

Dynamic DAGs – The New Horizon

...

Ash Berlin-Taylor

Who is this Ash character anyway?



PMC member of
Apache Airflow project;
ASF Member

ASTRONOMER

Director of Airflow Engineering,
Astronomer.io



Have your say:
<https://bit.ly/AirflowSurvey22>



"Dynamic" DAGs in the past

Nothing good, all hacky in some way

The Airflow you know, but more powerful

Do what you want, natively

Architecture

How we built it, and its limitations



Hacks on top of hacks; or what we used to do

A dark, dramatic landscape featuring a volcano erupting in the background. Bright orange lava flows are visible, spreading across the foreground. The sky is filled with dark, heavy clouds.

Create n tasks in your DAG

Might underuse slots when cluster idle

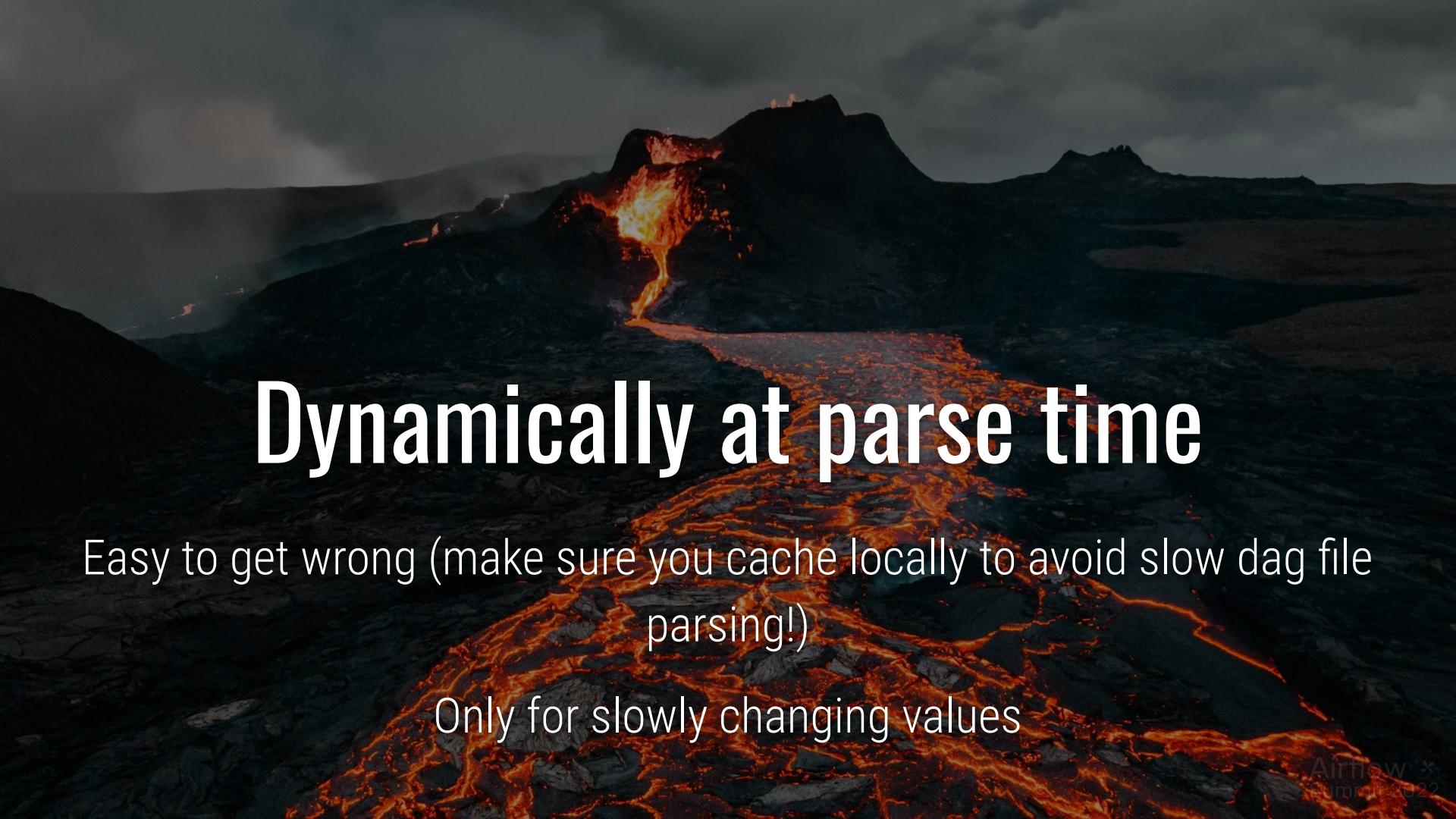
Have to pick an arbitrary "fixed" parallelism

