Dynamic Memory Allocation
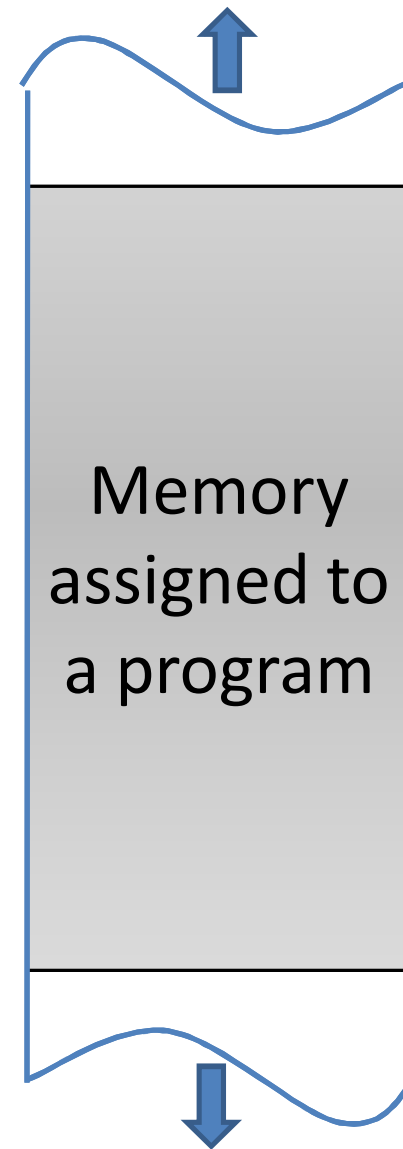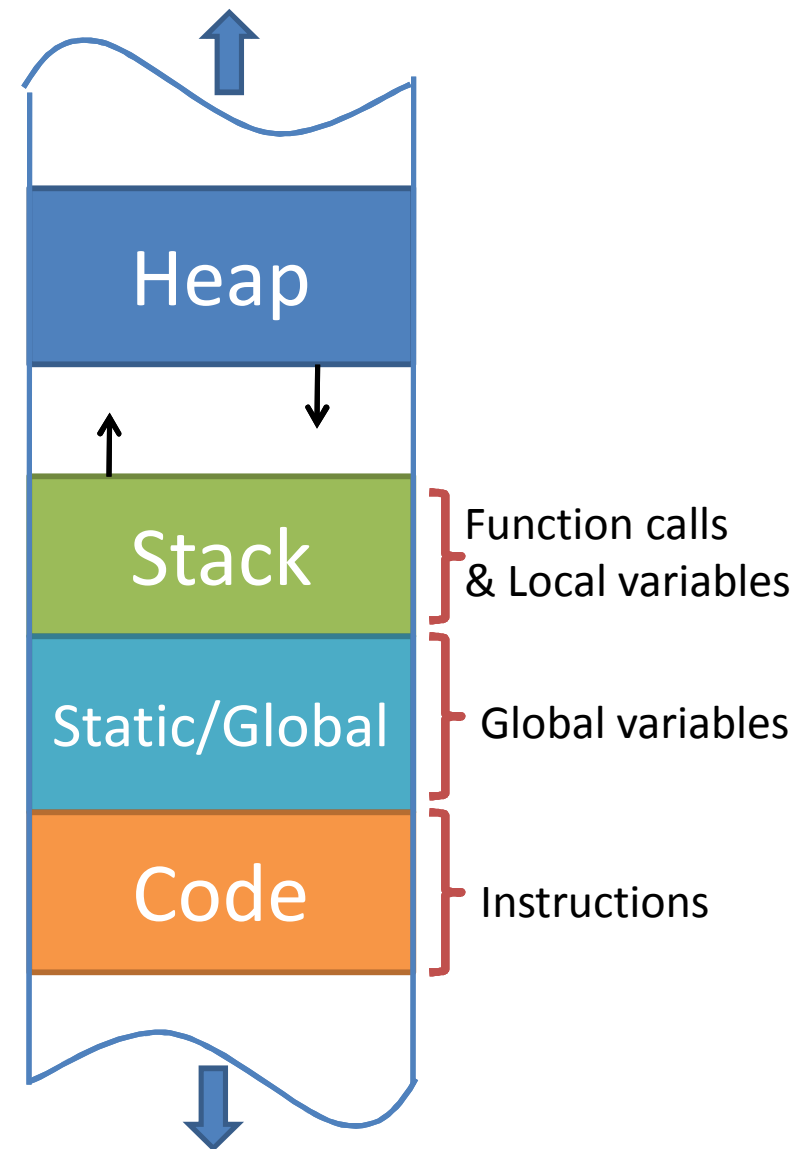
Memory of a Program

Whenever a program executes, the operating system allocates some space in memory for it.

Memory assigned to a program

Memory of a Program

The memory allocated to a program is divided into four parts.
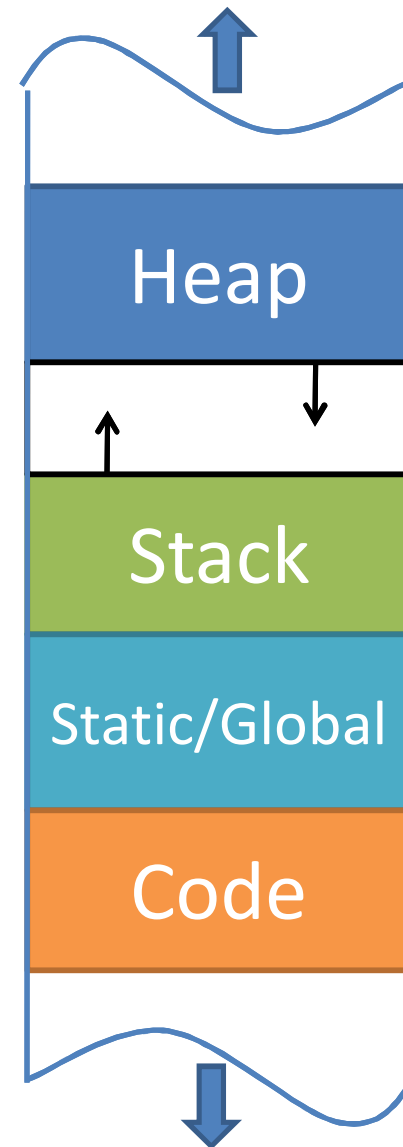
Memory of a Program

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}

int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}

int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```

Use of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}

int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}

int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```

Heap

Stack

Static/ Global

Code

Use of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}

int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}

int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```
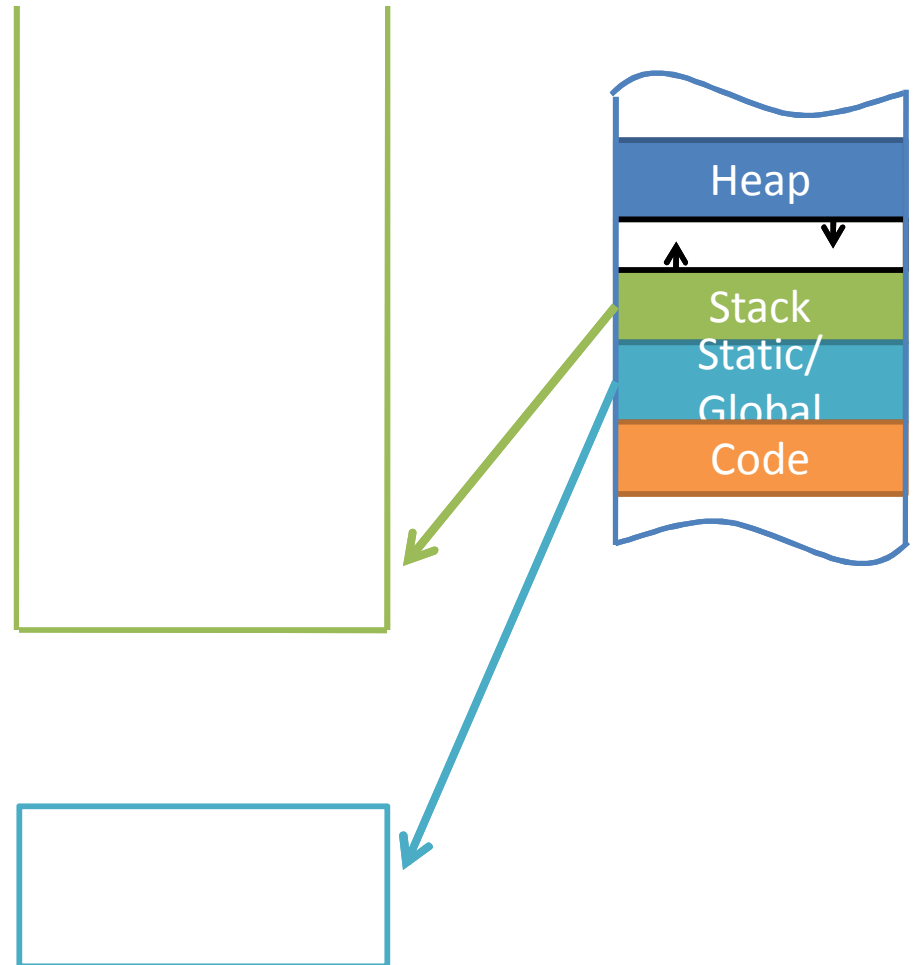
Heap

Stack

Static/
Global

Code

total

Use of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}


int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}


int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```
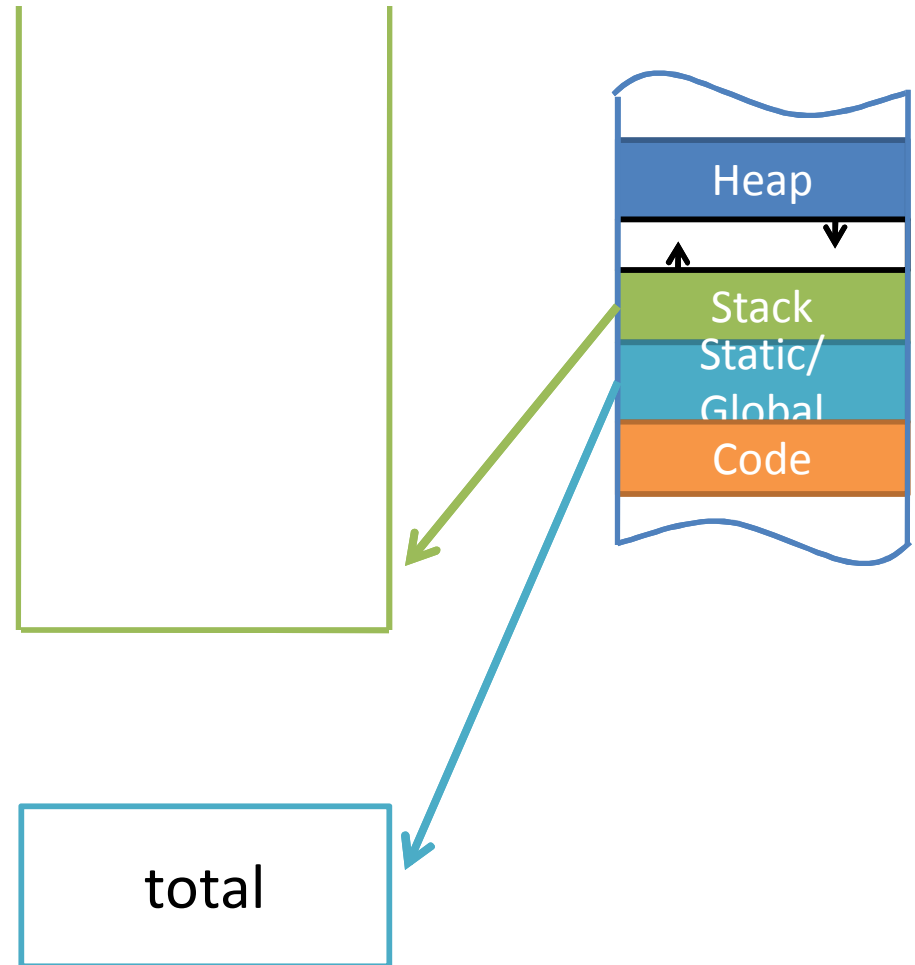
main()
a, b

total

Heap

Stack

Static/Global

Code

Stack frame for main() function

Use of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}


int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}


int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```
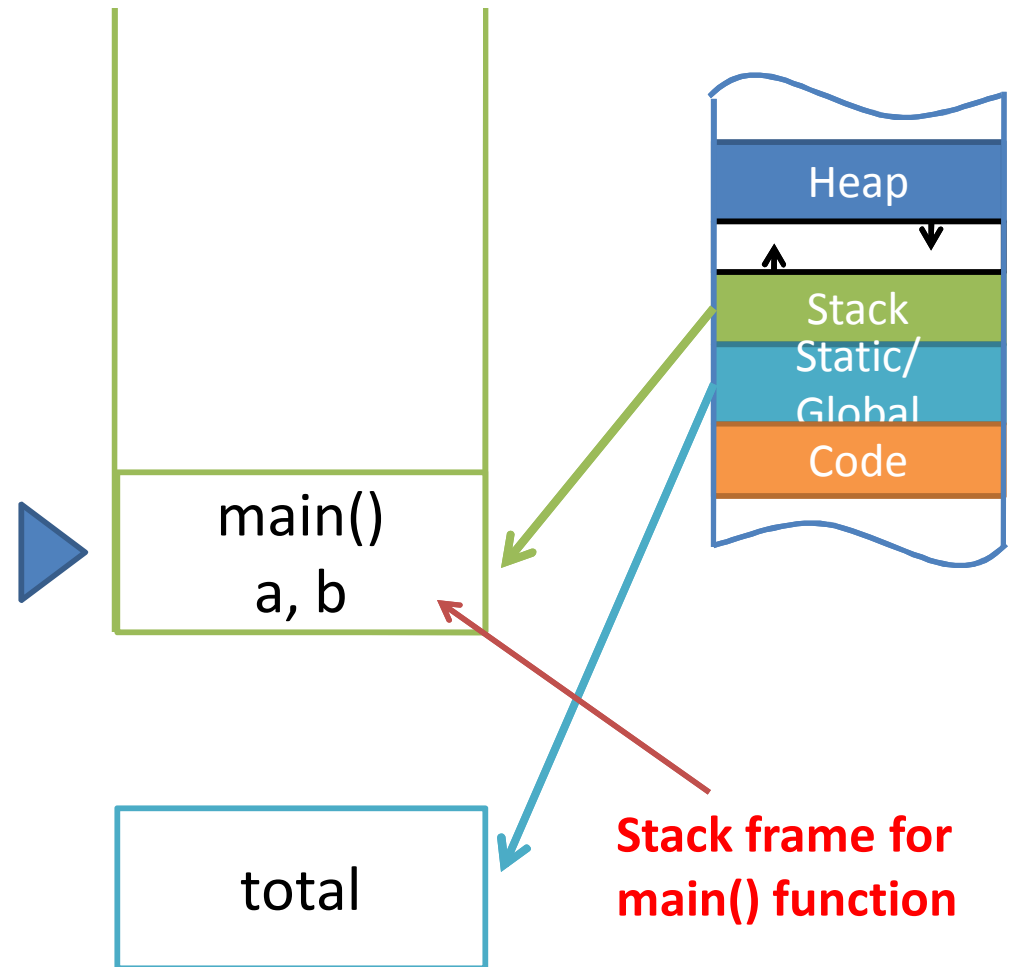
main()
a, b

Heap

Stack

Static/
Global

Code

total

Use of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}

int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}

int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```
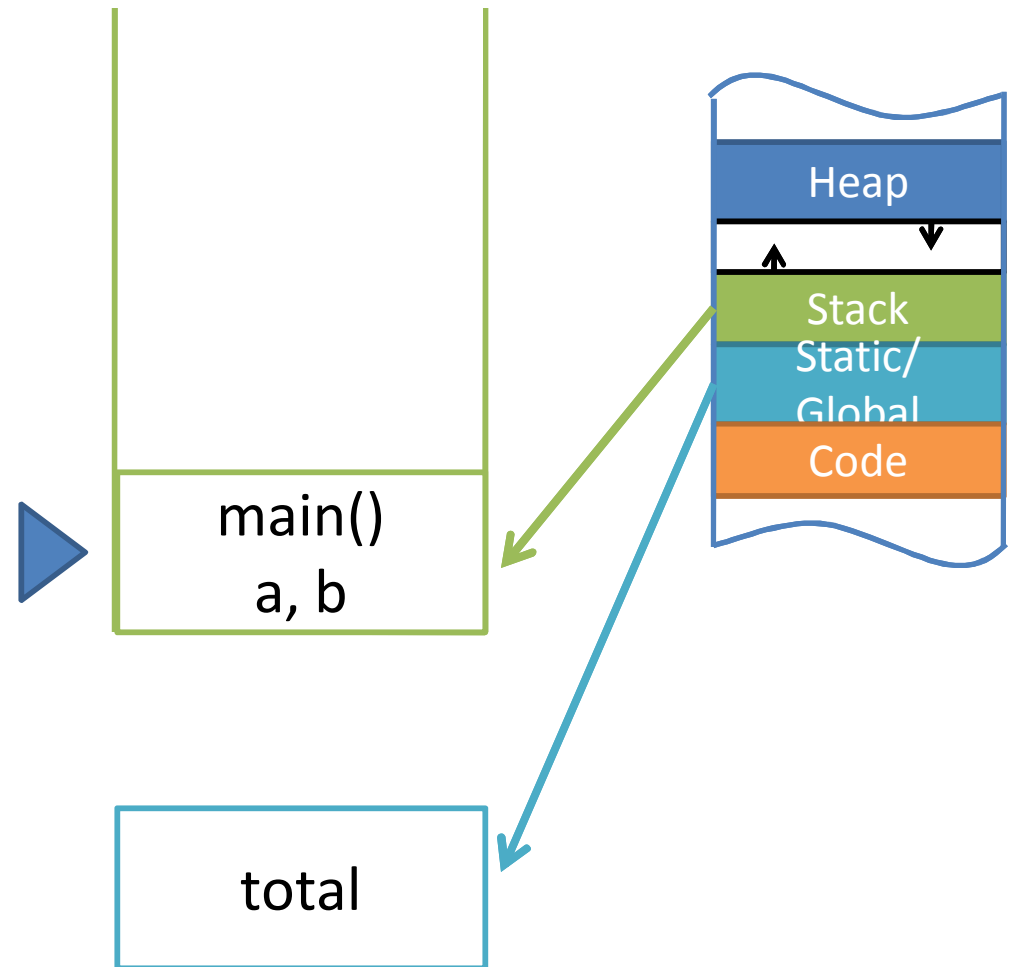
SquareOfSum()

x, y, z

main()

