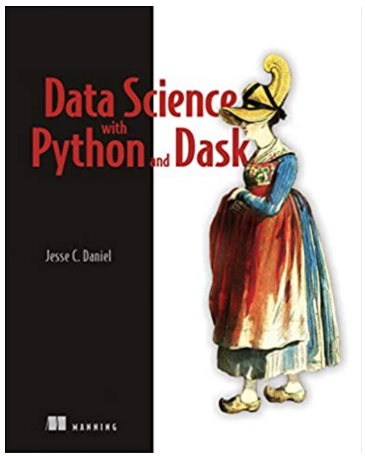
9.3: Python Dask

- **Instructor**: Dr. GP Saggese, gsaggese@umd.edu
- **Resources**
  - Web resources:
    - Dask project
    - Dask examples
  - Tutorial
    - Dask_tutorial
    - Dask_advanced_tutorial
  - Class project
  - Mastery
    - Data science with Python and Dask, 2019



Dataset Size Issues

- **Small datasets ($<$ 1 GB)**
  - Fits into RAM
  - No disk paging needed
- **Medium dataset ($<$ 1TB)**
  - Doesn't fit into RAM
  - Fits into local disk
    - Performance penalty with local disk
  - Need multiple CPU cores
    - Difficult to leverage parallelism with Python/Pandas
- **Large dataset ($>$ 1TB)**
  - Doesn't fit into RAM
  - Doesn't fit into local disk
  - Need multiple servers
    - Python/Pandas not built for distributed datasets
    - Use frameworks for massive datasets
    - E.g., Hadoop, Spark, Dask, Ray

Dataset Size Issues

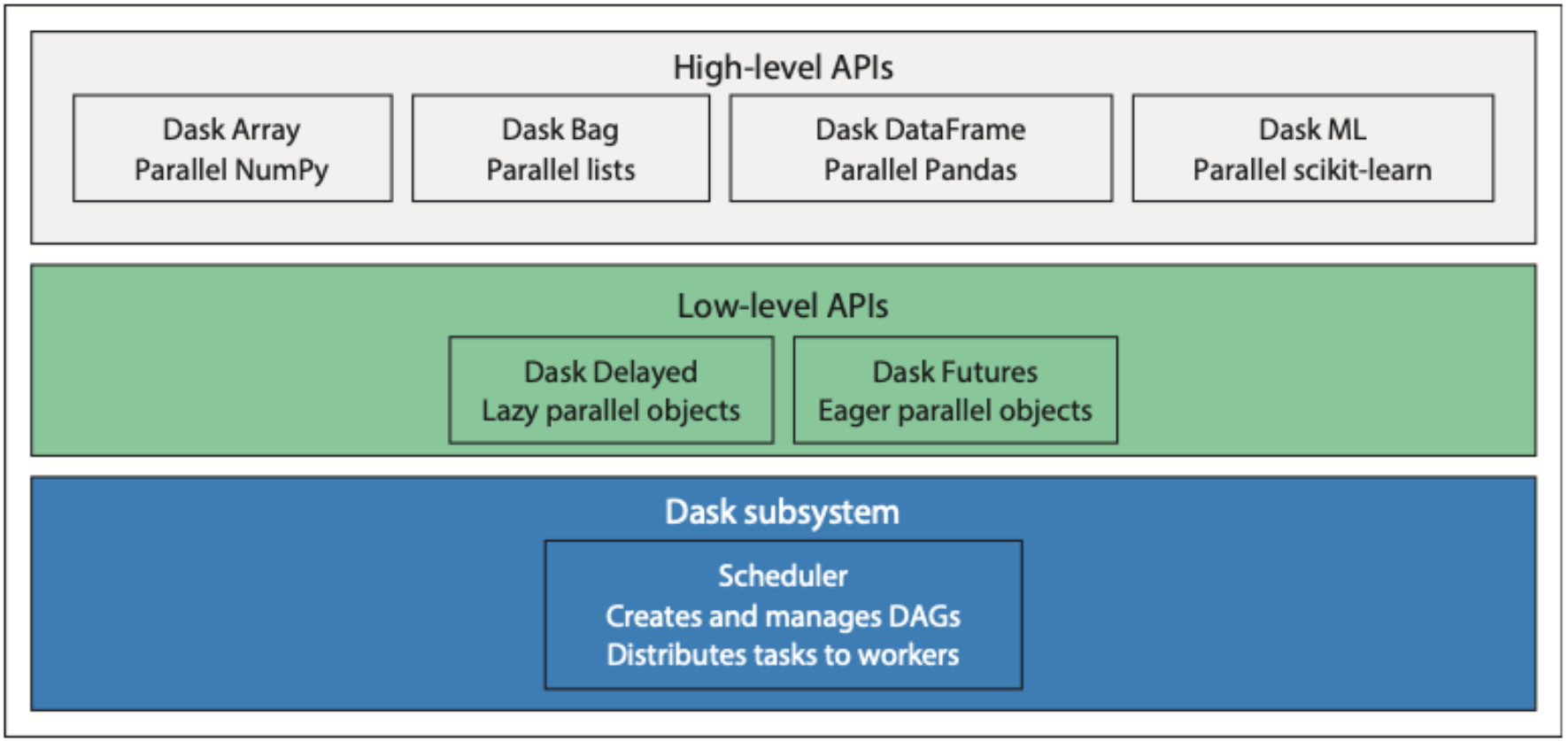
Category	Size
Small datasets	< 1 GB
Medium datasets	< 1 TB
Large datasets	> 1 TB

