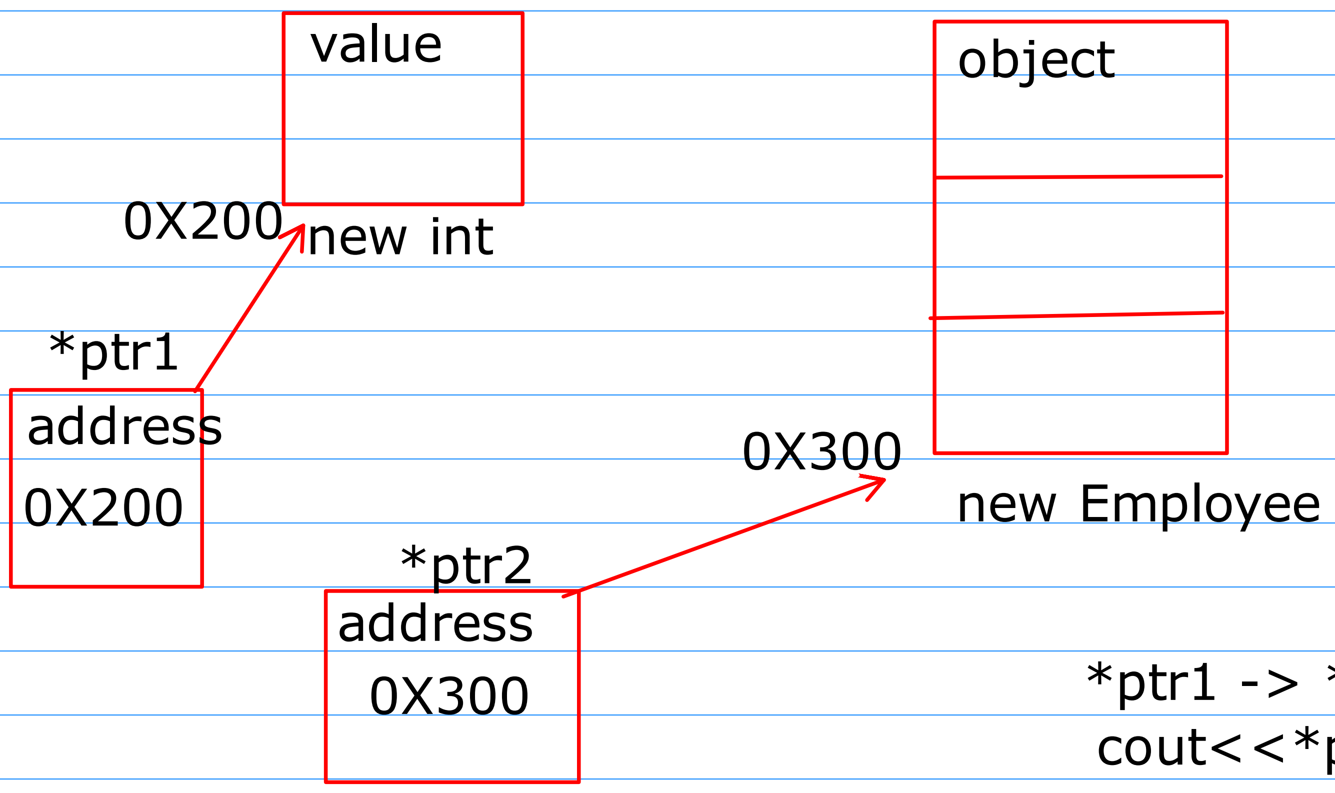
static -> Sharing

*ptr=new int[5]
delete[] ptr

int*
Employee*

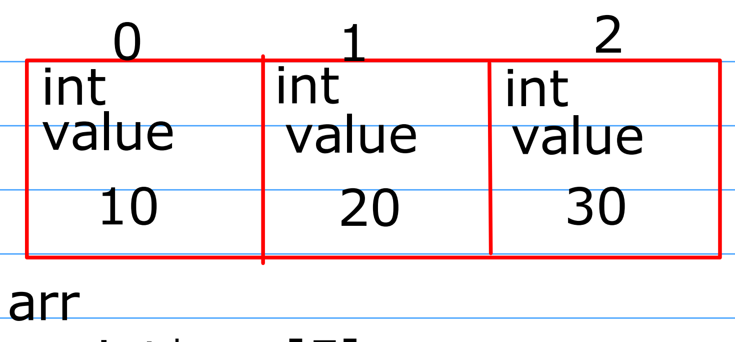
int *ptr1=new int;
int *ptr2=new Employee



