# The Helmholtz Analytics Toolkit (Heat)
## and its role in the landscape of massively-parallel scientific Python

C. Comito[1], J. Gutiérrez Hermosillo Muriedas[2], M. Götz[2], B. Hagemeier[1], F. Hoppe[3], P. Knechtges[3], K. Krajsek[1], A. Rüttgers[3], A. Streit[2], M. Tarnawa[1]

[1]Jülich Research Centre (FZJ), Institute for Advanced Simulation, Jülich Supercomputing Centre (JSC) [2]Karlsruhe Institute for Technology (KIT), Steinbuch Centre for Computing (SCC) [3]German Aerospace Center (DLR), Institute for Software Technology, High-Performance Computing


# What is Heat and what can I do with it?

# The Helmholtz Analytics Toolkit

- Free and open-source (MIT) Python library for **memory-distributed high-performance data analysis and machine learning** on CPU/GPU-clusters
- allows **scientists and engineers without an extensive background in HPC** to tackle data-intensive problems that go beyond the capabilities of a workstation
- Developed since 2018 at

DLR

P. Knechtges
F. von der Lehr
A. Rüttgers
M. Siggel

JÜLICH Forschungszentrum

C. Comito
B. Hagemeier
K. Krajsek
M. Tarnawa

KIT Karlsruher Institut für Technologie

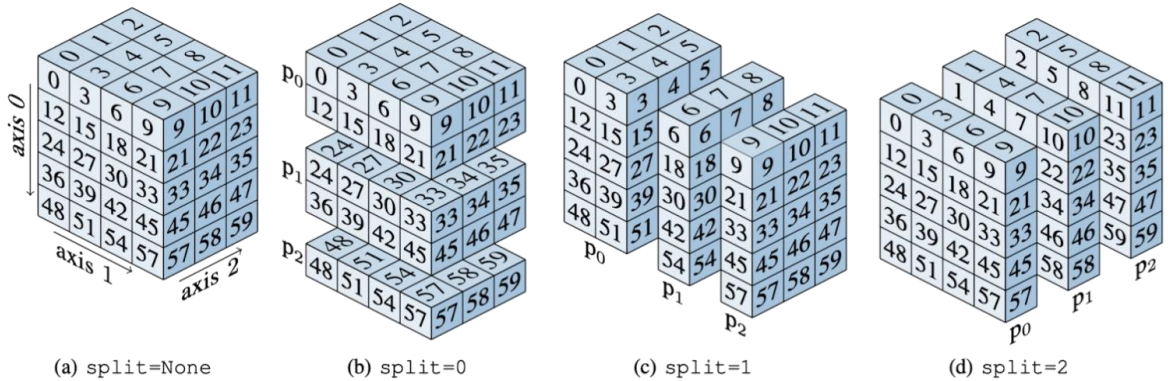
D. Coquelin
C. Debus
M. Götz
J. P. G. Hermosillo Muriedas

+ external *open-source contributors* on github (e.g. GSoC 2022) + *student workers*

# How it works…

**Vendor-independent GPU-support**
(Nvidia + AMD) inherited from PyTorch!

- **distributed N-dimensional array** („**`DNDarray`**“) as basic data type




(a) split=None (b) split=0 (c) split=1 (d) split=2

  - composed of „local“ **`torch`**-Tensors on each MPI-process
  - „MPI-glue” using **`mpi4py`**
  - **`numpy`**-like API

HEAT
Helmholtz Analytics Toolkit

**Clustering**
**(k-means, Spectral)**

**Deep Learning**
**(DASO, [3])**

**Regression**
**(QR, LASSO)**

**Naive Bayes**

***and more…***

PyTorch

Tensors
Linear Algebra

Automatic
Differentiation

NumPy-like
interface

GPU-support
(vendor-independent)

