This is CS50

data structures

abstract data types

queues

FIFO

enqueue

dequeue

stacks

LIFO

push

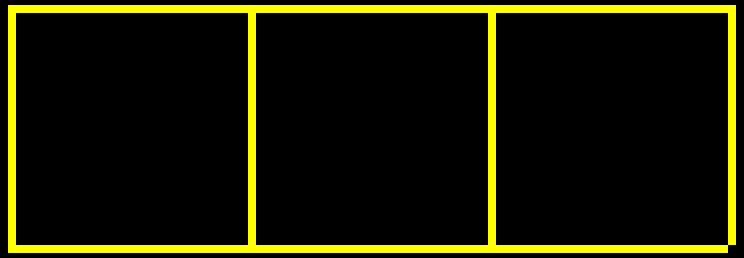
pop

```
typedef struct
{
    person people[CAPACITY];
    int size;
}
stack;
```

```
const int CAPACITY = 50;

typedef struct
{
    person people[CAPACITY];
    int size;
}
stack;
```

This is CS50

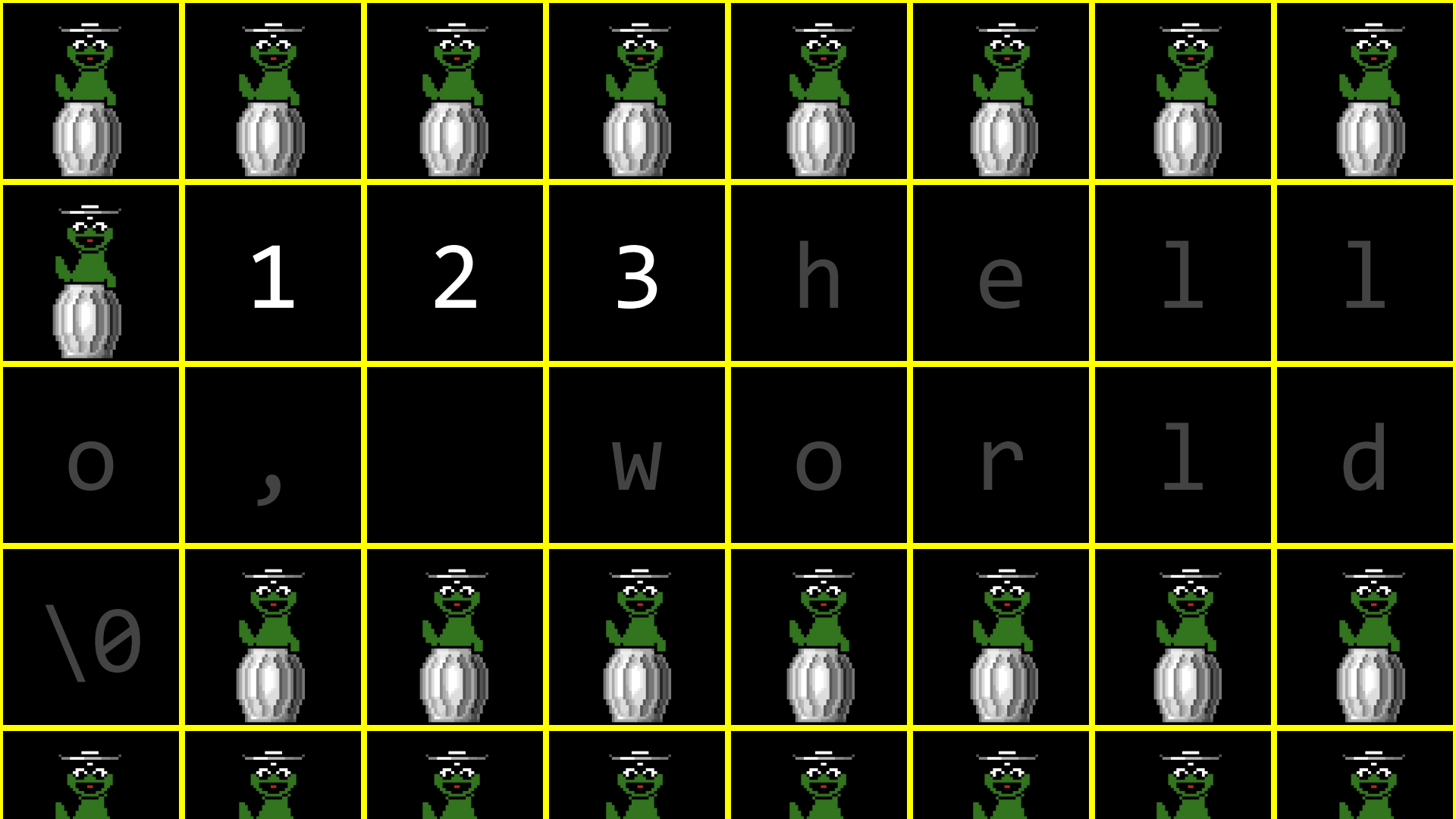
arrays



1	2	3

1	2	3	

1 2 3


1 2 3 h e l l
o , w o r l d
\0

1	2	3

1	2	3

1			

1	2	3

1	2		

1	2	3

1	2	3	

1	2	3

1	2	3	4

1	2	3	4

data structures

struct

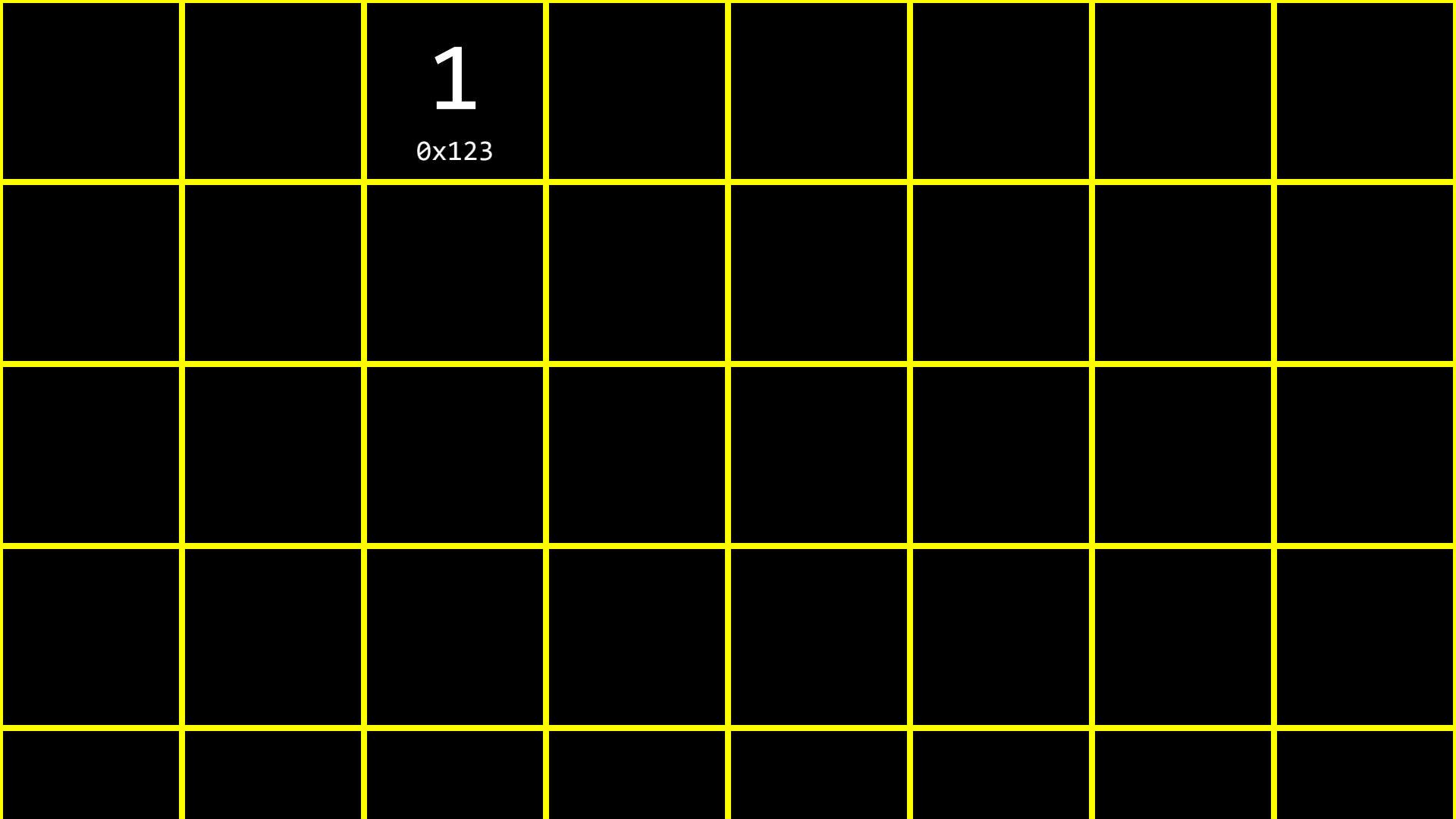
.

