dagster

Dagster Deep Dives: Thinking in Partitions

Tim Castillo

Developer Advocate - Dagster Labs

dagster

Dagster is a framework for orchestrating data pipelines

You have the flexibility to build pipelines outright or create a platform to enable others to build their own

As a framework, Dagster is grounded in having strong core building blocks

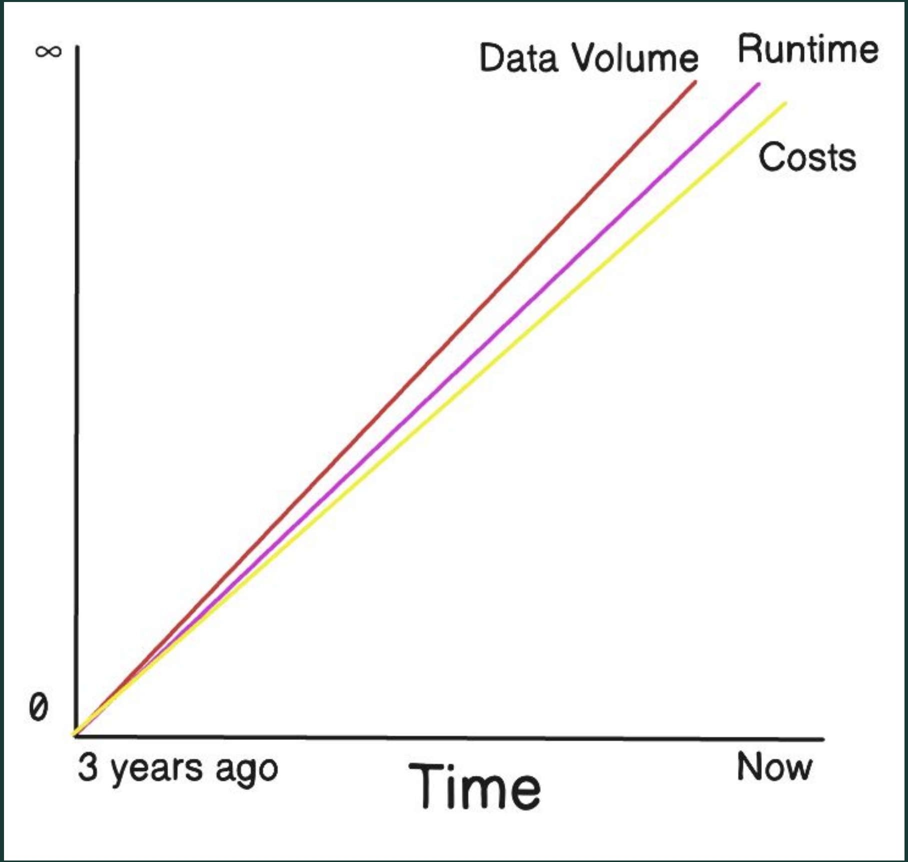
dagster

Today, we’ll go into detail on part of that core:
Partitions

We’ll cover the fundamentals of them, along with how to think of your data pipelines in partitions

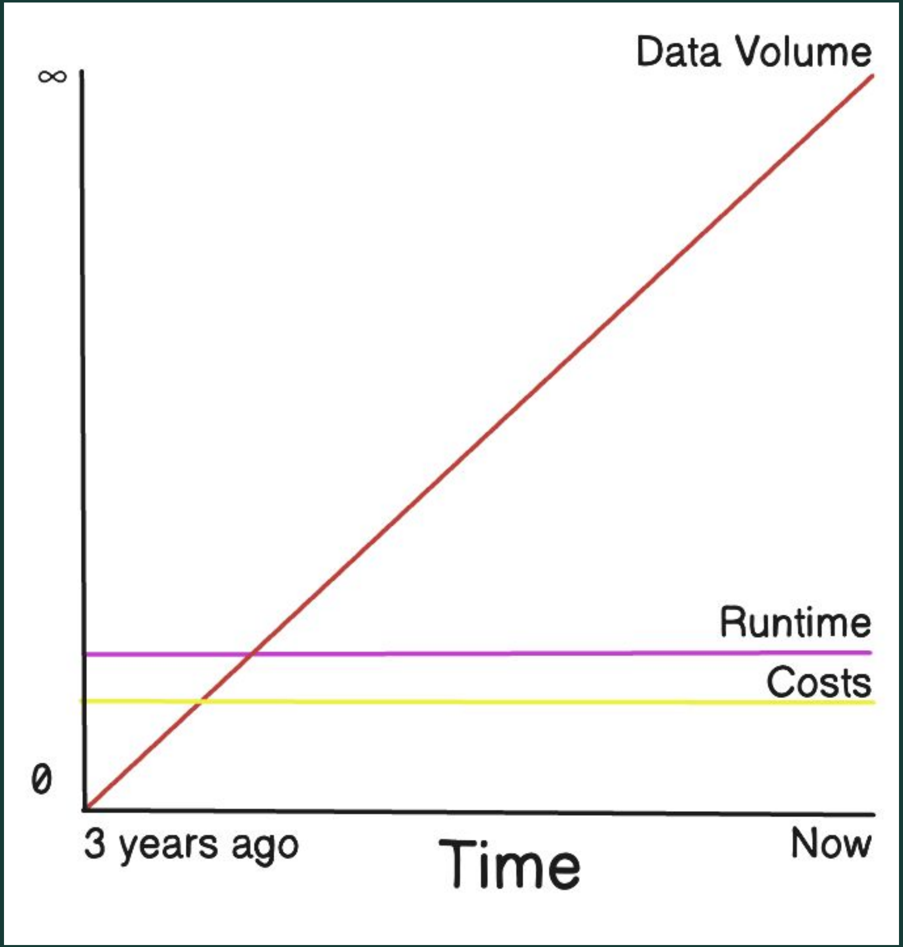
dagster

Data pipelines often scale linearly with the size your data, which includes increasing runtime and costs



dagster

How can you break the relationship between your growing data volume and the resources your pipelines take?



dagster

People have been trying to solve this problem for decades.

