From HPC to the Edge

Alpaka, LLAMA and other animals

Michael Bussmann

DFG - NFDI

DAPHNE 4NFDI

PUNCH 4NFDI

CASUS
CENTER FOR ADVANCED
SYSTEMS UNDERSTANDING

www.casus.science

HZDR
HELMHOLTZ ZENTRUM
DRESDEN ROSSENDORF

TECHNISCHE
UNIVERSITÄT
DRESDEN

CBG
Max Planck Institute
of Molecular Cell Biology
and Genetics

UFZ HELMHOLTZ
Centre for Environmental Research

Uniwersytet
Wrocławski

SPONSORED BY THE
Federal Ministry
of Education
and Research

STAATSMINISTERIUM
FÜR WISSENSCHAFT
KULTUR UND TOURISMUS

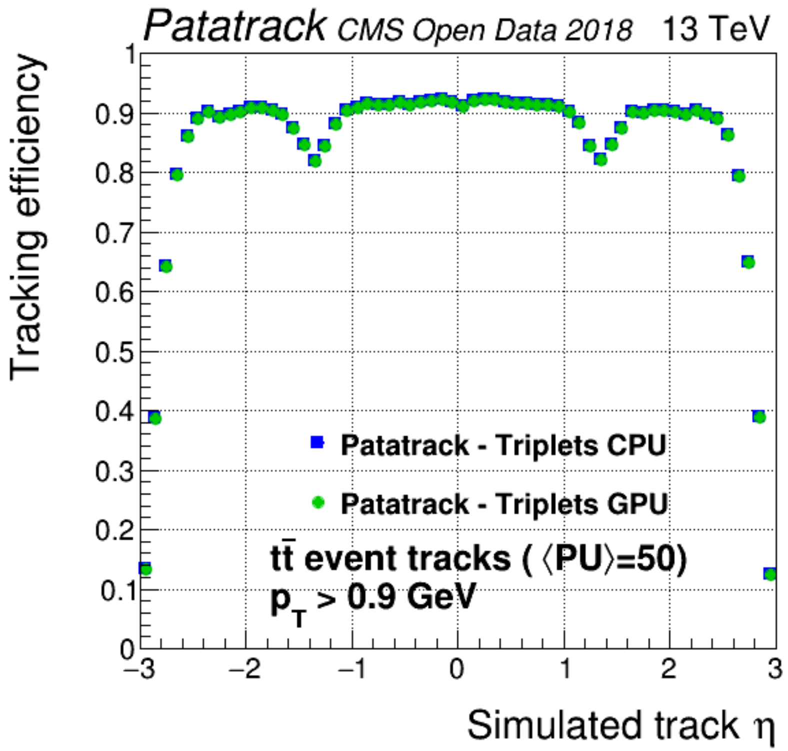
Freistaat
SACHSEN

Challenges in parallel programming today

High data rates, complex algorithms, Sustainability

Throughput & Sustainability

- **Memory bound**: Throughput is decisive to use your hardware efficiently
- **Development cycles**: Hardware is changing every two years
- **A zoo without a keeper**: CPUs, GPUs, FPGAs, ARM, RISC-V
- **Reproducibility & trust**: Algorithms have to do the same regardless of Hardware