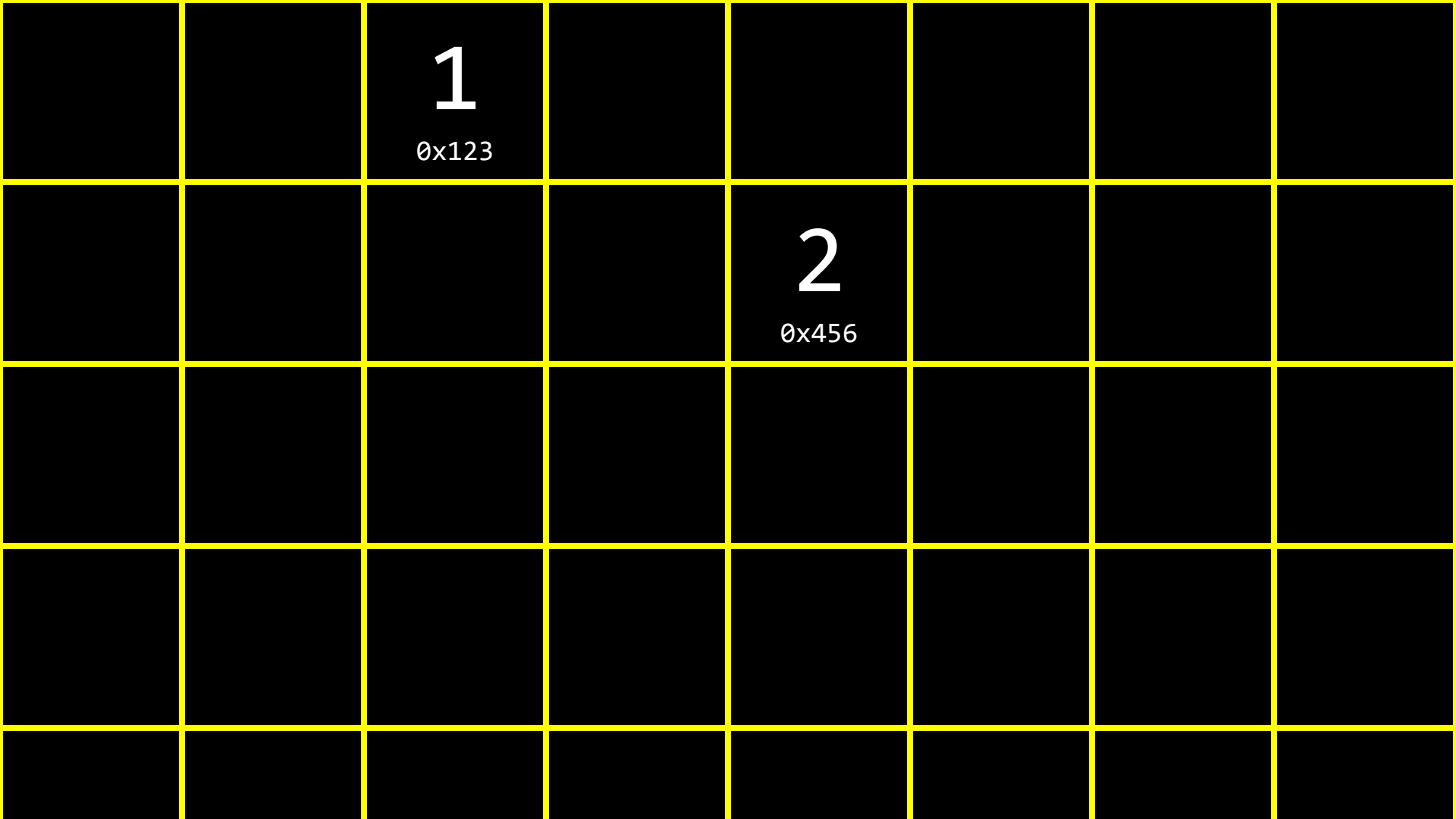
*

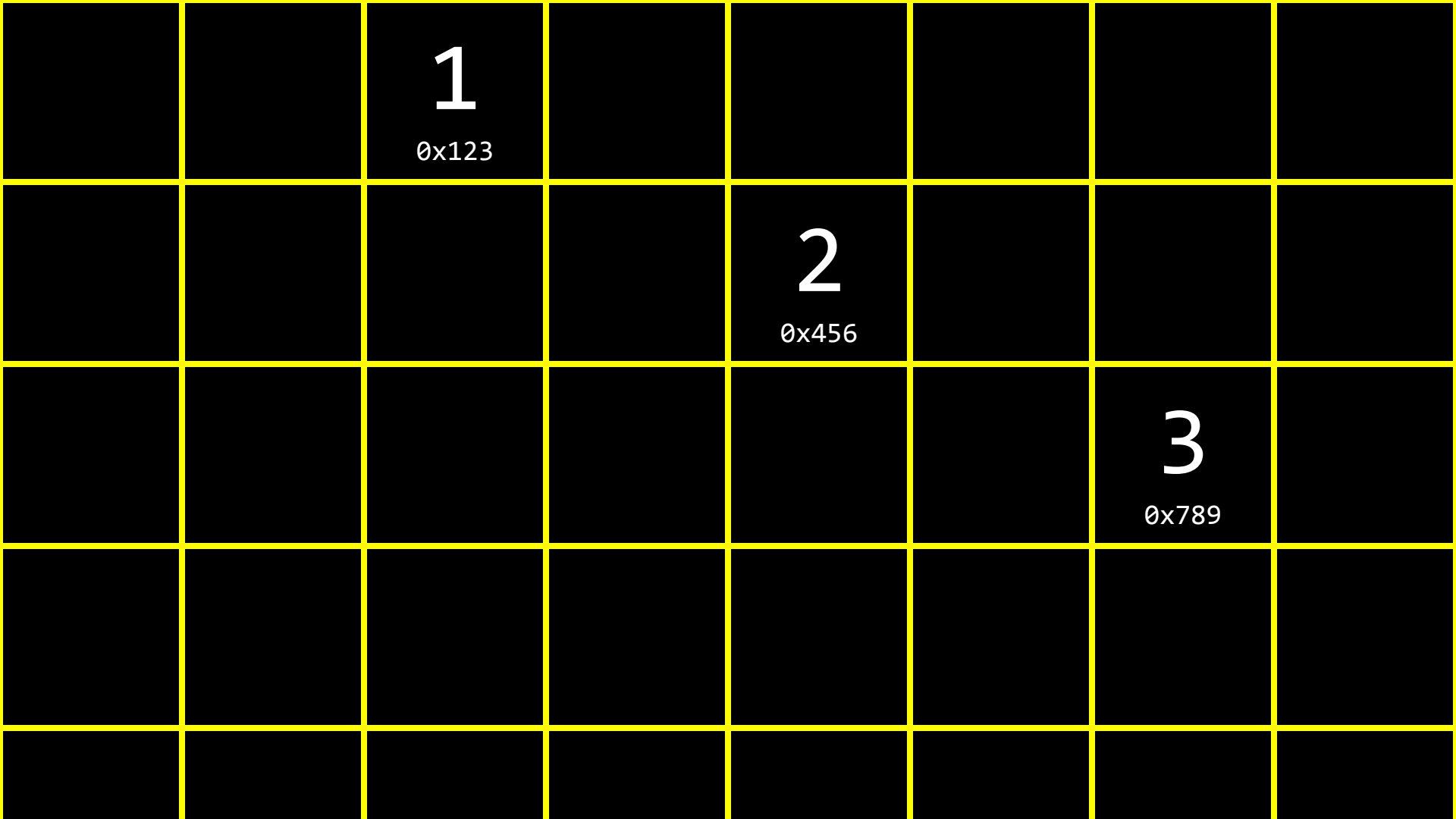
struct

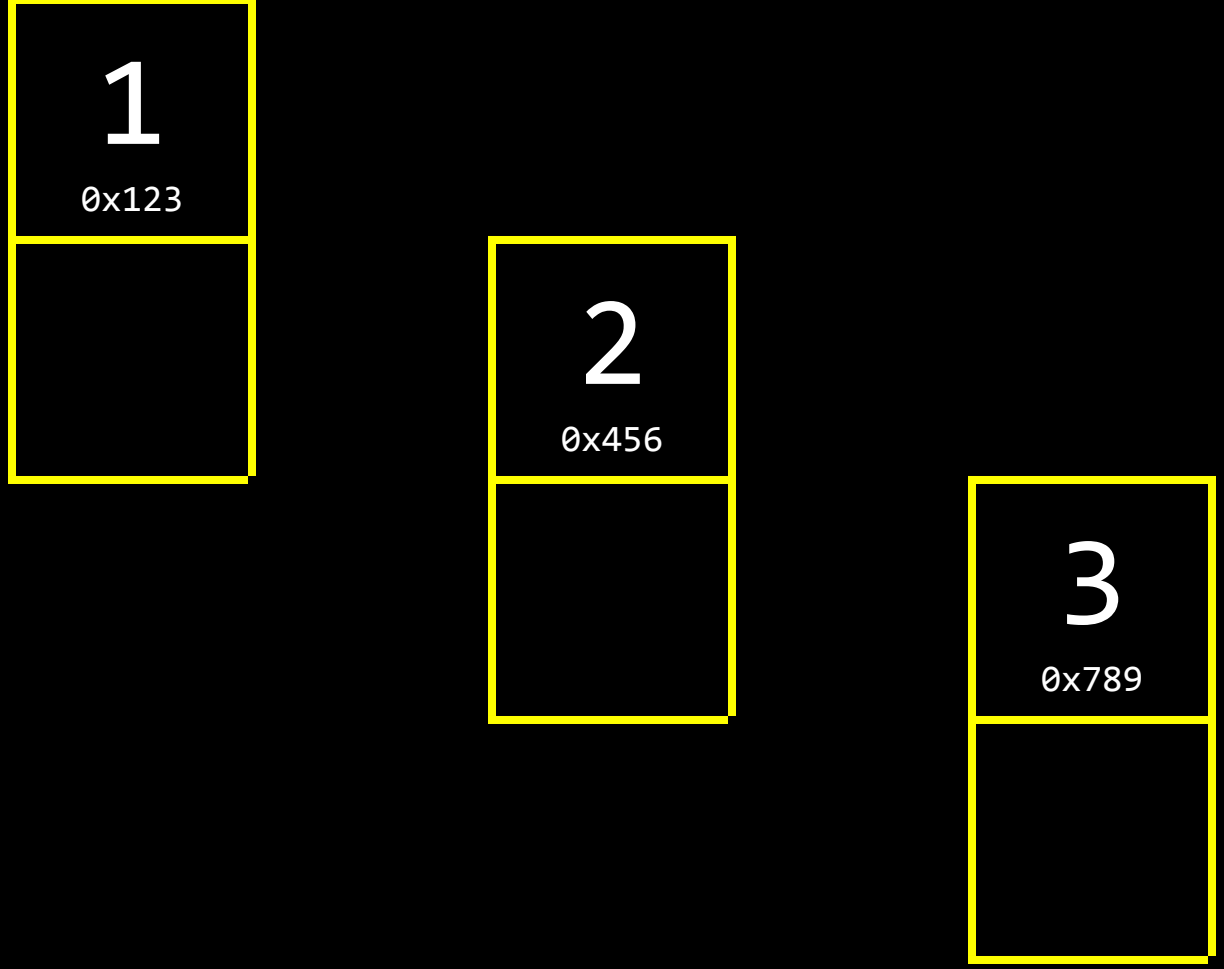
->

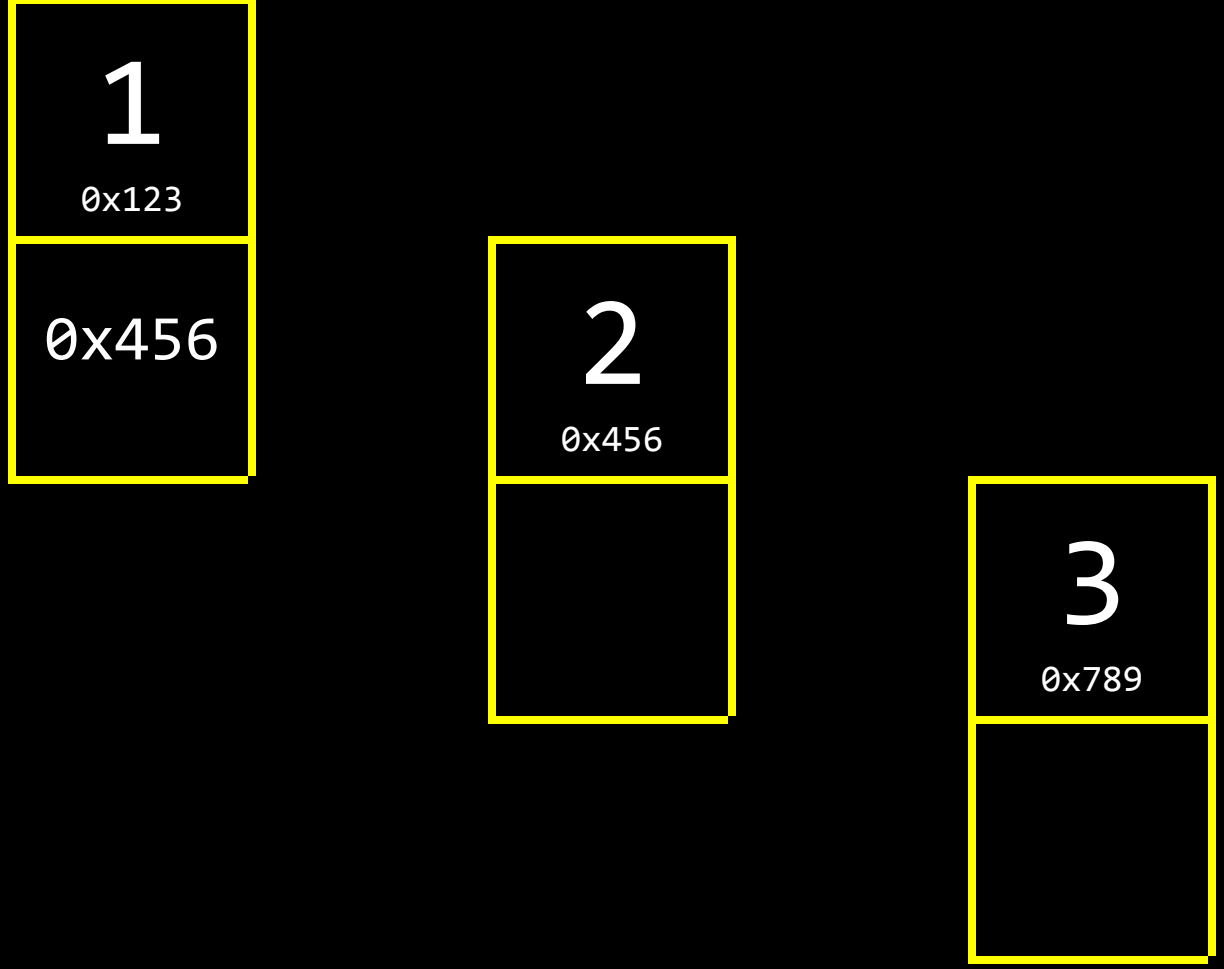
linked lists

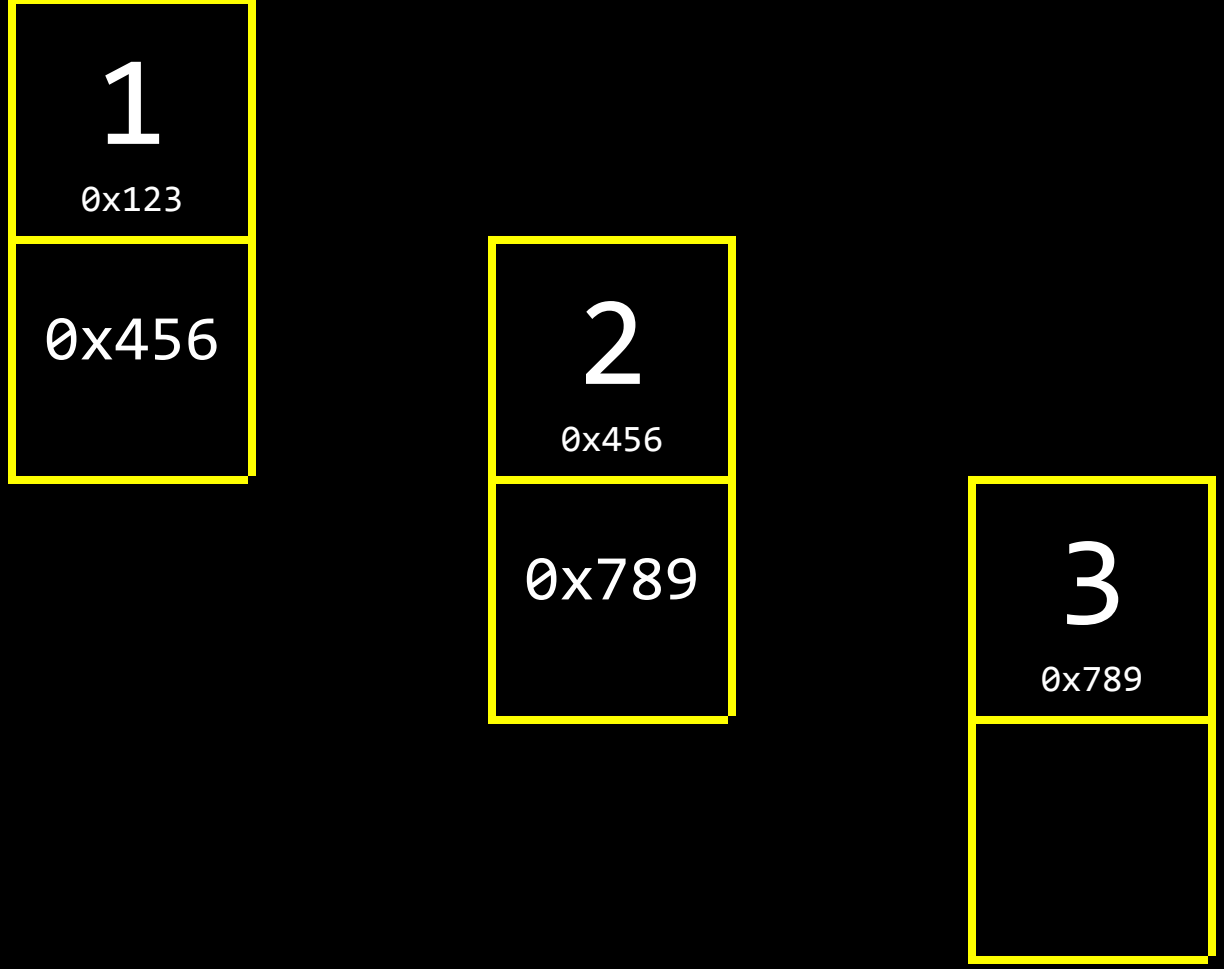

1
0x123

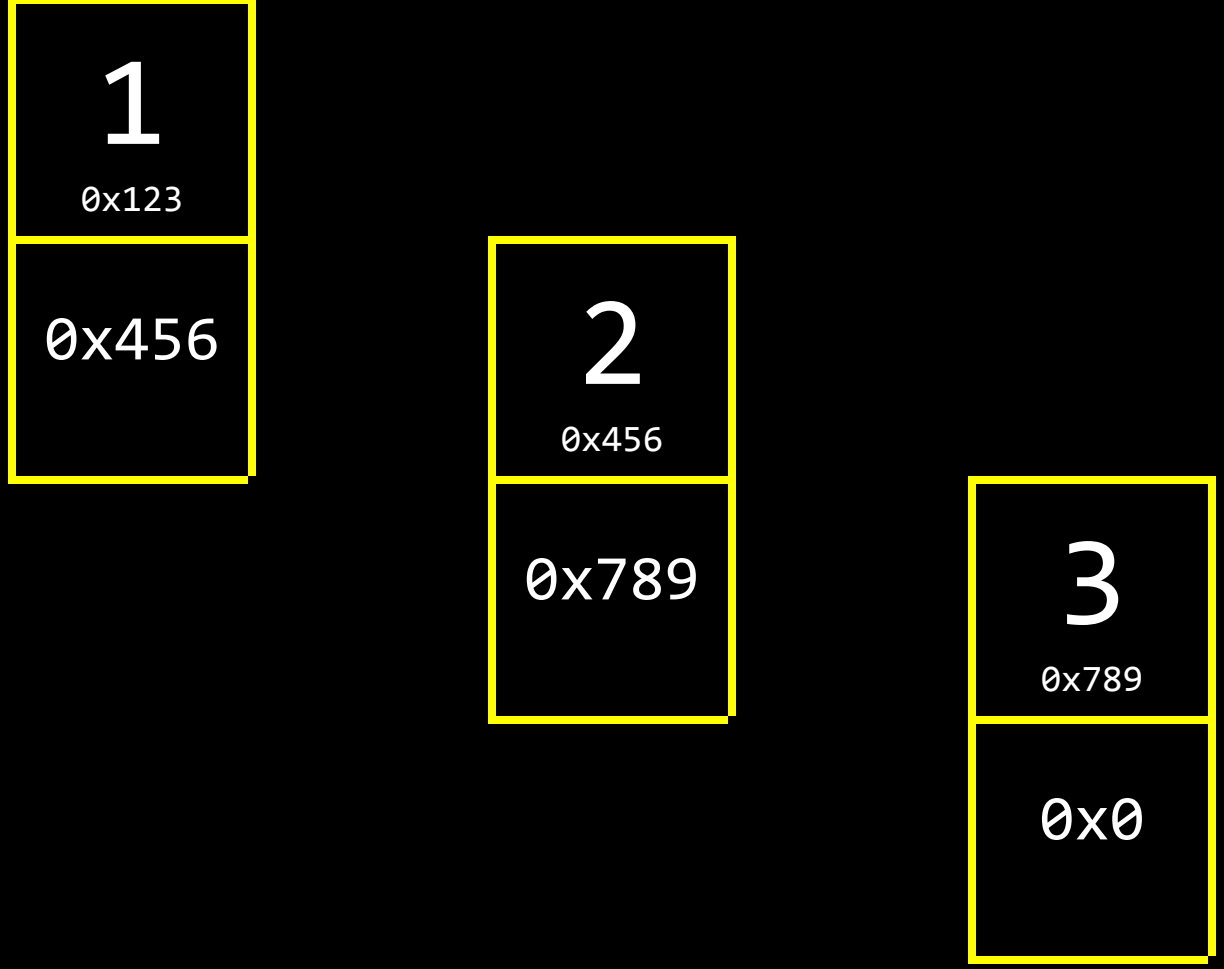

1
0x123
2
0x456

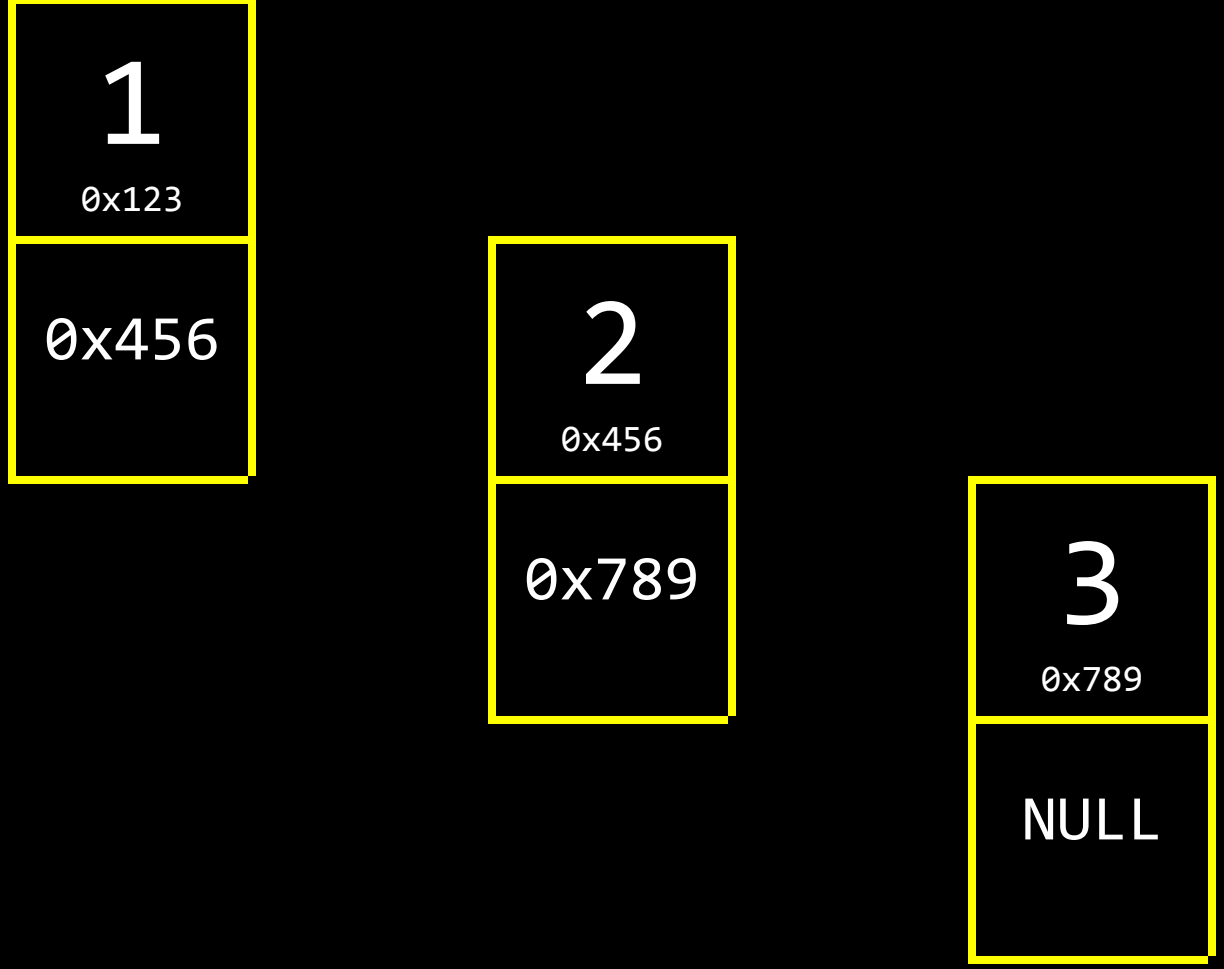

1
0x123
2
0x456
3
0x789

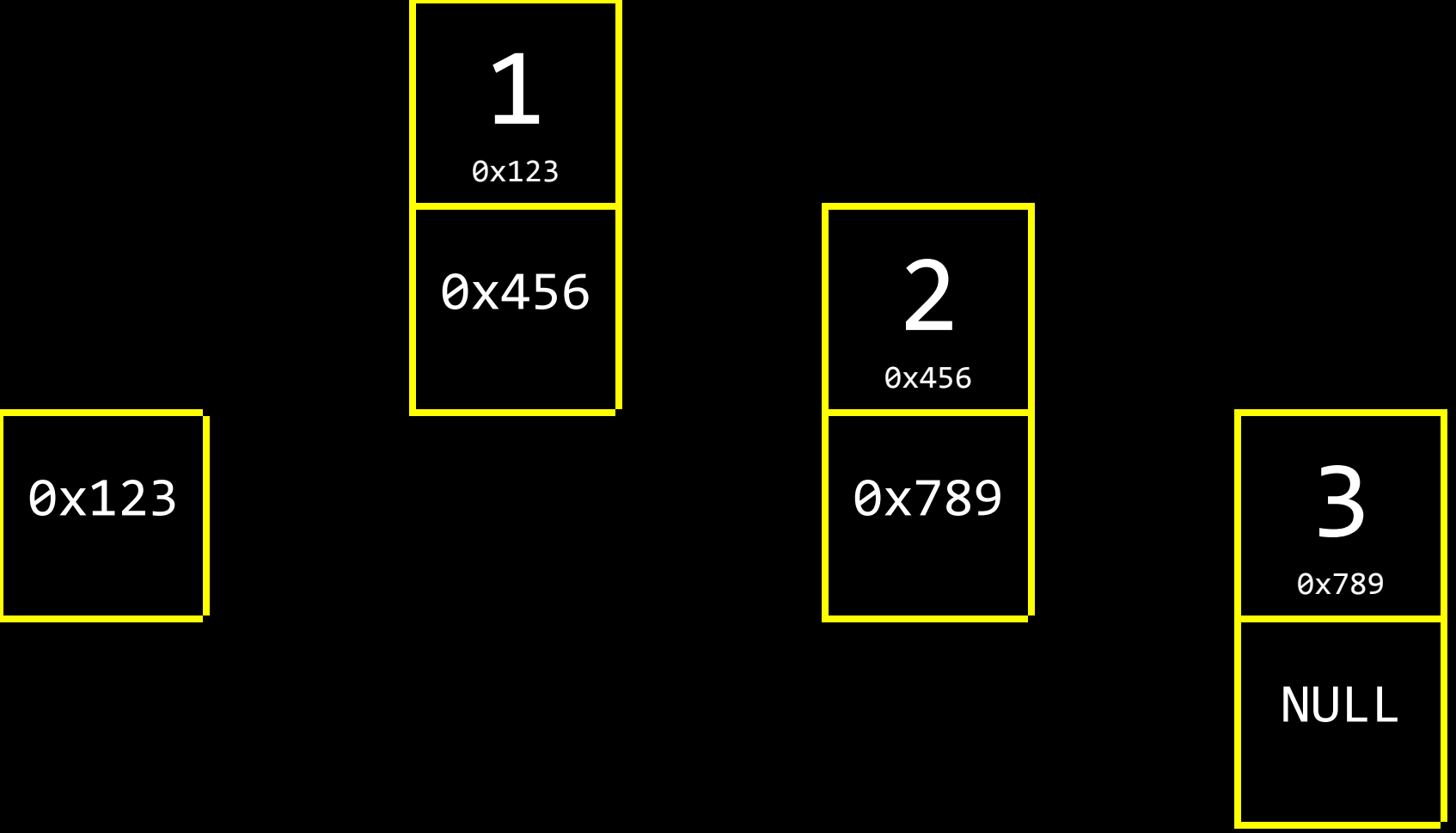

1
0x123
2
0x456
3
0x789

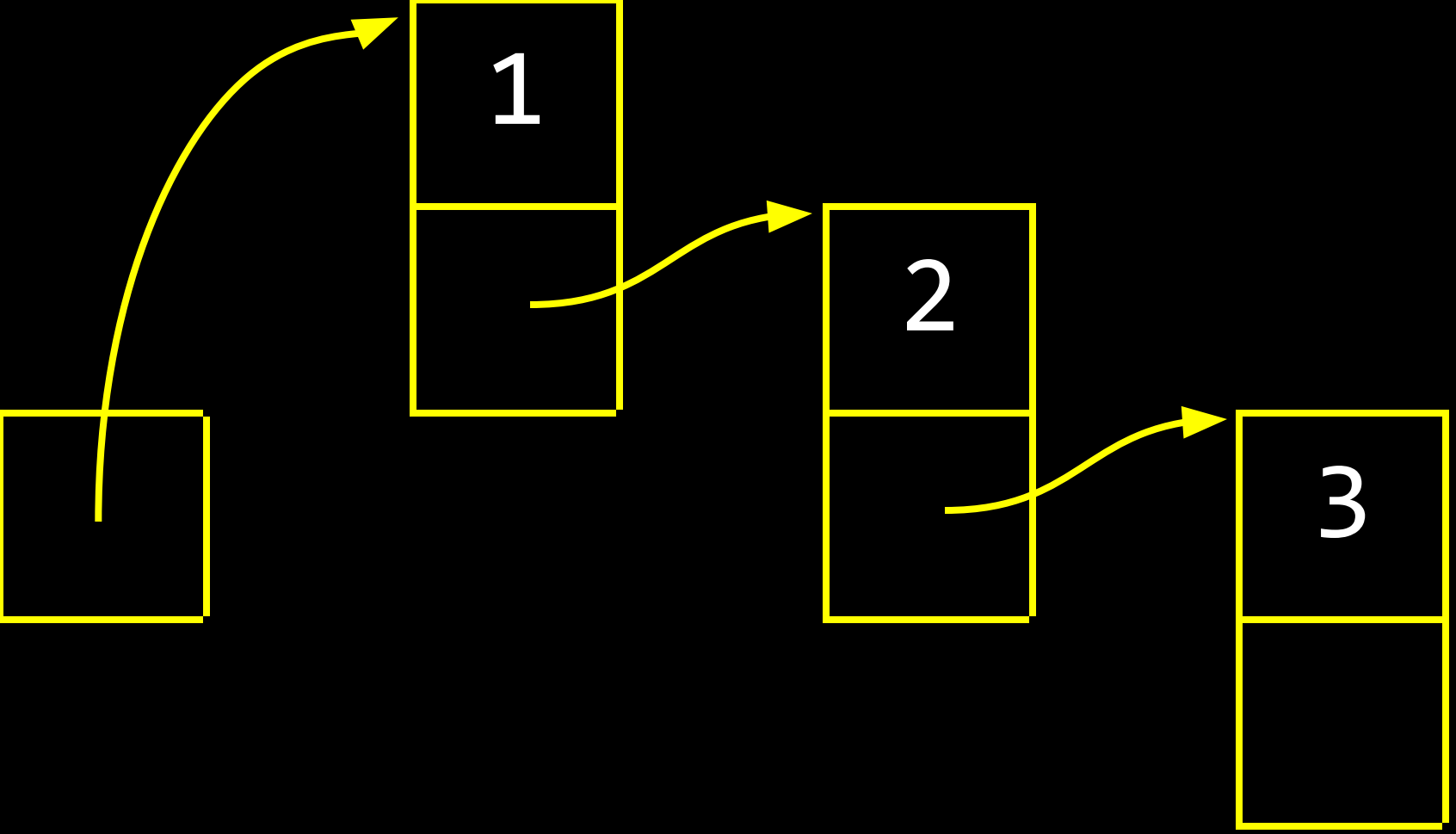

1
0x123
0x456
2
0x456
3
0x789


1
0x123
0x456
2
0x456
0x789
3
0x789


1
0x123
0x456
2
0x456
0x789
3
0x789
0x0


1
0x123
0x456
2
0x456
0x789
3
0x789
NULL


1
0x123
0x456
2
0x456
0x123
0x789
3
0x789
NULL


1
2
3

```
typedef struct
{
    string name;
    string number;
}
person;
```

```
typedef struct
{
    char *name;
    char *number;
}
person;
```

```
typedef struct
{


}
person;
```

```
typedef struct
{


}
node;
```

```
typedef struct
{
    int number;

}
node;
```

