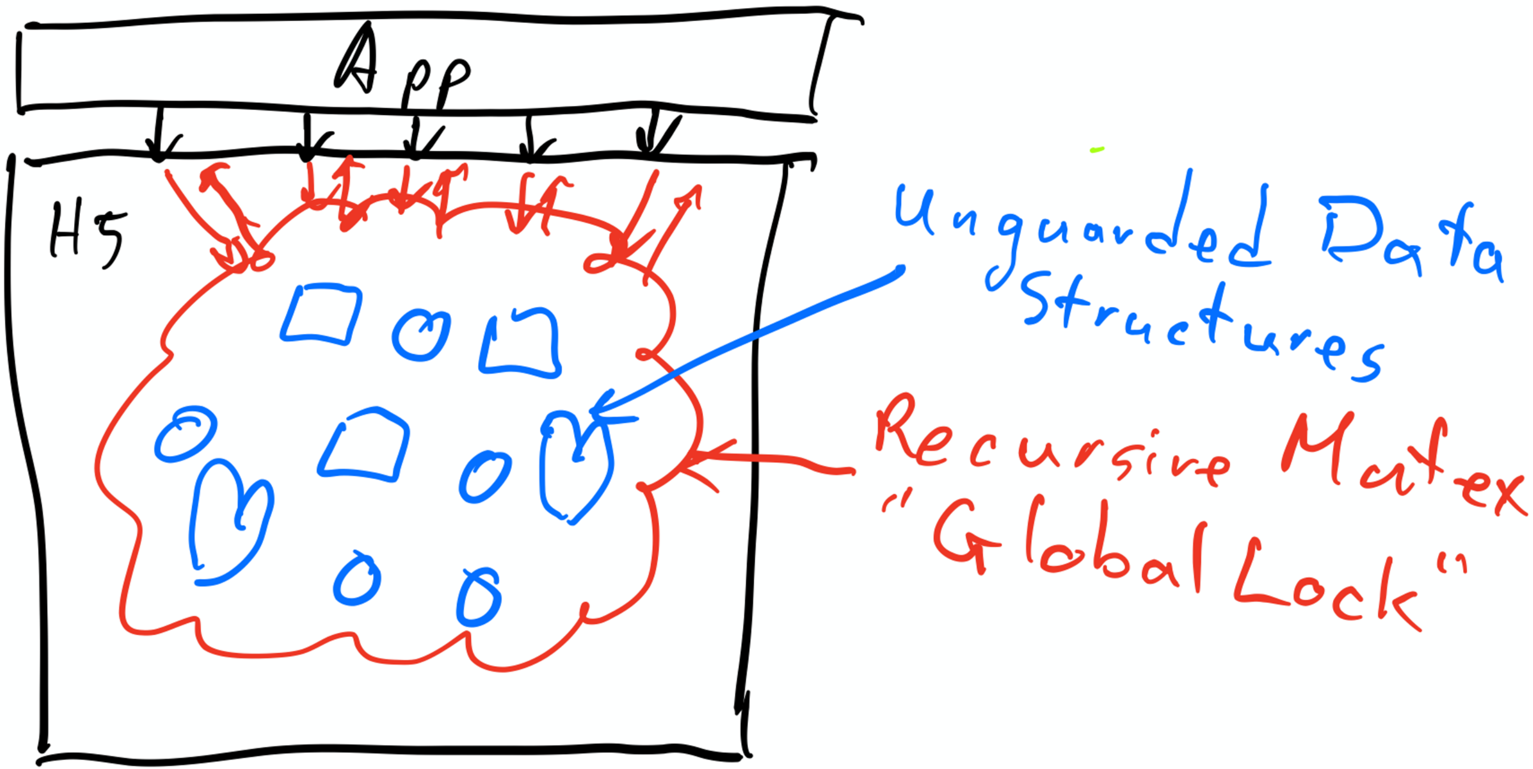
Concurrent HDF5: A Community Contribution Proposal

Quincey Koziol, LBNL
Chris Hogan, The HDF Group

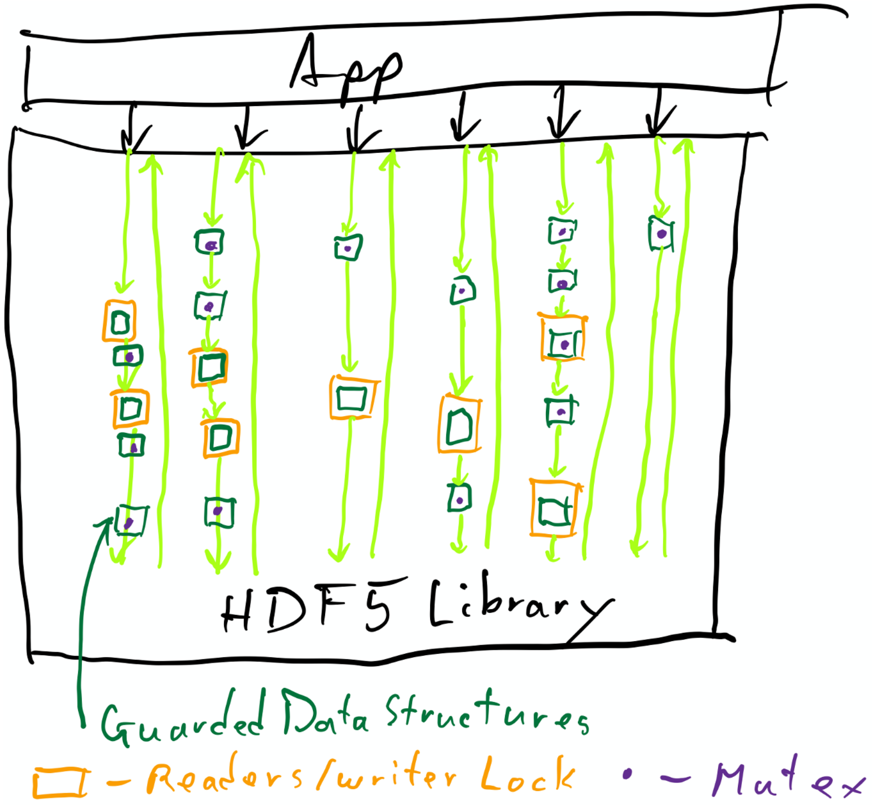
Goals for Concurrent Multi-Threaded Access

