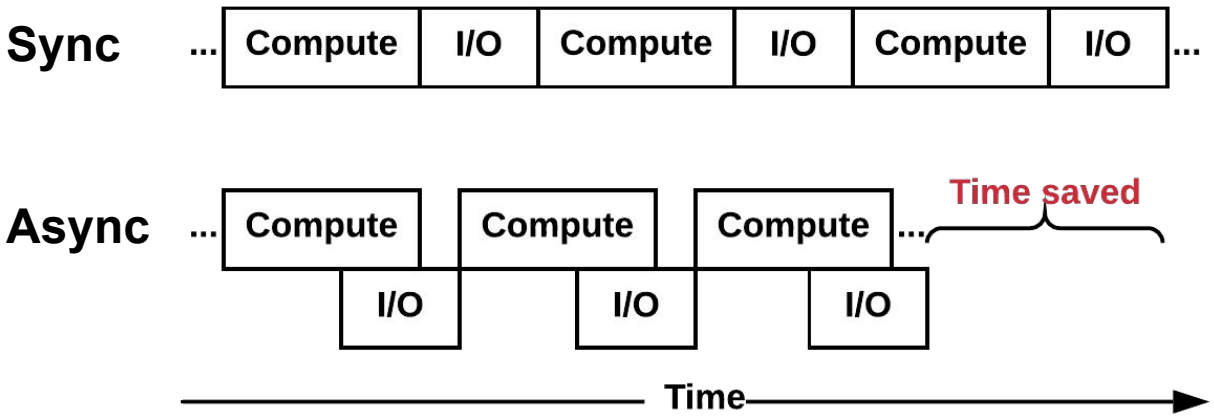
# Async I/O VOL: Transparent Asynchronous I/O using Background Threads

