ML for HPC
Magurele Summer School

GPU Accelerated Distributed Data Science

Marco Celoria – CINECA

Magurele 9 July 2025

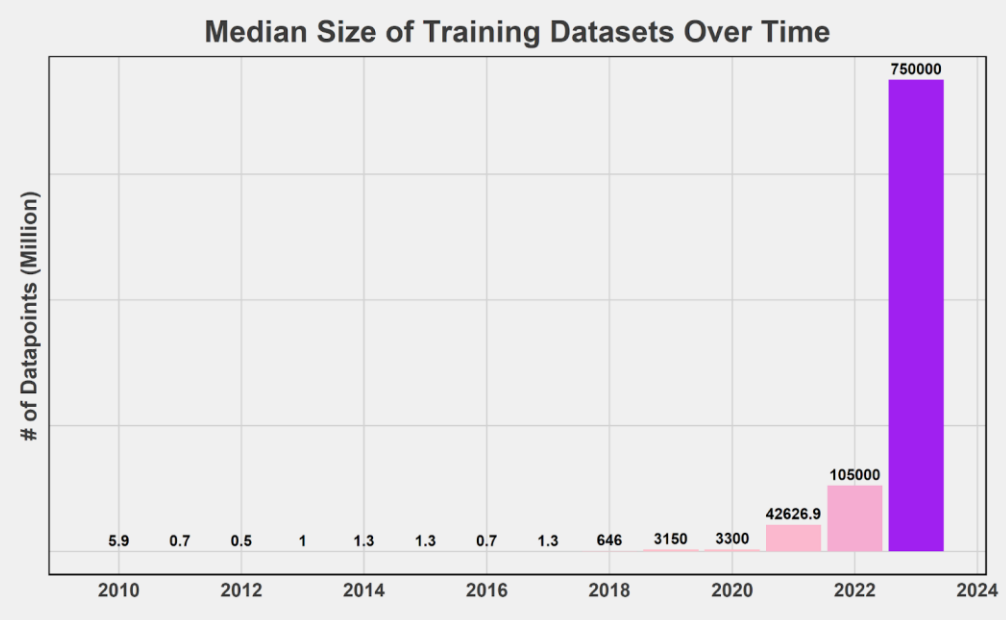
Machine Learning

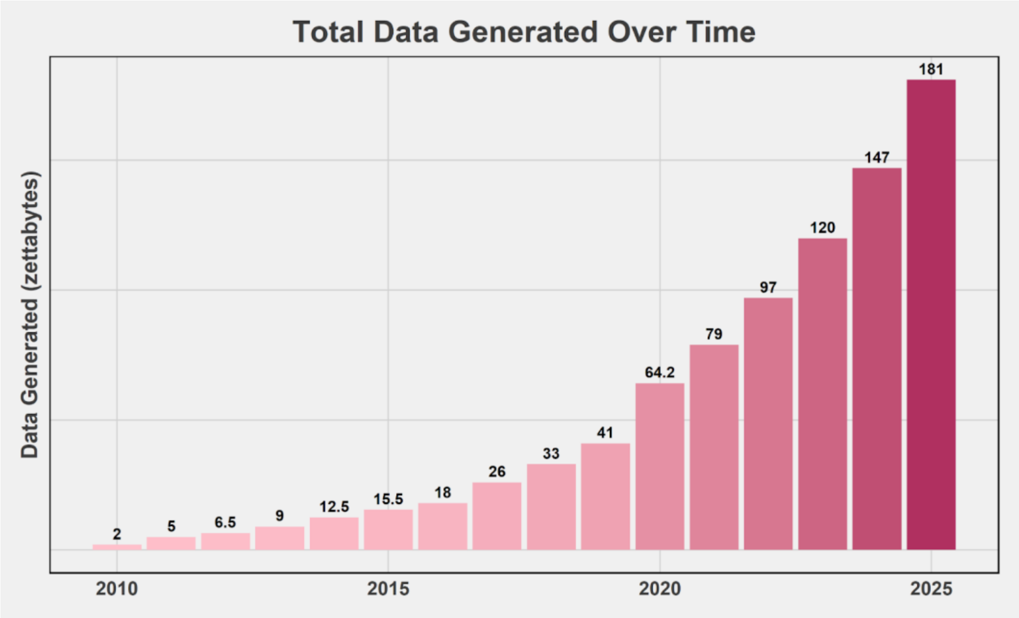
Machine Learning is building model from data using optimization.

High-dimensional non-convex optimization based on a growing wealth of data

- Larger and larger training sets
- GPUs used to speed up training times on such larger training sets
- New learning algorithms optimize convergence and generalization

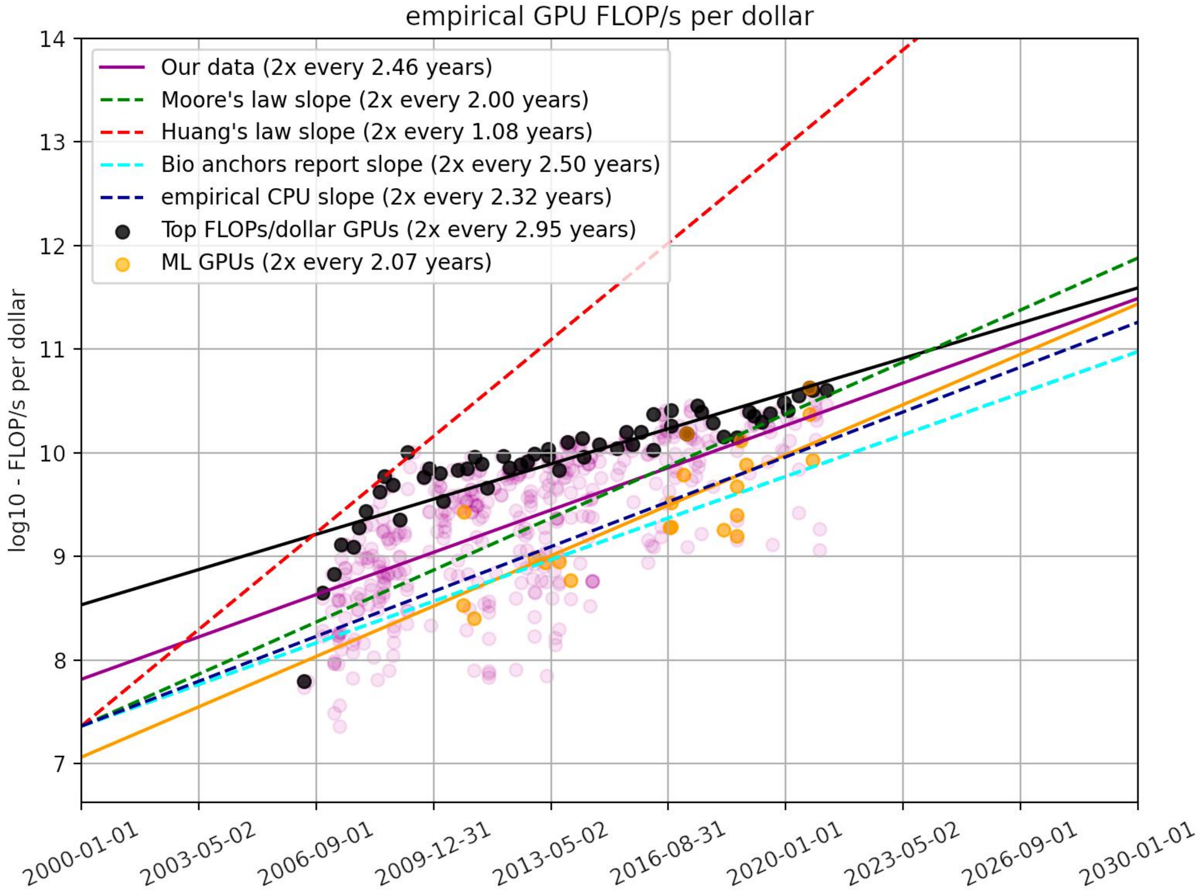
Machine Learning - Data





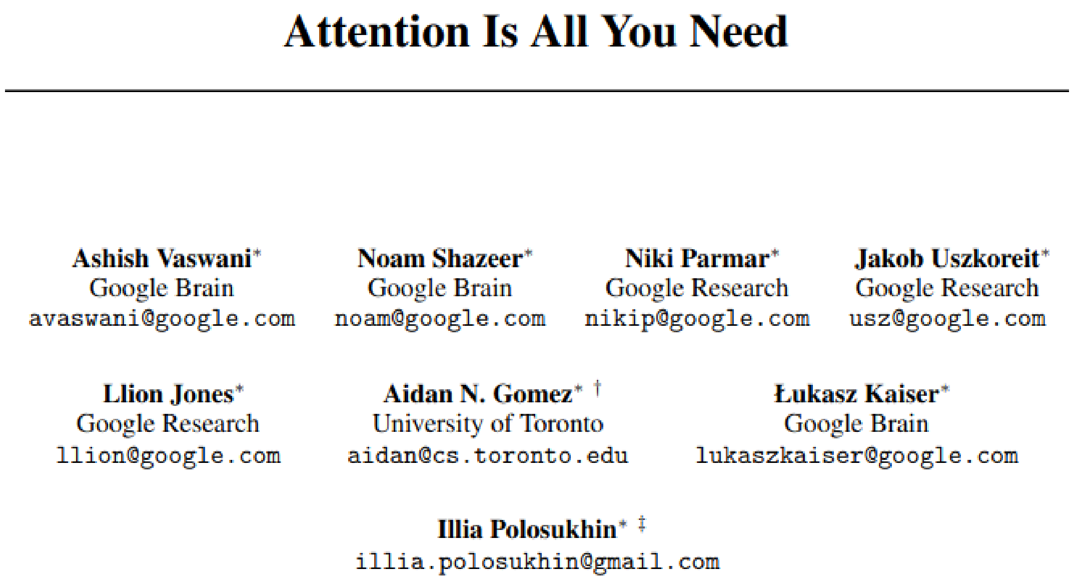
https://futuretech.mit.edu/news/what-drives-progress-in-ai-trends-in-data

Machine Learning - GPUs



Marius Hobbhahn and Tamay Besiroglu, "Trends in GPU Price-Performance".

