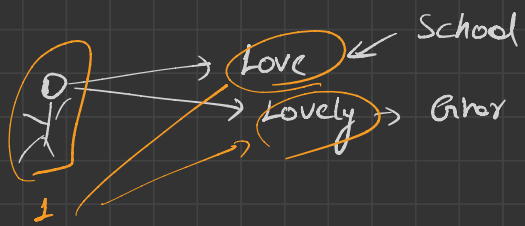
Lecture - 28

① Reference Variable




```
int i = 5;
```

→ 710

5

i

j

→ same memory but different names

H/w
- void pointer → ?
- address typecasting ?

→ `int i = 5;`

Create a Ref. variable → `int & j = i`

① ②

710

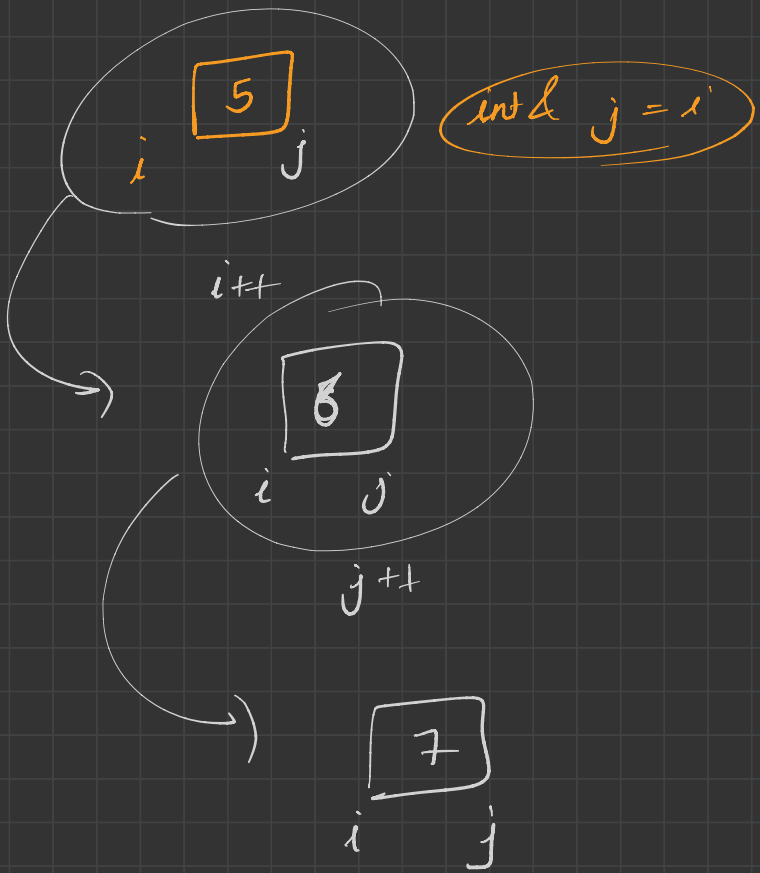
5 → 6 → 7

i

j

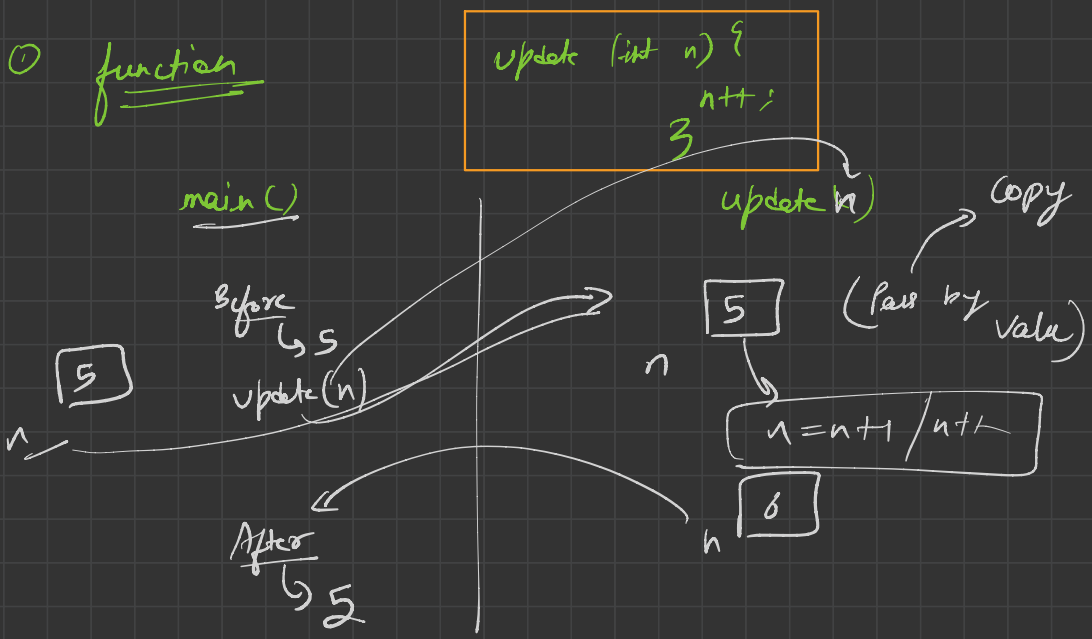
i++

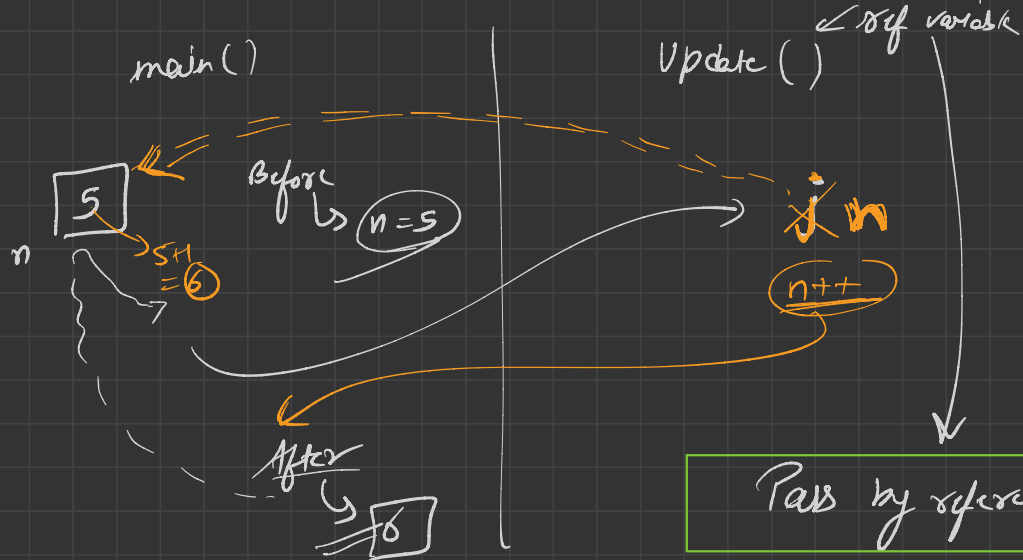
j++


5
i
j
int& j = i
i++
6
i
j
j++
7
i
j

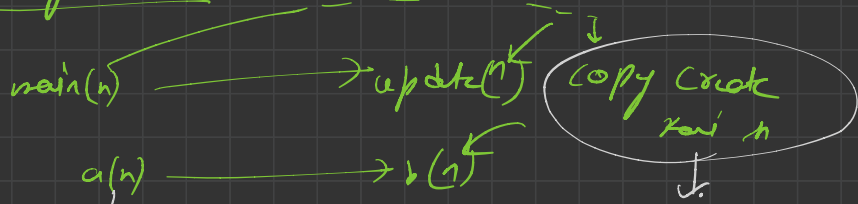
Why - ?

