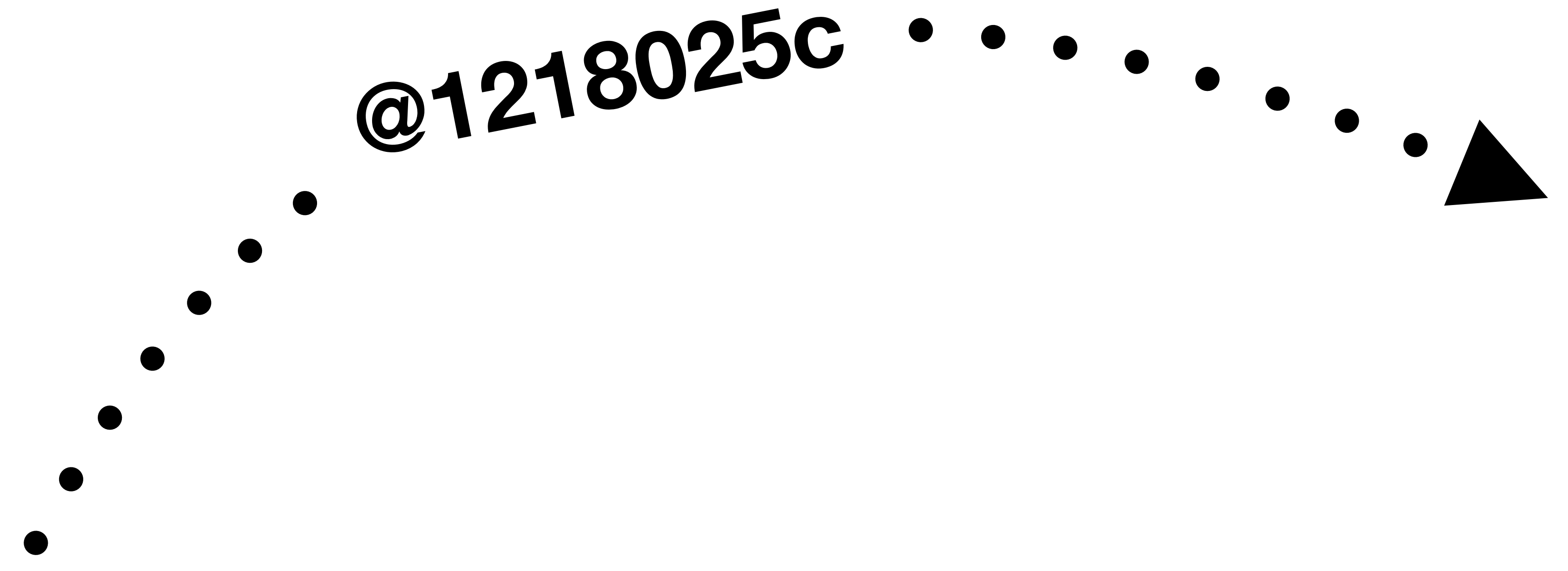
Essential Computing 1

Value & reference type

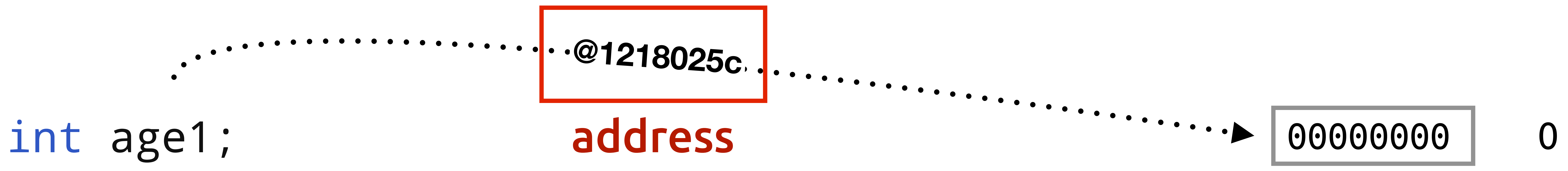


Recall that **integer is a "primitive data type"** (just like boolean, long, double, float and char)

```
int age1;
```

