LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

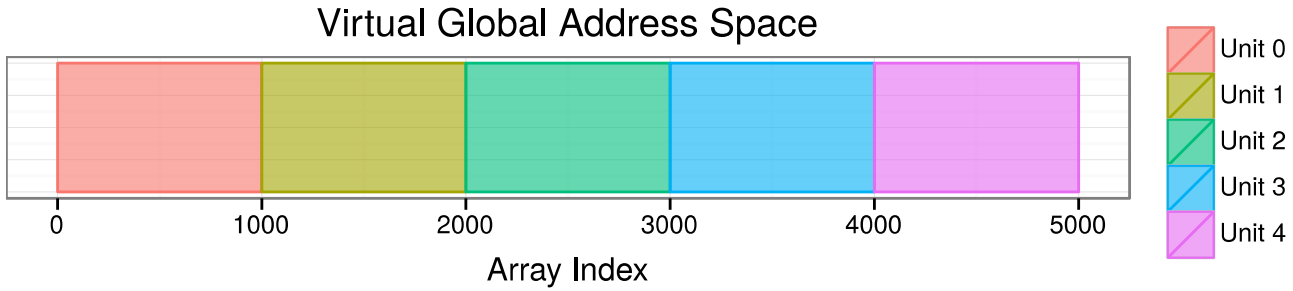
A Parallel I/O Abstraction for Partitioned Global Address Space Programming

www.dash-project.org

Felix Mößbauer
felix.moessbauer@campus.lmu.de
Ludwig-Maximilians-Universität München

PGAS - Overview

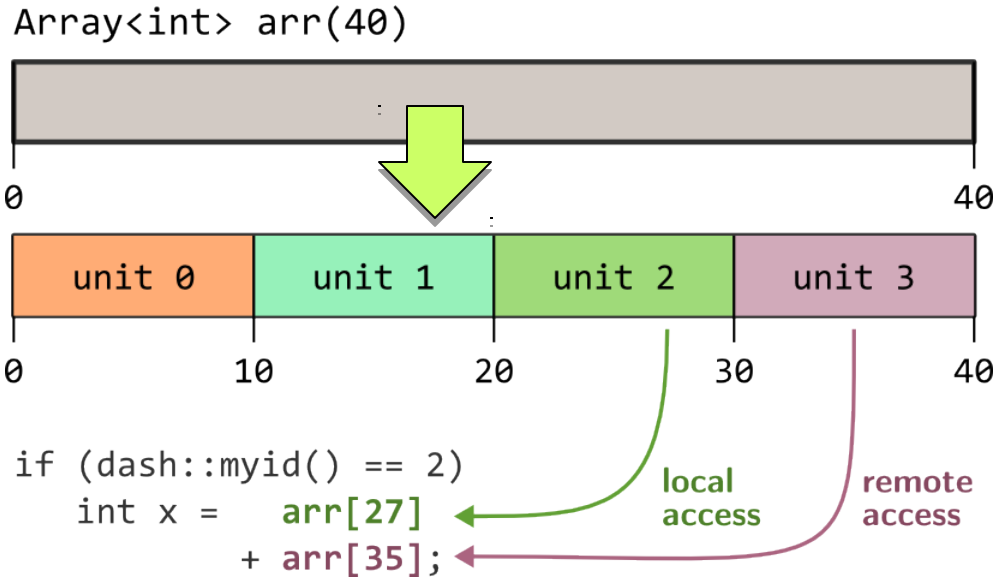
- PGAS
  - Data is distributed over units
  - Global data structures know its layoutDASH
- DASH
  - C++ library similar to UPC++
  - Implements PGAS approach
  - Provides distributed data structures and algorithms



DASH - Partitioned Global Address Space

- Data affinity
  - data has well-defined owner but can be accessed by any unit
  - data locality important for performance
  - support for the *owner computes* execution model

