A Python package for distributed ROOT RDataFrame analysis

Vincenzo Eduardo Padulano, Enric Tejedor

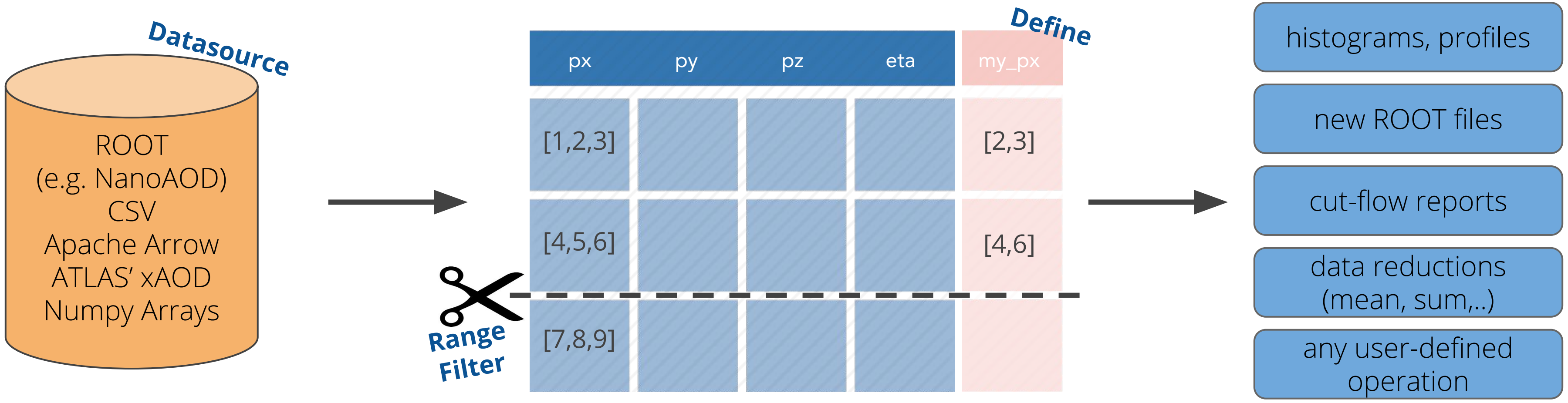
PyHEP 2021

ROOT
Data Analysis Framework
https://root.cern

Outline

1. RDataFrame: ROOT's declarative interface for analysis
2. Distributed Analysis with RDataFrame
   a. Programming model
   b. Backends
3. Demo on SWAN
   a. Dimuon analysis with RDataFrame
   b. Local, Spark and Dask: execution and monitoring
4. Performance tests
5. Summary

RDataFrame analysis interface

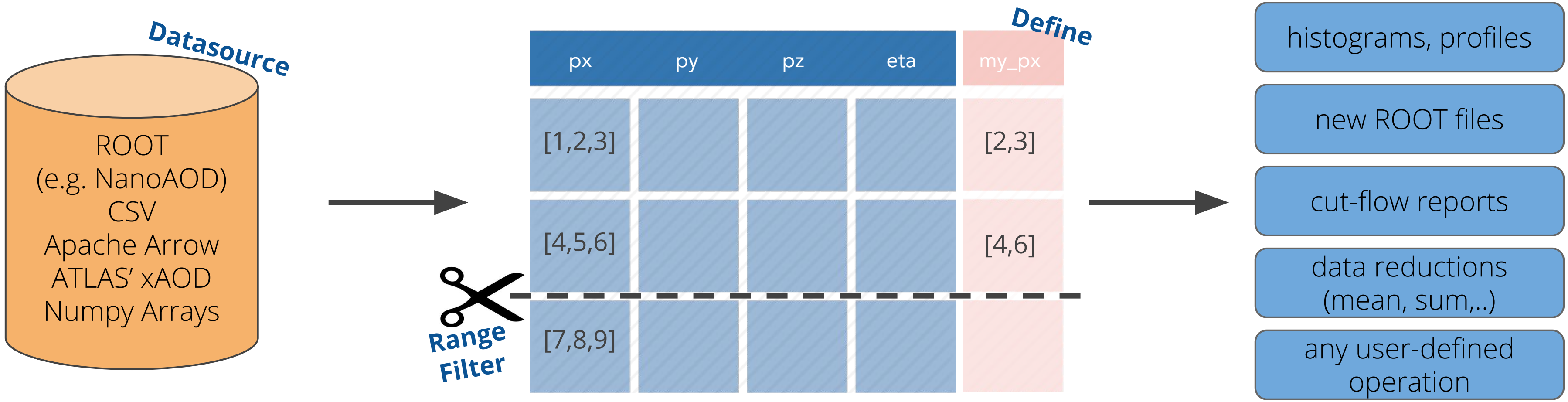



```
# enable multi-threading
ROOT.EnableImplicitMT()
df = ROOT.RDataFrame(dataset)
```

```
df = df.Range(2)
        .Define("my_px", "px[eta > 0]")
```

```
# filled in a single pass
h1 = df.Histo1D("my_px", "w")
h2 = df.Histo1D("px", "w")
```

