Lecture 6: Data - Dask

AC215

Pavlos Protopapas
SEAS/Harvard

Outline

1. Recap
2. Dask
3. Directed Acyclic Graph (DAGs)
4. Computational Resources
5. Task Scheduling
6. Tutorial

Outline

1. **Recap**
2. Dask
3. Directed Acyclic Graph (DAGs)
4. Computational Resources
5. Task Scheduling
6. Tutorial

Recap: Motivation

The 3 components for better Deep Learning

More Data

Better/Faster Models

Faster Hardware

Recap: Motivation

The 3 components for better Deep Learning

More Data

- Extraction
- Transformation
- Labeling
- Versioning
- Storage

- Processing
- Input to Training

Better/Faster Models

- SOTA Models
- Transfer Learning
- Distillation
- Compression

Faster Hardware

- Scaling data processing
- GPU, TPU
- Multi GPU Server Training

Motivation: Data Size

Dataset type	Size range	Fits in RAM?	Fits on local disk?
Small dataset	Less than 2-4 GB	Yes	Yes
Medium dataset	Less than 2 TB	No	Yes
Large dataset	Greater than 2 TB	No	No

Motivation: Data Size

Challenges:

- Medium datasets will not all fit in memory (RAM)
- Large datasets will not fit in disk (Hard drive)

Motivation: Data Size

Challenges:

- Medium datasets will not all fit in memory (RAM)
- Large datasets will not fit in disk (Hard drive)

Solution:

- Building data pipelines
  - Read data in batches which can fit in RAM
  - Feed data in batches to GPU
  - Read data from big data store in batches, so not all data need to be present in local hard drive

Motivation: Data Size

Tools:

- Google Cloud Storage (Big data store)
- Dask
- TensorFlow Data
- TensorFlow Records

Data Tools

Data Size:

- Google Cloud Storage
- Dask
- TensorFlow Data
- TensorFlow Records

Data Management:

- Label Studio
- DVC
- Kubeflow

Data Tools

Data Size:

- ✓ **Google Cloud Storage**
- Dask
- TensorFlow Data
- TensorFlow Records

Data Management:

- ✓ **Label Studio**
- ✓ **DVC**
- Kubeflow

Cloud Storage

Advantage of Cloud Storage:

- Unlimited capacity (Scalability)
- 99.99% uptime
- Distributed storage
- Data security
- Low cost

Examples:

- AWS S3
- Azure data lake
- GCS (Google Cloud Storage)

Outline

1. Recap
2. **Dask**
3. Directed Acyclic Graph (DAGs)
4. Computational Resources
5. Task Scheduling
6. Tutorial

What is Dask

Dask is a free and open-source library for parallel computing in Python. It allows you to work on arbitrarily large datasets and dramatically increases the speed of your computations.

Pandas

```
# read data using DataFrame API
df = pd.read_csv("path to csv")

print("Shape:", df.shape)
Shape: (100000, 43)

# Display the top rows
df.head()
```

Dask

```
# read data using DataFrame API
df = dd.read_csv("path to csv")

print("Shape:", df.shape)
Shape: (Delayed('int-weyus...'), 43)

# Display the top rows
df.head()
```

Dask's features

What is **unique** about Dask:

- It allows **to work with larger datasets** making it possible to parallelize computation. At the same time, Dask can be used effectively to work with both **medium datasets** on a single machine and **large datasets on a cluster**.

- It **simplifies** the operation and therefore reducing the cost of using more complex **infrastructure**.

Dask

- Dask is entirely built in Python and seamlessly scales libraries like NumPy, Pandas, and scikit-learn. This makes it easy to learn for data scientists familiar with Python's syntax, while also offering flexibility

- Dask can be used as a general framework for **parallelizing** most Python objects.

- Additionally, Dask has minimal configuration and maintenance requirements, simplifying its adoption and ongoing use.

