Stack Mechanism Discussion with Time Complexity

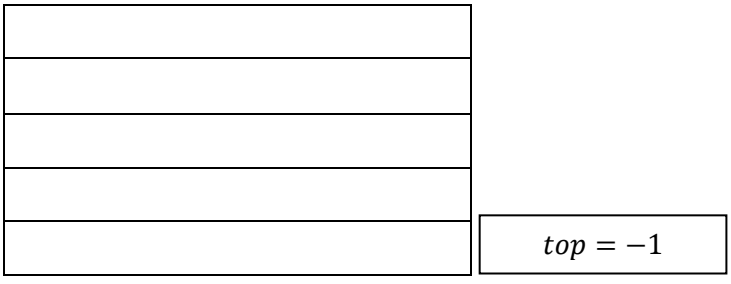
4. *Push Operation*

```
void push(Stack *st, int x)
{
    if (isFull(*st))
    {
        cout << "Stack Overflow" << endl;
    }
    else
    {
        st->top++;
        st->s[st->top] = x;
    }
}
```

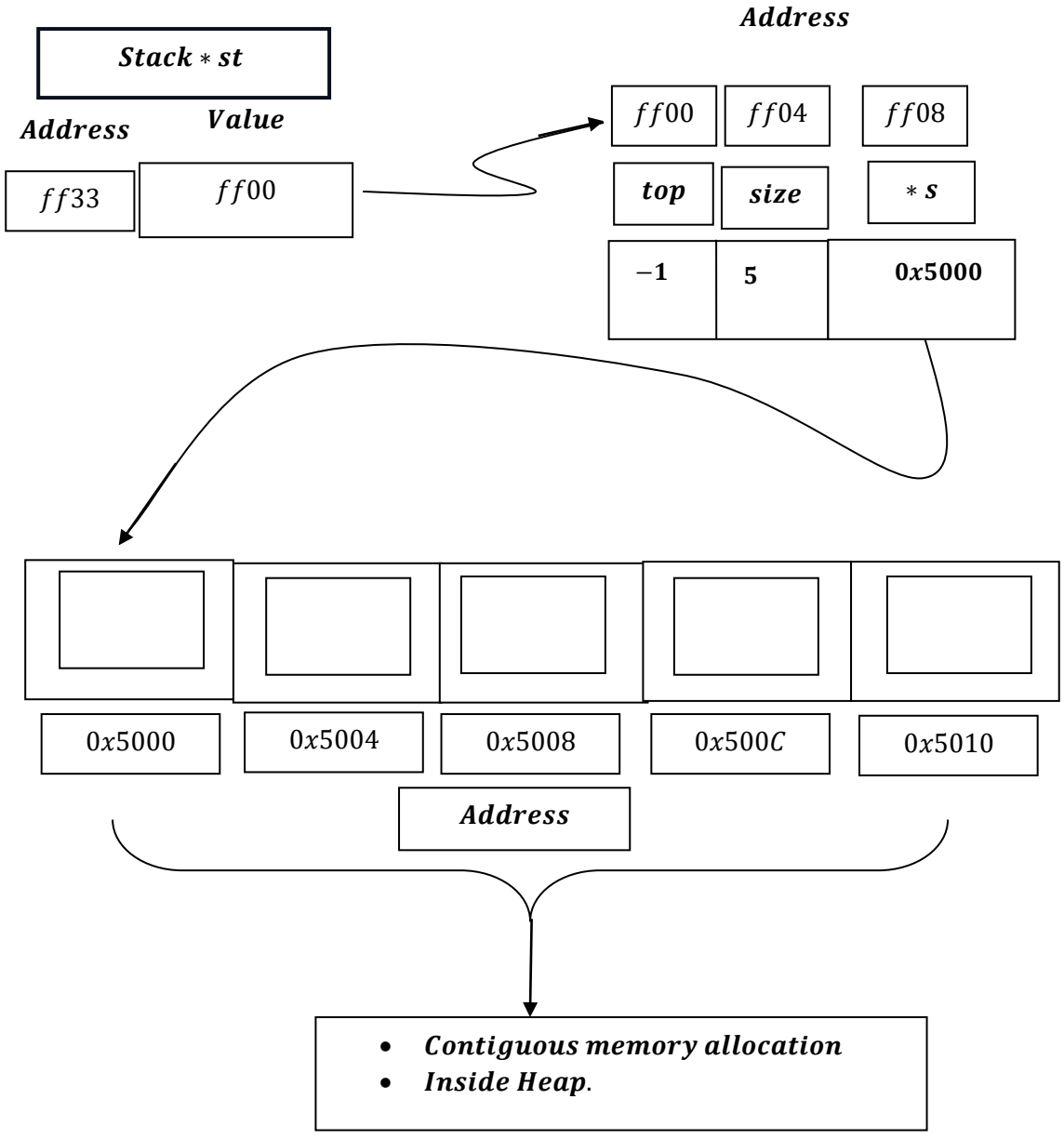
if stack is full then it will print : ``Stack Overflow``.