- **The thresholds are fuzzy and changing over time**
  - Scale computer 10x to get 10x bigger datasets
- **Problem with scaling datasets**
  - Long run times
  - Rewriting code for different dataset sizes
  - Plan what and how to do efficiently
  - Cumbersome framework (Pandas easy, Hadoop difficult)

Dask

- **Dask is written in Python**
  - Scales Numpy, Pandas, sklearn
  - Dask objects wrap library objects (e.g., Pandas DataFrame, numpy array)
  - Parallel parts are "chunks" or "partitions"
    - Queued for work
    - Shipped between machines
    - Worked locally
- **Pros**
  - Use familiar interfaces
  - Write code optimized for parallelism
    - Dask handles heavy lifting
- **Scaling Dask is easy**
  - Prototype on local machine, use cluster when needed
  - No code refactoring needed
  - No cluster-specific issues
    - E.g., resource management, data recovery, data movement
  - Runs on multi-core
  - Uses cluster managers
    - E.g., Yarn, Mesos, Kubernetes, AWS ECS

Dask Layers


High-level APIs
Dask Array
Parallel NumPy
Dask Bag
Parallel lists
Dask DataFrame
Parallel Pandas
Dask ML
Parallel scikit-learn
Low-level APIs
Dask Delayed
Lazy parallel objects
Dask Futures
Eager parallel objects
Dask subsystem
Scheduler
Creates and manages DAGs
Distributes tasks to workers

Scaling Up vs Scaling Out

- **Scaling up**
  - Replace equipment with larger, faster options
    - E.g., buy a larger pot, replace knife with food processor
  - **Pros**
    - Better hardware, no code changes needed
  - **Cons**
    - Exceed current machine capacity eventually
    - Cost: more powerful machines are expensive

- **Scaling out**
  - Divide work between many workers in parallel
    - E.g., buy more pots and hire more cooks
  - **Pros**
    - Task scheduler organizes computation, assigns workers to tasks
    - Cost-effective, no specialized hardware needed
  - **Cons**
    - Write code to expose parallelism
    - Maintain cluster costs

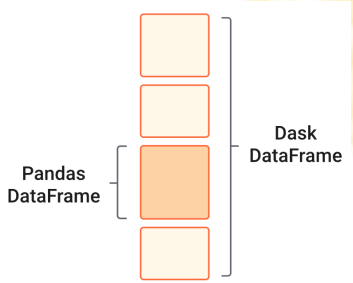
Dask: Computation

- **Lazy computations**
  - Define transformations on data
  - Define next computation without waiting
  - Operate in chunks to avoid loading entire data in memory
  - E.g.,
    - Split 2GB file into 32 64MB chunks
    - Operate on 8 chunks per server
    - Max memory use: 512MB = (8 x 64MB)
  - Track object dimensions and data types
    - No code execution
- `compute()`
  - Run computation (materialize)

