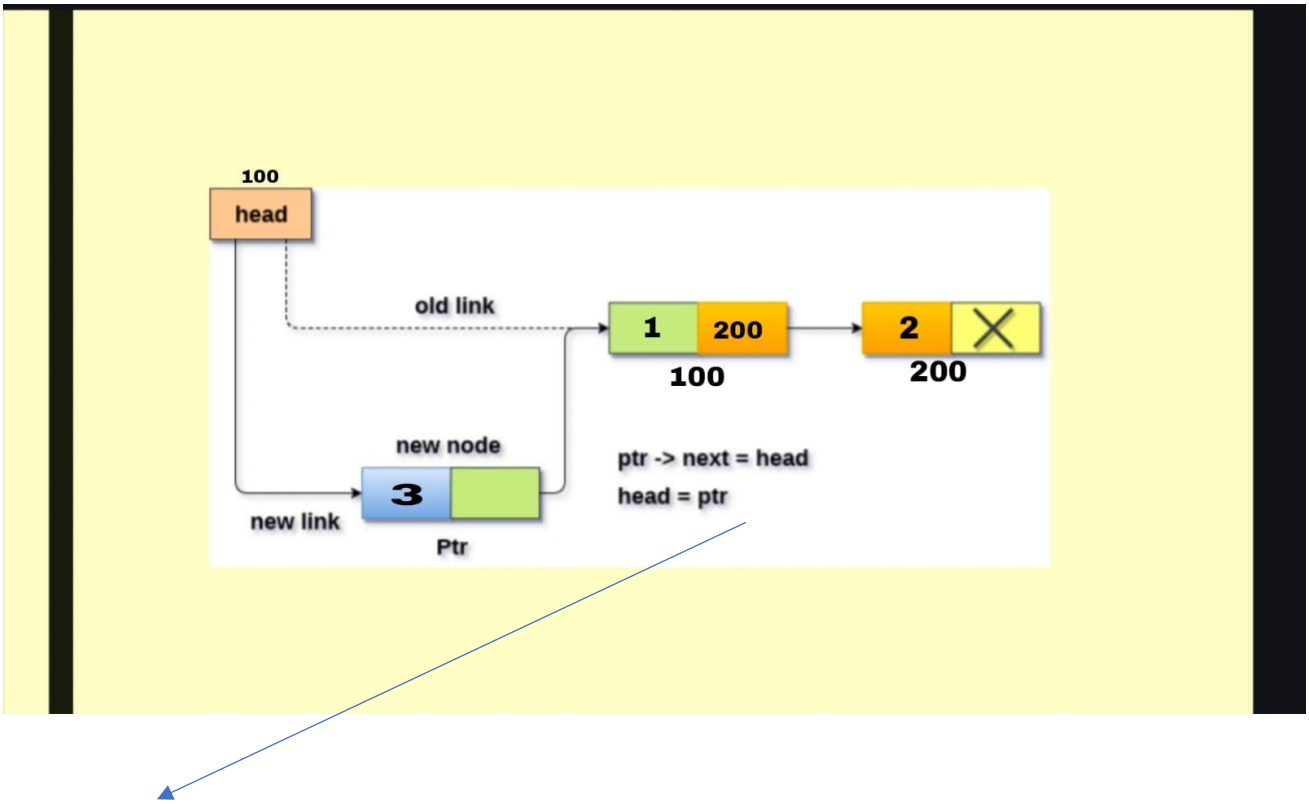
------SINGLY LINKED LIST-----

Inserting a Node at the beginning of the linked list (i.e between head and the 1st node):



LOGIC:

ptr -->next = head: means we are assigning(giving) head value to ptr-> next(100 is adress of 1st node as well as the head node)

ptr = head : means we are assigning (giving) ptr value to head.

USER: *Basically, linked list is a collection of nodes in which one node holds the address of the another node. Like the first node holds the address of the second node, the second node holds the address of the third node, the third node holds the address of the fourth node, and so on till it reaches the tail node. And in the tail node, it doesn't have any address, so it is called null address. It is called null address since after that node, it doesn't have any other node. So it is the tail node or terminated node. And the first part of the linked list is called the head node which doesn't have any variable. It just has the address part of the first node. And all other nodes after the head node hold some address as well as*

some variable. So this is the basic concept of linked list. That is the singly linked list.

Refined Version : A **linked list** is a collection of **nodes**, where each node stores:

1. **Some data** (like a number), and
2. The **address (or pointer)** of the **next node** in the list.

In a **singly linked list**, each node points to the **next node only** (not backward). For example:

- The **first node** points to the second,
- The **second node** points to the third,
- The process continues until the **last node**.