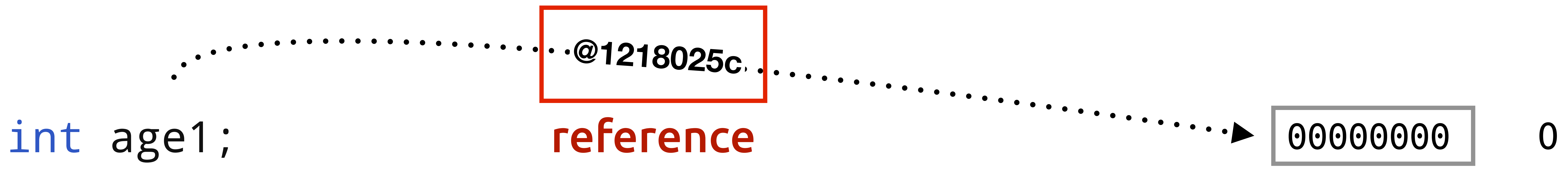
$\boxed{\texttt{00000000}}$ 0

Also recall that **a variable holds an address to data**



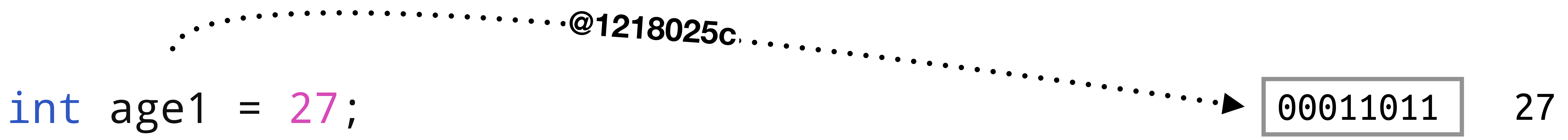
```
int age1;
```

Also recall that **a variable holds ~~an address~~ a reference to data**



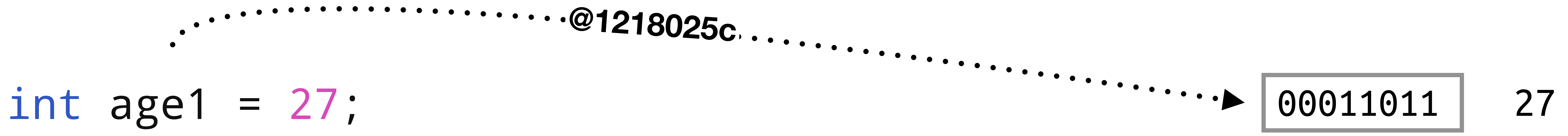
```
int age1;
```

… and **the reference is used to access the data**



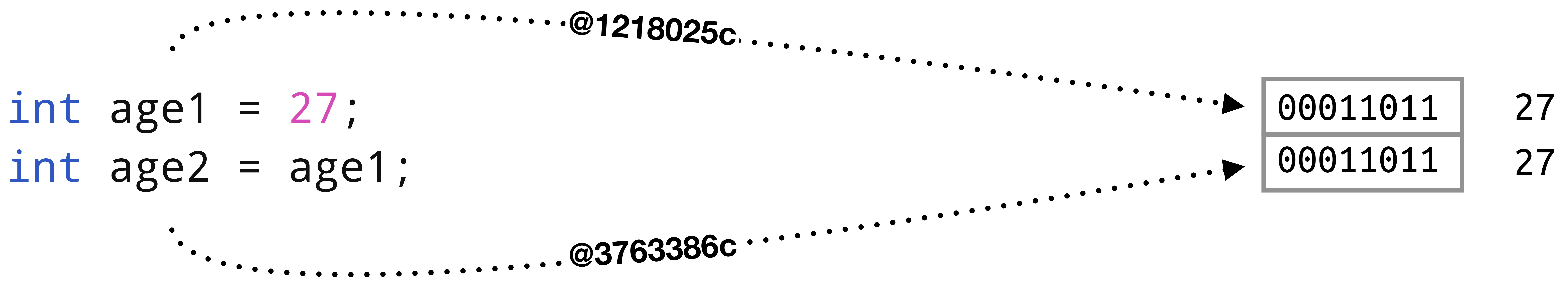
```
int age1 = 27;
```

News: **Primitive datatypes and strings are "value type"**



```
int age1 = 27;
```