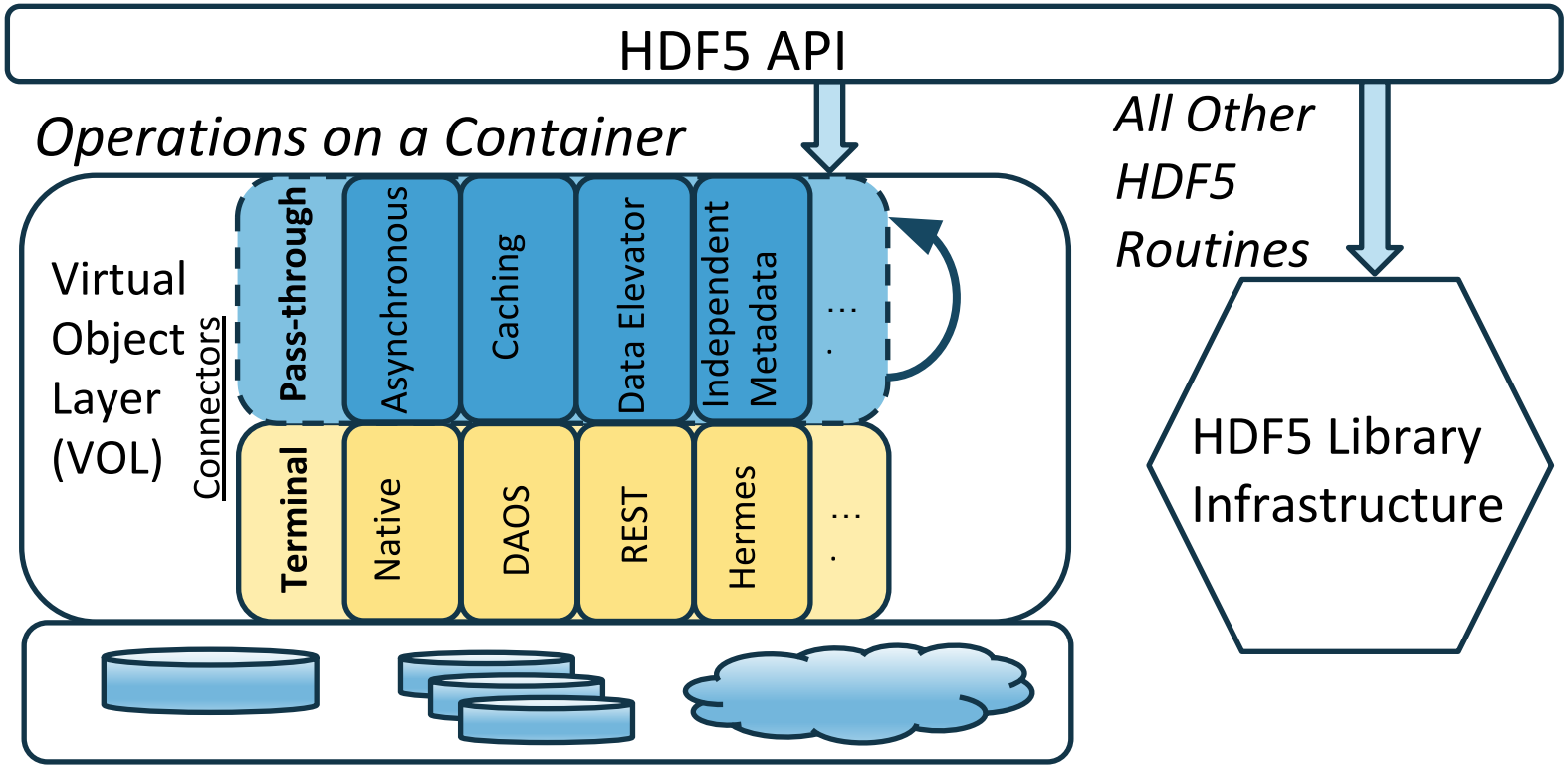
**Houjun Tang**[1], Quincey Koziol[1], Suren Byna[1], John Mainzer[2], Tonglin Li[1]

