7 Memory Management

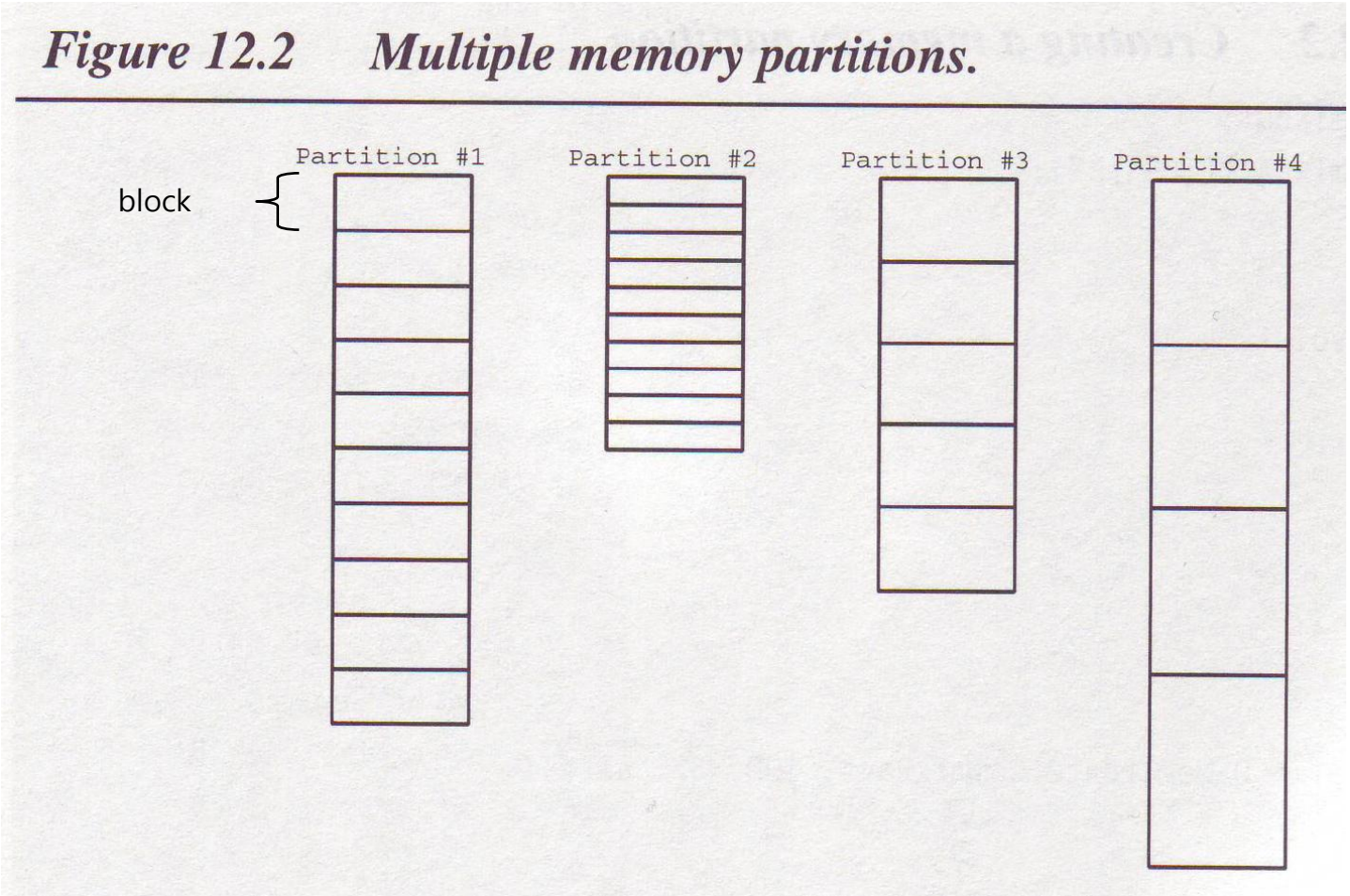
Contents

- Memory Control Block
- Creating a Partition: `OSMemCreate()`
- Obtaining a Memory Block: `OSMemGet()`
- Returning a Memory Block: `OSMemPut()`
- Obtaining Status of a Memory Partition: `OSMemQuery()`