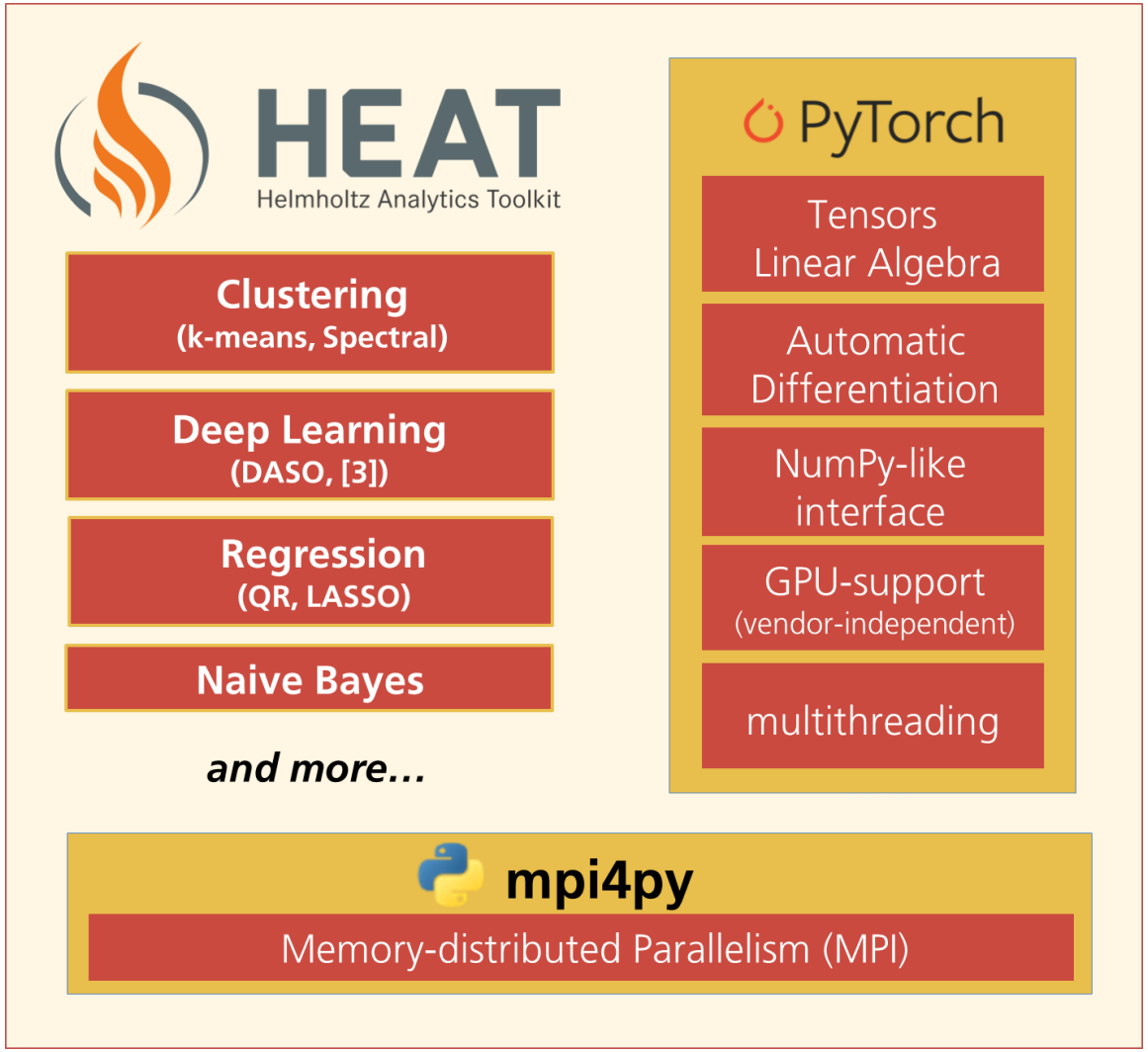
multithreading

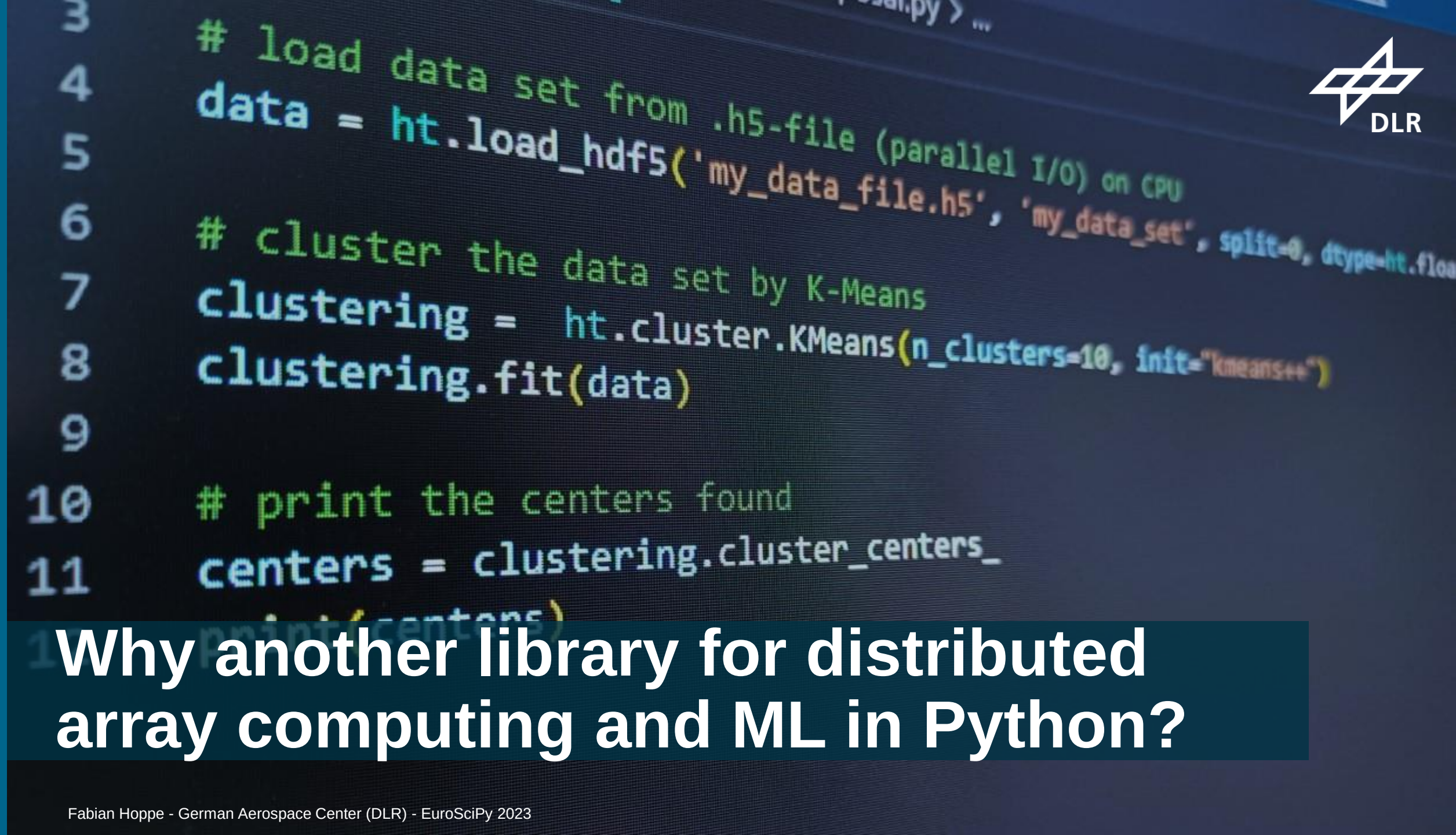
**mpi4py**

Memory-distributed Parallelism (MPI)

