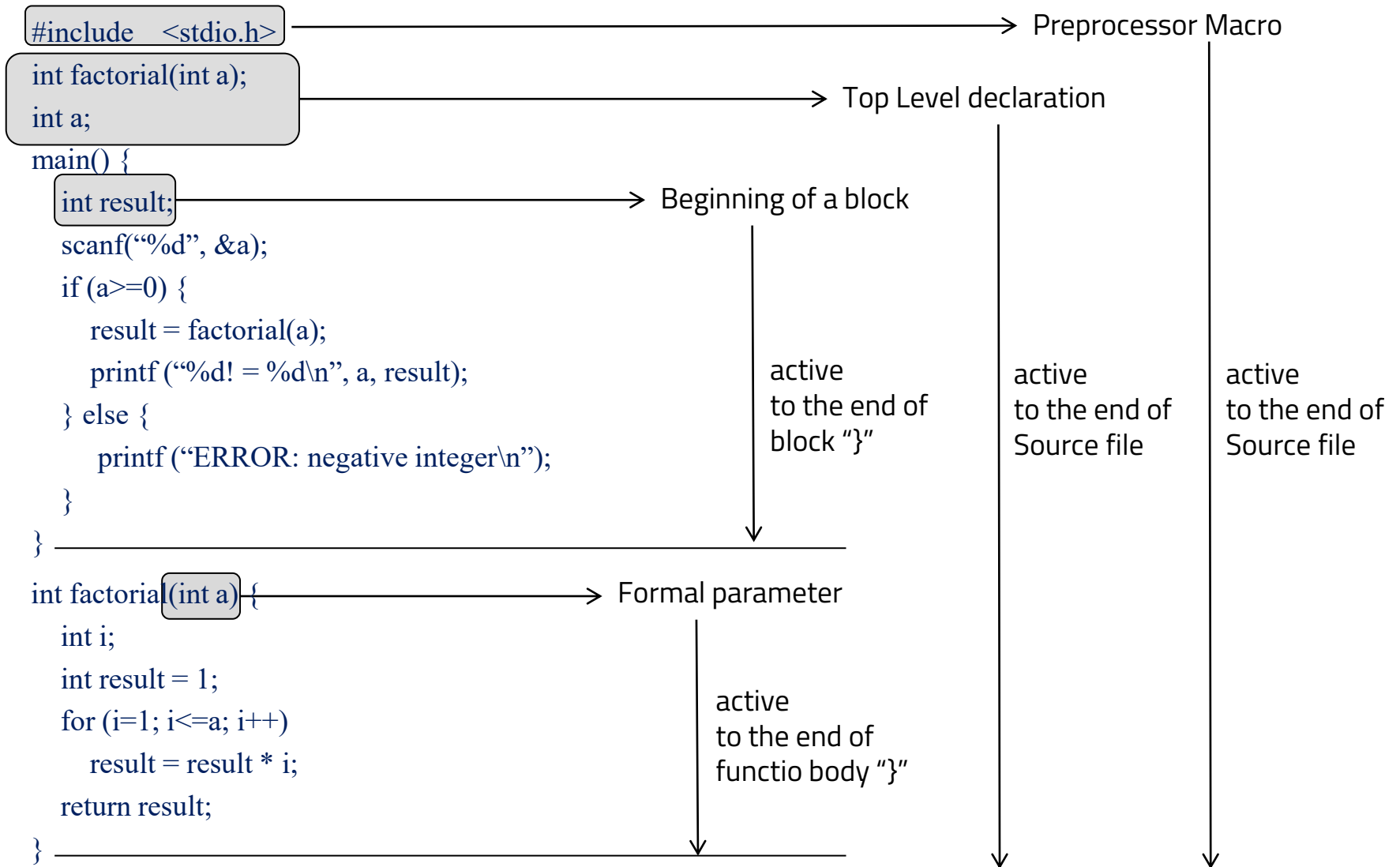


시스템 프로그래밍을 위한 C언어

Variable Liveness Scope

현대자동차 입문교육
박대진 교수

Active Region of Variable Declaration



Scope: Local vs. Global Variable

```
#include<stdio.h>
```

```
void func1();
```

```
int gl;
```

```
main() {
```

```
    int lo = 1, gl = 1;
```

```
    func1( );
```

```
    printf("initial value of global : %d\n", gl);
```

```
    printf("initial value of local : %d\n", lo);
```

```
}
```

```
func1() {
```

```
    int lo = 2;
```

```
    ...
```

```
}
```

not same (different memory space)

local
global

priority:

