Stacks (Data Structure)

Stacks (Data Structure)

- A stack is a special type of structure that can be used to maintain data in an organized way.

- This data structure is commonly implemented in one of two ways: as an **array** or as a **linked list**.

- In either case, the important rule is that when data is added to the stack, it sits “on top,” and so if an element needs to be removed, the most recently added element is the only element that can legally be removed.
    - *Last in, first out (LIFO)*

Stacks (Data Structure)

- There are only two operations that may legally be performed on a stack.
  - ***Push***: Add a new element to the top of the stack.
  - ***Pop***: Remove the most recently-added element from the top of the stack.

Stacks (Data Structure)

- Array-based implementation

```
typedef struct _stack
{
    VALUE array[CAPACITY];
    int top;
}
stack;
```

Stacks (Data Structure)

- Array-based implementation

```
typedef struct _stack
{
    VALUE array[CAPACITY];
    int top;
}
stack;
```

Stacks (Data Structure)

- Array-based implementation

```
typedef struct _stack
{
    VALUE array[CAPACITY];
    int top;
}
stack;
```

Stacks (Data Structure)

- Array-based implementation

```
typedef struct _stack
{
    VALUE array[CAPACITY];
    int top;
}
stack;
```

Stacks (Data Structure)

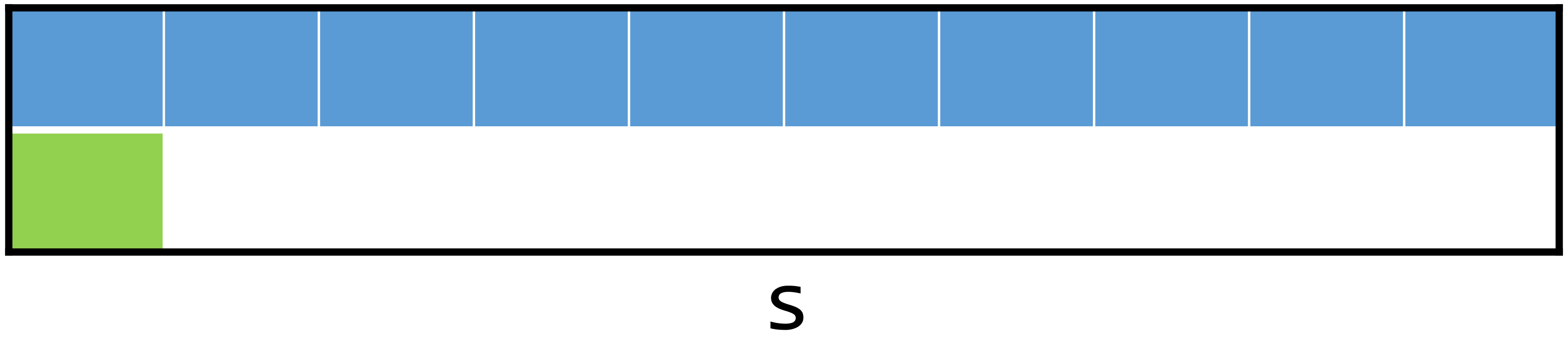
- Array-based implementation

```
stack s;
```

Stacks (Data Structure)

- Array-based implementation

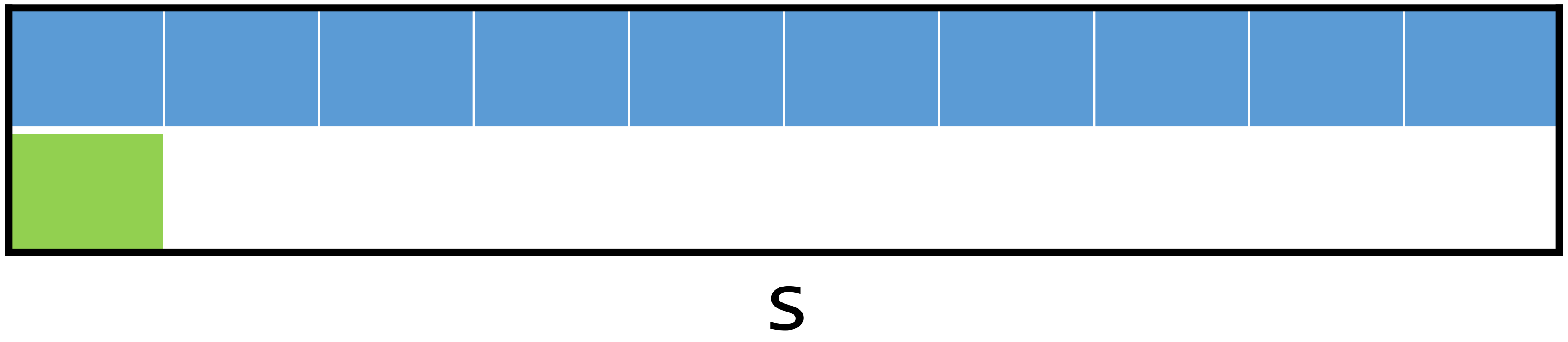
```
stack s;
```



Stacks (Data Structure)

- Array-based implementation

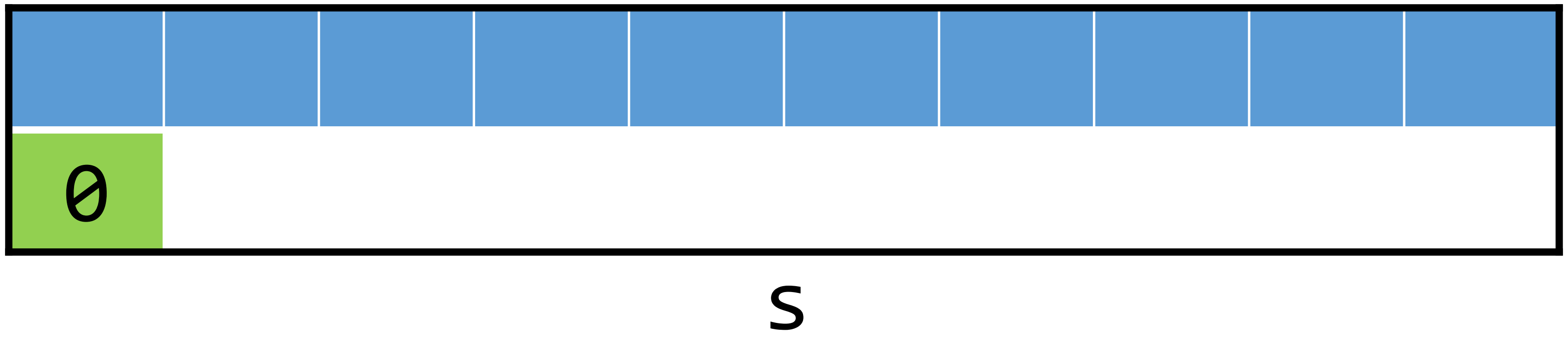
```
stack s;
s.top = 0;
```



Stacks (Data Structure)

- Array-based implementation

```
stack s;
s.top = 0;
```



Stacks (Data Structure)

- Array-based implementation
    - ***Push***: Add a new element to the top of the stack.

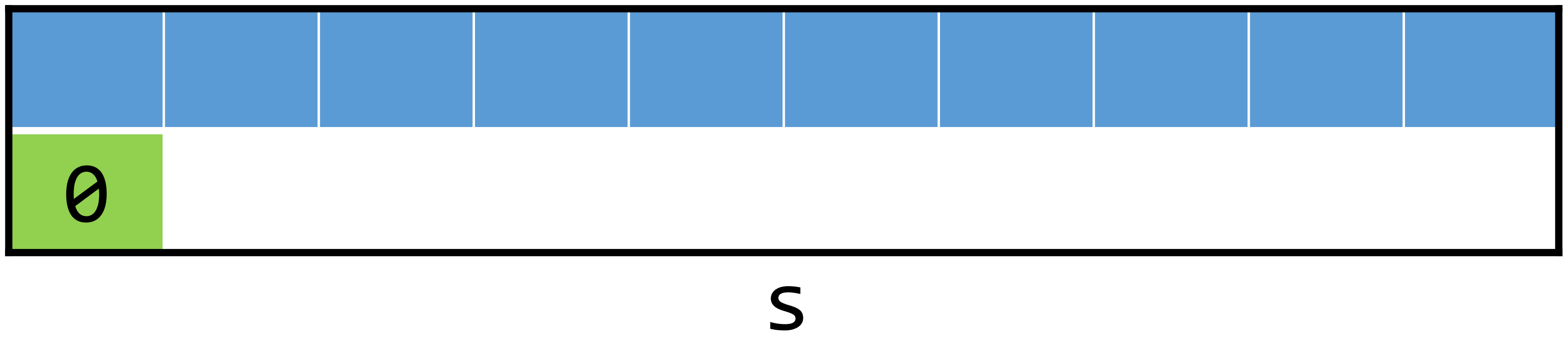
In the general case, `push()` needs to:

- Accept a pointer to the stack.
- Accept data of type VALUE to be added to the stack.
- Add that data to the stack at the top of the stack.
- Change the location of the top of the stack.

