FINNISH METEOROLOGICAL INSTITUTE

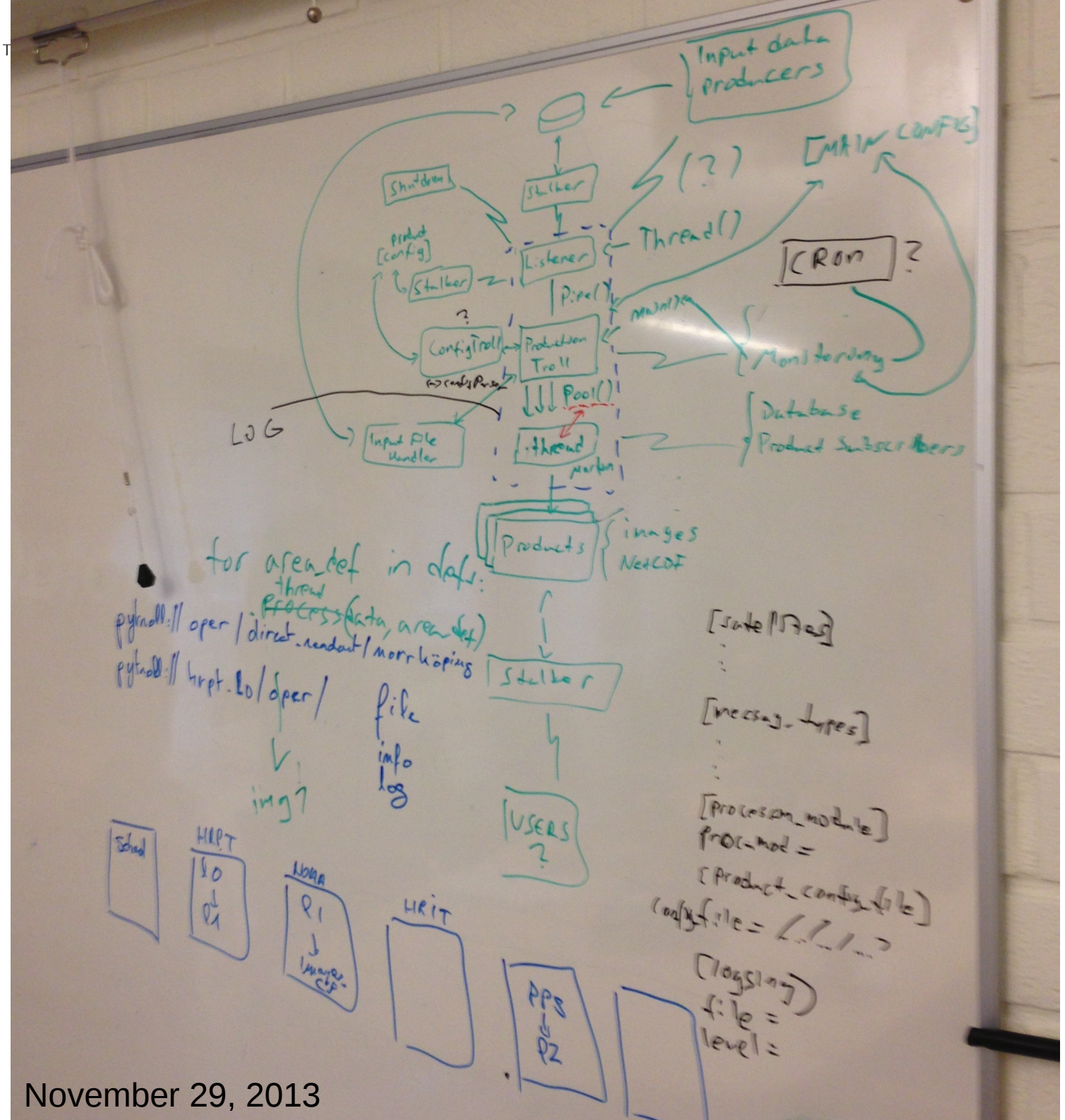
Operational satellite image generation with PyTroll

Panu Lahtinen

Finnish Meteorological Institute
panu.lahtinen@fmi.fi

Outline

- Trollduction
- Trollflow + Trollflow-sat
  - Plug-in Structure
    - Daemons
    - Workflow items
    - Trollflow internals
  - SatPy example



November 29, 2013

Trollduction

- Built on top of mpop
- Monolithic
- “Everything for everyone”
- Processor config in .ini
- Product config in XML

Trollflow

- Small workflow execution framework
- Runs plug-ins that communicate via queues
  - Daemons and workflow items
  - Each plug-in in own Thread which can be locked
    - No true parallelism
  - Small units that *should* do **one** thing
  - Only consecutive plug-ins need to know what is being passed
  - Crashed plug-ins are restarted automatically
  - Configuration in YAML

Trollflow

- Not specific to satellite data processing
- https://github.com/pytroll/trollflow
  - First draft by Rolf Helge Pfeiffer
  - Extended, production-readied and not-fully-understood by Panu Lahtinen
  - Used in operations, but code needs some clean-up
  - `trollflow/bin/flow_processor.py`

