This is CS50.

cs50.brianyu.me

Week 5

- Data Structures
- Linked Lists
- Trees
- Hash Tables
- Tries

What questions do you have?

Today

Linked Lists

Hash Tables, Trees, and Tries

Lab

Part One

Linked Lists

Arrays

1	2	3	4	5	6	7	8

Arrays

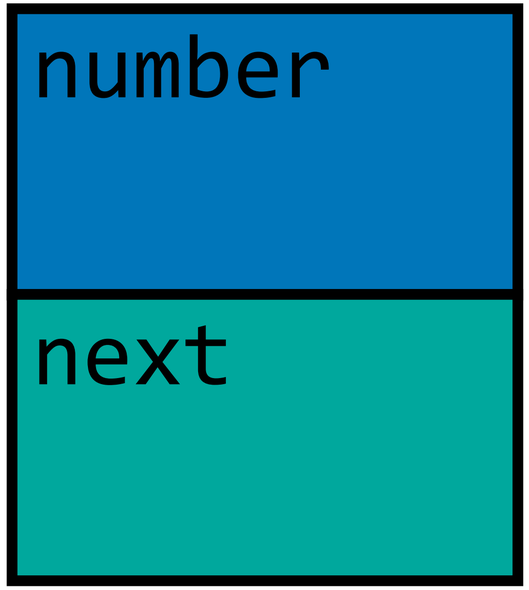
- Fixed size
- Contiguous in memory

Linked Lists

- Any size
- Not contiguous in memory

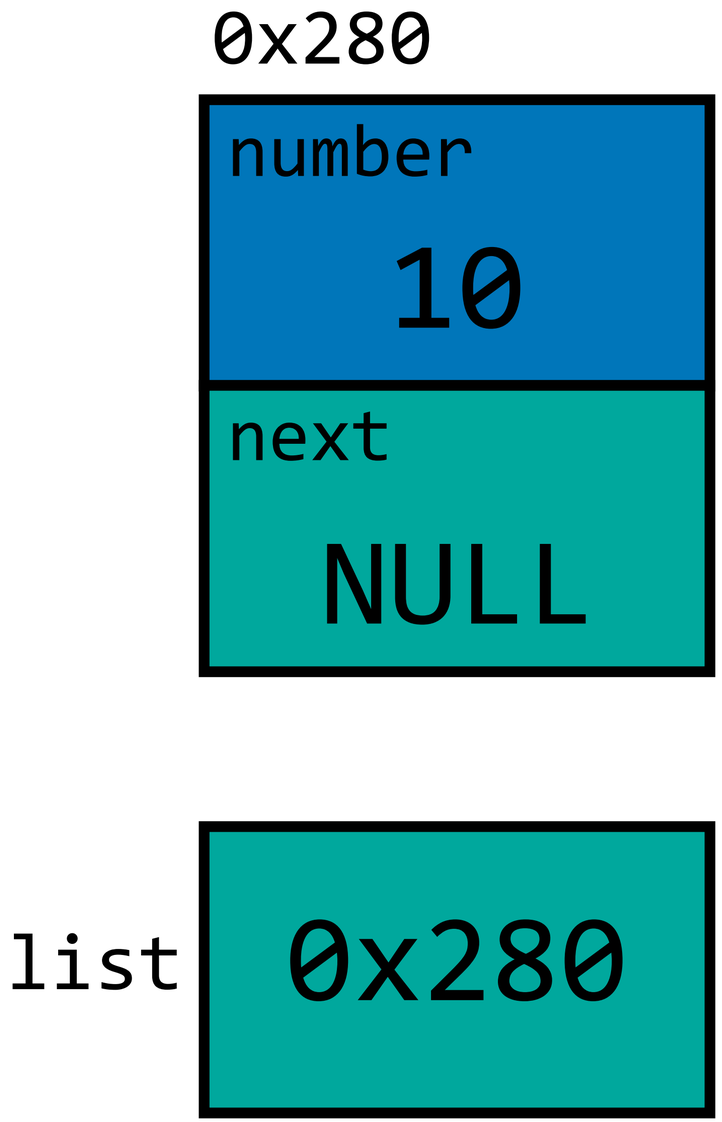
Linked List

```
typedef struct node
{
    int number;
    struct node *next;
}
node;
```

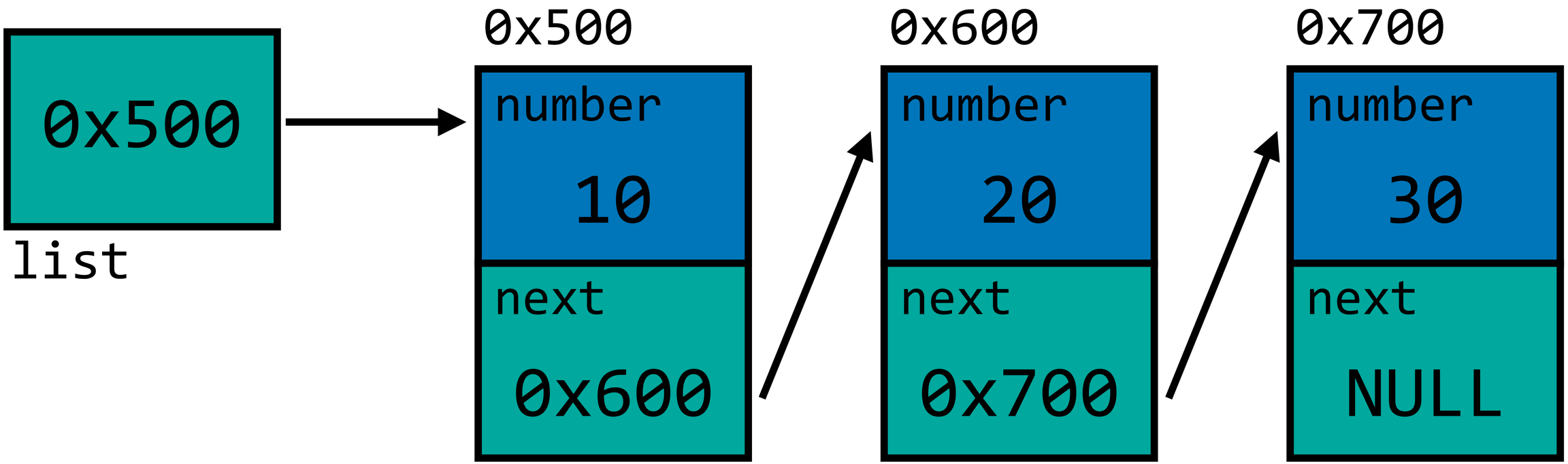


Linked List

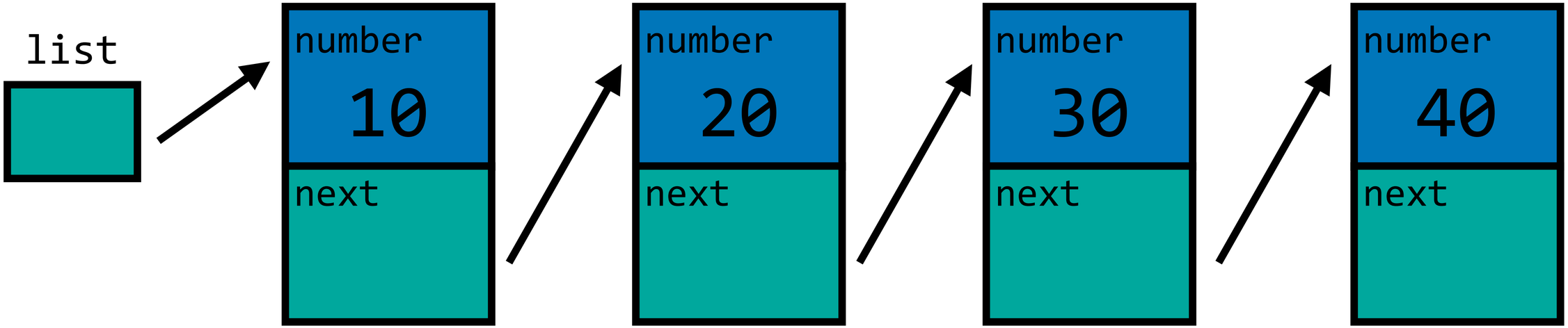
```
node *list = malloc(sizeof(node));

list->number = 10;

list->next = NULL;
```



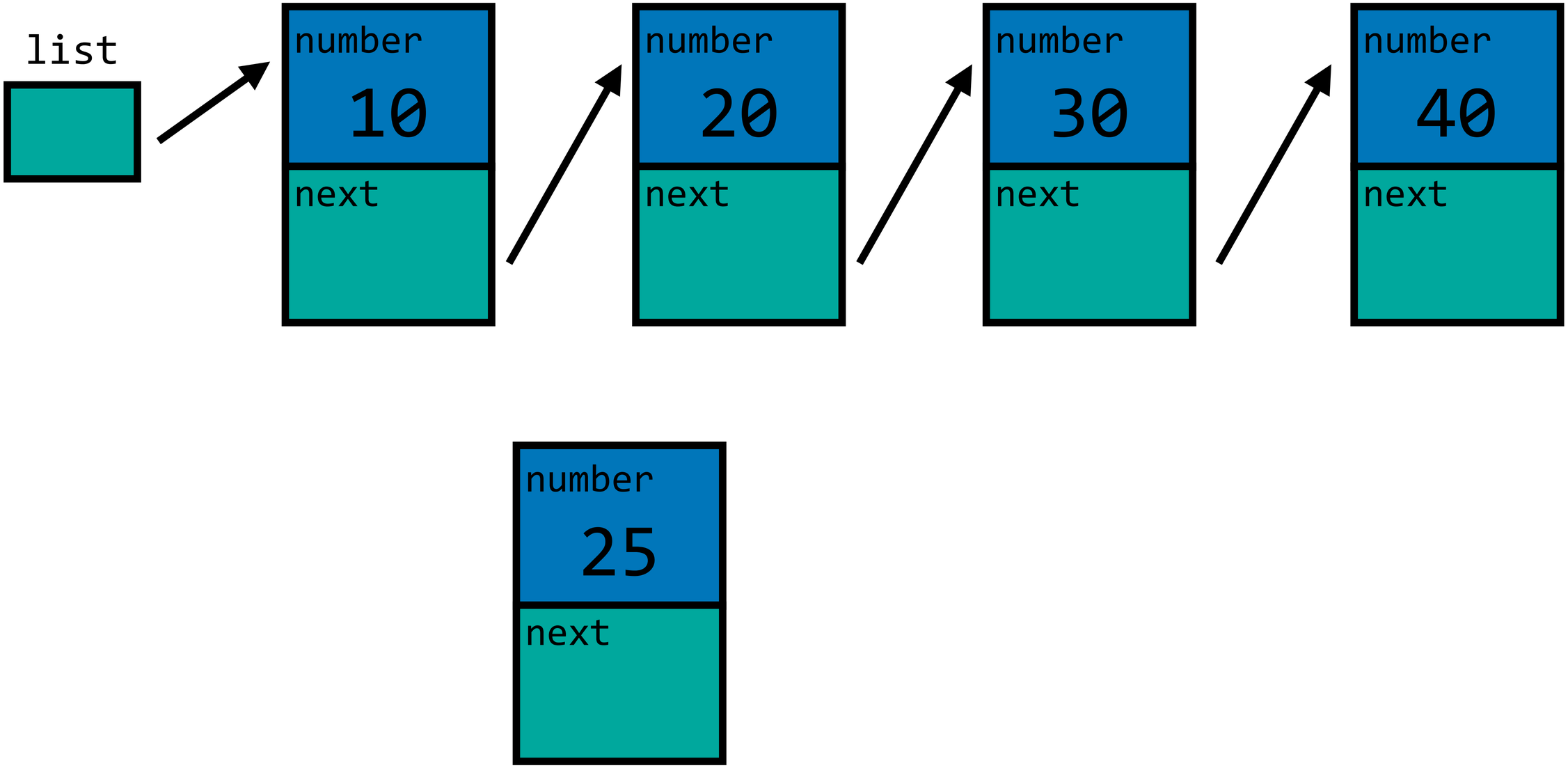
Linked List


0x500
list
0x500
number
10
next
0x600
0x600
number
20
next
0x700
0x700
number
30
next
NULL

