waluigi - Beyond luigi

Benjamin Fischer

PyHEP.dev
August 28th 2024

SPONSORED BY THE
Federal Ministry of Education and Research

IDIUM
RWTH AACHEN UNIVERSITY

Luigi's strengths:

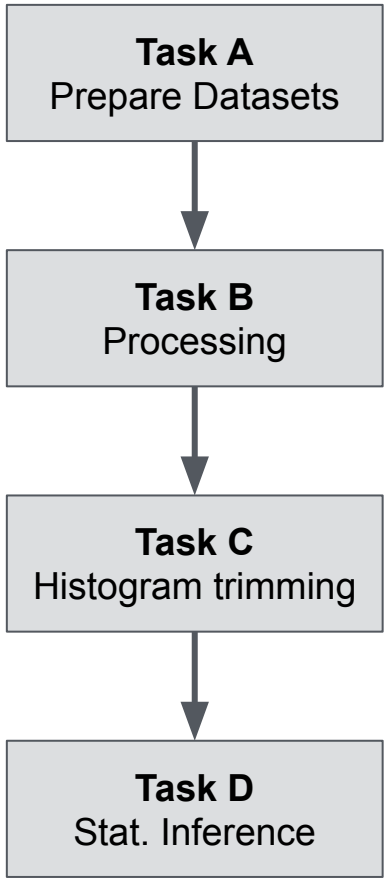
- nice web interface
- well supported
- powerful workflow description
  - multiple inputs & outputs
  - customizable “targets”
    i.e. local & remote files
  - **full task parametrization**

➜ similar to dataclasses

➜ effectively unique feature

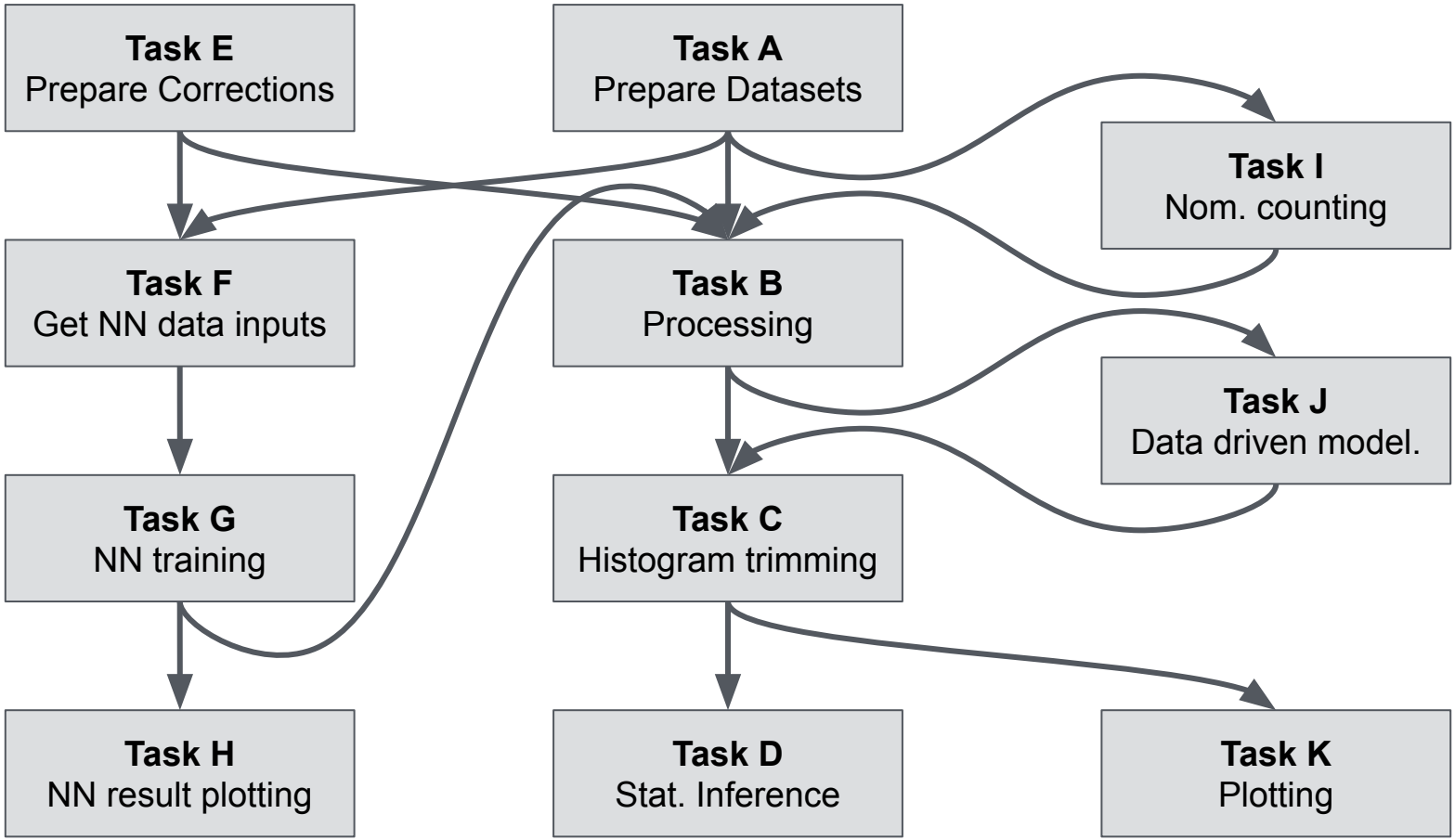
Example Graph:

Data flow

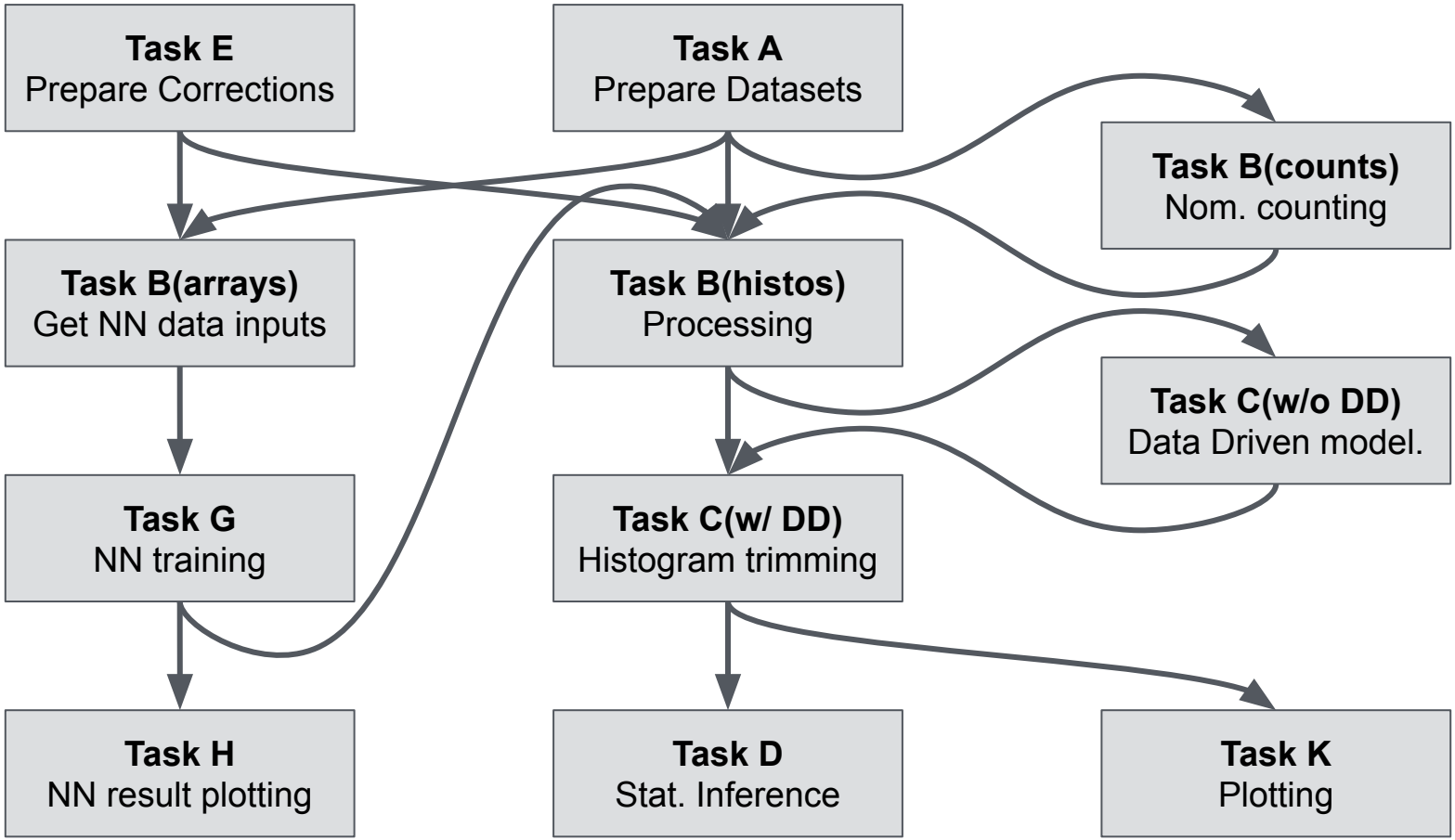


A more complex/realistic graph

RWTH AACHEN UNIVERSITY
Benjamin Fischer, August 28th 2024

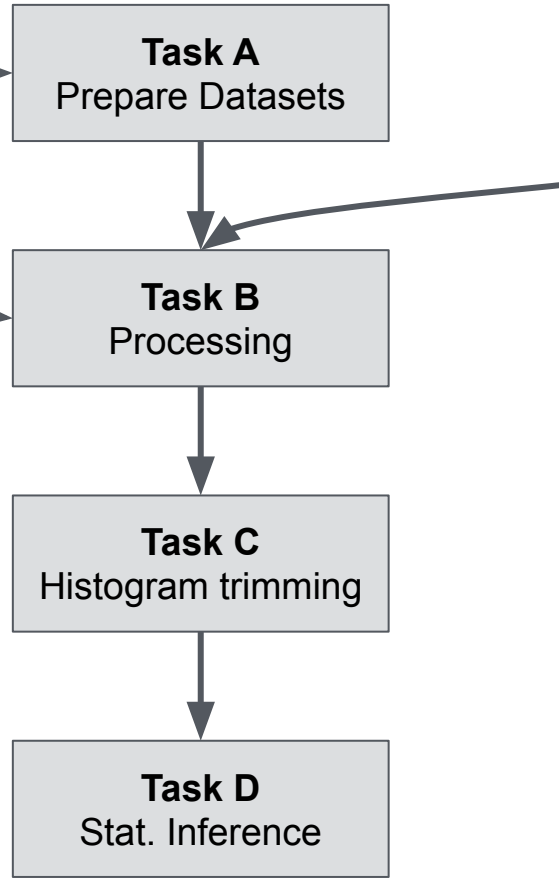


Now with parameters



5 An Example Task

```python
import luigi
from somewhere import TaskE

class TaskA(luigi.Task):
    ...

    def get_all_paths():
        ...

class TaskB(luigi.Task):
    what = luigi.ChoiceParameter(
        choices=["histos", "array", "counts"],
        description="what to produce when processing data files"
    )

    def requires(self):
        return {
            "datasets": TaskA(),
            "corrections": TaskE(),
        }

    def output(self):
        luigi.target.FileSystemTarget("/some/path/somewhere.pickle")

    def run(self):
        from coffea import magical_do_it_all_function, load, save

        output = magical_do_it_all_function(
            datasets = self.requires()["datasets"].get_all_paths(),
            corrections = load(self.inputs()["corrections"].path)
        )

        save(self.output().path, output)

