Storage Classes

Automatic Variables

- Consider the following C code

```
foobar(int n)
      {
      float xx;
      . . .
      }
```

The variable xx comes into existence only when the function foobar is called

Automatic Variable

[2]

Automatic Variables

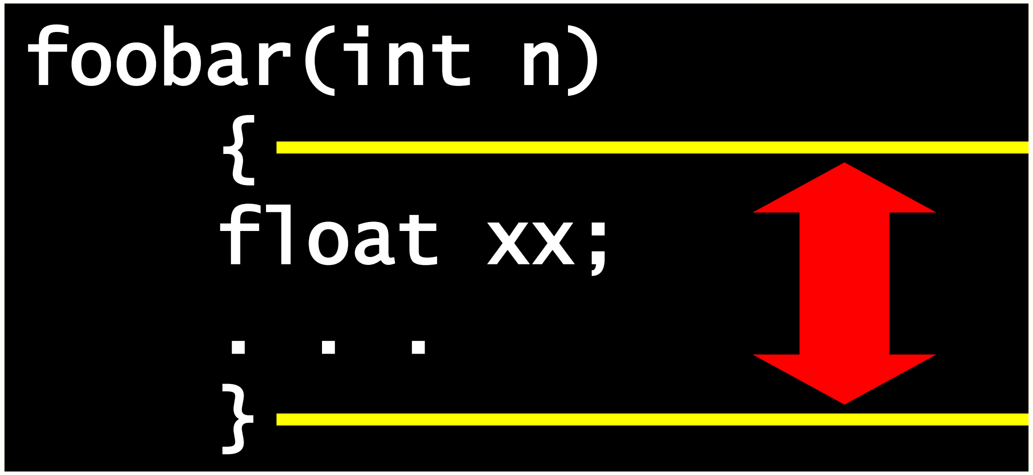
- Variable xx comes to existence when function foobar in which it is declared is called. i.e only then memory is allocated for it.
- As soon as we exit from the function foobar this memory area is lost and the variable xx does not exist any longer
- Also called as **Local Variable**

Automatic Variables

- By default a variable declared in any function is assumed to be an automatic variable but we can explicitly declare a variable to be of automatic storage class using the keyword **auto**.

```
auto float xx;
```

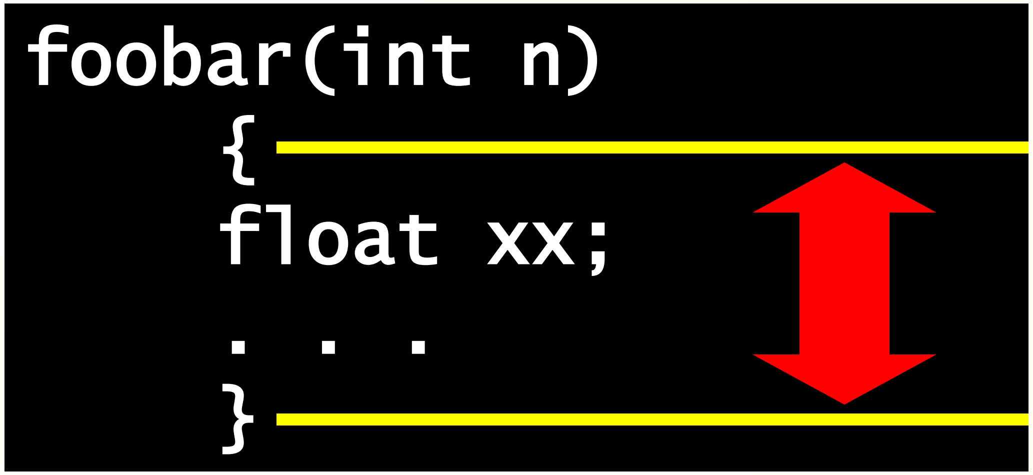
Automatic Variables(Lifetime)



```
foobar(int n)
    {
    float xx;
    . . .
    }
```

- their **lifetime** is limited to that of the function, i.e., they exist only for the duration of the function execution.

