CLASS - OVERVIEW

- A Class is like an object constructor, or a "blueprint" for creating objects.
- It contains states and behaviours.
- Behaviour - action performed on the class(functions).
- State - properties of the class(variables).

Instances or Objects

- Represents the physical entity of class.
- We can create any number of objects using a class.
- Each and every object are independent to each other. i.e., changes done in one object will not affect the other objects.

Syntax

Creating a class:

```
class ClassName:
    pass
```

Creating an instance:

```
obj = ClassName()
```

Example:

```
class Employee:
    fname = "steve"
    lname = "jobs"


emp1 = Employee()
emp2 = Employee()
```

Employee

	key	value
0x10	fname	steve
0x11	lname	jobs

Employee

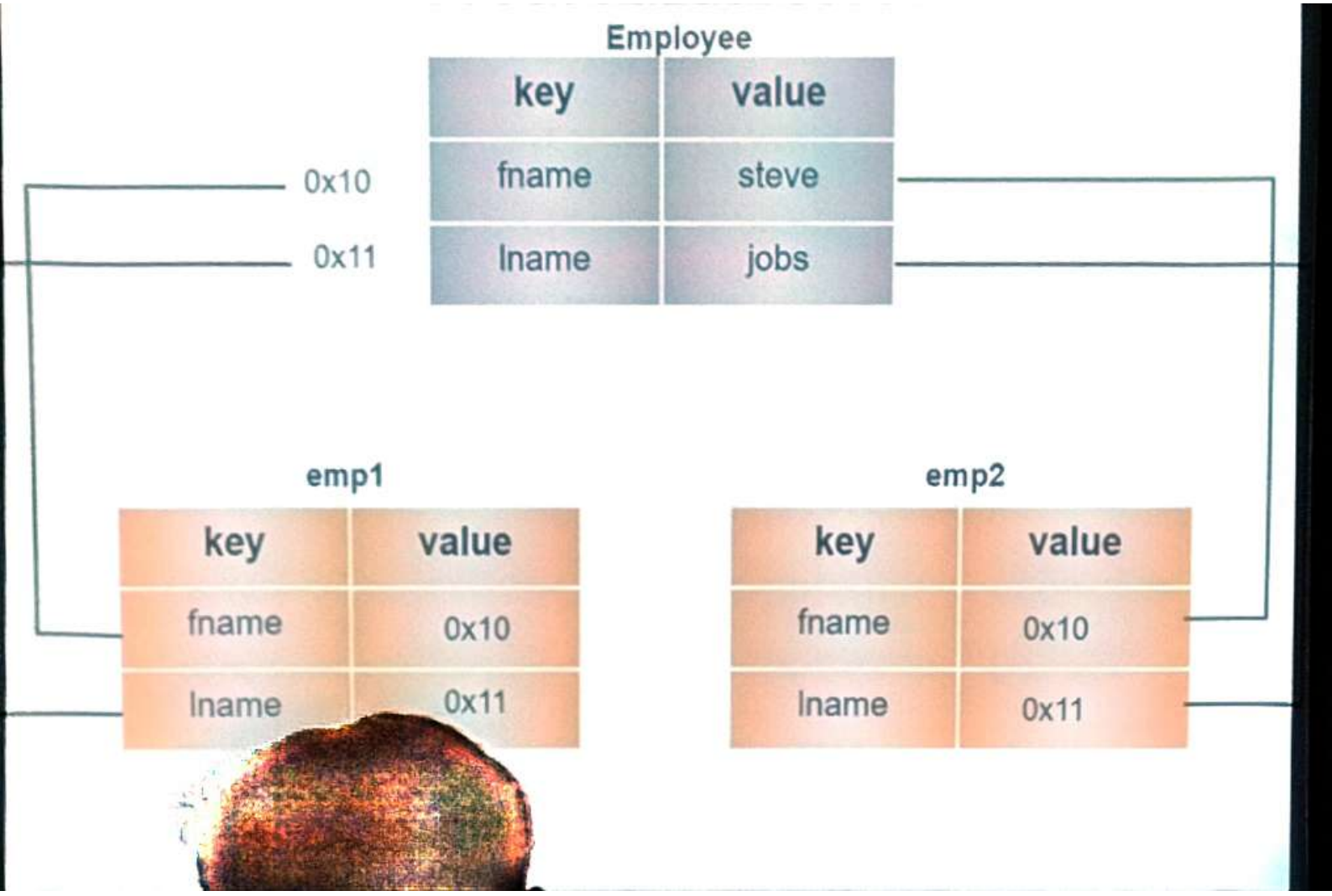
	key	value
0x10	fname	steve
0x11	lname	jobs

