Concurrent HDF5: A Community Contribution Proposal

Explore HDF5 MT-Safe Read

Quincey Koziol, LBNL
koziol@lbl.gov

November 13, 2020

Team Members and Contributors

- SNL: **Lee Ward**, Greg Sjaardema
- LBNL: Suren Byna, Houjun Tang, Tony Li
- The HDF Group: **Chris Hogan**, **John Mainzer**, Elena Pourmal
- LLNL: Mark Miller
- LANL: Brad Settlemyer
- Northwestern University: Kai-yuan Hou

Why Concurrent Multi-Threaded Access Now?

- New technology drivers multi-threaded
  - AI / ML packages like TensorFlow, PyTorch, etc
- CPU speed no longer the dominant form of improvements
  - Lots of cores on a chip, with newer big.LITTLE arrangements that can move I/O-bound code to slower, lower-power cores
- On-node / near-node memory & storage technologies incoming
  - Benefit from low-priority “data mover” threads that need multi-threading in HDF5
- Cloud storage here
  - Also benefits from low-priority “data mover” threads

Project Purpose and Expectations

- Purpose: Make HDF5 safe for concurrent access with multiple threads
- Expectations
  - Timely inclusion of the necessary changes within the production library
    - With comfort to all parties -- as in, risk minimized or significantly mitigated and that the changes contribute significantly in terms of performance and maintainability while minimizing additional technical debt
  - General buy-in:
    - That the strategy and approach is correct, and acceptable in the main
  - A critical technical review
    - Adapting and evolving the strategy and approach to the point where it can be implemented
  - This is a living document
    - Negotiate what/how to do this so that all are comfortable and believe the purpose will be achieved, as discussed above

You must engage and offer feedback!

Agenda

- Propose exploration of a concurrent, thread-safe H5Dread routine
  - Discuss technical strategy
  - But, this is a lengthy presentation so deep technical thoughts are best left to an offline forum
    - Want those thoughts, just not <u>as</u> the presentation is given
- Describe tests for correctness and robustness
  - Based on that technical strategy
- Discuss the opportunity for contributions to the effort by the HDF5 Community

Goals for Concurrent Multi-Threaded Access

- Long-Term
  - Allow fully concurrent execution of all HDF5 API routines, from multiple threads
- Immediate
  - Make a single HDF5 API routine threadsafe and concurrent when performing its primary function, possibly under limited circumstances:
    - Allow fully concurrent execution of H5Dread from multiple threads, all the way down to pread() in the sec2 (POSIX) VFD
  - Allow fully concurrent execution of multiple HDF5 API routines, down to a logically appropriate level:
    - Allow concurrent execution of all VOL operations, down to the callback to the VOL connector

Goals for Concurrent Multi-Threaded Access

- Long-Term
  - Allow fully concurrent execution of all HDF5 API routines, from multiple threads
- Immediate
  - Make a single HDF5 API routine threadsafe and concurrent when performing its primary function, possibly under limited circumstances:
    - Allow fully concurrent execution of H5Dread from multiple threads, all the way down to pread() in the sec2 (POSIX) VFD
  - Allow fully concurrent execution of multiple HDF5 API routines, down to a logically appropriate level:
    - Allow concurrent execution of all VOL operations, down to the callback to the VOL connector

MT-Safe H5Dread

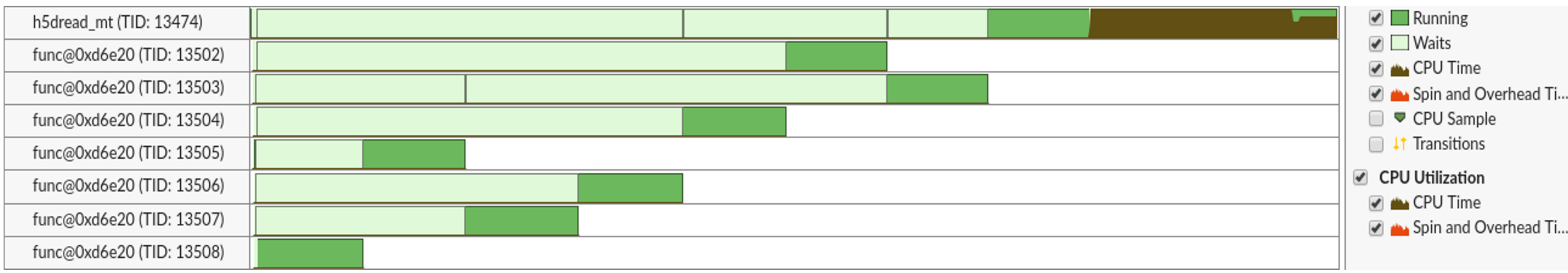
- Constraints:
    - Contiguous dataset layout
        - Chunked, virtual, etc. in later extensions
    - Atomic (fixed-length) datatypes
    - No datatype conversions
    - No data transforms (i.e. H5Pdata_transform)
    - Serial I/O
        - sec2 (POSIX) VFD
- Support:
    - H5Dread operations to same or different datasets
    - Error handling

Initial H5Dread Experiment

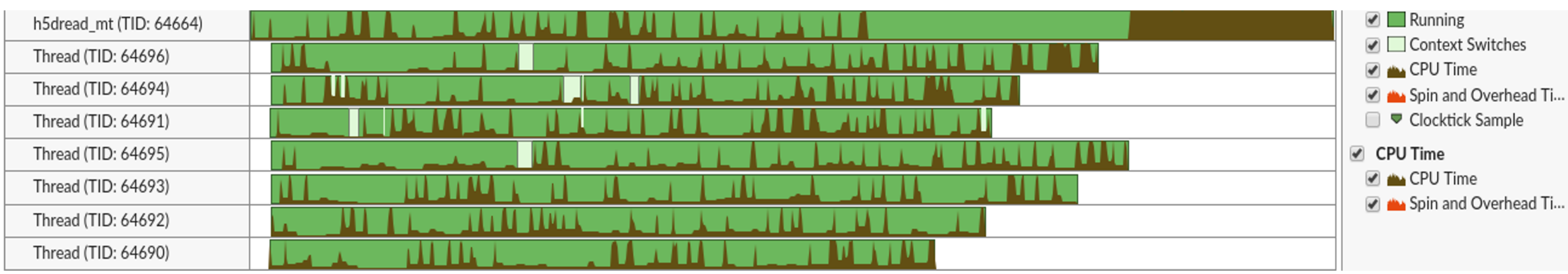
- Plan: Allow multiple threads to concurrently execute H5Dread operation
  - Remove the global API lock from H5Dread and reduce the lock granularity, while still protecting shared data.
- Assumptions and Constraints
  - No errors are encountered.
  - All library initialization is complete before any thread calls H5Dread.
  - No threads are doing anything except calling H5Dread.
  - Library's internal memory free lists are disabled.
  - "NDEBUG" is defined to disable asserts and other error checking.
- Test Program
  - Main thread opens an HDF5 file (8 64MiB datasets) and initializes the library.
  - Worker threads each read one dataset.
  - Workers join the main thread, which then closes the file and exits.

