Big data science Day 4

F. Legger - INFN Torino

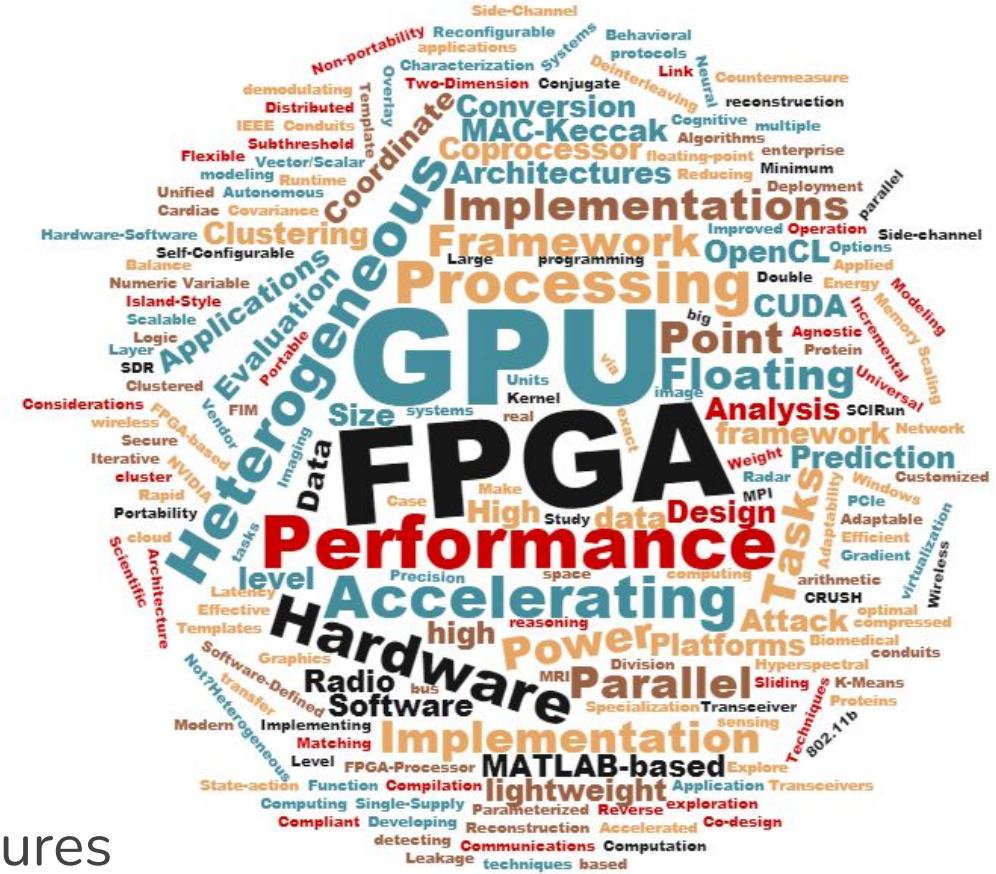
https://github.com/leggerf/MLCourse-2021