Trollflow-sat

- Trollflow plug-ins specific for satellite data
  - Messaging
  - Data ingestion
  - Area coverage calculations
  - Resampling
  - Image composite creation
  - Data writing
- Mpop and SatPy-based chains
- Templates for daemons and workflow items

Trollflow-sat

- https://github.com/pytroll/trollflow-sat
  - Fixes and additions by Martin Raspaud and Christian Kliche
- Small rewrite needed
  - Handle “extra” metadata properly
  - Changes? Xarray and Dask being introduced to SatPy
  - Handle resampling cache properly with SatPy
  - Move common features from flow_processor.py
- Documentation?
  - Example config files

Plug-in Structure

- The workers
  - Daemons
    - Active all the time
    - Trigger processing
    - Save data
    - Typically has only input *or* output queue
  - Workflow items
    - Actual processing units
    - Active only when `invoke()`'d
    - Connected to workers on both sides
      - Input queue
      - Output queue

Plug-in Structure

- The middle-man – WorkflowRunner
  - Initializes the **workflow** items
  - Calls the workflow items `invoke(context)`
  - Needs simplification, or merging to WorkflowStreamer

Plug-in Structure

- The out-most internal layer - WorkflowStreamer

```
class WorkflowStreamer(Thread):
    def __init__(...):
        self.input_queue = None
        self.output_queue = Queue()
        self.workflow = config or self.read_workflow(...)
        self.runner = WorkflowRunner(workflow)

    def run(self):
        while self._loop:
            data = self.input_queue.get()
            context = {'output_queue': self.output_queue}
            context['content'] = data
            self.runner.run(context)
```

