```
class Employee{
id,
name,
salary
}
```

```
cin>>id
getchar();
getline(cin,name)
cin>>salary
```

Constant
Datamember
member functions
Objects

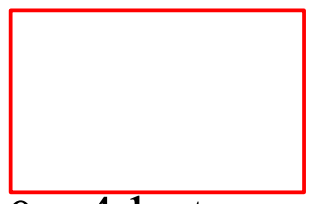
```
const Test *const this;
```

Dynamic Memory Management
malloc()
new -> heap

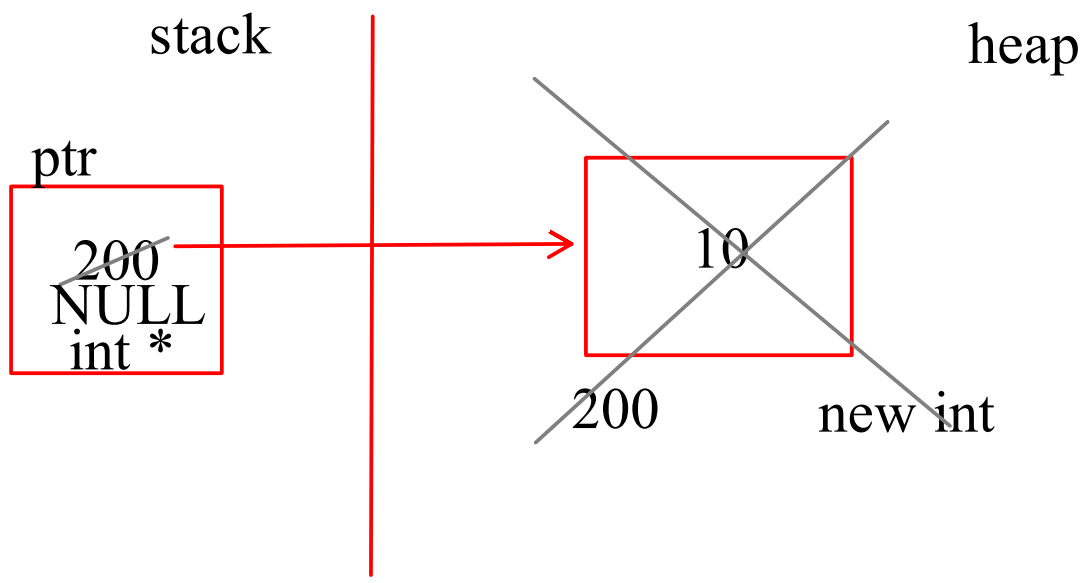
free()
delete ->

```
const int num1=10;
const int *const ptr = &num1;
```

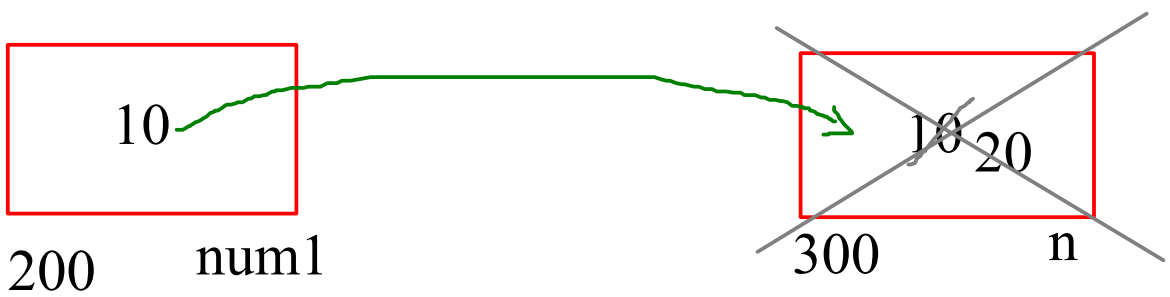
```
int *ptr = new int;
*ptr = 10;
delete ptr;
ptr = NULL;
```



