Building Robust Pipelines with Airflow

@erinshellman

Wrangle Conf
July 20th, 2017

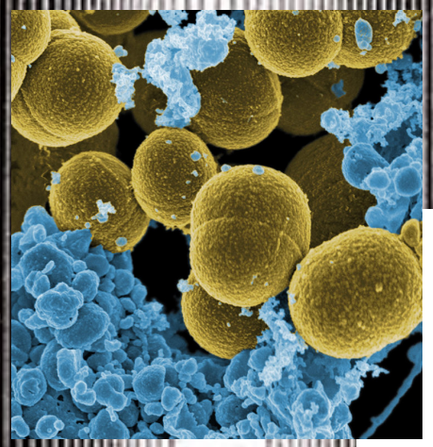
Zymology: is the science of fermentation and it's applied to make materials and molecules

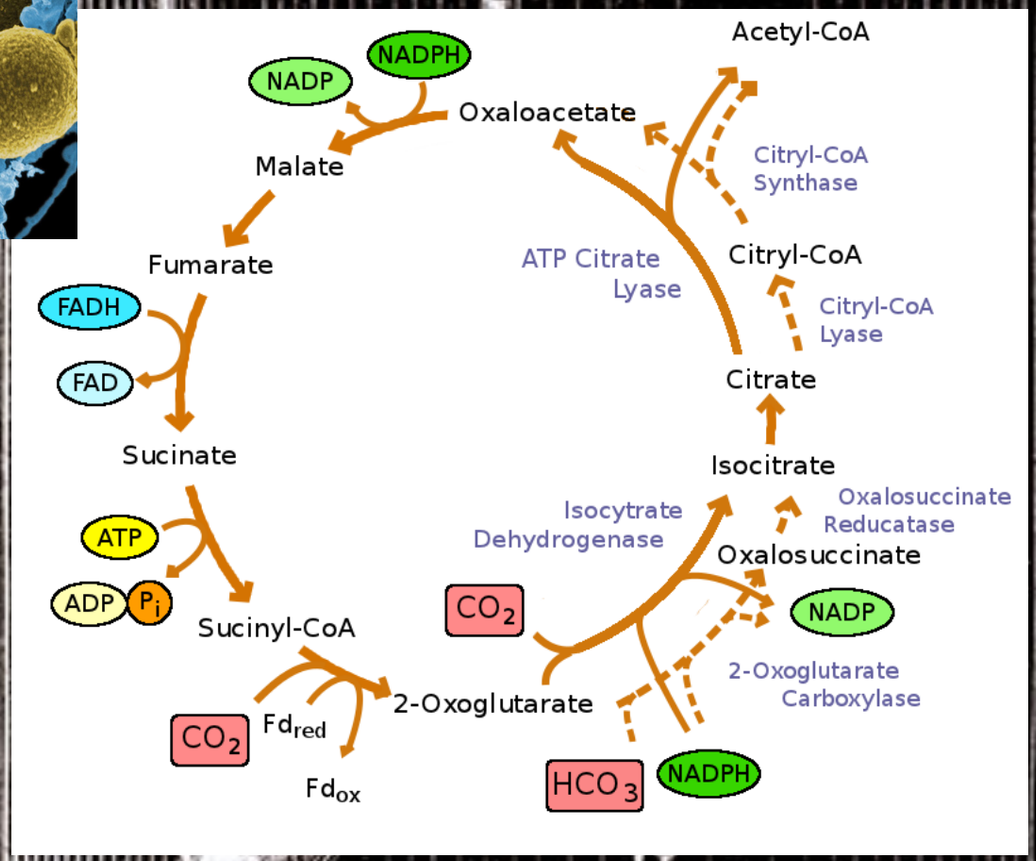
Beer

Insulin

Food additives

Plastics

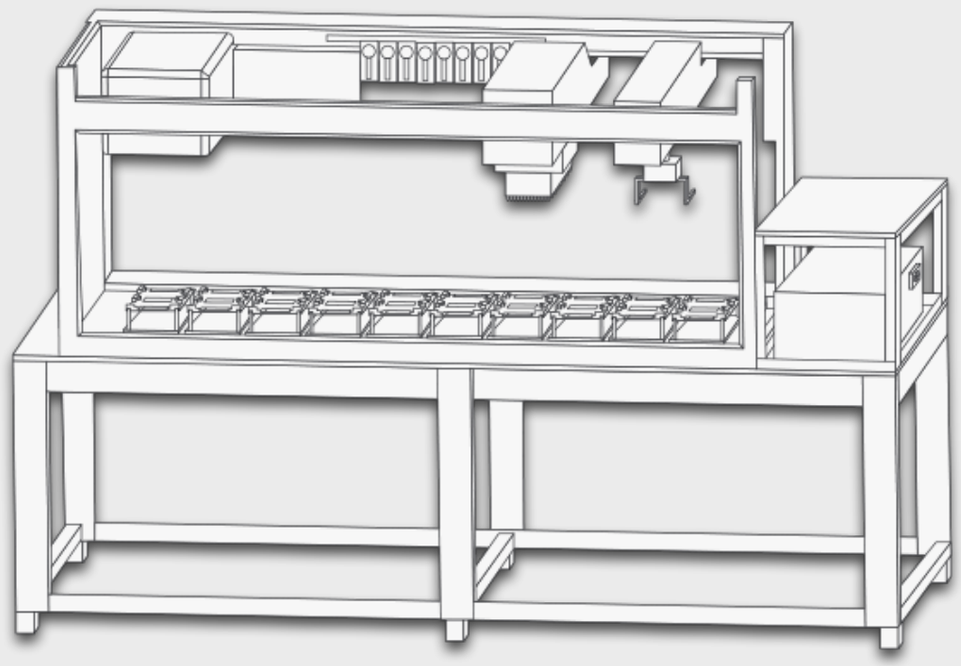



NADPH
NADP
Oxaloacetate
Malate
Fumarate
FADH
FAD
Sucinate
ATP
ADP
P_i
Sucinyl-CoA
CO_2
Fd_{red}
Fd_{ox}
2-Oxoglutarate
CO_2
HCO_3
NADPH
Isocytrate Dehydrogenase
Oxalosuccinate
NADP
2-Oxoglutarate Carboxylase
Isocitrate
Oxalosuccinate Reducatase
Citrate
Citryl-CoA Lyase
Citryl-CoA
ATP Citrate Lyase
Citryl-CoA Synthase
Acetyl-CoA

Zymergen provides a platform for rapid improvement of microbial strains through genetic engineering.

Robotic automation

Our experimentation is increasingly orchestrated with robotics and machine learning.