Machine Learning - Algorithms



Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

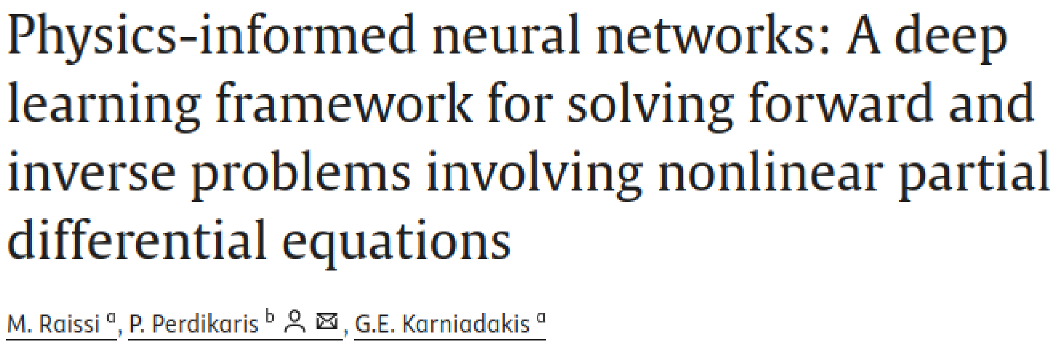
Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

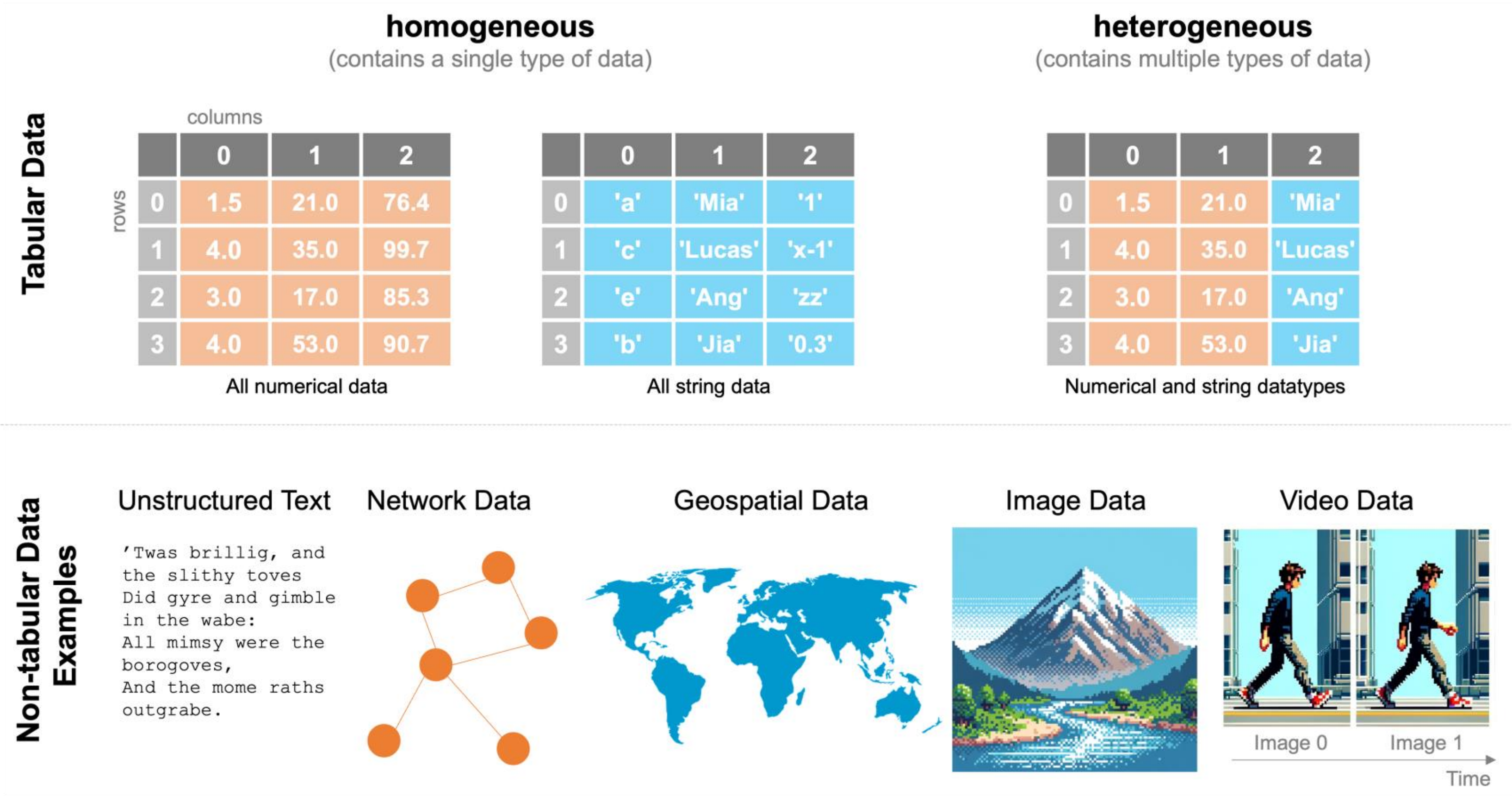
Illia Polosukhin* ‡
illia.polosukhin@gmail.com



Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations

M. Raissi [a], P. Perdikaris [b] ✉, G.E. Karniadakis [a]

Type of data


homogeneous
(contains a single type of data)
heterogeneous
(contains multiple types of data)
Tabular Data
columns
rows

	0	1	2
0	1.5	21.0	76.4
1	4.0	35.0	99.7
2	3.0	17.0	85.3
3	4.0	53.0	90.7

All numerical data

	0	1	2
0	'a'	'Mia'	'1'
1	'c'	'Lucas'	'x-1'
2	'e'	'Ang'	'zz'
3	'b'	'Jia'	'0.3'

All string data

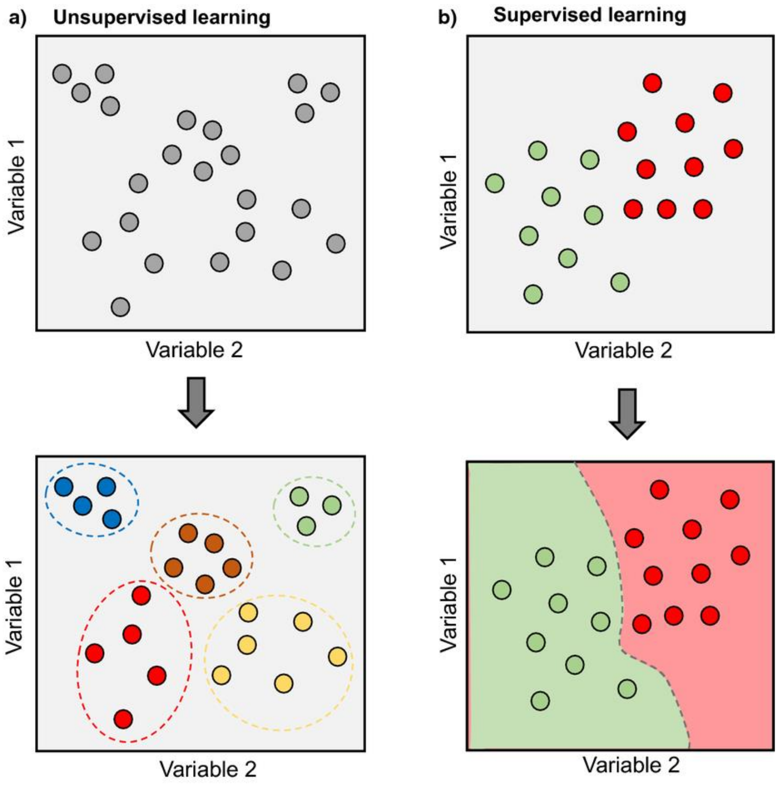
	0	1	2
0	1.5	21.0	'Mia'
1	4.0	35.0	'Lucas'
2	3.0	17.0	'Ang'
3	4.0	53.0	'Jia'

Numerical and string datatypes

```
'Twas brillig, and
the slithy toves
Did gyre and gimble
in the wabe:
All mimsy were the
borogoves,
And the mome raths
outgrabe.
```

https://www.practicaldatascience.org/notebooks/class_3/week_2/00_intro_to_pandas.html

