Dynamic Memory Allocation

“Control over the life-time of a variable”

Prerequisite: Pointer

Find more contents at
https://sites.google.com/view/cse105june18/home

Md. Saidul Hoque Anik
onix.hoque.mist@gmail.com

Scope of a Variable

Three places

```c
#include <stdio.h>

int g;                      //g -> global variable

void function(int n)        //n -> formal param
{

}

int main()
{
    int m = 10;                 //m -> local variable
}
```

Scope of a Variable

Scope of n and m

```c
#include <stdio.h>

int g;                          //g -> global variable

void function(int n)            //n -> formal param
{

}

int main()
{
    int m = 10;                     //m -> local variable
}
```

Scope of a Variable

Scope of g

```
#include <stdio.h>

int g;                          //g -> global variable

void function(int n)            //n -> formal param
{

}

int main()
{
    int m = 10;                     //m -> local variable
}
```

Lifetime of a local variable

Local variables are **destroyed** when they go **out-of-scope**

```c
#include <stdio.h>
void fn()
{
    int m = 0;
    m = m + 1;
    printf("%d\n", m);
}   //m is now destroyed

int main()
{
    fn();   //1
    fn();   //1
}
```

Reference to Local

Pointer referring to expired local variable

```c
#include <stdio.h>

int main()
{
    int n = 10;
    int * p;



}
```

Reference to Local

Pointer referring to expired local variable

```c
#include <stdio.h>

int main()
{
    int n = 10;
    int * p;

    if (n == 10)
    {
        int m = 20;      //m -> local variable
        p = &m;
    }



}
```

Reference to Local

Pointer referring to expired local variable (Dangling Pointer)

```c
#include <stdio.h>

int main()
{
    int n = 10;
    int * p;

    if (n == 10)
    {
        int m = 20;      //m -> local variable
        p = &m;
    }
    //m is now destroyed
    printf("%d", *p);

}
```

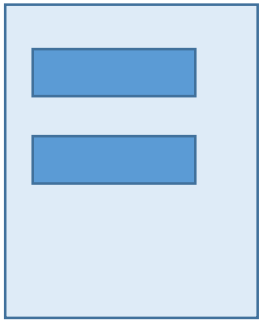
Reference to Local

Pointer referring to expired local variable (Dangling Pointer)

```c
#include <stdio.h>

int main()
{
    int n = 10;
    int * p;

    if (n == 10)
    {
        int m = 20;      //m -> local variable
        p = &m;
    }
    //m is now destroyed
    printf("%d", *p);    //Undefined behaviour
                         //can be 20, or show garbage, or crash
}
```

