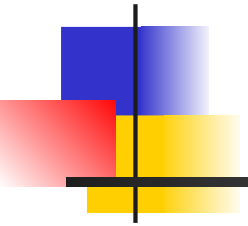


Object References

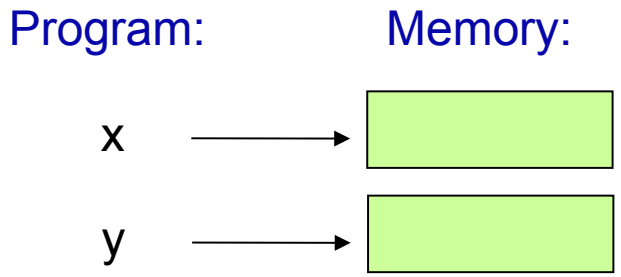
James Brucker

Variables

A **variable** is a name we use to *refer* to a **memory location**.

What is stored in the memory location?

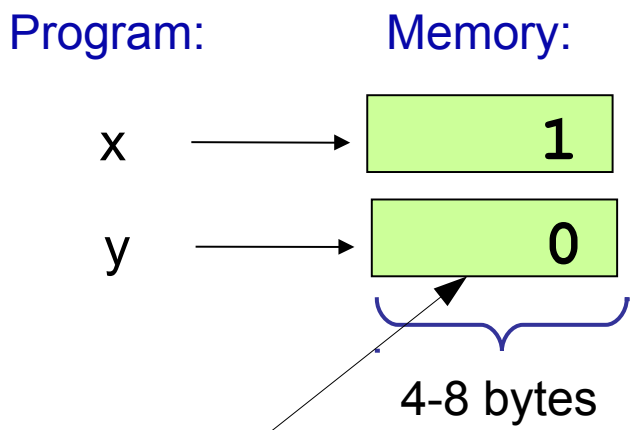
```
x = 1
y = "hello nerd"
```



Java, C, C#: values versus references

In some languages, "primitive" types like int or character are stored as values. Everything else is a reference to an object or data structure.

```
// Java
int x = 1;
String y;
```



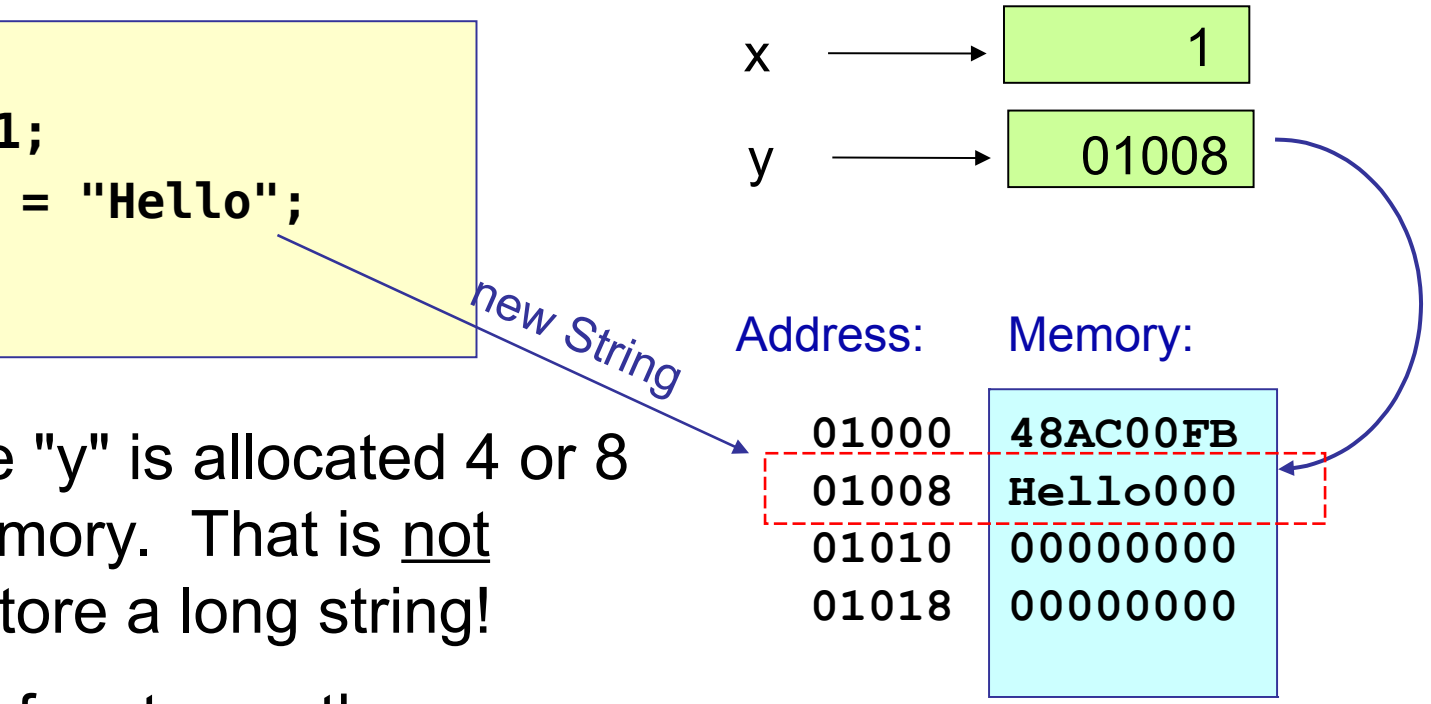
y is a reference to a String object.
A value `0` means `null` or `None` (no object).