Memory Management

- Dynamic memory allocation - ANSI C `malloc()` and `free()`
- Using `malloc()` and `free()` in embedded real-time system is dangerous because of fragmentation and non-deterministic
- uC/OS-II provides similar functions that require constant time for allocation and deallocation

Multiple Memory Partitions

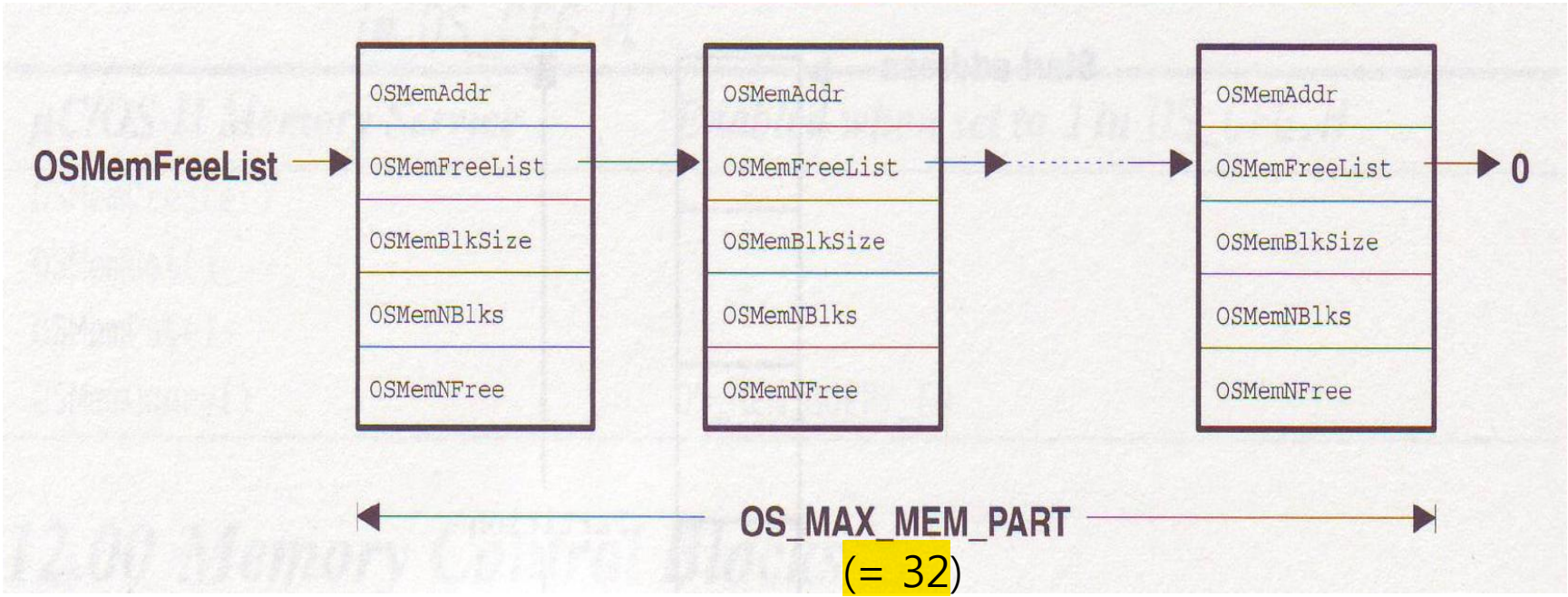


Memory Control Block

- Each memory partition has its own memory control block

```
typedef struct {
  void   *OSMemAddr;      // base address of partition
  void   *OSMemFreeList;  // next free MCB or block
  INT32U  OSMemBlkSize;   // block size
  INT32U  OSMemNBlks;     // total # of memory blocks
  INT32U  OSMemNFree;     // # of free memory blocks
} OS_MEM;
```

