Memory Management

Programming Languages:
Principles and Paradigms

Maurizio Gabbrielli · Simone Martini

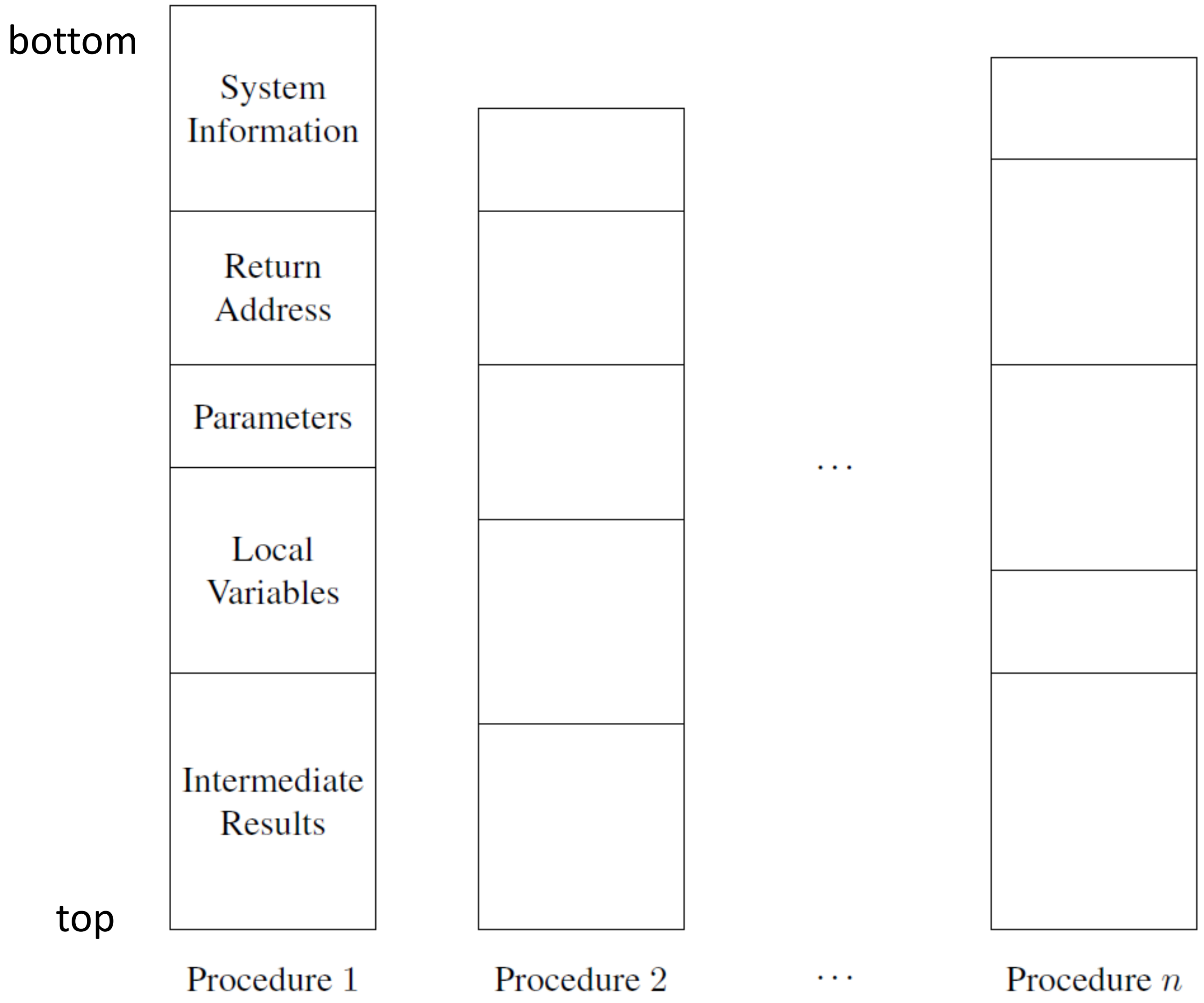
Techniques for Memory Management (MM)

