Doubly Linked Lists

Lecture 8

1107186 – Estruturas de Dados

Prof. Christian Azambuja Pagot
CI / UFPB

Universidade Federal da Paraíba
Centro de Informática

Array x Single Linked Lists

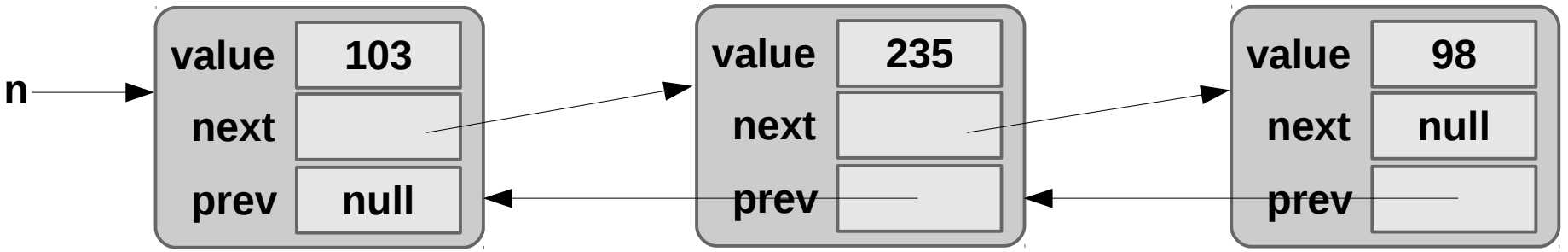
- **Arrays**:
  - Searching an element: constant cost.
  - Inserting a new element: linear cost (~*n*).
- **Single Linked Lists**:
  - Searching an element: linear cost (~n).
  - Inserting a new element: may be constant.

Doubly Linked Lists

- Their **elements** (nodes) may be (and are likely to be) **spread** over the memory.
- **Nodes** are **connected** to others through **two pointers**:
  - One pointer to the next element.
  - One pointer to the previous element.
- Have **varying sizes**.
- Each **position** is referenced through a **pointer**.