① function



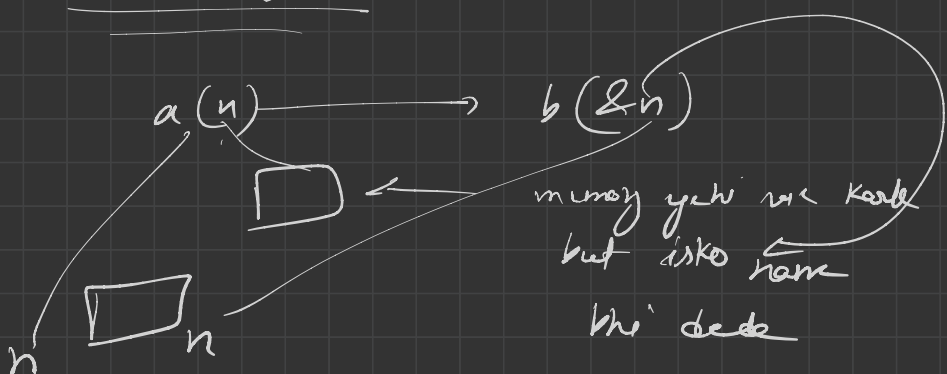


Pass by Value



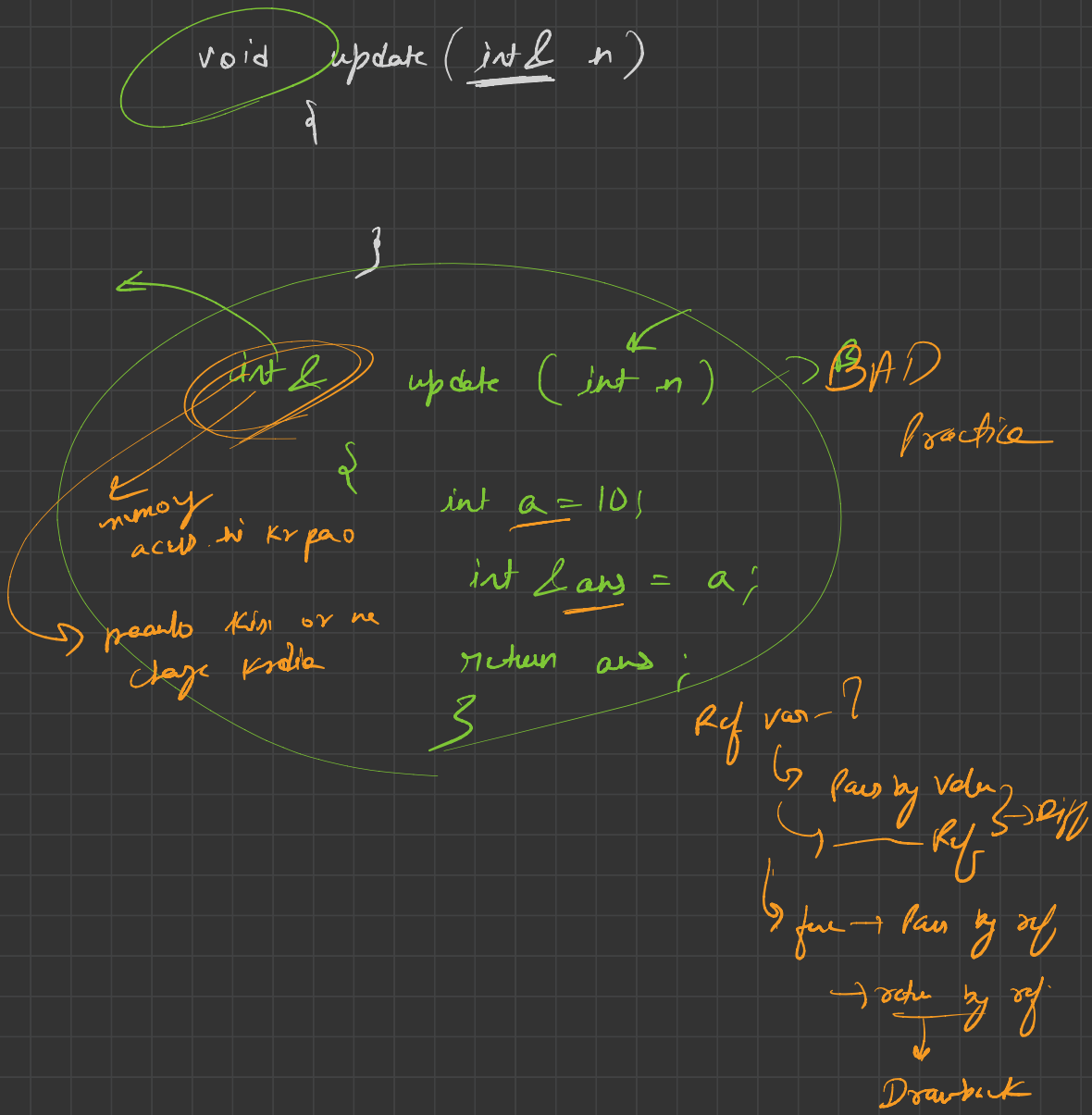
memory leti
alag se

Pass by reference



memory yehi rahi koode
but isko naam
bhi dedo

⟹ same memory
,
naam different
hai


void update (int& n)
{
}
int& update (int n)
BAD practice
memory access ki krpao
pseudo kisi or ne change krdia
int a = 10;
int& ans = a;
return ans;
Ref var - ?
Pass by value
Ref
fun → Pass by ref
→ return by ref.
Drawback