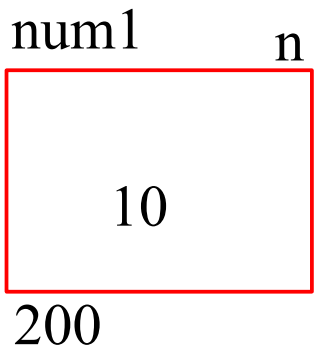

```
int main(){
//int *ptr = new int;
//*ptr = 10;
int *ptr = new int(10);
cout<<ptr<<endl;
cout<<*ptr<<endl;
delete ptr;
ptr = NULL;
}
```



Reference
- It is an alias for an existing memory location

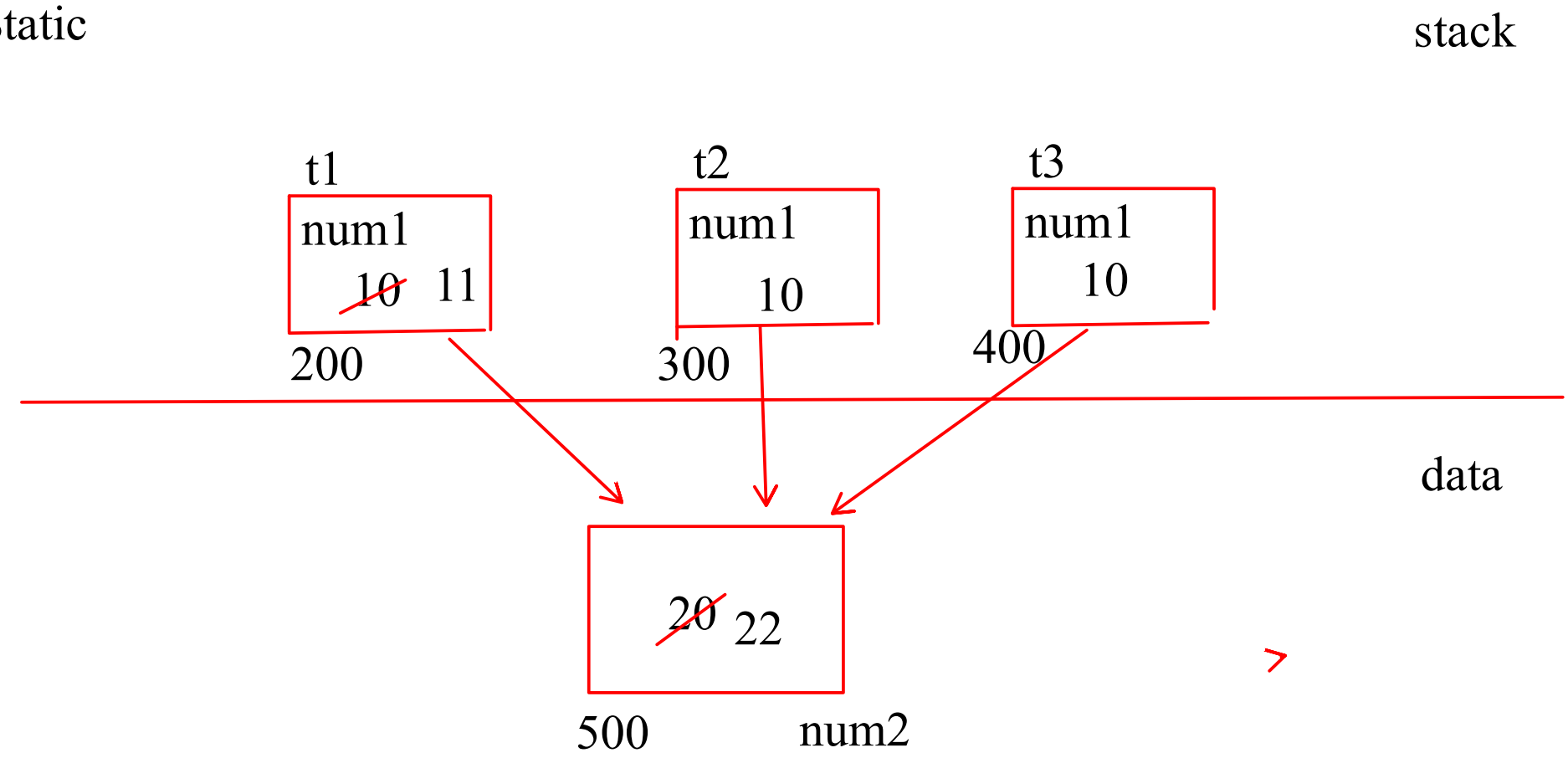


stack



Reference
- It is an alias for an existing memory location
- once the reference is initialize the referent cannot be changed
- reference helps us to avoid pointer complexity
- reference cannot be used to point at dynamic memory allocation

