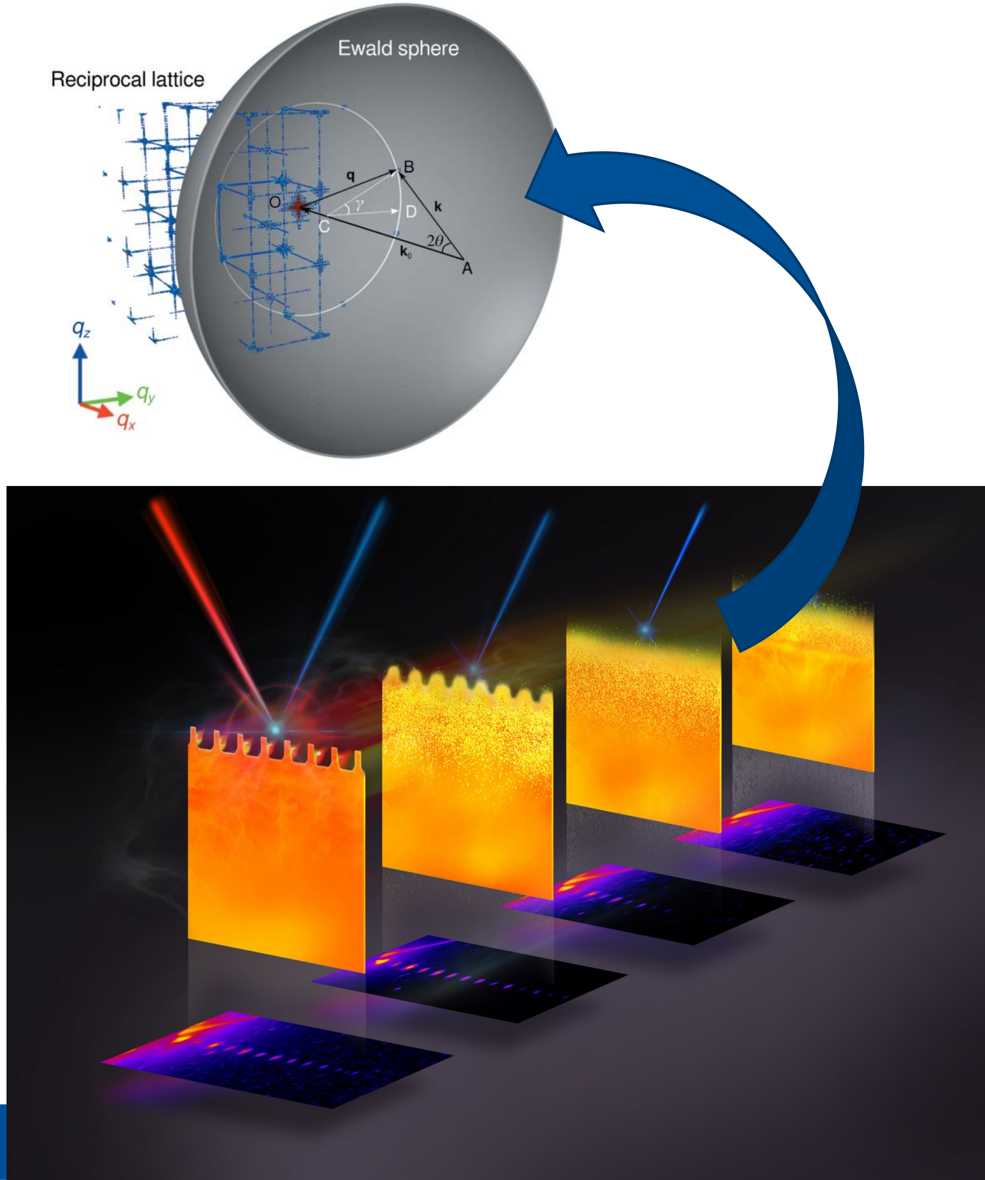
openPMD

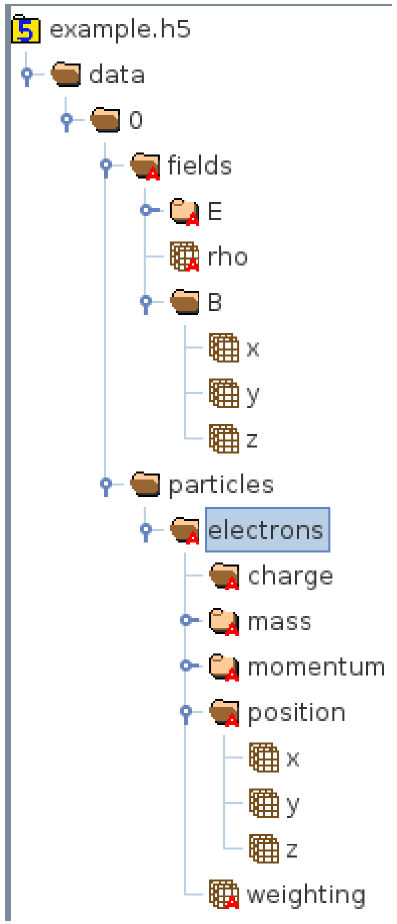
Brief Overview

Franz Poeschel | Alexander Debus | Axel Huebl
CASUS | HZDR| LNBL

Online NFDI NeXus Workshop
March 17-18, 2022



openPMD – a F.A.I.R. standard for physics data at the Exascale



