

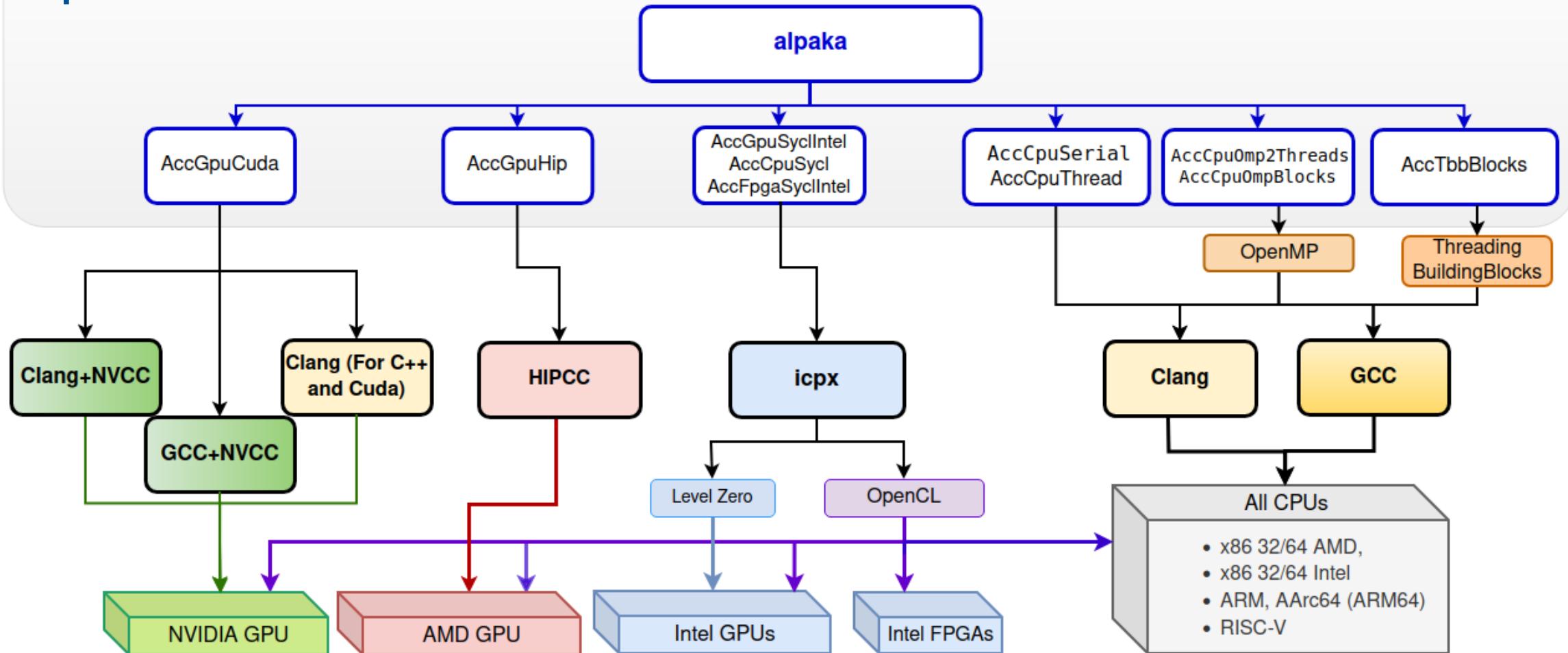
Plasma-PEPSC Workshop  
23 October 2024

# The Open Standard for Particle-Mesh Data

We saw this yesterday...

what about I/O?

## Alpaka abstraction tree



# ...let's keep talking about performance portability

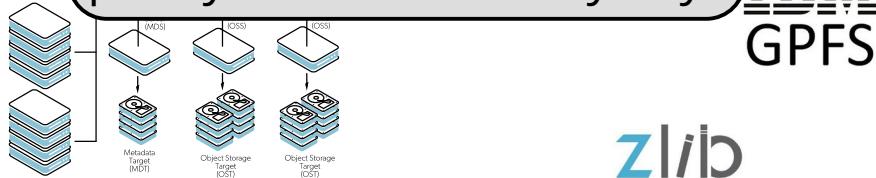
## compute resources

serial filesystems

parallel: Lustre, GPFS, ...

file striping? subfiling?

plenty of RAM? scarcely any?



Lustre File System

IBM GPFS

## compression

compute on CPU/GPU?

lower file size,

but more compute effort?

c-blosc, zfp, sz, bzip, zlib, ...



zlib

bzip2

## backends / file formats

HDF5? ADIOS?

ASCII: JSON, TOML, XML, ...

the new kid?

ADIOS

HDF  
The HDF Group



## data storage / transport

even write to a file?

or stream to a consumer?

shared memory?

or do we need the network?

Infiniband, Slingshot, WAN, ...?

## data path

synchronous? asynchronous?

Burst Buffers?

staging buffer in RAM?

serial or parallel I/O?

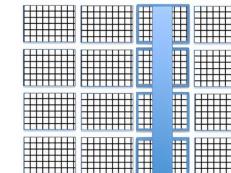
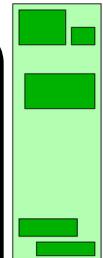
parallel aggregation?

## serialization

structured format?

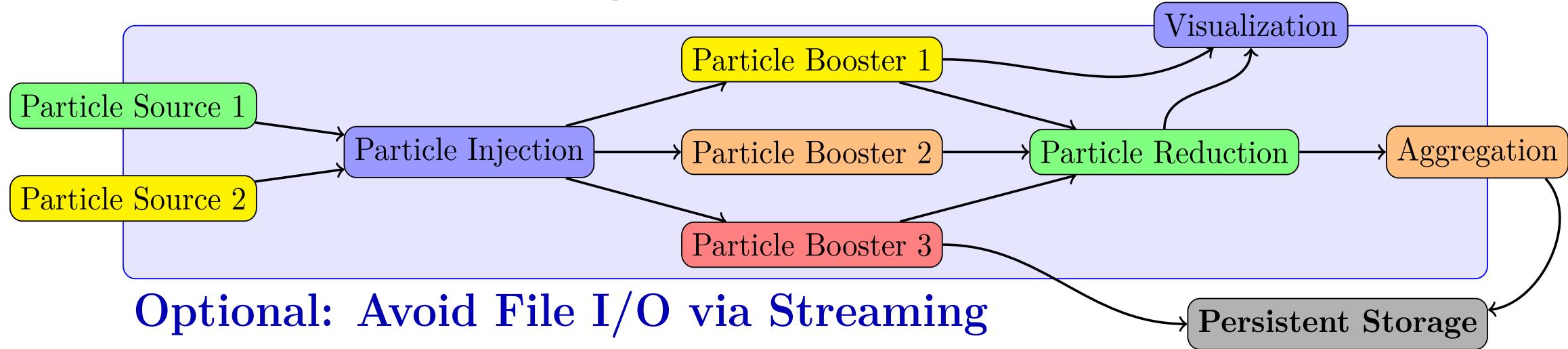
log format?

read patterns?



None of these questions has a clear answer!

Scientific workflows are complex:

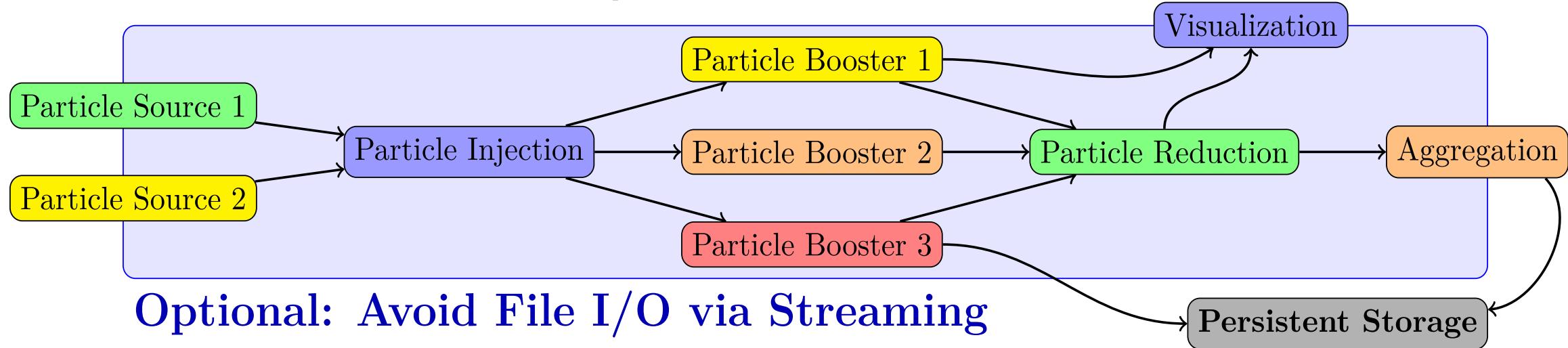


Optional: Avoid File I/O via Streaming

- need to span different **time** and **length scales**
- scientific modeling requires **multiple codes**,  
collaborating in a **data processing pipeline**
- **bridge heterogeneous models** by standardization of data

# Heterogeneity through Standardized Data

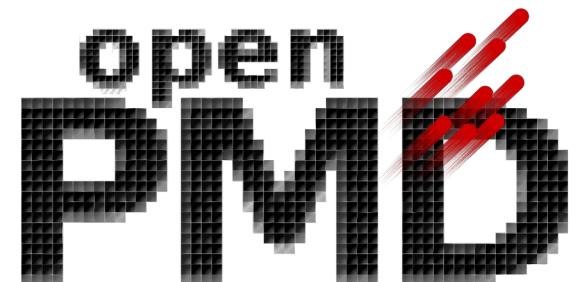
Scientific workflows are complex:



Optional: Avoid File I/O via Streaming

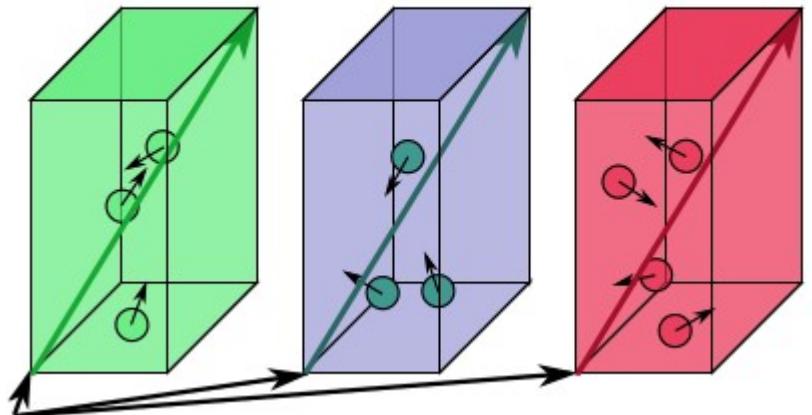
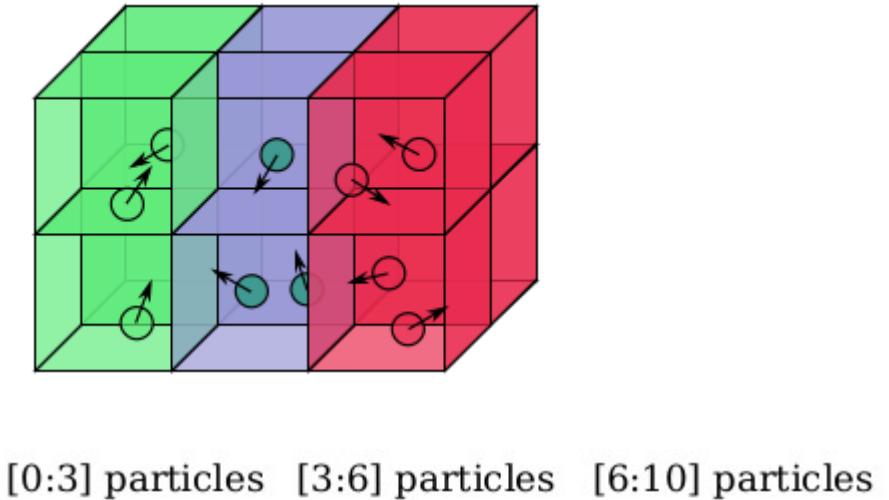


openPMD standard  
 for **particle-mesh data**  
 as communication layer



Axel Huebl et al. "openPMD: A meta data standard for particle and mesh based data". 2015. doi: 10.5281/zenodo.591699. url: <https://openPMD.org>  
 Franz Poeschel et al. "Transitioning from file-based HPC workflows to streaming data pipelines with openPMD and ADIOS2". 2021. doi:10.1007/978-3-030-96498-6\_6

# What is particle-mesh data?



## Mesh

n-dimensional space,  
 divided into discrete cells

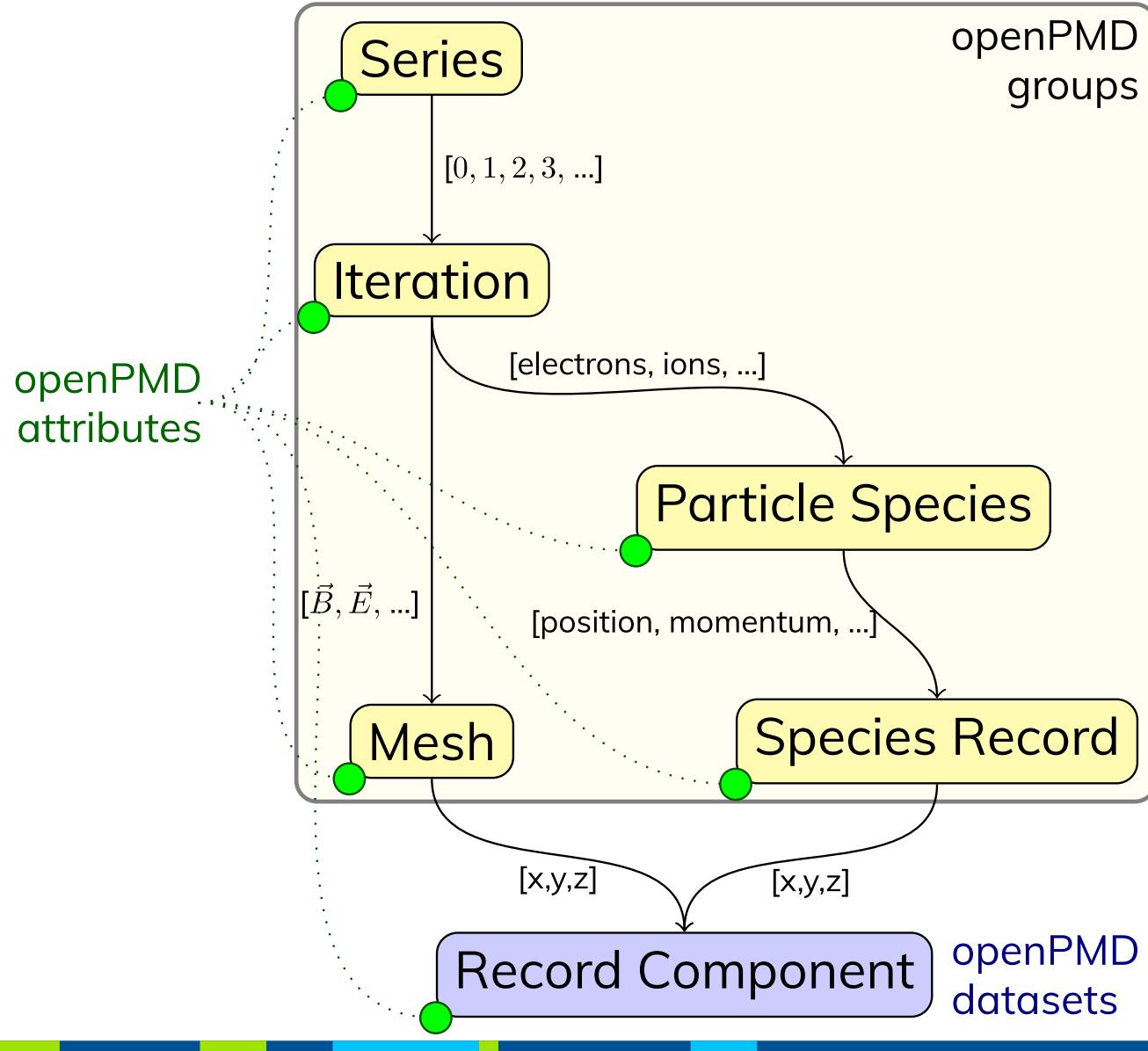
- e.g. temperature:  
 store a scalar number per cell
- e.g. electrical fields:  
 store a 3D vector per cell

