Machine Learning for Marine Scientists

Part 2: Data Preprocessing

29.09.2021

Frauke Albrecht, Caroline Arnold, Jakob Lüttgau, Tobias Weigel

Helmholtz AI Consultants for Earth and Environment

DEUTSCHES KLIMARECHENZENTRUM

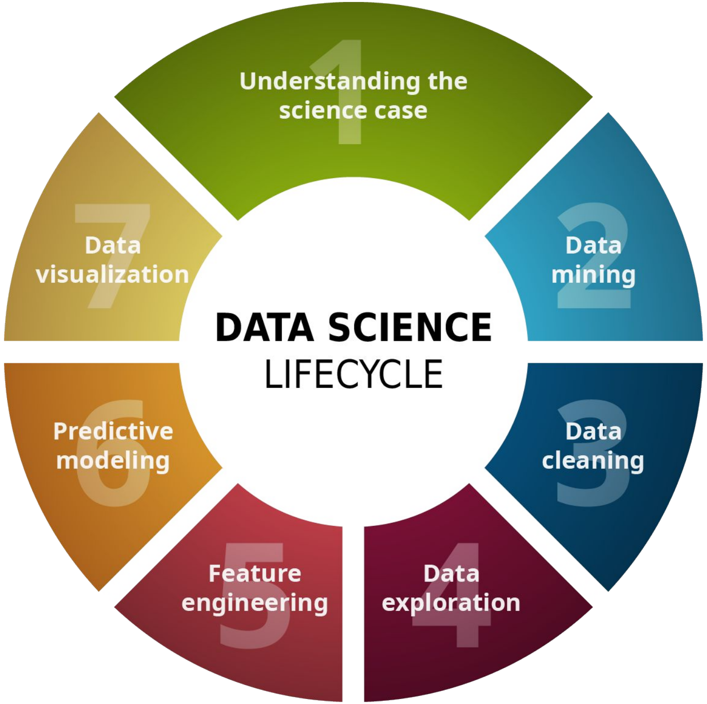
HELMHOLTZ AI

Logistics

- Virtual room
  - Please share your cameras :-)
  - Please post questions in the shared google doc
  - Breakouts for the tutorial part in the afternoon
- Timeline for today
  - 10:00-12:00 - Lecture
  - 13:00-15:15 - Tutorial with jupyterlab

Motivation

- Data is the most important resource in machine learning
- Scientific cases often use specialised data sets → specialised procedures
- Specialized hardware
- Scope today:
  - Everything that happens between raw data collection and the first ML training loop
  - Performance monitoring and improvement



From raw data to training data

Remote Sensing of Ocean Wind Speed

- 90% of observations cannot be used
- 60% of available features are not used

→ Filter the good samples and keep the interesting features

Atmospheric Chemistry

- Very large number of small samples
- Cannot be shuffled in-memory

→ Shuffle the samples beforehand and save them in an accessible file format

Satellite Image Super-Resolution

- Find and remove cloudy scenes
- Annotate samples (expensive calculation)

→ Annotate the filtered samples

Preprocessing

- Clearly separated from ML training
- Executed once - before ML optimization
- Save new train / validation / test

Modules in ML software

- Organize your code in a standardized way

```
cygnss_202003
|--- preprocessing
|------- preprocessing.py
|------- analysis.py
|--- training
|-------- Model.py
|--- utils
|------- mathematics.py
```

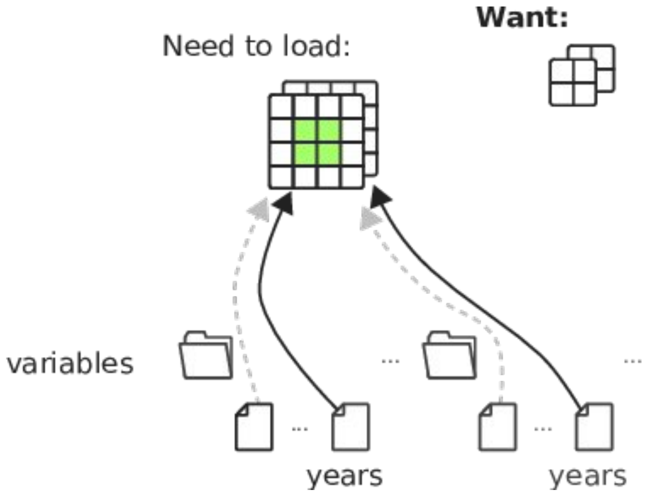
- ❖ Helps with documentation and readability
- ❖ Helps with debugging
- ❖ Separate tasks in separate programs - if you add new data, know which parts of the code you need to touch
- ❖ We recommend a **preprocessing** module in any ML project

```
from cygnss_202003.preprocessing import preprocessing as prp

prp.open_dataset(...)
```

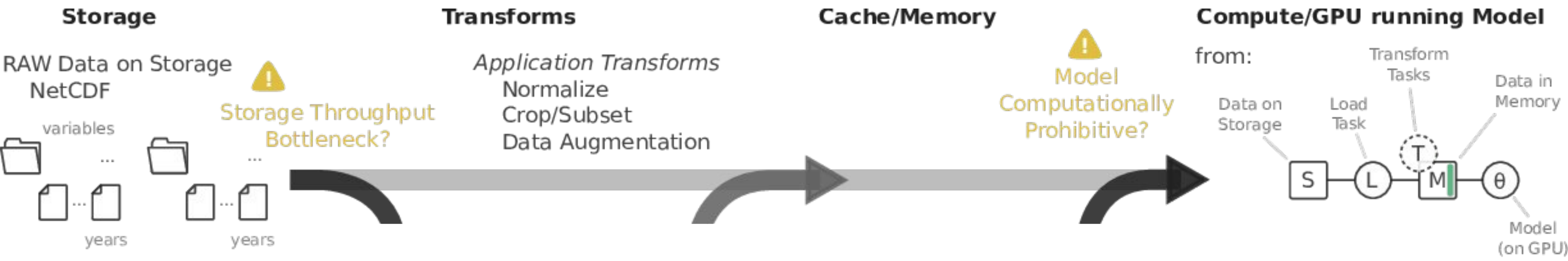
A First Overview

A workflow perspective

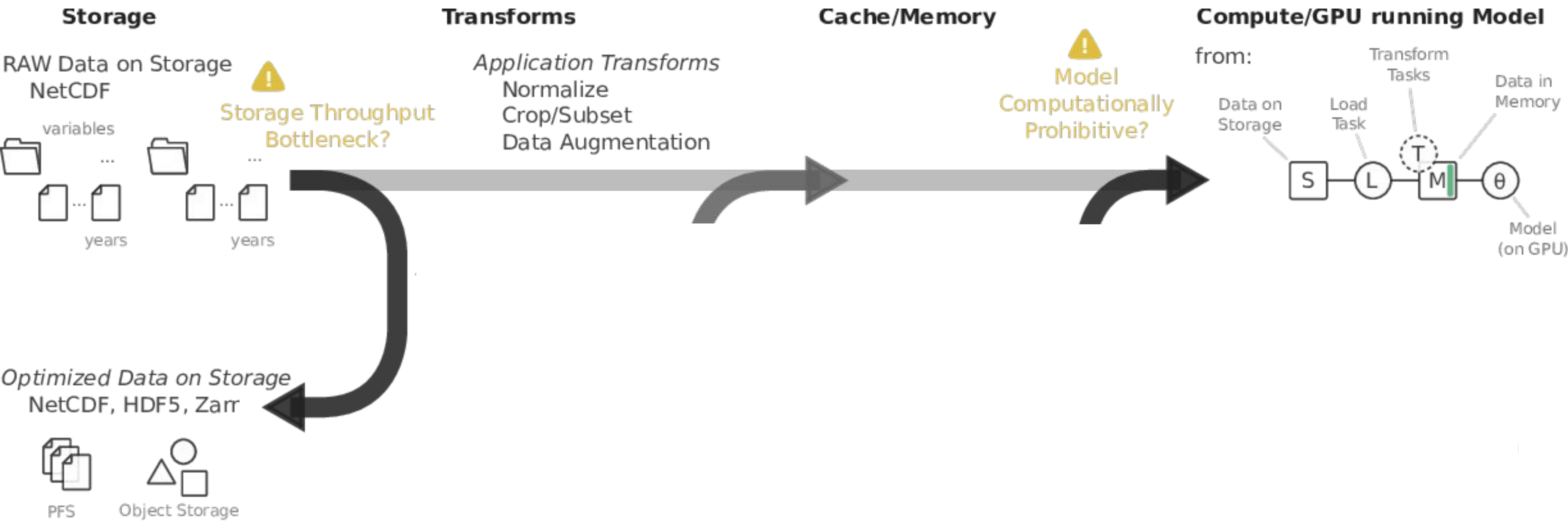
The Basic Problem for many Pre-Processing Tasks



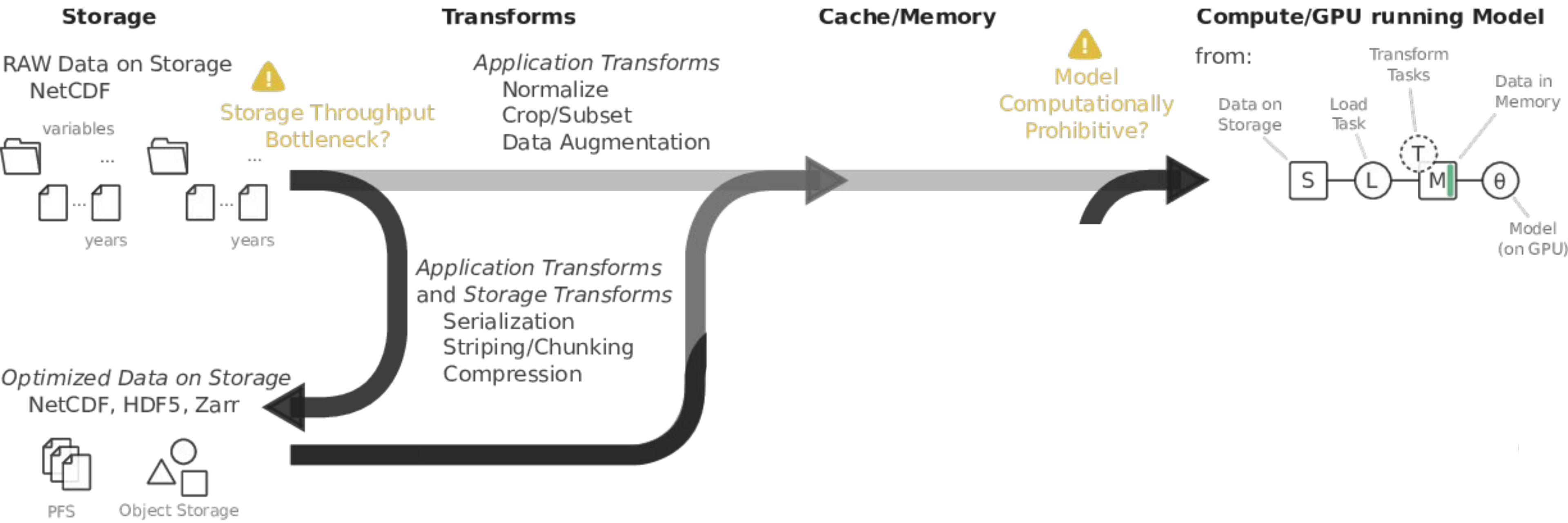
A look at your end-to-end ML Pre-Processing Workflow:



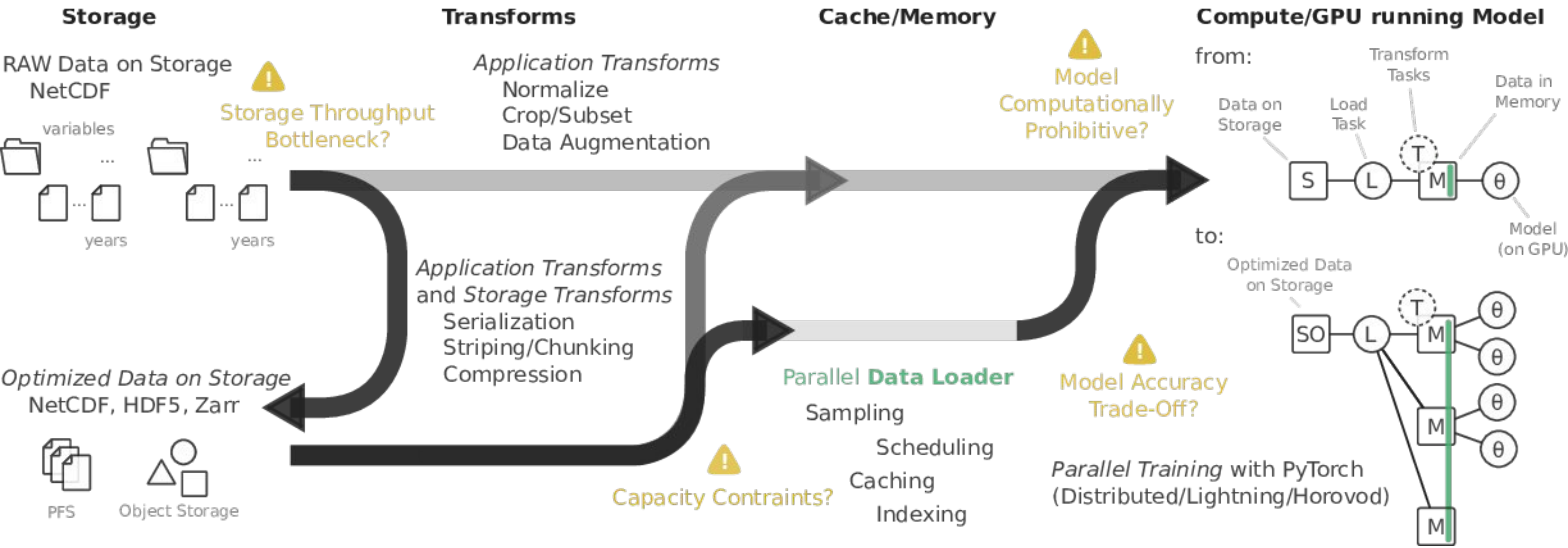
A look at your end-to-end ML Pre-Processing Workflow:



A look at your end-to-end ML Pre-Processing Workflow:



A look at your end-to-end ML Pre-Processing Workflow:



HPC Systems: A look behind the curtain

Memory Hierarchy, Cluster Computers, Storage Systems

What happens when you load your data?

```
with open(“/path/to/data”, “r”) as f:
    data = f.read()
```

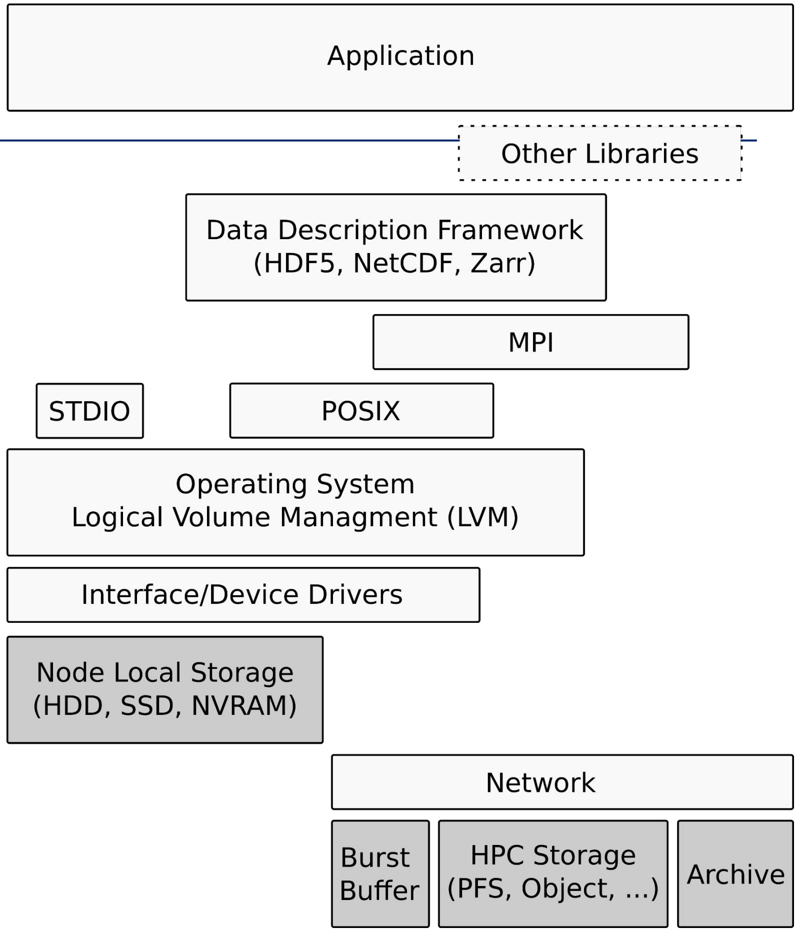
```
import pandas as pd
df = pd.read_csv(“/path/to/data.csv”)
```

```
import h5py
f = h5py.File('dataset.hdf5', 'r')
```

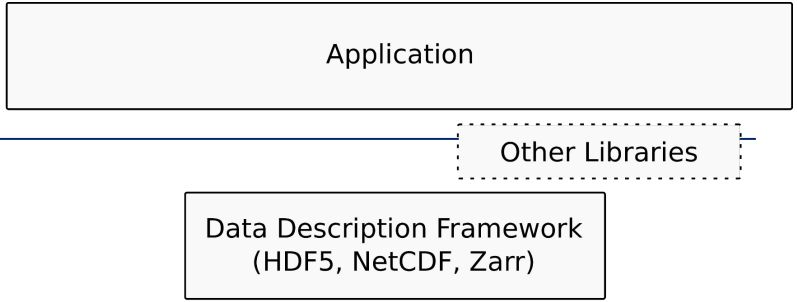
```
from netCDF4 import Dataset
rootgrp = Dataset("/path/to/dataset.nc", "r", format="NETCDF4")
```

```
import xarray as xr
ds = xr.open_dataset("/path/to/dataset.nc")
```

