This is CS50

Section, Week 6

TA: Andi Peng

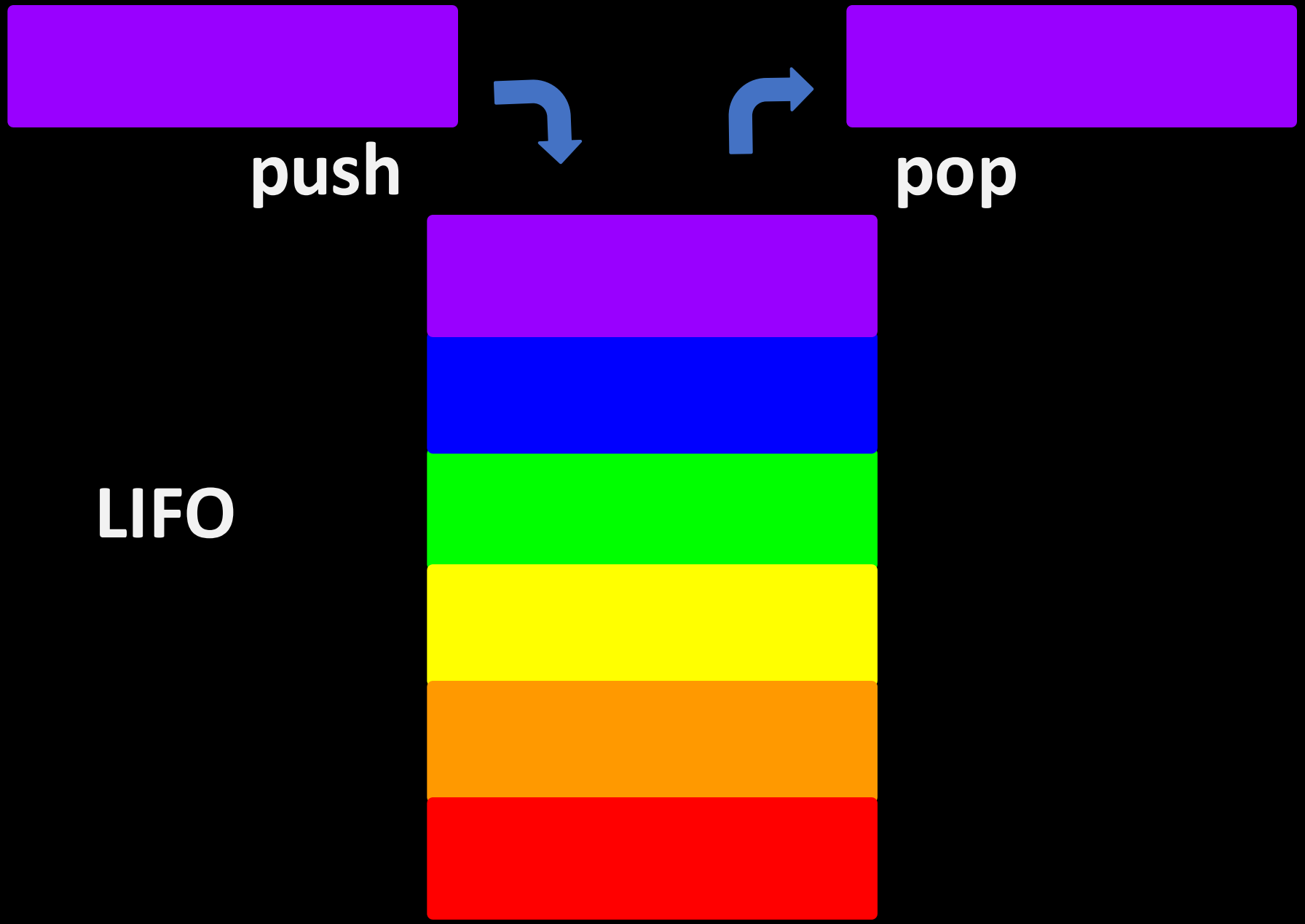
Agenda

- Announcements
- Stacks
- Queues
- Linked Lists (Again)
- Hash Tables
- Trees
- Tries
- pset5

Announcements

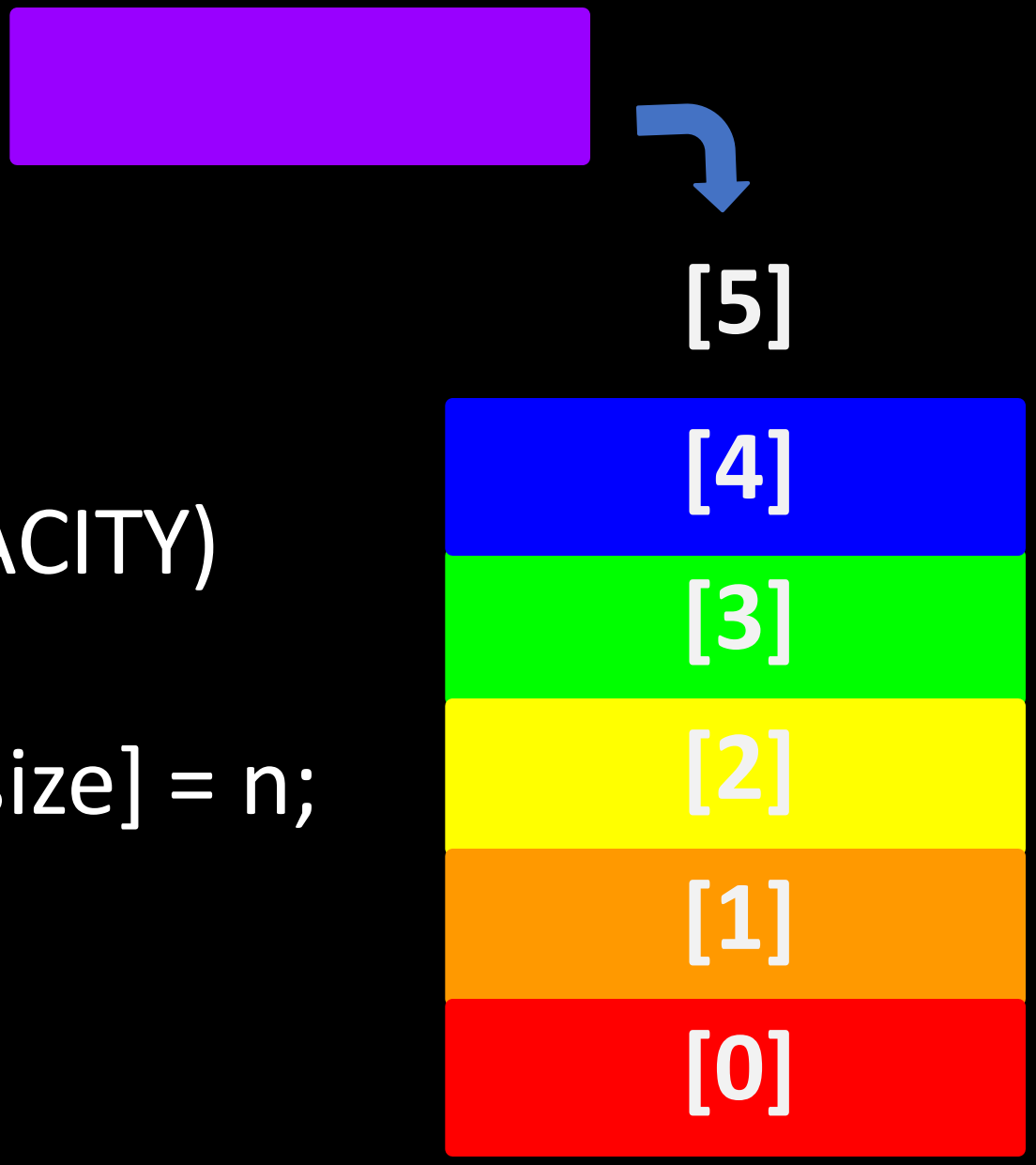
- Quiz 0
  - Feedback
  - Come get them at the end of section!
- pset5
  - Due Wednesday at noon
  - Office Hours Sunday and Monday
  - Workshops in sections

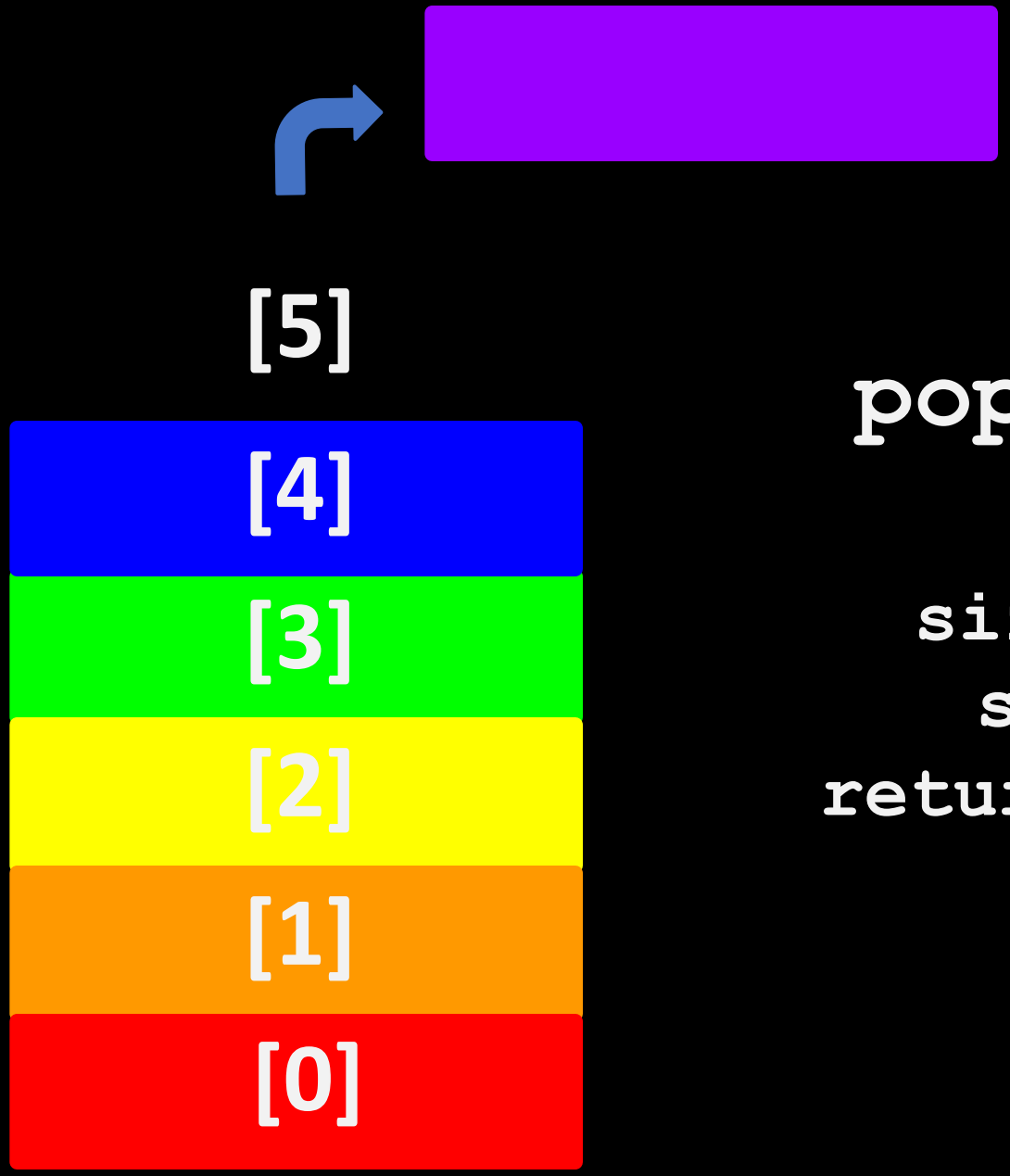
Stacks


push
pop
LIFO

```
typedef struct
{
  char* strings[CAPACITY];
  int size;
}
stack;
```

```
void push(int n)
{
        if (s.size != CAPACITY)
        {
                s.strings[s.size] = n;
                s.size++;
        }
}
```



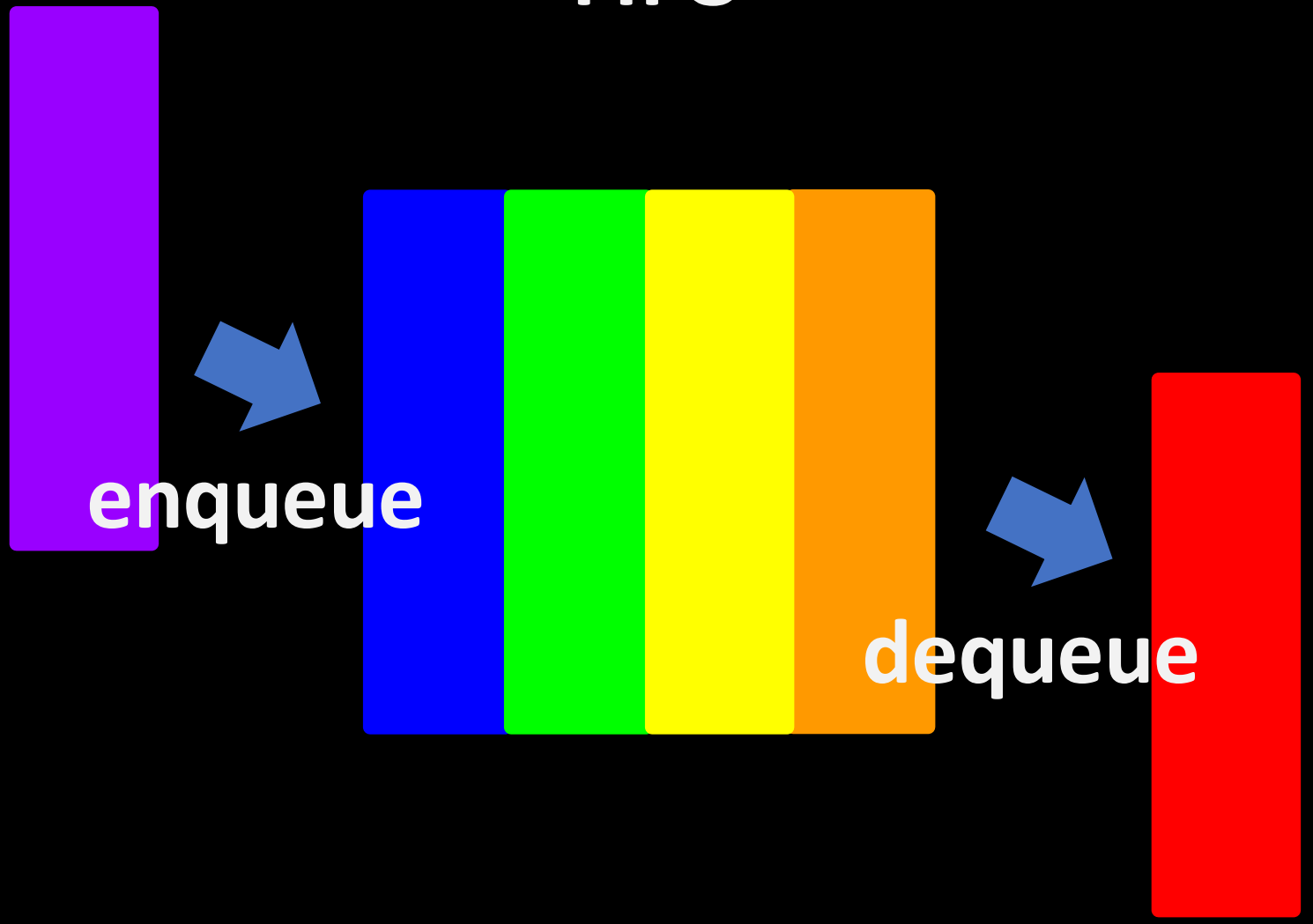


pop TODOs:

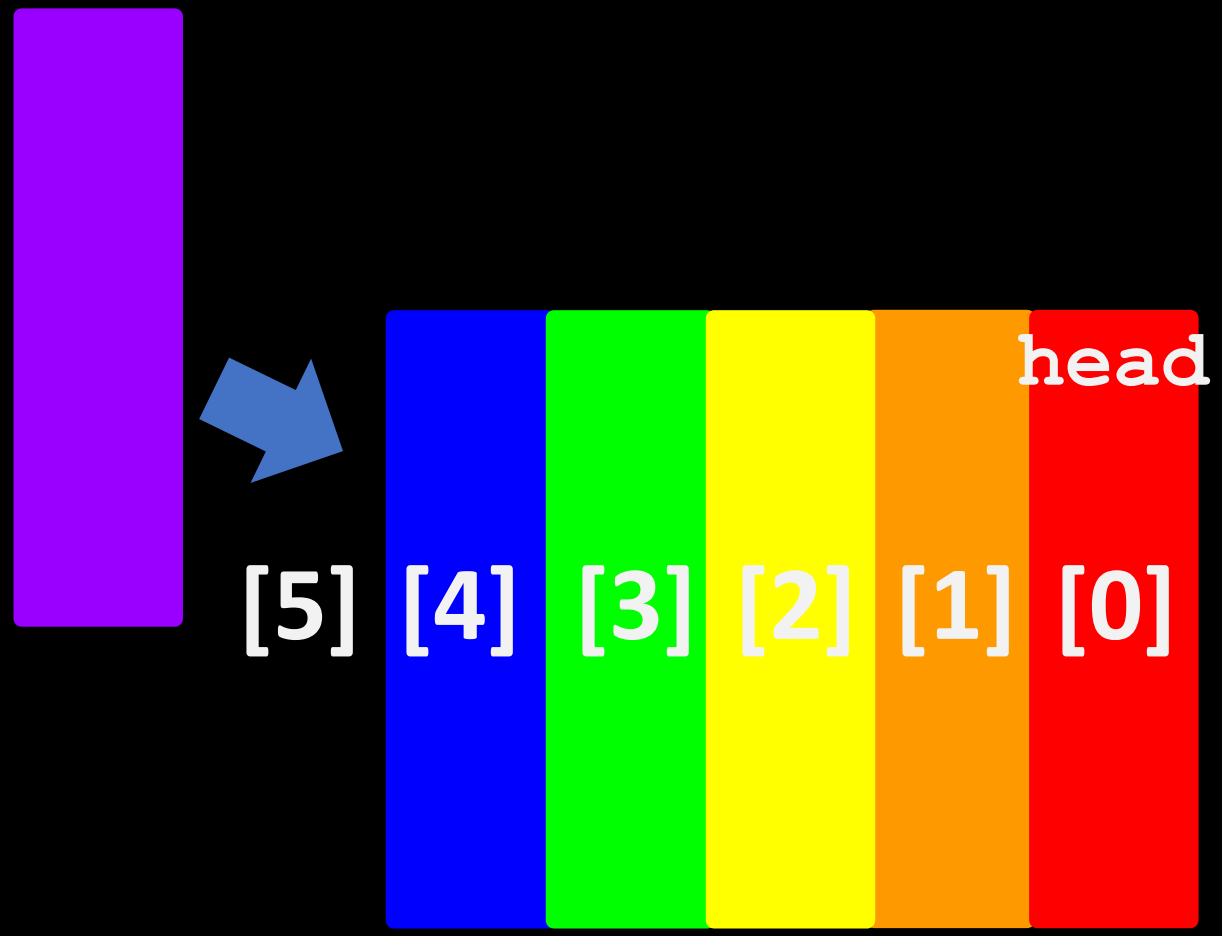
```
size > 0?
size--
return [size]
```

Queues

FIFO



```
typedef struct
{
    int head;
    char* strings[CAPACITY];
    int size;
}
queue;
```




```
void enqueue(int n)
{
        if (q.size != CAPACITY)
        {
                q.strings[(q.head+
q.size) % CAPACITY] = n;
                q.size++;
        }
 }
```

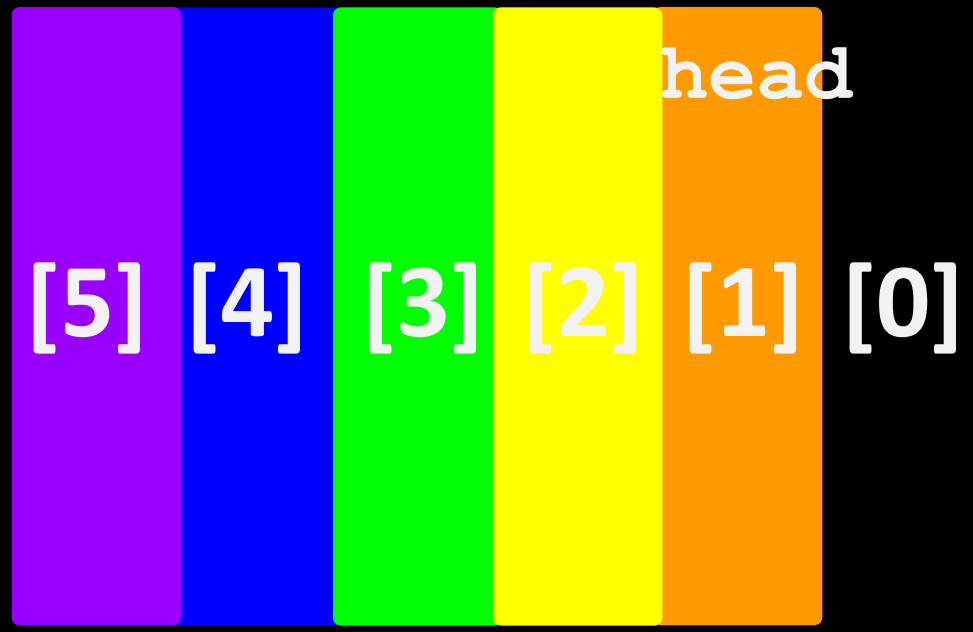
Dequeue TODOs:

`size > 0?`

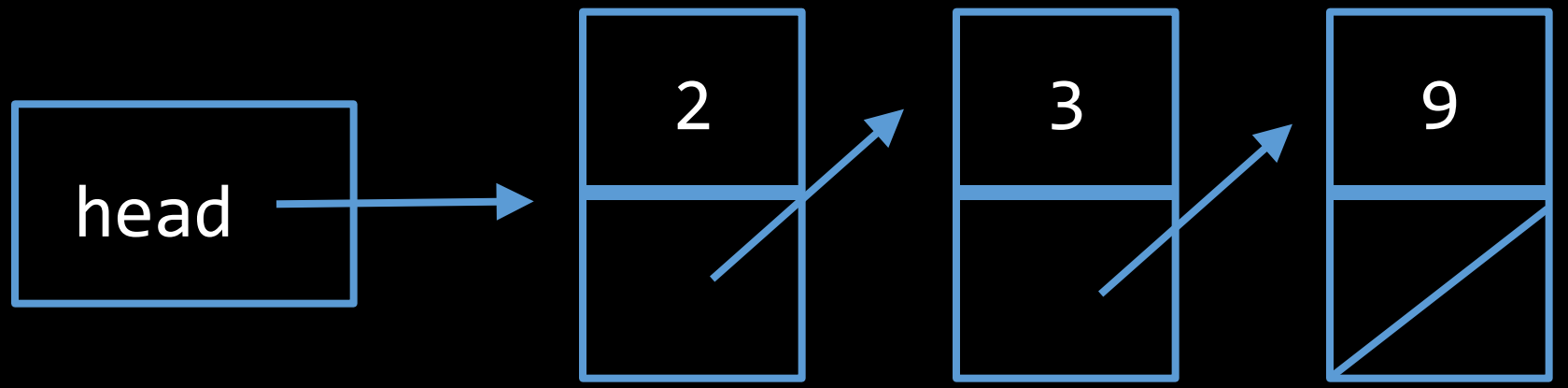
move `head`

`size--`

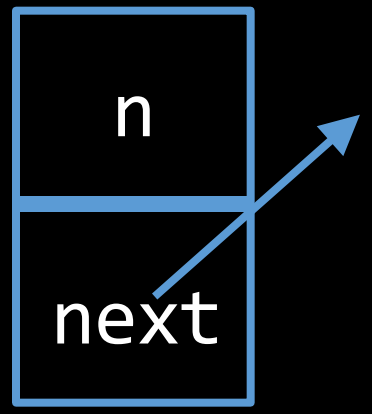
return element



Linked Lists

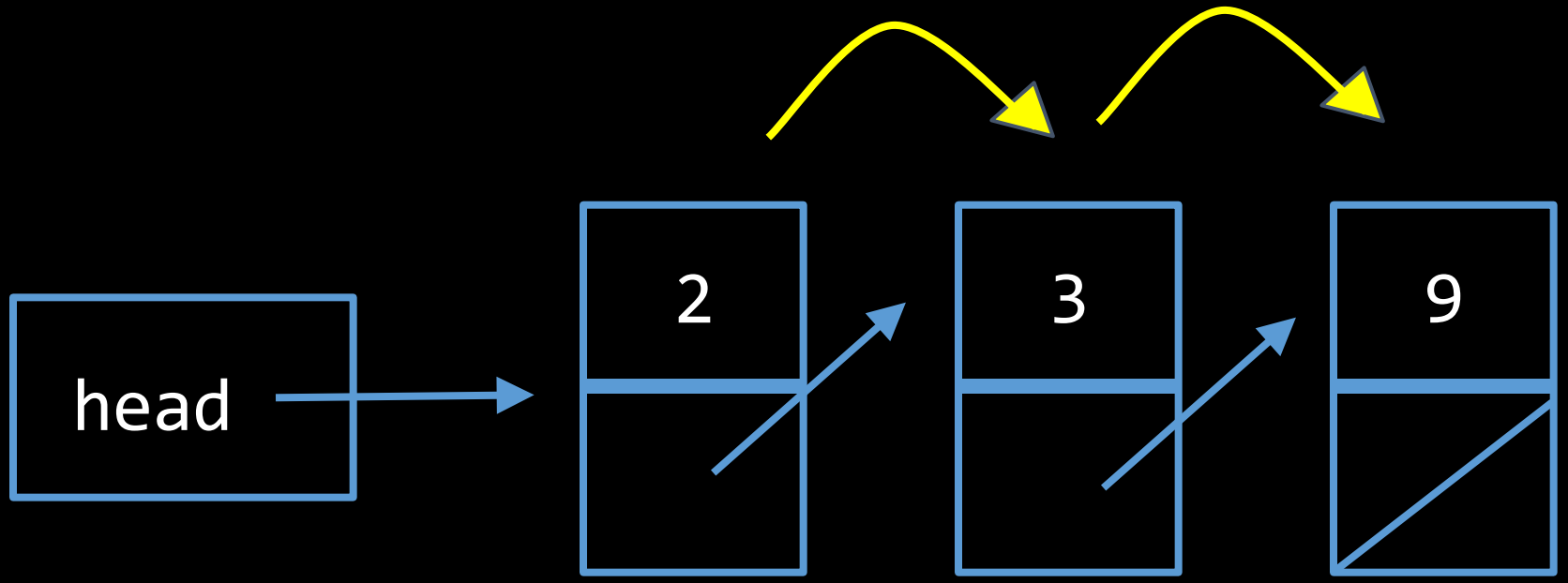


Nodes



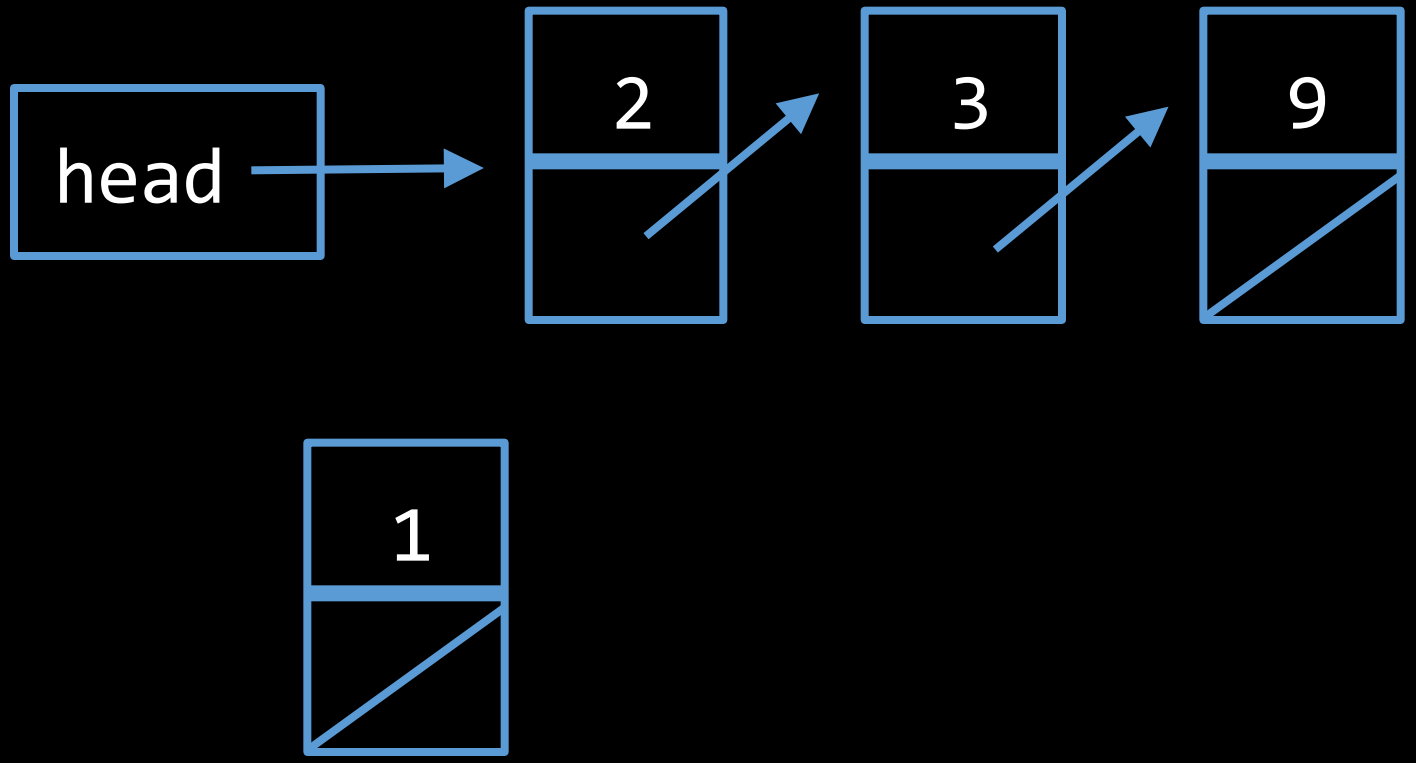

```
typedef struct node
{
        int n;
        struct node* next;
}
node;
```

Search

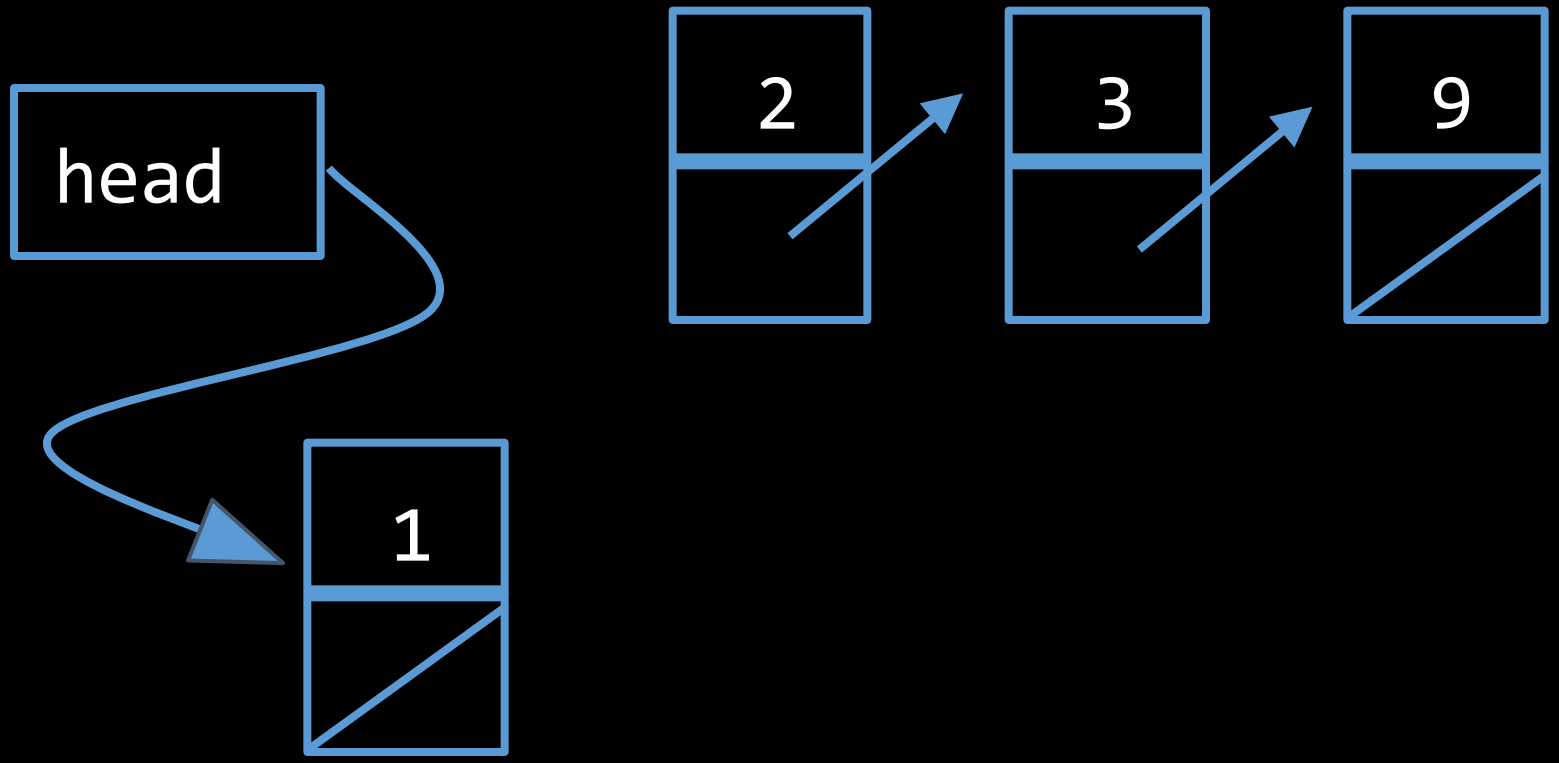


```
bool search(node* list, int n)
{
    node* ptr = list;

    while (ptr != NULL)
    {
            if (ptr->n == n)
      {
                return true;
      }
            ptr = ptr->next;
    }
    return false;
}
```

