Data Structure & Algorithms

Sunbeam Infotech

Parenthesis Balancing

- 5 + ([9 – 4] * (8 – { 6 / 2 }))

5 + 9 - 4 * 8 - 6 / 2

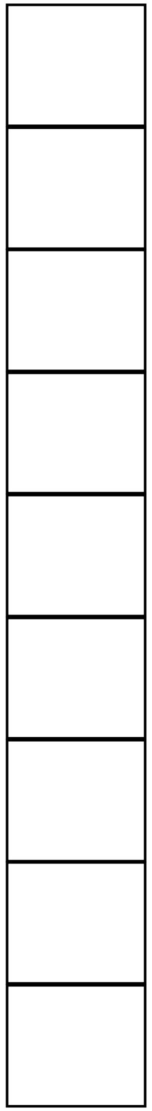
↑

① process each sym from left to right.

② if opening parenthesis is found push on stack.

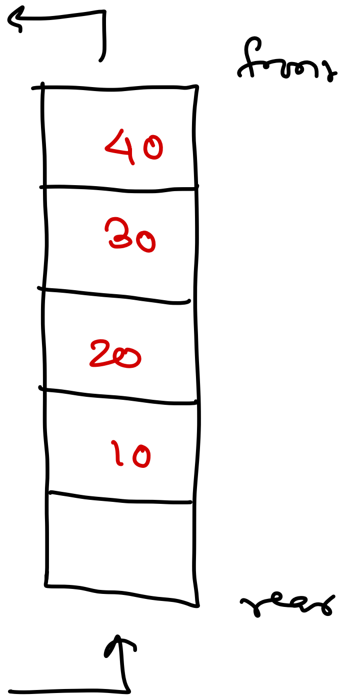
③ if closing parenthesis is found, check if top opening parenthesis in stack is matching with it.
if yes, pop & discard it.
if no, raise error.

④ at the end, if stack is not empty raise the error.



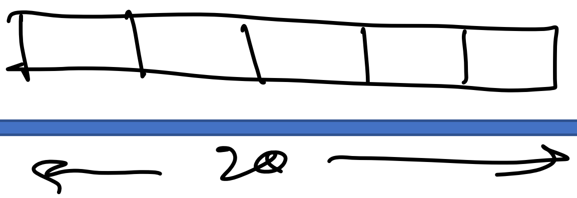
Stack using Queue

```
class my_stack {
    queue<int> q;
public:
    void push(int ele) {
        int n = q.size();
        q.push(ele);
        for(i=0; i<n; i++) {
            ele = q.front();
            q.pop();
            q.push(ele);
        }
    }
    void pop() {
        q.pop();
    }
    int peek() {
        return q.front();
    }
};
```



Arrays

int arr[5];



① cannot grow or shrink at runtime (efficiently),

② can lead to memory wastage (if too big).

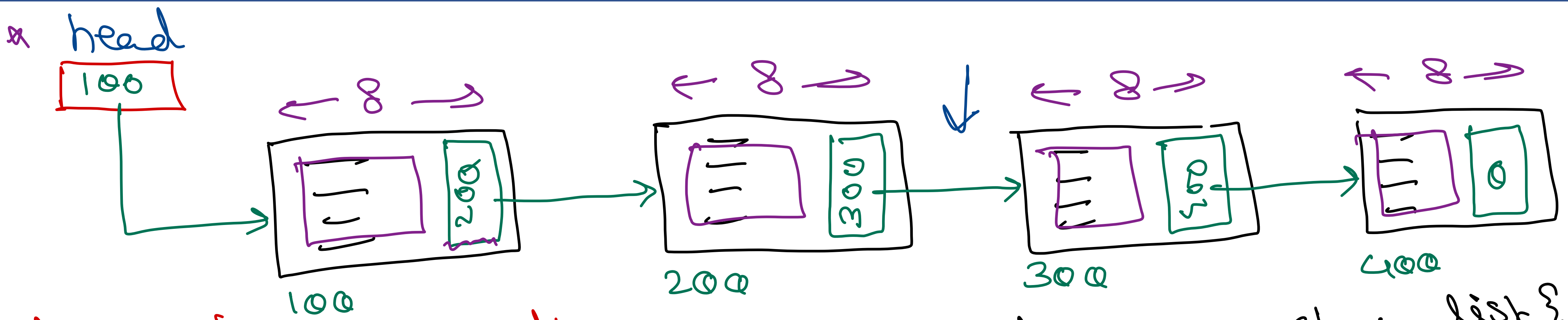
③ insert & del (in betn) is not efficient.