## Particles

A list of discrete objects,  
 located on the mesh

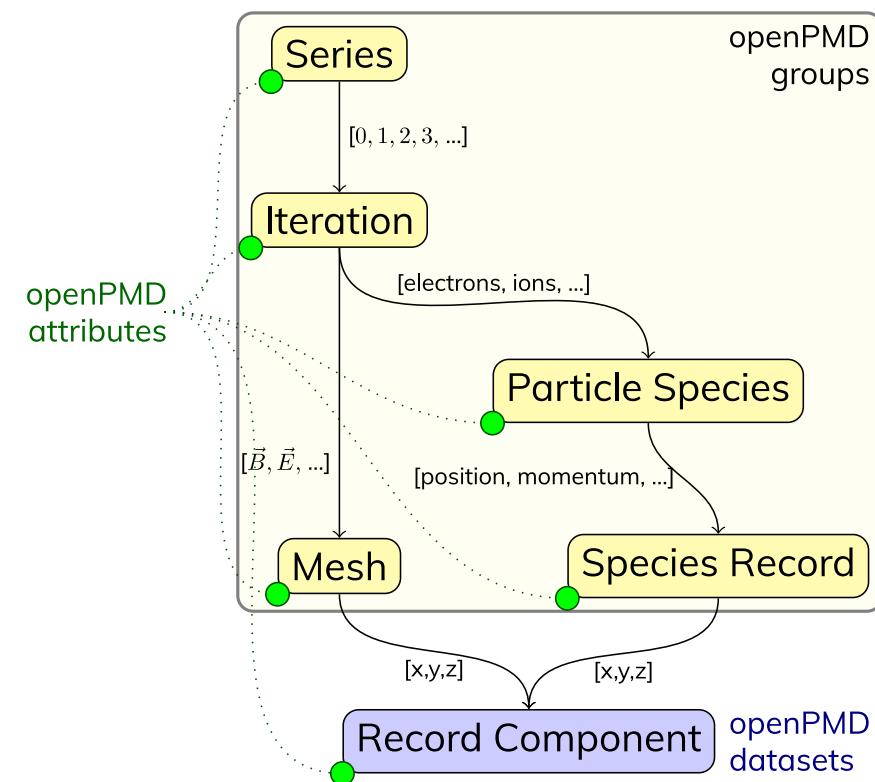
- for each particle: list its position
- optionally: list charge, weight, ...

# openPMD hierarchy

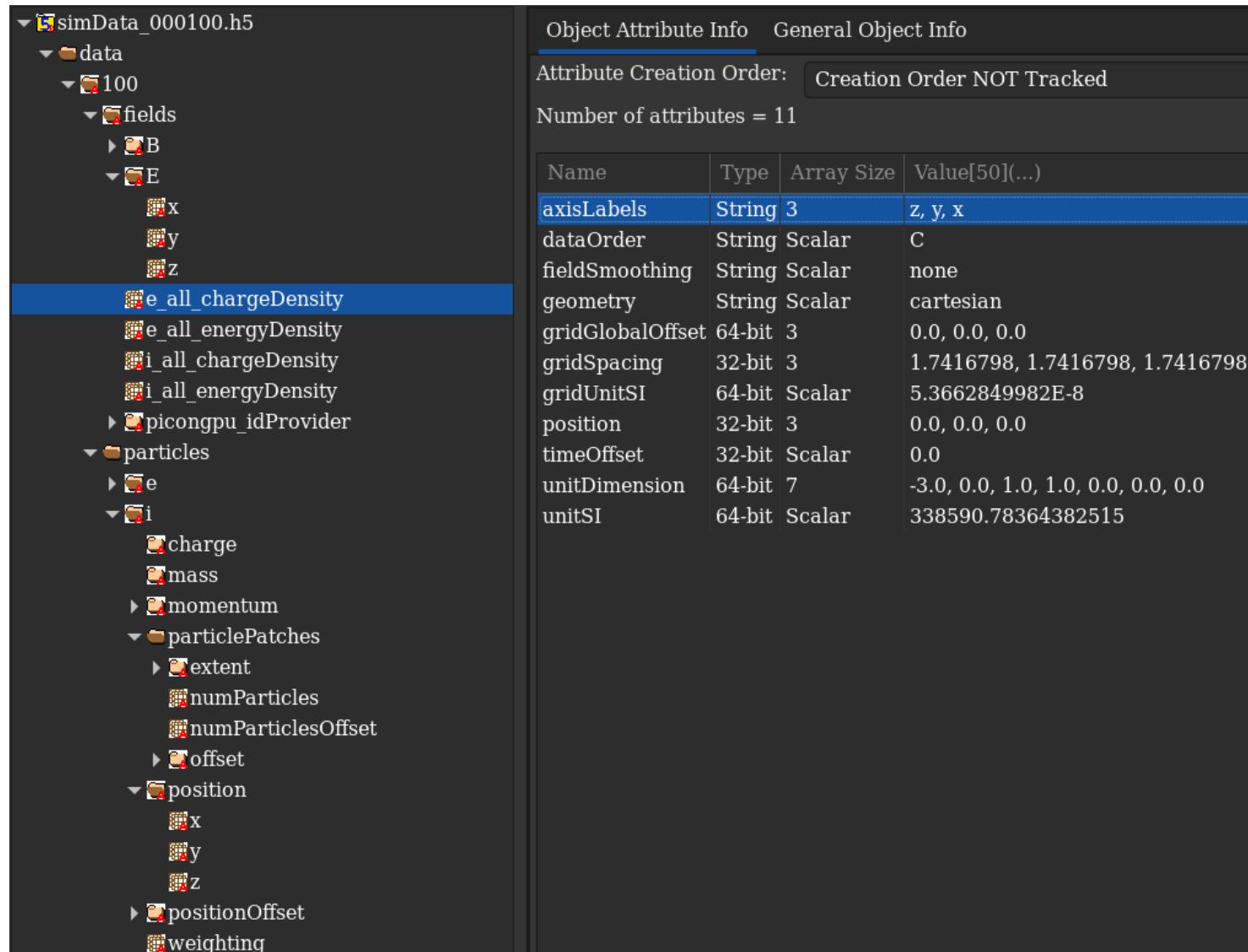


- **Structure** for series & snapshots encoded as either:
  - **files** (one file per iteration)
  - **groups** (reuse files)
  - **variables** (reuse files & variables in ADIOS2)
- Records for **physical observables** constants, mixed precision, complex numbers
- **Attributes**: unit conversion, description, relations, mesh geometry, authors, env. info, ...

# Example dataset: HDF5 backend



Sample data  
created with PIConGPU



# Example dataset: ADIOS2 backend

float	/data/50/fields/E/x	Hierarchical data organization	{128, 128, 128}	n-dim. datasets for heavyweight data
float	/data/50/fields/E/y		{128, 128, 128}	
float	/data/50/fields/E/z		{128, 128, 128}	
float	/data/50/particles/e/position/x		{50053105}	
float	/data/50/particles/e/position/y		{50053105}	
float	/data/50/particles/e/position/z		{50053105}	
int32_t	/data/50/particles/e/positionOffset/x		{50053105}	
int32_t	/data/50/particles/e/positionOffset/y		{50053105}	
int32_t	/data/50/particles/e/positionOffset/z		{50053105}	
string	/data/50/fields/E/axisLabels		attr = {"z", "y", "x"}	Attributes for self-description
string	/data/50/fields/E/dataOrder		attr = "C"	
string	/data/50/fields/E/fieldSmoothing		attr = "none"	
string	/data/50/fields/E/geometry		attr = "cartesian"	
double	/data/50/fields/E/gridGlobalOffset		attr = {0, 0, 0}	
float	/data/50/fields/E/gridSpacing		attr = {1.74168, 1.74168, 1.74168}	
double	/data/50/fields/E/gridUnitSI		attr = 5.36628e-08	
float	/data/50/fields/E/timeOffset		attr = 0	
double	/data/50/fields/E/unitDimension		attr = {1, 1, -3, -1, 0, 0, 0}	
float	/data/50/fields/E/x/position		attr = {0.5, 0, 0}	
double	/data/50/fields/E/x/unitSI		attr = 9.5224e+12	
float	/data/50/fields/E/y/position		attr = {0, 0.5, 0}	
double	/data/50/fields/E/y/unitSI		attr = 9.5224e+12	
float	/data/50/fields/E/z/position		attr = {0, 0, 0.5}	
double	/data/50/fields/E/z/unitSI		attr = 9.5224e+12	

# Example dataset: JSON/TOML backend

```
{
  "attributes": {
    "basePath": "/data/%T/",
    "date": "2023-10-11 09:57:53 +0200",
    "iterationEncoding": "fileBased",
    "iterationFormat": "simData_%06T",
    "meshesPath": "fields/",
    "openPMD": "1.1.0",
    "openPMDextension": 0,
    "particlesPath": "particles/",
    "picongpuIOVersionMajor": 2,
    "picongpuIOVersionMinor": 0,
    "software": "PICConGPU",
    "softwareVersion": "0.7.0-dev"
  },
  "data": {
    "100": {
      "attributes": {
        "cell_depth": 4.252342224121094,
        "cell_height": 1.0630855560302734,
        "cell_width": 4.252342224121094,
        "dt": 1,
        "eps0": 169.19711303710938,
        "many": "more"
      },
      .....
      "fields": {
        "e_all_energyDensity": {
          "attributes": {
            "axisLabels": [ "z", "y", "x" ],
            "geometry": "cartesian",
            "gridGlobalOffset": [ 0, 0, 0 ],
            "gridSpacing": [ 4.25, 1.06, 4.25 ],
            "gridUnitSI": 4.1671151662e-08,
            "position": [ 0, 0, 0 ],
            "timeOffset": 0,
            "unitDimension": [ -1, 1, -2, 0, 0, 0, 0, 0 ],
            "unitSI": 7869098408118.734
          },
          "datatype": "FLOAT",
          "data": [
            [
              [
                [
                  "multidimensional dataset here"
                ]
              ]
            ]
          ]
        }
      }
    }
  }
}
```

- Part of the package: No need to install 3rd-party dependencies
- Useful for debugging and prototyping
- Limited parallel support
- Courtesy to Nils Lohmann's JSON library for C++
- With recent release:  
Convert output to TOML  
Idea: openPMD formatted configuration files

# Schedule for the interactive sessions

## Today:

Introduction to the API and backends

- Modeling the HeatEquation's output as openPMD data
- Specifying backend options with configurations files loaded at runtime
- Visualizing the HeatEquation's output in a Jupyter Notebook
- Streaming I/O: Watching the HeatEquation while it is running

## Tomorrow:

Data Analysis, parallel I/O, advanced topics

- Using the openPMD-viewer to visualize data created by a PICOnGPU simulation
- Using Paraview to do the same in 3D
- More detailed modeling for openPMD data: particles, constant components
- Parallel I/O with ADIOS2 and HDF5
- Advanced session: Eliding memory copies by asking the I/O libraries to allocate buffers for us

**(Optional) Preparing for tomorrow's Paraview session:** Download version 5.13 from [https://www.paraview.org/download/rsync -avup lumi:/pfs/lustrep3/scratch/project\\_465001310/poeschel/2024-09-20/8\\_v1/simOutput/openPMD/simData\\_001800.bp5](https://www.paraview.org/download/rsync -avup lumi:/pfs/lustrep3/scratch/project_465001310/poeschel/2024-09-20/8_v1/simOutput/openPMD/simData_001800.bp5) ~/Downloads/



# Reference Implementation in C++ & Bindings: Python and Julia



## Online Documentation: openpmd-api.readthedocs.io

The screenshot shows the documentation page for the openpmd-api. It includes a sidebar with sections like Installation, Changelog, Upgrade Guide, Usage, Concepts, First Write, Open, Iteration, Attributes, Data, Record, Units, Register Chunk, and Flush Chunk. The main content area shows how to use the API after installation, providing code snippets for C++17 and Python. The C++17 snippet includes #include <openPMD/openPMD.hpp>, // example: data handling #include <numeric> // std::iota #include <vector> // std::vector, and namespace io = openPMD;. The Python snippet includes import openpmd\_api as io, # example: data handling import numpy as np, and series = io.Series("myOutput/data\_%05T.h5", io.Access.CREATE). Below this, there's an 'Open' section with instructions for writing into a new openPMD series in myOutput/data\_<00...N>.h5, mentioning support for .bp (ADIOS1/ADIOS2) or .json (JSON). Code examples for both are provided.

## Open-Source Development & Tests: github.com/openPMD/openPMD-api

The screenshot shows a GitHub Actions status page for a pull request. It displays a green checkmark icon and the text "All checks have passed" followed by "25 successful checks". Below this, it lists six successful builds with their details: macOS / appleclang12\_py\_mpi\_h5\_ad2 (pull\_request) Successful in 17m, Windows / MSVC w/o MPI (pull\_request) Successful in 6m, Intel / ICC C++ only (pull\_request) Successful in 7m, Tooling / Clang ASAN UBSAN (pull\_request) Successful in 58m, Nvidia / CTK@11.2 (pull\_request) Successful in 4m, and Linux / clang8 py38 mpich h5 ad1 ad2 newLayout (pull request) Successful in 29m. Each entry has a "Details" link to the full build log.

## Rapid and easy installation on any platform:



python3 -m pip install  
openpmd-api



brew tap openpmd/openpmd  
brew install openpmd-api



cmake -S . -B build  
cmake --build build  
--target install



conda install  
-c conda-forge  
openpmd-api



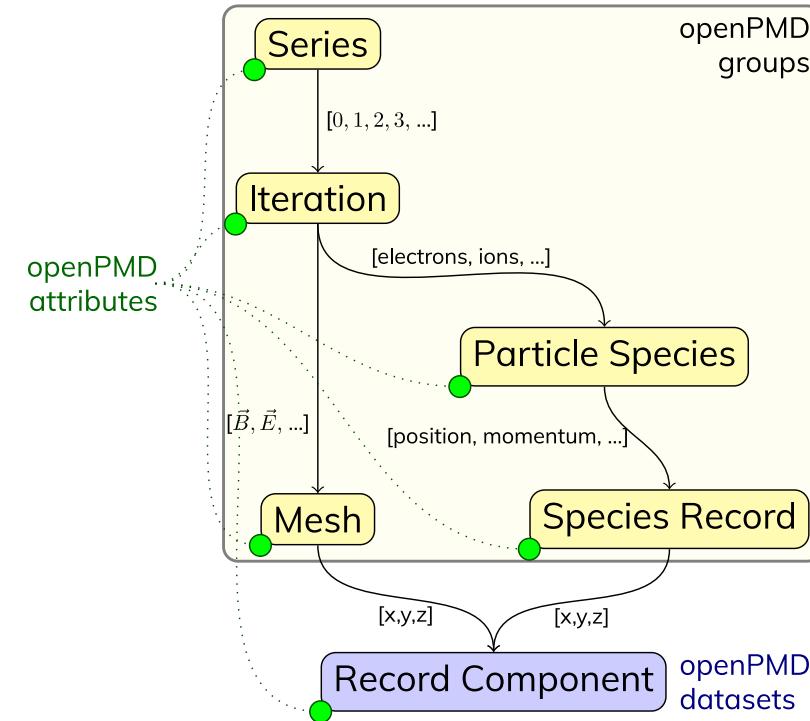
spack install  
openpmd-api



module load openpmd-api

A Huebl, F Poeschel, F Koller, J Gu, et al.  
"openPMD-api: C++ & Python API for Scientific I/O with openPMD" (2018) DOI:10.14278/rodare.27

# Hands-On: openPMD-api: basic object model



**Module environment** at `/project/project_465001310/workshop_software/env.sh`

**Materials** at [https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024\\_workshop](https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024_workshop)

Read the TODO comments inside `src/openPMDOutput.hpp`:

## unitDimension

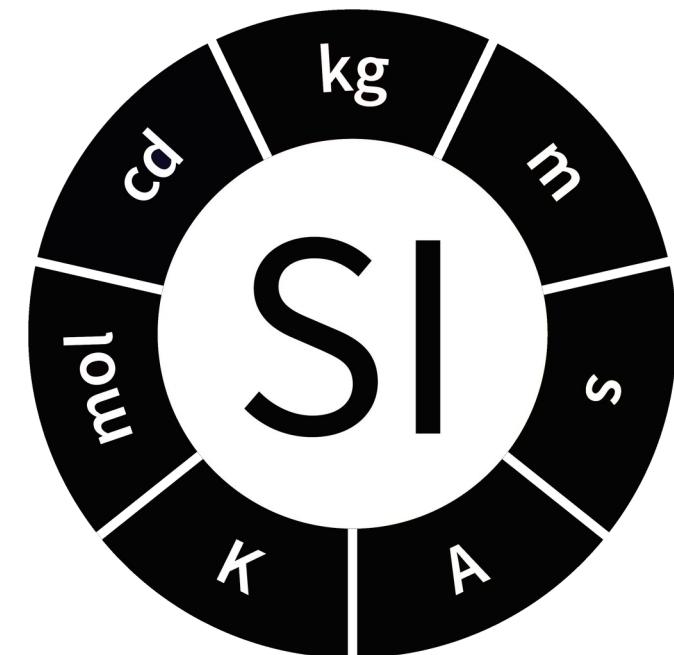
automated description of physical dimension  
only powers of base dimensions

length **L**, mass **M**, time **T**, electric current **I**,  
thermodynamic temperature **theta**,  
amount of substance **N**, luminous intensity **J**

Magnetic field:  $[B] = M / (I * T^2)$   
 $\rightarrow (0, 1, -2, -1, 0, 0, 0)$

## unitSI (recommended)

relation to an absolute unit system

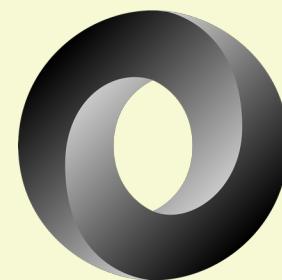


Wikimedia Commons

**Findable:** Standardized metadata to identify the data producer

```
string      /author           attr   = "franz"  
string      /software          attr   = "PICOnGPU"  
string      /softwareVersion    attr   = "0.5.0-dev"
```

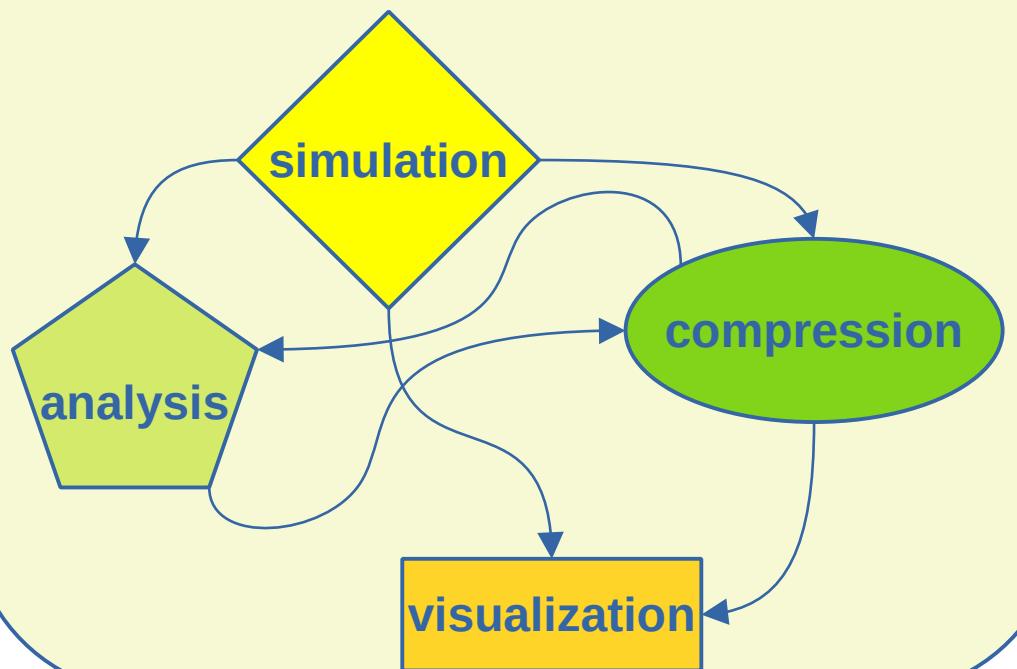
**Accessible:** Open standard, implementable in various formats



\*currently implemented,  
but not limited to

## Interoperable:

Data exchange spans applications, platforms and teams



## Reusable:

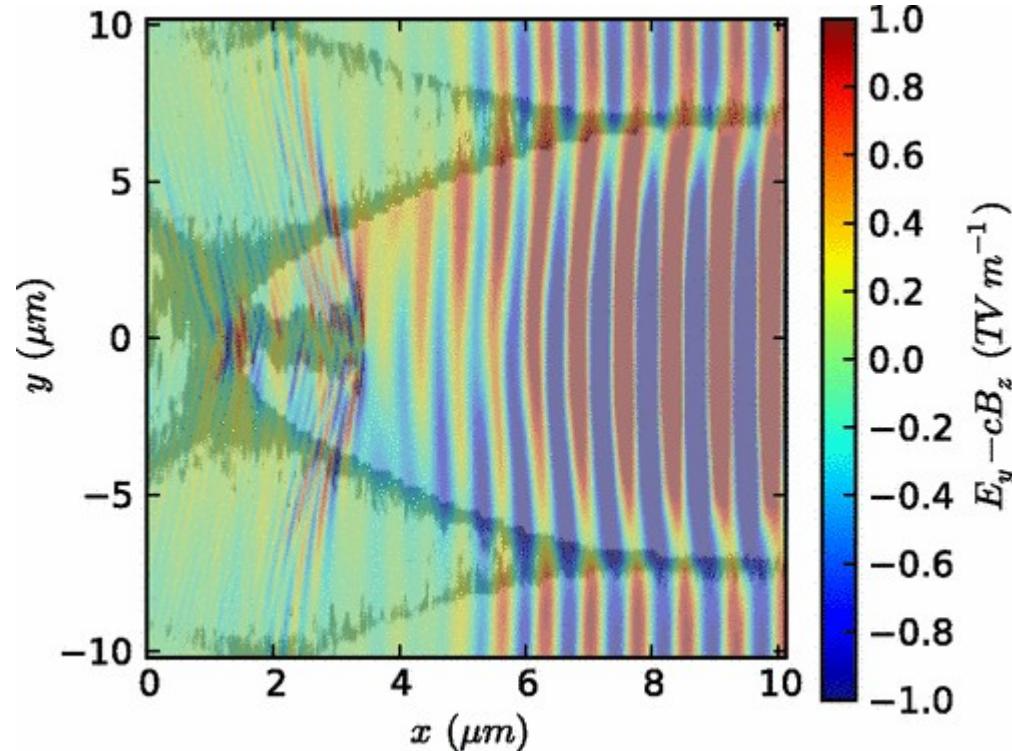
Rich and standardized description for physical quantities

Name	Value
axisLabels	[b'z' b'y' b'x']
dataOrder	b'C'
fieldSmoothing	b'none'
geometry	b'cartesian'
gridGlobalOffset	[0. 0. 0.]
gridSpacing	[4.252342 1.0630856 4.252342 ]
gridUnitSI	4.1671151662e-08
position	[0. 0. 0.]
timeOffset	0.0
unitDimension	[-3. 0. 1. 1. 0. 0. 0.]
unitSI	15399437.98944343

"The FAIR Guiding Principles for scientific data management and stewardship" (Mark D. Wilkinson et al.)

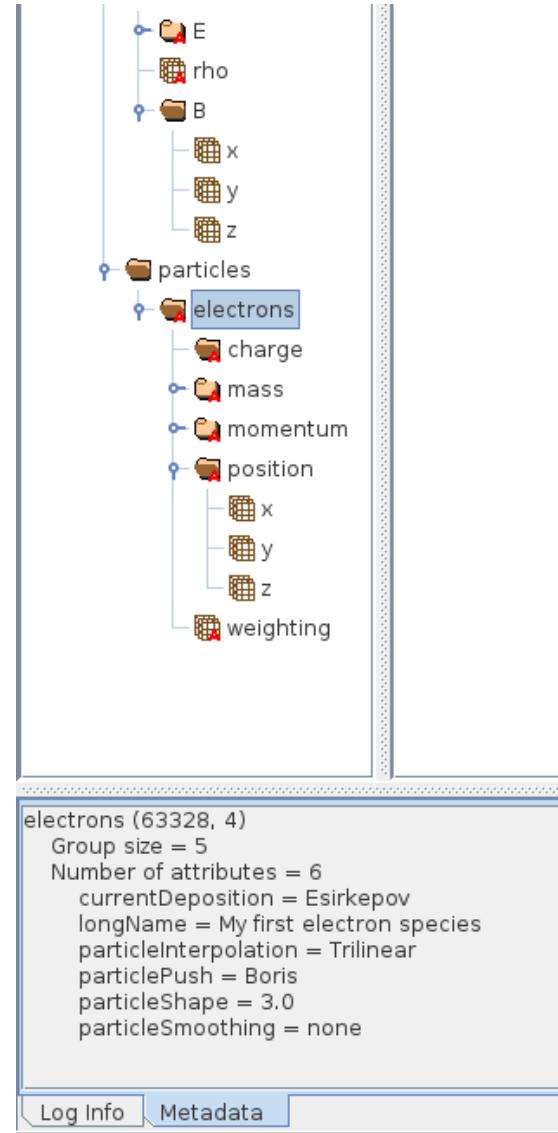
## Extensions: e.g. ED-PIC

**Field image** → field solver, smoothing



similar:

**Emittance** → particle push, field solver, shape



The screenshot shows a software interface for managing simulation data. On the left is a tree view of data categories and their sub-components. On the right, a specific configuration for an electron species is displayed.

```

  E
  rho
  B
  x
  y
  z
  particles
  electrons
  charge
  mass
  momentum
  position
  x
  y
  z
  weighting
  
```

electrons (63328, 4)  
 Group size = 5  
 Number of attributes = 6  
 currentDeposition = Esirkepov  
 longName = My first electron species  
 particleInterpolation = Trilinear  
 particlePush = Boris  
 particleShape = 3.0  
 particleSmoothing = none

Log Info   Metadata

# Modeling Mesh data

```

string      /data/1000/fields/E/axisLabels           attr   = {"z", "y", "x"}
string      /data/1000/fields/E/fieldSmoothing     attr   = "none"
string      /data/1000/fields/E/geometry            attr   = "cartesian"
double     /data/1000/fields/E/gridGlobalOffset    attr   = {0, 0, 0}
float      /data/1000/fields/E/gridSpacing        attr   = {4.25234, 1.06309, 4.25234}
double     /data/1000/fields/E/gridUnitSI          attr   = 4.16712e-08
float      /data/1000/fields/E/timeOffset          attr   = 0
double     /data/1000/fields/E/unitDimension       attr   = {1, 1, -3, -1, 0, 0, 0}
float     /data/1000/fields/E/x                  {192, 1536, 192} = 0 / 0
float      /data/1000/fields/E/x/position         attr   = {0.5, 0, 0}
double     /data/1000/fields/E/x/unitSI          attr   = 1.22627e+13
float     /data/1000/fields/E/y                  {192, 1536, 192} = 0 / 0
float      /data/1000/fields/E/y/position         attr   = {0, 0.5, 0}
double     /data/1000/fields/E/y/unitSI          attr   = 1.22627e+13
float     /data/1000/fields/E/z                  {192, 1536, 192} = 0 / 0
float      /data/1000/fields/E/z/position         attr   = {0, 0, 0.5}
double     /data/1000/fields/E/z/unitSI          attr   = 1.22627e+13

float     /data/1000/fields/e_all_energyDensity   {192, 1536, 192} = 0 / 0
string      /data/1000/fields/e_all_energyDensity/axisLabels attr   = {"z", "y", "x"}
string      /data/1000/fields/e_all_energyDensity/fieldSmoothing attr   = "none"
string      /data/1000/fields/e_all_energyDensity/geometry      attr   = "cartesian"
double     /data/1000/fields/e_all_energyDensity/gridGlobalOffset attr   = {0, 0, 0}
float      /data/1000/fields/e_all_energyDensity/gridSpacing   attr   = {4.25234, 1.06309, 4.25234}
double     /data/1000/fields/e_all_energyDensity/gridUnitSI    attr   = 4.16712e-08
float      /data/1000/fields/e_all_energyDensity/position      attr   = {0, 0, 0}
float      /data/1000/fields/e_all_energyDensity/timeOffset     attr   = 0
double     /data/1000/fields/e_all_energyDensity/unitDimension  attr   = {-1, 1, -2, 0, 0, 0, 0}
double     /data/1000/fields/e_all_energyDensity/unitSI         attr   = 7.8691e+12
  
```

Mesh attributes,  
including fieldSmoothing  
defined by ED-PIC extension

x/y/z components for vector-type field (struct-of-array)  
including component-specific metadata

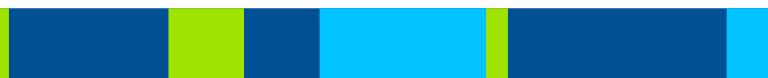
Scalar-type field  
x/y/z layer is skipped

# Hands-On: **openPMD-api: metadata**

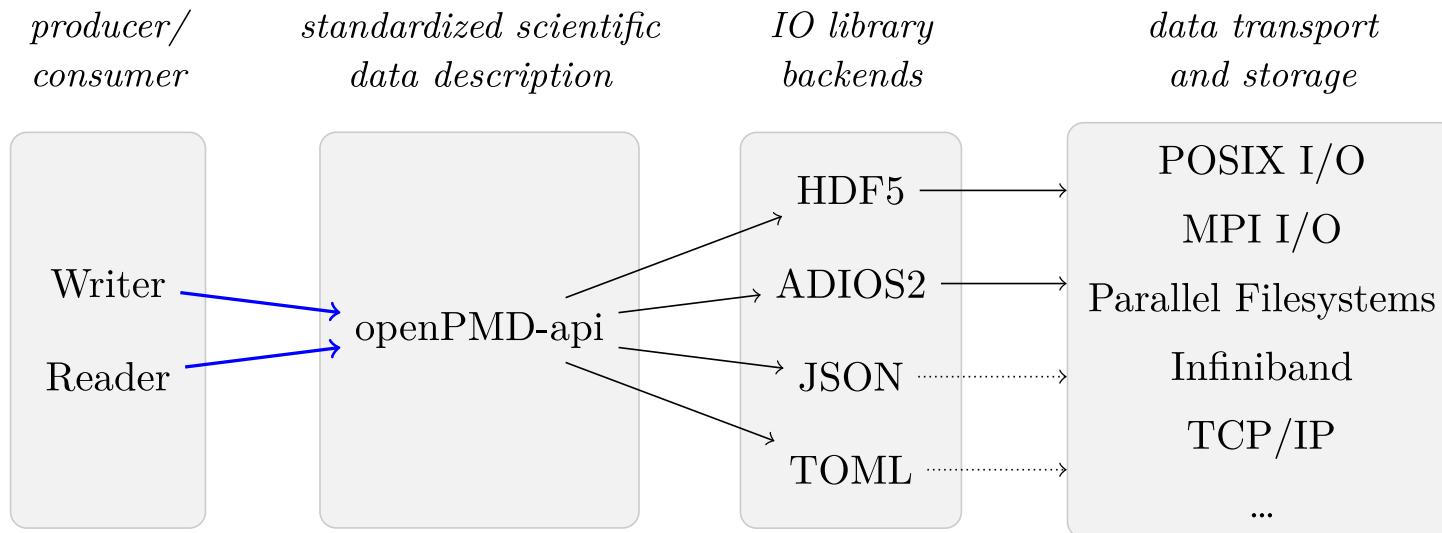
**Module environment** at `/project/project_465001310/workshop_software/env.sh`

**Materials** at [https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024\\_workshop](https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024_workshop)

Read the TODO comments inside `src/openPMDOutput.hpp`:



# openPMD-api – open stack for scientific I/O



```
import openpmd_api as io

# pick and configure backend via JSON/TOML or inferred from filename extension
adios_config = """
  backend = "adios2"
  [[adios2.dataset.operators]]
  type = "blosc" # activate compression
"""

mode = io.Access.create
series = io.Series("simOutput.h5", mode
                  """{"hdf5": {"vfd": {"type": "subfiling"}}}""")
series = io.Series("simOutput.bp5", mode, adios_config)
series = io.Series("simOutput.sst", mode, "@./or/load/config/from/file.json")
series = io.Series("simOutput.json", mode)
```

- MPI support at all levels
- Implemented in C++17
- Bindings in C++17, Python and (dev version only) Julia
- Specify backend at runtime: I/O library, transport, compression, streaming, aggregation, ...