The **last node** is called the **tail node**. It doesn't point to any other node because there's no node after it, so its **next pointer is set to NULL** — which means "no address."

The **head** of the linked list is a special pointer that stores the address of the **first node**. It's not a node itself, just a reference to the first node in the list.

So in summary:

- head points to the first node,
- Each node has a **data part** and a **next pointer**,
- The **last node's next pointer** is NULL, marking the **end** of the list.

✅ What you got right:

- Nodes point to each other using addresses.
- The last node has a NULL pointer.
- The head is used to access the first node.

🛠️ Minor corrections:

- The **head is not a node**, it's just a pointer to the first node.

------SINGLY LINKED LIST-----

- Every node, including the first, has both data and address parts (not just from the second node).

User: *So from the given image, we have to know some terminologies. The terminologies are Pointer, Next, Previous, and Head. In a given node, we have two parts, left-hand-side part and right-hand-side part. The left-hand-side part is called the previous part, which holds the variable. And the right-hand-side part is called the next part, which holds the address of the next node. That's all. And the head part is the beginning part, which doesn't have any variable. It just holds the address of the next node. And here, pointer is the name given to a new node. It's not a pointer. It's a pointer variable. It's just a name given to the new node, which we are to insert in the beginning of the first node. We are to insert that node named ptr(pointer) at the beginning. It's not the pointer that we actually study in C++. That is, star p. It's not the pointer that holds the address. It's just a name given to that particular node, which we are to insert. That should be kept in mind.*

Refined Version: From the given image, there are a few important terms to understand: **Pointer (ptr)**, **Next**, and **Head**.

🔹 **Node Structure:**

Each **node** in a singly linked list has **two parts**:

1. **Left-hand side** – This stores the **data** (like a number).
2. **Right-hand side** – This stores the **address (or reference) to the next node** in the list.

📌 Note: In a **singly linked list**, we do **not** have a "previous" part. The term “previous” is used in **doubly linked lists** only, where each node also points back to the previous one. So, avoid using the word "previous" for singly linked lists.

------SINGLY LINKED LIST-----

🔹 **Head:**

- The **head** is not a node.
- It’s a special pointer (variable) that **holds the address of the first node** in the linked list.
- It doesn't store any data—just helps us start the list.

🔹 **Next:**

- This refers to the **pointer part of a node** that stores the **address of the next node**.
- In code, it's usually written like node->next.

🔹 **Pointer (ptr):**

- In this image, ptr is the **name of the new node** we're inserting at the beginning of the list.
- It's not a traditional C++ pointer like int *p. Instead, it's just a **node variable** (a structure or object that represents a node).
- So when we write ptr->next = head, we're connecting the new node (ptr) to the existing first node.

📌 Important: The new node (ptr) becomes the new head, so we do head = ptr to update the head pointer.

🧠 **Summary (Singly Linked List):**

- A node contains **data + next pointer**.
- head holds the address of the **first node**.
- The **last node's next** is NULL.
- ptr is just a variable name for the **new node being inserted**.
- There is **no previous part** in singly linked lists.

------SINGLY LINKED LIST-----

USER: *So the main logic or the concept behind inserting the new node at the beginning of the link list is that we will first establish the connection of the pointer node with the Next node and then we will establish the connection with the head Because if this order is to be maintained Because if we first connect to head and then we connect the new node with the second node Then our address will be lost and we don't do in that way So basically we do in that way that first pointer node is connected with the Next node and then pointer(new) node is being connected to head . This is the order that should be followed This is the main logic*

Refined Version : The main concept behind inserting a new node at the beginning of a linked list is that **we must maintain the correct order of connections** so that we don't lose access to the existing list.

Here's the correct logic:

1. **First**, we connect the new node (often called ptr or newNode) to the current first node of the list by doing:

   ```
   newNode->next = head;
   ```

This step connects the new node to the rest of the list.

2. **Then**, we update the head to point to the new node:

   ```
   head = newNode;
   ```

Now the new node becomes the first node of the list.

We follow this order **because if we update the head first**, we lose the address of the original first node, and the rest of the list becomes unreachable (or lost in memory).

So the correct order is:

- **Step 1**: Connect the new node to the existing list.

- **Step 2**: Update the head to point to the new node.

This is the main logic and the safest way to insert a node at the beginning.