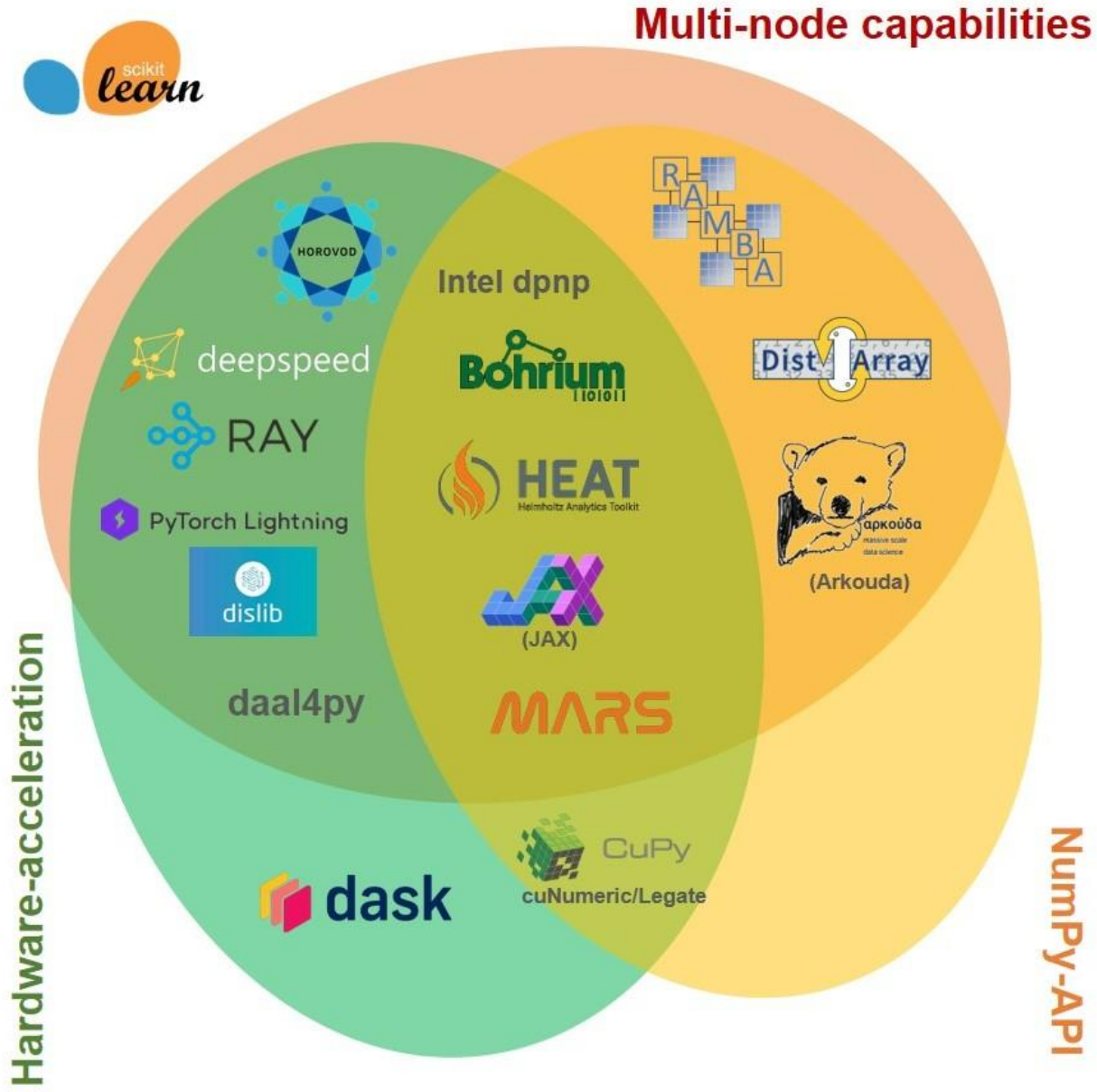
# …and what you can do with it
## (`main`-branch as of August 2023)

- **Parallel I/O** via hdf5, csv, netcdf
- **Array creation and manipulation:** get/set item, concatenate, reshape, zeros, ones, arrays with random entries, math functions…
- basic sparse array class
- **Statistics:** min, max, mean, percentile, variance, …
- **Linear algebra:** transpose, matmul, QR, Lanczos, (approximate) SVD, …
- **Machine learning:** Clustering (k-means, spectral), Regression (LASSO), Naive Bayes, …
- **Data parallel NNs** („DASO“)



