COMPUTERS

Programming Proverbs

From COBOL with Style.

1. Don’t Panic
2. Define the Problem completely.
3. Start the Documentation early.
4. Think First , Code Later.
5. Proceed Top Down.
6. Beware of other approaches
   i.e. Bottom up, Inside Out, Linear.
7. Code in logical units.

From COBOL with Style.

8. Use Perform , Functions and Procedures.
9. Don't GOTO
10. Pretty Print (Improve Listings)
11. Comment Effectively.
12. Use meaningful names.
13. Get the Syntax correct Now.
14. Plan for Change.

From COBOL with Style.

15. Don’t leave the reader in the dust
    (Don’t be clever and obscure).
16. Prepare to prove the pudding.
    (Put debug prints in the first version).
17. Have someone else read the work.
18. Read the manual again.
19. Don’t be afraid to start over.

COMPUTERS

The Basics of Data Structures

What is Data ?

- Numbers.
- Characters.
- Numbers and Characters
- Information in form of :-
- Digitized Pictures
- Digitized Sounds
- Any more suggestions ?

What are Data Structures ?

- Data held in an Organized form.
- Organized for ease of processing.
- Organized for ease of access.
- Grouped into meaningful sets.
- Arranged so that it can be sorted.
- Any more ideas ?

Where are Structures Stored?

- Data held in Computer Memory.
- Held on Disks.
- Held on Tape.
- Any more ideas ?

How are Structures Stored?

- Sequential.
- Direct or Random Access.
- Index Sequential.
- Any more ideas ?

Data Structures

Data Arrays

ARRAYS What are they ?

- The simplest data structure used to represent Vectors or Matrices.
- A organized collection of a similar data type.
- NOTE
- A Vector is a single dimension of data.
- A Matrix is a multi-dimensional data group.

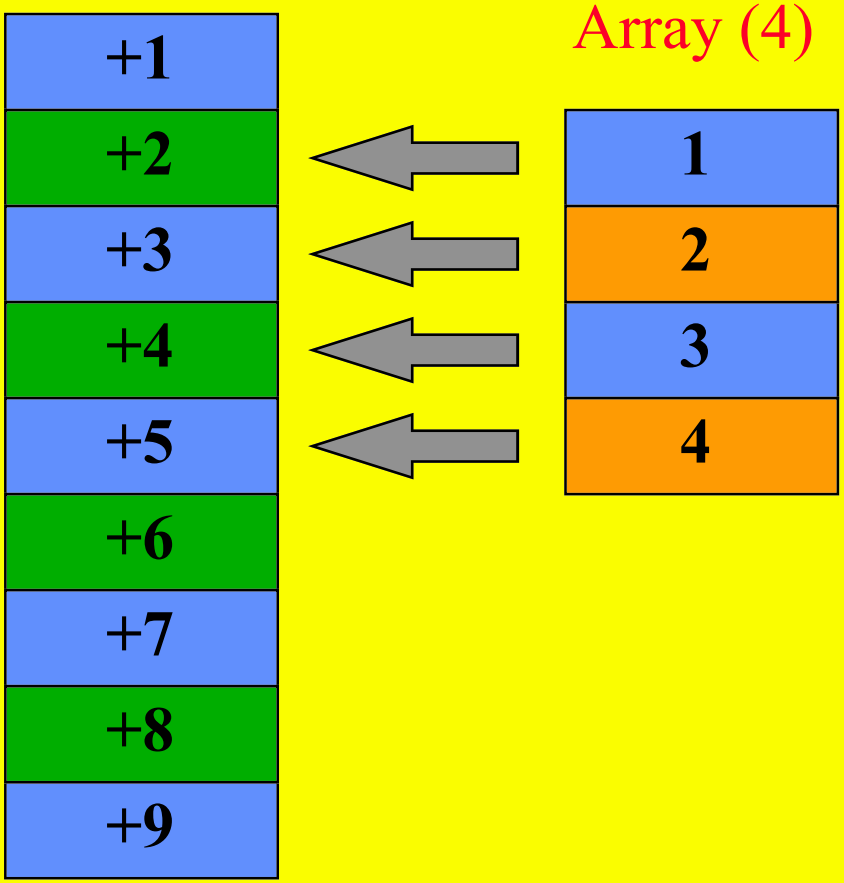
ARRAYS What are they ?

- A One dimensional array is represented in virtually all high level languages by symbols of the form :- Array(Index)
- Multi dimensional arrays are often represented as follows :-
- Array(Dimension 1 , Dimension 2 , ... n)

ARRAYS Storage.

- Arrays are normally stored in a linear memory space.
- So How are we going to store an array of data in memory ?
- Remember . . . .
- If you have to descend to assembly level language you will be responsible for the actual memory location allocations.

Data Storage One Dimension.



This is how the Array locations may be allocated to memory.

ARRAYS Storage.

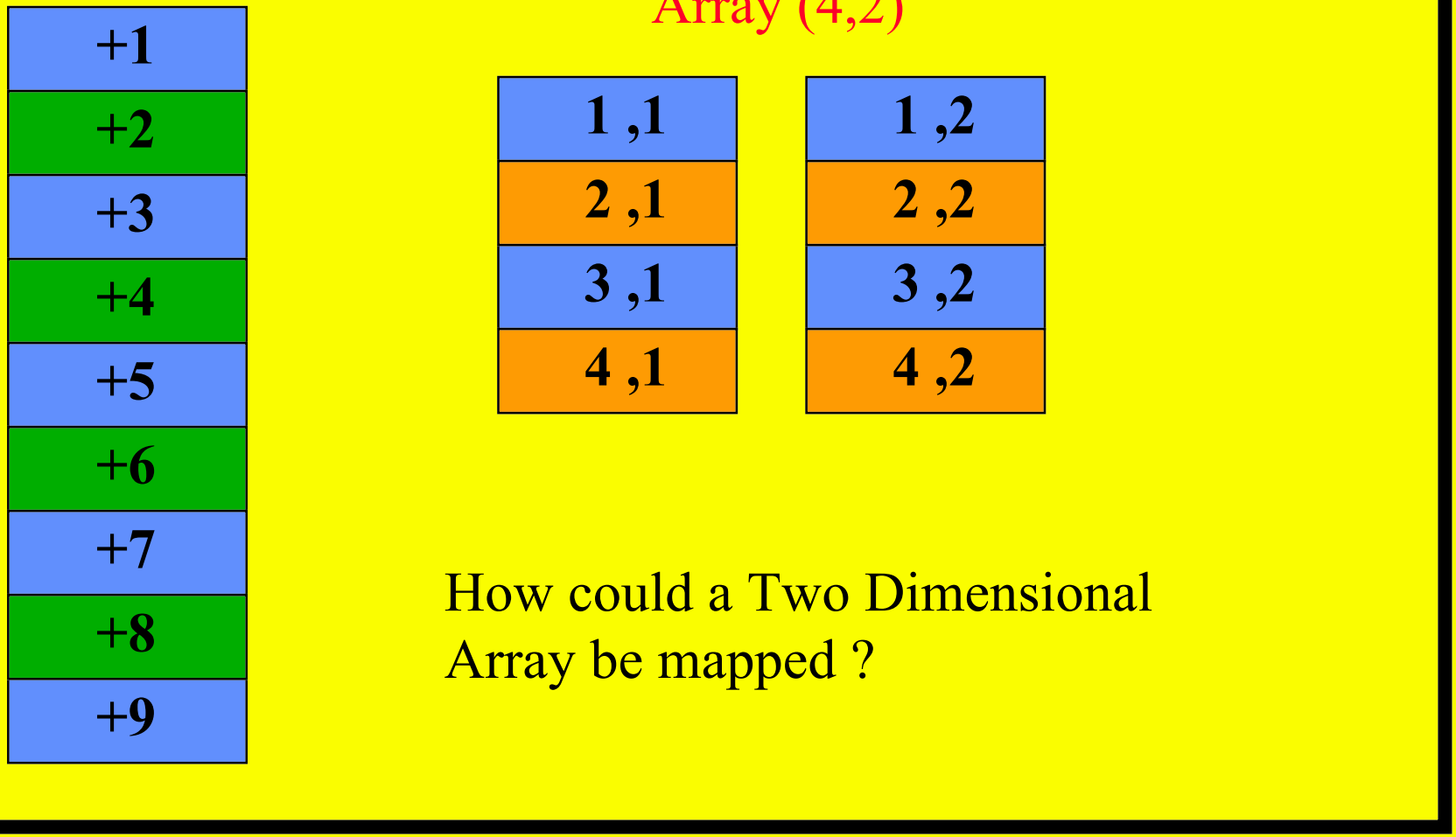
So what if we need a two dimensions array ?

ARRAYS Storage.

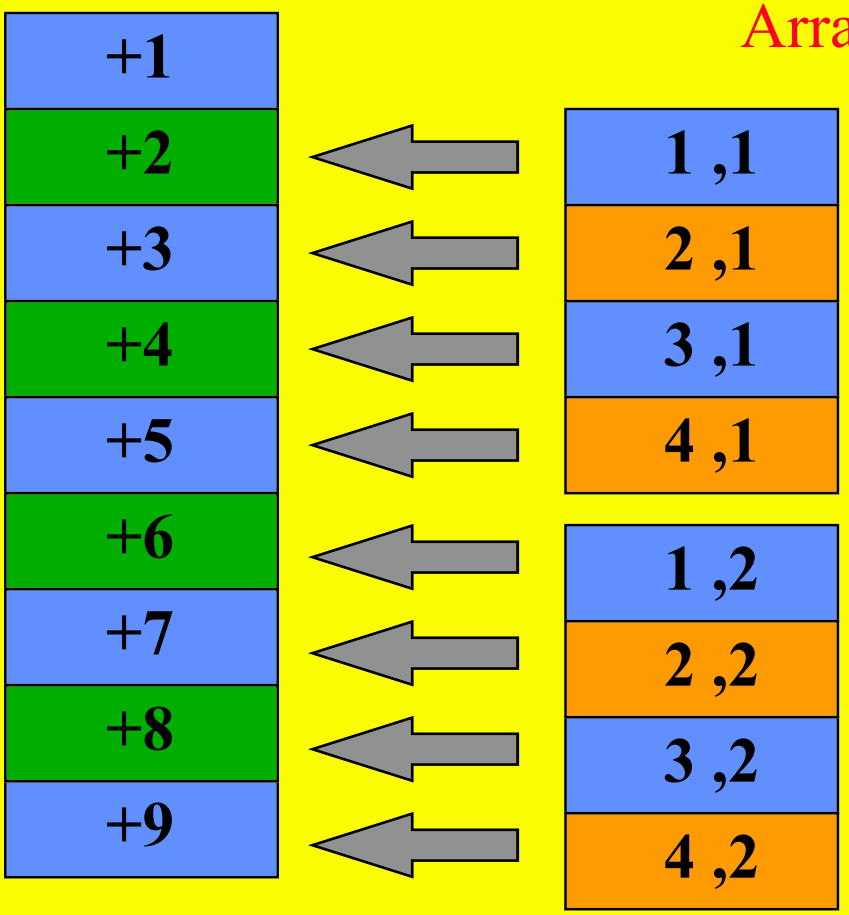
So what if we need a two dimensions array ?

WELL !

Data Storage Two Dimensions.



Data Storage Two Dimensions.



Notice How the Two Dimensional Array is actually stored in memory. It is Stored as a single Dimensioned Array.

ARRAYS Storage.

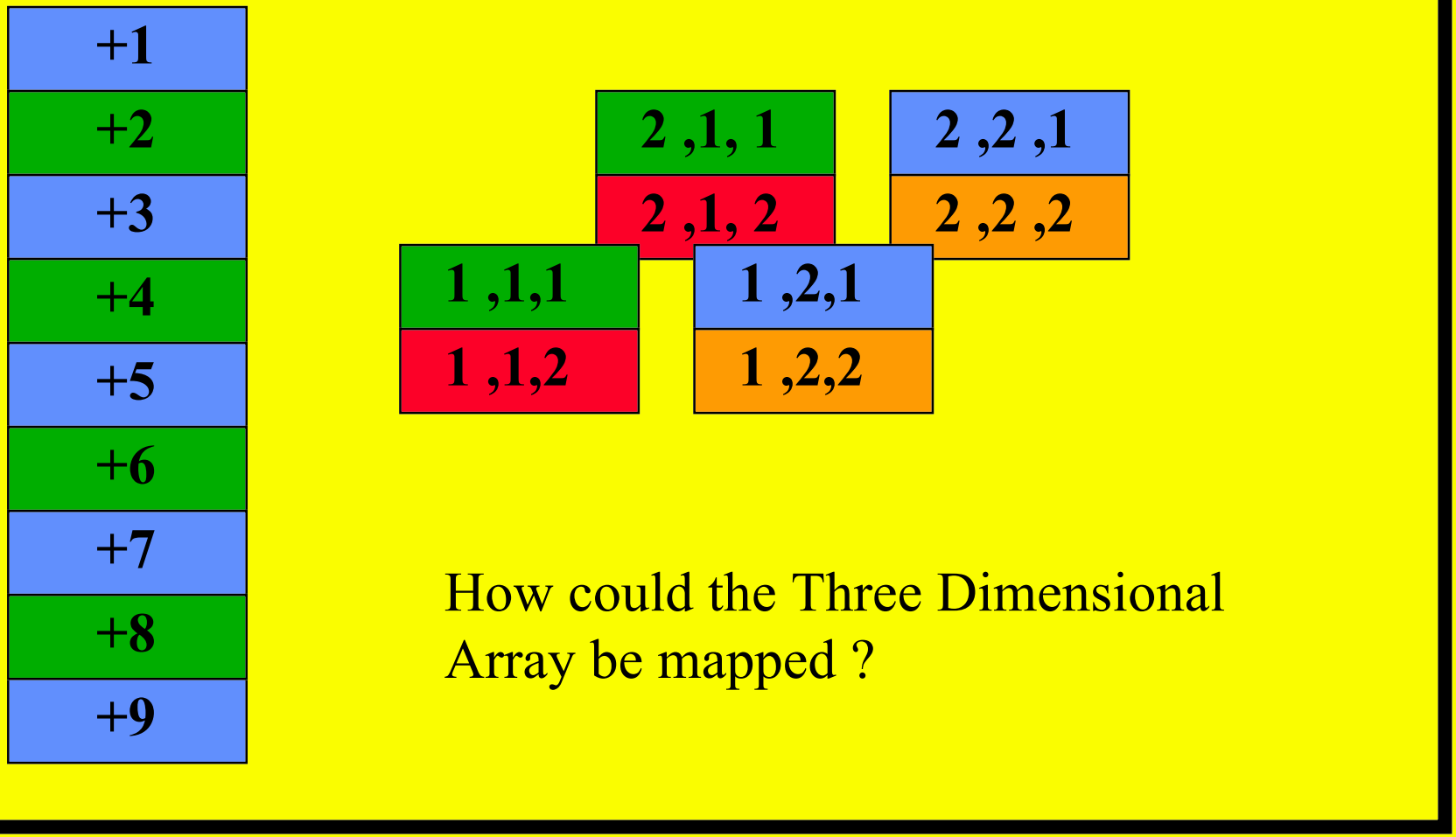
So what if we now need a three dimensional array ?

ARRAYS Storage.

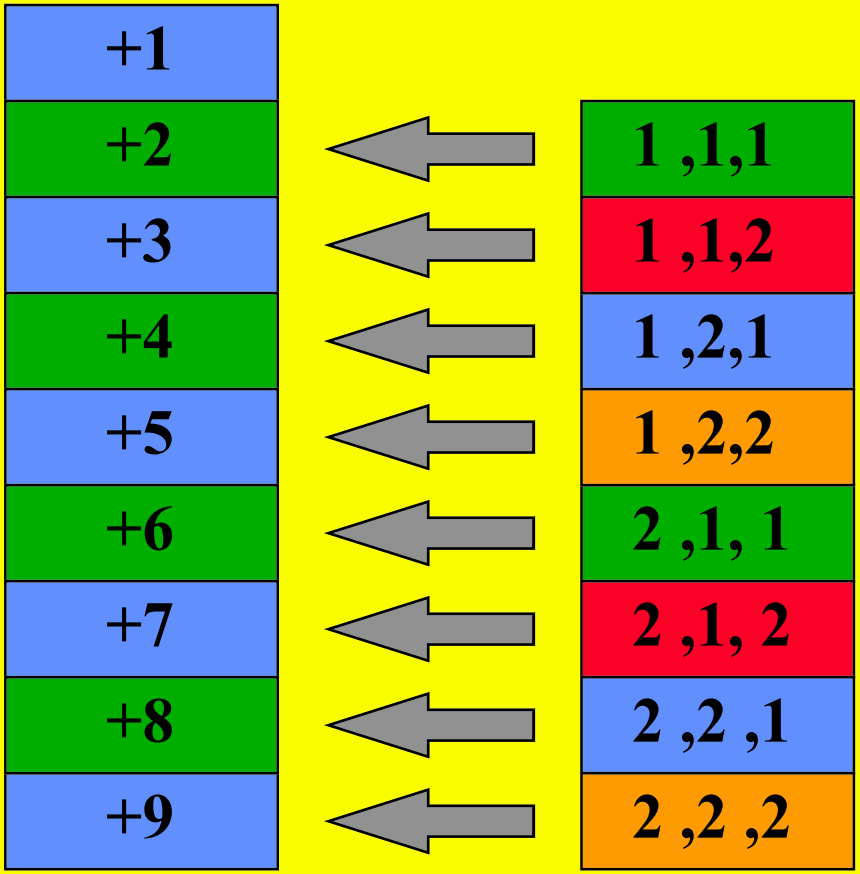
So what if we now need a three dimensional array ?

WELL !

Data Storage Three Dimensions.



Data Storage Three Dimensions.



This is a possible memory allocation.

Hence any number of dimensions Array can be mapped into a single dimension Array.

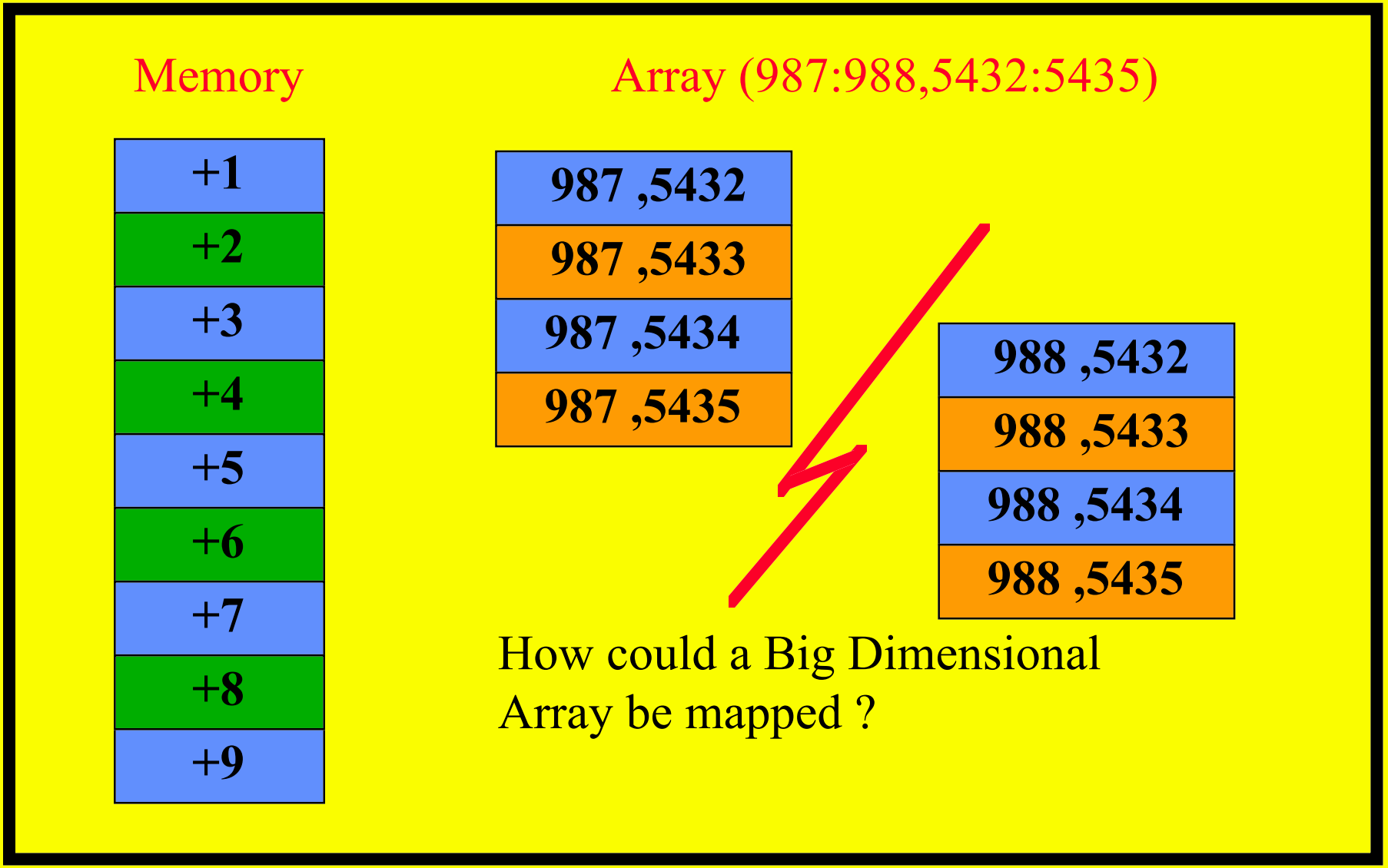
ARRAYS Storage.

- So what if we now need a two dimensional array that has very large indexes ?
- However we know that only a limited range of the indexes will be used.
- Say for example 987 to 988 and in the other dimension 5432 to 5435.
- What Now ?

ARRAYS Storage.

- Do we create an array of 998 by 5435 ?
- Maybe . . If we are Microsoft our array would probably be 1000 by 6000 (Large memory chips are cheap so we are told).
- However another possibility is using a restricted "bounds checked" array.
- Some compilers have this as a build option.
- So what does this look like ?

Data Storage Big Dimensions.



Data Storage Big Dimensions.

Becoming too obvious.

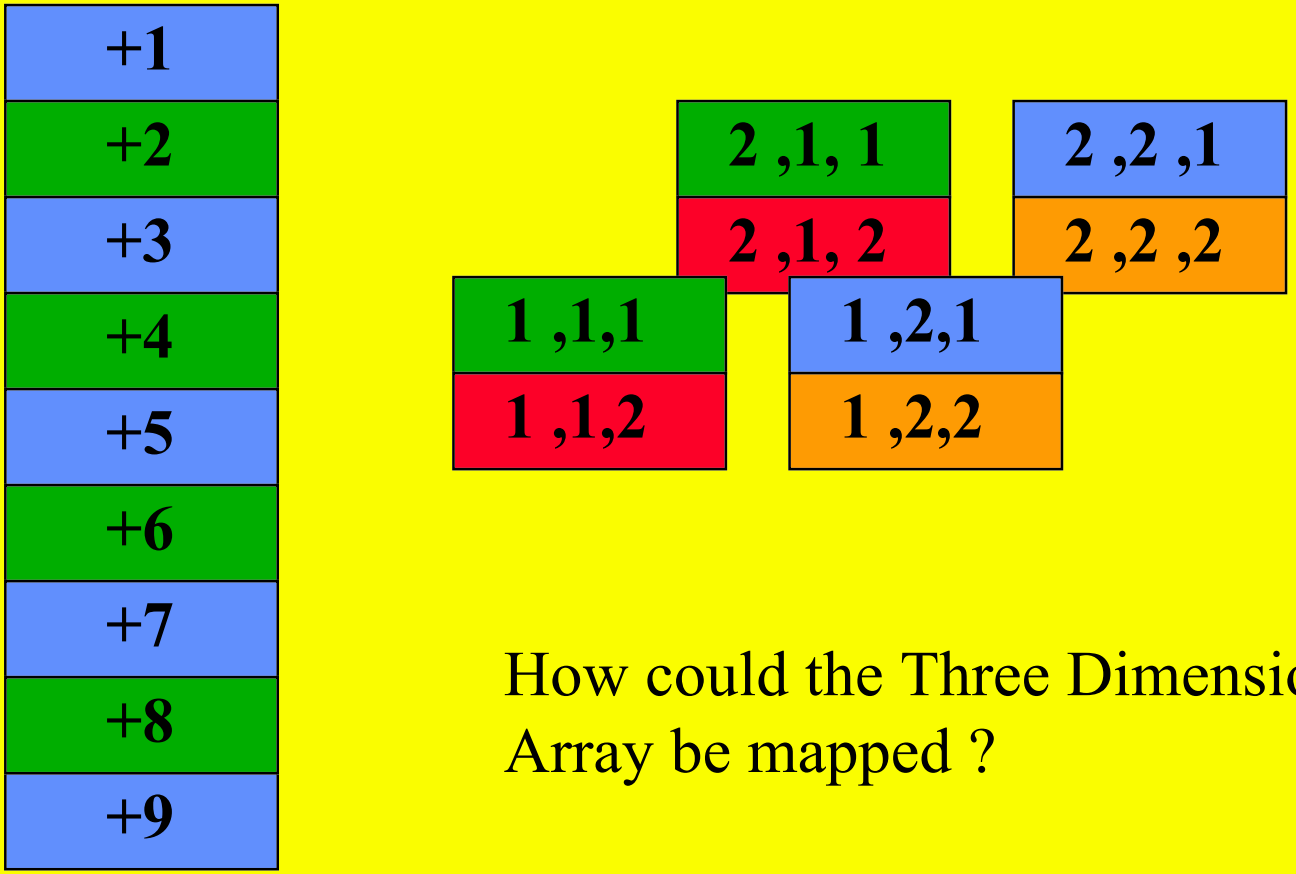
However you must check the bounds and correct for the offsets before access to the address.

ARRAYS Storage.

WARNING

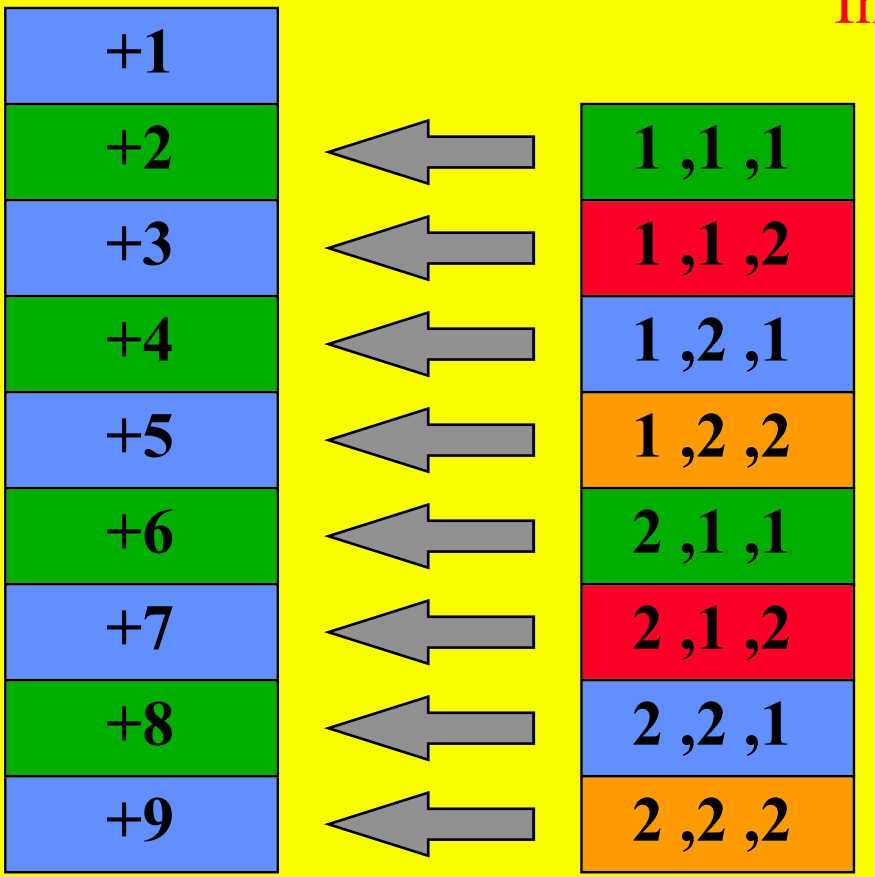
- NOTE High Level Languages DO store their Arrays Differently.
- Sometimes First dimension stored First
- Sometimes Last dimension stored First .
- This can have great importance when accessing Arrays at Machine level or via differing High Level Languages.

Data Storage Dimensions.



How could the Three Dimensional Array be mapped ?

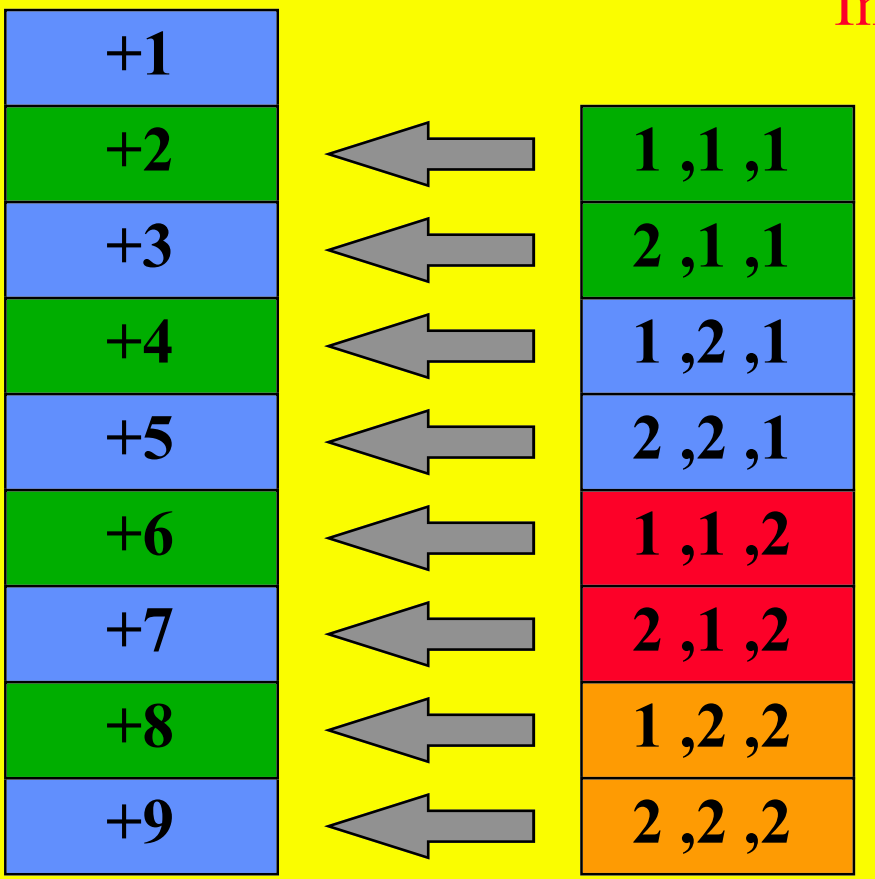
Data Storage Dimensions.



