AN INTRODUCTION TO HDF5 IN HPC ENVIRONMENTS

- IDEAS TO BYTES AND BACK -

GERD HEBER, THE HDF GROUP

2020-06-05

TABLE OF CONTENTS

WHY

HDF5 is our contribution to making the management of certain large complex data sets as simple as possible, but not simpler.

A PROMISE

At the end of this presentation you will know:

- The shortest path to being productive w/ HDF5
- The HDF5 building blocks which seem familiar but set it apart
- How to spot & diagnose performance issues in the use of HDF5
- Where to find help
- There's a lot more to discover

DAILY GRIND - A SECOND GLANCE

- Problems you might have come across
- How an HDF5-based approach is different

"JUST DO IT!"

I put my ideas into memory. Now, just…

```
favorite_color = { "lion": "yellow", "kitty": "red" }
# Save a dictionary into a pickle file.
import pickle
pickle.dump( favorite_color, open( "save.p", "wb" ) )
...
# Load the dictionary back from the pickle file.
favorite_color = pickle.load( open( "save.p", "rb" ) )
```

Source: wiki.python.org/moin/UsingPickle

Phew, one less thing to think about!

Not quite: Names? Performance? Portability? …*-lity?*

ML TRAINING SET

A tarball containing $\mathcal{O}(10^8)$ of small (<64KB) images.

WANTED: *Random access to the images.*

Why not just "use the file system?"

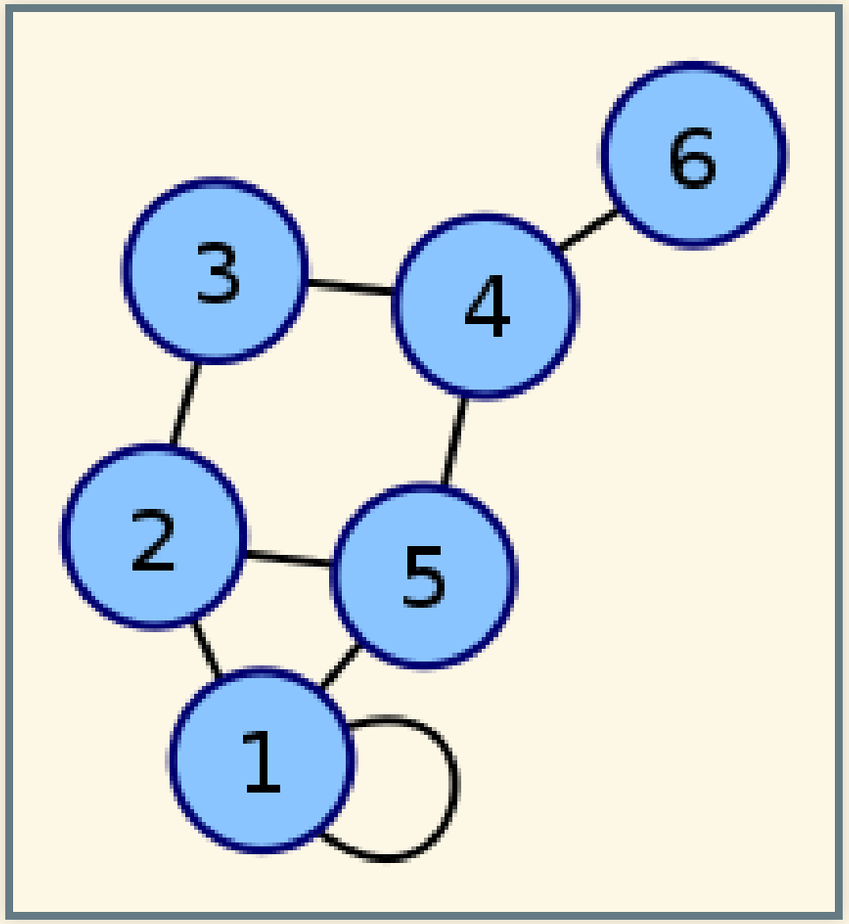
```
tar -xvf images.tar # => Meet your friendly admin staff!
```

Ideas

- Use libarchive and have a virtual tape drive
- Can we have some kind of "file system in a file" (?)
- Or a long byte stream plus a "marker array"
- …

STORING A GRAPH

Which one is "right?" (It depends…)



$$\begin{bmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Source: Wikipedia

Graph database?