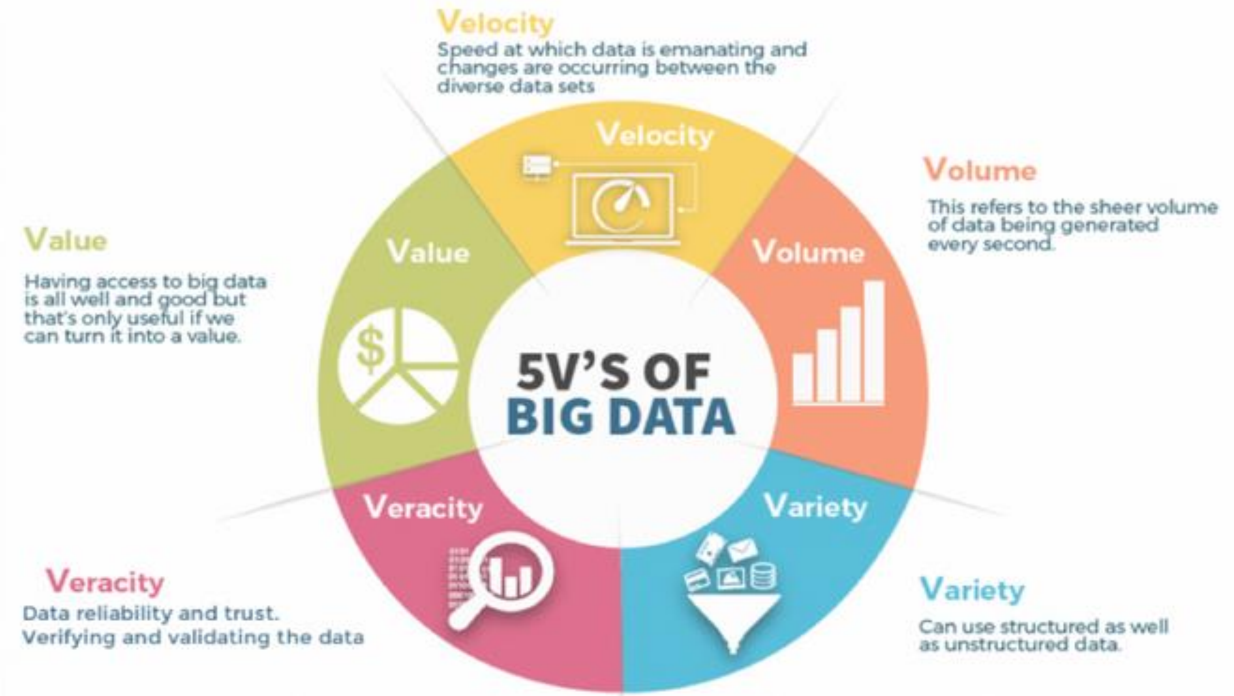
Supervised and unsupervised learning



Morimoto, Juliano & Ponton, Fleur. (2021). Virtual reality in biology: could we become virtual naturalists?

Big Data

https://www.devopsschool.com/blog/what-is-big-data-processing-tools-and-use-cases-of-big-data-processing-tools/



Velocity
Speed at which data is emanating and changes are occurring between the diverse data sets
Velocity
Volume
This refers to the sheer volume of data being generated every second.
Value
Having access to big data is all well and good but that's only useful if we can turn it into a value.
Value
Volume
5V'S OF BIG DATA
Veracity
Variety
Veracity
Data reliability and trust. Verifying and validating the data
Variety
Can use structured as well as unstructured data.

Big data is where parallel computing tools are needed to handle data

Fox, Data Science for Transport 2018

Hardware limitations

- On Leonardo, we have Lustre (open-source parallel filesystem) and the High Performance Storage is 5.7 PB (DDN Exascaler ES400NVX2)
- RAM of the compute node is 512 GiB

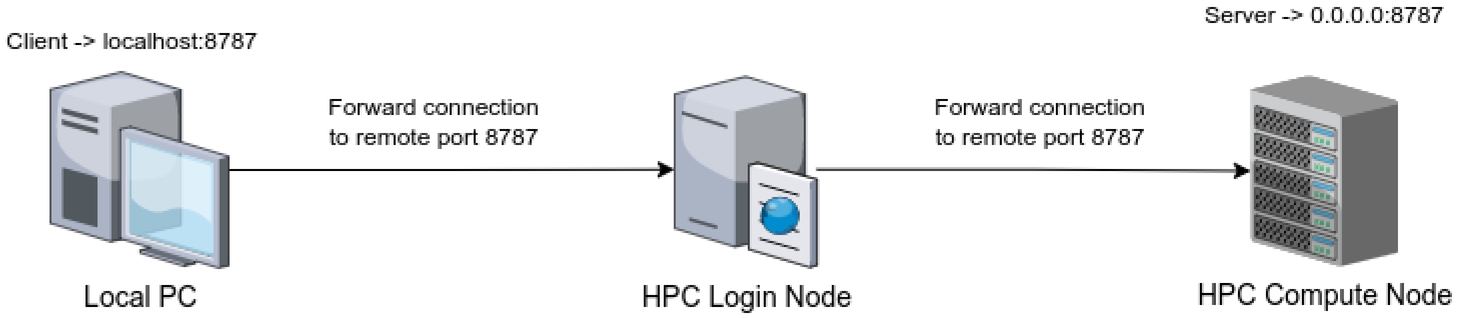
- Big data cannot fit in the memory of a computational node
- Most Pandas operations run on a single thread, while some scikit-learn estimators and utilities are parallelized using multi-cores CPU

Distributed DataFrames

- The idea is to use the memory of several computational nodes that share the filesystem and communicate via the network
- In addition, we parallelize the work on several multicore nodes
- The most popular frameworks for distributed computing are
  - Apache Spark
  - Ray
  - Dask

Distributed DataFrames on HPC

- Login nodes are the only nodes accessible from external networks.
- On Leonardo no connections from the computational nodes to the outside world are permitted.
- To use interactive tools like Jupyter Notebook using the client-server paradigm in HPC compute nodes:



Client -> localhost:8787

Forward connection to remote port 8787

Server -> 0.0.0.0:8787

Forward connection to remote port 8787

- Finally, we typically use SLURM as the job scheduler

Distributed DataFrames on the Coud

- By containerizing the model with Docker and orchestrating it with Kubernetes, we can ensure consistency, scalability, and reliability
- The key principles in modern MLOps workflows.
- Kubernetes clusters can deploy, manage, and scale AI and ML:
  - Infrastructure orchestration that supports GPUs for training at scale.
  - Flexible integration with distributed computing and data processing

What is Dask?

- Dask provides multi-core and distributed+parallel execution on larger-than-memory datasets
- Dask allow to work with larger datasets making it possible to parallelize computation
- Dask is in Python and scales NumPy, Pandas, and scikit-learn.
- Dask is a framework for parallelizing most Python objects.

When Dask?

- Dask is not helpul for small size datasets, it generates overhead. In this case, Pandas has typically better performances.

- Dask might be useful for medium size dataset as it allows to work in parallel on a local multi-core (or multi-GPU) node.

- Dask is essential for large datasets as Pandas, NumPy, and scikit-learn are not inherently built to operate on distributed datasets.

Dask overview

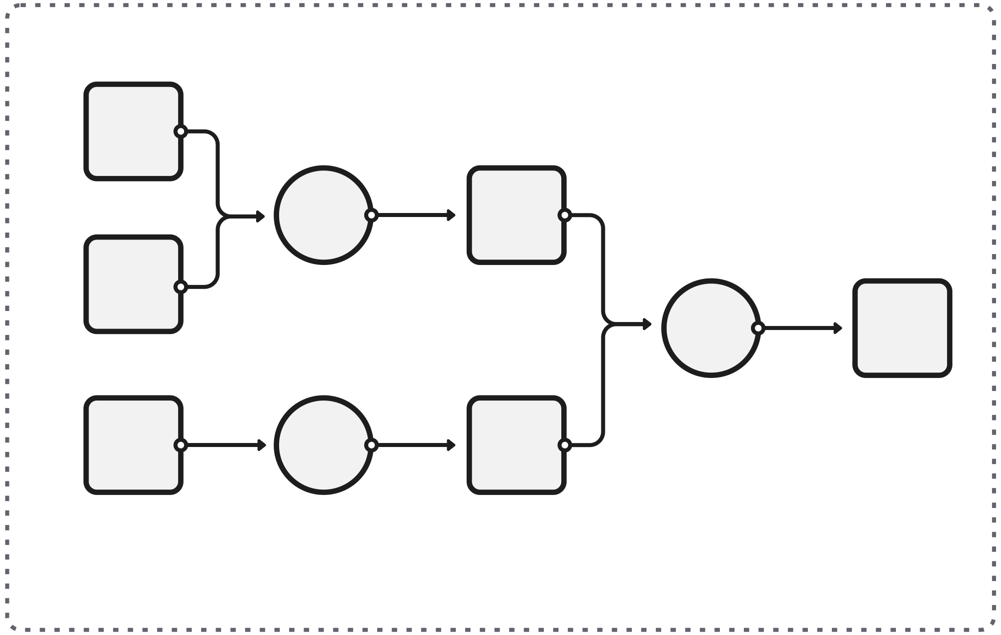
Collections

(create task graphs)

- Dask Array
- Dask DataFrame
- Dask Bag
- Dask Delayed
- Futures

→

Task Graph



→

Schedulers

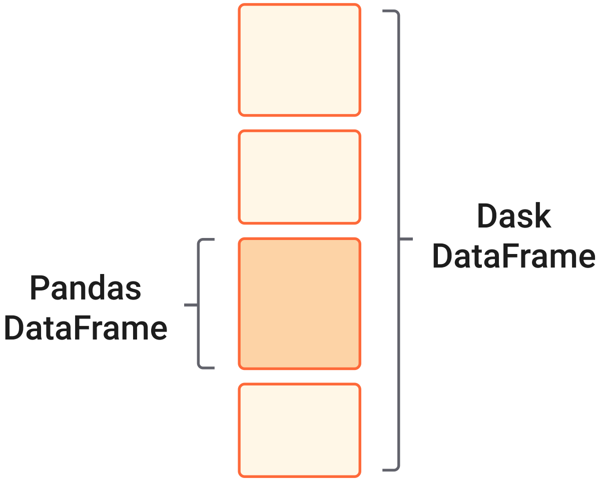
(execute task graphs)

