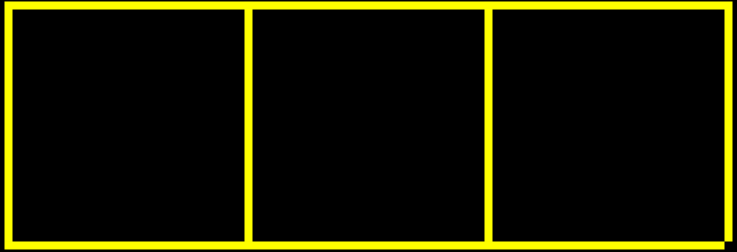
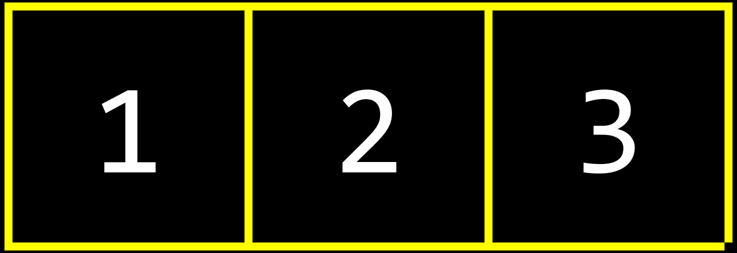
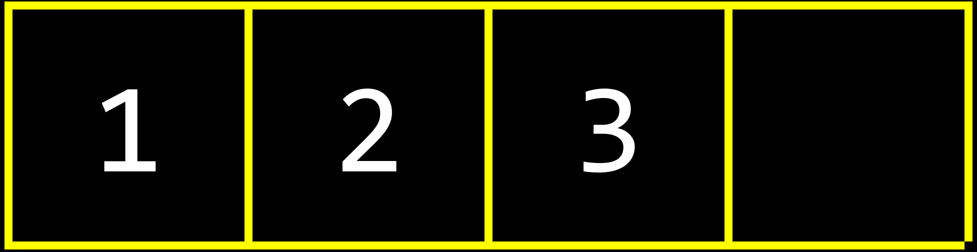
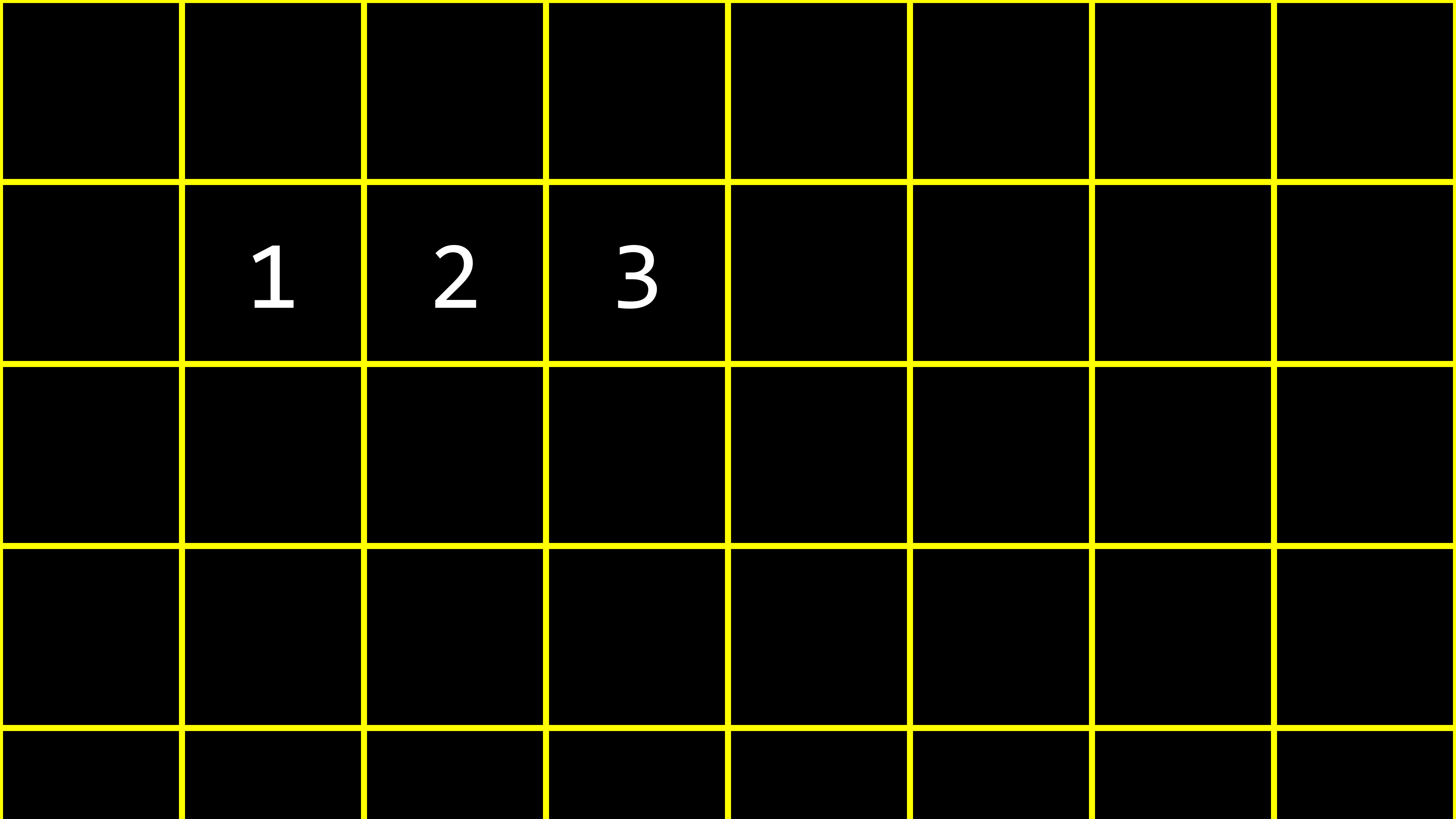
This is CS50

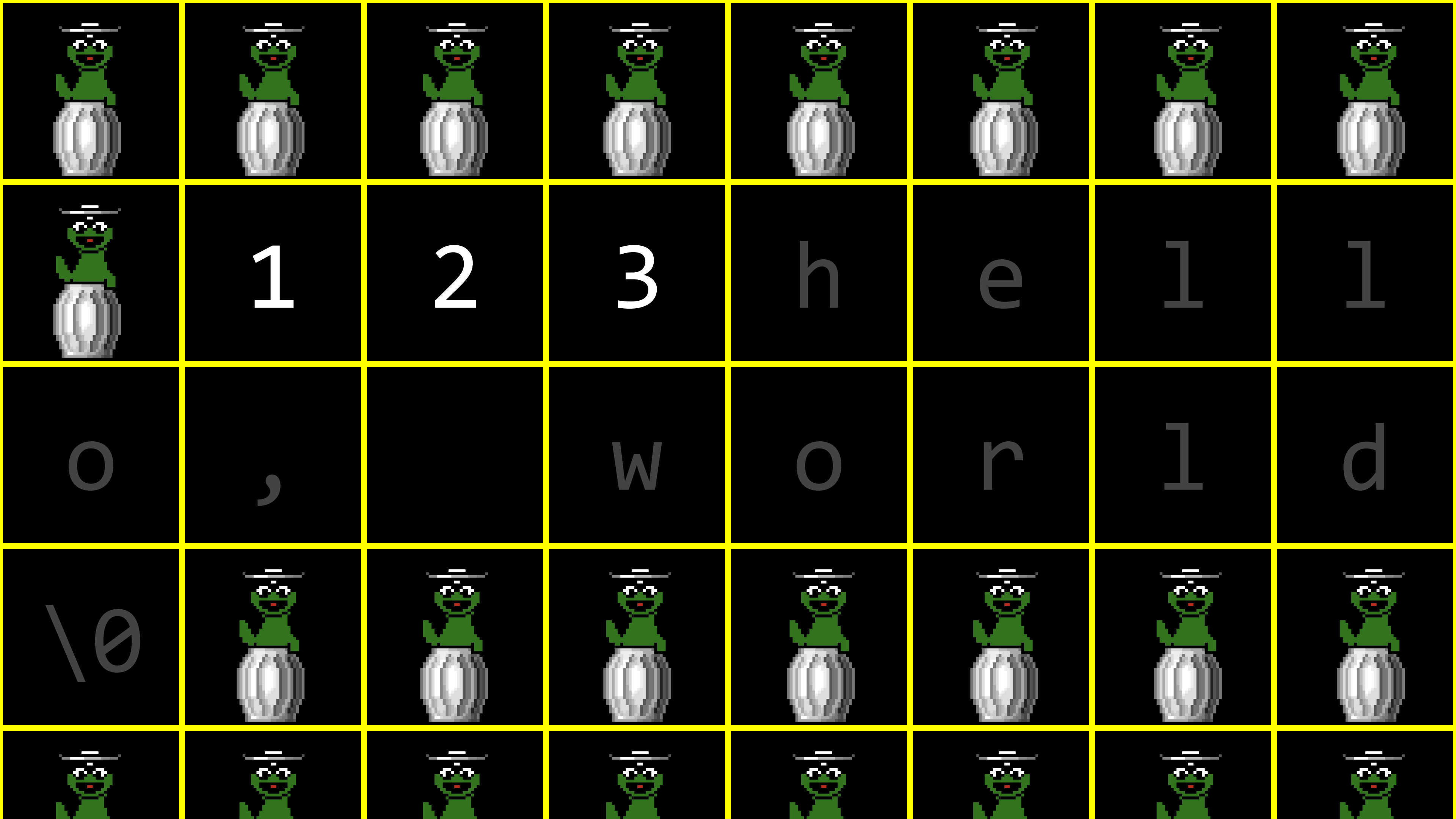
arrays



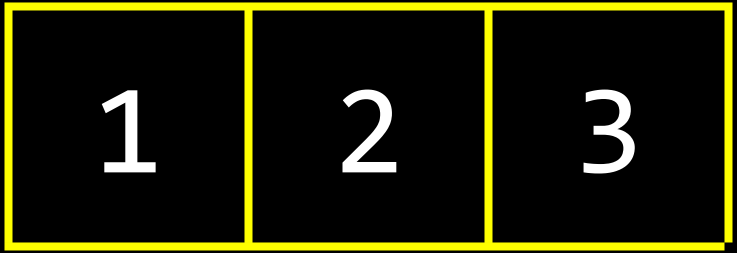

1
2
3

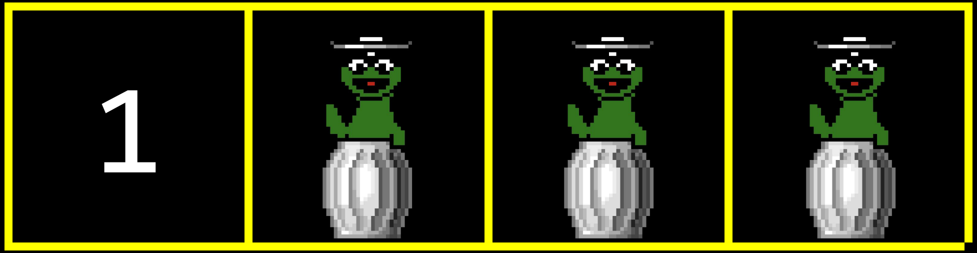

1
2
3

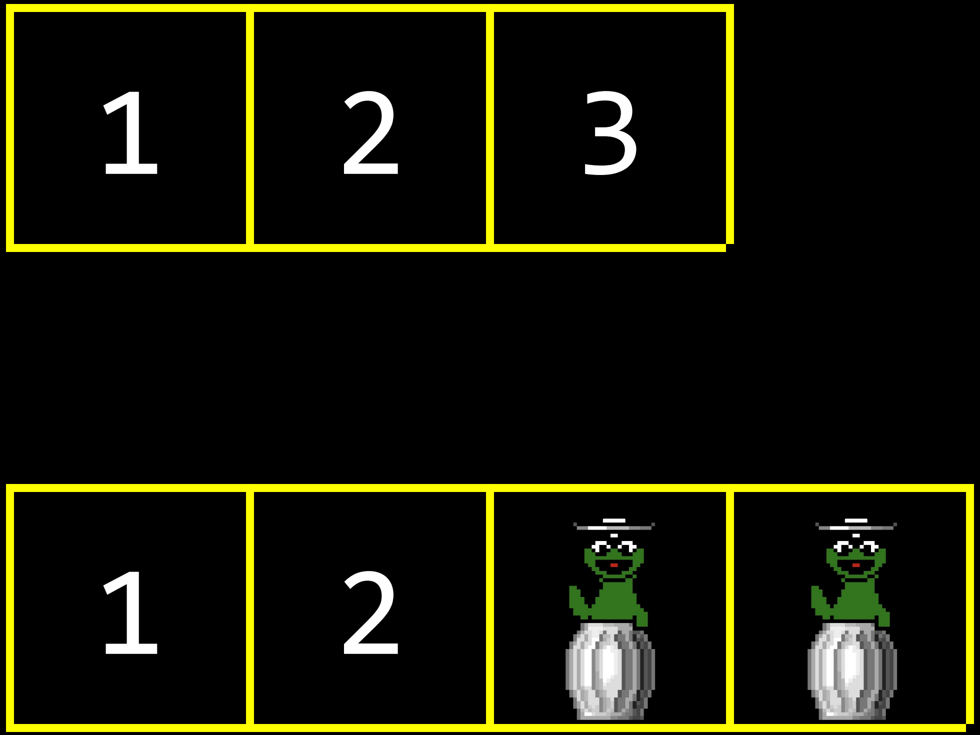

1
2
3

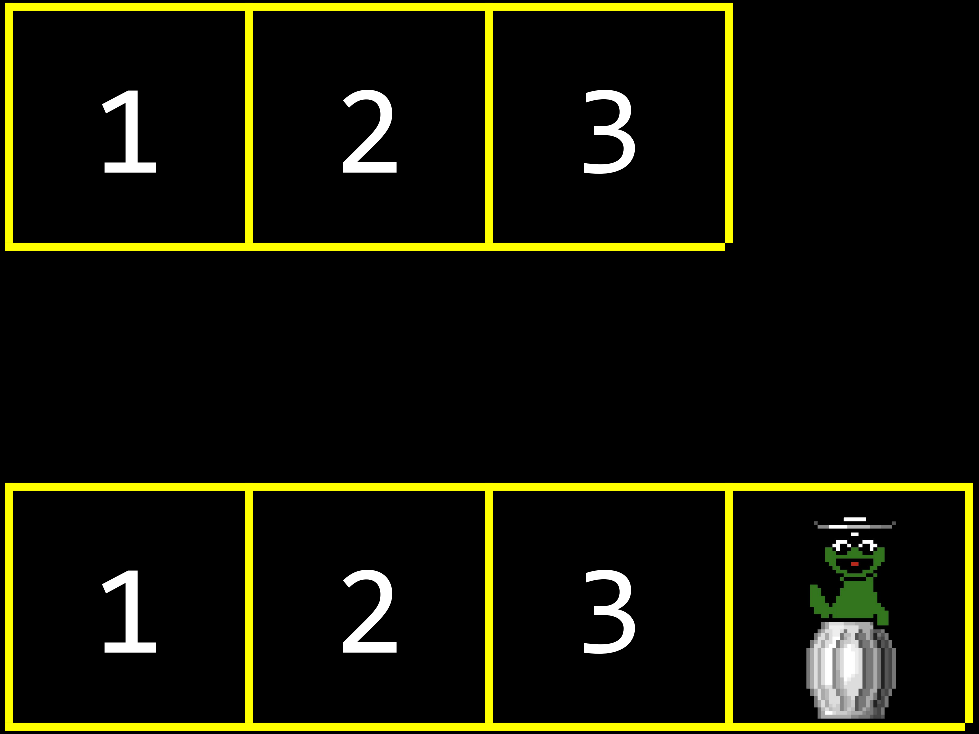

1
2
3
h
e
l
l
o
,
w
o
r
l
d
\0

1	2	3

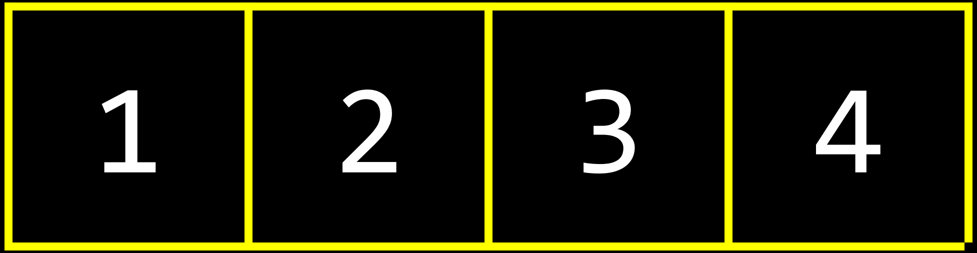

1
2
3


1


1
2
3
1
2


1
2
3
1
2
3

1	2	3	


1
2
3
4

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

$O(n^2)$	
$O(n \log n)$	
$O(n)$	
$O(\log n)$	search
$O(1)$	

$O(n^2)$	
$O(n \log n)$	
$O(n)$	insert
$O(\log n)$	search
$O(1)$	

$\Omega(n^2)$

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

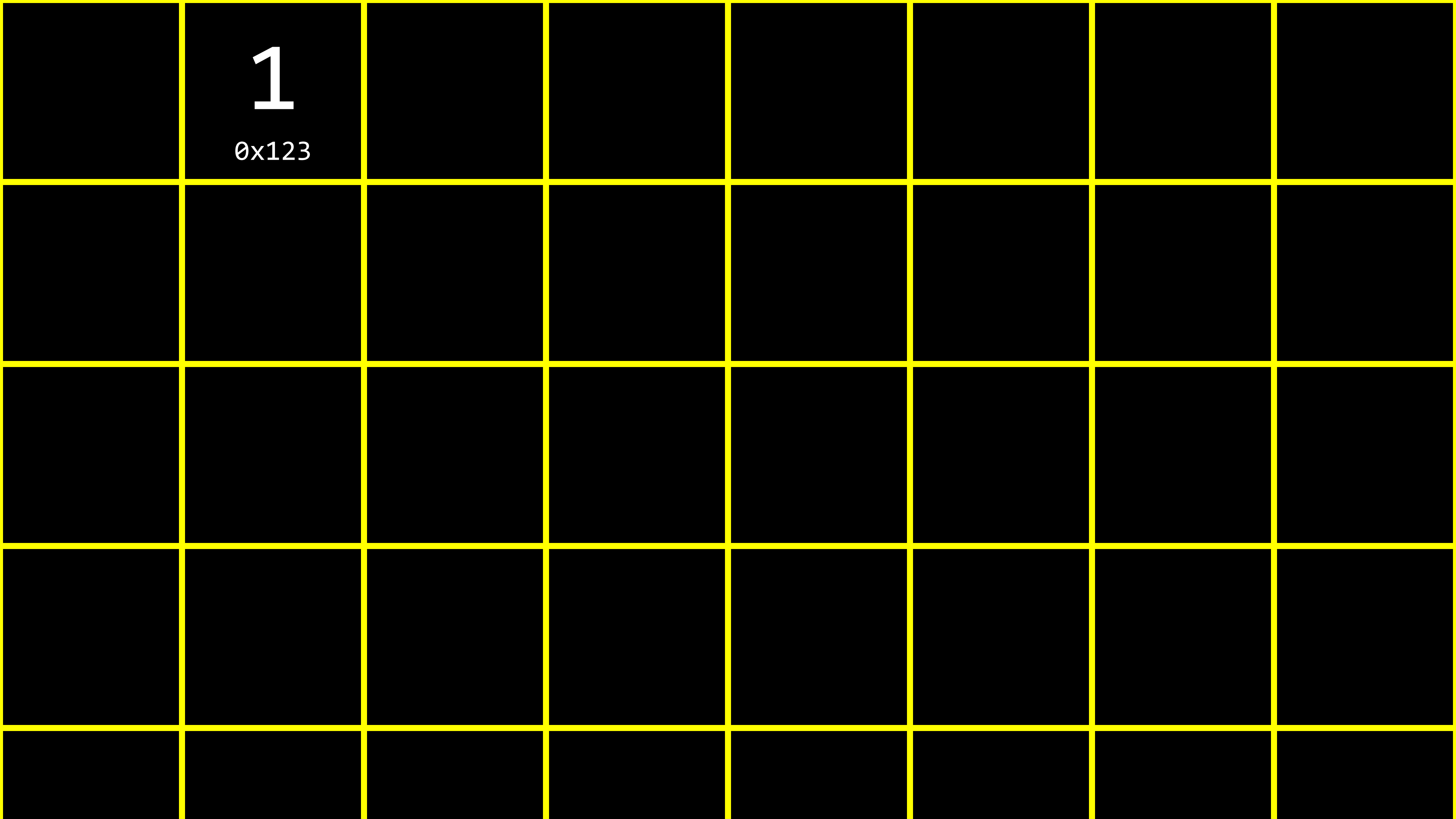
$\Omega(1)$

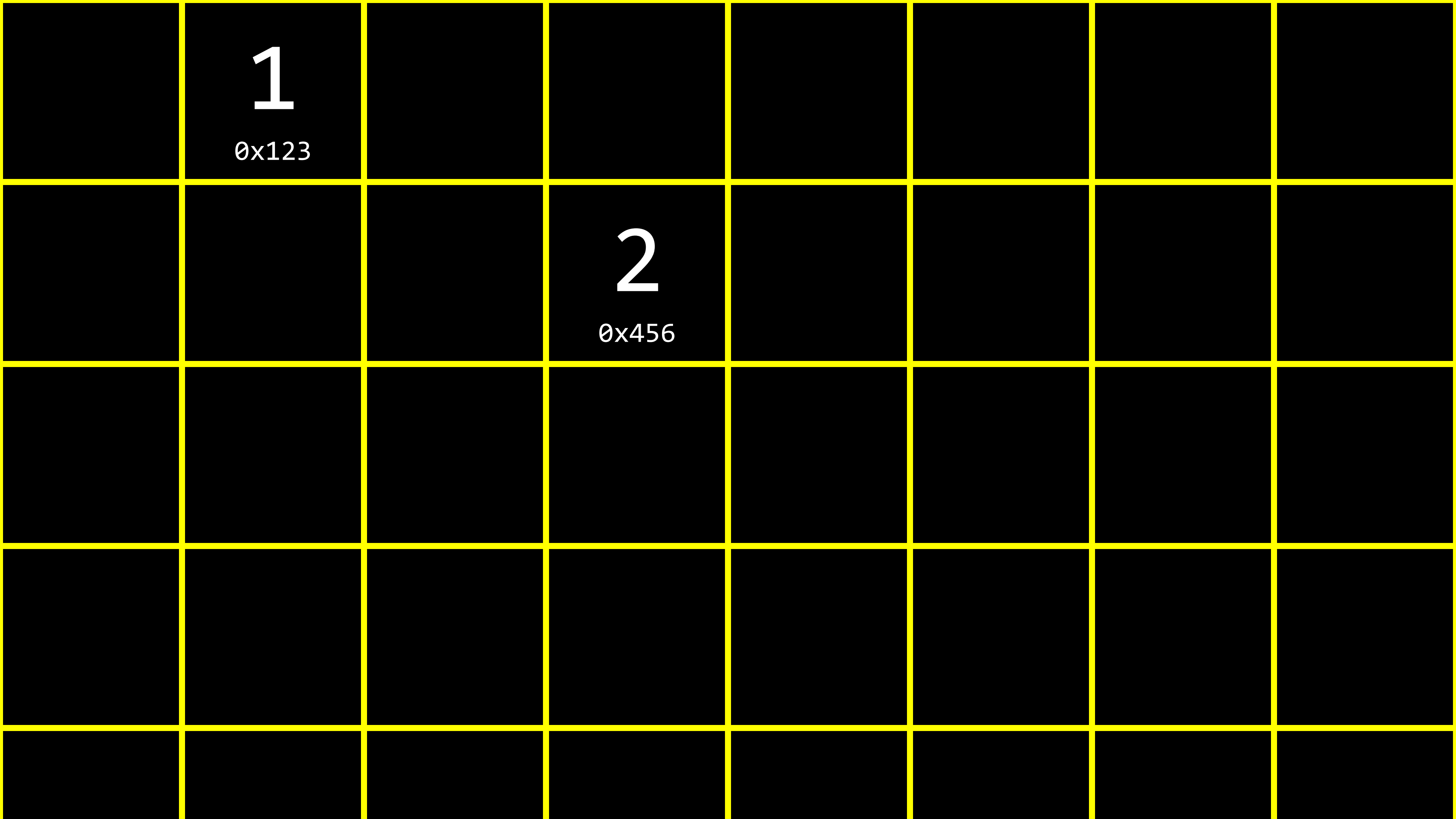
data structures

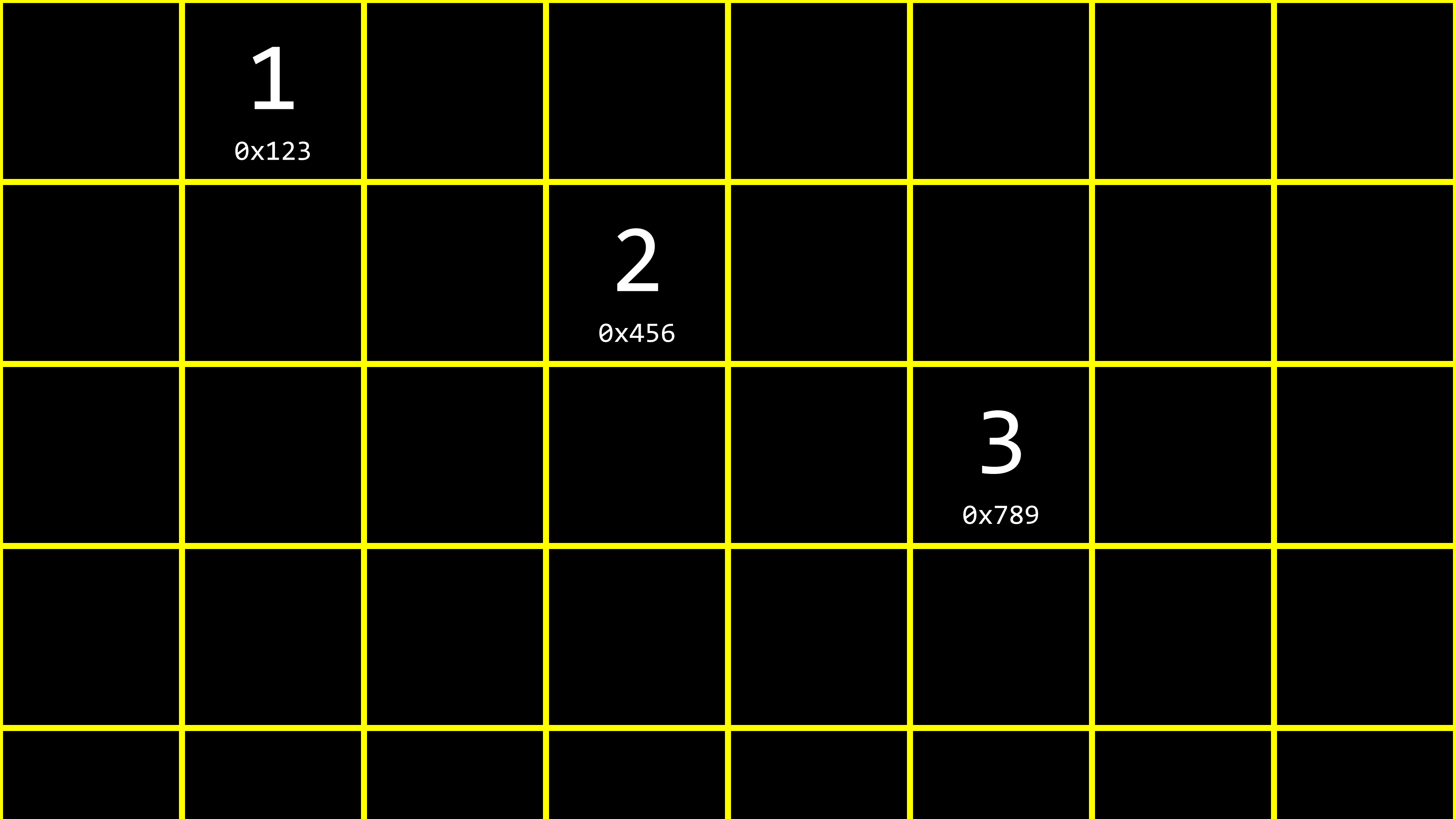
```
struct

.