*ptr1 -> *0X200
cout<<*ptr1

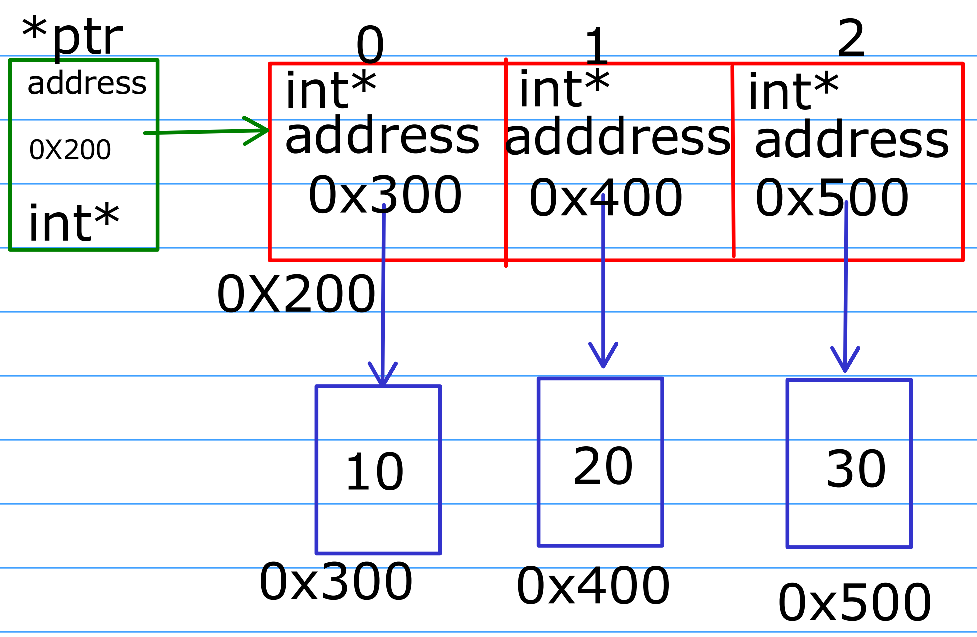
ptr2 -> 0X300
ptr2->name

int arr[5]



arr[0] ->10

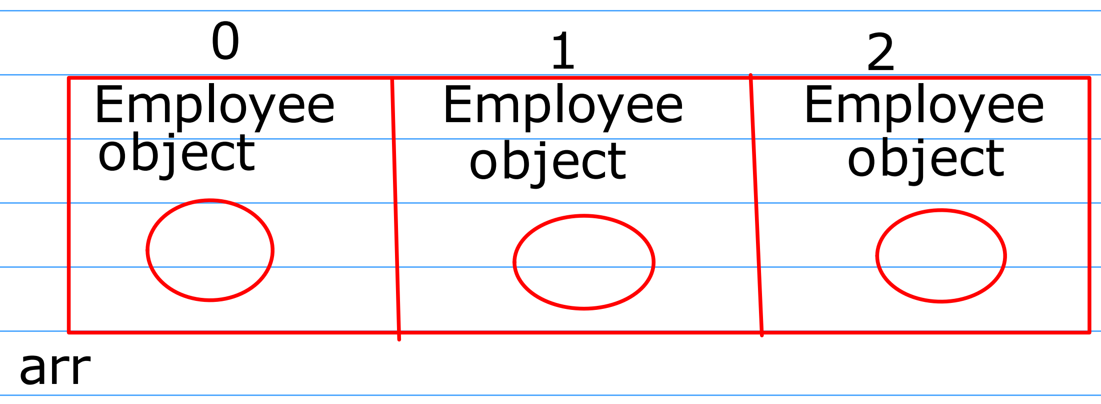
int* arr[5]