# Hands-On: **openPMD-api:** visualization, backend configuration

**Module environment** at `/project/project_465001310/workshop_software/env.sh`

**Materials** at [https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024\\_workshop](https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024_workshop)

**Launch Notebook** at <https://www.lumi.csc.fi/>

Read the instructions inside `src/next_steps.md`:



# Compute Performance Outpaces Storage Performance



	<b>Titan</b>
<b>Peak Performance:</b>	27 Pflop/s
<b>FS Throughput:</b>	1 TiByte/s
<b>FS Capacity:</b>	27 PiByte

	<b>Summit</b>
200 Pflop/s	
2.5 TiByte/s	
250 PiByte	

	<b>Frontier</b>
1.6 Eflop/s	
5~10 TiByte/s	
500~1000 PiByte	

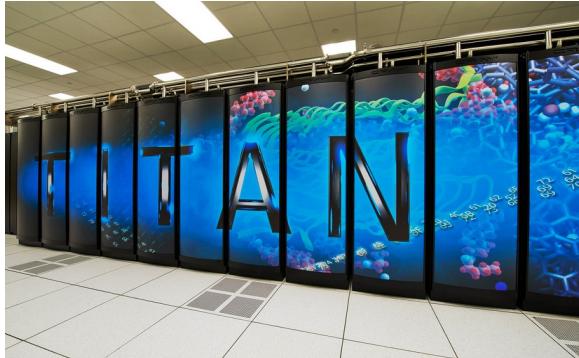
<b>Growth Factor</b>
~60
5~10
18~37

- **parallel bandwidth** insufficient for HPC at full scale
- **filesystem capacity** insufficient for HPC at full scale

Same trend in **experiments?**

- Increasing **camera resolutions and data rates**

# Compute Performance Outpaces Storage Performance



	<b>Titan</b>
<b>Peak Performance:</b>	27 Pflop/s
<b>FS Throughput:</b>	1 TiByte/s
<b>FS Capacity:</b>	27 PiByte

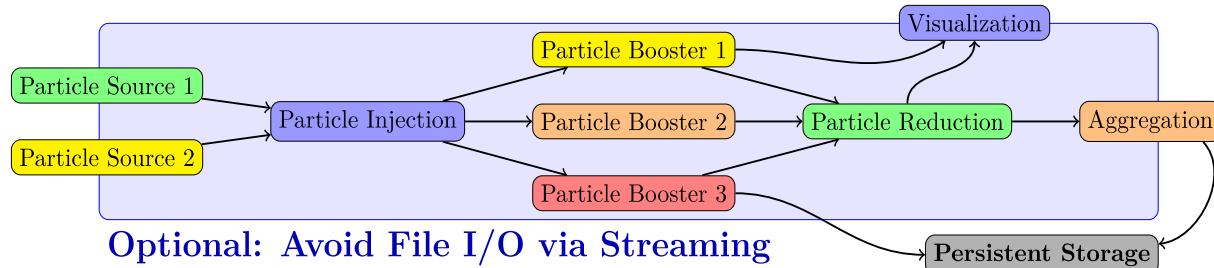
	<b>Summit</b>
200 Pflop/s	
2.5 TiByte/s	
250 PiByte	

	<b>Frontier</b>
1.6 Eflop/s	
5~10 TiByte/s	
500~1000 PiByte	

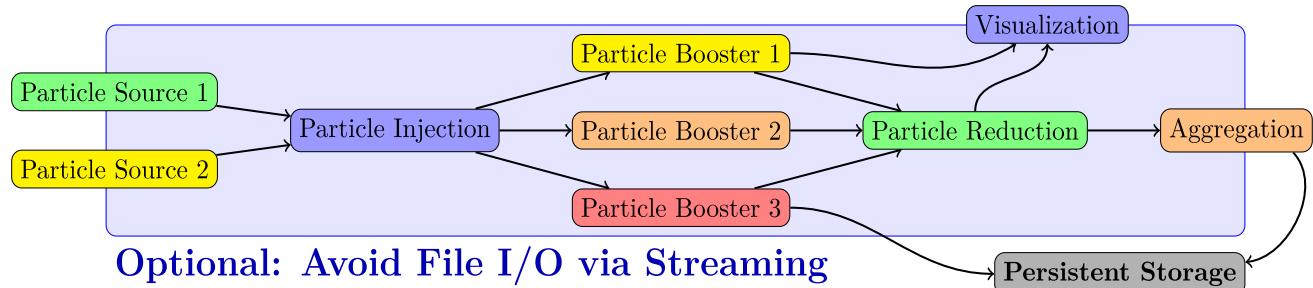
<b>Growth Factor</b>
~60
5~10
18~37

## Why does this concern us?

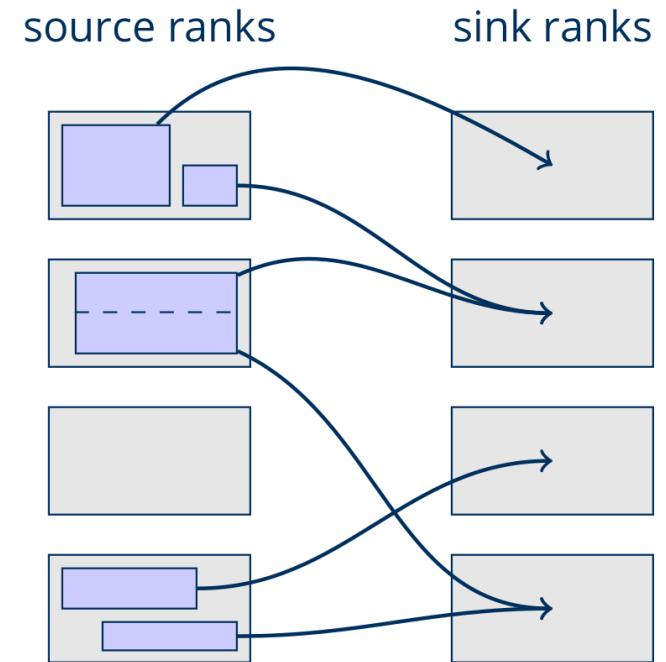
- Heterogeneous data processing pipelines traditionally have large I/O usage
- Scalable alternative: Streaming



# Streaming: Don't touch the Filesystem at all

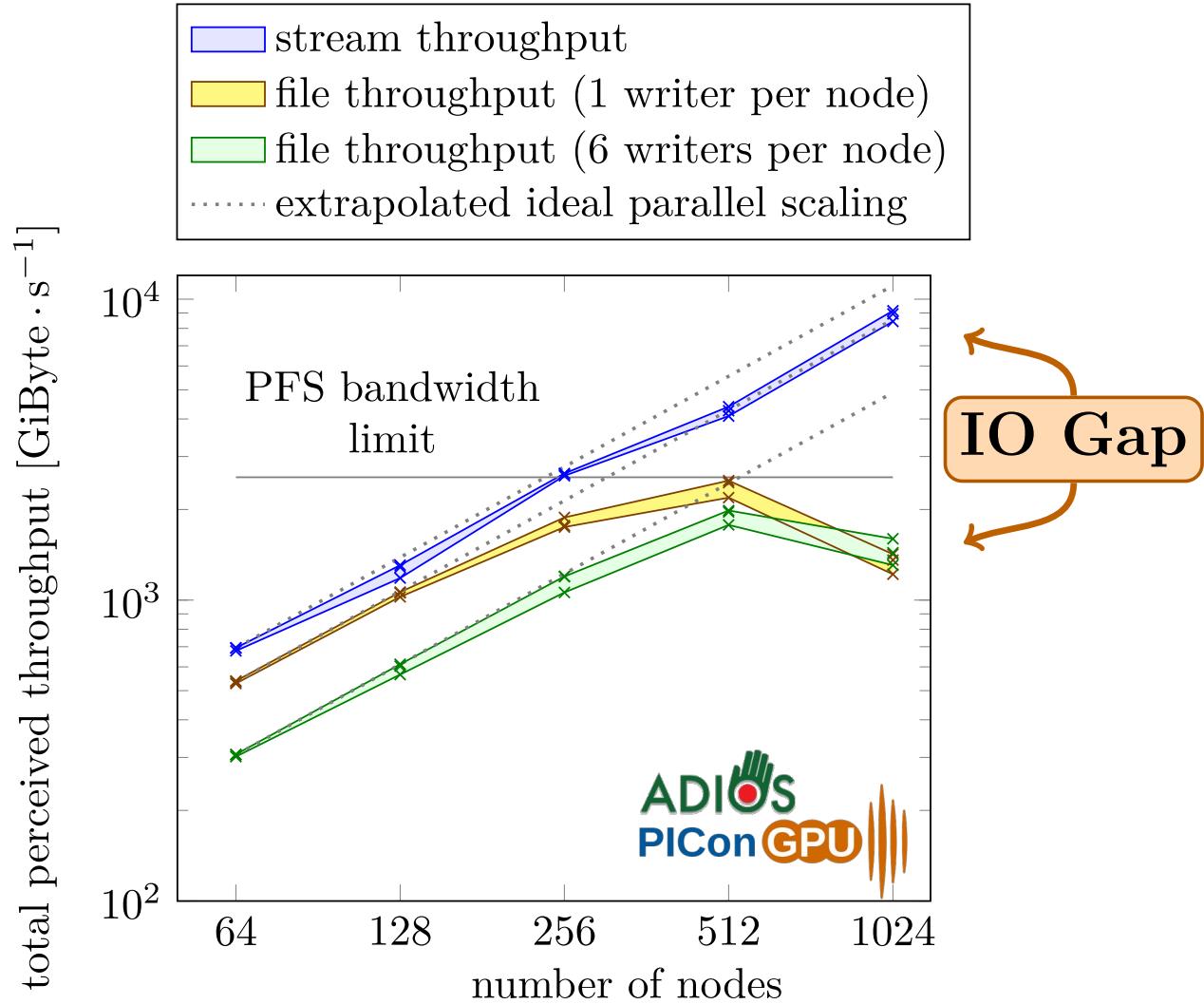


- Data processing pipelines and increasingly experiments setups have large I/O usage
- Scalable alternative: **Streaming**  
e.g. via Infiniband (on HPC systems)  
or wide area networks (in lab settings)



**Challenge:**  
Compute a balanced, aligned, local mapping between two applications that remains useful in the problem domain

# Break through Filesystem Bandwidth with Streaming

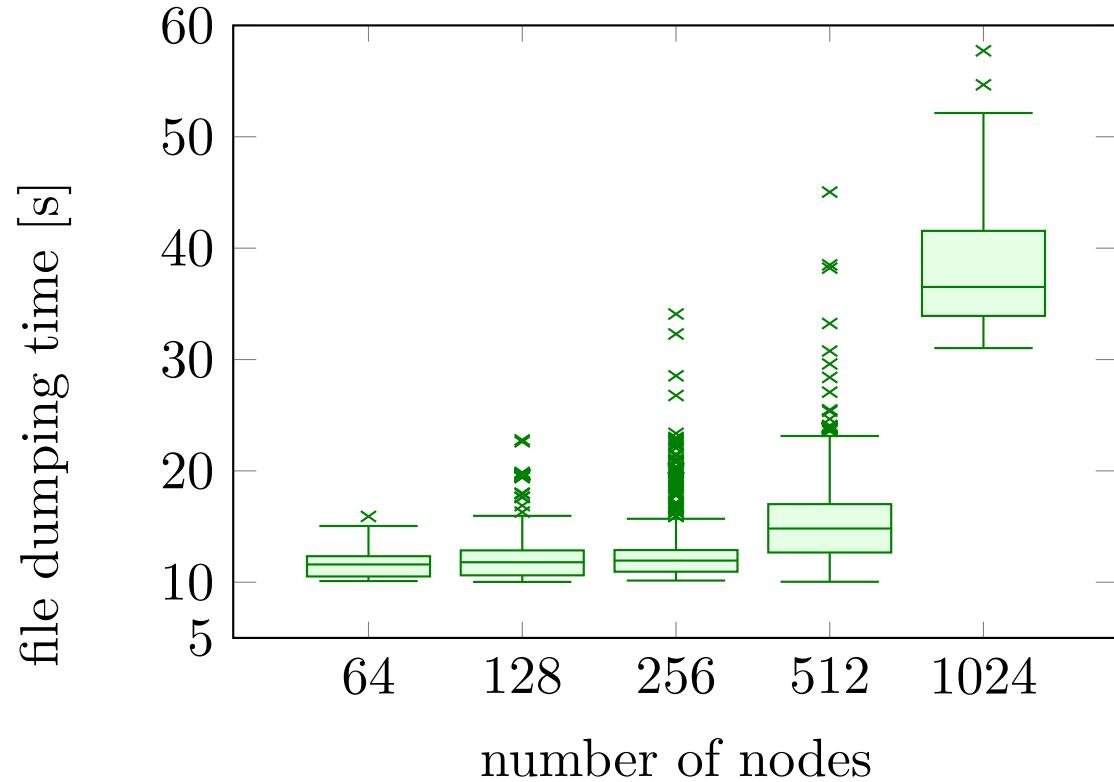


**Memory-bound simulations  
reach the I/O system limits  
at a fraction of full scale**

- Summit FS bandwidth (2.5TiByte/s) reached at 512 nodes (~11% of system size)
- Streaming workflows unaffected by filesystem bandwidth, use Infiniband hardware to scale beyond it

(benchmarks at 1024 nodes done after Summit system upgrade)

# Summit: Performance fluctuations on single ranks

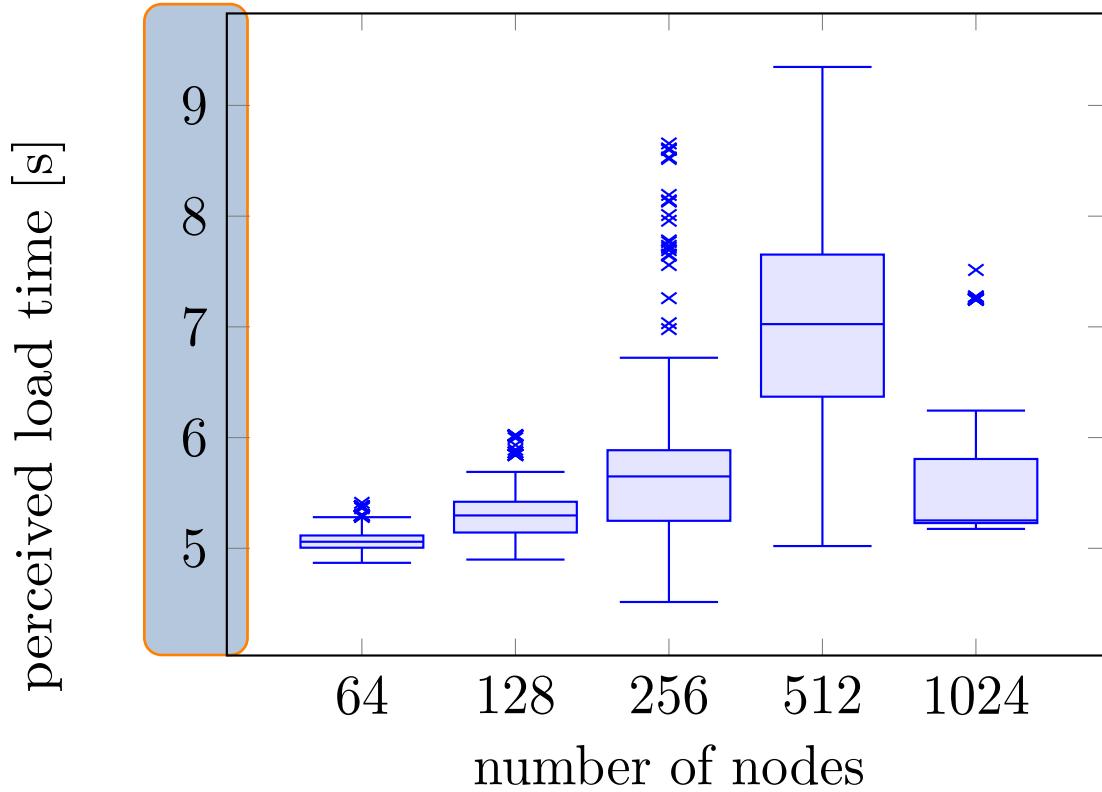


## Same results, different display:

- Plot every single measurement
- Visualize reproducibility
- Box: 50% of measurement points
- Whiskers: “normal” measurements
- Others: outliers