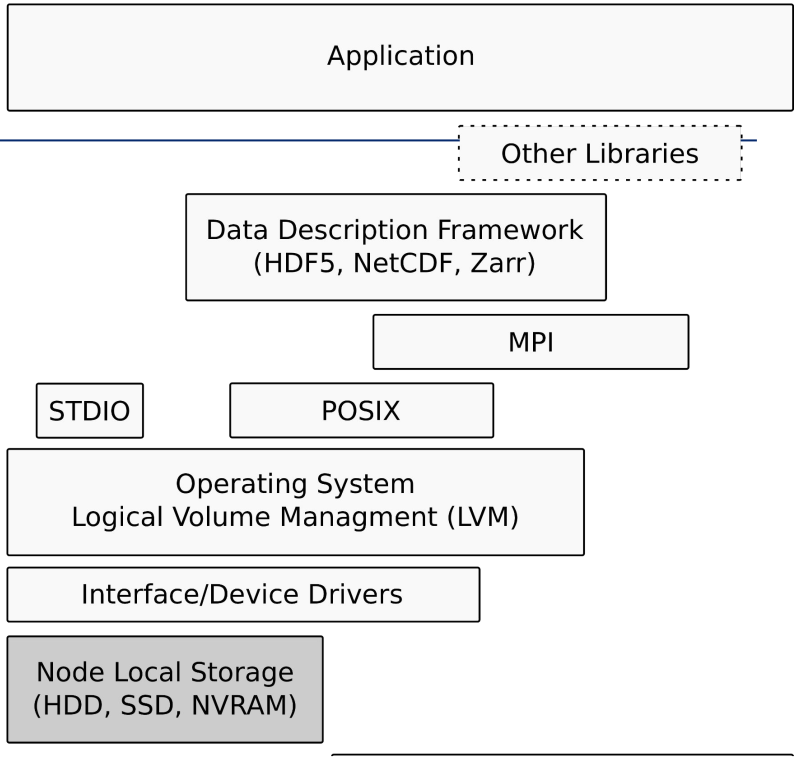
System: Software Stack



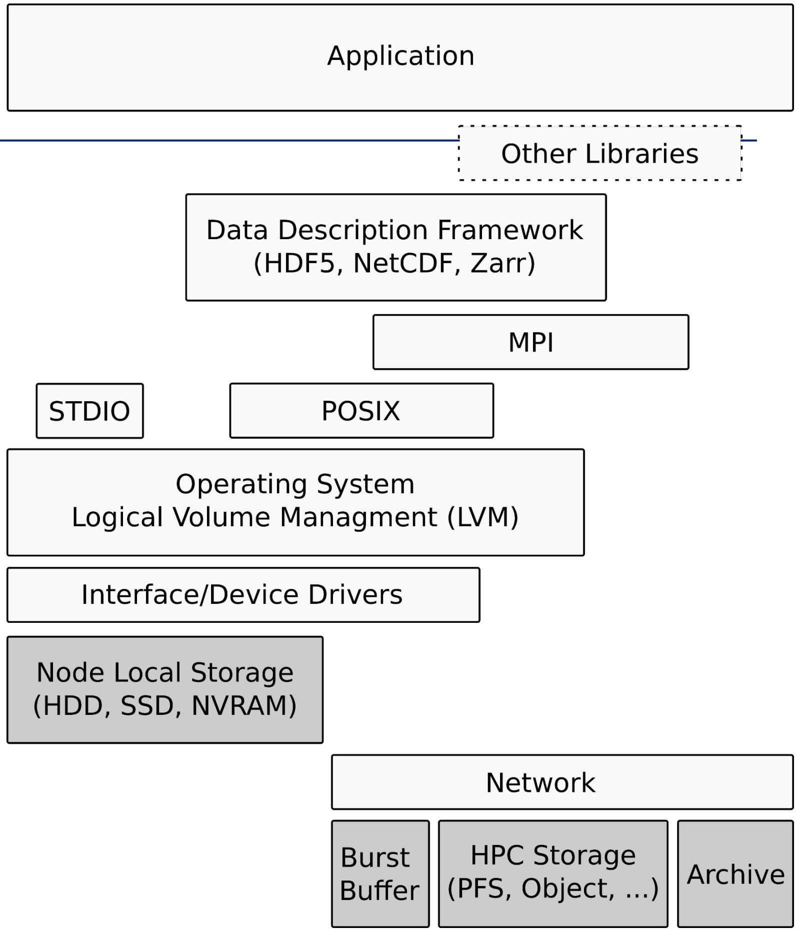
System: Software Stack



System: Software Stack

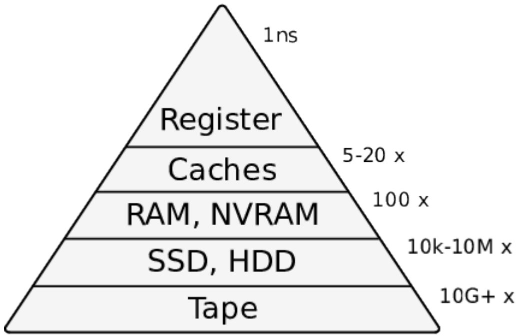


System: Software Stack



Memory Hierarchy

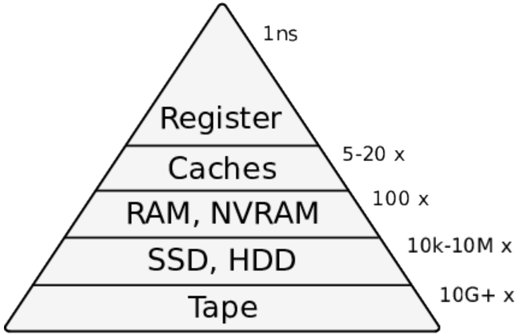
high cost, low latency



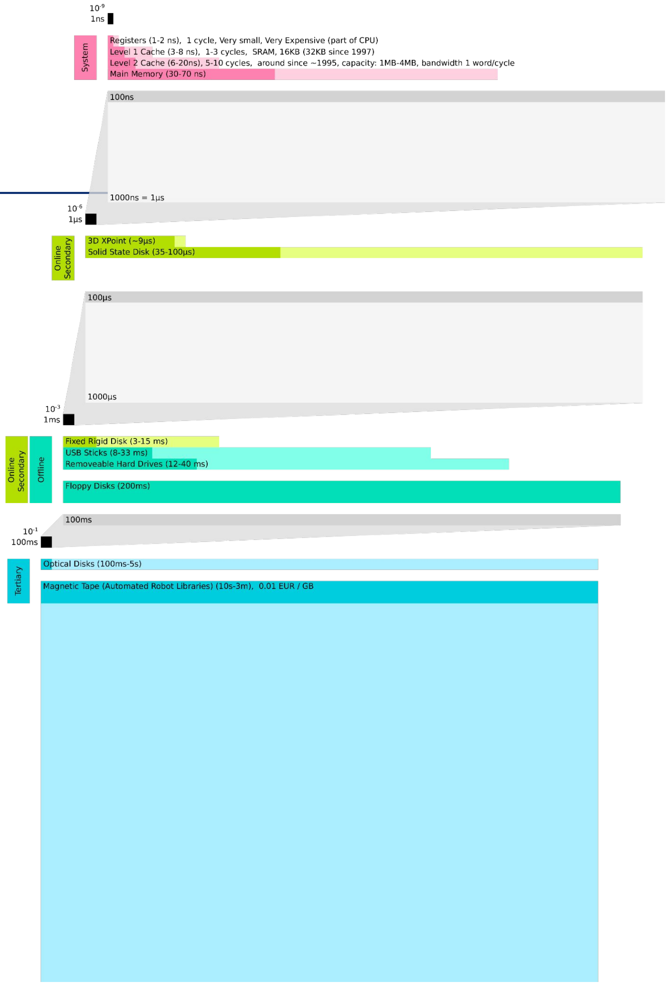
low cost, high capacity

Memory Hierarchy

high cost, low latency



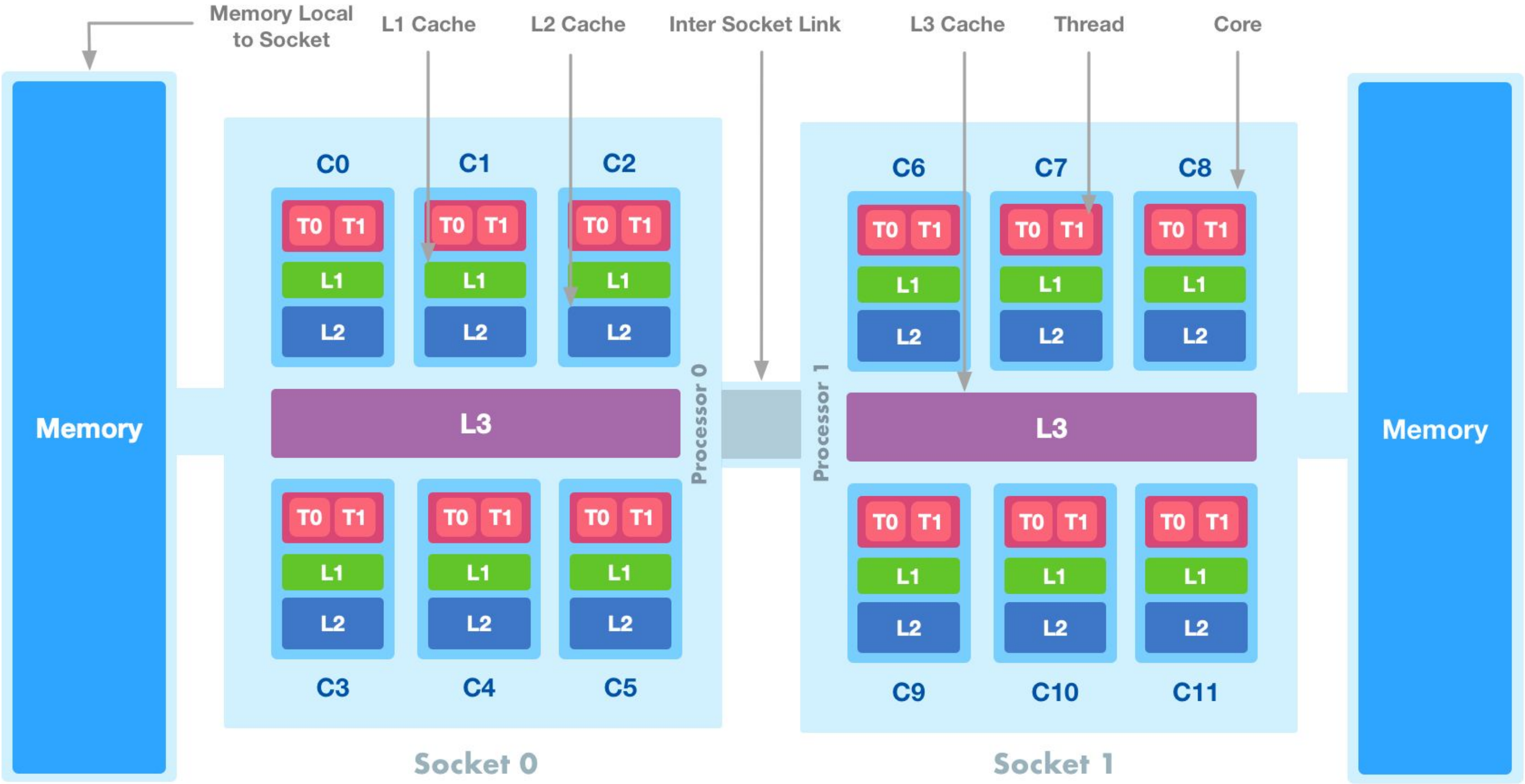
low cost, high capacity



The best case: Your data fits in memory

You won’t need to care about the storage complexity after you have loaded you data. But data movements within a single compute node may still hold you back.

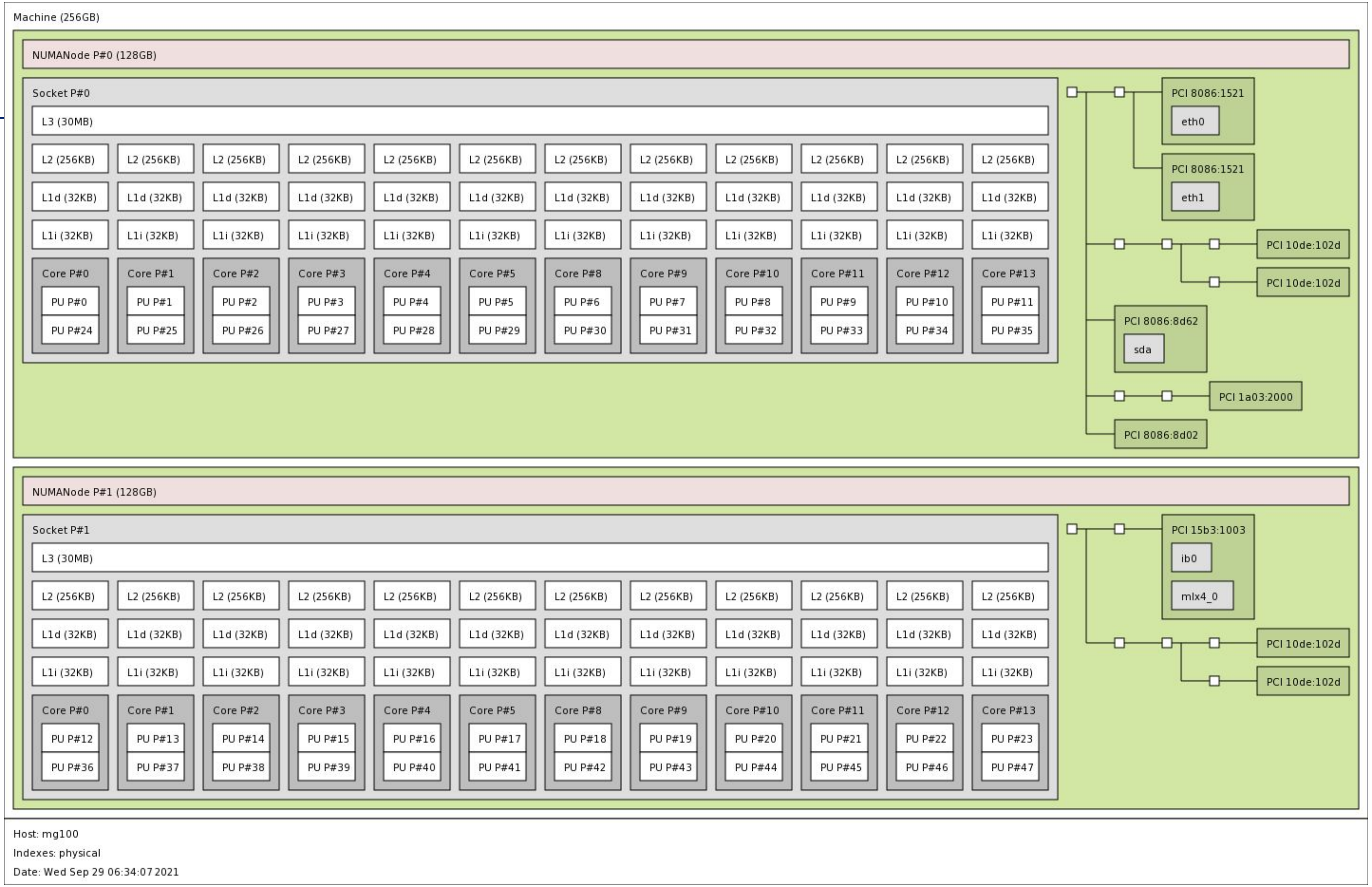
Let’s have a look at a typical compute node with GPUs to accelerate training.



