LEC 04

CSE 123

ArrayIntList

BEFORE WE START

Talk to your neighbors:

Did you eat breakfast today? If so, what?

Music: 123 24su Lecture Tunes

Instructor: Joe Spaniac

TAs:			
Andras	Eric	Sahej	Zach
Daniel	Nicole	Trien	

Questions during Class?

Raise hand or send here

sli.do #cse123

Lecture Outline

- **Announcements** ◀
- Arrays vs. ArrayLists
- ArrayIntList
  - Fields
  - Implementing add( )
  - Capacity & Resizing

Announcements

- Check-in 1 “Graded”! (on gradescope)
- Quiz 1 Completed! 😮‍💨
  - Congrats! Expect grades back in about a week (hopefully)
  - Practice metacognition: how did that go? What can you learn about your studying process and how can you incorporate it before the next quiz?
- Programming Assignment 1 due tonight (7/3) @ 11:59pm
  - Try to get something in before the initial submission such that you can get feedback
  - Extra credit due (7/3) as well – totally ok if you don’t complete it!
- Creative Project 1 Grades out after lecture
- Resubmission period 1 closes on Friday (7/5) @ 11:59pm
  - Assignments available: C1

Lecture Outline

- Announcements
- **Arrays vs. ArrayLists** ◀
- ArrayIntList
  - Fields
  - Implementing add( )
  - Capacity & Resizing

Arrays vs. ArrayLists

Arrays	ArrayLists
`int[] arr = new int[x];`	`List<Integer> al = new ArrayList<>();`
`int y = arr[0]`	`int y = al.get(0);`
-	`al.add(2);`
`arr[0] = 5;`	`al.set(0, 5);`
`int length = arr.length;   // Always x`	`int size = al.size();   // Matches # of // things added`

Fundamental data structure*	Class within java.util
Fixed length	Illusion of resizing

** Technically arrays are also Objects in Java, but for the purposes of this course / most of your CS career we'll treat them like fundamental data structures*

Lecture Outline

- Announcements
- Arrays vs. ArrayLists
- **ArrayIntList**
  - Fields
  - Implementing add( )
  - Capacity & Resizing

Implementing Data Structures

- No different from designing any other class!
  - Specified behavior (`List` interface):

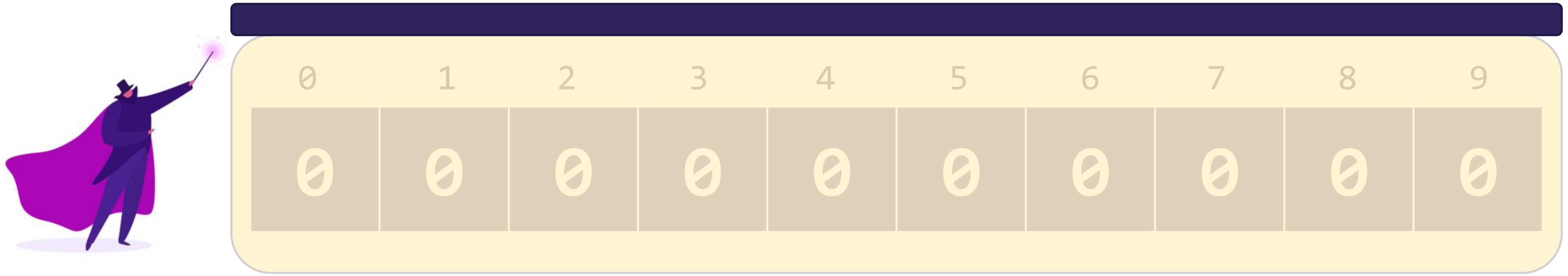
Method	Description
`add(E value)`	Adds the given value to the end of the list
`add(int index, E value)`	Adds the given value at the given index
`remove(E value)`	Removes the given value if it exists
`remove(int index)`	Removes the value at the given index
`get(int index)`	Returns the value at the given index
`set(int index, int value)`	Updates the value at the given index to the one given
`size()`	Returns the number of elements in the list

  - Choose appropriate fields based on behavior
- Just requires some thinking outside the box

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:

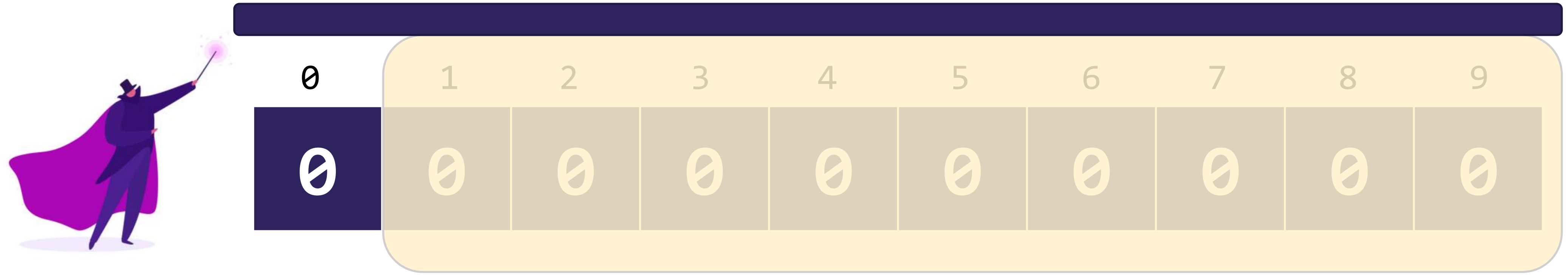
```
- int[] elementData;    // Where we store elements
- int size;             // Storage boundary
```



```
al.add(2);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`

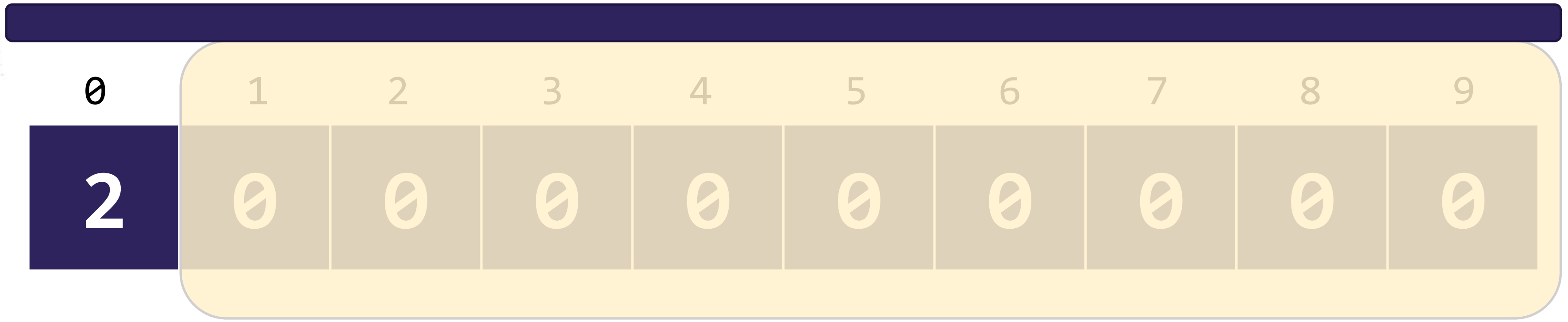


```
al.add(2);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:

```
- int[] elementData;     // Where we store elements
- int size;              // Storage boundary
```

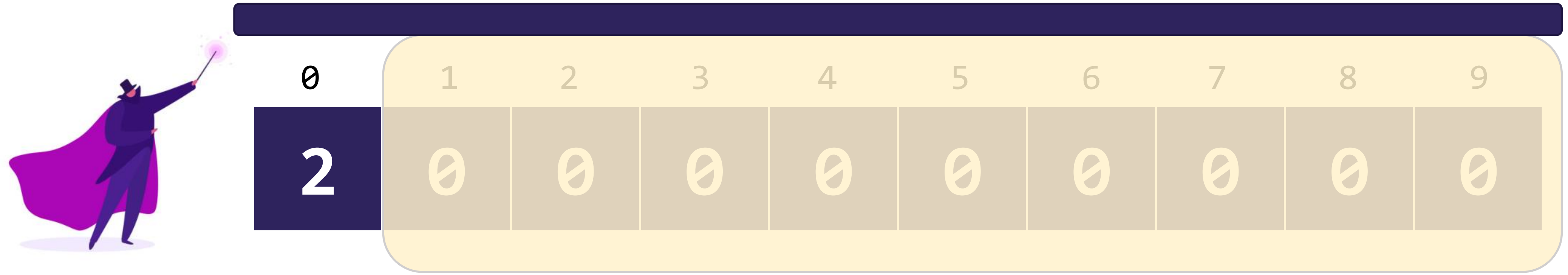


```
al.add(2);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:

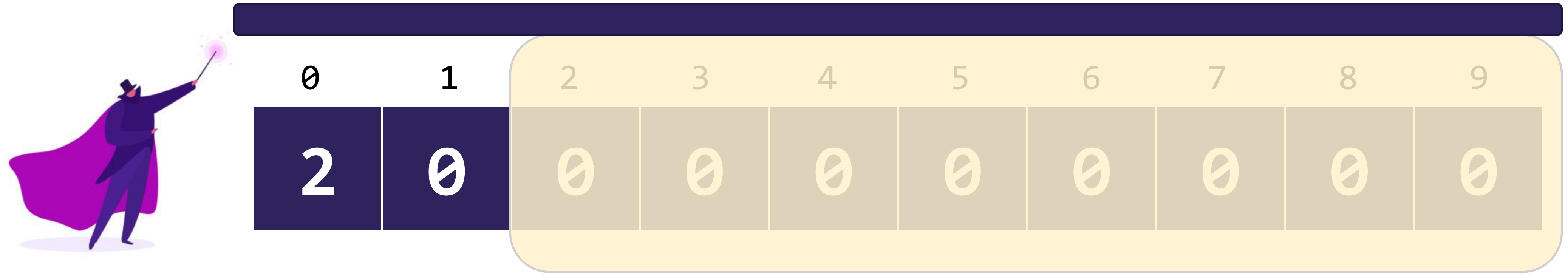
```
- int[] elementData;     // Where we store elements
- int size;              // Storage boundary
```