Results on Cori @ NERSC with Lustre

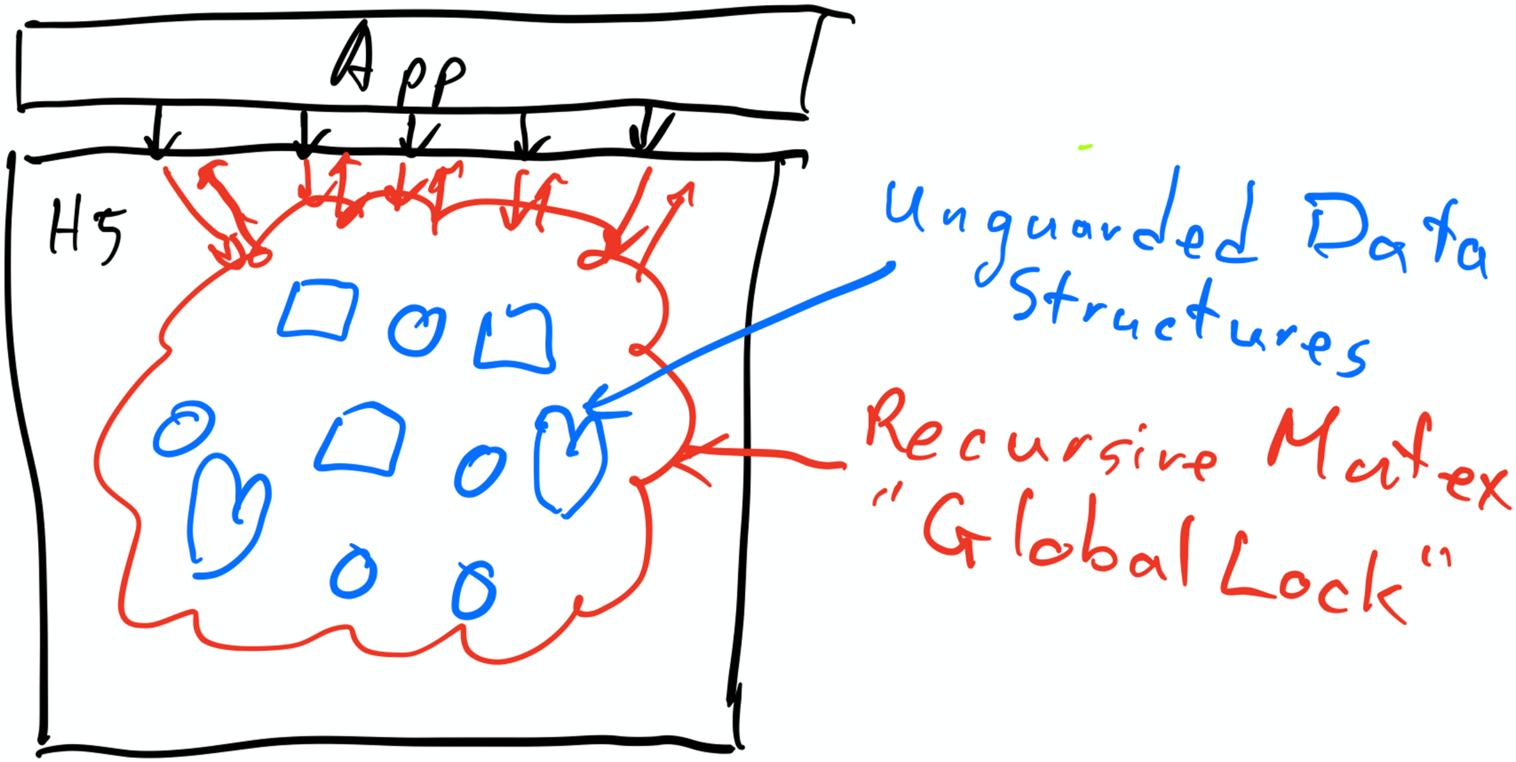
- With Global API Lock



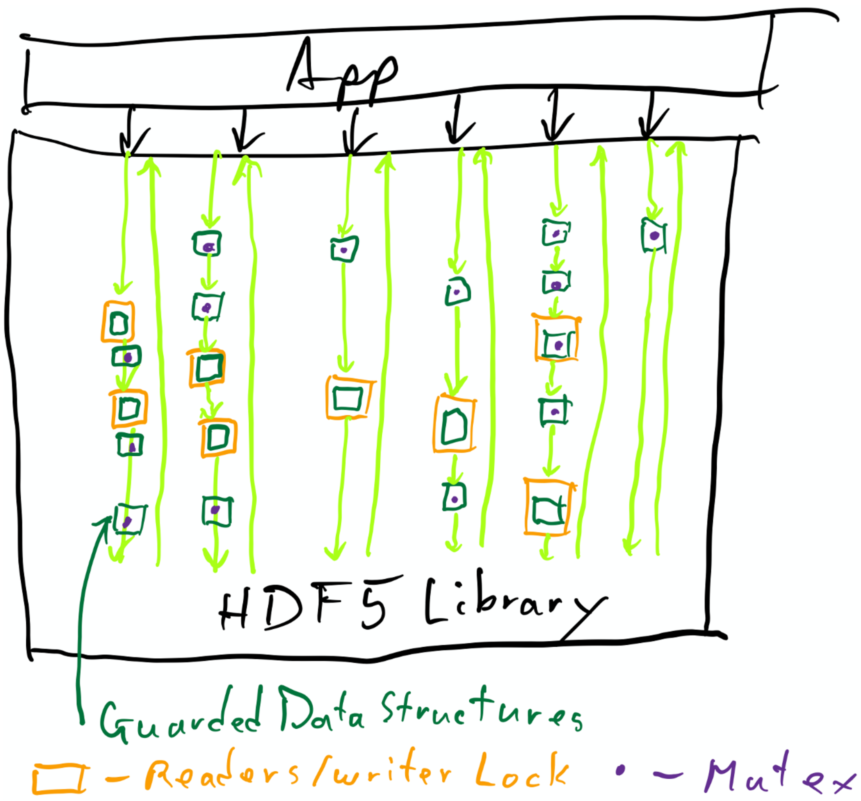
- Without Global API Lock