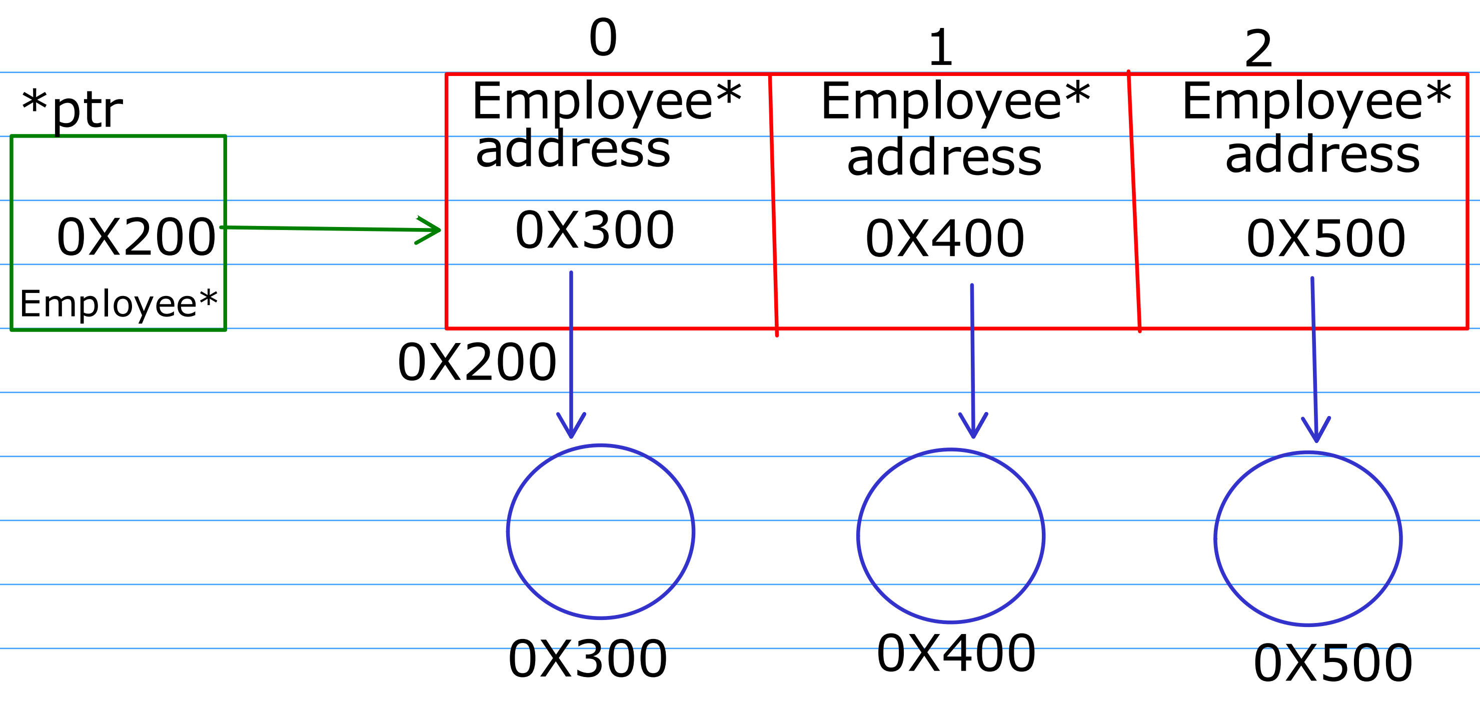
*arr[0] -> *0X300

cout<<0X300; -> 10
cout<<0X200[0] -> 0X300
cout<<*0X200[0]->10
cout<<*ptr[0]->10

Employee arr[3];
Employee* arr[3];
new Employee[3];
new Employee*[3];



arr[0].display()