```
al.add(5);
```

ArrayIntLists

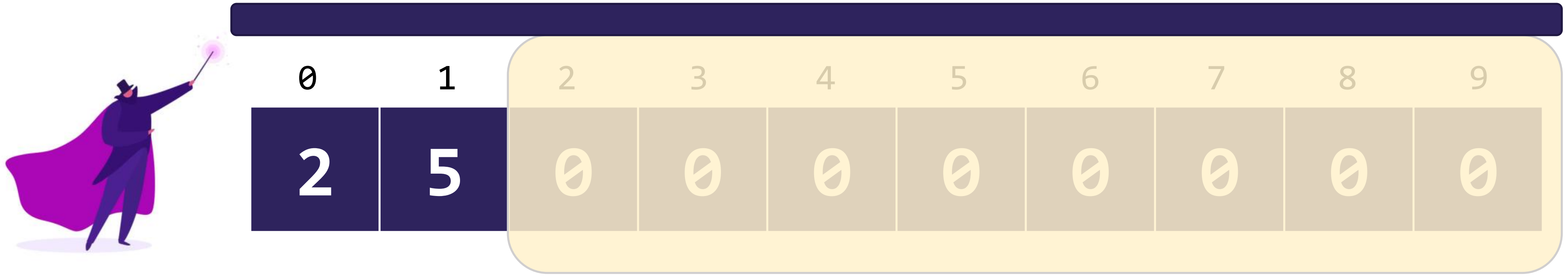
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;    // Where we store elements`
  - `int size;             // Storage boundary`



```
al.add(5);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`

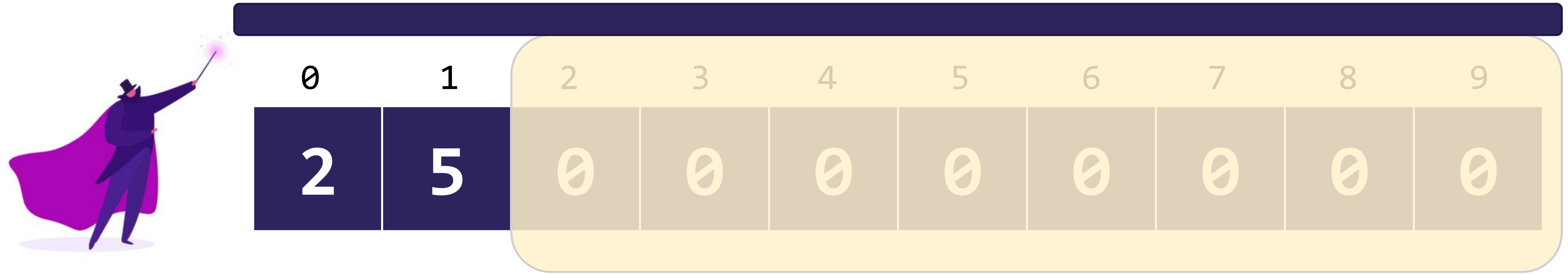


```
al.add(5);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:

```
- int[] elementData;     // Where we store elements
- int size;              // Storage boundary
```

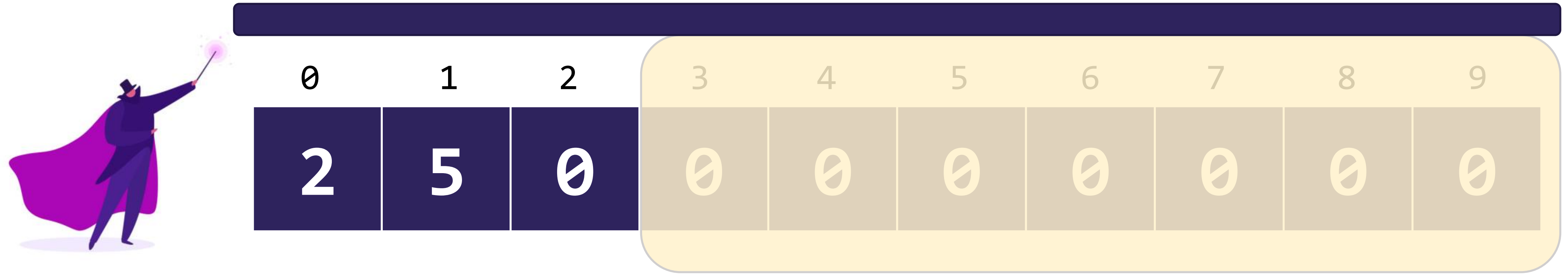


```
al.add(-1);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:

```
  - int[] elementData;    // Where we store elements
  - int size;             // Storage boundary
```

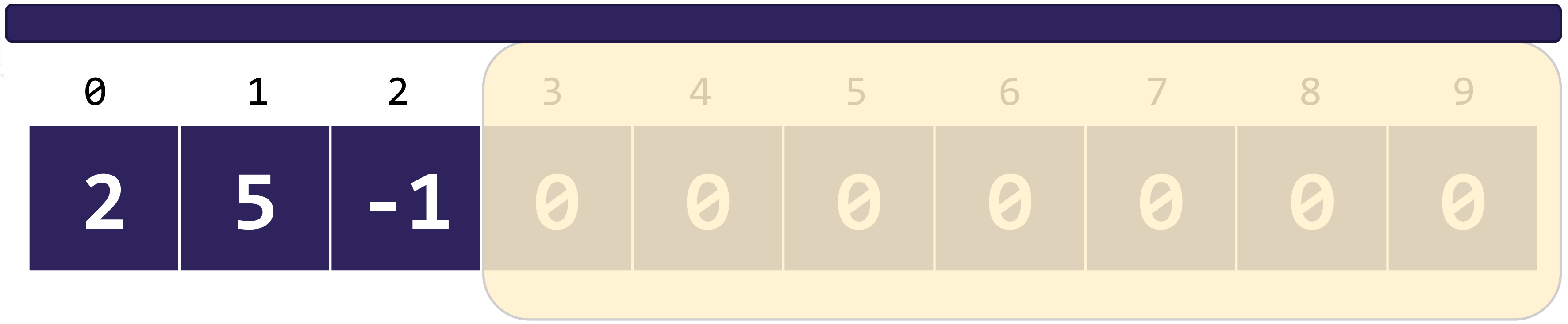


```
al.add(-1);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:

```
- int[] elementData;     // Where we store elements
- int size;              // Storage boundary
```



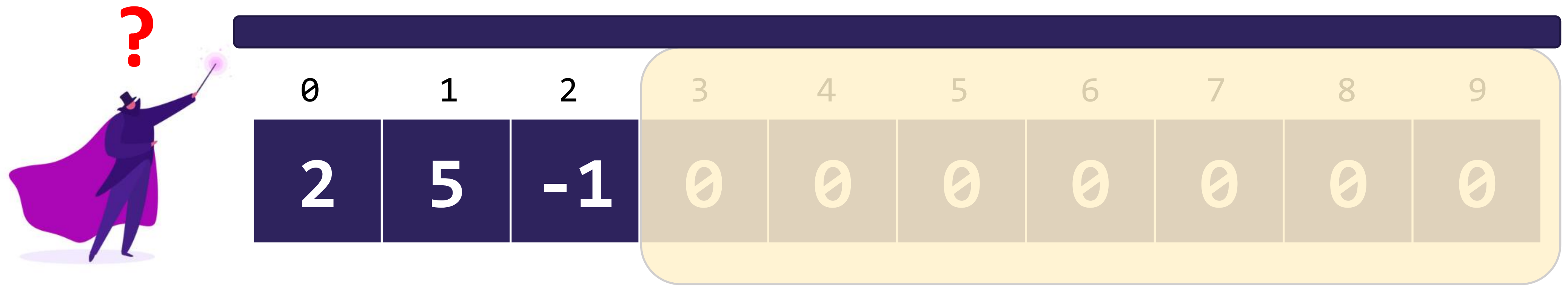
```
al.add(-1);
```

Lecture Outline

- Announcements
- Arrays vs. ArrayLists
- **ArrayIntList**
  - Fields
  - **Implementing add( )**
  - Capacity & Resizing

ArrayIntLists

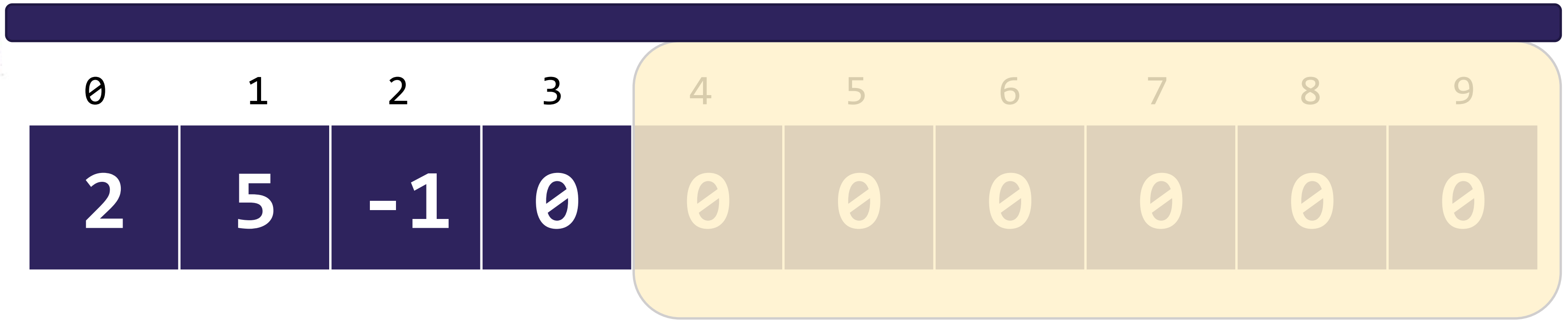
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  ```
  - int[] elementData;     // Where we store elements
  - int size;              // Storage boundary
  ```



```
al.add(0, 0);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  ```
  - int[] elementData;     // Where we store elements
  - int size;              // Storage boundary
  ```