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

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

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

This is CS50

```
node *list;
```

```
node *list;
```

list

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

list

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

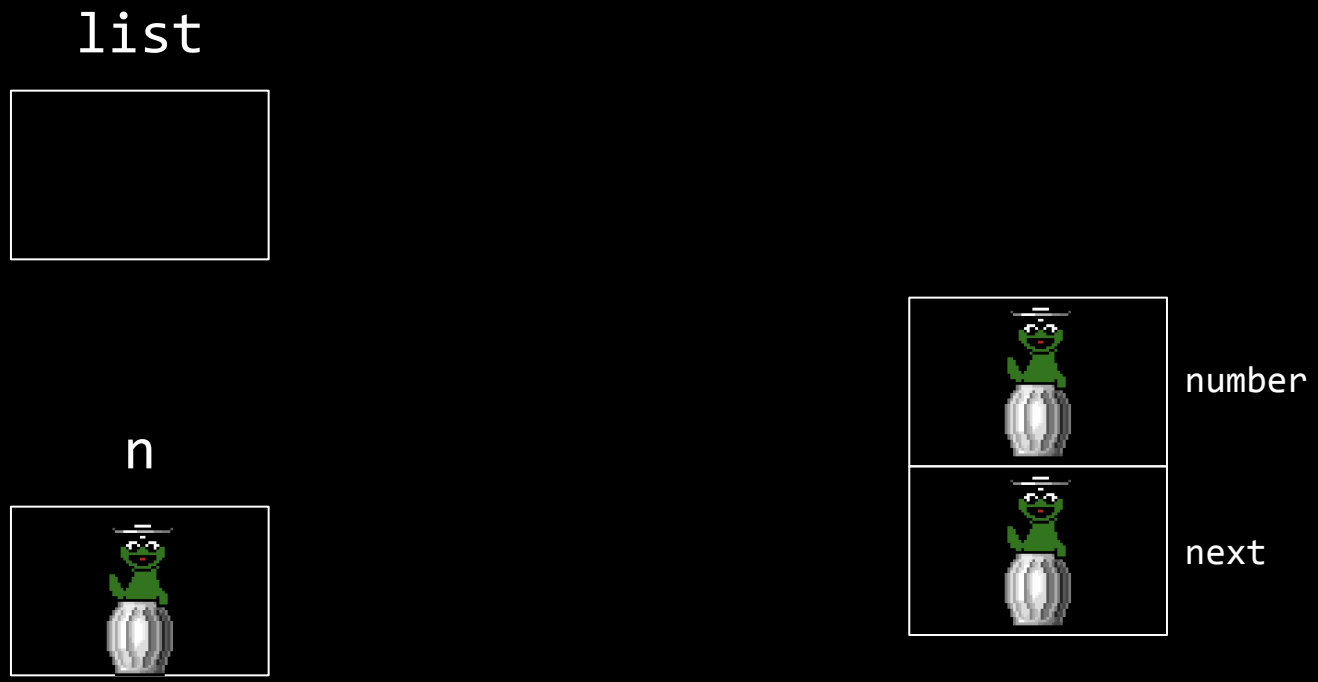
```
node *n = malloc(sizeof(node));
```

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

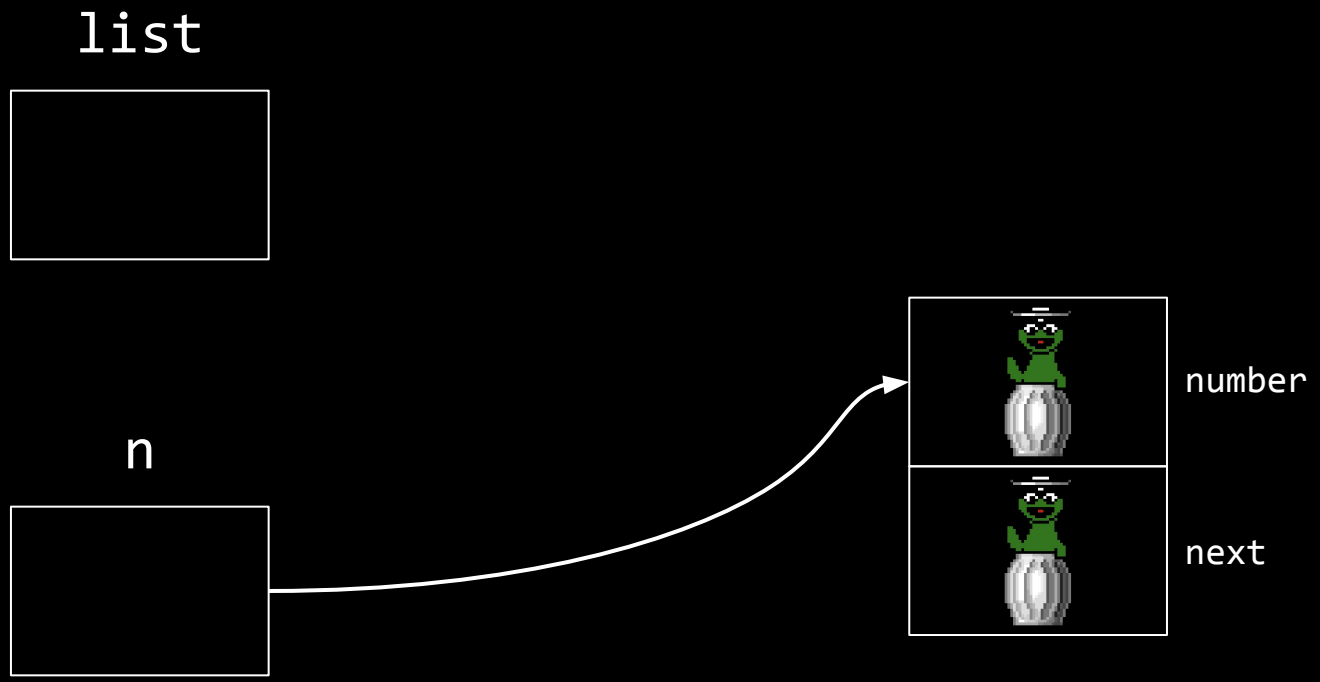
list

n

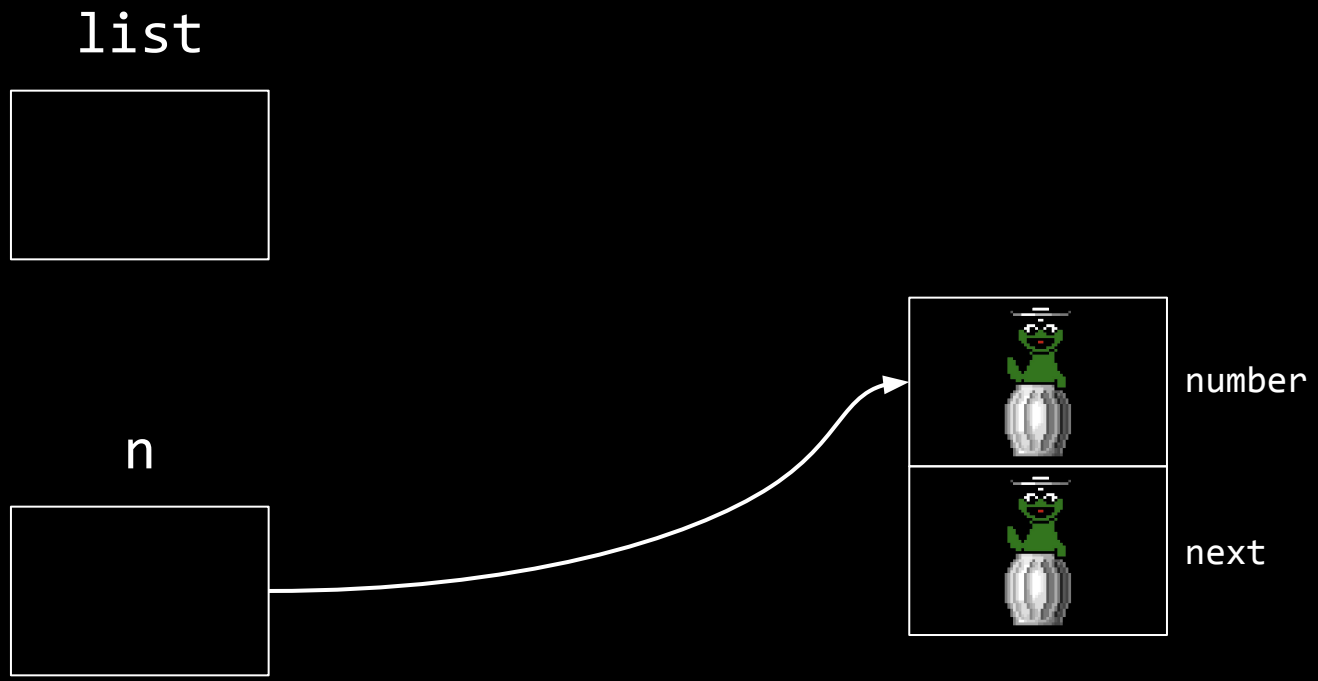
```
node *n = malloc(sizeof(node));
```



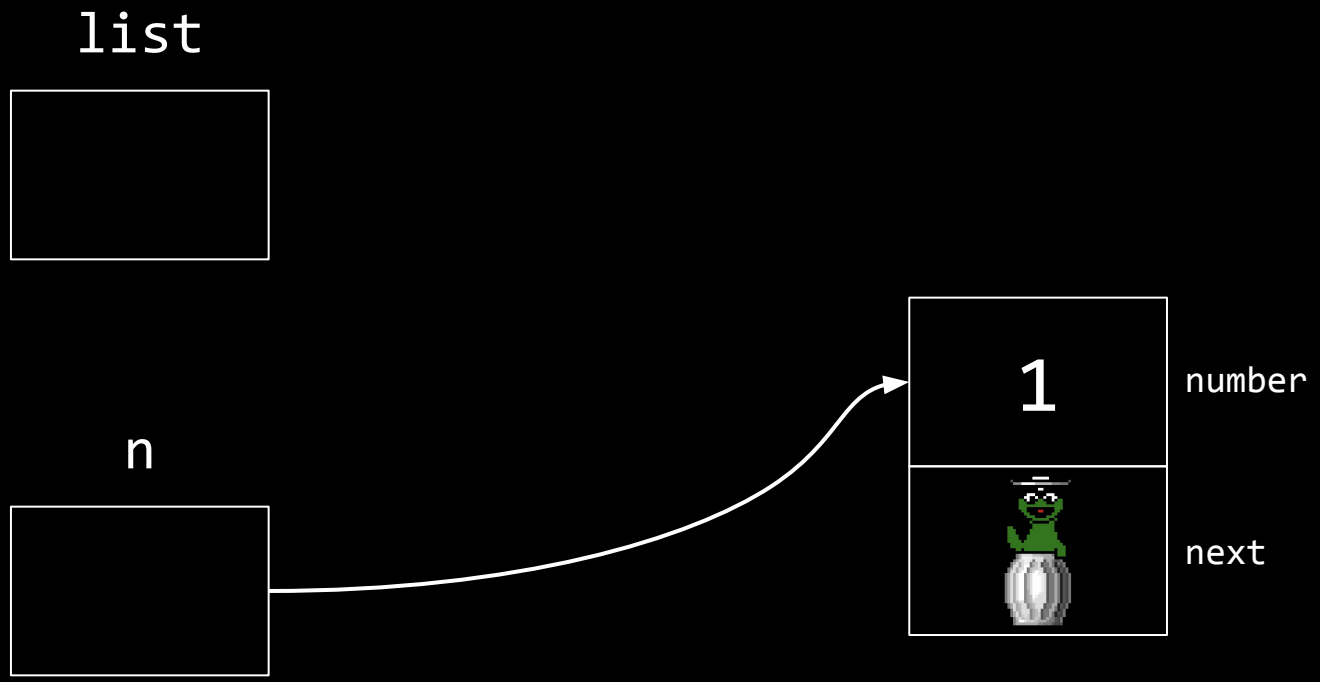
```
node *n = malloc(sizeof(node));
```



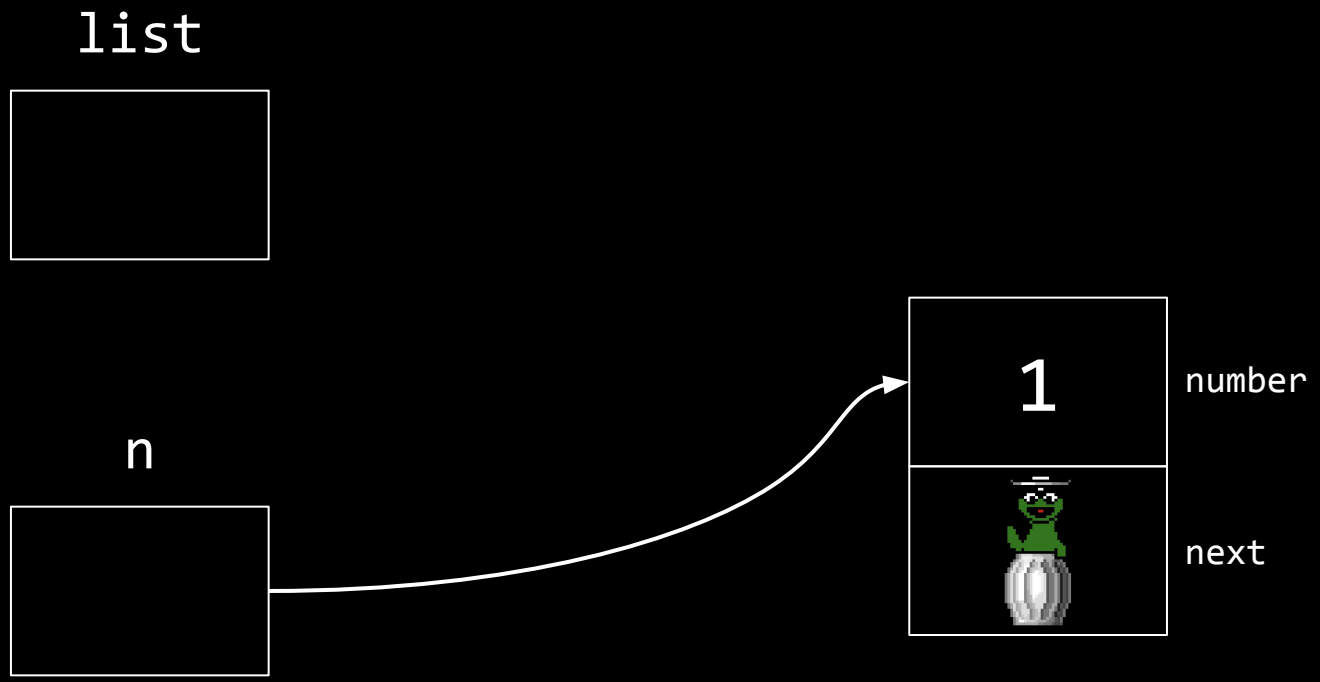
```
(*n).number = 1;
```



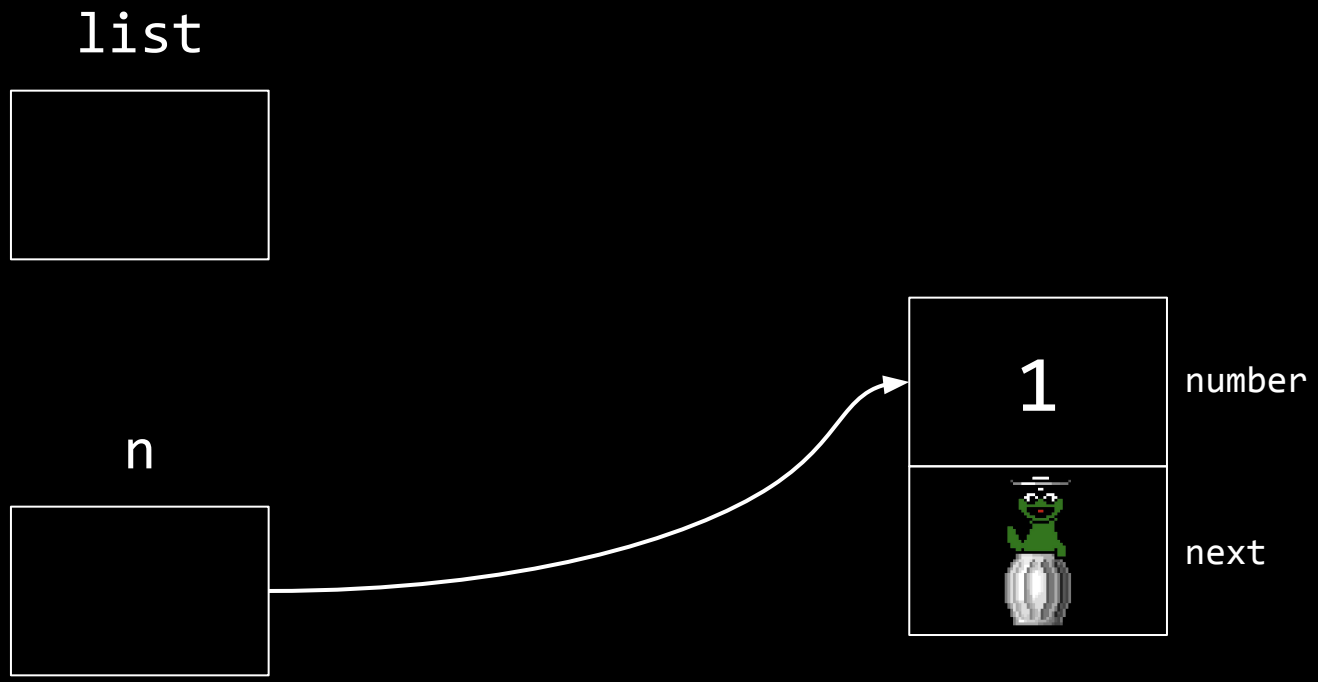
```
(*n).number = 1;
```



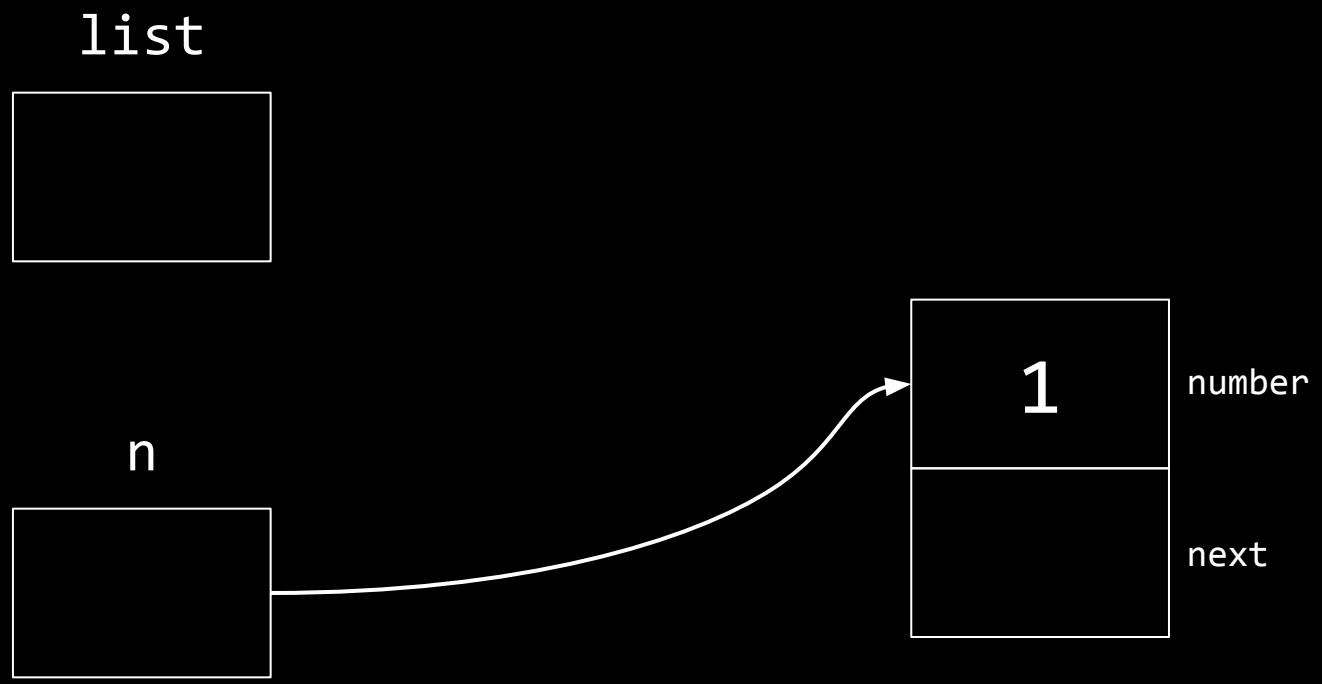
```
n->number = 1;
```



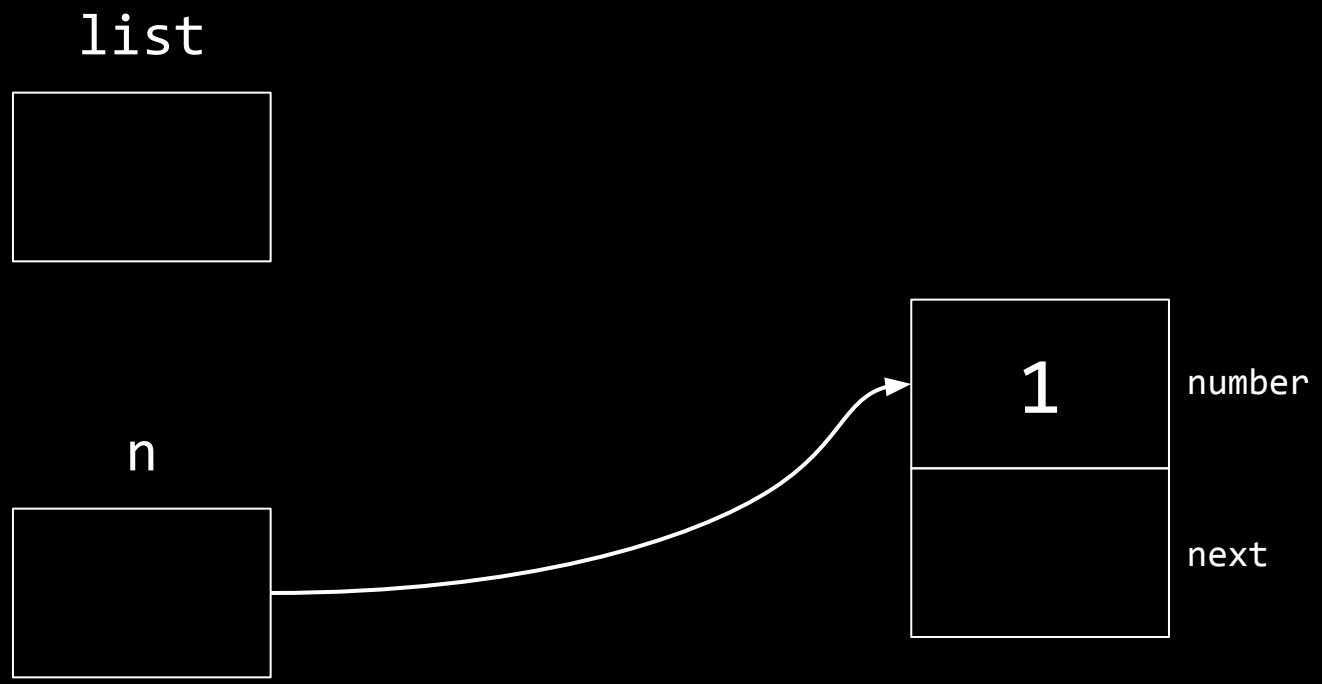
```
n->next = NULL;
```



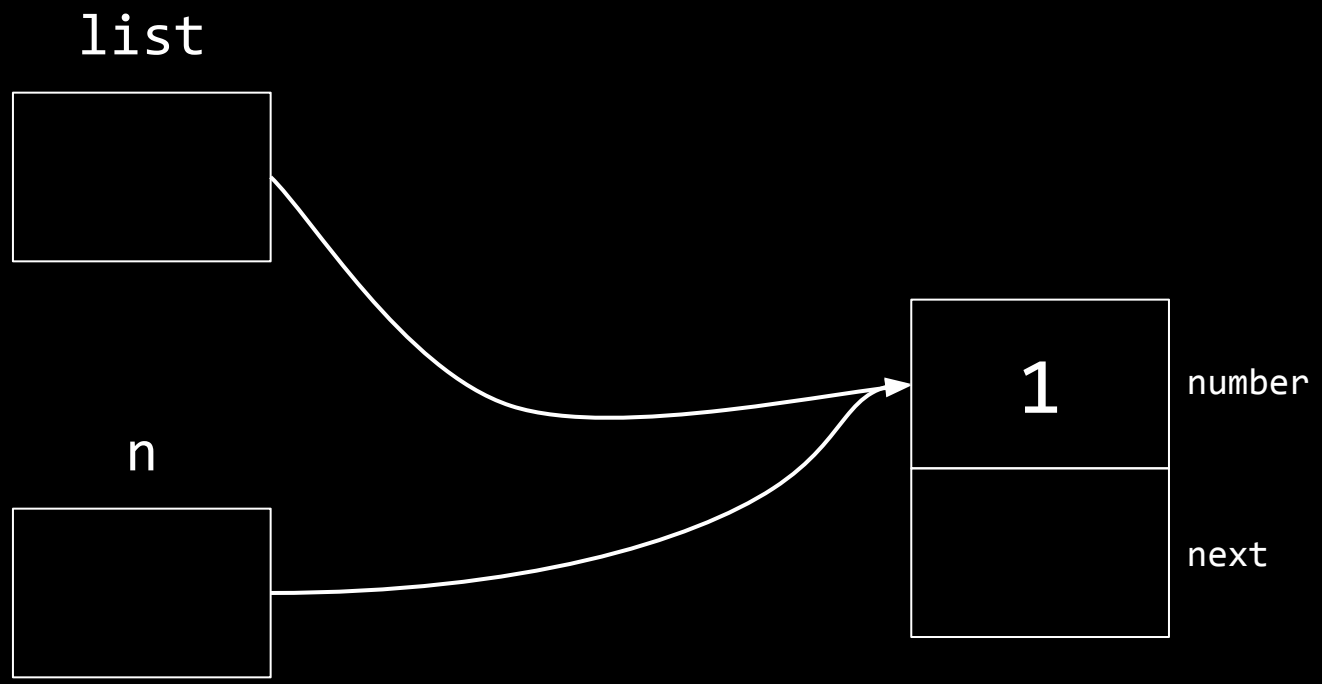
```
n->next = NULL;
```

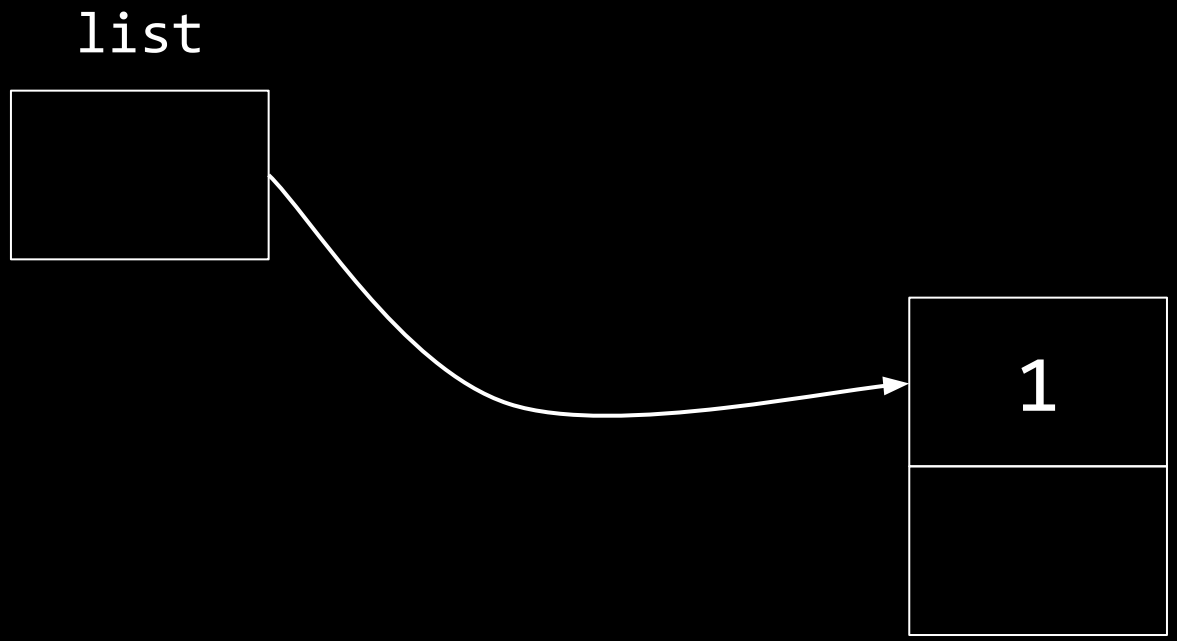


```
list = n;
```

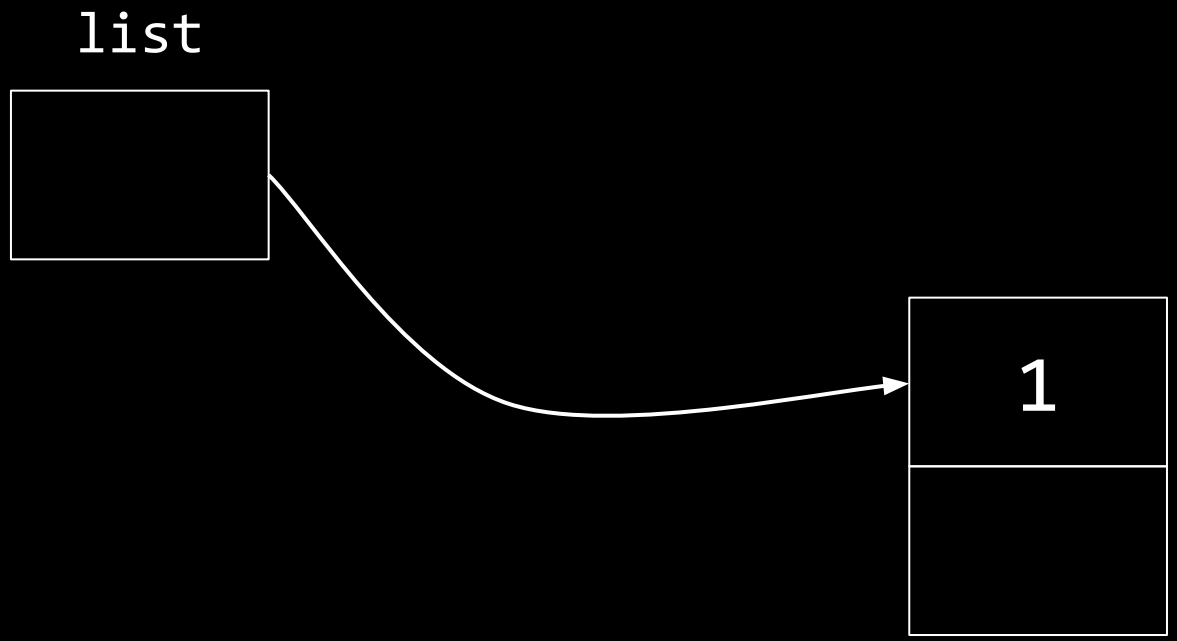


```
list = n;
```

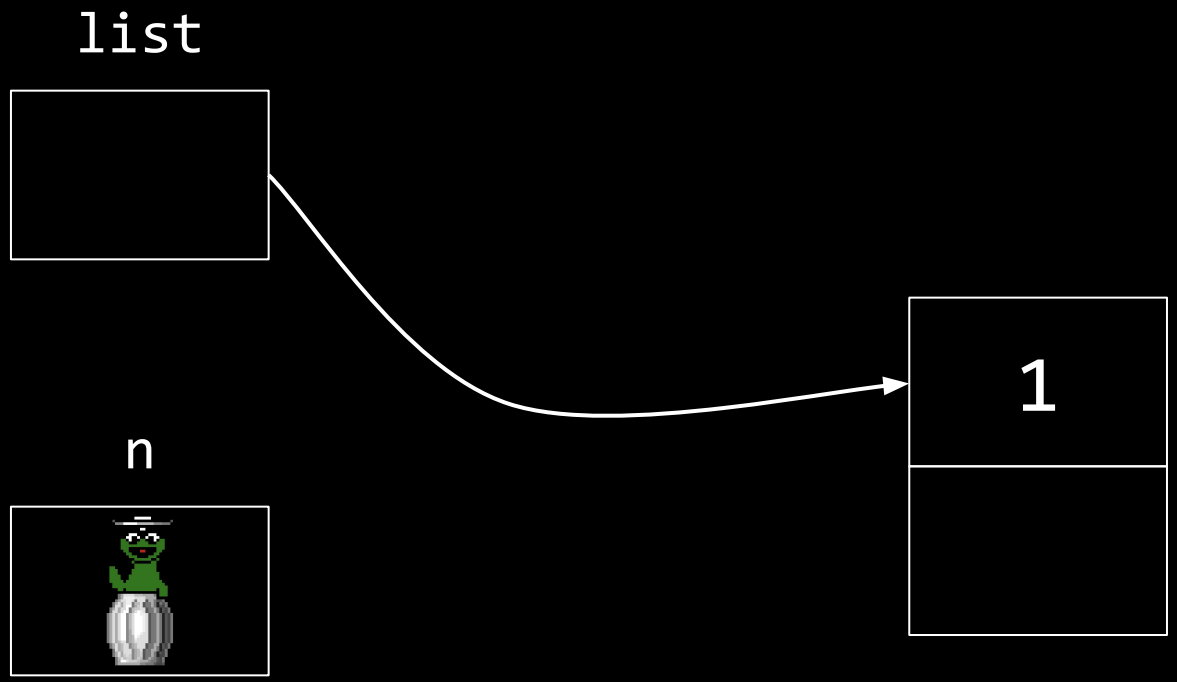



list
1

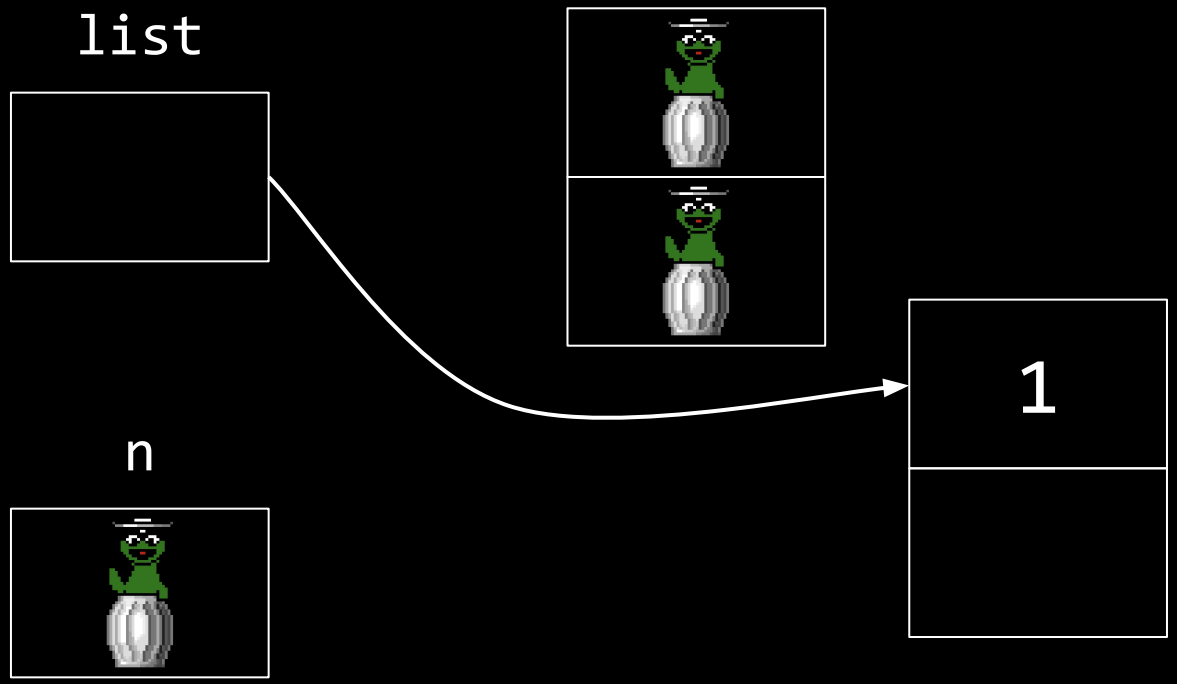
```
node *n = malloc(sizeof(node));
```



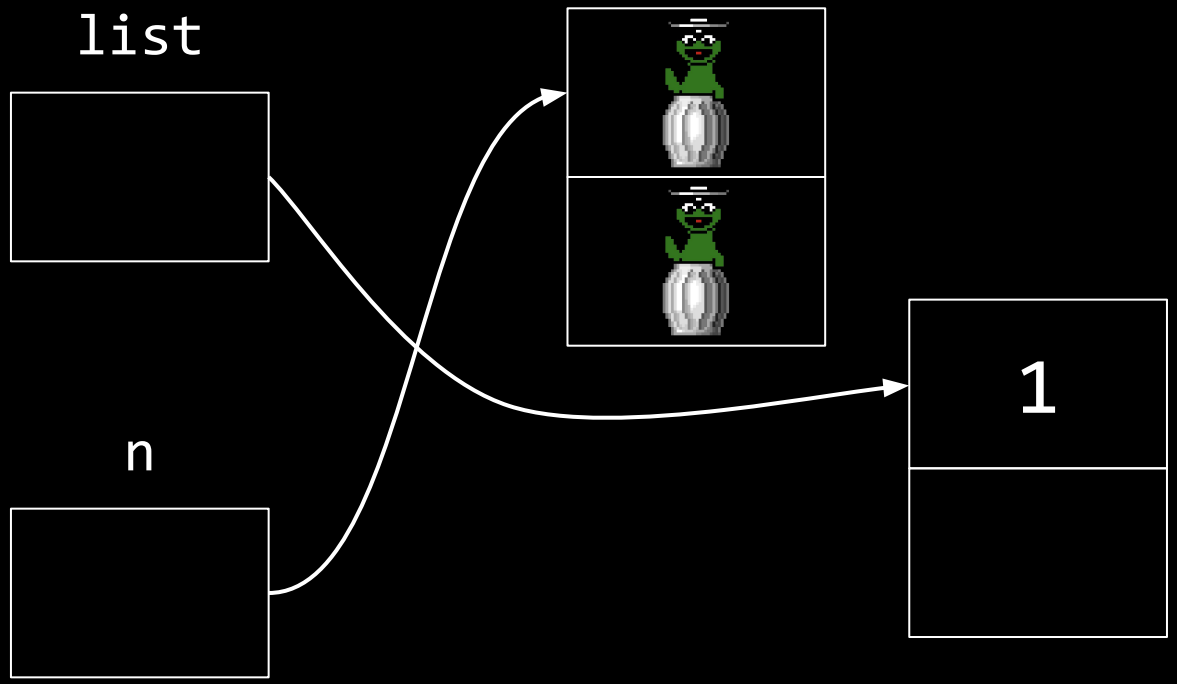
```
node *n = malloc(sizeof(node));
```



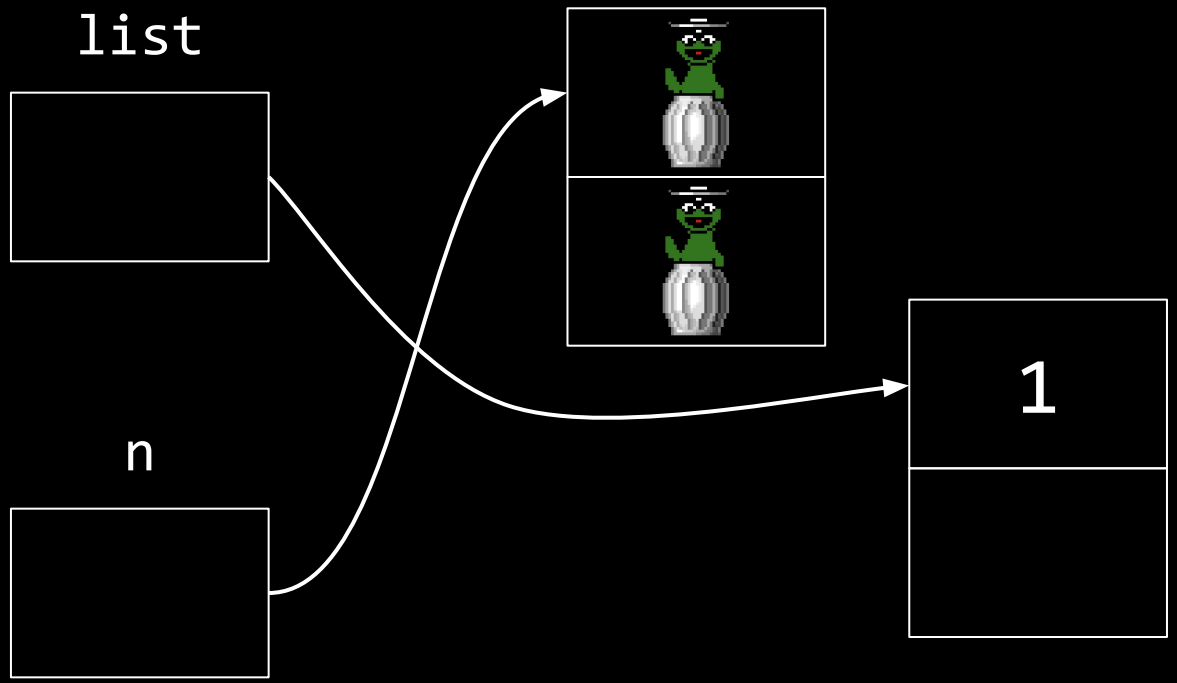
```
node *n = malloc(sizeof(node));
```



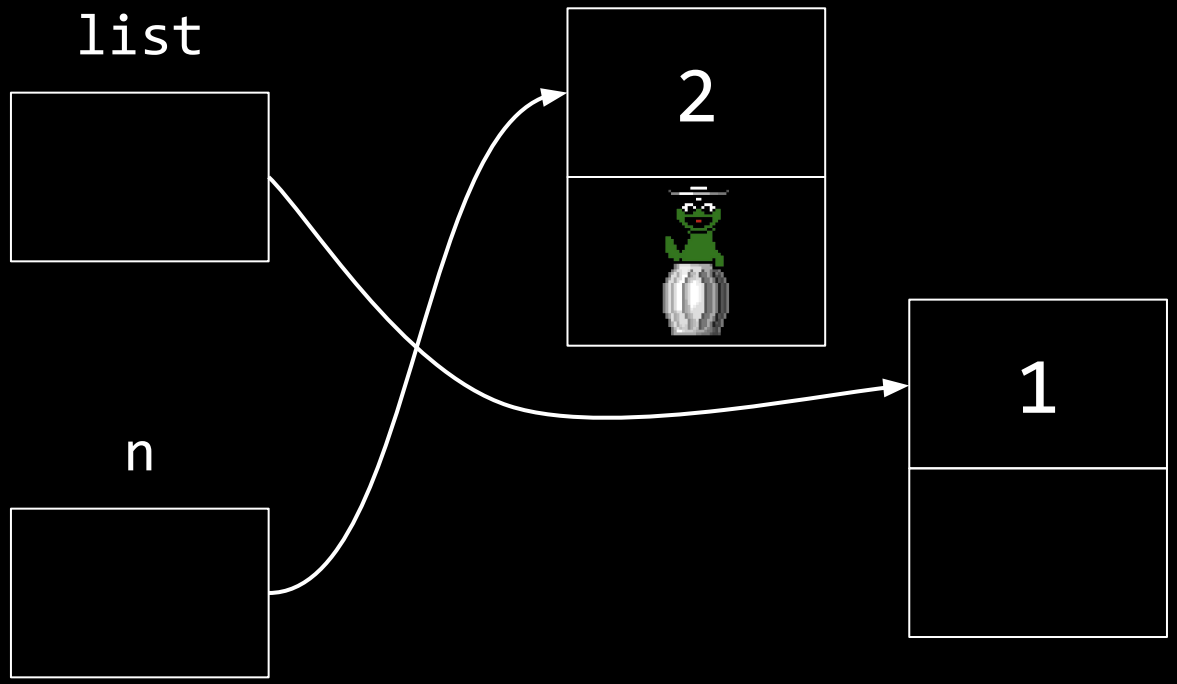
```
node *n = malloc(sizeof(node));