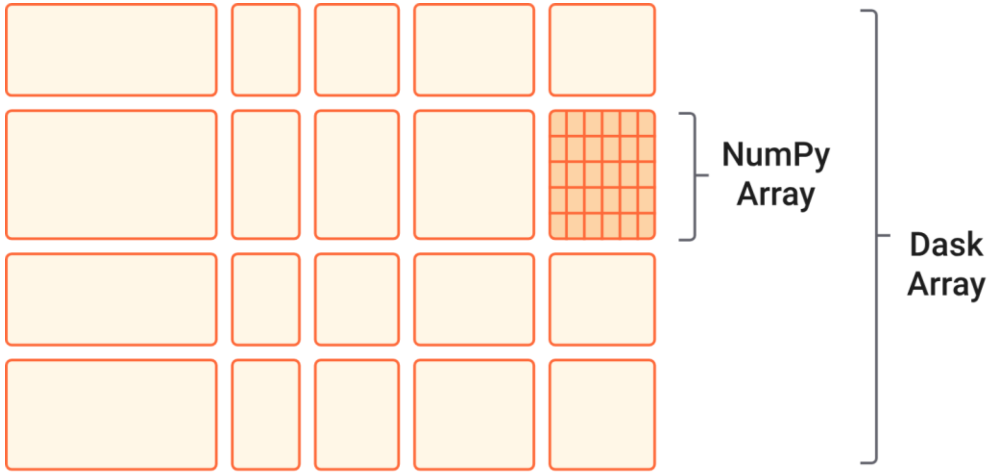
- Single-machine (threads, processes, synchronous)
- Distributed

High level collections are used to generate task graphs which can be executed by schedulers

Dask Collections

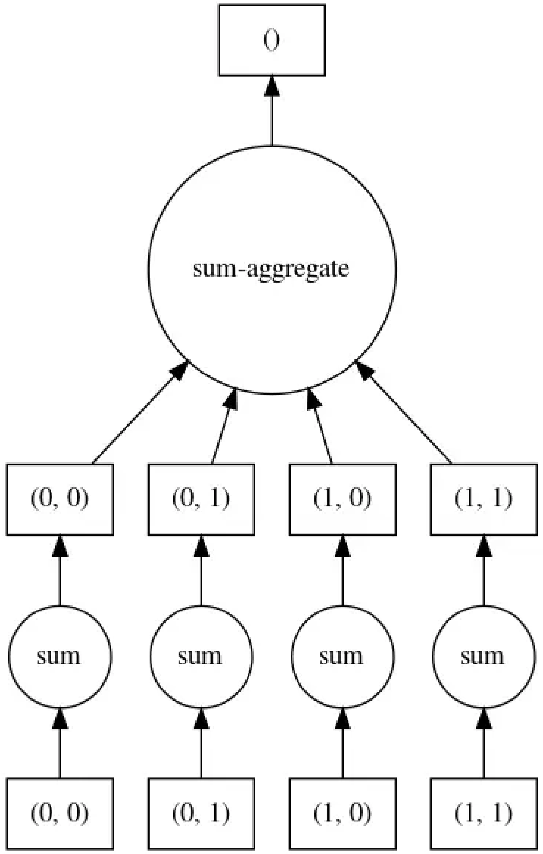
- Dask can be used to scale popular Python libraries such as Pandas and NumPy allowing to analyze dataset with greater size





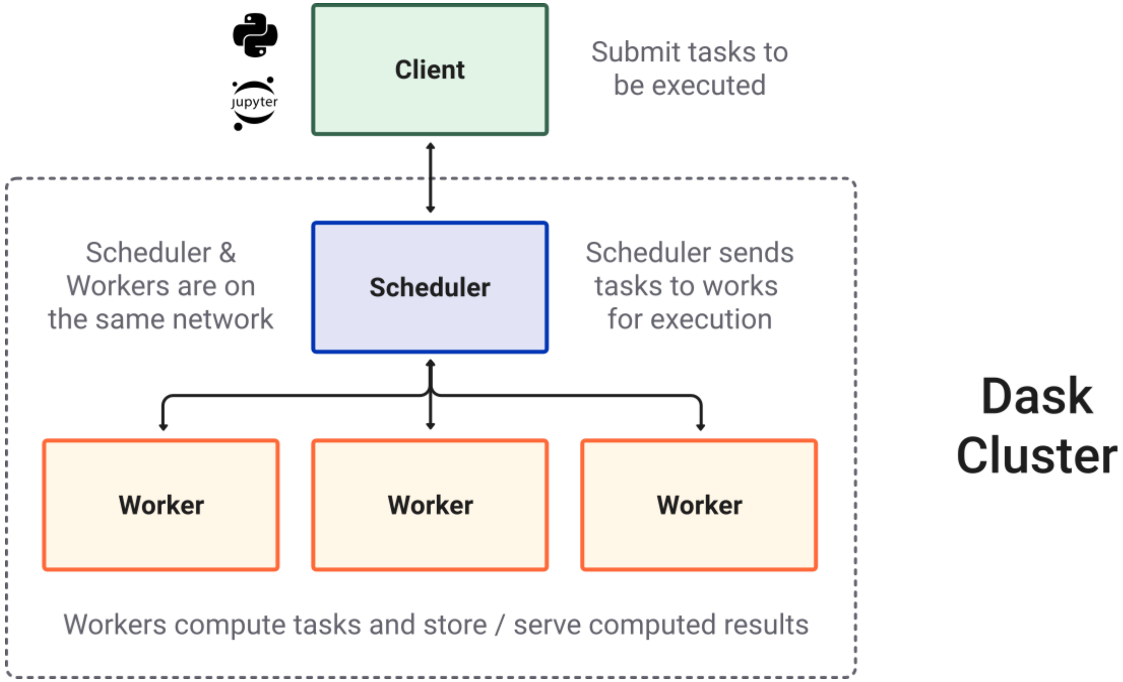
Task Directed Acyclical Graph (DAG)

- A graph is a representation of a set of objects that have a relationship with one another, consisted by:
  - node: a function, an object or an action
  - line: symbolize the relationship among nodes
- In DAG there is one logical way to traverse the graph. No node is visited twice.
- Dask uses DAG to coordinate execution of parallelized code across processors.
- Upstream actions are completed before downstream nodes.



Dask Cluster

- Usually, when using Dask, we will be using a distributed scheduler, which exists in the context of a Dask cluster.

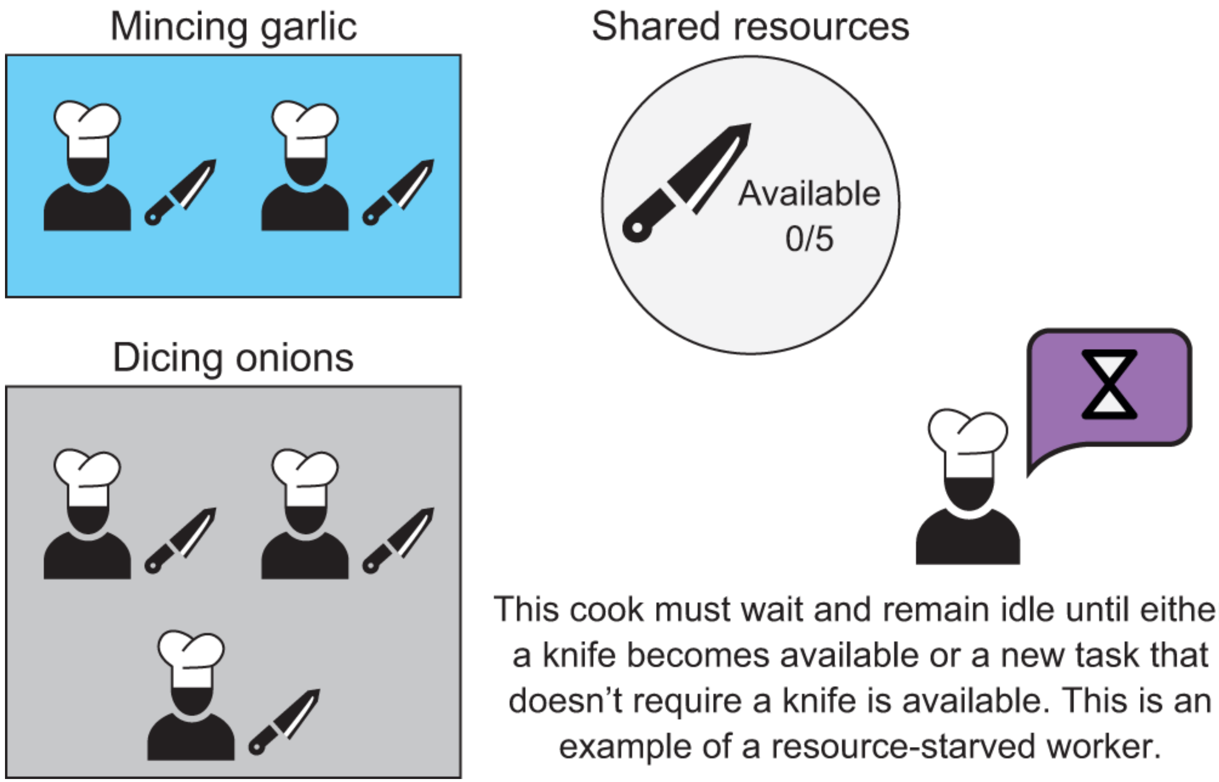

jupyter
Client
Submit tasks to be executed
Scheduler & Workers are on the same network
Scheduler
Scheduler sends tasks to works for execution
Worker
Worker
Worker
Workers compute tasks and store / serve computed results
Dask Cluster

Dask Scheduler and Workers

- Dask uses a central scheduler to orchestrate the work.
- Adding workers can improve performances of complex workloads, however, create overhead that can reduces gains.
- Workload will unlikely be perfectly balanced and shared equally among the workers.
- Some resources might be not fully exploited or might be idling because of insufficient shared resources (resource starvation).

