COMP20007 Workshop Week 2

Q&A: big-O, arrays, linked lists, stacks, queues

1 Arrays and Linked Lists (Tutorial Q1-Q2)

2 ADT: Stacks, Queues (Q3-Q4)

LAB

- C revision (dynamic arrays)
- intro. to multi-file programming (time permitted)

COMP20007 focuses on algorithm design and efficiency

Problem

???

Pseudocode (Algorithm) **Efficiency**

Our focus is building algorithms and analysing efficiency.

Pseudocode and **big-O** are the main tools.

Programs (in C)

We also do some C implementation.

examples

Problem
Having a collection of data Need to search for some specified elements

Algorithm	Efficiency	When Good?
• get data into an array • do sequential searches		
• get data into an array • sort the array • do binary searches		
…		

Programs

Topic 1: Arrays & Linked Lists

- What are they? What for?
- What are the main differences?

Q2.1: Arrays

Describe how you could perform the following operations on *sorted* and *unsorted arrays*, and decide if they are *O(1)*, *O(log n)*, or *O(n)*, where *n* is the number of elements initially in the array. Assume that there is no need to change the size of the array to complete each operation:

- Inserting a new element
- Searching for a specified element
- Deleting the final element
- Deleting a specified element

How to answer? What's expected?

Q2.1: Arrays

Describe how you could perform the following operations on *sorted* and *unsorted arrays*, and decide if they are *O(1)*, *O(log n)*, or *O(n)*, where *n* is the number of elements initially in the array. Assume that there is no need to change the size of the array to complete each operation.

Operation	Unsorted Arrays	Sorted Arrays
Searching for a specified element	**O(???)** • how (do what) ?	**O( )** •
Inserting a new element	**O( )** •	**O( )** •
Deleting the final element	**O( )** •	**O( )** •
Deleting a specified element	**O( )** •	**O( )** •

Q2.2: Linked Lists

Describe how you could perform the following operations on singly-linked and doubly-linked lists, and decide if they are *O(1)*, *O(log n)*, or *O(n)*, where *n* is the number of elements initially in the linked list. Assume that the lists need to keep track of their final element.

Operation	**Singly**	**Doubly**
Inserting a node at the start	**O( )**	
Inserting a node at the end		
Deleting the first node (at the start)		
Deleting last node (at the end)		

In general:

- *What complexity?*
- *What should be considered when deleting/inserting?*

Topic 2: Stacks & Queues

- Compare

arrays
and
linked lists

stacks
and
queues

?

Topic 2: Stacks & Queues

- Compare

arrays
and
linked lists:

concrete data structures

?

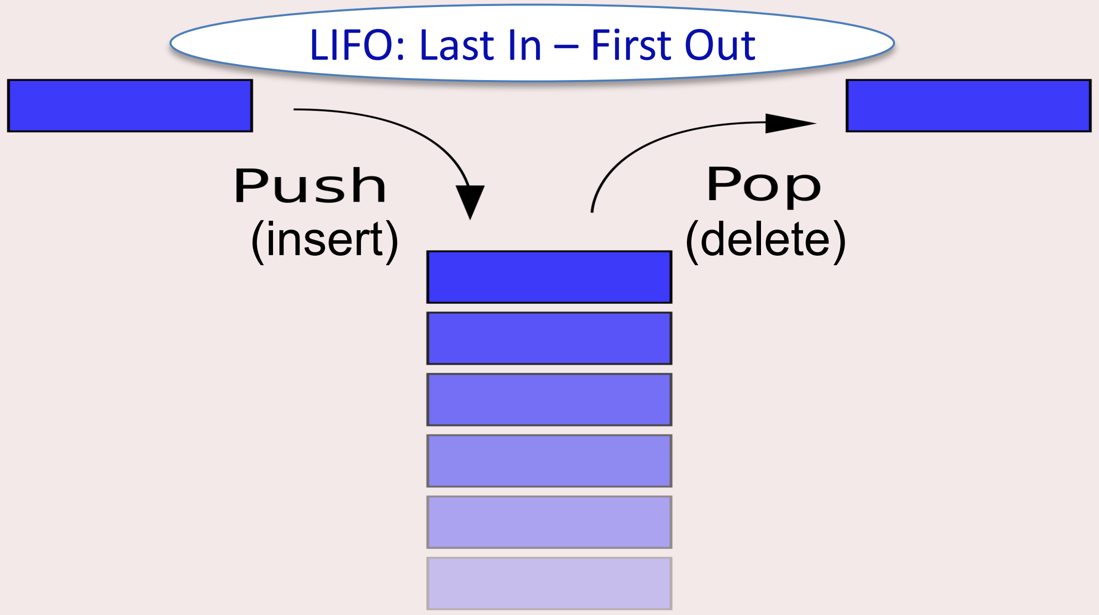
stacks
and
queues:

Abstract Data Types

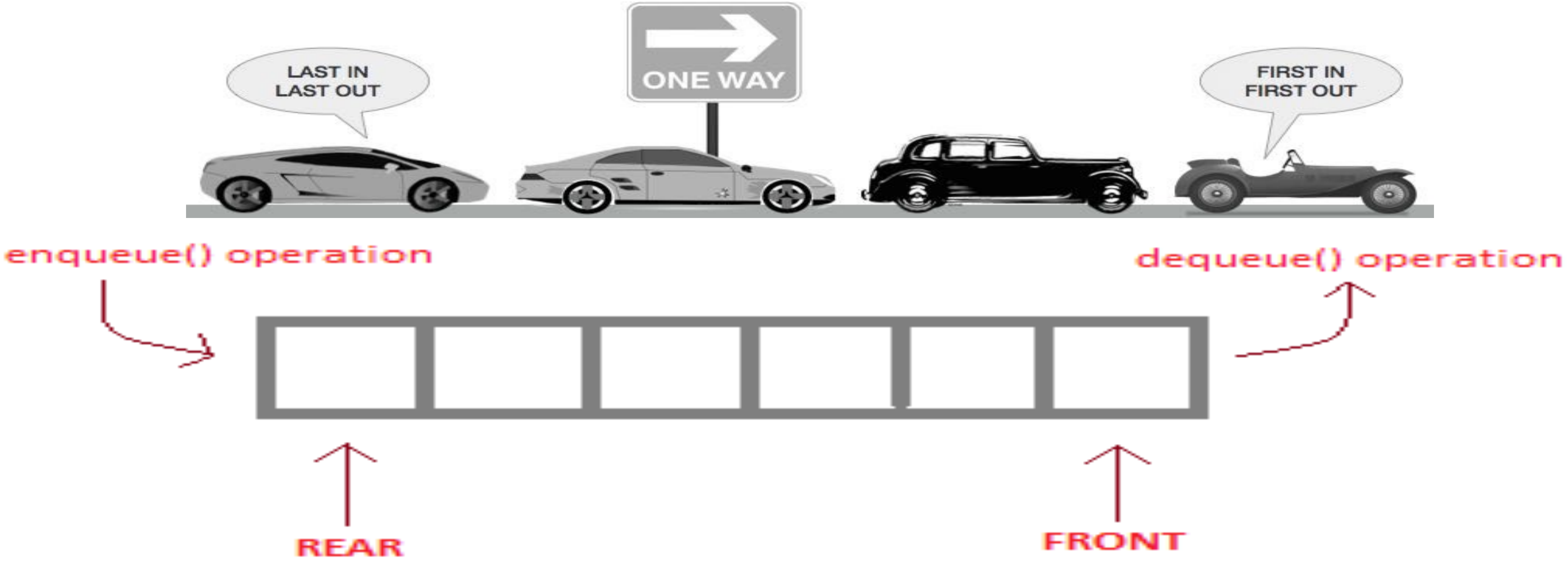
- representation in memory
- interface and implementation of related operations