→ Array

```
int n;
cin >> n;
int arr[n];
```

cin >> n; → runtime → Yes / No

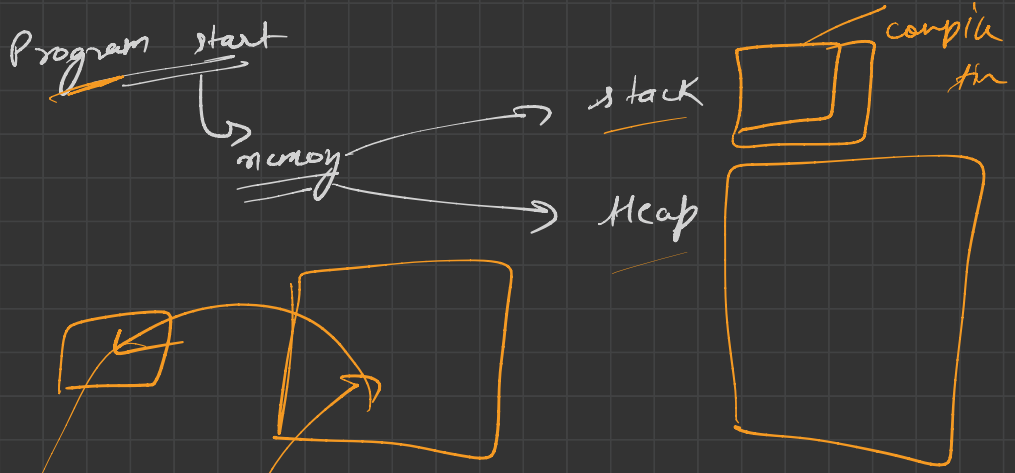
