Advanced C Programming & Lab

12. Dynamic Memory Allocation

Sejong University

Outline

1) **Dynamic Memory Allocation?**

2) Dynamic Memory Allocation: Procedures

3) Dynamic Memory Allocation: Examples

4) Dynamic Memory Allocation: Functions

1) Dynamic Memory Allocation?

- **Memory allocation**
  - Static memory allocation
  - Dynamic memory allocation

1) Dynamic Memory Allocation?

- **Memory allocation, so far (1/2)**
  - Pre-declare variables: variables, arrays, structures

➔ Pre-determine the size of the allocated memory

< Example >	Size of allocated memory

Variable Declaration	✓ char ch; // 1 byte
	✓ int num; // 4 bytes
	✓ float avg; // 4 bytes

Array Declaration	✓ int score[10]; // 40 bytes ✓ char address[50]; // 50 bytes

Structure Declaration	struct student {
	int id;
	float avg;
	};
	struct student st1; // 8 bytes

1) Dynamic Memory Allocation?

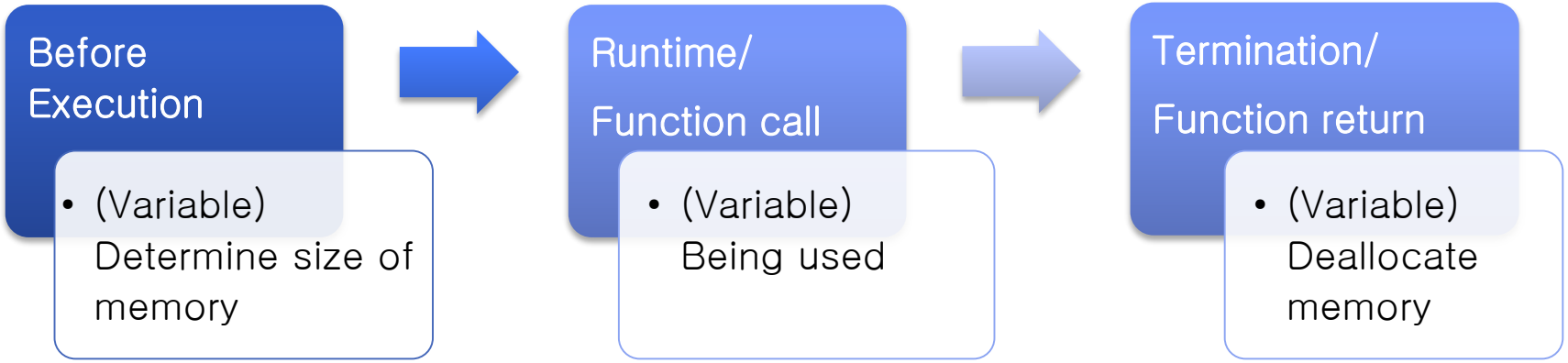
- **Memory allocation, so far (2/2)**
  - Size of an array is unknown?
    Declare an array of a sufficient size
  - Ex1) Launch a website. Want to store users' ID in an array.
    - ✓ Do not know how many users will register for the website
      → Assume quite many users will register
      → How many?
    - ✓ ex) int id[50000];

  ➔ if the size of a variable (or array) is unknown, allocate a sufficiently large size of memory

➔ **This is static memory allocation!**

1) Dynamic Memory Allocation?

- **Static memory allocation**
  - Allocation of memory at compile time
  - Before a program is executed, need to know how many variables and what kinds of variables we use
  - Before a program is executed, should declare all variables, arrays, and structures



< Static Memory Allocation >

1) Dynamic Memory Allocation?

- **Issues with Static Memory Allocation**
  - p.5 Ex1) int id[50000];
    - ✓ What if 50001th user tries to register,
      - ✓ While executing the program, cannot increase the size of the array
      - ✓ For a new user, need to change the array declaration and re-compile the program
    - ✓ What if only 1000 users register
      - ✓ Waste 49000 memory space

➔ **While executing a program, cannot increase or decrease the size of allocated memory**