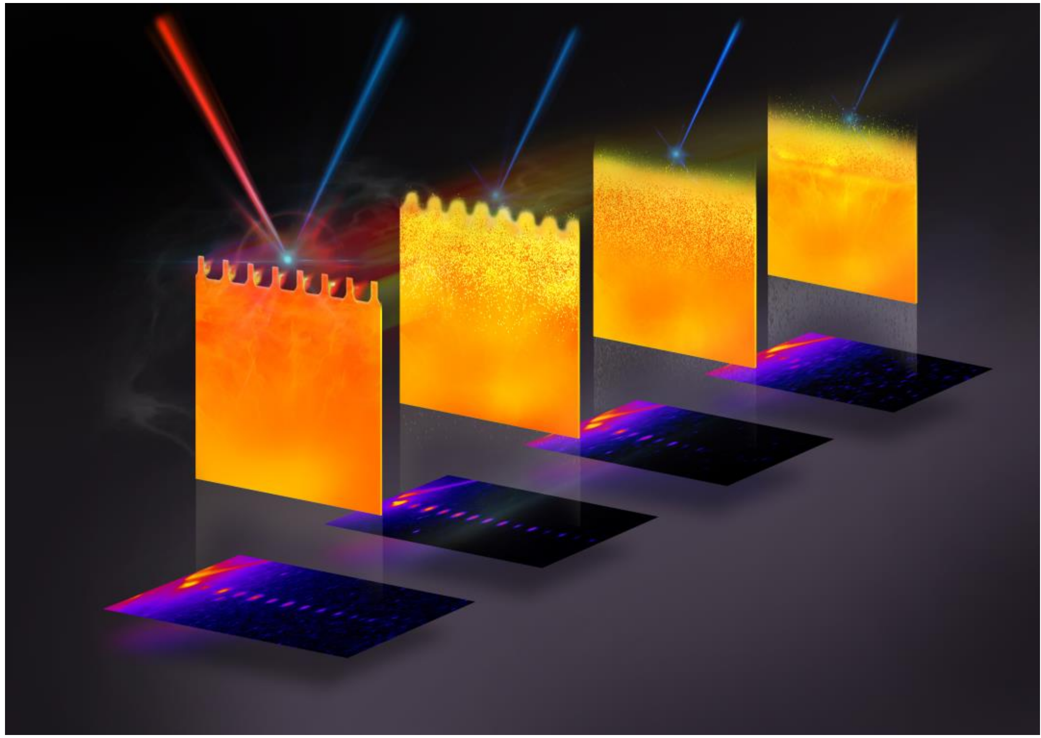


Challenges in parallel programming today

What it takes to use your hardware

How to use your hardware the best you can

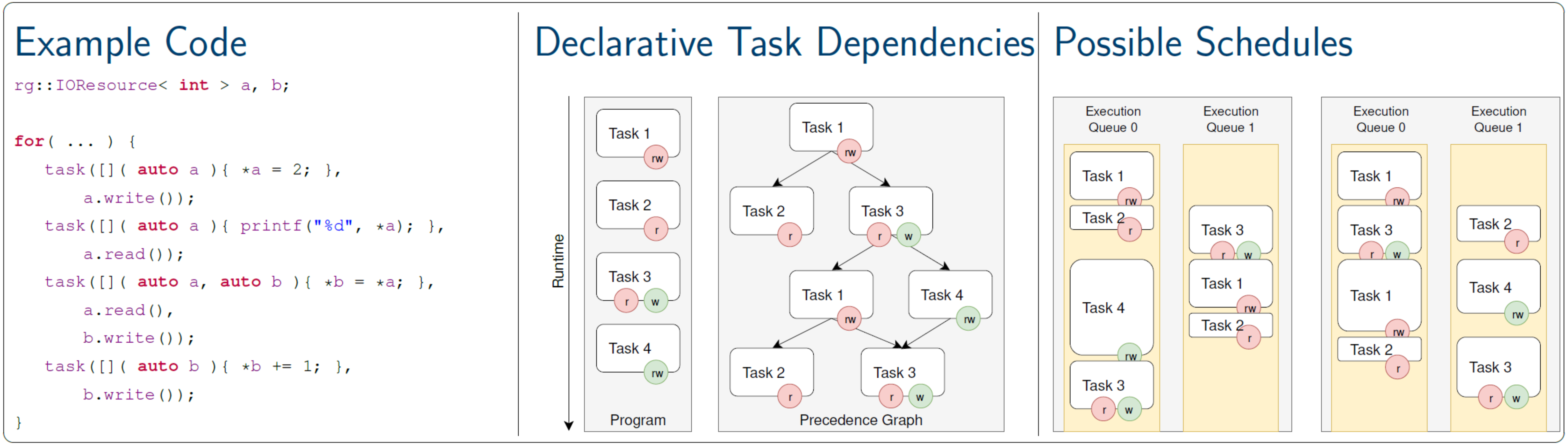
- **Data locality** is key, so you need to express your data dependencies
- **Data layout** is (still) important, so you need to be able to change it
- **Parallel efficiency** = Express both data + task parallelism
- Do not write to disk if you can, **stream your data**