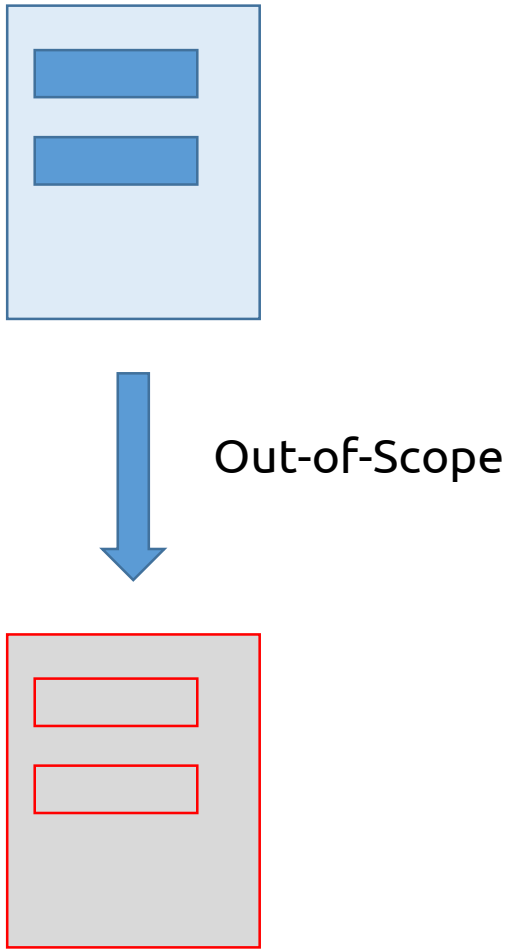
Dynamic Memory Allocation

The concept



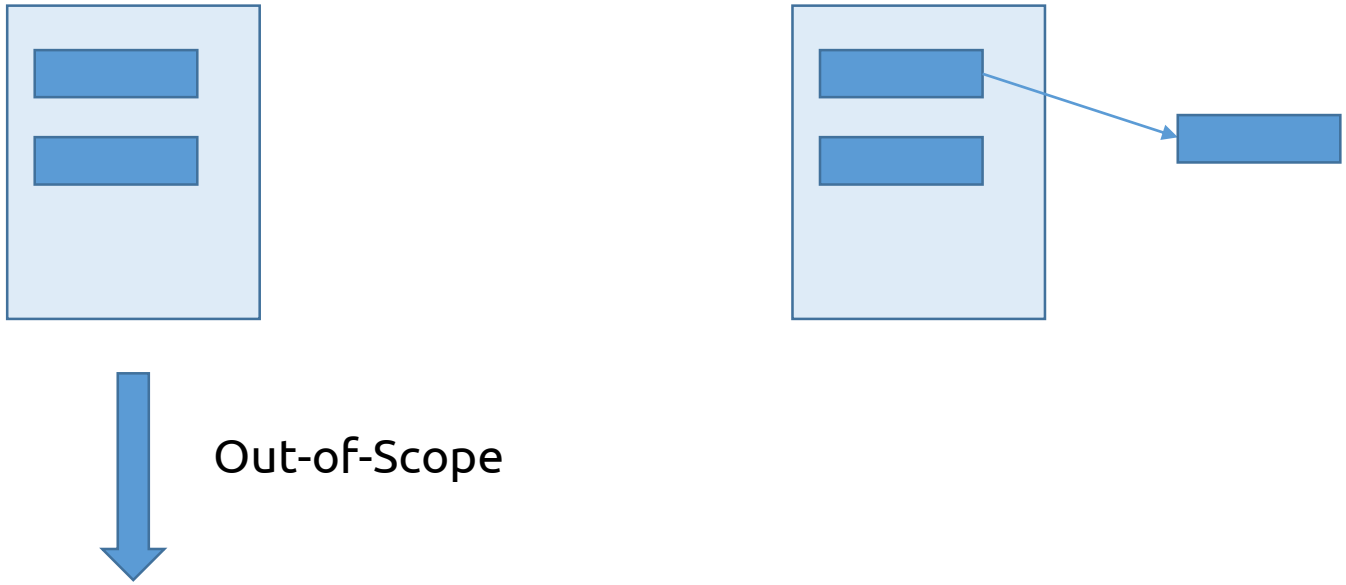
Dynamic Memory Allocation

The concept



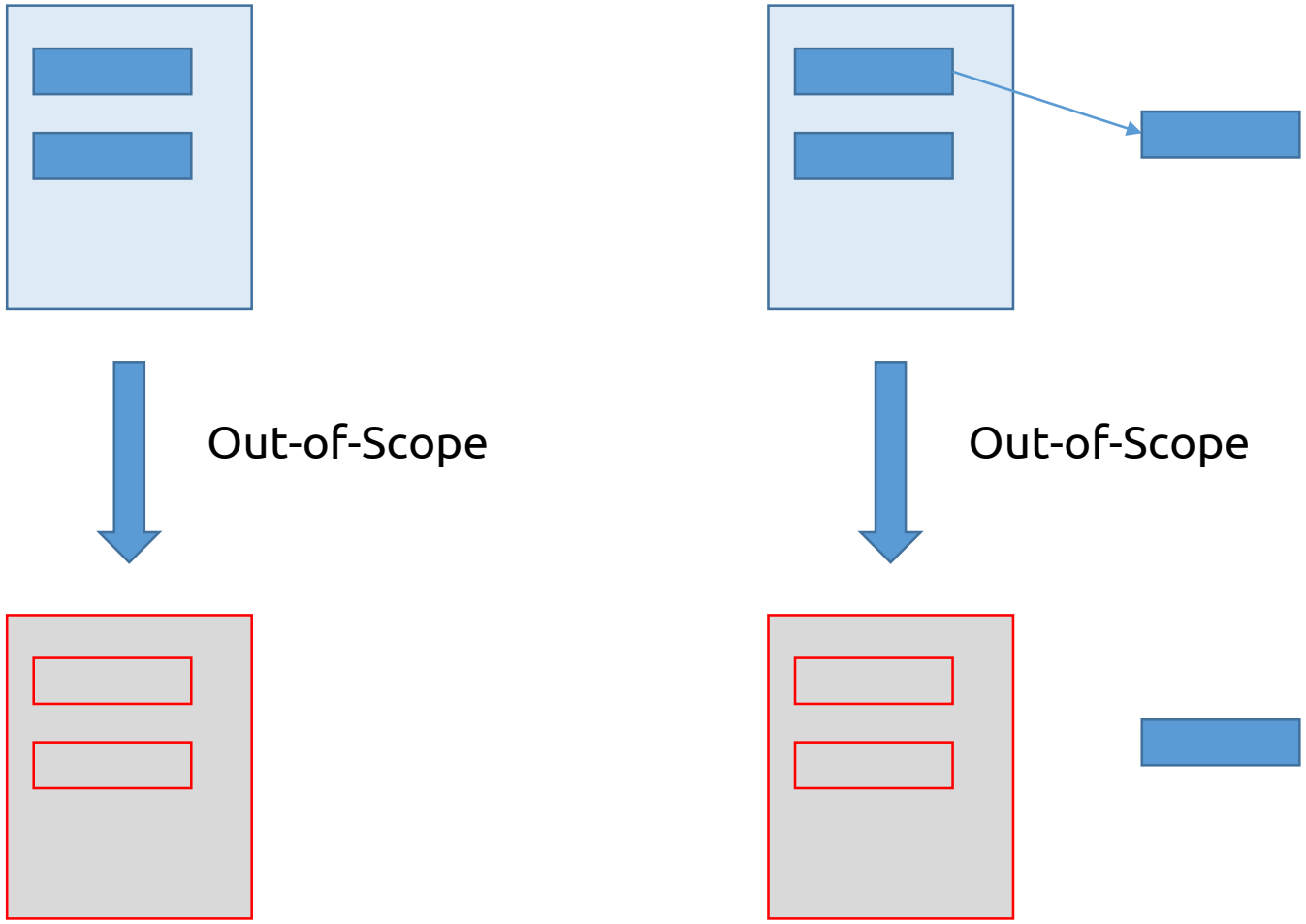
Dynamic Memory Allocation