- only interface of related operations

An Abstract Data Type (ADT): Stack (LIFO)

	 LIFO: Last In – First Out Push (insert) Pop (delete)
http://www.123rf.com/stock-photo/tyre.html	adapted from https://simple.wikipedia.org/wiki/Stack_(data_structure)
Stack Operations	**`push(x)`** : insert element `x` to (the top of) stack **`pop()`** : remove and return an element from (the top of) stack **`isEmpty()`** : check if stack is empty **`create()`** : create a new, empty stack

Another ADT: Queue (FIFO)



Queue Operations	**`enqueue(x)`**: add **`x`** to (the rear of) the queue **`dequeue()`**: remove and return the element from (the front of) the queue **`create()`**: create a new, empty queue **`isEmpty()`**: check if queue is empty

Q2.3: Stacks

Describe how to implement `push` and `pop` using

- an unsorted array?
- a singly-linked list?

Using an (unsorted) array	**Using a (singly-)linked list**

What if the array is full before `push`?

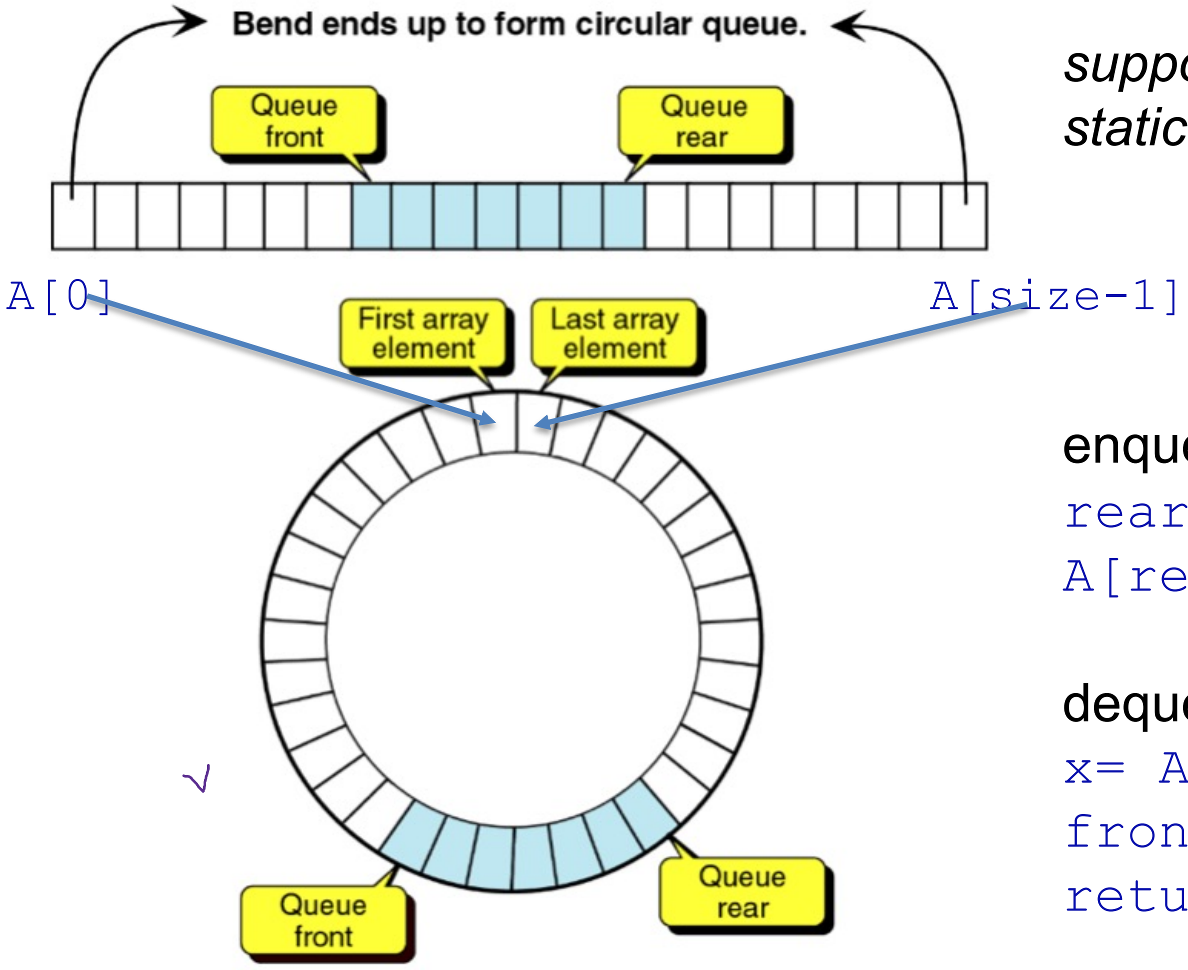
Q2.4: Queues

Describe how to implement **`enqueue`** and **`dequeue`** using an unsorted array, and using a singly-linked list. Is it possible to perform each operation in constant time?

Using an array	**Using a linked list**

What if the array border is crossed?
What if the array is full?

