HDF5 in the Julia Ecosystem

Mark Kittisopikul, Ph.D.
Software Engineer, SciComp, Janelia, HHMI

HDF5 User Group Meeting
August 16, 2023

HDF5 in the Julia Ecosystem

Why Julia?

An interactive, dynamic, open source, and natively JIT compiled language

HDF5 Julia Packages

Distributing the C library and binding the C API

HDF5.jl Usage

Example of basic usage of the HDF5.jl package

Future directions

Making HDF5 easier to use and abstract interfaces

HDF5 in the Julia Ecosystem

Why Julia?

hhmi | janelia
Research Campus

Why We Created Julia

14 February 2012 | **Jeff Bezanson Stefan Karpinski Viral B. Shah Alan Edelman**

We want a language that's **open source**, with a liberal license.

We want the **speed of C** with the dynamism of Ruby.

We want a language that's homoiconic, with true macros like Lisp, but with obvious, familiar **mathematical notation** like Matlab.

We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell.

Something that is dirt **simple** to learn, yet keeps the most serious hackers happy.

We want it **interactive** and we want it **compiled**.

https://news.mit.edu/2018/julia-language-co-creators-win-james-wilkinson-prize-numerical-software-1226

https://julialang.org/blog/2012/02/why-we-created-julia/

Why Julia?

HPC adoption of Julia

HPC wire

Search… Go

Julia Joins Petaflop Club

September 12, 2017

BERKELEY, Calif., Sept. 12, 2017 — Julia has joined the rarefied ranks of computing languages that have achieved peak performance exceeding one petaflop per second – the so-called 'Petaflop Club.'

The Julia application that achieved this milestone is called Celeste. It was developed by a team of astronomers, physicists, computer engineers and statisticians from UC Berkeley, Lawrence Berkeley National Laboratory, National Energy Research Scientific Computing Center (NERSC), Intel, Julia Computing and the Julia Lab at MIT.

julia

The Exanauts Team

The Argonne team webpage for the Exascale Computing Project (ECP)

ExaSGD

Pinned

ExaPF.jl Public
A Power Flow Solver for GPUs in Julia
Julia ☆ 48 5

ExaTron.jl Public
Julia implementation of TRON solver on GPUs
Julia ☆ 13 4

ProxAL.jl Public
Proximal Augmented Lagrangian solver for solving multiperiod contingency-constrained ACOPF
Julia ☆ 6 3

Argos.jl Public
Reduced-space optimization, for optimal power flow.
Julia ☆ 15 2

ExaData Public
Julia artifact source for ExaSGD power grid data
MATLAB

ExaAdmm.jl Public
Julia implementation of ADMM solver on multiple GPUs
Julia ☆ 13 1

Useful properties of Julia for using HDF5

- Built-in C foreign function interface
  - Procedural form

    ```
    ccall((:H5open, libhdf5), herr_t, ())
    ```

  - Macro form

    ```
    @ccall libhdf5.H5open()::herr_t
    ```

- Multiple Dispatch (multimethods)
- C compatible primitives
  - `julia> Cint`

    `Int32`

- C compatible structs with reflection

```
julia> struct Foo
           x::Float32
           y::Float64
       end

julia> fieldtypes(Foo)
(Float32, Float64)

julia> fieldoffset.(Foo, (1,2))
(0x0000000000000000, 0x0000000000000008)
```