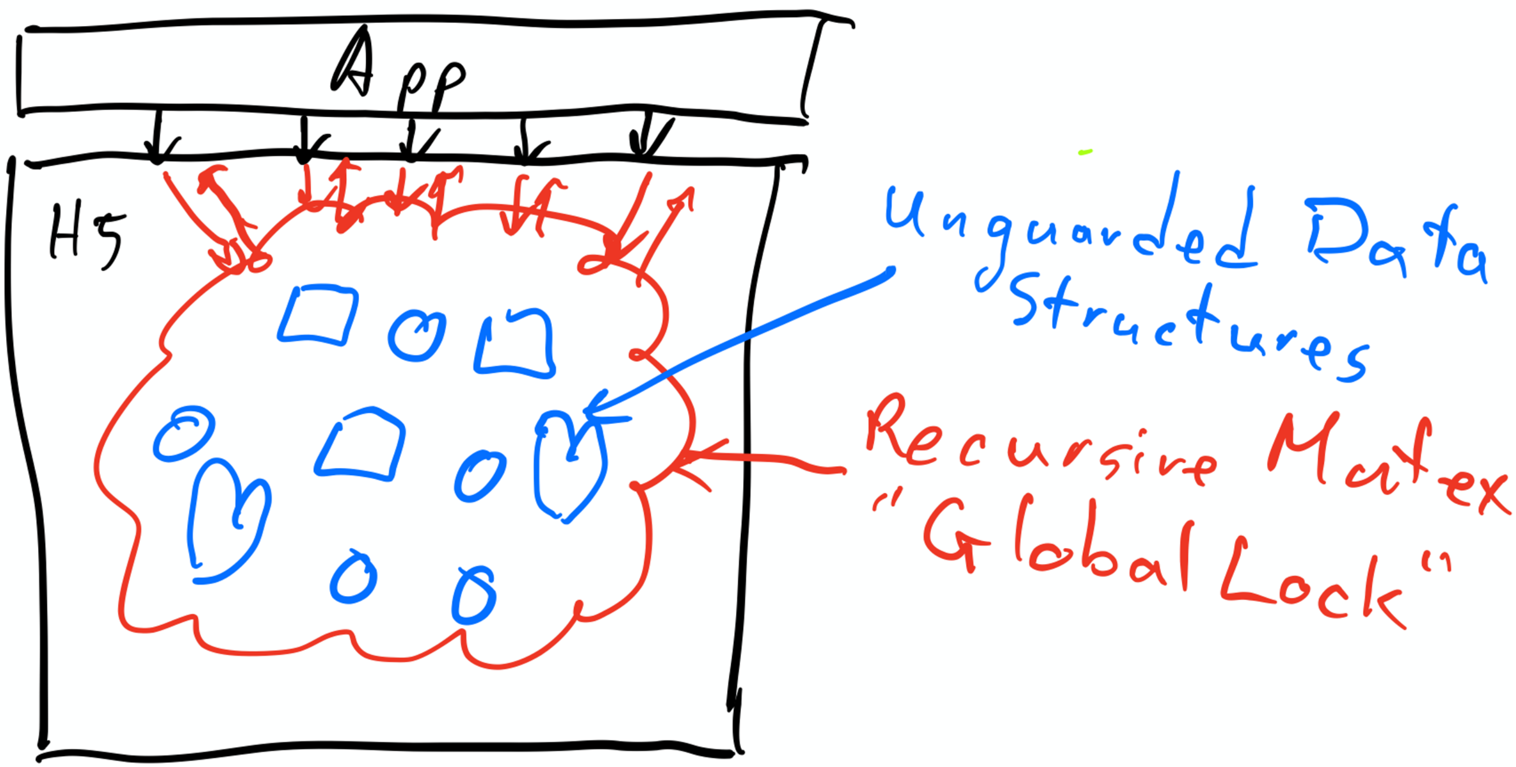
Current Concurrency Control in HDF5



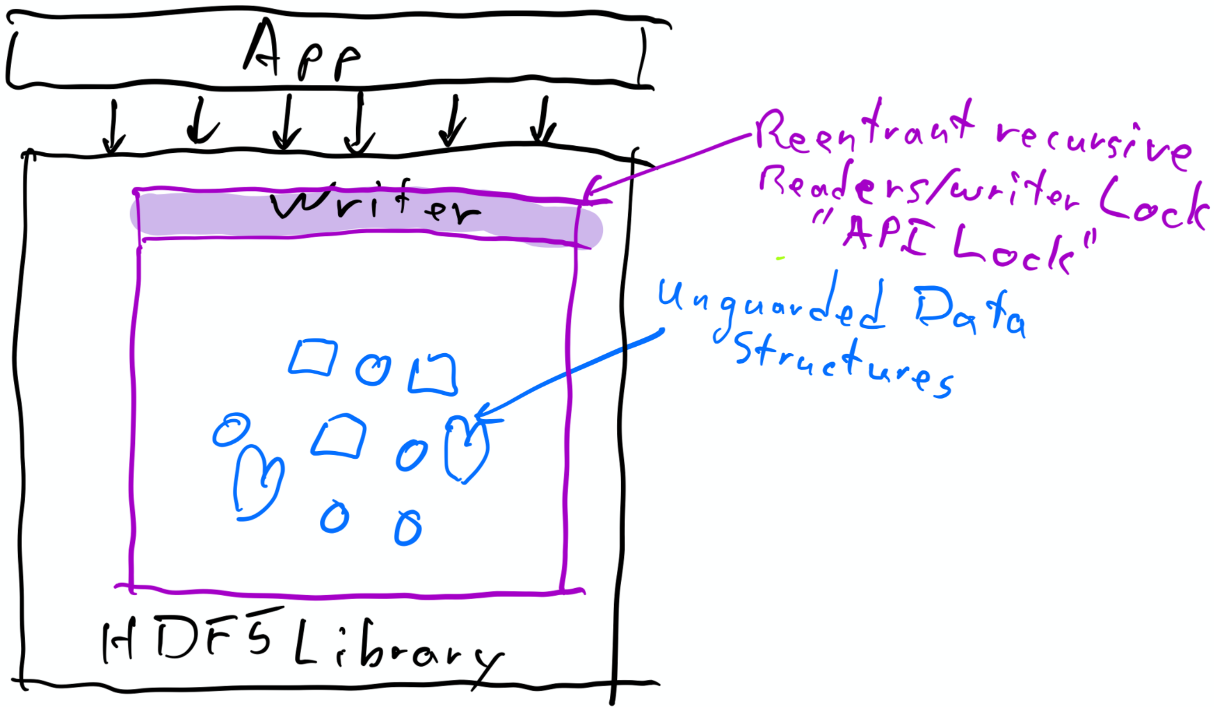
Future Concurrency Control in HDF5



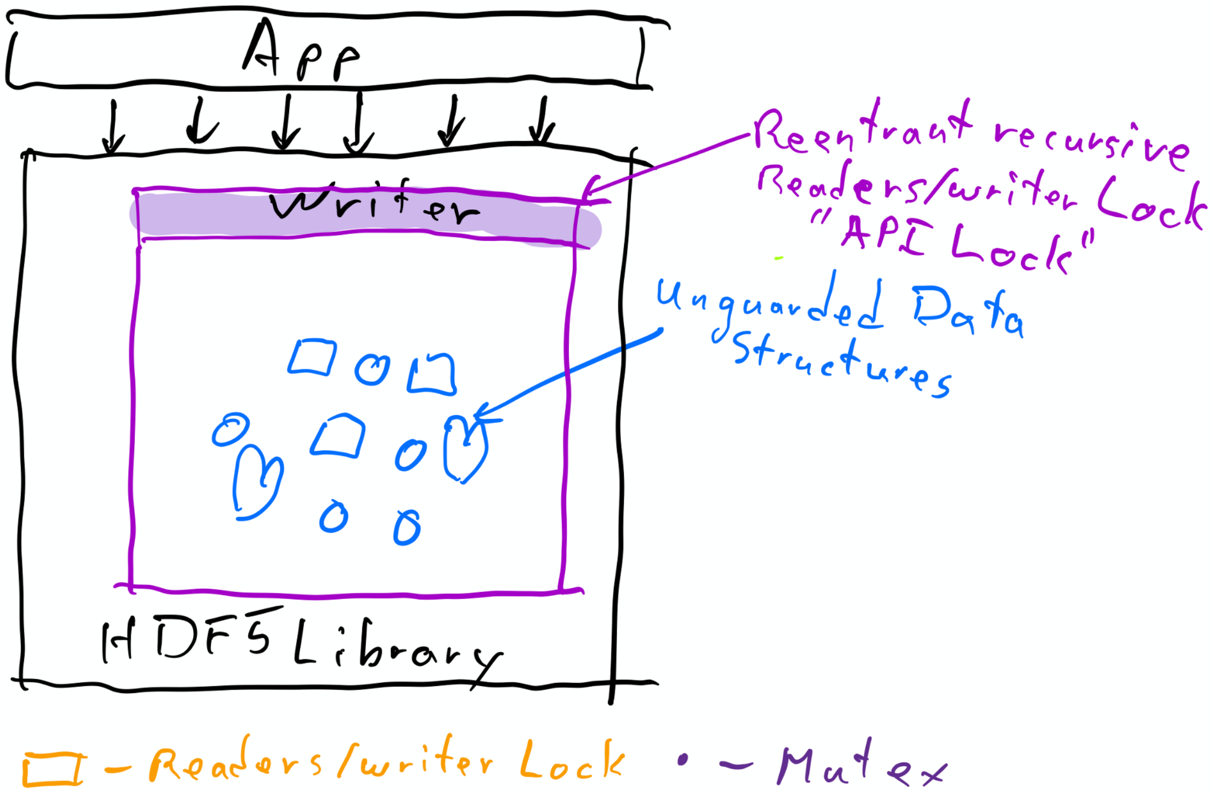