a, b

Heap

Stack

Static/Global

Code

total

Use of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}

int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}

int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```
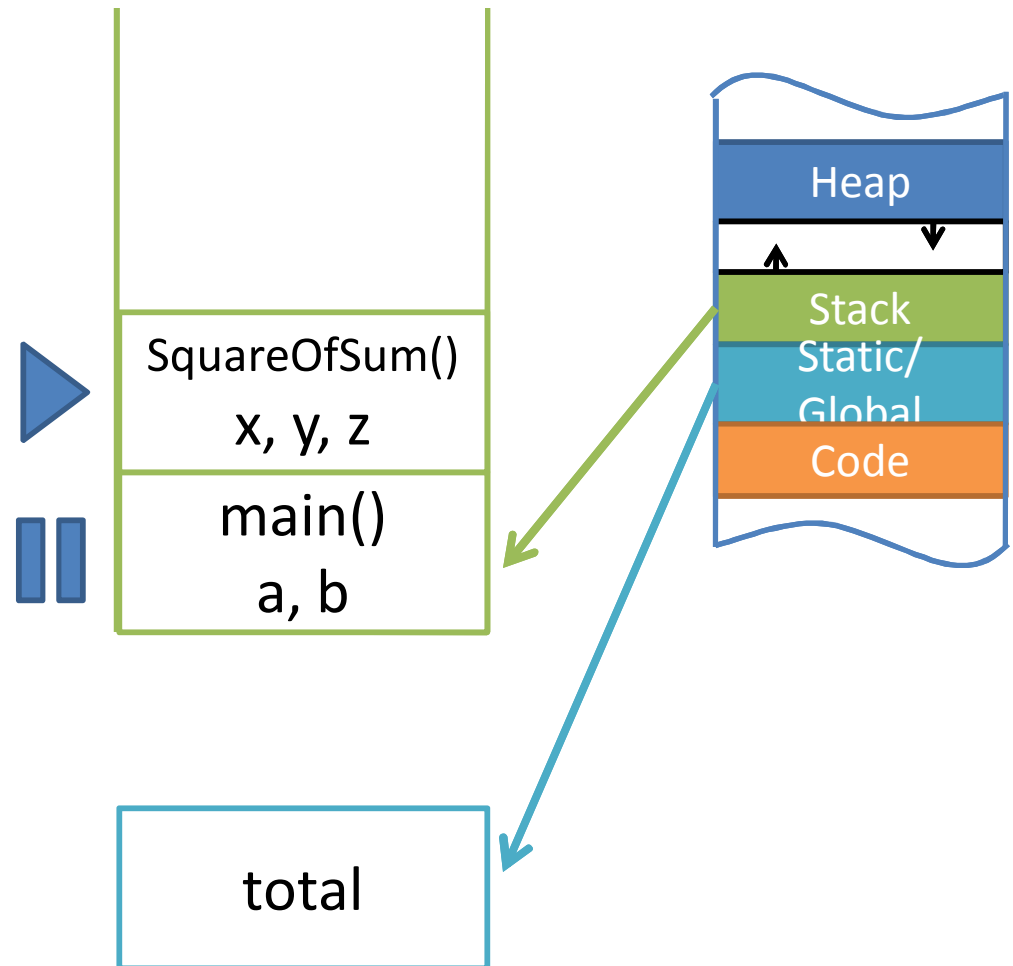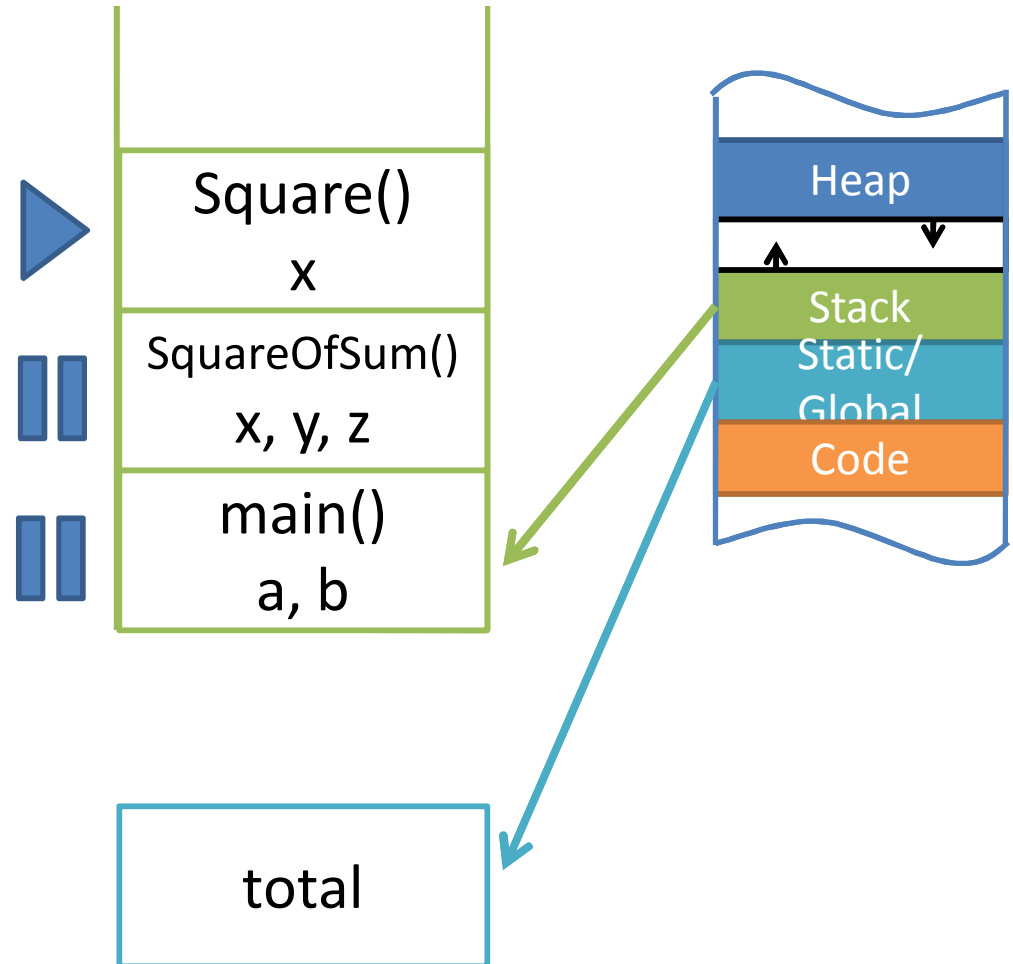
Use of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}

int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}

int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```

SquareOfSum()
x, y, z

main()
a, b

Heap

Stack

Static/
Global

Code
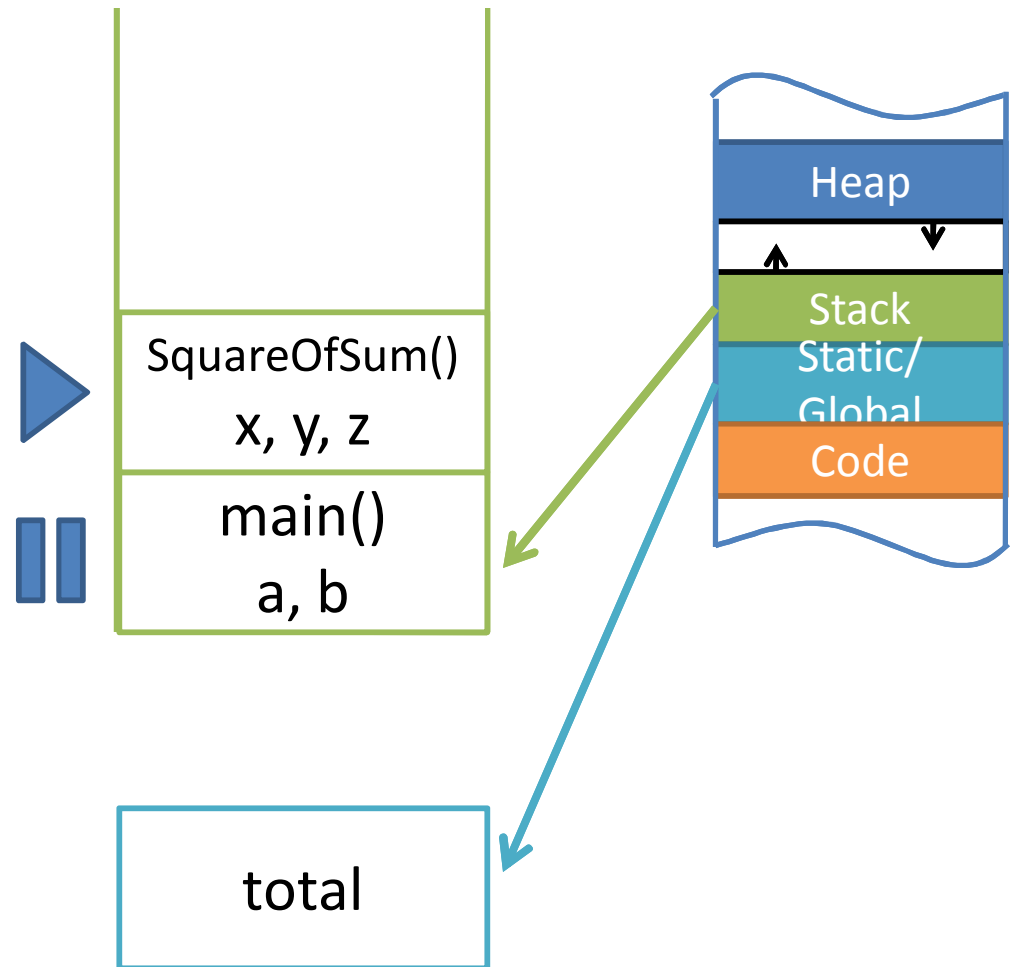
total

Use of Stack

```
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}


int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}


int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```

main()
a, b

total

Heap

Stack

Static/
Global

Code

Use of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}


int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}


int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```
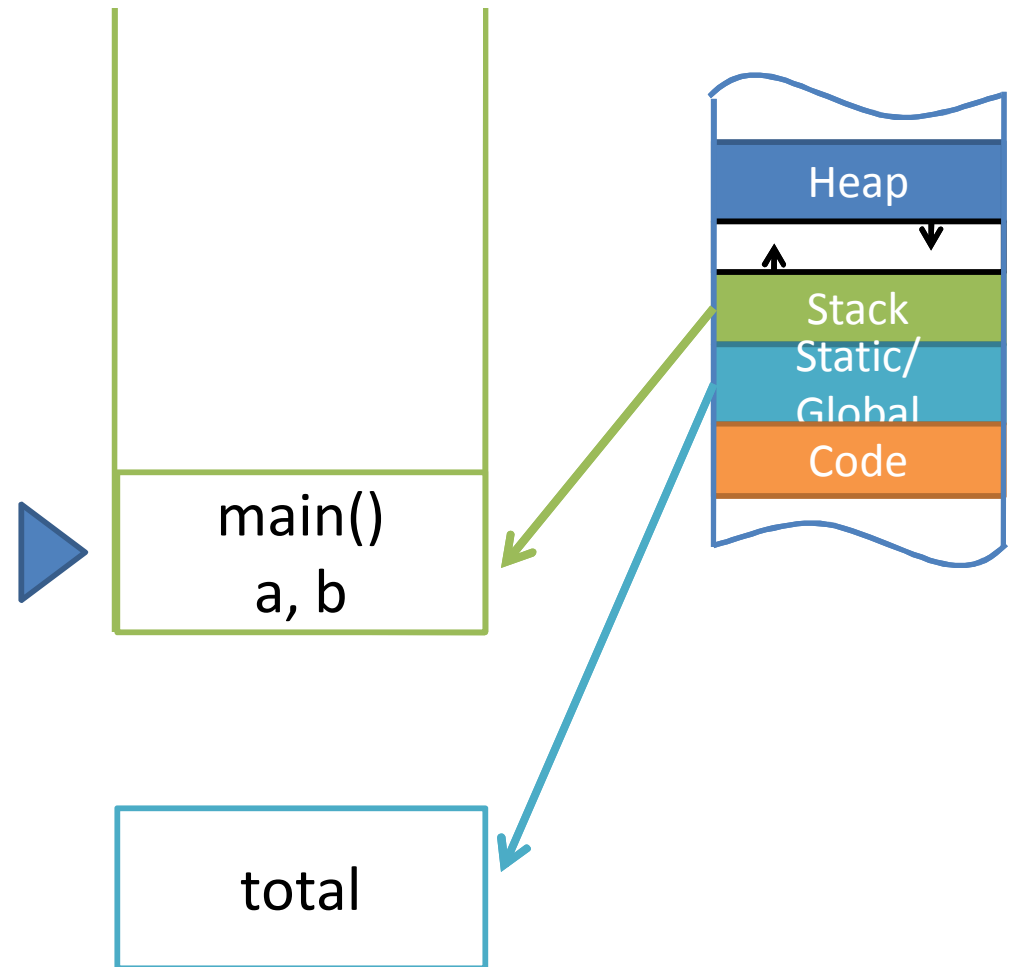
Use of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}


int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}


int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```
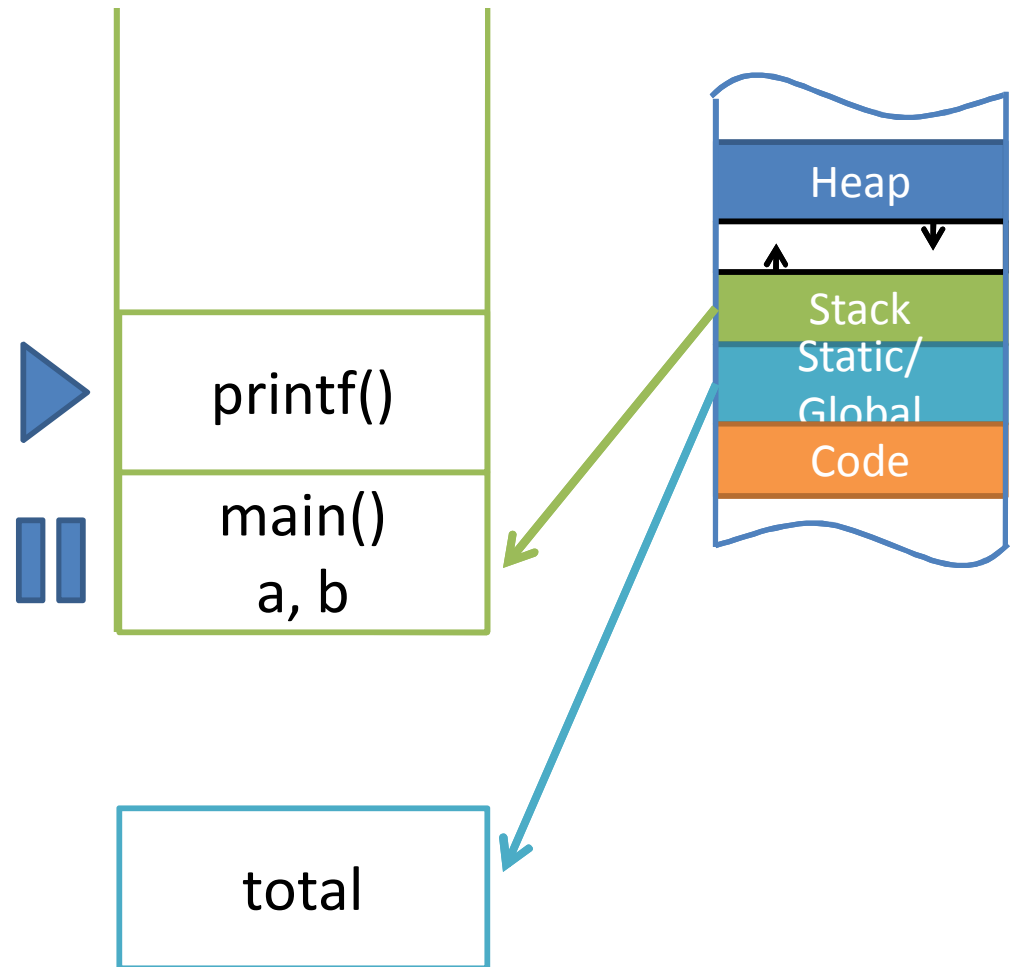
main()
a, b

total

Heap

Stack

Static/
Global

Code

Use of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}


int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}


int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```
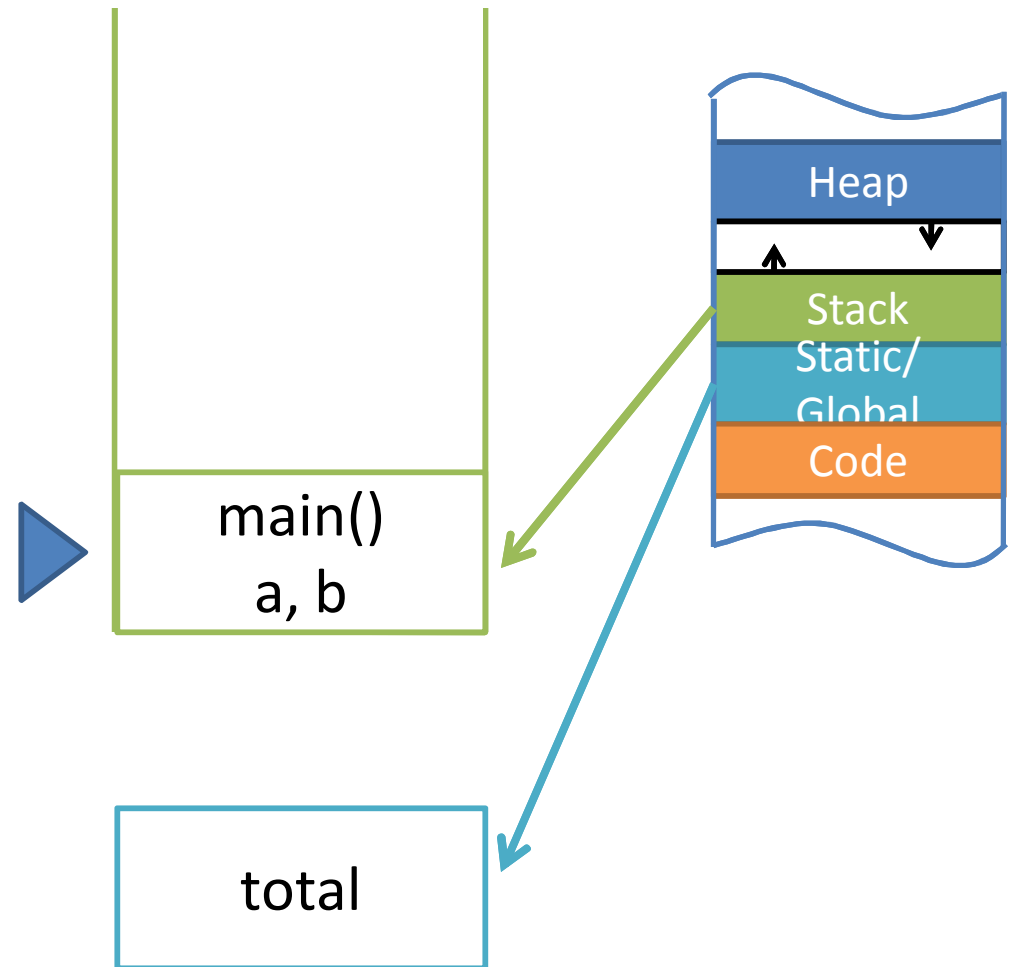
Heap

Stack

Static/ Global

Code

Use of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}


int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}


int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```
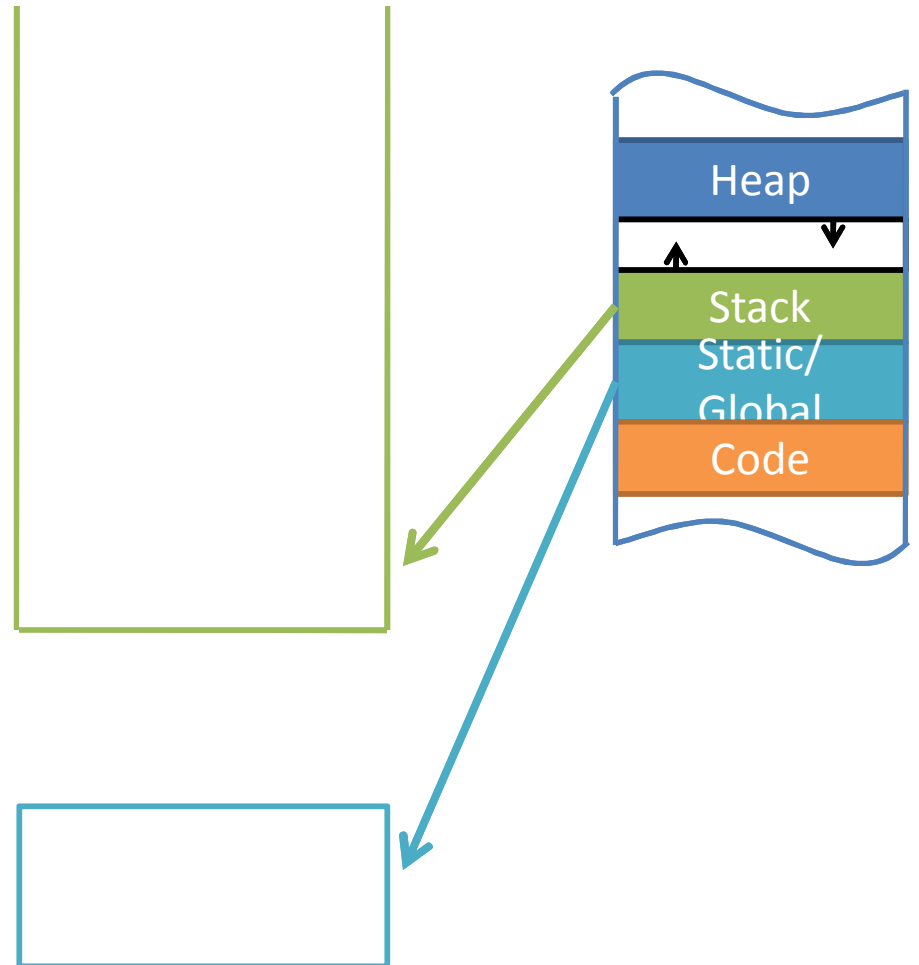
Heap

Stack

Static/
Global

Code

LIFO – Last In First Out

Limitation of Stack

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return(x*x);
}

int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return(z);
}

int main()
{
    int a=4, b=8;
    total = SquareOfSum(a,b);
    printf("Output=%d",total);
}
```
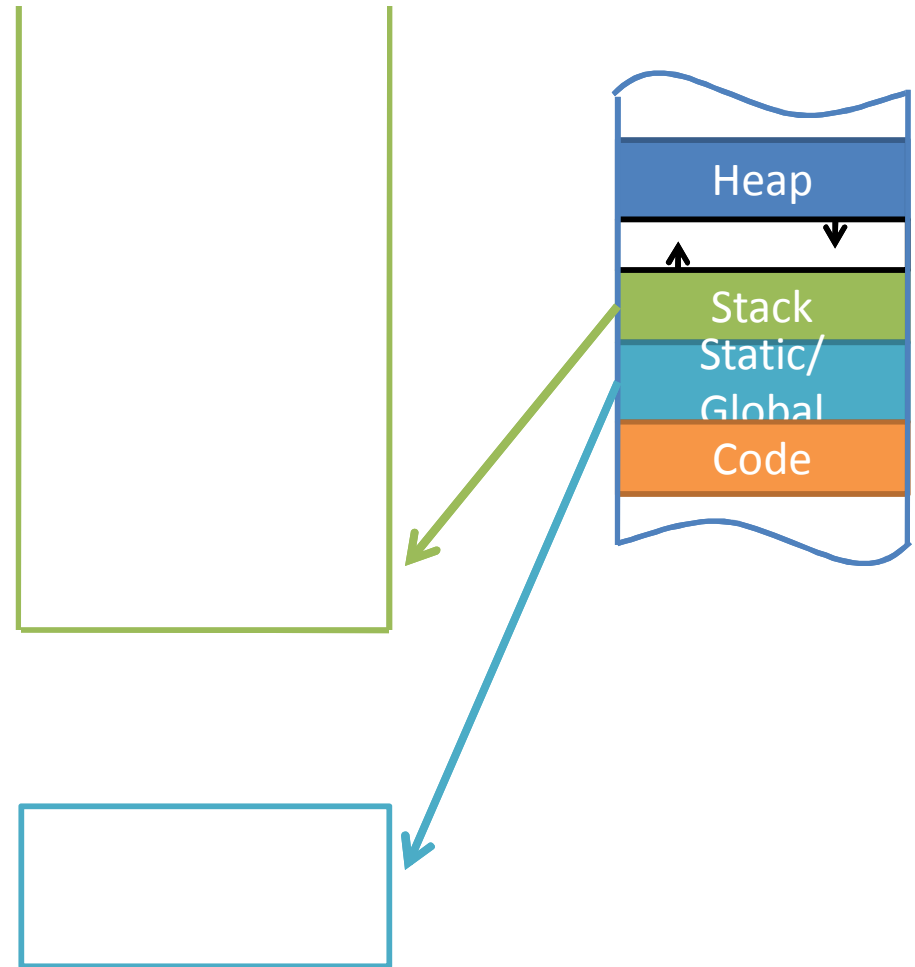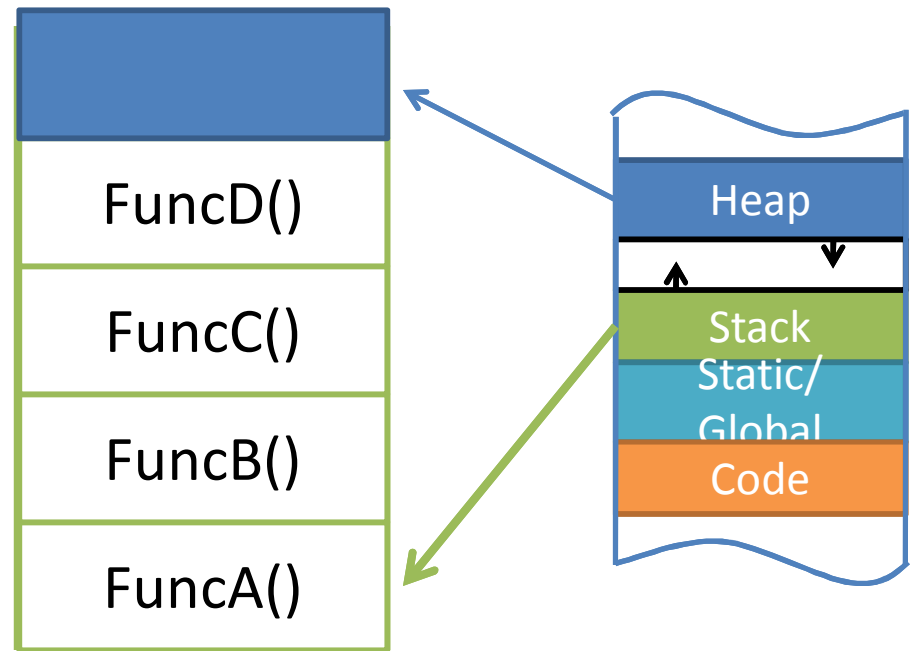
FuncD()

