COMP 1010- Summer 2015 (A01)

Jim (James) Young

young@cs.umanitoba.ca

jimyoung.ca

Array Initialization with Literals

arrays and literals..

reminder: a literal is a value typed directly into a program, and is not calculated:

```
        int i = 1; // 1 is a literal
        float f = 3.14; // 3.14 is a literal
        final String QUESTION = “gimme a number”;
                // “…” is a literal
        double d = i*f; // no literal here
```

int[] intArray = **<a literal>;**

We often like to pre load an array with values

We make the array. We store some data into it.

E.g.,

Make an array with the days of the week stored.

Remember our calendar example?

```
final int CAL_TOP = 100;
final int CAL_LEFT = 100;
final int CAL_DAYS = 31;
final int CAL_SPACE = 30;
int selected = 0;
void setup()
{
  size(500, 500);
}

void draw()
{
  background(0);
  fill(255);
  // draw title bar
  int bottom = CAL_TOP+CAL_SPACE;
  int left = CAL_LEFT;
  text("S", left, bottom);
  left += CAL_SPACE;
  text("M", left, bottom);
  left += CAL_SPACE;
  text("T", left, bottom);
  left += CAL_SPACE;
  text("W", left, bottom);
  left += CAL_SPACE;
  text("R", left, bottom);
  left += CAL_SPACE;
  text("F", left, bottom);
  left += CAL_SPACE;
  text("S", left, bottom);
  left += CAL_SPACE;

  boolean hit = false;
  for (int day = 1; day<= CAL_DAYS; day++)
  {
    int col = day%7;
    int row = day/7+1;

    left = CAL_LEFT+CAL_SPACE*col;
    int right = left+CAL_SPACE;
    int top = CAL_TOP+CAL_SPACE*row;
    bottom = top+CAL_SPACE;

    if (mousePressed &&
      mouseX >= left && mouseX < right &&
      mouseY >= top && mouseY < bottom)
    {
      selected = day;
      hit = true;
    }

    if (selected == day)
    {
      rect(left, top, CAL_SPACE, CAL_SPACE);
      fill(0);
    } else
    {
      fill(255);
    }
    text(str(day), left, bottom);
  }
  if (mousePressed && !hit)
  {
    selected = 0;
  }
}
```

Remember our calendar example?

Ugly code – update by using an array to store the days of the week

Much better! But still that ugly array initialization looks clunky

literal array initialization

```
type[] variableName = { element, element, …,
                                    element};
```

NOTICE the curly brackets!

e.g.,

```
int[] evenNumbers = { 2, 4, 6, 8, 10, 12, 14};
boolean[] isPrime = { false, true, true, true, false, true};
String[] names = { “John”, “Jack”, “Joe”, “Jim” };
```

literals and memory (**new**)

note: literals do not require the “new” keyword to create them – it is automatically created with your values:

```
double [] measurements = {4.22, 11.1, 123.4};
```

is equivalent to:

```
double [] measurements = new double[3];
measurements[0] = 4.22;
measurements[1] = 11.1;
measurements[2] = 123.4;
```

Update calendar example

Arrays and memory

reminder

note: an array is a list of data of a given type

in processing: you must

1) **declare** an array variable
2) **instantiate** a new array (create it!)

type [] variableName; // declare

variableName = new **type**[size]; // instantiate

```
int [] intvariable = new int[500];
```

why two stages??

note: IMPORTANT: the array variable does not store the array.

The array can have all kinds of sizes, so the computer stores it in it’s general memory, not your variables.

When you make an array (new), the computer “allocates” some memory for you.

But you need to keep track of where that is!

