common on a processor like Cell, where SPEs have just 256K local store, but the PPE is connected to a large main memory.

The idea in this use case is that Node 1 has a thread which is monitoring a thread on Node 2. The Node 1 thread sleeps until it receives a message from Node 2, either saying “stop”, or saying “I need more memory”. In the latter case, Node 1 allocates a new buffer of memory locally, and uses MRAPI to promote this to be remotely accessible, sending Node 2 the corresponding remote memory id. Node 2 can then use the memory as desired.

Once Node 2 completes, Node 1 can reclaim all the memory.

```
/*-------------------------------------------------*/
/* Definitions common to Node 1 and Node 2        */
/*-------------------------------------------------*/

/* Integer constants */
#define QUIT ...
#define MORE_DATA_PLEASE ...
#define AGREED_ACCESS_TYPE ...

/*-------------------------------------------------*/
/* Node 1 side of use case                        */
/*-------------------------------------------------*/

/* Helper functions for Node 1 - these are not part of MRAPI, and could be
   implemented using various appropriate mechanisms */

/* Function which uses some mechanism (e.g. MCAPI) to send a remote memory
id to Node 2 */
void send_id_to_node2(mrapi_rmem_id_t);

/* Function via which Node 1 waits for Node 2 to send an integer message */
void wait_for_message_from_node_2(int*);

/* Function via which Node 1 gets a fresh remote memory id */
mrapi_rmem_id_t get_fresh_rmem_id()


typedef struct Memory_Region_s
{
      char* pointer; /* A local buffer */
      mrapi_rmem_hndl_t handle; /* The remote memory handle associated with
this buffer */
} Memory_Region;

/* Array to keep track of memory which has been made available remotely */
Memory_Region buffers[MAX];


/* Code for Node 1 functionality */
int node1_remote_memory_use_case_2()
```