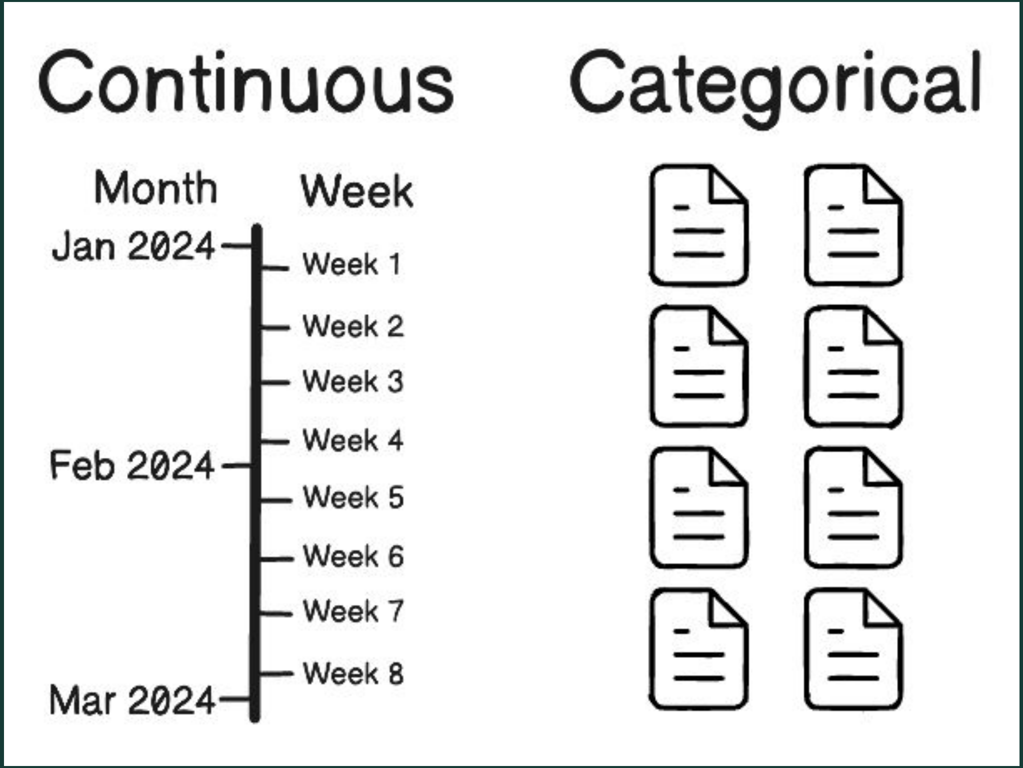
One of the best solutions is to **incrementally** load the data.

Working with data **incrementally** means only working with data that has not be loaded yet

dagster

Partitions are a way to model incremental data

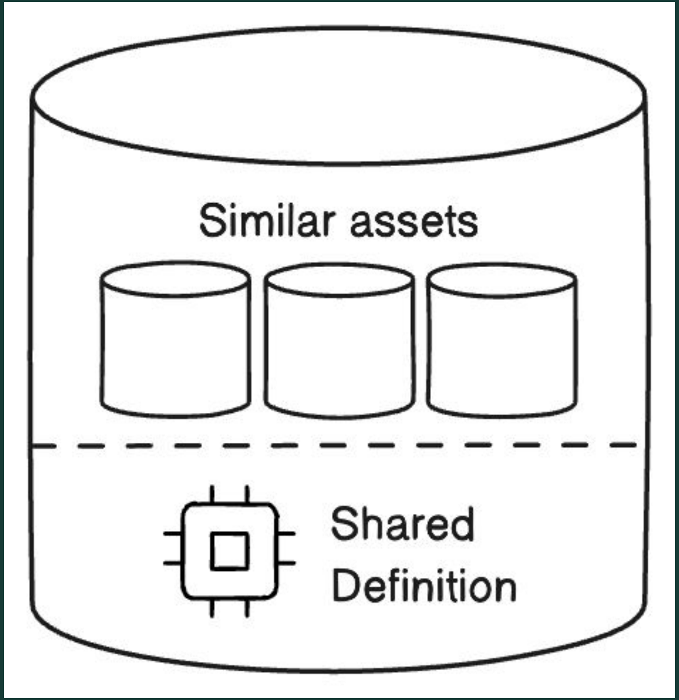
Data is partitioned by a shared dimension



dagster

Dagster has native support for **partitions** as one way to maintain incremental data assets