- Long-Term
  - Allow fully concurrent execution of all HDF5 API routines from multiple threads
- Immediate
  - Make a single HDF5 API routine thread-safe and fully concurrent when performing its primary function, possibly under limited circumstances
    - Ex: 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
    - Ex: Allow fully concurrent execution of all VOL operations, down to the callback to the VOL connector

Current Concurrency Control in HDF5



Future Concurrency Control in HDF5



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
- Support:
  - H5Dread operations to same or different datasets
  - Error handling

MT-Safe Infrastructure

- Infrastructure needed:
    - New portable lock:
        - Recursive readers/writer lock
    - New implementations of HDF5's internal macros:
        - "Private" FUNC_ENTER/LEAVE macros that acquire the global lock, for internal routines
        - ERROR 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

Paving the way for Community Contributions

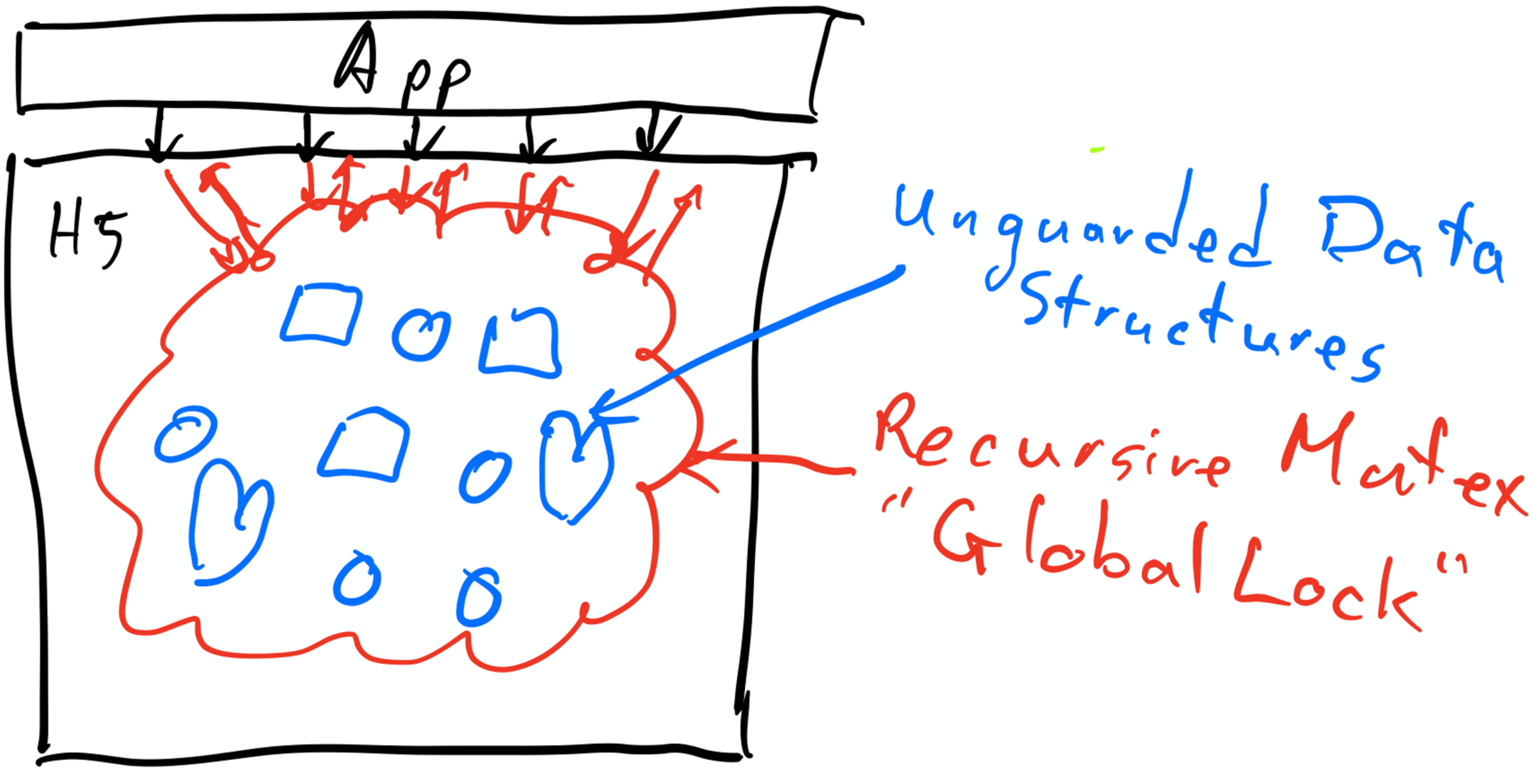
- We will modify the dataset open, read, and close paths, and the ID manager
  - Leaving the rest for other contributors or ourselves as follow-on activities
  - Most work is local in scope, restricted to compartments
    - Except the interfacing macros and changes to the dataset memory structure
- Set up for community contributions
  - We will have provided infrastructure changes
  - Others can leverage the strategy/approach and those changes, too, in other code paths
- A possible community contribution opportunity:
  - MT-Safe memory allocation would be a significant contribution
    - All threads serialize here, including our work as we will guard using a global lock
    - Making these routines MT-safe requires only internal, thus opaque, changes
    - Needed changes are independent of our work, and vice versa