FuncC()

FuncB()

FuncA()

Heap

Stack

Static/ Global

Code

Stack Overflow

Limitation of Stack

• Allocation and deallocation of stack memory are handled by Operating System. Programmer cannot control the lifetime of variables in stack memory.

- **Allocation:** Function starts (Push onto stack)

- **Deallocation:** Function finishes (Popped out of stack)

• Size of the **stack frame** for a function is known at the compile time.

• An array with **unknown size** (only known at run time) cannot be allocated in the stack memory.

- For this, we need to use heap memory.

Heap Memory

- A programmer can define the size of an array at run time using **dynamic memory allocation.**

- All variables/array defined using dynamic memory allocation are allocated memory from heap.

- The programmer can decide how long the allocated memory to be kept.

- Heap can grow as long as the program does not run out of the memory allocated to it.
  - Sometimes dangerous if proper care is not taken.

- Heap is called a free pool/store of memory
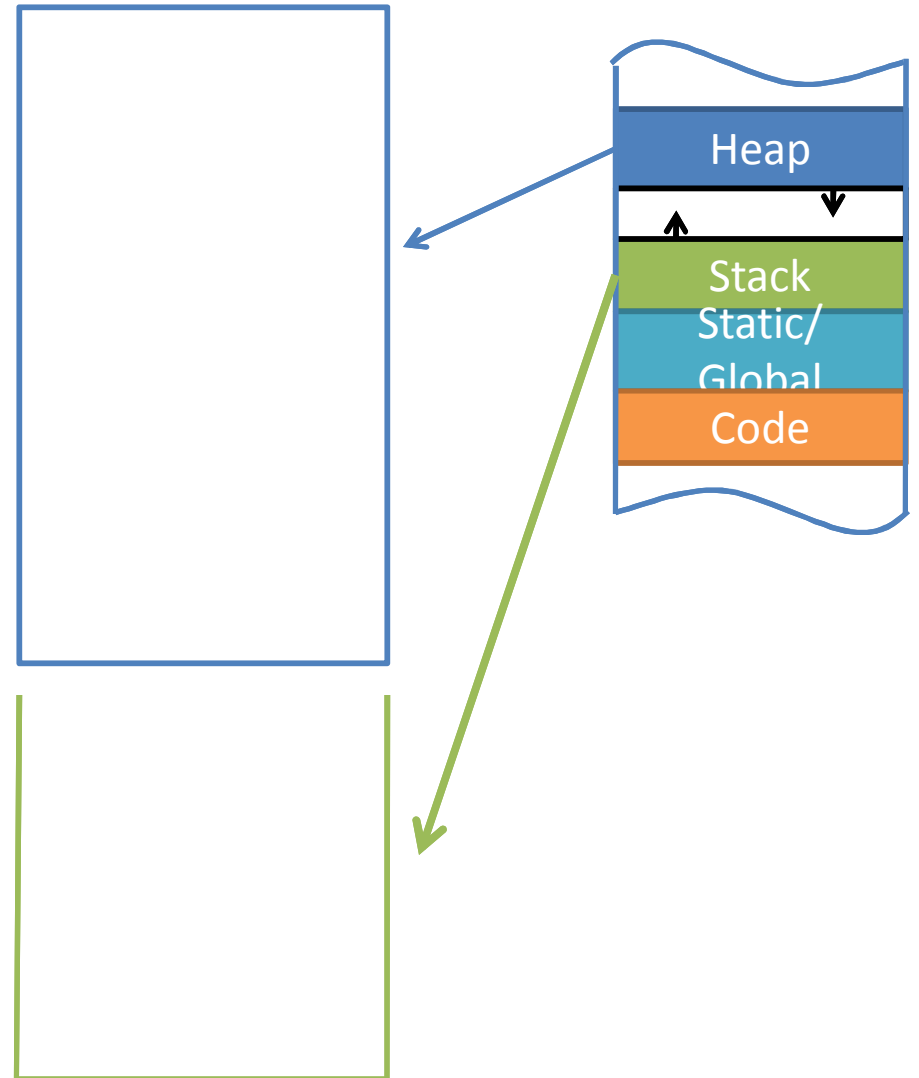
Use of Heap
(Dynamic Memory Allocation)

Dynamic Memory Allocation

C language

- **malloc()** – Allocates block of memory
- **calloc()** – Allocate multiple blocks and initialize to 0.
- **realloc()** – Reallocates block of memory
- **free()** – Frees up memory

C++ language

- **new** – Allocates block of memory
- **delete** – Frees up memory

Heap
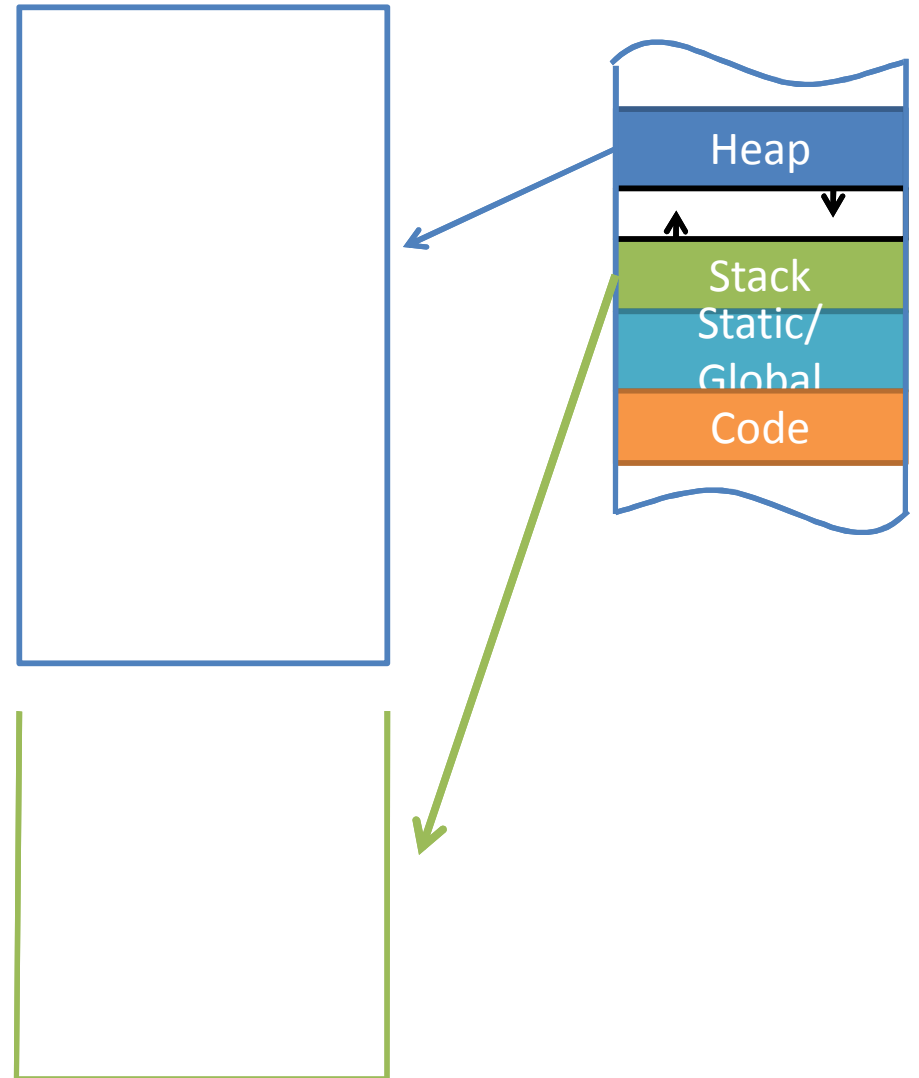
Stack

Static/
Global

Code

Use of Heap
(Dynamic Memory Allocation)

Dynamic Memory Allocation

C language (Example code)

#include<stdio.h>
#include<stdlib.h>

Heap

Stack

Static/
Global

Code

Use of Heap
(Dynamic Memory Allocation)

Dynamic Memory Allocation

C language (Example code)

#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=10;

Heap

Stack

Static/
Global

Code

main()

a   10

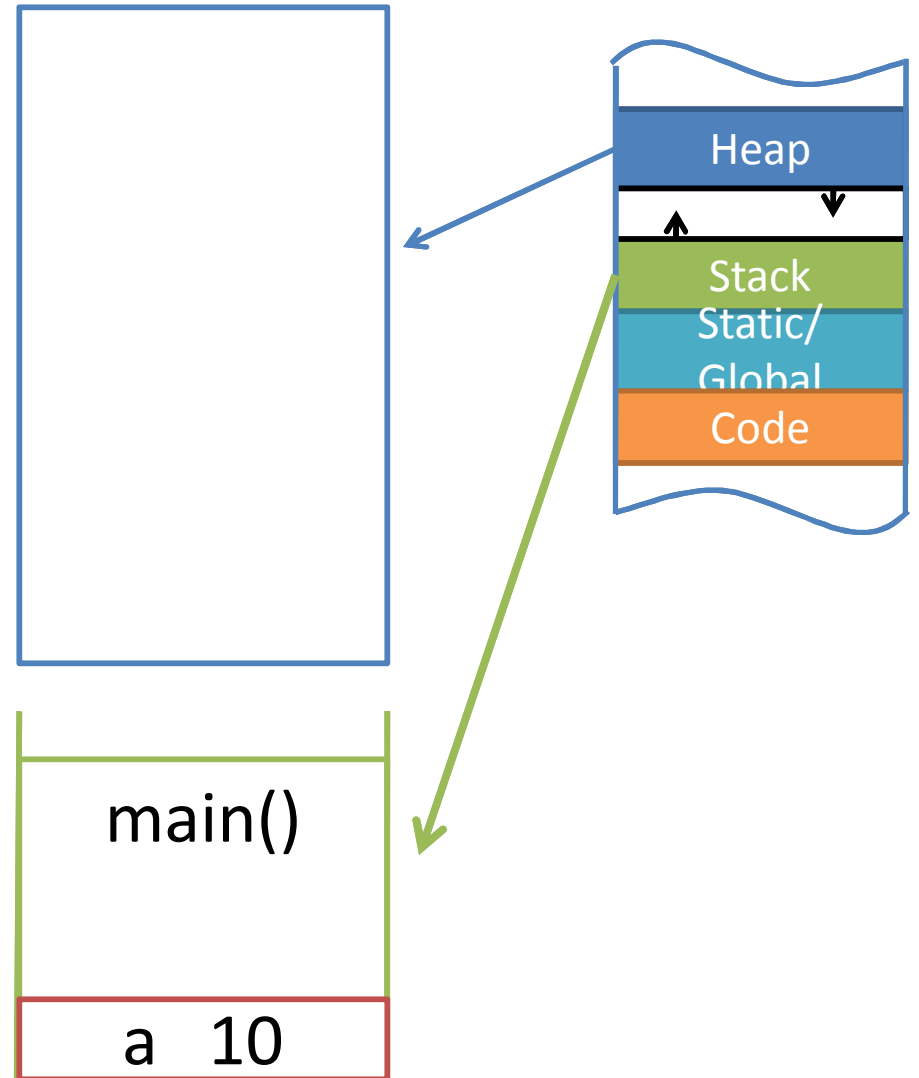# Use of Heap
(Dynamic Memory Allocation)

Dynamic Memory Allocation

<u>**C language (Example code)**</u>

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=10;
    int *p;
    p = (int*)malloc(sizeof(int));
```
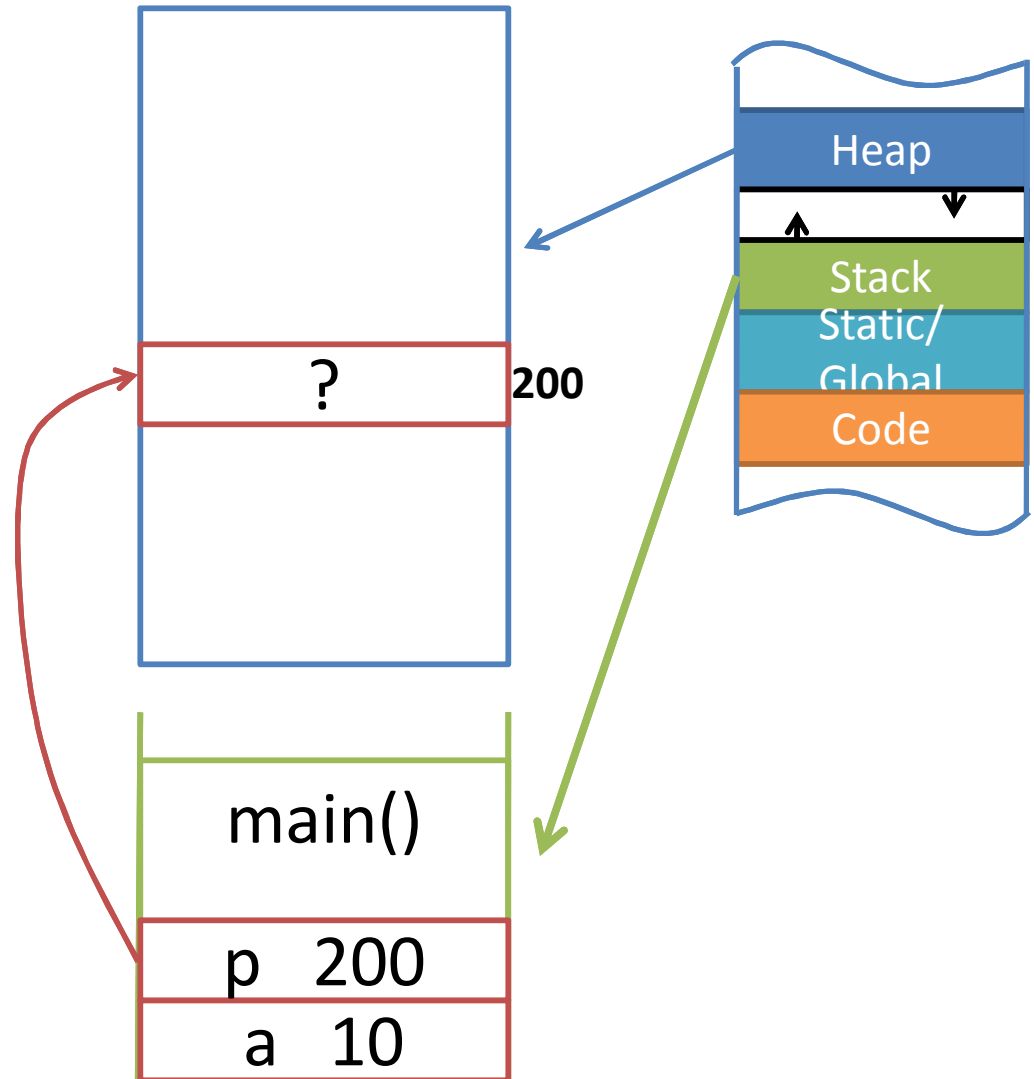
Use of Heap
(Dynamic Memory Allocation)

Dynamic Memory Allocation

C language (Example code)

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=10;
    int *p;
    p = (int*)malloc(sizeof(int));
    *p=15;
```

