Deep Learning Optimized on Jean Zay

Optimization of the data preprocessing

cnrs

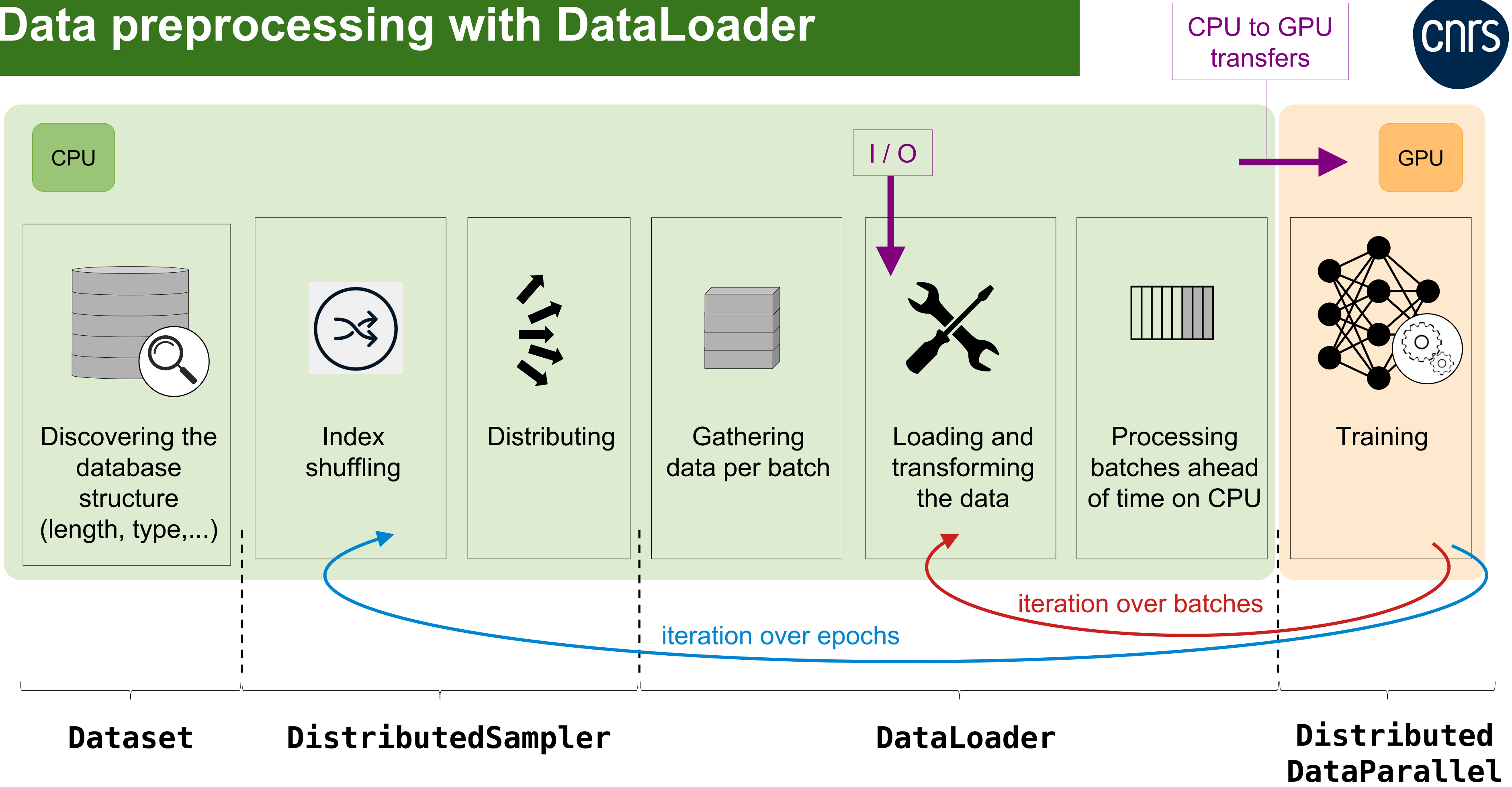
IDRIS

Optimization of the data preprocessing

Data preprocessing with DataLoader ◂

Optimization of the DataLoader ◂

Data preprocessing with DataLoader



Data preprocessing with DataLoader

- **`DataLoader`** (data preprocessing)

```
from torch.utils.data import DataLoader



data_loader = DataLoader(dataset,
                         batch_size=batch_size,
                         num_workers=<int>,
                         persistent_workers=<bool>,
                         prefetch_factor=<int>,
                         pin_memory=<bool>,
                         drop_last=<bool>
                        )
```

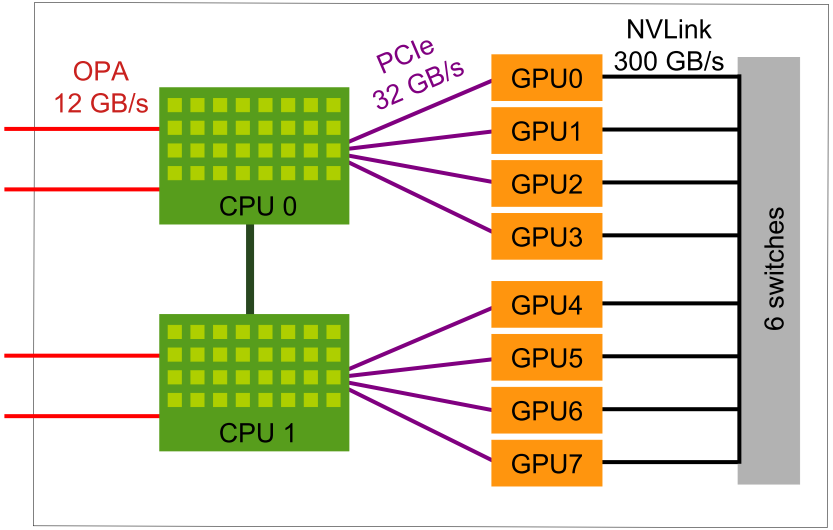
Optimization of the data preprocessing

Data preprocessing with DataLoader ◂

Optimization of the DataLoader ◂

Optimization of the DataLoader

- Crucial points regarding the performance of data preprocessing:



Node 8 × A100 80Go

1. Loading the data in memory and transforming it on the CPU

2. Data transfers from CPU to GPU

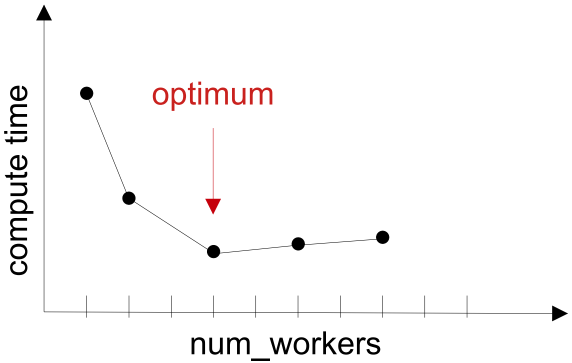
Optimization of the DataLoader

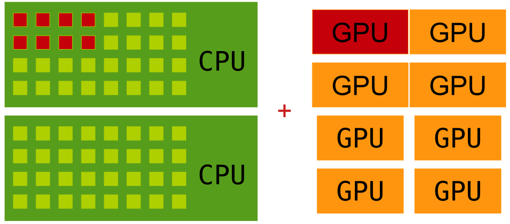
1. Loading the data in memory and transforming it on the CPU

- **`num_workers`** allows us to define the number of processes (CPU cores) which will work in parallel to preprocess the data on the CPU.

✓ Compute time speedup on CPU.

⚠ The multiprocessing environment which is created occupies some space in the CPU RAM.