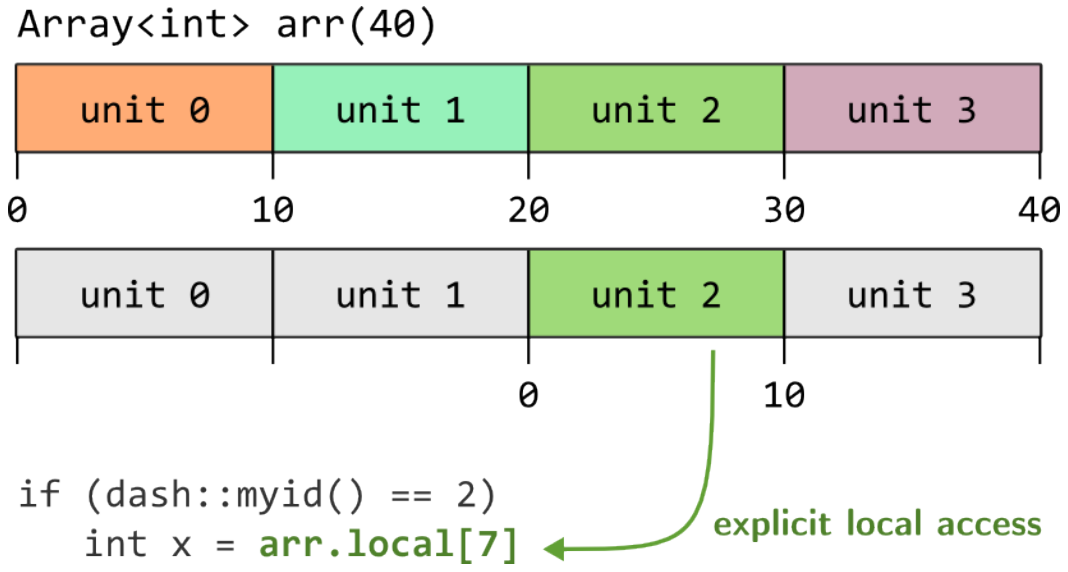
- DASH:
  - unified access to local and remote data in global memory space



DASH - Partitioned Global Address Space

- Data affinity
  - data has well-defined owner but can be accessed by any unit
  - data locality important for performance
  - support for the *owner computes* execution model

- DASH:
  - unified access to local and remote data in global memory space
  - and explicit views on local memory space



IO Abstraction Basics

- Why IO abstraction necessary
  - IO is often boilerplate
  - Parallel IO is tricky
  - Re-implementation of IO part in each application
- Abstraction idea
  - Import / export distributed data structures
  - various targets (HDF5, NetCDF, Image, ...)
  - Conversion ?

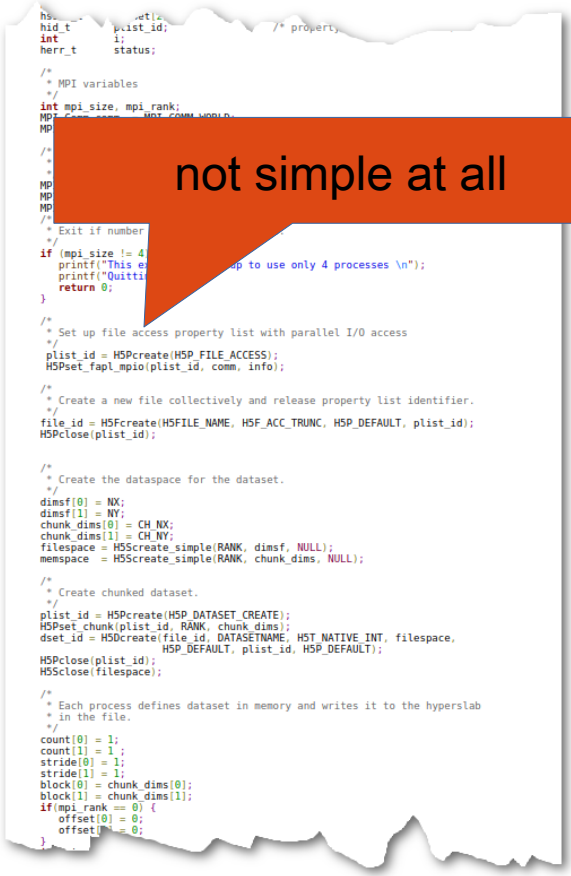
Design Goals

- Support portable, common data formats
- Implementation is portable (GPFS, NFS, ...)
- Minimal overhead
- Scalable
- Simple interface

does not scale

```
// distr  uted data structure
distdata data;
if(rank == 0){
  load_data(&data, file);
}
MPI_Barrier();
```