```

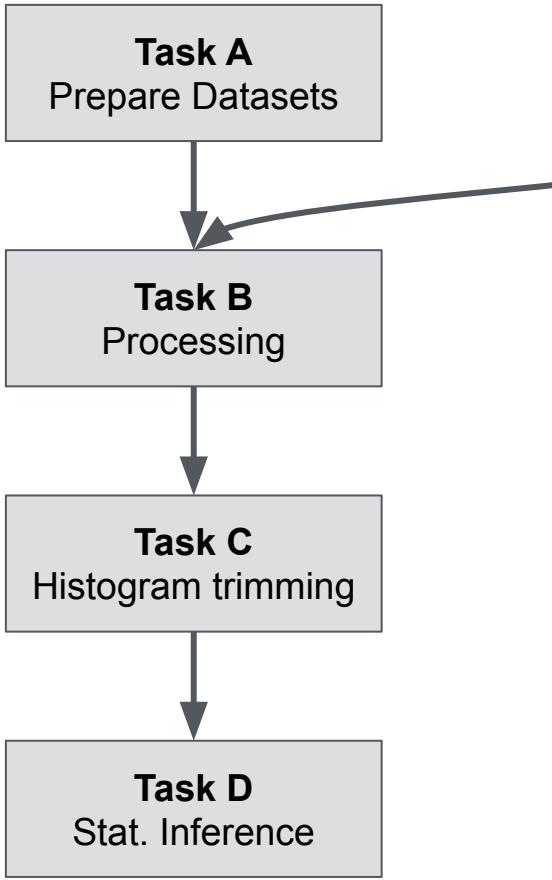


Parameters: Boon & Bane

- extremely useful - but
  - cumbersome with large graphs
- somewhat addressed with: luigi.util.inherits, …
  - not sufficient for dynamic/complex graphs

Example: Task A need a new parameter

```
5   class TaskA(luigi.Task):
6       year = luigi.IntParameter(default=2016, description="year of data taking")
7
8       ...
9
10  @luigi.util.inherits(TaskA)
11  class TaskB(luigi.Task):
12      def requires(self):
13          return {
14              "datasets": TaskA(year=self.year),
15              "corrections": TaskE(),
16          }
```



RWTH AACHEN UNIVERSITY
Benjamin Fischer, August 28th 2024

luigi is great!

But it could be better.

Up next:

- a rough task concept
  - based on several ideas
- addressing pain points:
  - first & second hand
- no implementation (or name) yet
  - only mockup API usage
- no intent to reinvent the wheel:
  - make use of law, etc

law
luigi analysis workflow

- feedback & discussion wanted!
  - what bugs you about luigi?

```
287
288 class FlatDatacardsTask(FlatDatacardsCommand, DHI, DatacardProvider):
289     pass
290
291
292 class MultiDatacardsTask(MultiDatacardsCommand, DHI, DatacardProvider):
293     pass
294
```

```
578 class FitDiagnosticsCombinedPostprocessed(FitDiagnosticsPostprocessed, FlatDatacardsTask):
579     def requires(self):
580         return FitDiagnosticsCombined.req(self)
581
582
583 class FitDiagnosticsFrequentistToysPostprocessed(
584     FitDiagnosticsFrequentistToysMixin, FitDiagnosticsPostprocessed, FlatDatacardsTask, PoolMap
585 ):
```

```
1182 class InferenceCombined(
1183     ModelMixin,
1184     RecipeMixin,
1185     CreateIssueMixin,
1186     CombinationTask,
1187     law.WrapperTask,
1188 ):
1189     dhi_command = DHITask.dhi_command
1190     no_poll = DHI.no_poll
```