Concurrency Control - Now



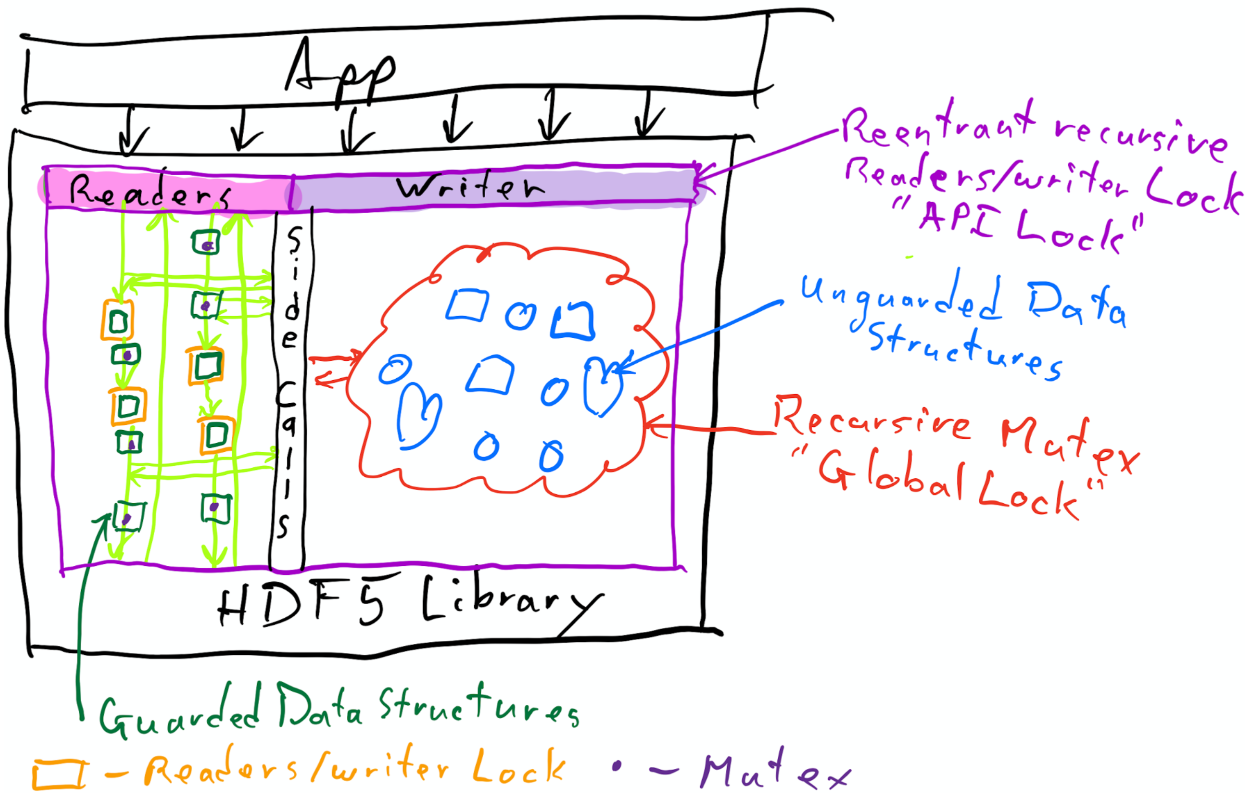
Concurrency Control - Step 1



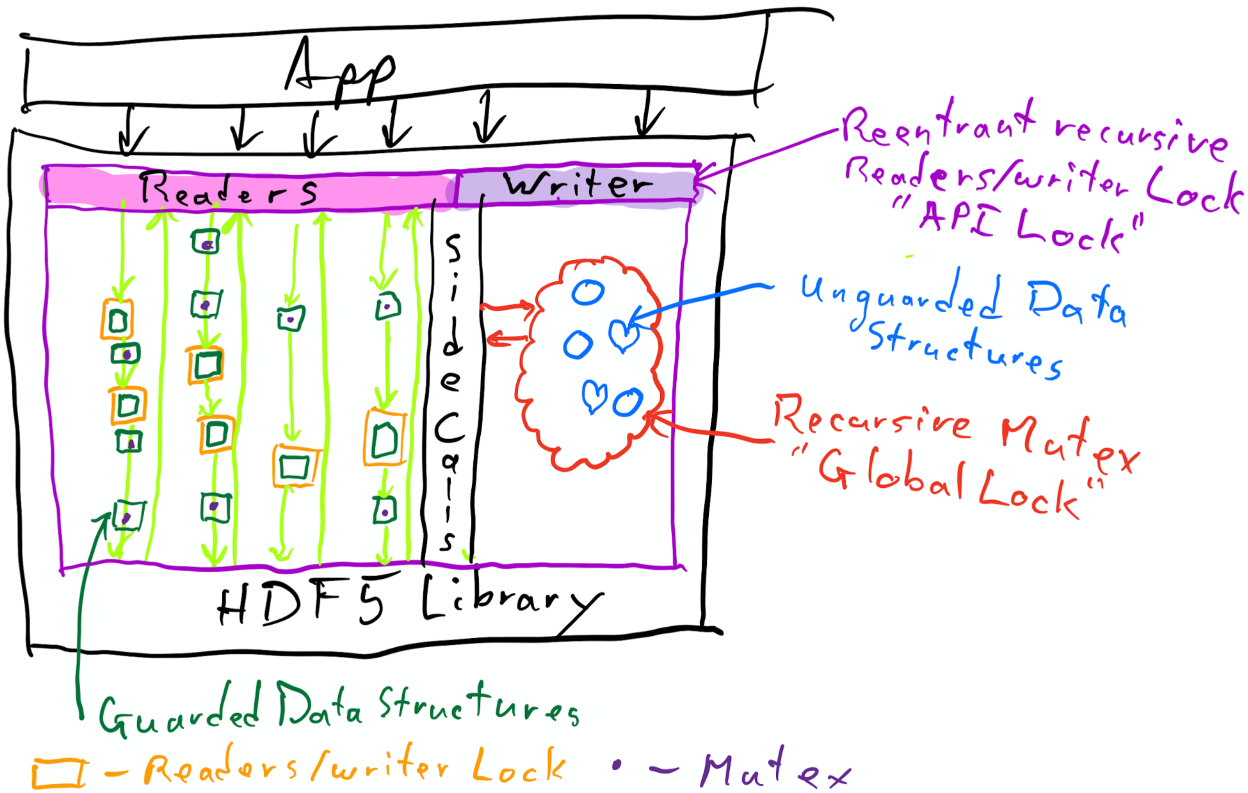
Concurrency Control - Step 1(a)



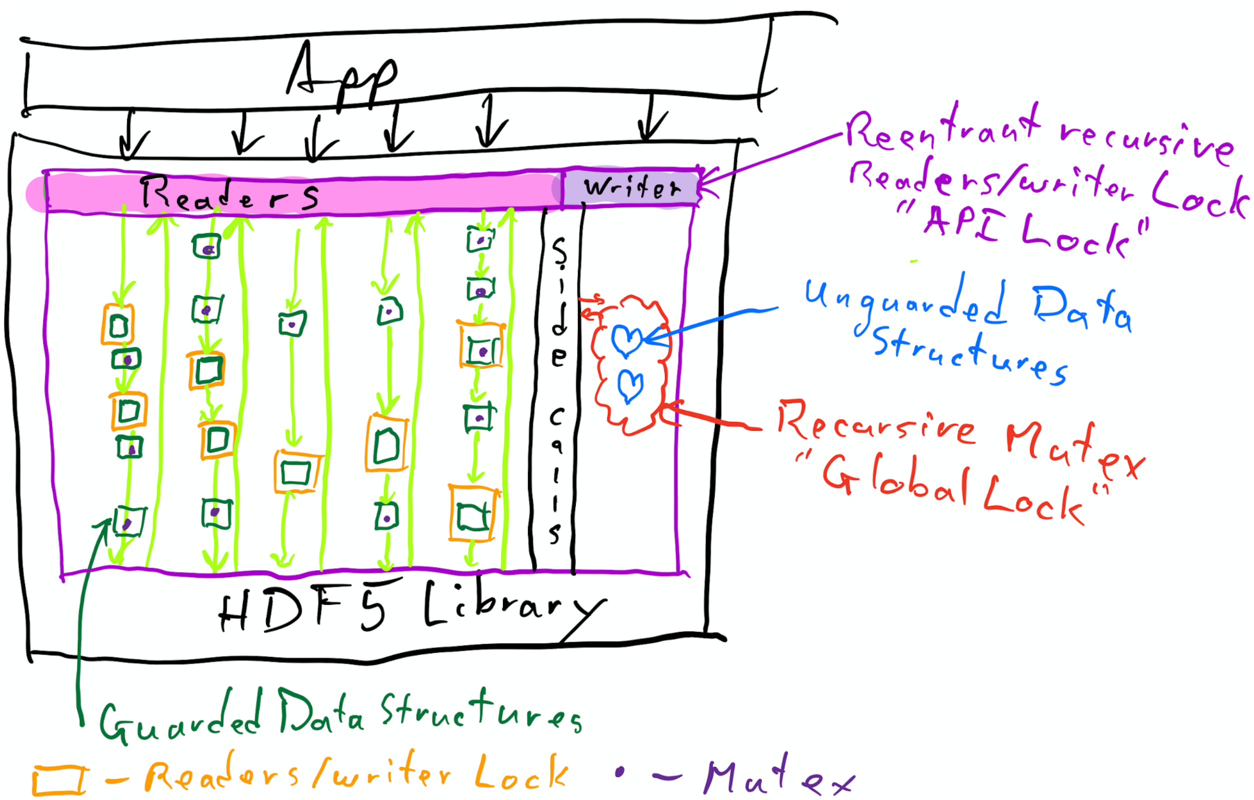