① no memory overheads.

② contiguous allocation → random access. (pointer arithmetic)

Linked List → List of records linked together.

↳ 4 bytes
↳ 32 bytes.



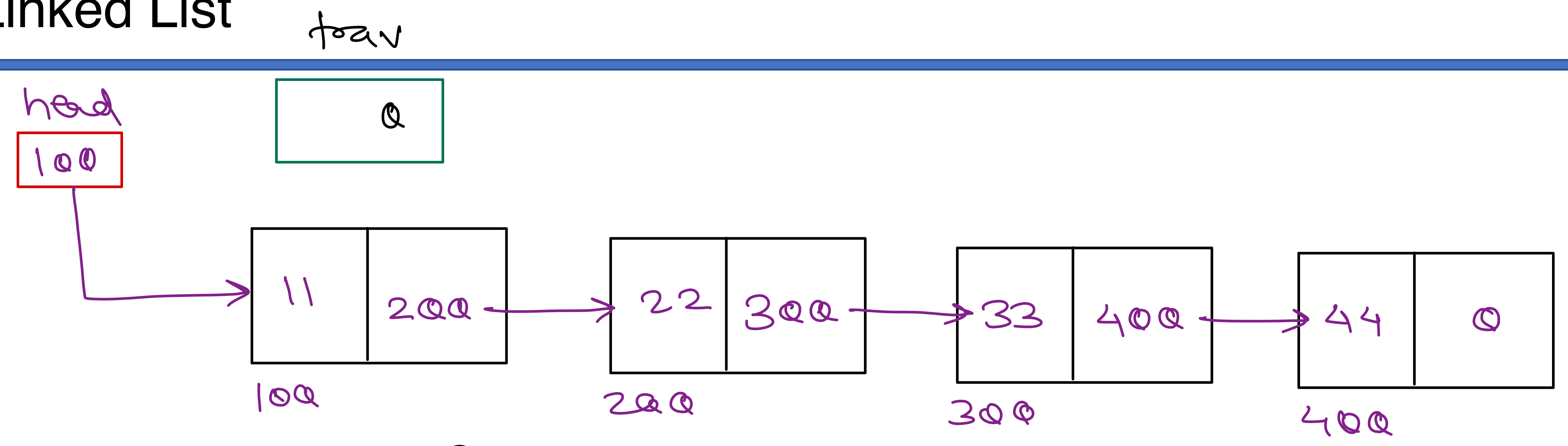
dynamic mem alloc:
- malloc()
- new operator

List operations:
- add_first()
- add_last()
- add_pos()
- del_first()
- del_last()
- del_pos()

```
class node
{
public:
    int data;
    node *next;
    node() {
        data = 0;
        next = NULL;
    }
    node (int val) {
        data = val;
        next = NULL;
    }
};
```

```
class list {
    node* head;
public:
    list() {
        head = NULL;
    }
    add_first();
    add_last();
    display();
    -
    -
    -
    -
};
```

