LEC 01

CSE 123

Implementing Data Structures; `ArrayIntList`

Questions during Class?
Raise hand or send here

sli.do #cse123

BEFORE WE START

Talk to your neighbors:

What did you have breakfast today?

Respond on sli.do!

Instructor: Trien Vuong

TAs: Cora, Jonah, Maitreyi, Nhan, Nichole, Shiven

Music: CSE 123 26su Lecture Tunes

Coming up…

- ❓ Complete the Introductory Survey
  - This helps us gather data about the students taking our classes and their backgrounds, to inform future offerings.
- 💻 The IPL opens Monday, June 29th
  - Schedule posted soon
- 🔎 Programming Assignment 0: Search Engine out!
  - Due Wednesday, July 1st, 11:59pm

Lecture Outline: Interface vs. Implementation

- Implementing `ArrayList`

Interface vs. Implementation

- Interface: what something *should* do

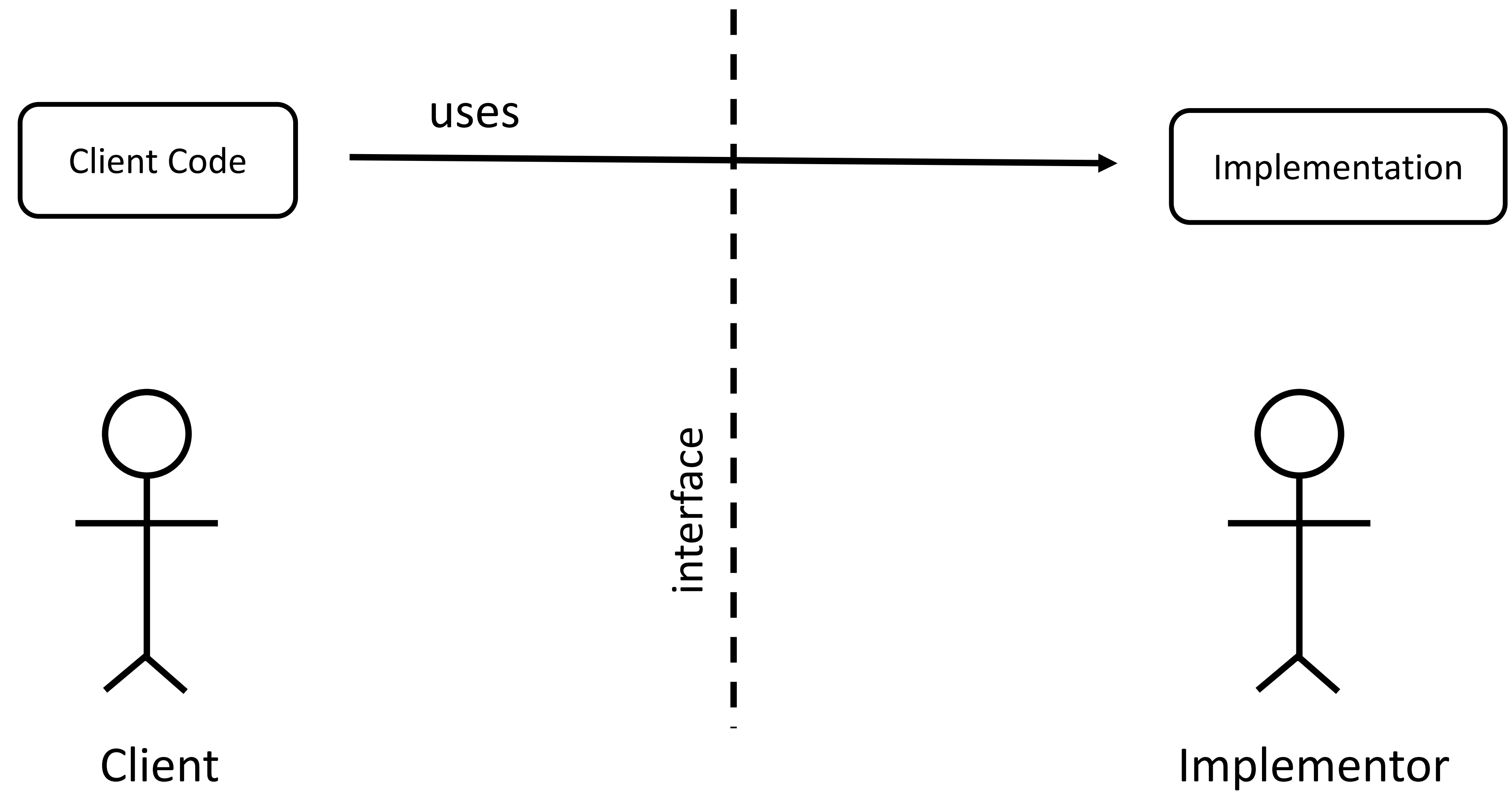
- Implementation: *how* something is done

- These are different!