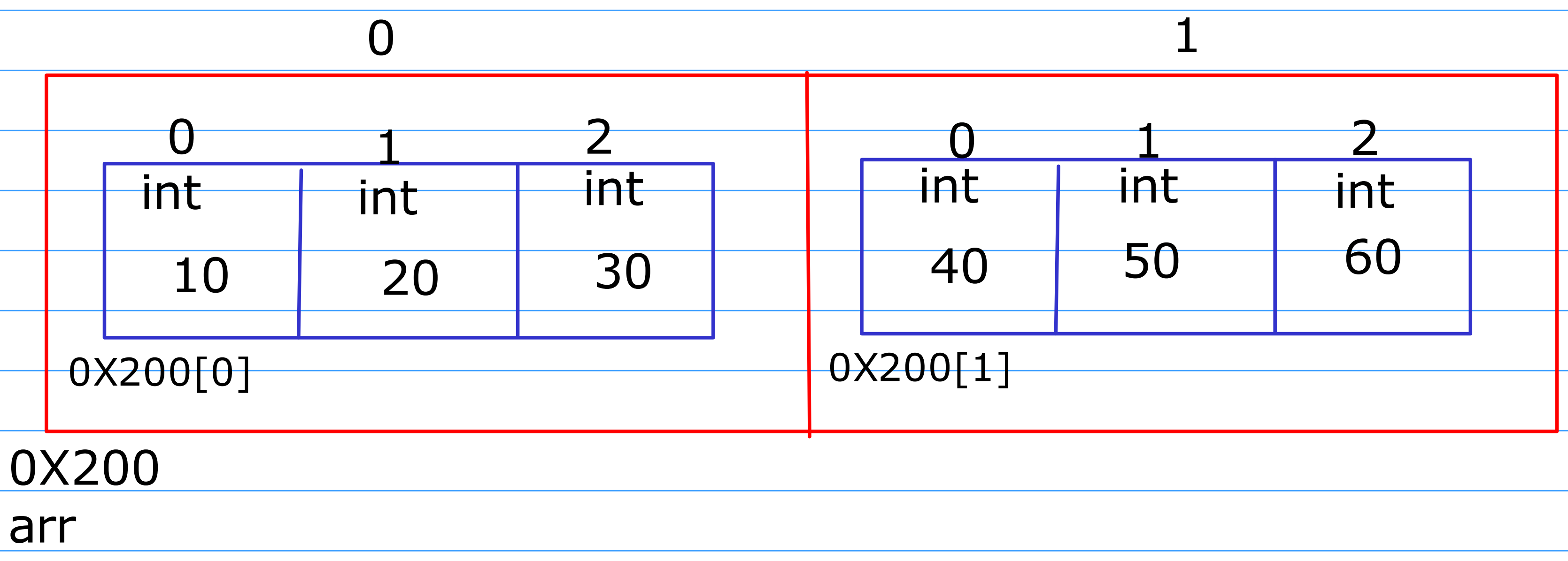
Employee*arr[3]

arr[0] -> 0X300
arr[0]->display()

Time t;
t.
Time *t = new Time();
t->

0X200[0]->display()
ptr[0]->display();

Multidimensional Array
int arr[2][3];