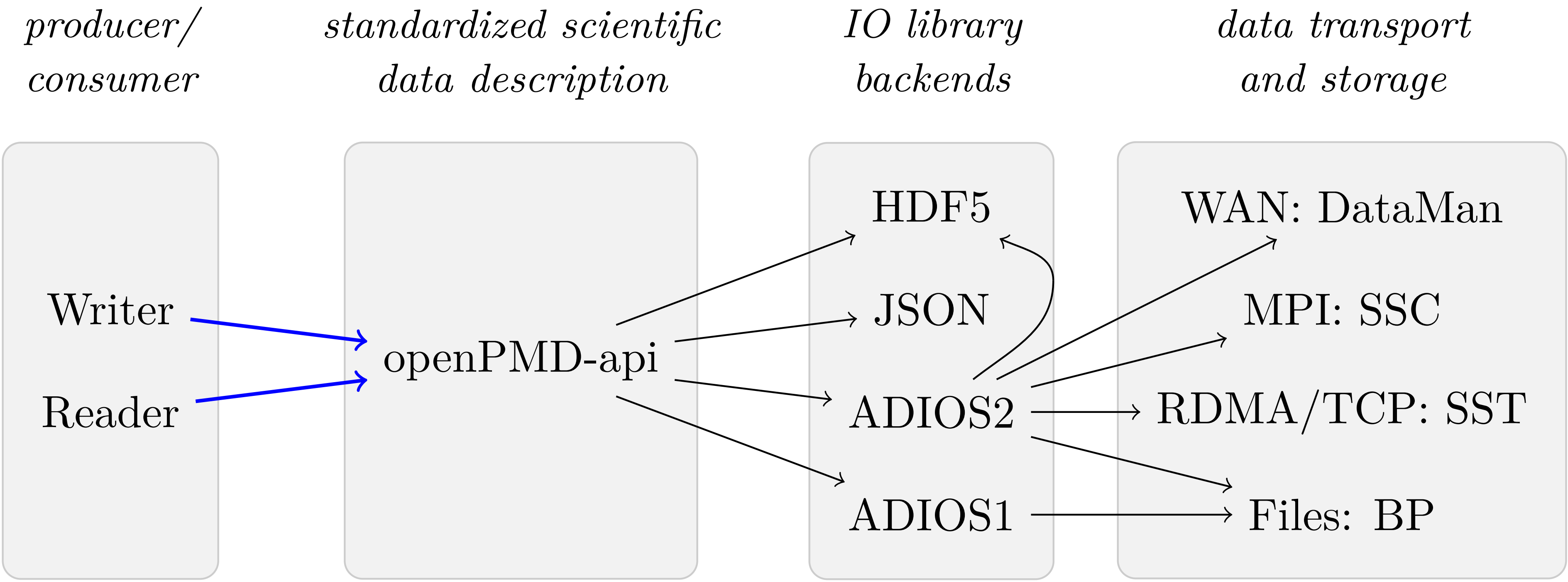
```
example.h5
  data
    0
      fields
        E
        rho
        B
          x
          y
          z
      particles
        electrons
          charge
          mass
          momentum
          position
            x
            y
            z
          weighting
```