Automatic Variables(Scope)



```
foobar(int n)
     {
     float xx;
     . . .
     }
```

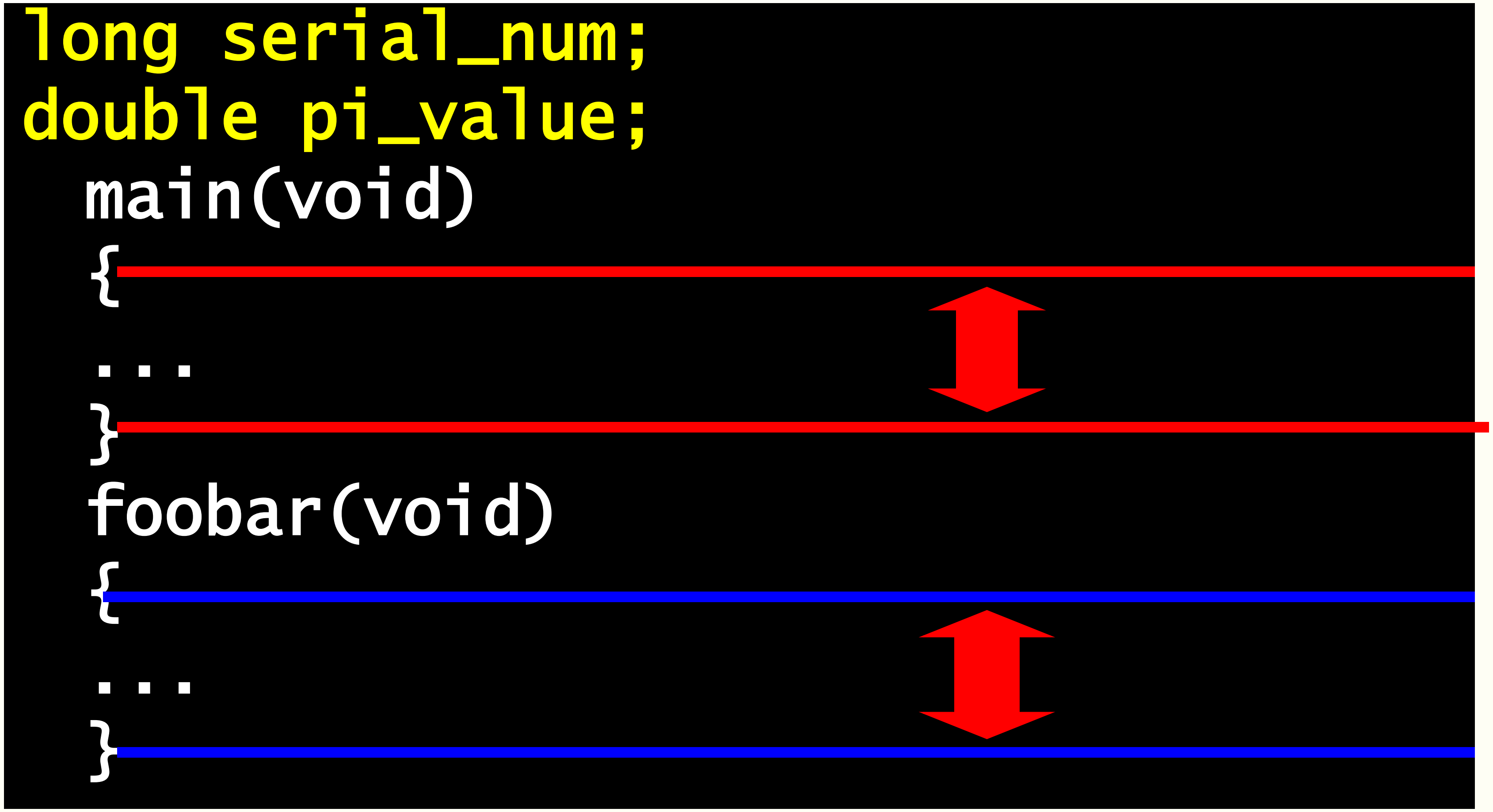
- The **scope** or the visibility of an auto variable is limited to the function itself

Global Variable

```
long serial_num;
double pi_value;
   main(void)
   {
   ...
   }
   foobar(void)
   {
   ...
   }
```

GLOBAL VARIABLE

Global Variables(Scope)



```
long serial_num;
double pi_value;
   main(void)
   {
   ...
   }
   foobar(void)
   {
   ...
   }
```

EXTVARS.C

This program demonstrates the global scope of external variables.