References refer to another location

Memory for objects is allocated on the "heap".

When you assign an object to a variable, it refers to its location.

```
// Java
int x = 1;
String y = "Hello";
```



The variable "y" is allocated 4 or 8 bytes of memory. That is not enough to store a long string!

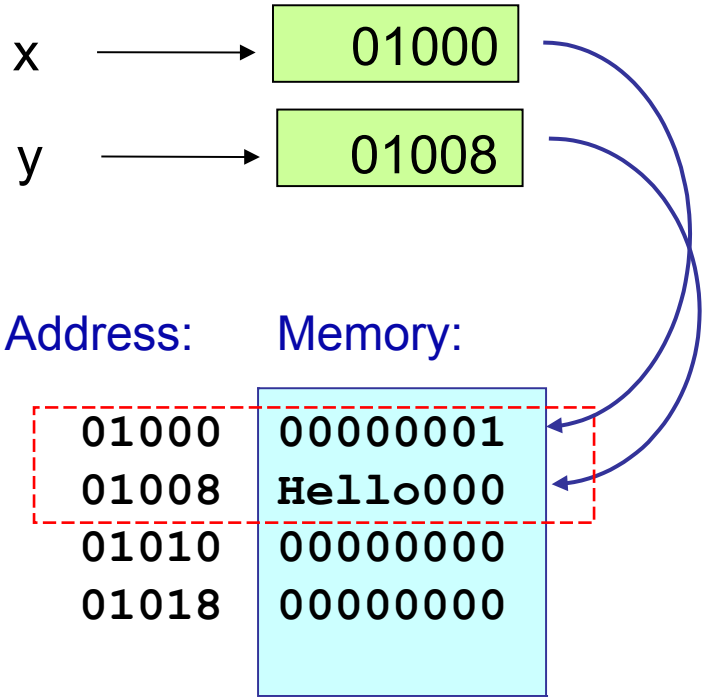
Instead, y refers to another address where the string object is located.

Python: Everything is an object

In Python, (almost) everything is a reference type.