Self-describing, **data format agnostic** standard for **frictionless exchange** of **particle-mesh data**

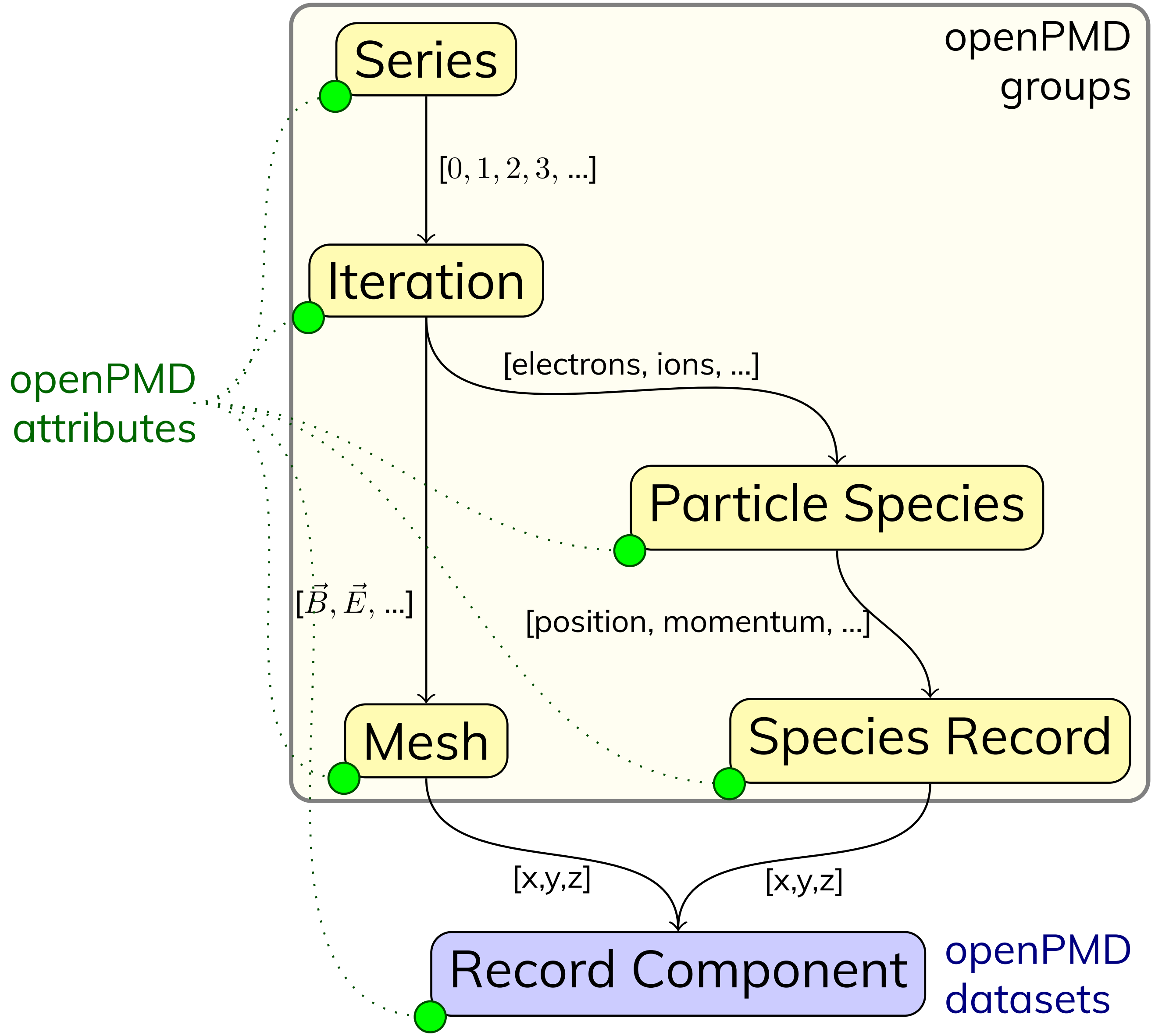
Flagship implementation: **openPMD-api**:

- Describe particle-mesh data in a unified way
- API in C++ and Python (upcoming: Julia)
- Flexibly store to / read from interchangeable backends:
  - ADIOS1/2
  - HDF5
  - JSON (serial only)

openPMD – open stack for scientific I/O



openPMD hierarchy



- **Structure** for series & snapshots
- Records for **physical observables**
- Attributes: **conversion, description**
- **n-dimensional mesh data:** e.g. n-dimensional images
- **Particle data:** e.g. data reduction via particle representation of image data
- Constants, mixed precision, complex numbers

Getting your feet wet: JSON backend

```
{
  "attributes": {
    "author": {
      "datatype": "STRING",
      "value": "franz"
    },
    "date": {
      "datatype": "STRING",
      "value": "2020-10-08 19:29:13 +0200"
    },
    "some more...": null
  },
  "data": {
    "0": {
      "attributes": {
        "cell_depth": {
          "datatype": "DOUBLE",
          "value": 4.252342224121094
        },
        "cell_height": {
          "datatype": "DOUBLE",
          "value": 1.0630855560302734
        },
        "cell_width": {
          "datatype": "DOUBLE",
          "value": 4.252342224121094
        },
        "many many more": null
      },
      "fields": {
        "B": {
            "attributes": {
                "axisLabels": {
                  "datatype": "VEC_STRING",
                  "datatype": "VEC_STRING",
                  "value": [
                    "z",
                    "y",
                    "x"
                  ]
                }
              },
              "x": {
                "attributes": {
                  "position": {
                    "datatype": "VEC_DOUBLE",
                    "value": [
                      0,
                      0.5,
                      0.5
                    ]
                  },
                  "unitSI": {
                    "datatype": "DOUBLE",
                    "value": 40903.82224060171
                  }
                },
                "data": [
                  [
                    [
                      "multidimensional dataset here"
                    ]
                  ]
                ]
    } } } } } }
```

- Part of the package: No need to install 3rd-party dependencies
- Useful for debugging and prototyping
- Serial usage only
- Courtesy to Nils Lohmann‘s JSON library for C++

openPMD – a dataset in ADIOS2

```
float     /data/43/particles/electrons\
                 /particlePatches/extent/x                   {2}
float     /data/…/particlePatches/extent/y                   {2}
uint64_t  /data/…/particlePatches/numParticles               {2}
uint64_t  /data/…/particlePatches/numParticlesOffset         {2}
float     /data/…/particlePatches/offset/x                   {2}
float     /data/…/particlePatches/offset/y                   {2}
float     /data/…/position/x                                 {123}
float     /data/…/position/y                                 {123}
uint64_t  /data/…/positionOffset/x                           {123}
uint64_t  /data/…/positionOffset/y                           {123}
```