Conclusion

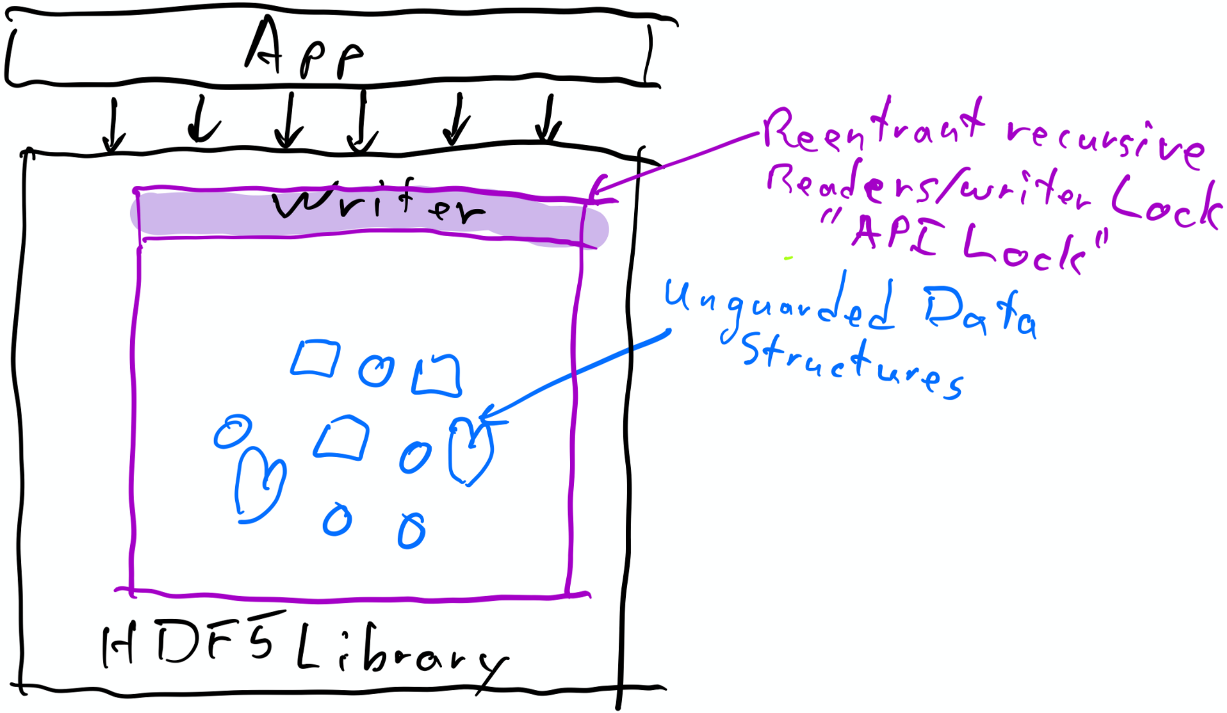
- Strategy for conversion of HDF5 library to full multi-threaded concurrency
  - Technically sound
  - Incrementally achievable
  - Testable
- Production-quality code contribution
  - Reduce technical debt (as code is refactored to be concurrent)
  - Implement necessary reusable infrastructure
  - Serve as example for others
- Opening for community contributions
  - Engage community to bring more incremental improvements for a greatly desired capability

Locking / Concurrency Details

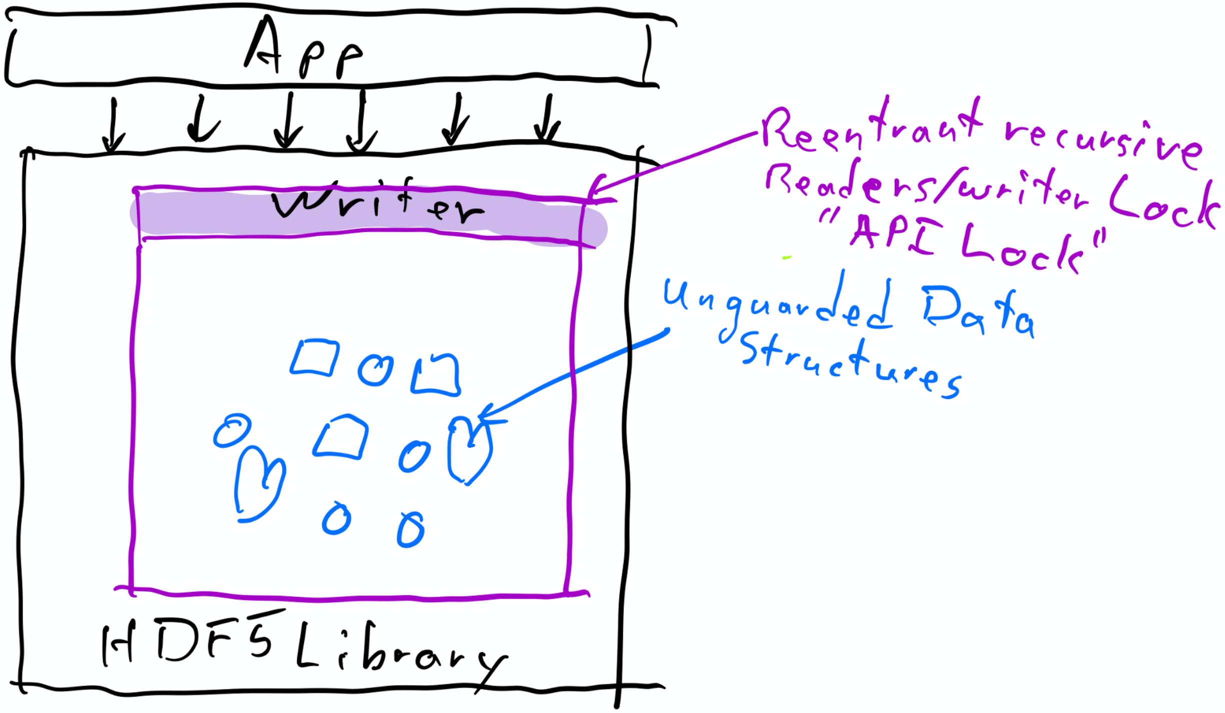
Concurrency Control - Now



Concurrency Control - Step 1



Concurrency Control - Step 1(a)