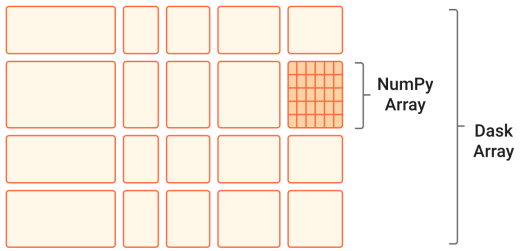
    ```
    missing_count_pct = missing_count.compute()
    ```
- `persist()`
  - Discard intermediate work to minimize memory
  - Re-run graph for additional computation on intermediate nodes
  - `persist()` keeps intermediate result in memory
  - Speeds up large, complex DAGs for reuse

Dask: Data Structures

- **Dask DataFrame**
  - Implements Pandas DataFrame
  - Tabular/relational data



- **Dask Array**
  - Implements numpy ndarray
  - Multidimensional array

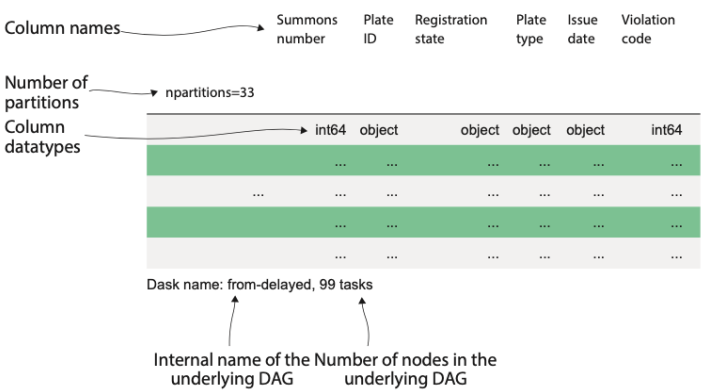


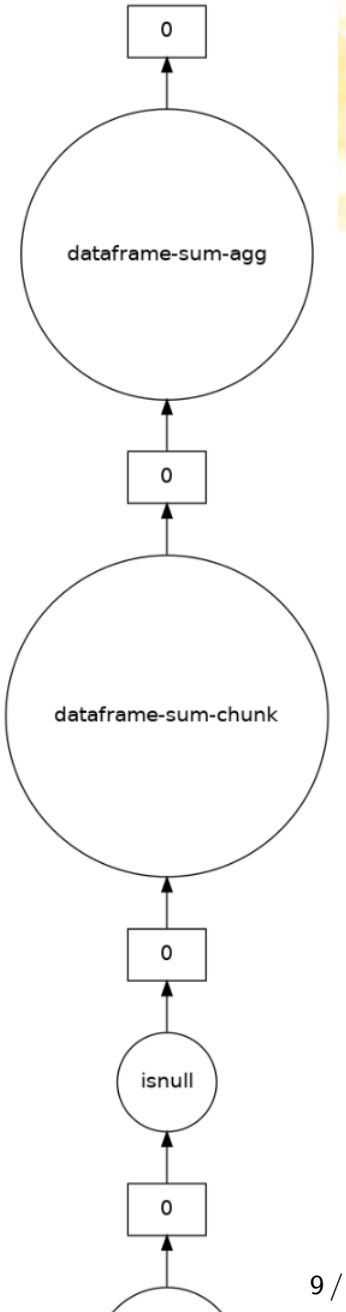
- **Dask Bag**
  - Coordinates Python lists of objects
  - Parallelize computations on unstructured/semi-structured data

```
[1, 2, 3, 4, 5]
[1, 2, 3] [4, 5]
```

Dask Reading Data

- dask.dataframe.read_csv()
  - Doesn't load the data in memory with
  - Tries to infer the types of the columns
    - By randomly sampling some data
    - Best to set the data types
    - Even better is to use Parquet since it stores data and types together
- Partitions = chunks of data that can be worked independently
  - E.g., 33 partitions
  - Graph is composed of 99 tasks
  - Each partition reads data, splits data, initializes df object