**Static



Static
We can make the data members as well as member functions as static

Data members as Static
- The memory will be allocated only once on data section while program loading
- it must be initialized outside the class globally using classname and ::
- These data memebrs are ment to be shared across multiple objects

Member functions as static
- We can access only static data memebrs inside them.
- We cannot access non static data members inside them.
- These are designed to be called on classname using ::

stack

a1

accno
1001
name
Mukesh
balance
10000

a2

accno
1002
name
Ramesh
balance
20000

data

~~1000~~ ~~1001~~ 1002

generate_accno;

Stack

c1

radius
5

c2

radius
7

c3

radius
9

3.14

PI

```
class Test{
int num1;
static int num2;
static int num3;

};
int Test::num2 = 20;
int Test::num3 = 30;
```

```
Test t1;
Test t2;
```

t1

num1

t2

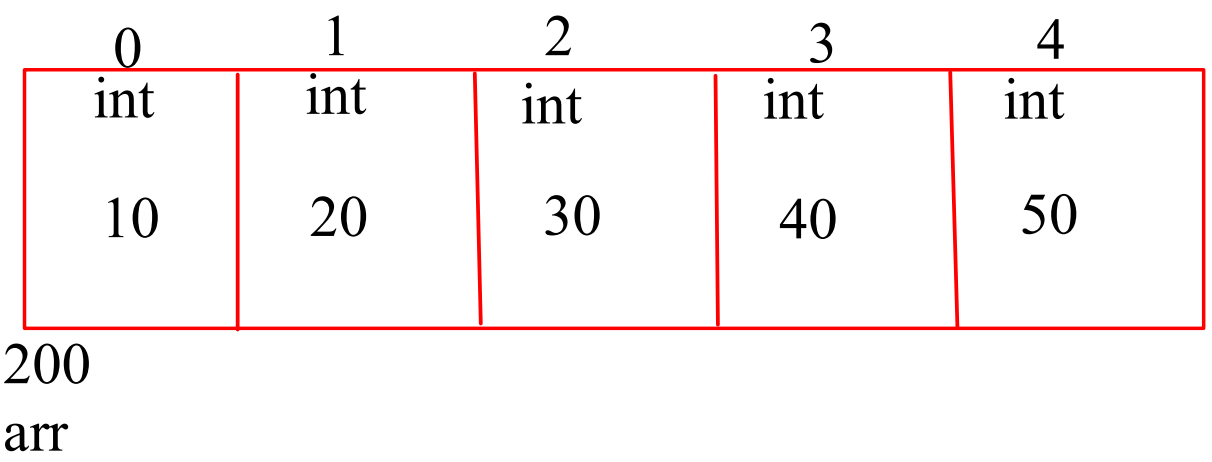
num1

num2

num3

Array

1. It is a datastructure that is used to store the elements of the same type in contigious memory location
2. The array size is fixed