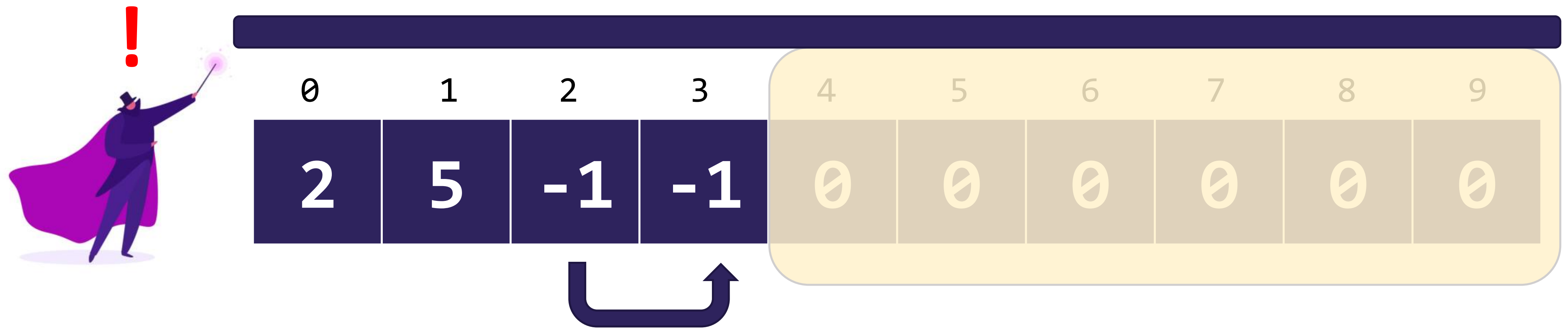


```
al.add(0, 0);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:

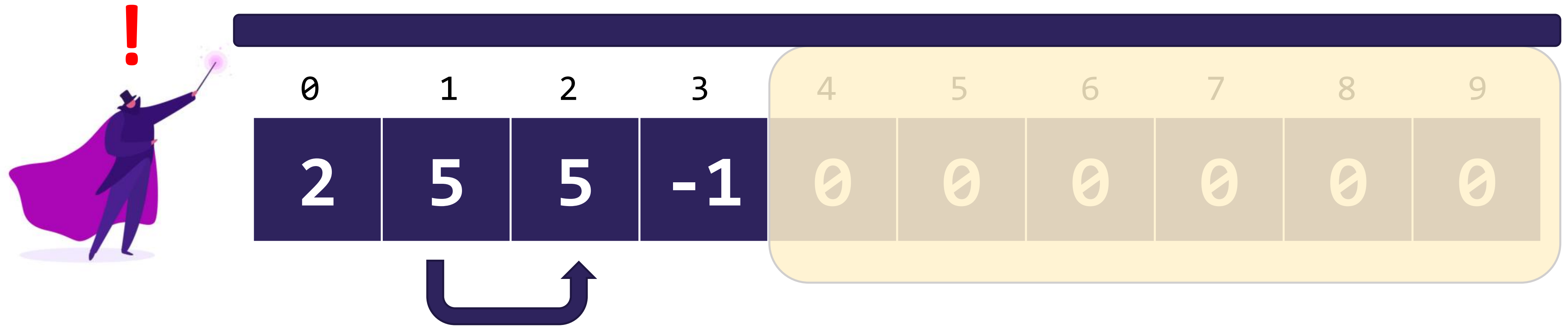
```
- int[] elementData;     // Where we store elements
- int size;              // Storage boundary
```



```
al.add(0, 0);
```

ArrayIntLists

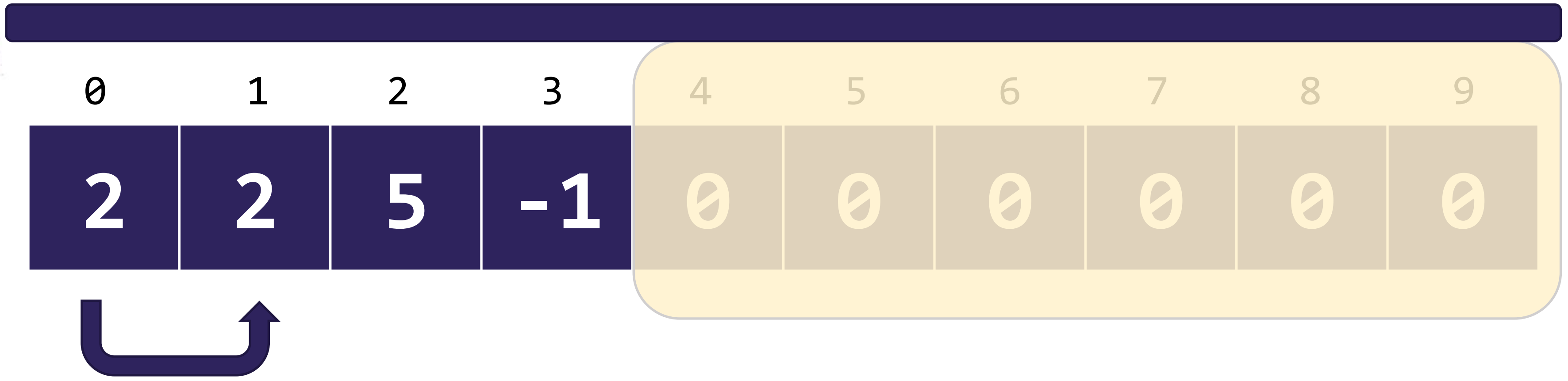
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`