- Big theme of CSE 123:

choose between different implementations of same interface

Client vs. Implementor



Arrays vs. ArrayLists (Review)

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

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

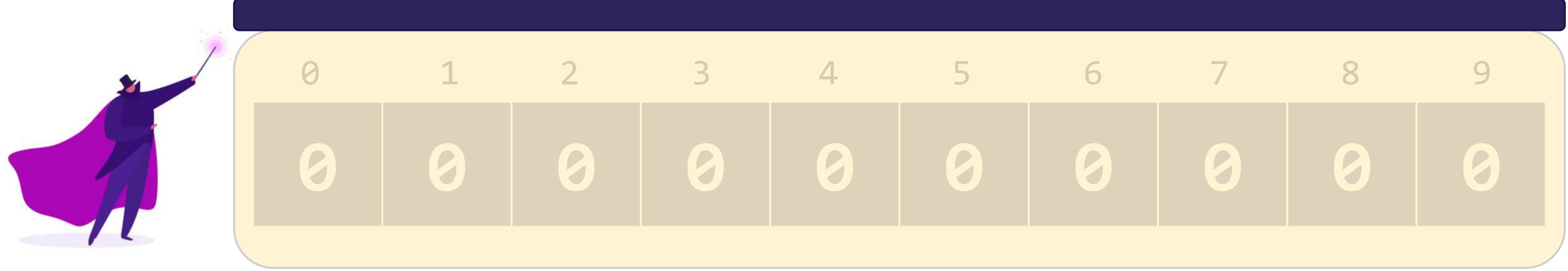
Lecture Outline: Implementing ArrayList

- Interface v. Implementation
- **Implementing `ArrayList`**

ArrayIntLists (step 1)

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

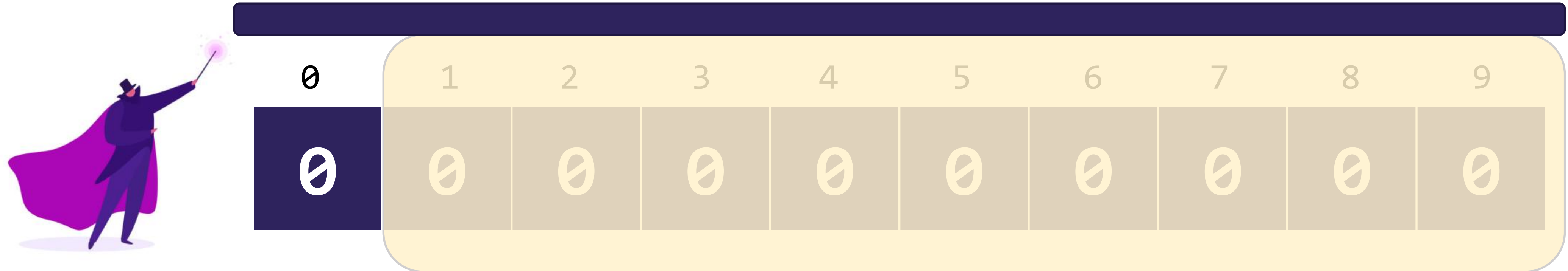
```
- int[] elementData;    // Where we store elements
- int size;             // Storage boundary
```




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

ArrayIntLists (step 2)

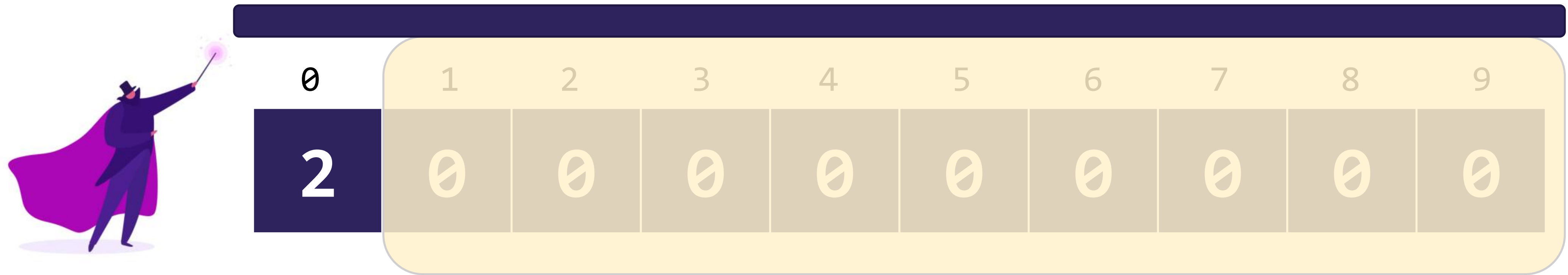
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;    // Where we store elements`
  - `int size;             // Storage boundary`




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

ArrayIntLists (step 3)

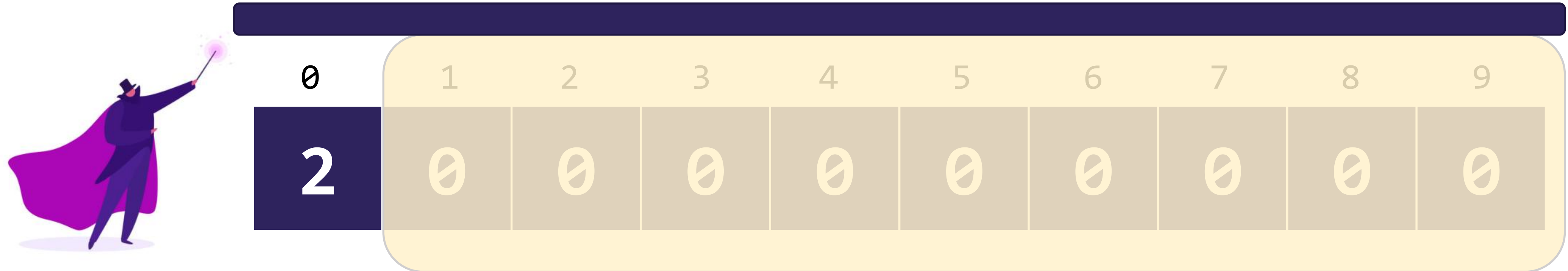
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`