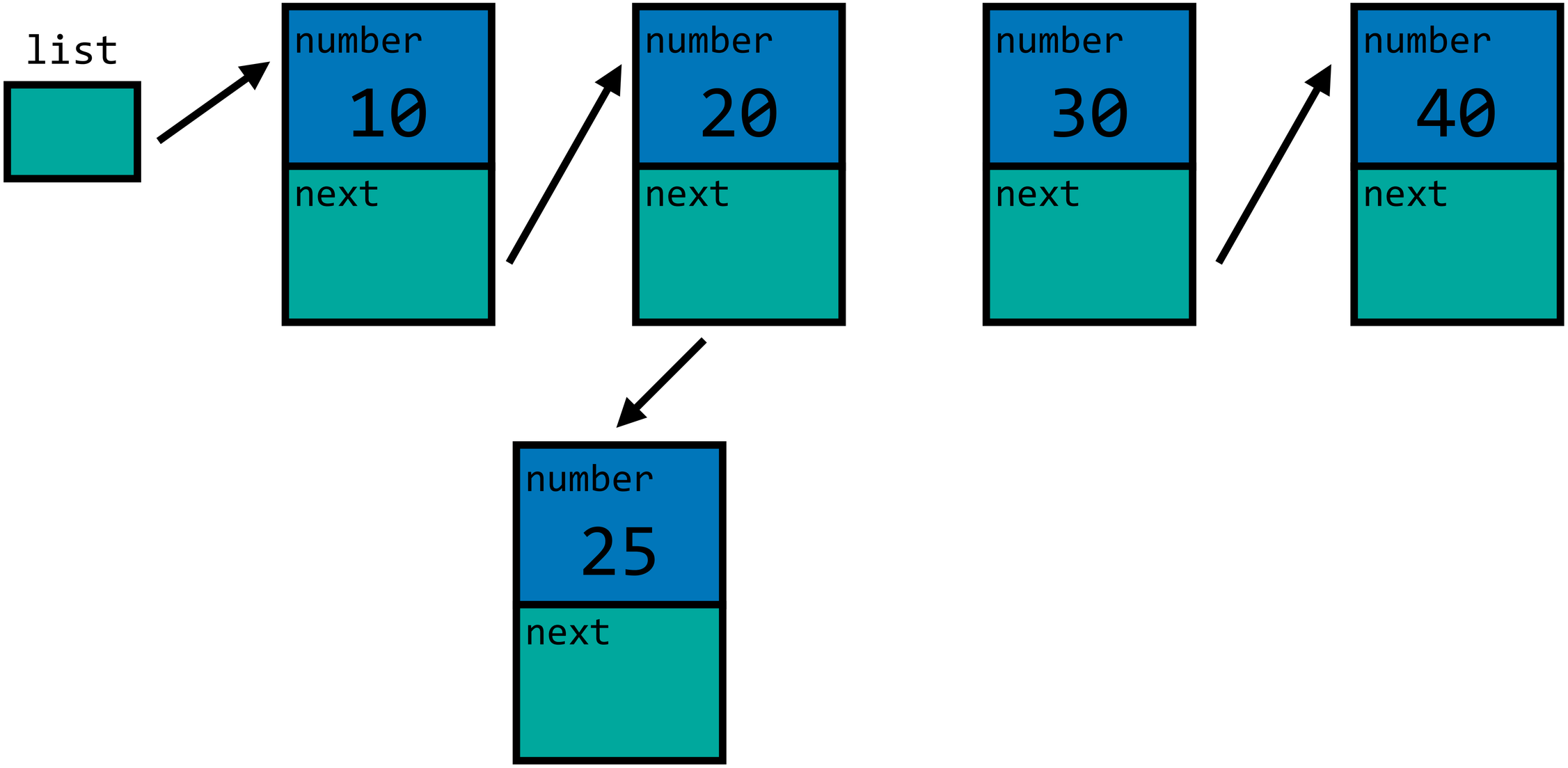
Insertion


list
number
10
next
number
20
next
number
30
next
number
40
next

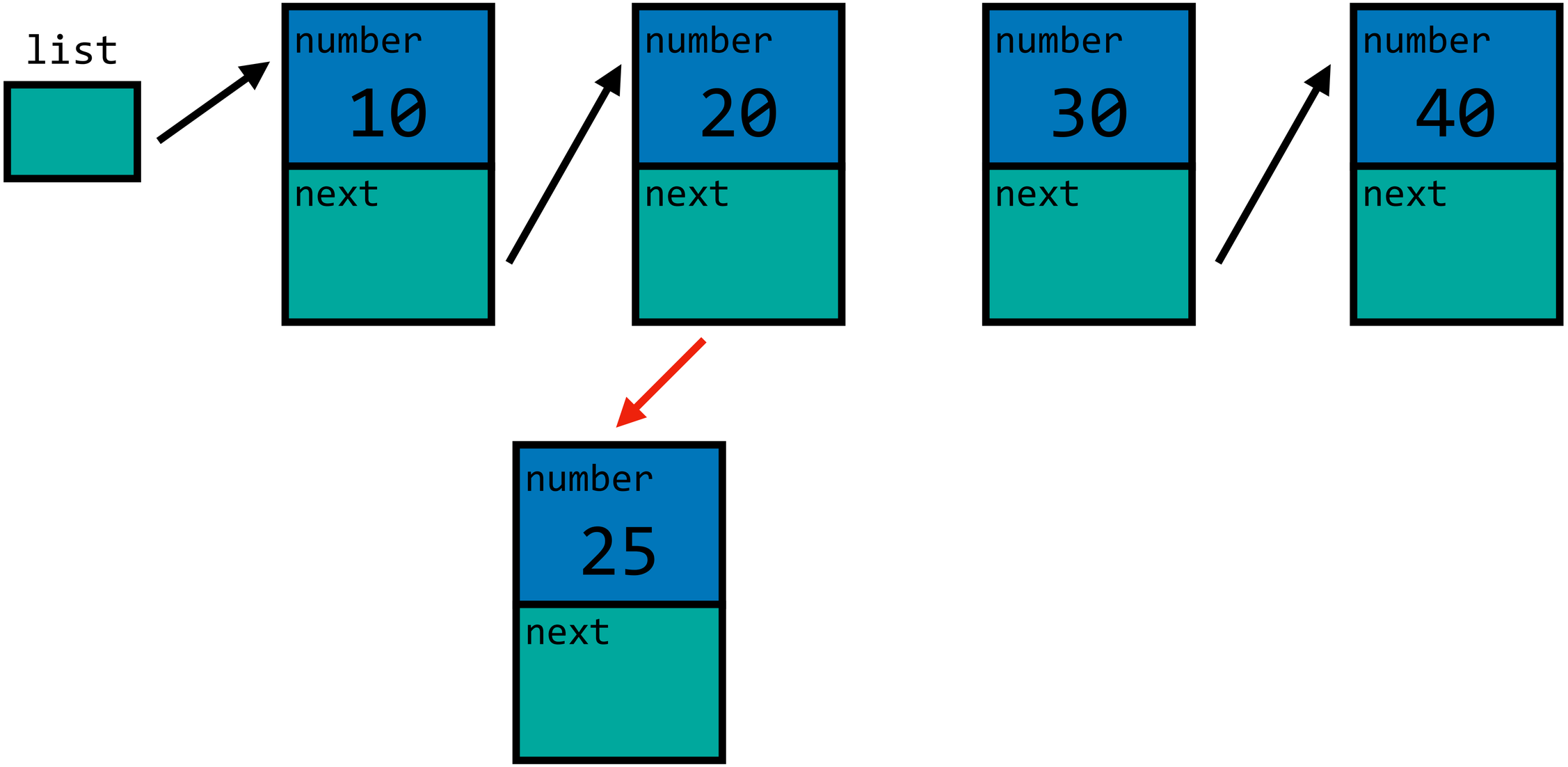
Insertion


list
number
10
next
number
20
next
number
30
next
number
40
next
number
25
next

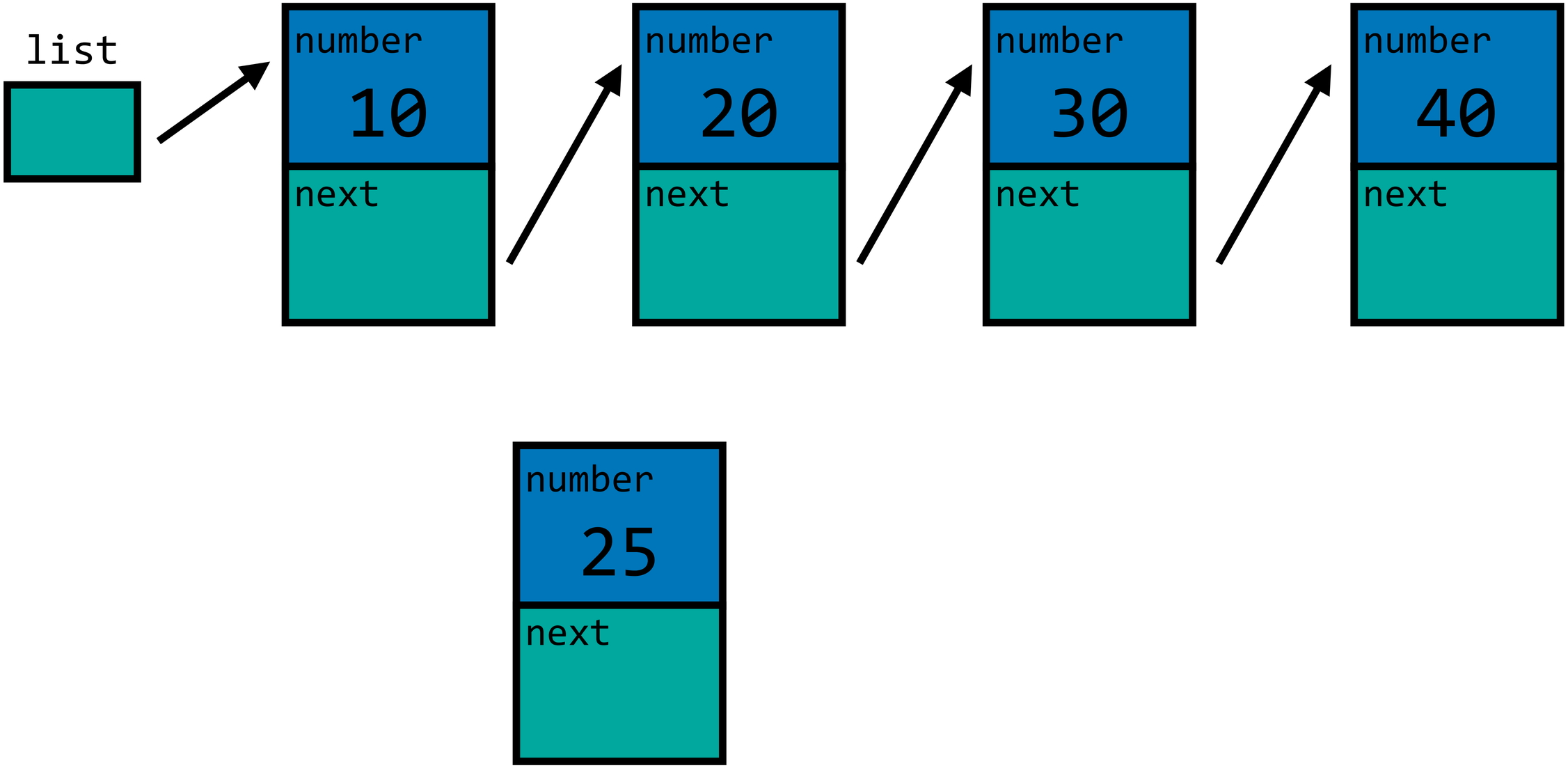
Insertion


list
number
10
next
number
20
next
number
30
next
number
40
next
number
25
next

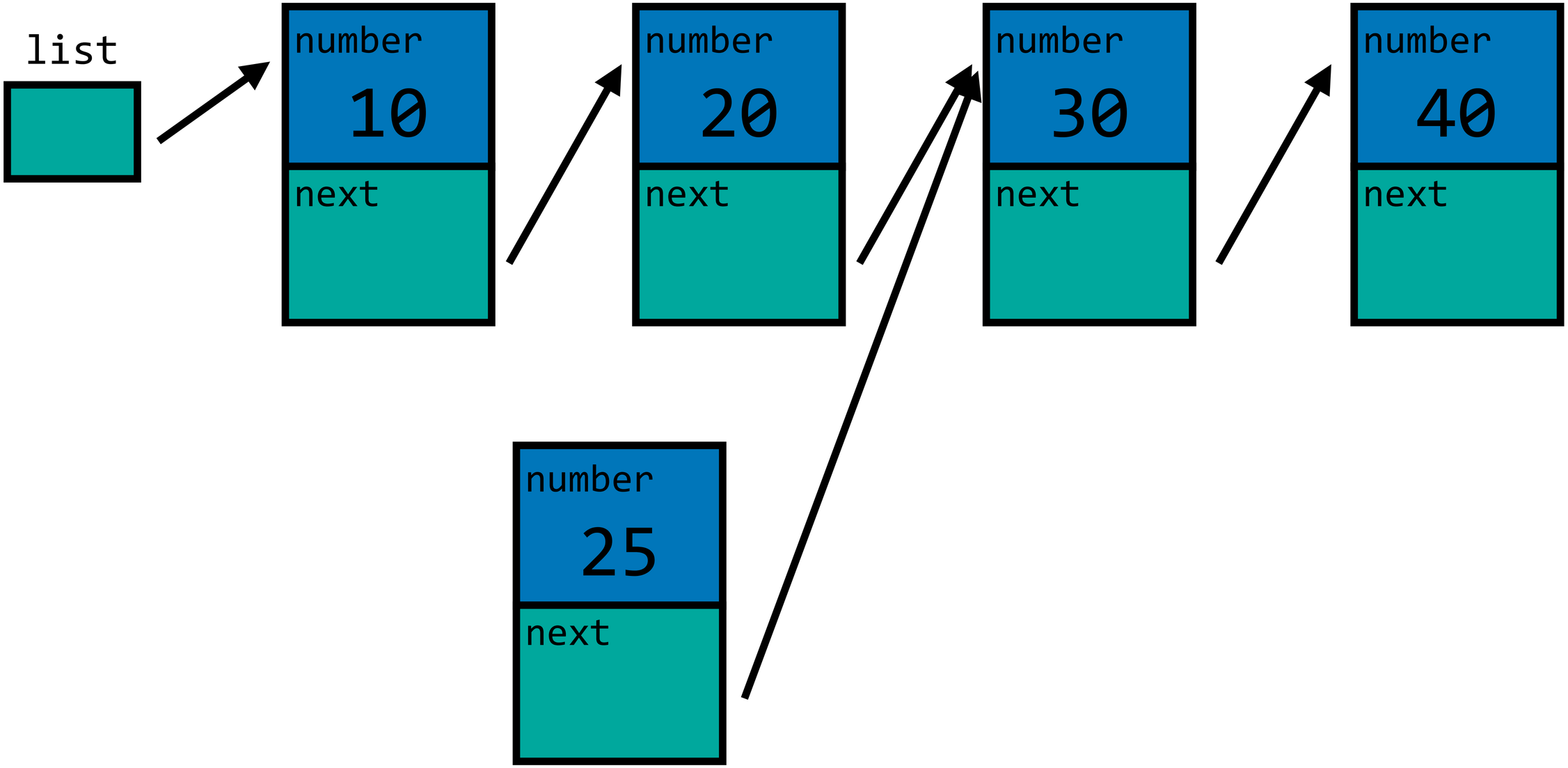
Insertion


list
number
10
next
number
20
next
number
30
next
number
40
next
number
25
next

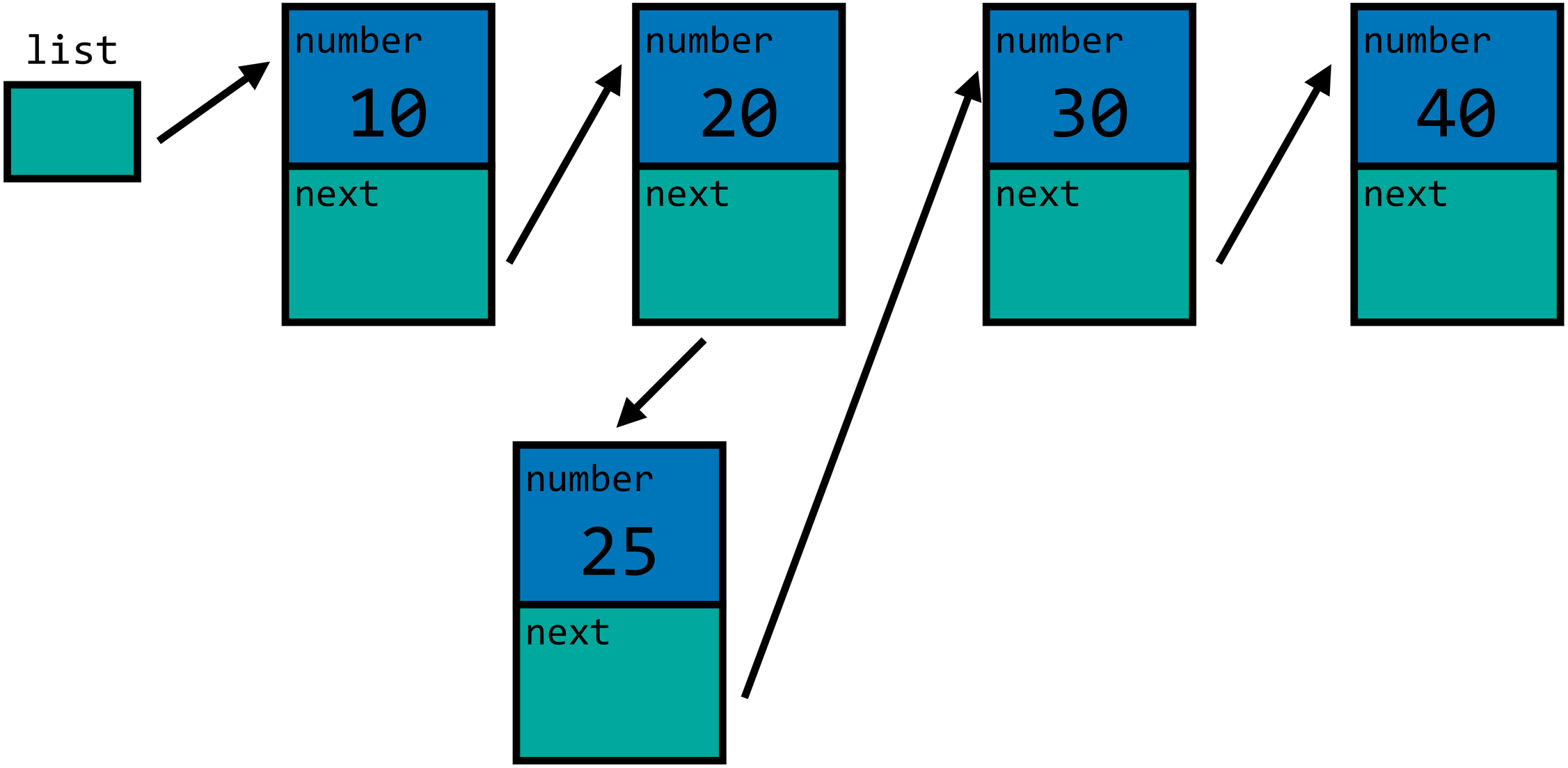
Insertion

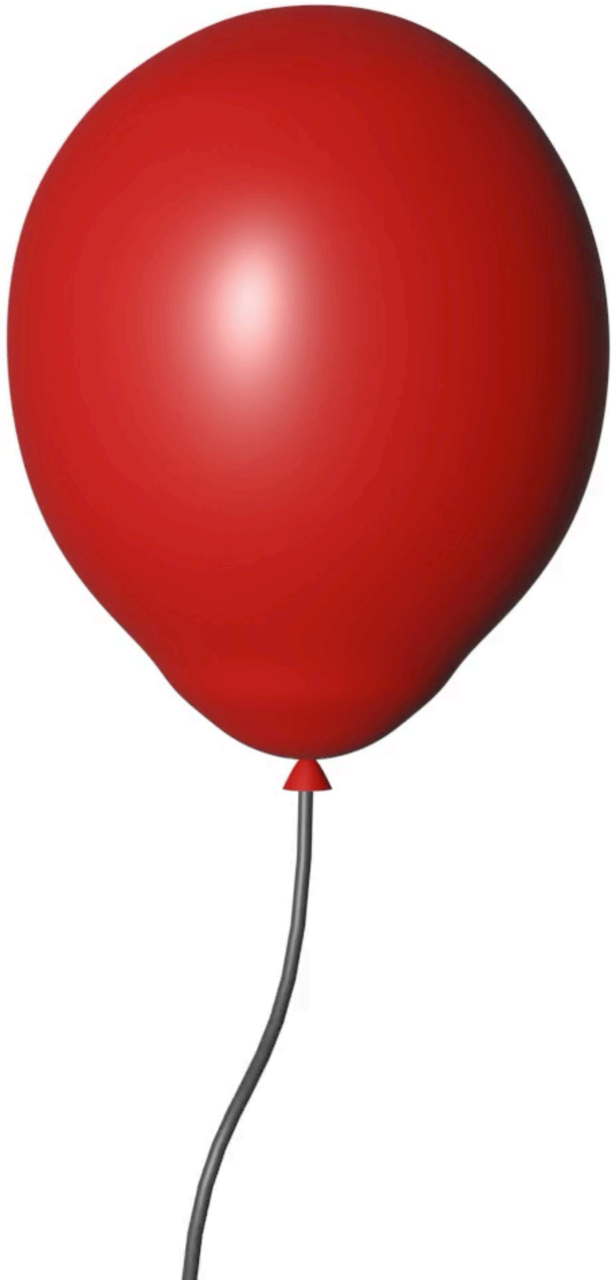

list
number
10
