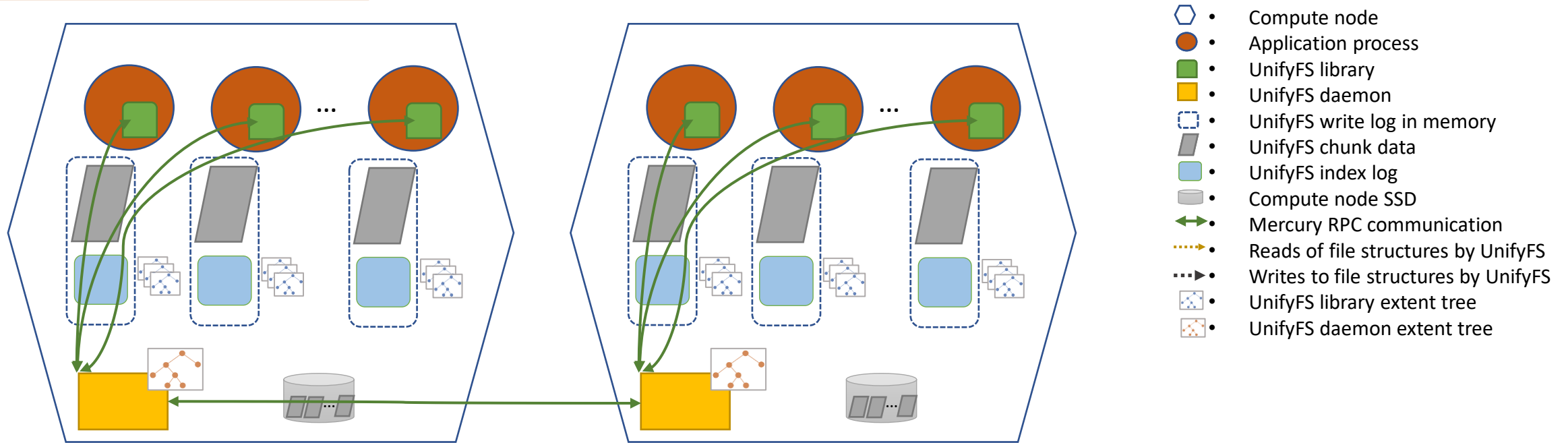




unifyFS Developer's Documentation

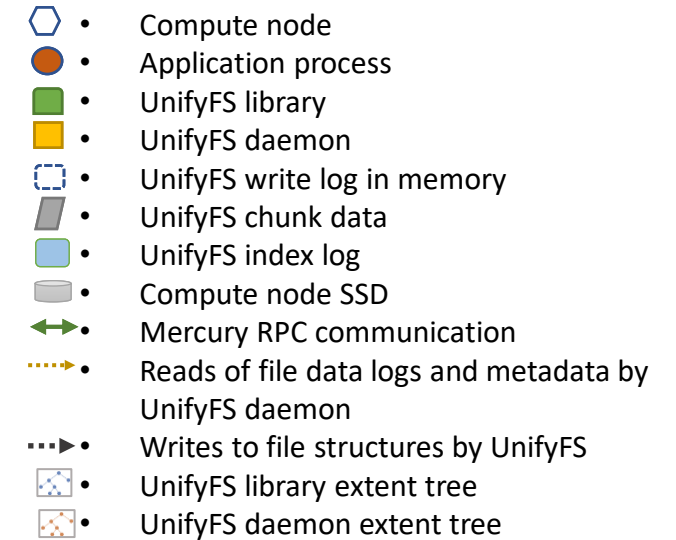
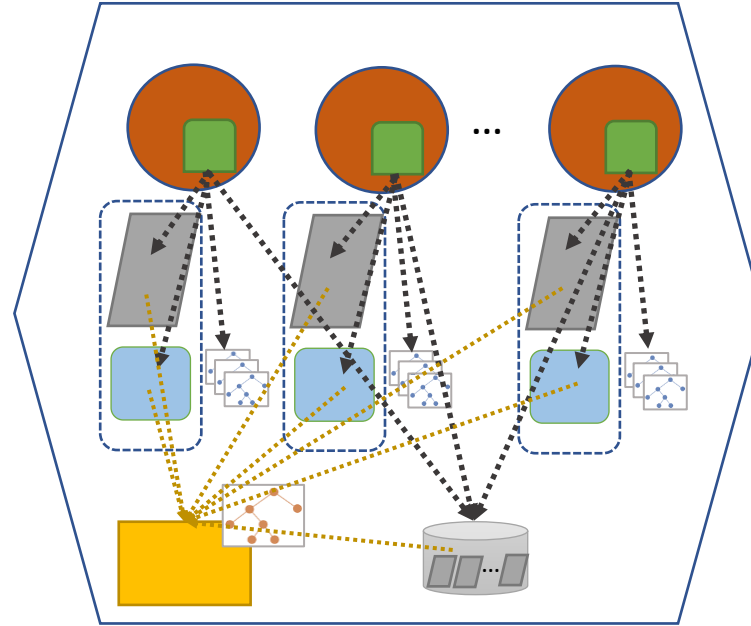
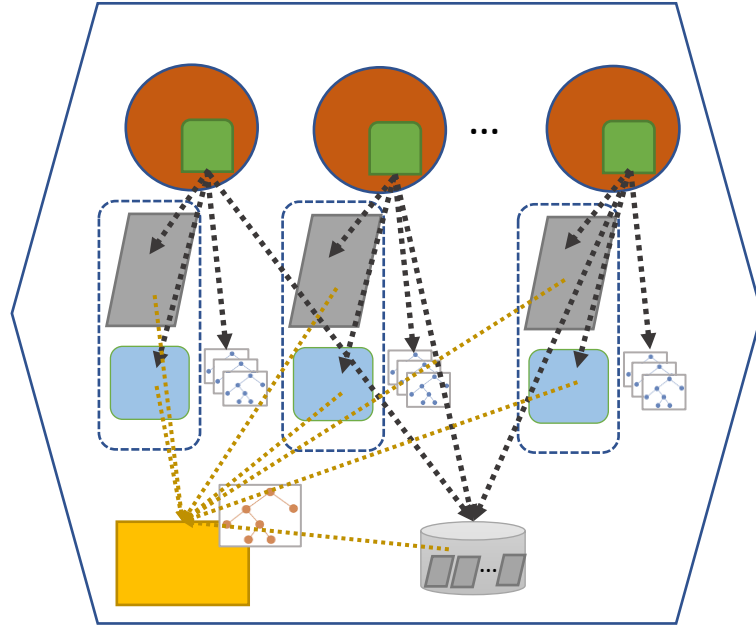
June 2020

High level design of UnifyFS



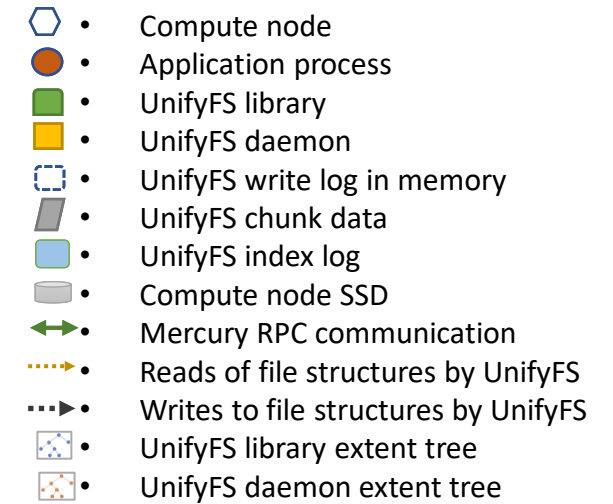
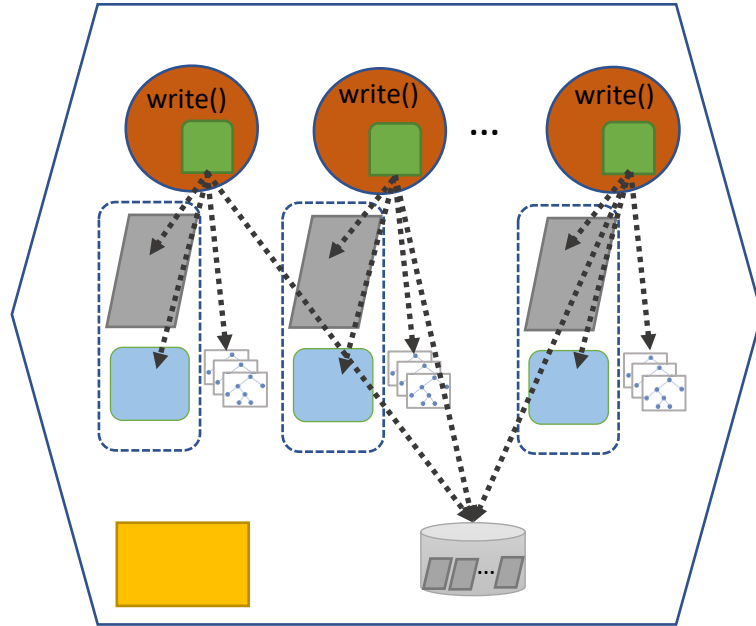
1. The UnifyFS library is linked into the user processes. The UnifyFS library intercepts I/O calls from the user process and manages file operations at the UnifyFS mount point (typically /unifyfs)
2. There is one UnifyFS daemon on each compute node.
3. Each UnifyFS library instance maintains several data structures:
 - a) Write log: Data structure that contains metadata and file data. File data can exist in memory or on SSD
 - i. Index log: Data structure for metadata in the write log
 - ii. Chunk data: Data structure for file data. Chunk data can exist in memory (primary location) or on SSD (as “spillover” if memory region is full)
 - b) Extent tree: Tree data structure for maintaining metadata for the process. There is one extent tree for each file.
4. Each UnifyFS daemon maintains an extent tree for metadata. The daemon’s extent tree contains global metadata information from all processes in the user’s job.
5. The UnifyFS library instances communicate with the local UnifyFS daemon via Mercury RPC. Data is exchanged via shared memory segments.
6. The UnifyFS daemons communicate with each other via Mercury RPC.

