CS 11: Introduction to Computer Science

Introducing Pointers

Mark Sheldon
Tufts University
Email: msheldon@cs.tufts.edu
Web: http://www.cs.tufts.edu/~msheldon

Noah Mendelsohn
Tufts University
Email: noah@cs.tufts.edu
Web: http://www.cs.tufts.edu/~noah

Goals for this session

- **Explain why pointers are useful**
- **Learn to declare and use pointer variables in C++**
- **Begin learning to use pointers in our programs**

The Spy Agency Library

The Spy Agency Library

The spy agency library…lots of papers and periodicals

* Uh, actually this is the RISD library, photo sourced from https://www.flickr.com/photos/93006384@N05/8468897073
License: https://creativecommons.org/licenses/by/2.0/ but it looks good!

The spy agency library…lots of papers and periodicals

The old system:

- Spies would fill out call slips identifying the periodical to be retrieved
- Librarians bring original documents to the spies…who annotate the documents with interesting notes and cross references
- Over time…the documents become more valuable with the *shared updates*.

* Uh, actually this is the RISD library, photo sourced from https://www.flickr.com/photos/93006384@N05/8468897073
License: https://creativecommons.org/licenses/by/2.0/ but it looks good!

The spy agency library…lots of papers and periodicals

The new system:

- Librarians deliver digitally printed *copies* of the documents
- Any notes are made on the copies…
- …no more sharing ☹

* Uh, actually this is the RISD library, photo sourced from https://www.flickr.com/photos/93006384@N05/8468897073
License: https://creativecommons.org/licenses/by/2.0/ but it looks good!

And the point is…the old system was more useful…

Jane's Defense Weekly
18 March 2013

Call slip #1

Jane's Defense Weekly
18 March 2013

Call slip #2

To share the same data, you all need to *point to the same copy*.

And your point is…

There are *multiple* call slips…

(in our computer programs, there may be two or more *pointer variables*)

Jane's Defense Weekly
18 March 2013

Call slip #1

Jane's Defense Weekly
18 March 2013

Call slip #2

To share the same data, you all need to *point to the same copy*.

And your point is…

But each object is typically known by the same *pointer value…*

…which we call the address of the or the location of the variable we're pointing to

Jane's Defense Weekly
18 March 2013

Call slip #1

Jane's Defense Weekly
18 March 2013

Call slip #2

To share the same data, you all need to *point to the same copy*.

In the computing word there will be three variables

Variable 2:

Contents are the address of variable 1

s Defense Weekly
18 March 2013

Variable 3:

Contents are the address of variable 1

Jane's Defense Weekly
18 March 2013

Call slip #2

Variable 1:

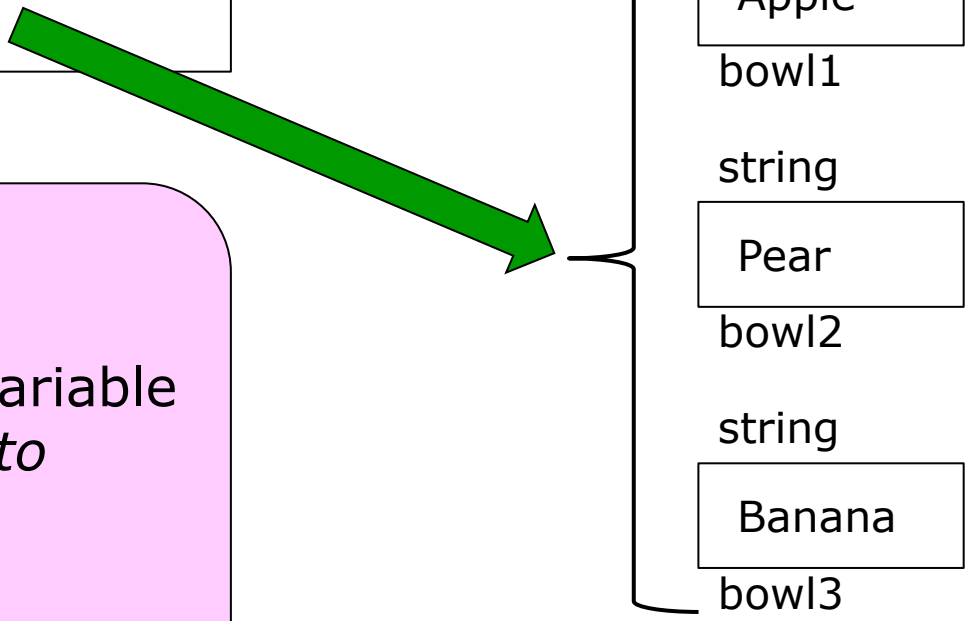
The shared object

C++ lets us start with a pointer variable like variable 2, *and use it to access or update the pointed-to-object (variable 1).*

A First Look at C++ Pointers

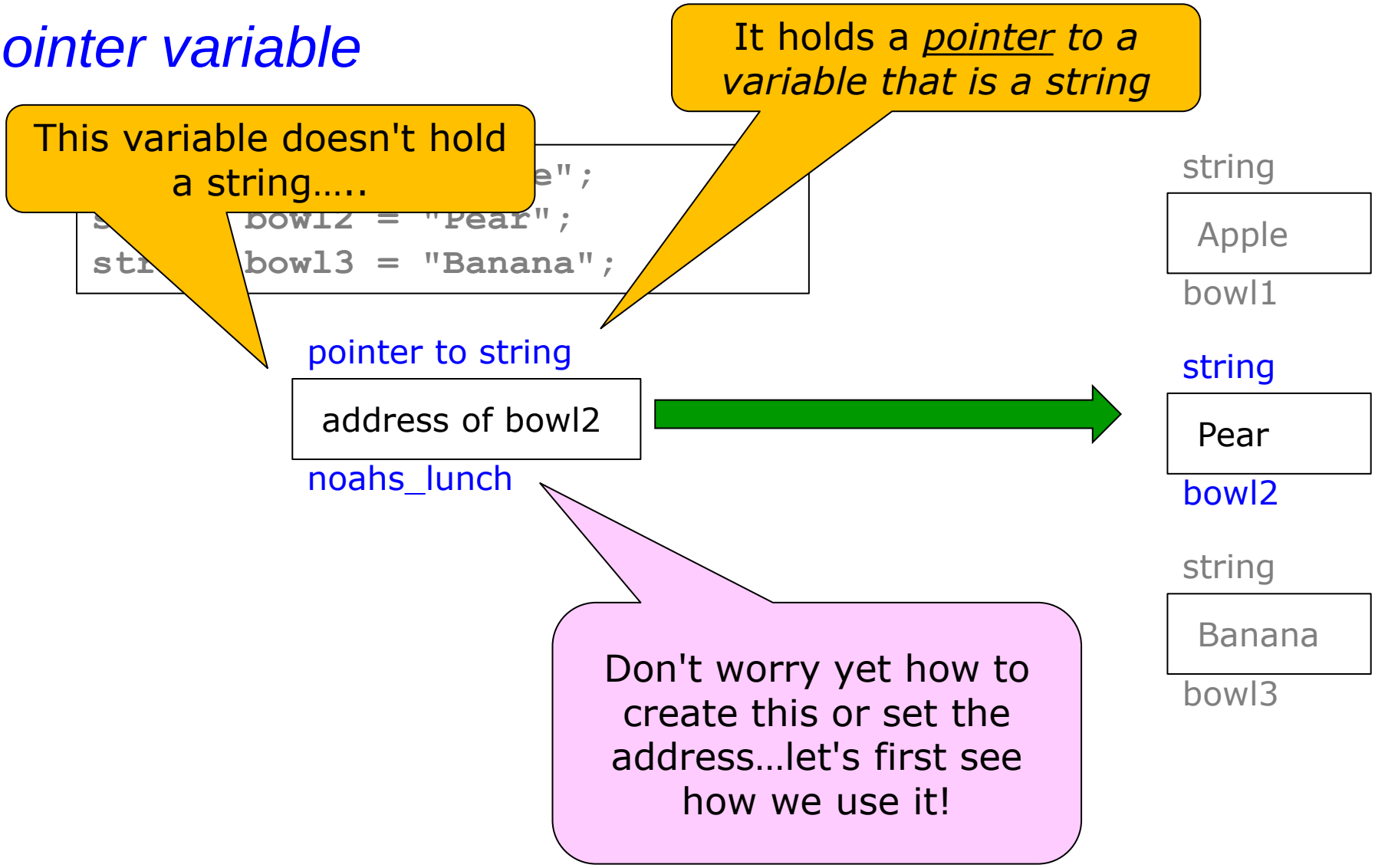
Some ordinary string variables

```
string bowl1 = "Apple";
string bowl2 = "Pear";
string bowl3 = "Banana";
```