else Top = Top + 1 and element `x` [user input] is pushed to top of the stack.

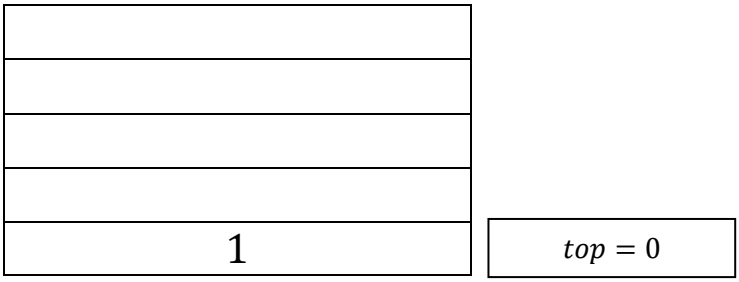
When Empty :

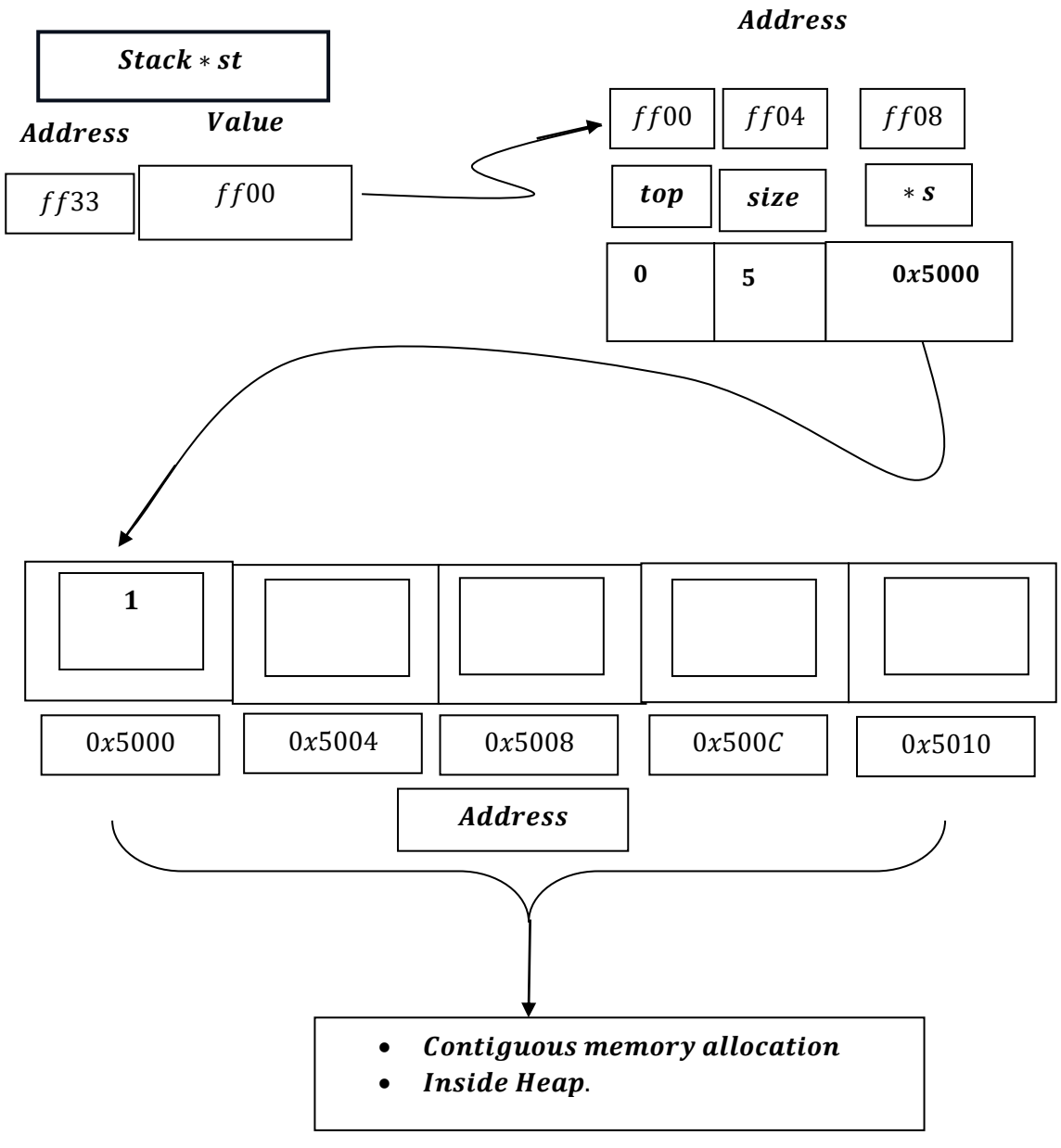


Empty stack


Address
Stack ∗ st
Address
Value
ff33
ff00
ff00
ff04
ff08
top
size
∗ s
−1
5
0x5000
0x5000
0x5004
0x5008
0x500C
0x5010
Address
Contiguous memory allocation
Inside Heap.