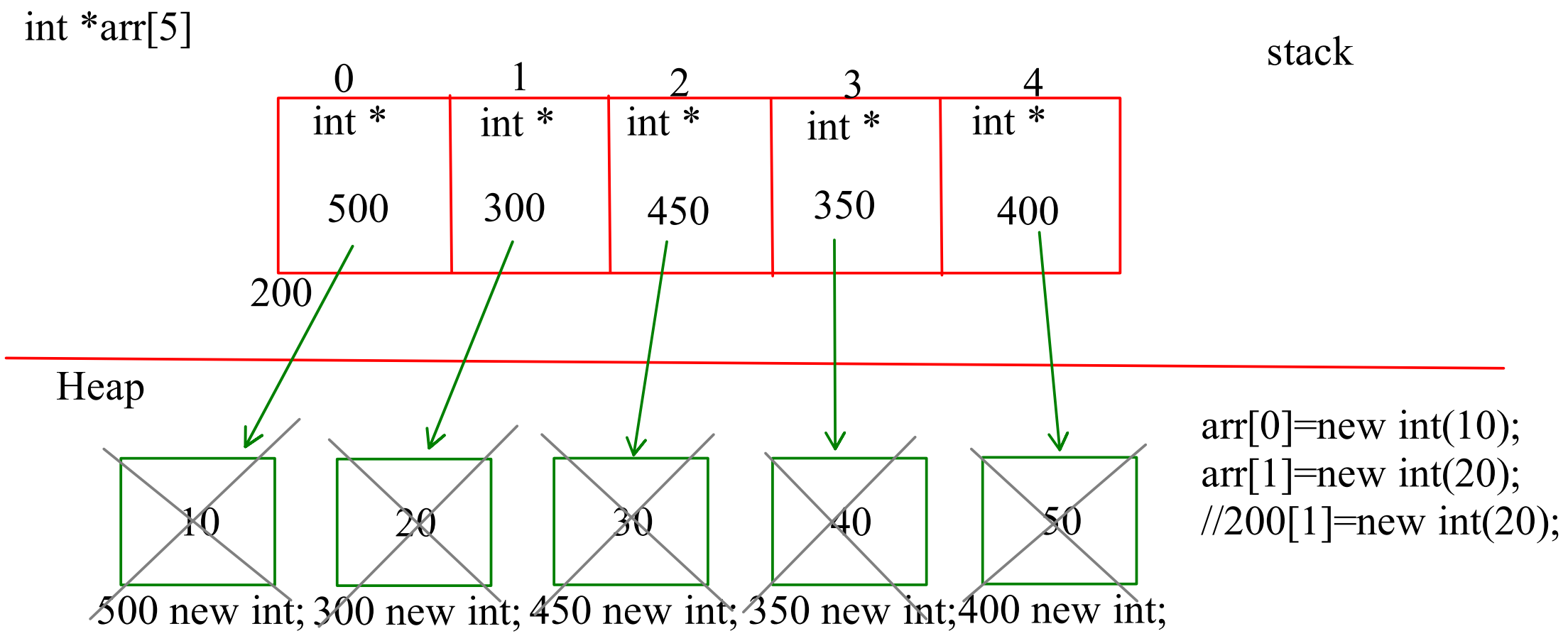



```
int arr[5];
arr[0] = 10;
//200[0] = 10;
arr[1]=20
```

```
arr[2]= 30;
arr[3]=40;
arr[4]= 50;
```

```
for(int i = 0;i<5;i++)
    cout<<arr[i];
```

int *arr[5]



