



Deep Learning Optimized - Jean Zay

Introduction – Jean Zay – GPU



INSTITUT DU
DÉVELOPPEMENT ET DES
RESSOURCES EN
INFORMATIQUE
SCIENTIFIQUE



DLO-JZ presentation

IDRIS ◀

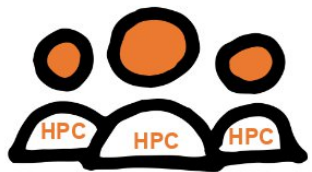
Agenda ◀

Presentation of the participants ◀

System



User Assistance



11 ingénieur·e·s



12 ingénieur·e·s

BLOOM on Jean Zay

Pre-training :
117 days (2022)
384 x 80GB A100 GPUs



Train modulaire de formations IDRIS

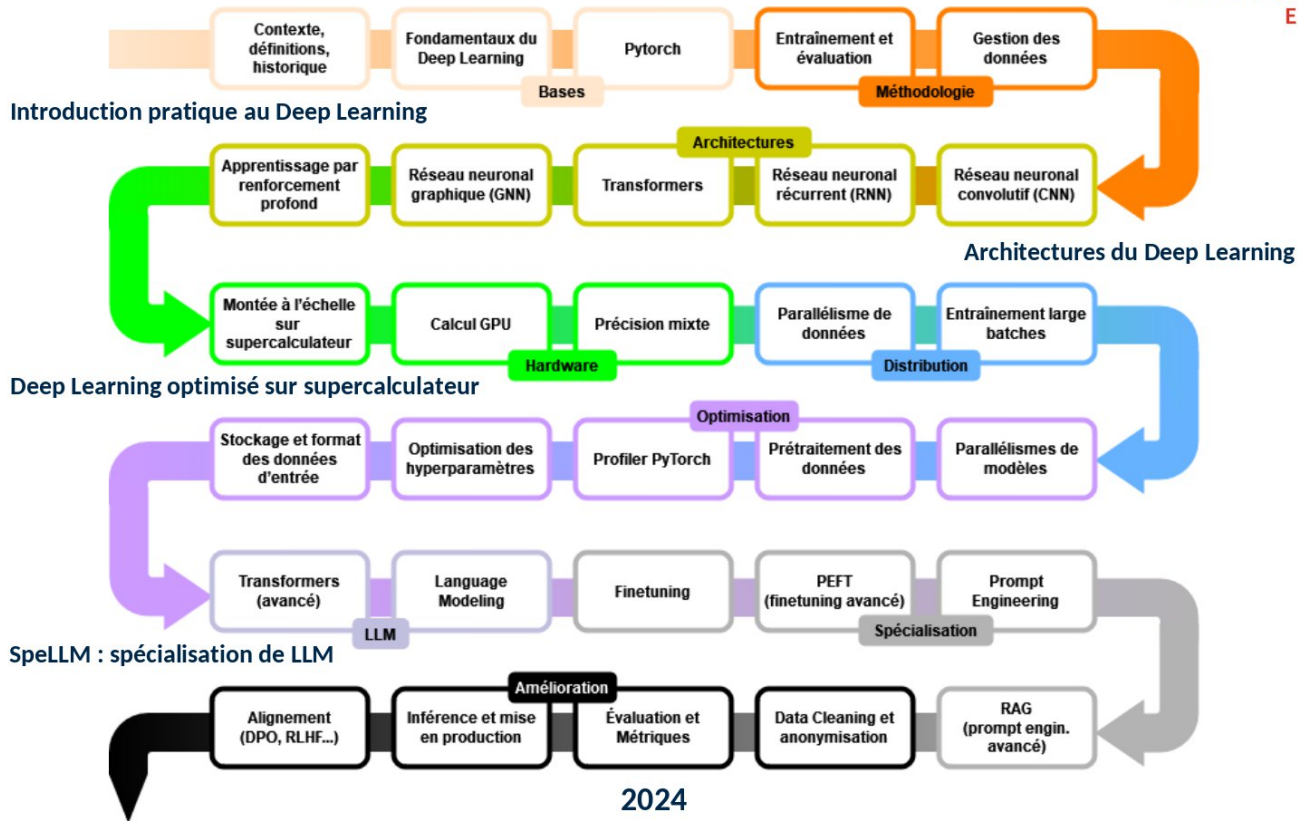


IDRIS

Les formations IA

Pour les inscriptions ou une formation sur mesure contacter

CNRS FORMATION
ENTREPRISES





DLO-JZ

Super Computing 21 Event



SC21

St. Louis,
MO | science
& beyond.

Featured Post

PyTorch 2.5 Release Blog

We are excited to announce the release of PyTorch® 2.5 (release note)!

This release features a ne...

[Read More](#) >

October 15, 2024

The Path to Achieve PyTorch Performance Boost on Windows CPU

The challenge of PyTorch's lower CPU performance on Windows compared to Linux has been a significant issue. There are multiple factors leading to this performance disparity. Through our investigation, we've identified several reasons for poor CPU performance on Windows, two primary issues have been pinpointed: the inefficiency of the Windows default malloc memory allocator and the absence of SIMD for vectorization optimizations on the Windows platform. In this article, we show how PyTorch CPU...

[Read More](#) >

October 08, 2024

Overview

Installing cuDNN on Linux

Installing cuDNN on Windows

Building and Running a cuDNN
Dependent Program

Cross-Compiling cuDNN Samples

BACKEND API

Backend API Overview

cuda_graph Library

cuda_ops Library

cuda_cnn Library

cuda_adv Library

DEVELOPER GUIDE

Overview

Core Concepts

cuDNN Handle

Tensors and Layouts

Tensor Core Operations

Graph API

Legacy API

Compatibility

Odds and Ends

REFERENCE

Support Matrix

Troubleshooting

Documentation Archives

Software License Agreement

Acknowledgements

Notices

cuda9.5.0.x

» Core Concepts

Core Concepts

Before we discuss the details of the graph and legacy APIs, this section introduces the key concepts that are common to both.

cuDNN Handle

The cuDNN library exposes a host API but assumes that for operations using the GPU, the necessary data is directly accessible from the device.

An application using cuDNN must initialize a handle to the library context by calling `cudaCreate()`. This handle is explicitly passed to every subsequent library function that operates on GPU data. Once the application finishes using cuDNN, it can release the resources associated with the library handle using `cudaDestroy()`. This approach allows the user to explicitly control the library's functioning when using multiple host threads, GPUs, and CUDA streams.

For example, an application can use `cudaSetDevice` (prior to creating a cuDNN handle) to associate different devices with different host threads, and in each of those host threads, create a unique cuDNN handle that directs the subsequent library calls to the device associated with it. In this case, the cuDNN library calls made with different handles would automatically run on different devices.

The device associated with a particular cuDNN context is assumed to remain unchanged between the corresponding `cudaCreate()` and `cudaDestroy()` calls. In order for the cuDNN library to use a different device within the same host thread, the application must set the new device to be used by calling `cudaSetDevice` and then create another cuDNN context, which will be associated with the new device, by calling `cudaCreate()`.

Tensors and Layouts

Whether using the graph API or the legacy API, cuDNN operations take tensors as input and produce tensors as output.

Tensor Descriptor

The cuDNN library describes data with a generic n-D tensor descriptor defined with the following parameters:

- > a number of dimensions from 3 to 8
- > a data type (32-bit floating-point, 64 bit-floating-point...)
- > an integer array defining the size of each dimension
- > an integer array defining the stride of each dimension (for example, the number of elements to add to reach the next element from the same dimension)

This tensor definition allows, for example, to have some dimensions overlapping each other within the same tensor by having the stride of one

 **Hugging Face**

Transformers

Ctrl+K

V4.41.3 EN 127,325

Interoperability with GGUF files

PERFORMANCE AND SCALABILITY

Overview

LLM inference optimization

Quantization

EFFICIENT TRAINING TECHNIQUES

Methods and tools for efficient training on a single GPU

Multiple GPUs and parallelism

Fully Sharded Data Parallel

DeepSpeed

Efficient training on GPU

Distributed CPU training

Training on TPU with TensorFlow

PyTorch training on Apple silicon

Custom hardware for training

Hyperparameter Search using Trainer API

OPTIMIZING INFERENCE

CPU Inference

GPU Inference

Instantiate a big model

Efficient Training on Multiple GPUs

If training a model on a single GPU is too slow or if the model's weights do not fit in a single GPU's memory, transitioning to a multi-GPU setup may be a viable option. Prior to making this transition, thoroughly explore all the strategies covered in the [Methods and tools for efficient training on a single GPU](#) as they are universally applicable to model training on any number of GPUs. Once you have employed those strategies and found them insufficient for your case on a single GPU, consider moving to multiple GPUs.

Transitioning from a single GPU to multiple GPUs requires the introduction of some form of parallelism, as the workload must be distributed across the resources. Multiple techniques can be employed to achieve parallelism, such as data parallelism, tensor parallelism, and pipeline parallelism. It's important to note that there isn't a one-size-fits-all solution, and the optimal settings depend on the specific hardware configuration you are using.

This guide offers an in-depth overview of individual types of parallelism, as well as guidance on ways to combine techniques and choosing an appropriate approach. For step-by-step tutorials on distributed training, please refer to the [Accelerate documentation](#).

While the main concepts discussed in this guide are likely applicable across frameworks, here we focus on PyTorch-based implementations.

Before diving deeper into the specifics of each technique, let's go over the rough decision process when training large models on a large infrastructure.

Scalability strategy

Begin by estimating how much vRAM is required to train your model. For models hosted on the  Hub, use our [Model Memory](#)

Efficient Training on Multiple GPUs

- Scalability strategy
- Data Parallelism
 - DataParallel vs DistributedData Parallel
- ZeRO Data Parallelism
- From Naive Model Parallelism to Pipeline Parallelism
- Tensor Parallelism
- Data Parallelism + Pipeline Parallelism
- Data Parallelism + Pipeline Parallelism + Tensor Parallelism
- ZeRO Data Parallelism + Pipeline Parallelism + Tensor Parallelism
- FlexFlow
- GPU selection
 - Number of GPUs
 - Order of GPUs

12

Agenda – Covered topics

Day 1

- Jean Zay
- Code review
- The challenges of scaling up
- **GPU computing**
- **Tensor Cores**
- Pytorch Profiler

Day 2

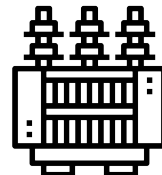
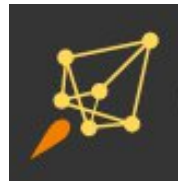
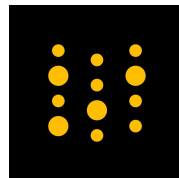
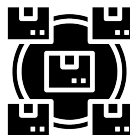
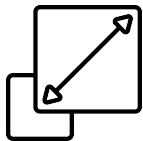
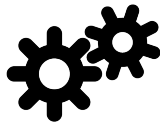
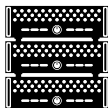
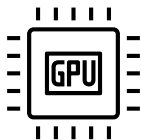
- **DataLoader optimizations**
- **Distribution - Data Parallelism** ♥
- Data storage and formats

Day 3

- JIT
- **Training and large batches**
- HyperParameter Optimization

Day4

- Visualization tools
- **Model parallelisms**
- Model parallelisms API
- Good practices



Practical Workshop

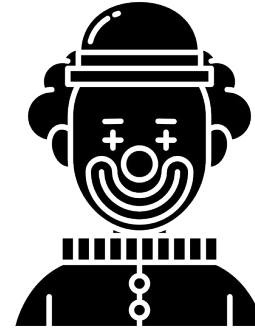
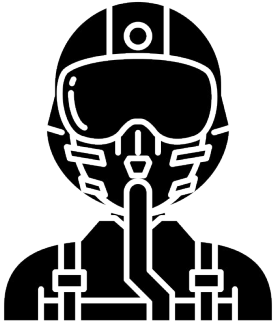
- Day 1, 2, 3, 4 :
 - System Optimization : GPU, Mixed Precision, Data Parallelism
 - Profiler
 - DataLoader
 - *Optimizers & LR scheduler*
 - *Hyper-Parameters Optimization (HPO)*
 - *FSDP (New)*



- Day 4 (à la carte) :
 - *Model parallelism* with Huge Model
 - *Tensor parallelism & 2D parallelism* from scratch
 - Data Augmentation
 - torch.compile & torchscript



Présentation des participant·e·s



Jean Zay

Supercomputer ◀

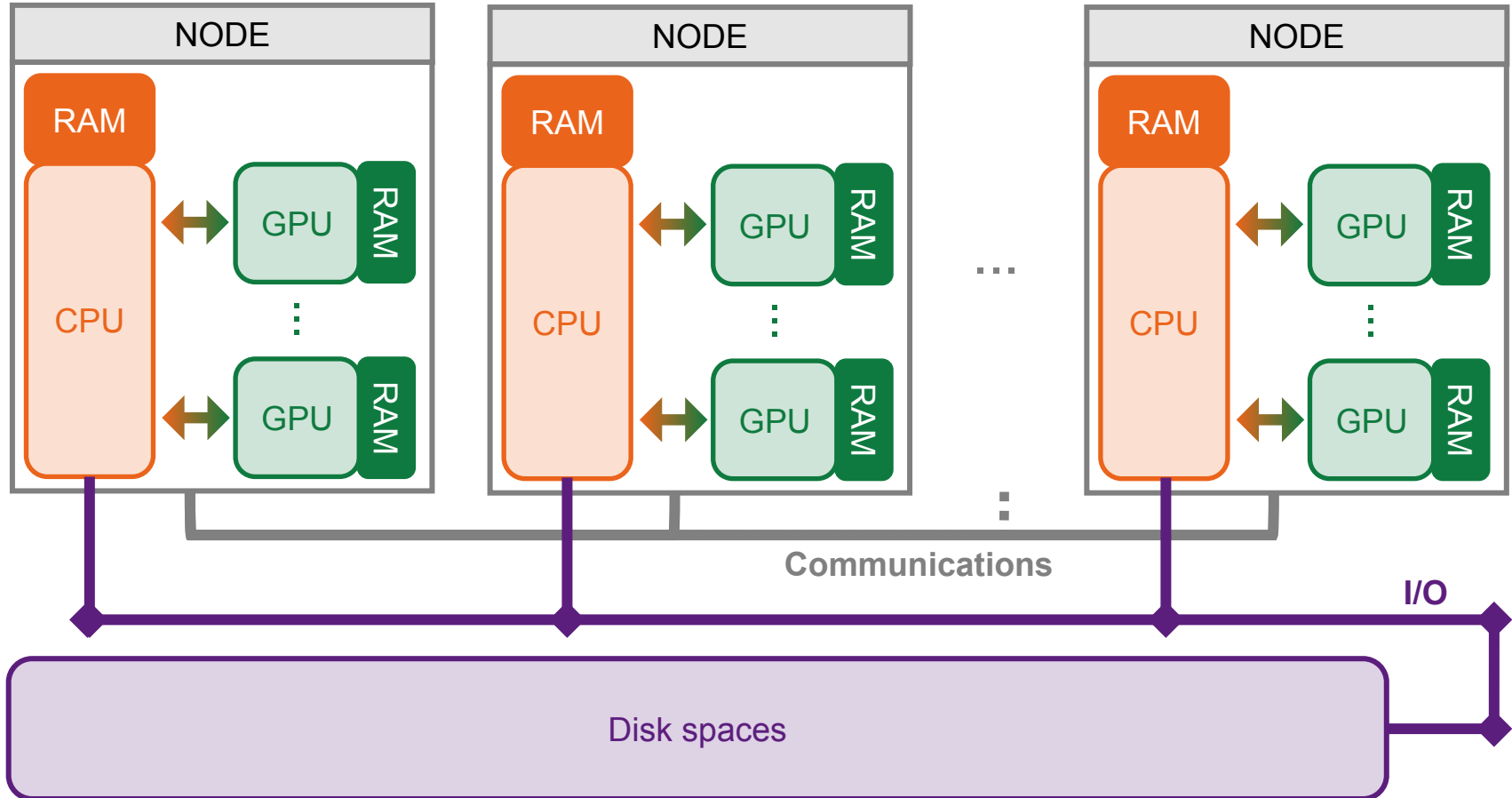
Jean Zay ◀

Job submission with Slurm ◀

JupyterHub on Jean Zay ◀

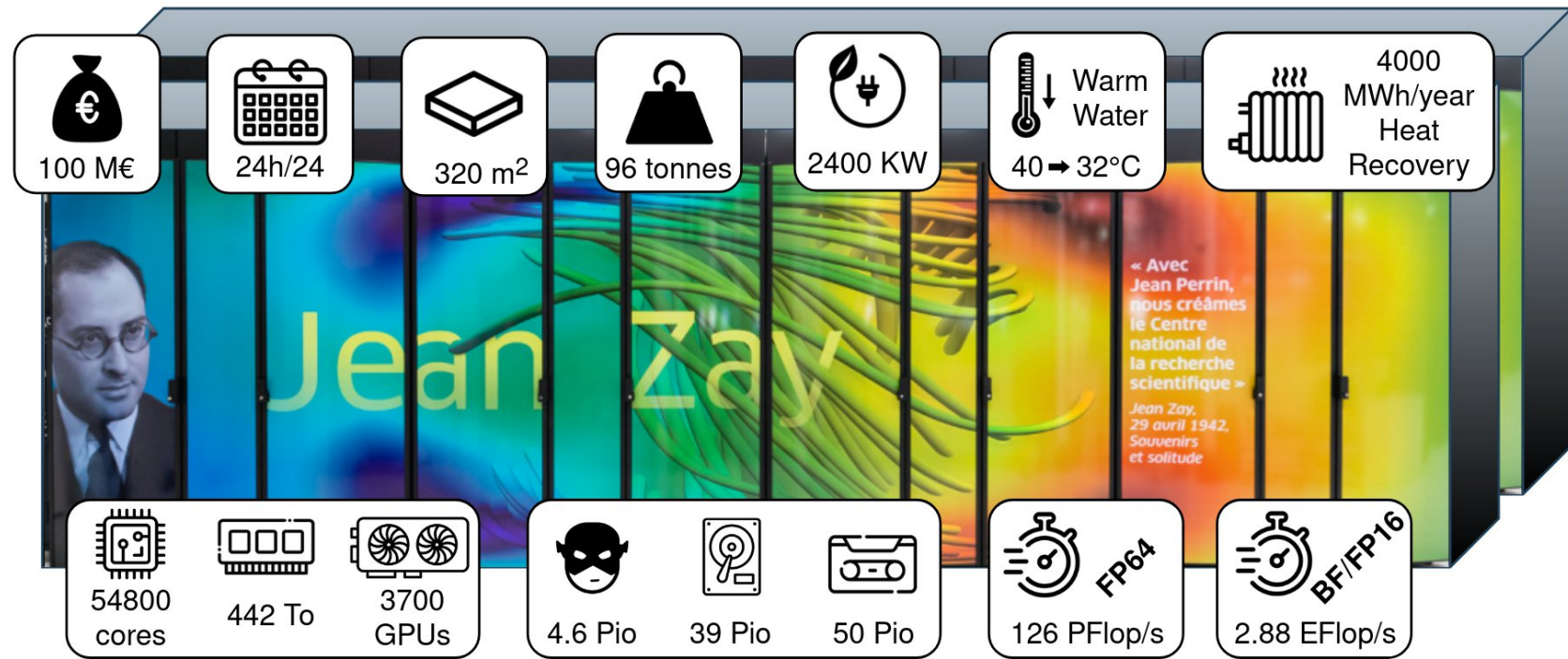
Slurm tools for python notebooks ◀

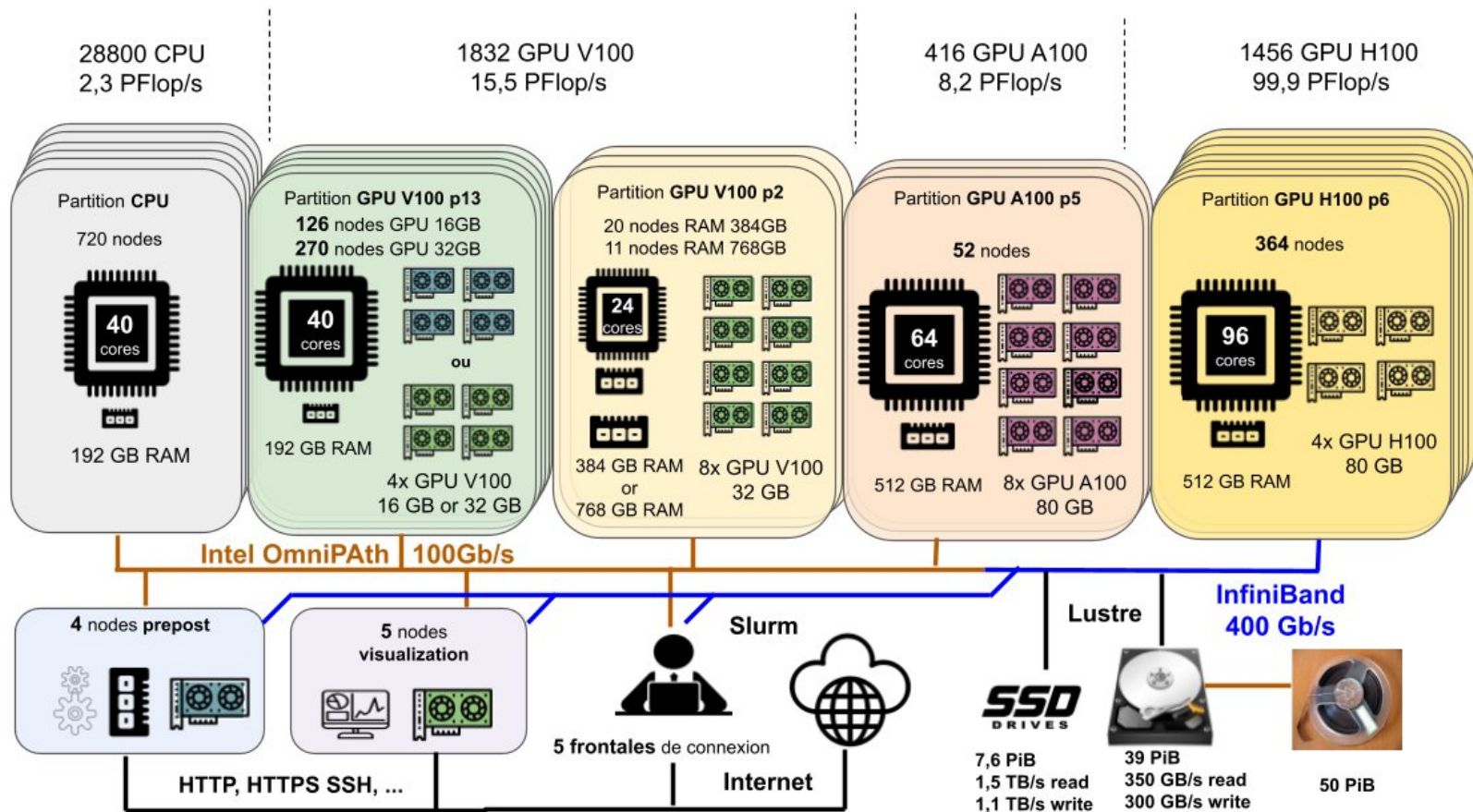
What's a supercomputer?