next
number
20
next
number
30
next
number
40
next
number
25
next

Insertion

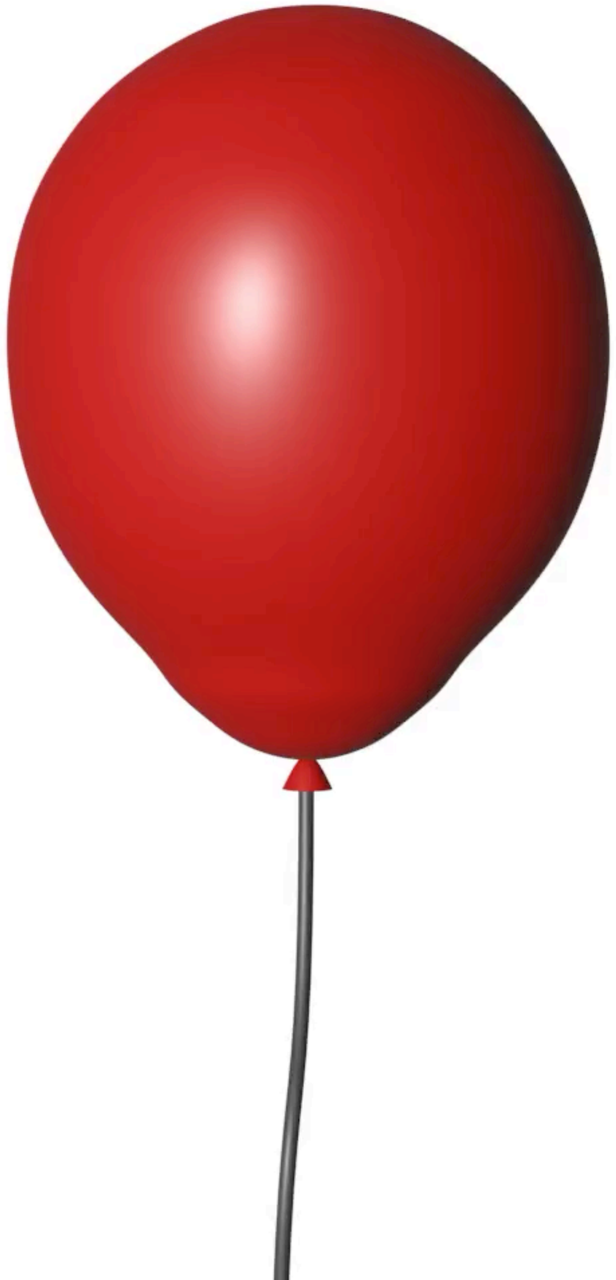

list
number
10
next
number
20
next
number
30
next
number
40
next
number
25
next

Insertion


list
number
10
next
number
20
next
number
30
next
number
40
next
number
25
next

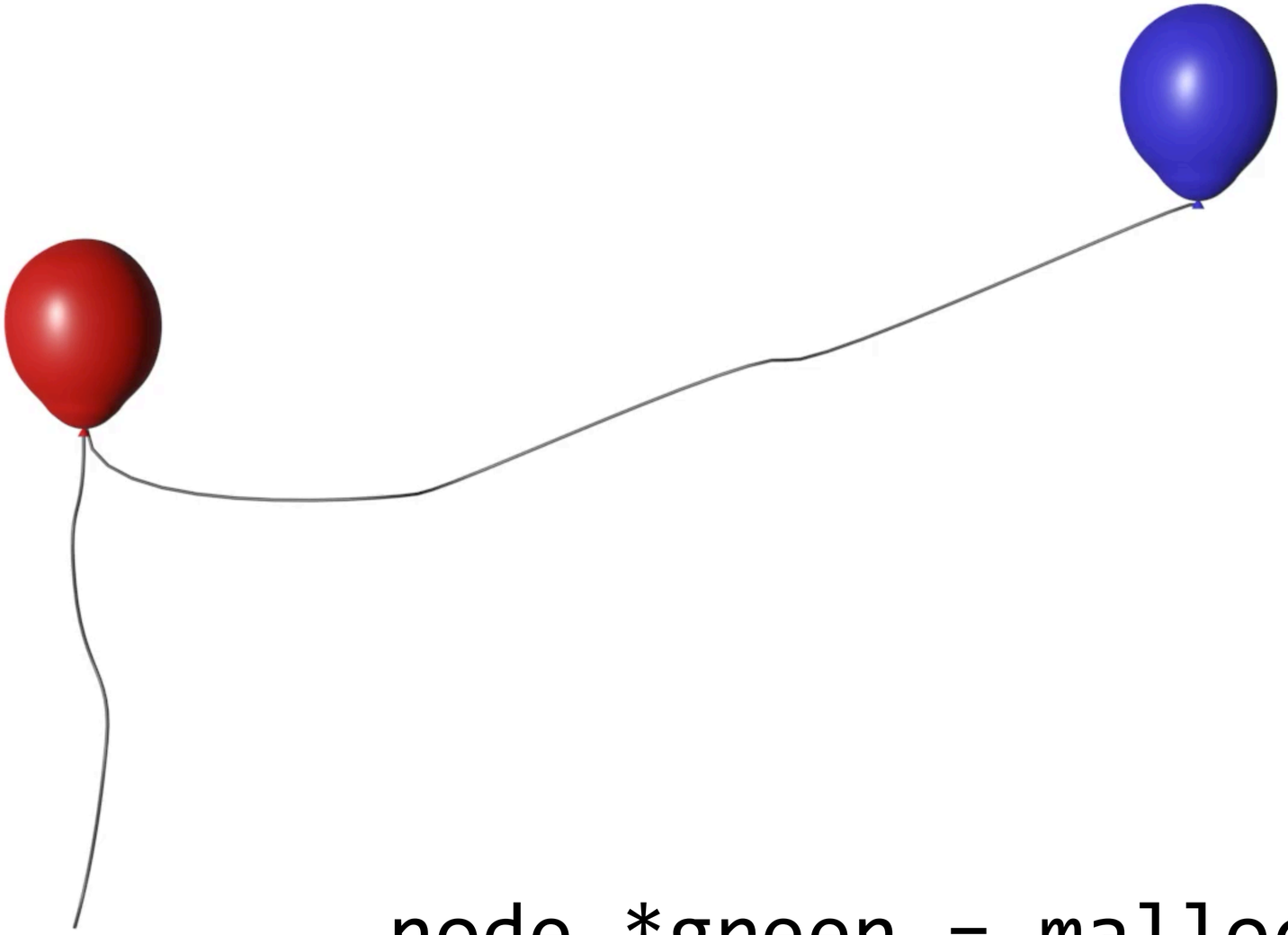


node *red

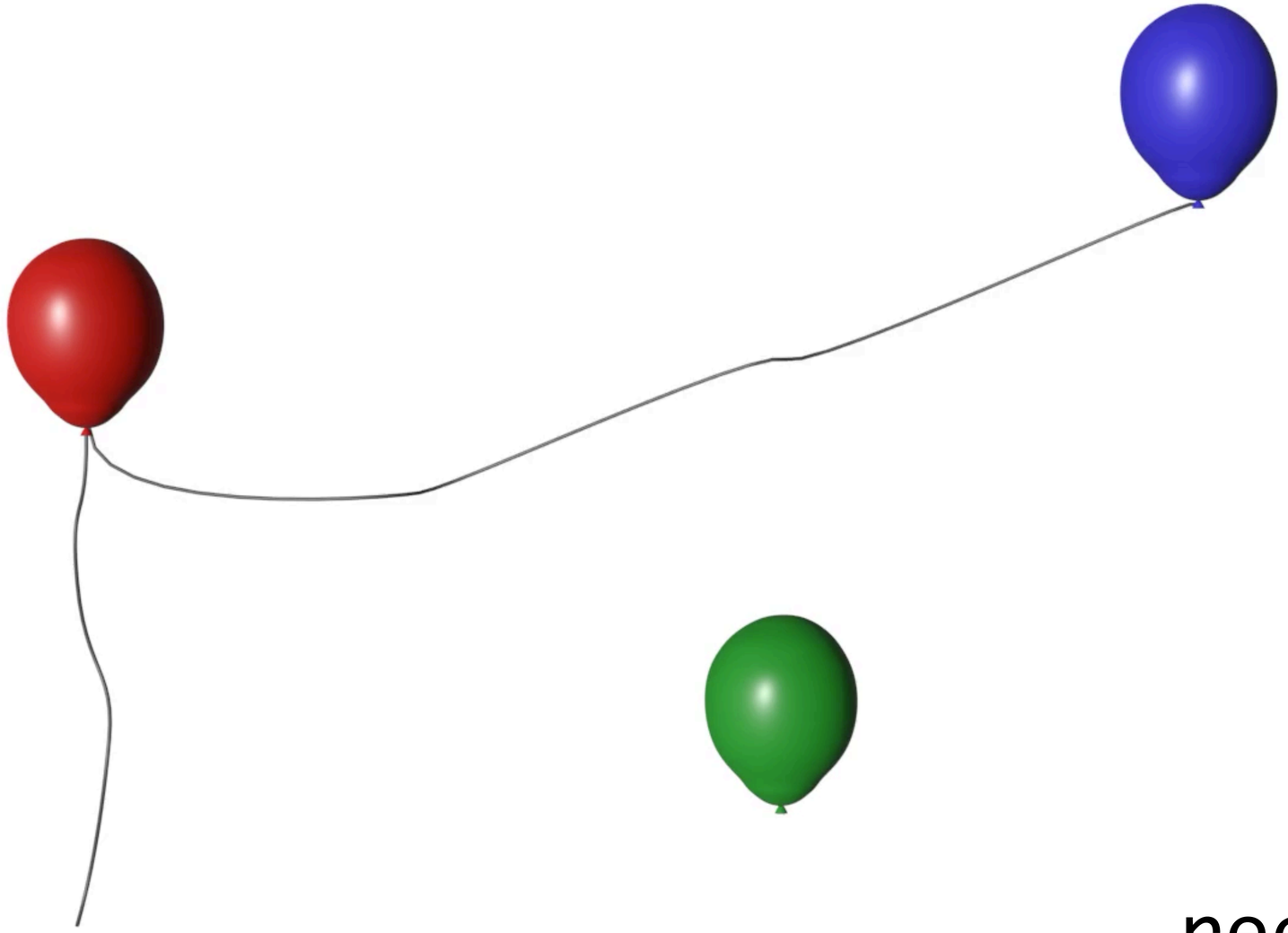


```
node *blue = malloc(sizeof(node));
```

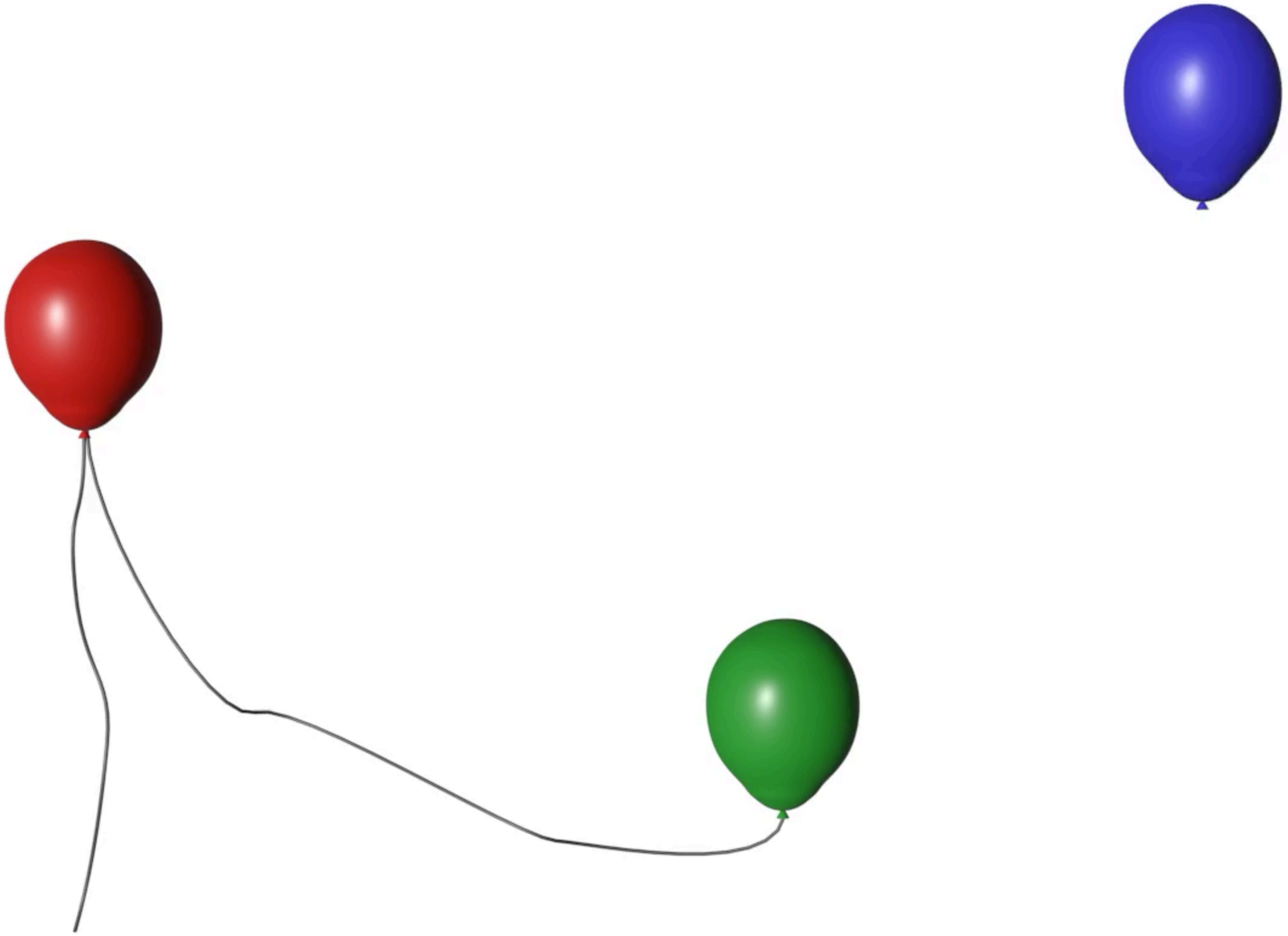
```
red->next = blue;
```

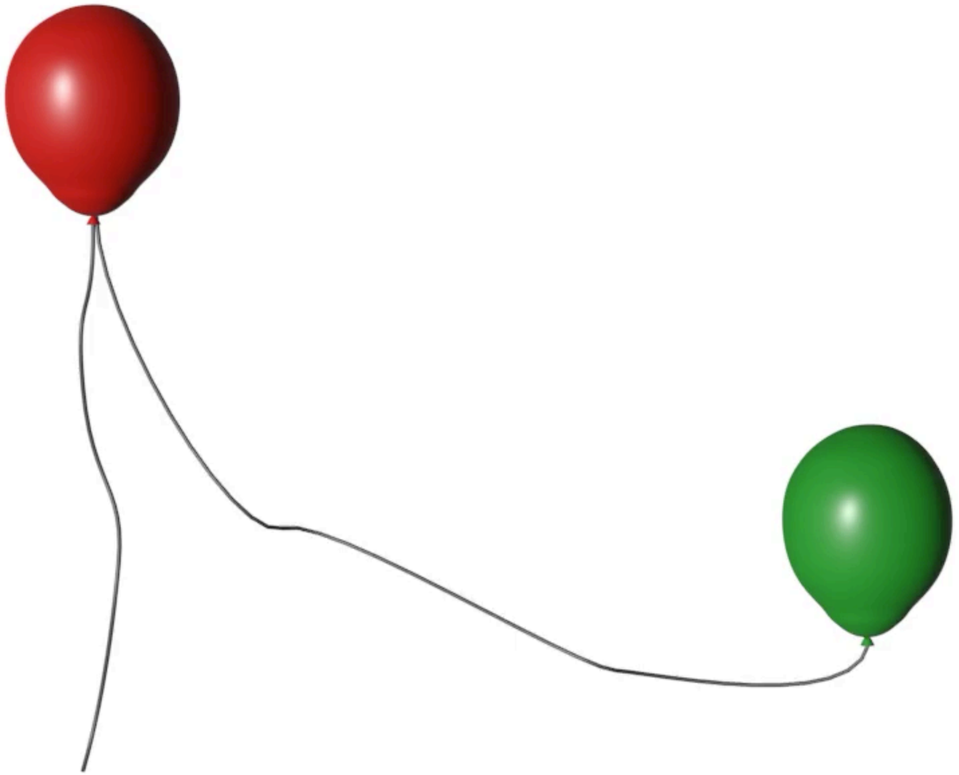


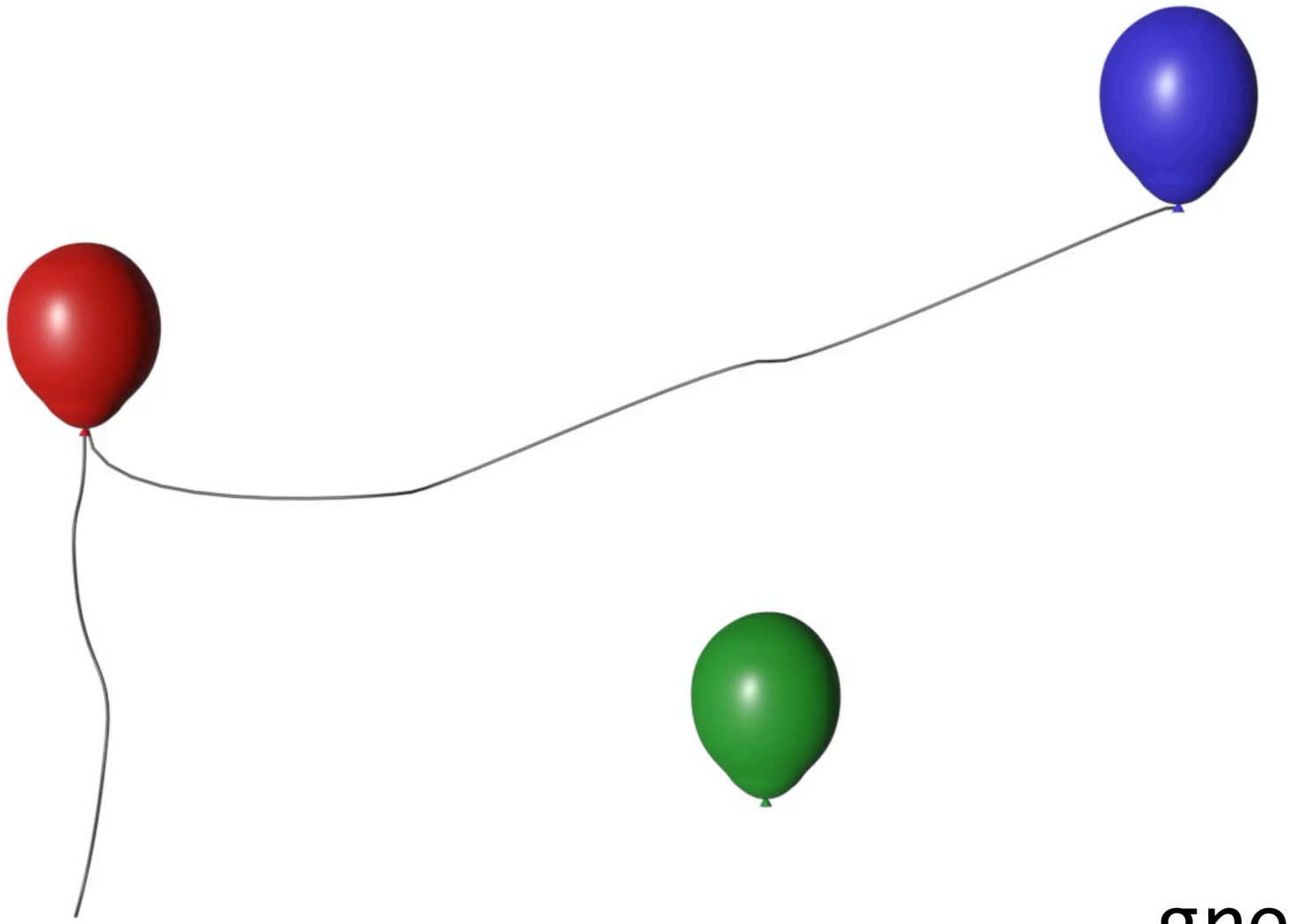
```
node *green = malloc(sizeof(node));
```