15   **200**

main()

p   200

a   10

Heap

Stack

Static/
Global

Code

Use of Heap
(Dynamic Memory Allocation)

Dynamic Memory Allocation

C language (Example code)

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=10;
    int *p;
    p = (int*)malloc(sizeof(int));
    *p=15;

    p=(int*)malloc(sizeof(int));
    *p=20;
```
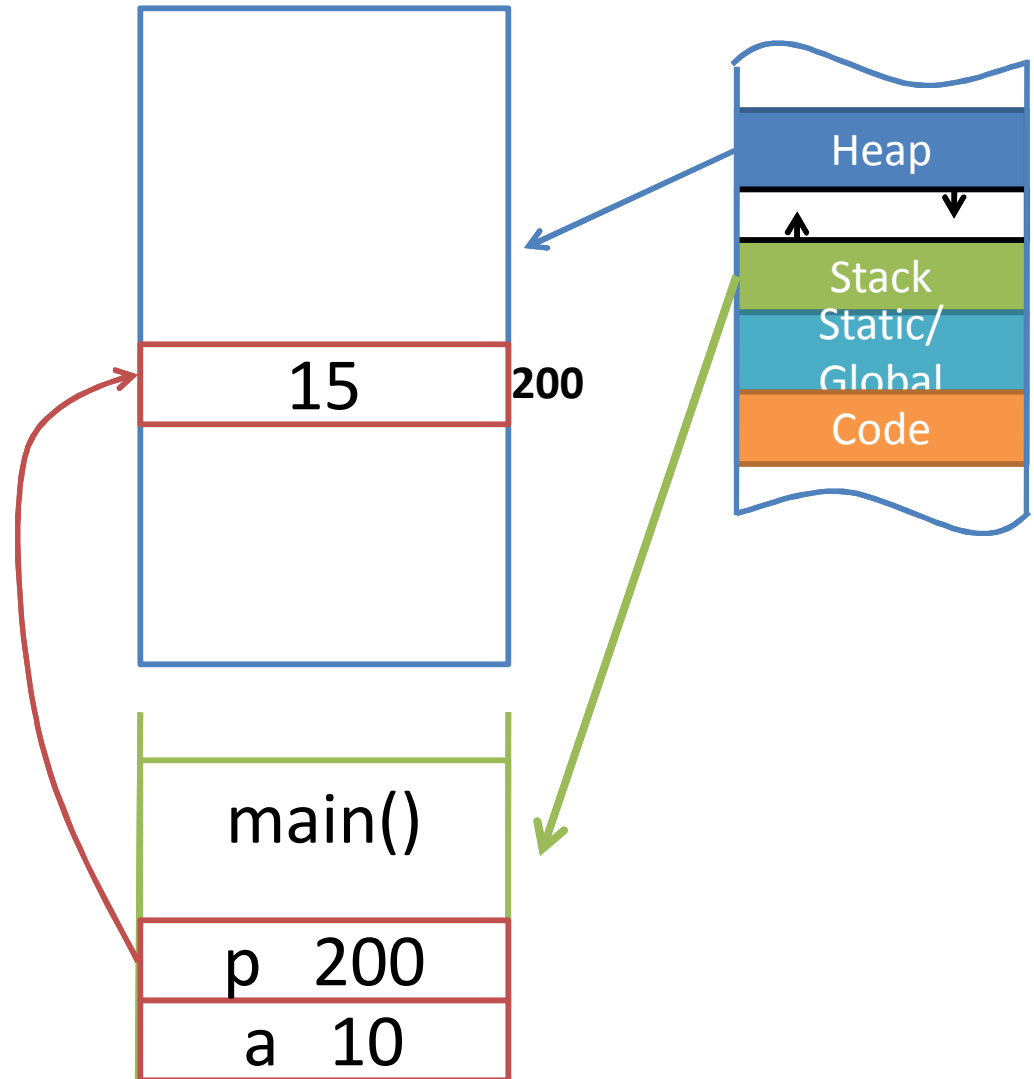
20   100

15   200

Memory Leak

main()

p   100

a   10

Heap

Stack

Static/Global

Code

Use of Heap
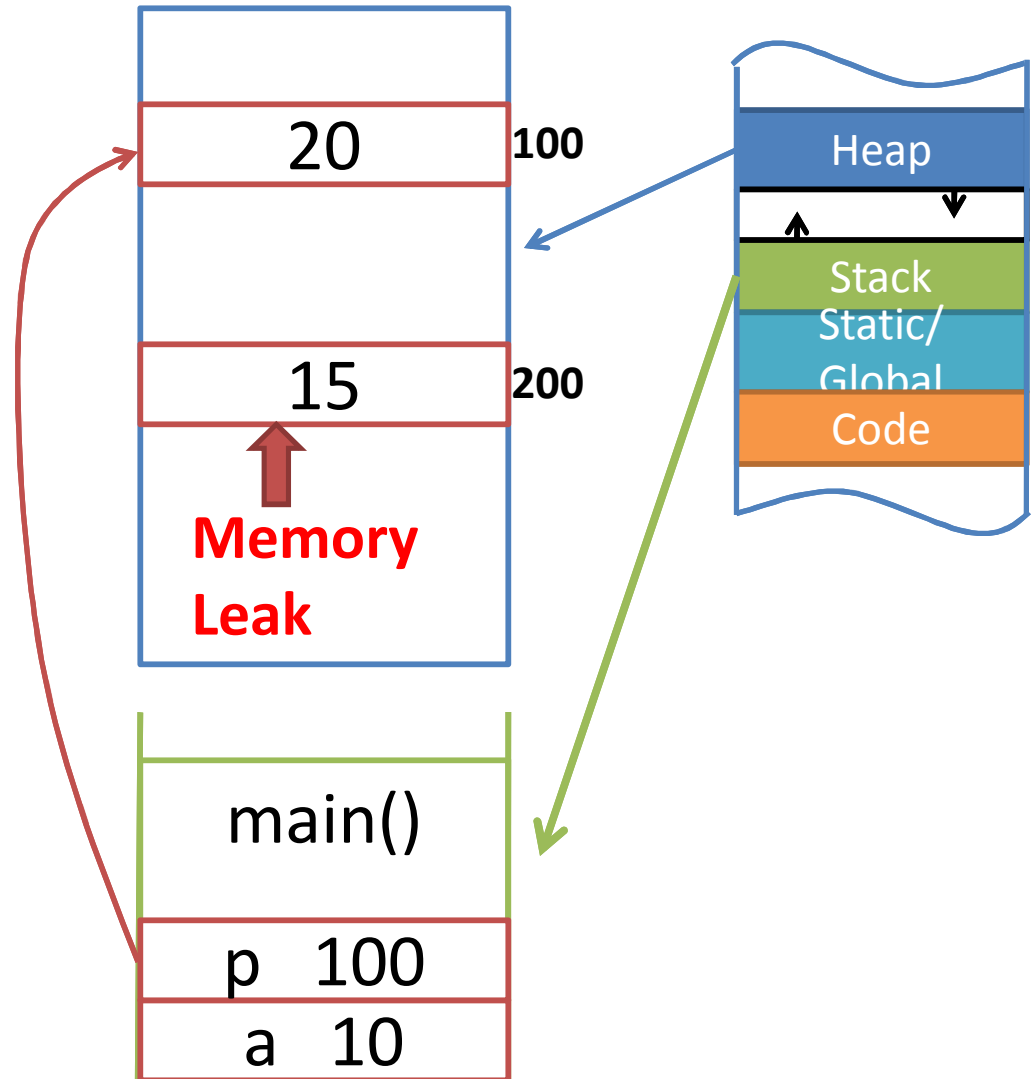# (Dynamic Memory Allocation)

Dynamic Memory Allocation

C language (Example code)

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=10;
    int *p;
    p = (int*)malloc(sizeof(int));
    *p=15;
    free(p);
    p=(int*)malloc(sizeof(int));
    *p=20;
}
```
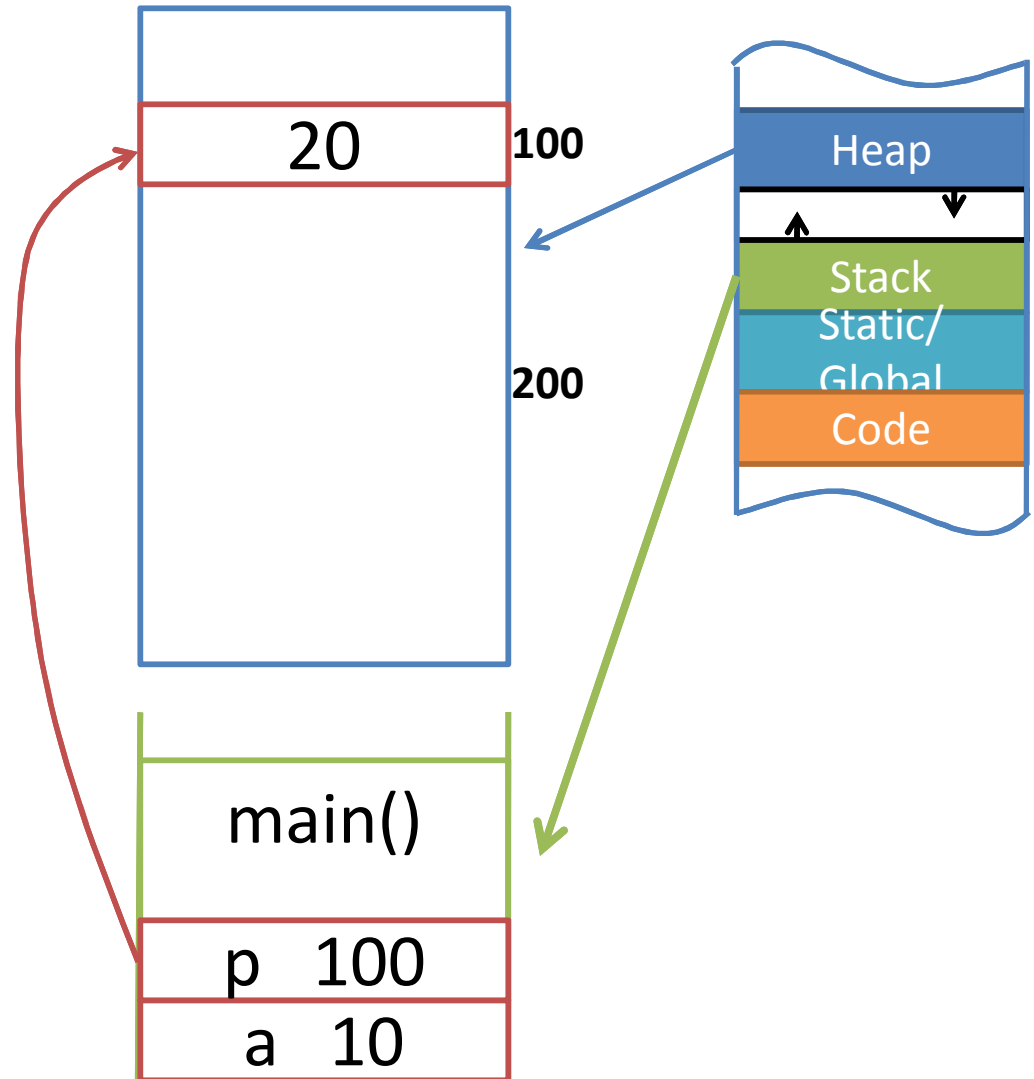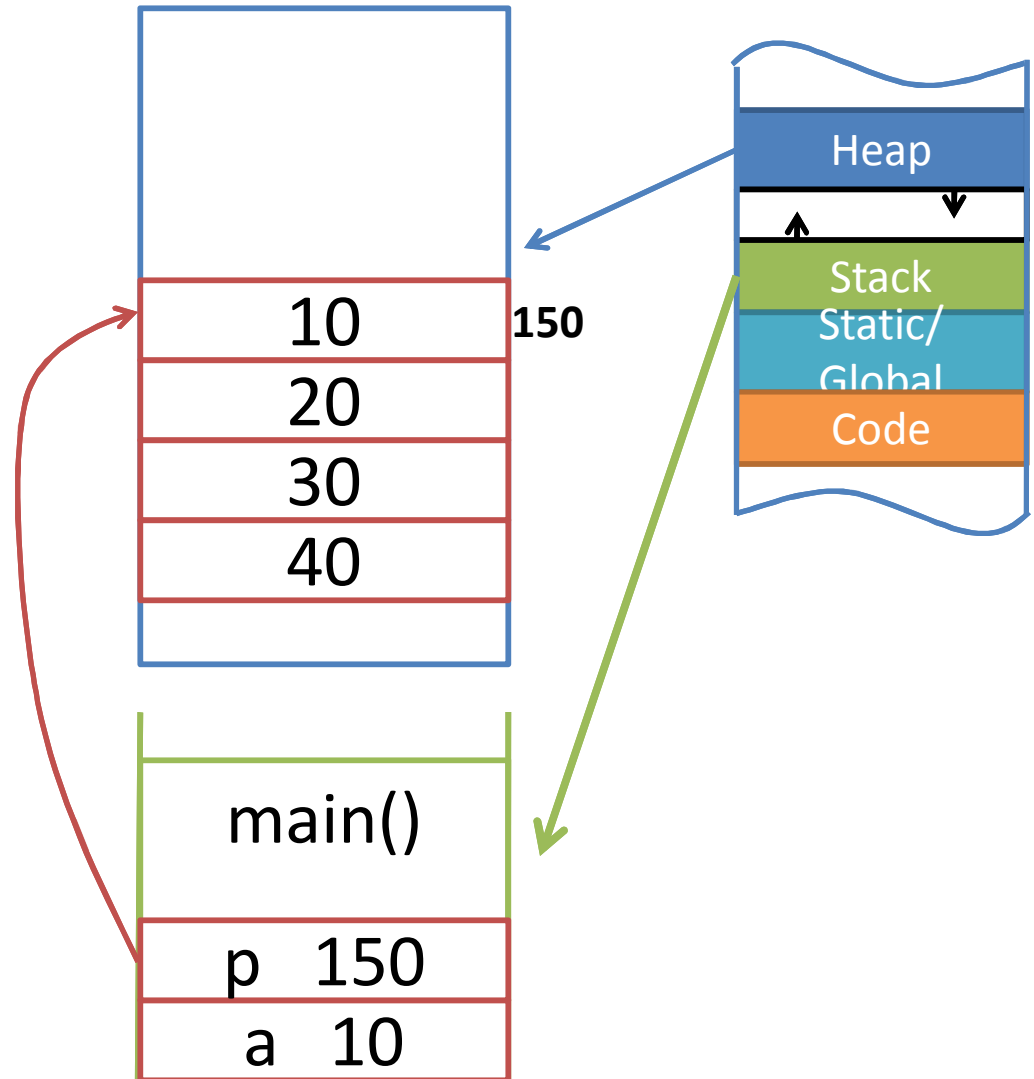
20   100

                      200

Heap

Stack

Static/
Global

Code

main()

p   100

a   10

Use of Heap
(Dynamic Memory Allocation)

Dynamic Memory Allocation

C language (Example code)

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=10;
    int *p;
    p = (int*)malloc(4*sizeof(int));
    p[0]=10;
    p[1]=20;
    *(p+2)=30;
    *(p+3)=40;
}
```

Use of Heap
(Dynamic Memory Allocation)

Dynamic Memory Allocation

C++ language (Example code)

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=10;
    int *p;
    p = new int;
    *p=10;
```
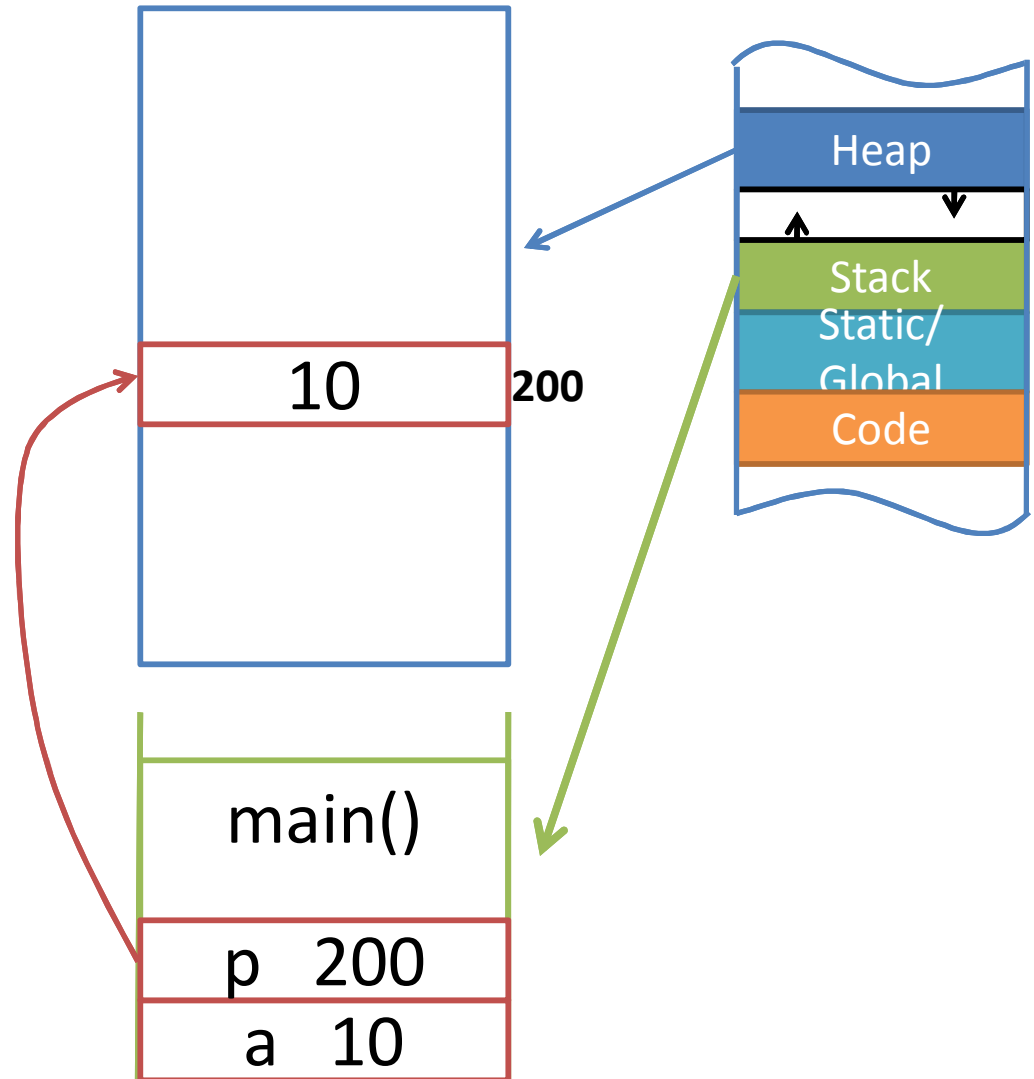
Use of Heap
(Dynamic Memory Allocation)