## Evaluation for file-only setup:

- Median time slightly raised at 512 nodes  
Scaling stops due to PFS limit after that
- Outliers increasing with scale
- Outliers fatal in parallel contexts



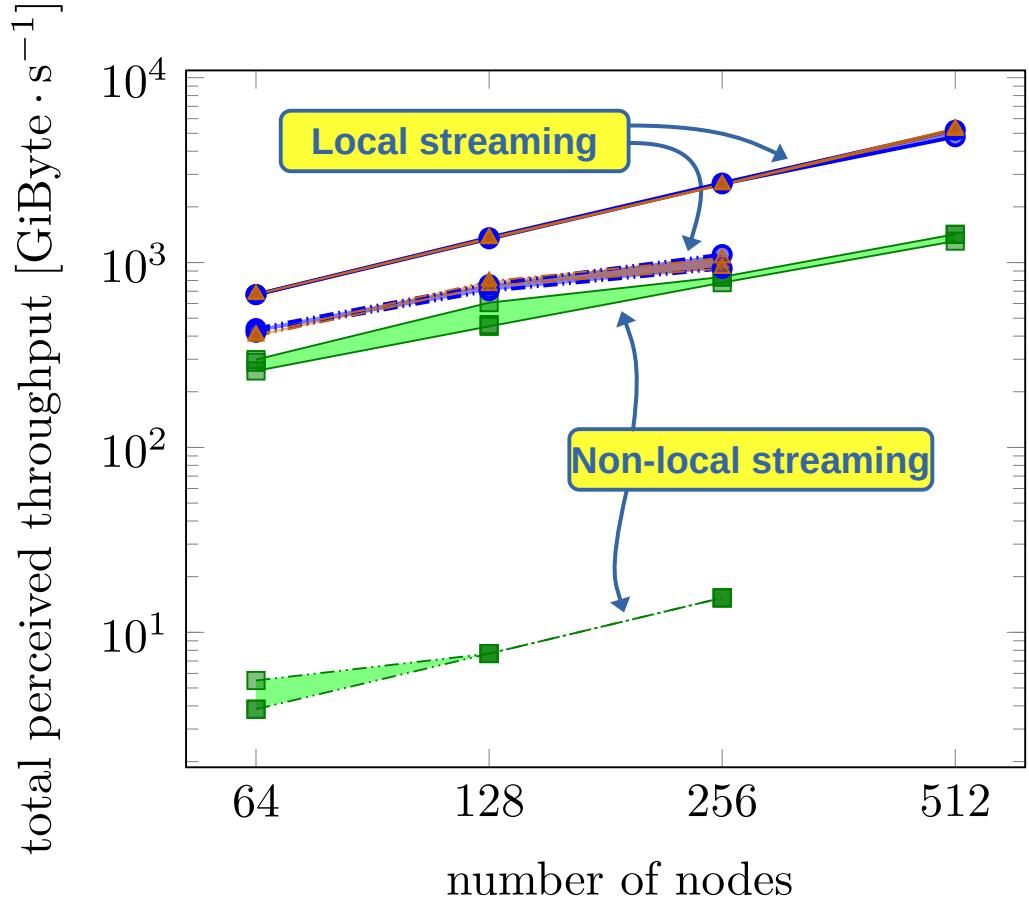
## Same results, different display:

- Plot every single measurement
- Visualize reproducibility
- Box: 50% of measurement points
- Whiskers: “normal” measurements
- Others: outliers

## stream+file setup (stream part):

- Overall times are lower
- Median between 5 and 7 seconds
- Outliers less dominating by far

# For good throughput: Local streaming patterns, Infiniband/RDMA



## Local streaming:

Distribute data chunks  
only within a node

## Non-local streaming:

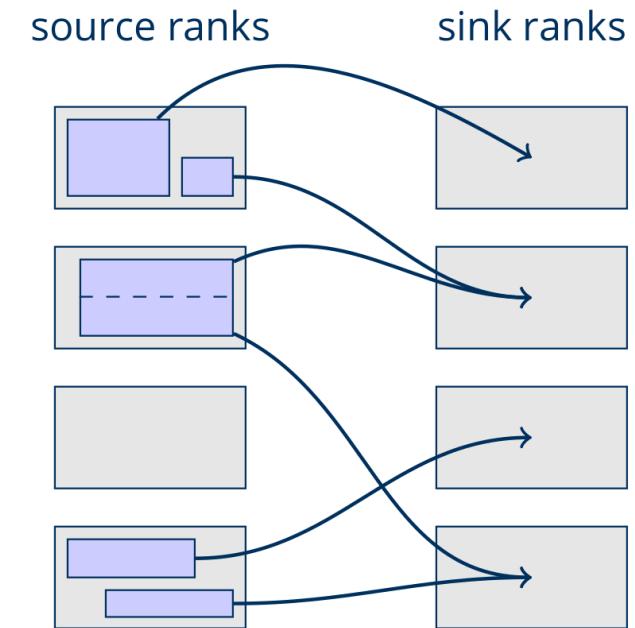
Distribute data chunks  
globally, optimize for  
balance and alignment

## Straight lines:

Infiniband/RDMA

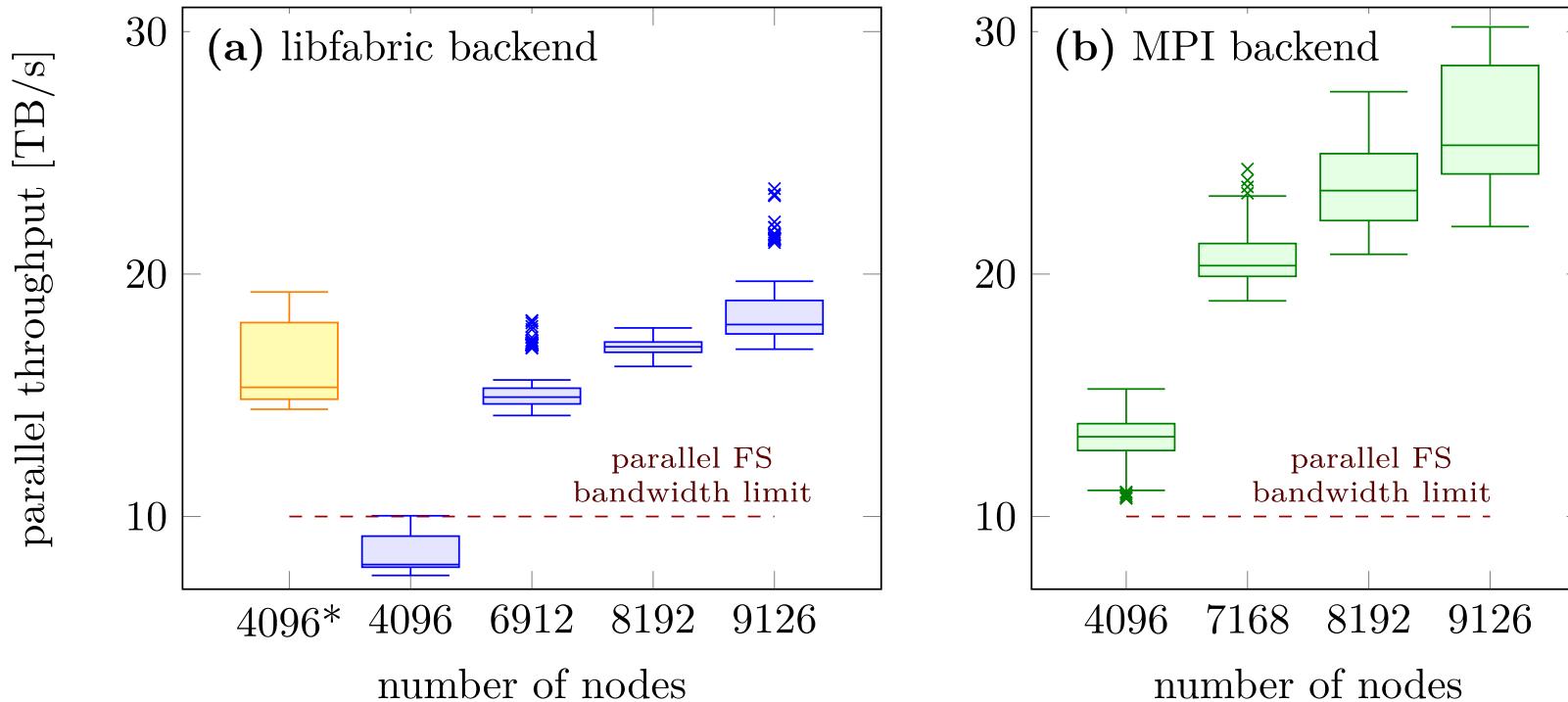
## Dashed lines:

TCP/sockets



Setup: Couple PIConGPU with a scattering code (GAPD)  
exchange particle data only

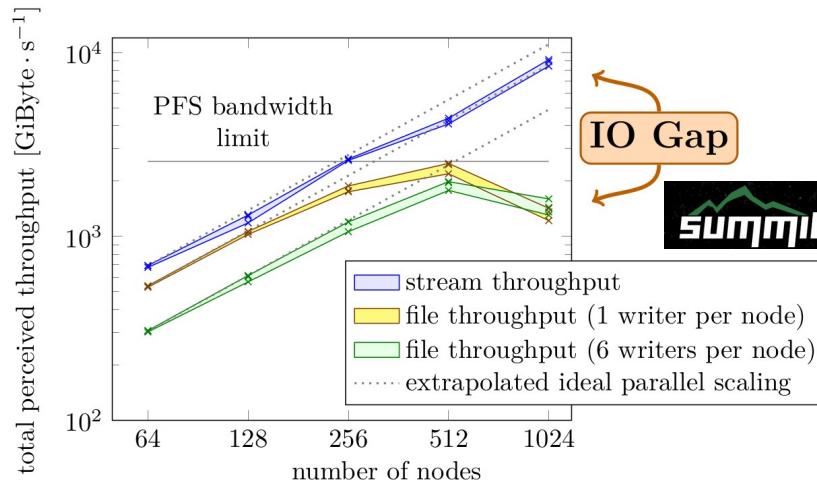
# Break through Filesystem Bandwidth with Streaming



**Full scale of Frontier:**  
 Leaving behind us  
 the theoretical bandwidth  
 of the parallel filesystem



# Performance: Data Layouts and no-file I/O



## Streaming Data Pipelines:

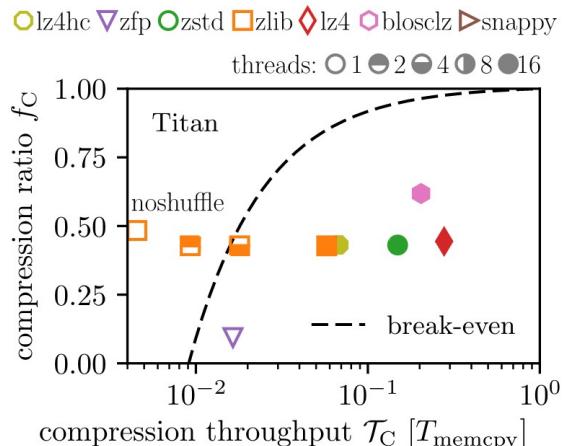
[DOI:10.1007/978-3-030-96498-6\\_6](https://doi.org/10.1007/978-3-030-96498-6_6)

by F Poeschel, A Huebl et al., SMC21 (2022)

## Online Data Layout Reorganization:

[DOI:10.1109/TPDS.2021.3100784](https://doi.org/10.1109/TPDS.2021.3100784)

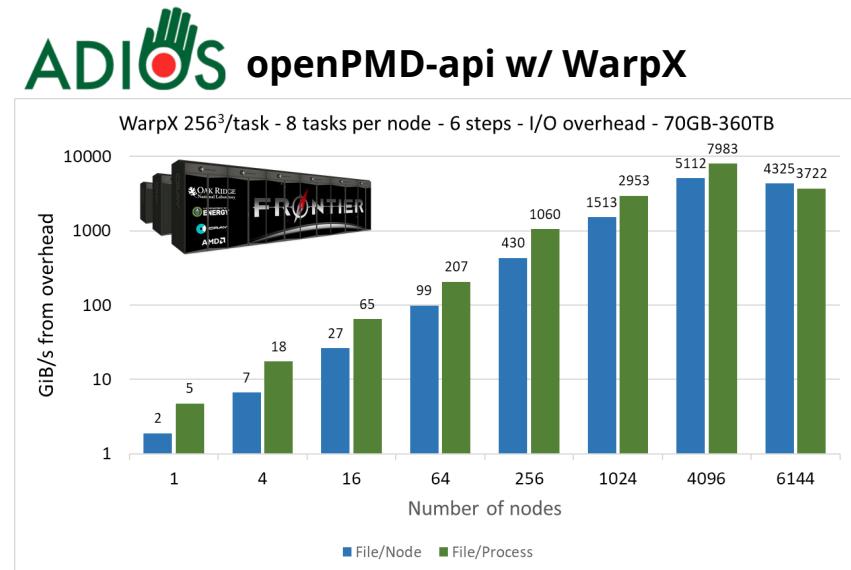
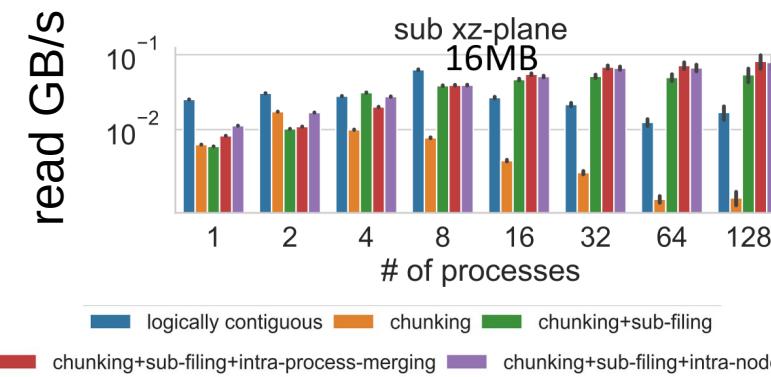
by L Wan, A Huebl et al., TPDS (2021)



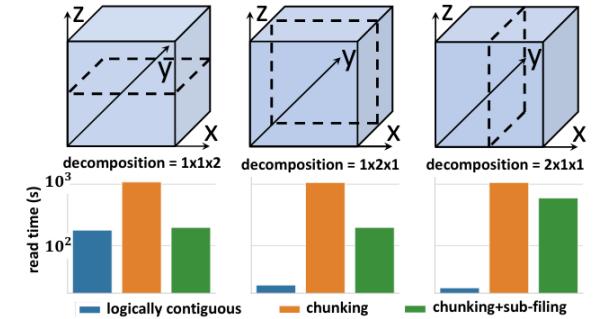
## Fast Compressors Needed:

[DOI:10.1007/978-3-319-67630-2\\_2](https://doi.org/10.1007/978-3-319-67630-2_2)

by A Huebl et al., ISC DRBSD-1 (2017)



>5.5TB/s FS BW: two-tier lustre w/ high-performance storage & progressive files



Impact of decomposition schemes when reading

# Hands-On: **openPMD-api:** streaming I/O

**Module environment** at `/project/project_465001310/workshop_software/env.sh`

**Materials** at [https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024\\_workshop](https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024_workshop)

**Launch Notebook** at <https://www.lumi.csc.fi/>

Read the instructions inside `src/visualize.py` (open as a Jupyter Notebook):



