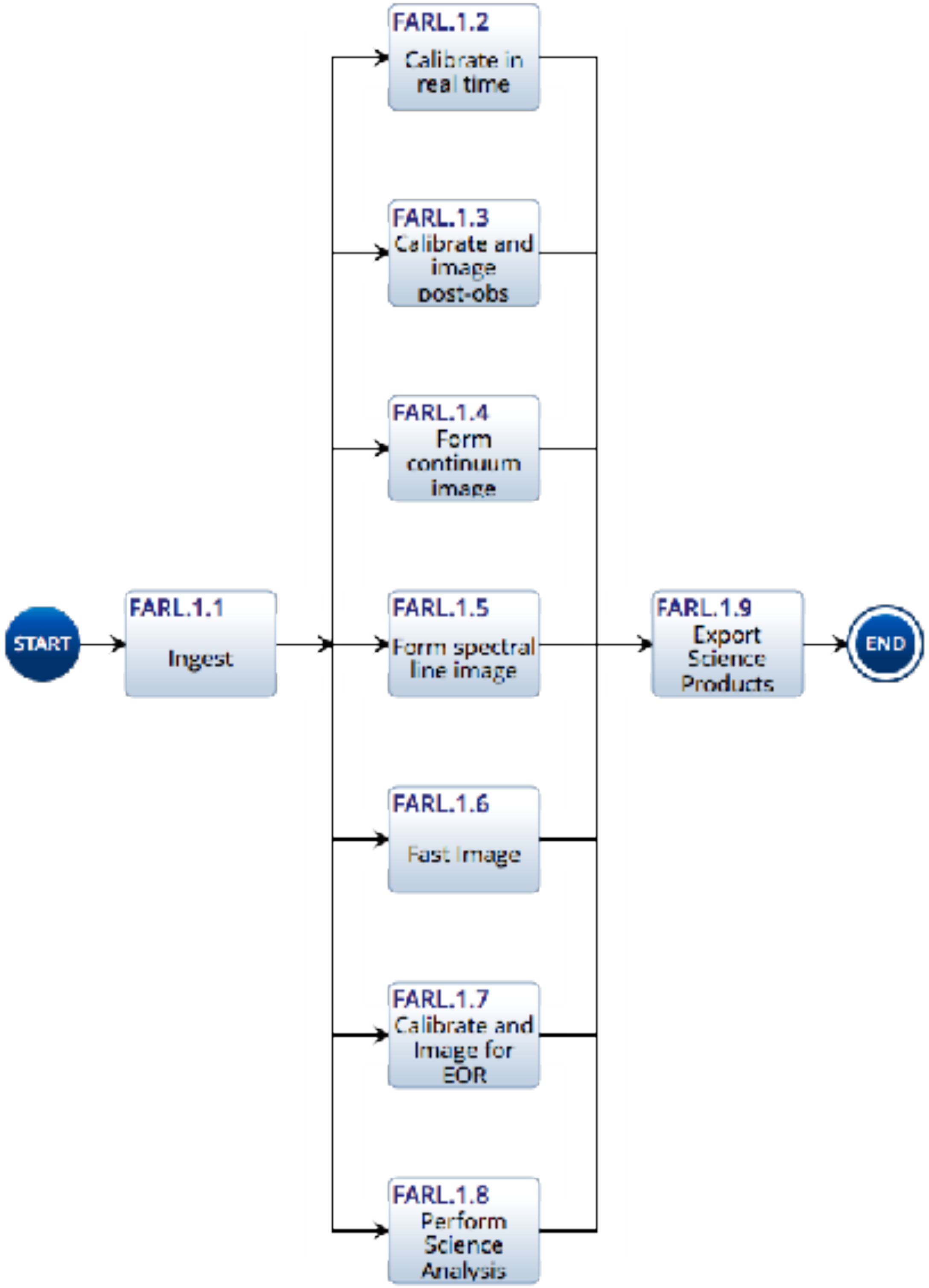
The Labyrinths of SKA Processing

Tim Cornwell
SKA Science Data Processing consortium

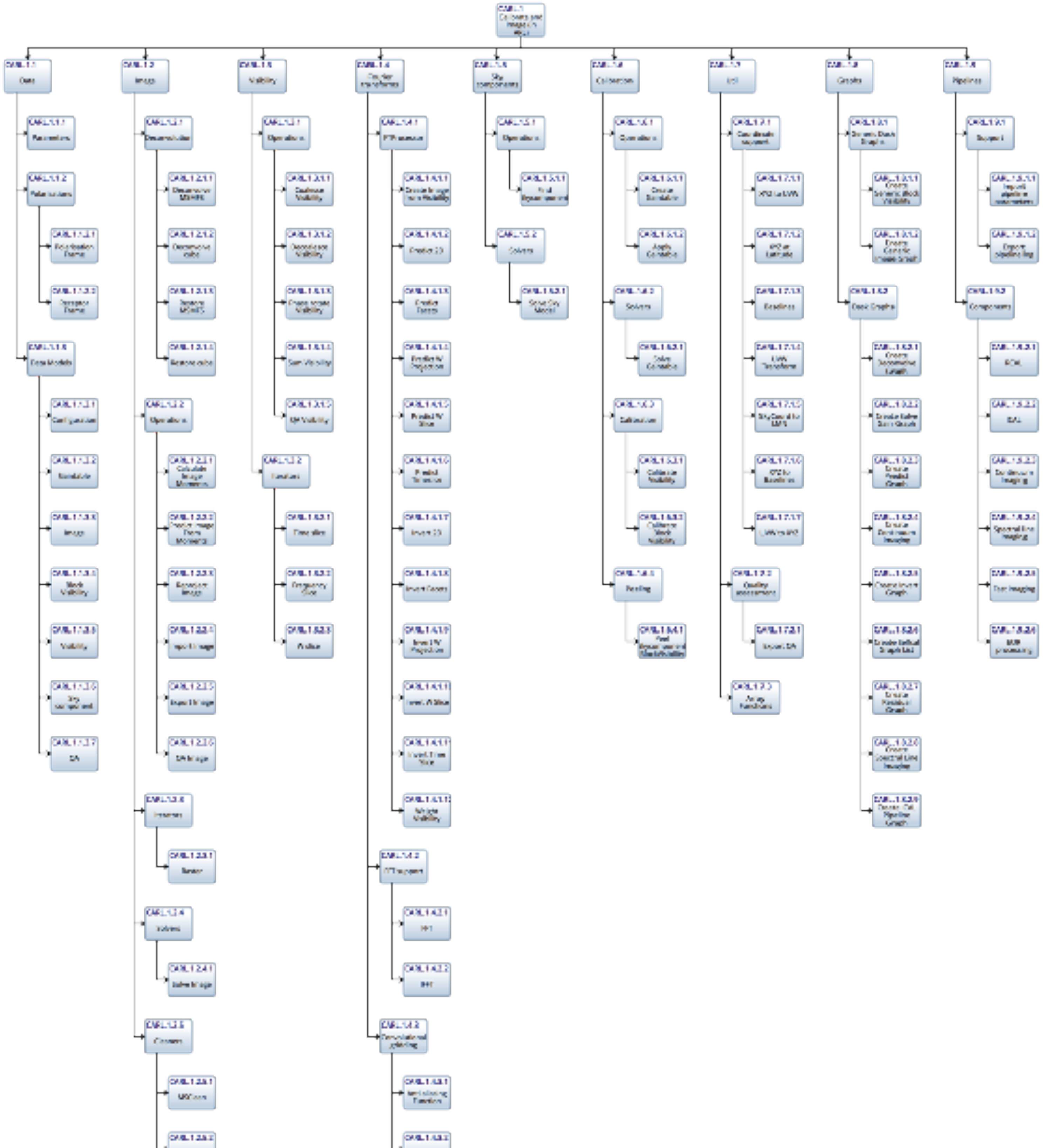
Tim Cornwell Consulting
Interferometry and Imaging