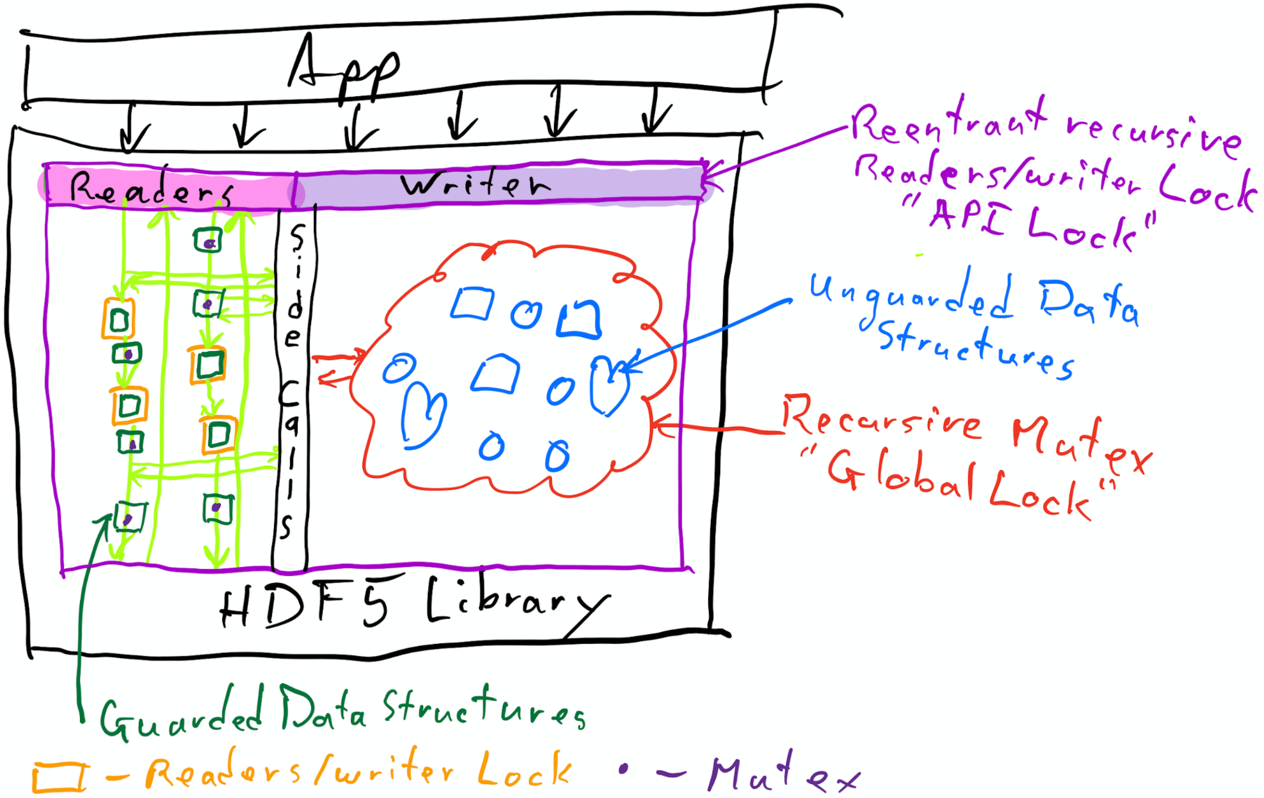


Reentrant recursive Readers/writer Lock "API Lock"