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

ArrayIntLists (step 4)

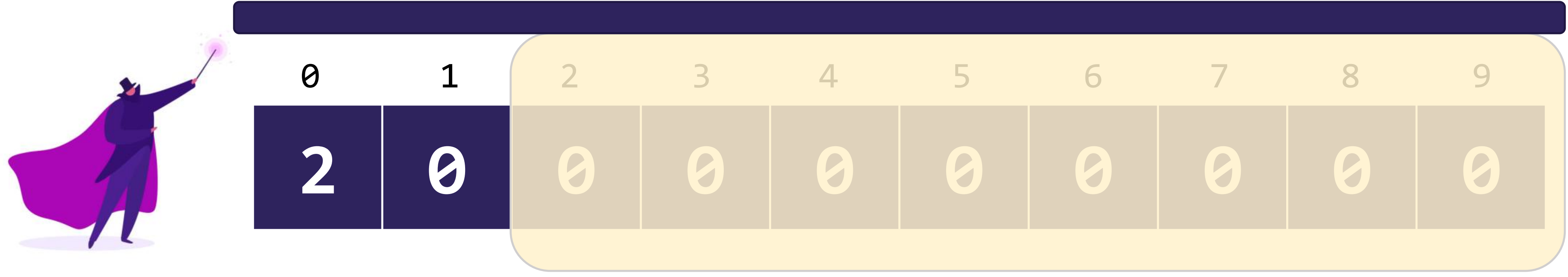
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`



`al.add(5);`

ArrayIntLists (step 5)

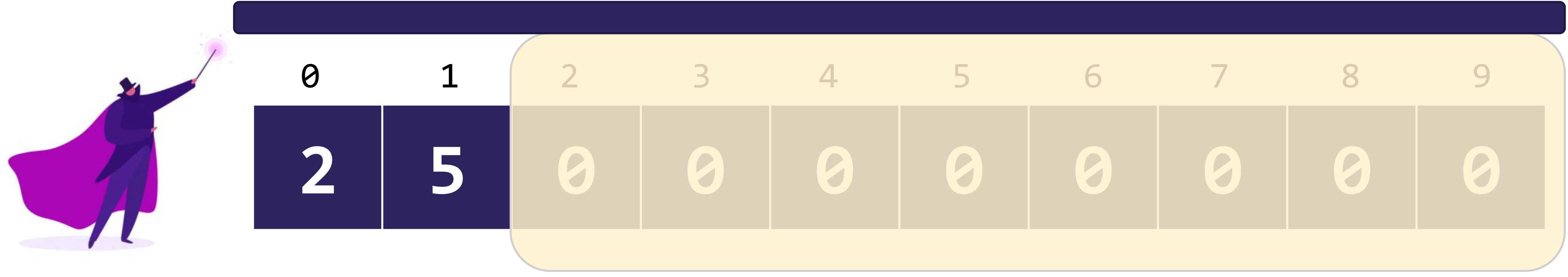
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;    // Where we store elements`
  - `int size;             // Storage boundary`



`al.add(5);`

ArrayIntLists (step 6)

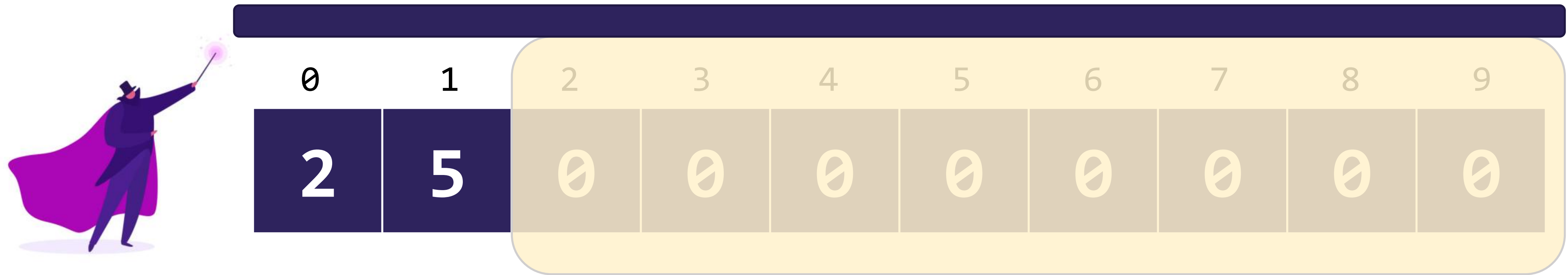
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;    // Where we store elements`
  - `int size;             // Storage boundary`




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

ArrayIntLists (step 7)