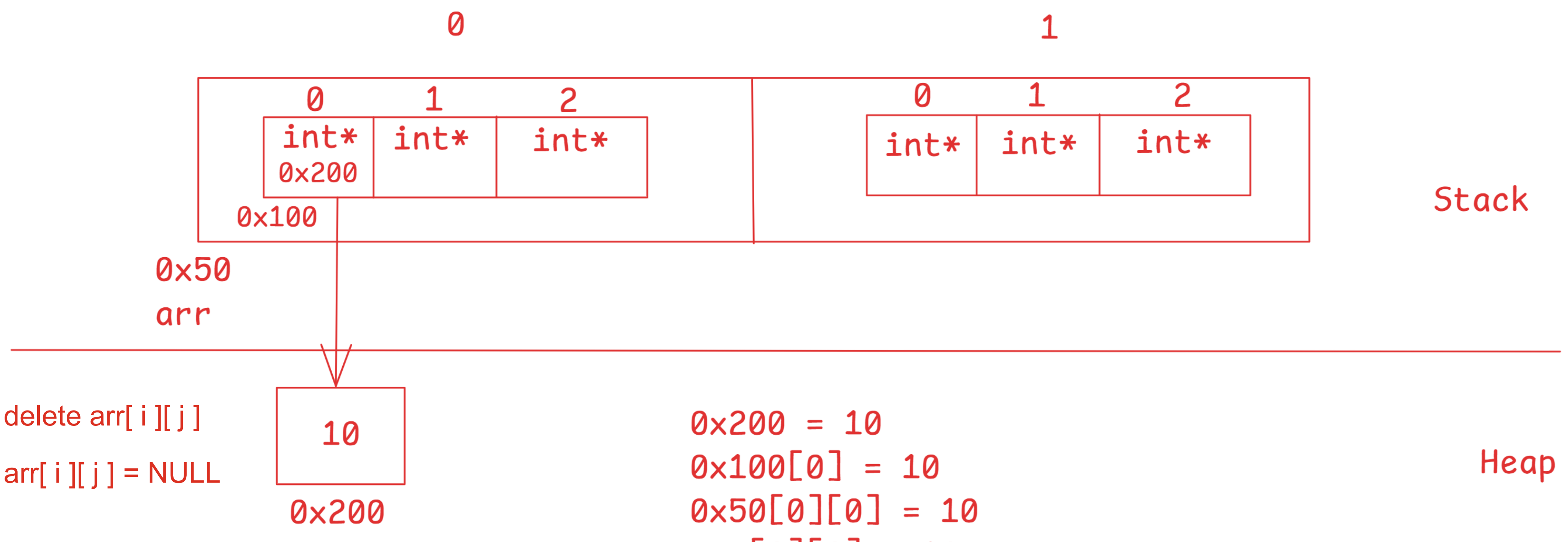


0X200[0][0] = 10;	0X200[1][0] = 40;
arr[0][0] = 10;	arr[1][0] = 40;
arr[0][1] = 20;	arr[1][1] = 50;
arr[0][2] = 30;	arr[1][2] = 60;

```
for(int i=0;i<2;i++){
        for(int j =0; j<3;j++){
                cout<<"Element = "<<arr[i][j]<<endl;
        }
}
```