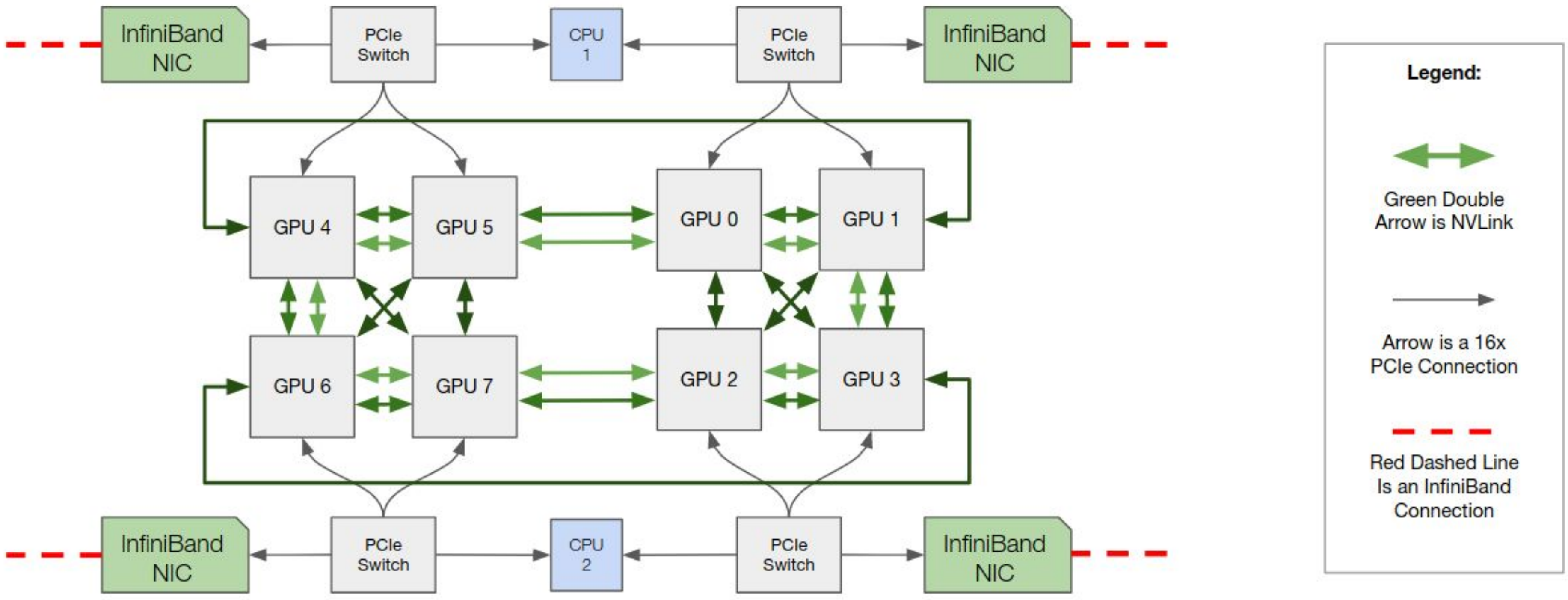
An Example of Compute Node Architecture

http://pramodkumbhar.com/2020/03/architectural-optimisations-using-likwid-profiler/

HPC Nodes


Machine (256GB)
NUMANode P#0 (128GB)
Socket P#0
L3 (30MB)
L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)
L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)
L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)
Core P#0 PU P#0 PU P#24
Core P#1 PU P#1 PU P#25
Core P#2 PU P#2 PU P#26
Core P#3 PU P#3 PU P#27
Core P#4 PU P#4 PU P#28
Core P#5 PU P#5 PU P#29
Core P#8 PU P#6 PU P#30
Core P#9 PU P#7 PU P#31
Core P#10 PU P#8 PU P#32
Core P#11 PU P#9 PU P#33
Core P#12 PU P#10 PU P#34
Core P#13 PU P#11 PU P#35
PCI 8086:1521 eth0
PCI 8086:1521 eth1
PCI 10de:102d
PCI 10de:102d
PCI 8086:8d62 sda
PCI 1a03:2000
PCI 8086:8d02
NUMANode P#1 (128GB)
Socket P#1
L3 (30MB)
L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)
L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)
L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)
Core P#0 PU P#12 PU P#36
Core P#1 PU P#13 PU P#37
Core P#2 PU P#14 PU P#38
Core P#3 PU P#15 PU P#39
Core P#4 PU P#16 PU P#40
Core P#5 PU P#17 PU P#41
Core P#8 PU P#18 PU P#42
Core P#9 PU P#19 PU P#43
Core P#10 PU P#20 PU P#44
Core P#11 PU P#21 PU P#45
Core P#12 PU P#22 PU P#46
Core P#13 PU P#23 PU P#47
PCI 15b3:1003 ib0 mlx4_0
PCI 10de:102d
PCI 10de:102d
Host: mg100
Indexes: physical
Date: Wed Sep 29 06:34:07 2021

A top of the line GPU nod: ~42 GB/s in/out of node
600 GB/s between GPUs within node

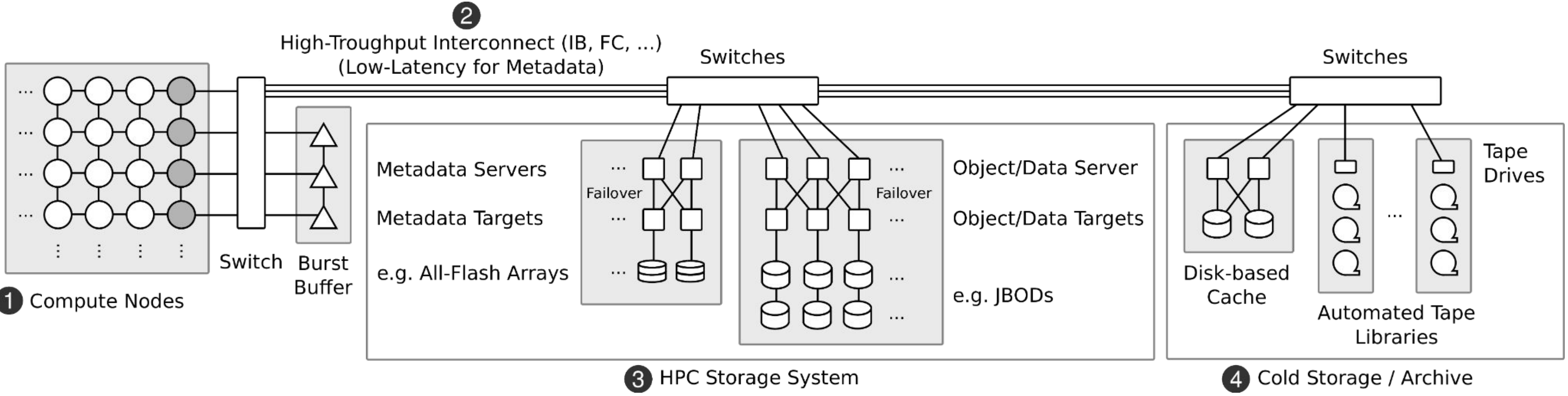


https://lambdalabs.com/deep-learning/servers/hyperplane-a100

HPC Storage?

```
[luettgau1@jwlogin06 ~]$ df -h
Filesystem                 Size  Used Avail Use% Mounted on
devtmpfs                   378G     0  378G   0% /dev
tmpfs                      378G  195M  377G   1% /dev/shm
tmpfs                      378G  4.1G  374G   2% /run
tmpfs                      378G     0  378G   0% /sys/fs/cgroup
/dev/mapper/centos-home     29G  4.8G   23G  18% /
/dev/md0                   990M  137M  787M  15% /boot
/dev/sdd2                  200M     0  200M   0% /boot/efi_disk2
/dev/sdc1                  200M  6.9M  193M   4% /boot/efi
/dev/mapper/centos-tmp      20G   59M   19G   1% /tmp
/dev/mapper/centos-var      48G  2.0G   44G   5% /var
arch                       1.1P  395T  630T  39% /p/arch
fastdata                   8.6P  7.2P  1.4P  85% /p/fastdata
project                    4.3P  2.5P  1.9P  57% /p/project
usersoftware                33T  423G   32T   2% /p/usersoftware
home                        61T   14T   48T  22% /p/home
largedata_restore          4.9P  4.5P  445T  92% /p/largedata_restore
juwels_software             17T  4.4T   12T  27% /p/software/juwels
largedata2                 8.0P  1.5P  6.5P  19% /p/largedata2
arch2                     1021T  388T  634T  38% /p/arch2
largedata                   27P   22P  4.9P  82% /p/largedata
scratch                     13P  6.8P  6.0P  54% /p/scratch
```

System: Hardware Perspective



Node Topology as exposed by Slurm:

Added I/O telemetry from log data:

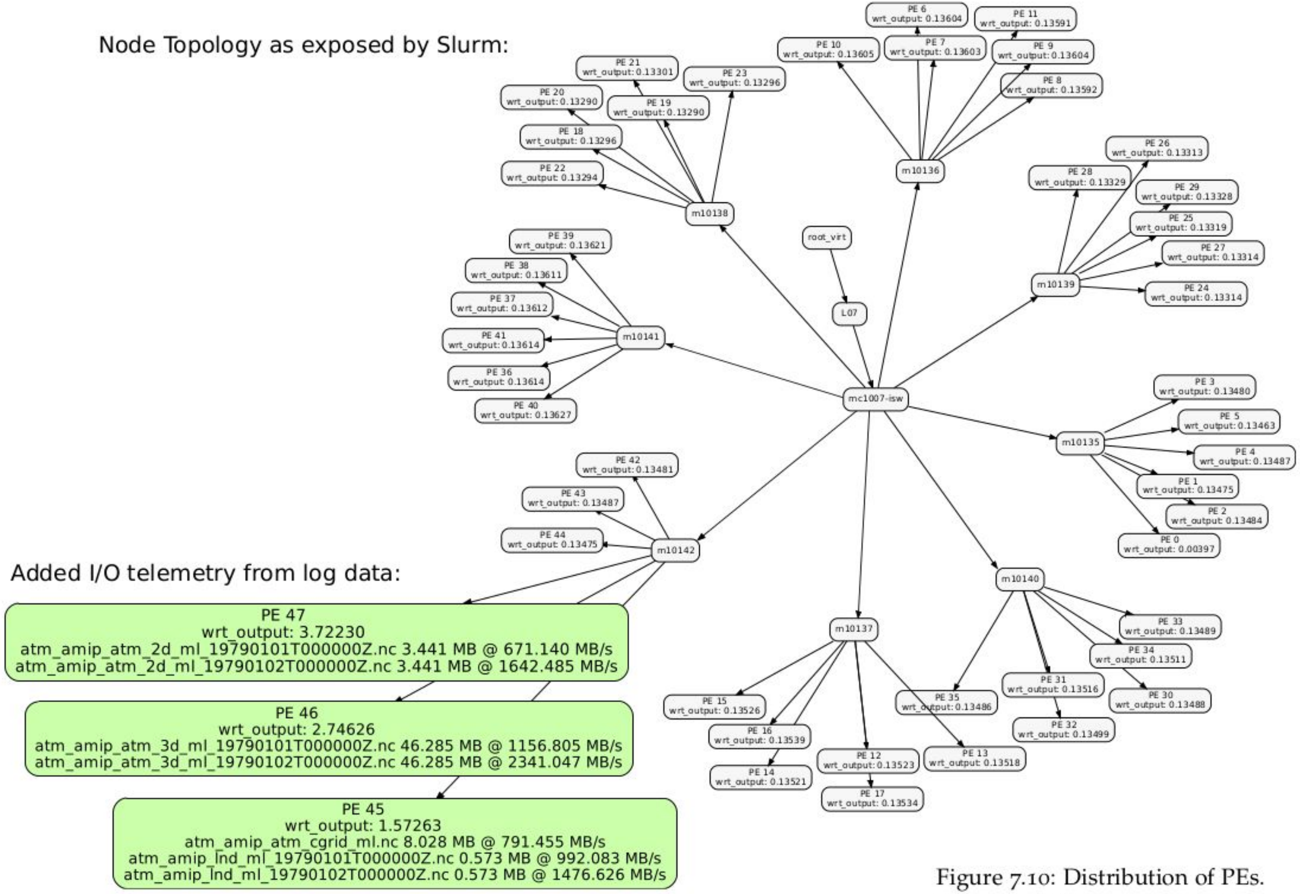
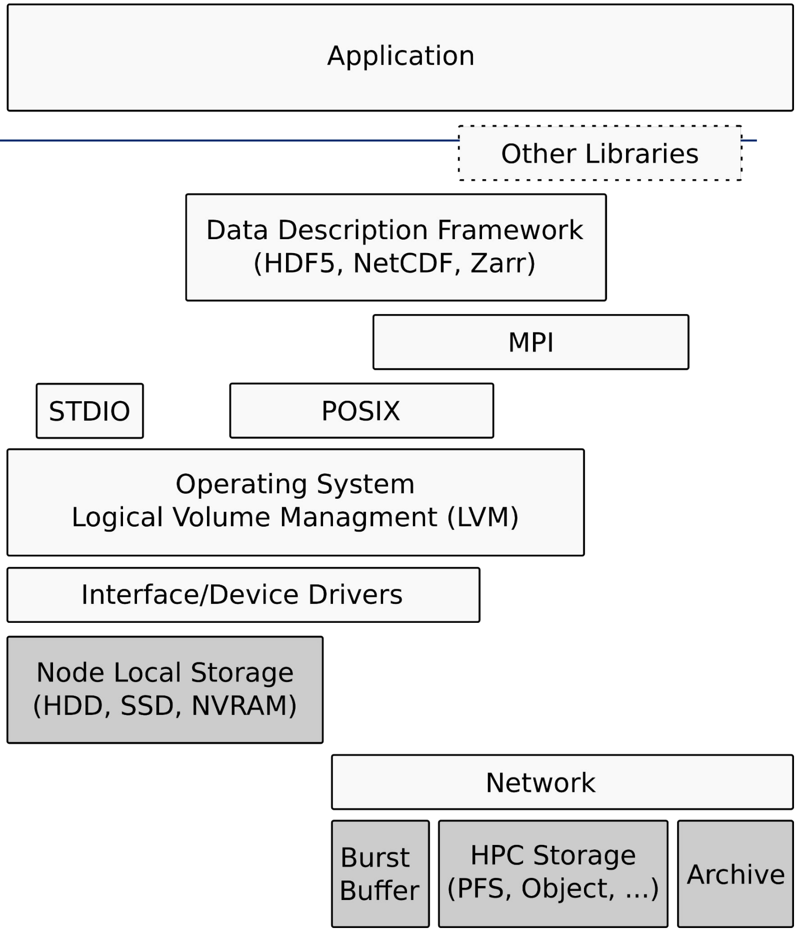


Figure 7.10: Distribution of PEs. Note the I/O PEs responsible for writing to the parallel file system.

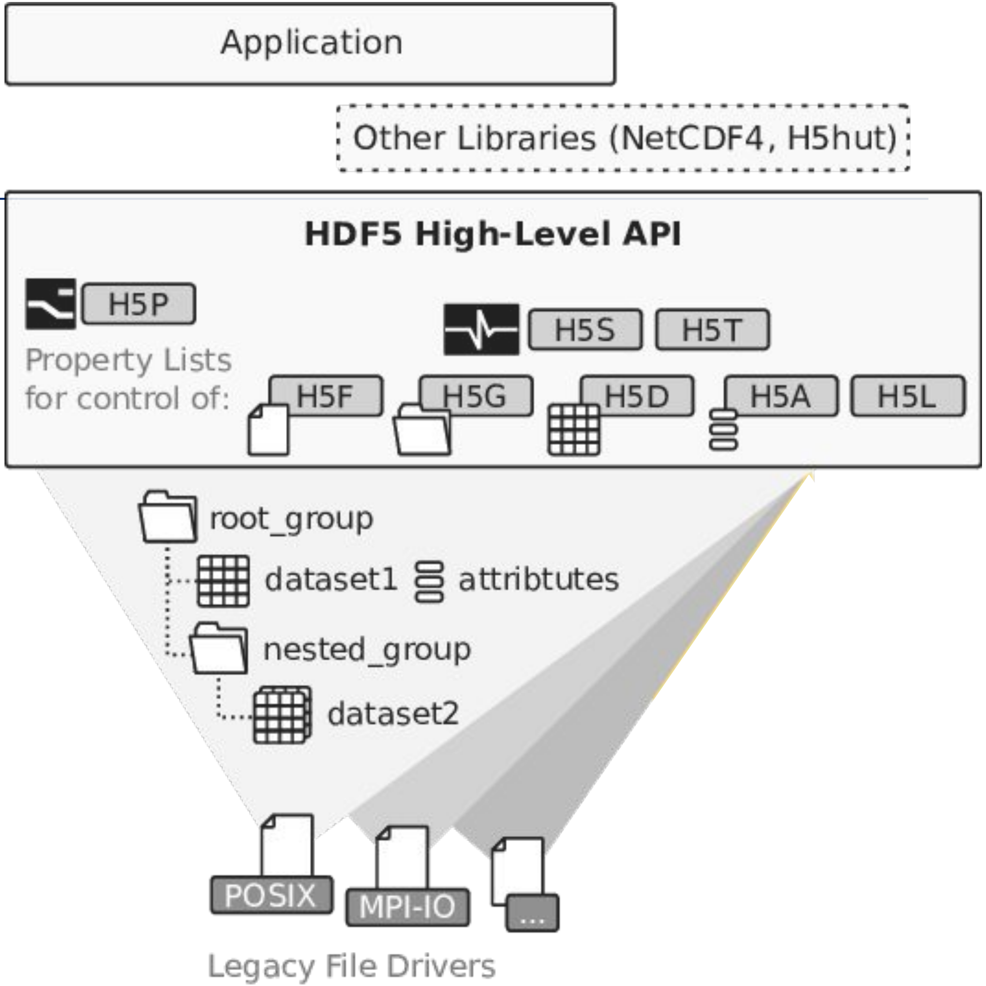
System: Software Stack



Self-Describing Data Formats

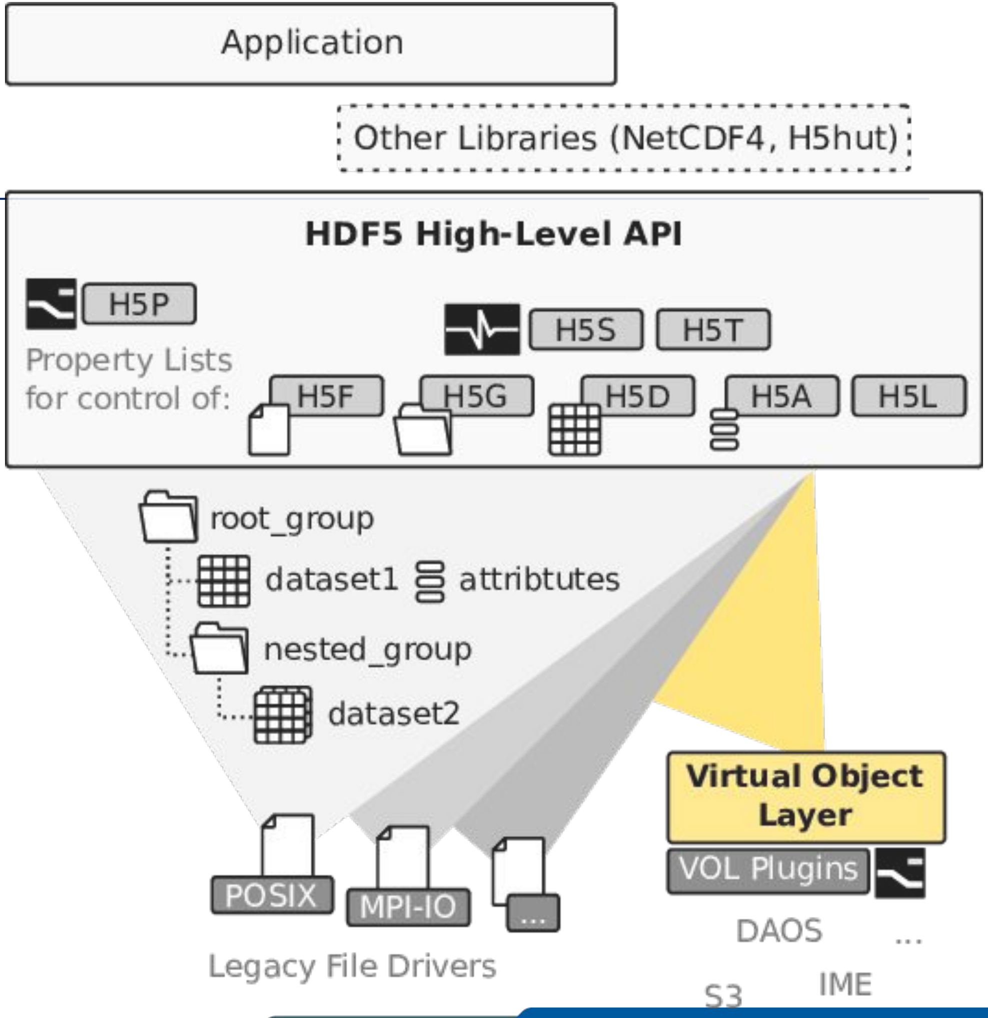
HDF5, NetCDF, Zarr, ...