n-dim. datasets
for heavyweight data

Hierarchical
data organization

```
string    /basePath                                          attr   = "/data/%T/"
double    /data/43/dt                                        attr   = 1
double    /data/…/particlePatches/extent/unitDimension       attr   = {0, 0, 0, 0, 0, 0, 0}
double    /data/…/particlePatches/extent/x/unitSI            attr   = 1
double    /data/…/particlePatches/extent/y/unitSI            attr   = 1
double    /data/…/particlePatches/numParticles/unitSI        attr   = 1
double    /data/…/particlePatches/numParticlesOffset/unitSI  attr   = 1
double    /data/…/particlePatches/offset/unitDimension       attr   = {0, 0, 0, 0, 0, 0, 0}
double    /data/…/particlePatches/offset/x/unitSI            attr   = 1
double    /data/…/particlePatches/offset/y/unitSI            attr   = 1
float     /data/…/position/timeOffset                        attr   = 0
double    /data/…/position/unitDimension                     attr   = {1, 0, 0, 0, 0, 0, 0}
double    /data/…/position/x/unitSI                          attr   = 1
double    /data/…/position/y/unitSI                          attr   = 1
float     /data/…/positionOffset/timeOffset                  attr   = 0
double    /data/…/positionOffset/unitDimension               attr   = {1, 0, 0, 0, 0, 0, 0}
double    /data/…/positionOffset/x/unitSI                    attr   = 1
double    /data/…/positionOffset/y/unitSI                    attr   = 1
…
```

Attributes
for self-descriptiveness

Our requirements to a modern scientific I/O stack

Efficiency:

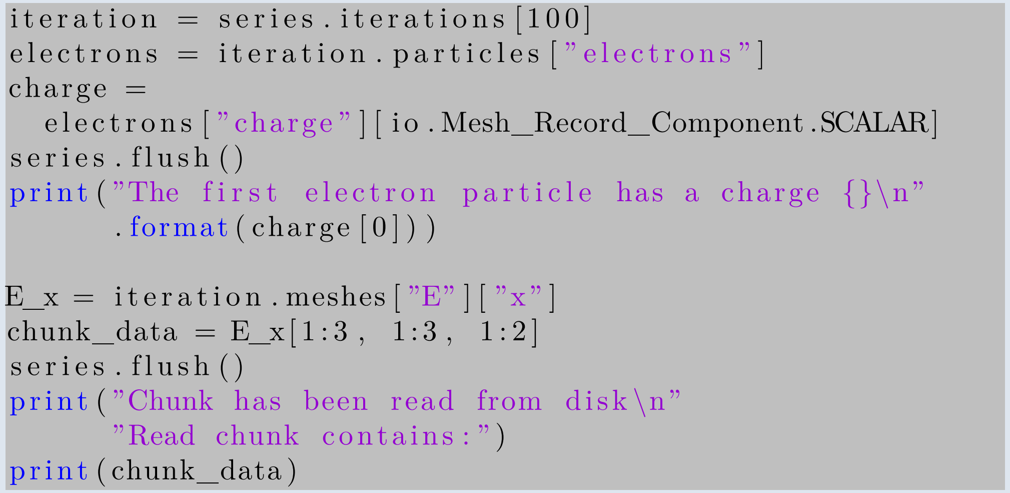
Scalable performance
in preparation for the Exascale era
provided by optimized backends

ADIOS

The HDF Group

Expressiveness:

Express scientific data naturally
within the problem's domain
Avoid dealing with low-level concepts
FAIR compliance



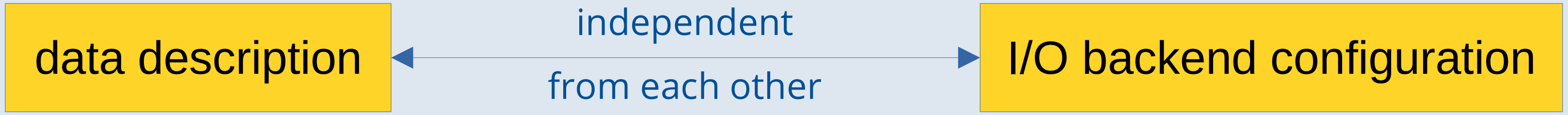
```
iteration = series.iterations[100]
electrons = iteration.particles["electrons"]
charge =
  electrons["charge"][io.Mesh_Record_Component.SCALAR]
series.flush()
print("The first electron particle has a charge {}\n"
      .format(charge[0]))

E_x = iteration.meshes["E"]["x"]
chunk_data = E_x[1:3, 1:3, 1:2]
series.flush()
print("Chunk has been read from disk\n"
      "Read chunk contains:")
print(chunk_data)
```

