

This is CS50

```
#include <stdio.h>
```

```
int main(void)
```

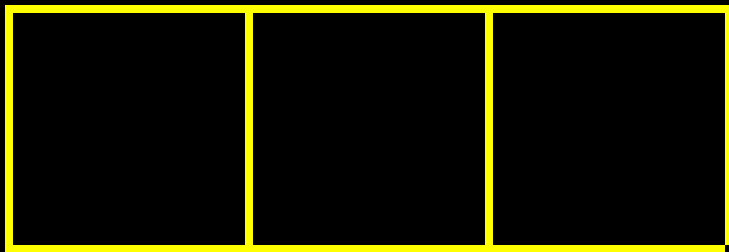
```
{
```

```
    printf("hello, world\n");
```

```
}
```

```
print("hello, world")
```

arrays

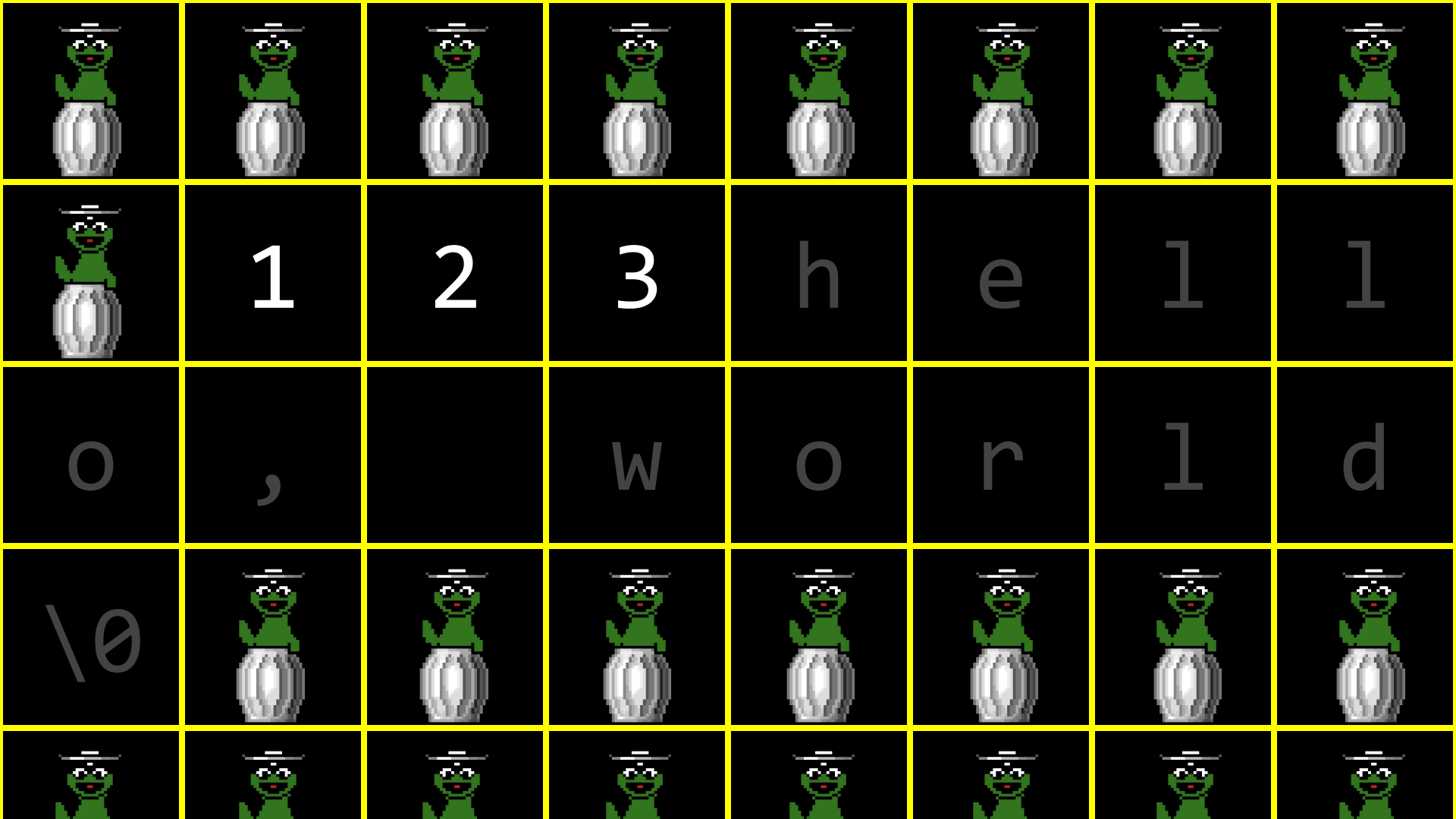


1	2	3
---	---	---

1	2	3	
---	---	---	--

	1	2	3				





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
---	---	---	---

$$O(n^2)$$

$$O(n \log n)$$

$$O(n)$$

$$O(\log n)$$

$$O(1)$$



$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$       search

$O(1)$

$O(n^2)$

$O(n \log n)$

$O(n)$       insert

$O(\log n)$       search

$O(1)$

data structures

struct

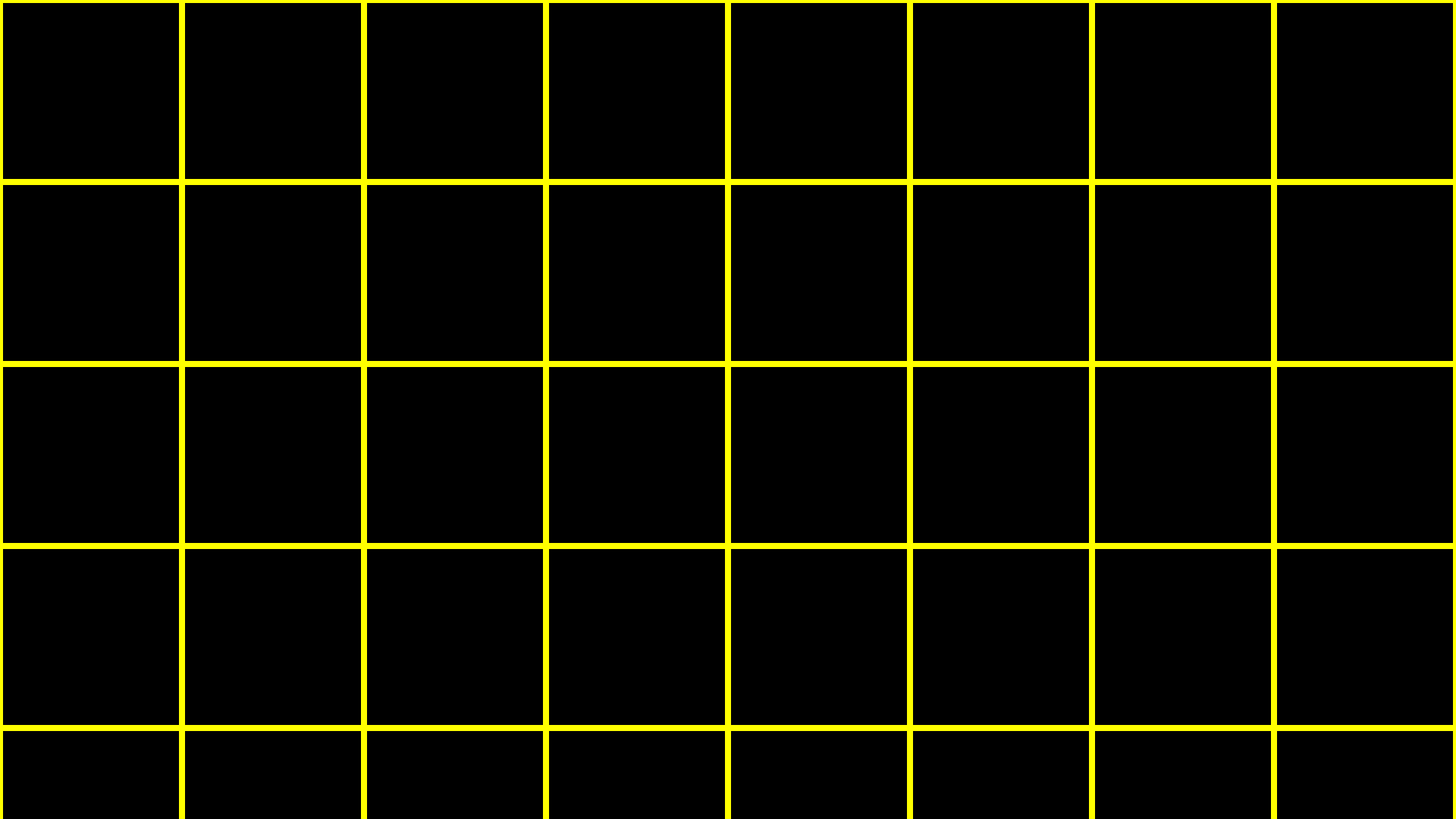
•

\*

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

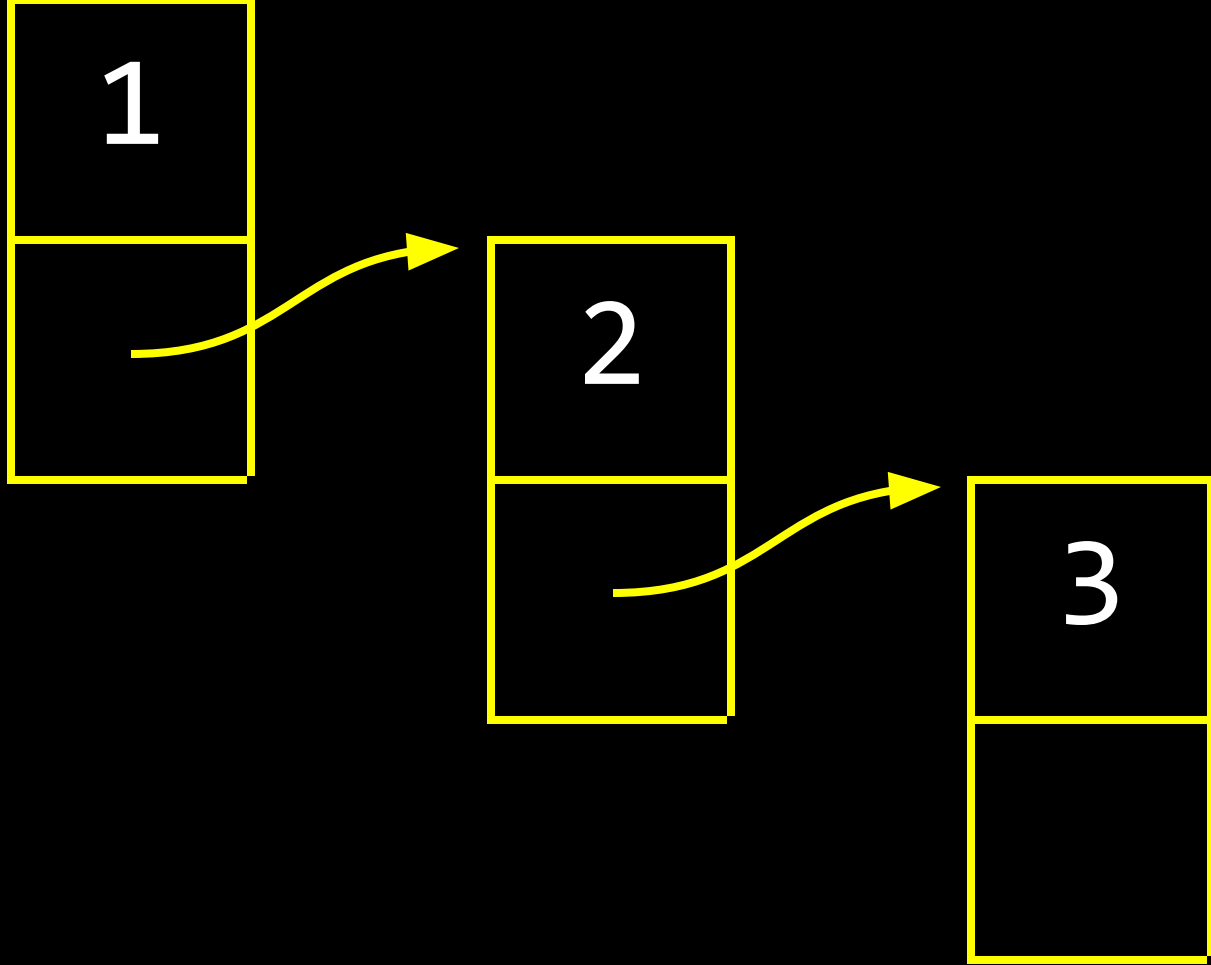
0x456

0x789

3

0x789

NULL





```
typedef struct
{
    string name;
    string 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;
```

```
node *list;
```



list



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

list

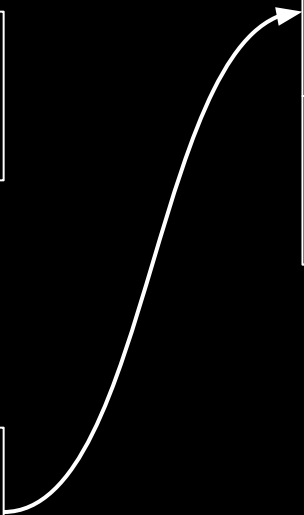


```
node *n = malloc(sizeof(node));
```

list



n



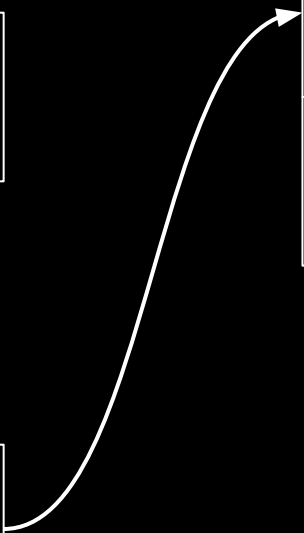
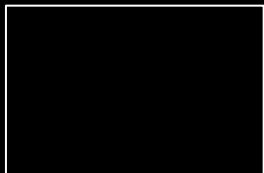
```
if (n != NULL)
{
    (*n).number = 1;
}
```

```
if (n != NULL)
{
    n->number = 1;
}
```

list



n



1



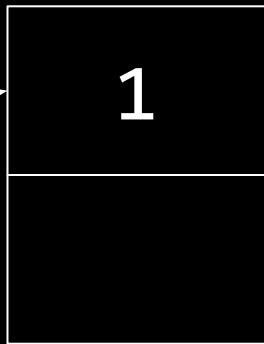


```
if (n != NULL)
{
    n->next = NULL;
}
```

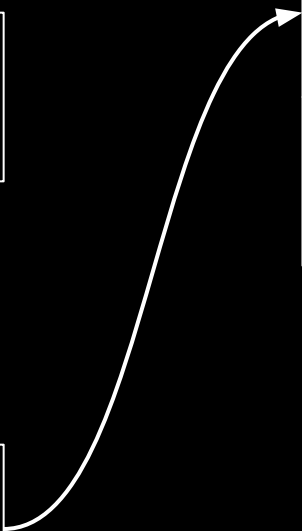
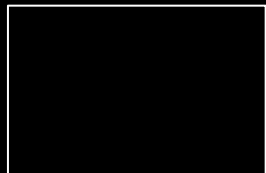
list



1



n

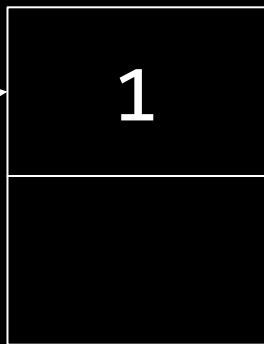


```
list = n;
```

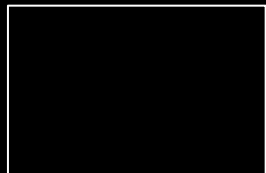
list



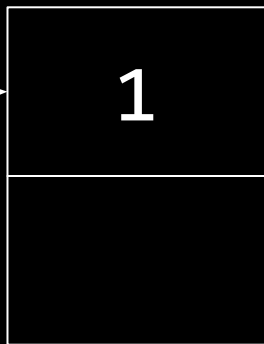
1



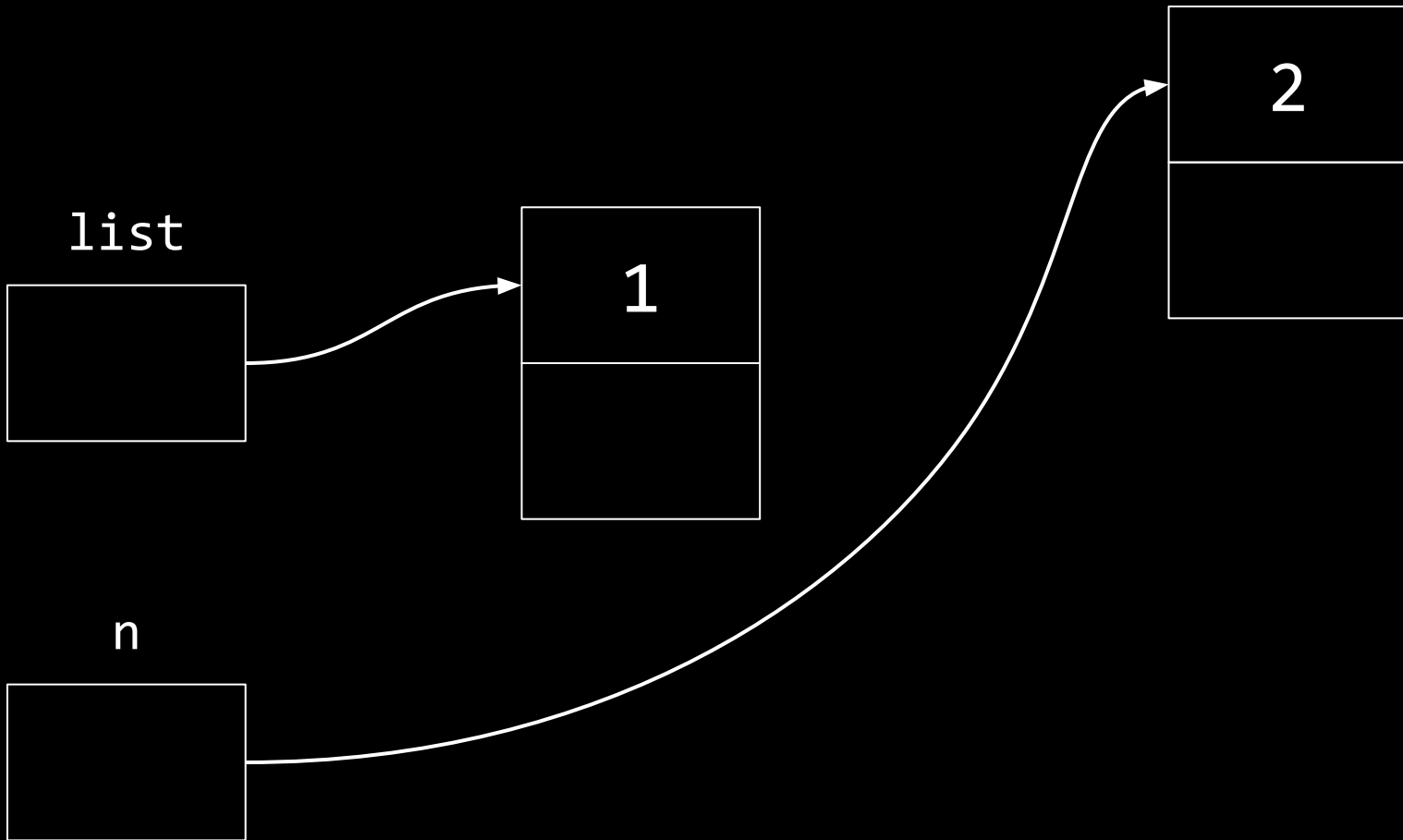
n



list

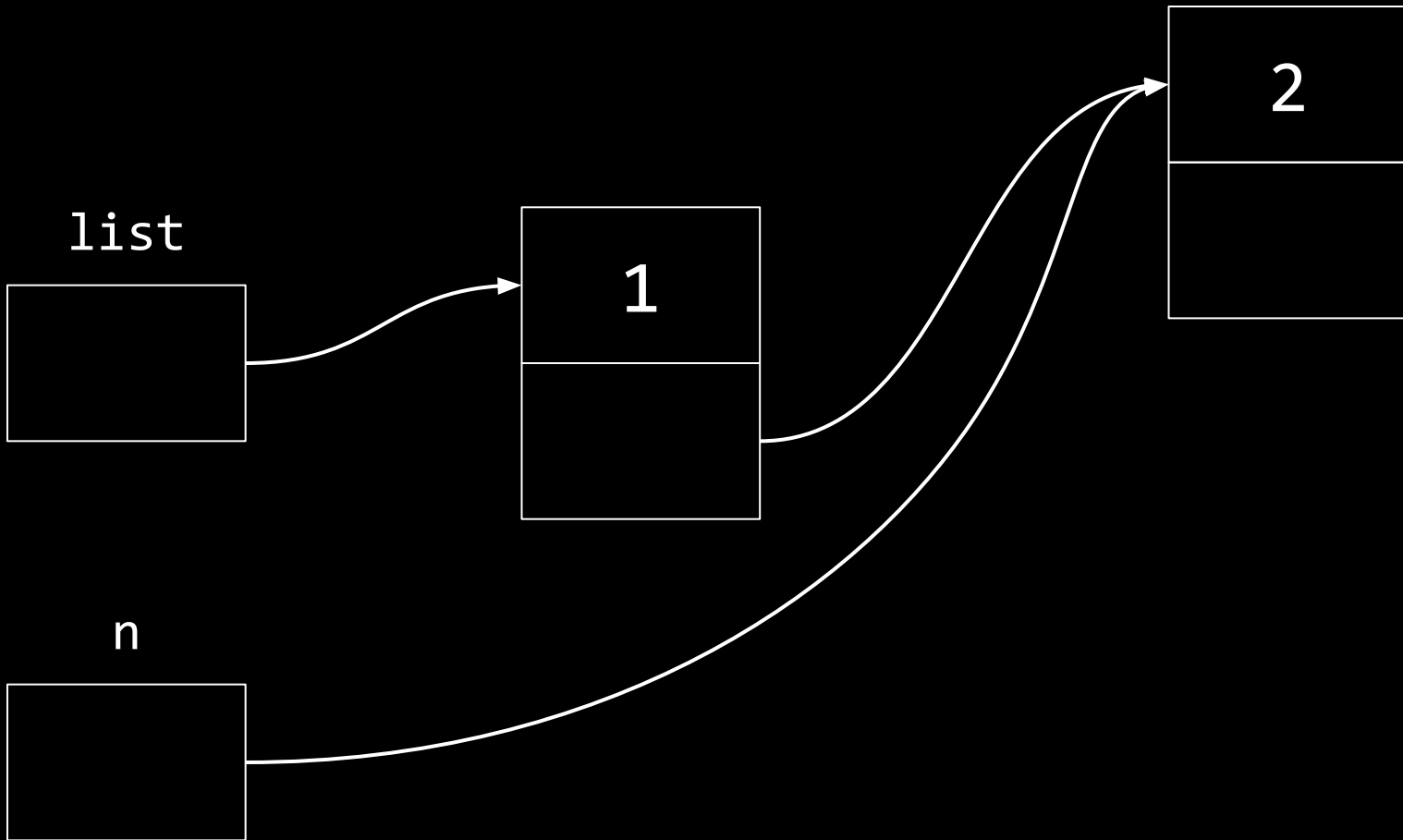


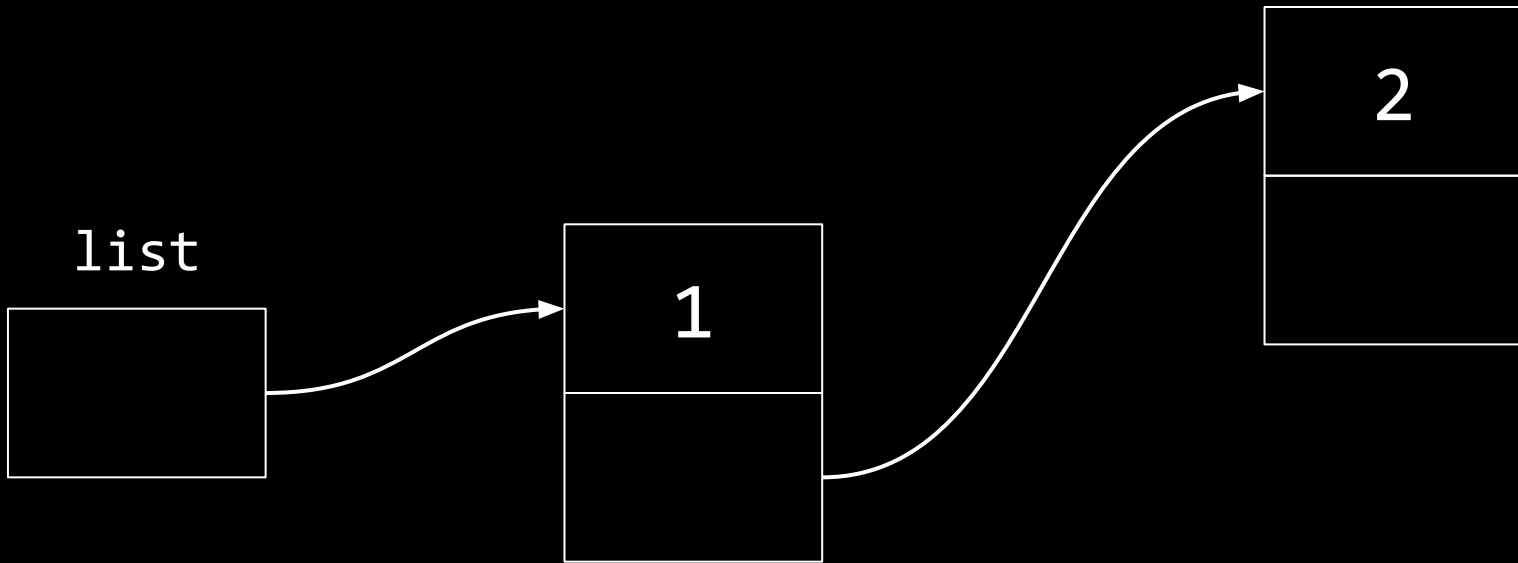
```
node *n = malloc(sizeof(node));  
if (n != NULL)  
{  
    n->number = 2;  
    n->next = NULL;  
}
```