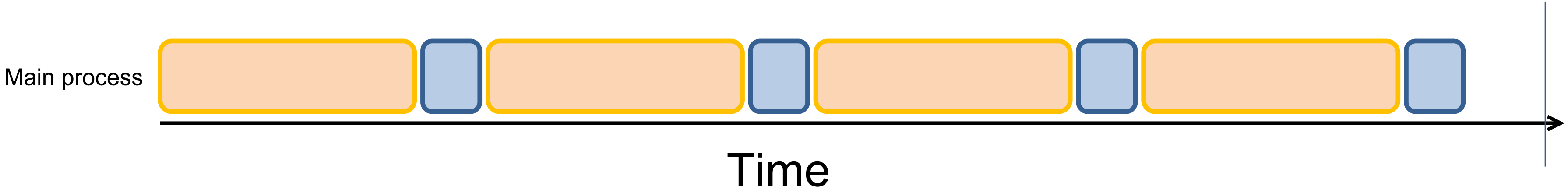

```
#SBATCH --ntasks=1
#SBATCH --gres=gpu:1
#SBATCH --cpus-per-task=8
```

Optimization of the DataLoader

num_worker = 0

DataLoader

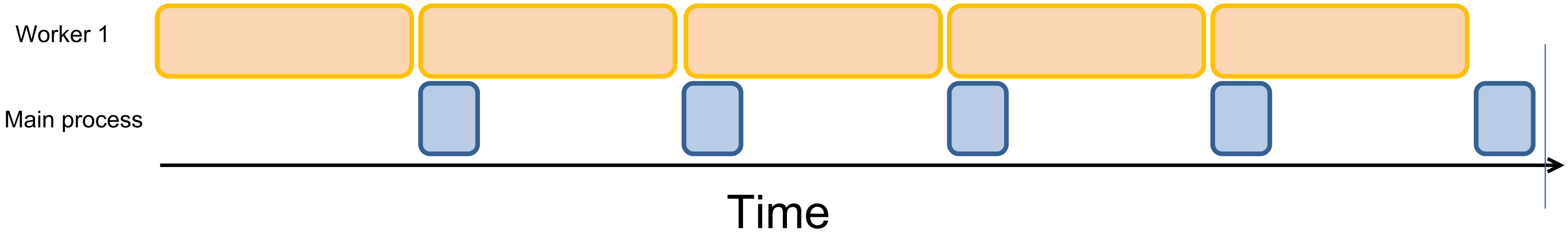
Forward/Backward



Optimization of the DataLoader

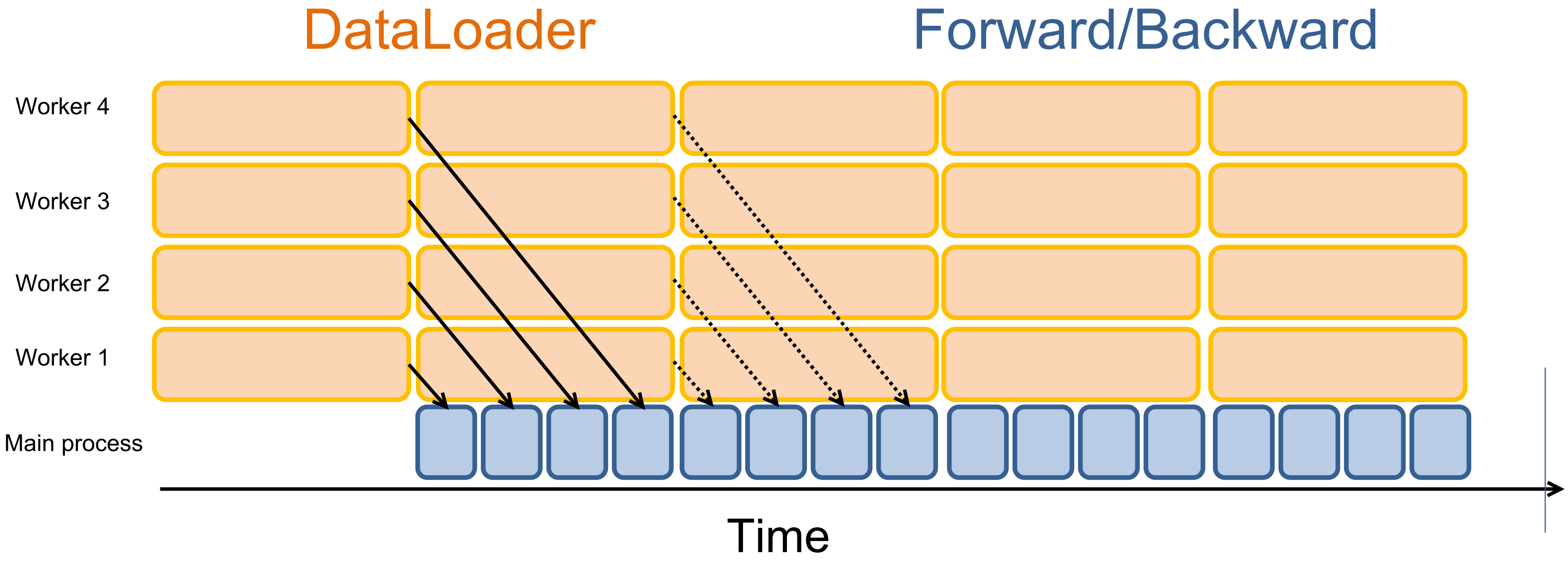
num_worker = 1

DataLoader Forward/Backward



Optimization of the DataLoader

num_worker = 4



10

Optimization of the DataLoader

1. Loading the data in memory and transforming it on the CPU

- `num_workers` allows us to define the number of processes (CPU cores) which will work in parallel to preprocess the data on the CPU.

- **`persistent_workers=True`** allows us to maintain the active processes throughout the training.

✔ Time gain: We avoid reinitializing the processes at each epoch.

⚠ Usage of the CPU RAM (can become an issue if multiple DataLoaders are used).

Optimization of the DataLoader

1. Loading the data in memory and transforming it on the CPU

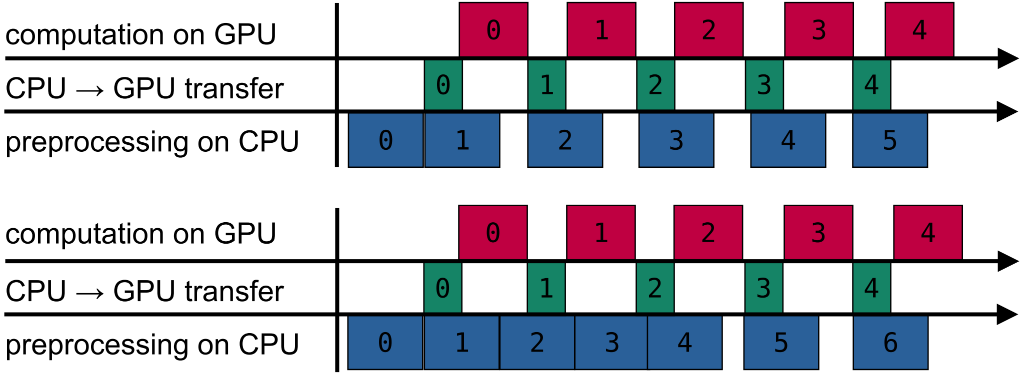
- **`prefetch_factor`** allows us to define the maximum number of batches the CPU can preprocess in advance.