*
```

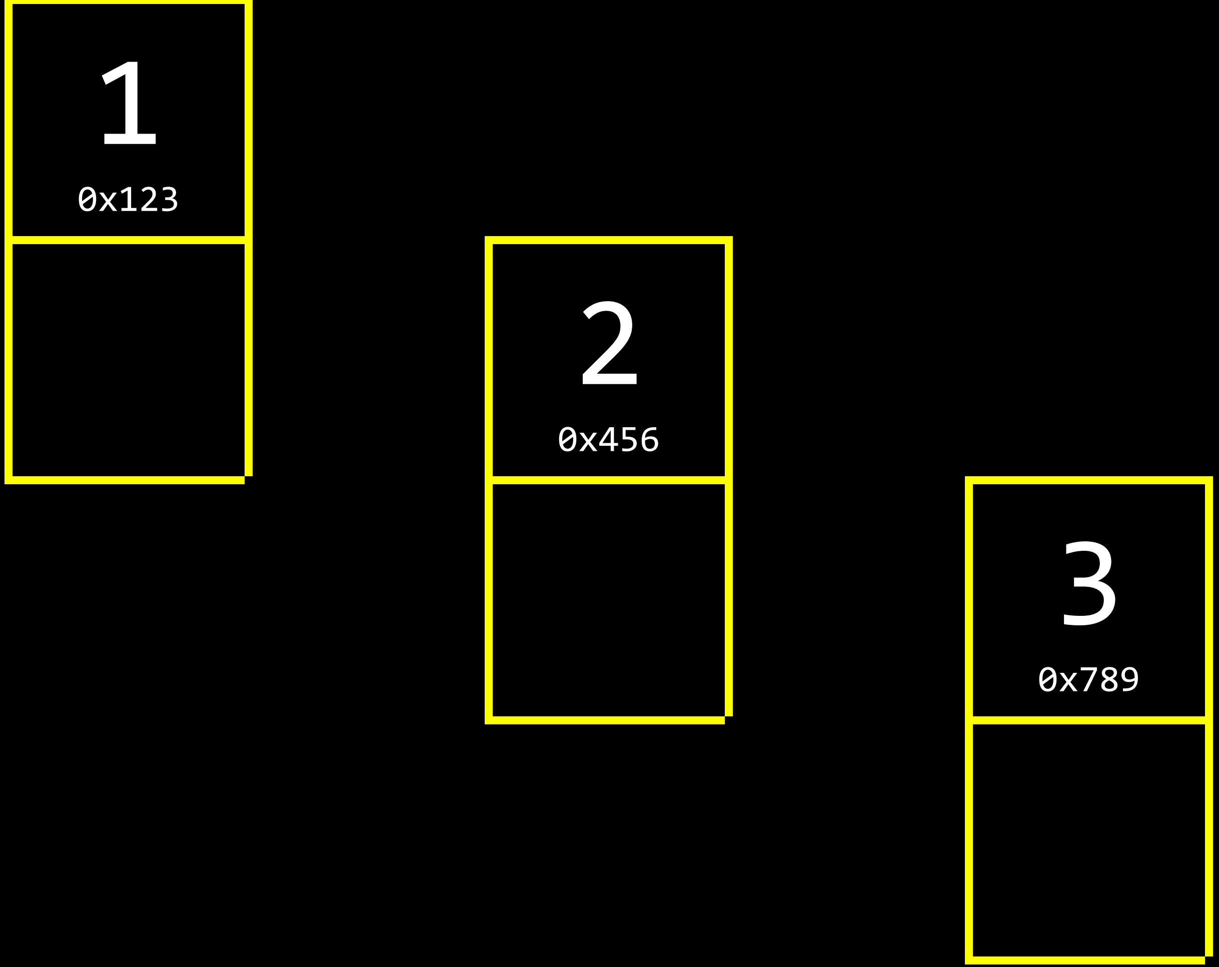
```
struct

->
```

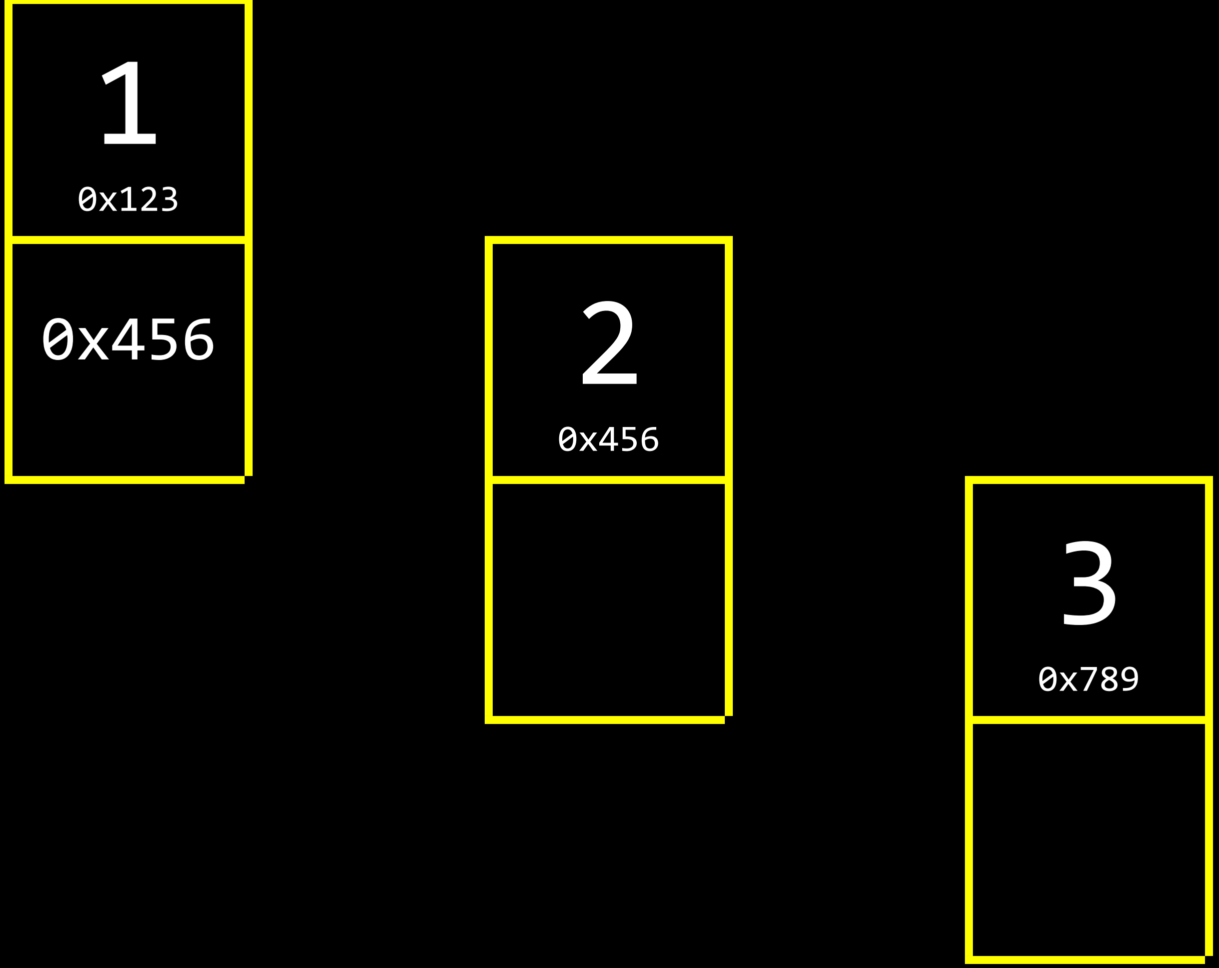
linked lists

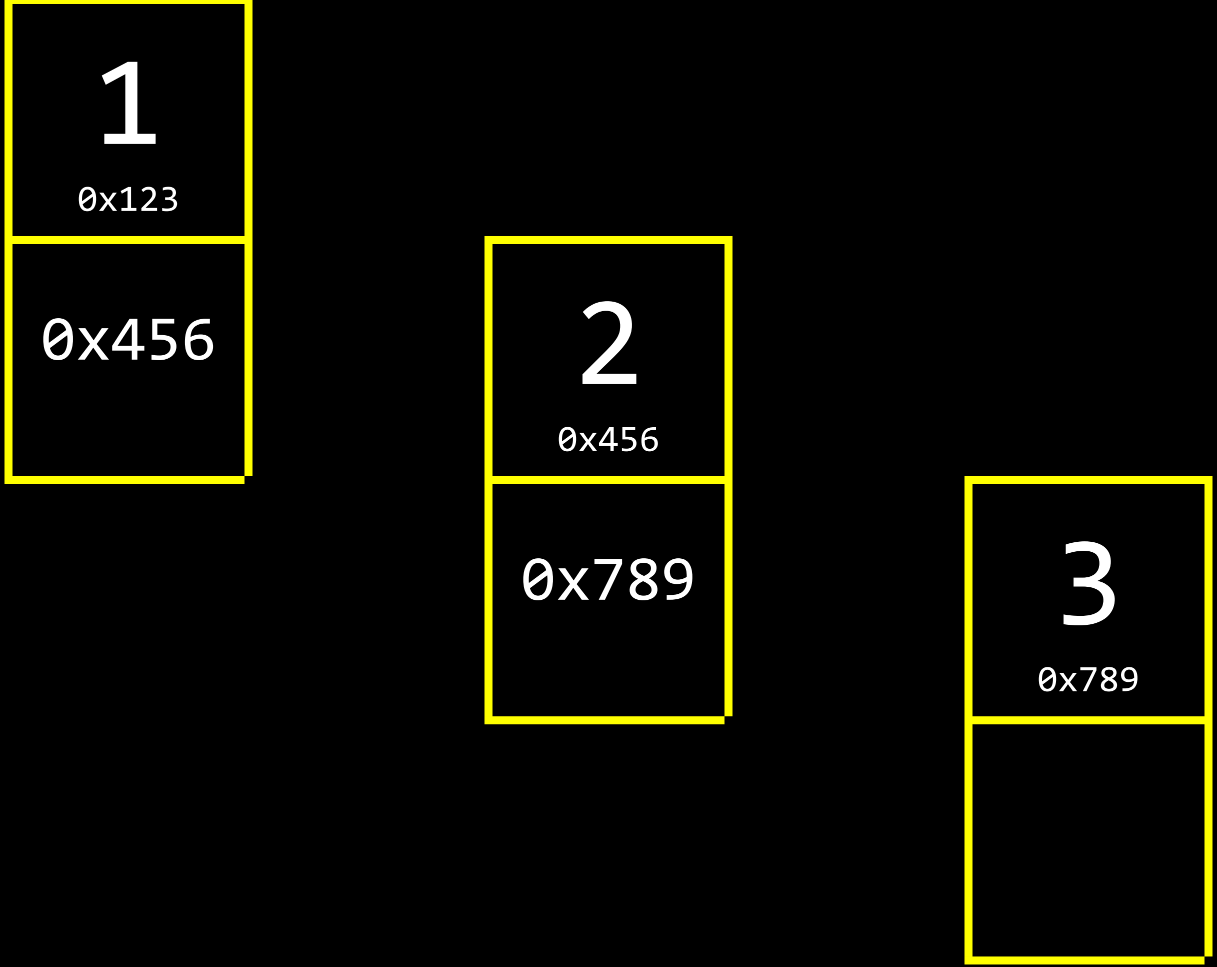

1
0x123

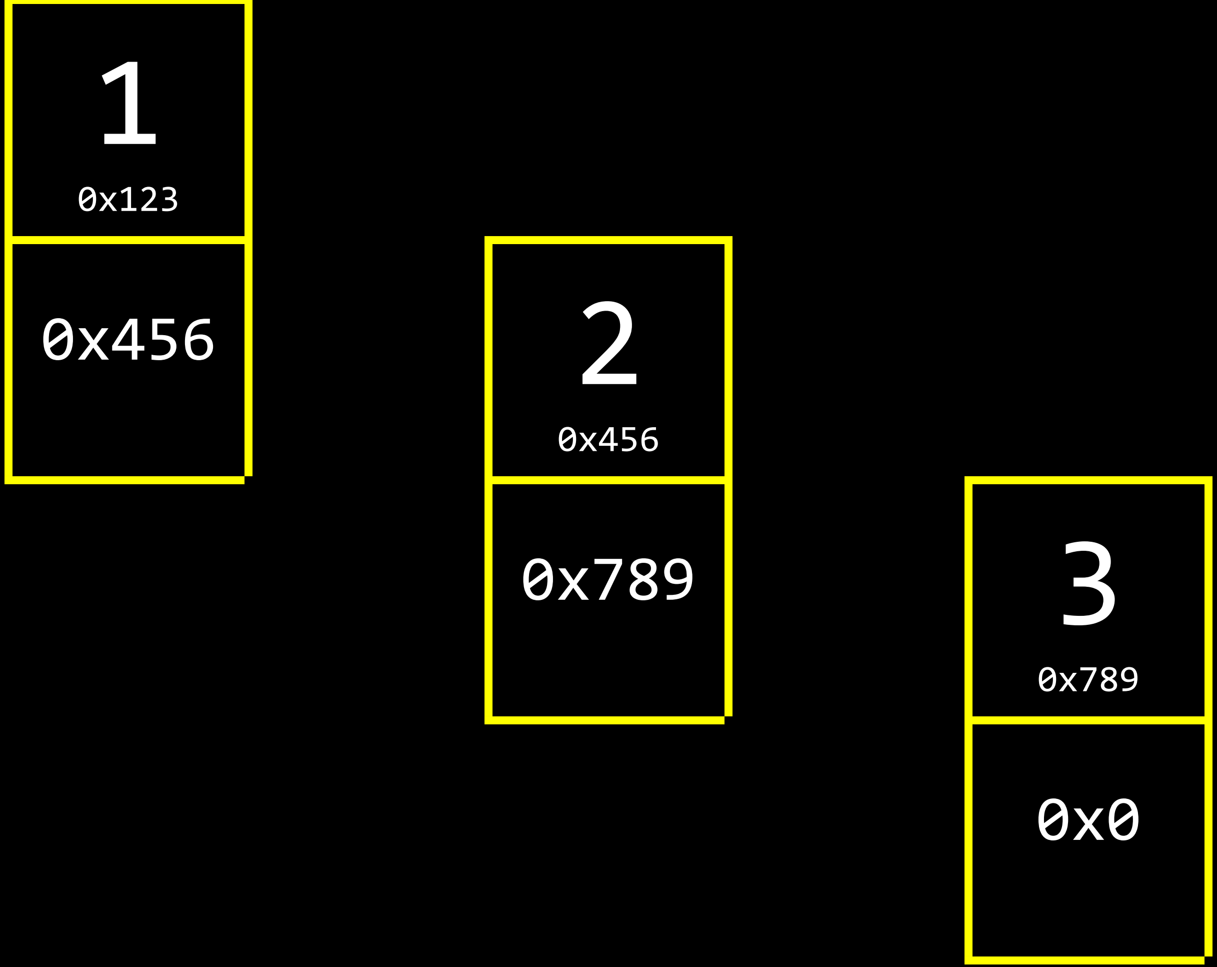

1
0x123
2
0x456

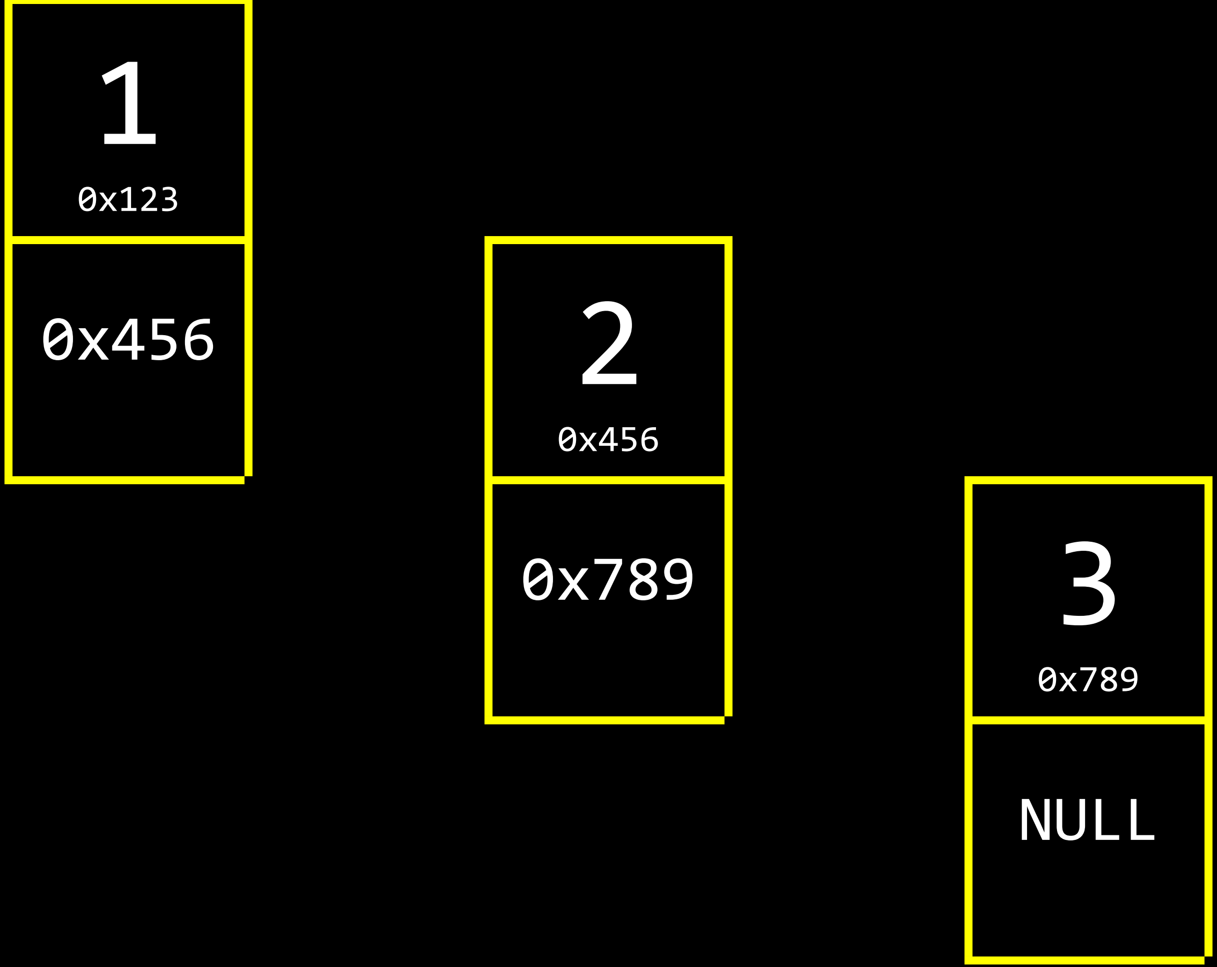

1
0x123
2
0x456
3
0x789

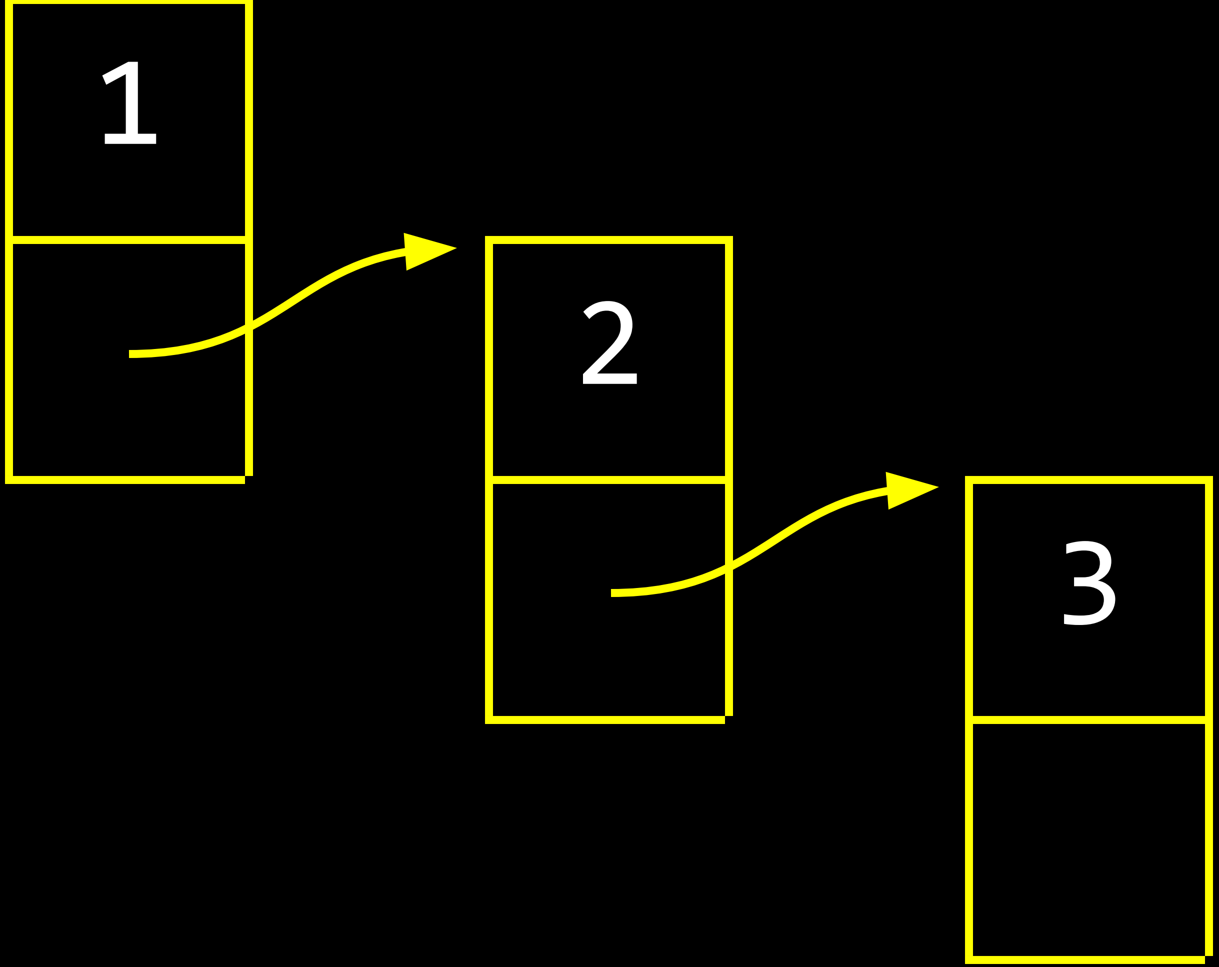

1
0x123
2
0x456
3
0x789


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


1
2
3

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

```
typedef struct
{


}
person;
```

```
typedef struct
{


}
node;
```

```
typedef struct
{
    int number;

}
node;
```

