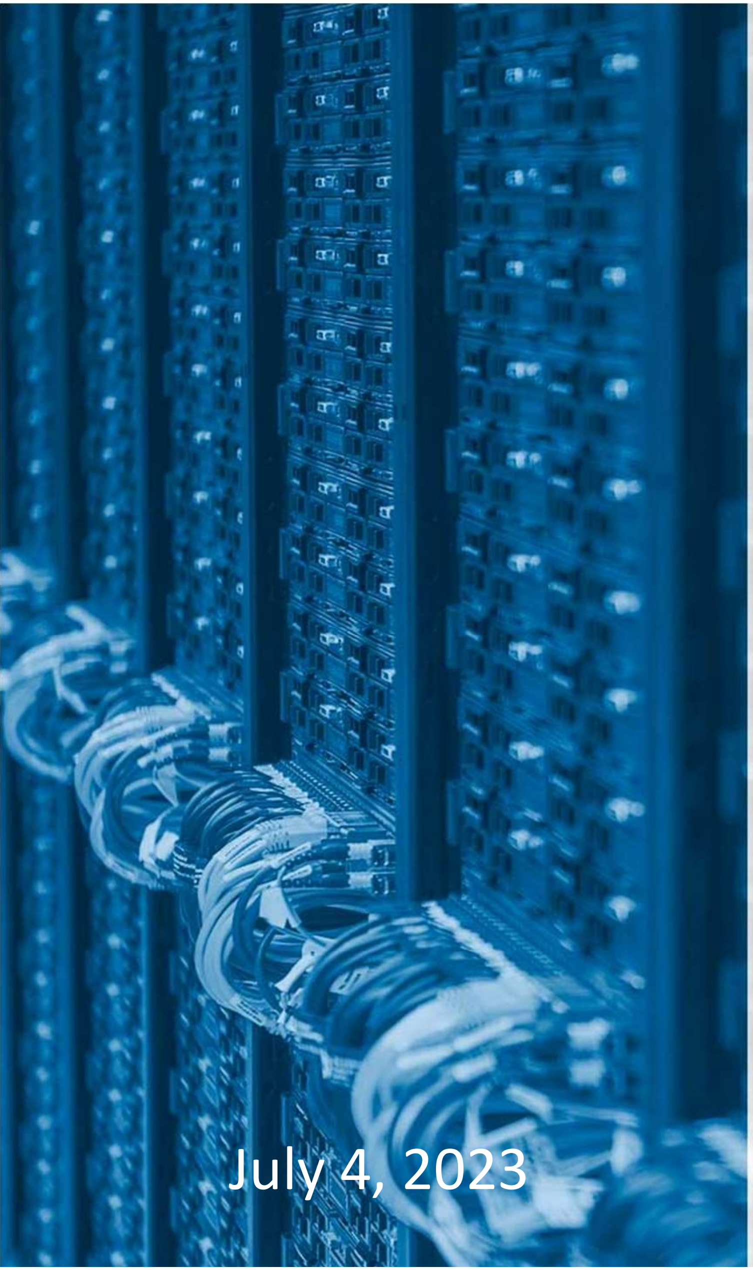


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Leveraging HPC techniques for data analytics

Anna Queralt, Francesc Lordan, Alex Barceló

Workflows and Distributed Computing Group

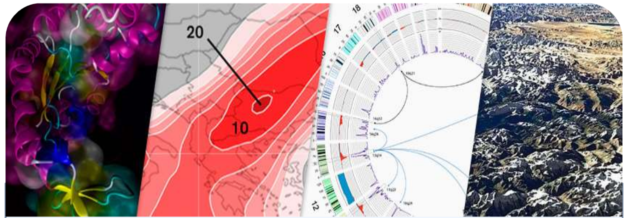
eBISS 2023

This project has received funding from the European Union’s Horizon 2020 JTI-EuroHPC research and innovation programme H2020-JTI-EuroHPC-2019-1, under grant agreements No: 955558 — eFlows4HPC, 956748 — ADMIRE

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

BSC-CNS objectives

Supercomputing services
to Spanish and EU researchers



R&D in Computer, Life, Earth and
Engineering Sciences

PhD programme, technology
transfer, public engagement

BSC-CNS is a consortium that includes

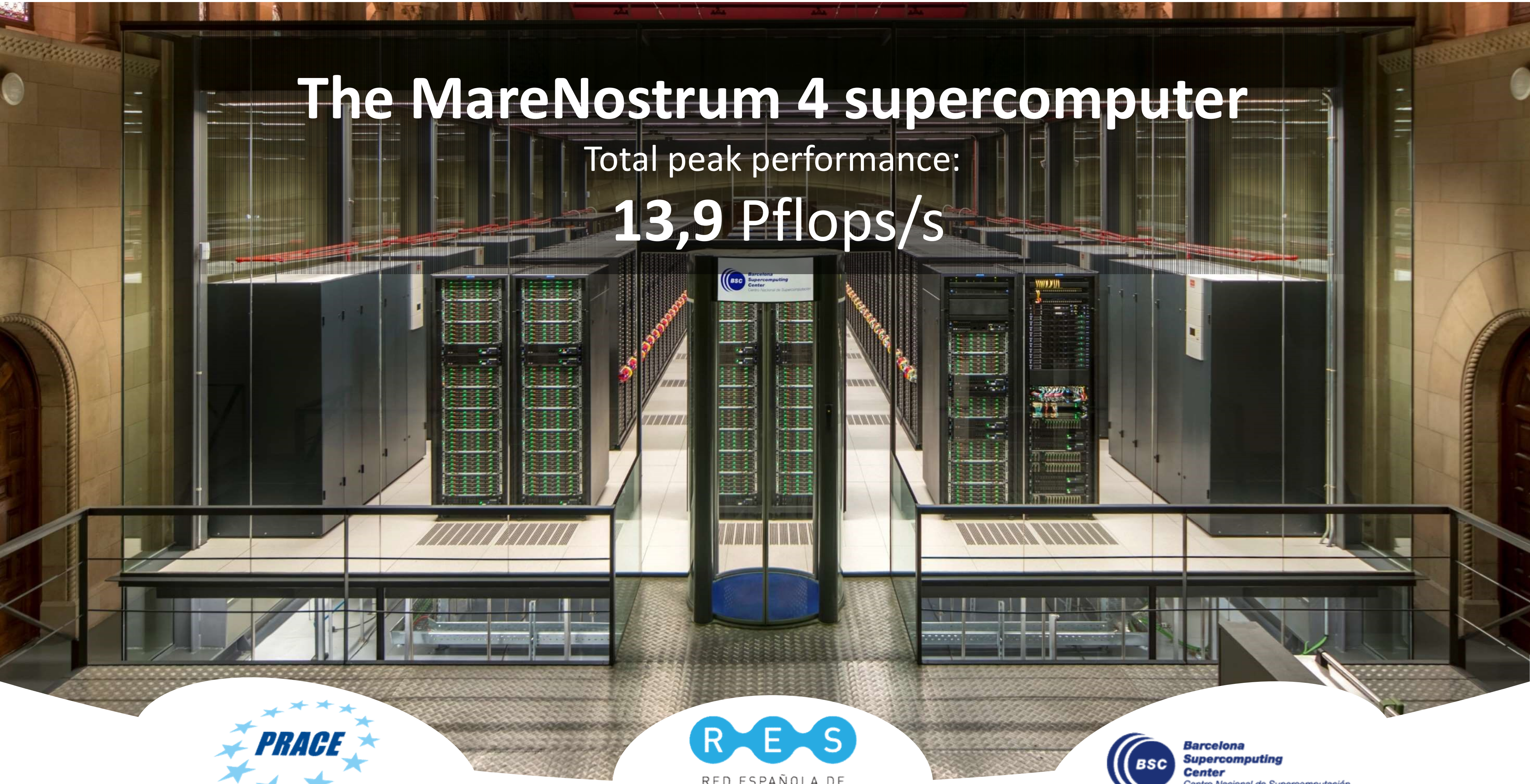
Spanish Government	60%	Gobierno de España, Ministerio de Ciencia, Innovación y Universidades
Catalan Government	30%	Generalitat de Catalunya Departament d'Empresa i Coneixement
Univ. Politècnica de Catalunya (UPC)	10%	Universitat Politècnica de Catalunya BarcelonaTech

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

The MareNostrum 4 supercomputer

Total peak performance:

13,9 Pflops/s



PRACE

Access: prace-ri.eu/hpc-access

RES

RED ESPAÑOLA DE SUPERCOMPUTACIÓN

Access: bsc.es/res-intranet

Barcelona Supercomputing Center

Centro Nacional de Supercomputación

Computer Sciences

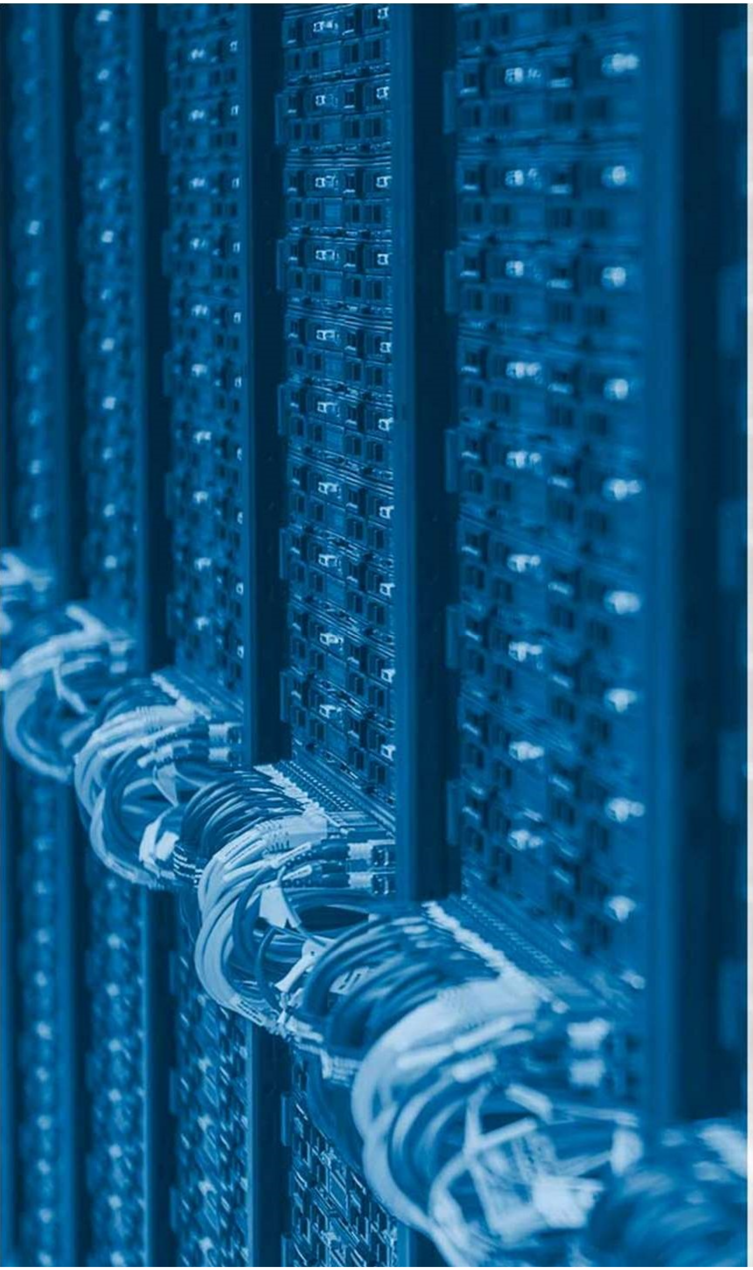
Influence the way machines are

- **BUILT** → **Holistic Computer Architecture Research**
- **PROGRAMMED** → **20 years innovating in Programming Models**
- **USED** → **Performance Analytics Tools: From Data to Insight**

In the context of

- Race towards Exascale
- HPC and Big Data convergence
- Artificial Intelligence era
- Multiple architectures for different applications domains

Barcelona Supercomputing Center
Centro Nacional de Supercomputación



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Leveraging HPC techniques for data analytics

Outline

- **Motivation**
  - BSC vision on data analytics
- **Distributed computing platform: COMPSs**
  - COMPSs overview
  - Demo

Coffee Break

- **Distributed data management platform: dataClay**
  - dataClay overview
  - Demo
- **COMPSs and dataClay for data analytics**
  - ML application
  - DL application

Until mid 2000's…

Relational databases and data warehouses

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Then…

- 2005: Facebook, Youtube
- 2006: Twitter, Amazon Web Services
- 2007: Netflix, iPhone
- 2008: Spotify
- 2009: Whatsapp, Instagram
- …

New data processing and management solutions are required:

- Complex analytics workflows including ML/DL
- More flexible (and efficient) data models
- Distribution and parallelism (clusters)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Photo by Sara Kurfeß in Unsplash

Now…

- Some platforms have been widely adopted in the data analytics community

hadoop · APACHE Spark · cassandra · MongoDB

- In the meantime, in HPC…

MPI · dask · IBM GPFS · daos

- Main differences/disadvantages:
  - Data analytics: “requires” memory, data movements
  - HPC: steep learning curve


Photo by Alexander Schimmeck in Unsplash

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

The best of both worlds

- From data analytics
  - Programmability
  - Abstraction of the infrastructure

- From HPC
  - Performance and scalability
  - Flexibility

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Photo by Ahtziri Lagarde in Unsplash

Achieving flexibility

…And programmability at the same time

	Spark	COMPSs
Algorithms	MLlib	dislib
Operators	Spark operators	DDS
Parallelism	Pre-defined tasks	Arbitrary tasks
Scheduling	Spark runtime	COMPSs runtime
Data types	RDD/DataFrame	Arbitrary types