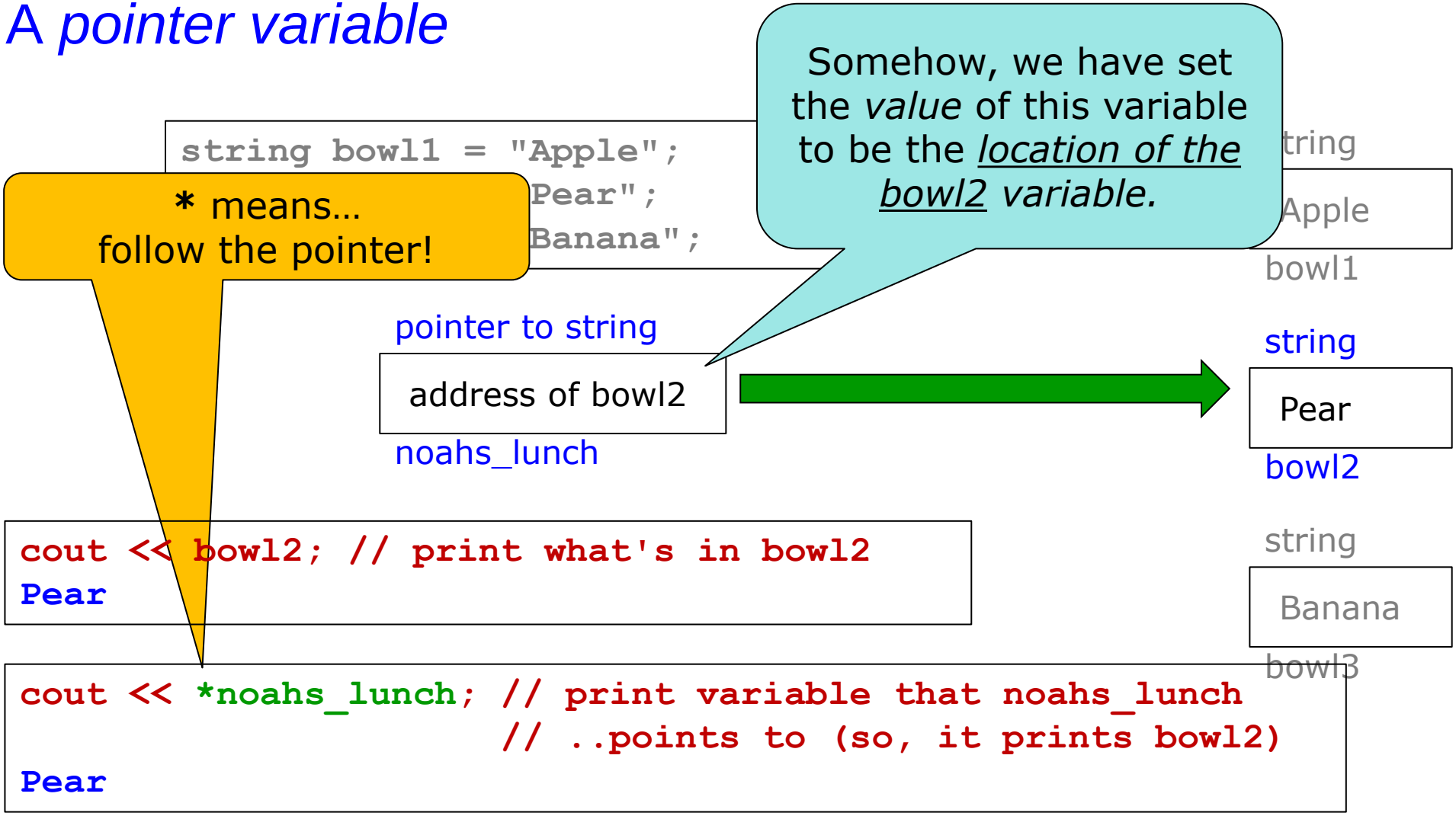


What if we could have a variable *with a value that pointed to another variable?*

A *pointer variable*



A *pointer variable*

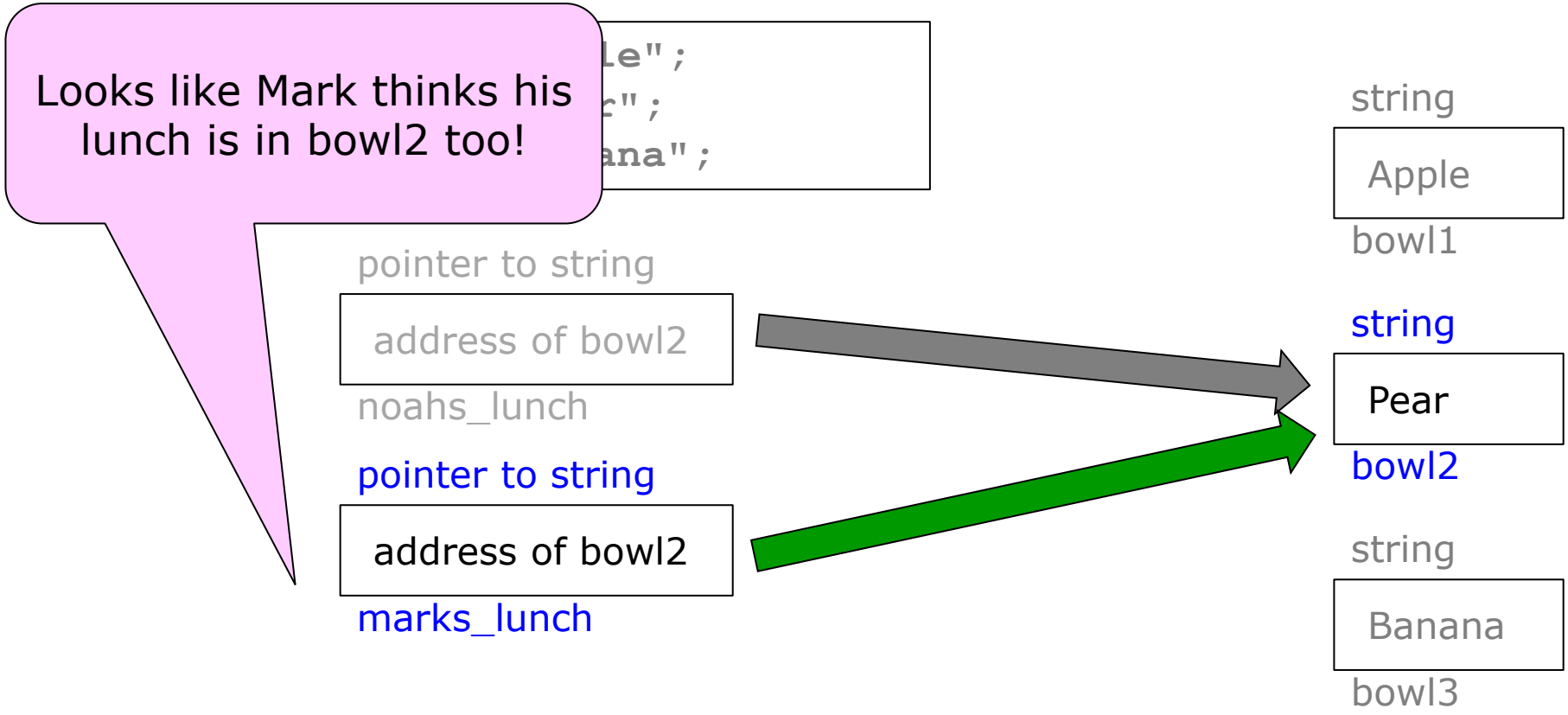


STOP!!