Just the Same as our earlier example.

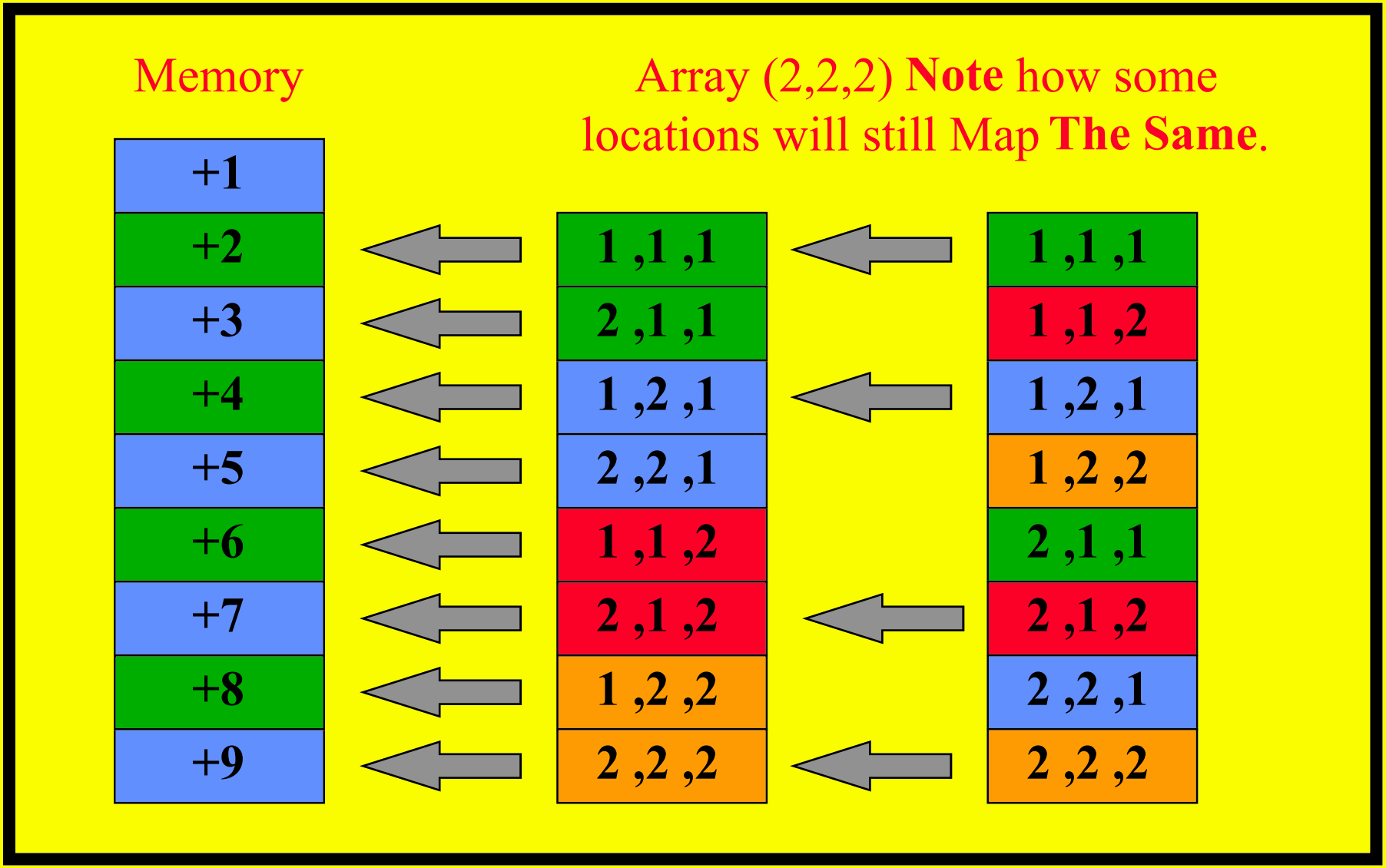
However . . .

Data Storage Dimensions.



Note how the order has changed from previous example.

Data Storage Dimensions.

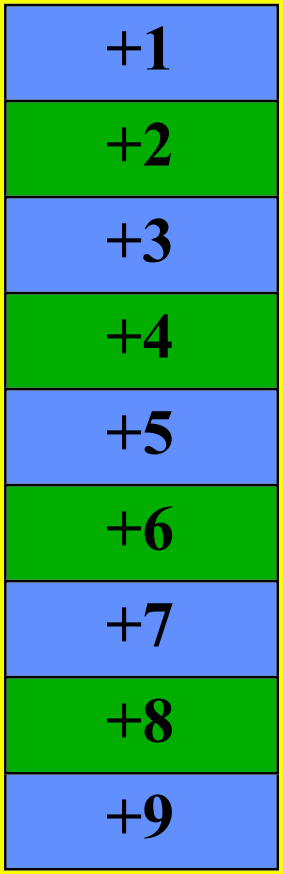


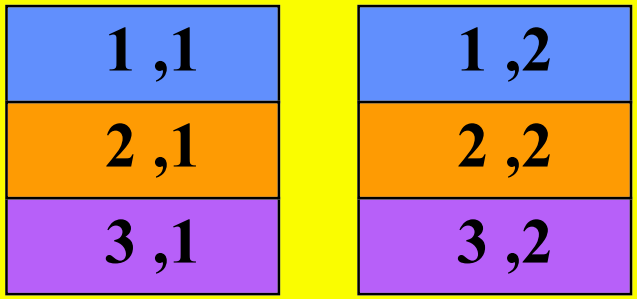
ARRAYS Storage.

WARNING

- NOTE High Level Languages Implement their Arrays Dimensions Differently.
- The Zero Dimension may or may NOT be available.
- The Zero Dimension may be OPTIONAL.
- This will effect your Access calculations.

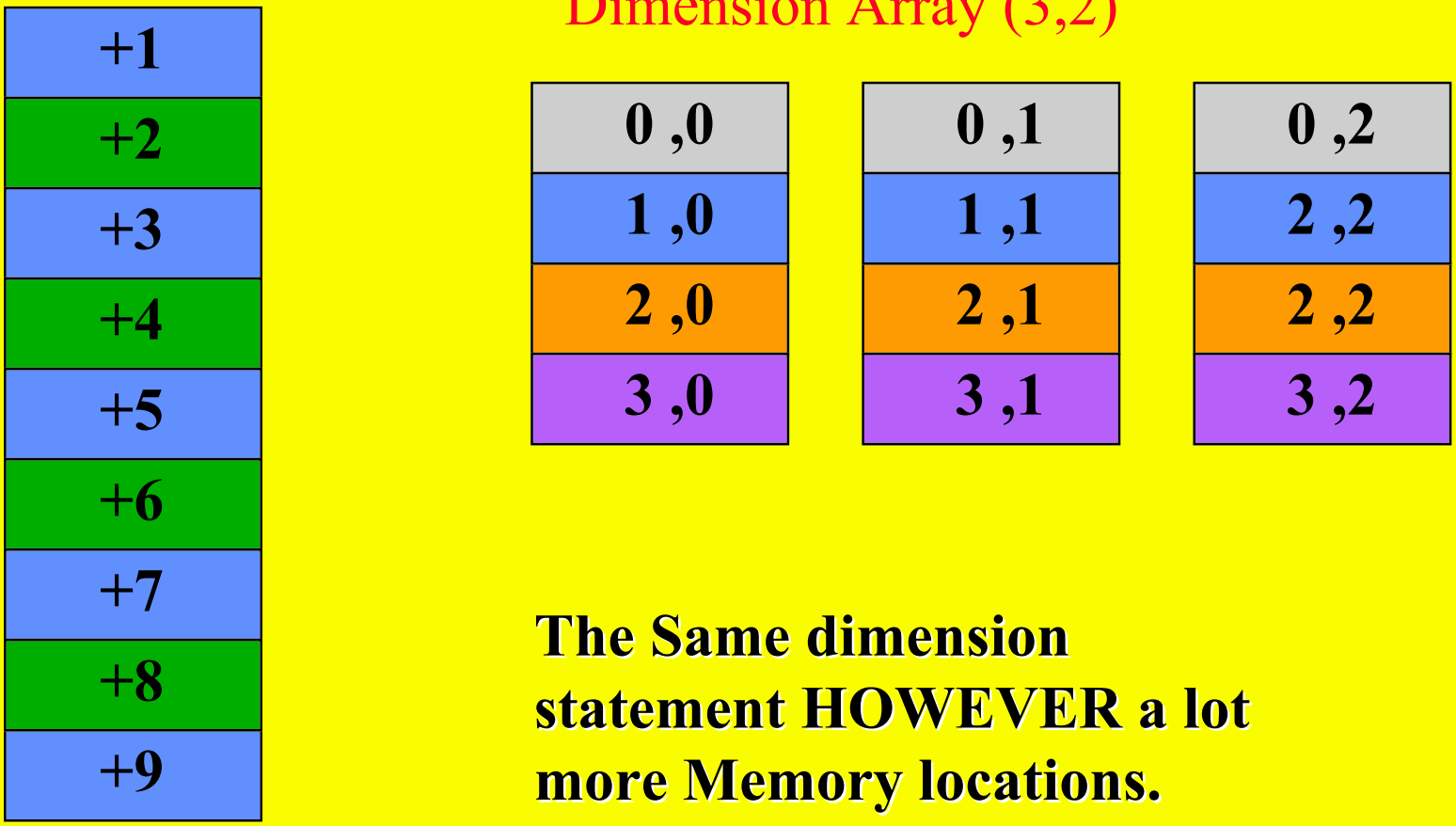
Data Storage Dimensions.





This is one option of Representation.

Data Storage Dimensions.



We have now covered

Single Dimension Arrays

We have now covered

Single Dimension Arrays

Multi Dimension Arrays

We have now covered

Single Dimension Arrays

Multi Dimension Arrays

Bounds Checked Arrays

So Now Its Your Turn

Optional Exercise No. 1.

Design a programme to Implement a Three Dimensional Array in an Single Integer Array.

The Three Dimensional Integer Array bounds are ({1001:1005},{503:510},{711:722}) .

REMEMBER

Check bounds and Error Report if Address Bounds supplied are NOT acceptable.

Data Structures

Data Arrays

Programmed Specials

ARRAYS the Specials.

- These are Arrays that are application specific and are managed under programme control only. Examples are :-
- Triangular Arrays.
- Ragged Arrays.
- They can be best described pictorially so starting with the Triangular Array.

The Triangular Array.

Distance in Km between Towns.

Distance	Calais	Dijon	Lyon	Paris
Calais		603	715	292
Dijon	603		192	313
Lyon	715	192		461
Paris	292	313	461	

Example of the Full Array.

The Triangular Array.

Distance in Km between Towns.

Distance	Calais	Dijon	Lyon	Paris
Calais				
Dijon	603			
Lyon	715	192		
Paris	292	313	461	

Array with Redundant Information Removed.

ARRAYS the Specials.

- Note that with Triangular Arrays More than 50% of the Information is redundant.
- Which begs the Question ...
- Why Store it in the First Place ?
- Note A little Extra code could save a large amount of Data Storage.

ARRAYS the Specials.

- The Ragged Arrays is a very effective way of storing data in a compact manner.
- Each element in the array will either have a terminator or a Second Array will hold the Data Length and its Access Address.
- And so the Ragged Array.

The Ragged Array.

Addr	Size	Contents
00	06	MONDAY
06	07	TUESDAY
13	09	WEDNESDAY
22	08	THURSDAY
30	06	FRIDAY

Note. The Addr Information is often held in another array so that array elements can be accessed quickly.

Optional Exercise No. 2.

Design a programme to access information both Ragged and Triangular Arrays.

What problems are you likely to experience when loading data into a Ragged Array.

What if an Entry in a Ragged array is to be Deleted or Changed (How do you cope) ?

Can the contents of a Ragged Array be Sorted ?

Data Structures

Data Arrays

Overlays .. etc.

Data Types.

- As we described earlier Data can be presented in a number of forms.
- However some computer languages allow the user to define their own Data Types.
- This adds what may appear an additional complication to Arrays.
- Consider the situation :-

Data Types.

- Our user defines a Data type as follows :-
- CHARACTER NAME(10)
- INTEGER GROUP
- CHARACTER TYPE(4)
- REAL-NUMBER WAGES

Data Types.

- And each item takes the following space :-
- CHARACTER NAME(10) 10 Bytes
- INTEGER GROUP 02 Bytes
- CHARACTER TYPE(4) 04 Bytes
- REAL-NUMBER WAGES 08 Bytes
- How much space needs to be reserved ?
- The total structure use is 24 Bytes

Data Types.

Addr	Size	Type
00	10	CHARACTER
10	02	INTEGER
12	04	CHARACTER
16	08	REAL
24	00	Next free location

Stored as shown above

Data Types.

- So although the Data type consists of many different data variants it can still be considered to be a single dimension Byte or Character Array.
- So an array of user defined data types in essence only increases the number of access dimensions by one.
- So What about access to the elements ?

Data Types.

- Just as easy . .
- To access any individual element the system only needs to keep an offset to that element.
- Add it to the start of the arrays element address and the entity can be selected.
- Quite often an Offset table will exist for the Data type.

Data Types.

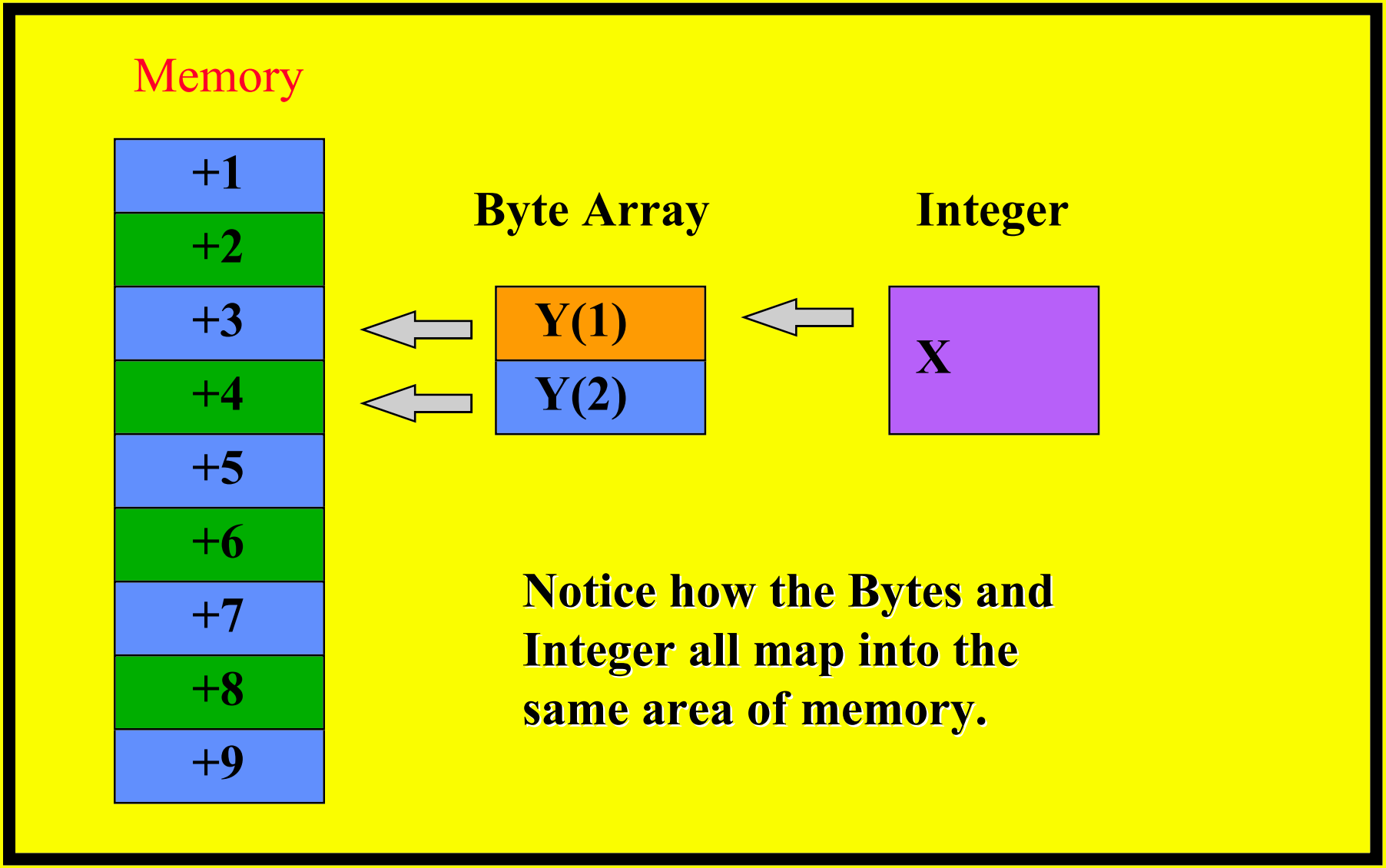
Offset	Size	Variable Name
+00	10	NAME
+10	02	TYPE
+12	04	GROUP
+16	08	WAGES

Example of an Offset Table.

Data Types.

- One other variation that you may encounter is the Overlay.
- Some languages do NOT support user defined data types so you implement them by overlaying one data type on another.
- OVERLAY INTEGER X WITH BYTE ARRAY Y(2)
- This would give :-

Data Storage Overlays.



Data Types.

- So why would we want to overlay data.
- Consider how much quicker it would be to convert 8 bit data stream to a 16 bit data stream with previous structure.
- Access the bytes or the full value immediately without the need to shift and mask values.
- You may be able to suggest a few reasons !

Data Structures

What is Next ?

Covering the Topics of

STACKS

Covering the Topics of

STACKS

QUEUES

Covering the Topics of

STACKS

QUEUES

Linked Lists

Data Structures

The Basics of Stacks and Heaps

What is a Stack or Heap ?

- A simple data structure that enables data to be transferred or stored in :-
- A First In Last Out order (FILO)
- or more commonly known as
- Last In First Out (LIFO)

How is a Stack Implemented ?

- Data is stored in a Linear structure.
- The Linear structure may be held in or on any directly accessible data store.
- A pointer is allocated to identify where the item of data is to be placed.
- Once the data is stored the pointer is adjusted ready for the next data load.

How is a Heap Implemented ?

- Data is stored in a Linear structure.
- The Linear structure may be held in or on any directly accessible data store.
- A pointer is allocated to identify where the item of data is to be placed.
- Once the data is stored the pointer is adjusted ready for the next data load.

Are Heaps & Stacks the same ?

- In a manner "YES" however there are some minor differences.
- The differences relate to how the data pointer is adjusted for the next store location access.

Are Heaps & Stacks the same ?

- In a manner "YES" however there are some minor differences.
- The differences relate to how the data pointer is adjusted for the next store location access.

SO . . .

Specifically the Stack

- A system that only has a Stack then it grows either from :-

. . . .

The Bottom of Memory Up

or

The Top of memory Down

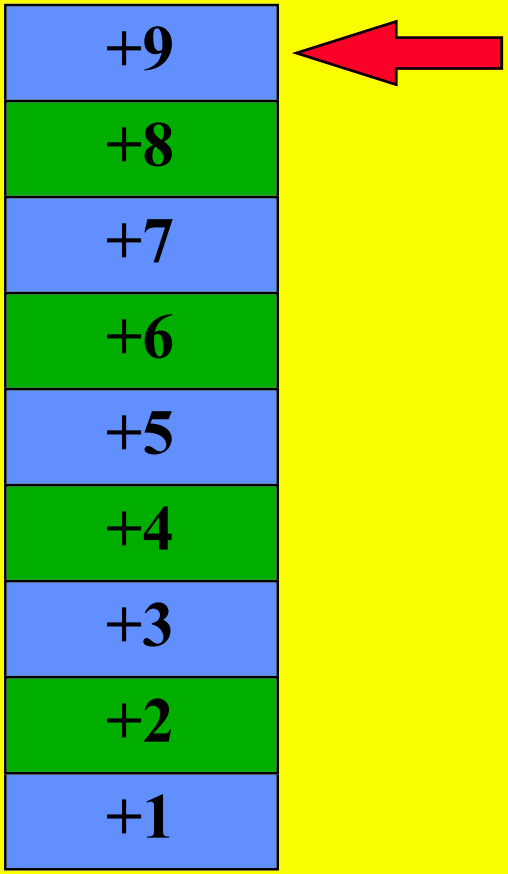
How Data is Stored.

Top of Memory

Build Direction

Bottom of Memory

Data Transfers in Stacks.

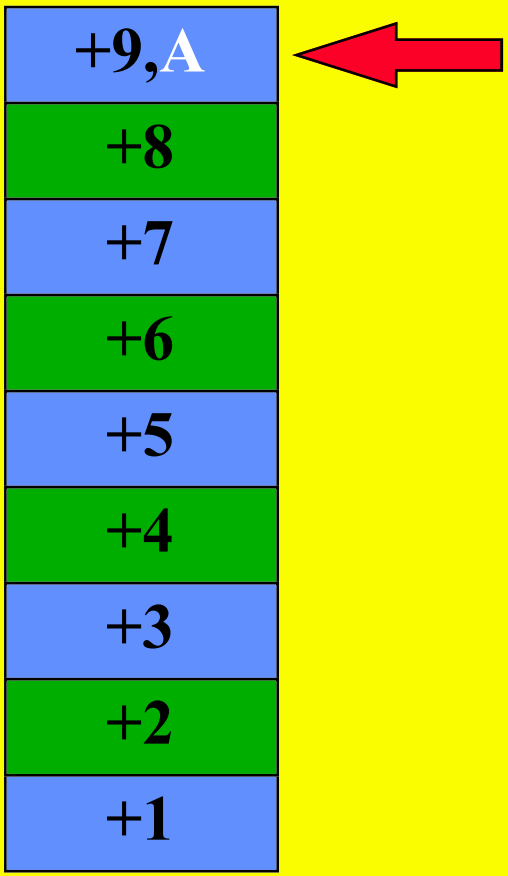


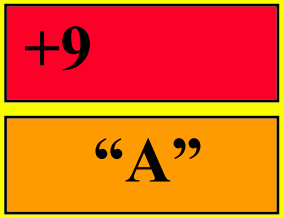
+9 Stack Pointer

Blank Stack Contents

Assume we wish to add the letter "A" to the Stack then we get :-

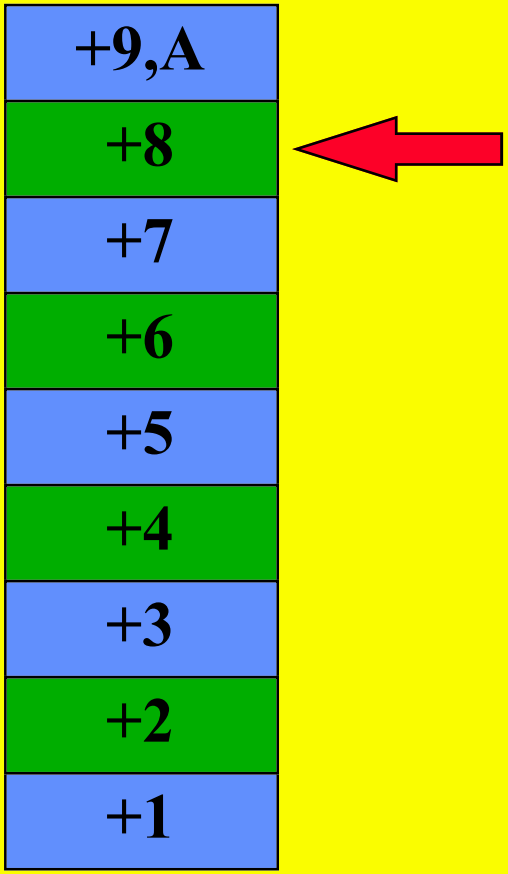
Data Transfers in Stacks.





The letter “A” placed on to the Stack next :-

Data Transfers in Stacks.

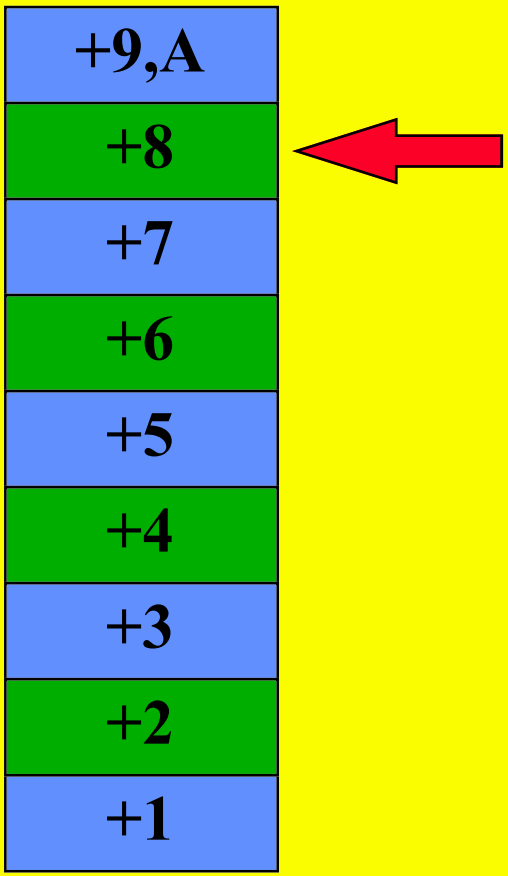


+8 Stack Pointer

Blank Stack Contents

The Pointers are adjusted ready for the next PUSH or PULL operation.

Data Transfers in Stacks.

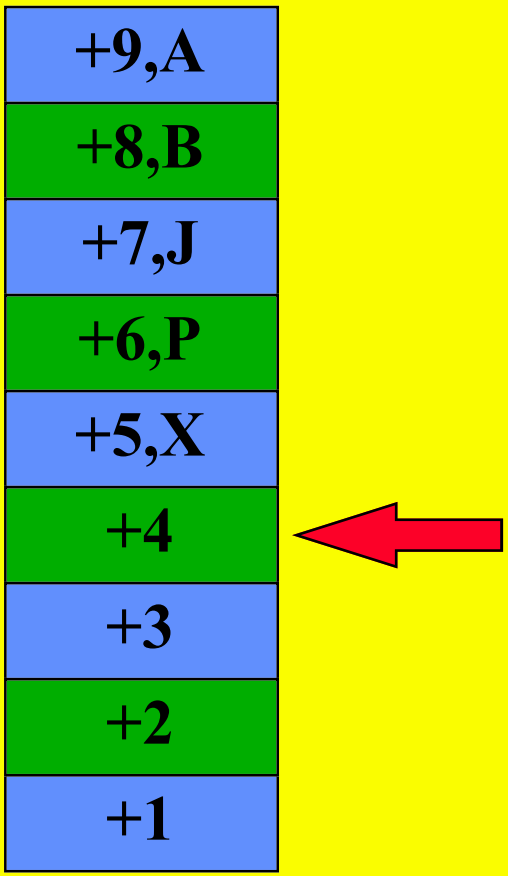


Stack Pointer

Stack Contents

Assume we wish to add more letters to the Stack then we get :-

Data Transfers in Stacks.

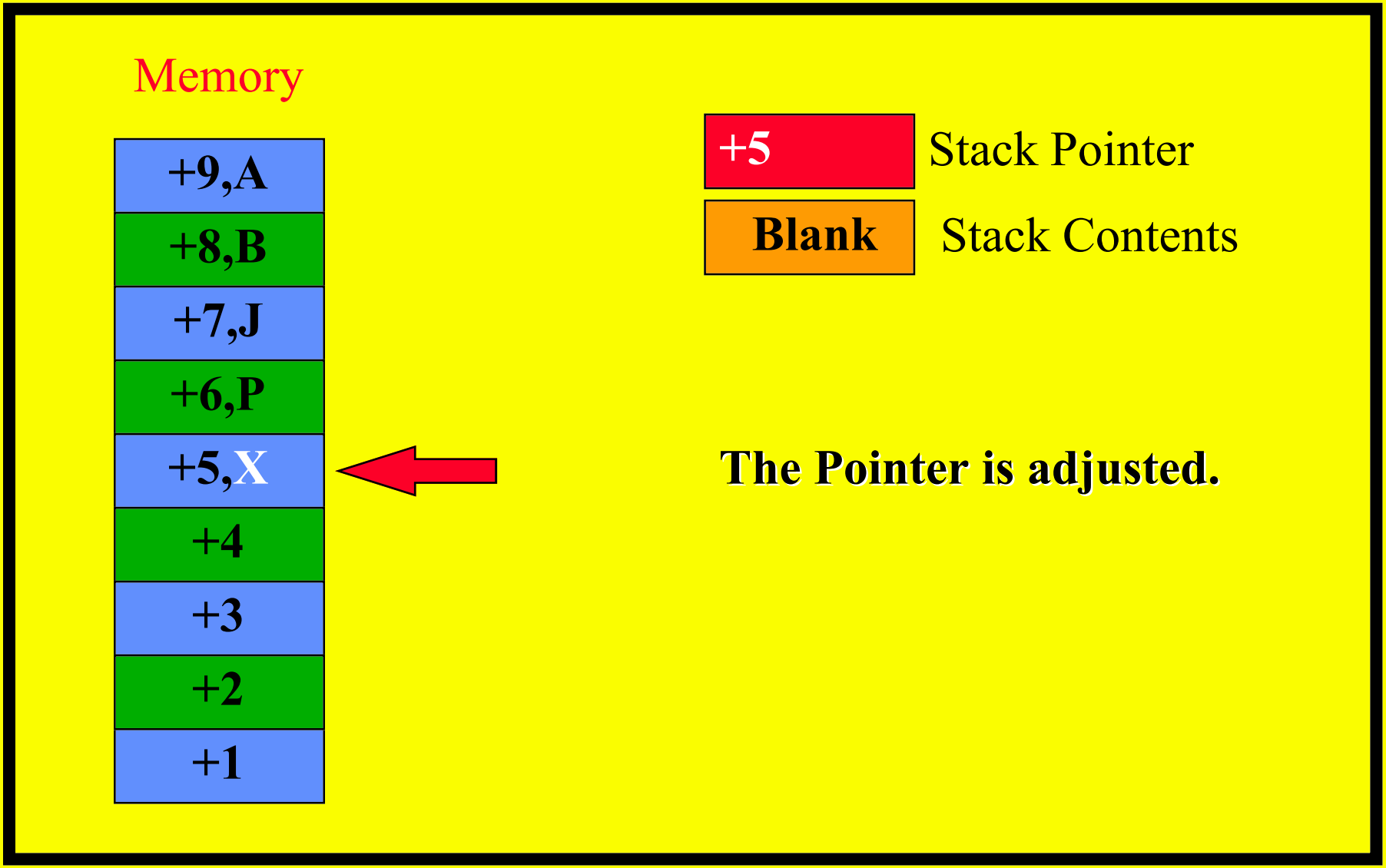


Stack Pointer

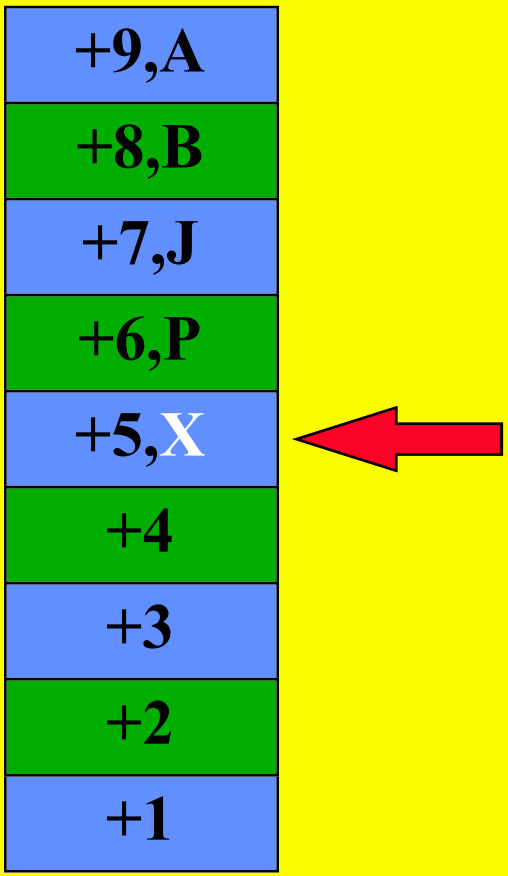
Stack Contents

Assume we wish to remove a letter from the Stack then we get :-

Data Transfers in Stacks.