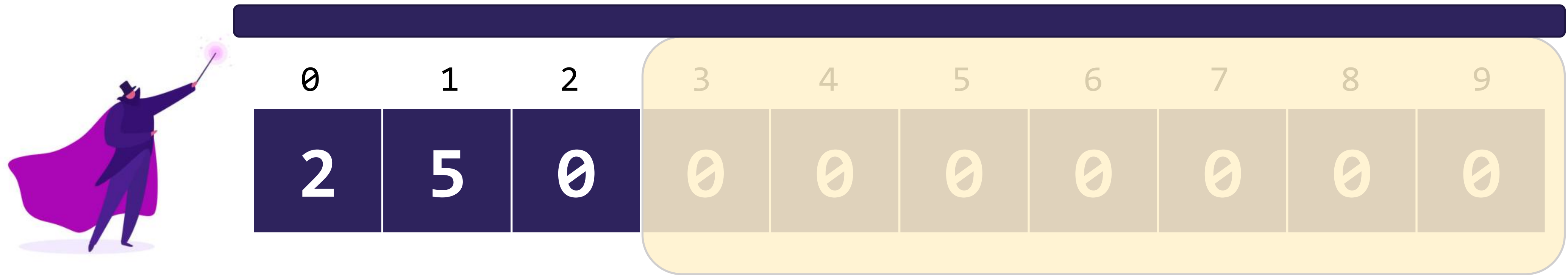
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`




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

ArrayIntLists (step 8)

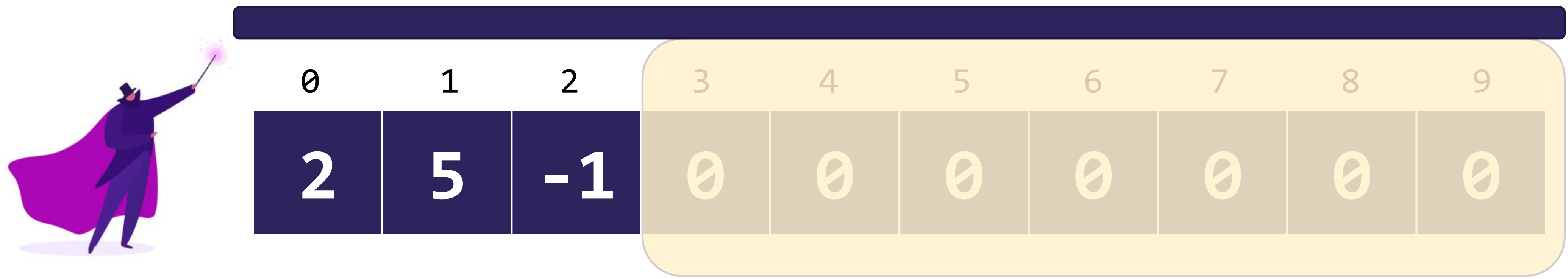
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`




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

ArrayIntLists (step 9)

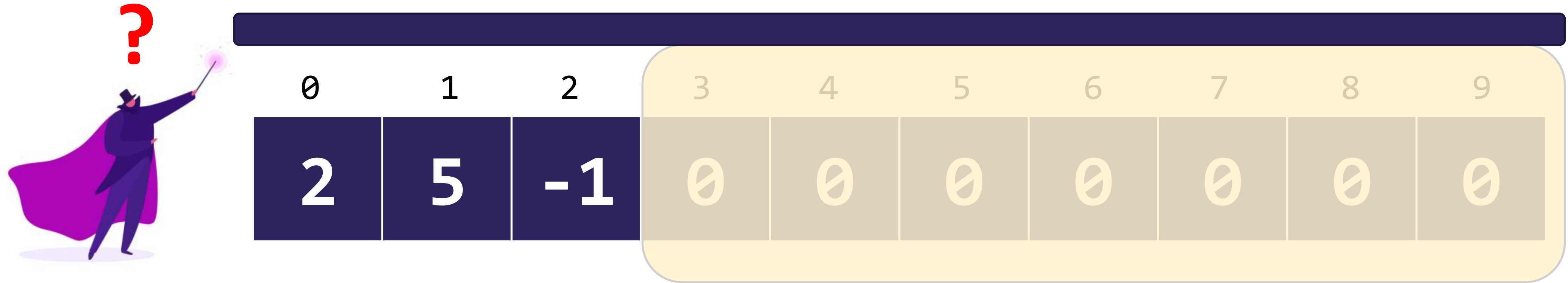
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;      // Where we store elements`
  - `int size;               // Storage boundary`




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

ArrayIntLists (step 10)

- 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`

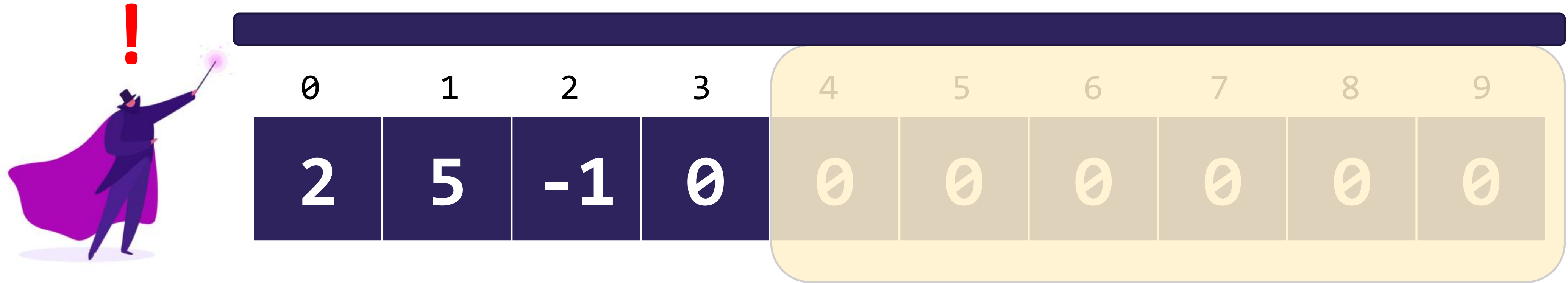



```
al.add(0, 0);
```

brainstorming

ArrayIntLists (step 11)