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

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

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

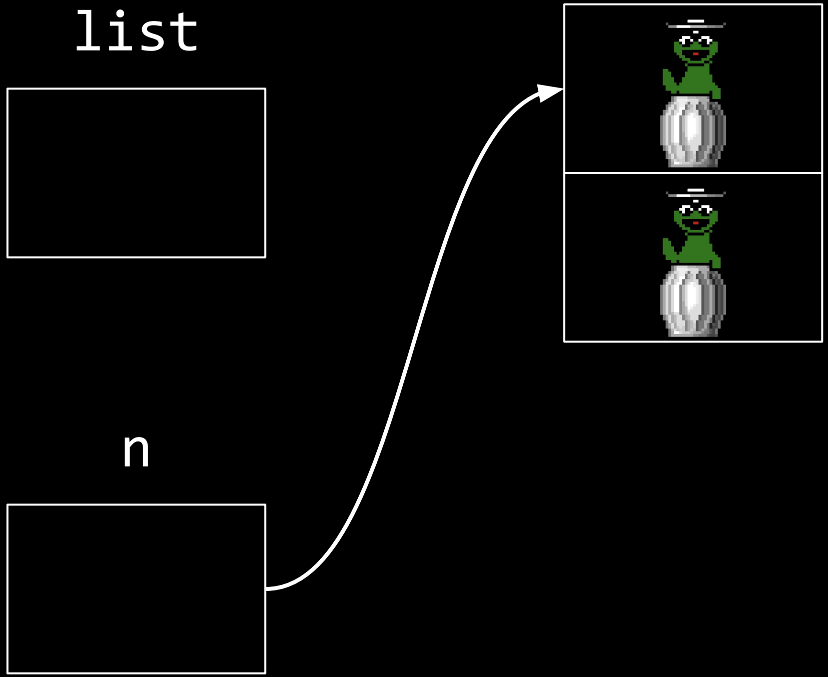
```
node *list;
```

list

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

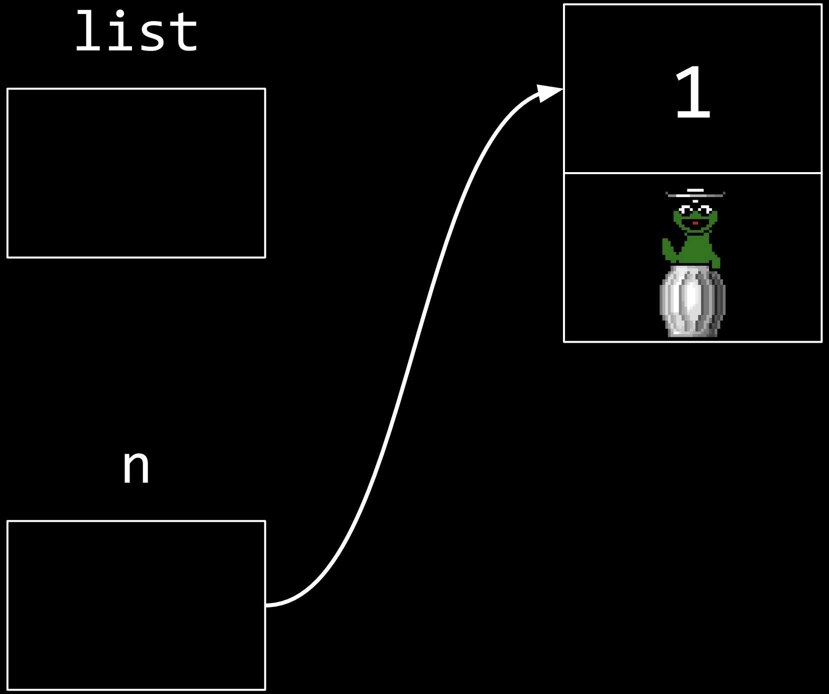
list

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

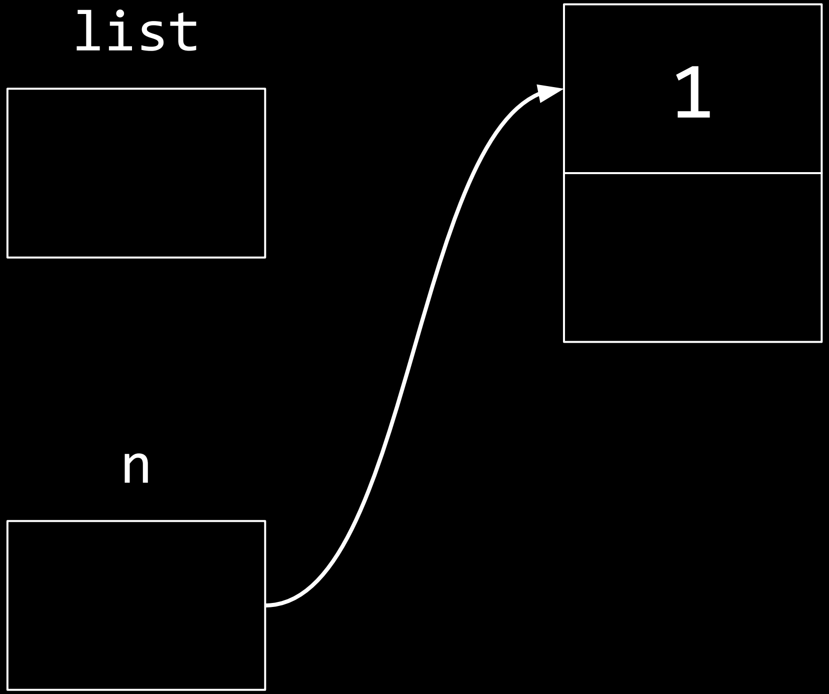

list
n

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

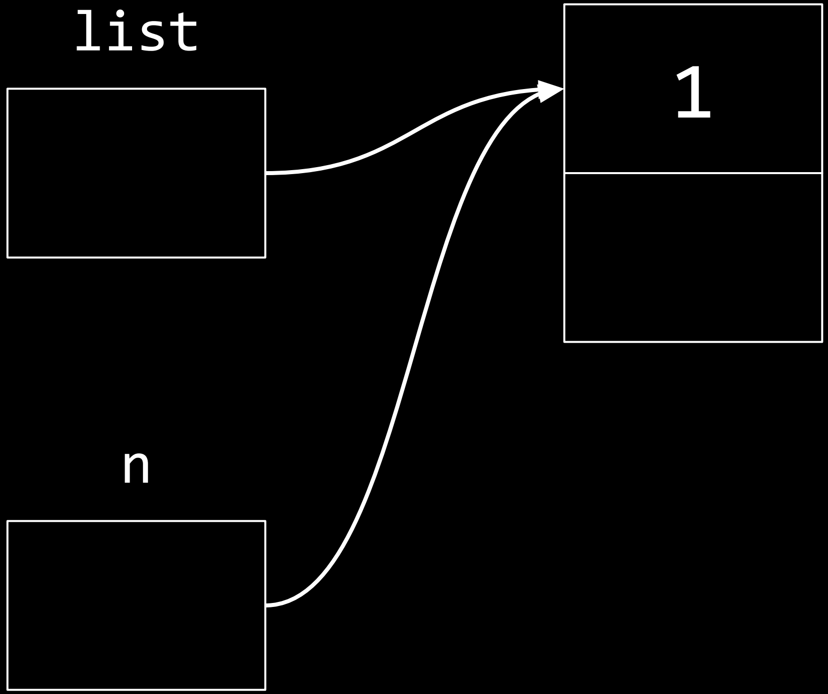
```
if (n != NULL)
{
    n->number = 1;
}
```

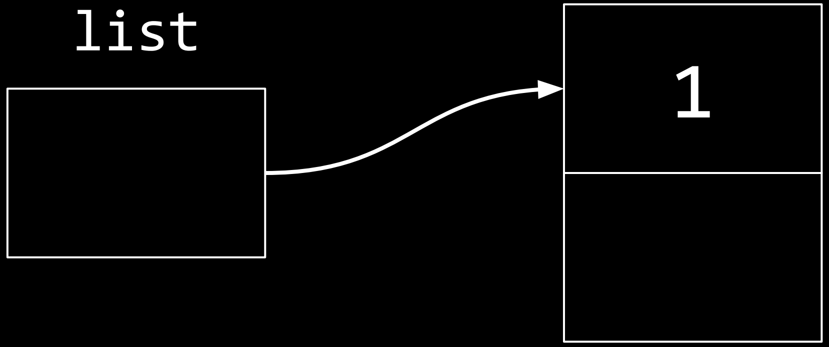

list
n
1

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

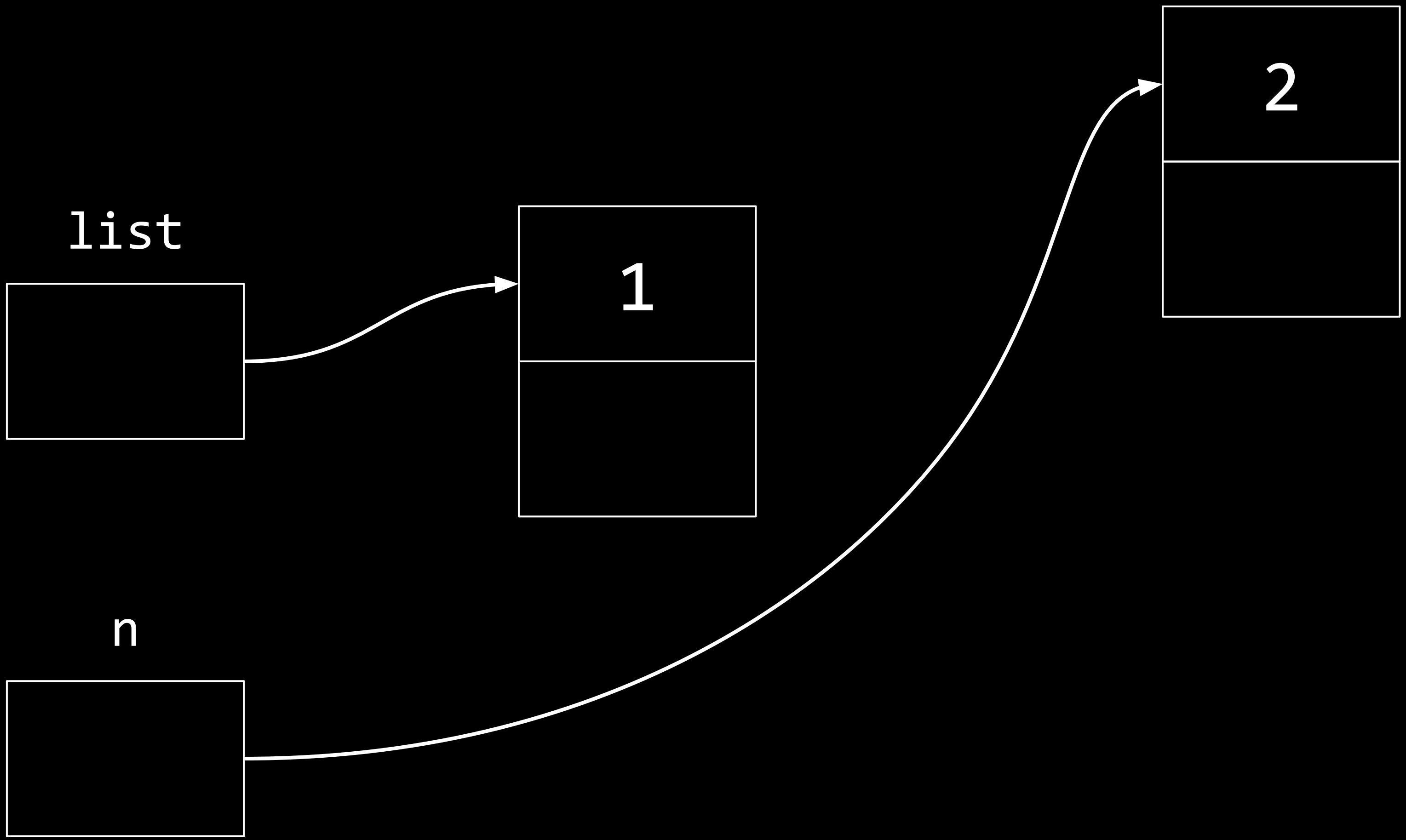

list
1
n

```
list = n;
```

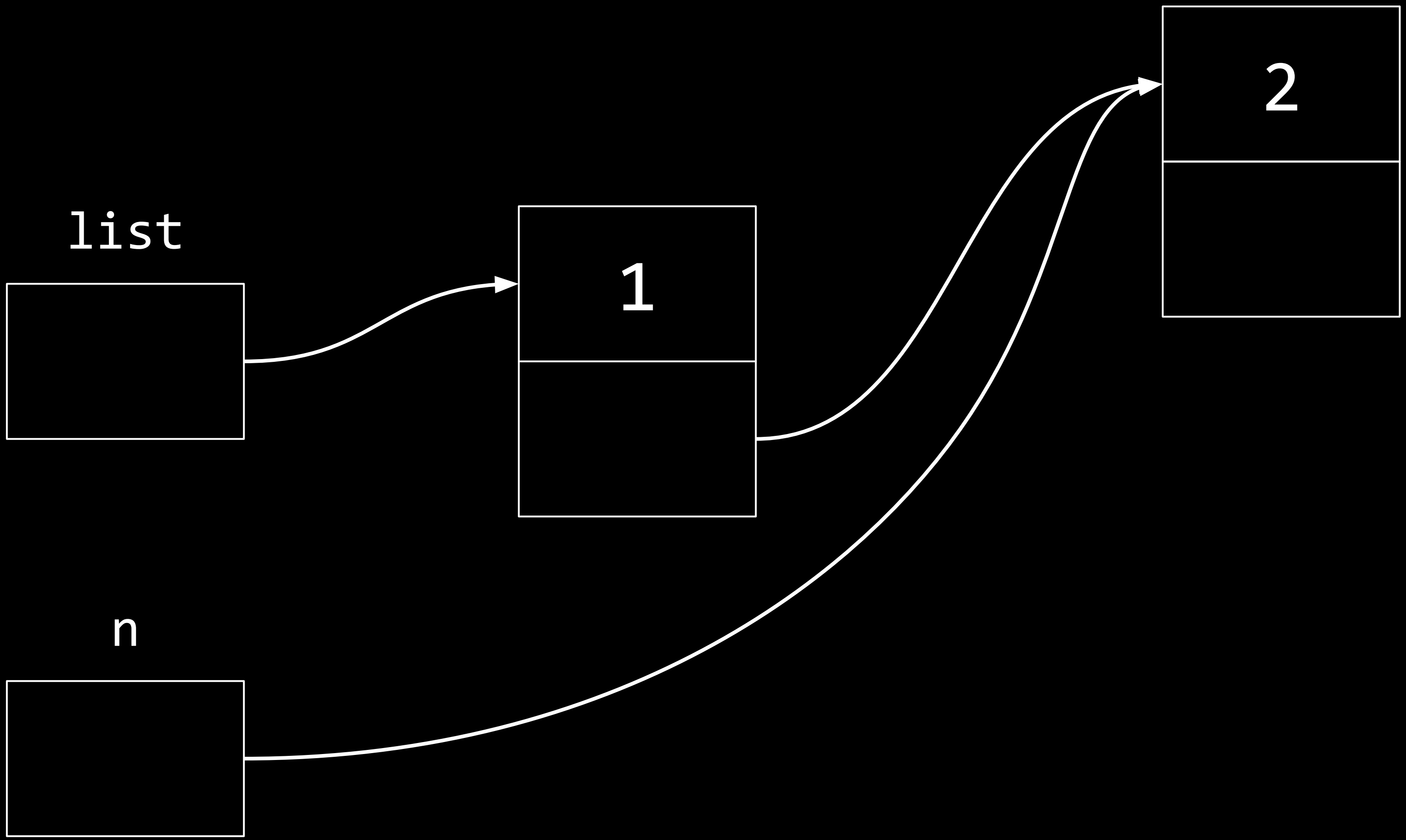

list
n
1

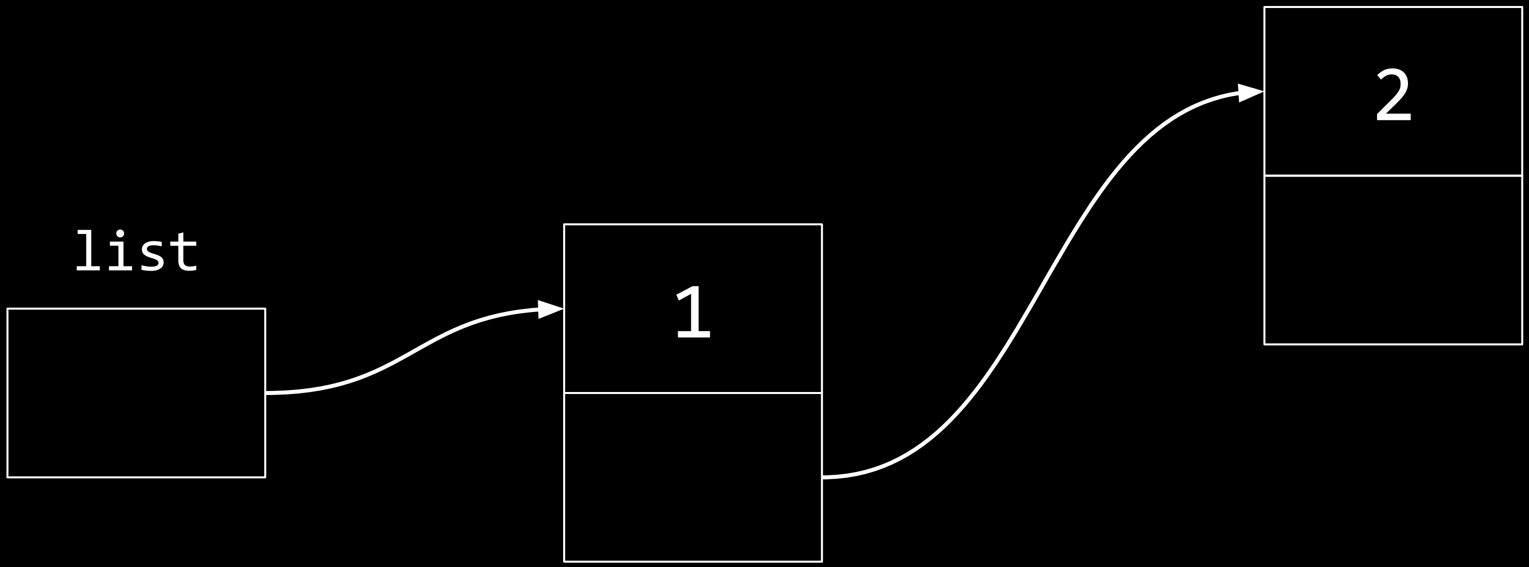

list
1

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

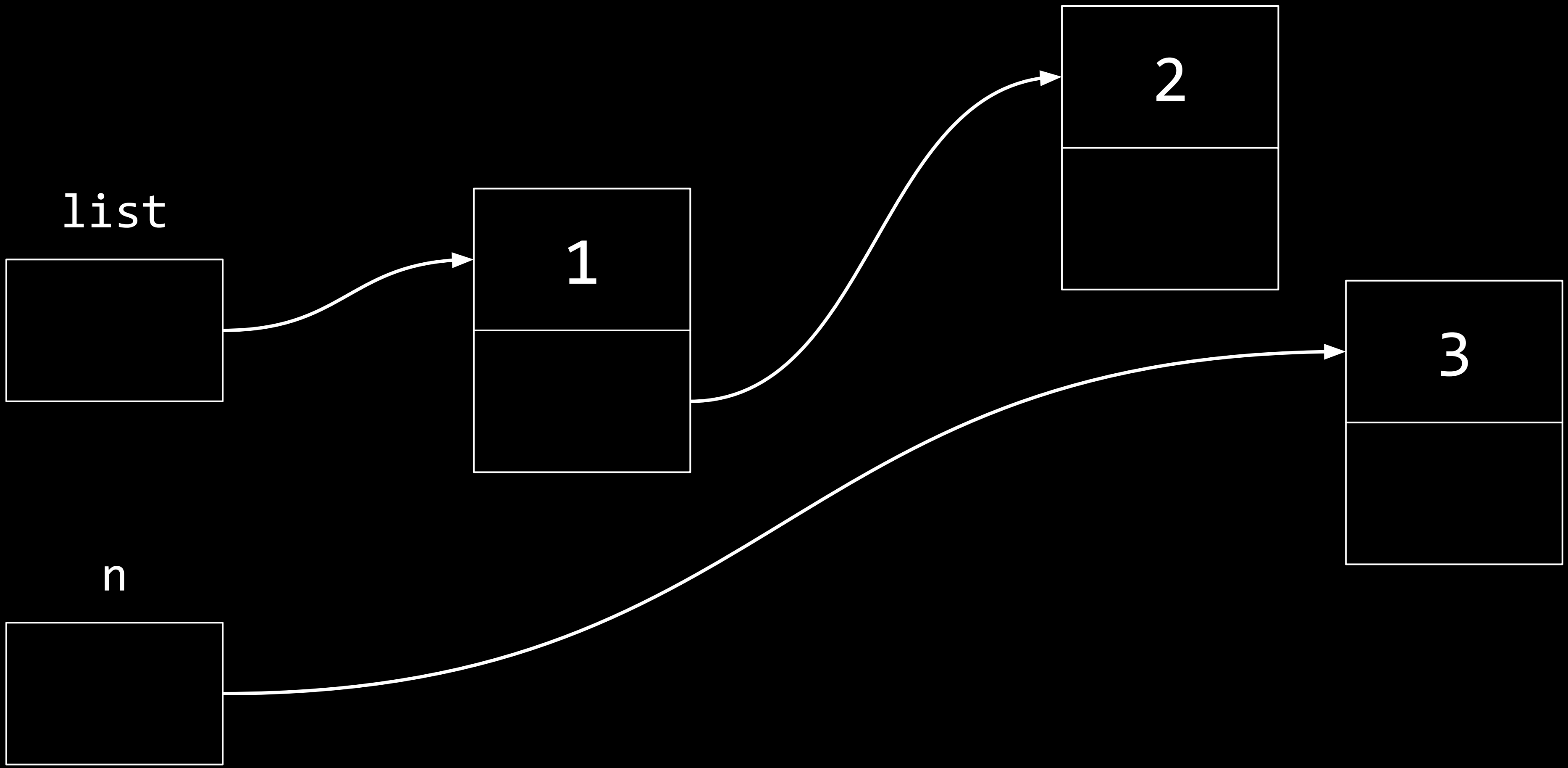

list
1
2
n

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

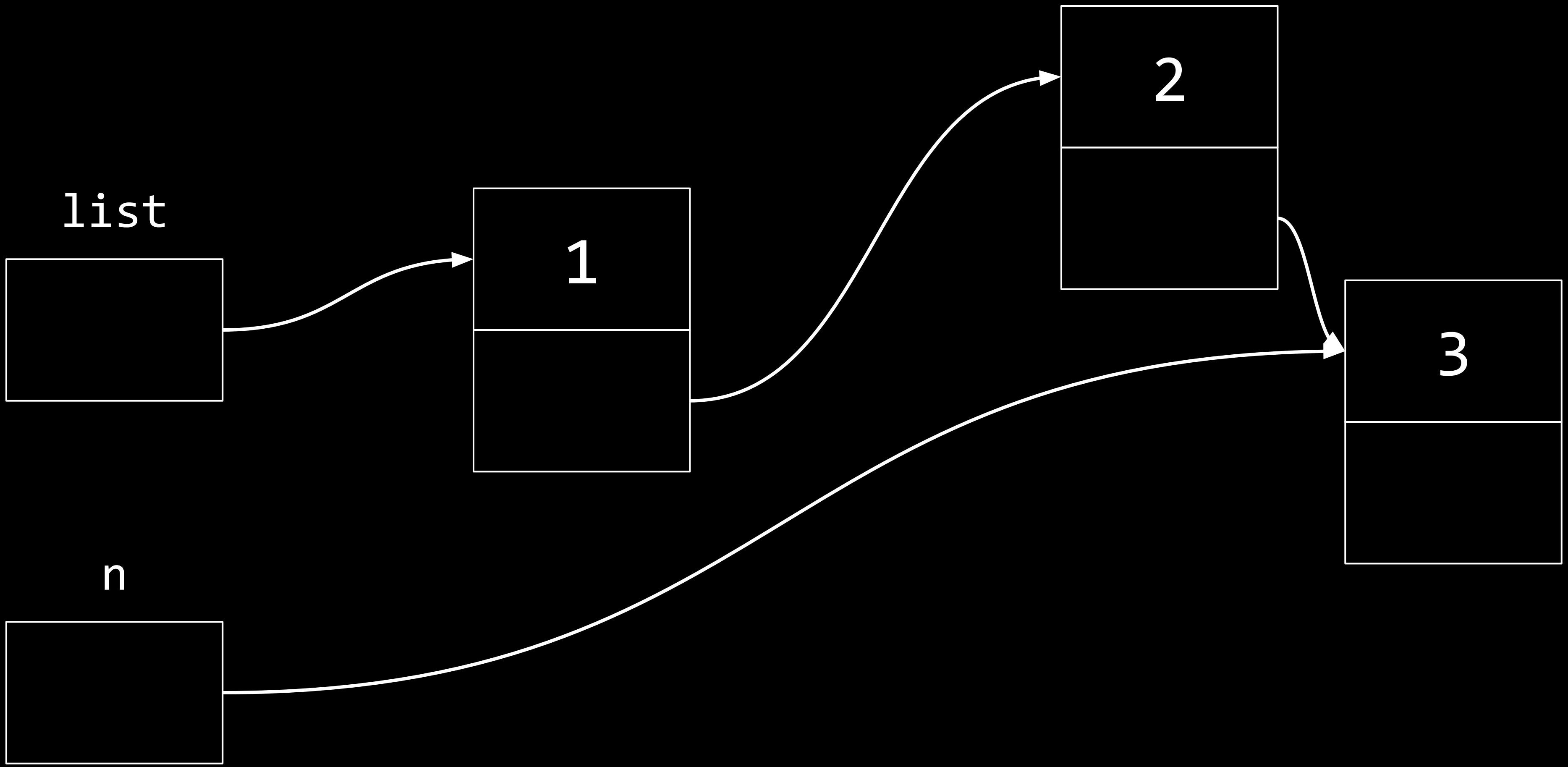

list
n
1
2

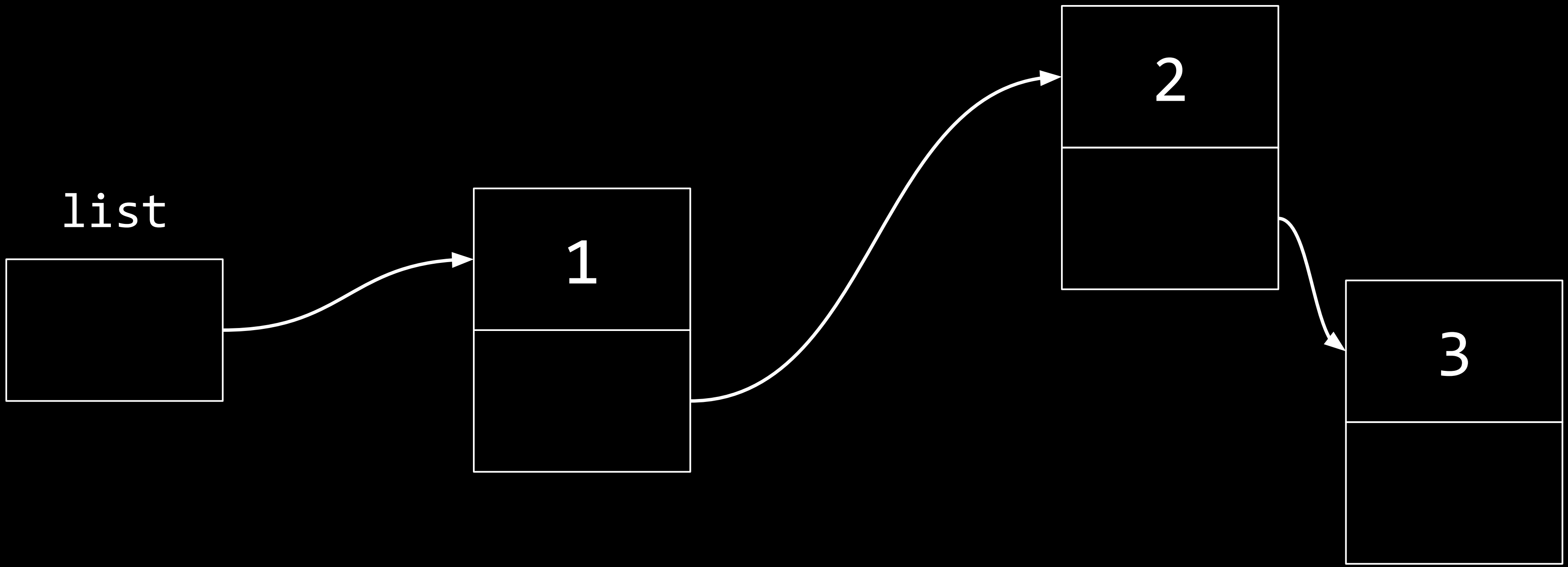

list
1
2

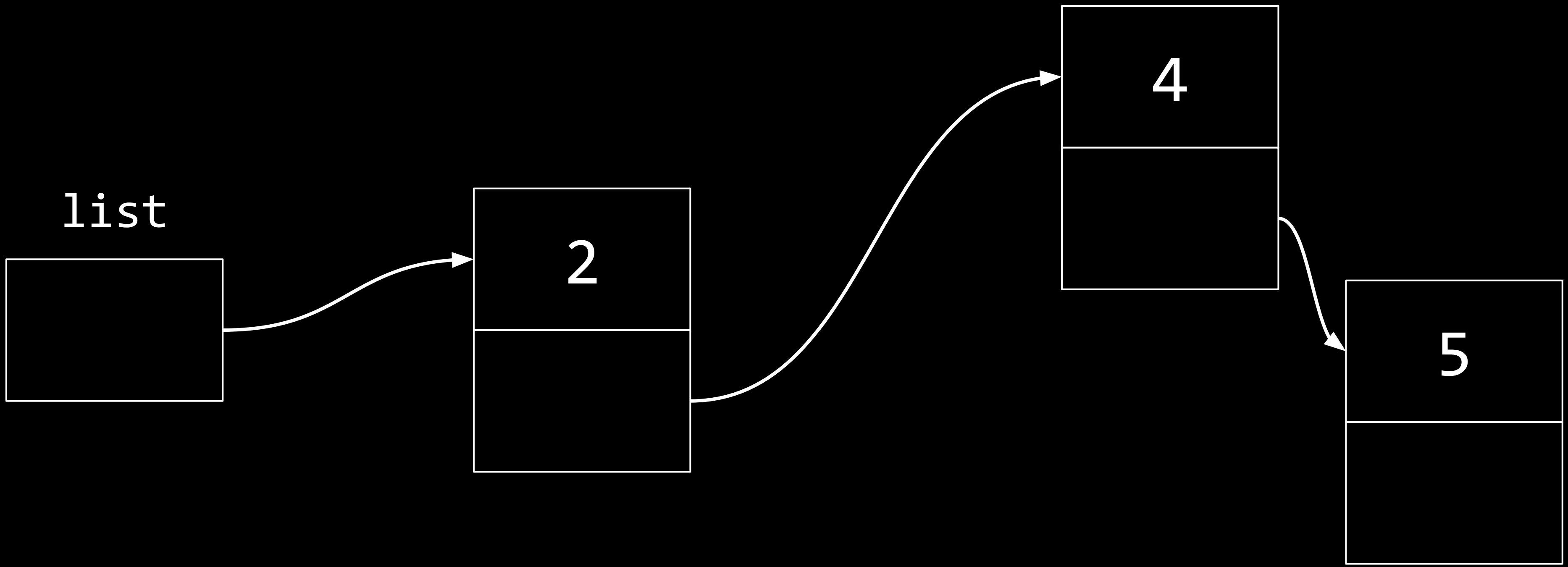
```
node *n = malloc(sizeof(node));
if (n != NULL)
{
    n->number = 3;
    n->next = NULL;
}
```