✓ Prevents GPU inactivity if CPU occasionally struggles

⚠ Usage of the CPU RAM

`prefetch_factor = 1`

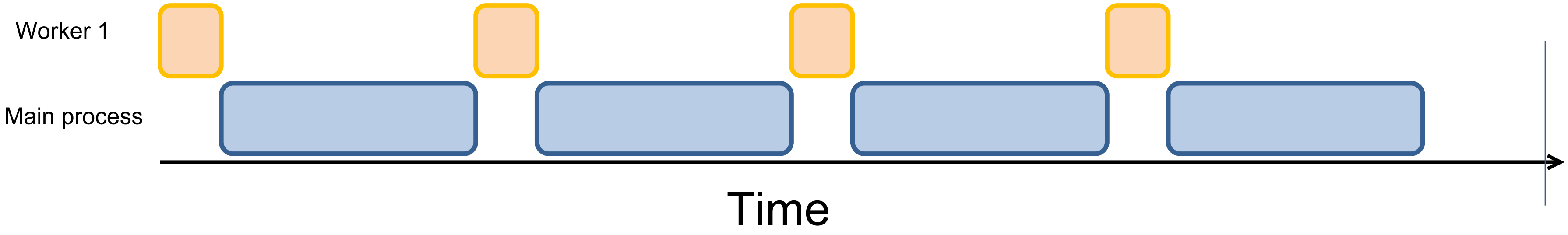
`prefetch_factor = 2`



Optimization of the DataLoader

num_workers = 1, prefetch_factor = None

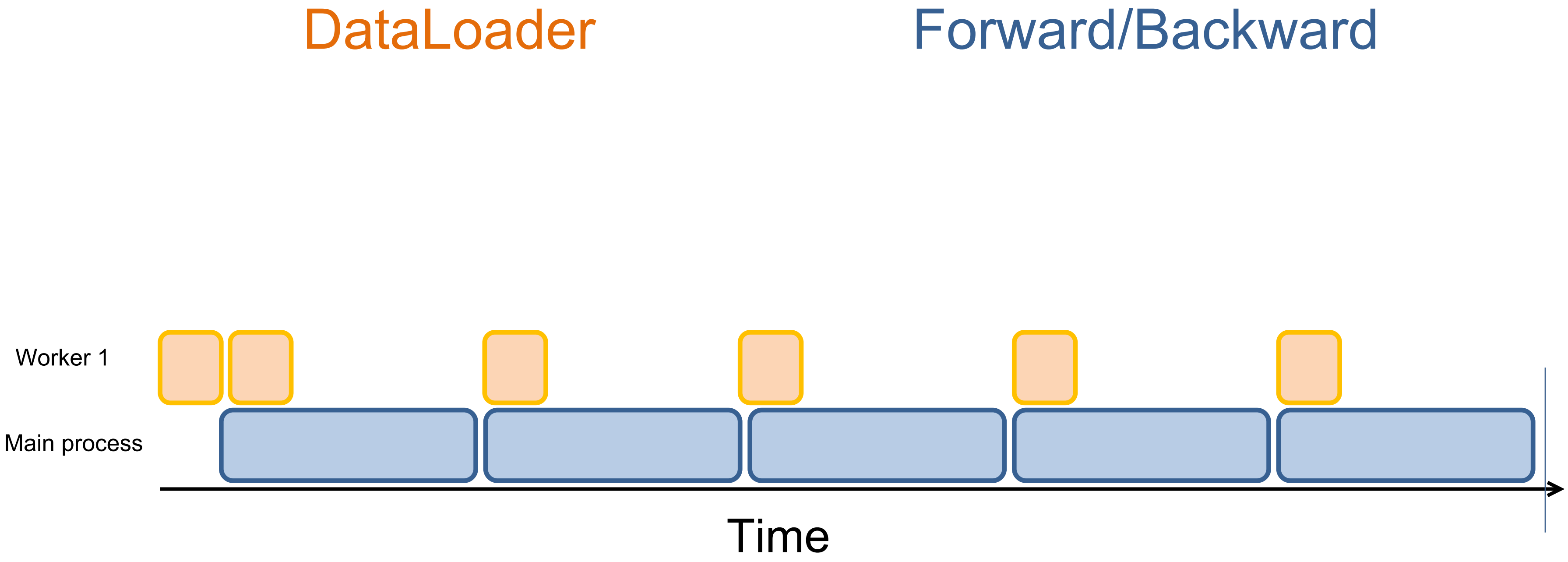
DataLoader Forward/Backward



13

Optimization of the DataLoader

num_workers = 1, prefetch_factor = 1

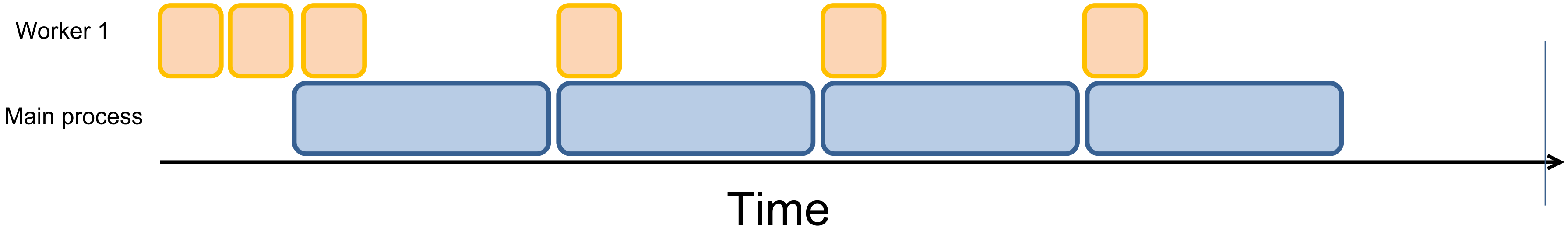


14

Optimization of the DataLoader

num_workers = 1, prefetch_factor = 2

DataLoader Forward/Backward

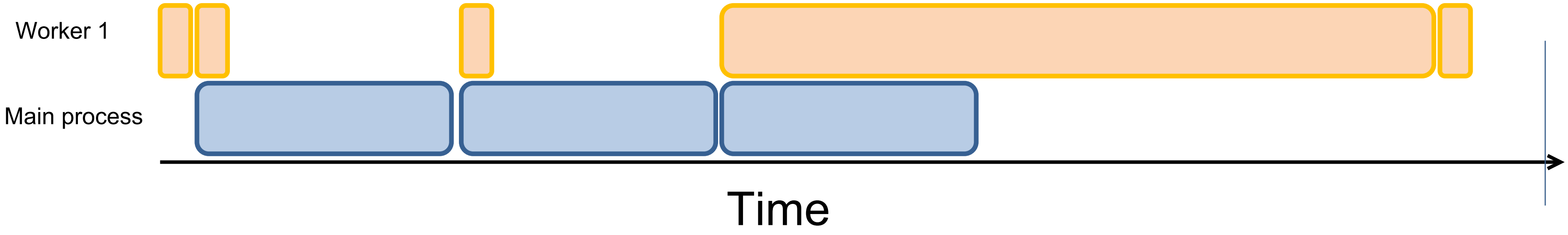


15

Optimization of the DataLoader

num_workers = 1, prefetch_factor = 1

DataLoader Forward/Backward

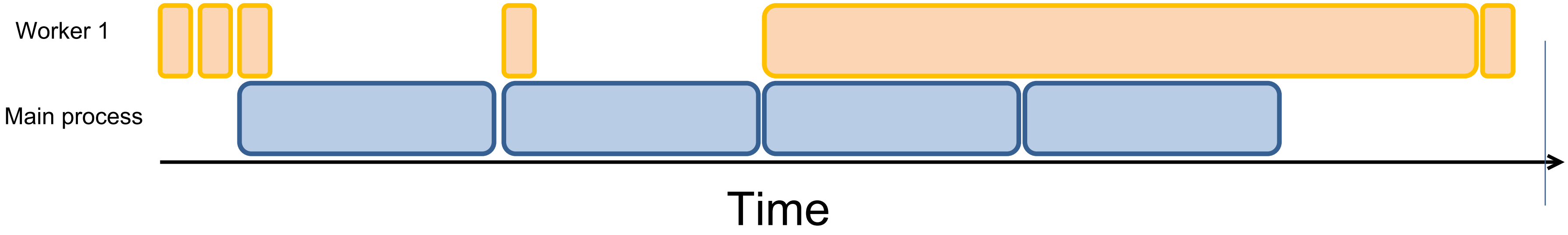


16

Optimization of the DataLoader

num_workers = 1, prefetch_factor = 2

DataLoader Forward/Backward

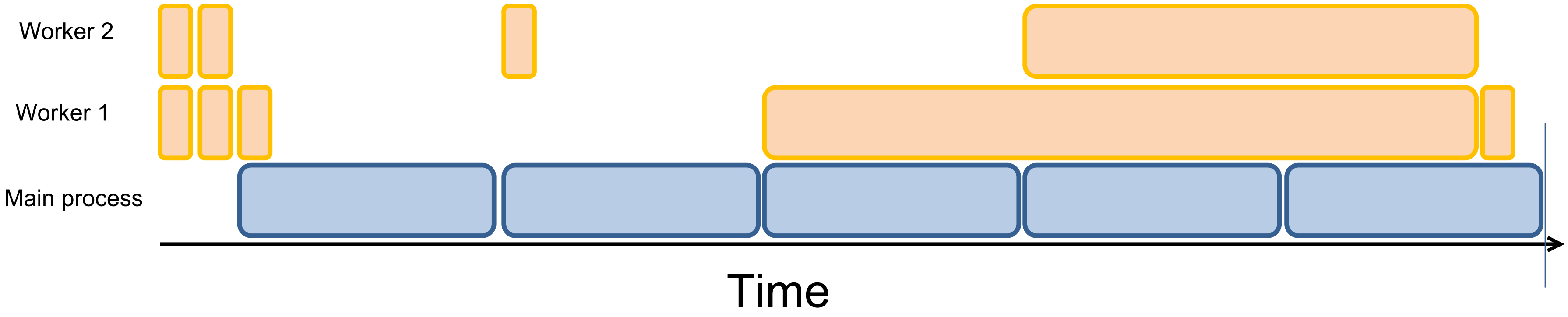


17

Optimization of the DataLoader

num_workers = 2, prefetch_factor = 2

DataLoader Forward/Backward



18

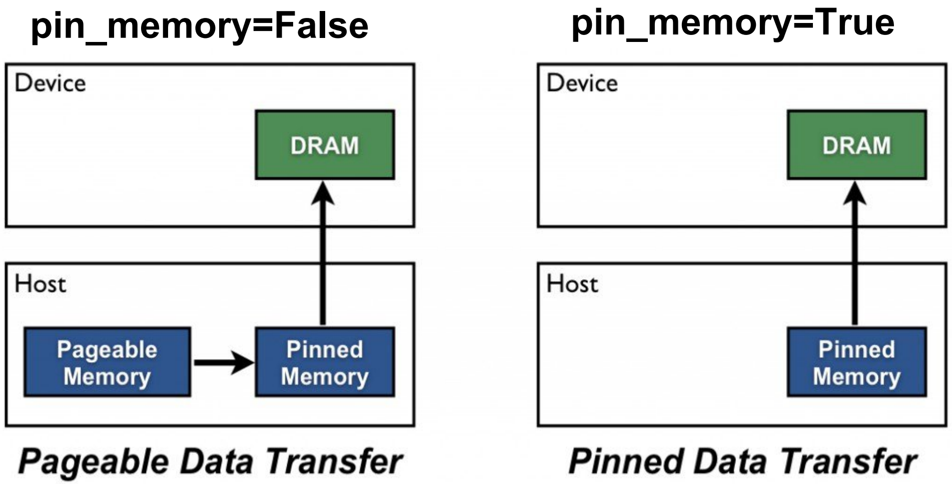
Optimization of the DataLoader

