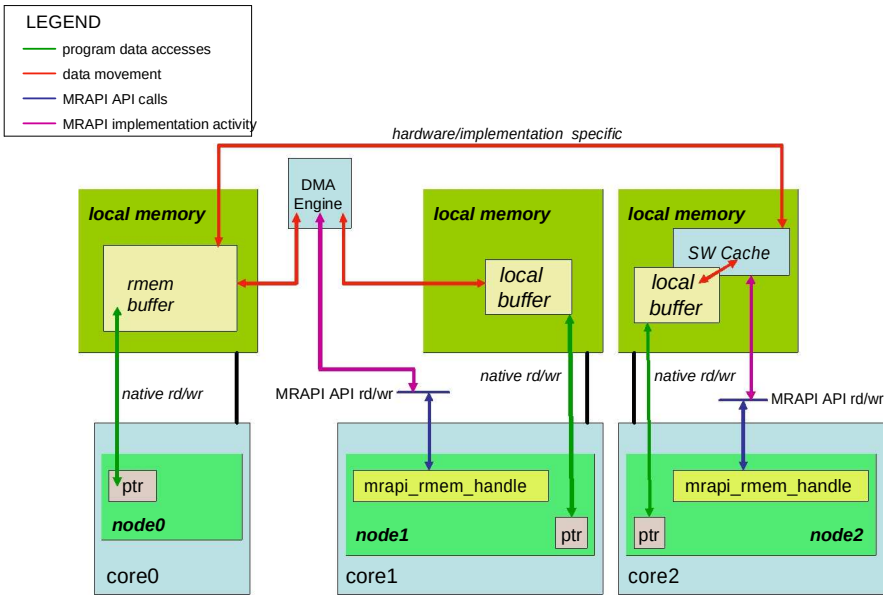

Figure 1 - Remote Memory Concepts

Figure 1 depicts the remote memory concepts in MRAPI. Access semantics are per rmem buffer instance, as follows:

- *Strict* semantics: the type of MRAPI access { DMA, sw cache, … } is defined at the time a rmem buffer is created. All MRAPI accesses to that rmem buffer must be of a uniform type. Each client of the buffer specifies an access type when on the get call and it is an error to request an access type other than that which was used to create the buffer.

- *Any* semantics: the type of MRAPI access {DMA, sw cache, …} is set to `MRAPI_RMEM_ATYPE_ANY` at the time the rmem buffer is created. When a client handle attaches it may specify any access type supported by the MRAPI implementation. Different types of accesses are supported concurrently. (Note that `MRAPI_RMEM_ATYPE_ANY` is only allowed for buffer creation, clients must call get using a specific access type, e.g., `MRAPI_RMEM_ATYPE_DEFAULT`, or other types provided by the implementation such as DMA, etc.)

Local pointer based read/write is always allowed (limited to access of local target buffers on clients). However, coherency issues must be managed by the application using MRAPI flush and synch calls (Sections 4.4.2.13 and 4.4.2.14). MRAPI implementations must guarantee that the effect of a synch operation must be complete before the next local read/write operation on the remote memory segment., and that the flush operation must block until it has completed.

Note that remote accesses (reads or writes) always results in a copy and must use MRAPI calls. Finally, implementations may define multiple access types (depending on underlying silicon capabilities), but must provide `MRAPI_RMEM_ATYPE_DEFAULT`, which has strict semantics and is guaranteed to work.

3.5.3 MRAPI Shared Memory Duplication (ABB Extension)

Shared memory is synchronized between processes by the operating system. The timing when updates are completed is non-deterministic and traditionally is guaranteed by use of locks. For some operating systems this heavy handed approach may not be needed and atomic operations

may be sufficient. For example, on Windows it is possible for multiple processes to reference the same physical memory by duplicating and exchanging a handle in a secure manner. Then atomic operations on the referenced shared memory work the same as if the processes were in the same address space. Shared memory IDs on non-Windows platforms are named globally so duplicating between processes is simply a matter of copying the ID, however simple atomic operations in the shared memory will not have the same effect.