Main2

```
int *arr[2][3];
arr[0][0] = new int(10);
```



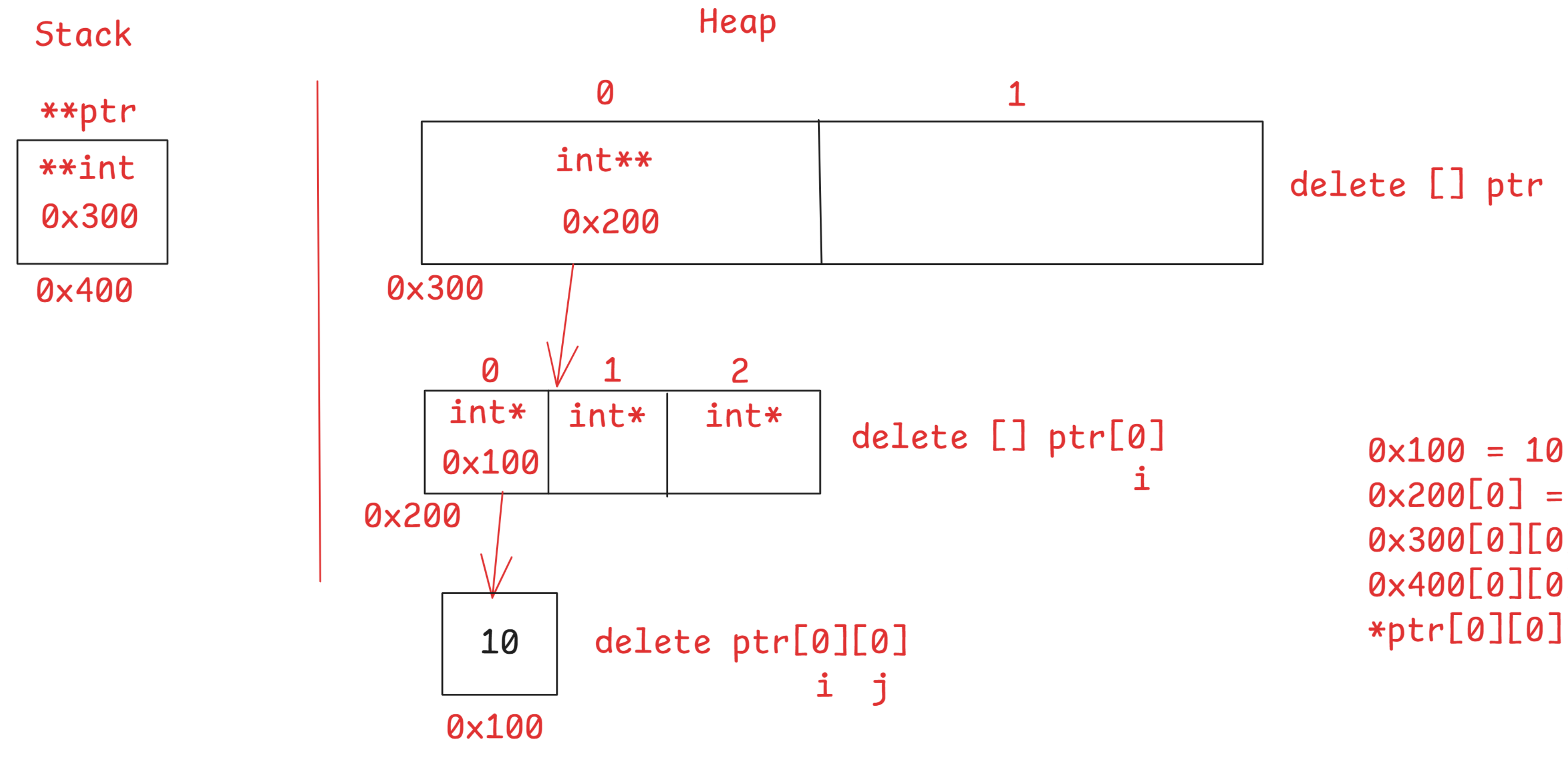
delete arr[i][j]

arr[i][j] = NULL

```
0x200 = 10
0x100[0] = 10
0x50[0][0] = 10
arr[0][0] = 10
```

Main4

```
int ***ptr = new int **[2];
    ptr[0] = new int *[3];
    ptr[1] = new int *[3];
    ptr[0][0] = new int(10);
```