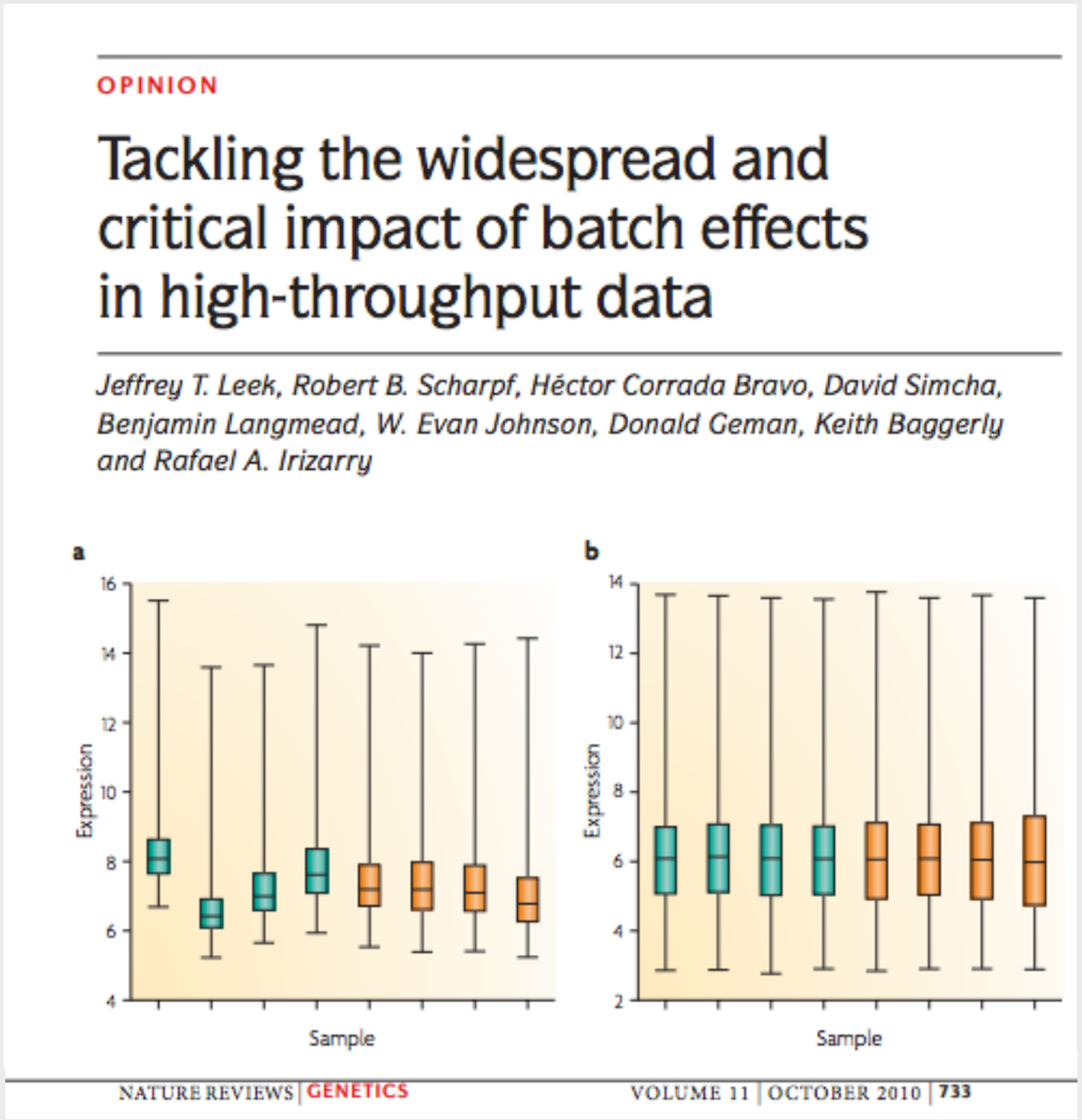
Learning *how* to efficiently navigate the genome is the mission of data science at Zymergen

Blocker: process failure

Orchestrating complex experiments with robots is hard, and there are process failures. These failures often cause sporadic, extreme measurement values.

Blocker: batch effects

We see temporal effects based on when experiments were performed



OPINION

Tackling the widespread and critical impact of batch effects in high-throughput data

Jeffrey T. Leek, Robert B. Scharpf, Héctor Corrada Bravo, David Simcha, Benjamin Langmead, W. Evan Johnson, Donald Geman, Keith Baggerly and Rafael A. Irizarry

a

b

Blocker:
different interpretations of results

We're building a platform that can support any microbe and any molecule.

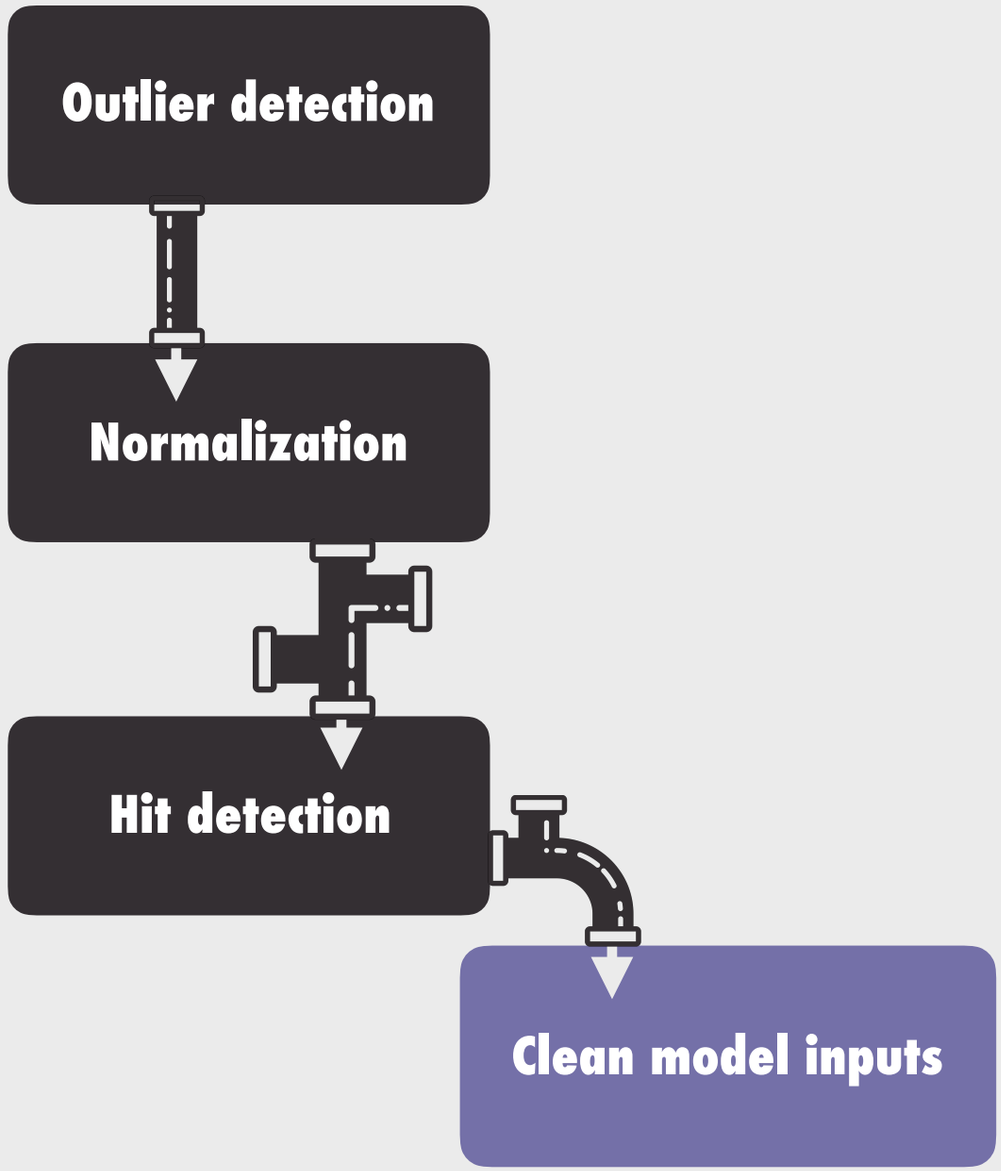
Sometimes that results in a proliferation of solutions with disagreement on which is best.

