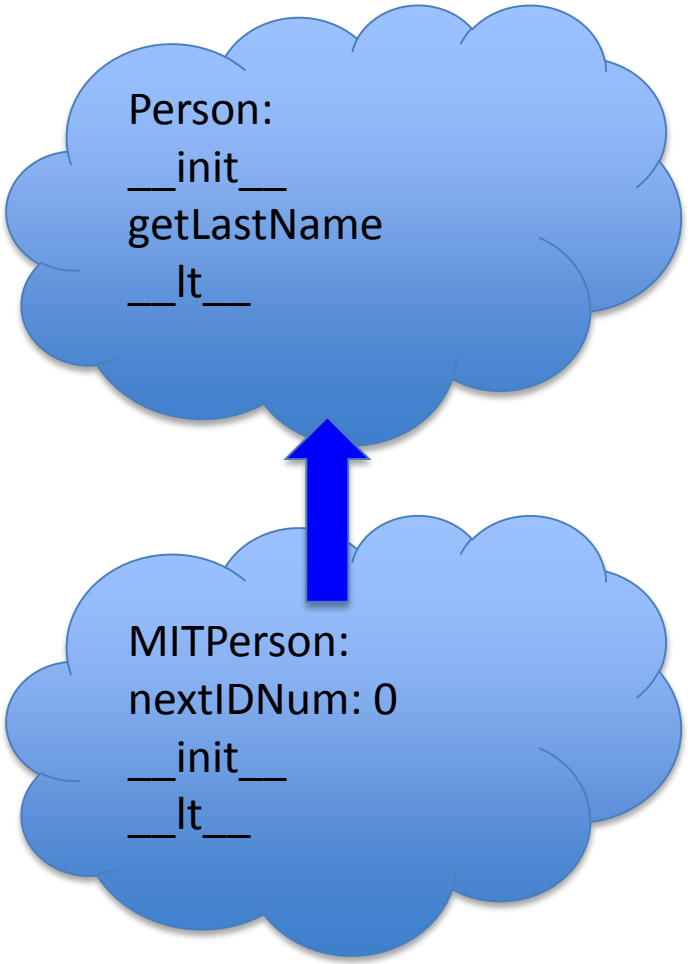
Using Inheritance

- Let's build an application that organizes info about people!
  - Person: name, birthday
    - Get last name
    - Sort by last name
    - Get age
  - MITPerson: Person + ID Number
    - Assign ID numbers in sequence
    - Get ID number
    - Sort by ID number

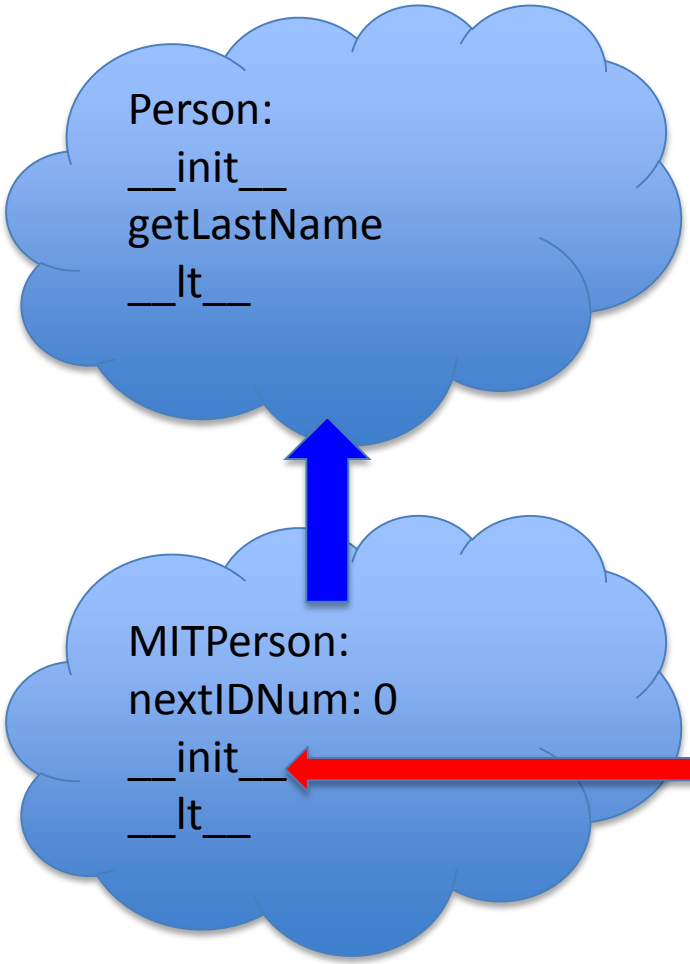
Building inheritance

```
class MITPerson(Person):
    nextIdNum = 0 # next ID number to assign

    def __init__(self, name):
        Person.__init__(self, name) # initialize Person
attributes
        # new MITPerson attribute: a unique ID number
        self.idNum = MITPerson.nextIdNum
        MITPerson.nextIdNum += 1

    def getIdNum(self):
        return self.idNum

    # sorting MIT people uses their ID number, not name!
    def __lt__(self, other):
        return self.idNum < other.idNum
```