```



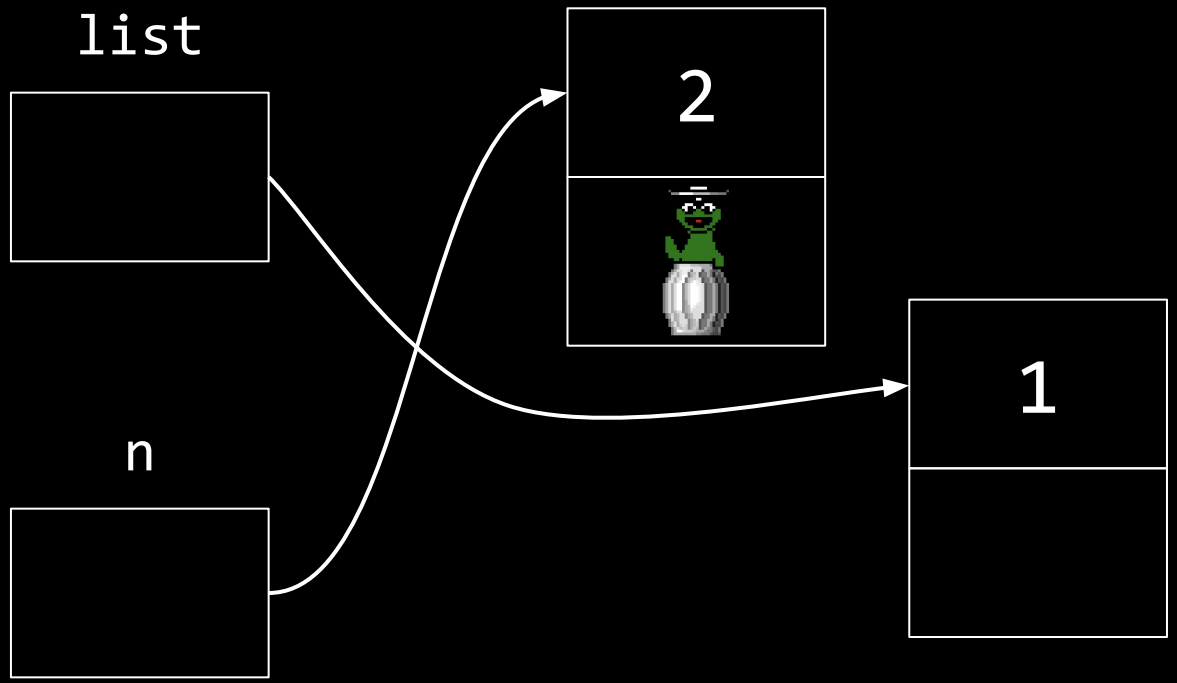
```
n->number = 2;
```



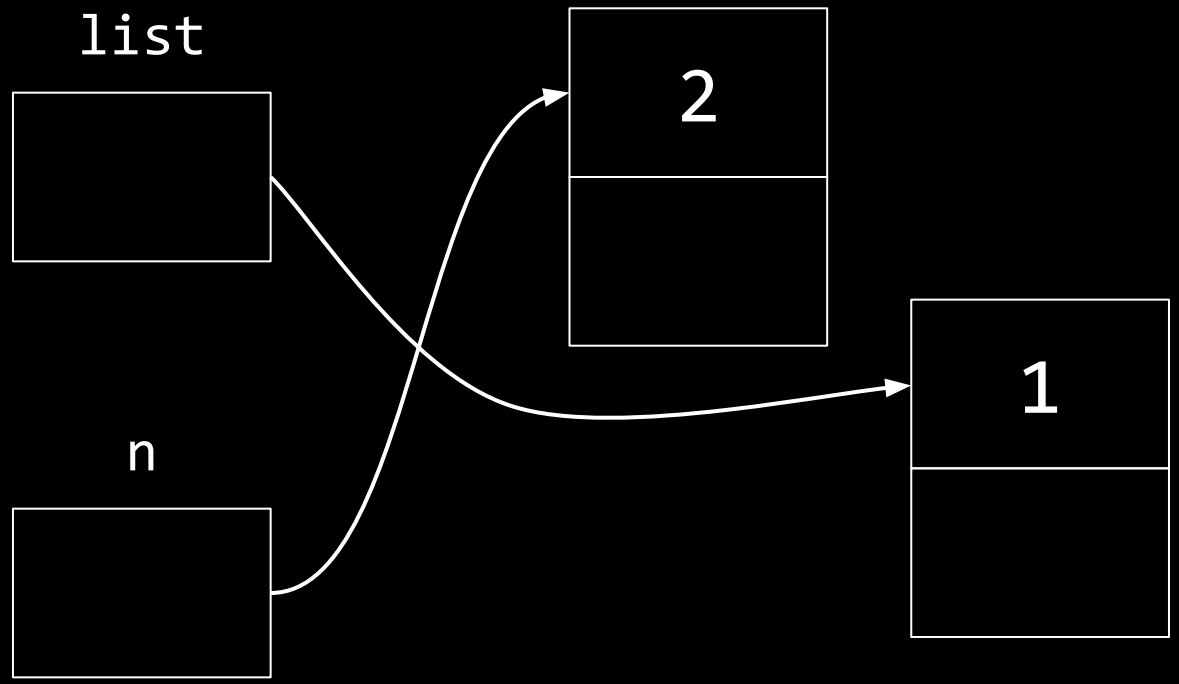
```
n->number = 2;
```

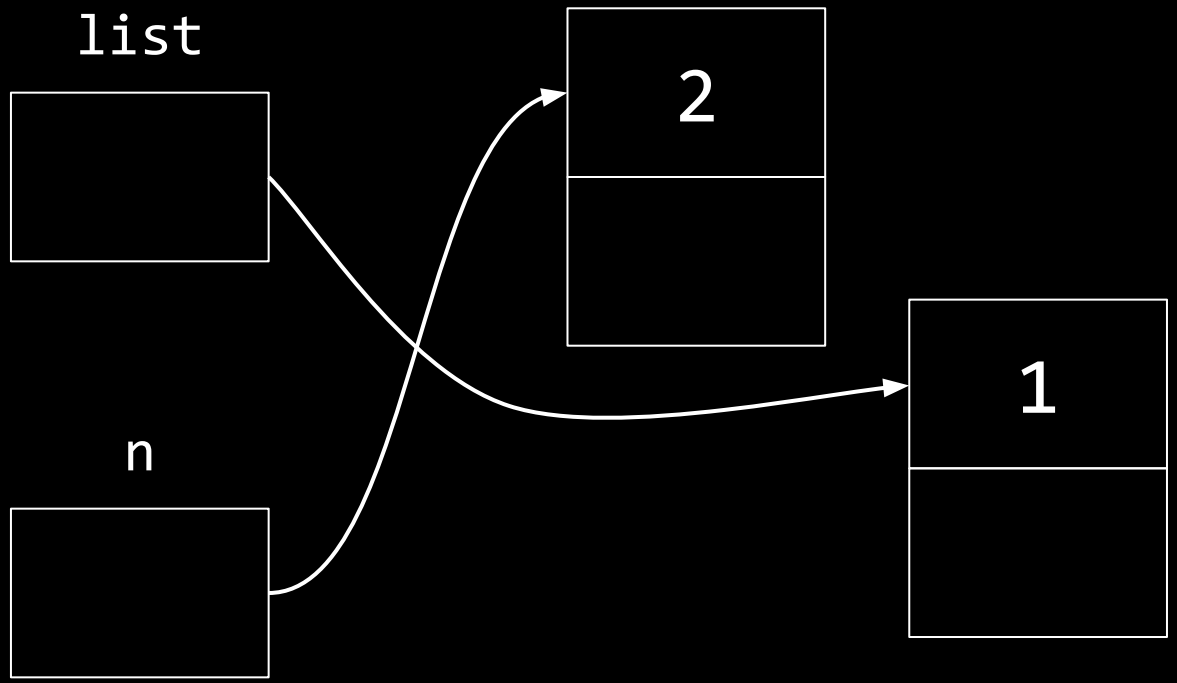


```
n->next = NULL;
```

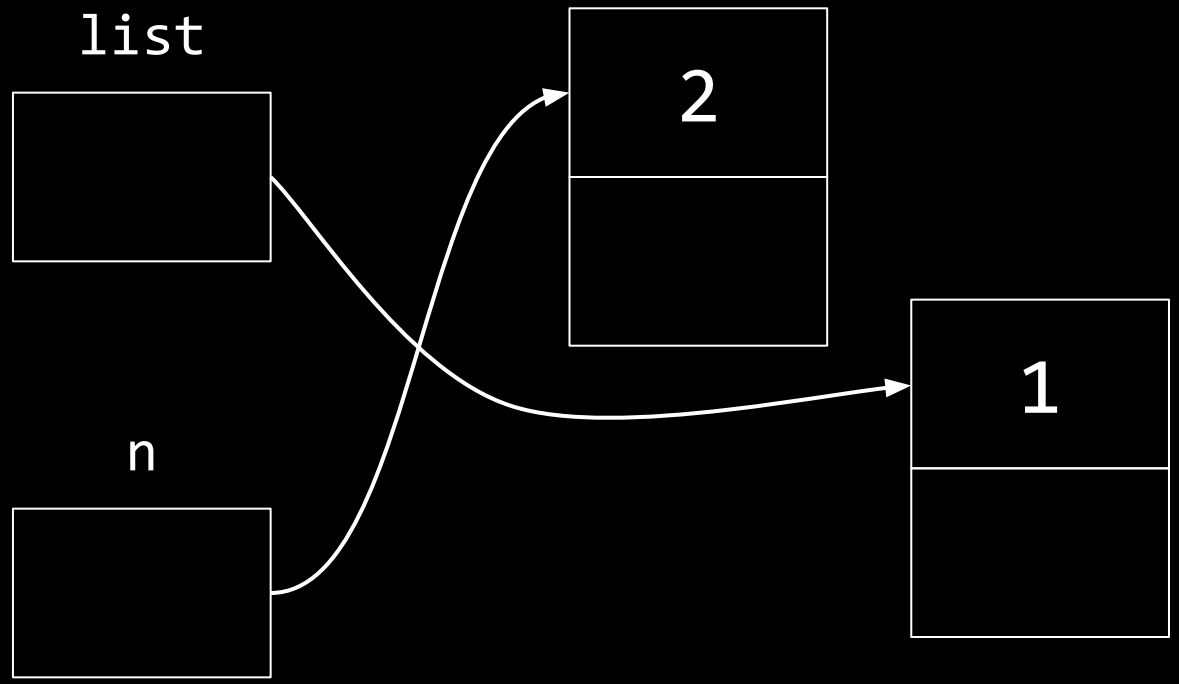


```
n->next = NULL;
```

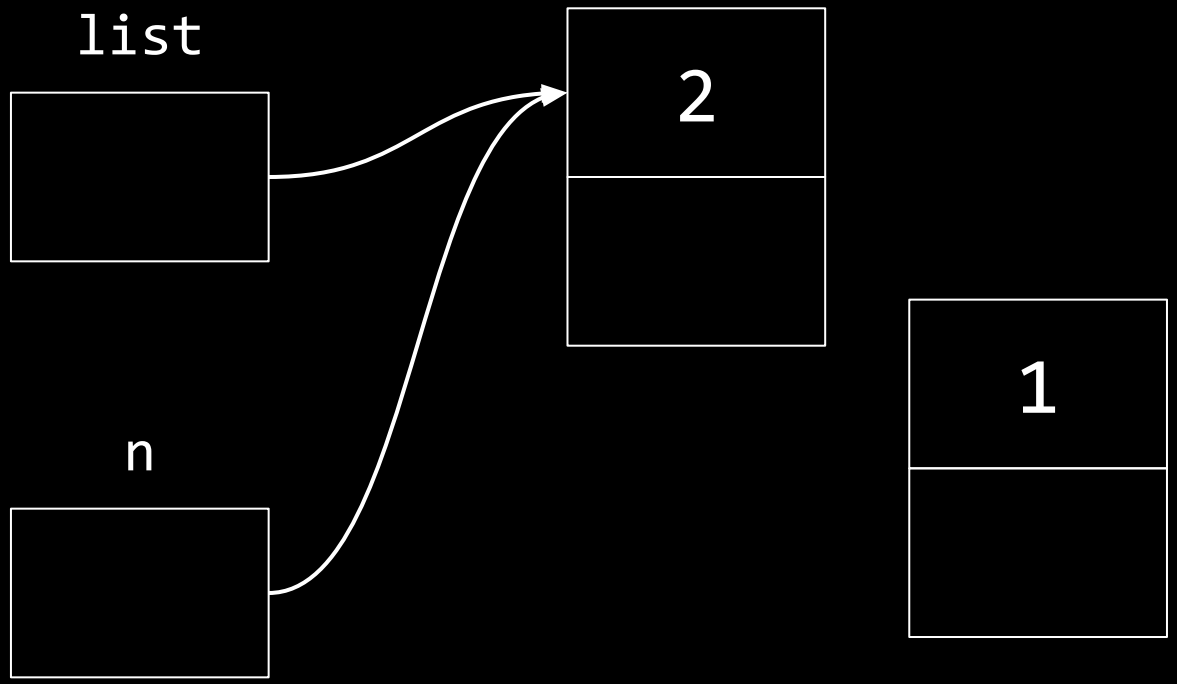



list
2
1
n

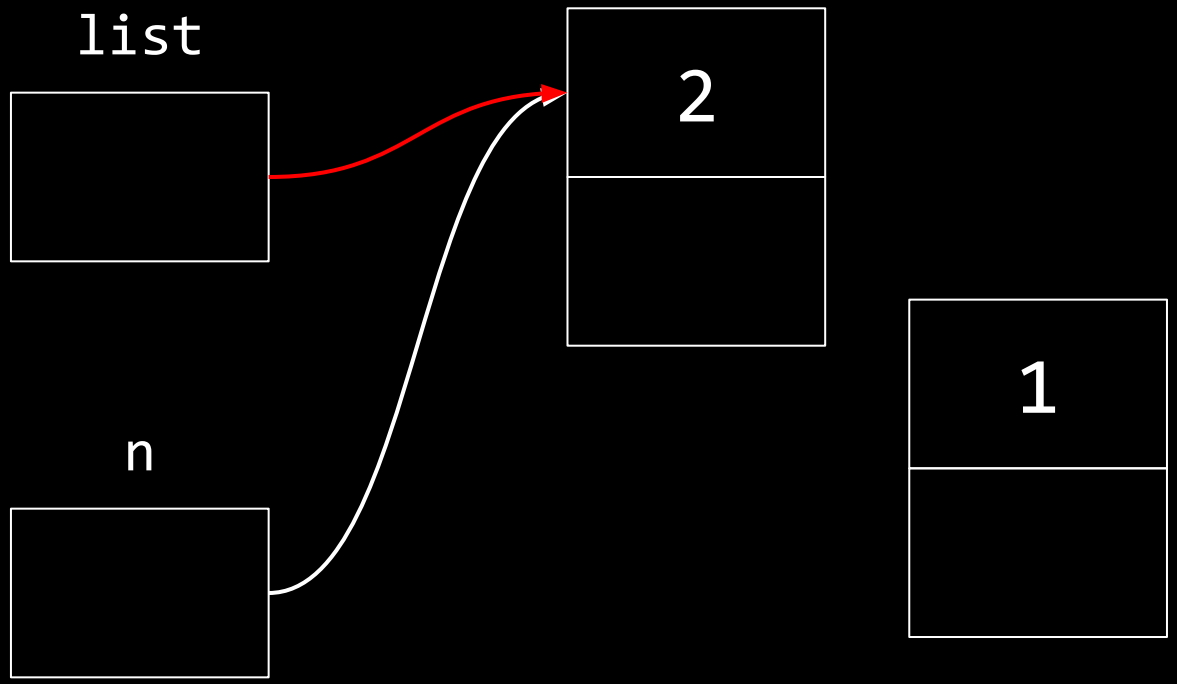
```
list = n;
```

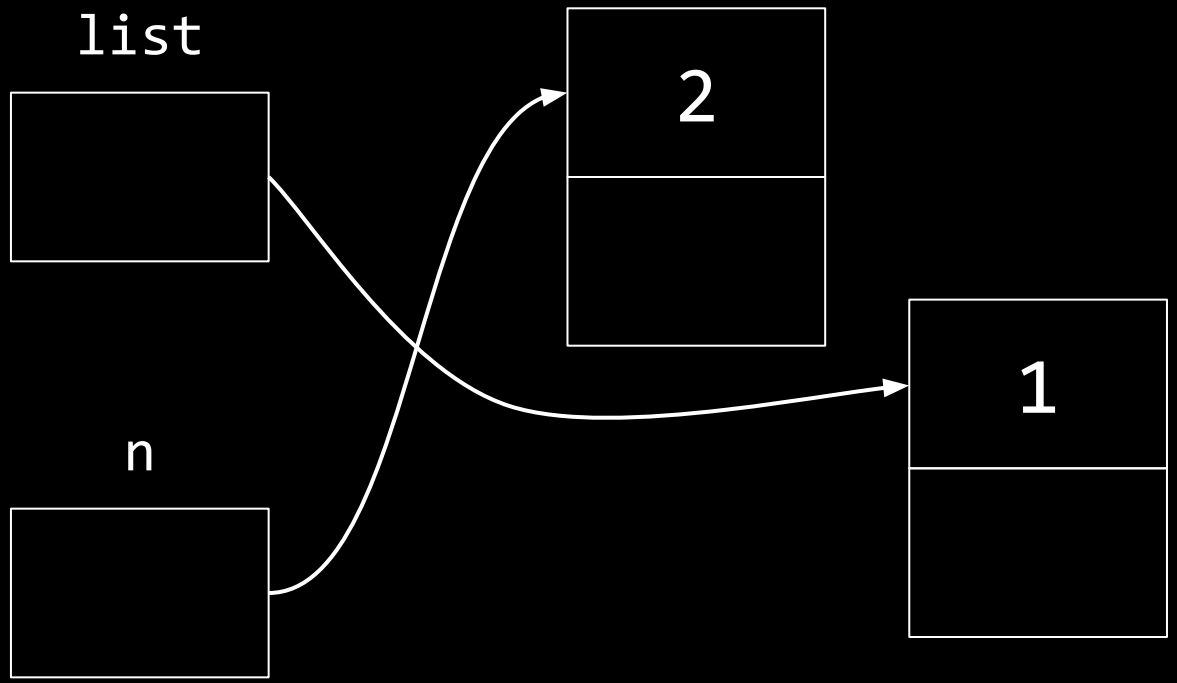


```
list = n;
```

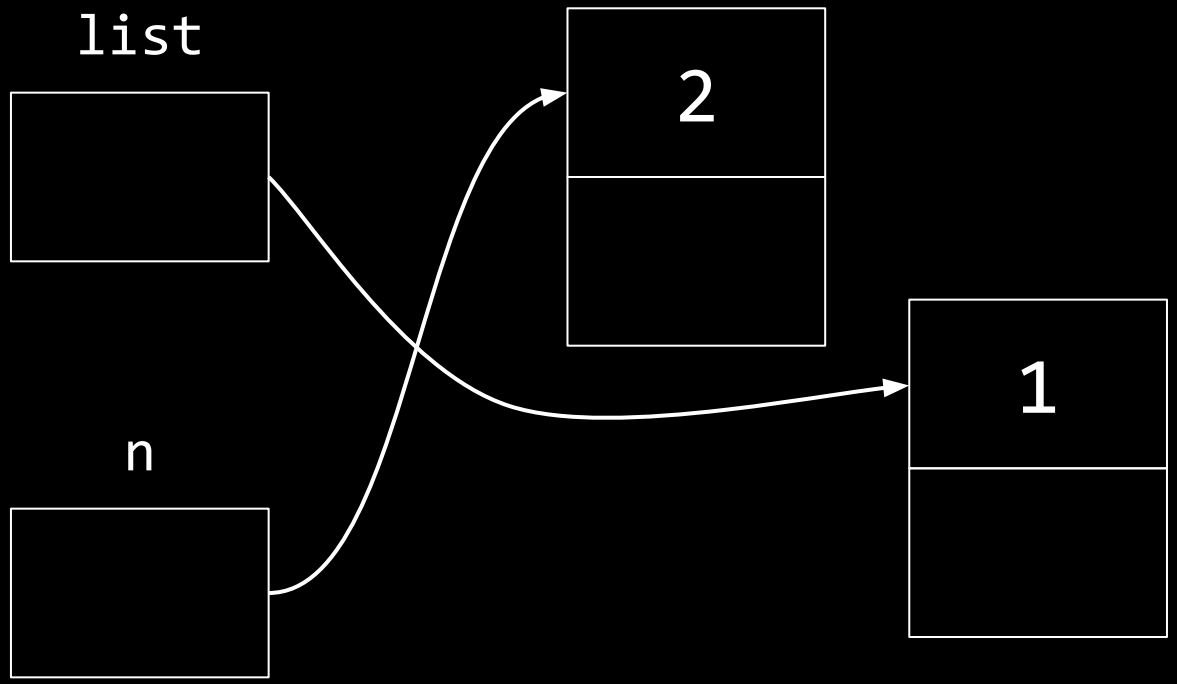


```
list = n;
```

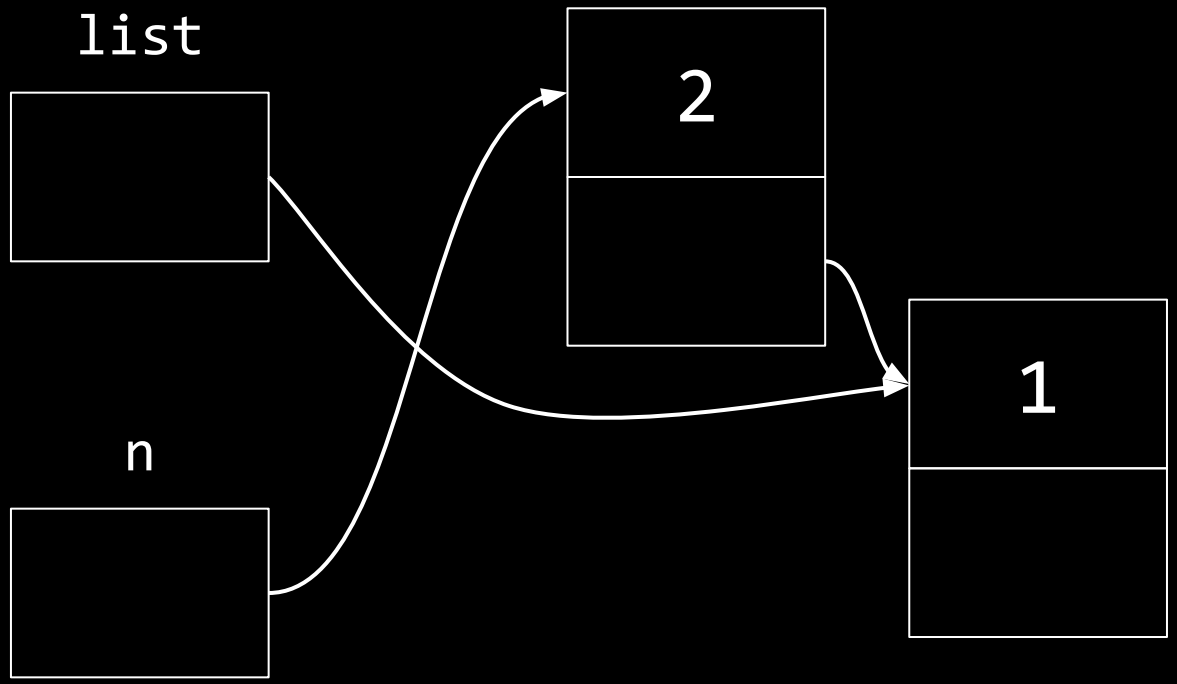



list
2
1
n

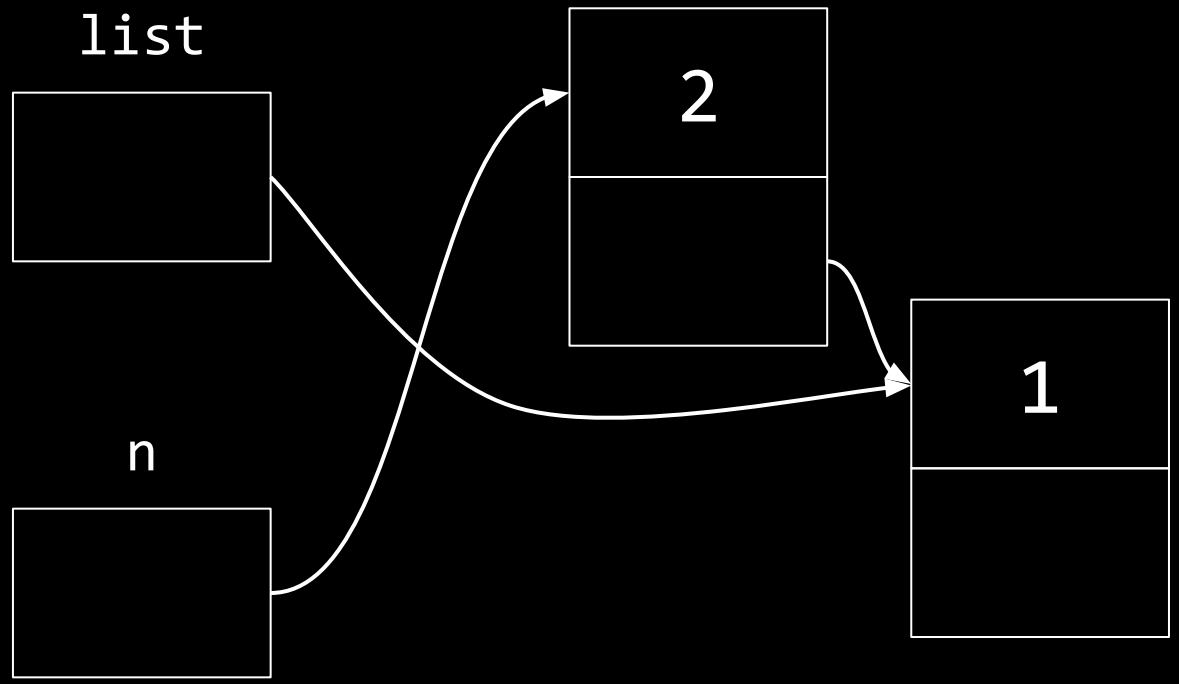
```
n->next = list;
```



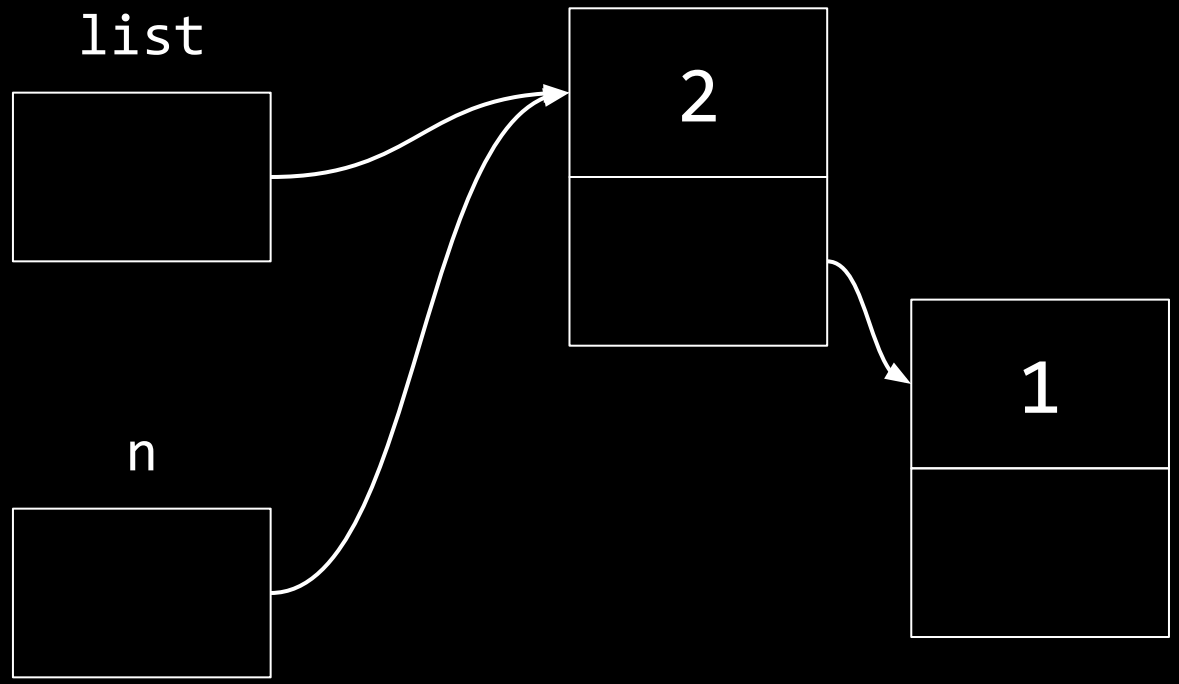
```
n->next = list;
```

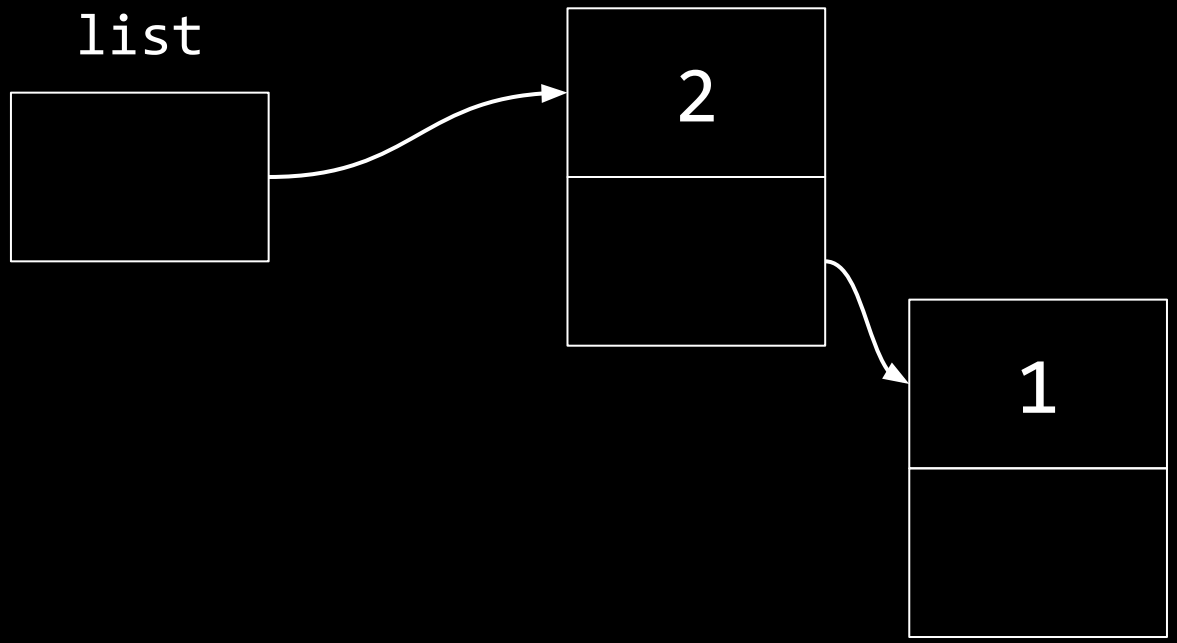


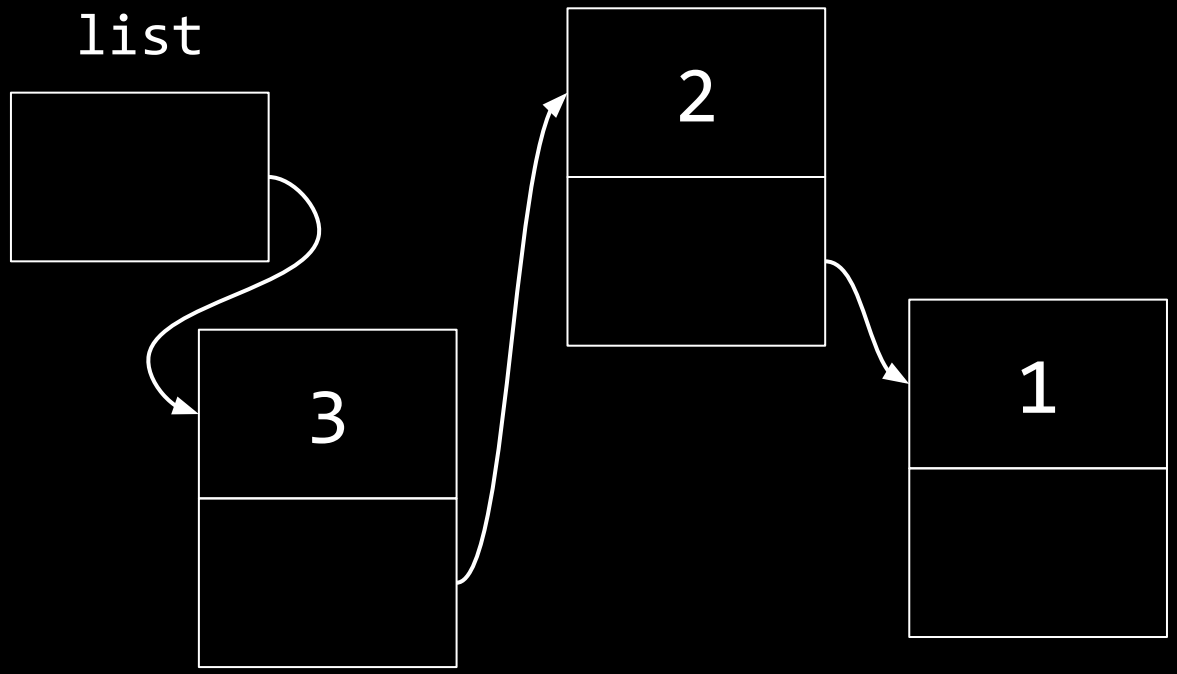
```
list = n;
```

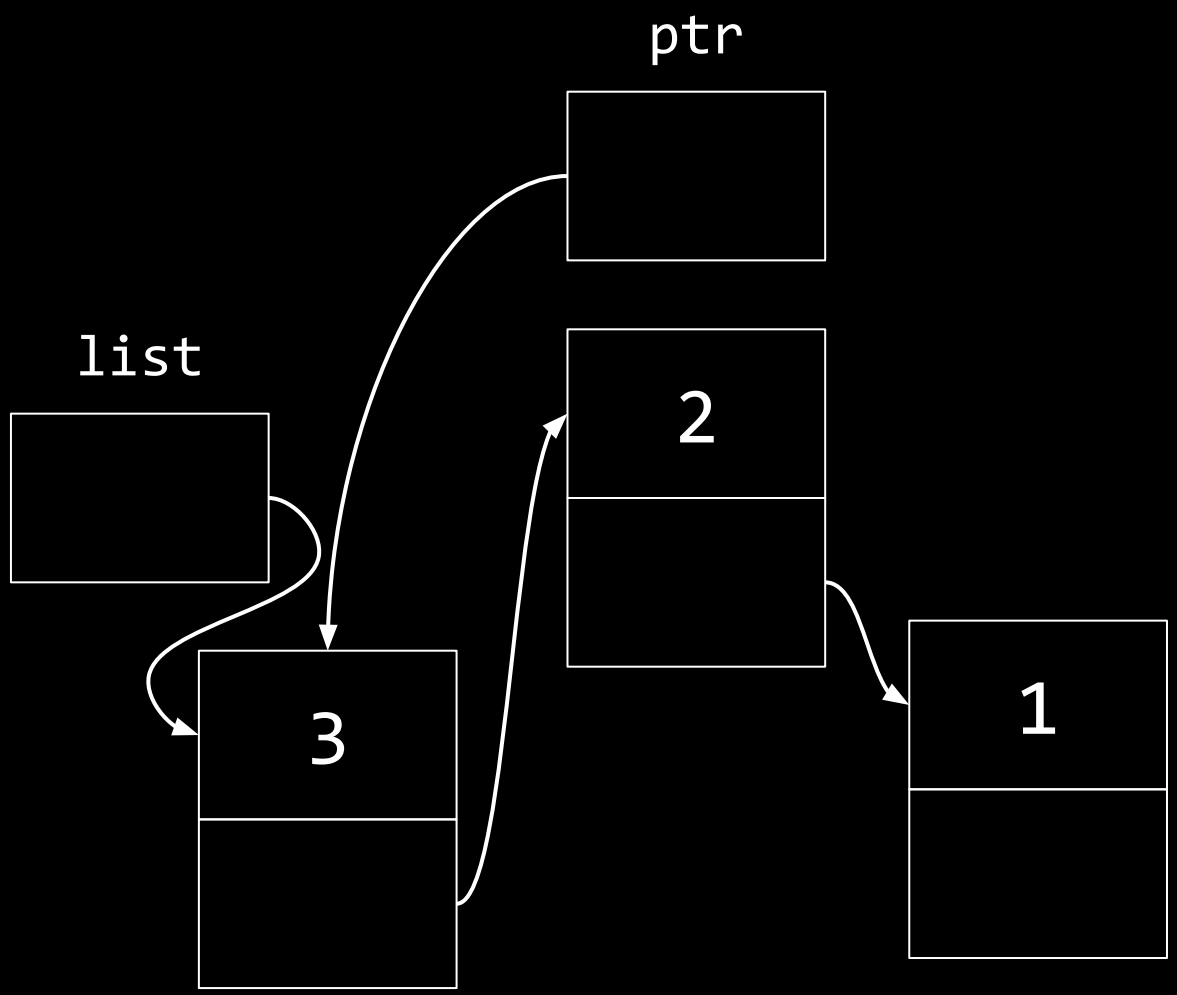


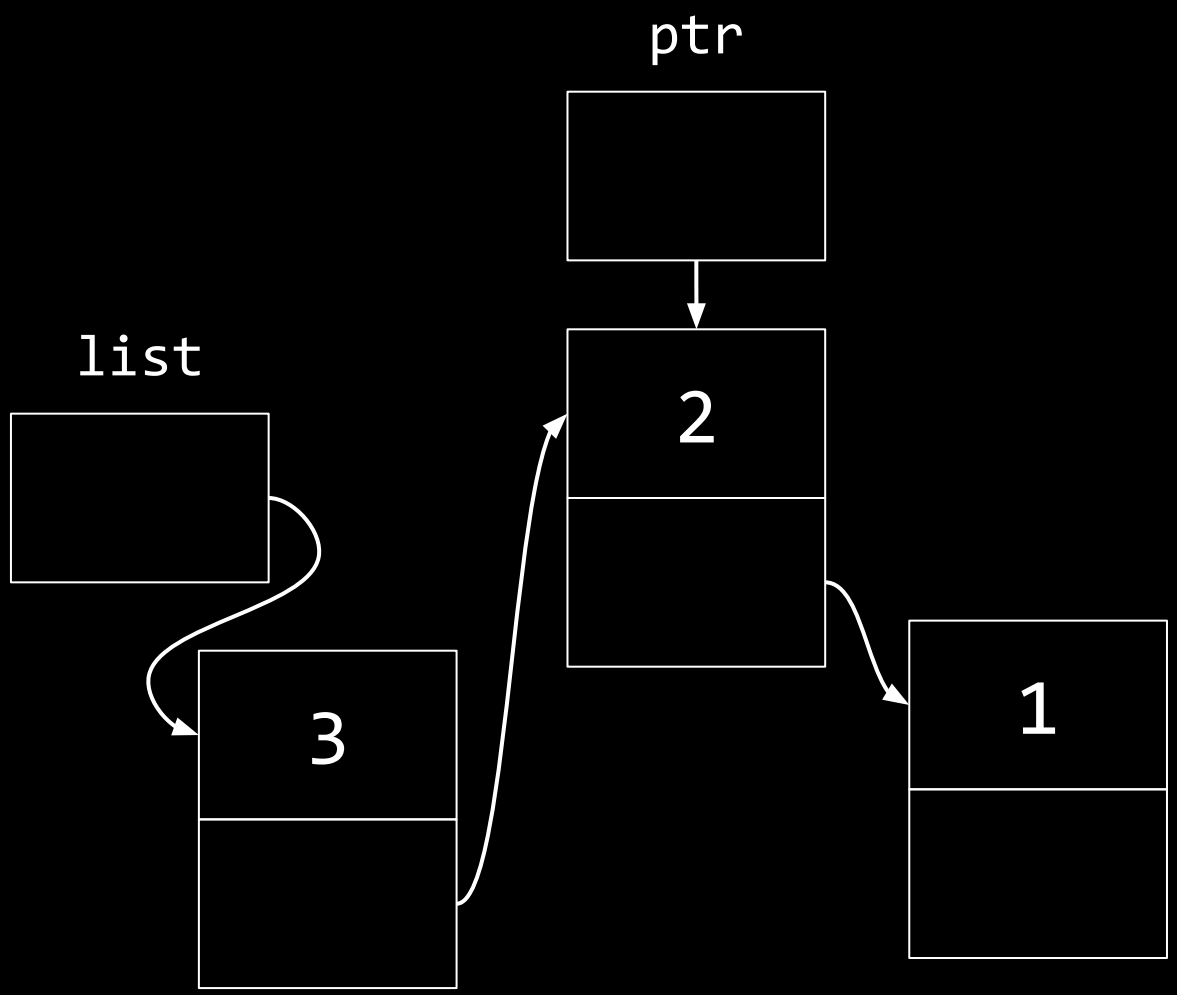
```
list = n;
```

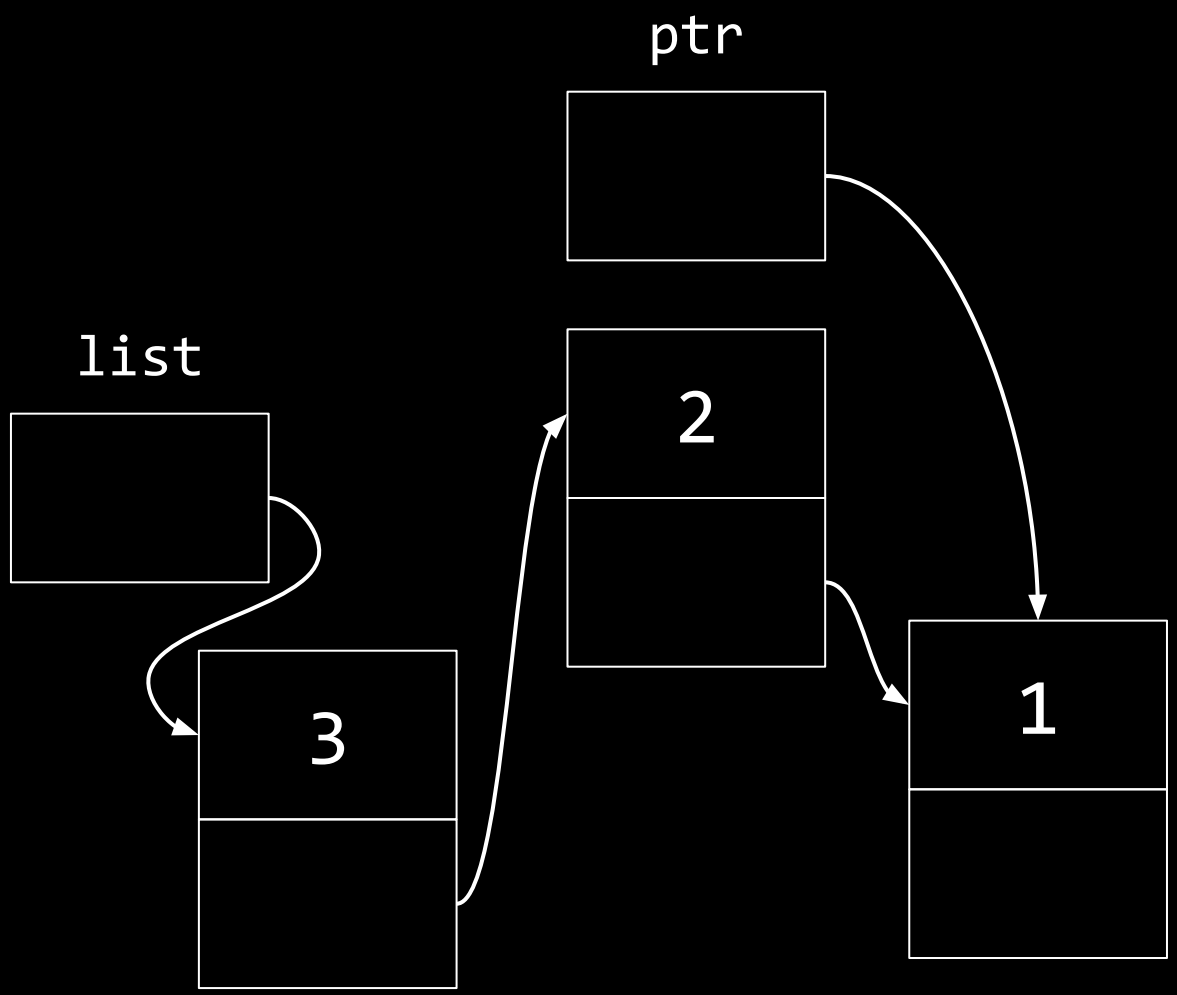


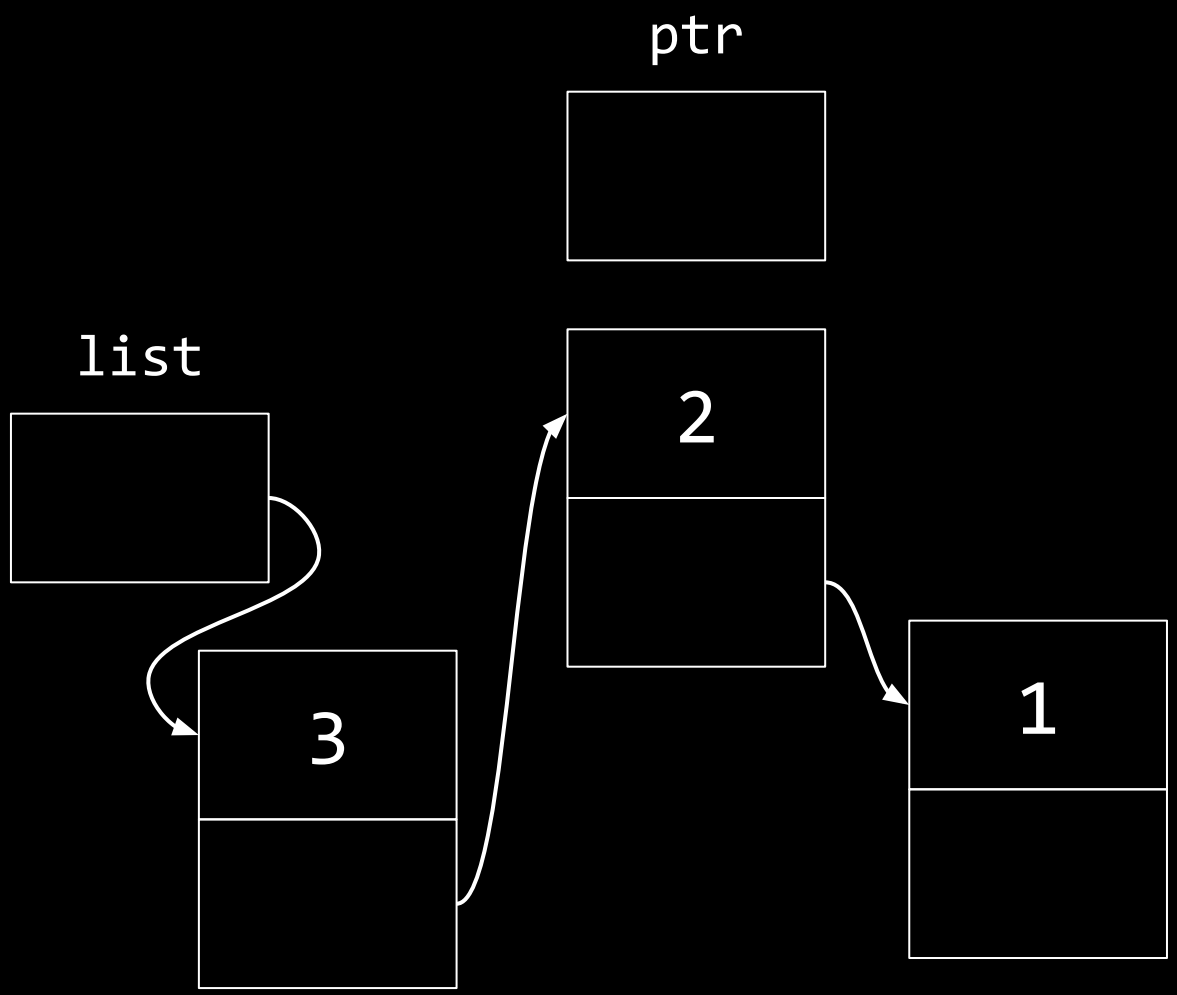

list
2
1

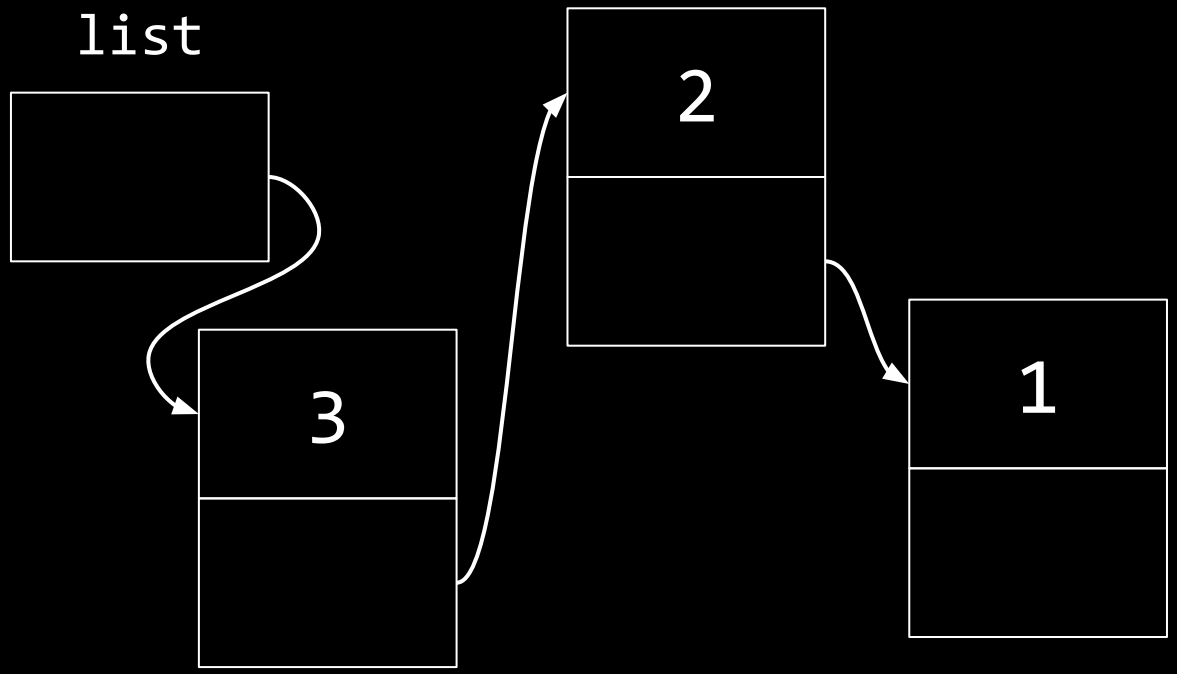

list
2
3
1


ptr
list
2
3
1


ptr
list
2
3
1


ptr
list
2
3
1


ptr
list
2
3
1


list
2
3
1

$O(n^2)$

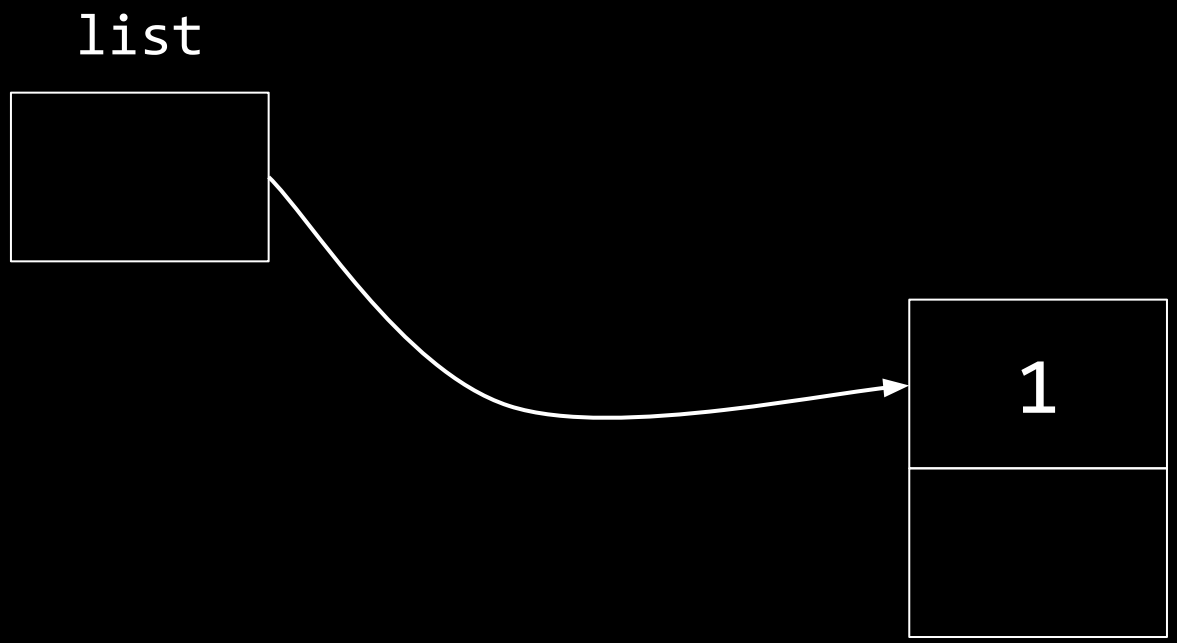
$O(n \log n)$

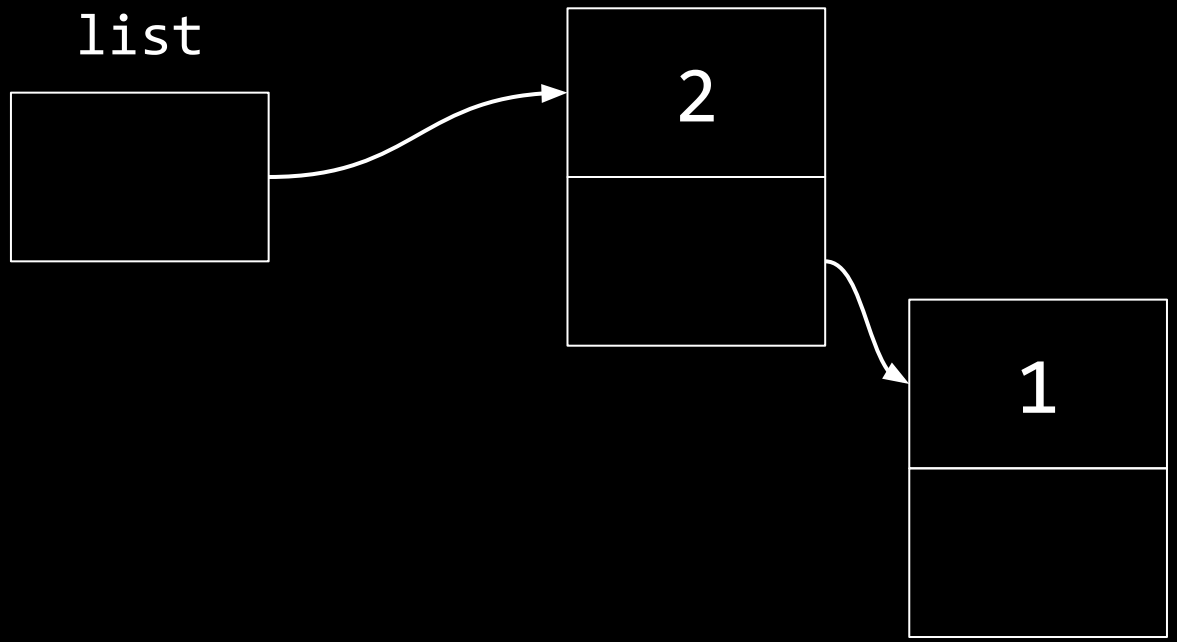
$O(n)$

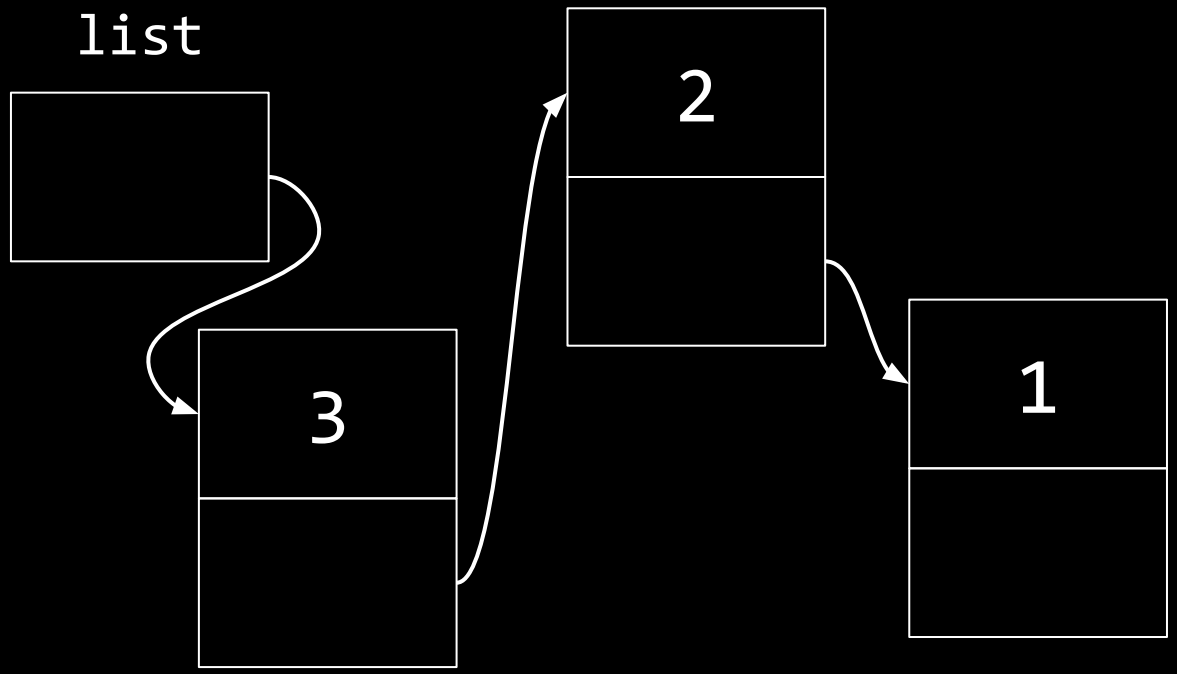
$O(\log n)$

$O(1)$

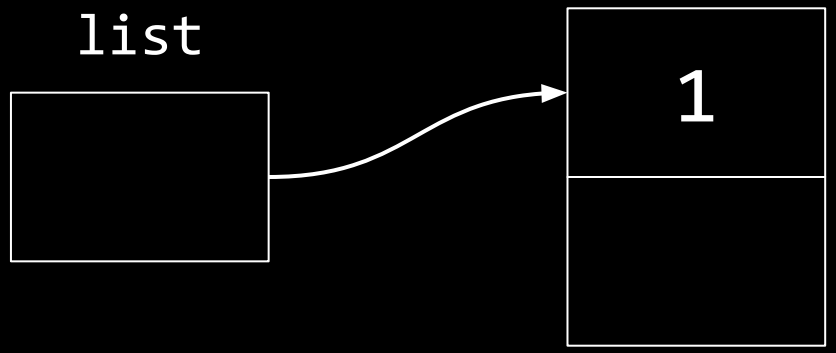
list

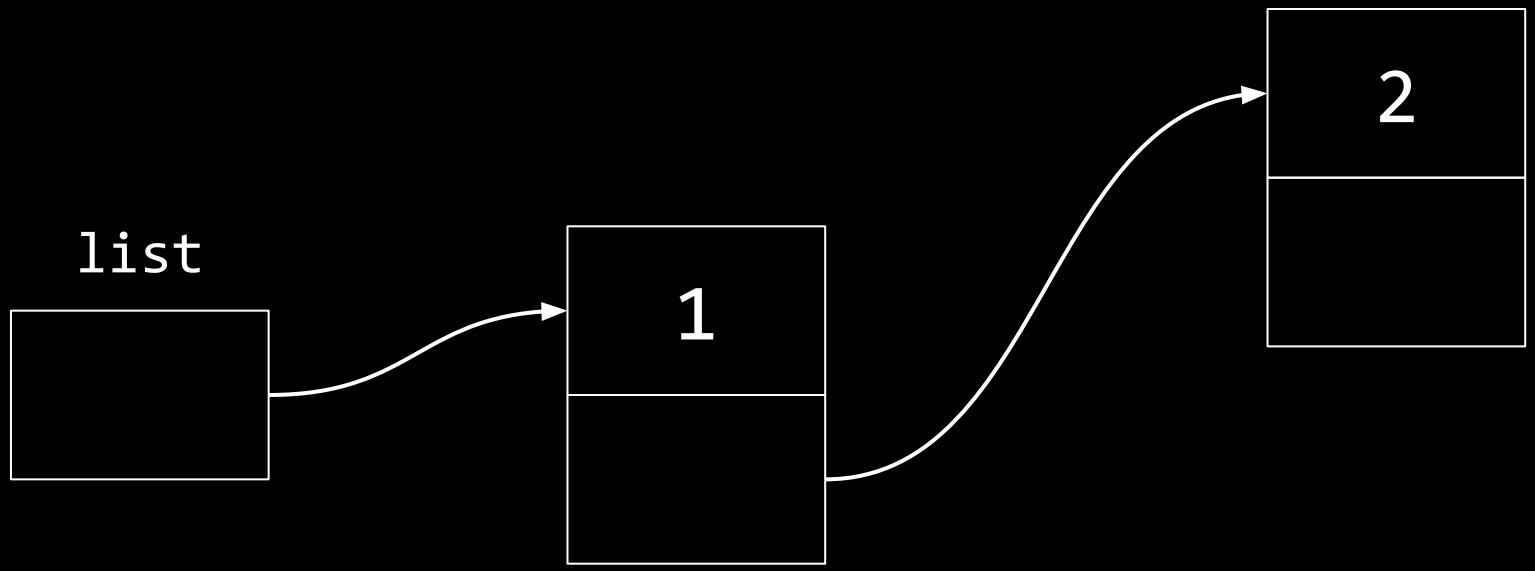

list
1

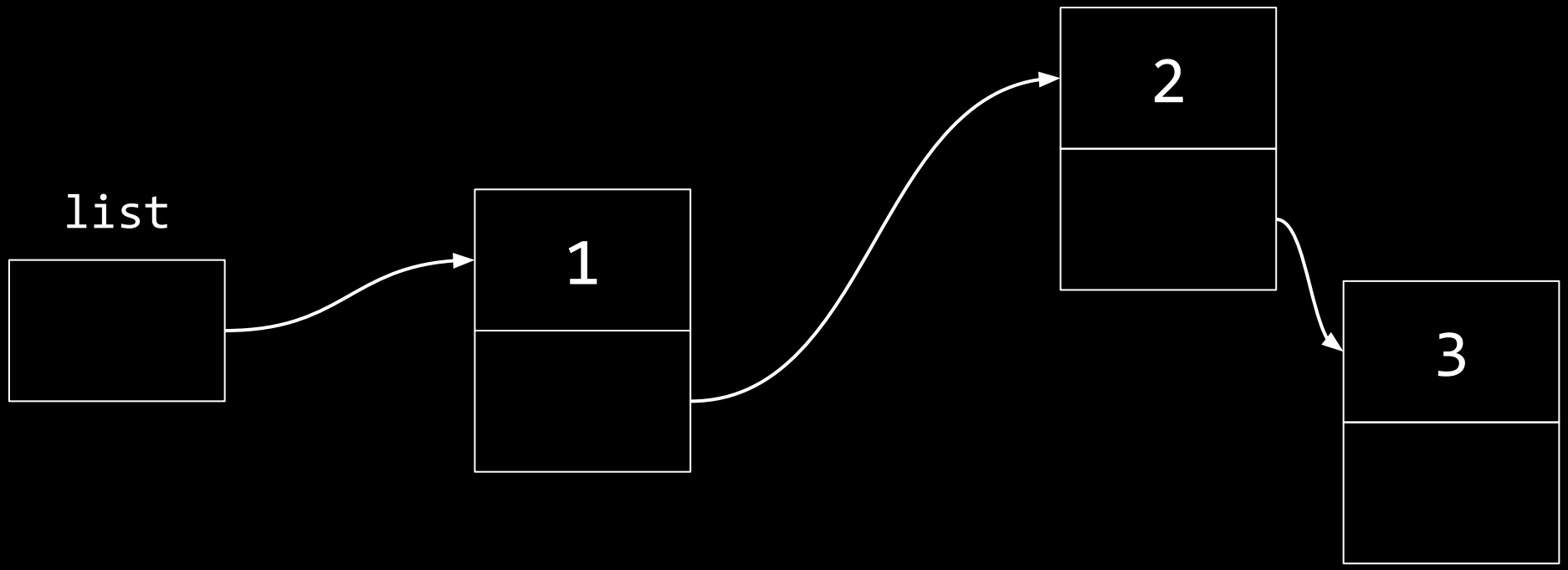

list
2
1


list
2
3
1

list


list
1


list
1
2


list