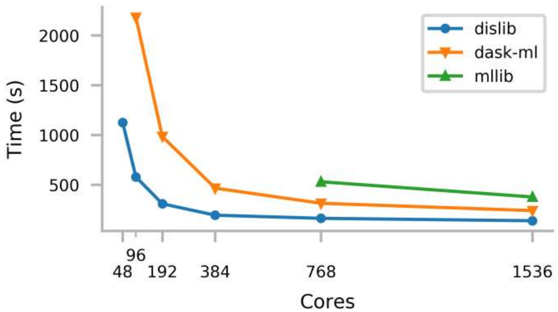
dataClay


Photo by Tim Mossholder in Unsplash

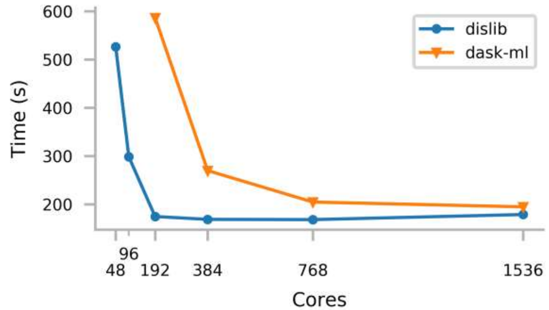
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

… and performance and scalability

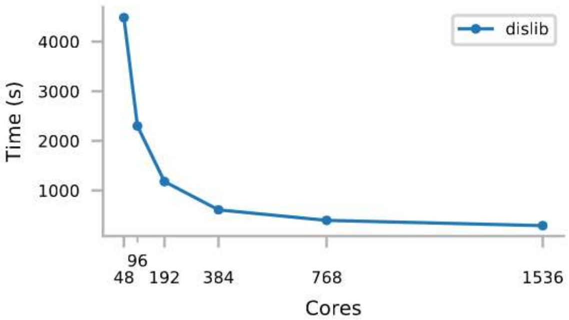
- K-means clustering
  - For very large sizes, MLlib and Dask-ML fail to finish the execution



500 million samples, 100 features

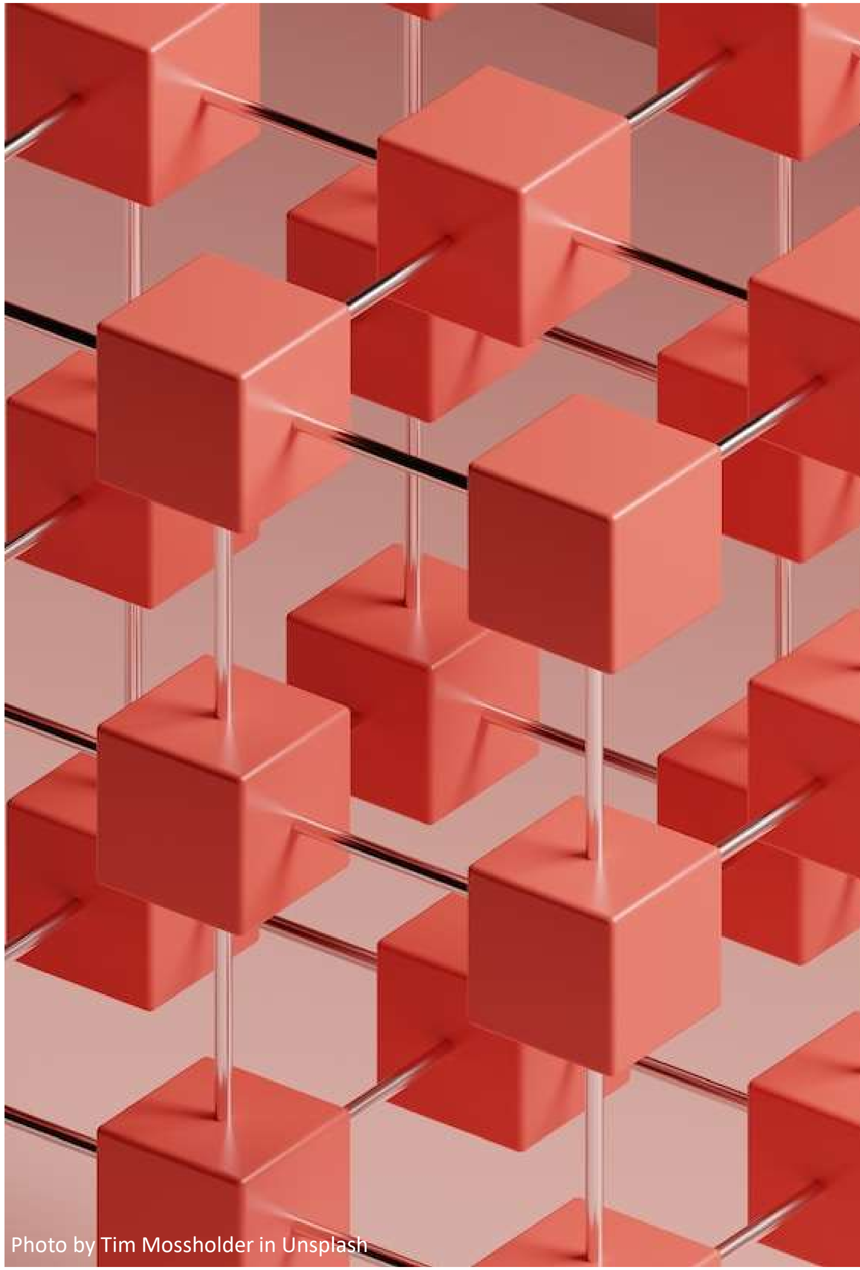


1 billion samples, 50 features



2 billion samples, 100 features

Barcelona Supercomputing Center
Centro Nacional de Supercomputación



… even beyond the datacenter

For different reasons than in HPC, in edge computing you also want to:

Distribute the processing

- Due to limited resource capabilities

Minimize data transfers

- To avoid network and privacy issues

Avoid disk accesses

- Many devices don't have them

Abstraction of the infrastructure for programmability

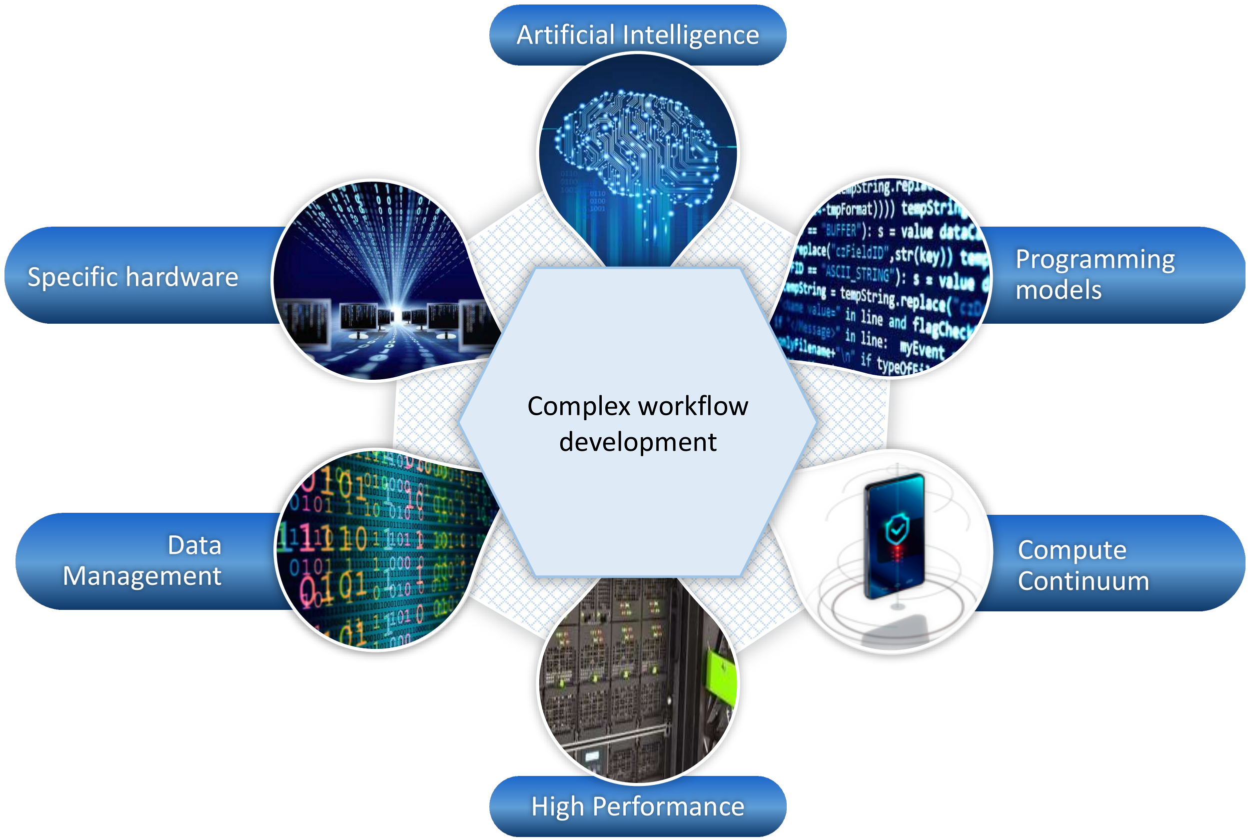
- Very complex and unstable

Edge-to-cloud environments can be seen as "a single" data processing platform

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

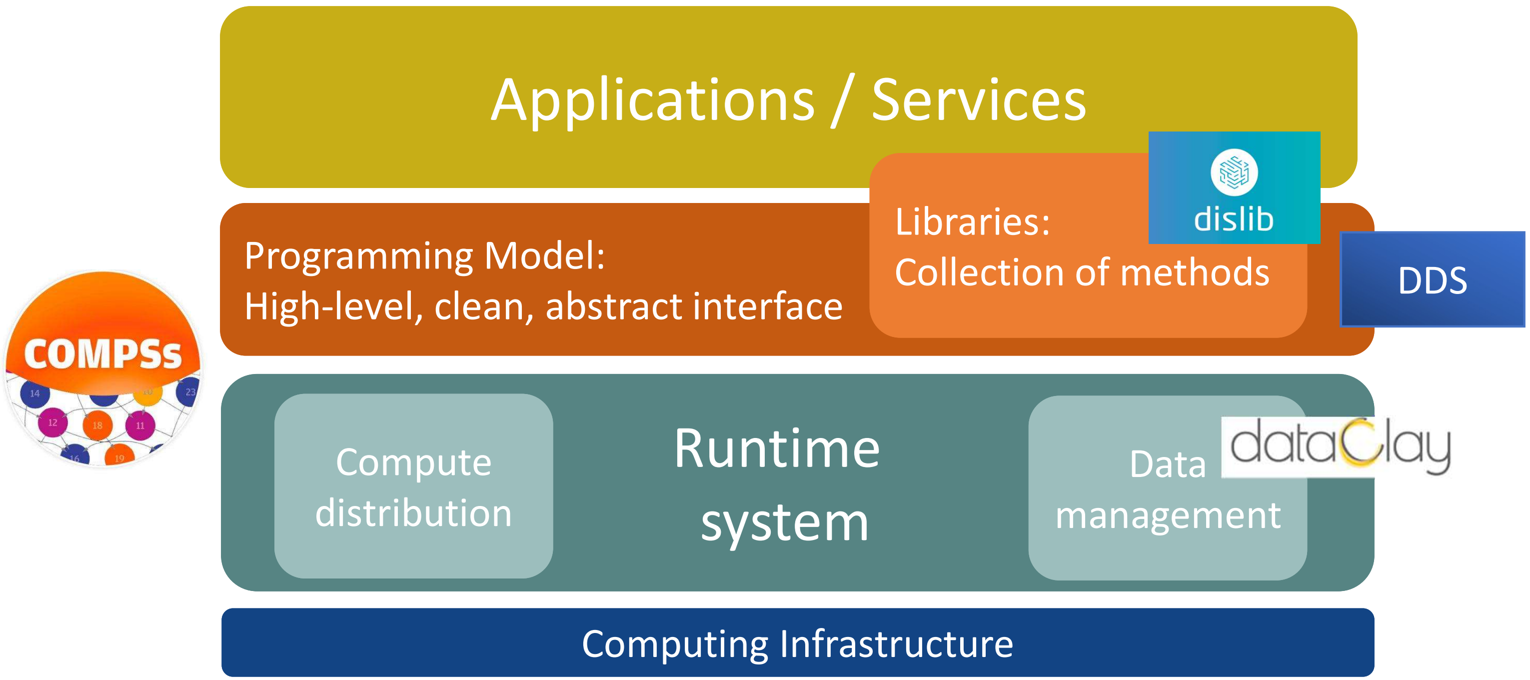
The Workflows and Distributed Computing group

- **Different** methodologies and software stacks are available
  - HPC simulation and modeling
  - Big Data
  - Artificial Intelligence
- There is a need for **integration** into a single application workflow
  - Facilitates complex workflow development
  - Enables global optimizations



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Software stack



PyCOMPSs ecosystem

Francesc Lordan

francesc.lordan@bsc.es

Motivation

New complex architectures constantly emerging

- Hardware Heterogeneity
  - Different HW characteristics (performance, memory, etc)
  - Different architectures -> compilation issues
- Network
  - Different types of networks
  - Instability
- Dynamicity
  - How to dynamically add/remove nodes to the infrastructure
- Trust and Security
- Power constraints from the devices in the edge

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Motivation

New complex architectures constantly emerging

- With their own way of programming them
  - Fine grain: e.g. Programming models and APIs to run with GPUs, NVMs (Non-Volatile Memories)
  - Coarse grain: e.g. APIs to deploy in Clouds
- **Difficulty** to develop applications
  - Higher learning curve / Time To Market (TTM)
  - What about non computer scientists???
- **Difficulty** to understand what is going on during execution (**Efficiency**)
  - Was it fast? Could it be even faster?
  - Am I paying more than I should?
- Tune your application for each architecture (or cluster)
  - E.g. partitioning data among nodes

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Motivation

- Create tools that make developers' life **easier**

  - Allow developers to focus on their problem
    - Integration of computational workloads (PyCOMPSs)
    - with machine learning and data analytics (dislib / DDS)

  - Intermediate layer: let the difficult parts to those tools
    - Act on behalf of the user
    - Distribute the work through resources
    - Deal with architecture specifics
    - Automatically improve performance