Stacks (Data Structure)

- Array-based implementation

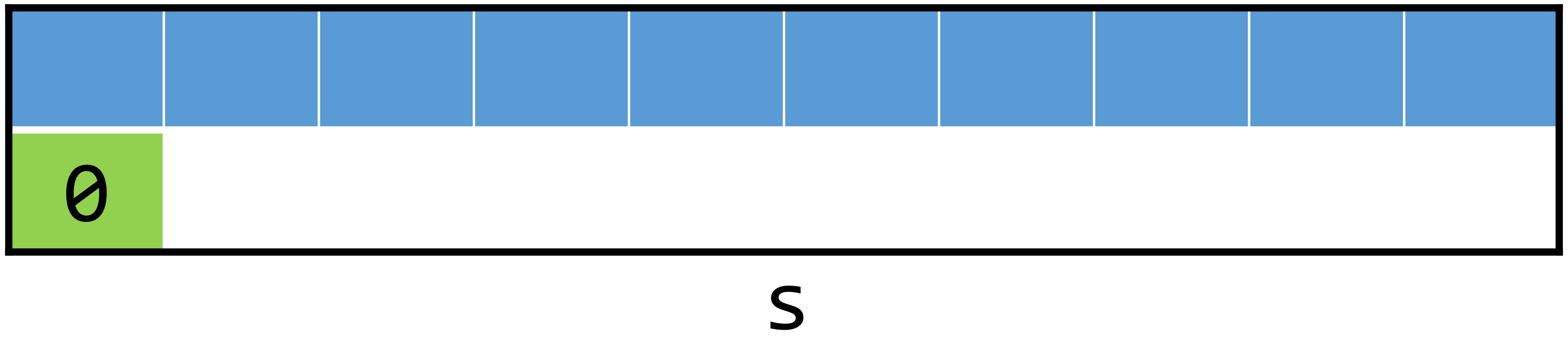
```
void push(stack* s, VALUE data);
```



Stacks (Data Structure)

- Array-based implementation

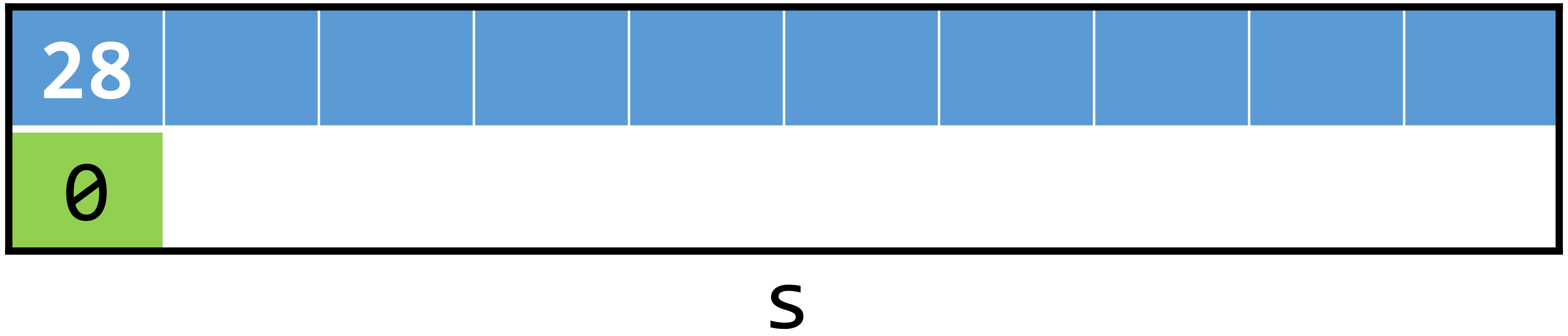
```
stack s;
s.top = 0;
push(&s, 28);
```



Stacks (Data Structure)

- Array-based implementation

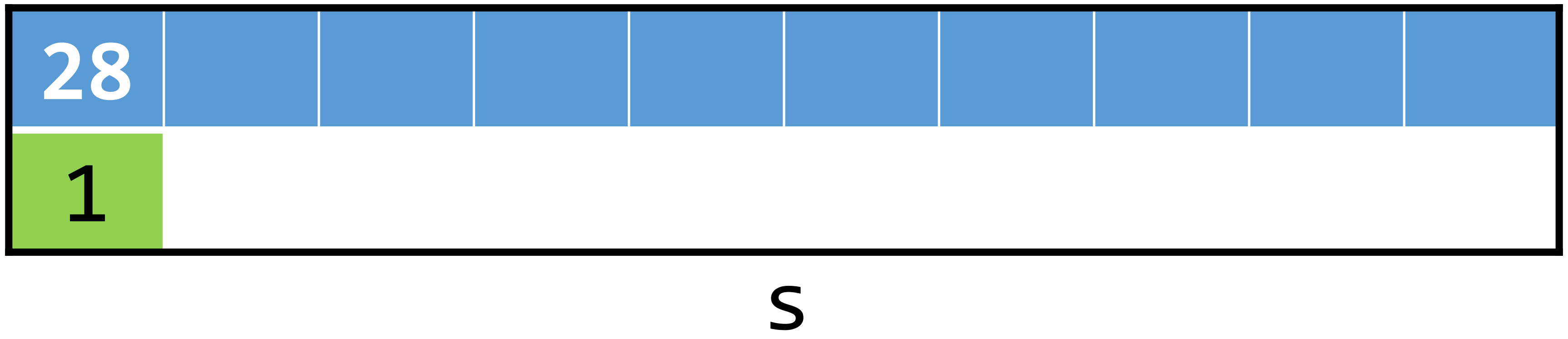
```
stack s;
s.top = 0;
push(&s, 28);
```



Stacks (Data Structure)

- Array-based implementation

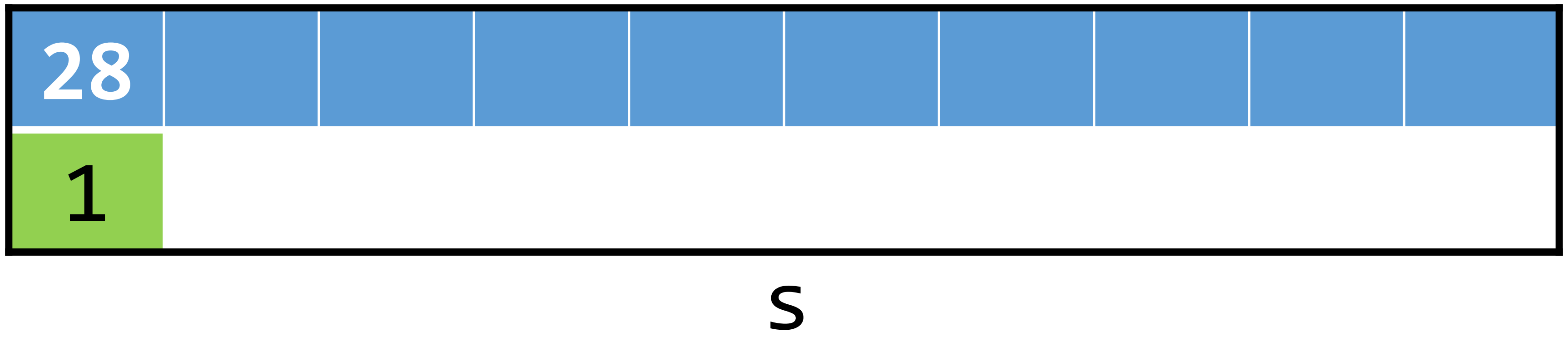
```
stack s;
s.top = 0;
push(&s, 28);
```



Stacks (Data Structure)

- Array-based implementation

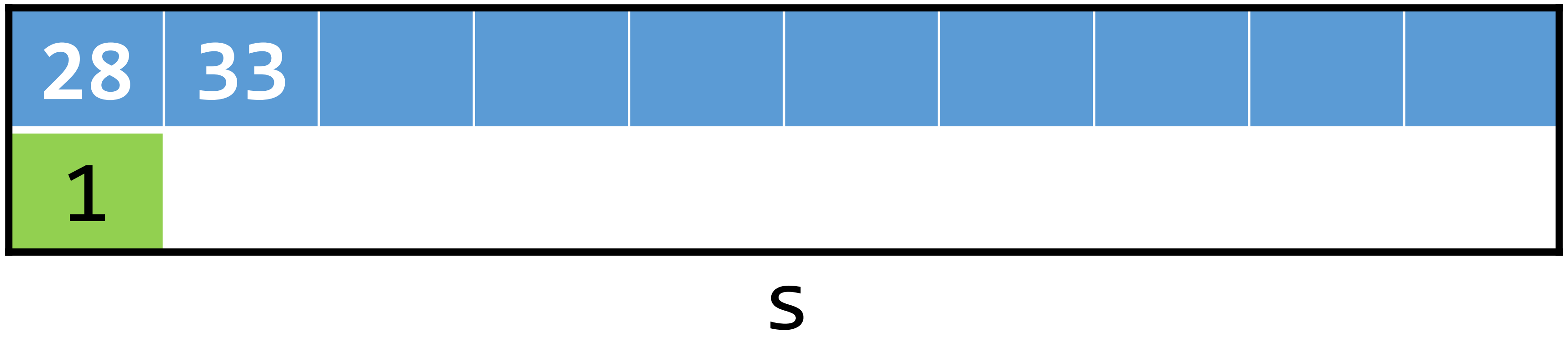
```
push(&s, 33);
```



Stacks (Data Structure)

- Array-based implementation

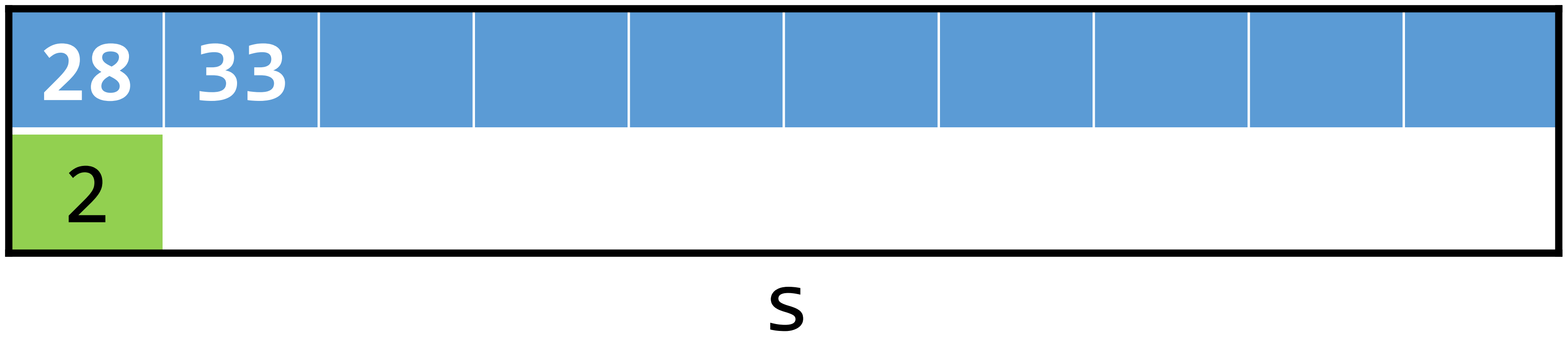
```
push(&s, 33);
```



Stacks (Data Structure)