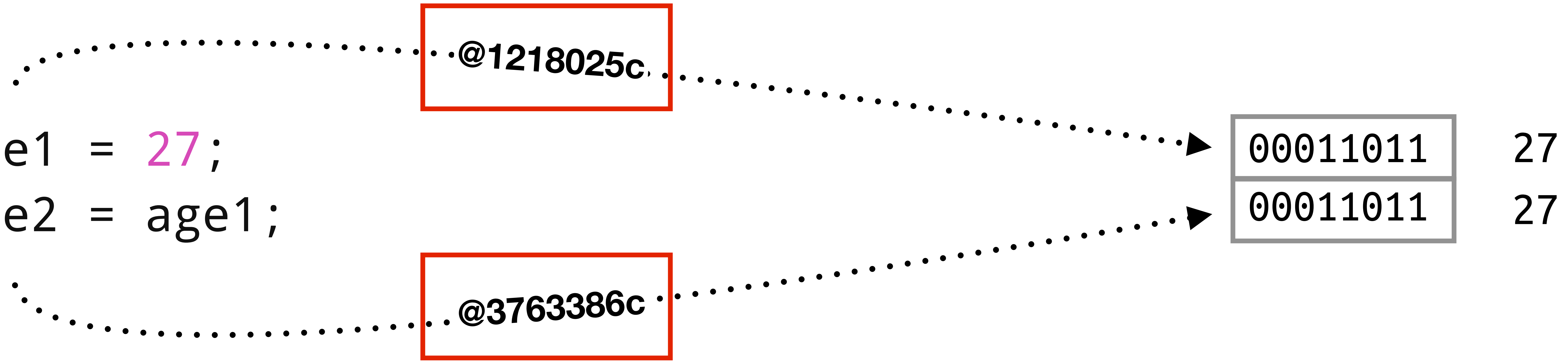
… meaning: **Copying a variable will copy the data**



```
int age1 = 27;
int age2 = age1;
```

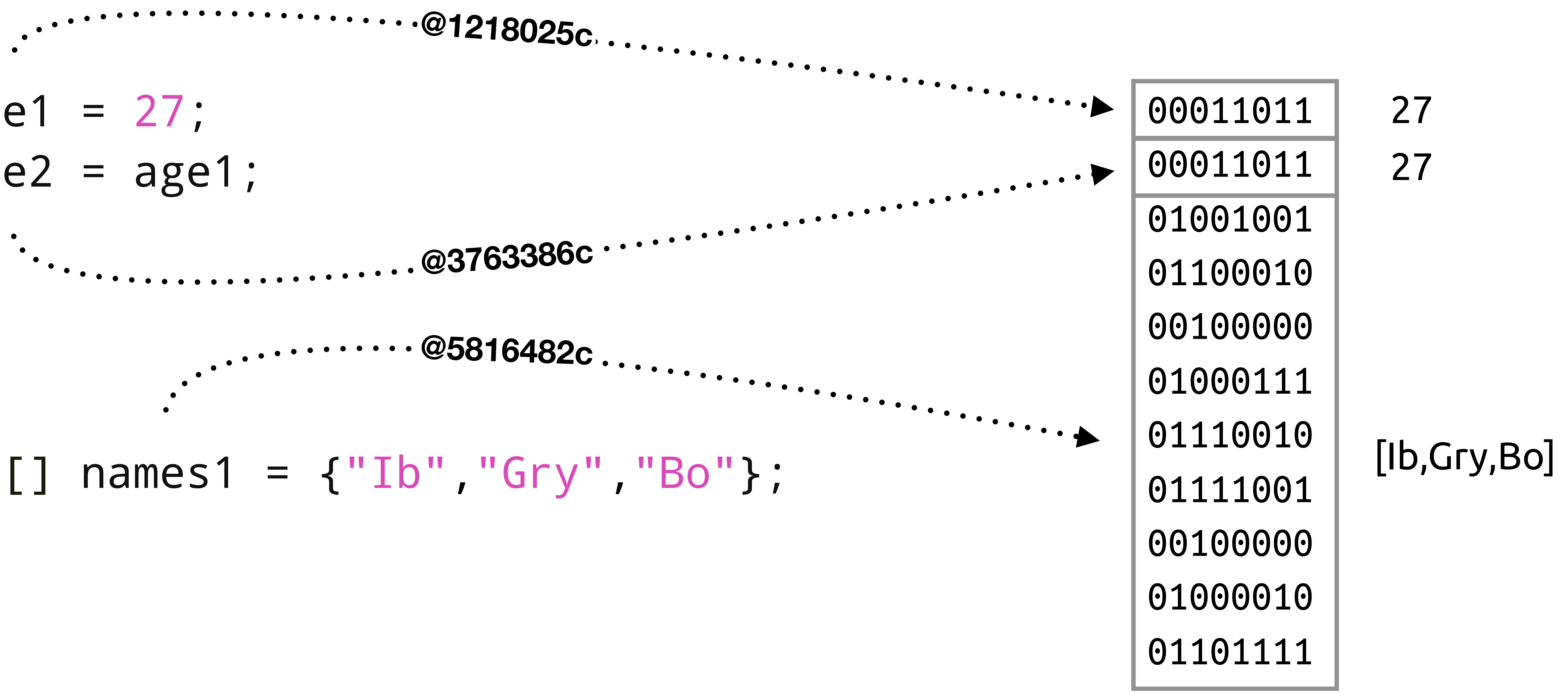
Result: **two variables with two different references**

```
int age1 = 27;
int age2 = age1;
```