Previously

- Big data
- Analytics
- Machine learning
- Deep learning

Today

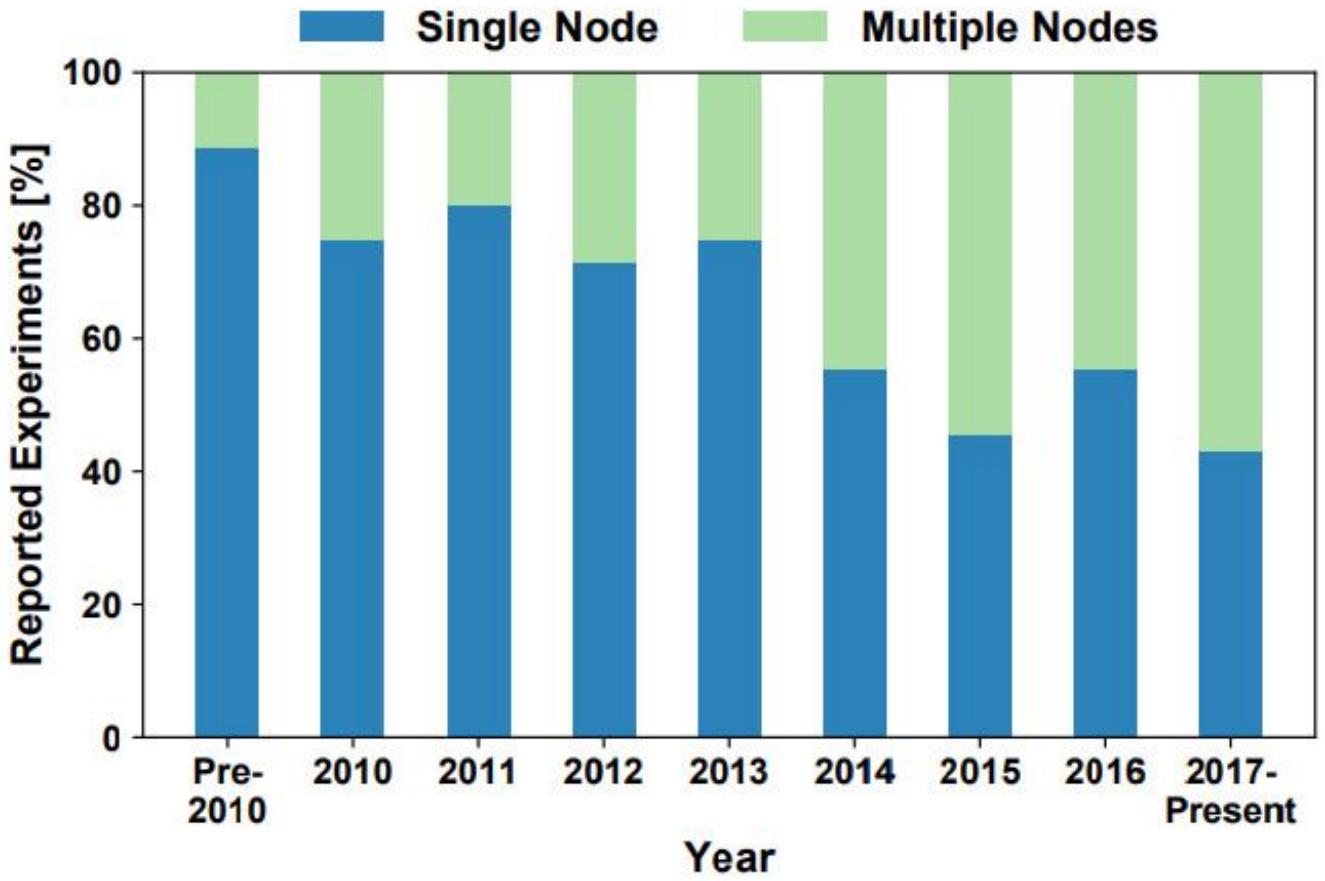
- Parallelisation
- Heterogeneous architectures