Dynamic Memory Allocation

C++ language (Example code)

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=10;
    int *p;
    p = new int;
    *p=10;
    delete p;
```

200

Heap

Stack

Static/
Global

Code

main()

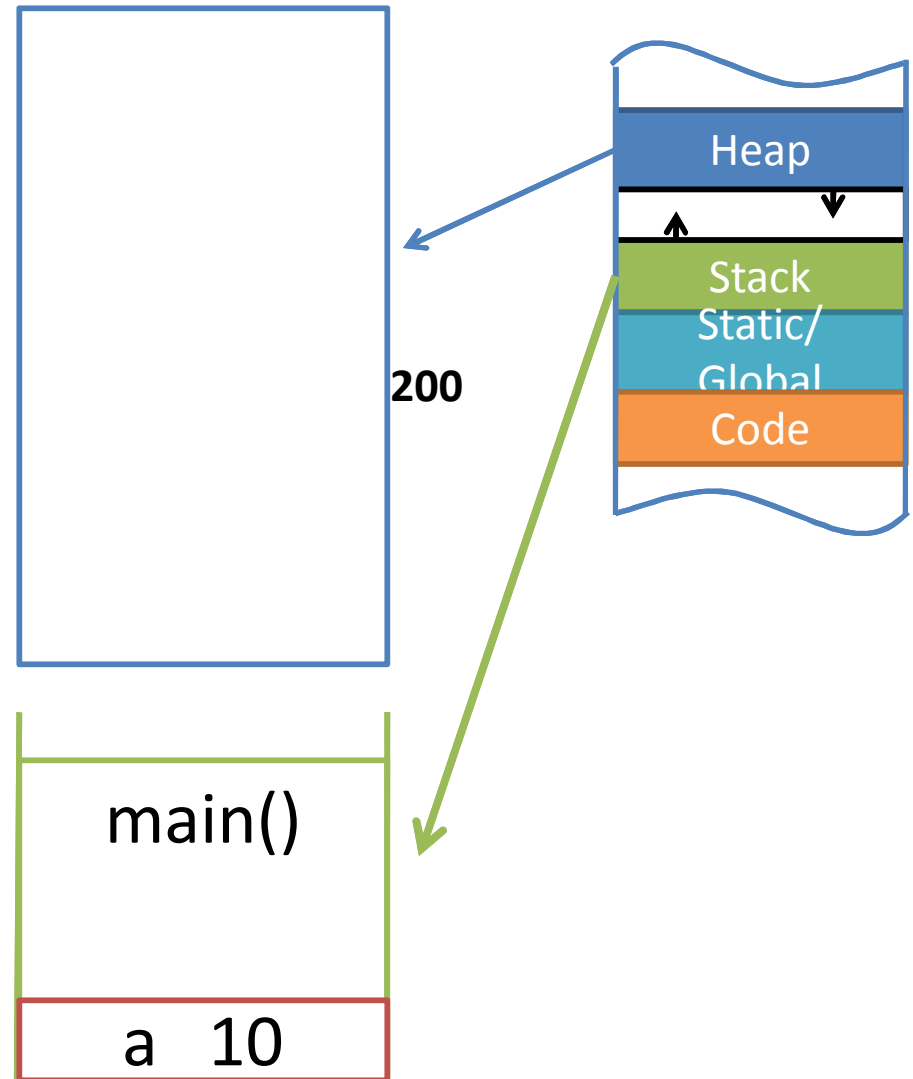a   10

Use of Heap
(Dynamic Memory Allocation)

Dynamic Memory Allocation

<u>C++ language (Example code)</u>

```cpp
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=10;
    int *p;
    p = new int;
    *p=10;
    delete p;
    p = new int[4];
    for(int i=0;i<4;i++)
        p[i]=0;
```

0   50

0

0

0

Heap

Stack

Static/
Global

Code

main()

p   50

a   10
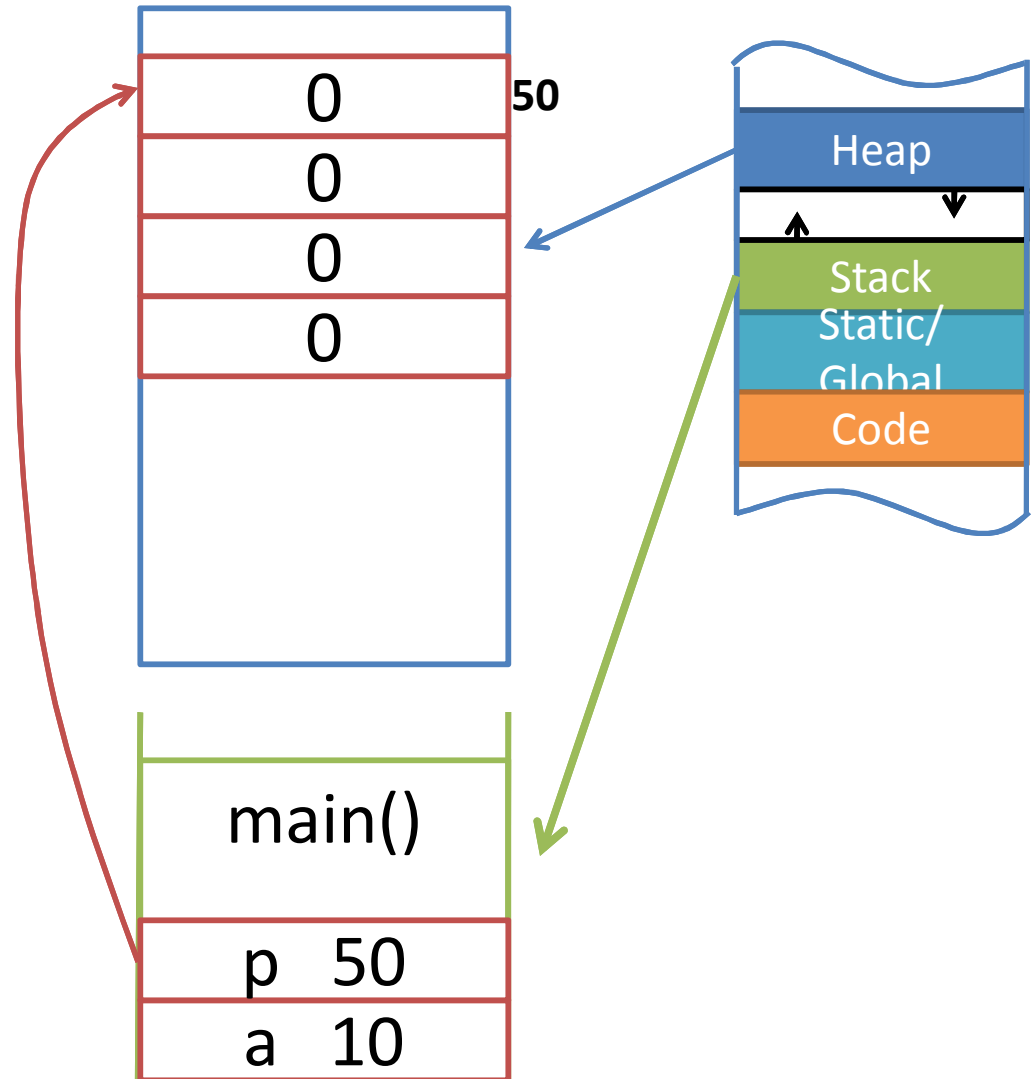
Use of Heap
(Dynamic Memory Allocation)

Dynamic Memory Allocation

C++ language (Example code)

```cpp
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=10;
    int *p;
    p = new int;
    *p=10;
    delete p;
    p = new int[4];
    for(int i=0;i<4;i++)
      p[i]=0;
    delete[ ] p;
}
```
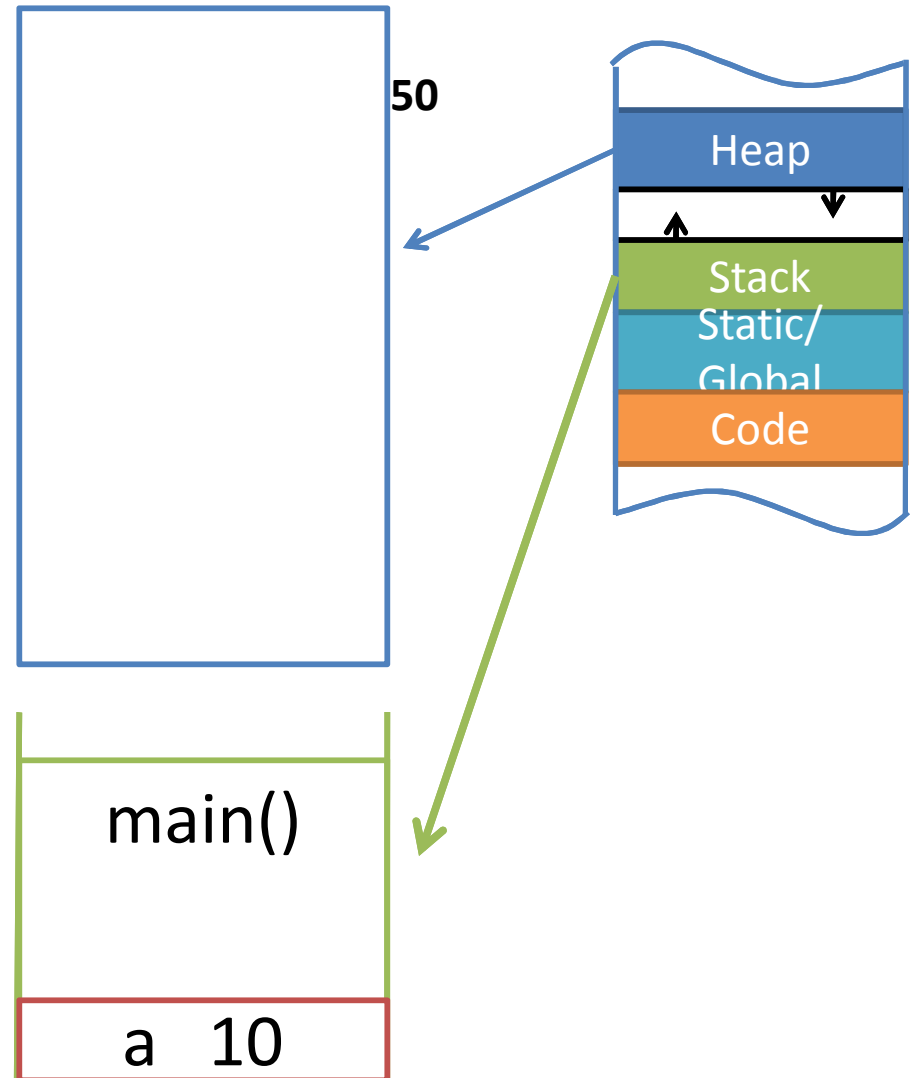
50

Heap

Stack

Static/
Global

Code

main()

a   10

Dynamic 2D Array Allocation in C
Using single pointer

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

int main()
{
    int r = 3, c = 4;
    int *arr = (int *)malloc(r * c * sizeof(int));

    int i, j, count = 0;
    for (i = 0; i < r; i++)
    for (j = 0; j < c; j++)
        *(arr + i*c + j) = ++count;

    for (i = 0; i < r; i++)
    for (j = 0; j < c; j++)
        printf("%d ", *(arr + i*c + j));

/* Code for further processing and free the
   dynamically allocated memory */

return 0;
}
```
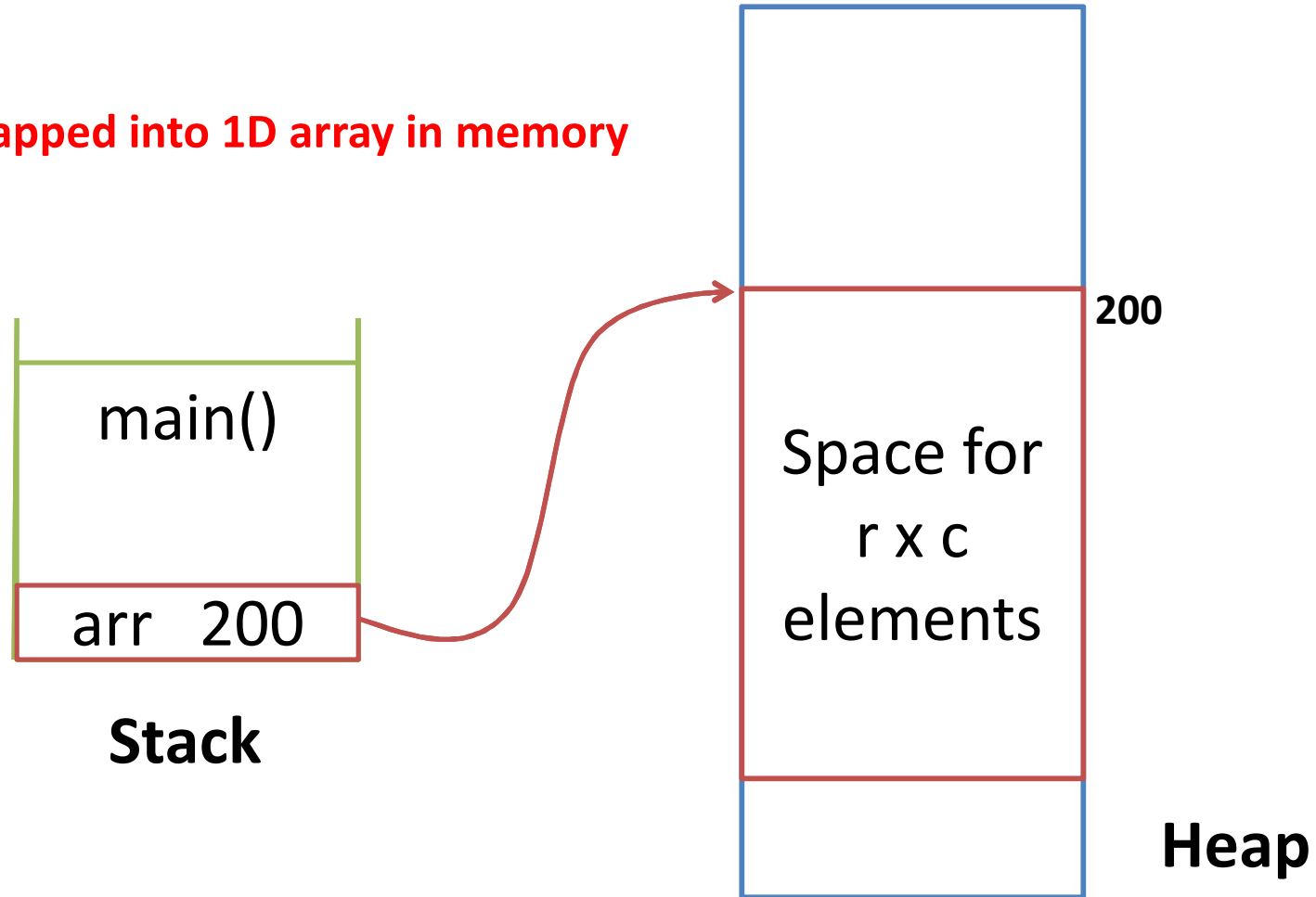
Dynamic 2D Array Allocation in C
Using single pointer

2D array mapped into 1D array in memory

main()

arr   200

Stack

Space for
r x c
elements

200

Heap

(i,j)th Element: *(arr+i*c+j) or arr[i*c+j]

Dynamic 2D Array Allocation in C
Using array of pointers

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

int main()
{
    int r = 3, c = 4, i, j, count;

    int **arr = (int **)malloc(r * sizeof(int *));
    for (i=0; i<r; i++)
        arr[i] = (int *)malloc(c * sizeof(int));

    // Note that arr[i][j] is same as *(*(arr+i)+j)
    count = 0;
    for (i = 0; i < r; i++)
    for (j = 0; j < c; j++)
        arr[i][j] = ++count; // OR *(*(arr+i)+j) = ++count

    for (i = 0; i < r; i++)
    for (j = 0; j < c; j++)
        printf("%d ", arr[i][j]);