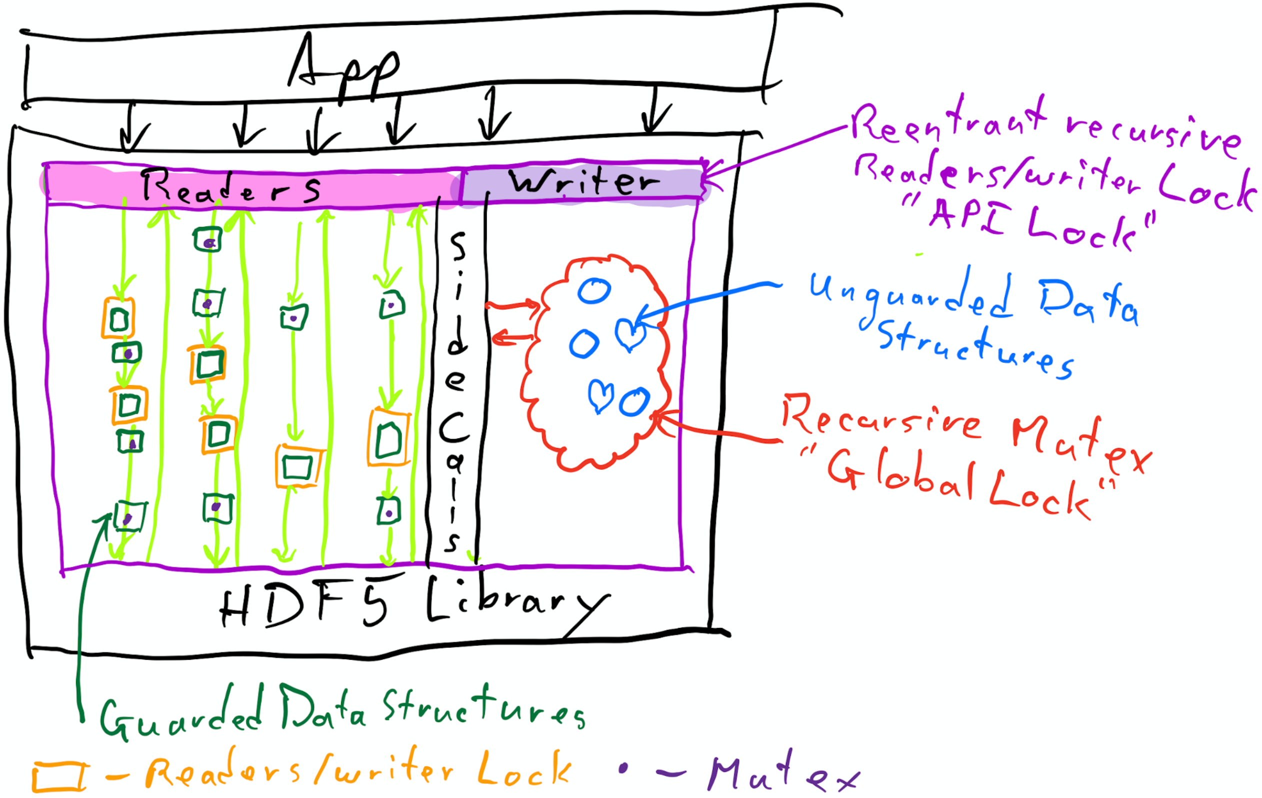
Unguarded Data Structures

□ - Readers/writer Lock • - Mutex

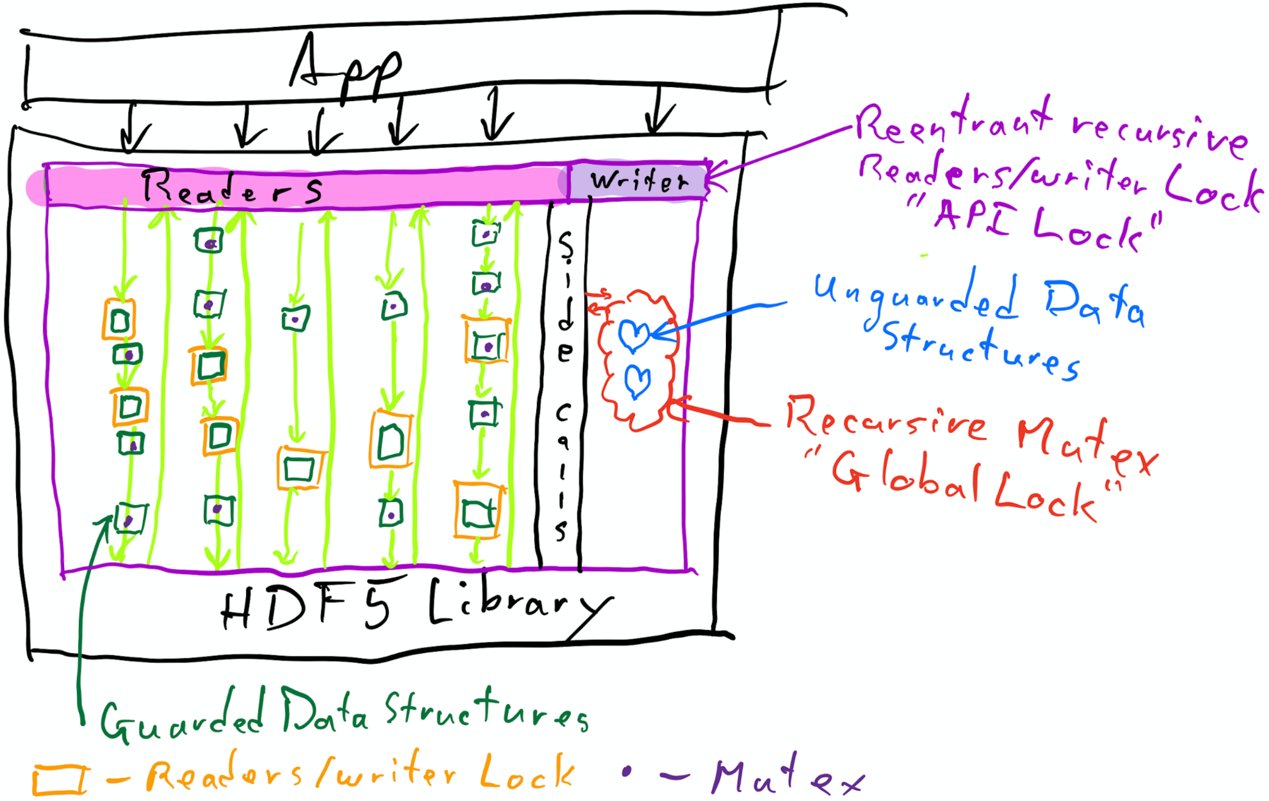
Concurrency Control - Step 2


App
Readers
Writer
Side Calls
HDF5 Library
Reentrant recursive Readers/writer Lock "API Lock"
Unguarded Data Structures
Recursive Mutex "Global Lock"
Guarded Data Structures
□ - Readers/writer Lock
• ~ Mutex

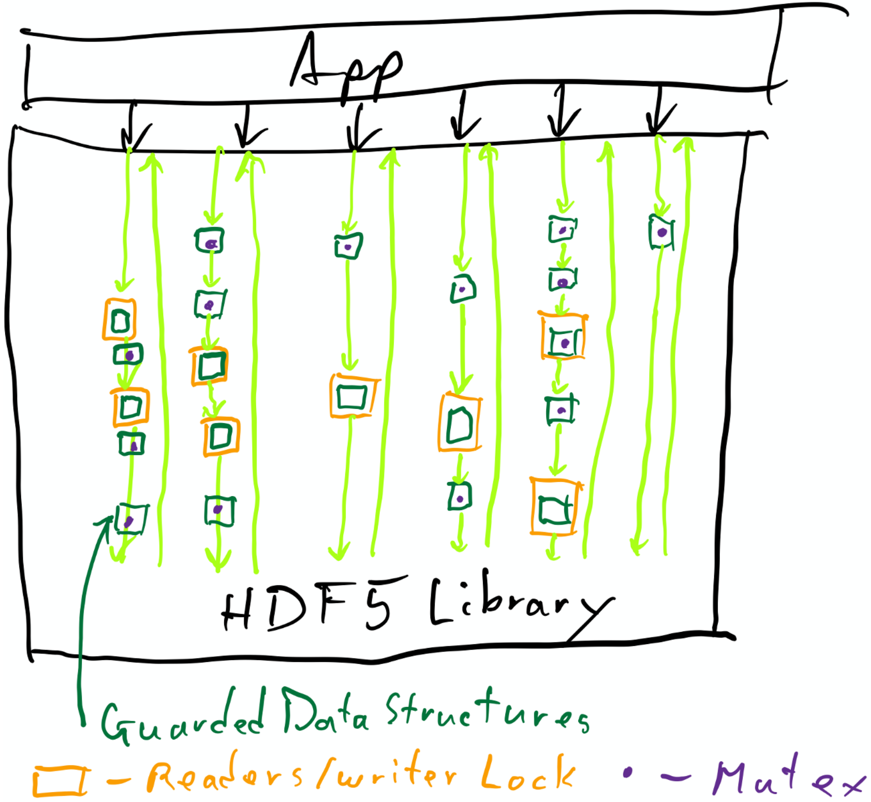
Concurrency Control - Under Way


App
Readers
Writer
Side Calls
HDF5 Library
Reentrant recursive Readers/writer Lock "API Lock"
Unguarded Data Structures
Recursive Mutex "Global Lock"
Guarded Data Structures
□ - Readers/writer Lock
• - Mutex