[1] Lawrence Berkeley Lab
[2] The HDF Group

# Why Async


Sync
...
Compute
I/O
Compute
I/O
Compute
I/O
...
Async
...
Compute
Compute
Compute
...
Time saved
I/O
I/O
I/O
Time

# Virtual Object Layer (VOL)



# *Implicit* and *Explicit* Asynchronous I/O Execution

- **Implicit**
  - For unmodified HDF5 applications
  - Can be transparently invoked by setting environment variable:
    - `export HDF5_VOL_CONNECTOR="async under_vol=0;under_info={}"`
    - `export HDF5_PLUGIN_PATH=<ASYNC_VOL_CONNECTOR_DIR>`
  - Dataset writes are non-blocking (with buffer copy), reads are always blocking

- **Explicit**
  - For applications that want more control of async operations
    - Uses an “event set” to manage async operations
  - Can extract more performance, e.g. enable async read, disable data copying

# Implicit Asynchronous Execution

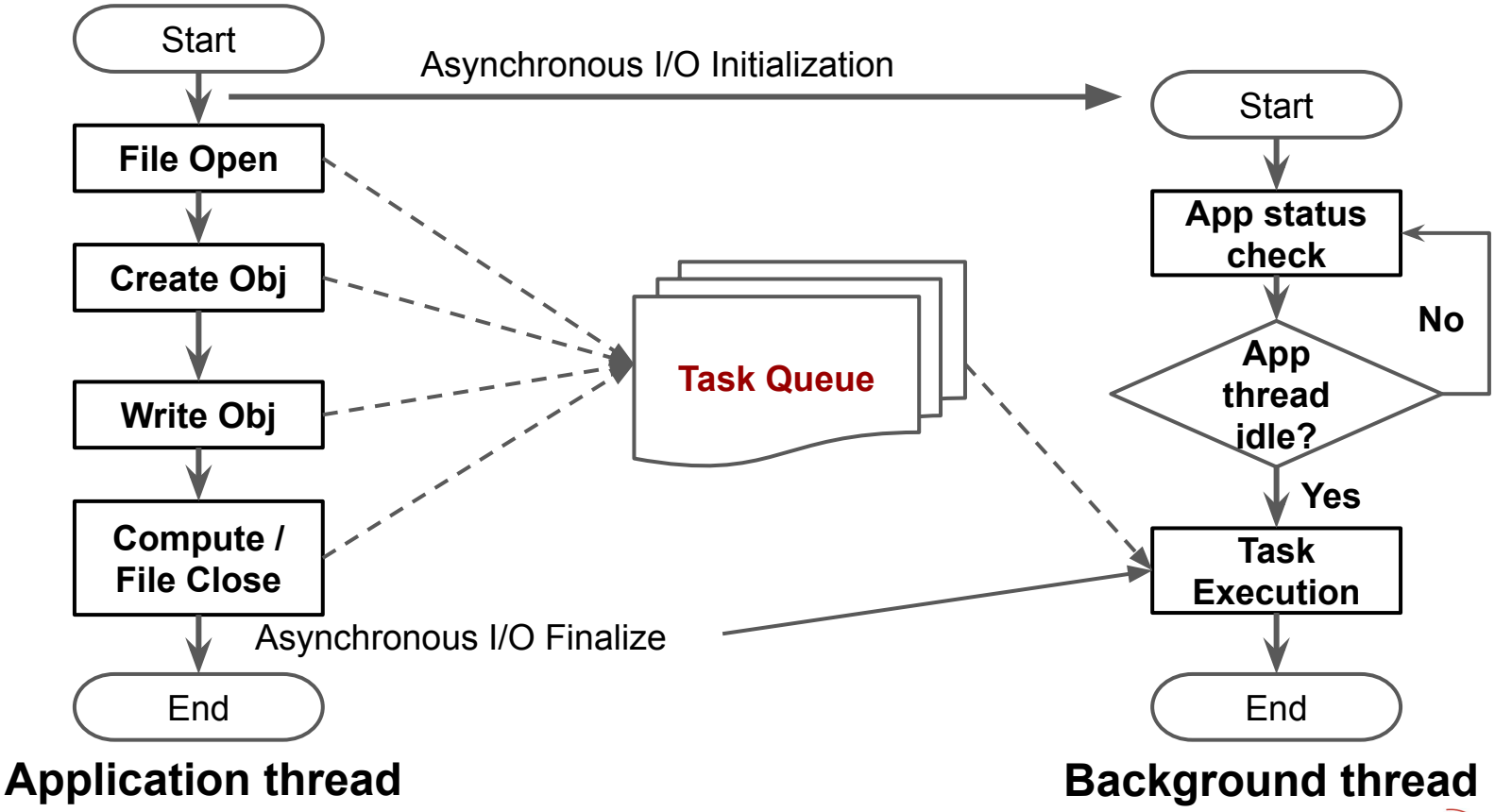
```
fid = H5Fopen(..);                              // Asynchronous, can start immediately
gid = H5Gopen(fid, ..);                         // Asynchronous, starts when H5Fopen completes
did = H5Dopen(gid, ..);                         // Asynchronous, starts when H5Gopen completes
status = H5Dwrite(did, ..);                     // Asynchronous, starts when H5Dopen completes
status = H5Dread(did, ..);                      // Synchronous, blocks until completes
...
<other user code>
...
did2 = H5Dopen(gid, ..);                        // Asynchronous, can start immediately
sid = H5Dget_space(did2);                       // Synchronous, starts when H5Dopen completes
status = H5Sget_simple_extent_dims(sid, ..);    // Synchronous
...
status = H5Fclose(fid);                         // Asynchronous

H5close() or end of main function               // Synchronous, waits for all previous
                                                // tasks operates on this file
```