The concept



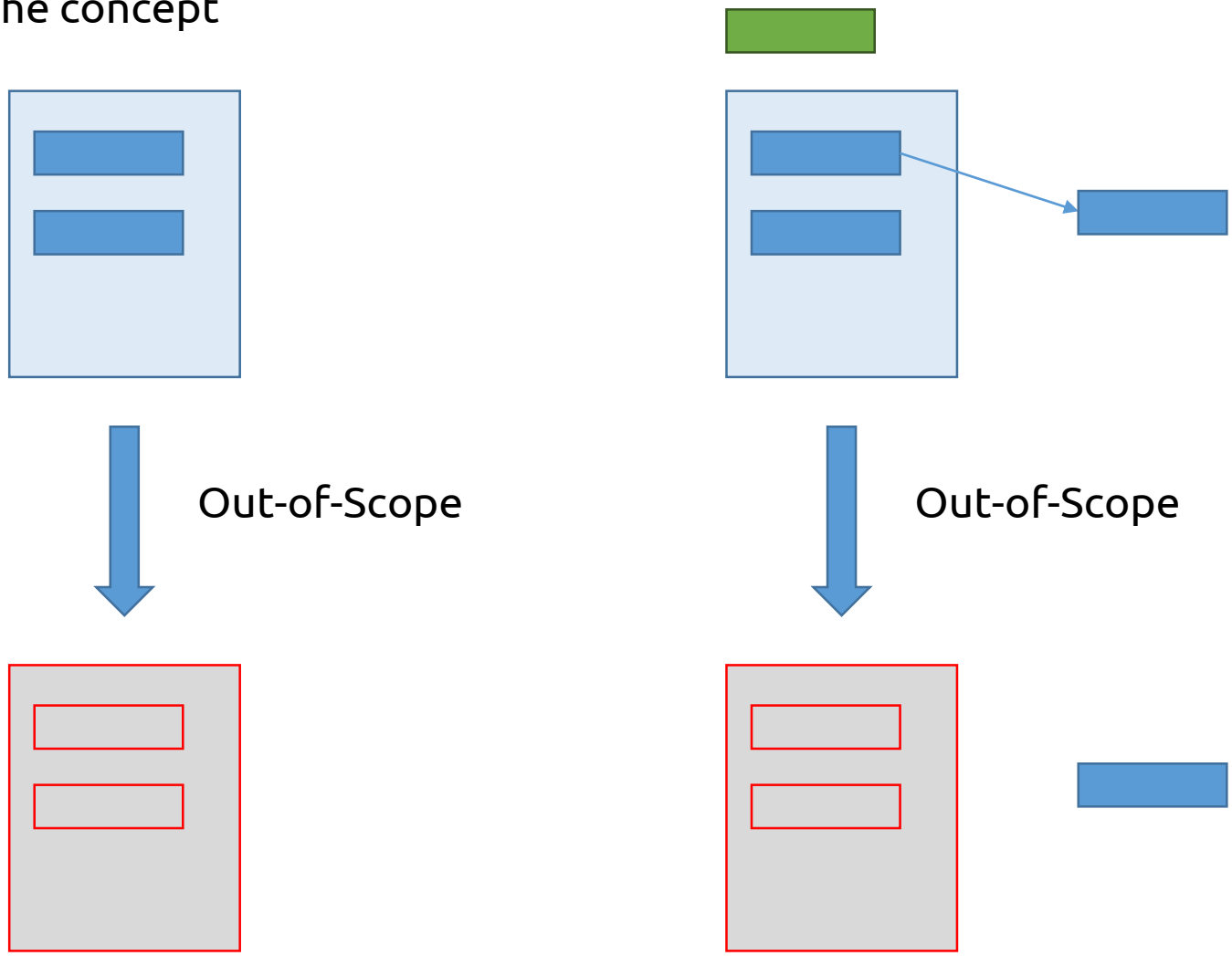
Dynamic Memory Allocation

The concept



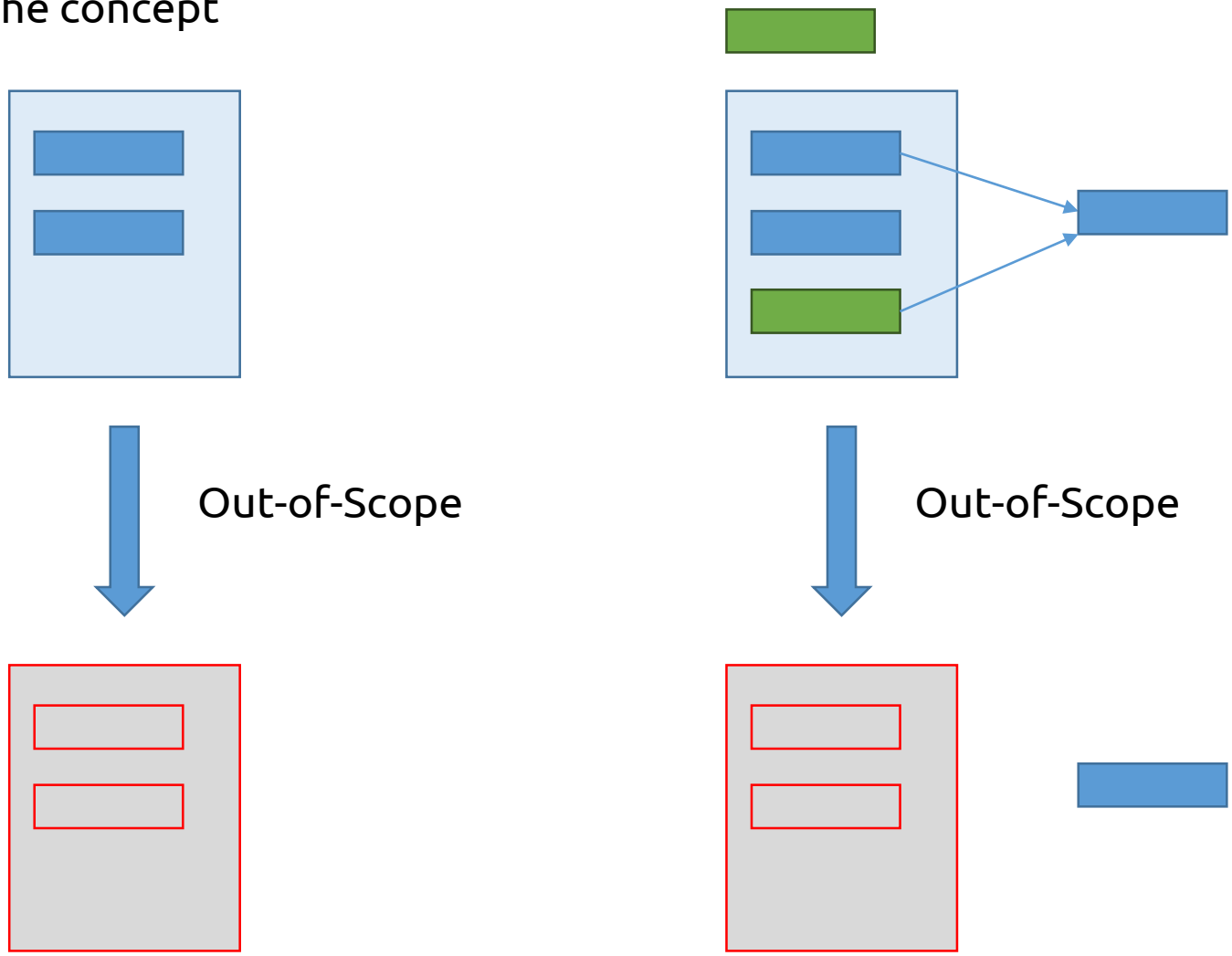
Dynamic Memory Allocation

The concept



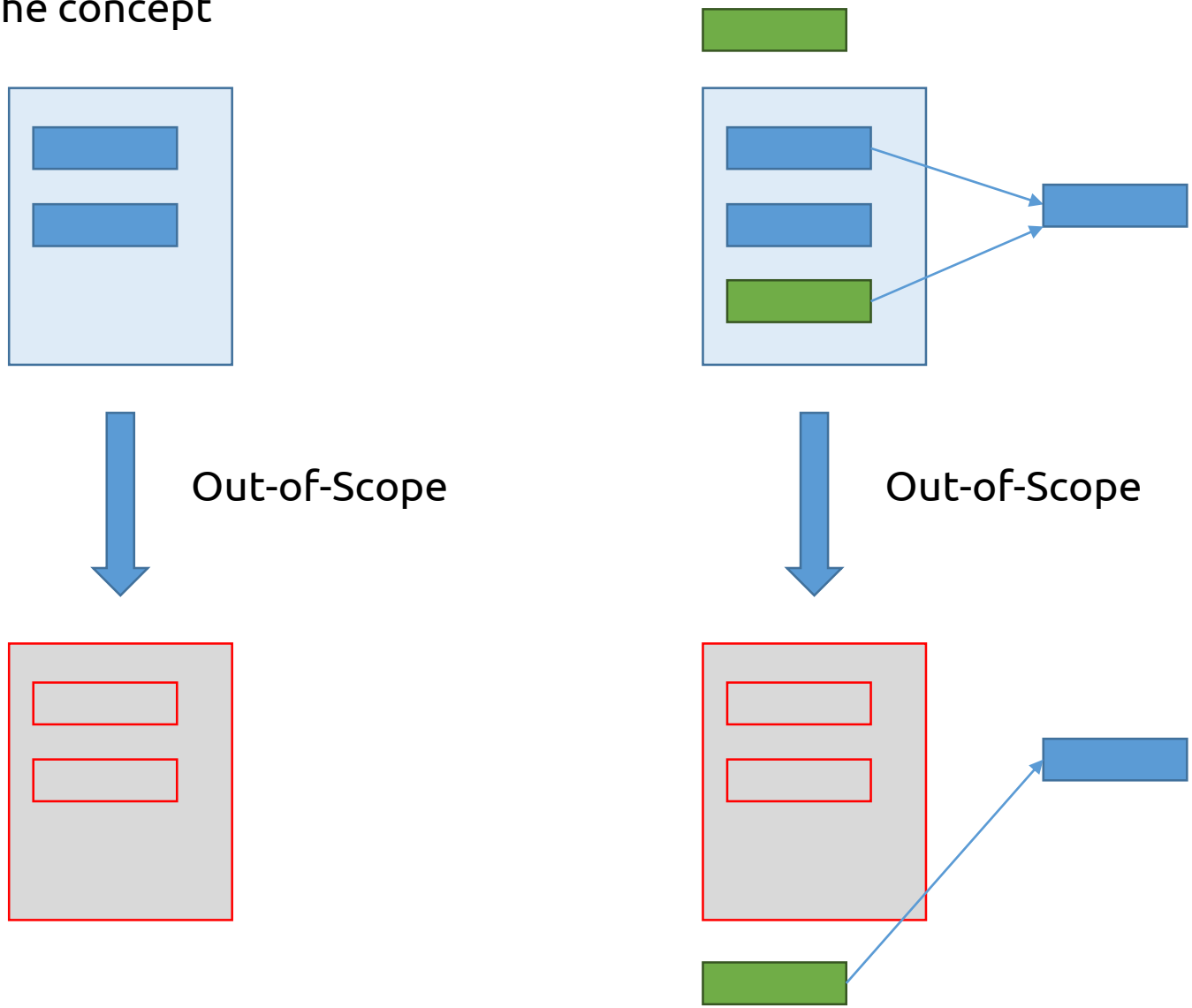