Data Transfers in Stacks.



The data is extracted.

Note the actual Data still remains on the stack until it is overwritten by the next PUSH operation

System with Stack & Heap.

- A system has both Stack and Heap then

. . . .

The HEAP builds Bottom of Memory Up

and

The STACK builds Top of memory Down

The Stack

- The Stack is one of the most important structure in any computer system.
- The Stack is so important that a pointer is usually implemented in Hardware.

The Stack Terminology

- PUSH is adding information to the Stack.
- POP is removing information from the Stack.
- NOTE With the More Complex Systems you may encounter more than one stack that needs to be managed.

Data Structures

Uses of
Stacks and Heaps

Who or What uses the Stack ?

- At Machine Level.
- The Stack is used to protect contents of Hardware Variables and Registers
- The Stack is used to remember programme Subroutine Return address.

Who or What uses the Stack ?

- At System Level.
- The Stack is used to Pass Parameters.
- The Stack is used to remember programme Function and Procedure Return addresses.
- The Stack can be used Return answers from Functions.
- Used for General Variables Workspace.

What other uses for the Stack ?

- At Programme Level.
- Workspace when parsing expressions.
- Workspace for temporary results.
- You suggest !!

Who or What uses the Heap?

- At Programme Level.
- Storage of some Programme Variables.
- Temporary Data Buffers or Arrays.
- Storage where Stack is inappropriate.

Data Structures

Exercises for Stacks and Heaps

What is Infix Notation?

- Its the way we normally write down mathematical expressions. i.e. X + Y
- The Evaluation rules are :-
- From Left to Right.
- Evaluate contents of brackets first.
- Raise to a power ^ before * or /
- * and / before + or - etc.

What is Post fix Notation?

- Also called Reverse Polish Notation.
- It is a system that removes any confusion about what order expressions are processed in.
- Example.
- Infix Notation is A + B
- Post Fix Notation is A B +

What is Post fix Notation?

- More Examples
- Infix Notation is X + Y * Z
- is the same as X + (Y * Z)
- Post Fix Notation is Y Z * X +
- . . . . . .
- Infix Notation is (X + Y) * Z
- Post Fix Notation is X Y + Z *

Why use Post fix Notation?

- Example
- 6 + 4 / 2 = is the answer 5 or 8 ?
- We have to consider the evaluation rules.
- OK We performs the Division first.
- So the Answer is 8
- Post fix converts expression to 4 2/ 6+ = 8 leaving no room for interpretation.

Optional Exercise No. 3.

Design a programme to Implement a Stack Structure in an Integer Array.

Modify Programme so that a Heap can also be incorporated in same Integer Array.

REMEMBER

Stack and Heap must not overwrite each other.

Optional Exercise No. 4.

Design a programme to Implement a Stack Structure.

Use a stack to parse a data string containing the following bracket types {} [] () <> and ensure all brackets match correctly.

Example test string ([{}<{{[<>]}}>])

Note ><)(etc. are illegal parings.

Optional Exercise No. 5.

Design a programme to Implement a Stack Structure.

Use a stack to convert infix notation to Reverse polish or postfix notation.

Example test string P+Q*R^2-1

Note Two stacks required, one for operands and the other for variables.

Data Structures

Review

Covered Topics

STACKS

QUEUES

Linked Lists

Data Structures

The Basics of Queues

What is a Queue ?

- A more complex data structure that enables data to be transferred or delayed in :-
- Last In Last Out order (LILO)
- or more commonly known as
- A First In First Out order (FIFO)

Who or What uses a Queue ?

- A Queue could be used whenever Data arrives at a speed too fast for immediate programme processing.
- Typical applications :-
- Buffering Serial or Network ports.
- Message passing between programmes.
- Data flow smoothing.
- You Suggest !

How is a Queue Implemented ?

- Data is stored in a Linear structure.
- The Linear structure may be held in or on any directly accessible data store.
- Pointers are allocated to identify where the items of data are Loaded and Extracted.
- Once the data is Stored or Extracted the pointers are adjusted ready for the next data transfers.

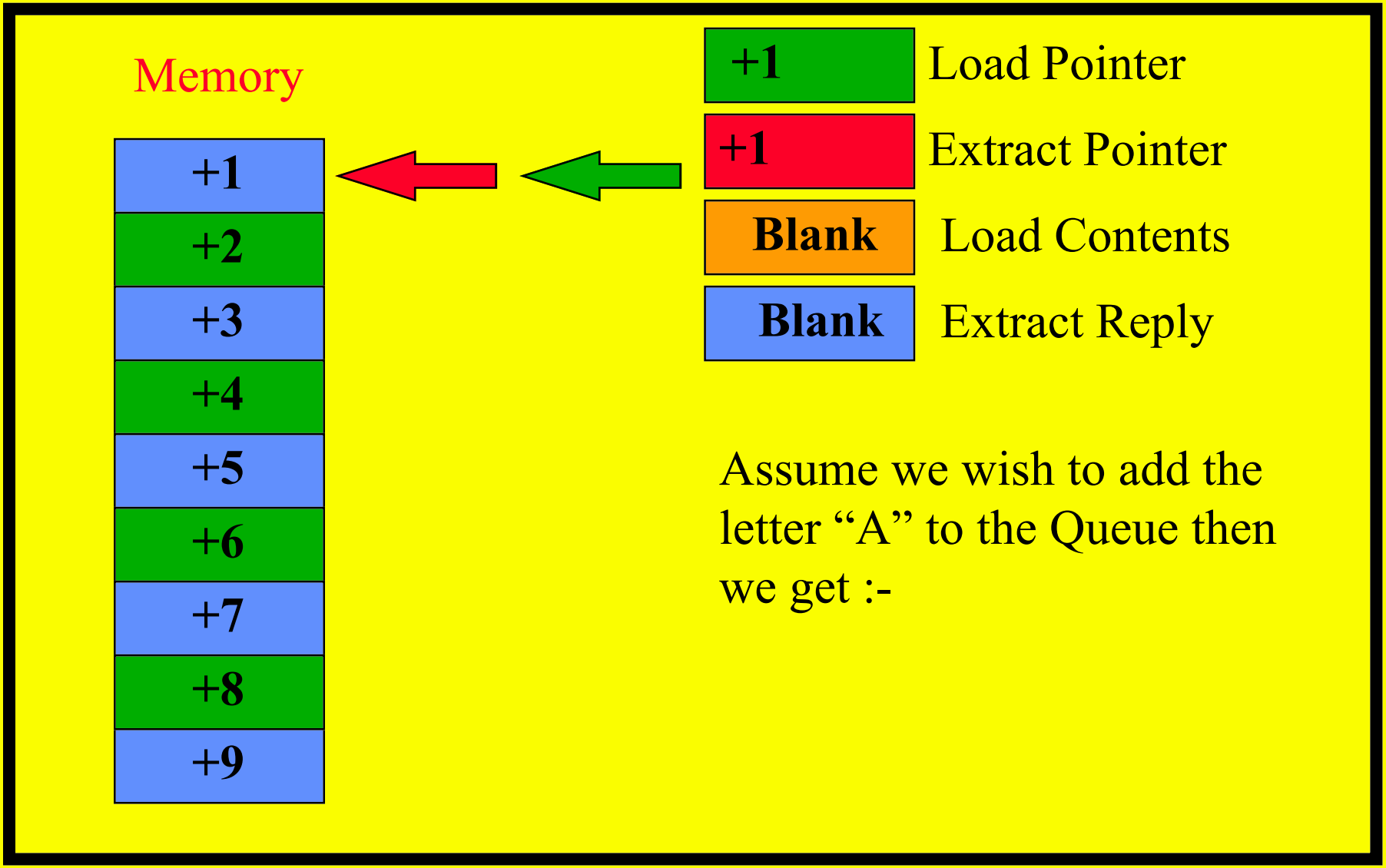
How is a Queue Implemented ?

- Additionally.
- The Queue will identify if it is Full.
- The Queue will identify if it is Empty.
- The Queue MUST be able to cope with worst case demand.

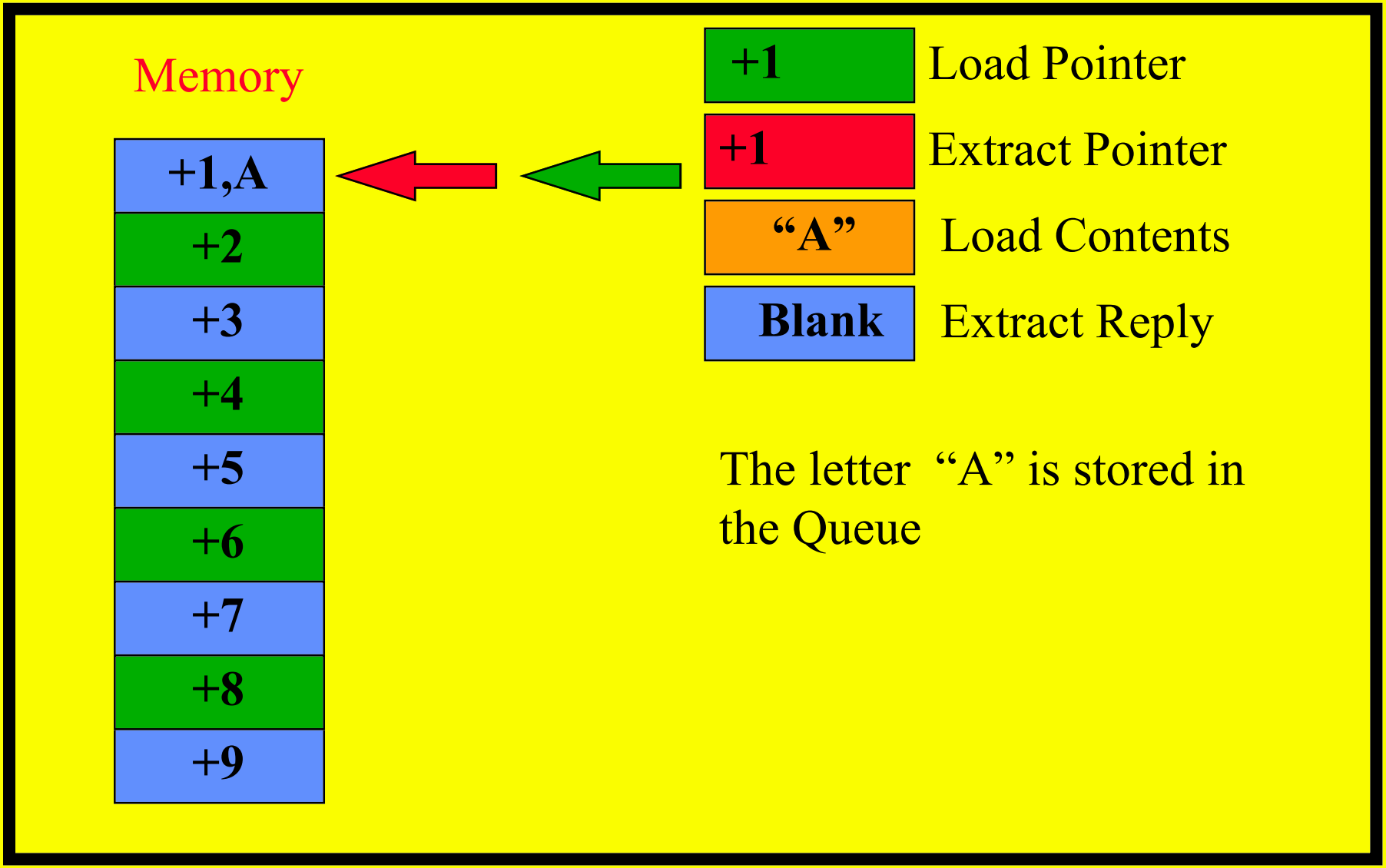
How is a Queue Implemented ?

- To construct a Queue will will need as a minimum the following structures :-
- A linear direct access data store.
- A pointer indicating where the next load will occur.
- A pointer indicating where the next unload will occur.

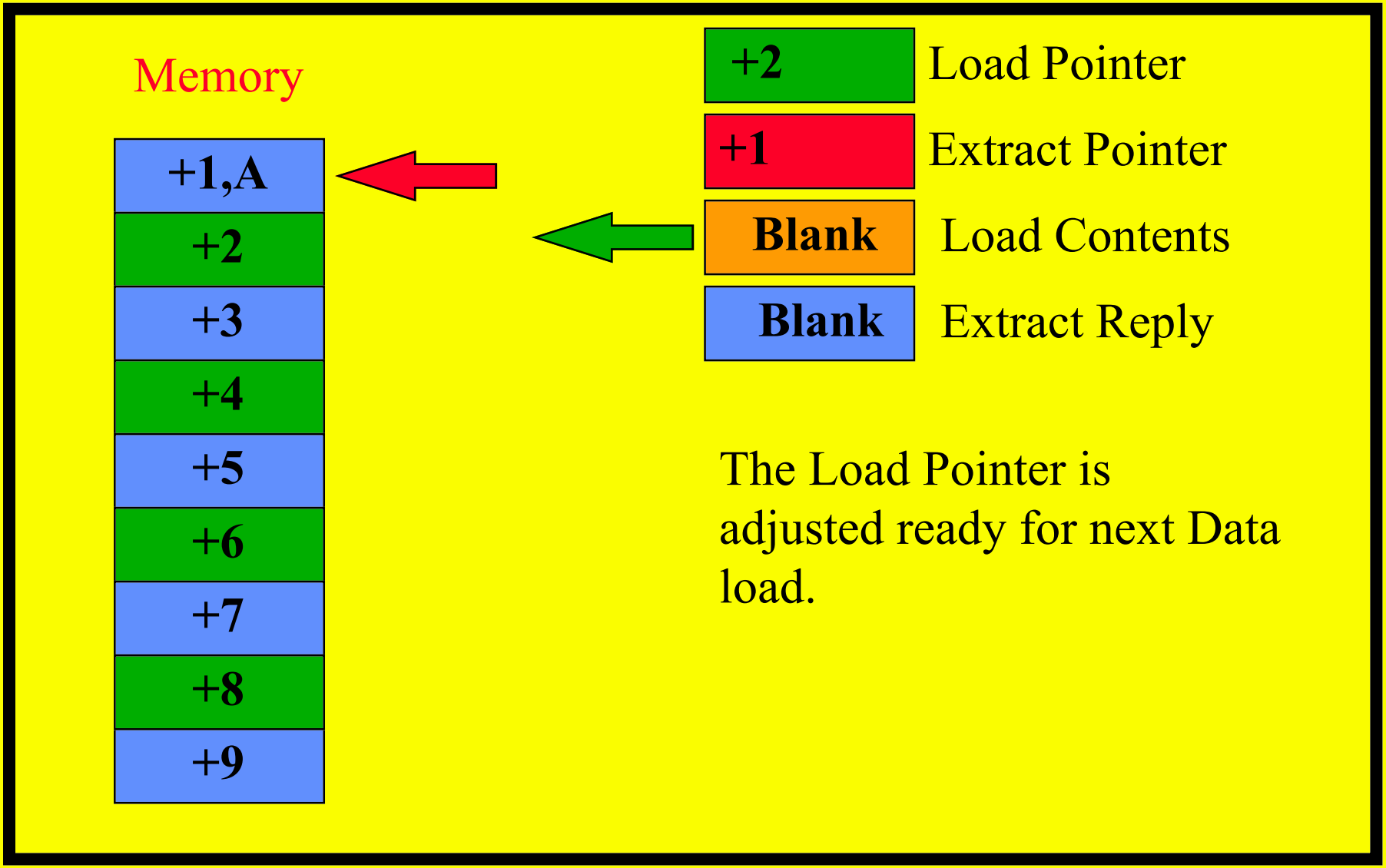
Data Transfer in Queues.



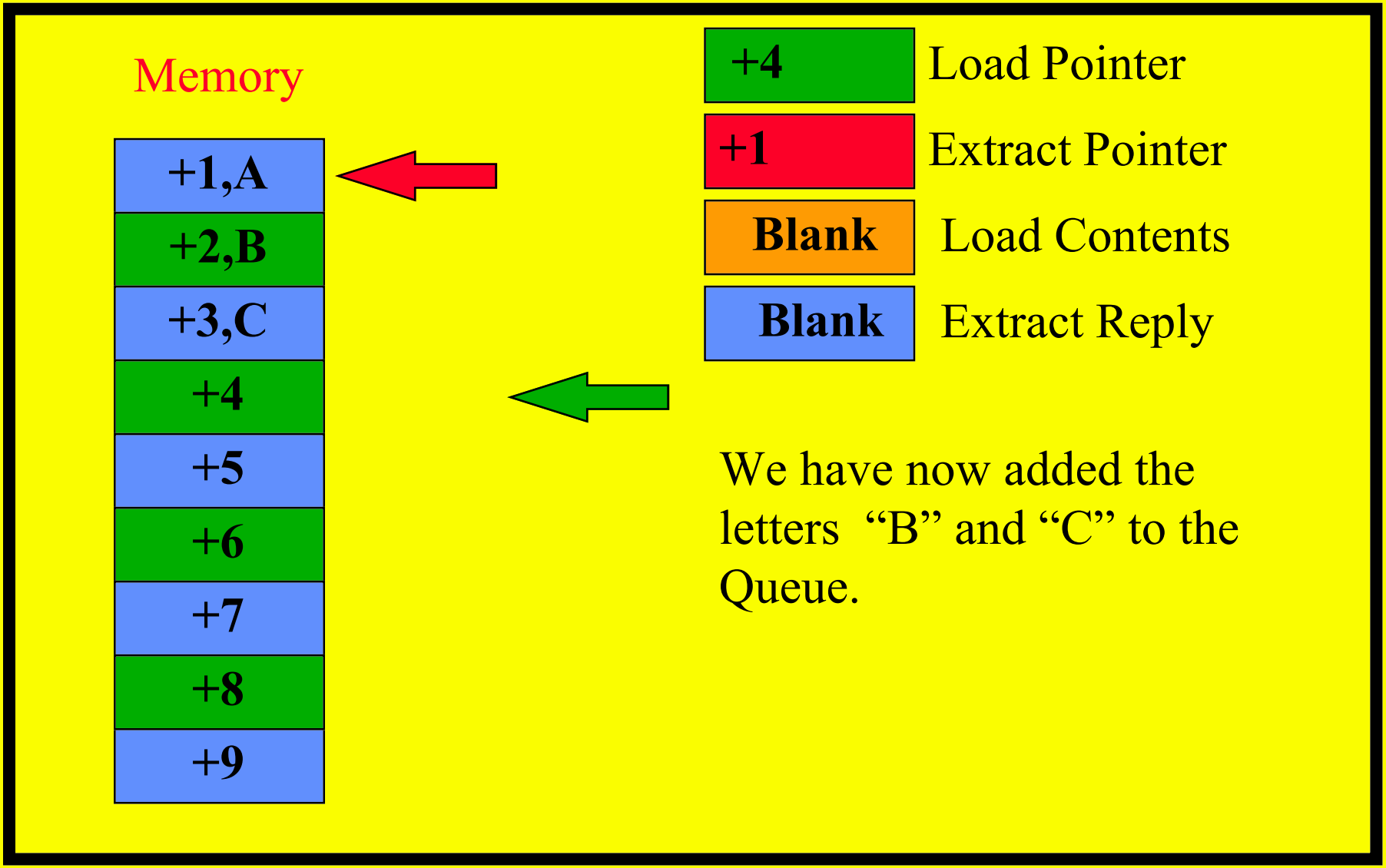
Data Transfer in Queues.



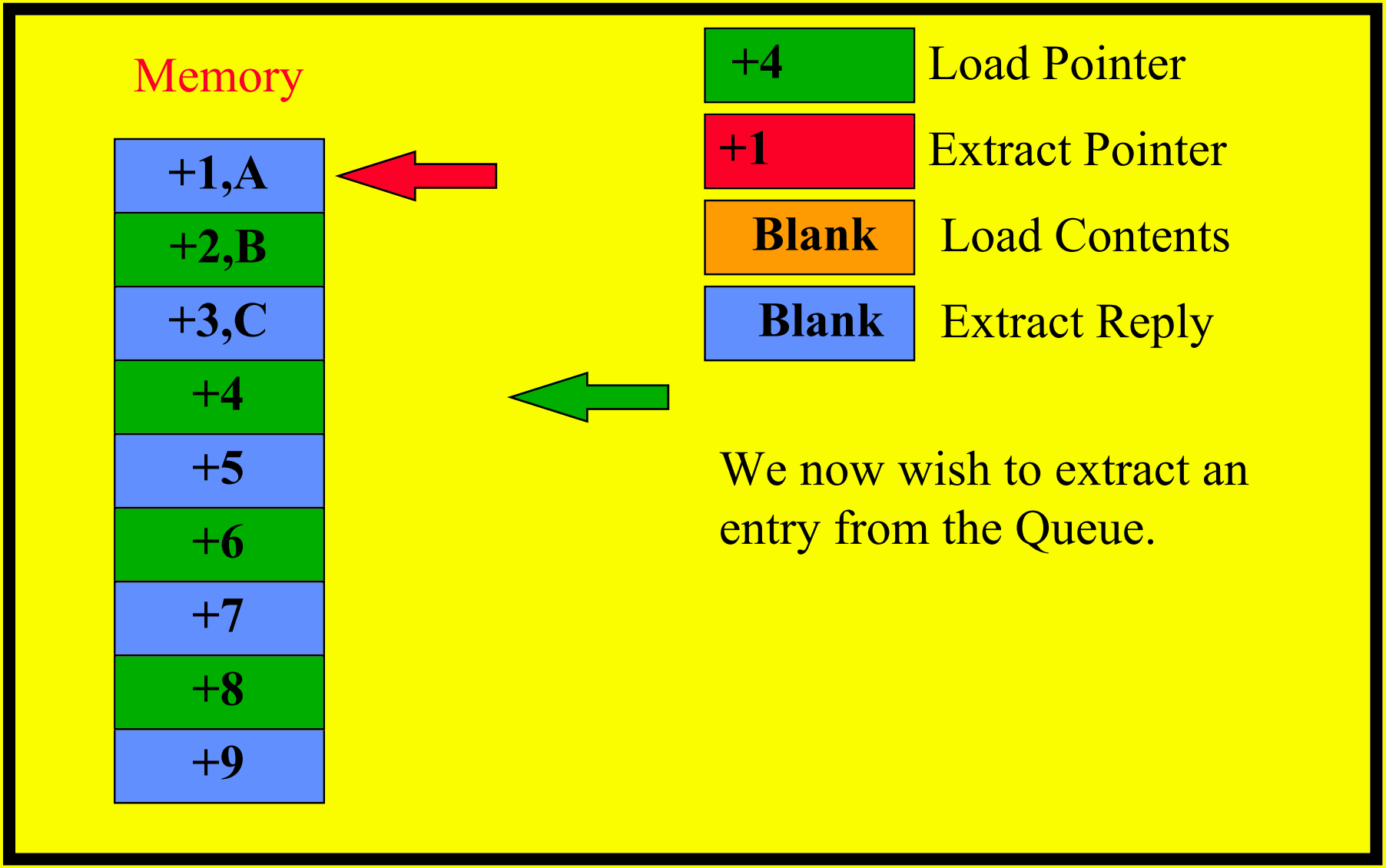
Data Transfer in Queues.



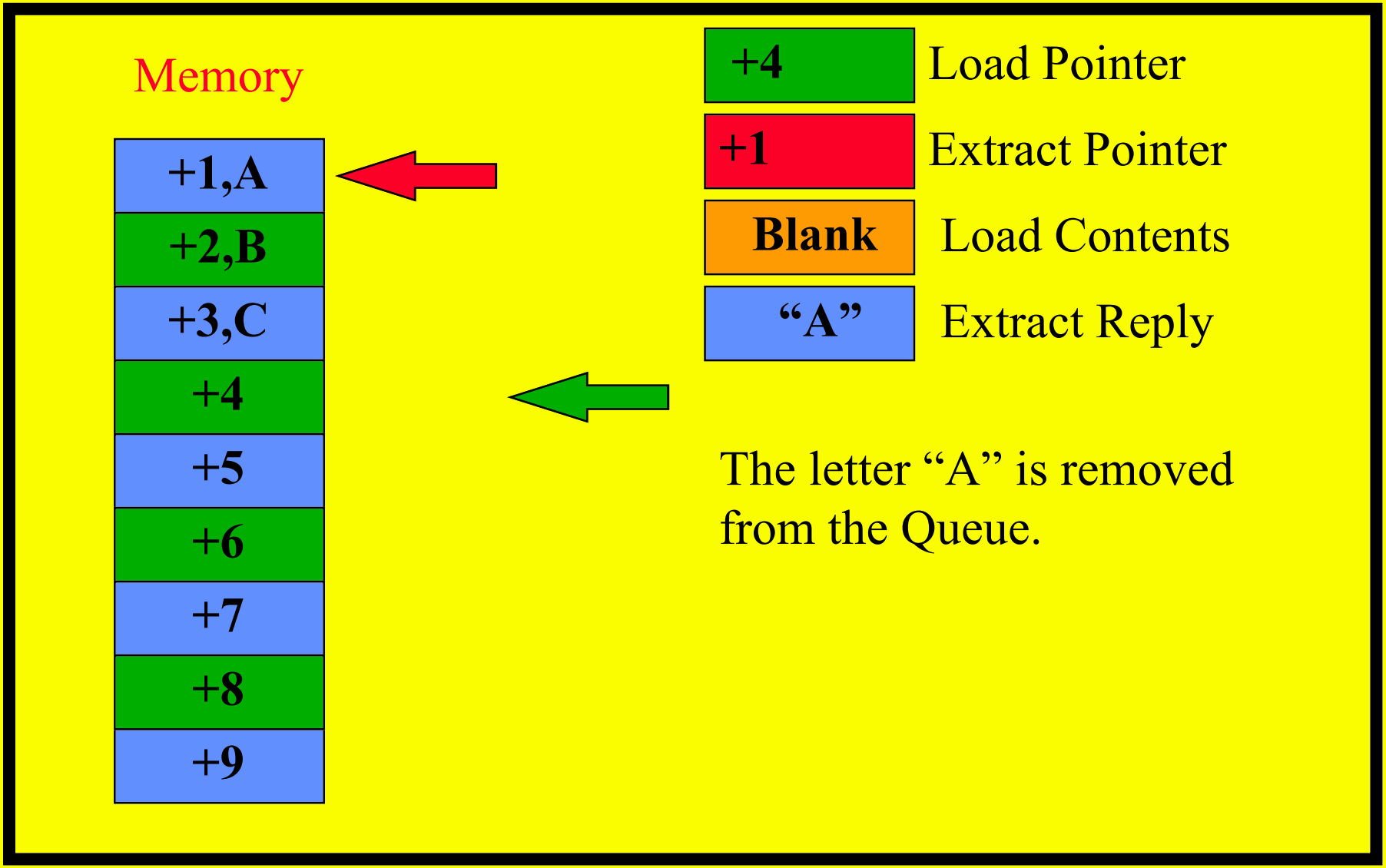
Data Transfer in Queues.



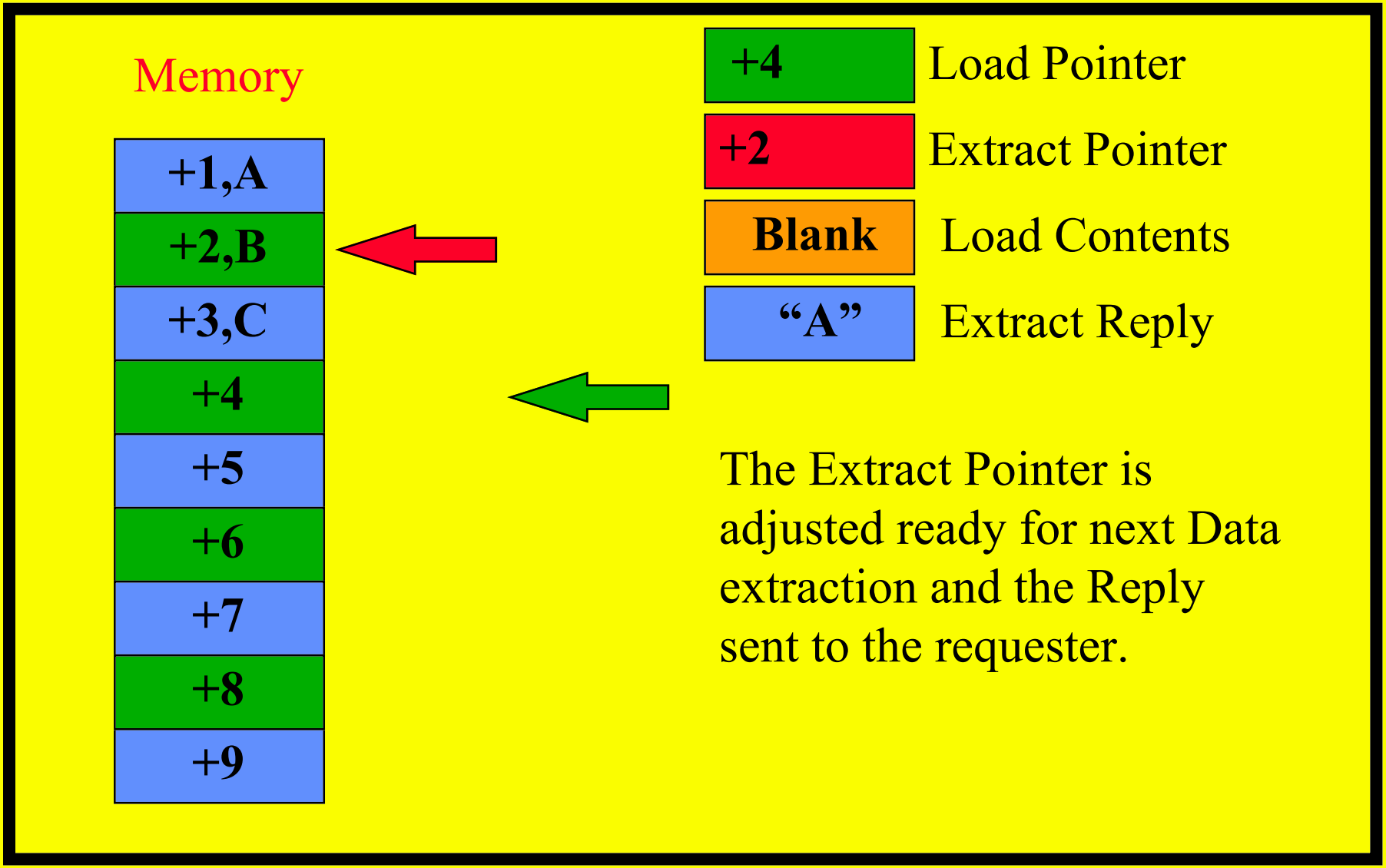
Data Transfer in Queues.