Queue: using circular arrays



suppose using a big-enough static array

enqueue `x`:

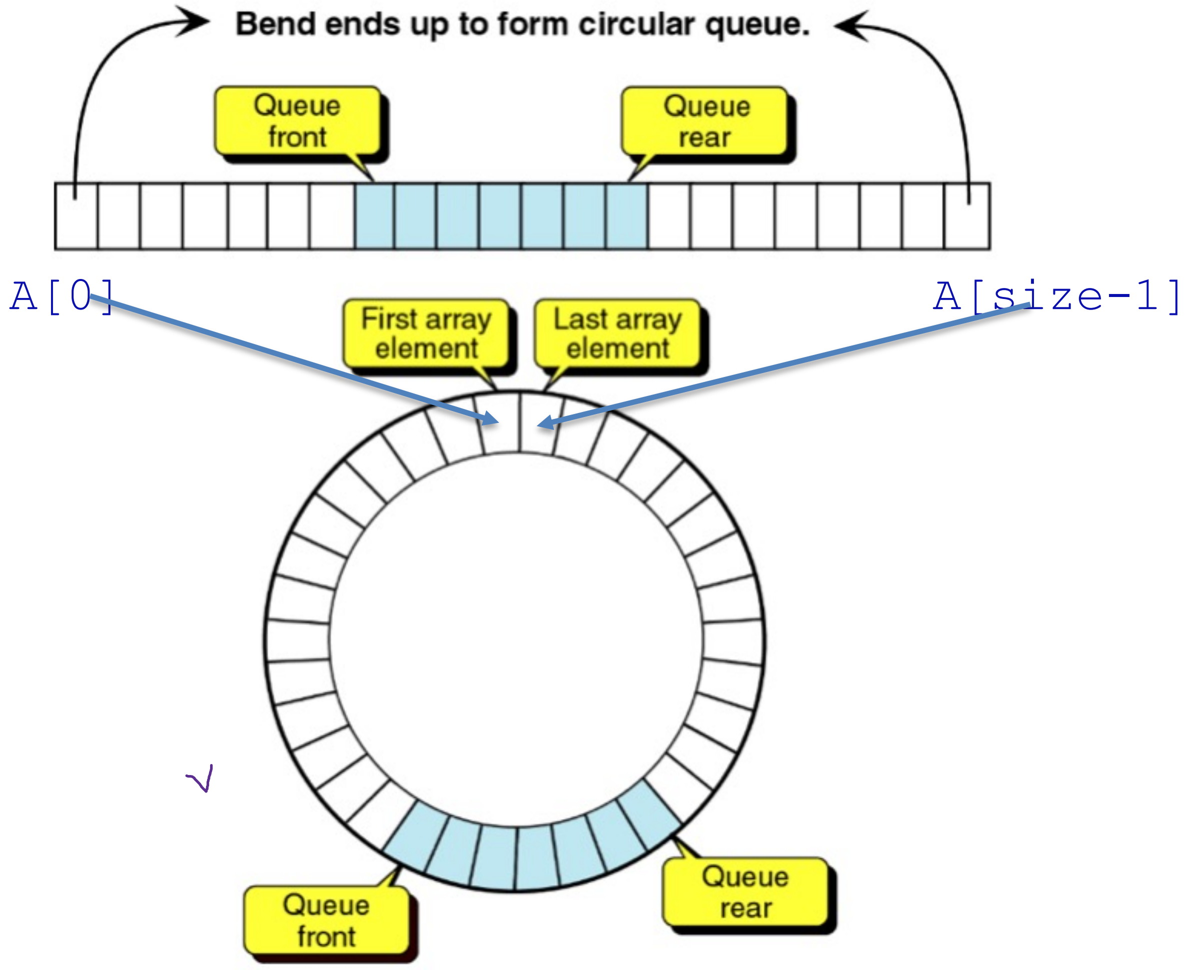
```
rear= rear+1;      ??
A[rear]= x;
```

dequeue:

```
x= A[front];
front= front+1;     ??
return x;
```

Picture source: https://www.chegg.com/homework-help/questions-and-answers/using-system-using-systemcollectionsgeneric-using-systemling-using-systemtext-using-system-q42462337

Queue: using circular arrays



Picture source: https://www.chegg.com/homework-help/questions-and-answers/using-system-using-systemcollectionsgeneric-using-systemling-using-systemtext-using-system-q42462337

suppose using a big-enough static array

enqueue `x`:

```
rear= (rear+1)%size;
A[rear]= x;
```

dequeue:

```
x= A[front];
front= (front+1)%size;
return x;
```

Q2.5 [homework]: Stacks & Queues

If you have access only to stacks and stack operations, can you faithfully implement a queue? How about the other way around?

using stacks to implement a queue

enqueue	dequeue

using queues to implement a stack

push	pop

5-minute break

stretch exercises

networking

Lab Time: Use Ed for exercises and assignments

1. (Together with Anh) Implement functions in `functions.c`, which reviews *function and function parameters*
2. *dynamically resizing arrays* with `malloc`/`realloc` and `free`.

Why `Ed`?

- Strong: powerful editor, shell, compilers, `valgrind`, `gdb`, …
- Safe : codes and files will never be lost
- Sound : codes/files can be accessed from any devices
- Sane : your assignments will be tested on `Ed`

3. (time permitted): break the resizing array code (Problem 2) into two .c and one .h file, then compile them together

Wrap Up

array and linked list as concrete data types.

stack and queue as Abstract Data Types (ADT):

- operations,
- implementation using array and linked list.

Additional Pages

Data Types & ADT

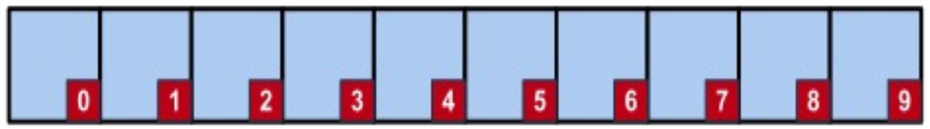
A *concrete data type*, such as array or linked list, specifies a representation of data, and programmers can rely on that to implement operations (such as `insert`, `delete`).

An *abstract data type* specifies possible operations, but not representation. Examples: stacks, queues, dictionaries.

