



DEEP
LEARNING
INSTITUTE



DLI Accelerated Data Science Teaching Kit

Lecture 13.1 - Using Dask and UCX with RAPIDS and BlazingSQL



The Accelerated Data Science Teaching Kit is licensed by NVIDIA, Georgia Institute of Technology, and Prairie View A&M University under the [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).

DASK

Introduction



A flexible library for parallel computing in Python.

DASK comprises of two major parts:

- **Dynamic task scheduling**
 - Optimized for interactive computational workloads.
- **“Big Data” collections**
 - Like parallel arrays, dataframes, and lists that extend common interfaces like NumPy, Pandas, or Python iterators to larger-than-memory or distributed environments
 - Run on top of dynamic task scheduler

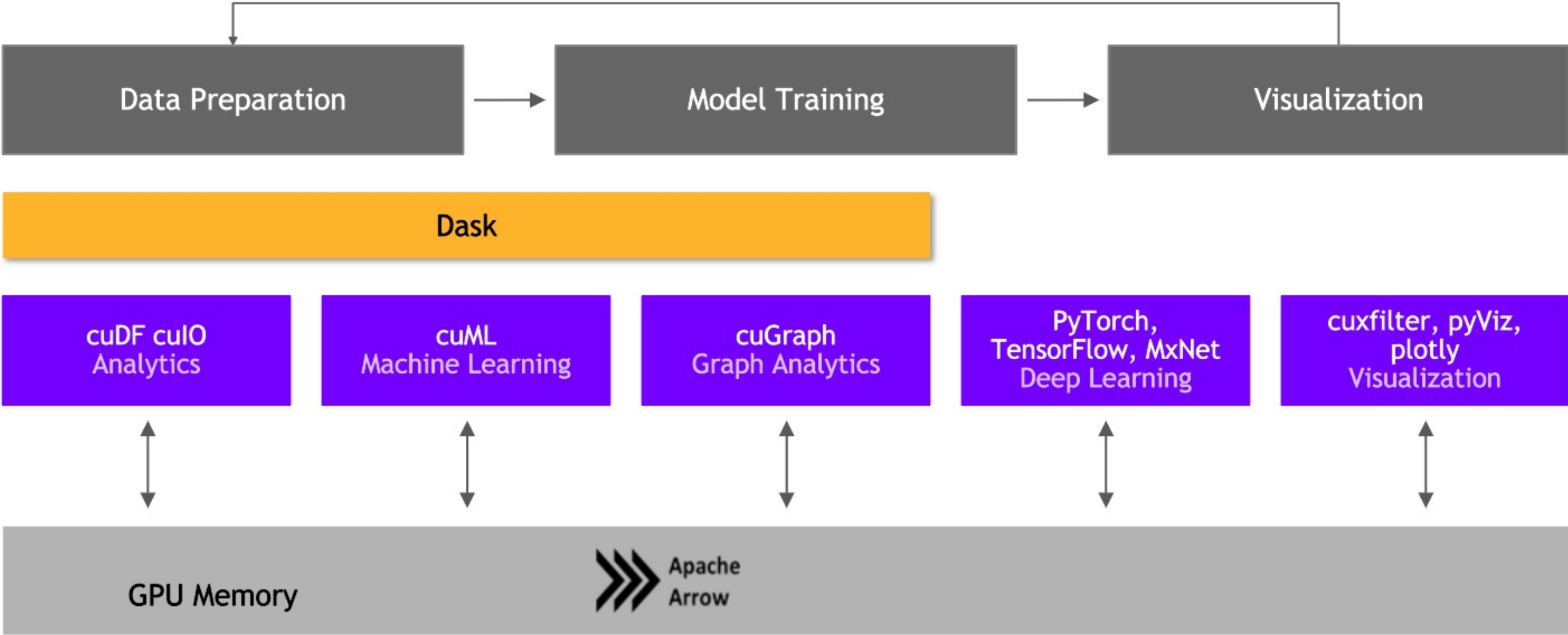
DASK

Why DASK?

- **Familiar:** Provides parallelized NumPy array and Pandas DataFrame objects
- **Flexible:** Provides a task scheduling interface for more custom workloads and integration with other projects.
- **Native:** Enables distributed computing in pure Python with access to the PyData stack.
- **Fast:** Operates with low overhead, low latency, and minimal serialization necessary for fast numerical algorithms
- **Scales up:** Runs resiliently on clusters with 1000s of cores
- **Scales down:** Trivial to set up and run on a laptop in a single process
- **Responsive:** Designed with interactive computing in mind, it provides rapid feedback and diagnostics to aid humans