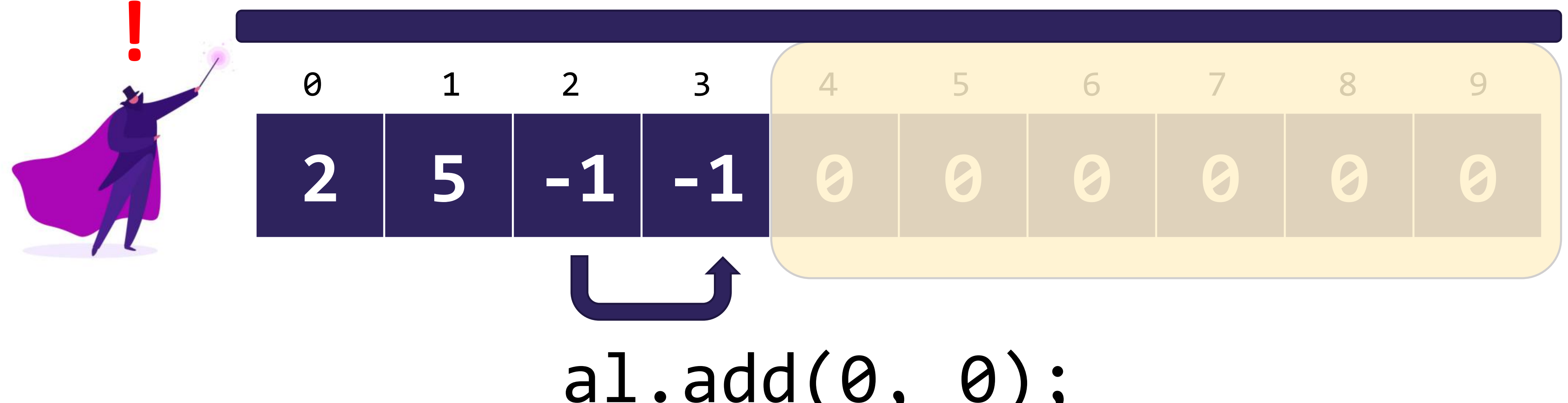
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`




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

ArrayIntLists (step 12)

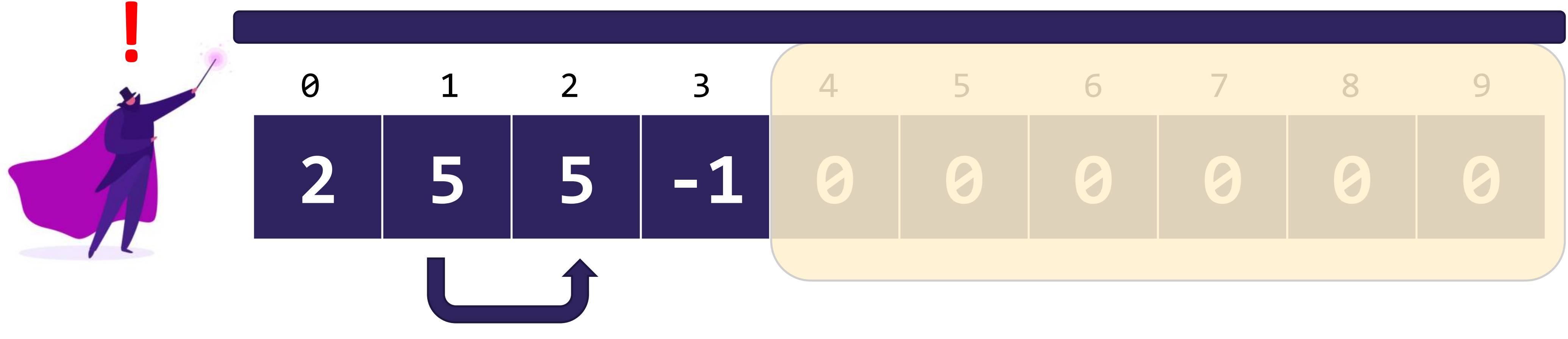
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`




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

ArrayIntLists (step 13)