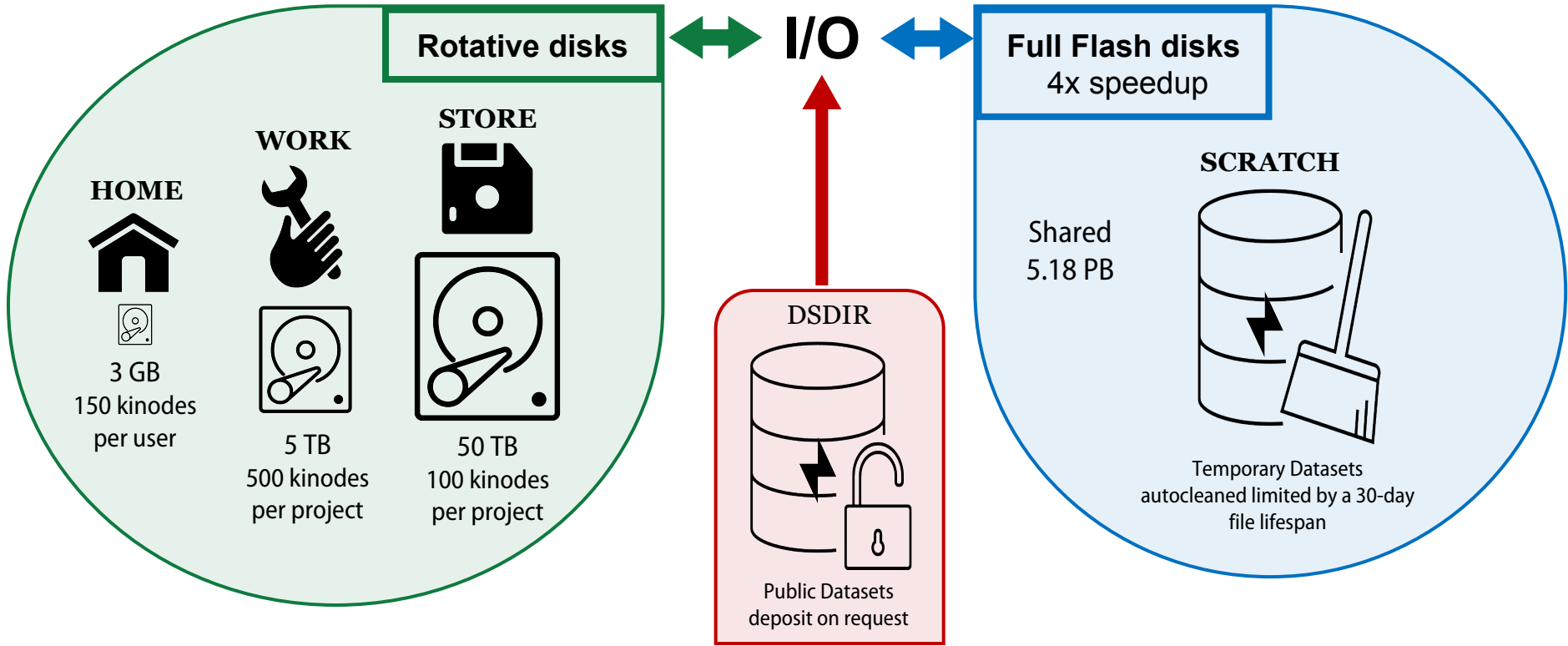
Jean Zay

First national converged supercomputer dedicated to Artificial Intelligence (AI) and High Performance Computing (HPC)





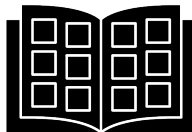
Jean Zay: Storage spaces



Jean Zay: Work environment



Catalogue of shared modules (conda environments)



- Installed by IDRIS
- Completed on request

```
login@jean-zay3:~$ module load pytorch-gpu/py3/1.11.0
Loading requirement: ...
(pytorch-gpu-1.11.0+py3.9.12) login@jean-zay3:~$
```

- Customizable

```
~$ pip install --user --no-cache-dir <paquet>
```



Conflicts between versions
Storage spaces saturation

Personal conda environments

```
login@jean-zay3:~$ module load anaconda-py3/2023.03
(base) login@jean-zay3:~$ conda create -n myenv
```



Storage spaces saturation ++

Singularity containers

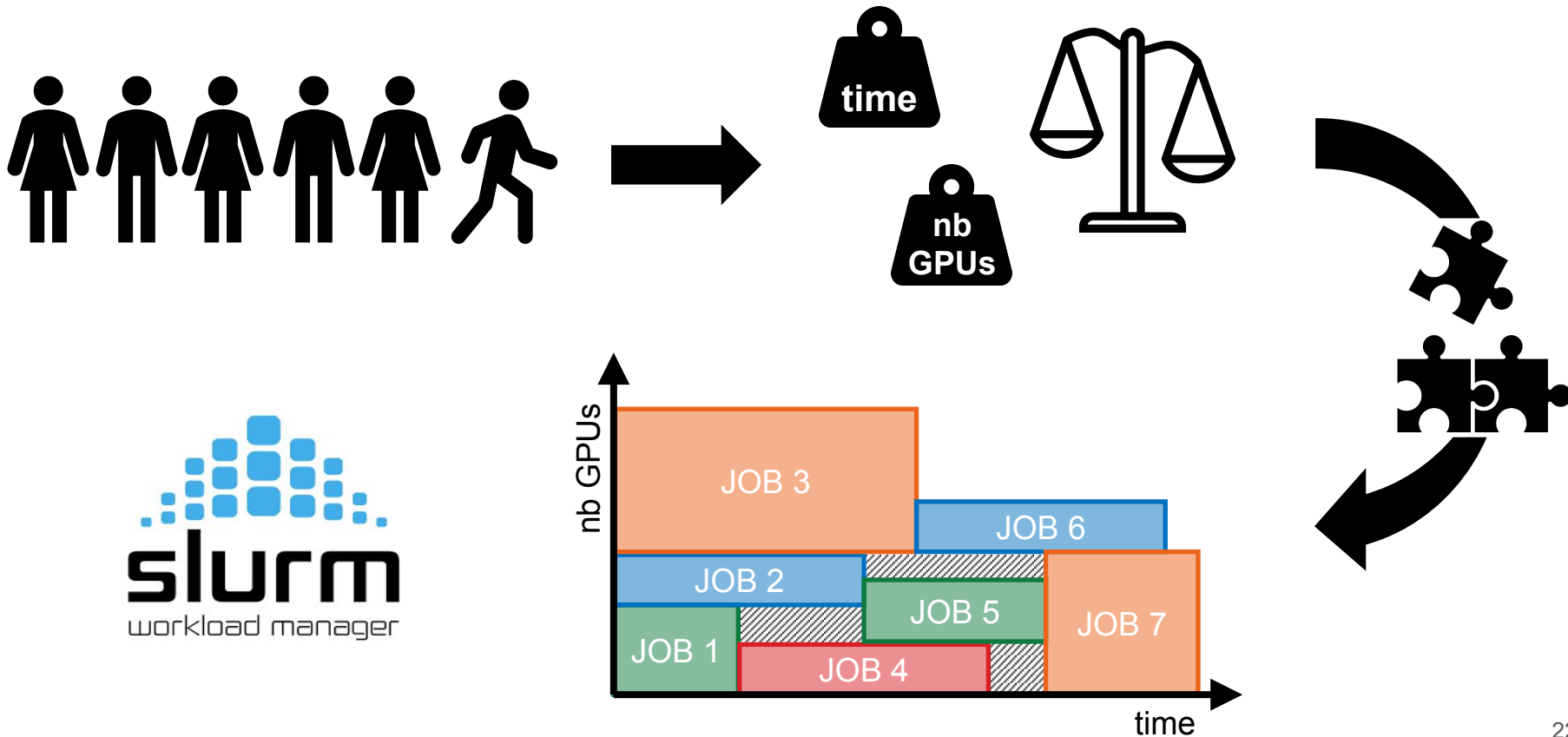
```
login@jean-zay3:~$ module load singularity
```

Import SIF images on Jean Zay

- From your PC
- From public deposits
- Possible conversion from docker

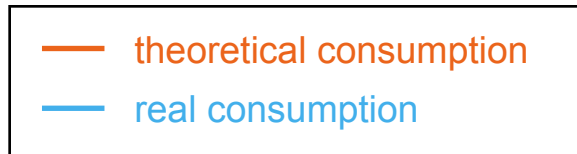
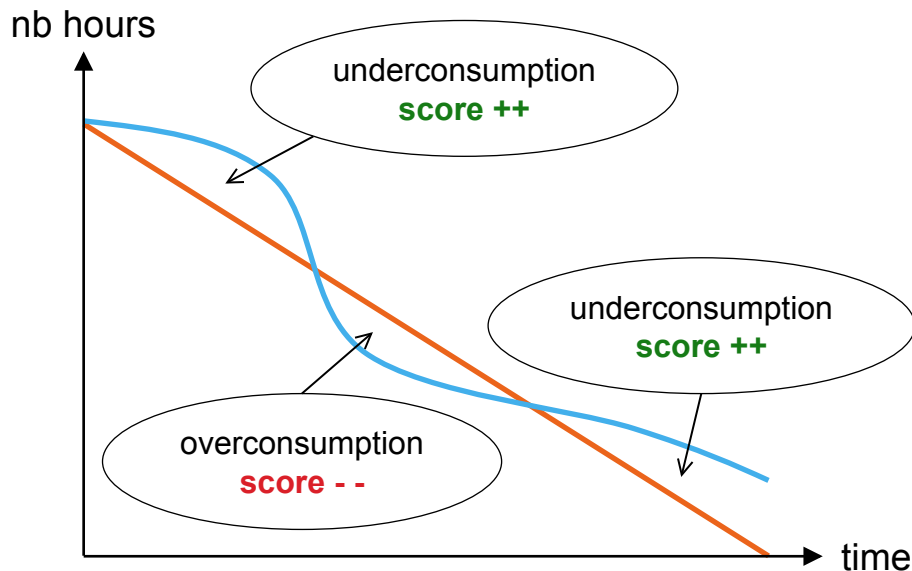


Job submission with Slurm



Job submission with Slurm

Slurm gives your job a **priority score** depending on your consumption.



Slurm compares all users' scores to define job position in the queue.

Job submission with Slurm

Adjust the **QoS (Quality of Service)** to improve the priority score of your job!

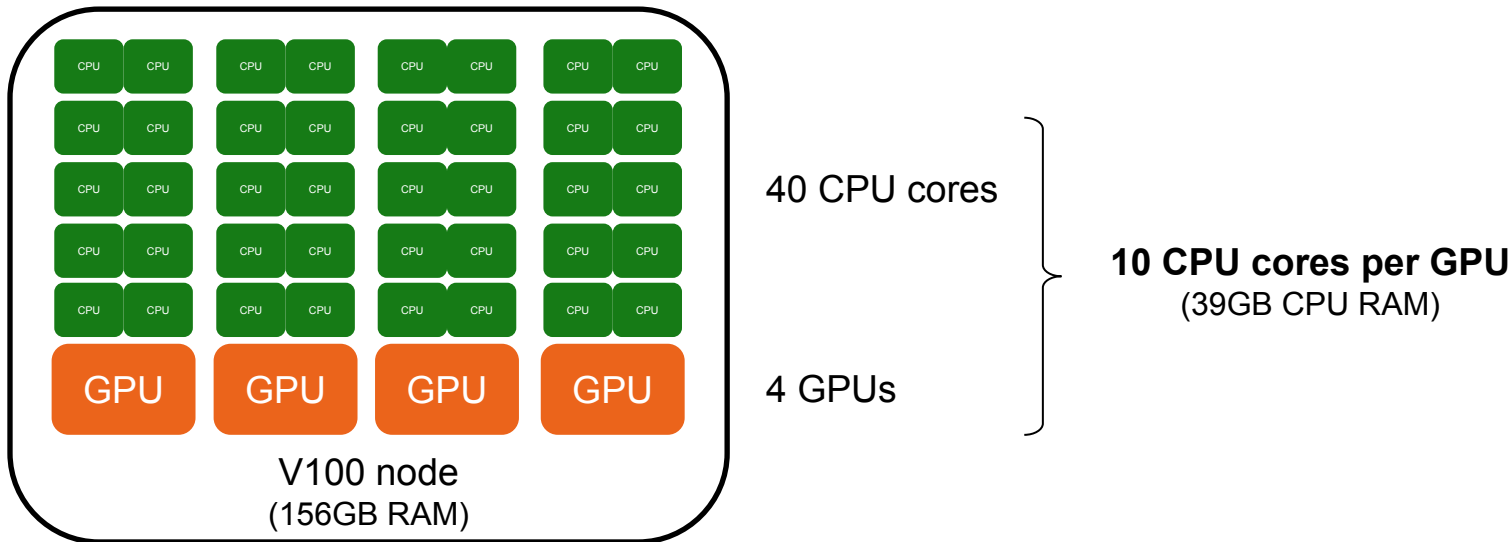
	QoS	Max elapsed time	Resource limits				
			Per job	Per user	Per project	Per QoS	
+ ↑ priority ↓ -	qos_gpu-dev	2h	32 GPUs	32 GPUs (10 jobs max at the same time)	32 GPUs	512 GPUs	} dev
	qos_gpu-t3 (default)	20h	512 GPUs	512 GPUs	512 GPUs		
	qos_gpu-t4 (V100)	100h	16 GPUs	96 GPUs	96 GPUs	256 GPUs	} prod

And keep your job in the queue, its priority score will increase with time.

Job submission with Slurm

How to configure my job?

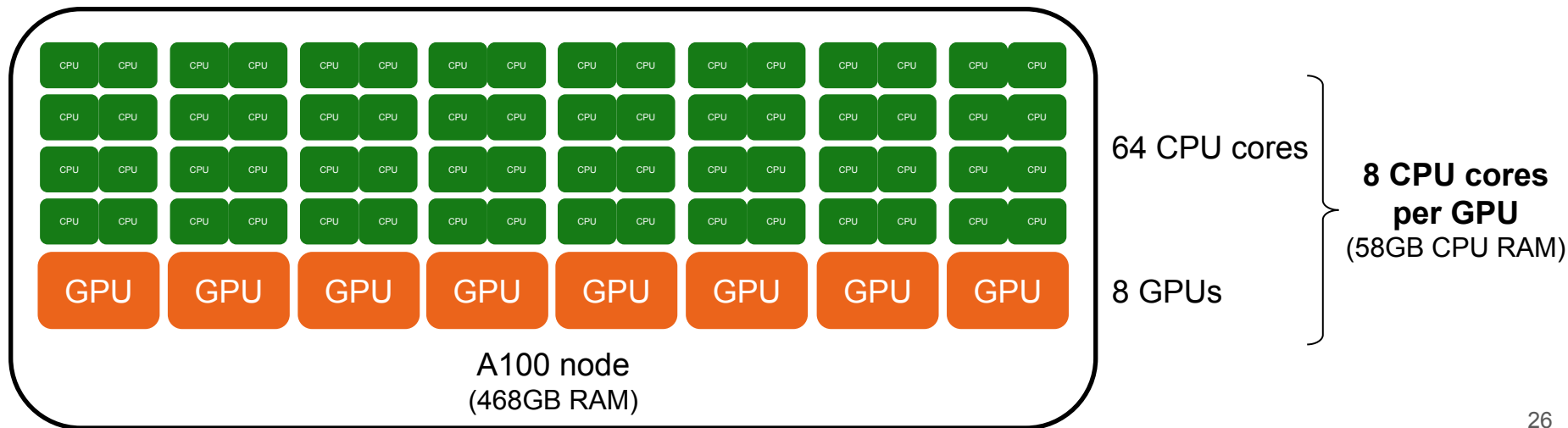
- How many GPUs?
- How many time?
- How many CPUs per GPU? → **1 CPU core = 3.9GB RAM** (on V100 nodes)
→ interesting to have **several CPU cores to feed a GPU**
(cf “DataLoader optimizations” part of this course)



Job submission with Slurm

How to configure my job?

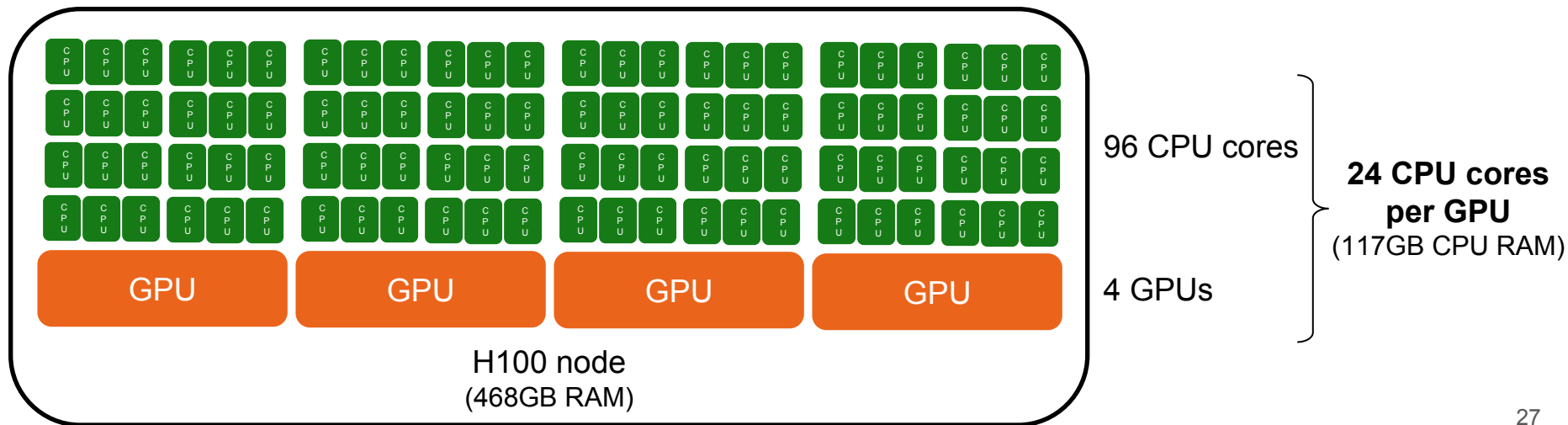
- How many GPUs?
- How many time?
- How many CPUs per GPU? → **1 CPU core = 7.3GB RAM** (on A100 nodes)
→ interesting to have **several CPU cores to feed a GPU**
(cf “DataLoader optimizations” part of this course)



Job submission with Slurm

How to configure my job?

- How many GPUs?
- How many time?
- How many CPUs per GPU? → **1 CPU core = 4.87GB RAM** (on H100 nodes)
→ interesting to have **several CPU cores to feed a GPU**
(cf “DataLoader optimizations” part of this course)



Job submission with Slurm

Example: reservation of 2 x 8 A100 GPUs

script.slurm

```
#!/bin/bash

#SBATCH --job-name="dlojz"           # name of the job
#SBATCH --output="dlojz%j.out"      # output file
#SBATCH --error="dlojz%j.err"       # error file
#SBATCH --nodes=2                   # nb of nodes
#SBATCH --gres=gpu:8                 # nb of GPUs/node
#SBATCH --ntasks-per-node=8          # nb of tasks/node
#SBATCH --cpus-per-task=8            # nb of CPU cores/task
#SBATCH --hint=nomultithread         # no hyperthreading
#SBATCH --time=01:00:00              # max execution time
#SBATCH --qos=qos_gpu_a100-dev      # adjust QoS

module load pytorch-gpu/py3/2.6.0   # environment

srun python script.py               # run script
```


Job submission with Slurm

script.slurm

```
#!/bin/bash
```

```
#SBATCH --job-name="dlojz"  
#SBATCH --output="dlojz%j.out"  
#SBATCH --error="dlojz%j.err"  
#SBATCH --nodes=2  
#SBATCH --gres=gpu:4  
#SBATCH --ntasks-per-node=4  
#SBATCH --cpus-per-task=10  
#SBATCH --hint=nomultithread  
#SBATCH --time=01:00:00  
#SBATCH --qos=qos_gpu-dev
```

```
module load pytorch-gpu/py3/2.4.0
```

```
srun python script.py
```

```
login@jean-zay3:~$ sbatch script.slurm
```

Job submission



Waiting in queue

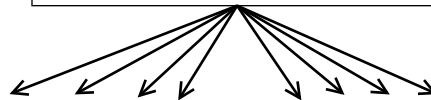
```
login@jean-zay3:~$ squeue --me
```

JOBID	PARTITION	NAME	USER	ST	PD	TIME	NODES	ODELIST(Reason)
223225	gpu_p13	dlojz				0:00	2	(Priority)



Launching job

```
srun python script.py
```



NODE 0

GPU0

GPU1

GPU2

GPU3

NODE 1

GPU0

GPU1

GPU2

GPU3

JupyterHub on Jean Zay



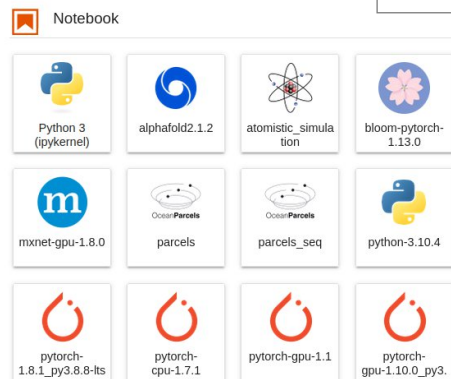
1. Authentication on <https://jupyterhub.idris.fr>

A screenshot of the JupyterHub sign-in interface. It has an orange header with the text "Sign in". Below the header, there are two input fields: "Username:" with the text "user" and "Password:" with masked characters "*****". At the bottom is an orange button labeled "Sign in".

2. Choose and configure an instance

A screenshot of the "JupyterLab Spawner Options" dialog box. It contains two buttons: "Interactive" and "SLURM". Arrows point from these buttons to the text "Run on a connection node" and "Run on a compute node" respectively.