30,000 ft view of SKA pipeline processing

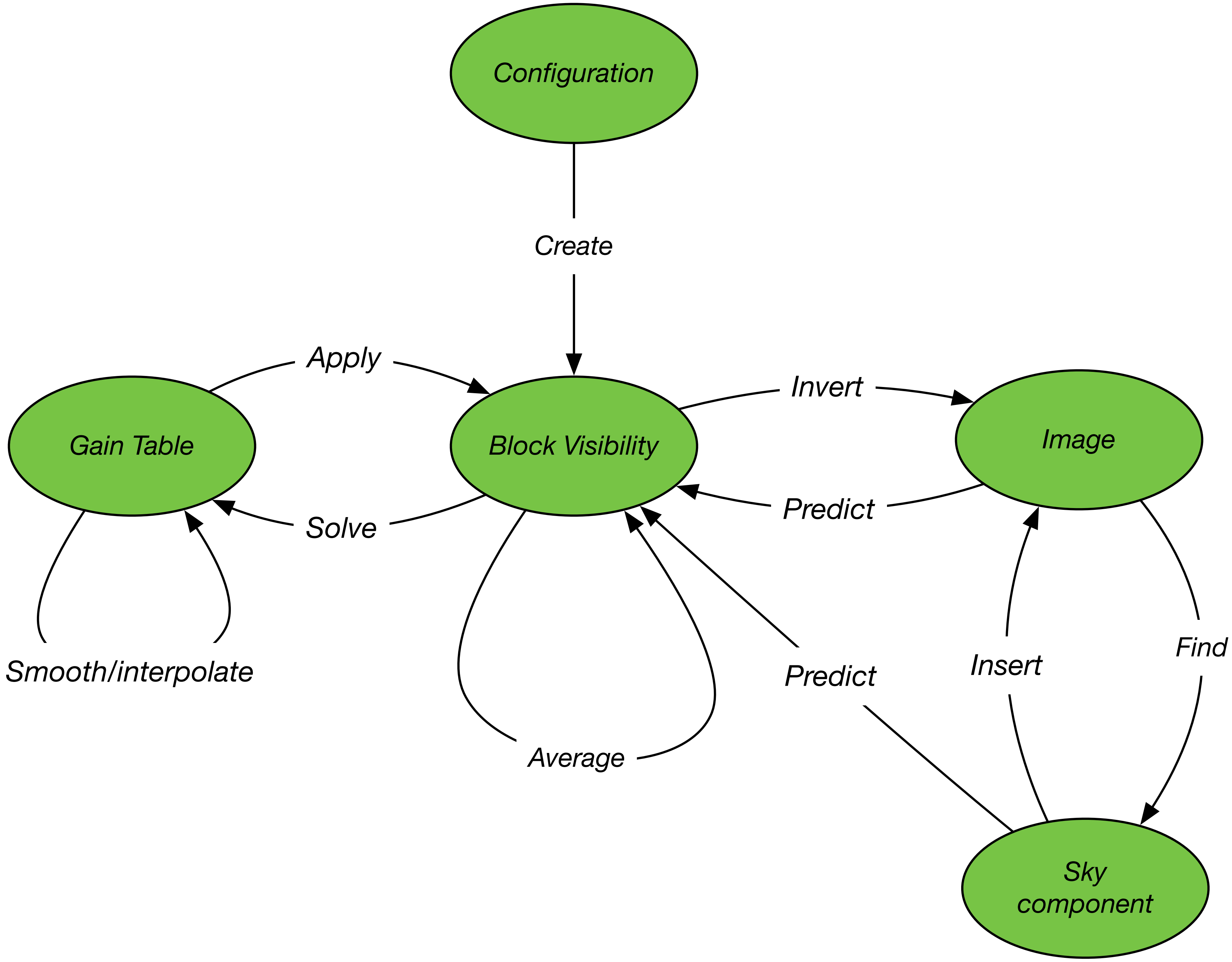


Algorithm Reference Library

- Algorithm Reference Library
- All major Calibration and Imaging algorithms
- Data models = 6 classes
- Components = O(240) state-free functions
- ~ 6000K LOC