- Management of the *allocation of memory* for programs/data
  - how they must be arranged in memory,
  - how much time they may *remain* and
  - which auxiliary structures are required to fetch information from memory.
- In a low-level abstract machine, MM is simple & *static*.
- In a high-level language, MM is complicated & *dynamic*.
  - For a *recursive* procedure, every procedure call requires its *own memory space* to store parameters, intermediate results, return addresses, etc.
  - Dynamic memory (de)allocation using a *stack* may be done.
  - Other case: Dynamic MM with *explicit* (de)allocation operation with a memory structure, called a *heap*.
    - E.g.) `malloc`/`free` command in C.

Static Memory Management

- Static MM performed by a *compiler before* the execution.
  - Statically allocated memory objects reside in a fixed zone of memory and they *remain* there for the entire duration of execution.
  - E.g.) global variables, object code instructions, constant, compiler-generated table.
- A language that doesn't support recursion performs static MM.
  - A local information of a subprogram:
    - local variables, parameters, return address, temporary values.
  - Successive calls to the same procedure share the same memory areas.

Static Memory Management

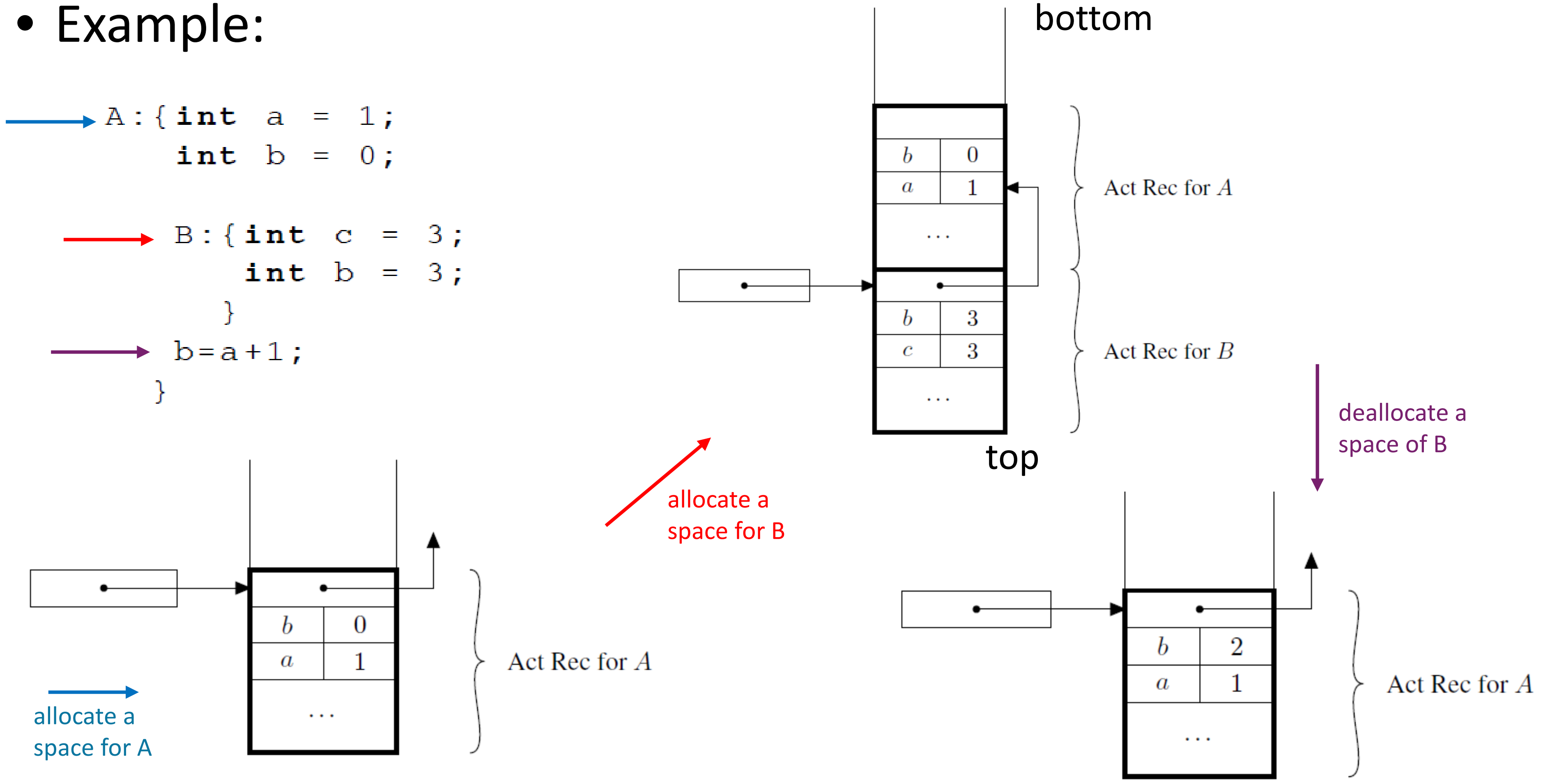


Dynamic Memory Management using Stacks

- Most modern PL with block structuring of programs.
- Blocks are entered/left using the LIFO scheme.
- Example:

```
A:{int a = 1;
   int b = 0;

   B:{int c = 3;
      int b = 3;
     }
   b=a+1;
  }
```

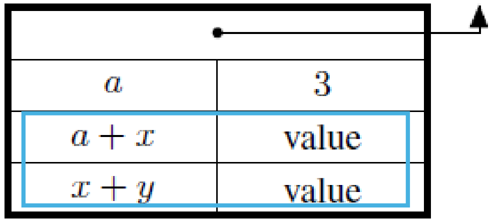


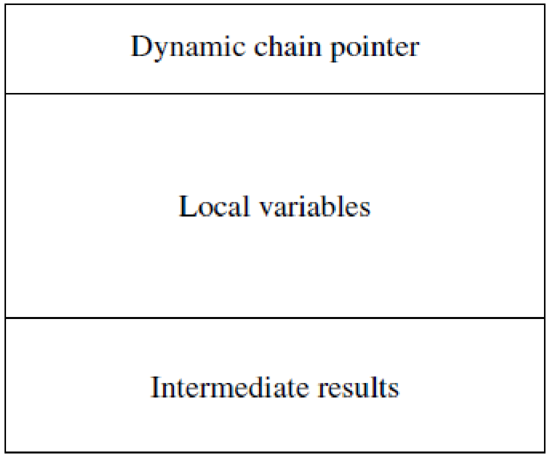
Dynamic MM using Stacks (cont.)

- Activation Records for *In-Line Blocks*
  - Intermediate Results

Example:

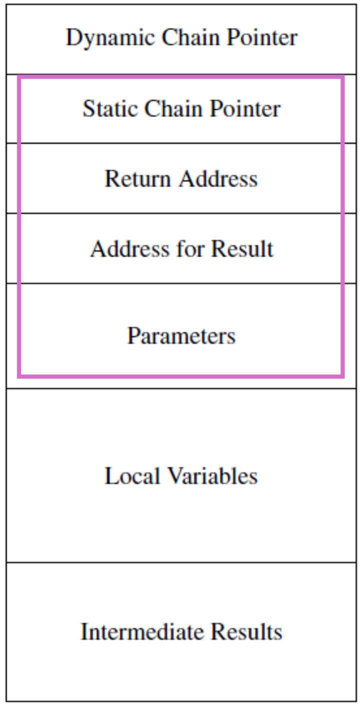
```
{int a =3;
 b= (a+x)/ (x+y);}
```





    - The need to store intermediate results on the stack depends on the compiler and on the architecture to which one is compiling. On many architectures they can be stored in registers.
  - Local variables
    - Those declared in the in-line blocks.
    - The size of memory space depend on the number & type of variables.
  - Dynamic link (= dynamic chain pointer = control link)
    - a pointer to the previous AR on the stack (or to the last AR created).