Even `int` and `float` refer to objects.

```
# Python
x = 1
y = "Hello"
```



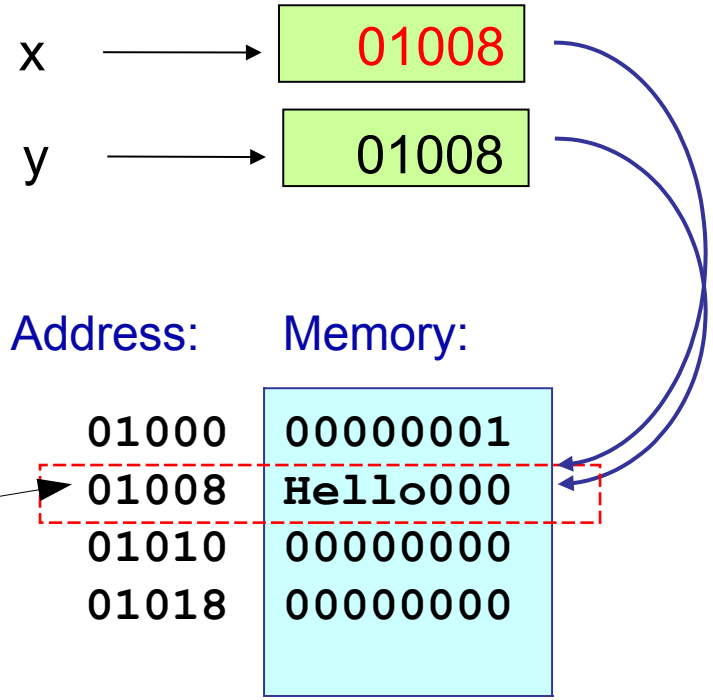
The variable "y" is allocated 4 or 8 bytes of memory. That is not enough to store a long string!

Instead, y refers to another address where the string object is located.

Assignment changes only the reference

Assignment "x = y" makes x refer to the same location that y refers to. It does not create a copy of the object.

```
# Python
x = 1
y = "Hello"
x = y
```



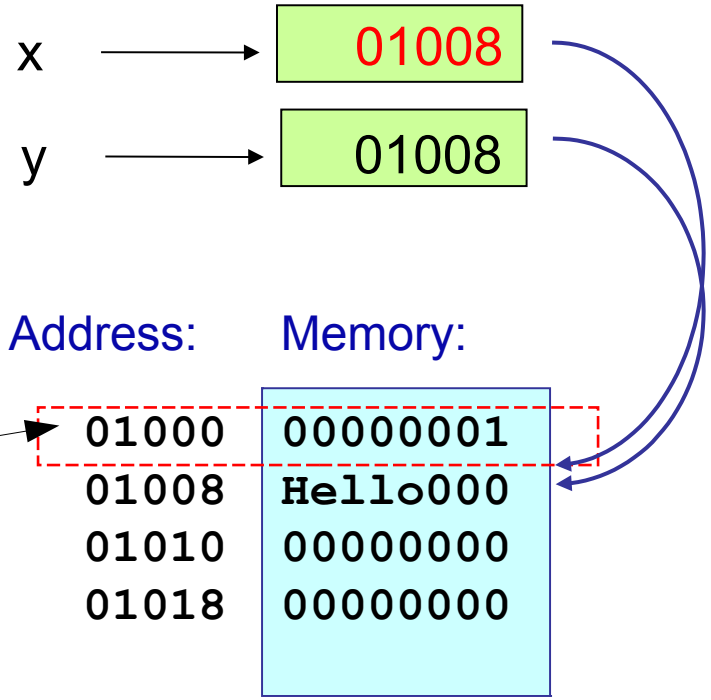
Now x and y refer to ("point to") the same object.

Garbage: unreferenced objects

After this code executes, x & y refer to the same string.

Nothing refers to the original "int" object (x = 1).

```
# Python
x = 1
y = "Hello"
x = y
```



This "int" object is still in memory, but nothing refers to it.

It's useless!

Garbage.

Lots of Garbage

How many string objects does this function create?

```
def copy_string(s: str, count: int) -> str:
    """copy a string count times, return the result"""
    result = ""
    while count > 0:
        result += s   # only for demo, this is stupid
        count -= 1
    return result

s = copy_string("hello ", 10)
print(s)
```

Use id(x) to show the object location

`id(x)` returns the virtual address of the object x refers to.

```
def copy_string(s: str, count: int) -> str:
    """copy a string count times, return the result"""
    result = ""
    while count > 0:
        result += s
        print("result address is ", id(result))
        count -= 1
    return result

s = copy_string("hello ", 10)
print(s)
```

Example output

Do you notice anything strange about the addresses?

```
> python3 copy_str.py
address result is 139884762706640
address result is 139884736673392
address result is 139884736704920
address result is 139884762529344
address result is 139884762529344
address result is 139884736618808
address result is 139884736656944
address result is 139884762520152
address result is 139884762520152
address result is 139884736569840
hello hello hello hello hello hello hello hello hello hello
```