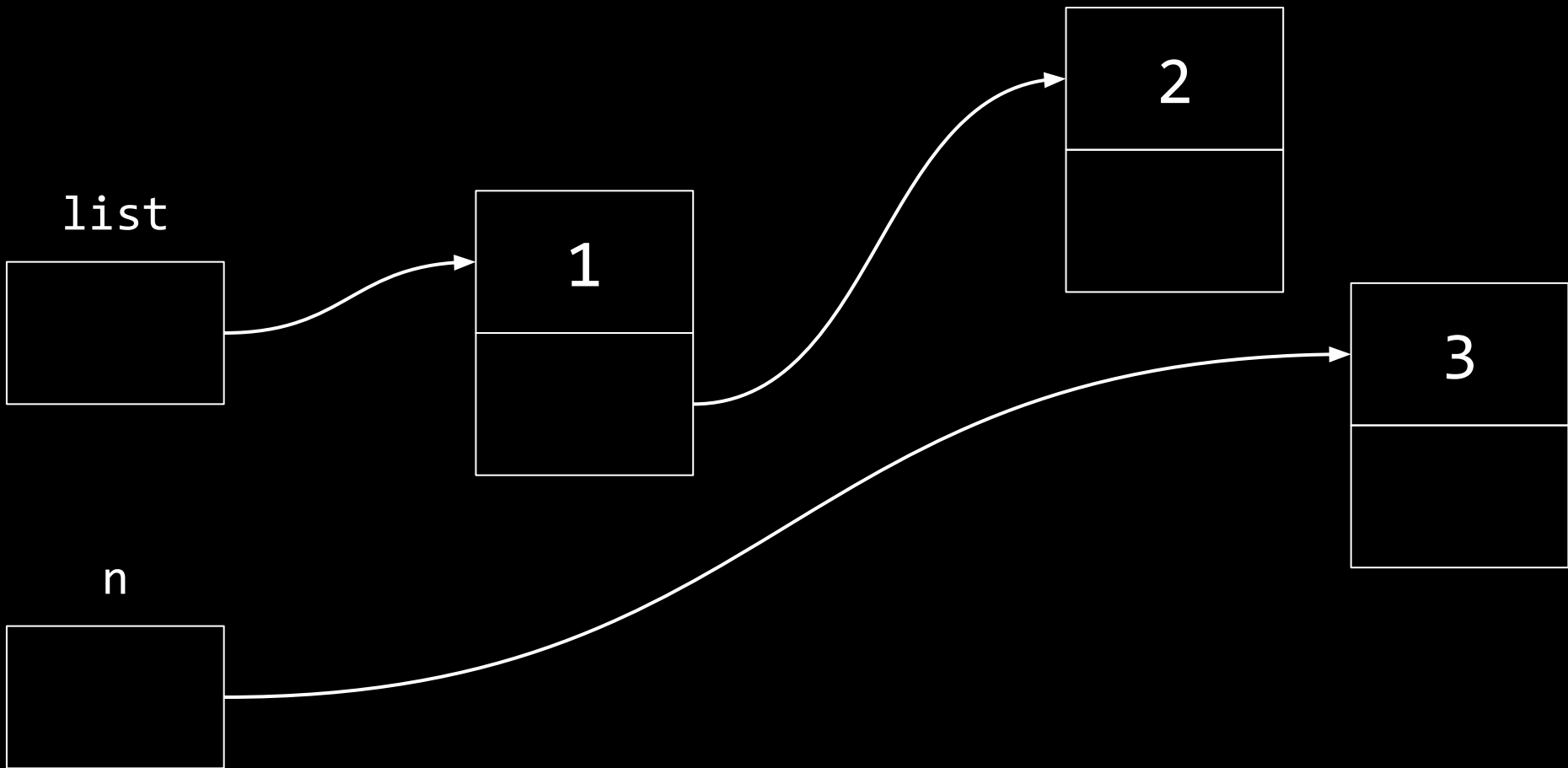
```
list->next = n;
```



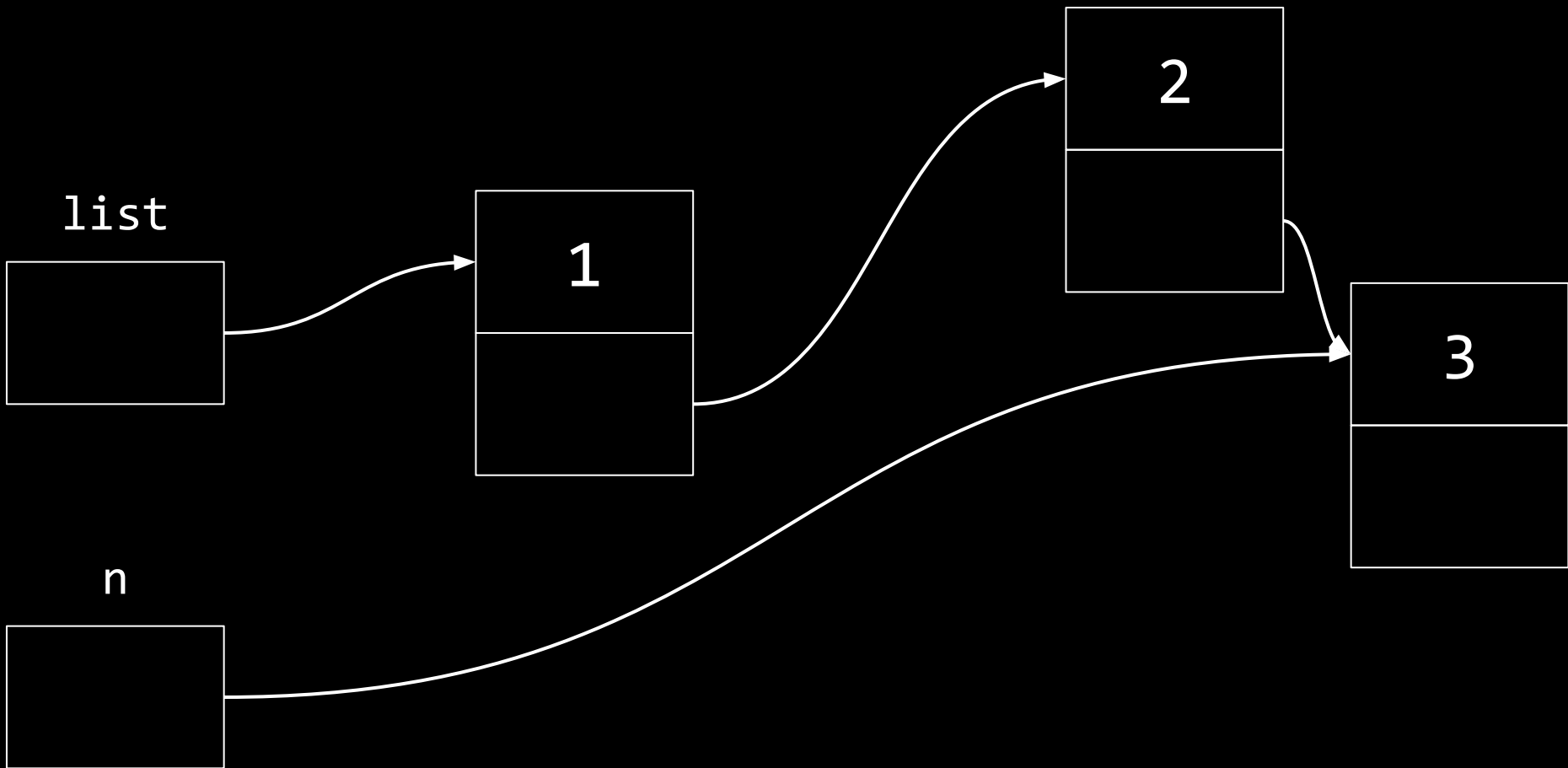


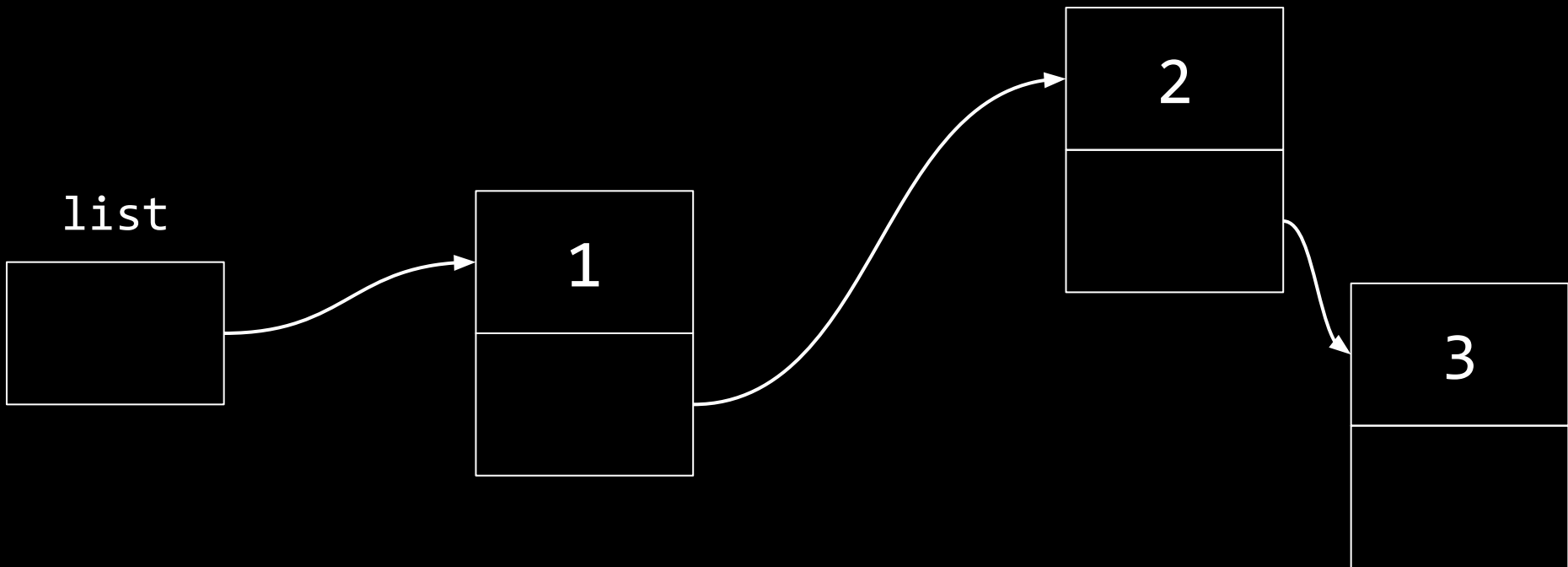


```
node *n = malloc(sizeof(node));  
if (n != NULL)  
{  
    n->number = 3;  
    n->next = NULL;  
}
```



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





list

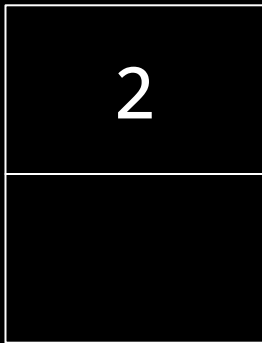




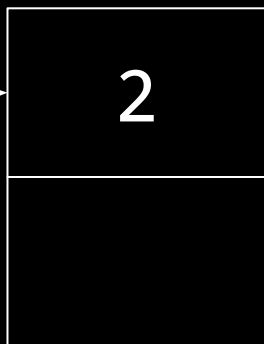
list



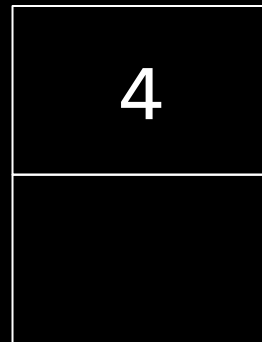
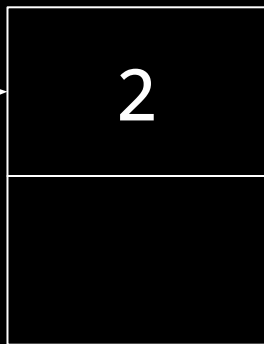
2