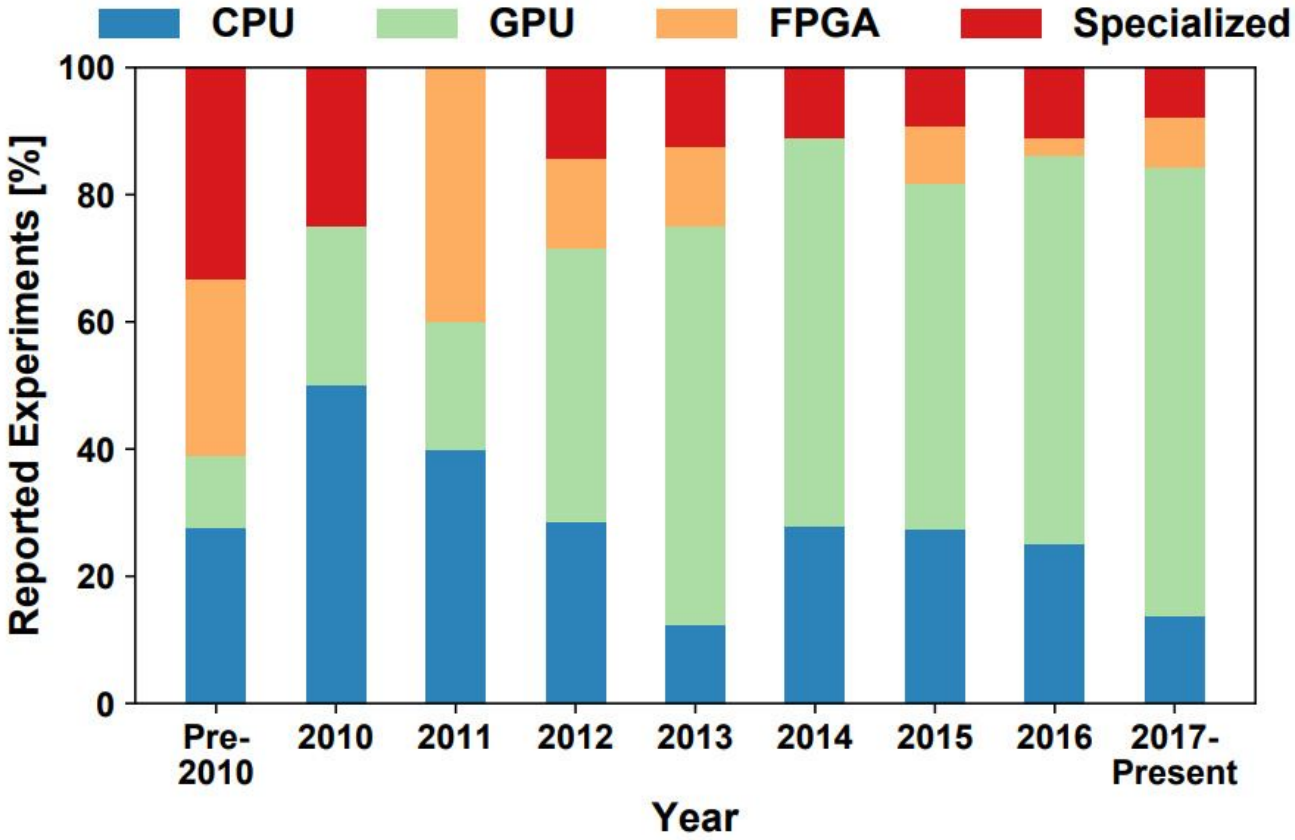
Summary

- Machine learning (ML): family of algorithms with ability to automatically learn and improve from experience without being explicitly programmed
  - potential to approximate linear and non-linear relationships
- For a given problem:
  - find algorithm that will give the best results
  - Train model
  - Tune hyperparameters
  - Do cross-validation
  - Do inference

- Most ML algorithms require significant amount of CPU, RAM and sometimes GPU in order to be applied efficiently
- **Does it fit on your laptop?**



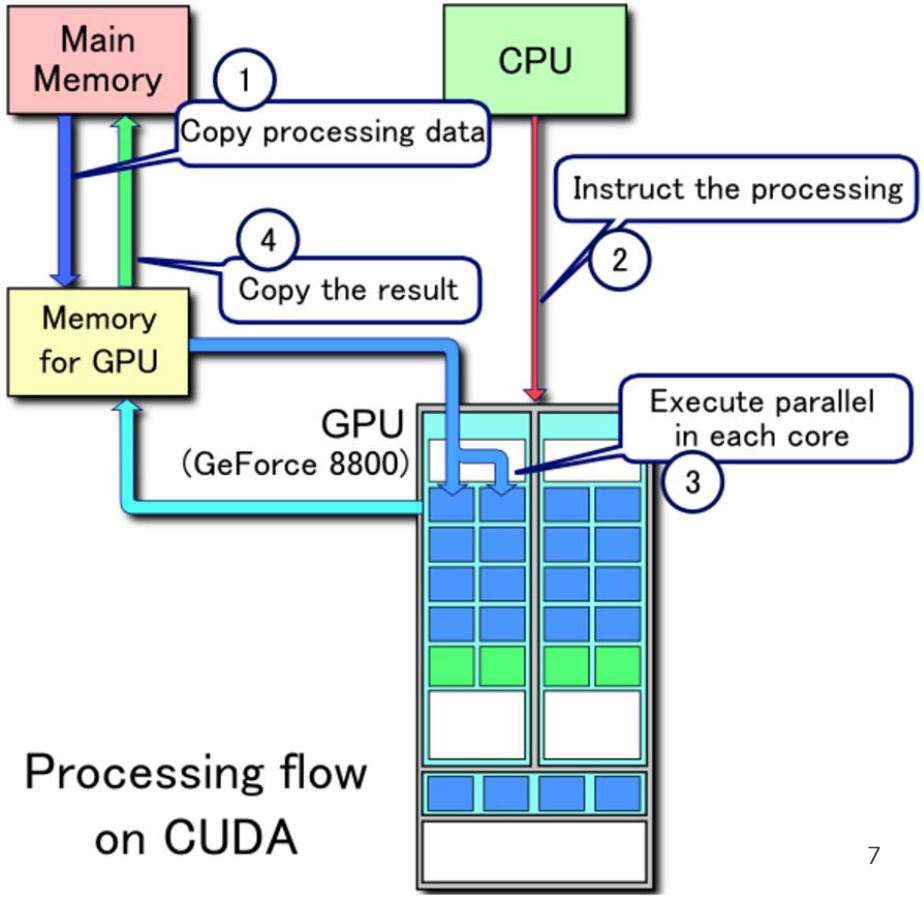
Fraction of non-distributed vs distributed deep learning over time



