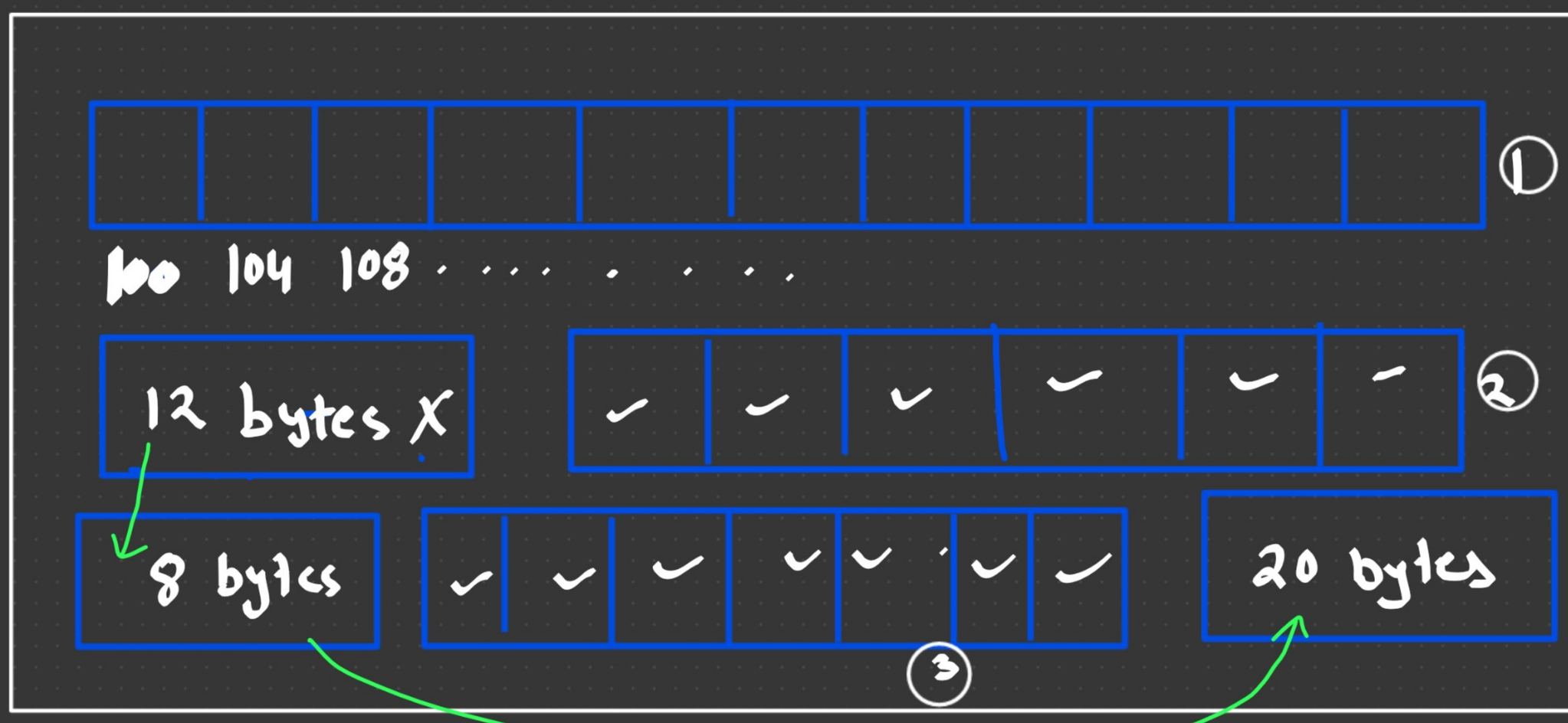
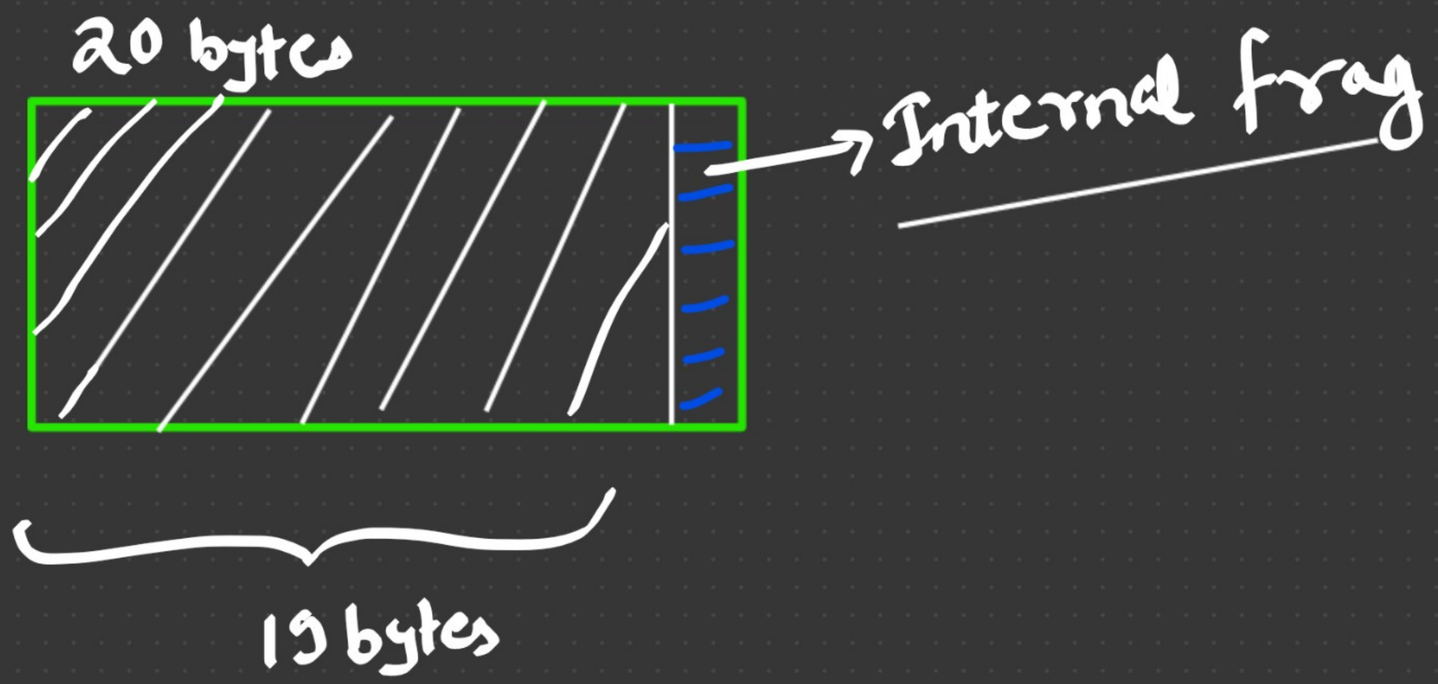
Linked list



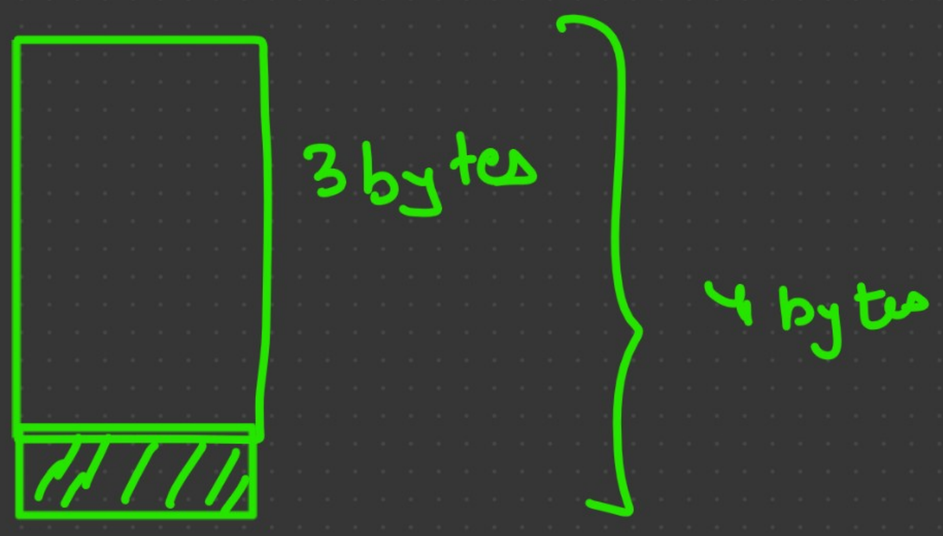
int a[10]; ⇒ 40 bytes ⇒ 20+12+8

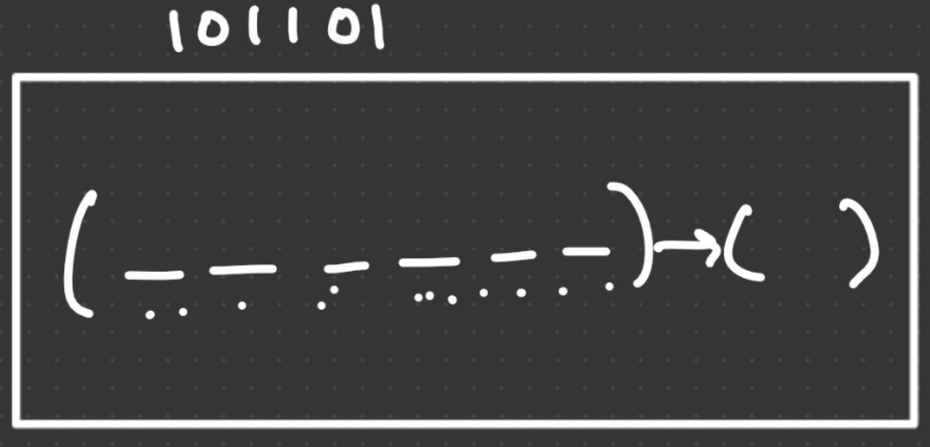
Fragmentation → Internal Frag. (X)
→ External frag. ✓