# Explicit Asynchronous Execution

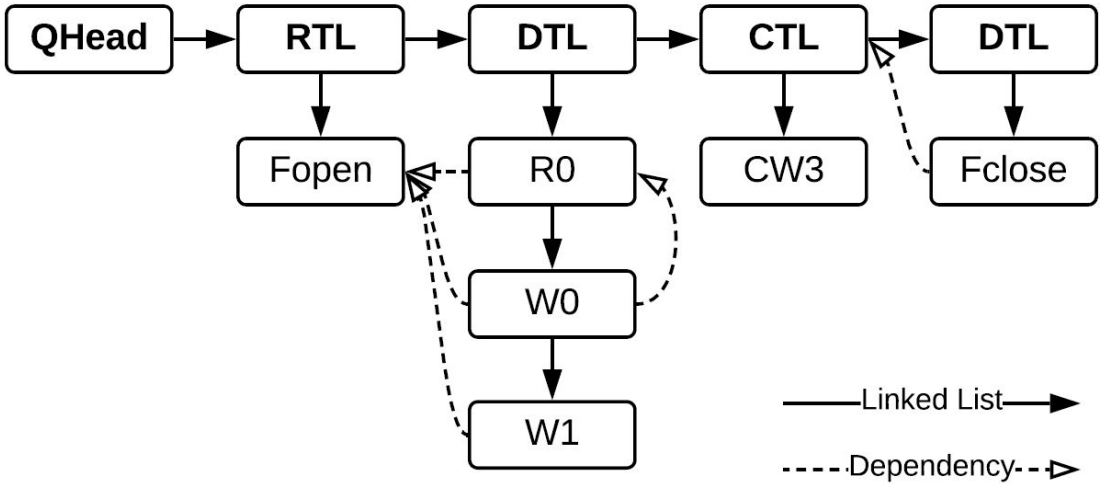
```
es_id = H5EScreate();                        // Create event set for tracking async operations
fid = H5Fopen_async(.., es_id);              // Asynchronous, can start immediately
gid = H5Gopen_async(fid, .., es_id);         // Asynchronous, starts when H5Fopen completes
did = H5Dopen_async(gid, .., es_id);         // Asynchronous, starts when H5Gopen completes
status = H5Dwrite_async(did, .., es_id);     // Asynchronous, starts when H5Dopen completes,
                                             //   may run concurrently with other H5Dwrite in event
                                  set
status = H5Dwrite_async(did, .., es_id);     // Asynchronous, starts when H5Dopen completes,
                                             //   may run concurrently with other H5Dwrite in event
                                  set
...
<other user code>
...
H5ESwait(es_id);                             // Wait for operations in event set to complete, buffers
                                             // used for H5Dwrite must only be changed after wait
```

# Workflow


Start
File Open
Create Obj
Write Obj
Compute / File Close
Asynchronous I/O Finalize
End
Application thread
Asynchronous I/O Initialization
Task Queue
Start
App status check
No
App thread idle?
Yes
Task Execution
End
Background thread

# Async I/O Task management

- Uses Argobots thread engine to manage task scheduling
- Regular tasks:
  - Fopen
- Dependent tasks:
  - R0, W0, W1, Fclose
- Collective task:
  - CW3