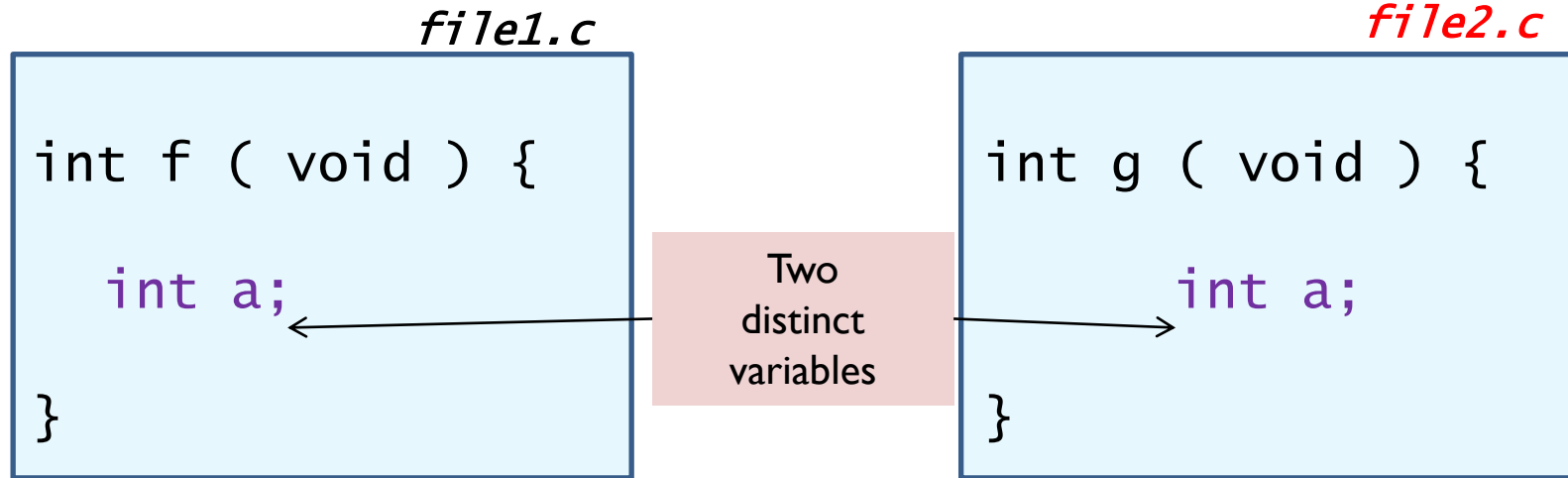
gl (in main) >

gl (out of main)