List of Free Memory Control Blocks



Creating a Partition: `OSMemCreate()`

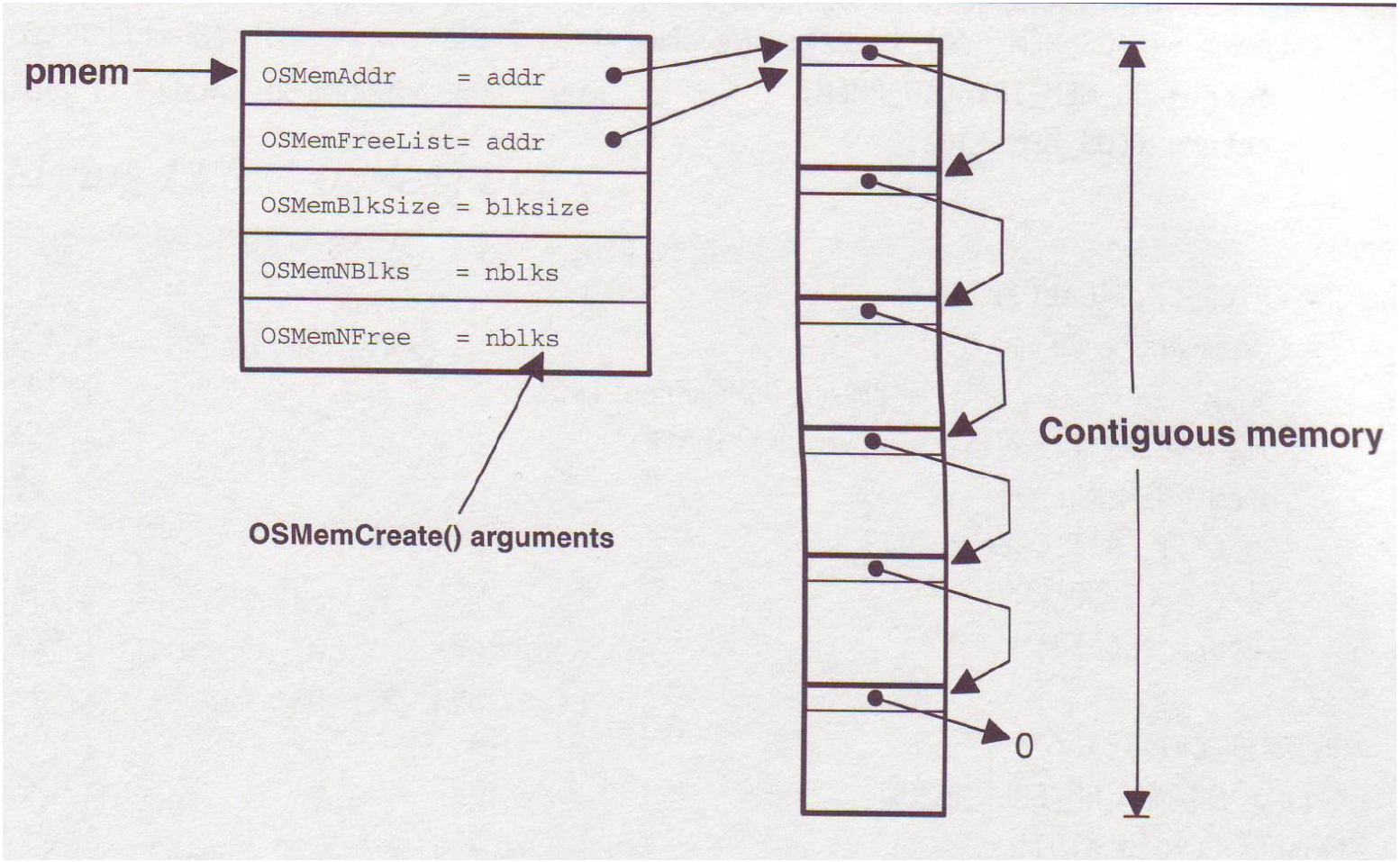
```
OS_MEM *CommTxBuf;
INT8U   CommTxPart[100][32];


void main (void)
{
    INT8U err;

    OSInit();
    .
    .
    CommTxBuf = OSMemCreate(CommTxPart, 100, 32, &err);
    .
    .
    OSStart();
}
```