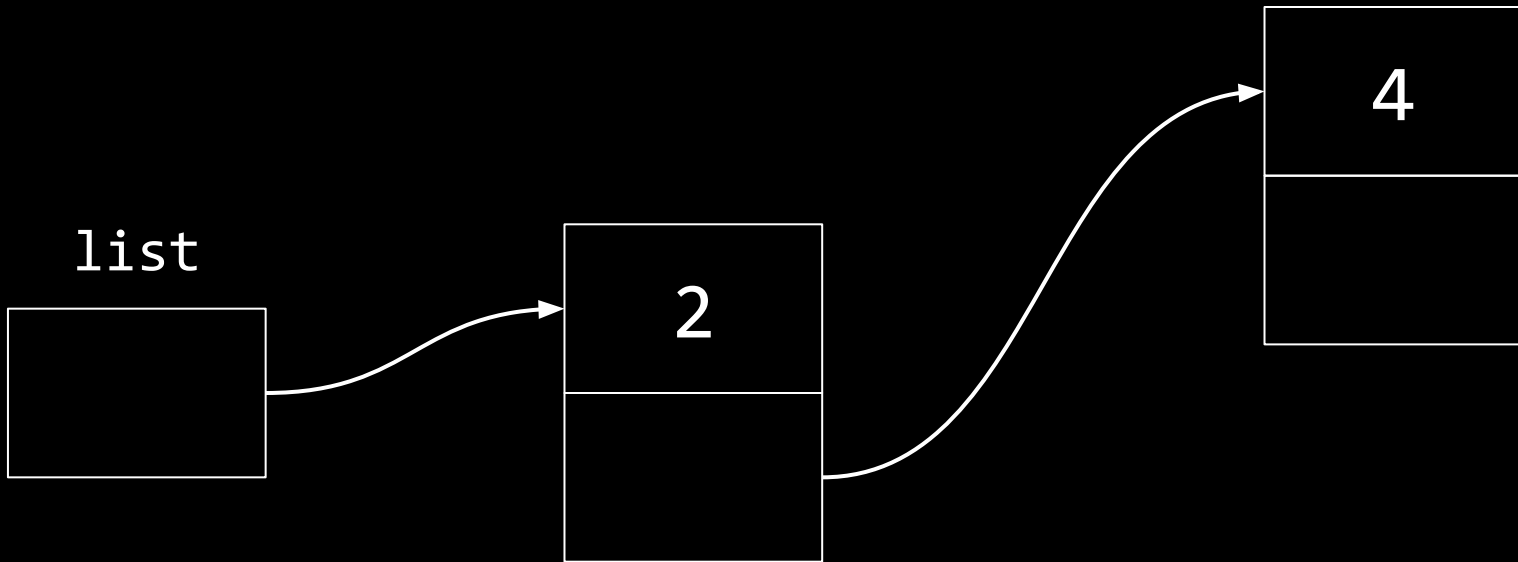


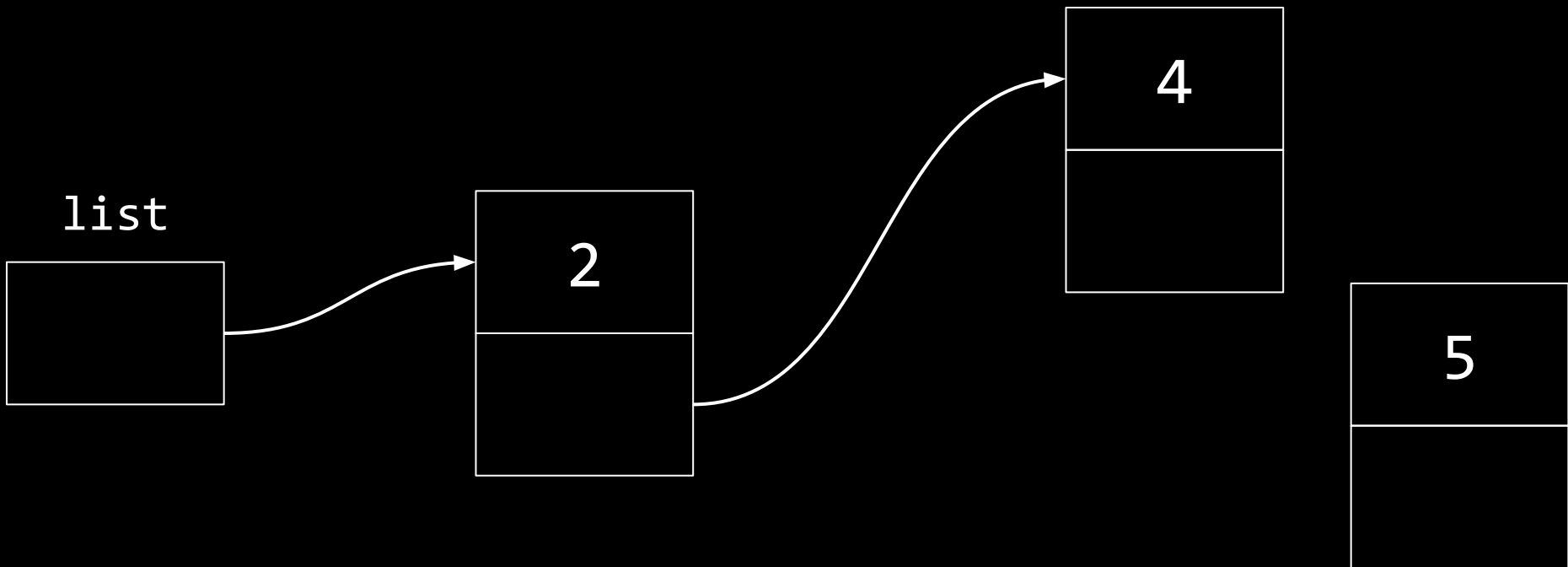
list

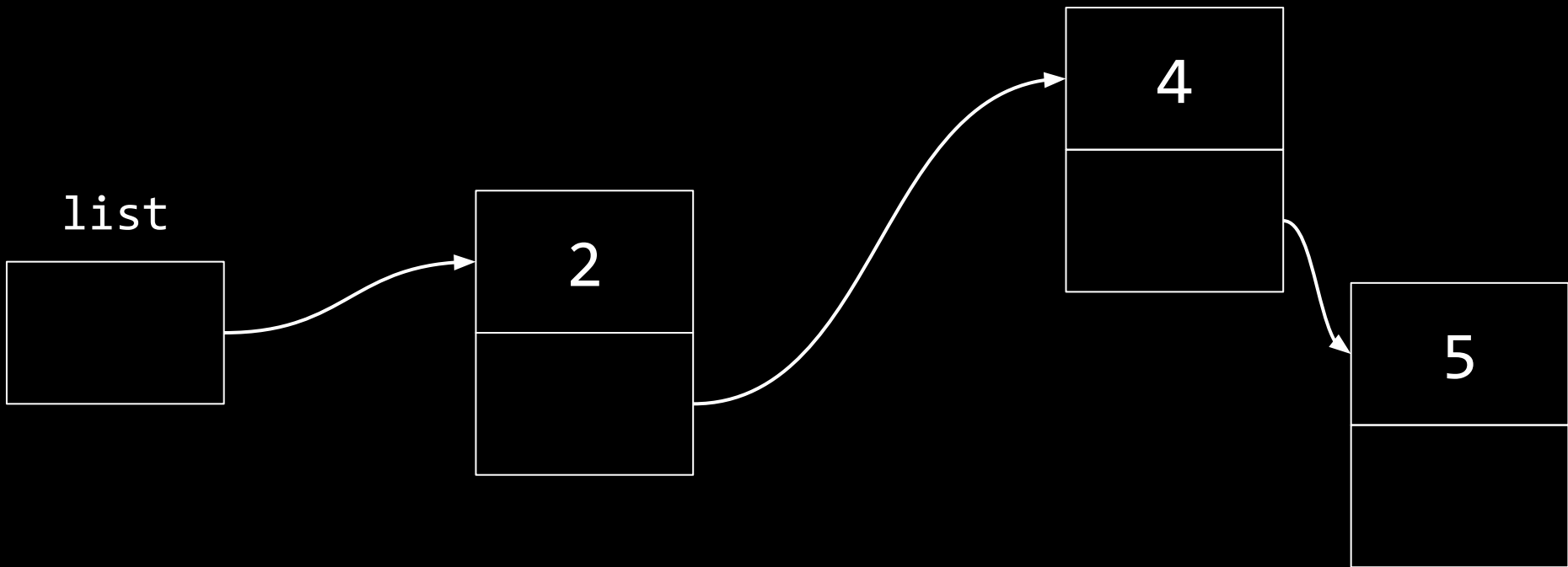


list

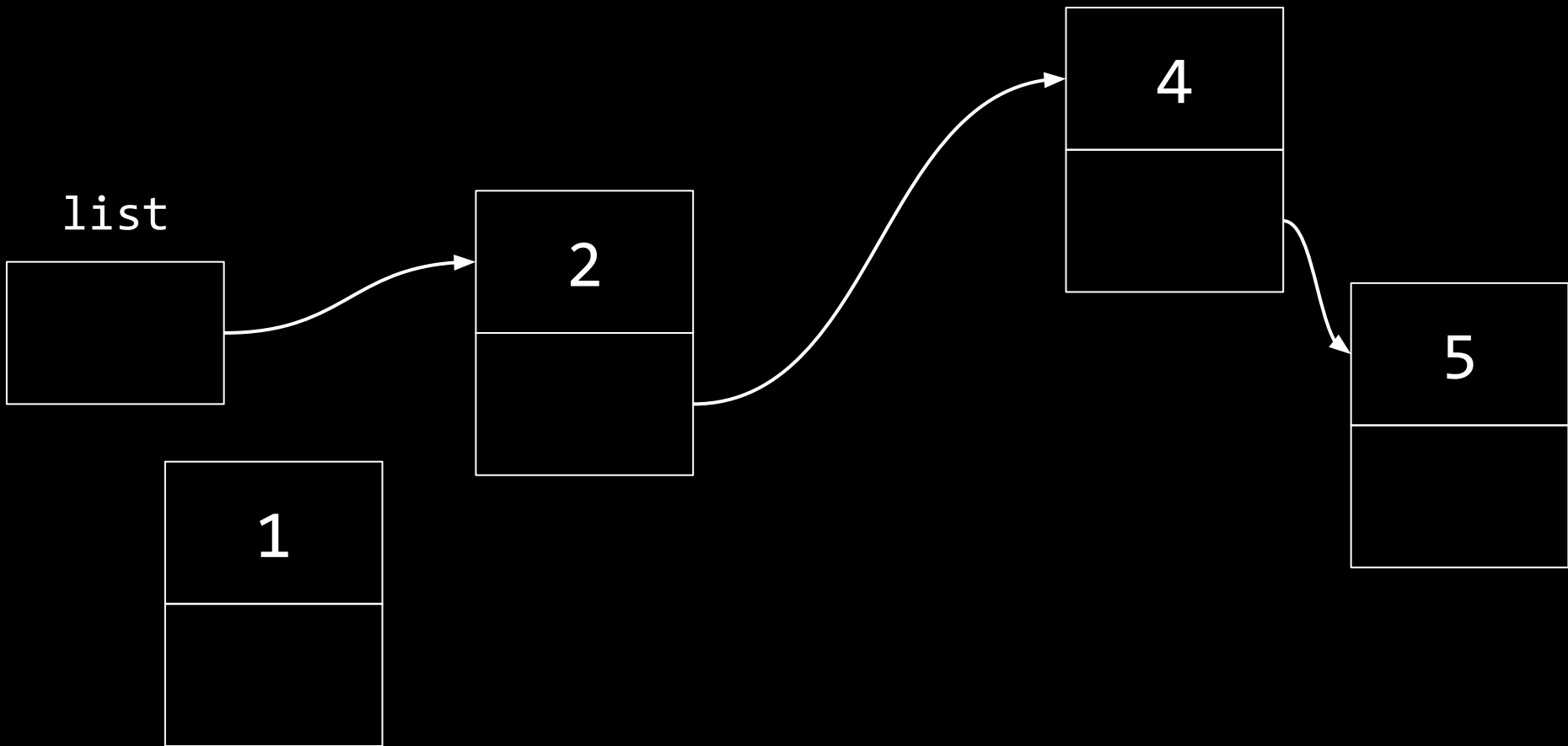








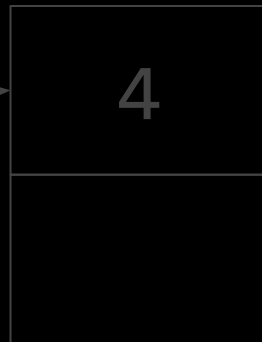
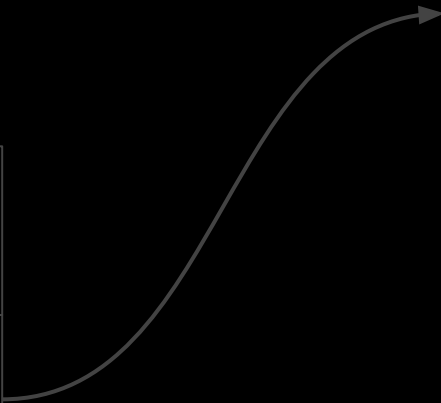
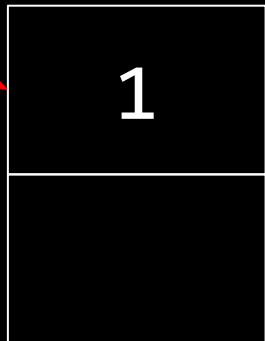
```
node *n = malloc(sizeof(node));  
if (n != NULL)  
{  
    n->number = 1;  
    n->next = NULL;  
}
```