User: *As we know that the head holds the address of the first node, and the first node holds the address of the second node, and so on. So, looking at the image, if we want to understand the main logic behind the insertion of the new node or the PTR node at the beginning, using the terms mentioned in the image, that is PTR → next =head, and head =PTR. So, basically the first term, that is PTR → next = head, it means that the new node, that is pointer(name of new node which we want to insert) node's next part, that is the right-hand side part or the address part, is equal to head means is equal to 100. So, the 100 means it is the head part, as well as it is also the address part of the first node of the old link. So, if we establish the connection between the address part of the new node(ptr node), it will get connected to the old link's first node(Greeen | orange part). If it is connected to the old link's first node, and in the second part, head is equal to PTR means the address of the head gets connected to the PTR node. So, , the head part's address 100 will be the address of the pointer (new node) 100, and the pointer's address will get transferred to the next node. So, basically this is the way we connected.*

Refined version: ✅ Cleaned & Refined Explanation:

In a linked list, the head holds the address of the first node, the first node holds the address of the second node, and this continues until the last node, which holds NULL.

Looking at the image and understanding the logic behind inserting a new node (called PTR) at the beginning, we use two key statements:

- Step 1:

```
PTR->next = head;
```

This means:

- The next part (right-hand side) of the new node (PTR) will store the address currently held by head.

- If head is holding address 100, which is the location of the old first node, then now:
    - PTR->next = 100
    - This connects the new node to the old first node.

🔹 Step 2:

head = PTR;

This means:

- The head pointer is now updated to store the address of PTR, i.e., the new node.
- Now the new node becomes the first node in the list.

📌 Conceptually:

- First, we link the new node to the existing list (PTR->next = head).
- Then we update the head to point to the new node (head = PTR).

🧠 Why this matters:

If we reverse the order — if we write head = PTR before connecting PTR->next = head, the original list starting from address 100 would be lost, and we wouldn't be able to access it anymore.

So we must first link the new node to the list, and then update the head.

🔁 Final Simple Summary:

1. PTR->next = head; → Connect new node to the list
2. head = PTR; → Make new node the first node

That's the correct and safe logic for inserting at the beginning of a singly linked list.

------SINGLY LINKED LIST-----

User: *Head is equal to pointer means we are assigning the new node pointer as to head just like in mathematics a is equal to 2 means we are assigning number 2 to a, a is equal to b means we are assigning the variable b to a we should understand this and the way to say so head is equal to ptr means we are assigning the pointer node to the head node so that the connection between the pointer node and the first node is first established and the head node and the pointer node is established and this is how the node is inserted at the beginning*

Refined Version: ✅ Refined & Correct Explanation:

When we write:

```
head = PTR;
```

We should understand this just like in mathematics. For example:

- a = 2 means we are assigning the value 2 to the variable a.
- Similarly, a = b means we are assigning the value of variable b to variable a.

In the same way, when we say:

```
head = PTR;
```

It means we are assigning the address of the new node (PTR) to the head. This makes PTR the new first node of the linked list.

But before doing this assignment, we must connect the new node to the existing list:

```
PTR->next = head;
```

This means the next part of the new node (PTR) will now point to the old first node (which head was pointing to earlier).

So the two steps are:

1. Connect the new node to the current list:
   PTR->next = head;
2. Update the head to point to the new node:
   head = PTR;

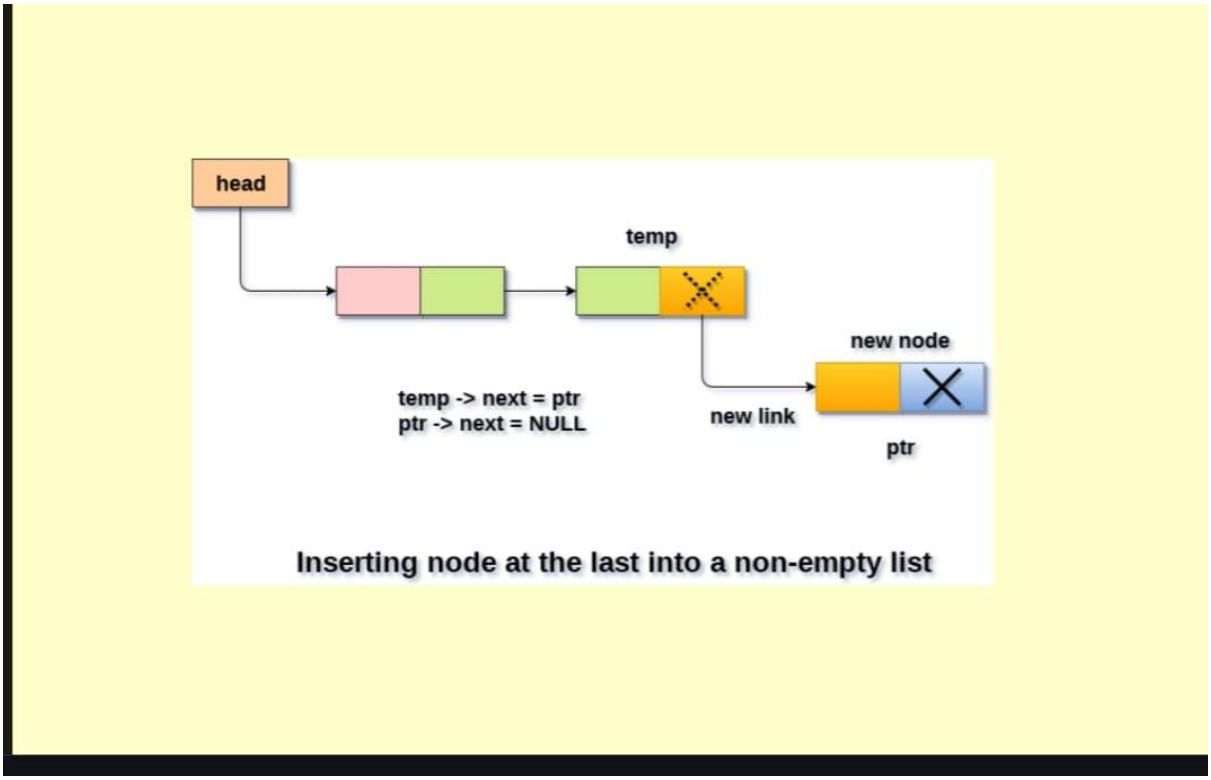
This ensures:

- The new node is linked to the old list, and
- The head now points to the new node, making it the first node in the list.

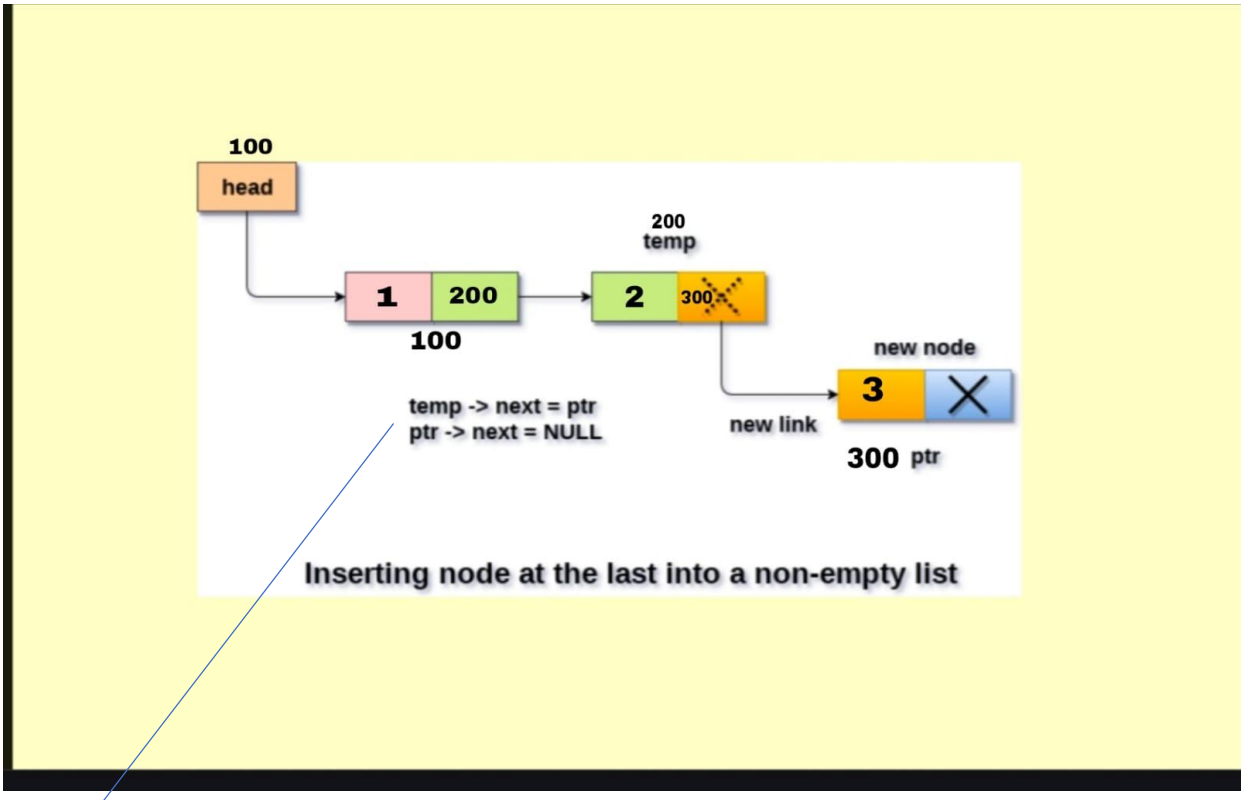
🧠 Key Understanding:

head = PTR; is not just a phrase — it's a memory-level operation. We are saying:
"From now on, the starting point (head) of our list will be this new node (PTR)."