Concurrency Control - Step 2



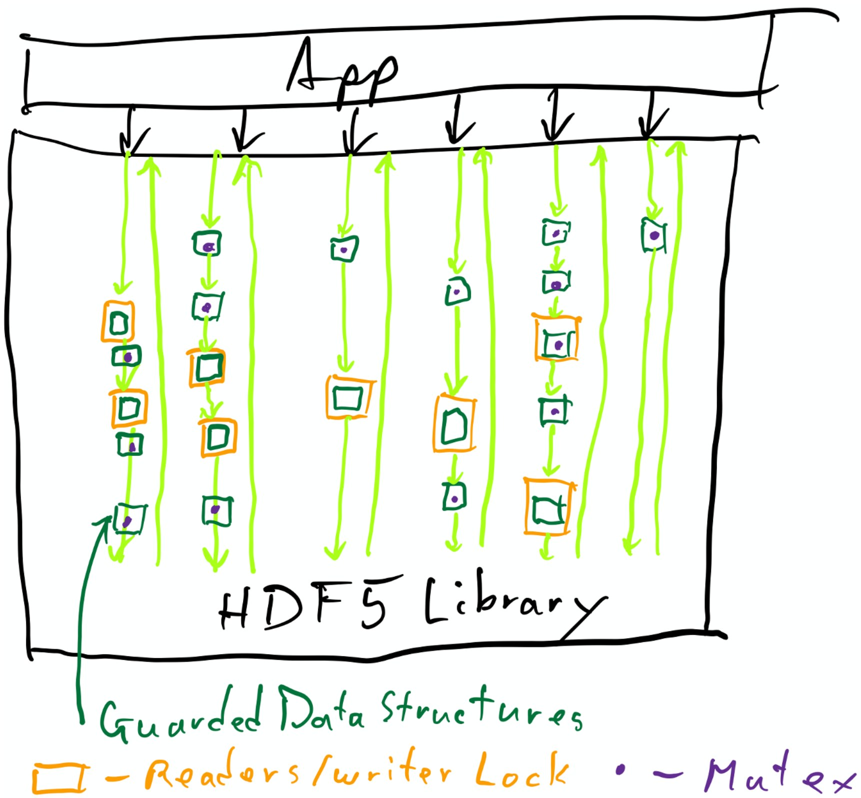
Concurrency Control - Under Way



Concurrency Control - Almost Done



Concurrency Control - Done!