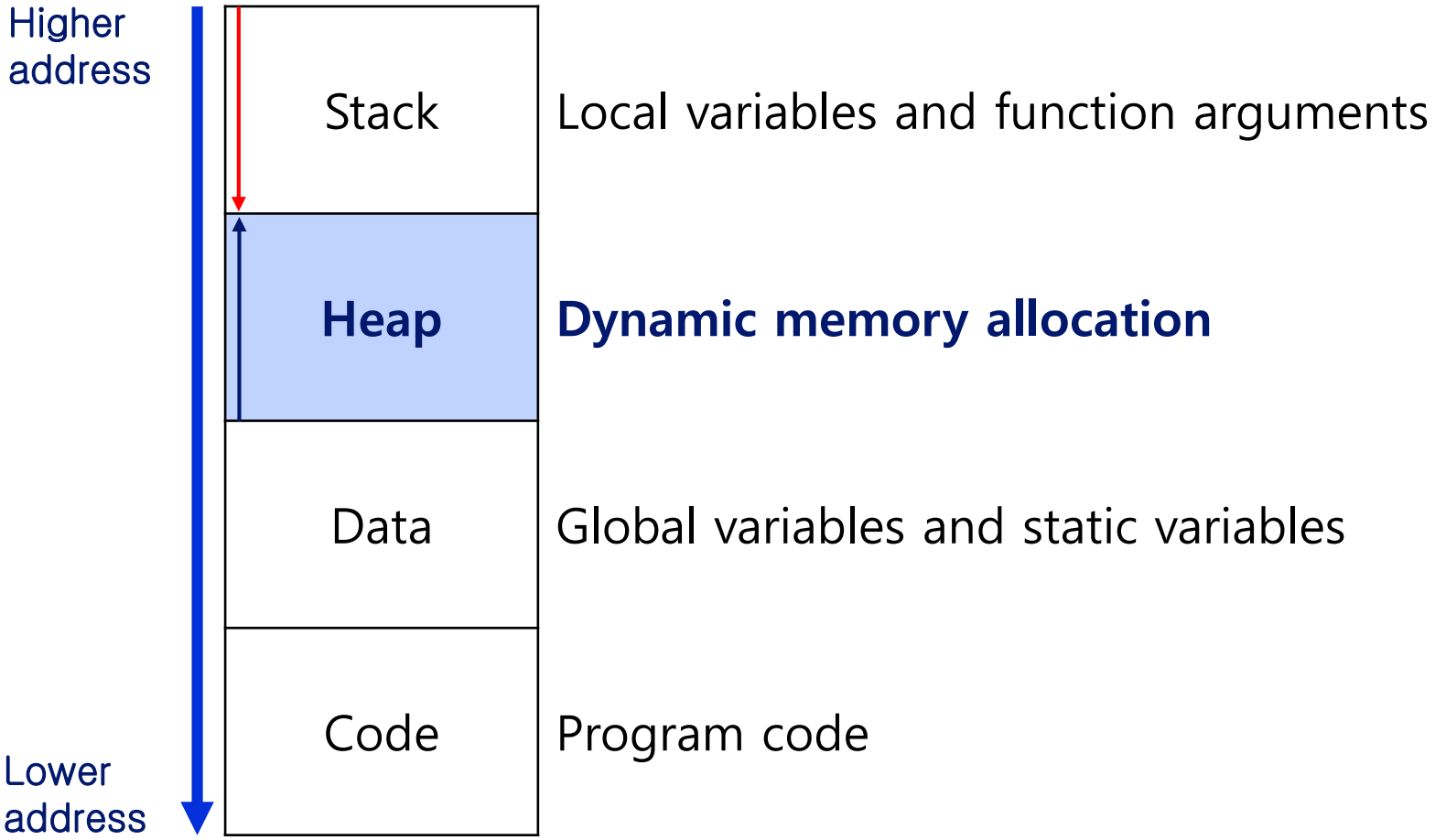
➔ **Do not know how much memory you need before executing a program, cannot efficiently manage memory**

1) Dynamic Memory Allocation?

- **Dynamic Memory Allocation**
  - While executing a program, allocate the requested amount of memory
    - ✓ No variable name
    - ✓ At compile time, the size of memory is unknown
  - Efficiently manage memory
  - Use heaps

1) Dynamic Memory Allocation?

- **Memory layout in C**



Outline

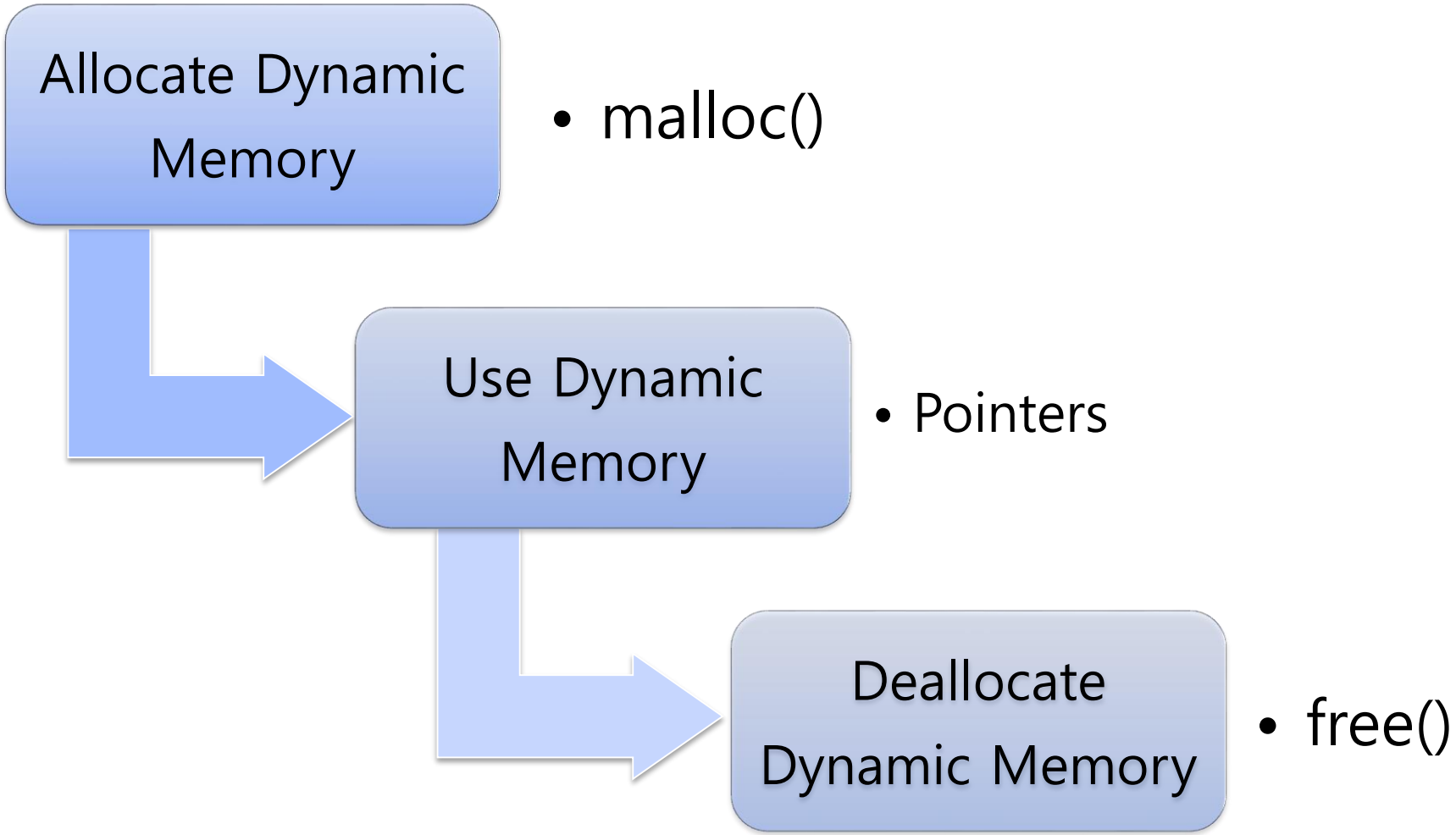
1) Dynamic Memory Allocation?