Processing pipeline

1. Identify process failures

2. Quantify and remove process-related bias

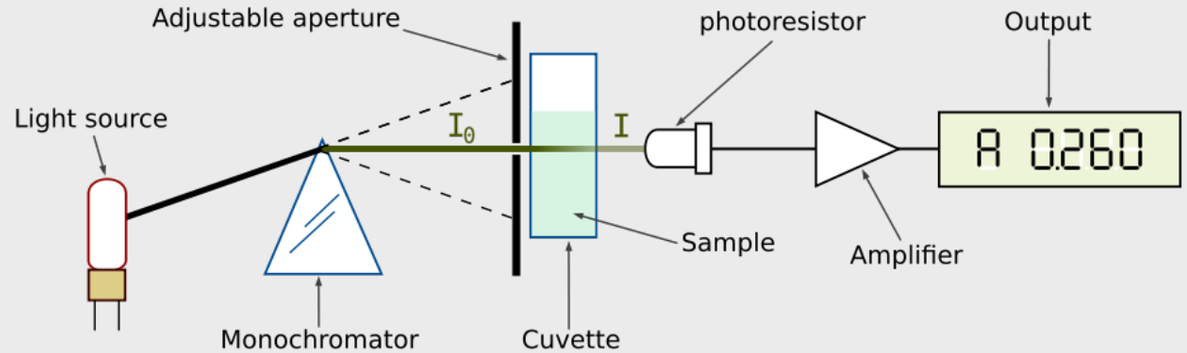
3. Identify strains that show improvement using consistent criteria

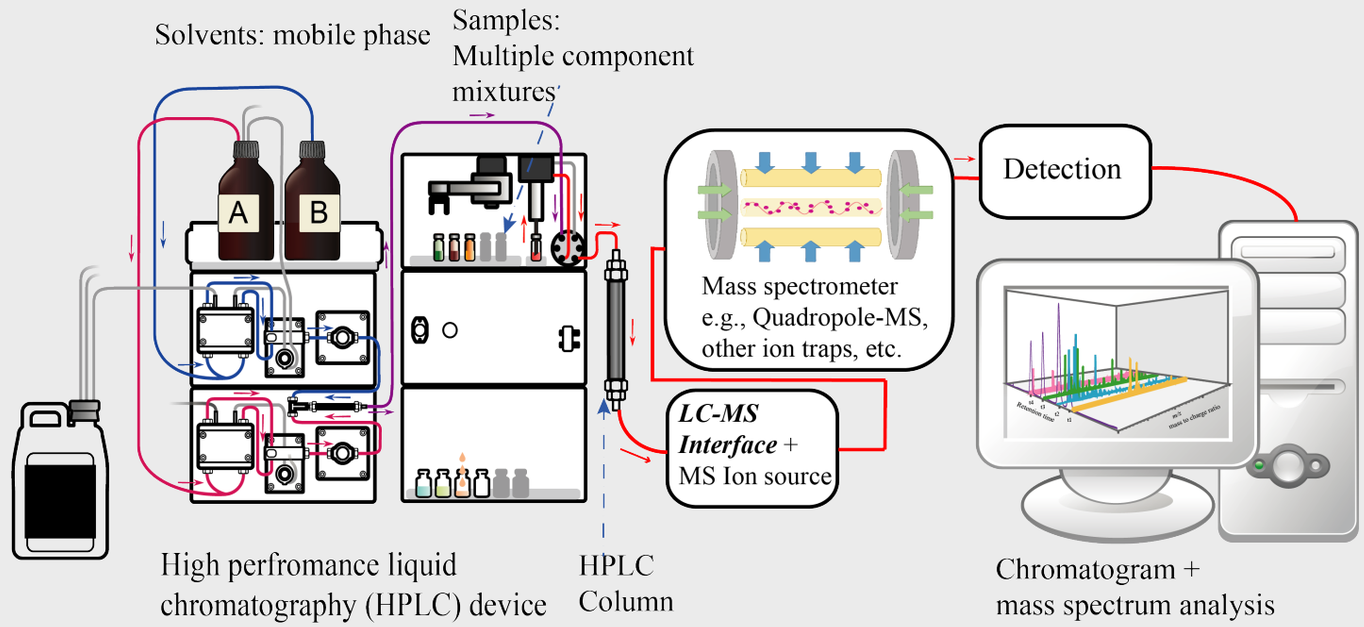


Rolling our own ETL pipeline

There are many ways to measure the concentration of a molecule.