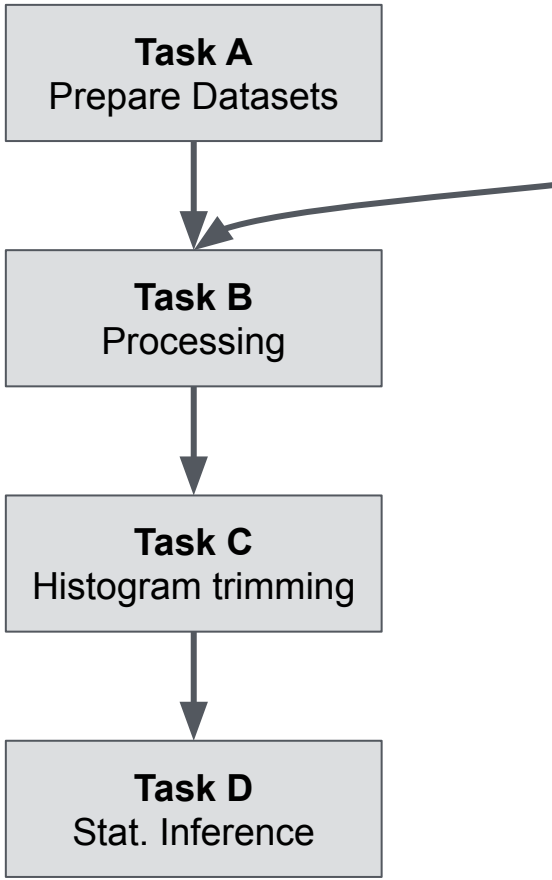
Idea: Parameterized Objects

- it get worse! i.e. with:
  - dynamic dependencies
  - multiple dependencies
    same Task w/ different parameters

➜ this needs addressing
  - but, unlikely to work within luigi's design

Idea:

1. dynamically "inherit" parameters of dependencies
2. dependencies are parameters
3. recurse through "parameterized objects"



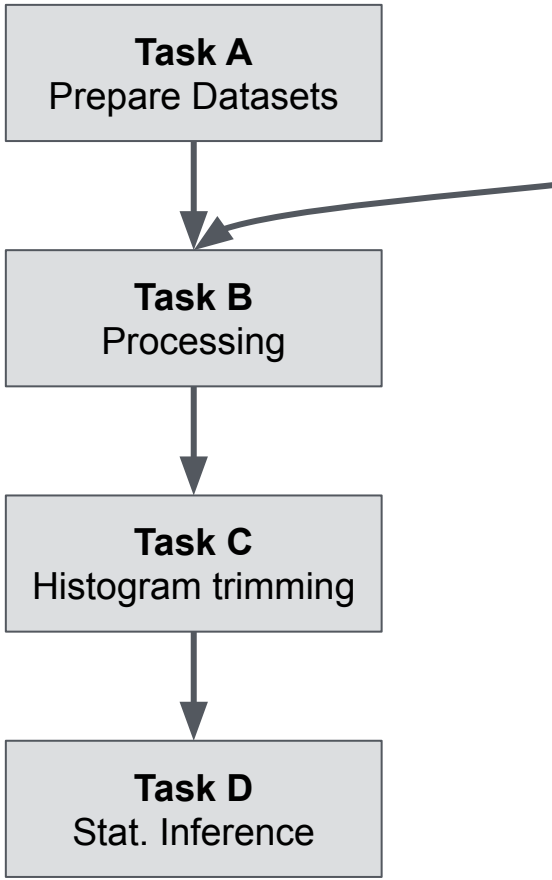
Idea: Parameterized Objects

- it get worse! i.e. with:
  - dynamic dependencies
  - multiple dependencies
    same Task w/ different parameters

➜ this needs addressing
  - but, unlikely to work within luigi's design

Idea:

1. dynamically "inherit" parameters of dependencies
2. dependencies are parameters
3. recurse through "parameterized objects"