If you understand what you just saw, you have the key idea of pointers...the rest is details!

More Than One Pointer Variable Can Point to the Same Thing

Two pointers to the same variable



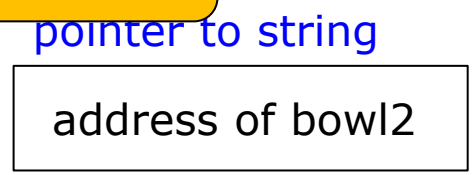

```
cout << *marks_lunch; // print variable that marks_lunch
                      // ..points to
Pear
```

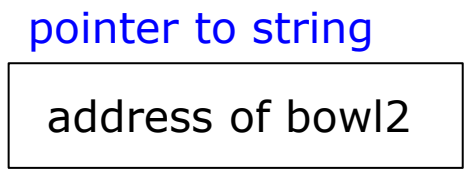
Here comes the good part…☺

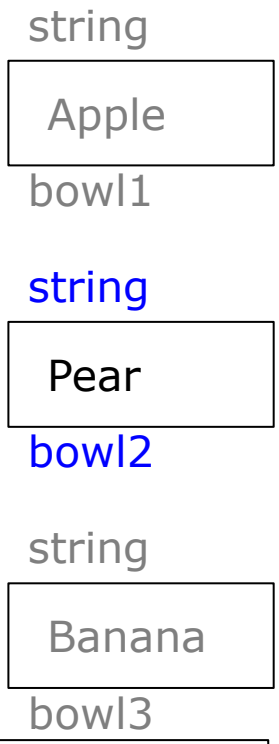
Updating the variable *we are pointing to!*


```
string bowl1 = "Apple";
string bowl2 = "Pear";
string bowl3 = "Banana";
```

Remember: * means… follow the pointer!



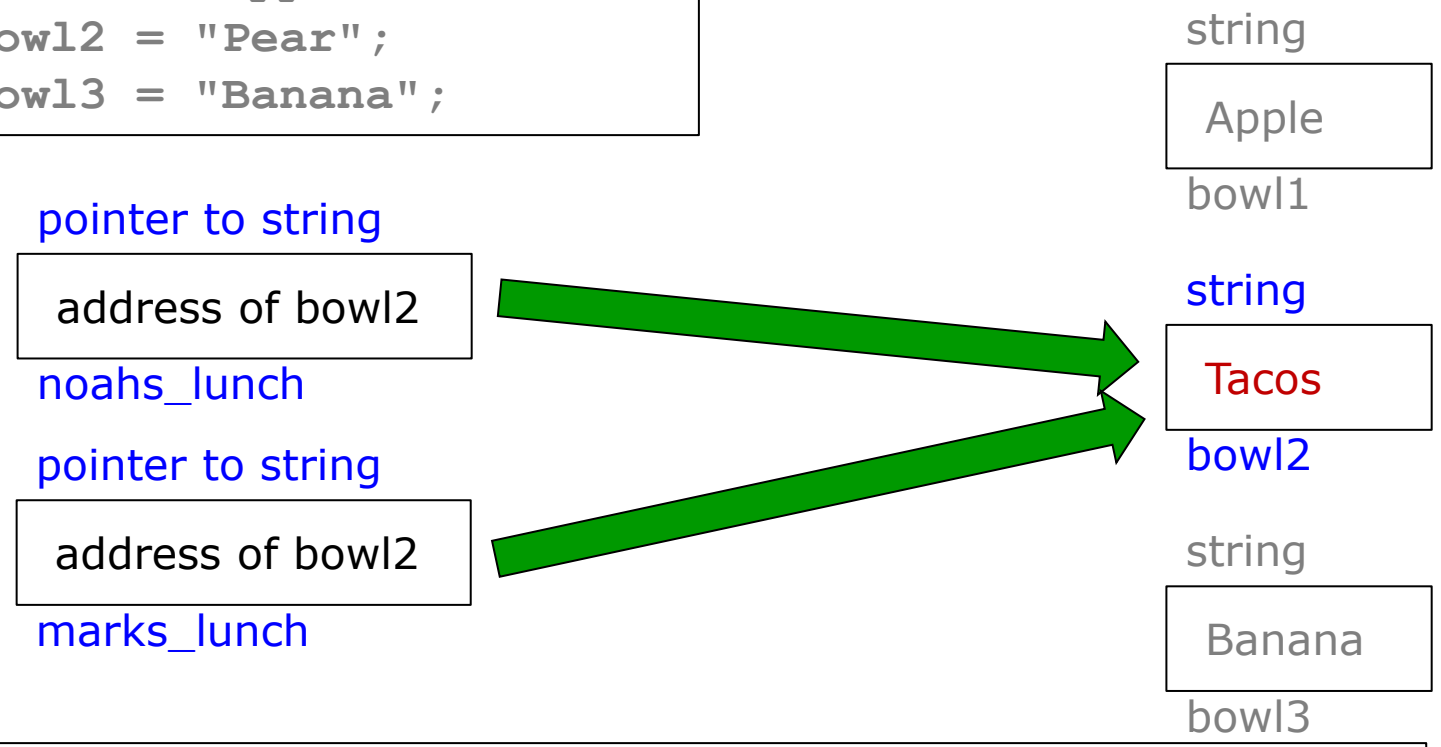





```
*noahs_lunch = "Tacos";   // Noah puts "tacos" in *noahs_lunch
```

Two pointers to the same variable

```
string bowl1 = "Apple";
string bowl2 = "Pear";
string bowl3 = "Banana";
```

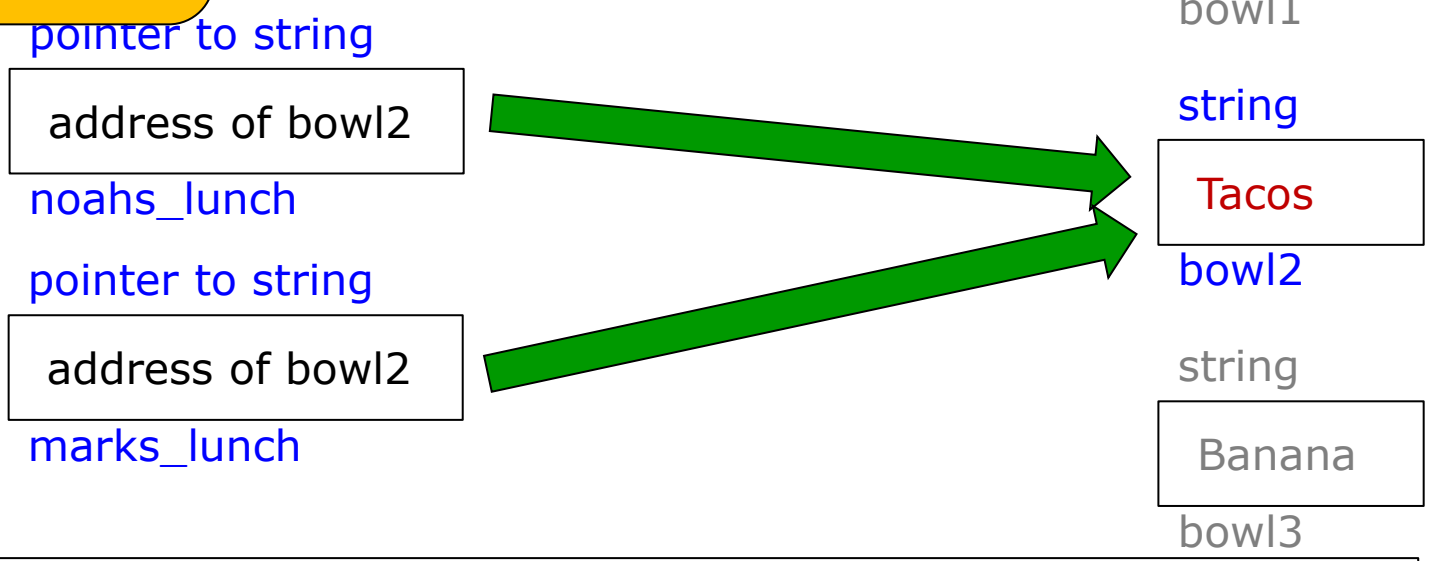



```
*noahs_lunch = "Tacos";   // Noah puts "tacos" in *noahs_lunch
cout << *noahs_lunch;
Tacos
```