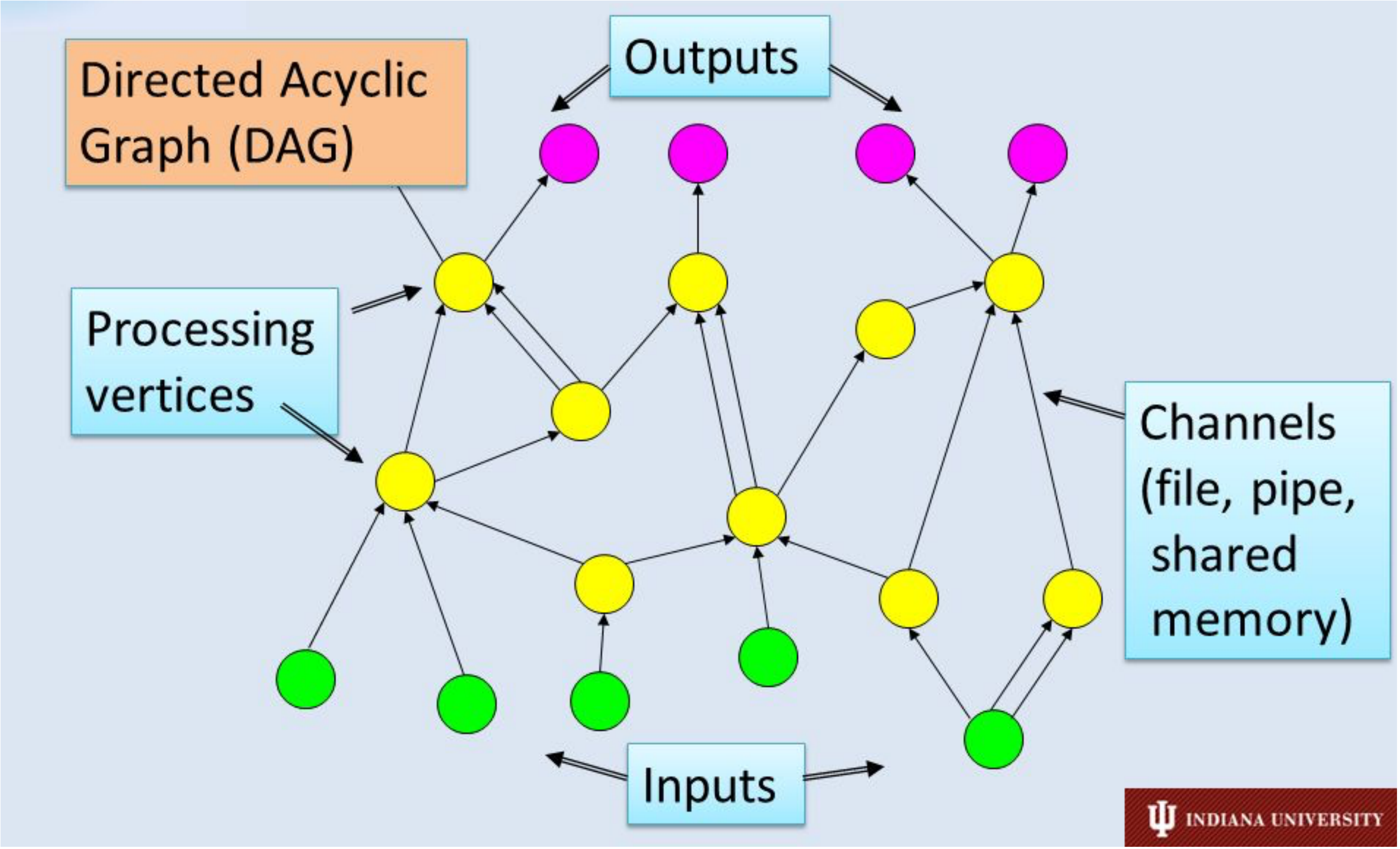


The things we do to data



Why is SKA processing hard?

- We know how to do most SKA calibration and imaging
- Why not do SKA calibration and imaging single threaded?
- Because a single project would take hundreds of days
- Alternative is to distribute processing over thousands of nodes
- Incur a large complexity problem
- But we are not alone in that…



Directed Acyclic Graph (DAG)

Outputs

Processing vertices

Channels (file, pipe, shared memory)