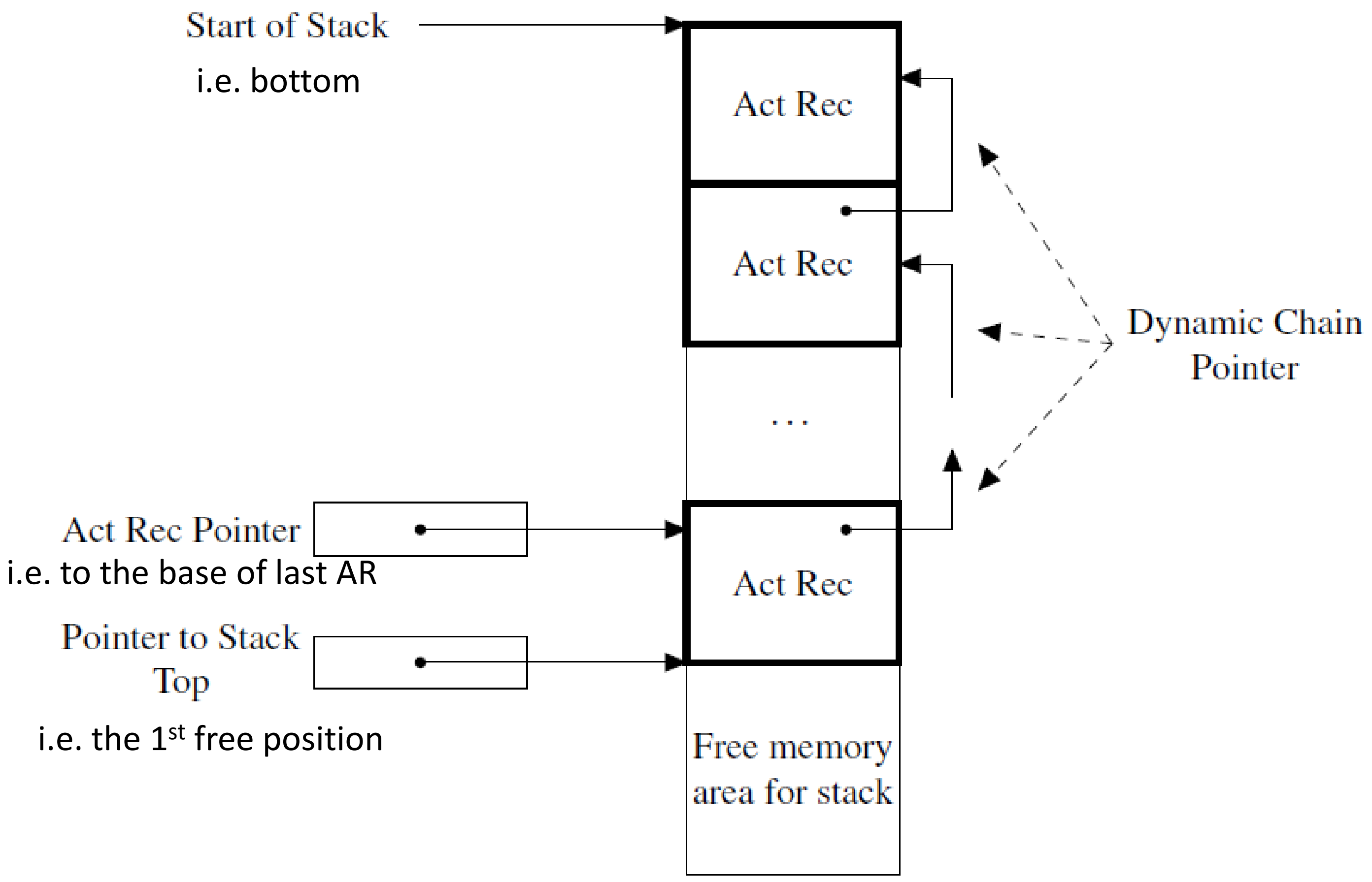
Dynamic MM using Stacks (cont.)