Strings and numbers are special cases

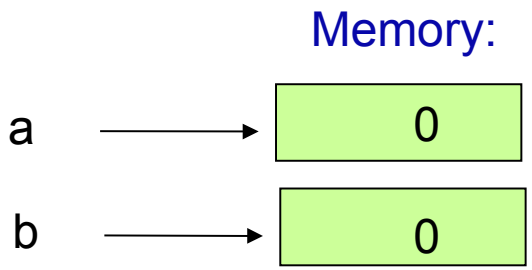
String and numbers are used so much that most languages have special handling to reduce object creation.

A better example would be a less-used type such as datetime.date

Another Example: BankAccount

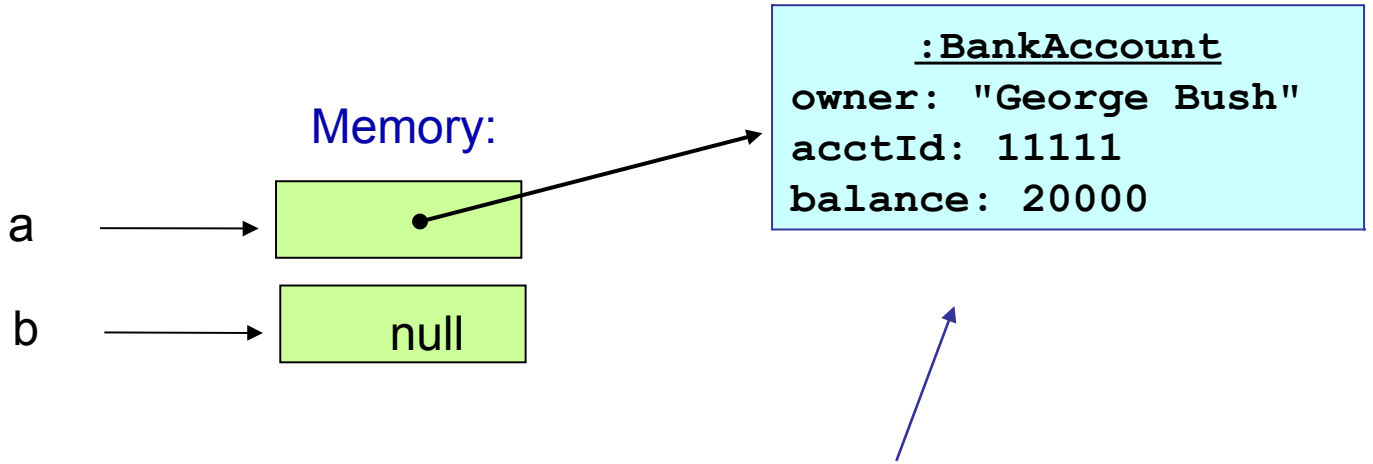
```
a: BankAccount
b: BankAccount
```

This declares a and b as BankAccount *references*, but does not create any BankAccount *objects*.