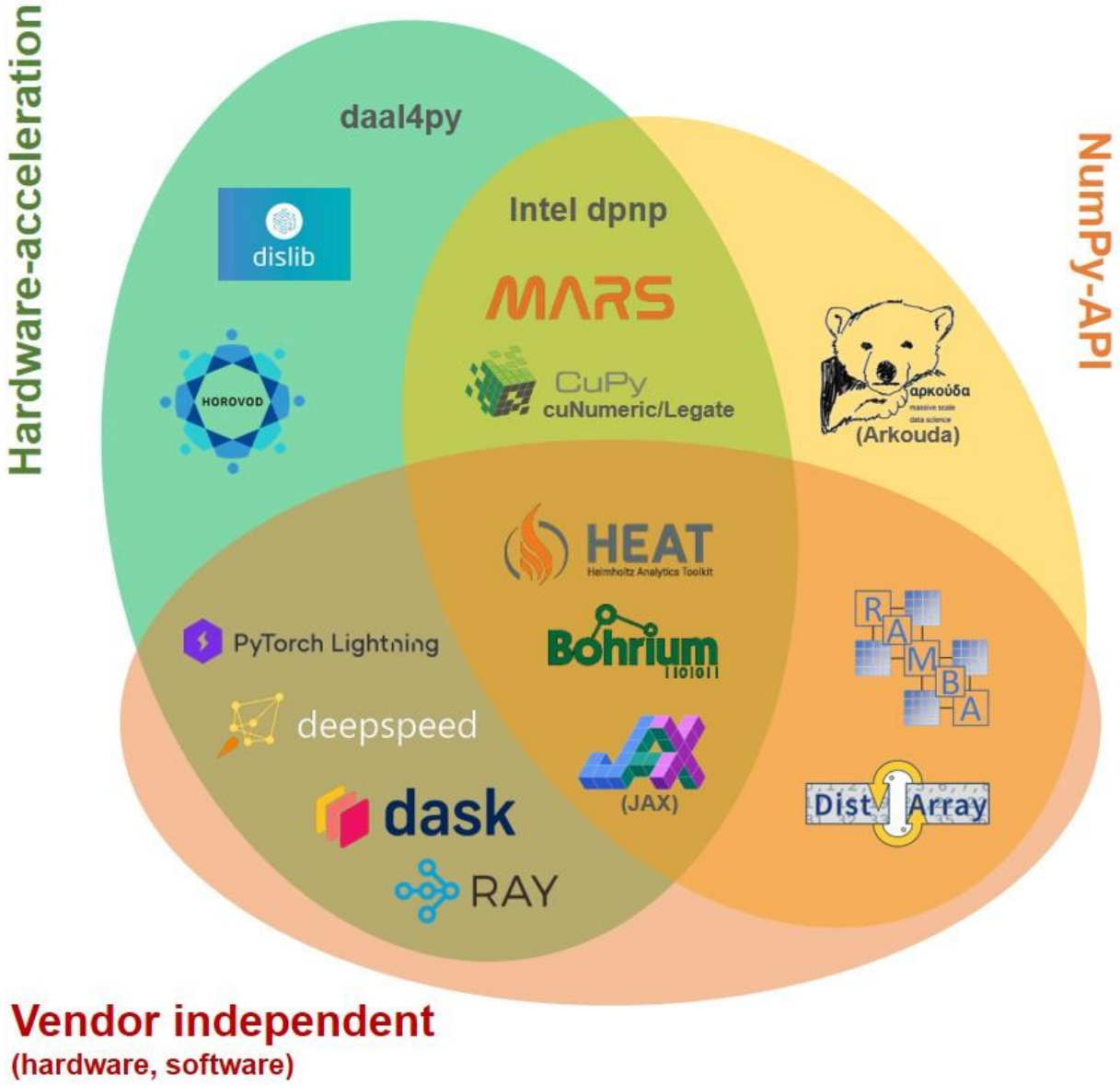



# Why another library for distributed array computing and ML in Python?



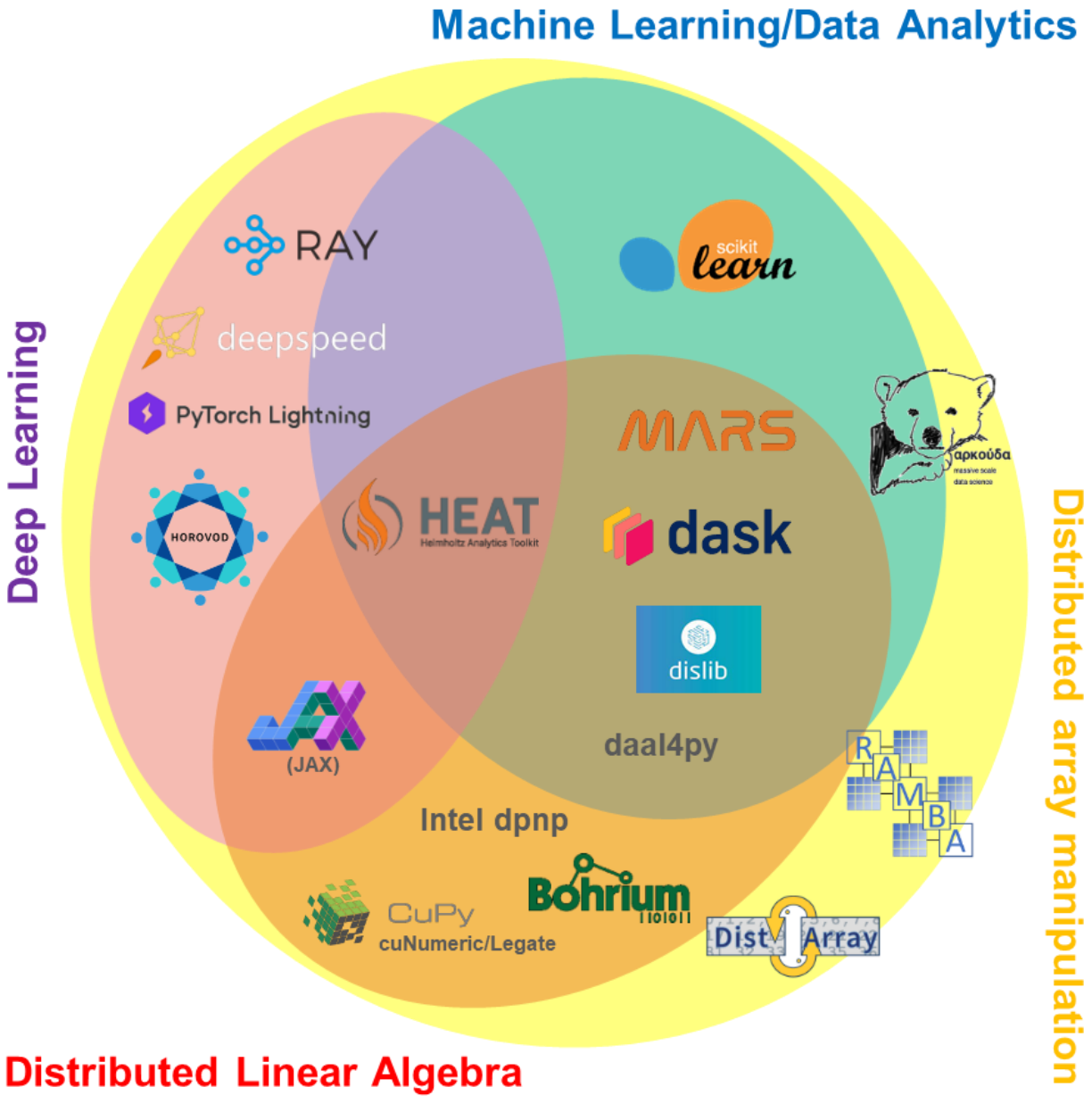

Heat…

- …allows for **memory-distributed**, i.e. **multi-node**, parallelism and is not limited to embarrassingly parallel applications
- …supports **hardware acceleration** (GPU)
- …has a **simple NumPy-/scikit-learn-like API** that allows for easy adaption of legacy code




Heat…

- …allows for **memory-distributed**, i.e. **multi-node**, parallelism and is not limited to embarrassingly parallel applications
- …supports **hardware acceleration** (GPU)
- …has a **simple NumPy-/scikit-learn-like API** that allows for easy adaption of legacy code
- …is **vendor-/platform-independent**
- …is **interoperable** (PyTorch)




Heat…

- …allows for **memory-distributed**, i.e. **multi-node**, parallelism and is not limited to embarrassingly parallel applications
- …supports **hardware acceleration** (GPU)
- …has a **simple NumPy-/scikit-learn-like API** that allows for easy adaption of legacy code
- …is **vendor-/platform-independent**
- …is **interoperable** (PyTorch)
- …is **general-purpose**
- …is developed **by scientists for scientists**.