RDataFrame analysis interface



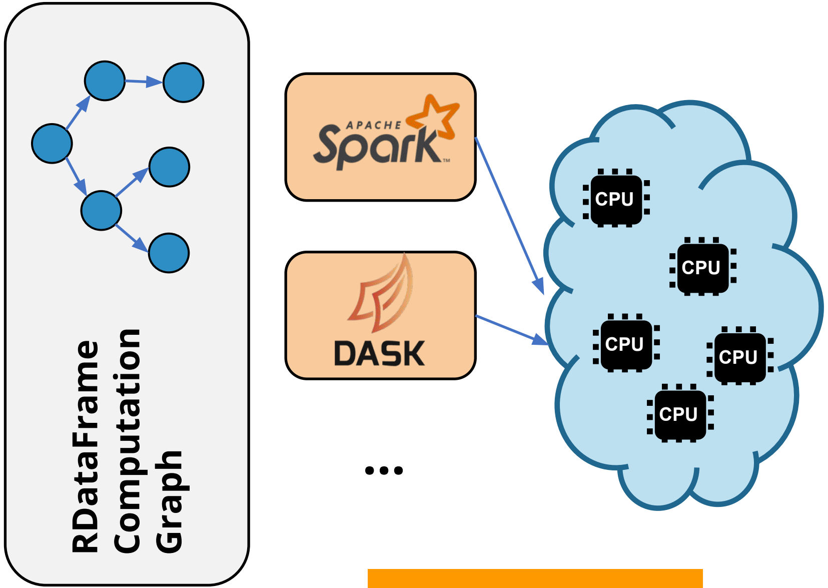
Goal

Best performance and ease of use across the board,
for most (all?) HEP analysis use cases

[RDataFrame reference guide](#) [RDataFrame tutorials](#)

Going Distributed

- Enable **interactive large-scale distributed** data analysis with RDataFrame
- Can run with different schedulers
  - **Spark**
  - **Dask**
  - **...**
- Analysis from start to end in a **single interface**



New in 6.24 (experimental)

Programming Model

Local | **Distributed**

Importing RDataFrame

```python
from ROOT import RDataFrame
```

```python
import ROOT
RDataFrame = \
ROOT.RDF.Experimental.Distributed.Dask.RDataFrame
```