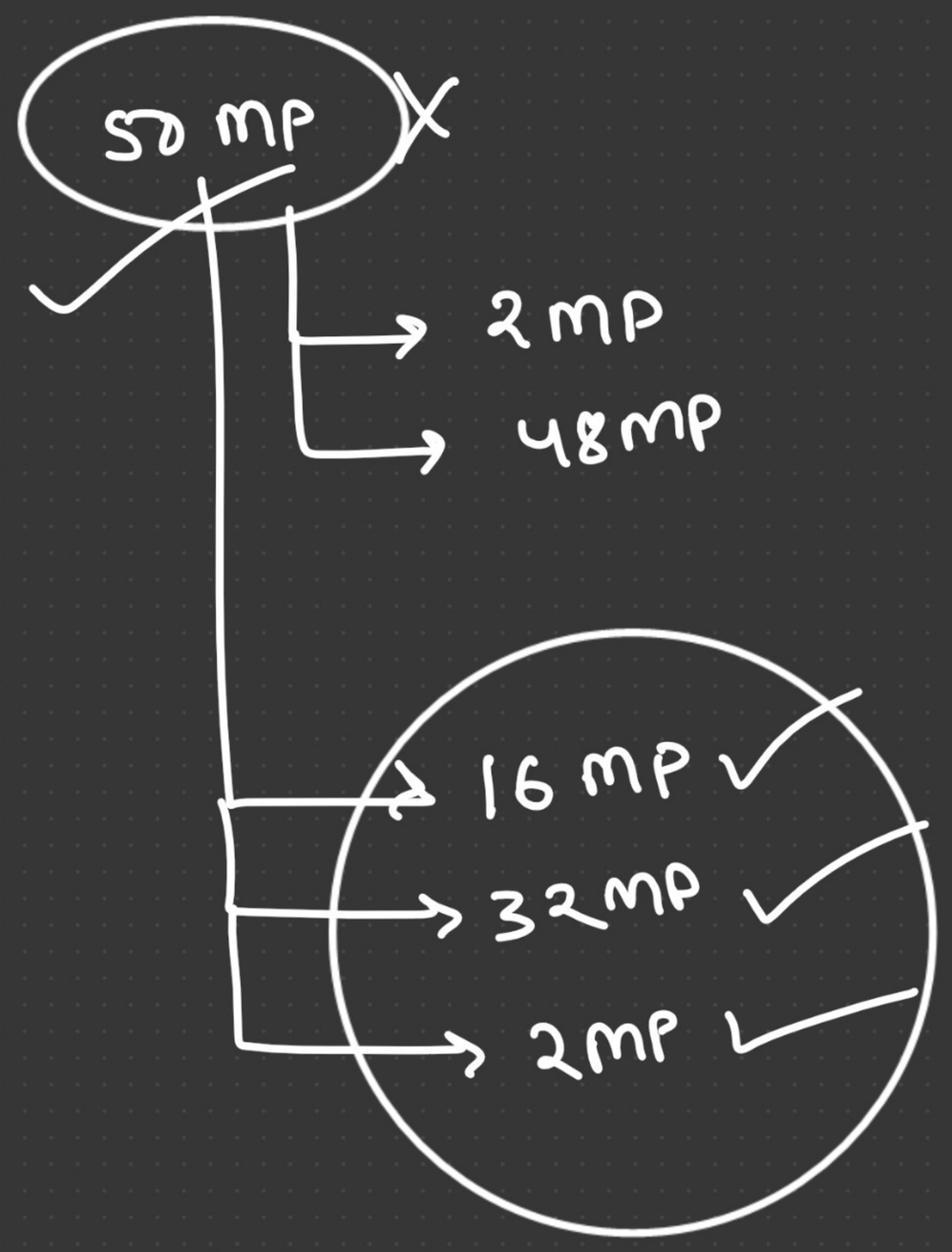
Memory Allocation :- 2^0 bytes, 2^1, 2^2, 2^3, ---- 2^n