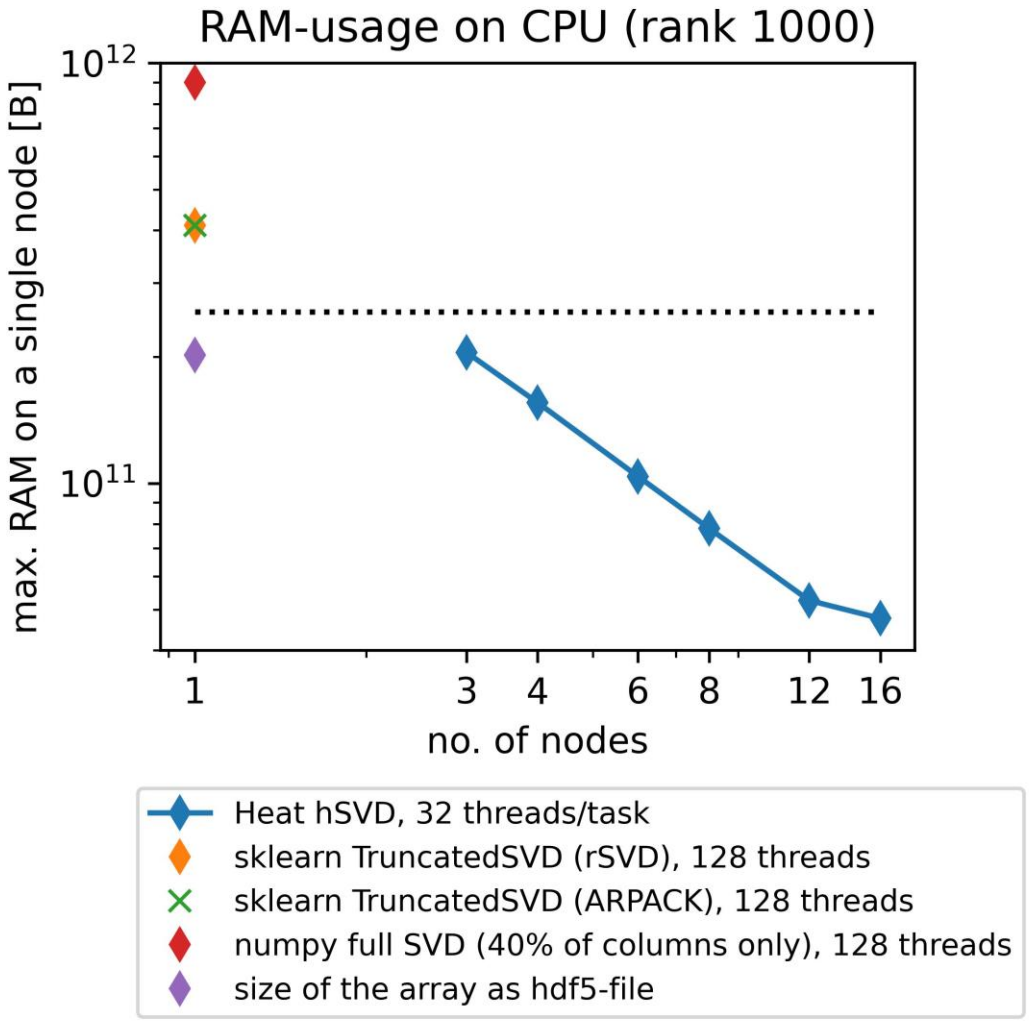

# In a bit more detail…

# Avoid memory becoming a bottleneck
## (a use-case from DLR)

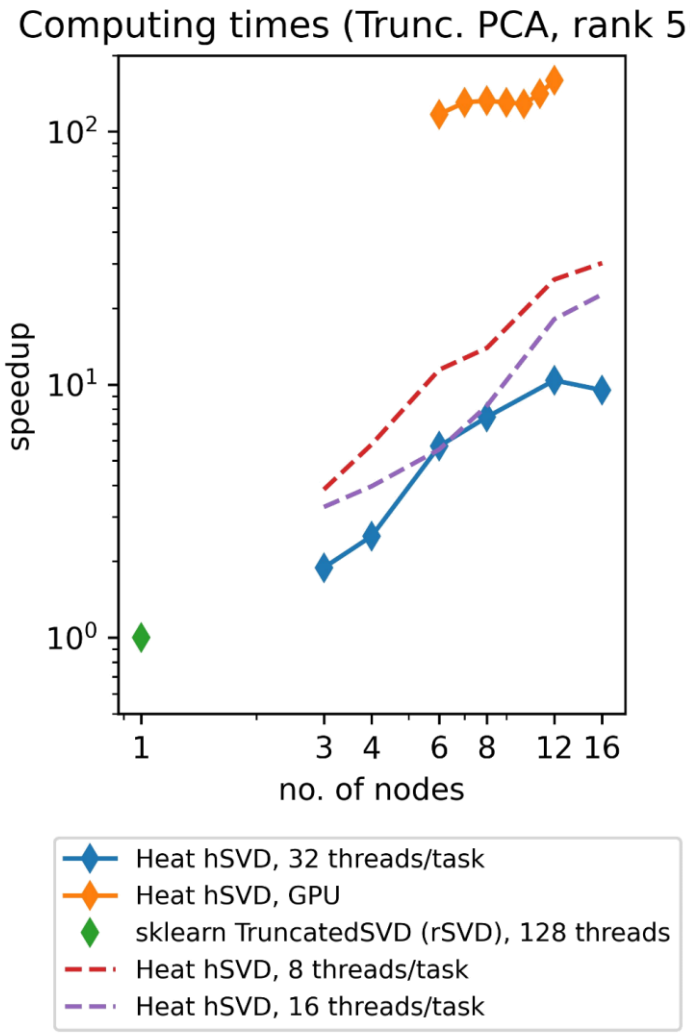
**Use-case:**
**analysis of a data set of 85487 RGB-images with 1024x192 pixels**

- 589824 x 85487 matrix **(~200GB)**
- PCA (here: ***truncated PCA with rank 1000***) and other typical tasks (e.g. clustering, matmul) are not data-parallel
- embarrassingly parallel batch-processing not possible!
- memory becomes a hard limit easily…

Experiments conducted on up to 16 ‚`medium`‘ compute nodes (Heat) and a single ‚`bigmem`‘ node (sklearn, numpy) of DLRs cluster CARO