Linked List

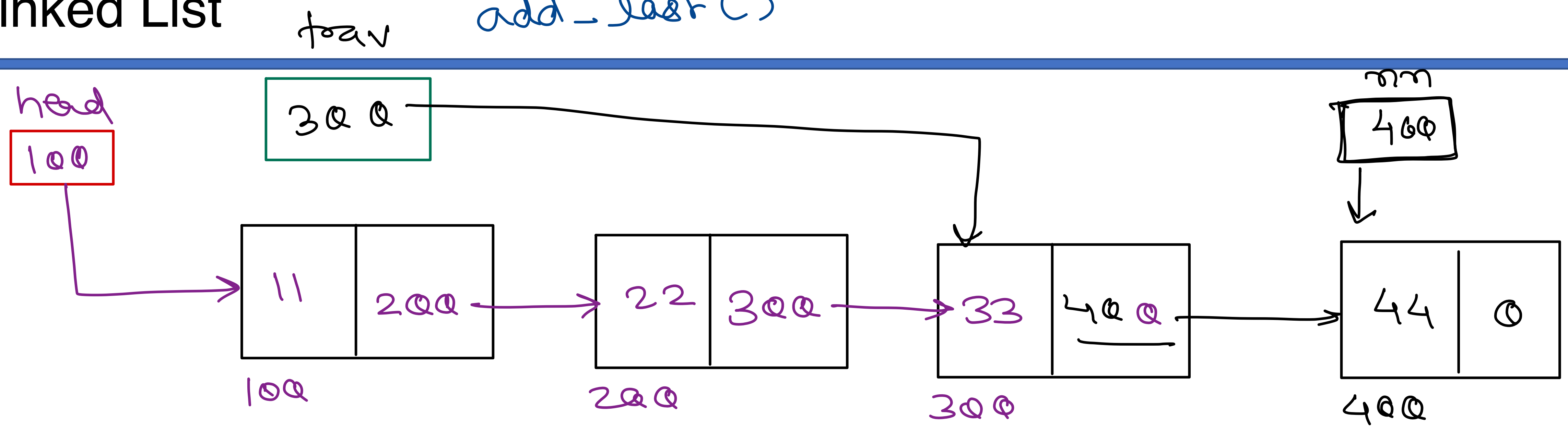


```
void display() {
    node * trav = head;
    while (trav != NULL) {
        cout << trav->data;
        trav = trav->next;
    }
}
```

11, 22, 33, 44

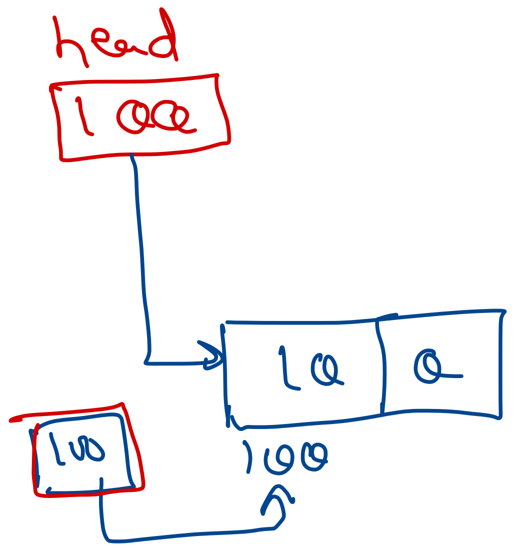
Linked List

add_last()



```
node *nn = new node(val);
node * trav;
if (head == NULL)
    head = nn;
else
{
    trav = head;
    while (trav->next != NULL)
        trav = trav->next;
    trav->next = nn;
}
```

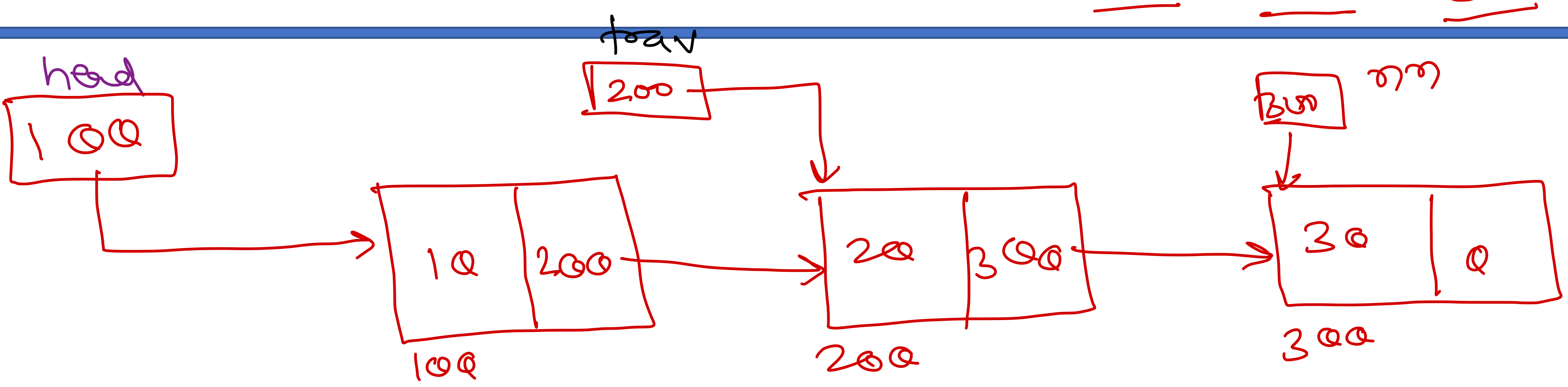
1. alloc & init new node
2. traverse till last node
3. last node next to new node.