  - Tools for visualization
    - Monitoring
    - Performance analysis

Applications

High-level, clean, abstract interface

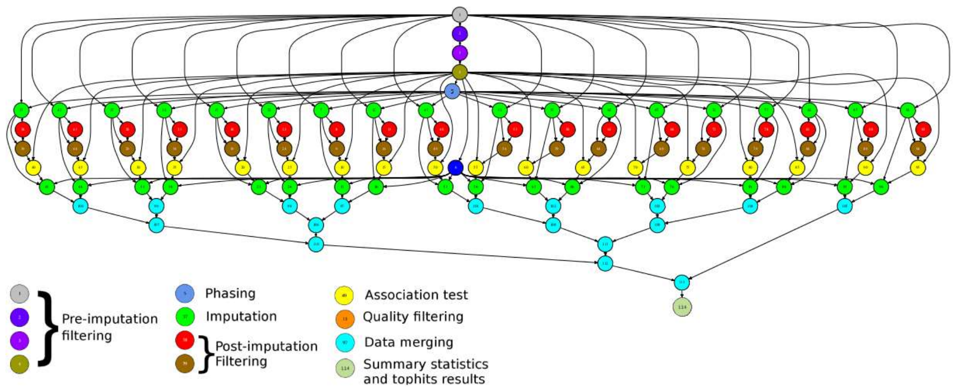
Power to the runtime

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Motivation

- Programming
  - Sequential programming
  - Agnostic of the target computing platform
  - Standard programming languages: Java, Python, C/C++
    - annotations/hints
  - Task based: task is the unit of work

- Runtime
  - Builds a task graph at runtime that express potential concurrency (workflow)
  - Exploitation of parallelism
  - Resource Management and Workload distribution



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

PyCOMPSs ecosystem overview

Applications

DDS

dislib

PyCOMPSs

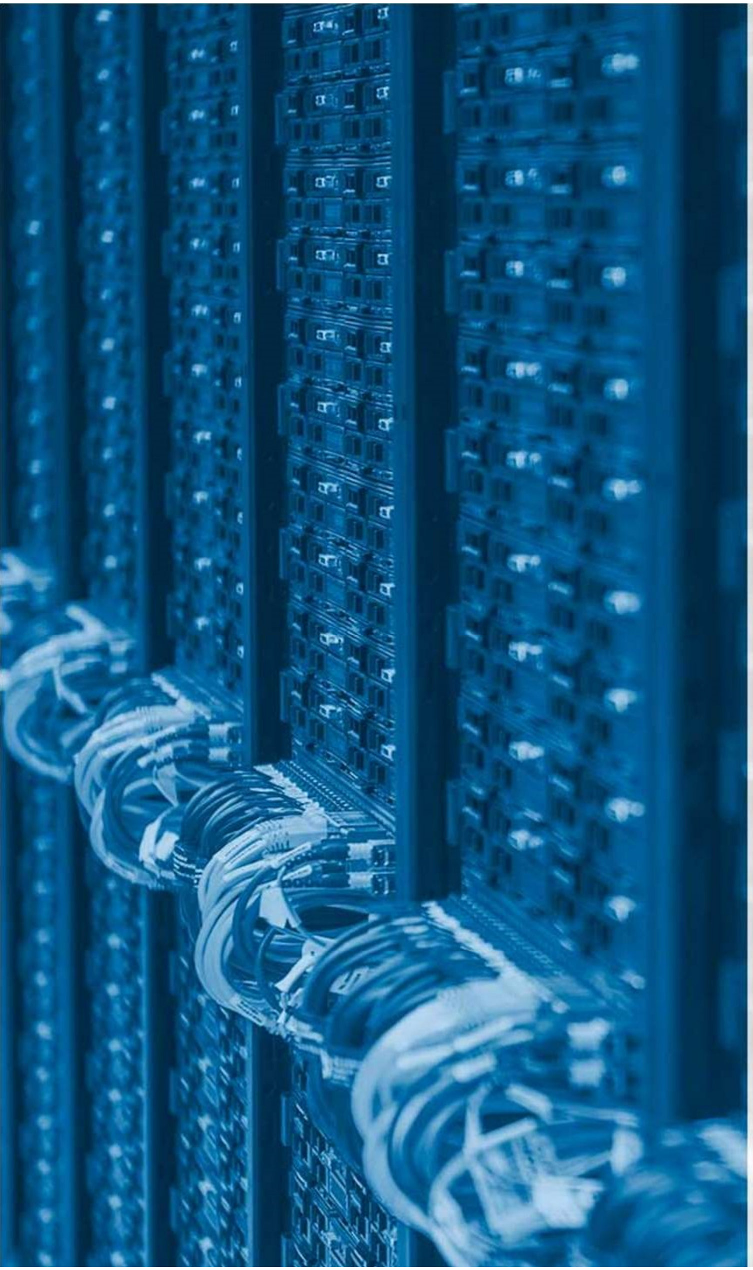
COMPSs
Runtime

Skeletons to operate on large datasets

Library of ML algorithms

General-purpose task-based programming model

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación



BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

EXCELENCIA
SEVERO
OCHOA

dislib

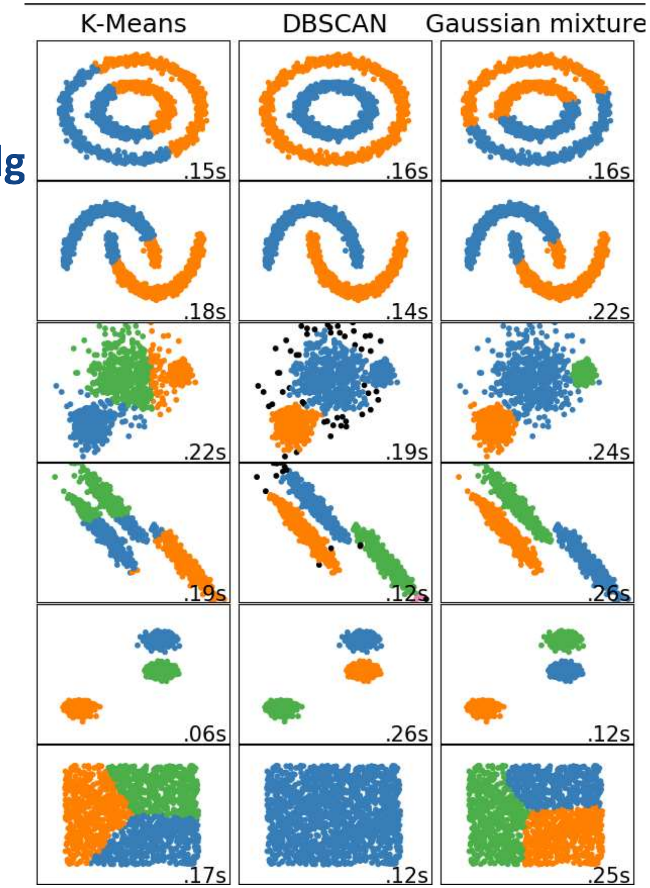
dislib: parallel machine learning

dislib: Collection of machine learning alg

- Unified interface, inspired in scikit-learn (fit-predict)
- Based on a distributed data structure (ds-array)
- Unified data acquisition methods
- Parallelism transparent to the user – PyCOMPSs parallelism hidden
- Open source, available to the community

Provides multiple methods:

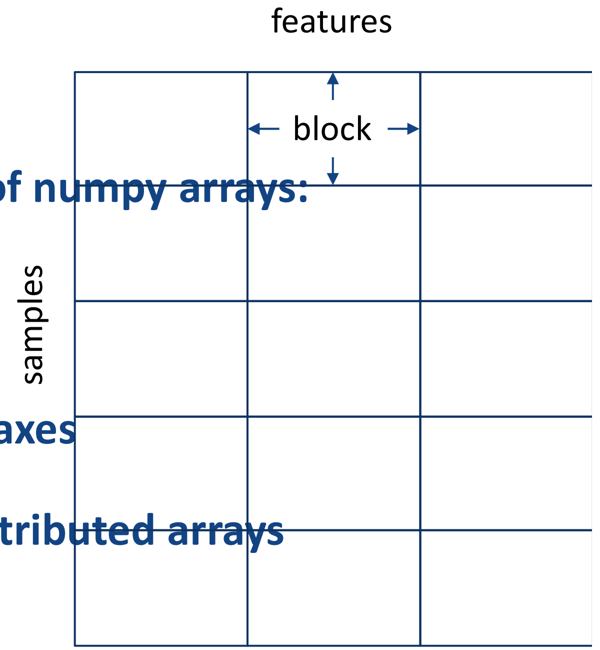
- data initialization
- Clustering
- Classification
- Model selection, ...



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Distributed array (ds-array)

- **2-dimensional structure (i.e., matrix)**
  - Divided in blocks (NumPy arrays)
- **Works as a regular Python object**
  - But not always stored in local memory!
- **Methods for instantiation and slicing with the same syntax of numpy arrays:**
  - Loading data (e.g., from a text file)
  - Indexing (e.g., x[3], x[5:10])
  - Operators (e.g., x.min(), x.transpose())
- **ds-arrays can be iterated efficiently along both axes**
- **Samples and labels can be represented by independent distributed arrays**
- **Data not always in memory:**
  - Inherent support for out-of-core operations, enabling large data-sets

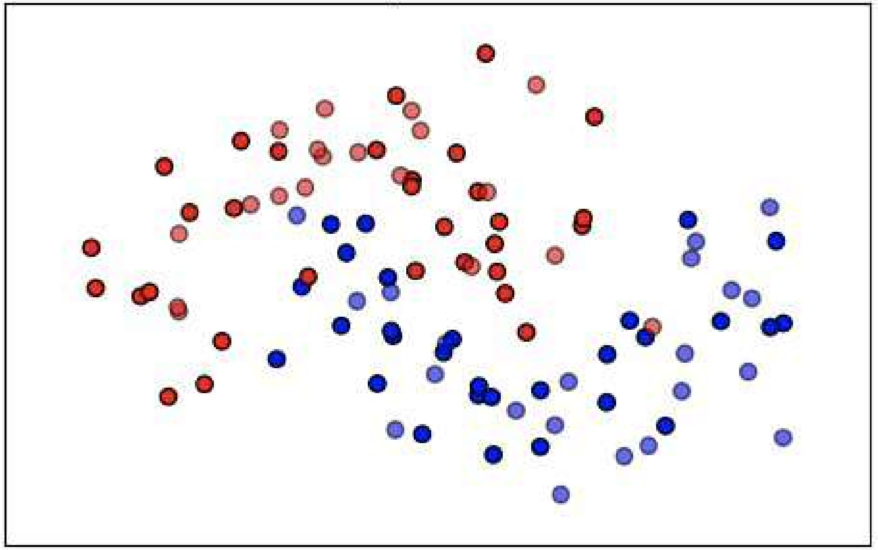


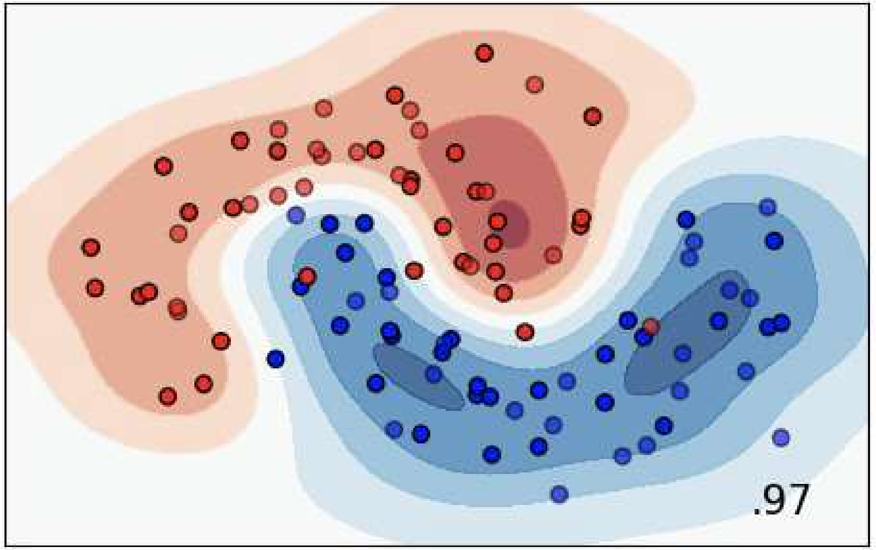
Supported Methods

- **Array creation routines**
  - random,
  - existing data
  - files
- **Matrix decomposition:**
  - Principal Component Analysis (PCA)
  - QR
  - TSQR
  - SVD
- **Clustering**
  - DBSCAN
  - K-Means
  - Gaussian Mixture
  - Daura (Gromos)
- **Neighbour queries**
  - K-nearest neighbours (KNN)
- **Classification**
  - CascadeSVM
  - RandomForest classifier
  - DecisionTree classifier
- **Recommendation:**
  - Alternating least squares (ALS)
- **Regression**
  - Linear regression
  - LASSO
  - RandomForest regressor
  - DecisionTree regressor
- **Model Selection**
  - GridSearch
  - RandomizedSearch
  - K-fold