Create n tasks in
your DAG

```
for i in range(NUM_PARALLEL_FILE_LOADERS):
    from_position = partial(get_from_position, i)

download_task = MyDownloadFromS3Operator(
    task_id=f"download_file_{i}",
    xcom_callable=from_position,
```

Create n tasks in
your DAG

A dark, dramatic photograph of a volcano erupting at night or in low light, with lava flows and smoke.

Dynamically at parse time

Easy to get wrong (make sure you cache locally to avoid slow dag file parsing!)

Only for slowly changing values

```

@task()
def update_experiment_list():
    s3_hook = CustomS3Hook(aws_conn_id='s3_default')
    ...
    Variable.set(key=experiment_var_name, value=experiments, serialize_json=True)

update_experiment_list_task = update_experiment_list()

ab_test_model_output_prefix = 'redshift/reporting/ab_testing.modeled_experiment_data'
experiment_name = None
with TaskGroup('run_model') as run_models:
    for experiment_name in Variable.get(key=experiment_var_name, default_var=[], deserialize_json=True):
        run_single_model = AbTestingModelOperator(
            task_id=f'{experiment_name or "none"}',
            output_key=f'{ab_test_model_output_prefix}/{experiment_name}.parquet',
            experiment_name=experiment_name,
        )
    if not experiment_name:
        run_single_model = DummyOperator(task_id='none')

redshift_load = RedshiftCopyOperator(
    database='reporting',
    schema='ab_testing',
    table='modeled_experiment_data',
    column_list=[
        Column('draw', 'BIGINT'),
        Column('variation', 'VARCHAR(255)'),
        Column('value', 'DOUBLE PRECISION'),
        Column('param', 'VARCHAR(255)'),
        Column('experiment', 'VARCHAR(255)'),
    ],
    bucket=s3_buckets.analytics_integration,
    key=f'{ab_test_model_output_prefix}/',
    copy_options=['PARQUET'],
    truncate_table=True,
)
chain_tasks(
    update_experiment_list_task,
    run_models,
    redshift_load,
)

```

"Dynamically" at
parse time

```

@task()
def update_experiment_list():
    s3_hook = CustomS3Hook aws_conn_id='s3_default'
    ...
    Variable.set(key=experiment_var_name, value=experiments, serialize_json=True)

update_experiment_list_task = update_experiment_list()

ab_test_model_output_prefix = 'redshift/reporting/ab_testing.modeled_experiment_data'
experiment_name = None
with TaskGroup('run_model') as run_models:
    for experiment_name in Variable.get(key=experiment_var_name, default=None, deserialize_json=True):
        run_single_model = AbTestingModelOperator(
            task_id=experiment_name or 'none',
            output_key=f'{ab_test_model_output_prefix}/{experiment_name}.parquet',
            experiment_name=experiment_name,
        )
    if not experiment_name:
        run_single_model = DummyOperator(task_id='none')

redshift_load = RedshiftCopyOperator(
    database='reporting',
    schema='ab_testing',
    table='modeled_experiment_data',
    column_list=[
        Column('draw', 'BIGINT'),
        Column('variation', 'VARCHAR(255)'),
        Column('value', 'DOUBLE PRECISION'),
        Column('param', 'VARCHAR(255)'),
        Column('experiment', 'VARCHAR(255)'),
    ],
    bucket=s3_buckets.analytics_integration,
    key=f'{ab_test_model_output_prefix}/',
    copy_options=['PARQUET'],
    truncate_table=True,
)

chain_tasks(
    update_experiment_list_task,
    run_models,
    redshift_load,
)

```

```

for experiment_name in Variable.get(key=experiment_var_name, deserialize_json=True):
    run_single_model = AbTestingModelOperator(
        task_id=experiment_name or 'none',

```

"Dynamically" at
parse time

TriggerDagRunOperator + sensor

Hard to view overall status at a glance

A dark, dramatic landscape featuring a volcano erupting with bright orange lava flowing down its slopes under a cloudy sky.

Just live with it being in one task

Slow

Not practical for high cardinality

A dark, dramatic landscape featuring a volcano erupting in the background, with bright orange and red lava flows cascading down its slopes. In the foreground, a massive, sprawling lava field covers the ground, its surface glowing with intense heat. The sky is filled with dark, heavy clouds, creating a somber and powerful atmosphere.