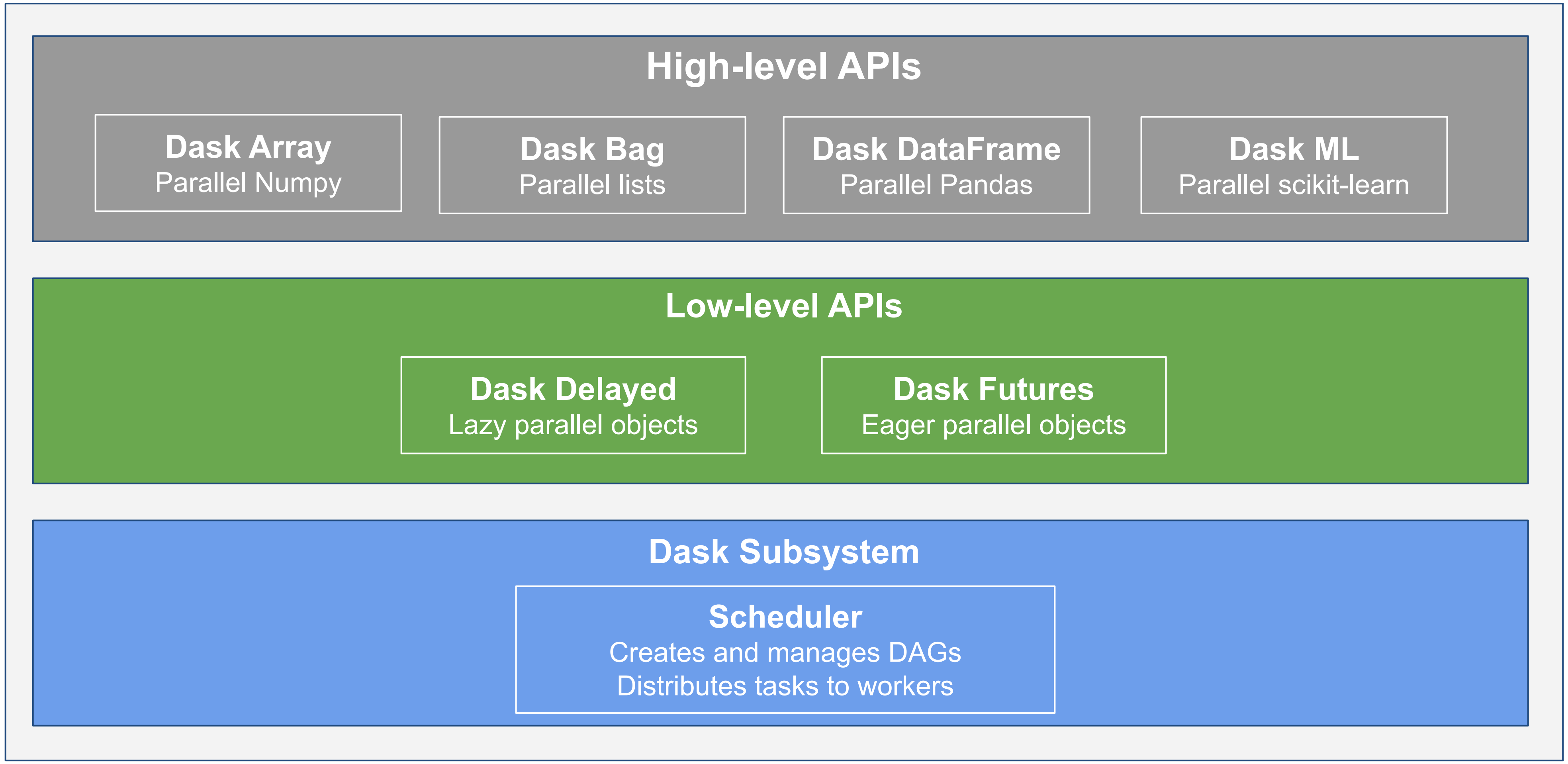
Dask Usage Across Different Dataset Sizes

- Small Datasets:Not particularly advantageous for small datasets due to the overhead it introduces. Complex operations can often be performed without needing to spill data to disk, which avoids slowing down the process.

- Medium Datasets:Highly beneficial for medium-sized datasets as it enables local machine processing. However, leveraging parallelism can be challenging since Pandas doesn't easily distribute work across multicore systems.

- Large Datasets:Great for handling large datasets. Traditional libraries like Pandas, NumPy, and scikit-learn are not optimized for these volumes, as they were not originally designed for distributed computing.

Dask Architecture



Outline

1. Recap
2. Dask
3. **Directed Acyclic Graph (DAGs)**
4. Computational Resources
5. Task Scheduling
6. Tutorial

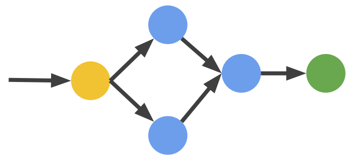
Directed Acyclic Graph (DAGs)

A graph is a representation of a **set of objects that have a relationship** with one another. It is used to representing a wide variety of information.