HDF5

- **HDF5 let's you map application- to storage-abstractions**
  - Pragmatic: many mappings are possible
- **Provides data portability (HPC, Cloud, Edge, …)**
  - No matter what the hardware or OS
- **Longevity++**
  - Open specifications and Free Open Source Software
- **HDF5 is a "trade-off toolkit"**
  - Find your *sweet-spot*
  - Let's you adapt when circumstances change
- **It cannot solve all of your problems.**

I can't wait to get started! How?

"ZERO KNOWLEDGE" HDF5 PRODUCTIVITY

Oxymoron?

PYTHON, JULIA, R, MATLAB, ...

```
 1: # My weather station...
 2: temperature = np.random.random(1024)
 3: wind = np.random.random(2048)
 4: f = h5py.File('weather.hdf5')
 5: f["/15/temperature"] = temperature
 6: f["/15/temperature"].attrs["dt"] = 10.0
 7: f["/15/wind"] = wind
 8:
 9: dataset[0:10:2]
10:
11: big_dataset = f.create_dataset("big",
12:                                shape=(1024, 1024, 1024, 512),
13:                                dtype='float32')
14: big_dataset[344, 678, 23, 36] = 42.0
15:
16: compressed_dataset = f.create_dataset("comp", shape=(1024,),
17:                                       dtype='int32',
18:                                       compression='gzip')
```

HDF5 just blends in - no need to learn an API!

Exascale?

A NEW LANGUAGE

C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly today than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster. (B Stroustrup, A Tour of C++, 2018)

A NEW SPIRIT

- **A new body**
  - Steven Varga (aka "Canada Dry")
  - Independent developer and researcher
  - H5CPP: Main Page, GitHub
- **Values**
  - *Idiomaticity* - modern C++17
  - *Parsimony* - four basic CRUD-like templates
  - *Openness* - batt. included, e.g., LinAlg. systems
  - *Performance* - zero overhead over the C-API
  - *Productivity* - zero HDF5-boilerplate

REMEMBER `pickle`?

```
favorite_object = ...
pickle.dump( favorite_object, open( "save.p", "wb" ) )
...
favorite_object = pickle.load( open( "save.p", "rb" ) )
```

Let's do it!

```
std::vector<double> v(10); // do something interesting with v
h5::write("save.h5","stl/vector/v", v); // one-shot write ...
...
using T = std::vector<double>;
auto v = h5::read<T>("save.h5","stl/vector/v"); // ... and read
```

Easy.

PYTHON CONVENIENCE AT THE SPEED OF C(++17)

```
 1: fvec temperature = arma::randu<fvec>(1024);
 2: fvec wind = arma::randu<fvec>(2048);
 3: auto fd = h5::create("weather.hdf5");
 4: auto ds = h5::write(fd, "/15/temperature", temperature);
 5: ds["dt"] = 10.0f;
 6: h5::write(fd, "/15/wind", wind);
 7:
 8: h5::read<fvec>(ds, h5::offset{0}, h5::count{5}, h5::stride{2});
 9:
10: auto big = h5::create<float>(fd,"big",
11:                              h5::current_dims{1024, 1024, 1024, 512},
12:                              h5::chunk{16, 16, 16, 8});
13: float value {42.0};
14: h5::write<float>(ds, &value,
15:                  h5::offset{344, 678, 23, 36},
16:                  h5::count{1, 1, 1, 1});
17:
18: auto comp = h5::create<int>(fd, "comp",
19:                             h5::current_dims{1024},
20:                             h5::chunk{64} | h5::gzip{4});
```

You are **not** expected to absorb this in 10 seconds.

Just note the similarity w/ Python & Co.