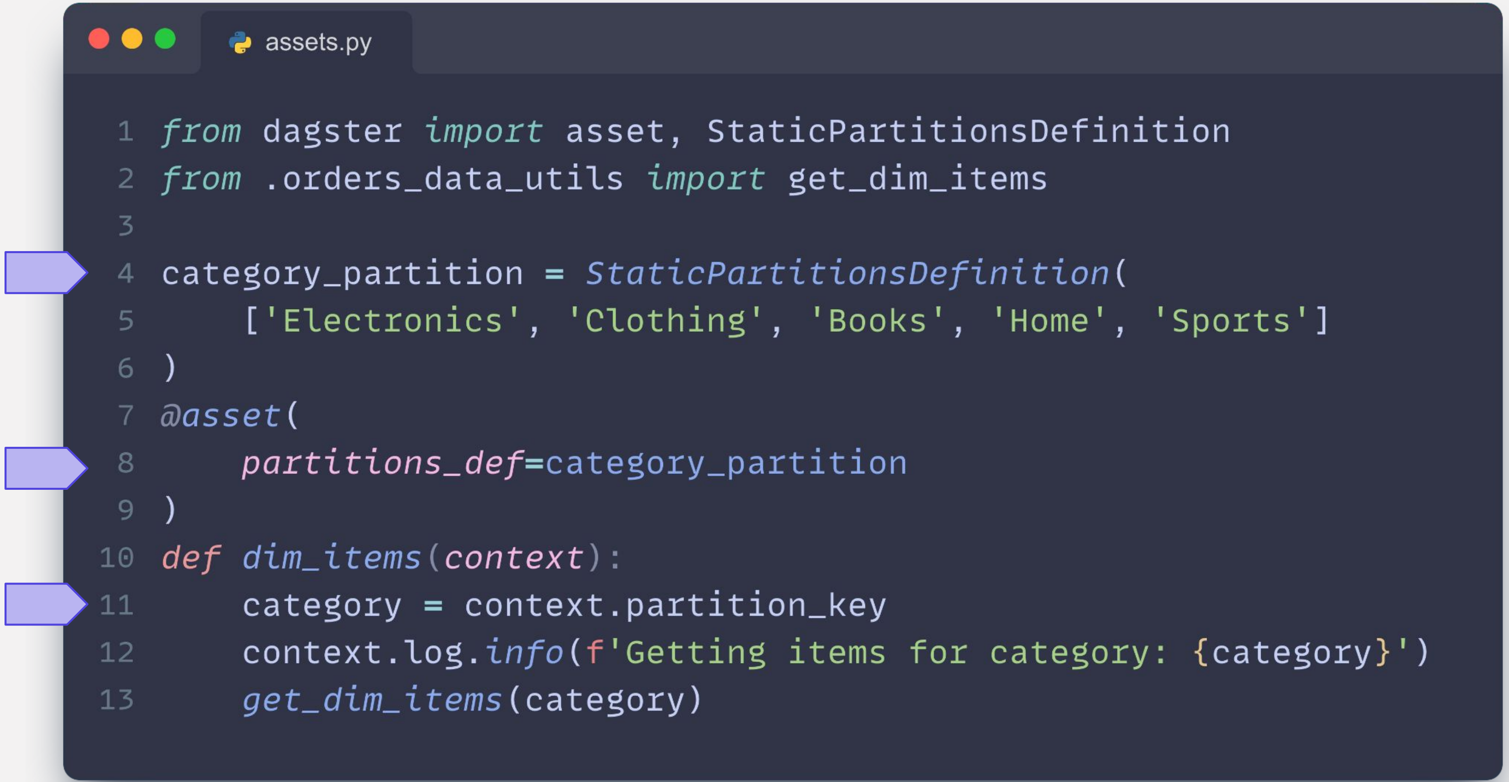
dagster



A partitioned asset is a collection of smaller assets that share the same definition (code/deps)

Dagster’s APIs for Partitions

- Categorical Partitions
  - `StaticPartitionsDefinition`
- Time-based Partitions
  - `[Daily][Weekly][Monthly]PartitionsDefinition`
  - `TimeWindowPartitionDefinition(cron_schedule=”0 0 1 1 *”)`
- Partition Mappings
  - `AssetDep`
  - `AllPartitionMapping`
  - `TimeWindowPartitionMapping`
- Accessors
  - `context.partition_key`
  - `context.partition_time_window`



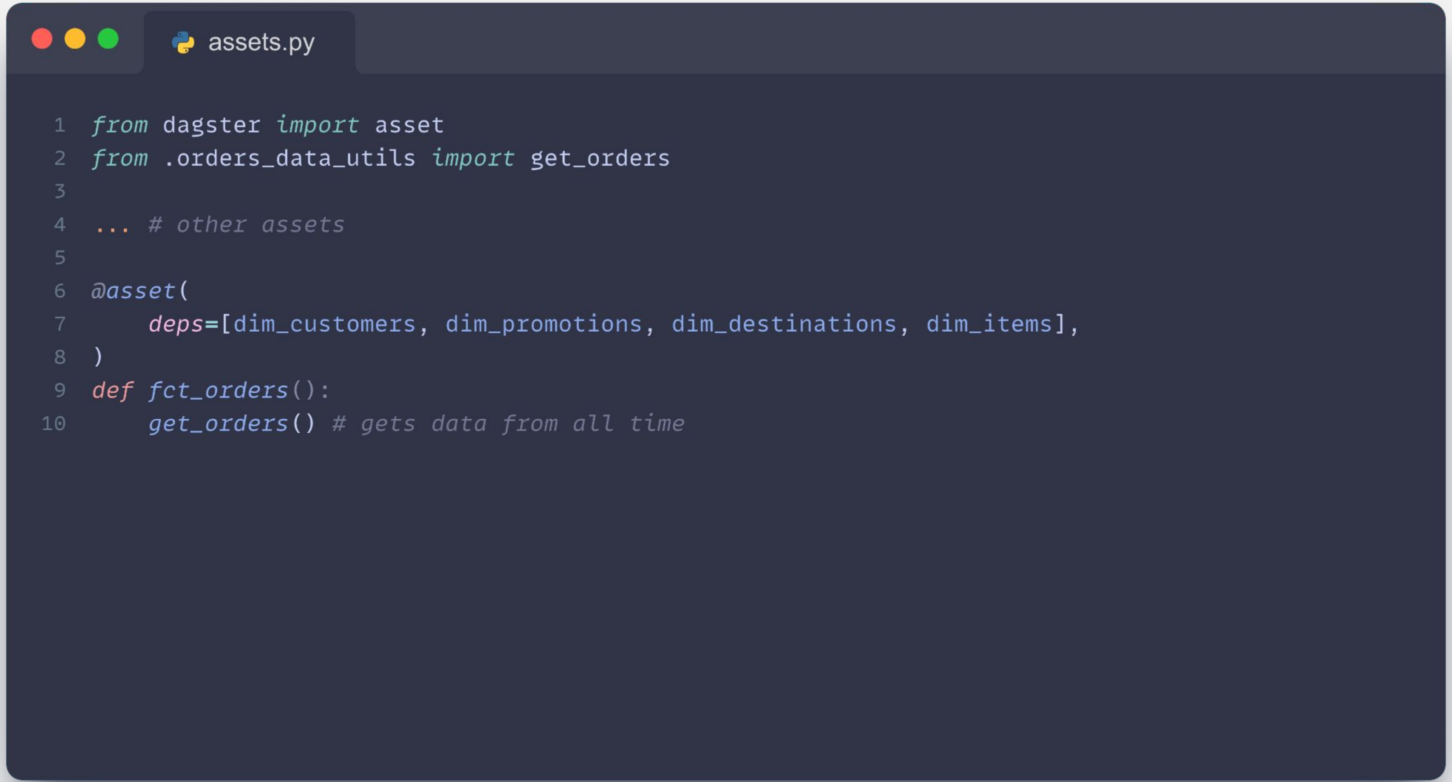
```python
from dagster import asset, StaticPartitionsDefinition
from .orders_data_utils import get_dim_items

category_partition = StaticPartitionsDefinition(
    ['Electronics', 'Clothing', 'Books', 'Home', 'Sports']
)
@asset(
    partitions_def=category_partition
)
def dim_items(context):
    category = context.partition_key
    context.log.info(f'Getting items for category: {category}')
    get_dim_items(category)
```