Inputs

Linear algebra

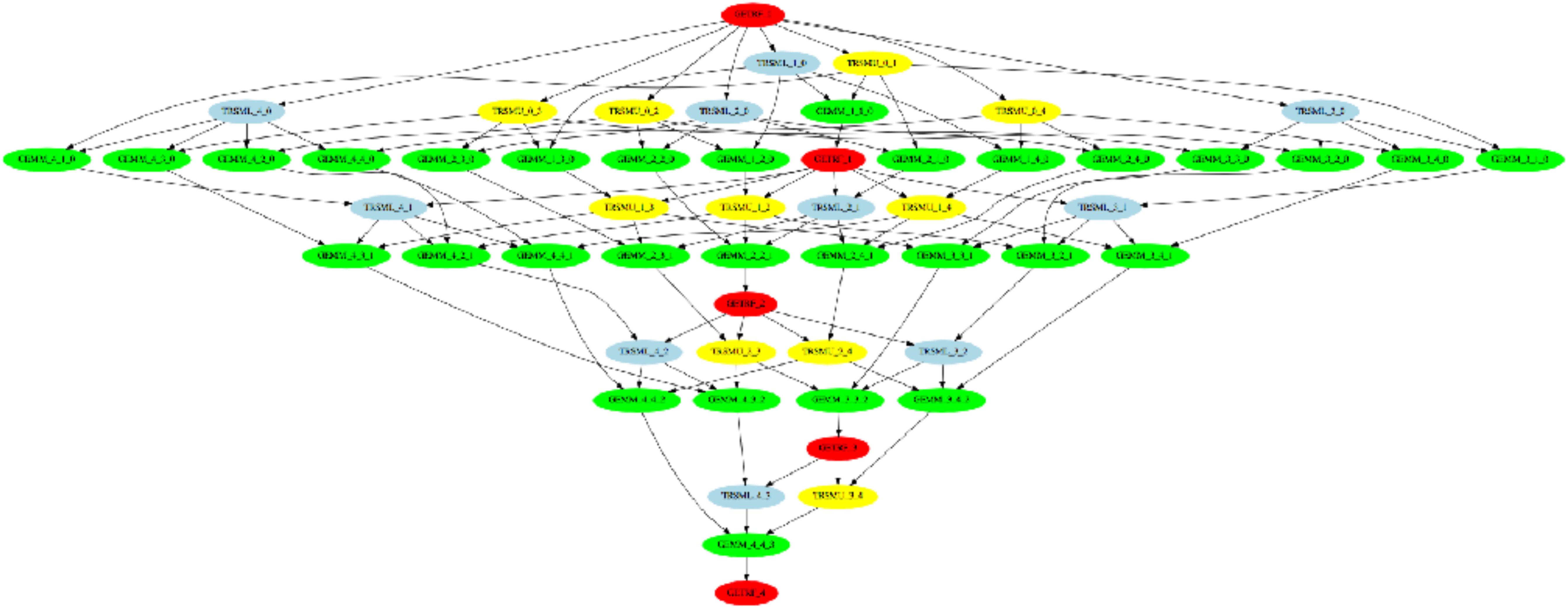


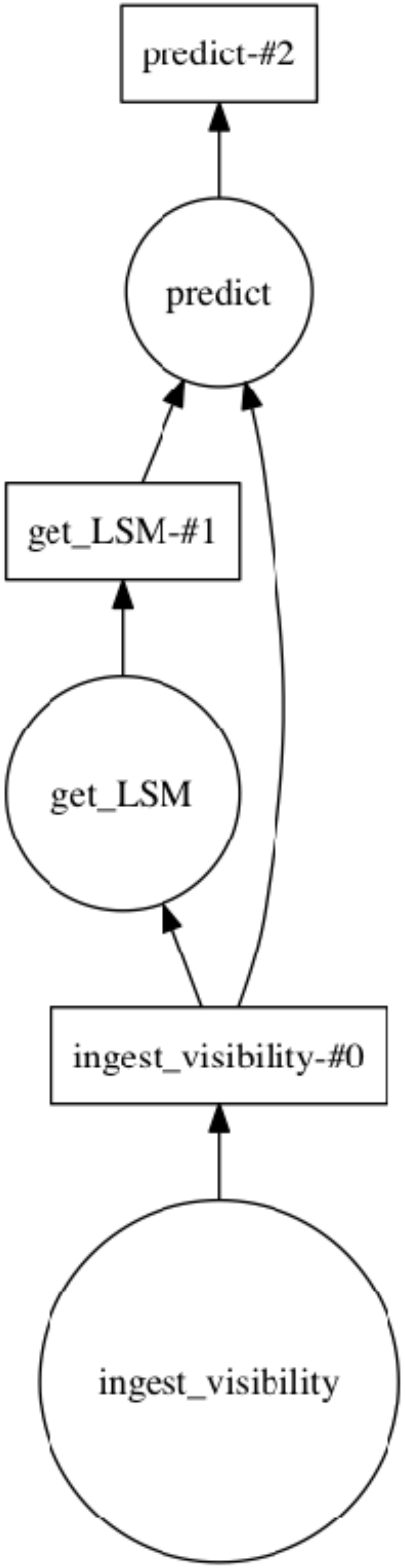
Figure 2: DAG of a LU factorization on a 5 × 5 tiled matrix

ARL and Directed Acyclic Graphs

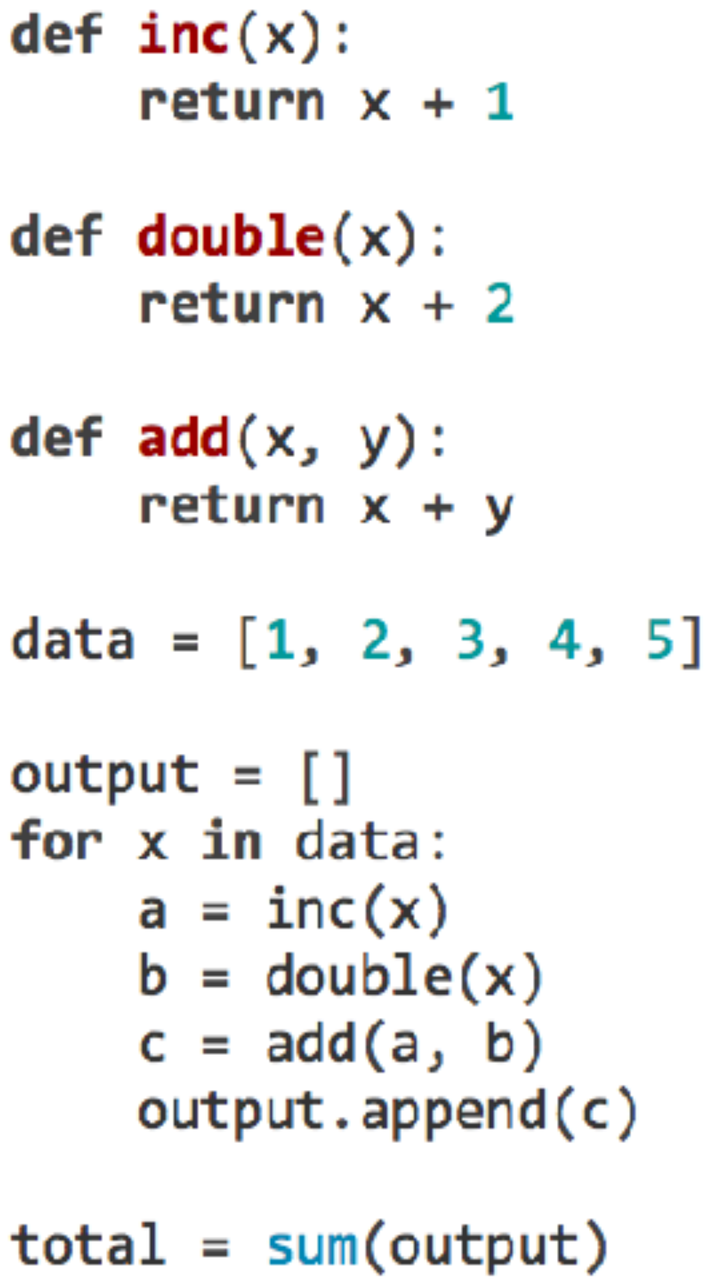
- DAGs are major part of SKA processing plans
- Dask is a python package for distributed processing, including DAGs
- Idioms supported: arrays, frames, bags, delayed
- Use “delayed” function to construct DAGs
- SKA will select substantial DAG packages e.g. Apache Spark
- Dask good to build quasi-realistic graphs