Where is data stored in UnifyFS?



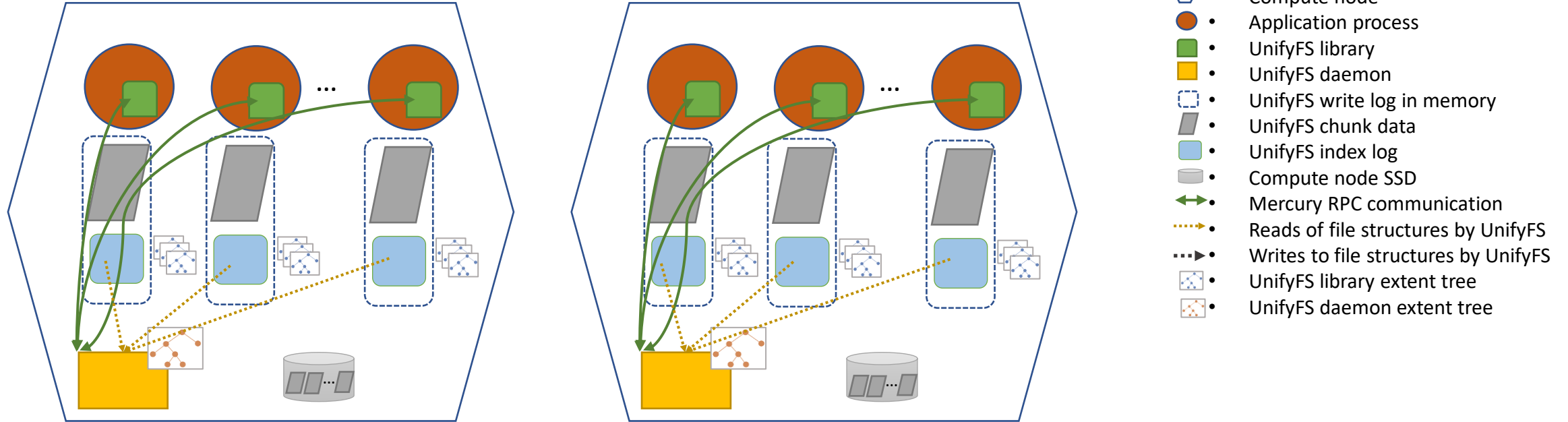
1. Application file data is stored in per-process write logs. For each process, there is one write log that contains data for all files written to by that process, and an entry in the write log for each write operation performed by that process. The write log consists of an index log that contains metadata, and chunk data storage that contains file data. The index log always exists in memory. By default, write log entries containing data are stored in chunk data in main memory until main memory is exhausted (limit set by user on a per process basis). If there is no free space in the in-memory write log, the data is stored in chunk data on the “spillover” location (SSD). Write log entries are written by the UnifyFS library. When in memory, the write log is stored in shared memory segments that are accessible by the UnifyFS daemon. The UnifyFS daemon accesses the write log when it receives a read request.
2. Metadata information in the UnifyFS library is stored in two structures: an index log that is shared for all files created by a process; and extent trees, one for each file written to by a process. The UnifyFS daemon reads the index log from shared memory when it receives an RPC message from the UnifyFS library (sync operation).
3. Metadata in the UnifyFS daemon is stored in an extent tree. The extent tree in the daemon contains global information from the entire job. The UnifyFS daemon’s extent tree is updated on Sync operations where the extent tree is updated to contain all metadata updates from local UnifyFS libraries as well as from all remote UnifyFS daemons in the job. (Note: currently, the UnifyFS daemon also stores metadata in MDHIM, but MDHIM is being phased out)