- Array-based implementation

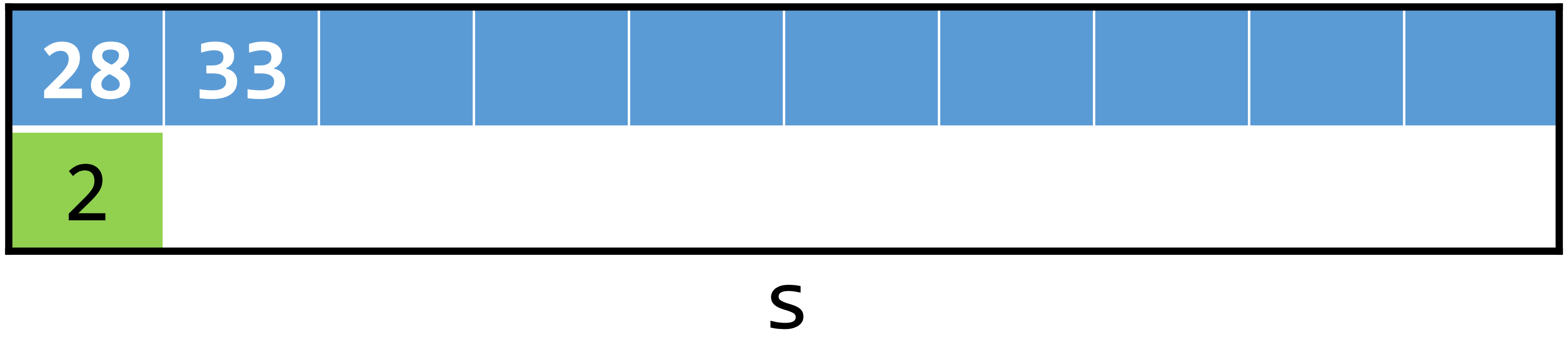
```
push(&s, 33);
```



Stacks (Data Structure)

- Array-based implementation

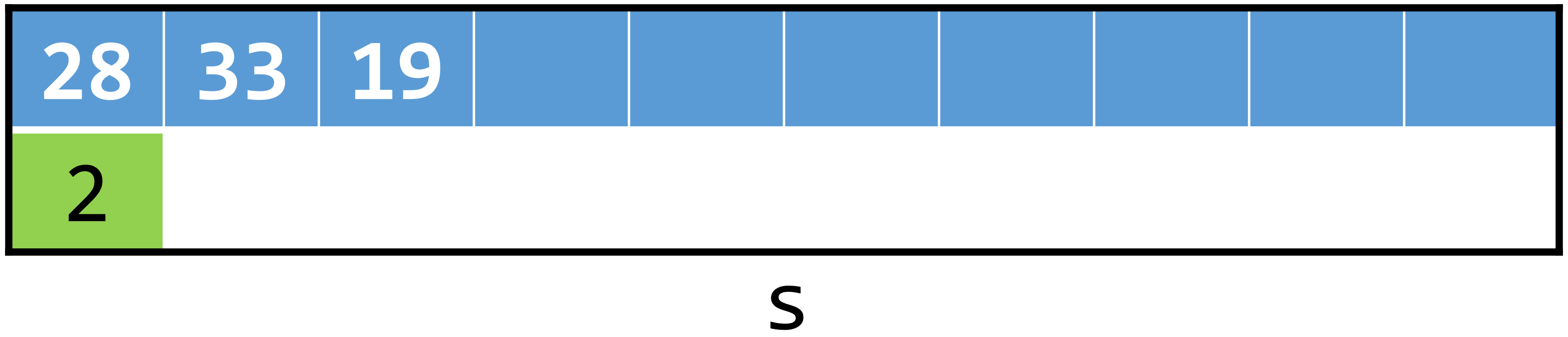
```
push(&s, 19);
```



Stacks (Data Structure)

- Array-based implementation

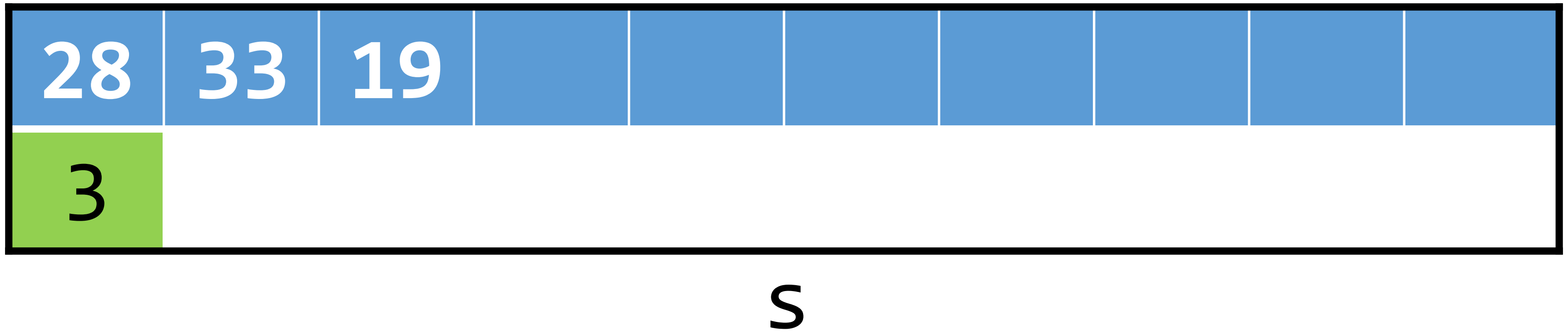
```
push(&s, 19);
```



Stacks (Data Structure)

- Array-based implementation

```
push(&s, 19);
```



Stacks (Data Structure)

- Array-based implementation
  - ***Pop***: Remove the most recent element from the top of the stack.

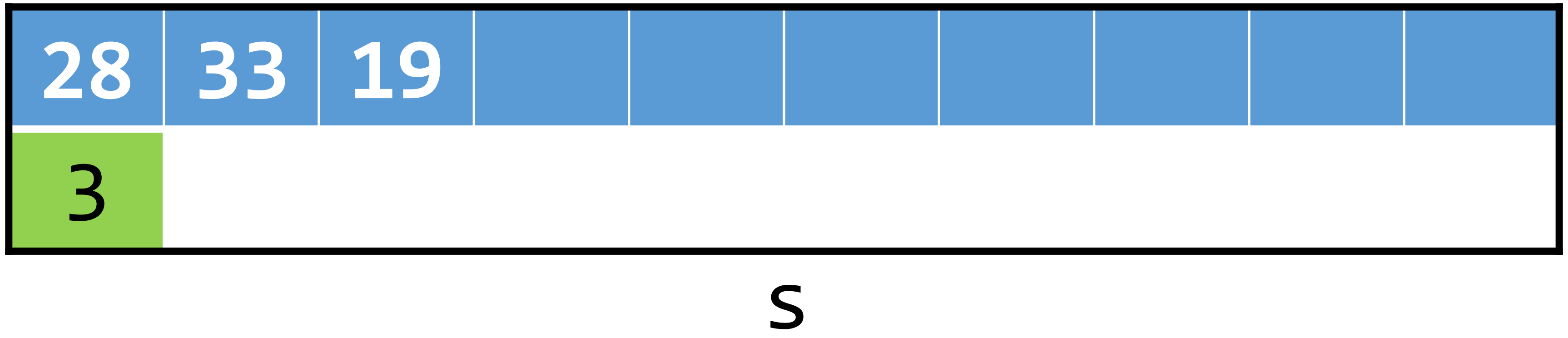
In the general case, `pop()` needs to:

- Accept a pointer to the stack.
- Change the location of the top of the stack.
- Return the value that was removed from the stack.

Stacks (Data Structure)

- Array-based implementation

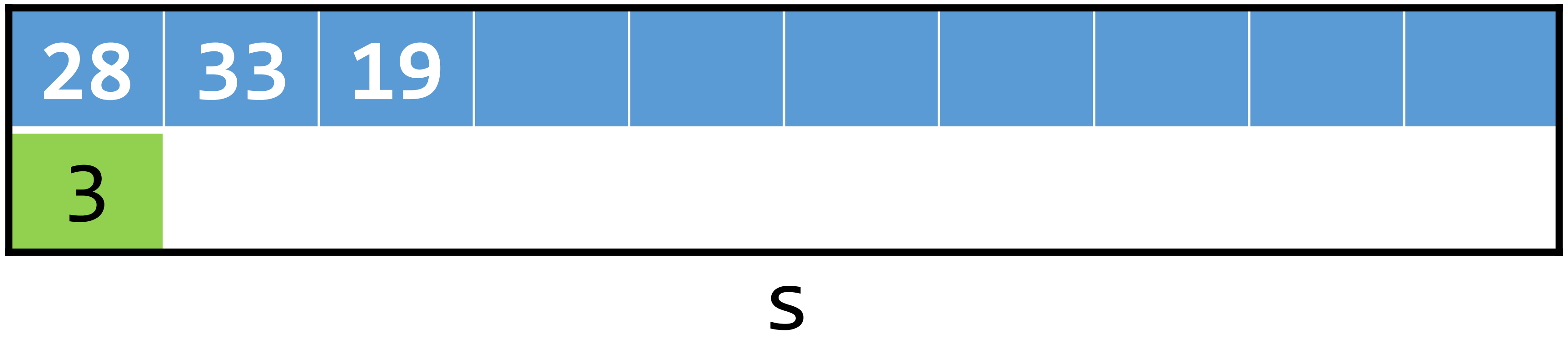
```
VALUE pop(stack* s);
```



Stacks (Data Structure)