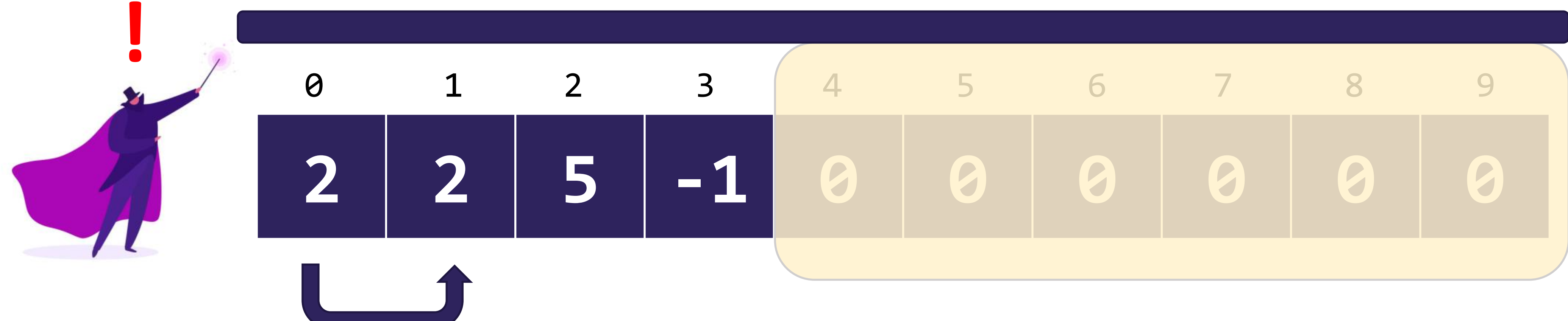
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;     // Where we store elements`
  - `int size;              // Storage boundary`




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

ArrayIntLists (step 14)

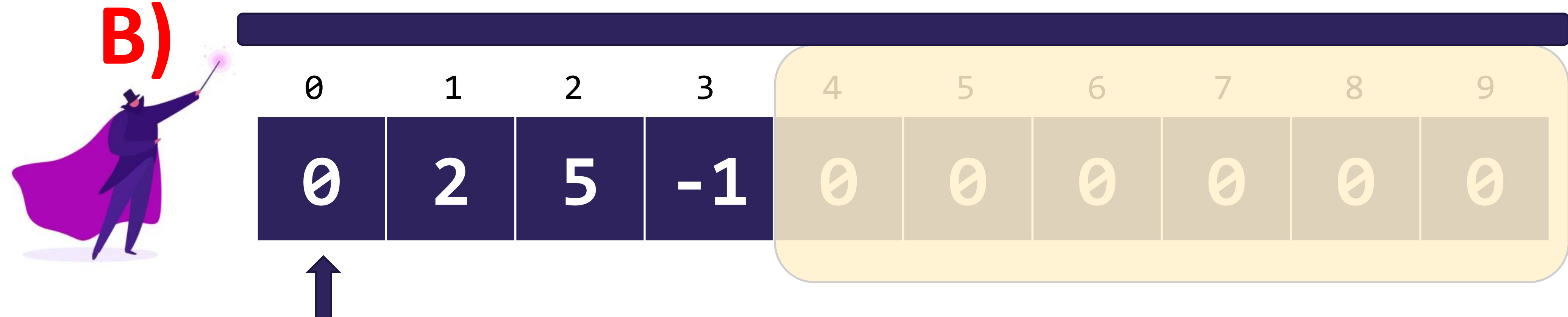
- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;      // Where we store elements`
  - `int size;               // Storage boundary`



`al.add(0, 0);`

ArrayIntLists (step 15)