global

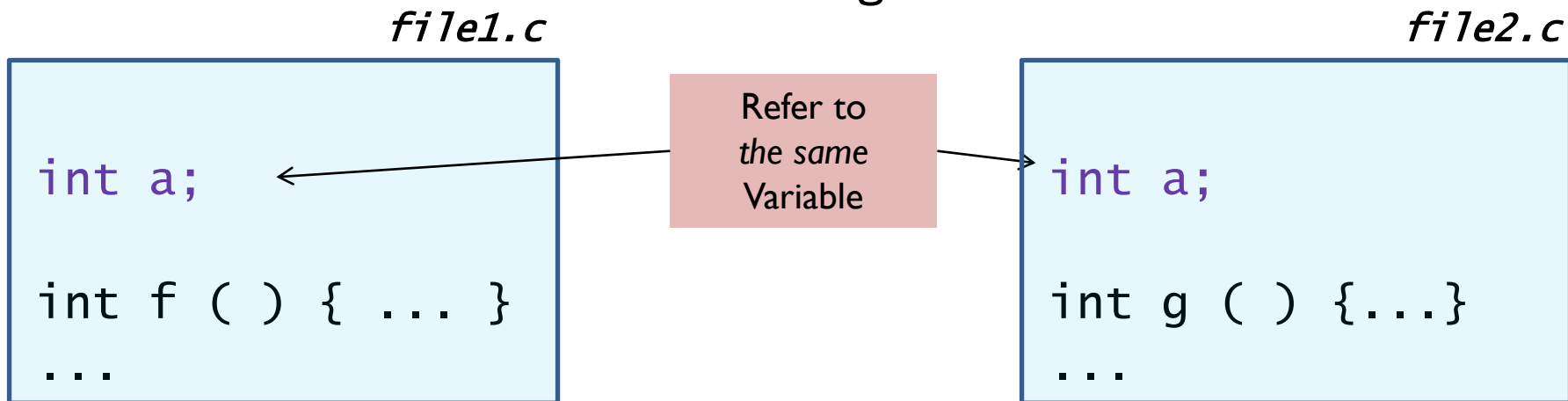
local

Memory Allocation on Multiple Files