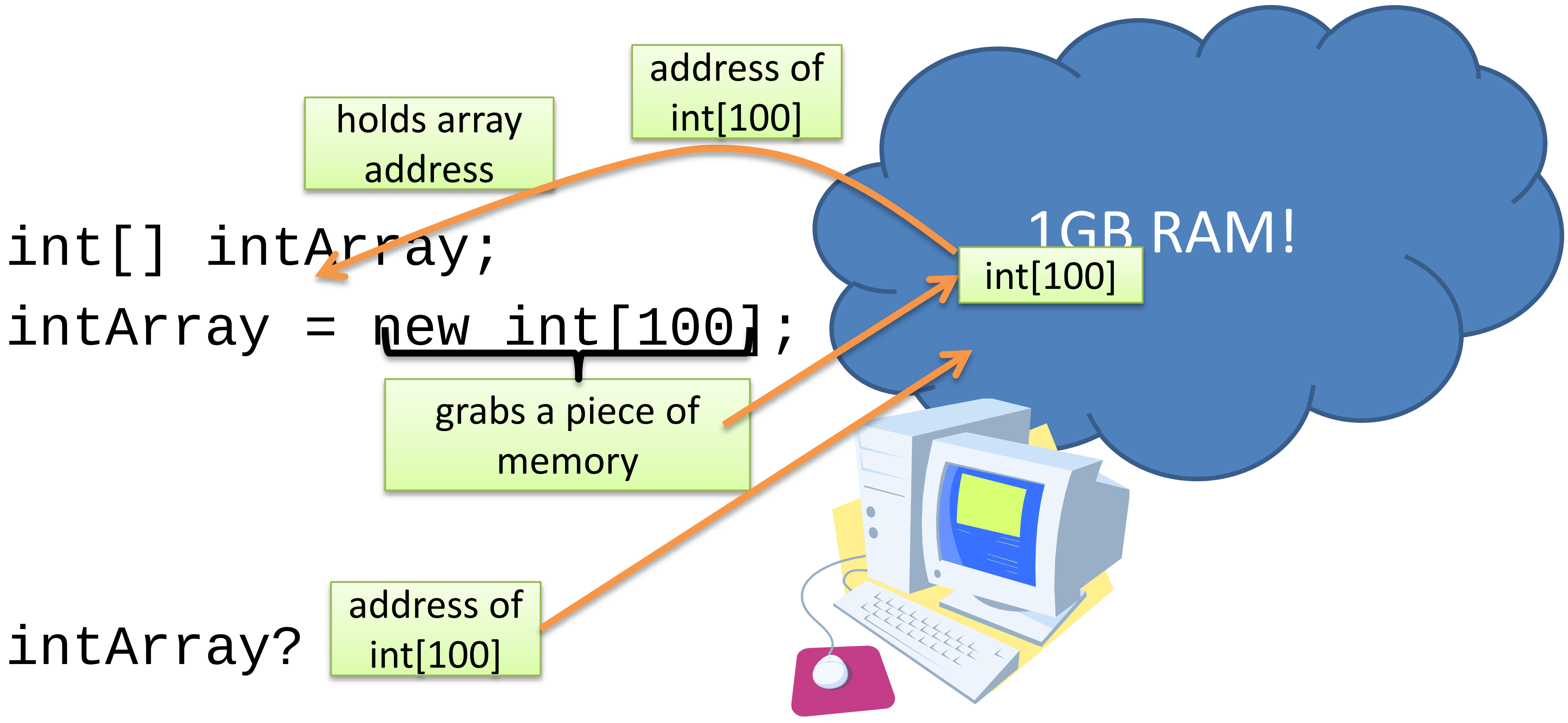
The variable stores an address of where the array is located in computer memory!!

why two stages??

The variable stores an address of where the array is located in computer memory!!

The variable does NOT store the array!!!

Again:



```
int[] intArray;
intArray = new int[100];

intArray?
```

(modern looking computer)

We can print the actual memory address using a silly trick:

println(“”+intArray); // don’t memorize

This becomes very important when using arrays with functions, copying them, or for comparing. For now, just remember the variable stores the address, the array is in memory somewhere

What happens?

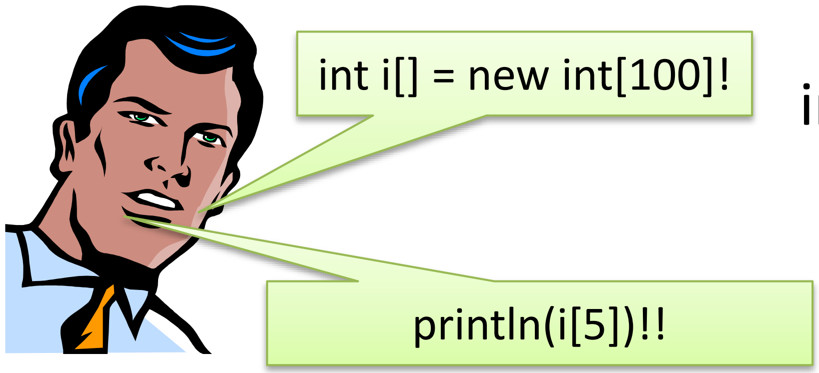
```
int variable;
println(variable);
```