Demo

dagster

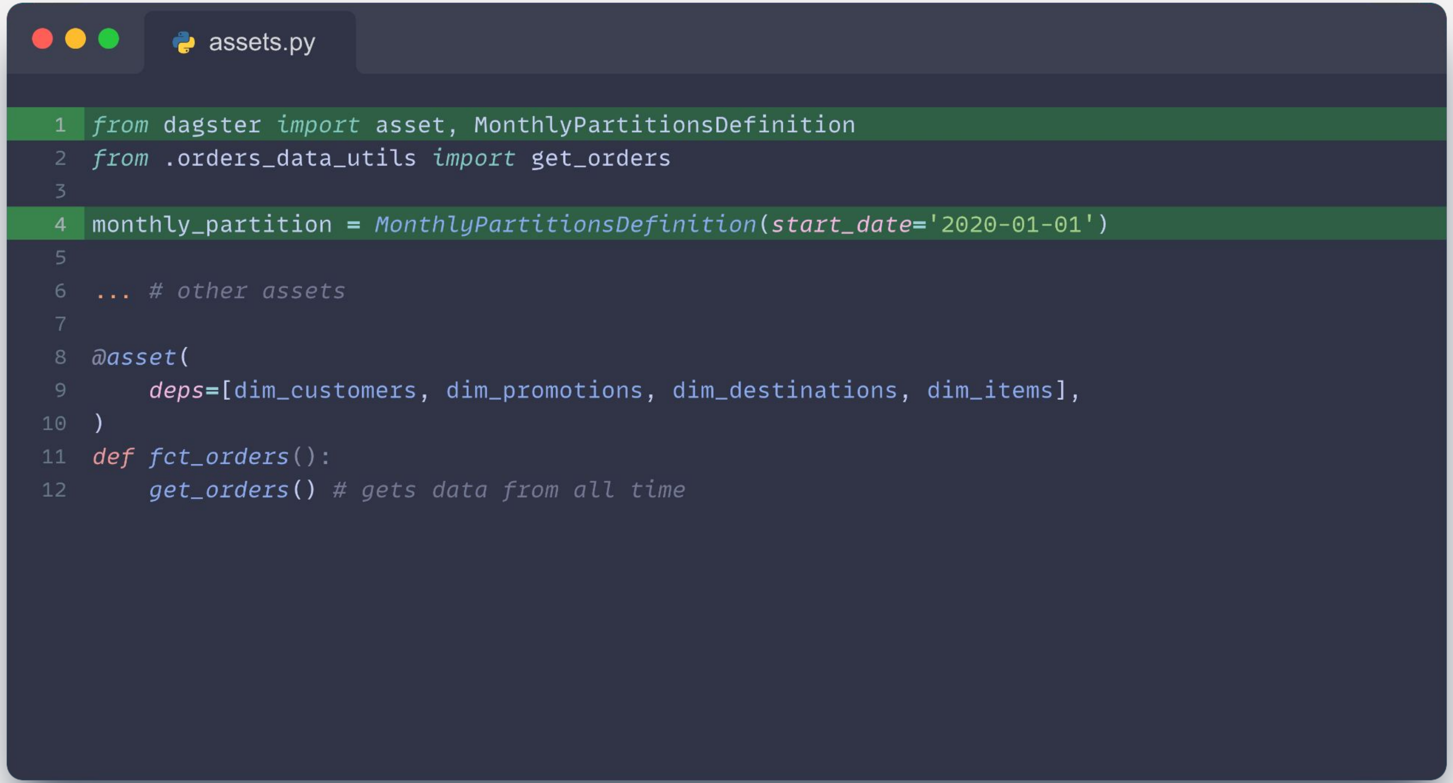
Let’s partition an asset by time

An expensive asset



```python
from dagster import asset
from .orders_data_utils import get_orders

... # other assets

@asset(
    deps=[dim_customers, dim_promotions, dim_destinations, dim_items],
)
def fct_orders():
    get_orders() # gets data from all time
```