Visualizing the hierarchy

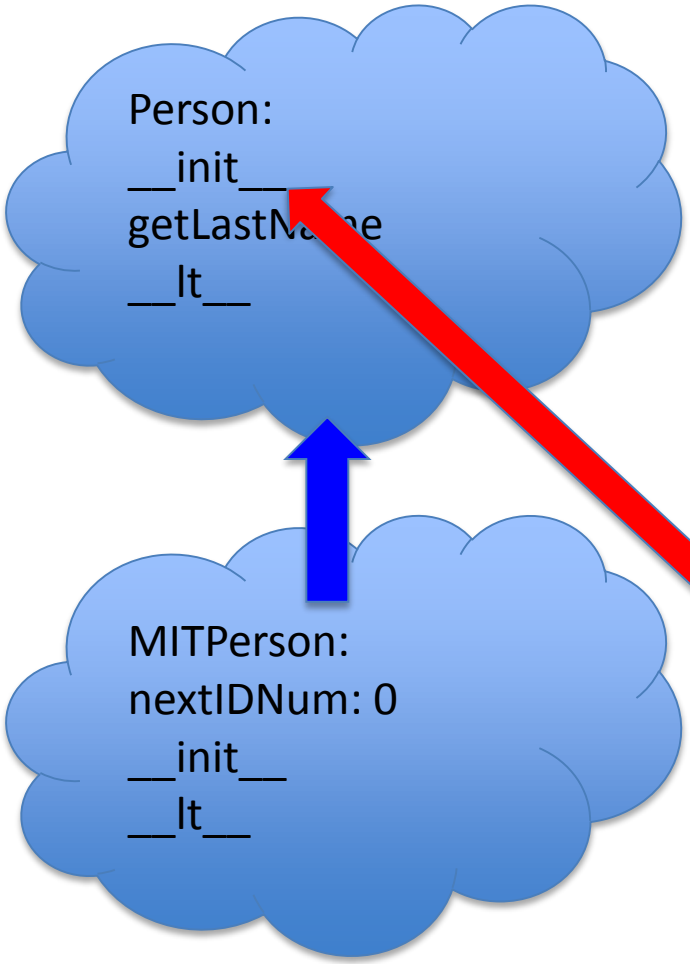


Visualizing the hierarchy



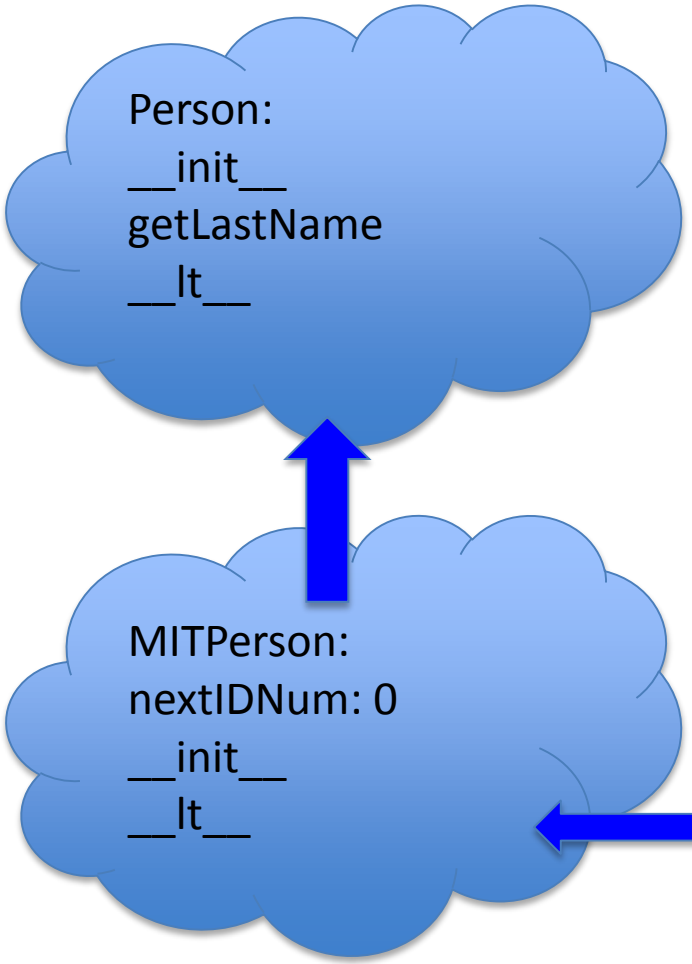
Calling MITPerson will use this __init__ procedure (because that is the one visible in MITPerson's environment)

Visualizing the hierarchy



That code uses
Person.__init__