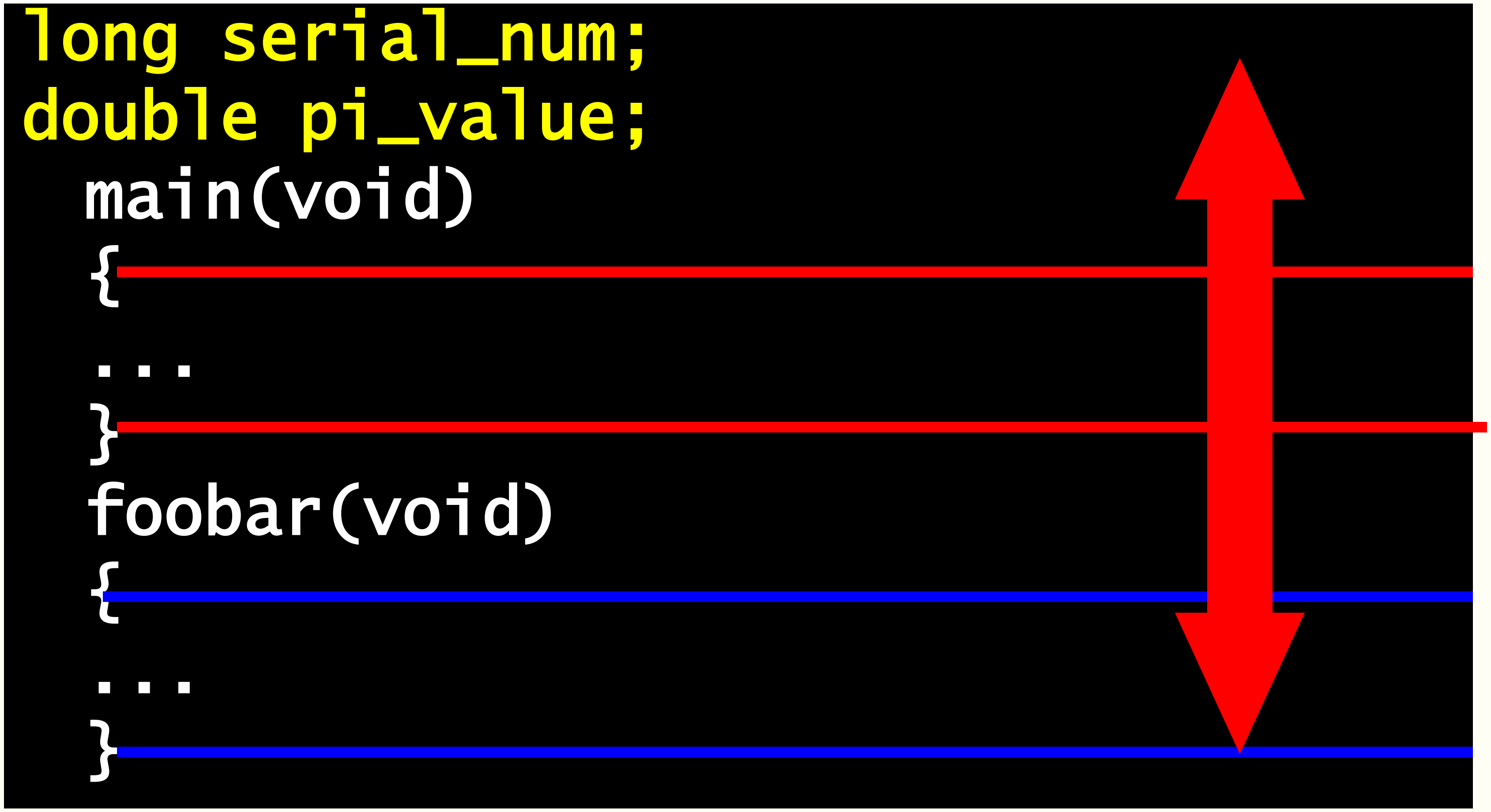
```
#include <stdio.h>
/* function prototype declaration */
void foobar(void);
/* external variables declared here */
int num=1;
float xx=33.33;
int main()
{
    printf("\nOriginal values : %d %f",num,xx);
    num += 6;
    xx += 10.0;
    printf("\nModified values : %d %f",num,xx);
```

EXTVARS.C

This program demonstrates the global scope of external variables.