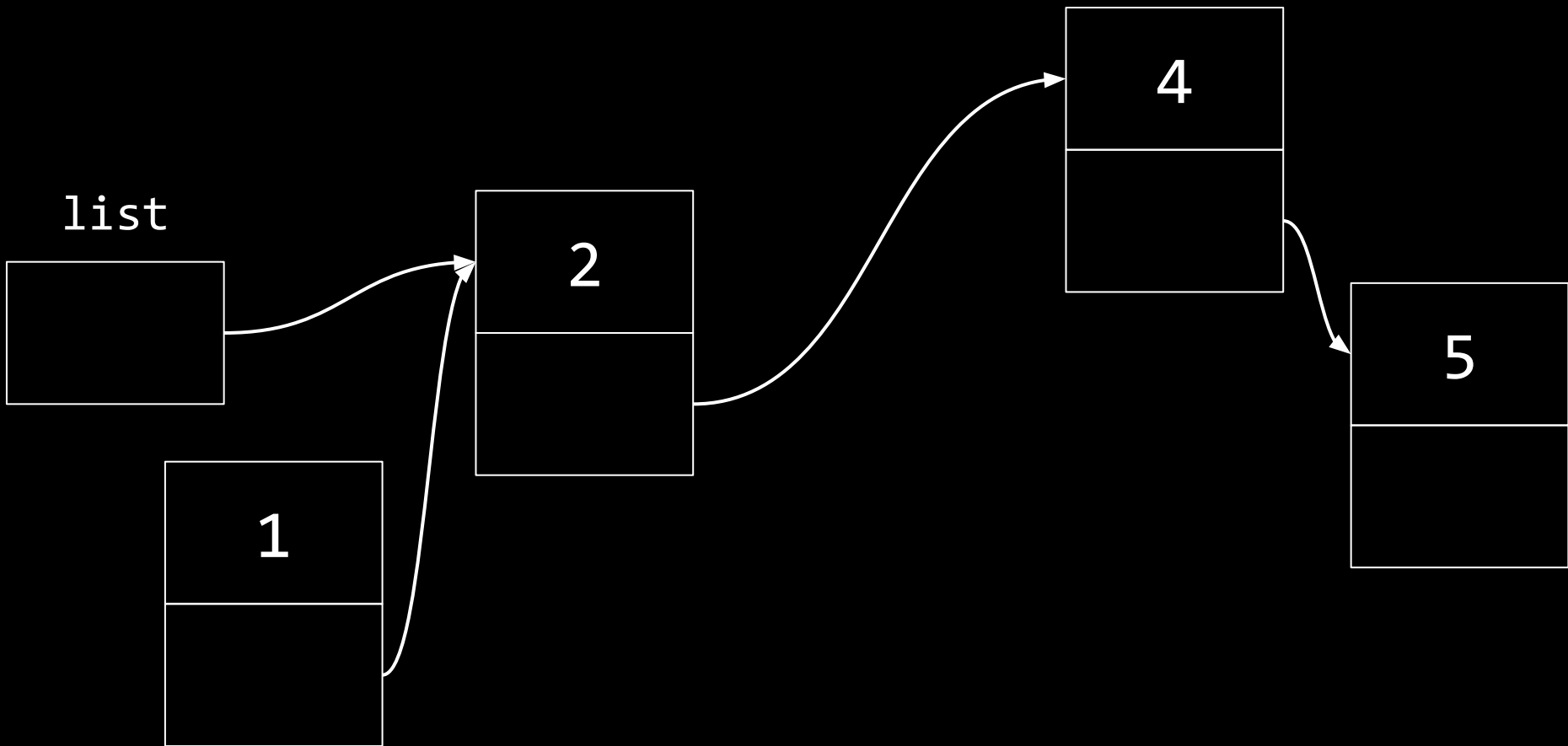




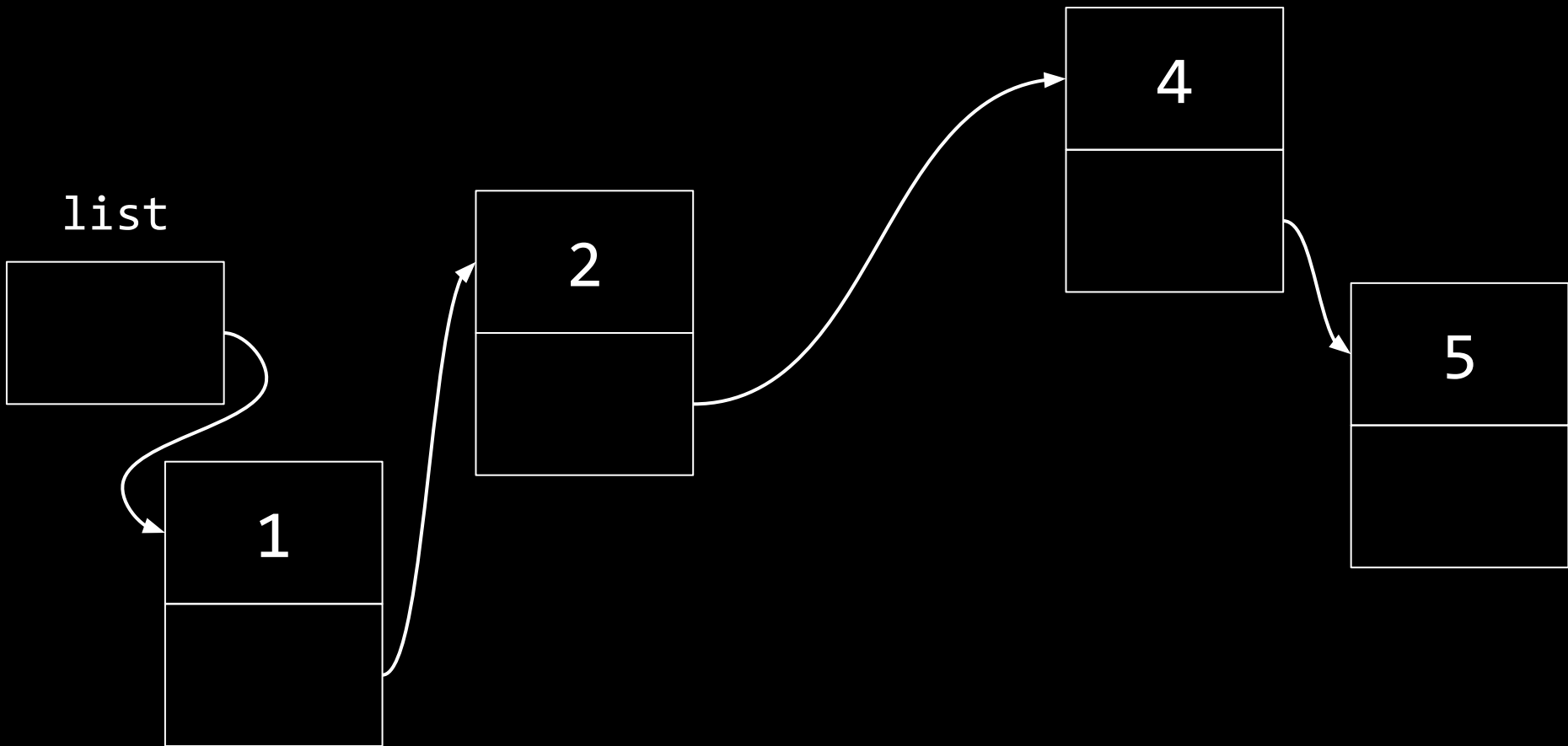
list

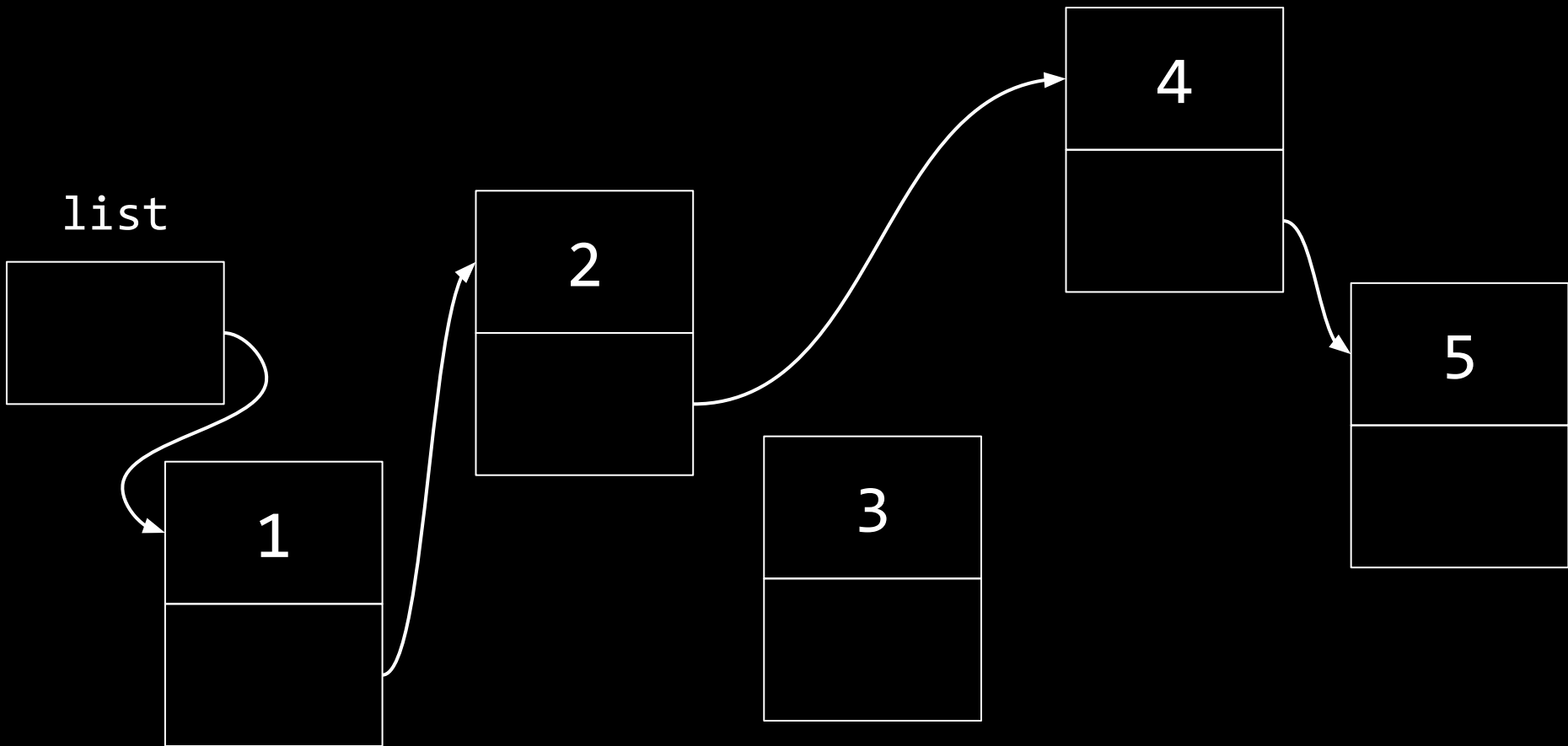


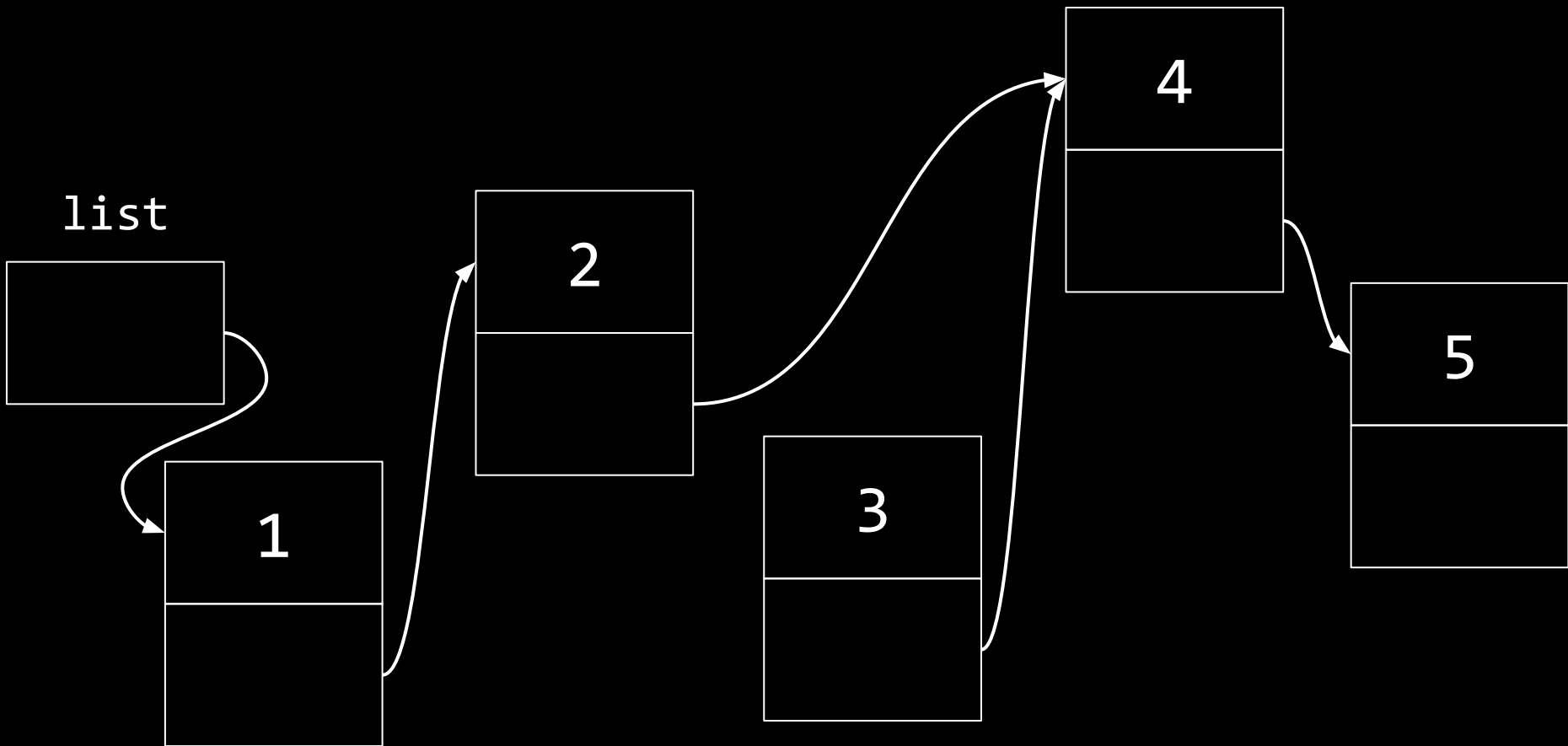
```
n->next = list;
```



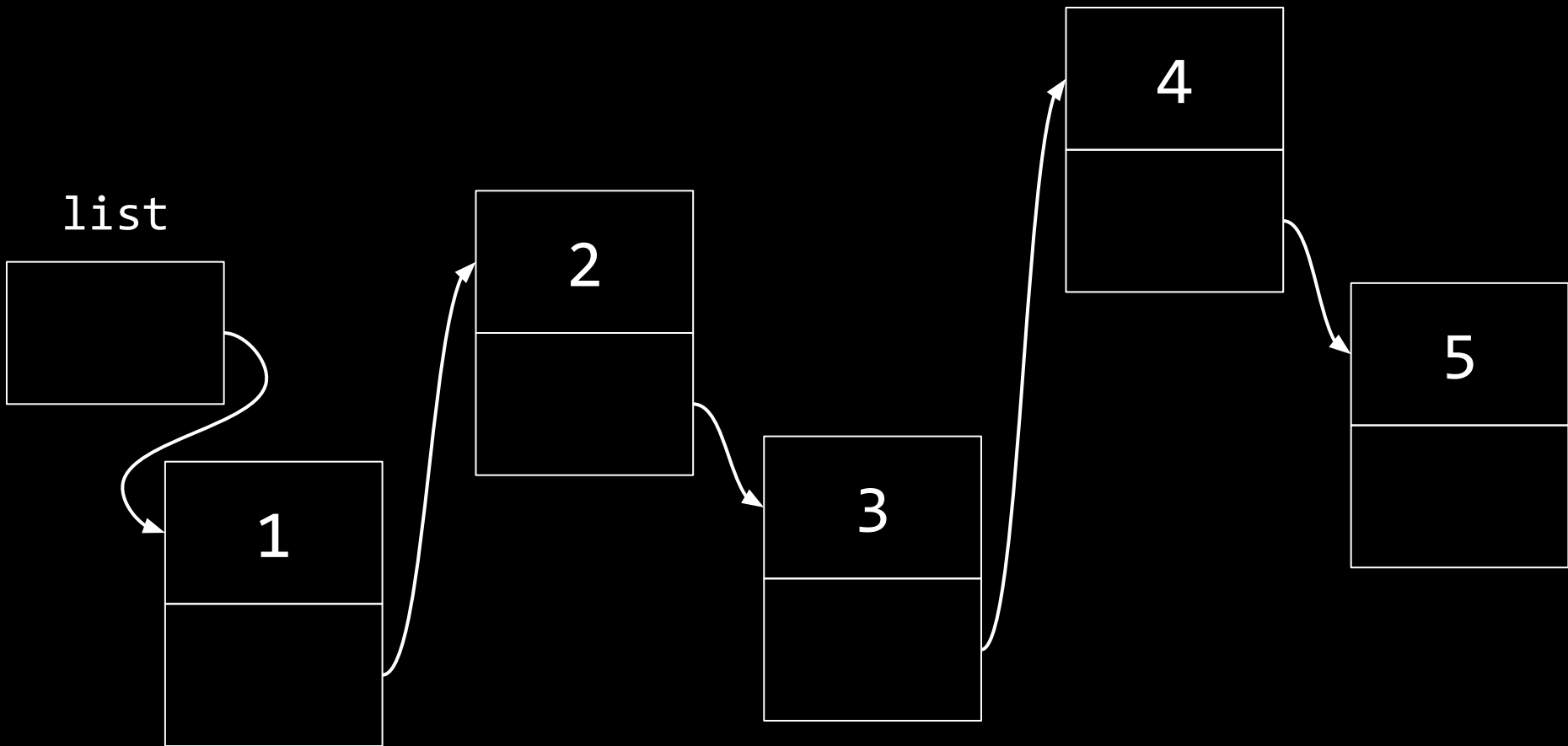
```
list = n;
```











$$O(n^2)$$

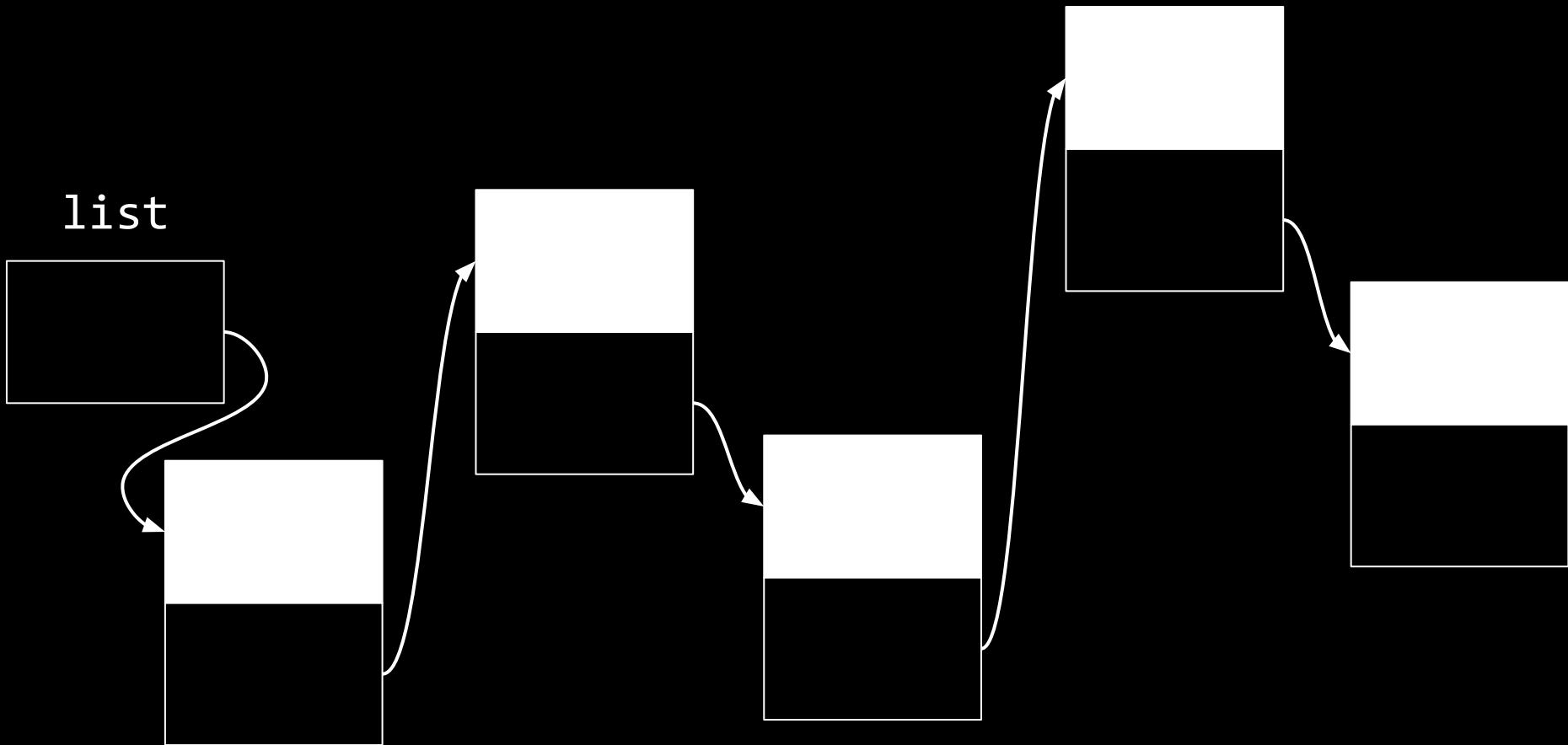
$$O(n \log n)$$

$$O(n)$$

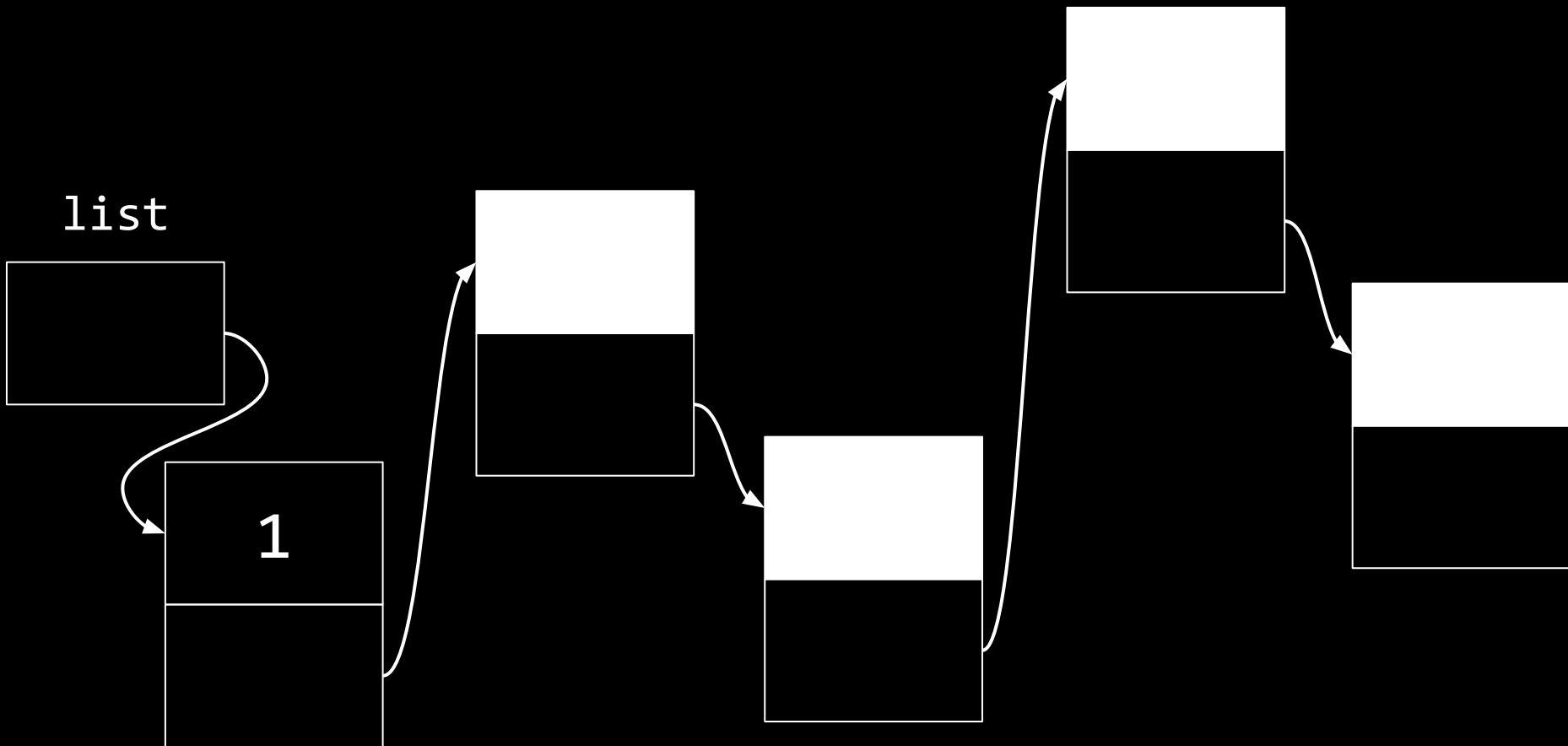
$$O(\log n)$$

$$O(1)$$

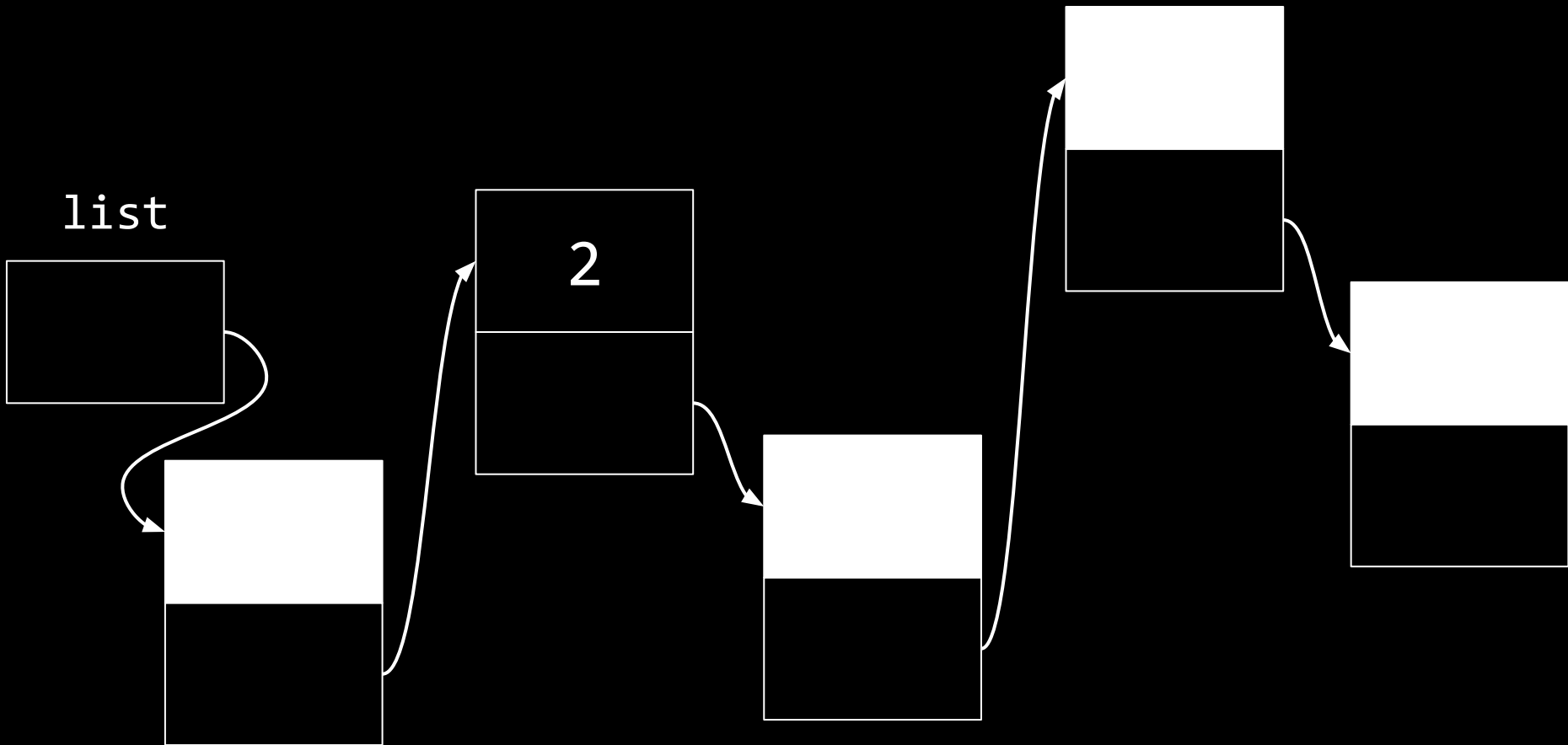
list



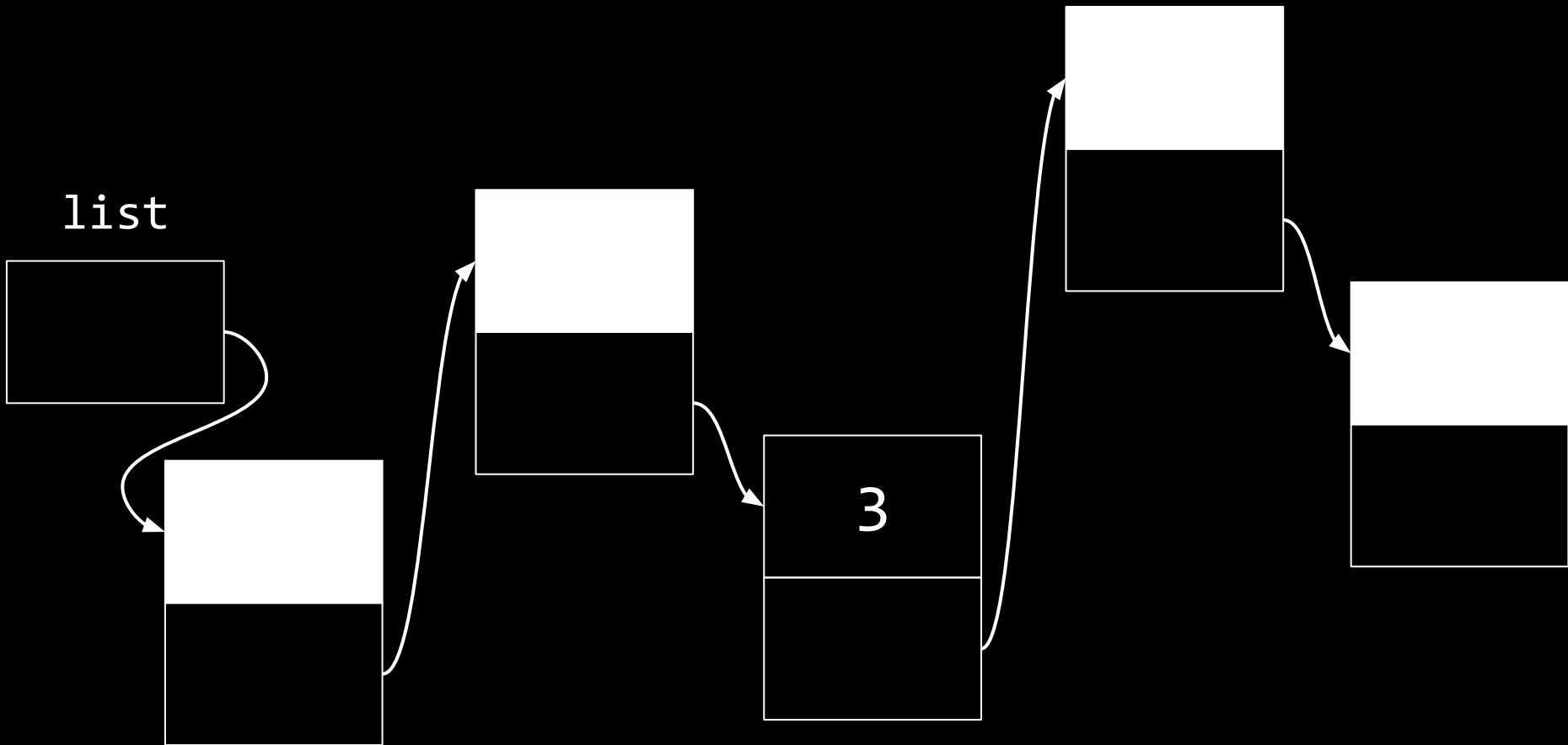
list



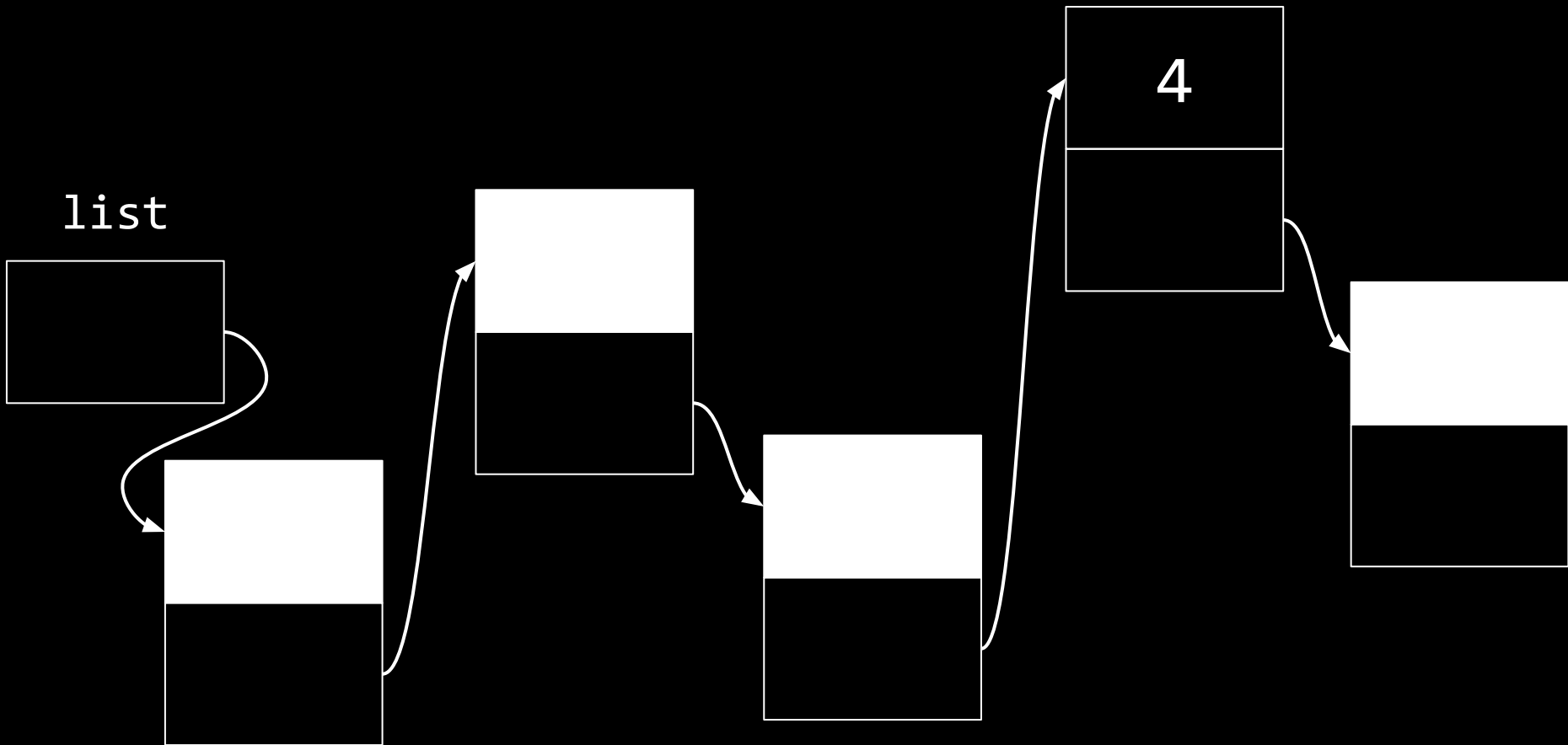
list



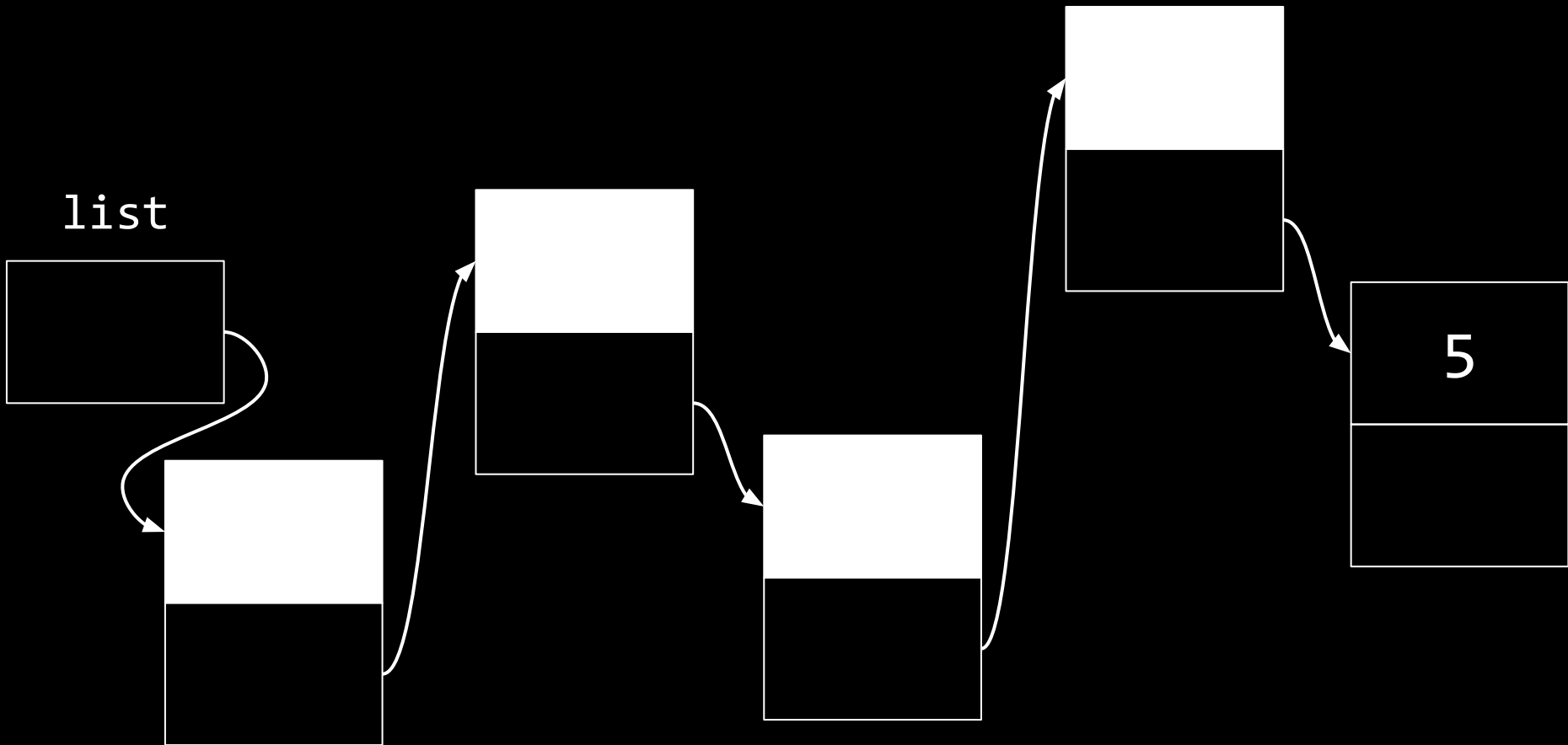
list



list

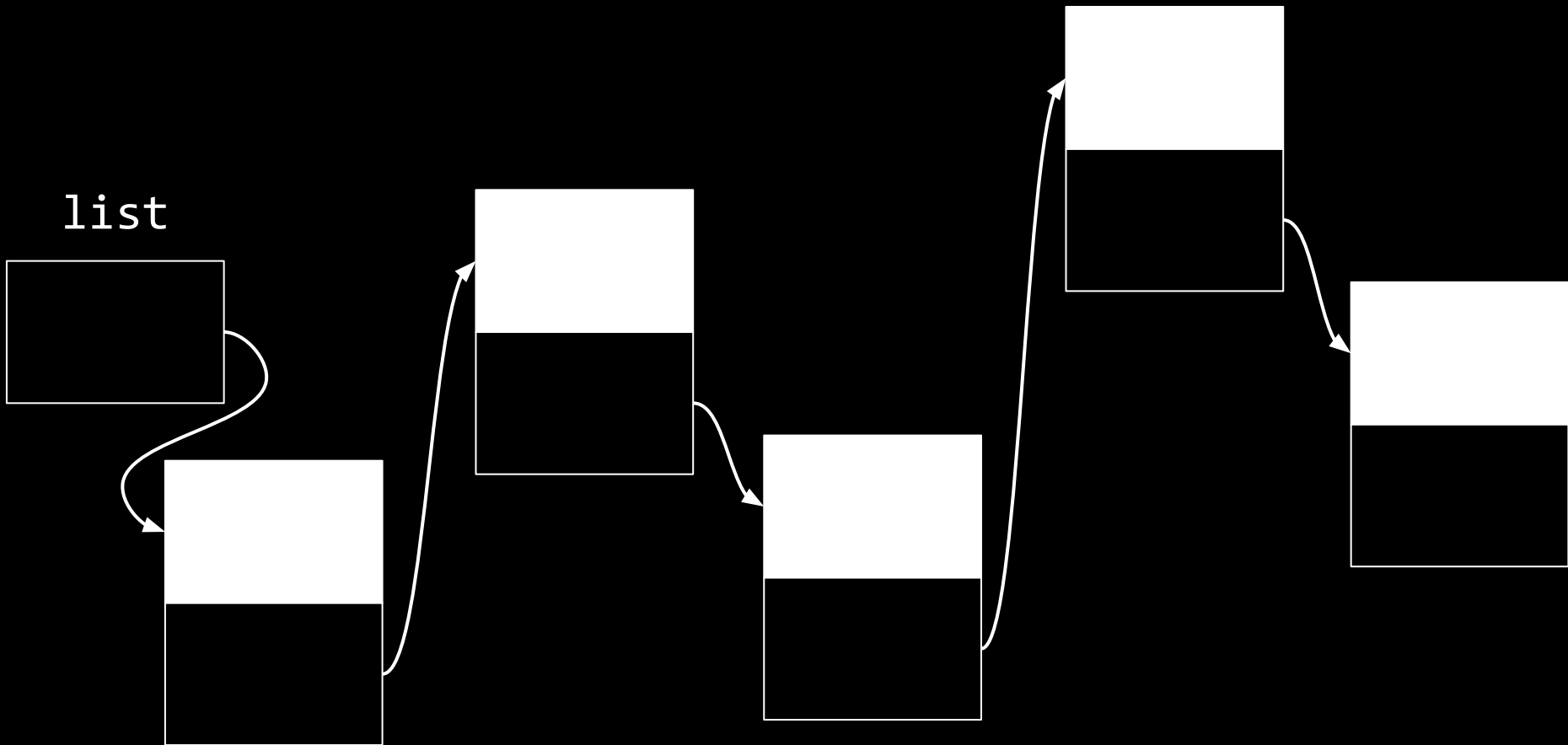


list





list



$O(n^2)$

$O(n \log n)$

$O(n)$       search

$O(\log n)$

$O(1)$

$O(n^2)$

$O(n \log n)$

$O(n)$       search, insert

$O(\log n)$

$O(1)$

$O(n^2)$

$O(n \log n)$

$O(n)$       search

$O(\log n)$

$O(1)$       insert

trees

binary search trees

1

2

3

4

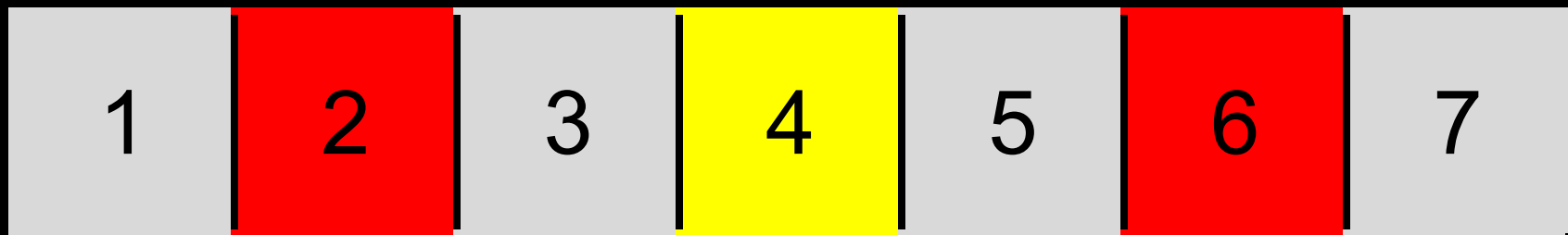
5

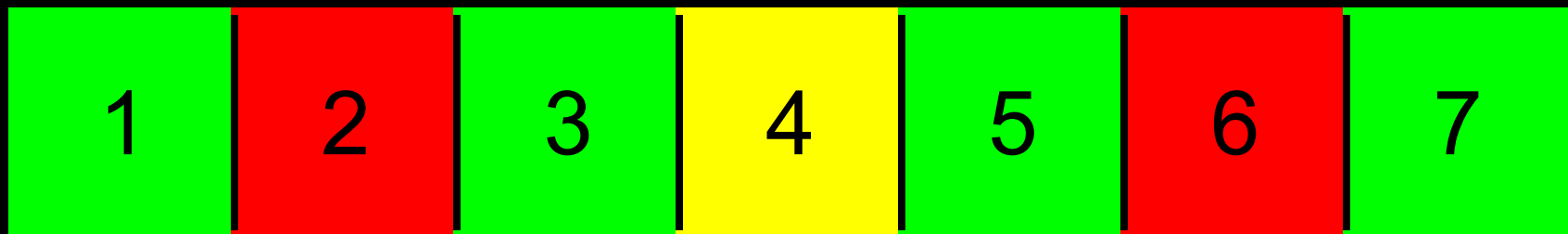
6

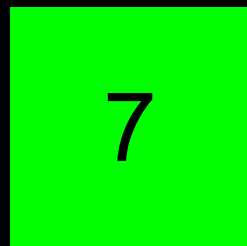
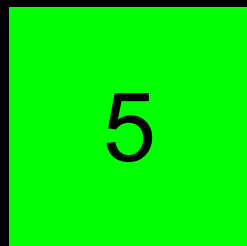
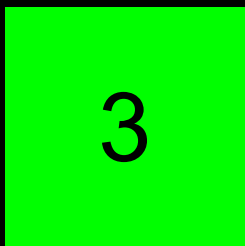
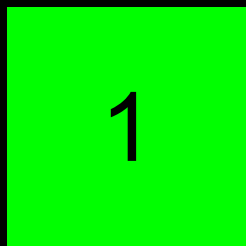
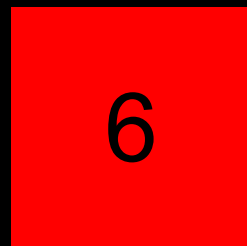
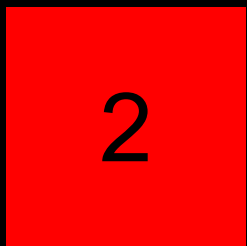
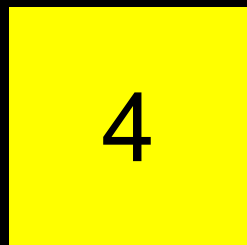
7

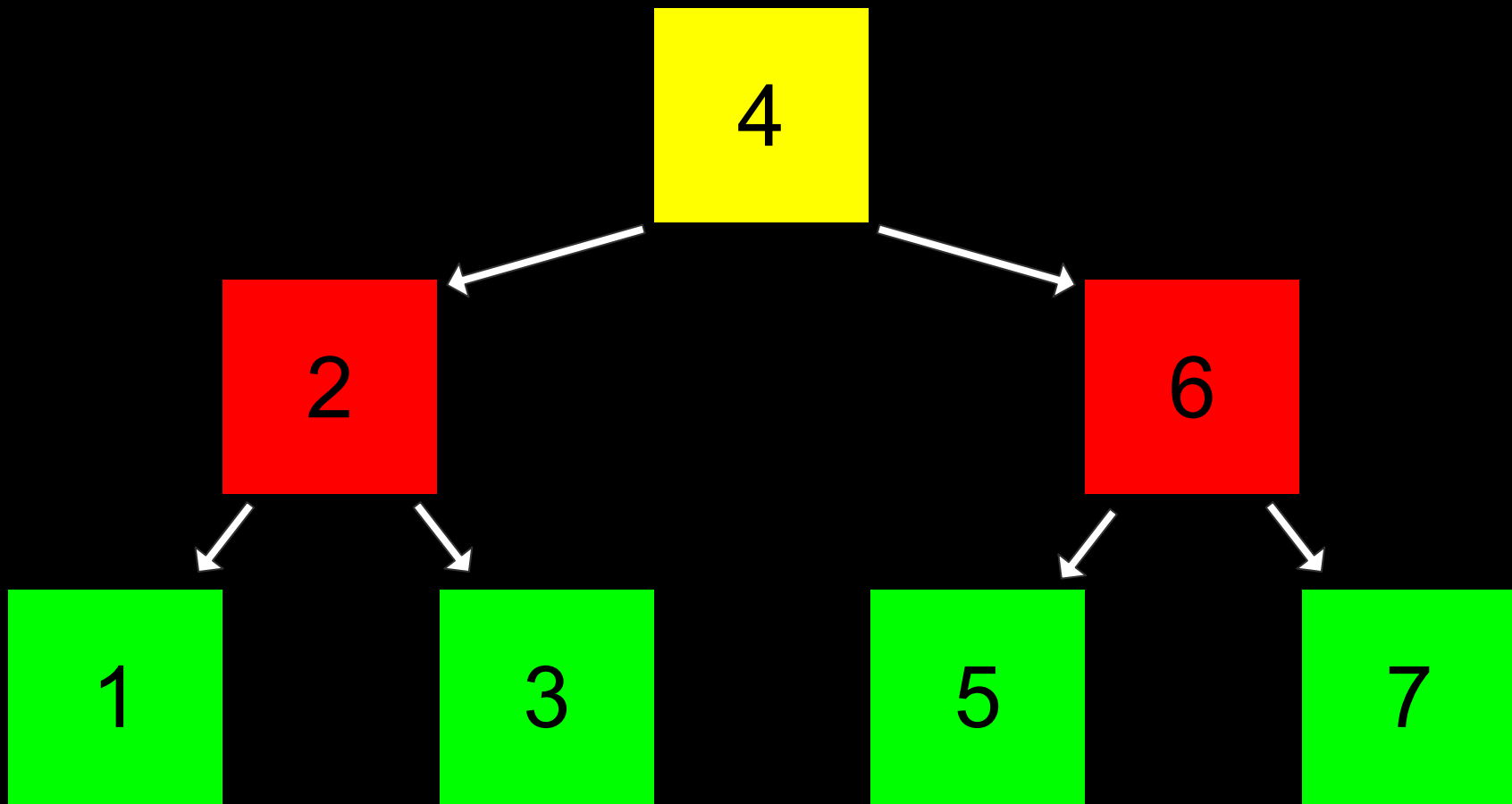
1	2	3	4	5	6	7
---	---	---	---	---	---	---











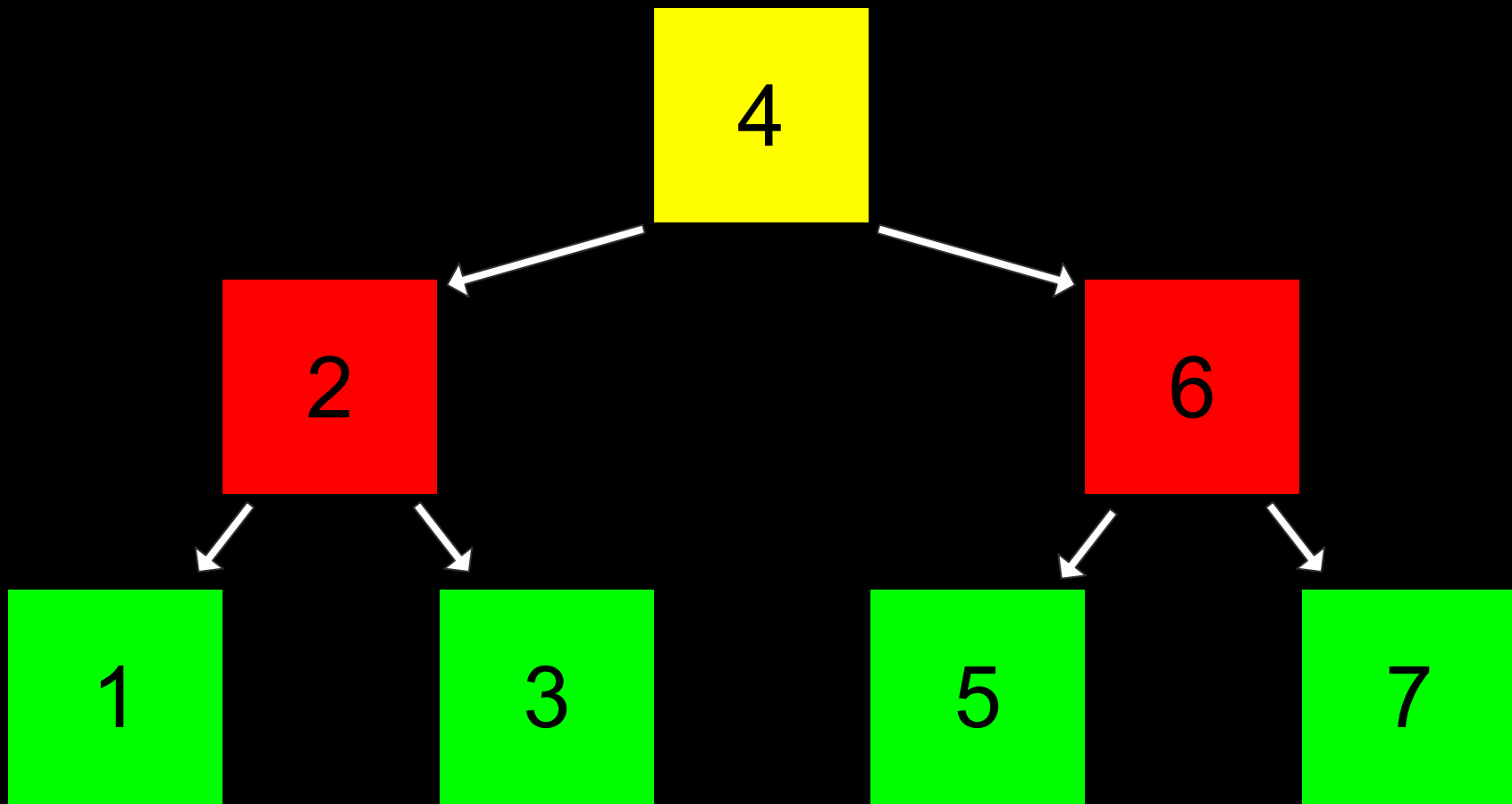
```
typedef struct node