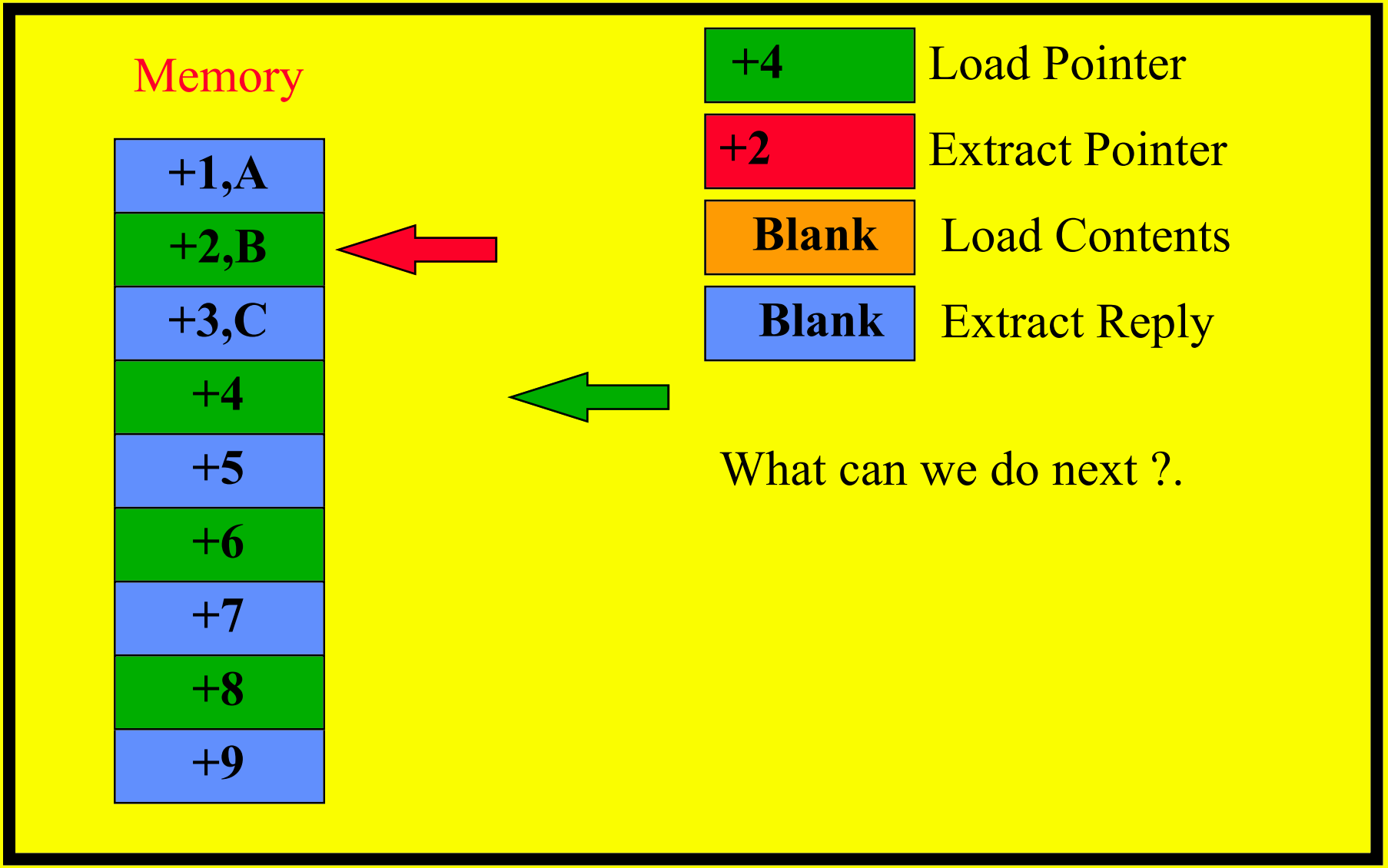
Data Transfer in Queues.



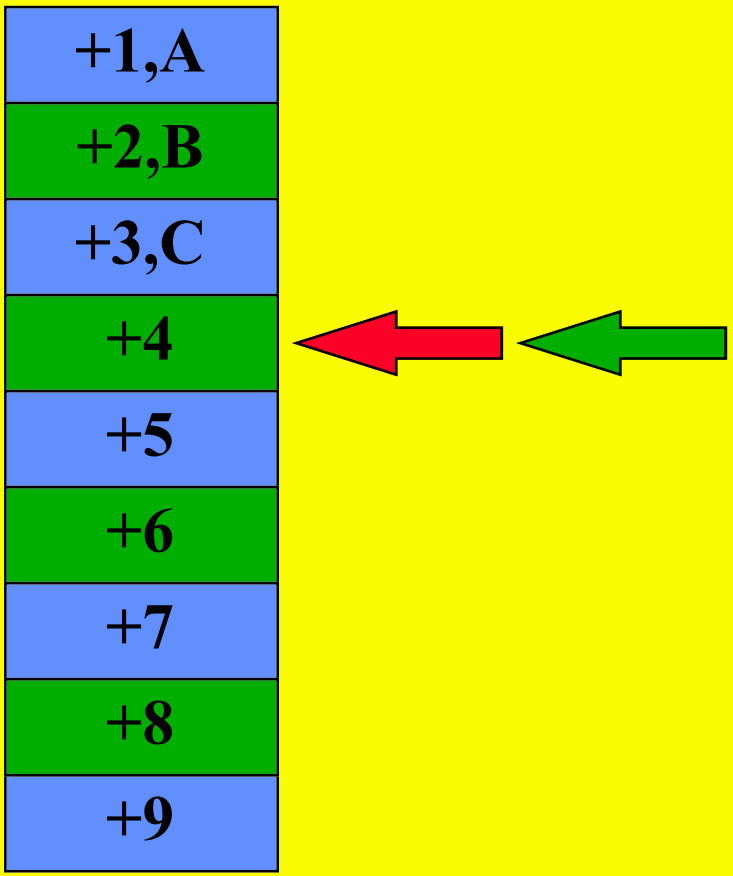
Data Transfer in Queues.

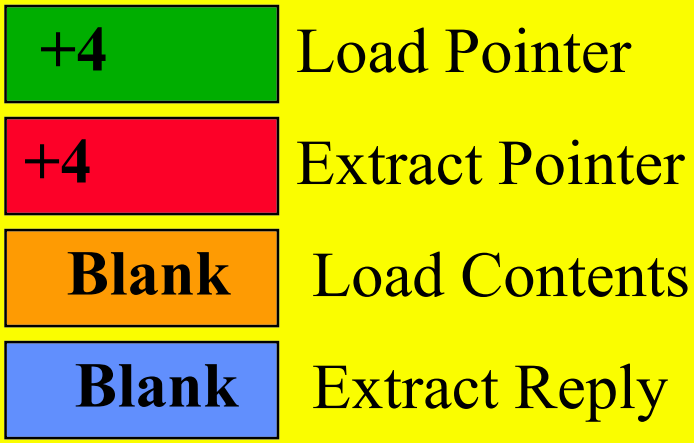


Data Transfer in Queues.



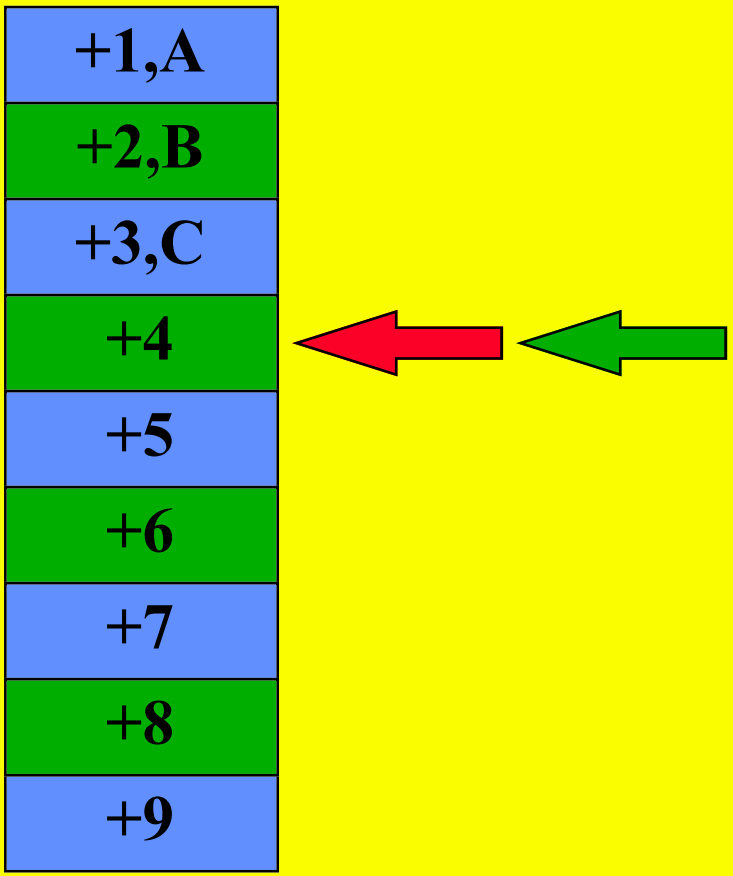
Data Transfer in Queues.

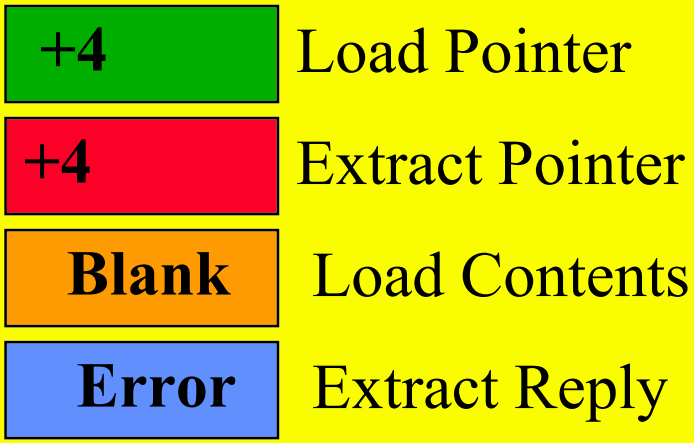




We are now at the state where two more Extract requests have been performed. What happens if another extract is requested from the Queue ?

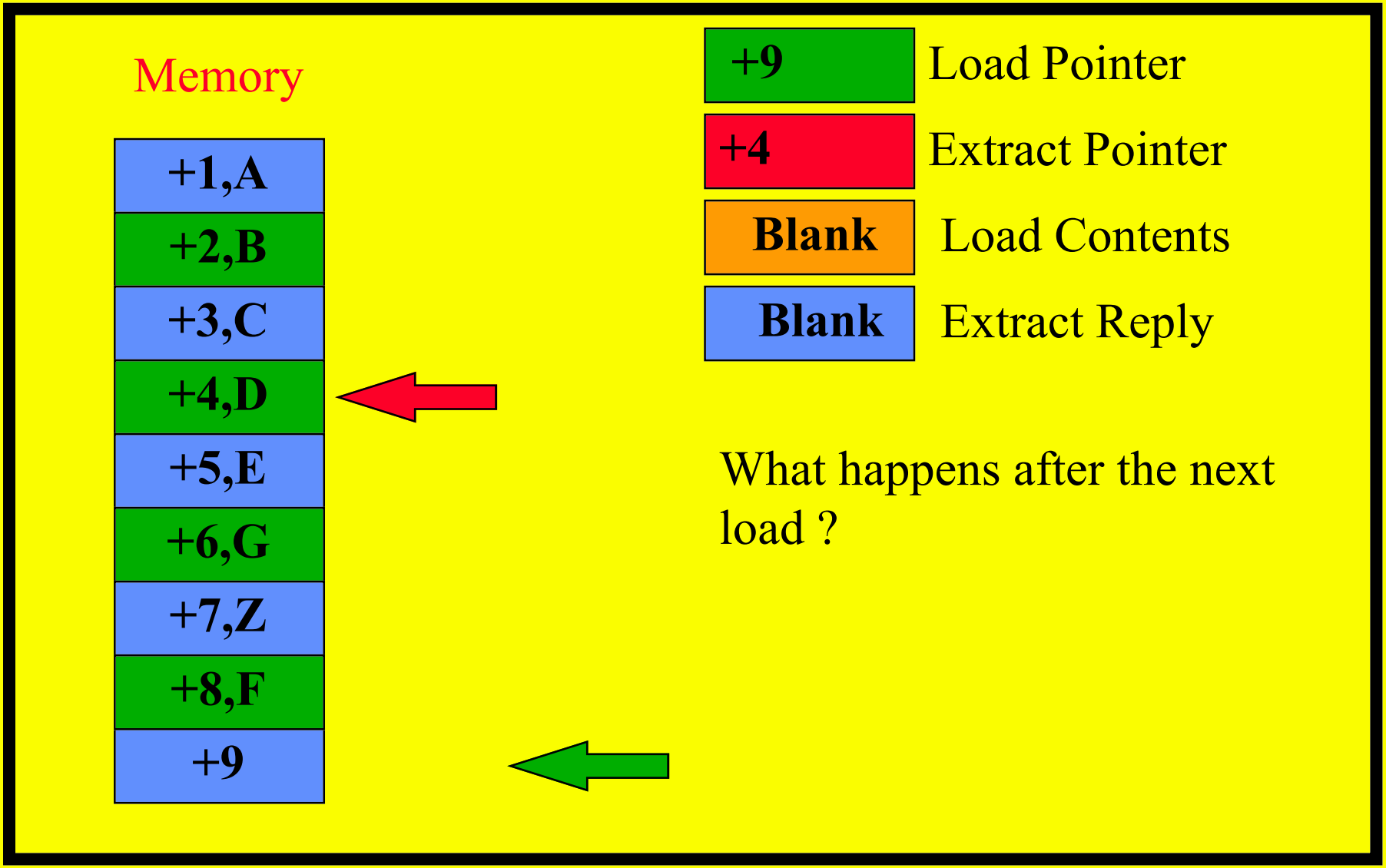
Data Transfer in Queues.



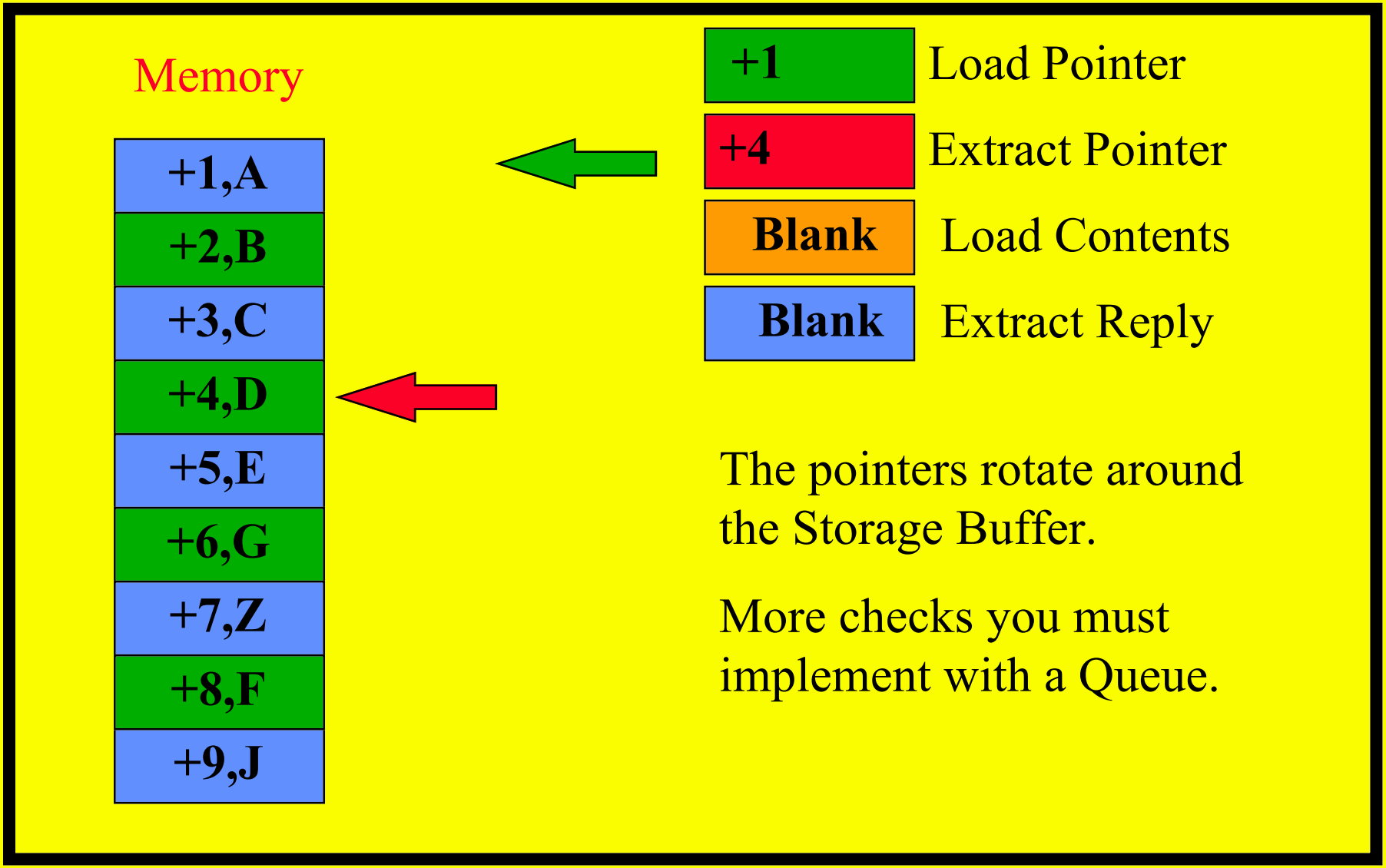


We return an **ERROR** message as the two pointers are at the same address in the Queue. This is the Queue Empty ERROR.

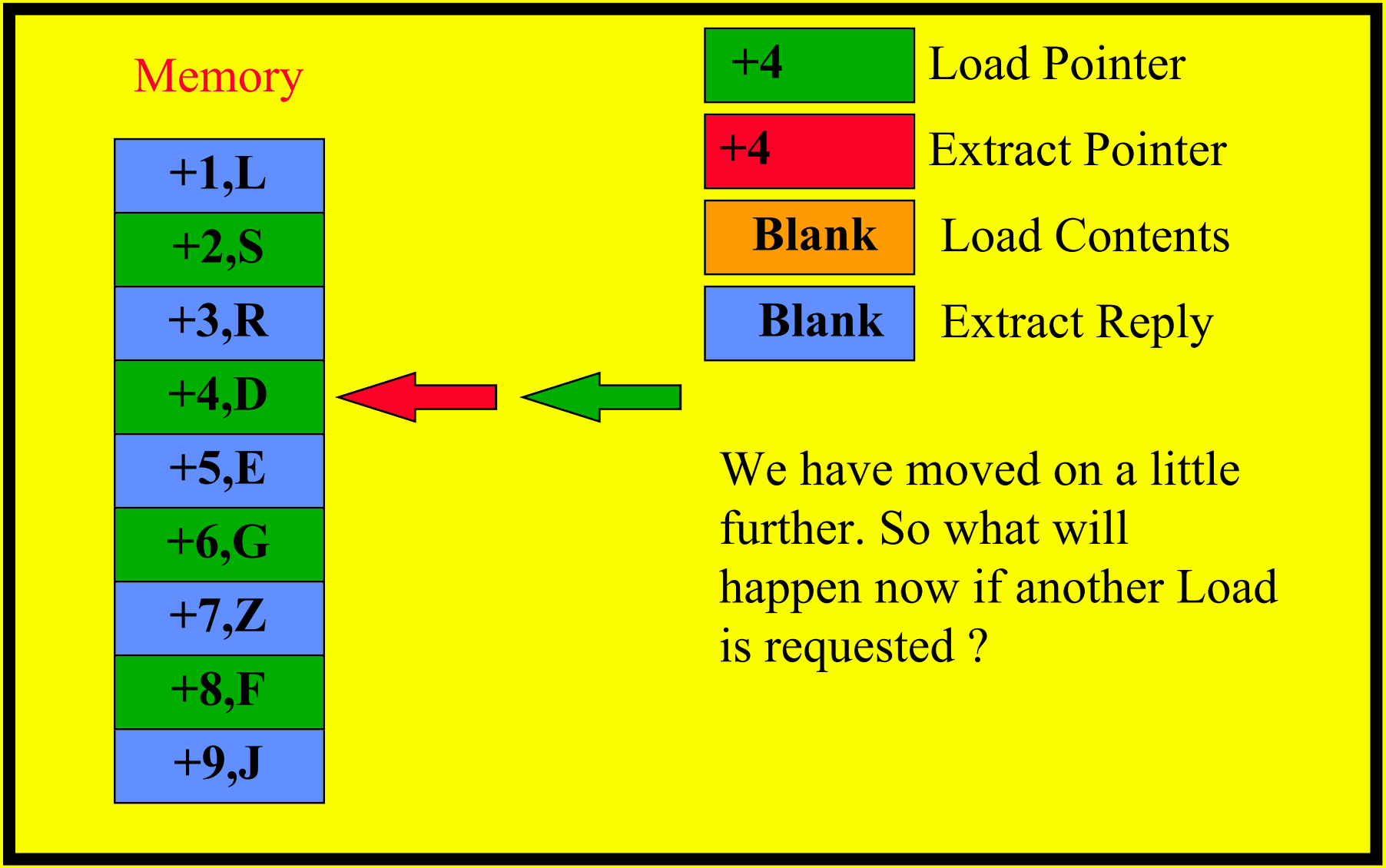
Data Transfer in Queues.



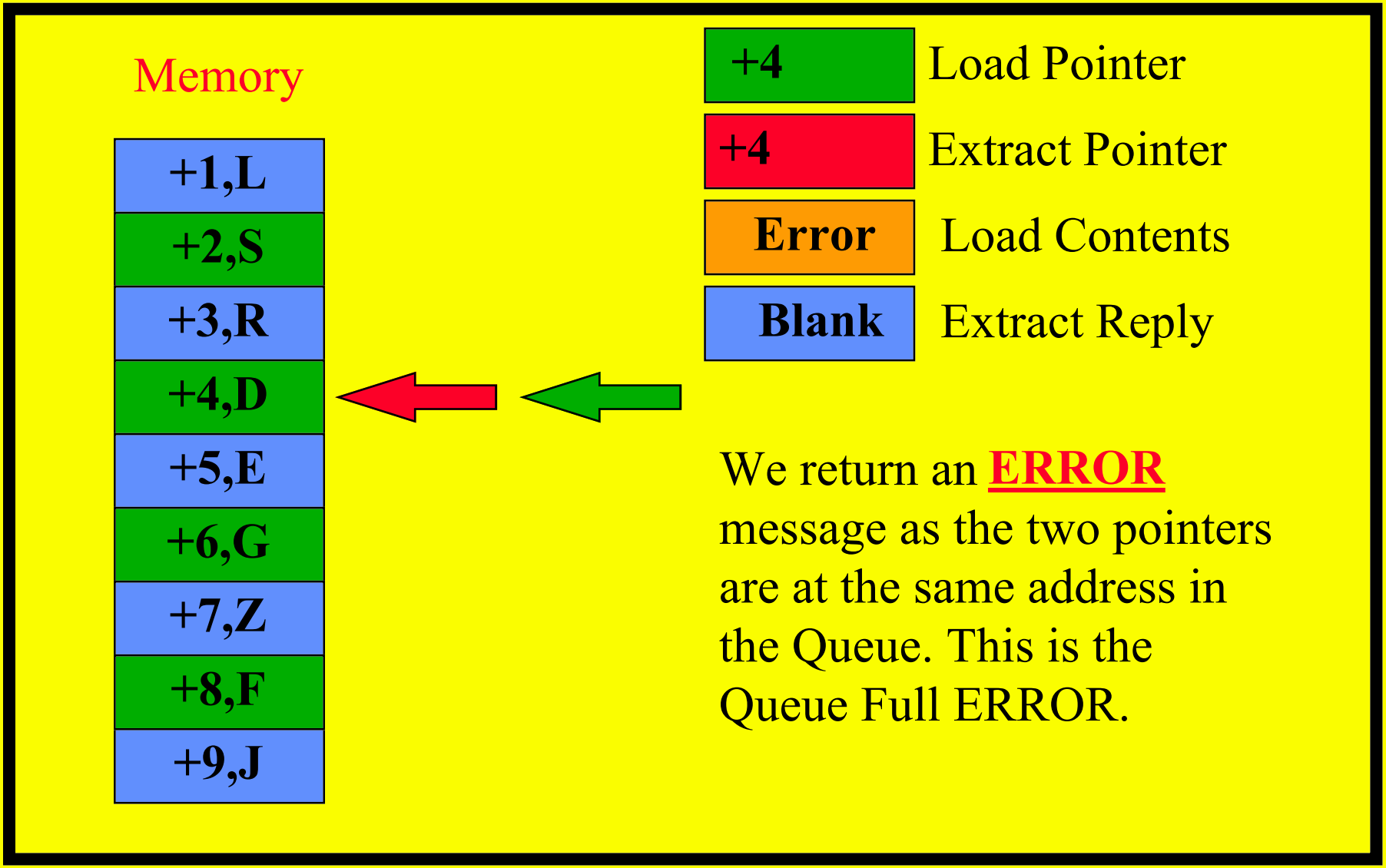
Data Transfer in Queues.



Data Transfer in Queues.



Data Transfer in Queues.



How is a Queue Implemented ? Tricks of the Trade.

- Make the length of the linear direct access Data store an exact binary multiple. i.e. 16, 32, 64 etc. units long.
- With Binary multiple lengths an Offset pointer can be masked with the (Length-1) value after incrementing the pointer and hence you cannot over run the Data store.
- You then Just add the offset to the the Data Store access pointer.

How is a Queue Implemented ? Tricks of the Trade.

- Introduce an extra variable into the Queue manager which holds a count of the number of character in the Queue,
- If extract mode and Characters = 0 then ERROR EMPTY.
- If Load and Characters = Buffer size then ERROR FULL.
- Much easier to programme.

How is a Queue Implemented ? Tricks of the Trade.

- Write a common routine.
- Keep Queue control parameters say at start of the Queue.
- Pass Queue Address as a Parameter.
- Then once you have tested and got your programme working well you then have something to rely on.

Data Structures

Exercises for Queues

Optional Exercise No. 6.

Design a programme to Implement a Queue Structure in an Integer Array.

REMEMBER ERROR CHECKING

Check that there is room free in the Queue before trying to Add more Data.

Likewise that Data exists in Queue before trying to remove the Data.

Data Structures

Review

Covered Topics

STACKS

QUEUES

Linked Lists

Data Structures

The Basics of Linked Lists

What is a Linked List ?

- A much more complex data structure that enables data to be connected in a specific order :-
- The List consists of two Parts :-
- (1) The link pointer section.
- This may also consist of two parts :-
- A Forward Pointer.
- Also an Optional Back Pointer.
- (2) The associated Linked Data.

How is a Linked List Implemented ?

- Data is stored in a Linear structure.
- The Linear structure may be held in or on any directly accessible data store.
- Pointers are allocated to chain each item of data into a specific order.
- Each List has a Start "NODE".
- Definition "A NODE is a Point of Interest."

The Basic Linked List.

How the Basic Information Entity is Structured.

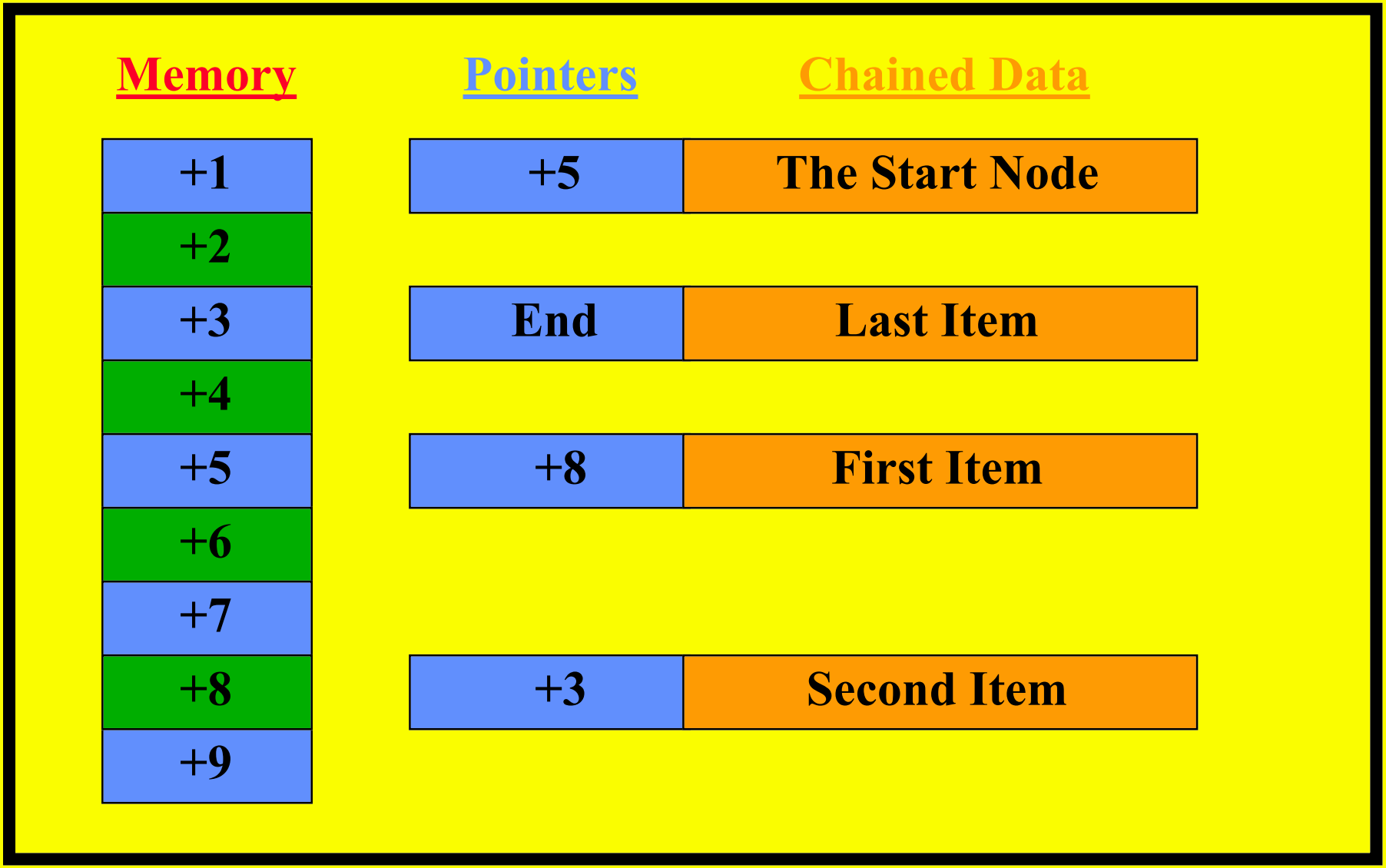
- **Each Entities Consists of :-**
- **The Information or Data Store Section.**
- **The Pointers Section.**
- **This consists of a Forward Pointer.**

The Basic Linked List.