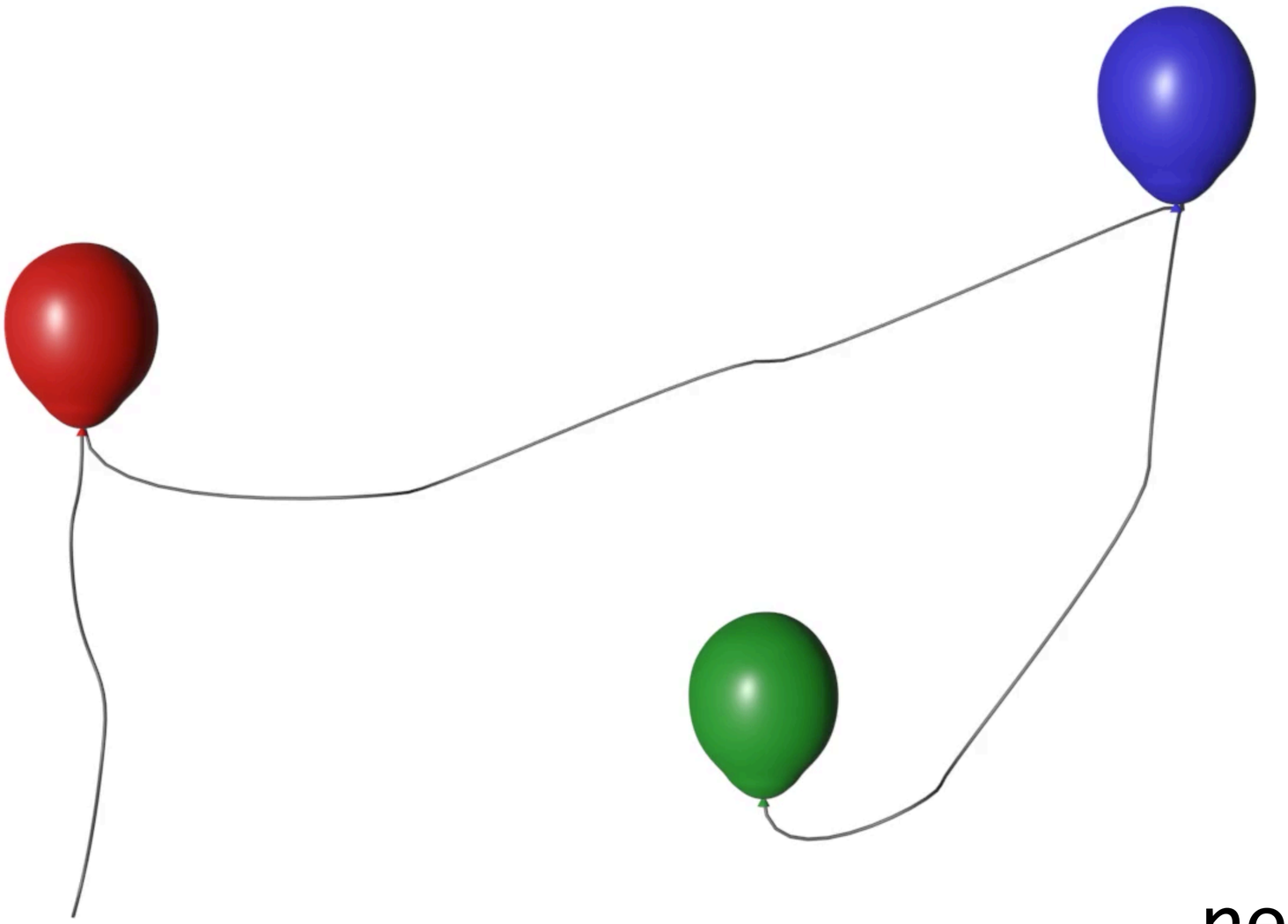
```
red->next = green;
```



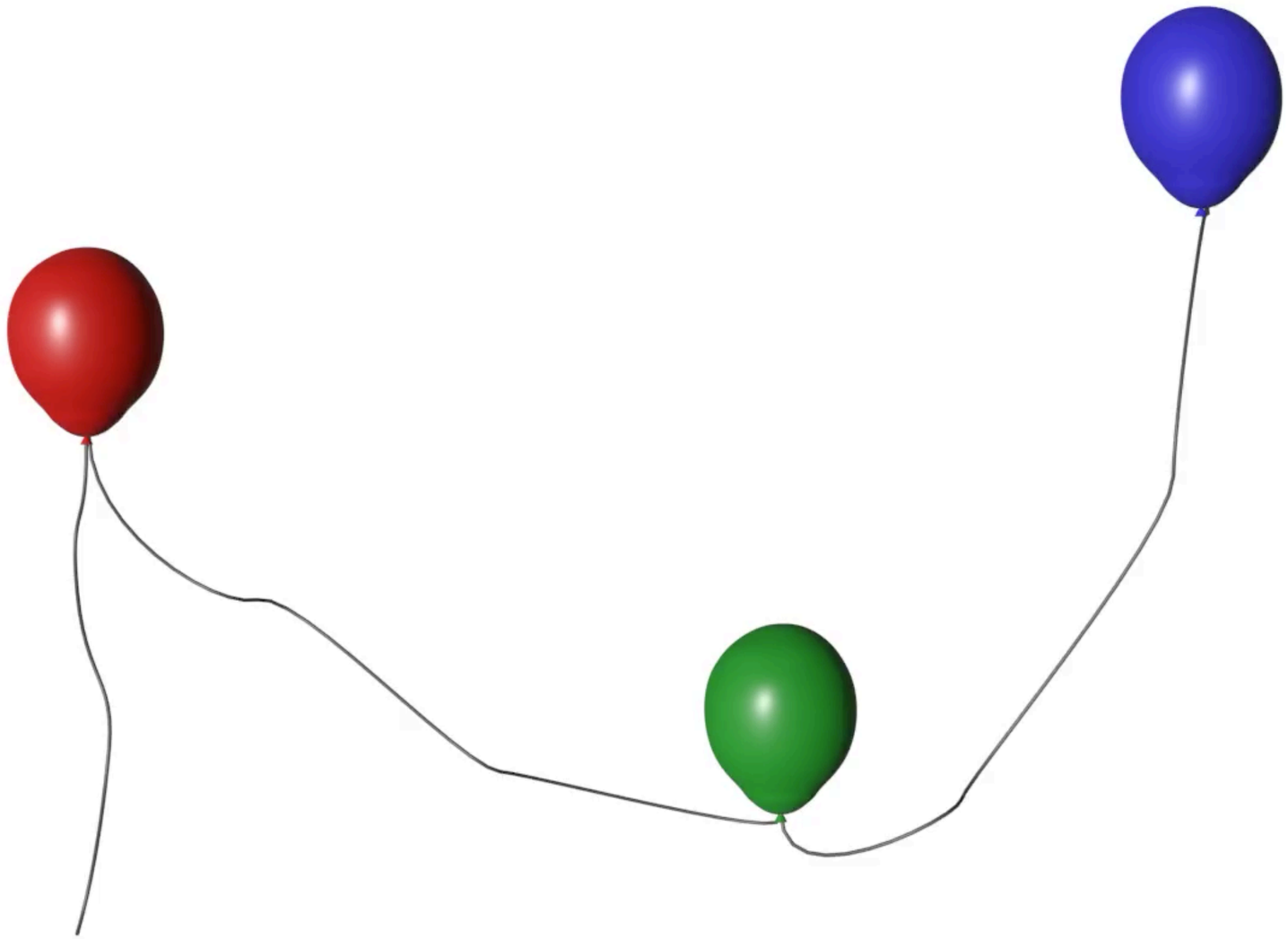




```
green->next = blue
```

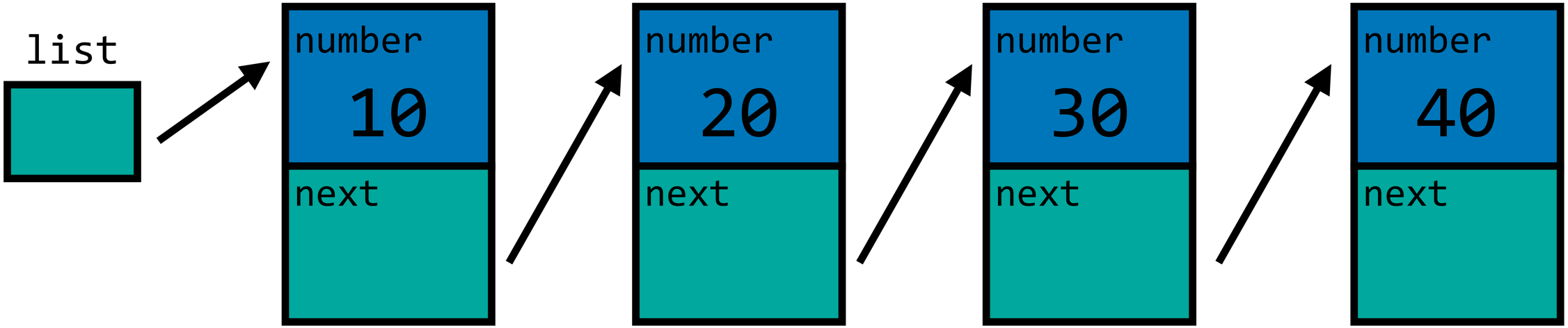


```
red->next = green
```

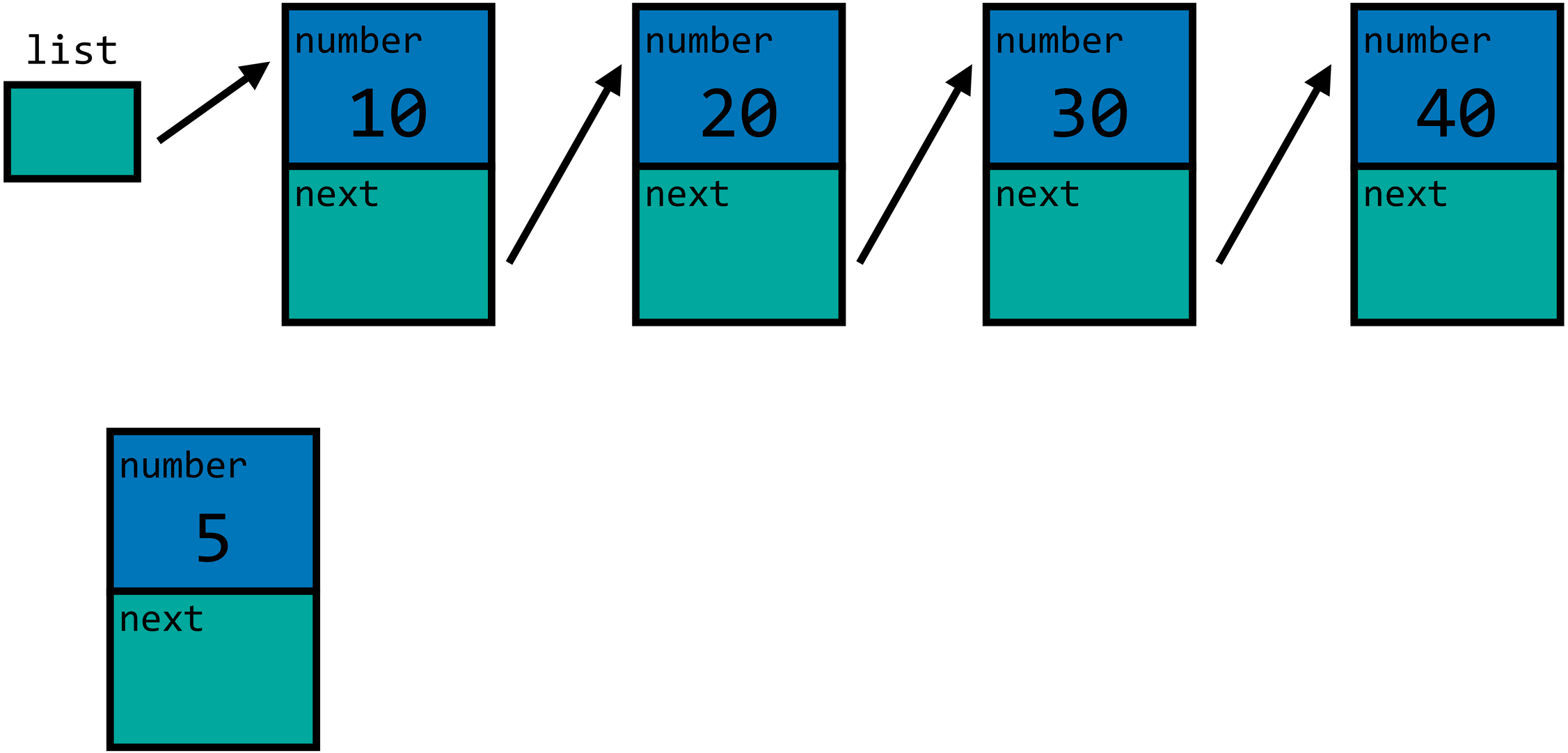


```
free(red);
free(green);
free(blue);
```

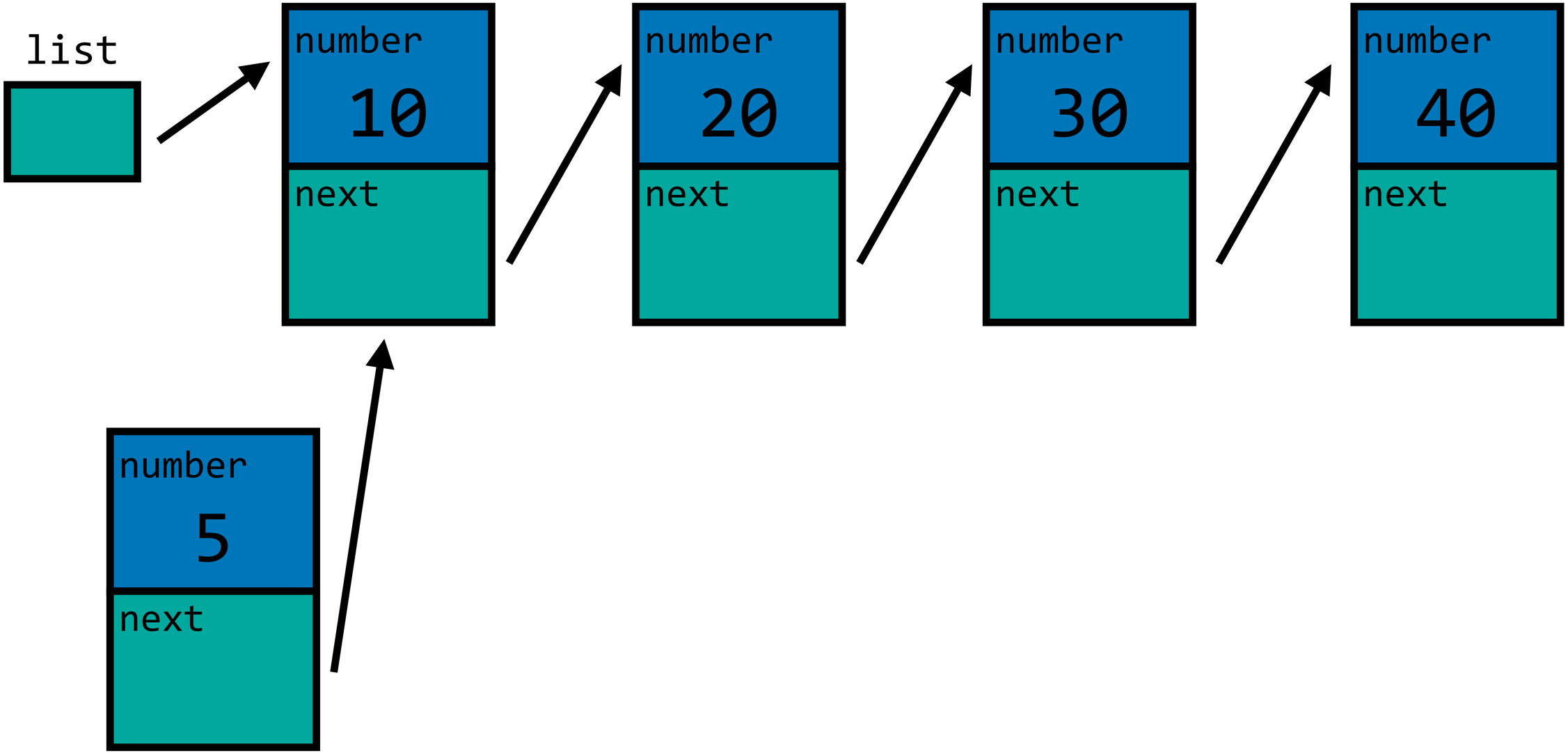
Insertion


list
number
10
next
number
20
next
number
30
next
number
40
next

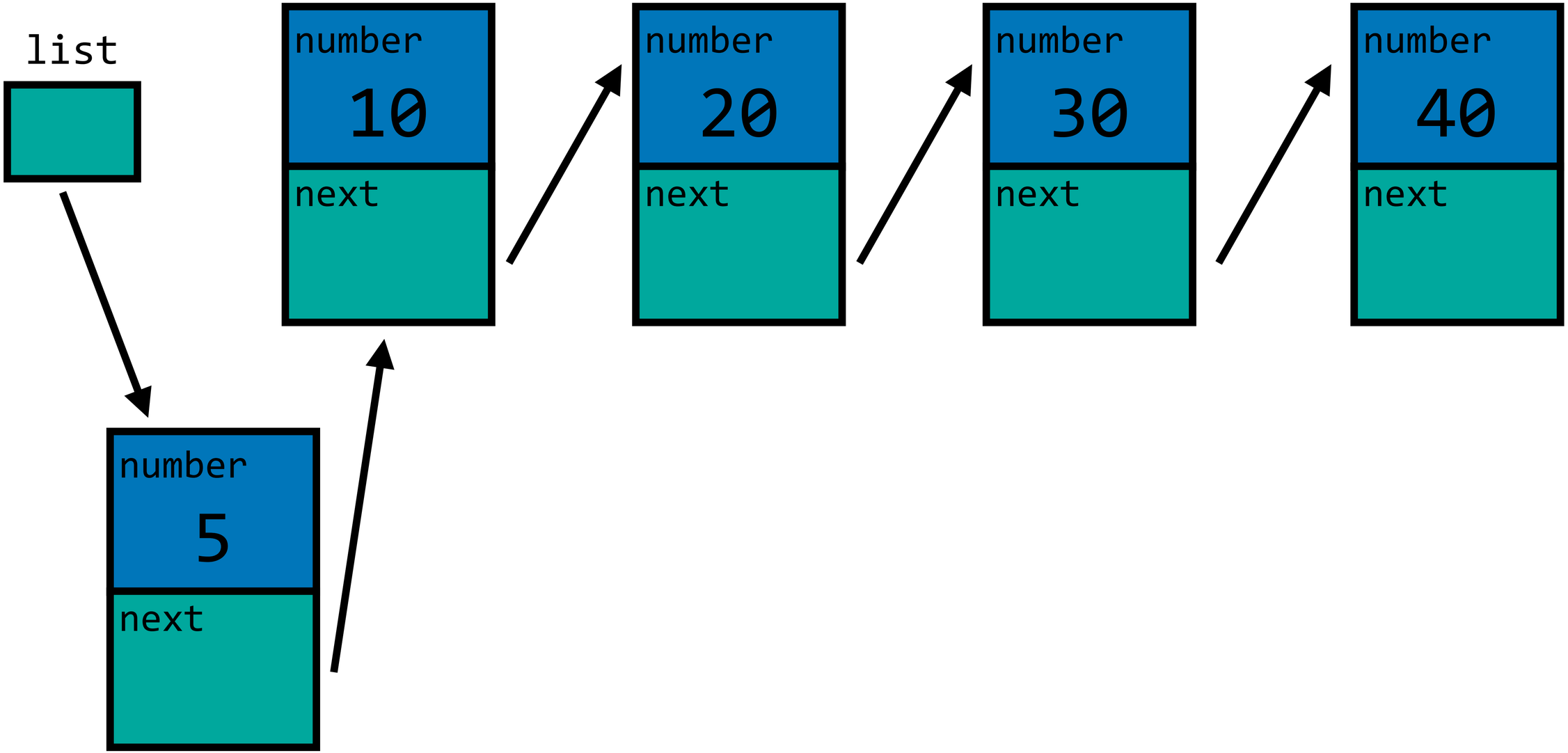
Insertion


list
number
10
next
number
20
next
number
30
next
number
40
next
number
5
next

Insertion

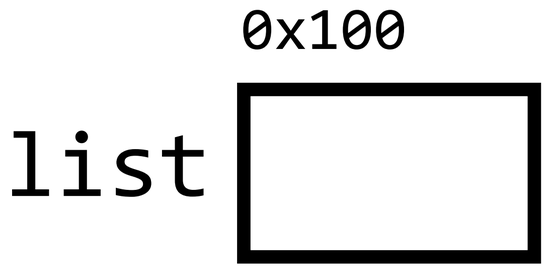

list
number
10
next
number
20
next
number
30
next
number
40
next
number
5
next

Insertion

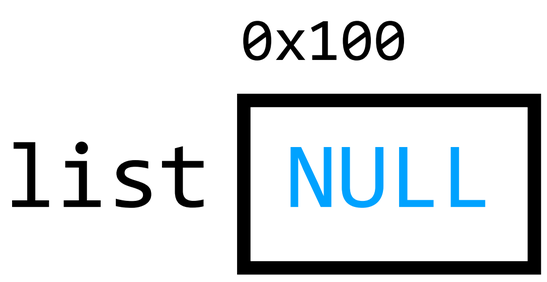

list
number
10
next
number
20
next
number
30
next
number
40
next
number
5
next

```
node *list = NULL;
```

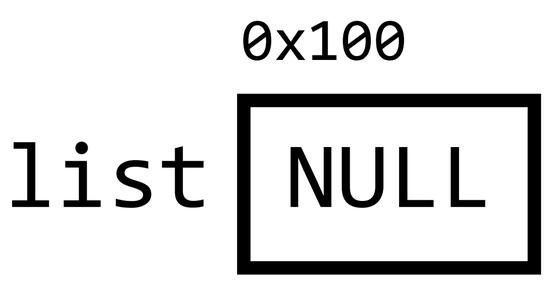
```
node *list = NULL;
```



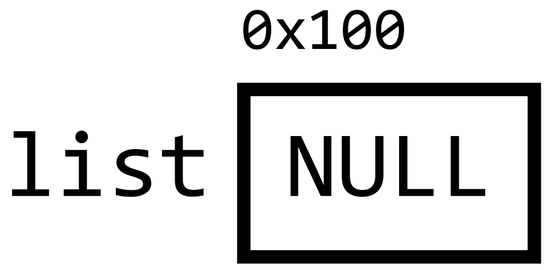
```
node *list = NULL;
```

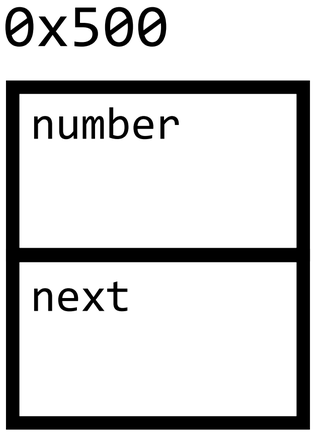


```
node *list = NULL;
node *n = malloc(sizeof(node));
```

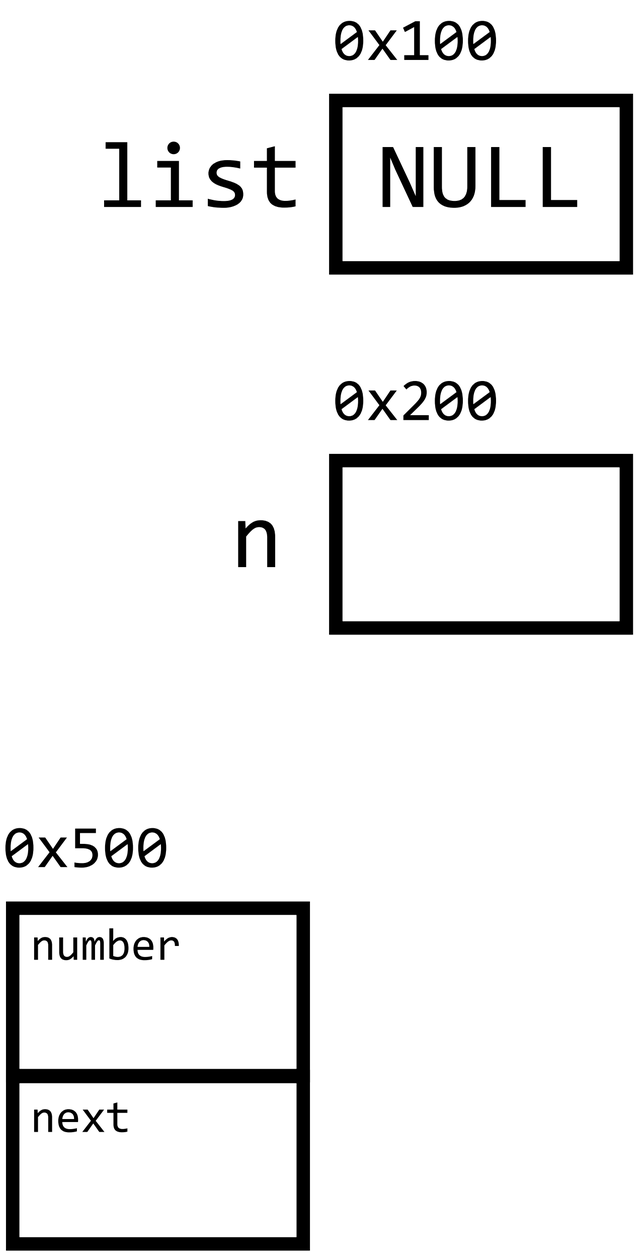


```
node *list = NULL;
node *n = malloc(sizeof(node));
```