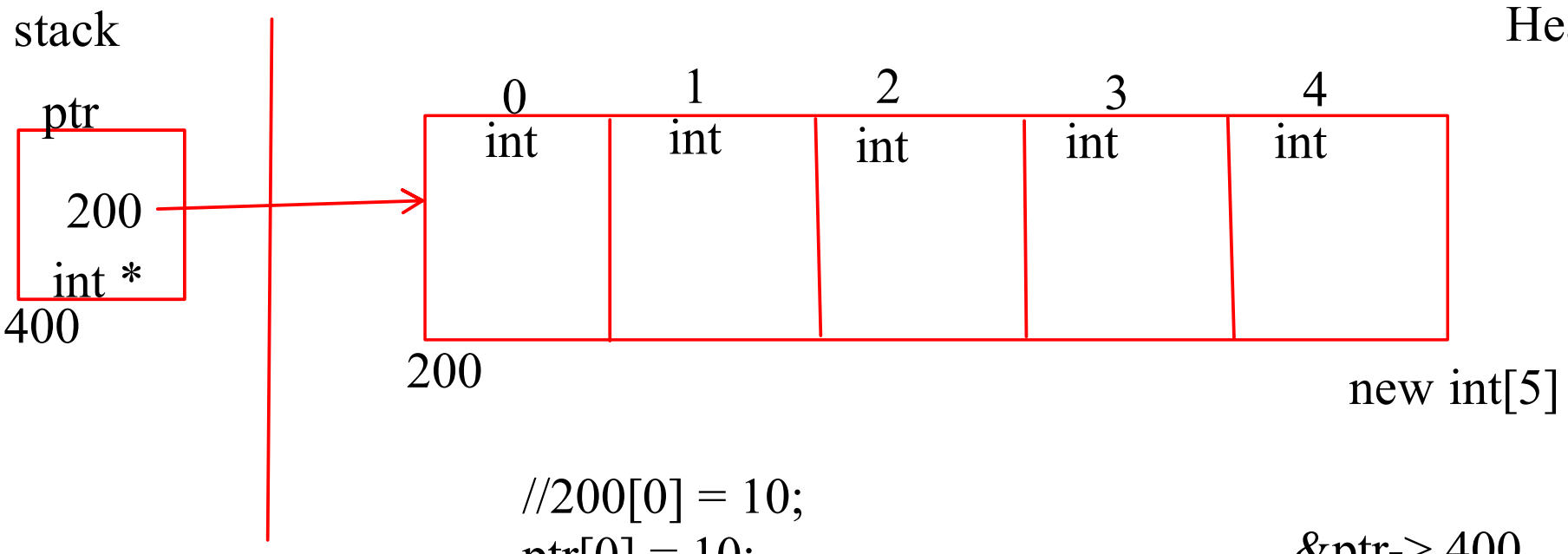

```
arr[0]=new int(10);
arr[1]=new int(20);
//200[1]=new int(20);
```

```
for(int i = 0;i<5;i++)
    cout<<*arr[i];
```

```
//delete 500;
//delete 200[0];
delete arr[0];

//delete 300;
//delete 200[1];
delete arr[1];
```

```
for(int i = 0;i<5;i++)
    delete arr[i];
    arr[i] = NULL;
```

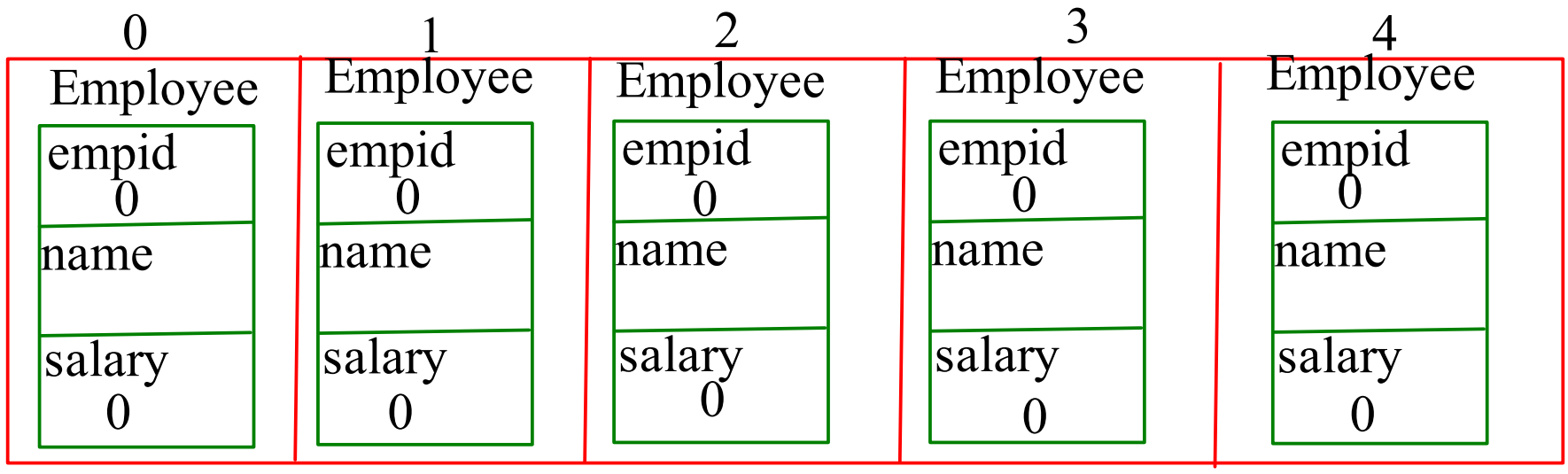


//200[0] = 10;
ptr[0] = 10;
ptr[1] = 20;