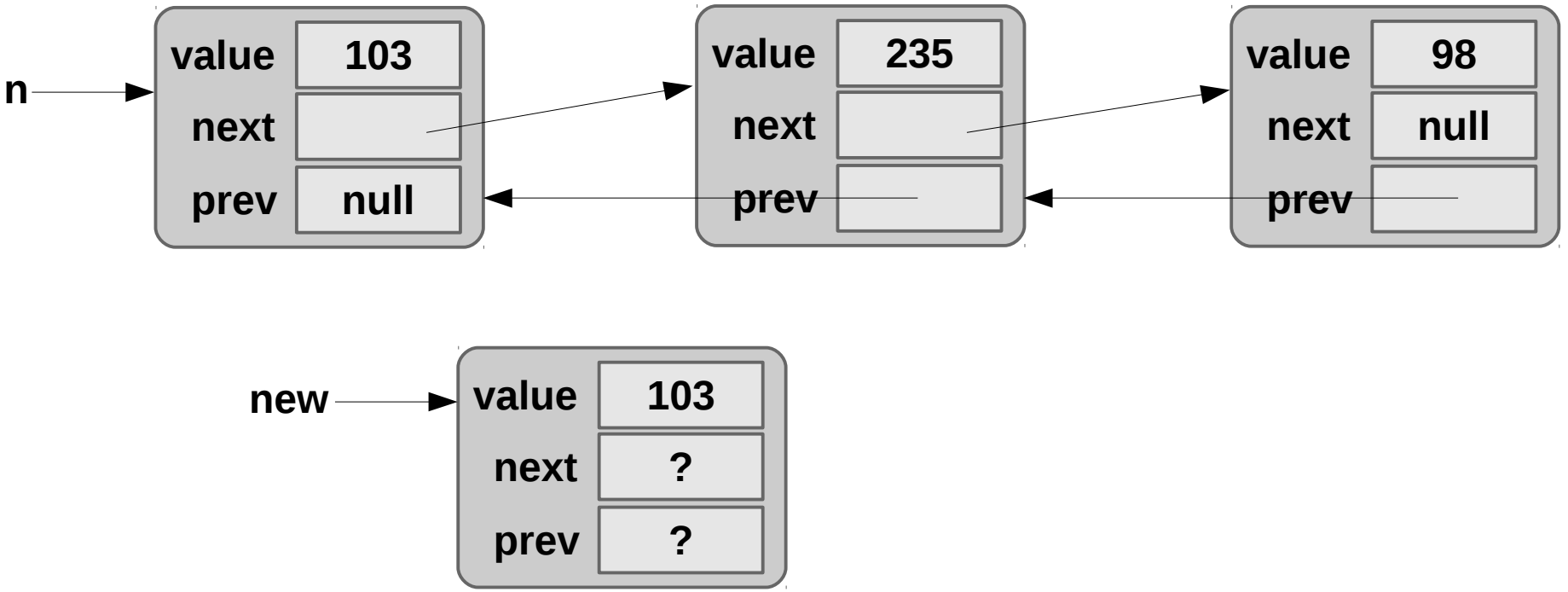
The Costs of Doubly Linked Lists

- Inserting a new element?
  - Once you have a pointer to the **current element**, the insertion **after** is **constant**.



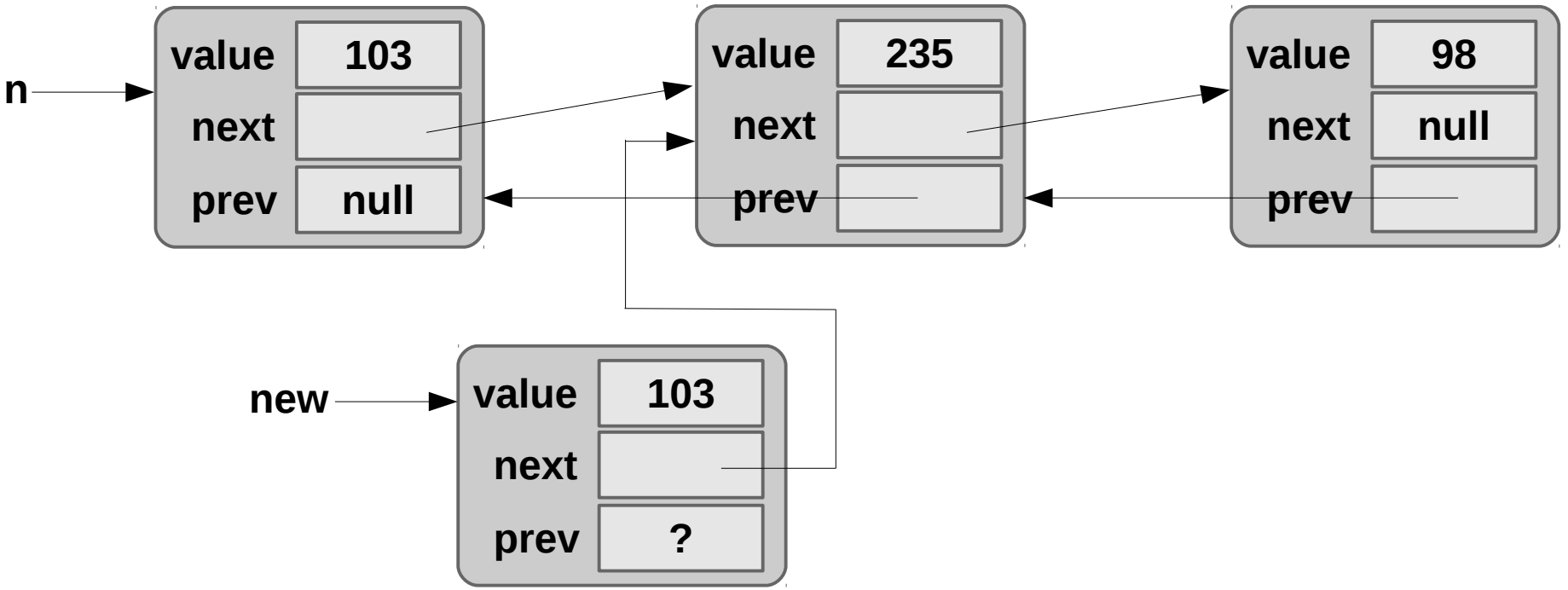
The Costs of Doubly Linked Lists

- Inserting a new element?
  - Once you have a pointer to the **current element**, the insertion **after** is **constant**.