emp1

key	value
fname	
lname	

emp2

key	value
fname	
lname	


Employee
key
value
0x10
fname
steve
0x11
lname
jobs
emp1
key
value
fname
0x10
lname
0x11
emp2
key
value
fname
0x10
lname
0x11

methods

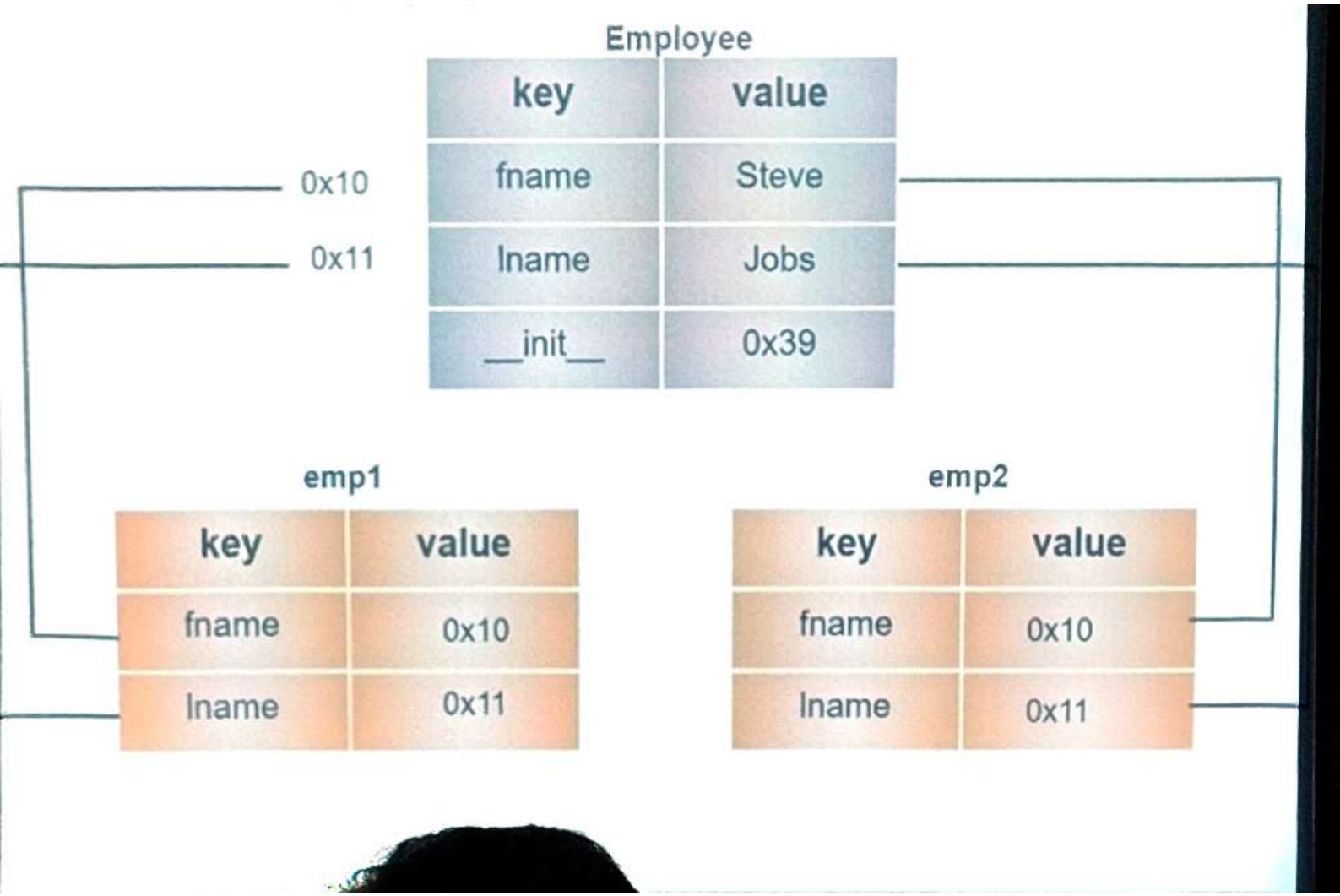
- The functions which are written inside a class are called methods.
- They are called class attributes and should have the first parameter as "self" **(python naming convention).**
- **Self** - holds the address of the instance which is invoking the method.