/* Code for further processing and free the
    dynamically allocated memory */

    return 0;
}
```
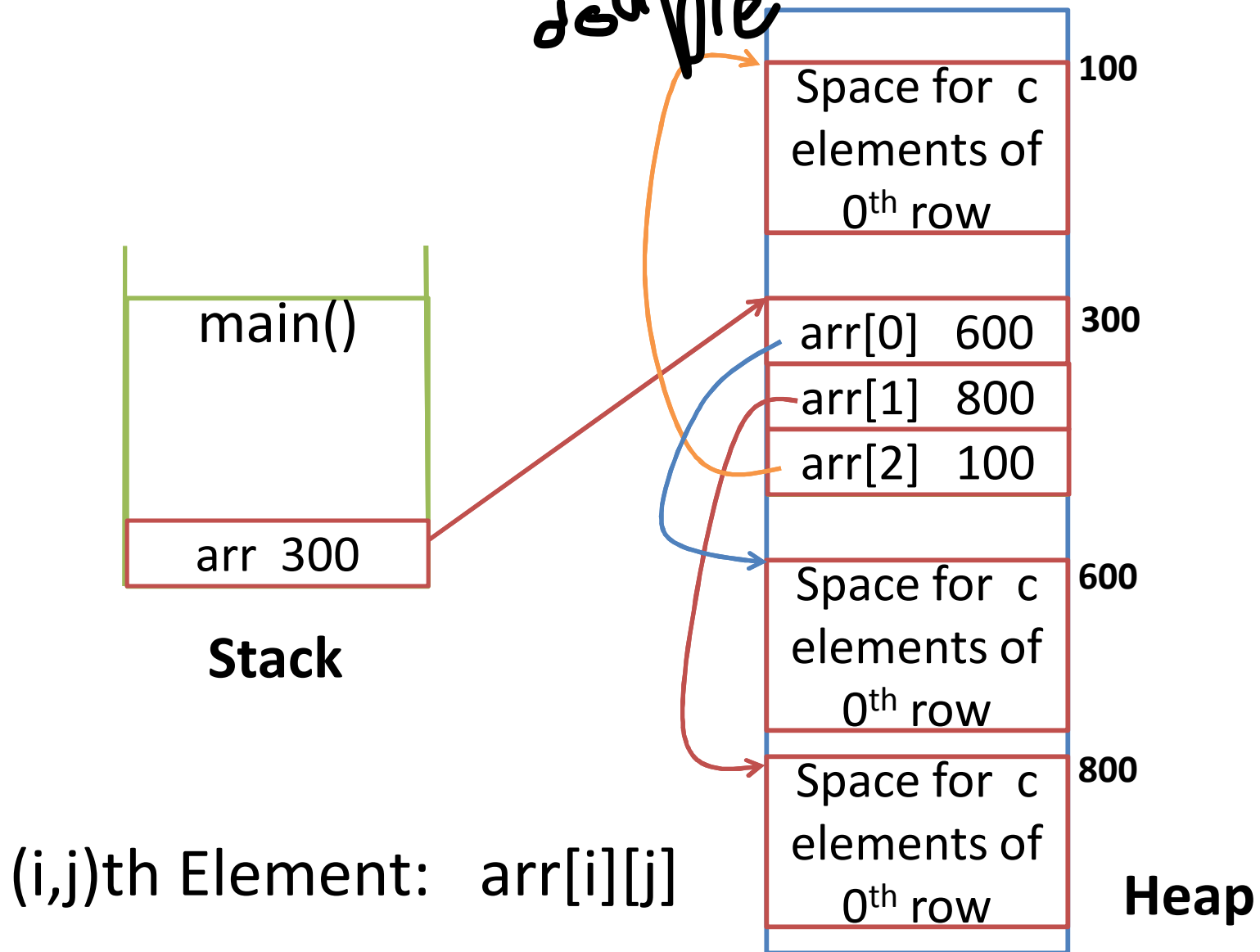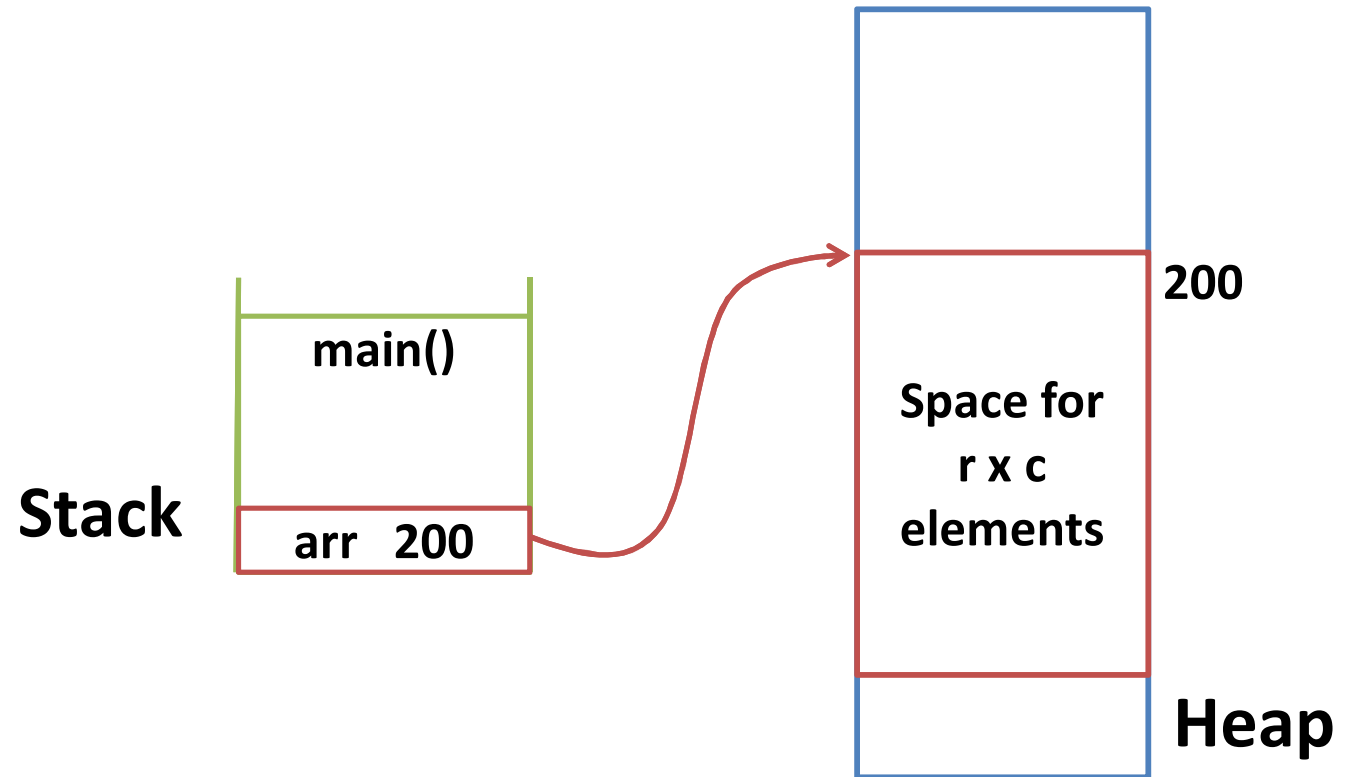
Dynamic 2D Array Allocation in C
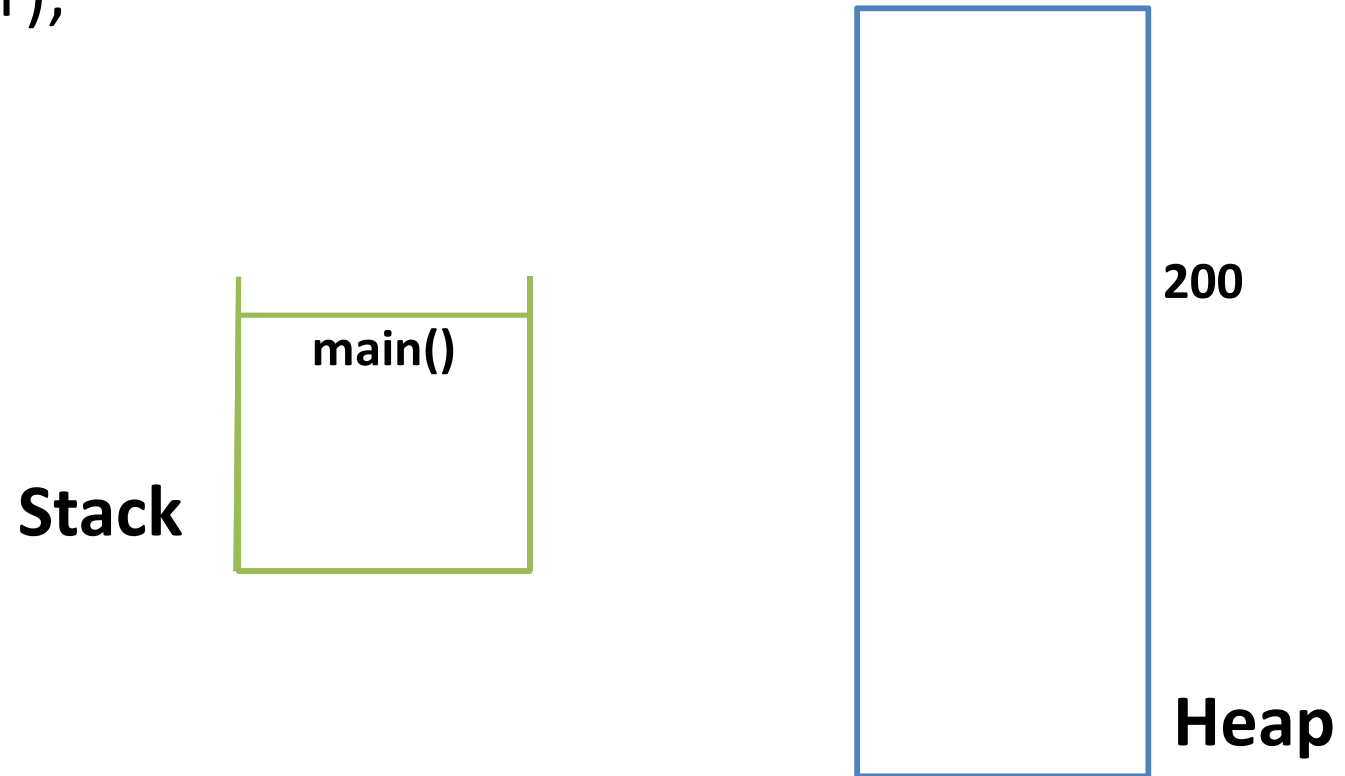## Using ~~single~~ double pointer

Stack

main()	
arr 300	

Heap

Space for c elements of 0th row	100
arr[0] 600	300
arr[1] 800	
arr[2] 100	
Space for c elements of 0th row	600
Space for c elements of 0th row	800

(i,j)th Element: arr[i][j]

Freeing Dynamic 2D Array in C

- **For single pointer:**

Stack

main()

arr   200

Space for r x c elements

200

Heap

Freeing Dynamic 2D Array in C

- **For single pointer:**
   free(arr);
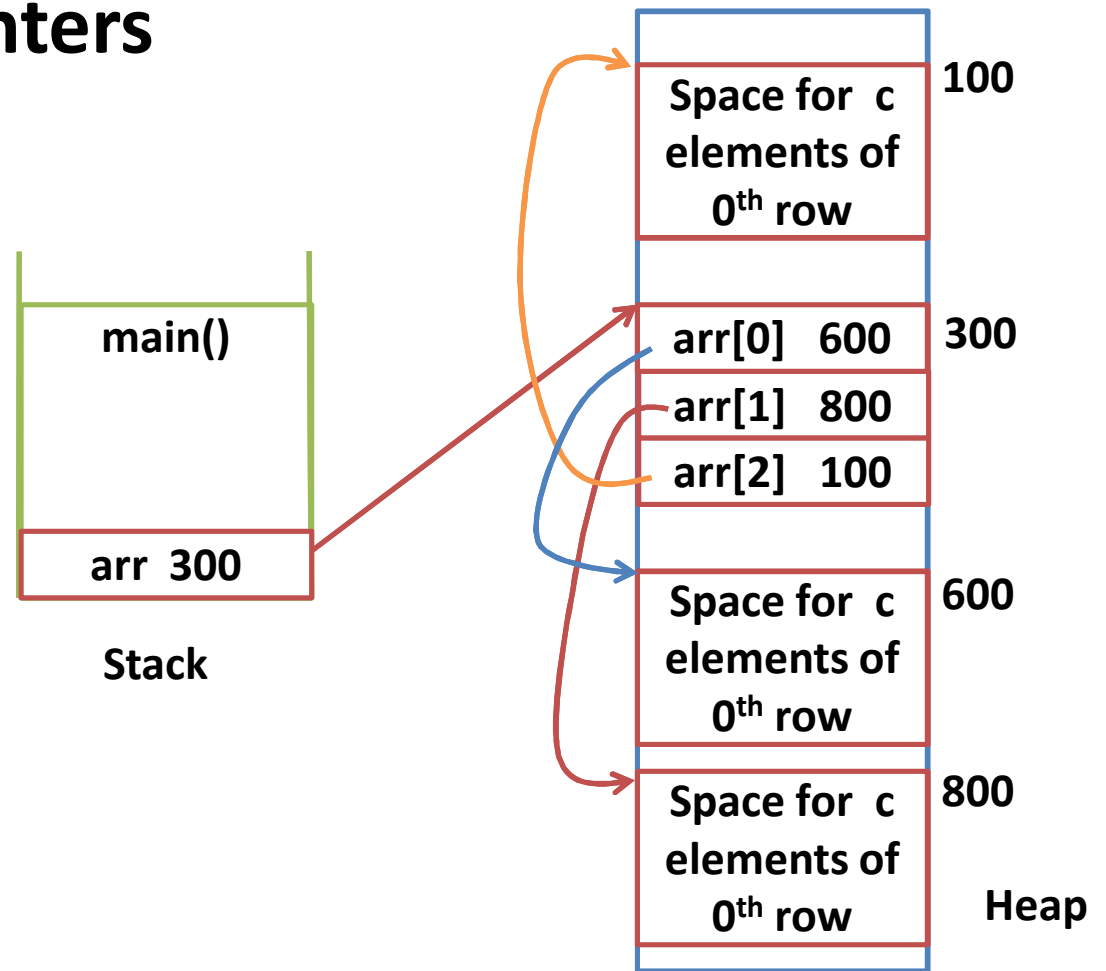
200

main()

Stack

Heap

Freeing Dynamic 2D Array in C

- **For array of pointers**

Space for c elements of 0th row — 100

arr[0] 600 — 300
arr[1] 800
arr[2] 100

main()

arr 300

Stack

Space for c elements of 0th row — 600

Space for c elements of 0th row — 800

Heap

Freeing Dynamic 2D Array in C

- **For array of pointers**

  for(i=0;i<r;i++){

     free(arr[i]);}

main()

arr 300

Stack

100

300

600

800

Heap
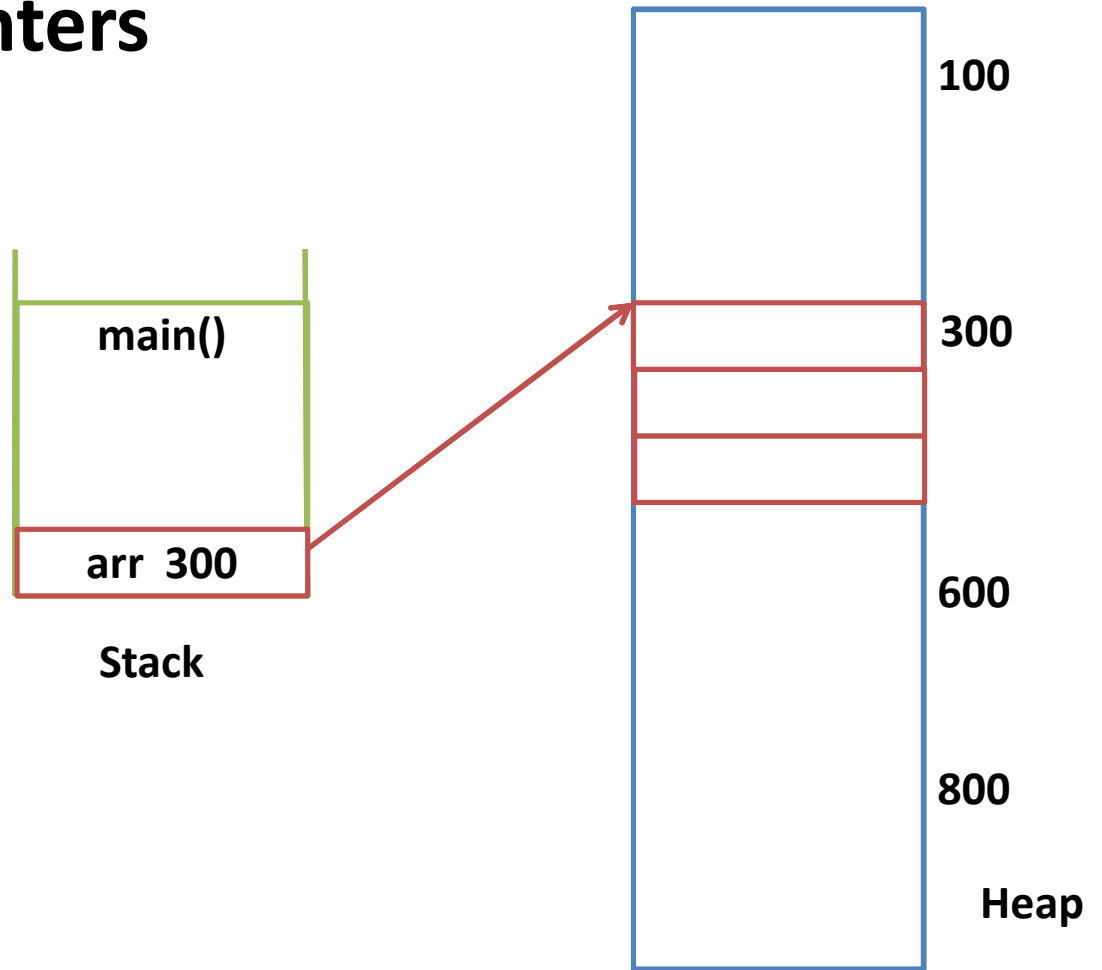
Freeing Dynamic 2D Array in C

- **For array of pointers**

  for(i=0;i<r;i++){

     free(arr[i]);}

  free(arr);

main()

Stack

100

300

600

800

Heap
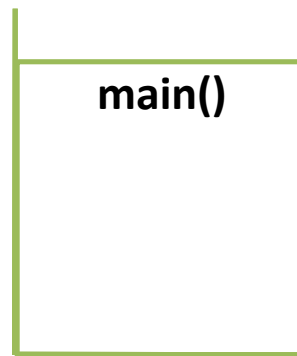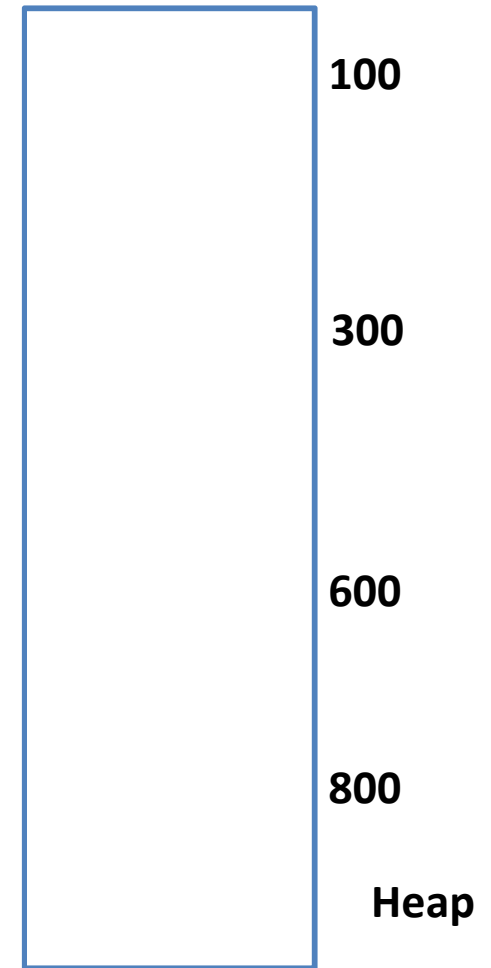
Dynamic 2D Array Allocation and Freeing in C++ Using single pointer

int r=3, c=4;

int** arr = new int[r*c];

(i,j)th element can be accessed by *(arr + i*c+j) or arr[i*c+j]

For freeing memory:
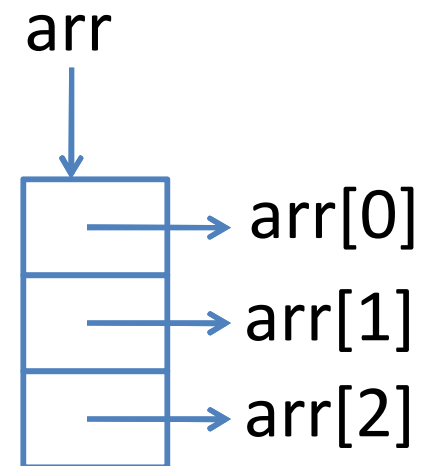
delete[] arr;

Dynamic 2D Array Allocation and Freeing in C++
Using array of pointers

```
int r=3, c=4;
int** arr = new int*[r];
for(i=0;i<r;i++)
    arr[i] = new int[c];
```
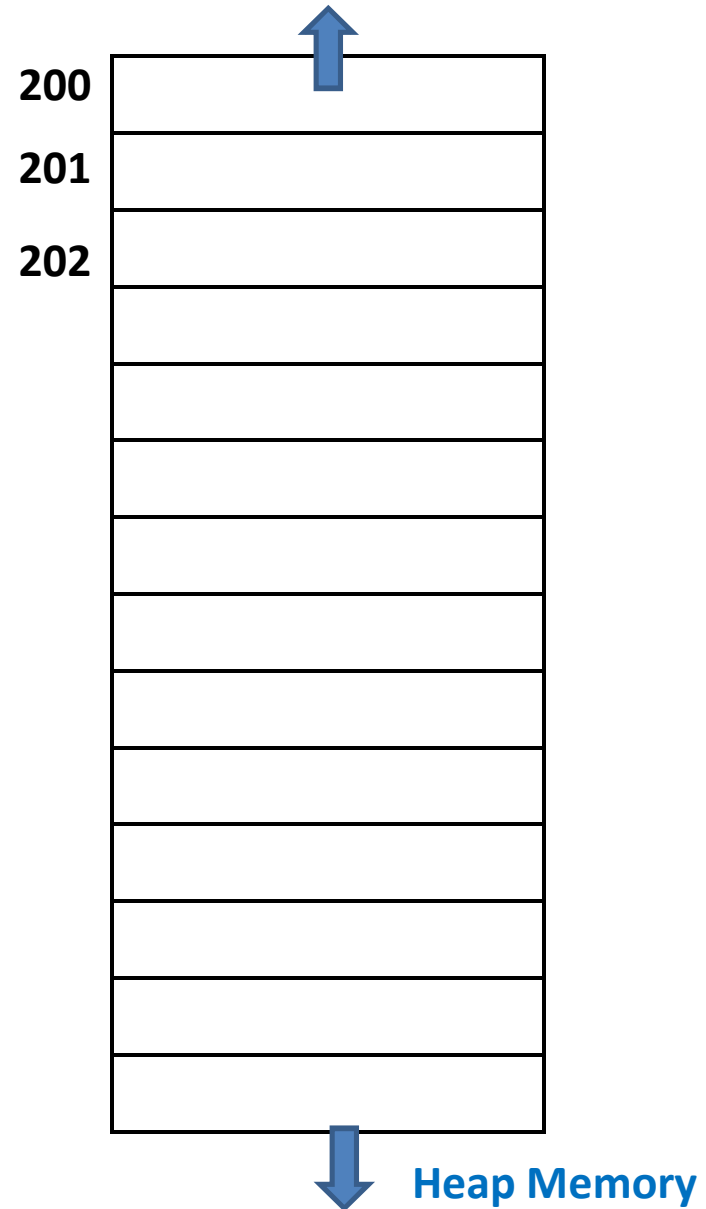
(i,j)th element can be accessed by arr[i][j];

For freeing memory:

```
for(i=0;i<r;i++){
    delete[] arr[i]);}

delete[] arr;
```

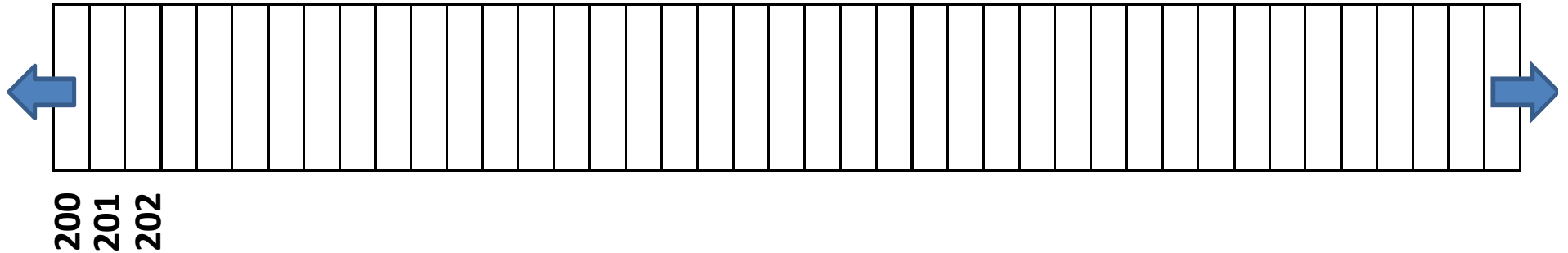arr

arr[0]

arr[1]

arr[2]

Limitations of Dynamic Array

200

201

202

Heap Memory

Limitations of Dynamic Array

Heap Memory (Horizontal View)

200 201 202

Limitations of Dynamic Array

Heap Memory (Horizontal View)