- C function pointers for callbacks

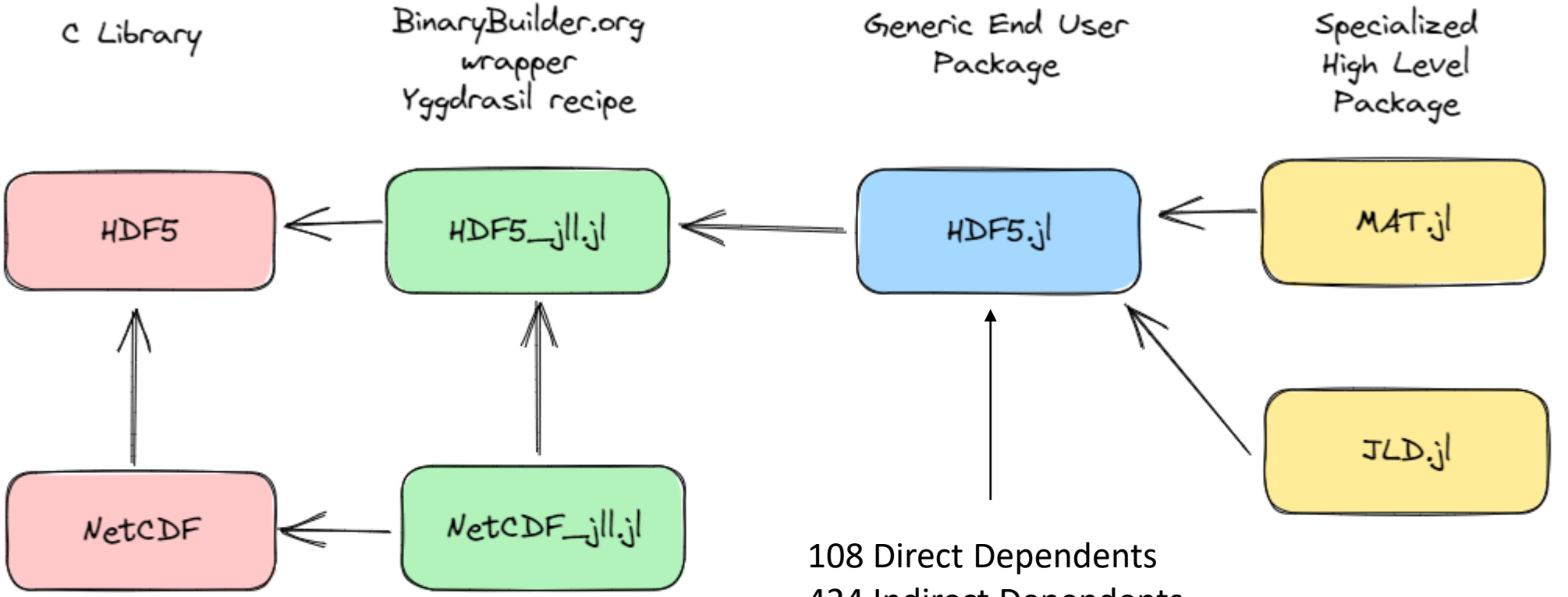
HDF5 in the Julia Ecosystem

HDF5 Julia Packages

hhmi | janelia Research Campus

https://github.com/JuliaPackaging

HDF5 Julia Packages



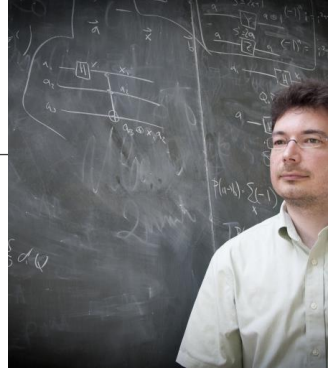
```
julia> println("Hello HUG23")
Hello HUG23

(@v1.9) pkg> activate @HUG23
  Activating new project at `C:\Users\kittisopikulm\.julia\environments\HUG23`

(@HUG23) pkg> add HDF5
   Resolving package versions...
    Updating `C:\Users\kittisopikulm\.julia\environments\HUG23\Project.toml`
  [f67ccb44] + HDF5 v0.16.15
    Updating `C:\Users\kittisopikulm\.julia\environments\HUG23\Manifest.toml`
  [34da2185] + Compat v4.9.0
  [f67ccb44] + HDF5 v0.16.15
  [692b3bcd] + JLLWrappers v1.4.1
  [3da0fdf6] + MPIPreferences v0.1.9
  [21216c6a] + Preferences v1.4.0
  [ae029012] + Requires v1.3.0
  [0234f1f7] + HDF5_jll v1.14.1+0
  [1d63c593] + LLVMOpenMP_jll v15.0.4+0
  [7cb0a576] + MPICH_jll v4.1.2+0
  [f1f71cc9] + MPItrampoline_jll v5.3.1+0
  [9237b28f] + MicrosoftMPI_jll v10.1.3+4
  [fe0851c0] + OpenMPI_jll v4.1.5+0
  [458c3c95] + OpenSSL_jll v3.0.10+0
  [477f73a3] + libaec_jll v1.0.6+1
```

HDF5.jl – Julia API

HDF5_jll.jl – C Library

MPI

SZIP compression



HDF5_jll.jl: Packaging the C Library

Tracking the latest stable release (1.14.1)

192 Binary Tarball Artifacts

Processor architectures:
i686, x86_64, aarch64, armv6l, armv7l, powerpc64le
Operating systems: Windows (MINGW), Linux, macOS, FreeBSD
C standard libraries: glibc, musl
gfortran versions: 3, 4, and 5
Libstdc++ versions: cxx03, cxx11
MPI: mpich, openmpi, microsoftmpi, mpitrampoline