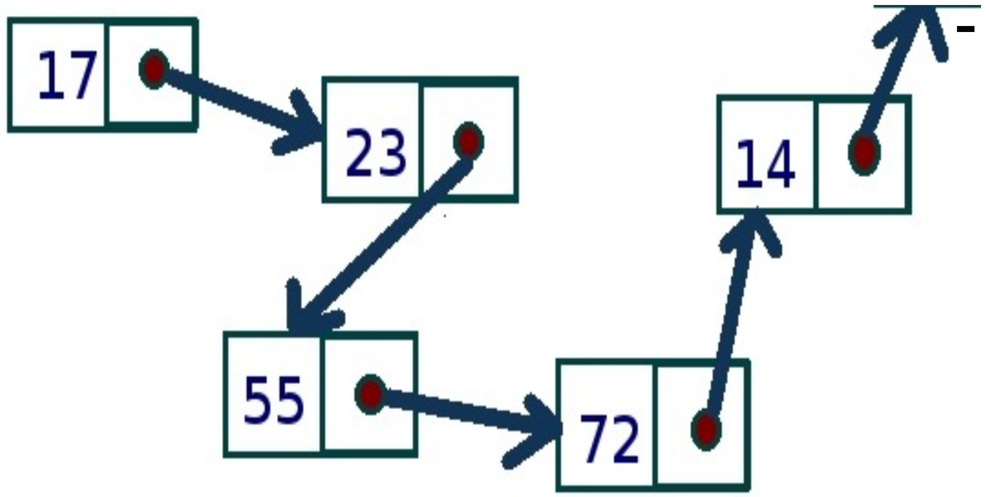
- When implementing an ADT, programmers use a concrete data type. For example, we might attempt to employ array to implement stack.
- When using an ADT, programmers just use its facilities and ignore the actual representation and the underlined concrete data type.

Two concrete data types: Arrays & Linked Lists

Access A[k] in O(1) time!



Access L[k] in O(n) time!



In C:

- How to specify an array? How to traverse it?
- How to specify a linked list? How to traverse it?

Example of using Stacks & Queues ?

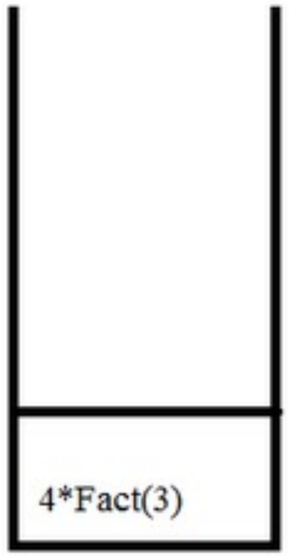
When function call happens previous variables gets stored in stack

Stack is widely used in implementation of programming systems. For example, compilers employ stacks for keeping track of function calls and execution.

Stack for :

Fact(4)

```
int Fact( int n ) {
 if ( n<=1 )
     return 1;
 return n*fact(n-1);
```



After the first call

Image source: http://stackoverflow.com/questions/19865503/can-recursion-be-named-as-a-simple-function-call

Stack is widely used in implementation of programming systems. For example, compilers employ stacks for keeping track of function calls and execution.

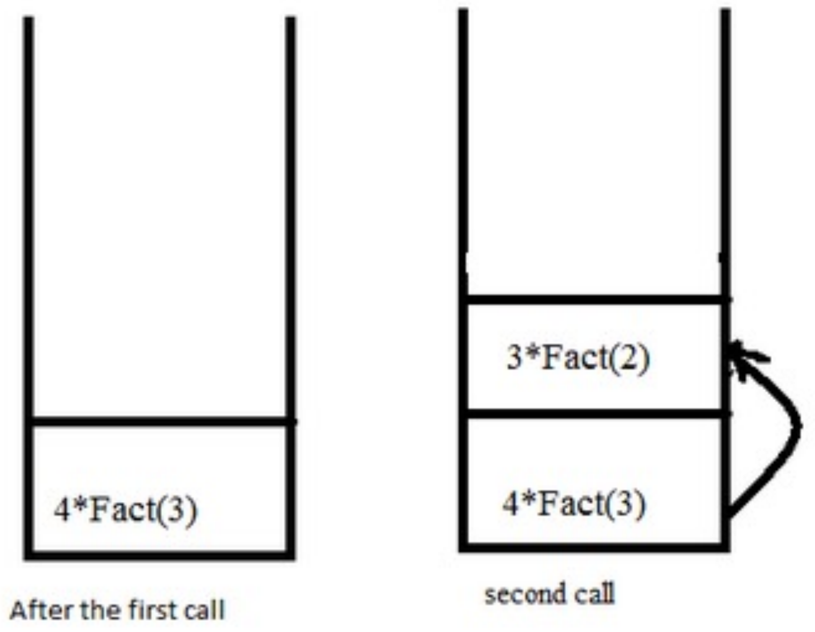
Stack for :

Fact(4)

```
int Fact( int n ) {
  if ( n<=1 )
      return 1;
  return n*fact(n-1);
```

Image source: http://stackoverflow.com/questions/19865503/can-recursion-be-named-as-a-simple-function-call

When function call happens previous variables gets stored in stack



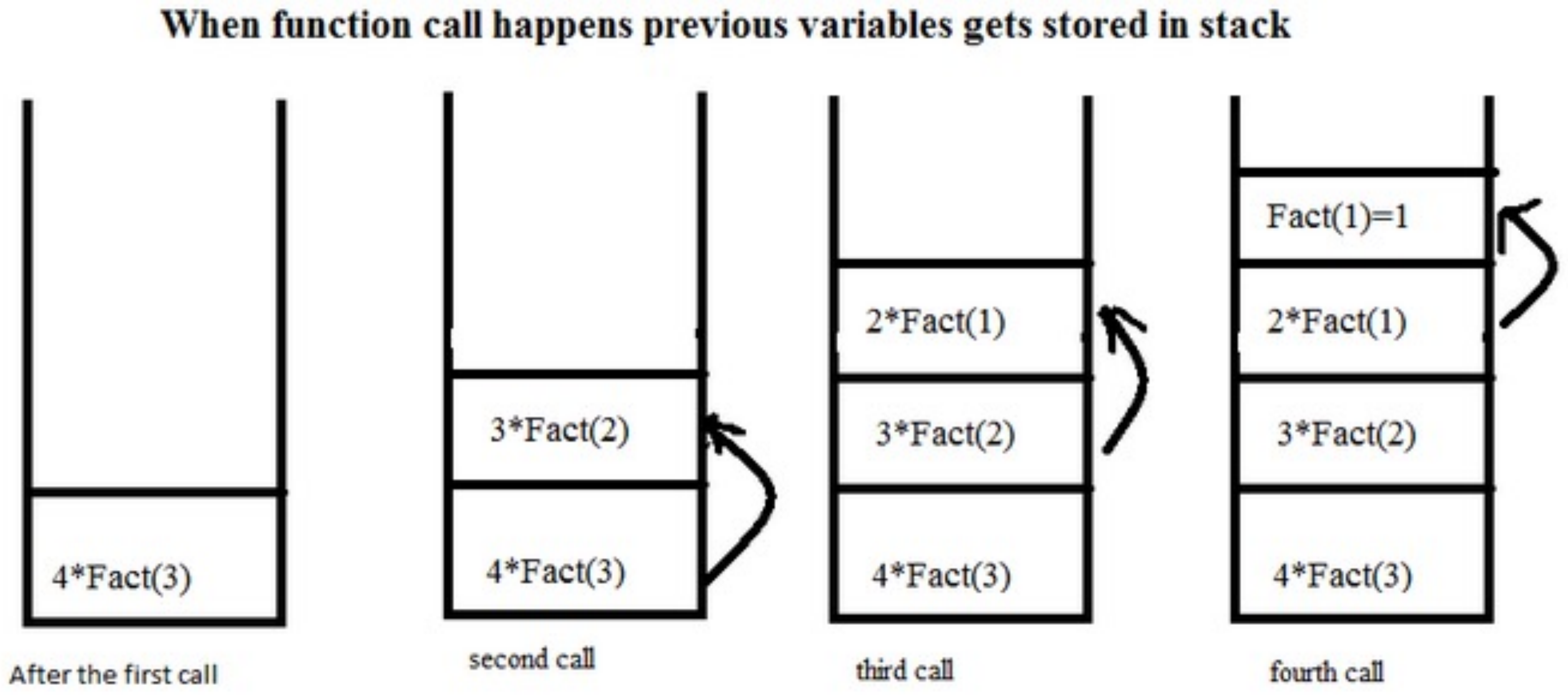
Stack is widely used in implementation of programming systems. For example, compilers employ stacks for keeping track of function calls and execution.