Parallelism in external systems

(eg Spark) - can be overkill

\$\$\$

A dark, atmospheric photograph of a living room. In the center is a small, round wooden coffee table. Behind it are two large, plush blue armchairs. To the right, there's a dark sofa. In the background, a fireplace is visible, with a bright orange glow emanating from its opening. The overall lighting is low, creating a cozy and intimate atmosphere.

Your DAGs can now dynamically size
themselves to fit your data!

Apache Airflow 2.2 and beyond

Airflow X
Summit 2021

Roadmap: A possible future



tor.partial(

```
    bucket = my_bucket  
    ) .map(key=my_files)
```

```
data = ingest.map(markets)  
rois = calculate_roi.map(market, data)  
stats = aggregate_rois(market, rois)
```

Airflow Summit 2021

Airflow X
Summit 2022

Apache Airflow



```
@task  
def get_files_from_s3():  
    return [...]  
  
my_files = get_files_from_s3()  
s3_delete_files = MyFileProcessOperator.partial(  
    aws_conn_id="my-aws-conn-id",  
    bucket="my-bucket"  
).map(key=my_files)
```

```
data = ingest.map(markets)  
rois = calculate_roi.map(market, data)  
stats = aggregate_rois(market, rois)
```

Airflow Summit 2021

```
Operator.expand(arg_name=iterable, ...)
```

```
Operator.partial(fixed_arg=...)  
    .expand(arg_name=iterable, ...)
```

A dark, atmospheric forest scene. Sunlight filters through tall, thin trees, creating bright rays and a path of light on a dark ground surface. Two small figures are visible in the distance along this lighted path.

CSV files \Rightarrow line-delimited JSON
files


```
@task
def get_inputs(input_bucket, prefix):
    return S3Hook().list_keys(bucket_name=input_bucket, prefix=prefix)

@task
def csv_to_json(input_bucket, key, output_bucket):
    hook = S3Hook()
    output = io.BytesIO()

    csv_data = hook.read_key(key, input_bucket)
    reader = csv.DictReader(io.StringIO(csv_data))

    for row in reader:
        output.write(json.dumps(row, indent=None).encode('utf-8') + b'\n')

    output.seek(0)
    hook.load_file_obj(output, key=key, bucket_name=output_bucket)

files = get_inputs(input_bucket=input_bucket, prefix="data_provider_a/{{ data_interval_end | ds }}/")
csv_to_json.partial(input_bucket=input_bucket, output_bucket="airflow-summit-2022-processed") \
    .expand(key=files)
```

```
@task
def get_inputs(input_bucket, prefix):
    return S3Hook().list_keys(bucket_name=input_bucket, prefix=prefix)

@task
def csv_to_json(input_bucket, key, output_bucket):
    hook = S3Hook()
    output = io.BytesIO()

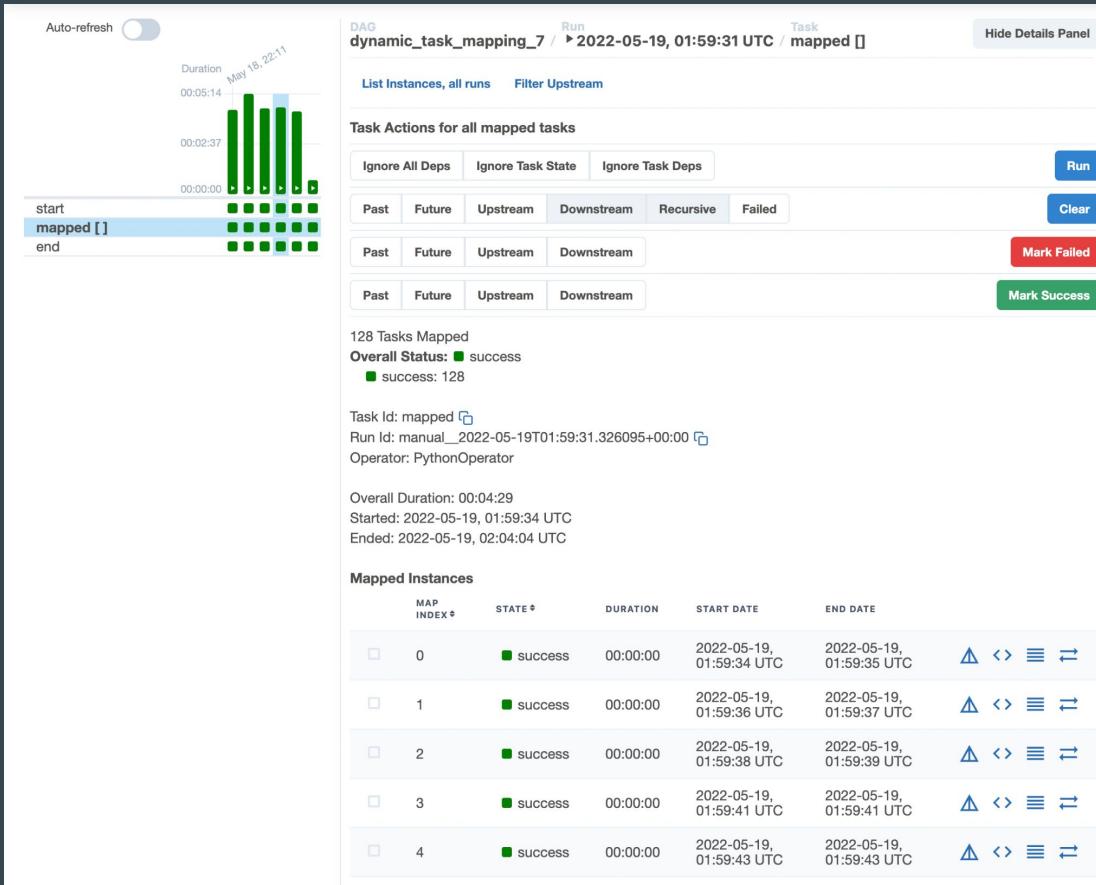
    csv_data = hook.read_key(key, input_bucket)
    reader = csv.DictReader(io.StringIO(csv_data))

    for row in reader:
        output.write(json.dumps(row, indent=None).encode('utf-8') + b'\n')

    output.seek(0)
    hook.load_file_obj(output, key=key, bucket_name=output_bucket)

files = get_inputs(input_bucket=input_bucket, prefix="data_provider_a/{{ data_interval_end | ds }}/")
csv_to_json.partial(input_bucket=input_bucket, output_bucket="airflow-summit-2022-processed") \
    .expand(key=files)
```