Two pointers to the same variable

But Mark is pointing to the same bowl, so his lunch changed too!

```
Apple";
Pear";
Banana";
```



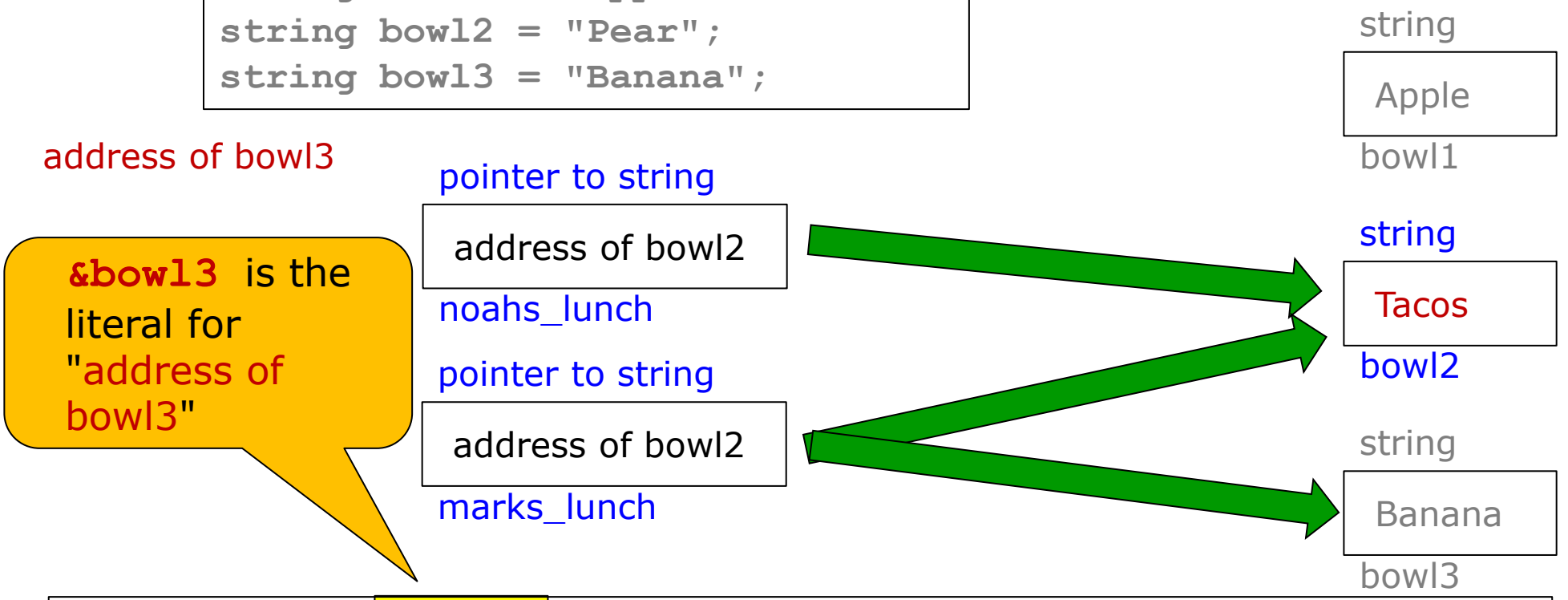

```
*noahs_lunch = "Tacos";    // Noah puts "tacos" in *noahs_lunch
cout << *noahs_lunch;
Tacos
cout << *marks_lunch;      // ... Hey, but they looked so good!
Tacos
```

Like Any Other *Variable* We Can Change What's in a Pointer Variable

…which changes what it points to!

But Mark prefers to have his own bowl!

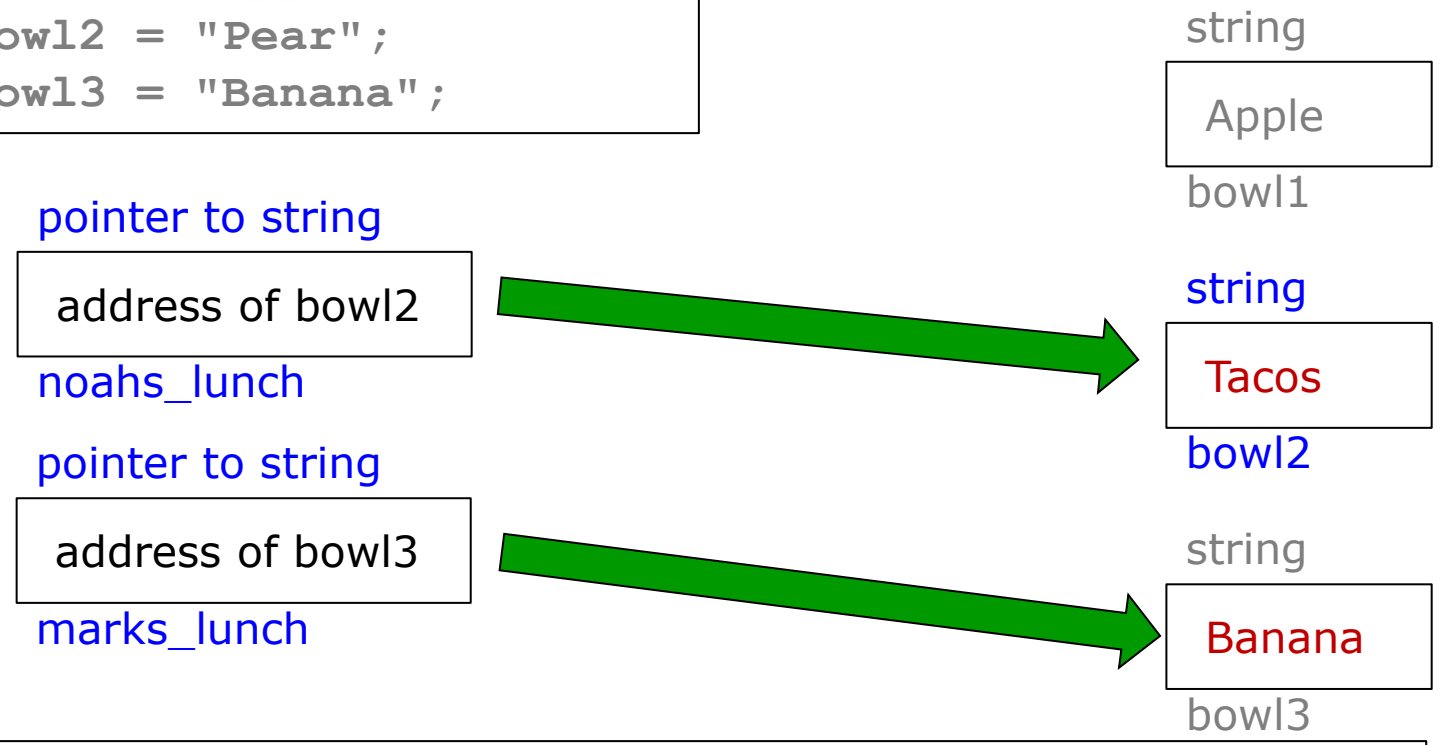
```
string bowl1 = "Apple";
string bowl2 = "Pear";
string bowl3 = "Banana";
```




```
marks_lunch = &bowl3;      // marks_lunch now points to bowl3
```

But Mark prefers to have his own bowl!