Our requirements to a modern scientific I/O stack

Flexibility: Migrate between systems and setups without changing I/O logic

Specify backends, compression, aggregation, chunking, … at runtime

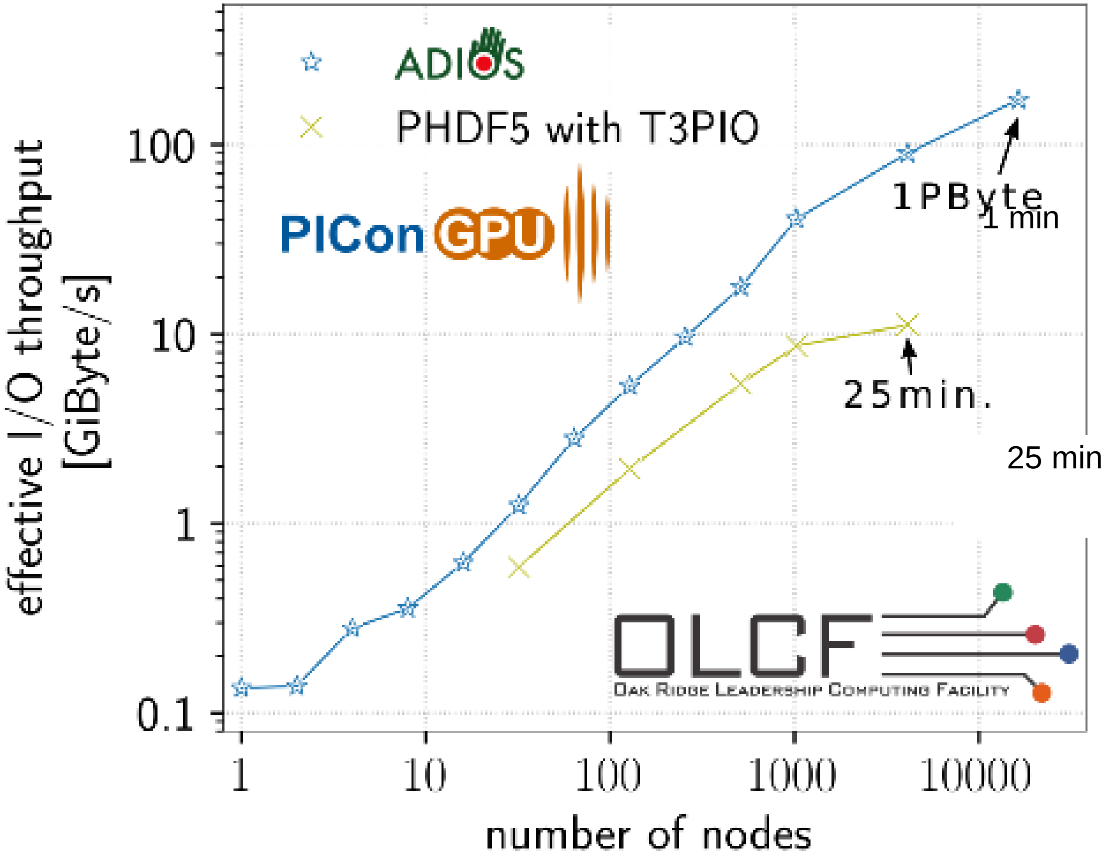


Streaming IO: Easy transition from file-based to streaming workflows

```python
import openpmd_api as io

# pick backend by filename extension
series = io.Series("simOutput.h5",   io.Access.create)
series = io.Series("simOutput.bp",   io.Access.create)
series = io.Series("simOutput.sst",  io.Access.create)
series = io.Series("simOutput.json", io.Access.create)
```