Linked List

add_last()

10 20 30



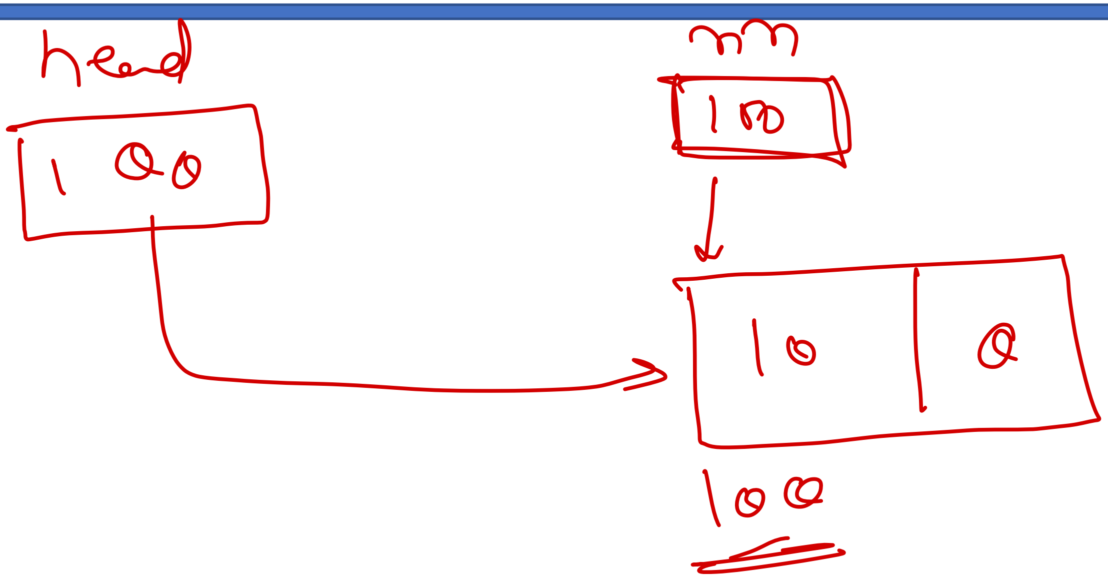
```
① node *nn = new node(val);
   node * trav;
   if (head == NULL)
       head = nn;
   else
   {
   ②  trav = head;
       while (trav->next != NULL)
           { trav = trav->next; }
   ③  trav->next = nn;
   }
```