Data Locality: Know and express your data dependencies

REDGRAPES: Express your task parallelism by data dependencies

Data Dependencies **Task Tree** **Scheduling Strategy**



Example Code

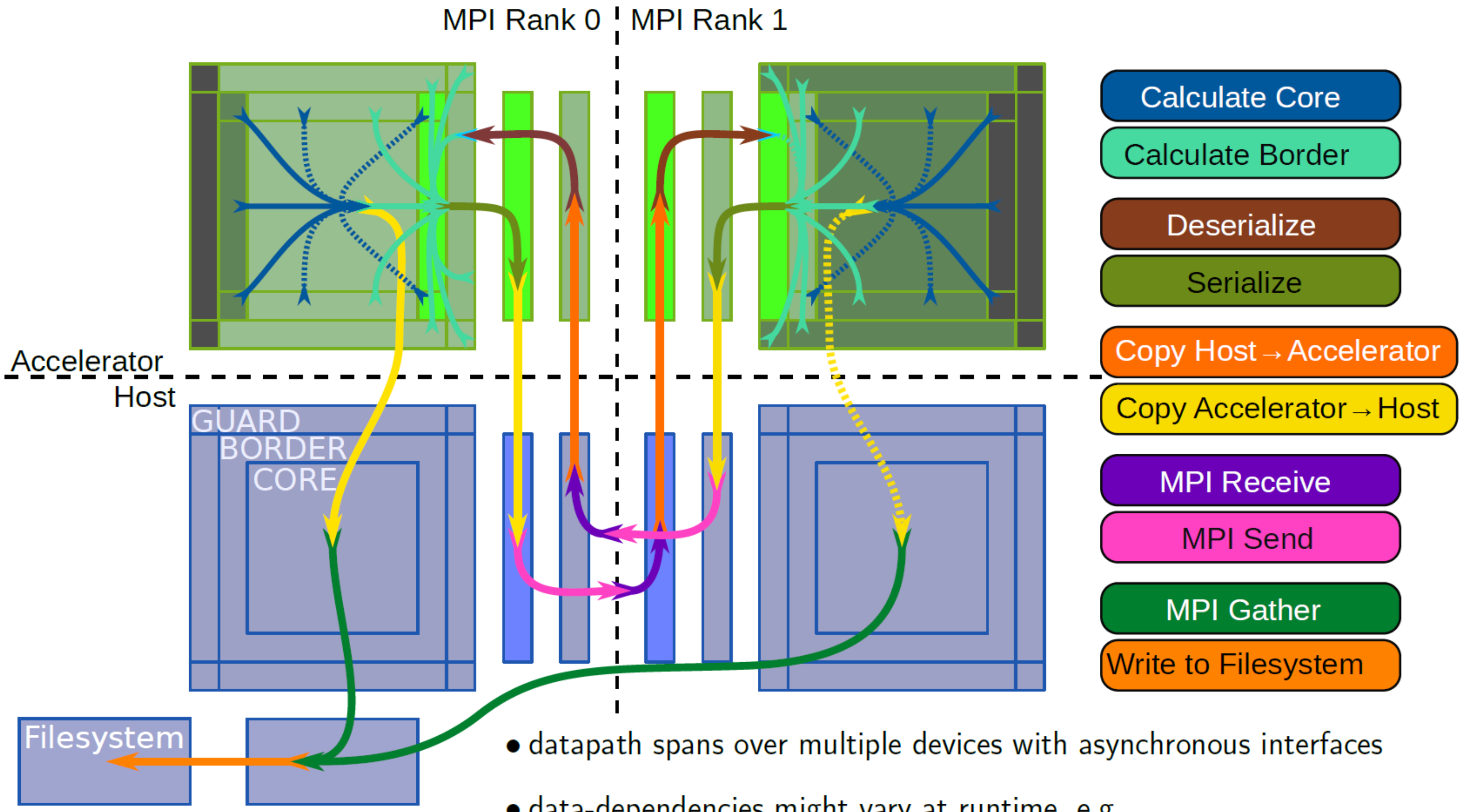
```
rg::IOResource< int > a, b;

for( ... ) {
    task([]( auto a ){ *a = 2; },
        a.write());
    task([]( auto a ){ printf("%d", *a); },
        a.read());
    task([]( auto a, auto b ){ *b = *a; },
        a.read(),
        b.write());
    task([]( auto b ){ *b += 1; },
        b.write());
}
```

Declarative Task Dependencies

Possible Schedules

Data Locality: Know and express your data dependencies

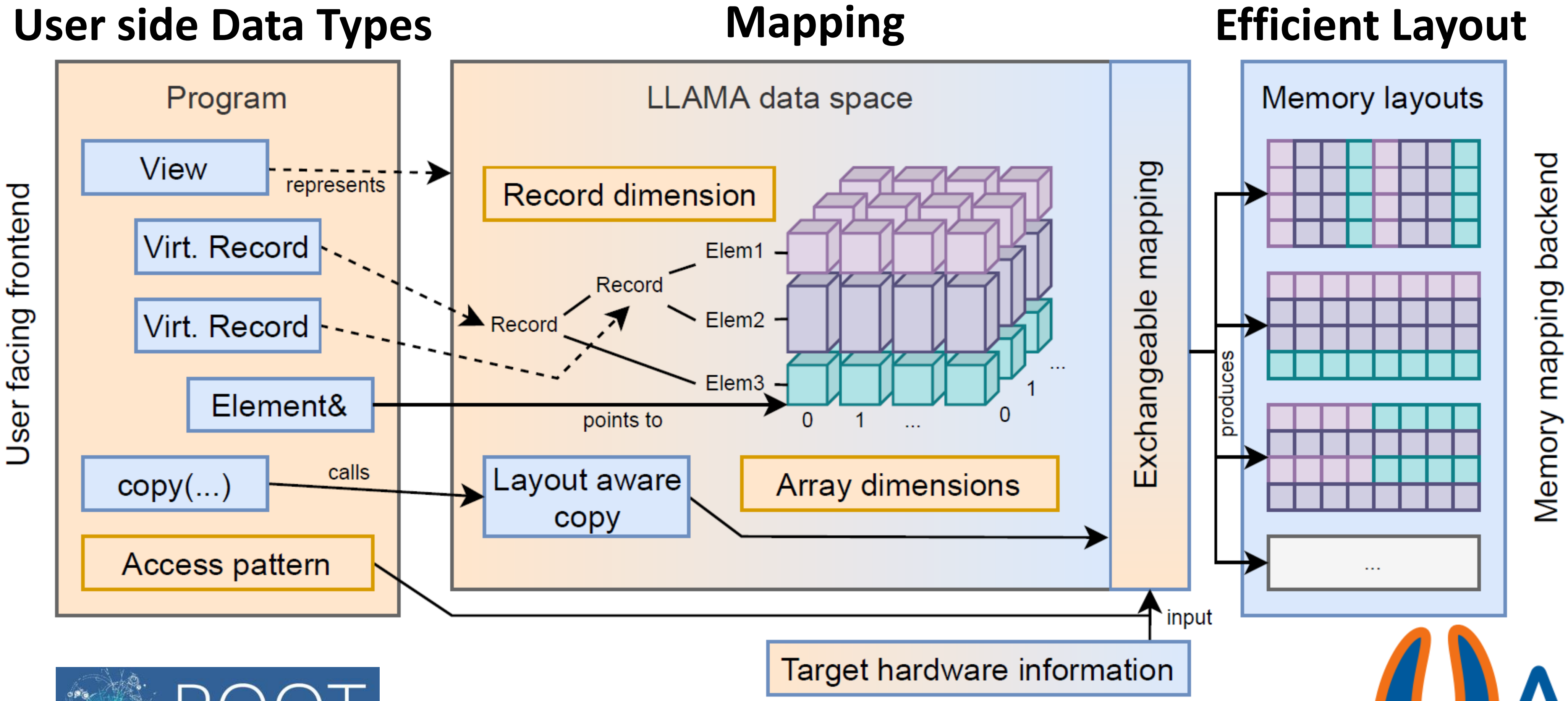
REDGRAPES: Express your task parallelism by data dependencies