Stack for :

Fact(4)

```
int Fact( int n ) {
  if ( n<=1 )
      return 1;
  return n*fact(n-1);
```

Image source: http://stackoverflow.com/questions/19865503/can-recursion-be-named-as-a-simple-function-call



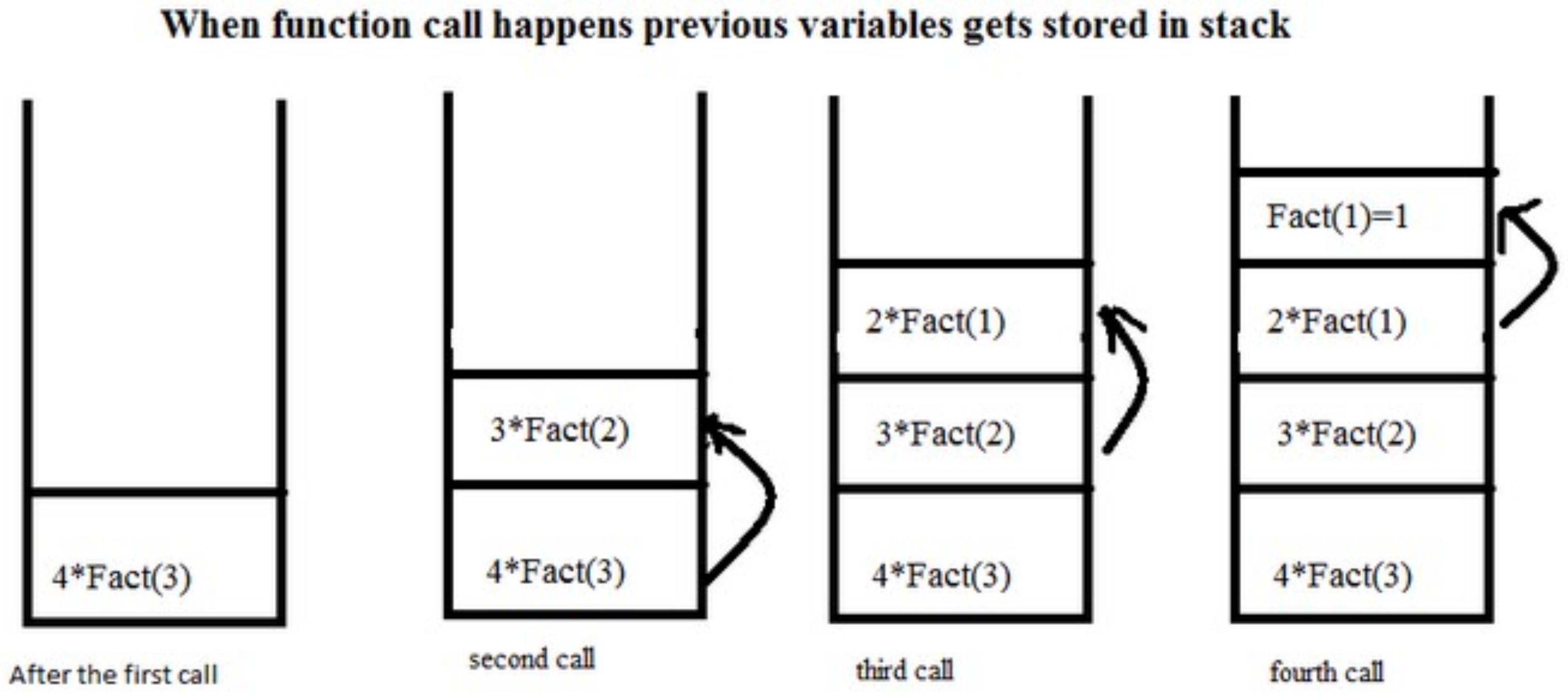
Stack is widely used in implementation of programming systems. For example, compilers employ stacks for keeping track of function calls and execution.