2) **Dynamic Memory Allocation: Procedures**

3) Dynamic Memory Allocation: Examples

4) Dynamic Memory Allocation: Functions

2) Dynamic Memory Allocation: Procedures



2) Dynamic Memory Allocation: Procedures

- **Must include <stdlib.h>**

- **Dynamic memory allocation: malloc() function**

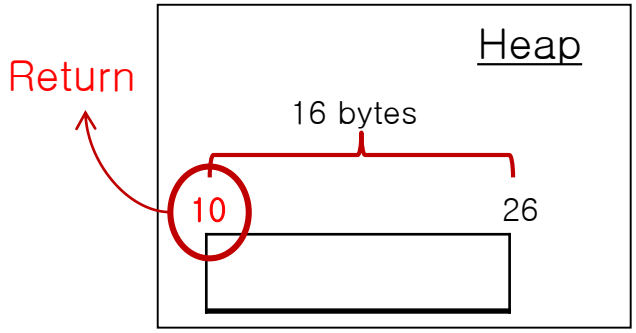
Function Prototytpe	**void * malloc(unsigned int size);**
Function Arguments	✓ Size of allocated memory ✓ In Byte ✓ Use sizeof() function
Function Return Type	✓ Return type: void type pointer ✓ Staring address of the allocated memory ✓ No space available → return NULL

2) Dynamic Memory Allocation: Procedures

- **Example: malloc()**
  - Allocate a space to store 4 integers…

Function call: malloc(4*sizeof(int));

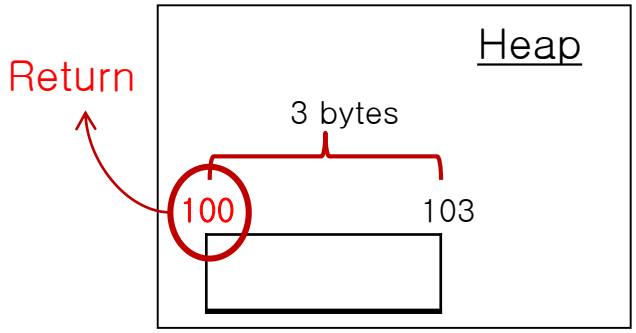
Function call: malloc(16);