```
al.add(0, 0);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`

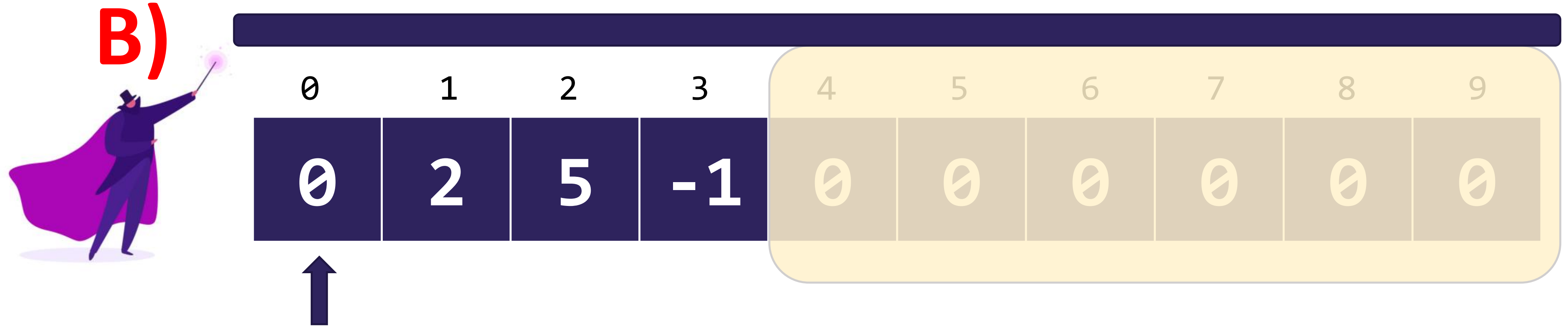


```
al.add(0, 0);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`

B)



```
al.add(0, 0);
```

ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:

```
- int[] elementData;    // Where we store elements
- int size;             // Storage boundary
```

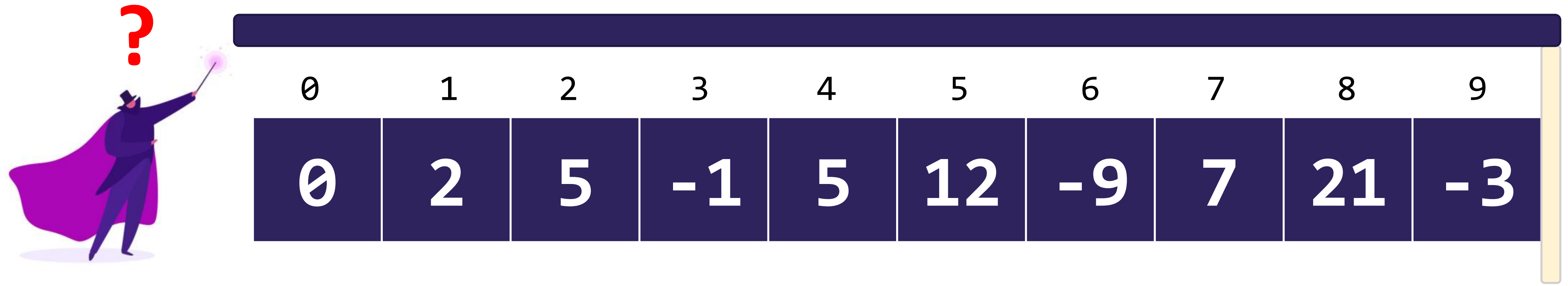
- Important points:
  - `size` represents how far the curtain is peeled back
    - Can’t use a hardcoded value!
  - Starting value is always at index 0
    - Adding to / removing from beginning requires shifting elements

Lecture Outline

- Announcements
- Arrays vs. ArrayLists
- ArrayIntList
  - Fields
  - Implementing add( )
  - **Capacity & Resizing**

Capacity and Resizing

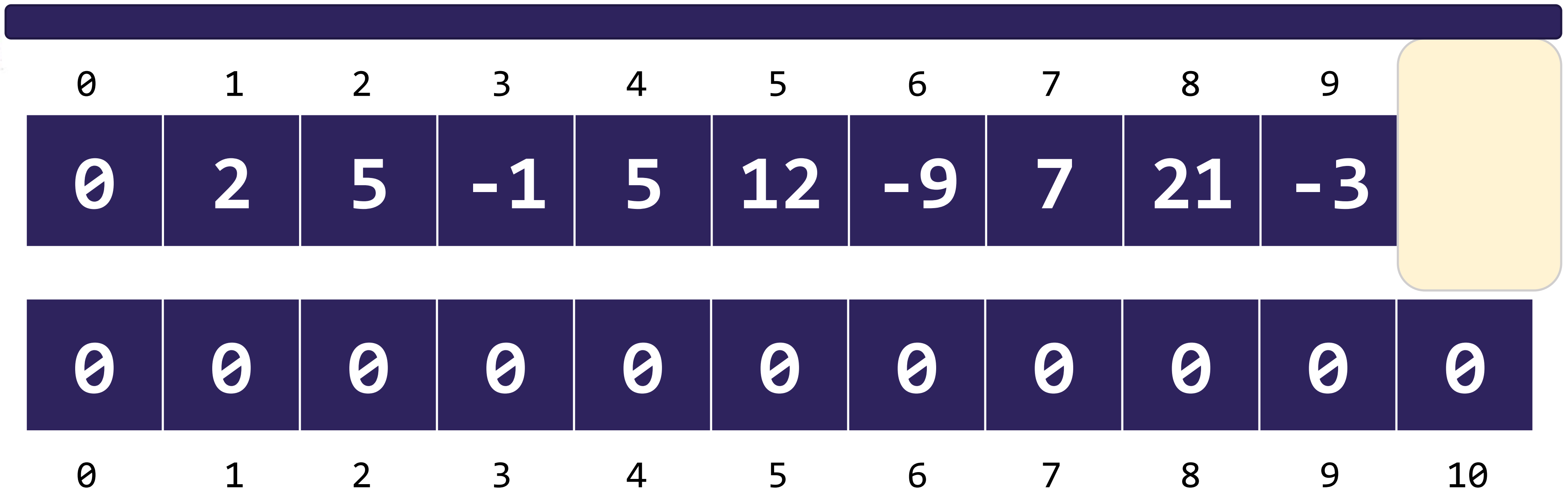
- `Capacity` = length of underlying array
- `Size` = number of user-added elements
- What happens if we run out of space? (`size == capacity`)



`al.add(2);`

Capacity and Resizing

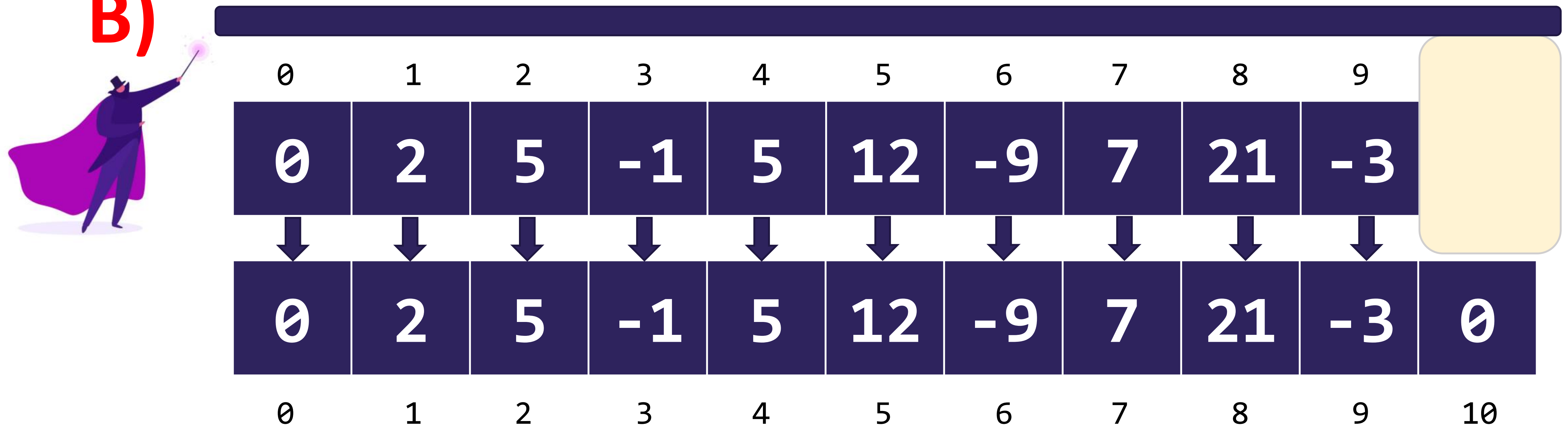
- `Capacity` = length of underlying array
- `Size` = number of user-added elements
- What happens if we run out of space? (`size == capacity`)



Capacity and Resizing

- `Capacity` = length of underlying array
- `Size` = number of user-added elements
- What happens if we run out of space? (`size == capacity`)

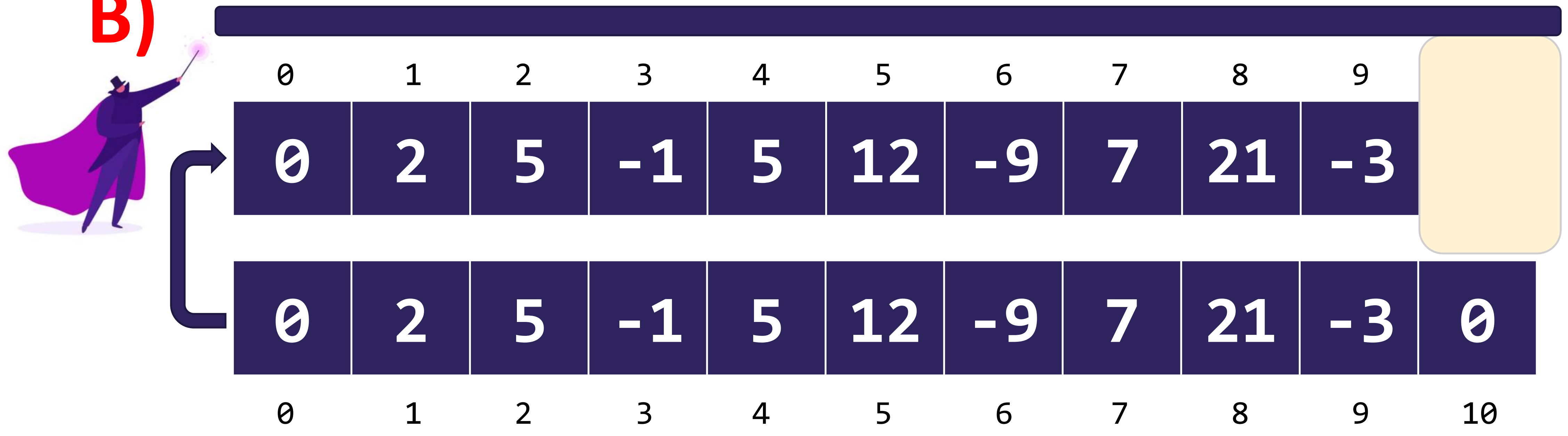
B)



Capacity and Resizing

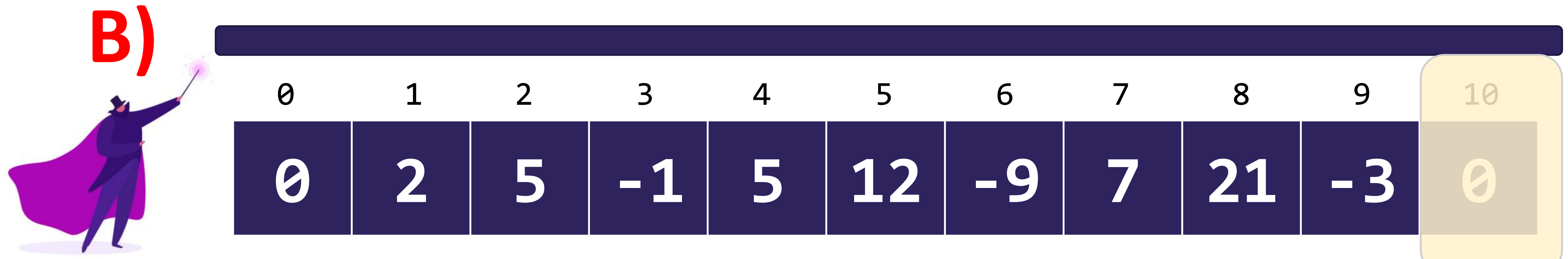
- `Capacity` = length of underlying array
- `Size` = number of user-added elements
- What happens if we run out of space? (`size == capacity`)

B)



Capacity and Resizing

- `Capacity` = length of underlying array
- `Size` = number of user-added elements
- What happens if we run out of space? (`size == capacity`)

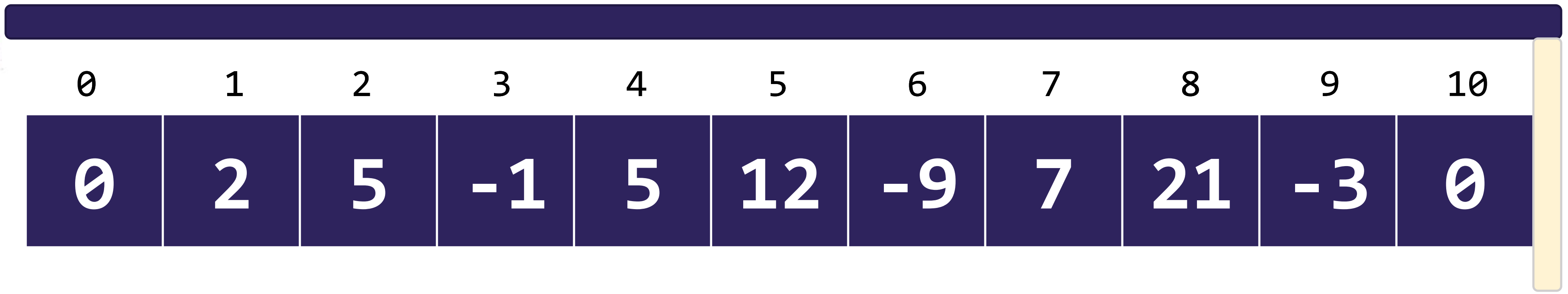