3. Choose a kernel
(pytorch-gpu-2.2.0)



Slurm tools for python notebooks

```
from idr_pytools import gpu_jobs_submitter
```

```
command = 'dlojz.py --batch-size 128 --image_size 176'
```

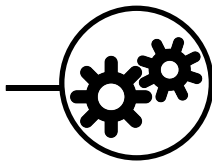
```
n_gpu = 8
```

```
MODULE = 'pytorch-gpu/py3/2.2.0'
```

```
name = 'dlojz'
```

```
jobid = gpu_jobs_submitter(command, n_gpu, MODULE, name=name, account='xyz@a100', time_max='05:00:00')
```

command
n_gpu
MODULE
name
account
time_max



script.slurm

```
#!/bin/bash

#SBATCH --job-name="dlojz"
#SBATCH --output="dlojz%j.out"
#SBATCH --error="dlojz%j.err"
#SBATCH --nodes=2
#SBATCH --gres=gpu:8
#SBATCH -C a100
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=8
#SBATCH --hint=nomultithread
#SBATCH --time=05:00:00
#SBATCH --account=xyz@a100

module load pytorch-gpu/py3/2.1.1

srun python dlojz.py --batch-size 128 --image_size 176
```

\$ sbatch script.slurm

jobid

Slurm tools for python notebooks

```
from idr_pytools import display_slurm_queue
```

```
name = 'dlojz'  
display_slurm_queue(name)
```



```
$ squeue --me -n <name>
```

```
from idr_pytools import search_log
```

```
jobid = ['12345']
```

```
search_log(contains=jobid)[0]
```



output filename

```
search_log(contains=jobid, with_err=True)[0]
```



error filename

The Challenges of Scaling

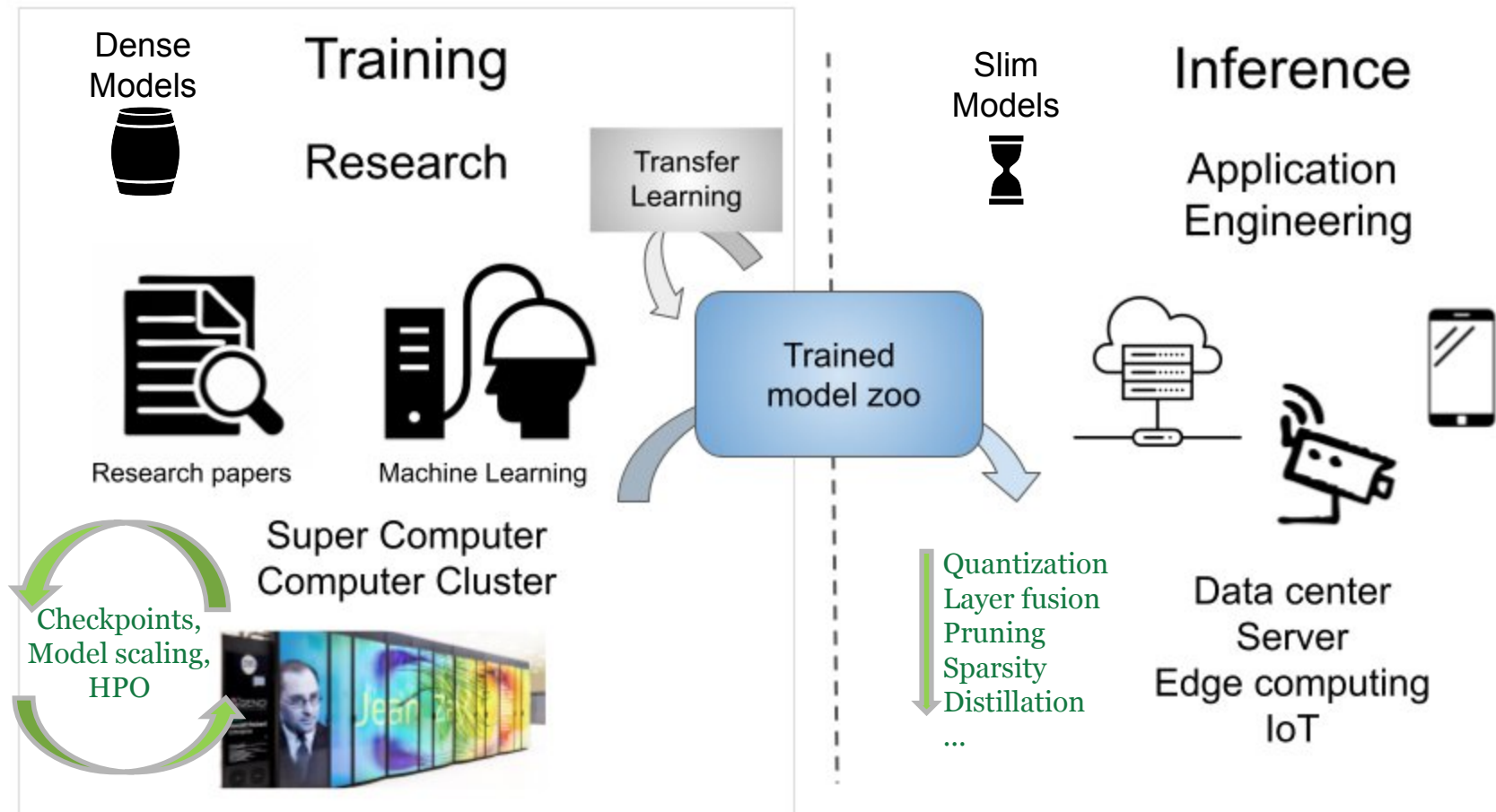
Training Time ◀

Memory Footprint ◀

Solutions ◀

Energy saving ◀

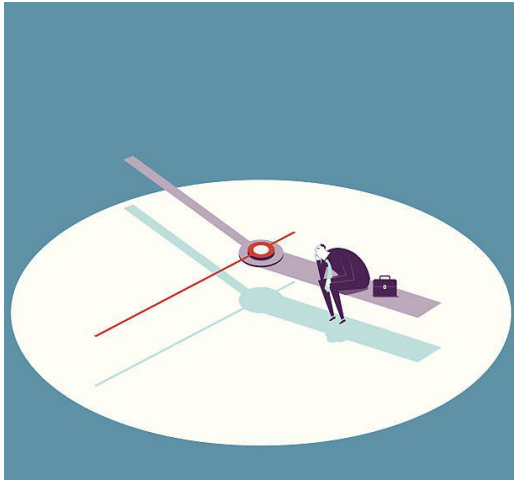
Training / Inference



Constraints of Deep Learning

2 problems to deal with:

Training Time



Memory Overconsumption (OOM)



Training Time

Training Resnet-50 on Imagenet



Goal:

Classification (1000 classes)

Model :

25 M parameters

Dataset:

1,2 M labeled images

14 Days

1 NVIDIA M40 GPU

Training Time

Training Resnet-50 on Imagenet

Facebook Caffe2	UC Berkeley, TACC, UC Davis Tensorflow	Preferred Network ChainerMN	Tencent TensorFlow	Sony Neural Network Library (NNL)	Fujitsu MXNet
1 hour	31 mins	15 mins	6.6 mins	2.0 mins	1.2 mins
Tesla P100 x 256	1,600 CPUs	Tesla P100 x 1,024	Tesla P40 x 2,048	Tesla V100 x 3,456	Tesla V100 x 2,048
Apr	Sept	Nov	July	Nov	Apr
2017				2018	2019

Fast.ai tips and engineering

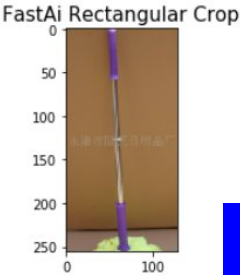


« Now anyone can train Imagenet in 18 minutes »

Our approach uses **128** processing units and costs around **\$40** to run.

 OneCycle lr scheduler
+ lr finder

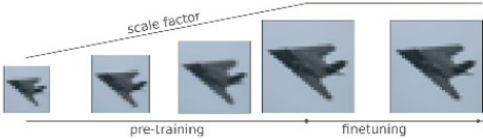
➡ Popularizes the works of [Leslie N. Smith](#)



Thanks to Global Average Pooling

**Test Rectangular
Validation Technique**

**Progressive image
resizing**



Dynamic batch size



Training Time



Goal:

Text Generation Foundation Model (LLM)

Model :

176 B parameters

Dataset:

366 B tokens

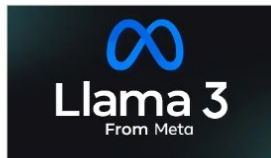


117 Days

384 A100 GPUs



Training Time



Goal:

Text Generation Foundation Model (LLM)

Model :

405 B parameters

Dataset:

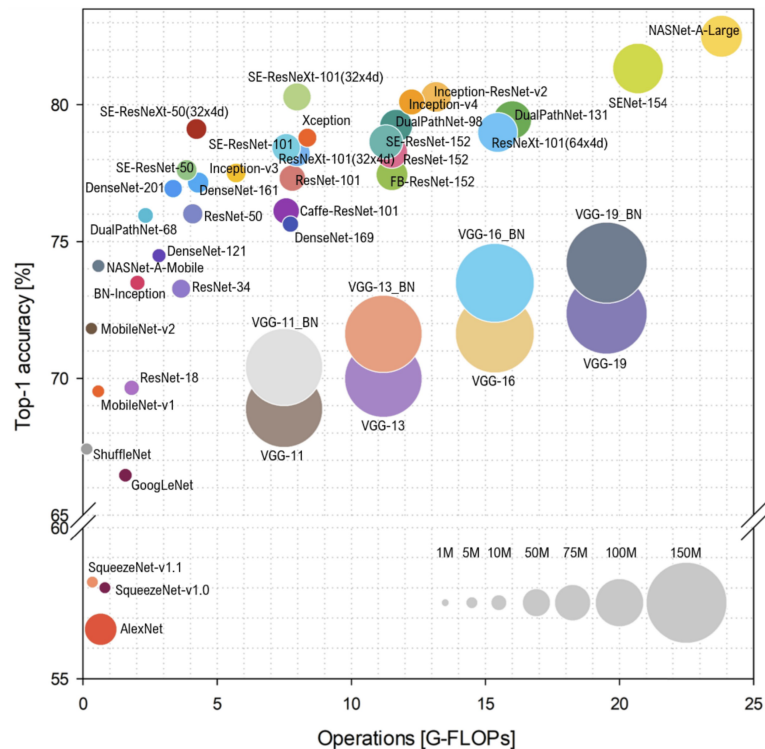
15 T tokens

54 Days

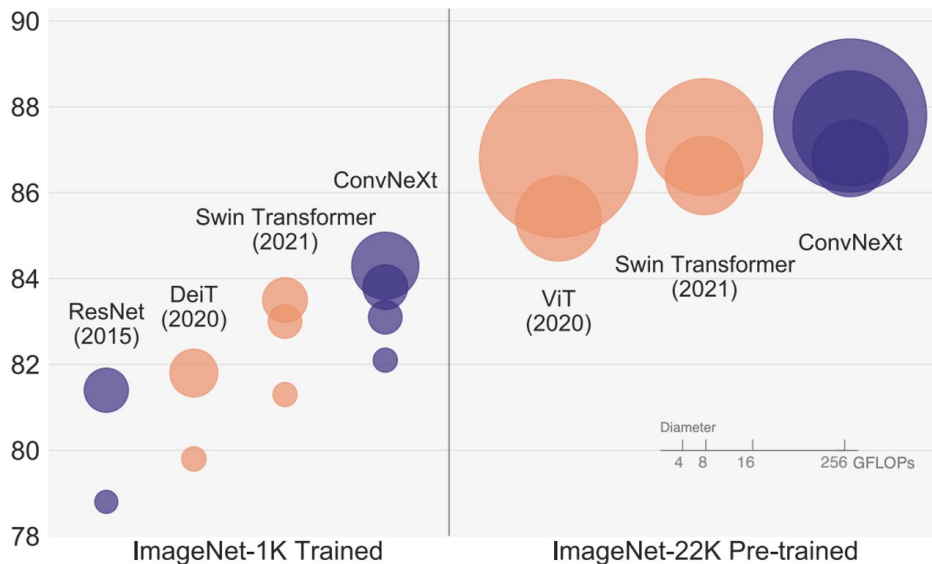
16 384 H100 GPUs

Large Model

Vision Neural Network



ImageNet-1K Acc.

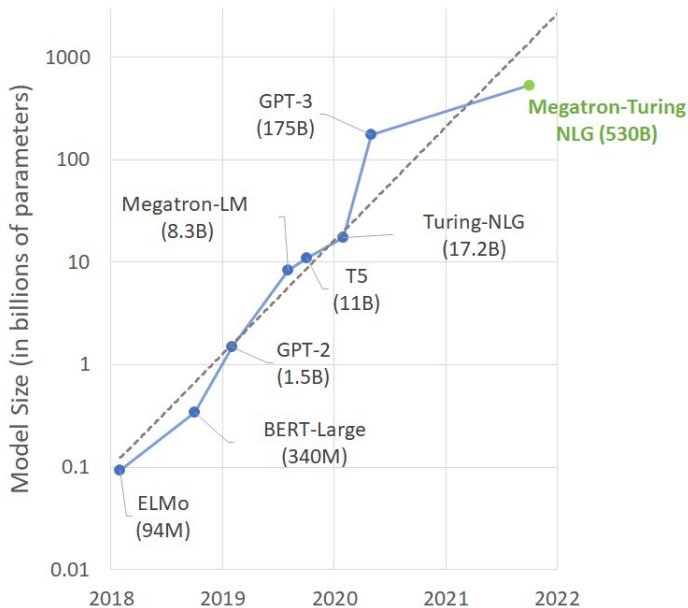


Large and deep models provide better accuracy metrics.

Large Model

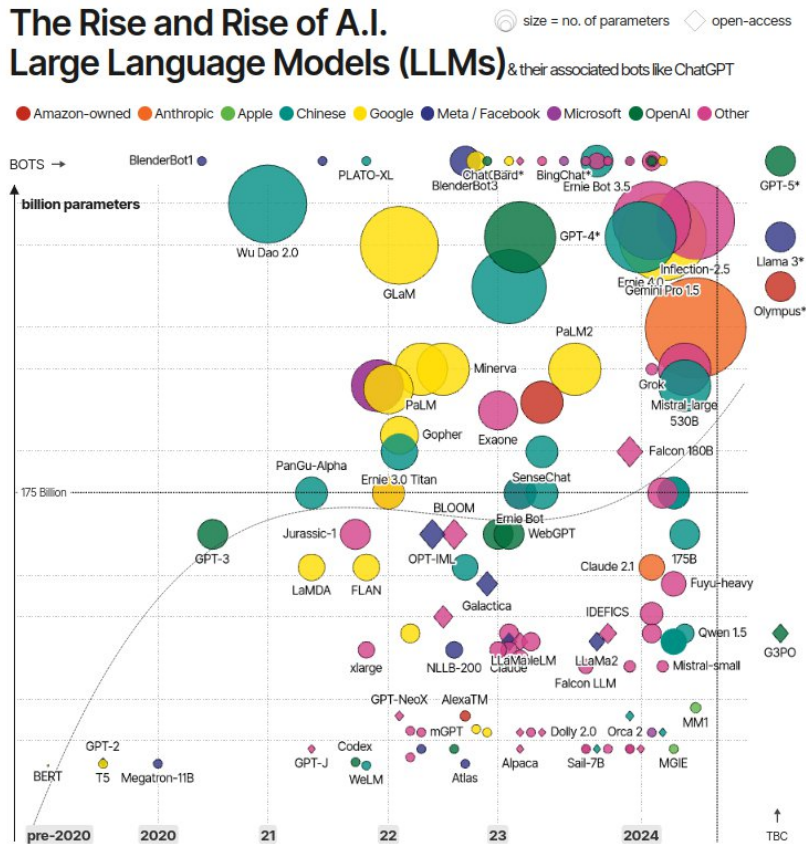
Huge models cause very expensive compute work times and large memory footprints (4 GB for a model with 1 billion parameters).

Transformers



The Rise and Rise of A.I.

Large Language Models (LLMs) & their associated bots like ChatGPT



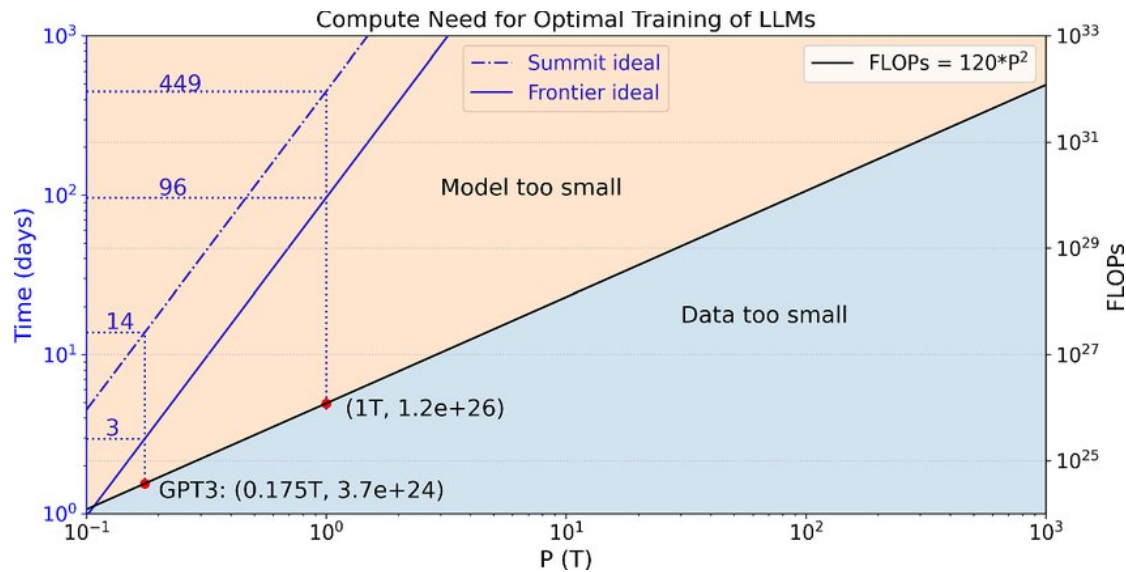
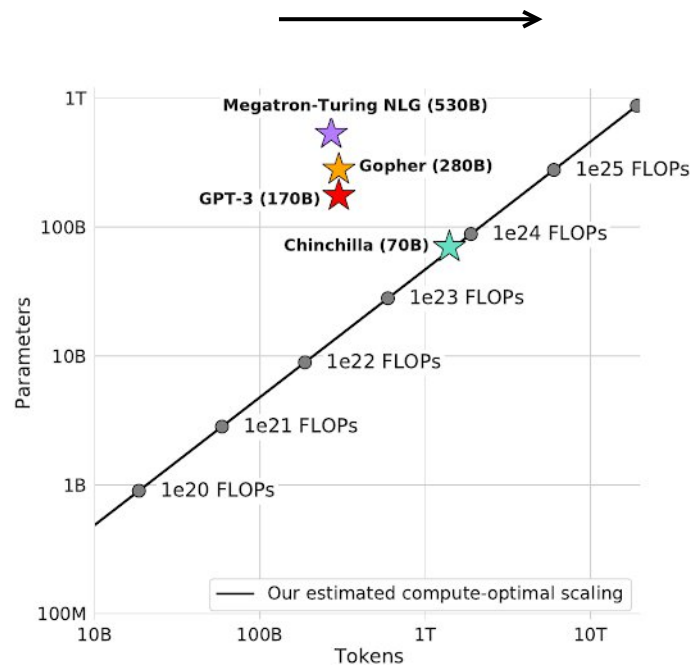
David McCandless, Tom Evans, Paul Barton
Information is Beautiful // UPDATED 20th Mar 24

* = parameters undisclosed // see the data

MADE WITH **VIZsweet**

Large Model

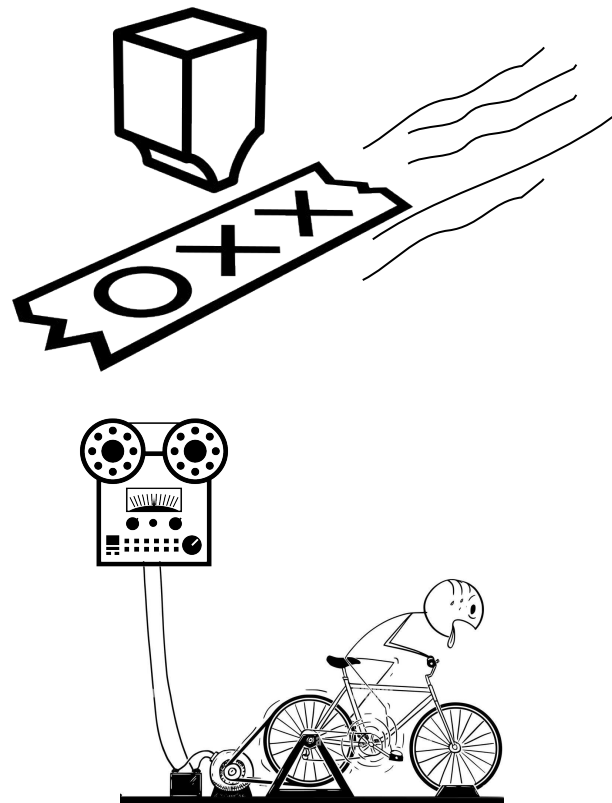
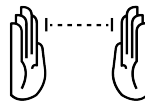
Chinchilla Law Impact



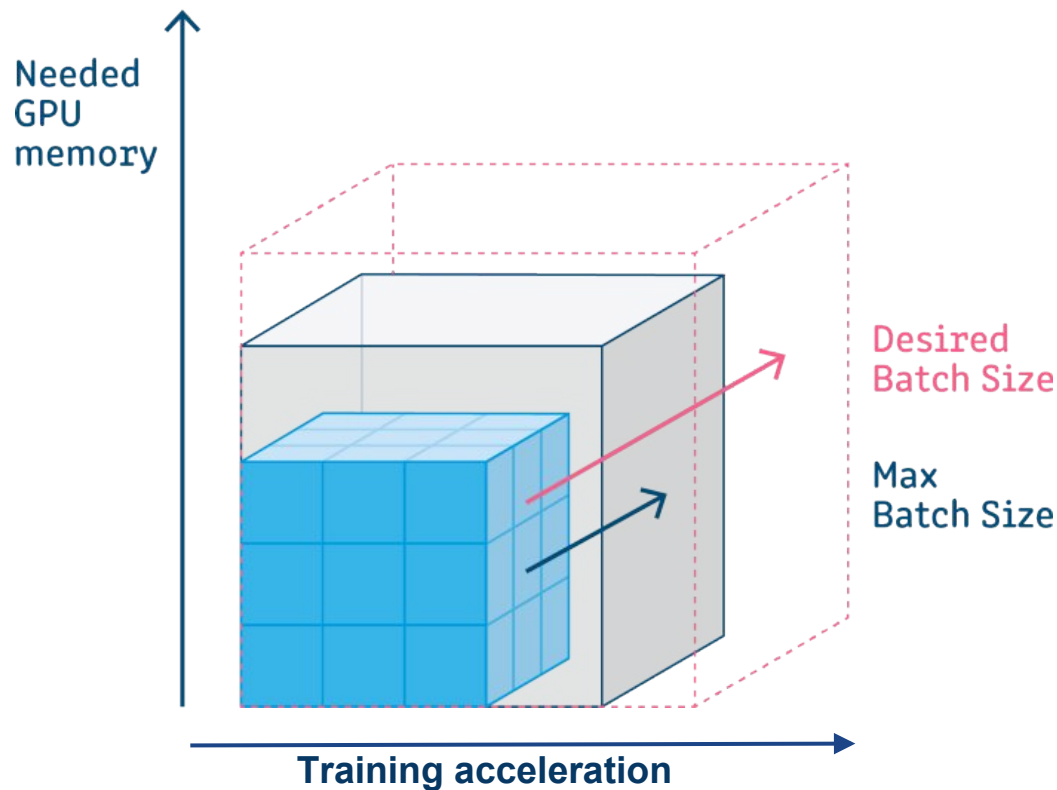
Compute Work Times

The compute time increases with the **number of FLOPs** required, depending on:

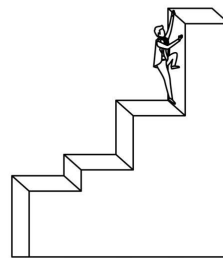
- The size of the model
- The depth of the model
- The size of the input data (image resolution, length of the sequence, etc.)
- The size of the dataset
- Number of epochs required



Batch Size & Memory Usage



Increasing the batch size and thus increasing the iteration step speeds up learning.