- datapath spans over multiple devices with asynchronous interfaces
- data-dependencies might vary at runtime, e.g.

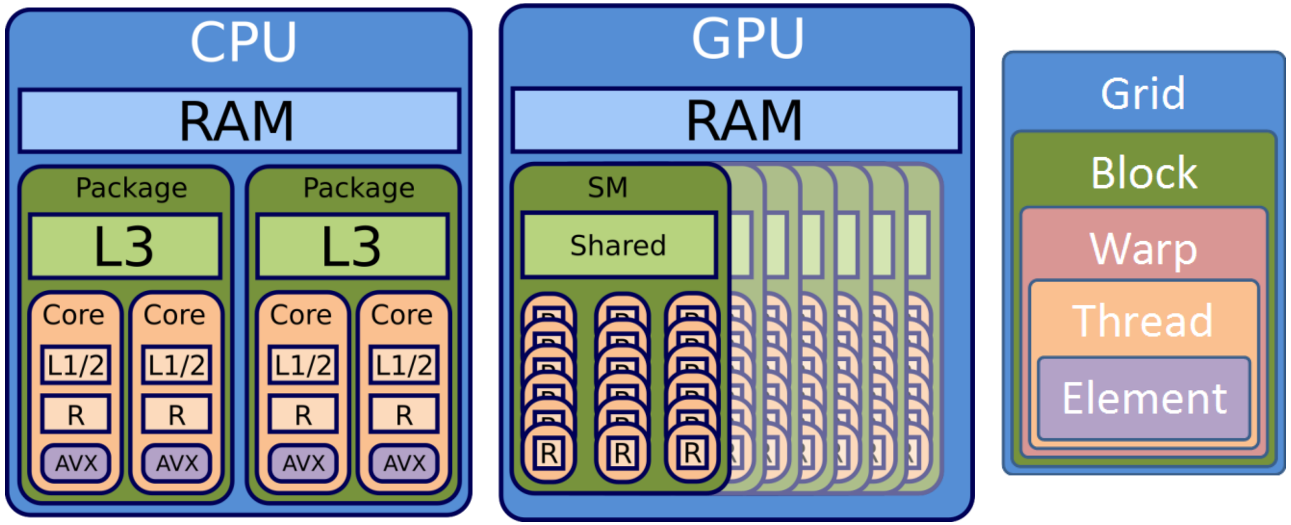
Data Layout: Layouts change, but code should not