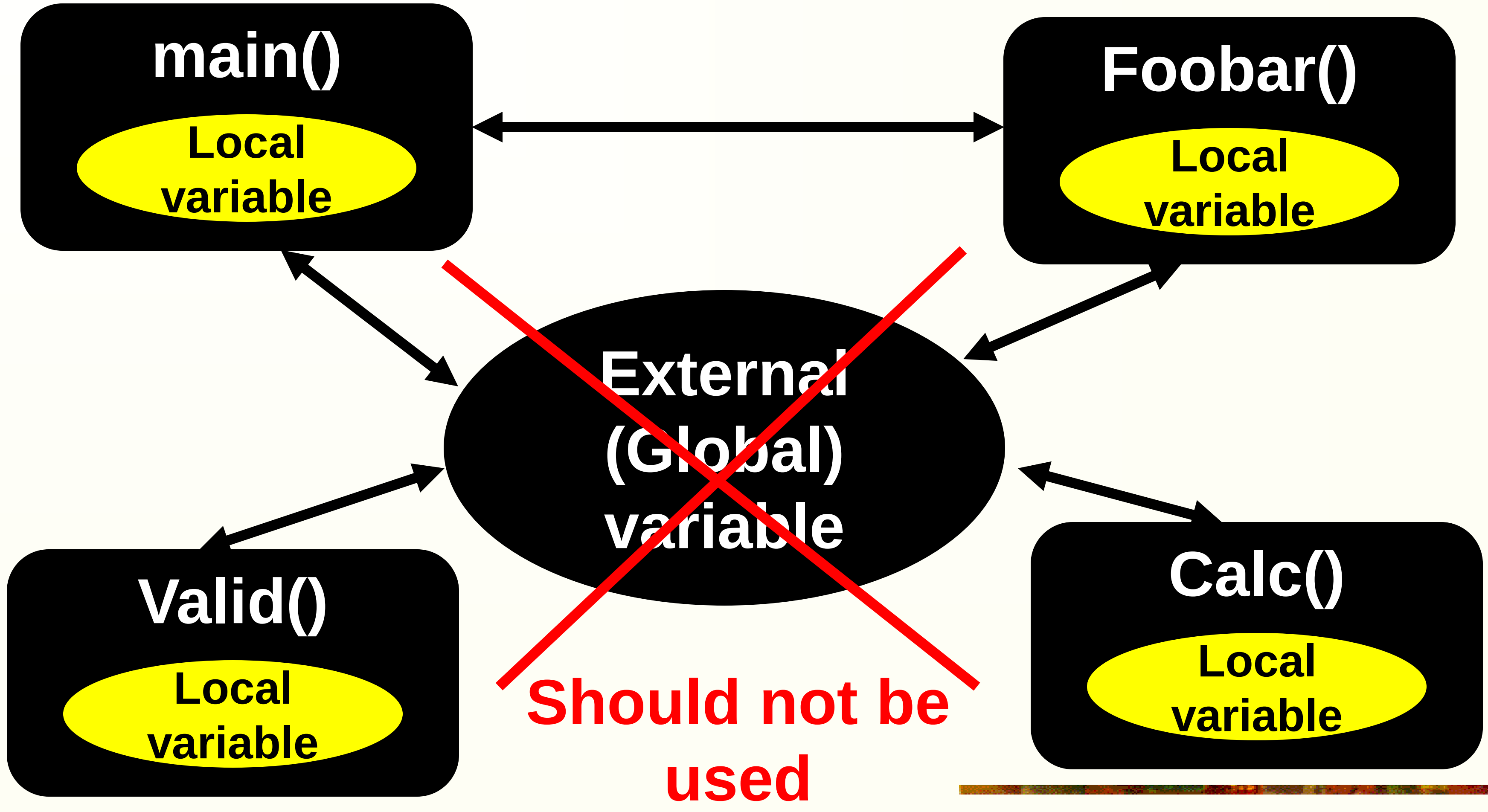
```
foobar();
    printf("\nAfter call to foobar : %d %f",num,xx);
    return 0;
}
/* function definition for foobar */
void foobar(void)
{
    num++;
    xx *= 2.0;
    return;
}
```

Global Variables(Lifetime)



```
long serial_num;
double pi_value;
   main(void)
   {
   ...
   }
   foobar(void)
   {
   ...
   }
```

Information Exchange



Static Variables

- **Static** variables declared inside a function have their scope limited to the function.
- They retain a fixed memory location until the end of the program execution.

Static Variables

- A static variable is assigned a memory location at the beginning of program execution.
- If an initial value is given, the initialization also takes place at the beginning of program execution.

NEXTNUM.C

```
/*Program demonstrating use of static variable in a function
 to generate a sequence of even integers.*/
#include <stdio.h>
int main()
{
    int j,nextnum(void);
    for(j=0;j<20;j++) printf("\n%d",nextnum());
    return 0;
}
int nextnum(void)
{
    static int first=0;
    return(++first);
}
```

Variables

Variable Type	Scope	Lifetime
auto (local)	function	function
external (global)	file (all functions in file)	program
static	function	program

Register Variables

- Register variables are used to indicate heavily used variables by placing the keyword **register** in front of the variable declaration.
- If the feature is supported by the compiler, these variables are stored in the cache memory for high speed access.

Points to Remember

- External variables have global visibility. It is recommended that the use of external variables should be minimized as much as possible.

- Register variables can provide fast CPU access to frequently used variables.

Storage classes

THANK YOU