# Exploit your existing hardware: strong scaling
## (use-case from DLR continued)



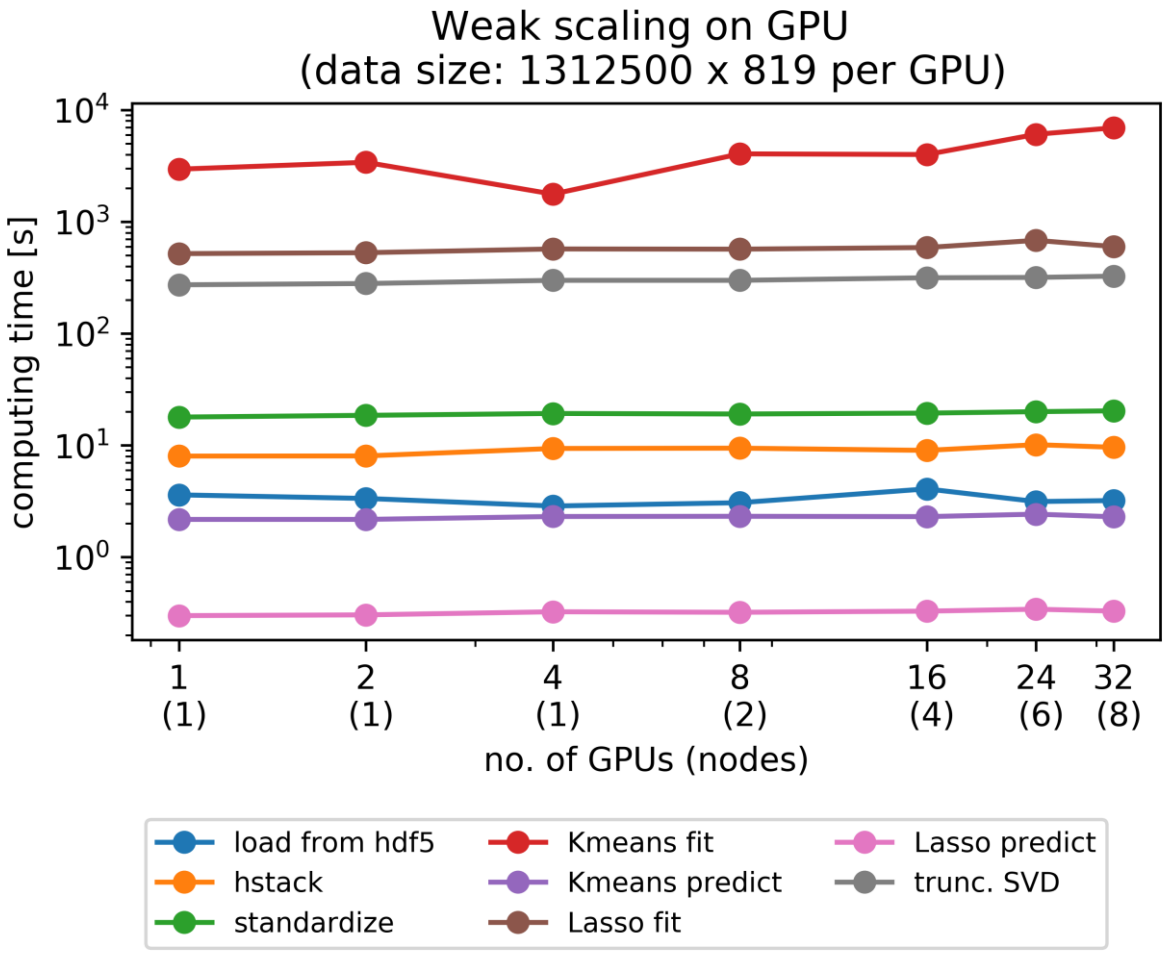

- **CPU:** hybrid parallelism (MPI+OpenMP) may achieve significant speedup compared to single node/single process
- **GPUs:** multi-GPU/multi-node setting is sometimes unavoidable due to memory limitations
- Heat so far runs on:
  - Intel and AMD CPUs
  - Nvidia GPUs
  - AMD GPUs (helmholtz.codebase runners)
  - Apple CPU+GPU (developers notebook, experimental)

Experiments conducted on:

- **sklearn:** one ‚`bigmem`‘ node of CARO
- **Heat CPU:** up to 16 ‚`medium`‘ compute nodes of CARO
- **Heat GPU:** up to 12 nodes (with 4 Nvidia A100 80GB each) of DLRs cluster Terrabyte


terrabyte

# Scale your applications: weak scaling




(Load with split=0, hstack along axis=1, Kmeans with 4 clusters, Lasso with regularization parameter 0.1 and tolerance 0.01, truncated SVD by hSVD with rel. tolerance 0.01)

**Test data set:**

*ATLAS Top Tagging Open Data Set* (42 mio. entries with 819 features each ~ 129GB (“train” part) )

[ATLAS collaboration (2022). ATLAS Top Tagging Open Data Set. CERN Open Data Portal. DOI:10.7483/OPENDATA.ATLAS.FG5F.96GA]

Experiments conducted on up to 8 GPU-nodes of DLRs cluster Terrabyte (with 4 Nvidia A100 80GB each)

# Benefits of the simple API

- Straight-forward **adaptation of legacy NumPy-/scikit-learn-workflows**:

```
import heat as ht


data = ht.load("your_hdf5_data_set.h5",
dataset="name_of_the_data", split=1, device="gpu")
clustering =  ht.cluster.KMeans(n_clusters=10,
init="kmeans++")
clustering.fit(data)
print(clustering.cluster_centers_)
```

- Simple specification of devices (similar to PyTorch) allows to **use GPUs with almost no additional effort**

# Benefits of the simple API

- Except for prescribing a split-direction, **parallelization does not enter your python-script** explicitely
  - no explicit chunking of the data required
  - easy to run on HPC-systems using SLURM:

```
# on GPU
srun --nodes=8 --ntasks-per-node=4 --gres:gpu=4 python
script.py
```

```
# on CPU (incl. multithreading)
srun --nodes=8 --ntasks-per-node=8 --cpus-per-task=20
python script.py
```

# Comparison with Dask (CPU)

## 129GB data set* on 8 CPU-nodes

| | HEAT | DASK |
|---|---|---|
| *load from hdf5* | **3 s** | 3 min |
| *stack data* | 3 s | **18 ms** |
| *standardize data* | **2 s** | 5 min |
| *K-means fit (4 clusters)…* | **40 min** | 12 h |
| *…and predict* | **1 s** | 7 min |
| *PCA (trunc. to rank 100)* | **31 s** | 10 min |
| *randn…* | 9 s | **3 ms** |
| *…and save to hdf5* | **41 s** | **48 s** |
| ***MaxRSS for all this:*** | **14 GB** | 460 GB |

Experiments conducted on up to 8 CPU-nodes of DLRs cluster Terrabyte (with 2 Intel Xeon Platinum 8380 40C 270W 2.3GHz each)