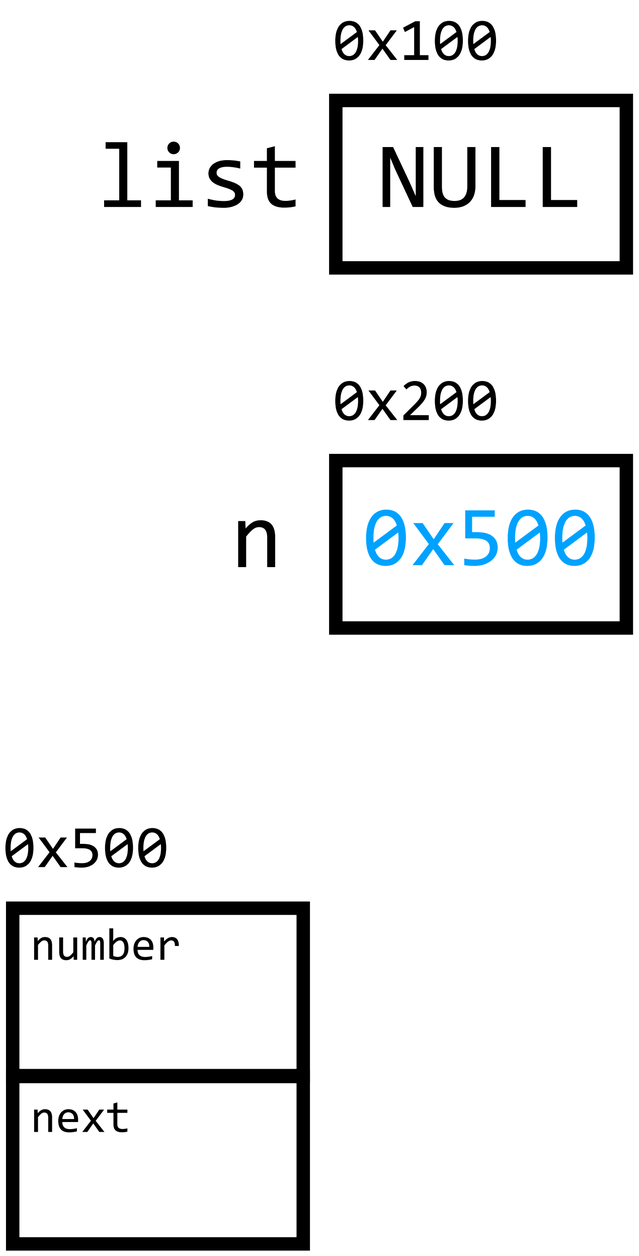




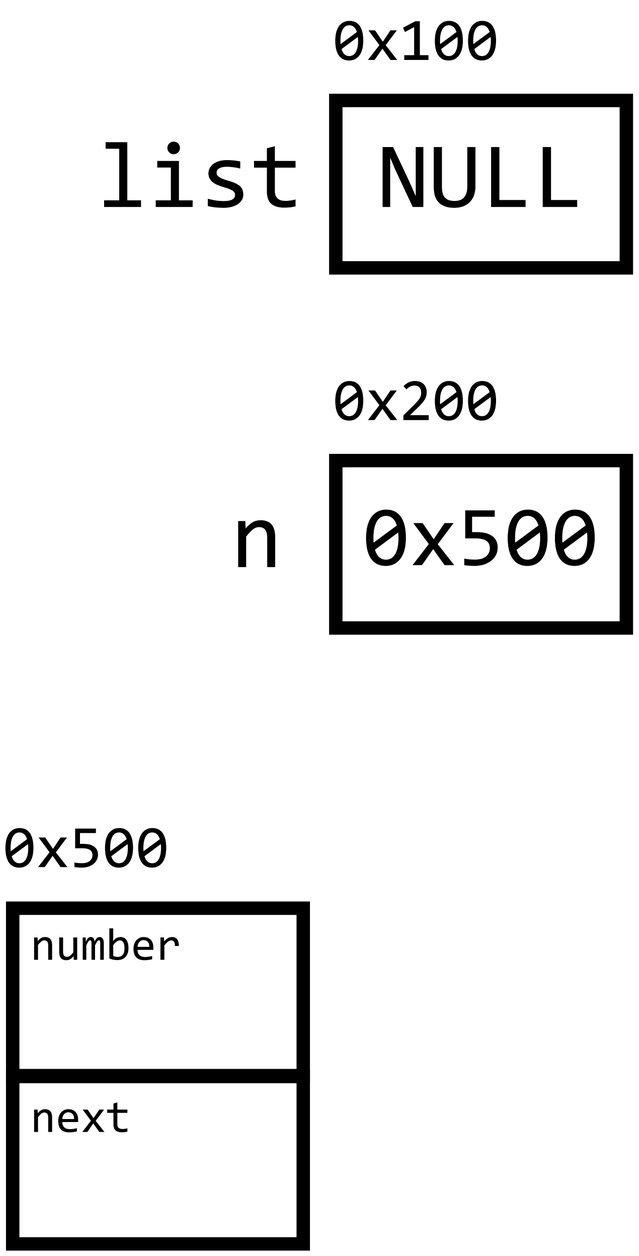
```
node *list = NULL;
node *n = malloc(sizeof(node));
```



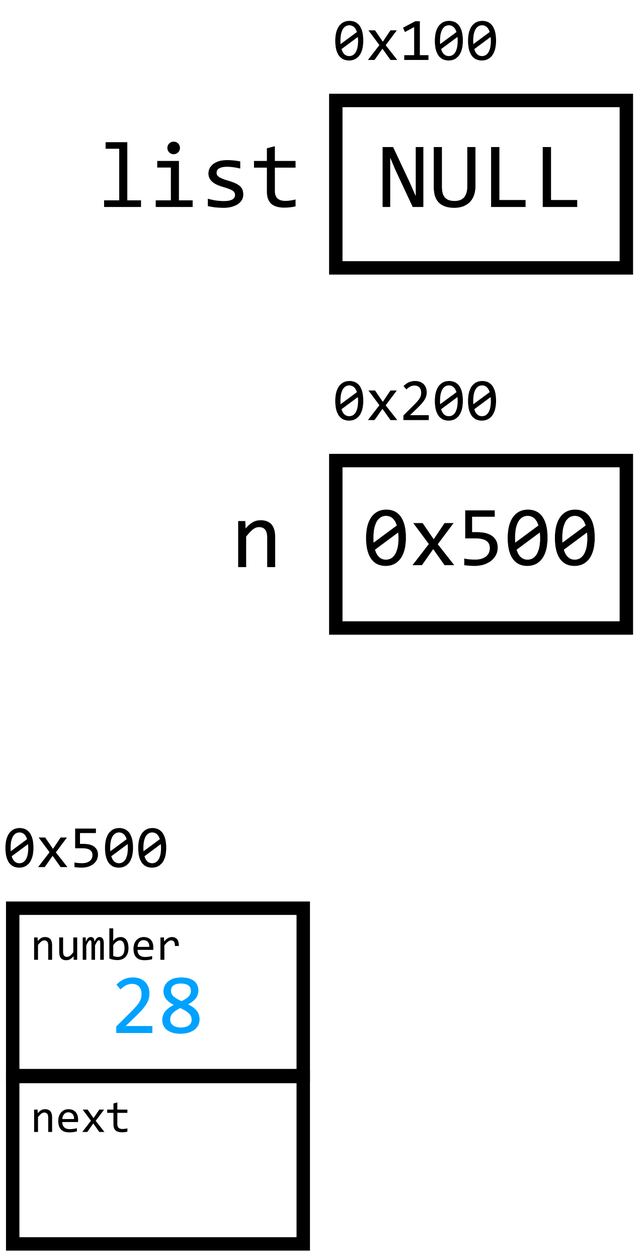
```
node *list = NULL;
node *n = malloc(sizeof(node));
```



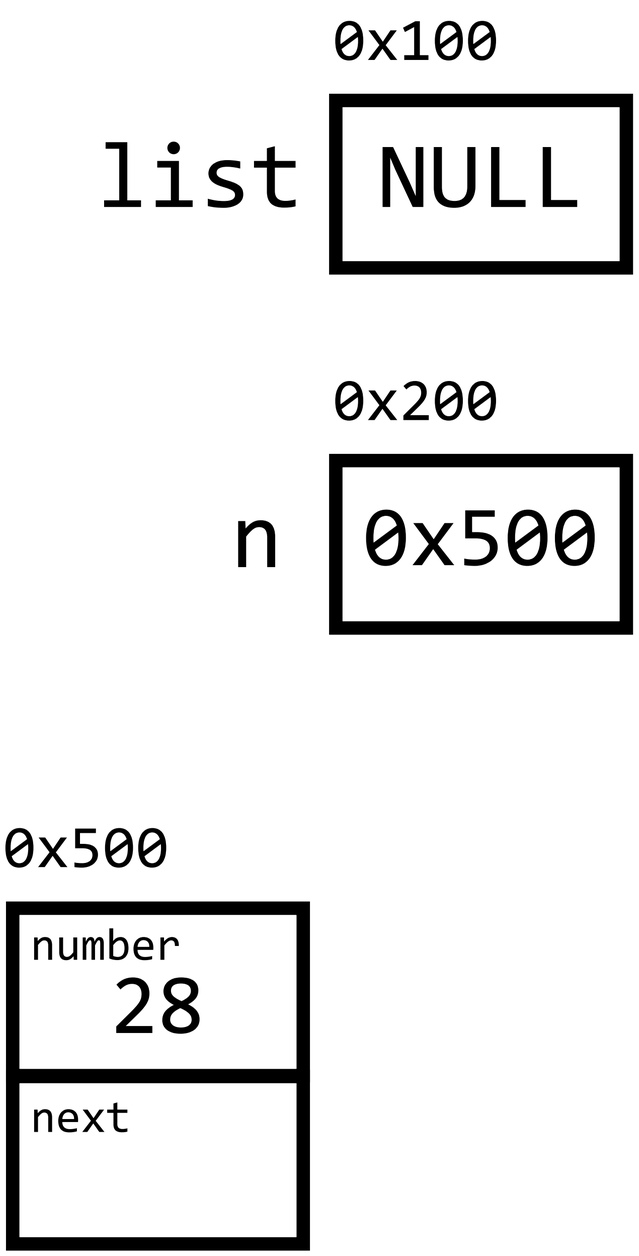
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
```



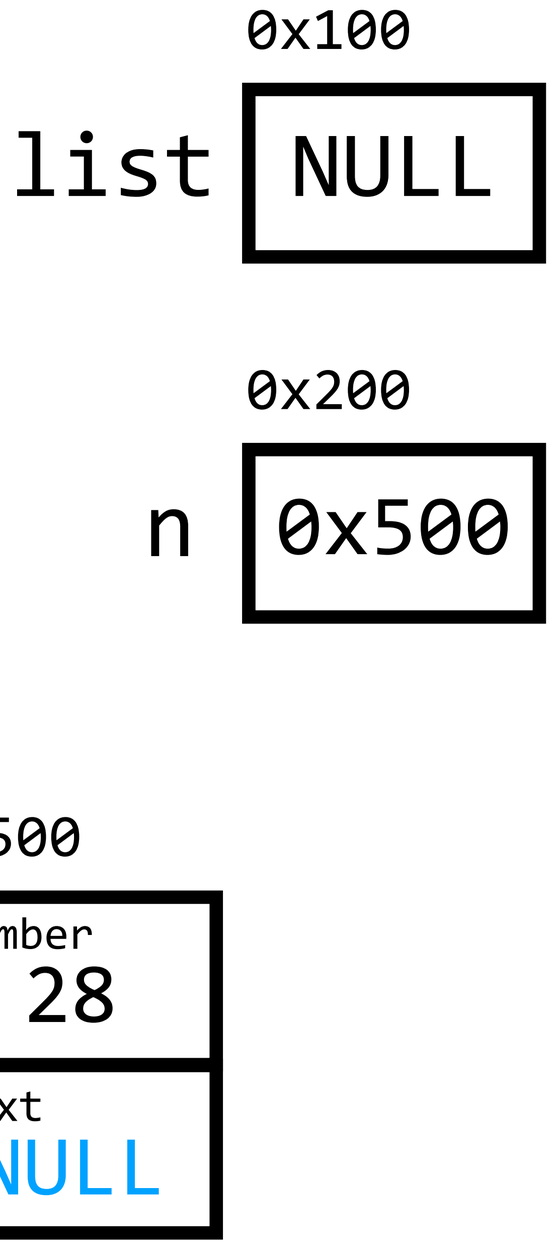
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
```



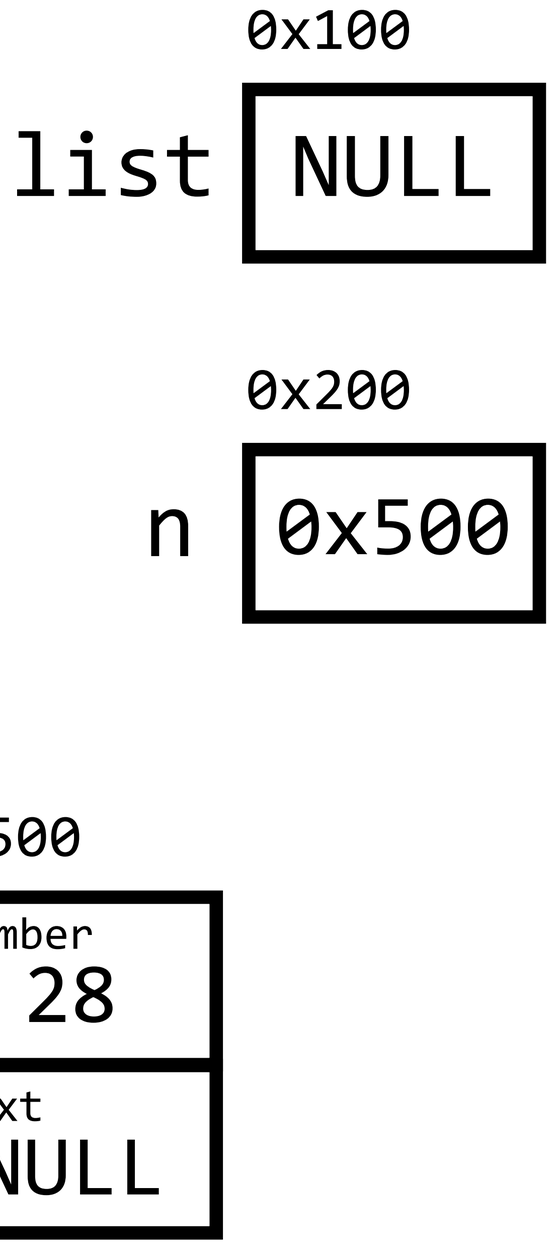
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
```



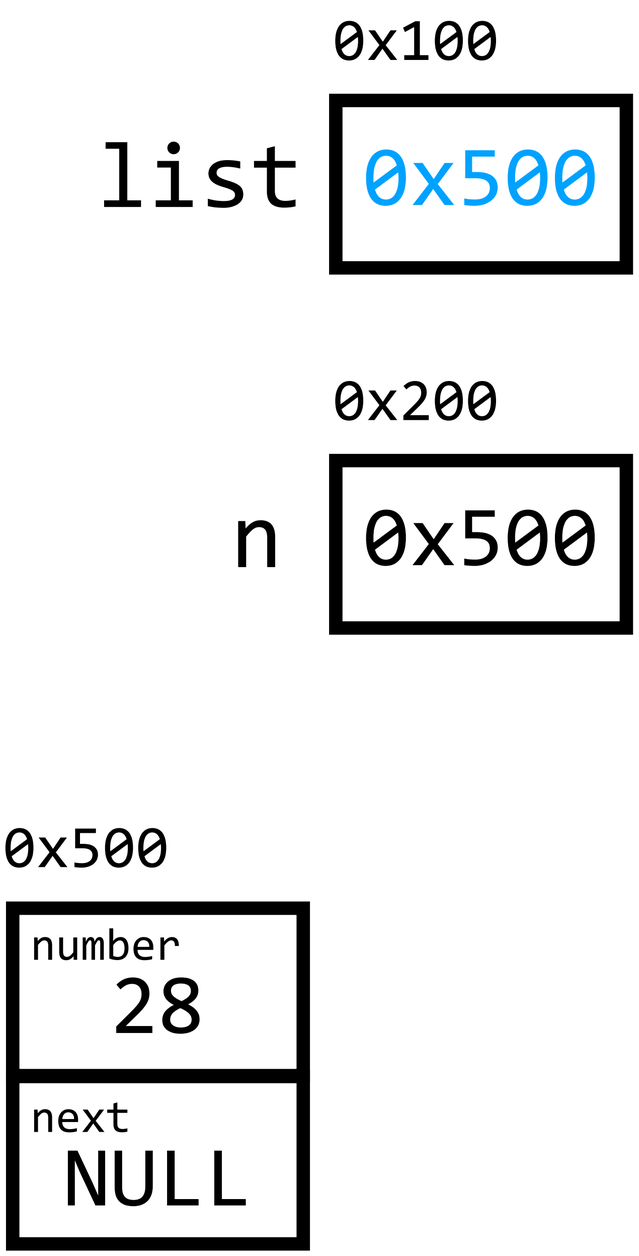
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;
```



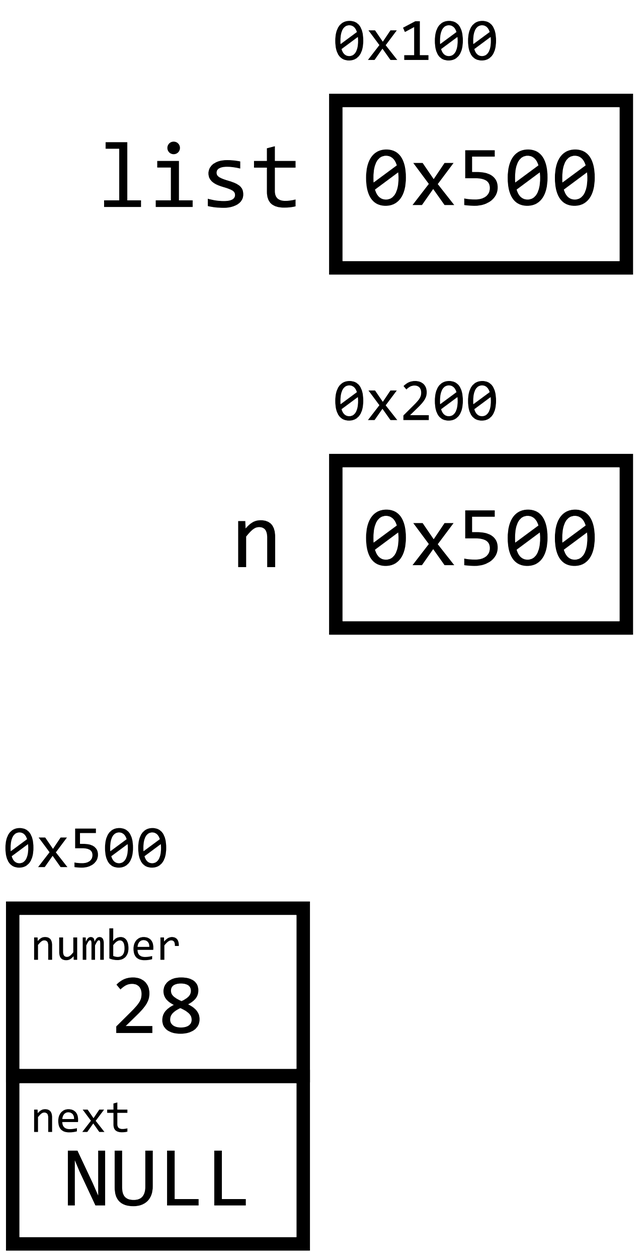
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;
```



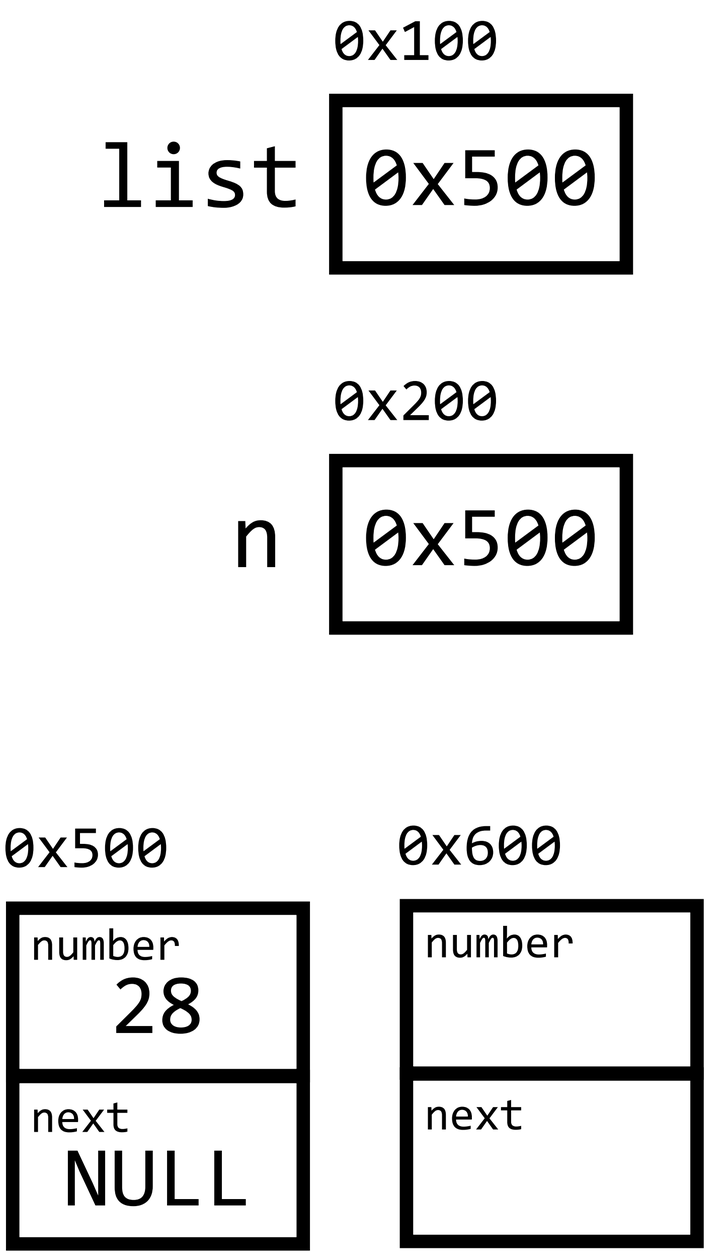
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;
```