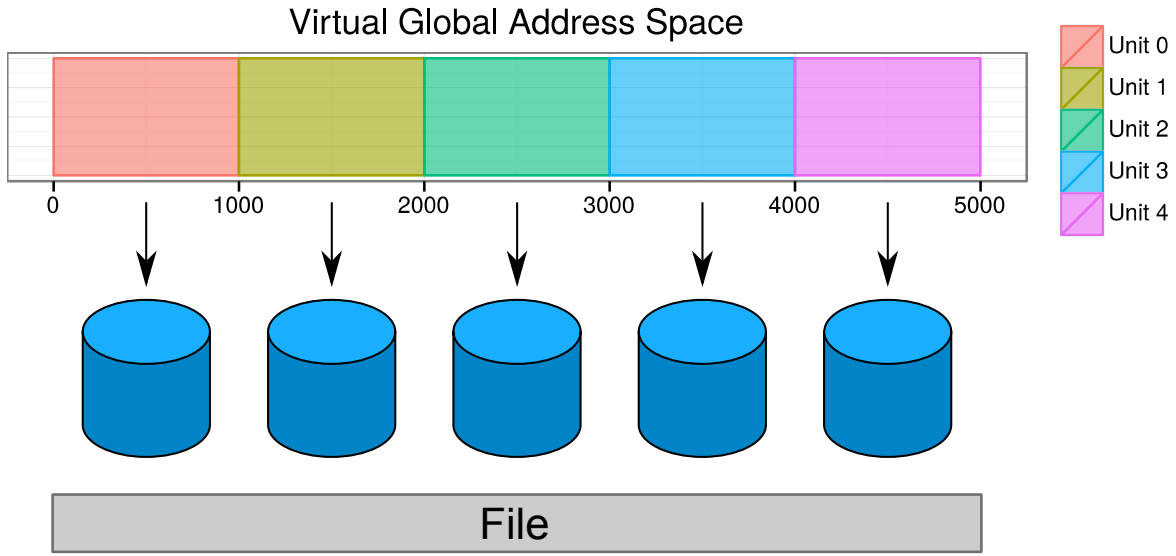
not simple at all



Code: HDF5 Example Hyperslab_by_chunk.c

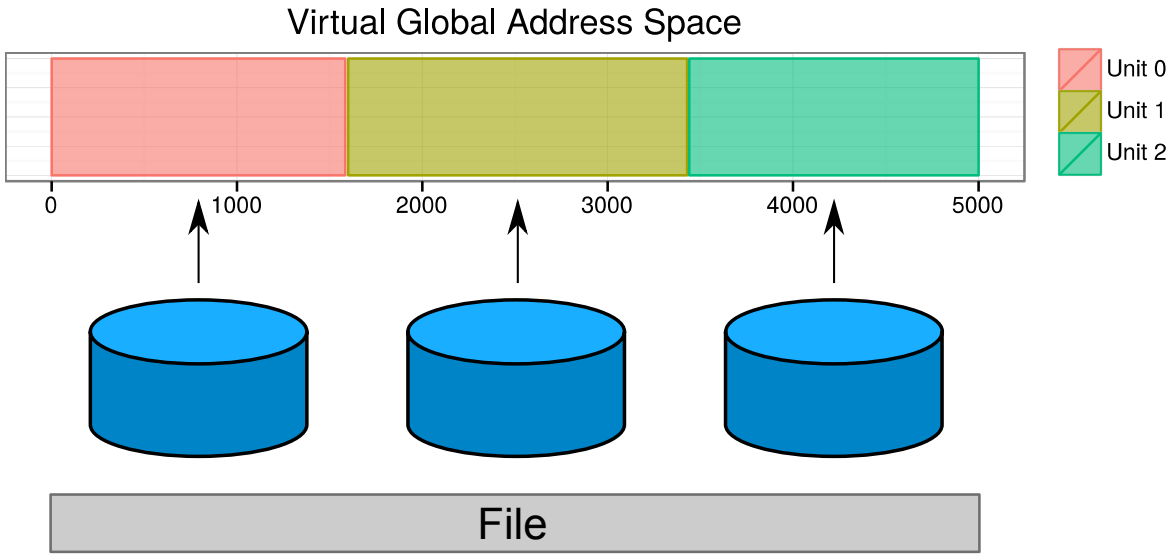
IO Abstraction Concept

- Similar to POSIX pipes
- Source → IO Adapter → Sink
- Collective operation



IO Abstraction Concept

- Similar to POSIX pipes
- Source → IO Adapter → Sink
- Collective Operation



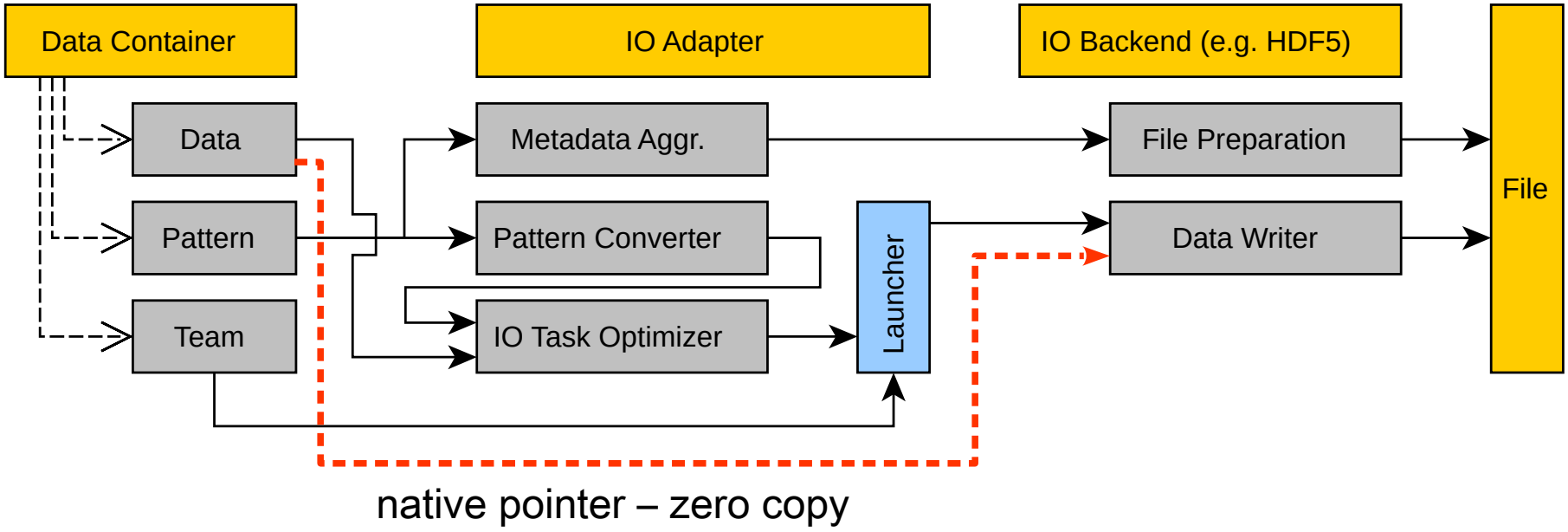
IO Adapter Basics

- Configuration using stream modifiers e.g.
  - dataset
  - store pattern
  - restore pattern
  - ...

```
namespace dio = dash::io::hdf5;
dash::NArray<int,3> data(10,50,30);
// [fill matrix]