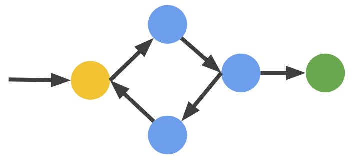
A graph consists of:

- **node**: a function, an object or an action
- **line**: symbolize the relationship among the nodes

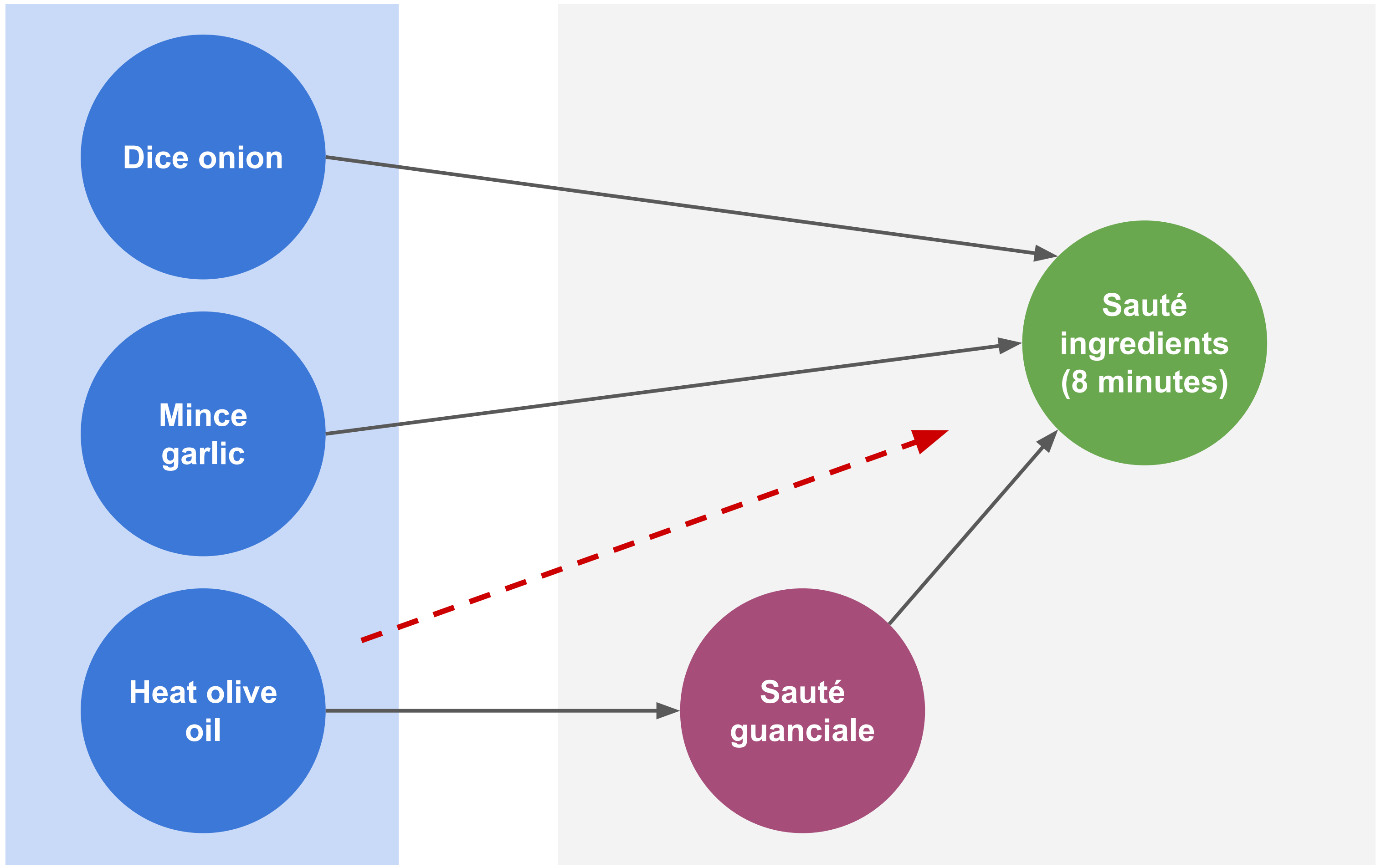
In a directed acyclic graph there **is one logical way to traverse** the graph. No node is visited twice.