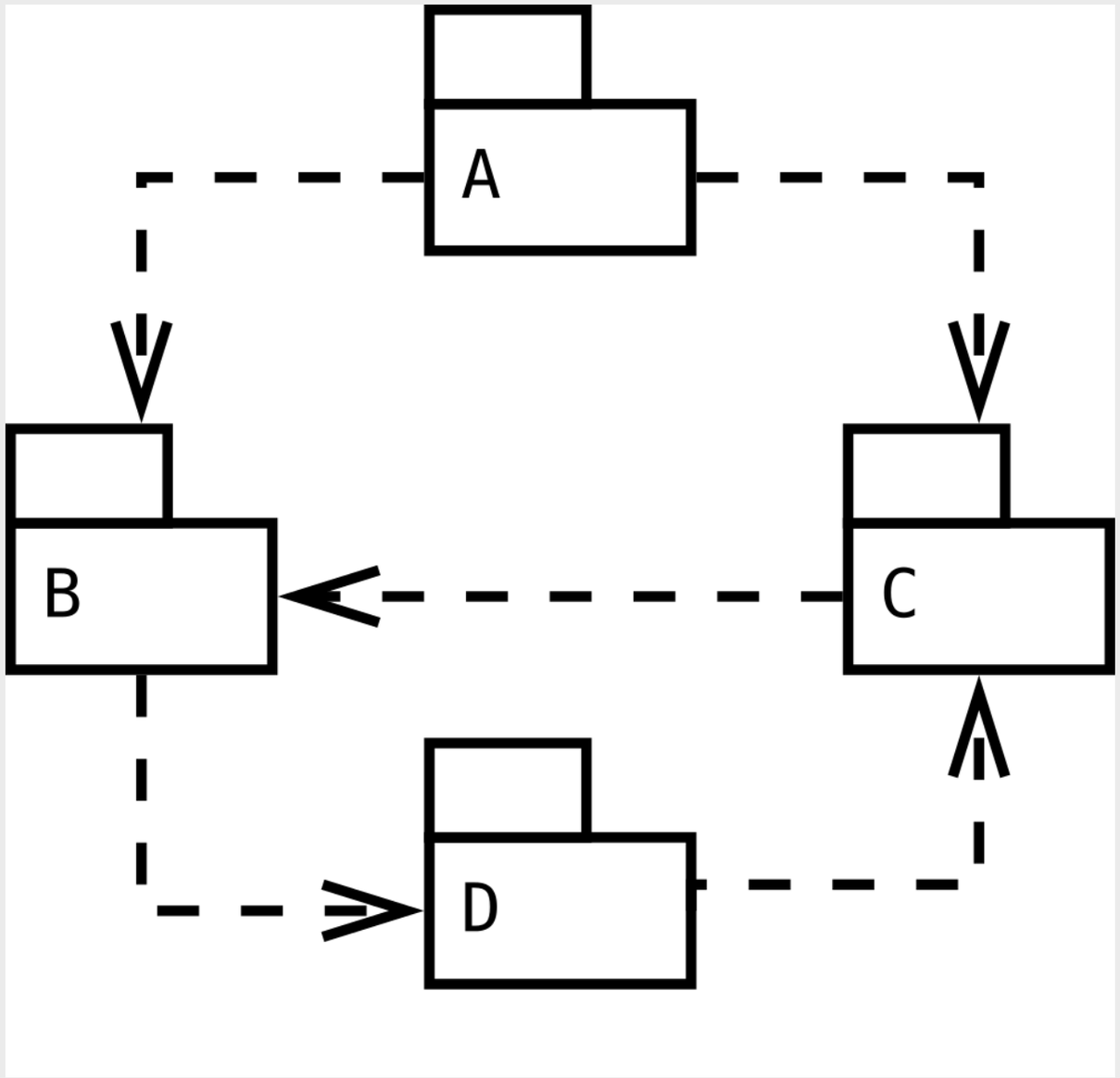
Any microbe, any molecule... any experiment, *many* data formats.



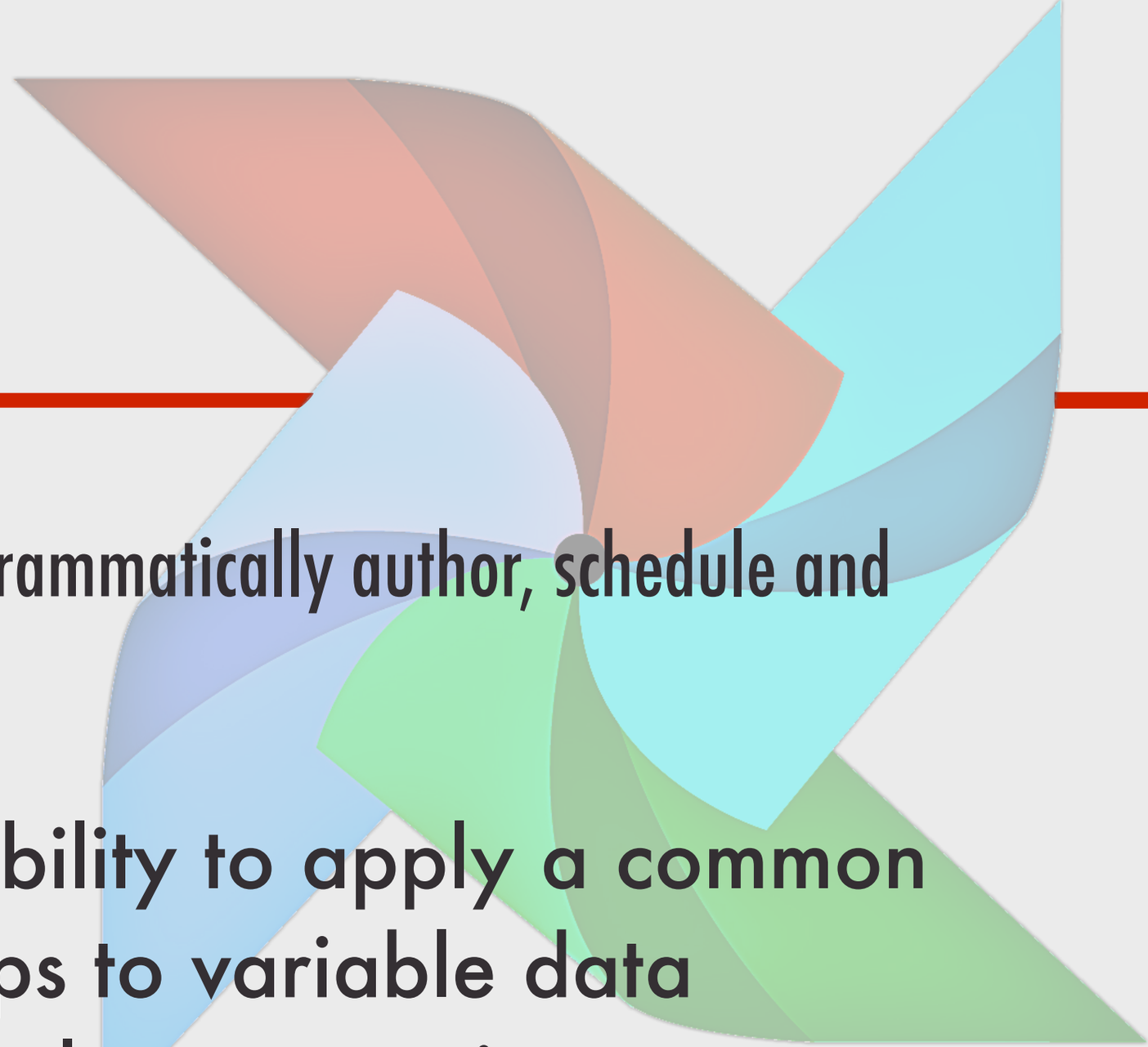


Rolling our own ETL pipeline

Describing complex processing dependencies is hard.