list
1
2
3
n

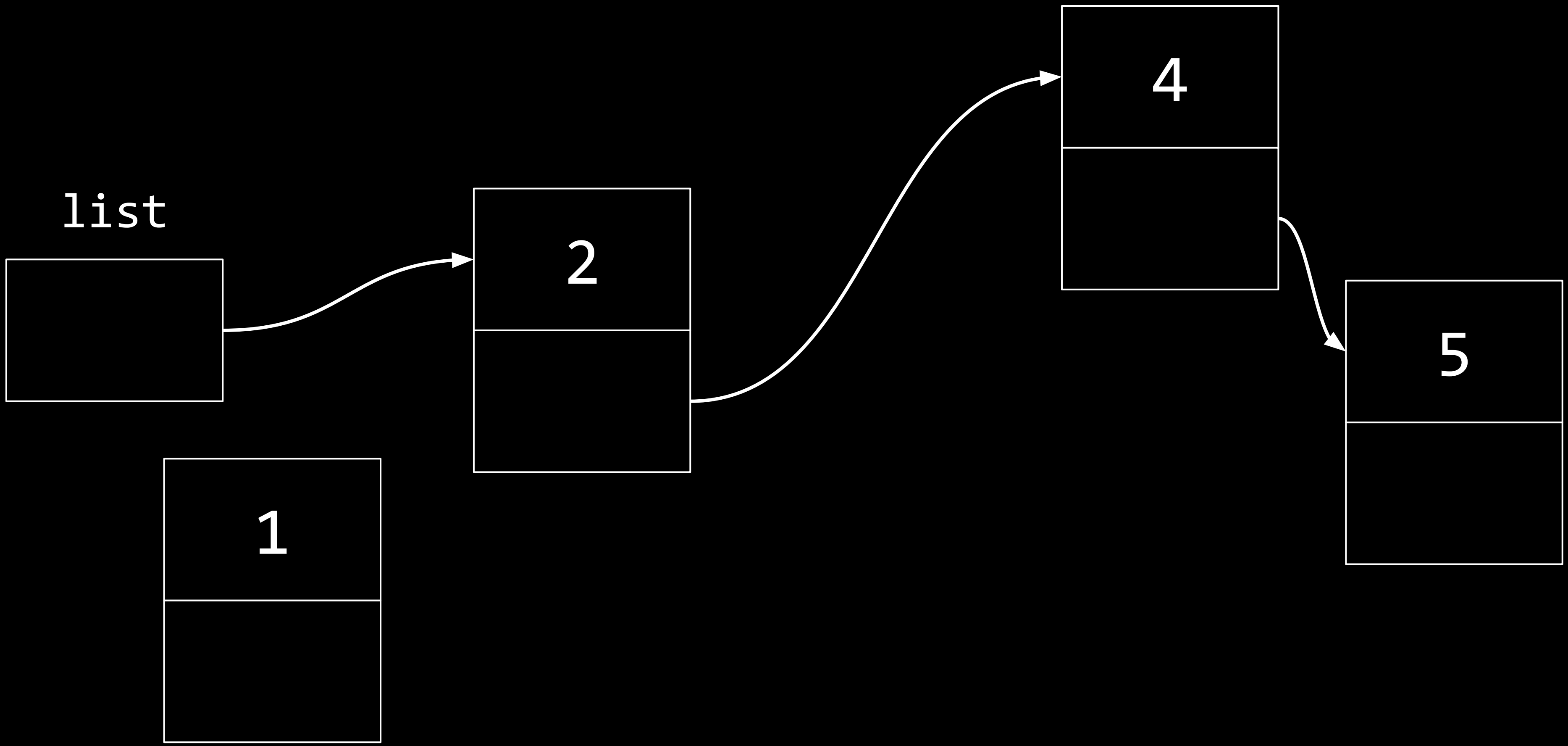
```
list->next->next = n;
```


list
n
1
2
3

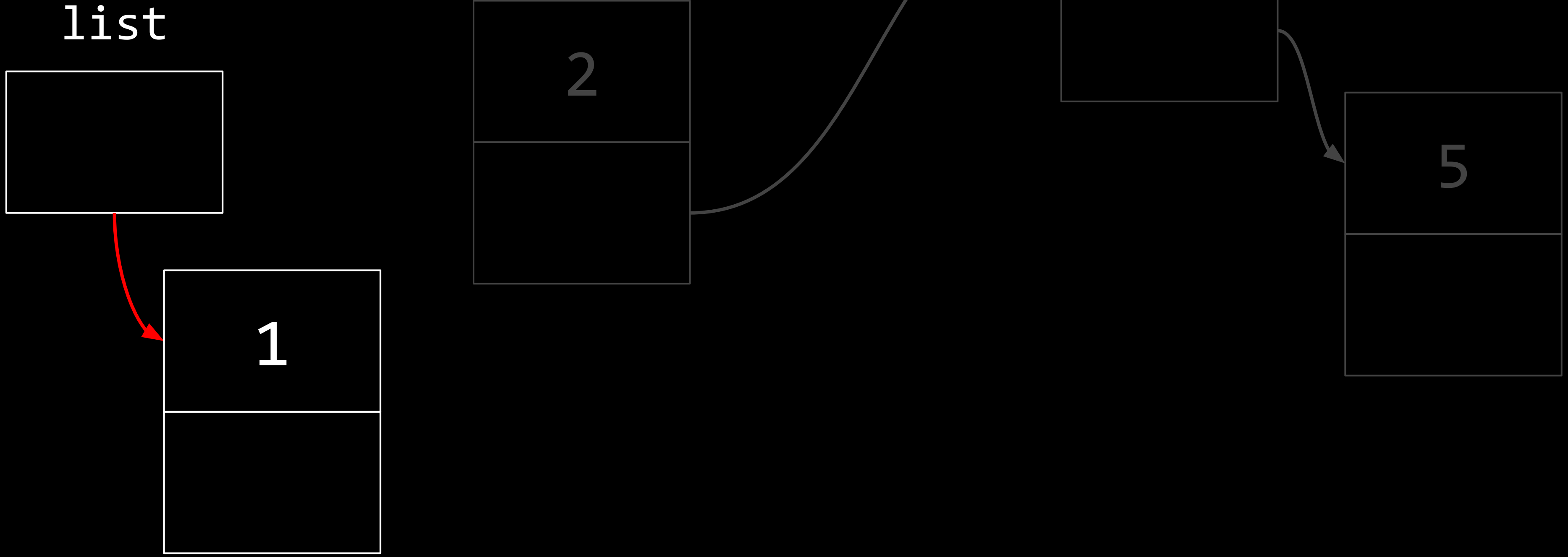

list
1
2
3


list
2
4
5

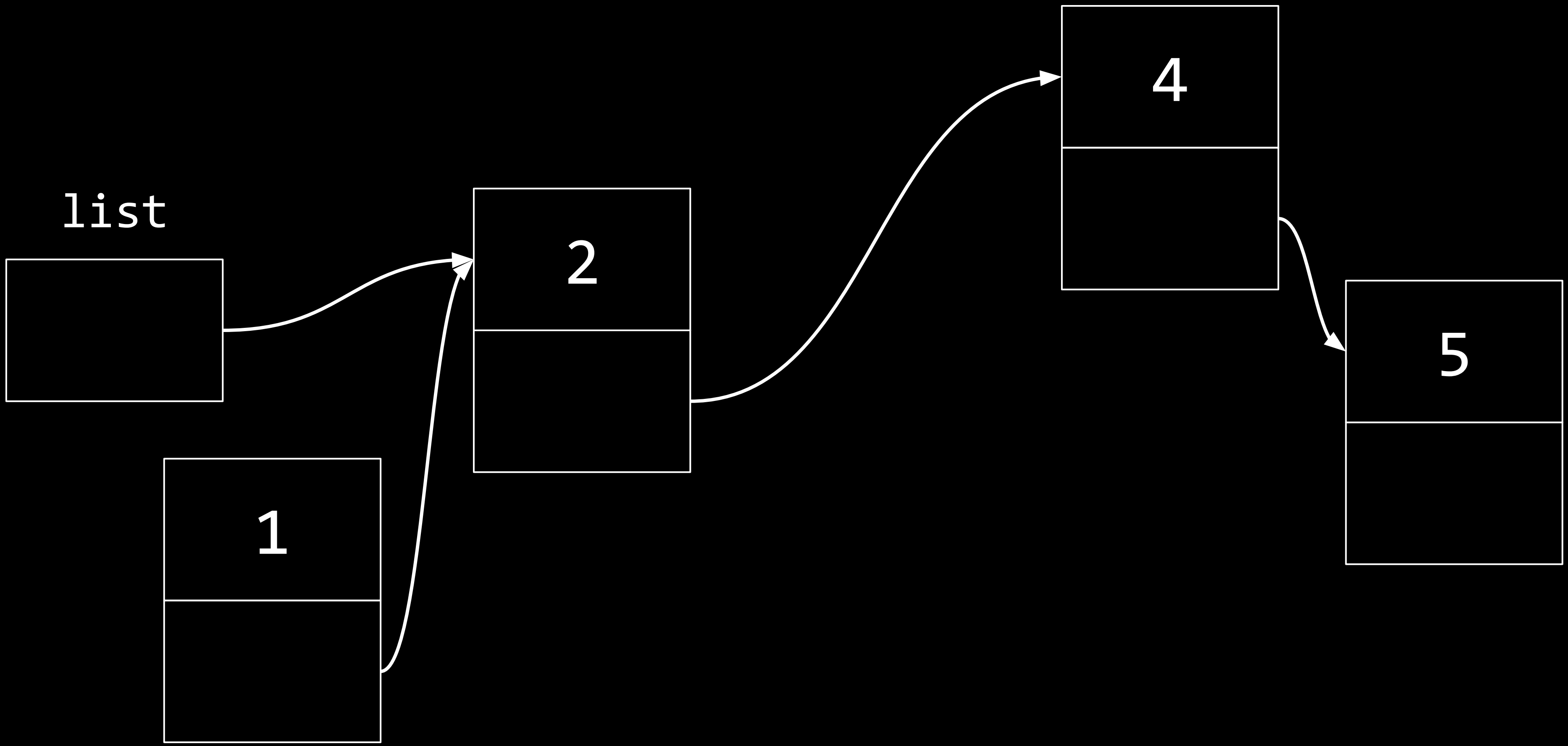
```
node *n = malloc(sizeof(node));
if (n != NULL)
{
    n->number = 1;
    n->next = NULL;
}
```