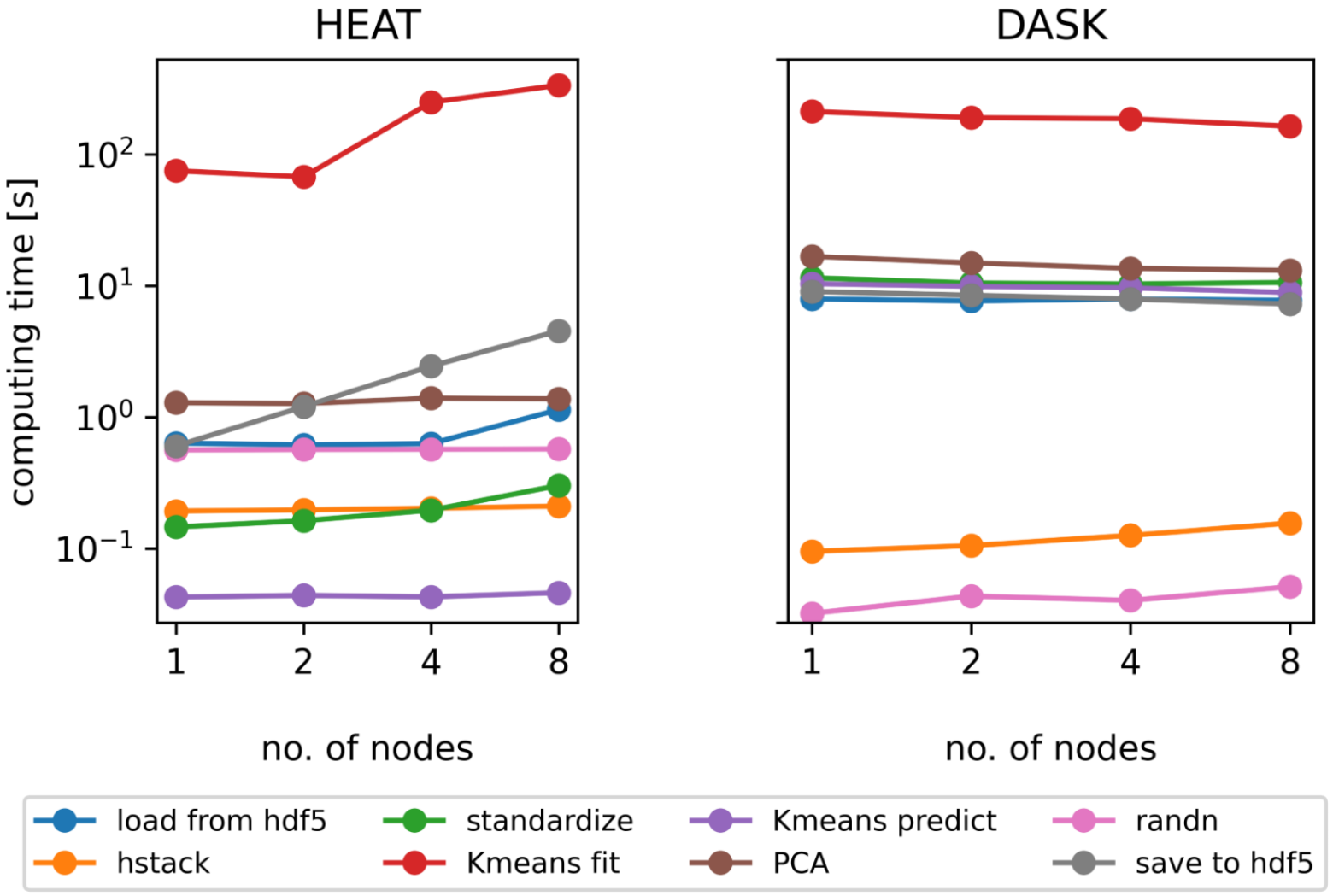
8 tasks per node, 20 threads per task


**Disclaimer:** These experiments have been run by a „Heat-Expert“, but a „Dask-Non-Expert“ – code available upon request.

***** „train“ part of *ATLAS Top Tagging Open Data Set*

# Comparison with Dask (CPU)

## Weak scaling of runtime on a small data set* (7.6GB)




Experiments conducted on up to 8 CPU-nodes of DLRs cluster Terrabyte (with 2 Intel Xeon Platinum 8380 40C 270W 2.3GHz each)

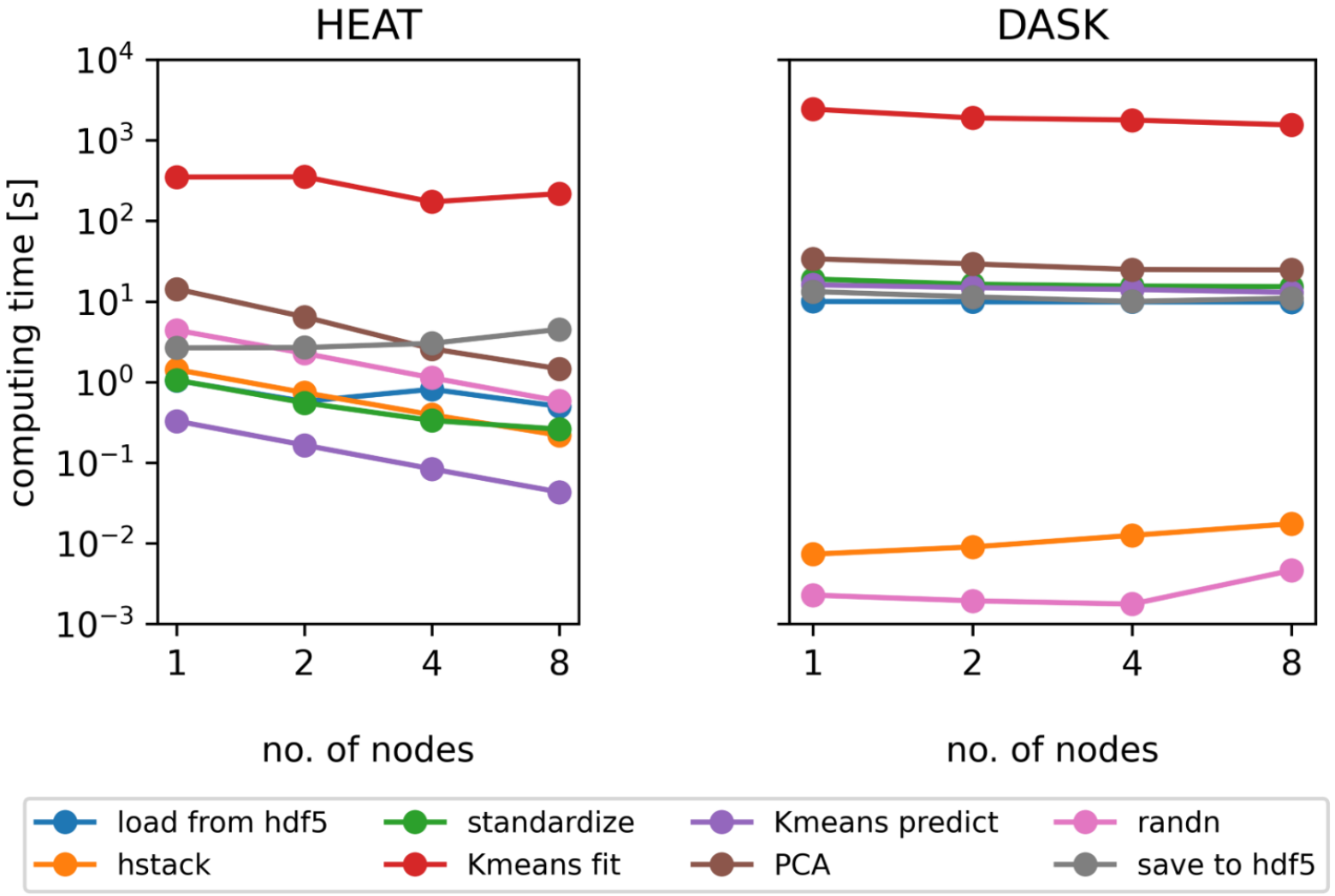
8 tasks per node, 20 threads per task

powered by terrabyte

***** „test“ part of *ATLAS Top Tagging Open Data Set*

# Comparison with Dask (CPU)

## Strong scaling of runtime on a small data set* (7.6GB)




Experiments conducted on up to 8 CPU-nodes of DLRs cluster Terrabyte (with 2 Intel Xeon Platinum 8380 40C 270W 2.3GHz each)