  - For 3 characters…

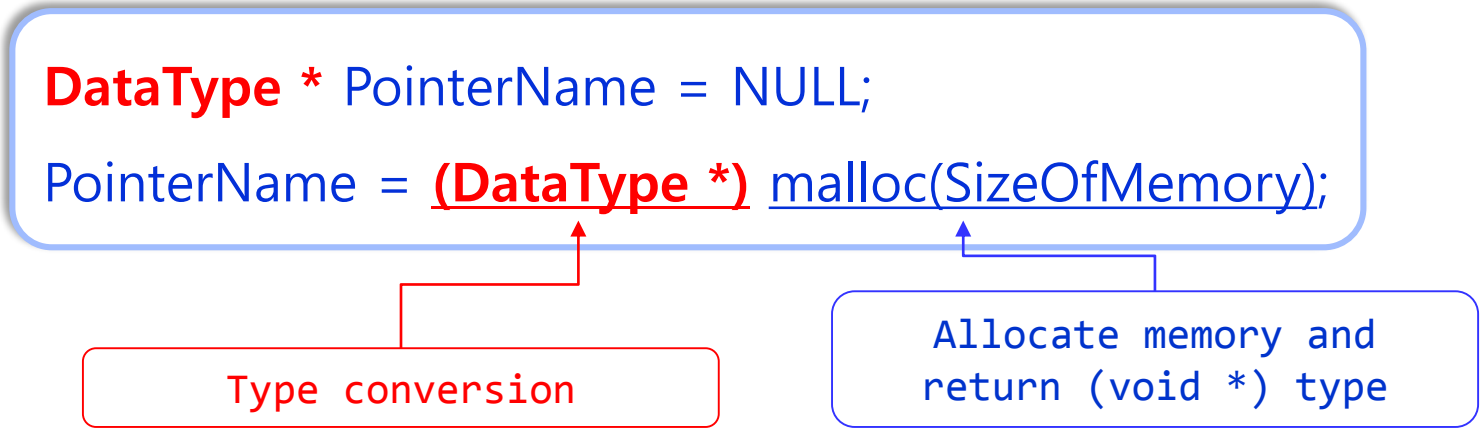
Function call: malloc(3*sizeof(char));

Function call: malloc(3);



2) Dynamic Memory Allocation: Procedures

- **Dynamic Memory Allocation: Pointers**
  - Use pointers to access the allocated memory in heap
  - Store the starting address
  - Use just like pointer variables or array names
  - Usage



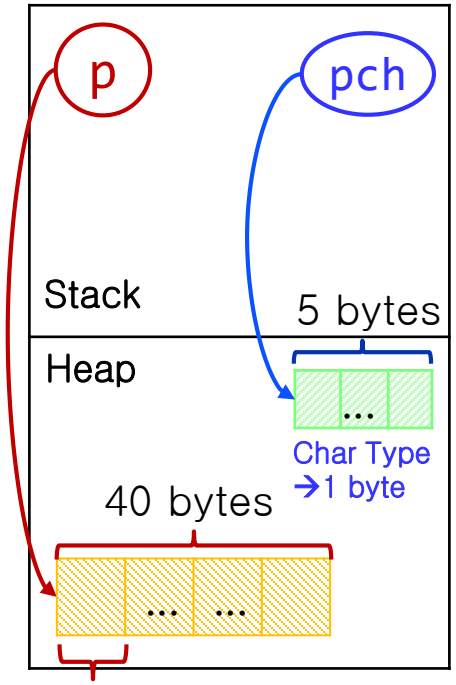
2) Dynamic Memory Allocation: Procedures

- **Example**
  - Ex1)

```
int *p = NULL;
p = (int *)malloc(10 * sizeof(int));
```

  - Ex2)

```
char *pch = NULL;
pch = (char *)malloc(5 * sizeof(char));
```



Type conversion: int type
→ 4bytes

2) Dynamic Memory Allocation: Procedures

- **Deallocate dynamic memory: free() function**
  - Do not automatically deallocate the dynamically allocated memory
  - Do not need the memory spaces, should deallocate them

Function Prototype	**void free(void * ptr);**
Function Arguments	A pointer to the memory space to be deallocated

2) Dynamic Memory Allocation: Procedures

- **Example: free()**
  - Ex1)

```
int *p = NULL;
p = (int *)malloc(4 * sizeof(int));
free(p);
```

  - Ex2)

```
char *pch = NULL;
pch = (char *)malloc(3 * sizeof(char));
free(pch);
```

2) Dynamic Memory Allocation: Procedures

- **Dynamic memory allocation: Practice**
  - Allocate memory space that can store one integer type number and deallocate the space
  - Allocate memory space that can store one float type number and deallocate the space
  - Allocate memory space that can store 15 double type number and deallocate the space
  - Allocate memory space that can store the following structure variable and deallocate the space

```
struct student{
   int id;
   char name[8];
   double grade;
};

struct student st1;
```

2) Dynamic Memory Allocation: Procedures

- **Caution 1**
  - Initialize the pointer to NULL

```
int *p = NULL;
p = (int *)malloc(10*sizeof(int));
```

  - After deallocation, initialize the pointer to NULL

→ Prevent dangling pointer

✓ Dangling pointer: the pointer to the deallocated memory space

```
int *p;
p = (int *)malloc(10*sizeof(int));
free(p);
p = NULL;
```

2) Dynamic Memory Allocation: Procedures

- **Caution 2**
  - After calling malloc(), need to check if it is successful
    - ✓ Return NULL pointer: couldn't allocate the consecutive memory space that is requested
  - Example

```
int *p = NULL;
p = (int *)malloc(10*sizeof(int));
if (p == NULL)
{
        printf("Not enough memory!");
        return -1;
}
```

2) Dynamic Memory Allocation: Procedures

- **Caution 3**
  - Before deallocation, check if the pointer points to NULL
  - Generally, use if statement

```
int *p = NULL;
p = (int *)malloc(10*sizeof(int));
if (p != NULL)
    free(p);
```

2) Dynamic Memory Allocation: Procedures

- **Statid vs. Dynamic memory allocation**

	Static Memory Allocation	**Dynamic Memory Allocation**
When to determine the size of memory	Writing a program	While executing a program
When to allocate and deallocate memory	OS automatically allocate and deallocate	Programmer allocates and deallocates
Memory size	Fixed	Changeable
Memory leak	No	Possible
Efficiency	X	o

Outline

1) Dynamic Memory Allocation?