First Push

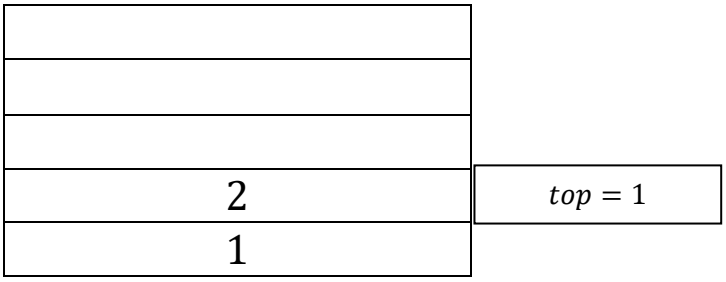

1
$top = 0$

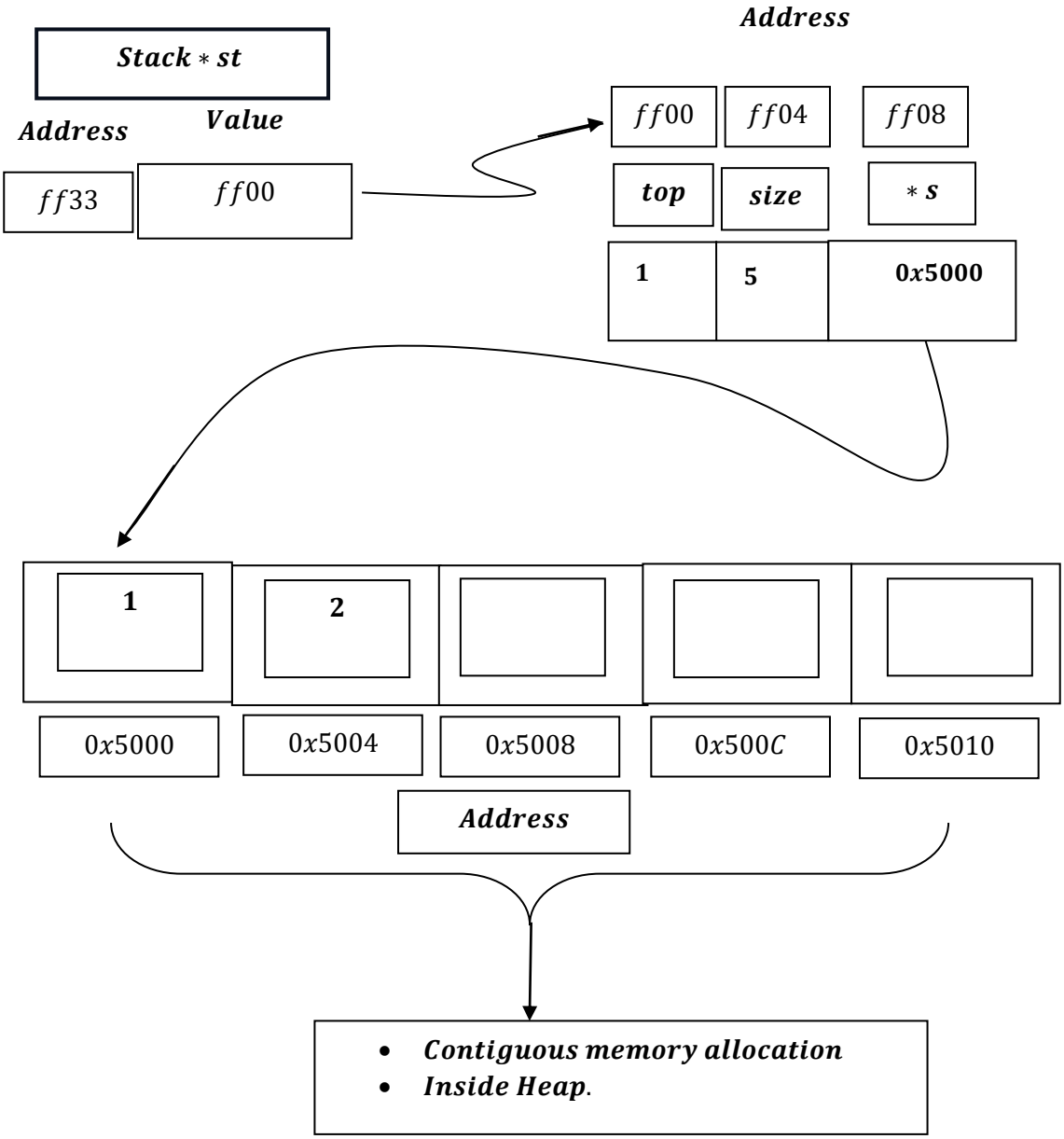

Address
Stack ∗ st
Address
Value
$ff33$
$ff00$
$ff00$
$ff04$
$ff08$
top
size
∗ s
0
5
0x5000
1
0x5000
0x5004
0x5008
0x500C
0x5010
Address
Contiguous memory allocation
Inside Heap.