2. Data transfers from CPU to GPU

- `pin_memory=True` allows storing batches directly in pinned memory.

✓ Speedup of CPU/GPU transfers

⚠ Slows CPU memory management



https://developer.nvidia.com/blog/how-optimize-data-transfers-cuda-cc/

Optimization of the DataLoader

2. Data transfers from CPU to GPU

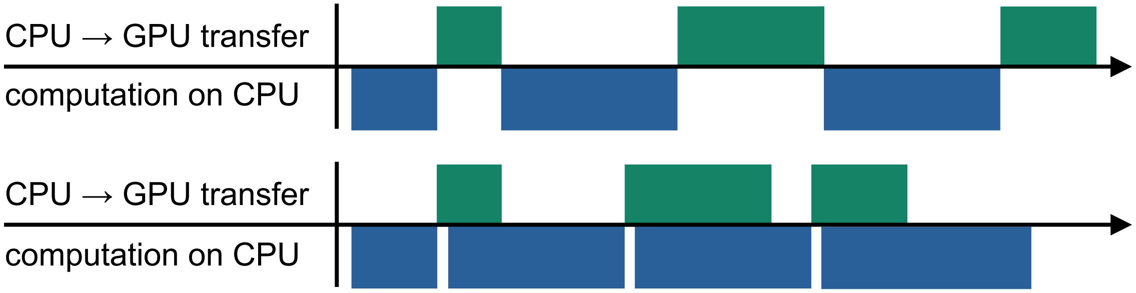
- `pin_memory=True` allows storing batches in pinned memory.

✓ Storing on pinned memory allows activating the **asynchronism** mechanism during the transfers of CPU to GPU : data = data.to(gpu, **non_blocking=True**).

⚠ Usage of the CPU RAM (intermediate memory buffers).

`non_blocking=False`

`non_blocking=True`



Optimization of the DataLoader

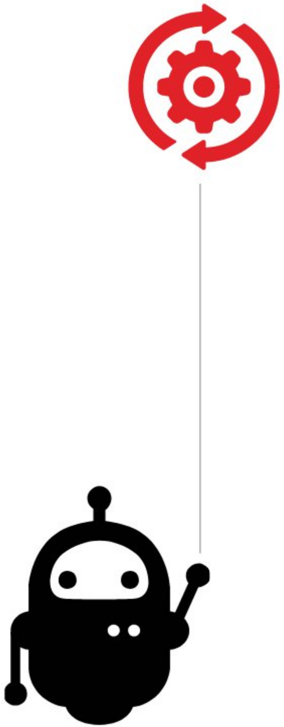
- Other DataLoader option:
  - **`drop_last=True`** allows us to ignore the last samples if the size of the dataset is not a multiple of the number of batches.

✓ The workload per process is balanced.

✓ We avoid the cost of treating an incomplete batch.

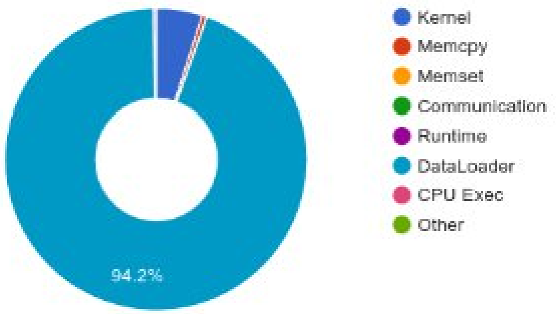
⚠ Loss of information?