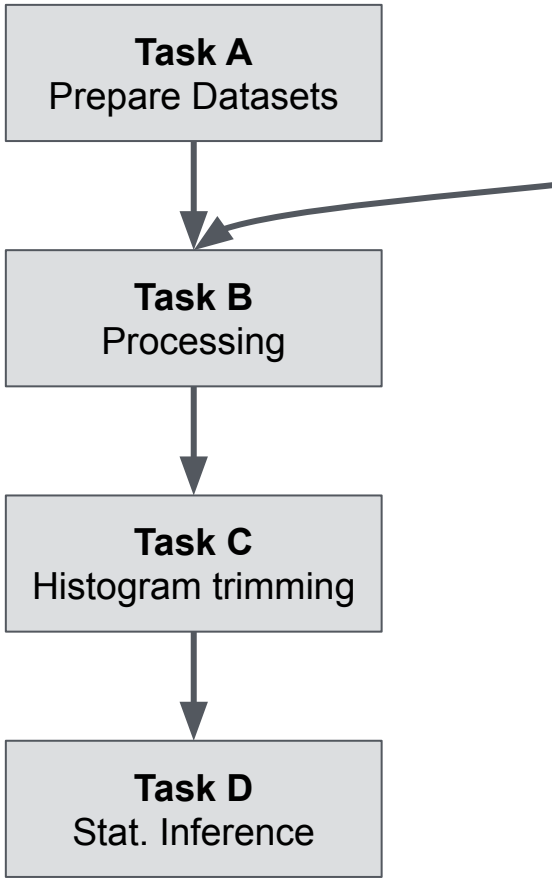
Idea: Path-like addressing parameters

use path-like schema to address parameter, for both:

1. **value access**
2. defining values

Example:

```python
from not_luigi import Task, Parameter

class TaskA(Task):
    year: int = Parameter(default=2012)

class TaskB(Task):
    datasets: TaskA = Parameter()

    def run(self):
        self.datasets.year
```



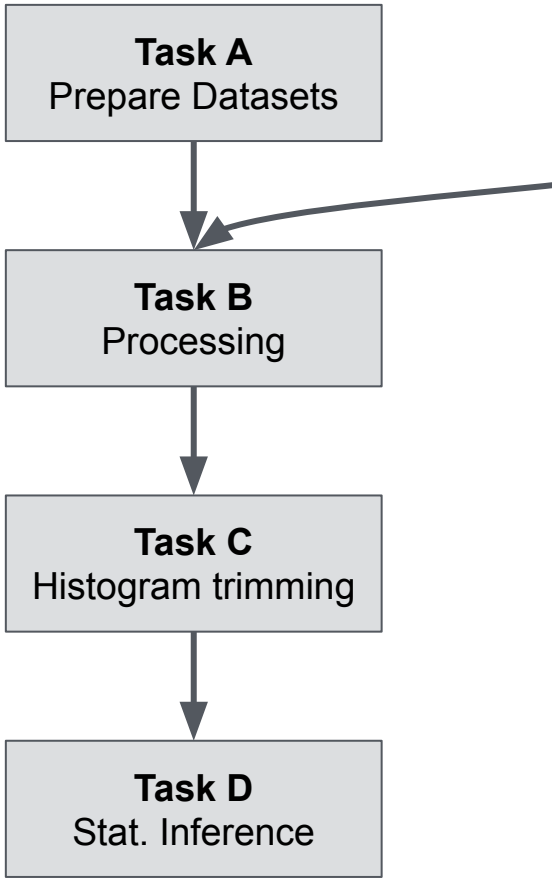
Idea: Path-like addressing parameters

use path-like schema to address parameter, for both:

1. value access
2. **defining values (python)**

Example:

```python
TaskC({
    # increasing specificity
    "year": 2024,
    ("TaskA", "year"): 2024,
    (TaskA, "year"): 2024,
    TaskA.year: 2024,
    # or even more explicit
    (TaskB, TaskA.year): 2024,
    (TaskB.datasets, TaskA.year): 2024,
    "TaskA.datasets.year": 2024,
})
```