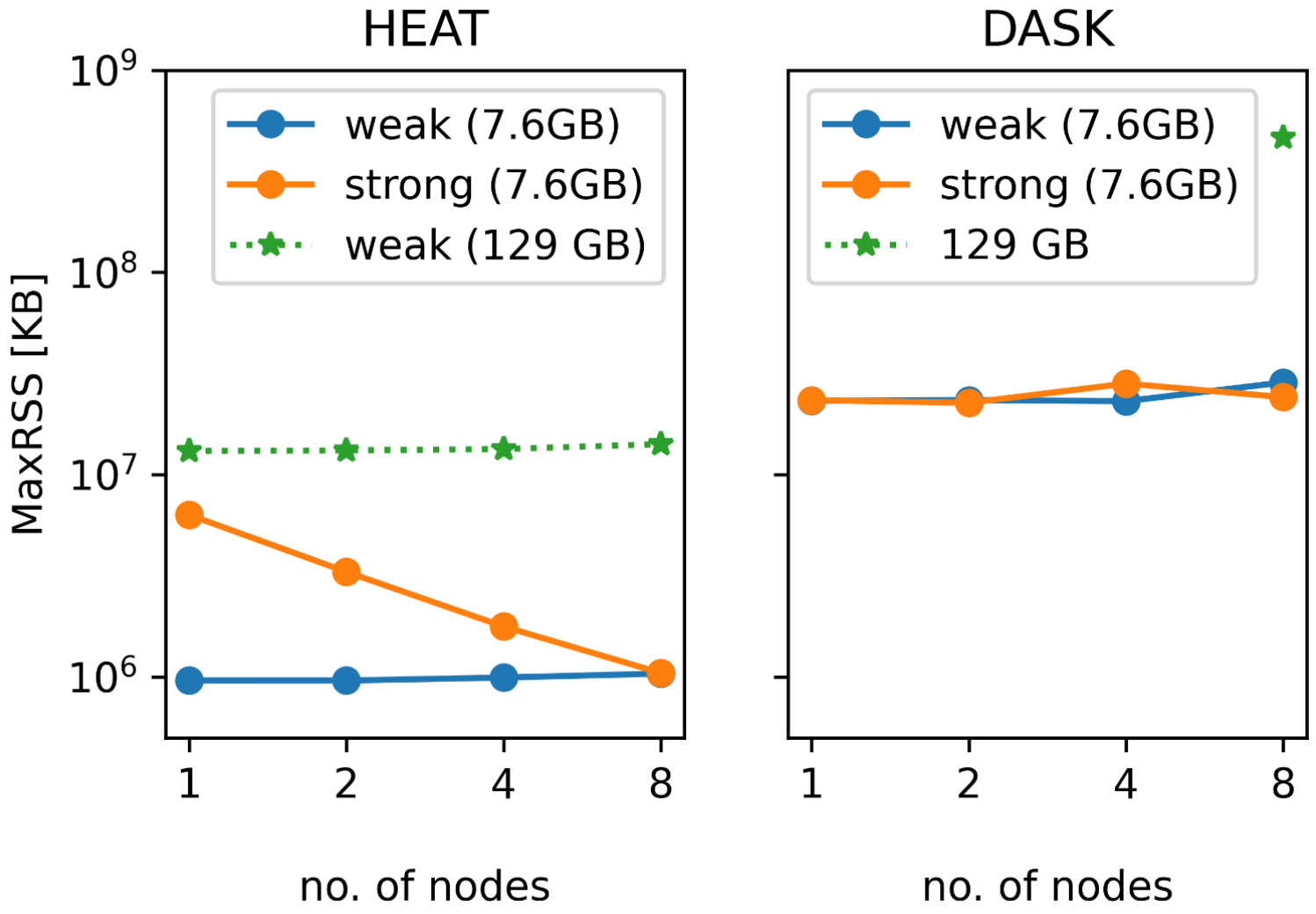
8 tasks per node, 20 threads per task

powered by terrabyte

* „test“ part of *ATLAS Top Tagging Open Data Set*

# Comparison with Dask (CPU)

## Weak and strong scaling of used RAM on a small data set* (7.6GB)




Experiments conducted on up to 8 CPU-nodes of DLRs cluster Terrabyte (with 2 Intel Xeon Platinum 8380 40C 270W 2.3GHz each)

8 tasks per node, 20 threads per task

powered by terrabyte

***** „test“ part of *ATLAS Top Tagging Open Data Set*


# Future development and challenges

# Future development and challenges

- **Software engineering:** keep/increase *software quality* by, e.g., …
  - continuous benchmarking of runtime, memory consumption, and energy consumption („green HPC“)
  - keep/extend interoperability w.r.t. the evolving ecosystem
- **Extend fuctionality**, e.g.,
  - implementation of support for distributed **`xarray`**‘s has just started (➔EOC-community)
  - implementation of an SVD/PCA/DMD-module (➔ ESA)
  - preprocessing module, FFT, … ➔ ***or maybe your suggestion?***
- **Community interaction**, e.g., …
  - extend documentation by tutorials/use cases etc.
  - user-support by workshops and/or individual collaboration/help:
    ***If you want to try out/use Heat for your research, we are glad to help you; we do <u>not</u> expect you to contribute to Heat.***

# Feel free to try out, to suggest features, or to contribute!


visit us on **github**:

# Impressum

| | |
|---|---|
| Thema: | **The Helmholtz Analytics Toolkit (Heat)** and its role in the landscape of massively-parallel scientific Python |
| Datum: | 2023-08 |
| Autor: | Fabian Hoppe (+ the Heat development team) |
| Institut: | Institut für Softwaretechnologie, High-Performance Computing |
| Bildcredits: | Alle Bilder „DLR (CC BY-NC-ND 3.0)“, sofern nicht anders angegeben |