- 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);
```

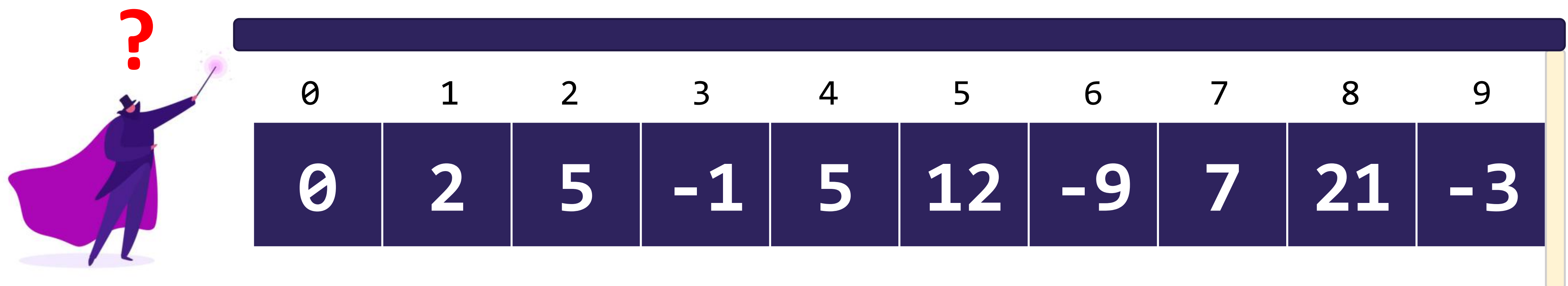
ArrayIntLists Summary

- 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

Capacity and Resizing

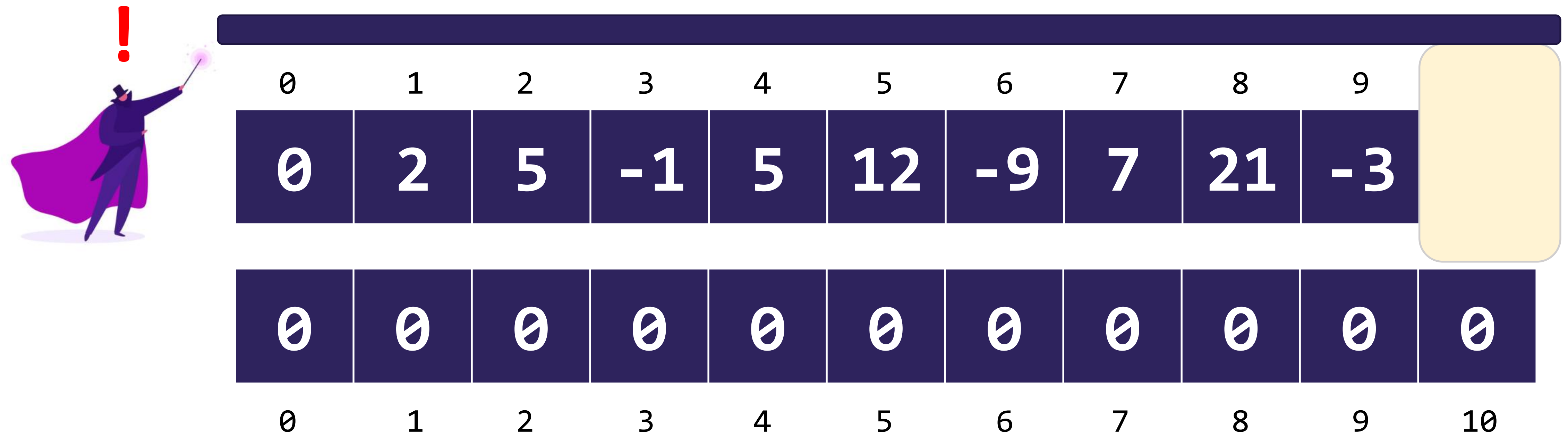
- `Capacity` = length of underlying array
- `Size` = number of user-added elements
- What happens if we run out of space? (`size == capacity`)




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

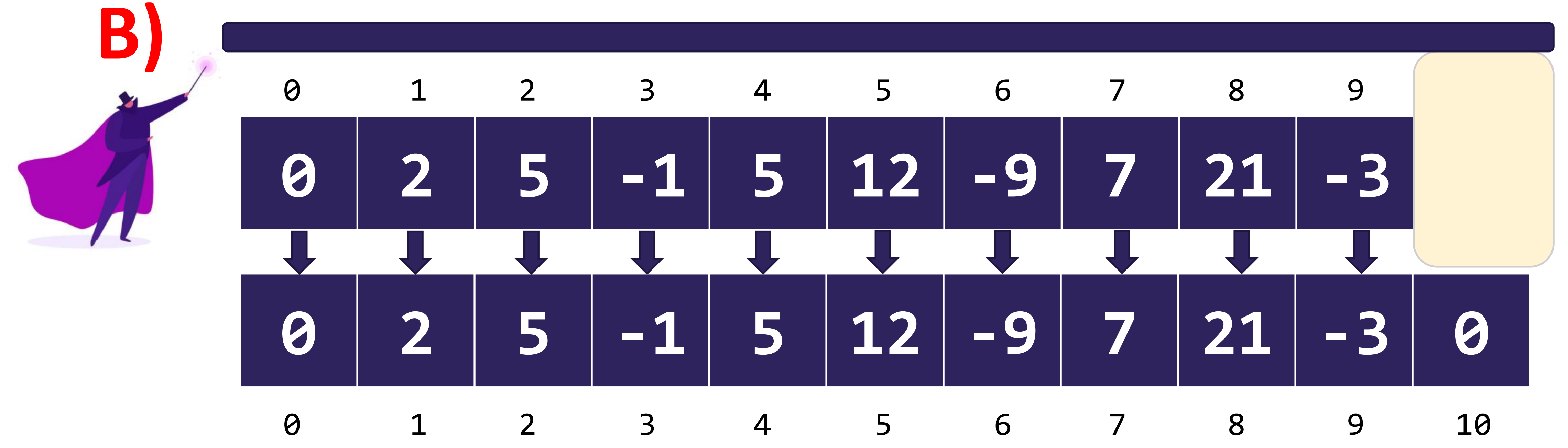
Capacity and Resizing (step 1)

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



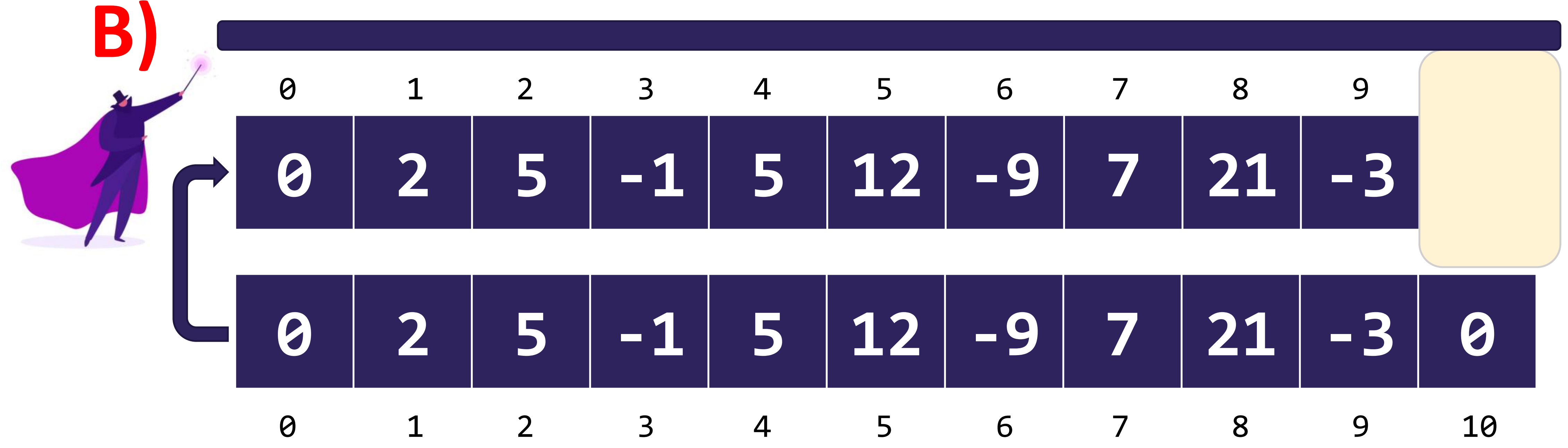
Capacity and Resizing (step 2)

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



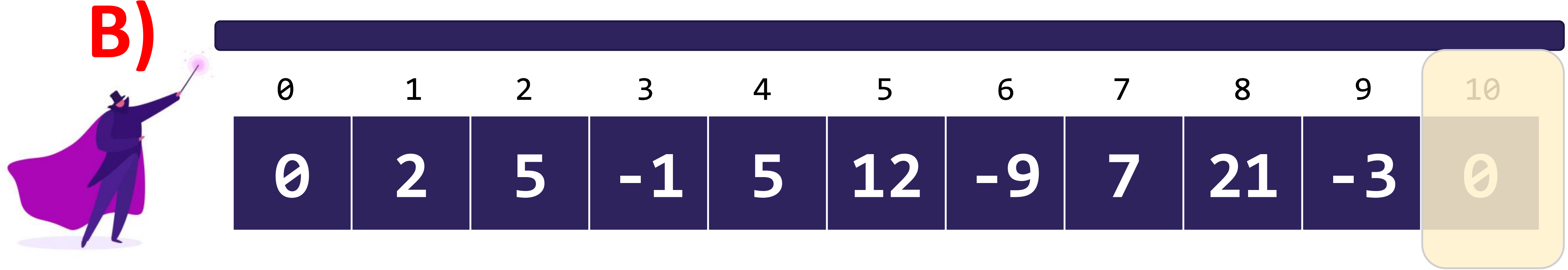
Capacity and Resizing (step 3)