In a *cyclic graph*: exist a feedback loop that allow to revisit and repeat the actions within the same node.



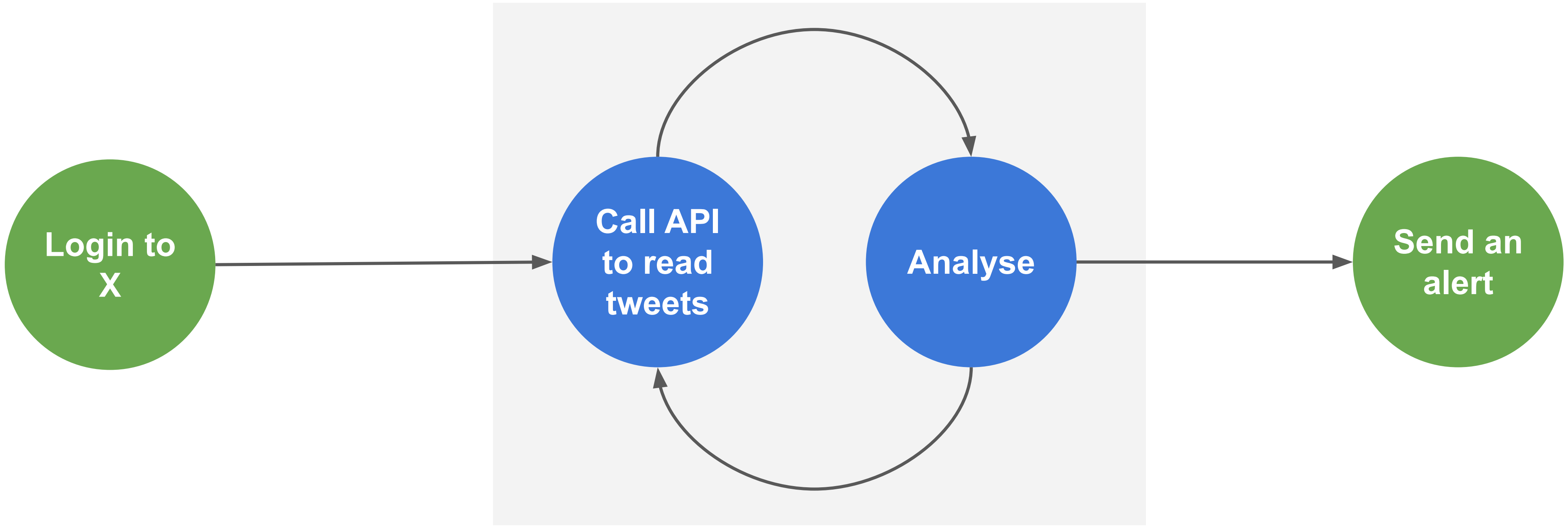
Directed Acyclic Graph [Cooking example]



No dependencies.
These tasks can be started in any order

These tasks can only be started when all nodes connected to them have been completed.

Directed ~~A~~cyclic Graph [Real Time Twitter Analysis]



These two tasks are connected to each other in an infinite feedback loop. There is no logical termination point in this graph.

Outline

1. Recap
2. Dask
3. Directed Acyclic Graph (DAGs)
4. **Computational Resources**
5. Task Scheduling
6. Tutorial

Managing Computational Resources with Dask

Two Main Strategies:

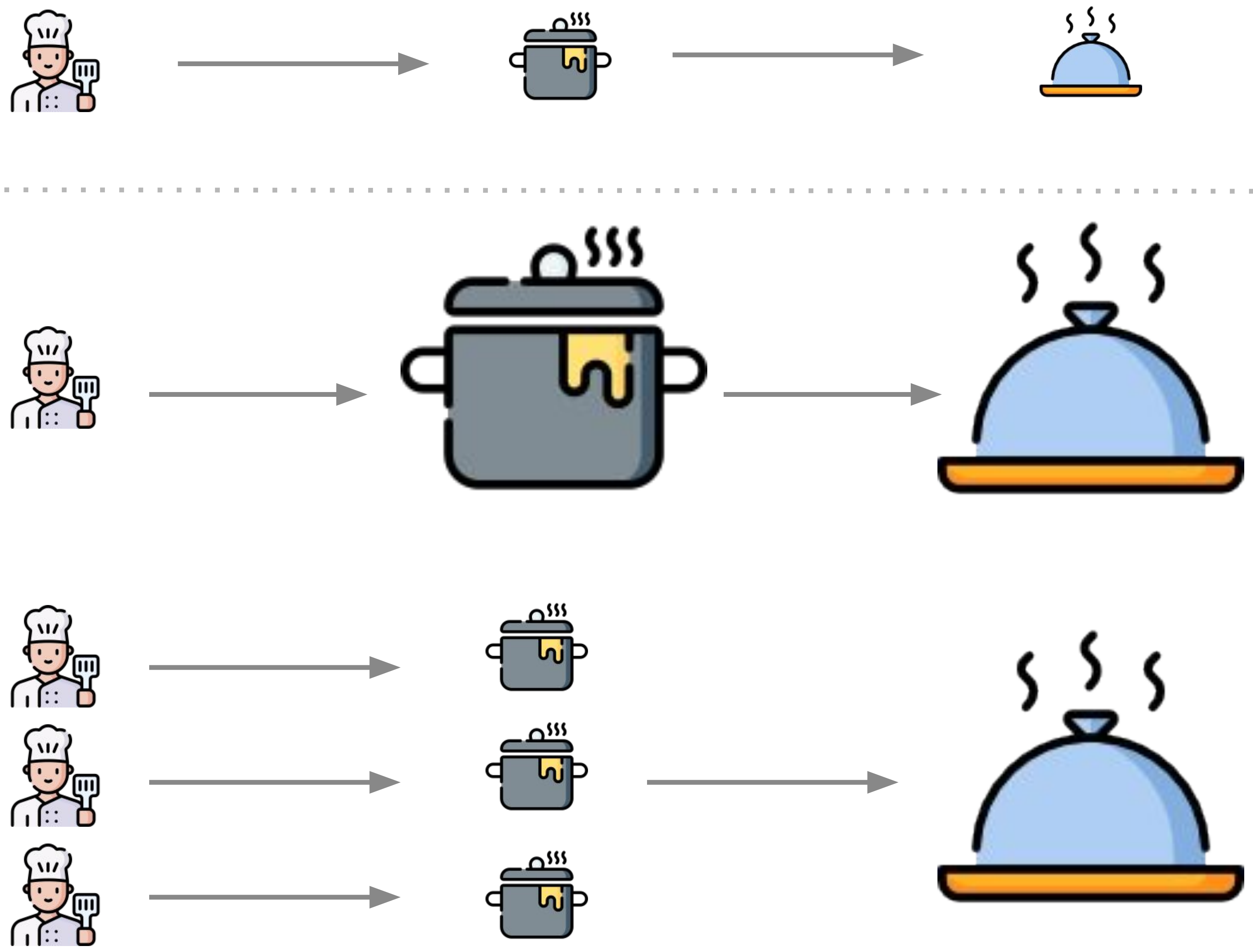
Scale Up

- **Description**: Increase the size of the available resource by investing in more efficient technology.
- **Pros**: Immediate boost in performance.
- **Cons**: Diminishing returns on investment.

Scale Out (Dask's Approach)

- **Description**: Add more, often cheaper, resources to the existing pool.
- **Pros**: Easier to manage; Cost-effective.
- **Cons**: Need to distribute workload efficient

Computational Resources



Concurrency & Resource Management in Dask

What is Concurrency?

When the volume of tasks increases, some computational resources may be underutilized, resulting in inefficiencies.

Resource Starvation

A situation where certain computational elements are idle due to insufficient availability of shared resources.

Role of Schedulers

Schedulers in Dask tackle this issue by dynamically allocating appropriate resources to different tasks, maximizing efficiency.

Computational Resources

Dicing onions
Shared resources
Mincing garlic

This cook must wait and remain idle until either a knife becomes available or a new task that does not require a knife is available.
This is an example of a resource-starved worker.

Dask's Resilience to Failures

Fault Tolerance in Dask

Dask is designed to recover from node failures gracefully, ensuring minimal disturbance to the ongoing process.

Types of Failures

- **Worker Failures**

  When a worker exits unexpectedly, Dask reassigns its tasks to another worker. This may cause delays but prevents data loss.

- **Data Loss Failures**

  In extreme cases, where data is lost, the scheduler halts and restarts the entire process from the beginning.

Outline

1. Recap
2. Dask
3. Directed Acyclic Graph (DAGs)
4. Computational Resources
5. **Task Scheduling**
6. Tutorial

Task Scheduling

Dask performs a so called lazy computation. Until you run the method `.compute()`, *Dask* only splits the process into smaller logical pieces.