- Array-based implementation

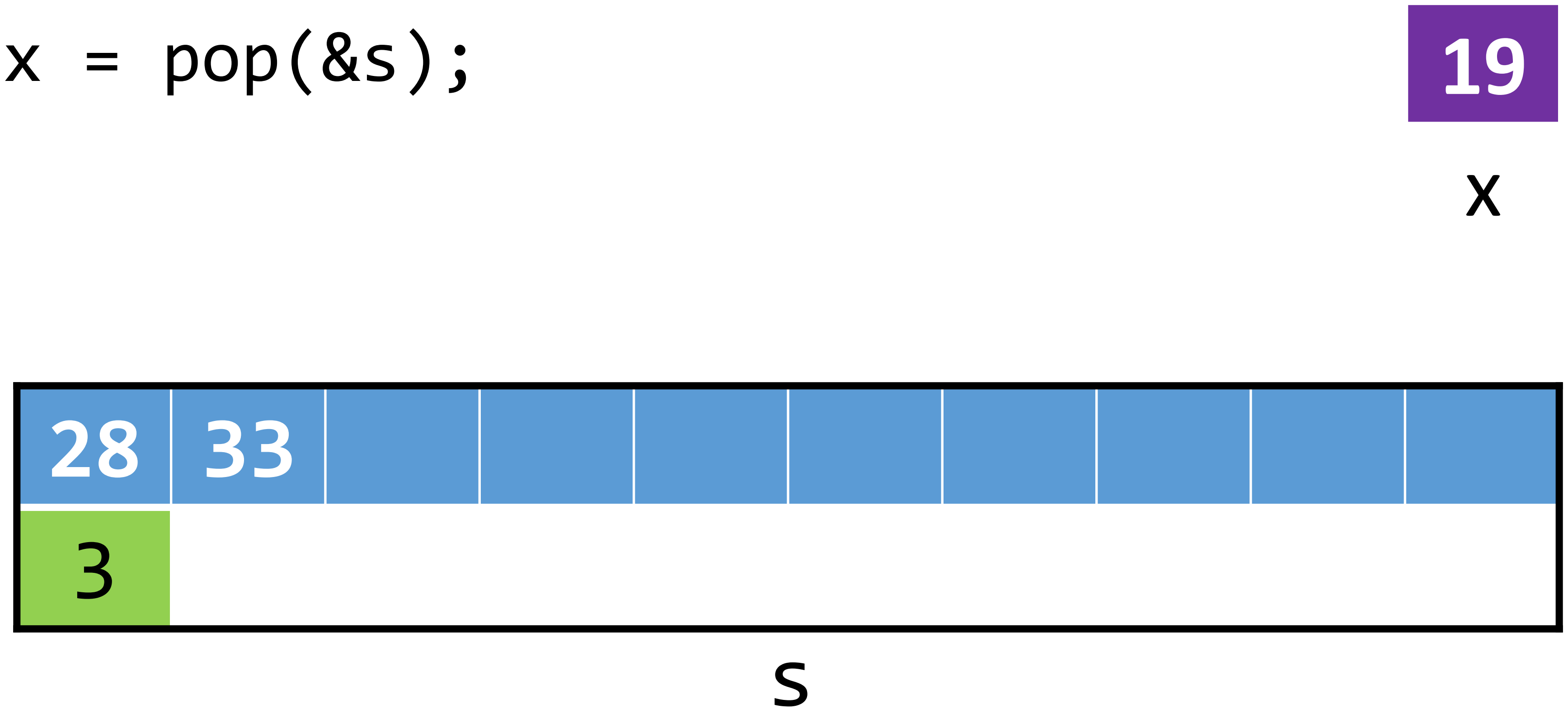
```
int x = pop(&s);
```



Stacks (Data Structure)

- Array-based implementation

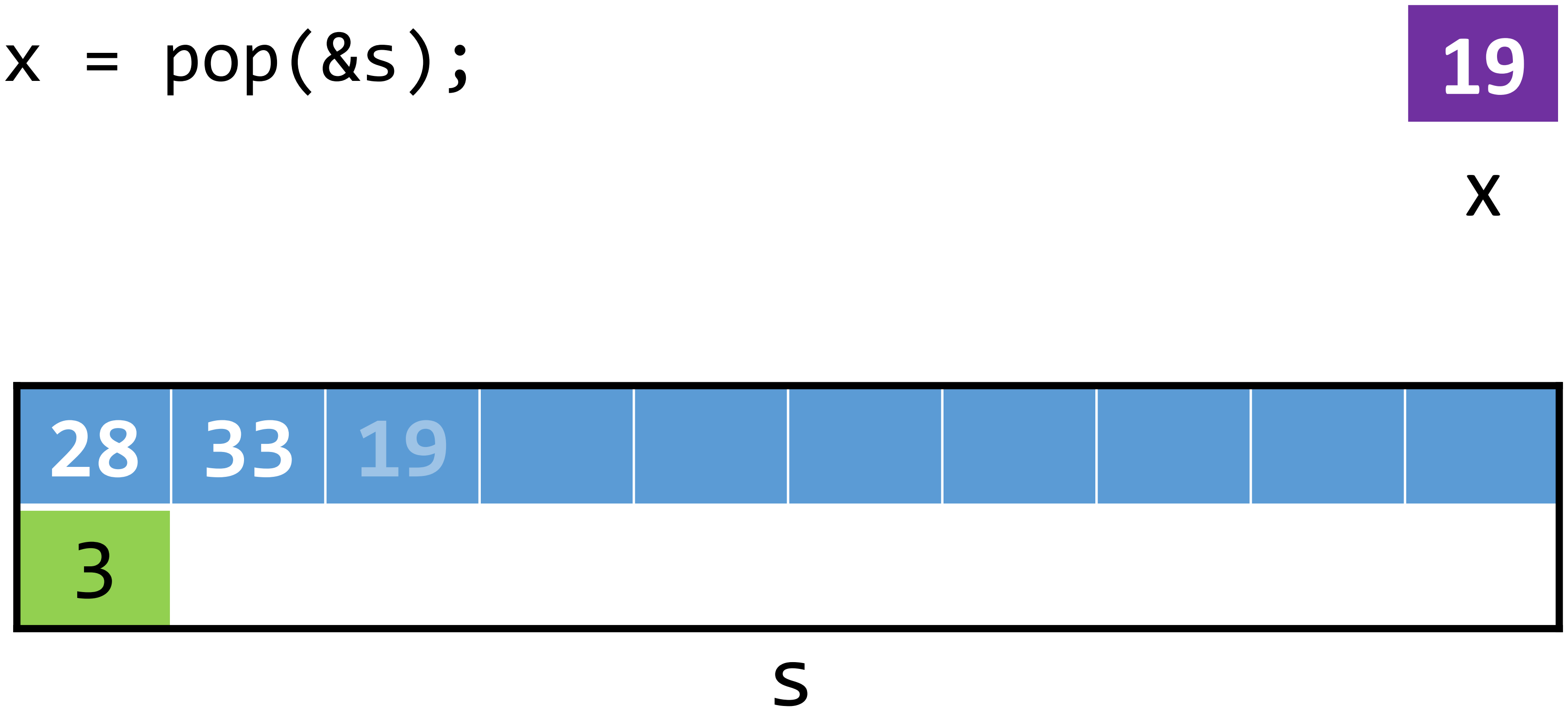
```
int x = pop(&s);
```



Stacks (Data Structure)

- Array-based implementation

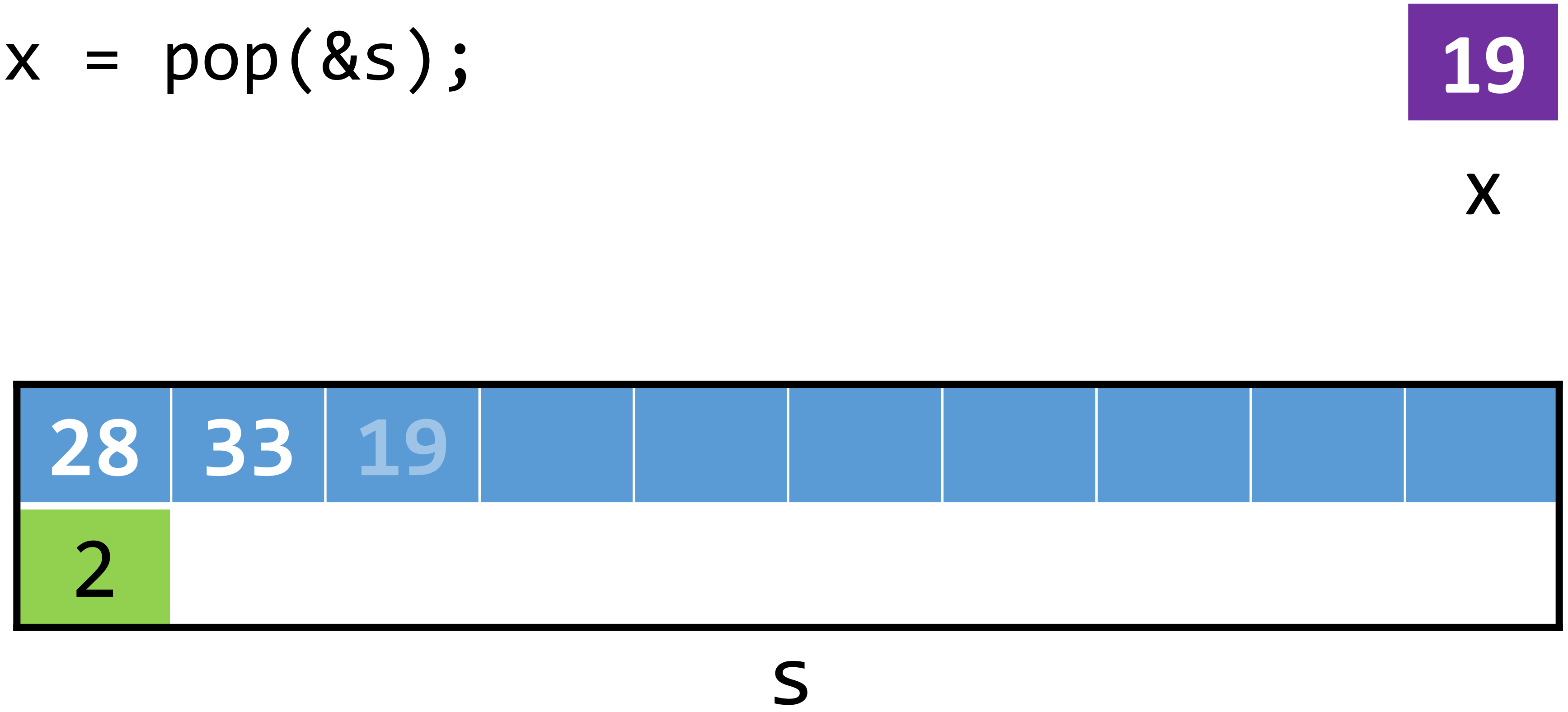
```
int x = pop(&s);
```



Stacks (Data Structure)