**Airflow 2.3 is vastly more
powerful and expressive
than Airflow 1.x**

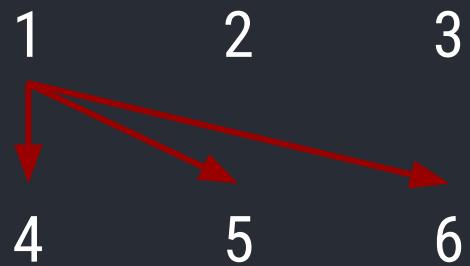


@bbovenzi's amazing new UI

The background is a dark, atmospheric landscape featuring two large, rugged mountains with rocky and mossy slopes. In the center foreground, a small, silhouetted figure stands on a path, illuminated from behind by a bright light source, casting a long shadow. The overall mood is mysterious and vast.

Toy example, real effect:
cross product

```
with DAG(dag_id='toy', start_date=datetime(2022, 5, 23)) as dag:  
    @task  
    def a():  
        return [1, 2, 3]  
  
    @task  
    def b():  
        return [4, 5, 6]  
  
    @task  
    def sum_it(vals):  
        return sum(vals)  
  
    @task  
    def multiply(a, b):  
        return a * b  
  
    combinations = multiply.expand(a=a(), b=b())  
    total = sum_it(combinations)
```



($a=1, b=4$) ($a=1, b=5$) ($a=1, b=6$)