How does write() work in UnifyFS?



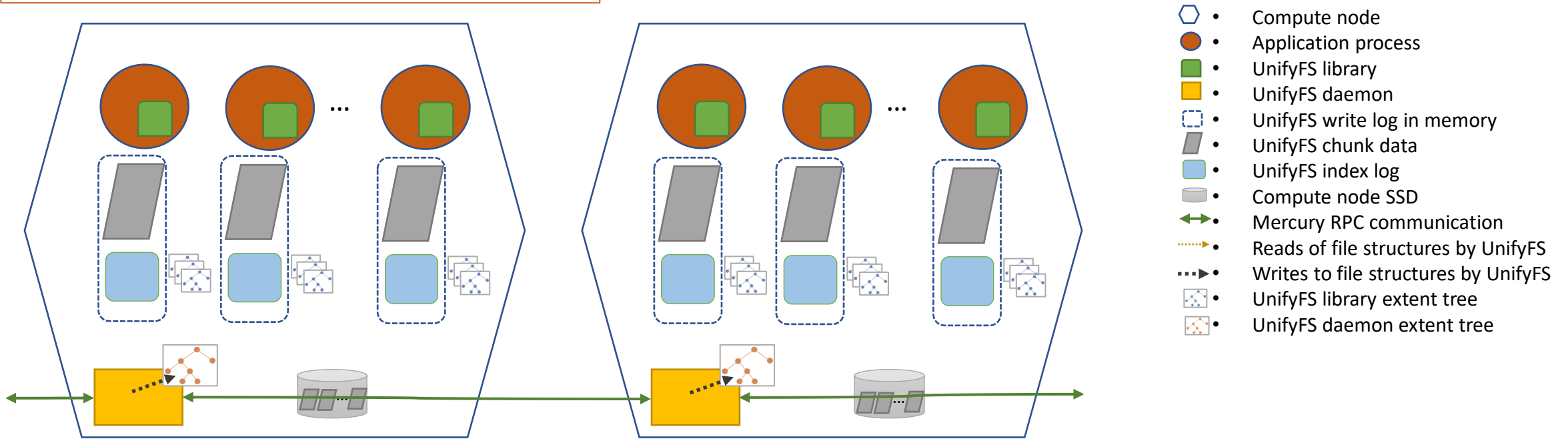
1. The user application executes write()
2. The write call is intercepted by the UnifyFS library. If the write operation applies to a file in /unifyfs then the operation is handled by UnifyFS (goto step 3). Otherwise the operation is passed to the system for handling and there is no further UnifyFS involvement in this operation.
3. The write operation is logged in local memory on the compute node by:
 - a. The UnifyFS library writes file data into chunk data in the write log using the first available free entry slot (slots can be made “free” on delete operations). If the chunk data space in the write log is full, the UnifyFS client writes the file data in chunk data in its write log on the SSD (spillover)
 - b. The UnifyFS library writes the metadata for the write operation into the index log which contains metadata for all files written to by the application process.
 - c. The UnifyFS library writes file metadata in the corresponding extent tree structure for the file

How is metadata synchronized in UnifyFS?