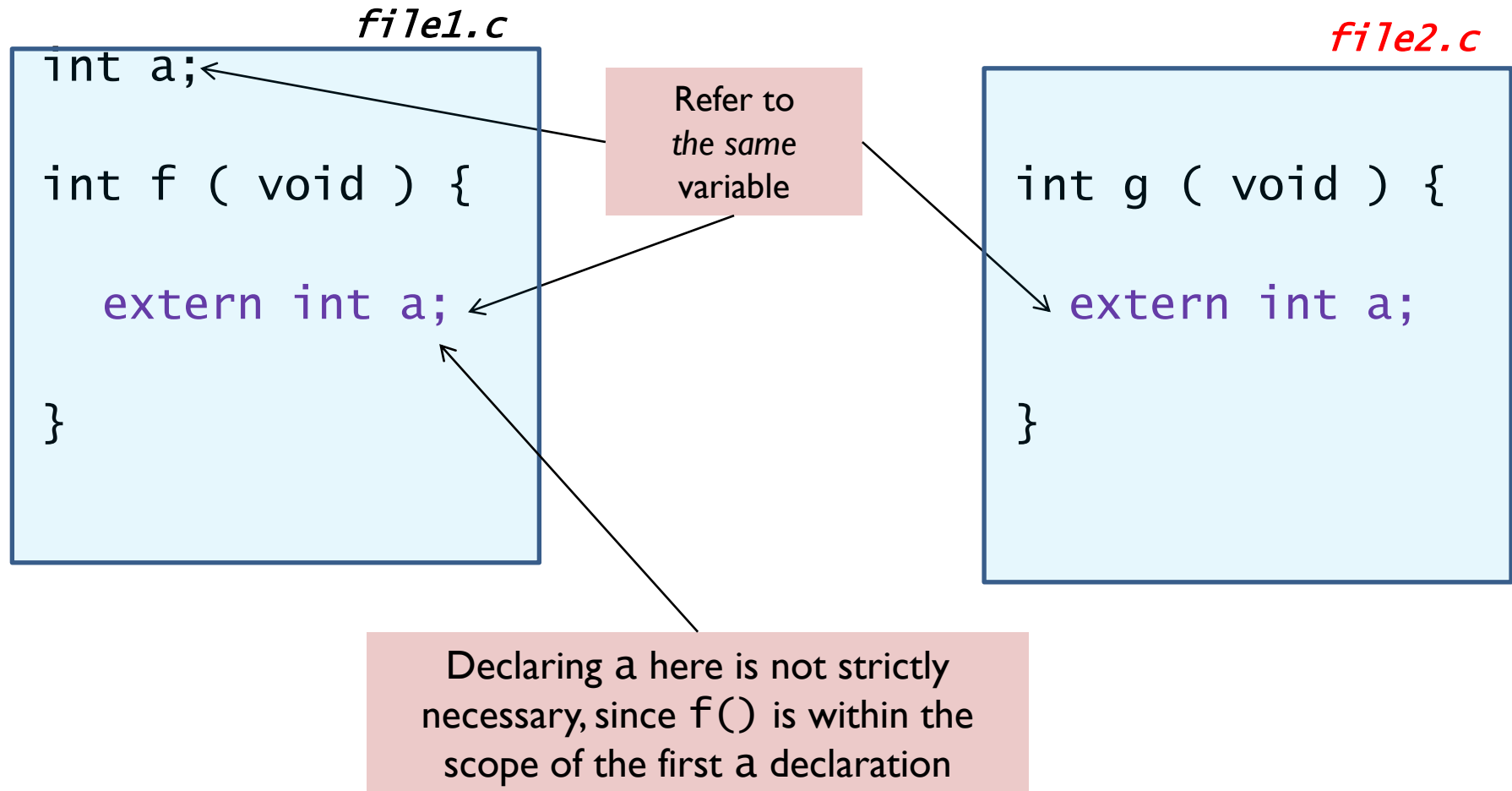
Both a's are auto storage class (local extent)