dio::OutputStream os("filename.hdf5");
os << dio::dataset("path/to/dataset");
// kicker which passes configuration to io-adapter
os << data;
```

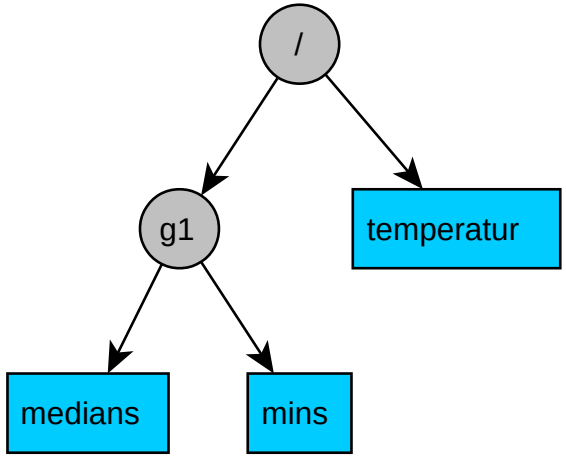
- Good defaults
- IO Team is derived from container
- Automatic type conversion (trivial types)
- User can pass custom type mapping function as lambda

IO Abstraction Model



Reference Implementation

- DASH: distributed datastructures
- HDF5
  - portable data format
  - Independent of OS / Arch.
  - Self-describing format
  - Organized as a tree
- Dataset consists of
  - Tables
  - Metadata
  - Attributes