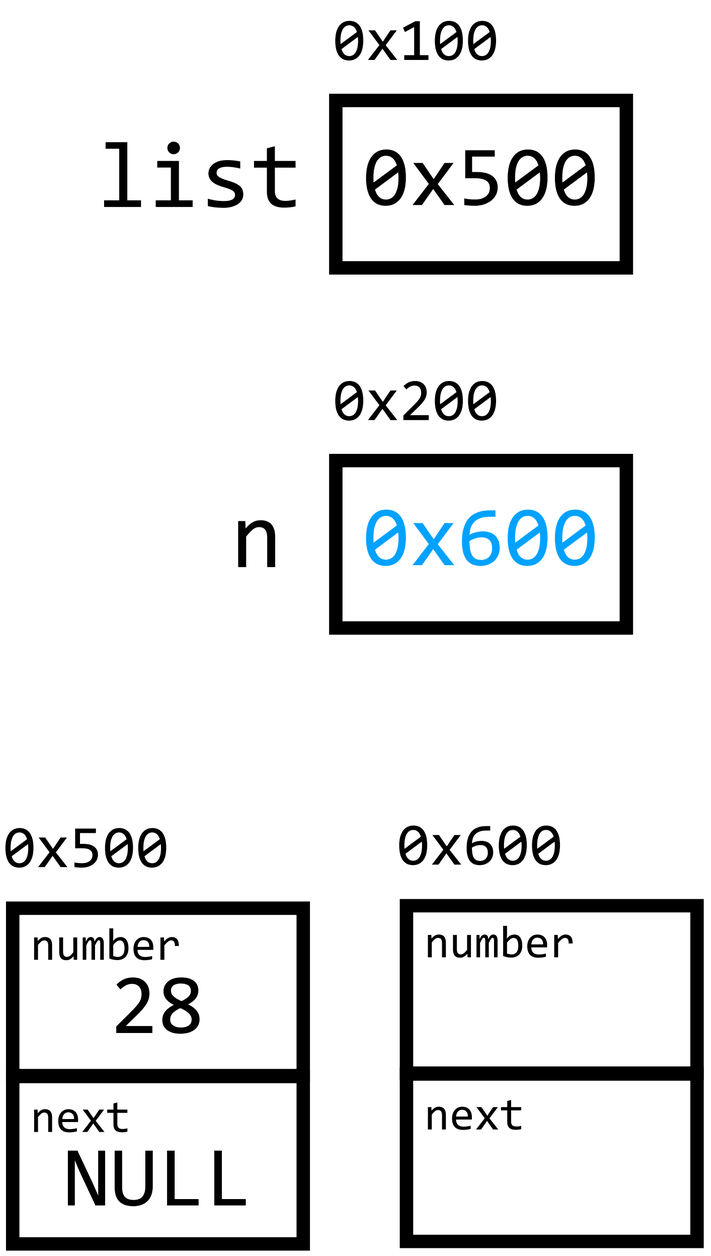
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
```



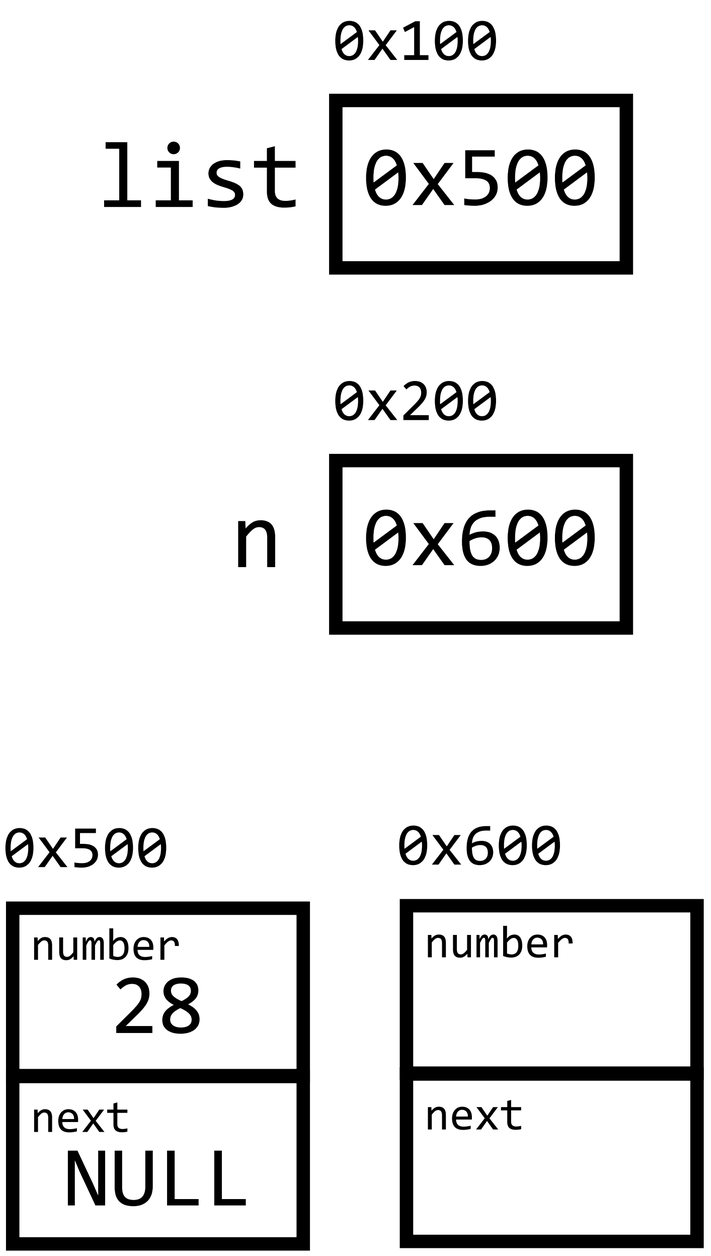
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
```



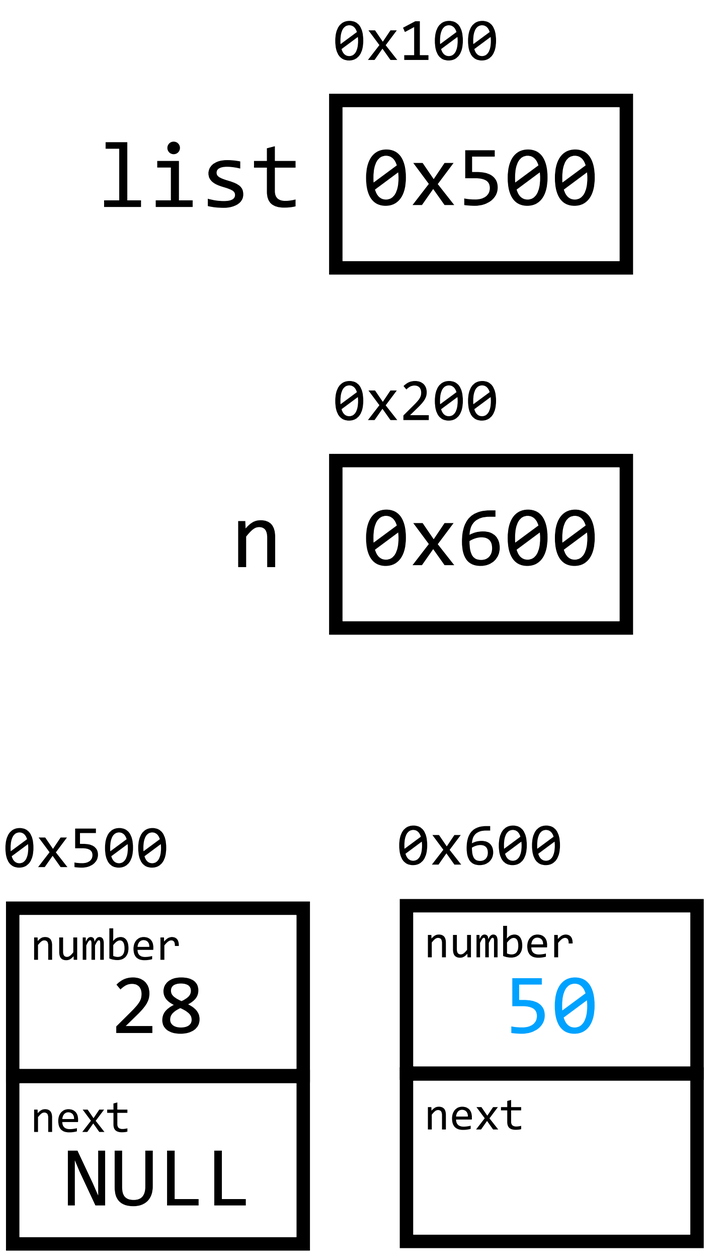
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
```



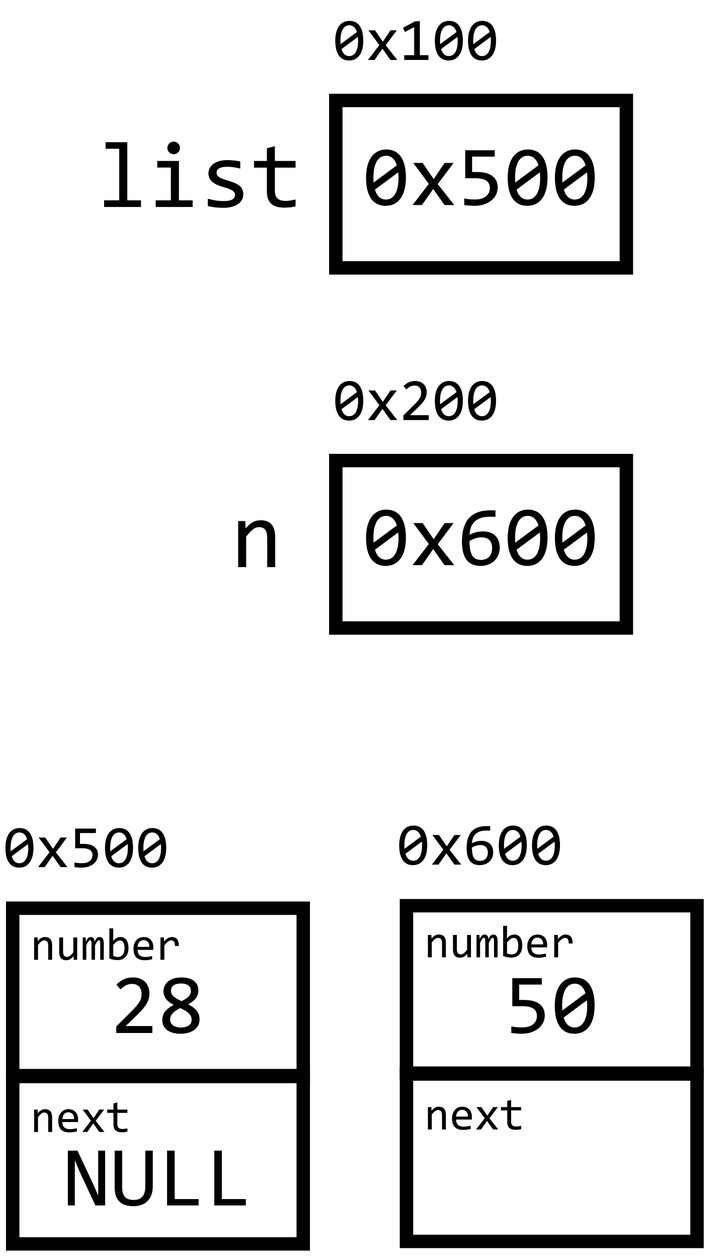
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
n->number = 50;
```



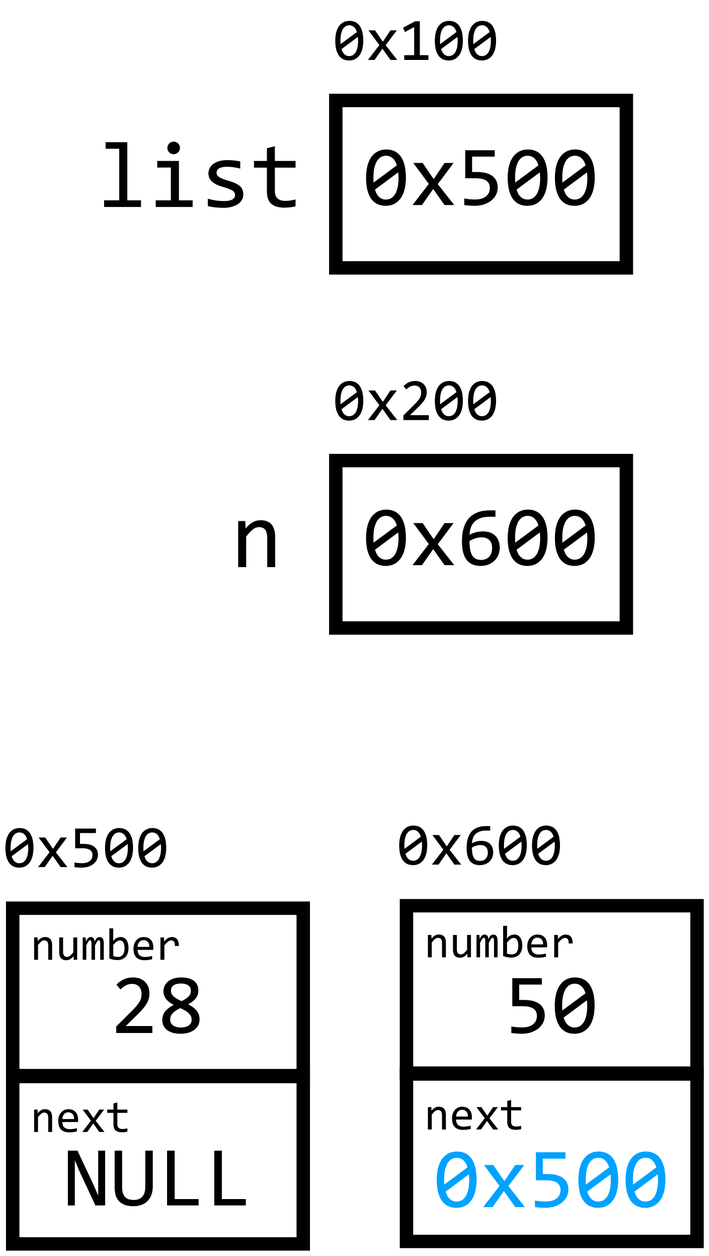
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
n->number = 50;
```



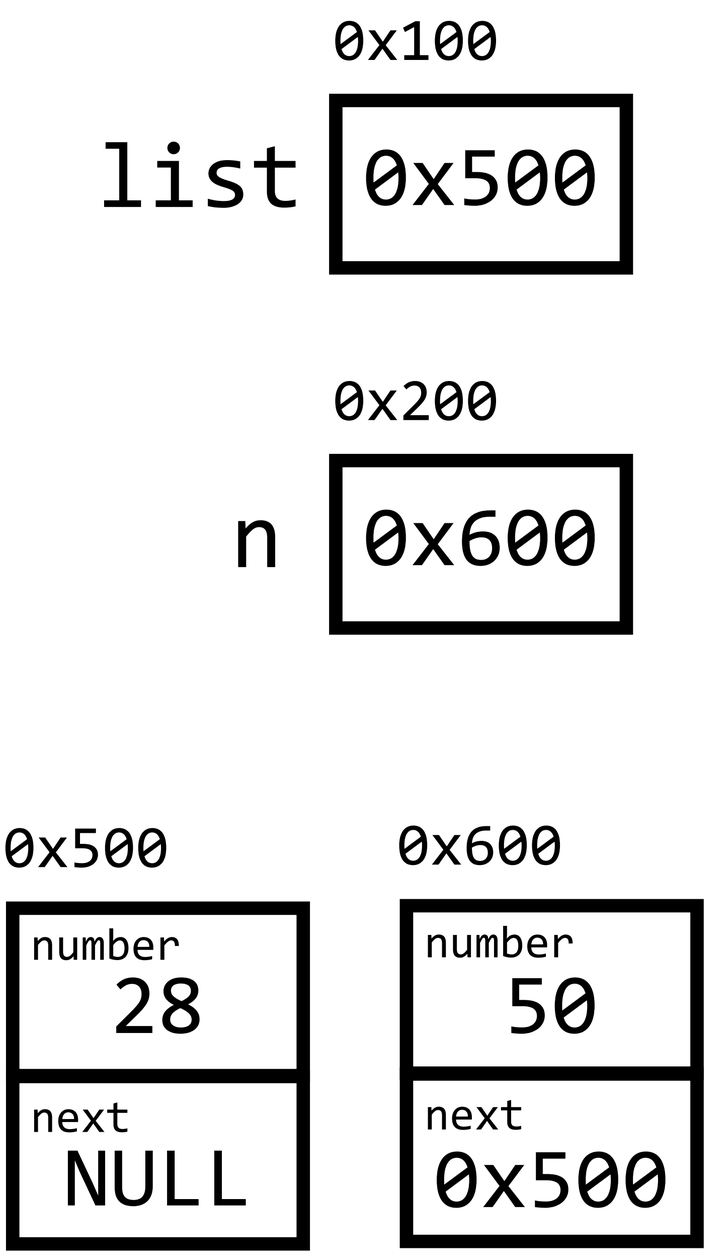
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
n->number = 50;
n->next = list;
```