{
    int number;
    struct node *next;
}
node;
```

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

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

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





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

```
}
```

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

}

}
```

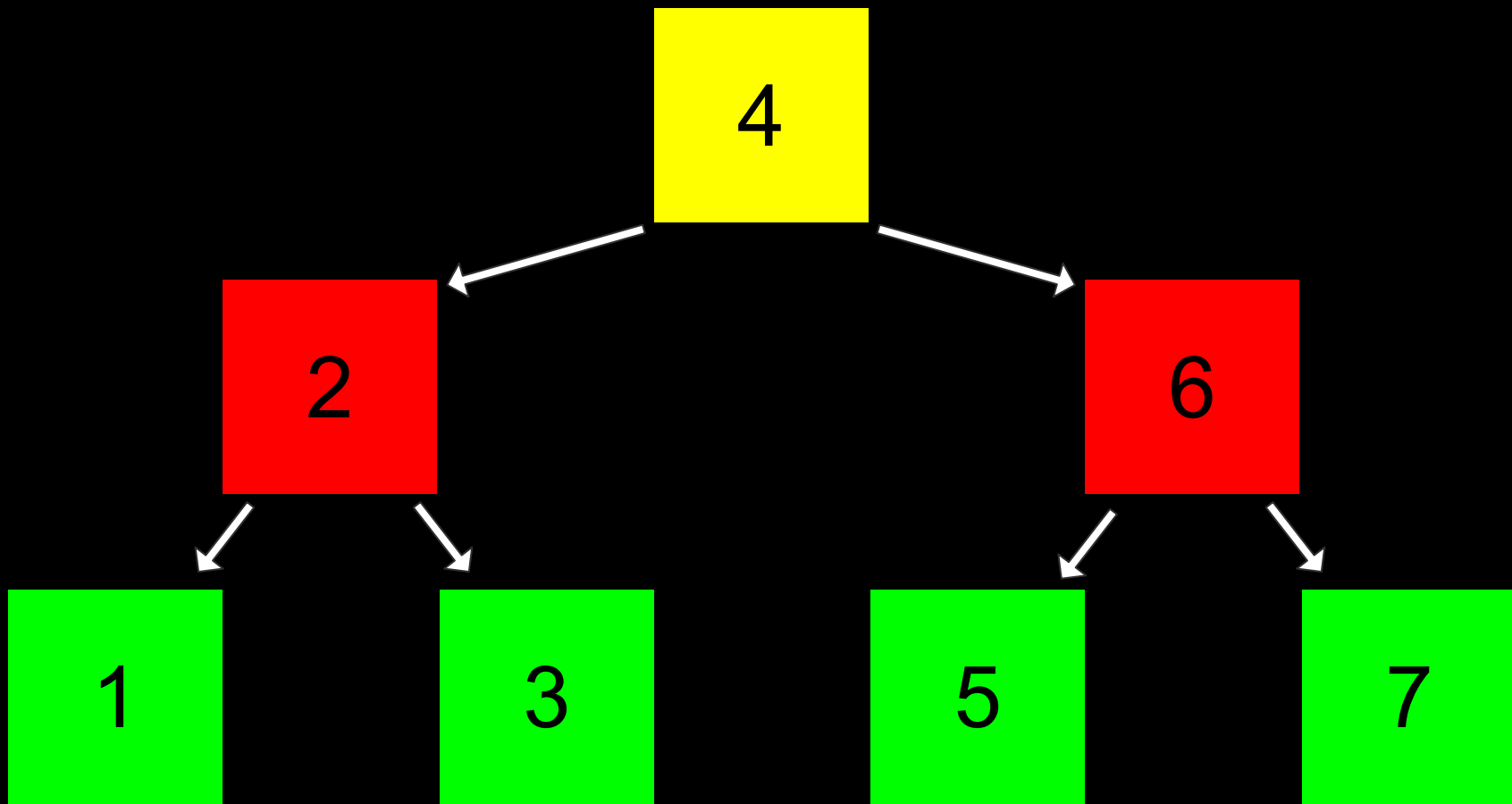
```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (number < tree->number)
    {
        return search(tree->left, number);
    }
}
```

```
}
```

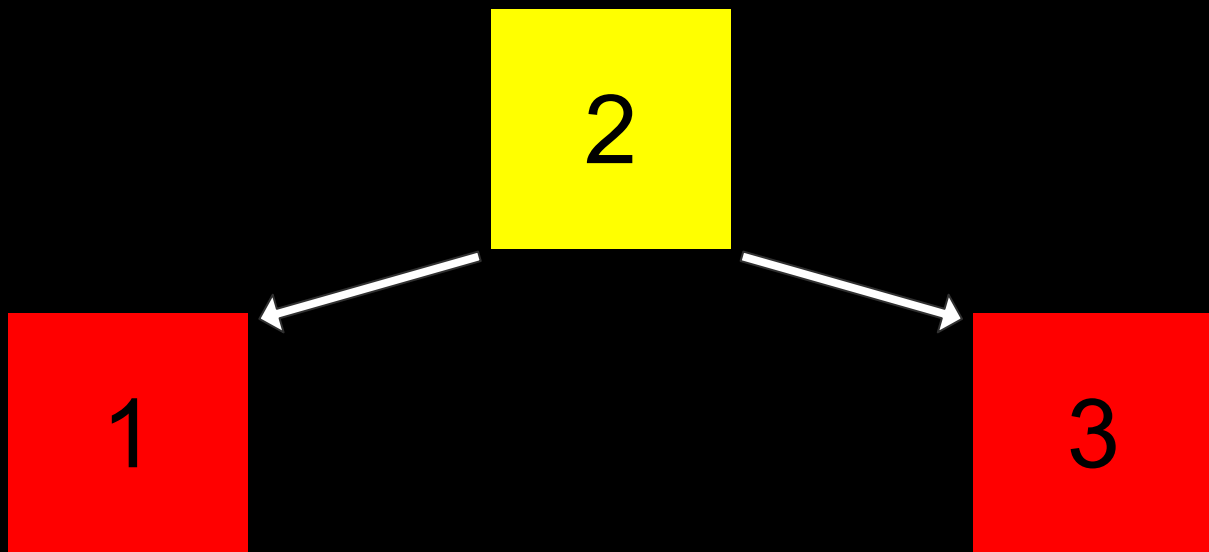
```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (number < tree->number)
    {
        return search(tree->left, number);
    }
    else if (number > tree->number)
    {
        return search(tree->right, number);
    }
}
```

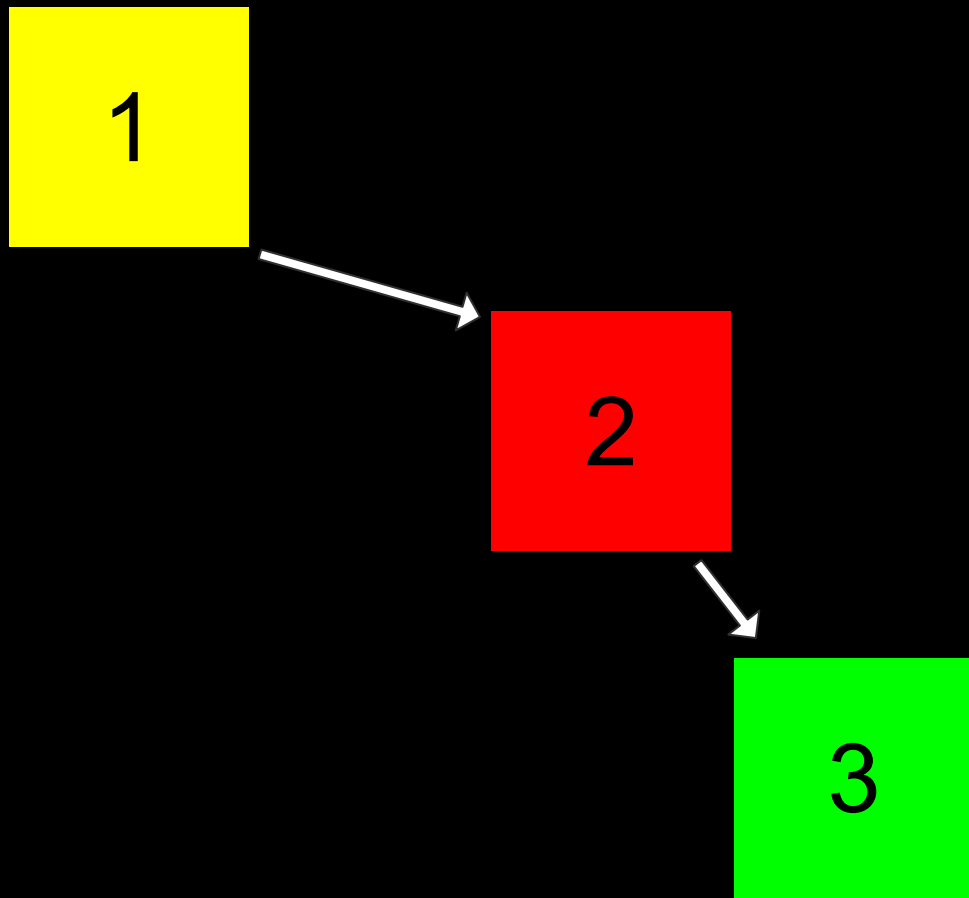
```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (number < tree->number)
    {
        return search(tree->left, number);
    }
    else if (number > tree->number)
    {
        return search(tree->right, number);
    }
    else if (number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









$$O(n^2)$$

$$O(n \log n)$$

$$O(n)$$

$$O(\log n)$$

$$O(1)$$

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$       search

$O(1)$

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$       search, insert

$O(1)$

hash tables

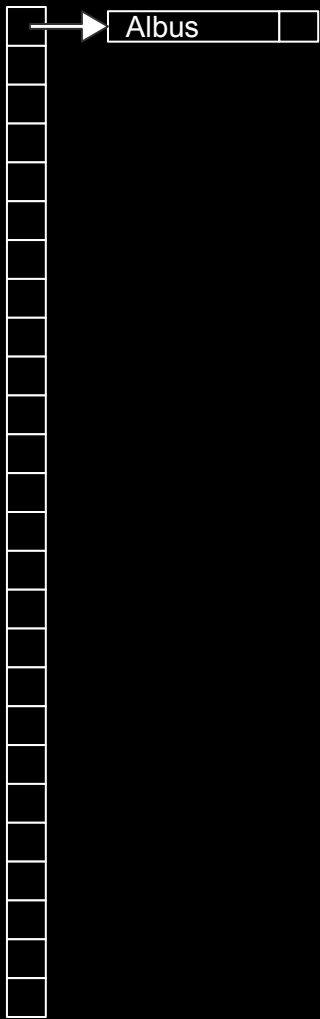
[illegible]