1
2
3

$O(n^2)$

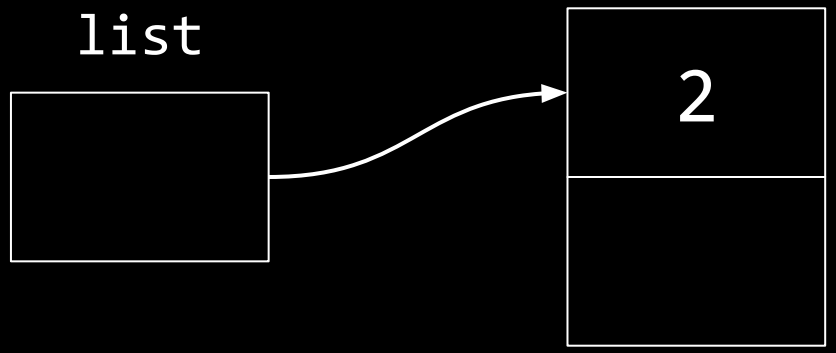
$O(n \log n)$

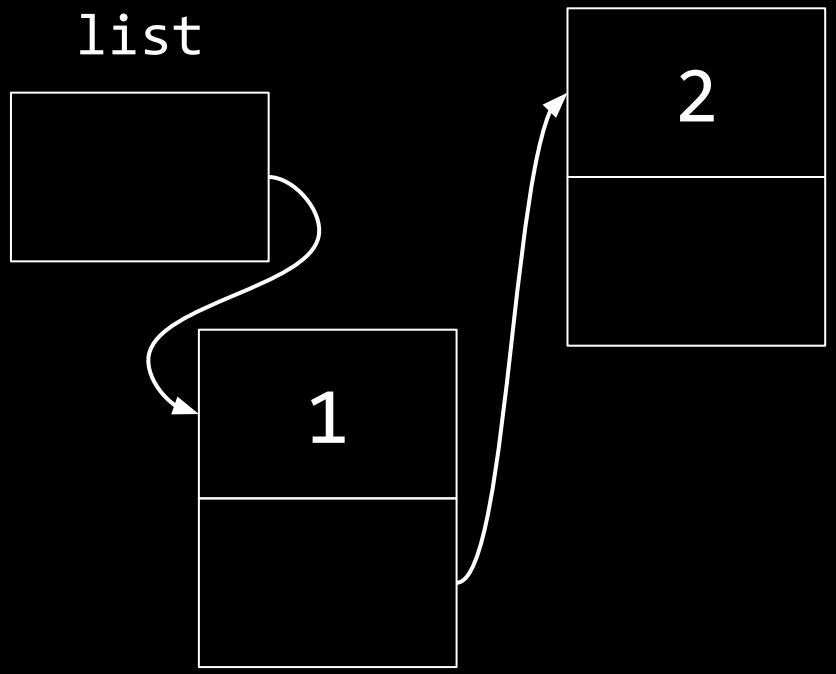
$O(n)$

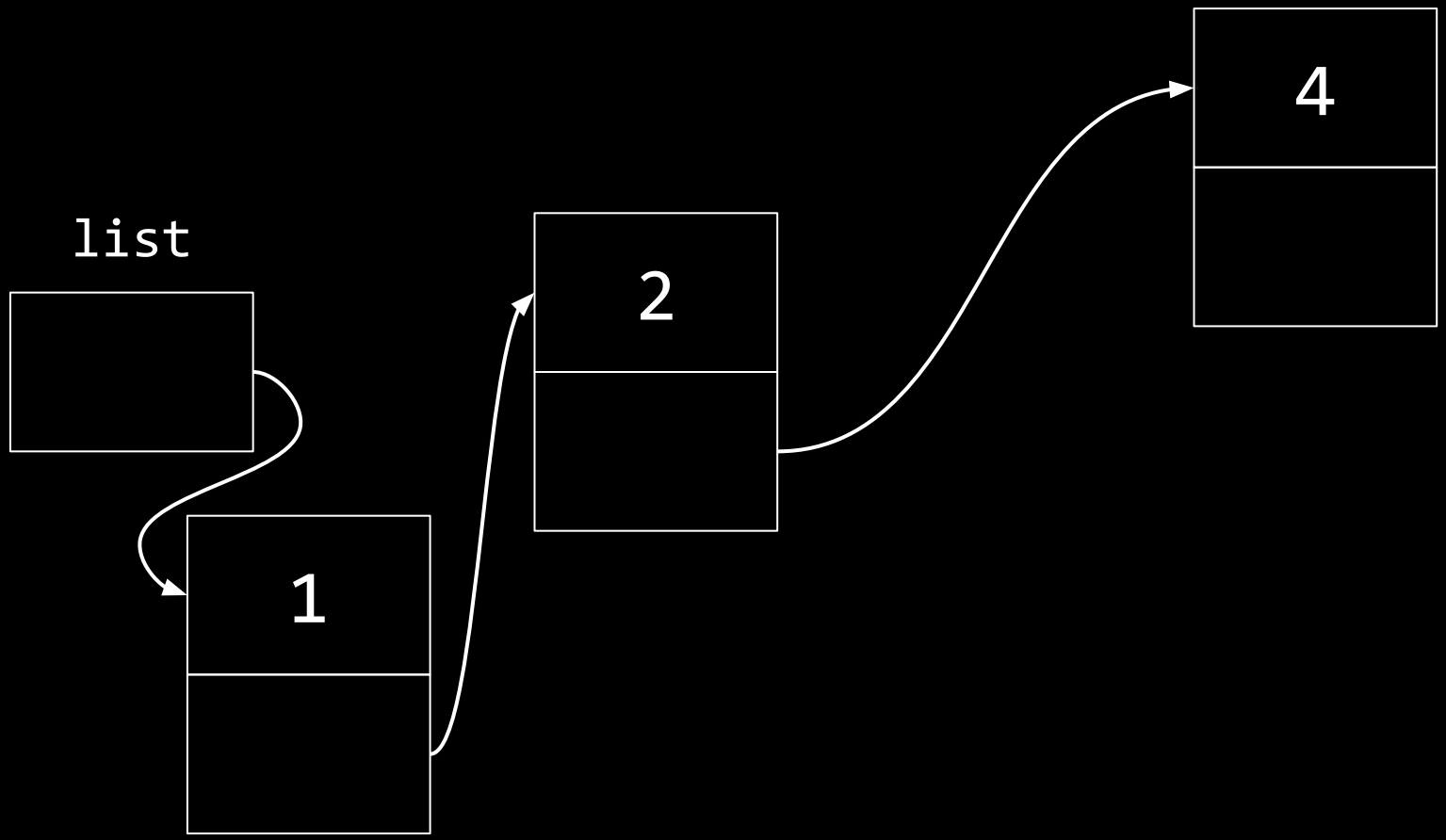
$O(\log n)$

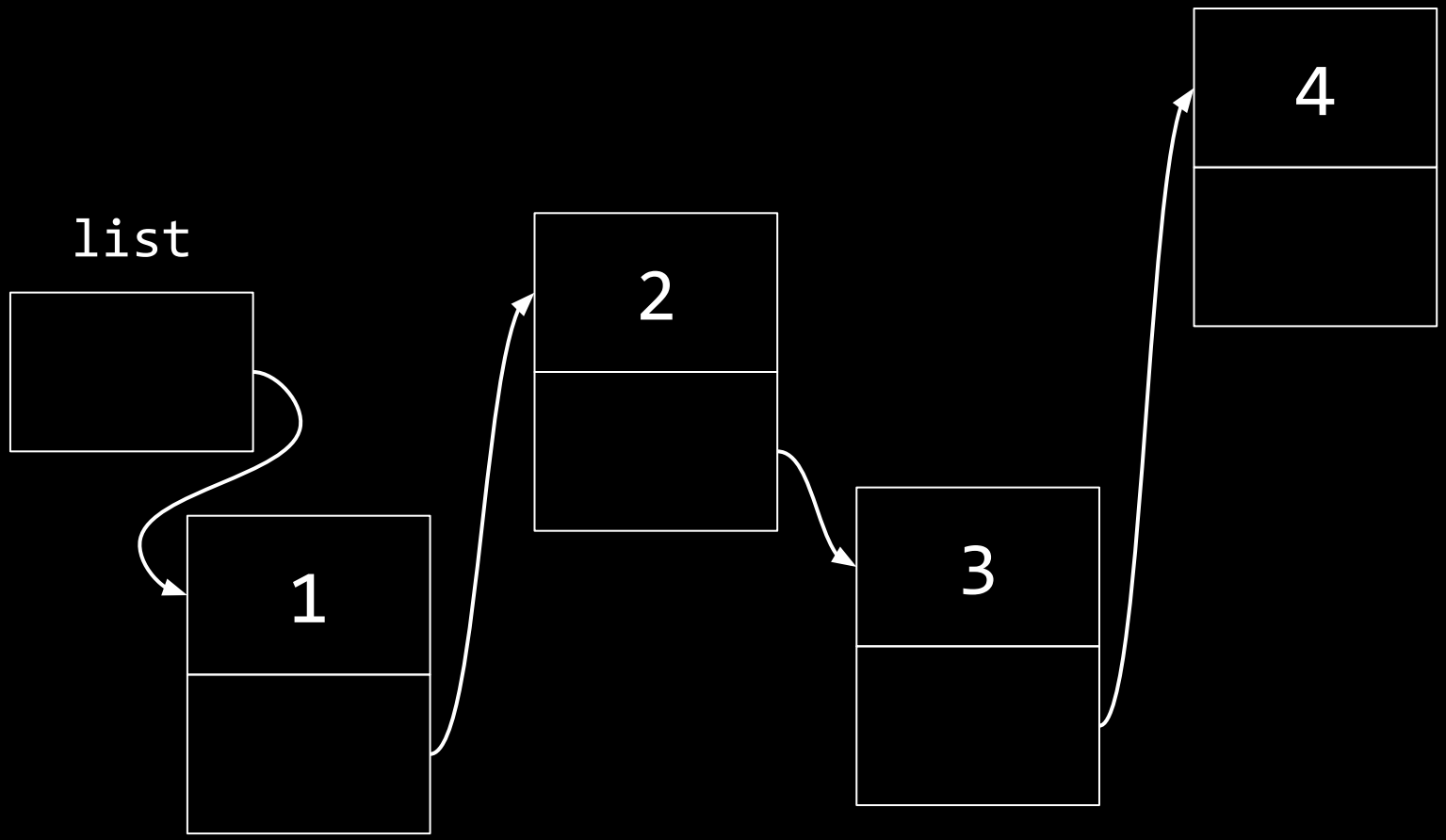
$O(1)$

list


list
2


list
2
1


list
1
2
4


list
1
2
3
4

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

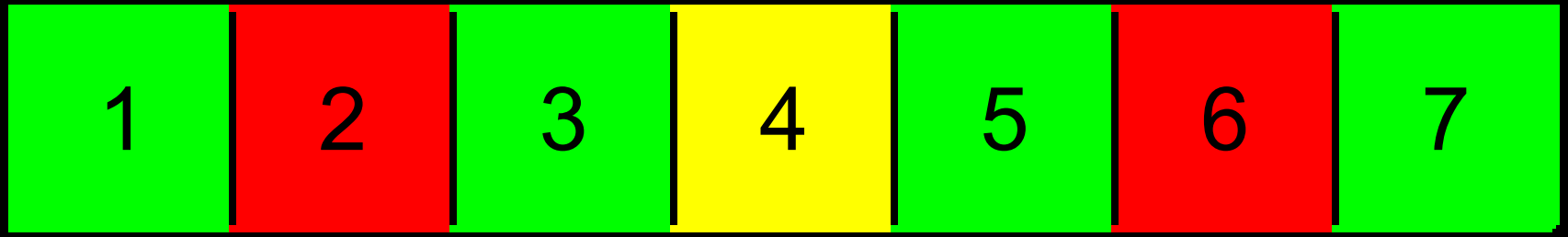
trees

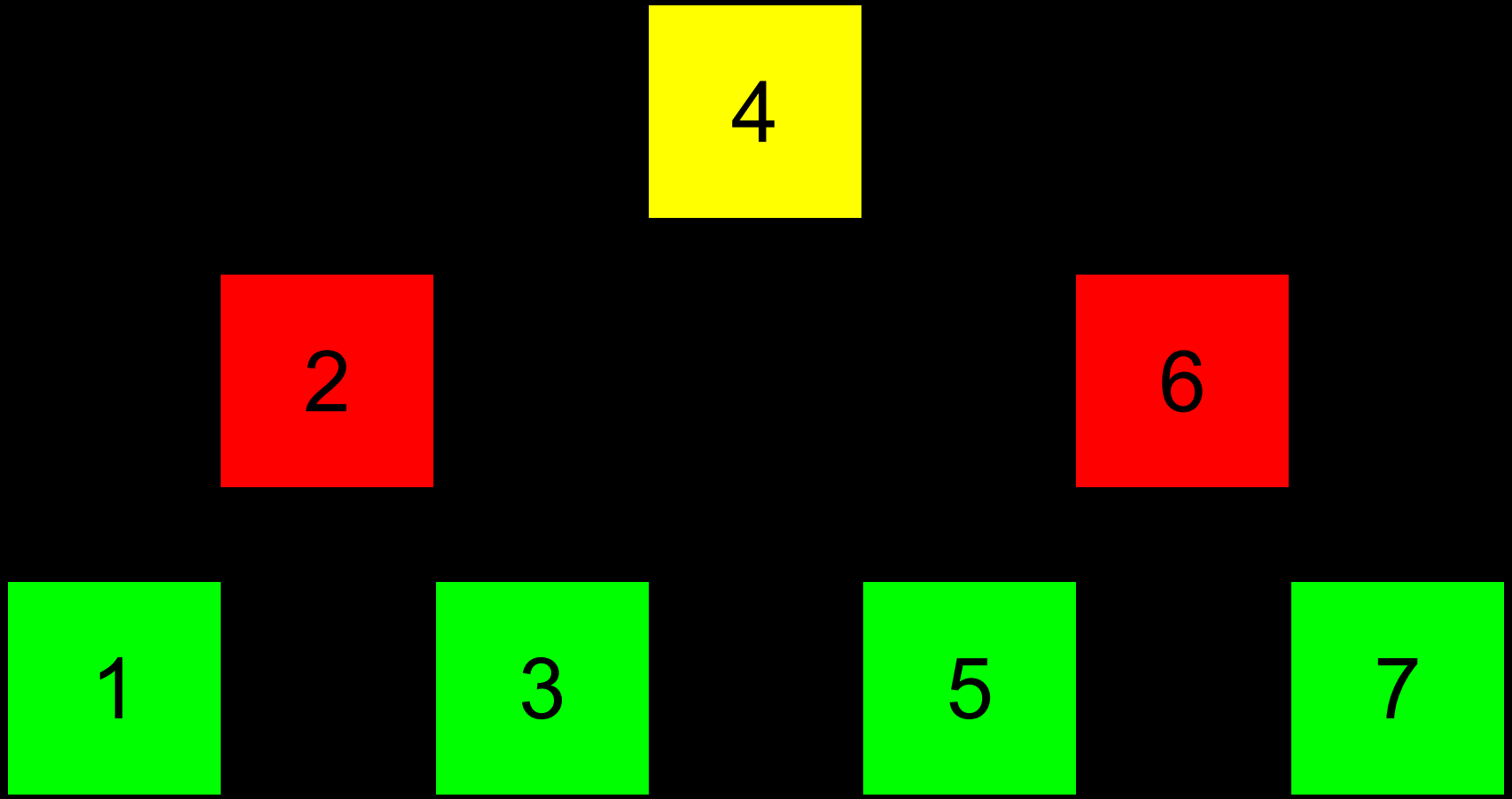
binary search trees

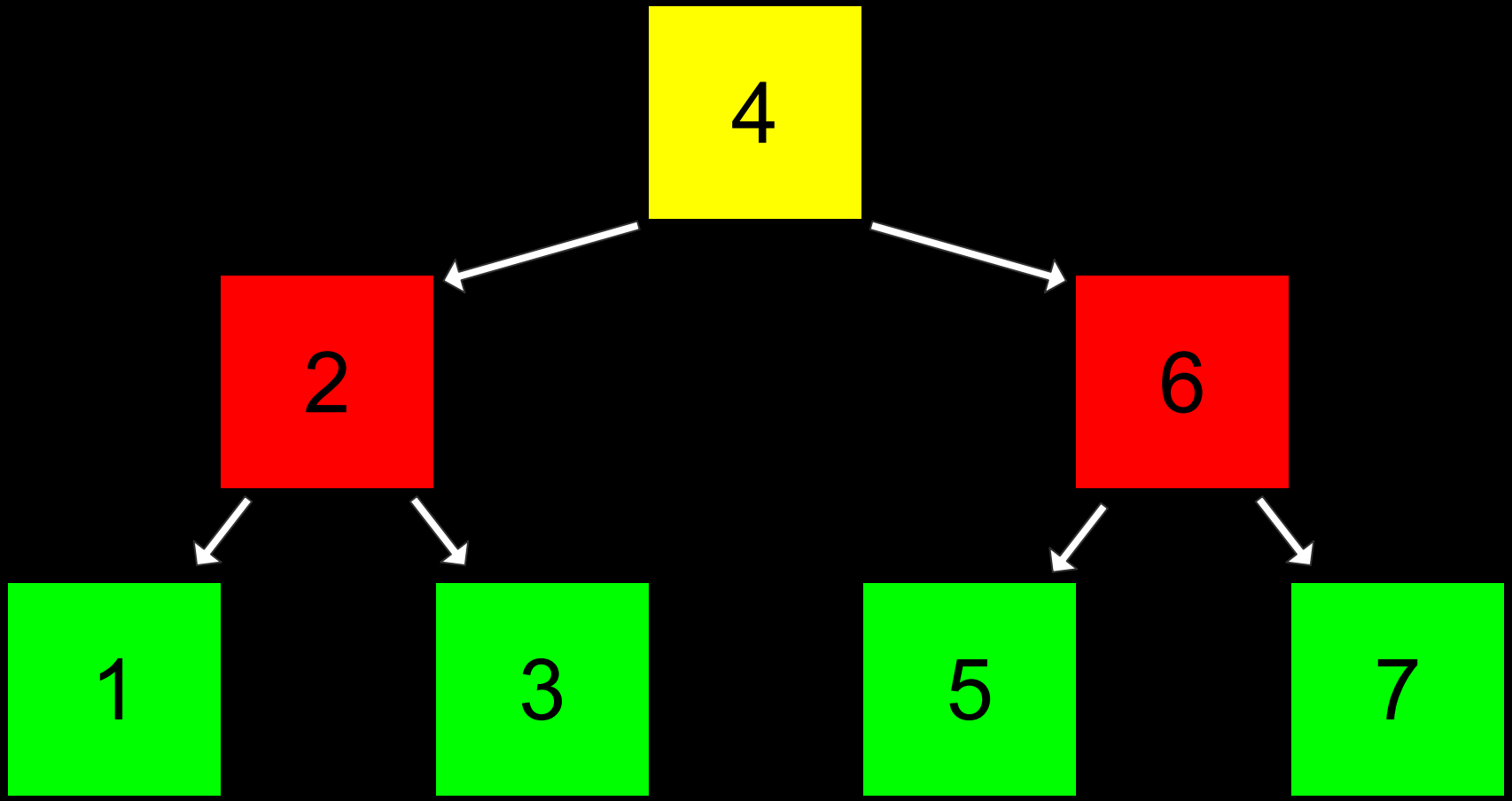
1	2	3	4	5	6	7

1	2	3	4	5	6	7

1	2	3	4	5	6	7


1
2
3
4
5
6
7


4
2
6
1
3
5
7

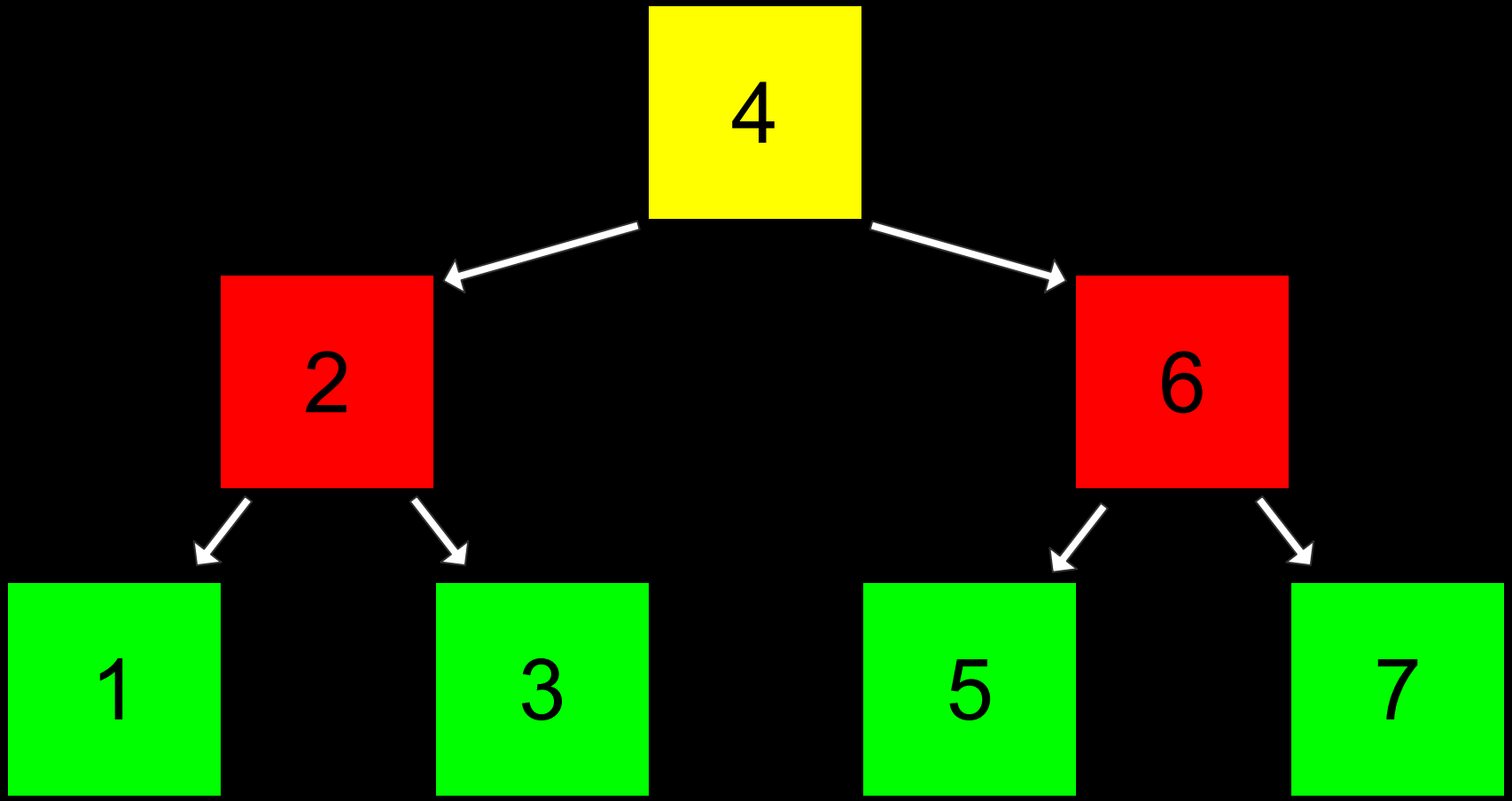

4
2
6
1
3
5
7

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

```
typedef struct node
{
    int number;

}
node;
```

```
typedef struct node
{
    int number;


}
node;
```

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


4
2
6
1
3
5
7

```
bool search(node *tree, int number)
{

}
```

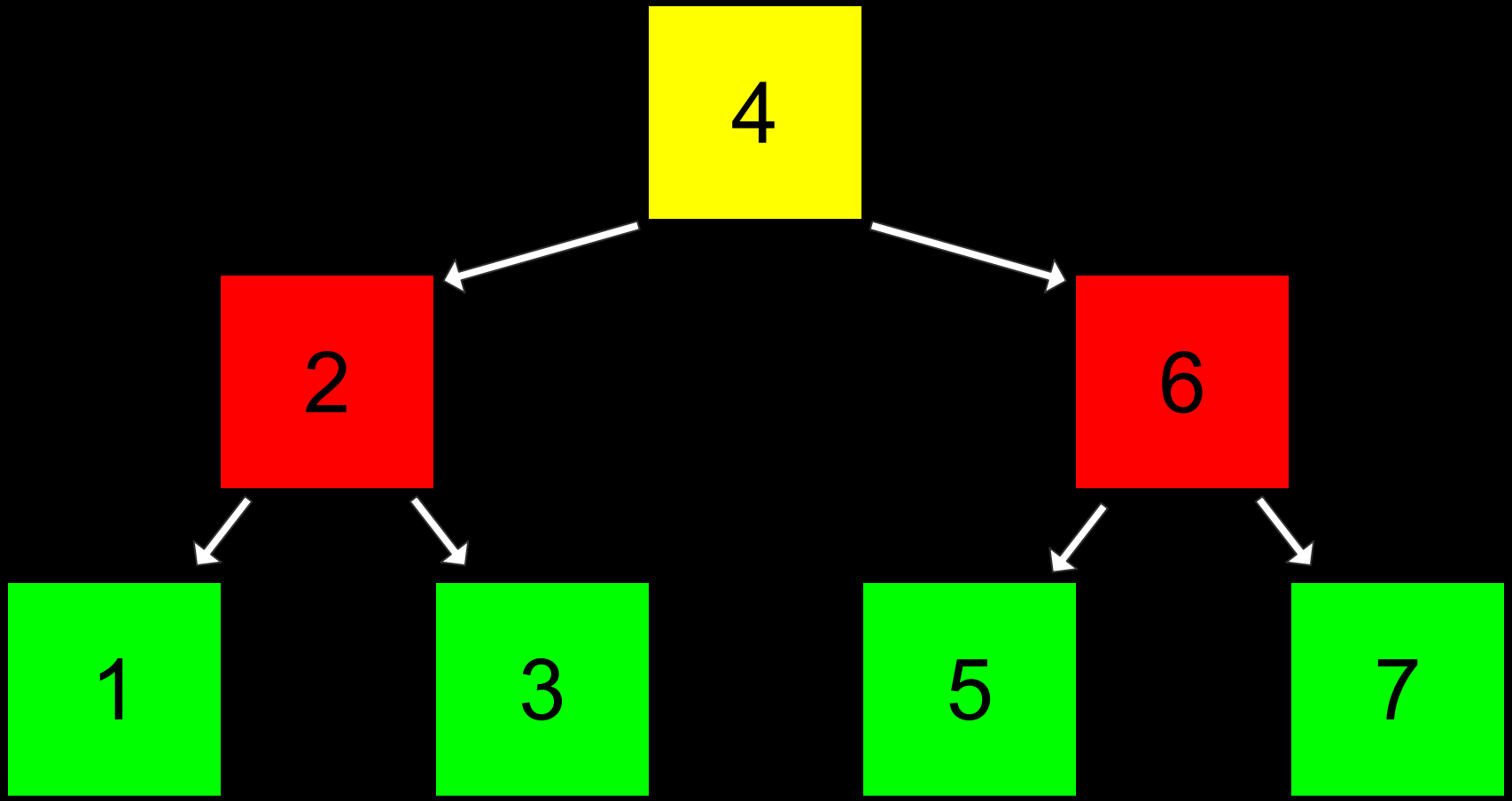
```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }



}
```

```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (number < tree->number)
    {
        return search(tree->left, number);
    }









}
```

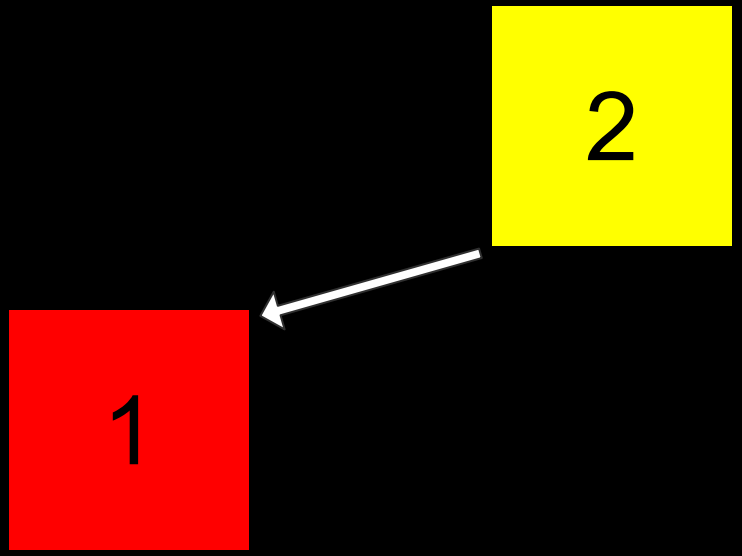
```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (number < tree->number)
    {
        return search(tree->left, number);
    }
    else if (number > tree->number)
    {
        return search(tree->right, number);
    }




}
```

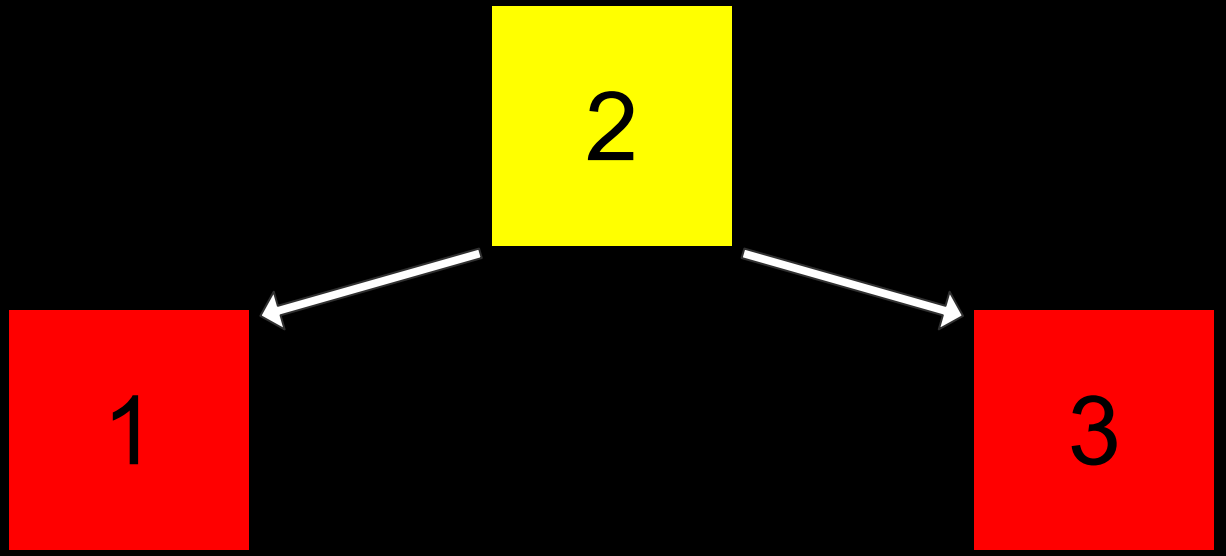
```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (number < tree->number)