ANOTHER ACE UP OUR SLEEVE: COMPILER ASSISTANCE

Source: ALA

This is a presentation for beginners…

TAKE-AWAY

- Little or no knowledge of HDF5 isn't a showstopper.
- The community went the *last mile* to HDF5 productivity.
- If your code looks like C, it's either C or you need help.

But what *is* HDF5?

HOW TO SPEAK HDF5

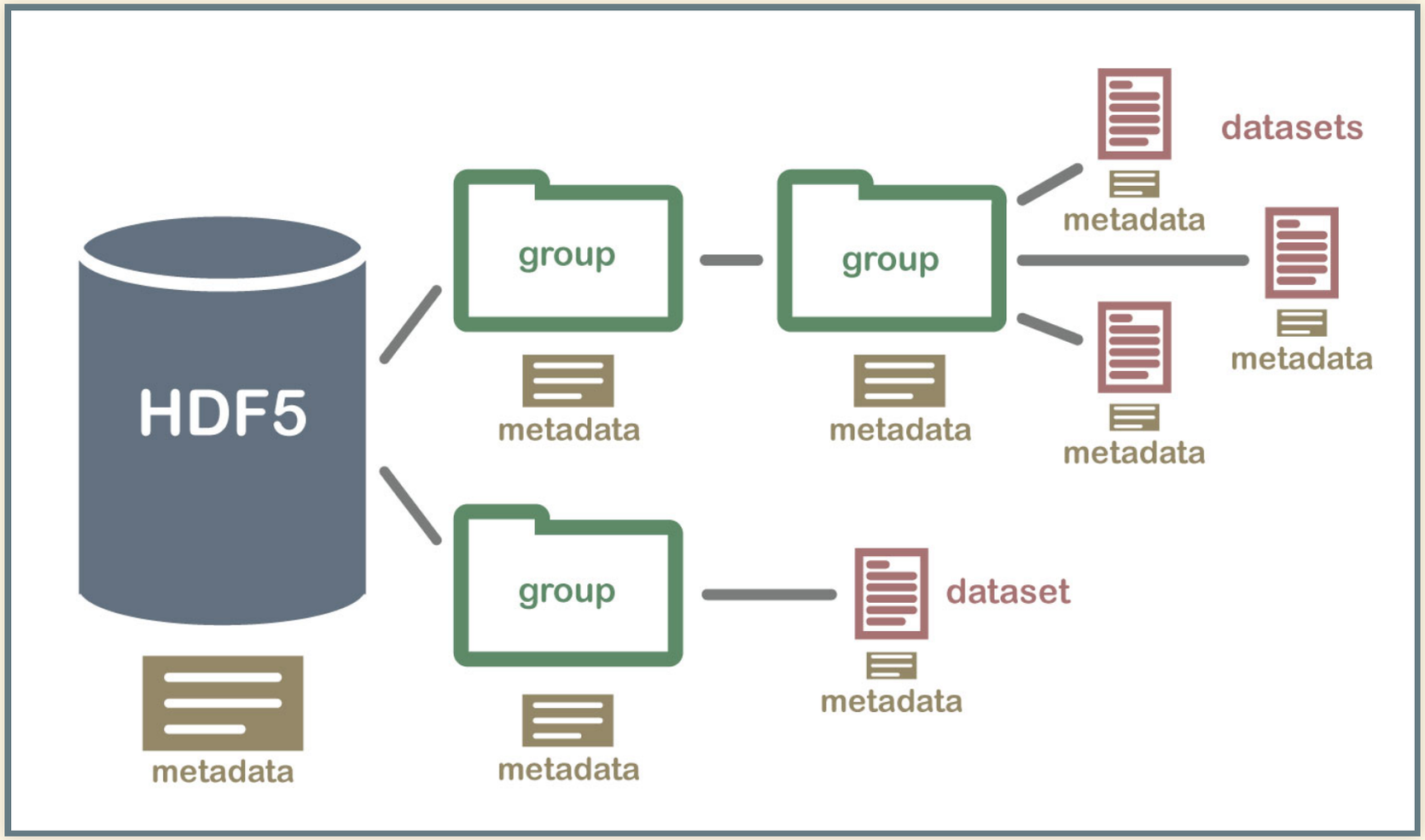
WHAT WE HAVE SEEN SO FAR

As keen observers, you will have noticed:

- HDF5 things are stored in files(?)
- We can refer to them by path names
  - Directories?
- We can use it to store arrays
  - Random access
  - Slice and stride
- We can tag arrays
  - Key-value pairs(?)
- *"Mr. Holmes, these are the footprints of a gigantic file system!" (Dr. Mortimer)*

Source: Wikipedia

HDF5 LOOKS A BIT LIKE THIS



Source: The National Ecological Observatory Network

HDF5 DATA MODEL

An HDF5 "file" is a rooted, decorated web of array variables.

- **One core concept:** (discretized) function
  - *The rest deals with the presentation of such functions.*
  - CS: grid function "=" array
- An array-valued variable is called *array variable*.
  - Element type, rank, and extent
- HDF5 array variables are used in two roles:
  1. *Datasets* - when linked in associations called *groups*
  2. *Attributes* - as named decorators of groups or datasets
- Each HDF5 "file" has a designated *root* group.

HDF5 IN ACTION

- **Basics**
  - Access HDF5 objects through path names (`/My/cool/Object`)
  - HDF5 attributes give context (`SAMPLING_RATE`)
- **Flexible type system**
  - An extensive built-in collection of (non-)numerical types
  - Use type-generators (e.g., records, arrays) to extend
    - Use sparingly
- **Slicing and dicing of array variables**
  - "Generalized block access" via *hyperslabs* and point sets
  - Partial I/O - limit ops. to certain subsets or record fields

TAKE-AWAY

- Like any useful data model, HDF5 is small.
  - (Implementation is another matter)
- You need to learn how to express your ideas in this "language."

How does all that relate to HPC?

PARALLEL HDF5

THE BASIC STRUCTURE OF AN MPI PROGRAM

```
 1: int main(int argc, char** argv) {
 2:   // <<MPI_preamble>>
 3:   int size, rank;
 4:   MPI_Init(&argc, &argv);
 5:   MPI_Comm_size(MPI_COMM_WORLD, &size);
 6:   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 7:   //================================
 8:
 9:   // Do something useful depending on size and rank!
10:   // We will do something with HDF5.
11:   // We will let multiple MPI ranks write to the same or
12:   // different HDF5 files.
13:
14:   // <<MPI_postamble>>
15:   MPI_Barrier(MPI_COMM_WORLD);
16:   MPI_Finalize();
17:   return 0;
18: }
```