list
2
4
5
1

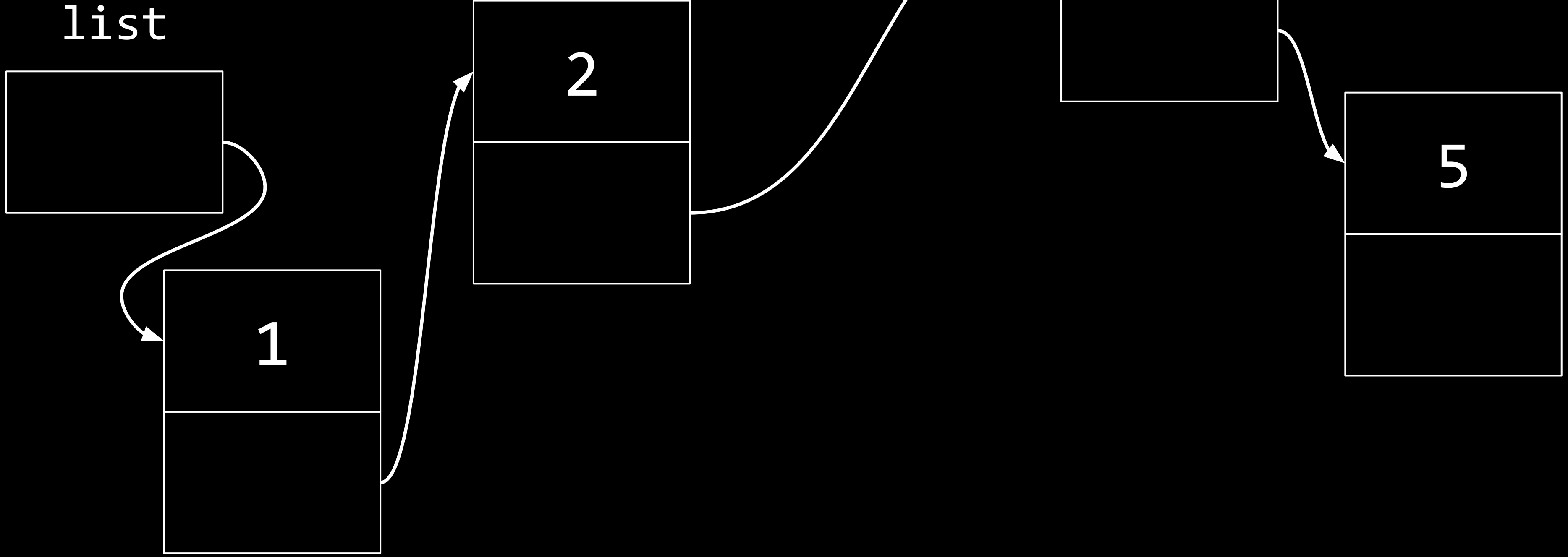
```
list = n;
```

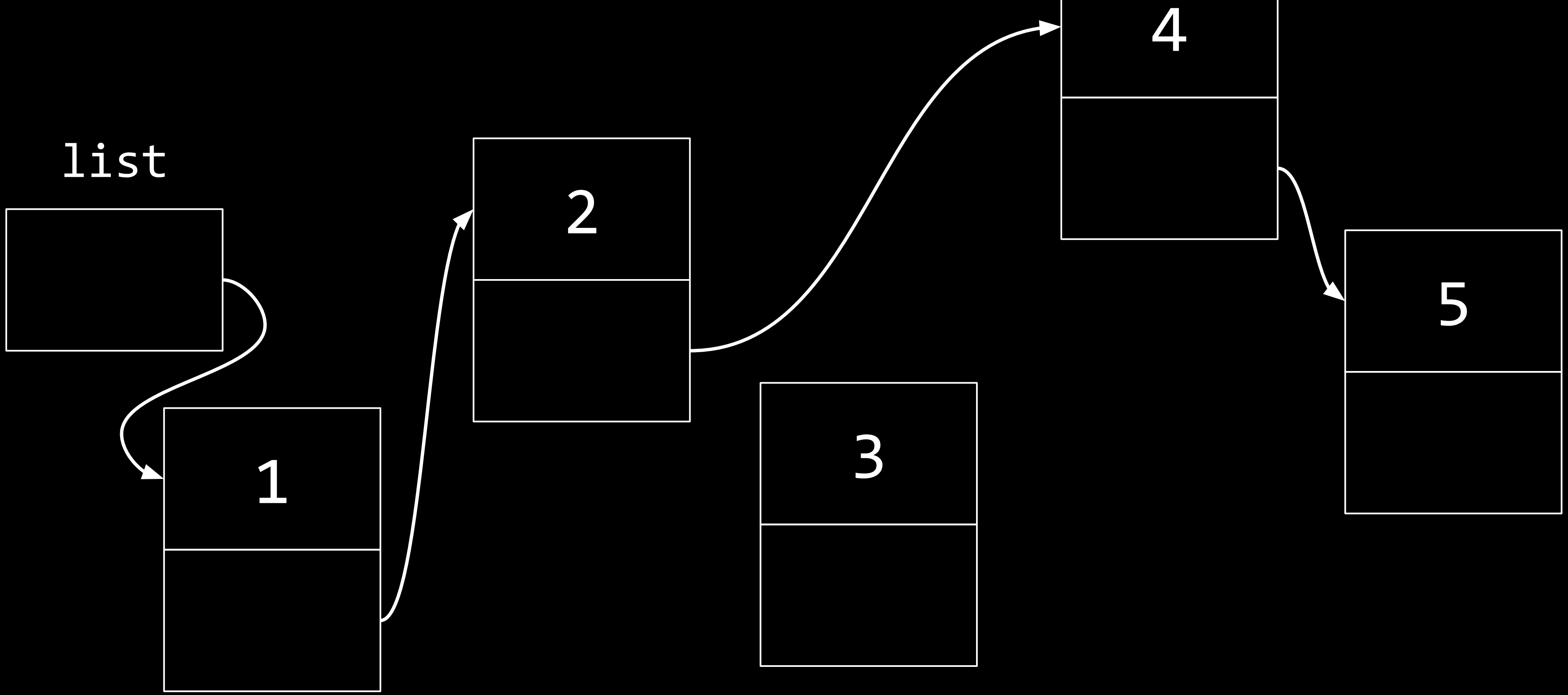

list
1
2
4
5

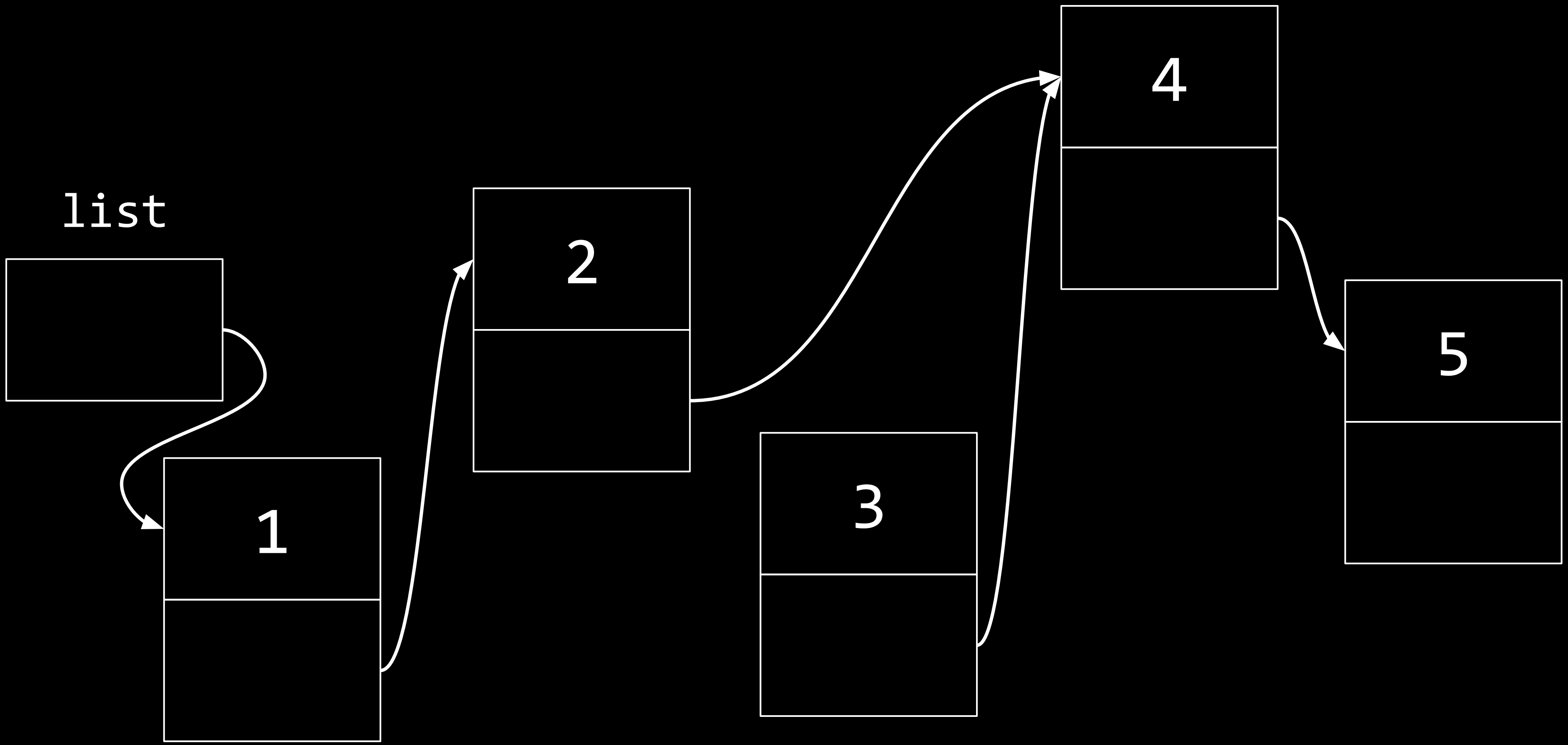
```
n->next = list;
```

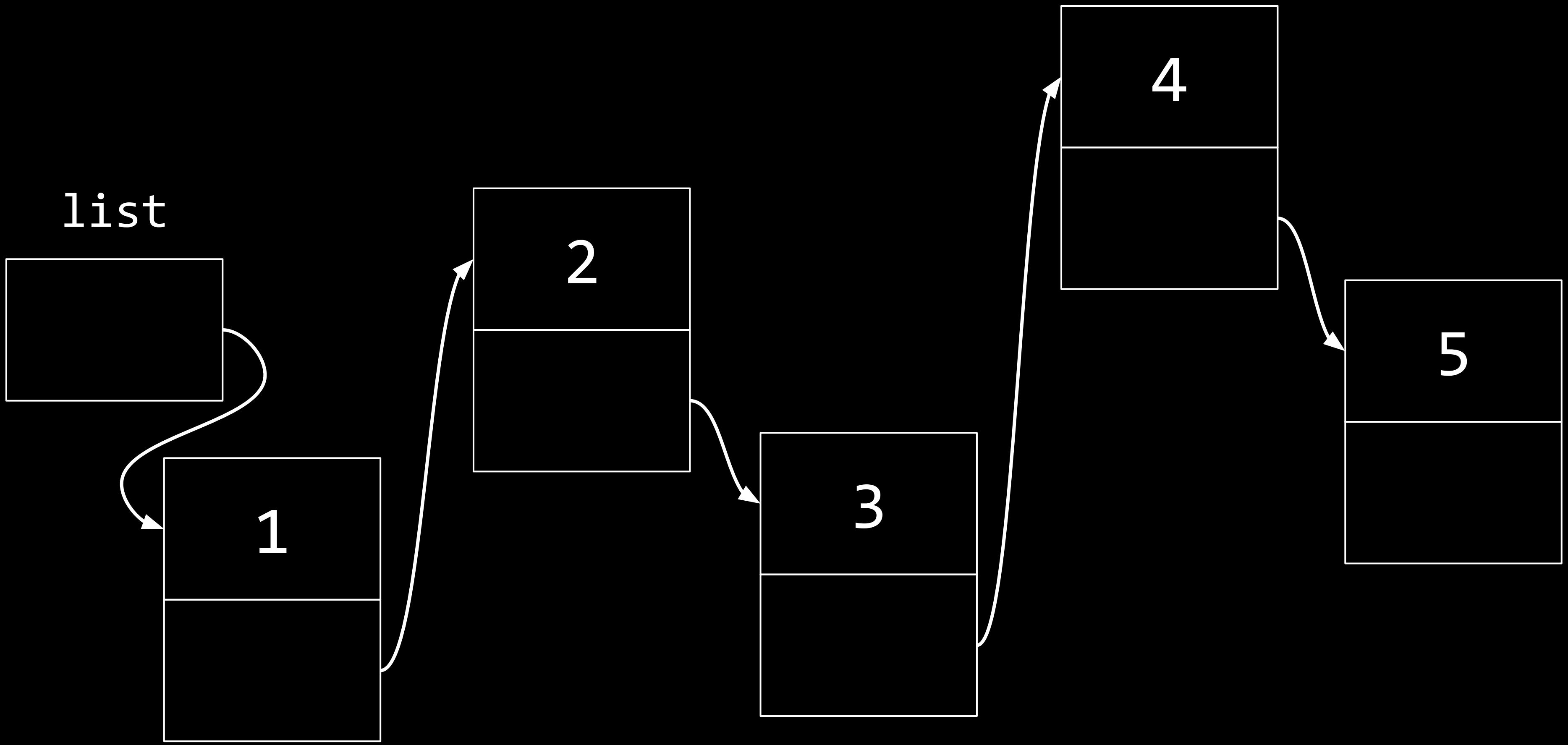

list
2
4
1
5

```
list = n;
```


list
1
2
4
5


list
1
2
3
4
5

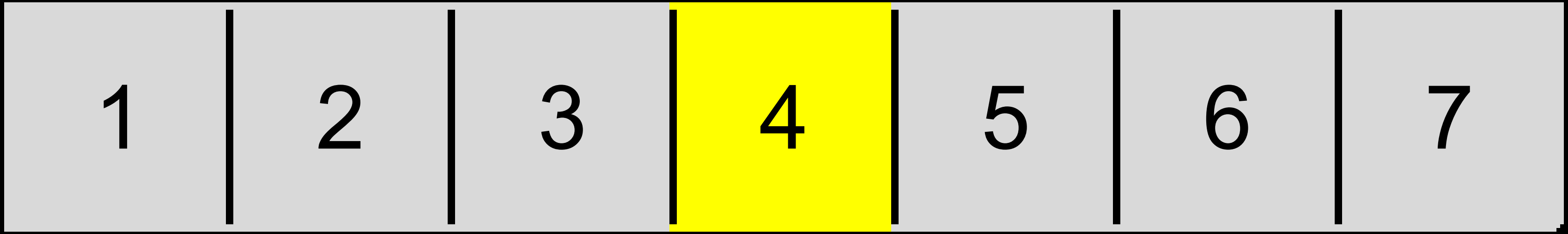

list
1
2
3
4
5

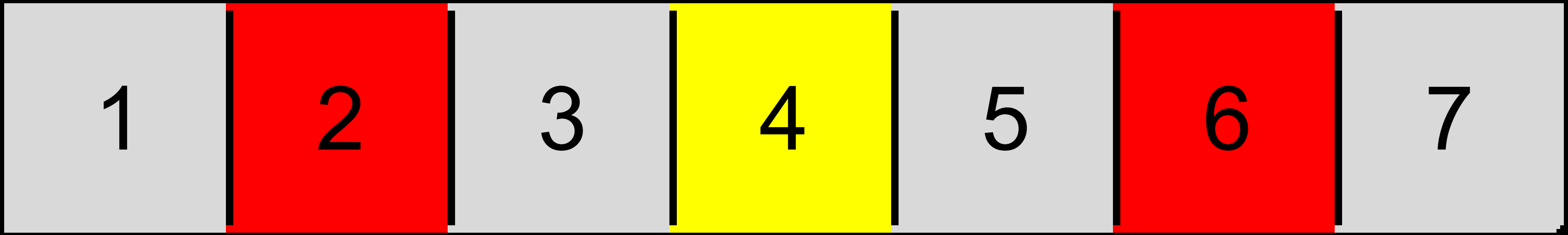

list
1
2
3
4
5

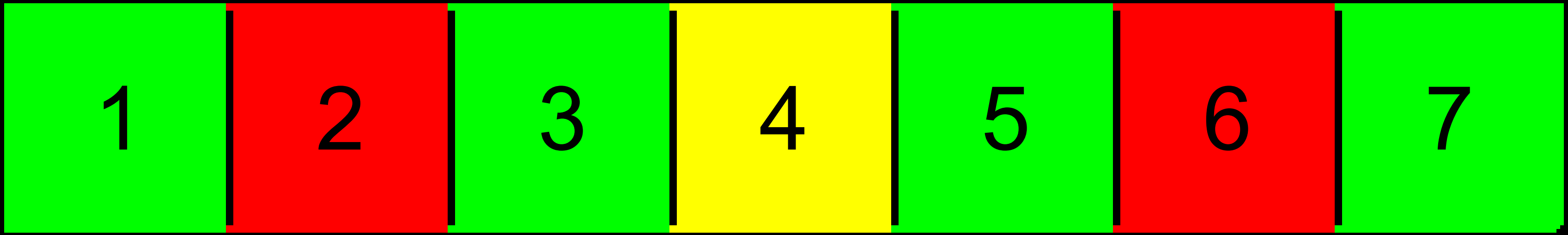
trees

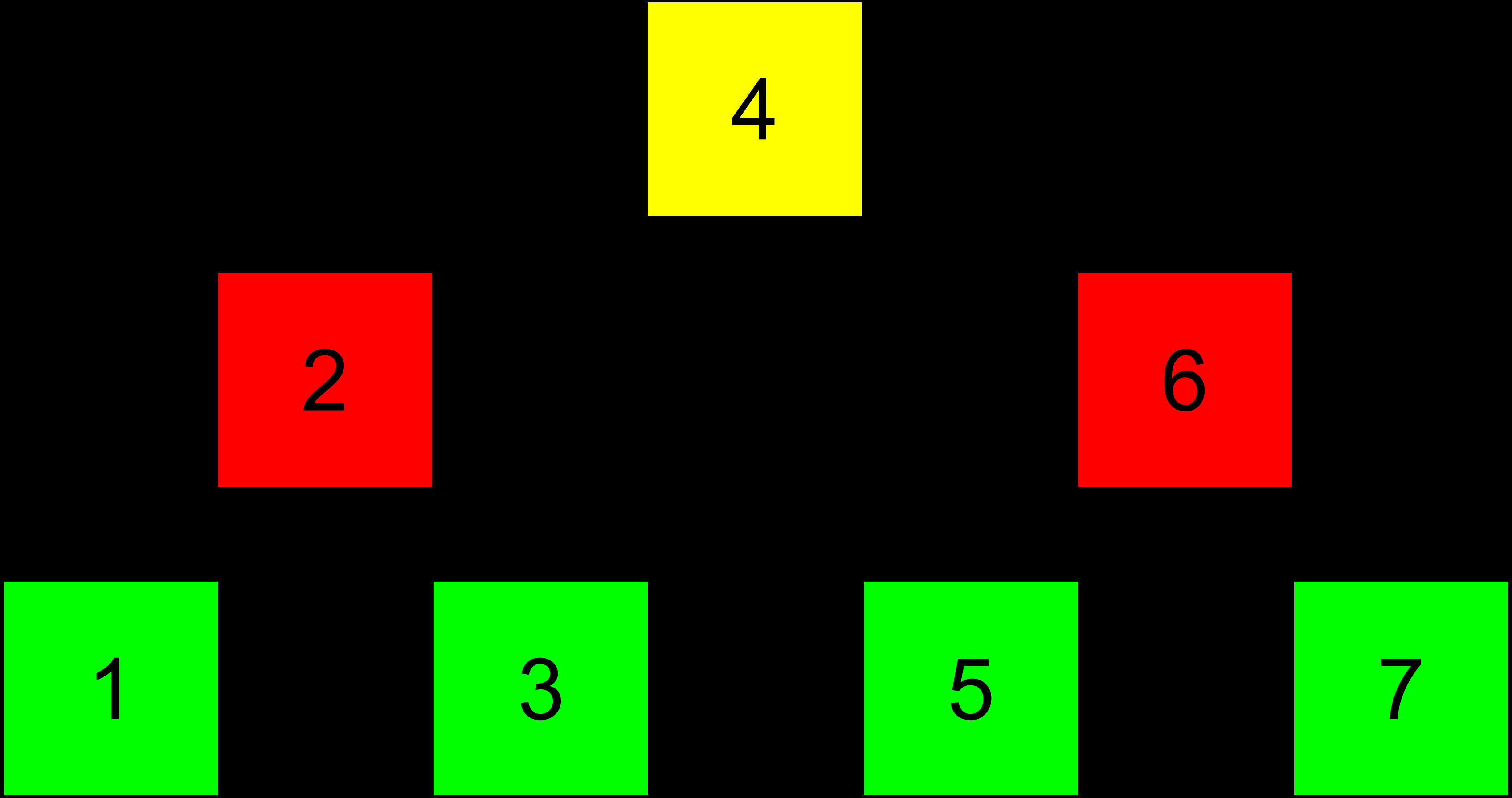
binary search trees

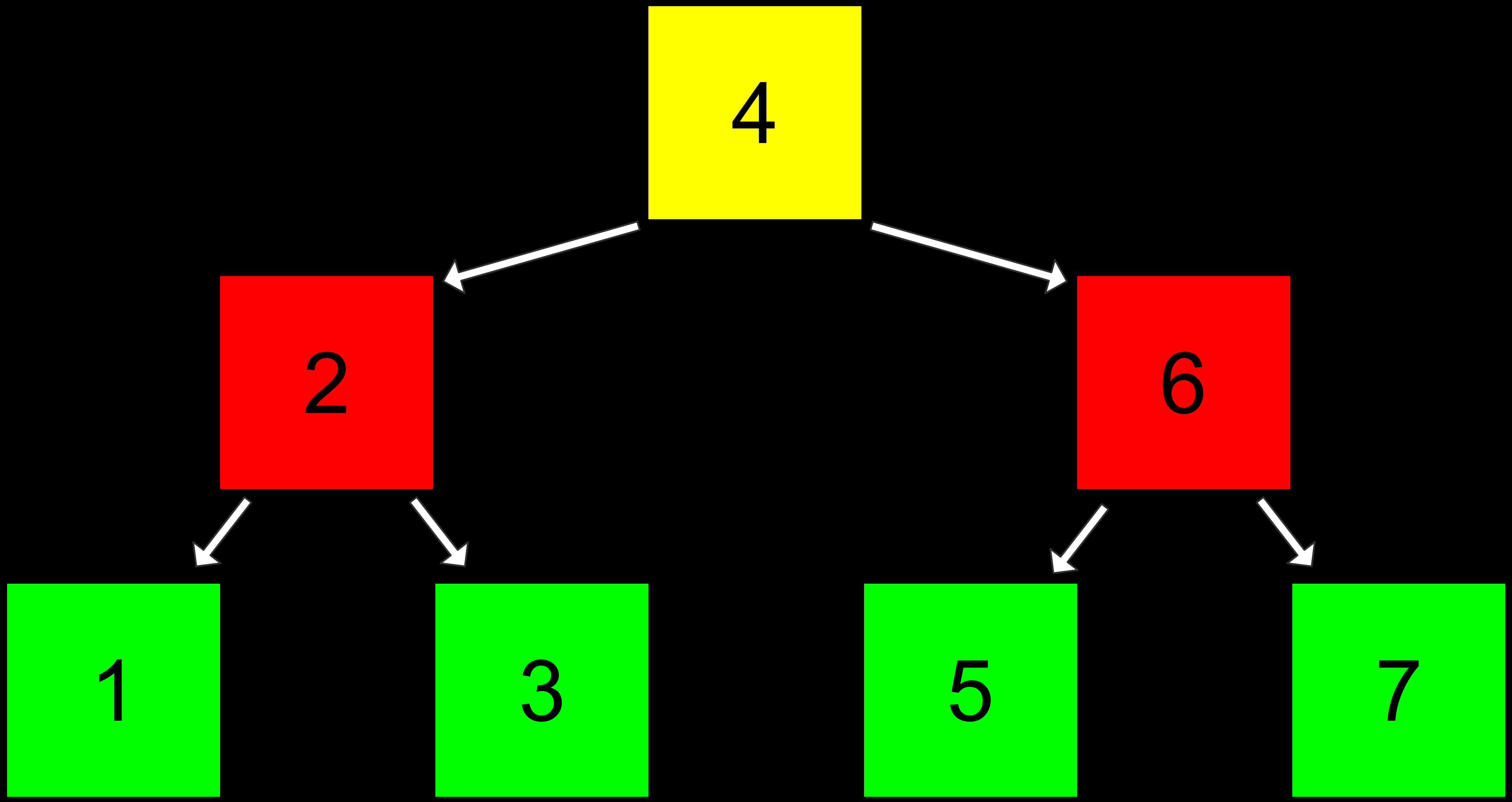
1	2	3	4	5	6	7


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

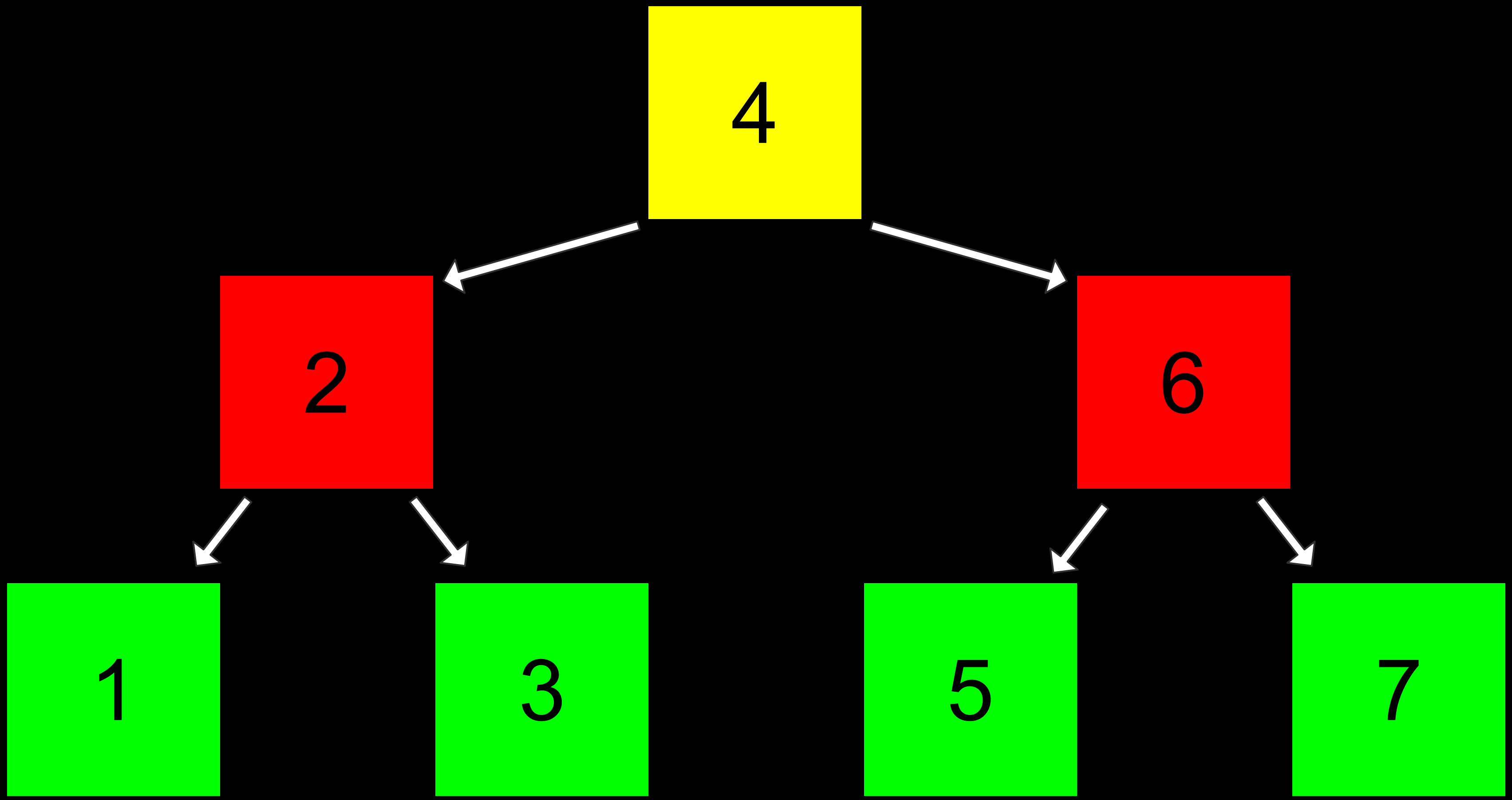

4
2
6
1
3
5
7

```
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)
{

















}
```

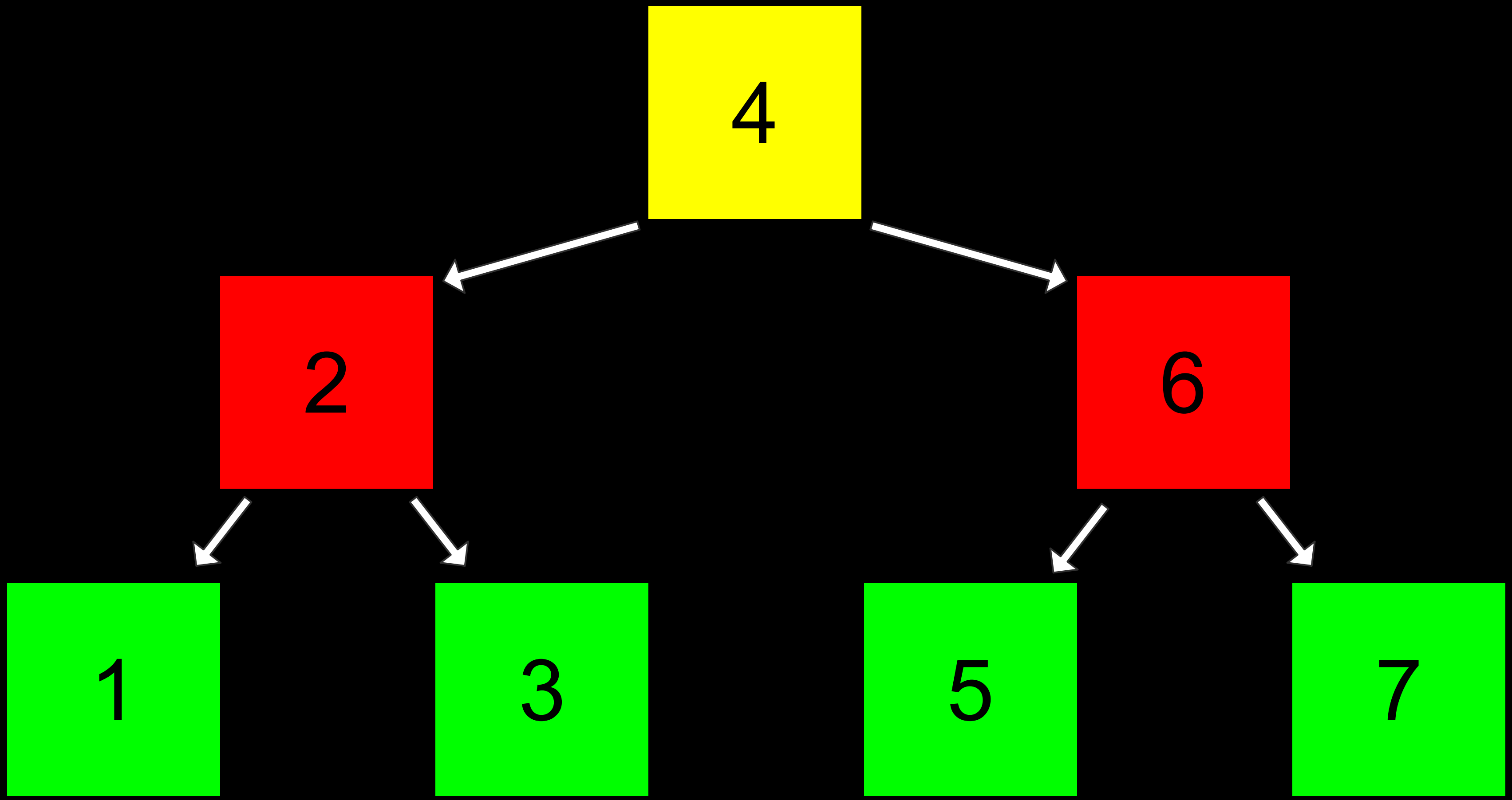
```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }









}
```

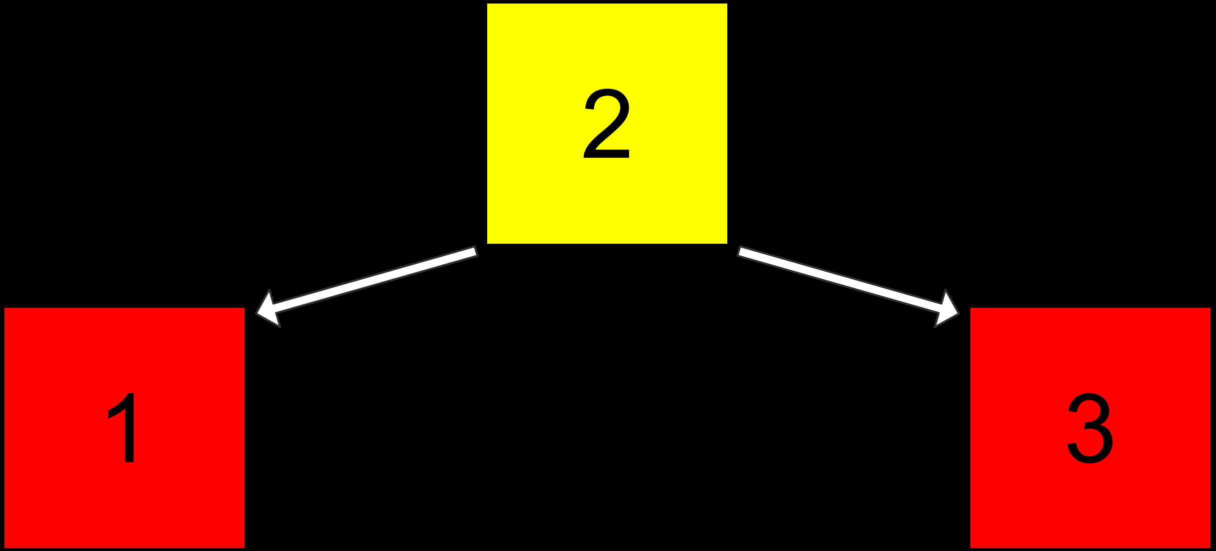
```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (number < tree->number)
    {
        return search(tree->left, number);
    }









}
```