$$\boldsymbol{Base\ Address + Offset[index\ \times size\ of\ int]}$$

$$\boldsymbol{i.e.\ 0x5000 + (0 \times 4\ bytes) =\ 0x5000 = 1}$$

2nd Push





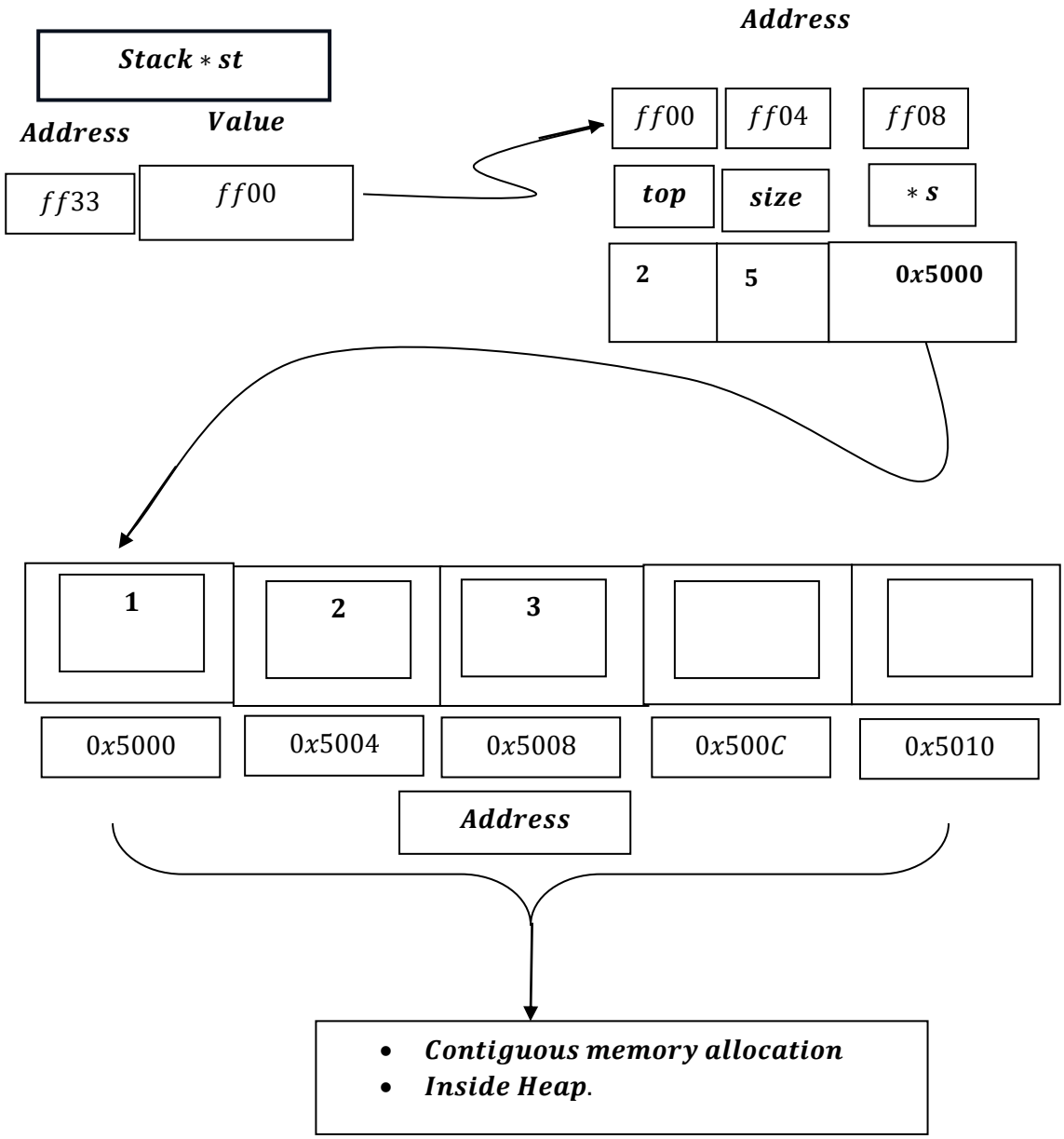
$$\boldsymbol{Base\ Address + Offset[index\ \times size\ of\ int]}$$