external linkage



Somewhere allocated declaration: 'extern'

A local variable declared as extern has an external linkage.



Memory Allocation: Static vs. auto(local)

◆ Static vs local extent

❖ static extent (cf: does not mean “global” which is related to scope)

- Allocation remains from the beginning of program execution to program termination
- Avoid overhead of allocating, initialization, de-allocating memory with each function call

❖ local extent

- Create upon entry to a block or function and destroy upon exit from the block or function

```
{  
    static int S = 0;  
    auto int L = 0;  
    L = L + 1; S = S + 1;  
    printf (“L = %d, S = %d \n”, L, S);  
}
```

After execution many times : L = 1, S = 1

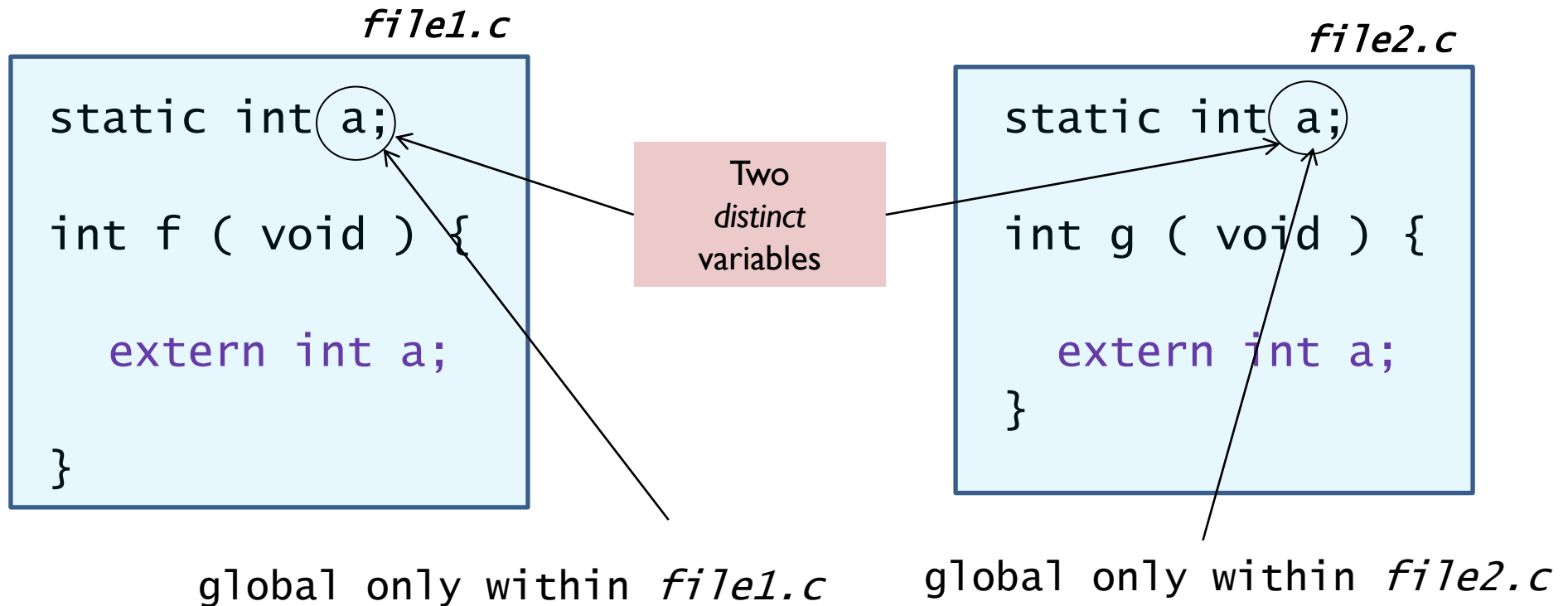
L = 1, S = 2

L = 1, S = 3

Static variable vs. global variable

Q: What if you have a “global” variable just within a file but not visible outside the file?

A: Declare it as static within the file:



Source code refactoring (1/5)

```
int g1;  
int g2;  
int foo1(int i) {  
    return i + g1;  
}  
  
int foo2(int i) {  
    int g2=100;  
    return i + g2;  
}
```

```
int main(...) {  
    ...  
    int sum;  
    g1 = 3  
    sum = foo1(g1)+g1;  
    ...  
}
```


Source code refactoring (2/5)

```
int g1;  
int g2;
```

```
int foo1(int i);  
int foo2(int i);
```

```
int main(...) {  
    .....  
}  
int foo1(int i) {  
    return i + g;  
}  
int foo2(int i) {  
    int g2=100;  
    return i + g2;  
}
```

Source code refactoring (3/5)

foo.h

```
int g1;  
int g2;  
  
int foo1(int i);  
int foo2(int i);
```

```
#include "foo.h"  
int main(...) {  
    .....  
}
```

```
#include "foo.h"  
int foo1(int i) {  
    ...  
}  
int foo2(int i) {  
    ...  
}
```

foo.c

Source code refactoring (4/5)

foo.h

```
int g1;  
int g2;  
  
int foo1(int i);  
int foo2(int i);
```

```
#include "foo.h"  
int main(...) {  
    .....  
}
```

foo1.c

```
#include "foo.h"  
int foo1(int i) {  
    ...  
}
```

foo2.c

```
#include "foo.h"  
int foo2(int i) {  
    ...  
}
```

Source code refactoring (5/5)

foo.h

```
int foo1(int i);  
int foo2(int i);
```

main.c

```
#include "foo.h"  
extern int g1;  
int main(...) {  
    int sum;  
    g1 = 3;  
    sum = foo1(g1)+g1;  
}
```