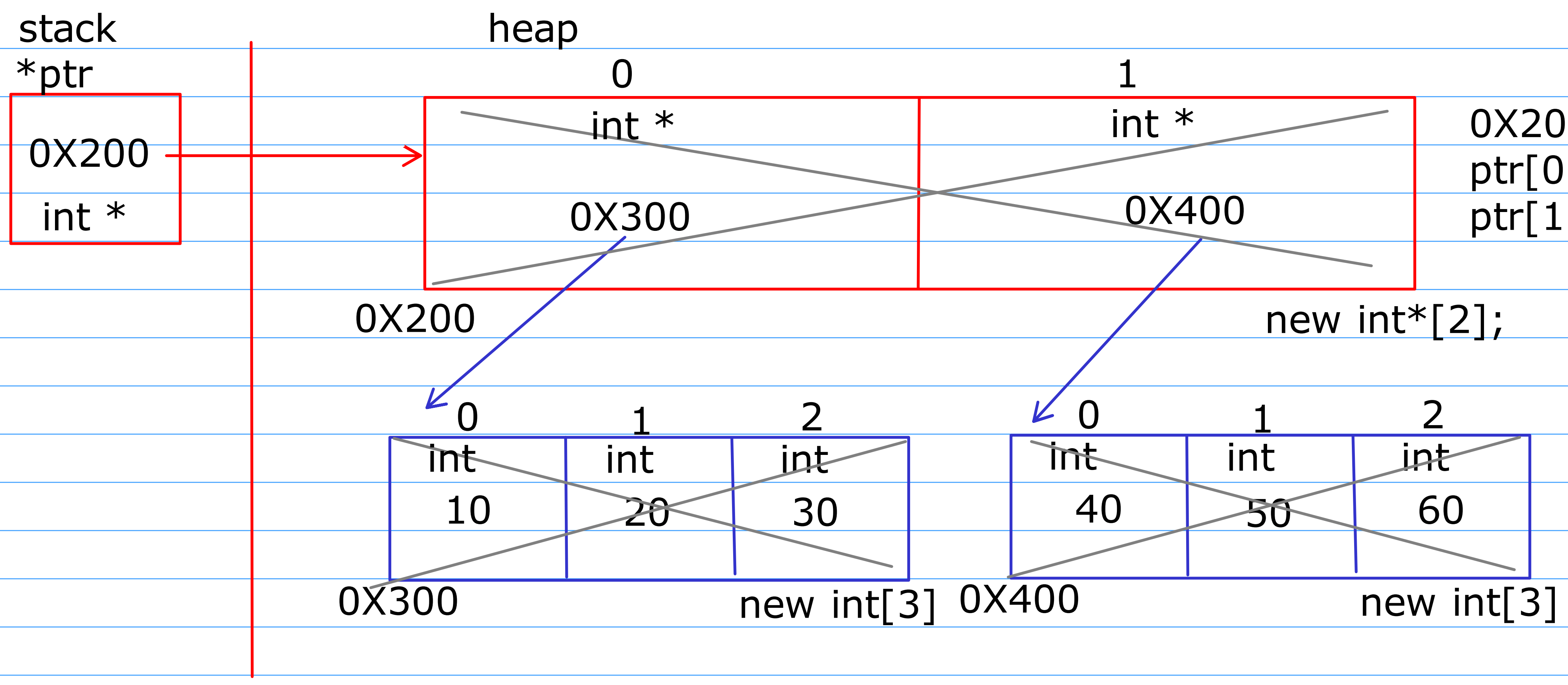


delete [] ptr

delete [] ptr[0]
i

delete ptr[0][0]
i j

```
0x100 = 10
0x200[0] = 10
0x300[0][0] = 10
0x400[0][0] = 10
*ptr[0][0] = 10
```