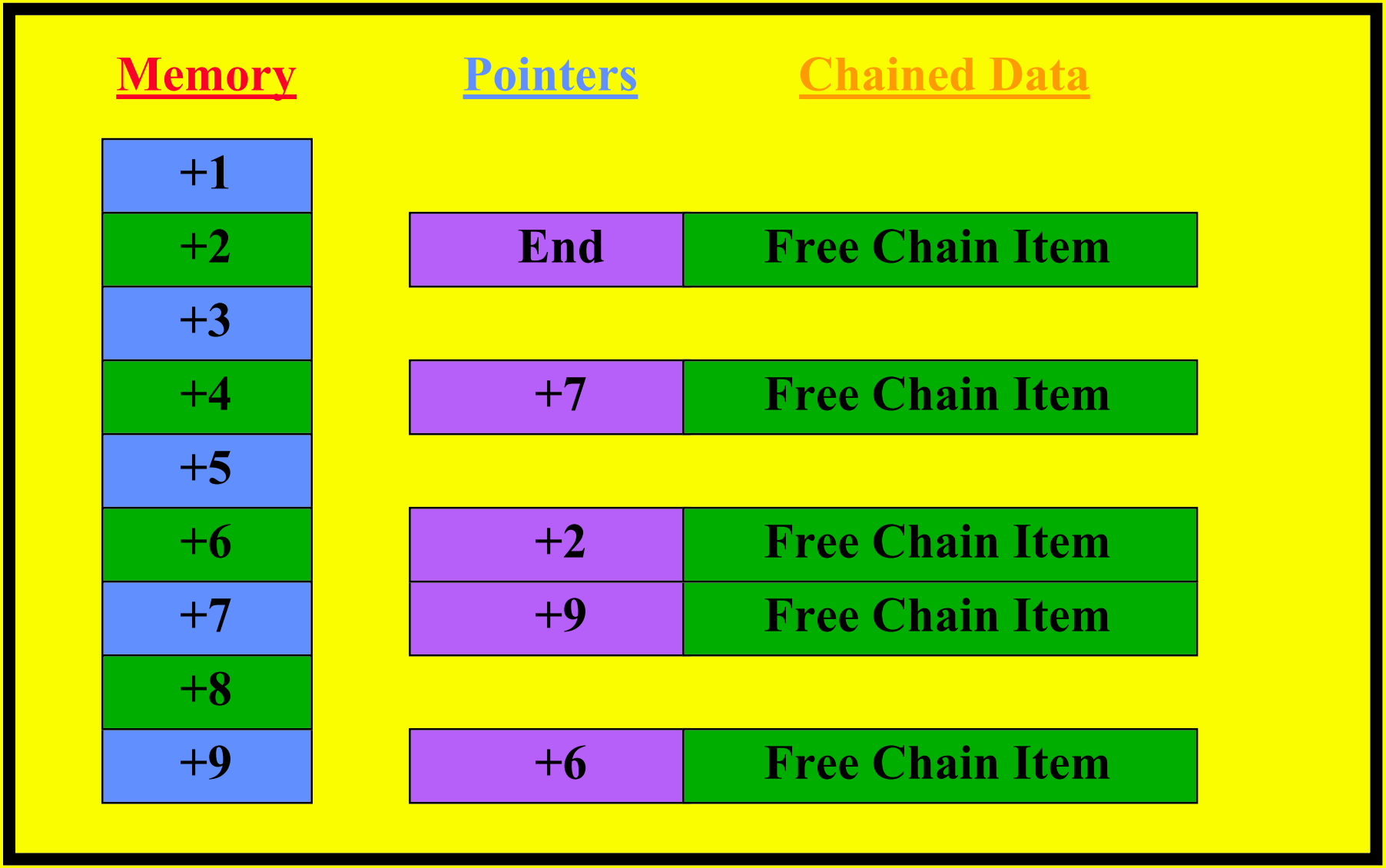
How the Basic Information Entity is Structured.

- Each Entities Consists of :-
- The Information or Data Store Section.
- The Pointers Section.
- This consists of a Forward Pointer.

The Basic Linked List.



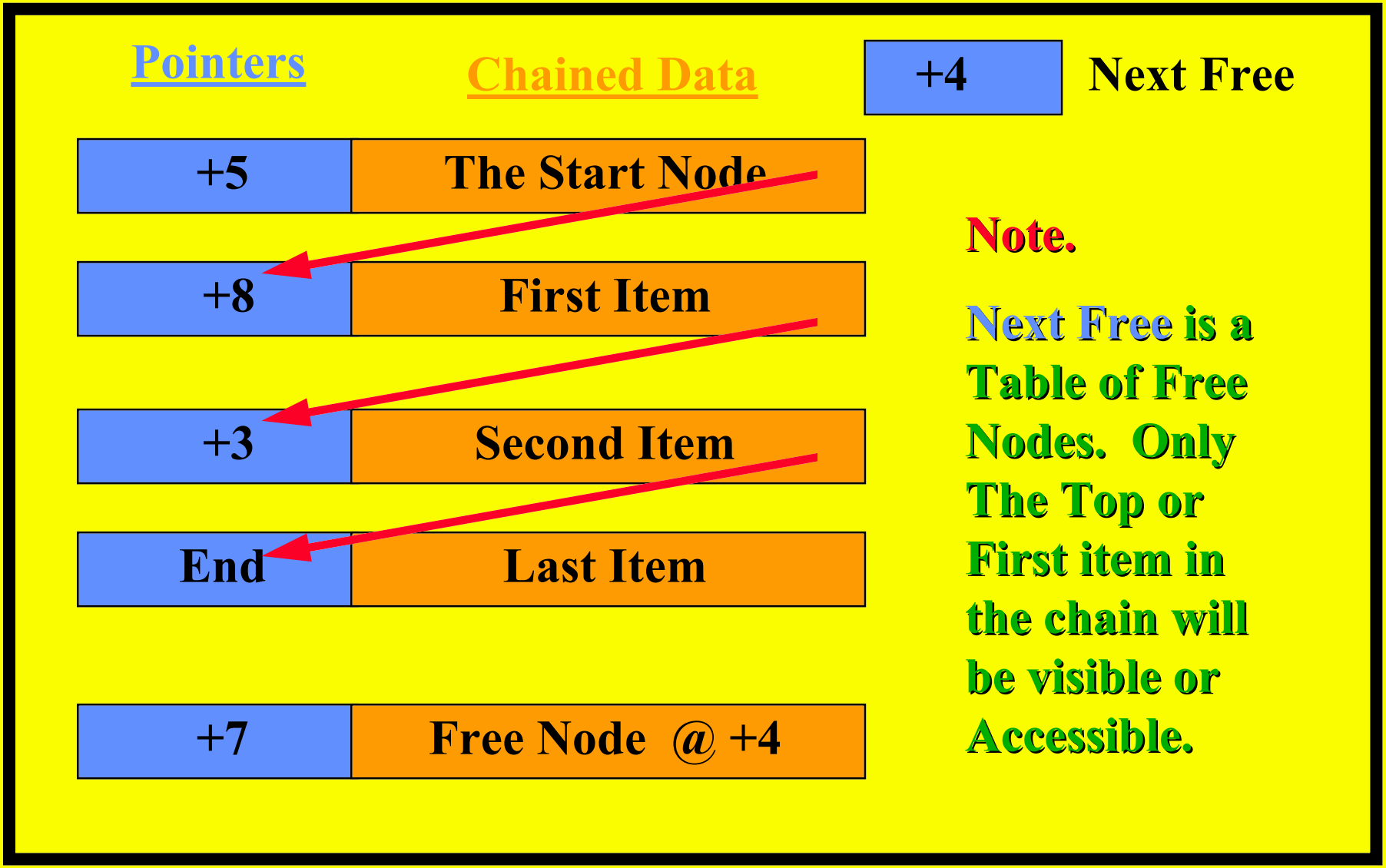
The Basic Linked List.



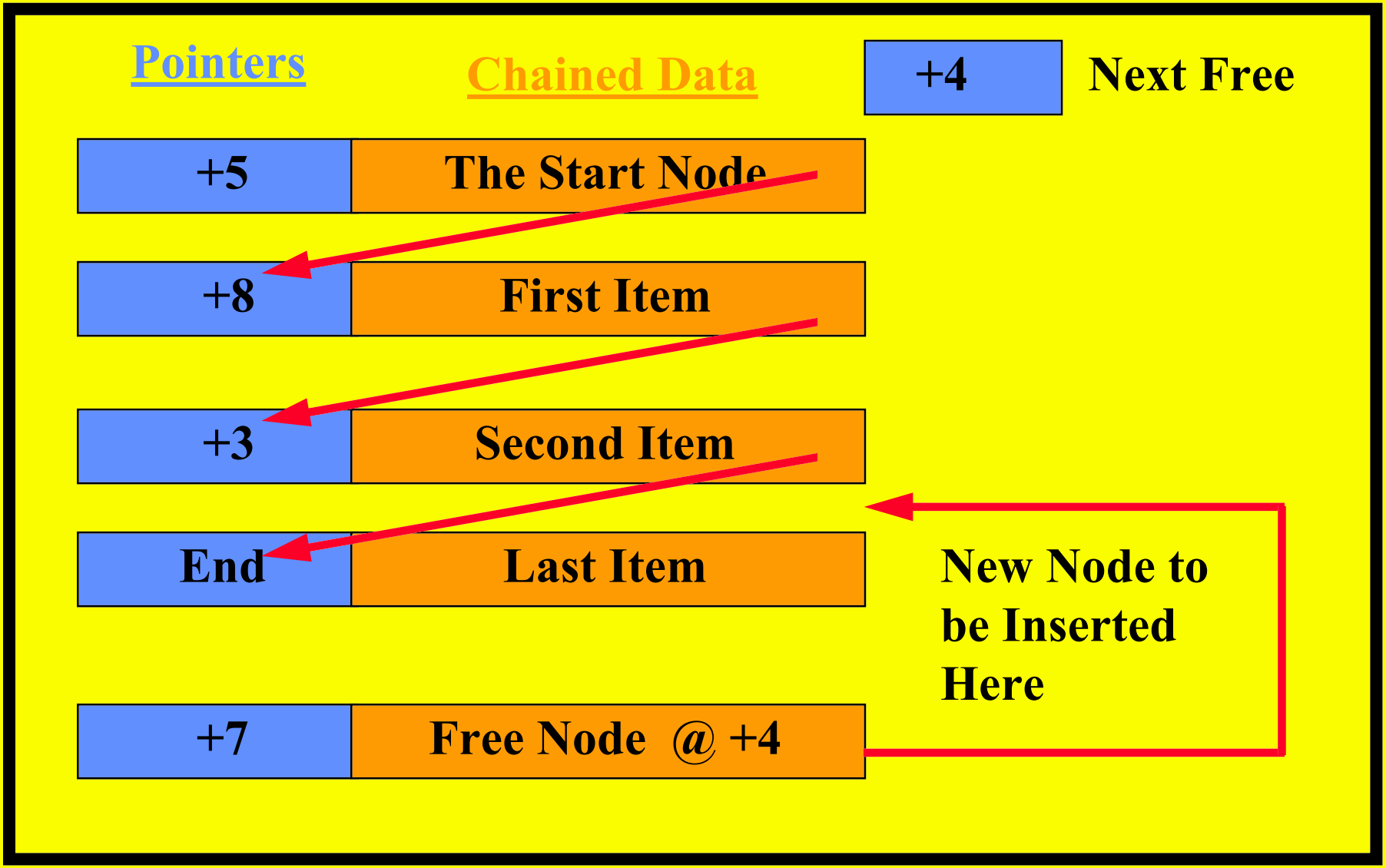
The Basic Linked List.

Memory	Pointers	Chained Data
+1	+5	The Start Node
+2	End	Free Chain Item
+3	End	Last Item
+4	+7	Free Chain Item
+5	+8	First Item
+6	+2	Free Chain Item
+7	+9	Free Chain Item
+8	+3	Second Item
+9	+6	Free Chain Item

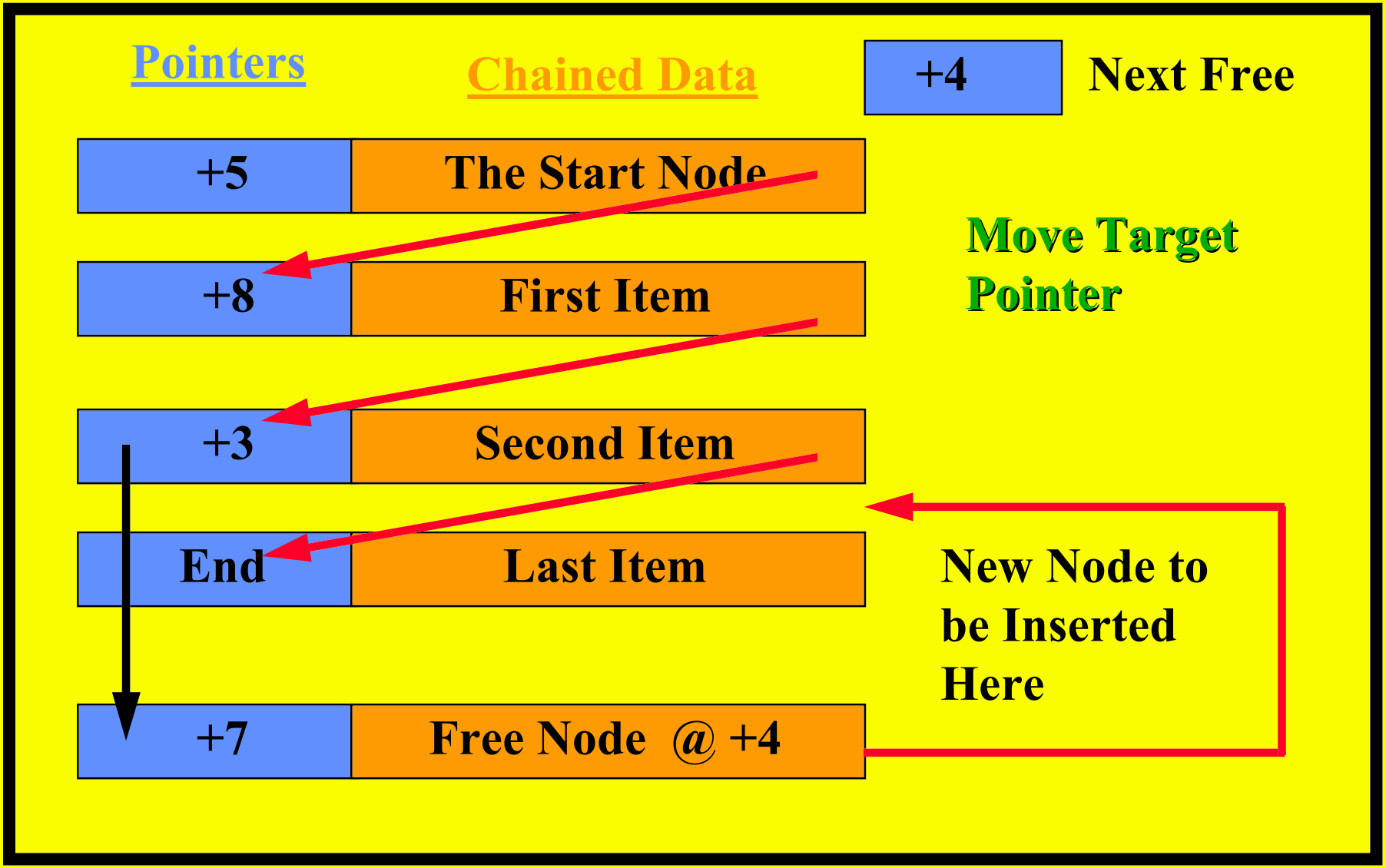
How the List Chained.



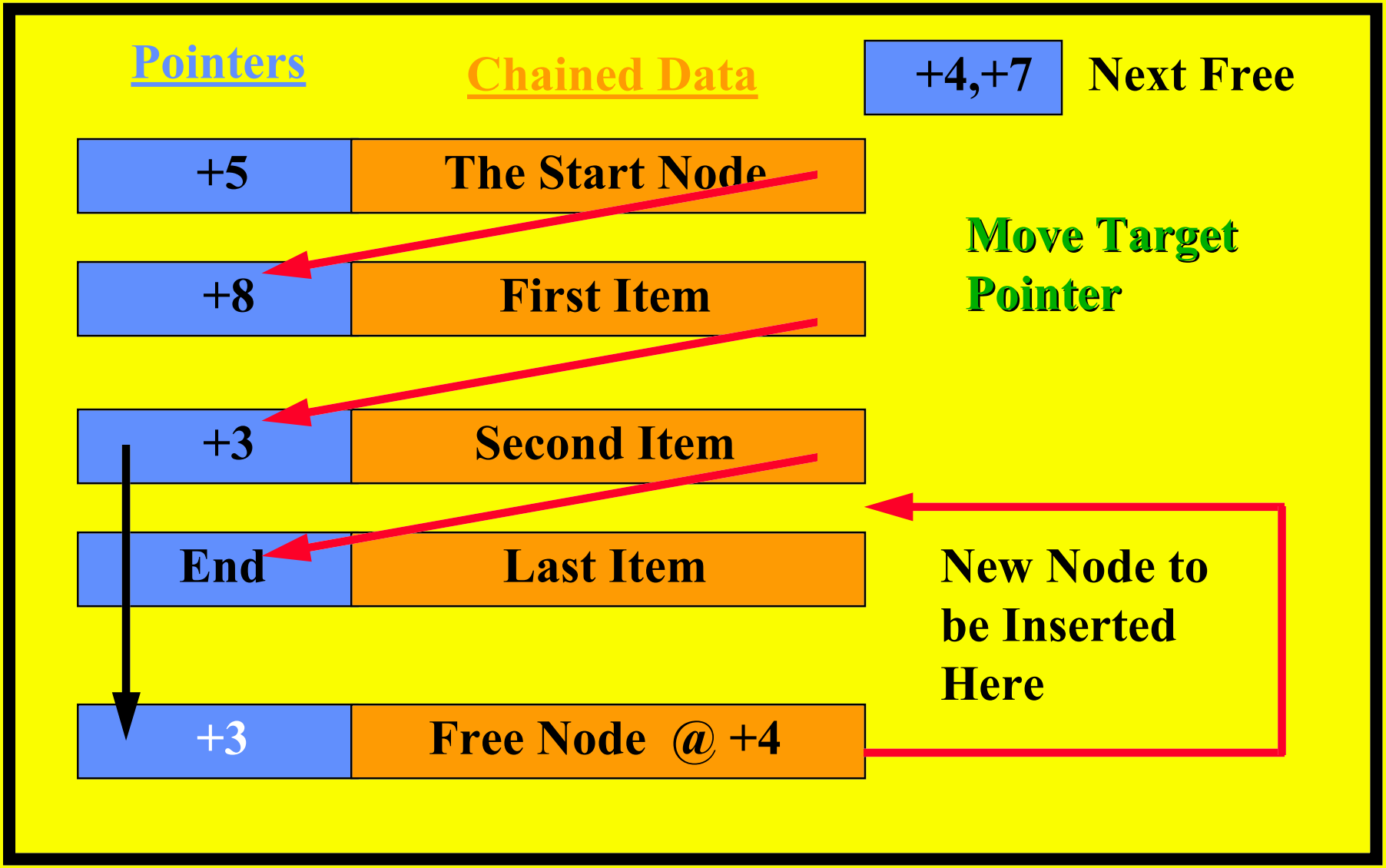
How Item is Inserted.



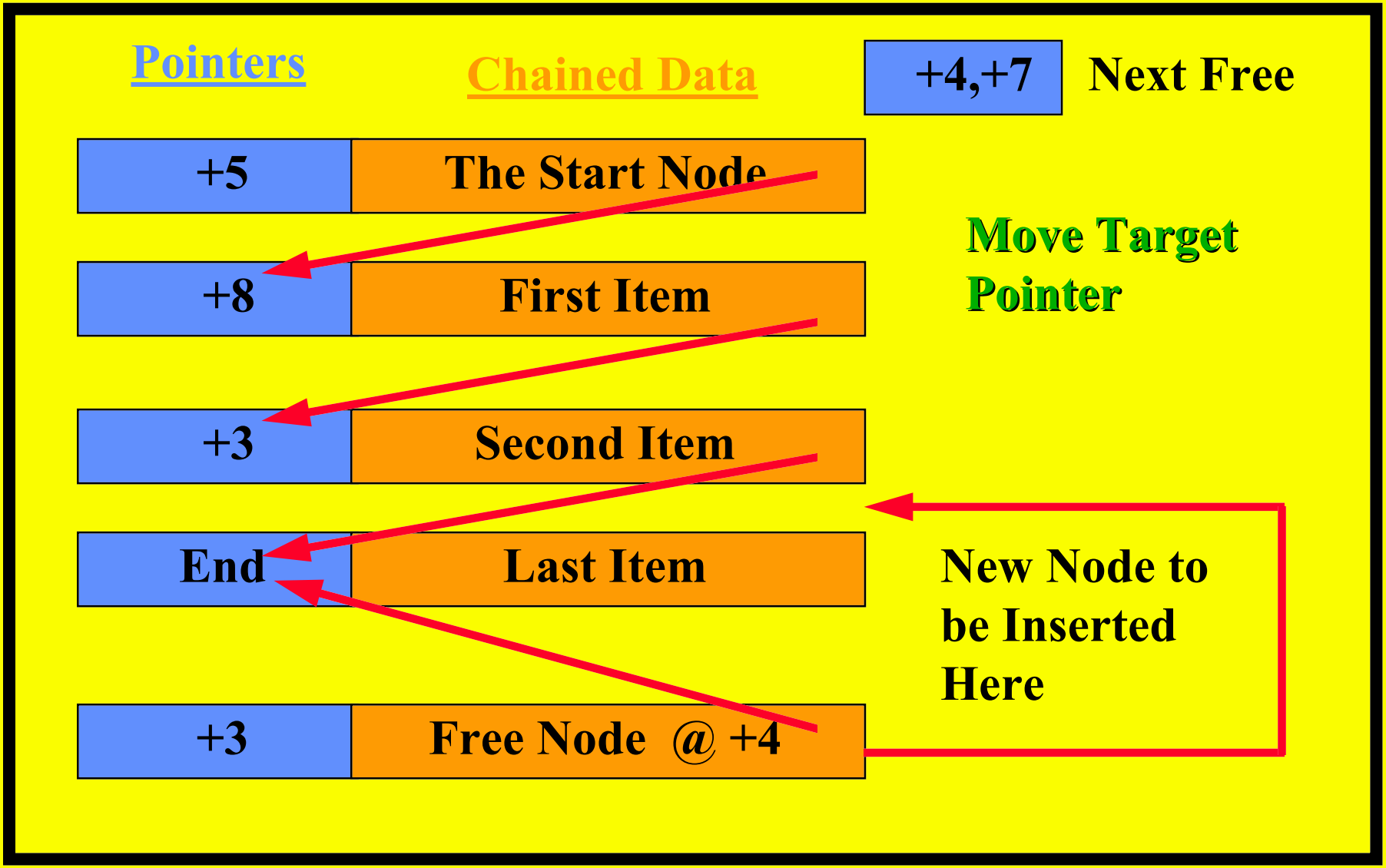
How Item is Inserted.



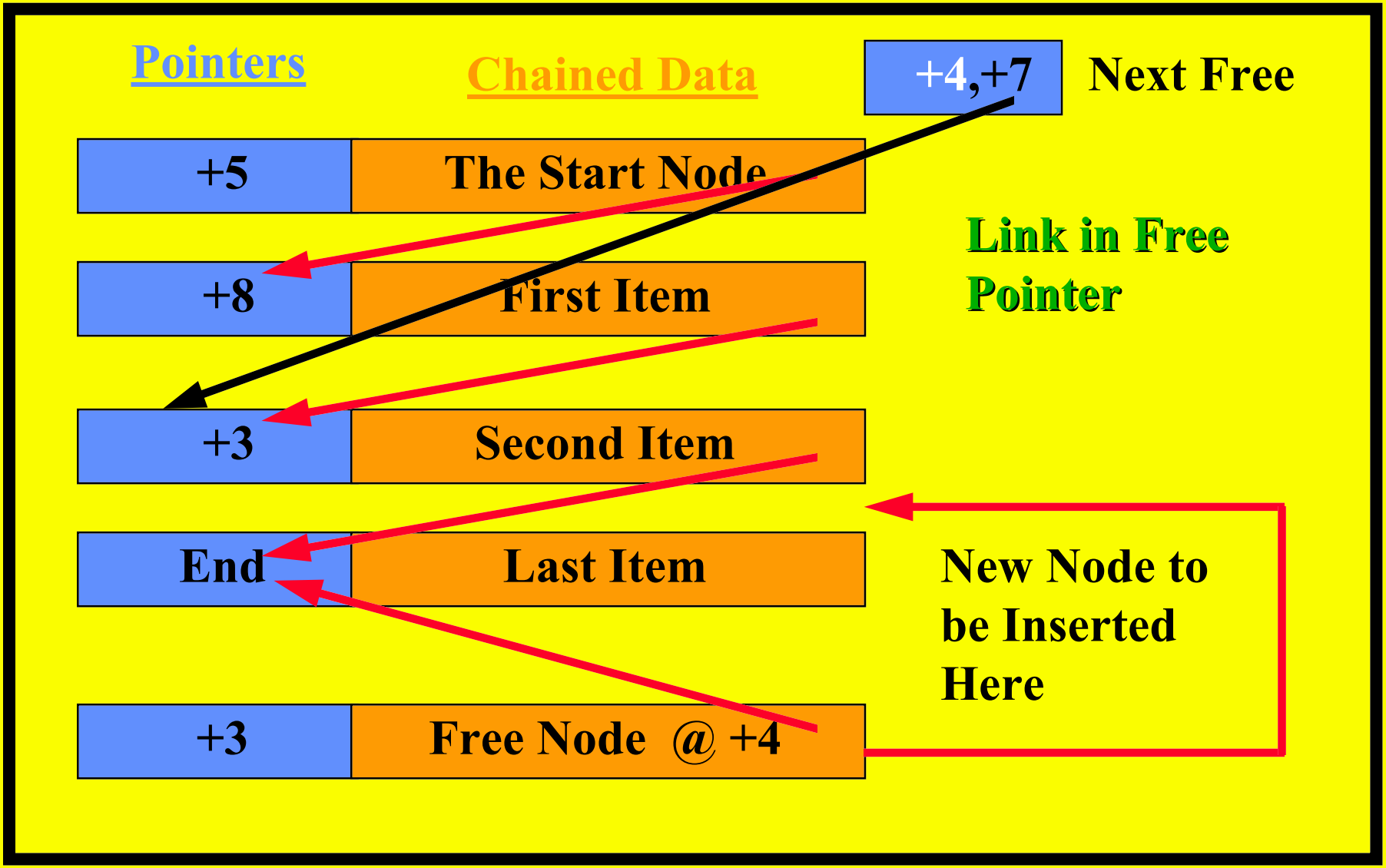
How Item is Inserted.



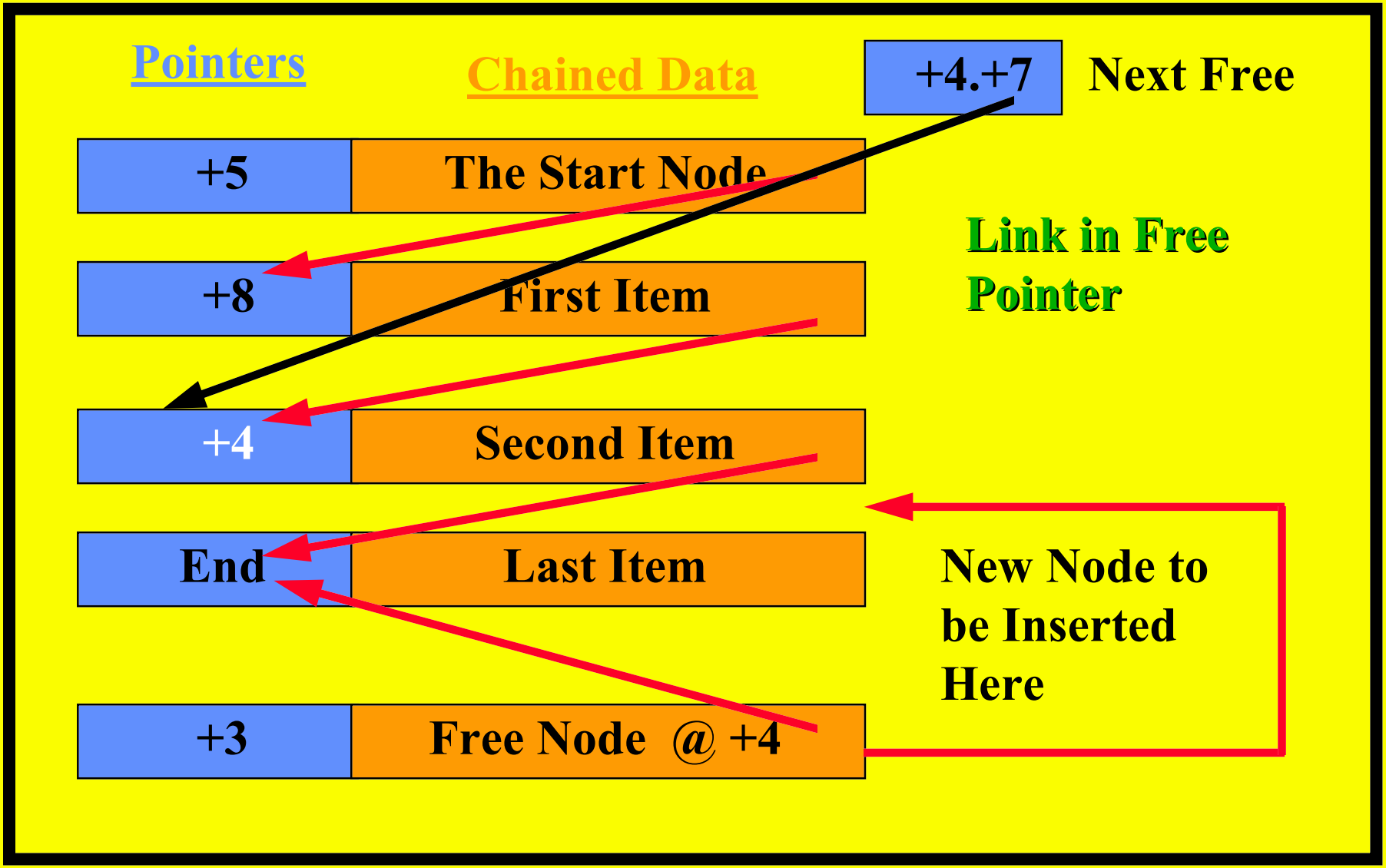
How Item is Inserted.