Create a BankAccount

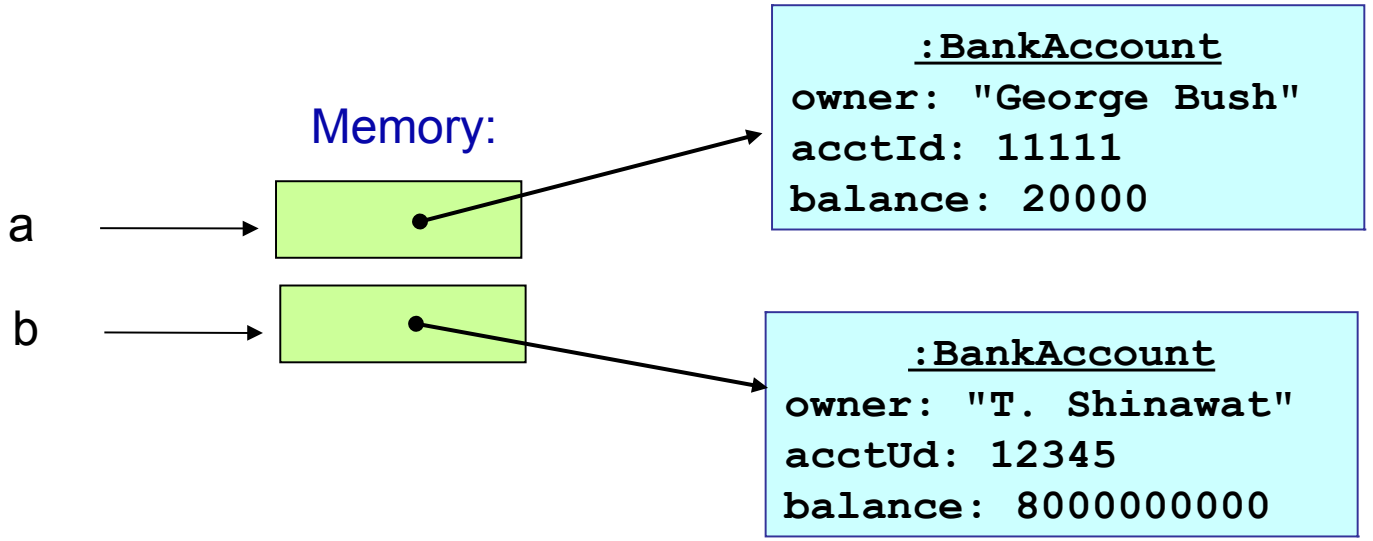
```
a = BankAccount("George Bush", 11111)
a.deposit(20000)
```



A conceptual view of the object. In memory, only the data values & a reference to the BankAccount class are stored.

Create Another Bank Account

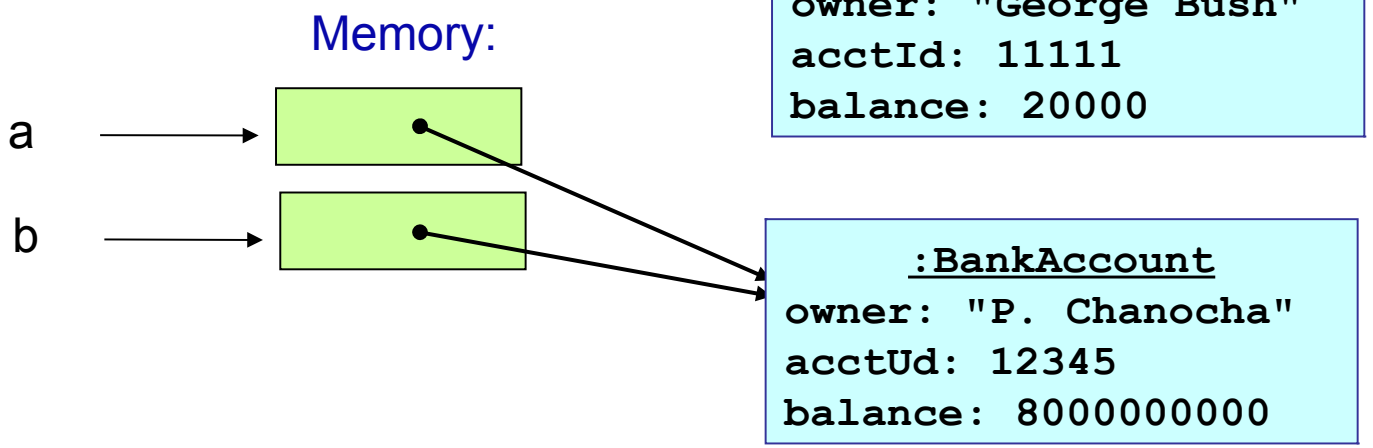
```
b = BankAccount("T. Shinawat", 12345)
b.deposit(8000000000)
```



assign a = b: copy or reference?