HDF5



Application
Other Libraries (NetCDF4, H5hut)
HDF5 High-Level API
H5P
Property Lists
for control of:
H5S
H5T
H5F
H5G
H5D
H5A
H5L
root_group
dataset1
attribtutes
nested_group
dataset2
POSIX
MPI-IO
...
Legacy File Drivers

HDF5


Application
Other Libraries (NetCDF4, H5hut)
HDF5 High-Level API
H5P
Property Lists
for control of:
H5S
H5T
H5F
H5G
H5D
H5A
H5L
root_group
dataset1
attribtutes
nested_group
dataset2
POSIX
MPI-IO
...
Legacy File Drivers
Virtual Object Layer
VOL Plugins
DAOS
...
S3
IME

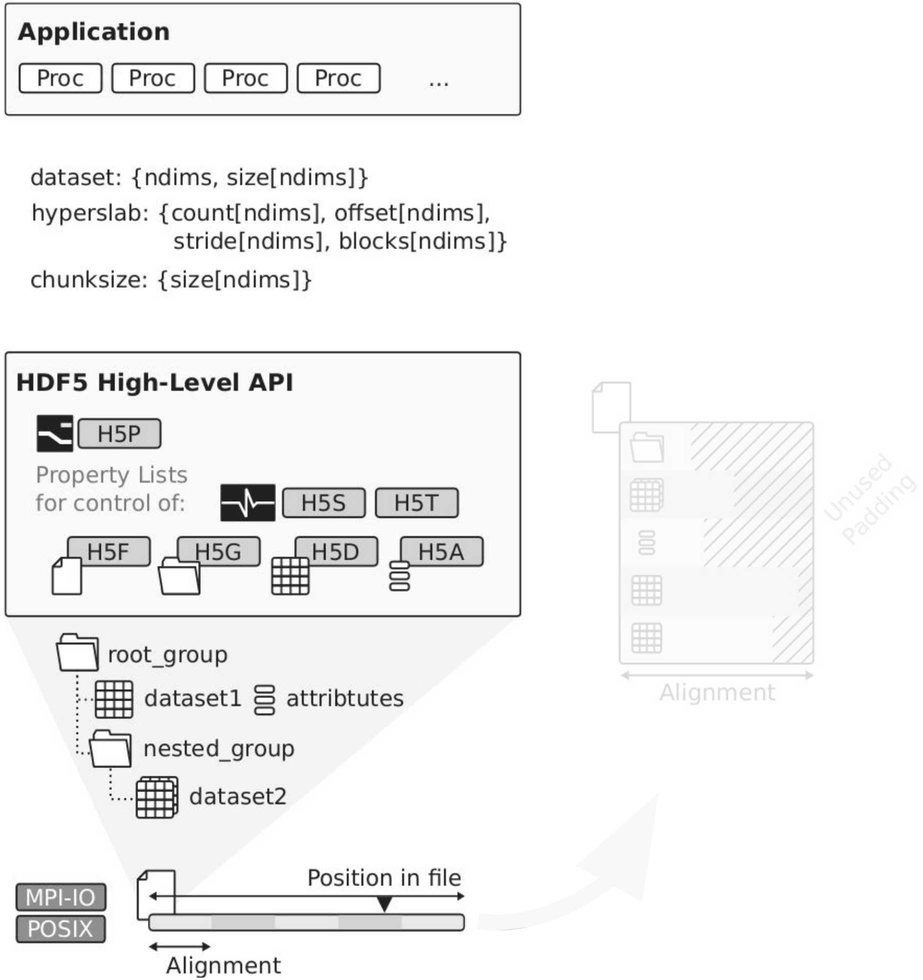
Application

Proc | Proc | Proc | Proc | ...

dataset: {ndims, size[ndims]}

hyperslab: {count[ndims], offset[ndims],
stride[ndims], blocks[ndims]}

chunksize: {size[ndims]}

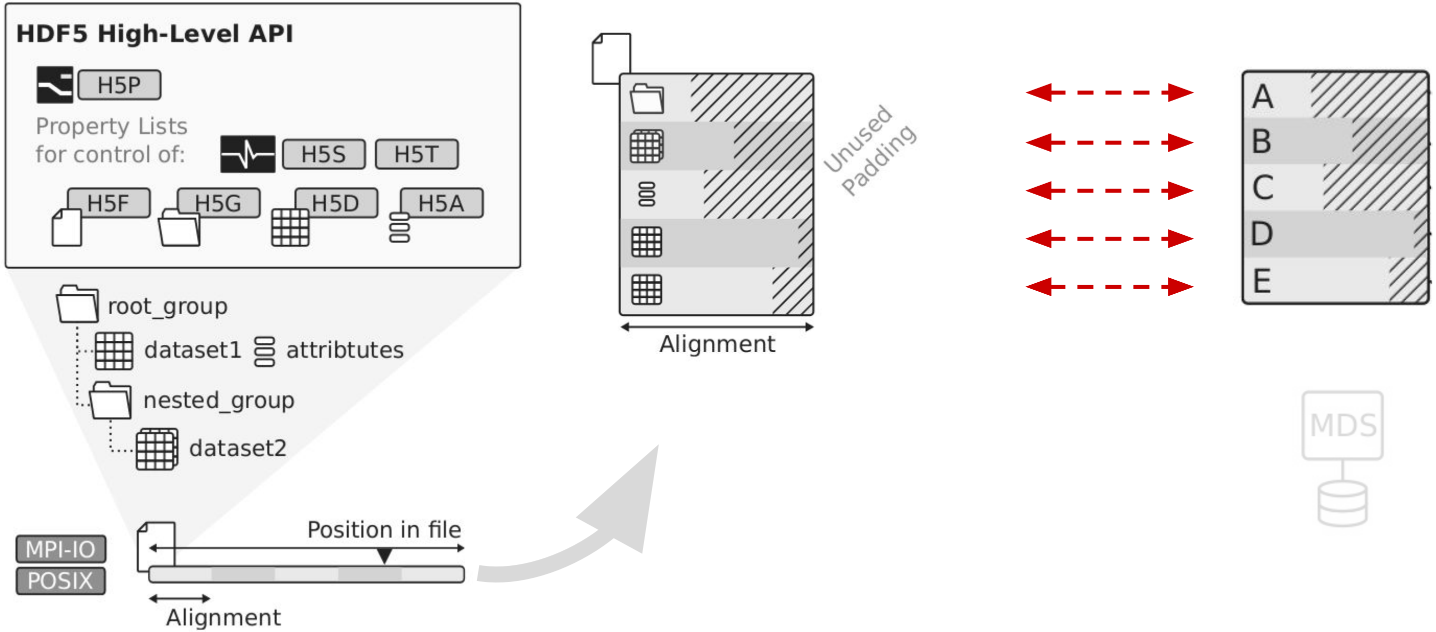


Storage Optimization from Application, over HDF5, to file I/O targeting Lustre.