LLAMA: Efficient data layouts without changing your code



Parallel Efficiency: Express parallelism across platforms

ALPAKA: Single-source programming for CPUs, GPUs & FPGAs



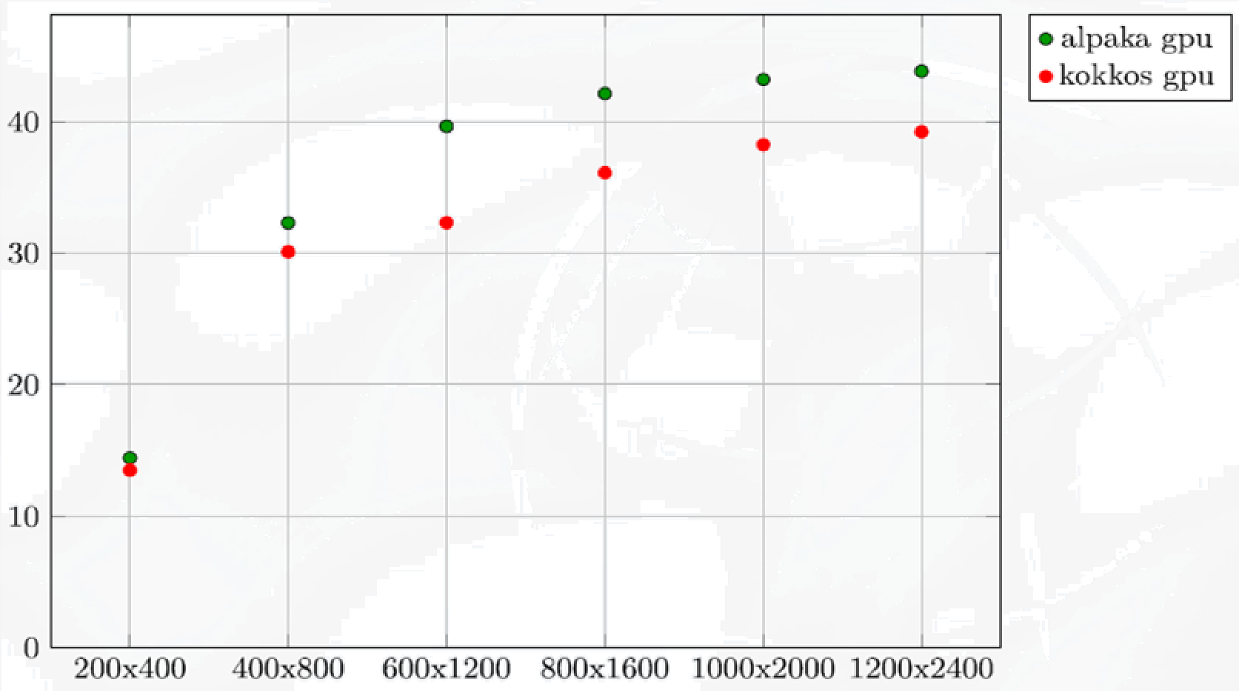
```
#ifdef CUDA_ENABLE
   // CUDA Kernel implementation
   // …

#elif OPENMP_ENABLE
   // OpenMP implementation
   // …

#else
   // Sequential CPU implementation
   // …

#endif
```