OOM
Process killed

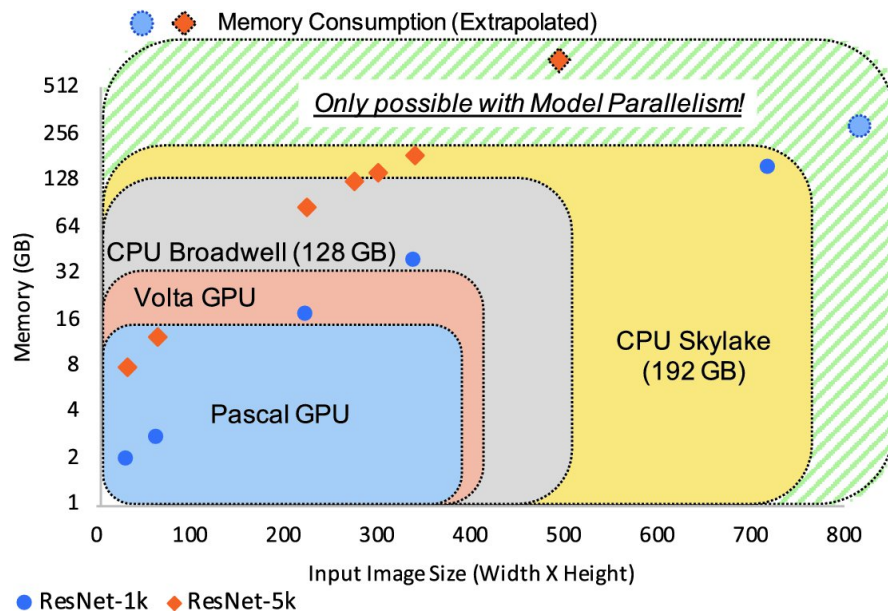
However, this increases the memory footprint and risks reaching the system limit.

Large size Input Data

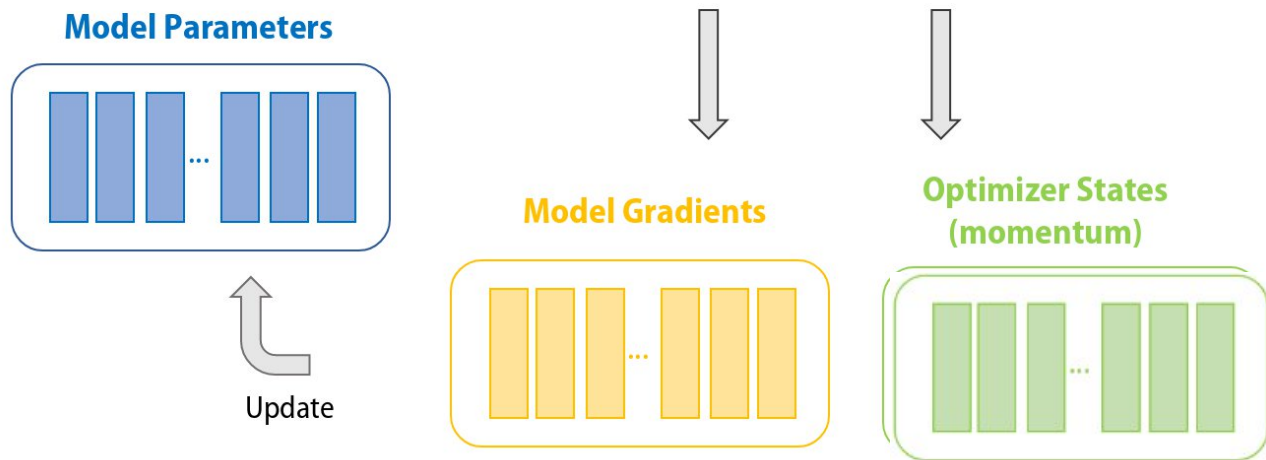
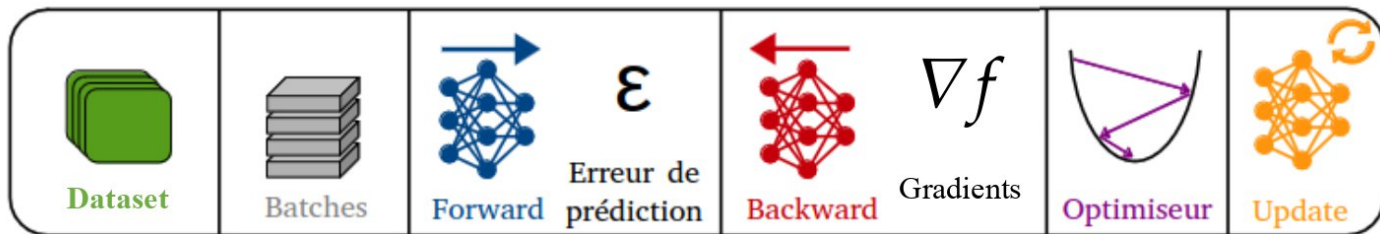
Large input data causes **serious memory occupancy problems** during training, **accentuated by the depth of the model**.

- Text (N, 100, 500) **~x1**
- Image 2D (N, 226, 226, 3) **~x3**
- Image 3D (N, 226, 226, 100, 3) **~x300**
- Video (N, 100, 226, 226, 3) **~x300**

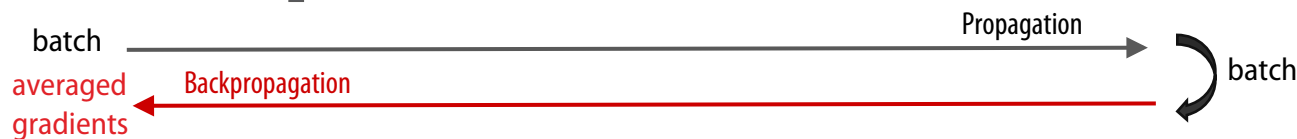
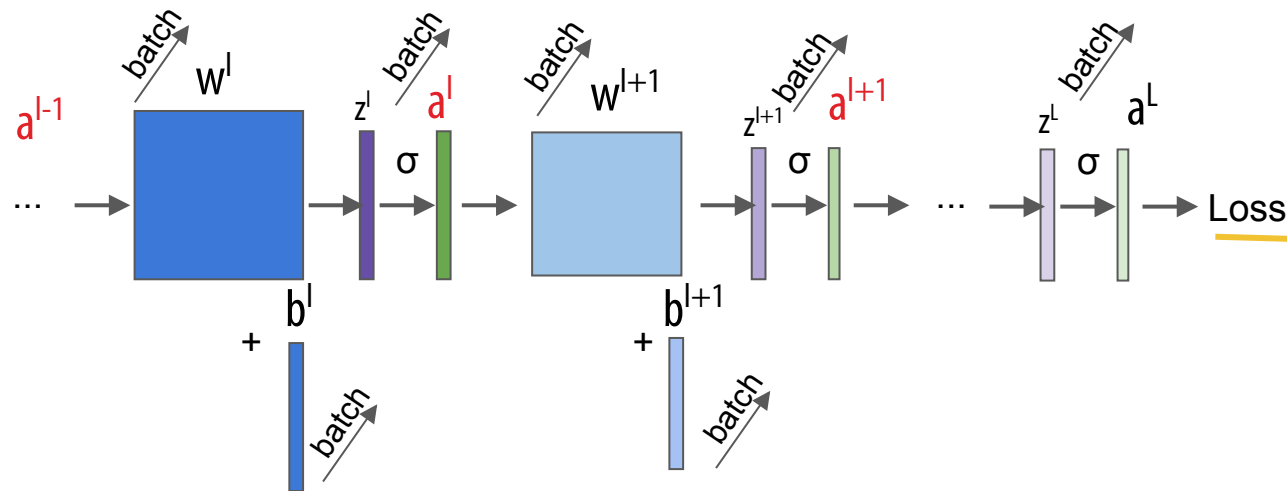
(GNN : Graph from tiny to huge !!)



Forward / Backward – Model Memory Occupancy



Forward / Backward – Activations Memory Occupancy



Propagation

$$a^l = \sigma(w^l a^{l-1} + b^l) = \sigma z^l$$

Backpropagation

$$\delta^l = \frac{\partial C}{\partial z^l} \quad \begin{aligned} w^l &\rightarrow w^l - \frac{\eta}{m} \cdot \frac{\partial C}{\partial w^l} \\ b^l &\rightarrow b^l - \frac{\eta}{m} \cdot \frac{\partial C}{\partial b^l} \end{aligned}$$

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

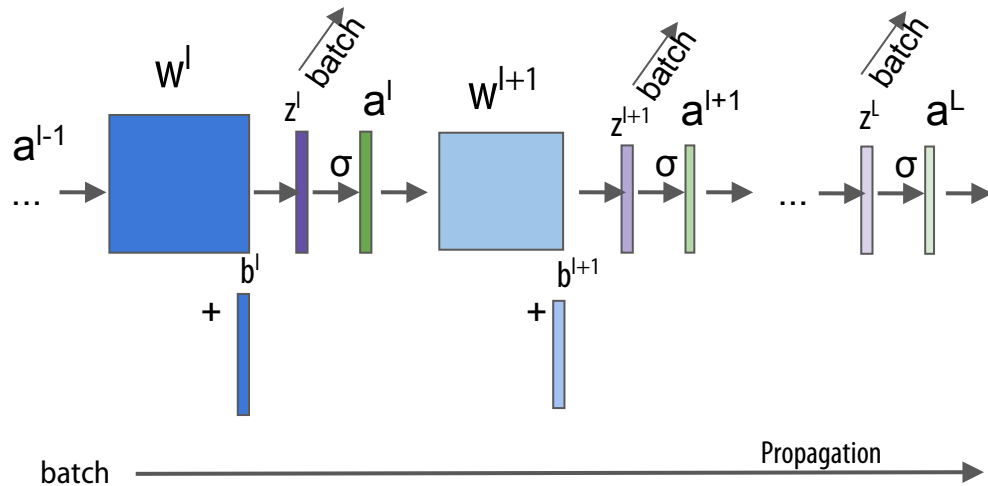
$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial w^l} = \delta^l (a^{l-1})^T$$

$$\frac{\partial C}{\partial b^l} = \delta^l$$

Note: For backpropagation, it is necessary to save **intermediate activations**.

Inference & evaluation



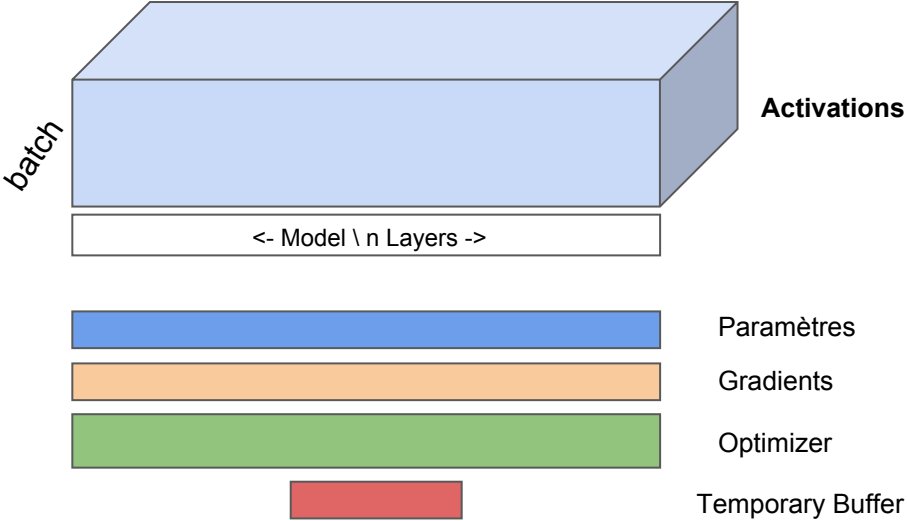
Propagation

$$a^l = \sigma(w^l a^{l-1} + b^l) = \sigma z^l$$

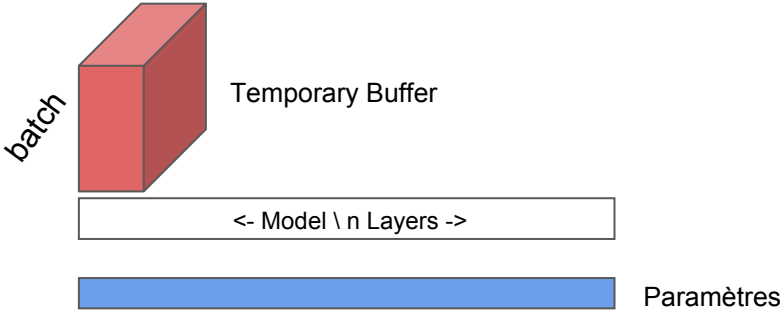
```
...  
with torch.no_grad():  
    val_outputs = model(val_images)  
    loss = criterion(val_outputs, val_labels)  
...
```

Memory Footprint

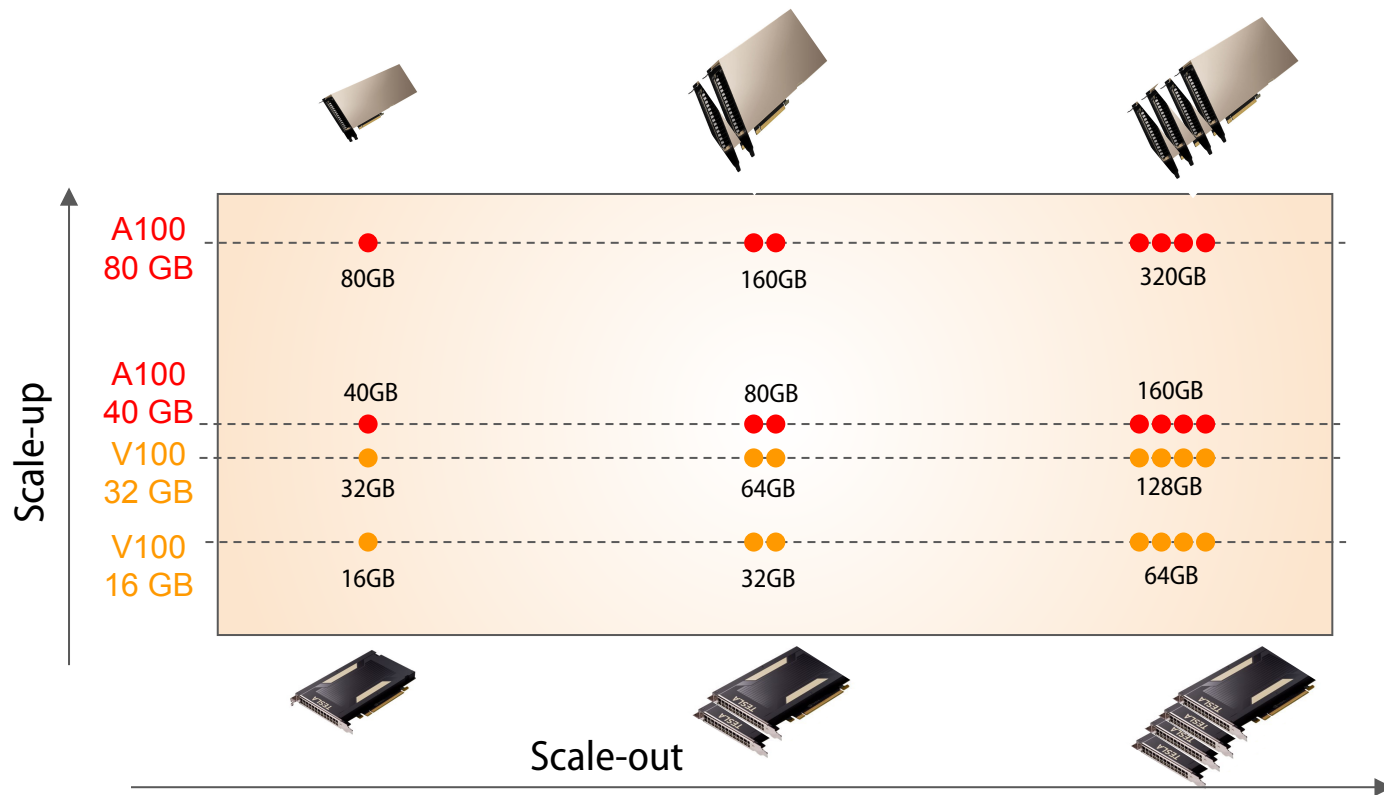
Training



Inference / Evaluation

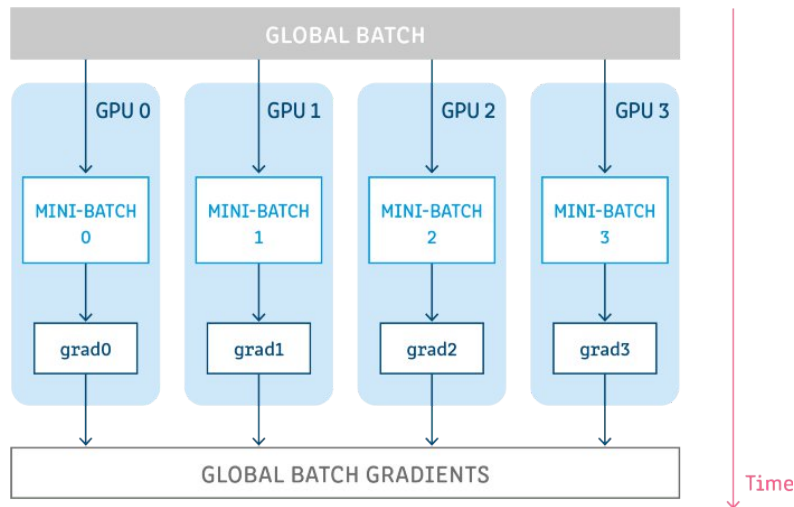


System Solutions

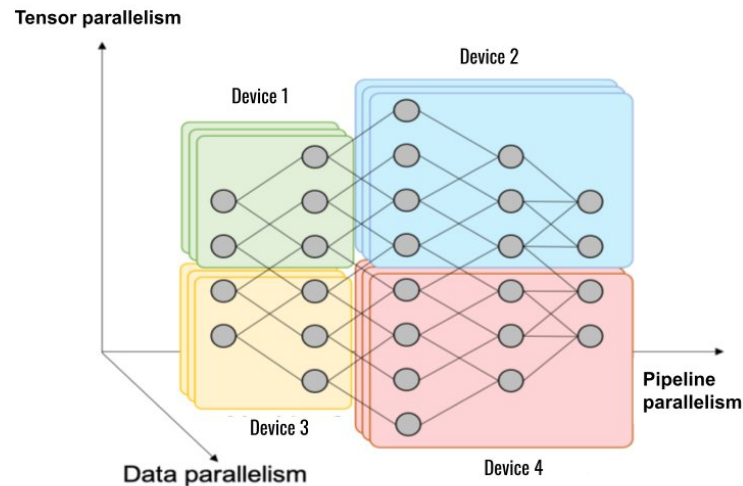


Solutions: Distribution – Scale-out

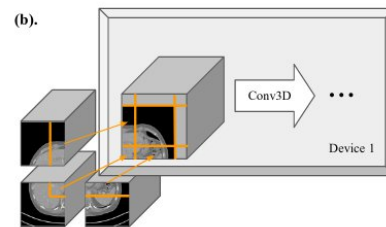
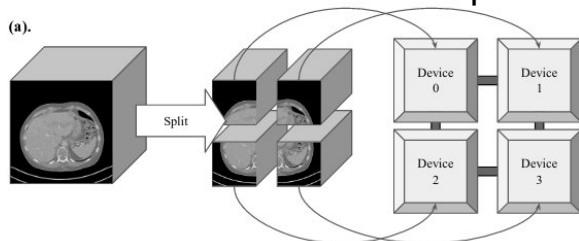
Data Parallelism



Model Parallelism

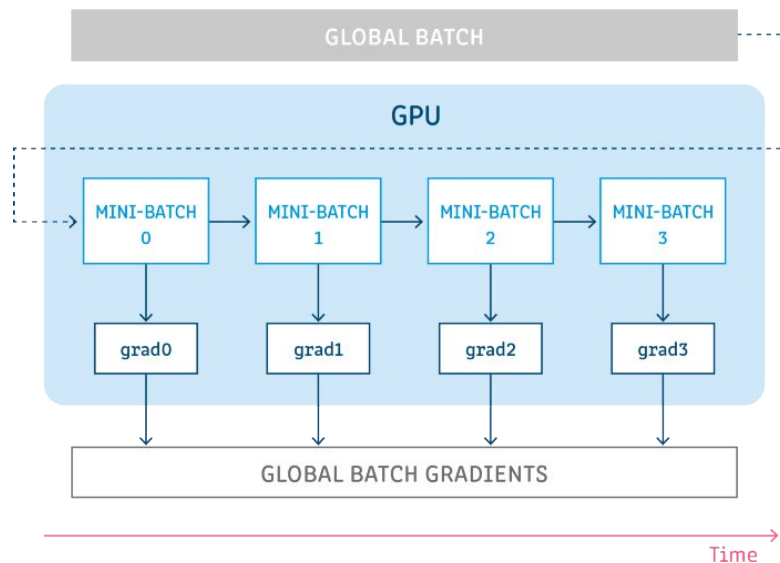


Spatial Partitioning



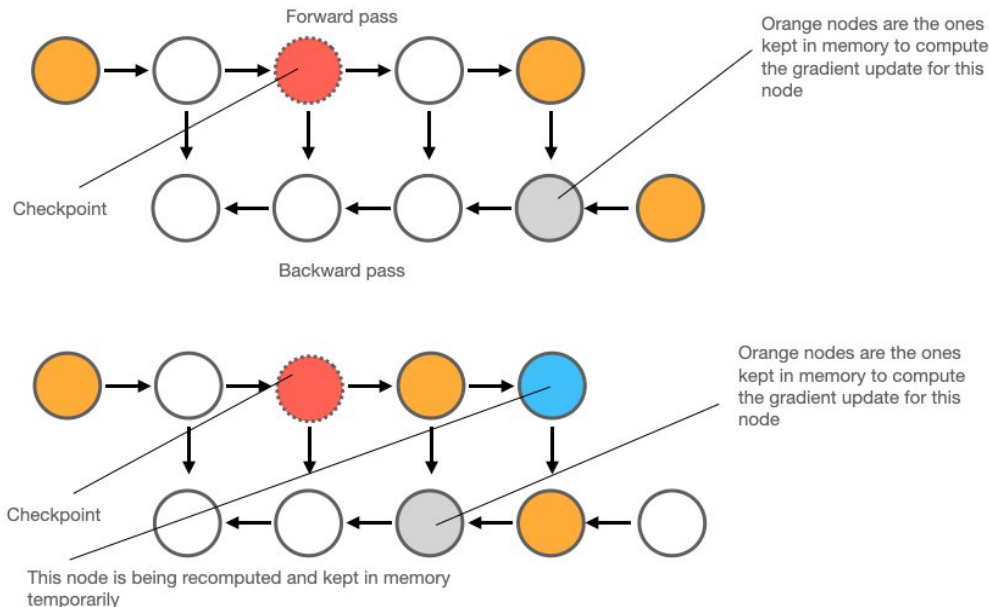
Workaround Solutions

Gradient aggregation



Gradient/activation checkpointing

Drawing inspired by <https://github.com/cybertronai/gradient-checkpointing>

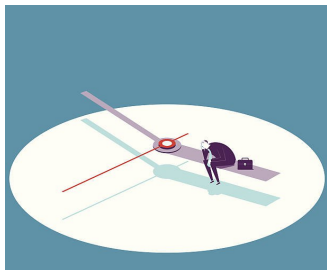


A 3rd problem to deal with ...