① alloc & init new node

② traverse till last node

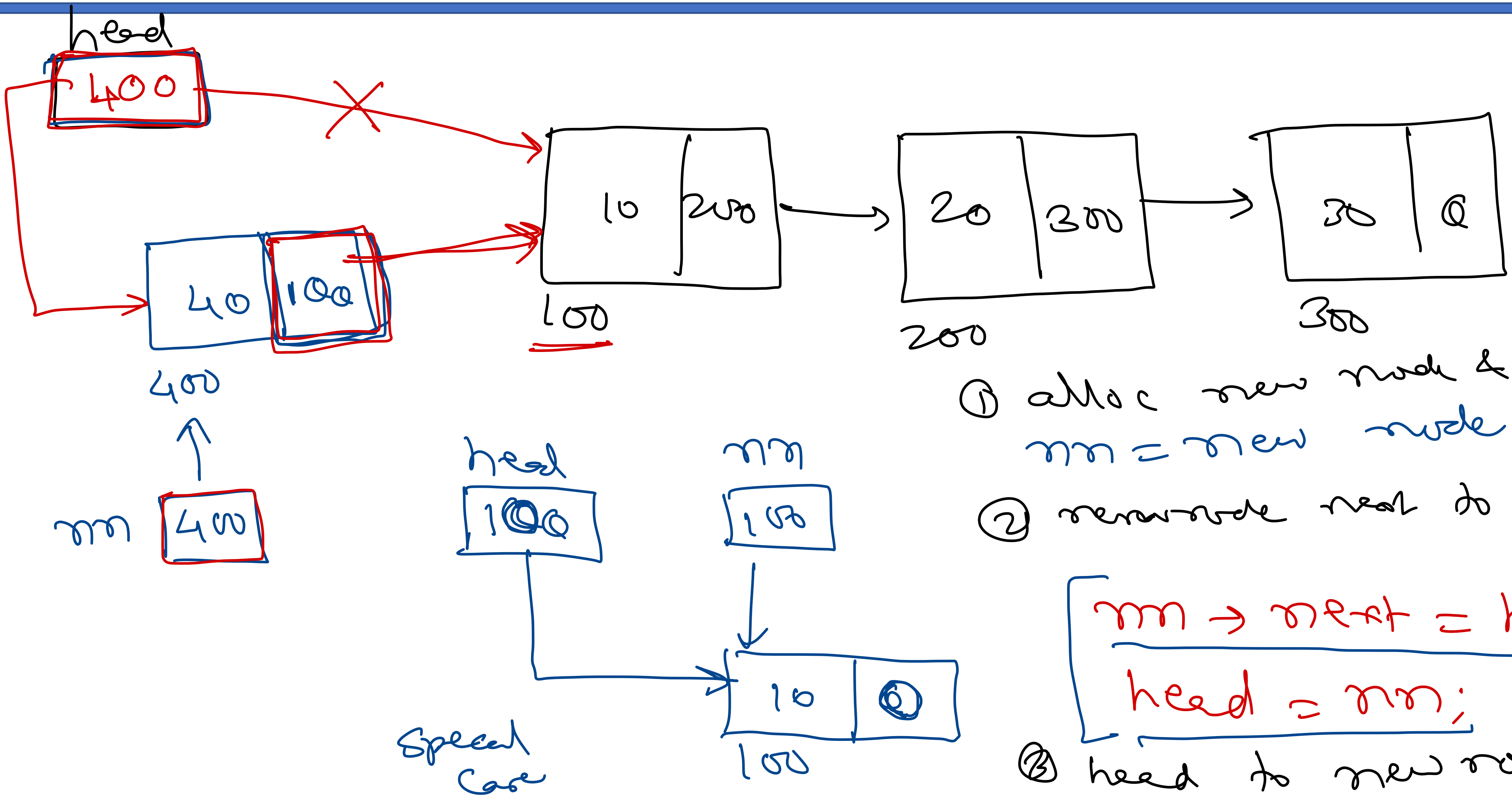
③ last node next to new node.