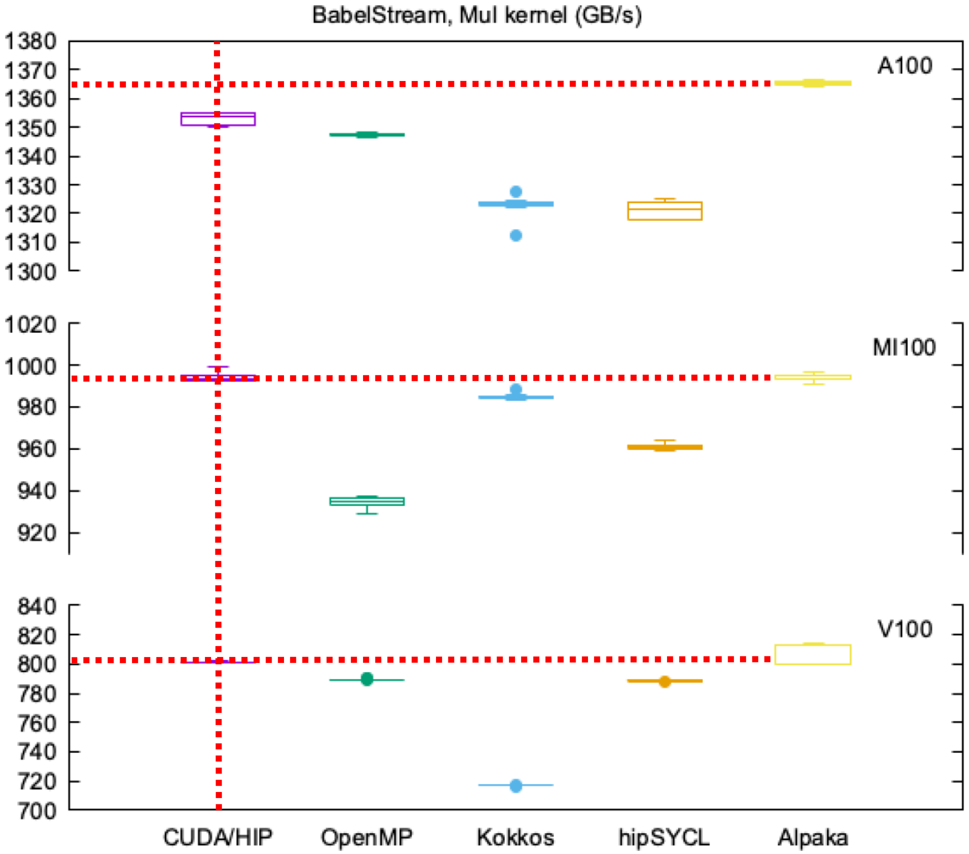
intel AMD NVIDIA ARM XILINX …

alpaka

ALPAKA: Single-source programming for CPUs, GPUs & FPGAs

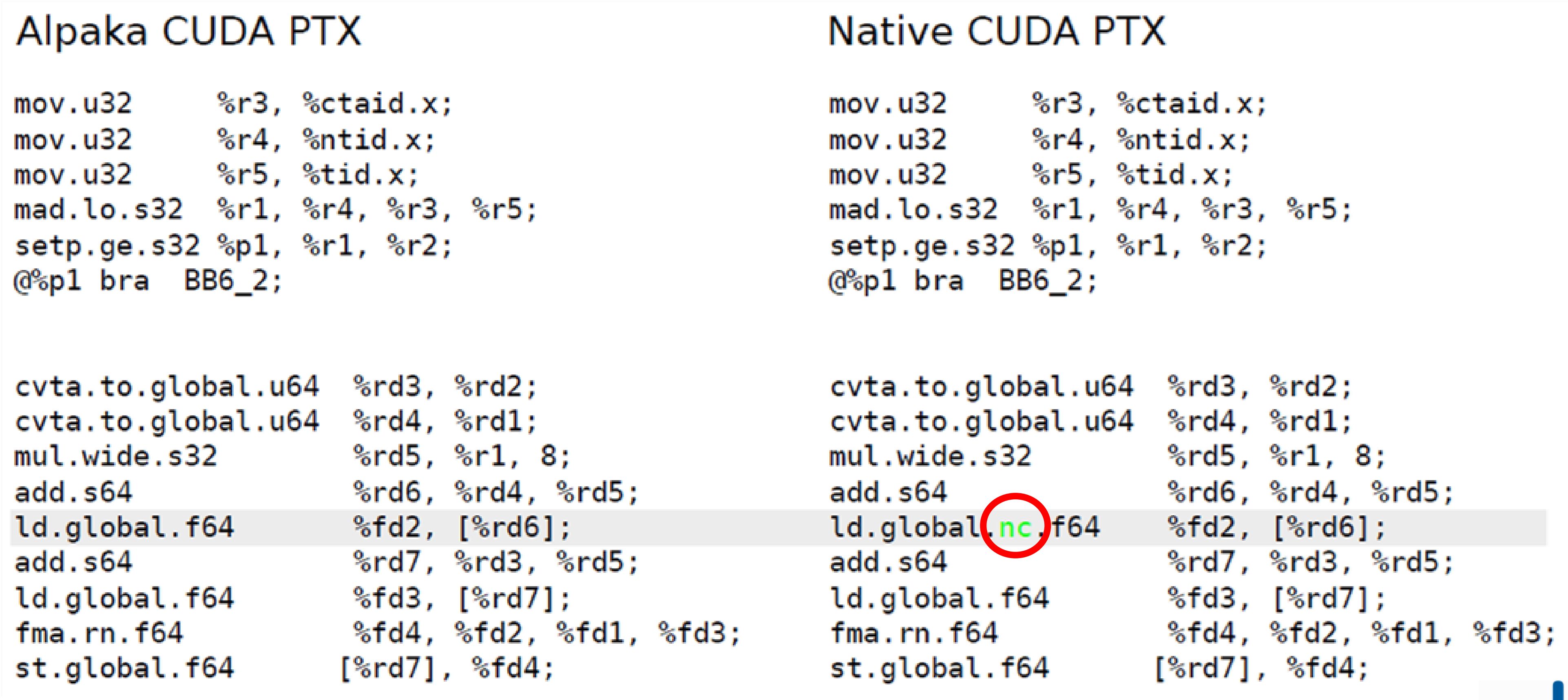
ALPAKA: Close to native performance





ALPAKA: Single-source programming for CPUs, GPUs & FPGAs

Close to native performance



Alpaka CUDA PTX

```
mov.u32      %r3, %ctaid.x;
mov.u32      %r4, %ntid.x;
mov.u32      %r5, %tid.x;
mad.lo.s32   %r1, %r4, %r3, %r5;
setp.ge.s32 %p1, %r1, %r2;
@%p1 bra   BB6_2;


cvta.to.global.u64   %rd3, %rd2;
cvta.to.global.u64   %rd4, %rd1;
mul.wide.s32         %rd5, %r1, 8;
add.s64              %rd6, %rd4, %rd5;
ld.global.f64        %fd2, [%rd6];
add.s64              %rd7, %rd3, %rd5;
ld.global.f64        %fd3, [%rd7];
fma.rn.f64           %fd4, %fd2, %fd1, %fd3;
st.global.f64       [%rd7], %fd4;
```

Native CUDA PTX

```
mov.u32      %r3, %ctaid.x;
mov.u32      %r4, %ntid.x;
mov.u32      %r5, %tid.x;
mad.lo.s32   %r1, %r4, %r3, %r5;
setp.ge.s32 %p1, %r1, %r2;
@%p1 bra   BB6_2;


cvta.to.global.u64   %rd3, %rd2;
cvta.to.global.u64   %rd4, %rd1;
mul.wide.s32         %rd5, %r1, 8;
add.s64              %rd6, %rd4, %rd5;
ld.global.nc.f64     %fd2, [%rd6];
add.s64              %rd7, %rd3, %rd5;
ld.global.f64        %fd3, [%rd7];
fma.rn.f64           %fd4, %fd2, %fd1, %fd3;
st.global.f64       [%rd7], %fd4;
```

I have a C++ CUDA code and am too lazy to port it

CUPLA: Making portable ALPAKA code without effort

Native CUDA Code

```
// CUDA kernel
__global__ void kernel(/* Args */)
{
    /* CUDA code */
}
```