Use of hardware for deep learning

GPU (Graphical Processing Unit)

- Typically composed of thousands of logical cores
- Excels at matrix and vector operations
  - Gaming, rendering...and ML
- Main vendor: NVidia
  - CUDA: parallel computing platform and programming model



Options

- **NVIDIA CUDA**
  - Market leader, large user community, best support from DL frameworks
  - Can be used in python, C/C++, fortran, Matlab
- **AMD HIP**
  - Limited support for pyTorch and Tensorflow
  - Slightly behind in terms of performances
- **INTEL**
  - Xeon Phi: Poor support
  - Habana Gaudi-based AWS EC2 instances will be available in 2021!
- **Google TPU**
  - Good performances, more powerful than cloud GPUs
  - best for training, for prototype and inference better use cheaper alternatives
- **Amazon AWS and Microsoft Azure**
  - Powerful, easy to scale, expensive

How to choose hardware

- **GPUs** have high performance for floating point arithmetic
  - limited amount of memory available
  - rate at which data can be moved from CPU to GPU
  - memory bandwidth for GPUs must be seen relative to the amount of FLOPS: if one has more floating point units, a higher bandwidth is needed to keep them occupied
- **Matrix operations:** typically bandwidth cost is larger than multiplication cost
  - Particularly important for many small multiplications

How to choose hardware

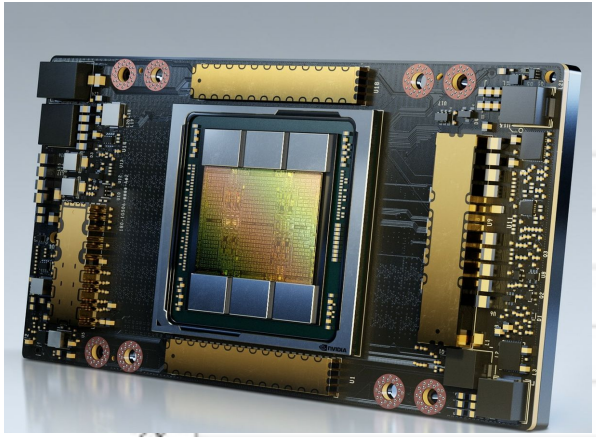
- CNN need a lot of computing power (need high number of cores, FLOPS)
- LSTM and RNN are made of small matrix multiplication: memory bandwidth matters
- Be aware of power consumption and overheating when placing multiple GPUs close to one another!

Performances

Architecture	INT8 [TOPS]	FP16 [TFLOPS]	FP32 [TFLOPS]	FP64 [TFLOPS]	Memory [GB]	Memory Bandwidth [GB/s]	PCIe [GB/s][a]	Proprietary Interconnect [GB/s][a]
AMD Instinct MI25 [65]	–	24.6	12.29	0.77	16	484	15.75	–
AMD Instinct MI50 [66]	53.6	26.8	13.4	6.7	16	1024	31.51	200 (2 x 100)
AMD Instinct MI60 [67]	58.9	29.5	14.7	7.4	32	1024	31.51	200 (2 x 100)
NVIDIA P100[b] [68] [69] [71]	–	21.2	10.6	5.3	16	732	15.75	160 (4 x 40)
NVIDIA V100[b] [71] [70] [72]	62.8[c]	31.4[c] / 125[d]	15.7	7.8	16 / 32	900	15.75	300 (6 x 50)
Intel Xeon Scalable 8180 (per socket) [73] [74]	–	–	3.0 / 4.2[e]	1.5 / 2.1[e]	768 (max)	119 (max)	15.75	–
AMD EPYC 7601 (per socket) [61] [62]	–	–	1.1 / 1.4[f]	0.56 / 0.69[f]	2000 (max)	159	15.75	–

[a] PCIe bandwidths are unidirectional. NVlink speeds are aggregated bidirectional bandwidth. For Infinity Fabric (IF) the bandwidth for one Infinity Fabric link is quoted; it is unspecified whether this is unidirectional, or aggregated bidirectional bandwidth.