1. The user application executes `fflush()`, `close()`, `fsync()` (referred to as “Sync call” or “Sync operation” for the rest of this discussion)
2. The Sync call is intercepted by the UnifyFS library. If the Sync operation applies to a file in `/unifyfs` then the operation is handled by UnifyFS (goto step 3). Otherwise the operation is passed to the system for handling and there is no further UnifyFS involvement in this operation.
3. The UnifyFS library executes `fsync()` on the file containing write log chunk data on the SSD to ensure that it is flushed from memory buffers to the SSD.
4. The UnifyFS library sends an RPC message to the UnifyFS daemon to indicate that the library has metadata ready for the daemon
5. The UnifyFS daemon reads the data from the index log in shared memory and copies it to a private data structure
6. The UnifyFS daemon sends a message back to the UnifyFS library that it is done reading the metadata
7. The UnifyFS client resets the index log counter to 0 (effectively clears the index log)

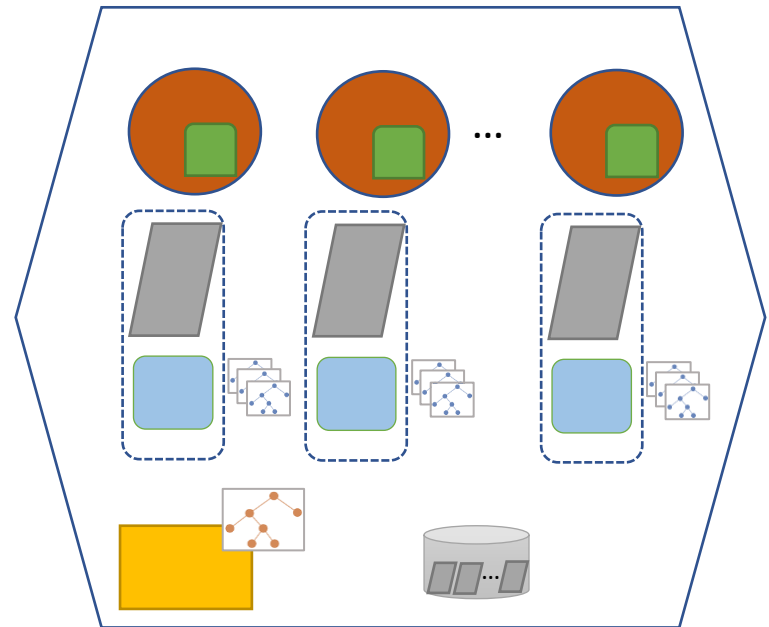
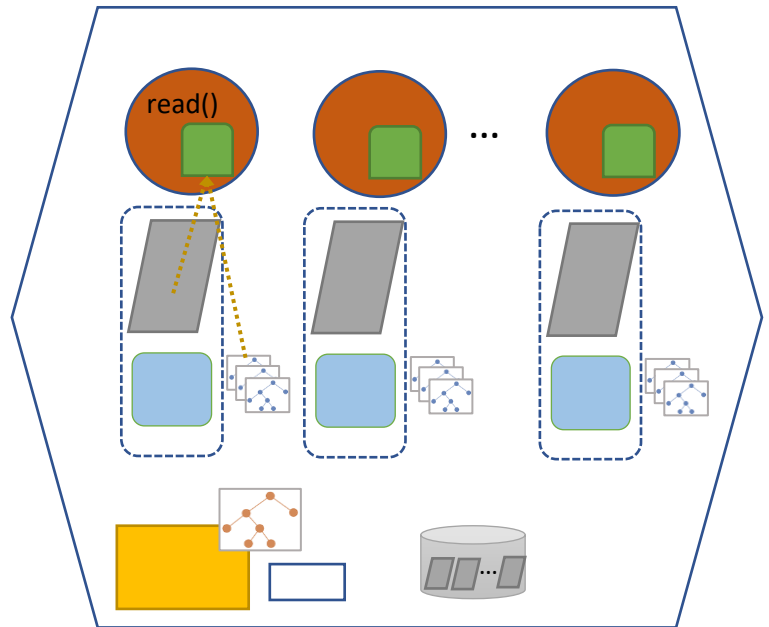
How is metadata synchronized in UnifyFS?

