Power Consumption !!

2 problems to deal with:

Training Time



Memory Overconsumption (OOM)



Power Consumption

	H100 PCIe	H100 SXM5	A100 PCIe	A100 SXM4	V100 PCIe	V100 SXM2
Max Power	350W	700W	250W	400W	250W	300W
Idle Power	~30W	~55W	~30W	~60W	~40W	~45W
Performance	x3.5	x4	x1.75	x2	x0.9	x1

For a node: The CPU (often 2 processors) consumes what approximately 1 GPU consumes.



Power consumption varies depending on partial or overall GPU usage.

However, the power efficiency ratio is in favor of full use of the GPU.

Energy Consumption / GPU Hours

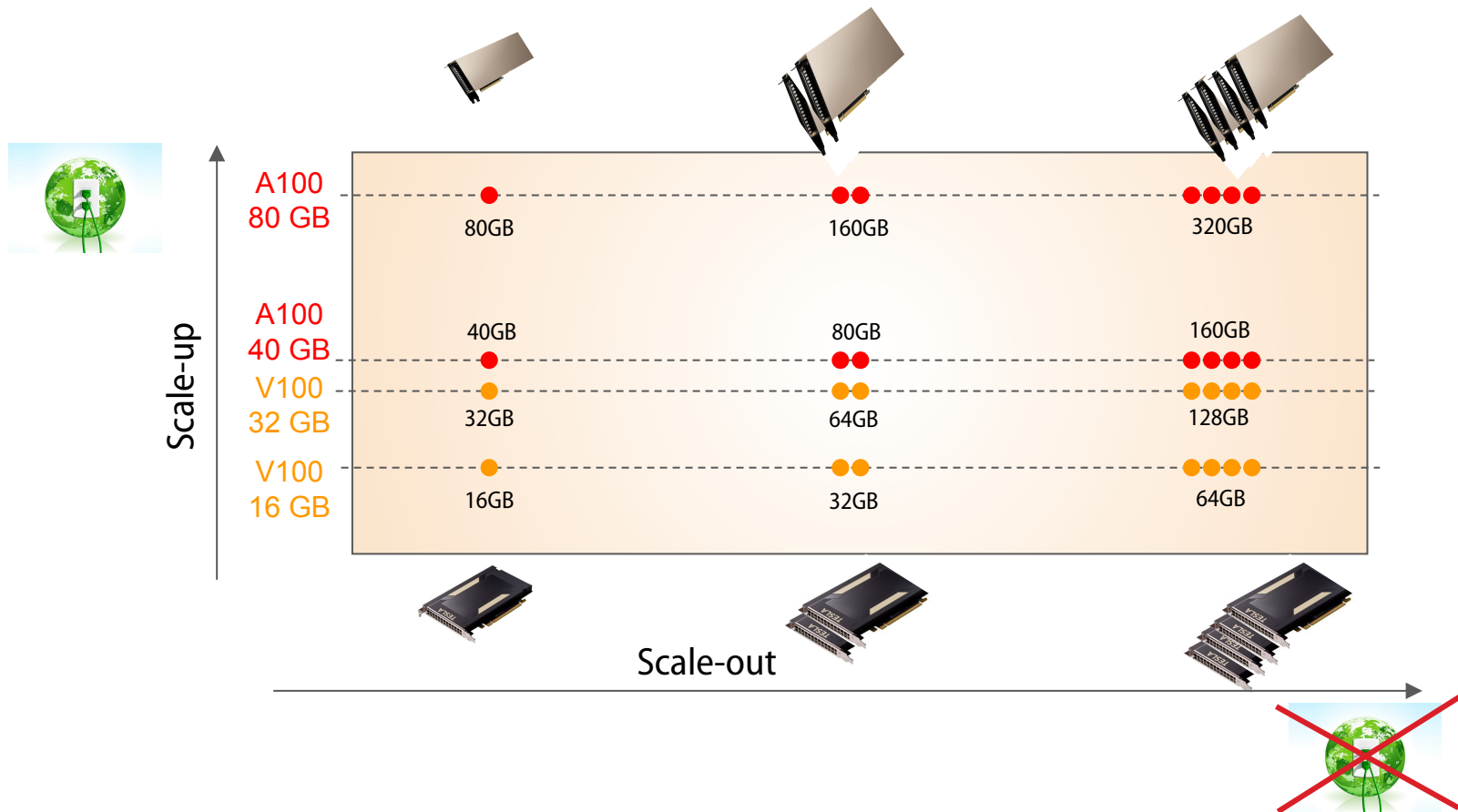
Energy Saving
 $\cong$
GPU Hours Saving



System optimization (DLO-JZ)

- Find the most important throughput
- Optimize data loading to eliminate GPU idle times
- Parallelize training to the right scale: neither too much nor too little

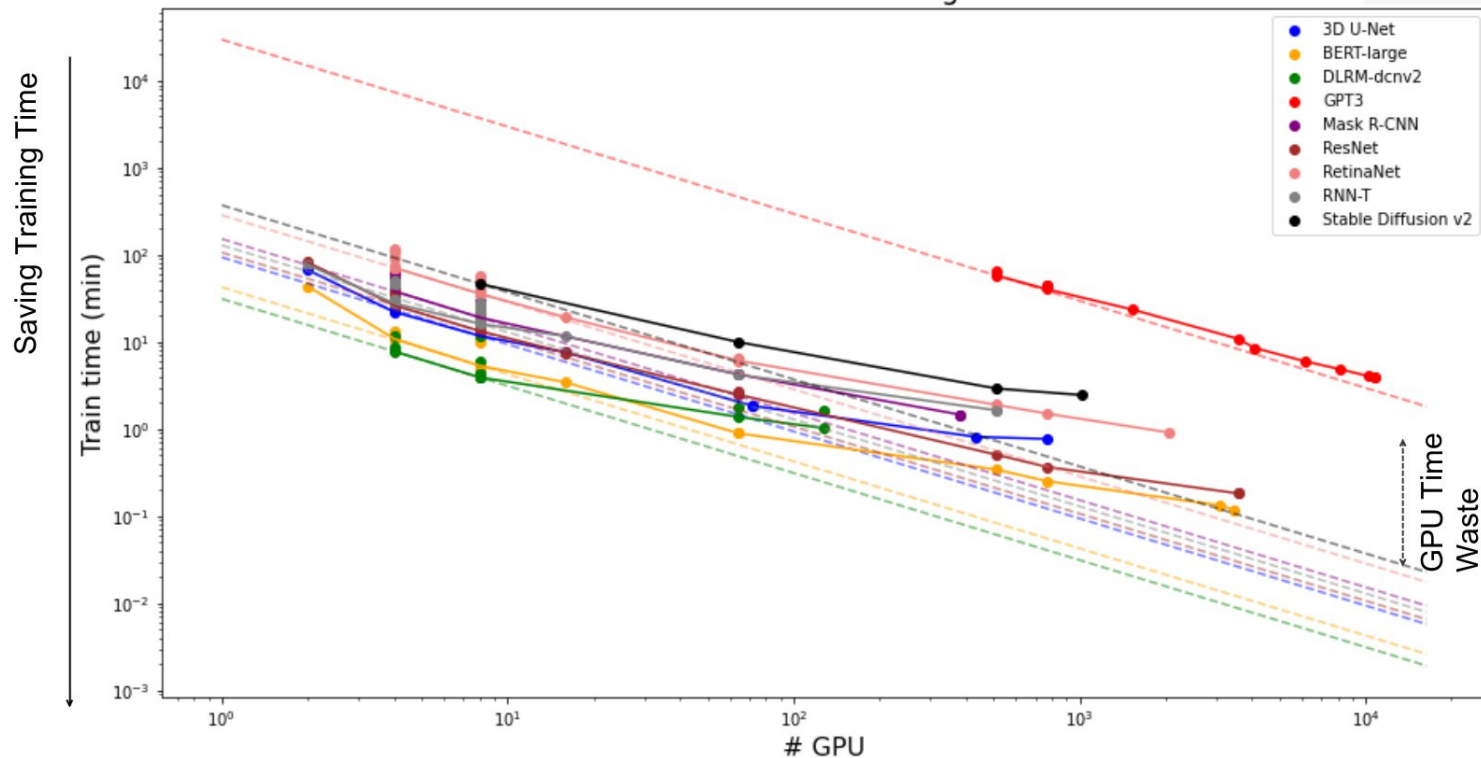
Energy Consumption / GPU Hours



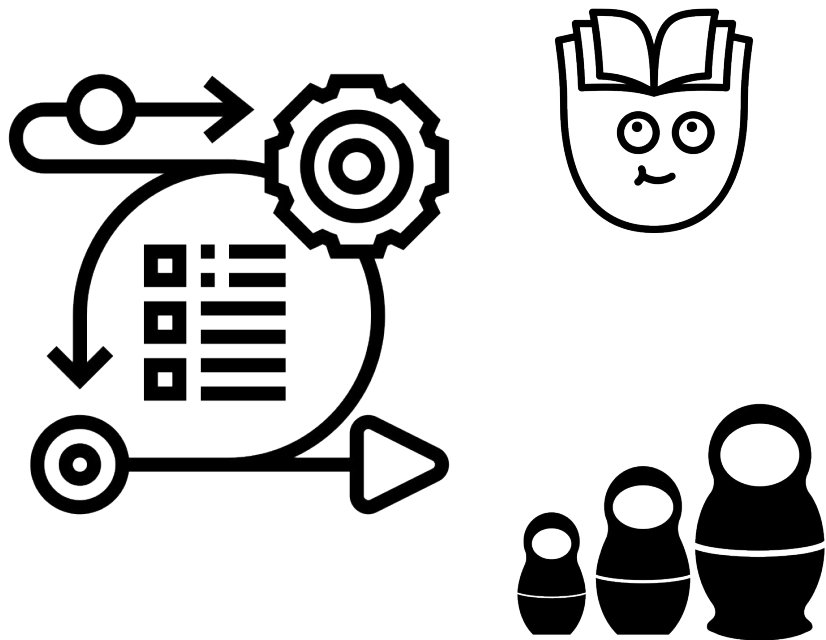
ML Perf Result - Scaling



MLPerf - H100 - Training v3.0



Energy Consumption / GPU Hours



Methodology (saving research, not repeating learning unnecessarily)

- Search for hyperparameters in publications and reproduce the state-of-the-art
- Find the right hyper parameters on smaller models, then apply them at scale
- *Hyper-Parameter Optimization* (HPO) techniques

Code review

General overview ◀

Detailed overview ◀

Data - Imagenet

Goal:

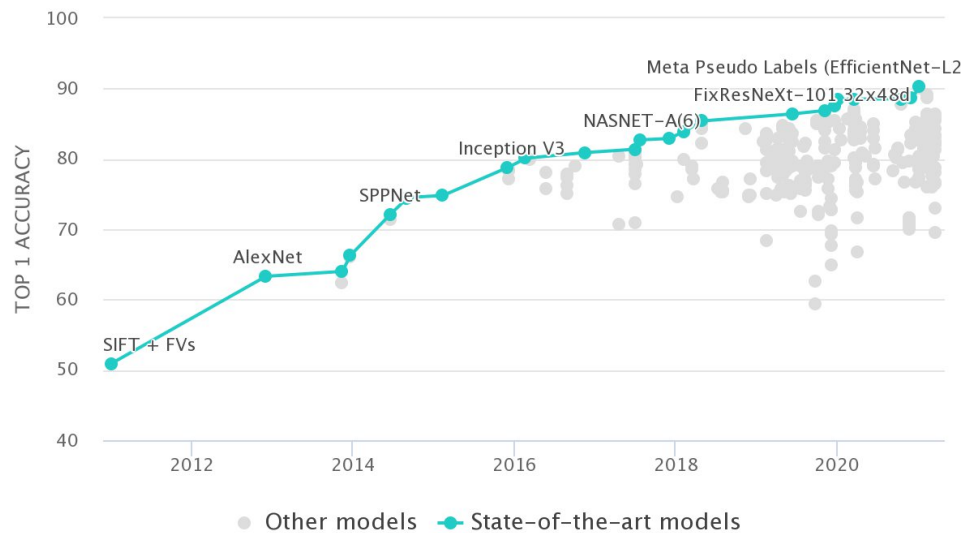
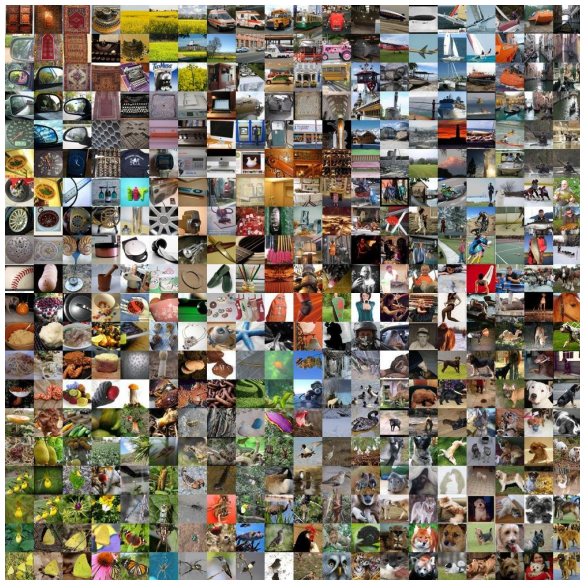
Classification (1000 classes)

Dataset:

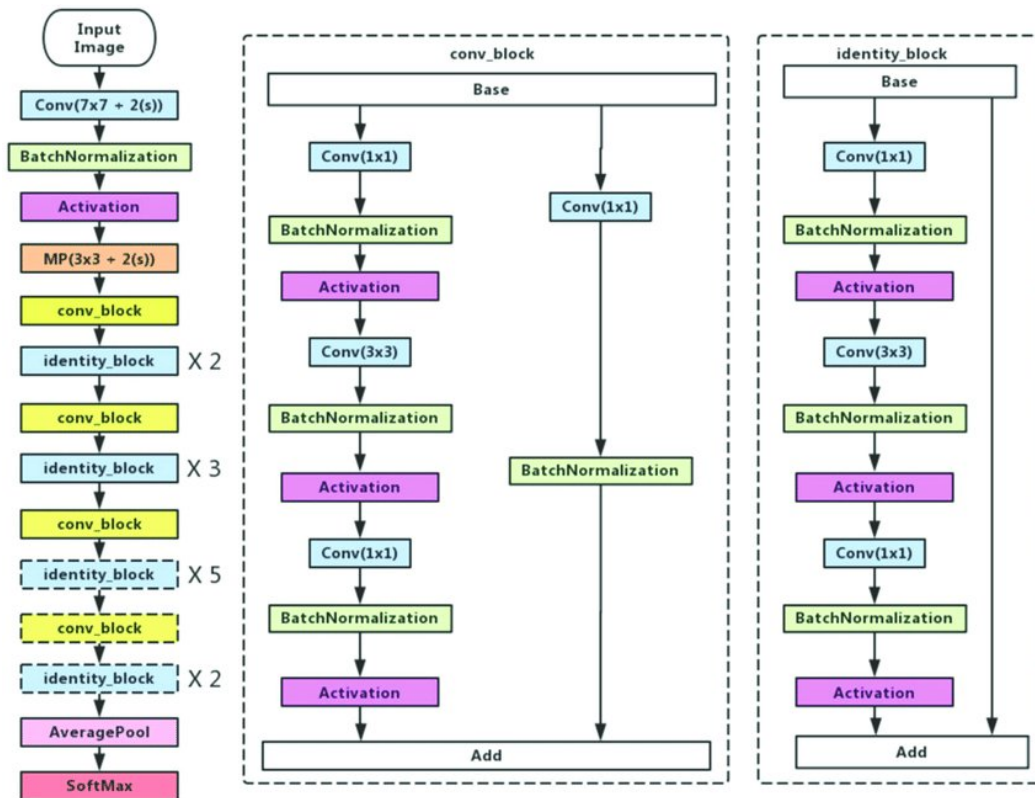
Train dataset: **1,2 M** labeled images

Validation dataset: **50 000** labeled images

<http://www.image-net.org/>



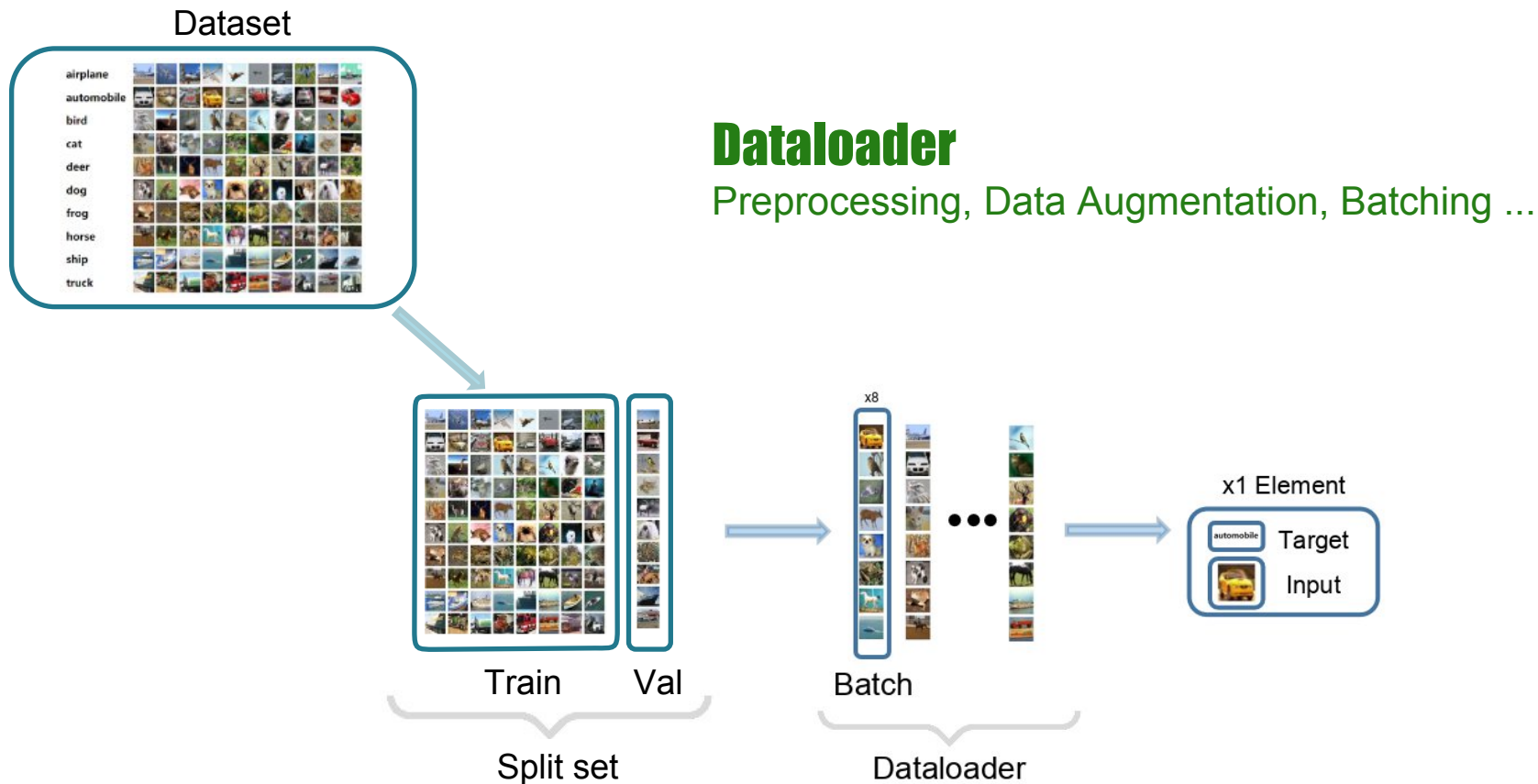
Imagenet - Resnet



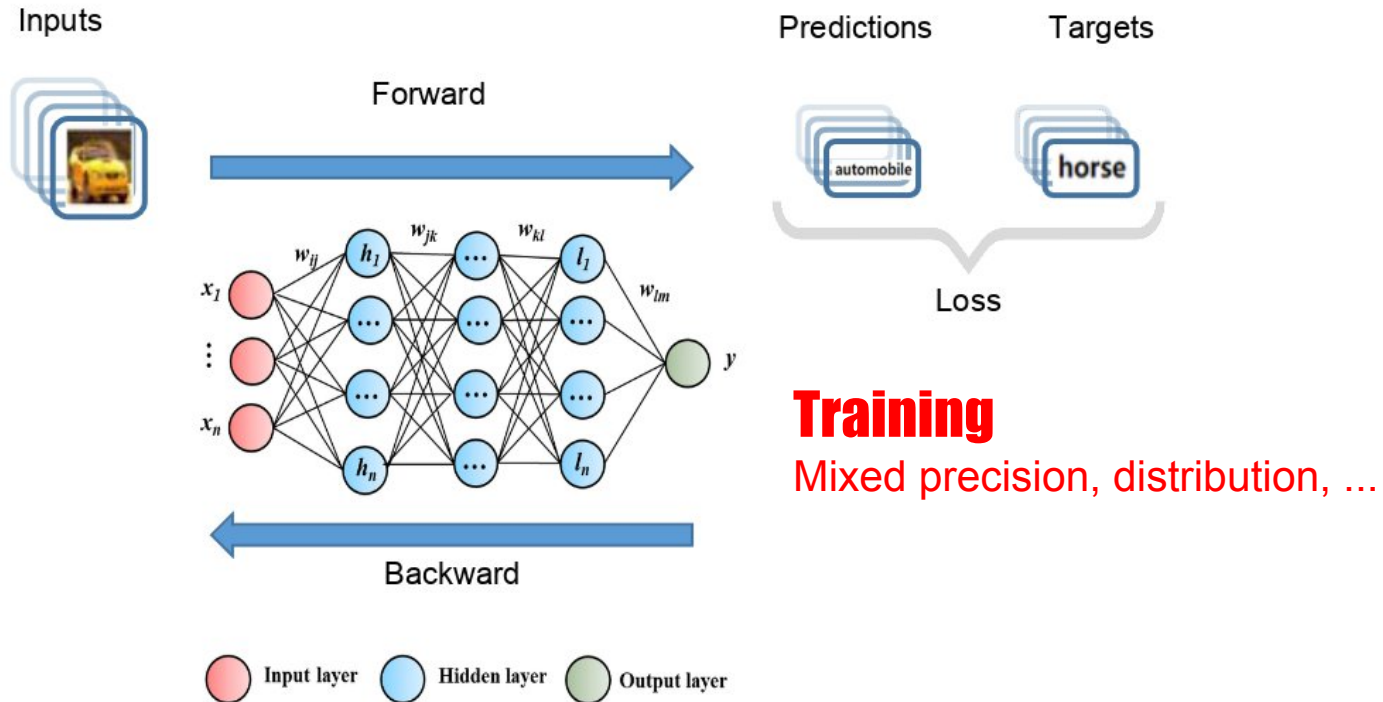
Resnet :

- Residual Learning
- BatchNorm layer
 - Instead of Bias layers with conv.
- Average Pooling
 - Makes the model independent of the size of the input images

Training Loop – DataLoader



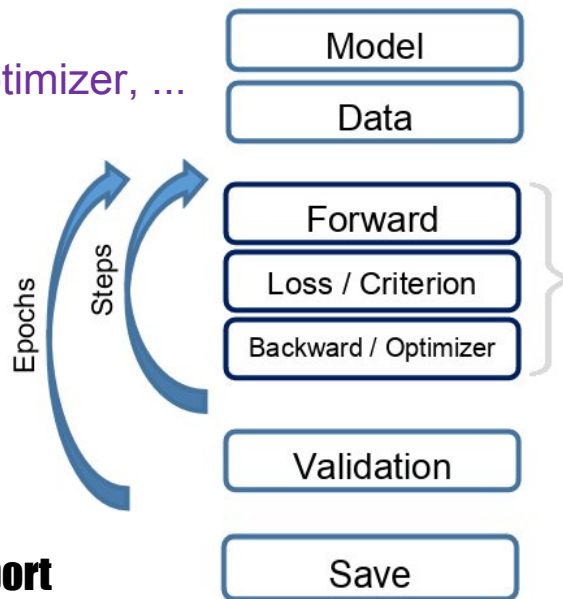
Training Loop – Forward/Backward



Training Loop

Instanciación

Model, distribution, optimizer, ...