2) Dynamic Memory Allocation: Procedures

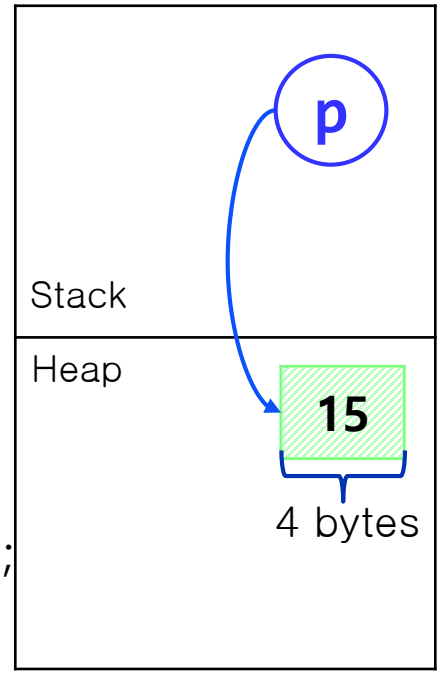
3) **Dynamic Memory Allocation: Examples**

4) Dynamic Memory Allocation: Functions

3) Dynamic Memory Allocation: Examples

- **Example1: Use dynamic memory allocation**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int *p = NULL;
        p = (int *)malloc(sizeof(int));
        if (p == NULL)
        {
                printf("Not enough memory!");
                return -1;
        }
        *p = 15;
        printf("dynamic memory allocation (int): %d", *p);
        free(p);
        return 0;
}
```

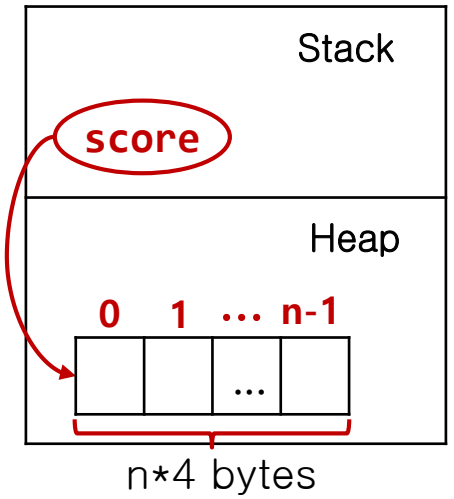


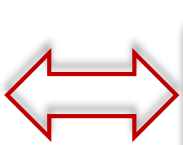
3) Dynamic Memory Allocation: Examples

- **Example2: Use dynamic memory allocation. Use a pointer as if it is an array name**
  - Problem: Read one student's test scores, compute average score, and print the average.
    - ✓ While executing the program, read n (# of subjects) from a user
    - ✓ Allocate the memory space that can store n integers
    - ✓ Store the starting address of the allocated space in a pointer score
    - ✓ Use score as if it is an array name (score[0]~score[n-1])

3) Dynamic Memory Allocation: Examples

```
int n, i, sum = 0;
int *score = NULL;
scanf("%d", &n);
score = (int *) malloc(n*sizeof(int));
if (score == NULL)
{
      printf("Not enough memory!");
      return -1;
 }
 for (i=0; i<n; i++)
 {
      scanf("%d", &score[i]);
      sum += score[i];
  }
 printf("%.1f", (double)sum/n);
 free(score);
```





```
scanf("%d", score+i);
sum += *(score+i);
```

3) Dynamic Memory Allocation: Examples

- **Example3: Process character strings (1/5)**
  - Use dynamic memory allocation to process a number of character strings of differing sizes
  - How to process them?
    ① Declare a sufficiently large character array to receive a character string
    ② Store a character string in a character array ①
    ③ Compute the size of a character string ②, dynamically allocate memory space (including NULL)
    ④ Copy the character string ① to the allocated space

3) Dynamic Memory Allocation: Examples

- **Example3: Process character strings (2/5)**
  - Problem: Write a program that stores n book titles (receive n from a user)
    - ✓ Header files: <stdio.h>, <stdlib.h>, <string.h>
    - ✓ Structure:

```
typedef struct book_title
{
        char *title;
} BINFO;
```

3) Dynamic Memory Allocation: Examples

- **Example3: Process character strings (3/5)**
  - main() : (1/3)

```
BINFO *bp = NULL;
int n, i;
char temp[100];          //① a character array to receive a character string

scanf("%d₩n", &n);                    // Use '₩n' or getchar()

bp = (BINFO *)malloc(n*sizeof(BINFO));
if (bp == NULL)
{
        printf("Not enough memory!");
        return -1;
}
```

3) Dynamic Memory Allocation: Examples

- **Example3: Process character strings (4/5)**
  - main() : (2/3)