Resources starvation

Scheduler needs to react to avoid bottlenecks



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

Adapted from
Data Science with Dask

The importance of being lazy

- Dask performs a lazy computation
- Until we run the method .compute(),
  Dask only splits the process into smaller logical pieces
- Even though the process is defined, the resources assigned and the place where the result will be stored are not assigned
- The scheduler assigns them dynamically,
  allowing to recover from possible worker failure
- In case of a failure, Dask reaches a node and repeat the action without disturbing the rest of the process.

More FLOPS/s from GPUs

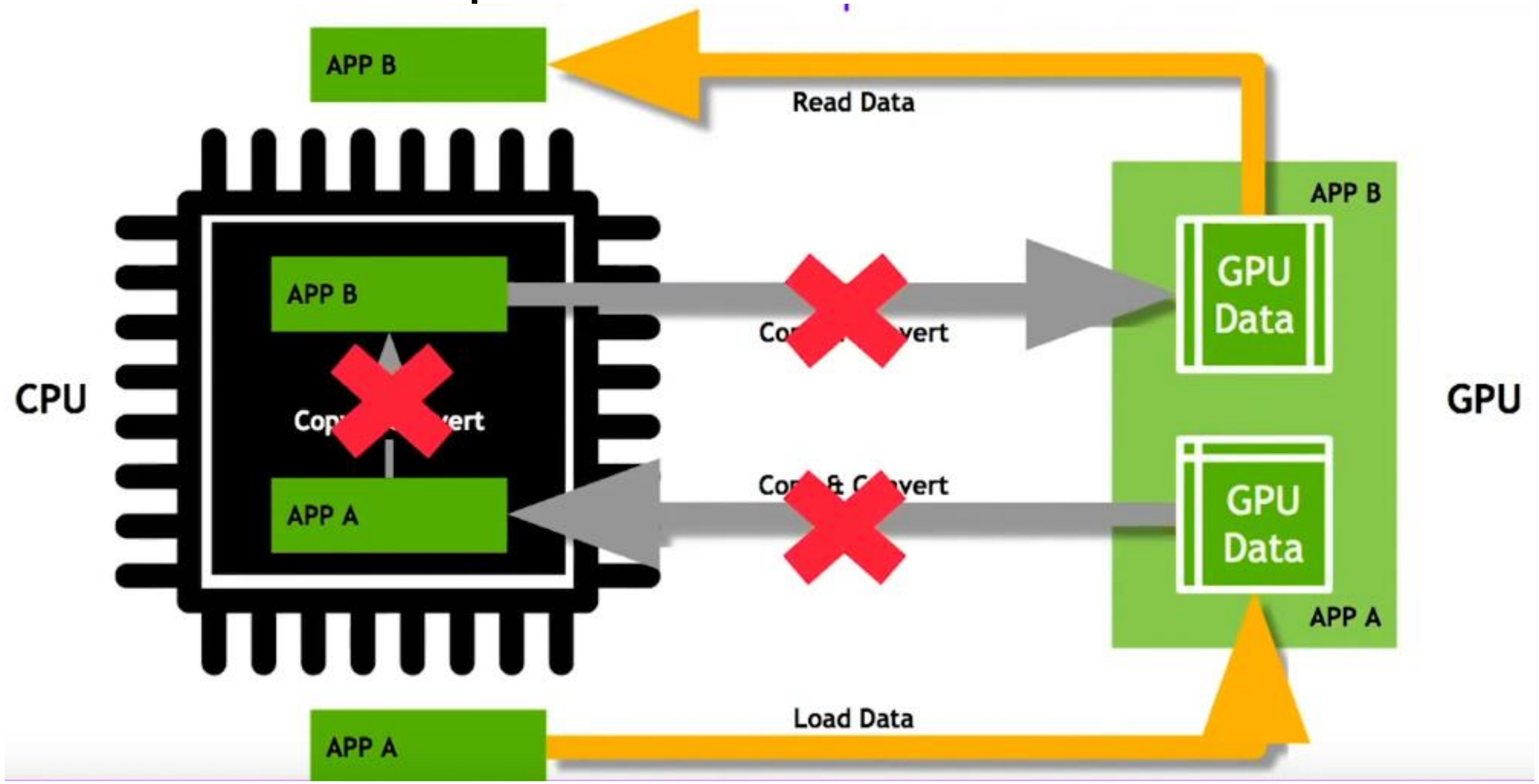
- CPUs are designed for more general-purpose tasks and have fewer, more powerful cores that excel at low-latency operations.
- GPUs are designed for highly parallel computations, with many-cores optimized for executing the same instructions on many data elements simultaneously (SIMT).

Source: NVIDIA

Data Movement and Transformation

- What if we could keep data on the GPU?

Source: NVIDIA RAPIDS

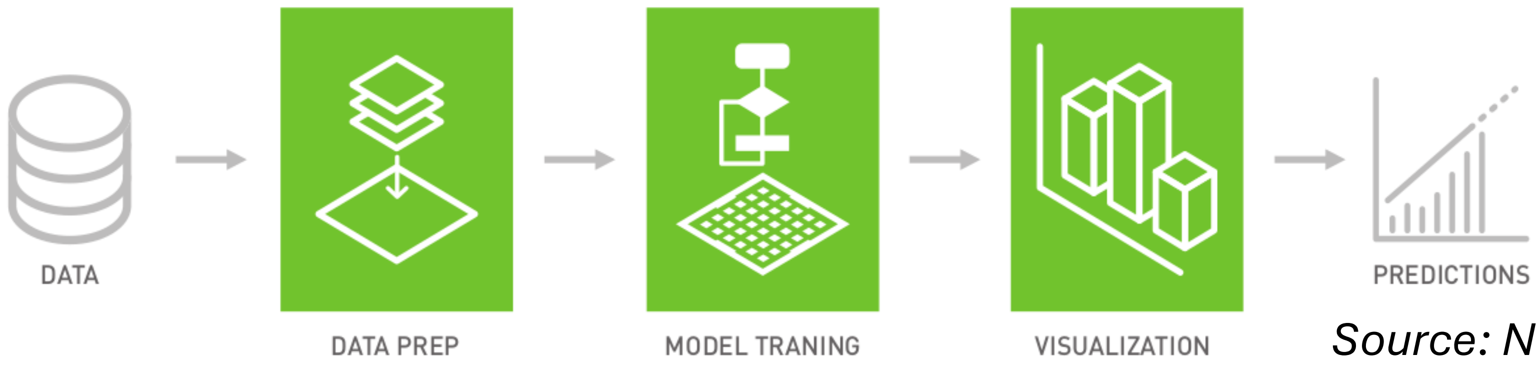


RAPIDS

RAPIDS by NVIDIA is a suite of open-source software libraries and APIs designed to accelerate all data science

- Data analysis, modelization and visualization entirely on GPUs
- Accelerated machine learning (K-means, PCA, XGBoost, ...)

Open Source, End-to-end GPU-accelerated Workflow Built On CUDA



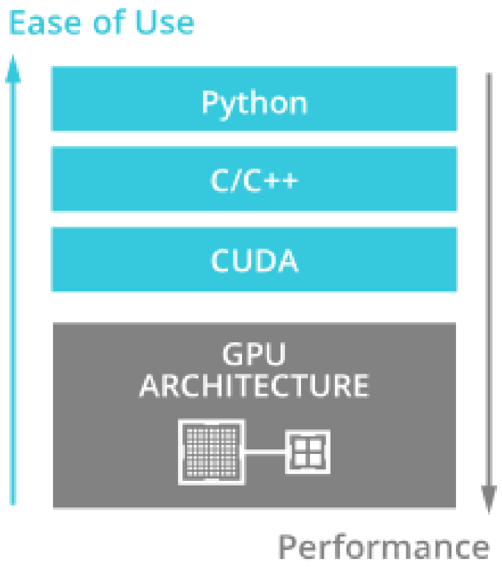
Source: NVIDIA RAPIDS

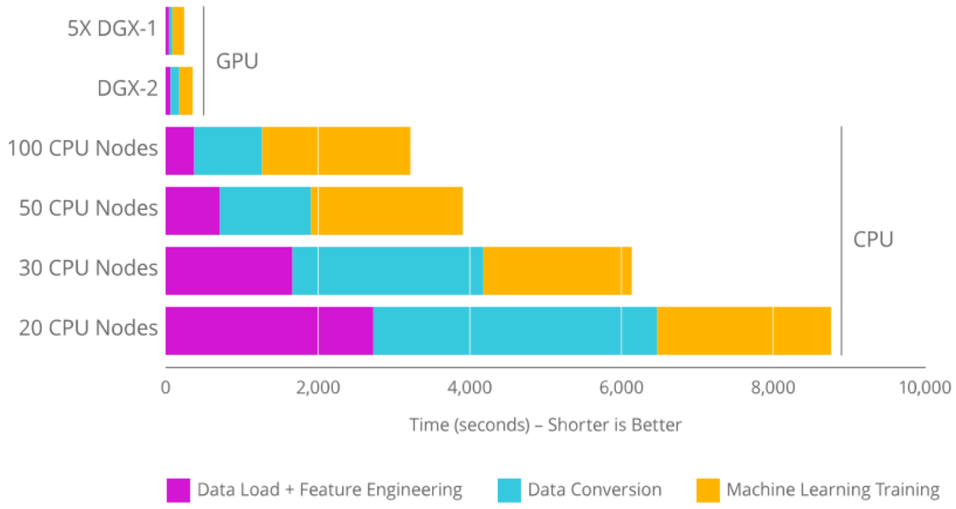
Why RAPIDS?