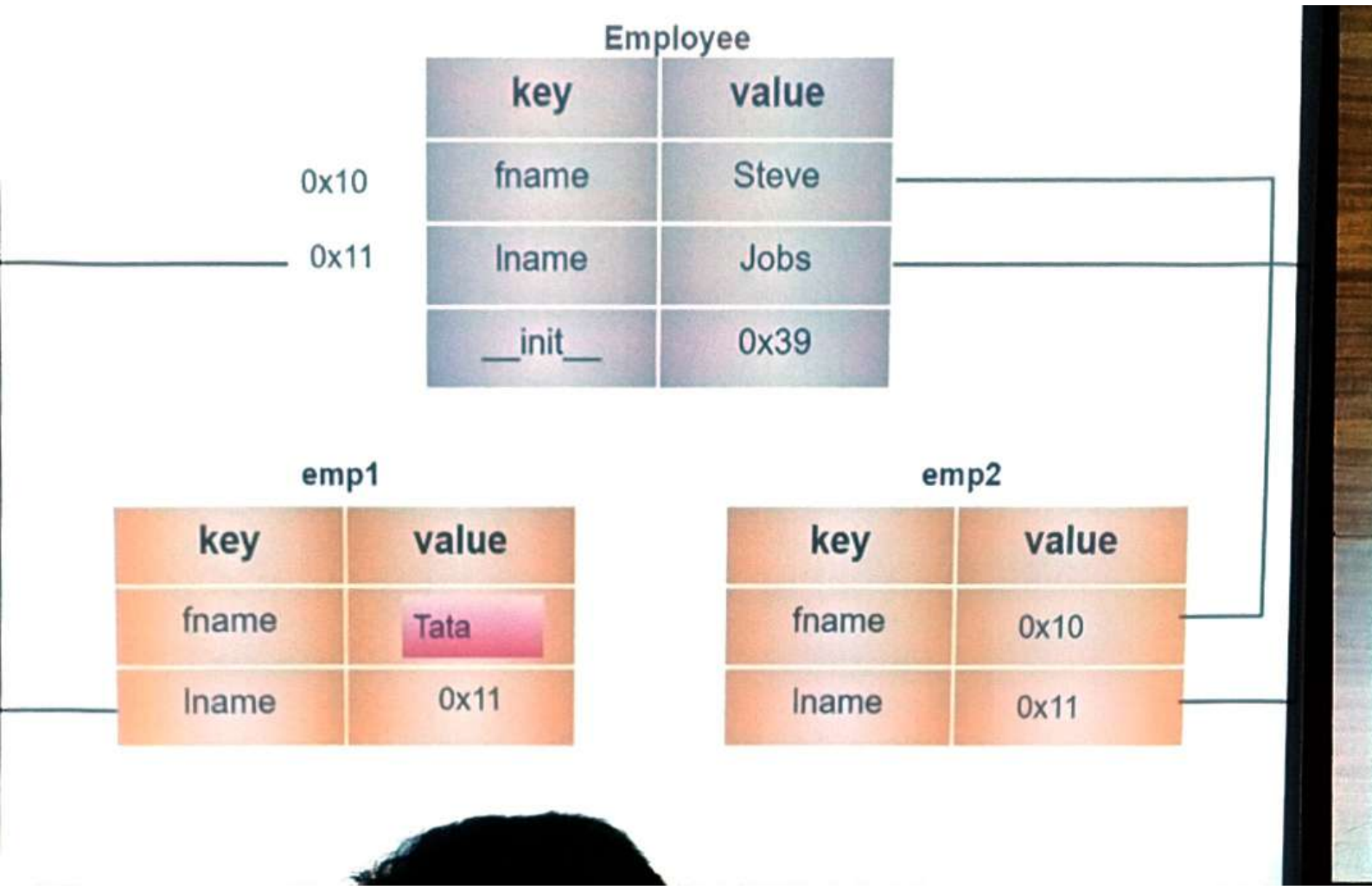
Example:

```
class Employee:
    def __init__(self, fname, lname):
        self.fname = fname
        self.lname = lname


emp1 = Employee("Steve", "Jobs")
emp2 = Employee("Tata", "Birla")
```