Dataloader

Preprocessing, Data Augmentation, Batching ...

Training

Mixed precision, distribution, ...

Validation

Mixed precision, distribution, ...

Checkpoint & report

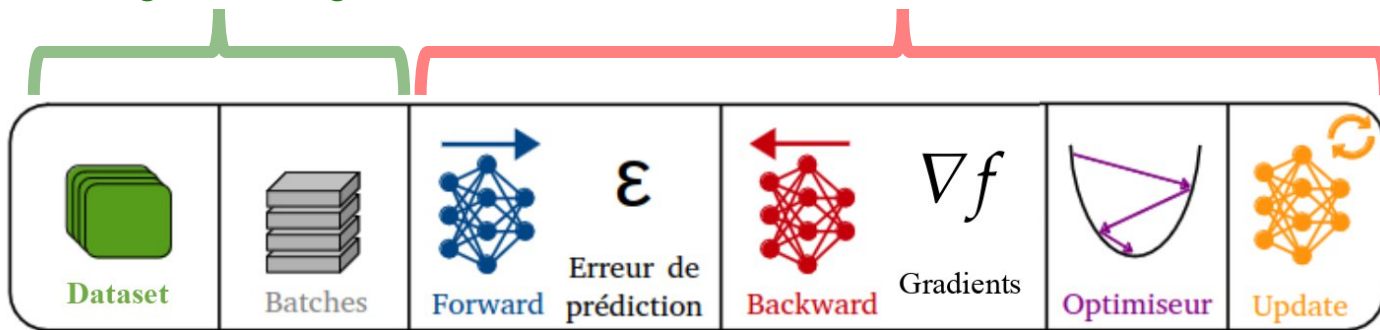
Training step

Dataloader

Preprocessing, Batching, ...

Training

Mixed precision, distribution, ...



on CPU

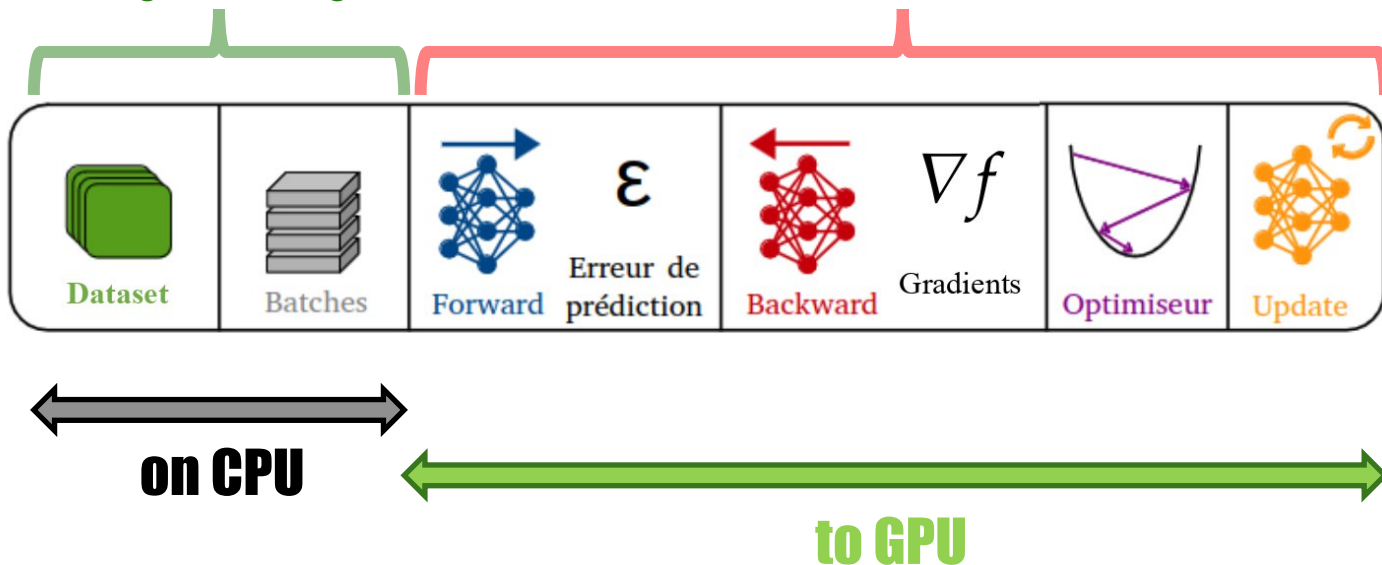
Training step – GPU Acceleration

Dataloader

Preprocessing, Batching, ...

Training

Mixed precision, distribution, ...



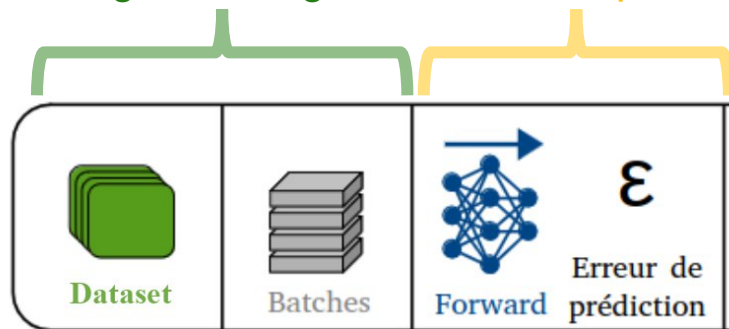
Validation step

Dataloader

Preprocessing, Batching, ...

Validation

Mixed precision, distribution, ...



on CPU

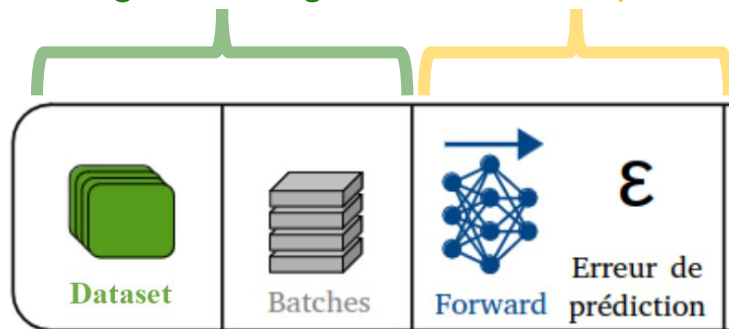
Validation step – GPU Acceleration

Dataloader

Preprocessing, Batching, ...

Validation

Mixed precision, distribution, ...



on CPU



to GPU

Code dlojz.py Review

Import

argparse : arguments

Instantiation

Model, distribution, optimizer, ...

Dataloader

Preprocessing, Batching, ...

Instantiation

Training

Mixed precision, distribution, ...

Validation

Mixed precision, distribution, ...

**Checkpoint & report
Runner**

dlojz.py – Import & run

```
import os
import contextlib
import argparse
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models
from torch.utils.checkpoint import checkpoint_sequential
import torch
import numpy as np
import apex

import idr_torch
from dloj_chrono import Chronometer
from dloj_torch import distributed_accuracy

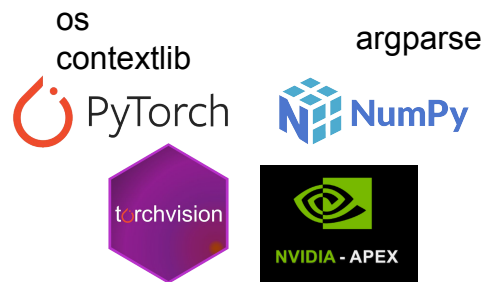
import random
random.seed(123)
np.random.seed(123)
torch.manual_seed(123)
```

reproducibility

idr_torch (JZ users)
distribution utils for Jean Zay

```
if __name__ == '__main__':
    # display info
    if idr_torch.rank == 0:
        print(">>> Training on ", len(idr_torch.hostnames), " nodes and ", idr_torch.size, " processes")
    train()
```

Import libraries



Chronometer (DLO-JZ)

time log & home profiler



distributed_accuracy (DLO-JZ)

home metric utils (torchmetric-like)



```
28 #*****
29 def train():
30     parser = argparse.ArgumentParser()
31     parser.add_argument('-b', '--batch-s
```

dlojz.py - arguments parser

```
## import ... ## Add here the libraries to import
```

```
VAL_BATCH_SIZE=256
```

```
def train():
```

```
    parser = argparse.ArgumentParser()
```

```
    parser.add_argument('-b', '--batch-size', default=128, type=int,  
                        help='batch size per GPU')
```

```
    parser.add_argument('-e', '--epochs', default=1, type=int,  
                        help='number of total epochs to run')
```

```
    parser.add_argument('--image-size', default=224, type=int,  
                        help='Image size')
```

```
    parser.add_argument('--lr', default=0.1, type=float,  
                        help='learning rate')
```

```
    parser.add_argument('--wd', default=0., type=float,  
                        help='weight decay')
```

```
    parser.add_argument('--mom', default=0.9, type=float,  
                        help='momentum')
```

```
    parser.add_argument('--test', default=False, action='store_true',      ## DON'T MODIFY #####  
                        help='Test 50 iterations')
```

```
    parser.add_argument('--test-nsteps', default='50', type=int,  
                        help='the number of steps in test mode')
```

```
    parser.add_argument('--num-workers', default=10, type=int,  
                        help='num workers in dataloader')
```

```
    parser.add_argument('--persistent-workers', default=True, action=argparse.BooleanOptionalAction,  
                        help='activate persistent workers in dataloader')
```

```
    parser.add_argument('--pin-memory', default=True, action=argparse.BooleanOptionalAction,  
                        help='activate pin memory option in dataloader')
```

```
    parser.add_argument('--non-blocking', default=True, action=argparse.BooleanOptionalAction,  
                        help='activate asynchronous GPU transfer')
```

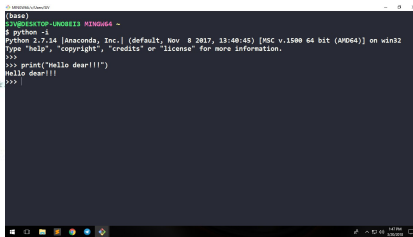
```
    parser.add_argument('--prefetch-factor', default=3, type=int,  
                        help='prefetch factor in dataloader')
```

```
    parser.add_argument('--drop-last', default=False, action=argparse.BooleanOptionalAction,  
                        help='activate drop last option in dataloader')
```

```
    # *****
```

```
    ## Add parser arguments
```

```
    args = parser.parse_args()
```



```
Python 3.10.10 [AMD64] on win32  
type 'help', 'copyright', 'credits' or 'license()' for more information.  
Hello dear!!!
```

Configurable Arguments :

- batch-size : batch size per GPU
- epochs : number of epochs
- image-size : image size

Optimizer :

- lr : learning rate
- wd : weight decay
- mom : momentum

Modes spéciaux :

- test : test mode
- test-nsteps : n steps for test mode

Optimisation du DataLoader :

- num-workers
- persistent-workers
- pin-memory
- non-blocking
- prefetch-factor
- drop-last

dlojz.py - instantiation

```
## chronometer initialisation
chrono = Chronometer()

# define model
model = models.resnet50()

archi_model = 'Resnet-50'

if idr_torch.rank == 0: print(f'model: {archi_model}')
if idr_torch.rank == 0: print('number of parameters: {}'.format(sum([p.numel()
                                                                    for p in model.parameters()])))

# distribute batch size (mini-batch)
num_replica = idr_torch.size
mini_batch_size = args.batch_size
global_batch_size = mini_batch_size * num_replica

if idr_torch.rank == 0:
    print(f'global batch size: {global_batch_size} - mini batch size: {mini_batch_size}')

# define loss function (criterion) and optimizer
criterion = torch.nn.CrossEntropyLoss(label_smoothing=0.1)
optimizer = torch.optim.SGD(model.parameters(), args.lr, momentum=args.mom, weight_decay=args.wd)

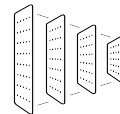
if idr_torch.rank == 0: print(f'Optimizer: {optimizer}')

# define metrics
train_metric = distributed_accuracy()
val_metric = distributed_accuracy()
```

```
##LR scheduler to accelerate the training time
scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr=args.lr,
                                                steps_per_epoch=N_batch, epochs=args.epochs)
```



Chronometer

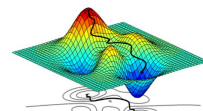


model : Resnet-152

mini batch size $\longleftrightarrow$ global batch size



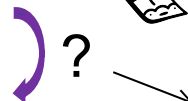
CrossEntropyLoss



SGD Optimizer



Metric



need N_{batch} , given by
dataloader



LR scheduler

dlojz.py - DataLoader

```
##### DATALOADER #####
# Define a transform to pre-process the training images.

if idr_torch.rank == 0: print(f"DATALOADER {args.num_workers} {args.persistent_workers} {args.pin_memory}")

transform = transforms.Compose([
    transforms.RandomResizedCrop(args.image_size), # Random resize - Data Augmentation
    transforms.RandomHorizontalFlip(),             # Horizontal Flip - Data Augmentation
    transforms.ToTensor(),                         # convert the PIL Image to a tensor
    transforms.Normalize(mean=(0.485, 0.456, 0.406),
                          std=(0.229, 0.224, 0.225))
])

train_dataset = torchvision.datasets.ImageNet(root=os.environ['ALL_CCFRSCRATCH']+'/imagenet',
                                              transform=transform)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=mini_batch_size,
                                           shuffle=True,
                                           num_workers=args.num_workers,
                                           persistent_workers=args.persistent_workers,
                                           pin_memory=args.pin_memory,
                                           prefetch_factor=args.prefetch_factor,
                                           drop_last=args.drop_last)

val_transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.CenterCrop(224),
    transforms.ToTensor(), # convert the PIL Image to a tensor
    transforms.Normalize(mean=(0.485, 0.456, 0.406),
                          std=(0.229, 0.224, 0.225))]
])

val_dataset = torchvision.datasets.ImageNet(root=os.environ['ALL_CCFRSCRATCH']+'/imagenet', split='val',
                                           transform=val_transform)

val_loader = torch.utils.data.DataLoader(dataset=val_dataset,
                                         batch_size=VAL_BATCH_SIZE,
                                         shuffle=False,
                                         num_workers=args.num_workers,
                                         persistent_workers=args.persistent_workers,
                                         pin_memory=args.pin_memory,
                                         prefetch_factor=args.prefetch_factor,
                                         drop_last=args.drop_last)

N_batch = len(train_loader)
N_val_batch = len(val_loader)
N_val = len(val_dataset)
```

train dataset :

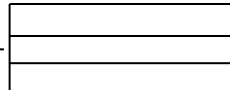
RandomResizedCrop
RandomHorizontalFlip
+ Normalize



Shuffling

DataLoader
optimization

spawn



batch

validation dataset :

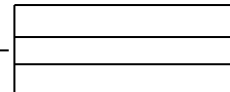
Resize
CenterCrop
+ Normalize



no shuffling

DataLoader
optimization

spawn



batch

dlojz.py - Training

```

chrono.start()

#### TRAINING #####
for epoch in range(args.epochs):

    if args.test: chrono.next_iter()
    if idr_torch.rank == 0: chrono.tac_time(clear=True)

    for i, (images, labels) in enumerate(train_loader):

        csteps = i + 1 + epoch * N_batch
        if args.test and csteps > args.test_nsteps: break
        if i == 0 and idr_torch.rank == 0:
            print(f'image batch shape : {images.size()}')

        if args.test: chrono.forward()

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)

        if args.test: chrono.backward()

        loss.backward()
        optimizer.step()

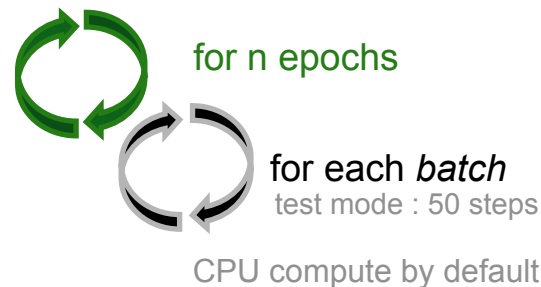
    # Metric mesurement
    train_metric.update(loss, outputs, labels)

    if args.test: chrono.update()

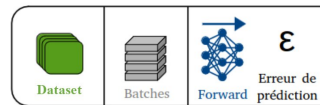
    if ((i + 1) % (N_batch//10) == 0 or i == N_batch - 1) and idr_torch.rank == 0:
        train_loss, accuracy = train_metric.compute()
        print('Epoch [{}/{}], Step [{}/{}], Time: {:.3f}, Loss: {:.4f}, Acc:{:.4f}'.format(
            epoch + 1, args.epochs, i+1, N_batch,
            chrono.tac_time(), loss_acc, accuracy, accuracy_top5))

    # scheduler update
    scheduler.step()

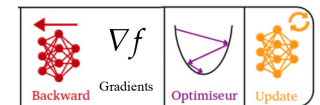
```



optimizer.zero_grad()
outputs = model(images)
loss = criterion(outputs, labels)



loss.backward()
optimizer.step()



Aggregate the metrics (loss, accuracy)
10x per epoch, compute and print the metrics

Log 10x per epoch



Step up LR scheduler

dlojz.py - Validation

```
#### VALIDATION #####
if ((i == N_batch - 1) or (args.test and i==args.test_nsteps-1)) :

    chrono.validation()
    model.eval()

    for iv, (val_images, val_labels) in enumerate(val_loader):

        # Runs the forward pass with no grad mode.
        with torch.no_grad():
            val_outputs = model(val_images)
            val_loss = criterion(val_outputs, val_labels)

        val_metric.update(val_loss, val_outputs, val_labels)

    if args.test and iv >= 20: break

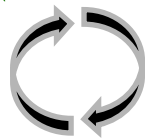
    val_loss, val_accuracy = val_metric.compute()

    model.train()
    chrono.validation()
    if not args.test and idr_torch.rank == 0:
        print('##EVALUATION STEP##')
        print('Epoch [{}/{}], Validation Loss: {:.4f}, Validation Accuracy: {:.4f}'.format(
            epoch + 1, args.epochs, val_loss, val_accuracy))
        print(">>> Validation complete in: " + str(chrono.val_time))

#### END OF VALIDATION #####
```

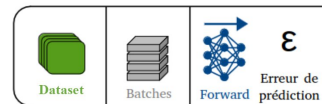


after each epoch
(or at the end of test mode)



for each *batch* of validation
(test mode : 20 steps)

```
# Runs the forward pass with no grad mode.
with torch.no_grad():
    val_outputs = model(val_images)
    loss = criterion(val_outputs, val_labels)
```



Aggregate the metrics (loss, accuracy)

when it is over:



compute & **Log**

[illegible]

[illegible][illegible][illegible]

dlojz.py – Distributed_accuracy



```
class distributed_accuracy():
    def __init__(self):
        self.dist = dist.is_initialized()
        self.correct = torch.tensor(0)
        self.total = torch.tensor(0)
        self.loss = torch.tensor(0, dtype=torch.float)

    def update(self, losses, outputs, labels):
        _, predicted = torch.max(outputs.data, 1)
        ## for mixed data augmentation
        if len(labels.size()) > 1: labels = torch.argmax(labels, dim=1)
        self.correct += (predicted == labels).sum().item()
        self.total += labels.size(0)
        self.loss += losses.sum().item()

    def clear(self):
        self.correct = torch.tensor(0)
        self.total = torch.tensor(0)
        self.loss = torch.tensor(0, dtype=torch.float)

    def compute(self):
        if self.dist and idr_torch.size > 1:
            self.correct = self.correct.to('cuda')
            self.total = self.total.to('cuda')
            self.loss = self.loss.to('cuda')
            dist.all_reduce(self.correct, op=dist.ReduceOp.SUM)
            dist.all_reduce(self.total, op=dist.ReduceOp.SUM)
            dist.all_reduce(self.loss, op=dist.ReduceOp.SUM)
        accuracy = (self.correct / self.total).item()
        loss = (self.loss / self.total).item()
        self.clear()
        return loss, accuracy
```

dlojz.py – Distributed_accuracy



```
class distributed_accuracy():
    def __init__(self):
        self.dist = dist.is_initialized()
        self.correct = torch.tensor(0)
        self.total = torch.tensor(0)
```

Equivalent to Torchmetric !!!

<https://lightning.ai/docs/torchmetrics/stable/pages/overview.html>

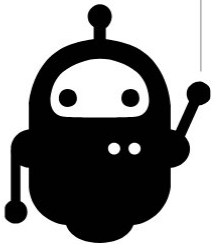
```
from torchmetrics.classification import MulticlassAccuracy
```

The **metrics API** provides `update()`, `compute()`, `reset()` functions to the user.

These metrics **work with DDP** in PyTorch and PyTorch Lightning by default. When `.compute()` is called in distributed mode, the internal state of each metric is synced and reduced across each process, so that the logic present in `.compute()` is applied to state information from all processes.

```
accuracy = (self.correct / self.total).item()
loss = (self.loss / self.total).item()
self.clear()
return loss, accuracy
```

TP0 : Préparation de l'environnement



- Lancer un terminal et faire les copies nécessaires

```
local:~$ ssh jean-zay
```

```
jz:~$ cd $WORK
```

```
jz:~$ git clone https://github.com/IDRIS-CNRS/DLO-JZ.git
```

- Lancer firefox
- Accéder à jupyterhub.idris.fr

TP0 : Accès et prise en main de JupyterHub

- Se connecter avec vos identifiants de formation

- Lancer une instance

List of JupyterLab instances

Every user may have 10 JupyterLab server(s) with names. This allows the t

DLO_TP	Add New JupyterLab Instance	
Instance name	URL	Node type

- Sélectionner le spawner 'Interactive'

Interactive	SLURM
-------------	-------

- Remplir la configuration

- Start

JupyterLab instance will be launched on a Jean Zay frontal node. Globally, the resources are limited to one CPU and 5 GB of memory for each user.