Graph of predict for 1 visibility set,

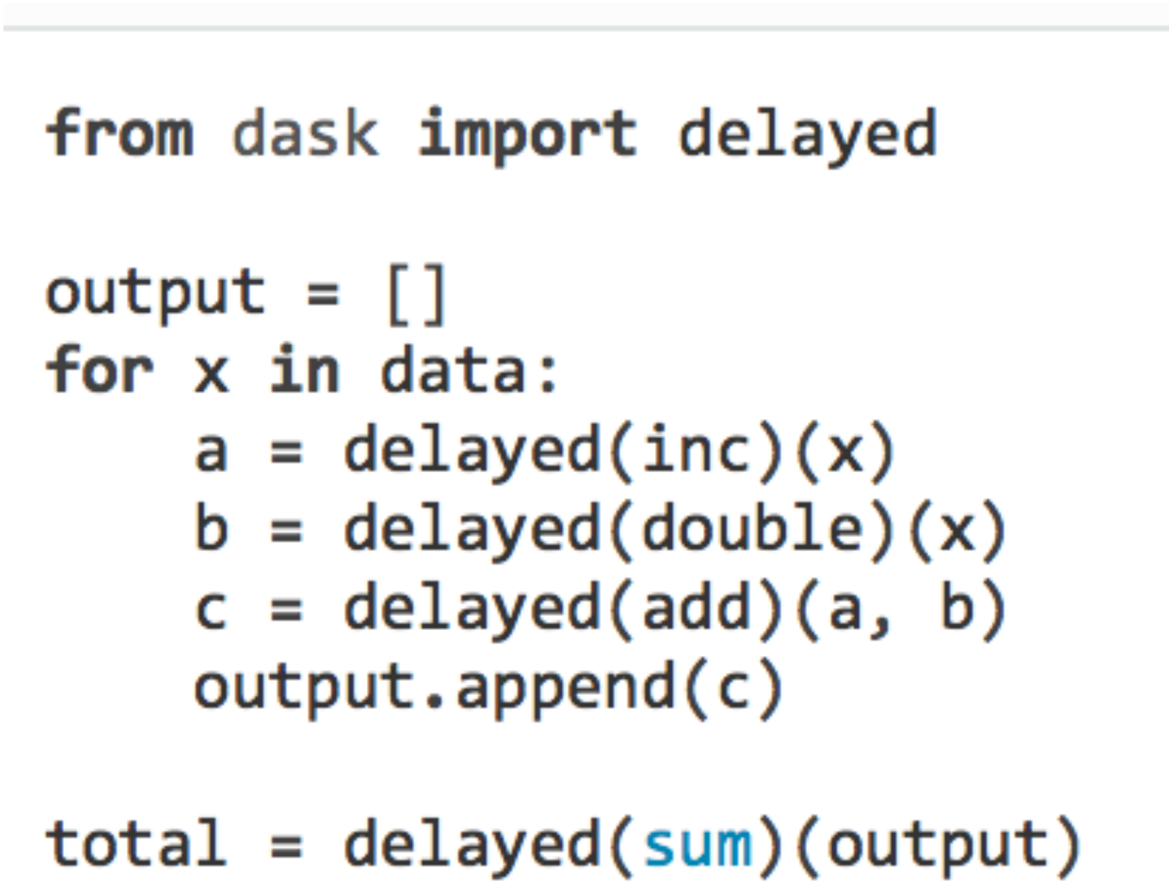
- Flows from bottom to top
- Boxs are data
- Circles are functions
- Ingest visibility
- Get the Local Sky Model for this visibility
- Predict the visibility
- Directed Acyclic Graph



Dask.delayed

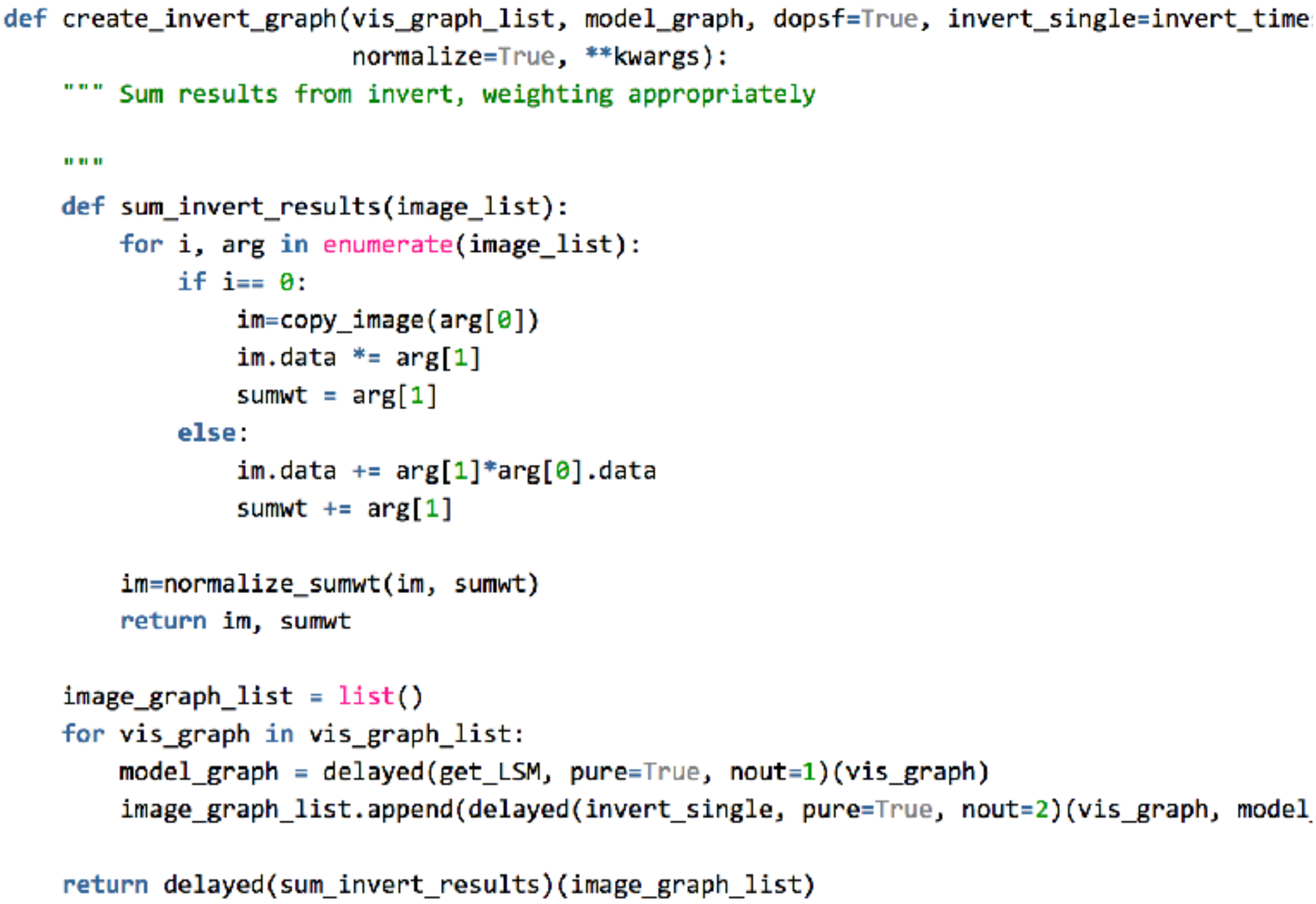


```python
def inc(x):
    return x + 1

def double(x):
    return x + 2

def add(x, y):
    return x + y

data = [1, 2, 3, 4, 5]

output = []
for x in data:
    a = inc(x)
    b = double(x)
    c = add(a, b)
    output.append(c)

total = sum(output)
```



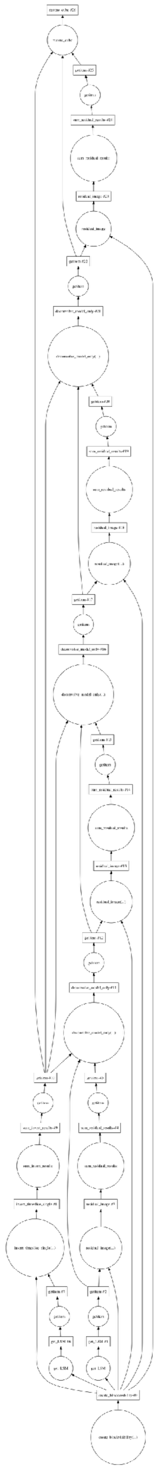
```python
from dask import delayed

output = []
for x in data:
    a = delayed(inc)(x)
    b = delayed(double)(x)
    c = delayed(add)(a, b)
    output.append(c)

total = delayed(sum)(output)
```