Example:

```
class Employee:
    fname = "Steve"
    lname = "Jobs"
    def __init__(self, fname, lname):
        self.fname = fname
        self.lname = lname


emp1 = Employee("Tata", "Birla")
emp2 = Employee("Mukesh", "Ambani")
```


Employee
key
value
fname
Steve
lname
Jobs
__init__
0x39
0x10
0x11
emp1
key
value
fname
0x10
lname
0x11
emp2
key
value
fname
0x10
lname
0x11


Employee
key
value
0x10
fname
Steve
0x11
lname
Jobs
__init__
0x39
emp1
key
value
fname
Tata
lname
0x11
emp2
key
value
fname
0x10
lname
0x11

Employee

	key	value
0x10	fname	Steve
0x11	lname	Jobs
	__init__	0x39

emp1

key	value
fname	Tata
lname	Birla

emp2

key	value
fname	Mukesh
lname	Ambani

NOTE:

1. In case of classes, when you look up for an attribute, Python tries to look for that attribute in the instance.
2. If the attribute exists in the instance, then it will return the value of the instance attribute.
3. If the attribute does not exist in the instance, it will lookup for the attribute at class level.
4. If the attribute exists in the class level, it will return the value of the class attribute.
5. If the attribute does not exist in instance and at class level,then **AttributeError** is raised.

Relationships

1. is a relationship (Inheritance)
2. has a relationship (Composition)

Is a relationship (Inheritance)

- It is one of the fundamental concepts of Object-Oriented Programming.
- In object-oriented programming, the concept of IS-A is a totally based on Inheritance, which can be of type Class Inheritance.
- It is just like saying "A is a B type of thing".

  eg: Apple is a Fruit, Car is a Vehicle etc.
- Inheritance is unidirectional. For example, House is a Building. But Building is not a House.

Has a relationship (Composition)

- It is one of the fundamental concepts of Object-Oriented Programming.
- In this concept, we will describe a class that references to one or more objects of other classes as an Instance variable.
- Here, by using the class name or by creating the object we can access the members of one class inside another class.
- It enables creating complex types by combining objects of different classes.
- It means that a class Composite can contain an object of another class Component.
- Composition(HAS-A) simply mean the use of instance variables that are references to other object

Inheritance

- Inheritance is the capability of one class to derive or inherit the properties from another class.
- Inheritance enables us to define a class that takes all the functionality from a parent class and allows us to add more.
- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class.

Types of inheritance

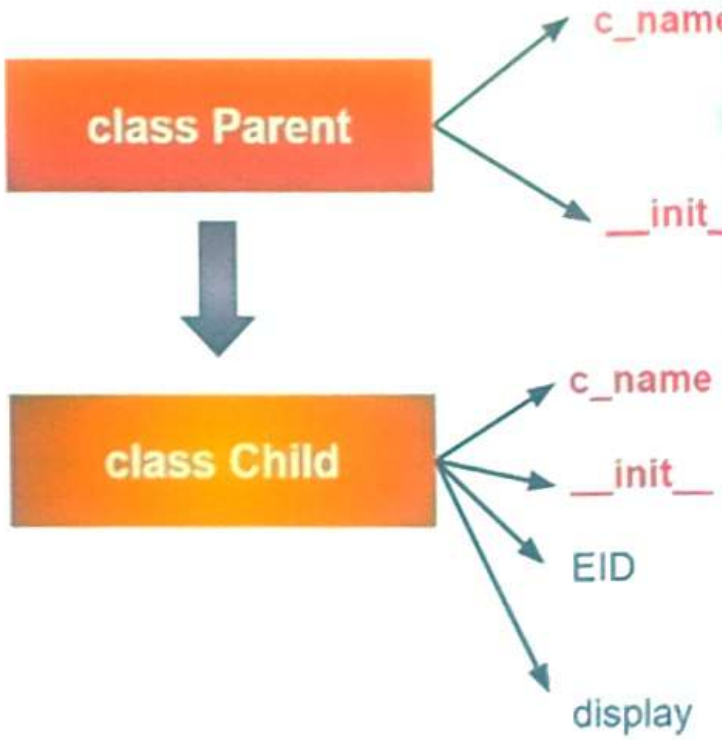
1. Single level inheritance.
2. Multi-level inheritance.
3. Multiple inheritance.
4. Hierarchical inheritance.