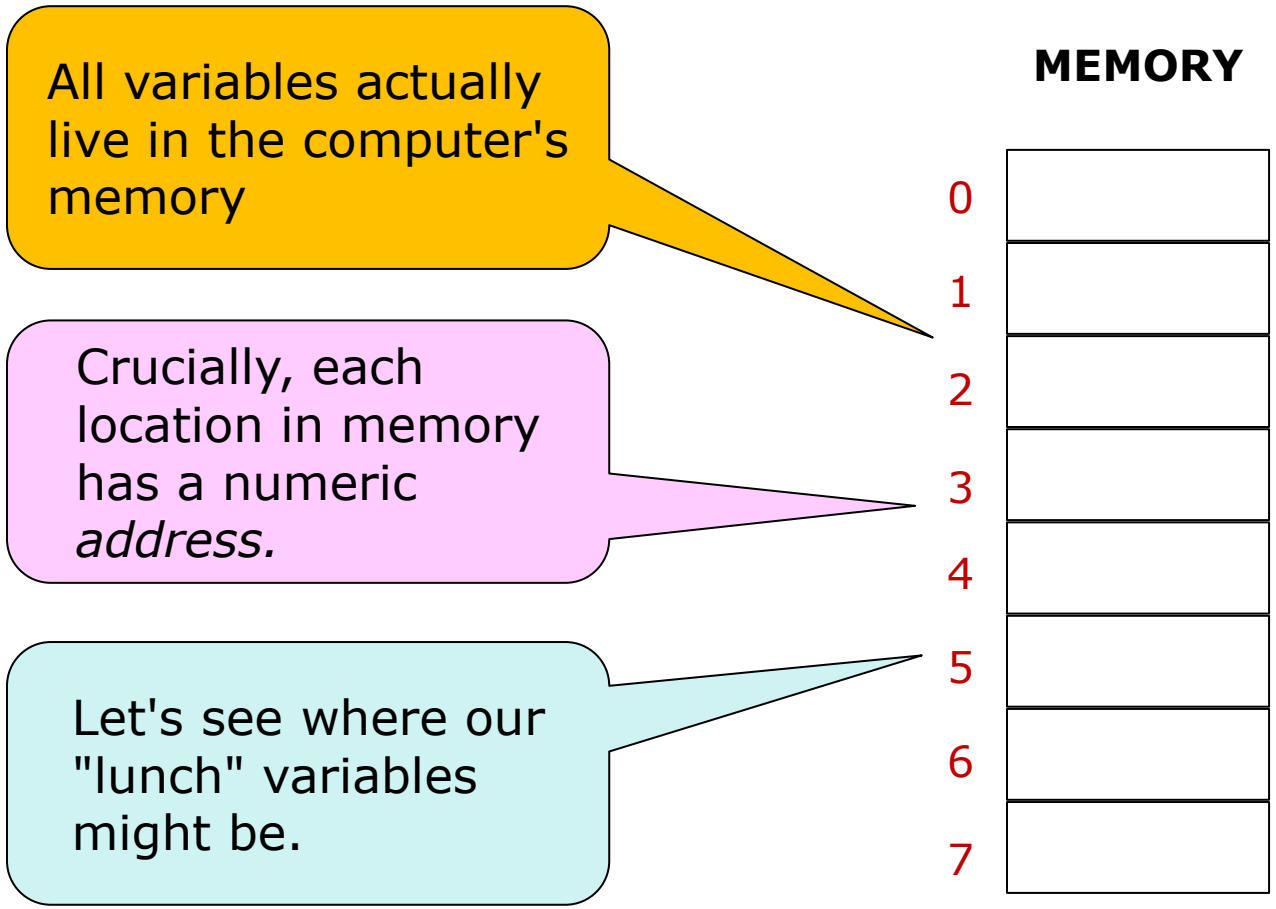
```
string bowl1 = "Apple";
string bowl2 = "Pear";
string bowl3 = "Banana";
```




```
marks_lunch = &bowl3;      // marks_lunch now points to bowl3
cout << *noahs_lunch;      // The * operator says: get me what
                           // ..it points to
Tacos
cout << *marks_lunch;      // ... ummm!
Bananas
```

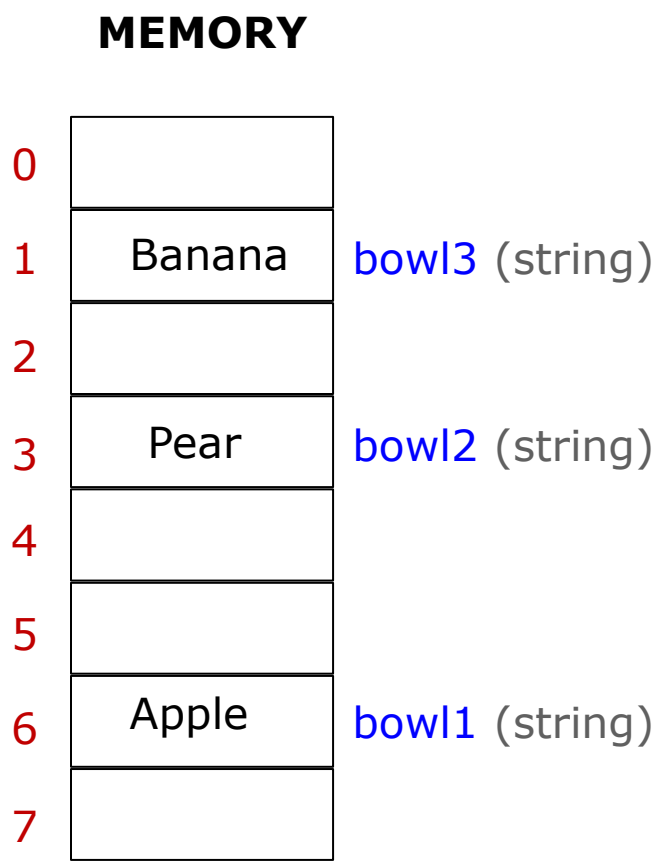
What's going on inside the computer

We showed this picture during our first class…



Let's repeat the same example, showing code to set ptrs…

```
string bowl1 = "Apple";
string bowl2 = "Pear";
string bowl3 = "Banana";
```