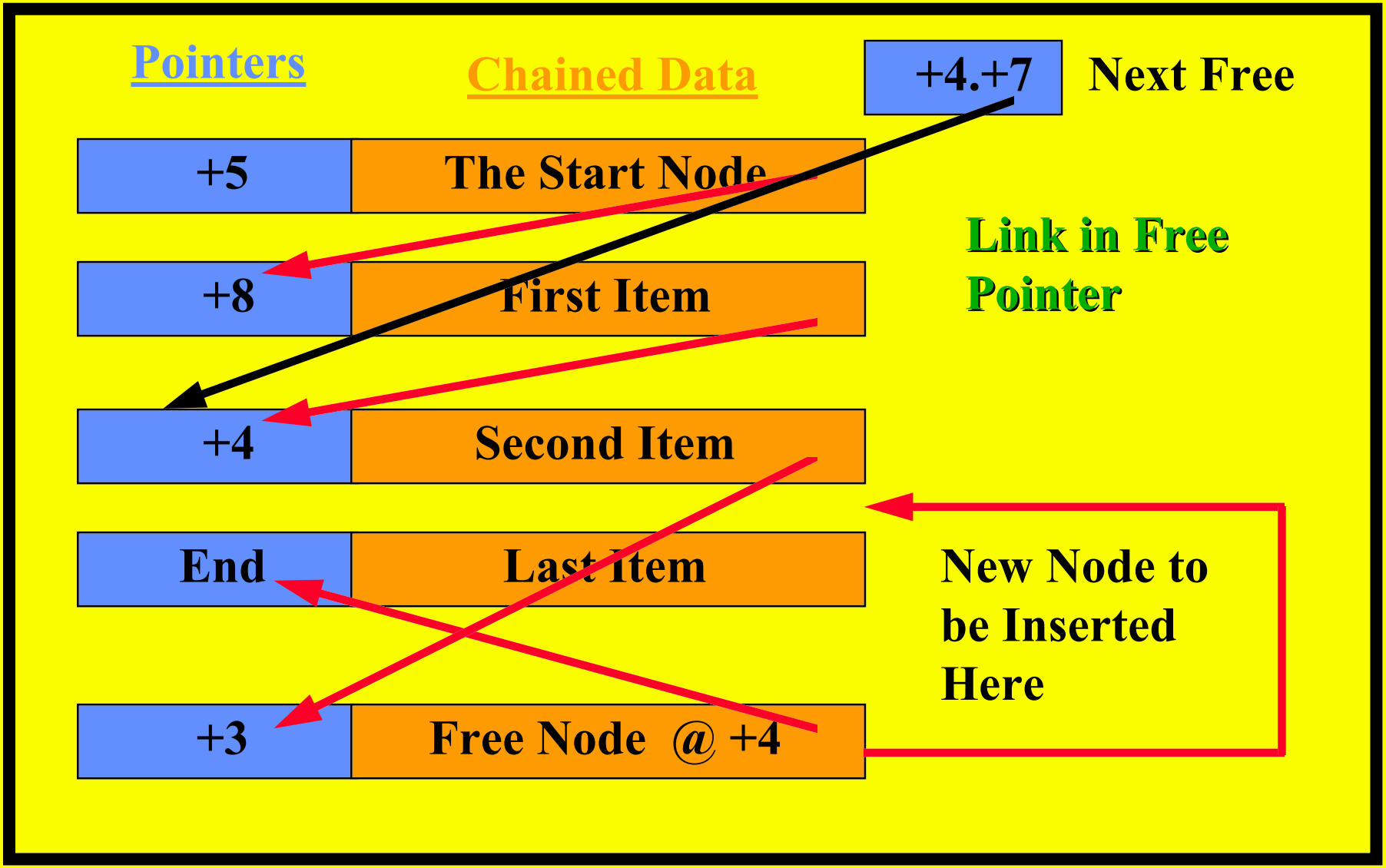
How Item is Inserted.



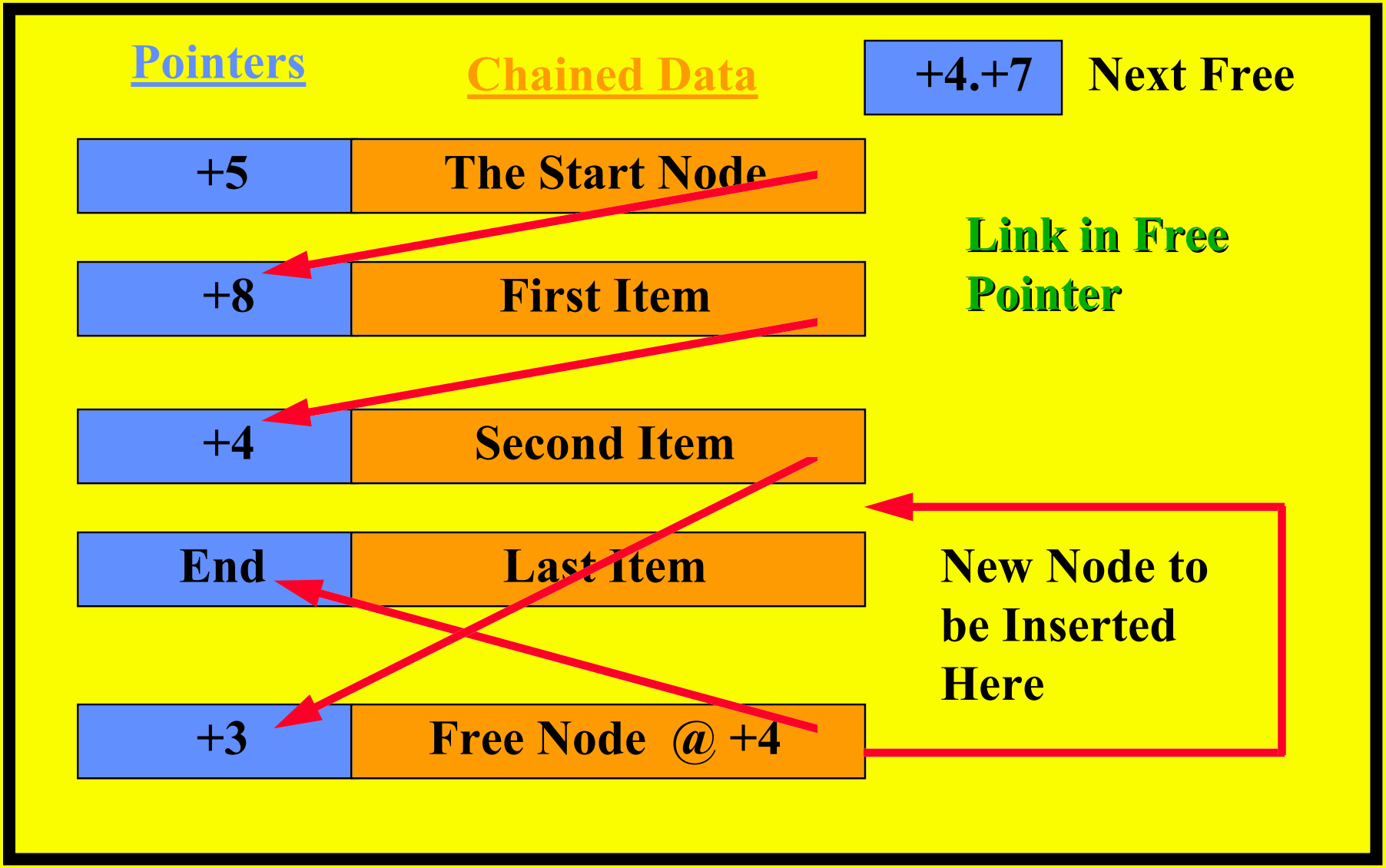
How Item is Inserted.



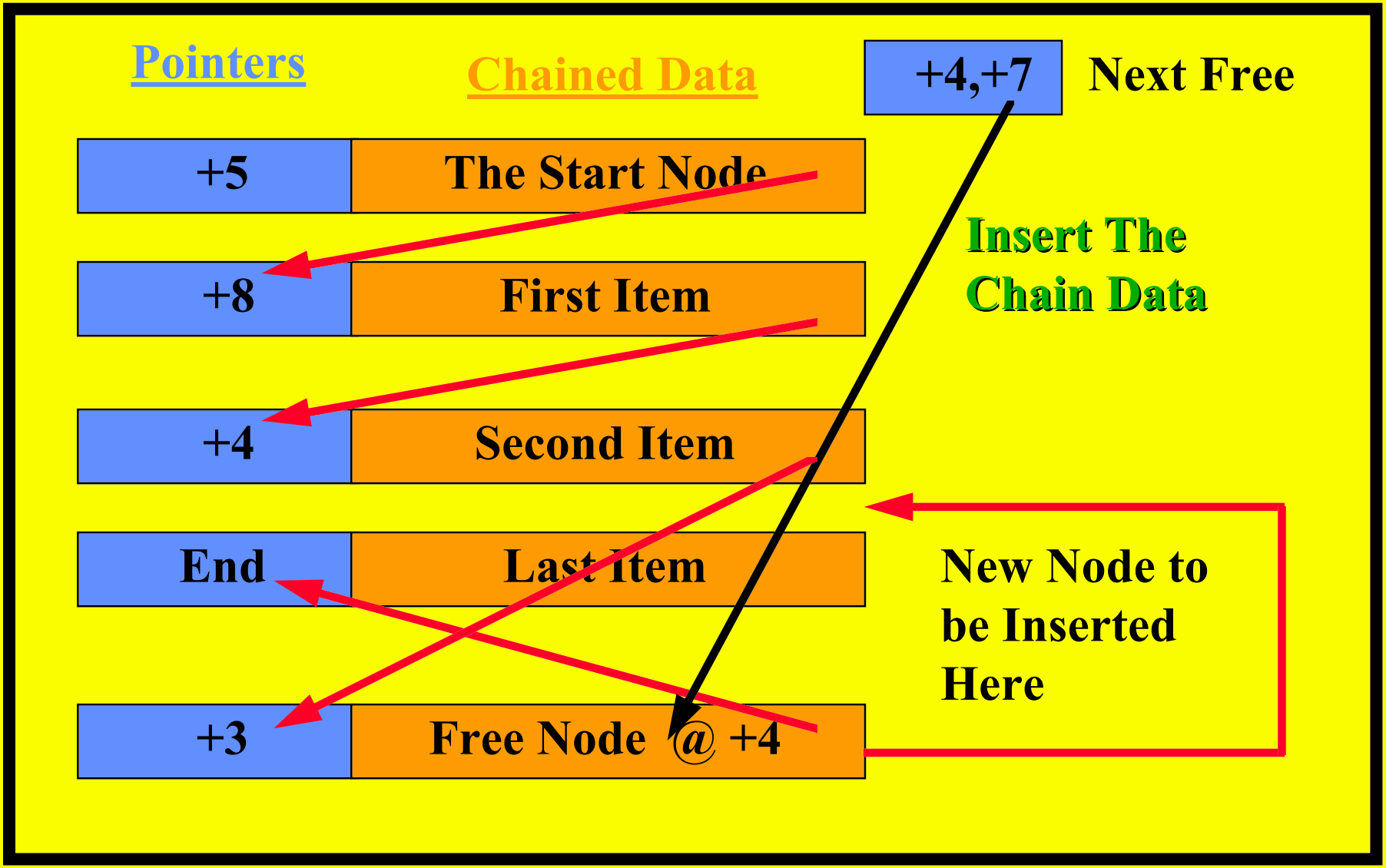
How Item is Inserted.



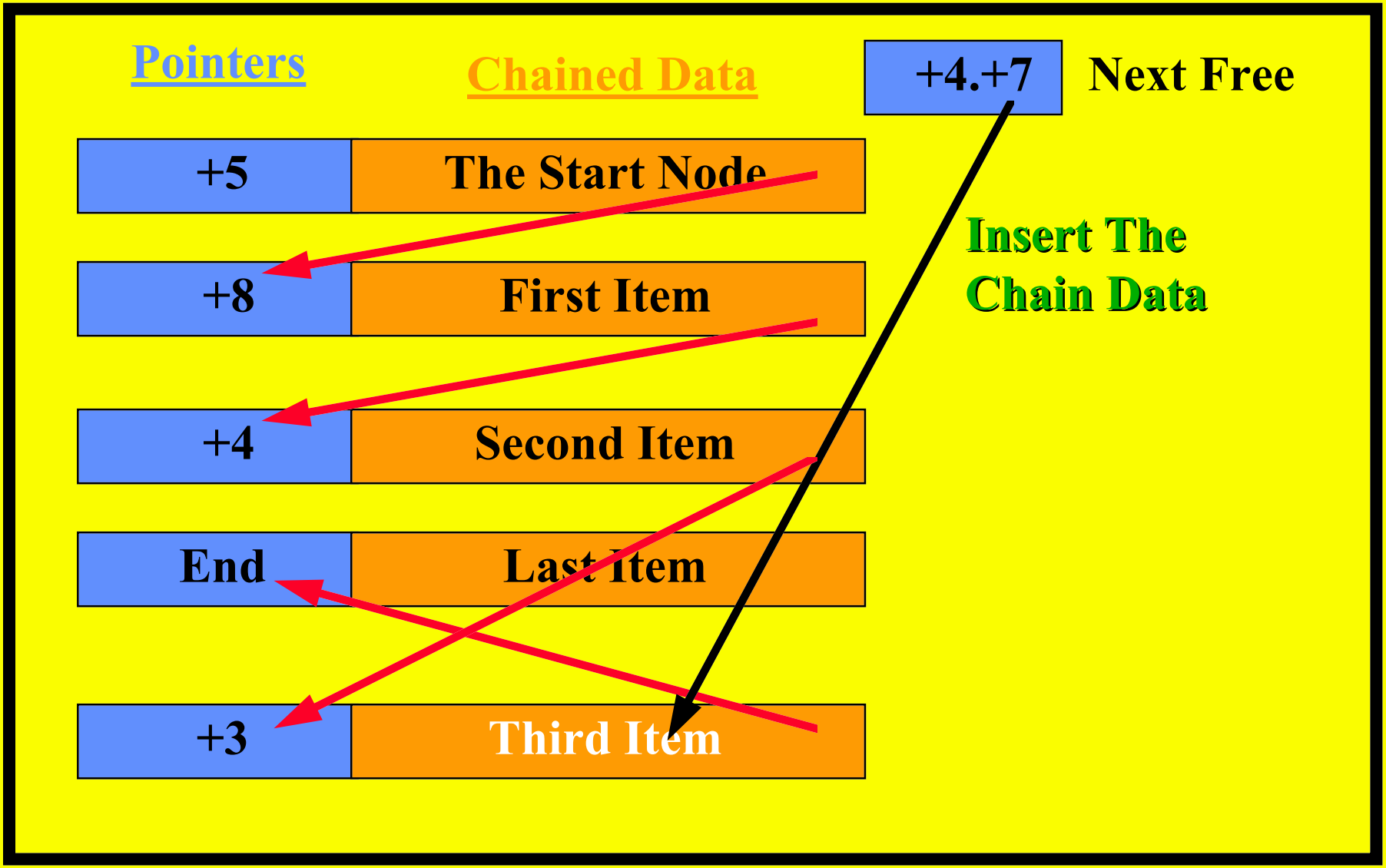
How Item is Inserted.



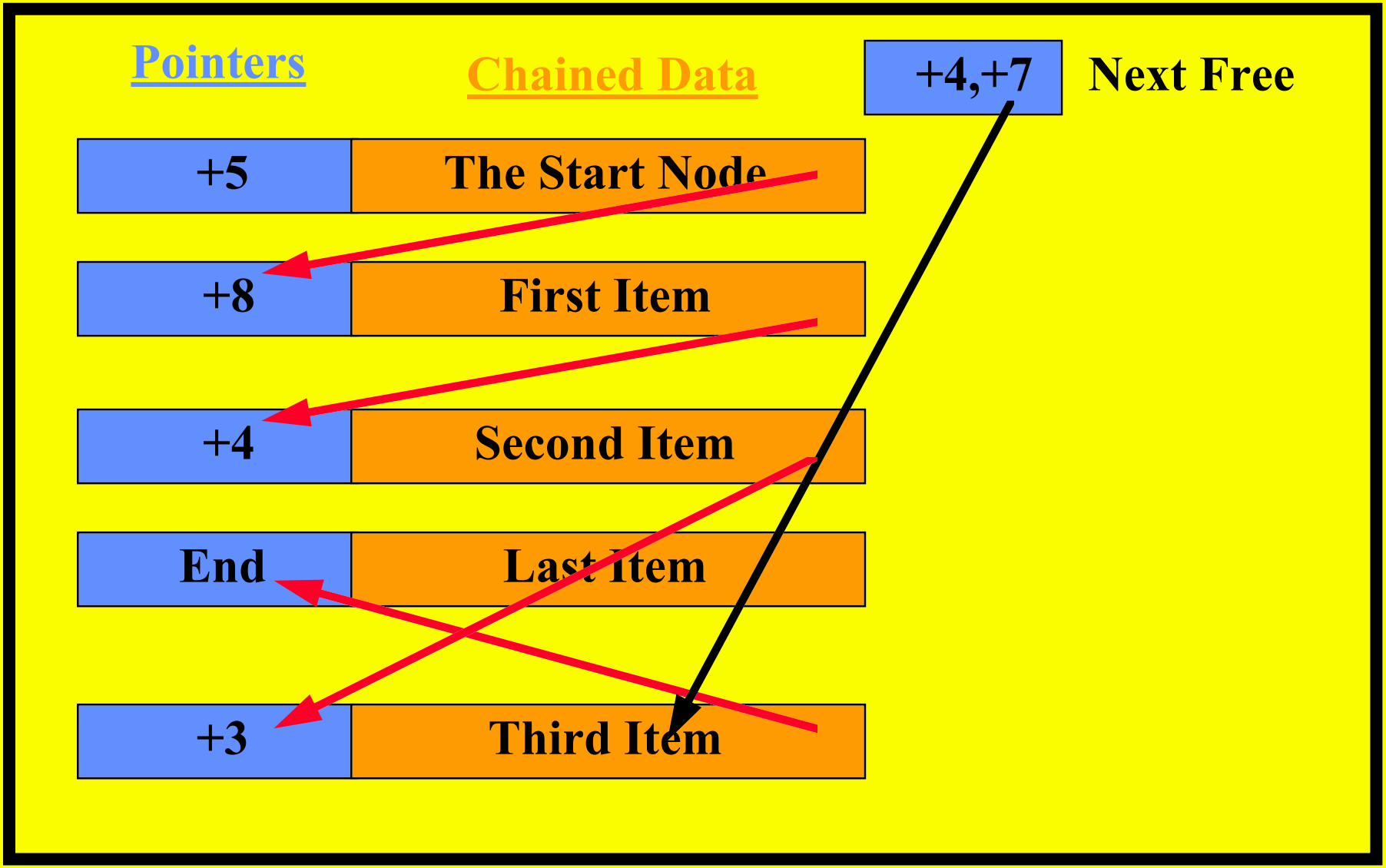
How Item is Inserted.



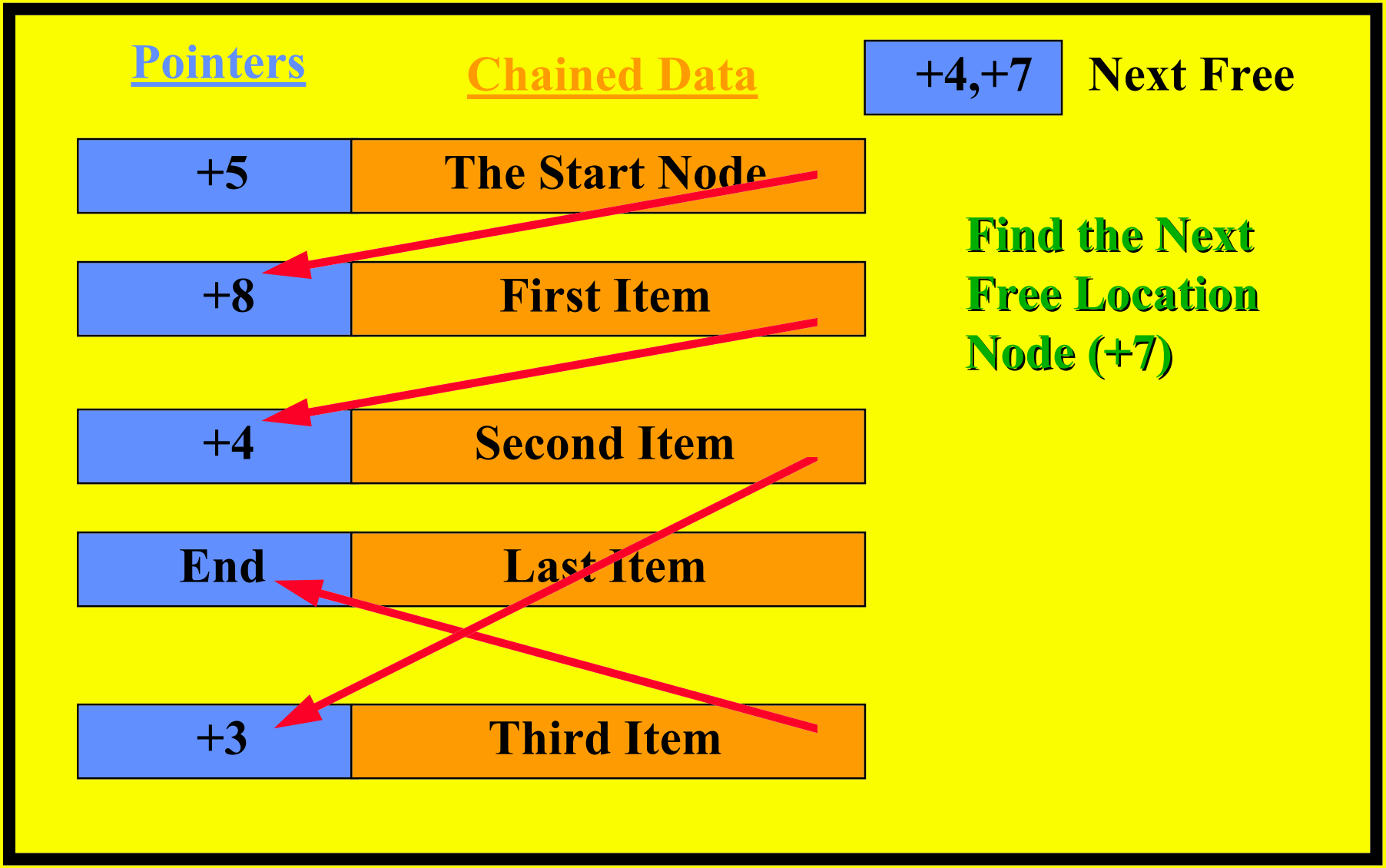
How Item is Inserted.



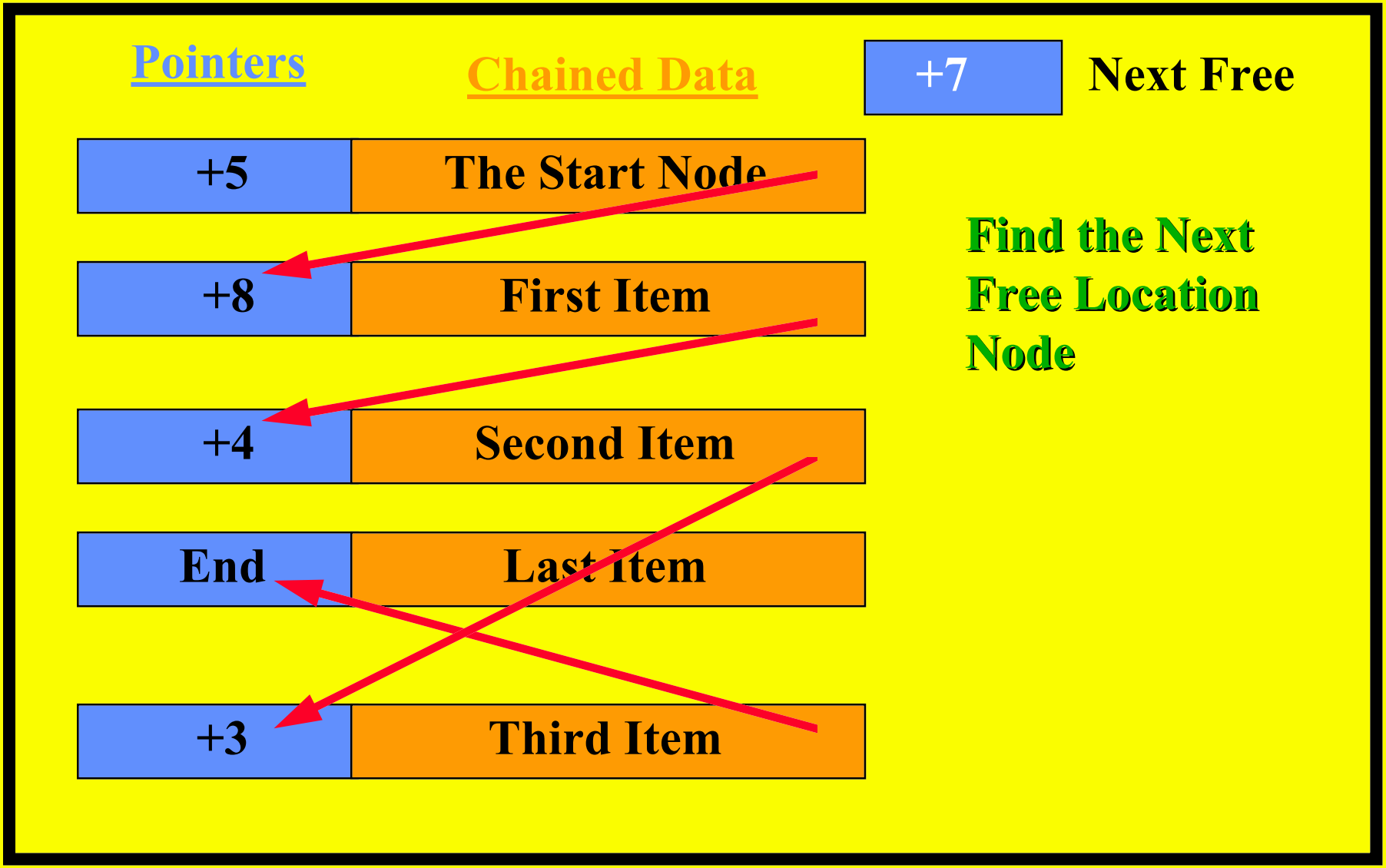
How Item is Inserted.



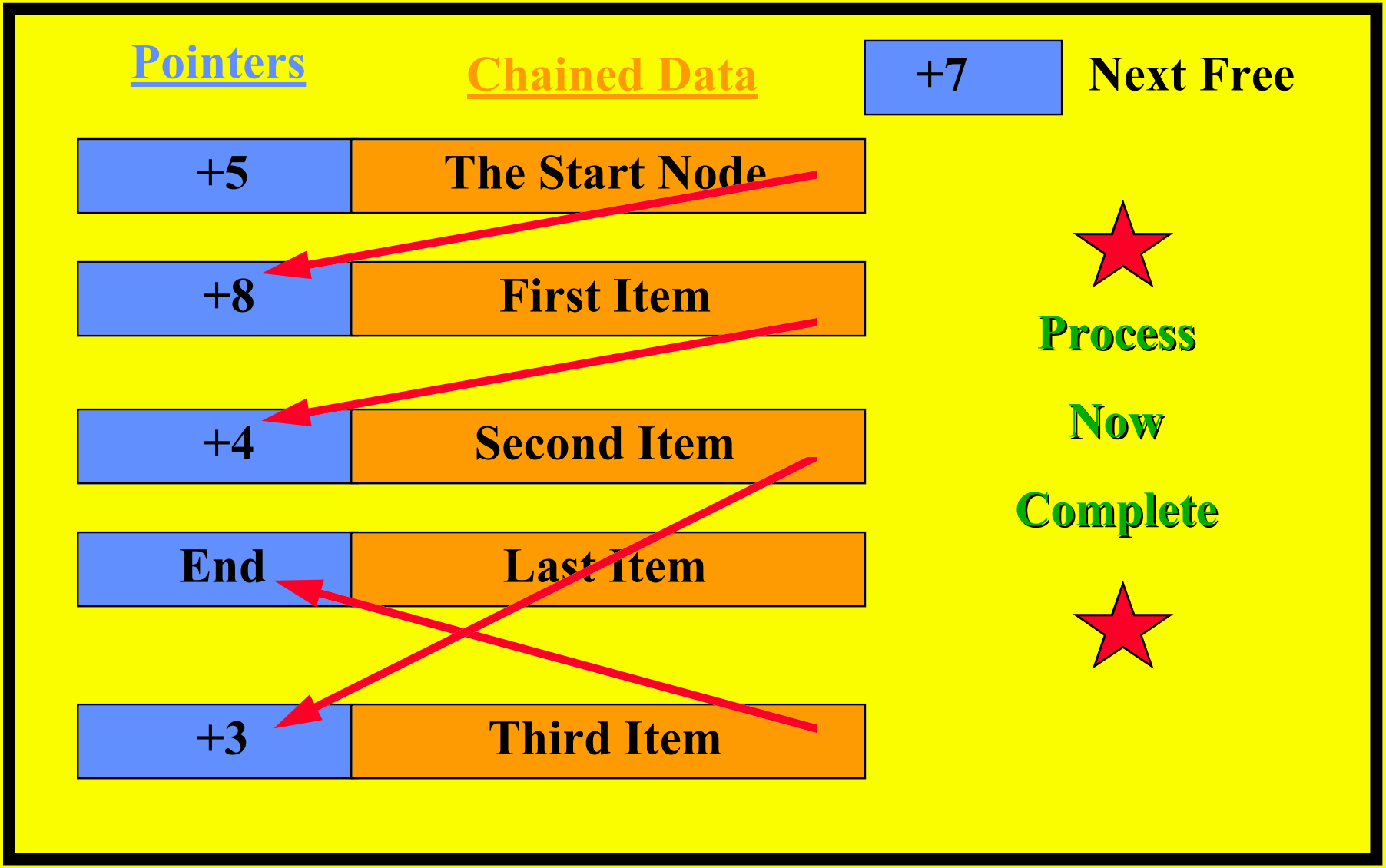
How Item is Inserted.