```
200 201 202
```
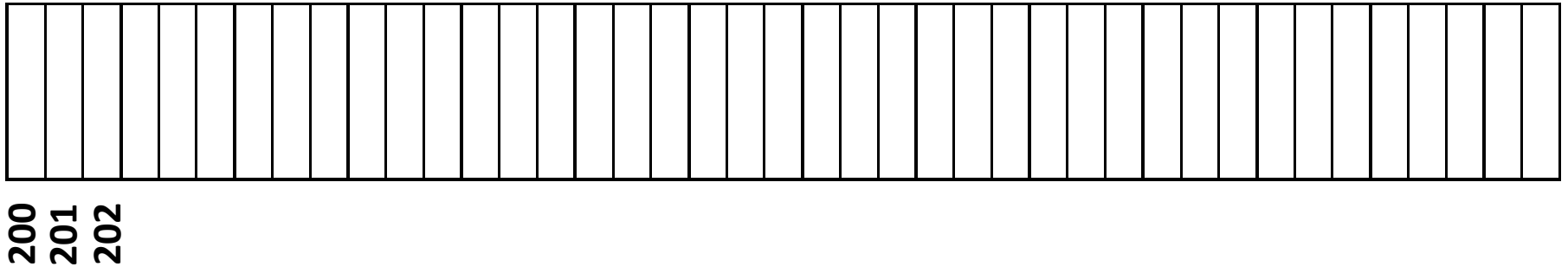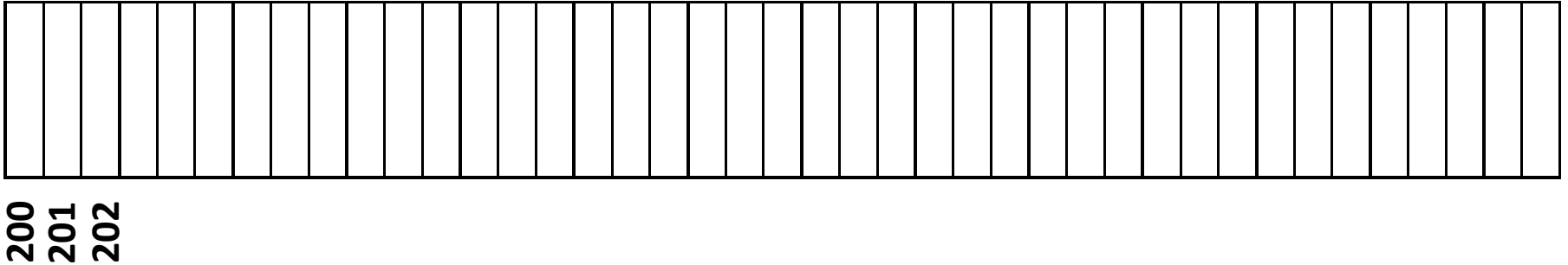
- Memory Manager (A part of operating systems) manages the memory.

  - Keeps track of free space

  - Allocates space on request from the programs

Limitations of Dynamic Array

Heap Memory (Horizontal View)
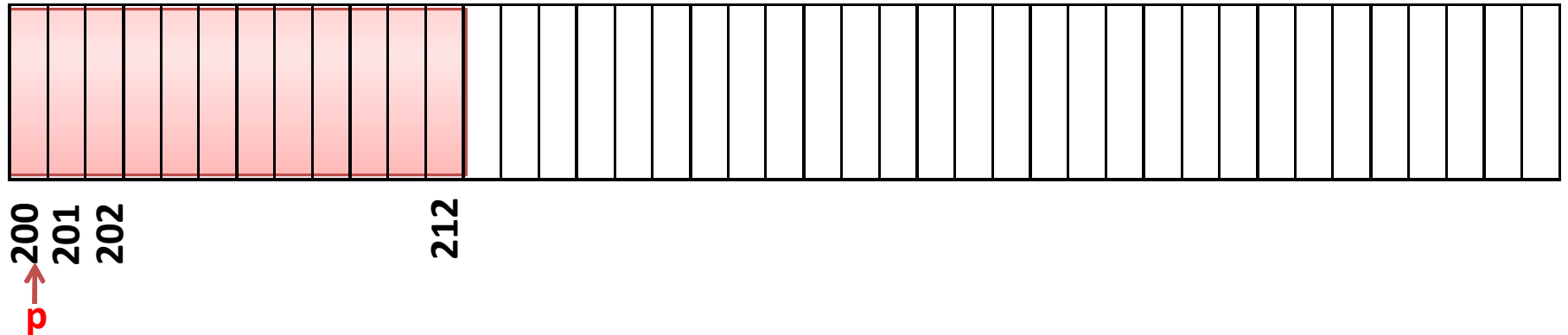
200 201 202

Consider a program segment:

int *p=(int*)malloc(3*sizeof(int));

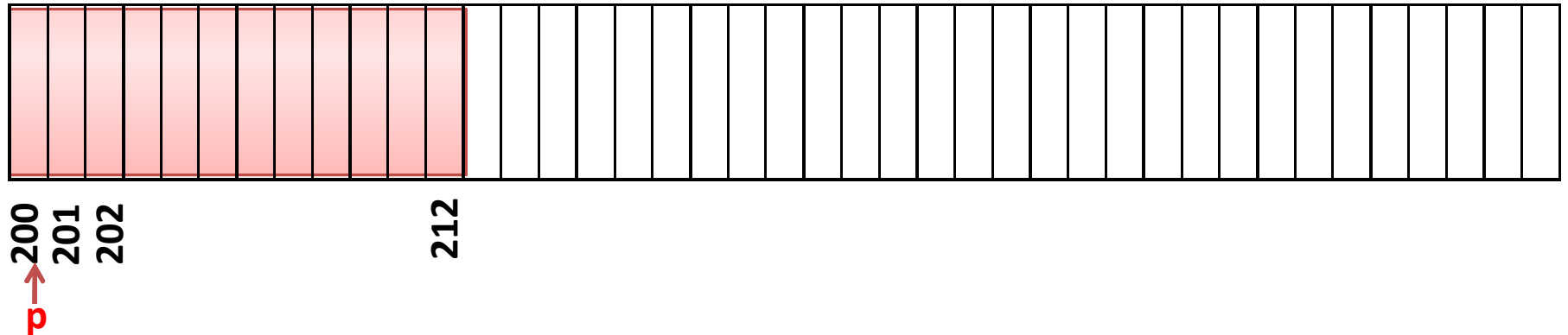# Limitations of Dynamic Array

200
201
202

212

p

Consider a program segment:

int *p=(int*)malloc(3*sizeof(int));   // int size is 4 bytes

Limitations of Dynamic Array

Heap Memory (Horizontal View)

200
201
202

212

↑
p

Consider a program segment:

int *p=(int*)malloc(3*sizeof(int));   // int size is 4 bytes
float *q=(float*)malloc(sizeof(float));   // float size is 8 bytes

Limitations of Dynamic Array

Heap Memory (Horizontal View)

Consider a program segment:

int *p=(int*)malloc(3*sizeof(int));   // int size is 4 bytes
float *q=(float*)malloc(sizeof(float));   // float size is 8 bytes

Limitations of Dynamic Array

Heap Memory (Horizontal View)

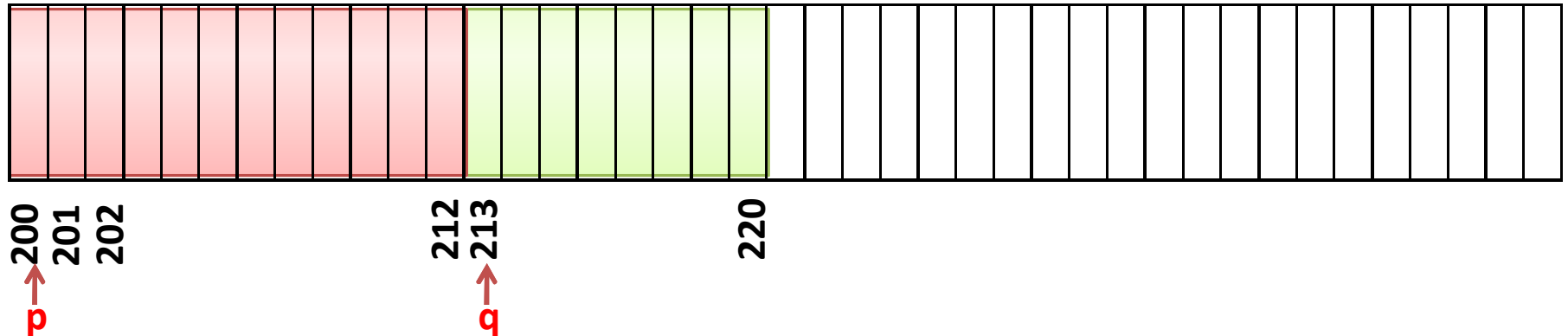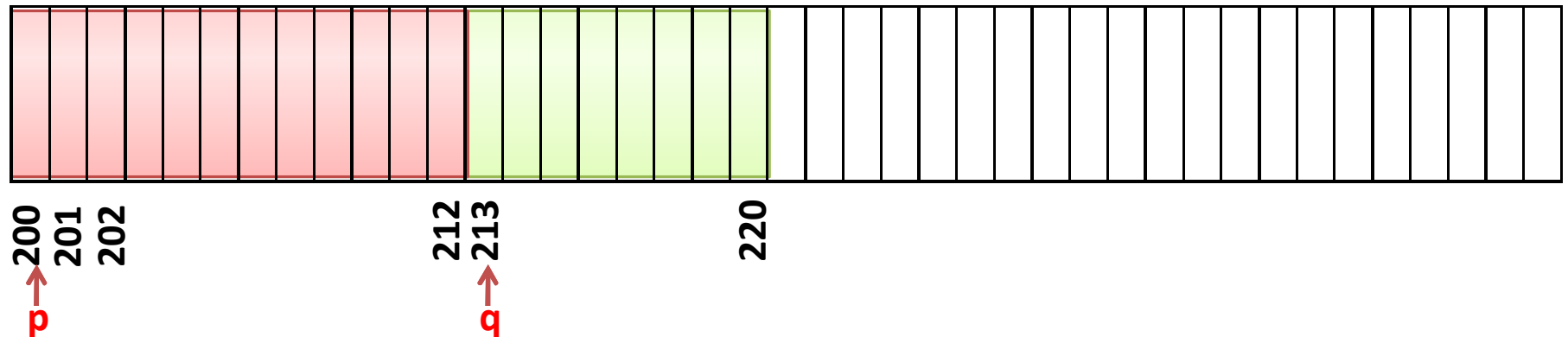**Consider a program segment:**

int *p=(int*)malloc(3*sizeof(int));   // int size is 4 bytes
float *q=(float*)malloc(sizeof(float));   // float size is 8 bytes
int *r=(int*)malloc(4*sizeof(int)); // int size is 4 bytes

Limitations of Dynamic Array

Consider a program segment:

int *p=(int*)malloc(3*sizeof(int));   // int size is 4 bytes
float *q=(float*)malloc(sizeof(float));   // float size is 8 bytes
int *r=(int*)malloc(4*sizeof(int)); // int size is 4 bytes

Limitations of Dynamic Array

Heap Memory (Horizontal View)
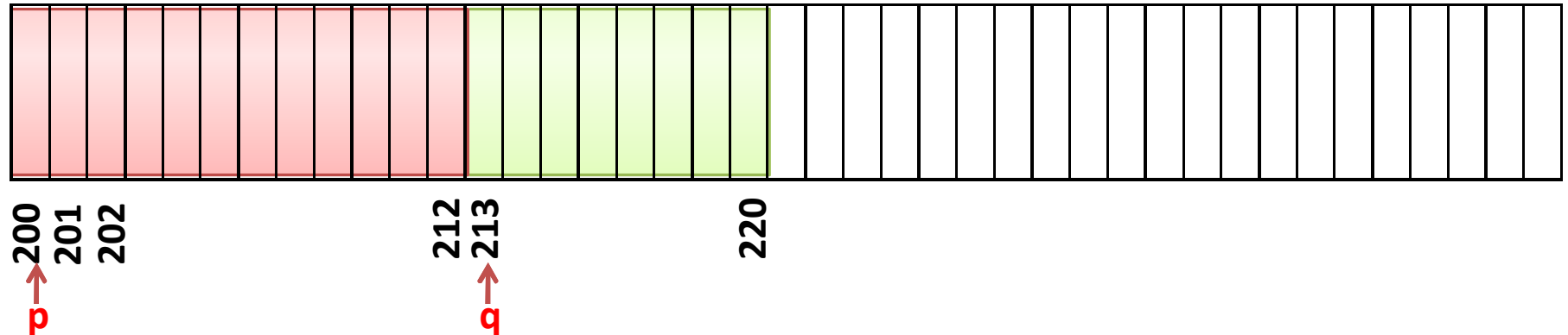
200
201
202
212
213
220
221
236

p

q

r

Consider a program segment:

int *p=(int*)malloc(3*sizeof(int));   // int size is 4 bytes
float *q=(float*)malloc(sizeof(float));   // float size is 8 bytes
int *r=(int*)malloc(4*sizeof(int)); // int size is 4 bytes

Limitations of Dynamic Array

Heap Memory (Horizontal View)

200
201
202

212
213

220
221

236

p

q

r

Consider a program segment:

int *p=(int*)malloc(3*sizeof(int));   // int size is 4 bytes
float *q=(float*)malloc(sizeof(float));   // float size is 8 bytes
int *r=(int*)malloc(4*sizeof(int)); // int size is 4 bytes
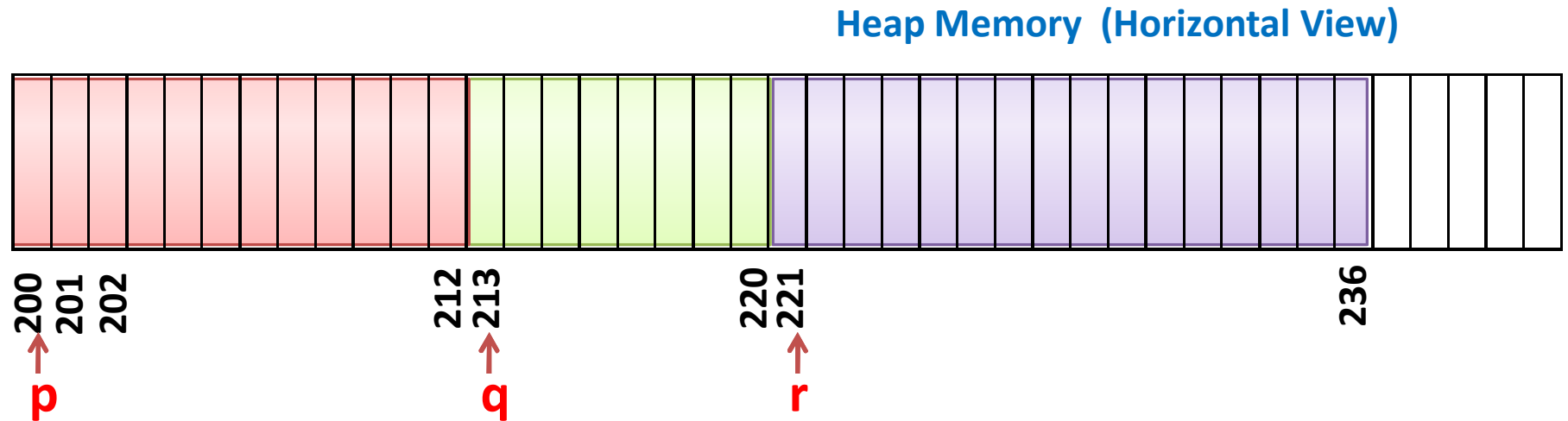free(q)

Limitations of Dynamic Array

Heap Memory (Horizontal View)

200
201
202

212
213

220
221

236

p

r

Consider a program segment:

int *p=(int*)malloc(3*sizeof(int));   // int size is 4 bytes
float *q=(float*)malloc(sizeof(float));   // float size is 8 bytes
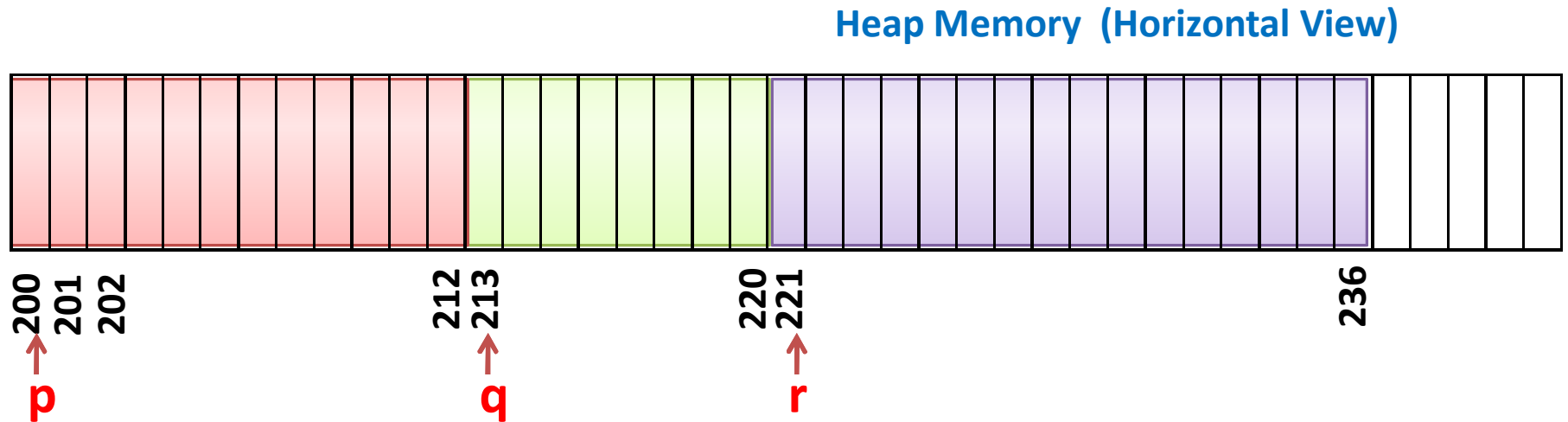int *r=(int*)malloc(4*sizeof(int)); // int size is 4 bytes
free(q)

Limitations of Dynamic Array

Heap Memory (Horizontal View)

200 201 202 ... **212 213** ... **220 221** ... **236**

p   **r**

Consider a program segment:

int *p=(int*)malloc(3*sizeof(int));   // int size is 4 bytes
float *q=(float*)malloc(sizeof(float));   // float size is 8 bytes
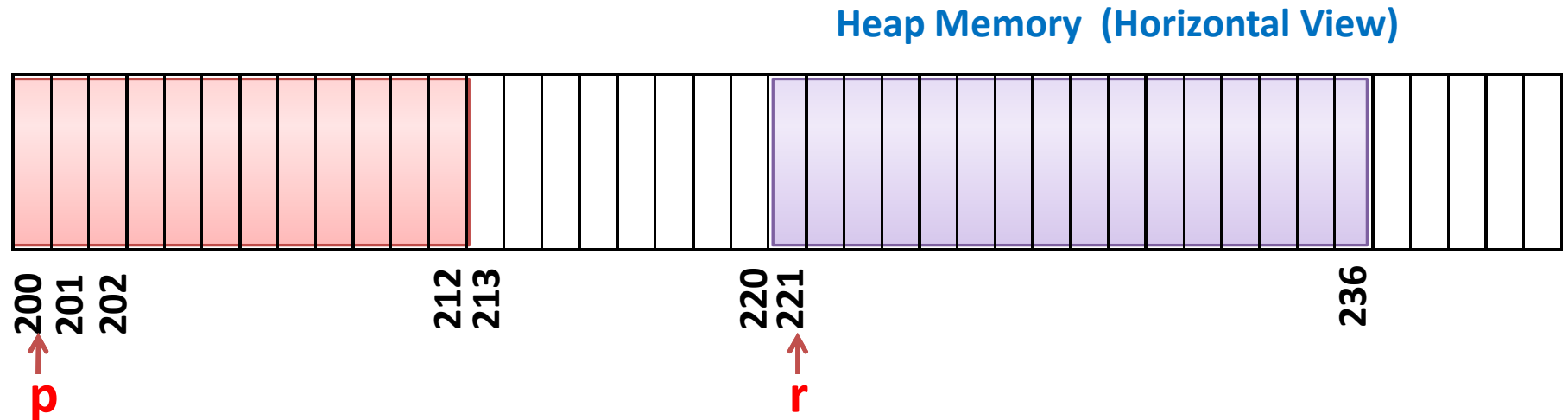int *r=(int*)malloc(4*sizeof(int)); // int size is 4 bytes
free(q)
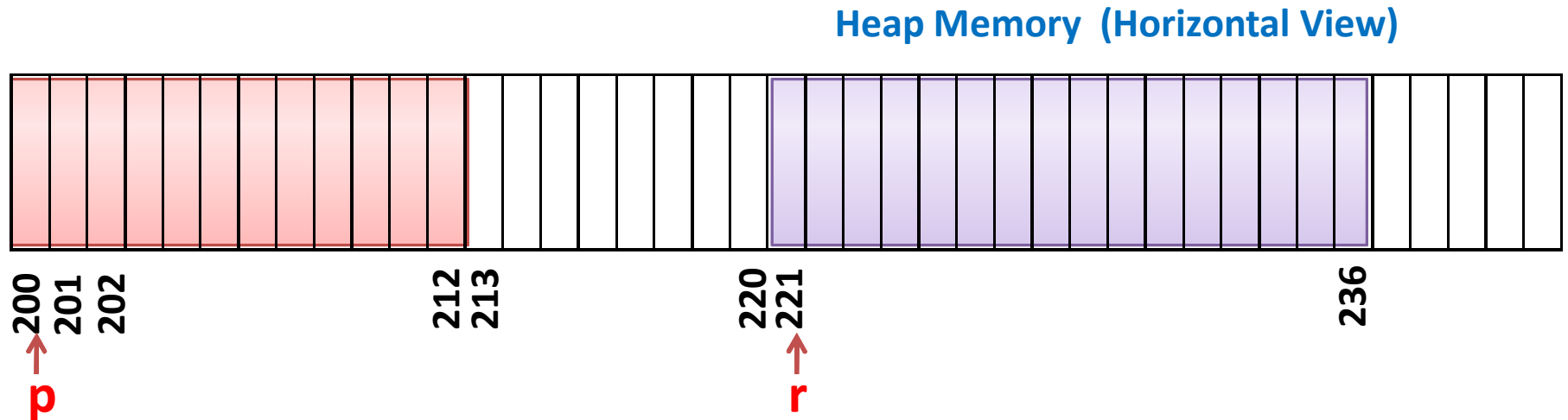int *s=(int*)malloc(3*sizeof(int));   // int size is 4 bytes

Although free memory is more than required 3 x 4 = 12 bytes, yet it cannot be allocated as it is not contiguous. Returns a null pointer.

Solution: Linked List

200 201 202   212 213   220 221   236
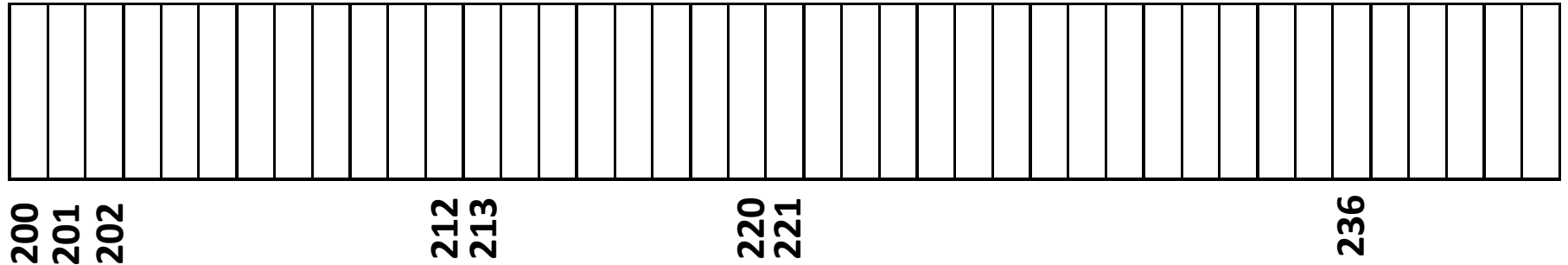
Suppose we need to store a list of 4 integers: 7, 10, 5, 9.

Instead of asking memory manager for an integer array of 4 elements, these 4 integers can be stored one at a time in memory in different places.

Solution: Linked List

Heap Memory (Horizontal View)

200 201 202   212 213   220 221   236

p
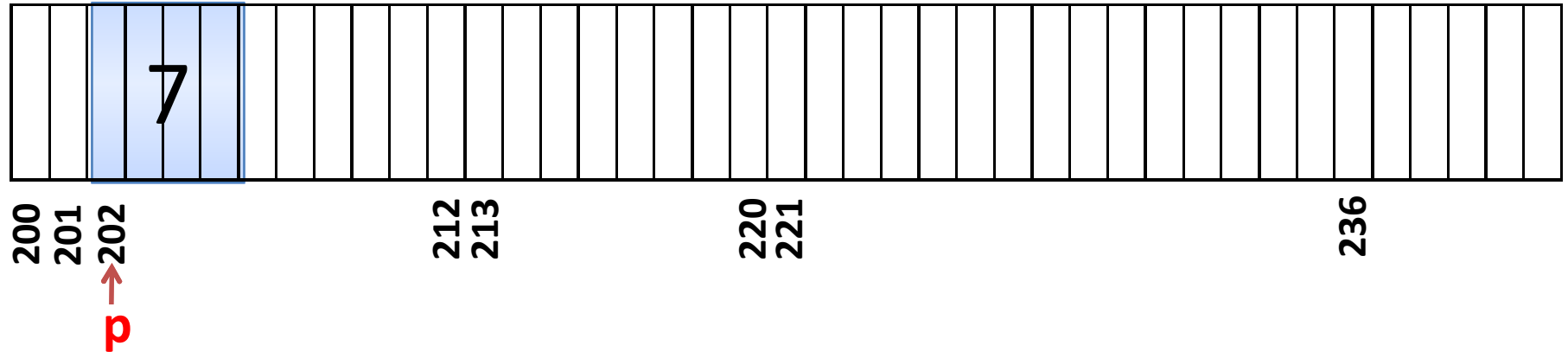
Suppose we need to store a list of 4 integers: 7, 10, 5, 9.

int *p=(int*)malloc(sizeof(int));   *p=7;

Solution: Linked List

Heap Memory (Horizontal View)

200 201 202   212 213   222   236

p   **q**

Suppose we need to store a list of 4 integers: 7, 10, 5, 9.

int *p=(int*)malloc(sizeof(int));   *p=7;
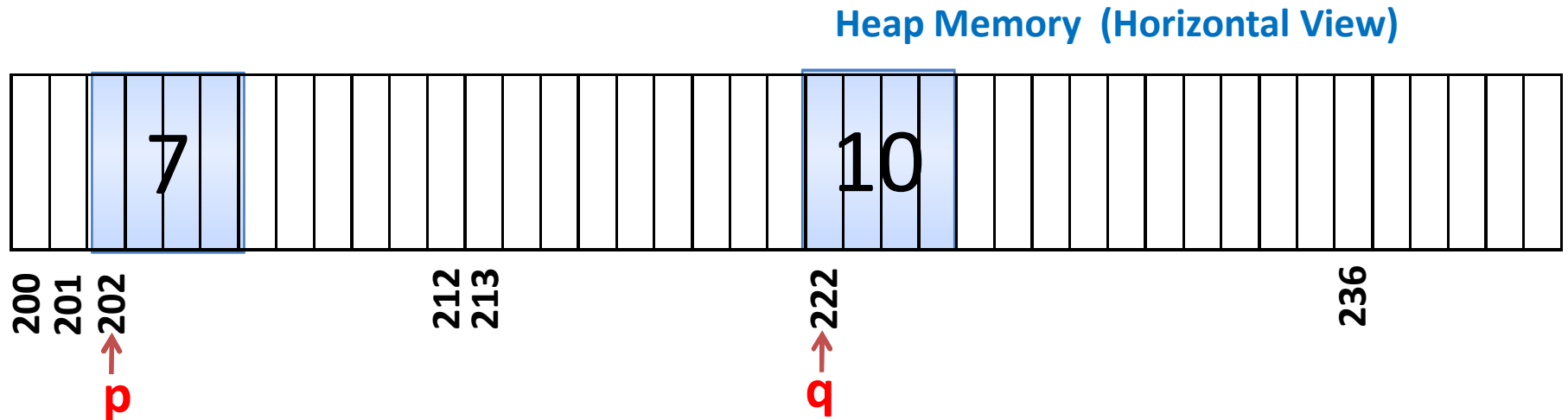int *q=(int*)malloc(sizeof(int));   *q=10;

Solution: Linked List

Heap Memory (Horizontal View)

Suppose we need to store a list of 4 integers: 7, 10, 5, 9.

int *p=(int*)malloc(sizeof(int));   *p=7;
int *q=(int*)malloc(sizeof(int));   *q=10;
int *r=(int*)malloc(sizeof(int));   *r=5;

Solution: Linked List

Heap Memory (Horizontal View)

Suppose we need to store a list of 4 integers: 7, 10, 5, 9.

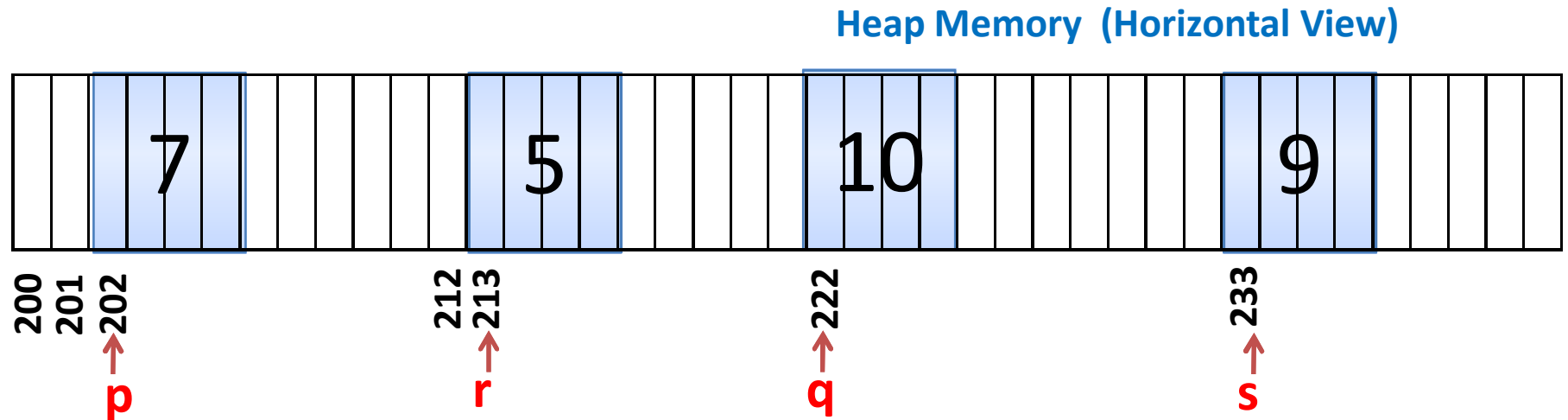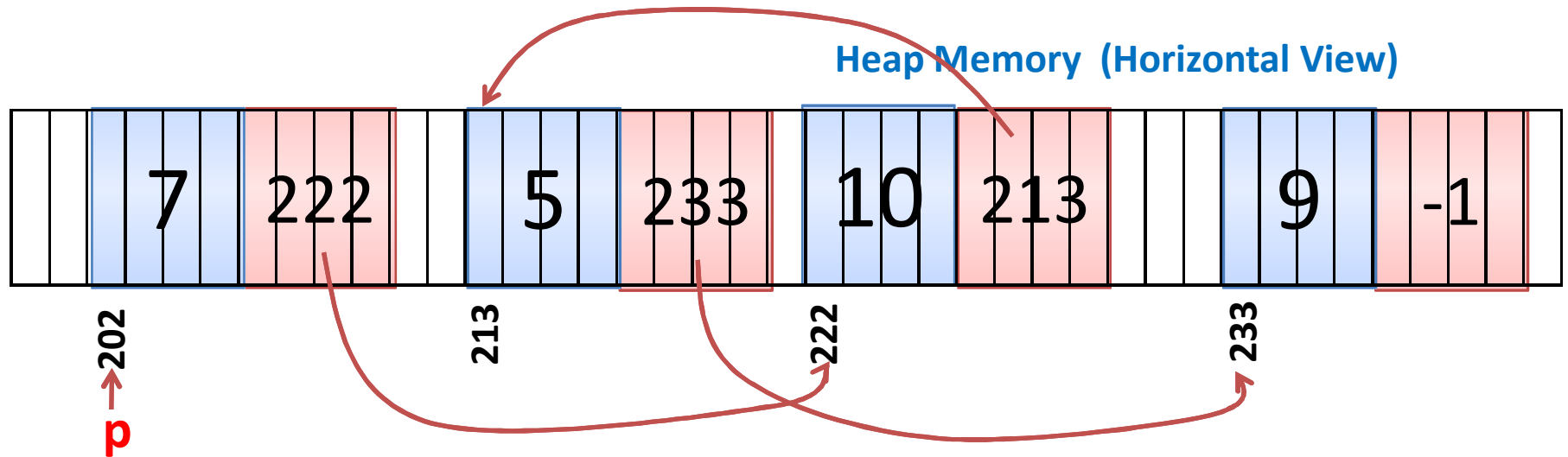int *p=(int*)malloc(sizeof(int));   *p=7;
int *q=(int*)malloc(sizeof(int));   *q=10;
int *r=(int*)malloc(sizeof(int));   *r=5;
int *s=(int*)malloc(sizeof(int));   *s=9;

Non-contiguous allocation: Hence it is not possible to traverse the list directly as done in array.

Solution: Linked List

Heap Memory (Horizontal View)

To traverse the elements of the list, one should store the address of the next element in the list with each element.

Solution: Linked List

Heap Memory (Horizontal View)

| 7 | 222 | | 5 | 233 | | 10 | 213 | | 9 | -1 |

202

p

213

222

233

To traverse the elements of the list {7, 10, 5, 9}, one should store the address of the next element (**pointer to next element**) in the list with each element.

It is sufficient to know the address of (pointer to) the first node (also called as head node) to retrieve all the elements of the list.

The last node points to null (represented by -1 here).

Logical View of Linked List

Head
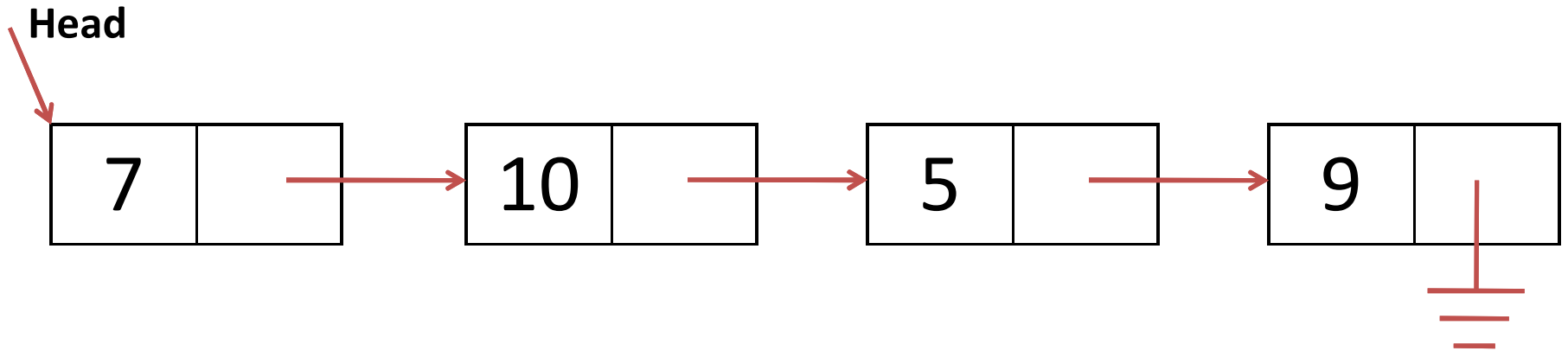
Structure of each node

```
struct node
{
int val;
struct node* next;
}
```