```
al.add(2);
```

Capacity and Resizing

- `Capacity` = length of underlying array
- `Size` = number of user-added elements
- What happens if we run out of space? (`size == capacity`)

B)

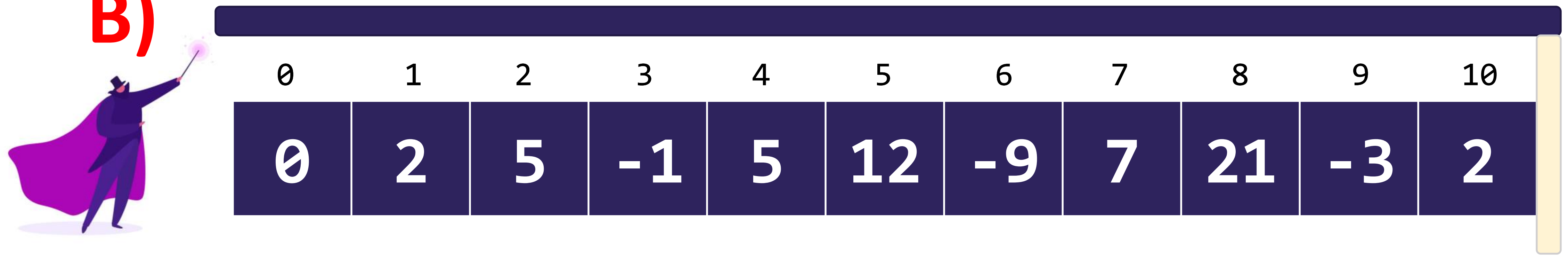


```
al.add(2);
```

Capacity and Resizing

- `Capacity` = length of underlying array
- `Size` = number of user-added elements
- What happens if we run out of space? (`size == capacity`)

B)



```
al.add(2);
```

Capacity and Resizing

- `Capacity` = length of underlying array
- `Size` = number of user-added elements
- What happens if we run out of space? (`size == capacity`)
    - We make a new (bigger array) and copy things over
    - Another layer to the resizing illusion!

- In reality, we don’t typically add a single spot
    - What happens if we add again?