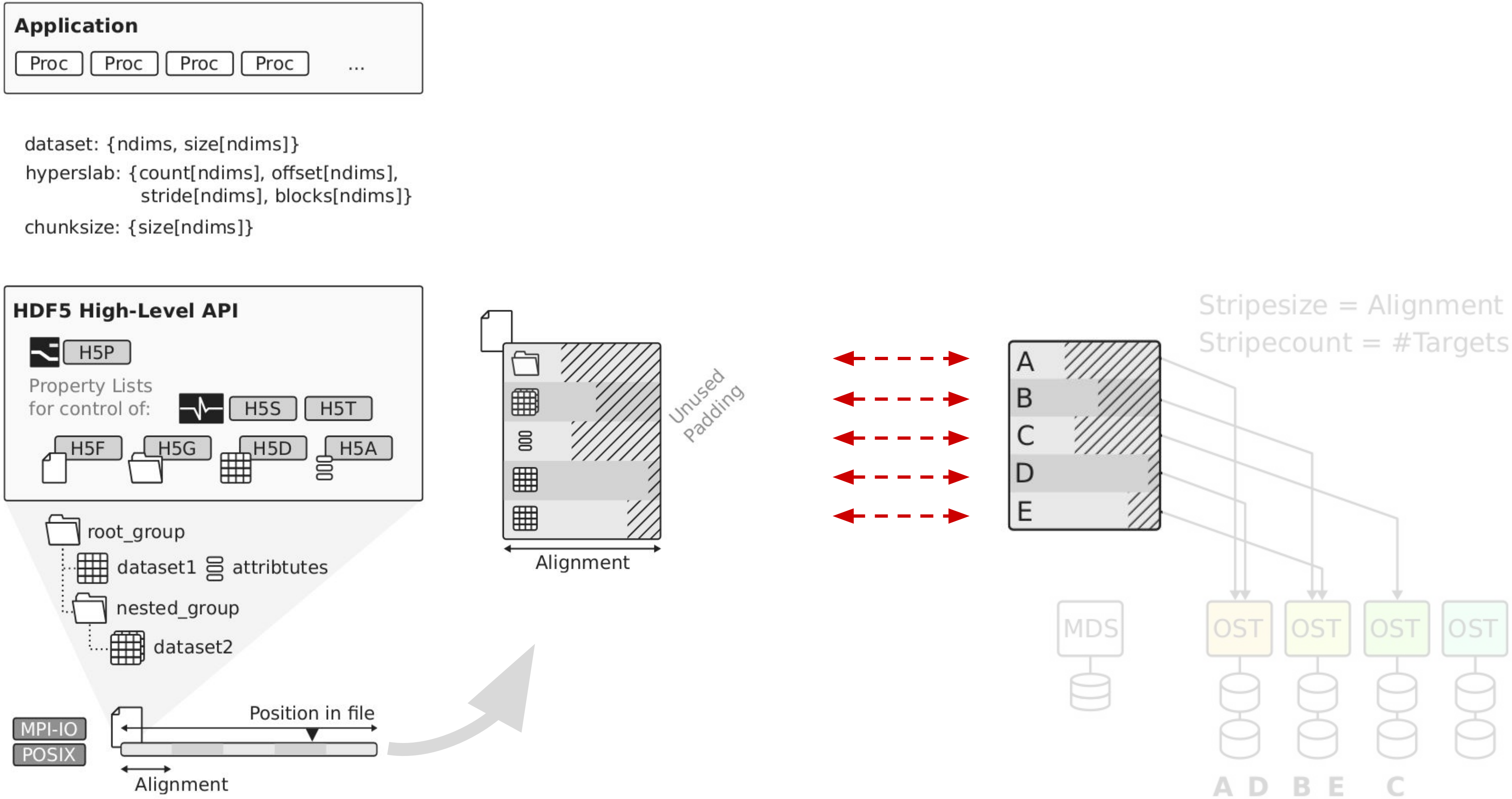
dataset: {ndims, size[ndims]}

hyperslab: {count[ndims], offset[ndims], stride[ndims], blocks[ndims]}

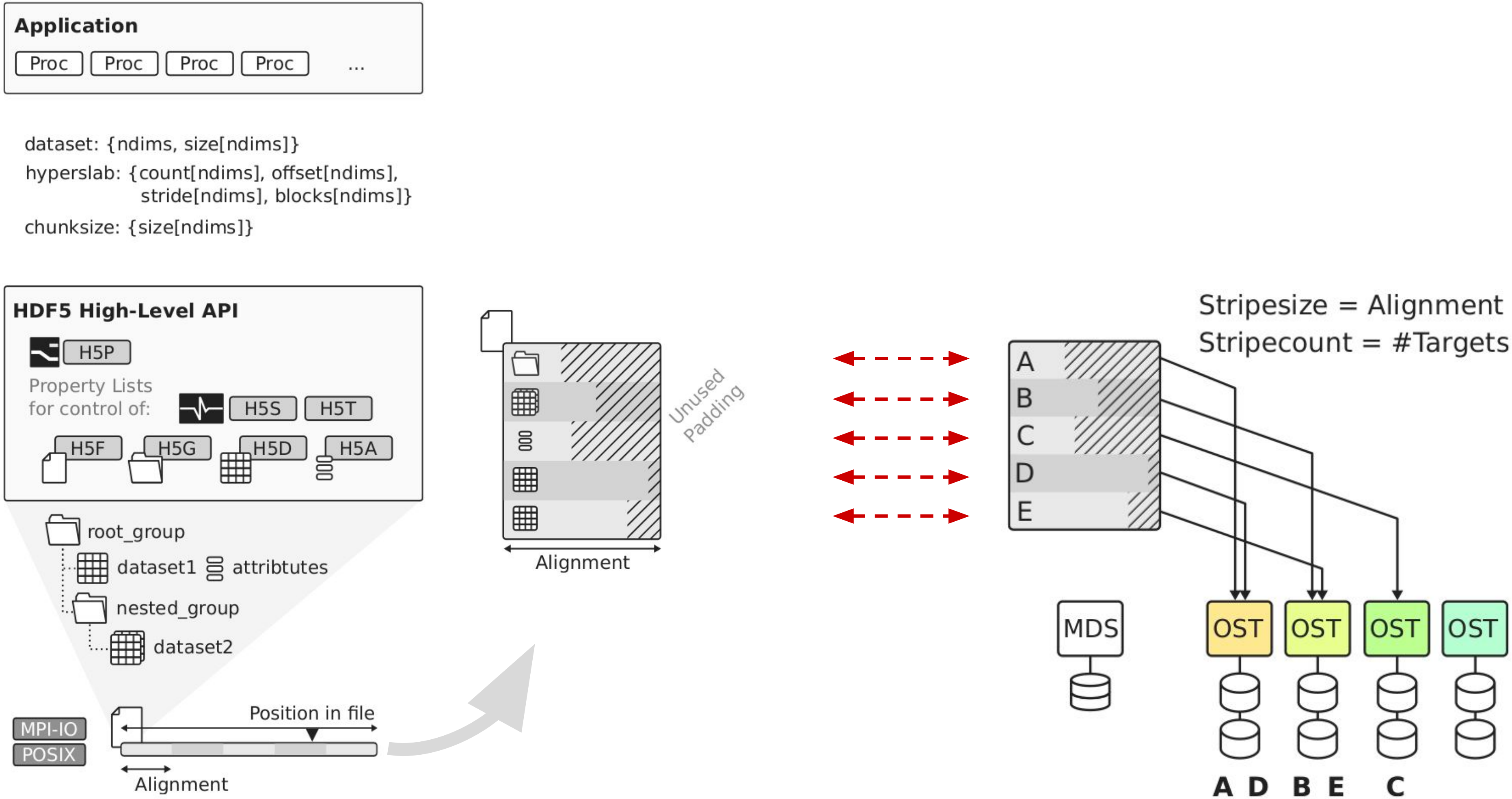
chunksize: {size[ndims]}



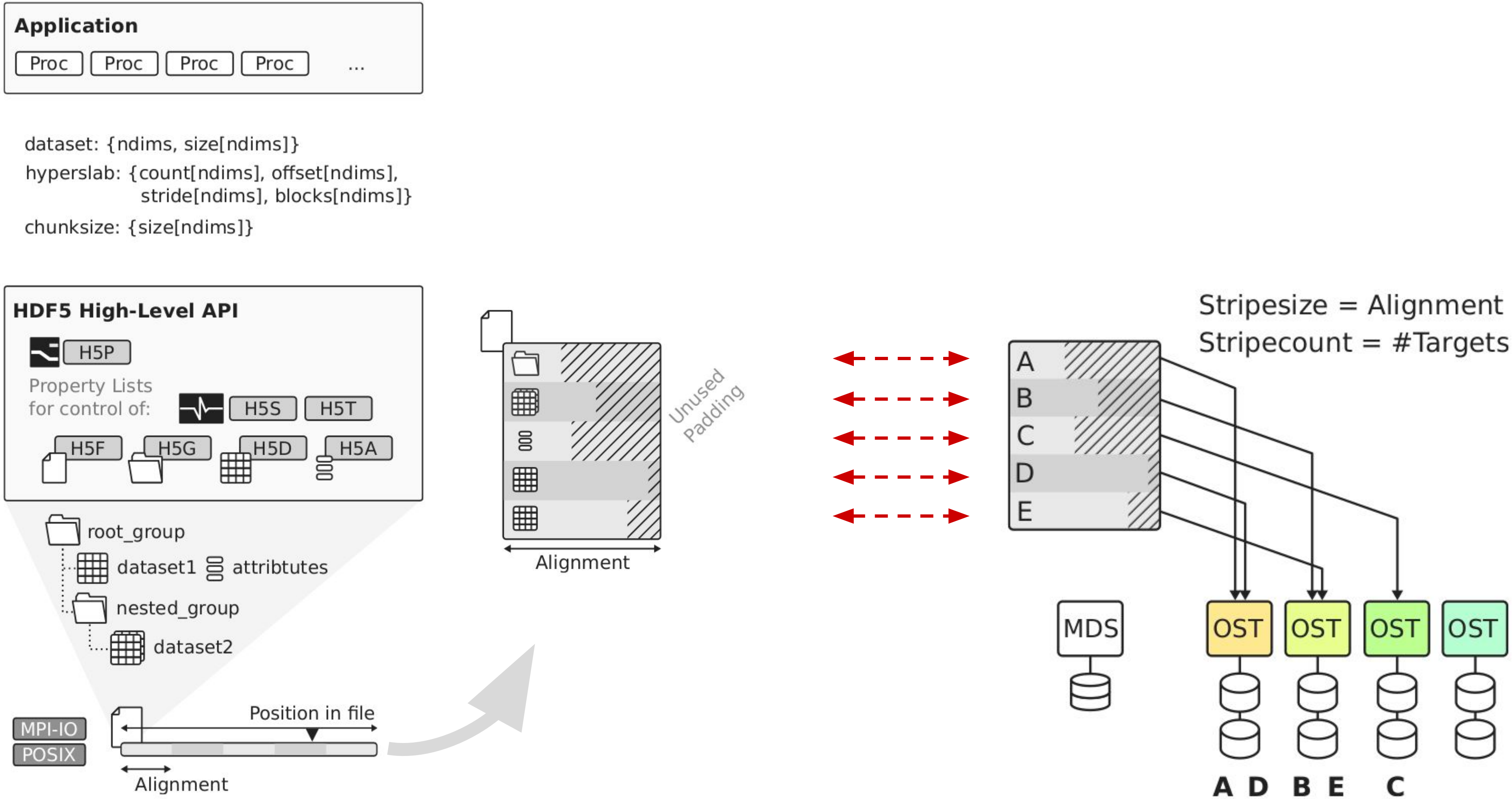
Storage Optimization from Application, over HDF5, to file I/O targeting Lustre



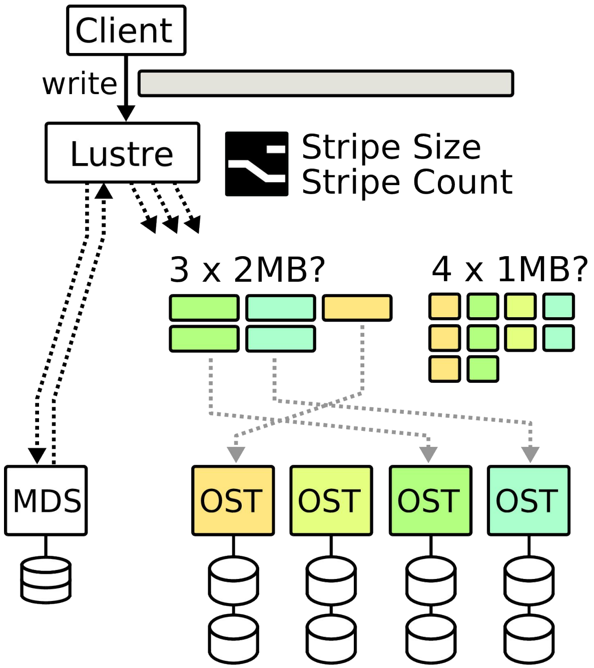
Storage Optimization from Application, over HDF5, to file I/O targeting Lustre.



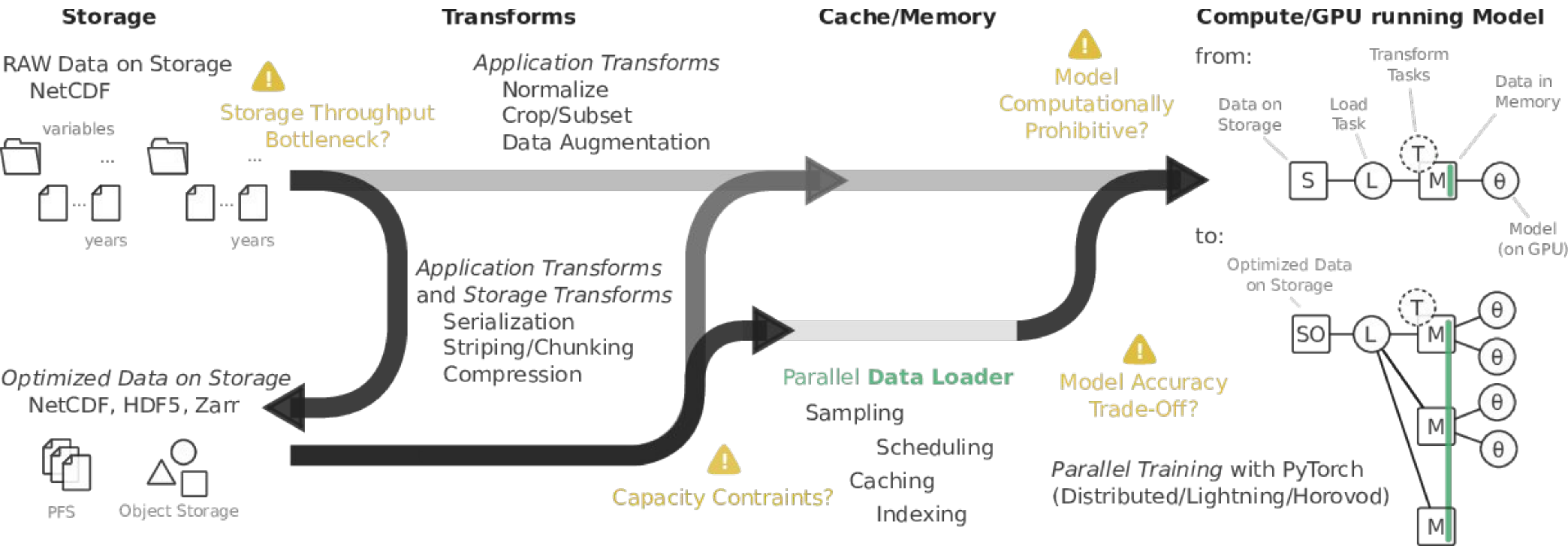
Storage Optimization from Application, over HDF5, to file I/O targeting Lustre.



Storage Optimization from Application, over HDF5, to file I/O targeting Lustre.