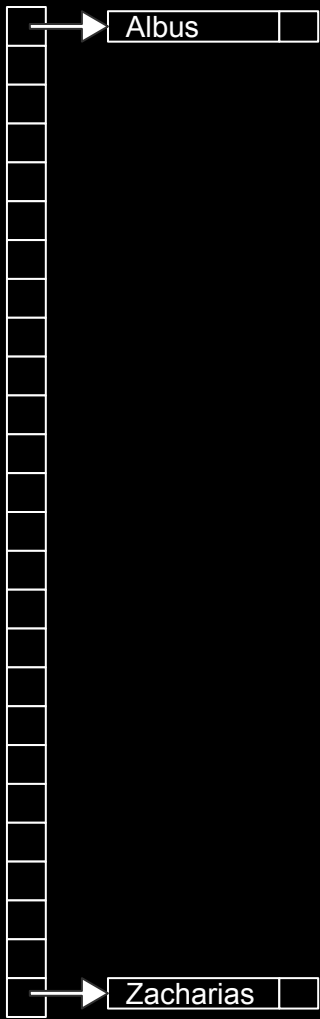
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

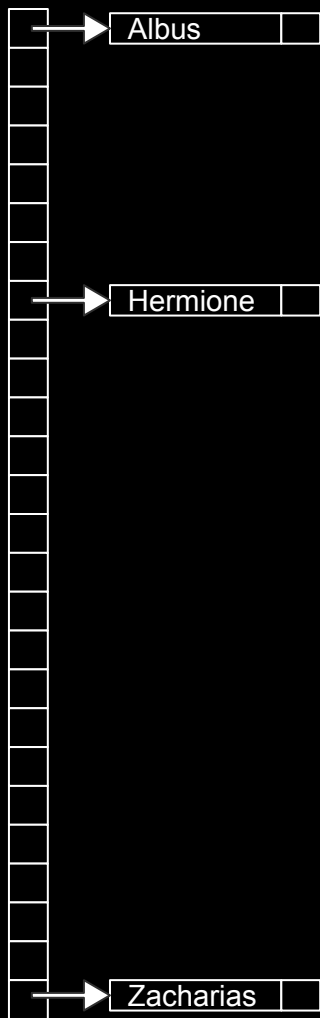


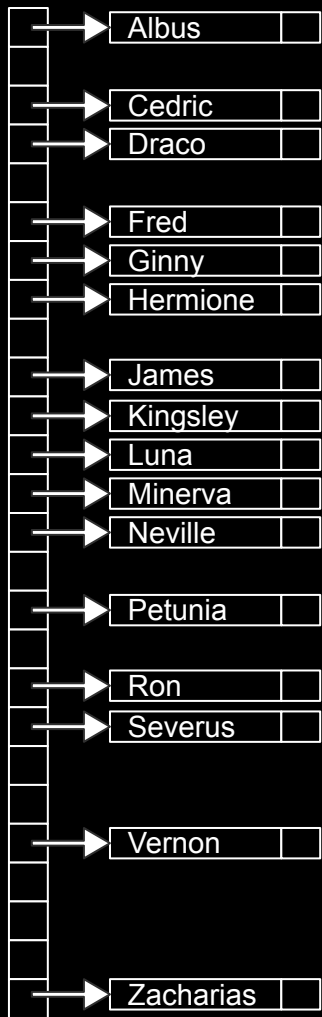
A	
B	
C	
D	
E	
F	
G	
H	
I	
J	
K	
L	
M	
N	
O	
P	
Q	
R	
S	
T	
U	
V	
W	
X	
Y	
Z	

[illegible]

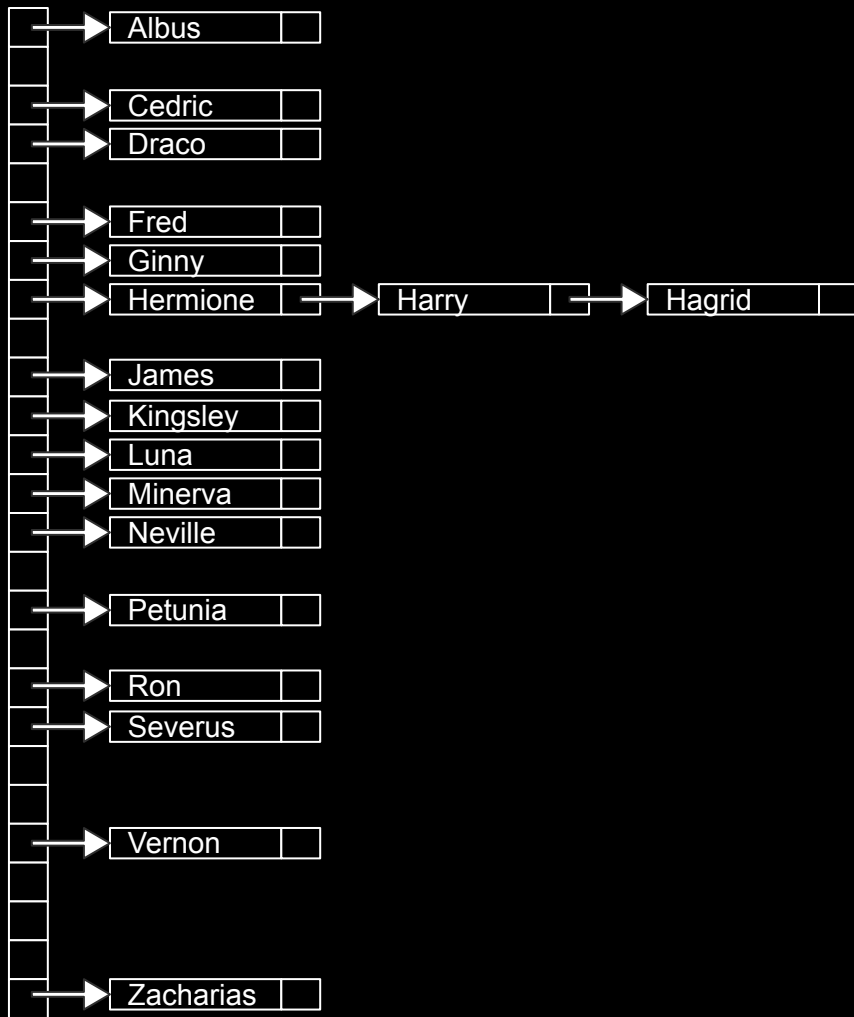




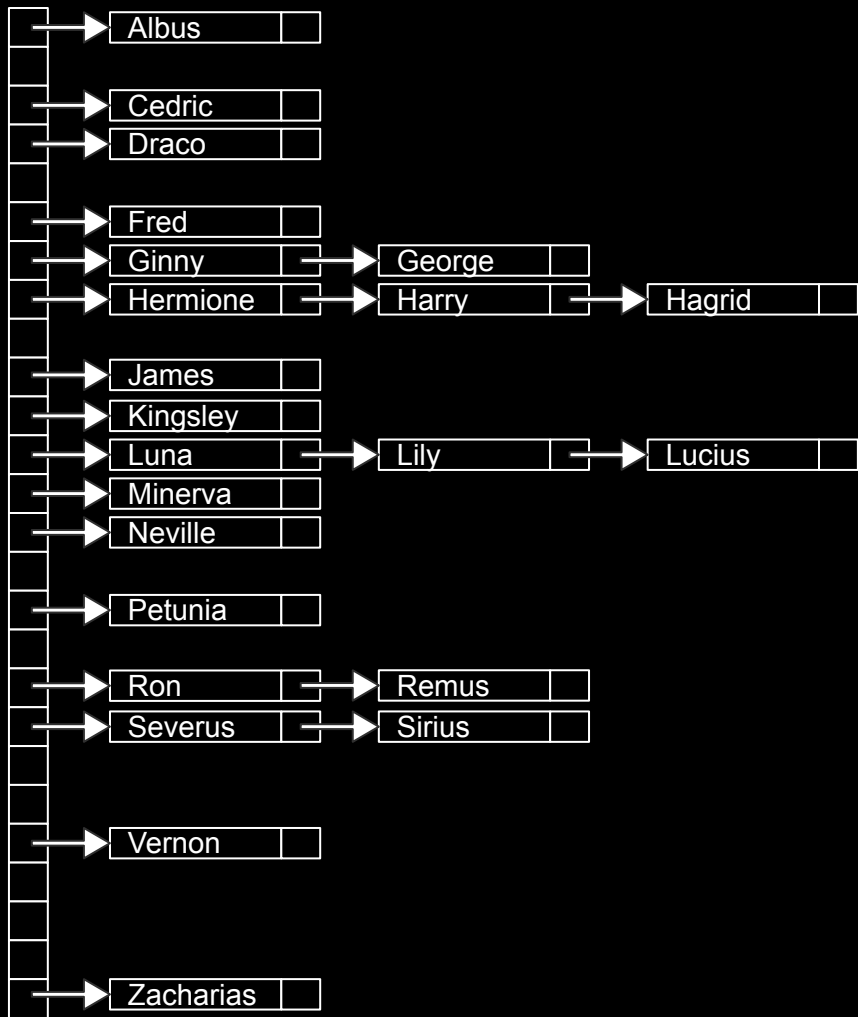












```
typedef struct node
{
    char word[LONGEST_WORD + 1];
    struct node *next;
}
node;
```

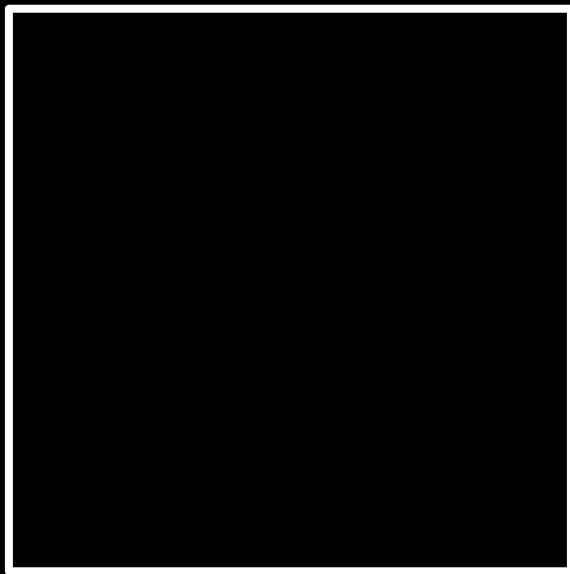
```
node *hash_table[NUMBER_OF_BUCKETS];
```





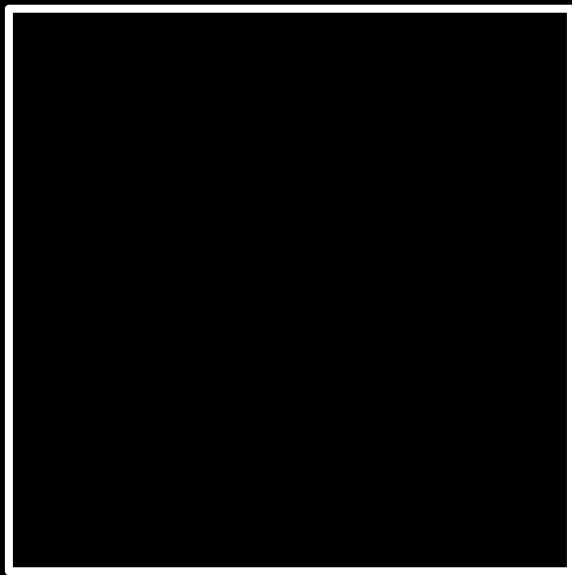
hash function

Albus

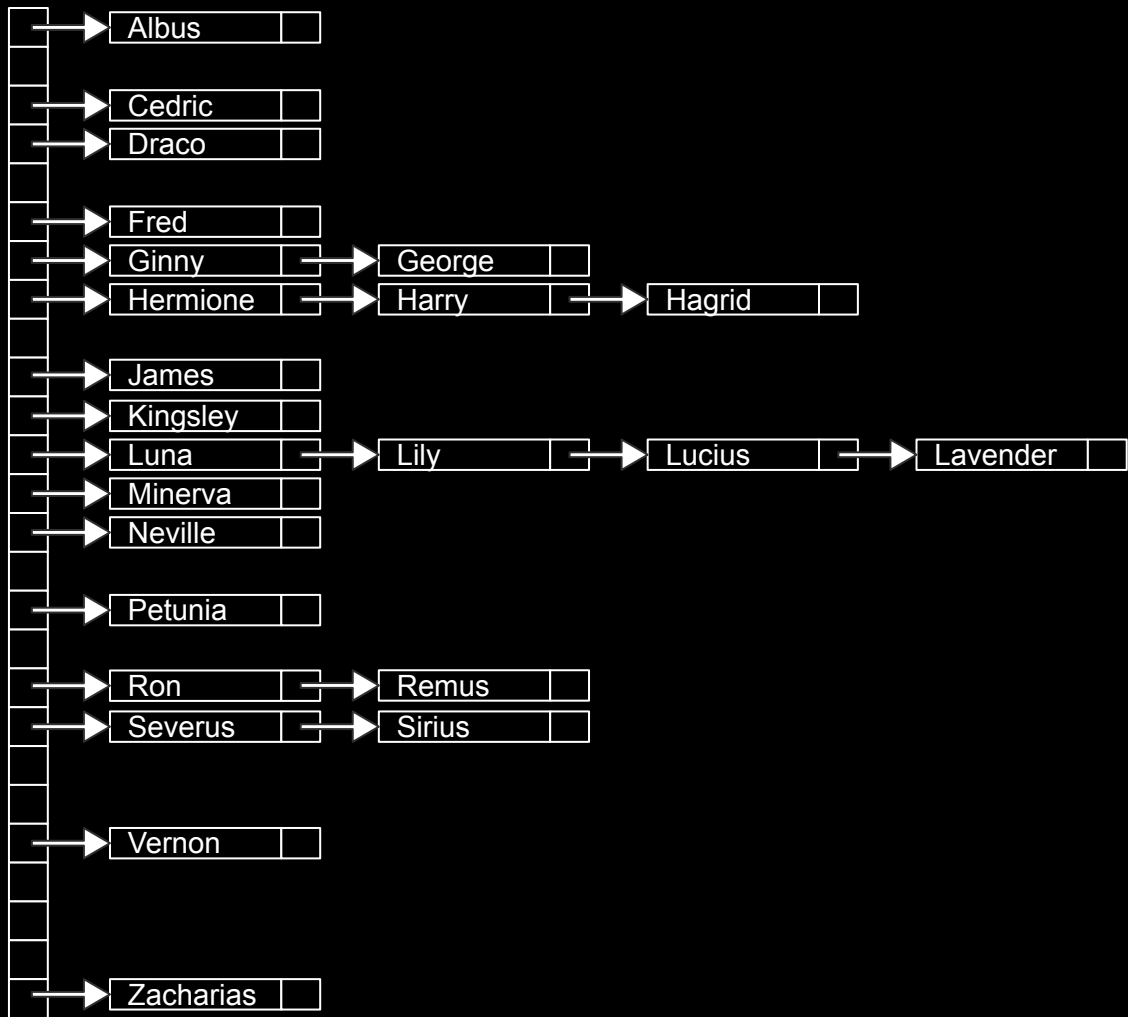


0

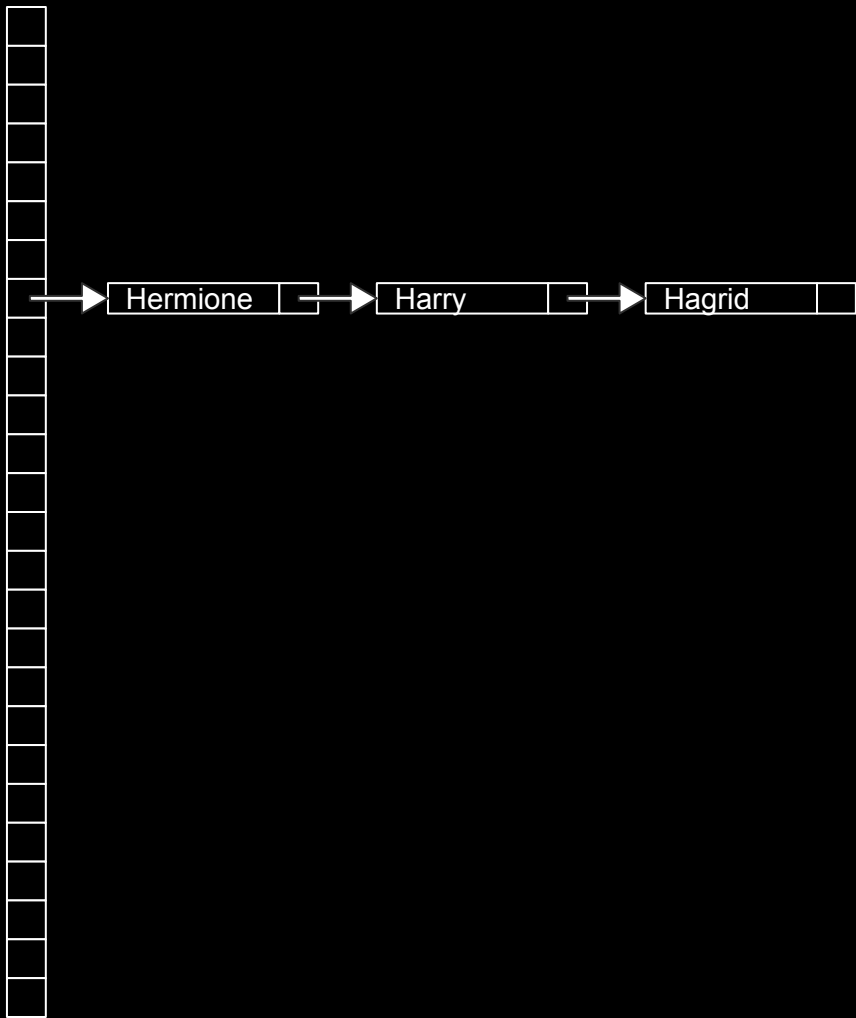
Zacharias →



→ 25









[illegible]

Ha

[illegible]