$$\boldsymbol{i.e.\ 0x5000 + (1 \times 4\ bytes) = 0x5004 = 2}$$

3rd Push

3
2
1

$top = 2$



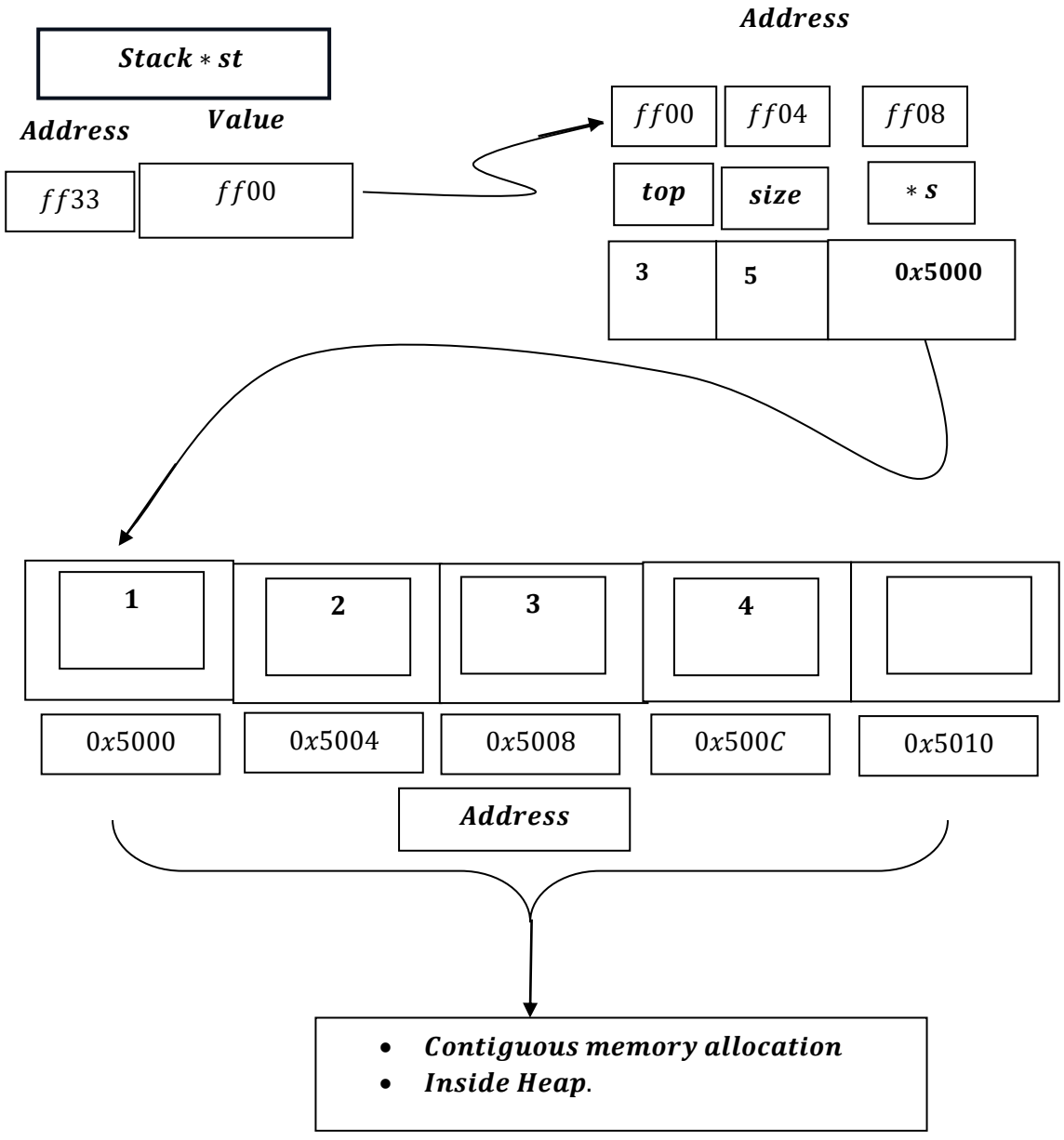
$$\boldsymbol{Base\ Address + Offset[index \times size\ of\ int]}$$