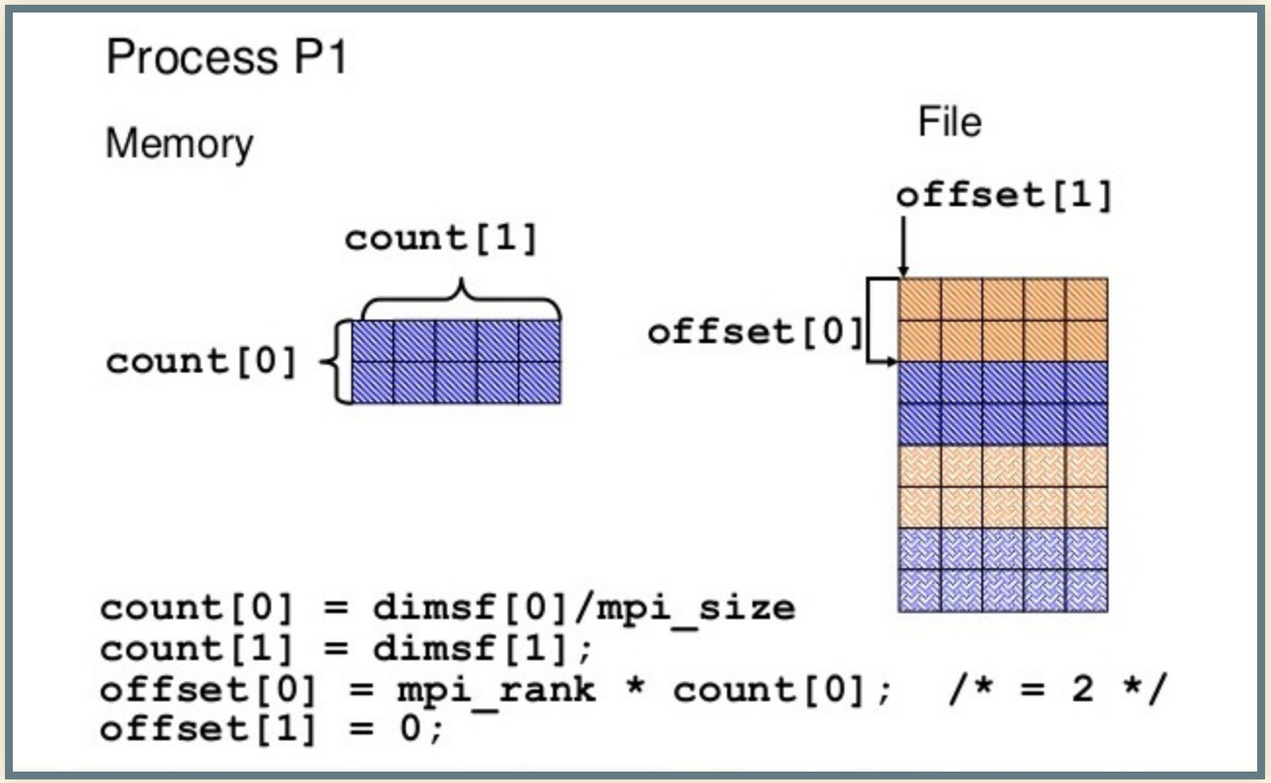
We won't repeat the MPI boilerplate.

PARALLEL I/O RECAP

- Parallel access paths
- Parallel file systems
  - Data striping
  - Replication
  - Fast interconnect
- Process coordination
  - Independent (no)
  - Collective (yes)
- File patterns
  - N-1, N-N, N-M (N>M)
- "Mechanics"
  - I/O forwarding
  - Tiered storage (e.g., burst buffers)
- Software "layer cake"
  - *Murphy's law applies*

LET'S TRY THIS

Let's create a 2D HDF5 dataset and write to it, in parallel!



(View from MPI process 1.)

A SEQUENTIAL PROGRAM WOULD LOOK LIKE THIS

```
 1: ...
 2: // Create a vector and initialize
 3: std::vector<double> v(100);
 4: std::fill(std::begin(v), std::end(v), 1);
 5:
 6: // create an HDF5 file
 7: auto fd = h5::create("sequential.h5", H5F_ACC_TRUNC);
 8:
 9: // write a (2,50) block at offset (2,0) of a (size,50) array
10: h5::write(fd, "dataset", v, h5::current_dims{2*size, 50},
11:           h5::offset{2, 0}, h5::count{2, 50});
12: ...
```

A parallel version would run this program (adjusted for `rank`) on all processes of an MPI communicator.

SYNTAX IS THE EASY PART

```
 1: ...
 2: // I'm an MPI rank in a size MPI_COMM_WORLD
 3: // Create a vector and initialize
 4: std::vector<double> v(100);
 5: std::fill(std::begin(v), std::end(v), rank);
 6:
 7: // create and open a file with MPI-IO
 8: auto fd = h5::create("parallel.h5", H5F_ACC_TRUNC,
 9:                      h5::mpiio({MPI_COMM_WORLD, MPI_INFO_NULL}));
10:
11: // write my data portion
12: h5::write(fd, "dataset", v, h5::current_dims{2*size, 50},
13:           h5::offset{2*rank, 0}, h5::count{2, 50});
14: ...
```

- Each process writes a `(2,50)` block at offset `(2*rank,0)`.
- All "magic" happens in line 9 (plus the adjustment in line 13).
- It's OK to write a 1D array to a 2D array.
- Ditto for reading the data back.