# openPMD powered Projects and Users

## Documents:

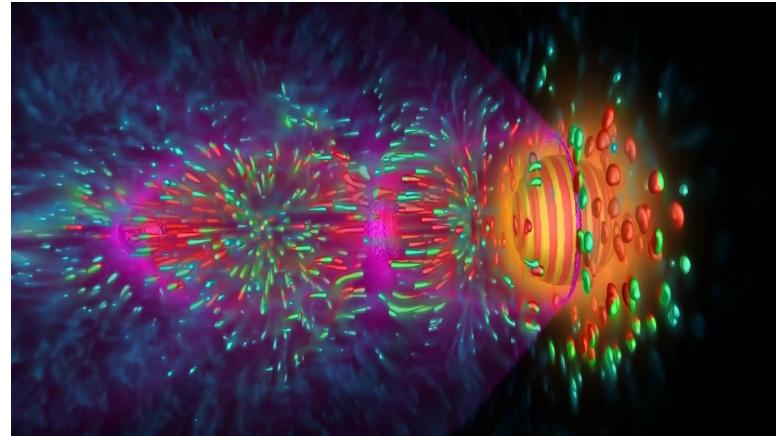
- **openPMD standard** (1.0.0, 1.0.1, 1.1.0)  
*the underlying file markup and definition*  
 A Huebl et al., doi: 10.5281/zenodo.33624

## Scientific Simulations:

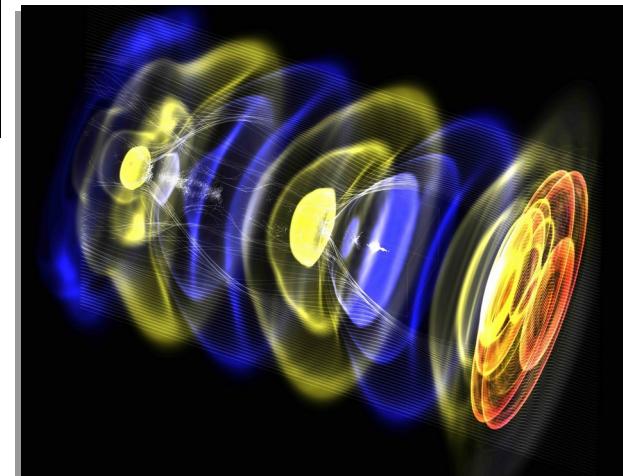
- **PIConGPU** (HZDR)  
*electro-dynamic particle-in-cell code*  
 maintainers: R Widera, S Bastrakov, A Debus et al.
- **WarpX** (LBNL, LLNL)  
*electro-dynamic/static particle-in-cell code*  
 maintainers: JL Vay, D Grote, R Lehe, A Huebl et al.
- **FBPIC** (LBNL, DESY)  
*spectral, fourier-bessel particle-in-cell code*  
 maintainers: R Lehe, M Kirchen et al.
- **SimEx Platform** (EUCALL, European XFEL)  
*simulation of advanced photon experiments*  
 maintainer: C Fortmann-Grote

## Language Binding:

- **openPMD-api** (HZDR, CASUS, LBNL)  
*reference API for openPMD data handling*  
 maintainers: A Huebl, J Gu, F Poeschel et al.



**WarpX**  
 PI: Jean-Luc Vay/LBNL



see also: <https://github.com/openPMD/openPMD-projects>



# openPMD powered Projects and Users

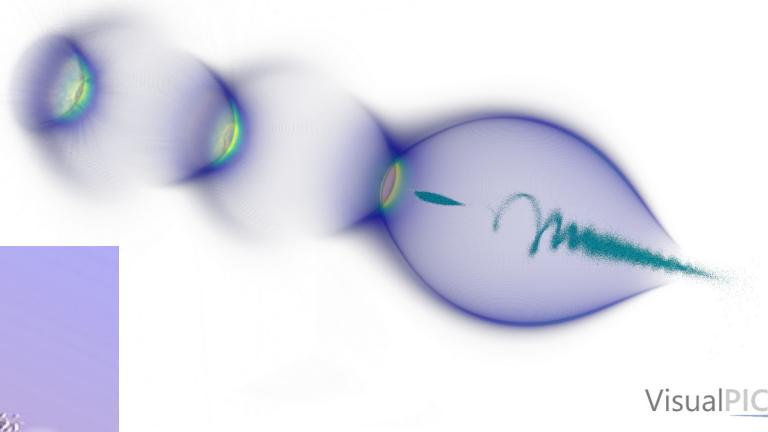
## Documents:

- **openPMD standard** (1.0.0, 1.0.1, 1.1.0)  
*the underlying file markup and definition*  
 A Huebl et al., doi: 10.5281/zenodo.33624

## Language Binding:

- **openPMD-api** (HZDR, CASUS, LBNL)  
*reference API for openPMD data handling*  
 maintainers: A Huebl, J Gu, F Poeschel et al.

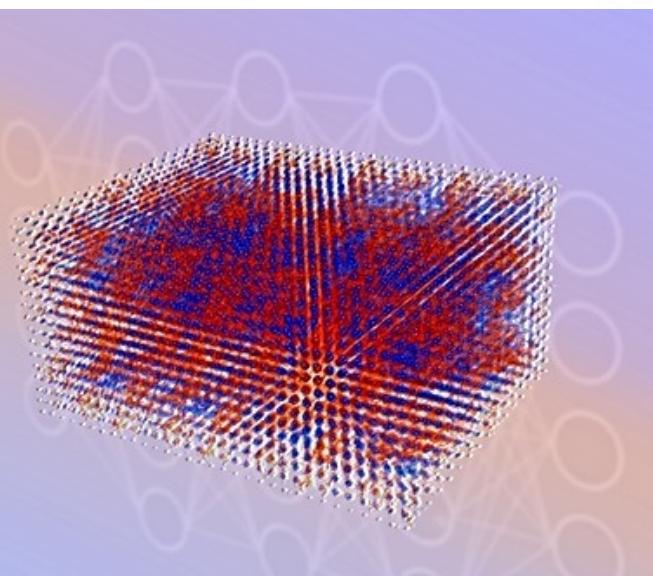
**HiPACE++** → VisualPIC  
 Credit: M.Thévenet & A. Ferran Pousa (DESY)



VisualPIC

- **Wake-T** (DESY)  
*fast particle-tracking code for plasma-based accelerators*  
 maintainer: A Ferran Pousa
- **HiPACE++** (DESY, LBNL)  
*3D GPU-capable quasi-static PIC code for plasma accel.*  
 maintainers: M Thevenet, S Diederichs, A Huebl
- **Bmad** (Cornell)  
*library for charged-particle dynamics simulations*  
 maintainers: D Sagan et al.
- **MALA** (CASUS, SNL)  
*ML models that replace DFT calculations in materials science*  
 maintainers: Attila Cangi & Sivasankaran Rajamanickam
- and more...

see also: <https://github.com/openPMD/openPMD-projects>



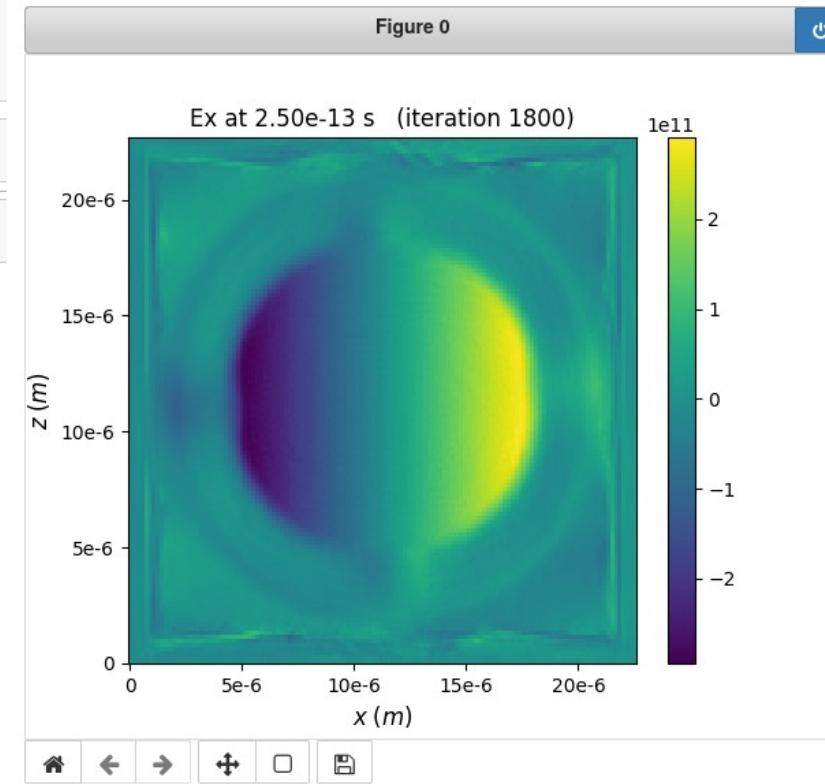
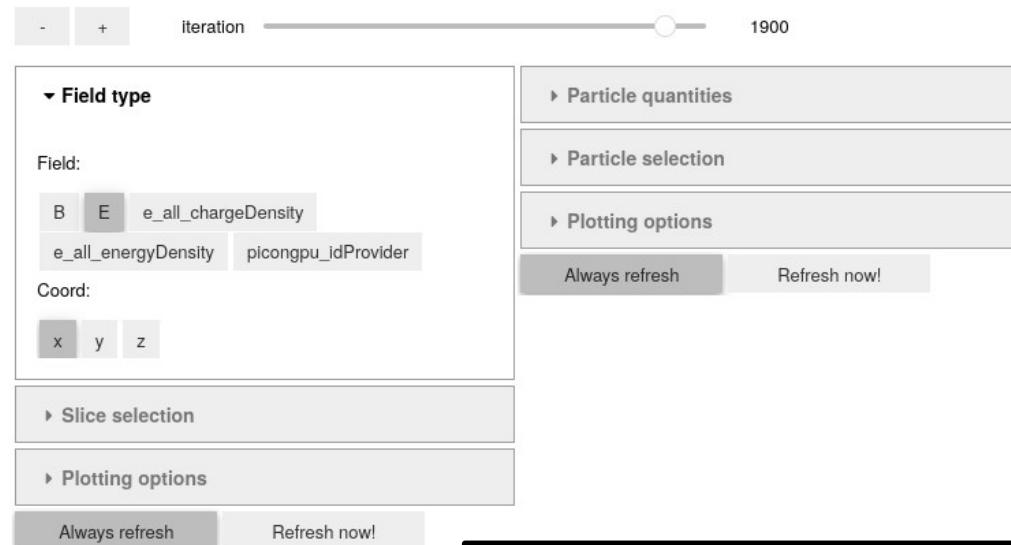
**MALA** → ParaView  
 Credit: A. Cangi (CASUS)

# Analysis and Visualization

```
In [1]: import numpy as np
%matplotlib notebook
# or '%matplotlib inline' for non-interactive plots
# or '%matplotlib widget' when using JupyterLab (github.com/matplotlib/jupyter-matplotlib)
import matplotlib.pyplot as plt
from openpmd_viewer import OpenPMDTimeSeries
```

```
In [2]: # Replace the string below, to point to your data
ts = OpenPMDTimeSeries('/home/franzpoeschel/singularity_build/pic_run/openPMD')
```

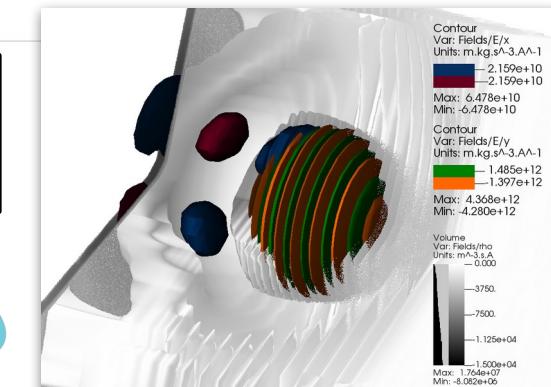
```
In [3]: # Interactive GUI
ts.slider()
```



openPMD/openPMD-viewer



VI-SIT



**Standardization of data**  
 → integration into modern scientific compute workflows

**RAPIDS**

**pandas**

**open  
PMD**



**DASK**

**/// ParaView**



Hands-On:  
**openPMD-viewer:**  
visualizing data written by  
a PIConGPU LaserWakefield simulation

**Module environment** at `/project/project_465001310/workshop_software/env.sh`

**Materials** at [https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024\\_workshop](https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024_workshop)

**Launch Notebook** at <https://www.lumi.csc.fi/>

Read the instructions inside `next_steps.md`:



# Comparing HDF5 and ADIOS2 files

- HDF5's output is structured
- Simplest case: An n-dimensional dataset is represented as contiguous buffer
- For most advanced features:  
Divide the dataset into **regular chunks**

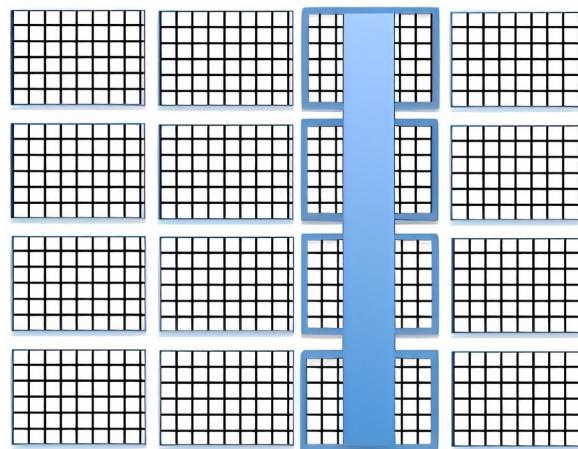
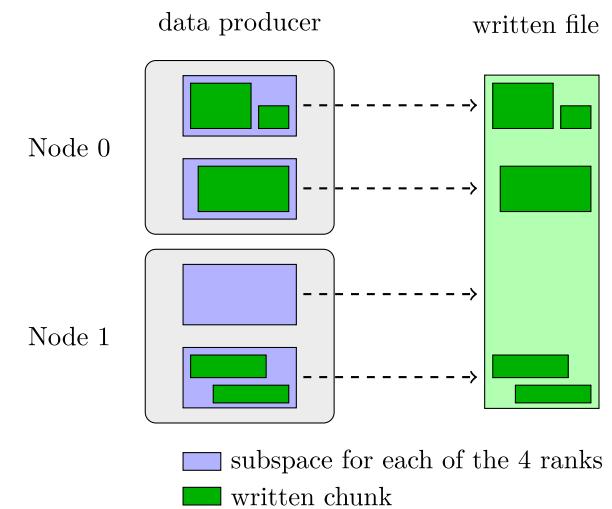


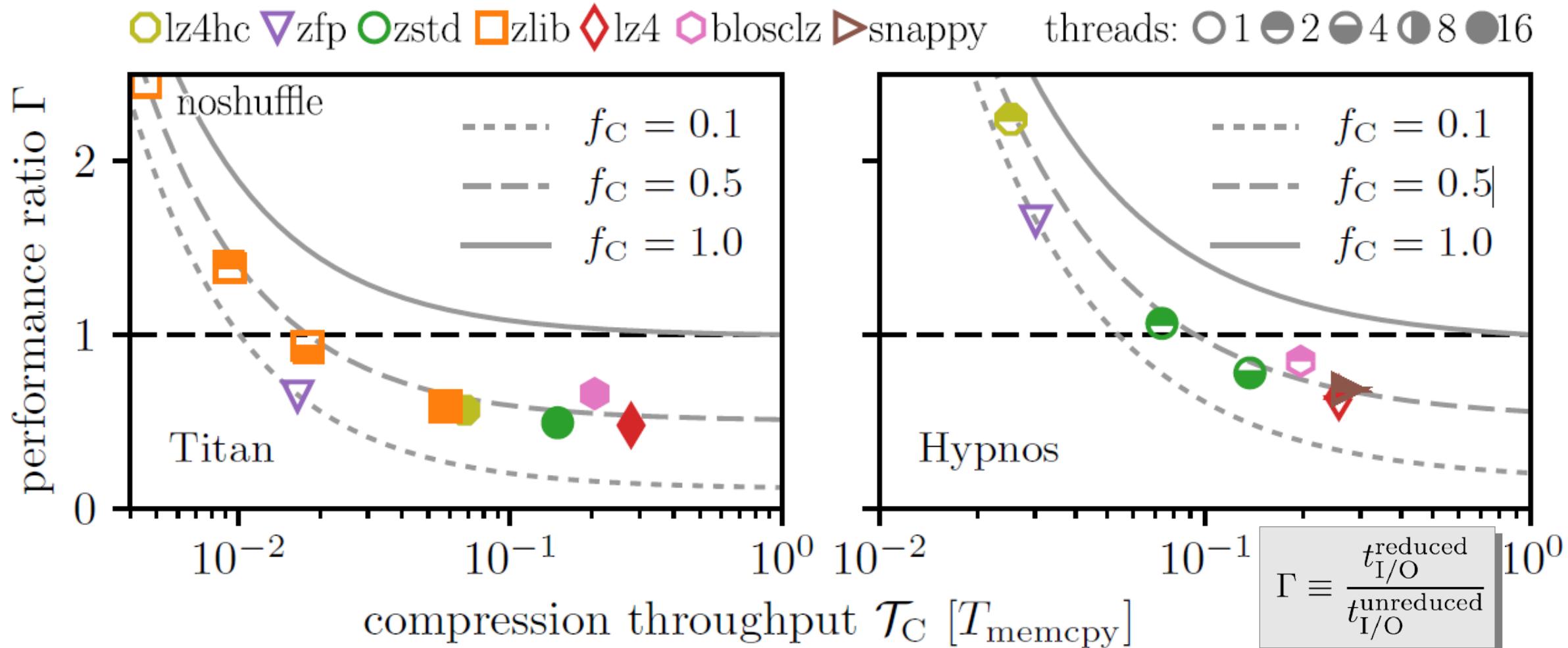
image from  
<https://www.hdfgroup.org/2017/05/24/hdf5-data-compression-demystified-2-performance-tuning/>

- openPMD includes heuristics for setting the size automatically

- ADIOS2's output is log-formatted (log as in logbook, not logarithm)
- → Just write blocks as they drop out of the data writer
- Very **efficient for writing**, read performance application-dependent (e.g.: PICoGPU output is very clean)
- Can handle **sparse datasets** and **overlapping regions**



Just compress the data and everything will be fine ...?