int arr[n]; → BAD practice

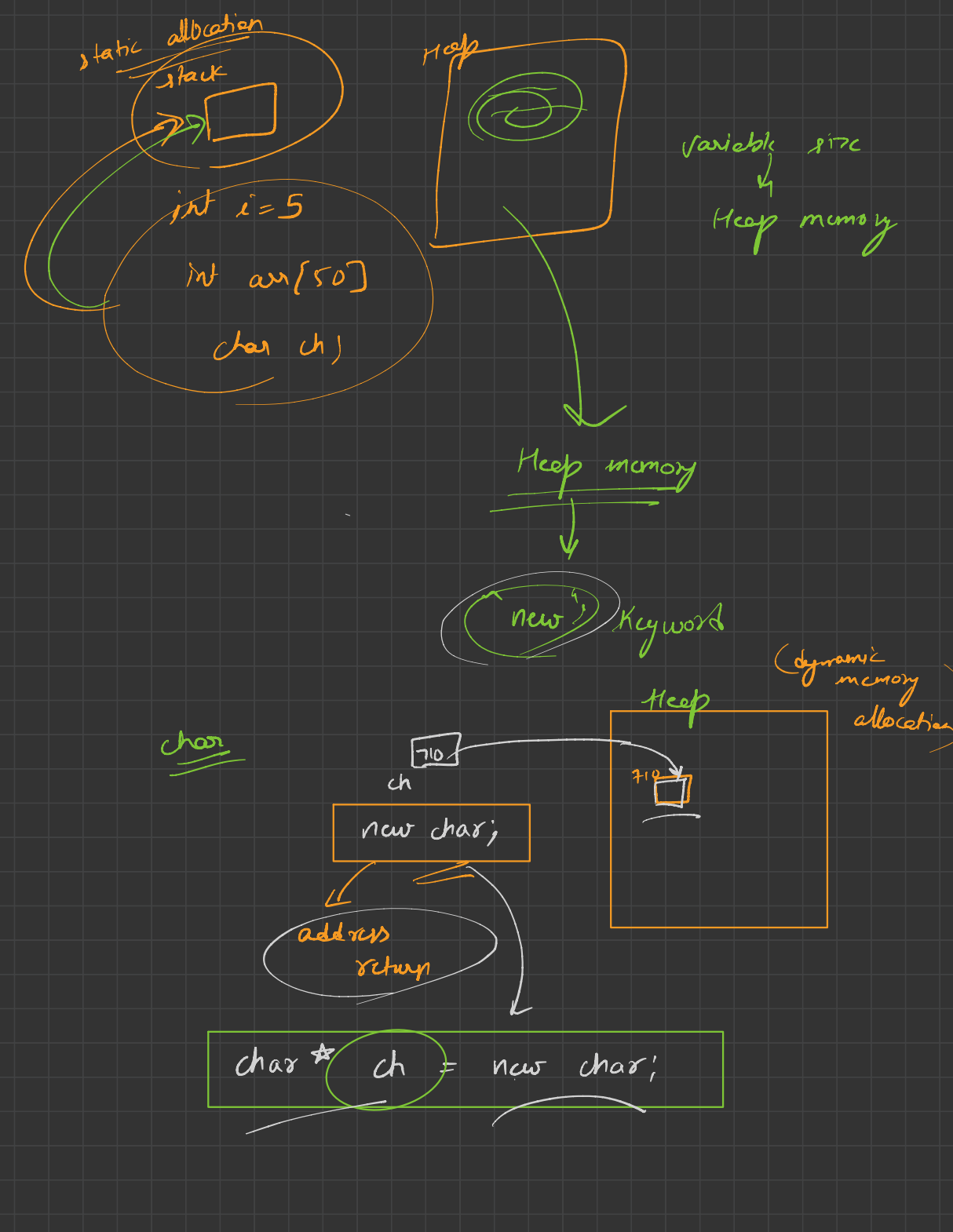
int arr[n]; → compile time