which will in turn call this
procedure

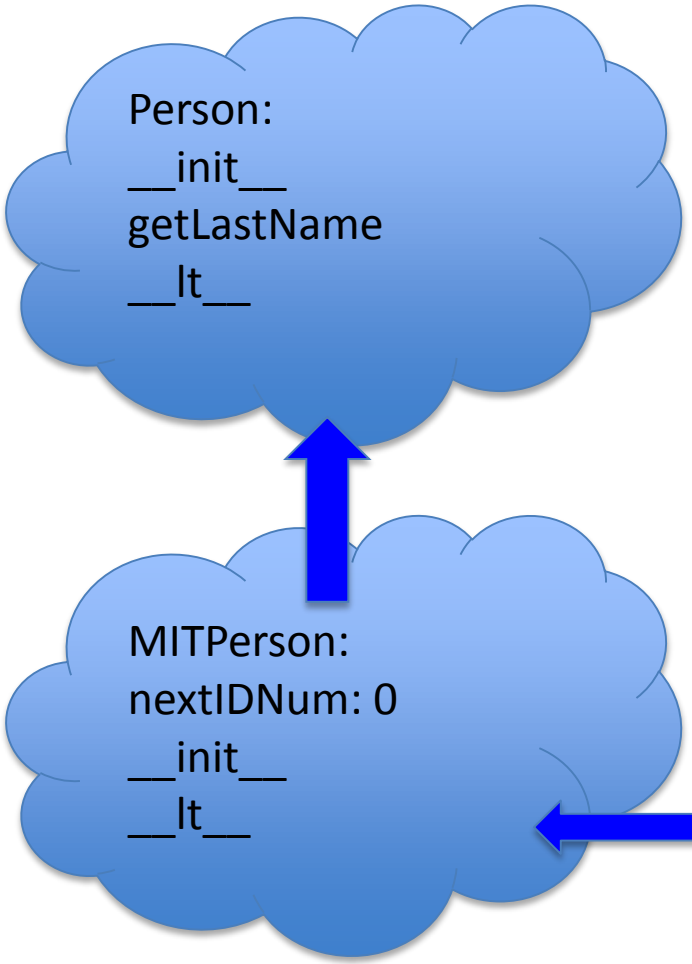
Visualizing the hierarchy



name	
birthday	
lastName	

And that creates an instance of MITPerson (because of the first call, which inherits from the class definition) but with bindings set by the inherit __init__ code