# Dependency management

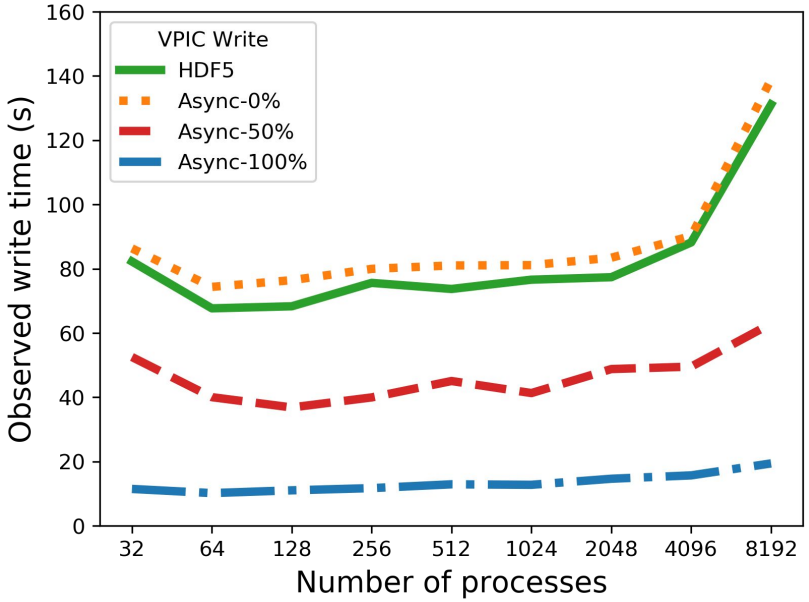
- File create / open operations must execute first.
- Any read / write operations to same object execute in application's order of issue.
- Collective operations on any object always execute in order, one at a time (never concurrently).
- File close starts after all existing tasks in the file have completed.

# Error Handling

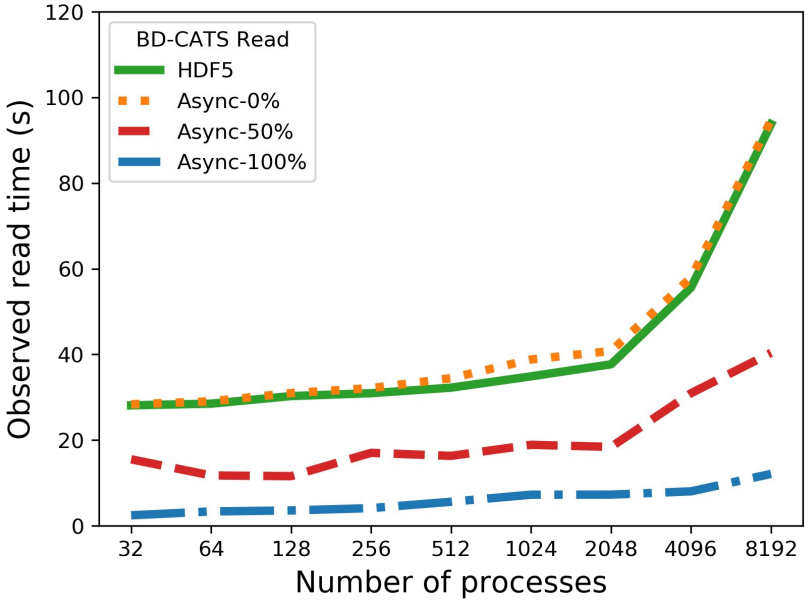
- If an async operation fails, all of its dependent children will not execute
- An additional error message indicating the parent's failure is appended the error to the error stack:

```
Async VOL-DIAG: Error detected in Async VOL (0.1) thread 0:
  #000: h5_vol_external_async_native.c line 5766 in async_dataset_create_fn(): Parent task failed
      major: Virtual Object Layer
      minor: Unable to create file
HDF5-DIAG: Error detected in HDF5 (1.13.0) thread 0:
  #001: ../../src/H5VLcallback.c line 3977 in H5VLgroup_create(): unable to create group
      major: Virtual Object Layer
      minor: Unable to create file
  #002: ../../src/H5VLcallback.c line 3904 in H5VL__group_create(): group create failed
      major: Virtual Object Layer
      minor: Unable to create file
  #003: ../../src/H5VLnative_group.c line 72 in H5VL__native_group_create(): unable to create group
      major: Symbol table
      minor: Unable to initialize object
...
```

