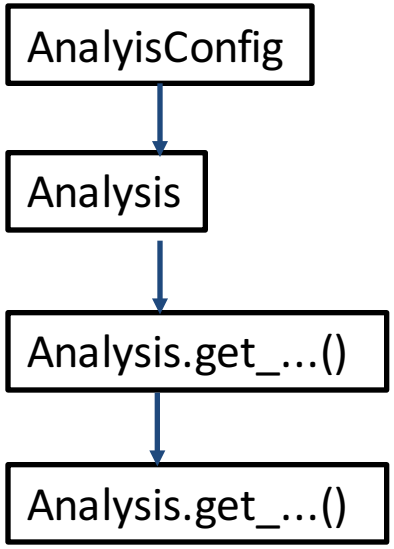
Analysis class re-design and parallelisation

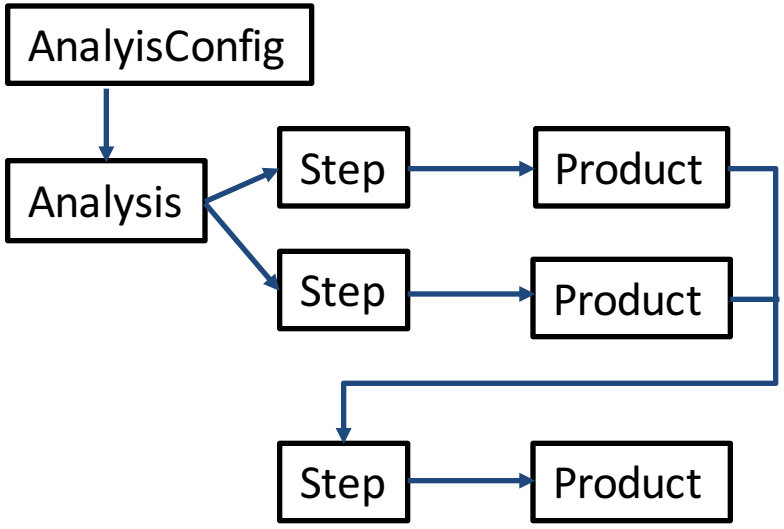


Run each analysis functions :

```
analysis = Analysis(config)
analysis.get_observations()
analysis.get_datasets()
analysis.get_excess_map()
```

Problems :

- Only one hard-coded workflow
- Everything into one class
- Config not fully validated so can fail late
- Not parallel



Run the full workflow :

```
analysis = Analysis(config)
analysis.run()
```

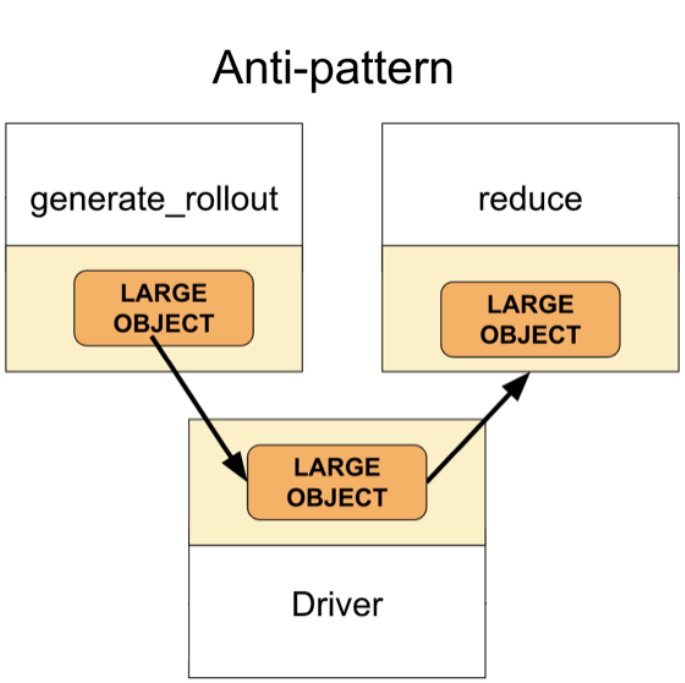
or from command line:

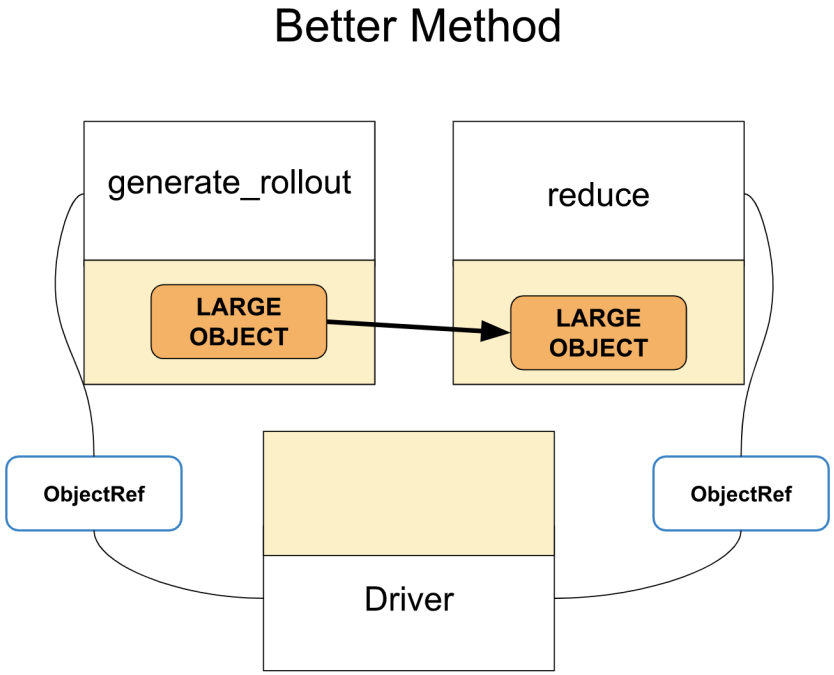
```
gammapy analysis run --filename config.yaml --out .
```

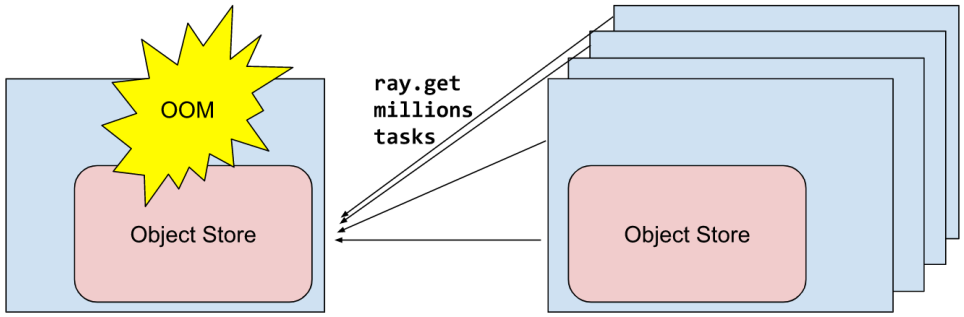
Advantages:

- More flexible and extensible
- Config not fully validated
- can check steps config on init
- Parallel

Exchange products between steps







- Avoid to store all products in the Analysis class but pass only references
- Exchange required products directly between steps

Example

```python
class Products(MutableSequence):
    def __init__(self, products=None):
        if products is None:
            products = []
        self._products = products

    @property
    def names(self):
        return [p.name for p in self._products]

    @property
    def data(self):
        return [p.data for p in self._products]

    def index(self, key):
        if isinstance(key, (int, slice)):
            return key
        elif isinstance(key, str):
            return self.names.index(key)

    def select(self, tag=None):
        #could select by step_name, tag, name)
        mask = np.ones(len(self), dtype=bool)
        for idx, p in  enumerate(self._products):
            mask[idx] = p.tag==tag
        return self[mask]

    def get(self):
        ind = np.where([p.needs_update for p in self._products])[0]
        refs_to_run = {data for data in np.array(self.data)[ind]} #unique objects
        results = ray.get(list(refs_to_run))
        products = [item for sublist in results for item in sublist] #flatten
        for product in products:
            ind = np.where(np.array(self.names) == product.name)[0]
            for idx in ind:
                self[int(idx)] = product
```

```python
class Product:
    def __init__(self, tag=None, step_name=None, data=None):
        self.tag = tag
        self.step_name = step_name
        self.name = make_name() #unique id from make_name
        self.data = data

    @property
    def needs_update(self):
        return isinstance(self.data, ray.ObjectRef)
```

```python
class AnalysisStep:
    def __init__(self, name=None, parallel=True):
        self.parallel = parallel
        self.name = make_name(name)
        self._data = Products([])

    @property
    def data(self):
        return self._data

    def set_data(self, data, extend=True):
        if isinstance(data, Product):
            data = Products([data])
        selection = self._select_data(data)
        if extend:
            self._data.extend(selection)
        else:
            self._data = selection

    def _select_data(self, data):
        selection = Products([])
        for req in self.required_products:
            selection.extend(data.select(**req))
        return selection

    def run(self, data, parallel=None, extend=True):
        self.set_data(data, extend=extend)
        #copied or reference if data is on the object store ?
        if parallel is None:
            parallel = self.parallel
        if parallel:
             run_parrallel = ray.remote(self.__class__._run)
             ref = run_parrallel.remote(self)
             for p in self.products:
                 p.data = ref
        else:
             self._run()
        return self.products
```

```python
class SumStep(AnalysisStep):
    tag = "sum"
    required_products = [{"tag":"value"}]

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.products = Products([Product(tag="value",
                                          step_name=self.name,
                                          data=None),
                                  Product(tag="log",
                                          step_name=self.name,
                                          data=None)
                                  ])

    def _sum(self):
        return np.sum(self.data.data)

    def _run(self):
        self.data.get() #wait other remote and set results on self.data
        time.sleep(1)
        self.products[0].data = self._sum()
        self.products[1].data = "done"
        return self.products
```

```python
step = SumStep()
step.run(data=Product(tag="value", data=1), parallel=True)
step.run(data=Product(tag="value", data=2), parallel=True)
step.run(data=step.products)
step.products.get()
```