Testing for correctness and robust function

- Create a standalone test that
  - Opens and closes multiple datasets concurrently
    - Which will always serialize because of the global lock but exercises breadth
  - More frequently
    - Issue multiple, concurrent read requests to all open datasets
      - Which are expected to proceed mostly concurrently
      - With validations for proper operation and function
- Compiling this test against
  - An unmodified HDF5 library allows a baseline performance metric
  - A library with concurrency modifications provides
    - Correctness and robustness testing
    - A performance metric
      - Can be used to demonstrate efficacy by comparing with the baseline, above
- When standalone test demonstrates correct operation
  - Add concurrency test(s) to HDF5 regression test suite

Paving the way for Community Contributions

- Plan to modify the dataset read, open, and close paths, and the internal ID manager code
  - Leaving the rest for other contributors or as follow-on activities
  - Most work is local in scope, restricted to compartments
    - Except the interfacing macros and changes to the dataset memory structure
- Low-hanging fruit for someone else:
  - MT-Safe memory allocation would be a significant contribution
    - All threads serialize here, including this work as it will guard when using a global lock
    - Making these routines MT-safe requires only internal, thus opaque, changes
    - Needed changes are independent of this work, and vice versa
  - Certainly other opportunities!
- Long term
  - The initial project have provided infrastructure changes
  - Others can leverage the strategy/approach and those changes, too, in other code paths

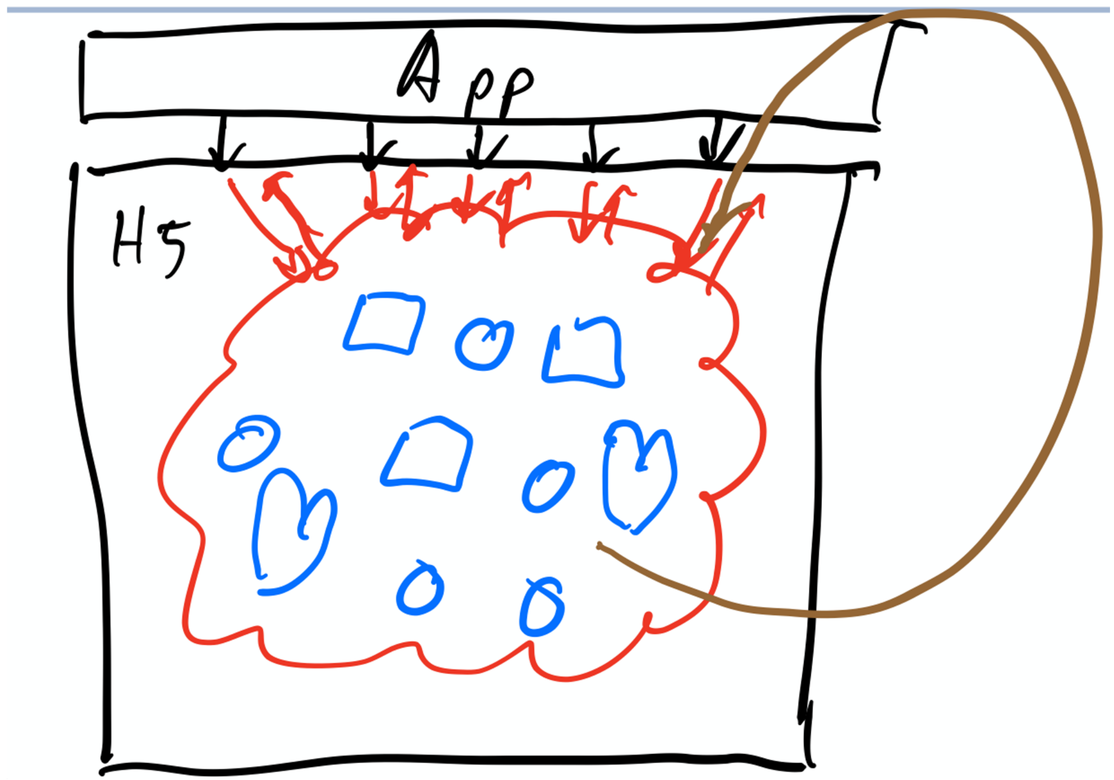
Conclusion

- Strategy for conversion of HDF5 library to full multi-threaded concurrency
  - Technically sound
  - Incrementally achievable
  - Testable
- Production-quality code contribution
  - Reduce technical debt
    - In new code, and in existing code, through the extensive code review required
  - Implement necessary reusable infrastructure
  - Satisfy current application needs
  - Serve as example for others
- Opening for community contributions
  - Want community to bring more incremental improvements, for a greatly desired capability
- Opportunity for HDF5, in general
  - Opens HDF5 to more use-cases / industries / fields

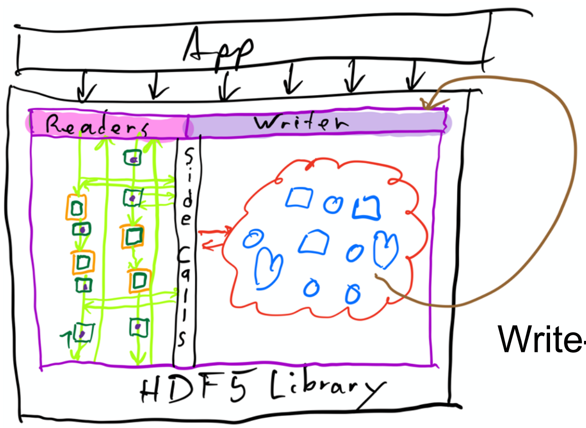
Questions / Feedback / Discussion?