WAIT A MINUTE...

1. Any difference between sequential and parallel HDF5 files?
    - Nope. *There's only one HDF5 file format.*
2. Are the `h5::[read,write]` calls collective or independent?
    - The default behavior is independent I/O.
    - Collective I/O can be configured as follows:

```
1: h5::write(fd, "dataset", v, h5::current_dims{size, 50},
2:           h5::offset{2*rank, 0}, h5::count{2, 50},
3:           h5::collective);
```

    (See line 3.) Easy.

3. *Why do my friends keep saying that parallel I/O is hard?*

IT'S A BALANCING ACT

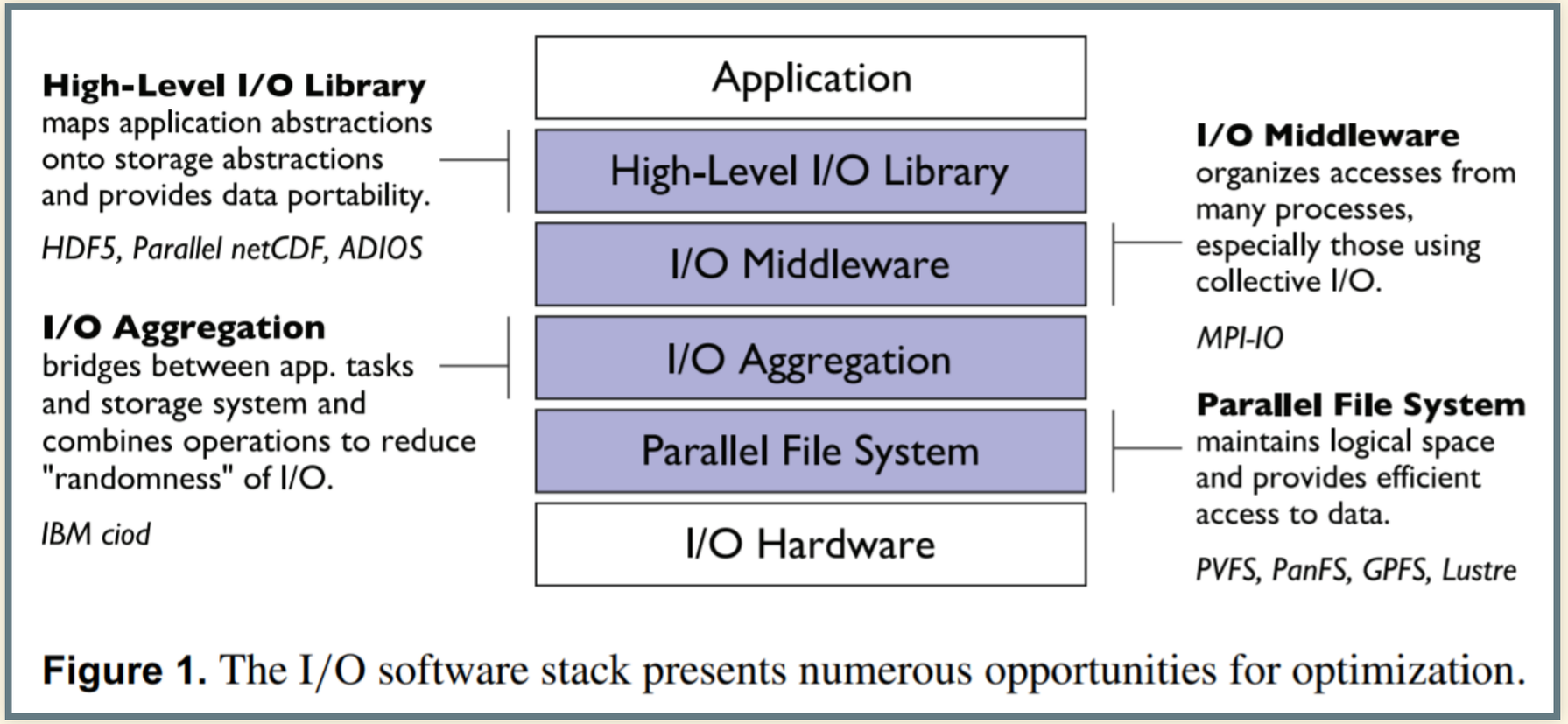


Figure 1. The I/O software stack presents numerous opportunities for optimization.

