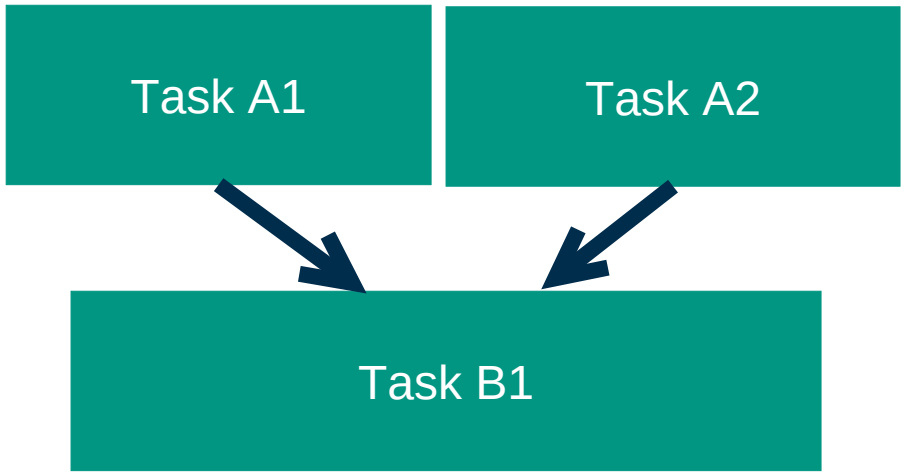
Luigi and the Workflow Managers in HEP

Jonas Eppelt, Alexander Heidelbach, Patrick Ecker, Giacomo de Pietro

What is a Workflow Manager?

From the README of an analysis repo

```
### tautau/mumu/etc ###
To generate the rarer samples look in './ntupleHandling/' where there are
'submit_rareMC_off(on)Resonance.py'. In these are options for which samples, exps and PID selection
to apply. The submit file runs the 'rareMC_ntuple_creation_task.py'. Once you have run the submit
file and the jobs are finished, you can check which ones failed using
'./ntupleHandling/logFiles_checks.ipynb'. Some probably failed due to memory problems, you can rerun
these using the './ntupleHandling/resubmit_off(on)Resonance_rareMC.py'.


### training BDT1 ###

Training of BDT1 is done using 'train_classifier.py'. In this file you can set the variables to be
trained on, and also you must specify the training and testing samples.

You can create the train/test samples using './MVA/create_train_and_test(__withRareMC).ipynb'. You
may want to run './MVA/sample_bgd(sig).py' first to create subsamples across all of the experiments.
To actually run these you should use './MVA/submit_bgd(sig)Sampling.py' where you can specify the
background types and experiments to use.
```