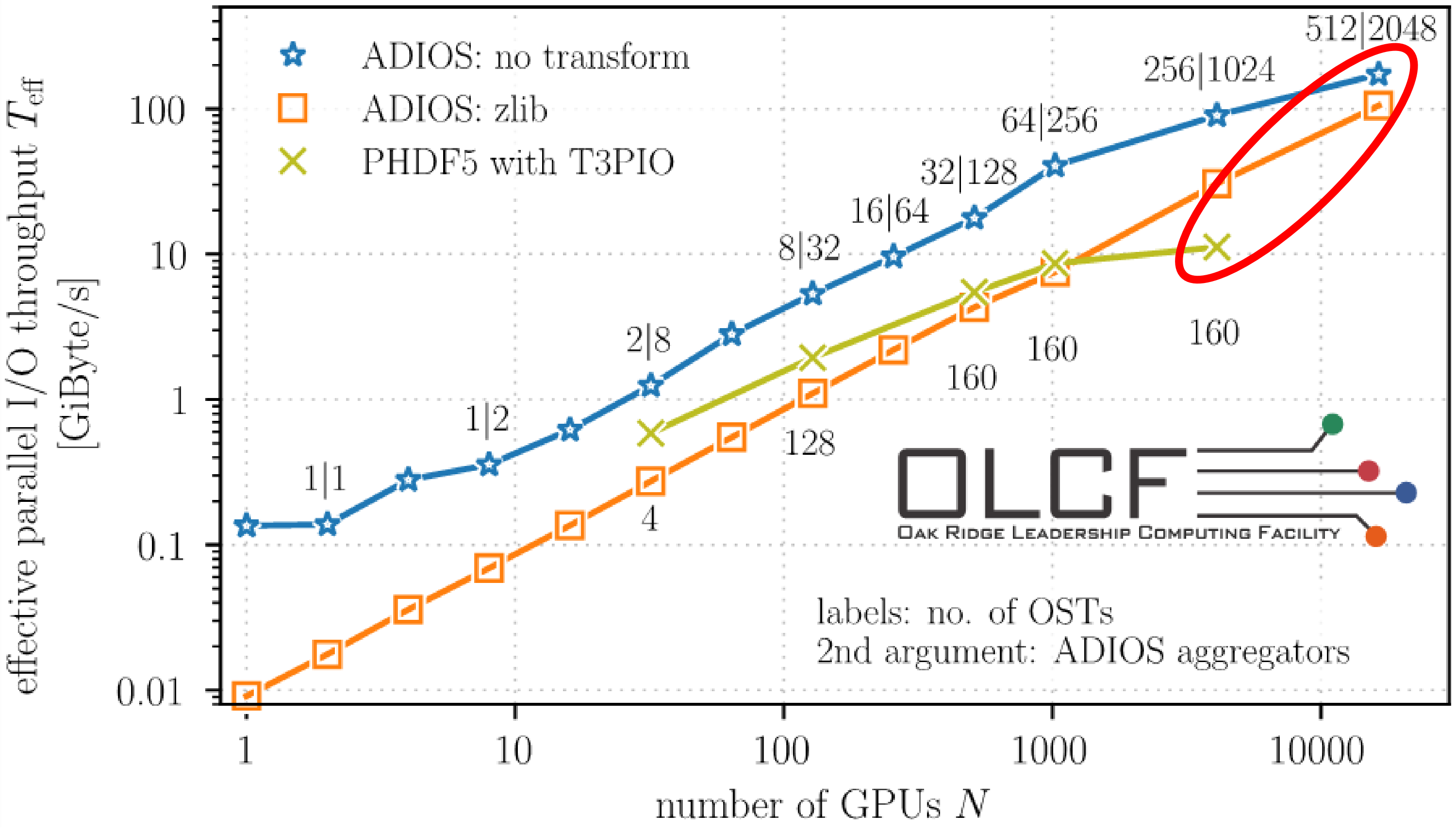
```
// Kernel launch
dim3 gridSize(42, 1, 1);
dim3 blockSize(256, 1, 1);
kernel<<<gridSize, blockSize>>>(/* Args */);
```