# Evaluation

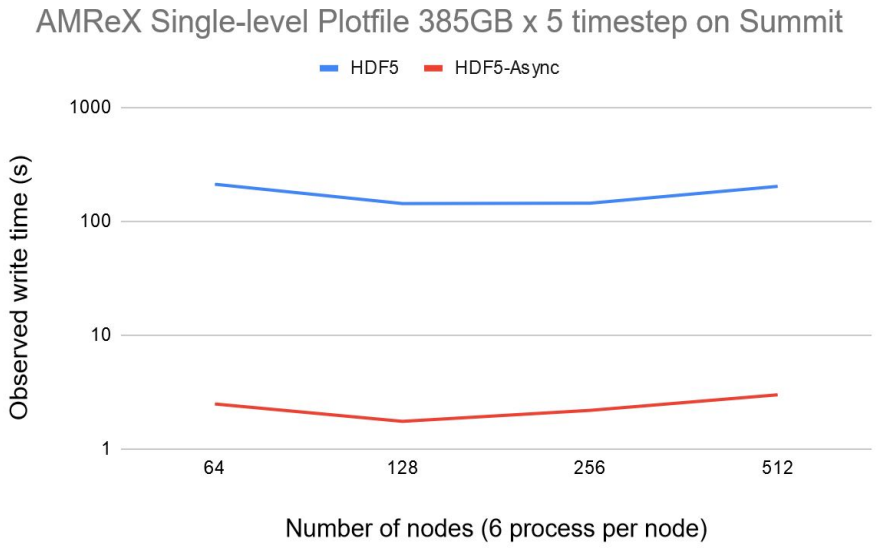


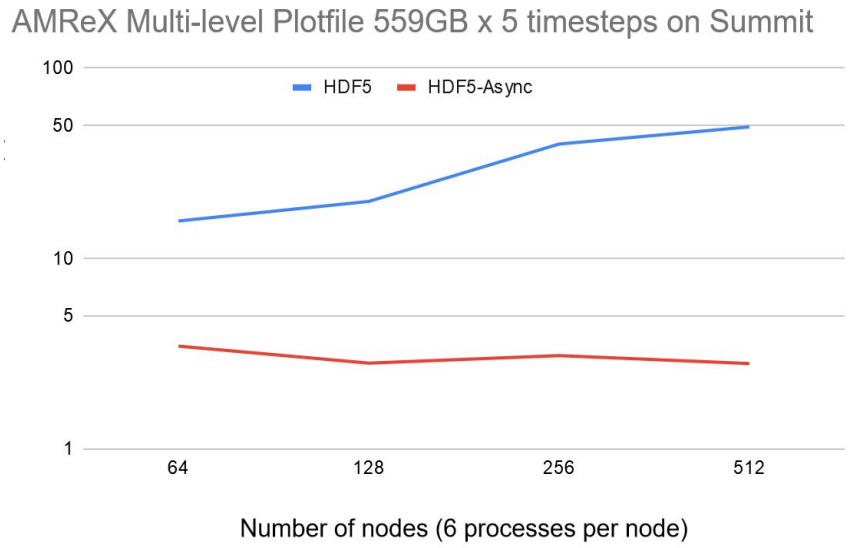

**VPIC-IO**




**BD-CATS-IO**

# Evaluation


AMReX Single-level Plotfile 385GB x 5 timestep on Summit
HDF5
HDF5-Async
1000
100
10
1
Observed write time (s)
64
128
256
512
Number of nodes (6 process per node)


AMReX Multi-level Plotfile 559GB x 5 timesteps on Summit
HDF5
HDF5-Async
100
50
10
5
1
64
128
256
512
Number of nodes (6 processes per node)

# Future Work

- Switch to TaskWorks thread engine
  - A portable, high-level, task engine designed for HPC workloads
  - Task dependency management, background thread execution.
- Merge compatible operations
  - If two async dataset write operations are putting data into same dataset, can merge into only one call to underlying VOL connector
  - Turn multiple ‘normal’ group create operations into a single ‘multi’ group create operation
- Multiple background threads
  - Needs HDF5 thread-safety work, to drop global mutex

https://bitbucket.hdfgroup.org/projects/HDF5VOL/repos/async