- Array-based implementation

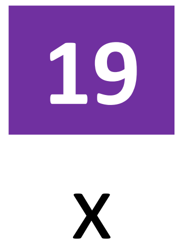
```
int x = pop(&s);
```

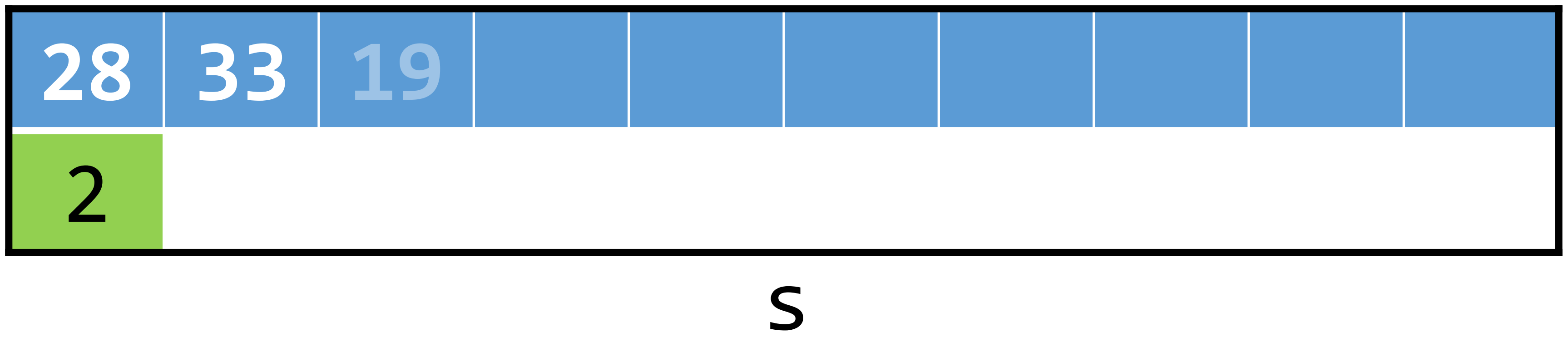


Stacks (Data Structure)

- Array-based implementation

```
int x = pop(&s);
```

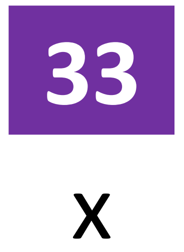


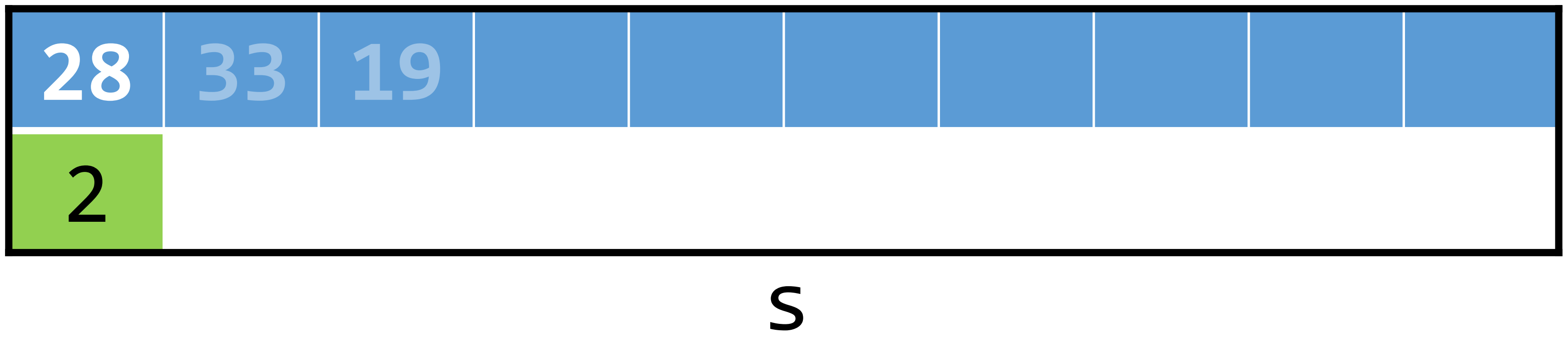


Stacks (Data Structure)

- Array-based implementation

```
int x = pop(&s);
```

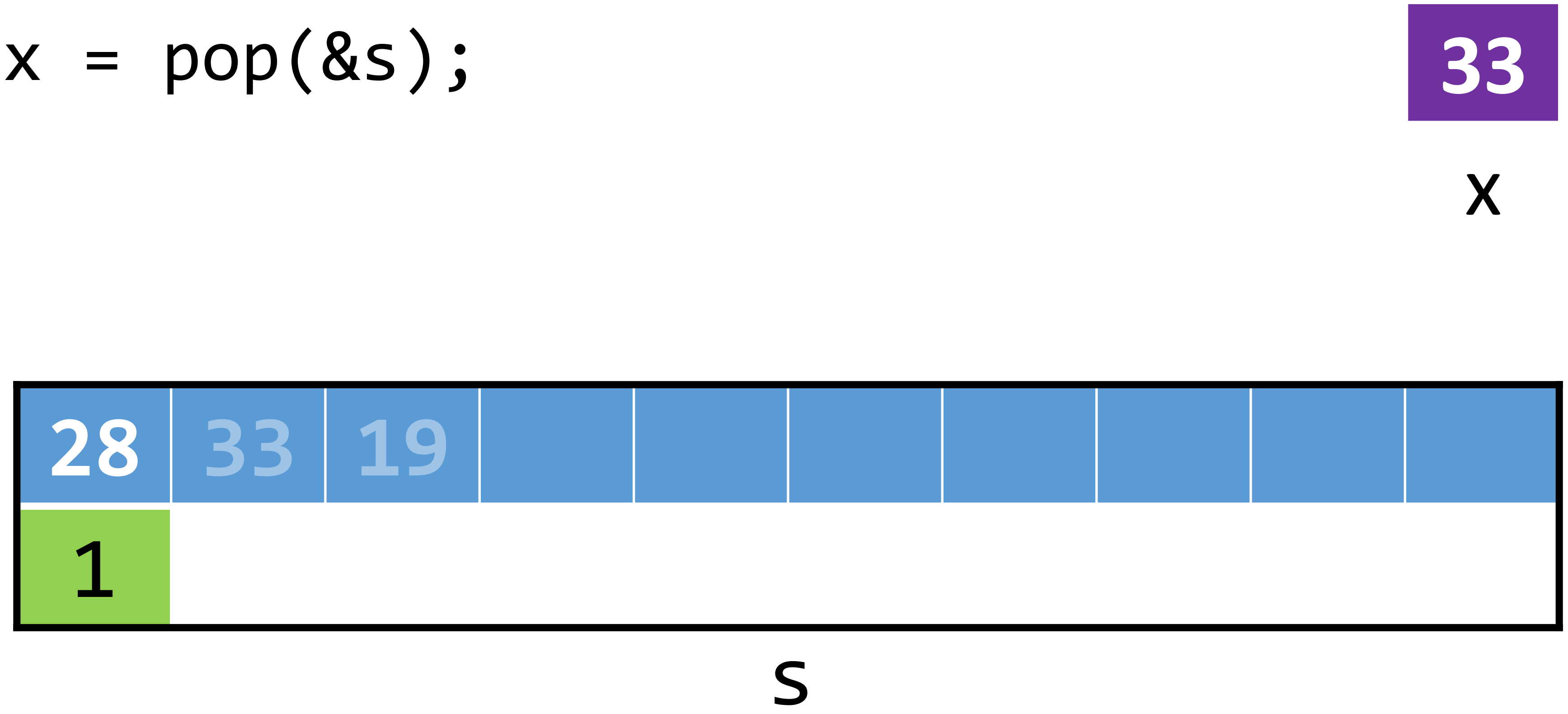




Stacks (Data Structure)

- Array-based implementation

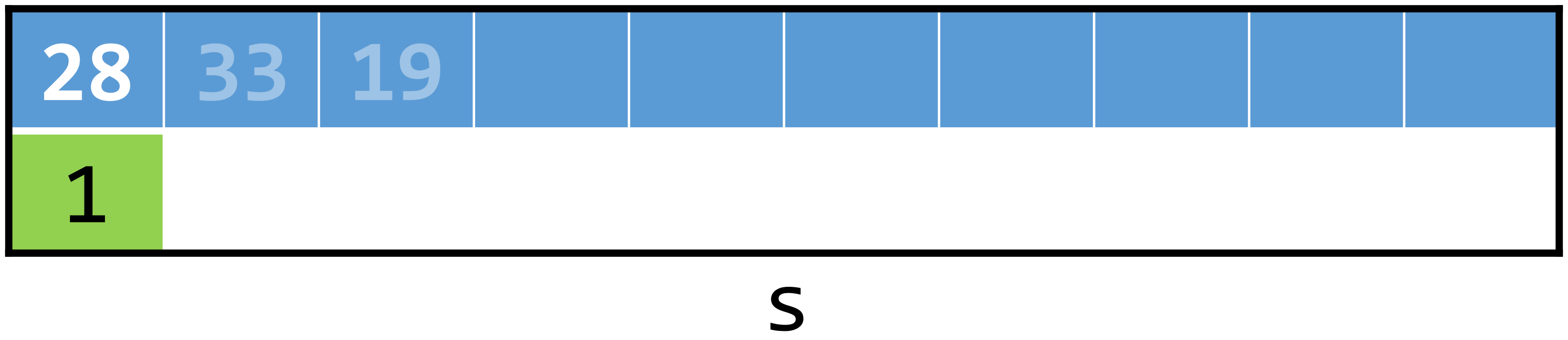
```
int x = pop(&s);
```



Stacks (Data Structure)

- Array-based implementation

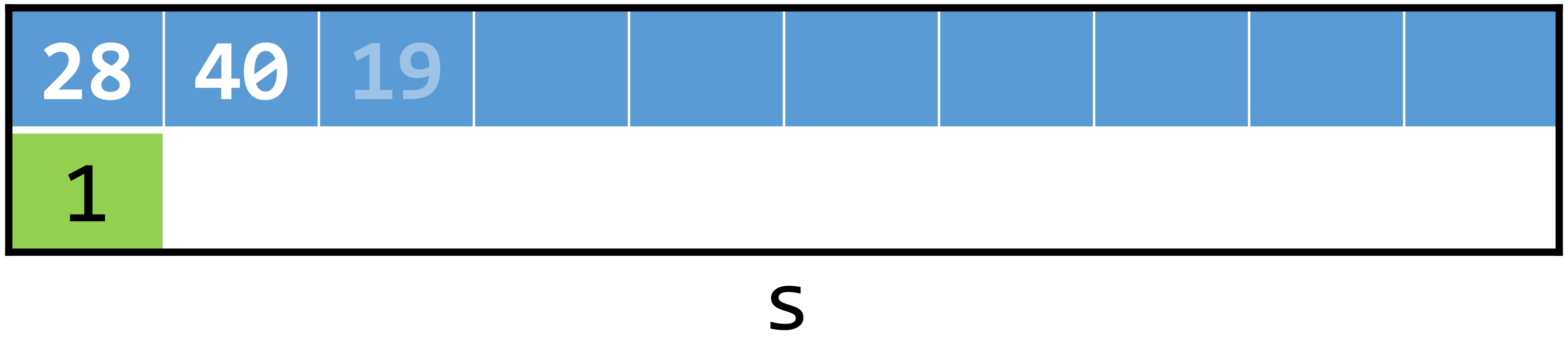
```
push(&s, 40);
```



Stacks (Data Structure)