Dynamic Memory Allocation

The concept



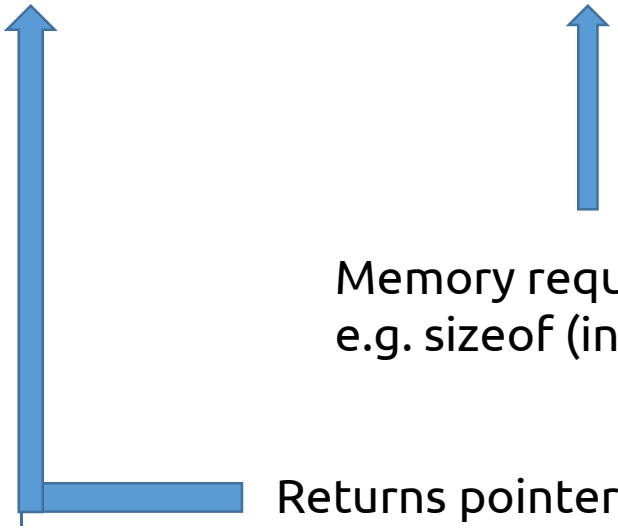
Dynamic Memory Allocation

The concept



The `malloc` function

```
void * malloc (size_t size);
```



Memory required in byte
e.g. sizeof (int)

Returns pointer to the beginning of the block

According to the 1999 ISO C standard (C99), size_t is an unsigned integer type of at least 16 bit. This type is used to represent the size of an object.