Objects (including arrays) are **reference type**

```
int age1 = 27;
int age2 = age1;

String[] names1 = {"Ib","Gry","Bo"};
```

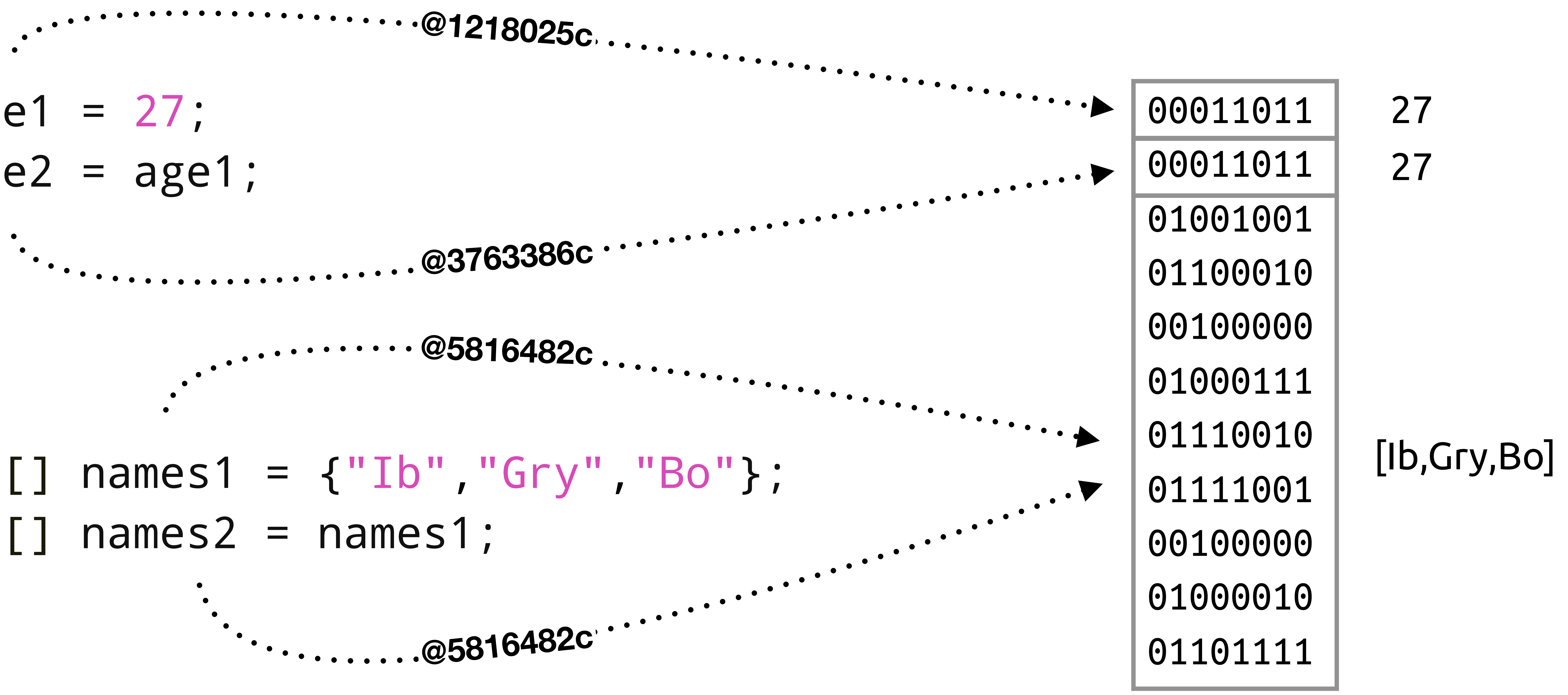


```
00011011
00011011
01001001
01100010
00100000
01000111
01110010
01111001
00100000
01000010
01101111
```

… meaning: **Copying a variable will copy the reference!**

```java
int age1 = 27;
int age2 = age1;
```