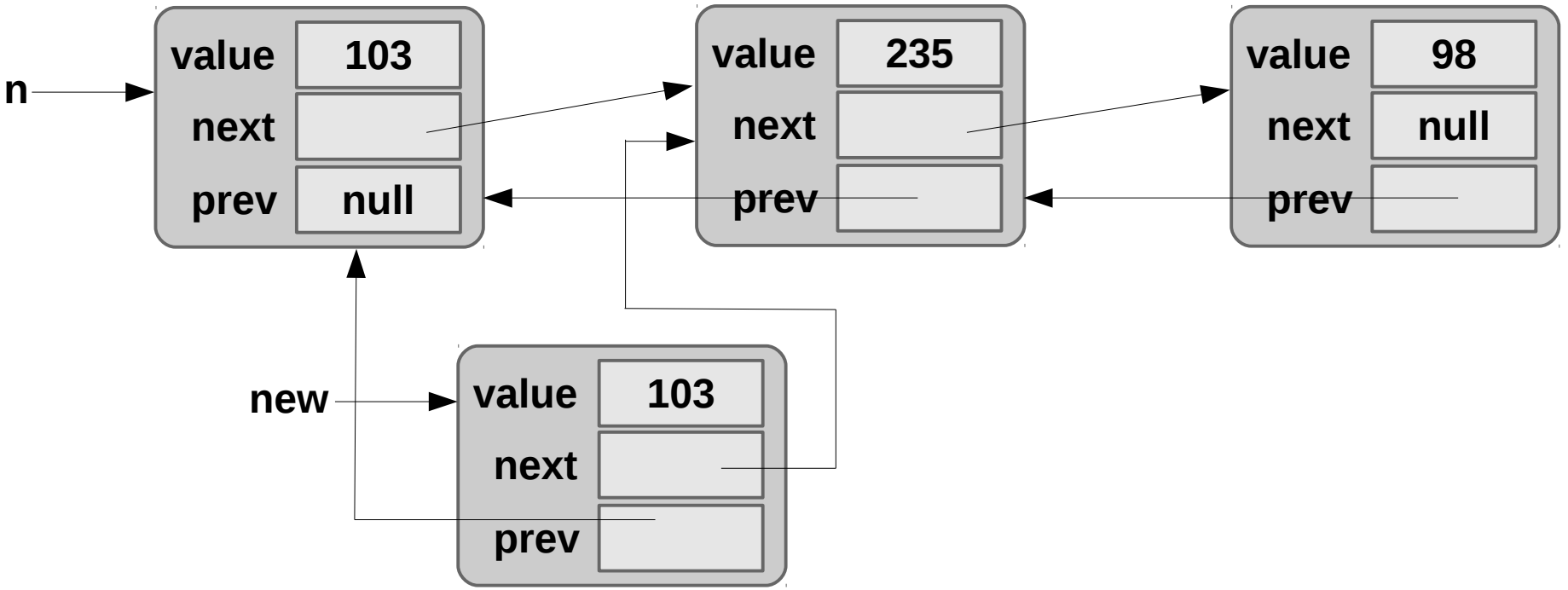
The Costs of Doubly Linked Lists

- Inserting a new element?
  - Once you have a pointer to the **current element**, the insertion **after** is **constant**.