IO flexibility matters for scaling

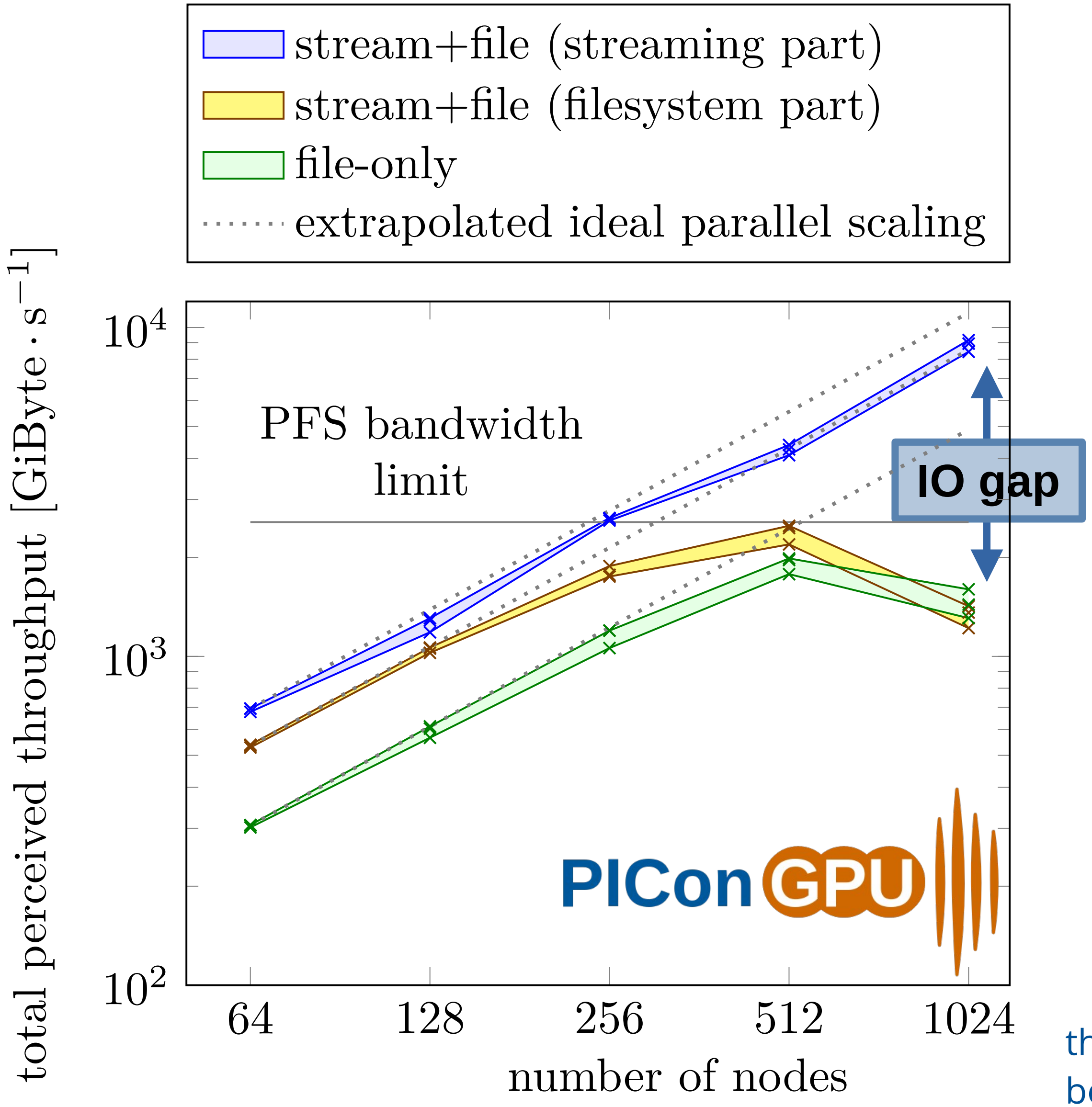


Throughput scaling on Titan

- IO requires special attention to stay **performant at extreme scale**
- **ADIOS** optimizes for this
- **openPMD:** describe data once, use either backend

A. Huebl et al., "On the Scalability of Data Reduction Techniques in Current and Upcoming HPC Systems from an Application Perspective", In: Lect. Notes Comput. Sci. 10524.4, pp.15-20 (2017)

Streaming matters for scaling



throughput scaling on Summit
benchmarks at 1024 nodes done after Summit system upgrade