Let's repeat the same example, showing code to set ptrs…

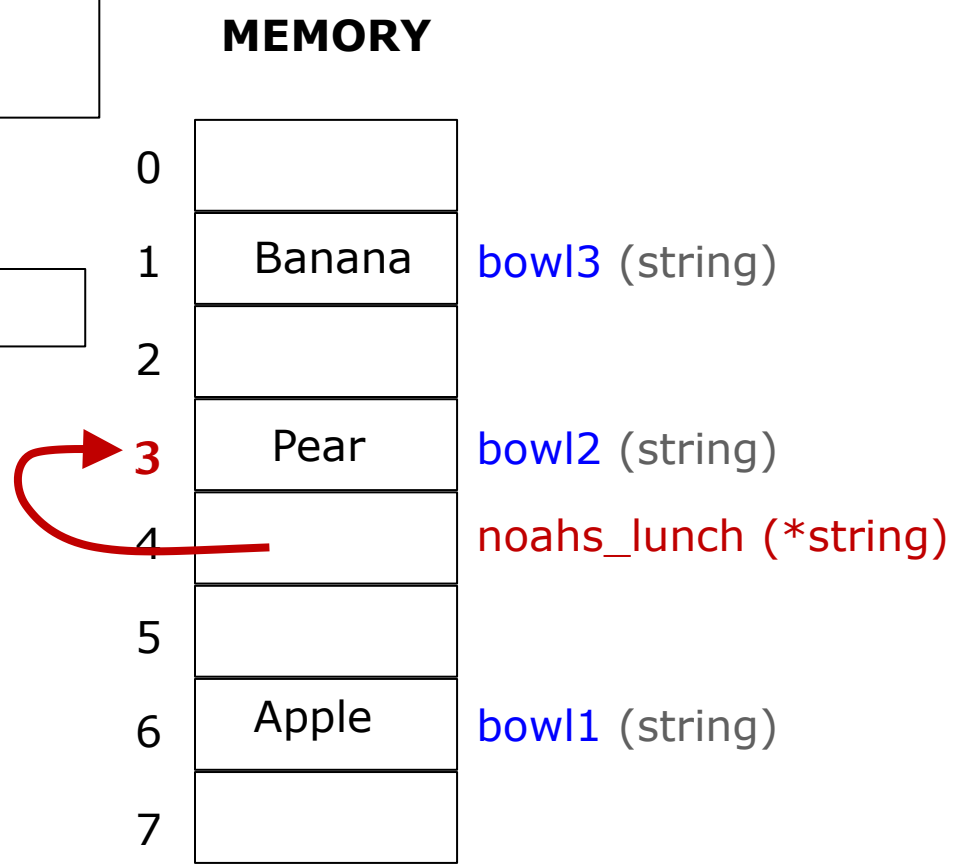
```
string bowl1 = "Apple";
string bowl2 = "Pear";
string bowl3 = "Banana";
```

```
string* noahs_lunch = &bowl2;
```

&bowl2

Is the *address* of bowl2…

…which is 3



Let's repeat the same example, showing code to set ptrs…

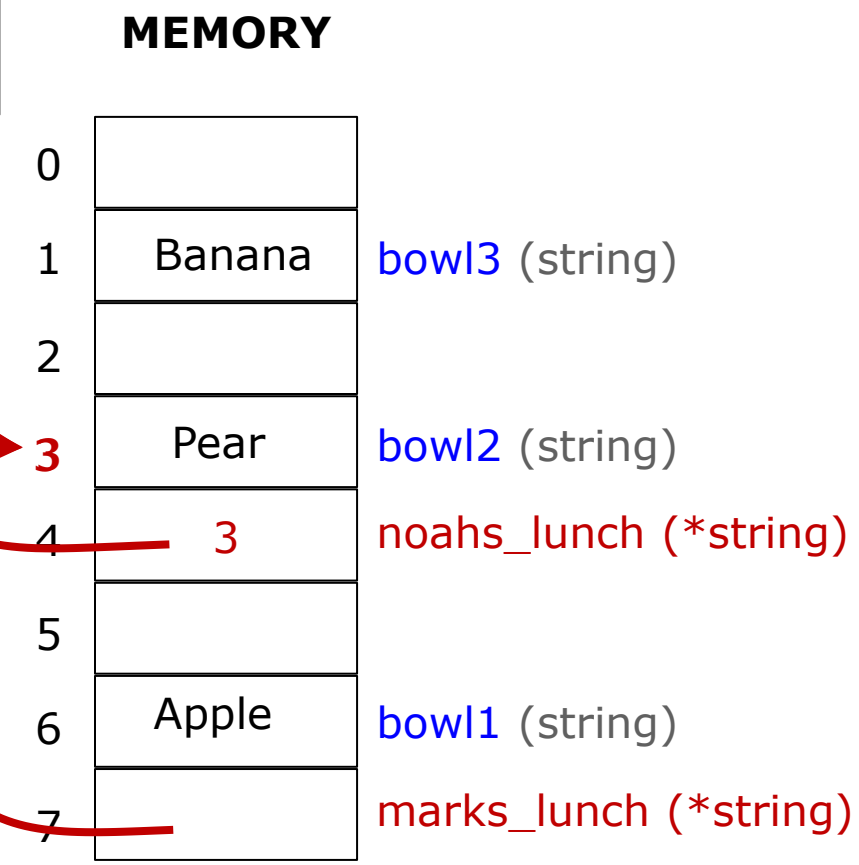
```
string bowl1 = "Apple";
string bowl2 = "Pear";
string bowl3 = "Banana";
```

```
string* noahs_lunch = &bowl2;
string* marks_lunch = &bowl2;
```

&bowl2

Is the *address* of bowl2…

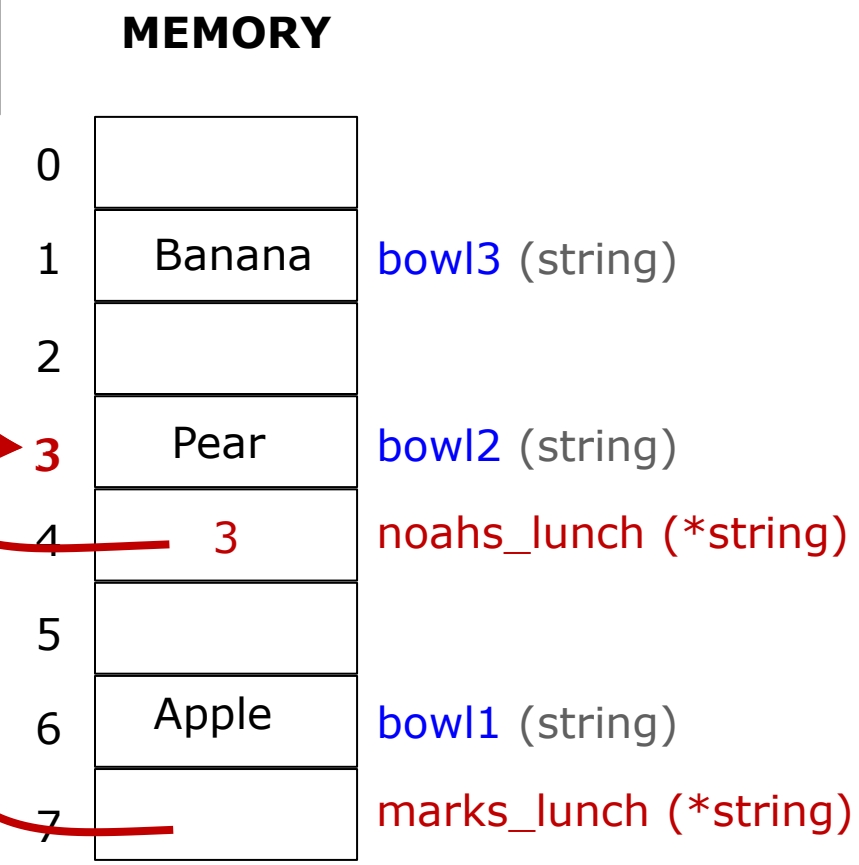
…which is of course still 3



Let's repeat the same example, showing code to set ptrs…

```
string bowl1 = "Apple";
string bowl2 = "Pear";
string bowl3 = "Banana";
```

```
string* noahs_lunch = &bowl2;
string* marks_lunch = &bowl2;
cout << *marks_lunch;
Pear
cout << *noahs_lunch;
Pear
```



Review of Pointer Basics

Highlights

- **All variables live in computer memory at some address**
- ***Pointer variables* use those addresses to *point to* other variables**
- **You can assign a pointer variable as you can any other:**

```
noahs_favorite_class = &comp11;   // & means "address of"
```