Airflow



"Airflow is a platform to programmatically author, schedule and monitor workflows."

Airflow gives us flexibility to apply a common set of processing steps to variable data inputs, schedule complex processing workflows, and has become a delivery mechanism for our products.

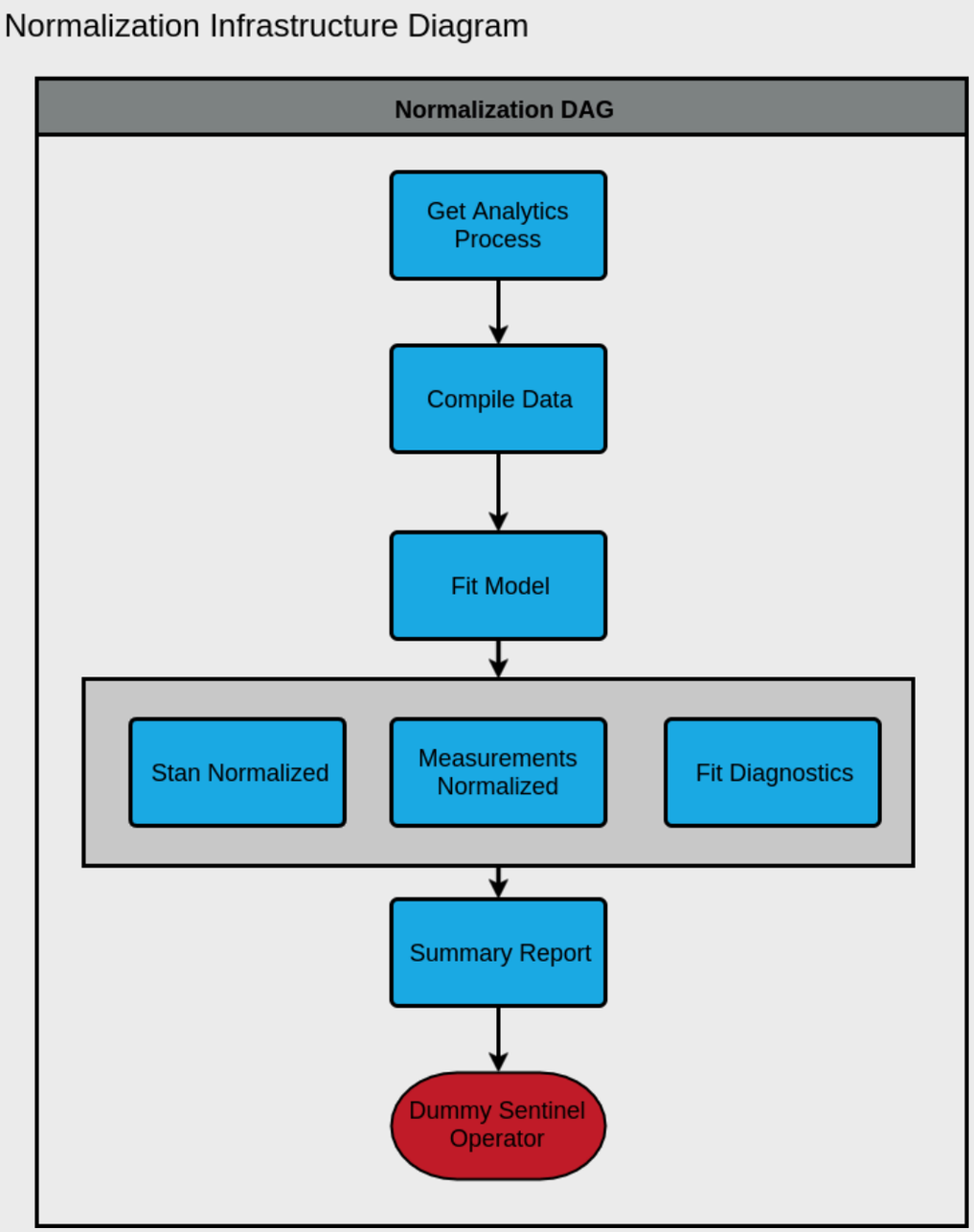
https://airflow.incubator.apache.org/

Structure and Flexibility

e.g. **Normalization**

Airflow workflows are described as **directed acyclic graphs** (DAGs).

Each task node in the DAG is an **operator**.



The anatomy of a DAG

Instantiate DAG →

Custom operators →

Ordering →

```
default_args = {
    'owner': 'airflow',
    'start_date': datetime.datetime(2017, 5, 1),
    'email': EMAIL,
    'email_on_failure': False,
    'email_on_retry': False,
    'catchup': False,
    'concurrency': 1,
    'retries': 0,
}

dag = DAG('normalization', default_args=default_args, schedule_interval=None)

compile_data = norm_ops.CompileData(
    task_id='compile_data',
    analytics_conn_id=ANALYTICS_CONN_ID,
    drawbridge_conn_id=DRAWBRIDGE_CONN_ID,
    dag=dag,
)

fit_data_task = norm_ops.FitModel(
    task_id="fit_model",
    analytic_process_run_ids="{{ ti.xcom_pull(task_ids='compile_data') }}",
    analytics_conn_id=ANALYTICS_CONN_ID,
    drawbridge_conn_id=DRAWBRIDGE_CONN_ID,
    dag=dag,
)

fit_data_task.set_upstream(compile_data)

invalidate_prior_data = norm_ops.InvalidatePriorData(
    task_id='invalidate_prior_data',
    analytic_process_run_ids="{{ ti.xcom_pull(task_ids='fit_model') }}",
    dag=dag,
    drawbridge_conn_id=DRAWBRIDGE_CONN_ID
)

invalidate_prior_data.set_upstream(fit_data_task)
```