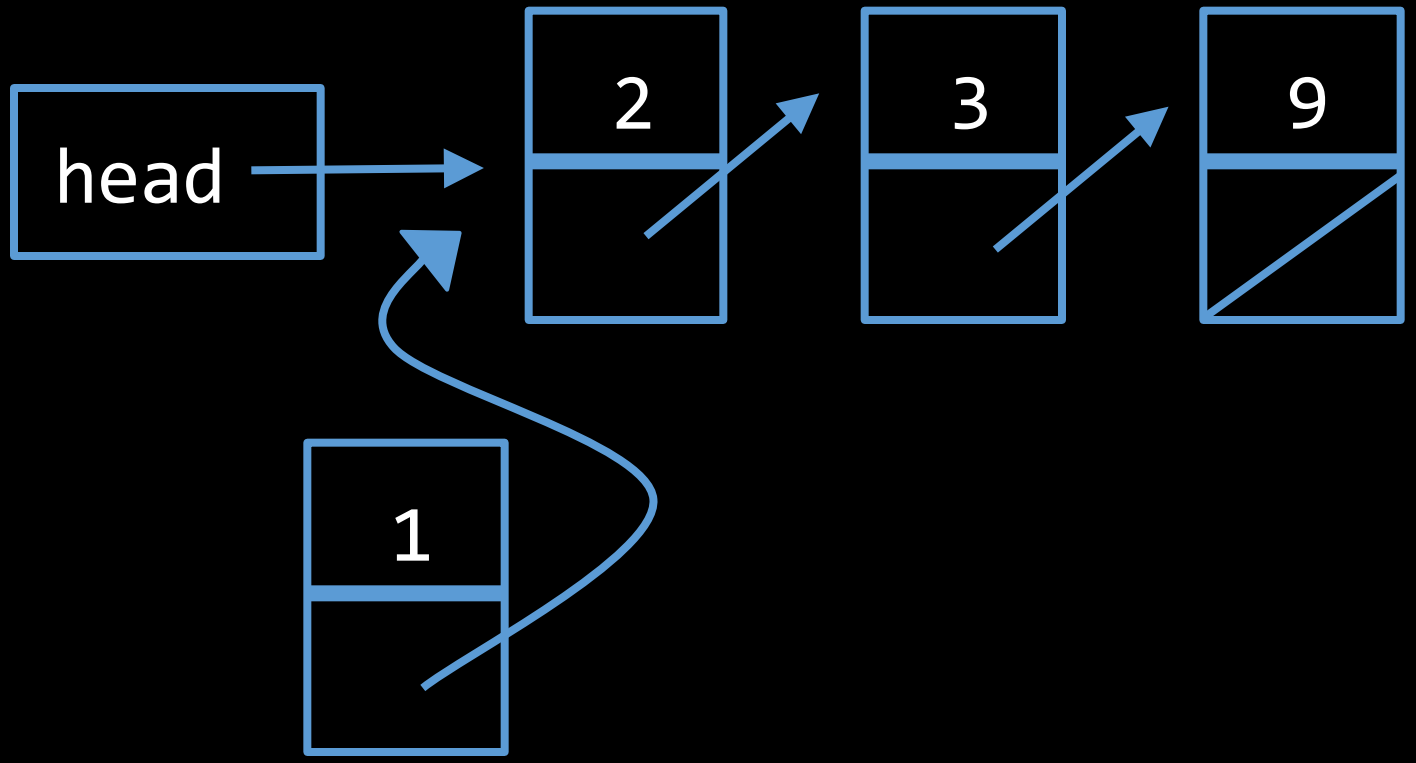
Insertion



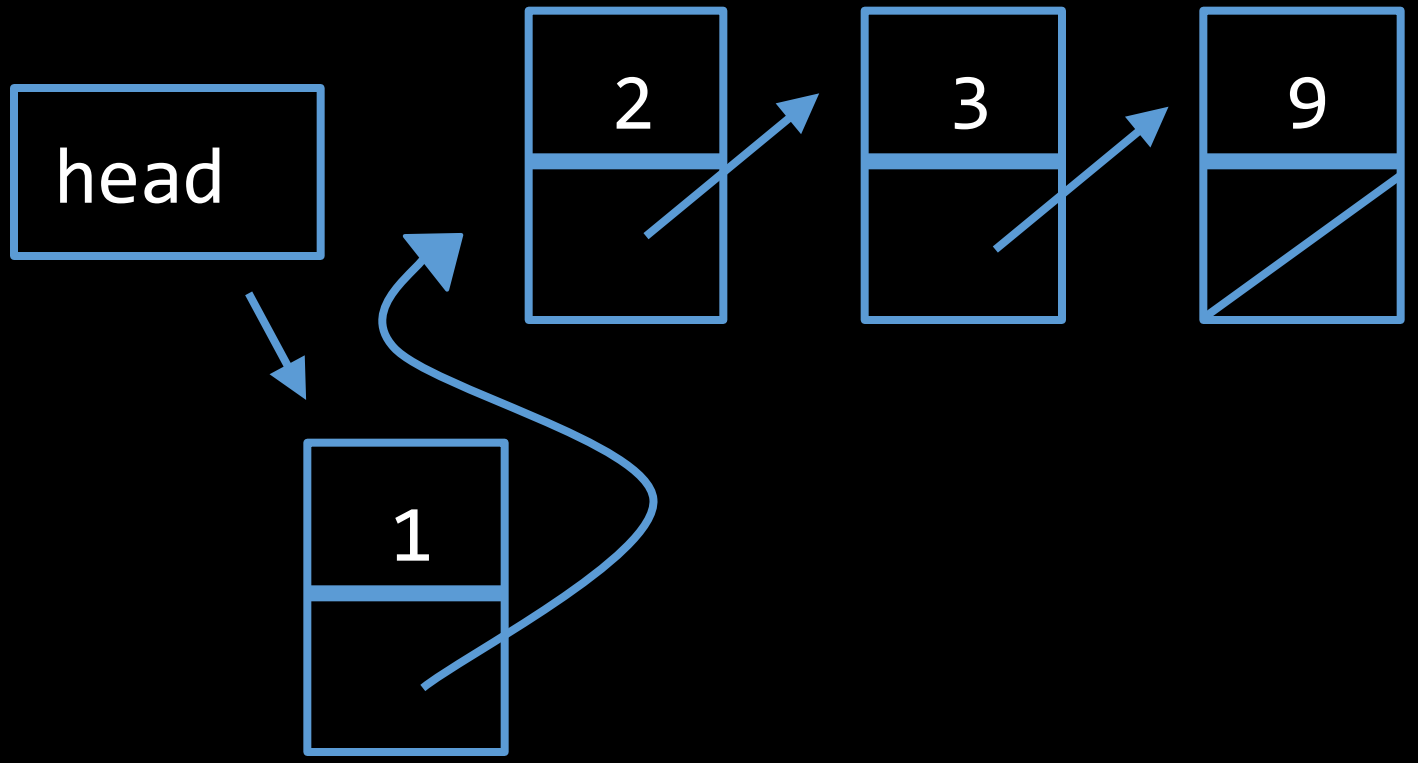
Insertion



Insertion



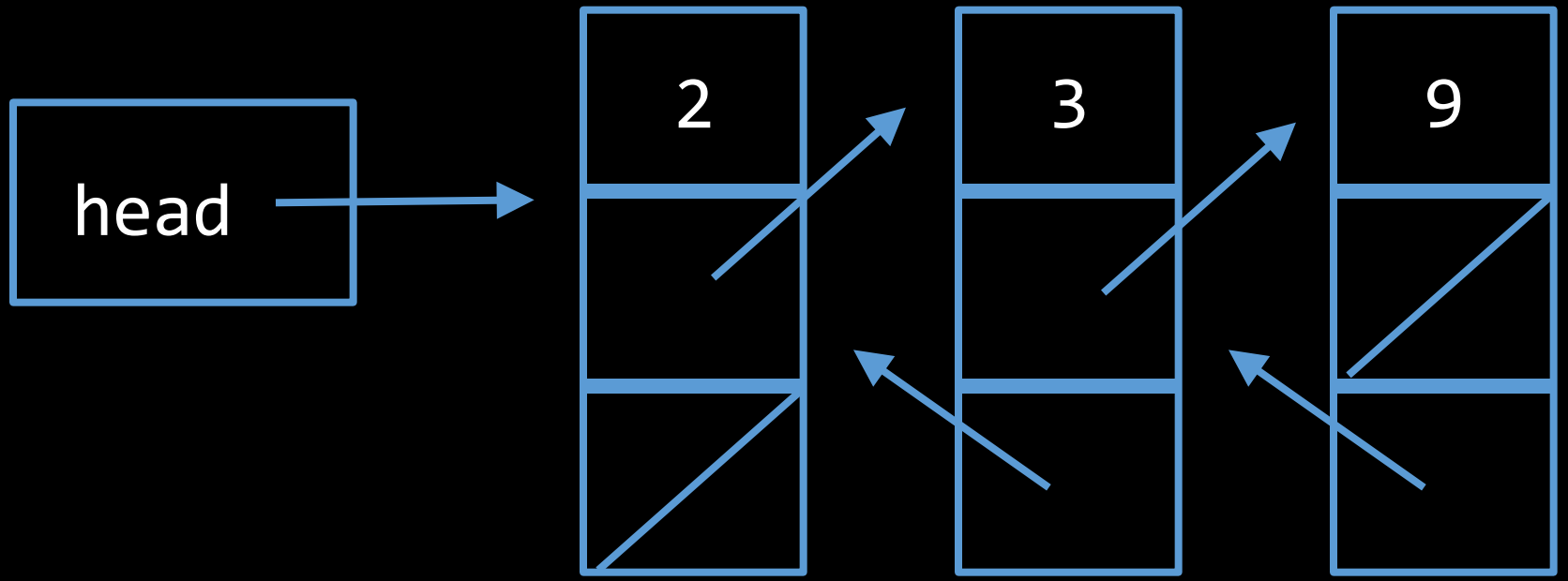
Insertion



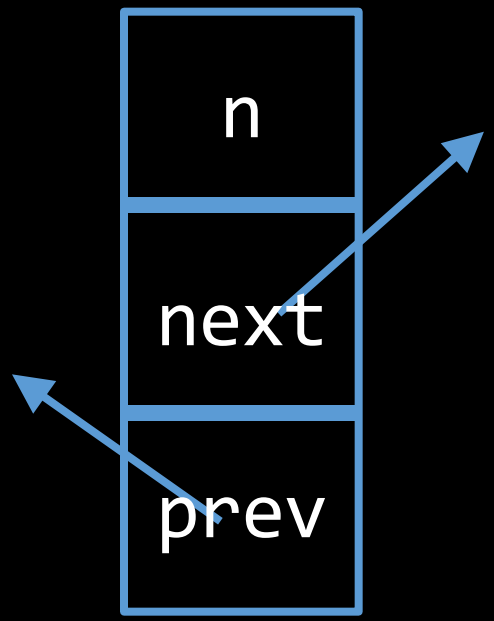
```c
void insert(int n)
{
    // create new node
    node* new = malloc(sizeof(node));

    // check for NULL
    if (new == NULL)
    {
        exit(1);
    }
    // initialize new node
    new->n = n;
    new->next = NULL;

    // insert new node at head
    new->next = head;
    head = new;
}
```

Inserting into the middle or the end?

Doubly Linked Lists

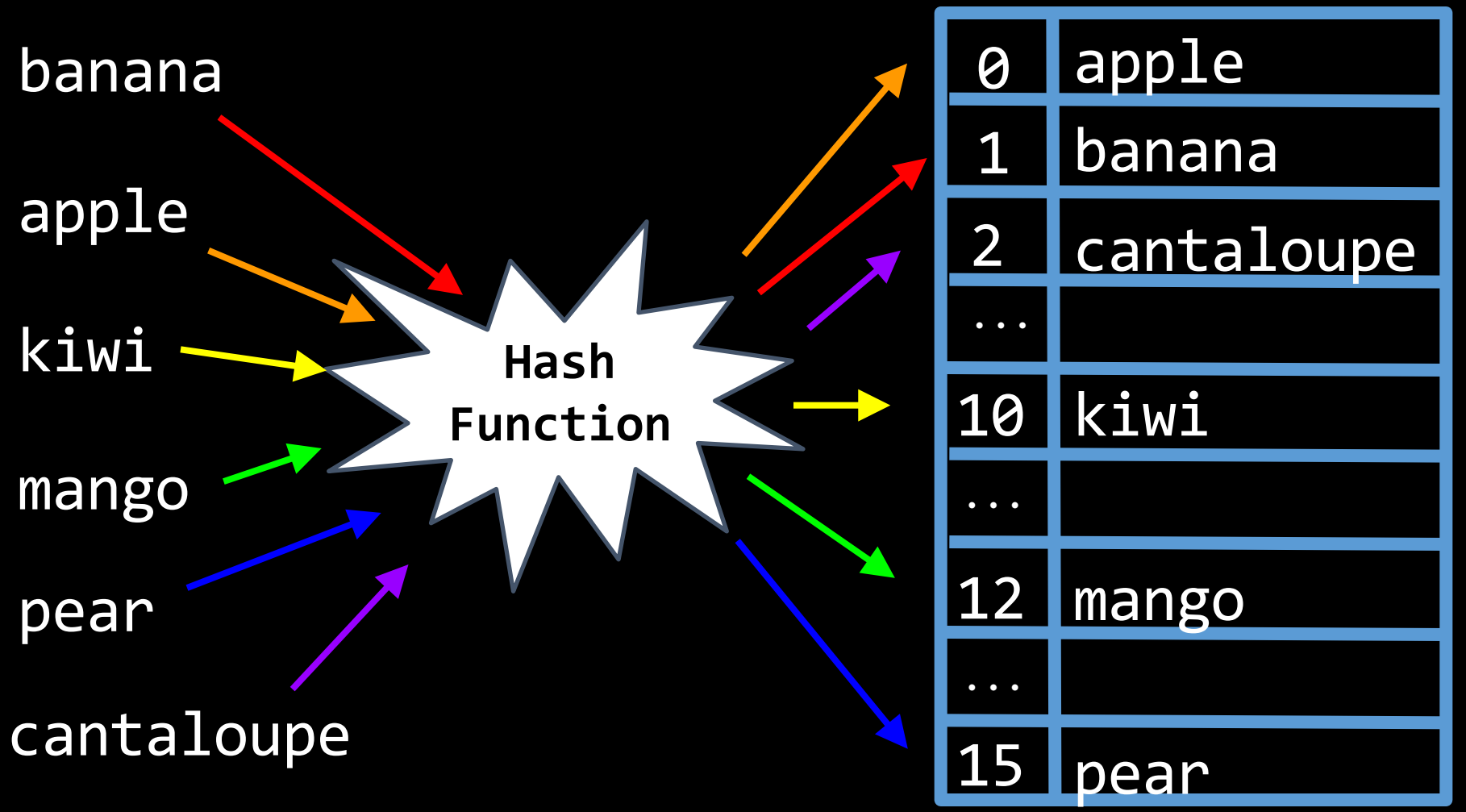


DLL Nodes

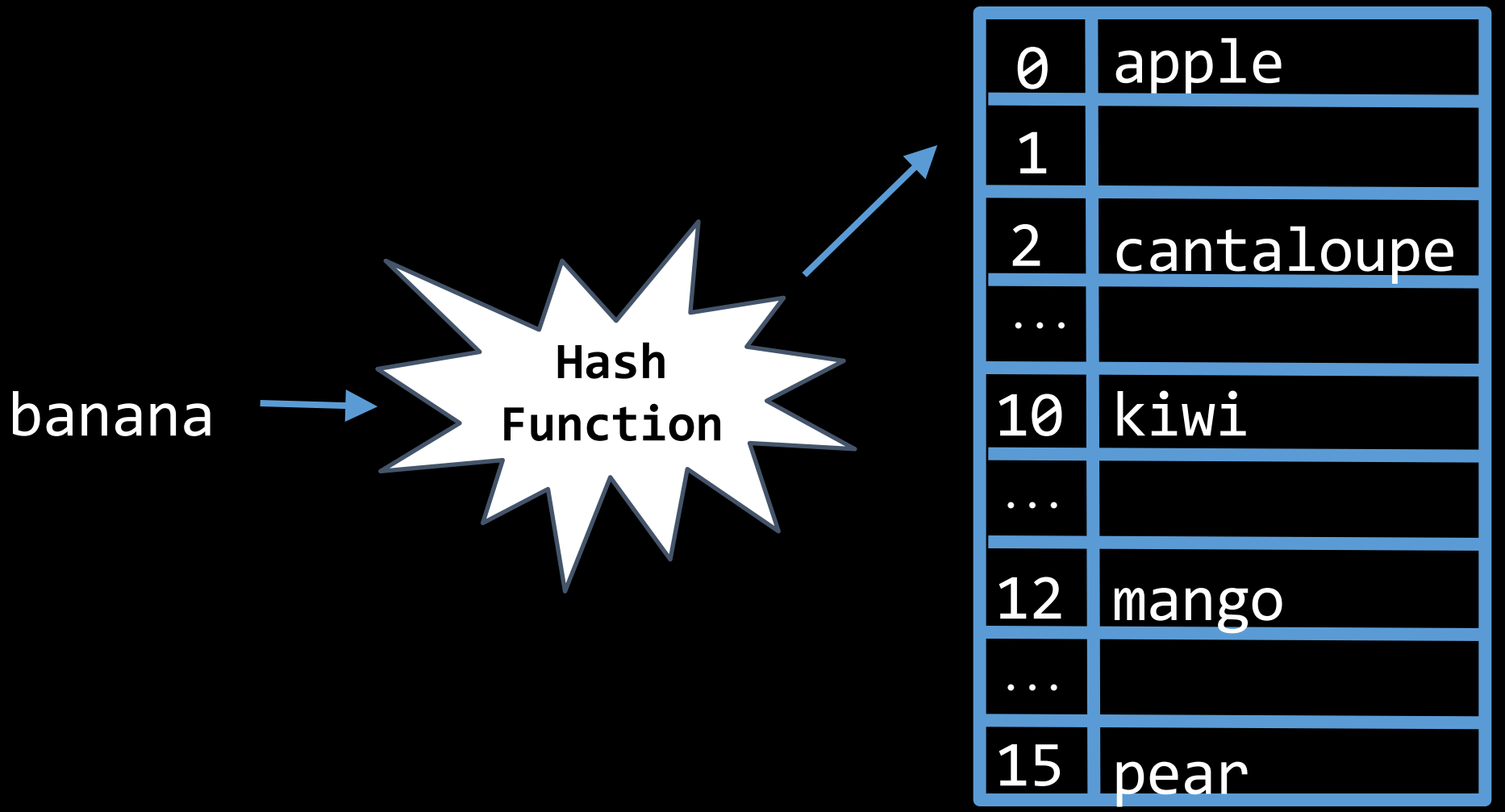



```
typedef struct  node
{
        int n;
        struct node* next;
        struct node* prev;
}
node;
```