Source: Latham et al. *A case study for scientific I/O: improving the FLASH astrophysics code* - https://doi.org/10.1088/1749-4699/5/1/015001

METHOD

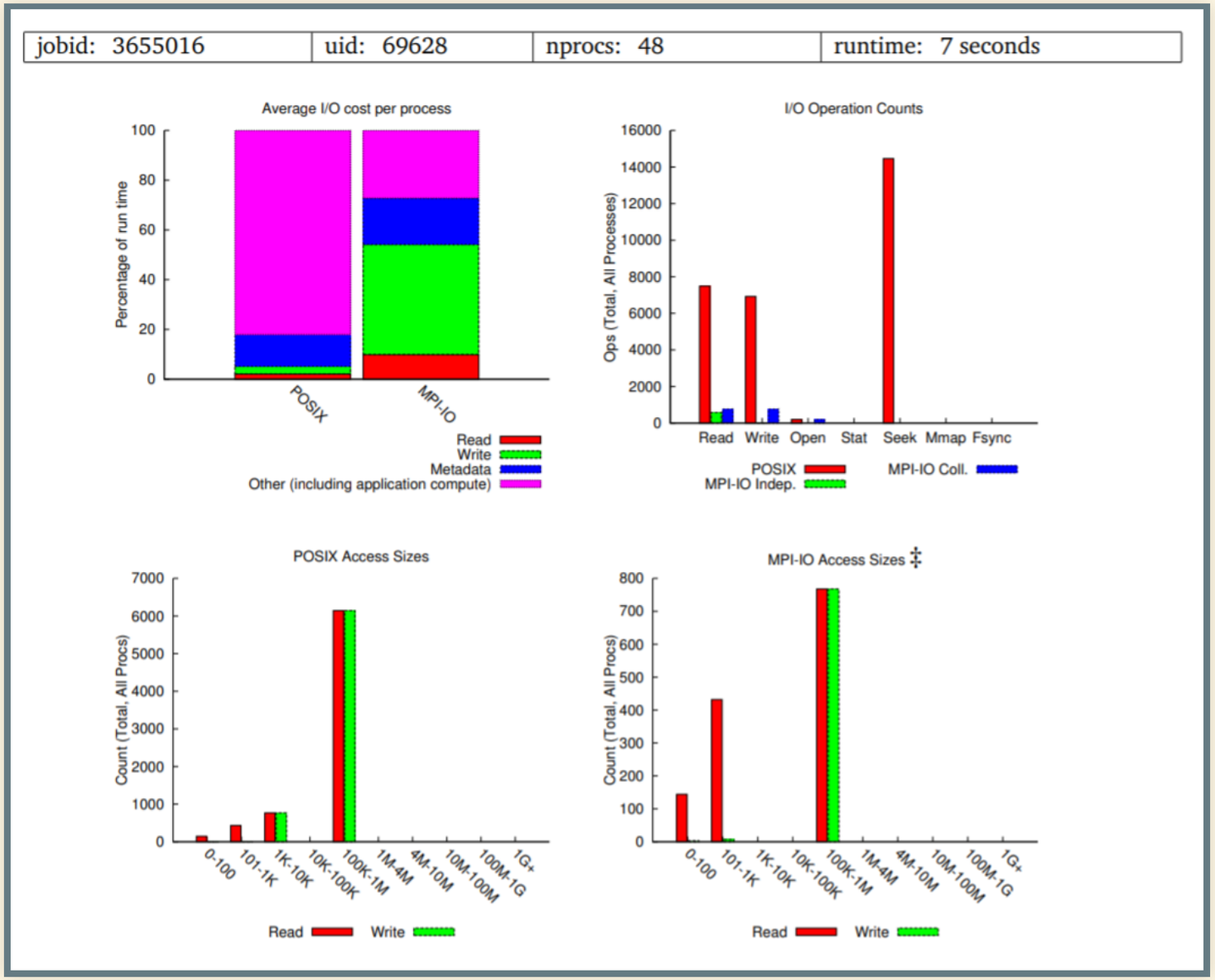
You can't improve what you can't measure.