- https://en.wikipedia.org/wiki/C_data_types#stddef.h
- https://stackoverflow.com/questions/2550774/what-is-size-t-in-c

The malloc function

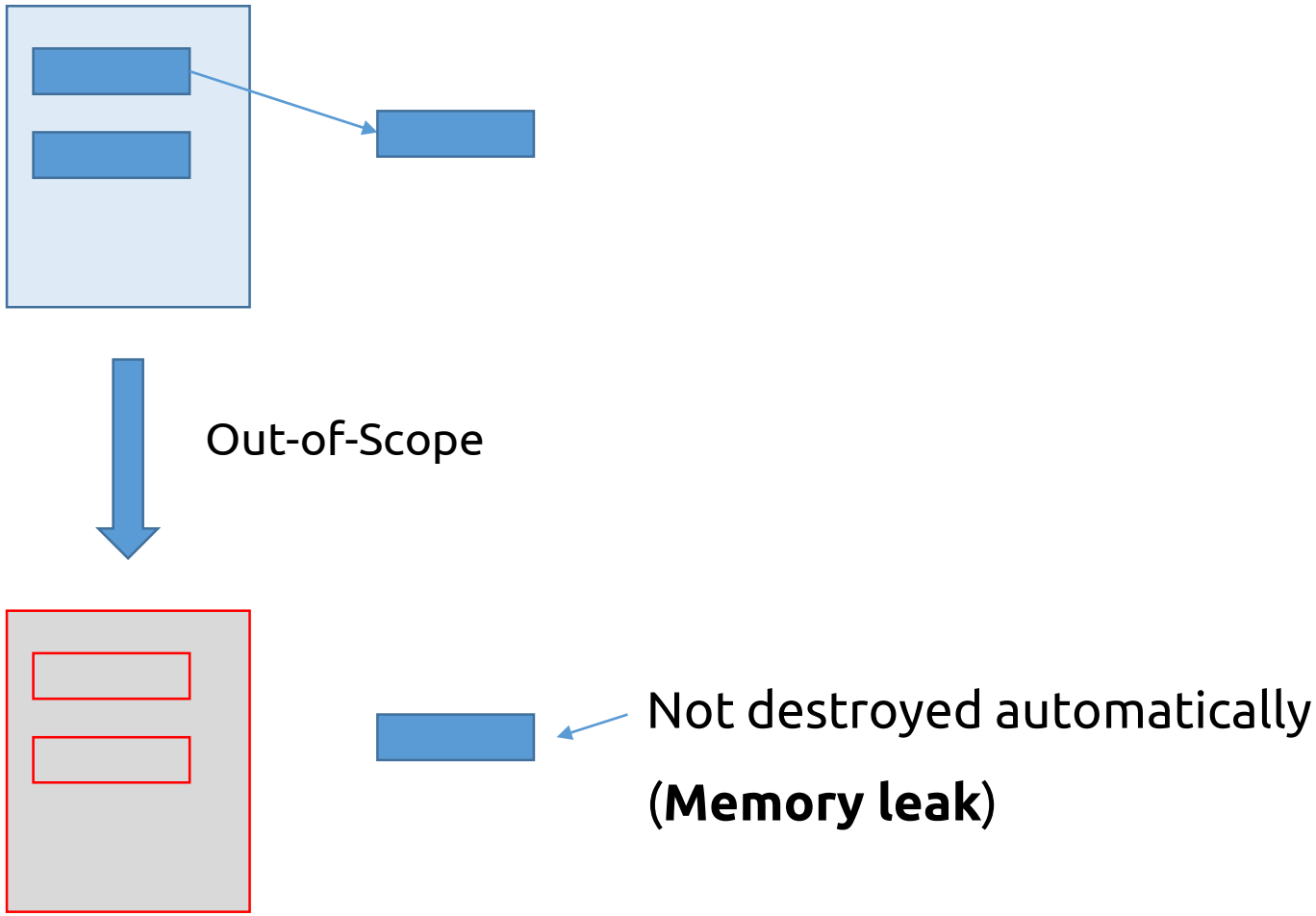
```c
#include <stdio.h>

int main()
{
    int n = 10;
    int * p;

    if (n == 10)
    {
        int * m;
        m = (int *)malloc(sizeof (int));
        *m = 20;
        p = m;
    }
    //m is now destroyed
    //but the location is preserved
    //which it pointed by p also
    printf("%d", *p);
}
```

Usage of `malloc` function

- Dynamic allocation of
  - Array
  - Struct
- Factory Methods

Destruction of Dynamic Memory

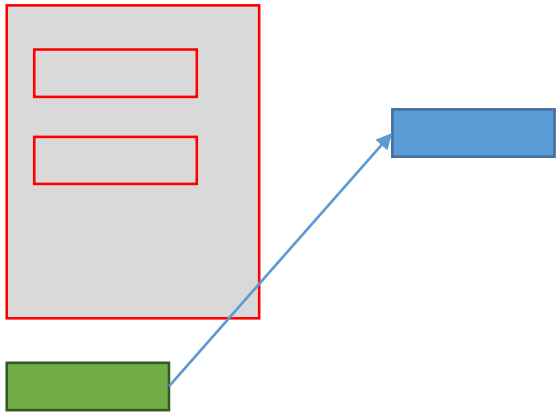
When is it destroyed?