Time (--time) (in hours)

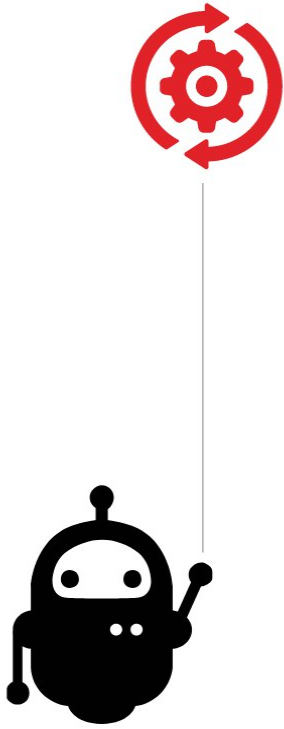
Notebook directory (--ServerApp.notebook_dir)

Root directory of the JupyterLab file explorer is also set to this path

Environment variables (one per line)

Custom environment variables can be defined here. Subshells are not supported

TP0 : Accès et prise en main du notebook



- Ouvrir le notebook DLO-JZ_Jour1.ipynb
- Choisir le kernel pytorch-gpu/py3/2.1.1 (en haut à droite) s'il n'est pas détecté automatiquement
- Choisir un pseudonyme
- Lancer un job
- Prendre en main le script de référence et les différentes fonctionnalités

GPU computing

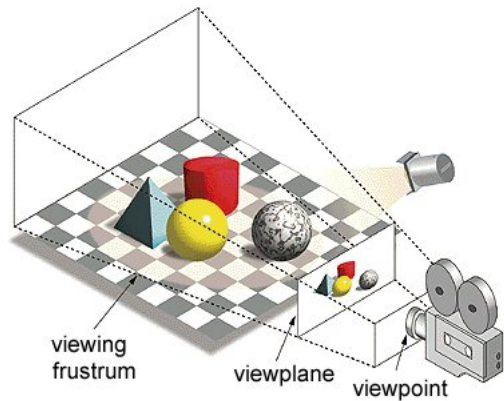
V100, A100, H100 ◀

CUDA ◀

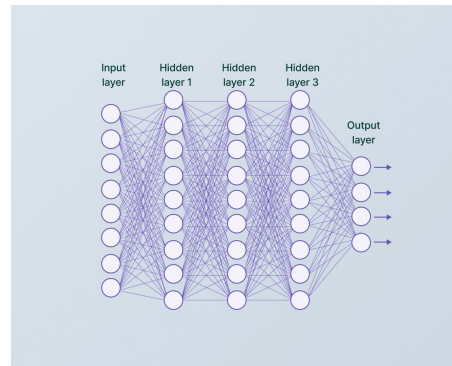
CuDNN ◀

AMP ◀

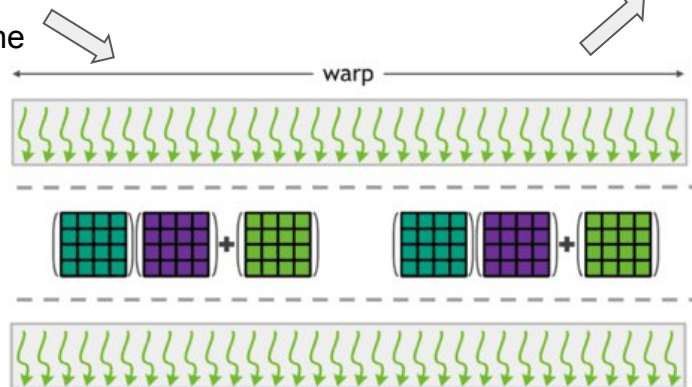
GPU computing



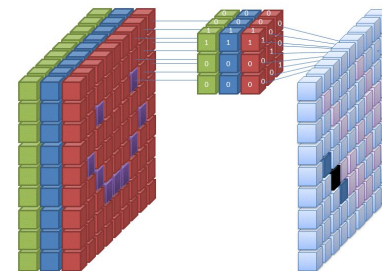
GPU Rendering & Game Graphics Pipeline



NN



Matrix Multiply-accumulate operations



CNN

NVIDIA Galaxy



RAPIDS

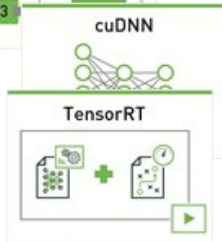
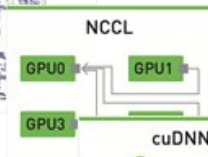
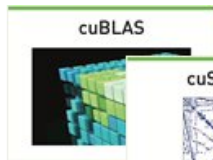
Open GPU Data Science



Fortran



OpenACC
Directives For Accelerators



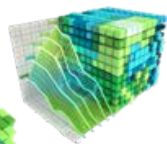
**CUDA
MEMCHECK**



Nsight IDE



**CUDA-GDB
Debugger**



**NVIDIA
Visual Profiler**

TRAINING AND INFERENCE

INFERENCE AT THE EDGE

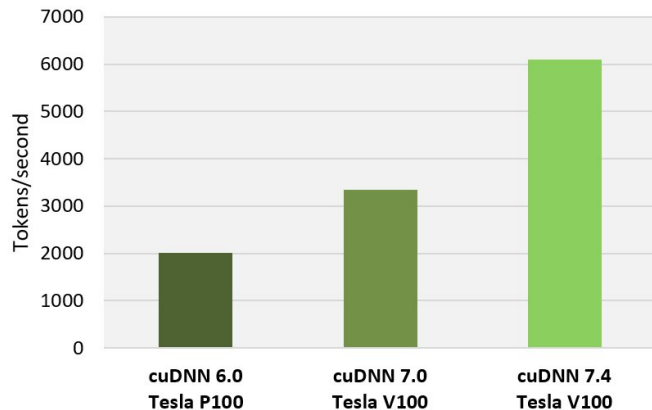
DESKTOP	DATACENTER AND CLOUD
 DGX Station  Titan V	 DGX-2  DGX-1  Tesla V100
AUTONOMOUS MACHINES	AI SELF-DRIVING PLATFORM
 Jetson TX2  Jetson TX1	 DRIVE Pegasus
NVIDIA DEEP LEARNING SDK and CUDA	

Source : [NVidia](https://www.nvidia.com)

CuDNN

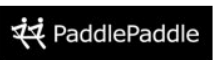


Up to 3x Faster RNN Training



TensorFlow performance (tokens/sec), Tesla P100 + cuDNN 6 (FP32) on 17.12 NGC container, Tesla V100 + cuDNN 7.0 (Mixed) on 18.02 NGC container, Tesla V100 + cuDNN 7.4 (Mixed) on 18.10 NGC container, OpenSeq2Seq (GNMT), Batch Size: 64





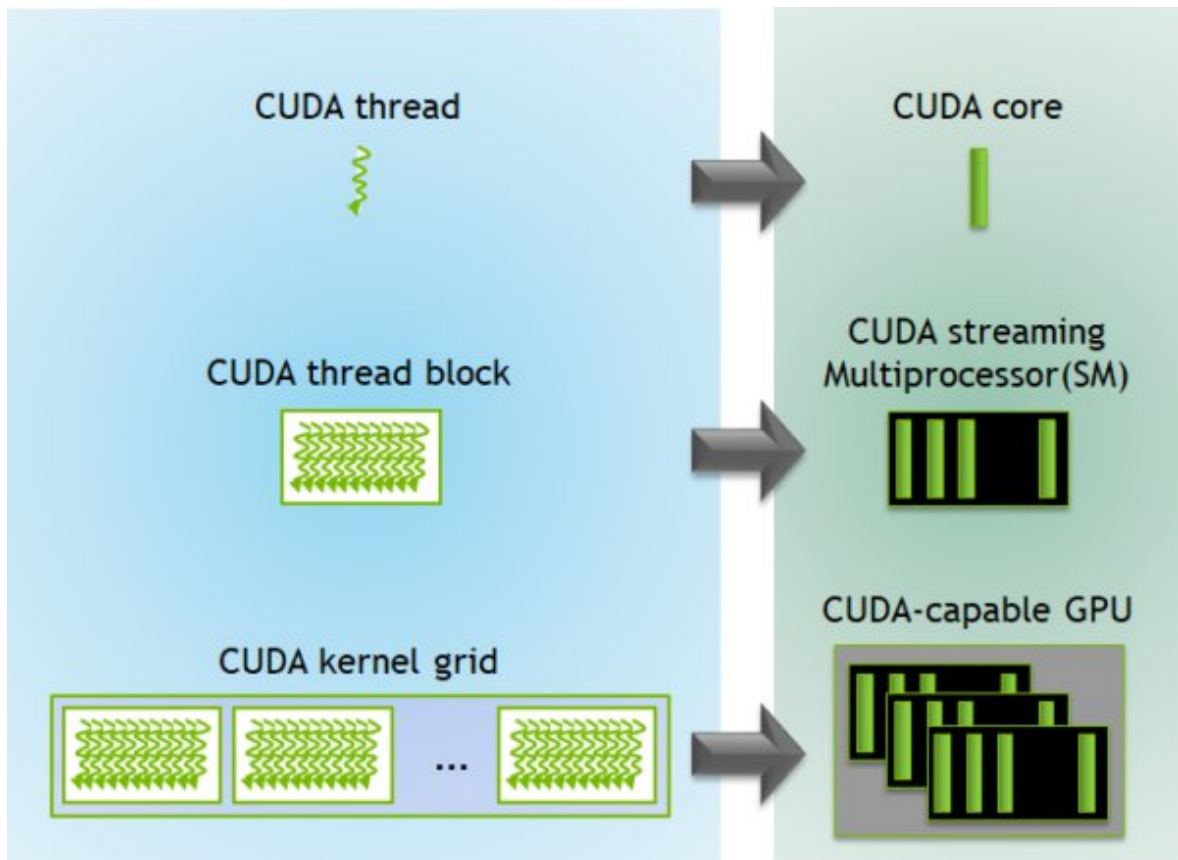
NVIDIA DEEP LEARNING SDK and CUDA

CUDA engineering for deep learning on GPU is handled by cuDNN.

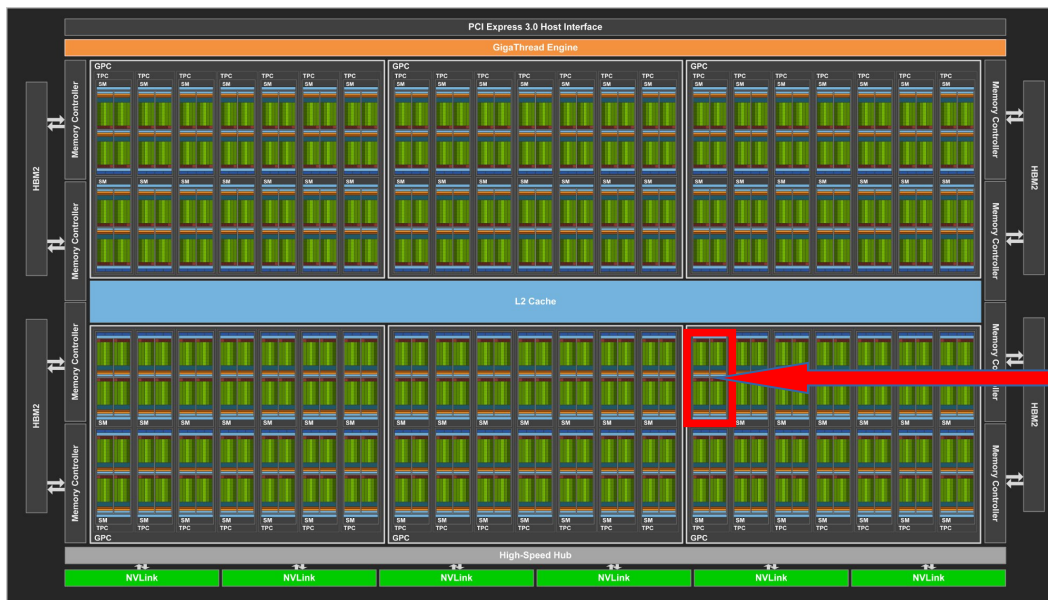
Thanks cuDNN!!

Recommendation: to optimize the use of Tensor Cores and Cuda Cores: Use tensors with dimensions (batch size, sample size, channel, layer dimension, etc.) **multiples of 8!!**

GPU computing : CUDA



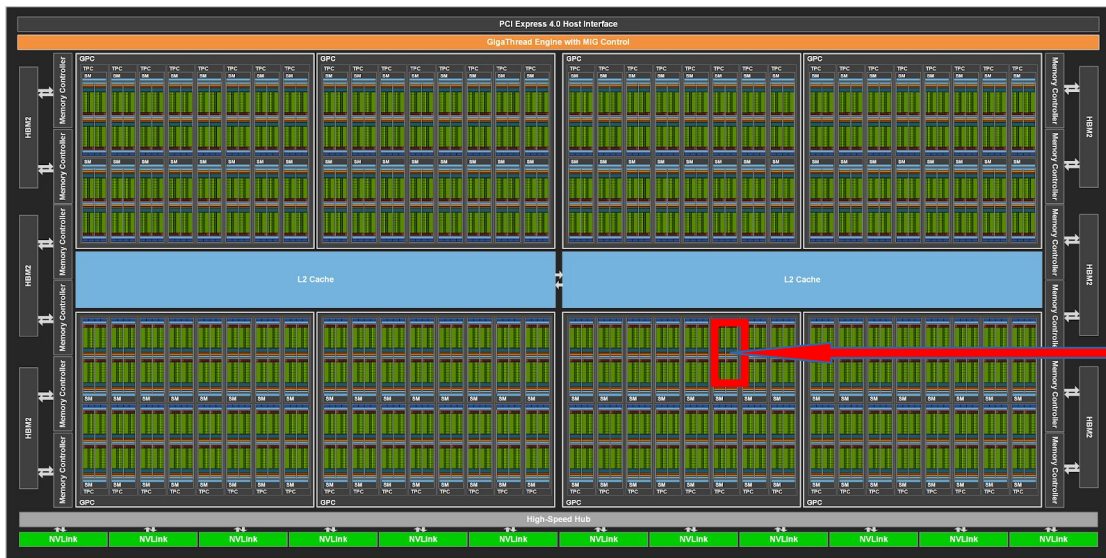
V100 Architecture



- 6 GPC
- **84** Streaming Multiprocessors (SMs)
- 5376 CUDA Cores
- **672 2eGen** Tensor Cores per full GPU

Source : [NVidia](#)

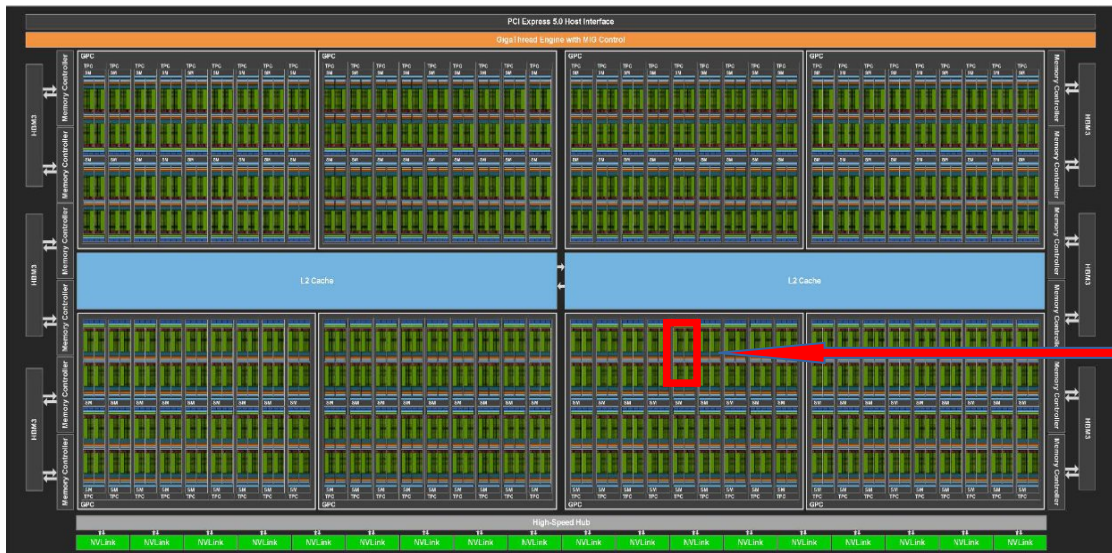
A100 Architecture



- 8 GPC
- **128** Streaming Multiprocessors (SMs)
- 8192 CUDA Cores
- **512 3eGen** Tensor Cores per full GPU

Source : [NVidia](#)

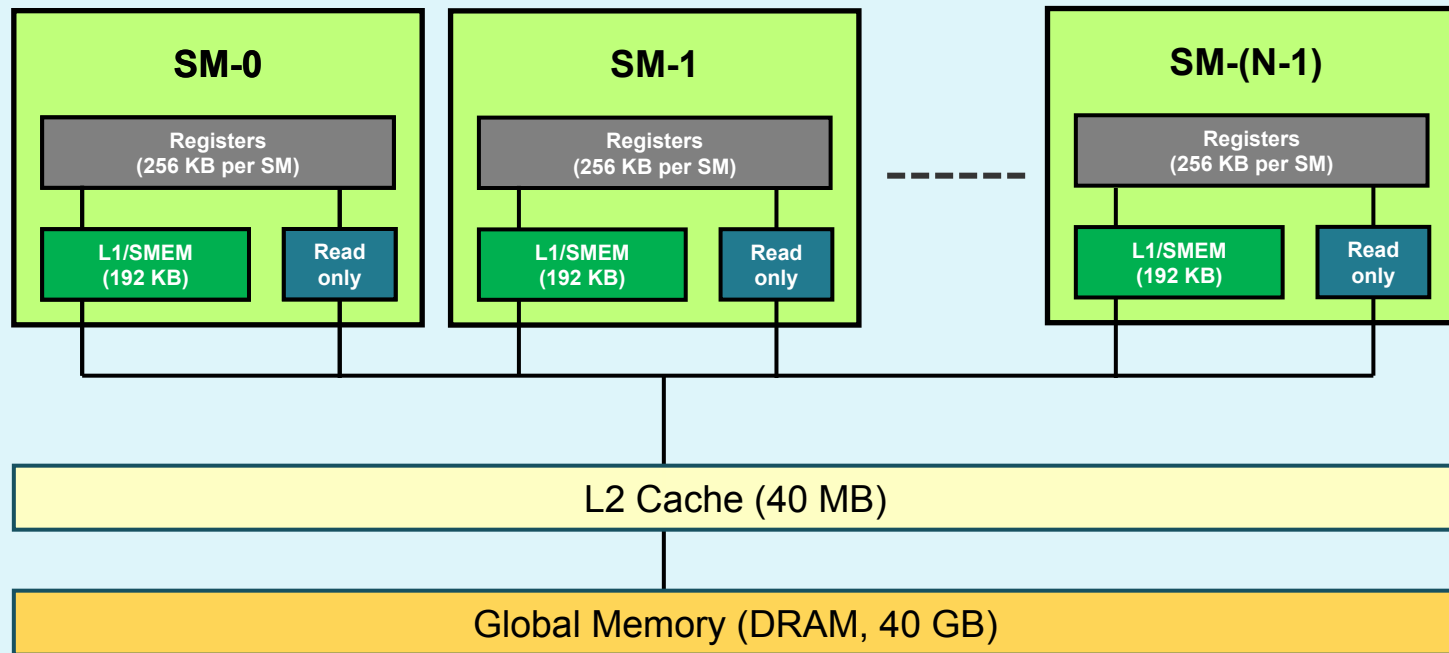
H100 Architecture



- 8 GPC
- **132** Streaming Multiprocessors (SMs)
- 16896 CUDA Cores
- **528 4eGen** Tensor Cores per full GPU

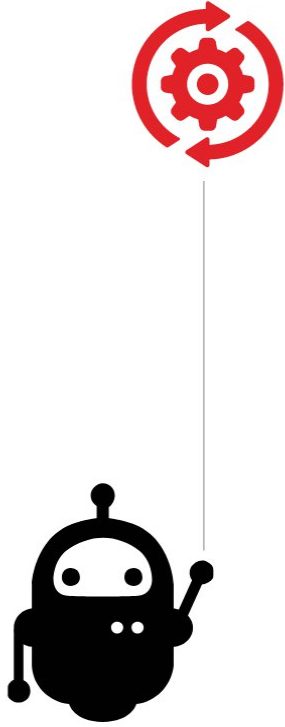
Source : [NVidia](#)

Optimized memory management





TP1 : Accélération GPU



- Envoyer le calcul sur le GPU
- Test Mémoire

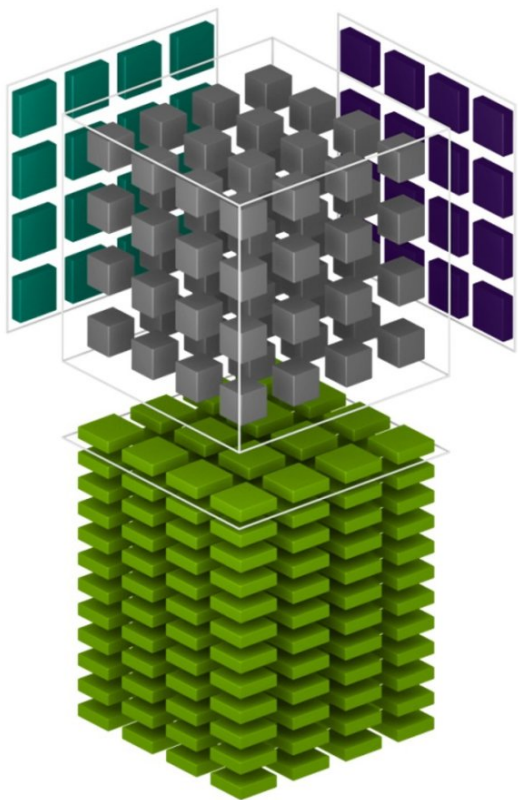
Tensor Cores

Tensor Cores ◀

Precisions ◀

AMP ◀

Channel last memory format ◀



CUDA Core are specialized for **vector computing**.

Tensor Cores are specialized for **matrix calculation**.

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

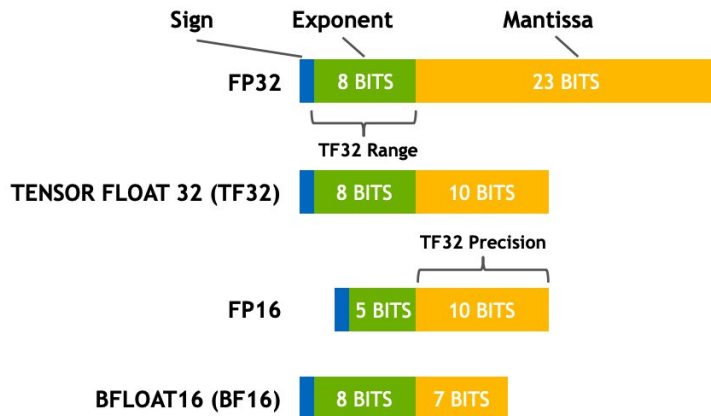
FP16 or FP32 FP16 FP16 FP16 or FP32

Each Tensor Core is capable of processing 64 operations in 1 clock time.