&ptr-> 400
ptr -> 200

delete 200;
delete[] ptr;

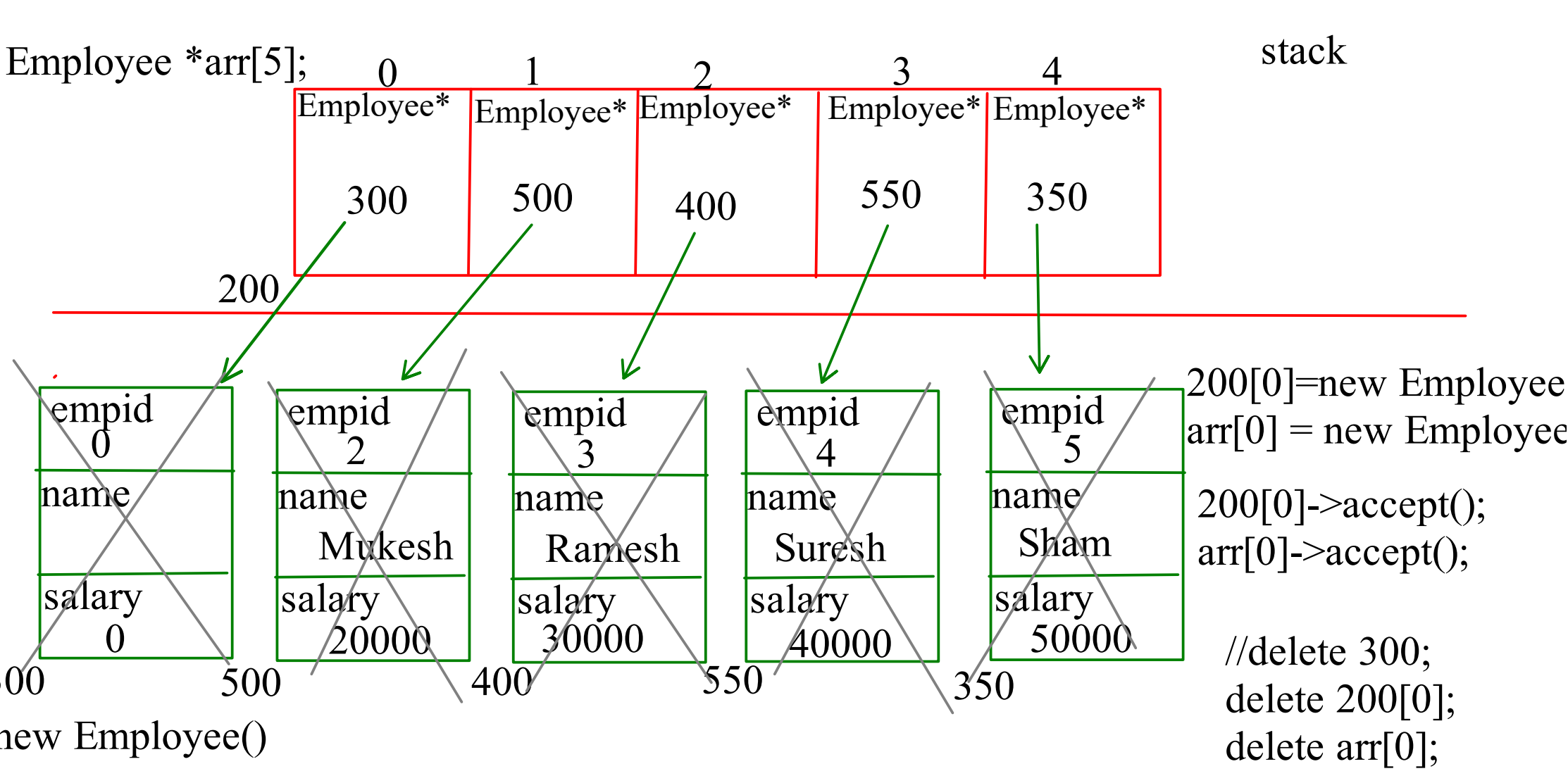
Employee arr[5];