The `free` function

When the outside (dynamic) memory is no longer needed

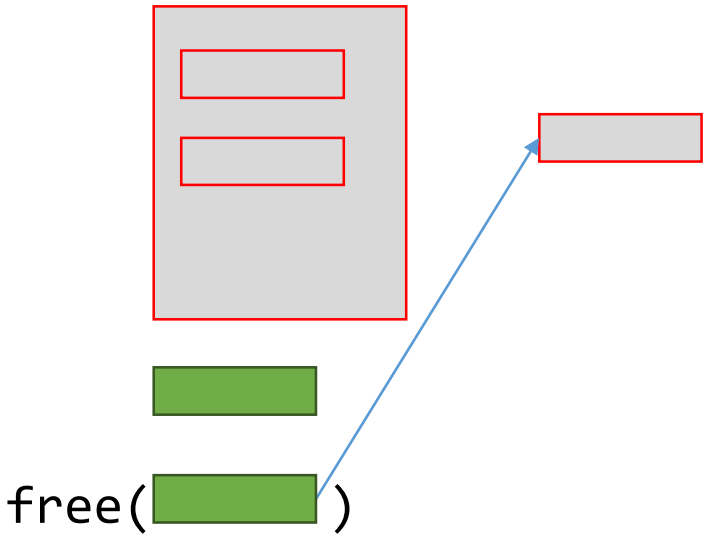
```
void free(void *ptr)
```



The `free` function

When the outside (dynamic) memory is no longer needed

```
void free(void *ptr)
```



The `calloc` function

Same as malloc

```
void *calloc(size_t nitems, size_t size)
```

How many items?

Size of each item

The following two lines produces similar allocation

```
malloc(20 * sizeof (int));

calloc(20, sizeof (int));
```

The `calloc` function

```
void *calloc(size_t nitems, size_t size)
```

- Initializes every bit to zero
- Slower than malloc

The `realloc` function

For resizing existing dynamic memory

```
void *realloc(void *ptr, size_t size)
```

Existing pointer

New size

Either memory is extended,

Or Previous items are copied to new larger location

The realloc function

```
void *realloc(void *ptr, size_t size)
```

Existing pointer

New size

```
int * m = (int *) calloc(2, sizeof(int));
```

The realloc function

```
void *realloc(void *ptr, size_t size)
```

Existing pointer

New size

```
int * m = (int *) calloc(2, sizeof(int));
m[0] = 10;
m[1] = 20;

printf("%d %d \n", m[0], m[1]);
```

The realloc function

```
void *realloc(void *ptr, size_t size)
```

Existing pointer

New size

```
int * m = (int *) calloc(2, sizeof(int));
m[0] = 10;
m[1] = 20;

printf("%d %d \n", m[0], m[1]);

m = (int *) realloc(m, 3 * sizeof(int));
```

The realloc function

```
void *realloc(void *ptr, size_t size)
```

Existing pointer

New size

```
int * m = (int *) calloc(2, sizeof(int));
m[0] = 10;
m[1] = 20;

printf("%d %d \n", m[0], m[1]);

m = (int *) realloc(m, 3 * sizeof(int));
m[2] = 30;
```

The realloc function

```
void *realloc(void *ptr, size_t size)
```

Existing pointer

New size

```
int * m = (int *) calloc(2, sizeof(int));
m[0] = 10;
m[1] = 20;

printf("%d %d \n", m[0], m[1]);

m = (int *) realloc(m, 3 * sizeof(int));
m[2] = 30;
printf("%d %d %d\n", m[0], m[1], m[2]);
free(m);
```

Alternative to Dynamic memory

- Global variables (Scope is increased)
- Static variables (Life-time is increased)

Global Variable Initialization

Automatically initialized

```c
#include <stdio.h>

int m;  //global variable

int main()
{
    printf("%d", m);     //0
}
```

Global Variable Initialization

Automatically initialized

Data Type	Initializer
int	0
char	'\0'
float	0
double	0
pointer	NULL

Credit: https://www.tutorialspoint.com/cplusplus/cpp_variable_scope.htm

Variable Shadowing

In case of same name, local variable takes preference

```c
#include <stdio.h>

int g = 10; //global

int main()
{
    int g = 20; //local

    printf("%d", g);    //20
}
```