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



Capacity and Resizing (step 4)

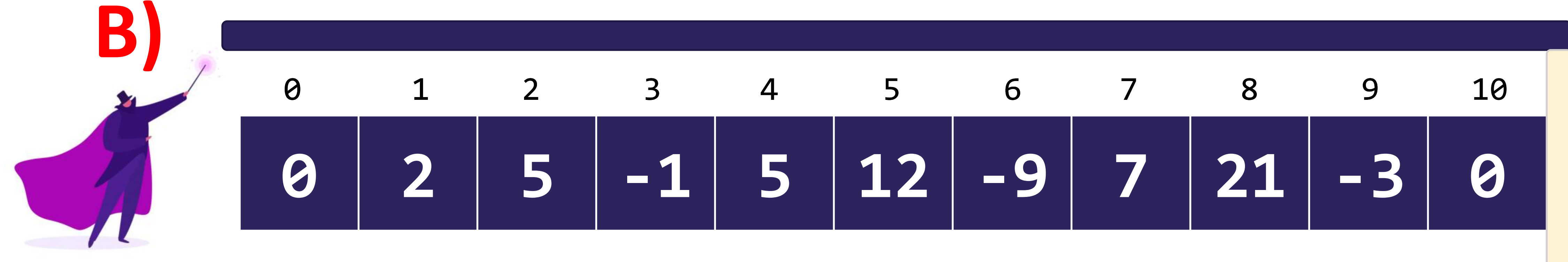
- `Capacity` = length of underlying array
- `Size` = number of user-added elements
- What happens if we run out of space? (`size == capacity`)




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

Capacity and Resizing (step 5)

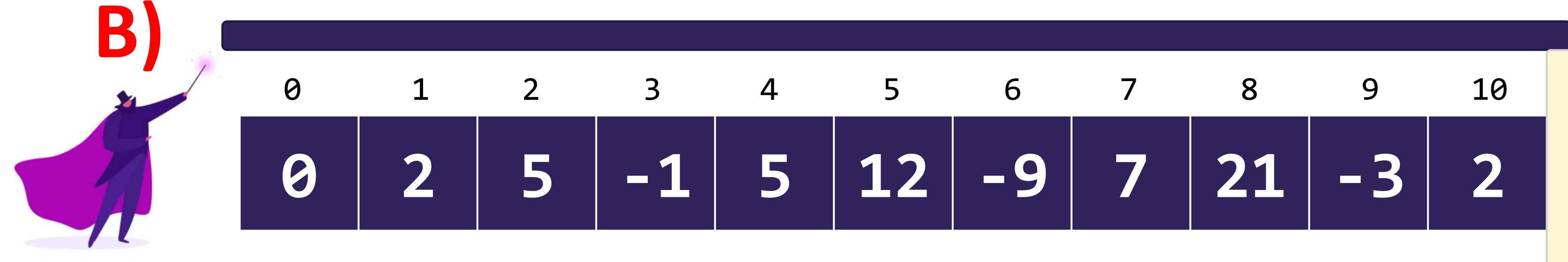
- `Capacity` = length of underlying array
- `Size` = number of user-added elements
- What happens if we run out of space? (`size == capacity`)




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

Capacity and Resizing (step 6)

- `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 Summary

- `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?