Inserting a node at the last of Linked List:



------SINGLY LINKED LIST-----



Notes/Logic: *Here, temp → next = ptr means assigning ptr value to temp →next, and ptr → next = null means assigning null value to ptr →next, or in other words assigning the ptr value to the address part of the temp(orange part of temp), and assigning the null value to the address part of the newnode (ptr)or blue part. And here, ptr means the address of the new node, which is 300 as per our example. Ptr is not the name of the new node. The name of the new node to be inserted is new node only, and the address of that particular new node is ptr.*

User: *So initially our connection was like, first we had head, then first node, then second node which is named as TEMP. So initially ,TEMP node had a right-hand side or the address part or the orange part set as NULL, since it didn't had any other further node connected after that so it was null. So our main target is to enter a new node after the TEMP node, so that the NULL value of the TEMP is no longer NULL, it will have the address of the new node which is PTR, and*

the new node's next part or the right-hand side part or the blue part becomes NULL now. This is how the connection is established.

Refined Explaination : ☑ Refined Explanation:

Initially, we have a singly linked list that contains at least two nodes:

- The head pointer points to the first node (with data 1).
- The second node (with data 2) is currently the last node in the list, and its next pointer (the orange part) is set to NULL, indicating the end of the list.

We want to insert a new node (with data 3) at the end of this list.

🔧 Steps to insert:

1. Traversal (finding the last node):
    - A temporary pointer, usually called temp, is used to traverse the list.
    - We keep moving temp to the next node using temp = temp->next until we reach the node whose next is NULL.
    - In the diagram, temp ends up pointing to the second node (with data 2).
2. Creating the new node:
    - A new node, pointed to by ptr, is created.
    - Its data field is assigned (e.g., 3), and its next field is initialized to NULL, since it will become the new last node.
3. Updating the links:
    - Now, we connect the current last node (temp) to the new node (ptr) by setting temp->next = ptr.
    - This removes the NULL from temp->next and replaces it with the address of the new node.
    - ptr->next remains NULL, as it's the new last node.

------SINGLY LINKED LIST-----

🧠 Key Concepts Reinforced:

- The orange part of a node is the link field or next pointer.
- The blue part (in the new node) is also a next pointer, and since this is the last node, it is NULL.
- This process ensures that the new node is correctly appended to the end of the list.

🔍 Small Fix in Your Words:

"TEMP node had a right-hand side or the address part or the orange part as NULL" → ☑ Correct, but you can say:
"Initially, the temp node (last node) had its next pointer (the orange part) as NULL."

"the NULL value of the TEMP is no longer NULL" → ☑ Instead, say:
"The NULL in temp->next is updated to point to the new node (ptr)."

User : *Here in the image the text temp next equal to ptr means we are assigning the ptr value to temp next means that we are assigning the address of the new node which is ptr to the temp next value the temp next here is the orange part which was initially null so once it is assigned with the ptr address the temp temp next is no longer null it has the address of the ptr and then in the next step we assign null value to ptr next that is ptr next means the new node's next value or the right hand side value or the blue part we are we made it null that means it is the end of the node and it has no longer connected to any other node so this is how the temp is inserted that is how the new node is inserted at the last of the linked list*

Refined: ✅ **Refined Version of Your Explanation**:

In the image, the line:

temp->next = ptr

means we are **assigning the address of the new node (ptr)** to the next field of the temp node.

- The temp->next is the **orange part** in the diagram, which was initially NULL because temp was the last node in the list.
- After this assignment, temp->next no longer holds NULL — instead, it holds the **address of the new node**, effectively connecting temp to ptr.

Next, we execute:

ptr->next = NULL

- Here, ptr->next refers to the **next pointer of the new node** (the **blue part**).
- We assign it NULL to indicate that this node is now the **last node** in the linked list and doesn't point to any further node.

This is how the new node is successfully **inserted at the end** of a non-empty singly linked list.

💡 **TL;DR Summary (in simple terms):**

- temp->next = ptr: Connect the last node to the new node.
- ptr->next = NULL: Mark the new node as the new last node.