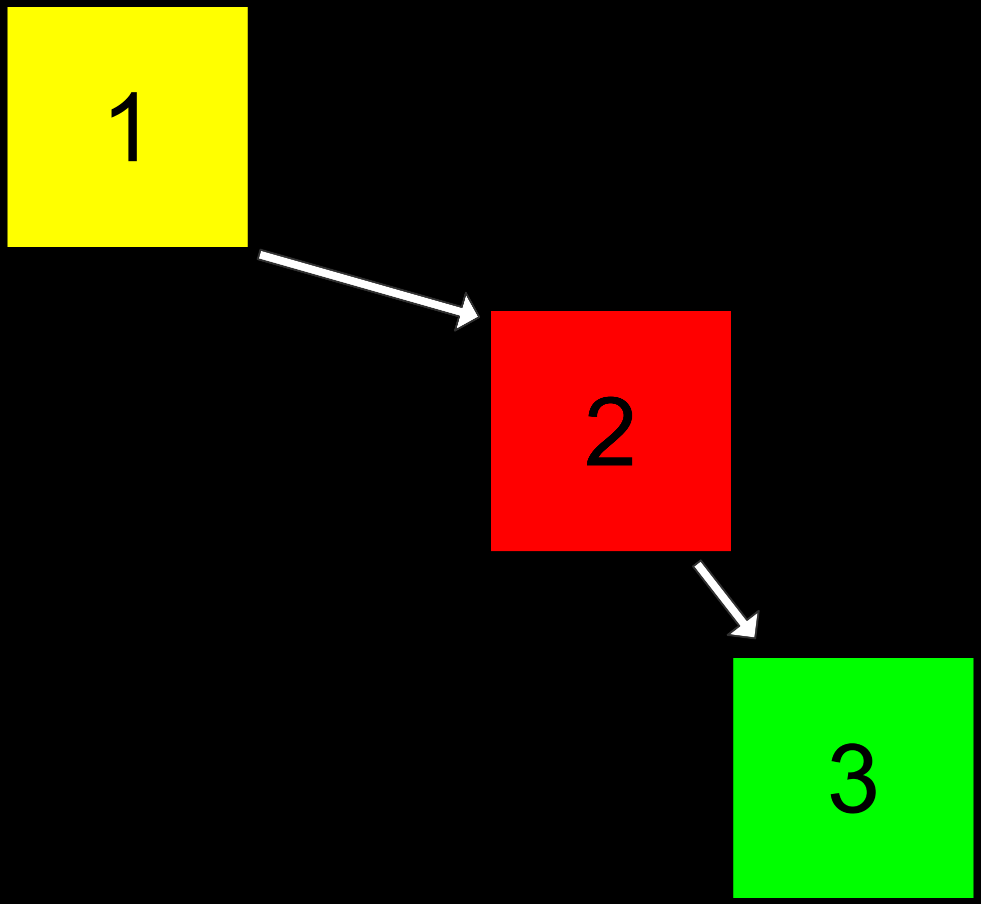
```
bool search(node *tree, int number)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (number < tree->number)
    {
        return search(tree->left, number);
    }
    else if (number > tree->number)
    {
        return search(tree->right, number);
    }




}
```

```
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
1
3


1
2
3

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$ search

$O(1)$

$O(n^2)$	
$O(n \log n)$	
$O(n)$	
$O(\log n)$	search, insert
$O(1)$	

$\Omega(n^2)$

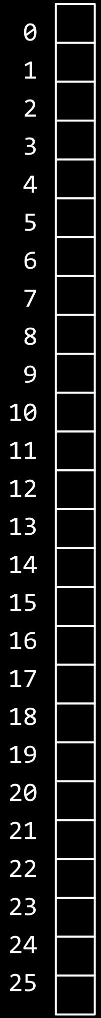
$\Omega(n \log n)$

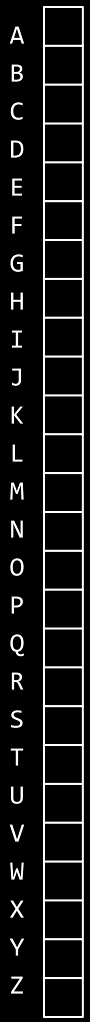
$\Omega(n)$

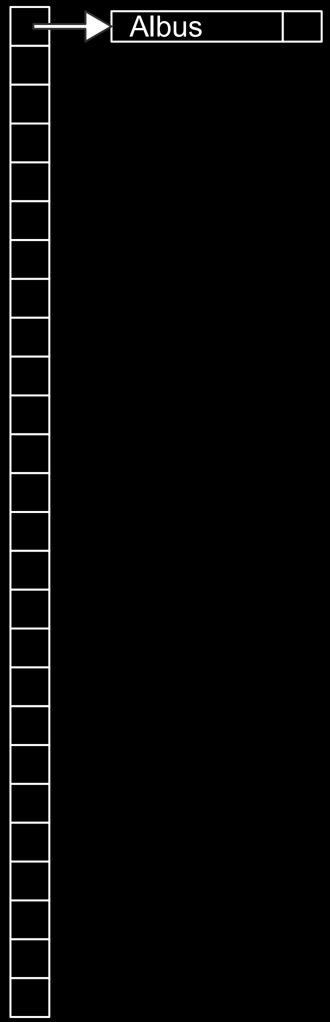
$\Omega(\log n)$

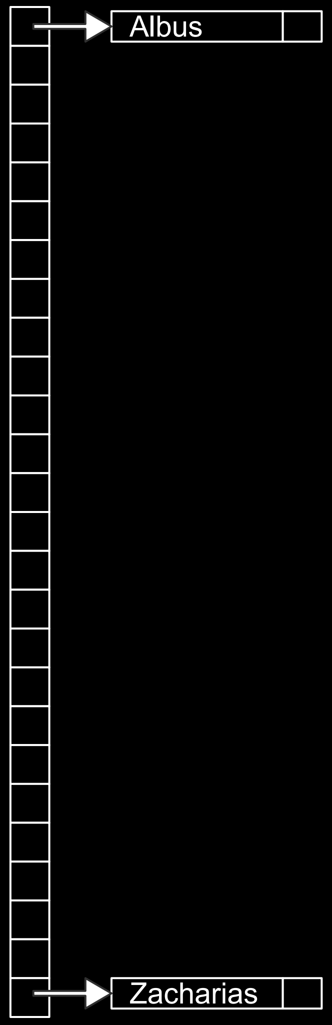
$\Omega(1)$

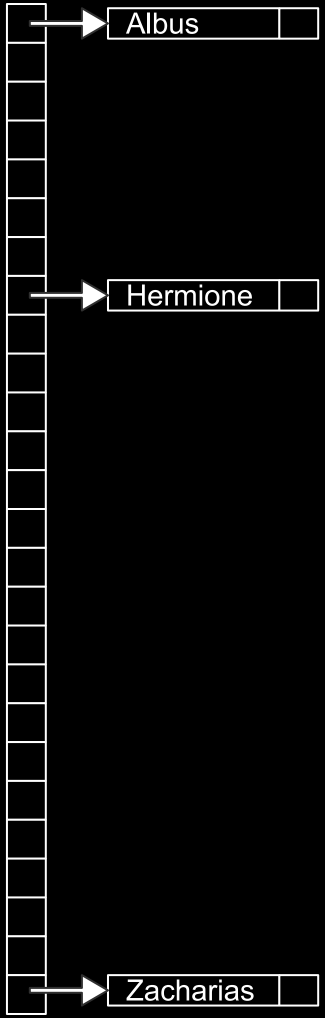
hash tables

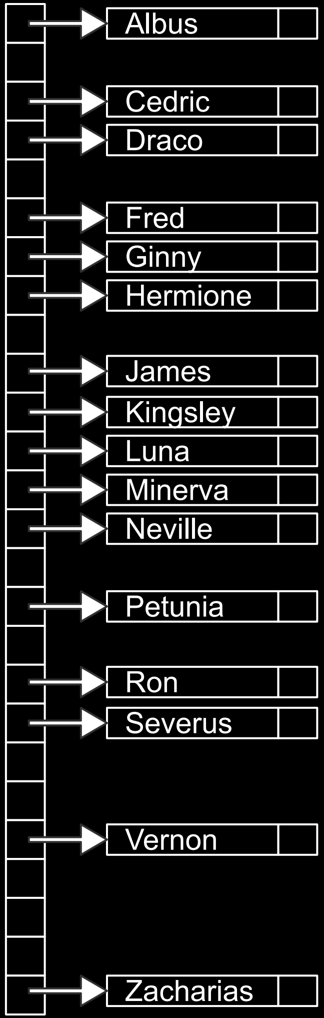

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25


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

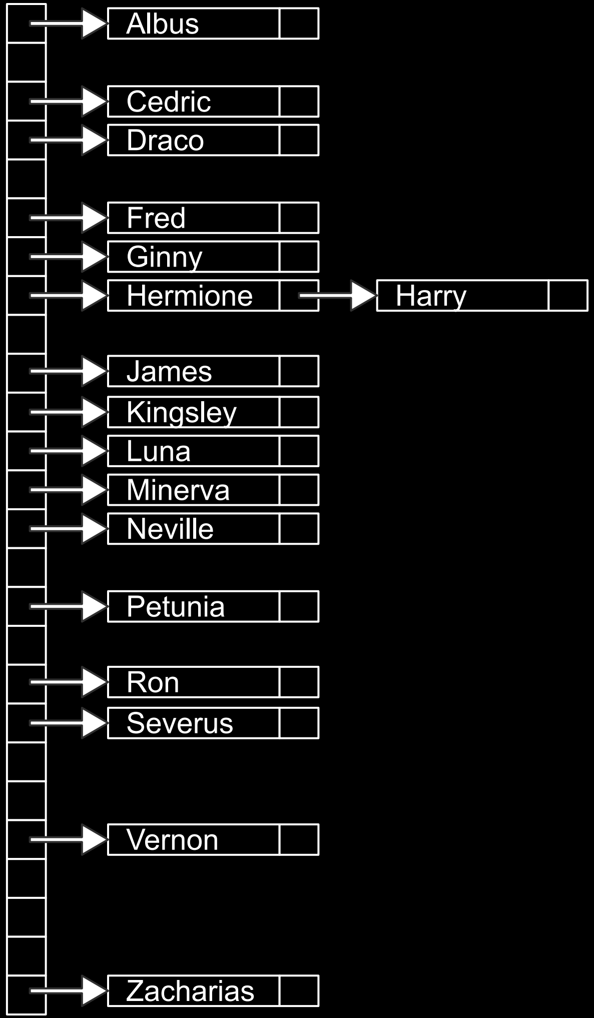

Albus

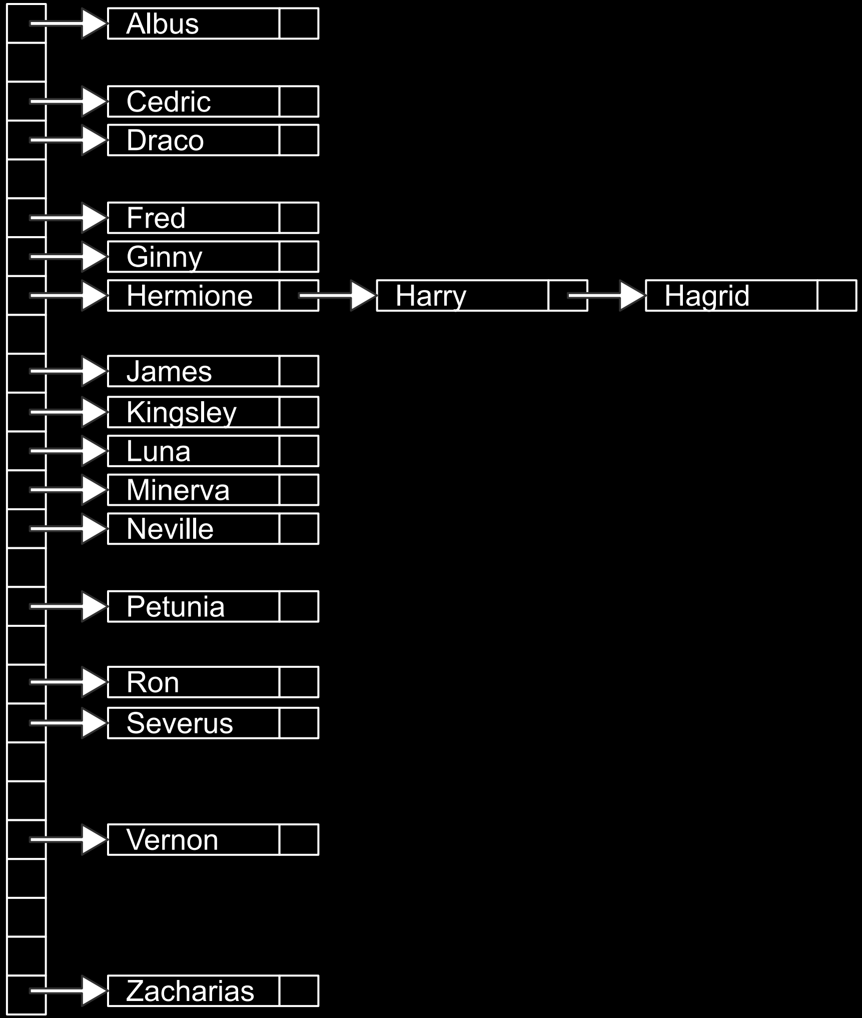

Albus
Zacharias

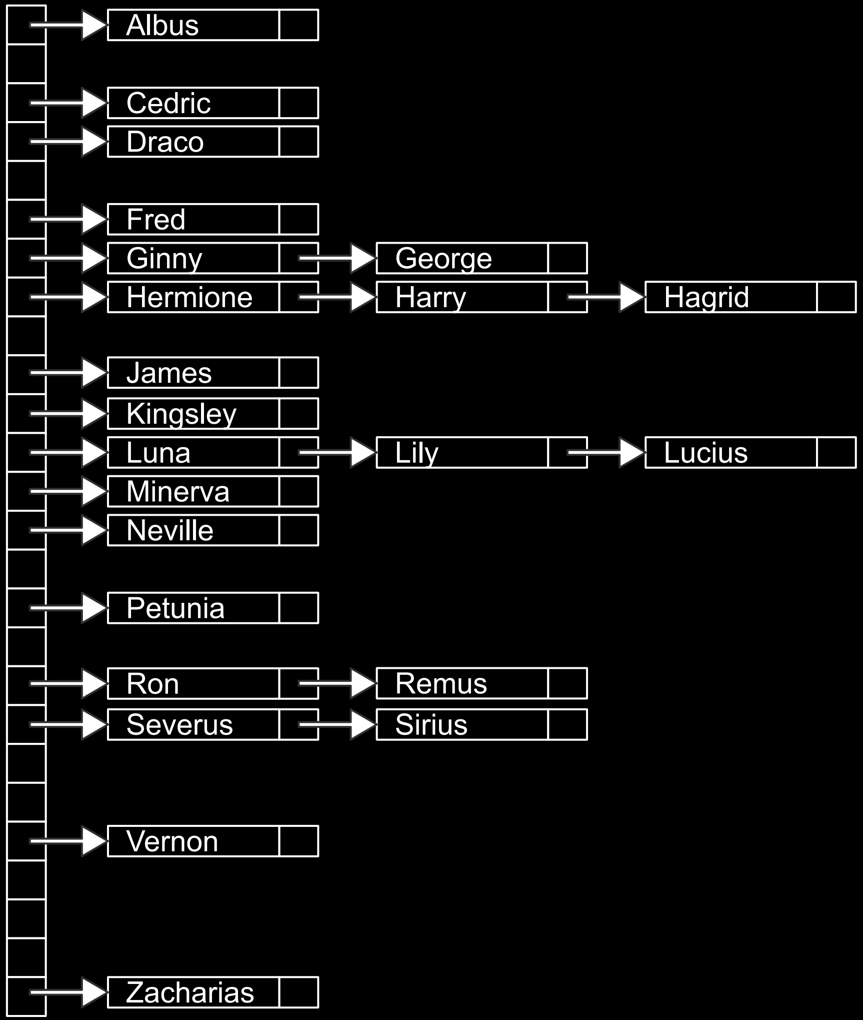

Albus
Hermione
Zacharias

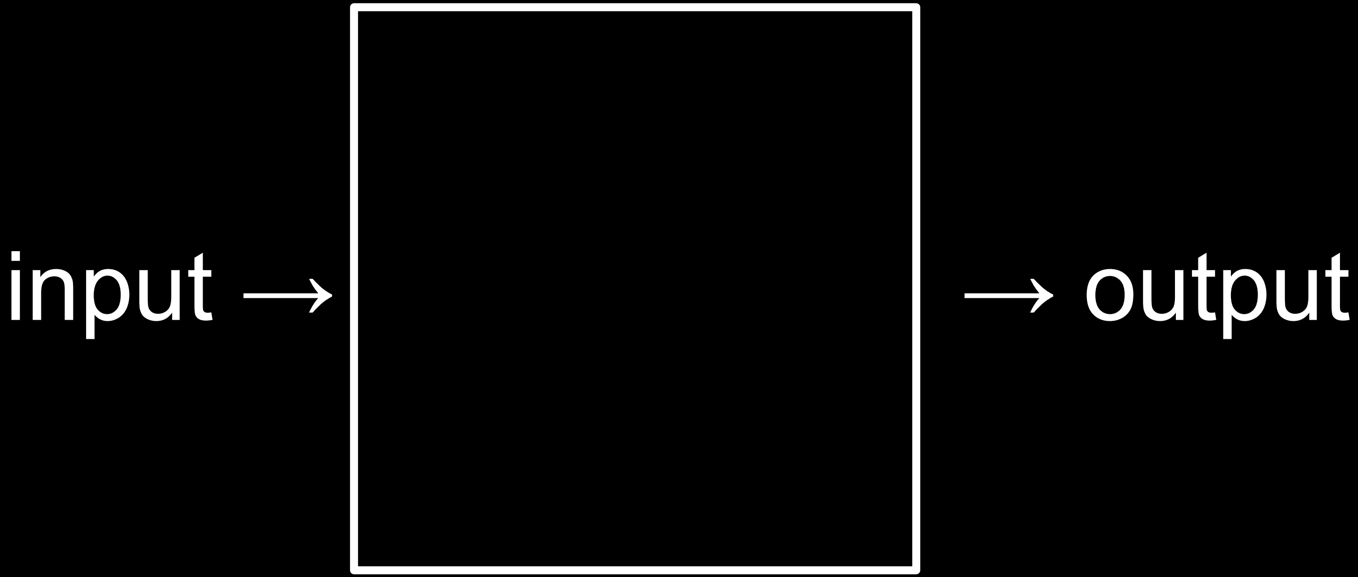


Albus
Cedric
Draco
Fred
Ginny
Hermione
James
Kingsley
Luna
Minerva
Neville
Petunia
Ron
Severus
Vernon
Zacharias

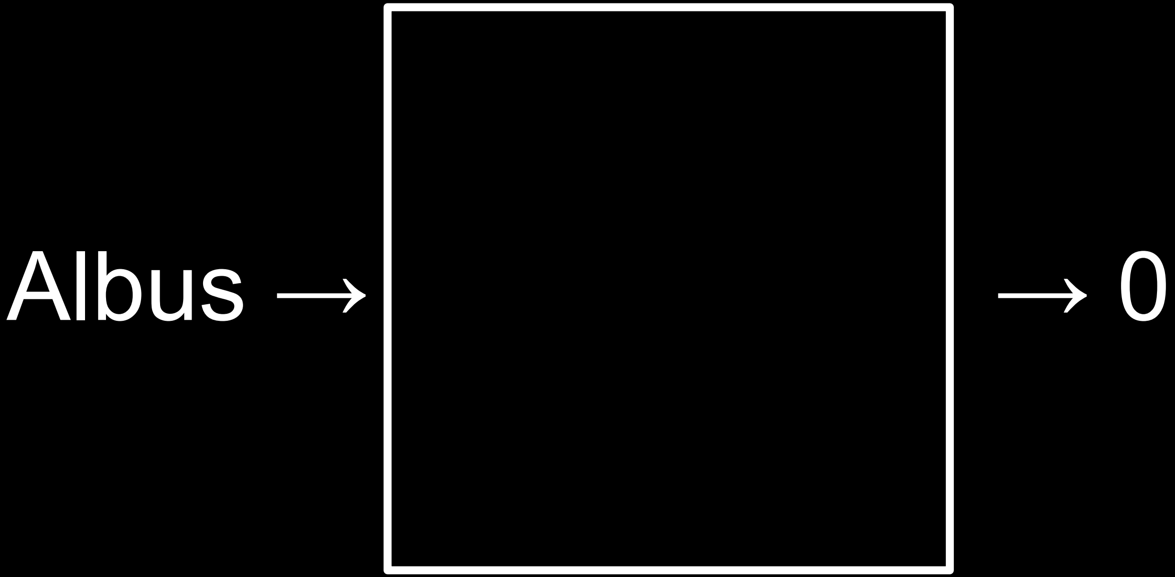

Albus
Cedric
Draco
Fred
Ginny
Hermione
Harry
James
Kingsley
Luna
Minerva
Neville
Petunia
Ron
Severus
Vernon
Zacharias


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

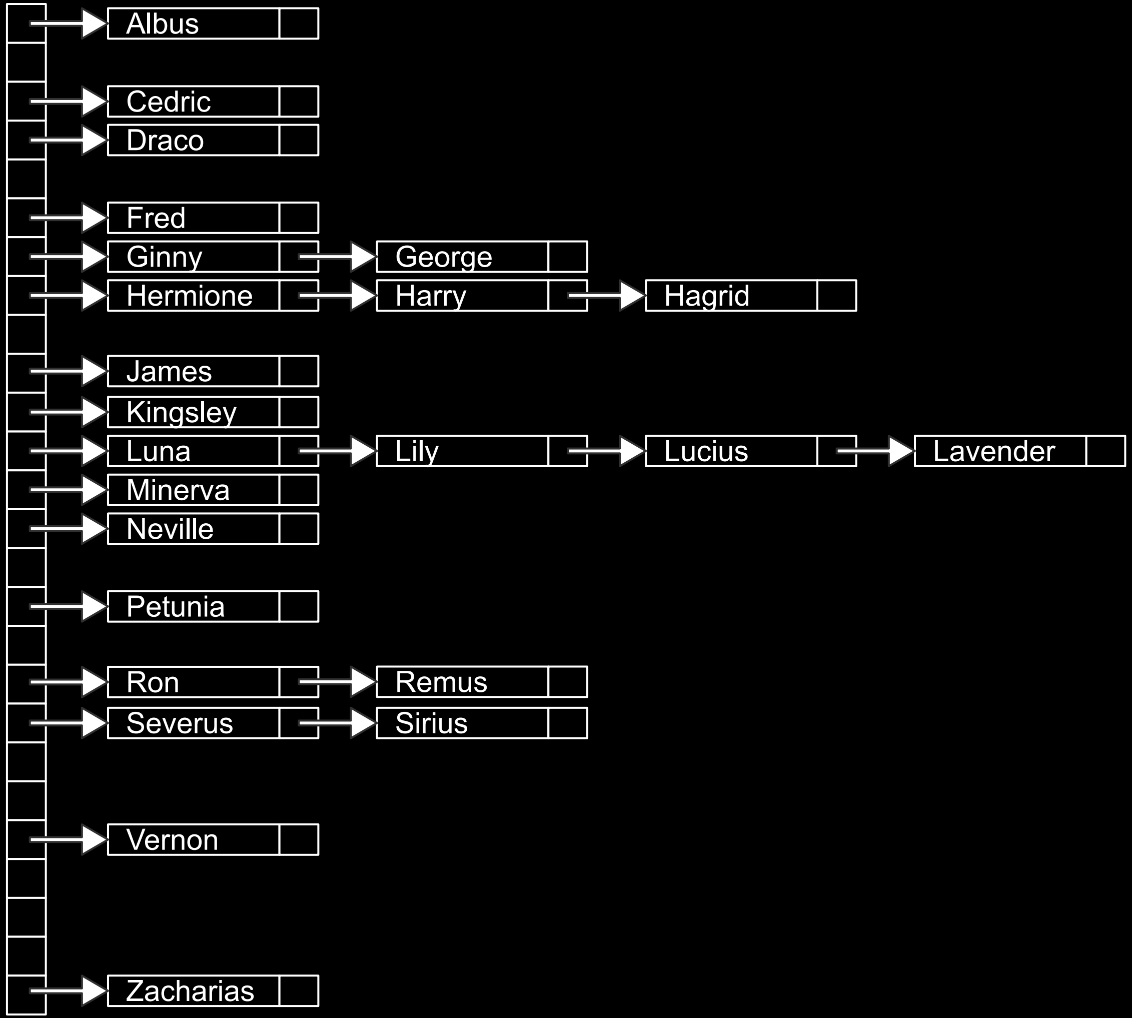

Albus
Cedric
Draco
Fred
Ginny
George
Hermione
Harry
Hagrid
James
Kingsley
Luna
Lily
Lucius
Minerva
Neville
Petunia
Ron
Remus
Severus
Sirius
Vernon
Zacharias

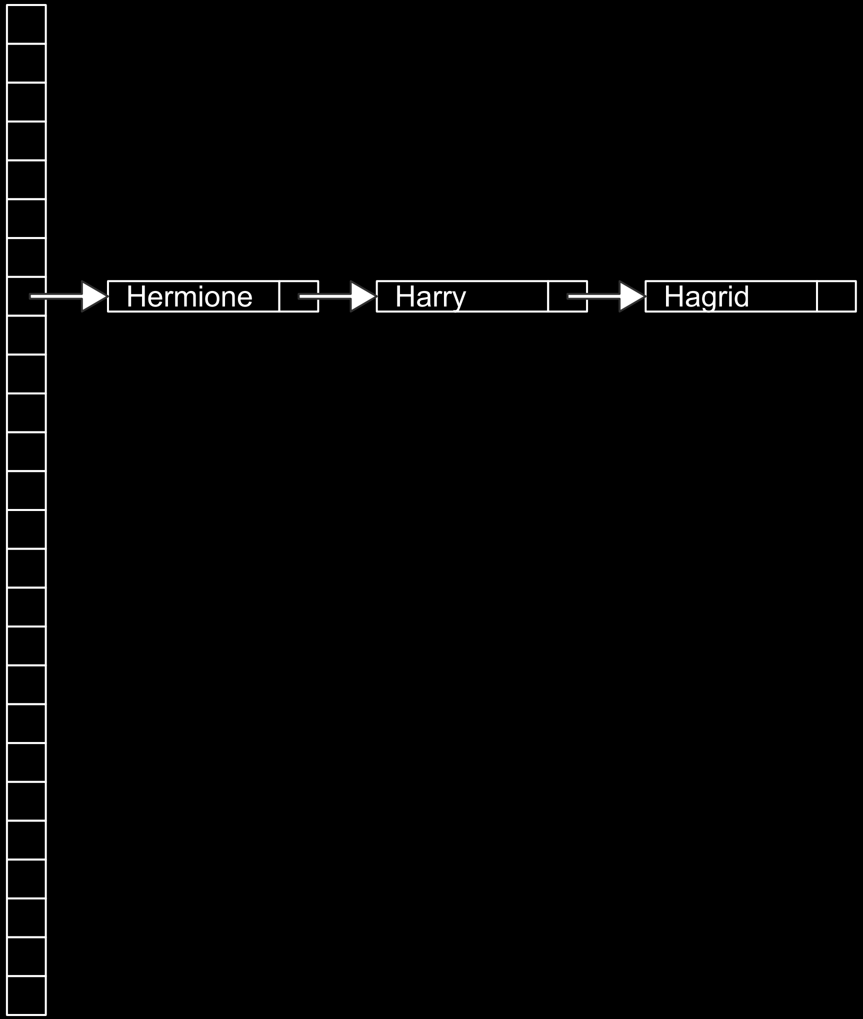