7. The UnifyFS daemon stores the metadata information in its extent tree (and currently also enters key,value pairs into MDHIM)
8. With extent trees, synchronization of metadata is achieved by:
 - a. The UnifyFS daemon broadcasts its newly discovered metadata information to all other UnifyFS daemons in the user's job using Mercury RPC
 - b. Each UnifyFS daemon that is a receiver of the metadata information updates its local extent tree with the new metadata information
 - c. After the Sync operation completes, all UnifyFS daemons have a copy of all metadata known by all daemons

(With MDHIM, synchronization of metadata done within MDHIM when key,value pairs are entered. To locate metadata information for a particular file, offset, the UnifyFS daemon performs a range query to MDHIM to get all extent key,value pairs that cover a portion of the read request.)

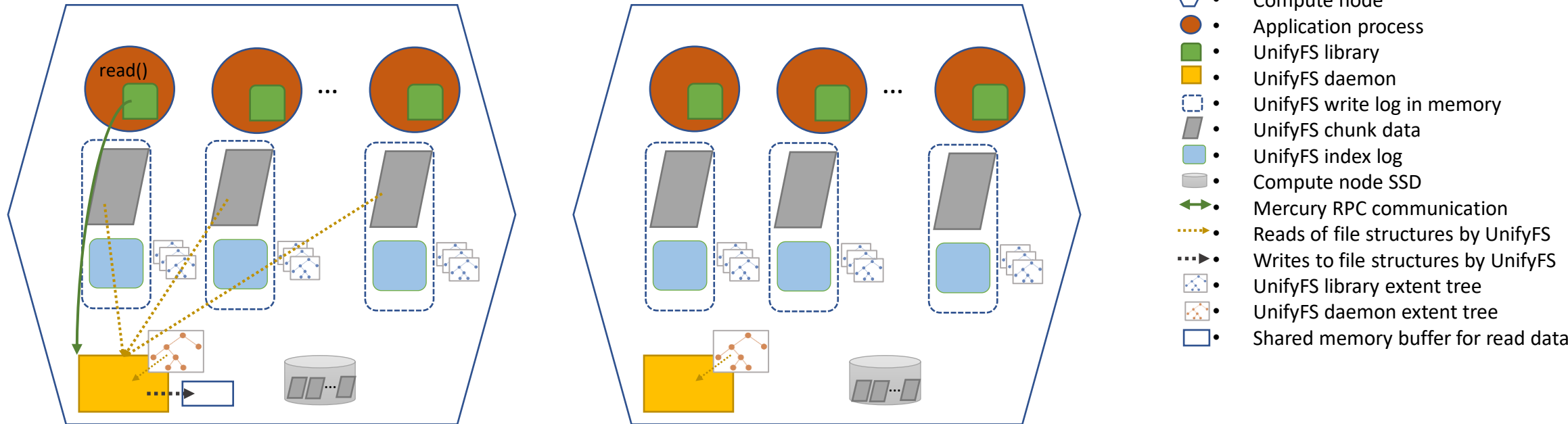
How does read() work in UnifyFS?



-  • Compute node
-  • Application process
-  • UnifyFS library
-  • UnifyFS daemon
-  • UnifyFS write log in memory
-  • UnifyFS chunk data
-  • UnifyFS index log
-  • Compute node SSD
-  • Mercury RPC communication
-  • Reads of file structures by UnifyFS
-  • Writes to file structures by UnifyFS
-  • UnifyFS library extent tree
-  • UnifyFS daemon extent tree
-  • Shared memory buffer for read data