- Array-based implementation

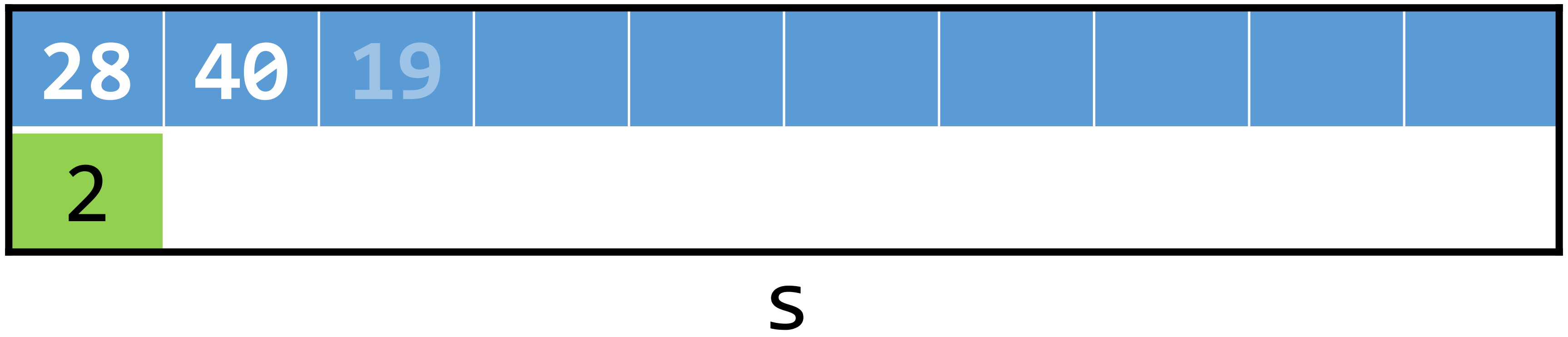
```
push(&s, 40);
```



Stacks (Data Structure)

- Array-based implementation

```
push(&s, 40);
```



Stacks (Data Structure)

- Linked list-based implementation

```
typedef struct _stack
{
    VALUE val;
    struct _stack *next;
}
stack;
```

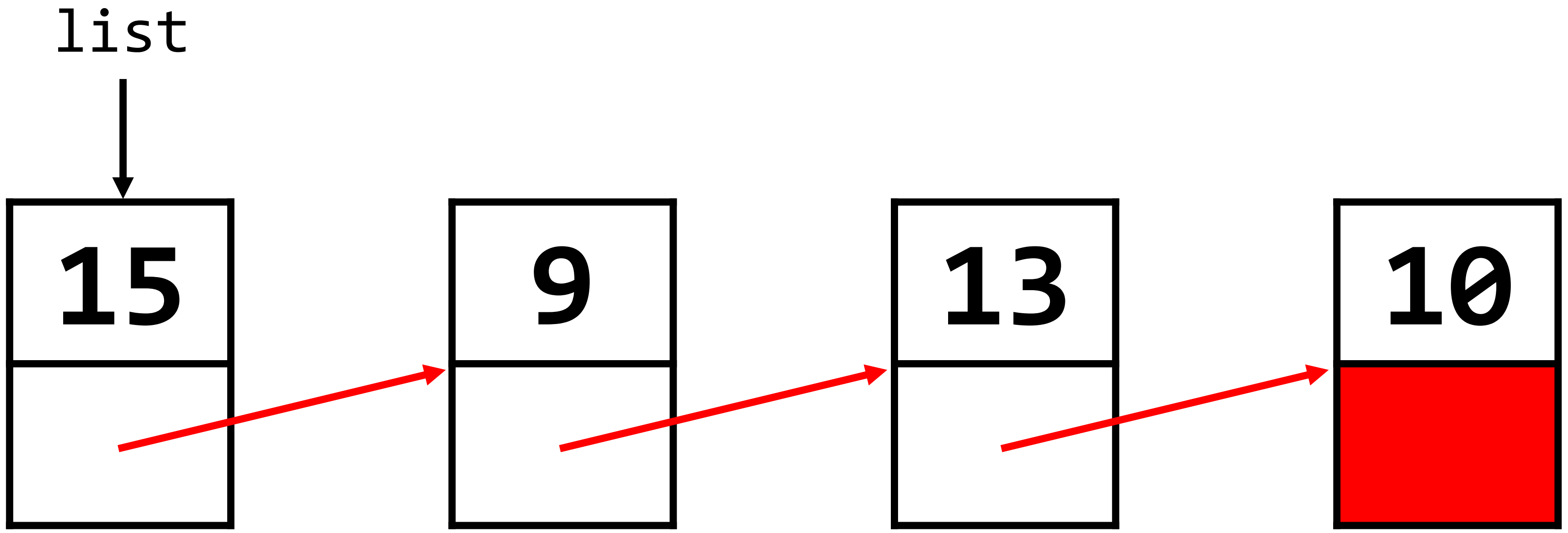
Stacks (Data Structure)

- Just make sure to always maintain a pointer to the head of the linked list!

- To **push**, dynamically allocate a new node, set its next pointer to point to the current head of the list, then move the head pointer to the newly-created node.

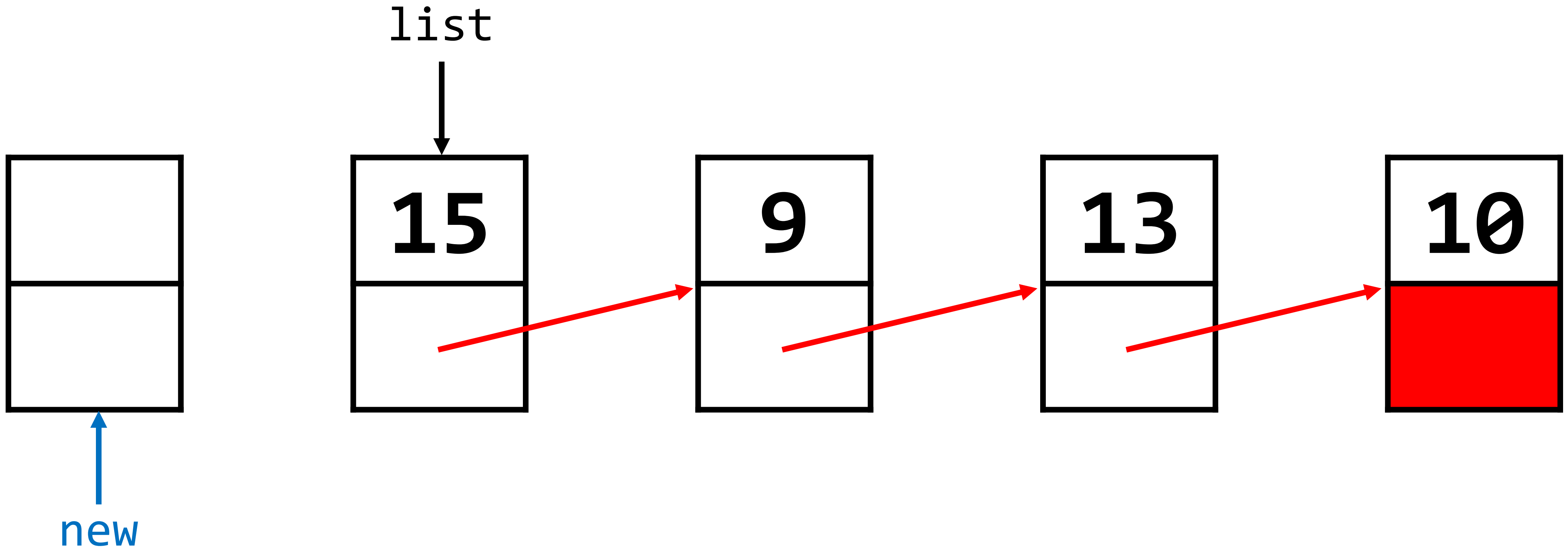
Stacks (Data Structure)

```
list = push(list, 12);
```



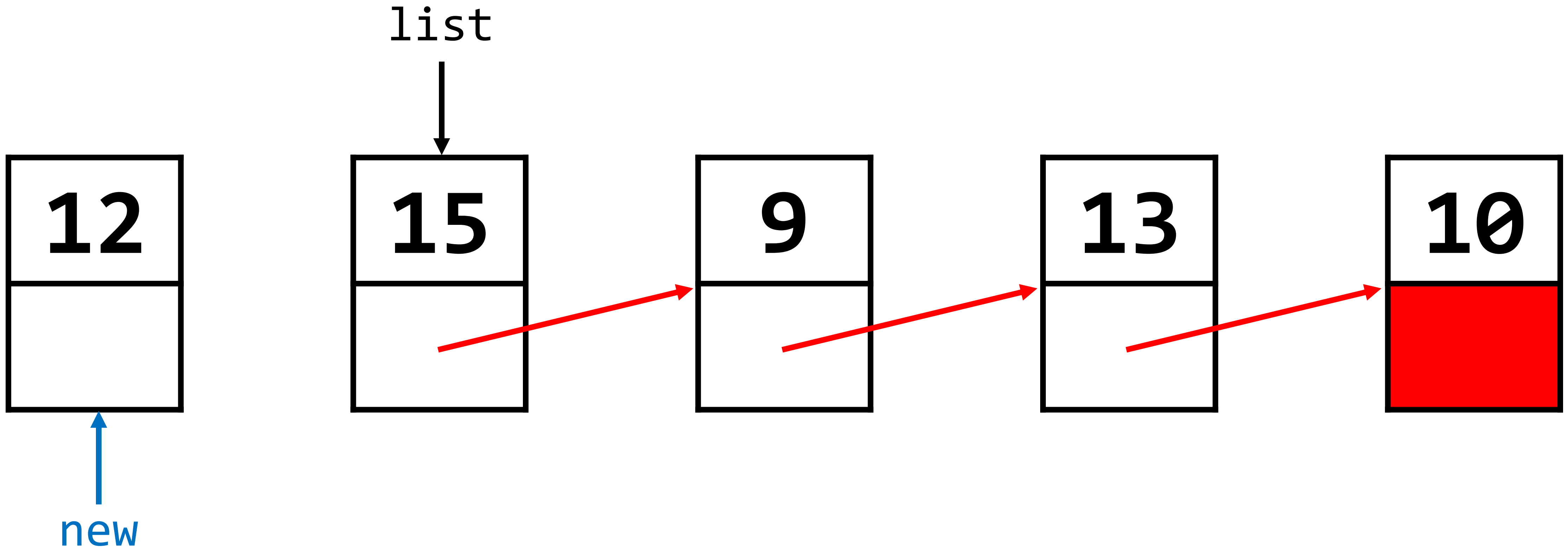
Stacks (Data Structure)

```
list = push(list, 12);
```