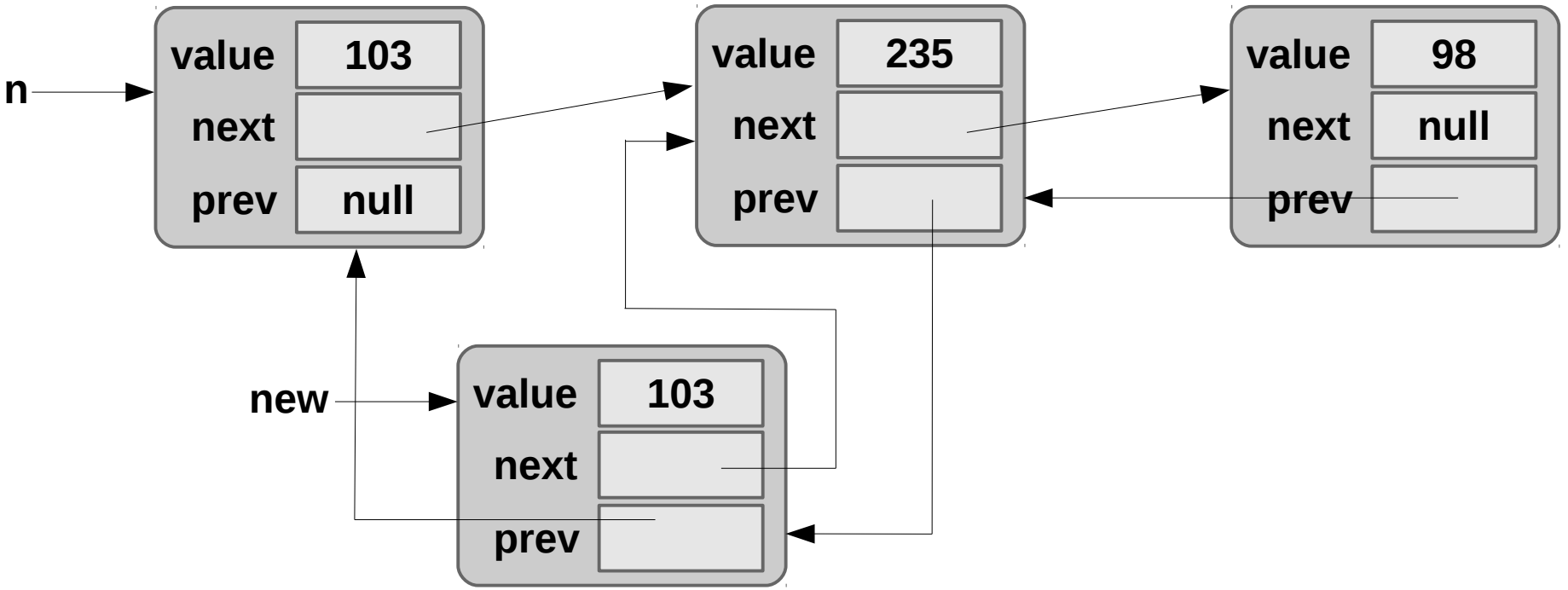
The Costs of Doubly Linked Lists

- Inserting a new element?
  - Once you have a pointer to the **current element**, the insertion **after** is **constant**.