1. Establish target system's characteristics
2. Baseline your application
3. Data collection
4. Hypothesis formulation
5. Configuration change - *One at a time!*
6. Hypothesis confirmation/rejection/put aside
7. Stop ? `DONE` : `GOTO 3`. (⇐ Can be difficult!)

APPLYING THE METHOD

- **Establish your target system's characteristics**
  - Published acceptance test results
  - Flexible I/O tester (FIO)
  - Other fine choices, e.g., IOR (see General I/O References)
- **Baseline your application**
  - Back-of-the-envelope calculations
  - Pick a typical set of inputs (to your application)
  - Use a tool such as Darshan or TAU

VISUALIZE



Source: Darshan-util installation and usage

TELLTALE SIGNS

Learning how to read a Darshan profile is time well spent.

A list of "usual suspects:"

- A large proportion of small (< 4KB) reads and writes
- A large gap between the fastest and slowest ranks
  - A large variance in I/O size and time between ranks
- A large number of `seek` operations or unaligned operations
- Non-collective I/O where collective was expected

TAKE-AWAY

- **The HDF5 library makes parallel applications *and* data portable**
  - Tools such as H5CPP make it convenient
- **Performance portability is an *aspiration***
  - Remember the "layer cake!"
- **Arm yourself with method and the right tools!**
  - Stay tuned for an upcoming tutorial on that topic!

Where do I go from here?

ONWARDS AND UPWARDS

You can go on to make your own mistakes.

We've already done it for you!

To make original mistakes is harder than it seems.

CHECK OUT THESE HDF5 RESOURCES

- Don't miss the "loss-leader" HDF View - download this first!
- The slides and examples from this presentation are on GitHub
- Learn more about The HDF Group and it's mission
- Looking for the HDF Support Portal?
- Binge-watch HDF5 training videos
- Get ideas from the HDF5 Blog, e.g., HDF5 under the SOFA
- Check out HDF5 presentations on SlideShare
- Connect with the community on the HDF forum
- Subscribe to the HDF Newsletter
- There's a vast HDF5 ecosystem

KICK THE TIRES

- Zero installation: HDF Kita Lab
  - JupyterLab environment
  - All HDF5 dependencies pre-installed + plenty of examples
- Easy installation:
  - Python: install `h5py` via package manager (`pip`, `conda`, ...)
  - Julia: `Pkg.add("HDF5")`
  - R: install `rhdf5` via the Bioconductor manager

```
install.packages("BiocManager")
BiocManager::install("rhdf5")
```

- HDFql: Download installers from www.hdfql.com
- H5CPP: Debian and RPM packages from h5cpp.org

GENERAL I/O REFERENCES

- Flexible I/O tester (FIO)
  - Documentation
  - Source
- IOR - HPC IO Benchmark Repository
- A Multi-purpose, Application-Centric, Scalable I/O Proxy Application (MACSio)
  - Documentation
  - Source
- Virtual Institute for I/O - IO500

PREVIEW OF INTERMEDIATE AND ADVANCED TOPICS

This is the first in a series of webinars. Stay tuned for:

- HDF5 for the Cloud
- HDF5 long-term data access
- In-depth HDF5 datatypes
- Compression
- Concurrency
- Referencing and Linkage
- Data and Metadata
- Granularity
- HDF5 library architecture
- HDF5 extensions

THANK YOU

Acknowledgment: This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences under Award Number DE-AC05-00OR22725.

QUESTIONS