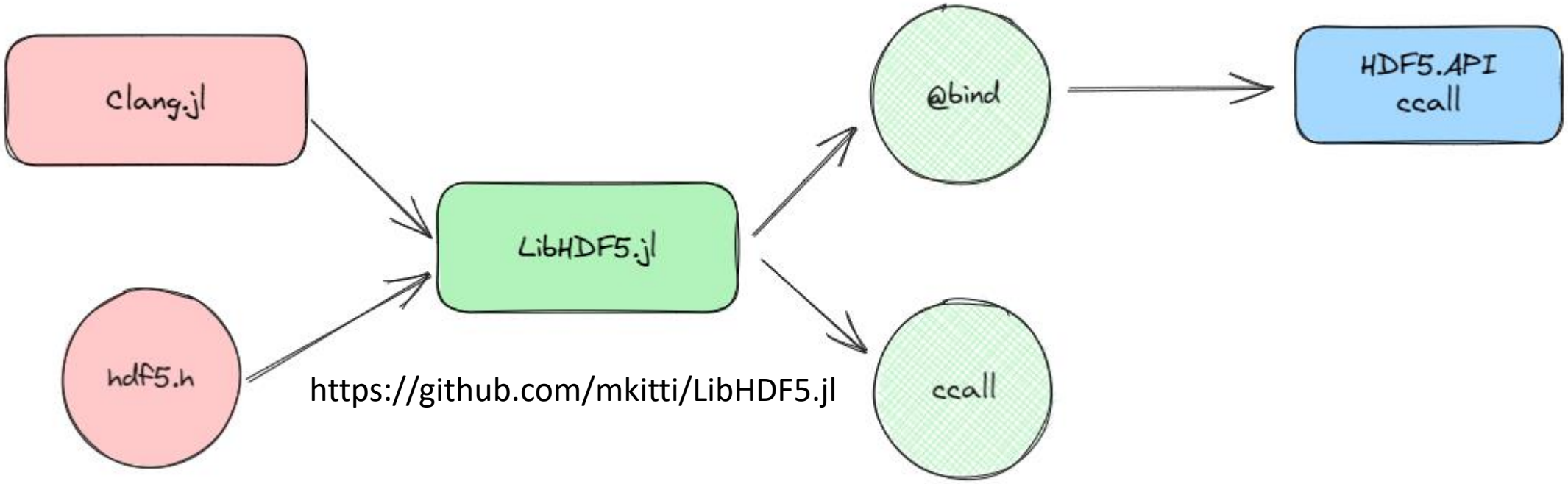
https://github.com/JuliaBinaryWrappers/HDF5_jll.jl

Erik Schnetter
Research Technologies Group Lead
Perimeter Institute for Theoretical Physics
Technology Services

HDF5.jl: Current Maintainers

- **Mustafa Mohamad** (Assistant Professor, UCalgary) is the lead maintainer of HDF5.jl, JLD.jl, and MAT.jl
- **Mark Kittisopikul** (Software Engineer, Janelia, HHMI) has been expanding low-leel API coverage, especially with chunking
- **Simon Byrne** (Lead Software Developer, CliMA, CalTech) has been working on package organization, filter interface, virtual datasets, and parallelization

HDF5.jl: Automated Generation of Low-Level Bindings via LibHDF5.jl



HDF5.API: Low Level API Bindings via ccall

```
julia> function H5open()
           ccall((:H5open, libhdf5), herr_t, ())
       end
H5open (generic function with 1 method)

julia> @code_llvm H5open()
;  @ REPL[29]:1 within `H5open`
; Function Attrs: uwtable
define i32 @julia_H5open_437() #0 {
top:
;  @ REPL[29]:2 within `H5open`
  %0 = call i32 inttoptr (i64 140710727853568 to i32 ()*)()
  ret i32 %0
}
```

```
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        subq    $32, %rsp
        movabsq $140710727853568, %rax
; | @ REPL[29]:2 within `H5open`
        callq   *%rax
        addq    $32, %rsp
        popq    %rbp
        retq
```