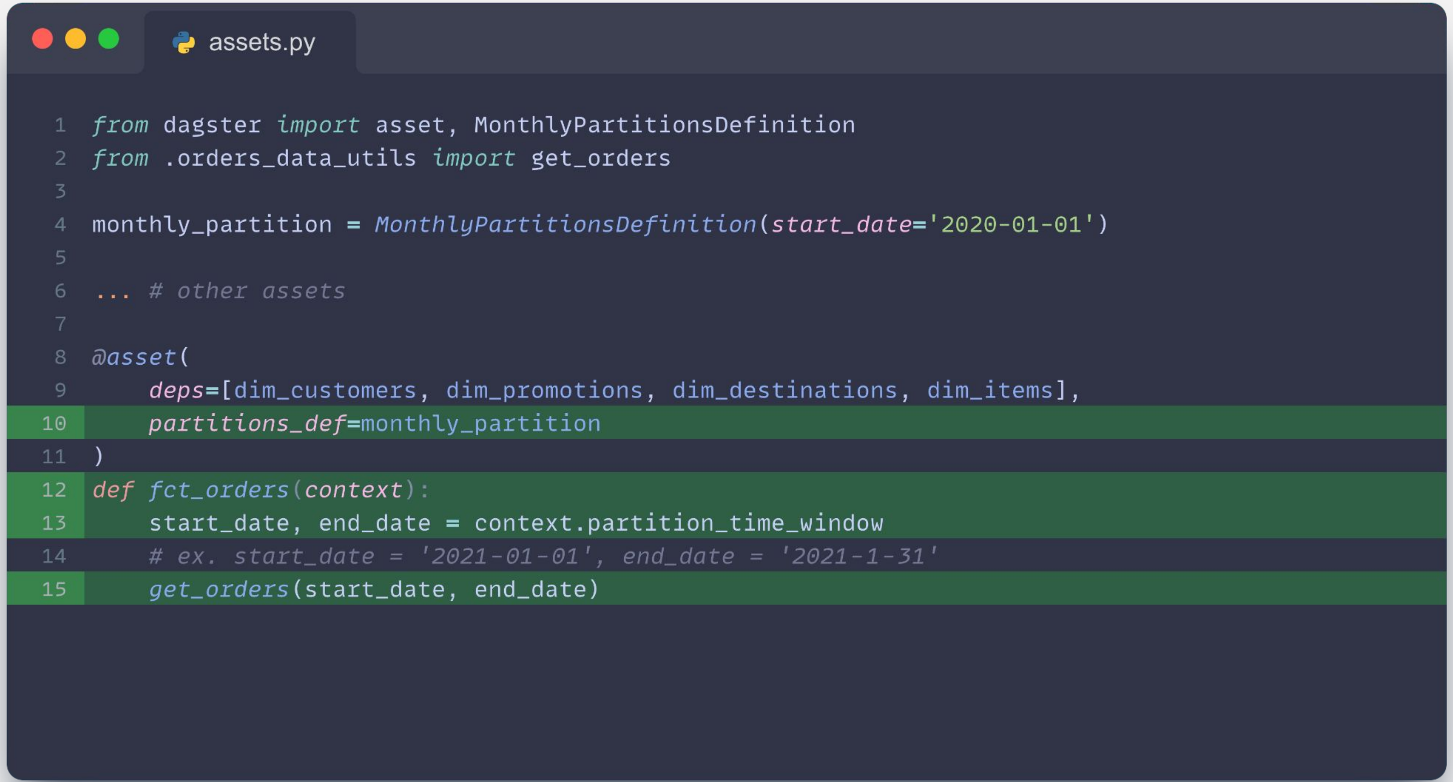
Define a partition



assets.py

```python
from dagster import asset, MonthlyPartitionsDefinition
from .orders_data_utils import get_orders

monthly_partition = MonthlyPartitionsDefinition(start_date='2020-01-01')

... # other assets

@asset(
    deps=[dim_customers, dim_promotions, dim_destinations, dim_items],
)
def fct_orders():
    get_orders() # gets data from all time
```

After


assets.py

```python
from dagster import asset, MonthlyPartitionsDefinition
from .orders_data_utils import get_orders

monthly_partition = MonthlyPartitionsDefinition(start_date='2020-01-01')

... # other assets

@asset(
    deps=[dim_customers, dim_promotions, dim_destinations, dim_items],
    partitions_def=monthly_partition
)
def fct_orders(context):
    start_date, end_date = context.partition_time_window
    # ex. start_date = '2021-01-01', end_date = '2021-1-31'
    get_orders(start_date, end_date)
```

Run It

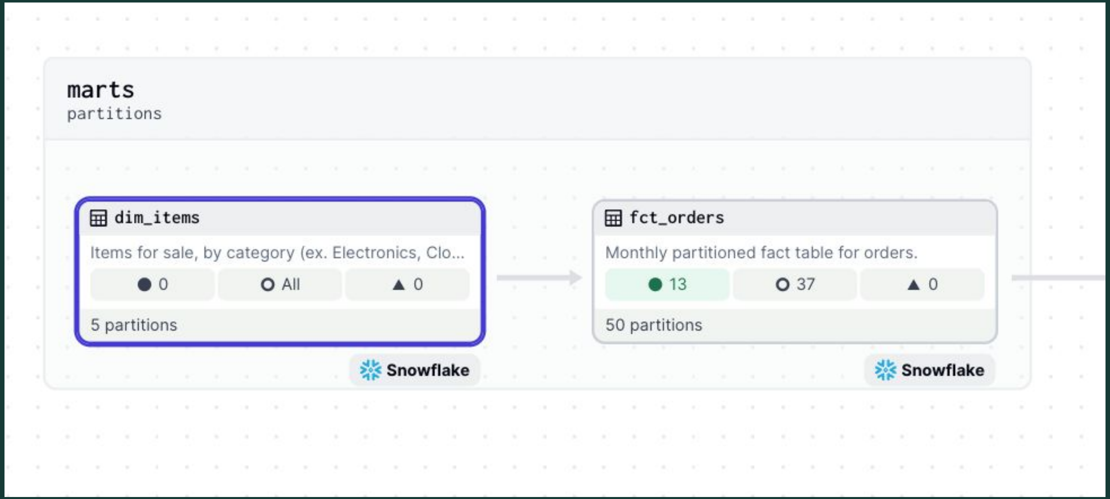
dagster

A **partitioned asset** is a collection of smaller assets that share the same definition