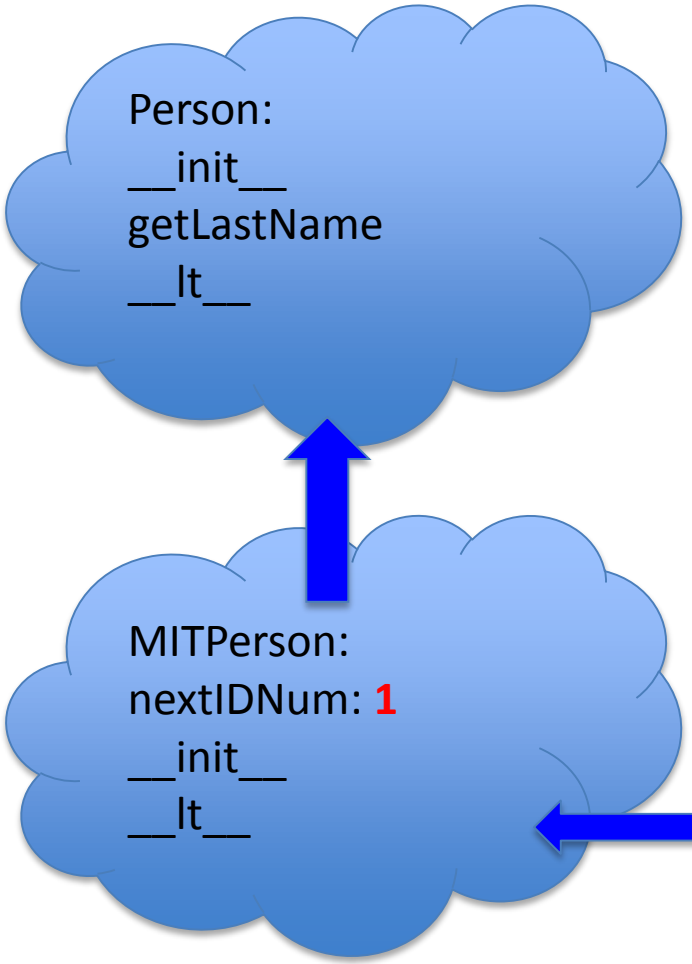
Visualizing the hierarchy



The rest of the original __init__ code calls self.idNum = MITPerson.nextIdNum which looks up nextIdNum in the MITPerson environment, and creates a binding in self (i.e. the instance)