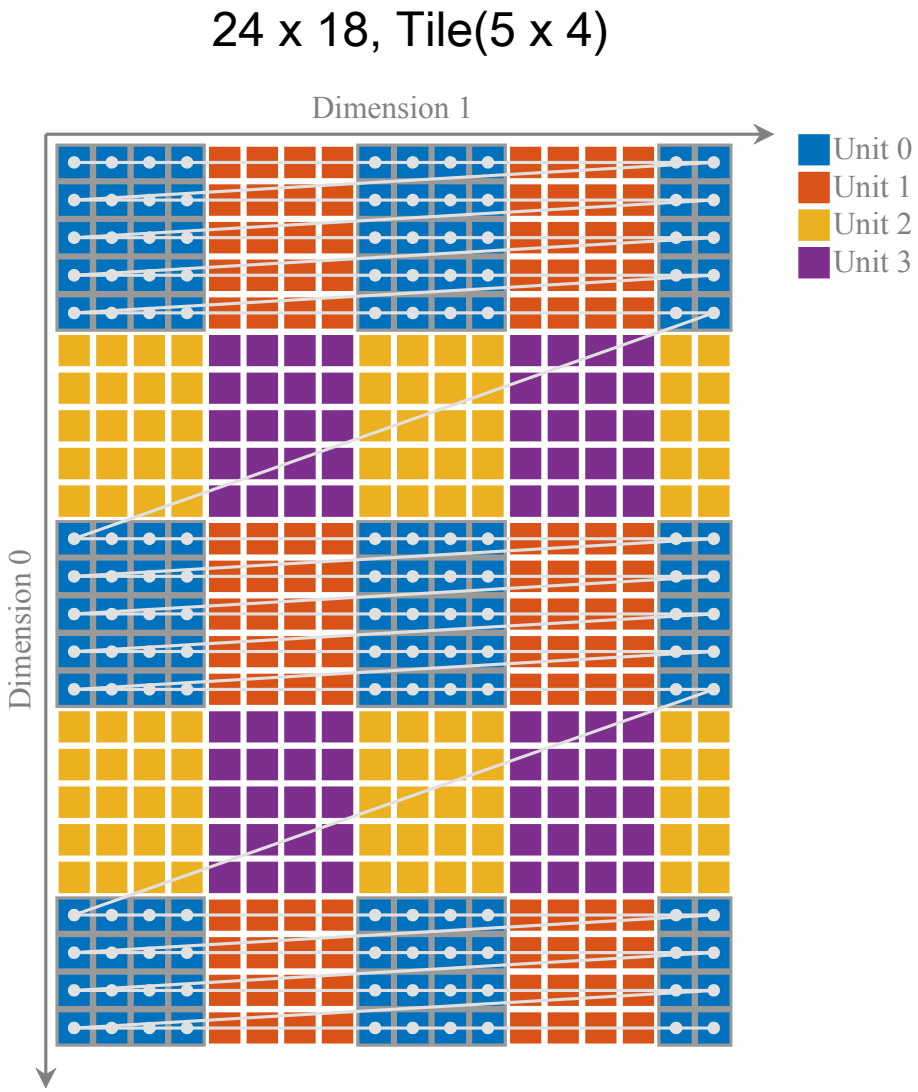
Use *h5dump < file >* to discover

HDF5 Sample File

```
DATASET "medians" {
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( 4, 4 ) / ( 4, 4 ) }
   DATA {
   (0,0): 0.2, 0.4, 0.4, 0.6,
   (1,0): 0.4, 0.2, 0.6, 0.6,
   (2,0): 0.6, 0.6, 0.2, 0.4,
   (3,0): 0.6, 0.6, 0.4, 0.2
   }
   ATTRIBUTE "DASH_PATTERN" {
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SIMPLE { ( 8 ) / ( 8 ) }
      DATA {
      (0): 4, 4, 4, 1, 4, 4, 1, 1
      }
   }
}
```

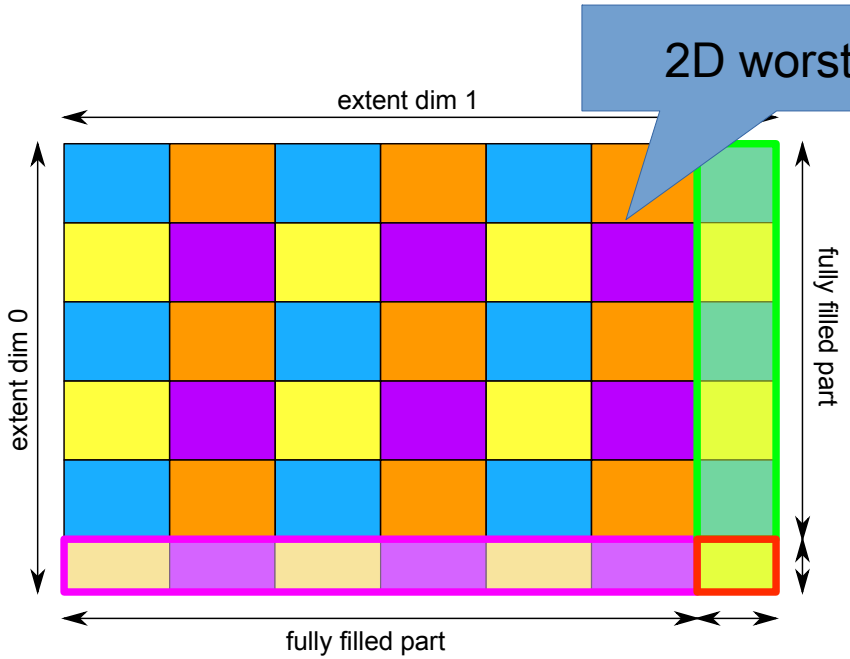
Implementation Aspects - Datastructure

- Container knows distribution
- Data is continuous memory
- Data domain is hyper-rectangle
  - N-Dim rectangle
  - Sample: 2 dim
  - Propably underfilled blocks