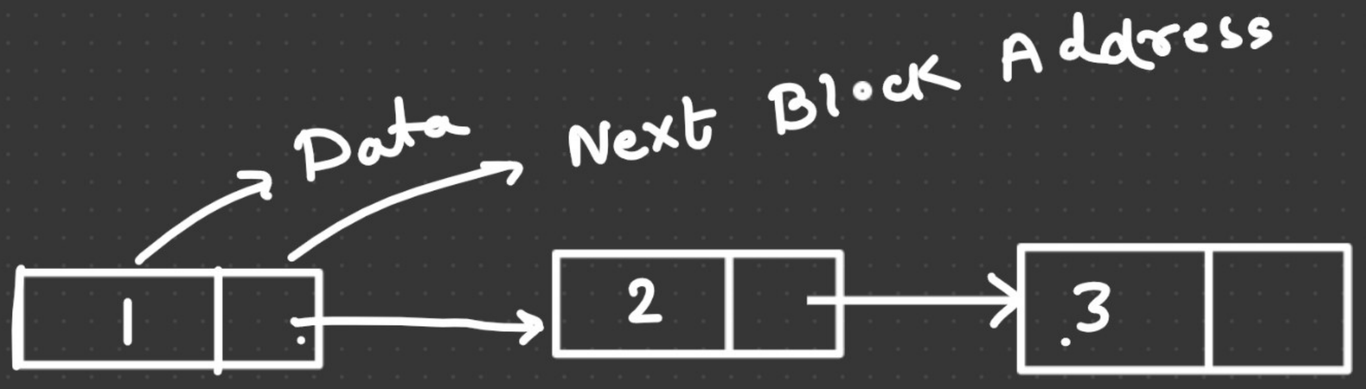
1, 2, 4, 8, 16, - - - -

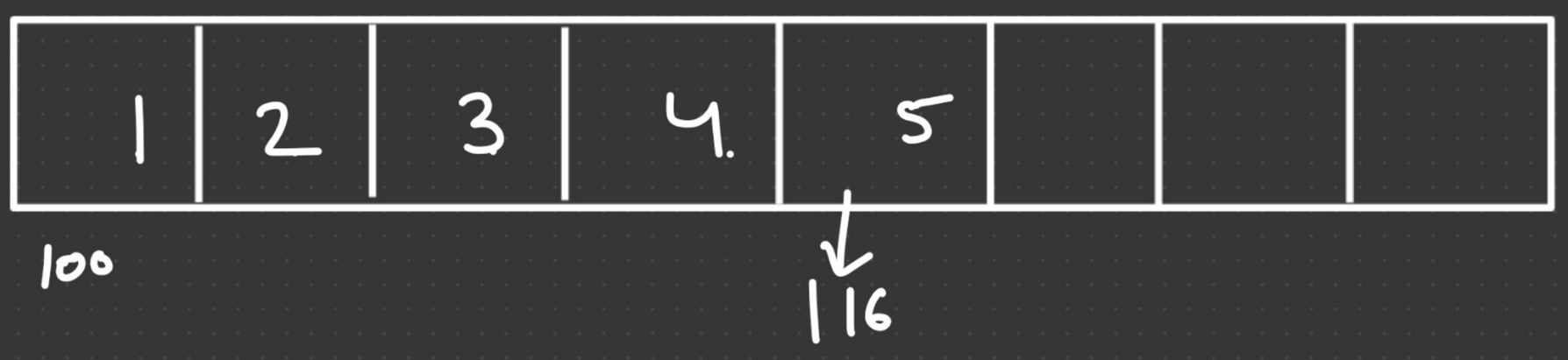


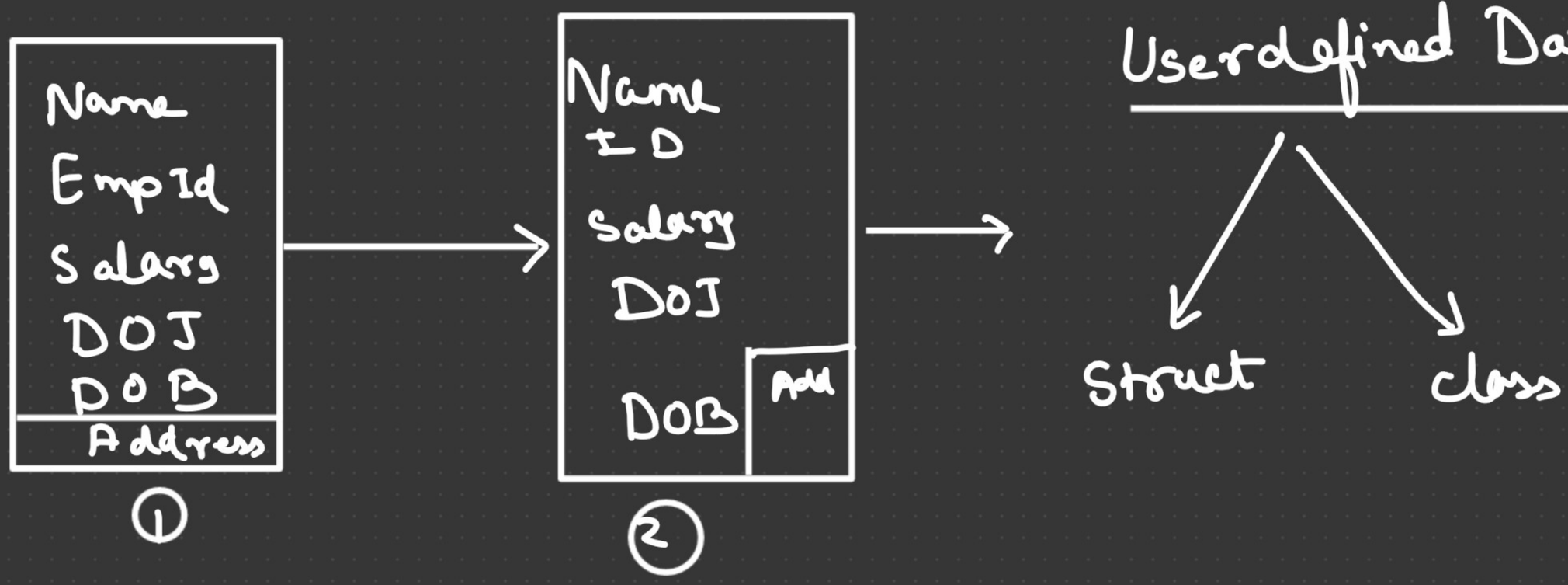


2 MP, 8 MP, 16 MP.

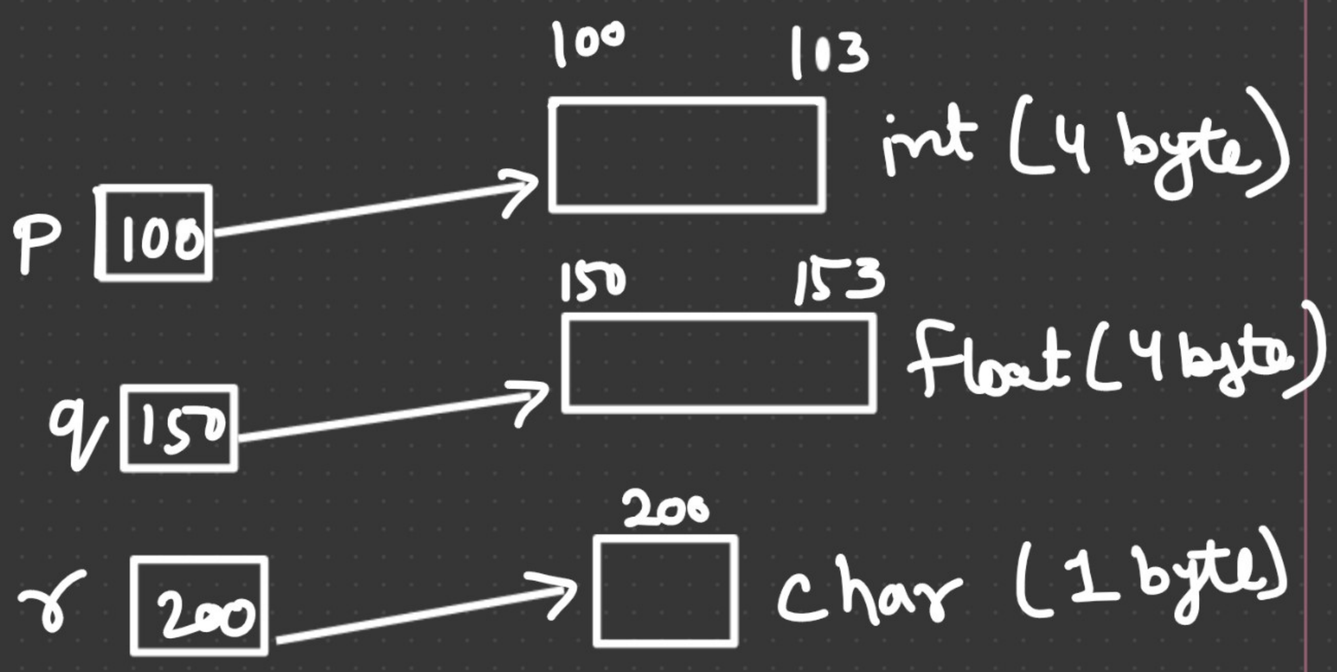

50 MP
2MP
48MP
16 MP
32MP
2MP


Data
Next Block Address
1
2
3


1
2
3
4.
5
100
116


Name
EmpId
Salary
DOJ
DOB
Address
1
Name
ID
Salary
DOJ
DOB
Add
2
Userdefined DataType
Struct
class


```
int *p;
float *q;
Char *r;
```




```
class node
{
   int value;
   node *next;
   node()
   {
    next = NULL;
   }
};
```

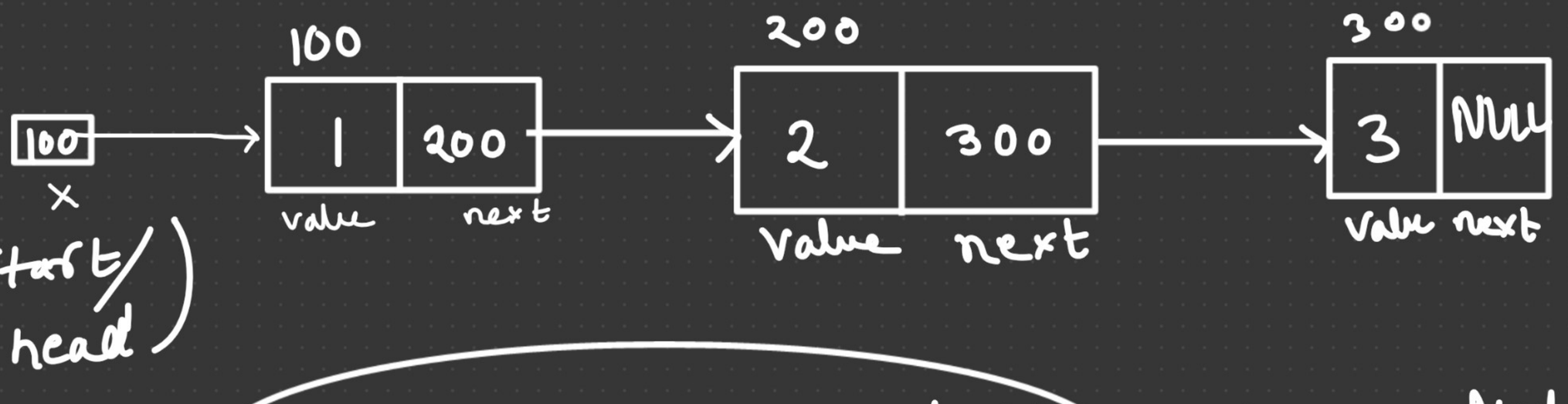
1) Create node
2) Insert new node
3) Print.

```
class linkedlist
{

  public:
        create node();
        Insert (value);
        print( );

};
```