Linked List



```
if(head==NULL)
    head=nn;
```

Linked List



① alloc new node & init

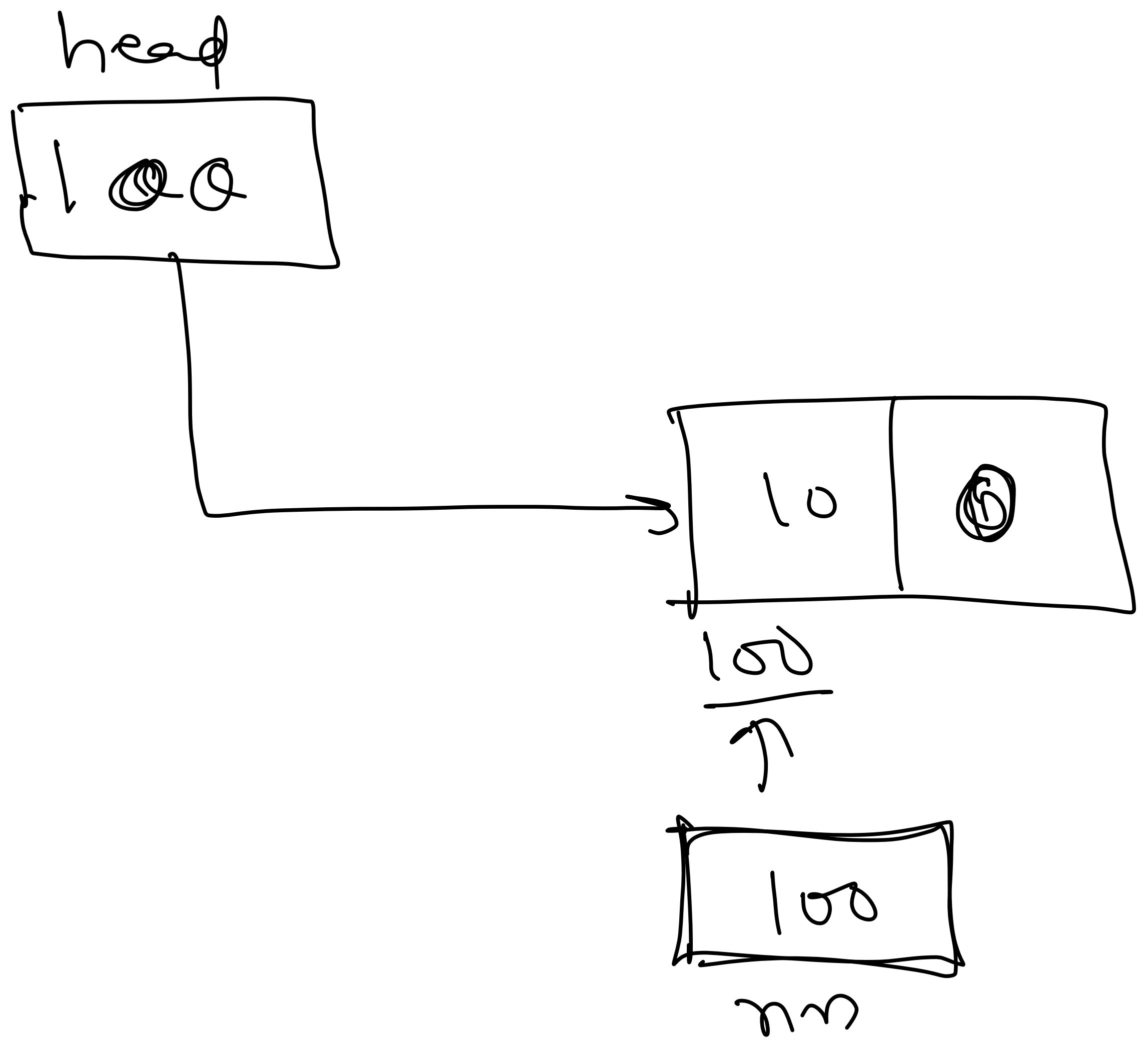
nn = new node(v);

② newnode next to head

$$\boxed{\begin{array}{l} nn \to next = head; \\ head = nn; \end{array}}$$

③ head to new node.