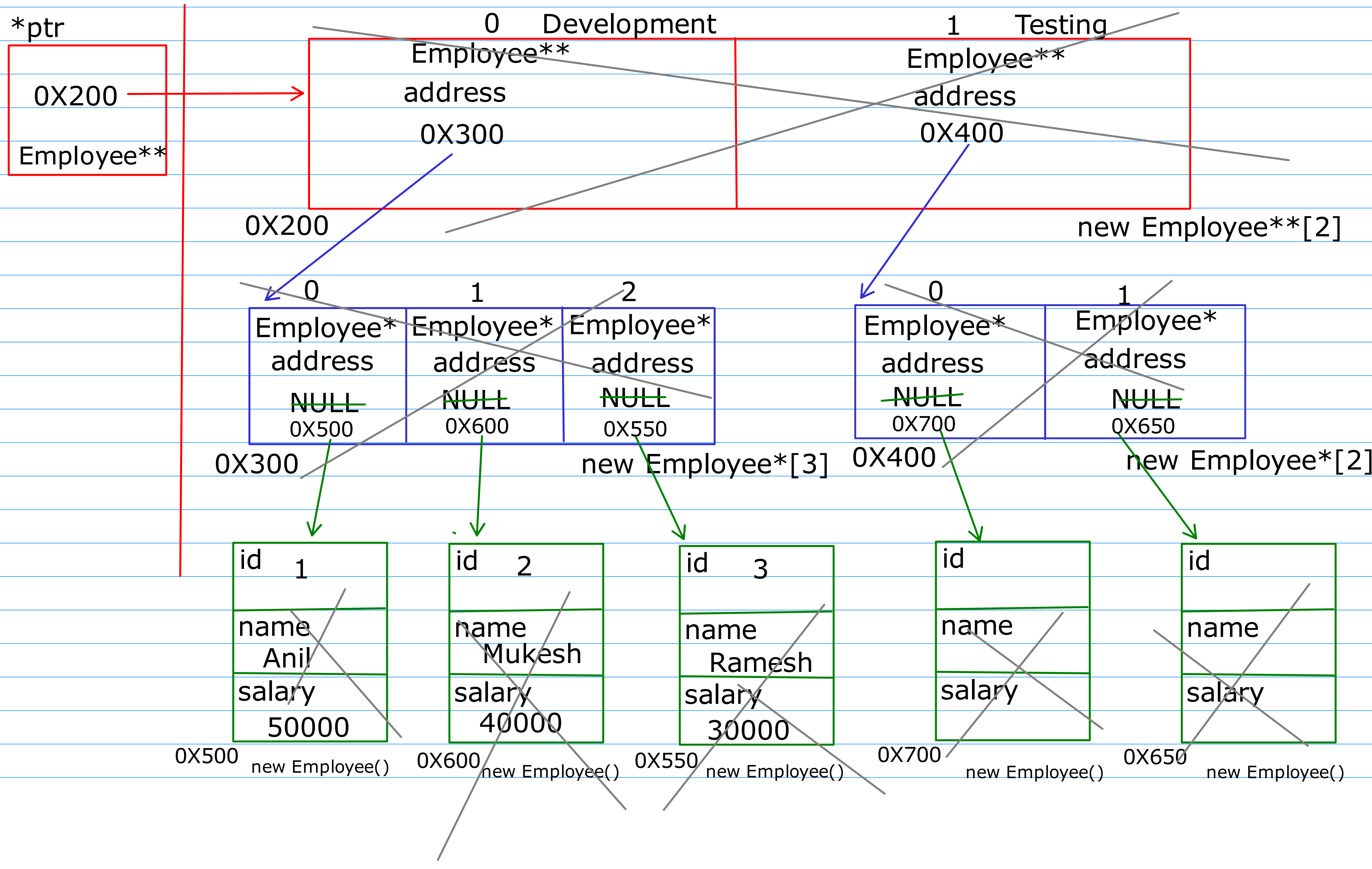
```
0X200[0] = new int[3];
ptr[0] = new int[3];
ptr[1] = new int[3];
```

```
delete []0X200;
delete[] ptr;
ptr = NULL;
```

```
0X300[0] = 10
0X200[0][0]=10
ptr[0][0] = 10;
```

```
delete []0X300;
delete[]0X200[0];
delete[] ptr[0];
ptr[0] = NULL;
```

Departments
Dev -> 3
Test -> 2



```
0X300[0] = new Employee(1,"Anil",50000);
0X200[0][1]=new Employee(2,"Mukesh",40000);
ptr[0][2]=new Employee(3,"Ramesh",30000);
```

```
0X400[0]=new Employee(4,"Suresh",20000);
0X200[1][0]=new Employee(4,"Suresh",20000);
ptr[1][0]=new Employee(4,"Suresh",20000);
ptr[1][1]=new Employee(5,"Ram",10000);
```

```
Students
DAC(240), KDAC(120)
new Student**[6]
new Student*[240]
new Student*[120]
new Student*[120]
new Student*[60]
new Student*[120]
new Student*[60]
```

OOP
Major
1. Abstraction
2. Encapsulation
3. Modularity
4. Hireracby

Minor
1. Polymorphism
    - compiletime
    - runtime
2. Persistance
3. Concurrency

Hireracby
- It represents reusability
- In hireracby the reusing of classes depends on the type of relationship that the entities form
- Their are two type of relationship
    1. has-a relationship represents Association
    2. is-a relationship represents Inheritance

association (has-a)
Human has-a Heart
Car has-a Engine
Room has-a Window

Employee has-a doj

Dependent Object
- Human,Car,Room,Employee, Customer

Dependency
- Heart,Engine,Window,Date,DateOfBirth

Association can be further classified into two types
1. Composition
    - It represents tight coupling between Dependent and Dependency Object
2. Aggegration
    - It represents loose coupling between Dependent and Dependency Object

When we want to have multiple objects of dependency class inside dependent class we should use Association

```
classs Employee{
Date doj;
Date dol;
Date dob;
}
```

```
class Product{
Date md;
Date ed;
Date od;
Date dd;
}
```

```
class Date{
day,month,year

}
```