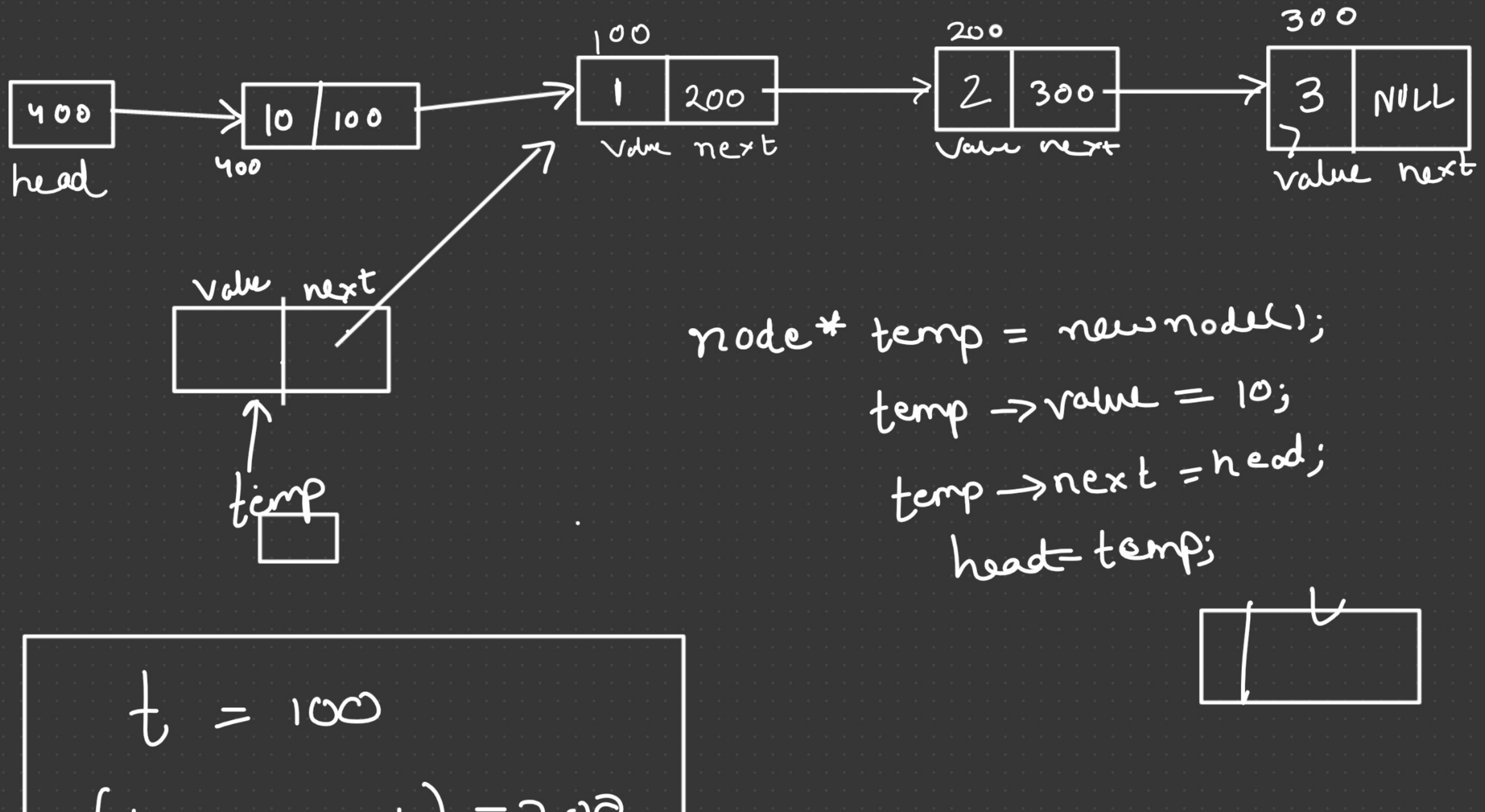
node *x = new node(); ⟶ 1) Create node



node *y = new node(); ⟶ new Node created

repeat

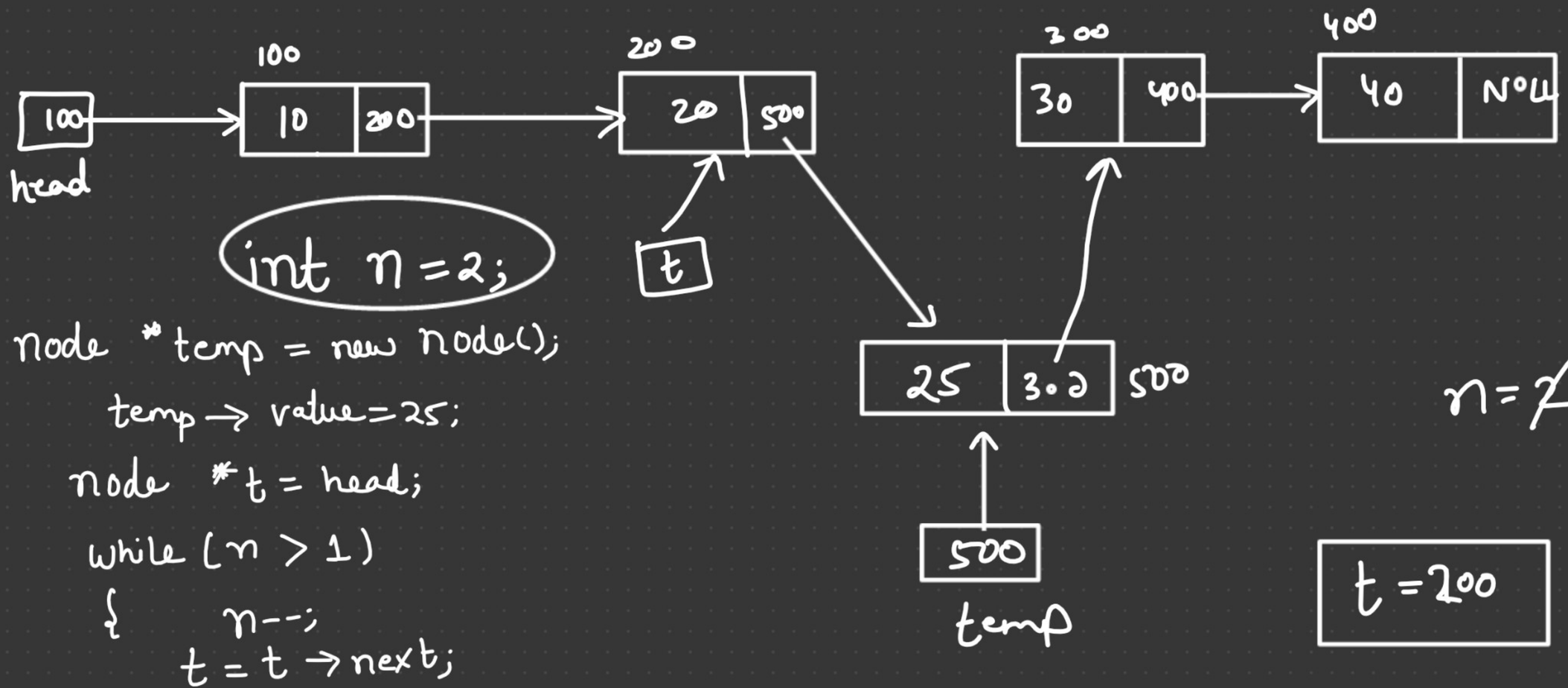
x → value = 1;	y → value = 3;
y → value = 2;	x → next → next = y;
x → next = y;	



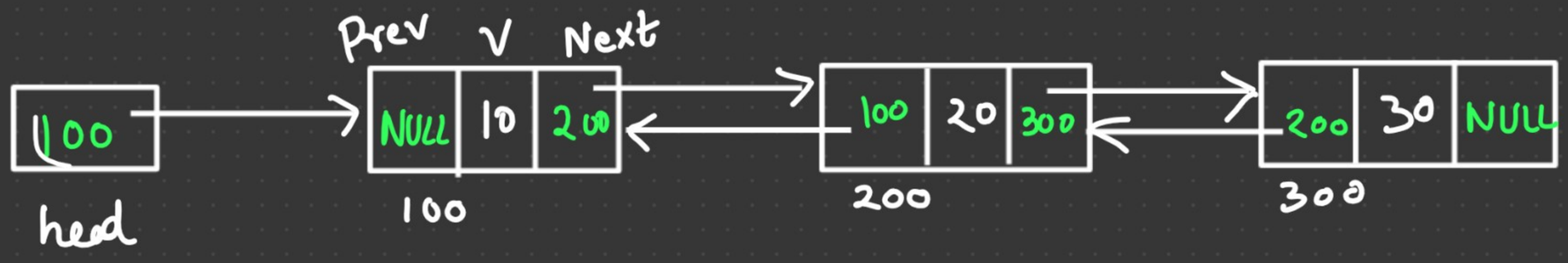

```
node* temp = new node();
temp -> value = 10;
temp -> next = head;
head = temp;
```

t = 100

(t → next) = 200



Doubly Linked link

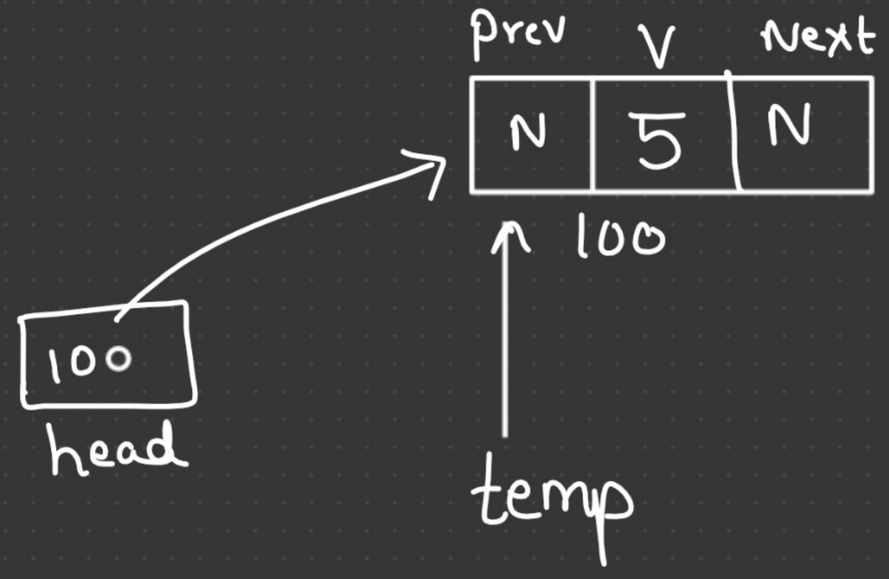



```
Class node
{
    int value;
    node * next;
    node *prev;
};
```