HDF5.jl: bind macro statements

```
###
### Dataset Interface
###

@bind h5d_chunk_iter(dset_id::hid_t, dxpl_id::hid_t, cb::Ptr{Nothing}, op_data::Any)::herr_t "Error iterati
@bind h5d_close(dataset_id::hid_t)::herr_t "Error closing dataset"
@bind h5d_create2(loc_id::hid_t, pathname::Cstring, dtype_id::hid_t, space_id::hid_t, lcpl_id::hid_t, dcpl_
@bind h5d_create_anon(loc_id::hid_t, type_id::hid_t, space_id::hid_t, dcpl_id::hid_t, dapl_id::hid_t)::hid_
@bind h5d_extend(dataset_id::hid_t, size::Ptr{hsize_t})::herr_t "Error extending dataset" # deprecated in f
@bind h5d_fill(fill::Ptr{Cvoid}, fill_type_id::hid_t, buf::Ptr{Cvoid}, buf_type_id::hid_t, space_id::hid_t)
@bind h5d_flush(dataset_id::hid_t)::herr_t "Error flushing dataset"
@bind h5d_gather(src_space_id::hid_t, src_buf::Ptr{Cvoid}, type_id::hid_t, dst_buf_size::Csize_t, dst_buf::
@bind h5d_get_access_plist(dataset_id::hid_t)::hid_t "Error getting dataset access property list"
@bind h5d_get_chunk_info(dataset_id::hid_t, fspace_id::hid_t, index::hsize_t, offset::Ptr{hsize_t}, filter_
@bind h5d_get_chunk_info_by_coord(dataset_id::hid_t, offset::Ptr{hsize_t}, filter_mask::Ptr{Cuint}, addr::P
@bind h5d_get_chunk_storage_size(dataset_id::hid_t, offset::Ptr{hsize_t}, chunk_nbytes::Ptr{hsize_t})::herr
@bind h5d_get_create_plist(dataset_id::hid_t)::hid_t "Error getting dataset create property list"
@bind h5d_get_num_chunks(dataset_id::hid_t, fspace_id::hid_t, nchunks::Ptr{hsize_t})::herr_t "Error getting
@bind h5d_get_offset(dataset_id::hid_t)::haddr_t "Error getting offset"
@bind h5d_get_space(dataset_id::hid_t)::hid_t "Error getting dataspace"
@bind h5d_get_space_status(dataset_id::hid_t, status::Ref{Cint})::herr_t "Error getting dataspace status"
@bind h5d_get_storage_size(dataset_id::hid_t)::hsize_t "Error getting storage size"
@bind h5d_get_type(dataset_id::hid_t)::hid_t "Error getting dataspace type"
```

HDF5.API: Low Level API Bindings via ccall

```
"""
    h5d_get_offset(dataset_id::hid_t) -> haddr_t

See `libhdf5` documentation for [`H5Dget_offset`](https://portal.hdfgroup.org/display/HDF5/H5D_GET_OFFSET).
"""
function h5d_get_offset(dataset_id)
    lock(liblock)
    var"#status#" = try
            ccall((:H5Dget_offset, libhdf5), haddr_t, (hid_t,), dataset_id)
        finally
            unlock(liblock)
        end
    var"#status#" == -1 % haddr_t && @h5error("Error getting offset")
    return var"#status#"
end
```

HDF5 in the Julia Ecosystem

HDF5.jl

Usage

hhmi | janelia Research Campus

Graphic artist: https://github.com/cormullion

Basic HDF5.jl Usage

```
using HDF5

# Write a HDF5 file
h5open("mydata.h5", "w") do h5f
    # Store an array
    h5f["group_A/group_B/array_C"] = rand(1024,1024)
    # Store an attribute
    attrs(h5f["group_A"])["access_date"] = "2023_07_21"
end

# Read a HDF5 file
C = h5open("mydata.h5") do h5f
    # Access an attribute
    println(attrs(h5f["group_A"])["access_date"])
    # Load an array and return it as C
    h5f["group_A/group_B/array_C"][:,:]
end
```

Exploring a HDF5 file with HDF5.jl

```
julia> h5f = h5open("mydata.h5")
🗂️ HDF5.File: (read-only) mydata.h5
└─ 📂 group_A
   ├─ 🏷️ access_date
   └─ 📂 group_B
      └─ 🔢 array_C

julia> C  = h5f["group_A"]["group_B"]["array_C"][1:16,1:16]
16×16 Matrix{Float64}:
...

julia> close(h5f)
```

Structs and HDF5 Types

```
julia> struct Foo
           x::Int64
           y::Float64
       end

julia> HDF5.datatype(Foo)
HDF5.Datatype: H5T_COMPOUND {
      H5T_STD_I64LE "x" : 0;
      H5T_IEEE_F64LE "y" : 8;
   }
```

Reading and writing structs

```
julia> h5open("mystruct.h5", "w") do h5f
           h5f["Foo"] = [Foo(1, 3.0)]
       end
1-element Vector{Foo}:
 Foo(1, 3.0)

julia> h5open("mystruct.h5", "r") do h5f
           h5f["Foo"][]
       end
1-element Vector{NamedTuple{(:x, :y), Tuple{Int64, Float64}}}:
 (x = 1, y = 3.0)

julia> h5open("mystruct.h5", "r") do h5f
           read(h5f["Foo"], Foo)
       end
1-element Vector{Foo}:
 Foo(1, 3.0)
```