Ha

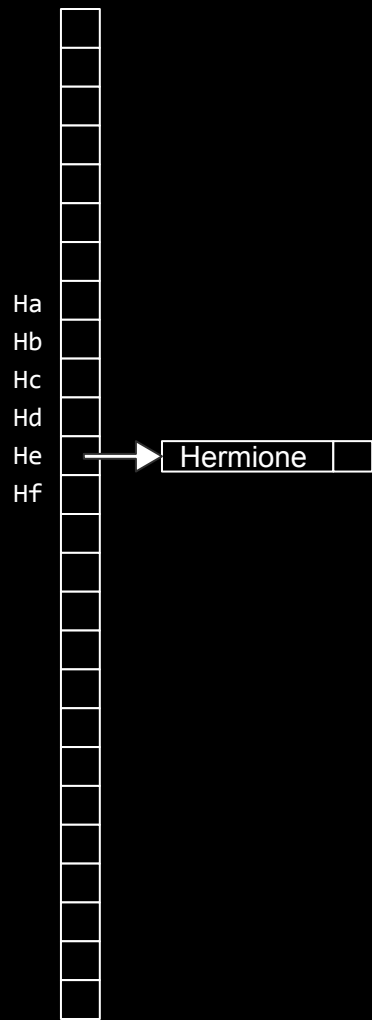
Hb

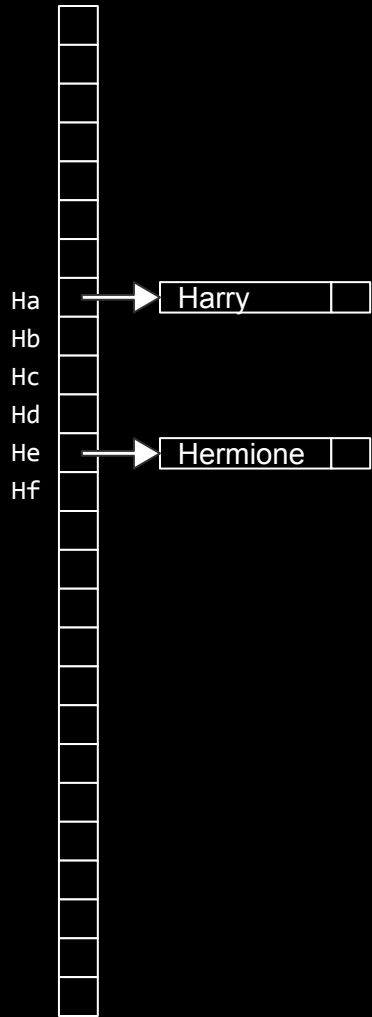
Hc

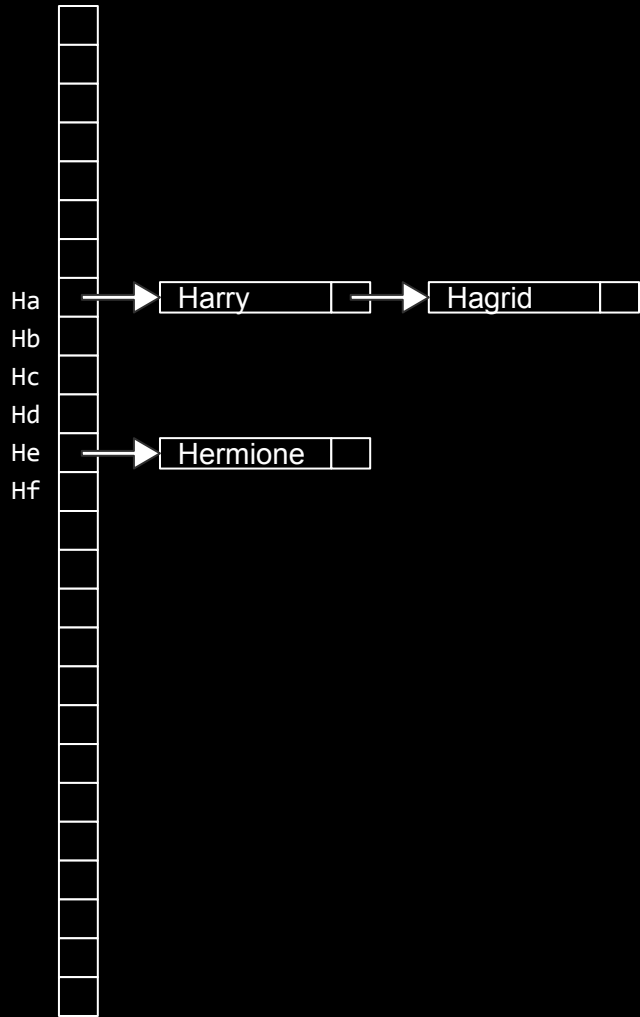
Hd

He

Hf









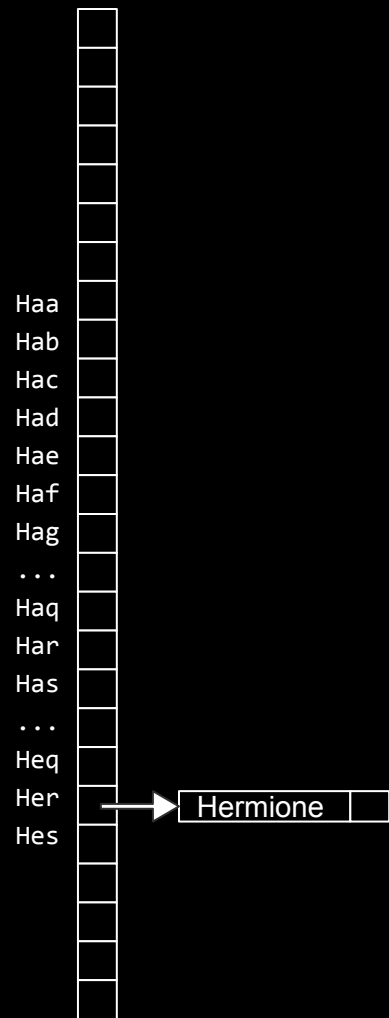
Ha

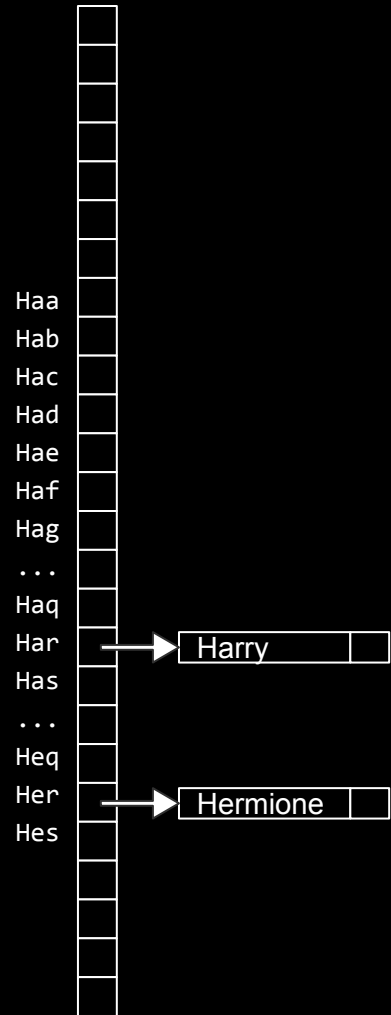
[illegible]

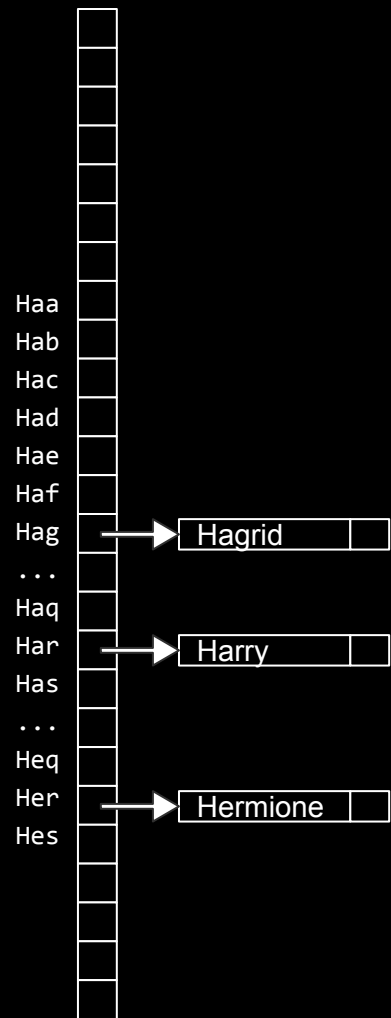
Haa

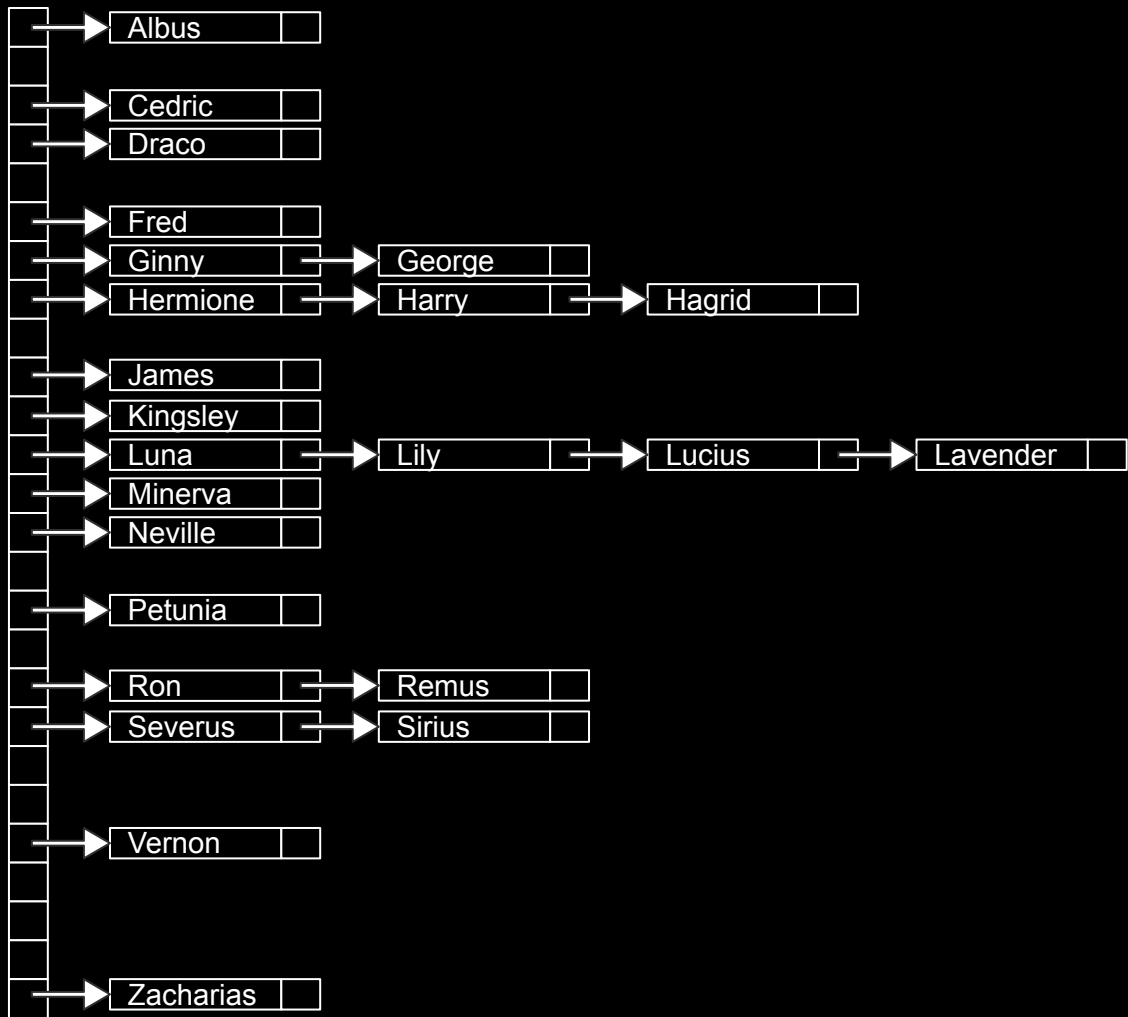
[illegible]