1) Create → end
2) Insertion begin/middle
3) Deletion → begin/end/middle
4) Search / Traverse

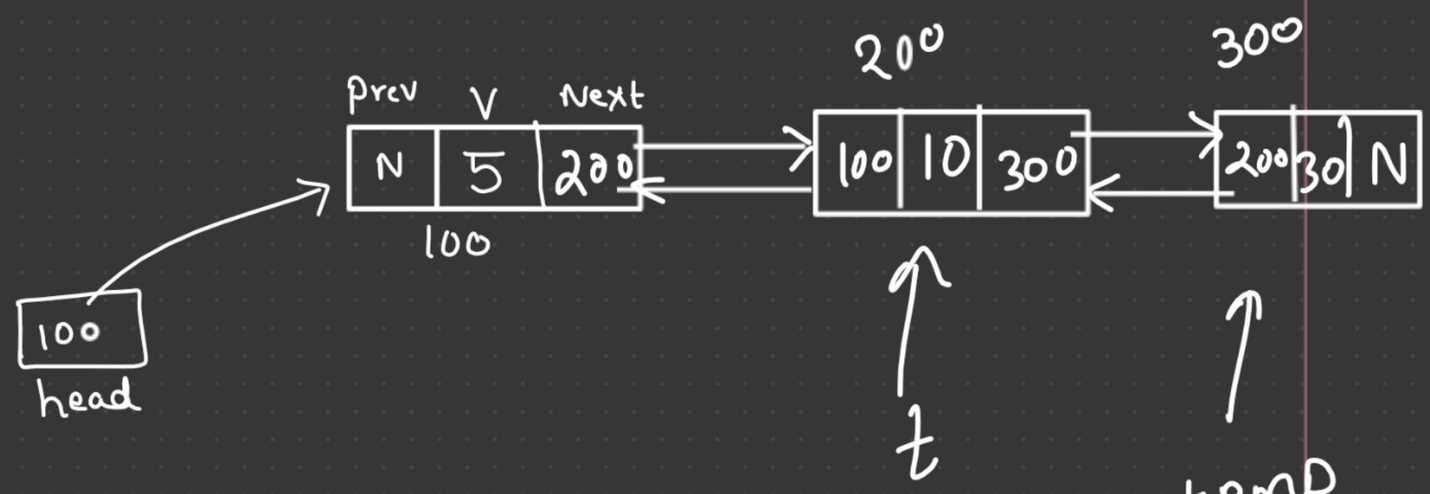
```
node()
{   value =0;
    next = NULL;
    prev = NULL;
}
```

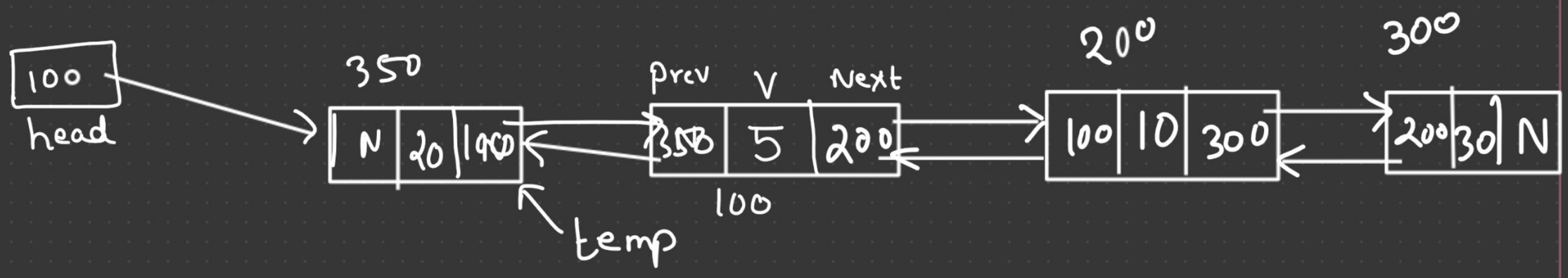
```
class DLL
{
   node *head;
 public:
        DLL() { head=NULL;}
        Create(){    }
          :
          :
};
```




```
DLL dl;
dl.create(V);
```

```
create(int v)
{
  node *temp = new node();
        temp->value =V;
  node * t;
if (head == NULL)
{
    head = temp;
 }
else
{
   t = head;
   while (t->next !NULL)
   {   t = t->next; }
   t->next = temp;
    temp->prev = t;
}
```





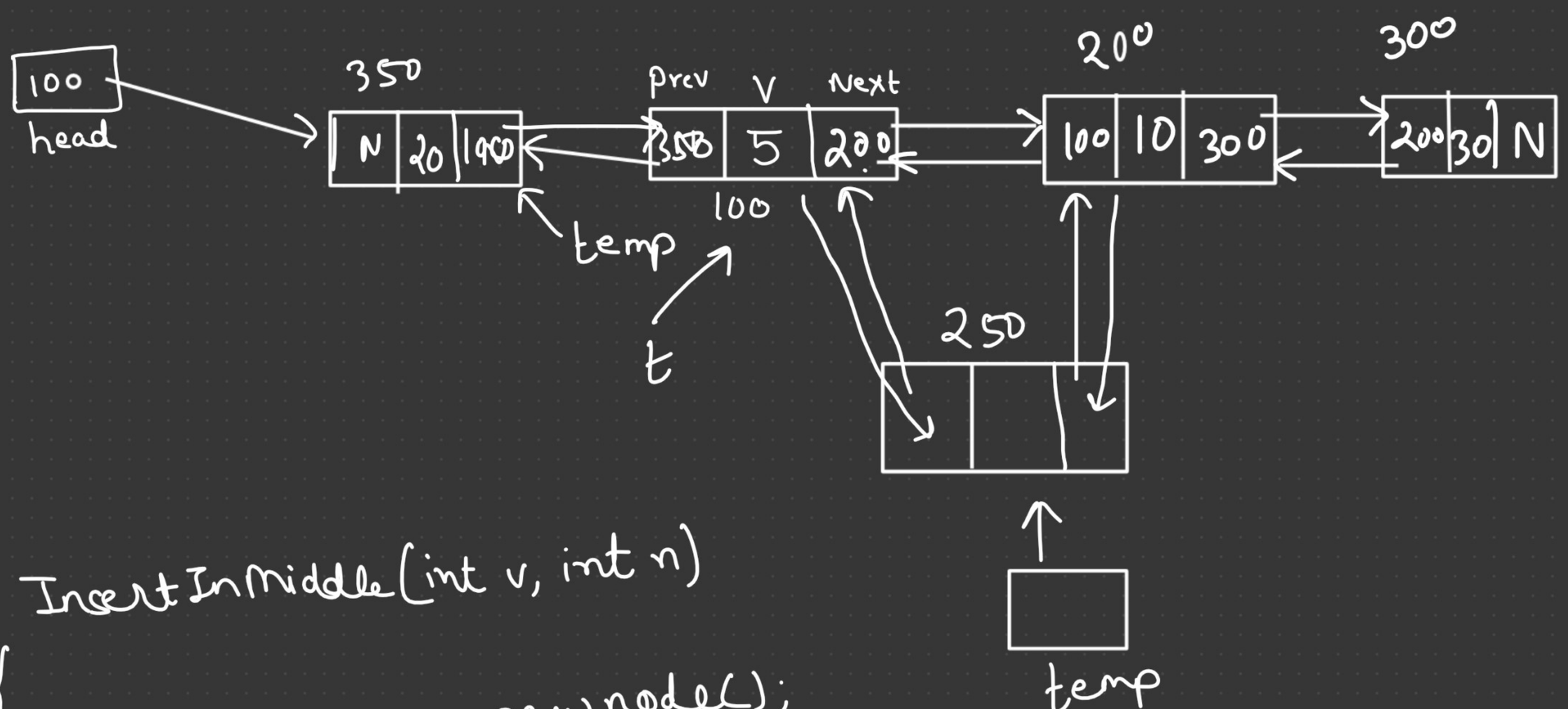
insertAtBegin(int V)

```
{
    node *temp = new node();
      temp->value = V;

    if (head == NULL)
     {   head= temp;
      }
    else
    {
        temp->next = head;
        head->prev = temp;
        head = temp;

    }

}
```