Locking / Concurrency Details

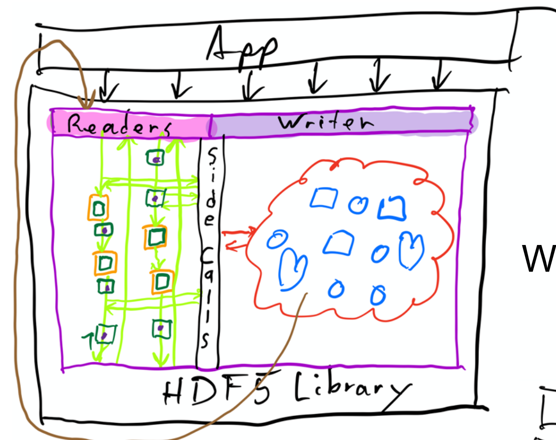
Library Re-entrancy Now



Library Re-entrancy During Conversion

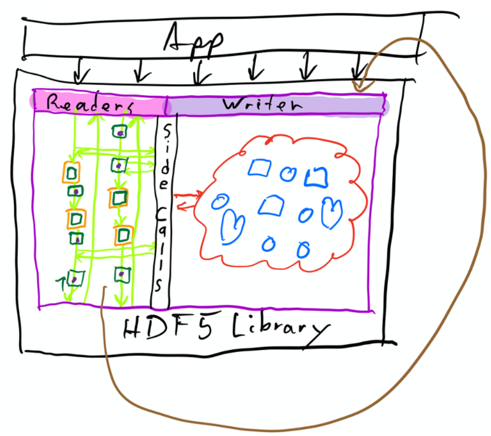


Write-Write



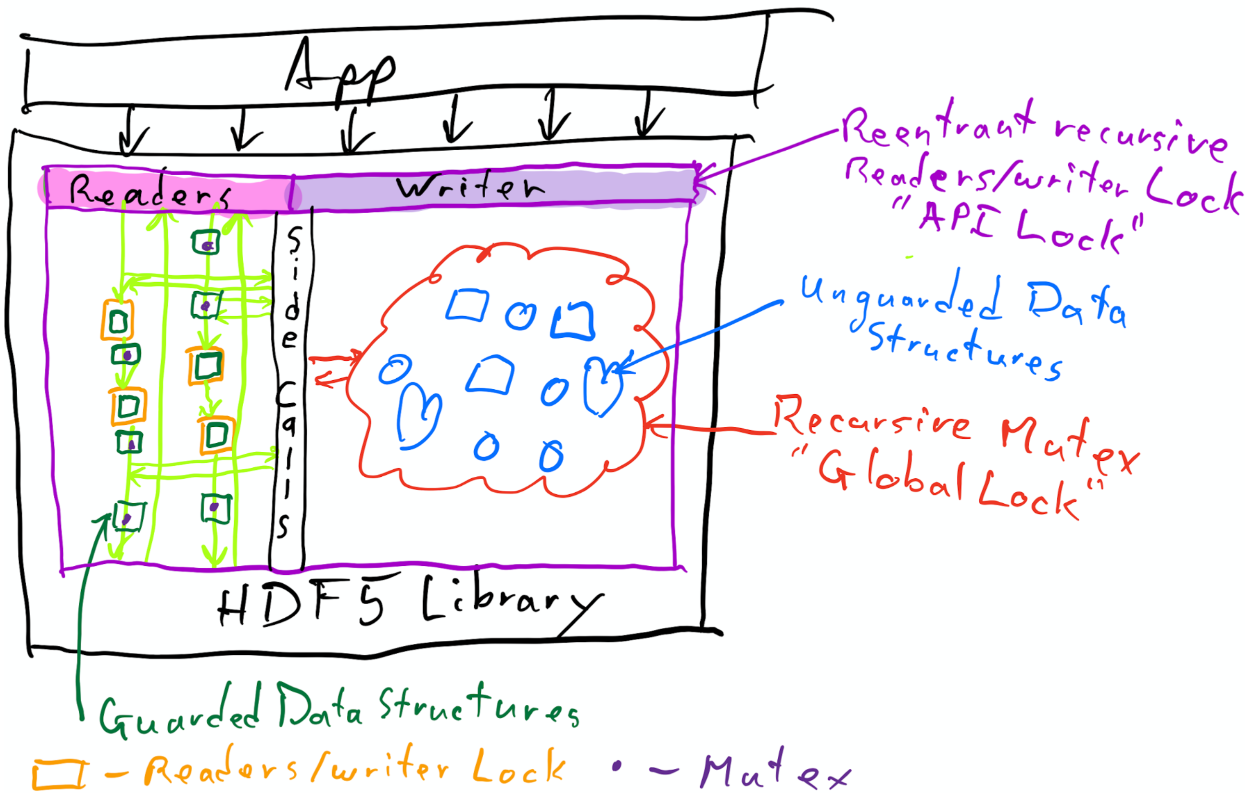
Write-Read

Read-Read

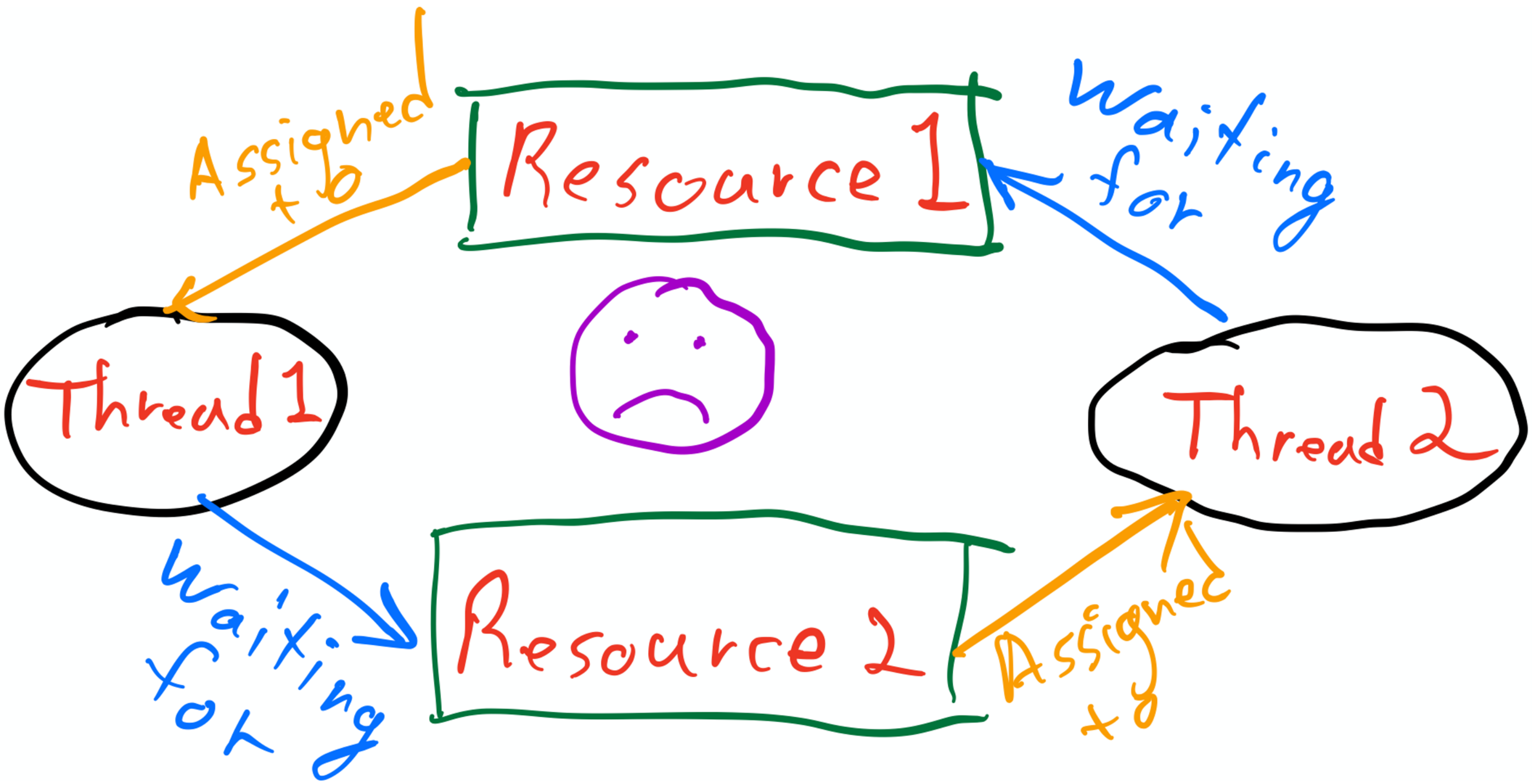


Read-Write

Are all of these locks required?



Avoiding Deadlocks



Coding Details

How to Make H5Dread MT-Safe

- Constraints:
  - Contiguous dataset layout
  - Atomic (fixed-length) datatypes
  - No datatype conversions
  - No data transforms
    - H5Pdata_transform
  - Serial I/O
    - sec2 (POSIX) VFD
- Supports:
  - H5Dread operations to same or different datasets
  - Error handling

MT-Safe Infrastructure/Support

- Infrastructure needed:
    - New portable locks: reentrant recursive readers/writer lock, readers/writer lock, mutex
        - Regular readers/writer lock and mutex not required to be recursive
    - New implementations of HDF5's internal macros:
        - "Private" FUNC_ENTER/LEAVE macros that acquire the global lock, for internal routines
        - ERROR handling macros that acquire the global lock
            - Or acquire it in the routines they invoke
        - API TRACE macros that acquire the global lock
            - Or acquire it in the routines they invoke
        - "Public" FUNC_ENTER/LEAVE macros that acquire reader or writer API Lock, for public API routines
    - Analyze definition of FUNC_ENTER/LEAVE macros that <u>don't</u> acquire the global lock for internal routines
        - Use new private, global lock-acquisition FUNC_ENTER/LEAVE macros in those routines

H5Dread Implementation (For Reference)

```
herr_t H5Dread(hid_t dset_id, hid_t mem_type_id, hid_t mem_space_id, hid_t file_space_id, hid_t dxpl_id, void *buf/*out*/)
{
    H5VL_object_t  *vol_obj     = NULL;
    herr_t  ret_value   = SUCCEED;      /* Return value */

    FUNC_ENTER_API(FAIL)
    H5TRACE6("e", "iiiiix", dset_id, mem_type_id, mem_space_id, file_space_id, dxpl_id, buf);

    /* Check arguments */
    if (mem_space_id < 0)
        HGOTO_ERROR(H5E_ARGS, H5E_BADVALUE, FAIL, "invalid memory dataspace ID")
    if (file_space_id < 0)
        HGOTO_ERROR(H5E_ARGS, H5E_BADVALUE, FAIL, "invalid file dataspace ID")

    /* Get dataset pointer */
    if (NULL == (vol_obj = (H5VL_object_t *)H5I_object_verify(dset_id, H5I_DATAS
        HGOTO_ERROR(H5E_ARGS, H5E_BADTYPE, FAIL, "dset_id is not a dataset ID")

    /* Get the default dataset transfer property list if the user didn't provide
    if (H5P_DEFAULT == dxpl_id)
        dxpl_id = H5P_DATASET_XFER_DEFAULT;
    else
        if (TRUE != H5P_isa_class(dxpl_id, H5P_DATASET_XFER))
            HGOTO_ERROR(H5E_ARGS, H5E_BADTYPE, FAIL, "not xfer parms")

    /* Read the data */
    if ((ret_value = H5VL_dataset_read(vol_obj, mem_type_id, mem_space_id, file_