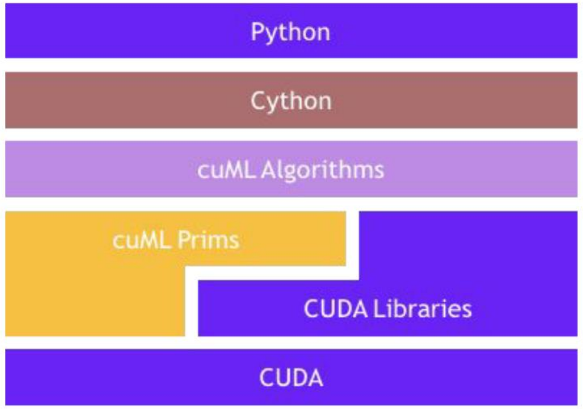
- **Speed**: RAPIDS leverages the power of GPUs to process data, often delivering performance gains of 10x or more compared to traditional CPU-based methods.
- **Familiarity**: Data scientists can continue using familiar tools and workflows, with RAPIDS offering drop-in replacements for many common Python data science libraries.
- **Ecosystem**: RAPIDS integrates with popular data science frameworks like Apache Spark and Dask, making it easy to incorporate GPU acceleration into your existing infrastructure.
- **Scale**: RAPIDS scales seamlessly to handle massive datasets, enabling you to tackle big data problems that were previously impractical or computationally expensive.

RAPIDS is easy and fast

- RAPIDS provides a similar interface to Python libraries like pandas, scikit-learn and NetworkX
- RAPIDS becomes advantageous, when the data being manipulated reaches multiple GiBs



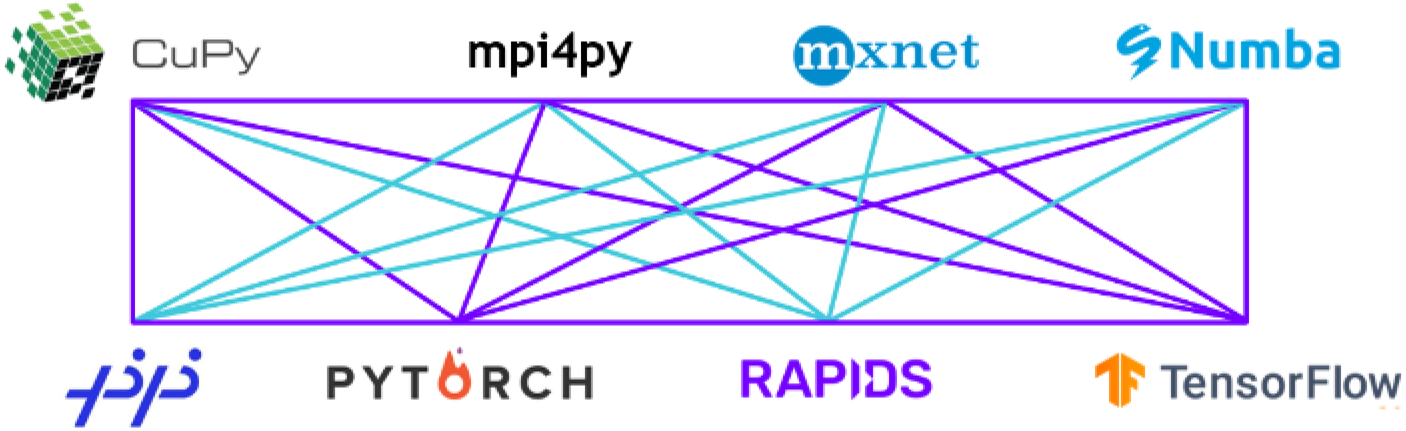




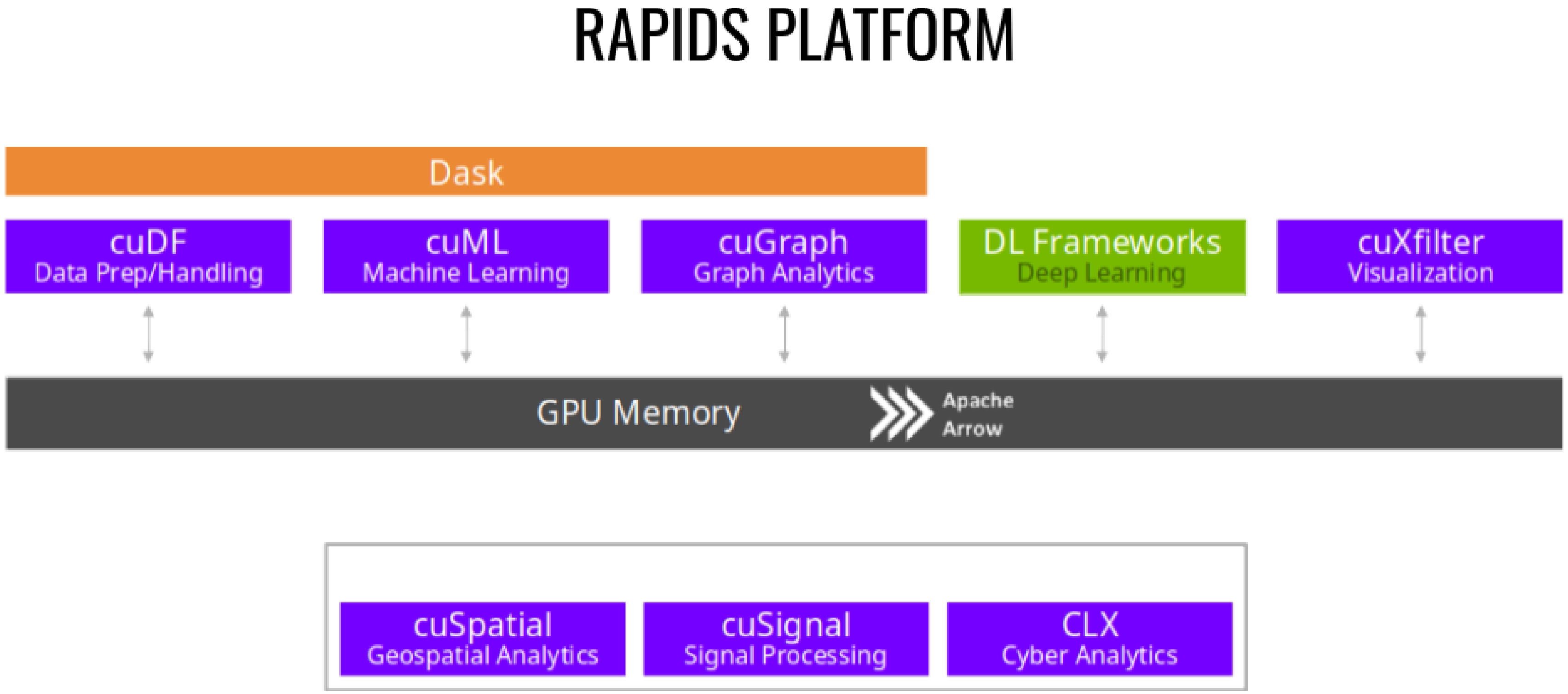
Source:
Documentation RAPIDS

RAPIDS Interoperability

- Real-world workflows often need to share data between libraries
- RAPIDS supports device memory sharing between data science and deep learning libraries
- Keeps data on the GPU avoids costly copies to host memory
- Any library that support __cuda_array_interface__ will allow for sharing of memory buffers with RAPIDS