- **You can access or update the variable that a pointer points to:**

```
cout << *noahs_favorite_class;  // prints what's in comp11
```

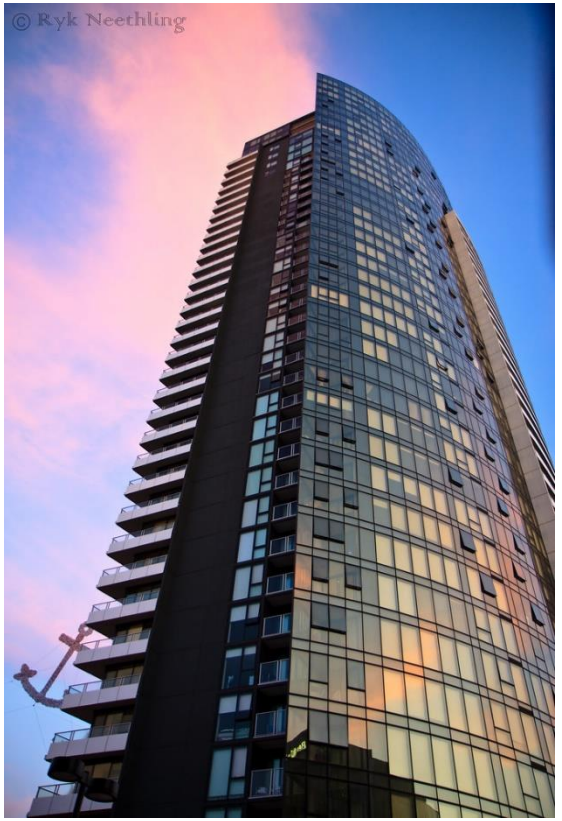
We call that "dereferencing the pointer"

Why Pointers Are So Important

The address of a skyscraper is no bigger than the address of a coffee shop

123 Main Street, Boston, MA

123 Main Street, Chicago, IL

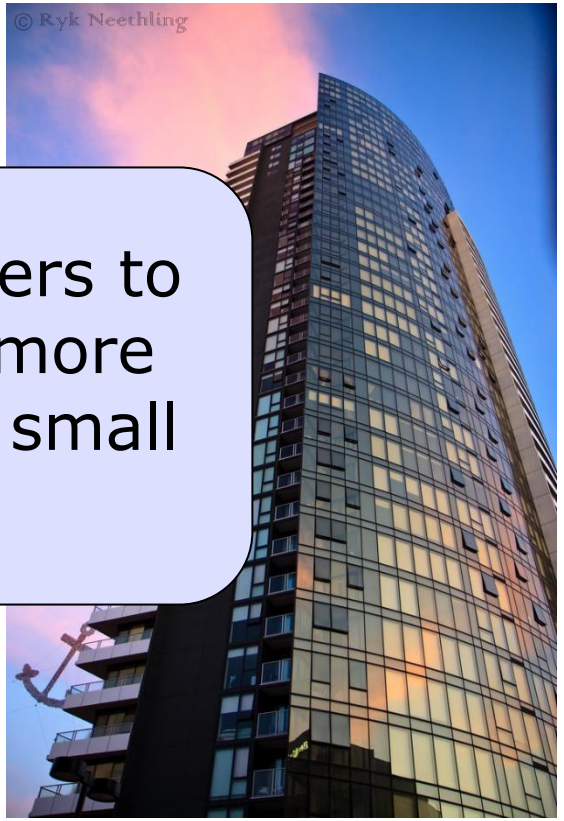


The address of a skyscraper is no bigger than the address of a coffee shop

123 Main Street, [illegible]

123 Main Street, Chicago, IL

In our programs, pointers to huge objects take no more space than pointers to small ones.



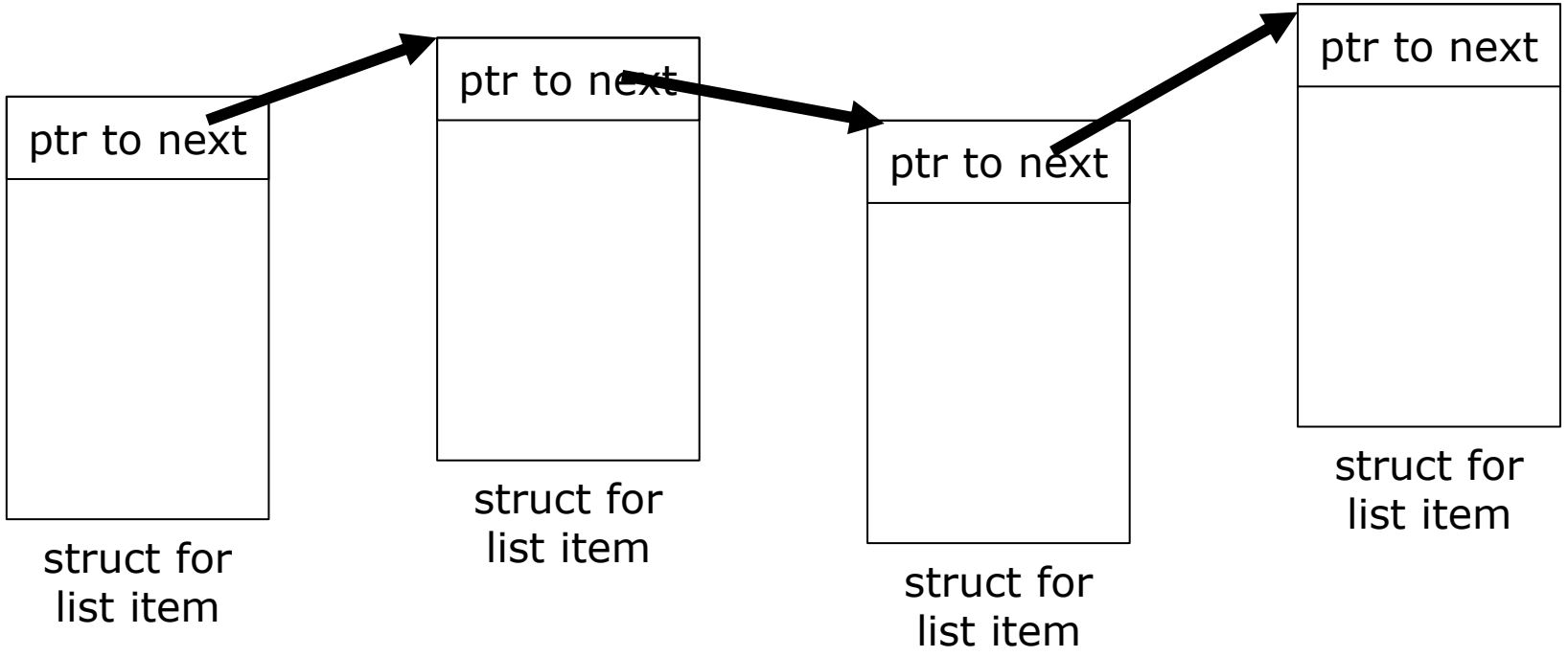
The address of a skyscraper is no bigger than the address of a coffee shop