Single level inheritance

- When a child class inherits from only one parent class, it is called single inheritance.

```
class Company:
    c_name = "ABC"

    def __init__(self, fname, lname):
        self.fname = fname
        self.lname = lname


class Employee(Company):
    EID = 001

    def display(self):
        print(self.fname)
```

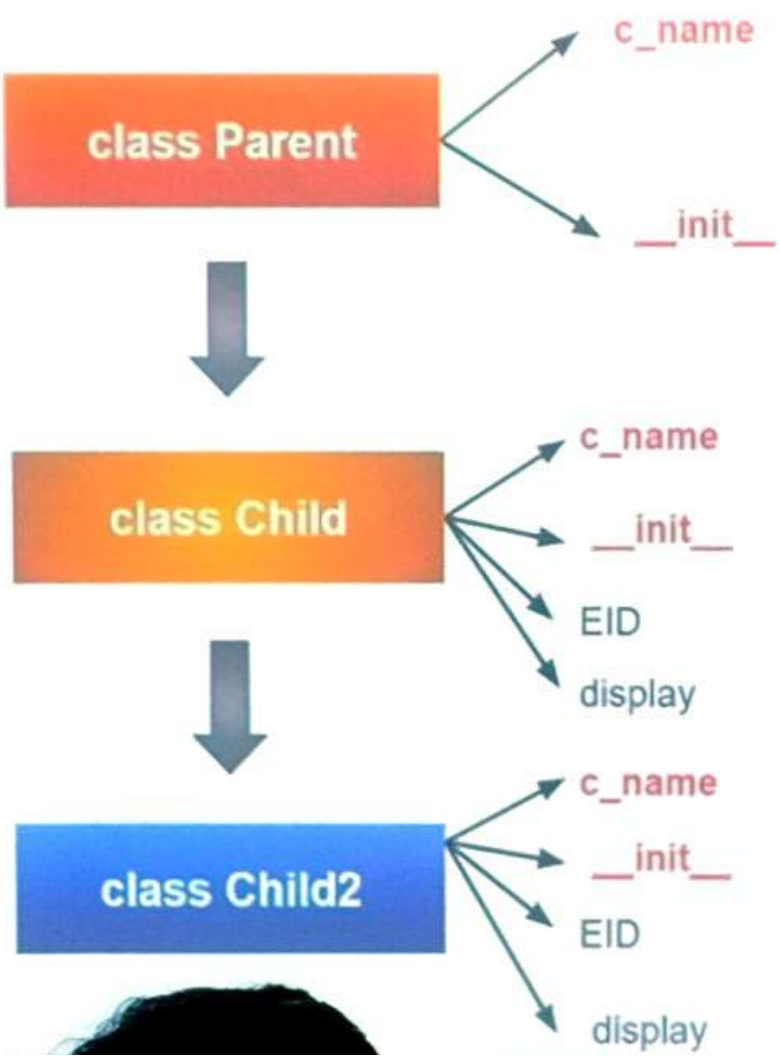
```
class Company:
    c_name = "ABC"

    def __init__(self, fname, lname):
        self.fname = fname
        self.lname = lname


class Employee(Company):
    EID = 001

    def display(self):
        print(self.fname)
```



Multi-level Inheritance

- Multi-level inheritance is achieved when a derived class inherits another derived class.
- There is no limit on the number of levels.

```
class Parent:
    c_name = "ABC"
    def __init__(self, fname, name):
        self.fname = fname
        self.lname = lname

class Child(Parent):
    EID = 001
    def display(self):
        print(self.fname)

class Child2(Child):
    pass
```