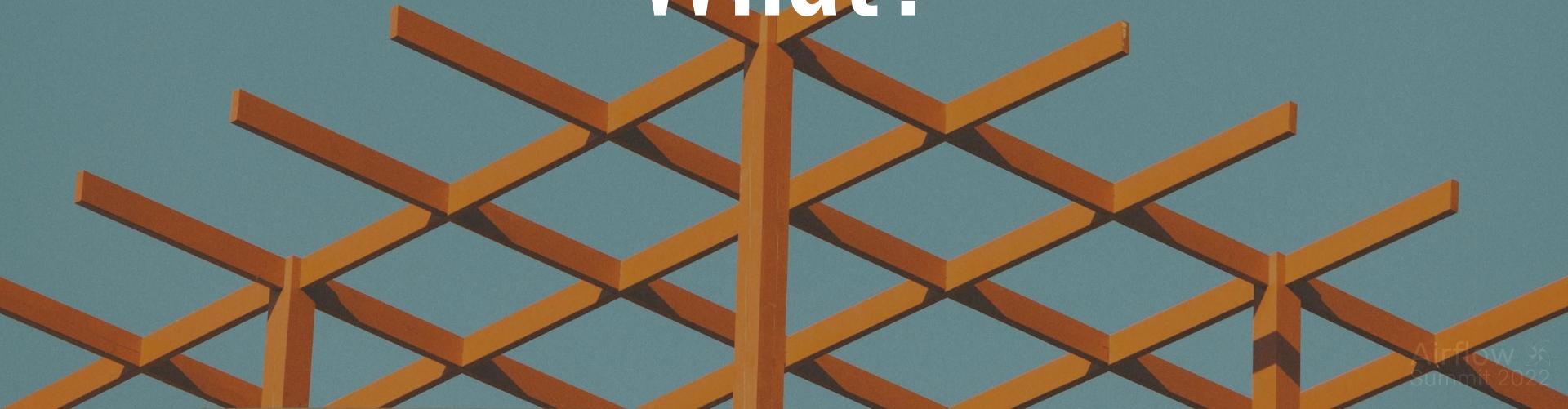


($a=1, b=4$) ($a=1, b=5$) ($a=1, b=6$) ($a=2 b=4$) ($a=2 b=5$) ($a=2 b=6$) ($a=3 b=4$) ...

A large commercial airplane is shown from a rear three-quarter perspective, flying towards the viewer. The aircraft is silhouetted against a vibrant sunset or sunrise sky, which transitions from deep orange at the horizon to a darker blue-grey at the top. The sun is partially visible on the left side of the frame, casting a warm glow. The airplane's tail, engines, and wings are clearly visible against the bright background.

What? Where?
When? How?

What?



List or a dict

Appearing directly in the dag file (or not as the result of a Task)

Result of a TaskFlow operator

"Generating"/Upstream Task must return a list or a dict, else it will fail

XComArg object created for "Classic" operator

xComArg(files) – yes, this is a bit clunky

```
@task  
  
def cmds():  
  
    return ["cmd1", "cmd2"]  
  
  
BashOperator.expand(bash_command=["echo", cmd1()]) # Wrong
```

```
@task  
  
def cmd1():  
  
    return [[{"cmd": "echo", "value": "cmd1"}, {"cmd": "echo", "value": "cmd2"}]]
```

```
BashOperator.expand(bash_command=cmd1()) # Only at top level
```

```
@task  
def cmds():  
    return ["cmd1", "cmd2"]
```

```
BashOperator.expand(bash_command=cmds().map(  
    lambda v: ["echo", v])  
)
```

Sneak peak - 2.4?



Where?

Usually in the Task Runner

Mapped TaskInstances are created in "mini scheduler" as upstreams finish

But fallback/"last resort" in Scheduler

"Static" expansions are expanded at DagRun creation

i.e. lists/dicts

The background of the slide is a photograph of a dense, misty forest. The scene is filled with tall, thin trees, their trunks partially obscured by thick vines and lush green foliage. The atmosphere is hazy, with light filtering through the canopy in a way that creates a sense of depth and mystery. The overall color palette is dominated by shades of green and grey, reflecting the natural environment.