- Activation Records for *Procedures*
  - Intermediate Results,
  - Local variables
  - Dynamic link (= dynamic chain pointer)
  - Static link (=static chain pointer)
    - the information needed to implement the static scope rules
  - Return address
    - the address of the 1st instruction to execute after the call to the current procedure/function has terminated execution.
  - Returned result
    - Only in functions
  - Parameters
  - Note: The fields of AR vary from implementation to implementation.

Dynamic MM using Stacks (cont.)

- Activation Records (AR) for Procedures (cont.)



The stack of activation records

Dynamic MM using Stacks (cont.)

- Stack Management
  - As well as handling other control information, a piece of code called the calling sequence is inserted into the caller.
  - The calling sequence is executed in part immediately before the procedure call;
  - the remainder of this code is executed immediately after the termination of the call.
  - In the callee two pieces of code are added:
    - a prologue, to be executed immediately after entering the call, and
    - an epilogue, which is executed before the procedure ends execution.
  - These three code fragments manage the different operations needed to handle activation records and correctly implement a procedure call.
  - The exact division of what the caller and callee do depends on the compiler and on the specific implementation.

Dynamic MM using Stacks (cont.)

- Stack Management

  The calling sequence (in the caller) & prologue (of callee) must handle the following tasks:

  - Modification of Program Counter
  - Allocation of Stack space
  - Modification of AR pointer
  - Parameter passing
  - Register save
  - Execution of initialization code

Dynamic MM using Stacks (cont.)

- Stack Management

  When the callee terminates and *control returns to the caller*,

  the epilogue (in the callee) and the calling sequence (in the caller) perform the following operations:

  - Update of Program Counter
  - Value return: the values of parameters, those computed by a function must be stored in the caller's AR and accessible to the callee's AR.
  - Return of registers: restore the old value (of AR pointer)
  - Execution of finalization code: in some languages – before destroying any local objects.
  - Deallocation of stack space: the pointer to the stack is modified.