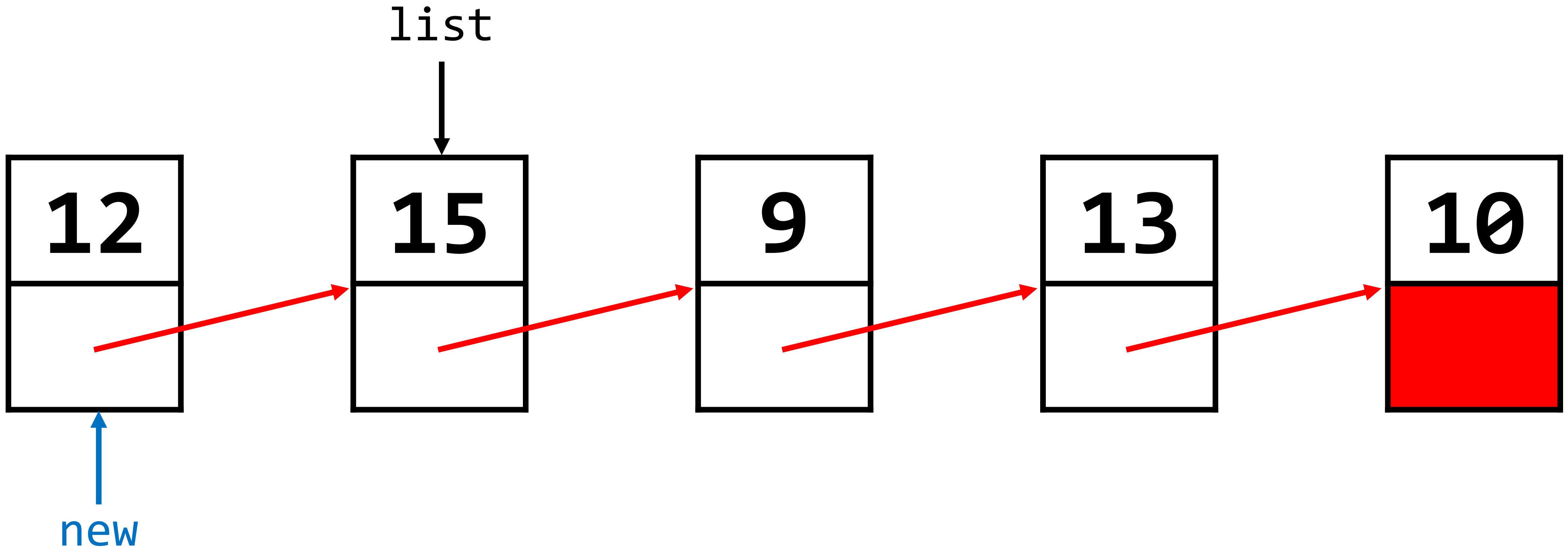
Stacks (Data Structure)

```
list = push(list, 12);
```



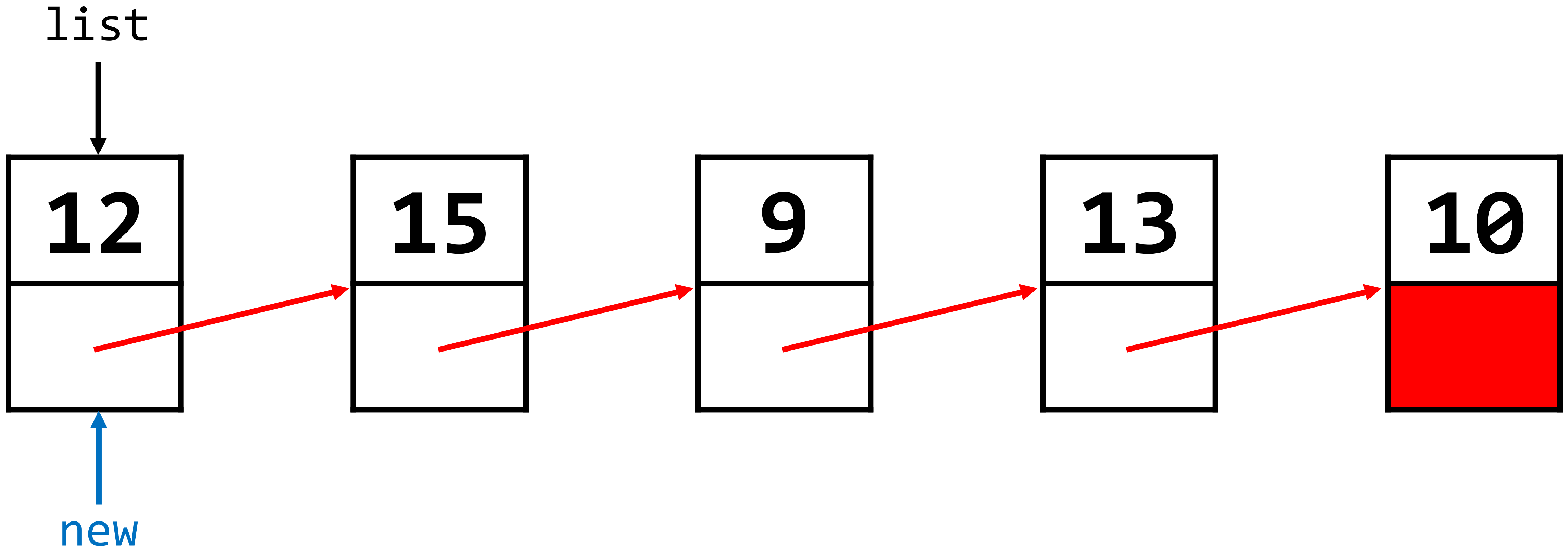
Stacks (Data Structure)

```
list = push(list, 12);
```



Stacks (Data Structure)

```
list = push(list, 12);
```

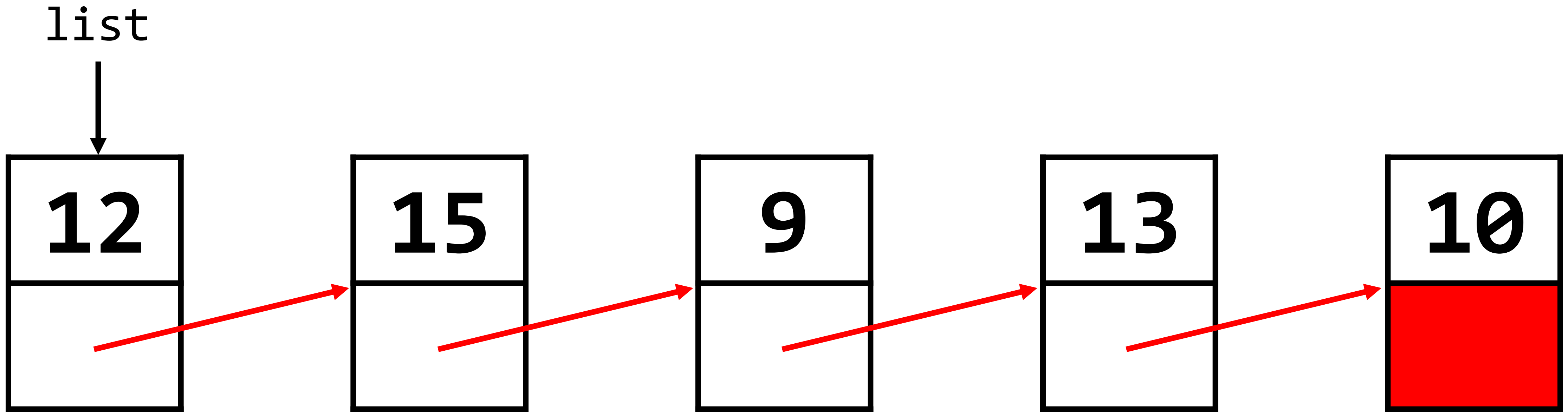


Stacks (Data Structure)

- To **pop**, traverse the linked list to its second element (if it exists), free the head of the list, then move the head pointer to the (former) second element.

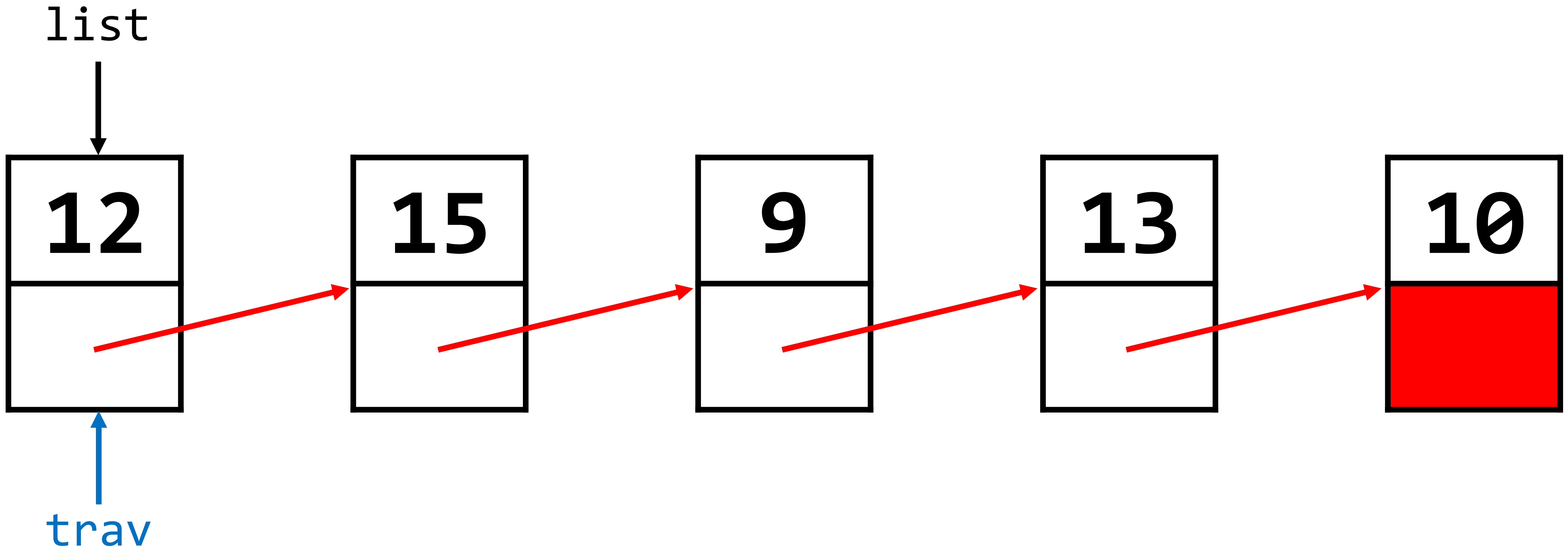
Stacks (Data Structure)

```
pop(list);
```