DASK and RAPIDS

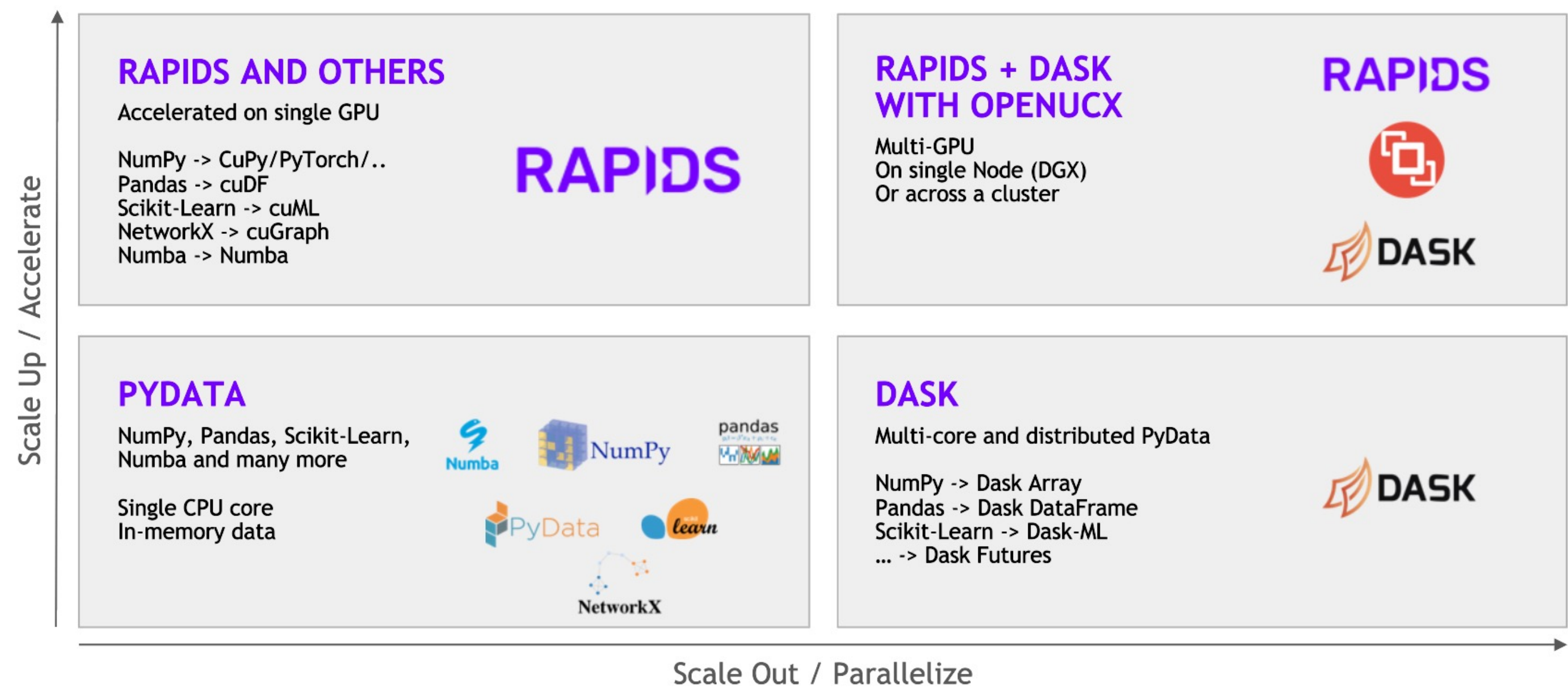
Scaling up and Scaling out



DASK and RAPIDS

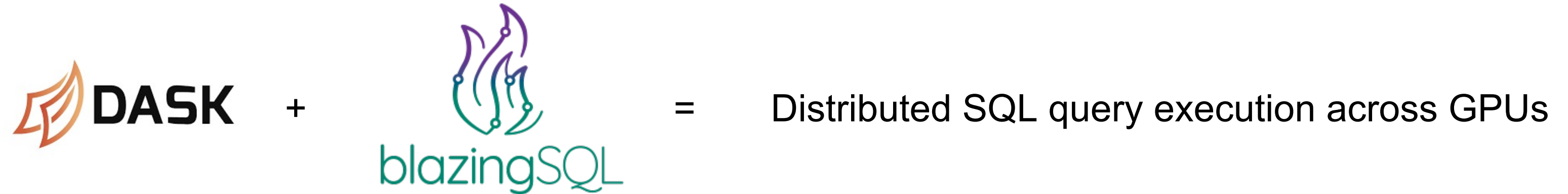
Scaling up and Scaling out

 **DASK** + **RAPIDS** = Scalable distributed performance



DASK and BlazingSQL

Scaling up and Scaling out



- BlazingSQL: Makes it easy to scale up a workload
- Pass Dask to BlazingSQL to identify the where the worker nodes are
- Make a BlazingSQL cluster by running a BSQL Engine process on each GPU
- Run queries on any arbitrary number of GPUs with very few code changes!
- Return results as `dask_cudf` dataframes to continue using python on them

DASK and BlazingSQL

Usage

Example: Using a local CUDA cluster

```
from blazingsql import BlazingContext
from dask_cuda import LocalCUDACluster
from dask.distributed import Client
cluster = LocalCUDACluster()
client = Client(cluster)
bc = BlazingContext(dask_client = client, network_interface = 'lo')
```


UCX

Introduction

- Unified Communication X is a framework that facilitates an easy and efficient way to construct widely used high-performance computing (HPC) protocols
 - Example protocols: MPI tag matching, RMA operations, rendezvous protocols, stream, fragmentation, remote atomic operations, etc.
- Accelerated networking library designed for low-latency, high-bandwidth transfers for host and GPU device memory objects
- UCX-Py is the python interface for UCX which is a high-level library that is easy for users to interact with

UCX with RAPIDS and BlazingSQL

Improved performance

- Leverage UCX-Py to handle communication bottleneck for distributed computation
- UCX-Py allows RAPIDS to use hardware interconnects like NVLink and InfiniBand
- Applications using DASK can easily make use of UCX with small code changes:
 - Instead of the default TCP protocol, use UCX protocol!
- To enable to UCX protocol in applications using Dask, set:
 - `cluster = LocalCUDACluster(protocol= "ucx", enable_tcp_over_ucx= True)`
- Efficient passing of single message passing GPU objects between endpoints



DEEP
LEARNING
INSTITUTE



DLI Accelerated Data Science Teaching Kit

Thank You