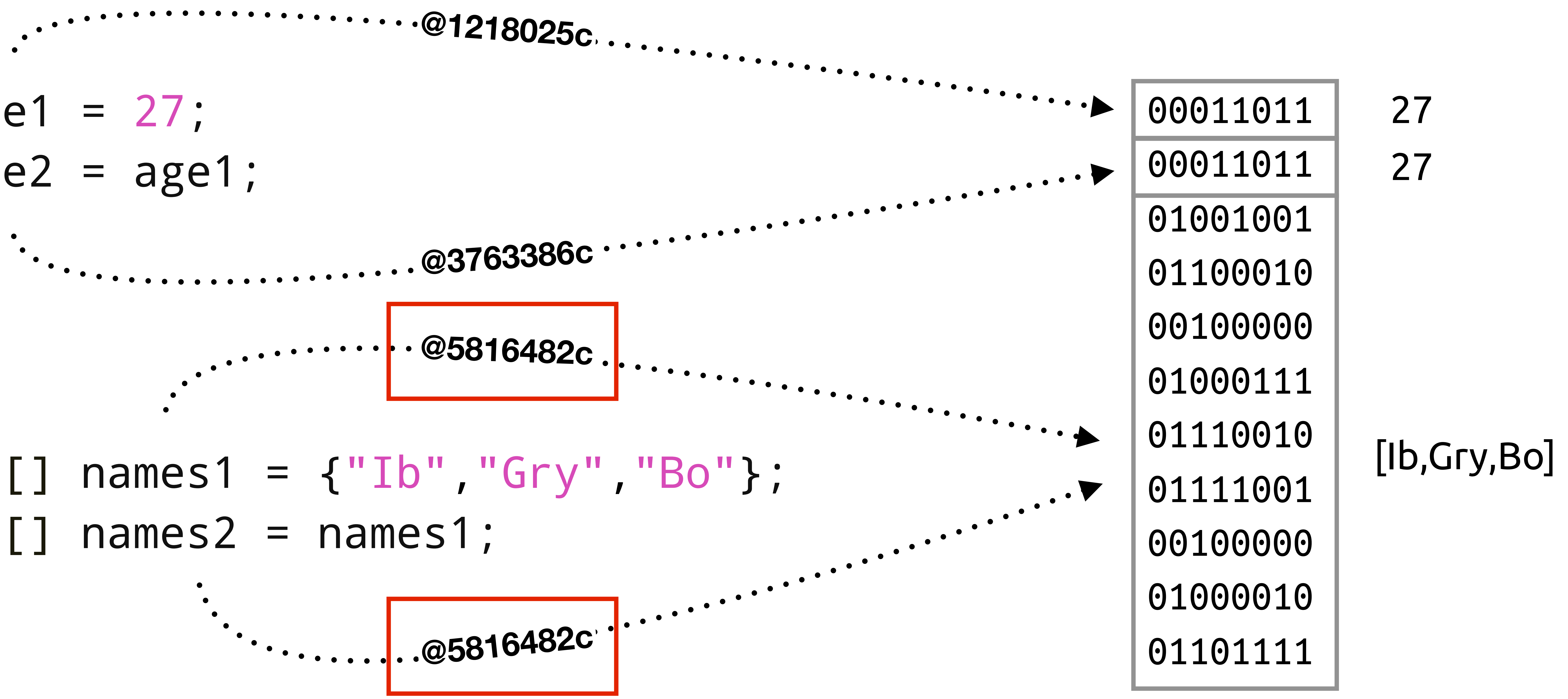
```java
String[] names1 = {"Ib","Gry","Bo"};
String[] names2 = names1;
```



Result: **Two variables with same reference (same data)!**

```
int age1 = 27;
int age2 = age1;
```

```
String[] names1 = {"Ib","Gry","Bo"};
String[] names2 = names1;
```



Consequences ....