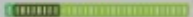
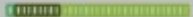
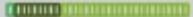
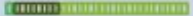
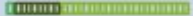
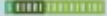
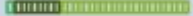
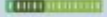
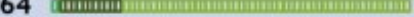
Precisions & Tensor Cores

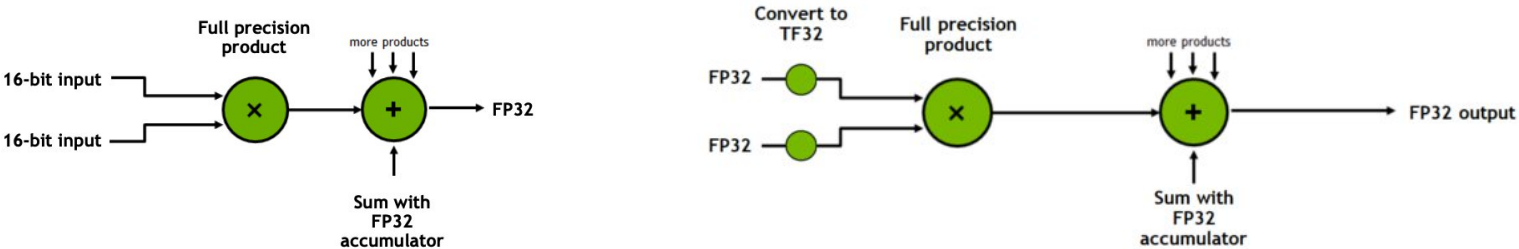


	NVIDIA H100	NVIDIA A100	NVIDIA Volta
Supported Tensor Core Precisions	FP8, FP64, TF32, bfloat16, FP16, ...	FP64, TF32, bfloat16, FP16, INT8, INT4, INT1	FP16
Supported CUDA [®] Core Precisions	FP64, FP32, FP16, bfloat16, INT8	FP64, FP32, FP16, bfloat16, INT8	FP64, FP32, FP16, INT8

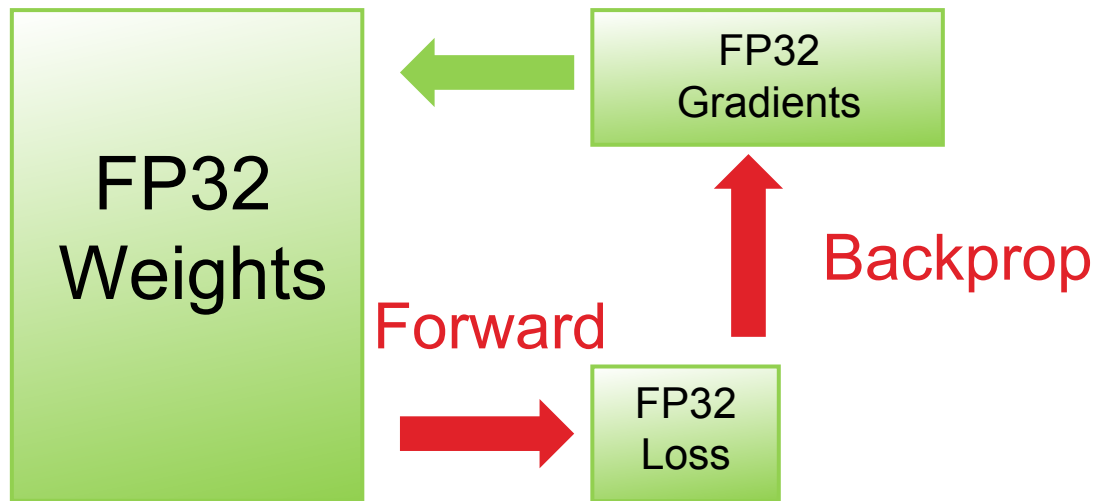


Precisions & Tensor Cores

	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA	SPARSE TOPS	SPARSE X-factor vs. FFMA
V100	FP32 	FP32 	15.7	1x	-	-
	FP16 	FP32 	125	8x	-	-
A100	FP32 	FP32 	19.5	1x	-	-
	TF32 	FP32 	156	8x	312	16x
	FP16 	FP32 	312	16x	624	32x
	BF16 	FP32 	312	16x	624	32x
	FP16 	FP16 	312	16x	624	32x
	INT8 	INT32 	624	32x	1248	64x
	INT4 	INT32 	1248	64x	2496	128x
	BINARY 	INT32 	4992	256x	-	-
	IEEE FP64 		19.5	1x	-	-

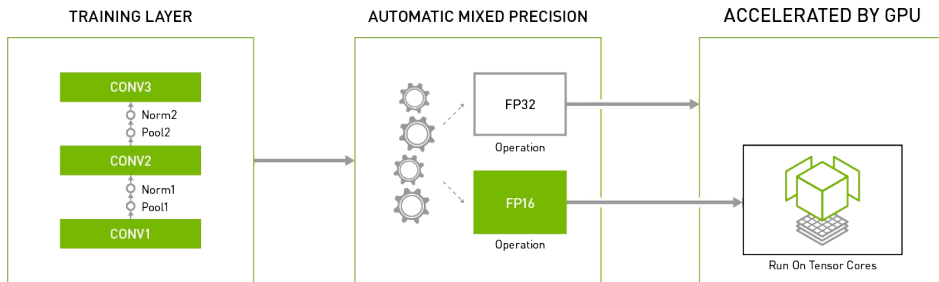


Normal Precision Training Loop

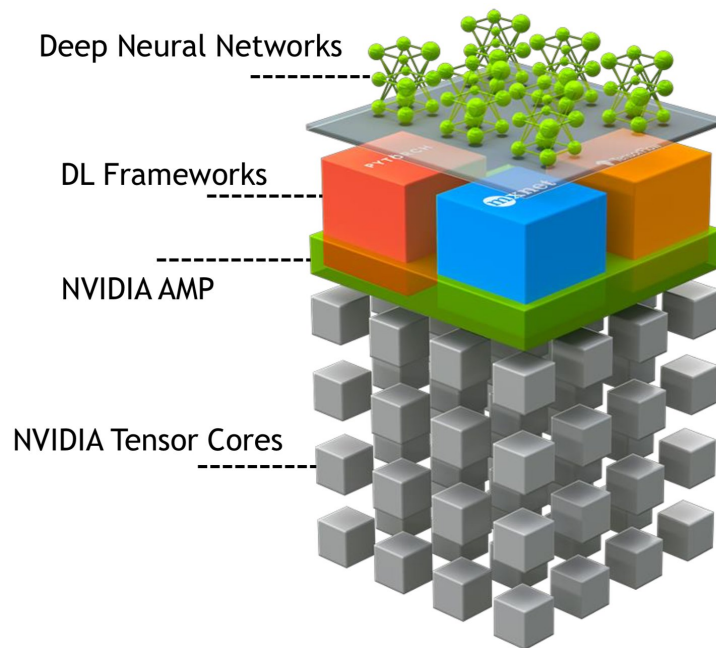


Automatic Mixed Precision

- Automatic Mixed Precision :
 - Necessary with V100 to use Tensor Core
 - The A100s use Tensor Cores with or without MP



- Pros:
 - **Insignificant** loss of precision for model training (gradient, loss, accuracy)
 - **Reduces memory** footprint
 - **Speeds up** calculations
- 2 steps to code :
 - Transforms eligible layers into FP16
 - Uses scaling to calculate gradients

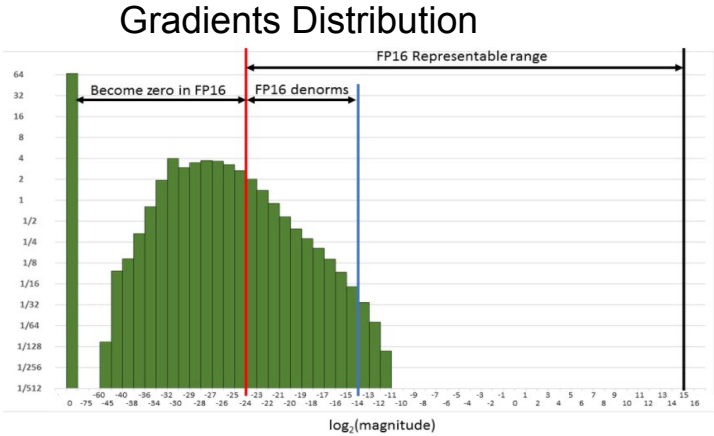


AMP FP16 with Scaler

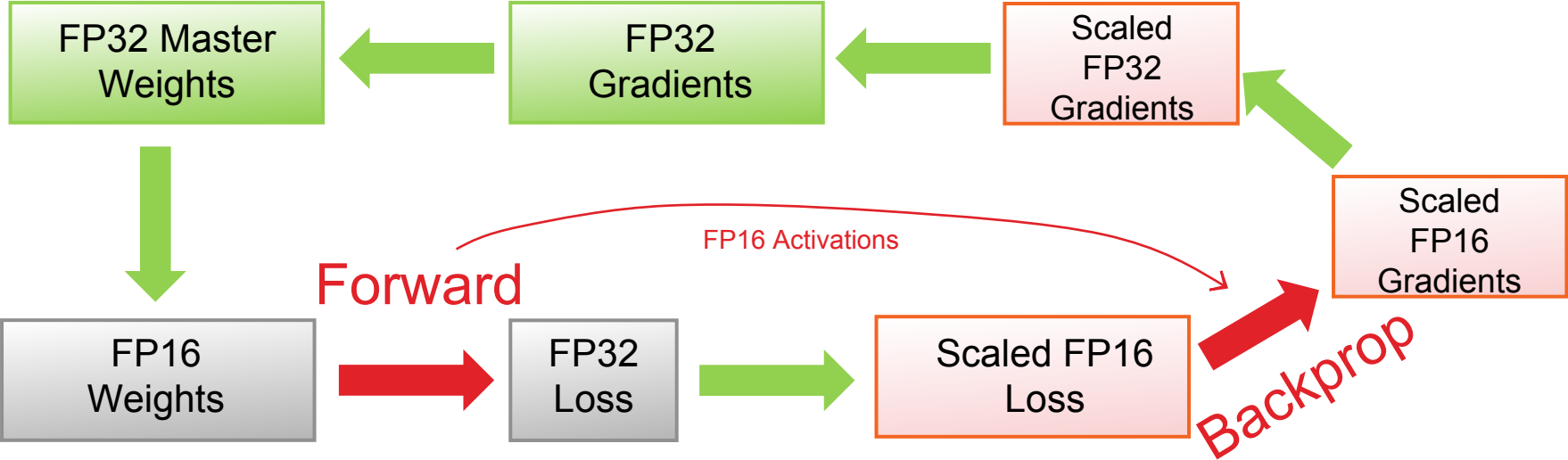
FP16



In FP16, values lower than 2^{-24} ($5.96e^{-8}$) are considered 0.



Optimizer

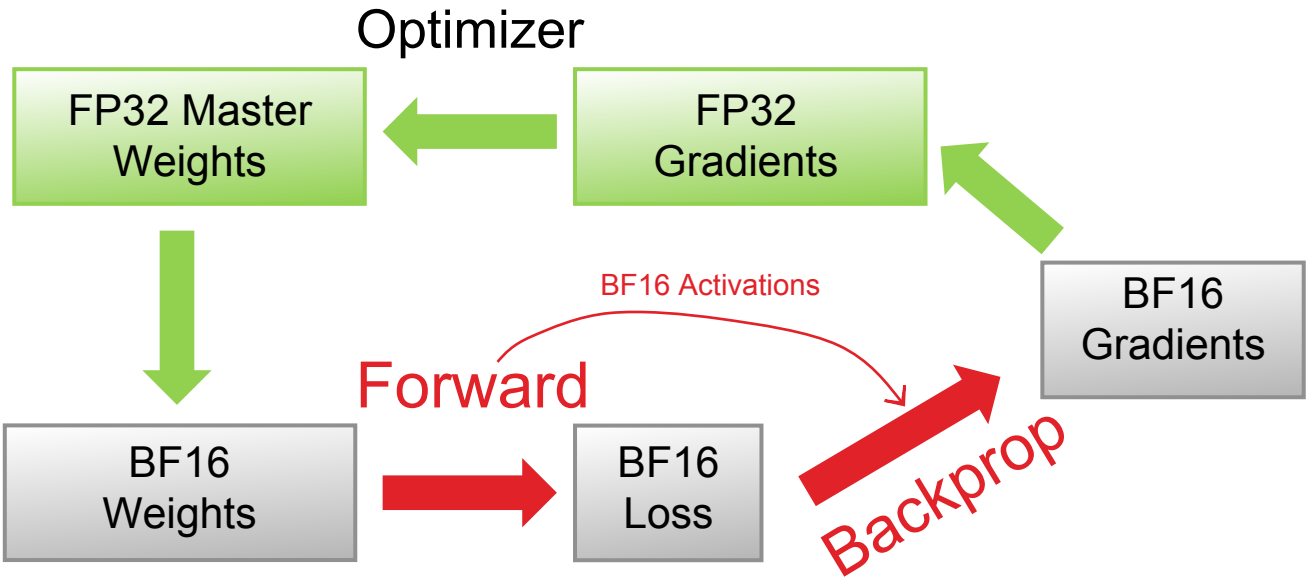


AMP BF16 without Scaler

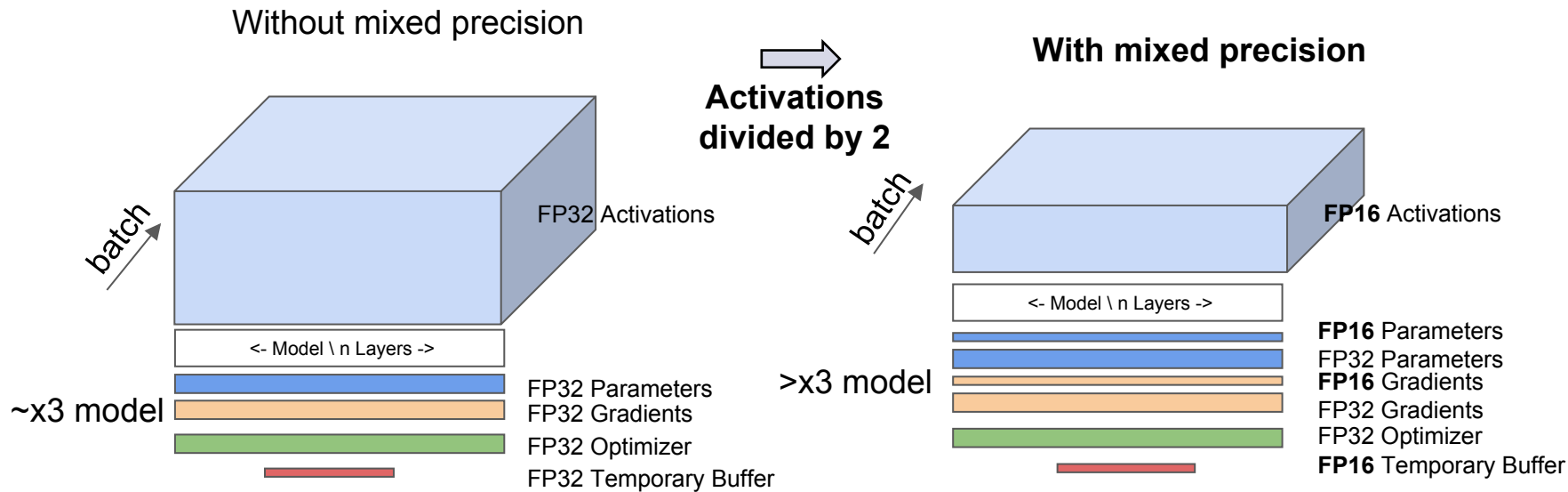
BF16



No Scaler !



Memory Footprint with Mixed Precision

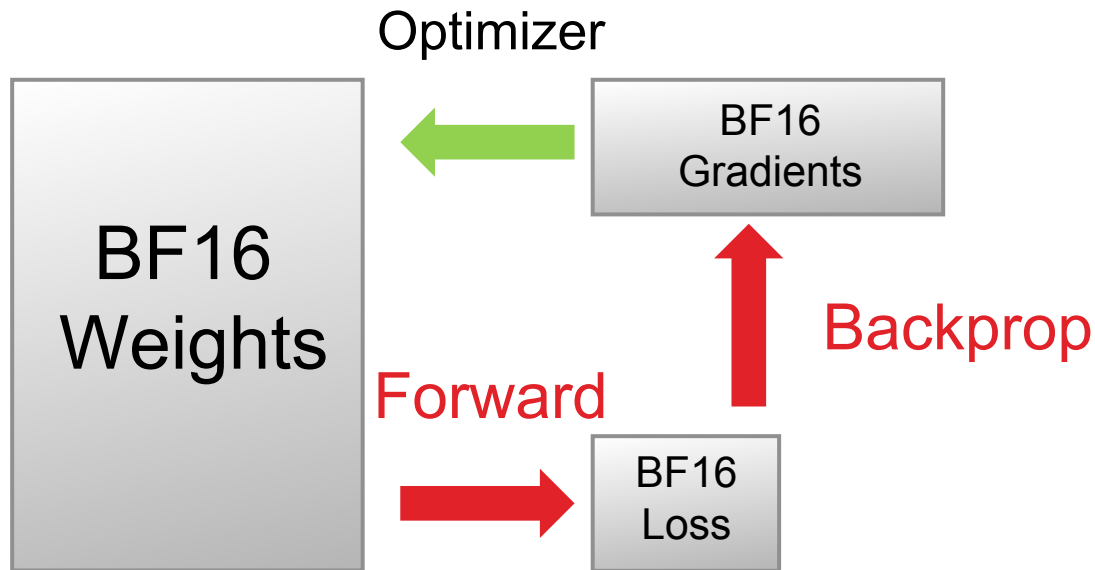


Full Degraded Precision (BF16)

BF16

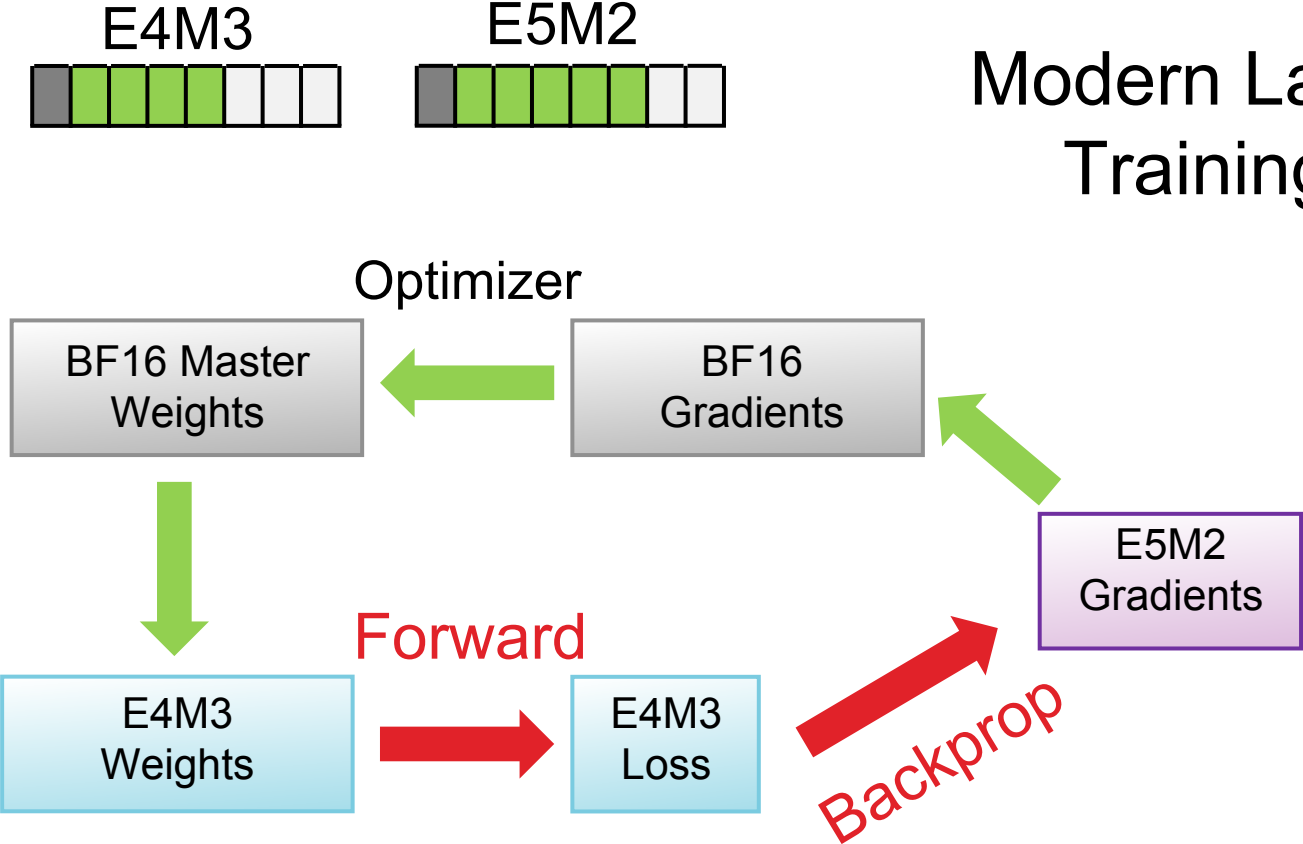


Modern Large Model
Training Way !



FP8 Mixed Precision

Modern Large Model
Training Way !



Channel last memory format

batch channel height width

NCHW

`.shape()`



memory contiguity by default

classic (contiguous) memory storage of NCHW tensor :

`.stride()`

b 0x: 0 1 2 3 4 5 6 7 8 9 a b c d e f 0 1 2 3 4 5 6 7 8 9 a b c d e f 0 1 2 3 4 5 6 7 8 9 a b c d e f

a 0x: 0 1 2 3 4 5 6 7 8 9 a b c d e f 0 1 2 3 4 5 6 7 8 9 a b c d e f 0 1 2 3 4 5 6 7 8 9 a b c d e f

Channels last memory format orders data differently:

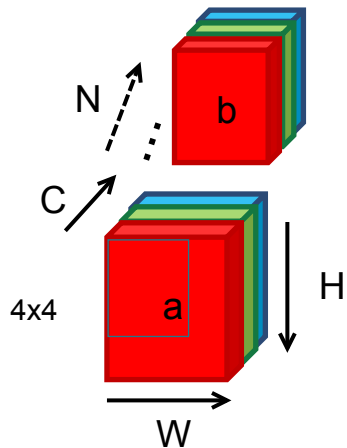
`.stride()`

b 0x: 0 0 0 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6 7 7 7 8 8 8 9 9 9 a a a b b b c c c d d d e e e f f f

a 0x: 0 0 0 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6 7 7 7 8 8 8 9 9 9 a a a b b b c c c d d d e e e f f f

3x3

Convolution filter



TP2&3 : Automatic Mixed Precision



- Activer l'Automatic Mixed Precision
- Test Mémoire
- Activer le channel last memory format