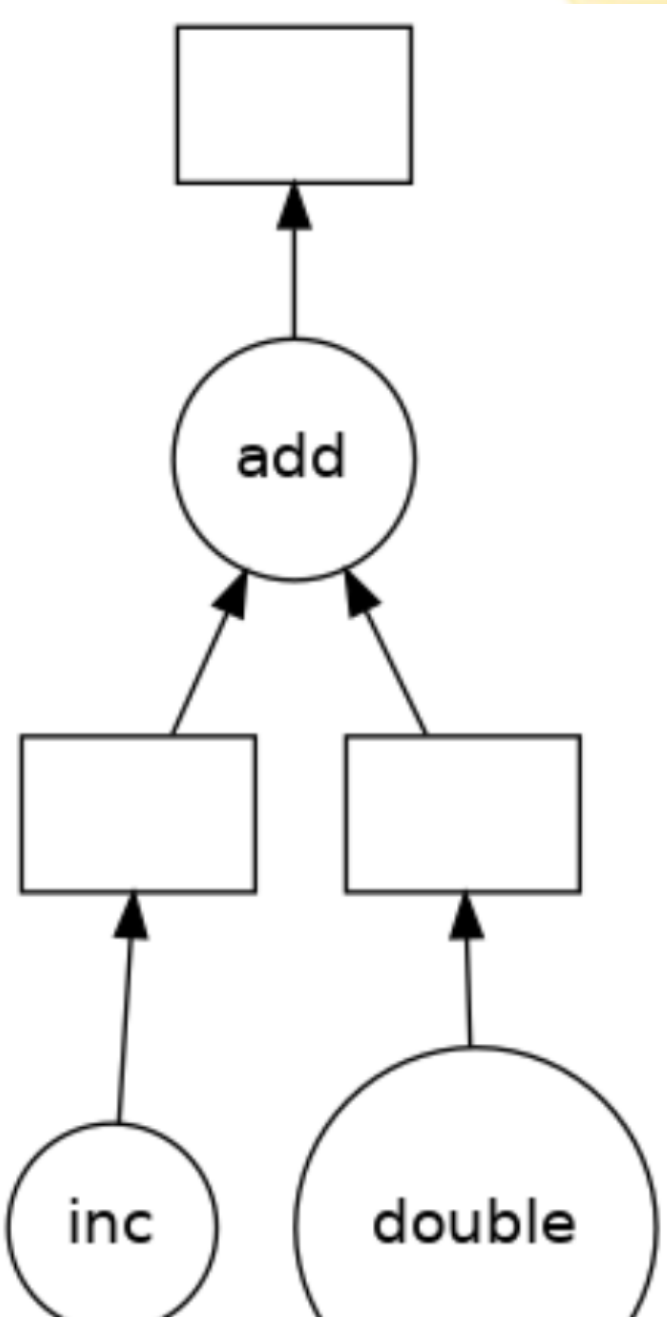




Low Level APIs: Delayed

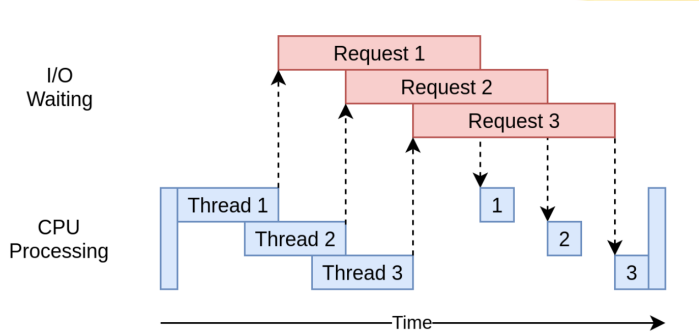
- Handle computations that don't fit in native Dask data structures (e.g., Dask DataFrame)
- In the example below there is parallelism that can be exploited

```python
def inc(x):
    return x + 1

def double(x):
    return x * 2

def add(x, y):
    return x + y

data = [1, 2, 3, 4, 5]

output = []
for x in data:
    # (x + 1) + (x * 2) = 3x + 1
    a = inc(x)
    b = double(x)
    c = add(a, b)
    # 1 -> 4
    # 2 -> 7
    # 3 -> 10
    # 4 -> 13
    # 5 -> 16
    output.append(c)

# 4 + 7 + 10 + 13 + 16 = 20 + 20 + 10 = 50
total = sum(output)
print(total)
```



Low Level APIs: Futures

- In parallel programming, a "future" encapsulates the asynchronous execution of a callable, representing the eventual result of the operation
- Futures is the most general way of specifying concurrency in Dask
  - Everything can be expressed in terms of futures
  - User can specify what's blocking and what's not blocking
- Python**concurrent.futures**
  - High-level interface for asynchronously executing callables
  - Thread pool or Process




```
def inc(x):
    return x + 1

def add(x, y):
    return x + y

a = client.submit(inc, 10)
b = client.submit(inc, 20)
```

```
>>> a
<Future: status: pending, key: inc-b8aaf26b99466a7a1980efa1ade6701d>
```

```
>>> a
<Future: status: finished, type: int, key: inc-b8aaf26b99466a7a1980efa1ade6701d>

>>> a.result()  # blocks until task completes and data arrives
11
```