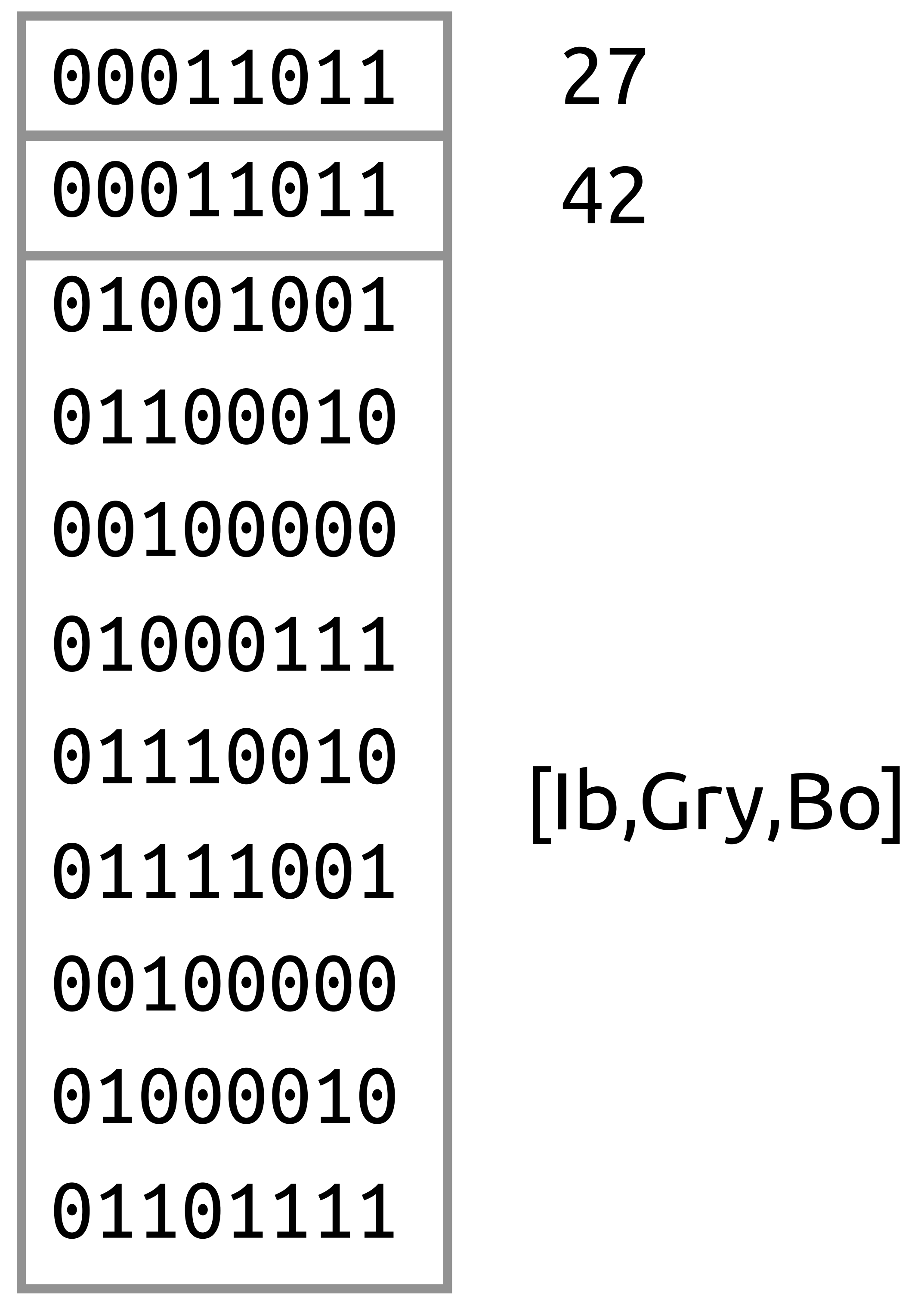
```java
int age1 = 27;
int age2 = age1;



String[] names1 = {"Ib","Gry","Bo"};
String[] names2 = names1;
```

00011011	27
00011011	27
01001001	
01100010	
00100000	
01000111	
01110010	[Ib,Gry,Bo]
01111001	
00100000	
01000010	
01101111	

