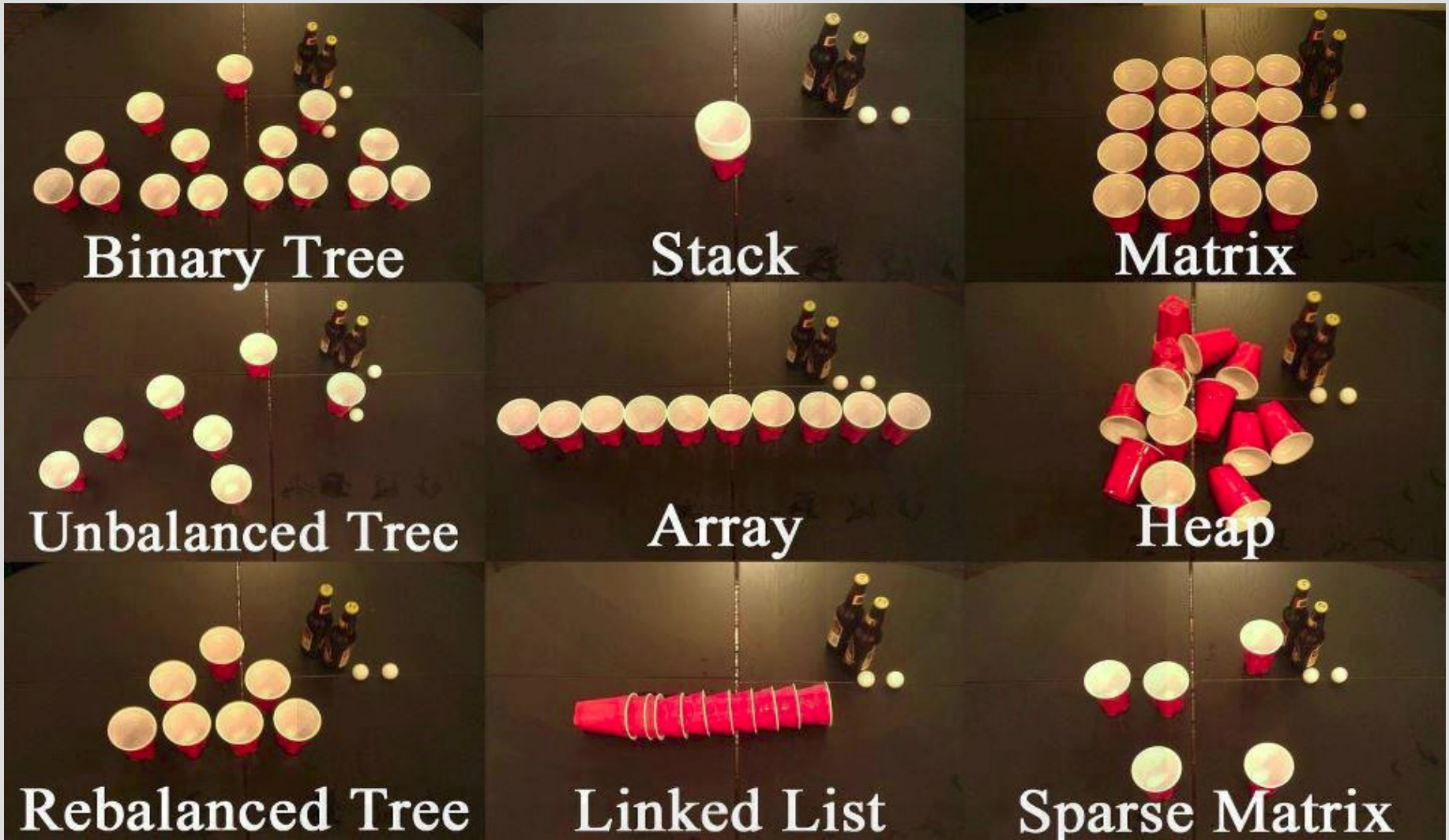