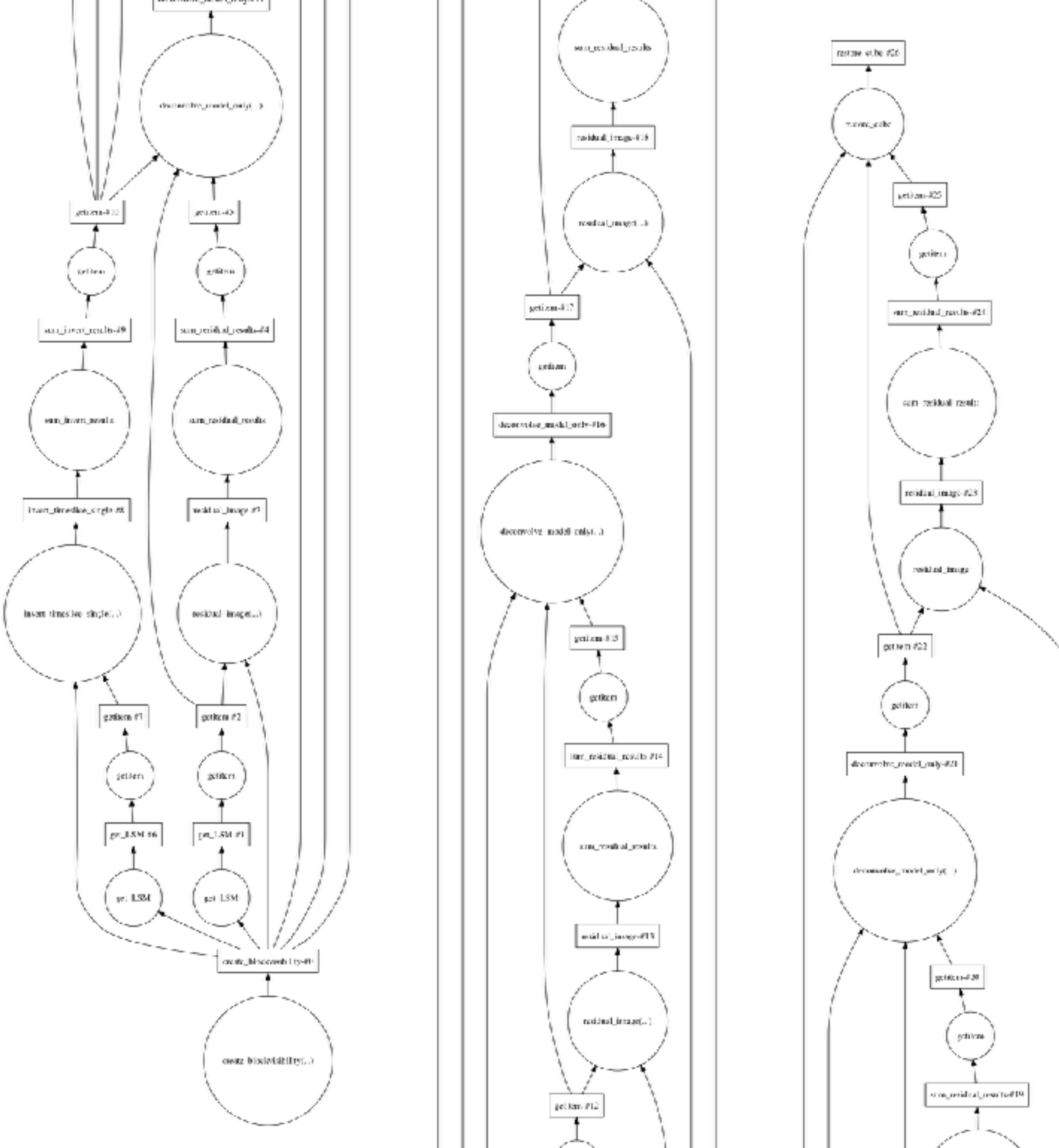
Wrapping Invert into graph



```
def create_invert_graph(vis_graph_list, model_graph, dopsf=True, invert_single=invert_time
                        normalize=True, **kwargs):
    """ Sum results from invert, weighting appropriately

    """
    def sum_invert_results(image_list):
        for i, arg in enumerate(image_list):
            if i== 0:
                im=copy_image(arg[0])
                im.data *= arg[1]
                sumwt = arg[1]
            else:
                im.data += arg[1]*arg[0].data
                sumwt += arg[1]

        im=normalize_sumwt(im, sumwt)
        return im, sumwt

    image_graph_list = list()
    for vis_graph in vis_graph_list:
        model_graph = delayed(get_LSM, pure=True, nout=1)(vis_graph)
        image_graph_list.append(delayed(invert_single, pure=True, nout=2)(vis_graph, model_

    return delayed(sum_invert_results)(image_graph_list)
```

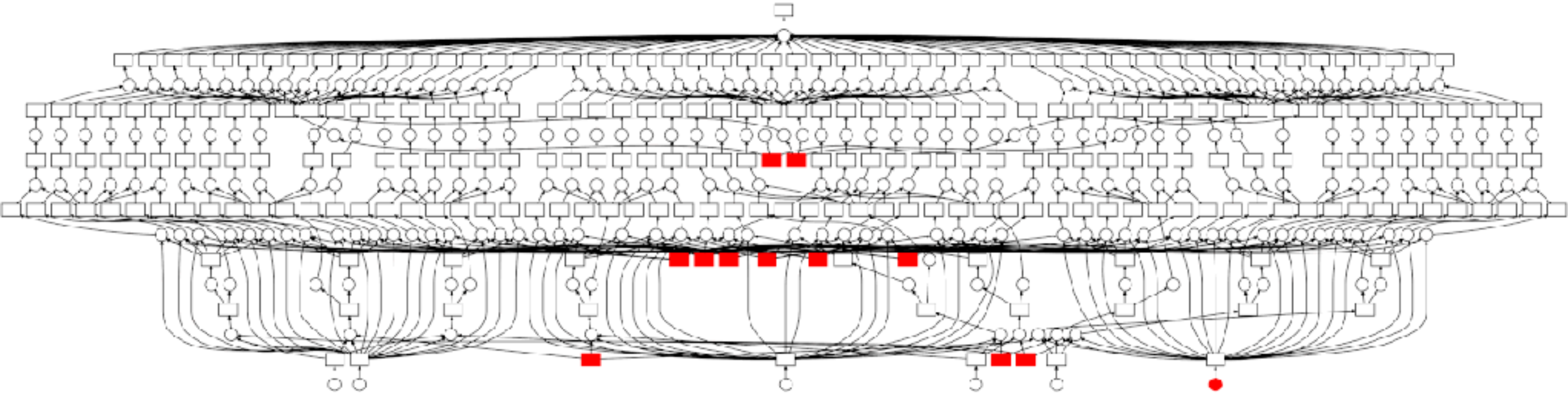
1 ingest, 3 major cycles, continuum pipeline