NVIDIA GPUs



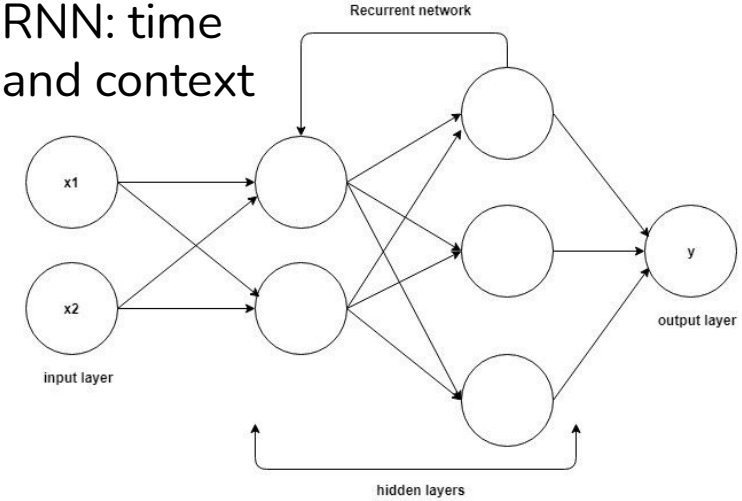
	Peak Performance
Transistor Count	54 billion
Die Size	826 mm^2
FP64 CUDA Cores	3,456
FP32 CUDA Cores	6,912
Tensor Cores	432
Streaming Multiprocessors	108
FP64	9.7 teraFLOPS
FP64 Tensor Core	19.5 teraFLOPS
FP32	19.5 teraFLOPS
TF32 Tensor Core	156 teraFLOPS | 312 teraFLOPS*
BFLOAT16 Tensor Core	312 teraFLOPS | 624 teraFLOPS*
FP16 Tensor Core	312 teraFLOPS | 624 teraFLOPS*
INT8 Tensor Core	624 TOPS | 1,248 TOPS*
INT4 Tensor Core	1,248 TOPS | 2,496 TOPS*
GPU Memory	40 GB
GPU Memory Bandwidth	1.6 TB/s
Interconnect	NVLink 600 GB/s PCIe Gen4 64 GB/s
Multi-Instance GPUs	Various Instance sizes with up to 7MIGs @5GB
Form Factor	4/8 SXM GPUs in HGX A100
Max Power	400W (SXM)

With its multi-instance GPU (MIG) technology, A100 can be partitioned into up to seven GPU instances, each with 10GB of memory.

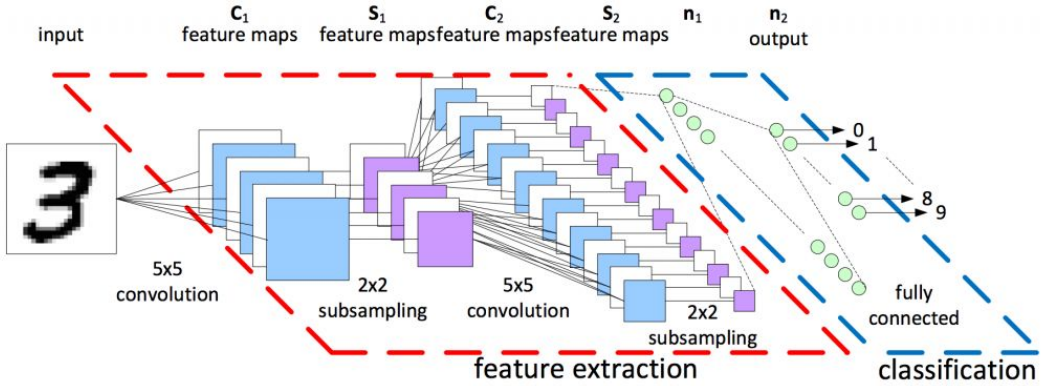
Parallelisation on multiple GPUs

- ‘Easy’ for RNNs and CNNs
- Fully connected networks with transformers poorer performances
- multiple GPUs can be used for tasks trivial to parallelise such as hyperparameter scan

RNN: time and context



CNN: image recognition



Local training

- Both model and data fit on a single machine (**multi cores + GPU**)
  - Multi-core processing:
    - *Embarrassingly parallel process*: use the cores to process multiple images at once, in each layer
    - Use multiple cores to perform SGD of multiple mini-batches in parallel.
  - Use GPU for computationally intensive subroutines like matrix multiplication.
  - Use both multi-core processing and GPU where all cores share the GPU and computationally intensive subroutines are pushed to the GPU.