Term	Represents	Example
ptr	Pointer holding address of node that we are to insert at last (i.e address of new node)	Let's say: ptr = 300
New node	The actual node being created and inserted at last	Contains: data = 3, next = NULL
temp	adress of initial last node before insertion of new node	Let's say: temp = 200

------SINGLY LINKED LIST-----

Term	Represents	Example
temp->next	The next field(address filed) of node at temp	Was NULL, becomes 300

Conclusion : Here, when we say:

temp->next = ptr

It means we are **assigning the value of ptr** (which is the address of the new node) to temp->next.

In simple words:
➡️ We are connecting the last node (temp) to the new node by putting the address of the new node into temp's next part — the orange part.
So, now the orange part of temp is **no longer NULL**. It holds the address of the new node.

Next, we write:

ptr->next = NULL

This means we are **assigning NULL to ptr->next**.
In other words:

➡️ The new node's address part (its blue part) is set to NULL, meaning it's now the **last node** in the list and does not point to anything else.

Now, a very important thing to remember:

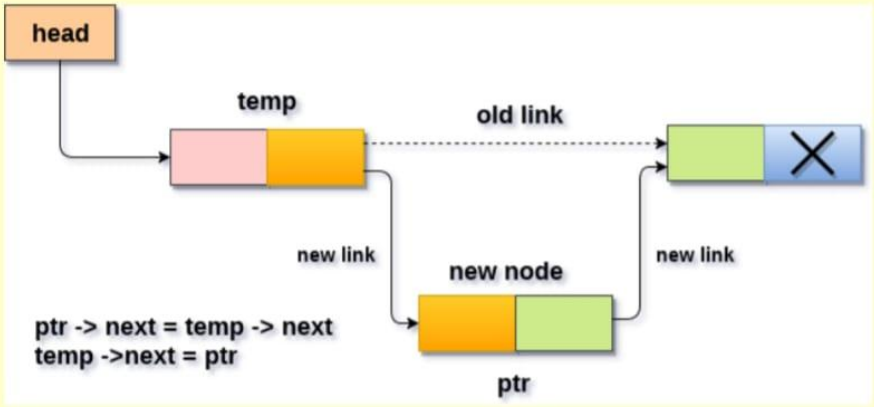
- **ptr is not the name of the new node.**
  ptr is just a pointer — it **holds the address** of the new node (like 300 in our example).
- The **new node** is the actual memory block or structure we created to insert into the list.

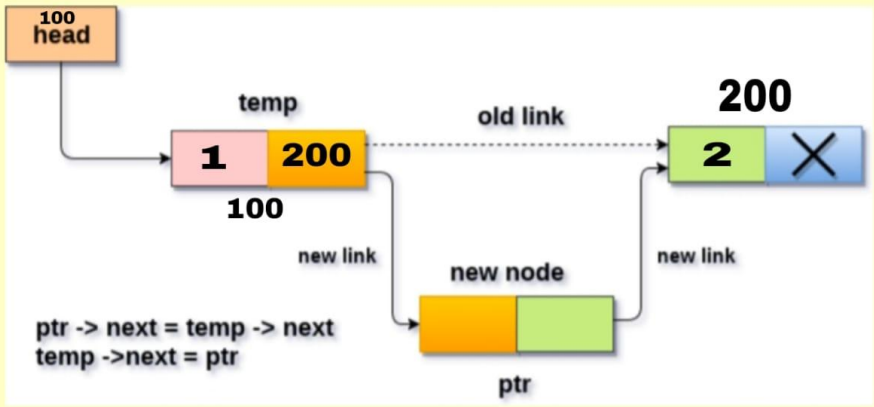
- So whenever we say ptr, we are **referring to the address of the new node**, not the node itself.