Constructing RDataFrame

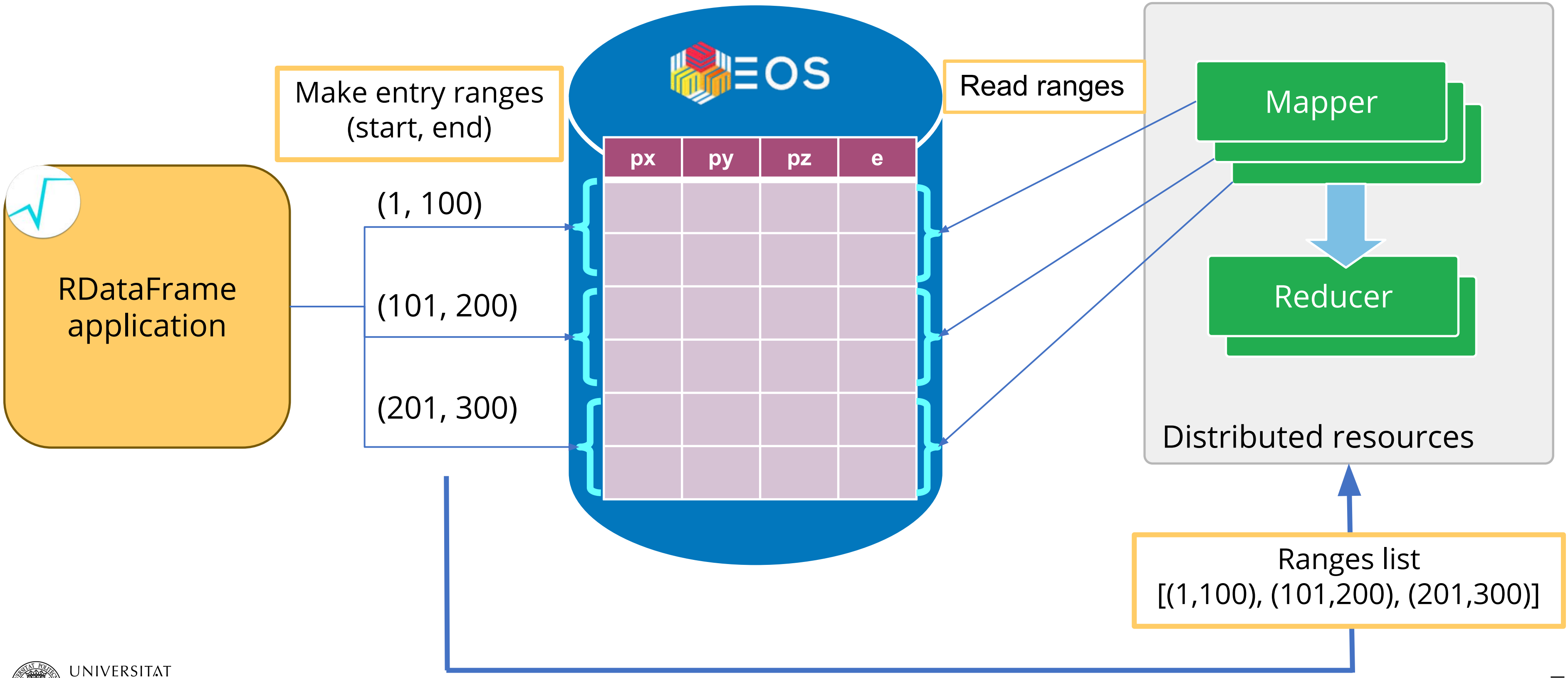
```python
df = RDataFrame('treename', 'filename.root')
```

```python
from dask.distributed import Client
df = RDataFrame('treename', 'filename.root',
      daskclient = Client('tcp://hostname:port'))
```

Rest of application

```python
df2 = df.Filter(...).Define(...)
h1 = df2.Histo1D(...)
h1.Draw()
```

Behind distributed RDataFrame

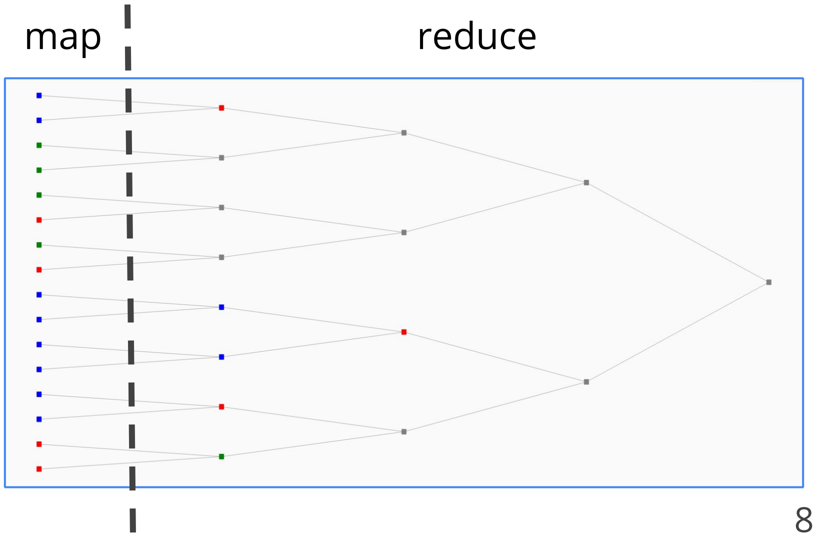


UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Backends

Dask vs Spark for distributed RDataFrame:

