Multiple stacks and queues

Two stacks

m[0], m[1], …, m[n-2], m[n-1]

bottommost → bottommost ←

stack 1 stack 2

More than two stacks (n)

memory is divided into n equal segments

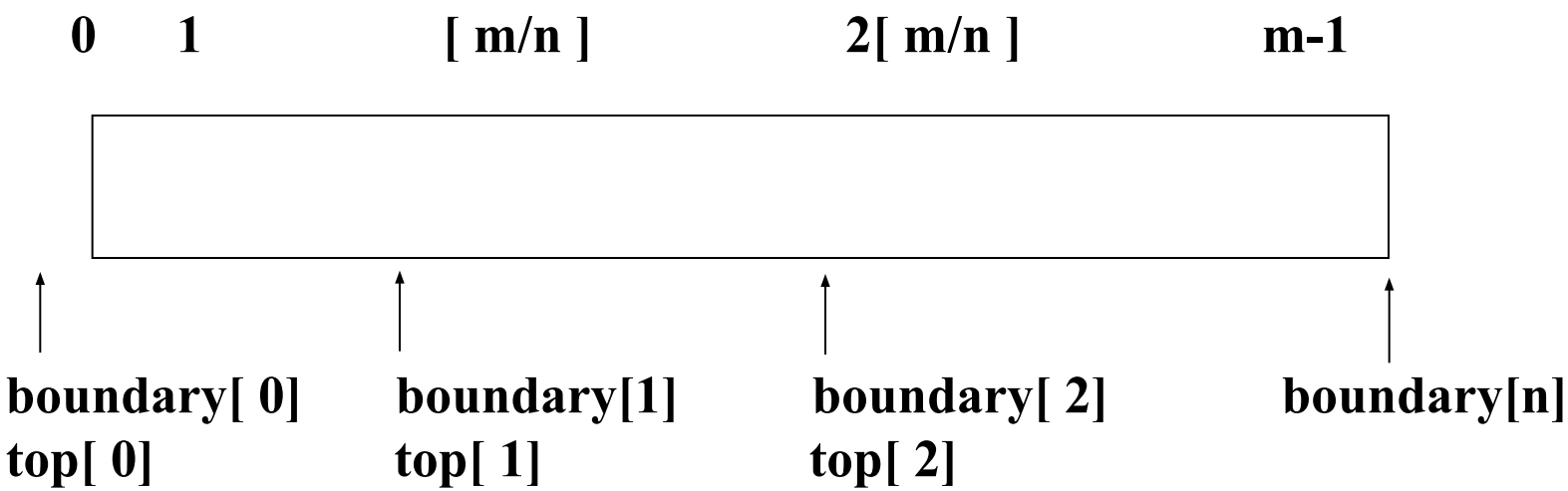
boundary[stack_no]

$0 \leq$ stack_no $<$ MAX_STACKS

top[stack_no]

$0 \leq$ stack_no $<$ MAX_STACKS

Initially, boundary[i]=top[i].



All stacks are empty and divided into roughly equal segments.

***Figure 3.18:** Initial configuration for *n* stacks in memory [m]. (p.129)

```
#define MEMORY_SIZE 100    /* size of memory */
#define MAX_STACK_SIZE 100
            /* max number of stacks plus 1 */
/* global memory declaration */
element memory[MEMORY_SIZE];
int top[MAX_STACKS];
int boundary[MAX_STACKS];
int n; /* number of stacks entered by the user */
```

*(p.128)

```
top[0] = boundary[0] = -1;
for (i = 1; i < n; i++)
  top[i] =boundary[i] =(MEMORY_SIZE/n)*i;
boundary[n] = MEMORY_SIZE-1;
```

*(p.129)

```
void add(int i, element item)
{
    /* add an item to the ith stack */
    if (top[i] == boundary [i+1])
        stack_full(i);     may have unused storage
        memory[++top[i]] = item;
}
```

***Program 3.12:** Add an item to the stack *stack-no* (p.129)

```
element delete(int i)
{
    /* remove top element from the ith stack */
    if (top[i] == boundary[i])
       return stack_empty(i);
    return memory[top[i]--];
}
```

***Program 3.13:** Delete an *item* from the stack *stack-no* (p.130)

Find j, stack_no < j < n (往右)

such that top[j] < boundary[j+1]

or, 0 ≤ j < stack_no (往左)

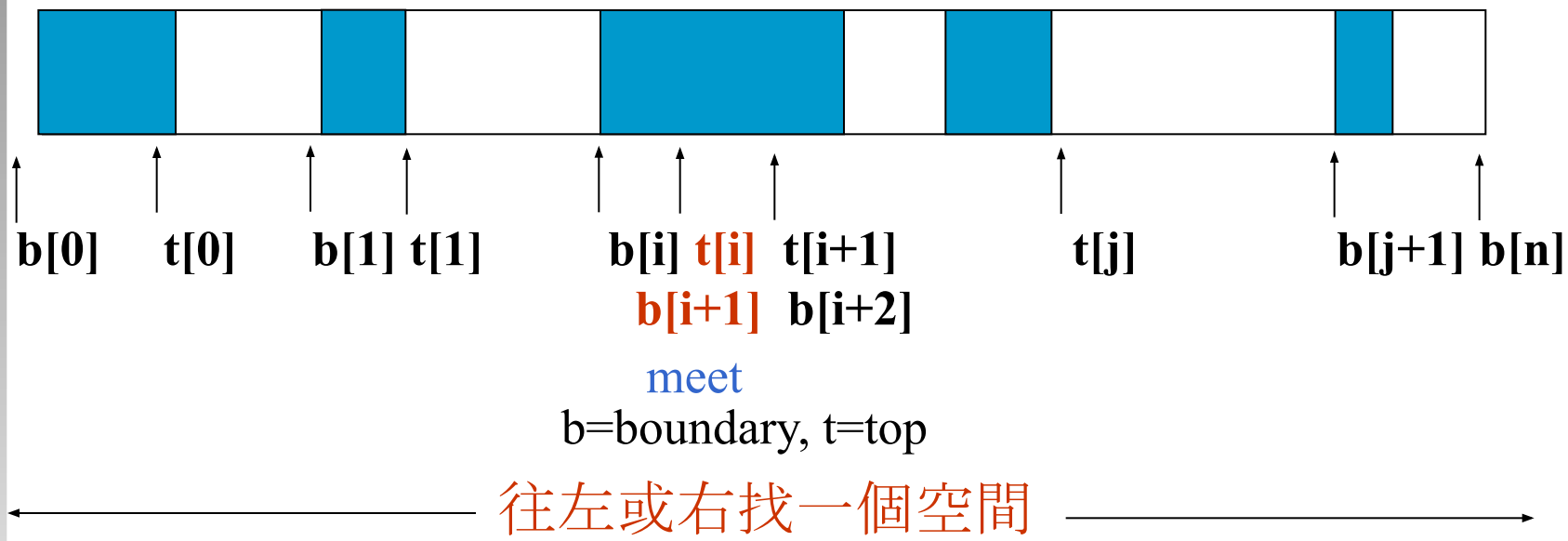


***Figure 3.19:** Configuration when stack i meets stack i+1, but the memory is not full (p.130)