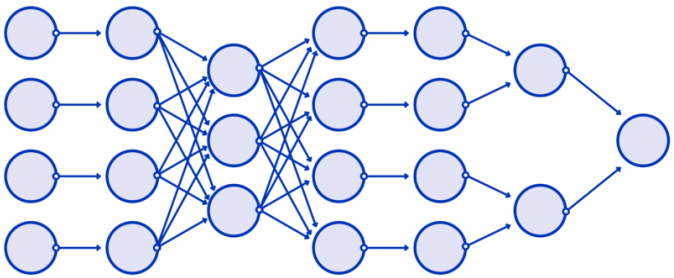
SCIENCE ACADEMY

Different Types of Parallel Workload

- Break program in medium-size tasks of computation
  - E.g., a function call

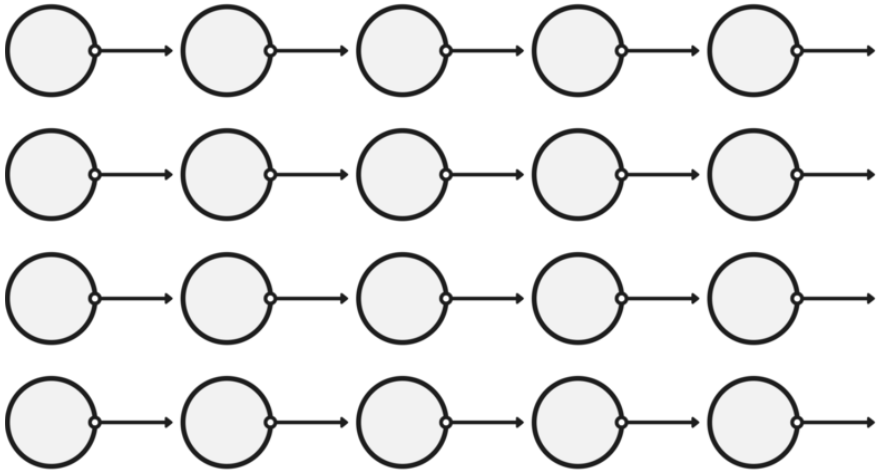
MapReduce

Hadoop/Spark/Dask



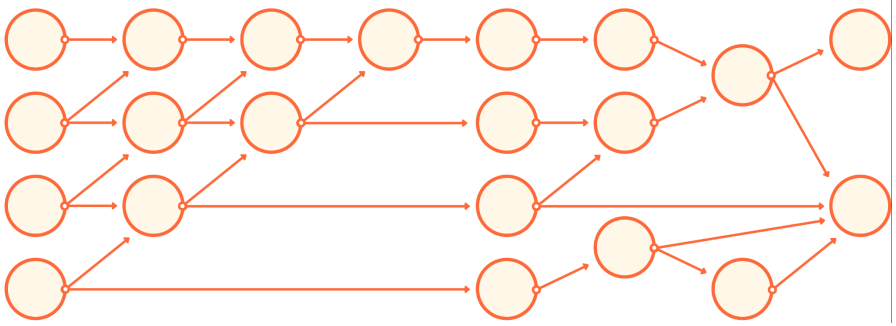
Embarrassingly Parallel

Hadoop/Spark/Dask/Airflow/Prefect



Full Task Scheduling

Dask/Airflow/Prefect



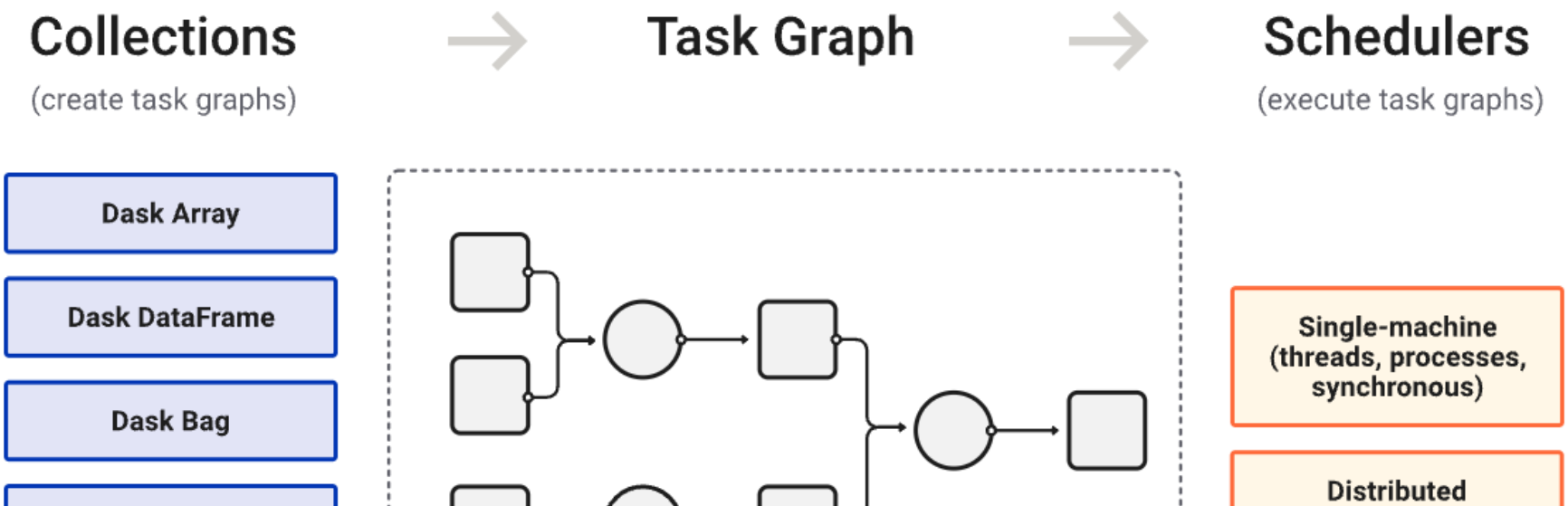
Encoding Task Graph

- Dask encodes tasks in terms of Python dicts and functions ::: columns ::::

```
def inc(i):
    return i + 1

def add(a, b):
    return a + b

x = 1
y = inc(x)
z = add(y, 10)
```

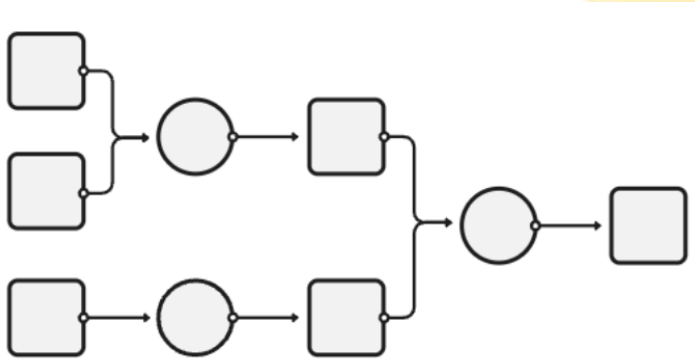
Task Scheduling

- Data collections (Bags, Arrays, DataFrame) and their operations create task graphs
  - Nodes in the task graph are Python functions
  - Edges are dependencies (e.g., output from one task used as input in another task)
- Task graphs are scheduled for execution
- Single-machine scheduler
  - Use local process or thread pool
  - Simple but it can only run on a single machine
- Distributed scheduler
  - It can run locally or distributed across a cluster



Task Scheduling

- **Dask task scheduler orchestrates the work dynamically**
  - Not a static scheduling of operations like a relational DB
  - When the computation takes place, Dask dynamically assesses:
    - What tasks has been completed
    - What tasks is left to do
    - What resources (CPUs) are free
    - Where the data is located
- **This dynamic approach handles a variety issues:**
  - Worker failure
    - Just re-run
  - Workers completing work at different speeds



Dask vs Spark

- Spark has
- **Pros**
  - Popular framework for analyzing large datasets
  - In-memory alternative to MapReduce / Hadoop
- **Cons**
  - Spark is a Java library, supporting Python through PySpark API
    - Python code is executed on JVM through `py4j`
    - Difficult to debug since execution occurs outside Python
  - Different DataFrame API than Pandas
    - Learn how to do things "the Spark way"
    - You might need to implement things twice to go from exploratory analysis to large experiments / production
  - Optimized for MapReduce operations over a collection
  - Difficult to set-up and configure

Tutorial

Tutorial - From the official documentation
https://docs.dask.org/en/stable/10-minutes-to-dask.html