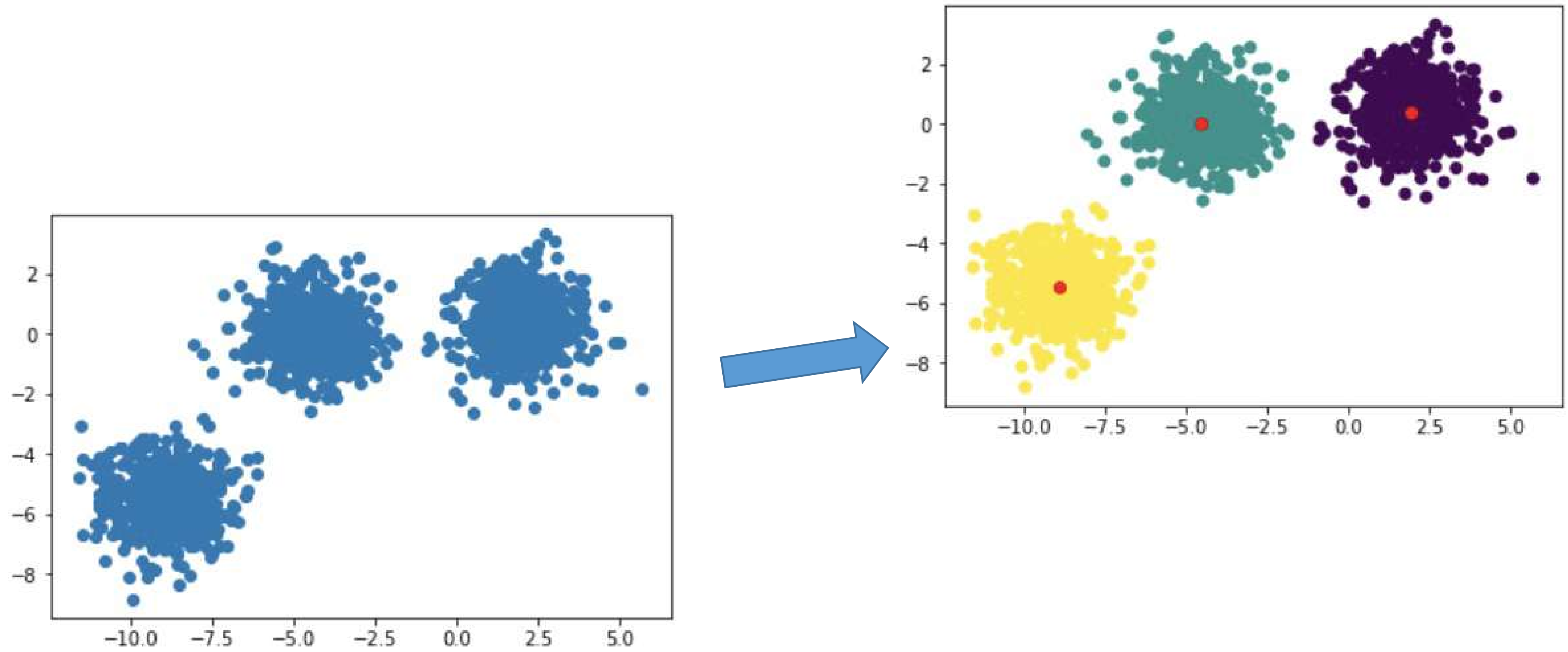
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Classification – Labeled data





Clustering – Unlabeled data



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Typical program using dislib

Block size

1. Read input data from file/s

2. Instantiate estimator with parameters

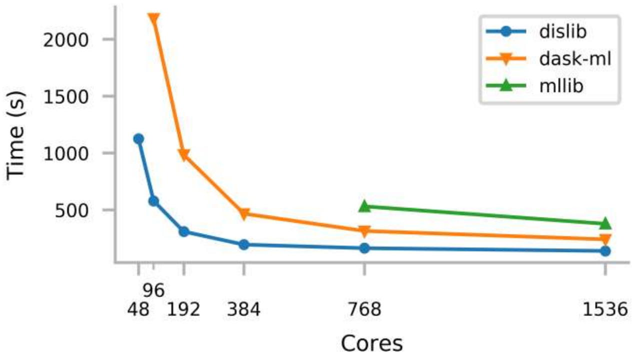
3. Fit estimator with training data

4. Make predictions on test data

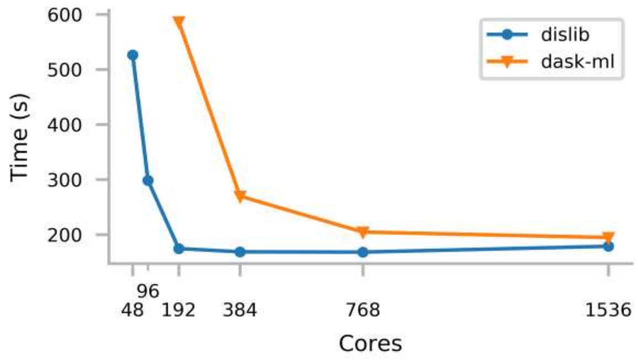
```
x = load_txt_file("train.csv", (10, 780))
x_test = load_txt_file("test.csv", (10, 780))

kmeans = KMeans(n_clusters=10)

kmeans.fit(x)

kmeans.predict(x_test)
```

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

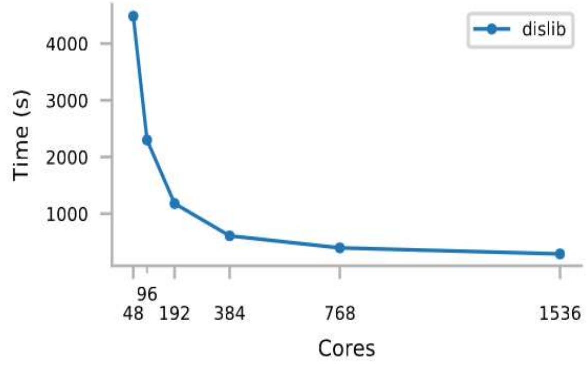
Performance evaluation



500 million samples
100 features



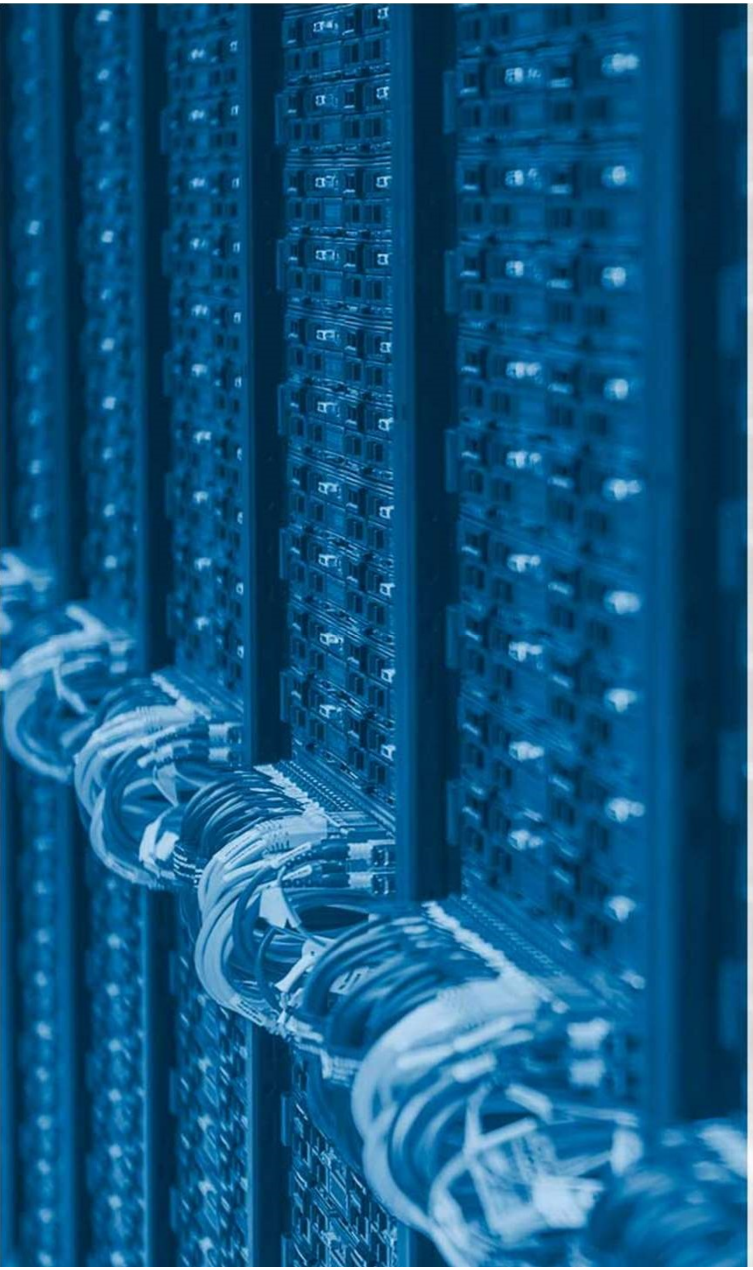
1 billion samples
50 features



2 billion samples
100 features

For very large sizes, dislib can obtain results while MLlib and dask fail to finish the execution

Barcelona Supercomputing Center
Centro Nacional de Supercomputación



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

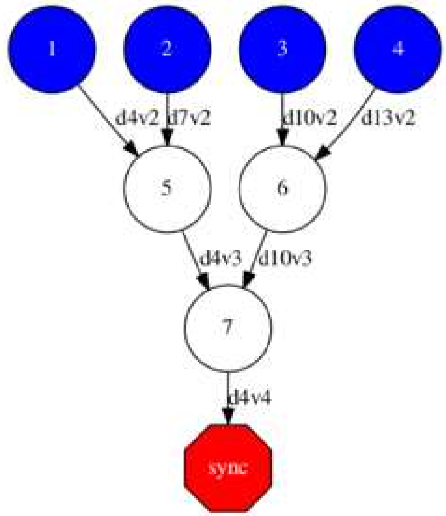
EXCELENCIA SEVERO OCHOA

DDS

DDS

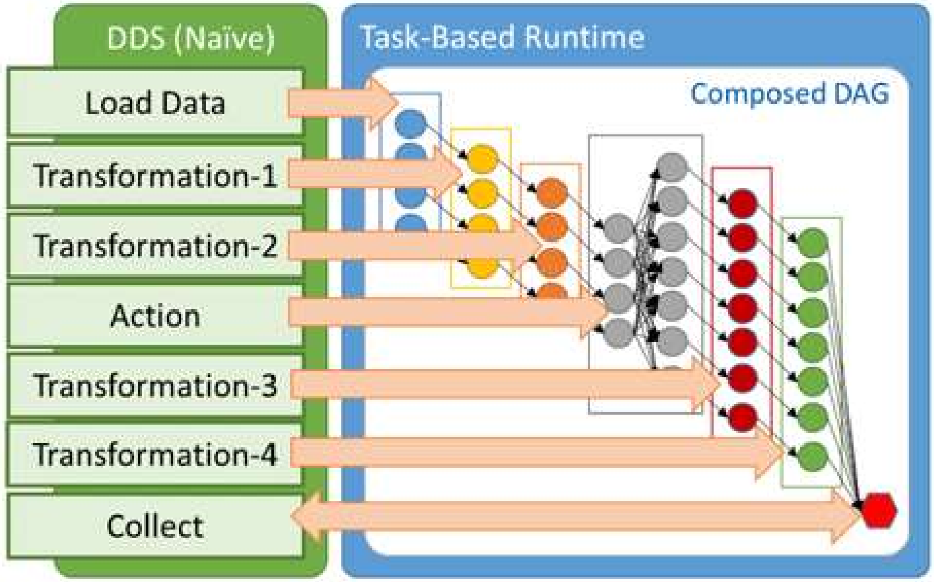
```python
# DDS implementation
from dds import DDS
wc_dds = DDS().load_files("<dir_path>")\
  .flat_map(lambda x: x[1].split()) \
  .map(lambda x: ''.join(e for e in x if e.isalnum())) \
  .count_by_value()

# RDD implementation
from pyspark import SparkContext
wc_rdd = SparkContext().wholeTextFiles("<dir_path>") \
  .flatMap(lambda pair: pair[1].split()) \
  .map(lambda x: ''.join(e for e in x if e.isalnum())) \
  .countByValue()
```