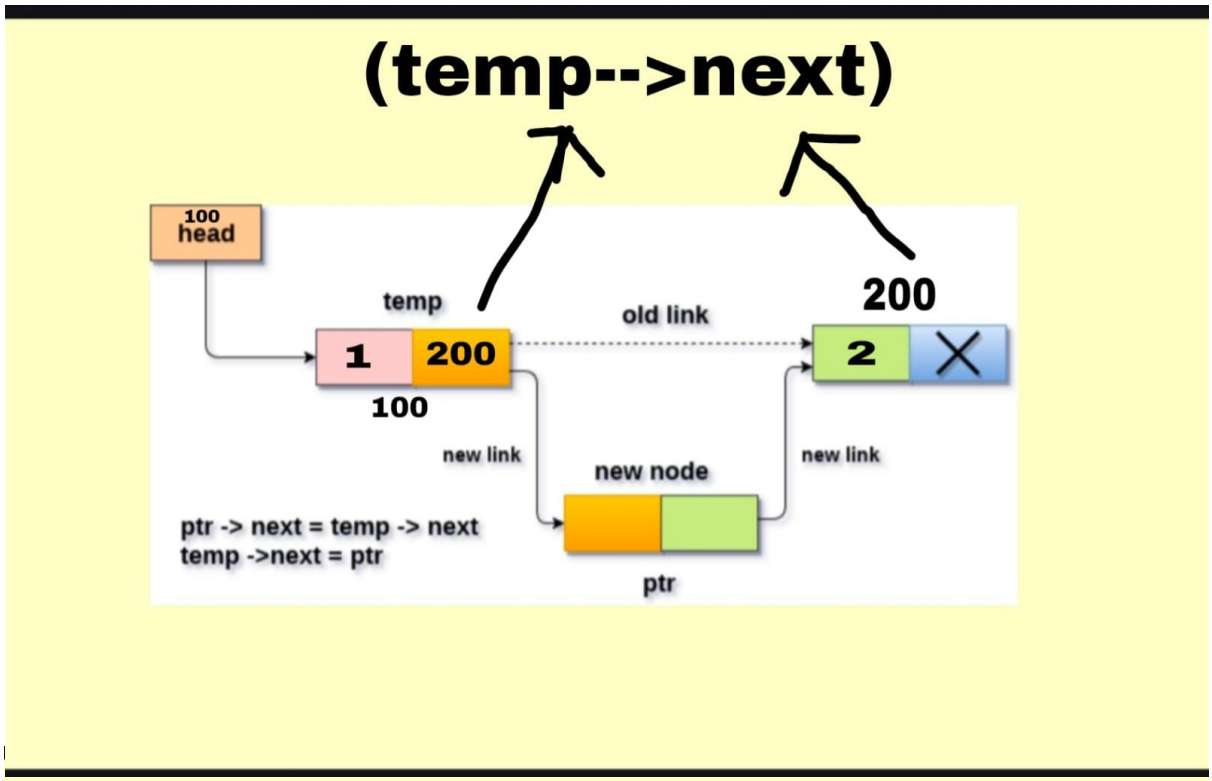
You can say:

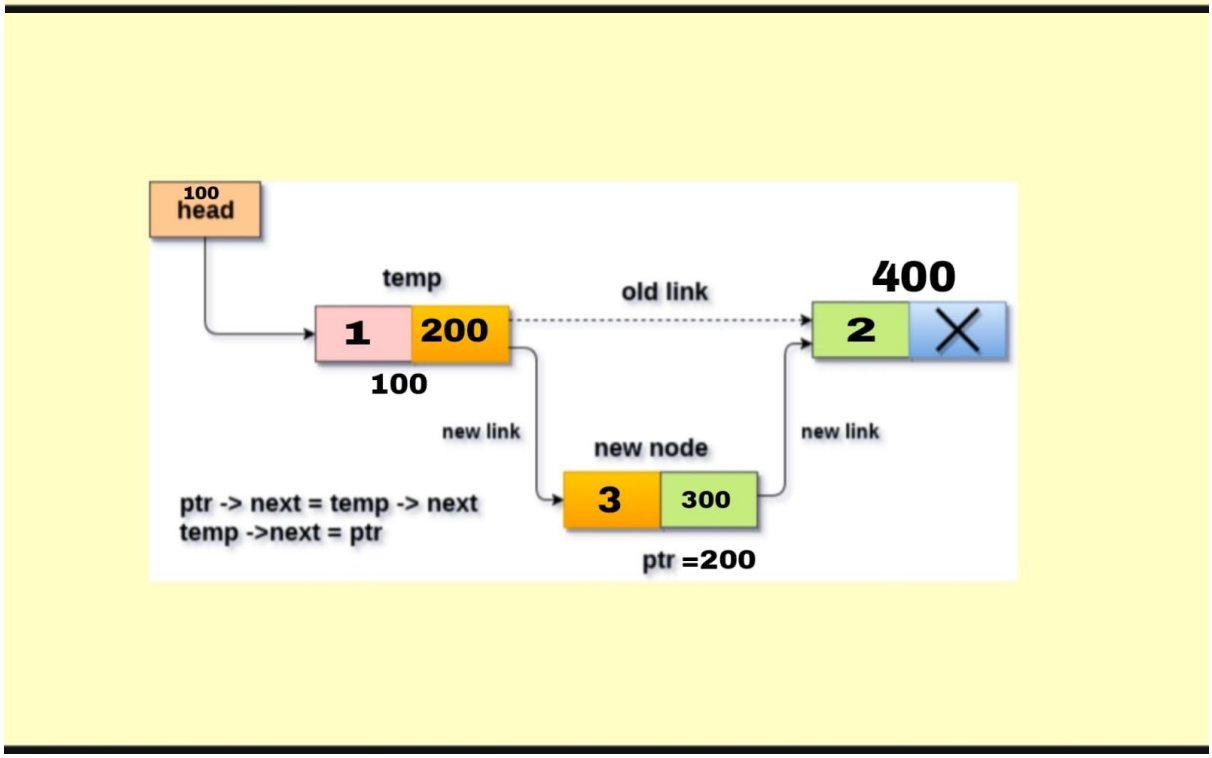
🟢 “The new node is the one being inserted, and ptr is the pointer variable that stores its address.”