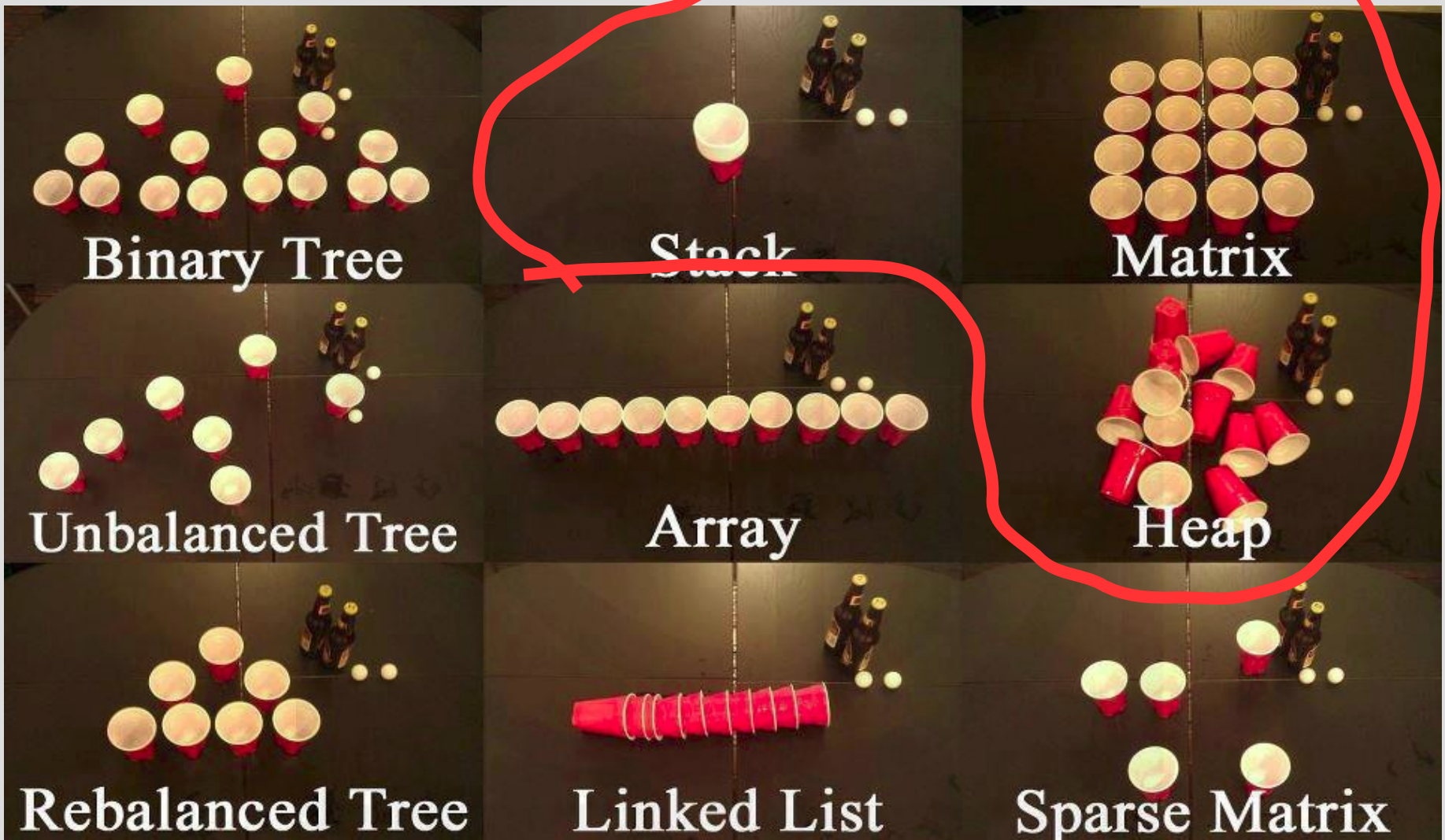
Data structures, part 2