(a) Code comparison

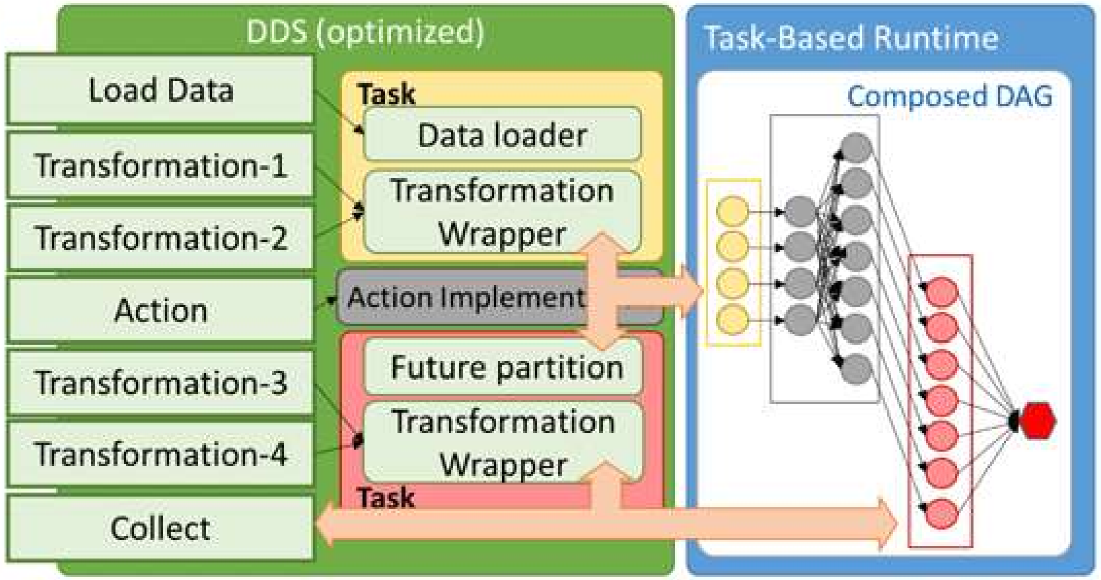


(b) Task Graph

DDS: Optimizations



(a) Naive solution for DDS

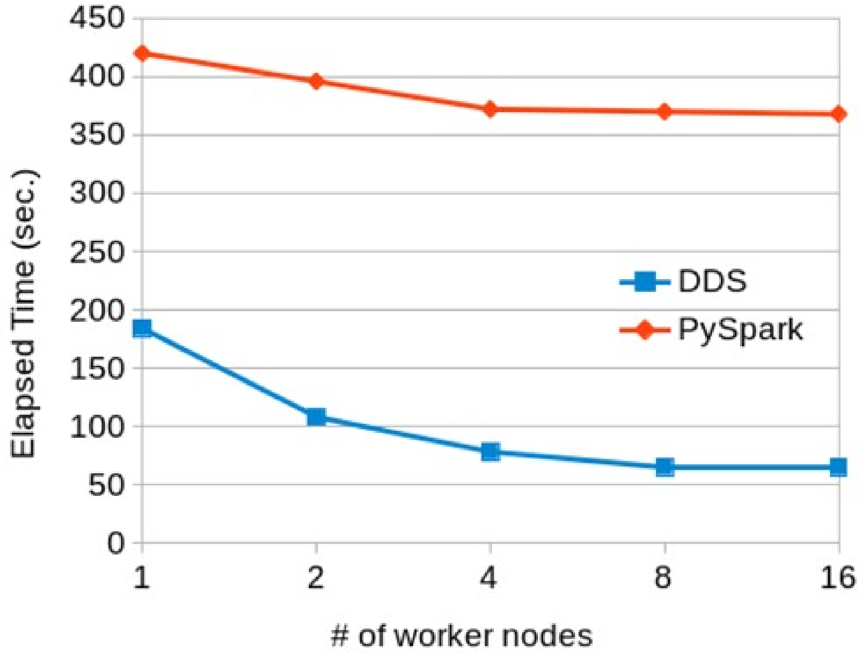


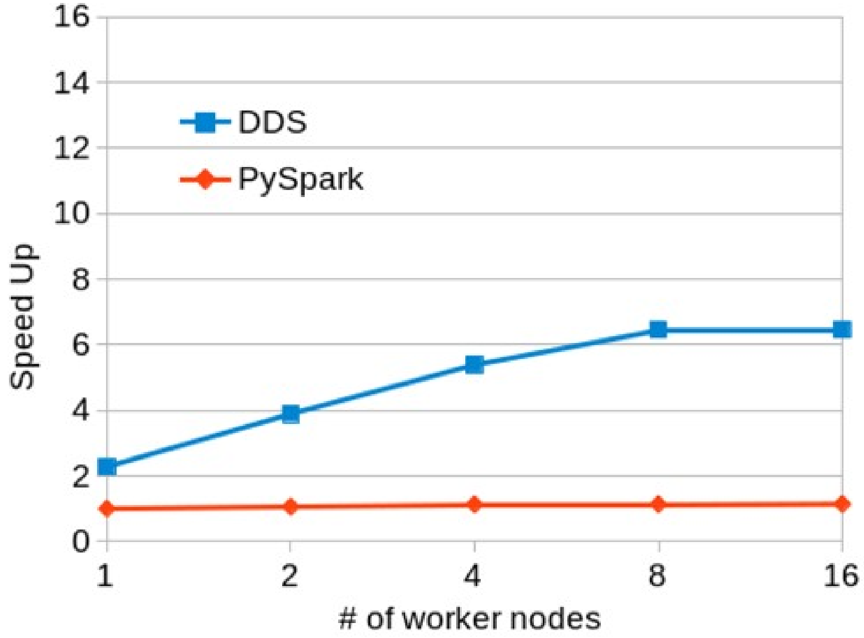
(b) Optimized solution for DDS

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Performance

Wordcount Gutenberg dataset (80GB)

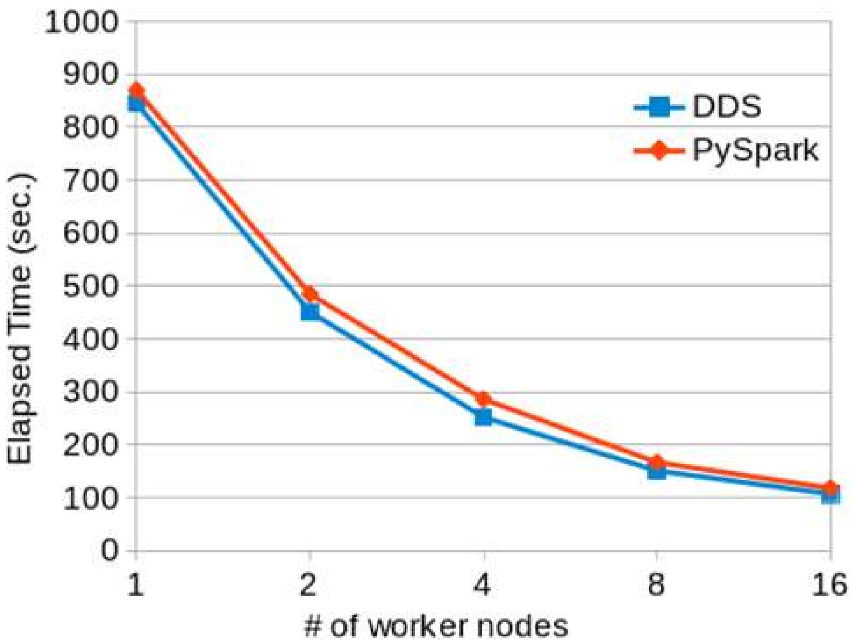


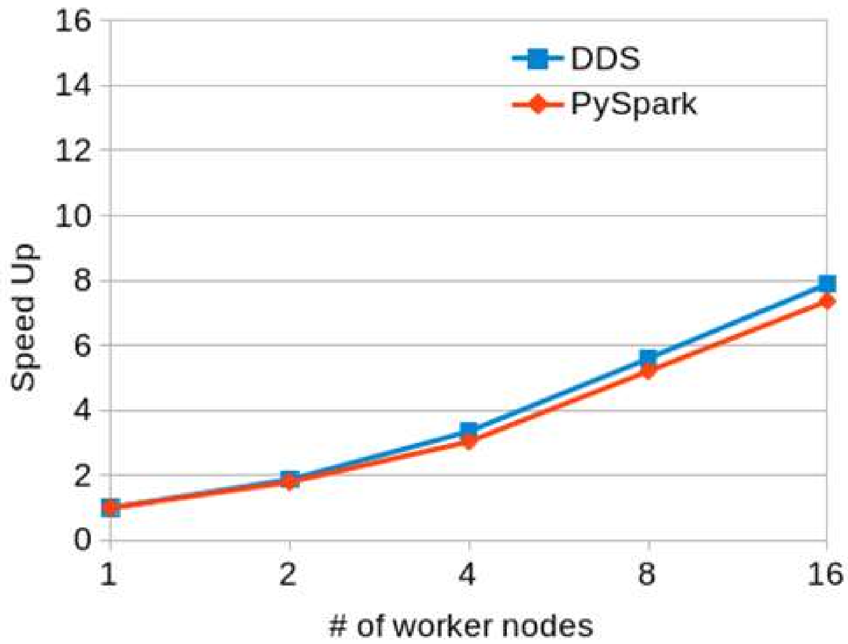


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Performance

Wordcount Lorem Ipsum dataset (100GB)

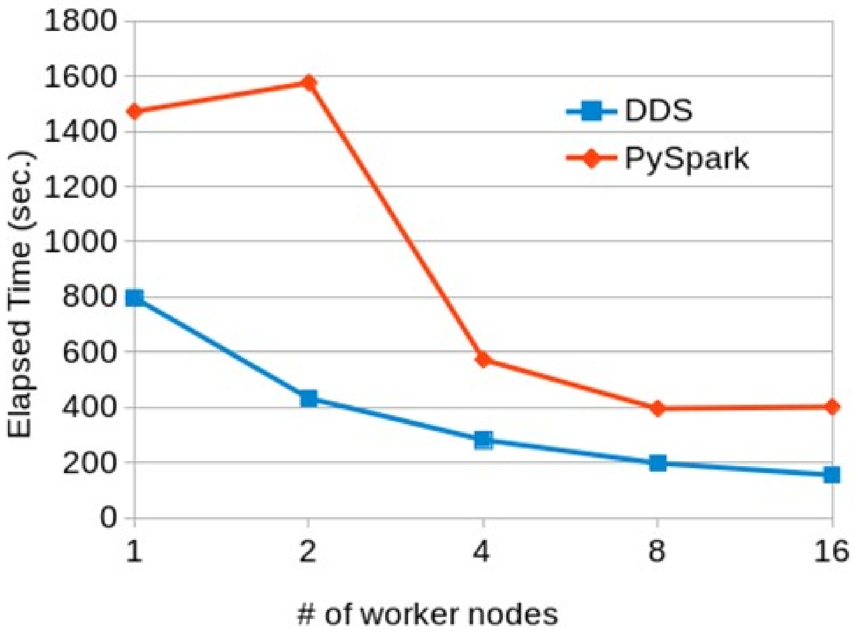


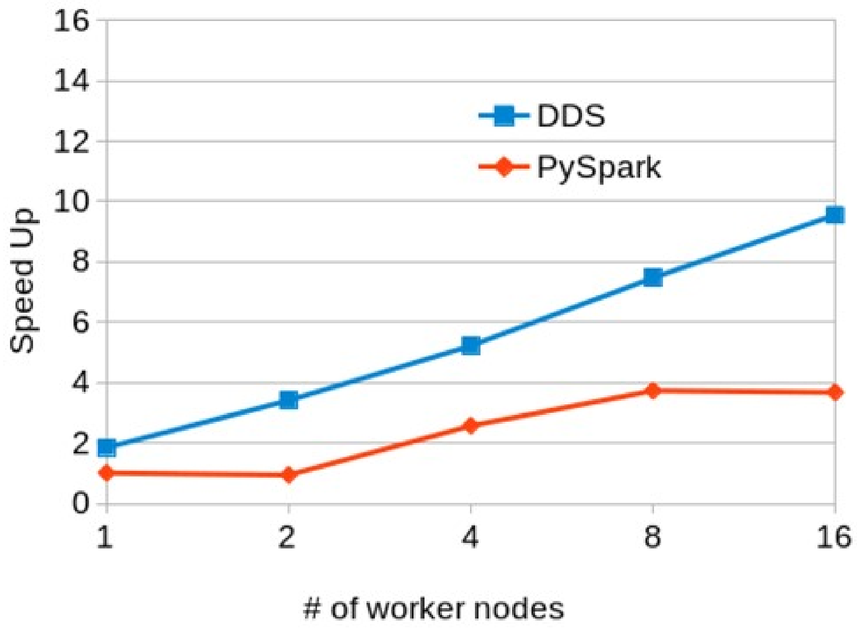


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Performance

Terasort (200GB dataset)

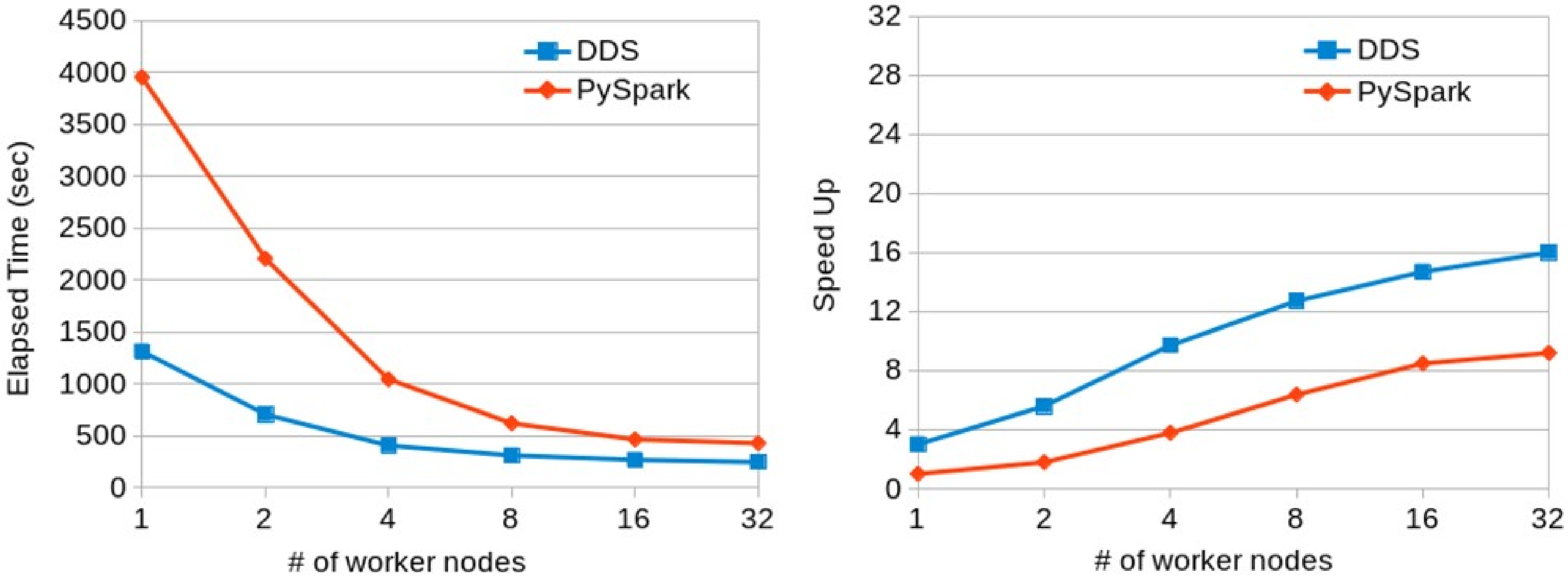


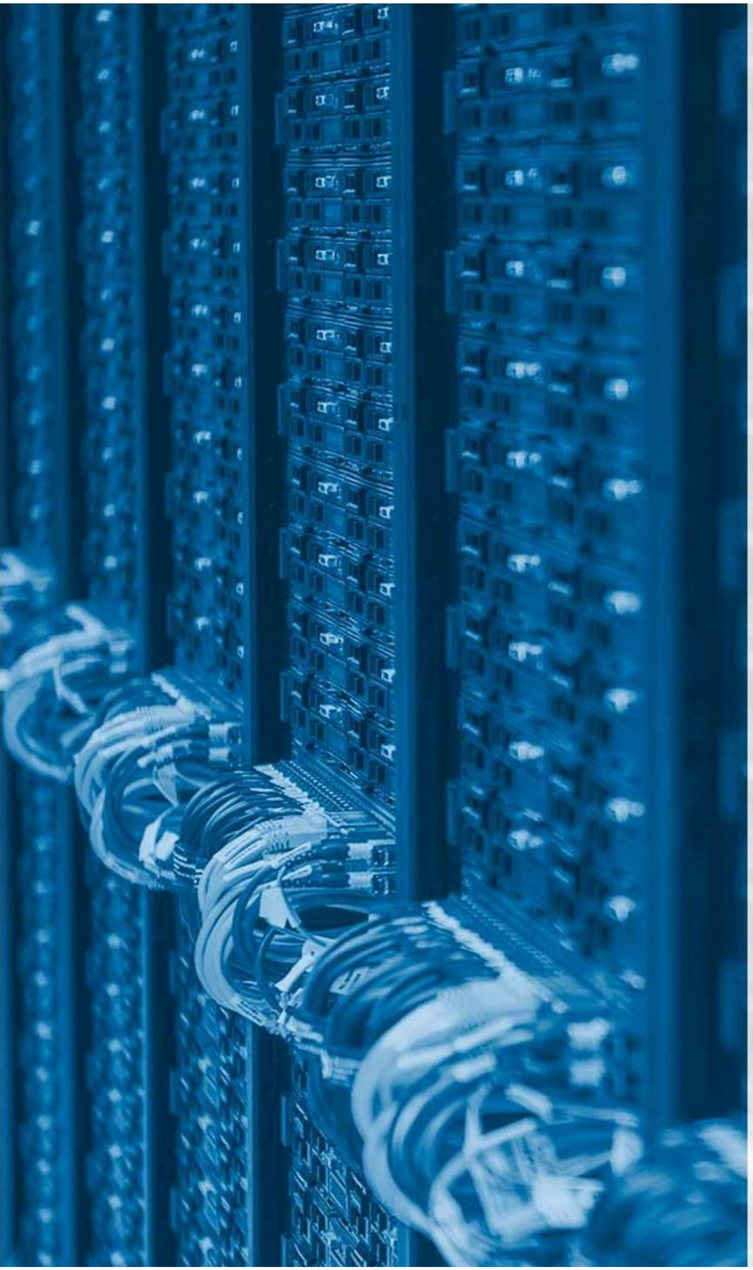


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Performance

Transitive Closure (15GB dataset)