input →
→ output

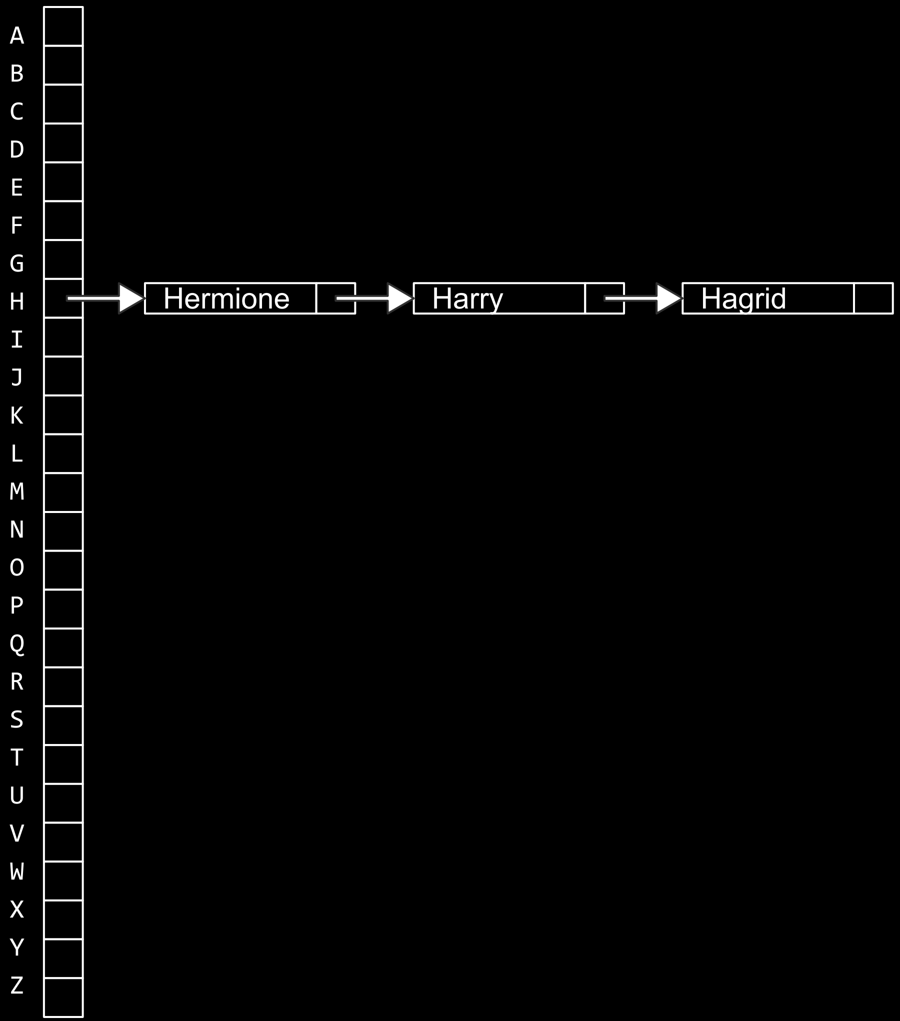
hash function

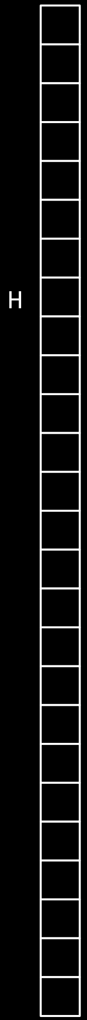

Albus →
→ 0

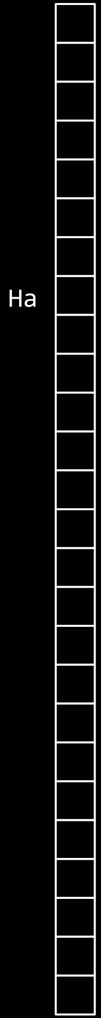
$$\text{Zacharias} \rightarrow \boxed{\phantom{\rule{4cm}{4cm}}} \rightarrow 25$$

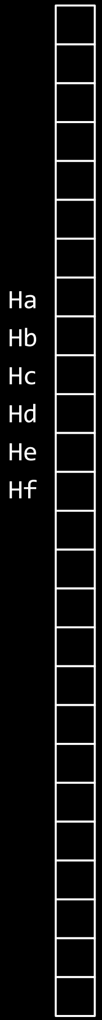

Albus
Cedric
Draco
Fred
Ginny
George
Hermione
Harry
Hagrid
James
Kingsley
Luna
Lily
Lucius
Lavender
Minerva
Neville
Petunia
Ron
Remus
Severus
Sirius
Vernon
Zacharias

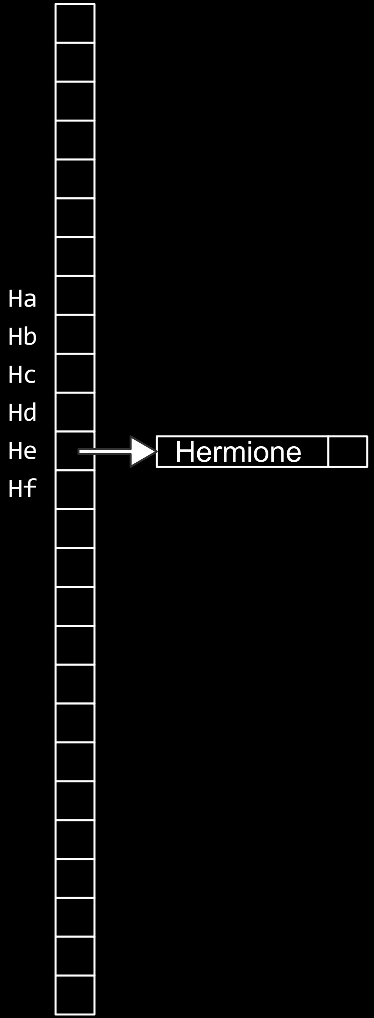

Hermione
Harry
Hagrid

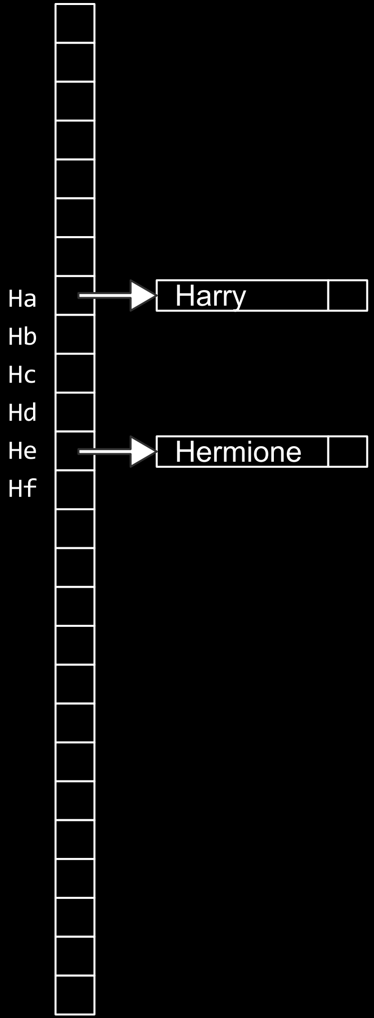

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
Hermione
Harry
Hagrid

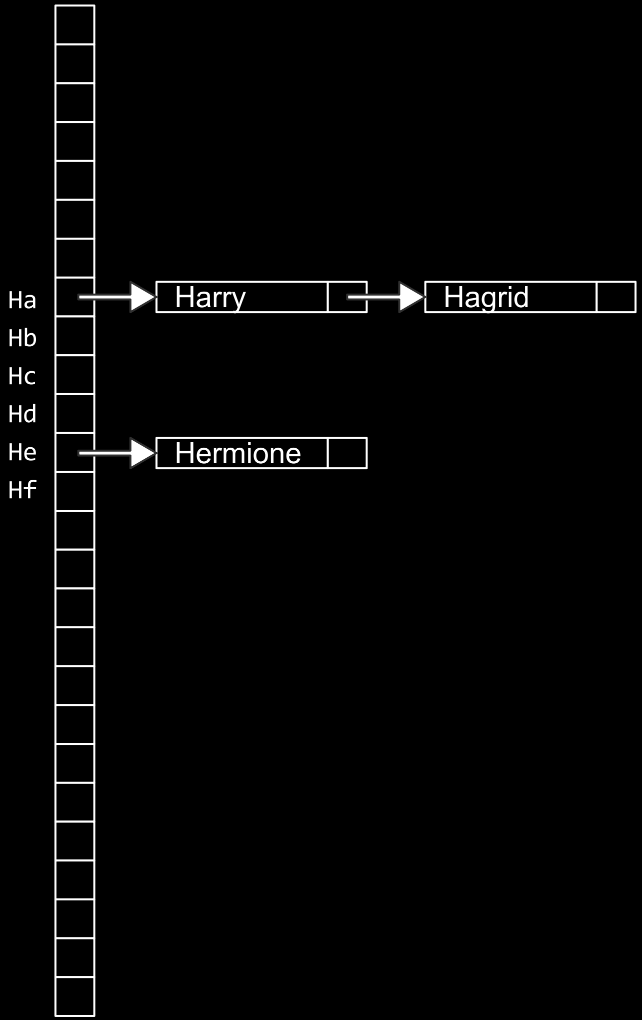

H

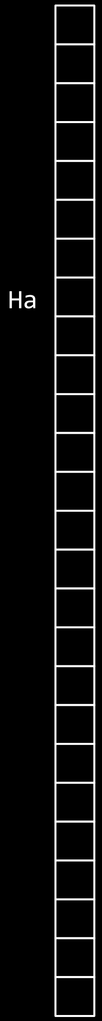

Ha

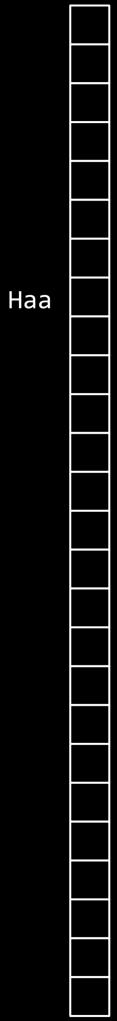

Ha
Hb
Hc
Hd
He
Hf

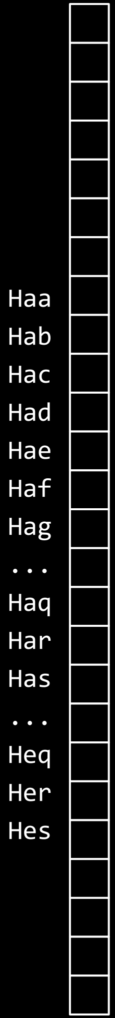

Ha
Hb
Hc
Hd
He
Hf
Hermione

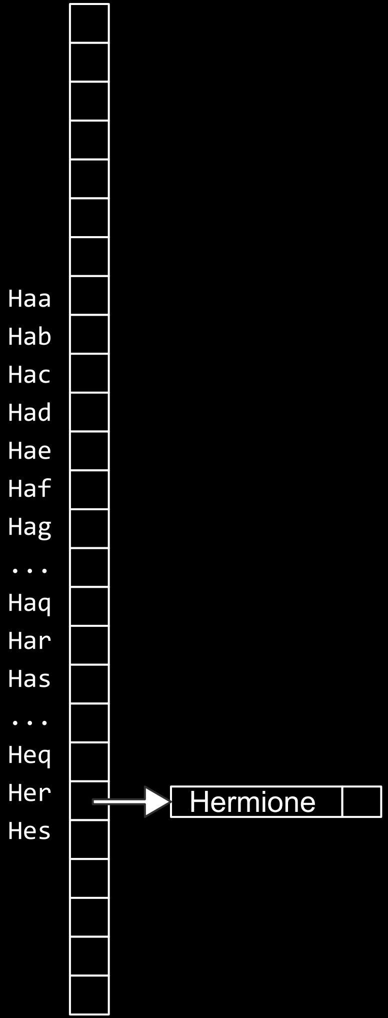

Ha
Hb
Hc
Hd
He
Hf
Harry
Hermione

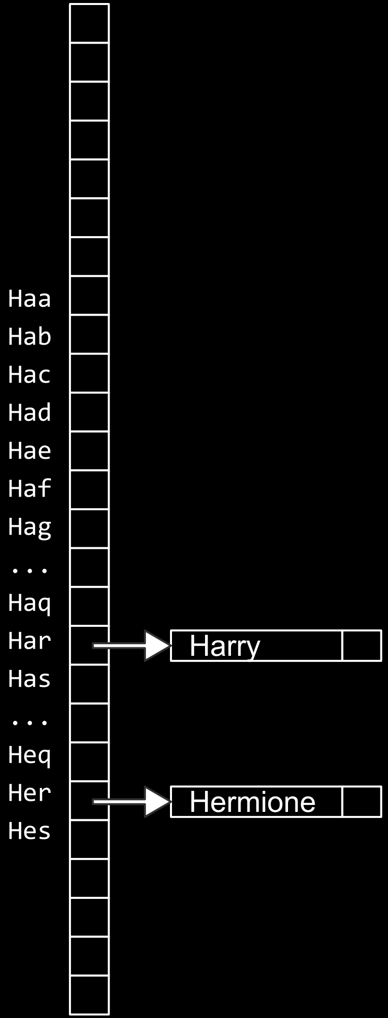

Ha
Hb
Hc
Hd
He
Hf
Harry
Hagrid
Hermione

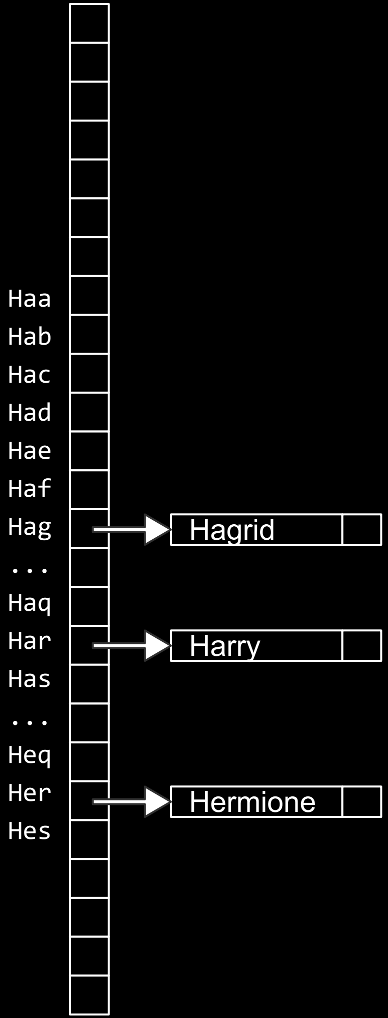

Ha

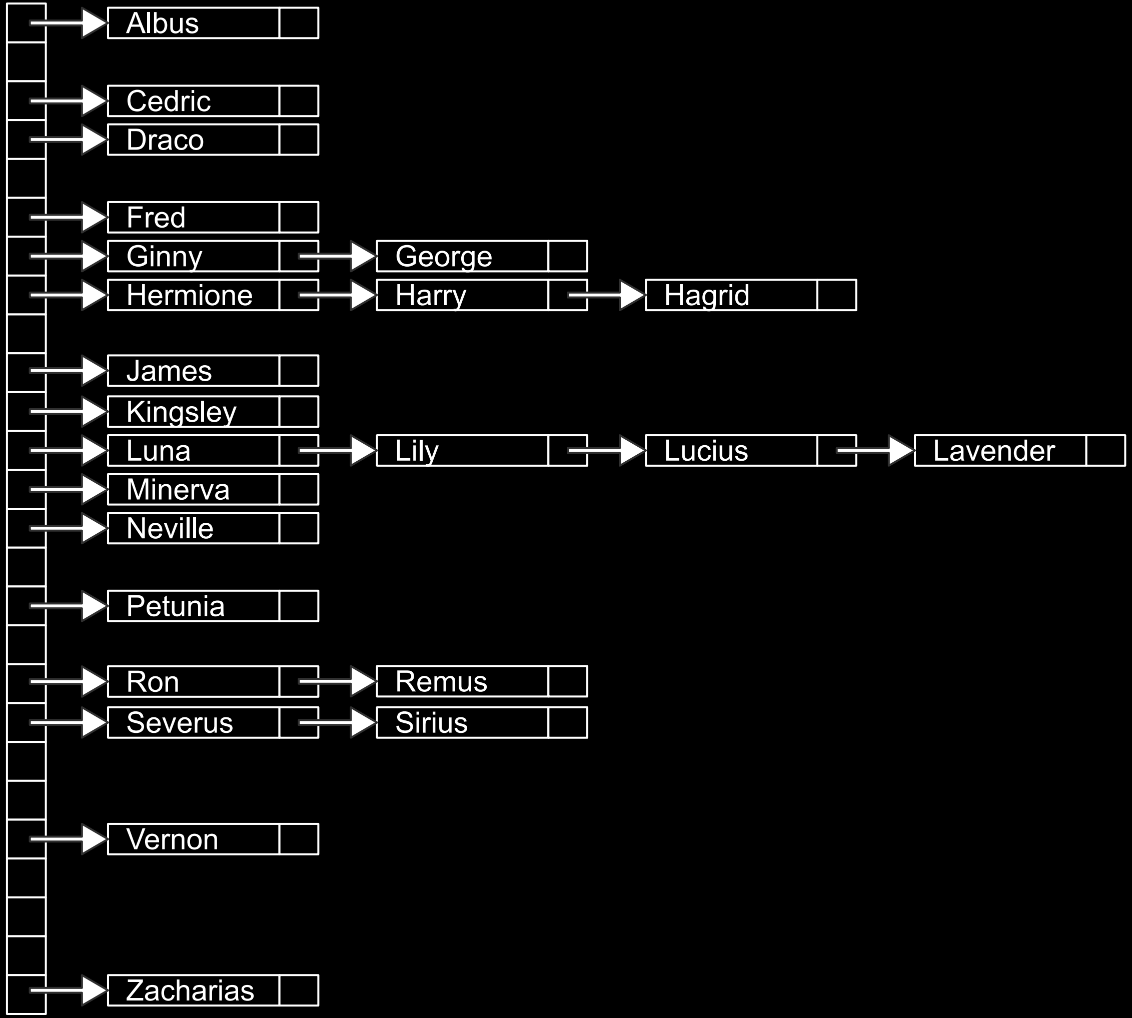

Haa


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


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

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

$O(n^2)$	
$O(n \log n)$	
$O(n)$	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

$\Omega(n^2)$

$\Omega(n \log n)$

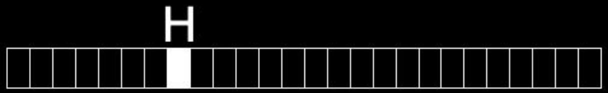
$\Omega(n)$

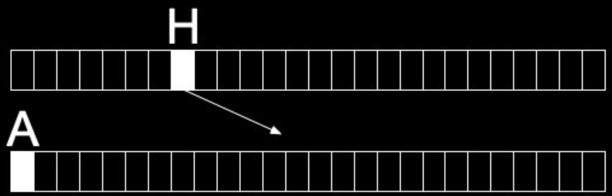
$\Omega(\log n)$

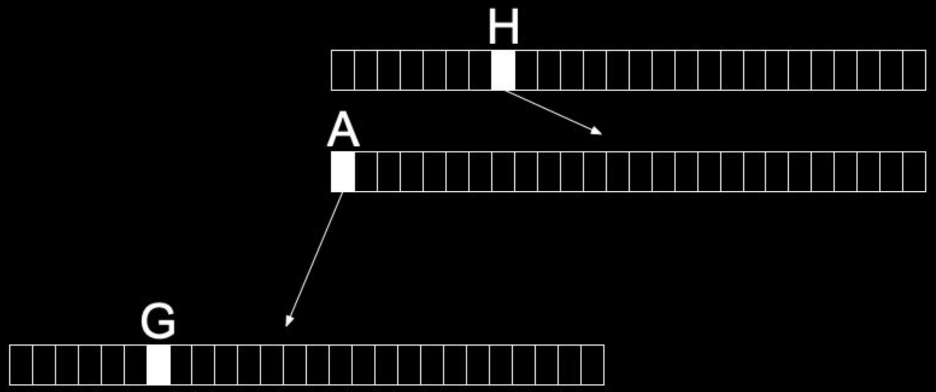
$\Omega(1)$

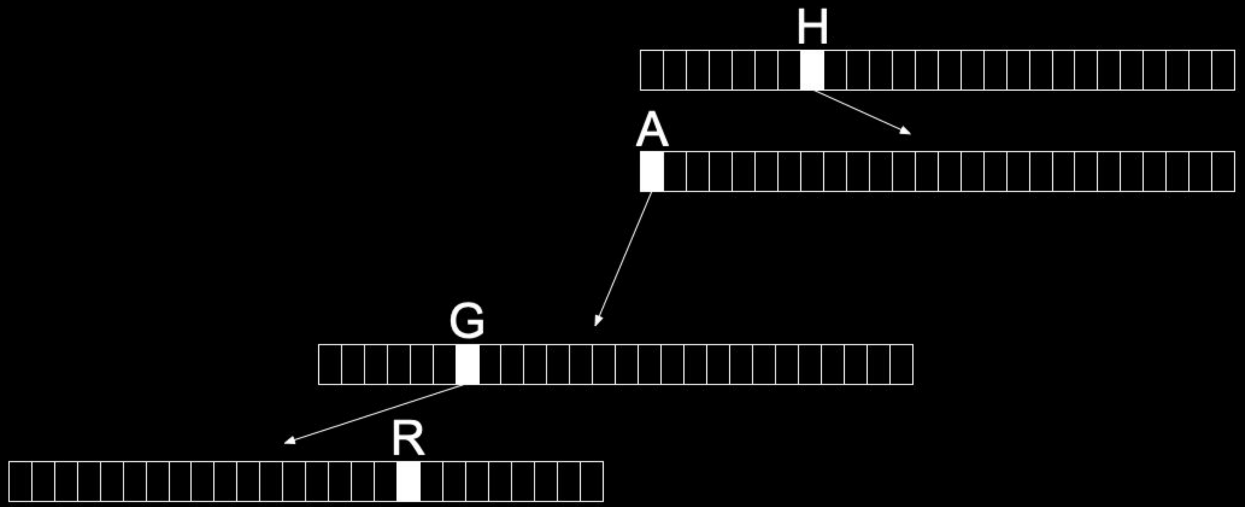
tries

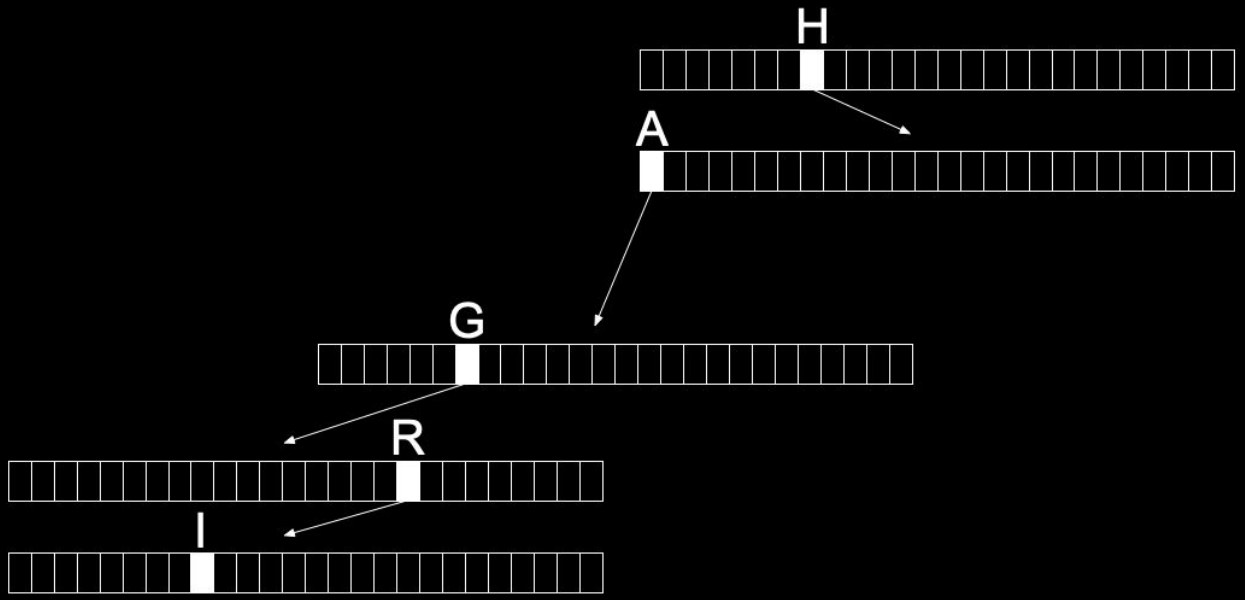
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

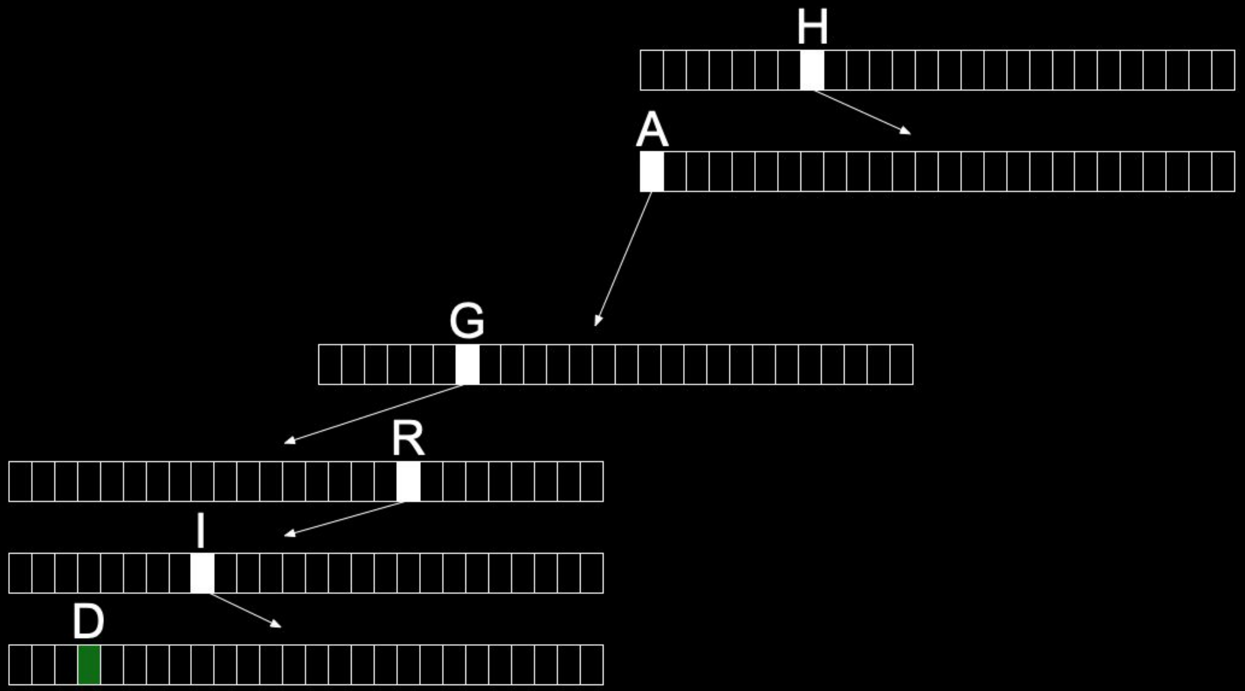

H