The `static` Keyword

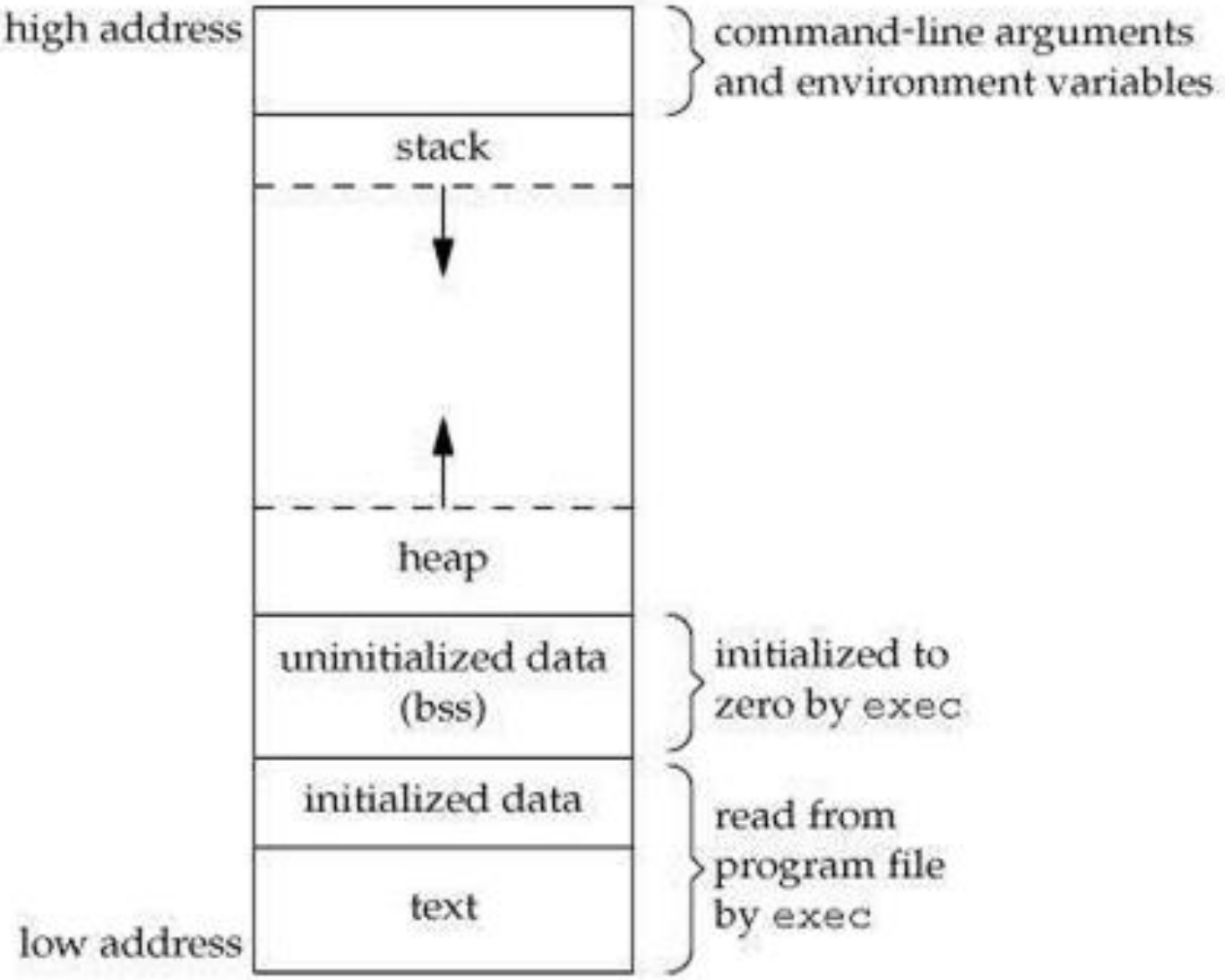
Prevents local variable from being destroyed until the program terminates

The `static` Keyword

Prevents local variable from being destroyed until the program terminates

```c
#include <stdio.h>
void fn()
{
    static int m = 0;    //initializes only once
                         //in the lifetime
    m = m + 1;
    printf("%d\n", m);
}   //m is not destroyed

int main()
{
    fn();    //1
    fn();    //2
}
```

Memory Layout of a C Program



https://www.hackerearth.com/practice/notes/memory-layout-of-c-program/

Task 1

Create a resizable array of integers with the following options

1. Add a new number to the array
2. Display all numbers in the array
3. Delete an existing integer (by index)

The storage should be flexible so that no memory is wasted.

Task 2

Design a record book that will hold the name, id and total marks of a student. The following options should be available.

1. Add a new record
2. Display all records
3. Edit an existing record
4. Delete an existing record
5. Search for a record by name or id

The storage should be flexible so that no memory is wasted.

Task 3

Integrate Persistent storage (File) in Task 2, i.e. all the contents of the array should be written to file at the end of the program, and it should load all the contents from the file at the beginning of the program.