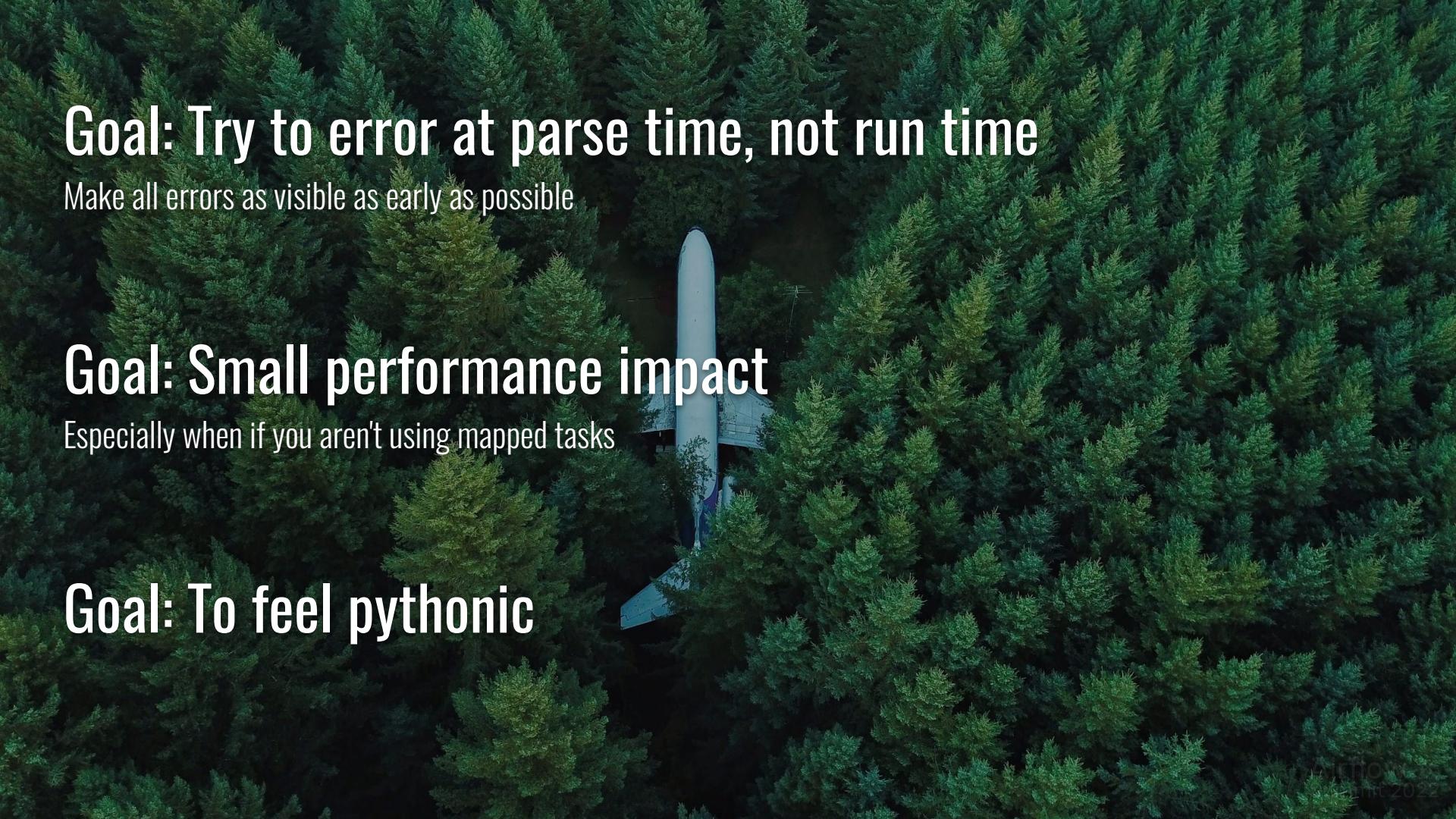
When?

"Just in time"

Mapped TaskInstances are created as upstreams finish –
in worker (or scheduler)

An aerial photograph showing the upper portion of a Boeing 747 aircraft fuselage resting on a bed of tall, dark green coniferous trees. The plane's white body and blue tail are visible against the dark foliage.

How?

The background of the slide is a photograph of a dense forest from an aerial perspective. A small white airplane is visible, positioned centrally, appearing as a white shape against the dark green foliage.

Goal: Try to error at parse time, not run time

Make all errors as visible as early as possible

Goal: Small performance impact

Especially when if you aren't using mapped tasks

Goal: To feel pythonic

TaskMap table

Scheduler shouldn't pull (possibly large!) XCom rows, so pre-compute what we need

Add map_index to TaskInstance primary key

And all related tables (Fail, Reschedule etc). -1 = not mapped

(Configurable) limit on max number of expansions

Mostly just to prevent mistakenly creating millions of mapped Tasks

+16272 -6127

85 PRs from 7 authors (@ashb, @bbovenzi, @dstandish, @ephraimbuddy, @norm,
@tanelk, @uranusjr)

Special thanks to @MatrixManAtYrService for breaking it so often



Still more to add...

Airflow 2.4 and beyond

- "zip" (rather than cross-product) [#23803](#)
 - Expand TaskGroups (postponed in 2.3)
 - New expansion sources (Variables, DagParams)
 - Multiple args from a single source (**kwargs style)
-

We're hiring (of course)
<https://www.astronomer.io/careers>

ASTRONOMER