Even though the process is defined, the number of resources assigned and the place where the result will be stored are **not assigned** because the scheduler assigns them dynamically. This allow to recover from worker failure.

Dynamic Scheduling in Dask

Centralized Scheduling

Dask employs a central scheduler that coordinates task distribution across multiple workers.

Load Imbalances

Task distribution may vary due to server capabilities and data access, leading to imbalances in workload, power, and data access.

Adaptive Scheduling

To mitigate bottlenecks and improve performance, the Dask scheduler continuously adapts to changes, optimizing runtime.

Dask's Versatile Scheduling Options

Multi-threaded `scheduler='threads’`

- Ideal for I/O-bound tasks or tasks that benefit from shared memory.
- Note: May not be efficient for CPU-bound tasks due to Python's GIL.

Multi-process `scheduler='processes’`

- Suitable for CPU-bound tasks.
- Overcomes Python's GIL limitations.

Distributed Scheduler `client`

- Scalable to multiple servers.
- Offers advanced functionalities like data locality and worker constraints.

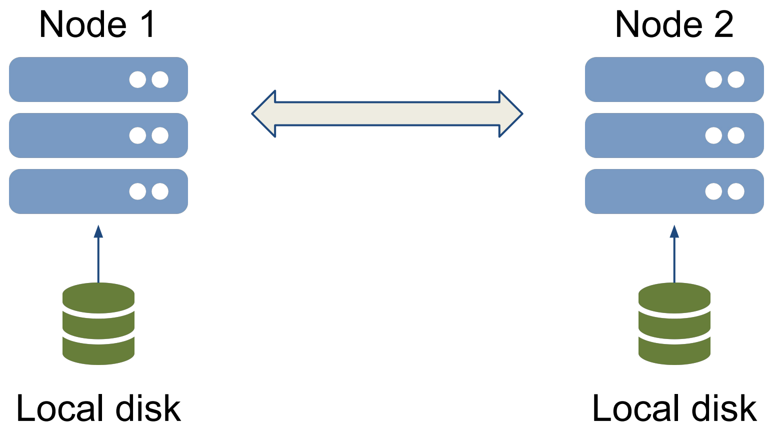
Single-threaded (Debugging) `scheduler='single-threaded’`

- Executes tasks in a single thread.
- Useful for debugging and profiling.

Task Scheduling

Assuming there are two nodes for computation.

Data needs to be replicated for each node in order to perform computation across nodes.

