSX">
...
```

Pegasus Job Description

- Python library for generating workflows:

```
dax = ADAG("pipeline")
input1 = File("input.dat")
intermediate1 = File("intermediate.dat")
output1 = File("output.dat")

job1 = Job("task1.exe")
job1.addArguments("-o", intermediate1, "-i", input1)
job2.uses(input1, link=Link.INPUT)
job2.uses(intermediate1, link=Link.OUTPUT)
dax.addJob(job2)

job2 = Job("task2.exe")
job2.addArguments("-o", output1, "-i", intermediate1)
job2curl.uses(intermediate1, link=Link.INPUT)
job2curl.uses(output1, link=Link.OUTPUT)
dax.addJob(job2)
```

Pegasus

- Verbose / heavy duty!
- Not suited to testing on personal machines - requires an HTCondor installation
    - This was a major killer for me - dramatically reduces ability to test at smaller scale

Parsl

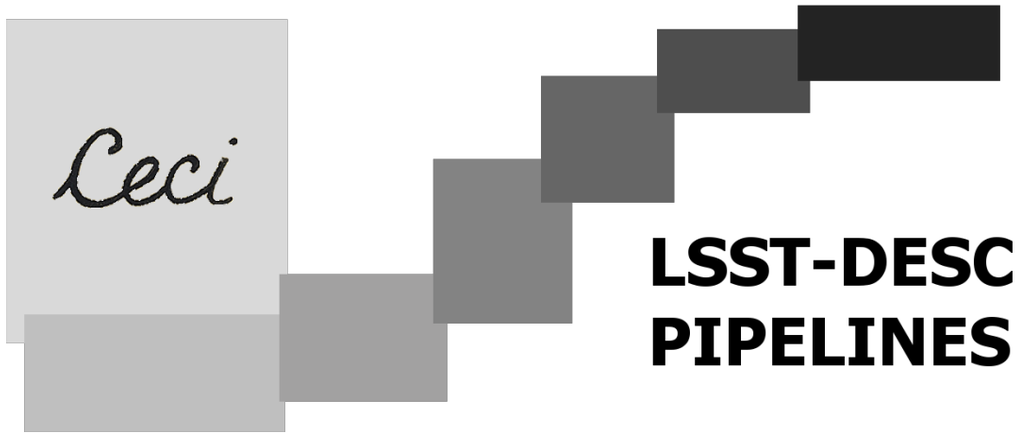
- parsl.readthedocs.io
- External project
    - Authors now DESC members, working closely
- Models pipeline stages as python functions or bash strings and examines their inputs and outputs
- Execution library knows about submission to various job managers and systems
- Dramatically less boilerplate than Pegasus

Parsl Example

```python
from parsl.app.app import bash_app
from parsl.data_provider.files import File

# Existing file
input1 = File('input.txt')

# Created files
intermediate_name = 'intermediate.txt'
output_name = 'output.txt'

# define tasks
@bash_app
def task1(inputs=[], outputs=[]):
    return 'task1.exe -i {} -o {}'.format(inputs[0], outputs[0])

@bash_app
def task2(inputs=[], outputs=[]):
    return 'task2.exe -i {} -o {}'.format(inputs[0], outputs[0])

# Run first task
results1 = task1(inputs=[input1], outputs=[intermediate_name])

# Second task, depending on output of first. Will not run until ready
intermediate = results1.outputs[0]
results2 = task2(inputs=[intermediate], outputs=[output_name])

# Wait for final task to finish
results2.result()
print("Complete")
```

Ceci

- Wanted to impose more structure on pipelines than Parsl's function structure allowed
    - Wanted pipelines to clearly express their inputs, outputs, and config in a list and then use these to connect stages
    - Also easier to run individual stages and to debug them
- Wrote a wrapper around Parsl, *ceci*
- Class to automate connecting a job to a wider pipeline - hide Parsl from users
- Targeted at relatively simple pipelines with clear connections



- This was targeted specifically at DESC pipeline problems, not a more general solution

Ceci Pipeline Stage

```
class TXPhotozStack(PipelineStage):
    """
    Naively stack photo-z PDFs in bins according to previous selections.

    """
    name='TXPhotozStack'
    inputs = [
        ('photoz_pdfs', PhotozPDFFile),
        ('tomography_catalog', TomographyCatalog)
    ]
    outputs = [
        ('photoz_stack', NOfZFile),

    ]
    config_options = {
        'chunk_rows': 5000,  # number of rows to read at once

    }


    def run(self):
        """
        Run the analysis for this stage.

         - Get metadata and allocate space for output
         - Set up iterators to loop through tomography and PDF input files
         - Accumulate the PDFs for each object in each bin
         - Divide by the counts to get the stacked PDF
        """
```

Common Workflow Language

- A standard is developing for workflow languages, CWL

- Could be useful for future workflows

- Verbose but flexible, understands containers

- Not the time to adopt wholeheartedly, but definitely worth keeping an eye on

COMMON
WORKFLOW
LANGUAGE

Recommendations

- Understand distinction between different kinds of pipeline - be clear on target use case, consider different engines for different pieces

- There is clear value to workflow management, especially for distributed collaborations

- Don’t require your users to write pegasus/parsl/other directly - have a thin interface layer for your system

- Maintain workflow-agnostic component design