Disk-based benchmarks (yellow, green):
Limit of the filesystem (2.5TiB/s) reached after a fraction of the system size

Streaming IO (blue):
Provides scalability beyond what the filesystem can give us

openPMD-api:
Pick IO strategy without recompiling
Close IO gap by seamlessly transitioning from disk IO to streaming IO

openPMD powered Projects and Users

CASUS
CENTER FOR ADVANCED
SYSTEMS UNDERSTANDING

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: S Bastrakov, A Debus, A Huebl et al.
- **WarpX** (LBNL, LLNL)
  electro-dynamic/static particle-in-cell code
  maintainers: JL Vay, D Grote, R Lehe 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
- …and more

Data processing and visualization:

- **openPMD-viewer** (LBNL, DESY)
  high-level python API & interactive jupyter notebook GUI
  maintainer: R Lehe
- **Paraview** (Kitware + third party)
  multi-platform data analysis and visualization application
  maintainers: Kitware
- **VisualPIC** (DESY)
  post-processing and visualization for particle-in-cell data
  maintainer: A Ferran Pousa
- **postpic** (IOQ Jena)
  post-processing and visualization for particle-in-cell data
  maintainer: S Kuschel
- **yt project** (third party + HZDR: reader implementation)
  framework for parallel analysis and visualization
  maintainer: the yt team (HZDR: contribution)
- **VisIt** (LLNL)
  parallel post-processing and 3D visualization
  maintainer: LLNL (NERSC: contribution)

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

Libraries and language bindings:

- **pyDive** (HZDR)
  parallel numpy for ipython notebook
  maintainer: H Burau
- **libsplash** (HZDR, TU Dresden)
  high-level C++ HDF5 library for mesh and particle records
  maintainers: F Schmitt, A Huebl
- **openPMD-api** (HZDR)
  reference API for openPMD data handling
  maintainers: A Huebl, J Gu, F Poeschel et al.

Tools and converters:

- **file validators** (HZDR, LBNL)
  development scripts
  maintainer: A Huebl, R Lehe
- **XDMF creation** (TU Dresden, HZDR)
  xml meta file creation for (serial) reading in VTK
  maintainer: HZDR
- **HDF Compass** (third party + HZDR: ADIOS implementation)
  viewer for HDF5 files and related formats
  maintainer: HDF Group (HZDR: contribution)
- **VisIt** (LLNL)
  parallel post-processing and 3D visualization
  maintainer: LLNL (NERSC: contribution)

Exhaustive list:

https://github.com/openPMD/openPMD-projects

openPMD and NeXus – potential for joining forces

github.com/openPMD

- **openPMD is backend agnostic**,

  suitable for **any kind of hierarchical, self-describing** data format, such as,

  but not limited to **HDF5, ADIOS2, JSON…**

- **openPMD is exascale-ready as data volumes scale to 100s of TBs to PBs**,

  with granular control over data sources, sinks and aggregators.

- **openPMD supports in-memory streaming**,

  which becomes essential for handling ever growing data rates, allowing for in-situ data analysis and filtering before data is written to disk.

- **openPMD is open source and extensible meta-standard**,

  featuring a rich ecosystem of tools and APIs.

openPMD and NeXus – potential for joining forces

- **Laser-plasma experimental community**
  - needs to handle large data volumens and rates. Currently single shot (▭ 100GB per day), aiming for 10 Hz operation and beyond (▭ 10TB per day).
  - needs to adopt and extend NeXus for its domain.
- The potential of openPMD and NeXus is not in converting between the existing vocabularies of both standards, but rather to add and extend each others‘ unique capabilities.

NeXus: Rich vocabulary and tools describing experimental data,
openPMD: Backend-agnostic, open, scalable I/O including in-memory streaming support.

For getting started

- openPMD could be integrated into NeXus as a non-breaking change (regarding HDF5), extending NeXus to be backend-agnostic, while retaining existing HDF5 functionality.
- The NeXus semantics and APIs could be added to the openPMD-project as an domain-specific extension.