    {
        return search(tree->left, number);
    }
    else if (number > tree->number)
    {
        return search(tree->right, number);
    }
    else if (number == tree->number)
    {
        return true;
    }
}
```

```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (number < tree->number)
    {
        return search(tree->left, number);
    }
    else if (number > tree->number)
    {
        return search(tree->right, number);
    }
    else
    {
        return true;
    }
}
```

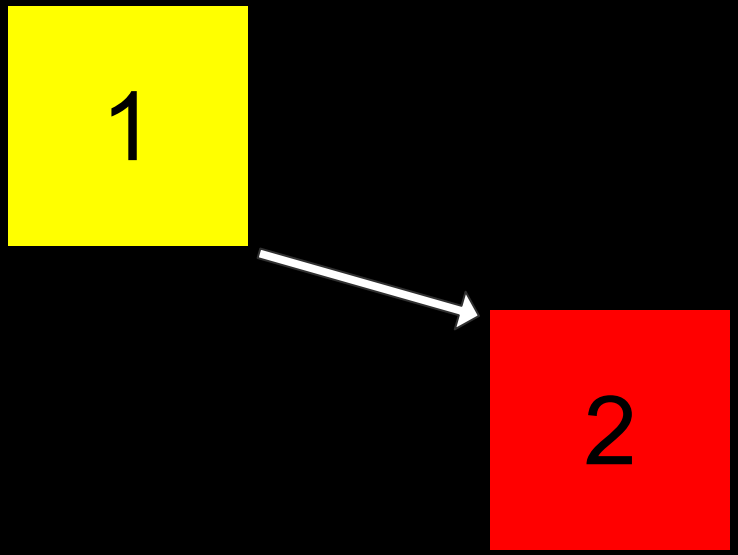

4
2
6
1
3
5
7

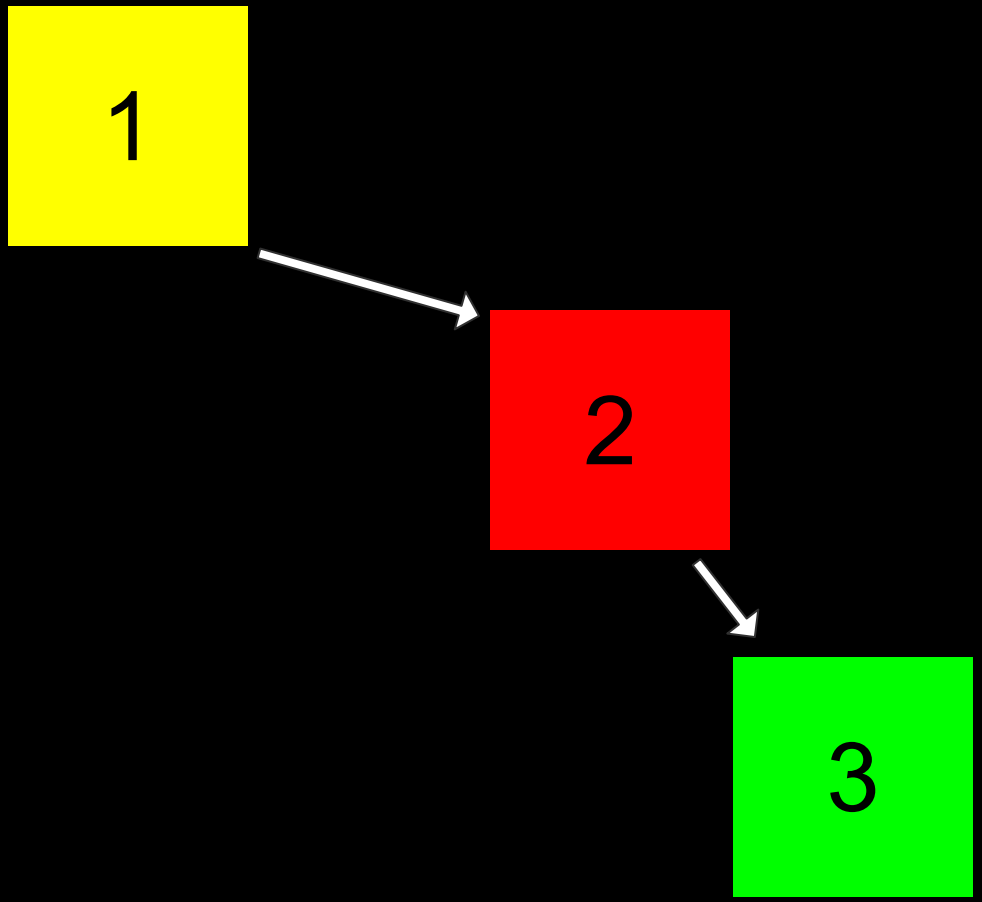
2


2
1


2
1
3

1


1
2


1
2
3

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

dictionaries

word	definition

key	value

Groups

Contacts

Search

A

Albus

C

Cedric

D

Draco

F

Fred

G

George

Ginny

H

Hagrid

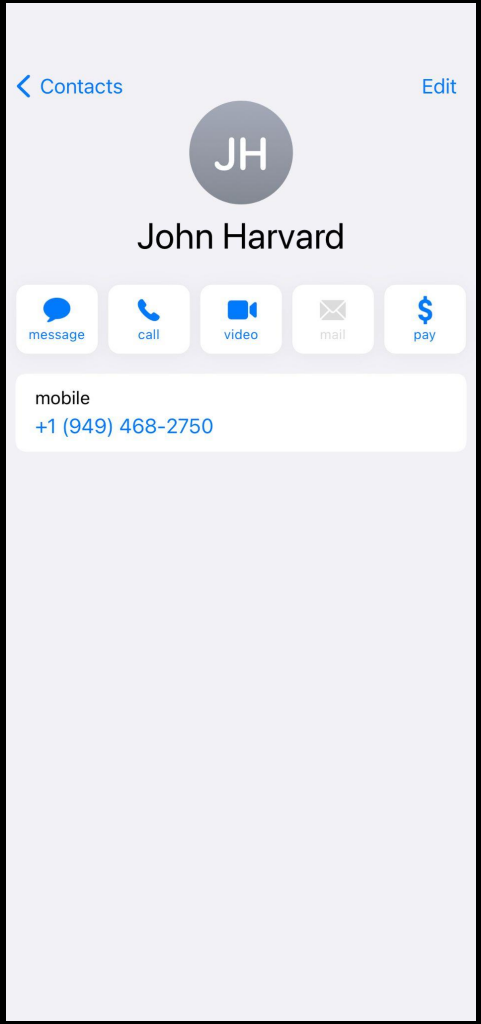
Harry

Hermione

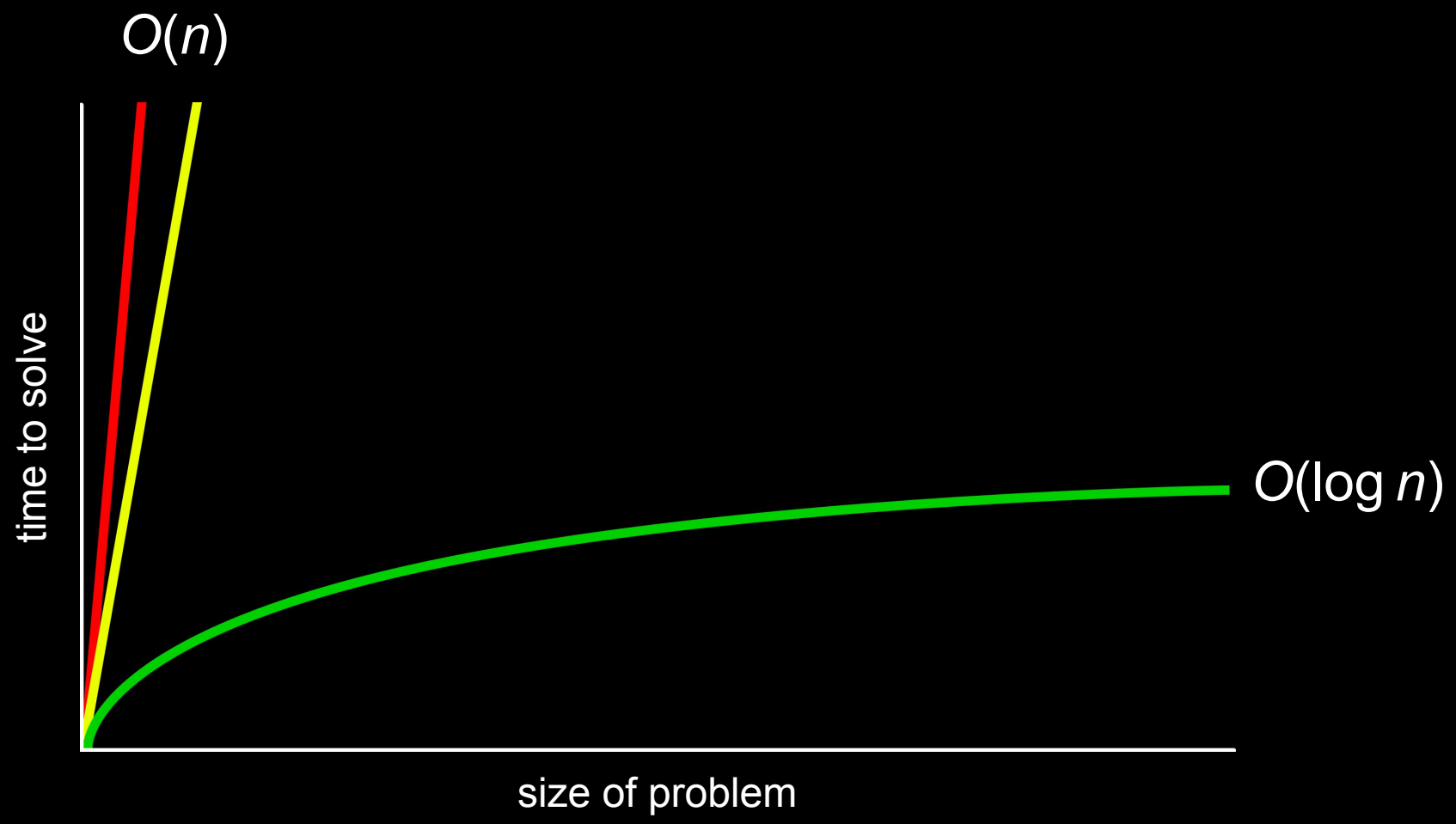
J

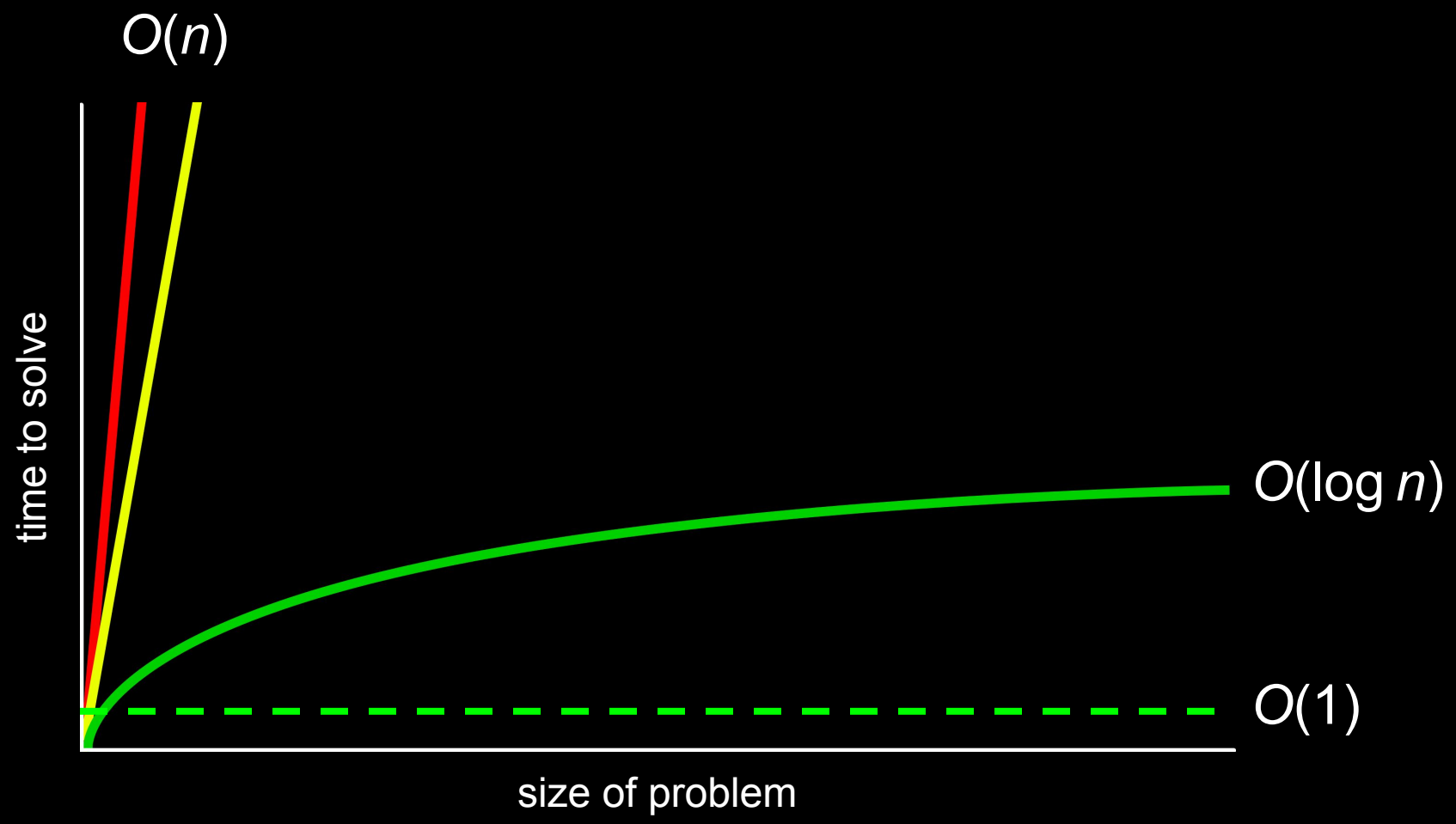
James

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z #


Contacts
Edit
JH
John Harvard
message
call
video
mail
pay
mobile
+1 (949) 468-2750

name	number


$O(n)$
time to solve
$O(\log n)$
size of problem


O(n)
time to solve
O(log n)
O(1)
size of problem

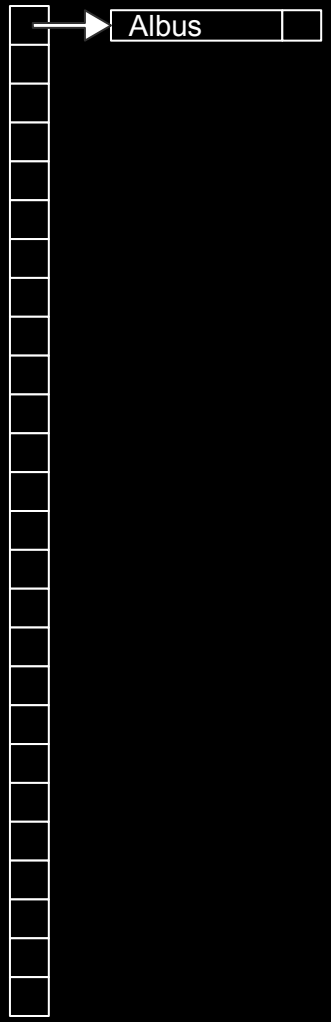
hashing

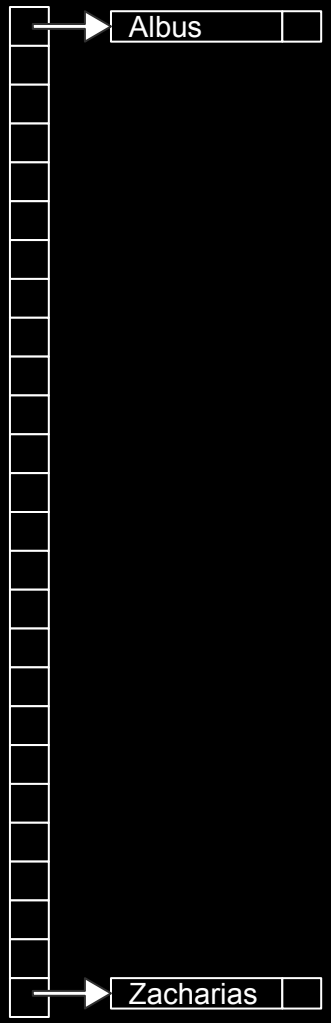
hash function

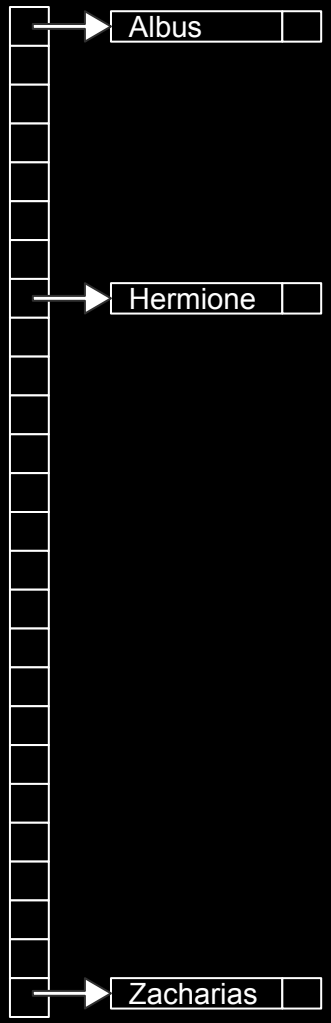
hash tables

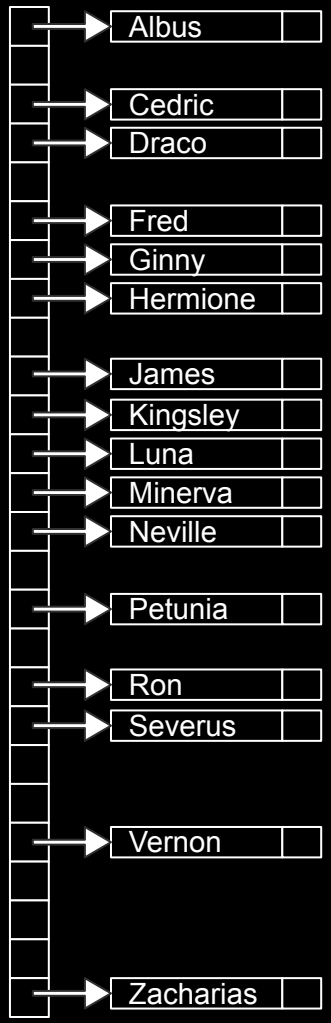
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