Only init no heavy computation

Run step 1 & 2 in parallel, wait completion, run step 3

Config example

```
AnalysisConfig
    general:
        log: {level: info, filename: null, filemode: null, format: null, datefmt: null}
        outdir: .
        n_jobs: 1
        datasets_file: null
        models_file: null
    observations:
        datastore: $GAMMAPY_DATA/hess-dl3-dr1
        obs_ids: []
        obs_file: null
        obs_cone: {frame: icrs, lon: 83.633 deg, lat: 22.014 deg, radius: 5.0 deg}
        obs_time: {start: null, stop: null}
        required_irf: [aeff, edisp, psf, bkg]
    datasets:
        type: 3d
        stack: true
        geom:
            wcs:
                skydir: {frame: icrs, lon: 83.633 deg, lat: 22.014 deg}
                binsize: 0.02 deg
                width: {width: 2.0 deg, height: 2.0 deg}
                binsize_irf: 0.2 deg
            selection: {offset_max: 2.5 deg}
            axes:
                energy: {min: 1.0 TeV, max: 10.0 TeV, nbins: 10}
                energy_true: {min: 0.5 TeV, max: 20.0 TeV, nbins: 20}
        map_selection: [counts, exposure, background, psf, edisp]
        background:
            method: fov_background
            exclusion: $GAMMAPY_DATA/joint-crab/exclusion/exclusion_mask_crab.fits.gz
            parameters: {method: scale}
        safe_mask:
            methods: [aeff-default]
            parameters: {}
        on_region: {frame: null, lon: null, lat: null, radius: null}
        containment_correction: true
    excess_map:
        correlation_radius: 0.1 deg
        parameters: {}
        energy_edges: {min: null, max: null, nbins: null}
    steps:
    -   name: hess1_data-reduction
        tag: data-reduction
        products: datasets_hess1
        config:
            observations:
                datastore: data/hess1
    -   name: hess1u_data-reduction
        tag: data-reduction
        products: datasets_hess1u
        config:
            observations:
                datastore: data/hess1u
    -   name: hess_excess_map
        tag: excess_map
        required_products: [datasets_hess1, datasets_hess1u]
        products: hess_flux_maps
```

How to proceed ?

- PR 3852 : github.com/gammapy/gammapy/pull/3852
  move analysis.get_...() methods to AnalysisStep classes
  add analysis.run() method and cli equivalent

- Add AnalysisProduct and AnalysisProducts classes

- Add PARALLEL_BACKEND = … (where ?)
  and add a mecanism to switch from multiprocessing to ray.util.multiprocessing
  (in TSmapEstimator and DatasetsMaker)
  docs.ray.io/en/latest/ray-more-libs/multiprocessing.html

- Introduce parallel support in AnalysisStep and AnalysisProducts (optional by default)

- Refactor AnalysisStep to define all maker/Estimators on init (an run them on run)

- Provide new configs to run various analysis workflows

- Add tutorial notebook