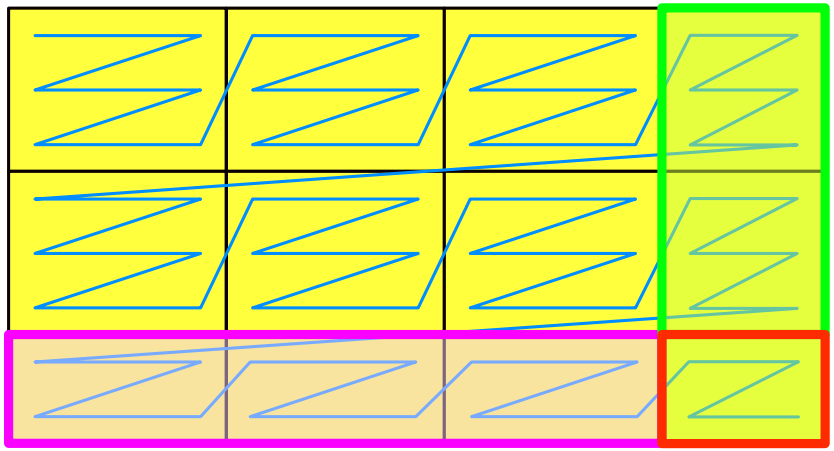


IO Adapter – Pattern Converter

- IO Subtask
  - Has to be executed collectively
  - Size of contribution might differ
  - Order of tasks irrelevant for correctness
  - No underfilled blocks