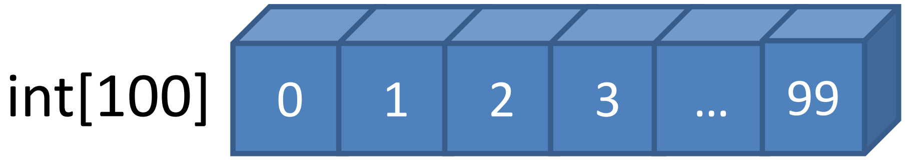
Processing works hard to check if a variable has been initialized. Not trivial.

Gets harder with arrays!

Processing avoids the issue – fills arrays with default values

default value of array entries??





if you create an array, but do not set the bins to any value…
what's stored in the bins by defualt?

depends in the type!

- default for numerical:
  - 0, 0.0
- default for boolean?
  - false
- default for String?
  - **null**! no string!

null

null [nʌl]*adj*

1. without legal force; **invalid**; (esp in the phrase **null and void**)

2. without value or consequence; **useless**

3. lacking distinction; characterless *a null expression*

4. nonexistent; amounting to nothing

null!

null cannot be used for primitives!

```
int i = null; // error, requires an int!!
float f = null; // error, requires a double!
boolean b = null; // error, requires a  bool
char c = null; // error!! requires a character
```

reminder: a variable type starting with a Capital represents an object:

```
String
```

note: objects and **arrays** can be set to null

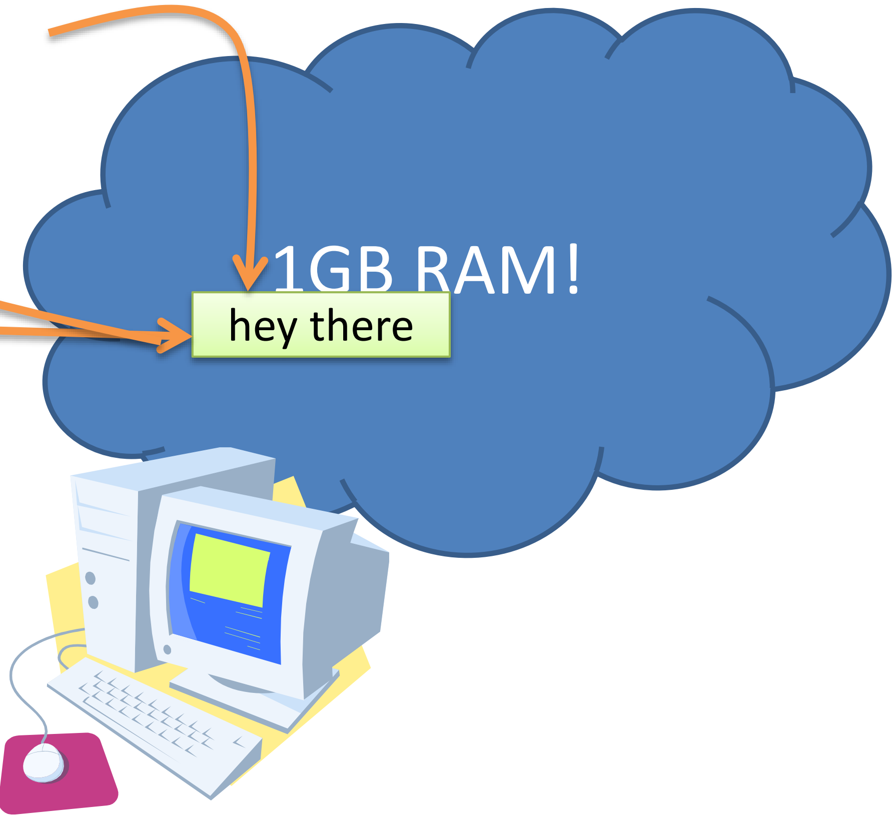
```
int[] someArray = null; // valid but not usable!
```

null and memory - Strings??