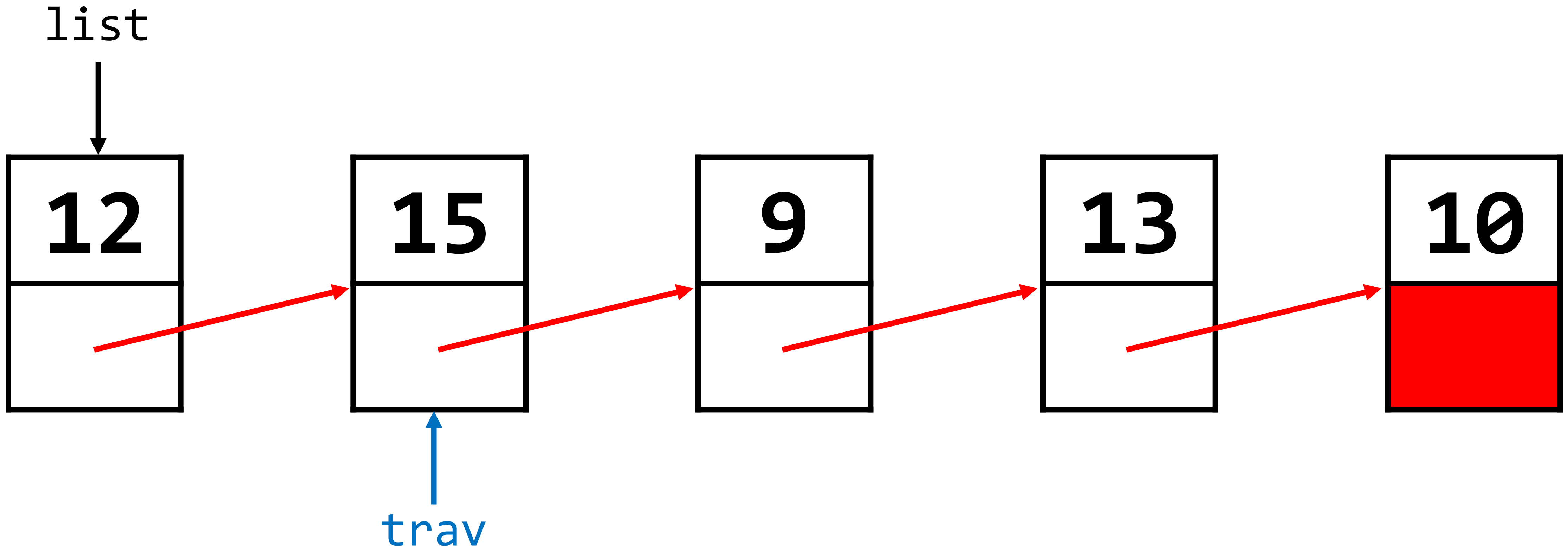
Stacks (Data Structure)

```
pop(list);
```



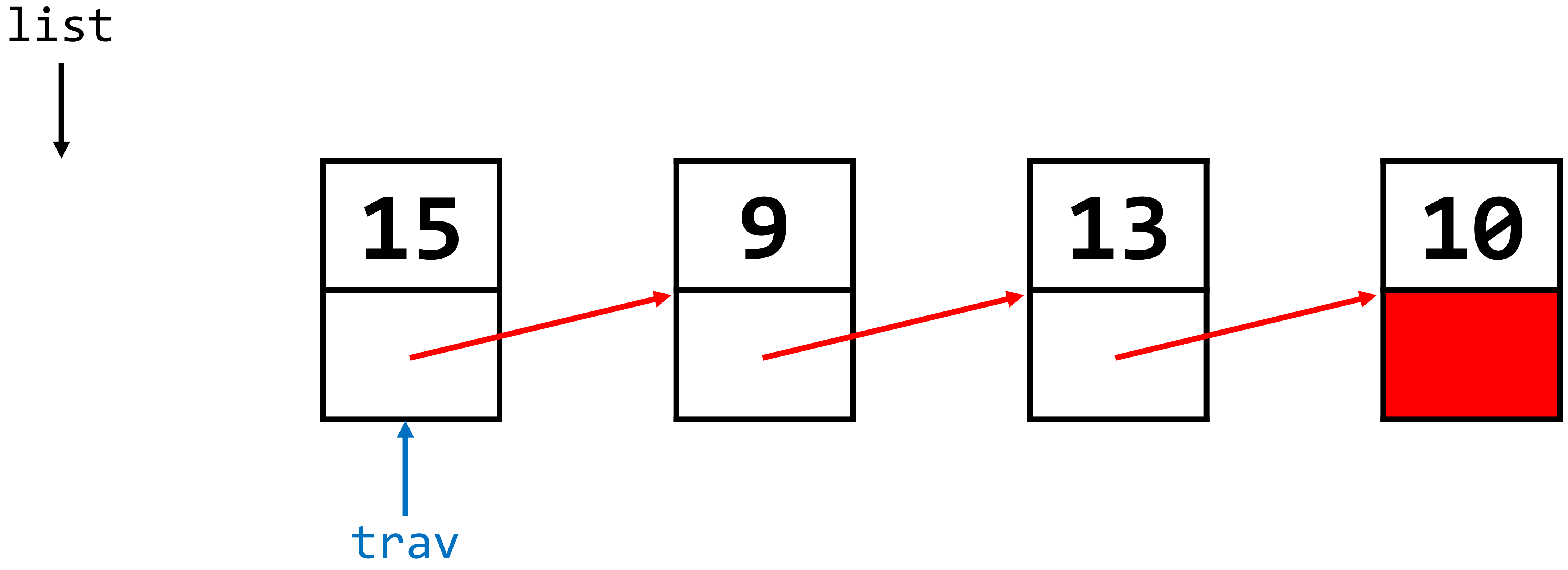
Stacks (Data Structure)

```
pop(list);
```



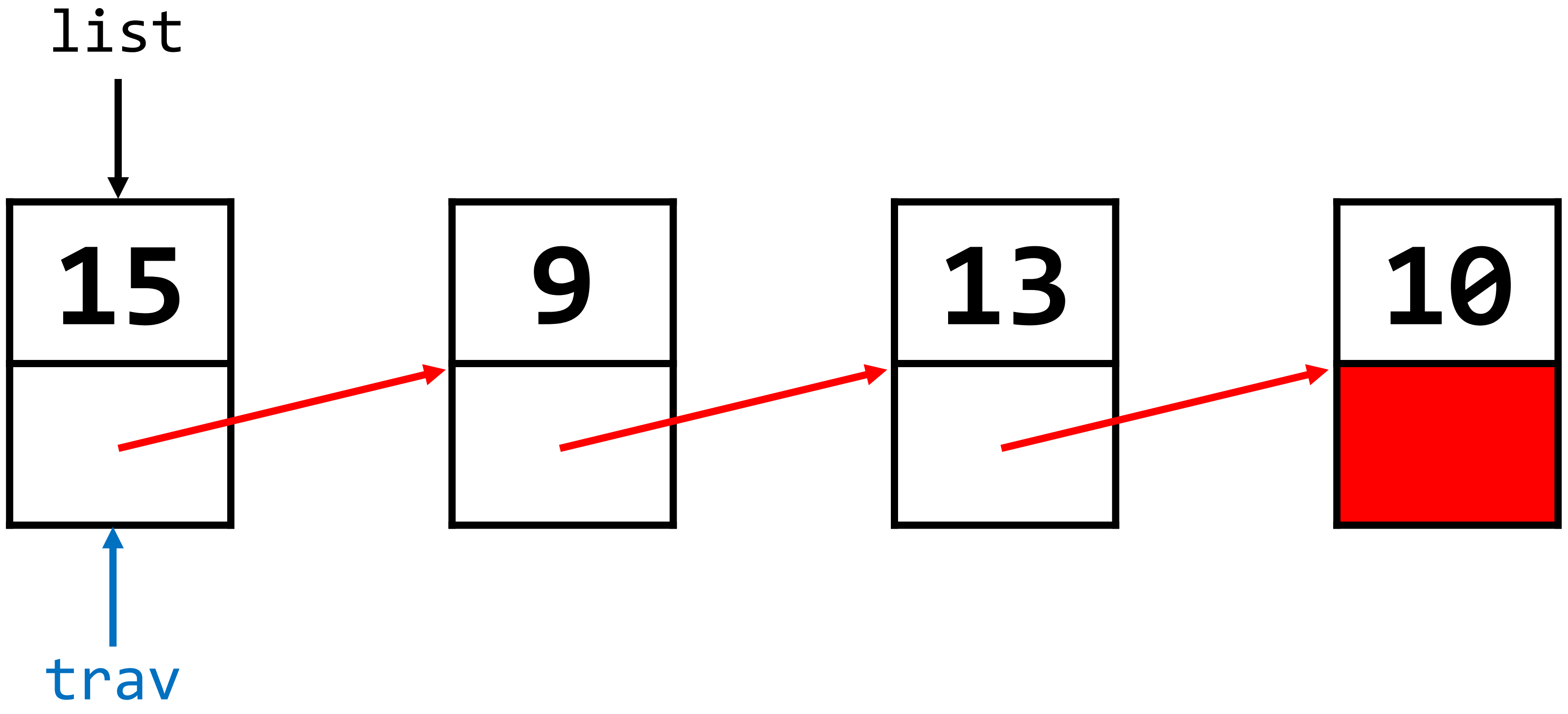
Stacks (Data Structure)

```
pop(list);
```



Stacks (Data Structure)

```
pop(list);
```


list
15
9
13
10
trav