Barcelona Supercomputing Center
Centro Nacional de Supercomputación

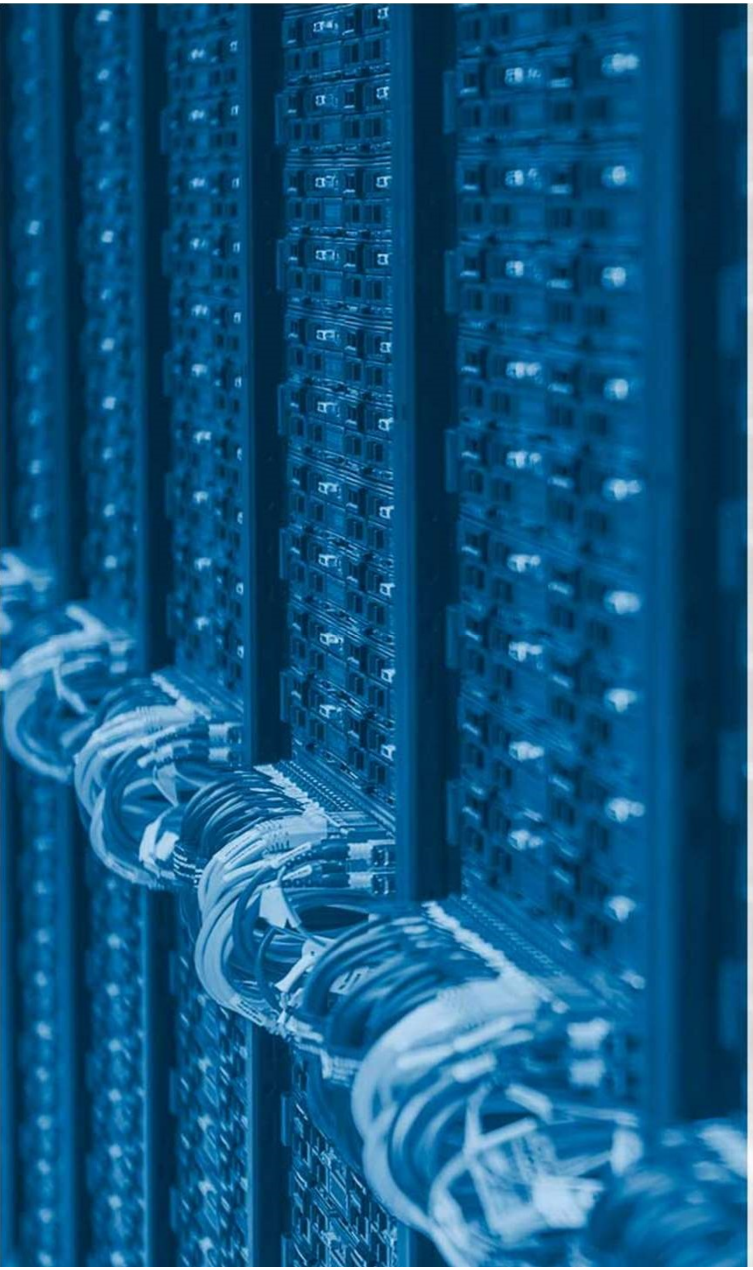
EXCELENCIA SEVERO OCHOA

pyCOMPSs

COMPSs Documentation

- Official Website:
  - http://compss.bsc.es
- Github repository
  - https://github.com/bsc-wdc/compss
- Documentation
  - https://compss-doc.readthedocs.io/en/stable/
- Tutorials
  - https://compss-doc.readthedocs.io/en/stable/Sections/10_Tutorial.html

Barcelona Supercomputing Center
Centro Nacional de Supercomputación



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

dataClay

Active objects across the network

Anna Queralt, Alex Barceló

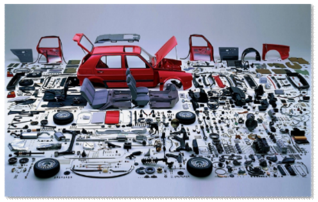
anna.queralt@bsc.es

alex.barcelo@bsc.es

From the application point of view

- There is still a gap between computation and data
- Physical separation → Communications
  - Serialization/deserialization is the main cost in data analytics applications [NSDI15]

- Different data models → Transformations
  - Overcoming the *impedance mismatch* amounts to up to 30% of the code [ICSE14]

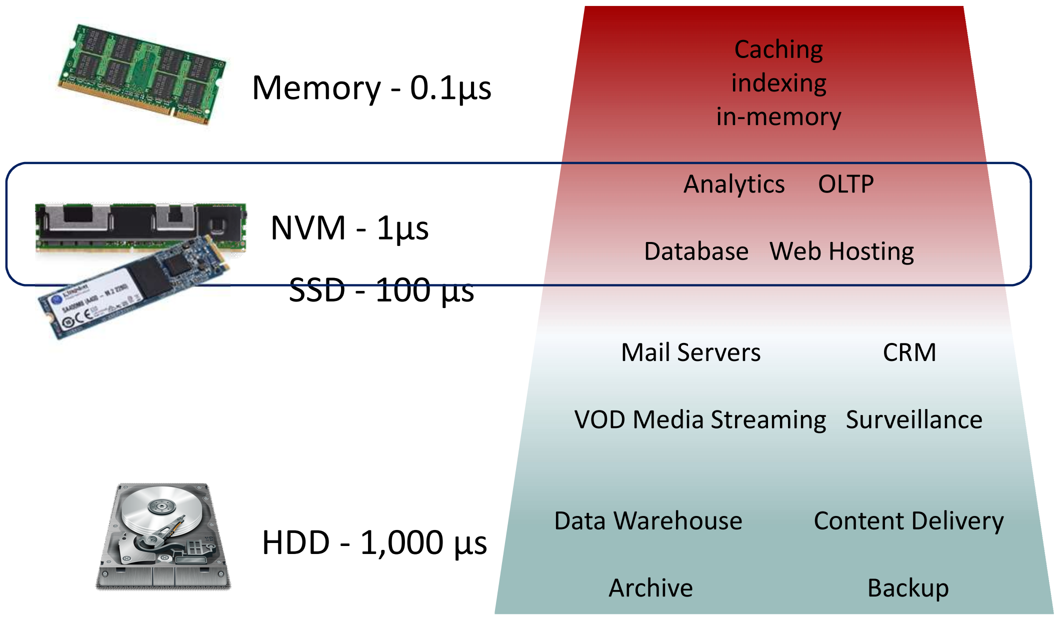


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

[NSDI15] K.Ousterhout et al. Making sense of performance in Data Analytics Frameworks. USENIX Symposium on Networked Systems Design and Implementation, 2015)
[ICSE14] T.H.Chen et al. Detecting performance anti-patterns for applications using object-relational mapping. Intl. Conf. Software Enginneering 2014

From the hardware point of view

- New storage devices, called Non-Volatile Memories or Persistent Memories, are becoming available
- Performance similar to memory, capacity similar to disk
- Byte-addressable: can store objects, not files



Computation can be done directly on stored data, without moving it to RAM

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Goal

- Building a distributed data management solution that:
- Maximizes performance and scalability
  - Bringing computation close to data by design
    - Enabling the execution of arbitrary code within the data store
  - Providing mechanism without imposing policy
    - Enabling customization (replication, placement, consistency guarantees…) according to application semantics
  - Can take advantage of new storage devices
- Lets the programmer focus on the domain
  - Supporting the data structures needed by the application
    - All data accessed in the same way, regardless if it's local or remote, persistent or volatile
  - Providing the illusion of infinite memory

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

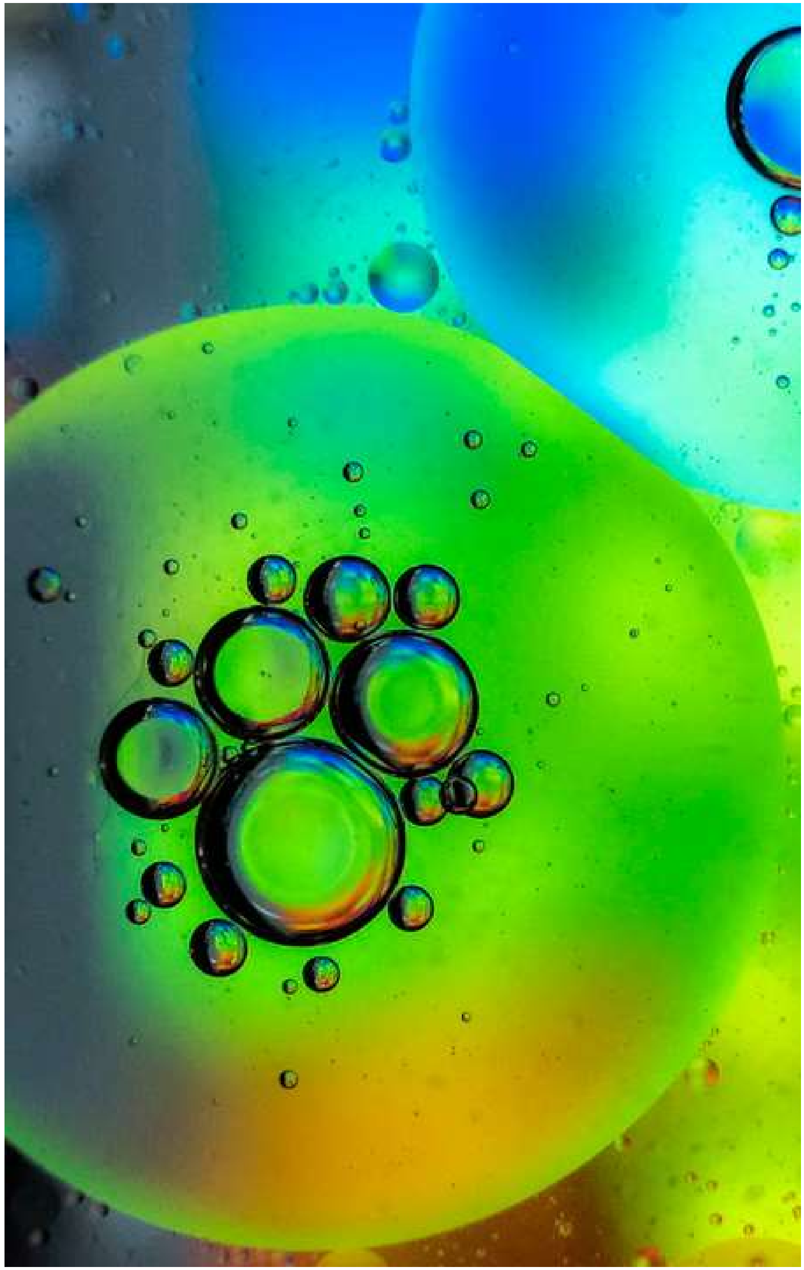
Approach

- Taking object stores as a starting point
  - Provides scalability
- Adding semantics to objects
  - Enables managing data at fine granularity
- Associating behavior to stored objects
  - Avoids data movements

Implementation using object-oriented abstractions

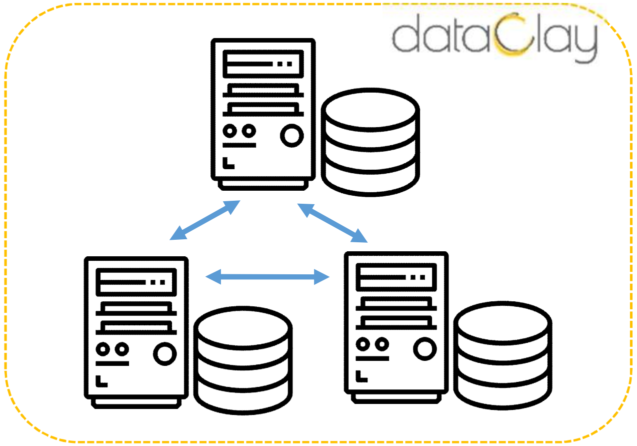
- Natural way of joining data and computation → performance
- Structure and behavior customizable per type → scalability
- Arbitrary data structures → representation of the domain