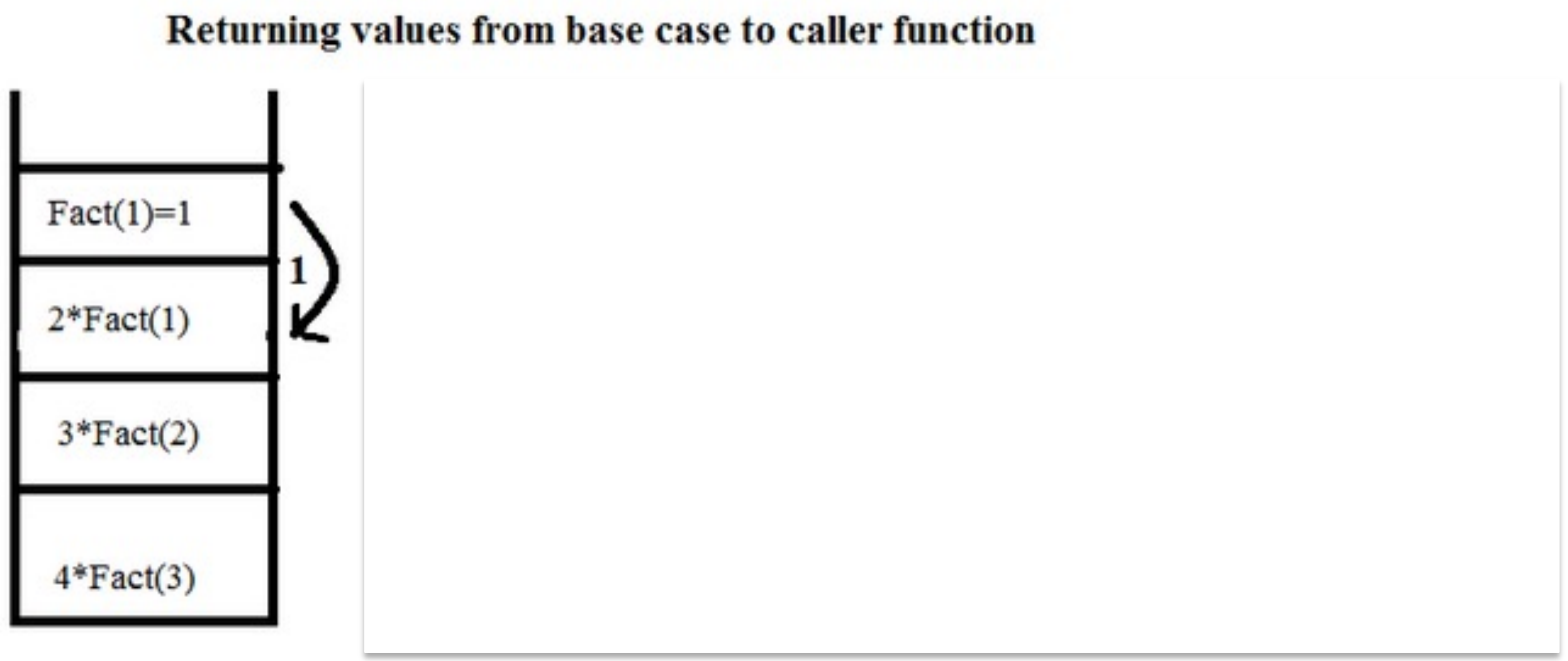
Stack for :

Fact(4)

```
int Fact( int n ) {
  if ( n<=1 )
     return 1;
  return n*fact(n-1);
```

Image source: http://stackoverflow.com/questions/19865503/can-recursion-be-named-as-a-simple-function-call





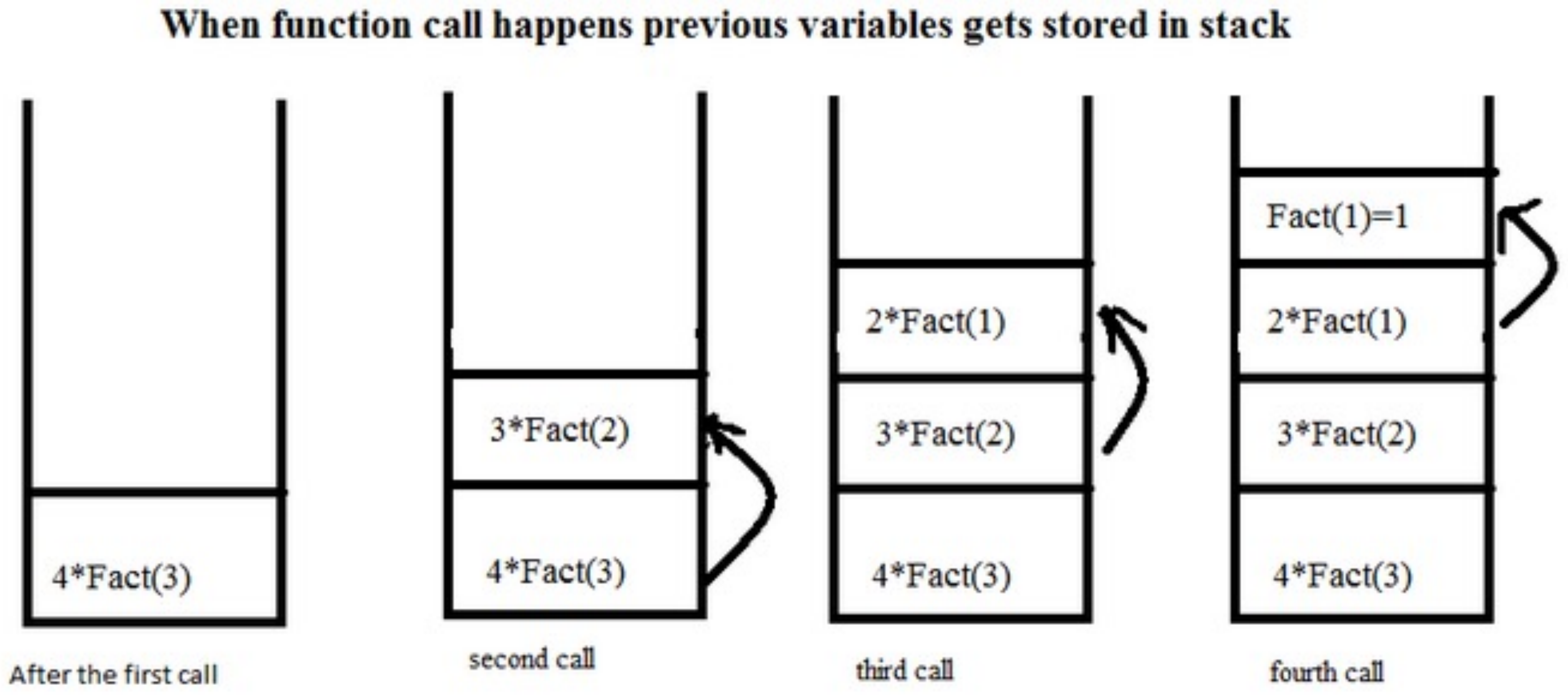
Stack is widely used in implementation of programming systems. For example, compilers employ stacks for keeping track of function calls and execution.