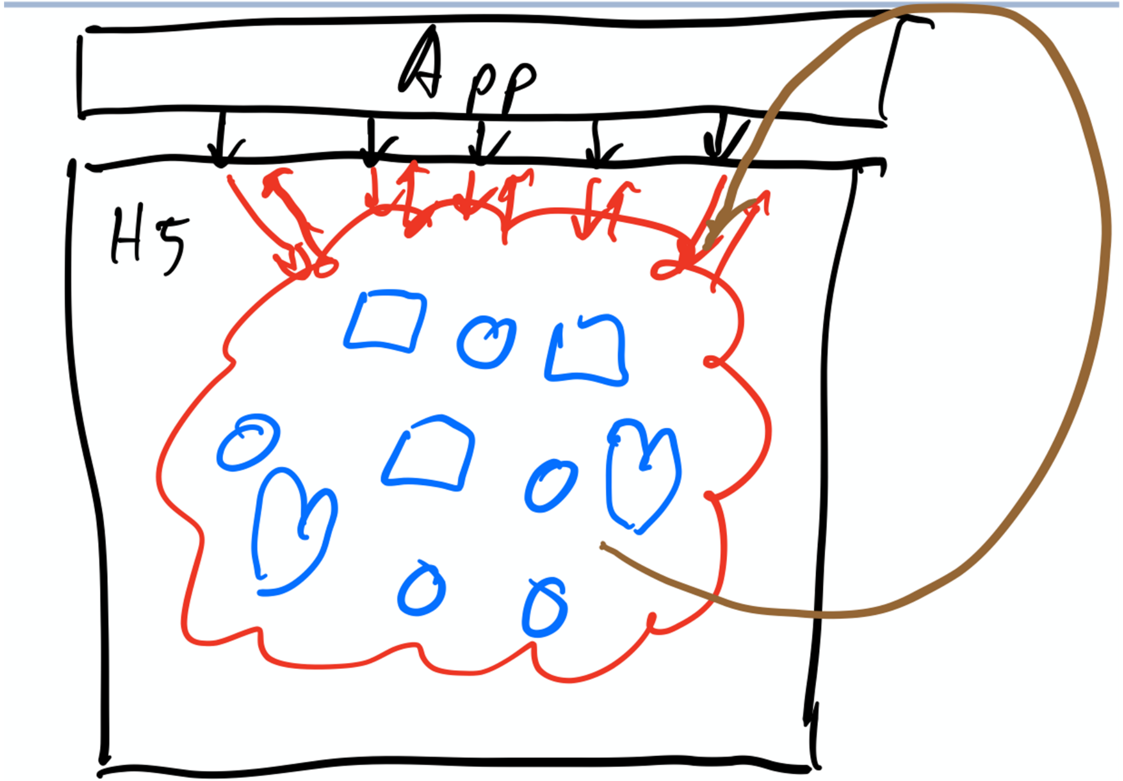
Concurrency Control - Almost Done


App
Readers
Writer
Side calls
HDF5 Library
Reentrant recursive Readers/writer Lock "API Lock"
Unguarded Data Structures
Recursive Mutex "Global Lock"
Guarded Data Structures
□ - Readers/writer Lock
• - Mutex

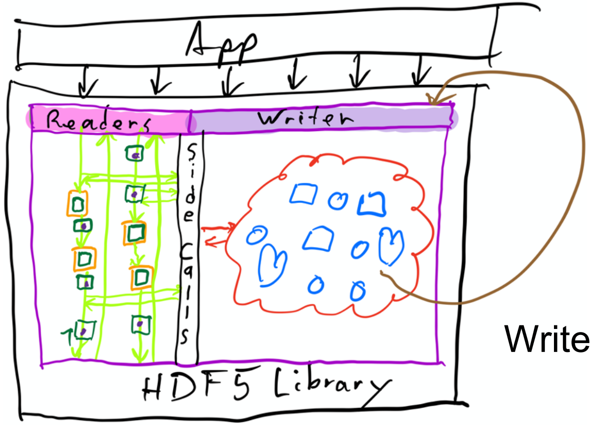
Concurrency Control - Done!