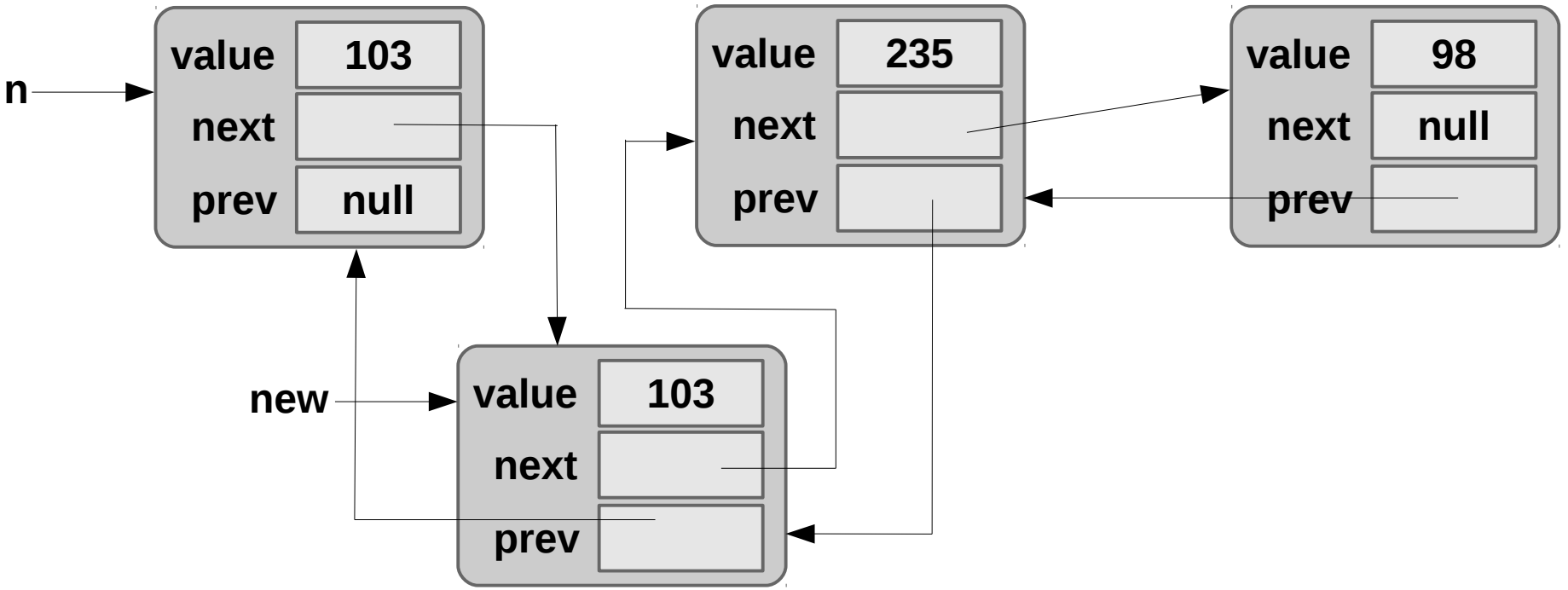
The Costs of Doubly Linked Lists

- Inserting a new element?
  - Once you have a pointer to the **current element**, the insertion **after** is **constant**.



The Costs of Doubly Linked Lists

- Inserting a new element?
  - Once you have a pointer to the **current element**, the insertion **after** is **constant**.



A Doubly Linked List Implementation

C code excerpt:

List node structure.

```
struct Node
{
    int value;
    struct Node* next;
    struct Node* prev;
};
...
```

Pointer to a node.

```
struct Node* n1;
```

Initialization of the 1st. node.

```
n1 = (struct Node*) malloc (sizeof(struct Node));
n1->value = -1;
n1->prev = NULL;
n1->next = NULL;
```

Insert element after node n1.

```
InsertAfter(n1, -2);
```

InsertAfter(...)

C code excerpt:

```
void InsertAfter(struct Node* n, int val)
{
    struct Node* new;

    new = (struct Node*) malloc (sizeof(struct Node));
    new->value = val;
    new->prev = n;
    new->next = n->next;
    n->next->prev = new;
    n->next = new;
}
```

This function **cannot** insert nodes in a **empty list** or **after the last** list ement. How the code could be **changed** in order to **handle those situations**?

List Descriptor

- The descriptor contains a **pointer** to the **head,** a **pointer** to the **tail,** and the number of **elements** of the **doubly linked list**:

C code excerpt:

```
struct List
{
    struct Node* head;
    struct Node* tail;
    int size;
};
```

It can also contain important information about the list, such as the length.

```
struct Node
{
    int value;
    struct Node* prev;
    struct Node* next;
};
```