//200[0].accept();
arr[0].accept();

//200[0].display();
arr[0].display();

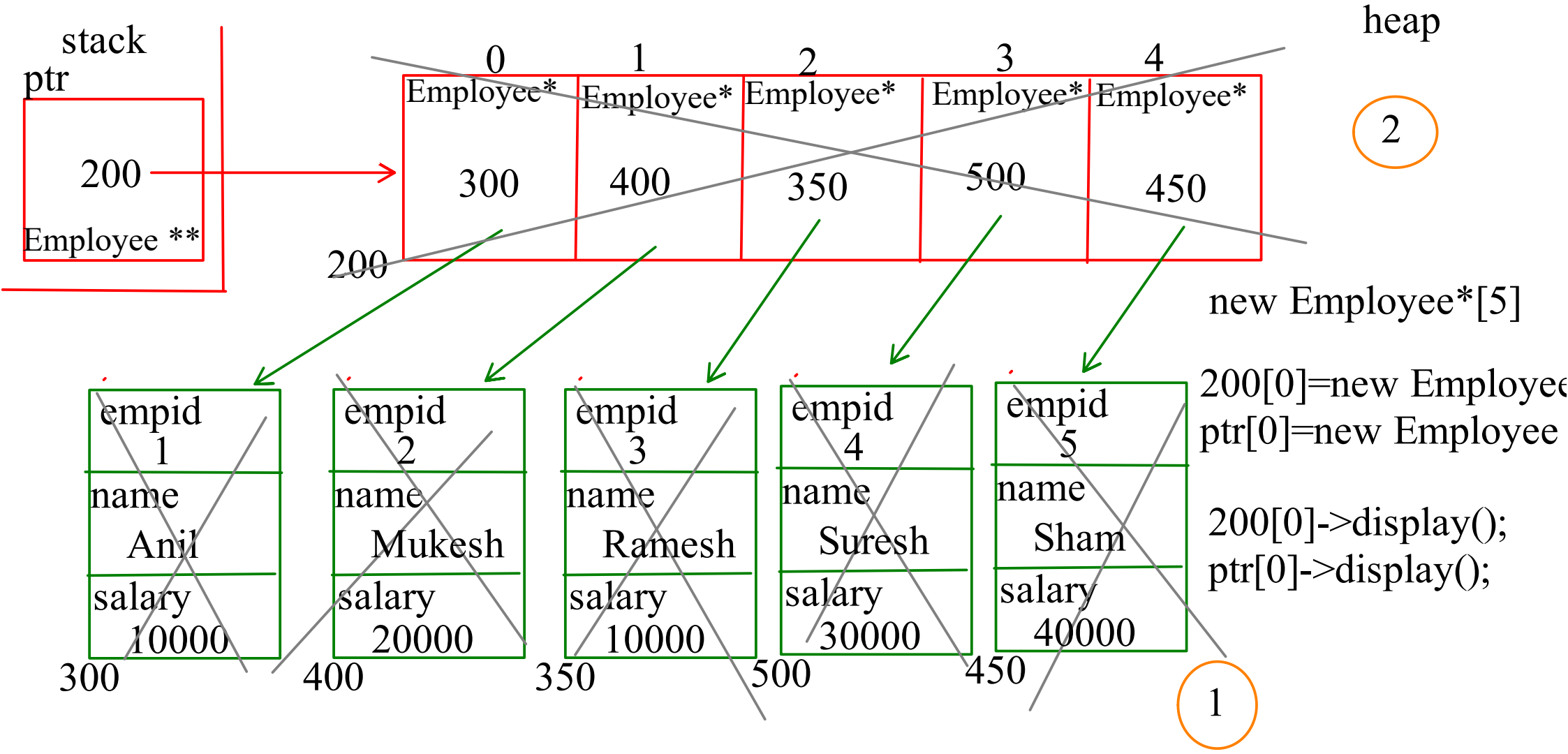
Employee *arr[5];