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









$$O(n^2)$$

$$O(n \log n)$$

$$O(n)$$

$$O(\log n)$$

$$O(1)$$



$O(n^2)$

$O(n \log n)$

$O(n)$       search

$O(\log n)$

$O(1)$

$O(n^2)$

$O(n \log n)$

$O(n)$       search, insert

$O(\log n)$

$O(1)$

$O(n^2)$

$O(n \log n)$

$O(n)$       search

$O(\log n)$

$O(1)$       insert

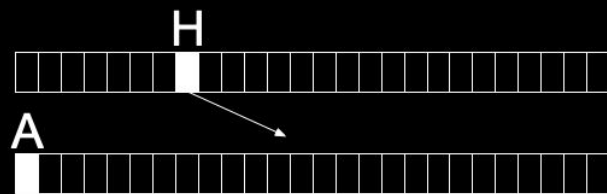
tries



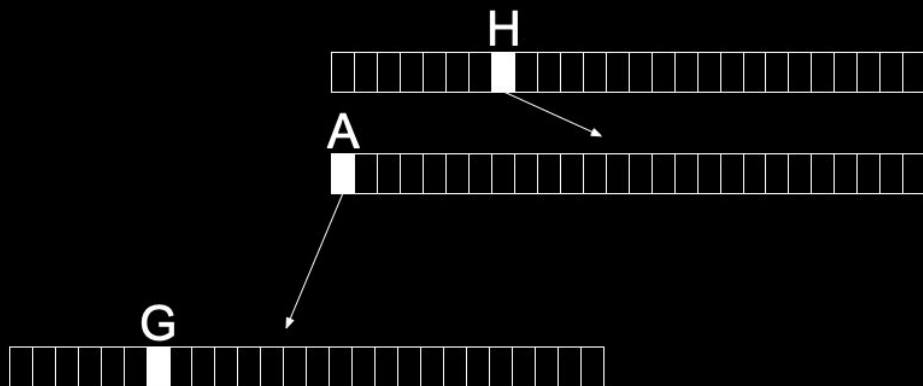


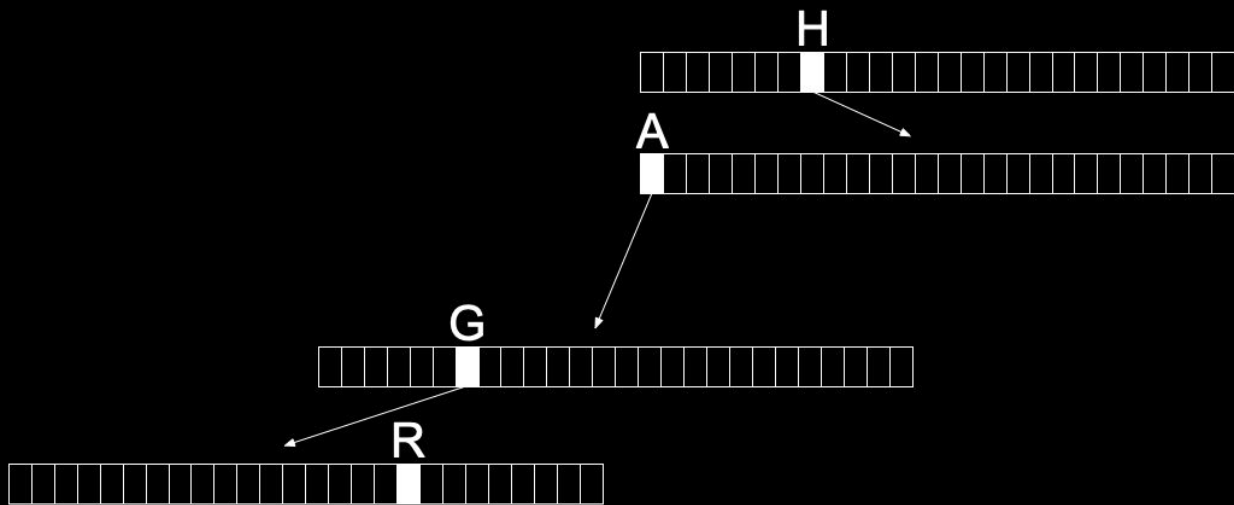
H

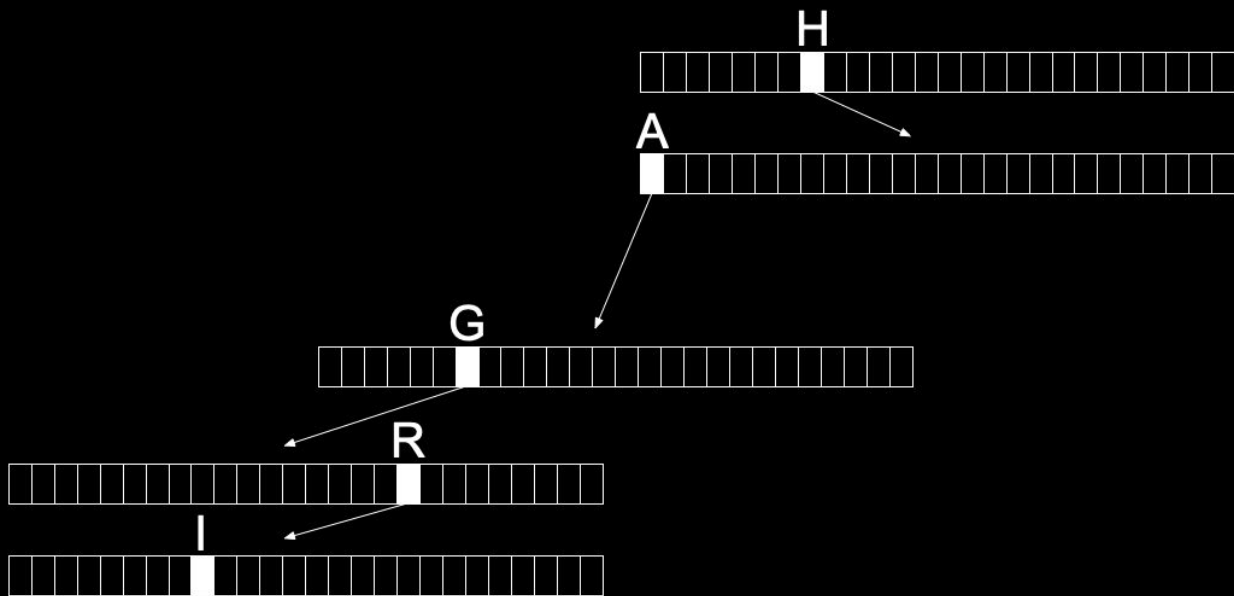


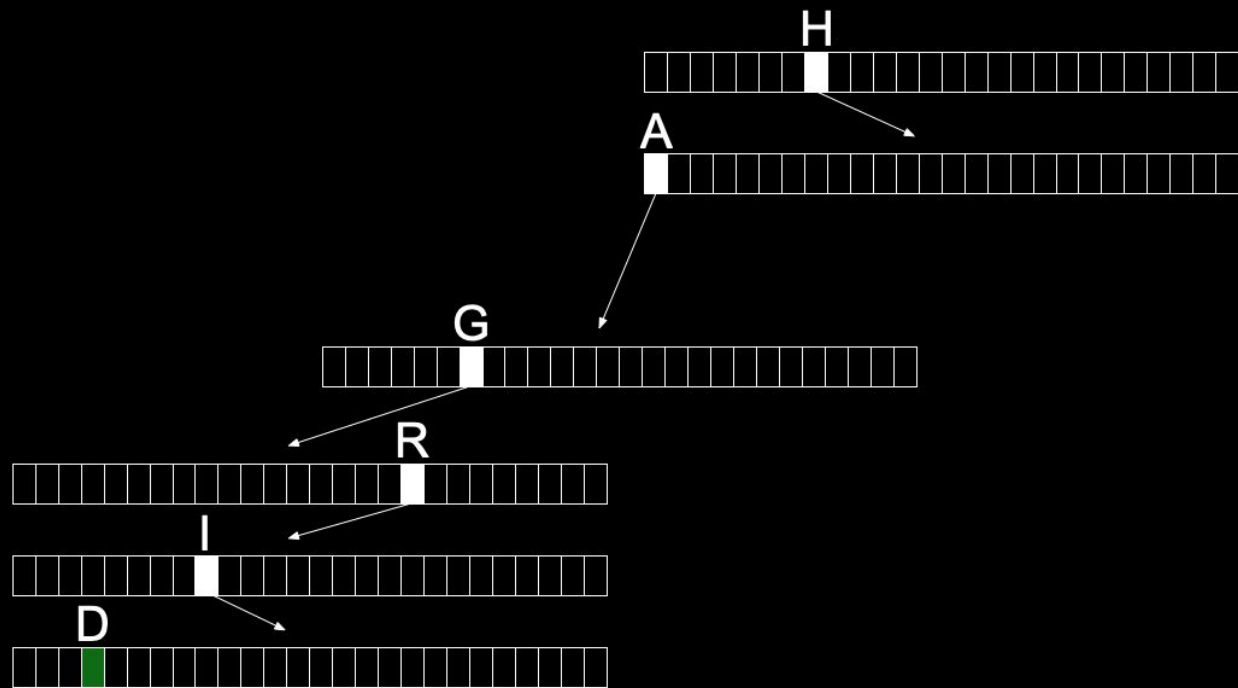


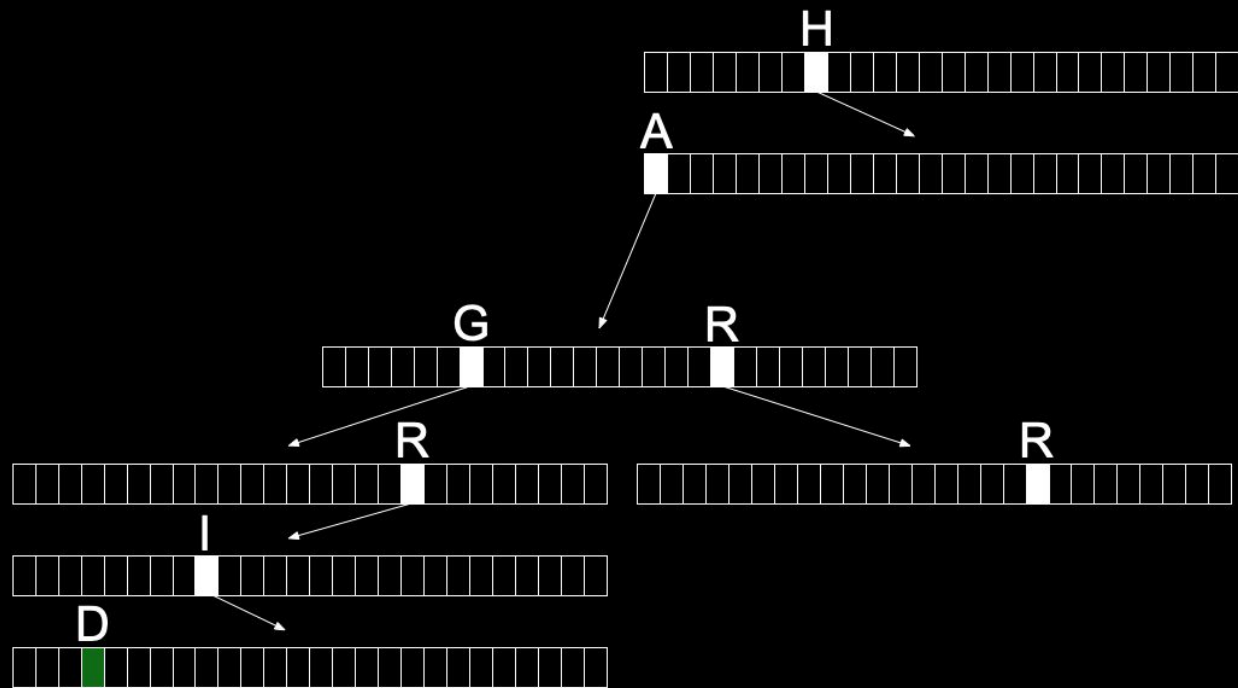


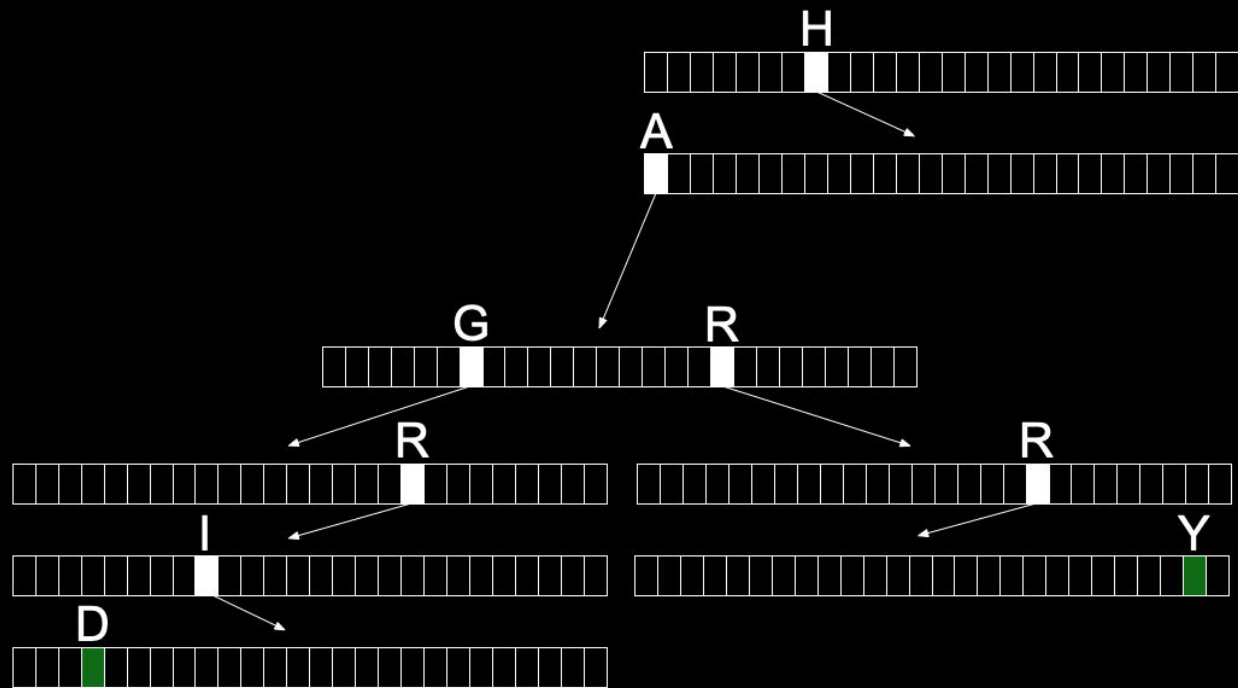


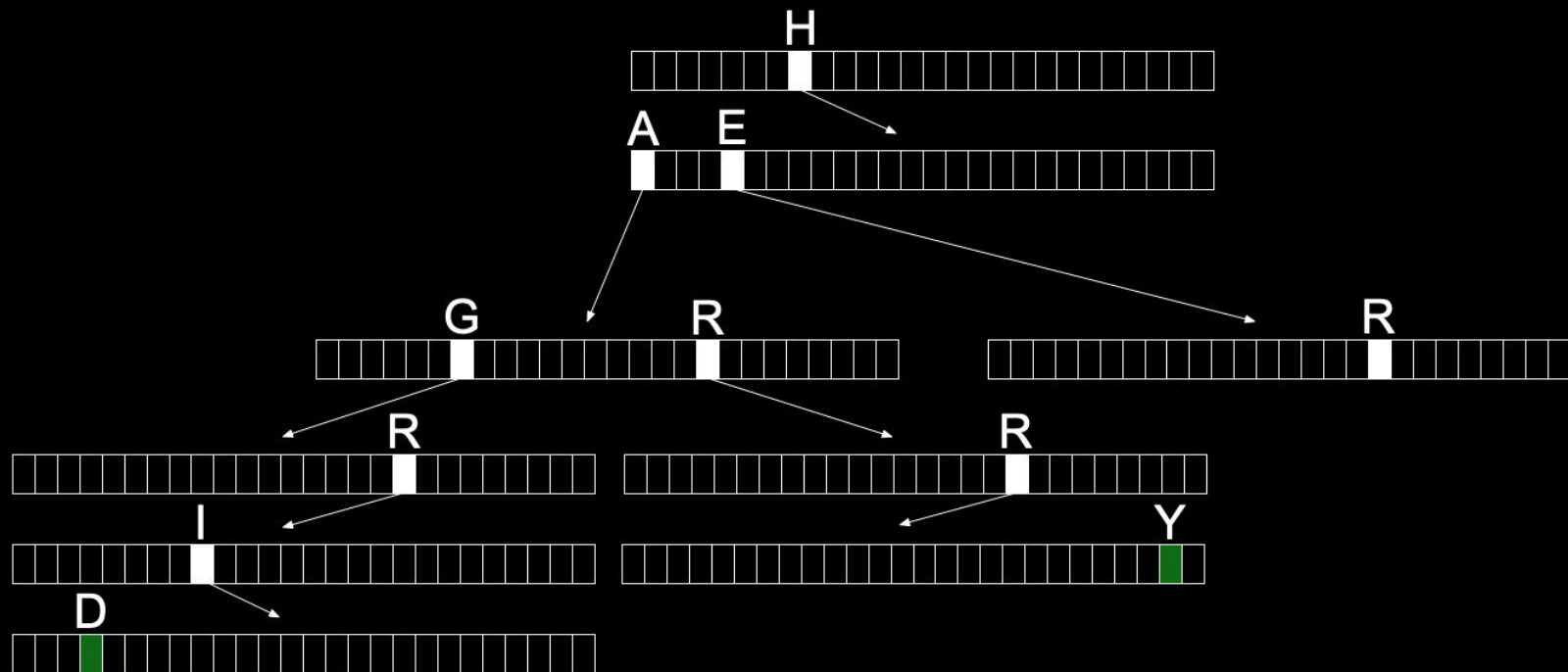


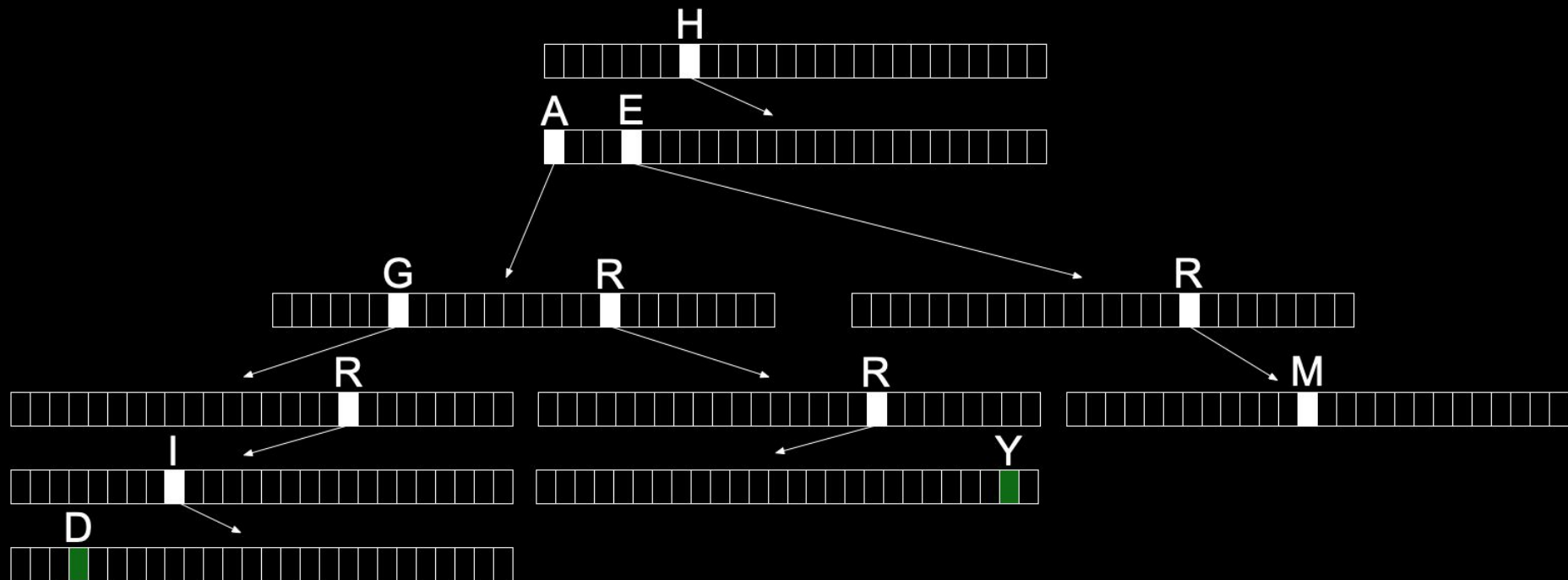




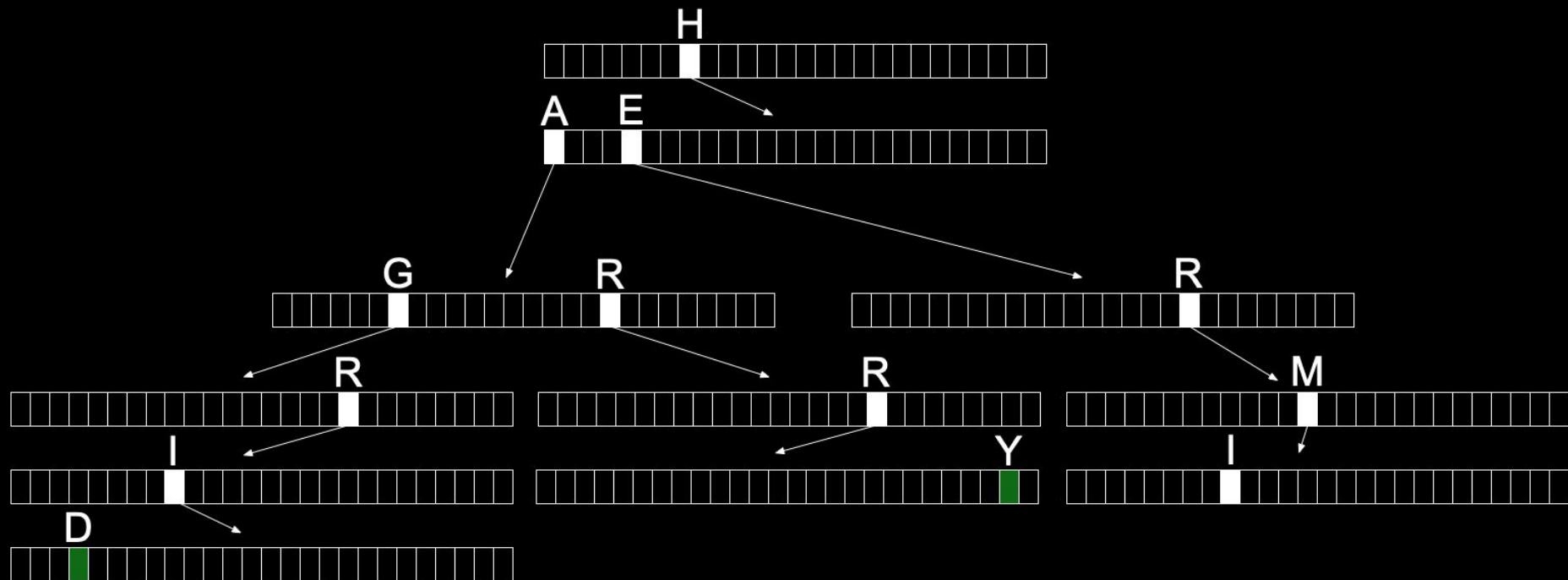


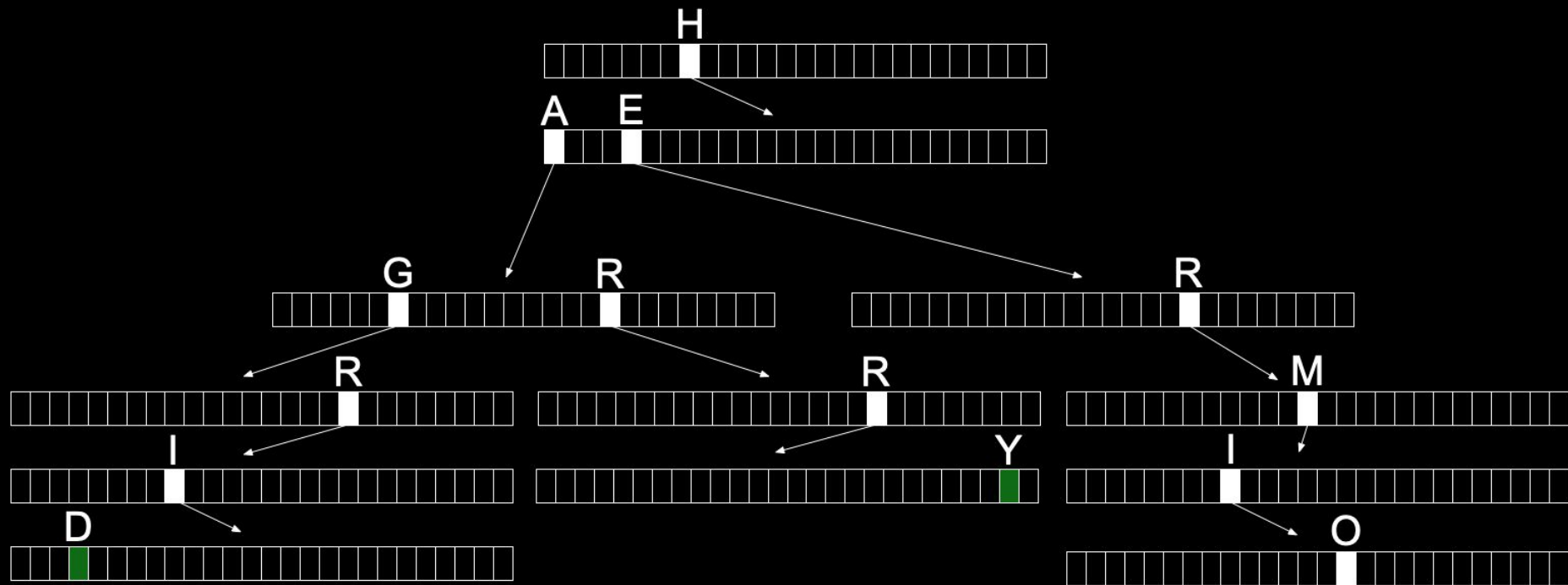


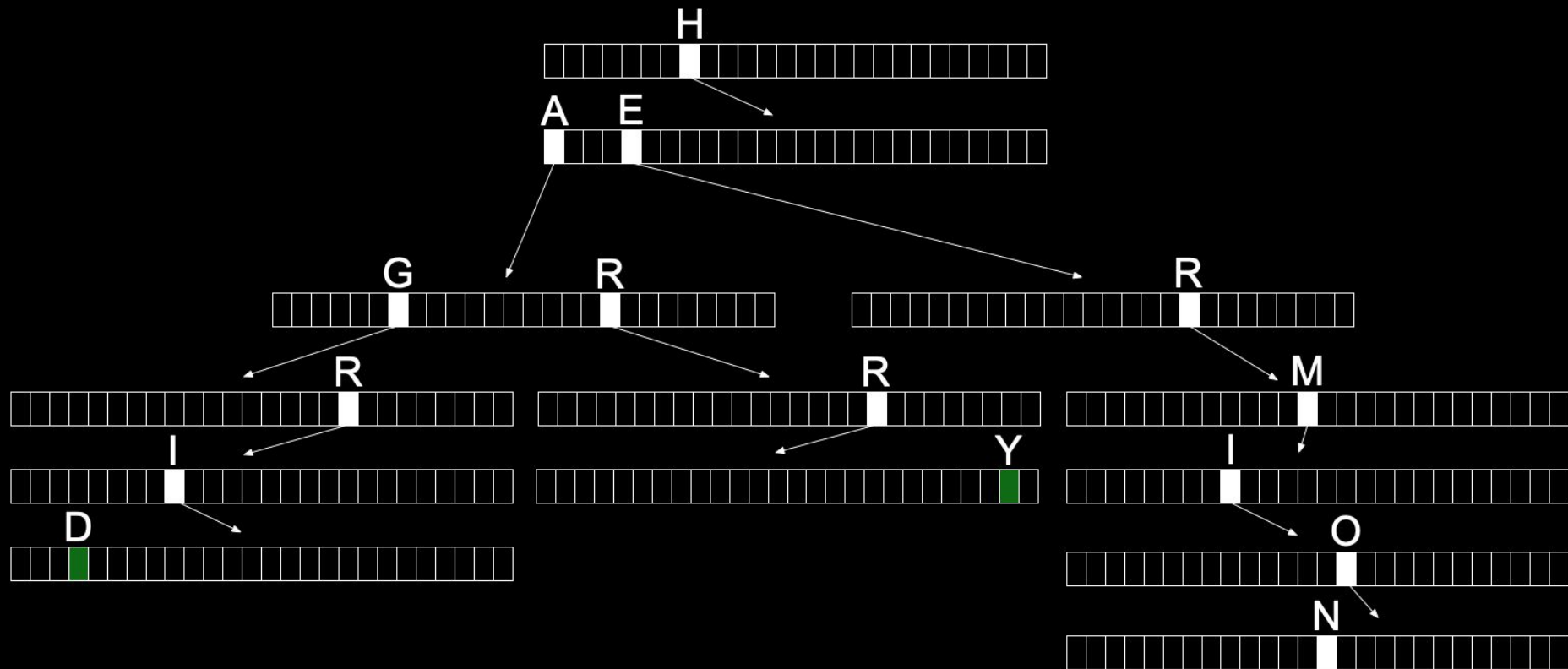


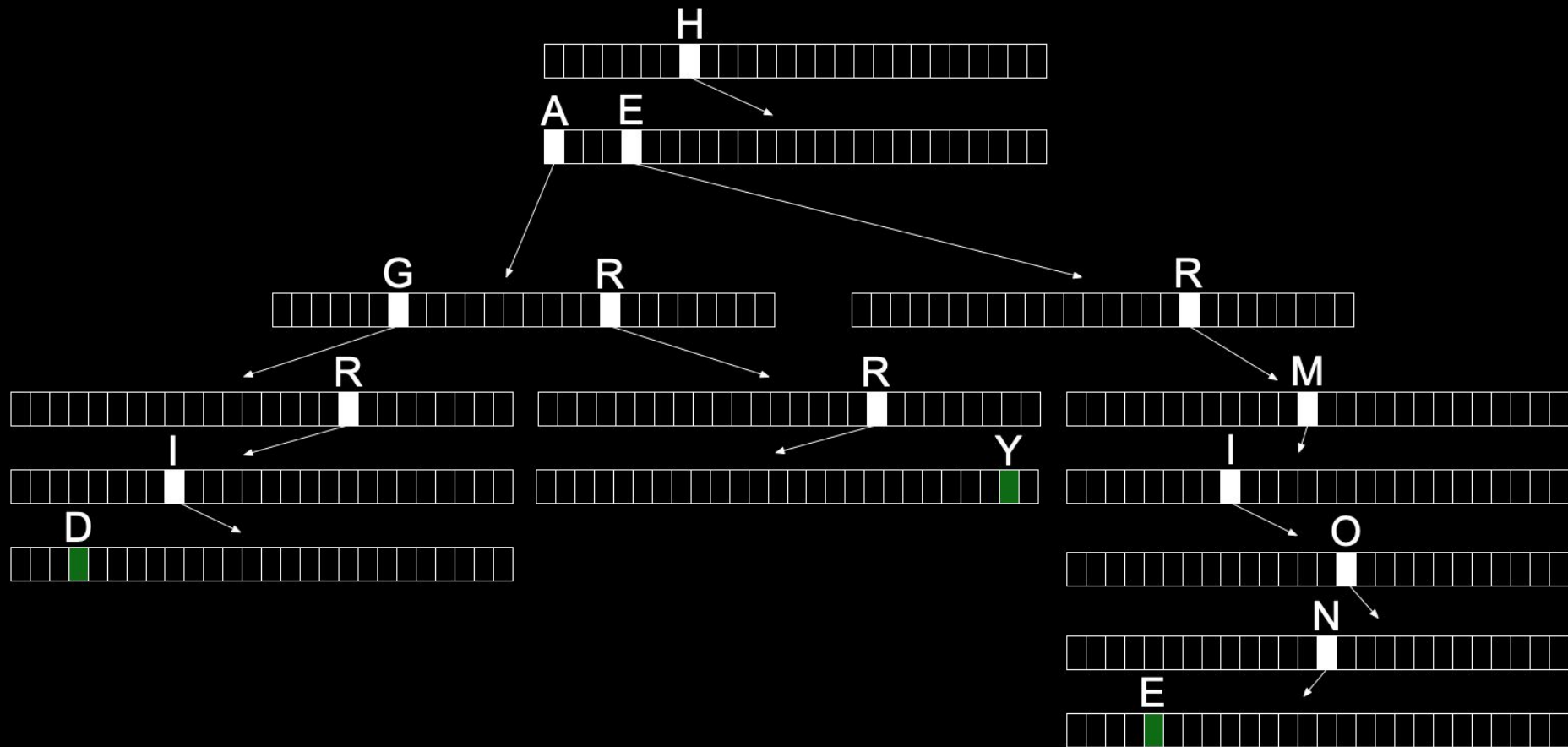






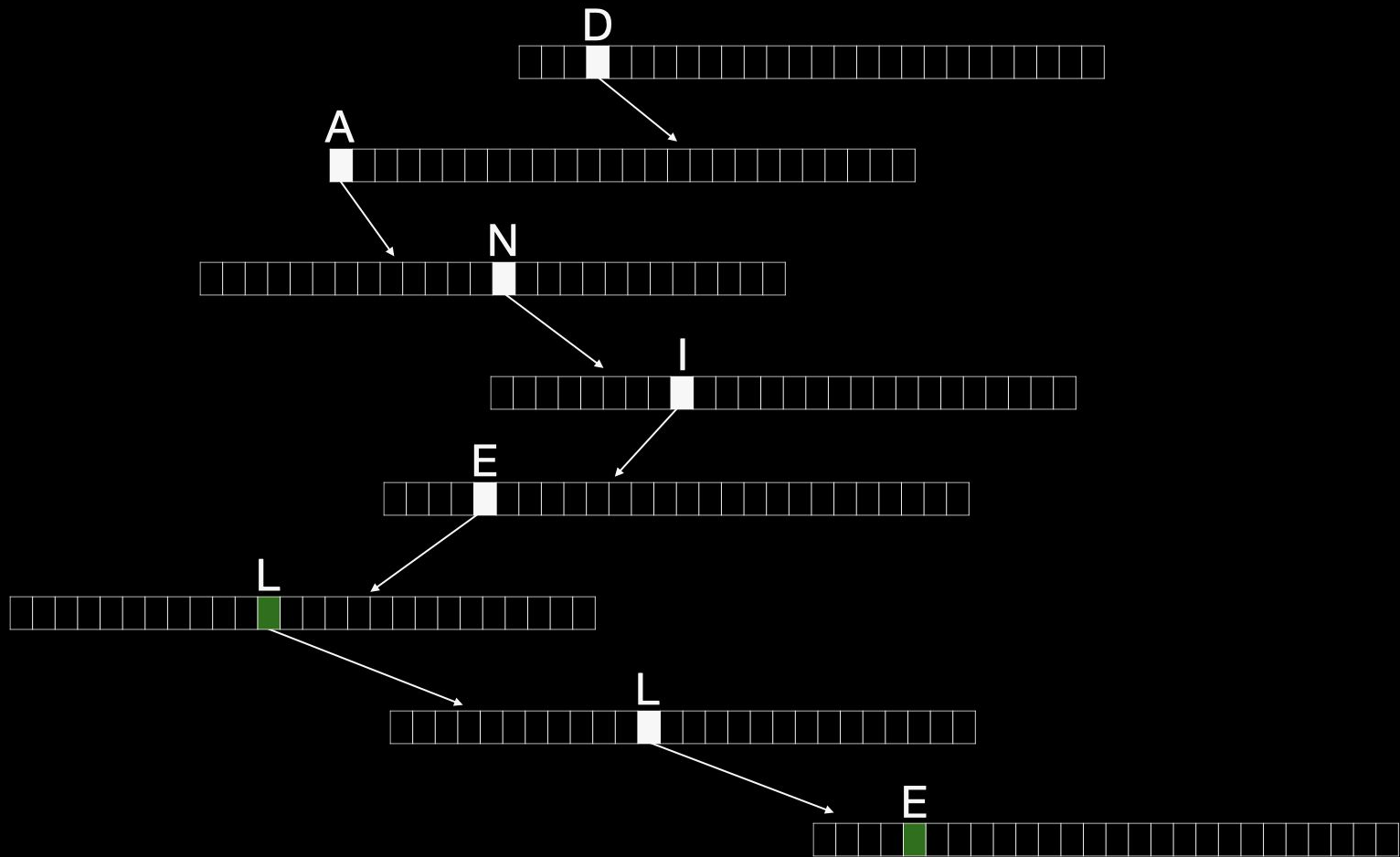






```
typedef struct node
{
    bool is_word;
    struct node *children[SIZE_OF_ALPHABET];
}
node;
```

```
node* trie;
```



$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$       search, insert



abstract data structures

queues

FIFO

enqueue

dequeue

stacks

LIFO

push

pop





dictionaries

PICK ME UP



Small white label on the top shelf.

B

C

D

E

F

G

H

I

K

L

M



N

O

P

Q

R

T

U-

V

W



X

Y

Z

CHOKIN  
FROM 10 ARMS  
VISION STOPS REPAIRING

5 5 5

```
typedef struct node
{
    char ch;
    struct node *next;
}
node;
```

```
node *list;
```

This was

This was C

This was CS

This was CS5



This was CS50