StaticPartitionDefinition can be used for categorical data

MonthlyPartitionDefinition can be used for continuous time data

dagster


marts
partitions
dim_items
Items for sale, by category (ex. Electronics, Clo...
● 0
○ All
▲ 0
5 partitions
Snowflake
fct_orders
Monthly partitioned fact table for orders.
● 13
○ 37
▲ 0
50 partitions
Snowflake

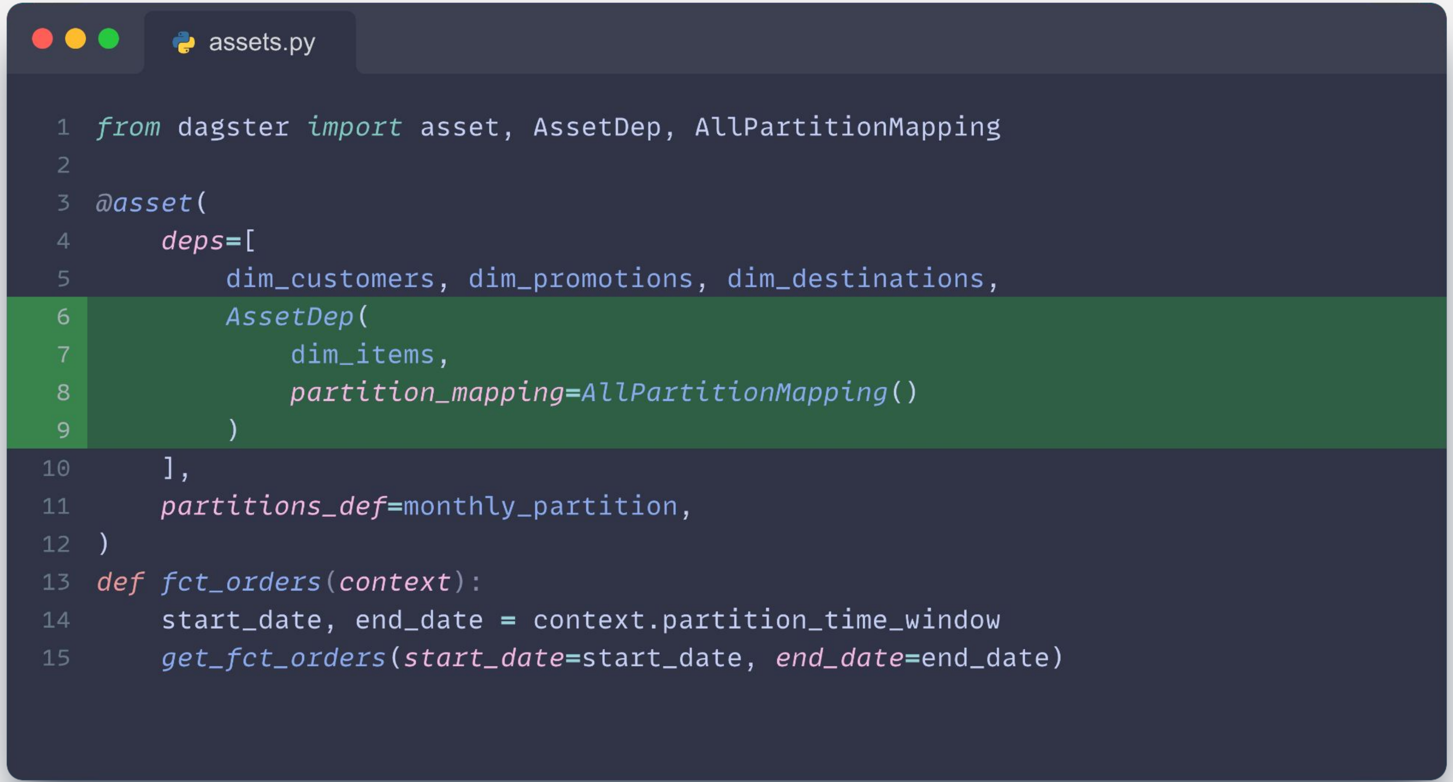
dagster

Partition Mappings configure how assets depend on partitioned assets

This lets you express dependencies like:

- Depend on all partitions
- Depend on the same partition as itself
- Depend on a specific set of partitions
- etc.