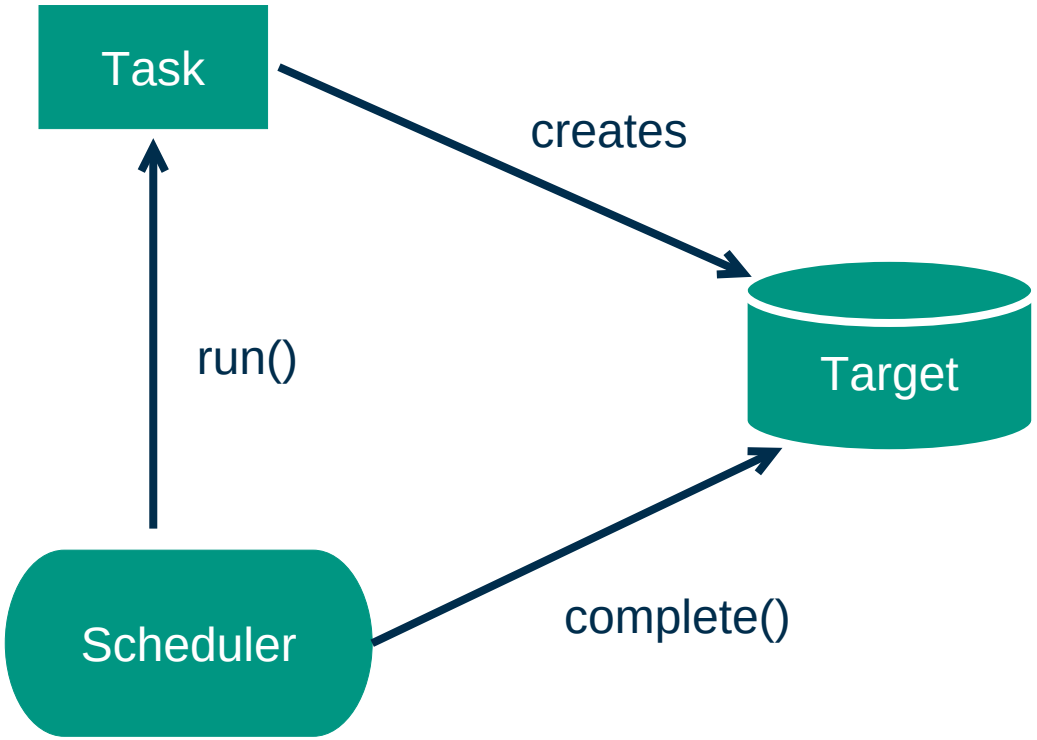


A Workflow Manager
- resloves task dependencies.
- orchastrates the task execution.
- handles task failures.
- makes things reproducible.
- handles computing systems.

What is Luigi?

- Python package for Workflow Management
- Developed by Spotify in 2012
  - Maintained by Spotify
  - But no new features anymore

```
class MyTask(luigi.Task):
  parameter = luigi.Parameter()

  def run(self):
    func(self.parameter)

  def output(self):
    yield luigi.Target(„output.txt“)

  def requires(self):
    yield MyPreviousTask()
```



Why do we need something on top of luigi?

- Connect with common HEP computing tools
  - HTCondor, WLCG, Slurm, Root, XRootD,....
- (HEP specific) improvements
  - Better CLI interface
  - Boilerplate reduction
- Working around luigi‘s limitations
  - Process number limit

law
luigi analysis workflow

HEP adaption of Luigi

B2luigi – bringing batch 2 luigi

- Created 2018 by Nils Braun
- Since 2023 maintained by Belle II collaboration
- Also non-Belle II adaption

Law – luigi analysis workflow

- Created 2016 by Marcel Rieger
- Widespread adaption within CMS and other LHC-experiments

Philosophies	
• Light-weight, Drop-in replacement for luigi • Goal: Make batch systems simple for luigi.	• Build „on-top“ of luigi • Goal: Build complex workflows with luigi

Common Features	
• Improved CLI • Remote Target support	• Batch system submission • (some) Sandboxing

Highlights	
• Automatic file management • (upcoming) TUI	• Extensive contrib ecosystem • Support for complex & large pipelines

Feature Highlight: Automatic File Management

Activates temporary file context:
- Copy input files on entry
- Copy output files on exit

Access temporary input file

Creates temporary output file

Creates a Target with path ../param=<self.param>/out.txt

```python
class MyTask(b2luigi.Task):
  param = luigi.Parameter()

  @b2luigi.on_temporary_files()
  def run(self):

    result = func(
               self.param,
               self.get_input_file_names(„in.txt“)
             )
    write_output(
      self.get_output_file_names(„out.txt“)
    )

  def output(self):
    yield self.add_to_output(„out.txt“)
```

Feature Highlight: law Workflows

A law workflow is a Task with several branches.

- Allows to „package“ many simple Tasks into one large one
- Circumvents parallel process limits on submit machines
- Can make large Workflows easier to handle

```python
class MyTask(law.Task, law.LocalWorkflow):

  def create_branch_map(self):
    return {i: i in range(10)}

  def run(self):
    num=self.branch_data
    output = self.output()
    output.dump(num)
```

The Road ahead

Law:

- Release 1.0 planned for late this summer
  - Improved internals (CI/CD, linting, type checks)

B2luigi:

- Release 1.3 planned for this autumn
  - Beta version of TUI
  - New CLI
  - Small improvements & Bugfixes
- Planned for the future:
  - Support for agentic development (skills, AGENT.md)
  - Improved Documentation
  - Re-building GUI
  - ...

Looking beyond luigi:
Luigi is old and has its limits.
What is the non-HEP world using?

Disclaimer:
I have not used any of the following tools so far.
Please take the following as an incomplete comparison.