- `dask.delayed` vs `spark.map` & `spark.treeReduce`
- Same MapReduce tree pattern, different API
- Spark processes in stages
  - first map, then reduce
- Dask can reduce two completed mappers while others are still being processed



Distributed C++

- RDataFrame interfaces (in Python) with schedulers such as Dask or Spark for task distribution
- RDataFrame tasks run mostly in **C++** via PyROOT
  - I/O and event loop execute in C++

Demo

Notebooks on GitHub

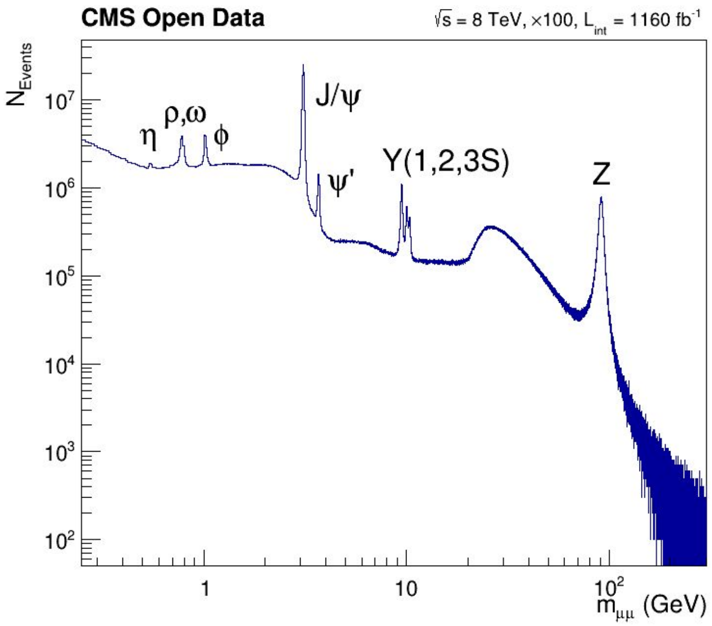
Test setup: cluster

Cluster of CERN OpenStack VMs:

- 13 VMs
  - 4 cores, 7.3 GB RAM, 40 GB spinning disk
- 1 reserved for the Dask scheduler / Spark master
- 12 Dask/Spark workers → 48 cores, 88 GB RAM