TP2_3: Optimization of the DataLoader



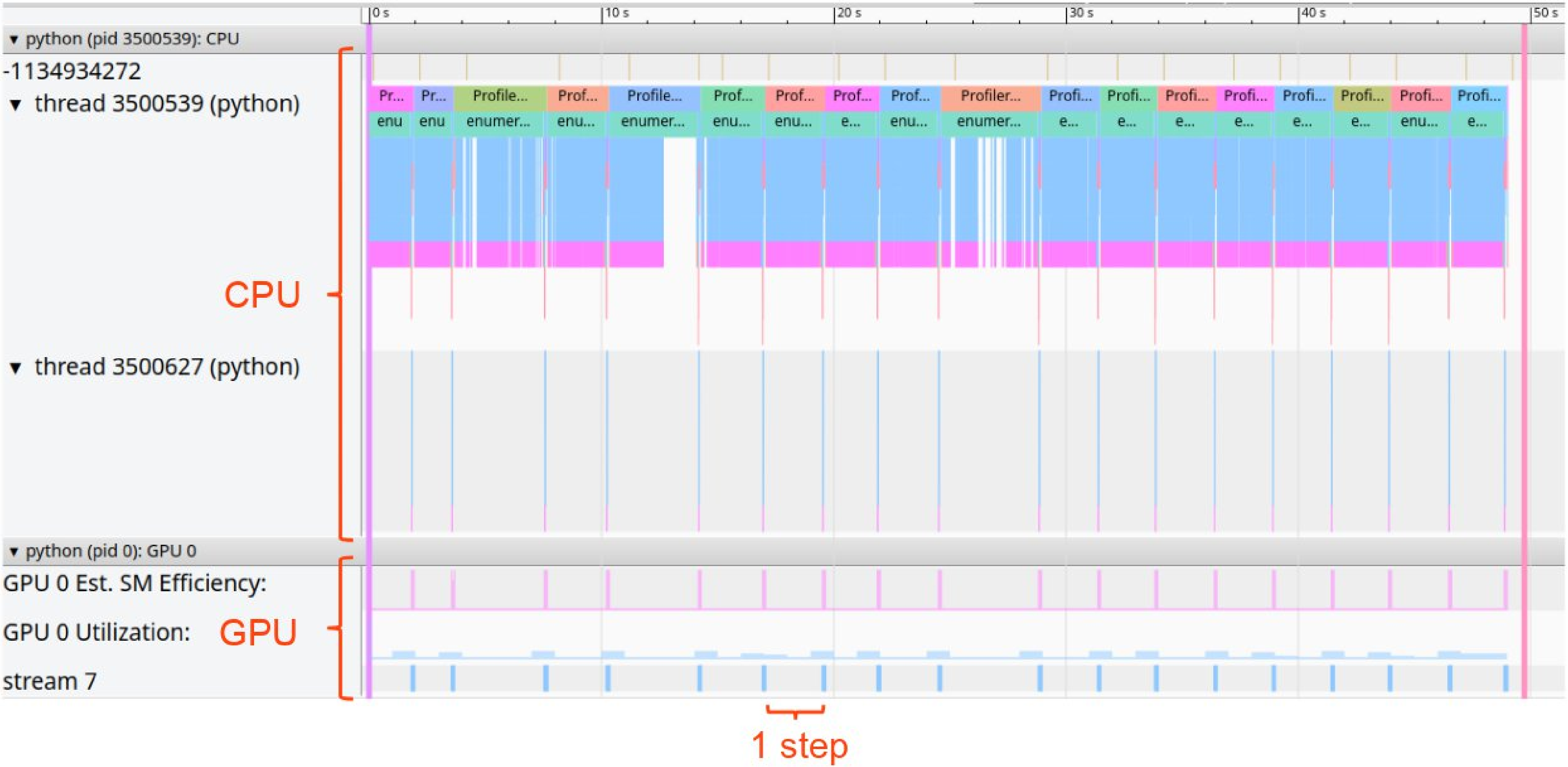
- Modify the DataLoader options.
- Measure the time gain on a few steps.

TP2_1: Profiler PyTorch (conclusion)

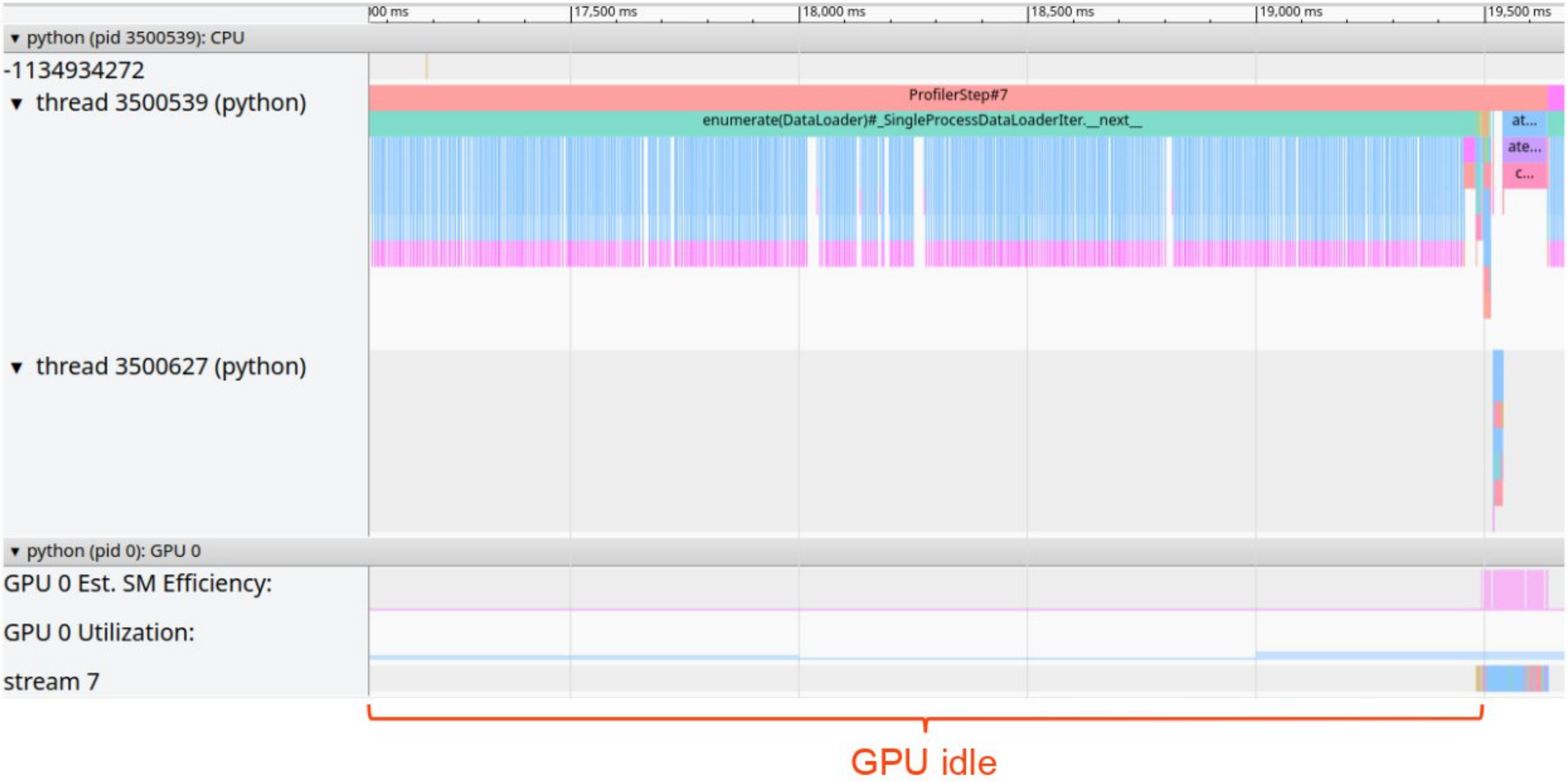


After seeing the traces, it is obvious that the optimization efforts need to concentrate on the DataLoader.

TP2_2: Profiler Trace



TP2_2: Profiler Trace (1 step)

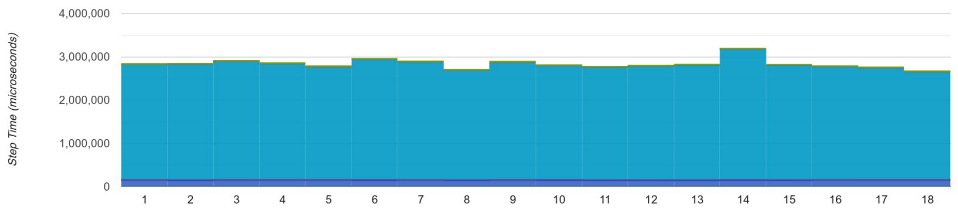


TP2_2: Profiler Trace (1 step - CPU)

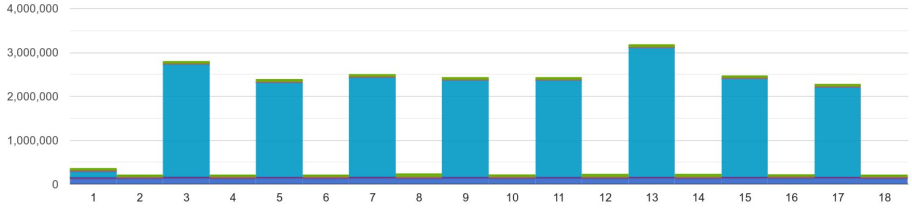
TP2_1: Optimization of the DataLoader

- The most efficient optimization is the increase of `num_workers`.