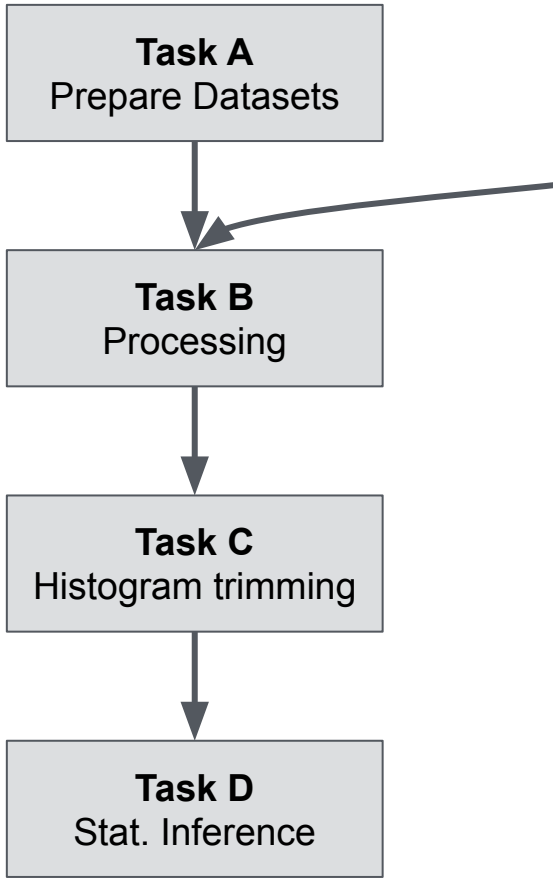
Idea: Path-like addressing parameters

use path-like schema to address parameter, for both:

1. value access
2. **defining values (shell)**

Example:

```
not_luigi TaskC --year=2024
not_luigi TaskC --TaskA.year=2024
not_luigi TaskC --TaskA.year=2024
not_luigi TaskC --TaskB:TaskA.year=2024
not_luigi TaskC --TaskB.datasets:TaskA.year=2024
not_luigi TaskC --TaskB.datasets.year=2024
```



Idea: Automatic Task Collections

Task Collection:

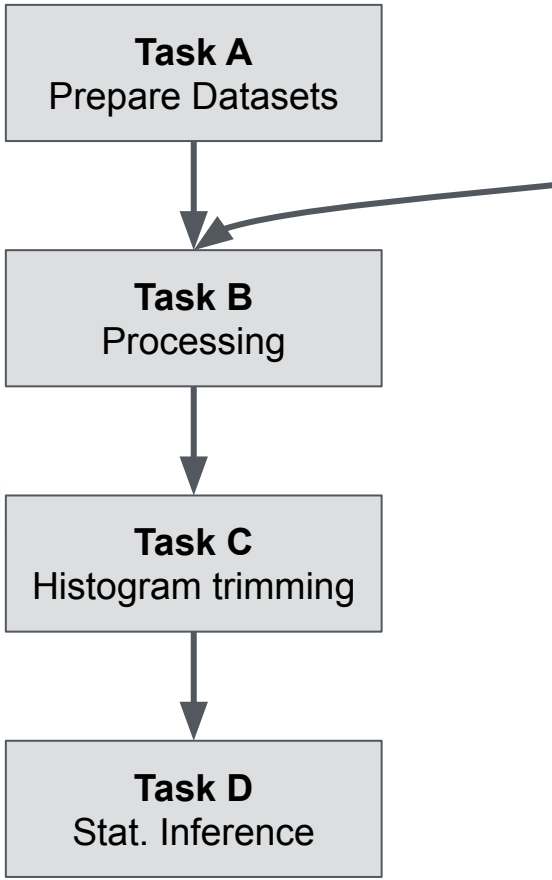
- instances of task with different parameters
- in luigi: need to write dedicated tasks “Wrappers”
- should be automatic feature

Example:

- python: `TaskA(year=[2016, 2017, 2018])`
- shell: `not_luigi TaskA --year=2016,2017,2018`

Details:

- can occur in deeply nested dependencies
- propagated via Exceptions (similar to StopIteration)
- caught & handled whenever desired (reduction Task)



Idea: Task Substitution

- replace a task (implementation)with another one
- deeply within the dependency graph
- without, need to rewrite everything

Examples:

- python:

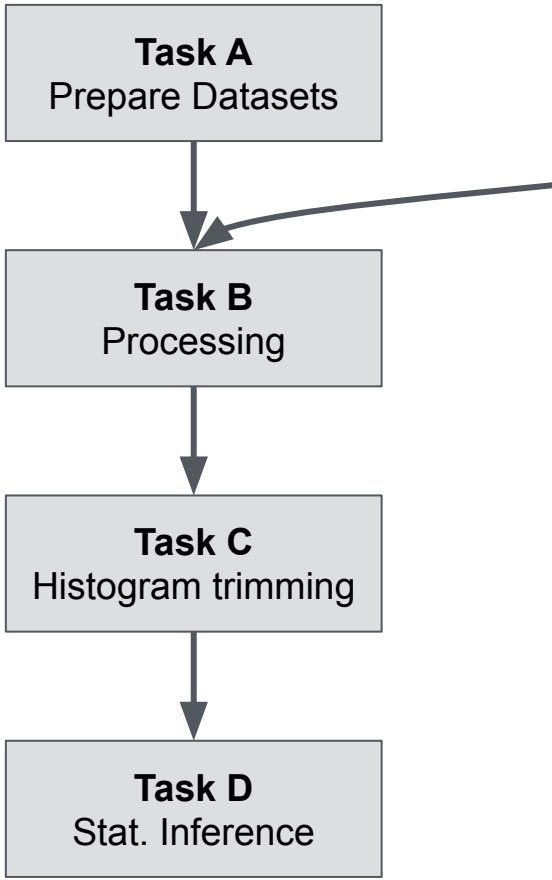
```
21  class TaskBcustom(TaskB):
22      ...
23
24  TaskZ({TaskB: TaskBcustom})
```

- shell:

```
not_luigi TaskZ --TaskB=TaskBcustom
```

improves reusability, both way:

- reuse a graph, plugging in new tasks
- transplanting task/graph portions into own graph



More Parameter Ideas

- parameter carry arbitrary auxiliary data
  e.g. sampling information for hyperopt.

special parameter types:

- const parameter:
  - e.g. indicate code changes
- dynamic parameter:
  - like a property, return anything
- actions:
  - methods to be called
  - in order given (even multiple times)

```
not_luigi TaskG --cleanup
```

```python
class TaskG(Task):
    learning_rate: float = Parameter(
        default=1e-3,
        range=[1e-2, 1e-5],
        sample="loguniform",
    )

    code_version = ConstParameter(1)

    @Parameter.dynamic
    def maybe_important(self):
        if self.other_parameter:
            return TaskO(self)

    @Action
    def cleanup(self):
        ... # do someting

    ...
```

is nice - should be be better - but not compatible with core design

➜ time to think about something new

Design Ideas

- Parameterized Objects
- Dependencies via parameters
- Path-link parameter addressing
- Automatic Task Collections
- Task Substitution
- Special Parameters:
  Const~, Dynamic~, Action

```python
from not_luigi import Task, Parameter

class TaskA(Task):
    year: int = Parameter(default=2012)

class TaskB(Task):
    datasets: TaskA = Parameter()

    def run(self):
        self.datasets.year
```

```python
TaskC({
    # increasing specificity
    "year": 2024,
    ("TaskA", "year"): 2024,
    (TaskA, "year"): 2024,
    TaskA.year: 2024,
    # or even more explicit
    (TaskB, TaskA.year): 2024,
    (TaskB.datasets, TaskA.year): 2024,
    "TaskA.datasets.year": 2024,
})
```