Ch ???



Highlights


Binary Tree
Stack
Matrix
Unbalanced Tree
Array
Heap
Rebalanced Tree
Linked List
Sparse Matrix

Multi-dimension arrays

2D arrays have a row and column index, however this is a bit misleading

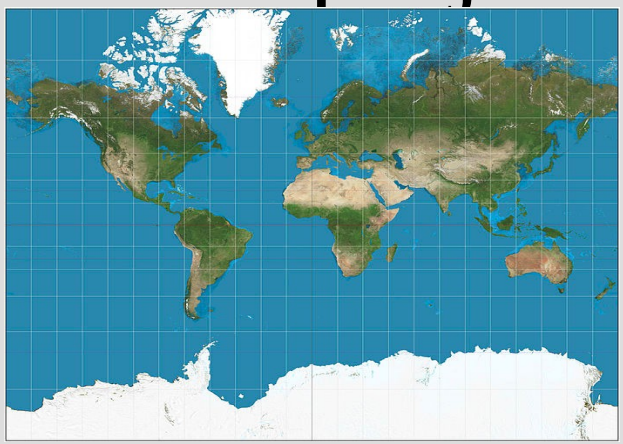
Computers actually only have a 1D memory...

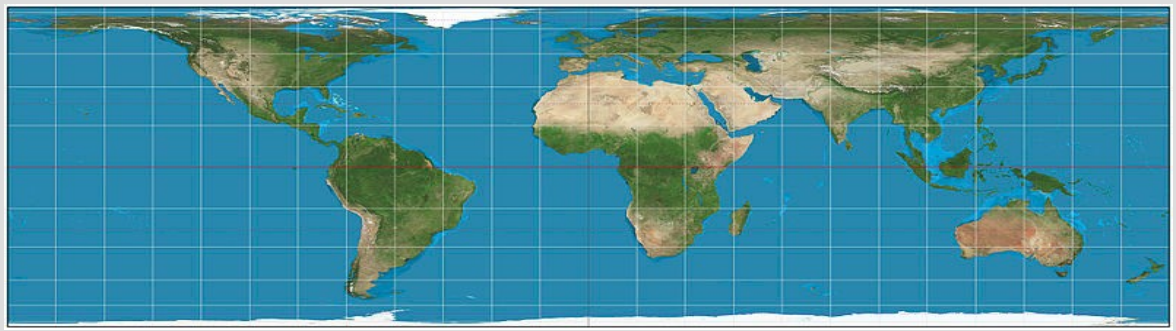
We are just pretending like there is more...

How can we do this?

Multi-dimension arrays

Same way we have 2D maps:
make some assumptions then project





Multi-dimension arrays

A 2D matrix is split up by rows, for example:

```
int x[3][5];
```

We think of this as:

x0,0	x0,1	x0,2	x0,3	x0,4
x1,0	x1,1	x1,2	x1,3	x1,4
x2,0	x2,1	x2,2	x2,3	x2,4

But the computer sees:

x0,0 x0,1 x0,2 x0,3 x0,4 x1,0 x1,1 x1,2...

Multi-dimension arrays

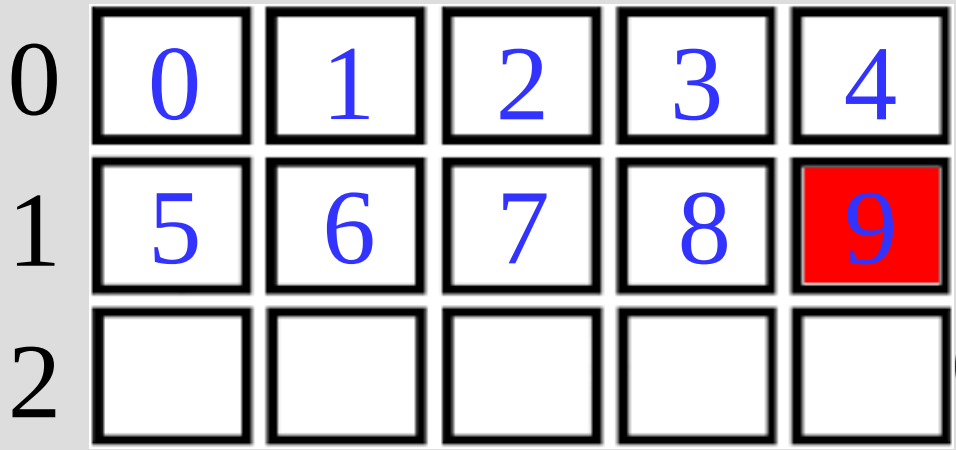
So even if we declare x as:

```
int x[3][5];
```