        HGOTO_ERROR(H5E_DATASET, H5E_READERROR, FAIL, "can't read data")

done:
    FUNC_LEAVE_API(ret_value)
} /* end H5Dread() */
```

How to Make H5Dread MT-Safe

- Fundamental Step: Make H5Dread entry-point thread-safe
  - Modifications to H5Dread
    - Use new global lock-acquisition TRACE macro
    - Use new global lock-acquisition ERROR macros
    - Use new reader API Lock-acquisition public FUNC_ENTER/LEAVE macros
  - For each “side call”: H5I_object_verify, H5P_isa_class
    - Use new global lock-acquisition private FUNC_ENTER/LEAVE macro
  - For “main call”: H5VL_dataset_read
    - Leave with <u>non-lock-acquisition</u> private FUNC_ENTER/LEAVE macros
    - Use new global lock-acquisition ERROR macros
    - Use new global lock-acquisition private FUNC_ENTER/LEAVE macro in each “side call”
    - Repeat these “main call” steps as the call chain continues down internal routines, until the pread() call in the sec2 (POSIX) VFD is reached:
      - H5VL__dataset_read => H5VL_native_dataset_read => H5D__read => H5D__contig_read => H5D__select_read => H5D__select_io => … => pread()

How to Make H5Dread MT-Safe

- Advanced Steps: Make a “side call” thread-safe
    - [[[Describe how to make H5I_object_verify thread-safe and concurrent]]]
    - [[[ID manager discussed here?]]]

Dataset Memory Object Modifications

- Object acquisition/use as serialization point
  - Removes need for long-lived critical sections of code
  - Allows management of multiple, conflicting atomic changes to object
  - Implement; Add reference count to track liveness
  - Implement; Add ISLOCKED flag to manage exclusive use
- Reference() and release(); Atomically {in,de}crease the reference count
  - When reference count goes to zero => destroy (AKA “kill”) the record
- Lock() and unlock(); Atomically wait then set and unset the ISLOCKED flag
- Get() and put(); ref + lock and unlock + release
- Modify Lookup(by ID); Create or return object given an ID
  - Object is returned referenced and locked
  - If caller did not want that, just drop the offending portion with unlock or release
    - Or, pass a flag indicating whether caller wants the lock as this would be the usual, but not normal, case

But the close routine can't!

- Destruction no longer explicit, must be able to defer it
- Solution; Zombies!
  - Implement; Add ISZOMB flag to dataset record/handle
  - ID manager must be careful to block attempts by caller to reopen until the associated record/handle has been killed
- Gone(); Remove/Stall association, then put() + set ISZOMB flag
  - Refactor close routine into a call to gone
  - Moving the real destruction into a "kill" routine, used by the release routine
- Other threads can continue normally
  - Until they drop their last reference, of course
  - Though they might need to exercise care when reacquiring locks