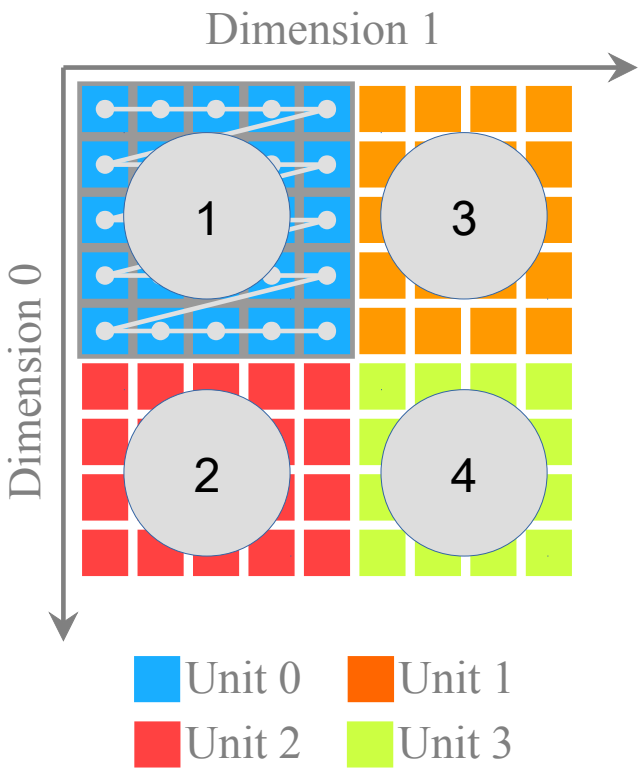


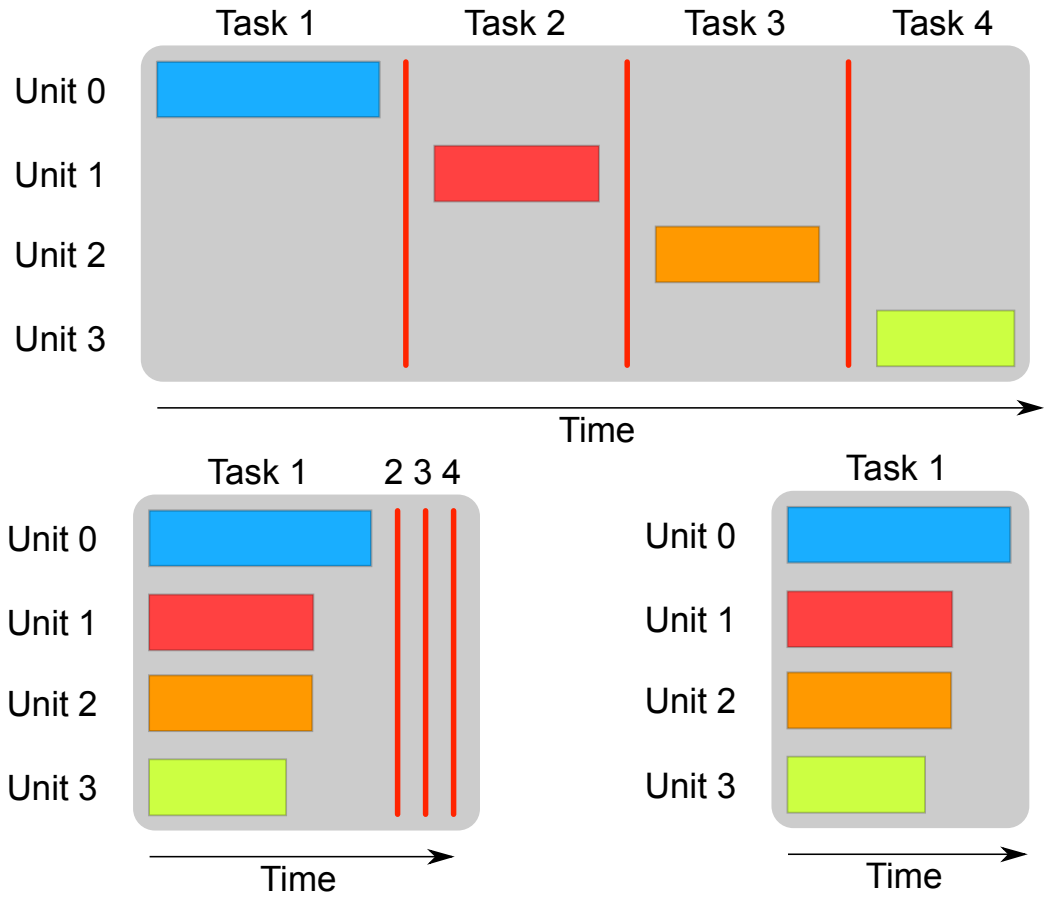
Local memory layout (yellow unit)



Scheduler Optimization

- Order tasks by local contribution
- Drop all-empty tasks





IO Adapter – Async IO (1)

- Observations
  - IO tasks are long-running
  - IO libs internally use MPI2-IO / ROMIO
  - No support for non-blocking IO
- Solution
  - Use MPI multithreading
  - Create IO tasks as threads
  - Works for versions >= MPI2
- Problem
  - Simultaneous IO tasks are horribly slow
  - Buggy in some MPI versions (e.g. mpich)

IO Adapter – Async IO (2)

- Simultaneous tasks are chained internally
- Sync / async selection via launch-policies
- Example:

```
dash::Array<double> array_a(ext_x);
dash::Array<double> array_b(lext_x);
dash::Array<double> array_c(ext_x);

// Fill [...]

OutputStream os(dash::launch::async, "file.hdf5");
os << dio::dataset("array_a") << array_a << dio::dataset("g1/array_b")
   << array_b << dio::dataset("g1/g2/array_c") << array_c;

// Do some computation intense work
// wait for completion of IO
os.flush();
```

Launch policy

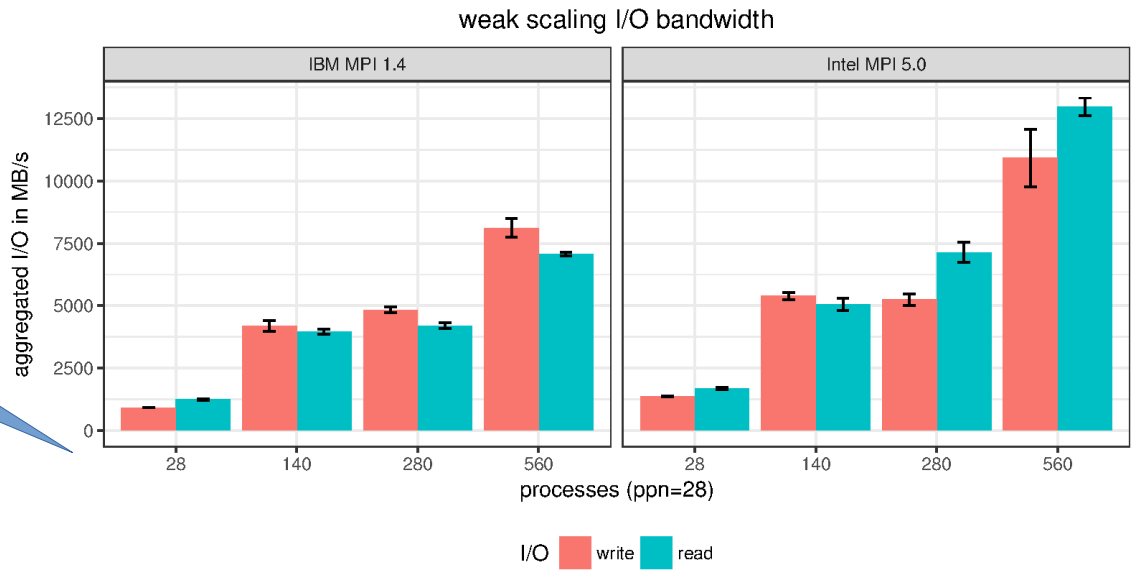
Compatible Patterns

- Each pattern has tags
  - Layout (blocked, row_major, ...)
  - Mapping (balanced, shifted, ...)
  - Partitioning (balanced, regular, ...)
- Checked using type traits at compile time
  - Zero overhead at runtime
- At runtime using exceptions
  - Important for reading data
  - Not all pattern support underfilled blocks