$$\boldsymbol{i.e.\ 0x5000 + (2 \times 4\ bytes) = 0x5008 = 3}$$

4rth Push

4
3
2
1

$top = 3$



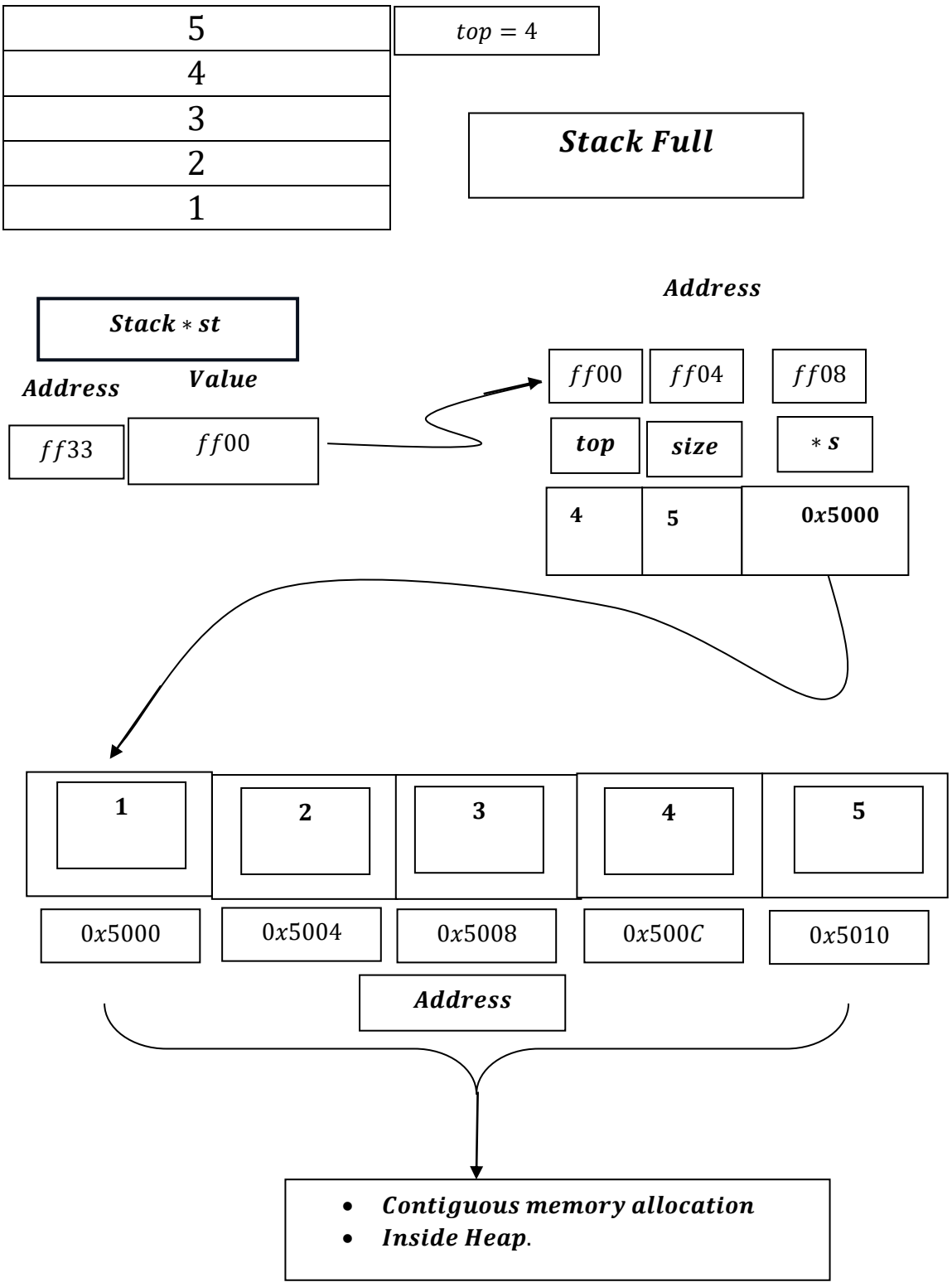
- ***Contiguous memory allocation***
- ***Inside Heap.***