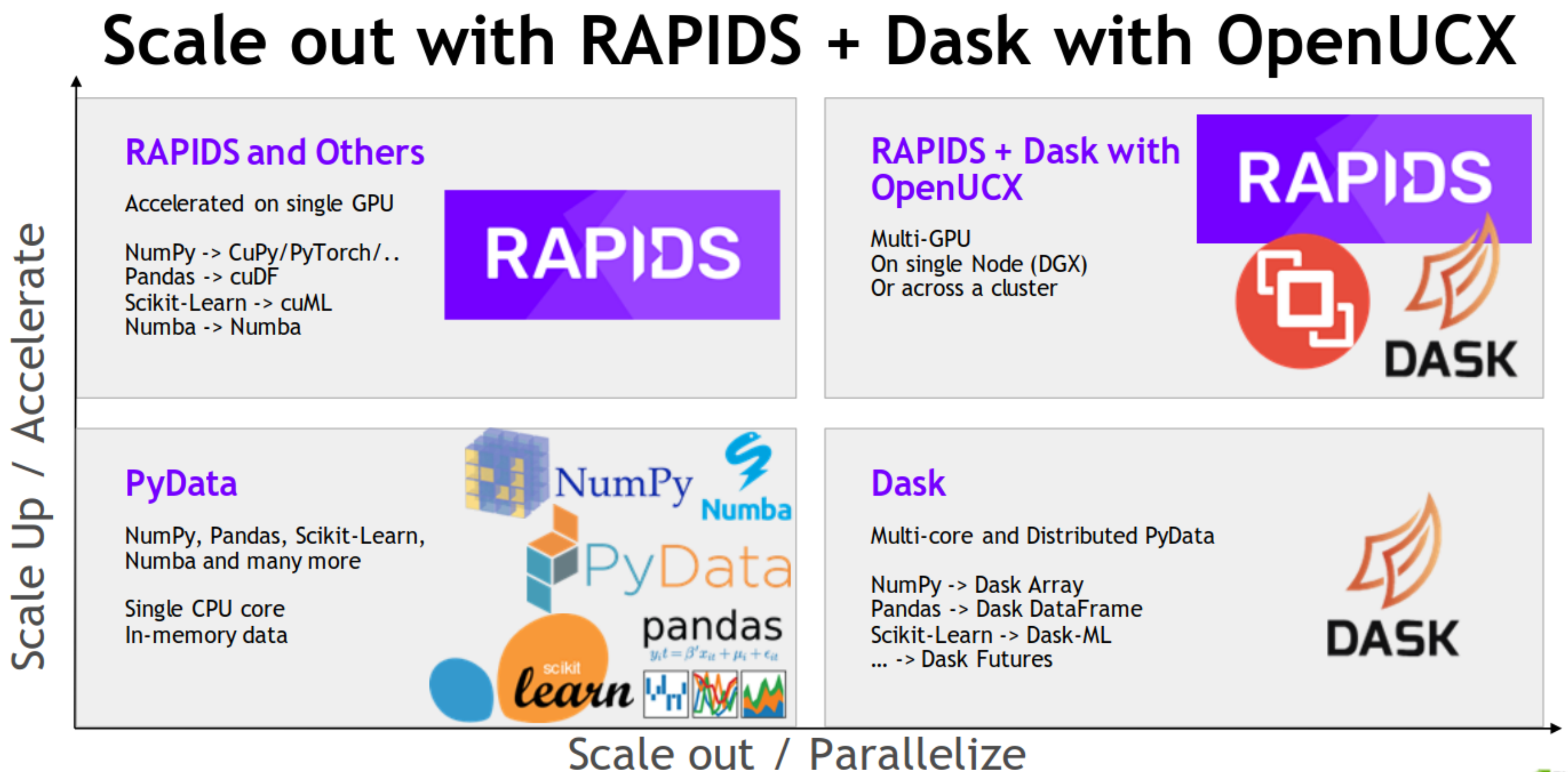


RAPIDS Platform



Source: Documentation RAPIDS

Pandas, Scikit-Learn, Dask, Rapids



Scale out with RAPIDS + Dask with OpenUCX

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

RAPIDS + Dask with OpenUCX

Multi-GPU
On single Node (DGX)
Or across a cluster

PyData

NumPy, Pandas, Scikit-Learn, Numba and many more

Single CPU core
In-memory data

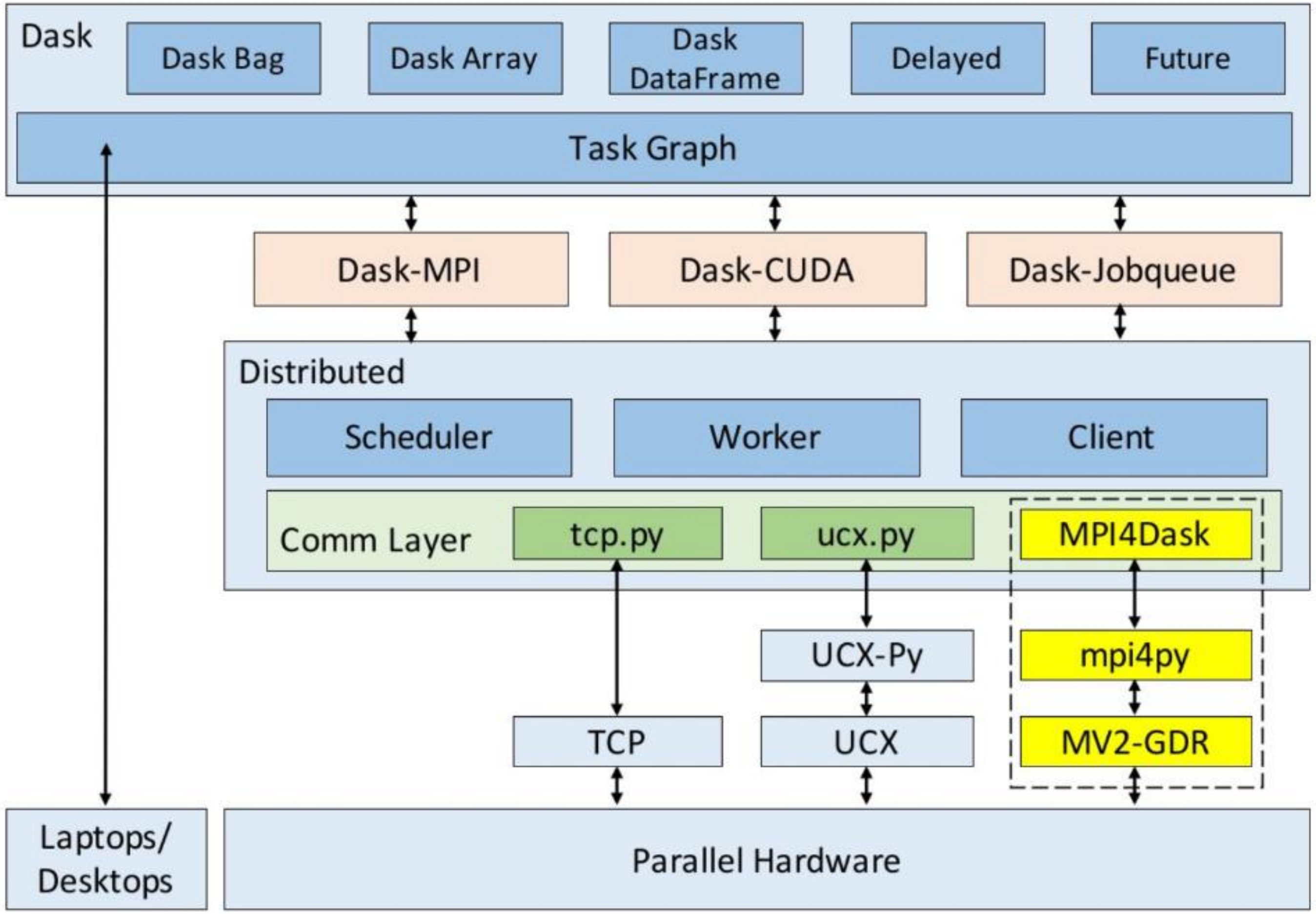
Dask

Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
... -> Dask Futures

Source: Documentation RAPIDS

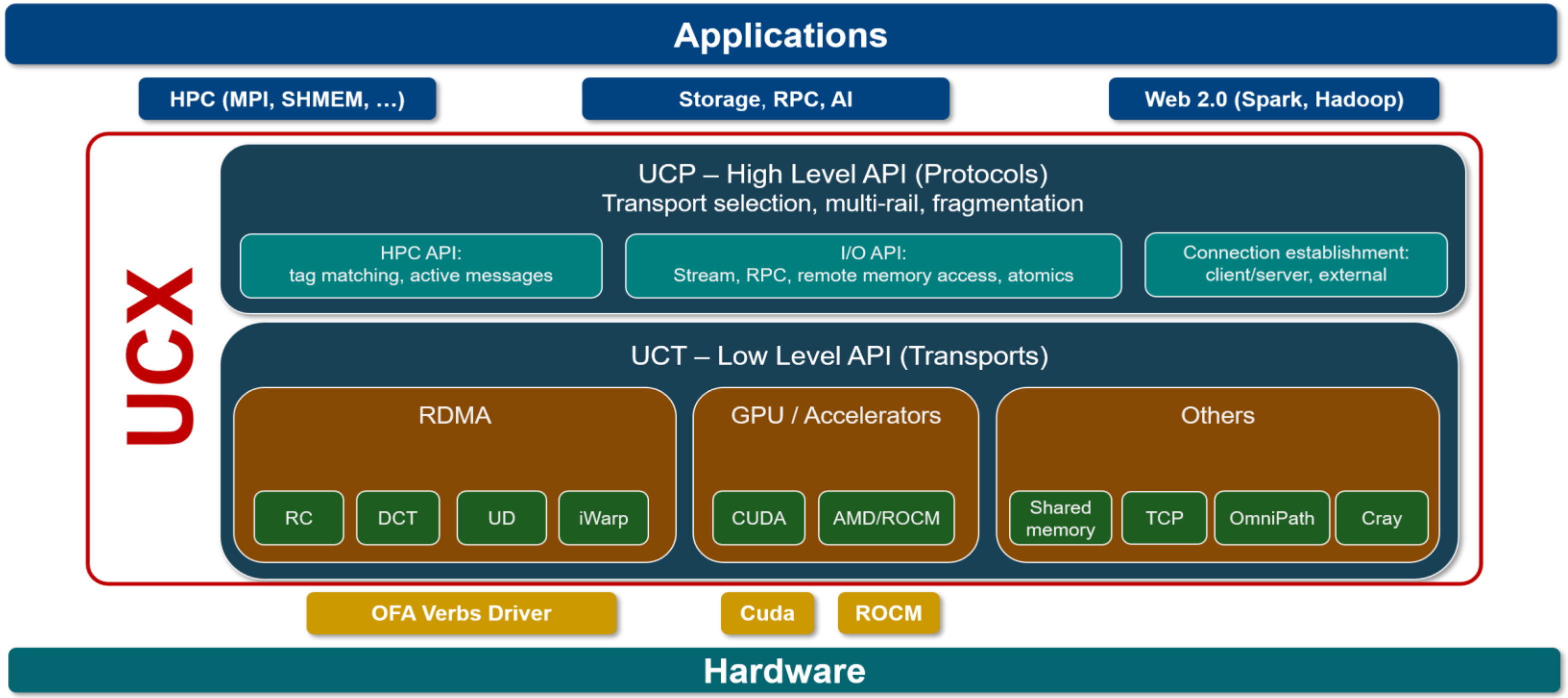
Dask Summary



Unified Communication X

UCX is an optimized communication framework for high-bandwidth and low-latency networks