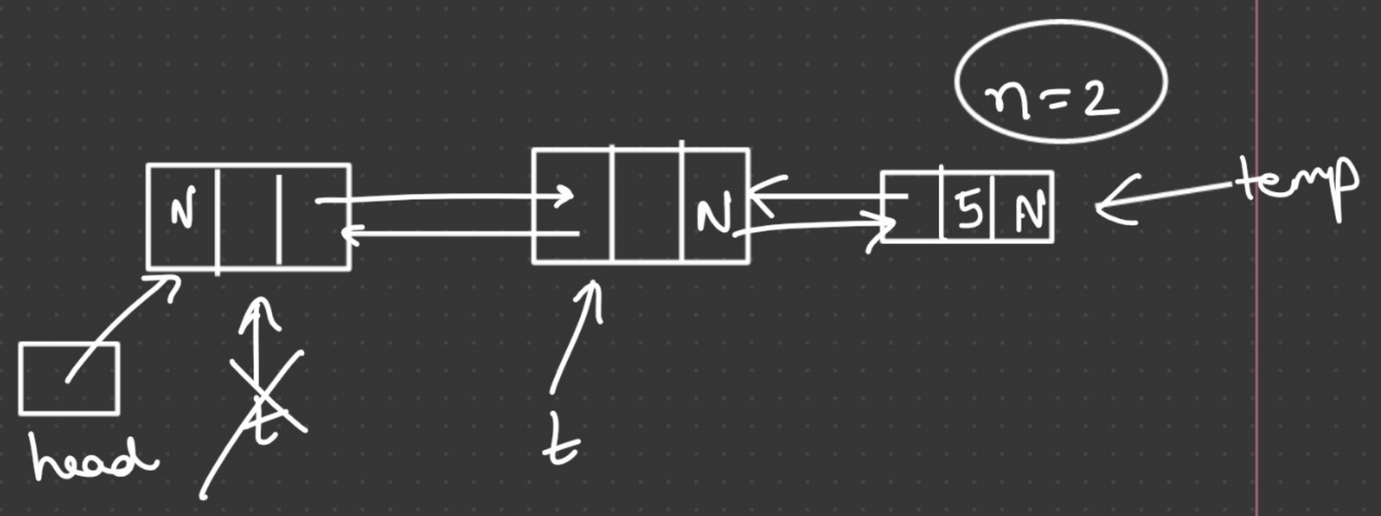
InsertInMiddle(int v, int n)

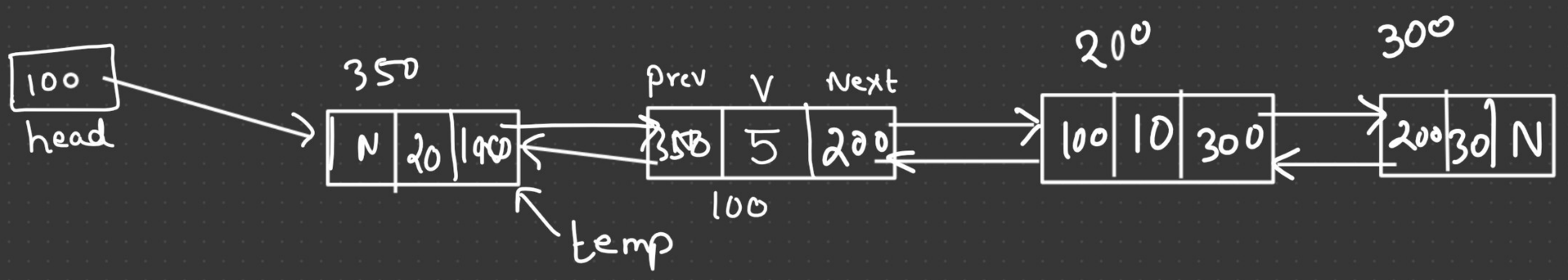
```
{
    node * temp = new node();
        temp -> value = v;

    node * t = head;

    while (n > 1)
    {       n--;
        t = t -> next;
    }

    temp -> next = t -> next;
    temp -> prev = t;
    if (t -> next == NULL)  t -> next = temp
    t -> next -> prev = temp;
     t -> next = temp;
}
```