Dynamic MM using a Heap

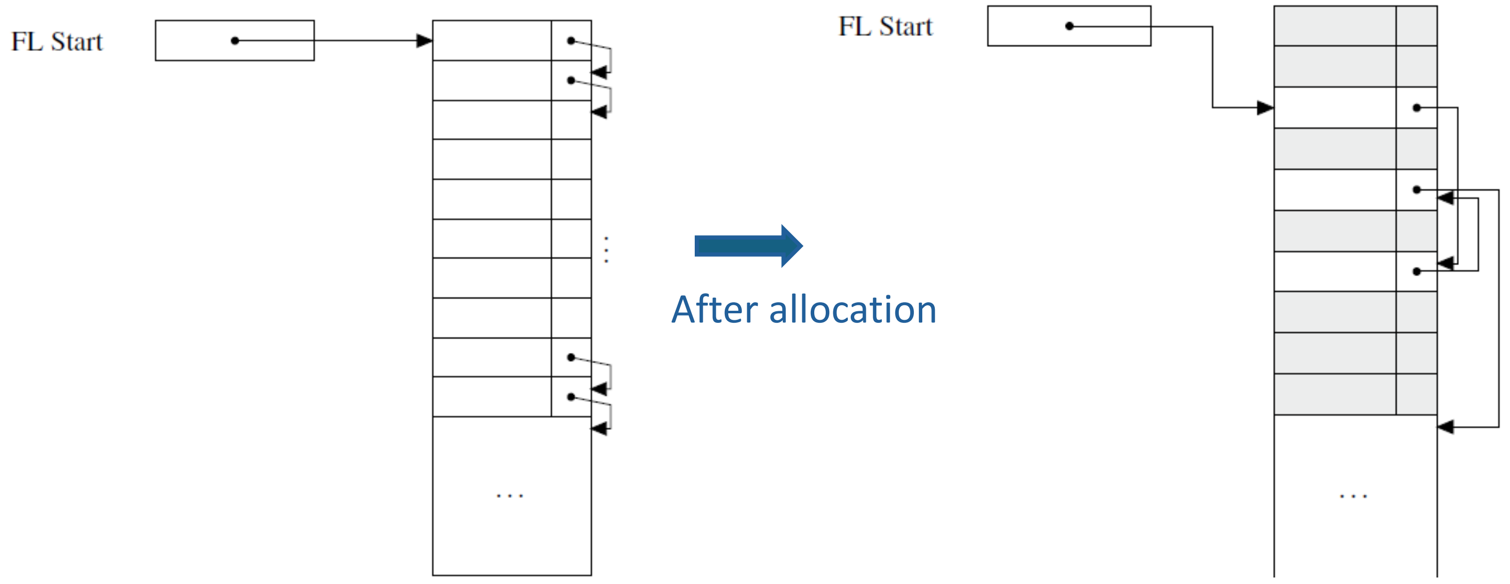
- Explicit commands for memory allocation in C and Pascal.

```
int  *p,  *q;  /* p,q   NULL pointers to integers */
p  =   malloc (sizeof (int));
          /* allocates the memory pointed to by p */
q  =   malloc (sizeof (int));
          /* allocates the memory pointed to by  q */
*p = 0;       /* dereferences and assigns */
*q = 1;       /* dereferences and assigns */
free(p);      /* deallocates the memory pointed to by p */
free(q);      /* deallocates the memory pointed to by q */
```

- Explicit memory (de)allocation at any time from a particular area of memory, called a *heap,* in which blocks of memory can be (de)allocated freely, not in LIFO order.
- Heap management:
  - the memory blocks are fixed-length
  - The memory blocks are variable length.

Dynamic MM using a Heap (cont.)

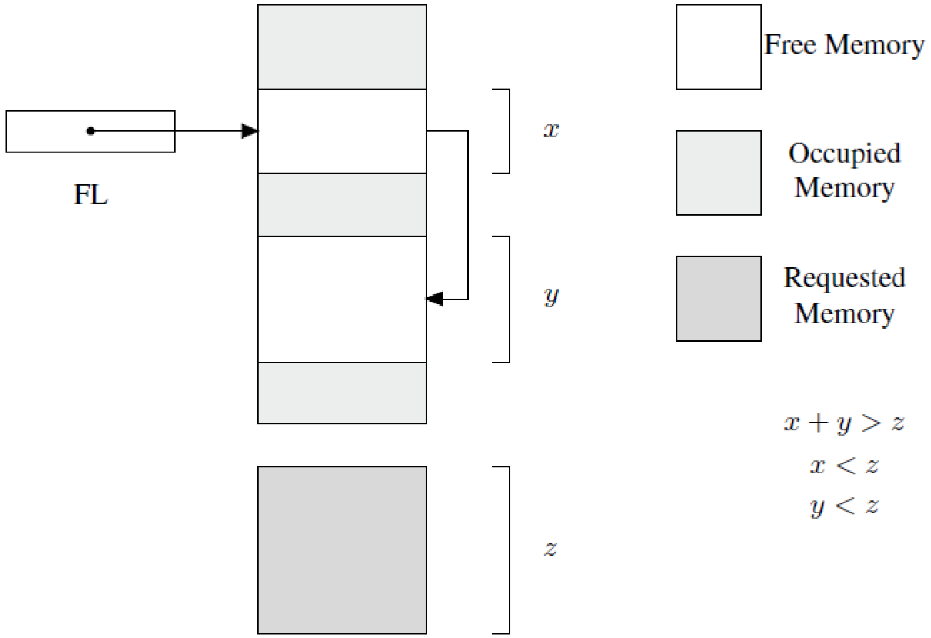
- Heap management with fixed-length block
  - the heap is divided into a certain number of blocks of small fixed length, linked into a list structure called the free list.
  - Allocation:
    - removal of the 1st block of the free list.
    - The pointer to this block is returned to the operation.
    - The pointer to the free list is updated, pointing to the next element.
  - Deallocation:
    - The freed block is linked to the *head* of the head of the free list.