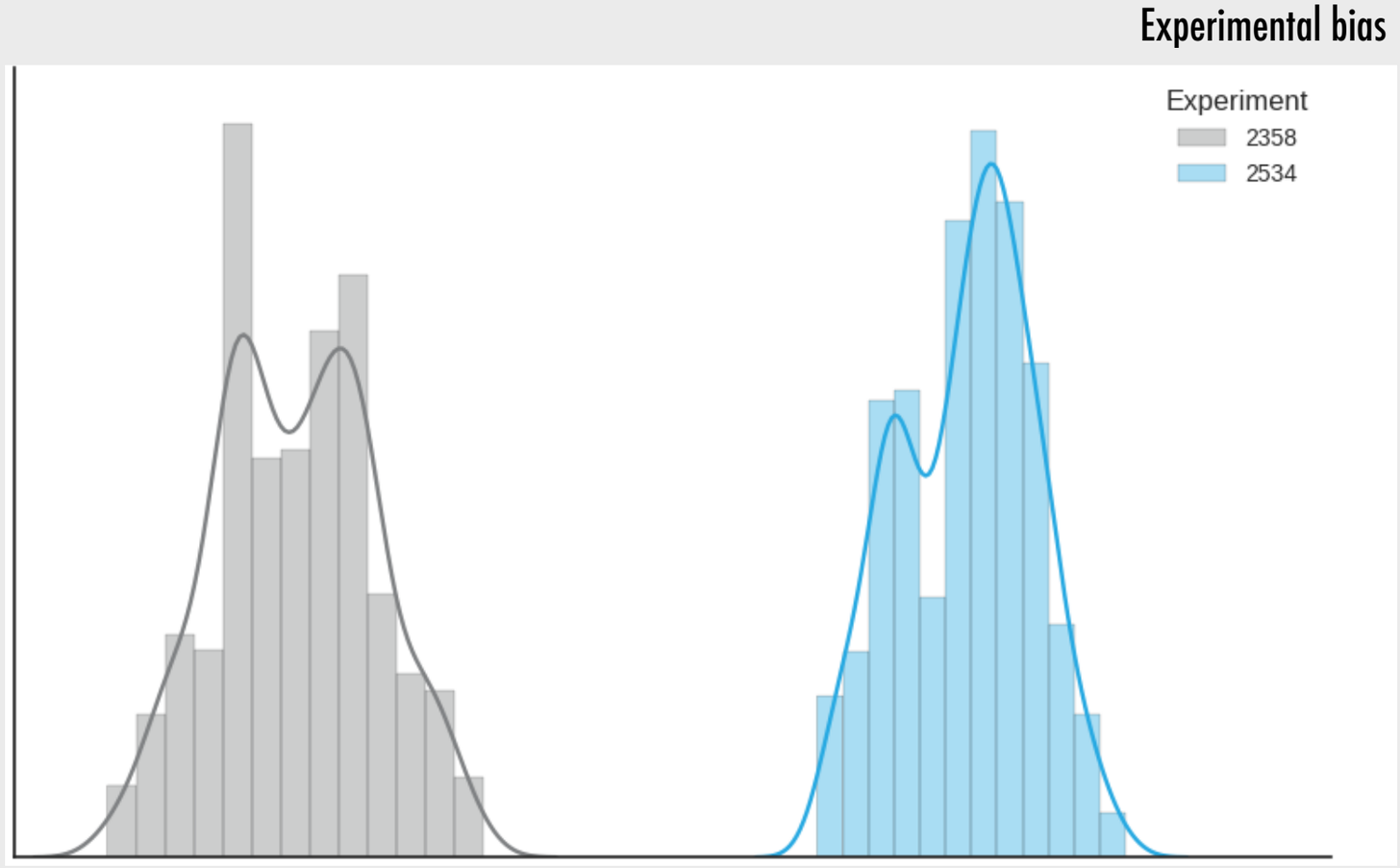
Modularity and flexibility

```
class FitModel(ZFlowBaseOperator):
    """
    An operator to fit the data stored in a dataset with the appropriate model.
    """
```

```
def execute(self, context):

    for analytic_process_run_id in runs:

        analytic_process_run = AnalyticProcessRun.find(analytic_process_run_id)
        analytic_process = AnalyticProcess.find(analytic_process_run.analyticProcessId)

        normalization_model = normalization.get_normalization_config(
            project_name, analytic_process.modelName, analytic_process.modelVersion
        )
```

Airflow + PyStan

With Bayesian hierarchical models we estimate (and monitor) the distribution of batch effects.



DropBox

- Scientists at Zymergen work with data using many different tools including JMP, SQL, and Excel.
- We use a custom DropBox hook to make quick data ingestion pipelines.

Alerting / Communication

3rd-party hooks & operators

slack

HipChat

docker

Dropbox

MySQL™

BASH
THE BOURNE-AGAIN SHELL

slack Operator

Delta Strain Consolidation Recommendations Report

Strain Consolidation Recommendations Report - Delta - 2017-03-05

Status
Success 😃

Execution Date
2017-03-05

Yield R^2
[redacted]

Productivity R^2
[redacted]