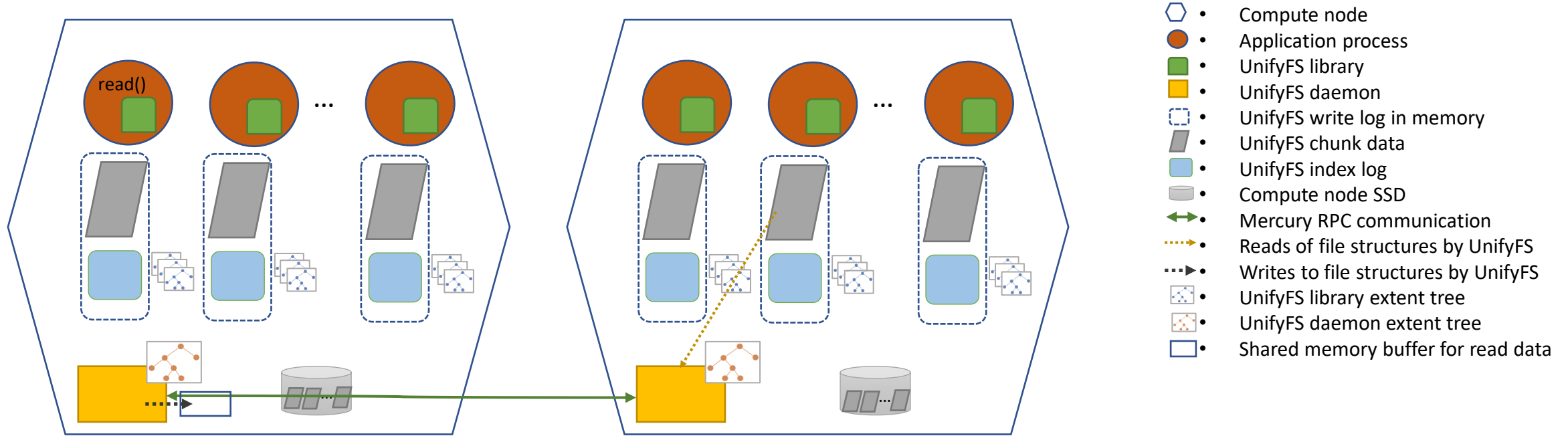
1. The user application executes read(). In this example, only one process is executing read (leftmost process)
2. The read call is intercepted by the UnifyFS library. If the read operation applies to a file in /unifyfs then the operation is handled by UnifyFS (goto step 3). Otherwise the operation is passed to the system for handling and there is no further UnifyFS involvement in this operation.
3. If "local read optimization" is turned on, then the UnifyFS library looks in its extent tree to see if it has all of the needed metadata information to locate the data. If all metadata is found in the UnifyFS library extent tree, then the UnifyFS library copies the requested data from the local write log into the return buffer. The operation is complete and the user application returns from read().

How does read() work in UnifyFS?



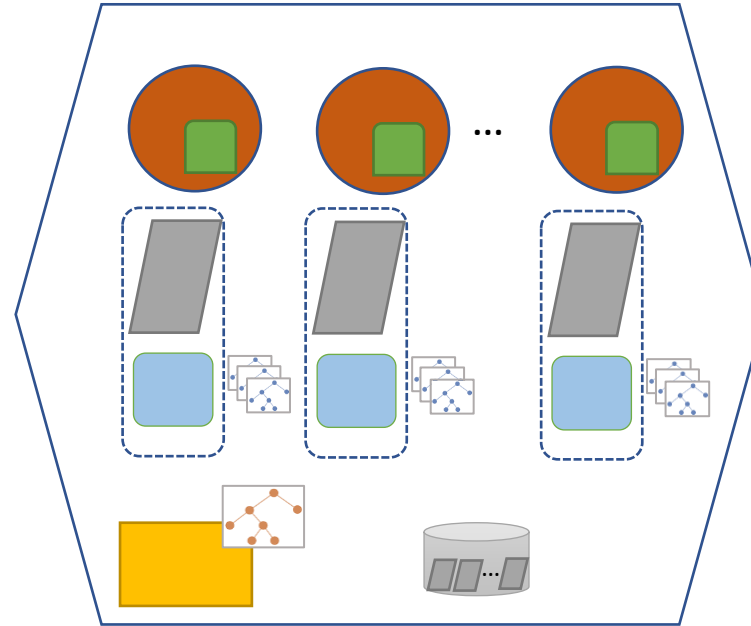
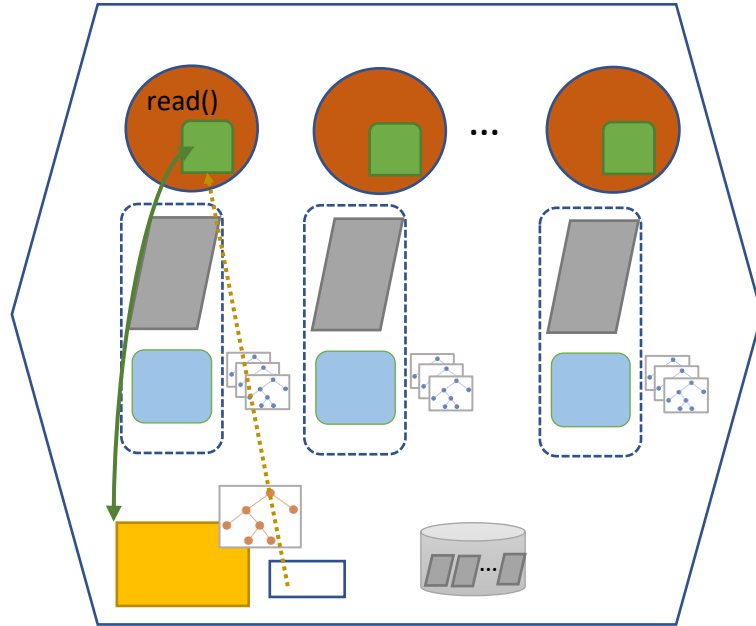
4. Else //(If “local read optimization” is not on) or (if “local read optimization” is on but not all needed metadata can be found in the local extent tree)
 - a) The UnifyFS library sends an RPC to its local UnifyFS daemon to request a read of (fd, offset, bytes)
 - b) The UnifyFS daemon looks in its extent tree to find the metadata for the read request. The UnifyFS daemon computes which subsets of the request exist on the local compute node and on remote compute nodes. Note that the data for a single user request may be stored across multiple nodes. In this example, on the local node, data to fulfill the request can be found on 3 processes including the requesting process. (For MDHIM the daemon performs a range query to find all key,value pairs matching the request. The value returned contains the ids of the remote UnifyFS daemons and the locations of the write log entries)
 - c) For data located on the local compute node, the UnifyFS daemon copies the requested data from the write log into a buffer in shared memory