# Hands-On: **compression**

**Module environment** at `/project/project_465001310/workshop_software/env.sh`

**Materials** at [https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024\\_workshop](https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024_workshop)

**Launch Notebook** at <https://www.lumi.csc.fi/>

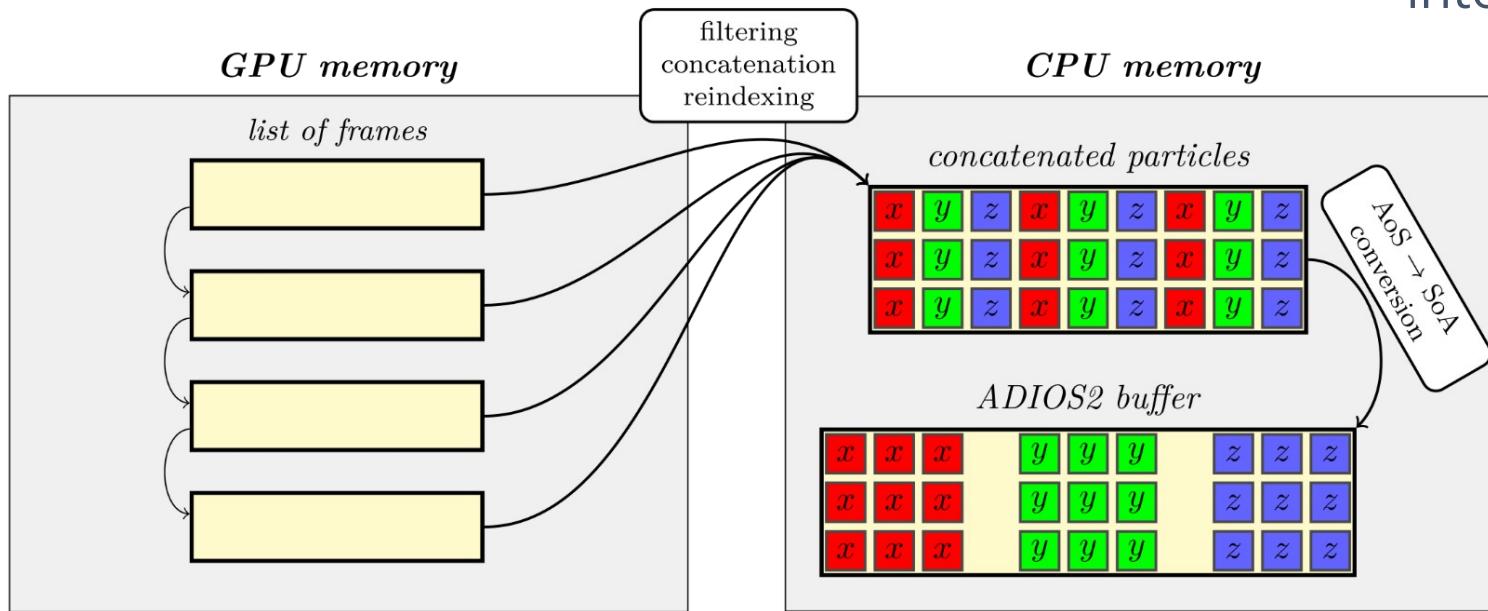
Continue with the instructions in the openPMD-viewer Notebook



# Data path in a realistic application

- In real-world applications, data is often kept in a custom, algorithm-driven format
- Reorganization needed before output

- However: ADIOS2 (optionally) buffers data before output for I/O performance
- Few big operations better than many small operations
- Elide one data copy by writing directly into those internal buffers



**Example:**  
 Transforming  
 PICoGPU particle data  
 for output

# Advanced Hands-On: **openPMD-api:** Span API

**Module environment** at `/project/project_465001310/workshop_software/env.sh`

**Materials** at [https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024\\_workshop](https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024_workshop)

**Launch Notebook** at <https://www.lumi.csc.fi/>

Read the instructions inside `src/openPMDOutput.hpp`



# Modeling particle data

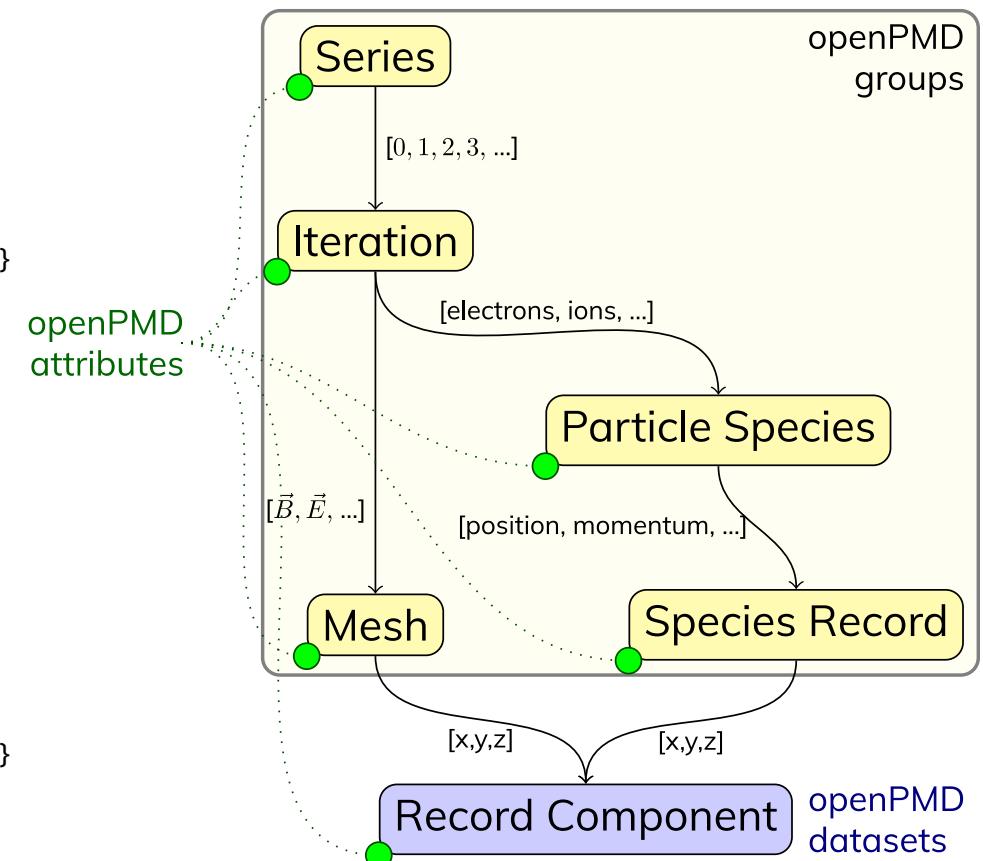
```

uint32_t /data/particles/e/position/macroWeighted      attr   = 0
float    /data/particles/e/position/timeOffset        attr   = 0
double   /data/particles/e/position/unitDimension     attr   = {1, 0, 0, 0, 0, 0, 0}
double   /data/particles/e/position/weightingPower    attr   = 0
float   /data/particles/e/position/x                 {109134176} = 0 / 0
double   /data/particles/e/position/x/unitSI          attr   = 1.772e-07
float   /data/particles/e/position/y                 {109134176} = 0 / 0
double   /data/particles/e/position/y/unitSI          attr   = 4.43e-08
float   /data/particles/e/position/z                 {109134176} = 0 / 0
double   /data/particles/e/position/z/unitSI          attr   = 1.772e-07

uint32_t /data/particles/e/positionOffset/macroWeighted attr   = 0
float    /data/particles/e/positionOffset/timeOffset    attr   = 0
double   /data/particles/e/positionOffset/unitDimension attr   = {1, 0, 0, 0, 0, 0, 0}
double   /data/particles/e/positionOffset/weightingPower attr   = 0
int32_t /data/particles/e/positionOffset/x           {109134176} = 0 / 0
double   /data/particles/e/positionOffset/x/unitSI     attr   = 1.772e-07
int32_t /data/particles/e/positionOffset/y           {109134176} = 0 / 0
double   /data/particles/e/positionOffset/y/unitSI     attr   = 4.43e-08
int32_t /data/particles/e/positionOffset/z           {109134176} = 0 / 0
double   /data/particles/e/positionOffset/z/unitSI     attr   = 1.772e-07

float   /data/particles/e/weighting                  {109134176} = 0 / 0
uint32_t /data/particles/e/weighting/macroWeighted    attr   = 1
float    /data/particles/e/weighting/timeOffset        attr   = 0
double   /data/particles/e/weighting/unitDimension     attr   = {0, 0, 0, 0, 0, 0, 0}
double   /data/particles/e/weighting/unitSI            attr   = 1
double   /data/particles/e/weighting/weightingPower    attr   = 1
  
```

Let's have a detailed look  
at the electron species  
written by PICoGPU...



# Modeling particle data: mandatory records, vector vs. scalar

```

uint32_t /data/particles/e/position/macroWeighted
float   /data/particles/e/position/timeOffset
double  /data/particles/e/position/unitDimension
double  /data/particles/e/position/weightingPower
float  /data/particles/e/position/x
double  /data/particles/e/position/x/unitSI
float  /data/particles/e/position/y
double  /data/particles/e/position/y/unitSI
float  /data/particles/e/position/z
double  /data/particles/e/position/z/unitSI
  
```

```

uint32_t /data/particles/e/positionOffset/macroWeighted
float   /data/particles/e/positionOffset/timeOffset
double  /data/particles/e/positionOffset/unitDimension
double  /data/particles/e/positionOffset/weightingPower
int32_t /data/particles/e/positionOffset/x
double  /data/particles/e/positionOffset/x/unitSI
int32_t /data/particles/e/positionOffset/y
double  /data/particles/e/positionOffset/y/unitSI
int32_t /data/particles/e/positionOffset/z
double  /data/particles/e/positionOffset/z/unitSI
  
```

```

float  /data/particles/e/weighting
uint32_t /data/particles/e/weighting/macroWeighted
float   /data/particles/e/weighting/timeOffset
double  /data/particles/e/weighting/unitDimension
double  /data/particles/e/weighting/unitSI
double  /data/particles/e/weighting/weightingPower
  
```

```

attr   = 0
attr   = 0
attr   = {1, 0, 0, 0, 0, 0, 0}
attr   = 0
{109134176} = 0 / 0
attr   = 1.772e-07
{109134176} = 0 / 0
attr   = 4.43e-08
{109134176} = 0 / 0
attr   = 1.772e-07
  
```

```

attr   = 0
attr   = 0
attr   = {1, 0, 0, 0, 0, 0, 0}
attr   = 0
{109134176} = 0 / 0
attr   = 1.772e-07
{109134176} = 0 / 0
attr   = 4.43e-08
{109134176} = 0 / 0
attr   = 1.772e-07
  
```

```

{109134176} = 0 / 0
attr   = 1
attr   = 0
attr   = {0, 0, 0, 0, 0, 0, 0}
attr   = 1
attr   = 1
  
```

**position:** mandatory record for particles

position is a **vector quantity**, i.e. has x/y/z struct-of-array-like layout:

3 vectors of same length

**positionOffset:** also mandatory specifies the base for the position used for numerical reasons

**weighting:** custom record in PICoNGPU to model macro particles scalar quantity, x/y/z is skipped

# Modeling particle data: constant components

```

string /data/particles/e/currentDeposition           attr = "Esirkepov"
string /data/particles/e/particleInterpolation       attr = "uniform"
string /data/particles/e/particlePush                attr = "Boris"
double /data/particles/e/particleShape               attr = 2
string /data/particles/e/particleSmoothing          attr = "none"

int32_t /data/particles/e/charge/macroWeighted      attr = 0
uint64_t /data/particles/e/charge/shape             attr = {109134176}
double /data/particles/e/charge/timeOffset          attr = 0
double /data/particles/e/charge/unitDimension        attr = {0, 0, 1, 1, 0, 0, 0}
double /data/particles/e/charge/unitSI              attr = 1.11432e-15
double /data/particles/e/charge/value              attr = -0.00014378
double /data/particles/e/charge/weightingPower       attr = 1

int32_t /data/particles/e/mass/macroWeighted        attr = 0
uint64_t /data/particles/e/mass/shape              attr = {109134176}
double /data/particles/e/mass/timeOffset            attr = 0
double /data/particles/e/mass/unitDimension          attr = {0, 1, 0, 0, 0, 0, 0}
double /data/particles/e/mass/unitSI                attr = 6.33564e-27
double /data/particles/e/mass/value                attr = 0.00014378
double /data/particles/e/mass/weightingPower         attr = 1

uint32_t /data/particles/e/momentum/macroWeighted attr = 1
float /data/particles/e/momentum/timeOffset         attr = 0
double /data/particles/e/momentum/unitDimension      attr = {1, 1, -1, 0, 0, 0, 0}
double /data/particles/e/momentum/weightingPower     attr = 1
float /data/particles/e/momentum/x                 attr = {109134176} = 0 / 0
double /data/particles/e/momentum/x/unitSI           attr = 1.89938e-18
float /data/particles/e/momentum/y                 attr = {109134176} = 0 / 0
double /data/particles/e/momentum/y/unitSI           attr = 1.89938e-18
float /data/particles/e/momentum/z                 attr = {109134176} = 0 / 0
double /data/particles/e/momentum/z/unitSI           attr = 1.89938e-18
  
```

particle attributes defined by the ED-PIC extension

## charge and mass:

are fixed for one particle species,  
i.e. record has one constant value

→ define via attributes shape and value instead of writing the whole array scalar quantities as well

## momentum:

again a non-constant vector quantity

# Modeling particle data: particle patches

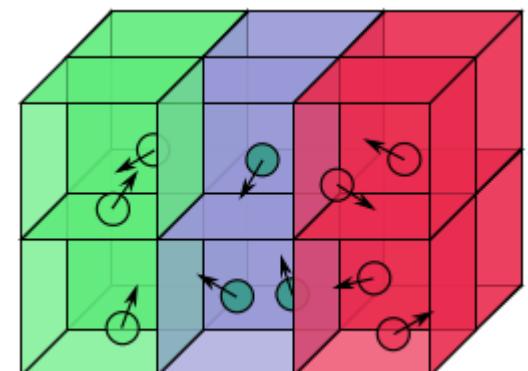
<code>uint64_t /data/1000/particles/e/particlePatches/offset/x</code>	{8}
(0) 0 0 0 96 96 96 96	
<code>uint64_t /data/1000/particles/e/particlePatches/offset/y</code>	{8}
(0) 0 512 1024 1536 0 512 1024 1536	
<code>uint64_t /data/1000/particles/e/particlePatches/offset/z</code>	{8}
(0) 0 0 0 0 0 0 0	
<code>uint64_t /data/1000/particles/e/particlePatches/extent/x</code>	{8}
(0) 96 96 96 96 96 96 96	
<code>uint64_t /data/1000/particles/e/particlePatches/extent/y</code>	{8}
(0) 512 512 512 0 512 512 512 0	
<code>uint64_t /data/1000/particles/e/particlePatches/extent/z</code>	{8}
(0) 192 192 192 192 192 192 192	
<code>uint64_t /data/1000/particles/e/particlePatches/numParticlesOffset</code>	{8}
(0) 0 16916087 35692896 54567264 54567264 71482787 90259808 109134176	
<code>uint64_t /data/1000/particles/e/particlePatches/numParticles</code>	{8}
(0) 16916087 18776809 18874368 0 16915523 18777021 18874368 0	
 <b>green:</b> GPU 0	