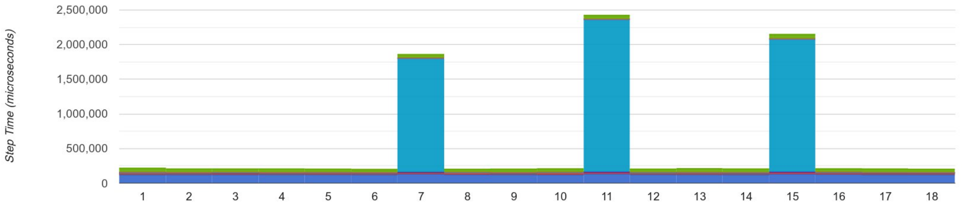
`num_workers=0`



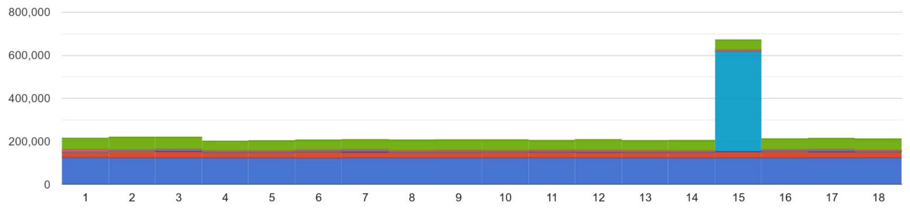
`num_workers=2`



`num_workers=4`



`num_workers=8`



Kernel Memcpy Memset Communication Runtime DataLoader CPU Exec Other

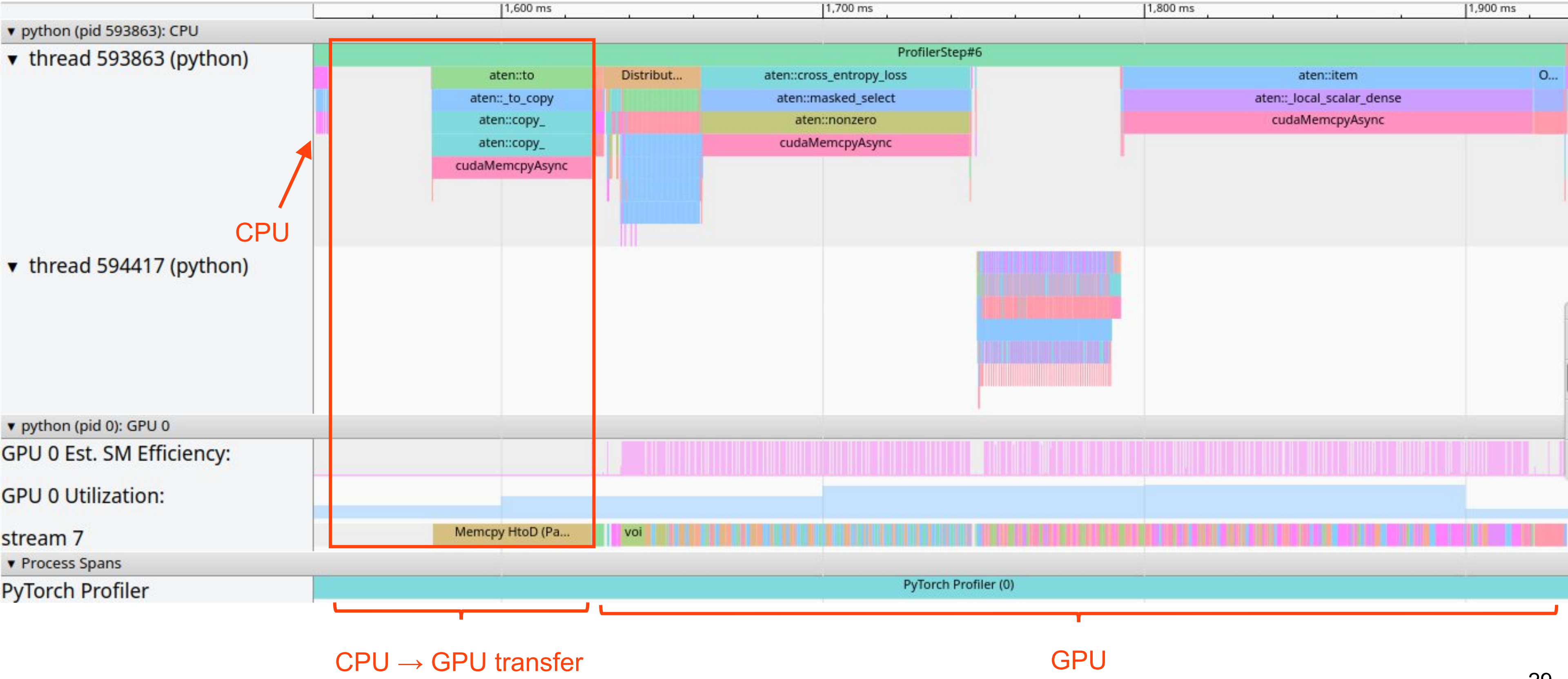
TP2_1: Optimization of the DataLoader

Intermediate conclusion about `num_workers` setting:

- Increase `num_workers` progressively and observe if the DataLoader scales or not on a few steps.
- For low CPU workload, `num_workers` can be a multiple of cpus-per-task.
- Setting too many workers creates bottlenecks or Out Of Memory failures.
- Be aware that few steps are not completely representative.
- IOs on Jean Zay are erratic.