Base address of partition

No. of blocks

Size of each block

Memory Partition Created by `OSMemCreate()`



Creating a Partition: `OSMemCreate()`

```
OS_MEM  *OSMemCreate (void *addr,
    INT32U nblks, INT32U blksize, INT8U
    *err)
{
  OS_MEM    *pmem;
  INT8U     *pblk;
  void     **plink;
  INT32U     i;

  OS_ENTER_CRITICAL();
  pmem = OSMemFreeList;
  if (OSMemFreeList != (OS_MEM *)0) {
    OSMemFreeList = (OS_MEM *)
                    OSMemFreeList-
    >OSMemFreeList;
  }
  OS_EXIT_CRITICAL();
  if (pmem == (OS_MEM *)0) {
    *err = OS_MEM_INVALID_PART;
    return ((OS_MEM *)0);
  }
  plink = (void **)addr;
  pblk  = (INT8U *)addr + blksize;
  for (i = 0; i < (nblks - 1); i++) {
    *plink = (void *)pblk;
    plink  = (void **)pblk;
    pblk   = pblk + blksize;
  }
  *plink               = (void *)0;
  pmem->OSMemAddr      = addr;
  pmem->OSMemFreeList  = addr;
  pmem->OSMemNFree     = nblks;
  pmem->OSMemNBlks     = nblks;
  pmem->OSMemBlkSize   = blksize;
  *err                 = OS_NO_ERR;
  return (pmem);
}
```

Initialize the partition

Initialize MCB

Obtaining a Memory Block: OSMemGet()

```
void  *OSMemGet (OS_MEM *pmem, INT8U *err)
{
  void      *pblk;

  OS_ENTER_CRITICAL();
  if (pmem->OSMemNFree > 0) {
    pblk = pmem->OSMemFreeList;
    pmem->OSMemFreeList = *(void **)pblk;
    pmem->OSMemNFree--;
    OS_EXIT_CRITICAL();
    *err = OS_NO_ERR;
    return (pblk);
  }
  OS_EXIT_CRITICAL();
  *err = OS_MEM_NO_FREE_BLKS;
  return ((void *)0);
}
```

Returning a Memory Block: OSMemPut()

```
INT8U  OSMemPut (OS_MEM  *pmem, void *pblk)
{
  OS_ENTER_CRITICAL();
  if (pmem->OSMemNFree >= pmem->OSMemNBlks) {
    OS_EXIT_CRITICAL();
    return (OS_MEM_FULL);
  }
  *(void **)pblk = pmem->OSMemFreeList;
  pmem->OSMemFreeList = pblk;
  pmem->OSMemNFree++;
  OS_EXIT_CRITICAL();
  return (OS_NO_ERR);
}
```

Obtaining Status of a Memory Partition: OSMemQuery()

```
INT8U  OSMemQuery (OS_MEM *pmem, OS_MEM_DATA
  *pdata)
{
  ...
}


typedef struct {
  void   *OSAddr;        // base address of the partition
  void   *OSFreeList;    // pointer to the free list of blocks
  INT32U  OSBlkSize;     // size of each block
  INT32U  OSNBlks;       // total # of blocks
  INT32U  OSNFree;       // # of free blocks
  INT32U  OSNUsed;       // # of used blocks
} OS_MEM_DATA
```