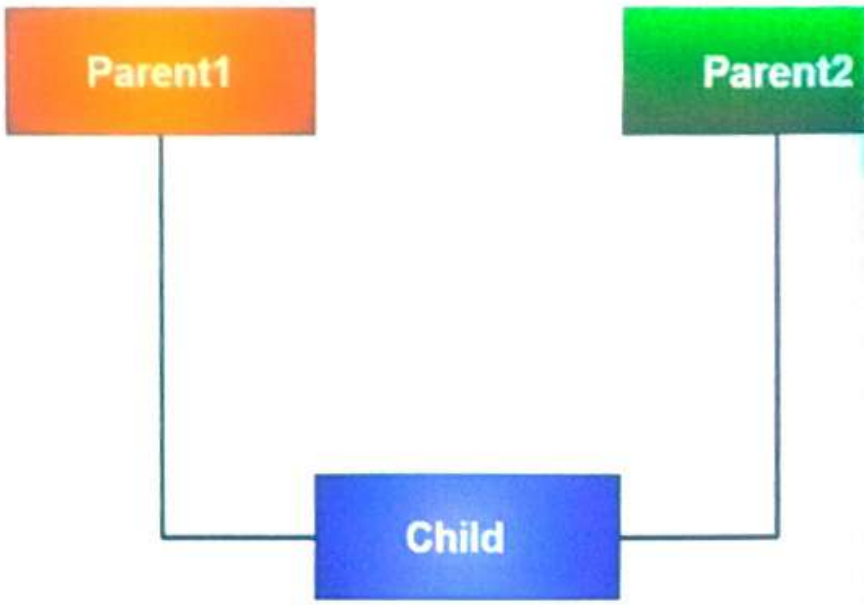

class Parent
c_name
__init__
class Child
c_name
__init__
EID
display
class Child2
c_name
__init__
EID
display

Multiple Inheritance

- Multiple inheritance is achieved when a derived class inherits from more than one Base class.

```
class Parent1:
    def spam(self):
        print('Child1.Spam')

class Parent2:
    def spam(self):
        print('Child2.Spam')

class Child(Parent1, Parent2):
    pass
```

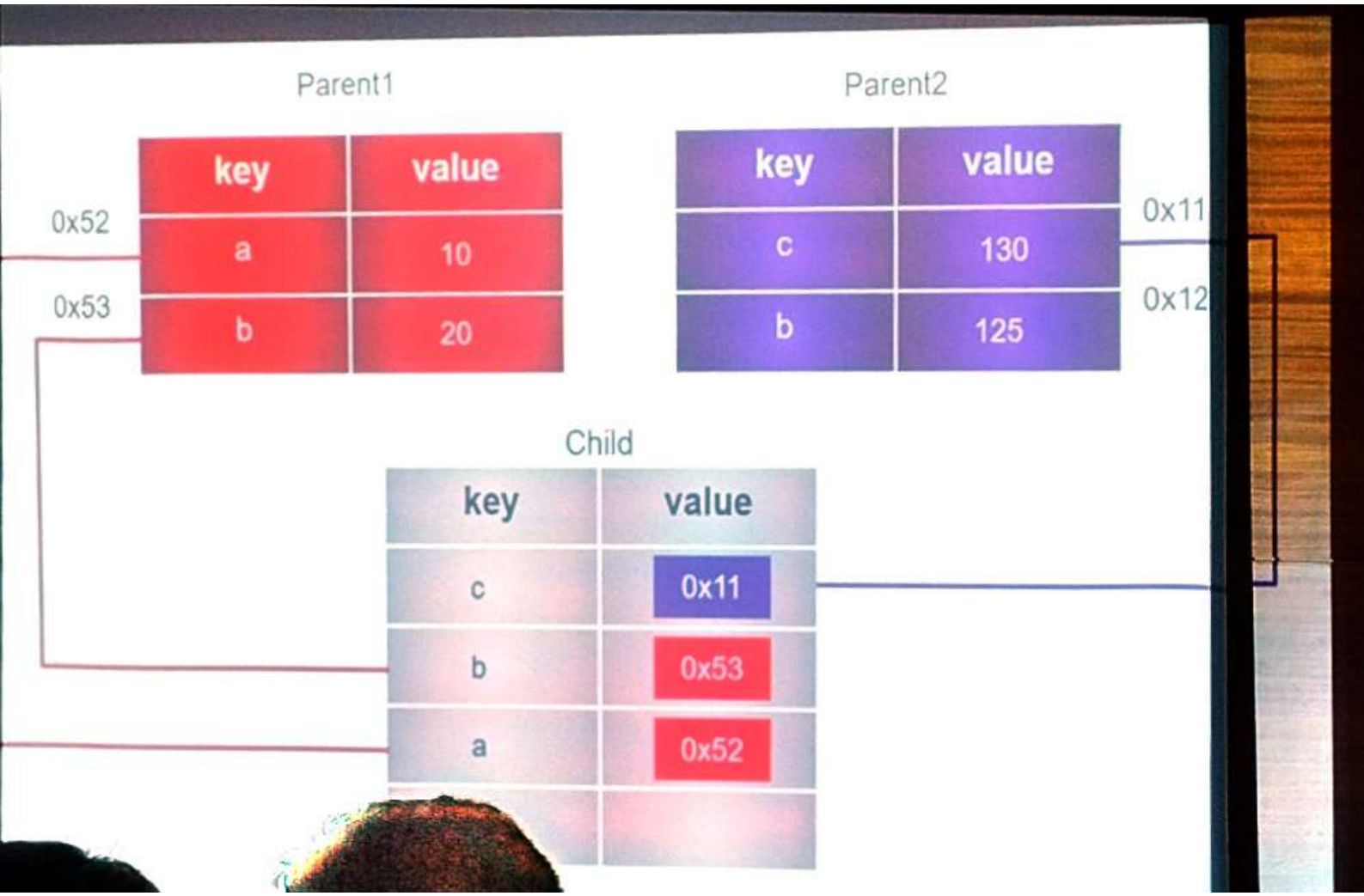
```
class Parent1:
    def spam(self):
        print('Child1.Spam')

class Parent2:
    def spam(self):
        print('Child2.Spam')

class Child(Parent1, Parent2):
    pass
```