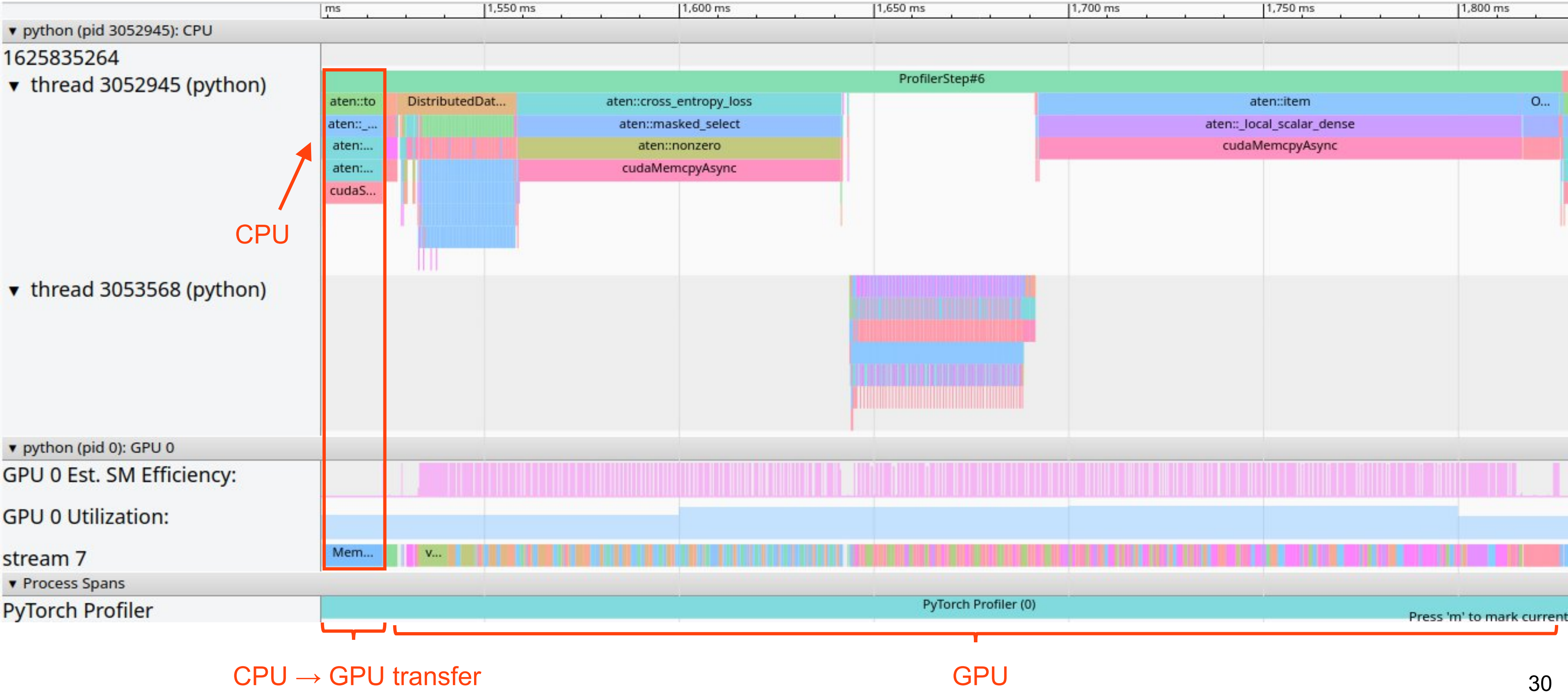
TP2_1: Optimization of the DataLoader

`pin_memory=False, non_blocking=False`



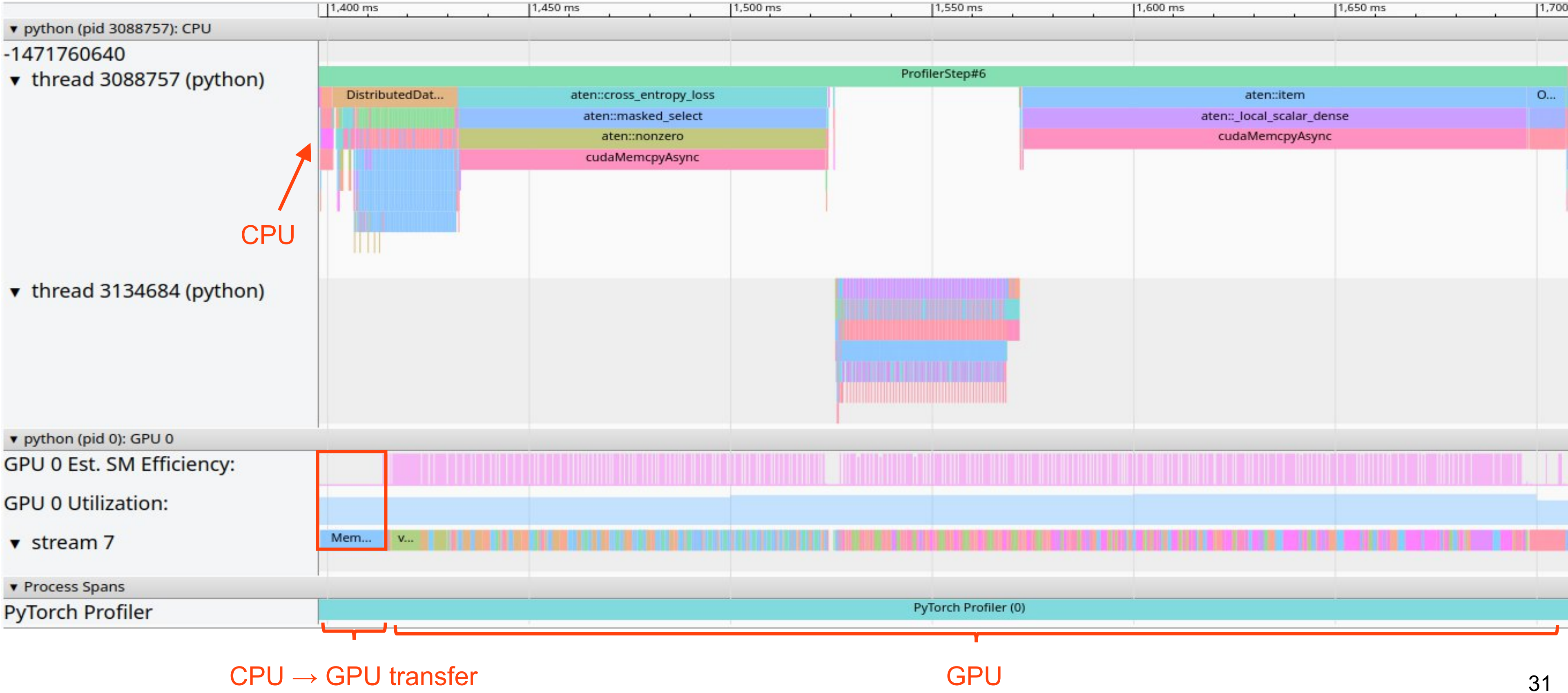
TP2_1: Optimization of the DataLoader

`pin_memory=True, non_blocking=False`



TP2_1: Optimization of the DataLoader

`pin_memory=True, non_blocking=True`



TP2_1: Optimization of the DataLoader

- Chosen optimizations:

```
num_wokers = 16
persistent_workers = True
pin_memory = True
non_blocking = True
prefetch_factor = 2
```

Configuration

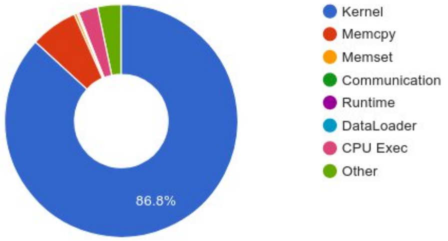
Number of Worker(s)	1
Device Type	GPU

GPU Summary

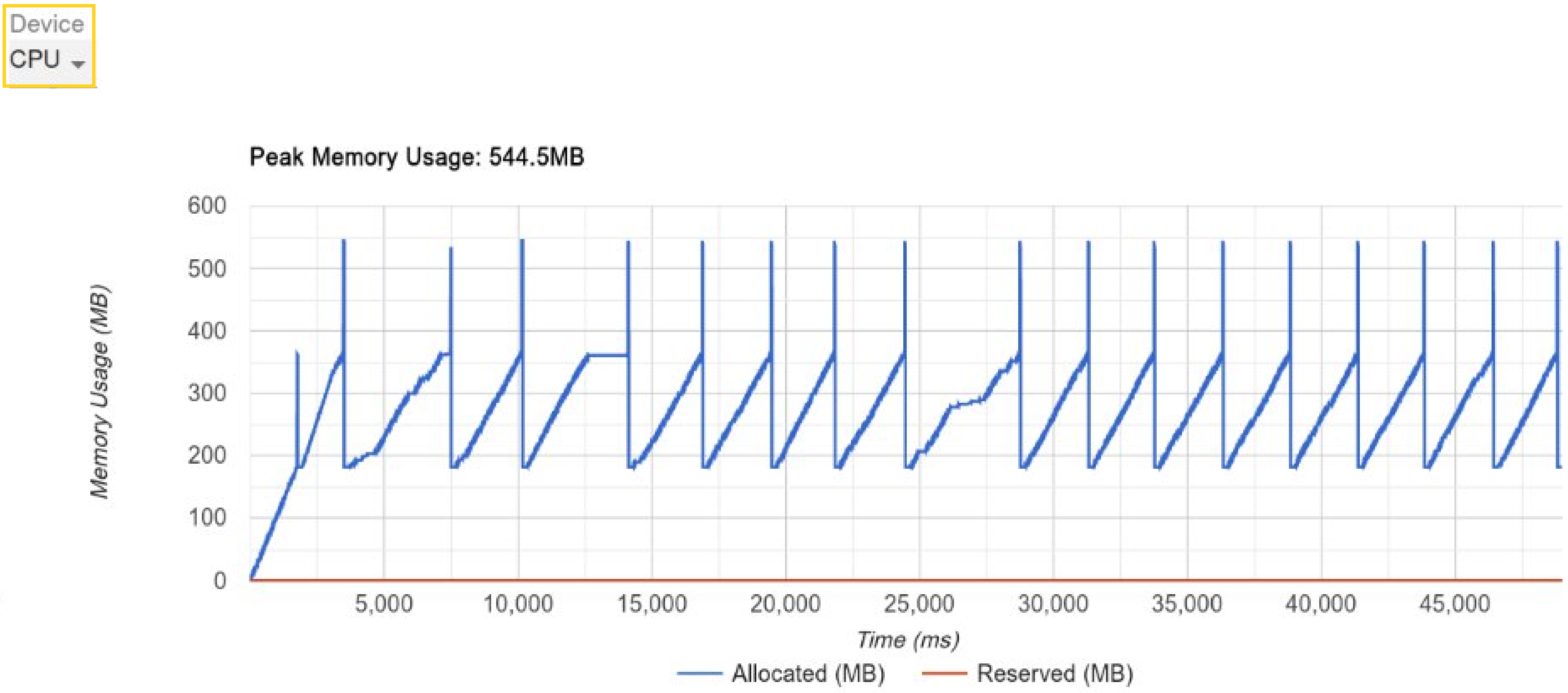
GPU 0:	
Name	NVIDIA A100-SXM4-80GB
Memory	79.15 GB
Compute Capability	8.0
GPU Utilization	86.84 %
Est. SM Efficiency	85.55 %
Est. Achieved Occupancy	32.15 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	142,633	100
Kernel	123,861	86.84
Memcpy	9,311	6.53
Memset	558	0.39
Communication	39	0.03
Runtime	0	0
DataLoader	327	0.23
CPU Exec	3,862	2.71
Other	4,675	3.28