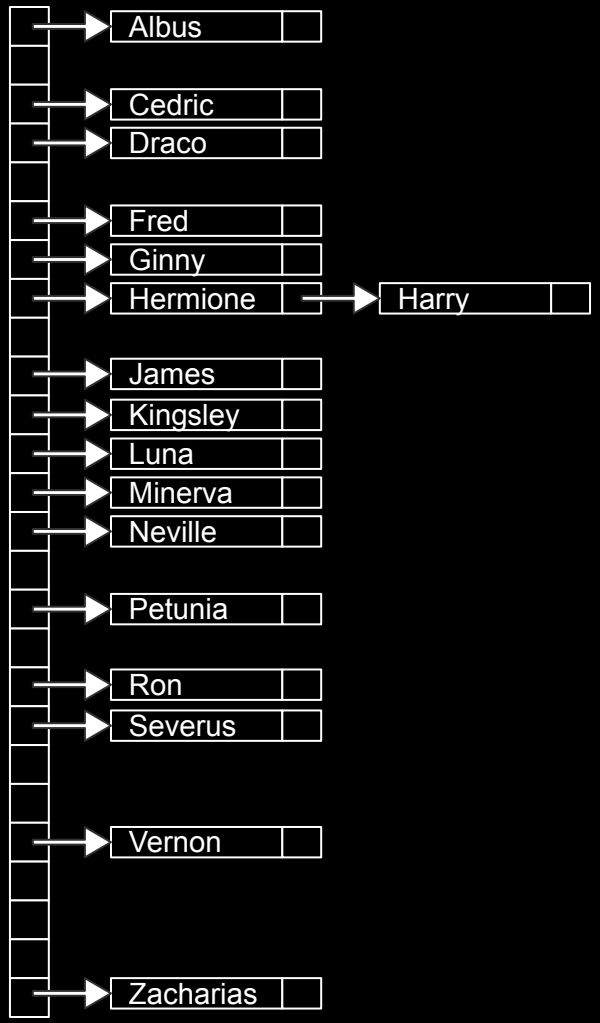
A	
B	
C	
D	
E	
F	
G	
H	
I	
J	
K	
L	
M	
N	
O	
P	
Q	
R	
S	
T	
U	
V	
W	
X	
Y	
Z	


Albus


Albus
Zacharias

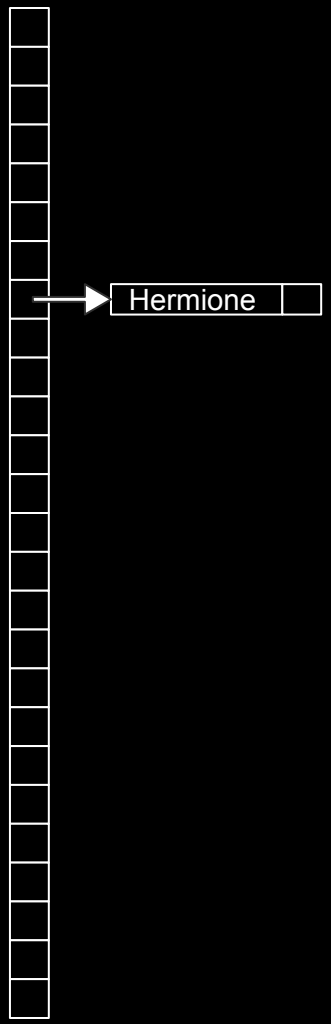

Albus
Hermione
Zacharias

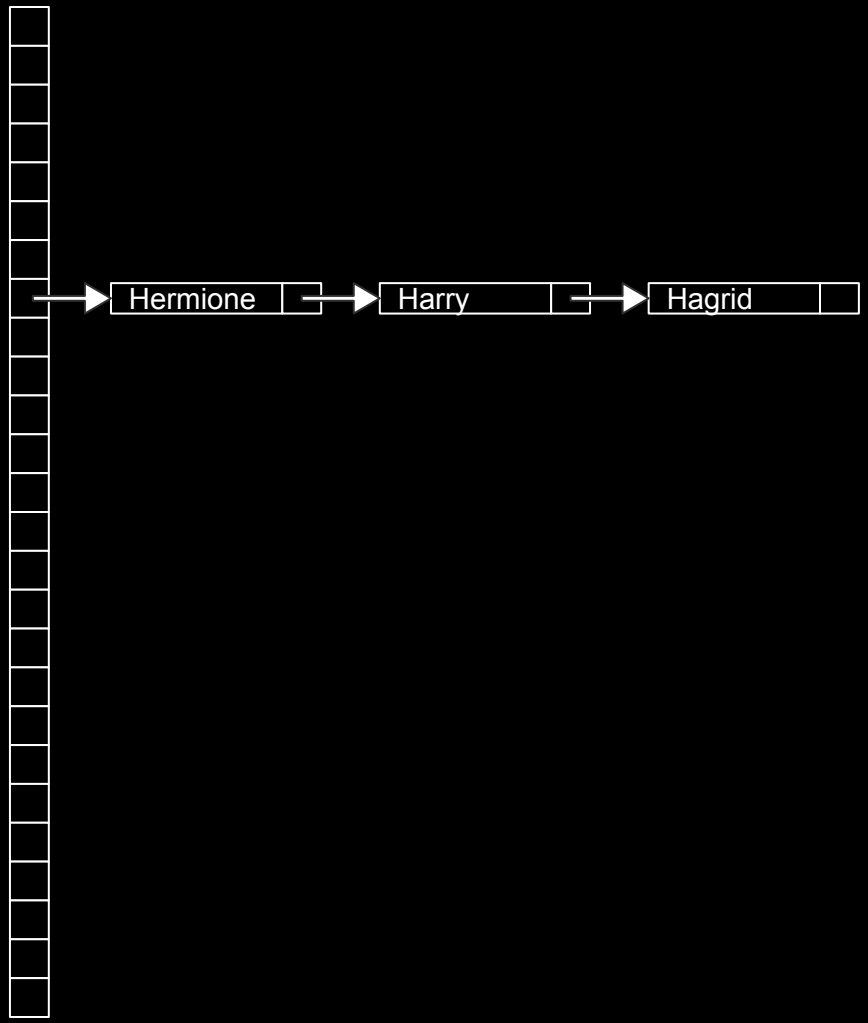

Albus
Cedric
Draco
Fred
Ginny
Hermione
James
Kingsley
Luna
Minerva
Neville
Petunia
Ron
Severus
Vernon
Zacharias

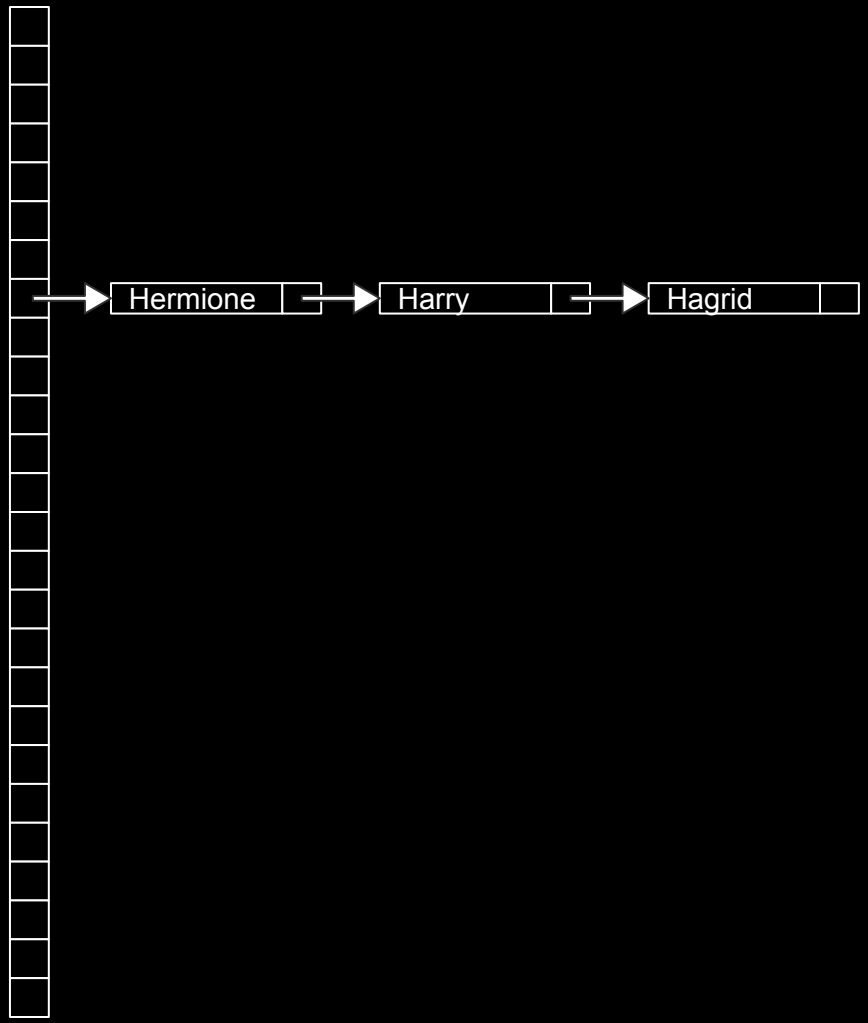

Albus
Cedric
Draco
Fred
Ginny
Hermione
Harry
James
Kingsley
Luna
Minerva
Neville
Petunia
Ron
Severus
Vernon
Zacharias

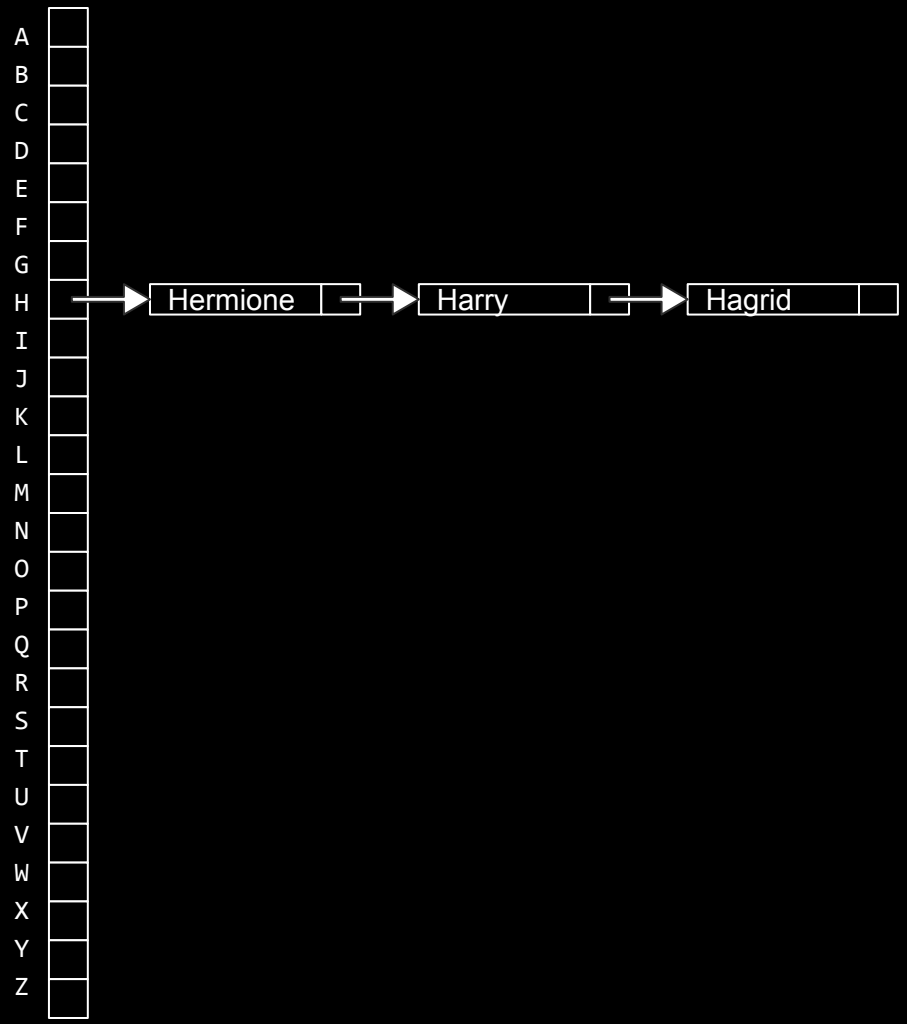

Albus
Cedric
Draco
Fred
Ginny
Hermione
Harry
Hagrid
James
Kingsley
Luna
Minerva
Neville
Petunia
Ron
Severus
Vernon
Zacharias


Albus
Cedric
Draco
Fred
Ginny
George
Hermione
Harry
Hagrid
James
Kingsley
Luna
Lily
Lucius
Minerva
Neville
Petunia
Ron
Remus
Severus
Sirius
Vernon
Zacharias


Hermione


Hermione
Harry
Hagrid

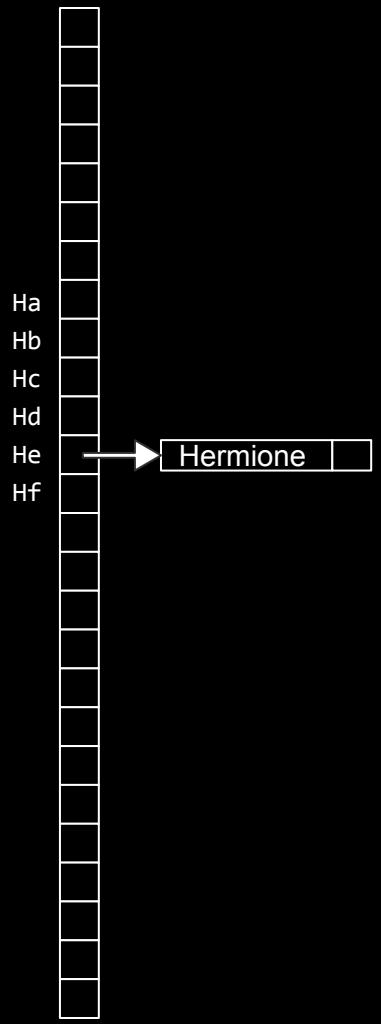

Hermione
Harry
Hagrid

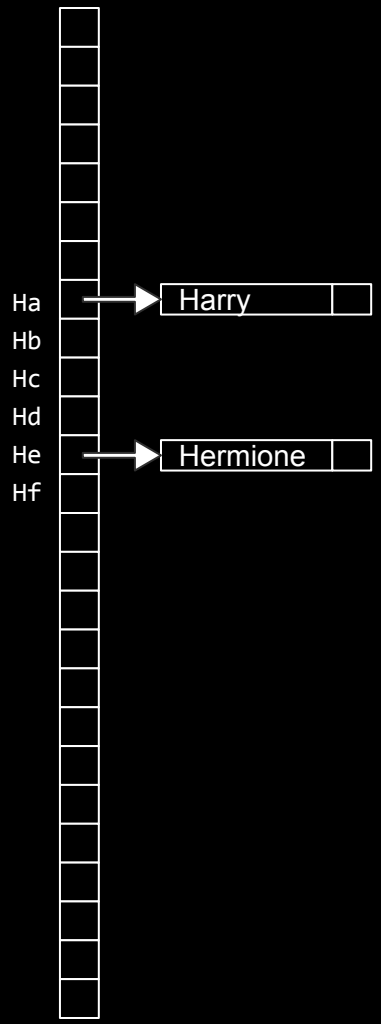

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
Hermione
Harry
Hagrid

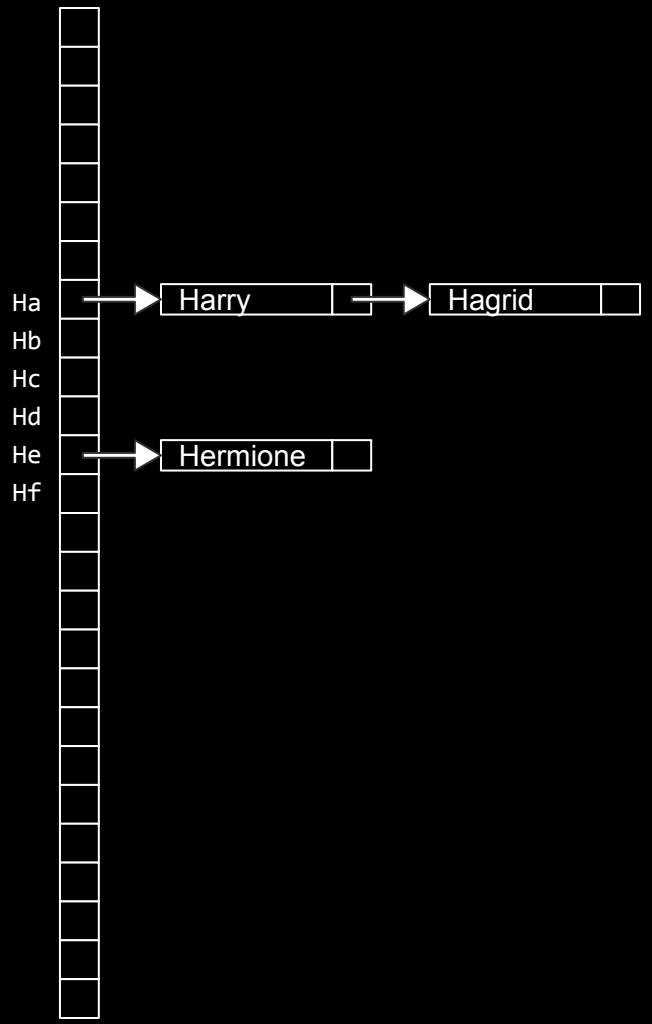
H

Ha

Ha
Hb
Hc
Hd
He
Hf


Ha
Hb
Hc
Hd
He
Hf
Hermione

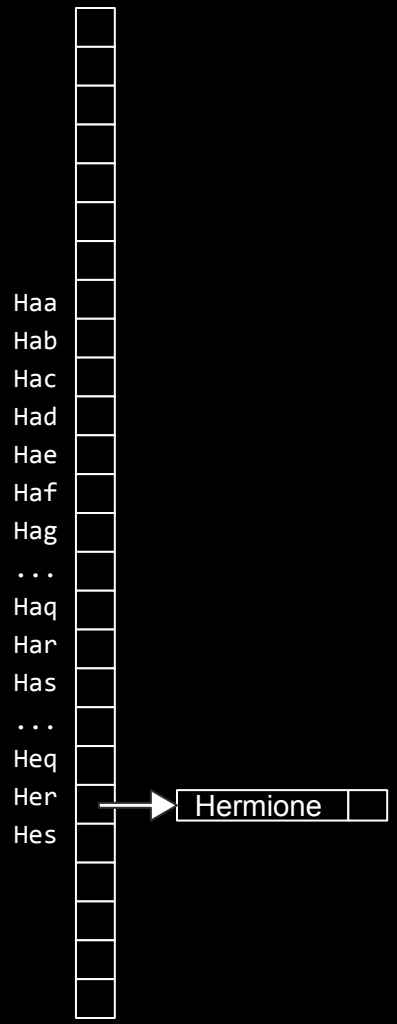

Ha
Hb
Hc
Hd
He
Hf
Harry
Hermione

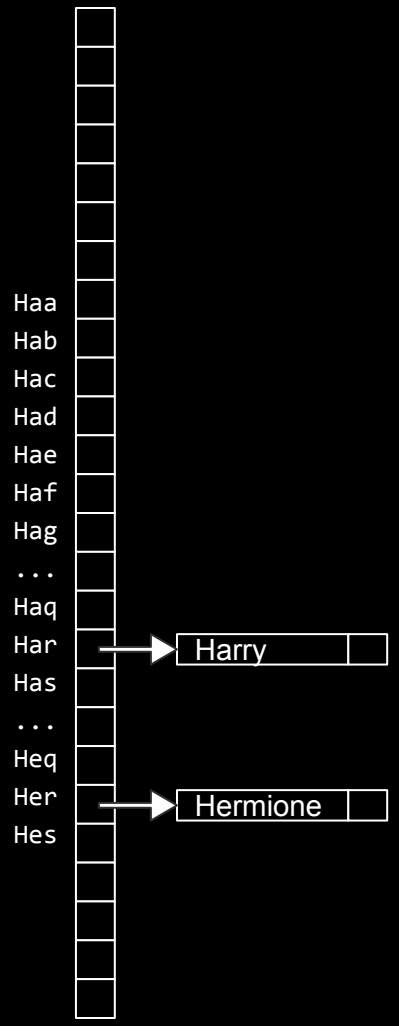

Ha
Hb
Hc
Hd
He
Hf
Harry
Hagrid
Hermione

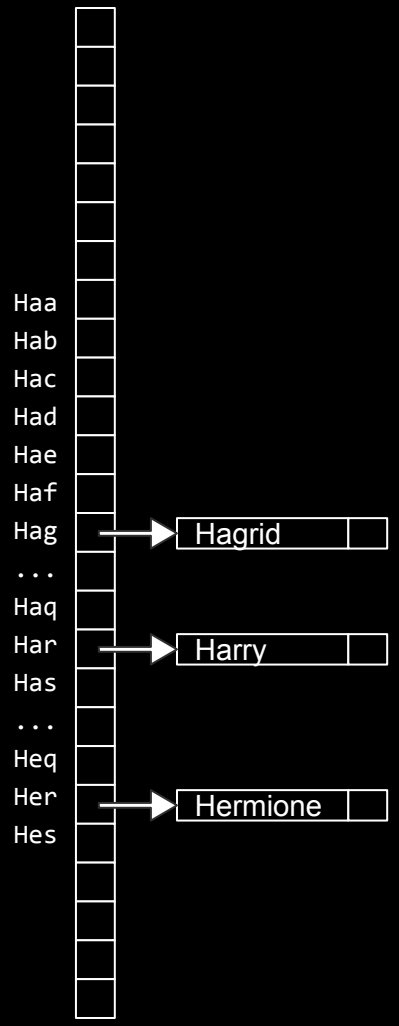
Ha

Haa

Haa
Hab
Hac
Had
Hae
Haf
Hag
...
Haq
Har
Has
...
Heq
Her
Hes


Haa
Hab
Hac
Had
Hae
Haf
Hag
...
Haq
Har
Has
...
Heq
Her
Hes
Hermione


Haa
Hab
Hac
Had
Hae
Haf
Hag
...
Haq
Har
Has
...
Heq
Her
Hes
Harry
Hermione


Haa
Hab
Hac
Had
Hae
Haf
Hag
...
Haq
Har
Has
...
Heq
Her
Hes
Hagrid
Harry
Hermione

$O(n^2)$

$O(n \log n)$

$O(n)$

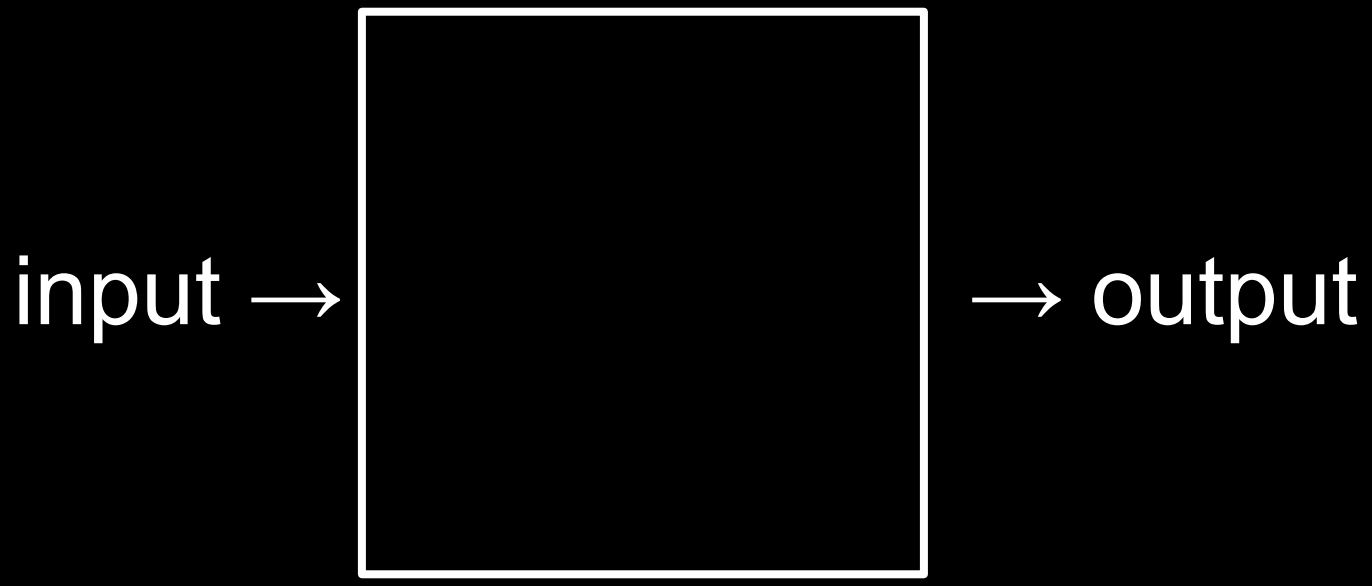
$O(\log n)$

$O(1)$

```
typedef struct
{
    char *name;
    char *number;
}
person;
```

```
typedef struct node
{
    char *name;
    char *number;
    struct node *next;
}
node;
```

```
node *table[26];
```


input →
→ output

hash function

Albus → [redacted] → 0

Zacharias → [redacted] → 25

$O(n)$

$O(n/k)$

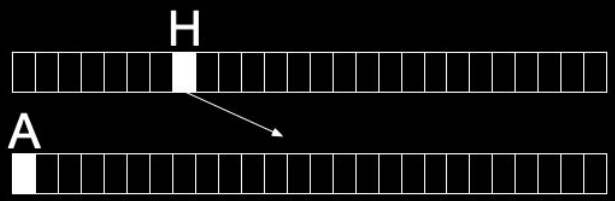
$O(n)$

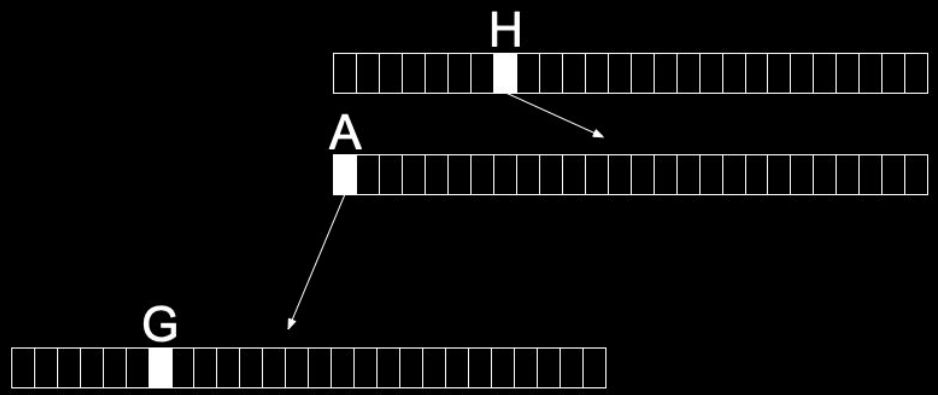
$O(1)$

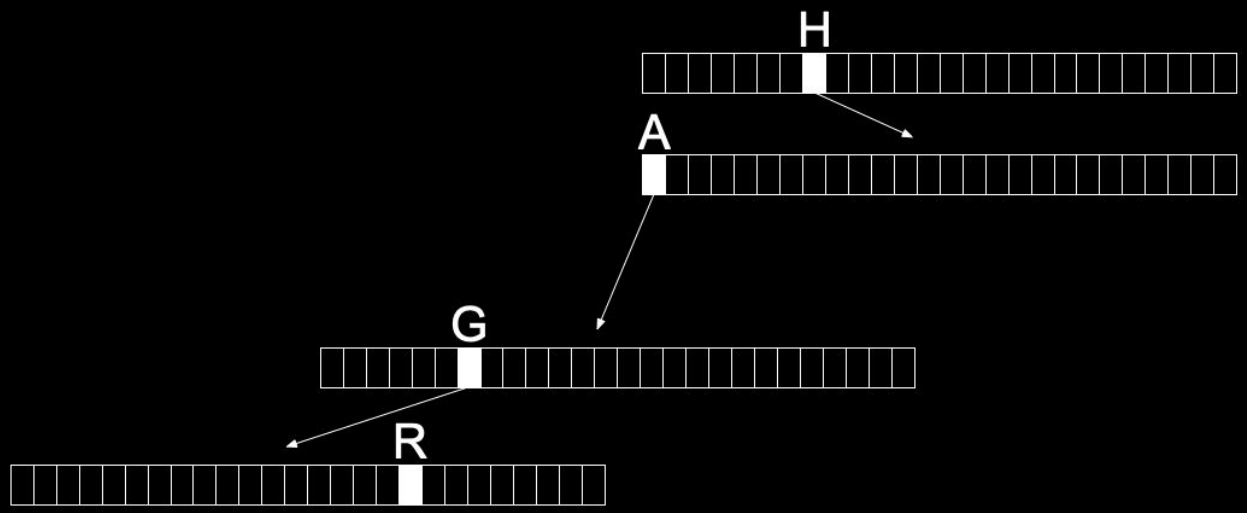
tries

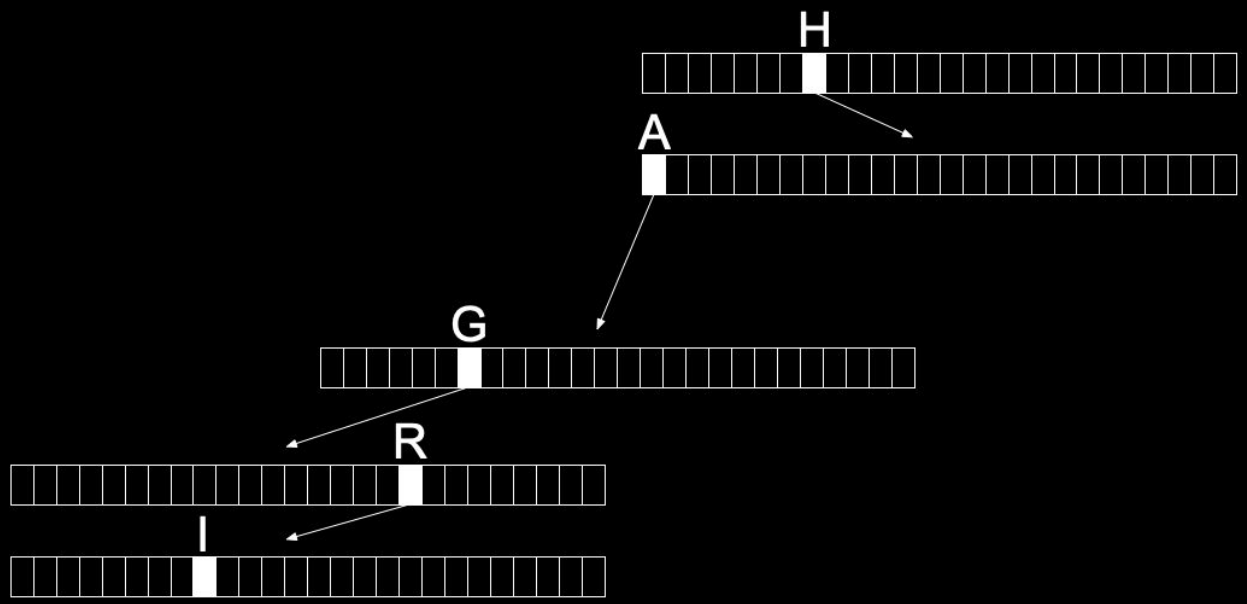
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