Updating age2 behaves as expected

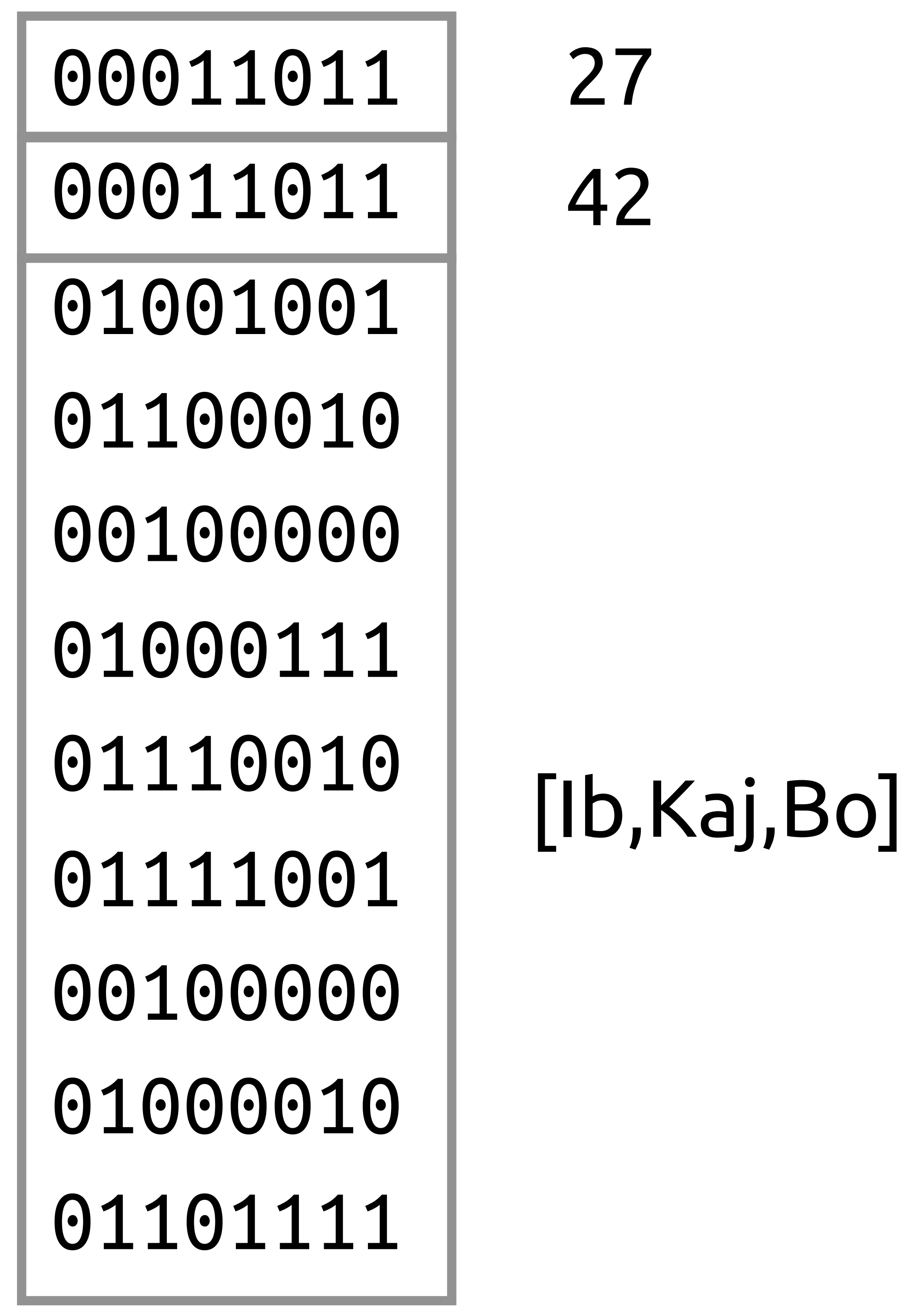
```
int age1 = 27;
int age2 = age1;
age2 = 42;
```

```
String[] names1 = {"Ib","Gry","Bo"};
String[] names2 = names1;
```



… but, **updating names2 also updates names1!**

```
int age1 = 27;
int age2 = age1;
age2 = 42;



String[] names1 = {"Ib","Gry","Bo"};
String[] names2 = names1;
names2[1] = "Kaj";
```



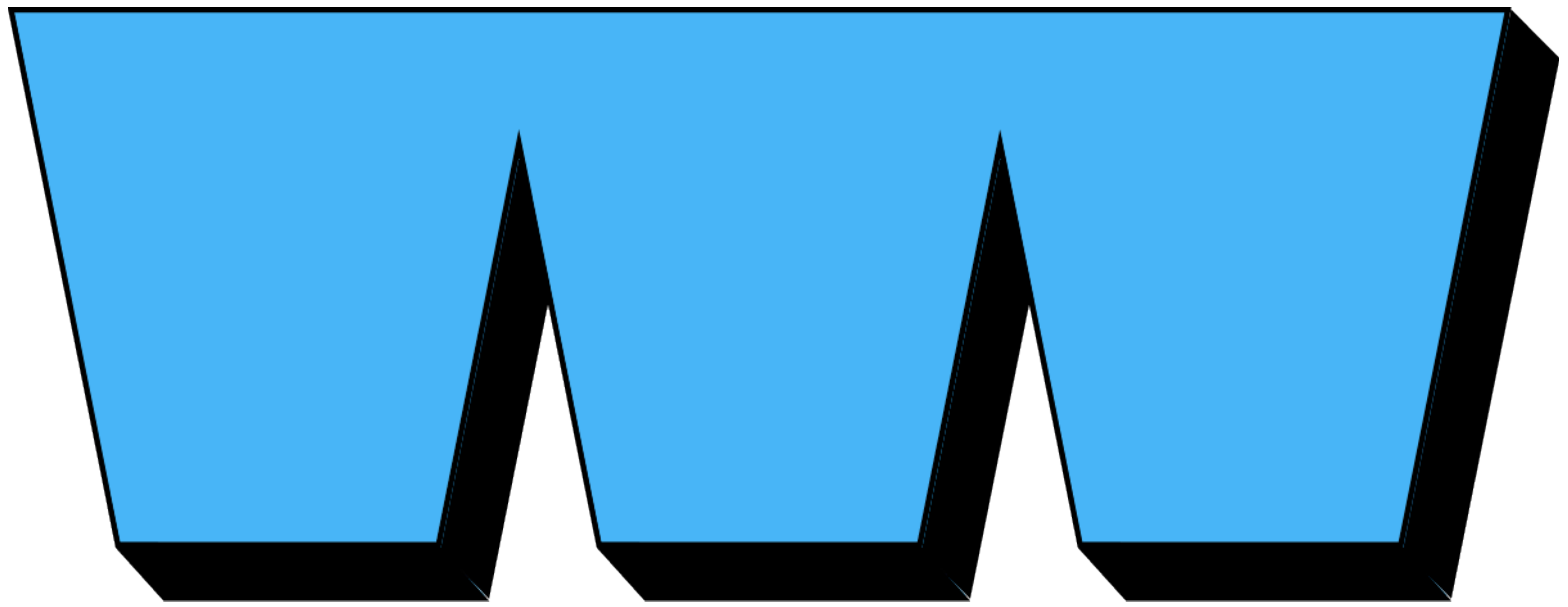
To copy an array, you have to copy all elements.

```
String[] names1 = {"Ib","Gry","Bo"};
String[] names2 = new String[names1.length];
for( int i=0; i<names1.length; i++ ){
 names2[i] = names1[i];
}

// Now we can update names2 without influencing names1
names2[1] = "Kaj";
```