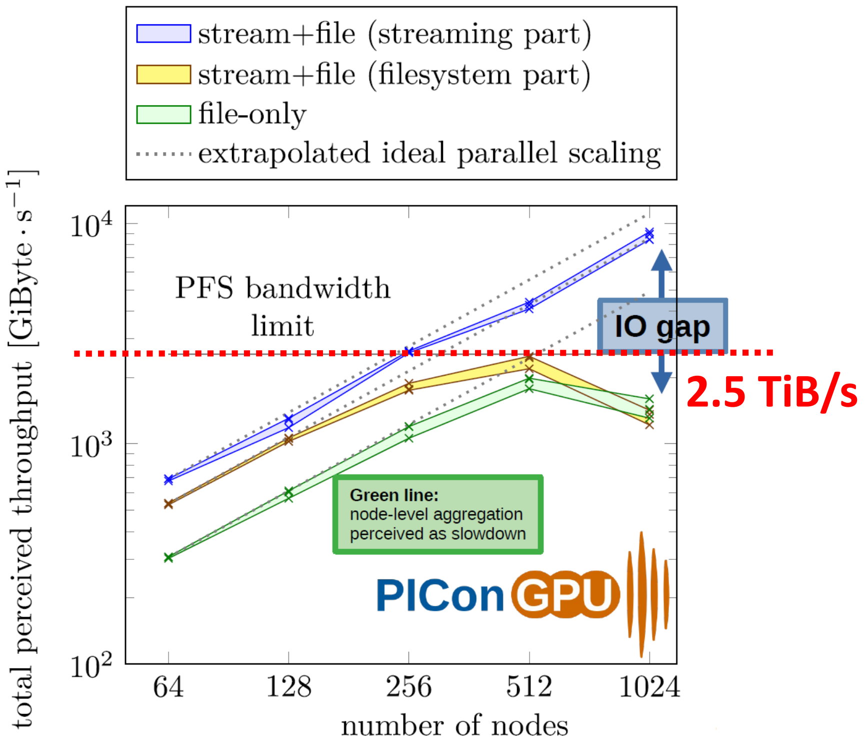
Portable CUPLA Code

```
// include CUPLA-to-CUDA header
#include <cuda_to_cupla.hpp>

// replace kernel definition with functor definition
struct Kernel {
    template<typename TAcc>
    ALPAKA_FN_ACC
    void operator()(TAcc const& acc,
                    /* Args */) const
    {
        /* CUDA code */
    }
};

// Kernel launch
dim3 gridSize(42, 1, 1);
dim3 blockSize(256, 1, 1);
CUPLA_KERNEL(Kernel)(gridSize, blockSize)(/* Args */);
```

I/O is seriously limited

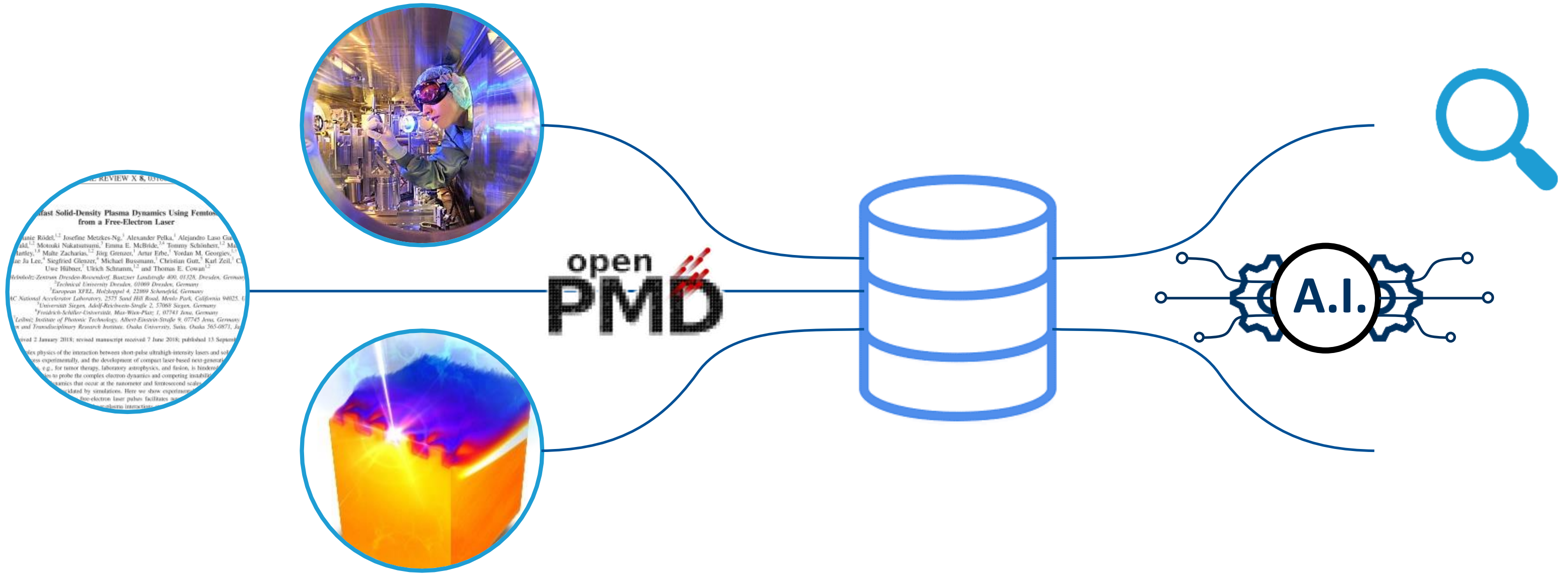
OPENPMD: F.A.I.R. I/O and streaming for the Exascale era





OPENPMD: F.A.I.R. I/O and streaming for the Exascale era

OPENPMD: Streaming workflows for Analysis, Simulation & AI


fast Solid-Density Plasma Dynamics Using Femto…
from a Free-Electron Laser
open
PMD
A.I.

European
XFEL

open
PMD

Tools for the NFDI Data Challenge from HPC to Edge

Open, F.A.I.R. & fast

alpaka

cupla

LLAMA

openPMD

DAPHNE
4NFDI

PUNCH
4NFDI