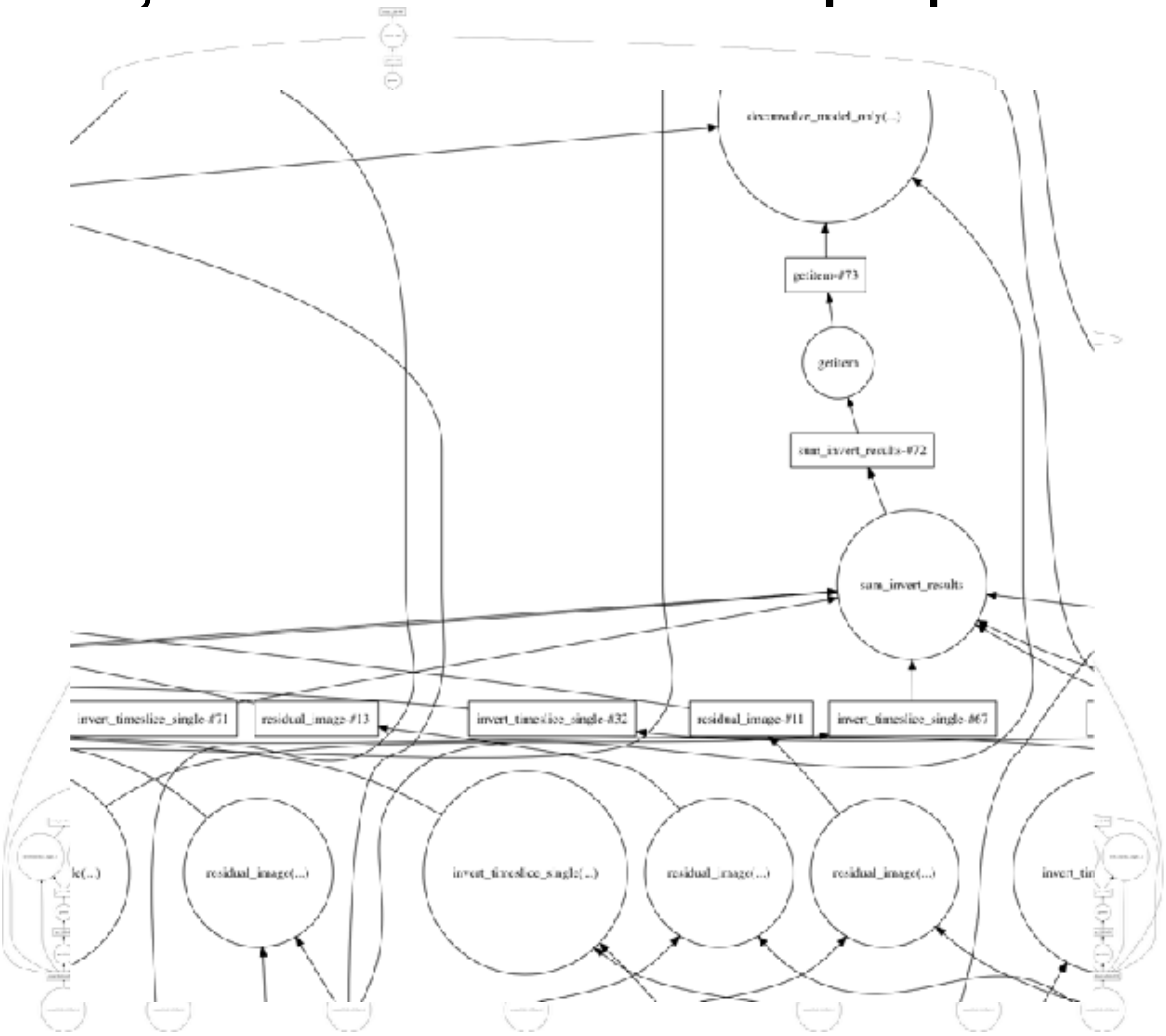


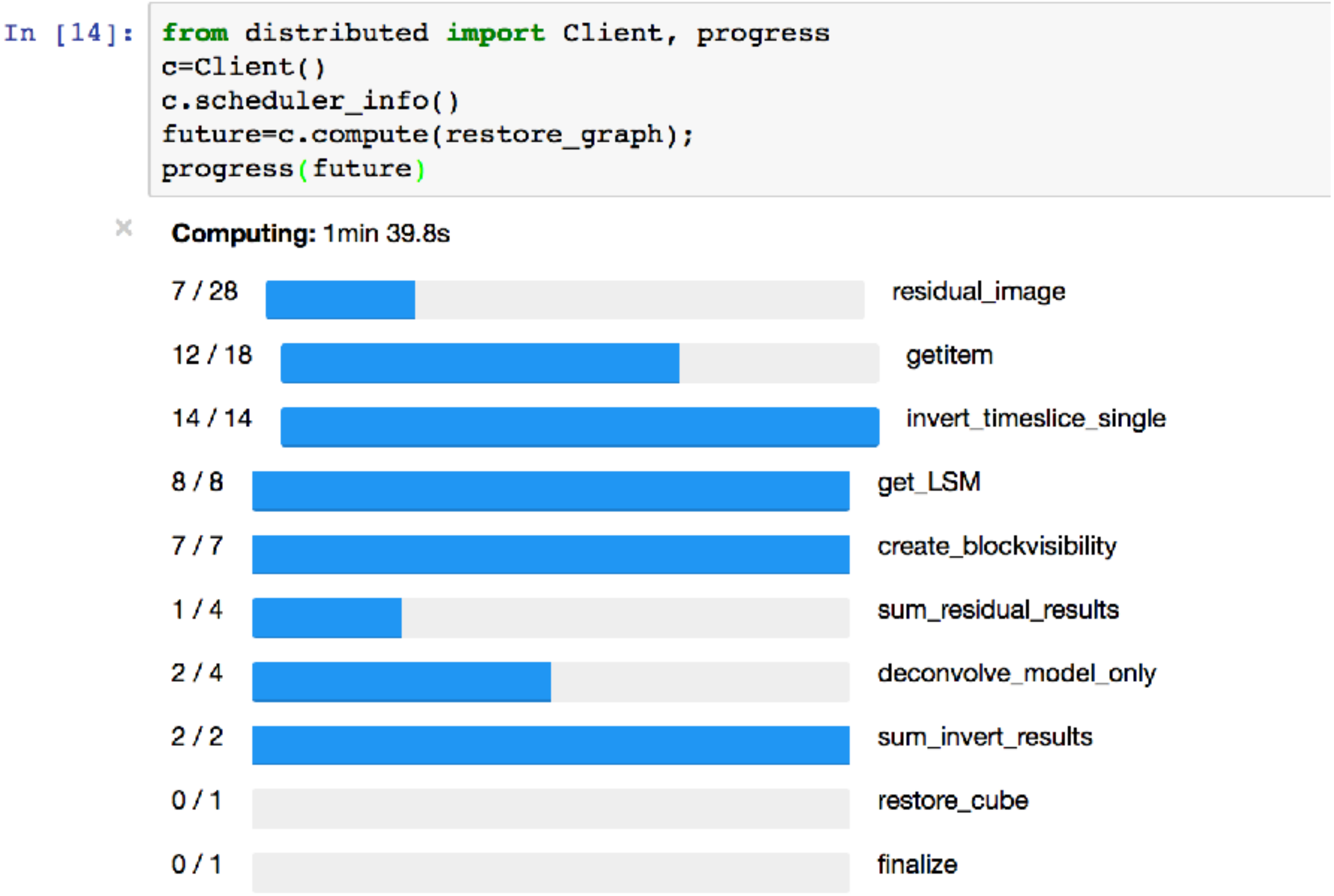
Processing graph

- Not SDP (yet)!
- From tutorial: https://github.com/dask/dask-tutorial



7 way ingest, 3 major cycles, continuum pipeline

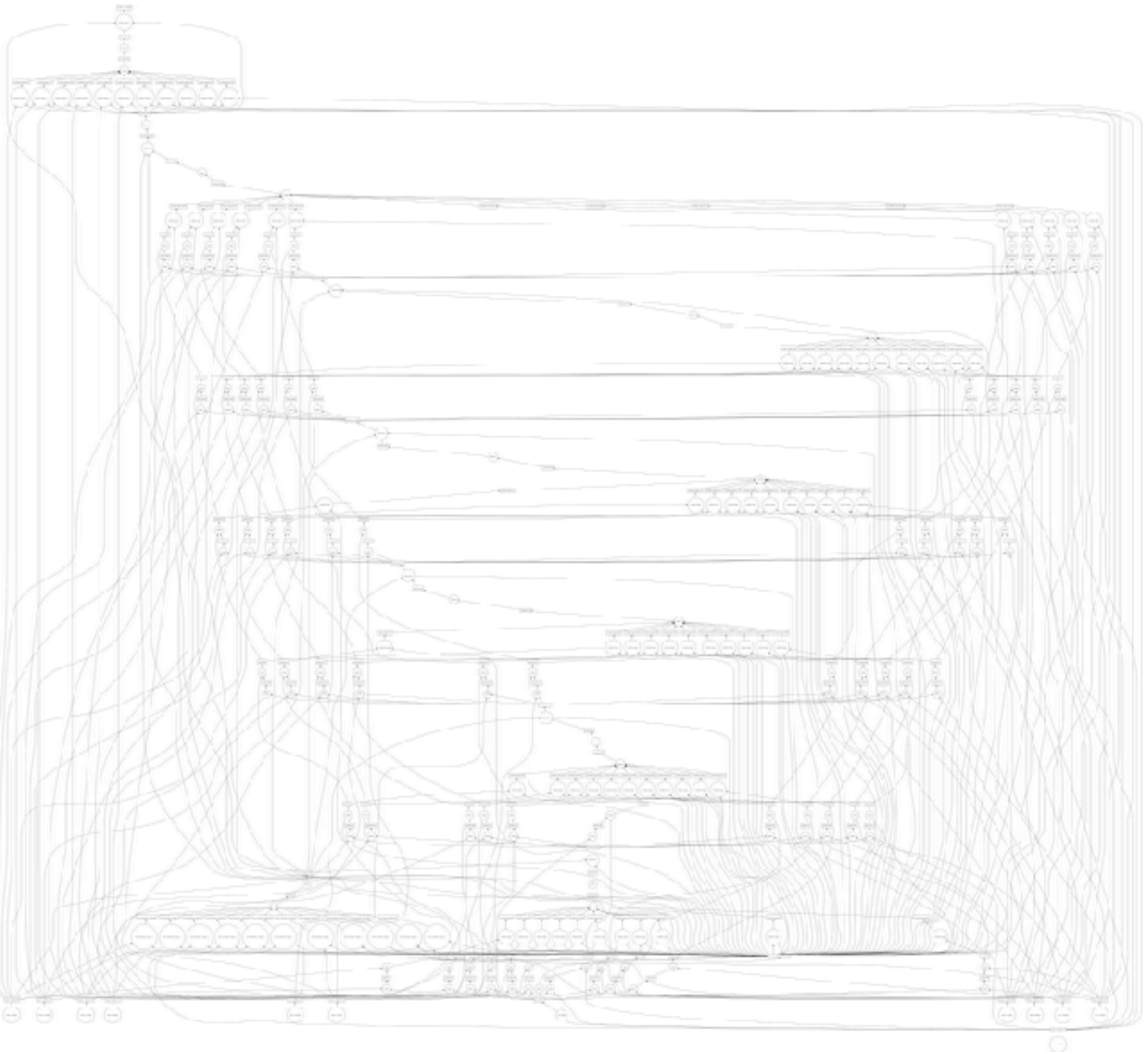




In [14]:
```python
from distributed import Client, progress
c=Client()
c.scheduler_info()
future=c.compute(restore_graph);
progress(future)
```

```
Computing: 1min 39.8s

7 / 28     residual_image
12 / 18    getitem
14 / 14    invert_timeslice_single
8 / 8      get_LSM
7 / 7      create_blockvisibility
1 / 4      sum_residual_results
2 / 4      deconvolve_model_only
2 / 2      sum_invert_results
0 / 1      restore_cube
0 / 1      finalize
```

11 inputs, 5 cycles



Summary

- DAGs help describe and deploy SKA processing
- Python + jupyter + dask + laptop/desktop = fast development
- Python + dask + cluster = way to learn about real graph processing at scale
- We expect to push upwards in graph complexity
- Currently at 5,000 - 10,000 vertices
- Understand performance, memory use, network bandwidth, schedulers