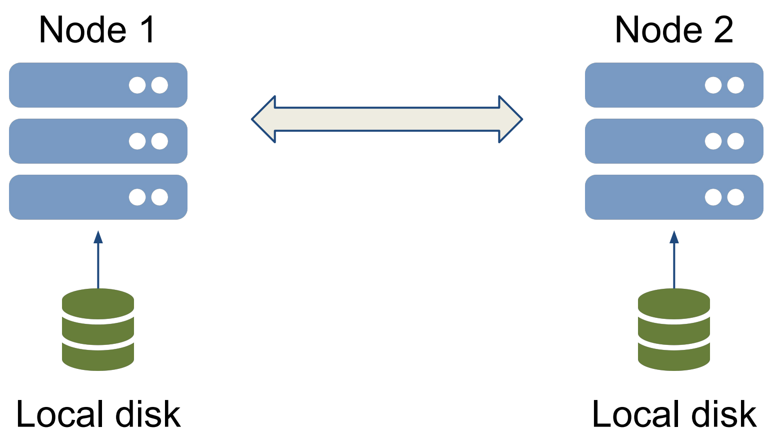


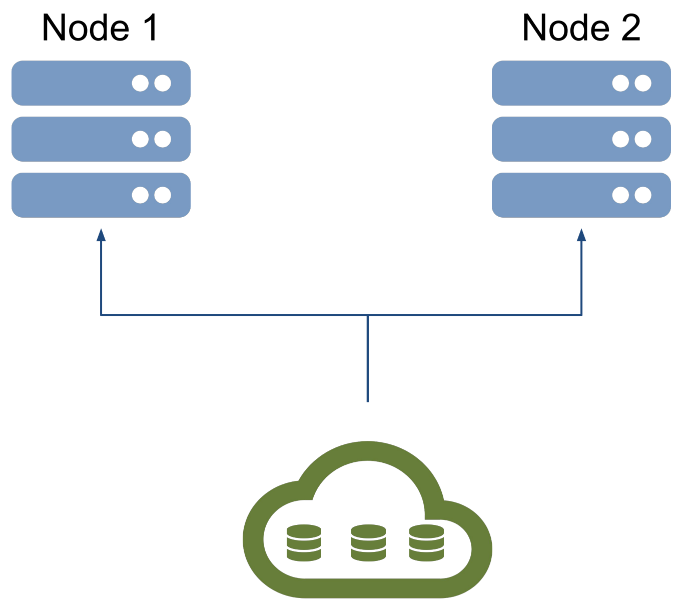
Task Scheduling

Assuming there are two nodes for computation.

Data needs to be replicated for each node in order to perform computation across nodes.

The remedy is to split data minimizing the number of data to broadcast across different local nodes.



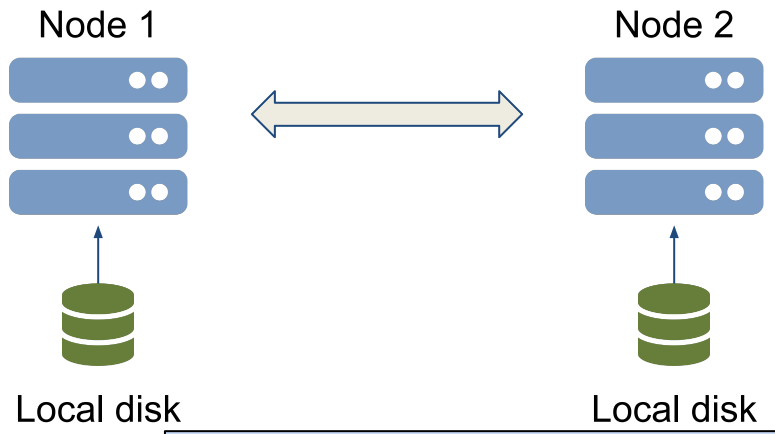


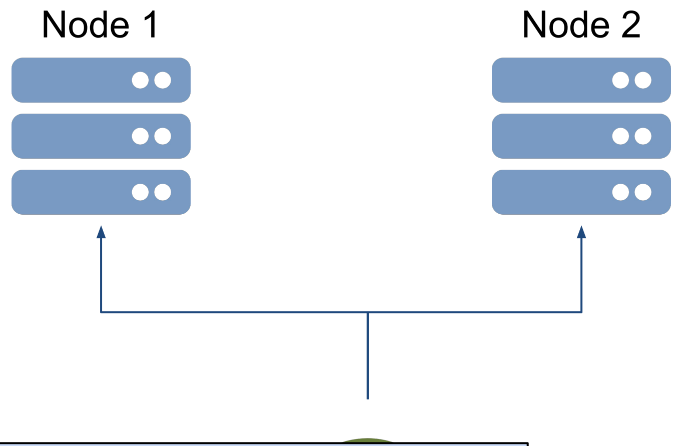
Task Scheduling

Assuming there are two nodes for computation.

Data needs to be replicated for each node in order to perform computation across nodes.

The remedy is to split data minimizing the number of data to broadcast across different local nodes.





For best performance, a Dask cluster should use a distributed file system (S3, HDFS, GCS) as a data storage.

Dask Review

- Dask can be used to scale popular Python libraries such as Pandas and NumPy allowing **to analyze dataset with greater size** (>8GB).
- Dask uses **directed acyclic graph to coordinate execution** of parallelized code across processors.
- Upstream actions are completed before downstream nodes.
- **Scaling out** (i.e. add workers) can improve performances of complex workloads, however, create overhead that can reduces gains.
- In case of failure, the step to reach a **node can be repeated** from the beginning without disturbing the rest of the process.

Dask Limitations

- Dask dataframe are immutable. Functions such as `pop` and `insert` are not supported.
- Dask does not allow for functions with a lot of data shuffling like `stack/unstack` and `melt`.
  - Do major filter and preprocessing in Dask and then dump the final dataset into Pandas.
- `Join, merge, groupby`, and `rolling` are supported but expensive due to shuffling.
  - Do major filter and preprocessing in Dask and then dump the final dataset into Pandas or limit operations only on index which can be pre-sorted.

Tutorial

Goals of the tutorial are

- Familiarize with Dask API
- Use Dask to compute dataset metrics
- 03_tutorial_data_dask.ipynb

THANK YOU