200[0]=new Employee
arr[0] = new Employee

200[0]->accept();
arr[0]->accept();

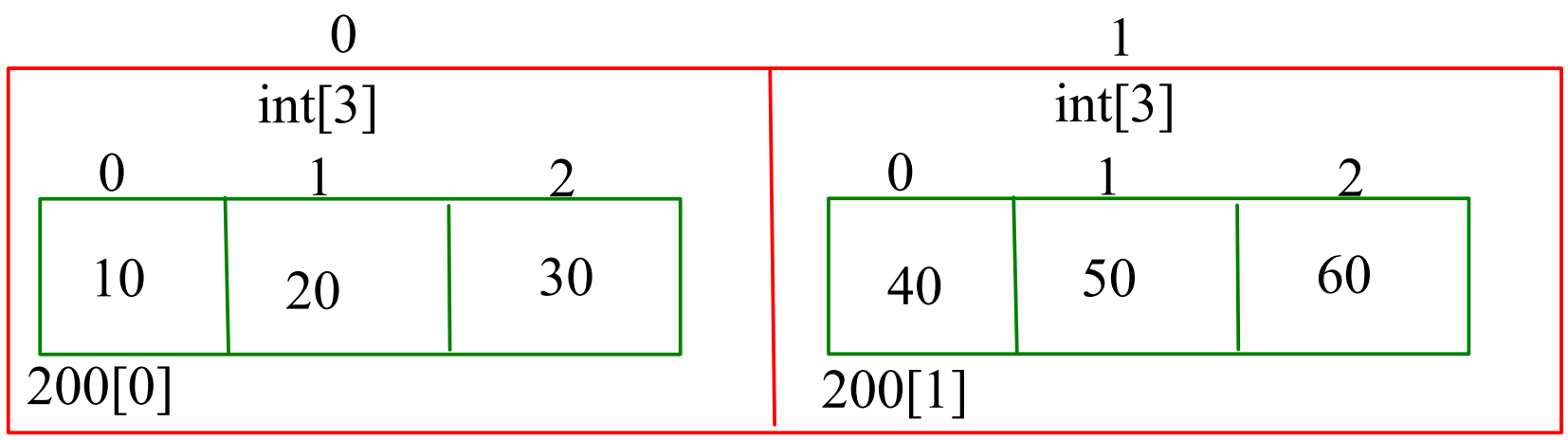
//delete 300;
delete 200[0];
delete arr[0];




```
delete 300;
delete 200[0];
delete ptr[0];
```

```
delete []200;
delete []ptr;
```

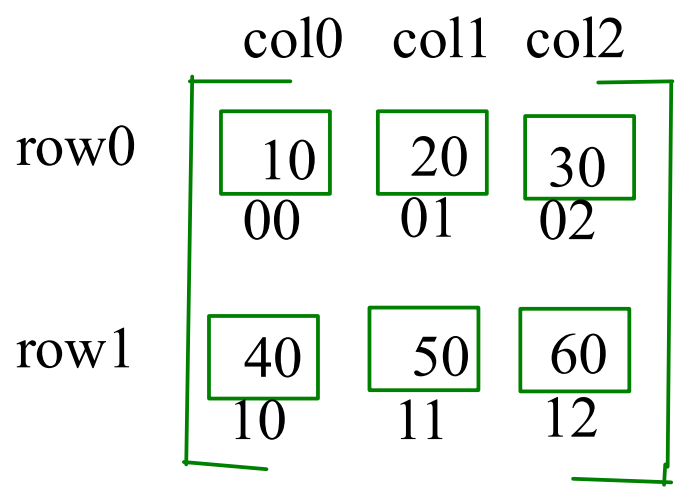
int arr[2][3]



200

```
//200[0][0] = 10;
arr[0][0] = 10;
arr[0][1]=20;
arr[0][2]=30;
```

```
//200[1][0] = 40;
arr[1][0] = 40;
arr[1][1] = 50;
arr[1][2] = 60;
```



[row][col]