We can access it by either:

```
x[1][4]
```

```
((int*)x)[1*5 + 4]
```



(see: arrayCheat.cpp)

Stack

I have mentioned a stack a few times before...

This is how function calls work, and they are a specific type of linked list, but with only two simple actions

1. Push (add new item to “top” of stack)
2. Pop (take top item off stack)

Stack

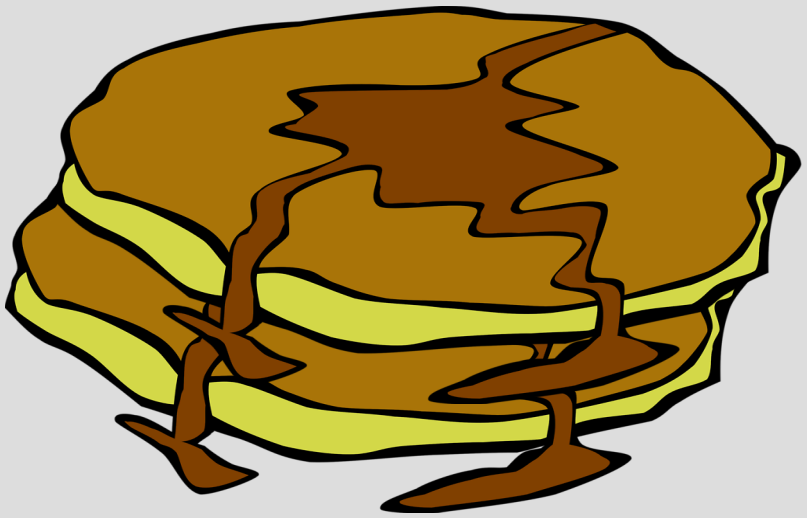
Suppose we have this stack (pancake... yum!):

In this case if we “push”, we flip another pancake on top

Stack

Suppose we have this stack (pancake... yum!):

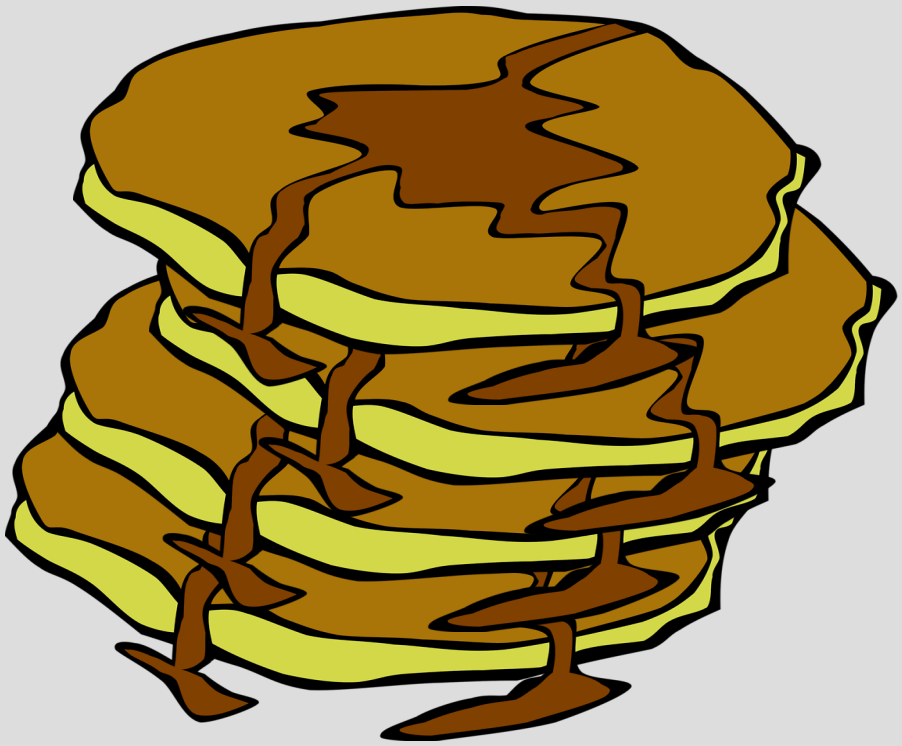
In this case if we “push”, we flip another pancake on top