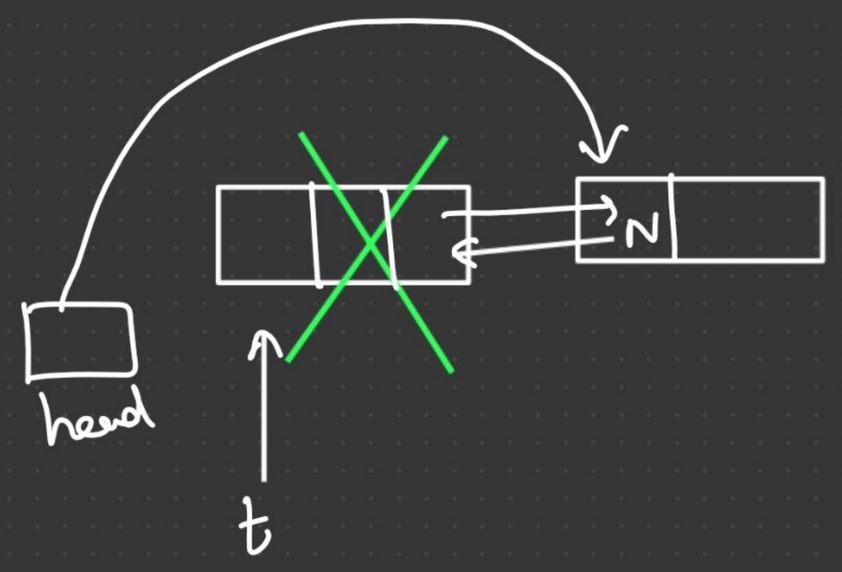


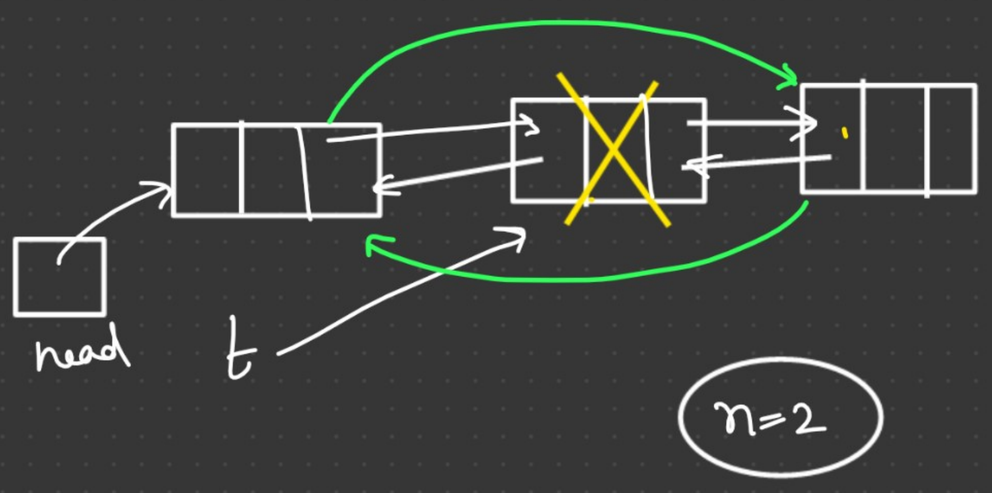


Deletion :-

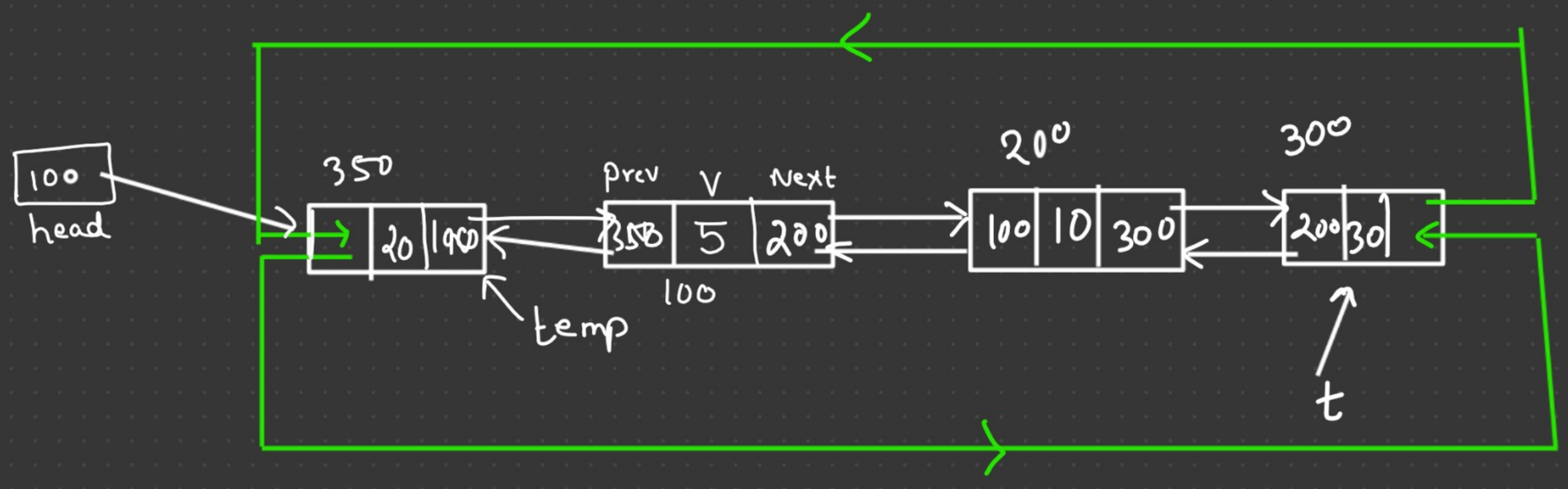
// Deletion of 1st Node

```
DeletionDLL(int n)
{
    node *t = head;
    if (head == NULL)
    { cout<<" Position Not Found";
      return;
    }
  if (n == 1)
  {
    head = t->next;
    head->prev = NULL;
    delete t;
    return;
  }
while (n>1)
{     n--;
   t=t->next;
}
   t->prev->next = t->next;
(  t->next->prev = t->prev;)
   delete t;
```





Circular DLL:-



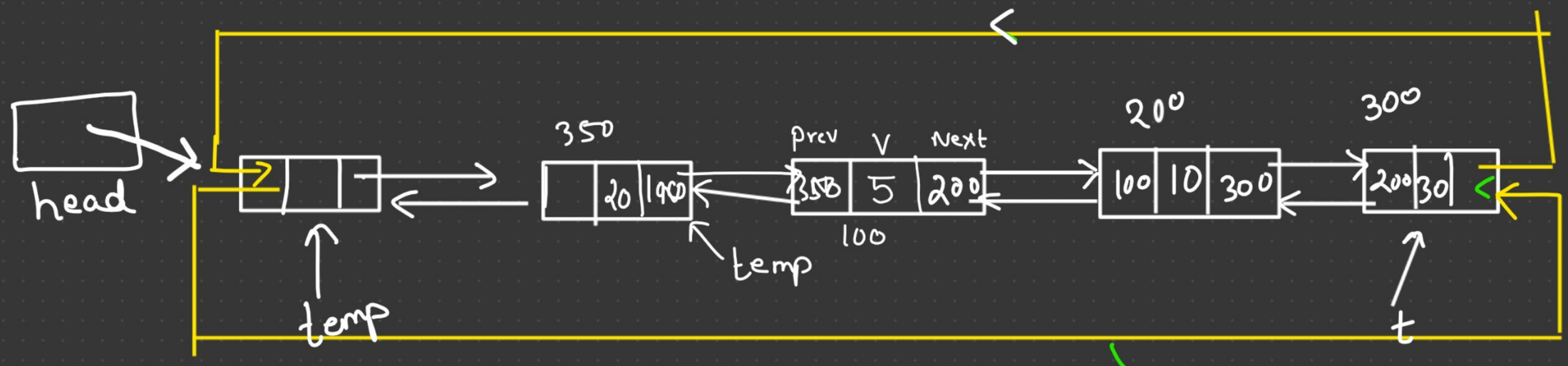
①

```
node *t = head;
while( t != NULL)
{
   t = t ->next;
}
```

} t will point to last node

```
while ( t->next != head)
{
   t = t -> next;
}
```

② Insert at Begin:-



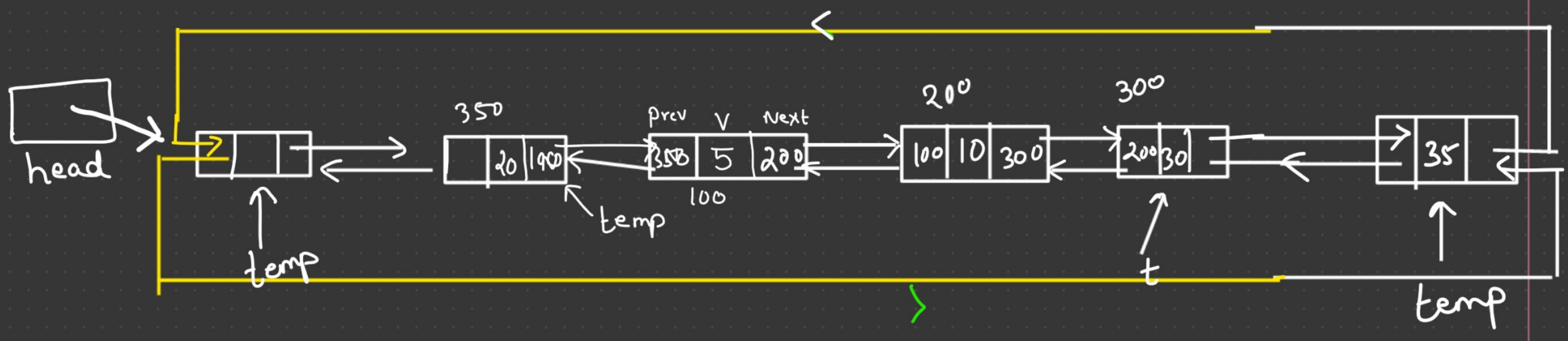

```
else
{
   temp->next  = head;
   temp -> prev = head->prev;
    head = temp;
   temp->next->prev = temp;
   temp->prev->next = temp;
}
```

Last Node

insert at Begin

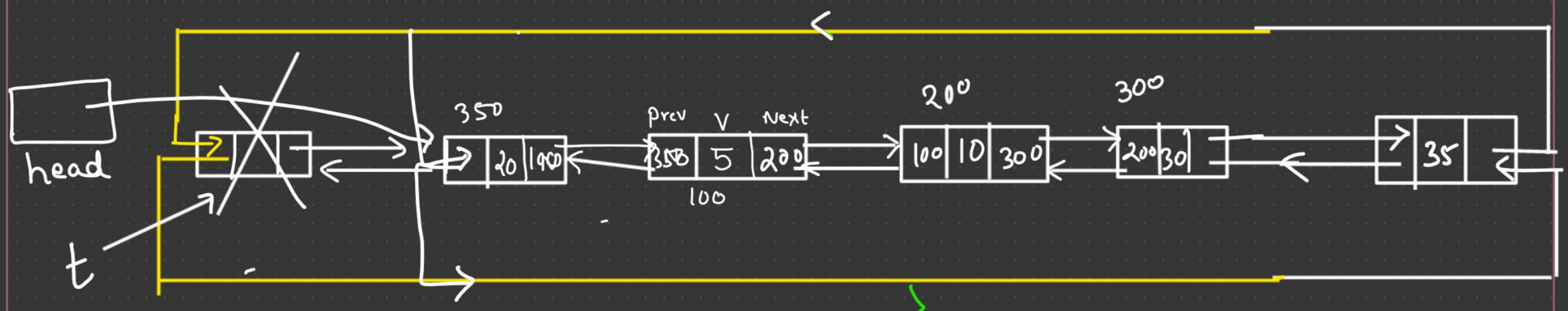
```
node * temp = new node();
     temp->value = V;

if (head == NULL)
{      head = temp;
       temp->prev = temp;
       temp-> next = temp;
}
else
{
    temp->next  = head;
    temp -> prev = head->prev;
     head = temp;
    temp->next->prev = temp;
    temp->prev->next = temp;
{
```