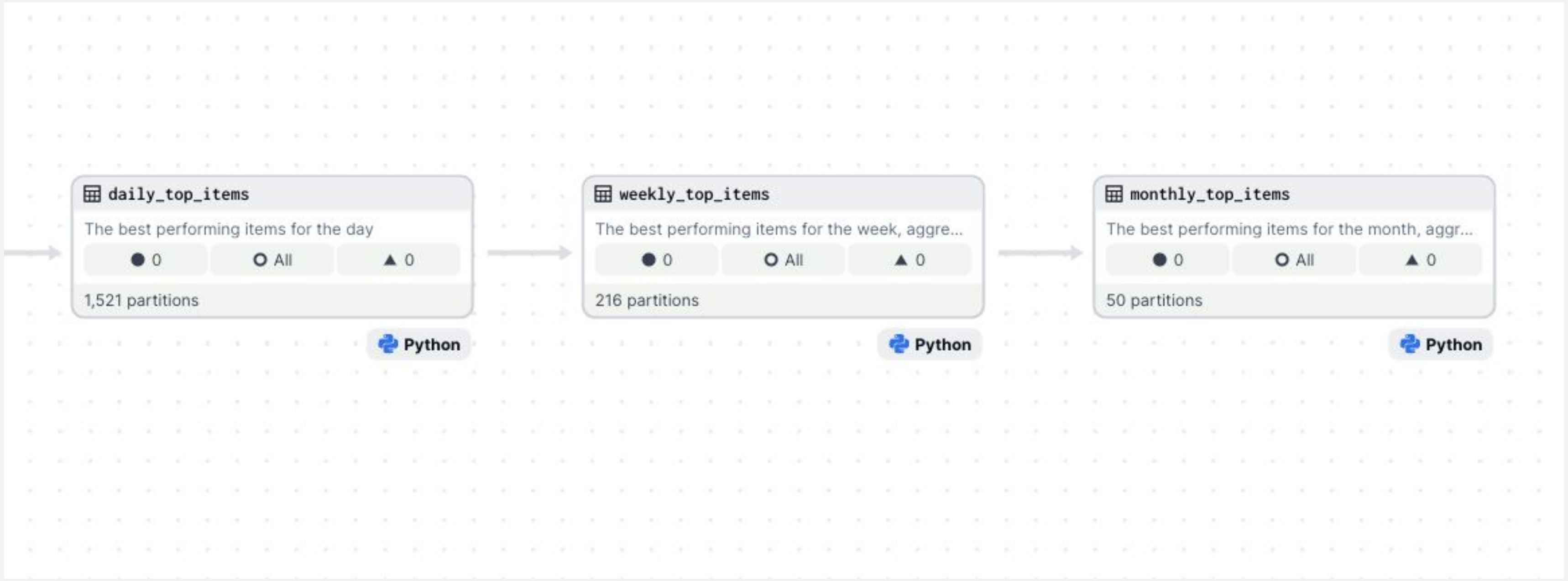
Configure a dependency



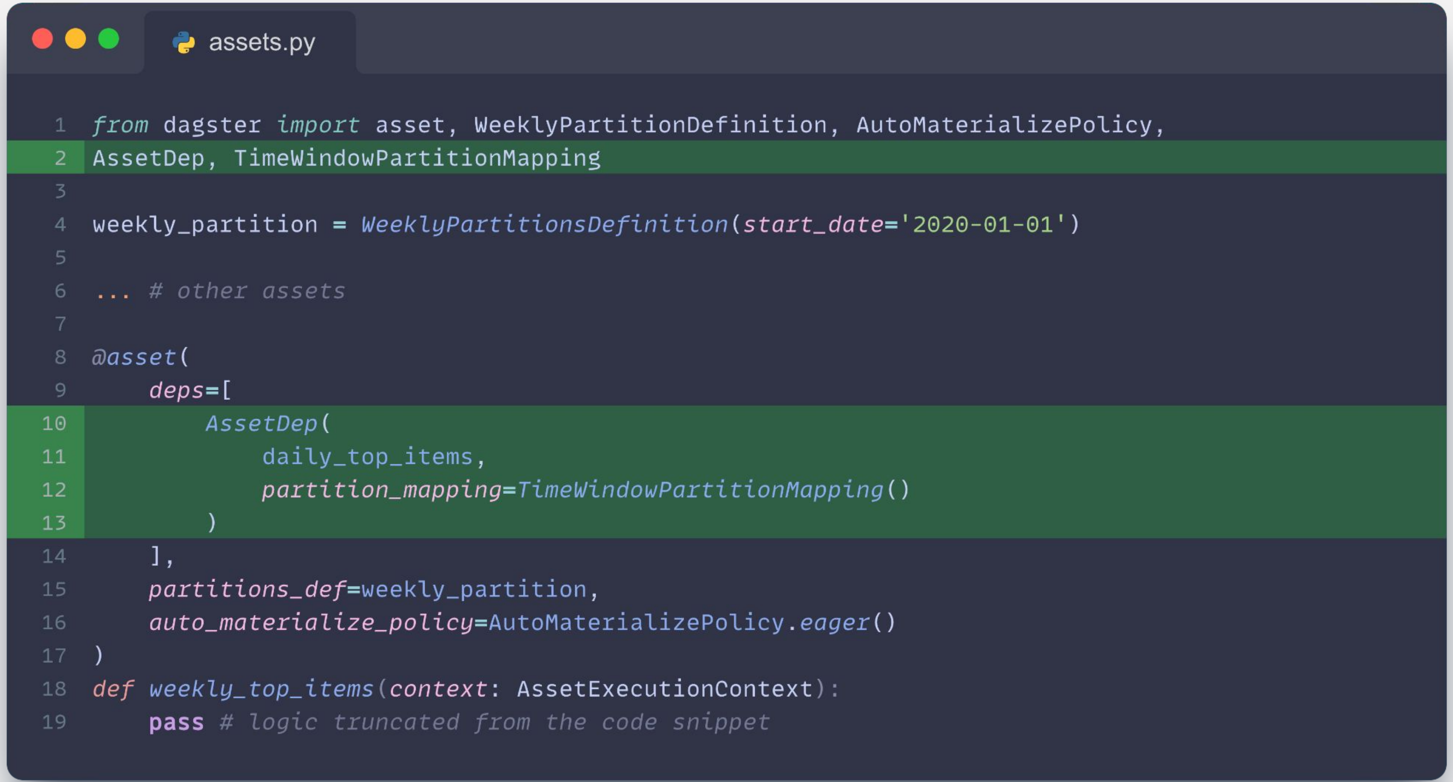
assets.py

```python
from dagster import asset, AssetDep, AllPartitionMapping

@asset(
    deps=[
        dim_customers, dim_promotions, dim_destinations,
        AssetDep(
            dim_items,
            partition_mapping=AllPartitionMapping()
        )
    ],
    partitions_def=monthly_partition,
)
def fct_orders(context):
    start_date, end_date = context.partition_time_window
    get_fct_orders(start_date=start_date, end_date=end_date)
```