How does read() work in UnifyFS?



4. Else //(If “local read optimization” is not on) or (if “local read optimization” is on but not all needed metadata can be found in the local extent tree)
 - a) The UnifyFS library sends an RPC to its local UnifyFS daemon to request a read of (fd, offset, bytes)
 - b) The UnifyFS daemon looks in its extent tree to find the metadata for the read request. The UnifyFS daemon computes which subsets of the request exist on the local compute node and on remote compute nodes. Note that the data for a single user request may be stored across multiple nodes. In this example, on the local node, data to fulfill the request can be found on 3 processes including the requesting process. (For MDHIM the daemon performs a range query to find all key,value pairs matching the request. The value returned contains the ids of the remote UnifyFS daemons and the locations of the write log entries)
 - c) For data located on the local compute node, the UnifyFS daemon copies the requested data from the write log into a buffer in shared memory
 - d) For any data not located on the local compute node
 - i. the UnifyFS daemon sends an RPC message to the UnifyFS daemon that owns the data on another compute node for the subset of the request that is on the remote node (fd, offset, bytes)
 - ii. The remote UnifyFS daemon locates the requested data in the local write log, copies it into a buffer, and sends an RPC message with the data to the requesting (local) UnifyFS daemon. In this example, the data on the remote node was written by one process (2nd from the left).
 - iii. The requesting UnifyFS daemon copies the received data into the shared memory buffer

How does read() work in UnifyFS?



- Compute node
- Application process
- UnifyFS library
- UnifyFS daemon
- UnifyFS write log in memory
- UnifyFS chunk data
- UnifyFS index log
- Compute node SSD
- Mercury RPC communication
- Reads of file structures by UnifyFS
- Writes to file structures by UnifyFS
- UnifyFS library extent tree
- UnifyFS daemon extent tree
- Shared memory buffer for read data

- Once all local and remote data has been collected by the local UnifyFS daemon, the following loop continues while there is still data to return to the UnifyFS library
 - The UnifyFS daemon copies requested read data into the buffer in shared memory (until the buffer is full)
 - The UnifyFS daemon notifies the UnifyFS library that the data is ready to be read via an RPC message
 - The UnifyFS library copies the data from the buffer in shared memory into a buffer to return to the user
 - The UnifyFS library sends a "done" RPC message to the UnifyFS daemon to indicate that the data has been read and the shared memory buffer can be reused
- When all requested data has been given to the UnifyFS library via the shared buffer, the server sends an RPC message to the UnifyFS library that it is done sending data
- The UnifyFS library returns the requested data to the user (returns from `read()`)