name	
birthday	
lastName	
idNum	0

Visualizing the hierarchy



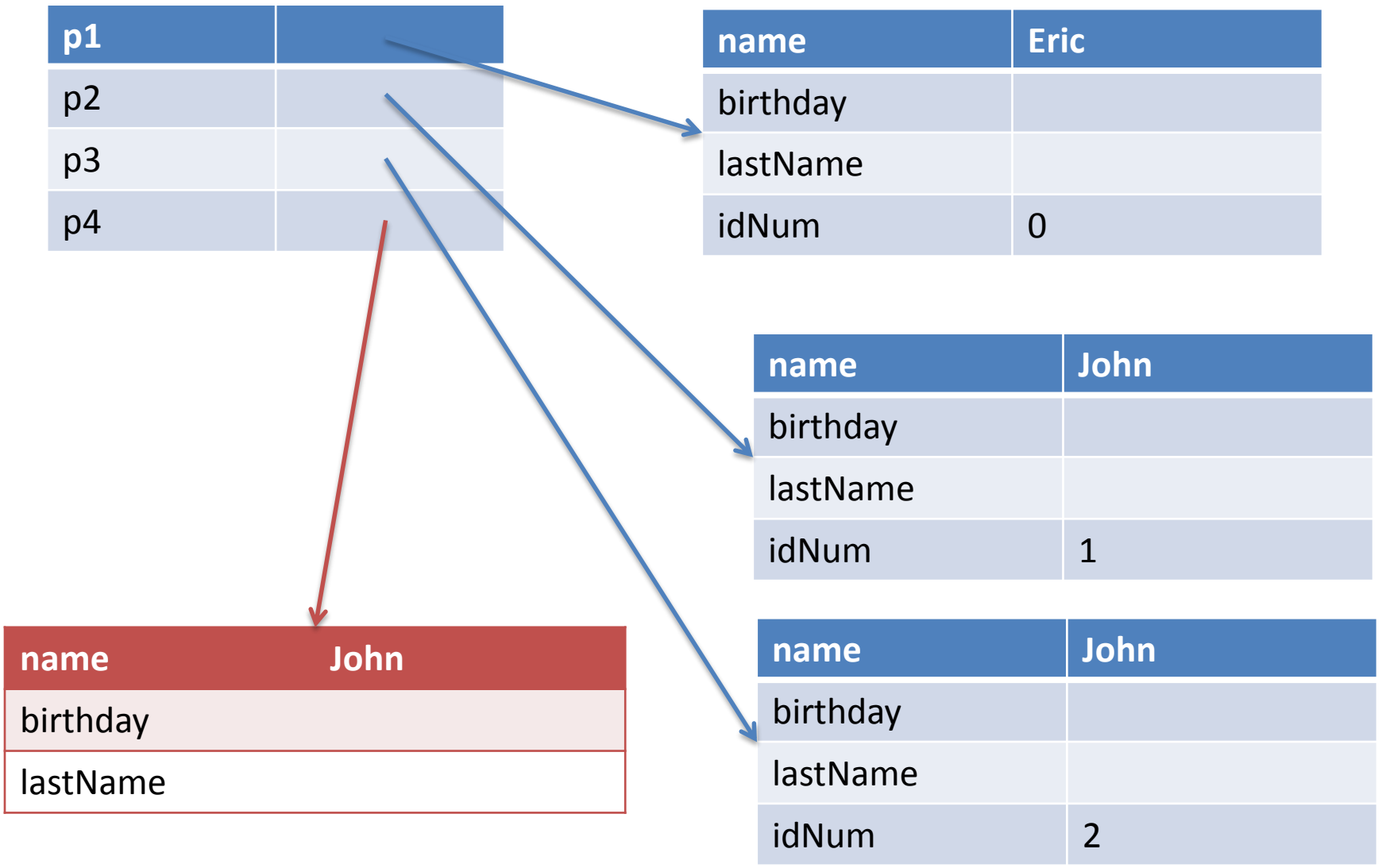
The rest of the original __init__ code calls
self.idNum = MITPerson.nextIdNum
which looks up nextIdNum in the MITPerson environment, and creates a binding in self (i.e. the instance)
And then updates nextIdNum in the MITPerson environment

name	
birthday	
lastName	
idNum	0

Examples of using this hierarchy

```
p1 = MITPerson('Eric')
p2 = MITPerson('John')
p3 = MITPerson('John')
p4 = Person('John')
```

Visualizing the hierarchy



Suppose we want to compare things

- Note that MITPerson has its own `__lt__` method
- This method "shadows" the Person method, meaning that if we compare an MITPerson object, since its environment inherits from the MITPerson class environment, Python will see this version of `__lt__` not the Person version
- Thus, `p1 < p2` will be converted into `p1.__lt__(p2)` which applies the method associated with the type of `p1`, or the MITPerson version

Who inherits

- Why does p4 < p1 work, but p1 < p4 doesn't?
  - p4 < p1 is equivalent to p4.__lt__(p1), which means we use the __lt__ method associated with the type of p4, namely a Person (the one that compares based on name)
  - p1 < p4 is equivalent to p1.__lt__(p4), which means we use the __lt__ method associated with the type of p1, namely an MITPerson (the one that compares based on IDNum) and since p4 is a Person, it does not have an IDNum