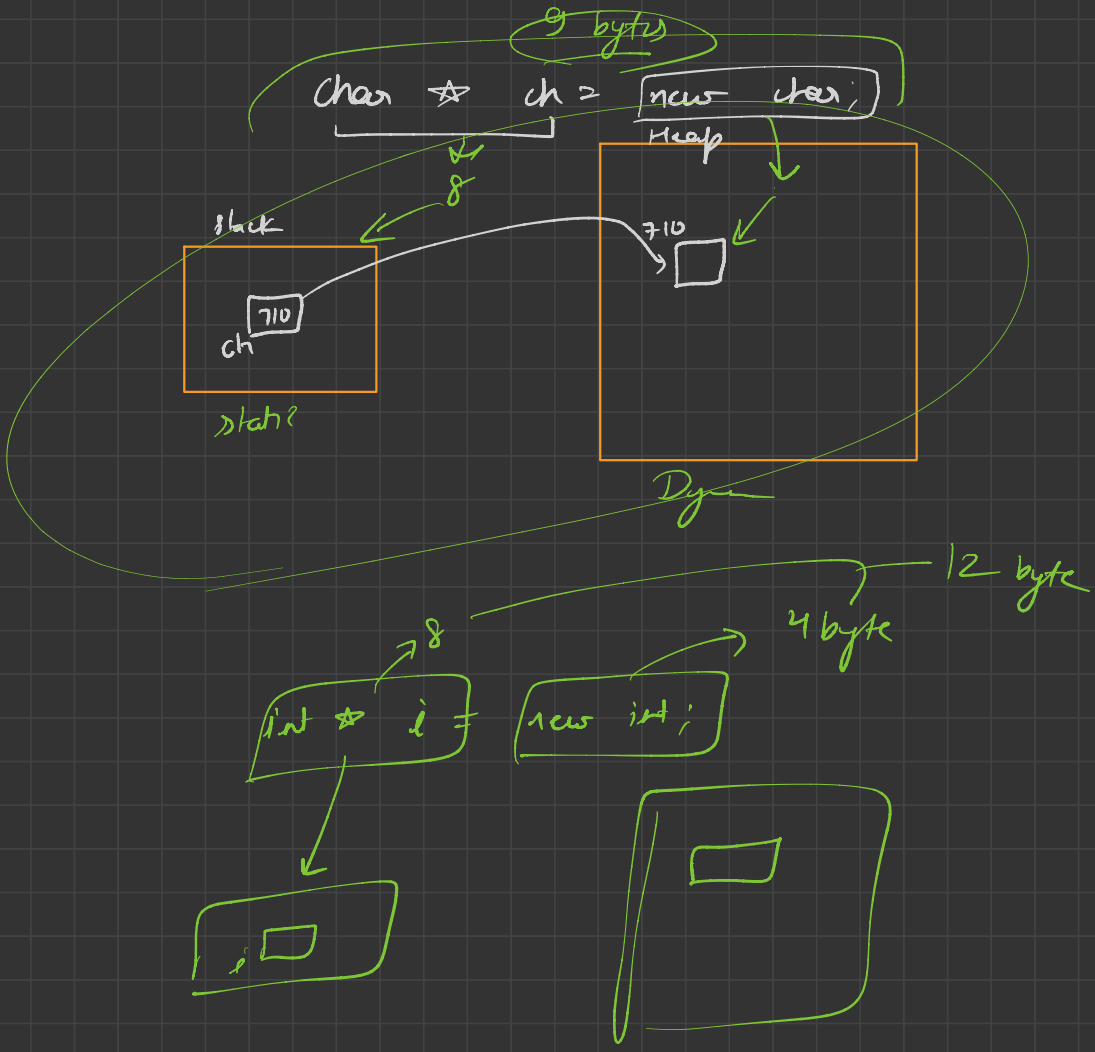
→ why ?

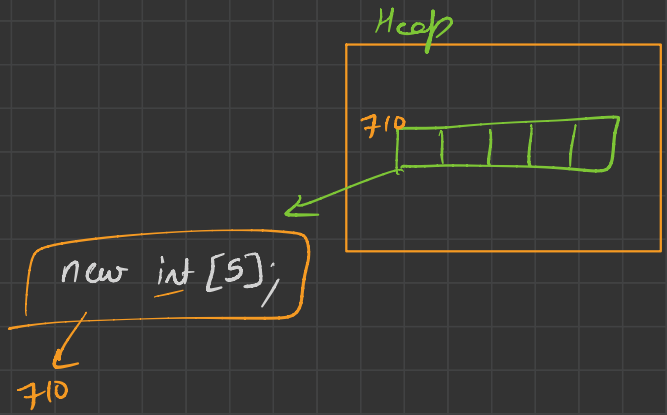
int arr[10000];