Barcelona Supercomputing Center
Centro Nacional de Supercomputación



dataClay in a nutshell

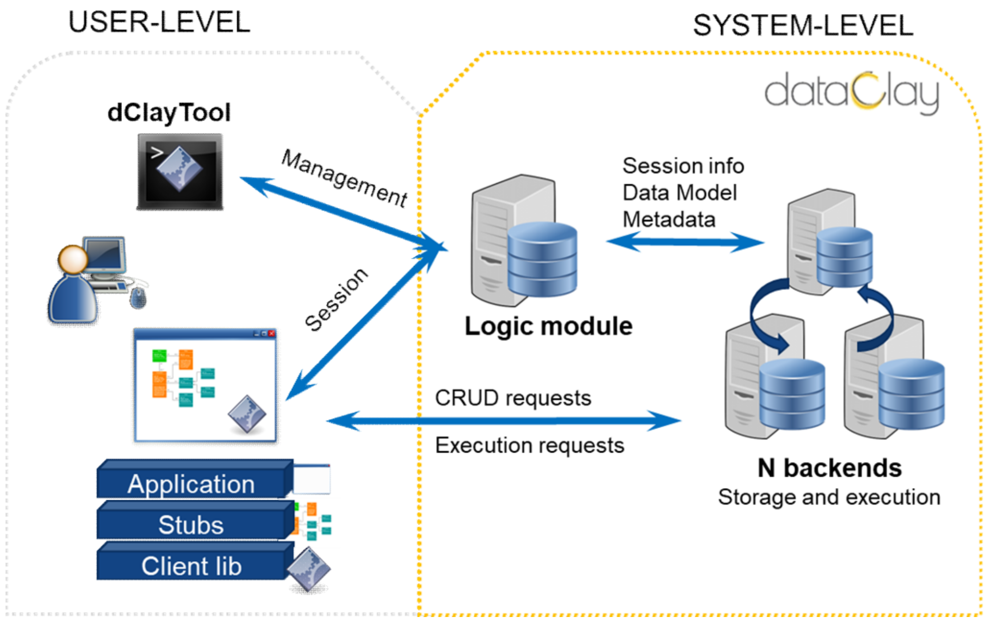
- Distributed *active object store* for HPC and data analytics applications
  - Facilitates the development
  - Optimizes execution
- A single data model to manage transparently:
  - Persistent and volatile data
  - Local and remote data
- Inherently exploits data locality
  - Objects = data + methods
- In-memory
  - Objects ready to be used
  - No transformations or serializations



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

dataClay architecture

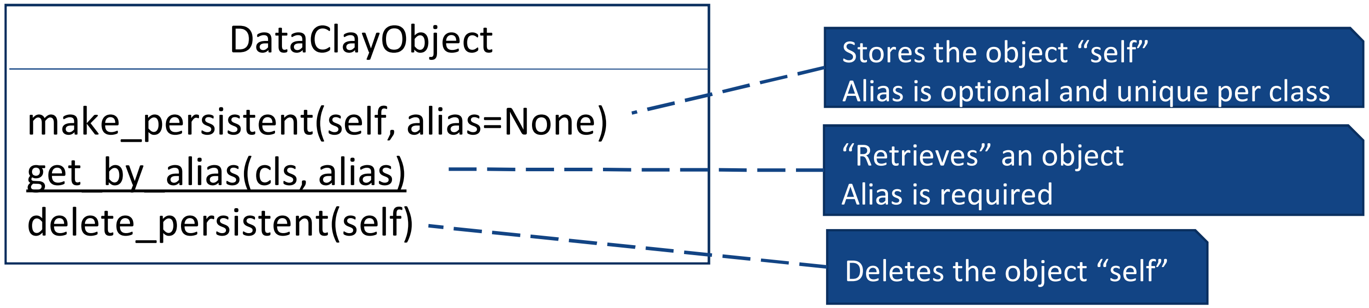
- Several backends for storage and execution of methods
- Logic module as entry point
- Persistent classes are registered in dataClay
  - Remote method execution
  - Fast serialization
- dataClay transparently forwards method execution to the backends



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

DataClayObject

- A class that enriches Python objects with generic data storage and access functionalities



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Example: a simple persistent class

```
from dataclay import DataClayObject, activemethod


class SampleClass(DataClayObject):
     number: int
     mean: float
     another_object: B
     a_list: list[B]
     a_dict: dict[str, tuple[float, float]]


     @activemethod
     def func(self, incr: int) -> int:
          return self.number + incr
```

- **Derive class from *DataClayObject***
- Specify properties and their types
- Annotate methods with *@activemethod*
- Specify types of parameters

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Example: a simple persistent class

```
from dataclay import DataClayObject, activemethod

class SampleClass(DataClayObject):
     number: int
     mean: float
     another_object: B
     a_list: list[B]
     a_dict: dict[str, tuple[float, float]]

     @activemethod
     def func(self, incr: int) -> int:
          return self.number + incr
```

- Derive class from *DataClayObject*
- **Specify properties and their types**
- Annotate methods with *@activemethod*
- Specify types of parameters

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Example: a simple persistent class

```python
from dataclay import DataClayObject, activemethod


class SampleClass(DataClayObject):
    number: int
    mean: float
    another_object: B
    a_list: list[B]
    a_dict: dict[str, tuple[float, float]]


    @activemethod
    def func(self, incr: int) -> int:
        return self.number + incr
```

- Derive class from *DataClayObject*
- Specify properties and their types
- **Annotate methods with *@activemethod***
- **Specify types of parameters**

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Example: an application with persistent objects

```
class SampleClass(DataClayObject):
      number: int
      mean: float
      another_object: B
      a_list: list[B]

      def func(self, incr: int)...
```

```
from my_classes.simple import SampleClass

...

a = SampleClass()
a.number = 24
a.make_persistent()

a.another_object = B()
b = a.another_object

c = SampleClass()
c.make_persistent("sample36")
c.a_list = [b]

c = SampleClass.get_by_alias("sample35")
d = c.func(e)
```

- **Code application using the classes**
- To store an object, call its *make_persistent* method
    - Related objects will be made persistent recursively
- To update a persistent object, just use the assignment instruction
    - Changes will be persistent, too
- To access a persistent object, either:
    - Retrieve it by alias
    - Navigate through relationships

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Example: an application with persistent objects

```
class SampleClass(DataClayObject):
      number: int
      mean: float
      another_object: B
      a_list: list[B]

      def func(self, incr: int)...
```

```
from my_classes.simple import SampleClass

...

a = SampleClass()
a.number = 24
a.make_persistent()

a.another_object = B()
b = a.another_object

c = SampleClass()
c.make_persistent("sample36")
c.a_list = [b]

c = SampleClass.get_by_alias("sample35")
d = c.func(e)
```

- Code application using the classes
- **To store an object, call its *make_persistent* method**
  - **Related objects will be made persistent recursively**
- To update a persistent object, just use the assignment instruction
  - Changes will be persistent, too
- To access a persistent object, either:
  - Retrieve it by alias
  - Navigate through relationships

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Example: an application with persistent objects

```
class SampleClass(DataClayObject):
      number: int
      mean: float
      another_object: B
      a_list: list[B]

      def func(self, incr: int)...
```

```
from my_classes.simple import SampleClass

...

a = SampleClass()
a.number = 24
a.make_persistent()

a.another_object = B()
b = a.another_object

c = SampleClass()
c.make_persistent("sample36")
c.a_list = [b]

c = SampleClass.get_by_alias("sample35")
d = c.func(e)
```

- Code application using the classes
- To store an object, call its *make_persistent* method
  - Related objects will be made persistent recursively
- **To update a persistent object, just use the assignment instruction**
  - **Changes will be persistent, too**
- To access a persistent object, either:
  - Retrieve it by alias
  - Navigate through relationships

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Example: an application with persistent objects

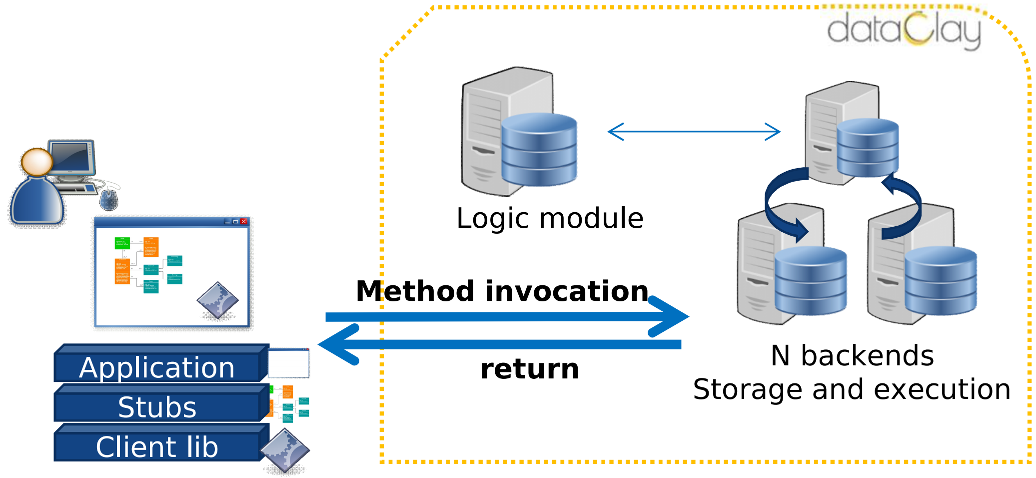
```
class SampleClass(DataClayObject):
      number: int
      mean: float
      another_object: B
      a_list: list[B]

      def func(self, incr: int)...
```

```
from my_classes.simple import SampleClass

...

a = SampleClass()
a.number = 24
a.make_persistent()

a.another_object = B()
b = a.another_object

c = SampleClass()
c.make_persistent("sample36")
c.a_list = [b]

c = SampleClass.get_by_alias("sample35")
d = c.func(e)
```

- Code application using the classes
- To store an object, call its *make_persistent* method
  - Related objects will be made persistent recursively
- To update a persistent object, just use the assignment instruction
  - Changes will be persistent, too
- **To access a persistent object, either:**
  - **Retrieve it by alias**
  - **Navigate through relationships**

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Active storage



Application

```python
from my_classes.simple import SampleClass

...

a = SampleClass()
a.make_persistent()
print(a.func(4))

...
```

Registered class

```python
class SampleClass(DataClayObject):
      number: int
      mean: float
      another_object: B
      a_list: list[B]

     @activemethod
      def func(self, incr: int) -> int:
           return self.number + incr
```

Why is this useful?

Without using active storage features

```
from my_classes.sim import Observations

temps = Observations.get_by_alias("temps")

n = 0
avg = 0
for value in temps.values:
    avg += value
    n += 1
avg /= n

print(avg)
```

The whole content of “temps” is transferred from dataClay to the application to find the average

Using active storage features

```
from my_classes.sim import Observations

temps = Observations.get_by_alias("temps")

avg = temps.get_average()

print(avg)
```

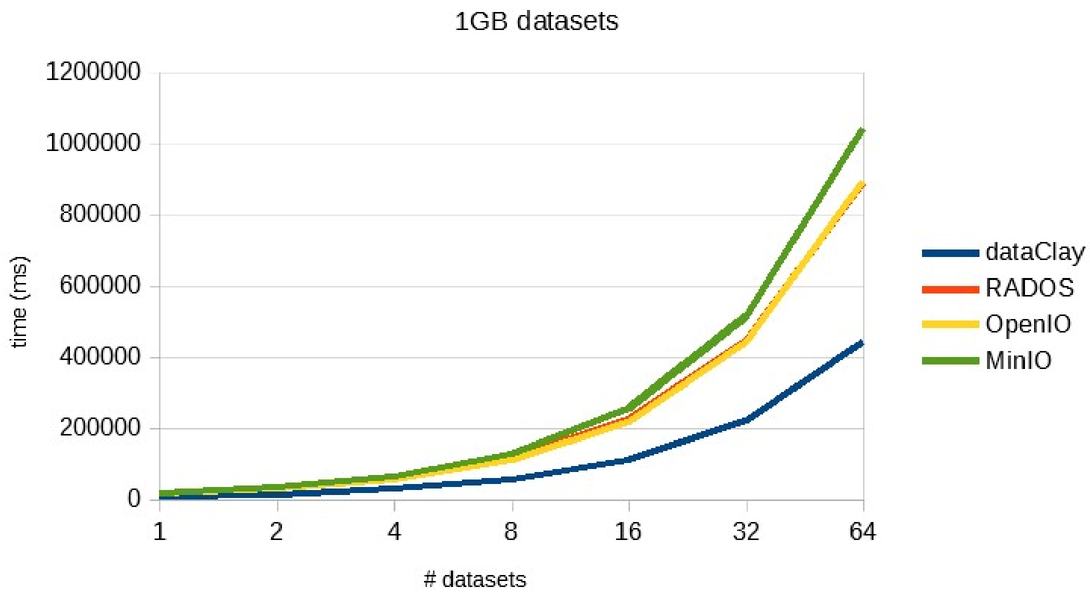
The r
data
trans
bigge

```
from dataclay import DataClayObject, activemethod

class Observations(DataClayObject):
    values: list[int]

    @activemethod
    def get_average(self) -> float:
        n = 0
        avg = 0
        for value in temps.values:
            avg += value
            n += 1
        avg /= n
```

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Performance improvement

- Processing a dataset (an int array) and returning 50% of its elements as a result
  - 1GB per dataset
- Executed in Grid5000, 4 nodes
  - 128 GB RAM, 48 cores
- Comparison with other object stores
  - With RADOS, OpenIO and MinIO the datasets are returned to the application, where the processing is applied
  - As the amount of data grows, performance gains with dataClay increase
  - Same happens as the size of the result decreases (not shown)
- The difference is even higher when using Non-Volatile Memories instead of disk



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

dataClay
Active objects across the network

dataClay demo

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Links of interest

- eBISS 2023 demos: https://github.com/bsc-dom/eBISS-2023

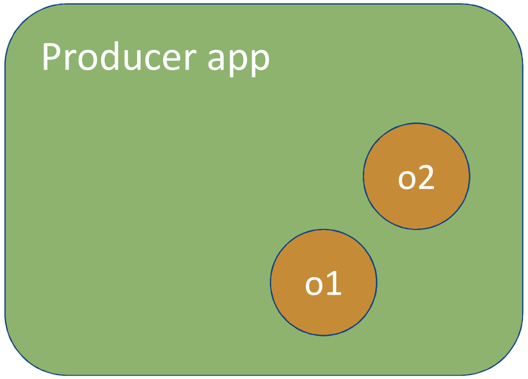
- dataClay site: http://www.bsc.es/dataClay
- Documentation: https://dataclay.readthedocs.io
- Repository: https://github.com/bsc-dom
  - Source code
  - Examples
  - Demos