Hash Tables


banana
apple
kiwi
mango
pear
cantaloupe
Hash Function
0 apple
1 banana
2 cantaloupe
...
10 kiwi
...
12 mango
...
15 pear

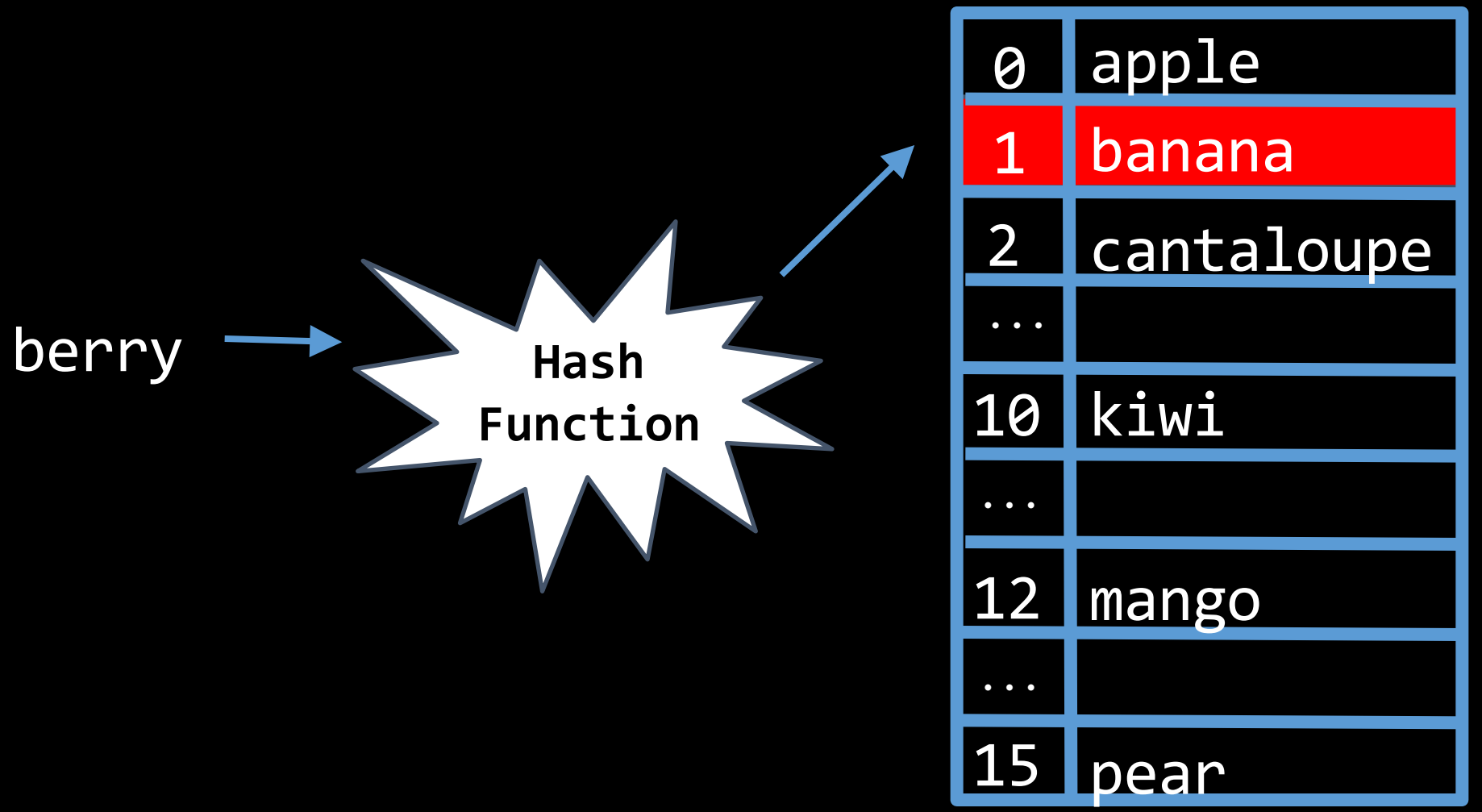
Hash Function



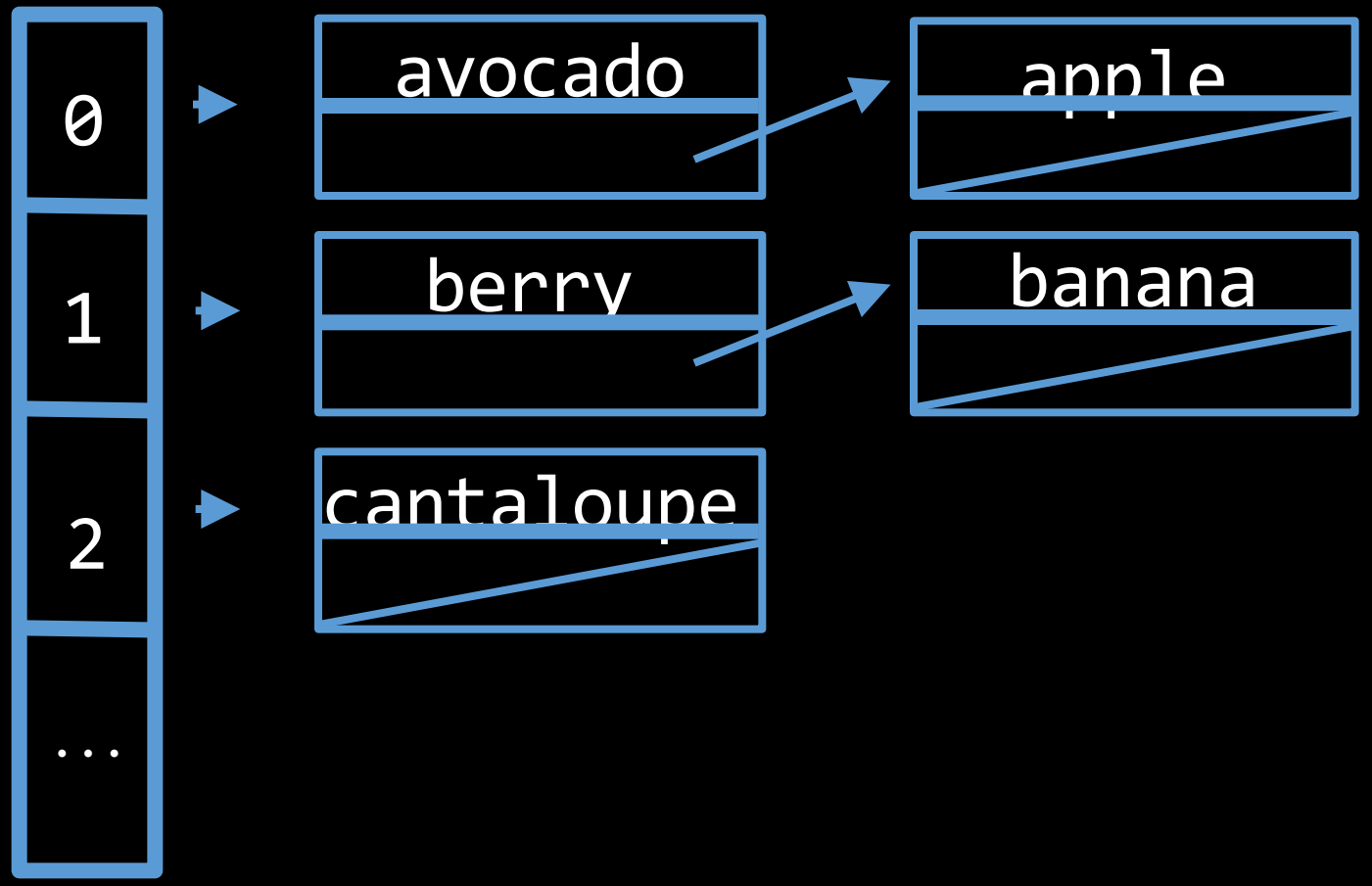
Hash Function Example

```
int hash_function(char* key)
{
        // hash on first letter of string
        int hash = toupper(key[0]) - 'A';

    return hash % SIZE;
}
```

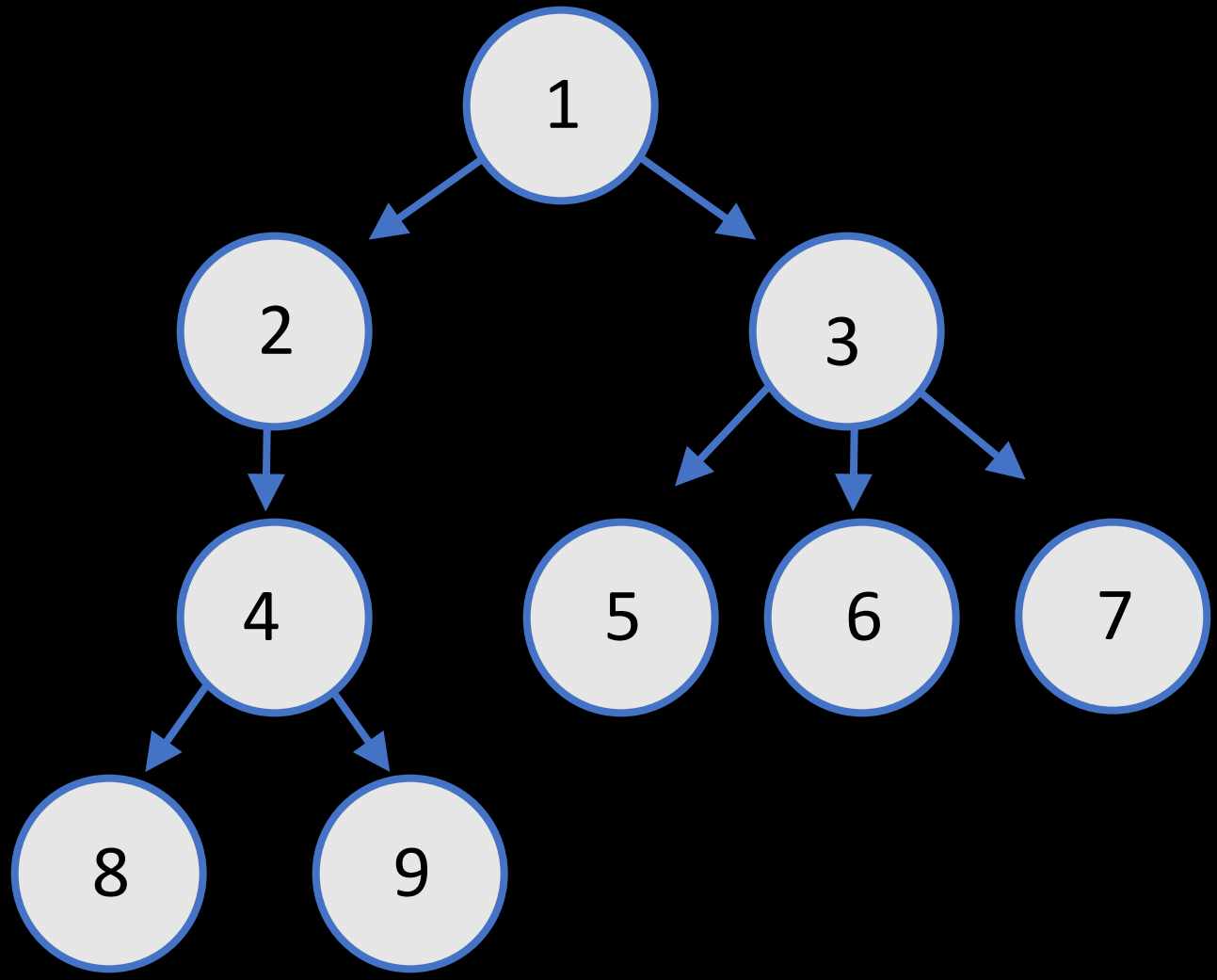
Collisions

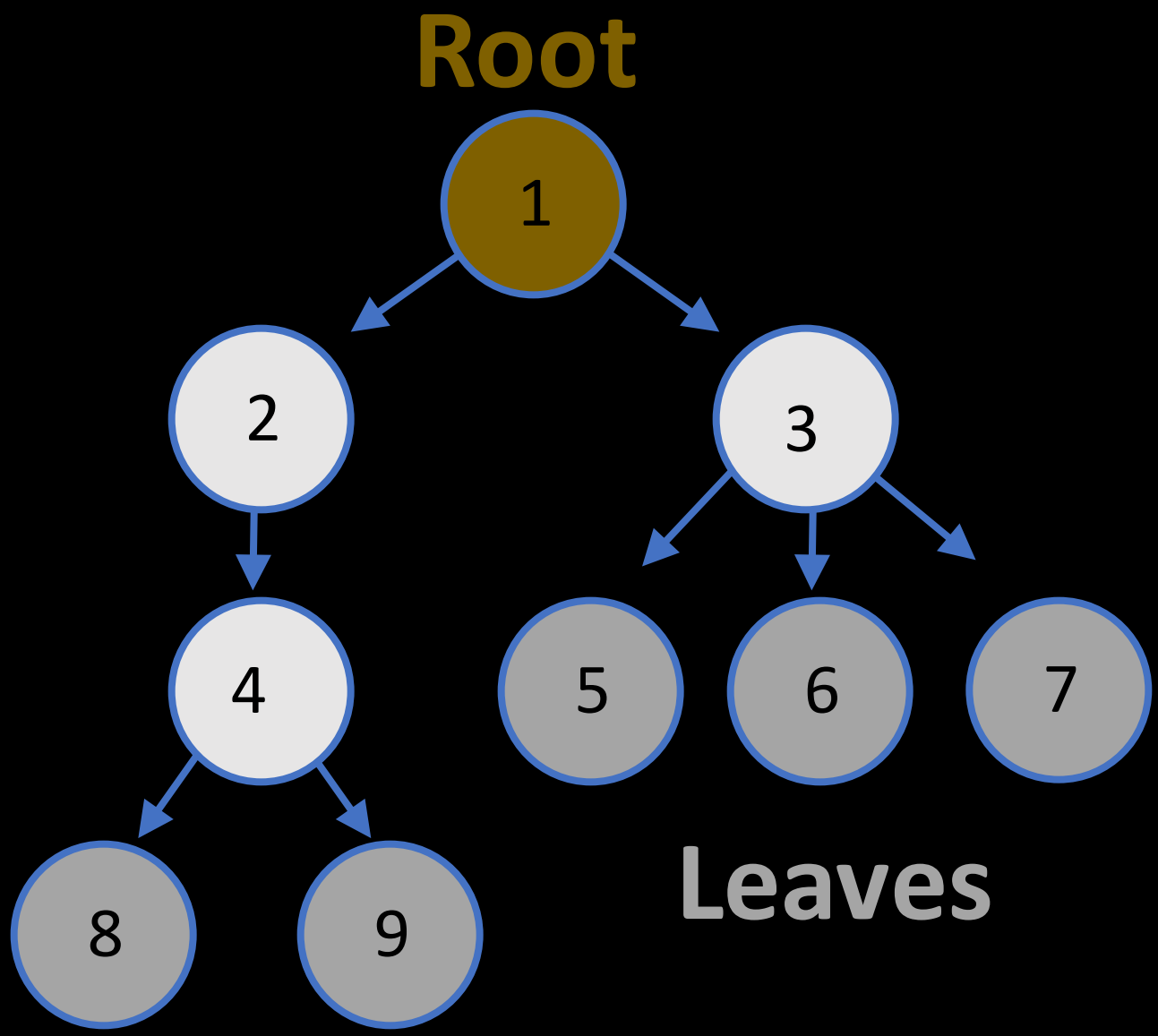

berry
Hash Function
0 apple
1 banana
2 cantaloupe
...
10 kiwi
...
12 mango
...
15 pear