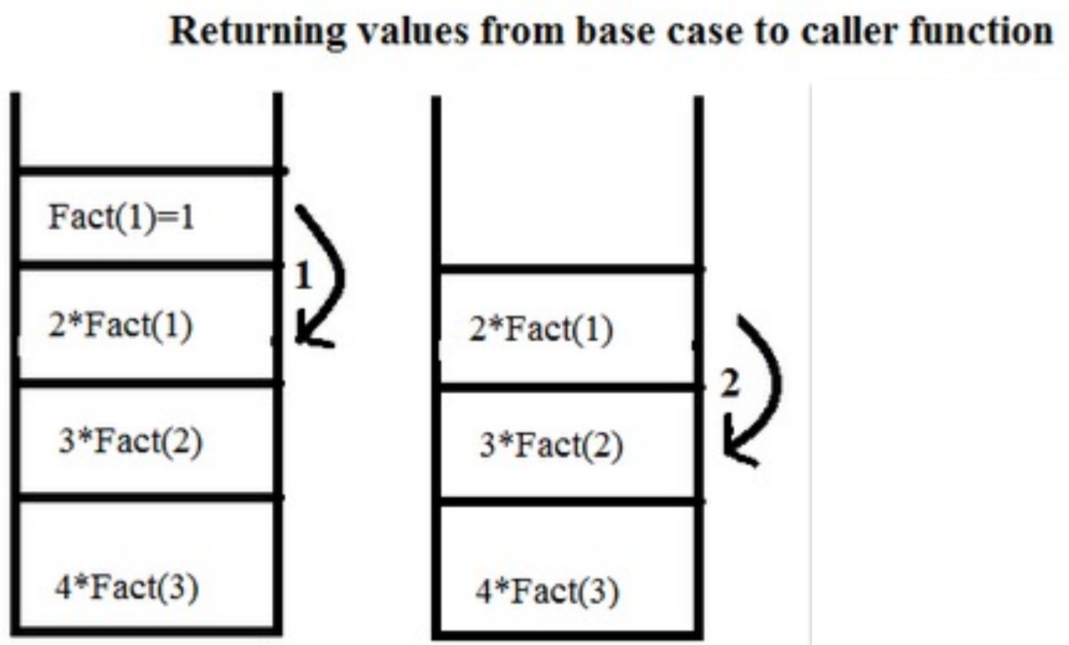
Stack for :

Fact(4)

```
int Fact( int n ) {
  if ( n<=1 )
     return 1;
  return n*fact(n-1);
```

Image source: http://stackoverflow.com/questions/19865503/can-recursion-be-named-as-a-simple-function-call





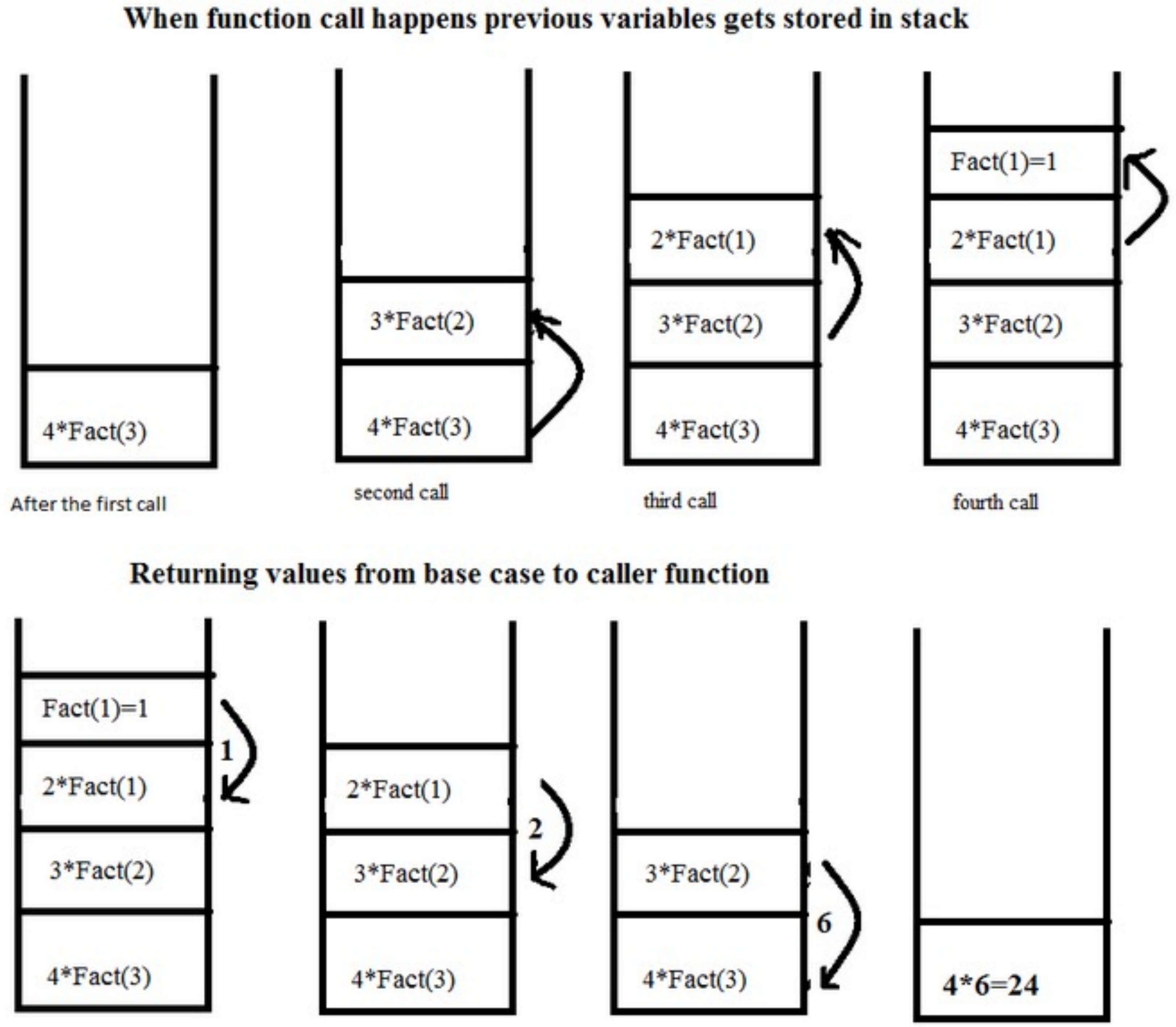
Stack is widely used in implementation of programming systems. For example, compilers employ stacks for keeping track of function calls and execution.