```
for (i=0; i<n; i++)  {
    gets(temp);                    //② Receive a character string and put it in temp

    bp[i].title = (char*)malloc((strlen(temp)+1)*sizeof(char));
    // ③ calculate the size of a character string in temp,
    // allocate memory space (including NULL)

    if (bp[i].title == NULL)
    {
        printf("Not enough memory!");
        return -1;
    }
    strcpy(bp[i].title, temp);
    //④ Copy the character sting in temp to the memory space
}
```

3) Dynamic Memory Allocation: Examples

- **Example3: Process character strings (5/5)**
  - main() : (3/3)

```
for (i=0; i<n; i++)
{
        printf("%s ₩n", bp[i].title);
}

for (i=0; i<n; i++)
        free(bp[i].title);

free(bp);
```

3) Dynamic Memory Allocation: Examples

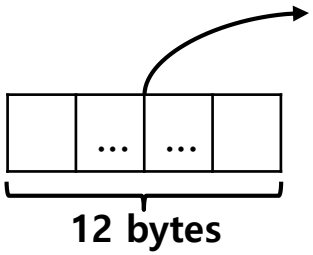
- **Example4: Allocate 2-dimensional array (1/4)**
  - Want to dynamically allocate memory space for char array[3][4]

    ➔ Ex) char *pch = (char *) malloc(3*4*sizeof(char));

    ✓ Store 12 characters, 1-dimensional character array!

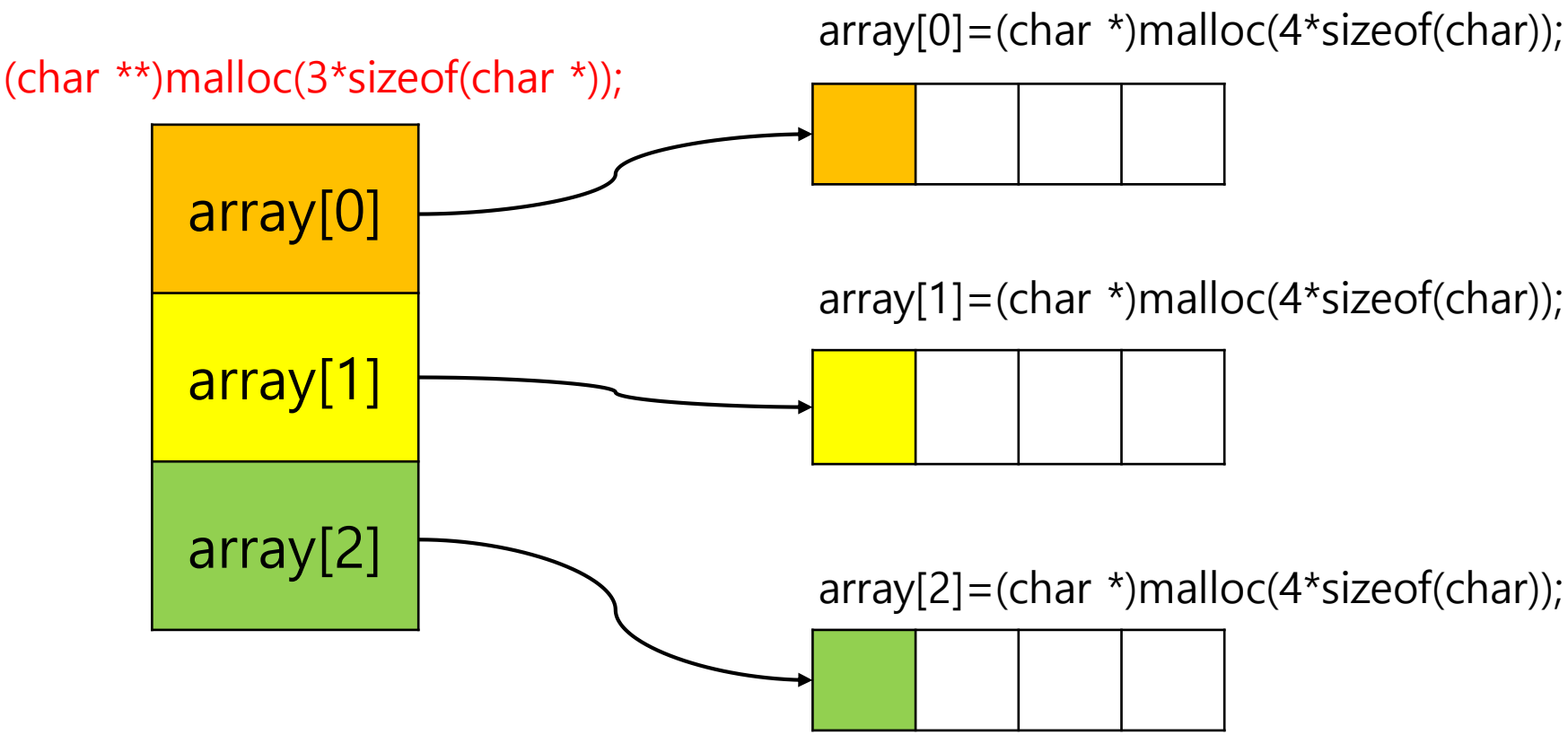
    ✓ However, 3 character strings, of each comprises 4 characters

    ✓ Use **Pointers**!



3) Dynamic Memory Allocation: Examples

- **Example4: Allocate 2-dimensional array (2/4)**
  - Dynamic memory allocation: char array[3][4]



3) Dynamic Memory Allocation: Examples

- **Example4: Allocate 2-dimensional array (3/4)**
  - Dynamic memory allocation for char array[3][4], print character strings