**Common case:** Each GPU computes one sub-area of the global simulation space

→ Particles are written in patches, each GPU's patch is spatially bounded

- offset/extent define the box wherein the particles are located
- numParticlesOffset/numParticles define the patch's position in the list

e.g.: GPU 0 computes the box (0, 0, 0)–(96, 512, 192) (in simulation units), its electron data is found from index 0 to 16916087  
**Note:** extent is always relative to the offset



# Advanced Hands-On: **openPMD-api:** Python API: constant components

**Module environment** at `/project/project_465001310/workshop_software/env.sh`

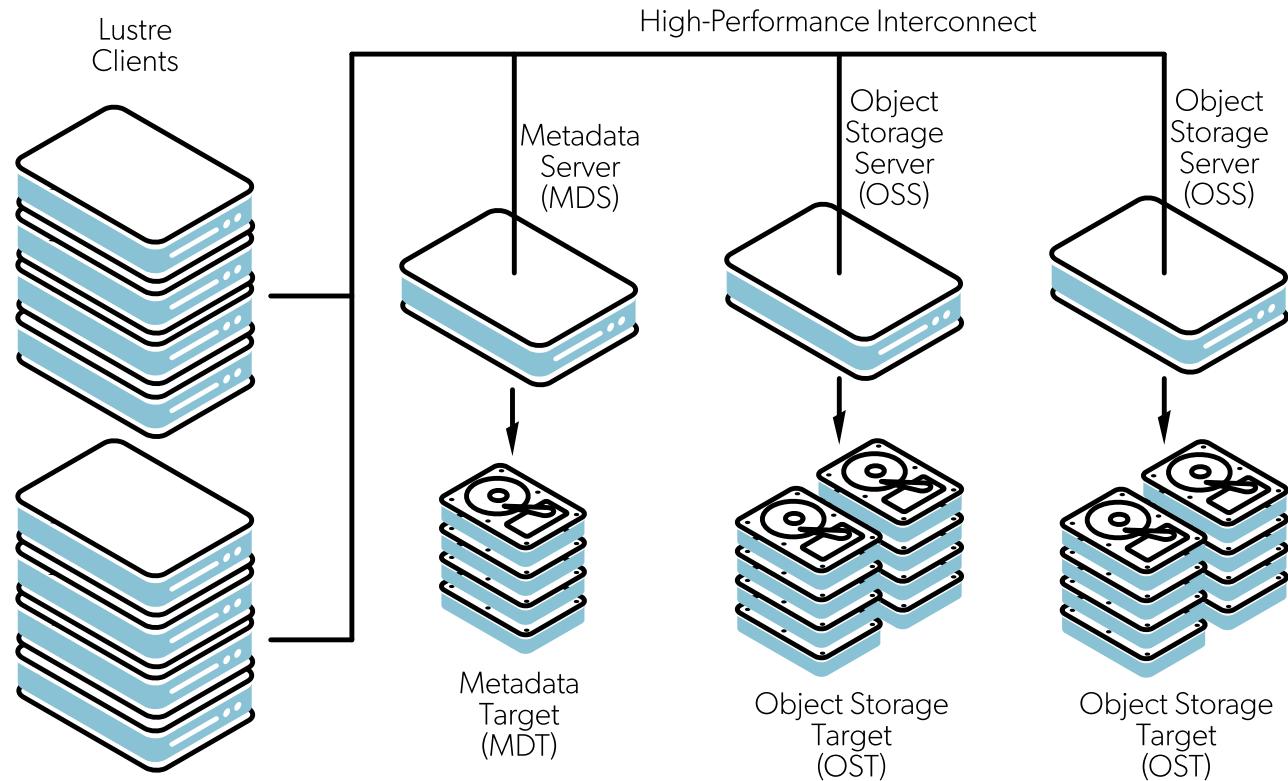
**Materials** at [https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024\\_workshop](https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024_workshop)

**Launch Notebook** at <https://www.lumi.csc.fi/>

Read the TODO comments inside `write_parallel.py`



# Using parallel filesystems



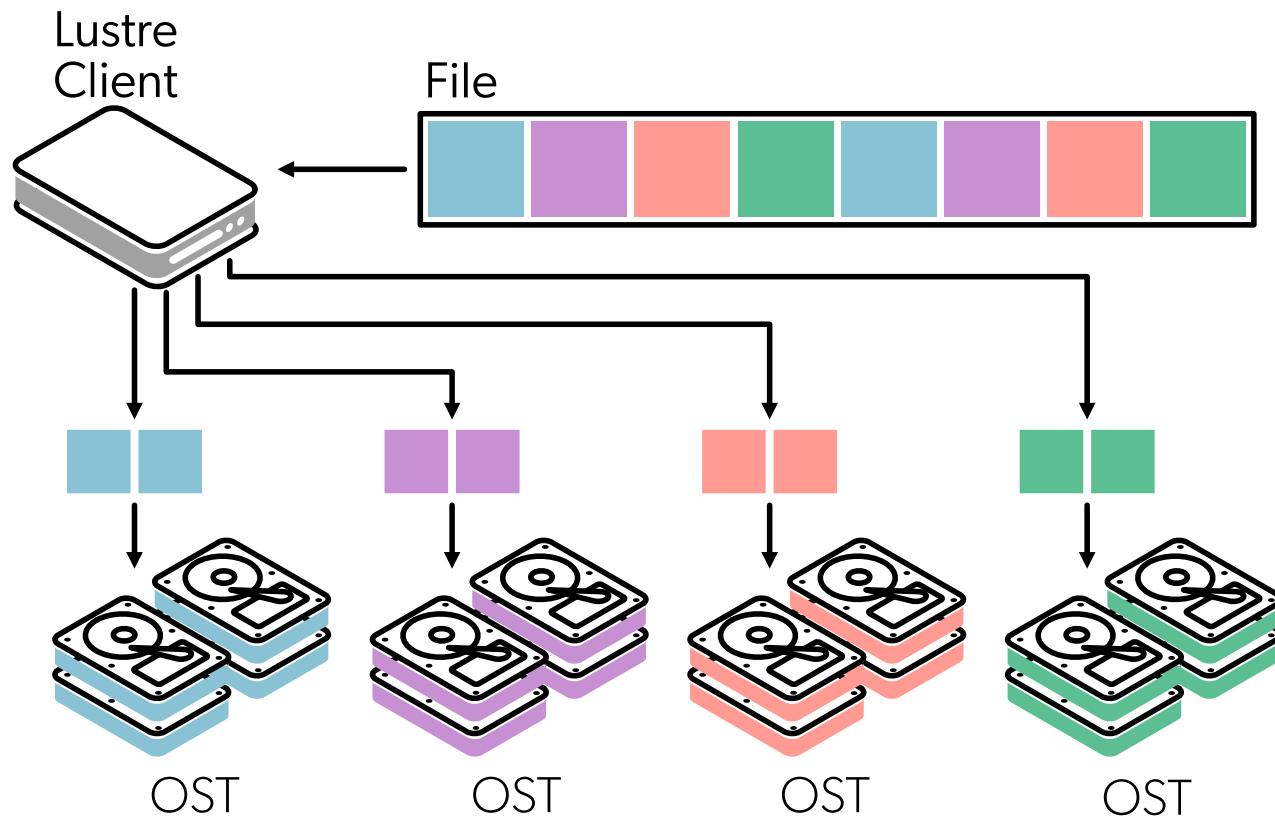
Lumi uses Lustre,  
a popular parallel filesystem

- Parallel filesystems are built from distributed hardware
- One or more metadata servers manage one or more metadata targets
- Many object storage servers manage each one or more object storage targets
- Key to I/O performance:  
Use the parallel resources  
(but mind that they are shared with other users)

Image from

<https://docs.lumi-supercomputer.eu/storage/parallel-filesystems/lustre/>

# File striping - an approach at parallel I/O performance



Write file in parallel by striping it across multiple OSTs

- **stripe\_size** sets the homogeneous length of each stripe
- **stripe\_count** sets the number of used OSTs

Main use:

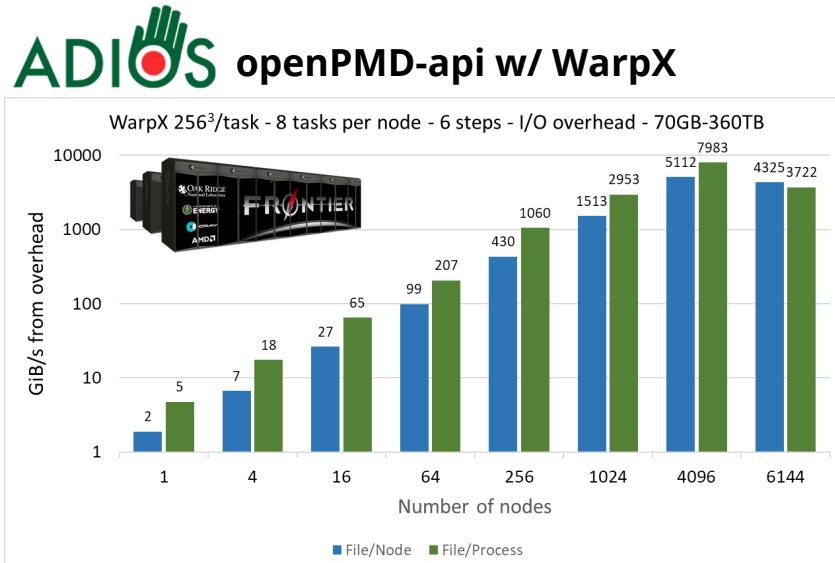
- Improving speed of **serial I/O**
- Improving speed of parallel I/O to a **single file**
- At low scale: Improving speed of parallel I/O to **multiple files**

Better: Write **multiple files** from the start already → **Subfiling**

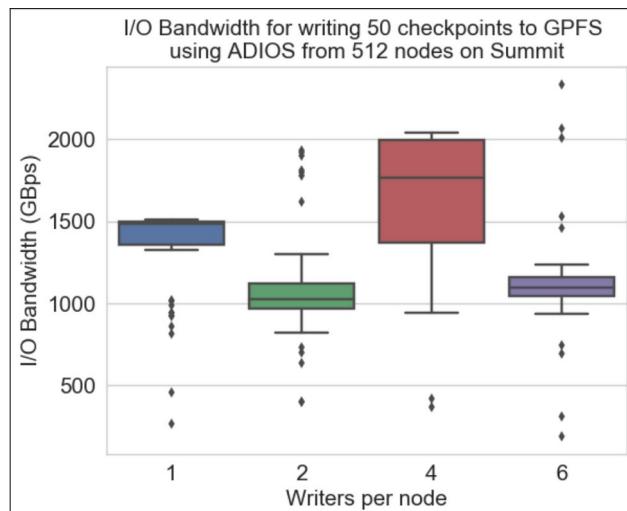
Image from

<https://docs.lumi-supercomputer.eu/storage/parallel-filesystems/lustre/>

# Subfiling - key for performance at full scale



>5.5TB/s FS BW: two-tier lustre w/ high-performance storage & progressive files



L. Wan et al.: Data Management Challenges of Exascale Scientific Simulations:  
A Case Study with the Gyrokinetic Toroidal Code and ADIOS

Subfiling is NOT independent I/O

- Mapping processes to files N:1 scales badly
- Mapping processes to files N:N scales up to a mediocre size, then overloads the filesystem
- Subfiling: N:M mapping where M≤N
- Many different aggregation schemes in ADIOS2 and HDF5, one size does not fit all

**ADIOS2:** Aggregate to one writer per node by default

**HDF5:** Complex N→N aggregation scheme due to structured output to disk

# Advanced Hands-On: **openPMD-api:** Python API: Parallel Writing

**Module environment** at `/project/project_465001310/workshop_software/env.sh`

**Materials** at [https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024\\_workshop](https://github.com/alpaka-group/alpaka-workshop-slides/tree/oct2024_workshop)

**Launch Notebook** at <https://www.lumi.csc.fi/>

Read the TODO comments inside `write_parallel.py`



# Where to find us

→ head to <https://github.com/openPMD/>



openPMD  
Open Standard for Particle-Mesh Data Files  
<https://www.openPMD.org> [axelhuebl@lbl.gov](mailto:axelhuebl@lbl.gov)



Repositories 17 Packages People 50 Teams 5 Projects

## Pinned repositories

[openPMD-standard](#)  
Open Standard for Particle-Mesh Data Files  
41 stars, 17 forks

[openPMD-projects](#)  
Overview on Projects around openPMD  
4 stars, 4 forks

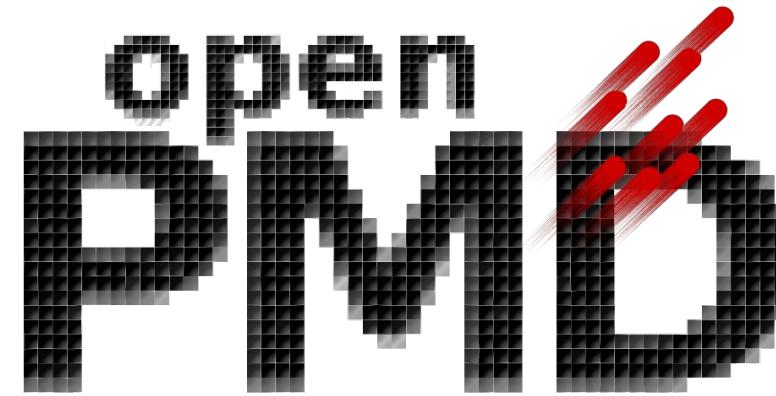
[openPMD-viewer](#)  
Python visualization tools for openPMD files  
35 stars, 26 forks

[openPMD-api](#)  
C++ & Python API for Scientific I/O  
C++ language, 55 stars, 30 forks

[openPMD-visit-plugin](#)  
Plugin allowing VisIt to read openPMD files  
C language, 8 stars, 3 forks

[openPMD-example-datasets](#)  
HDF5 Example Files  
Python language, 5 stars, 1 fork

...and of course <https://openpmd-api.readthedocs.io/>



The **openPMD standard** is co-authored by [Axel Huebl](#), [Rémi Lehe](#), Jean-Luc Vay, David P. Grote, Ivo F. Sbalzarini, Stephan Kuschel, David Sagan, Frédéric Pérez, Fabian Koller, [Franz Poeschel](#), Carsten Fortmann-Grote, Ángel Ferran Pousa, Juncheng E, [Maxence Thévenet](#), and Michael Bussmann.

The authors are thankful for the **community contributions** to libraries, software ecosystem, user support, review and integrations. Particularly, thank you to Yaser Afshar, Lígia Diana Amorim, James Amundson, Weiming An, Igor Andriyash, Ksenia Bastrakova, [Jean Luca Bez](#), Richard Briggs, Heiko Burau, Jong Choi, Ray Donnelly, Dmitry Ganyushin, Marco Garten, Lixin Ge, Berk Geveci, Daniel Grassinger, Alexander Grund, [Junmin Gu](#), Marc W. Guetg, Ulrik Günther, Sören Jalas, Manuel Kirchen, John Kirkham, Scott Klasky, Noah Klemm, Fabian Koller, Mathieu Lobet, Christopher Mayes, Ritiek Malhotra, Paweł Ordyna, Richard Pausch, [Norbert Podhorszki](#), David Pugmire, Felix Schmitt, [Erik Schnetter](#), Dominik Stańczak, Klaus Steiniger, Michael Sippel, Frank Tsung, Lipeng Wan, René Widera, and Erik Zenker!