Performance Evaluation

- Minimal overhead (in pattern conversion)
- zero-copy of data
- Performance depends on FS
- Large tiles are better
- SuperMUC Intelmpi, HDF5 on GPFS
  - Read: 1.2 GB/s per node
  - Write: 0.8 GB/s per node

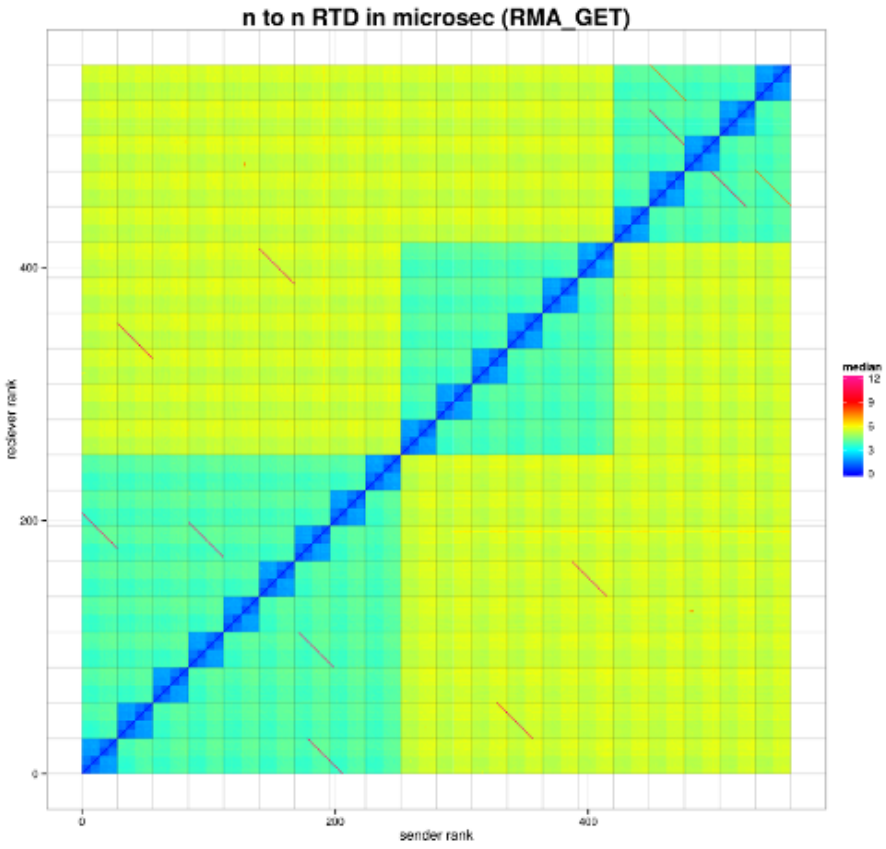
without FS tuning,
300 MB per process



Sample Application

- All-Pairs latency benchmark
  - Measures n-to-n latency multiple times
  - Data is stored in Narray
- Post Processing
  - Import HDF5 data in R
  - Analyze and Plot

SuperMUC II, 20 Nodes



On-going and Future Work

- Subsetting Data Structures
- Adios Bindings
  - Instant support of various IO backends
  - Converter DASH Pattern → Adios Pattern
- DASH Coarray (Similar to CAF 2008)
  - Fulfills the Narray container concept → IO works out of the box

Summary

- IO Abstraction
  - Independent of data and grid layout
  - Easy to use, but fast
  - No special target FS
- Use cases
  - Import and export data
  - Checkpointing
- Reference Implementation part of DASH

Acknowledgements

DASH on GitHub:
https://github.com/dash-project/dash

IO stuff part of release 0.3.0

- Funding

- The DASH Team

T. Fuchs (LMU), R. Kowalewski (LMU), D. Hünich (TUD), A. Knüpfer (TUD), J. Gracia (HLRS), C. Glass (HLRS), H. Zhou (HLRS), K. Idrees (HLRS), J. Schuchart (HLRS), F. Mößbauer (LMU), K. Fürlinger (LMU)