Separate Chaining

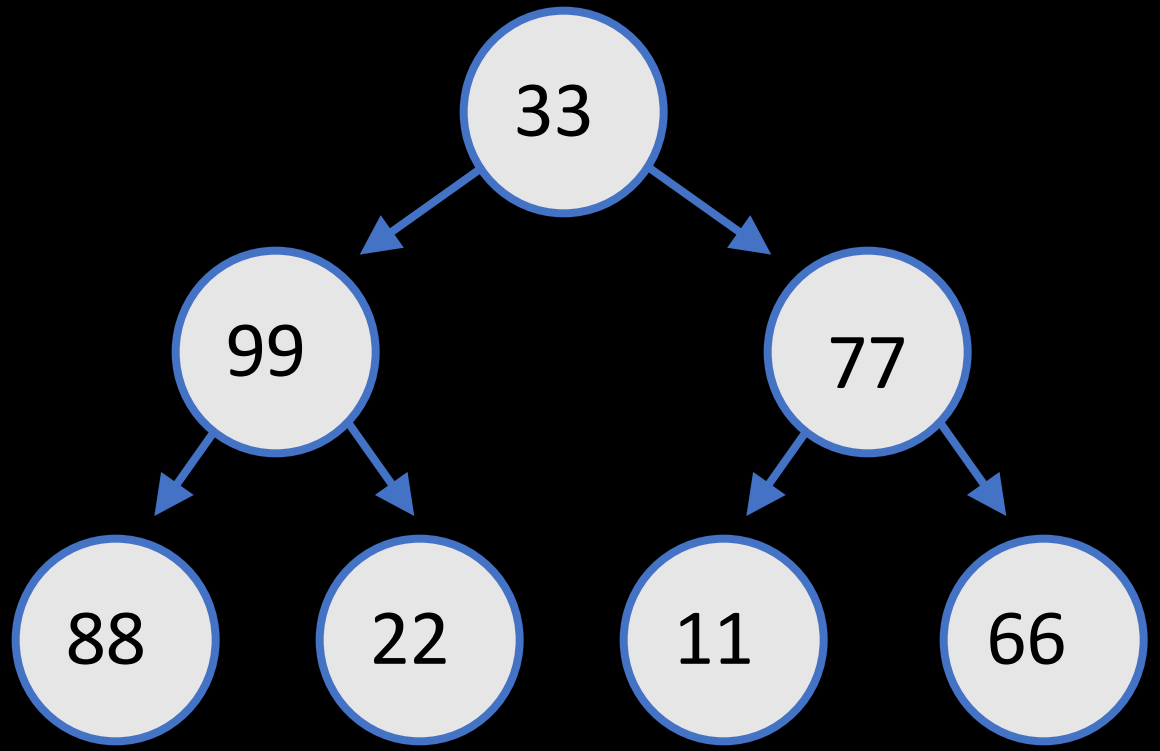


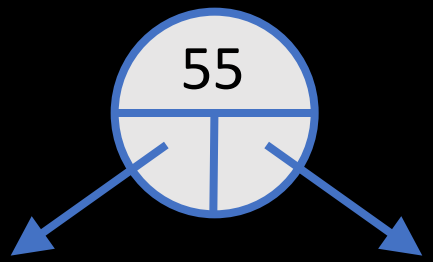
Tree




Root
1
2
3
4
5
6
7
8
9
Leaves

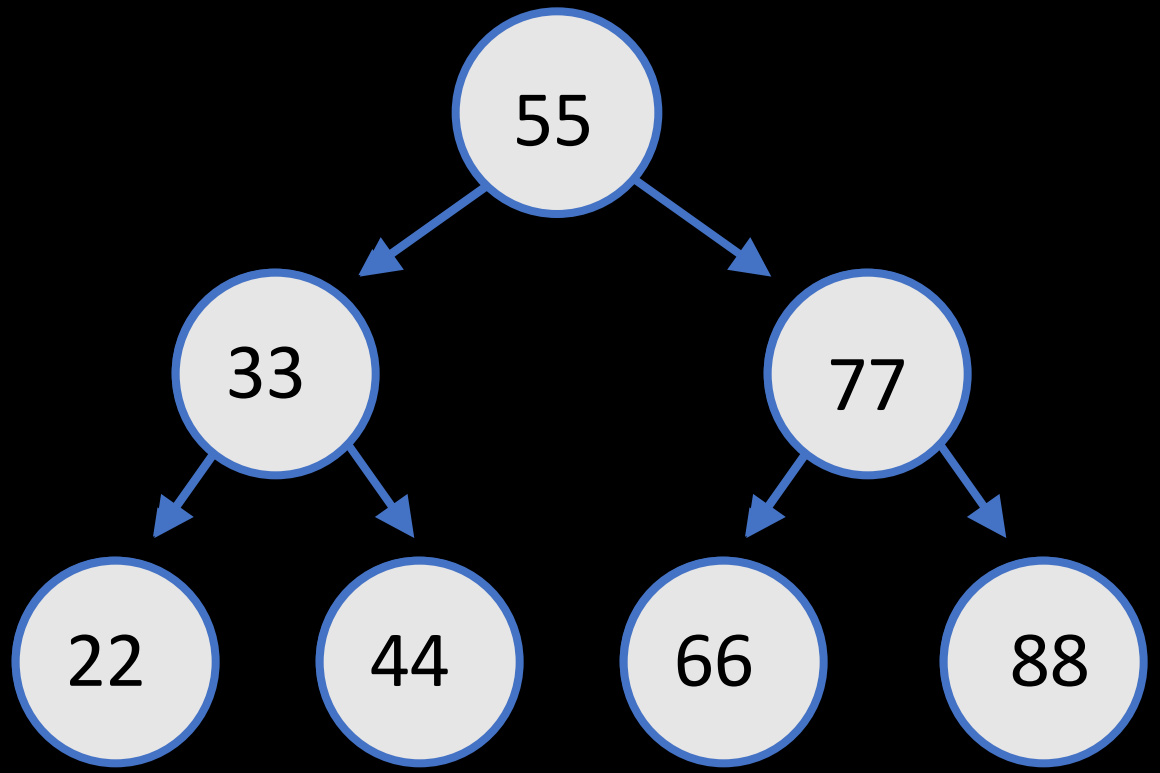
Binary Tree






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

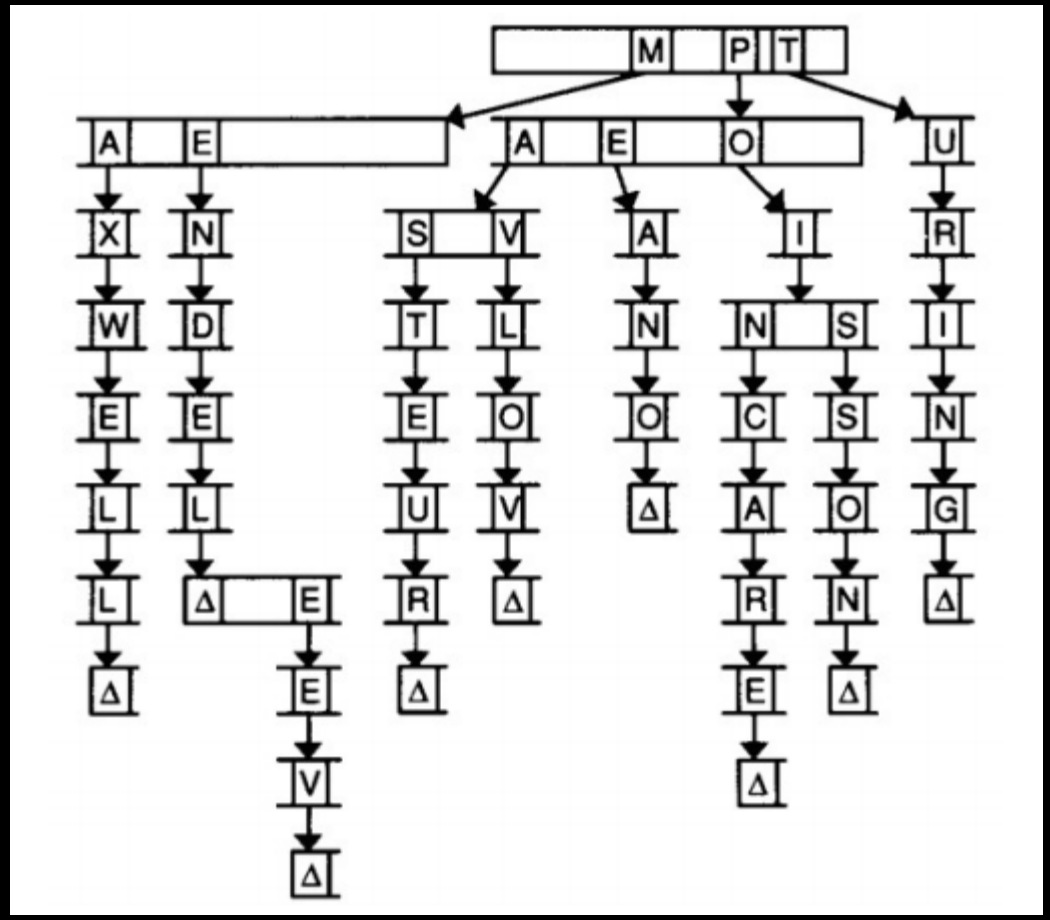
Binary Search Tree



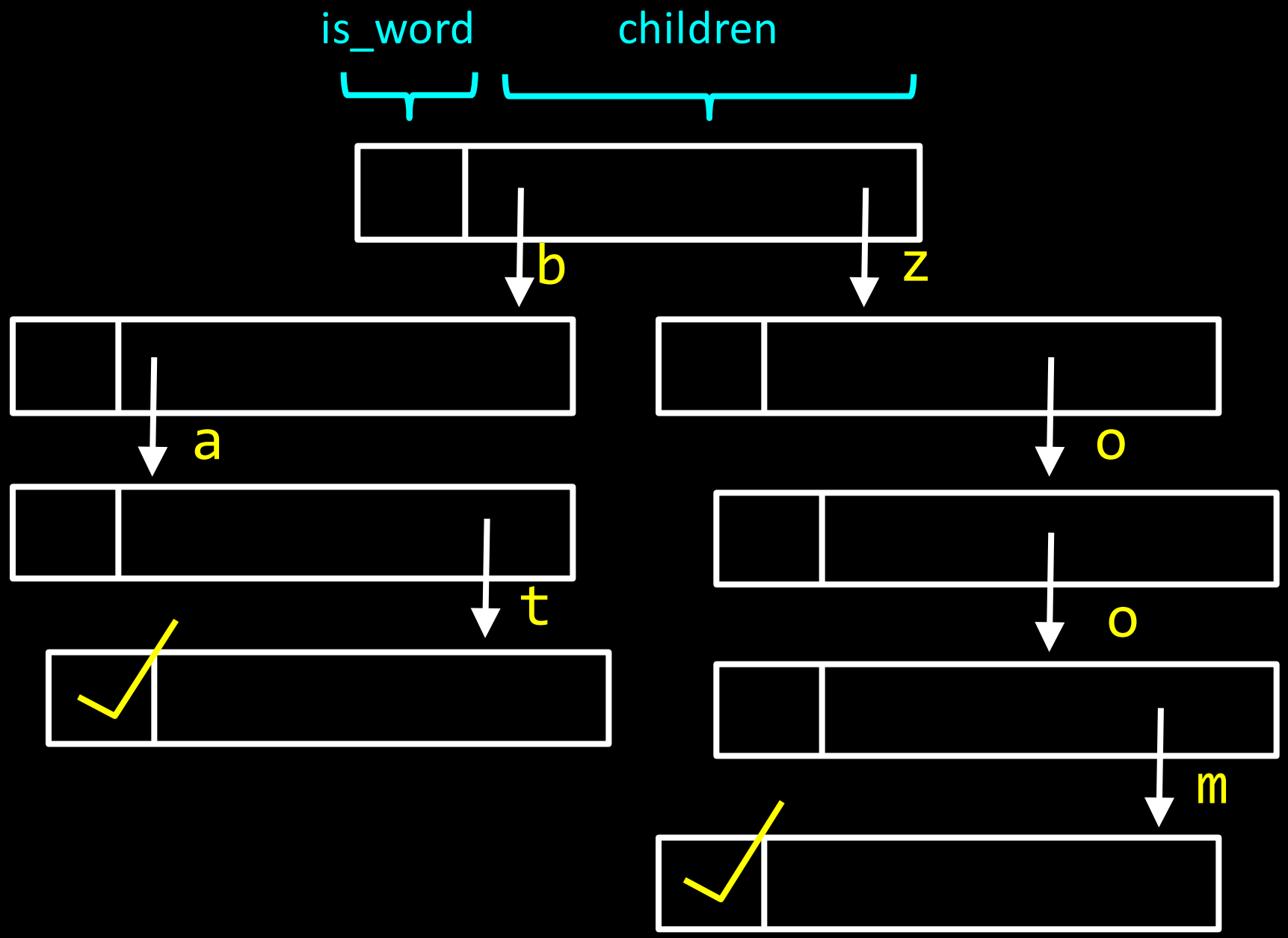
Pseudocode for search!

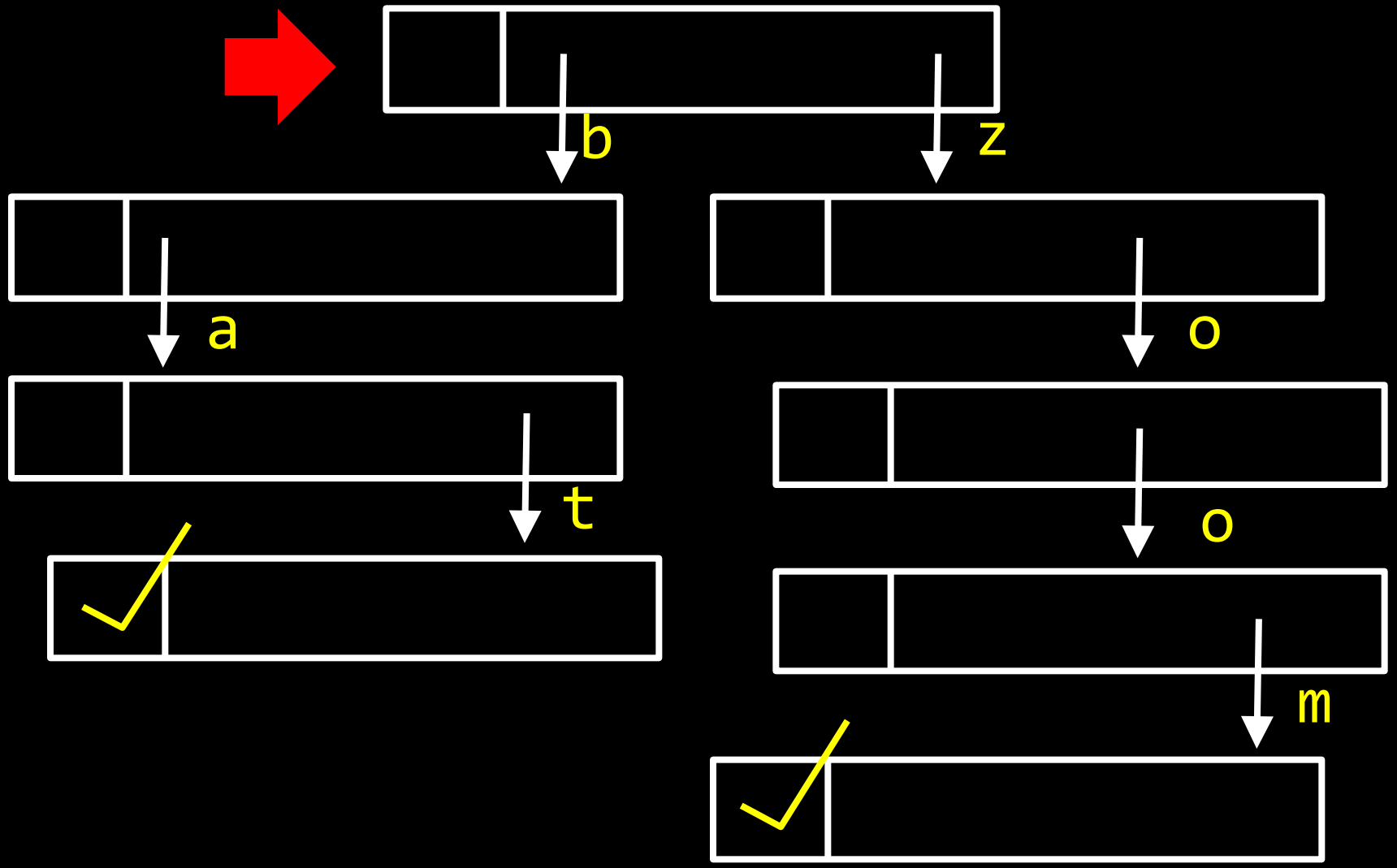
```
bool search(node* root, int val)
{
    if root is NULL
        return false.

    if root->n is val
        return true.

    if val is less than root->n
        search left child

    if val is greater than root->n
        search right child
}
```