UCX exposes abstract communication primitives that utilize the best of available hardware resources and offload, including RDMA, TCP, GPUs, shared memory, and atomics operations



Applications

HPC (MPI, SHMEM, …)

Storage, RPC, AI

Web 2.0 (Spark, Hadoop)

UCX

UCP – High Level API (Protocols)
Transport selection, multi-rail, fragmentation

HPC API:
tag matching, active messages

I/O API:
Stream, RPC, remote memory access, atomics

Connection establishment:
client/server, external

UCT – Low Level API (Transports)

RDMA

RC

DCT

UD

iWarp

GPU / Accelerators

CUDA

AMD/ROCM

Others

Shared memory

TCP

OmniPath

Cray

OFA Verbs Driver

Cuda

ROCM

Hardware

Dask-CUDA

- Dask allows users to create a cluster with scheduler and workers
- Each Dask-CUDA worker is attached to a dedicated GPU
- Dask client can connect to the cluster and distribute data on it
- Once the data has been distributed to the workers,
  the workers can start the distributed operations
- Dask-CUDA is an addition to Dask that allows the creation of GPU clusters while optionally configuring networking via the high-performance UCX-Py

Dask-CUDA: GPU Memory Management

- When using Dask-CUDA it’s best to use an RMM pool to pre-allocate memory on the GPU.
- One can easily make hundreds of thousand or even millions of allocations in trivial workflows causing significant performance degradations.
- With an RMM pool, allocations are sub-sampled from a larger pool and this greatly reduces the allocation time and thereby increases performance.
- For Multi-GPU node we can simply create a LocalCUDACluster
-
- cluster = LocalCUDACluster(CUDA_VISIBLE_DEVICES="0,1", protocol="ucx",
-                          rmm_pool_size="30GB")
-

Dask-CUDA: GPU Memory Management

- We also recommend allocating most, though not all, of the GPU memory space.
  We do this because the CUDA Context takes a non-zero amount (typically 200-500 MBs) of GPU RAM on the device.

- `rmm_async: bool, default False` uses the underlying cudaMallocAsync memory allocator, which greatly reduces memory fragmentation at a minor to negligible performance cost.

- `protocol="ucx", rmm_pool_size and rmm_async can also be set by the command line interface when using dask-cuda-worker and dask scheduler:`

```
dask-cuda-worker --protocol ucx --rmm-async --rmm-pool-size=30GB &
```

Dask-CUDA: Spilling from Device

- Moving data from device to host (spilling) can be implemented generally, but often it comes at the expense of performance.

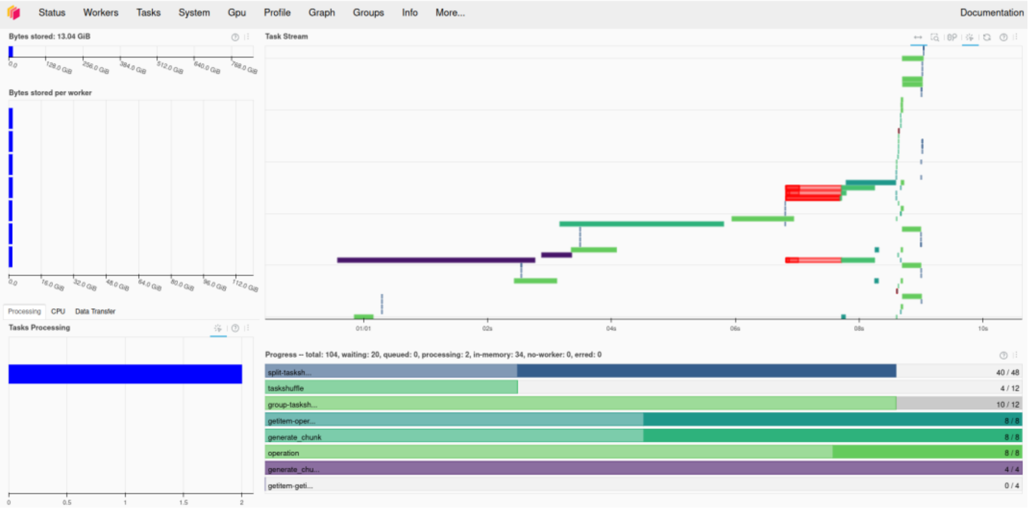
- Dask-CUDA and cuDF have several spilling mechanisms: `device-memory-limit`, `memory-limit`, `jit-unspill`, `enable-cudf–spill`.

- `enable-cudf-spill` enables cuDF's internal spilling mechanism that will move data (cuDF buffers) from device to host if objects require more memory than is available on the GPU.

- For tabular-based workloads, using `enable-cudf-spill` is often faster and more stable compared with the other Dask-CUDA options

Dask Dashboard Overview

- The Dask Dashboard is a web-based monitoring tool that provides real-time insights into the performance and status of a Dask cluster
- By default, when starting a scheduler on your local machine the dashboard will be served at `http://localhost:8787/status`. You can type this address into your browser to access the dashboard, but may be directed elsewhere if port 8787 is taken. You can also configure the address using the `dashboard_address` parameter

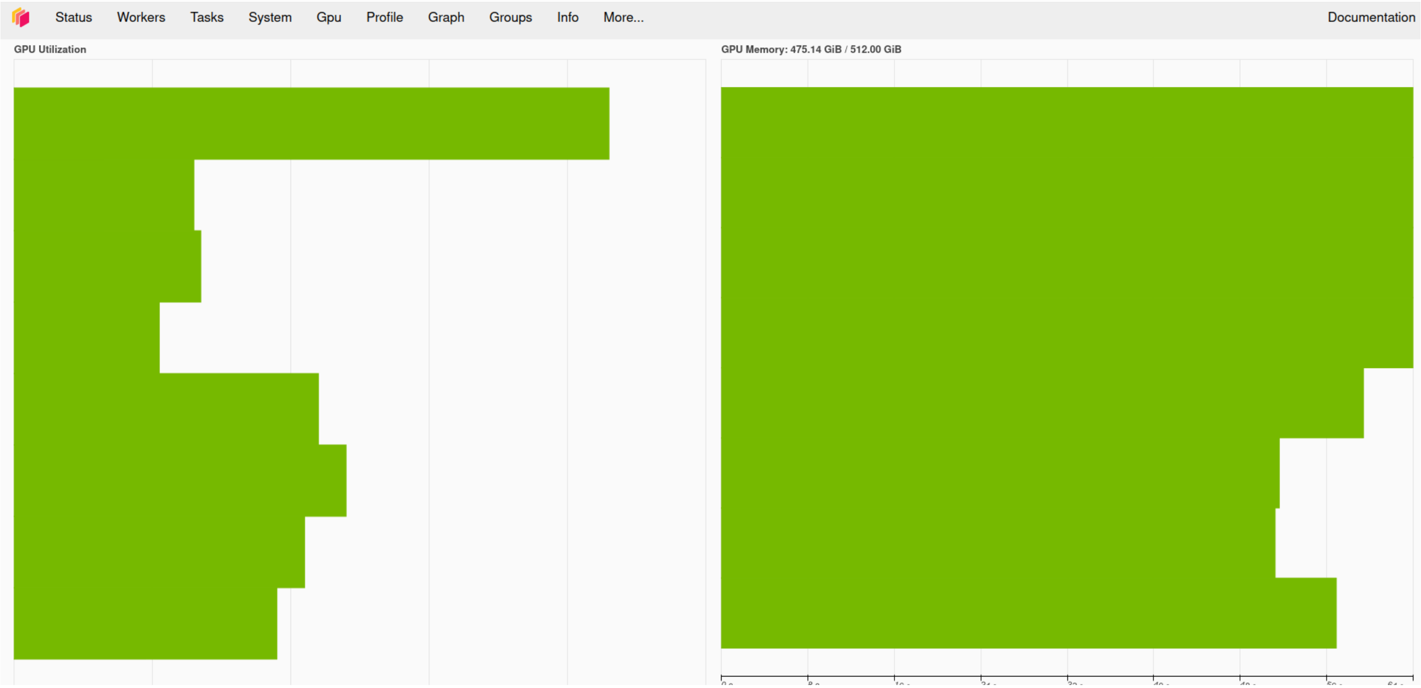


Progress -- total: 104, waiting: 20, queued: 0, processing: 2, in-memory: 34, no-worker: 0, erred: 0

split-tasksh...	40 / 48
taskshuffle	4 / 12
group-tasksh...	10 / 12
getitem-oper...	8 / 8
generate_chunk	8 / 8
operation	8 / 8
generate_chu...	4 / 4
getitem-geti...	0 / 4

Dask Dashboard Overview

- The Dask Dashboard helps users understand how tasks are distributed, how resources are utilized, and where potential bottlenecks exist
- The dashboard is useful for debugging, performance tuning, and optimization



Install RAPIDS on Leonardo

- **pip Issues:** Infiniband is not supported yet.
- **Docker Issues:** On Leonardo, we use Apptainer and not Docker
- **Conda Issues:**
  If the Conda is older than `22.11`, update to the latest version.
  This will include libmamba, a Mamba-powered Conda solver to significantly accelerate environment solving.
  If the Conda installation is version `22.11` or newer, run:

  ```
  $> conda install -n base conda-libmamba-solver
  $> conda create --solver=libmamba ...
  ```
- RAPIDS can be used with any conda distribution e.g. miniforge
- Also available as module: module load rapids/2023.12