Stack

Suppose we pushed a few times to get this:

Then a “pop” would remove the top pancake (most recent)



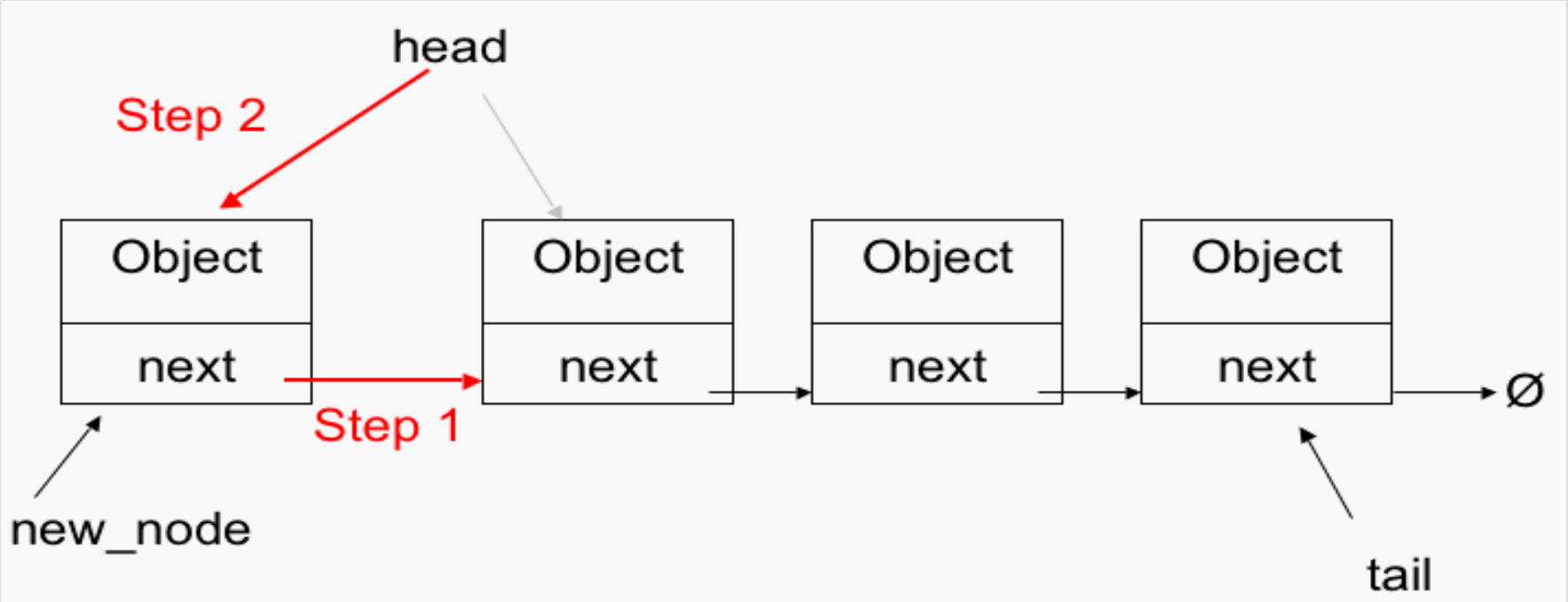
Stack

Suppose we pushed a few times to get this:

Then a “pop” would remove the top pancake (most recent)

Stack

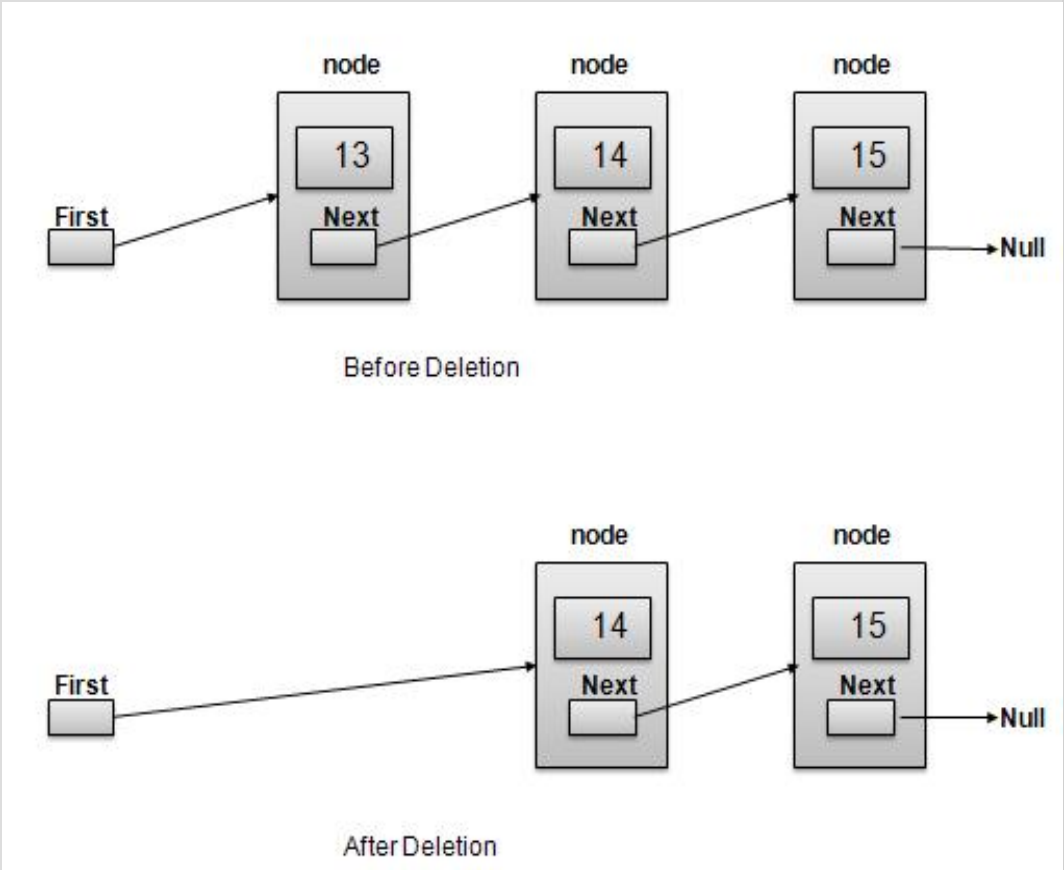
"Pushing" is similar to inserting in linked list:
(Step 0. Make new box)
Step 1. Point new box to old top (next box)
Step 2. Change top to point a new box



Stack

"Popping" can be done by simply changing the "top" to the one below (but memory leak)

The proper way is:
Step 1. Save old top
(so you don't lose it)
Step 2. Change top to one below
Step 3. Delete top
(see: stack.cpp)



Stack vs Heap

There are actually two different parts of memory:

Stack = figures out “early” (normally)

Heap = put here if you use “new”

The way the stack is implemented gives us all our scoping rules
(see: pointerPlaces.cpp)

Stack vs Heap

Differences?

Stack:

- space limited
- automatically handled
- (assumes fixed sizes)

Heap:

- basically unlimited space
- slower to access