Distributed training

- Data or model stored across multiple machines
- **Data parallelism**: data is distributed across multiple machines
  - data is too large to be stored on a single machine or to achieve faster training
  - **Synchronous update**: all loss gradients in a given mini-batch are computed using the same weights and full information of the average loss in a given mini-batch is used to update weights
  - **Asynchronous update**: as soon as a machine finishes computing updates, the parameters in the driver get updated. Any machine using the parameters will fetch the updated parameters from the server.

Data parallelism

- large batch of input data split across a collection of workers, each holding the full model
  - the forward pass involves no communication
  - the backward pass involves aggregating the gradients computed by each individual worker with respect to its separate part of the “global batch”
- scaling out:
  - the “global batch size” (i.e. the total number of examples across all workers that are seen in a single forward pass) increases
  - Affects convergence of DL algorithms and does not reach same accuracy levels that can be obtained with smaller batch sizes

Model parallelism

- model layers split across a collection of workers
- the batch size stays constant, and large models that would not fit the memory of a single device can be trained
- active communication also in the forward pass, thus requiring a lot more communication than a data parallel approach
- unless the advantages of model parallelism are absolutely critical (model doesn't fit on one machine), most research on large-scale training is done using data parallelism.
- Schemes involving a mix of data and model parallelism also exist: *hybrid parallelism*.

Hardware issues

- **I/O contention** with large data sets, or data sets with many small examples, and the data has to be physically stored on shared storage facilities.
- **Communication** bottlenecks during gradient-aggregation, due to high ratio between the number of parameters and amount of computation
- **Memory** bottlenecks for large networks, particularly when using memory-limited GPUs
- On large system such as **HPCs**, data are typically stored on shared file systems
    - Might be faster to copy data locally to worker node

Architectural choices

- **Your model doesn’t fit in the GPU memory?**
  - tune the model e.g. by reducing its connectivity (if that is acceptable) to fit the hardware
  - use model parallelism to distribute the model over multiple GPUs
  - use a data parallel approach on CPUs (if it does fit in CPU memory)
- **Your dataset doesn’t fit in the GPU memory?**
  - Make sure your data arrives fast enough to the GPU, otherwise revert to CPU
- Do you have access to a **system with a few GPU nodes, but thousands of CPUs**?
  - Development stage: run a limited number of examples (potentially even with a smaller model) on a single GPU, which allows quick development cycles
  - Production: may require CPUs because of memory constraints

Requirements for ML framework

- Run any ML algorithm of our choice in parallel
- Large datasets to be processed
- Multi-tenancy, so different users can request simultaneously ML pipelines
- An efficient resource management system for the cluster
- Support heterogeneous architectures
    - CPUs, GPUs, ...

ML as a Service (MLaaS)

CHALLENGES

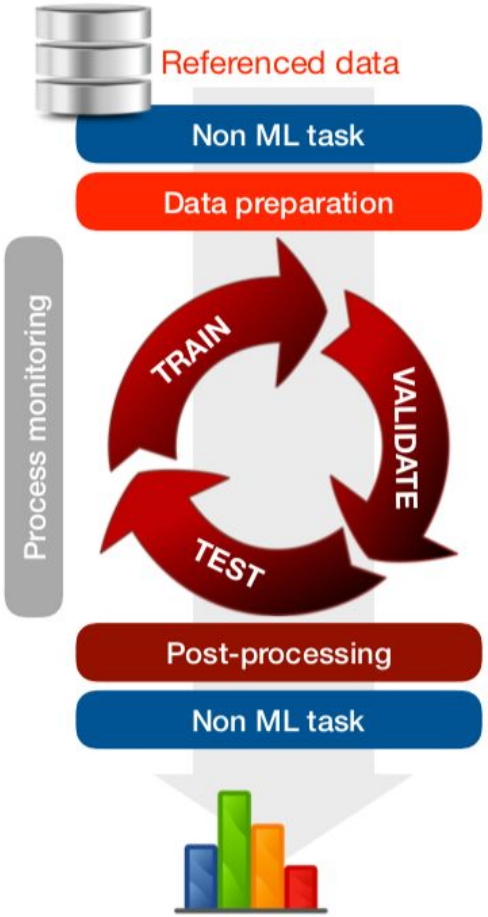
- Reconstruction
- Analysis
- Trigger
- Data quality
- Detector monitoring
- Computing operations
- Monte Carlo tuning
- ...