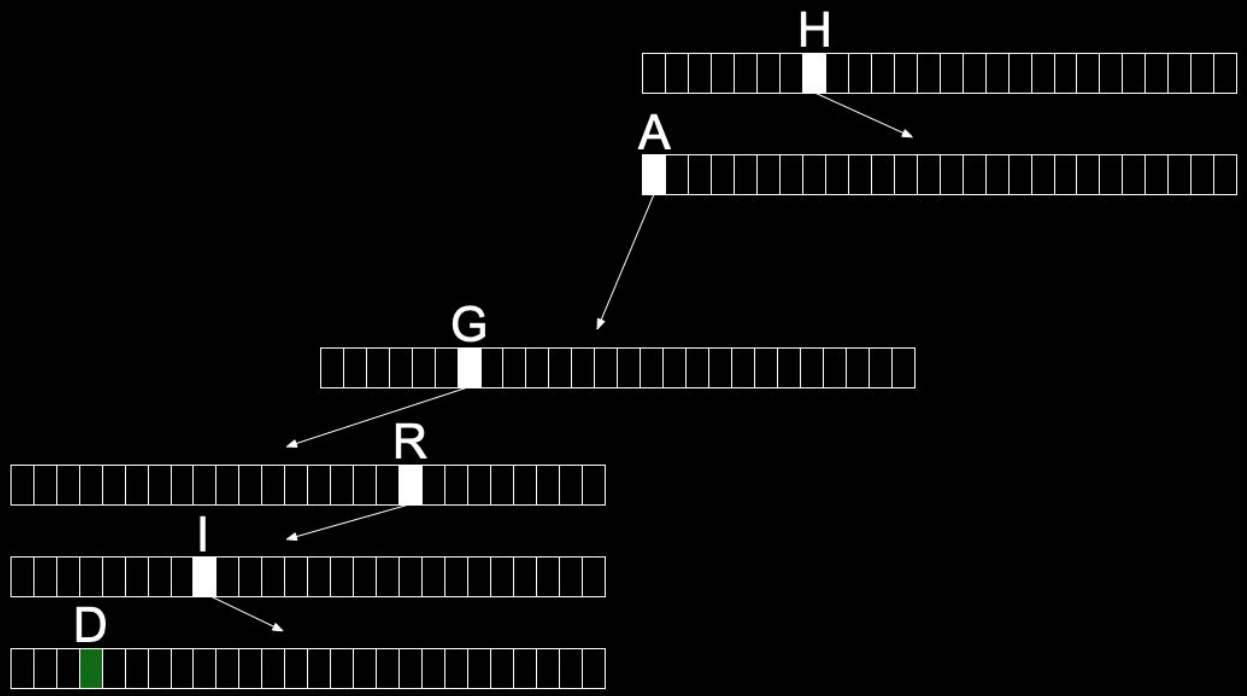

H

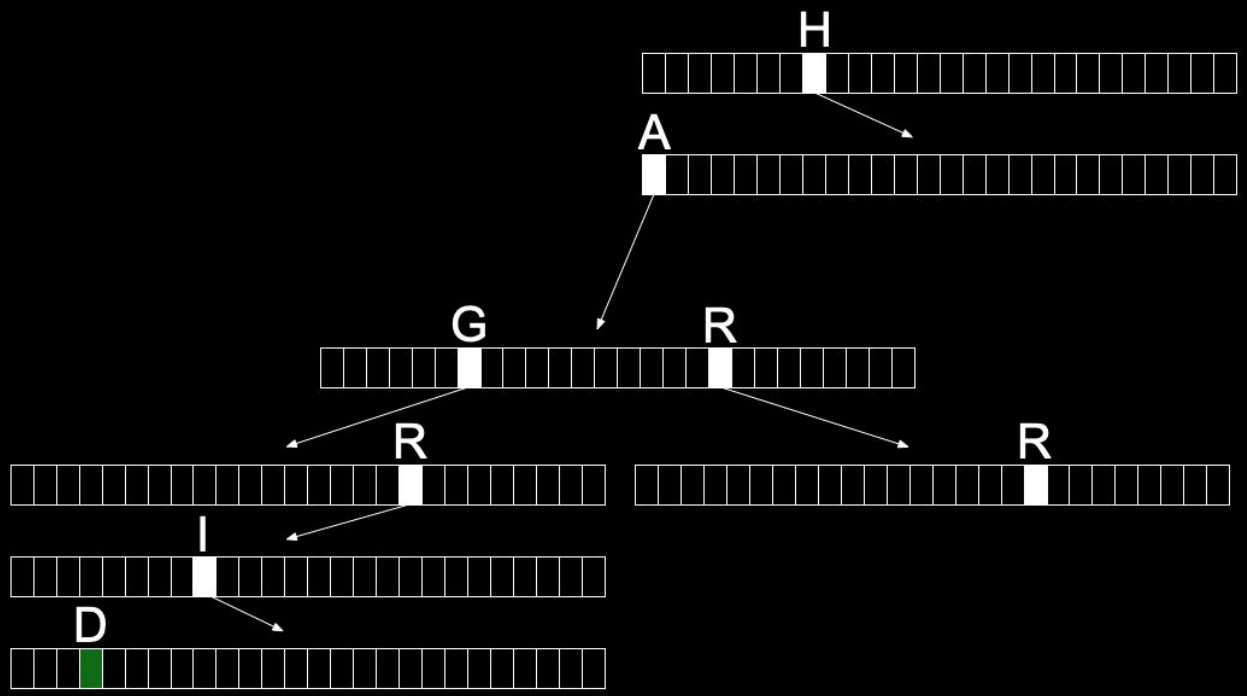

H
A

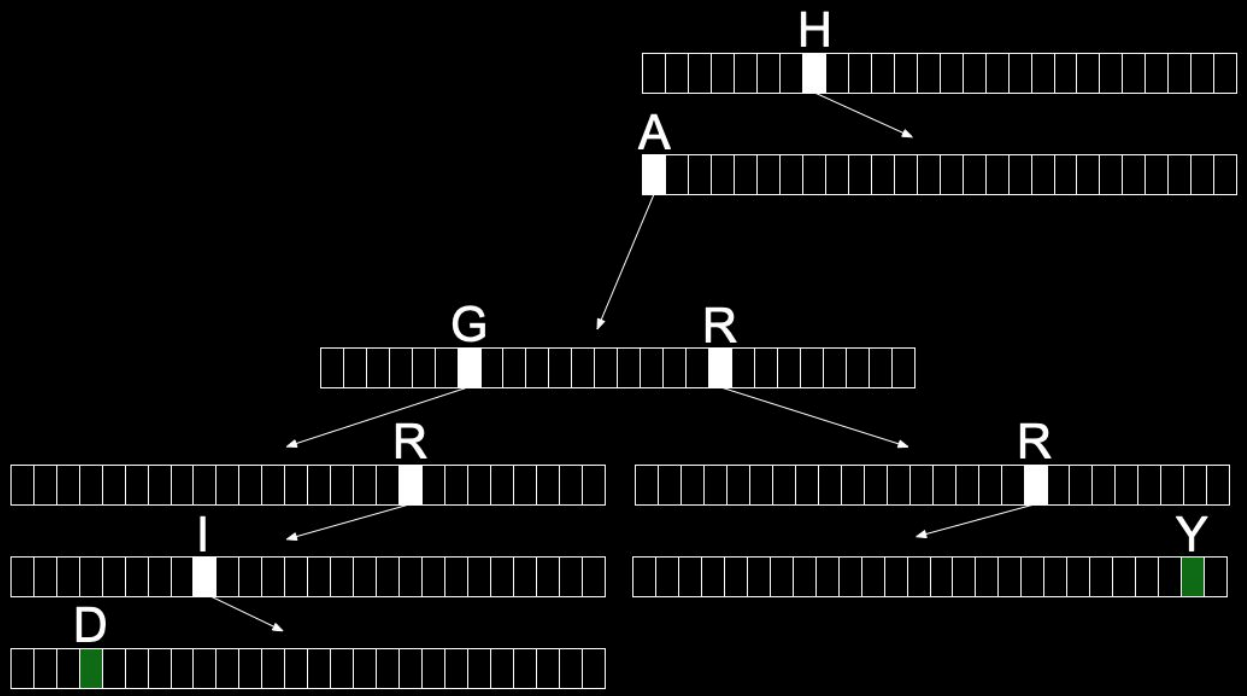

H
A
G

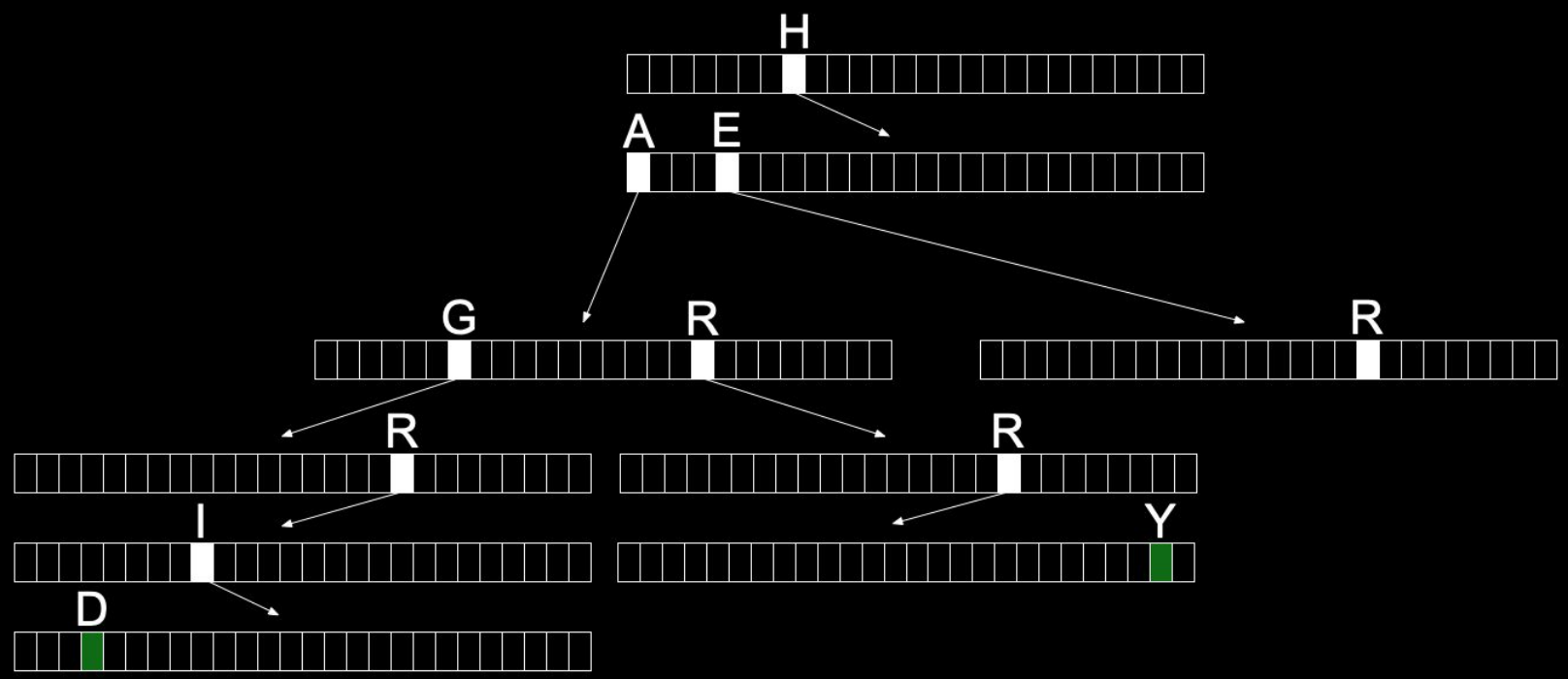

H
A
G
R

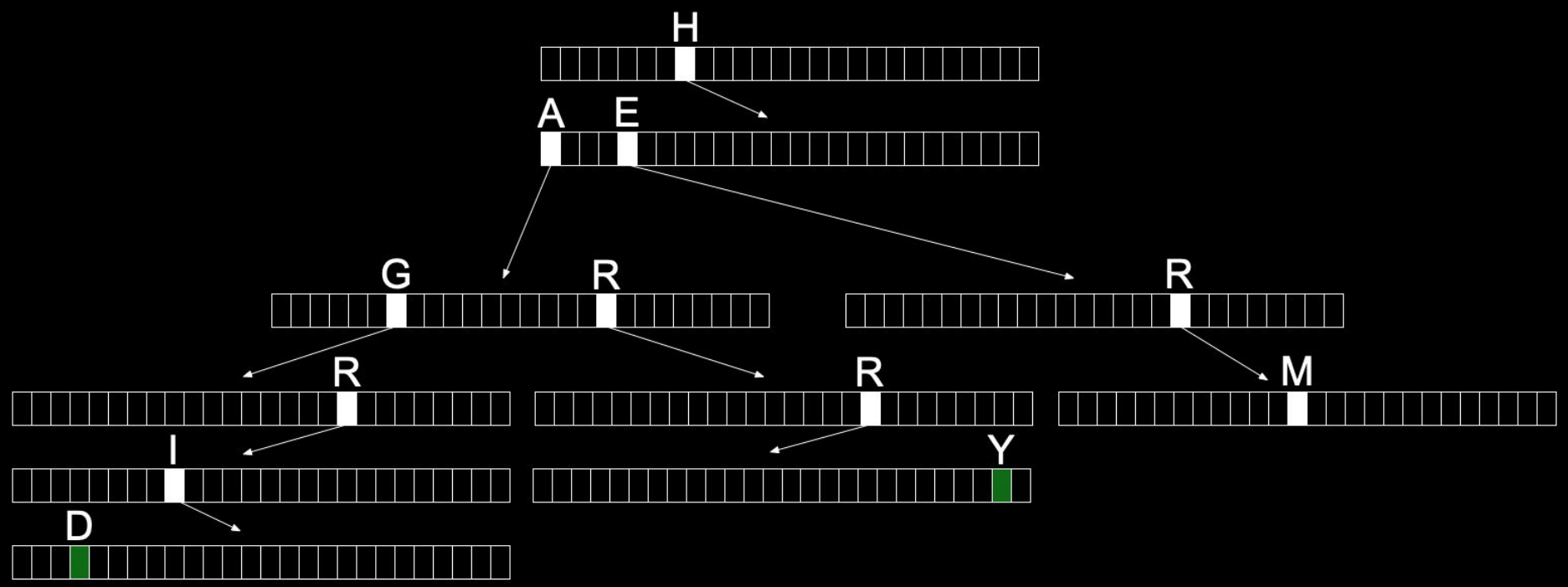

H
A
G
R
I

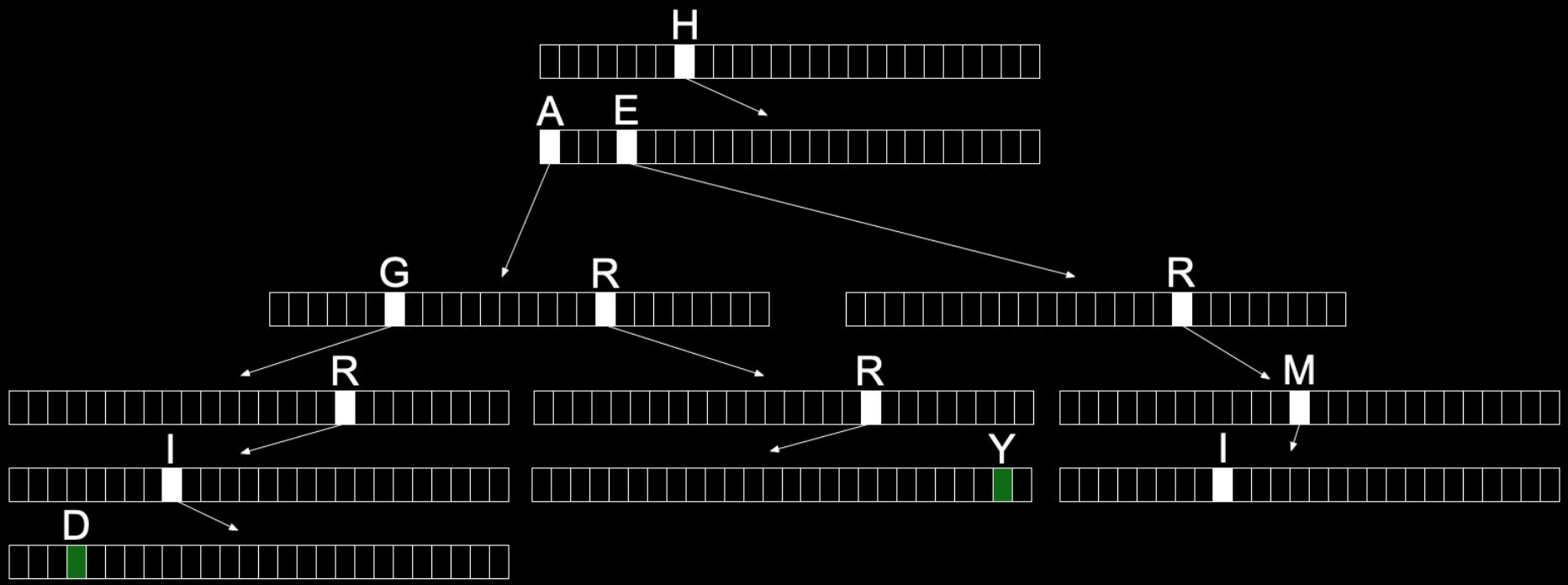

H
A
G
R
I
D

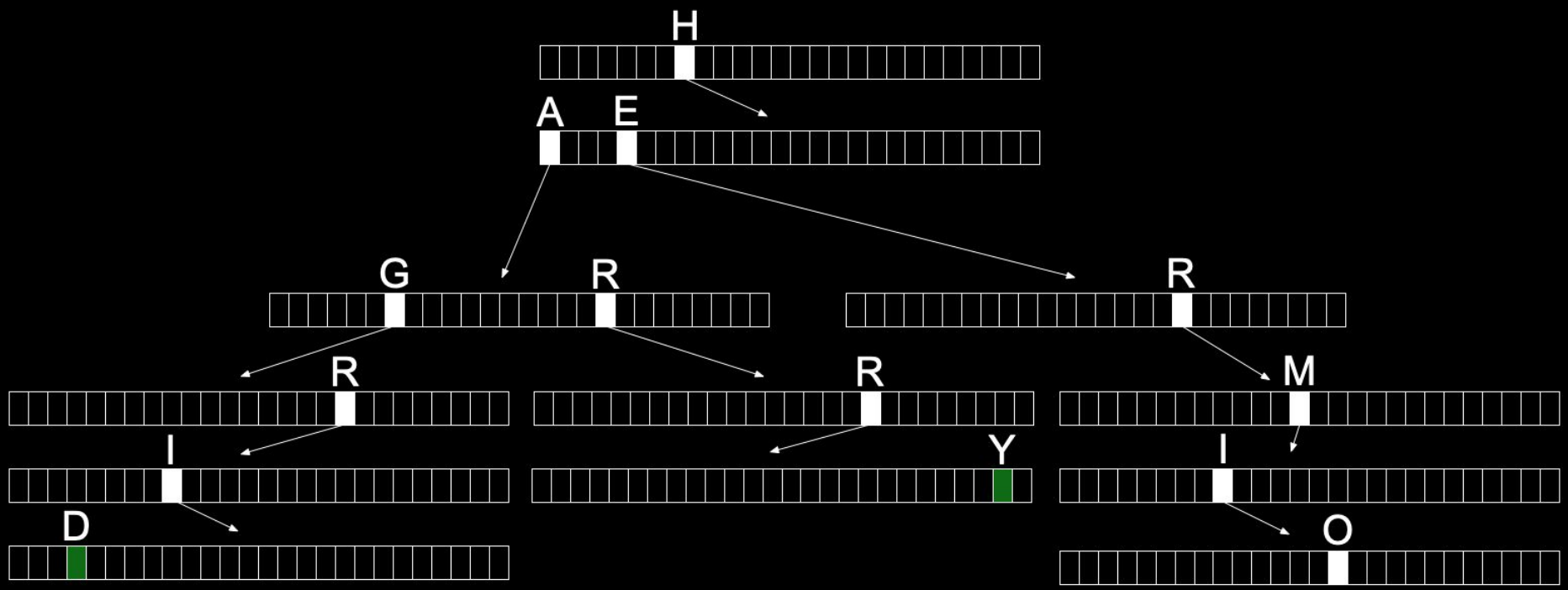

H
A
G
R
R
R
I
D

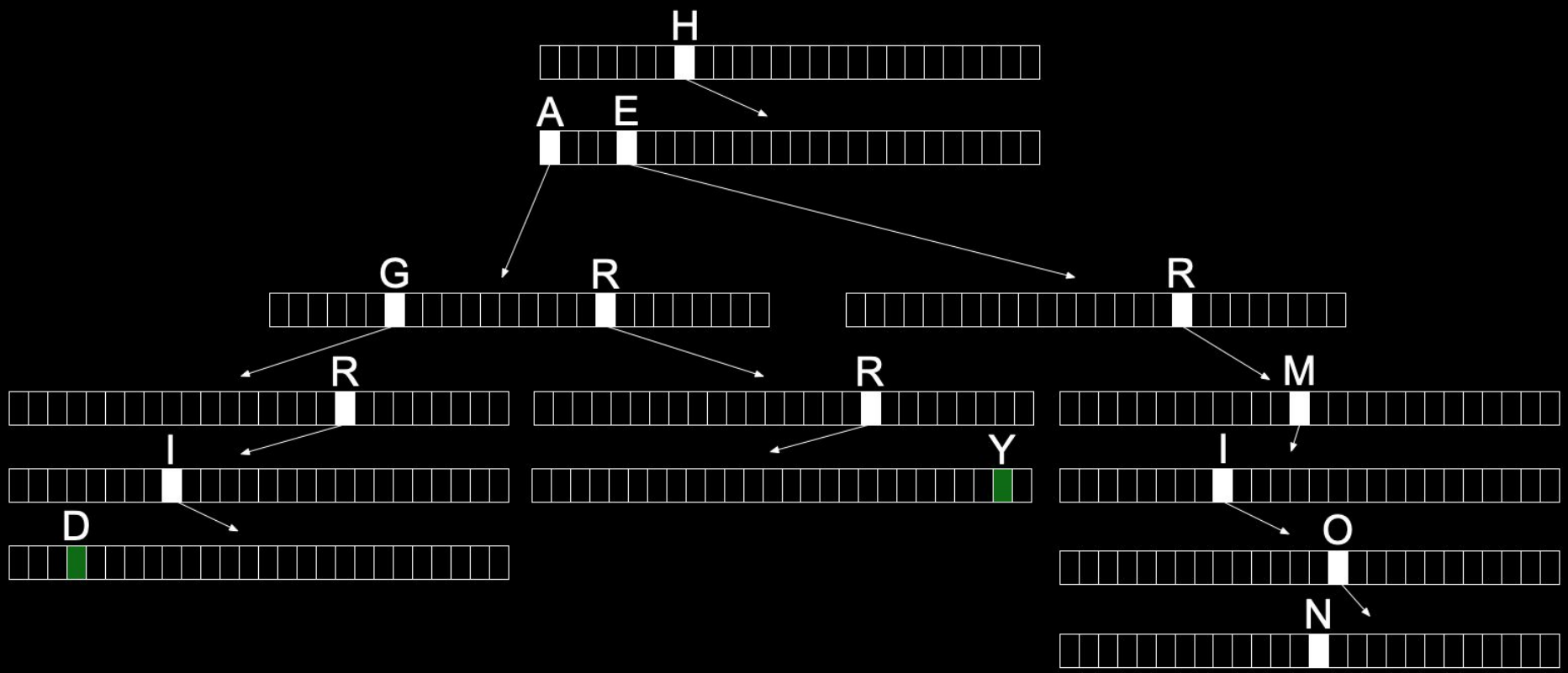

H
A
G
R
R
R
I
Y
D

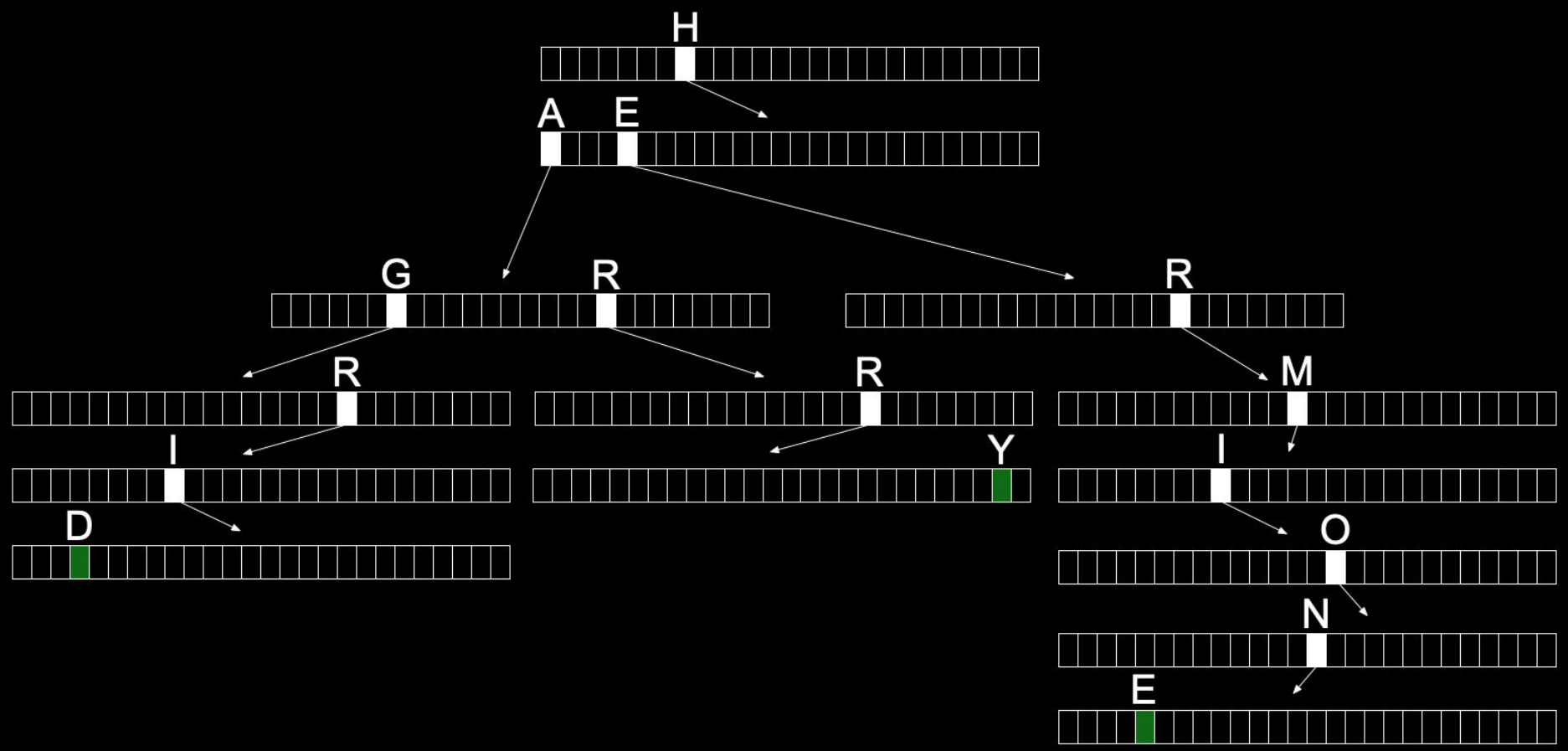

H
A
E
G
R
R
R
R
I
Y
D


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


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

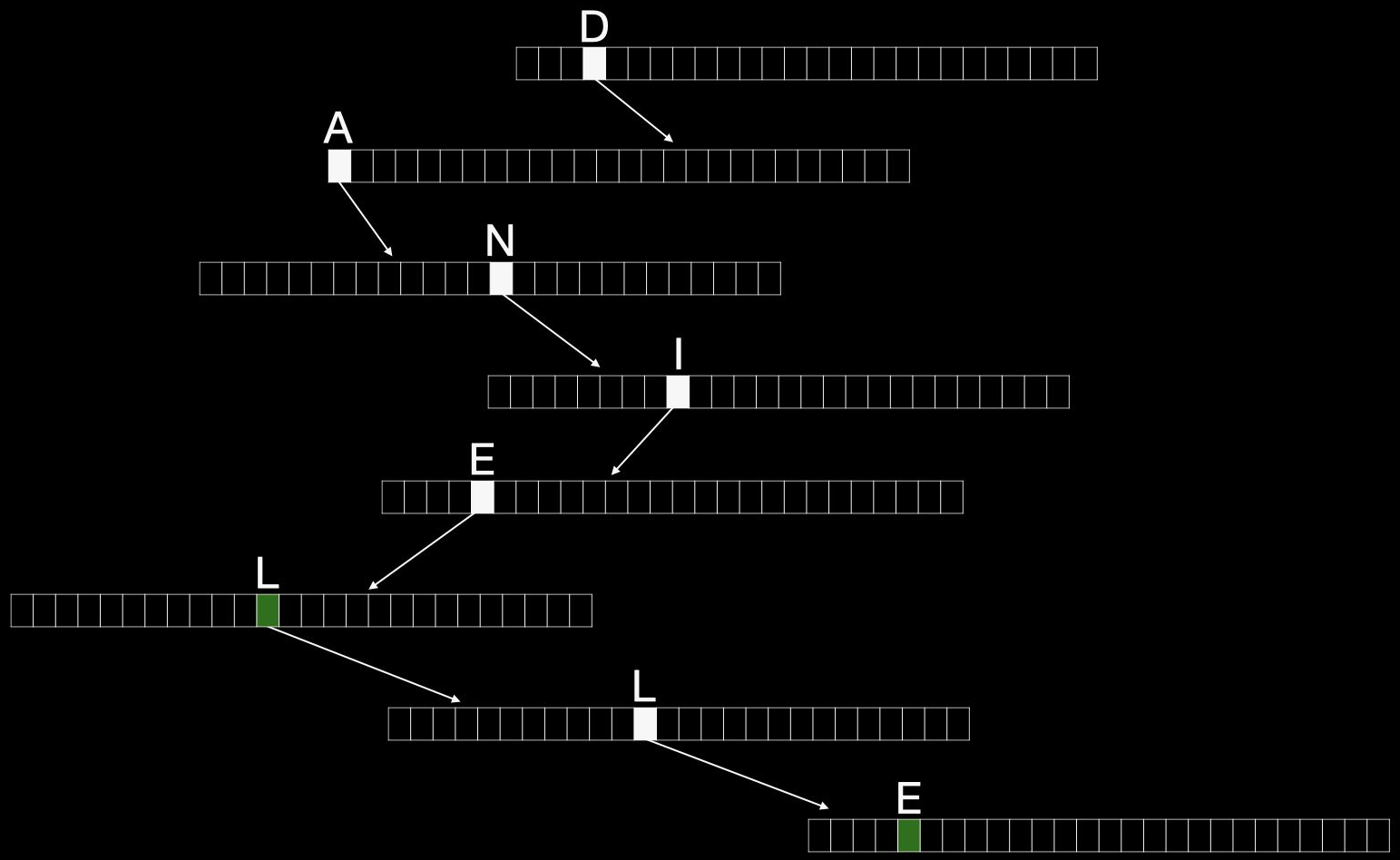

H
A
E
G
R
R
R
R
M
I
Y
I
D
O


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


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

```
typedef struct node
{
    char *number;
    struct node *children[26];
}
node;
```

```
node *trie;
```


D
A
N
I
E
L
L
E

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$


ICK ME UP
B C D E F G H I
K L M N O P Q R
T U V W X Y Z

This is CS50