123 Main Street,

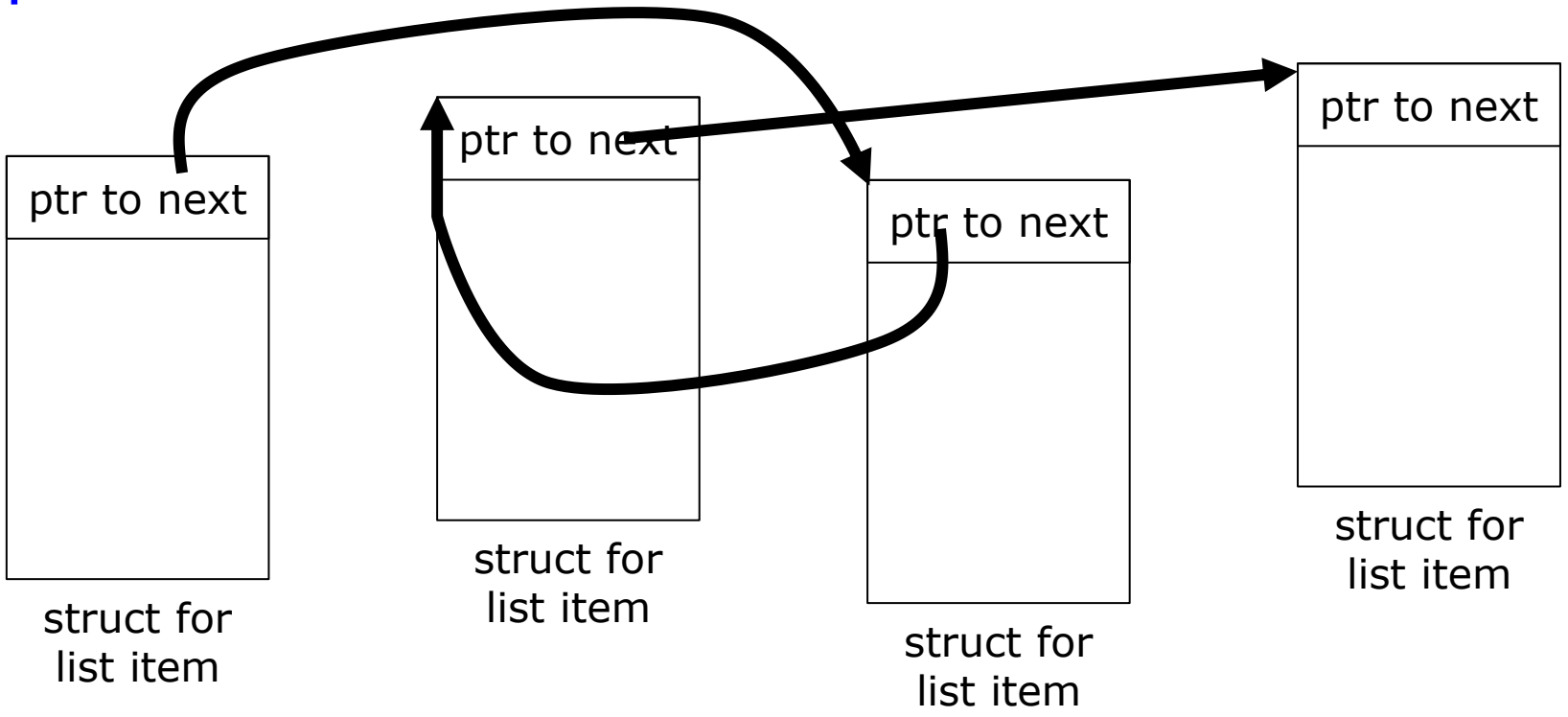
123 Main Street, Chicago, IL

This is a great way to efficiently give your functions access to data structures that are too large to copy…pass in (or return) a pointer!

Complex data structures - lists

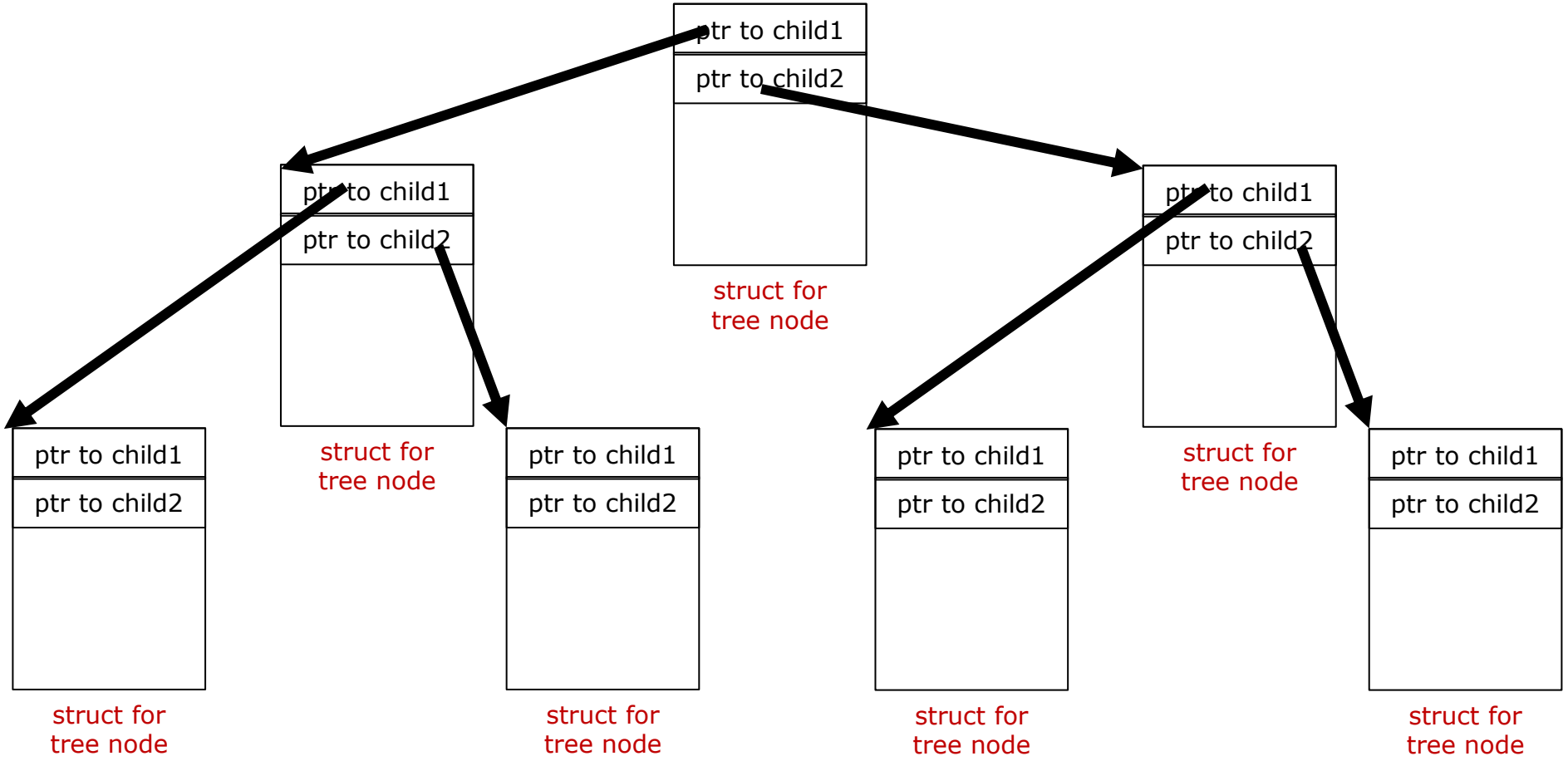


Complex data structures - lists



We an insert, delete, and rearrange items in the list by changing the pointers!

Complex data structures – trees (recursive data structure!)



Summary

Pointer review

- **Very simple idea: one variable contains the address of another**
- **C++ lets you declare variables with pointer types (e.g. *int is pointer to integer)**
- **Using C++ vars:**
  -
```
int *my_int_pointer;                // get a new pointer variable
my_int_pointer = &your_variable;    // & gets the address
cout *int_pointer;                  // I'm printing your variable
*int_pointer = 3;                   // I'm changing your variable
f(int_pointer);                     // passing pointer to function
```
- **Variables are useful for many many reasons:**
  - Named references
  - Sharing data (the lunch bowls)
  - Efficient sharing of large objects
  - Letting functions update caller's data (pass by reference)
  - Creating complex data structures (lists, trees)

We will explore all of this in future classes!