Application Example: Big Integers

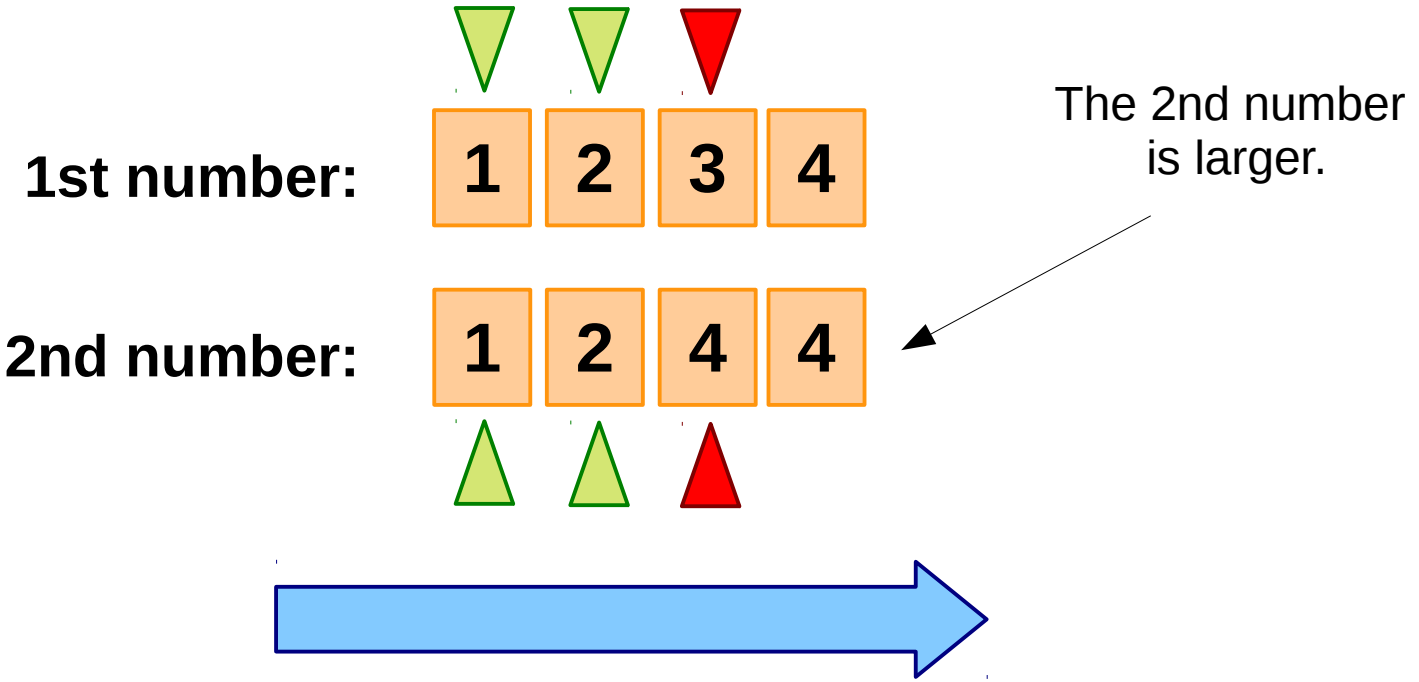
- Suppose you have to implement a system capable to store very **large positive integers**, composed of **hundreds** or **thousands** of **digits**.
- Consider that this system must be able to **sum** two such numbers.
- Consider also that the system must **check** which, of two numbers, is the **larger**.

Application Example: Big Integers

- **How does the check for the larger number works?**
  - The number with more digits is the larger.
  - If the number of digits is equal:
    - We must compare the corresponding digits of each number, from left to right, until they are different.
    - The number with the larger digit is the larger number.