```
int i;
char **pch;
pch = (char **)malloc(3*sizeof(char *));
for (i=0; i<3; i++)
        pch[i] = (char *)malloc(4*sizeof(char));

strcpy(pch[0], “aaa”);
strcpy(pch[1], “bbb”);
strcpy(pch[2], “ccc”);

for (i=0; i<3; i++)
        puts(pch[i]);

for (i=0; i<3; i++)
        free(pch[i]);

free(pch);
```

3) Dynamic Memory Allocation: Examples

- **Example4: Allocate 2-dimensional array (4/4)**
  - int 2-dimensional array array[row][col]

```
int **pa;
pa = (int **)malloc(row * sizeof(int *));
for (i=0; i<row; i++)
        pa[i] = (int *)malloc(col * sizeof(int));
```

  - Deallocate 2-dimensional array[row][col]

```
for(i=0; i<row; i++)
        free(pa[i]);
free(pa);
```

Outline

1) Dynamic Memory Allocation?
2) Dynamic Memory Allocation: Procedures
3) Dynamic Memory Allocation: Examples
4) **Dynamic Memory Allocation: Functions**

4) Dynamic Memory Allocation: Functions

- **calloc() function**

Function Prototype	**void * calloc(unsigned int num, unsigned int size);**	
Function Arguments	num	Number of elements to allocate
	size	Size of each element (Byte)
Return Type	✓ Return type: void type pointer ✓ Allocate (num*size) bytes, initialize to 0, return the starting address ✓ Cannot allocate the requested space → NULL	

- **Ex)**