String s = “hey there”;

s? address of “hey there”

s = null; no address

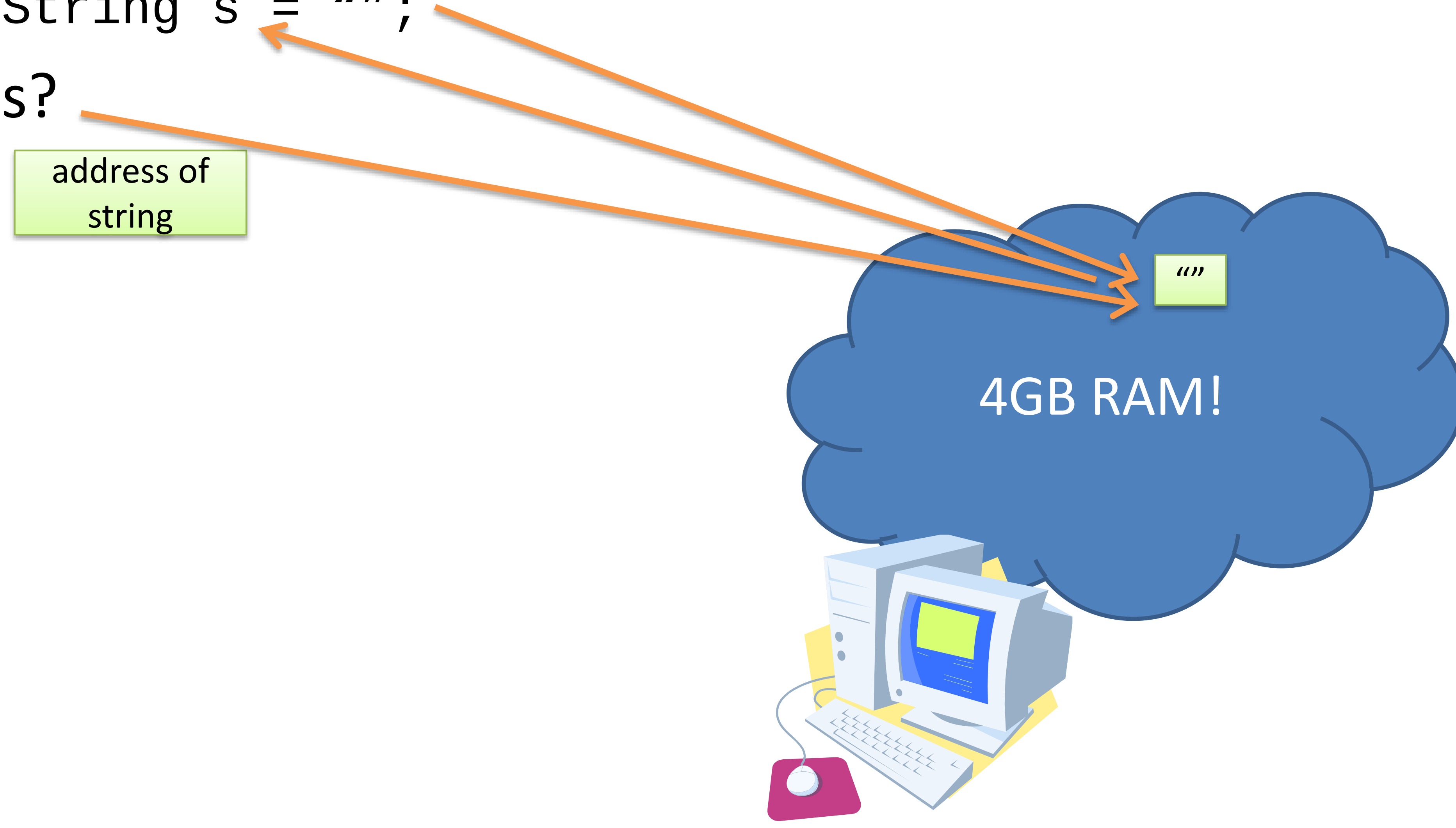


(modern looking computer)

String – empty string?