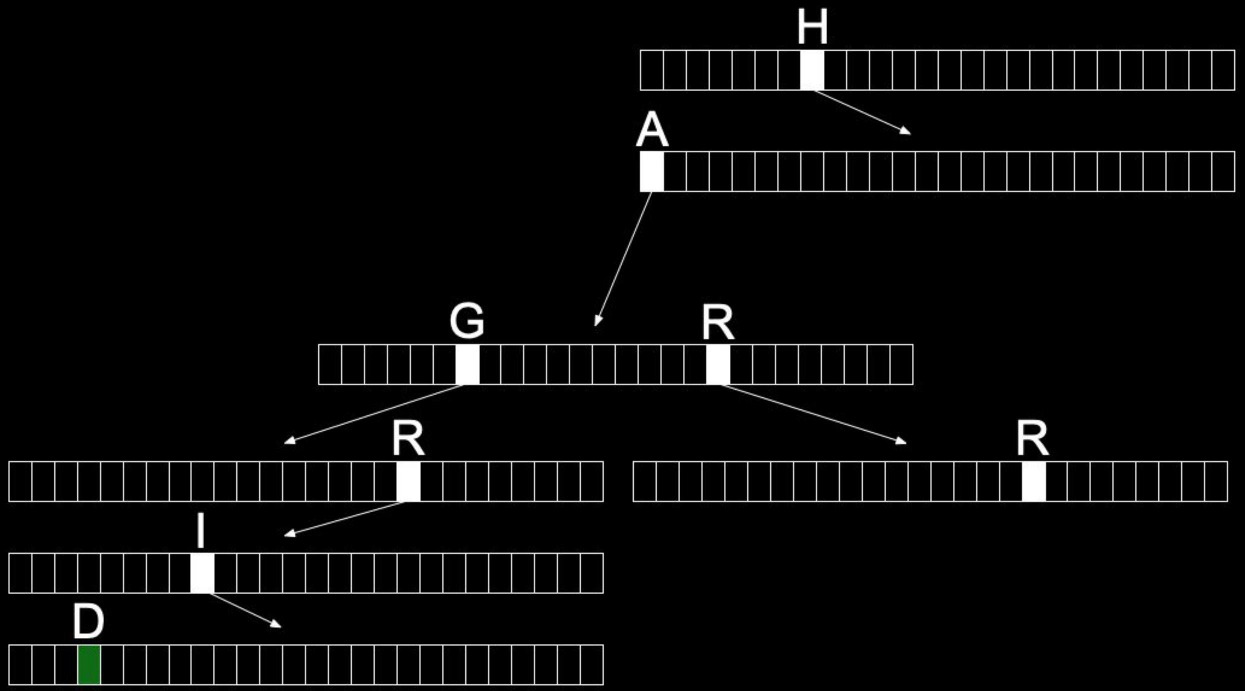

H
A

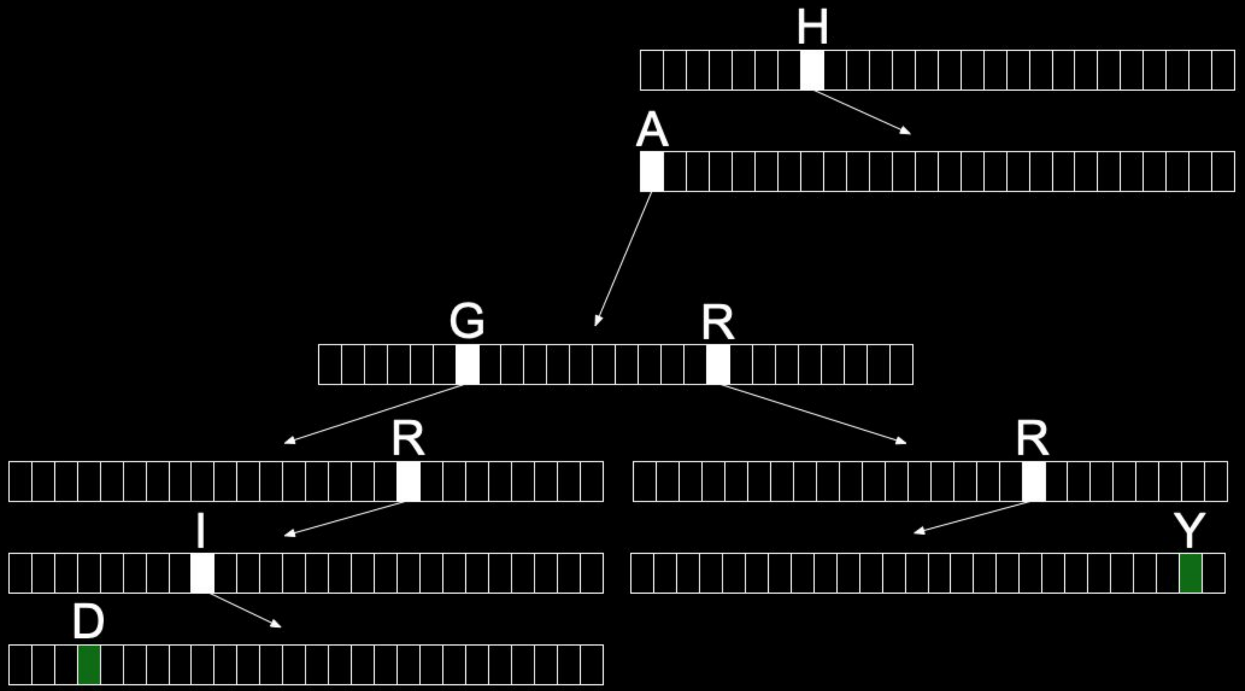

H
A
G

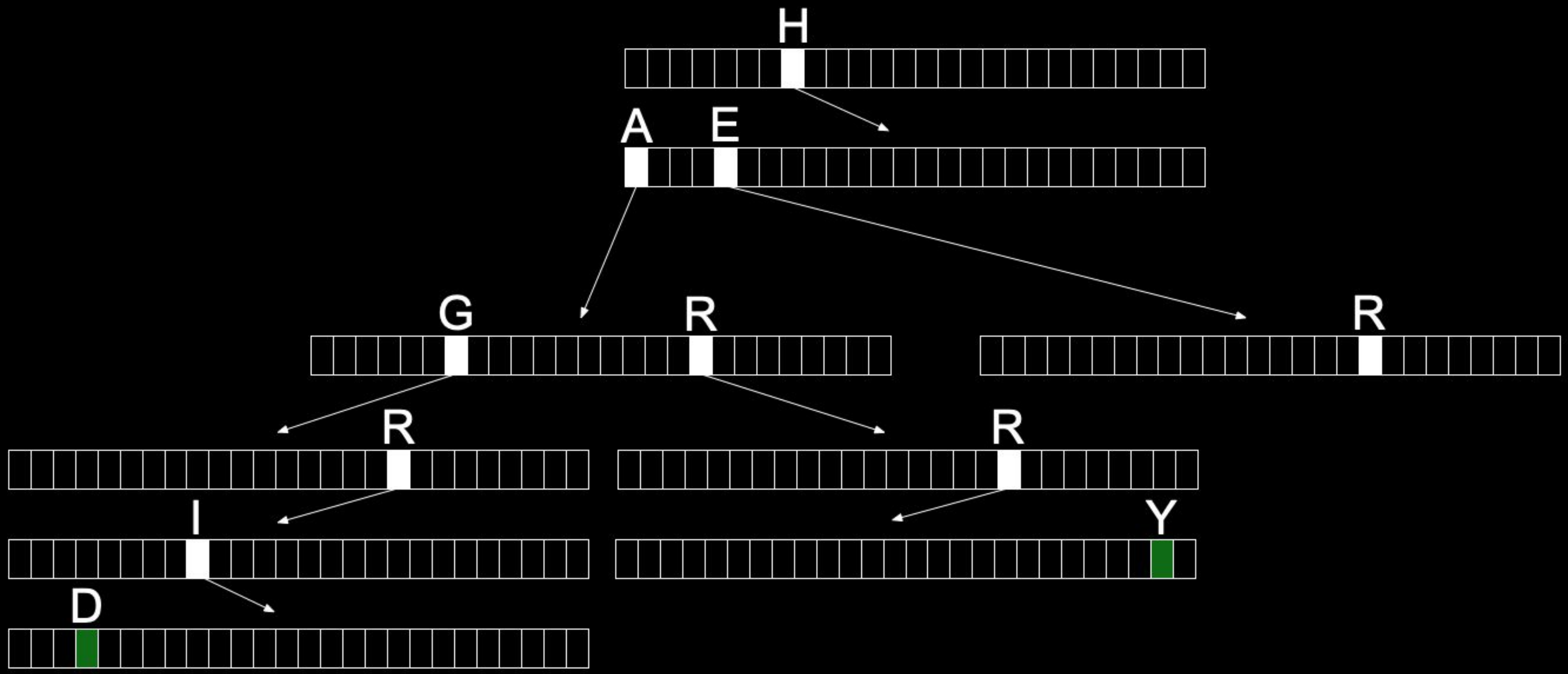

H
A
G
R

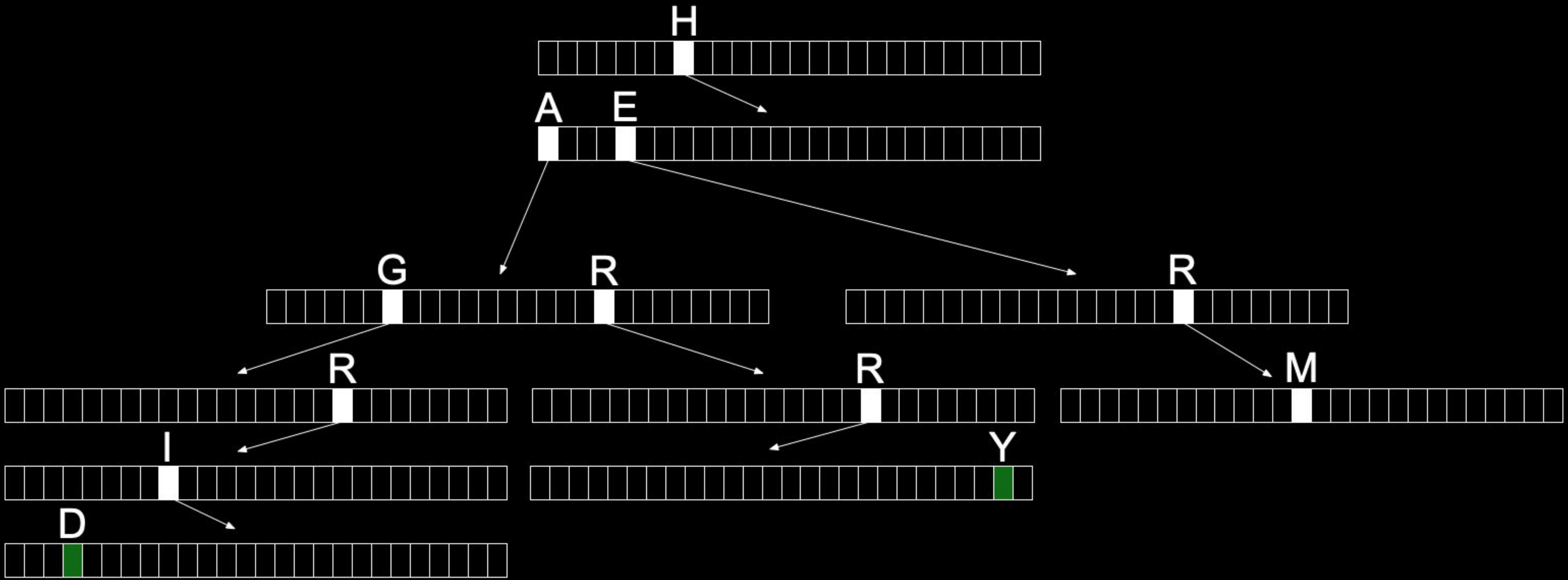

H
A
G
R
I

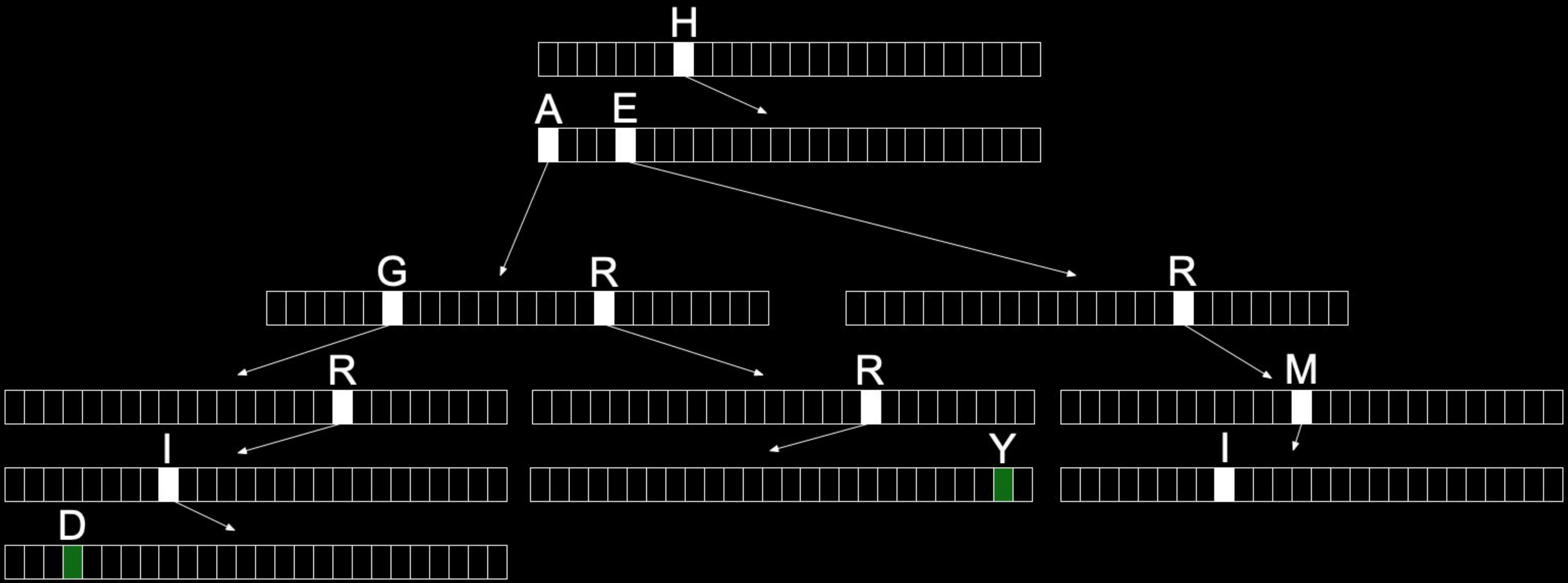

H
A
G
R
I
D

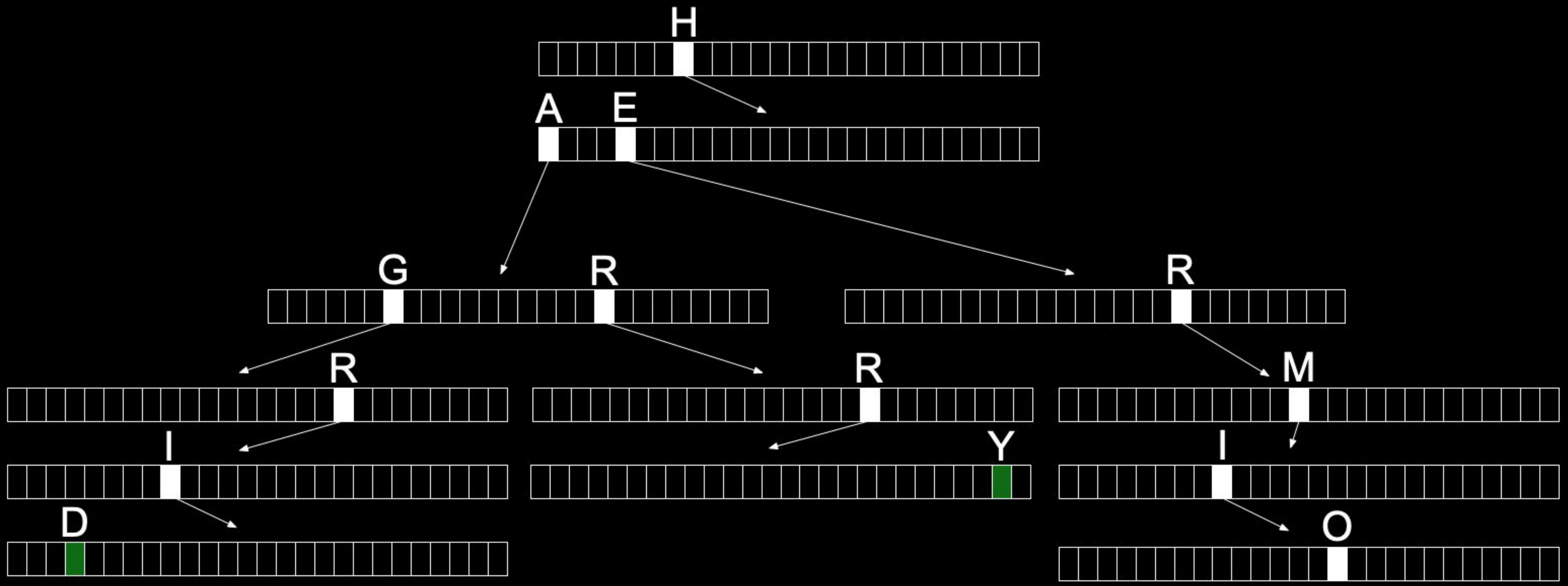

H
A
G
R
R
R
I
D

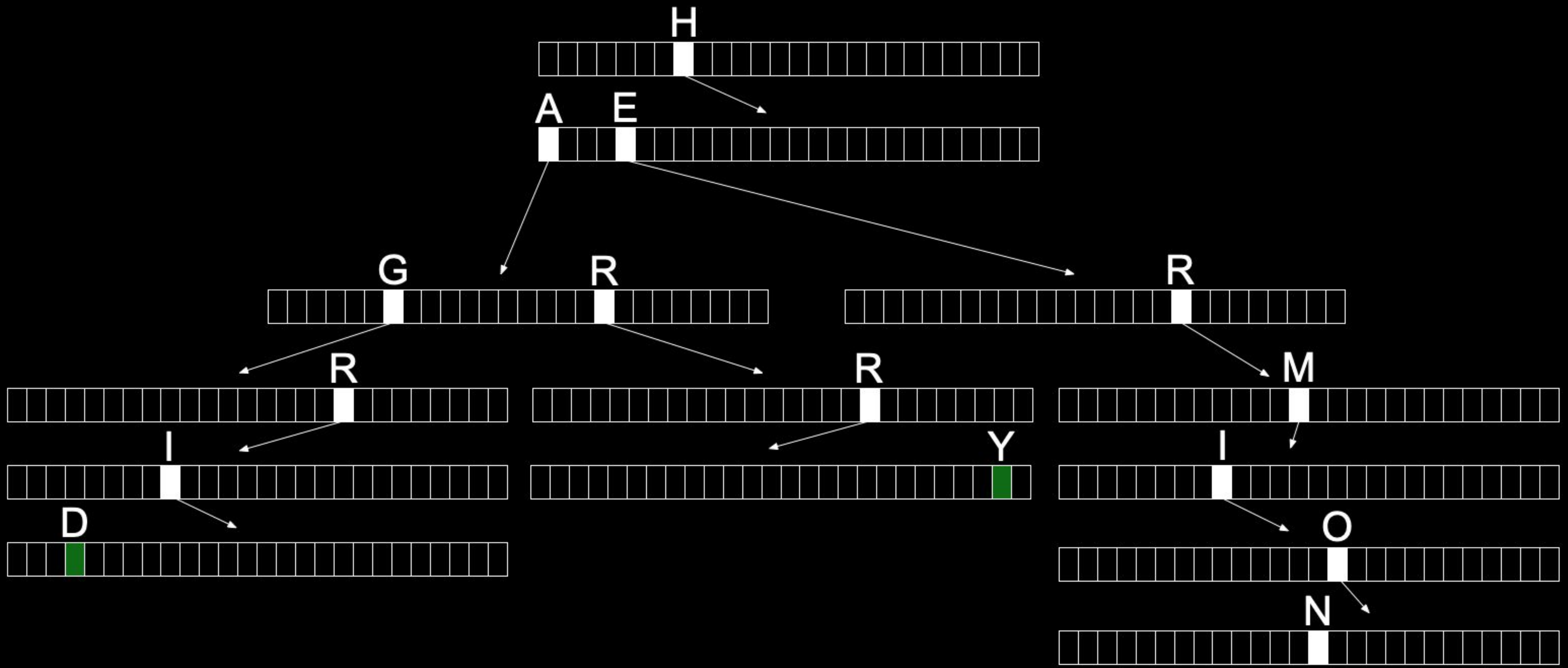

H
A
G
R
R
R
I
Y
D

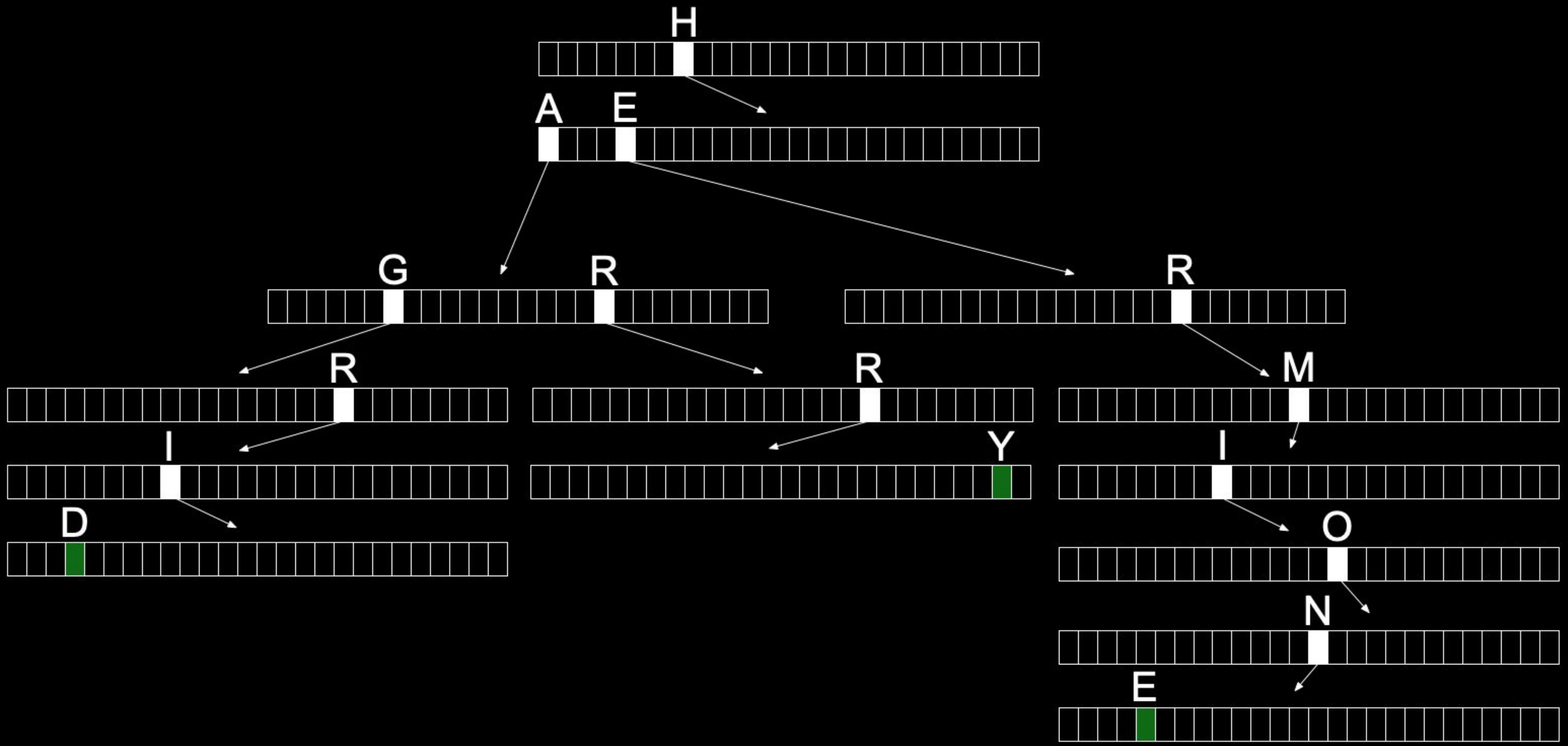

H
A
E
G
R
R
R
R
I
Y
D


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

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

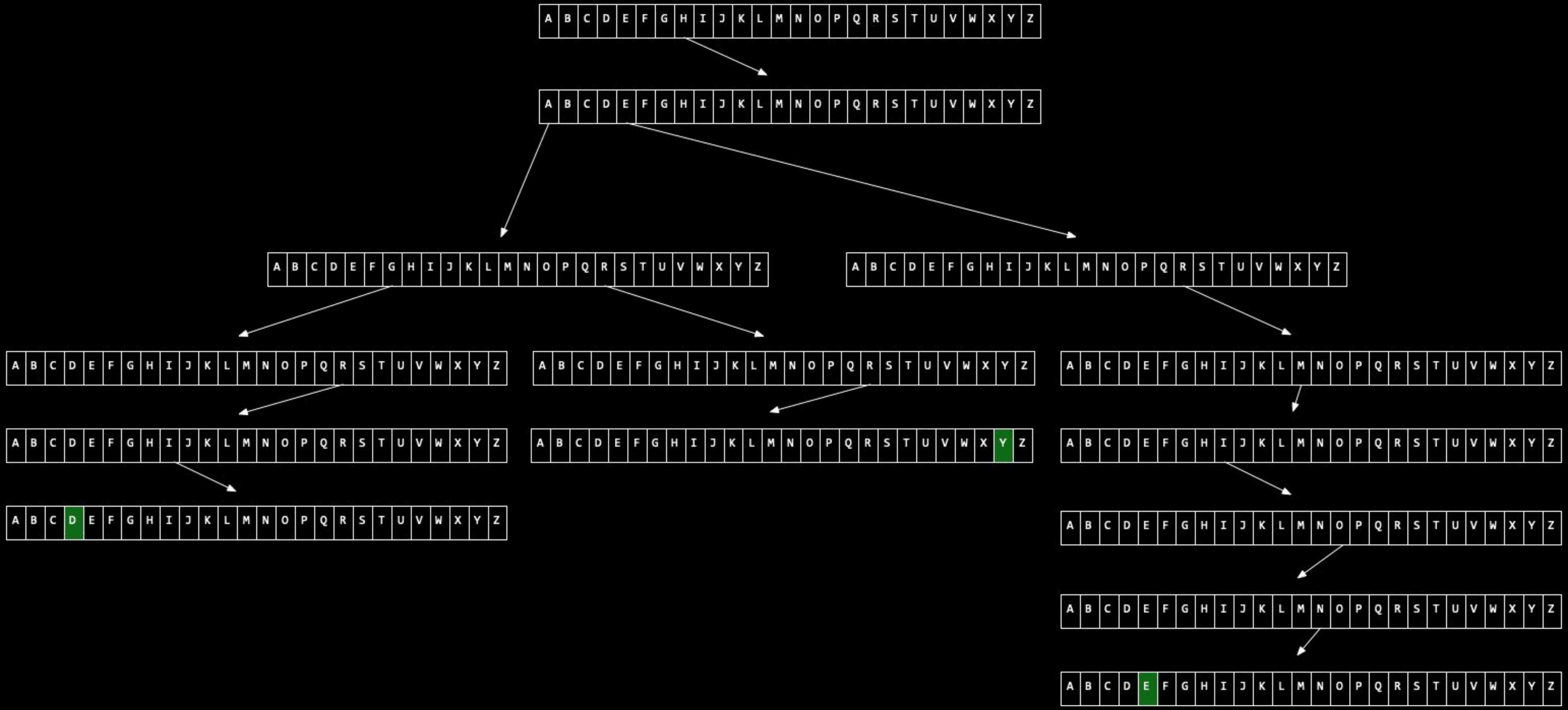
$O(1)$

$O(k)$ search

$O(k)$	search, insert

$O(1)$ search, insert

$O(n^2)$	
$O(n \log n)$	
$O(n)$	
$O(\log n)$	
$O(1)$	search, insert


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
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
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
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
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
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
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
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
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
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
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
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
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
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

abstract data structures

queues

FIFO

enqueue

dequeue

stacks

LIFO

push

pop

dictionaries

ICK ME UP
ServSafe CERTIFICATION
ServSafe CERTIFICATION
Safety And First-aid Education
FOOD ESTABLISHMENT PERMIT
B
C
D
E
F
G
H
I
K
L
M
N
O
P
Q
R
T
U-
V
W
X
Y
Z
CHOKIN

This is CS50