Dynamic MM using a Heap (cont.)

- Heap management with variable-length block
  - The runtime allocation of variable-length memory space

    E.g.) store an array of variable dimensions
  - the aim of increasing memory occupation and execution speed for heap management operations
  - Goal: To avoid the memory fragmentation
    - Internal fragmentation: x, y > z. Then, x-z or y-z will be wasted.
    - External fragmentation: x + y > z but x < z and y < z; i.e. x and y are scattered.



Dynamic MM using a Heap (cont.)

- Heap management with variable-length block (cont.)
  - Goal: To avoid the memory fragmentation
  - The memory allocation techniques tend to *compact* free memory, merging contiguous free blocks to avoid external fragmentation. → Merging operations can increase the load imposed by the management methods and reduce efficiency.
  - Single Free-List
    - Initially, divide the entire heap into many small blocks.
    - Allocation of blocks from the beginning according to the request while collecting deallocated blocks on a free list.
    - How to *reuse deallocated memory* at the end of heap's space?
  - How to reuse deallocated memory in a single free-list?
    - Direct use of the free-list
    - Free memory compaction

Dynamic MM using a Heap (cont.)

- Heap management with variable-length block (cont.)
  - Direct Use of the Single Free-List
    - A free-list of variable-sized blocks is used.
    - In the allocation of size *n*, find a block of size *k* > *n*.
    - the unused part of the block of size *k*-*n* is inserted into the free list to form a new block.
    - Finding the sufficient sized block:
      - First Fit: Find the 1st block of sufficient size. Favors processing time.
      - Best Fit: Find a block whose size is the least of those of sufficient size.

        Favors memory occupation.
    - If the blocks are held in the increasing block size, the cost of insertion of a block into the free list increases.
    - When a deallocated block is returned to the free-list, it has to determine whether the *physically adjacent blocks are free* in order to reduce external fragmentation

      → they're compacted into a *single* block.

      -- a *partial* compaction because it compacts only adjacent blocks.

Dynamic MM using a Heap (cont.)

- Heap management with variable-length block (cont.)
  - Direct Use of the Single Free-List
  - Free memory compaction
    - When the end of the heap space is reached, all active blocks are moved to the end -- they cannot be returned to the free list, leaving all the free memory in *a single contiguous block*.
    - At this point, the heap pointer is updated, pointing to the start of the single block of free memory and allocation starts all over again.
    - For this technique to work, the blocks of allocated memory must be movable, something that is not always guaranteed (consider blocks whose addresses are stored in pointers on the stack).

Dynamic MM using a Heap (cont.)

- Heap management with variable-length block (cont.)
  - Multiple Free-List
    - To reduce the block allocation cost, different-sized *multiple free-lists* for blocks are used.
    - When a block of size *n* is requested, a block from the list of blocks of size $\geq n$ is chosen.
    - The size of the blocks can be static or dynamic.
    - Two management methods in the case of *dynamic sizes*:
      - Buddy system
        - The block size in the free-lists are powers of 2.
      - Fibonacci heap
        - The block size is Fibonacci number.
        - As the Fibonacci sequence grows more slowly than 2^n, it leads to less internal fragmentation.