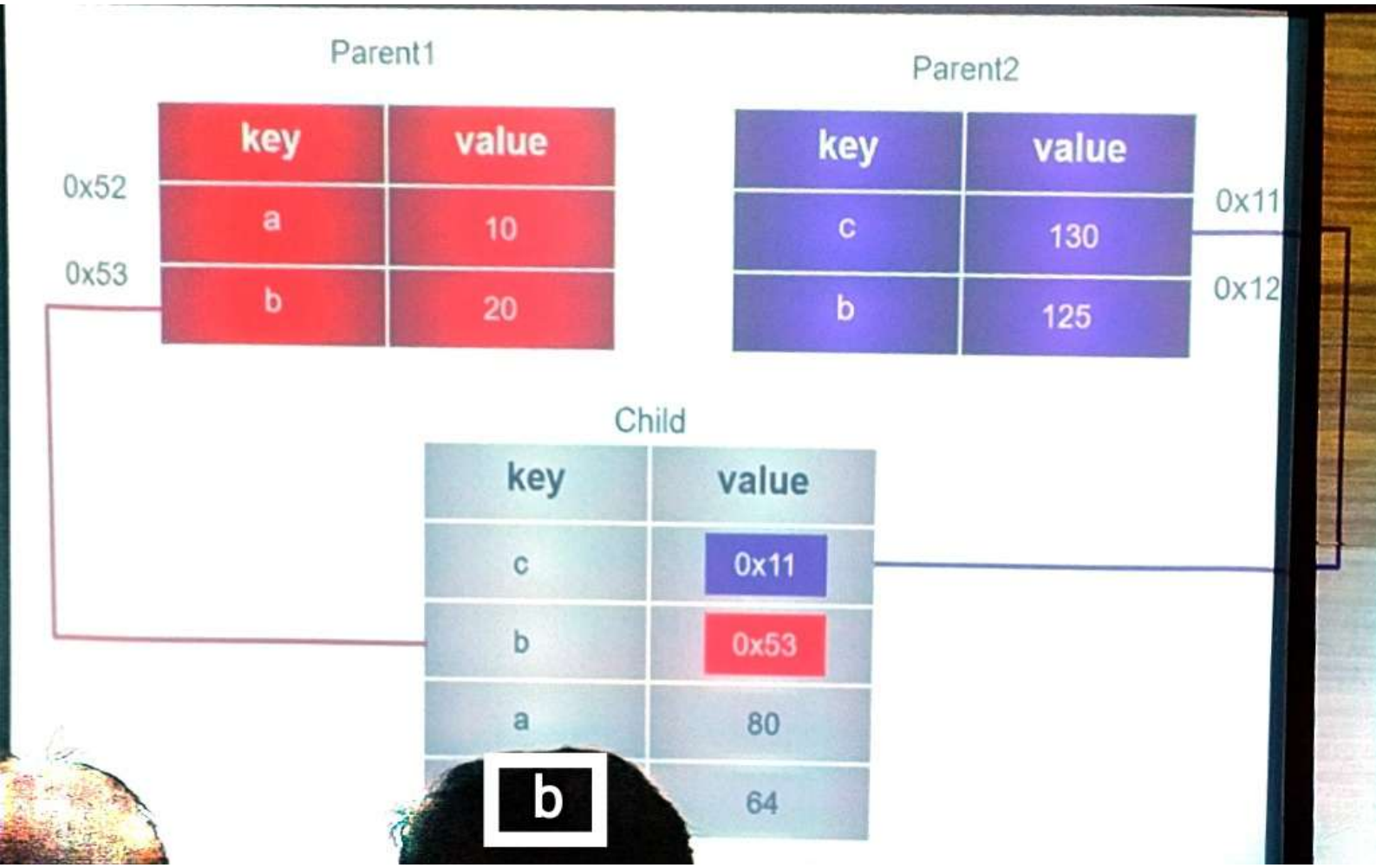


Order of inheritance: In multiple inheritance, the inheritance will take place from right to left. I.e., the rightmost class will be inherited first and the leftmost will be inherited last.

MRO(Method Resolution Order): MRO is the order followed to look up for an attribute in classes. In multiple inheritance the MRO will takes place from left to right.

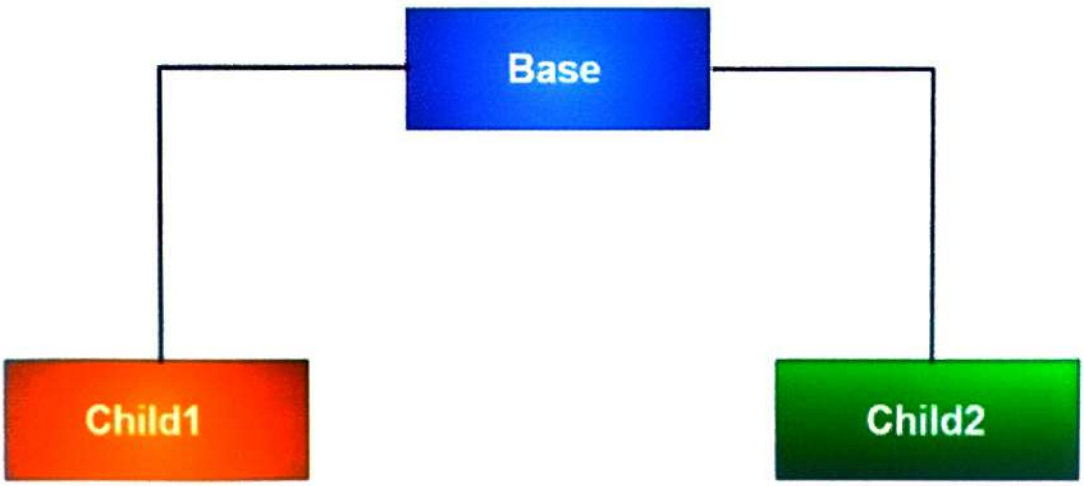
```
class Parent1:
    a = 10
    b = 20

class Parent2:
    c = 30
    b = 25

class Child(Parent1, Parent2):
    a = 80
    d = 64
```


Parent1
key
value
0x52
a
10
0x53
b
20
Parent2
key
value
0x11
c
130
0x12
b
125
Child
key
value
c
0x11
b
0x53
a
0x52


Parent1
key
value
0x52
a
10
0x53
b
20
Parent2
key
value
0x11
c
130
0x12
b
125
Child
key
value
c
0x11
b
0x53
a
80
64
b

Hierarchical Inheritance

The properties of a single base class will be inherited by many derived classes.



method/constructor chaining

- The process of calling parent class method after overriding it in the child class is called chaining.
- In order to achieve chaining, super() is used.

NOTE: MRO(Method Resolution Order) is a concept that is used to search the attributes in a class especially in inheritance hierarchy. Here MRO will be from left to right.

Eg: class A(B, C): -----> A → B → C