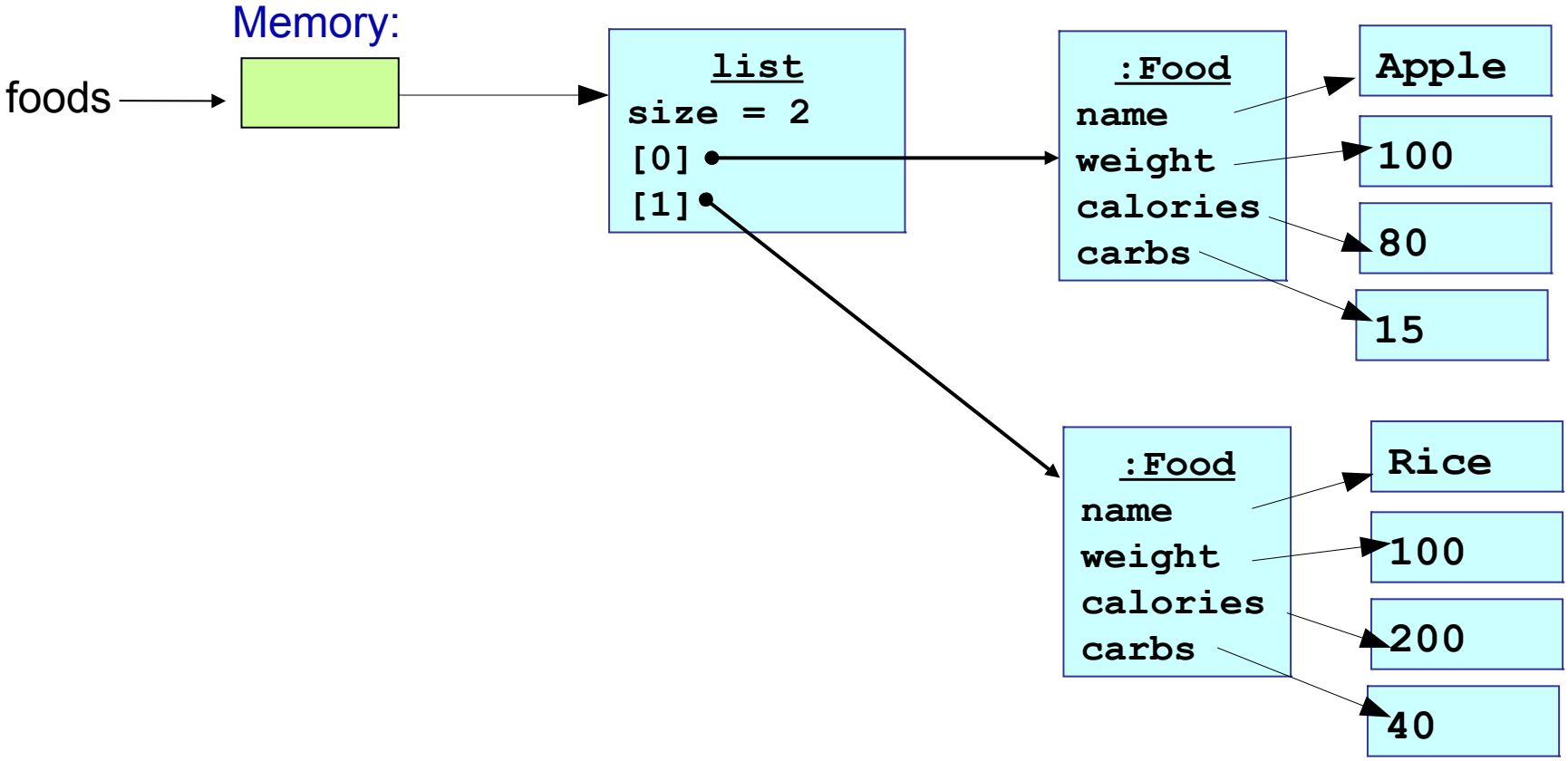
```
a = b
a.set_owner("P. Chanocha")
```

No copy! It makes **a** "point to" the same object as **b**.



What about Lists or Arrays?

```
foods = [Food("Apple", 100, 80, 15),
         Food("Rice", 100, 200, 40) ]
```



Try This

```
foods = [Food("Apple", 100, 80, 15),
         Food("Rice", 100, 200, 40)]
rice = foods[1]
print(rice)
'Rice (100g)'
# double the amount of rice!
rice = rice + rice
print(rice)
'Rice (200g)'
# What is output?
print( foods[1] )
???
```

Addition creates a new object

To save space, Food attributes are shown as values instead of references (conceptual view).