REQUIREMENTS

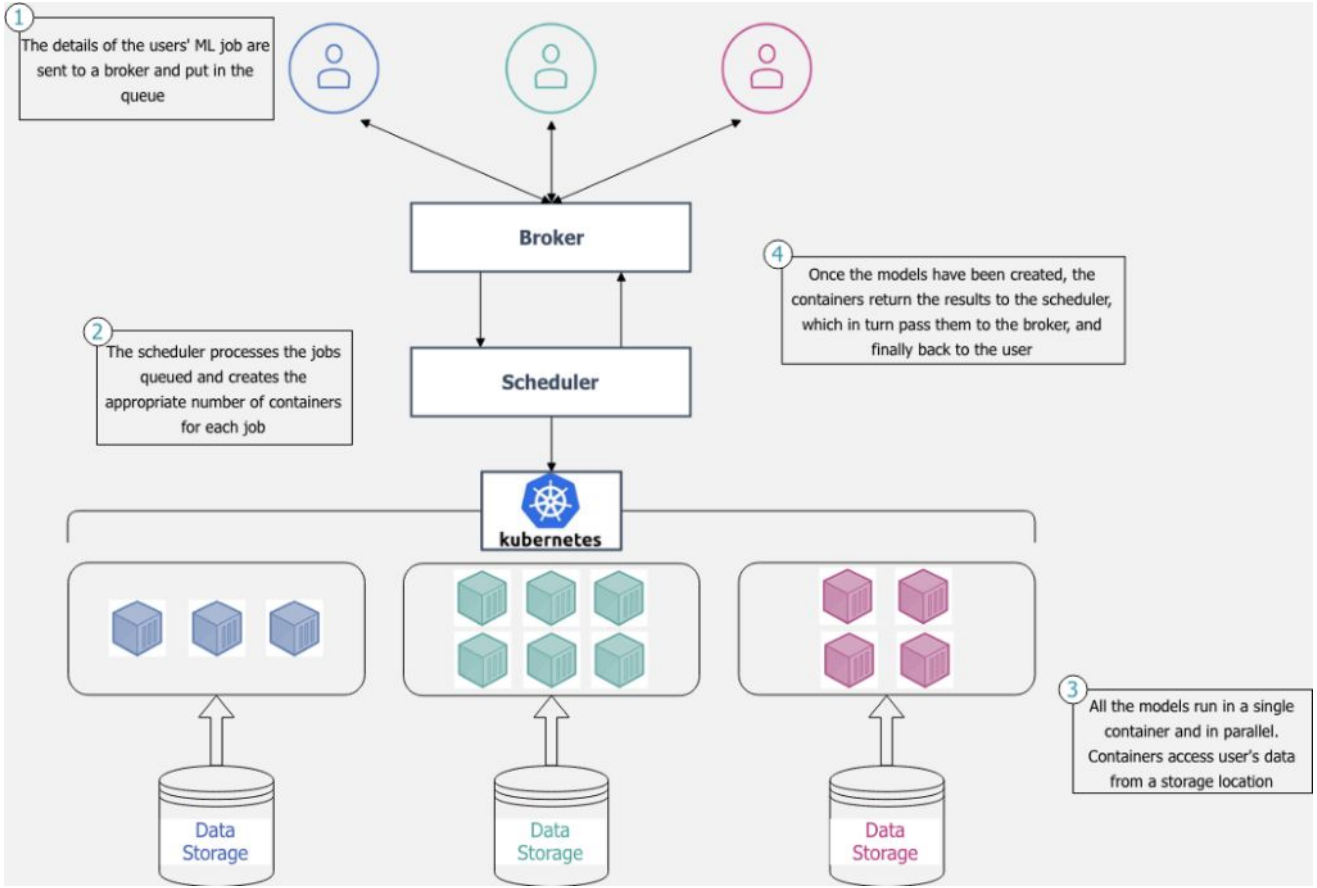
- Workflow definition
  - Results reproducibility
- Multi-tenancy (scheduling, authentication...)
- Parallel execution and scaling
- Data handling
- Ease of use and management
- ...