- Contact and support: support-dataclay@bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

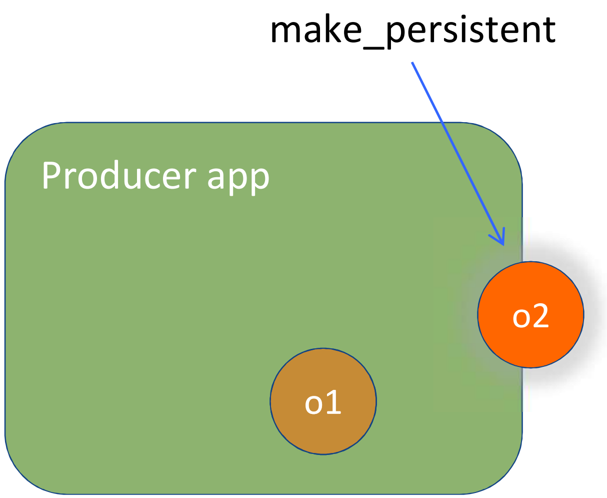
An integrated demo

dataClay within a workflow



- Applications can operate on persistent and in-memory objects seamlessly
- Producer-consumer workflows can easily share objects
- Flat object space shared across nodes
- Data transfers are avoided (in-situ processing)

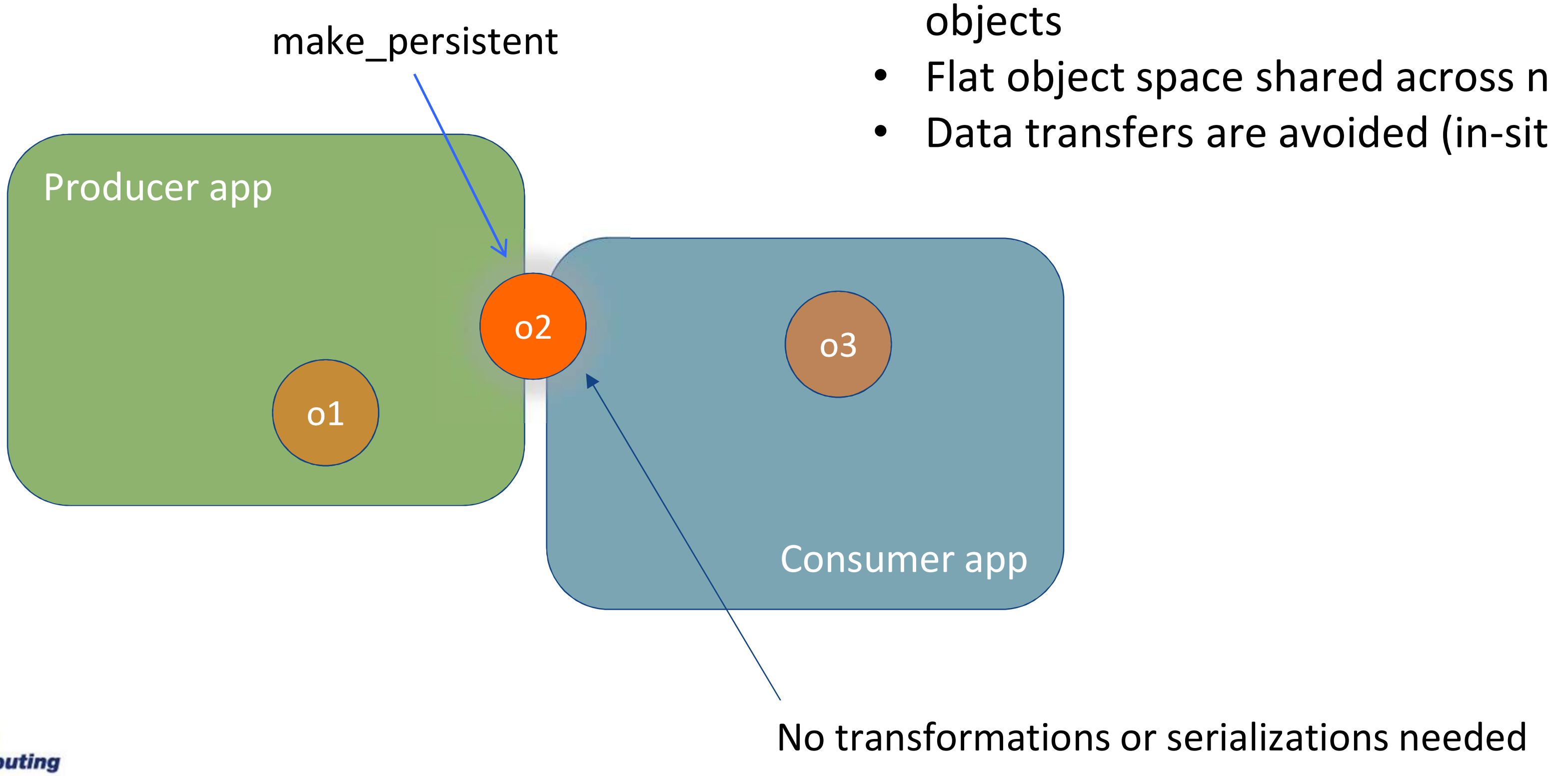
dataClay within a workflow



- Applications can operate on persistent and in-memory objects seamlessly
- Producer-consumer workflows can easily share objects
- Flat object space shared across nodes
- Data transfers are avoided (in-situ processing)

dataClay within a workflow

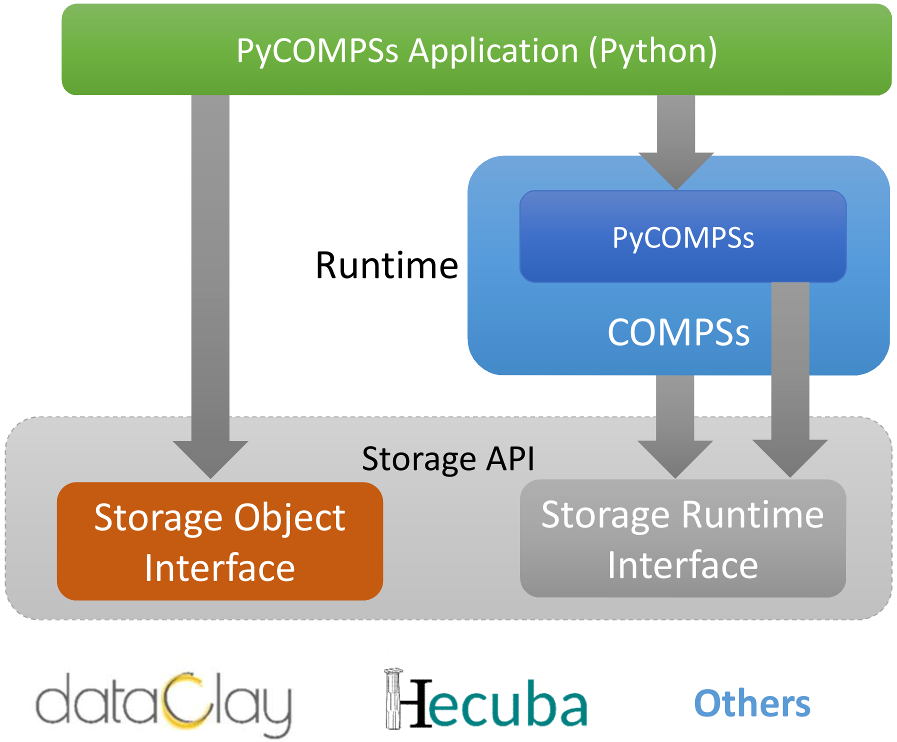
- Applications can operate on persistent and in-memory objects seamlessly
- Producer-consumer workflows can easily share objects
- Flat object space shared across nodes
- Data transfers are avoided (in-situ processing)



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

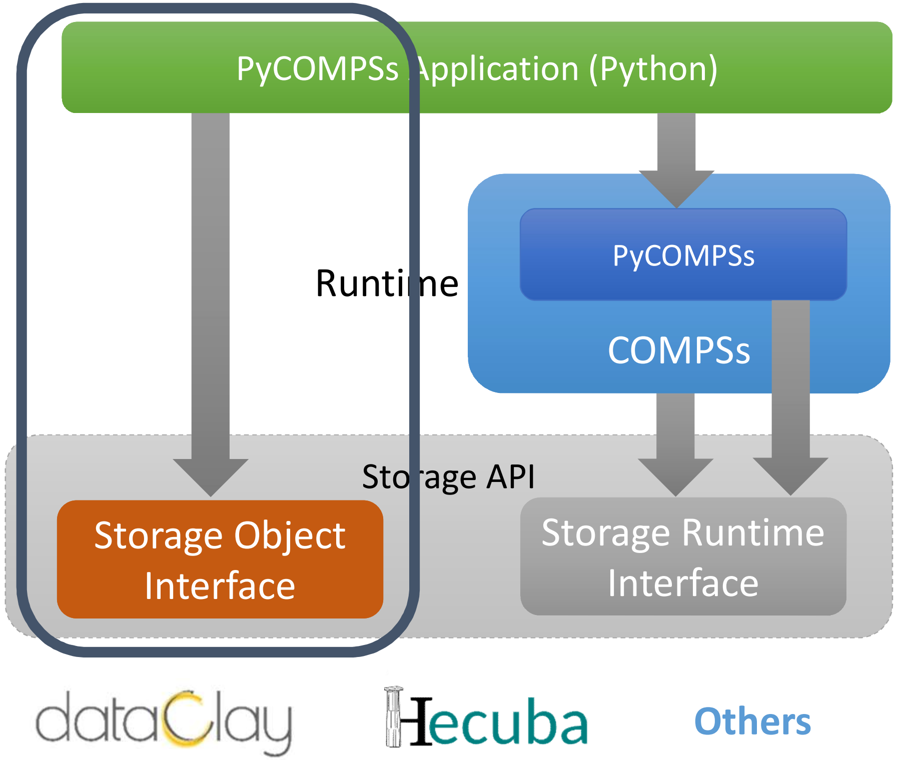
Integration with COMPSs

- Abstract *Storage Object Interface*
- Two types of functionalities:
  - For the application developer
    - *Storage Object Interface*
  - For the programming model runtime
    - *Storage Runtime Interface*



Integration with COMPSs

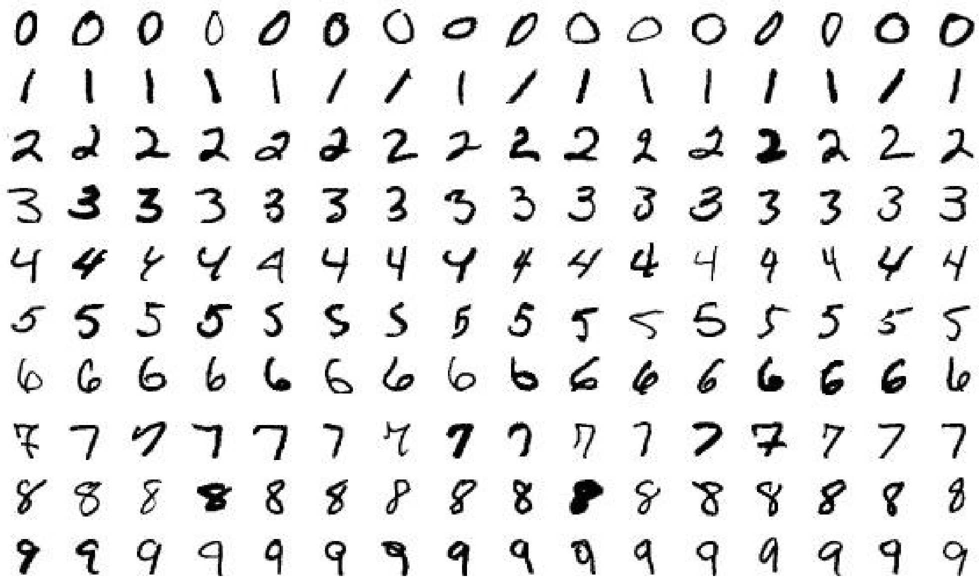
- Abstract *Storage Object Interface*
- Two types of functionalities:
  - For the application developer
    - *Storage Object Interface*
  - For the programming model runtime
    - *Storage Runtime Interface*



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Demo

- Handwritten digits recognition
- 2 alternatives
  - Machine Learning version
    - K-nearest neighbors
    - Implemented using COMPSs (and dislib) and dataClay
  - Deep Learning version
    - CNN
    - Implemented using PyTorch and dataClay



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Bibliography

- ServiceSs: an interoperable programming framework for the Cloud. F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Álvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, and R. M. Badia, Journal of Grid Computing 12(1): 67-91 (2014)
- PyCOMPSs: Parallel computational workflows in Python. Enric Tejedor, Yolanda Becerra, Guillem Alomar, Anna Queralt, Rosa M. Badia, Jordi Torres, Toni Cortes, Jesús Labarta, IJHPCA 31(1): 66-82 (2017).
- dataclay: A distributed data store for effective inter-player data sharing. Jonathan Martí, Anna Queralt, Daniel Gasull, Alex Barceló, Juan José Costa, Toni Cortes. Journal of Systems and Software, Volume 131: 129-145 (2017).
- dislib: Large Scale High Performance Machine Learning in Python. J. Álvarez Cid-Fuentes, S. Solà, P. Álvarez, A. Castro-Ginard, and R. M. Badia, in 15th International Conference on eScience: 96-105 (2019)
- Workflow Environments for Advanced Cyberinfrastructure Platforms. R.M. Badia, J. Ejarque, F. Lordan, D. Lezzi, J. Conejero, J. Álvarez, Y. Becerra, A. Queralt. ICDCS 2019: 1720-1729 (2019).
- DDS: Integrating data analytics transformations in task-based workflows. N. Mammadli, J. Ejarque, J. Álvarez, R.M. Badia, Open Research Europe 2023, 2:66.
- Revisiting active object stores: Bringing data locality to the limit with NVM. Future Generation Computer Systems 129: 425-439 (2022).

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Thank you

This project has received funding from the European Union's Horizon 2020 JTI-EuroHPC research and innovation programme H2020-JTI-EuroHPC-2019-1, under grant agreements No: 955558 — eFlows4HPC, 956748 — ADMIRE