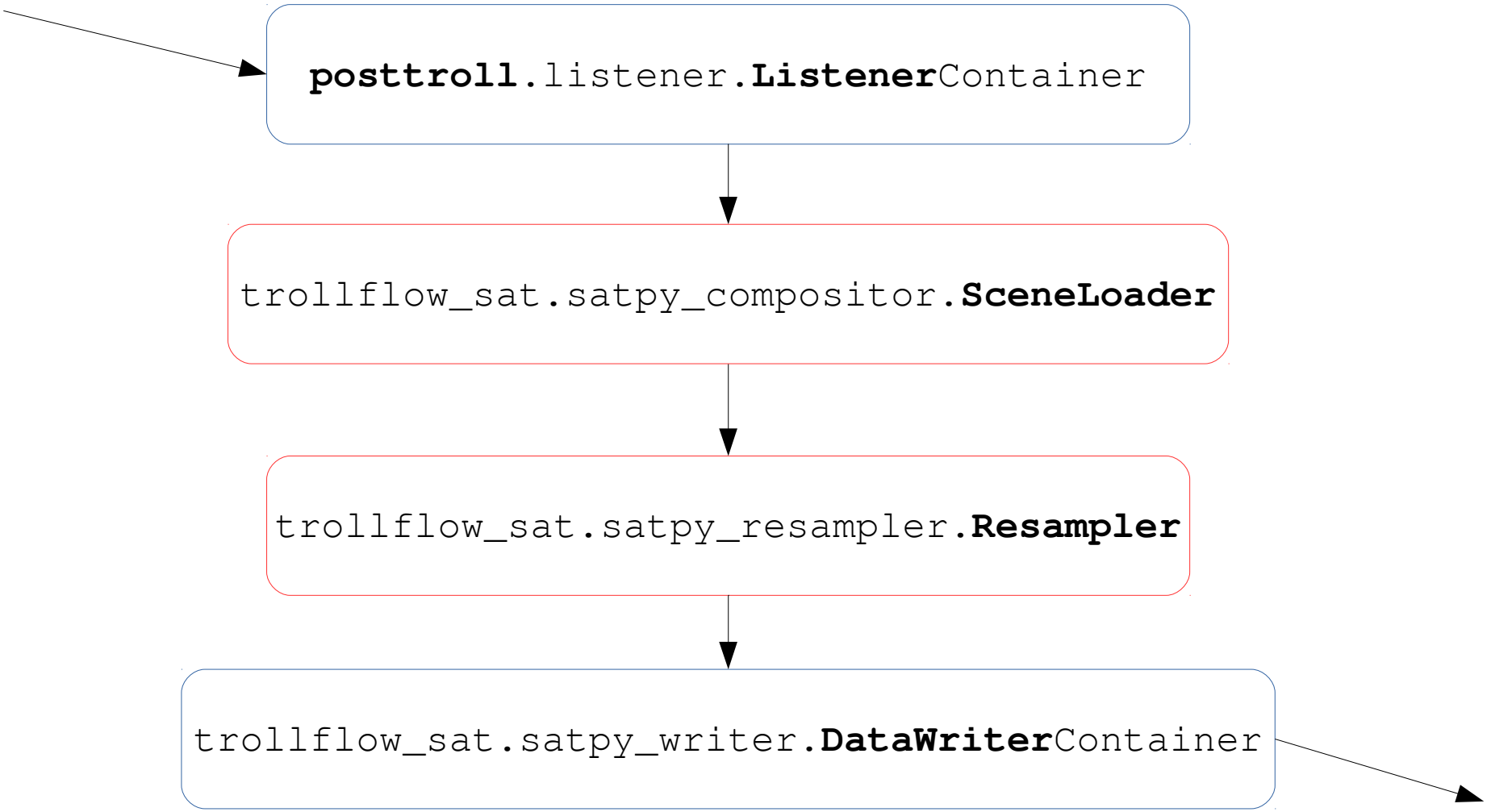
Plug-in Structure

- The `main()` - flow_processor.py

```
TYPES = {'daemon': generate_daemon,
         'workflow': generate_thread_workflow,
         }

def generate_daemon(config_item):
    return config_item['components'][-1]['class']

def generate_thread_workflow(config_item):
    wfs = WorkflowStreamer(config=config_item)
    wfs.setDaemon(True)
    wfs.start()
    return wfs

def create_threaded_workers(config):
    # Create workers
    workers = []
    for cfg in config['work']:
        workers.append(TYPES[cfg['type']](cfg))

    # Connect queues
    queue = None
    for worker in workers:
        if queue is not None:
            worker.input_queue = queue
        queue = worker.output_queue
```

Example with SatPy

- Example workflow



Example with SatPy

- Worklow configuration file (YAML)

```
work:
  - type: daemon
    name: Listener
    components:
      - class: !!python/object:posttroll.listener.\
                                          ListenerContainer
        topics: [“/eggs/bacon/spam”, “/spam/spam/spam/spam/baked_beans/spam”]

  - type: workflow
    name: compositor
    Workflow:
      - trollflow_sat.satpy_compositor.SceneLoader
        product_list: &product_list
          “/path/to/config/product_config.yaml”

  - type: workflow
    name: resampler
    Workflow:
      - trollflow_sat.satpy_resampler.Resampler
        product_list: *product_list
        proj_method: bilinear  # + several other optional arguments

  - type: daemon
    name: OutDaemon
    components:
      - class: !!python/object:trollflow_sat.satpy_writer.DataWriterContainer
        save_settings: [...]
```

Example with SatPy

- Product configuration file (YAML)

```
groups:
  europe:
    - euron1

product_list:
  euron1:  # Name of the area definition
    areaname: euron1
    products:
      overview:  # Name of the composite definition
        productname: overview
        output_dir: /tmp
        fname_pattern:
          "{time:%Y%m%d_%H%M}_{areaname}_{productname}.{format}"
        formats:
          - format: tif
            writer: null
          - format: nc
            writer: cf
```

Example with SatPy

- Daemon items – I/O “plug-ins”

```
class ListenerContainer(object):
    def __init__(self, topics):
        self.thread = None
        self.output_queue = Queue()
        self.listener = None
        self._init_listener()

    def _init_listener(self):
        self.listener = Listener(topics=topics,
                                 queue=self.output_queue)
        self.thread = Thread(target=self.listener.run)
        self.thread.setDaemon(True)
        self.thread.start()

    def stop(self):
        self.listener.stop()
        self.thread.join()
        self.thread = None
```

Example with SatPy

- Message listening daemon

```
class Listener(Thread):
    def __init__(self, topics=None, queue=None):
        self.subscriber = None
        self.topics = topics
        self.create_subscriber()
        self.queue = queue
        self._loop = False

    def run(self):
        while self._loop:
            message_data = self.subscriber.get()
            self.queue.put(message_data)

    def stop(self):
        self._loop = False
        self.subscriber.stop()
```

Example with SatPy

- Workflow item: SceneLoader

```
class SceneLoader(AbstractWorkflowComponent):

    def invoke(self, context):
        msg = context['content']
        output_queue = context['output_queue']

        with open(context['product_list'], 'r') as fid:
            product_config = yaml.load(fid)

        for area_grp in product_config['groups']:
            glbl_scn = Scene(**parse_message(msg))
            glbl_scn.load(get_composites(product_config,
                                         group))
            glbl_scn.info.update(get_metadata(product_config,
                                              area_grp))
            output_queue.put(glbl_scn)
```

Example with SatPy

- Workflow item: Resampler

```
class Resampler(AbstractWorkflowComponent):

    def invoke(self, context):
        glbl_scn = context['content']
        output_queue = context['output_queue']

        with open(context['product_list'], 'r') as fid:
            product_config = yaml.load(fid)
            prod_list = product_config['product_list']

        for area_id in prod_list:
            lcl_scn = \
                global_data.resample(**get_kwargs(context))
            lcl_scn.info.update(get_metadata(prod_list,
                                             area_id))
            output_queue.put(lcl_scn)
```

Example with SatPy

- Output daemon - DataWriter

```
class DataWriter(Thread):

    def __init__(self, queue=None):
        self.queue = queue  # input queue!
        self._loop = False

    def run(self):
        while self._loop:
            scn = self.queue.get(True, 1)
            prod_ids = scn.info['product_ids']
            fnames = scn.info['filenames']

            for i in len(prod_ids):
                scn.save_dataset(prod_ids[i],
                                 filename=fnames[i])
```

My satellite is full of eels!