Delta Strain Consolidation Recommendations Report

Strain Consolidation Recommendations Report - Delta - 2017-03-15

Status
Failure 💩

Execution Date
2017-03-15

Yield R^2
None

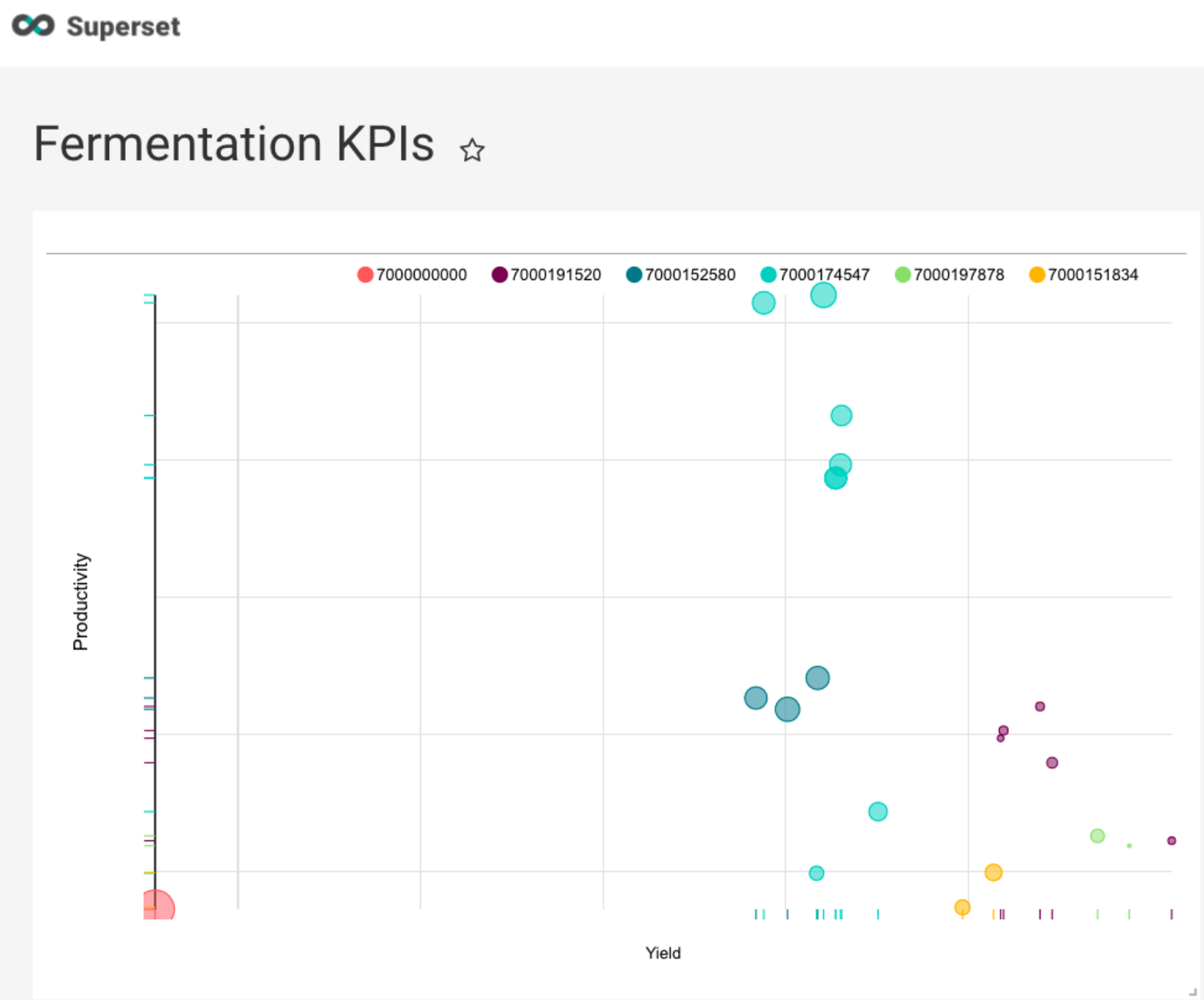
Productivity R^2
None

airflow-bot APP 5:01 PM
Plate LS Job Complete: New experiments for 2017-03-17

Trent Hauck 5:01 PM
I clearly need more emojis.

Michael Flashman 5:27 PM
lol

Pairs well with Superset!



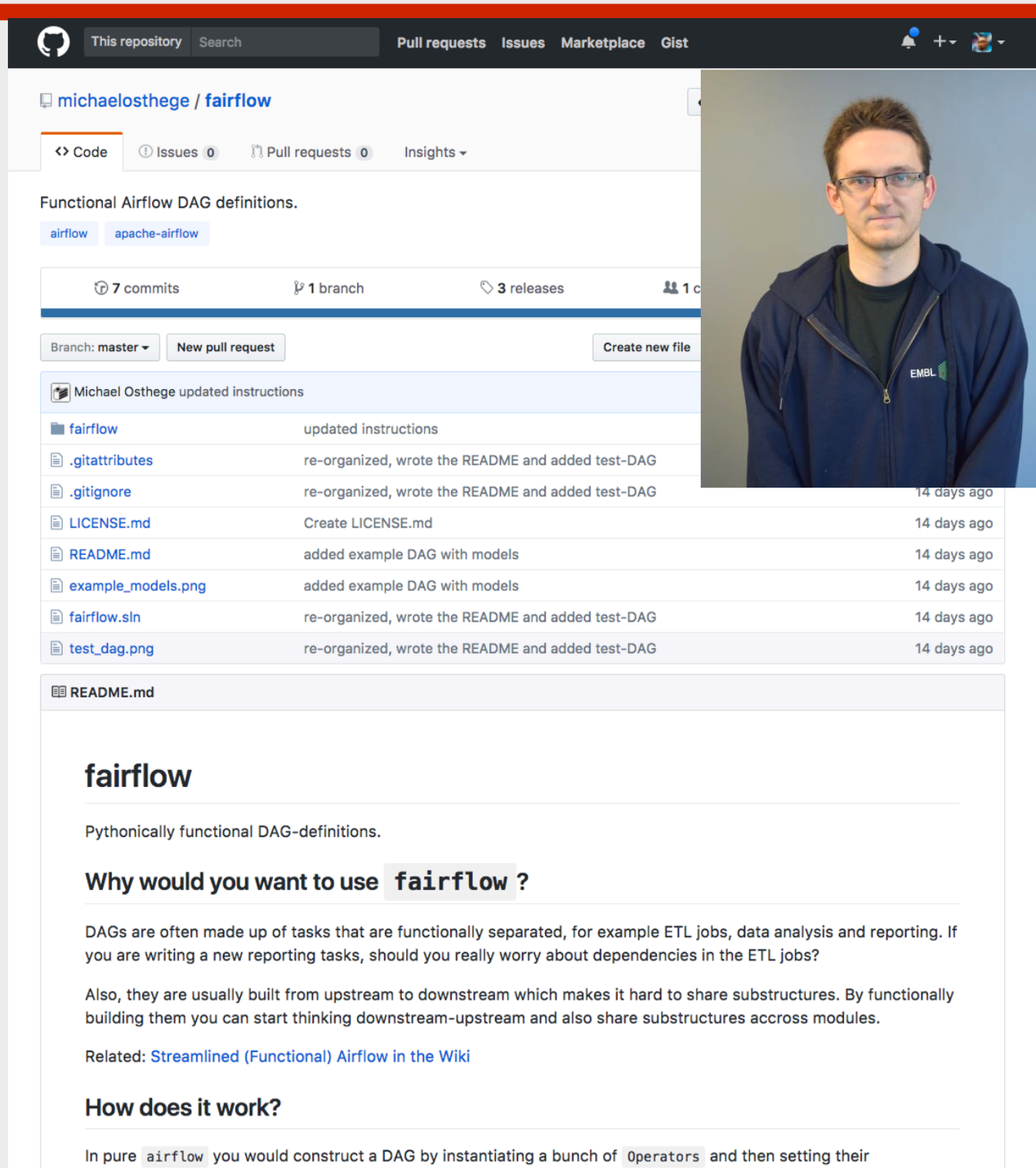
"Apache Superset is a modern, enterprise-ready business intelligence web application"

https://github.com/apache/incubator-superset

Constructing machine learning workflows

Fairflow: Functional Airflow

- The core of Fairflow is an abstract base class *foperator* that takes care of instantiating your Airflow operators and setting their dependencies.
- In Fairflow, DAGs are constructed from f*operators* that create the upstream operators when the final foperator is called.



```
class Compare(fairflow.FOperator):
    """A task that compare the LinearModel with the PolynomialModel. Returns: pandas.DataFrame"""
    def __init__(self, fops_models, id=None):
        self.fops_models = fops_models
        return super().__init__(id)

    @staticmethod
    def compare(**context):
        """Accumulates the results of upstream tasks into a DataFrame"""
        task_ids = fairflow.utils.get_param("model_taskids", context)                    # get the
        comparison = pandas.DataFrame(columns=["modelname", "result"])
        for task_id in task_ids:
            modelresult = context["ti"].xcom_pull(task_id)                               # pull the
            comparison.loc[-1] = task_id, modelresult
        return comparison
```