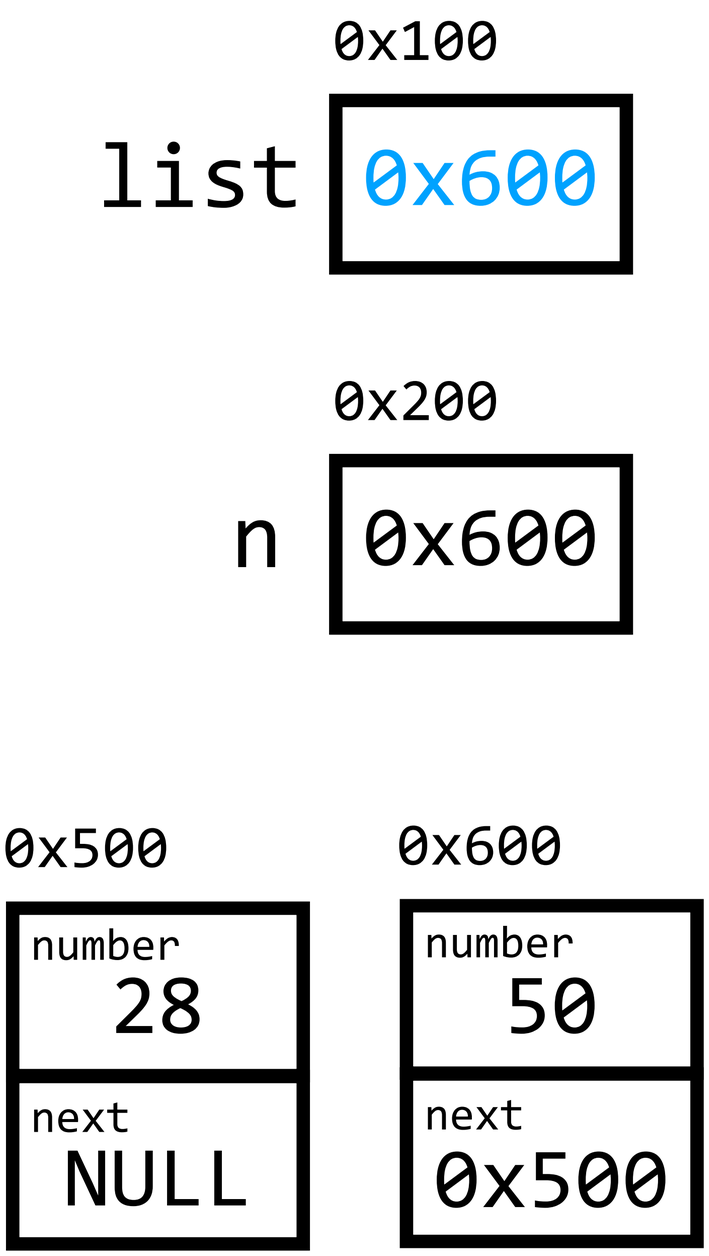
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
n->number = 50;
n->next = list;
```



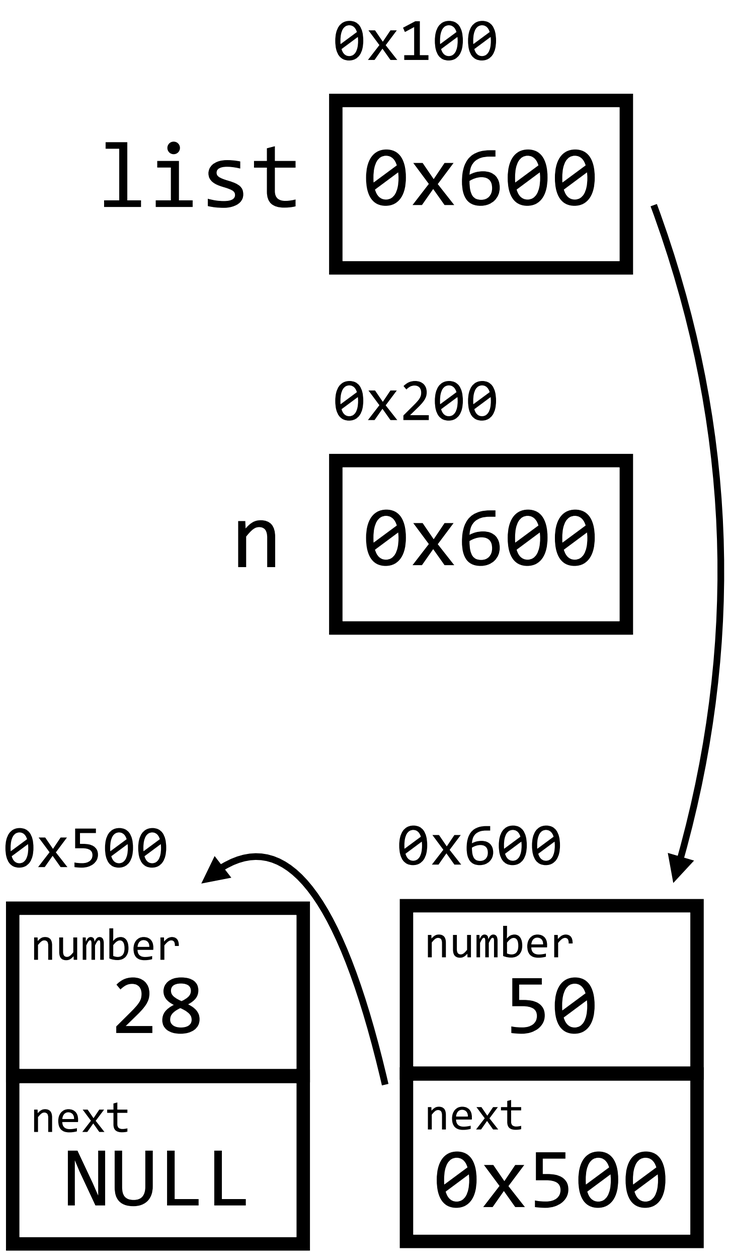
```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
n->number = 50;
n->next = list;

list = n;
```



```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
n->number = 50;
n->next = list;

list = n;
```



```
node *list = NULL;

node *n = malloc(sizeof(node));
n->number = 28;
n->next = NULL;

list = n;

n = malloc(sizeof(node));
n->number = 50;
n->next = list;

list = n;
```



Exercise

Download distribution code at
`https://cs50.brianyu.me/list.c`

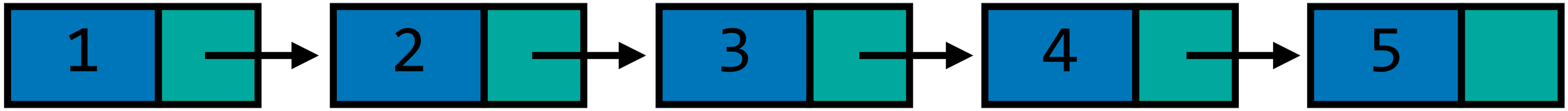
Update `list.c` to:

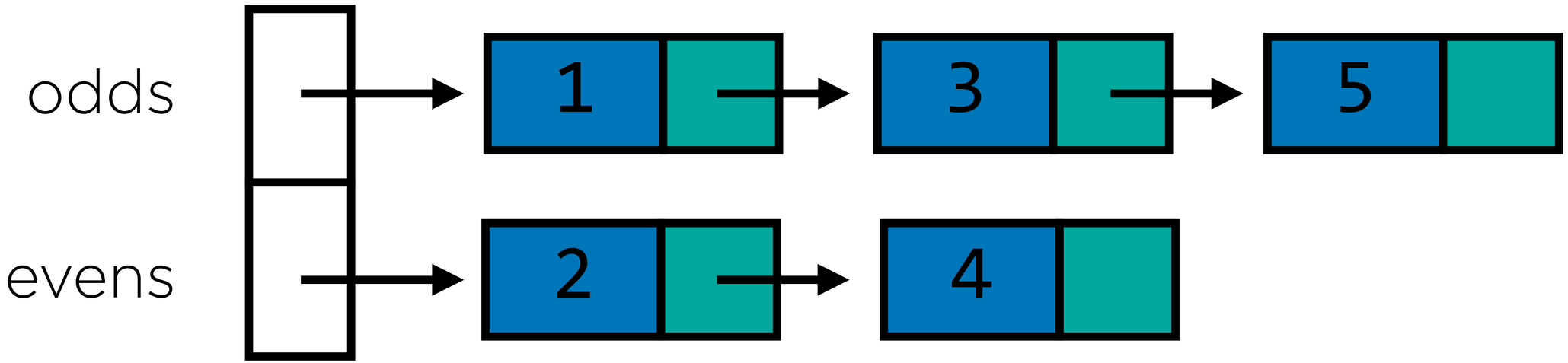
1) Create a new node and store the number inside.
2) Add the node to the `list`.
3) When all nodes are added, print the value in each node.
4) Free all nodes.

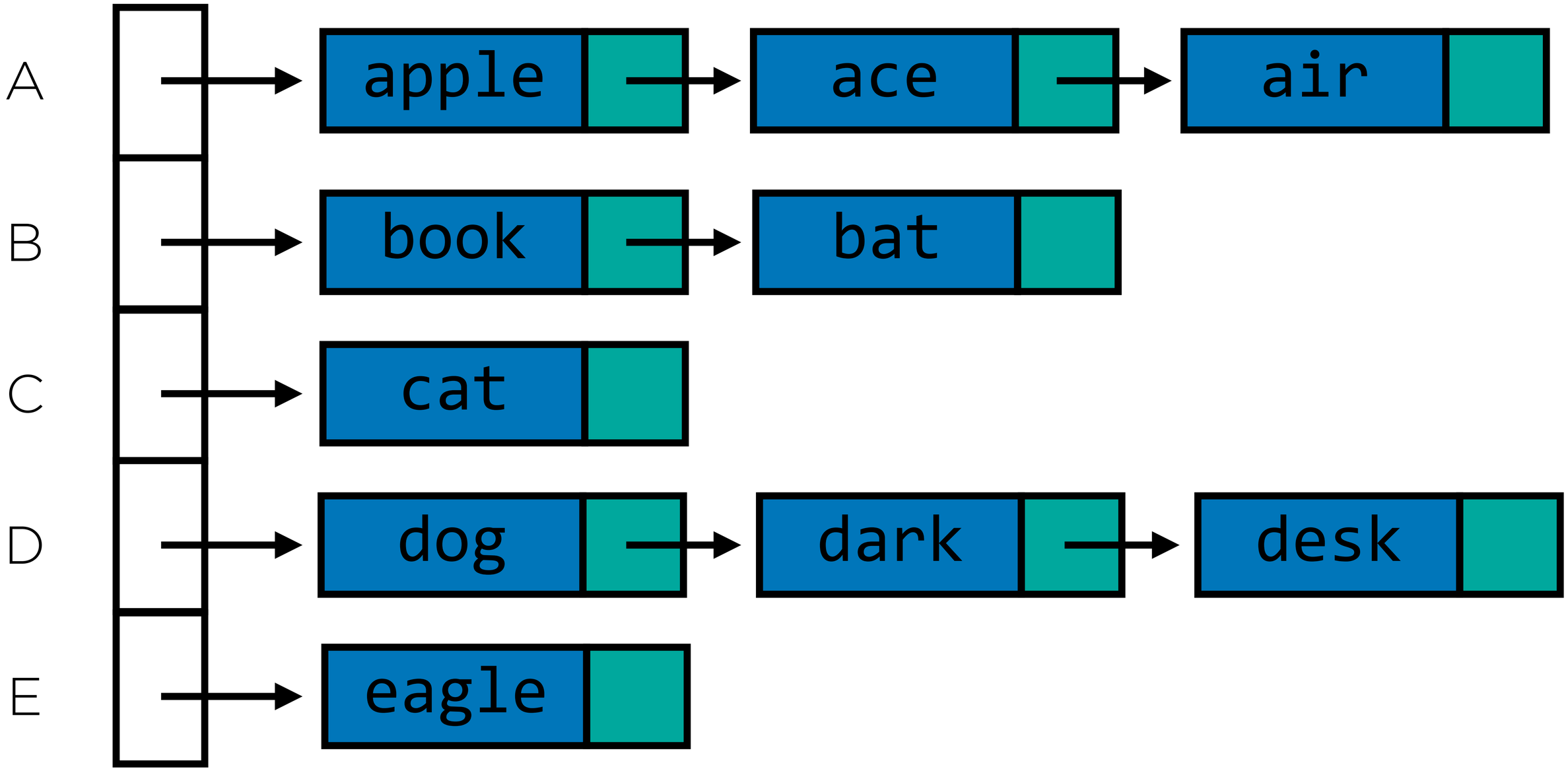
Part Two

Hash Tables, Trees, and Tries

Hash Tables


1
2
3
4
5


odds
evens
1
3
5
2
4


A
B
C
D
E
apple
ace
air
book
bat
cat
dog
dark
desk
eagle

Hash Table

- Array of linked lists
- Use a **hash function** to take an input, and pick a corresponding linked list