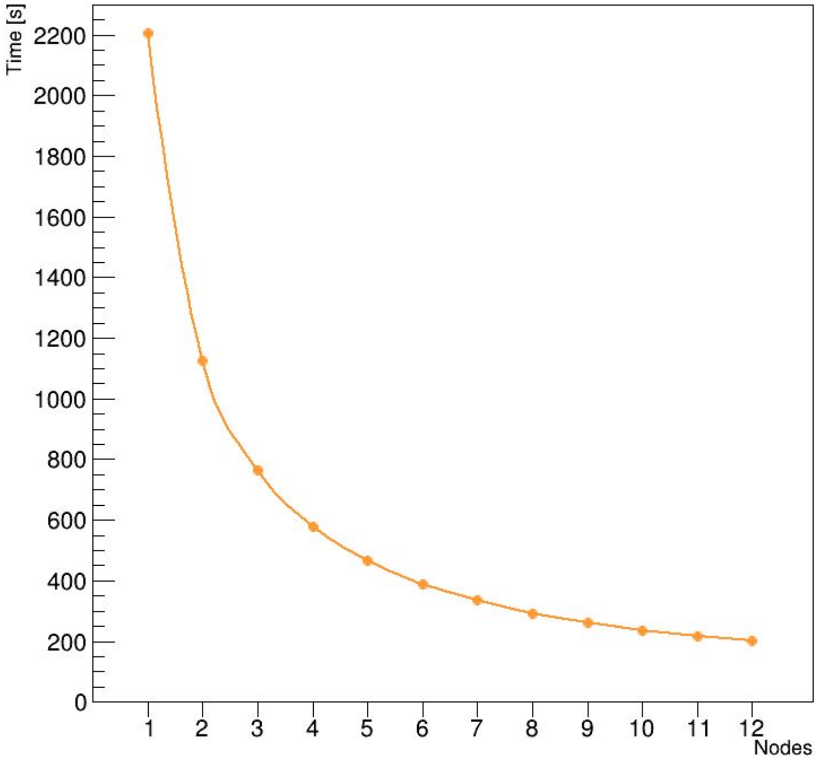
Test setup: dataset

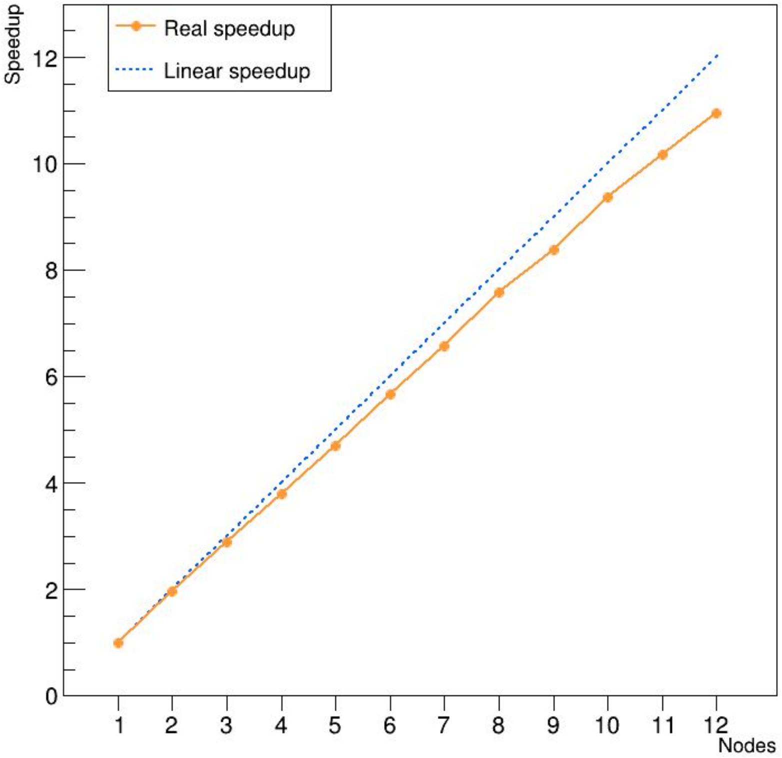
- Dimuon analysis
- 100x original dataset
  - 210 GB
  - ZLIB compressed
- All data is read and processed in the analysis



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

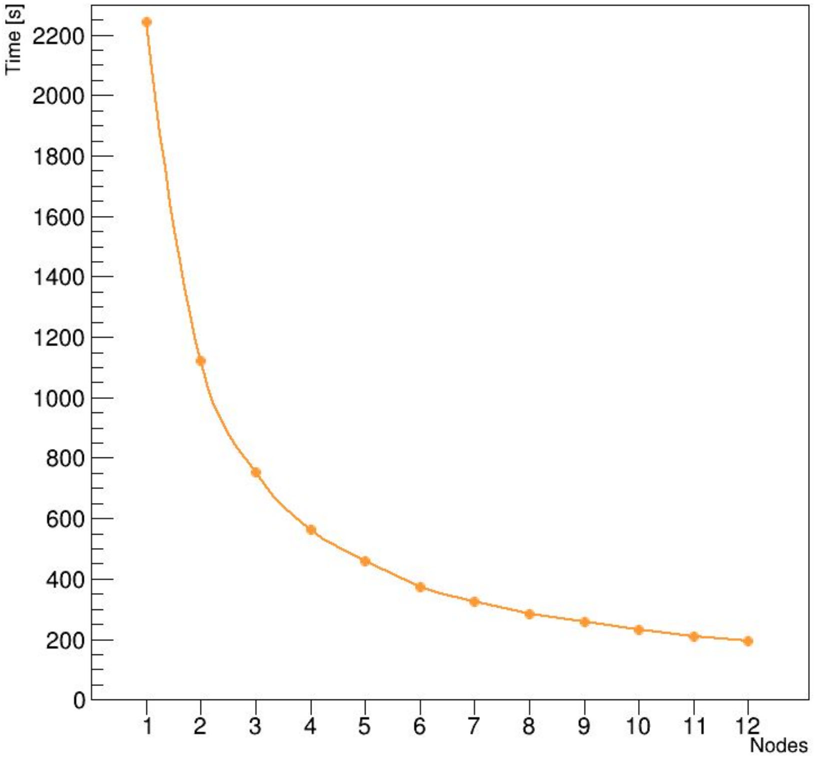
Dimuon analysis performance (Dask)