$$\boldsymbol{Base\ Address + Offset[index\ \times size\ of\ int]}$$

$$\boldsymbol{i.e.\ 0x5000 + (3 \times 4\ bytes) = 0x5000 + 12 = 0x5000 + C}$$

$$\boldsymbol{= 0x500C = 4}$$

5th Push



$$Base\ Address + Offset[index\ \times size\ of\ int]$$

$$i.e.\ 0x5000 + (4 \times 4\ bytes) = 0x5000 + 16 = 0x5000 + 10$$

$$= 0x5010 = 5$$

Time Complexity of IsFull function

```
void push(Stack *st, int x)
{
    if (isFull(*st))
    {
        cout << "Stack Overflow" << endl;
    }
    else
    {
        st->top++;
        st->s[st->top] = x;
    }
}
```

1. ***Function Call Overhead: Calling push()*** $\rightarrow$ ***constant time*** $\rightarrow$ $C_1 \rightarrow O(1)$.
2. ***isFull()*** $\rightarrow$ ***Discussed earlier*** $\rightarrow$ ***run constant time*** $\rightarrow$ $O(1)$.
3. ***Increment Operation*** $\rightarrow$ ***constant time*** $\rightarrow O(1)$.
3. ***At array index element is inserted*** $i.e.\left(Base + (index\ \times sizeof(int))\right)$ $\rightarrow O(1)$.

Total Time Complexity : $O(1) + O(1) + O(1) = O(1)$

What is Function Call Overhead?

Function call overhead means the small amount of extra work the system does when a function is called.

```
void push(Stack *st, int x)
{
    if (isFull(*st))
    {
        cout << "Stack Overflow" << endl;
    }
    else
    {
        st->top++;
        st->s[st->top] = x;
    }
}
```

Function call:

```
        push(&stck, x);
```

Even if our function does almost nothing, the system still has to:

1. Save the current execution state
2. Pass arguments to the function
3. Allocate space for local variables
4. Jump to the function's memory location
5. After execution, return back to the caller

All of this takes a **constant amount of time**.

Stack Data structure basically shows how call stack frame works i.e. in LIFO method , but in Stack Data Structure we manually do push() and pop() , where in call stack frame, it creates stack frame for entire function containing local variables , parameters , return address and saved registers and adds or push another stack frame if called again and pops whole stack frame automatically thus different from Stack datastructure , yet conceptually similar.