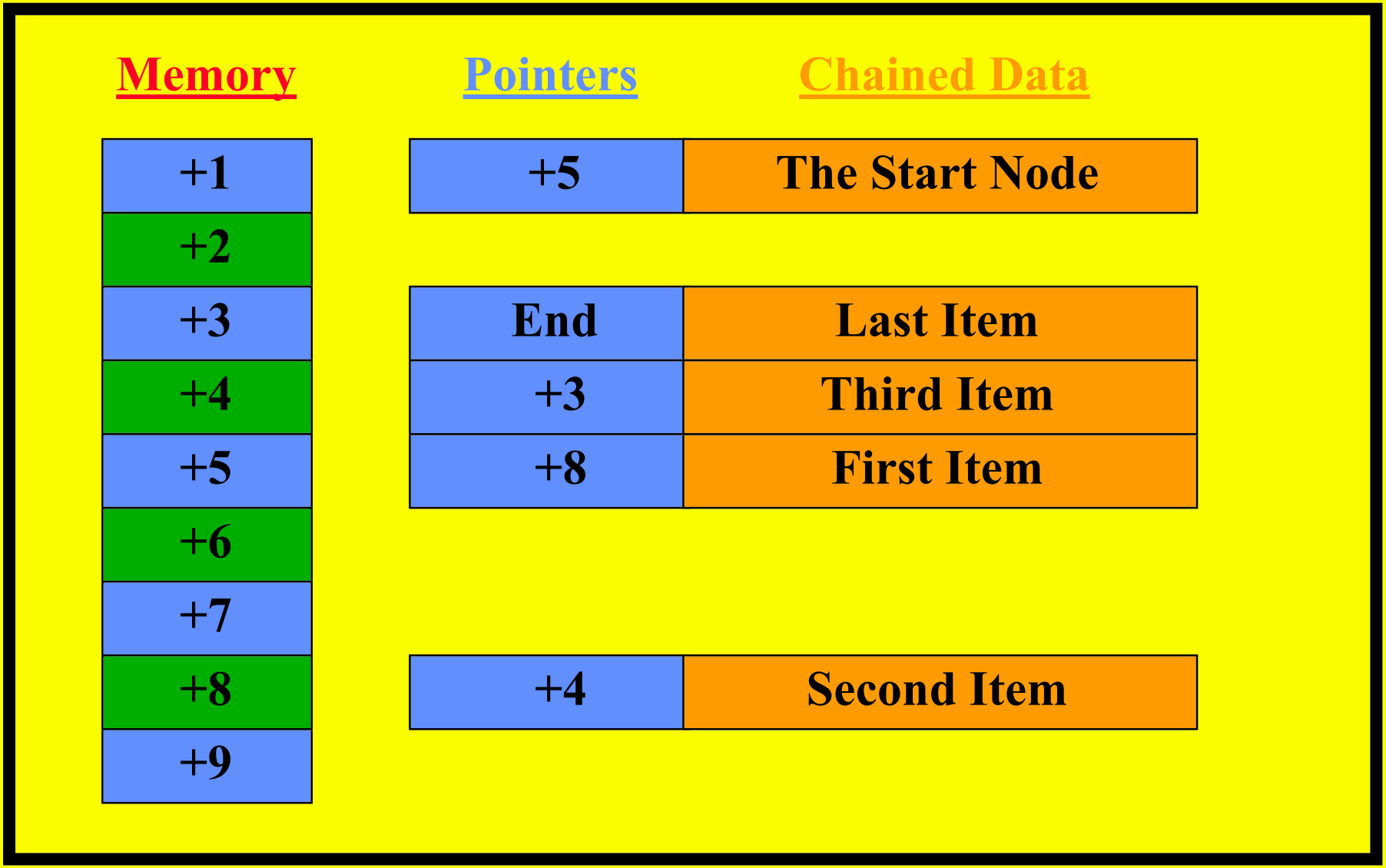
How Item is Inserted.



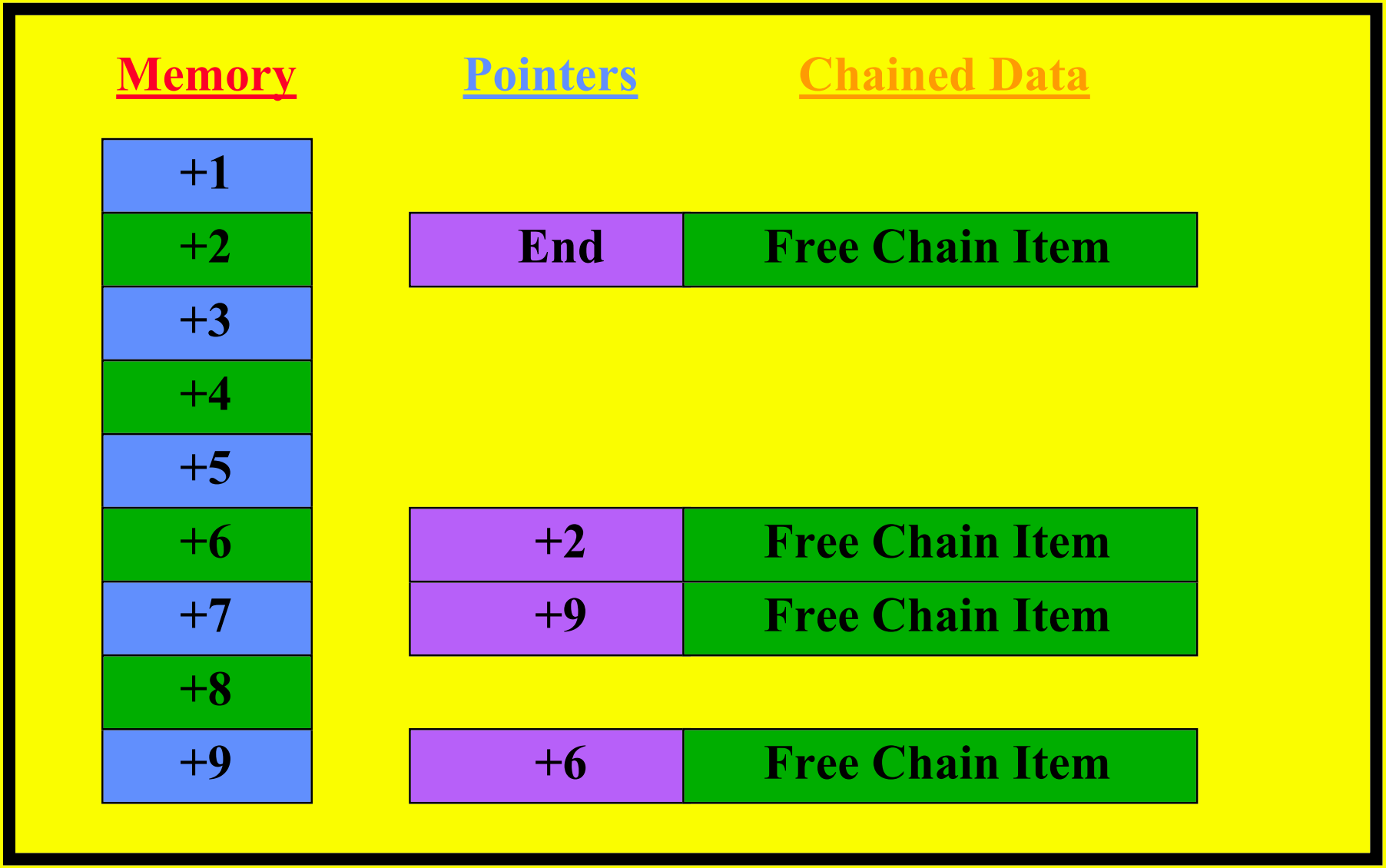
How Item is Inserted.



The Basic Linked List.



The Basic Linked List.



The Basic Linked List.

Memory	Pointers	Chained Data
+1	+5	The Start Node
+2	End	Free Chain Item
+3	End	Last Item
+4	+3	Third Item
+5	+8	First Item
+6	+2	Free Chain Item
+7	+9	Free Chain Item
+8	+4	Second Item
+9	+6	Free Chain Item

What is a Linked List ?

- As you have just seen, the Insertion of a Node in a linked list. This is quite complex operation.
- A similar process is used Delete a Node and the address of the free node is returned to the Free List pointer Table.
- Perhaps this is the time to have a go at developing your own programme.

How is a List Implemented ?
Tricks of the Trade.

- You will need a number of special modules.
- One Module to Initialize the List.
- One Module to Create Extra Free Nodes.
- One Module to Add a Node to the List.
- One Module to Delete a Node in the List.
- Other things to consider :

How is a List Implemented ?
Tricks of the Trade.

- One Module to Modify the Nodes Contents.
- It could have advantages if One Module could switch the Pointers of the Data Set BUT more about that later.
- Note The List may only contain pointers to other structures which have fixed positions.

Optional Exercise No. 7.

Design a programme to Implement a

Linked List.

REMEMBER ERROR CHECKING

Initially build a list for storing Free Nodes.

Then Adapt the List for Insertion and Deletions.

Adapt the Programme to Modify the contents of the Data Store Section.

What is a Linked List ?

- It is worth understanding this structure as it is very flexible and used in many applications such as :-
- Databases , Compilers, Editors and most importantly it is how most Disk File Systems are Referenced and Structured.

What is a Linked List ?

- With Disk File Systems you will have many List Head Pointers. (Directory Entries)
- The chaining Pointers may rather than be tied to the Data could be kept in a separate structure of its own.
- e.g. The File Allocation Table (FAT)

What is a Linked List ?

- Another Structure within this same theme is the Double Linked List.
- This variant not only allows you to travel forwards through the Data but also has information about the return traversing path.
- This is without doubt the most complex structure to update, however it just takes a little “thinking about” to succeed.

The Double Linked List.

How the Basic Information Entity is Structured.

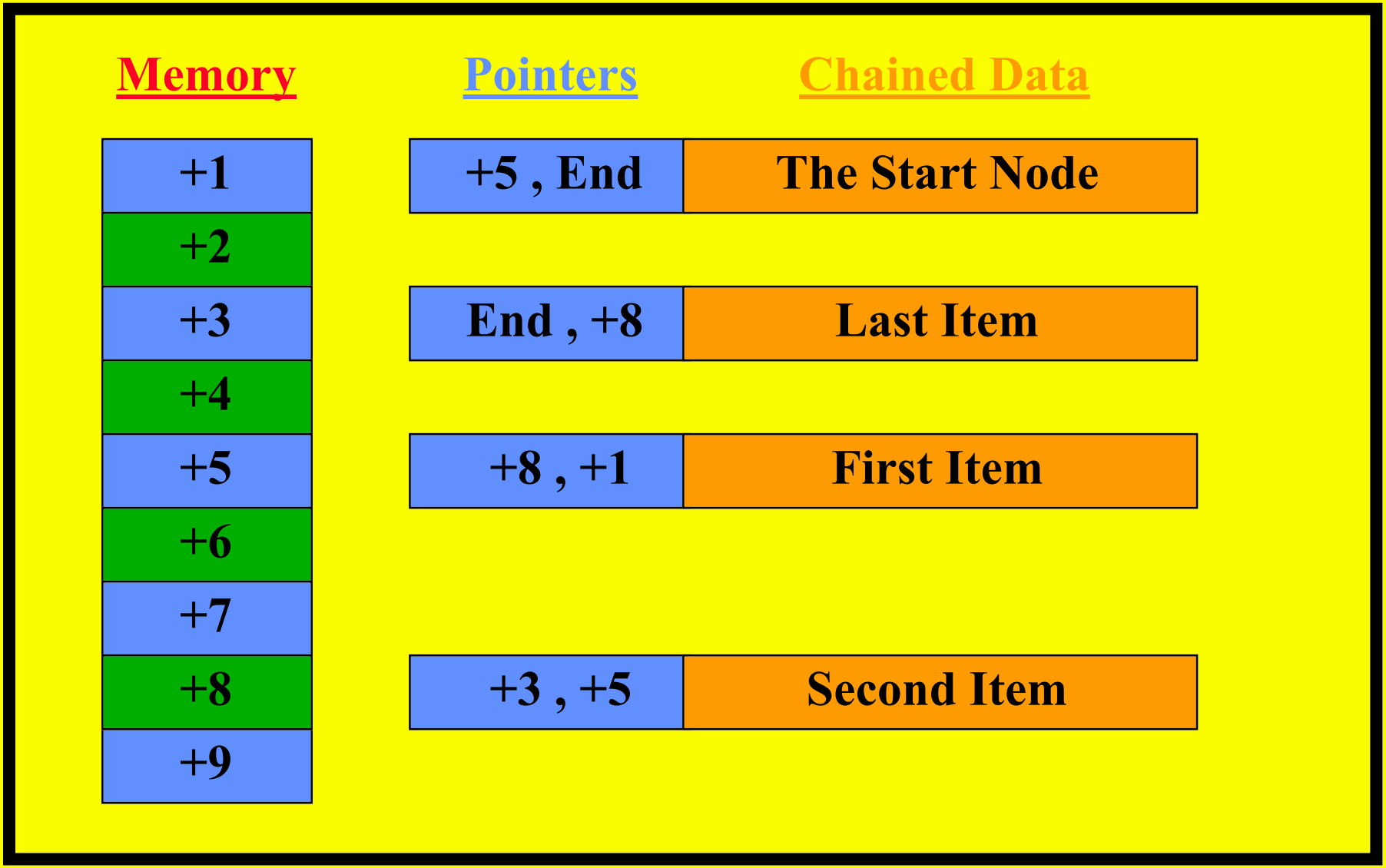
- Each Entities Consists of :-
- The Information or Data Store Section.
- The Pointers Section.
- This consists of a Forward Pointer.
- It also consists of a Backward Pointer.

The Double Linked List.

How the Basic Information Entity is Structured.

- Each Entities Consists of :-
- The Information or Data Store Section.
- The Pointers Section.
- This consists of a Forward Pointer.
- It also consist of a Backward Pointer.

The Double Linked List.



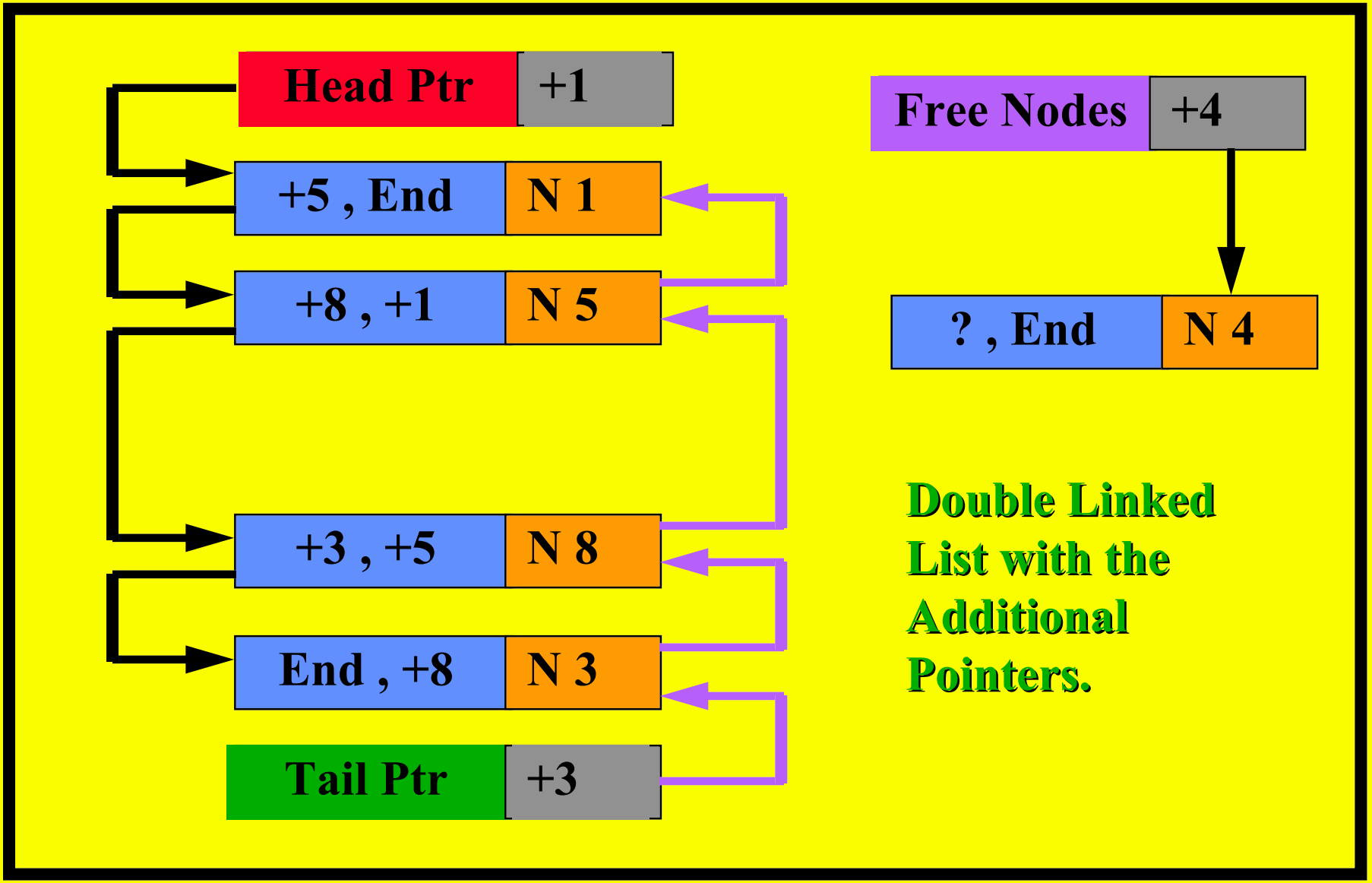
The Double Linked List.

To Implement this Structure

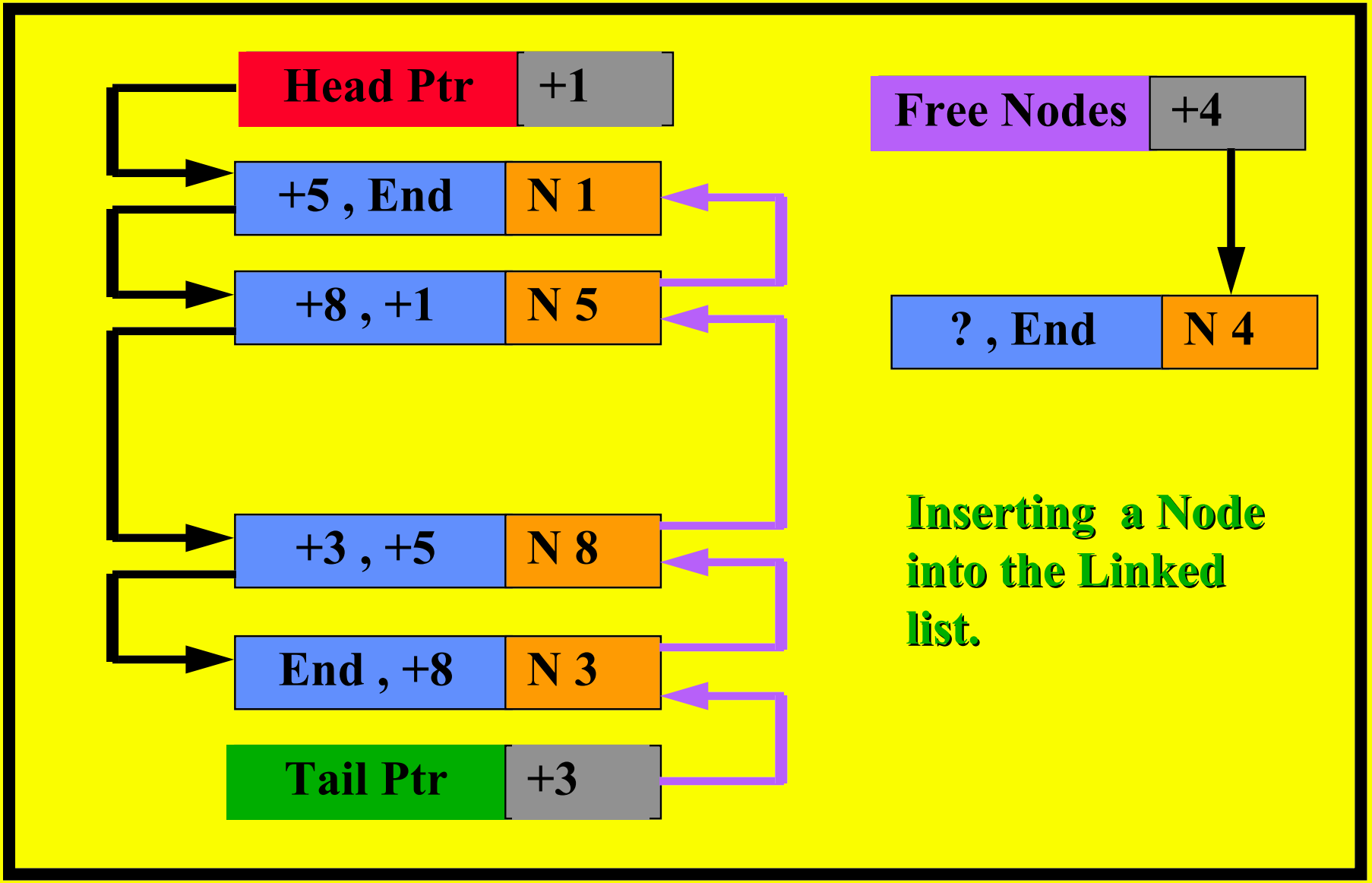
We need a few Extra Variables and these are :-

- **A List Head Pointer.**
- **A List Tail Pointer**
- **A Free List Table and Pointers.**

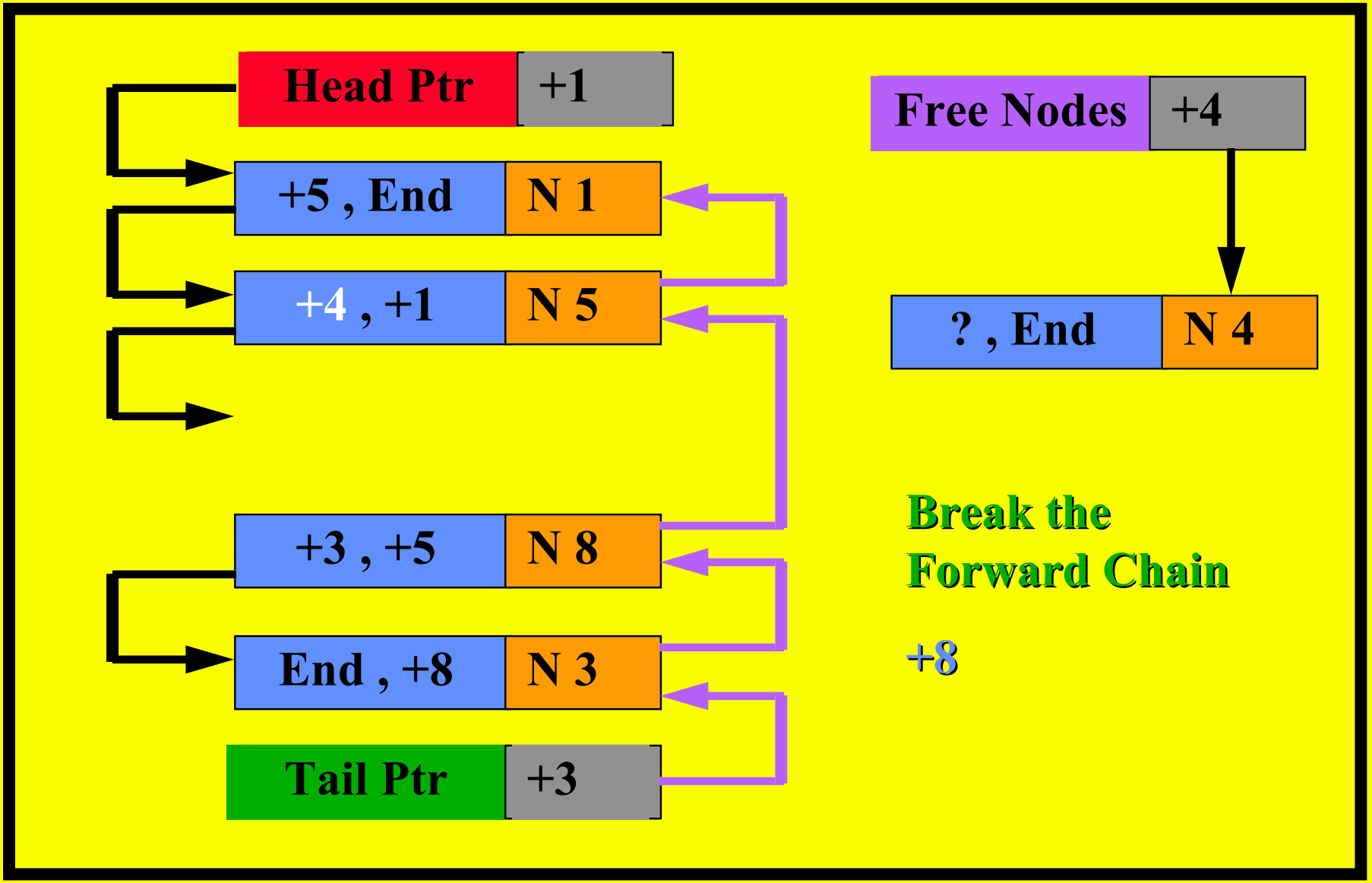
The Double Linked List.



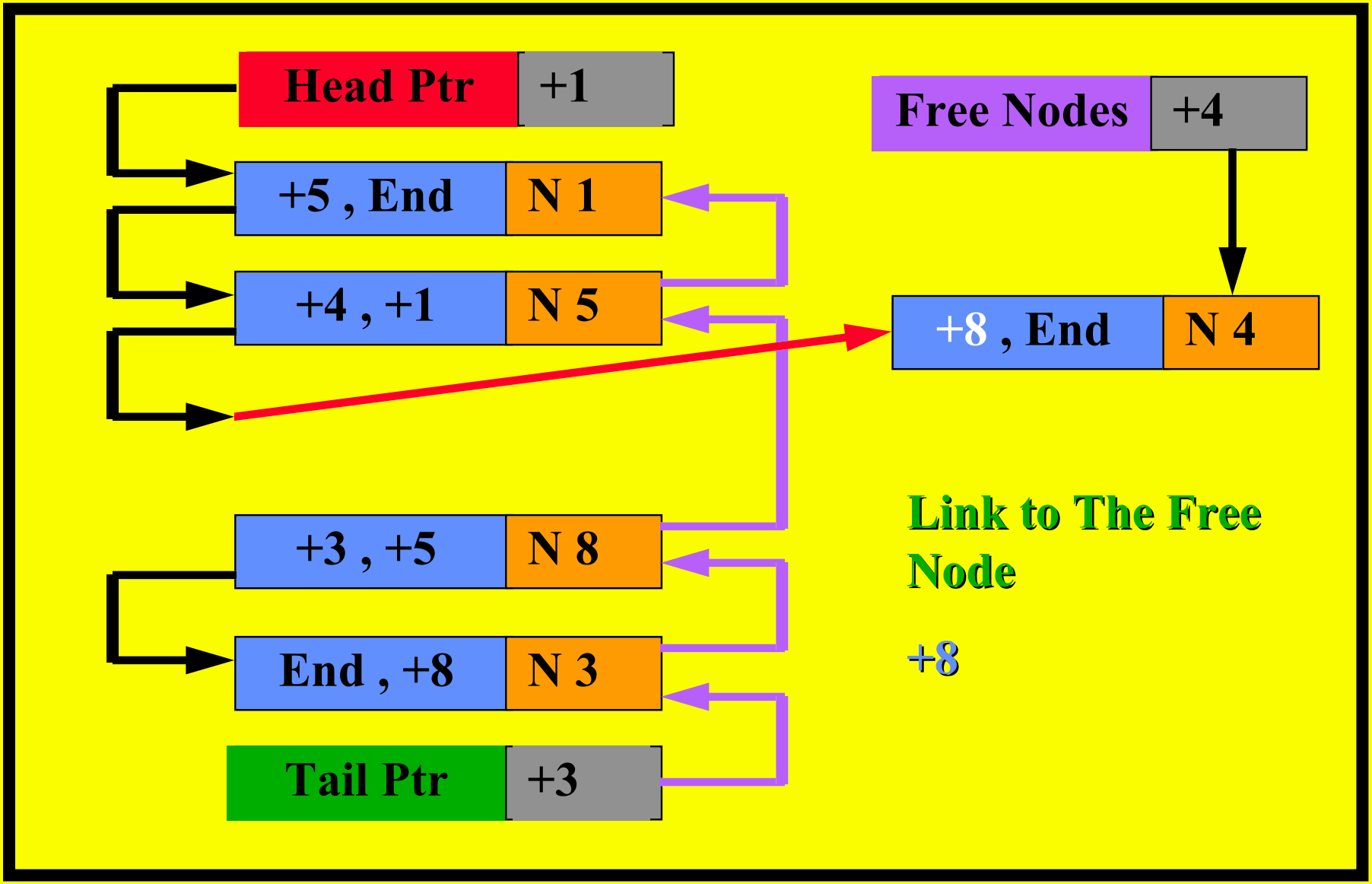
The Double Linked List.