Inserting a node at any particular position between two node :









USER: *To insert a new node between two nodes, we do it by traversing from head pointer to that particular node after which the new node is to be inserted. For example, we have eight nodes and we want to insert our new node at fifth position. So we will traverse from head pointer to our fourth position using temp variable and we will insert our new node at that particular position. Since like array, we can access the elements or the access each index of the array*

using indexing numbers but in case of linked list we don't have index so we have to do it using temp variable one after one and traverse to that particular point when new node is to be inserted. This is the first concept.

Refined: ☑ **What you got right:**

1. **Linked List Traversal:**
   - You correctly said that we cannot access a position directly like arrays (using indexing).
   - Instead, we **start from the head** and move one node at a time using a **temporary pointer (temp)** until we reach the node **just before** the insertion point.
2. **Position-based Insertion:**
   - Yes, to insert at position 5, you must stop at position 4 and perform the insertion there.

💡 **Analogy:**

3. Think of it like inserting a new person into a queue — you have to go to the person just before where you want to insert, and then tell the new person to hold hands with the next, and the previous person to hold hands with the new one.

USER: *So, by looking at the image, the initial linked list was, it was having HeadPointer, first node, and the second node. It was our old linked list. Now, we want to insert our new node, which is also named as PTR (new node name), in between the first node and the second node. We have to connect this. So, for this connection, the main logic is, PTR →next = TEMP → next, means, we are assigning the TEMP → next(address of temp) value to the PTR → next(adress of ptr). Here, TEMP →next is the address of the temporary node, which is 200. This 200 is also the address of the second node of the initial connection(initial linked list before insertion , or the old lnk). So, basically, by assigning the TEMP → next value to PTR → next, means, our PTR →next is getting connected to the second node (last node in diagram of linked list) So, when the connection between the PTR (new node name) and the last part of the linked list is*

established, then we will assign our PTR to TEMP → next, that is, our address of the new node will be stored in TEMP. This is how the connection is established. First, the connection is established from new node to the second node(last node) , and then the new node is connected to the TEMP(1st node after head) in reverse order connectivity is done , so that the address is not lost.

Refined: ☑ **What you explained correctly:**

1. **Initial State of Linked List:**
   - You started with a linked list of two nodes:
     - head -> first node (data=1, next=200)
     - second node (data=2, next=NULL)
   - That's 100 → 200 → NULL (old list), which is clearly shown in the image.
2. **Goal:**
   - Insert a new node (ptr) between the first and second nodes.
3. **Understanding ptr->next = temp->next:**
   - Spot on! This connects the new node (ptr) to the second node.
   - Since temp->next is pointing to the second node (address 200), assigning it to ptr->next makes ptr also point to node 2.
   - Now the new node is linked **forward** to the rest of the list — the first half of the job is done.
4. **Understanding temp->next = ptr:**
   - Beautifully said: this connects the **first node** to the new node (ptr), completing the insertion.
   - temp is currently pointing to the first node, so updating temp->next = ptr means:
     the first node now points to the new node.
5. **Order of Operations:**

- ☑ Yes, **first connect the new node to the second**, then connect the first node to the new — otherwise you risk losing the link to the rest of the list.

🔧 **Just a Small Refinement:**

You wrote:
"then we will assign our PTR to TEMP next, that is, our address of the new node will be stored in TEMP."

That line is a little **confusing**, because:

- We **do not** assign ptr to temp->next just to store it.
- We assign temp->next = ptr because we want the **first node** (where temp points) to point **to the new node**.

So instead, you can say:

"Then we assign temp->next = ptr to make the first node point to the new node."

☑ **Final Sequence Recap (in your style, with slight polishing):**

1. Traverse to the node **after which** the new node should be inserted (using temp).
2. **Connect new node to next node**:
   ptr->next = temp->next
   → New node now points to the second node (200).
3. **Connect previous node to new node**:
   temp->next = ptr
   → First node now points to the new node.

This ensures the chain remains unbroken:
Head -> 1 -> 3 (new) -> 2 -> NULL