Linked List



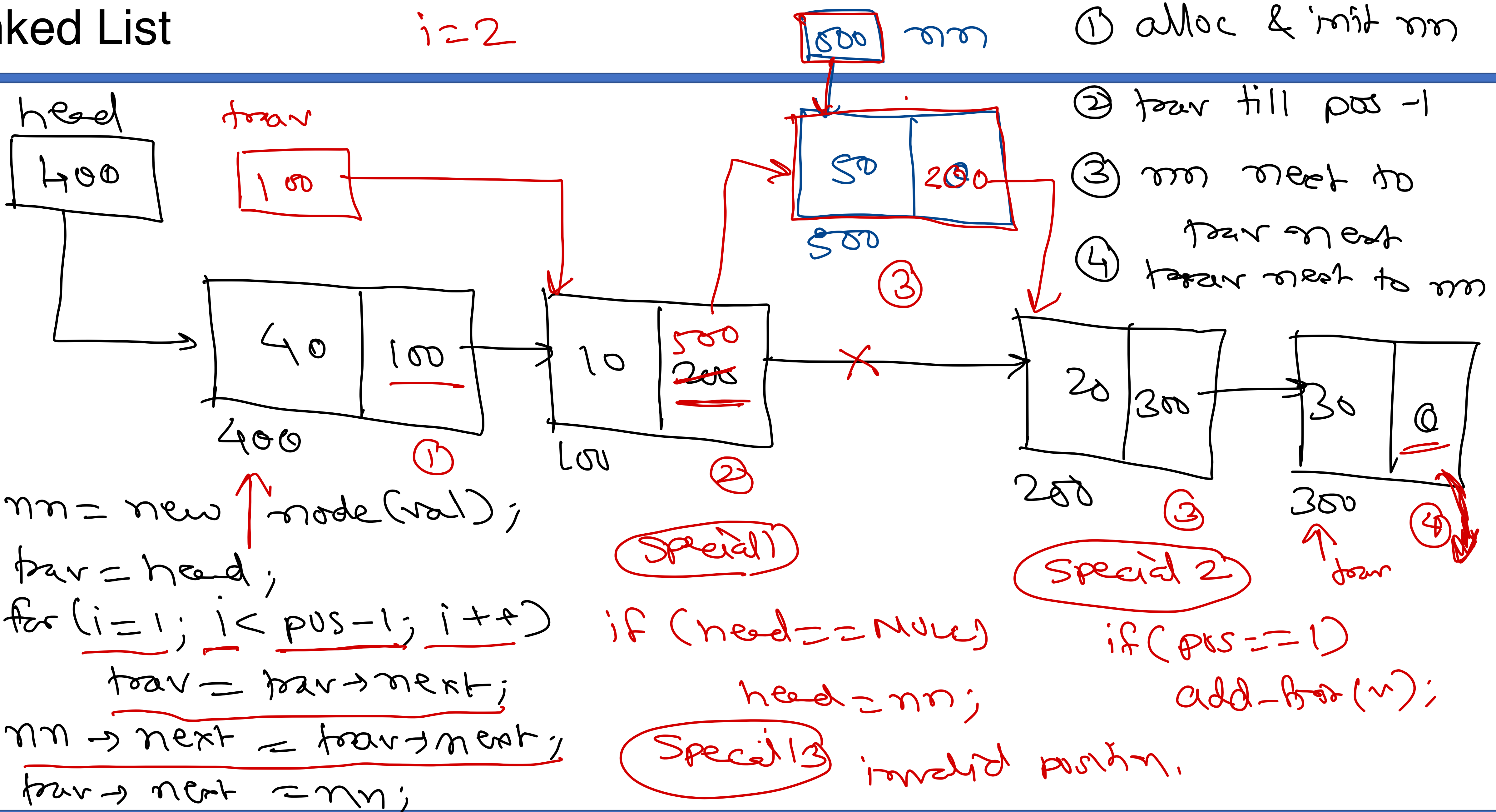
nn = new node(n);

nn->next = head;

head = nn;

Linked List

i=2



1. alloc & init nn
2. trav till pos -1
3. nn next to trav next
4. trav next to nn

① nn = new node(val);
② trav = head;
for (i=1; i< pos-1; i++)
trav = trav->next;
③ nn -> next = trav->next;
④ trav -> next = nn;

Special 1

if (head==NULL)
head = nn;

Special 2

if (pos==1)
add_first(v);

Special 3 invalid position.

Linked List

special 1

```
if (head == NULL)
    head = nn;
```

special 2

```
if ( pos == 1)
    add_first (val);
```

special 3 invalid pos

```
trav = head;
for(i = 1; i < pos-1; i++){
    if (trav->next == NULL)
        break;

    trav = trav->next;
}
```

=

SUNBEAM

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>