Partitions depending on certain partitions



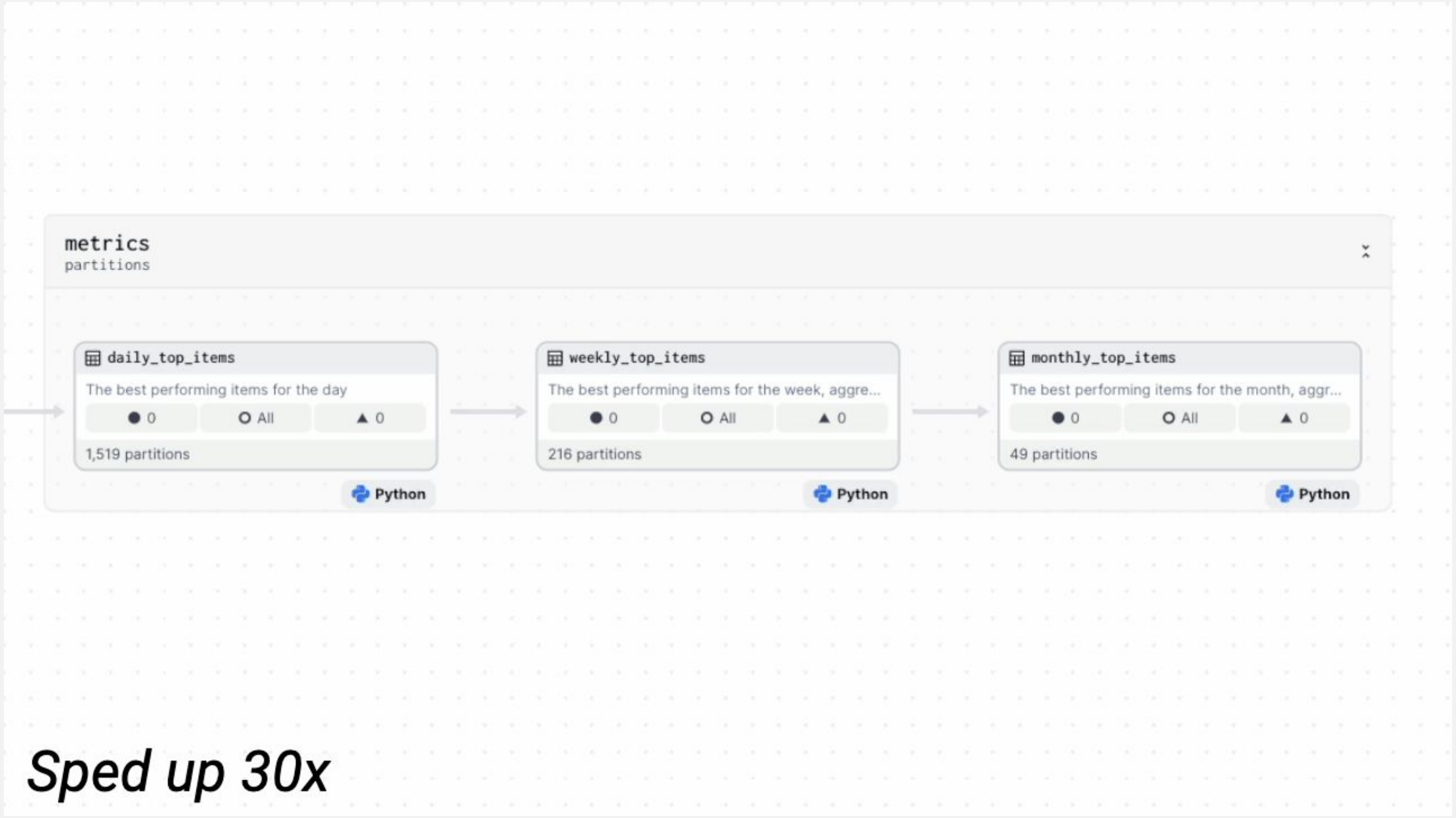
Define a partition



assets.py

```python
from dagster import asset, WeeklyPartitionDefinition, AutoMaterializePolicy,
AssetDep, TimeWindowPartitionMapping

weekly_partition = WeeklyPartitionsDefinition(start_date='2020-01-01')

... # other assets

@asset(
    deps=[
        AssetDep(
            daily_top_items,
            partition_mapping=TimeWindowPartitionMapping()
        )
    ],
    partitions_def=weekly_partition,
    auto_materialize_policy=AutoMaterializePolicy.eager()
)
def weekly_top_items(context: AssetExecutionContext):
    pass # logic truncated from the code snippet
```

And here it is



Sped up 30x

dagster

Partition Mappings configure how an asset depends on a partitioned asset

You have many mappings available to you, ex. `AllPartitionMapping`, `TimeWindowPartitionMapping`, `IdentityPartitionMapping`, `LastPartitionMapping`, etc.

dagster

Partitioning data makes pipelines faster and cheaper

Dagster has great native support for partitions

How downstream assets depend on partitions is configurable

Next steps & resources

Join us in Slack

Connect with other data practitioners. Share knowledge or find help

bit.ly/DagSlack

Sign up for Dagster Cloud

Sign up for Dagster Cloud and get started with a free 30 day trial

bit.ly/DagCloud

Get the Newsletter

Stay up-to-date on the latest events and news

bit.ly/DagNews

Star the GitHub Repo

Let us know you enjoyed this presentation!

bit.ly/DagDemo