Appendix: Profiler Memory View (CPU)

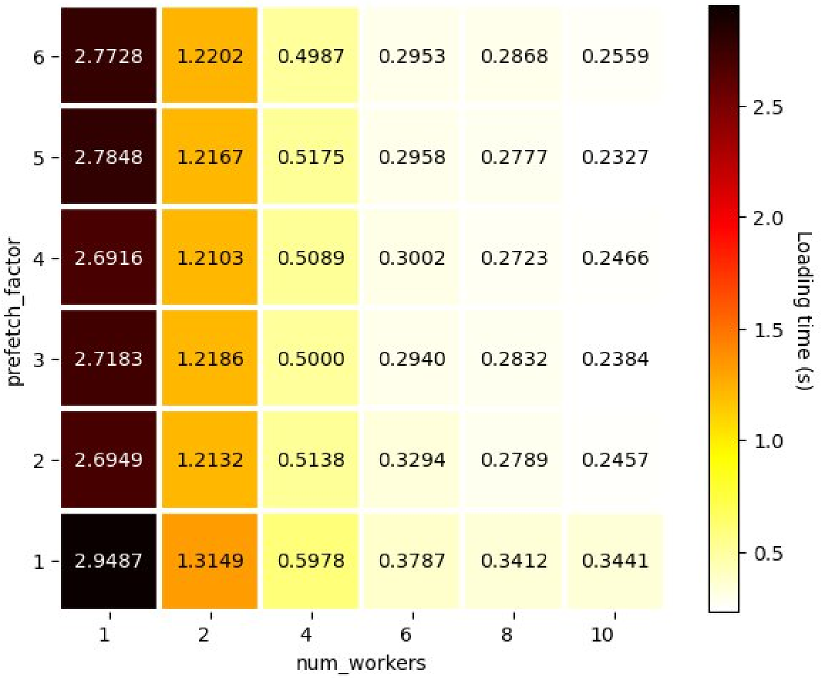


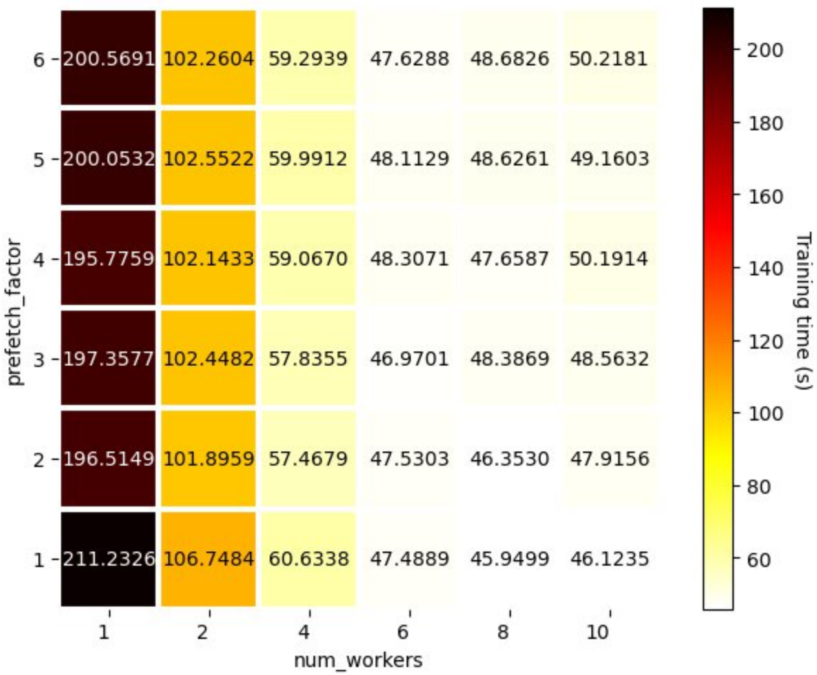
Appendix: Optimization of the DataLoader

- Impact of the **prefetch factor**

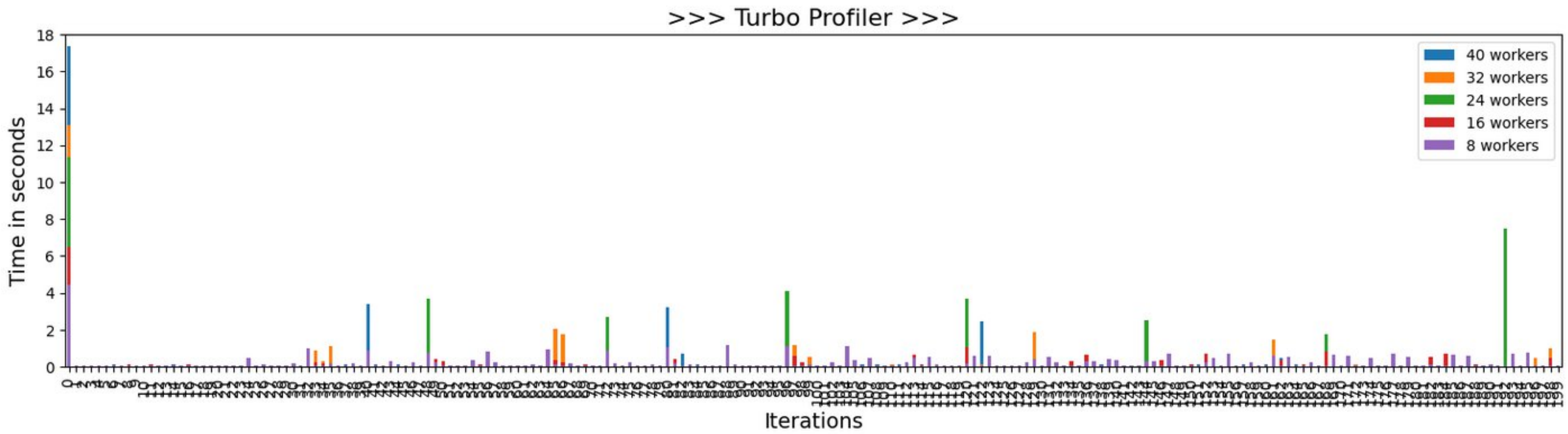
`dlojz.py` - 50 iterations - test partition `gpu_p4`

NB: These results don’t correspond to our usage case but still illustrate the influence of the parameters.





Appendix: Optimization of the DataLoader (resnet 50)



	jobid	num_workers	persistent_workers	pin_memory	non_blocking	prefetch_factor	drop_last	loading_time	training_time
1	830199	16	False	False	False	2	False	0.140631	81.492809s
3	830217	32	False	False	False	2	False	0.145662	146.490717s
4	830224	40	False	False	False	2	False	0.147003	150.194498s
2	830213	24	False	False	False	2	False	0.200591	151.584189s
0	830180	8	False	False	False	2	False	0.204219	87.450866s