For convenience, **import java.util.Arrays to use copyOf**

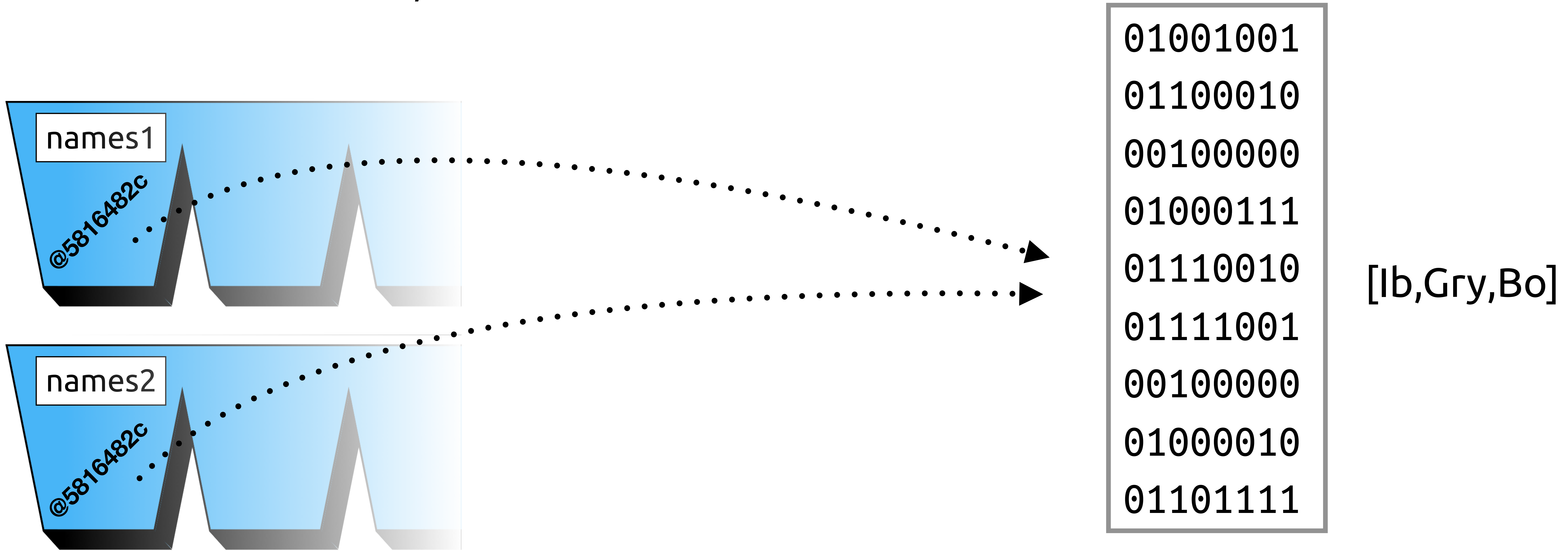
```
String[] names1 = {"Ib","Gry","Bo"};
String[] names2 = Arrays.copyOf( names1, names1.length);



// Now we can update names2 without influencing names1
names2[1] = "Kaj";
```

Does the variable-as-bin metaphor "hold water"?



In reality, The "bin" always holds a reference to data. Value type and reference type decides what is copied.

```
String[] names1 = {"Ib","Gry","Bo"};
String[] names2 = names1;
```


names1
@5816482c
names2
@5816482c
01001001
01100010
00100000
01000111
01110010
01111001
00100000
01000010
01101111
[Ib,Gry,Bo]