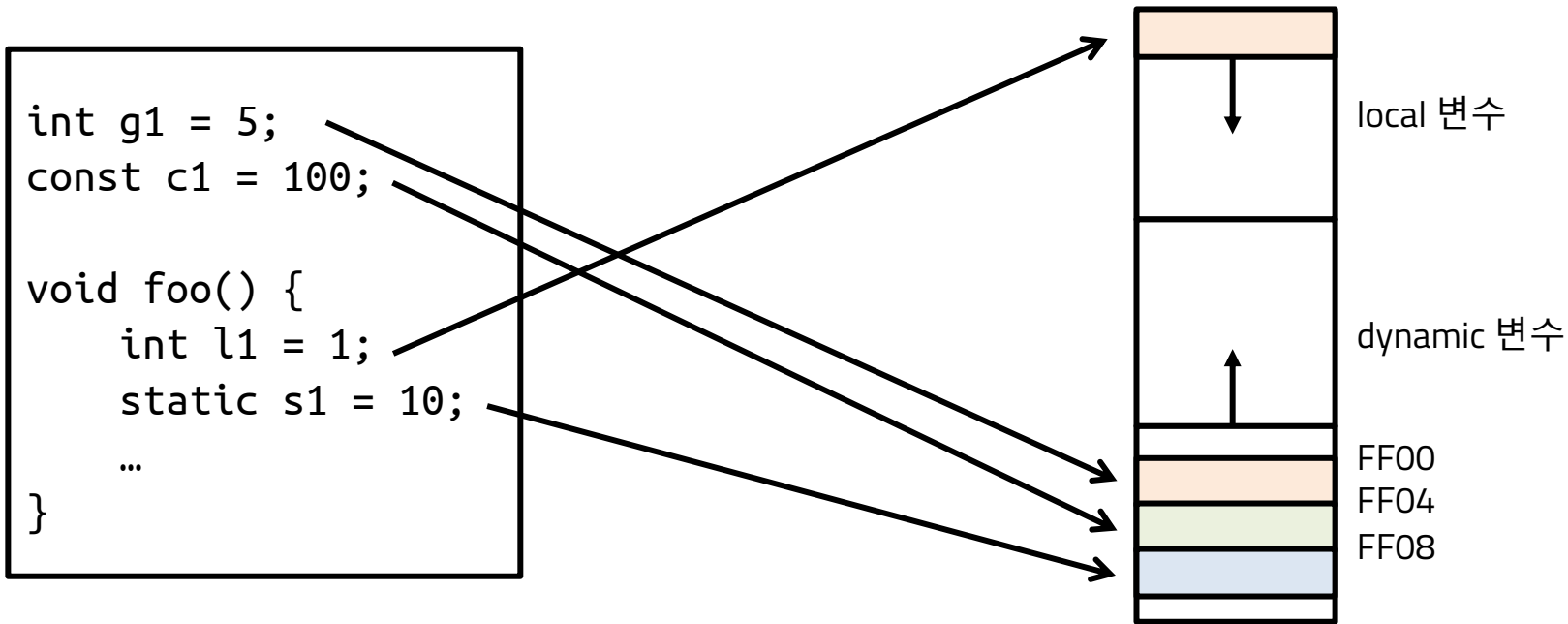
foo1.c

```
#include "foo.h"  
int g1;  
int foo1(int i) {  
    ...  
}
```

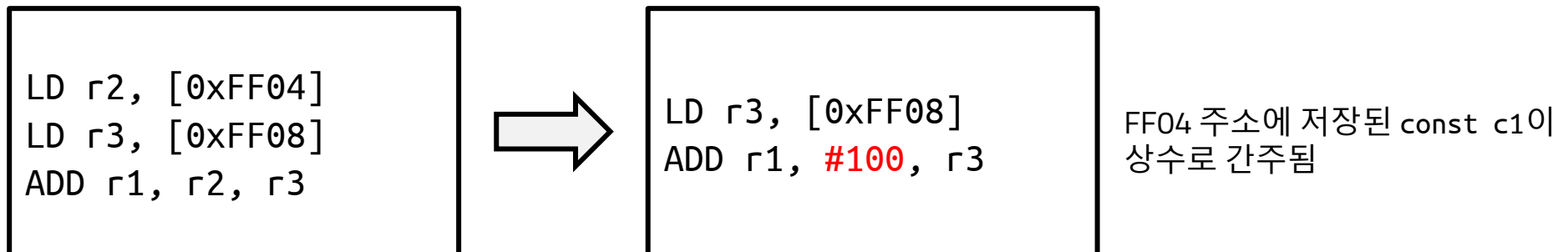
foo2.c

```
#include "foo.h"  
static int g2;  
int foo2(int i) {  
    ...  
}
```

Variable Allocation (global, static, const)



const 키워드 사용 결과



변수의 scope와 lifetime

함수, 하나의 파일, 모든 파일

scope

- 변수에 접근할 수 있는 범위

C 코드

lifetime

- 변수의 값이 살아있는 기간