```
int *p = NULL;
p = (int *)calloc(5, sizeof(int));
```

4) Dynamic Memory Allocation: Functions

- **realloc() function**

Function Prototype	**void * realloc(void *ptr, unsigned int size);**	
Function Argument	ptr	Pointer to the memory space to be reallocated
	size	New size (Bytes) – total size
Return Type	✓ Return type: void type pointer ✓ Change the size of memory space pointed by ptr to size Bytes, return the starting address ✓ Cannot allocate the requested space → NULL	

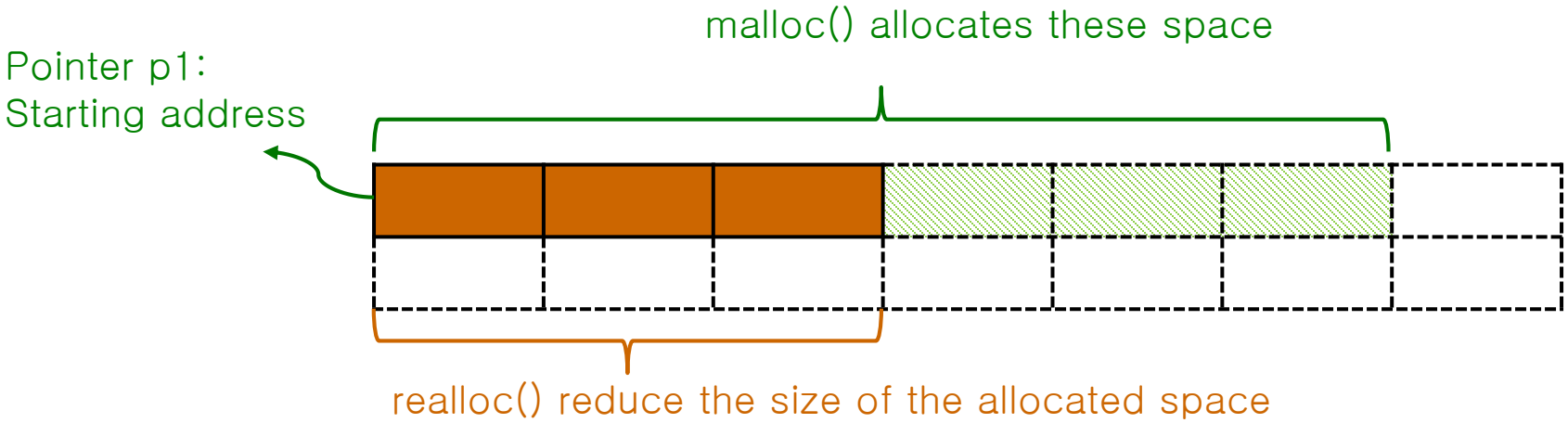
4) Dynamic Memory Allocation: Functions

- **realloc() Function: 3 cases**

  1) Reduce the size of memory space

     ✓ Cut out the existing contents

     ✓ Return the same starting address

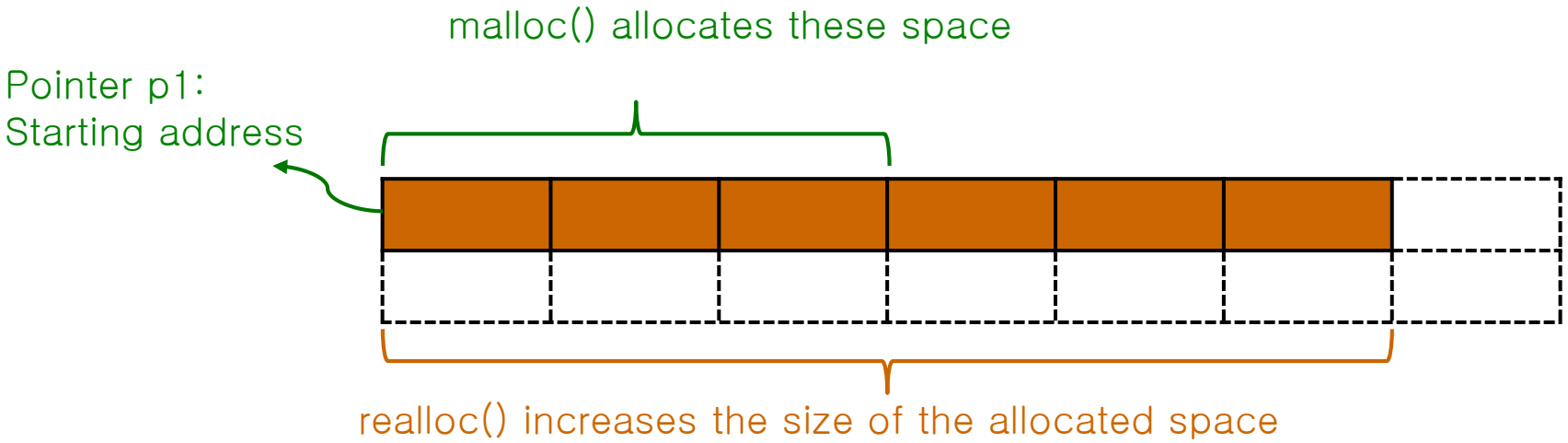


4) Dynamic Memory Allocation: Functions

- **realloc() Function: 3 cases**

  2) Increase the size of memory space

  ✓ Return the same starting address

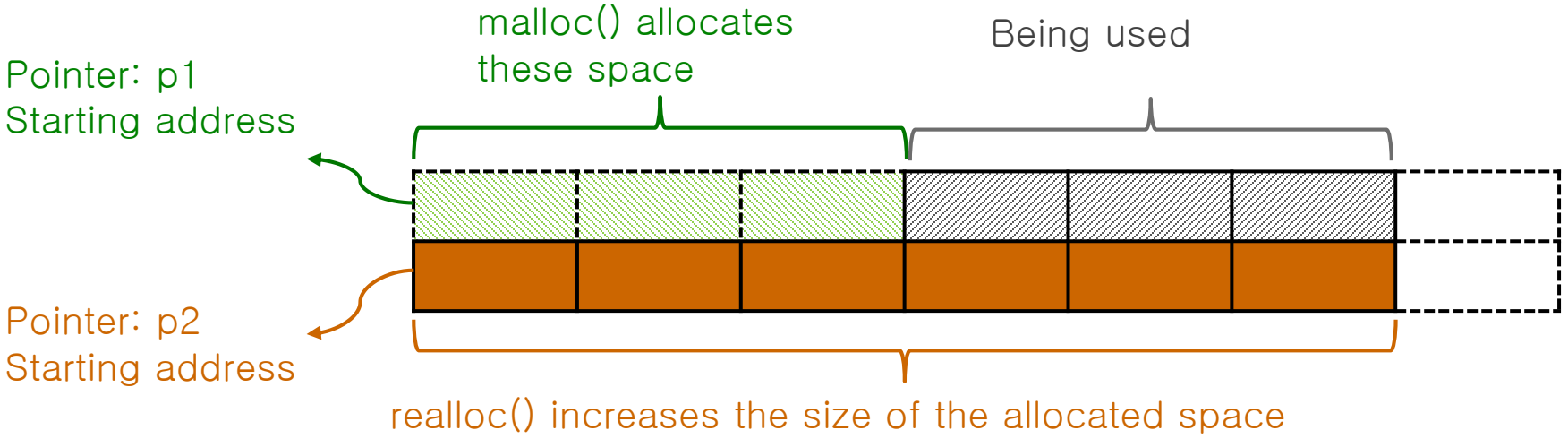


4) Dynamic Memory Allocation: Functions

- **realloc() Function: 3 cases**

  3) Unable to increase the size of memory space

  - ✓ Allocate new space, copy the contents to the new space
  - ✓ Previous address/space (p1) is automatically deallocated
  - ✓ Return new address (p2)



4) Dynamic Memory Allocation: Functions

- **Example: calloc() and realloc() – in main()**

```
int *p;
int size = 3;
int num, i, index = 0;

p = (int *)calloc(size, sizeof(int));
if (p == NULL) {
   printf("Not enough memory!");
   return -1;
}
for (i=0; i<10; i++){
    scanf("%d", &num);
    if(index < size){
          p[index++]=num;
    } else{
          size += 3;
          p = (int *)realloc(p, size*sizeof(int));
          p[index++]=num;
          }
}
free(p);
```

4) Dynamic Memory Allocation: Functions

- **malloc() vs. calloc() vs. realloc()**

	property	**common**
malloc()	Memory allocation	free(): deallocate the memory space
calloc()	Memory allocation + initialization	
realloc()	Change the pre-allocated memory space	