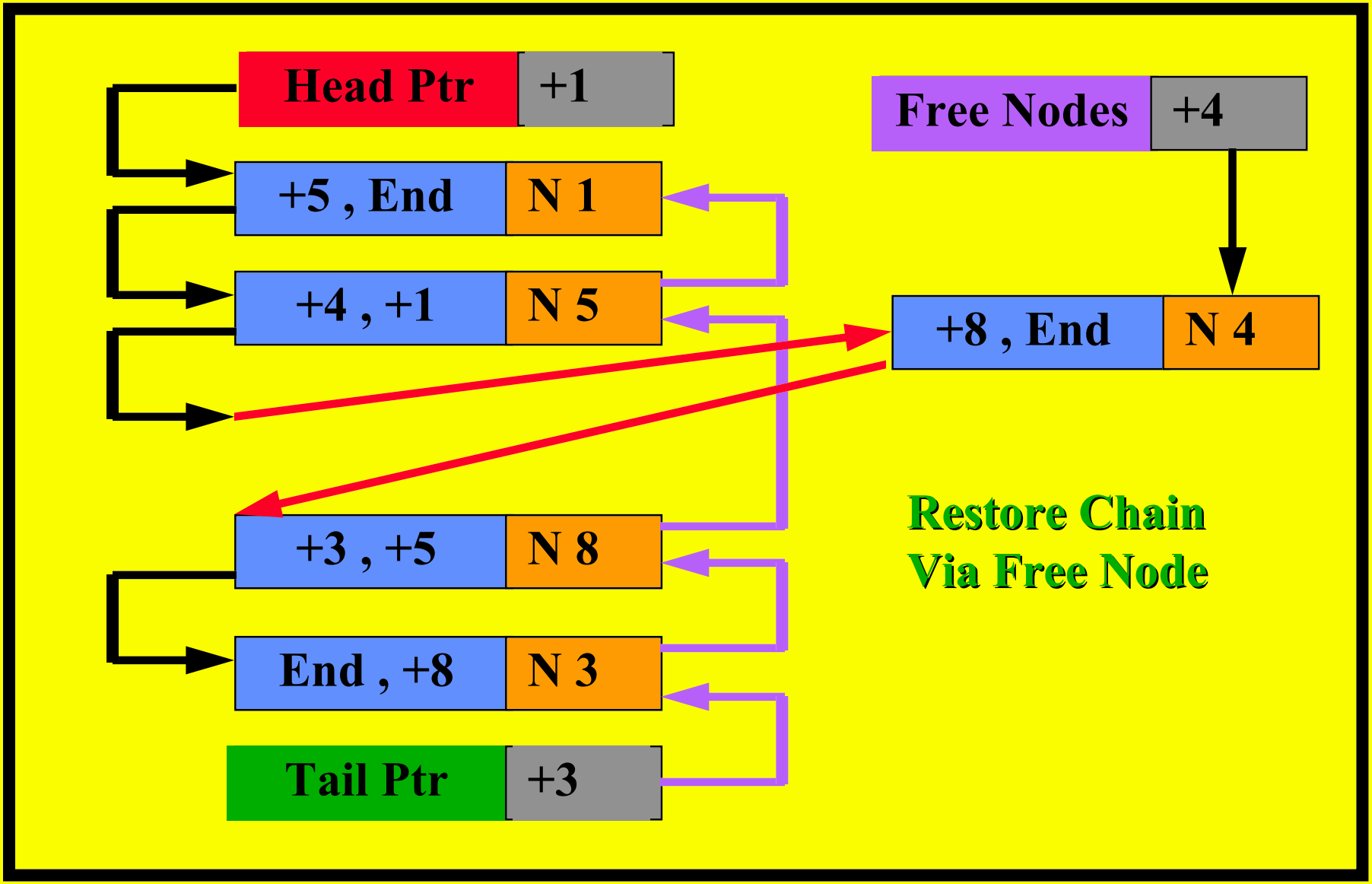
The Double Linked List.



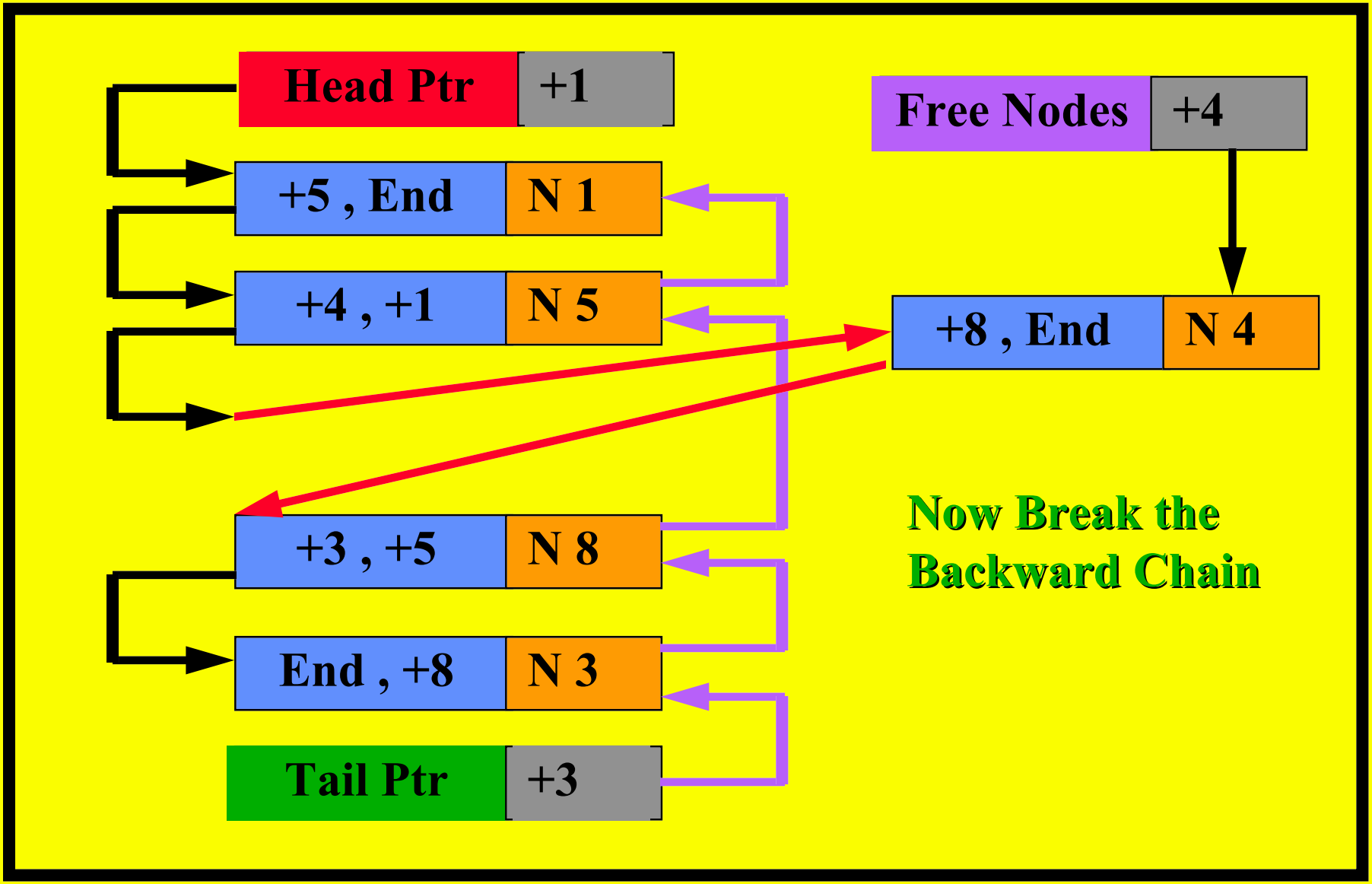
The Double Linked List.



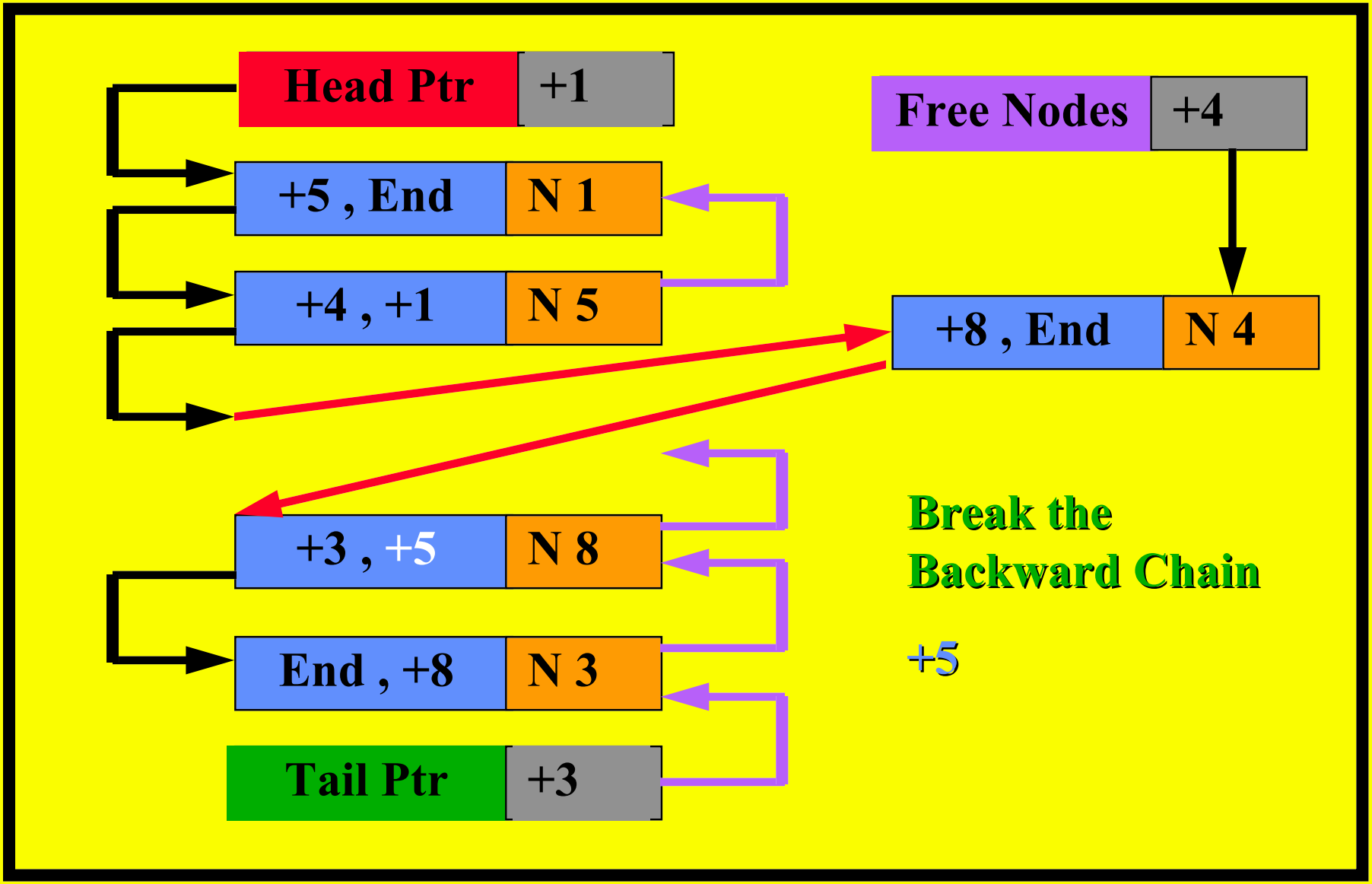
The Double Linked List.



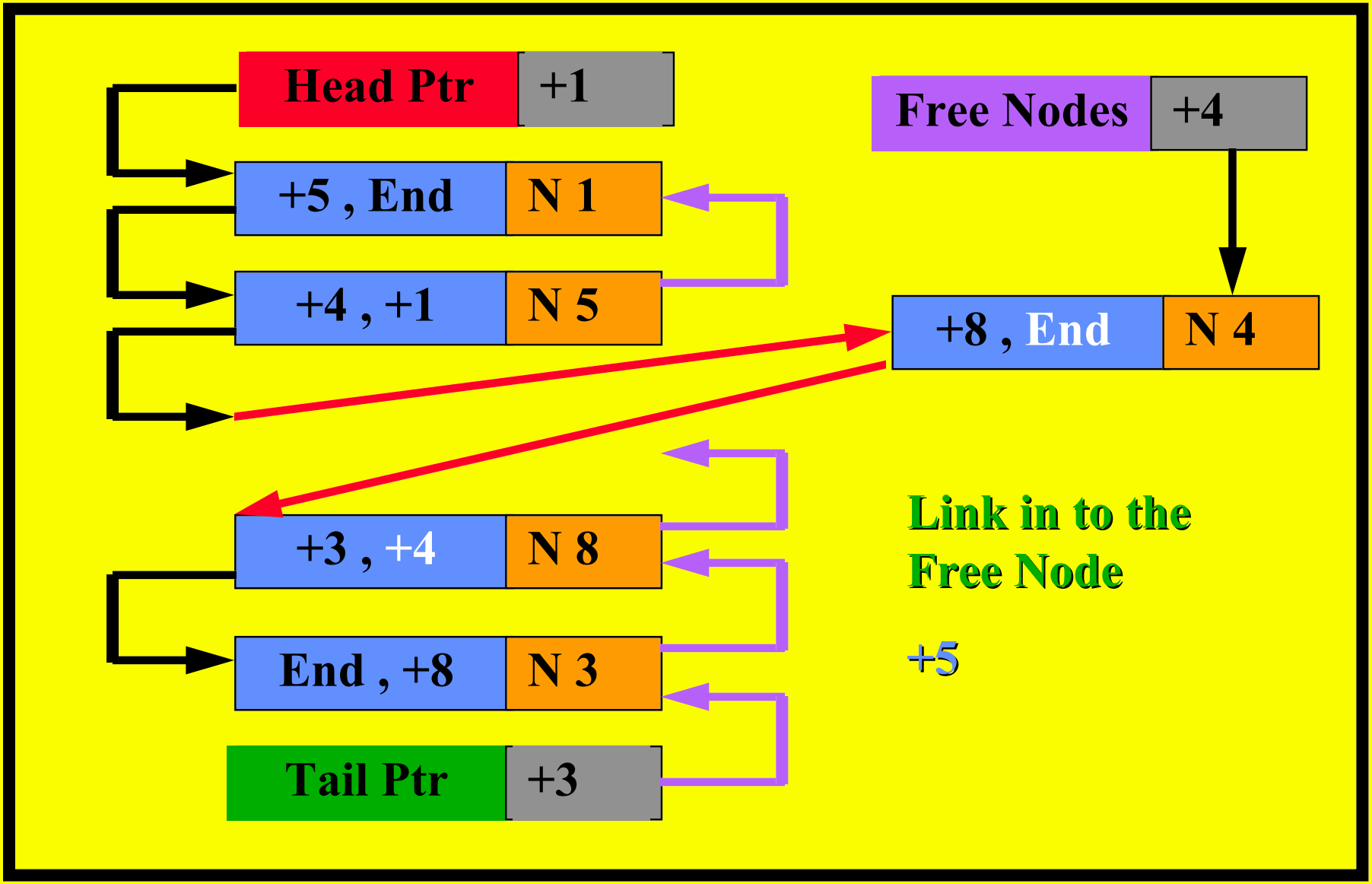
The Double Linked List.



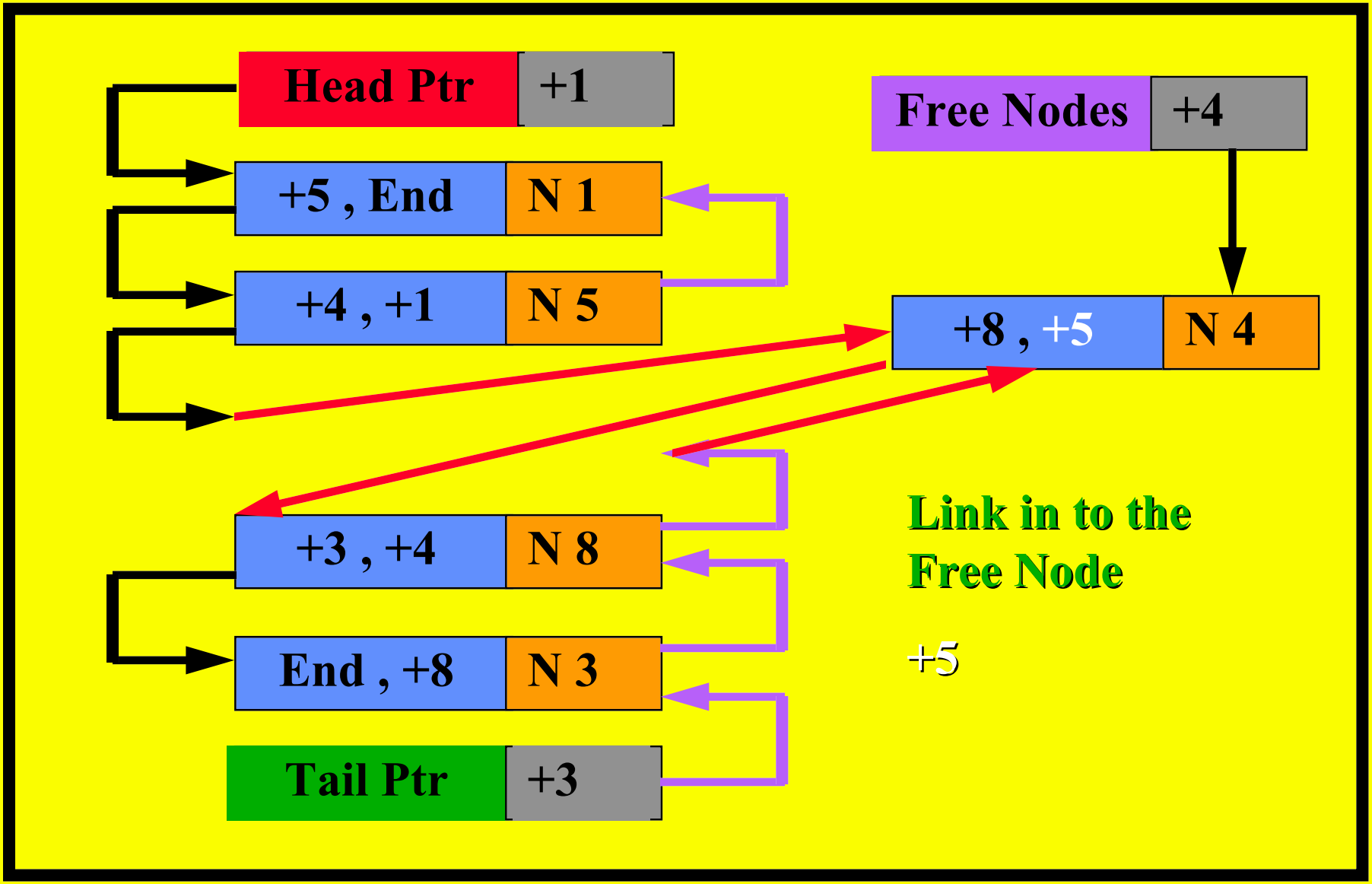
The Double Linked List.



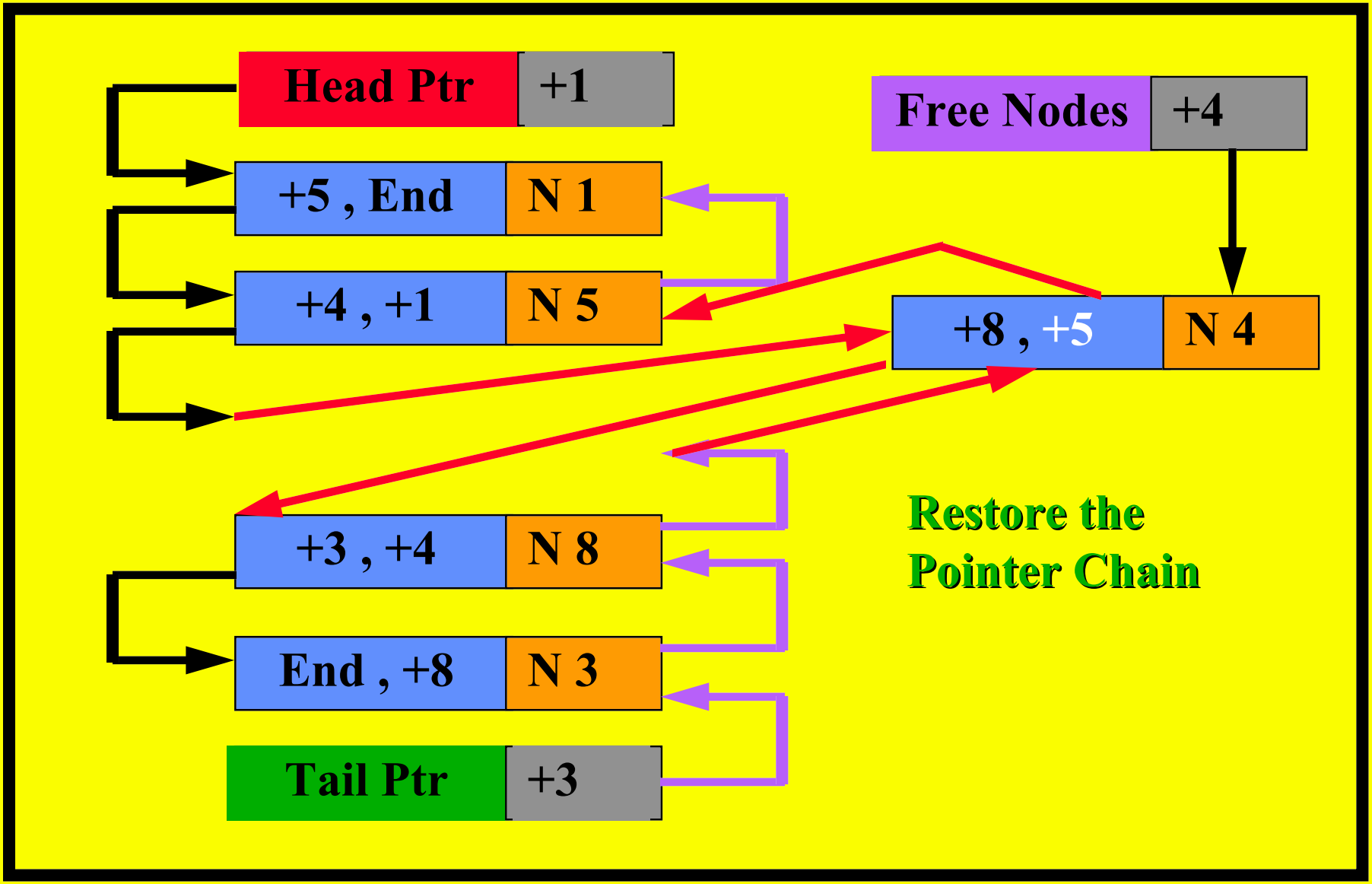
The Double Linked List.



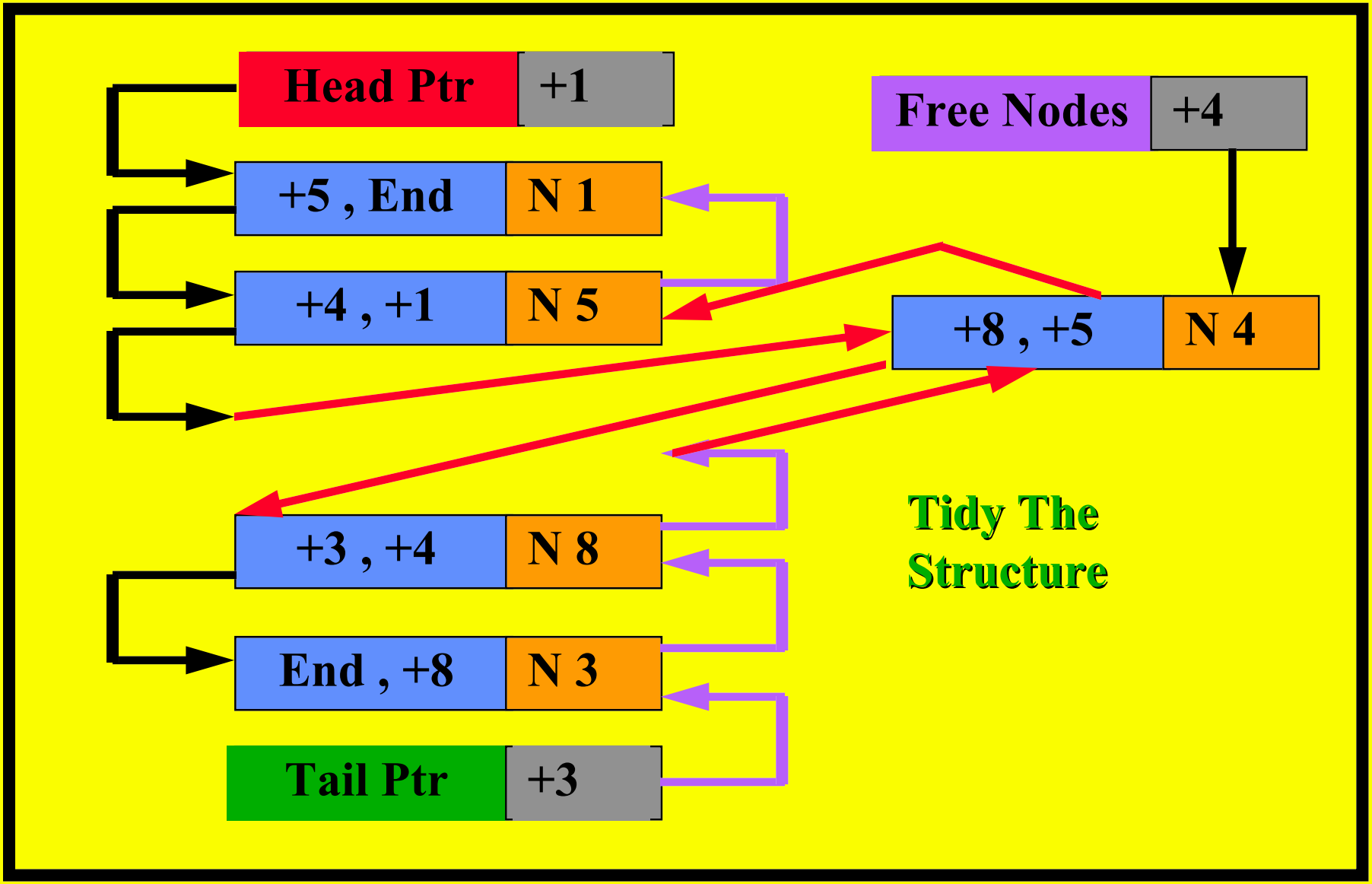
The Double Linked List.



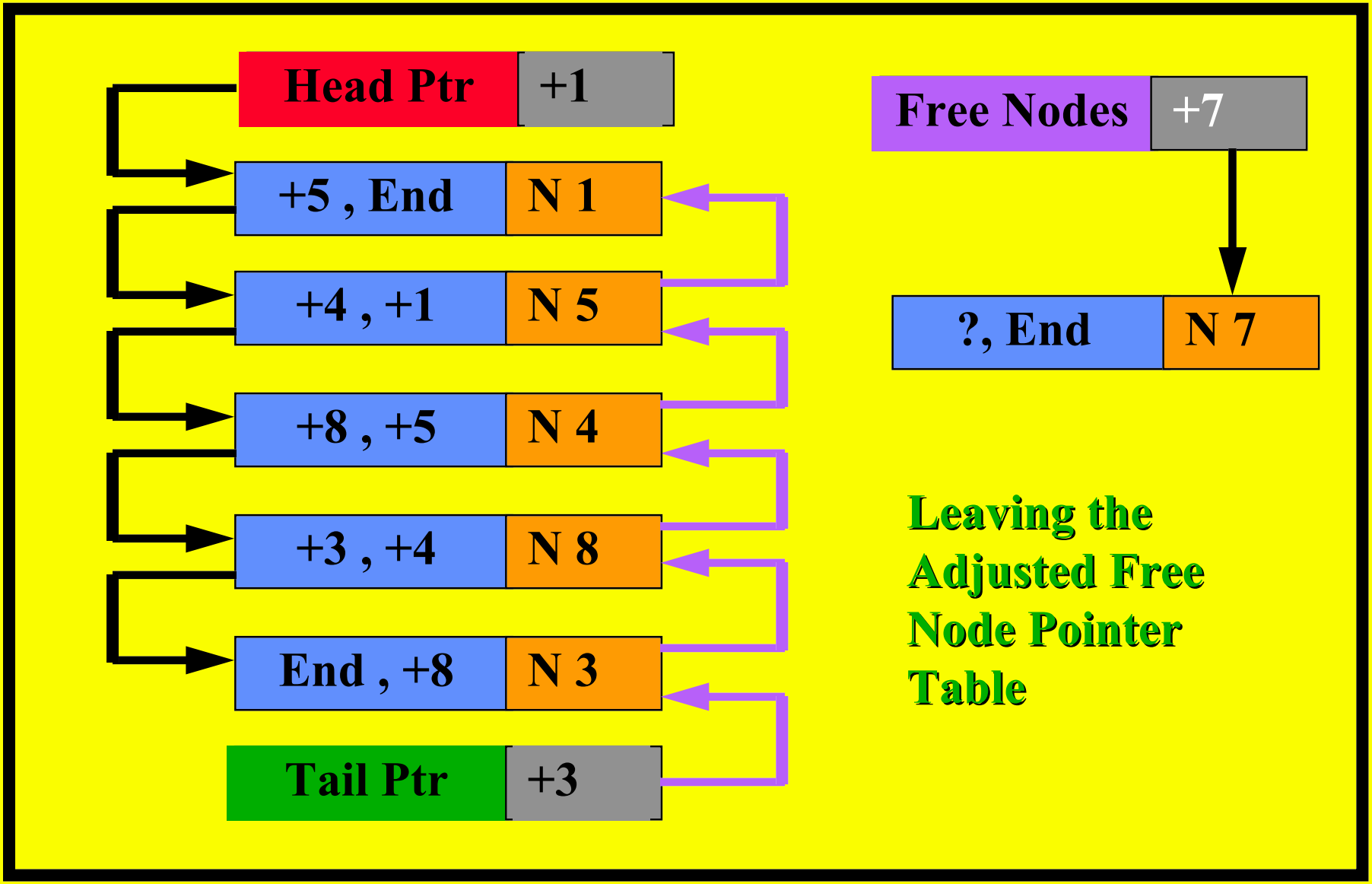
The Double Linked List.



The Double Linked List.



The Double Linked List.



Optional Exercise No. 8.

Design a programme to Implement a

Double Linked List.

REMEMBER ERROR CHECKING

Adapt the Single Linked List for New Features.

Then Adapt the List for Insertion and Deletions.

Adapt the Programme to Modify the contents of the Data Store Section.

Data Structures

Review

Why use Double Linked Lists ?

- Consider the situation :
- Add information with the pointer always Set to the Head of the List.
- Remove Information with the pointer always Set to the Head of the List.
- What have you got
- You have got ... a **STACK or HEAP**
- Which can Even handle complex data.

Why use Double Linked Lists ?

- Another situation :
- Add information with the pointer always Set to the Tail of the List.
- Remove Information with the pointer always Set to the Head of the List.
- What have you got
- You have got ... a **QUEUE**
- Which can ALSO handle complex data.

Why use Double Linked Lists ?

- Yet another situation :
- As you can enter Data at any point in the list this could be an effective basis to produce a Sort system.
- What if you had many sets of Head and Tail Pointers.
- "Wow" Multiple Unique Accessible data sets all in the Same Data Structure.

Double Linked Lists

YES

THEY MUST BE WORTH CONSIDERING

Covered Topics

STACKS

QUEUES

Linked Lists

Data Structures

Any Questions ?

Presentation End

Revision Page

Title	Data Structures including Stacks, Queues and Linked Lists
Author	R. J. Spriggs
Last Update	28/April/2004
Version	2.01
Edit	0101