10000 → size

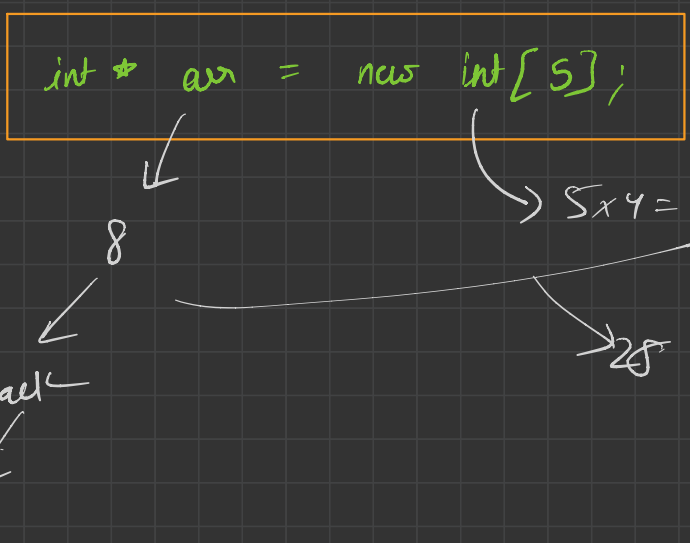


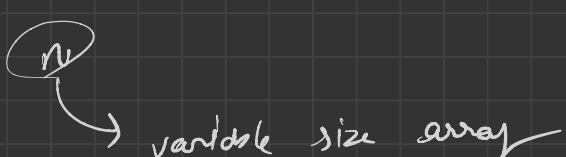

static allocation
stack
int i=5
int arr[50]
char ch
Heap
variable size
Heap memory
Heap memory
new Keyword
(dynamic memory allocation
Heap
char
710
ch
new char;
710
address return
char * ch = new char;

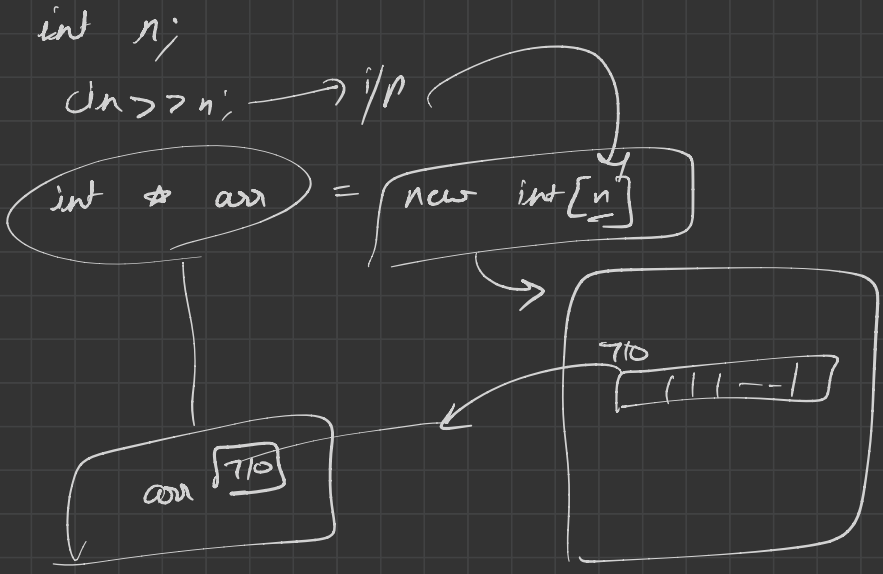

9 bytes
char * ch = new char;
Heap
8
Stack
710
710
ch
Stack
Dyn
12 byte
8
4 byte
int * i = new int;
i


Heap
710
new int [5];
710

int * arr


int * arr = new int [5];
8
5 x 4 = 20
Heap
20
stack
static
dynamic


n
variable size array



arr[i] = *(arr + i)

Diff b/w :-

Static vs Dynamic

① `int arr[50]`

$50 \times 4 = 200$ byte

↓

Stack

`int * arr = new int[50]`

8 byte → Stack

$50 \times 4 \rightarrow 200$ (Heap)

208

case -> 1

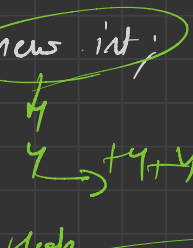
```
while (true)
{
    int i = 5;
}
```

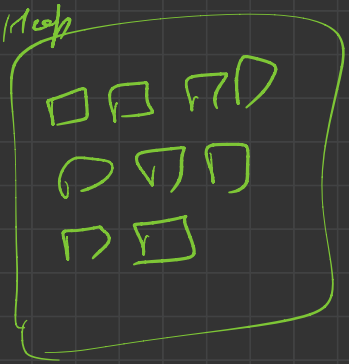
→ 1 int memory

4 byte

Call - 2

```
while (true)
{
    int * p = new int;
}
```





Program crash

static allocation

↓

memory automatically release

Dynamic allocation

↓

manually release

↓

"delete" keyword

int

```
int* i = new int
```

for single element deletion →

```
delete i
```

int arr[4]

arr

```
int* arr = new int[50];
```

```
delete [] arr;
```

for array deletion →