```
int hash(char *s)
{
    return s[0] - 'A';
}
```

Hash Function

- Deterministic: always maps same input to the same output
- Minimize collisions: fewer collisions means shorter linked lists

Linked List

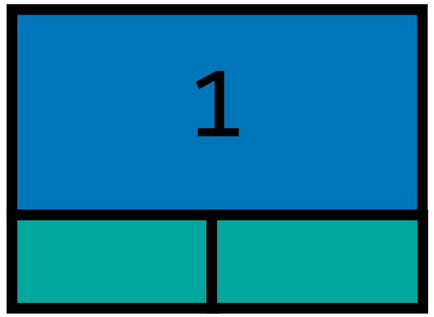
```
node *list;
```

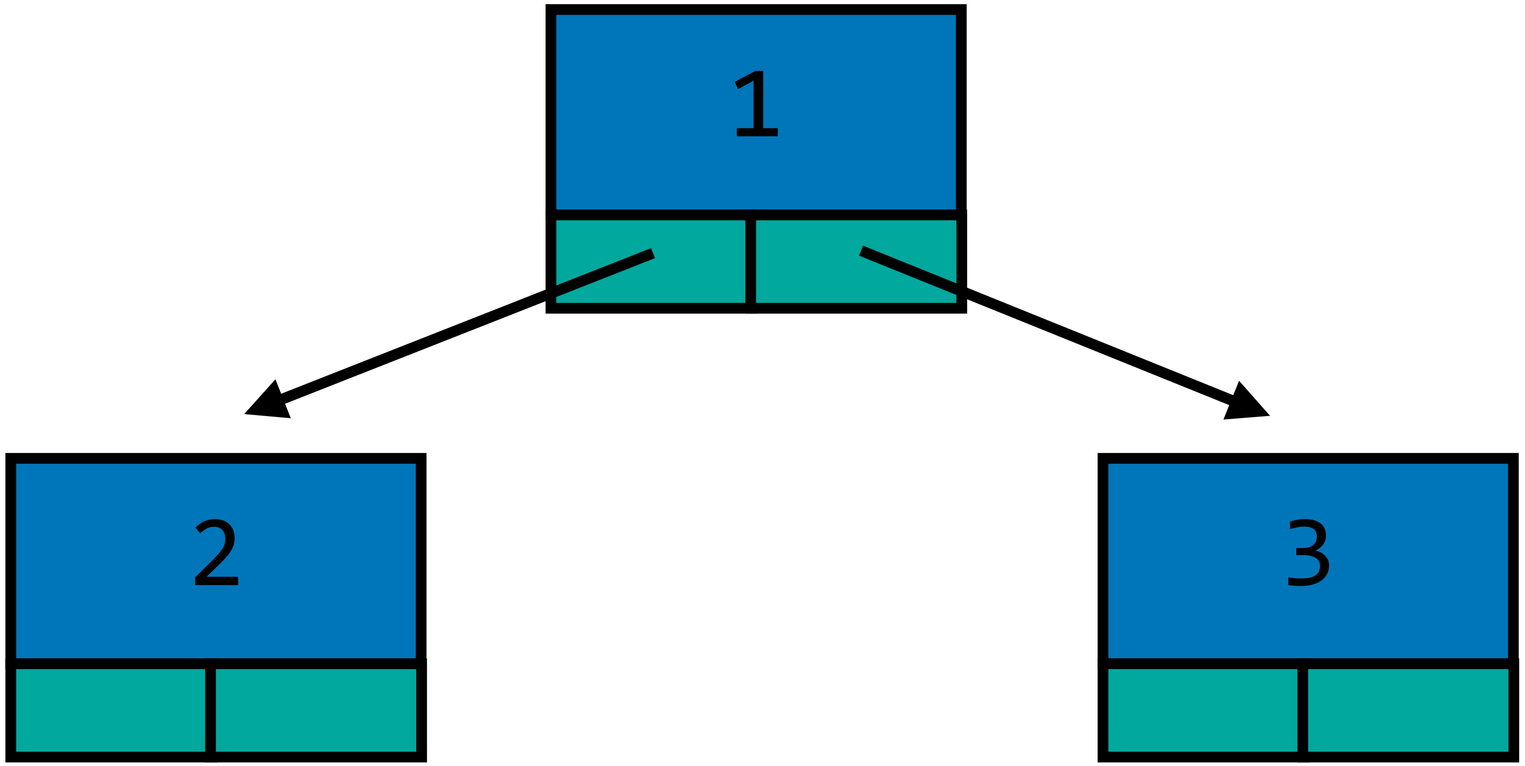
Hash Table

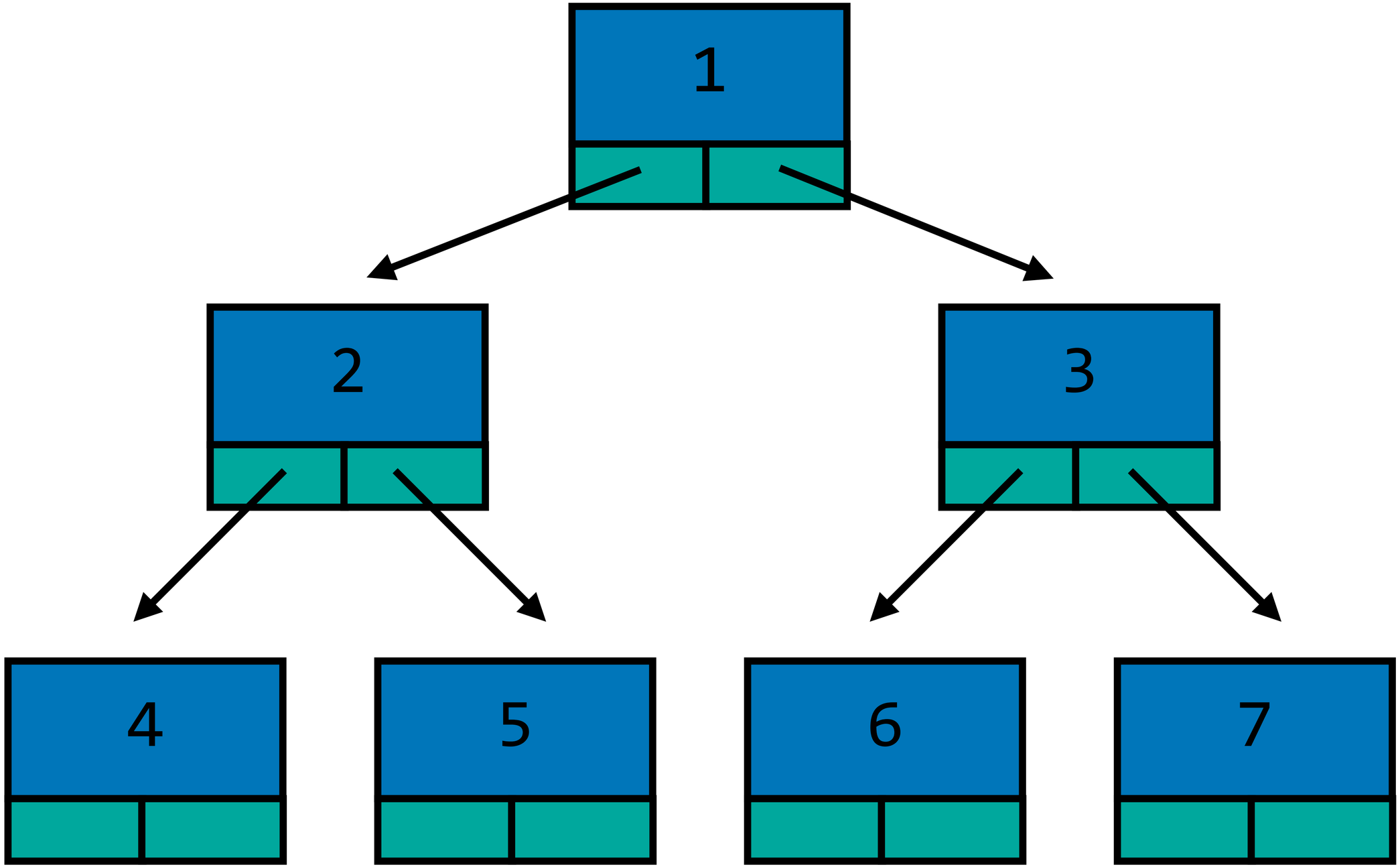
```
node *table[50];
```

Trees

Binary Trees

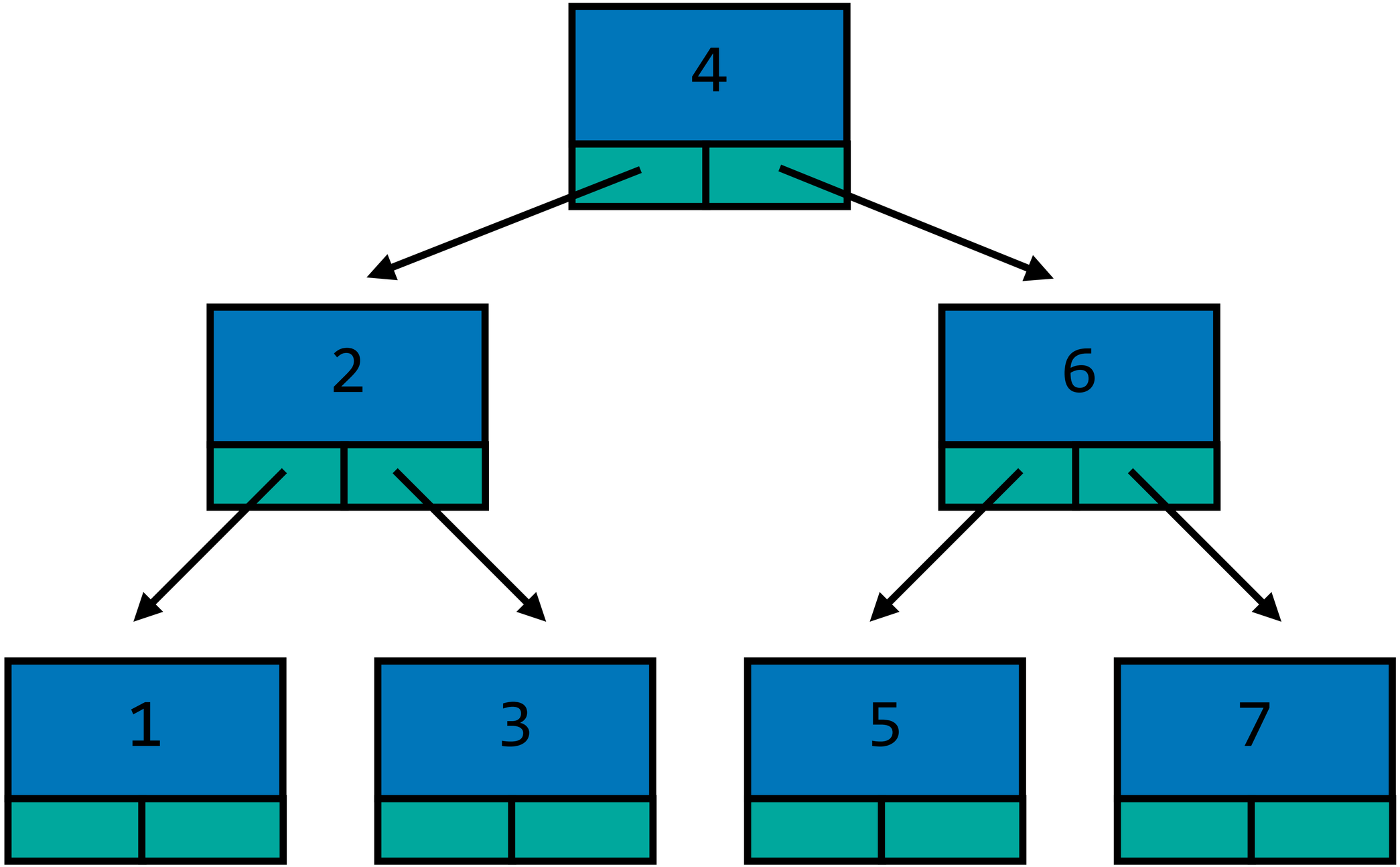

1

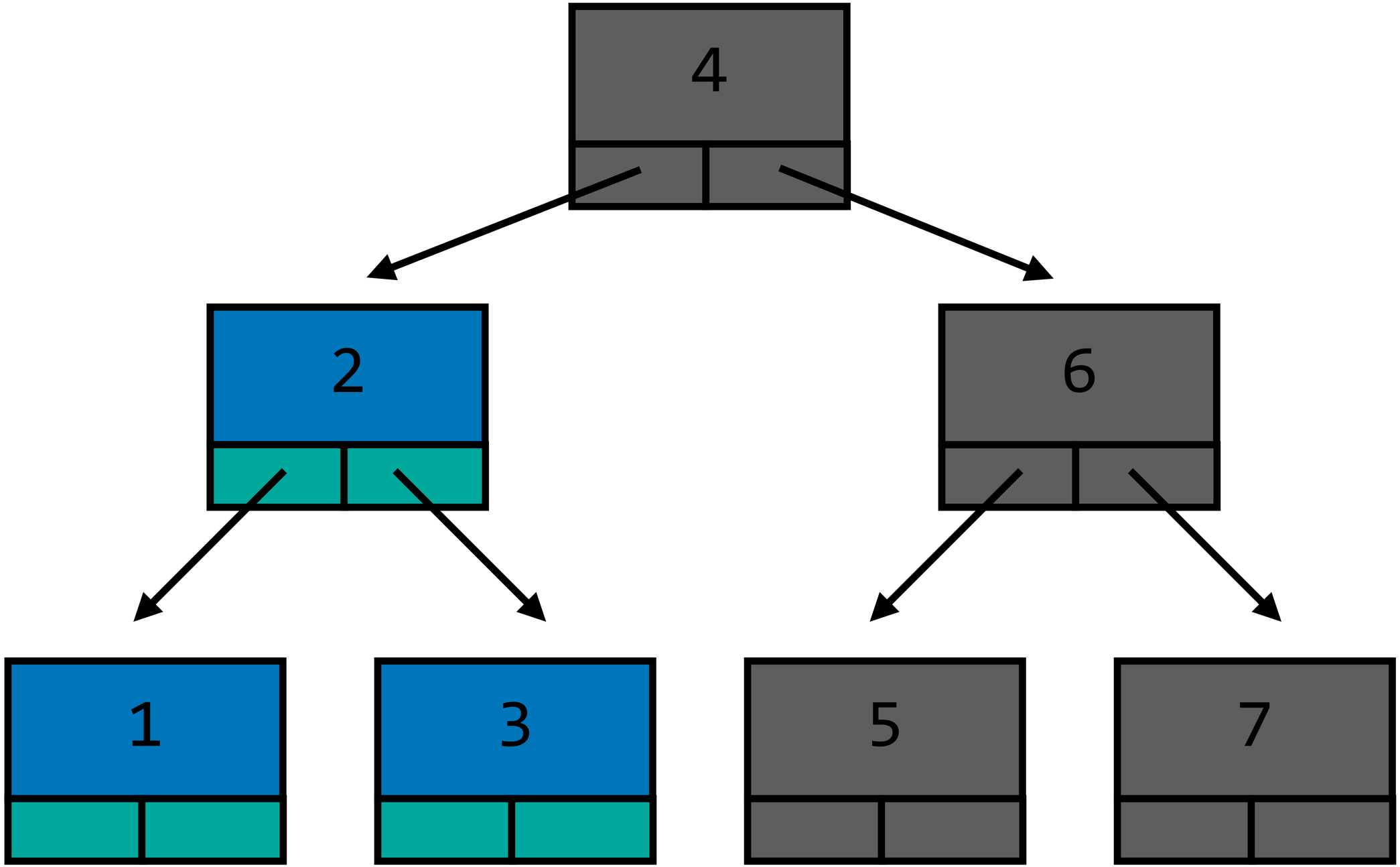

1
2
3

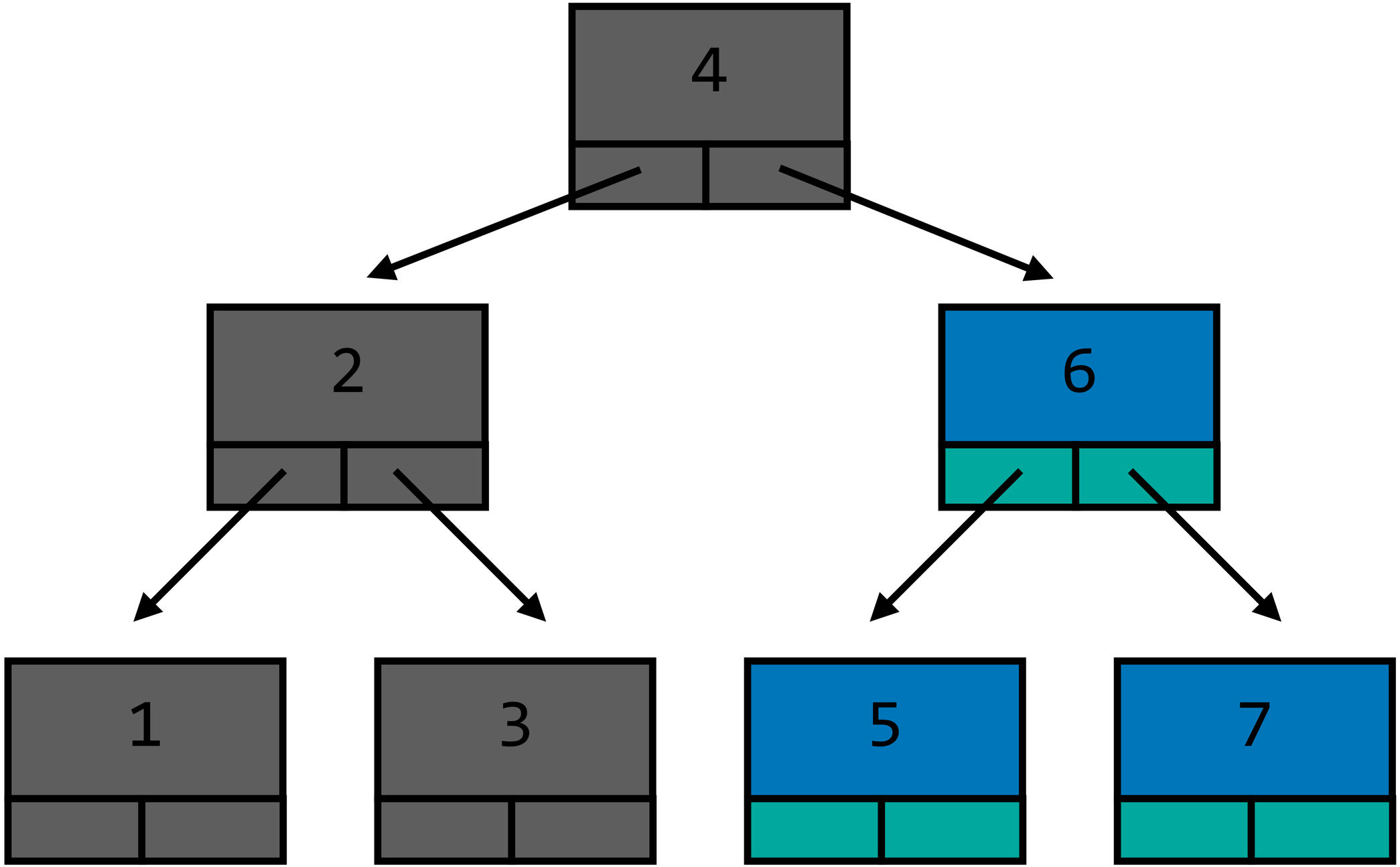

1
2
3
4
5
6
7

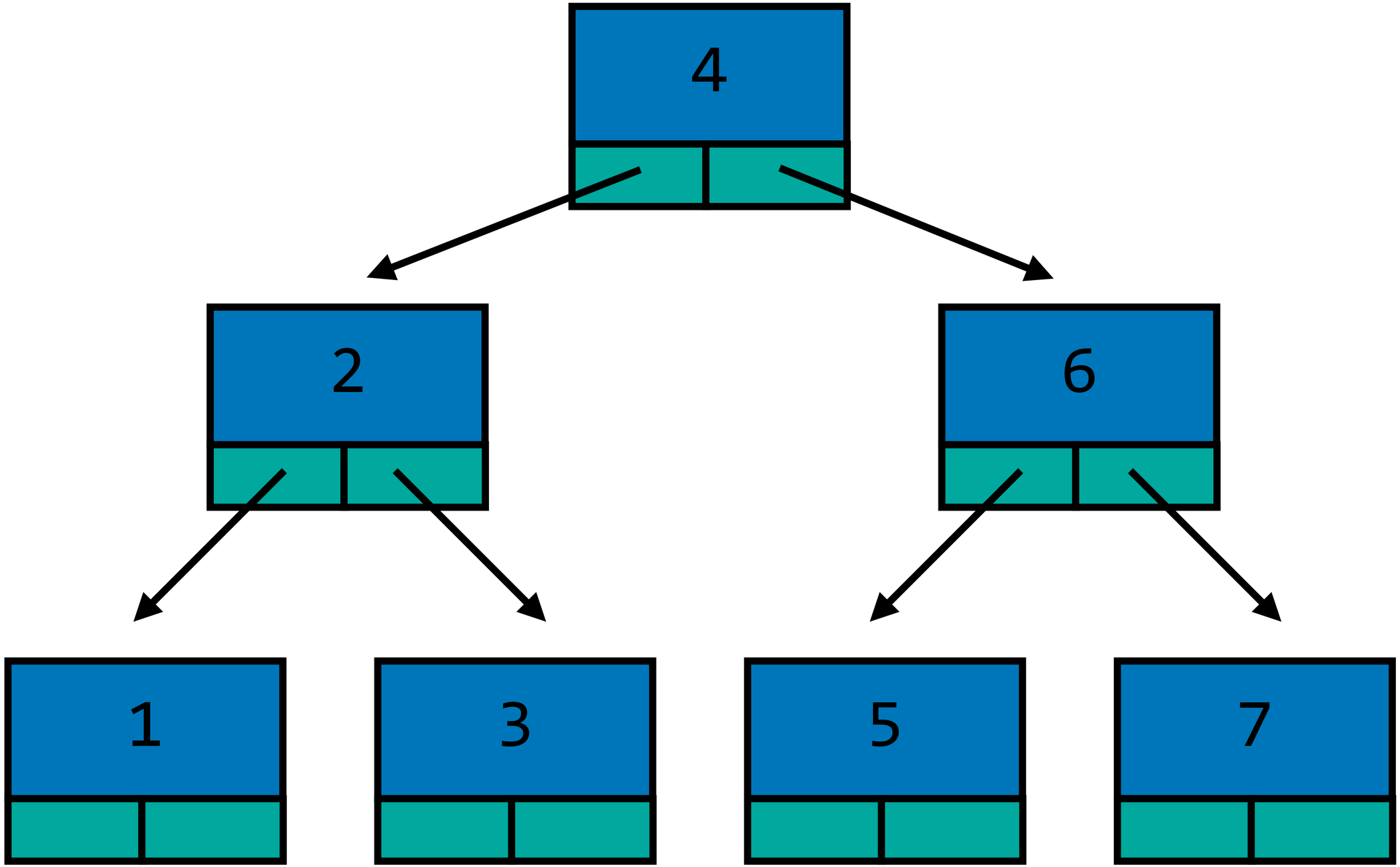
```
typedef struct node
{
    int number;
    struct node *left;
    struct node *right;
}
node;
```

Binary Search Trees

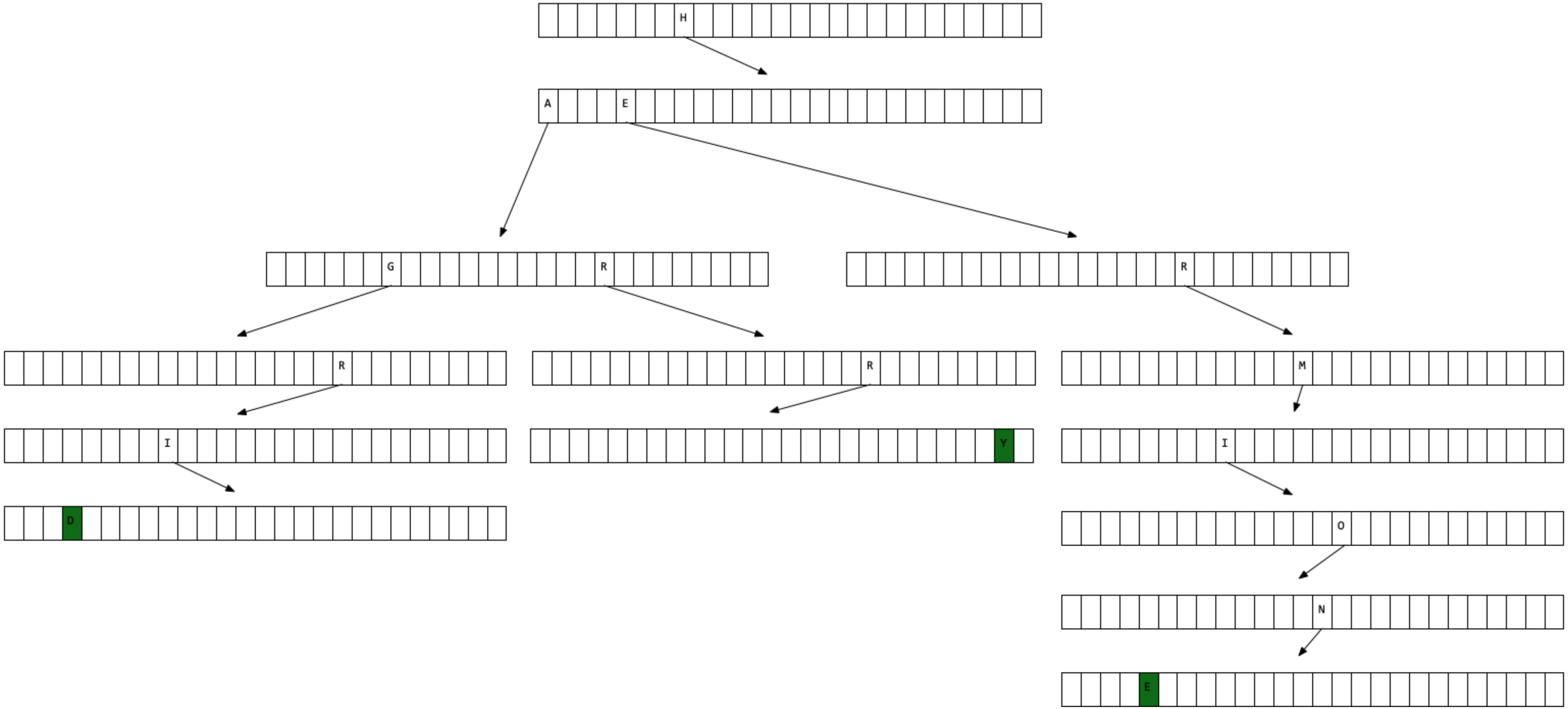

4
2
6
1
3
5
7


4
2
6
1
3
5
7


4
2
6
1
3
5
7


4
2
6
1
3
5
7

Tries


H
A
E
G
R
R
R
R
M
I
Y
I
D
O
N
E

```
typedef struct node
{
    bool word;
    struct node *children[26];
}
node;
```

Part Three

Lab

Problem Set 5

Problem Set 5

- Speller

speller

speller.cs50.net/cs50/problems/2020/spring/challenges/speller

Apps GradescopeTools

Big Board speller

Rank	Name	Time	Load	Check	Size	Unload	Memory	Heap	Stack
1	CS50 Staff Solution Staff	7.445 s	0.825 s	6.165 s	0.000 s	0.455 s	8.0 MB	8.0 MB	2.9 kB

Time is a sum of the times required to spell-check `texts/*.txt` using `dictionaries/large`. **Memory** is a measure of maximal heap and stack utilization when spell-checking `texts/holmes.txt` using `dictionaries/large`.

This is CS50.