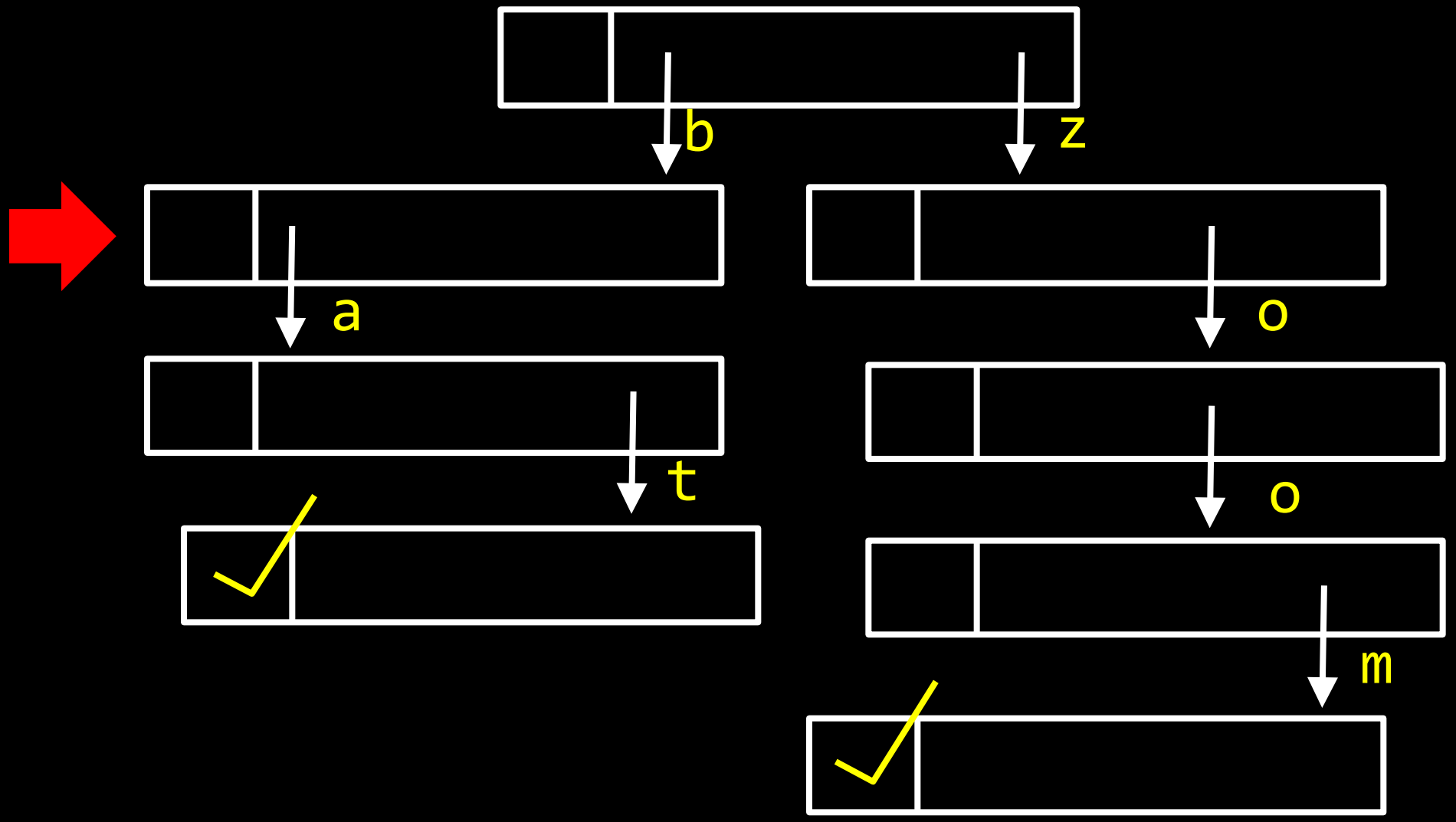
Tries

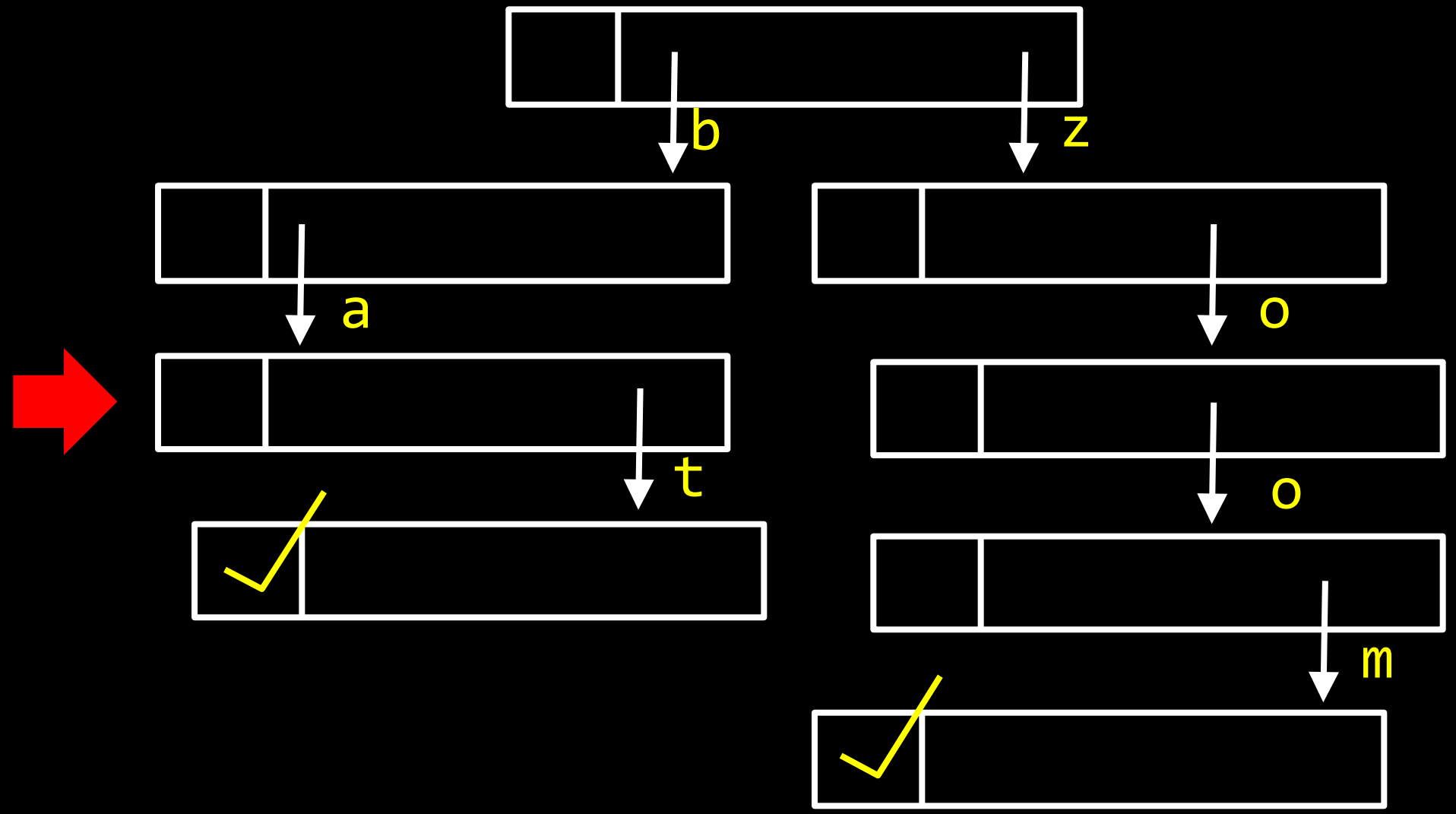

M P T
A E
A E O
U
X N
S V
A
I
R
W D
T L
N
N S
I
E E
E O
O
C S
N
L L
U V
Δ
A O
G
L Δ E
R Δ
R N Δ
Δ E
Δ
E Δ
V
Δ

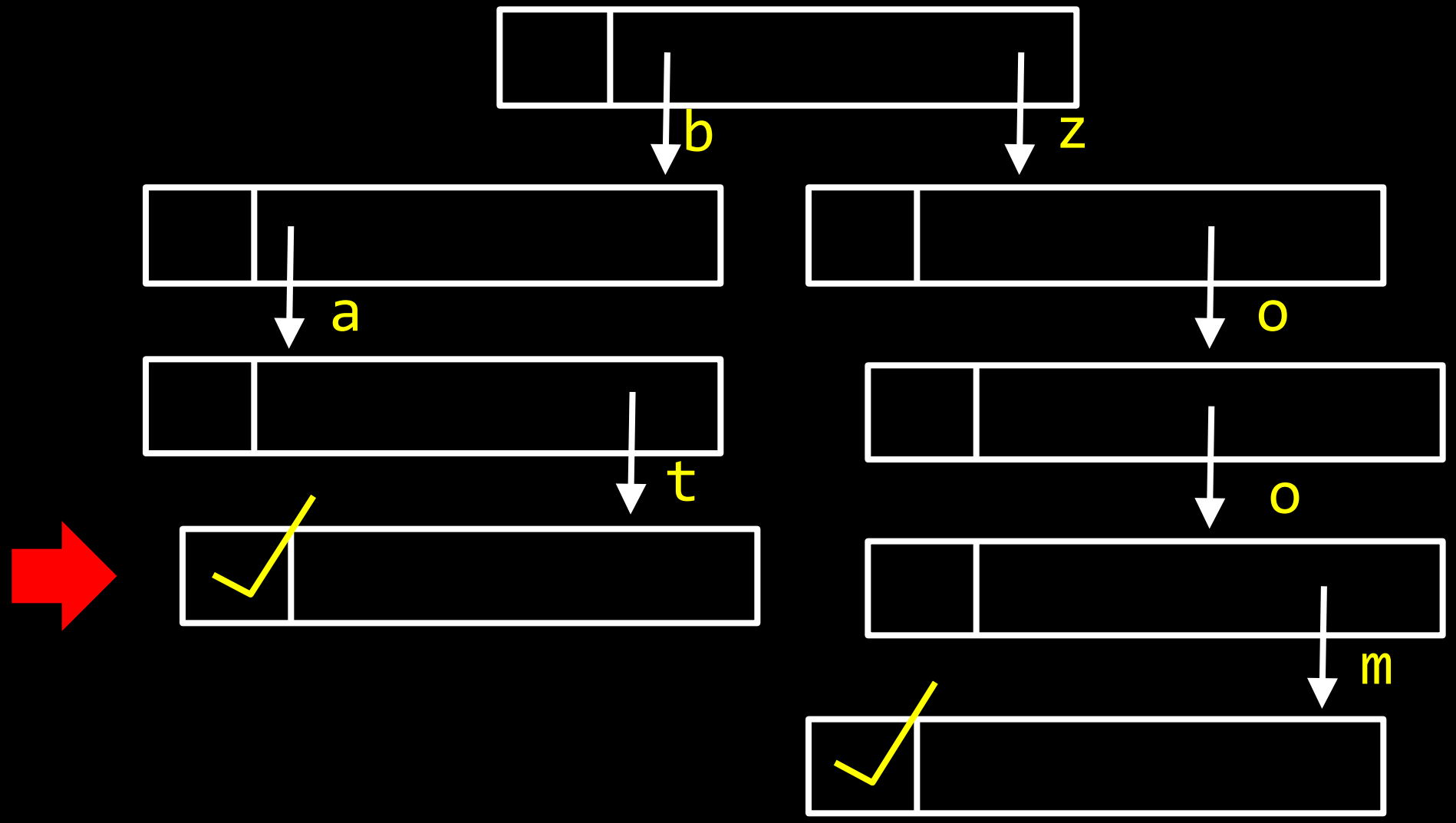
```
typedef struct node
{
    // marker for end of word
     bool is_word;

     // pointers to other nodes
     struct node* children[27];
}
node;
```