Stack for :

Fact(4)

```
int Fact( int n ) {
  if ( n<=1 )
      return 1;
  return n*fact(n-1);
```

Image source: http://stackoverflow.com/questions/19865503/can-recursion-be-named-as-a-simple-function-call



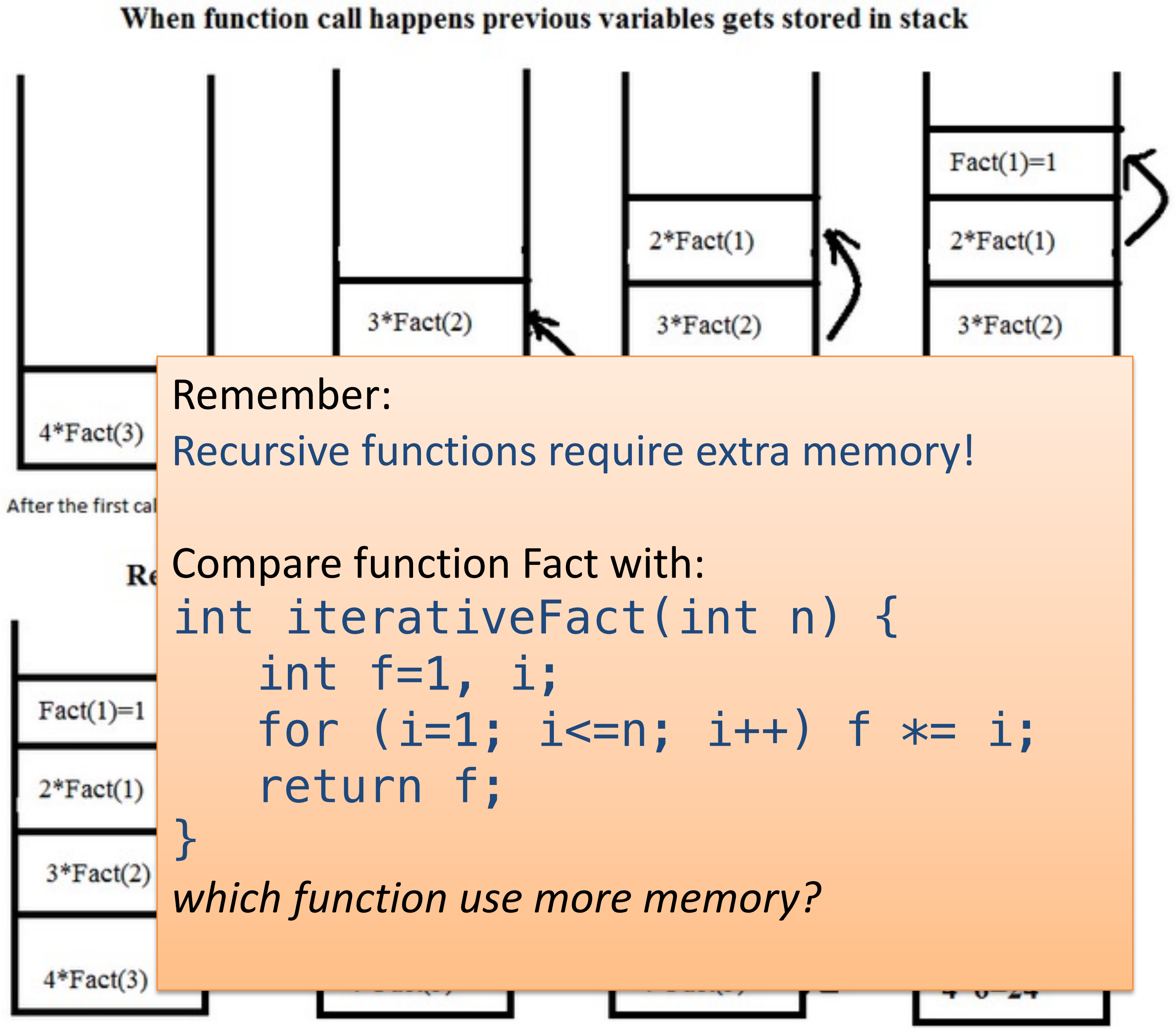
Stack is widely used in implementation of programming systems. For example, compilers employ stacks for keeping track of function calls and execution.

Stack for :

Fact(4)

```
int Fact( int n ) {
 if ( n<=1 )
    return 1;
 return n*fact(n-1);
```

Image source: http://stackoverflow.com/questions/19865503/can-recursion-be-named-as-a-simple-function-call



Remember:

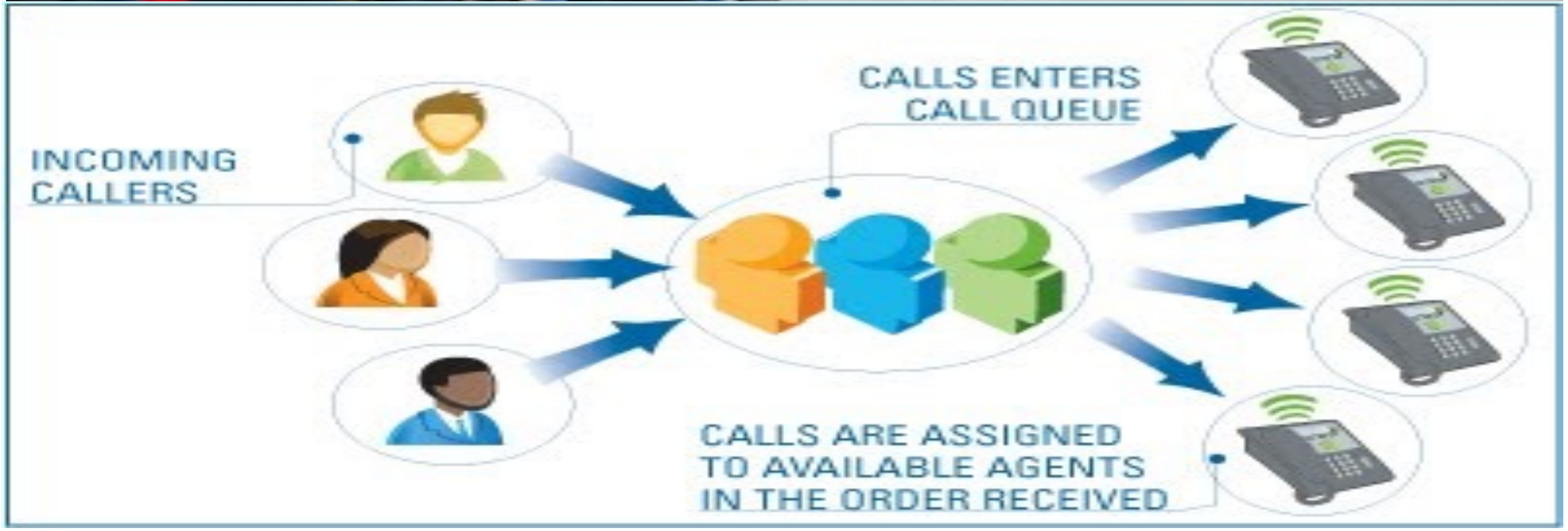
Recursive functions require extra memory!

Compare function Fact with:

```
int iterativeFact(int n) {
   int f=1, i;
   for (i=1; i<=n; i++) f *= i;
   return f;
}
```

which function use more memory?

ADT: Queue (FIFO)


INCOMING CALLERS
CALLS ENTERS CALL QUEUE
CALLS ARE ASSIGNED TO AVAILABLE AGENTS IN THE ORDER RECEIVED