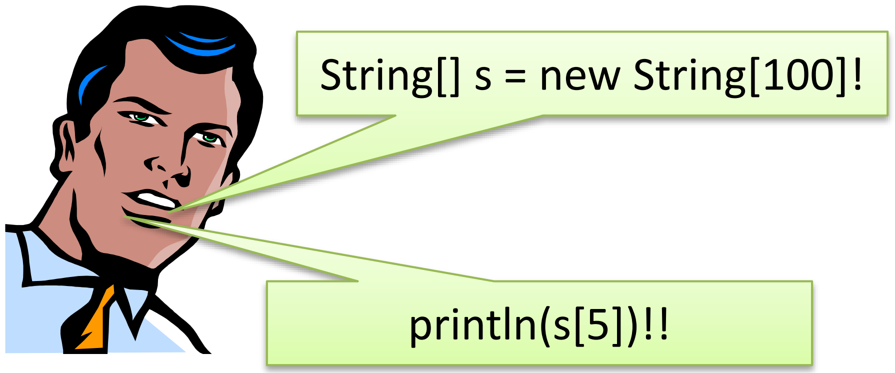
```
String s = "";
```

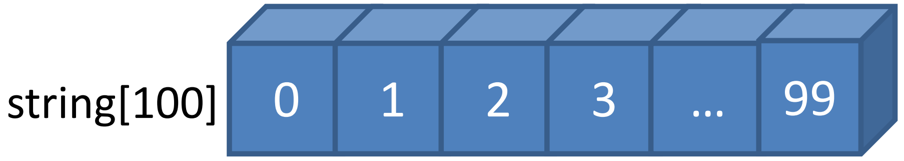
s?



(modern looking computer)

default value of array entries!





default for numerical:

0, 0.0

default for boolean?

false

default for String?

null! no string!

Default values: careful!

This is a Processing and Java thing! Not all languages treat this the same.

copy an array

reminder: to copy an int or float?

```
int i = 1982;
int j = i; //← copy i into j
i = 1999;
println(i+ “ “+j); // what is the output?
```

the output is: “1999 1982”

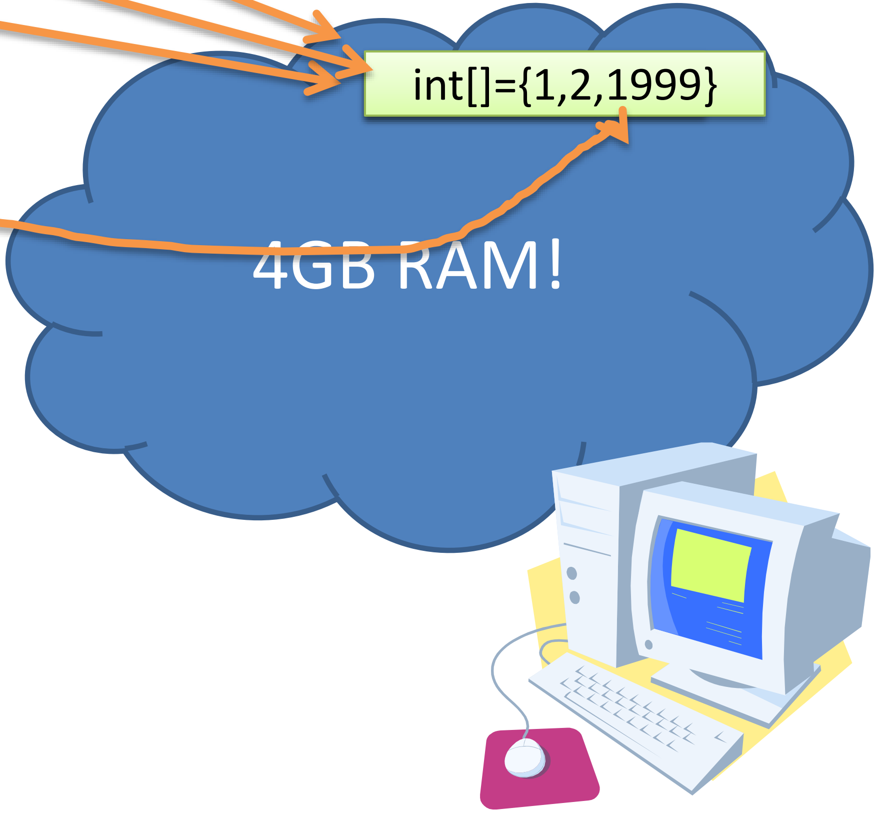
lets do something similar with an array

```
int[] i = {1, 2, 3};
int[] j = i; //← copy i into j..?
i[0] = 1999;
println(i);
println(j);
```

what happened?

```
int[] i = {1, 2, 3};
int[] j = i; //← copy i into j..?
i[2] = 1999;
```