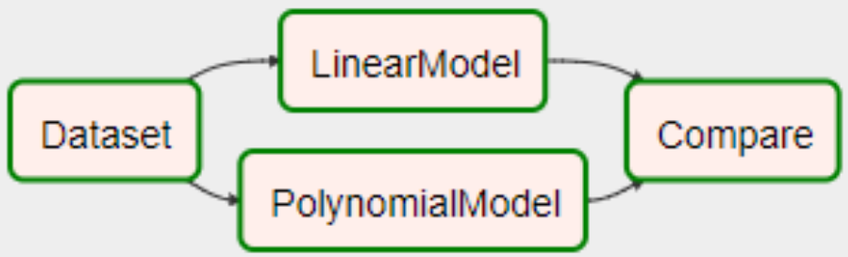
```
    def call(self, dag):
        """Instantiate upstream tasks, this task and set dependencies. Returns: task"""
        model_tasks = [                                    # instantiate tasks for running the different mode
            f(dag)                          # by calling their FOperators on the current `dag`
            for f in self.fops_models   # notice that we do not know about the models upstream dependencie
        ]
        t = python_operator.PythonOperator(
            task_id=self.__class__.__name__,
            python_callable=self.compare,
            provide_context=True,
            templates_dict={
                "model_taskids": [mt.task_id for mt in model_tasks]
            },
            dag=dag
        )
        t.set_upstream(model_tasks)
        return t
```

Defining ML workflows

In the DAG definition, create an instance of the task.

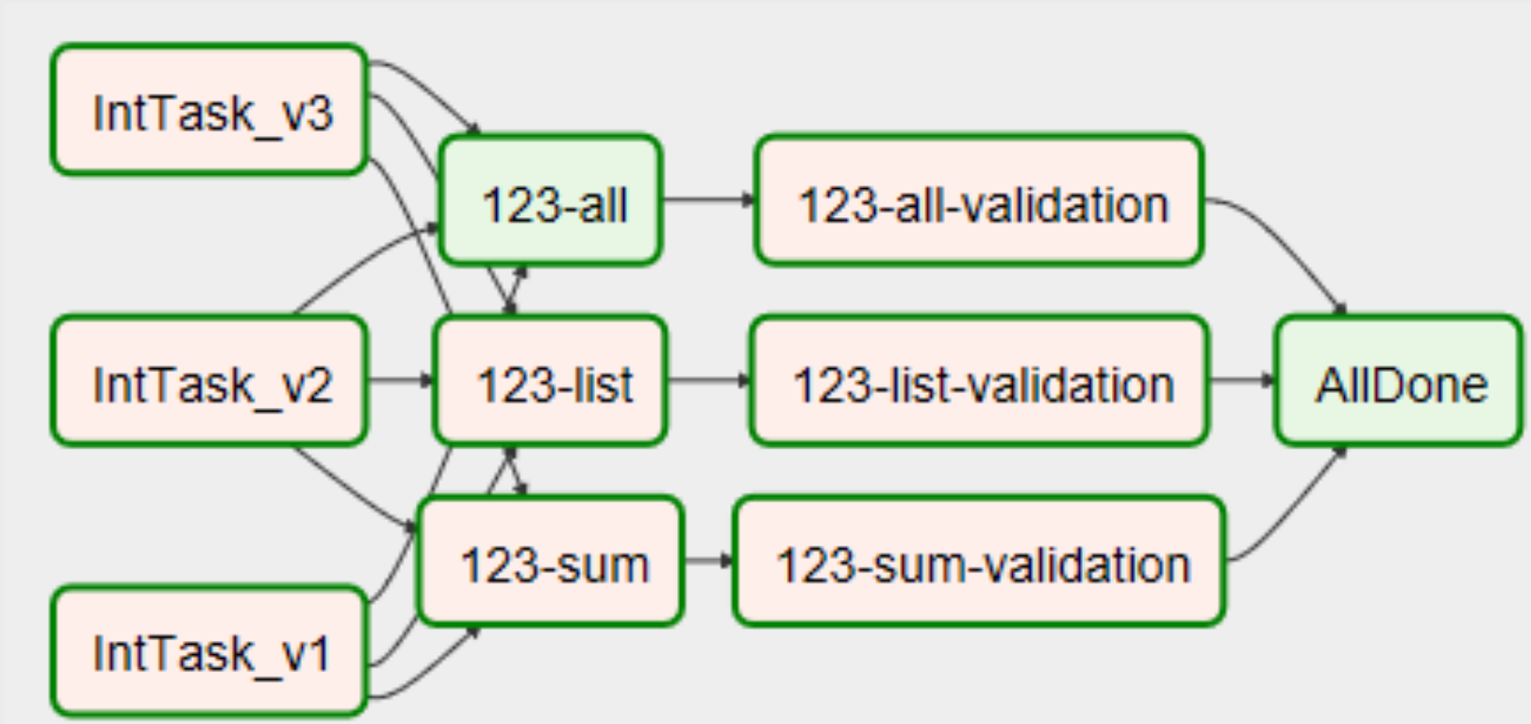
Then, instantiate a DAG like usual and call the compare task on the DAG.

```
f_linear = LinearModel()
f_poly = PolynomialModel(degree=3)
f_compare = Compare([f_linear, f_poly])

dag = airflow.DAG('model-comparison',
    default_args=default_args,
    schedule_interval=None
)

t_compare = f_compare(dag)
```



Defining ML workflows

The design allows for simple creation of complicated experimental workflows with arbitrary sets of models, parameters, and evaluation metrics.



Is Airflow for you?

☑ Do you have heterogeneous data sources?

☑ Do you have complex dependencies between processing tasks?

☑ Do you have data with different velocities?

☑ Do you have constraints on your time?

Probably!

Thanks team!