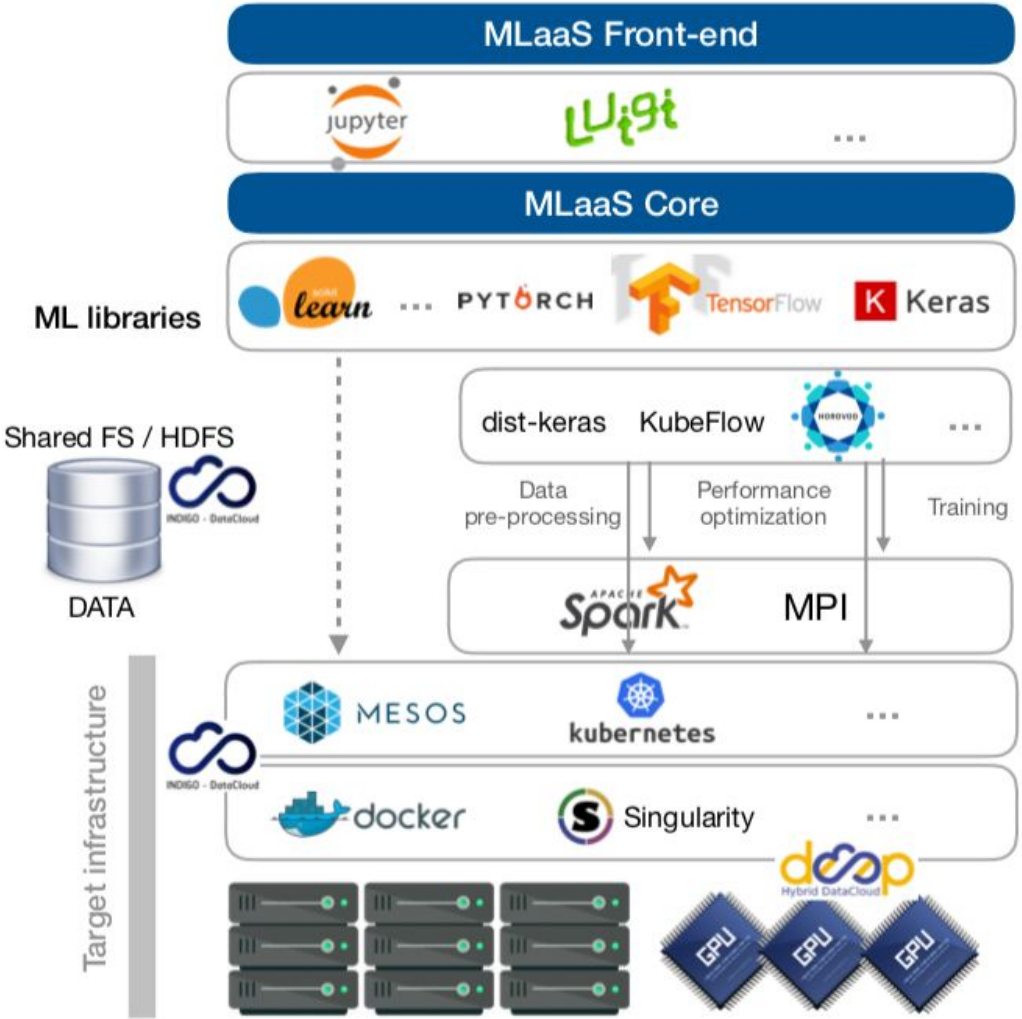
IMPLEMENTATION

- Lightweight virtualization
- Modularity
- Flexibility
- Heterogeneous back-end infrastructures
- ...



Example architecture





- Workflow definition
- Process Monitoring
- Authentication

Deep Learning framework

Distributed DL libraries

Cluster framework (parallelize task)

Orchestrator (schedule on resources)

Packetization and virtualization

Resources:

- Bare metal
- Infrastructure as a Service

Hands-on test

- Evaluation might be based on:
  - Improve ML model trying to improve performances vs training time
  - Measure scaling performances vs complexity
- May include cross-validation and hyper-parameter tuning:
  - K-fold training
  - Grid search

Hands-on test

1. **Point your browser to:** https://yoga.to.infn.it
2. **Open a terminal:**
   - `cd MLCourse-2021`
   - `git pull`
   - `cp Notebooks/Day4/* ../`
3. **From JupyterHub Home tab:**
   - start and run *BigDL.ipynb*

- *Deep learning with* BigDL
  - FF NN

Useful commands

From the terminal (replace username accordingly):

- `kubectl get pods`
- `kubectl describe pod jupyter-leggerf-1571836347893-exec-1`
- `kubectl describe node vdummy06.to.infn.it`
- `kubectl get nodes`
- `kubectl describe farm`

To remove pods in error state:

- `kubectl get pods -n leggerf | grep Error | awk '{print $1}' | xargs kubectl delete pod -n leggerf`