Snakemake

- Created 2012 by Johannes Köster (Bioinformatics)
- Focus on bridging different software/programming language
- Wide adoption in many sciences
  - Included in e.g. Reana

- What luigi has, that Snakemake has not
  - Python-native

- What Snakemake has, that luigi has not
  - Changed input detection
  - (limited) Intra-Task checkpointing
  - Python independent
  - CWL support

```
rule NAME:
    input:
        "path/to/inputfile",
        "path/to/other/inputfile"
    output:
        "path/to/outputfile",
        "path/to/another/outputfile"
    script:
        "scripts/script.py"
```

Airflow

Apache
Airflow

- Created 2014 by Airbnb
- Similar concept to luigi, but more features

- What luigi has, but Airflow does not:
  - Build in Task idempotency
  - Simplicity

- What Airflow has, but luigi does not:
  - Large ecosystem
  - Pipeline scheduling (when to run a pipeline)
  - Intra-Task checkpoints
  - Infrastructure Management
  - Feature rich GUI

```python
t1 = BashOperator(
    task_id="print_date",
    bash_command="date",
)

t2 = BashOperator(
    task_id="sleep",
    depends_on_past=False,
    bash_command="sleep 5",
    retries=3,
)
```

```python
t1.set_downstream(t2)

# This means that t2 will depend on t1
# running successfully to run.
# It is equivalent to:
t2.set_upstream(t1)

# The bit shift operator can also be
# used to chain operations:
t1 >> t2

# And the upstream dependency with the
# bit shift operator:
t2 << t1

# Chaining multiple dependencies becomes
# concise with the bit shift operator:
t1 >> t2 >> t3

# A list of tasks can also be set as
# dependencies. These operations
# all have the same effect:
t1.set_downstream([t2, t3])
t1 >> [t2, t3]
[t2, t3] << t1
```

Dask/Spark

- Main use cases:
  - Heavy parallelization
  - Scaling up of interactive computing
    - Jupyter Notebooks
    - DataFrames, Arrays
    - Shared Storage

- What luigi has, but Dask/Spark does not:
  - Partial DAG execution
  - Task Idempotentcy
  - Task State persistence
  - Opportunistic resources

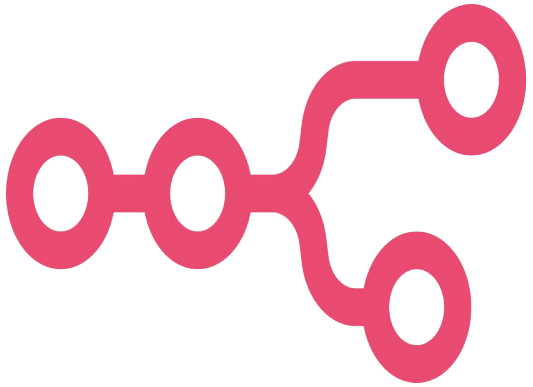
- What Dask/Spark has, but luigi does not:
  - Heavy parallelization
  - Interactive computing
  - Native scaling of DataFrames/Arrays
  - Intra-Task checkpoints

```
def load(filename):
    ...

def clean(data):
    ...

def analyze(sequence_of_data):
    ...

def store(result):
    with open(..., 'w') as f:
        f.write(result)

dsk = {'load-1': (load, 'myfile.a.data'),
       'load-2': (load, 'myfile.b.data'),
       'load-3': (load, 'myfile.c.data'),
       'clean-1': (clean, 'load-1'),
       'clean-2': (clean, 'load-2'),
       'clean-3': (clean, 'load-3'),
       'analyze': (analyze, ['clean-%d' % i for i in [1,
       'store': (store, 'analyze')}

from dask.threaded import get
get(dsk, 'store')  # executes in parallel
```

Freemium Software

- Software usually with similar concept to luigi/airflow, but
  - Often more features
  - Less boilerplate
  - „cloud-native“

- Typical buisness model:
  - Opensource, python-based Workflow Manager
  - Pay for:
    - Priority support
    - Managed scheduler instances (AWS, Azure,...)
    - Implementation
    - Premium features (UI, Multi-user, ...)



Conclusion

- There is no alternative, that exactly replicates luigi functionality

- Snakemake is a viable alternative to luigi
  - Is already in use within HEP

- Airflow often seems too complex
  - More suited for production settings
  - Not so well suited for quickly developing pipelines

- Dask/Spark
  - Widely used in HEP
  - Complementary approach to Workflow management

- Freemium Software
  - A nice place for inspiration