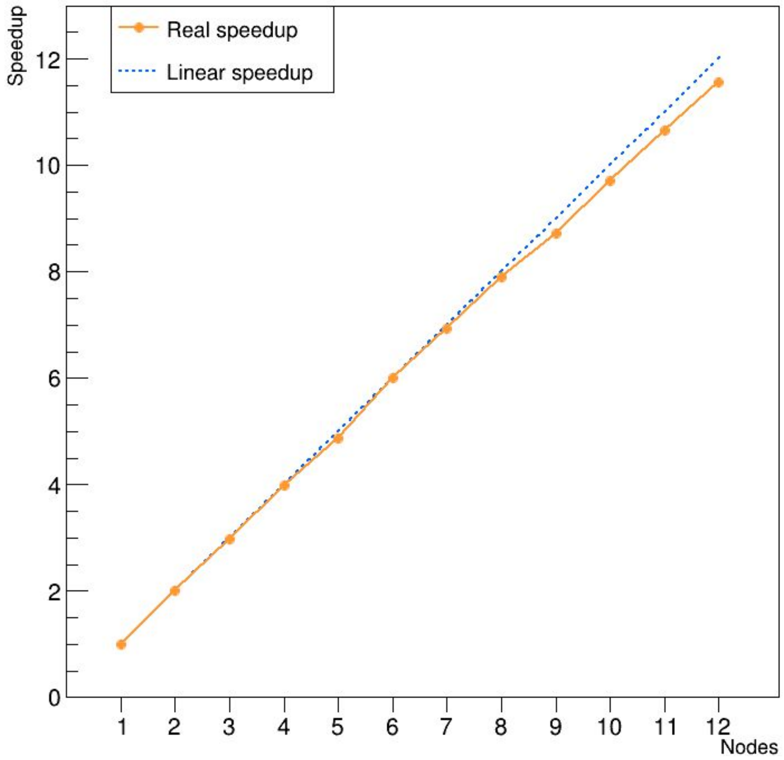




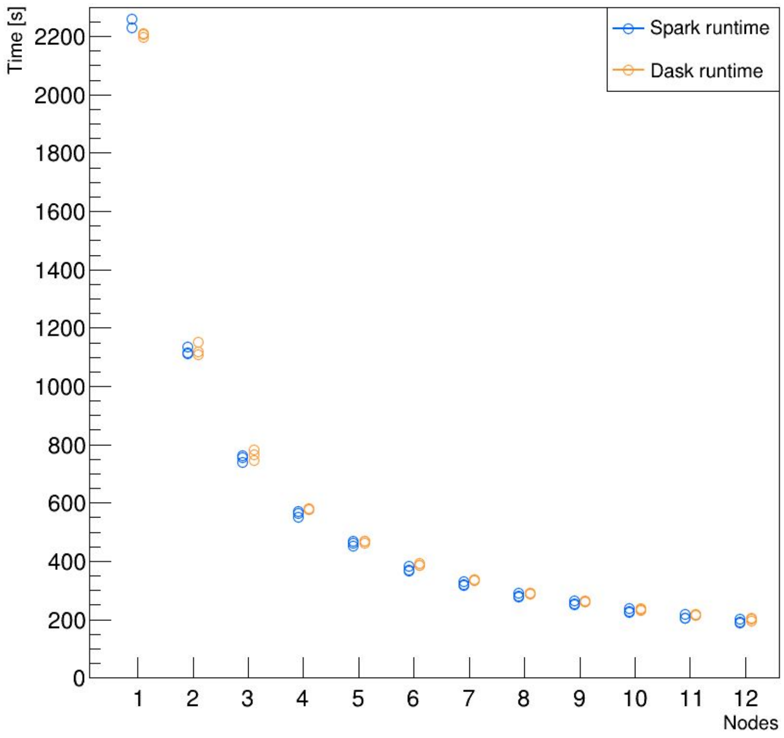
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dimuon analysis performance (Spark)





Spark vs Dask comparison



Summary

- RDataFrame: ROOT's declarative interface for data analysis
- Since ROOT 6.24: also **distributed**!
  - Requires minimal changes to original (local) application
  - Task scheduling with Spark or Dask (for now!)
- Useful links
  - RDataFrame reference guide
  - Tutorial: distributed RDataFrame with Spark