Chunking and Built-in Gzip Compression Usage

In HDF5.jl version 0.16 we introduced a new general filter keyword allowing for the definition of filter pipelines.

```
using HDF5

h5open("simple_chunked.h5", "w", libver_bounds=v"1.12") do h5f
    h5ds = create_dataset(h5f, "gzipped_data", UInt8, (16,16),
        chunk=(4,4),
        filters=[HDF5.Filters.Deflate()],
        alloc_time = :early
    )
end
```

Compression Filter Plugin Packages

Glue code written in Julia.

- H5Zblosc.jl - Blosc.jl (Thank you, Steven G. Johnson)
- H5Zzstd.jl - CodecZstd.jl
- H5Zlz4.jl - CodecLZ4.jl
- H5Zbzip2.jl - CodecBzip2.jl
- H5Zbitshuffle.jl

Future: Let's figure out how to share these with JLD2.jl!

Chunking and Filter Plugin Usage

```
using HDF5, H5Zzstd

h5open("zstd_chunked.h5", "w", libver_bounds=v"1.12") do h5f
    h5ds = create_dataset(h5f, "zstd_data", UInt8, (16,16),
        chunk=(4,4),
        filters=[ZstdFilter(3)]
    )
end
```

TODO: Use a package extension loading mechanism when CodecZstd.jl is present.

Using External Native Plugin Filters

The HDF5 C library has a filter plugin mechanism. Plugins are shared libraries located in `/usr/local/hdf5/lib/plugin` or as specified by `$HDF5_PLUGIN_DIR`.

```
using HDF5.Filters

bitshuf = ExternalFilter(32008, Cuint[0, 0])
bitshuf_comp = ExternalFilter(32008, Cuint[0, 2])

data_A = rand(0:31, 1024)
data_B = rand(32:63, 1024)

filename, _ = mktemp()
h5open(filename, "w") do h5f
    # Indexing style
    h5f["ex_data_A", chunk=(32,), filters=bitshuf] = data_A
    # Procedural style
    d, dt = create_dataset(h5f, "ex_data_B", data_B, chunk=(32,), filters=[bitshuf_comp])
    write(d, data_B)
end
```

Using MPI + HDF5

Simon Byrne

Load and initialize MPI

```
using MPI, HDF5
MPI.Init()
```

Pass MPI communicator to `h5open`, e.g.

```
h5 = h5open("data.h5", "w", MPI.COMM_WORLD)
```

- Needs to be *collective* (all processes at the same time), with the same arguments.
- File needs to be on accessible from all processes (e.g. on a shared file system if distributed).

HDF5 in the Julia Ecosystem

Future Directions

hhmi | janelia Research Campus

Ease of Use

- **Update documentation, add links to HDF5 Doxygen**
- **Expose efficient iteration interfaces via Channels and co-routines**
- **Use package extensions to automatically load filter plugins**
- **Improve tab completion**

Abstraction

- **Julia's type system has an N-dimensional AbstractArray interface.**
  - **Should a HDF5 dataset implement the AbstractArray interface?**
- **DiskArrays.jl implements abstraction for chunked array read from disk**
  - **Currently implemented by NetCDF.jl and Zarr.jl**
- **Abstract plugin code for use by JLD2.jl and Zarr.jl**

HDF5.jl Early and Recent Contributors

- There are many contributors
- Konrad Hisen initiated Julia's support for HDF5
- Tim Holy and Simon Kornblith were the initial primary authors
- Tom Short, Blake Johnson, Isaih Norton, Elliot Saba, Steven Johnson, Mike Nolta, Jameson Nash
- Justin Willmert improved many aspects C to Julia API interface
- Other recent contributors: t-bltg, Hendrik Ranocha, Nathan Zimmerberg, Joshua Lampert, Tamas Gal, David MacMahon, Juan Ignacio Polanco, Michael Schlottke-Lakemper, linwaytin, Dmitri Iouchtchenko, Lorenzo Van Munoz, Jared Wahlstrand, Julian Samaroo, machakann, James Hester, Ralph Kube, Kristoffer Carlsson

Thank you

hhmi | janelia Research Campus

Mark Kittisopikul, Ph.D.

Software Engineer II
Scientific Computing Software
kittisopkulm@janelia.hhmi.org
https://dot.cards/mkitti