Client
write
Lustre
Stripe Size
Stripe Count
3 x 2MB?
4 x 1MB?
MDS
OST
OST
OST
OST

The Pre-Processing Workflow Revisited



BREAK (10 min)

Convenient Interface to Gridded Data in Python

xarray

xarray

- Scientific data is often labeled
- Python formats
  - numpy: efficient but hard to interpret
  - Python pandas: self-explained but not performant

- xarray provides labeled arrays & datasets
- Very useful for working with netCDF
- Integrates with dask for parallel computing (which we do not cover today)
-

```
[4]: url = "https://opendata.arcgis.com/datasets/dd4580c810204019a7b8eb3e0b329dd6_0.csv"
     %time data = pd.read_csv(url)

     Wall time: 57 s

[13]: data
```

[13]:

	ObjectId	IdBundesland	Bundesland	Landkreis	Altersgruppe	Geschlecht	AnzahlFall	AnzahlTodesfall	Meldedatum
0	1	1	Schleswig-Holstein	SK Flensburg	A00-A04	M	1	0	2020/09/30 00:00:00+00
1	2	1	Schleswig-Holstein	SK Flensburg	A00-A04	M	1	0	2020/10/29 00:00:00+00
2	3	1	Schleswig-Holstein	SK Flensburg	A00-A04	M	1	0	2020/11/03 00:00:00+00
3	4	1	Schleswig-Holstein	SK Flensburg	A00-A04	M	1	0	2020/11/20 00:00:00+00
4	5	1	Schleswig-Holstein	SK Flensburg	A00-A04	M	1	0	2020/11/23 00:00:00+00
...	...	...	...	...	...	...	...	...	...
2400782	2400783	16	Thüringen	LK Altenburger Land	A80+	W	1	0	2021/08/07 00:00:00+00
2400783	2400784	16	Thüringen	LK Altenburger Land	A80+	W	1	0	2021/08/24 00:00:00+00

```
[15]: data.values[:5]

[15]: array([[1, 1, 'Schleswig-Holstein', 'SK Flensburg', 'A00-A04', 'M', 1, 0,
             '2020/09/30 00:00:00+00', 1001, '28.09.2021, 00:00 Uhr', 0, -9,
             '2020/09/30 00:00:00+00', 0, 1, 0, 'Nicht übermittelt'],
            [2, 1, 'Schleswig-Holstein', 'SK Flensburg', 'A00-A04', 'M', 1, 0,
             '2020/10/29 00:00:00+00', 1001, '28.09.2021, 00:00 Uhr', 0, -9,
             '2020/10/29 00:00:00+00', 0, 1, 0, 'Nicht übermittelt'],
            [3, 1, 'Schleswig-Holstein', 'SK Flensburg', 'A00-A04', 'M', 1, 0,
             '2020/11/03 00:00:00+00', 1001, '28.09.2021, 00:00 Uhr', 0, -9,
             '2020/11/03 00:00:00+00', 0, 1, 0, 'Nicht übermittelt'],
            [4, 1, 'Schleswig-Holstein', 'SK Flensburg', 'A00-A04', 'M', 1, 0,
             '2020/11/20 00:00:00+00', 1001, '28.09.2021, 00:00 Uhr', 0, -9,
             '2020/11/19 00:00:00+00', 0, 1, 1, 'Nicht übermittelt'],
            [5, 1, 'Schleswig-Holstein', 'SK Flensburg', 'A00-A04', 'M', 1, 0,
             '2020/11/23 00:00:00+00', 1001, '28.09.2021, 00:00 Uhr', 0, -9,
             '2020/11/18 00:00:00+00', 0, 1, 1, 'Nicht übermittelt']],
           dtype=object)
```

Access data with xarray

```
import xarray

ds = xarray.open_dataset(data_file)

ds
```

Labeled datasets

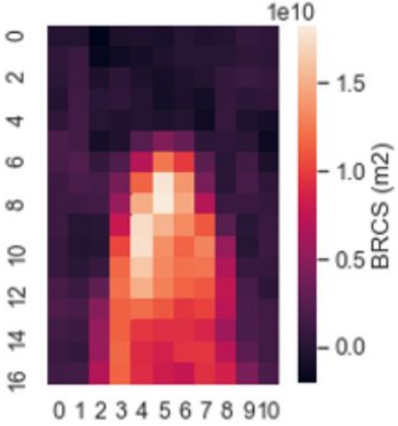
→ Panda-like access to variables

```
ds["windspeed"]

windspeed = ds.windspeed
```

xarray in Jupyterhub

- View the dataset
- Coordinates
- Dimensions
- Data variables: type (float32, int, bool, …), values
-

BRCS (17 x 11 image)




```
[36]: ds # View the dataset
```

```
[36]: <xarray.Dataset>
Dimensions:         (delay: 17, doppler: 11, sample: 24873)
Coordinates:
  * sample          (sample) int64 0 1 2 3 4 5 ... 24868 24869 24870 24871 24872
Dimensions without coordinates: delay, doppler
Data variables:
    windspeed       (sample) float32 ...
    ddm_timestamp   (sample) float32 ...
    brcs            (sample, delay, doppler) float32 -134927950.0 ... 756307260.0
```

Lazy loading

Lazy loading

- Data is only loaded in memory on request
- Computations etc. can be conducted without loading data
- Useful for large datasets!

Example: unit transformation

- Define the arithmetic operation
- Only when data is loaded, it will be exectued

```
windspeed = ds.windspeed

# this would not load the data
windspeed_kmh = 3.6 * windspeed

# this would load the data
windspeed_kmh.values
```

Data cleaning with xarray

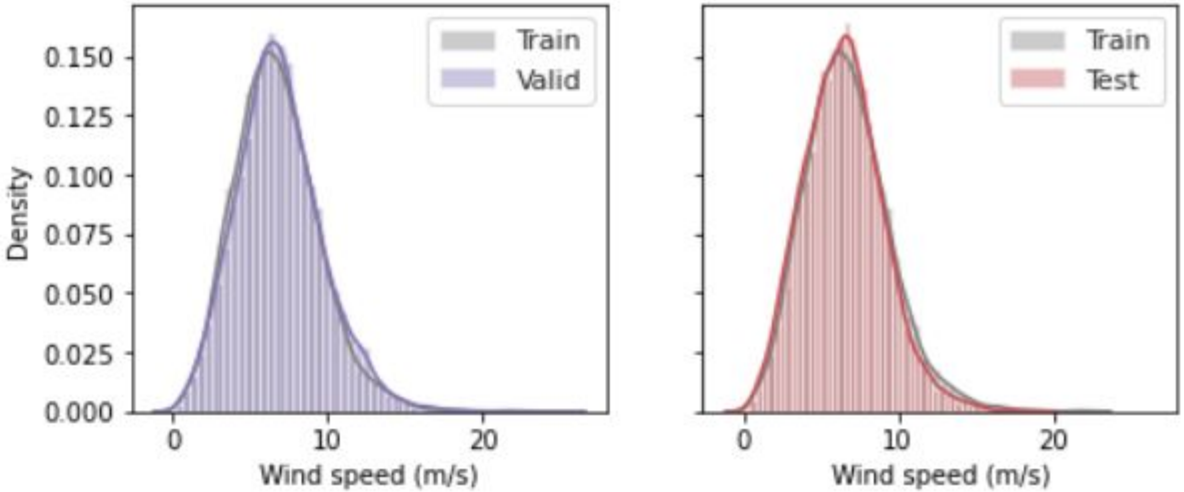
- Create a mask to select samples with None values
- Drop the samples

```
mask = xarray.ufuncs.isnan(ds_train.windspeed)
mask
```

```
ds_train = ds_train.sel(sample=~mask, drop=True)
ds_train
```

Train / Validation / Test Dataset

- Divide before improving on the ML algorithm
- Final evaluation is done on a test set - typically in distribution!
- Check the distribution of features and labels



PyTorch Dataset

- Load preprocessed data
- Dataset holds samples (features and labels)

```
from torch.utils.data import Dataset, DataLoader
```

- Object oriented programming
- Your dataset class is a subclass of `Dataset`
- In subclass, overwrite
  - `__len__`
  - `__getitem__`

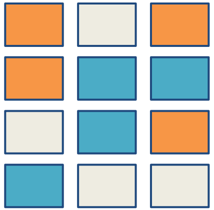
```
class CyGNSSDataset(Dataset):
    def __init__(self, flag):
        '''
        Load data from hdf5 file

        Parameters:
        -----------
        flag : string
            Any of train / valid / test. Defines dataset.
        -----------
        Returns: dataset
        '''
        self.h5_file = h5py.File(flag + '_data.h5', 'r')

        self.y = self.h5_file['windspeed'][:].astype(np.float32)
        self.X = self.h5_file['brcs'][:].astype(np.float32)

        print(f'load {flag} input data: {self.X.shape} ({self.X.nbytes // 1e6}MB)')
        print(f'load {flag} labels: {self.y.shape} ({self.y.nbytes // 1e6}MB)')

    def __len__(self):
        '''required function for the pytorch dataloader: returns len(samples)'''
        return self.X.shape[0]

    def __getitem__(self, idx):
        '''required function for the pytorch dataloader: yields sample at idx'''
        X = self.X[idx]
        y = self.y[idx]
        return (X, y)
```

PyTorch Dataloader

Dataset with N = 12 samples

Batch size k = 4

Shuffled minibatches

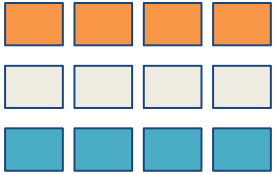


DataLoader retrieves the minibatch:

1st batch

2nd batch

3rd batch



The DataLoader is a Python *iterable*

- Returns its members one at a time
- Implements `__iter__()`
- Implements `__getitem__()` → Dataset
- Use e.g. in for-loops

```
from torch.utils.data import DataLoader

train_loader = DataLoader(train_dataset)

for i, sample in enumerate(train_loader):
    print(i, sample)
```

0,

Measuring Performance

Timing and Counting

Measuring Performance

Consider the following example?

What are some information you might care about to plan resource usage or efficiency of your pre-processing workflow?

```
with open('filename.txt') as fp:
    for line in fp:
        data.append(line)
```

1) Latency: Time elapsed / wallclock time
2) Amount: Amount of data process
3) Throughput
   a) Bytes processed per second
   b) Operations per second

Measuring Performance

Consider the following example?

What are some information you might care about to plan resource usage or efficiency of your pre-processing workflow?

```
with open('filename.txt') as fp:
    for line in fp:
        data.append(line)
```

Measuring Performance

```
from timeit import default_timer
```

```
nbytes = 0
```

```
start = default_timer()
```

```
with open('filename.txt') as fp:
    for line in fp:
        data.append(line)
```

```
        nbytes += len(line)
```

```
end = default_timer()
```

```
elapsed_time = end - start
throughput = nbytes / elapsed_time
```

1) Latency: Time elapsed / wallclock time
2) Amount: Amount of data process
3) Throughput
   a) Bytes processed per second
   b) Operations per second

Measuring time in IPython and Jupyter notebooks

- Line magic: `%time <python command>`
- Cell magic: `%%time`

```
%%time
ds_train_hdf5['brcs'][:];
```

Execute a command several times, and calculate the average runtime: `%%timeit`