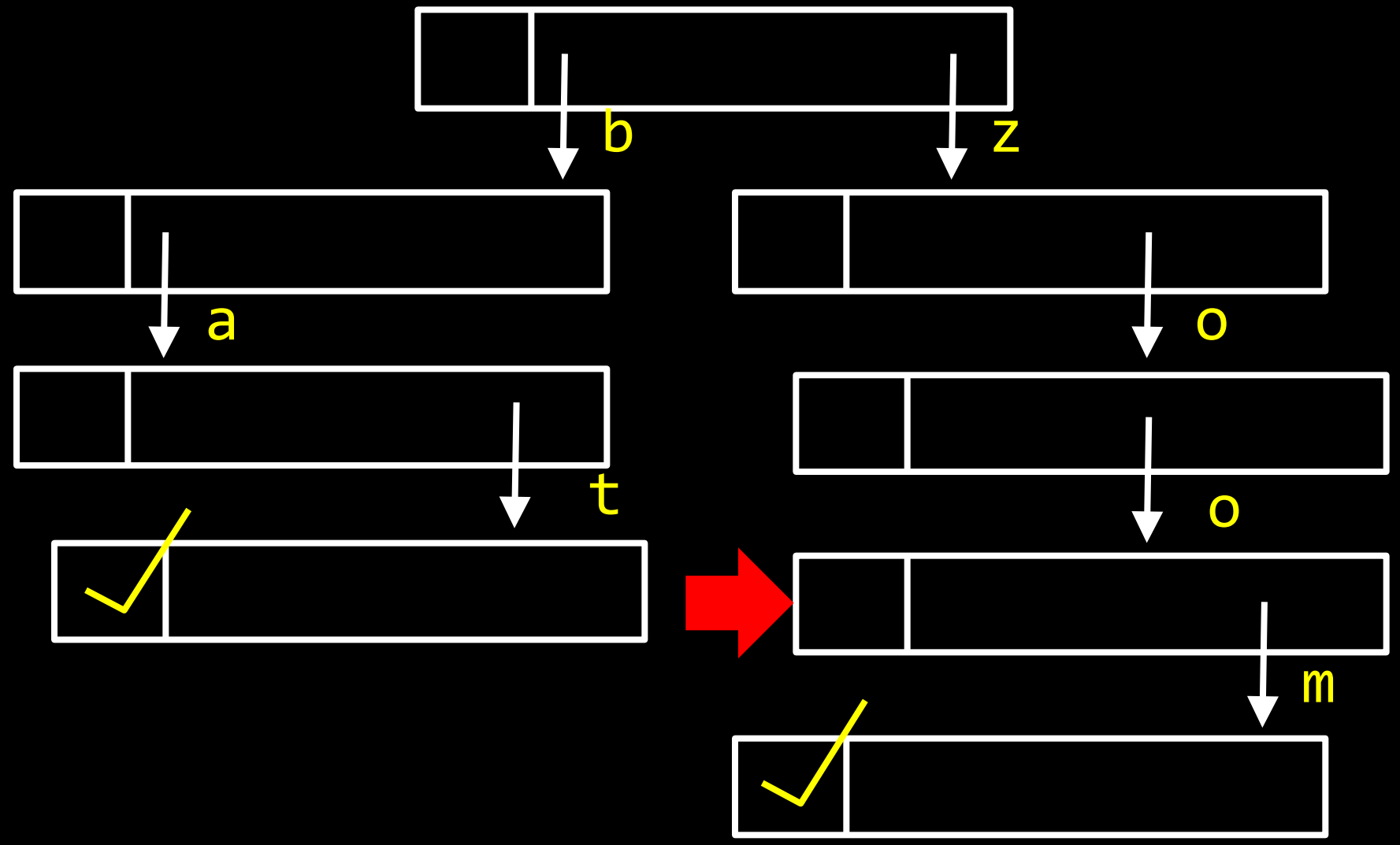

is_word
children
b
z
a
o
t
o
m

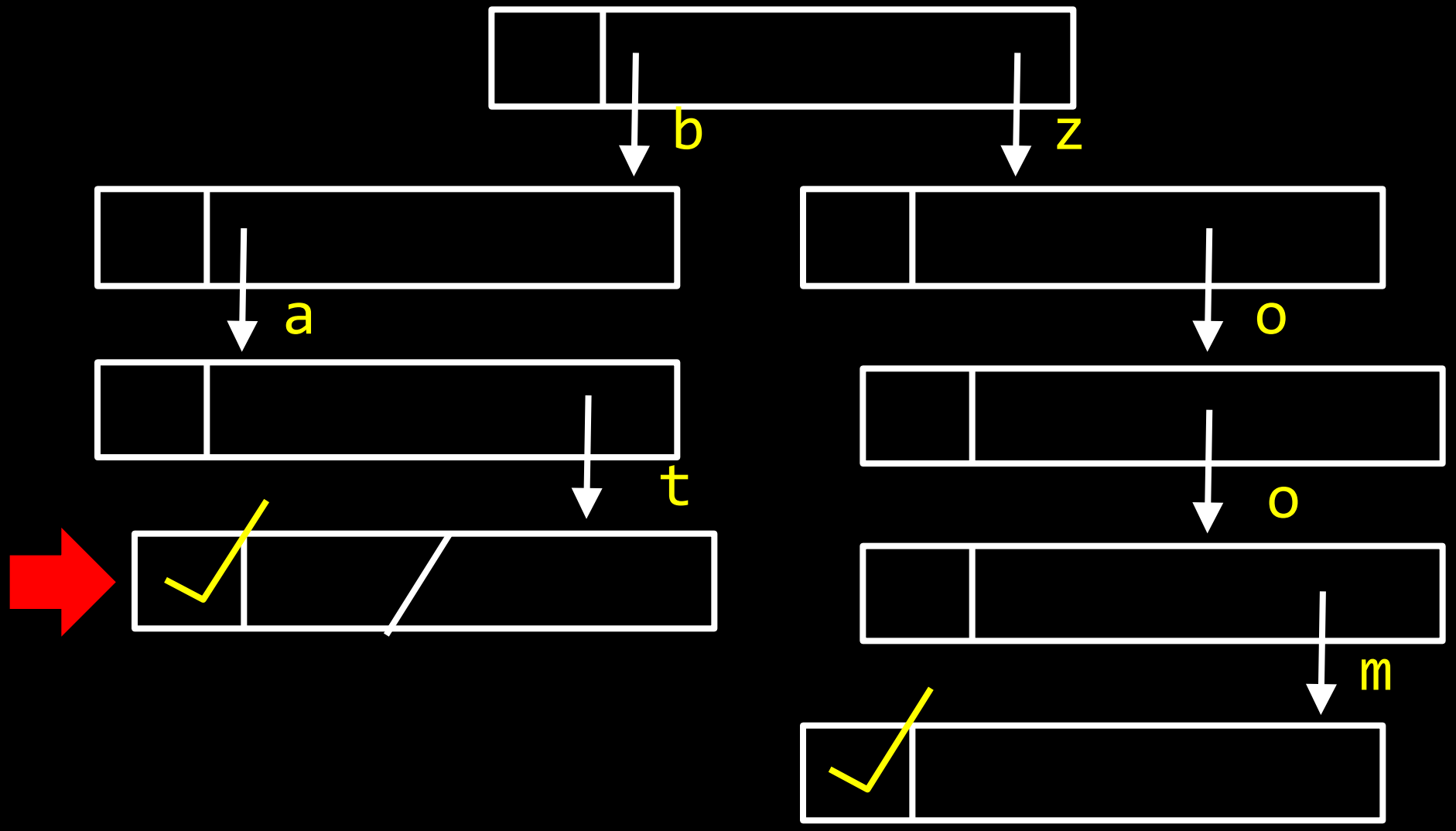

b
z
a
o
t
o
m

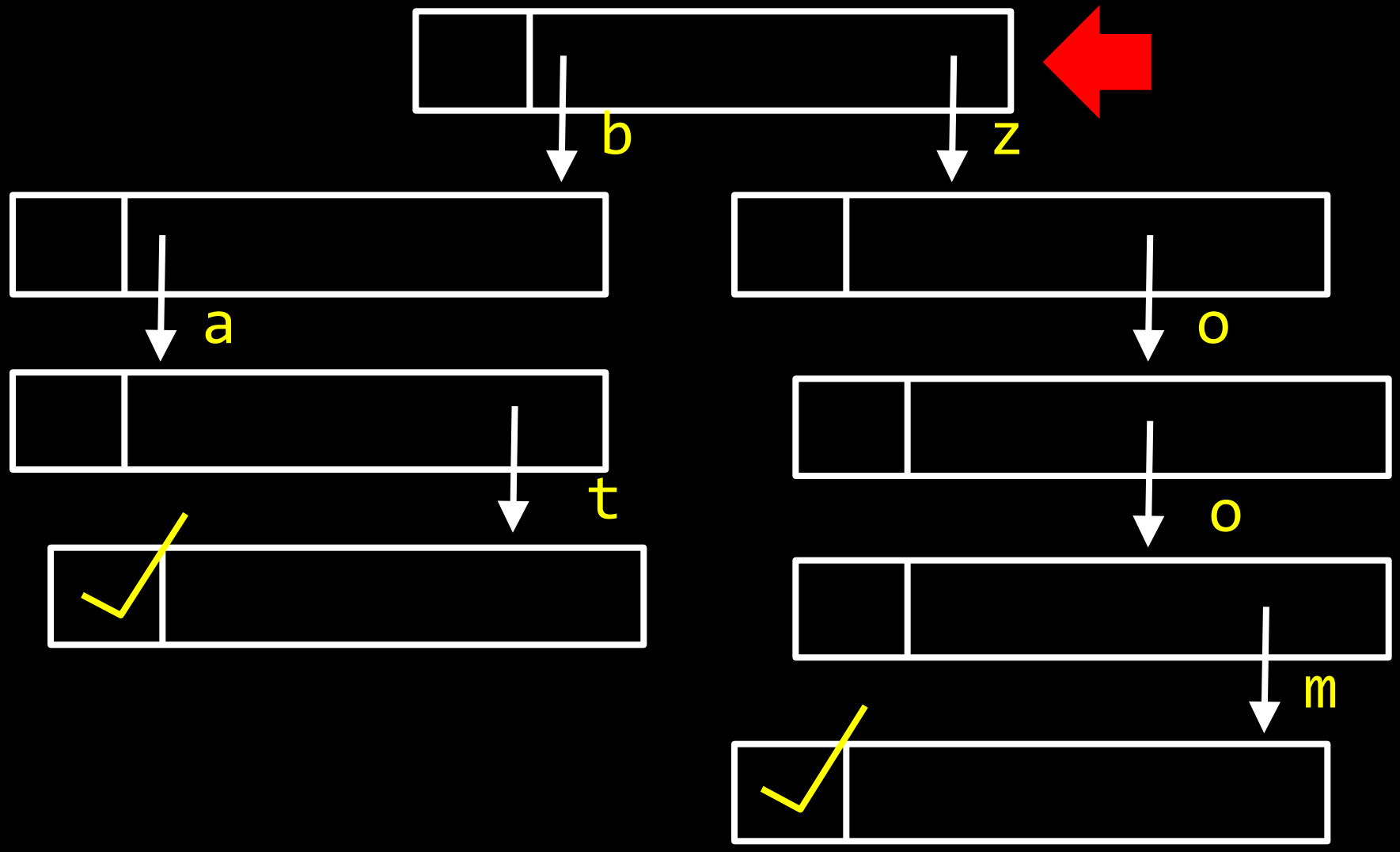

b
z
a
o
t
o
m

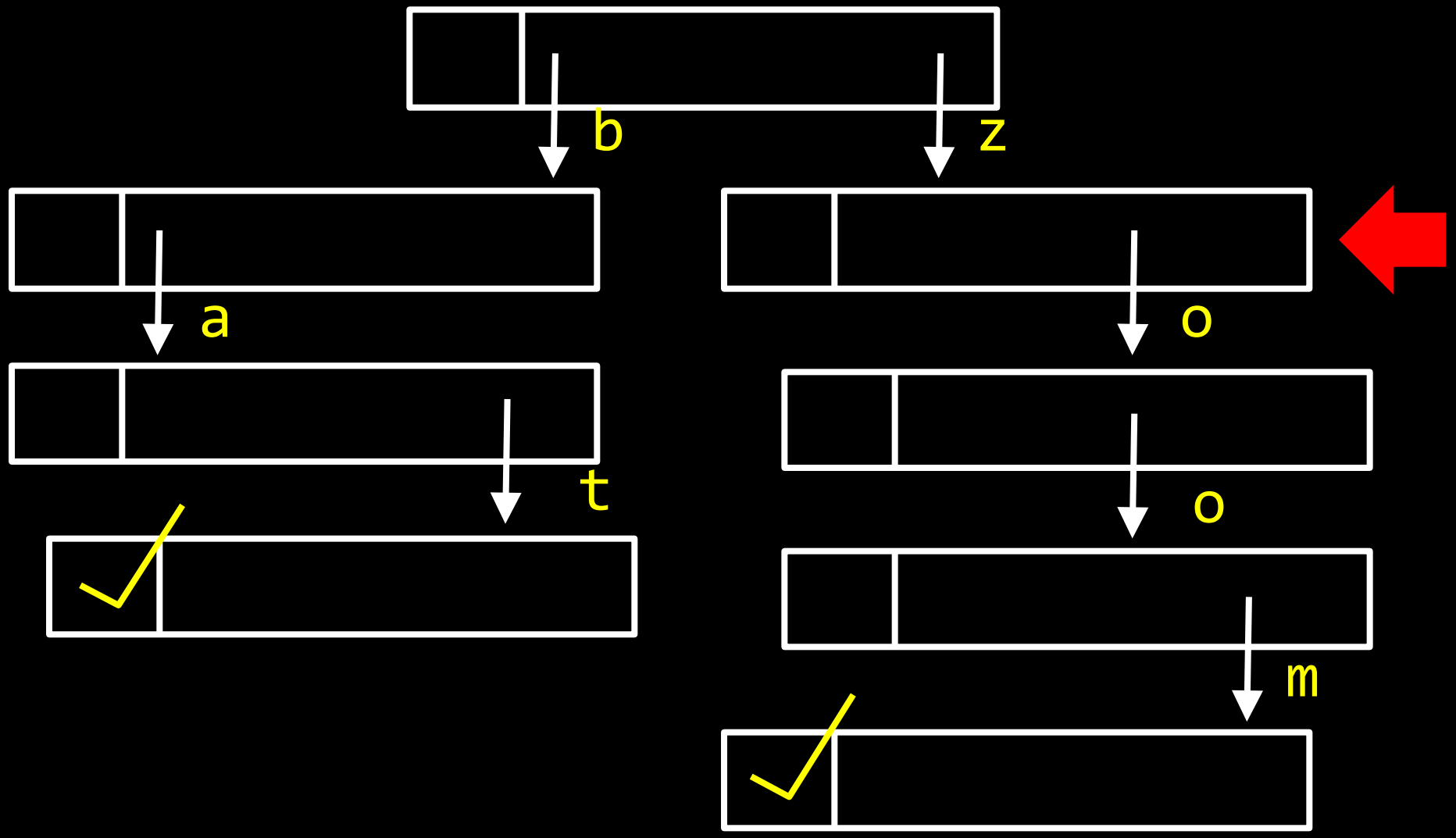

b
z
a
o
t
o
m

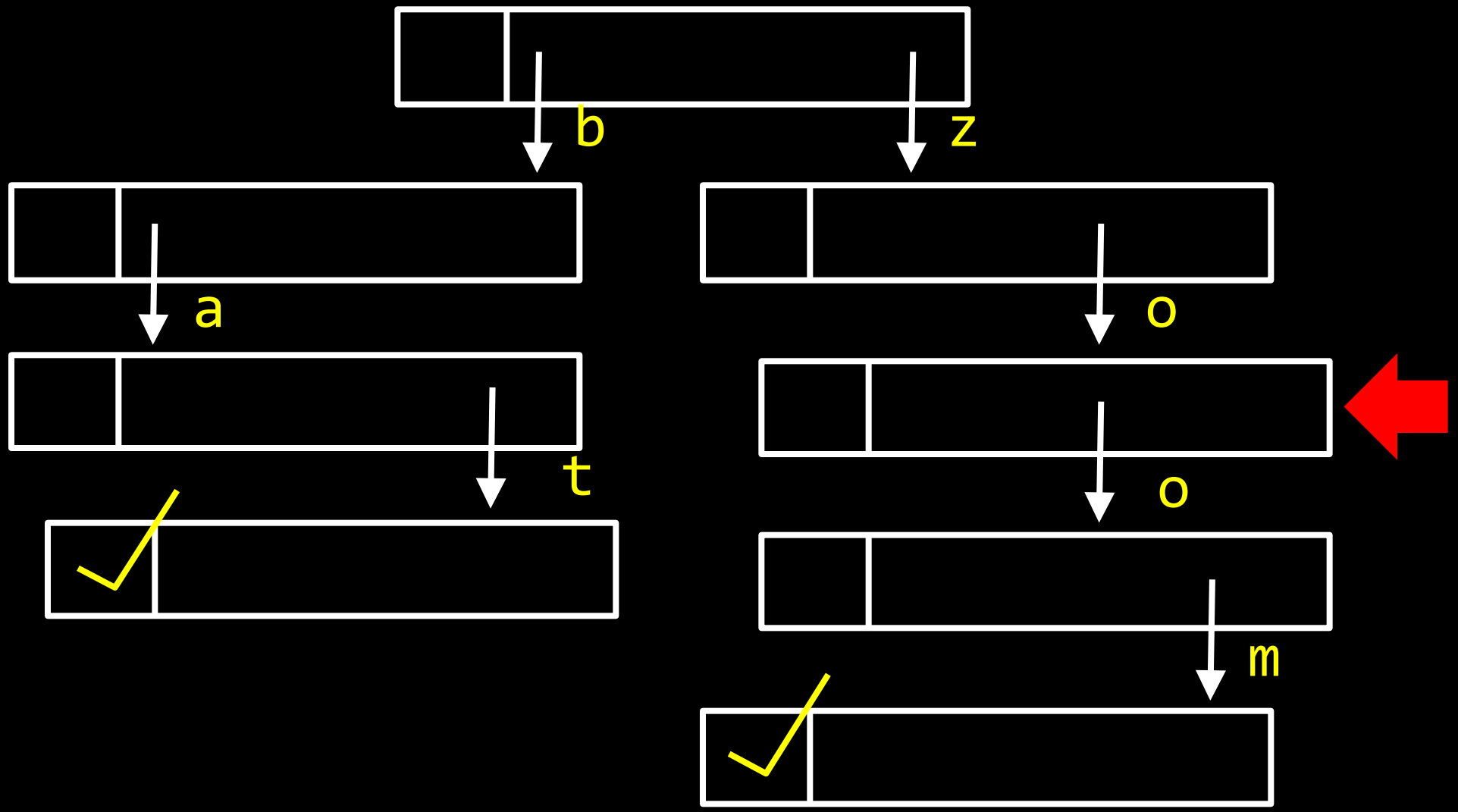

b
z
a
o
t
o
m

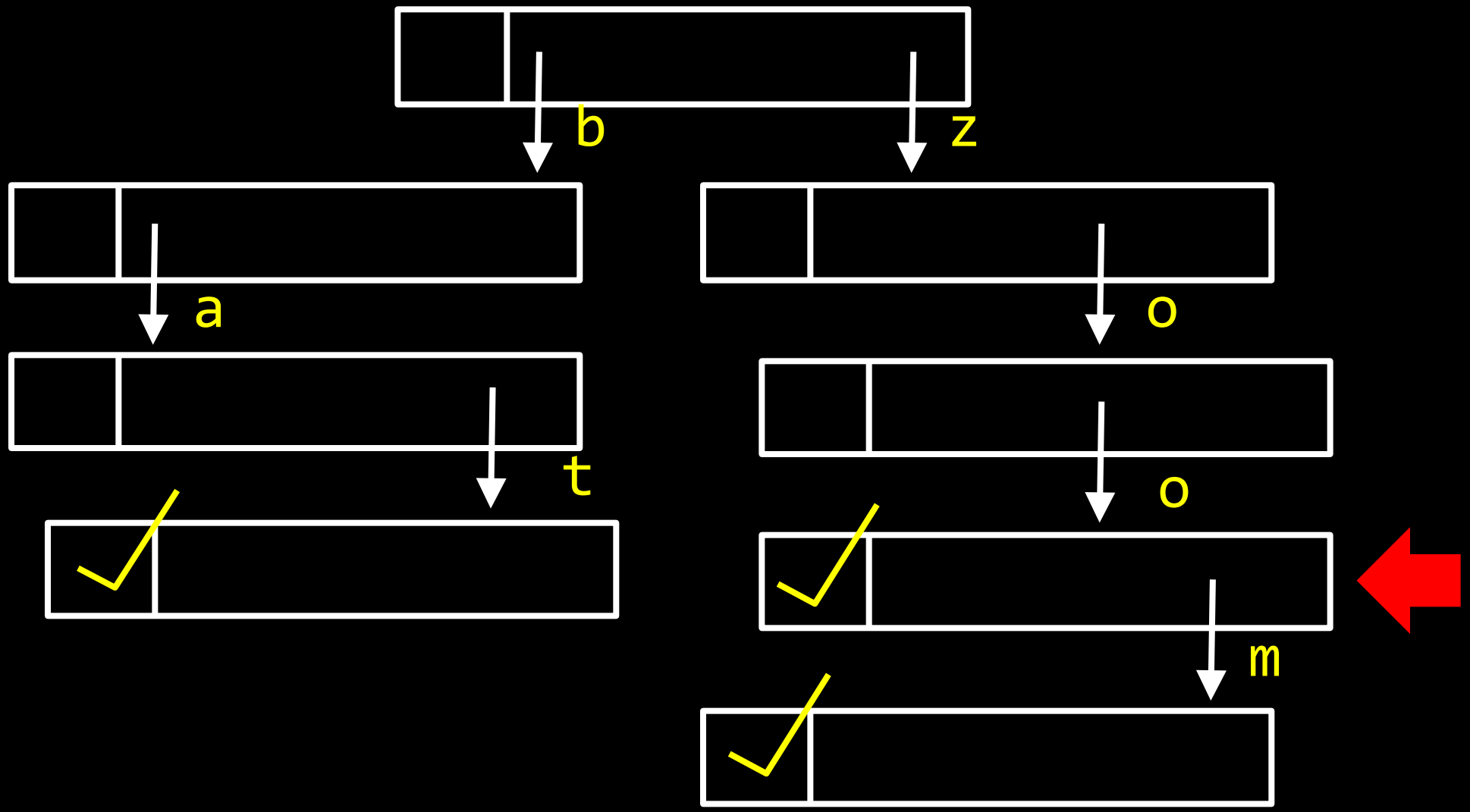

b
z
a
o
t
o
m

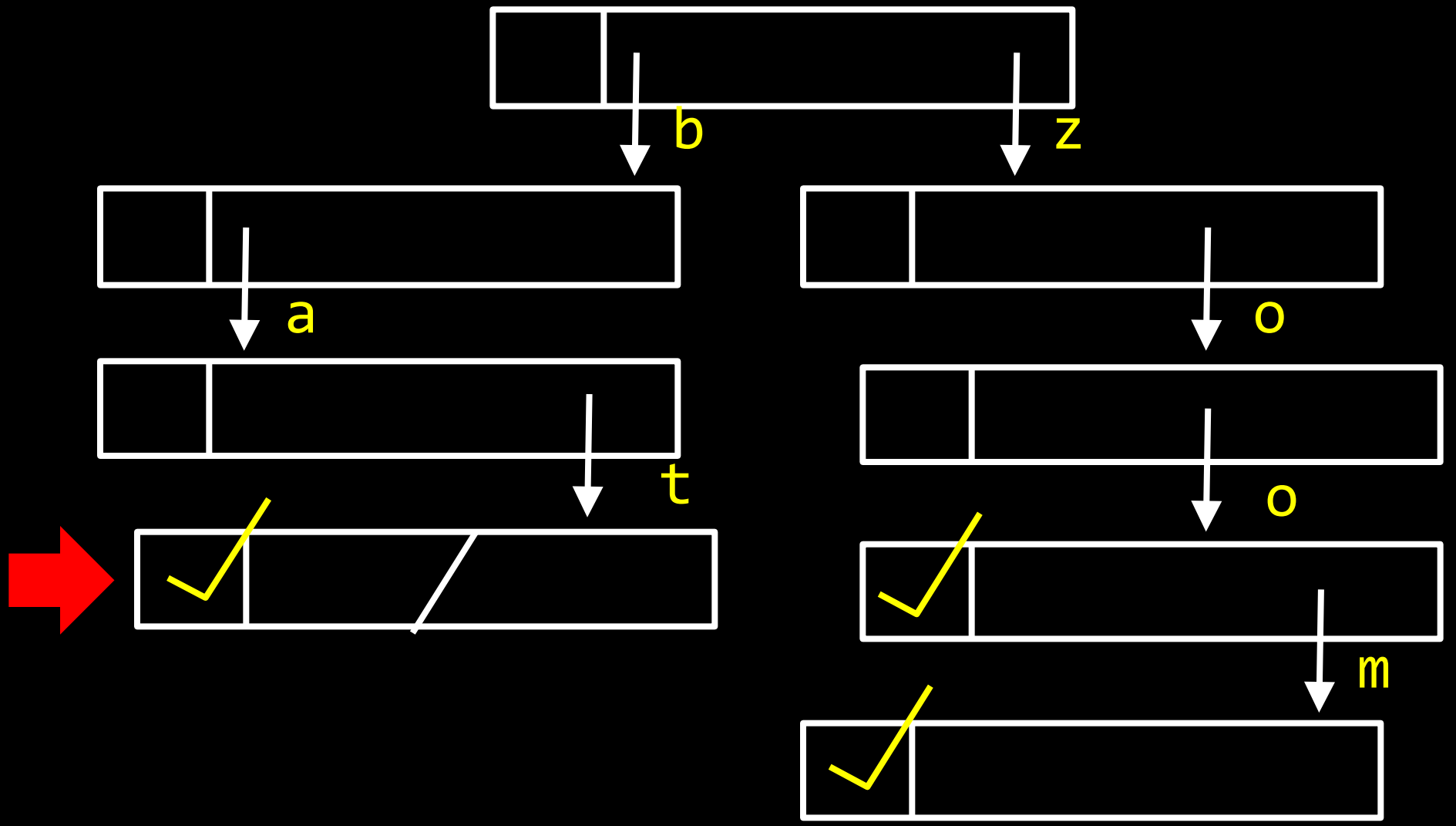

b
z
a
o
t
o
m

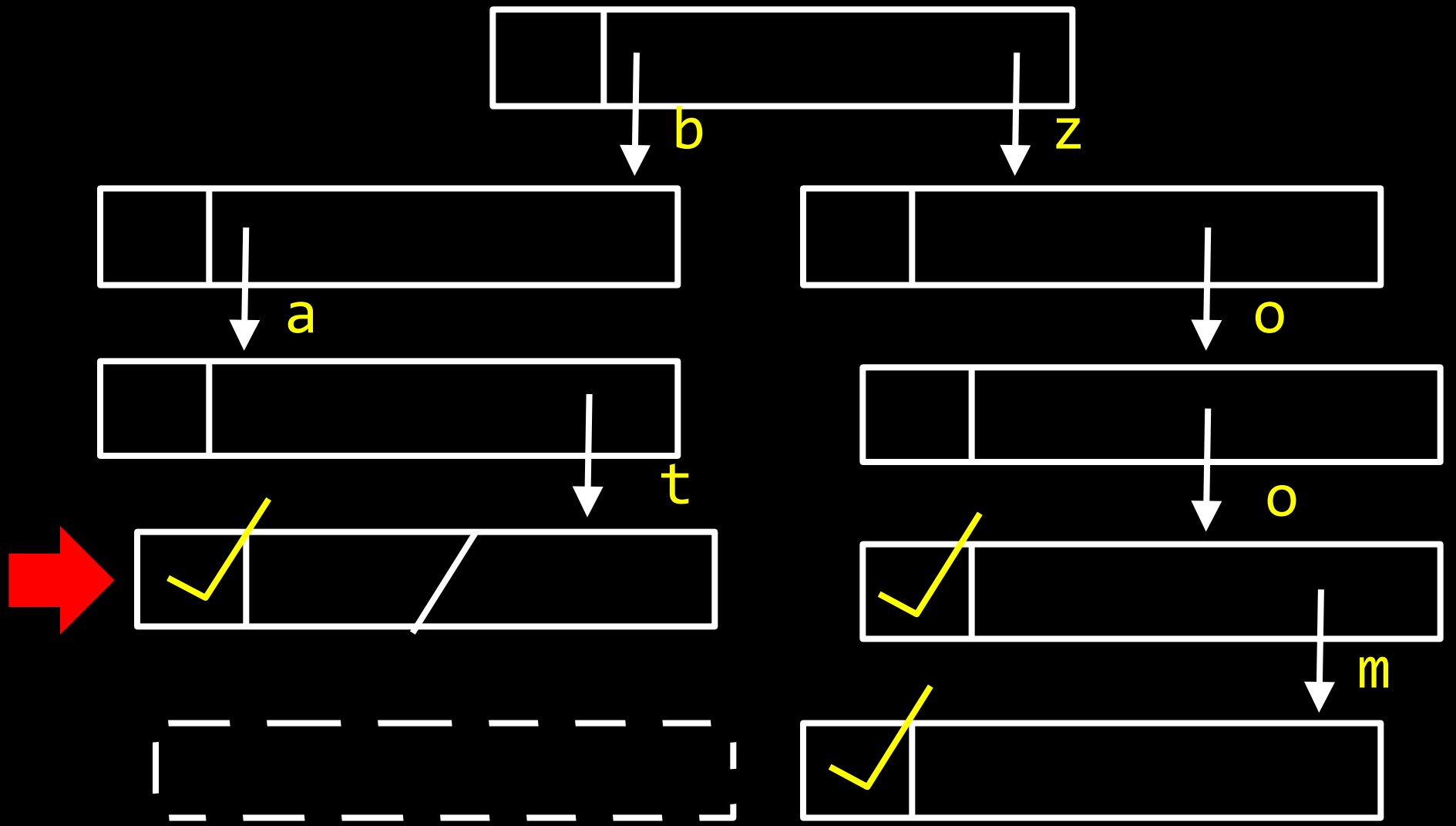

b
z
a
o
t
o
m

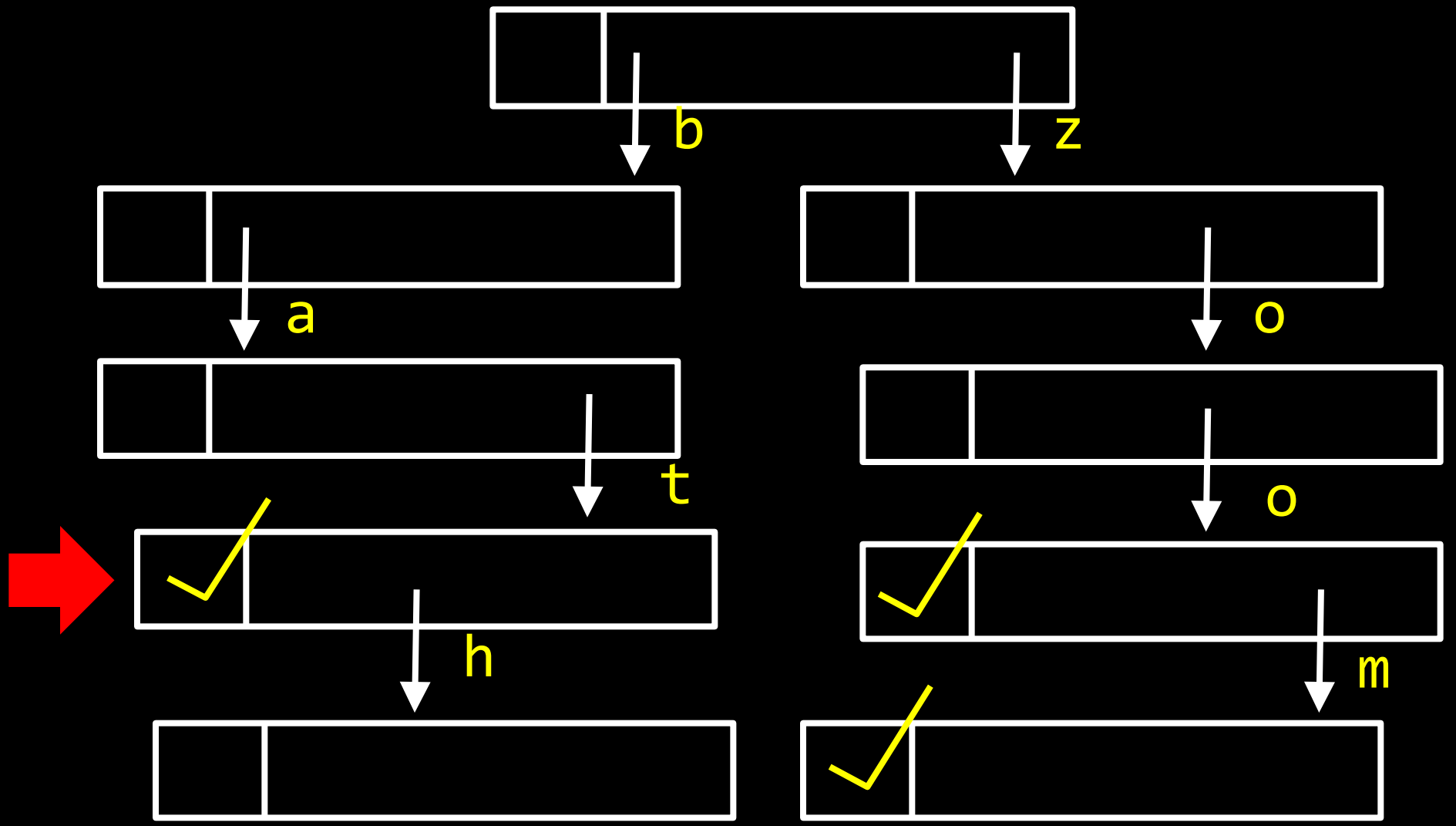

b
z
a
o
t
o
m

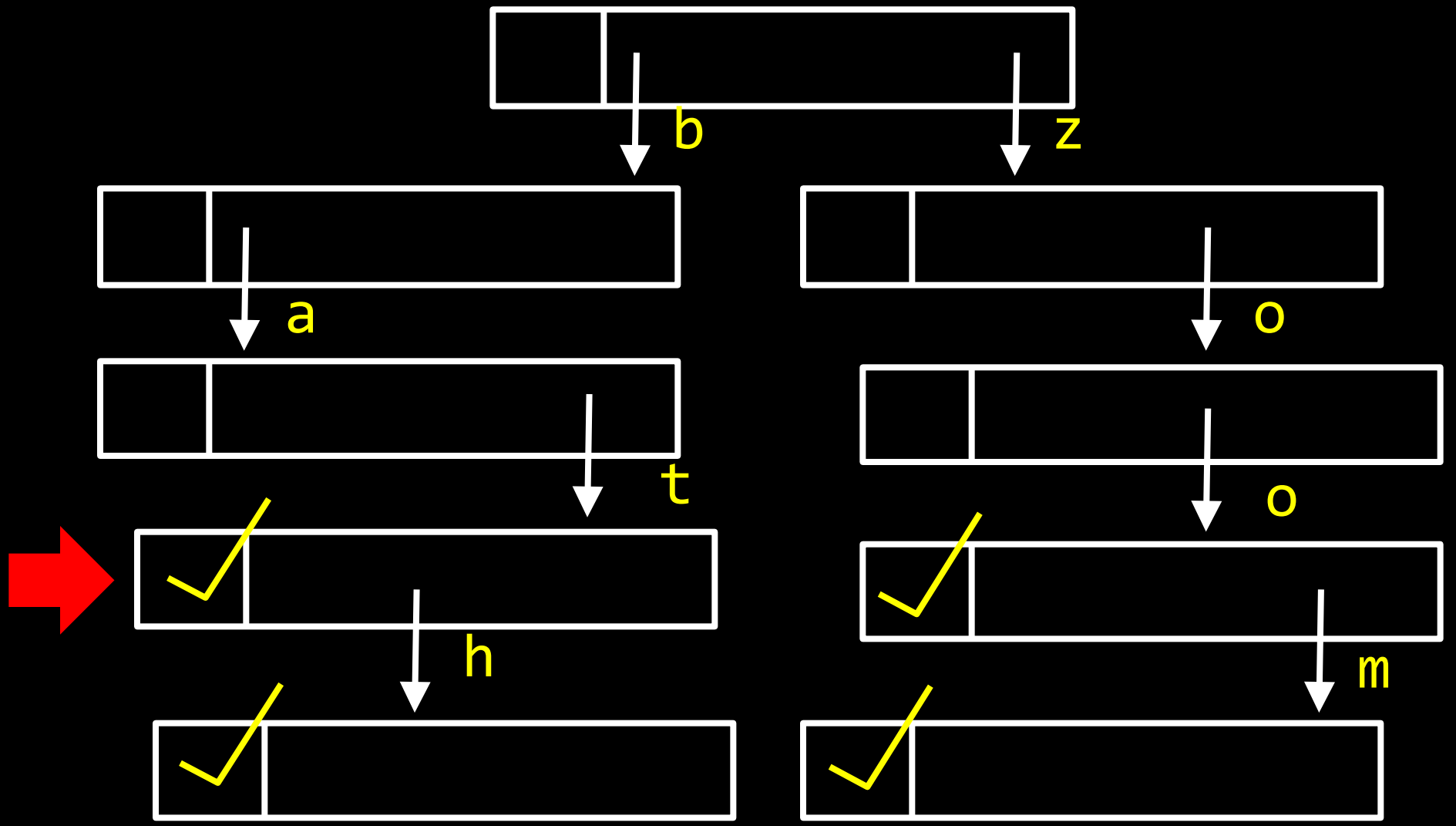

b
z
a
o
t
o
m


b
z
a
o
t
o
m


b
z
a
o
t
o
m


b
z
a
o
t
o
m


b
z
a
o
t
o
h
m


b
z
a
o
t
o
h
m

Pset5: Misspellings

Dictionaries, keys, texts

- `dictionary.c`
- `dictionary.h`
- `speller.c`

Pset5: Misspellings

`speller.c`

- Don’t mess with it!
- By way of `getrusage`, we can test the performance of your spell checker

Pset5: Misspellings

`dictionary.c`

- Implement a spell checker to load 143,091 words into the dictionary
- Organize into a hash table or trie!
- Be careful to resolve collisions
- Really keep track of all your pointers
- Test the performance of your program on the Big Board!