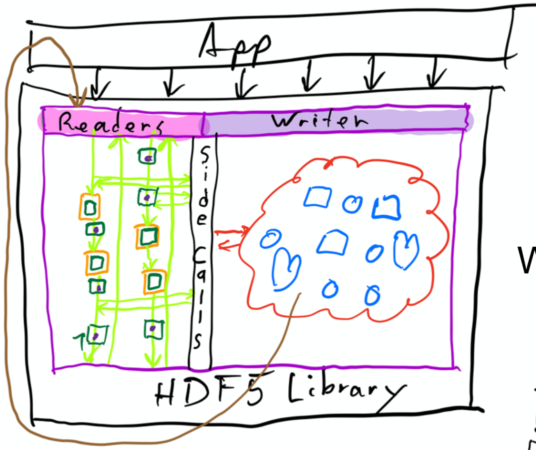
Library Re-entrancy Now



Library Re-entrancy During Conversion

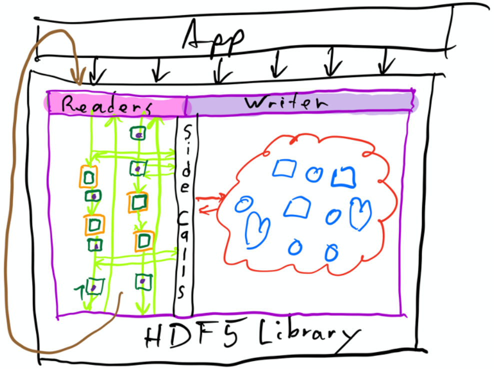


Write-Write

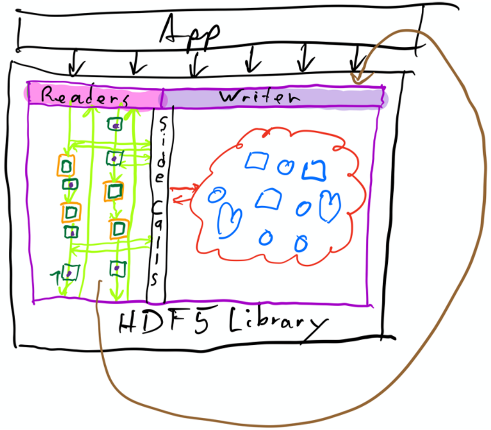


Write-Read

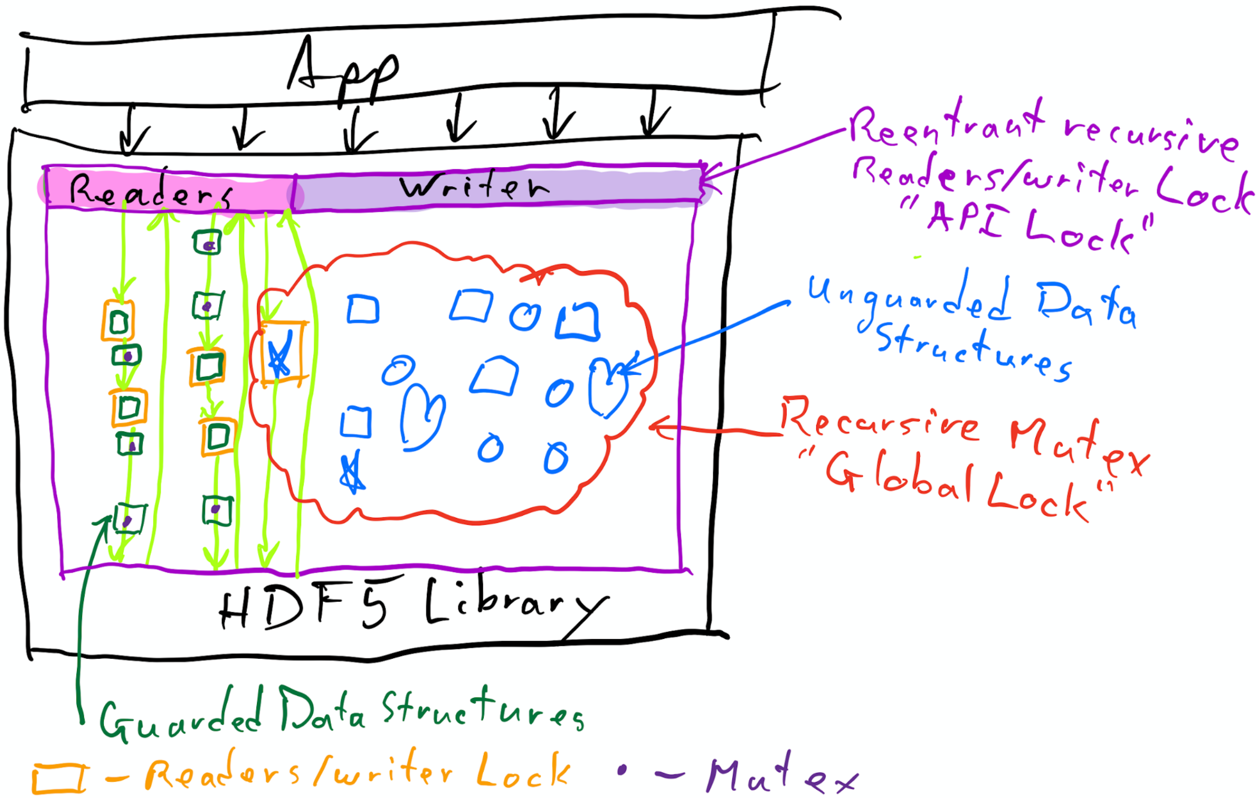
Read-Read



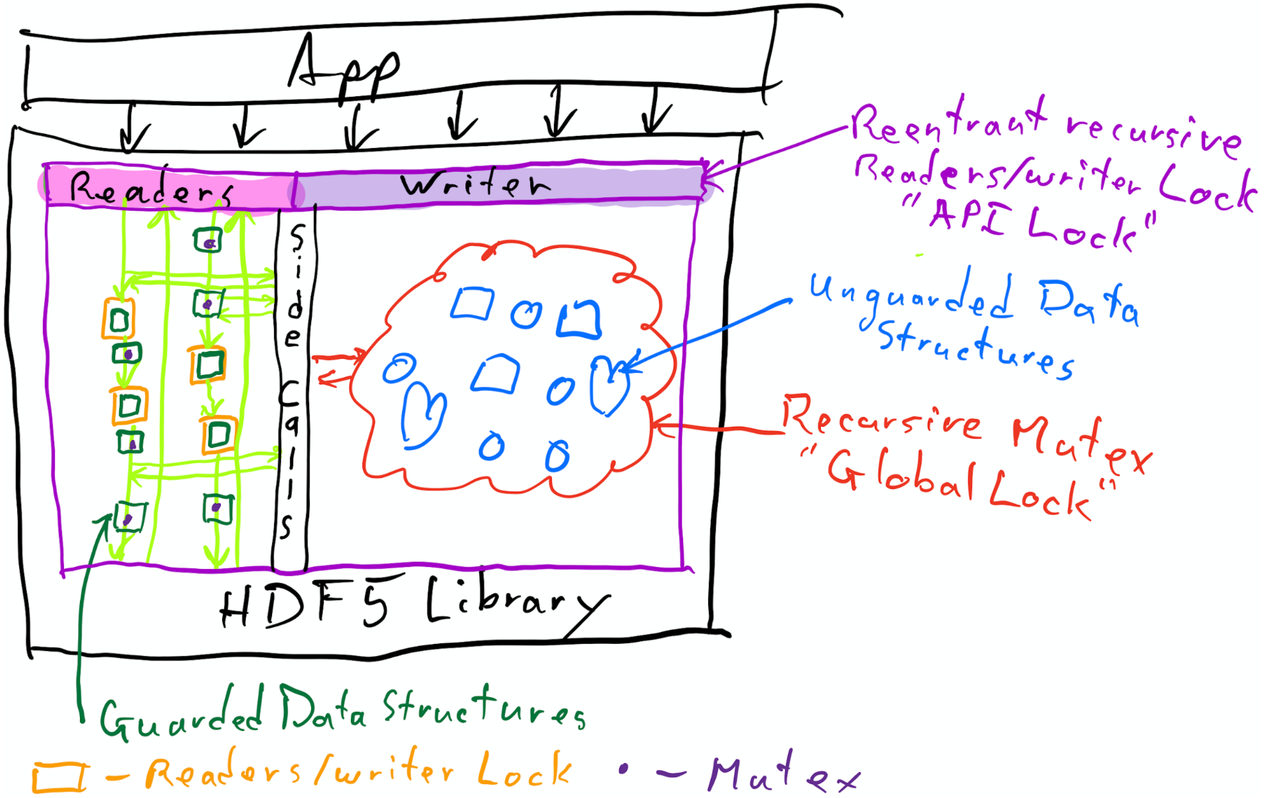
Read-Write



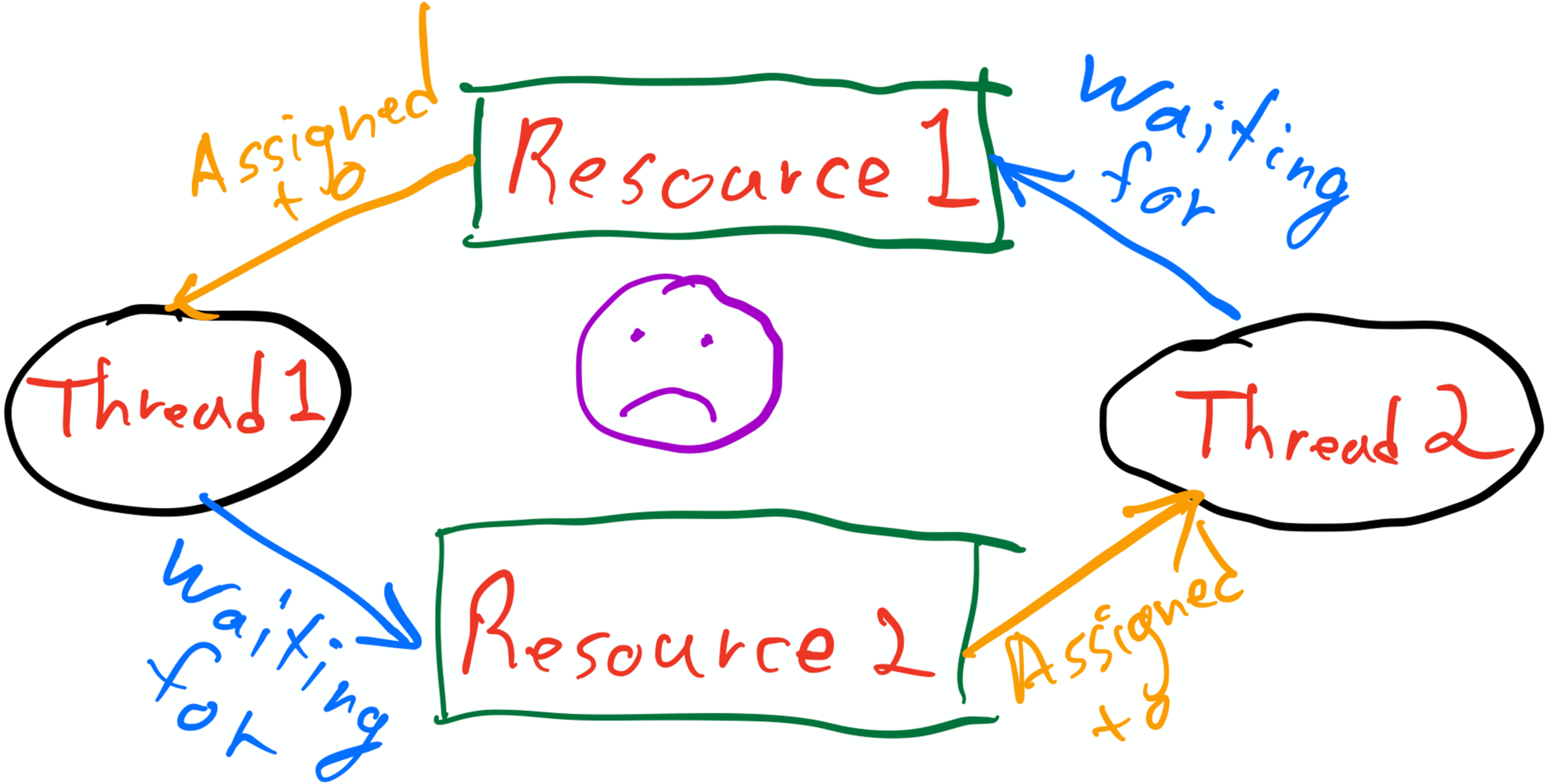
Guarded & Unguarded Access to Same Data Structure


App
Readers
Writer
Reentrant recursive Readers/writer Lock "API Lock"
Unguarded Data Structures
Recursive Mutex "Global Lock"
HDF5 Library
Guarded Data Structures
□ - Readers/writer Lock
• - Mutex

Are all of these locks required?



Avoiding Deadlocks



Coding Details

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 non-lock-acquisition 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