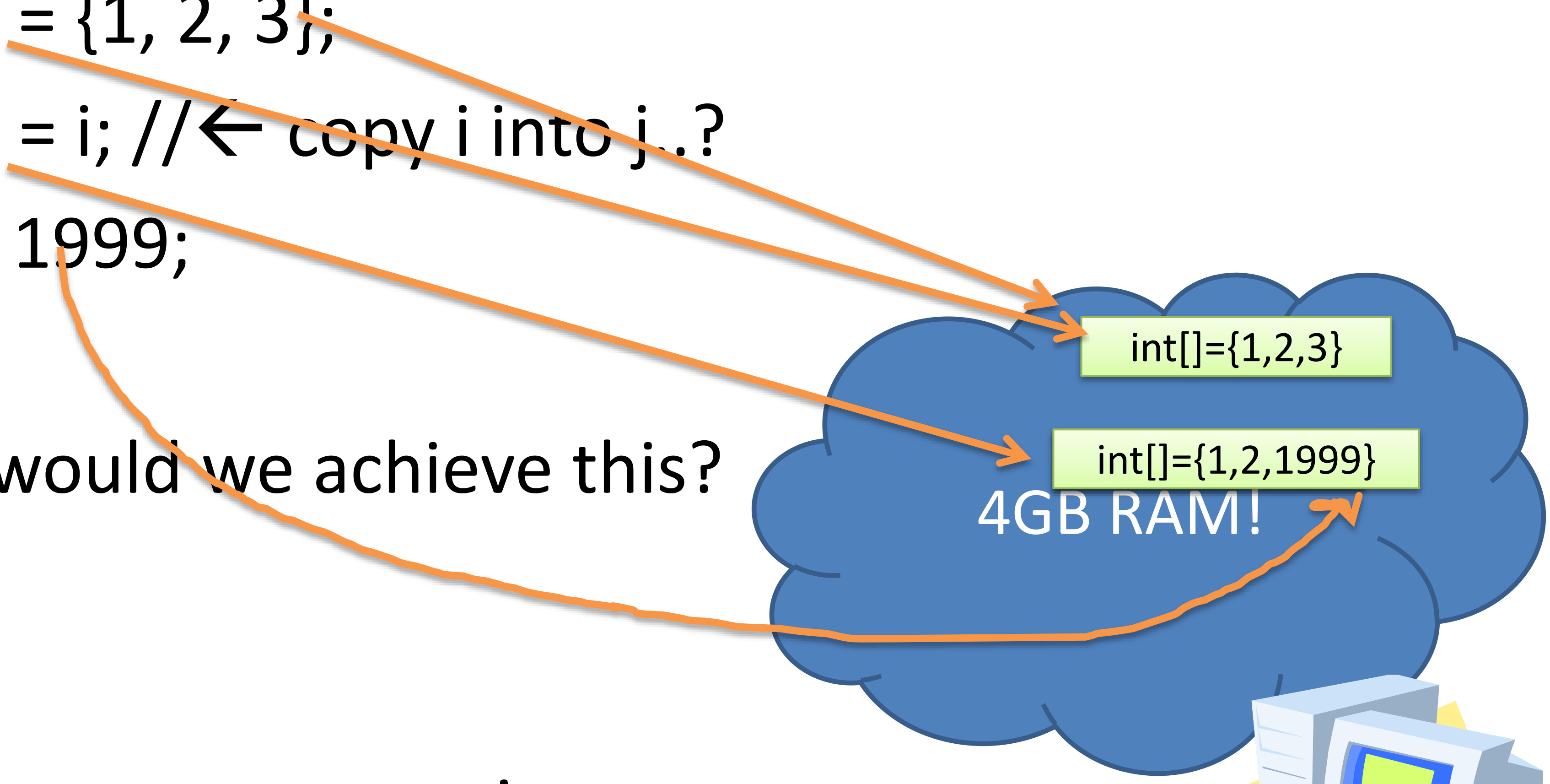
note: array variables only record the address, or the **reference to** the array off in computer memory. When you copy the variable, you **only copy the reference, not the array.**



(modern looking computer)

goal: what we wanted

```
int[] i = {1, 2, 3};
int[] j = i; //← copy i into j..?
j[2] = 1999;
```



how would we achieve this?

1. create a new array in memory
2. copy the contents over yourself

(modern looking computer)

1. Create – existing syntax
2. Copy – use a for loop