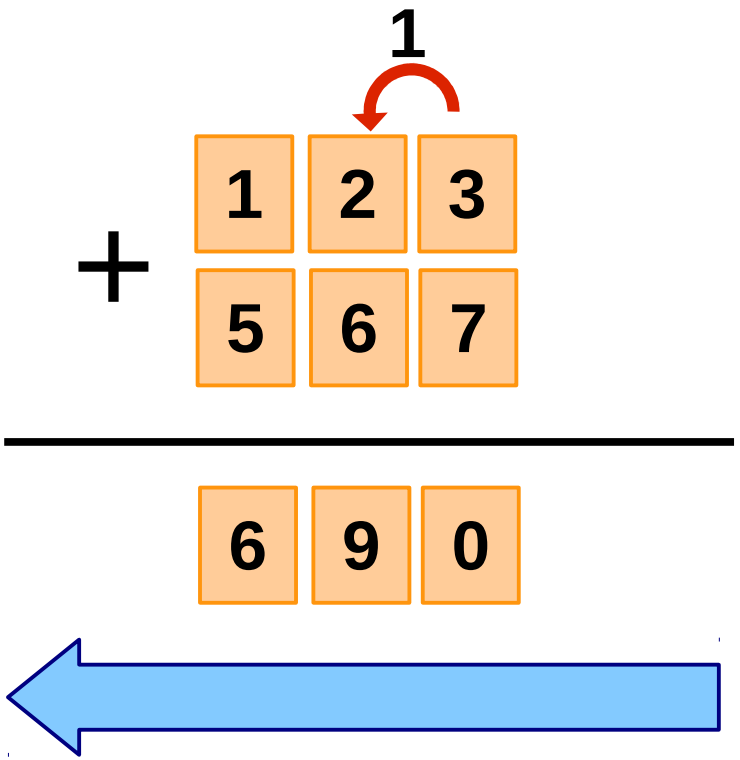
Application Example: Big Integers

- **How does the check for the larger number works?**
  - **Example**:



Application Example: Big Integers

- **How does the sum works?**
  - Consider the following expression: 123 + 456



A Doubly Linked List Implementation of Big Integers

- The number of digits may be large:
  - **Digits** will be represented by **nodes** in a linked list.
- We have to loop over the list from both directions:
  - The list will be **doubly linked**.
- Each loop must start at one end of the list:
  - There will be a list descriptor with pointers to the **head** and **tail** of the list.

A Doubly Linked List Implementation of Big Integers

- We start by implementing the list descriptor and node:

C code excerpt:

```
struct List
{
    int size;
    struct Node* head;
    struct Node* tail;
};
```

```
struct Node
{
    int value;
    struct Node* prev;
    struct Node* next;
};
...
```

A Doubly Linked List Implementation of Big Integers

- We need a function that inserts digits into the list

C code excerpt:

```
void Prepend(struct List* l, int val)
{
    struct Node* new;

    if (l->head == NULL)
    {
        l->head = (struct Node*) malloc (sizeof(struct Node));
        l->tail = l->head;
        l->head->value = val;
        l->head->prev = NULL;
        l->head->next = NULL;
    }
    else...
}
...
```

A Doubly Linked List Implementation of Big Integers

- We need a function that inserts digits into the list

C code excerpt:

```
void Prepend(struct List* l, int val)
{
    ...
    else
    {
        new = (struct Node*) malloc (sizeof(struct Node));
        new->value = val;
        new->prev = NULL;
        new->next = l->head;
        l->head->prev = new;
        l->head = new;
    }

    l->size++;
}
...
```

A Doubly Linked List Implementation of Big Integers

- We need code that compares two integers:

C code excerpt:

```
struct List* l1;
struct List* l2;

...

if (l1->size > l2->size)
    printf("l1 is bigger than l2.\n");
else
    if (l1->size < l2->size)
        printf("l2 is bigger than l1.\n");
    else

    ...
```

A Doubly Linked List Implementation of Big Integers

- We need code that compares two integers:

C code excerpt:

```
else {
    struct Node* n1 = l1->tail;
    struct Node* n2 = l2->tail;

    while ((n1 != NULL) && (n1->value == n2->value)) {
        n1 = n1->prev;
        n2 = n2->prev;
        printf("[n1: %p]  -  [n2: %p]\n", n1, n2);
    }

    if (n1 == NULL)
        printf("l1 and l2 are equal.\n");
    else
        if (n1->value > n2->value)
            printf("l1 is bigger than l2.\n");
        else
            printf("l2 is bigger than l1.\n");
}
```

Short circuit!!!

A	B	A && B
F	F	F
F	V	F
V	F	F
V	V	V

A Doubly Linked List Implementation of Big Integers

- We need code that sums two integers:

It is trivial and left as an exercise :)

Discussion

- **Considering the doubly linked lists**:
  - Which types of data structures would benefit from doubly linked lists?
  - Application examples.

Think about it !!!