Insert at the End:-

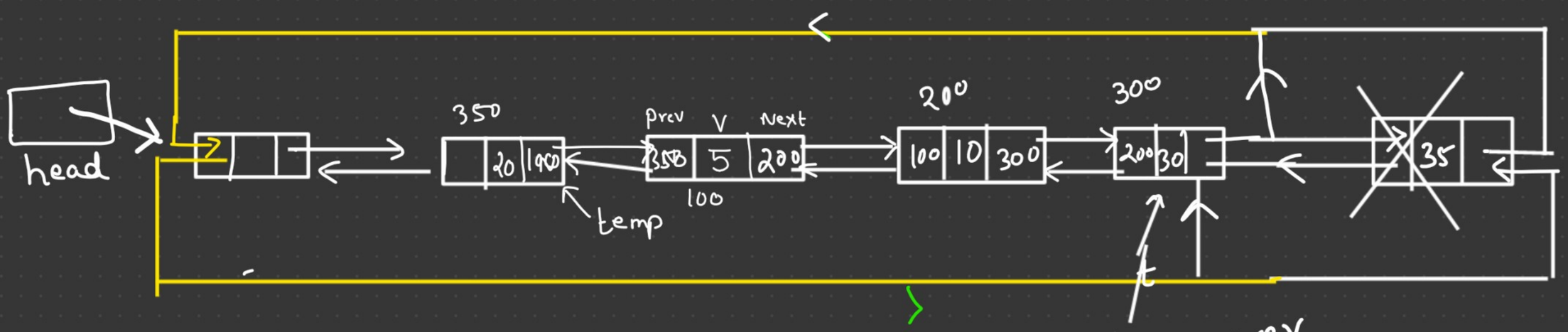
```
temp->next = head;
temp->prev = head->prev;
head->prev->next = temp;
head->prev = temp;
```



Deletion from begin :-

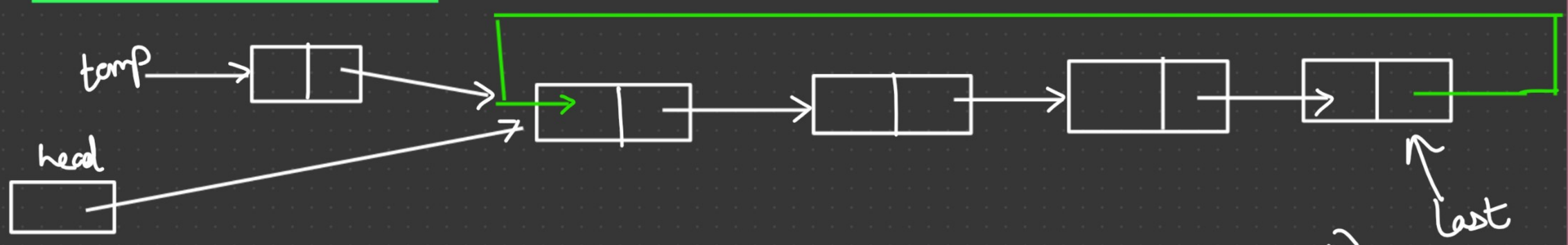
```
node *t = head;
head = t->next;
head->prev = t->prev;
t->prev->next = head;
delete t;
```

Deletion from end:-




```
node *temp = head->prev;
temp->prev->next = head;
head->prev = temp->prev;
delete temp;
```

Circular SLL:-



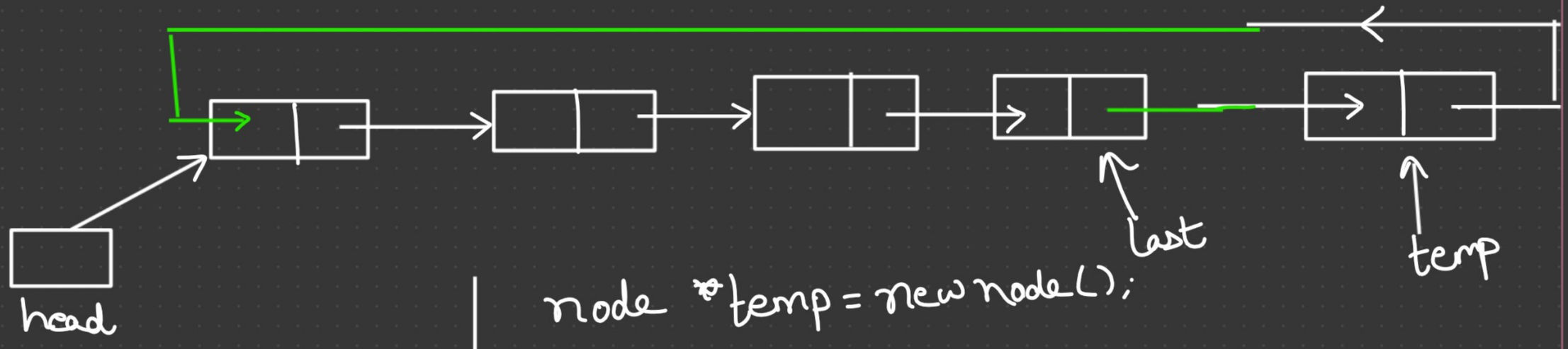
while (t != NULL) ⟶ while (t->next != head)

Insertion at Begin :-

```
node *last = head;
while (last -> next != head)
{
  last = Last -> next;
}
```

```
node *temp = new node();
temp -> value = v;
temp->next = head;
last -> next = temp;
head = temp;
```

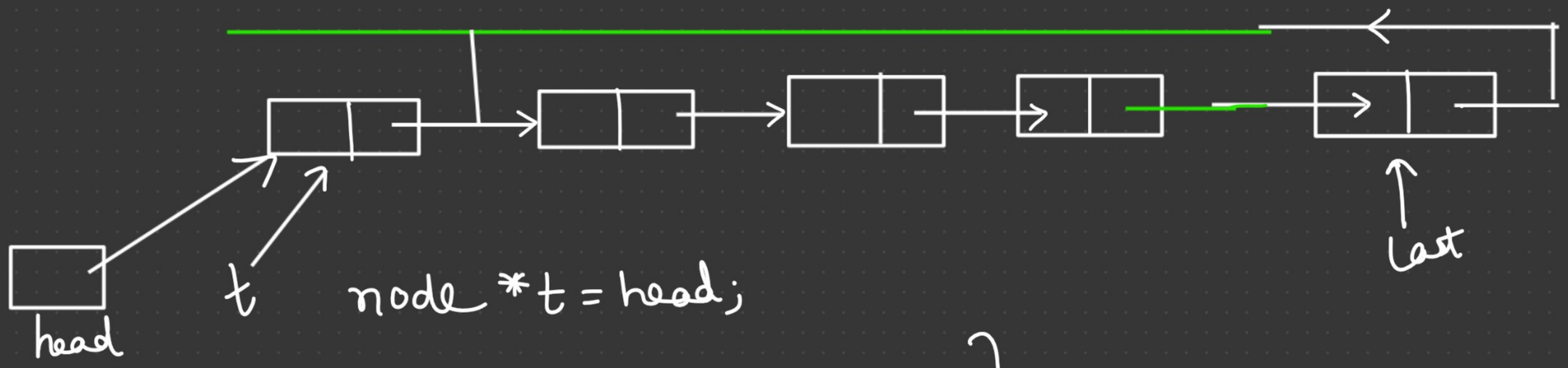
Insert at End:-




```
node *last = head;
while (last -> next != head)
{
  last = Last -> next;
}
```

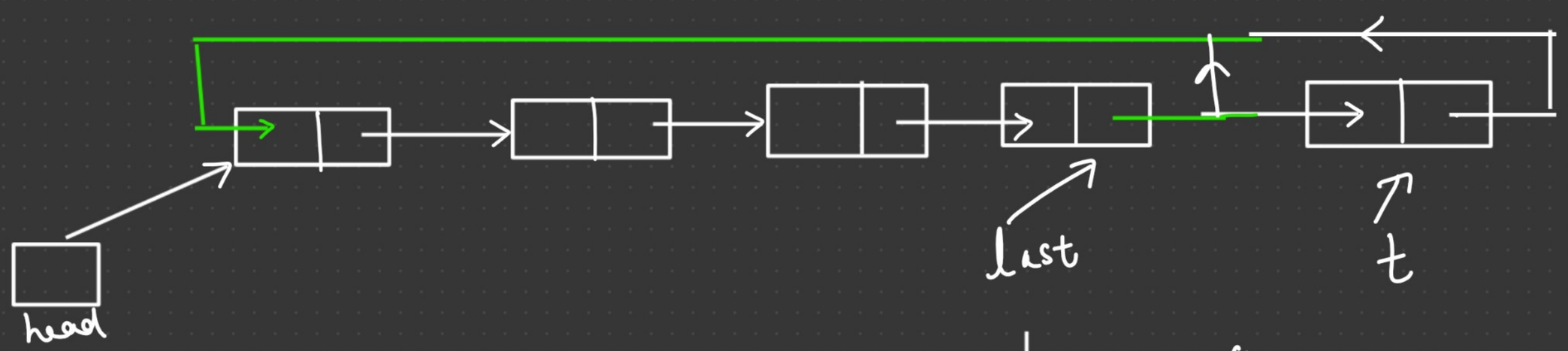
```
node *temp = new node();
temp -> value = v;
temp -> next = head;
last -> next = temp;
```

Deletion at the Begin:-




```
node *t = head;
last -> next = t -> next;
head = t -> next;
delete t;
```

Delete at the end:-




```
node * t = last -> next;
last -> next = head;
delete t;
```

```
while (last -> next -> next != head)
{
    last = last -> next;
}
```