



estruturas de **DADOS**

BY RAFA

estruturas de **DADOS**

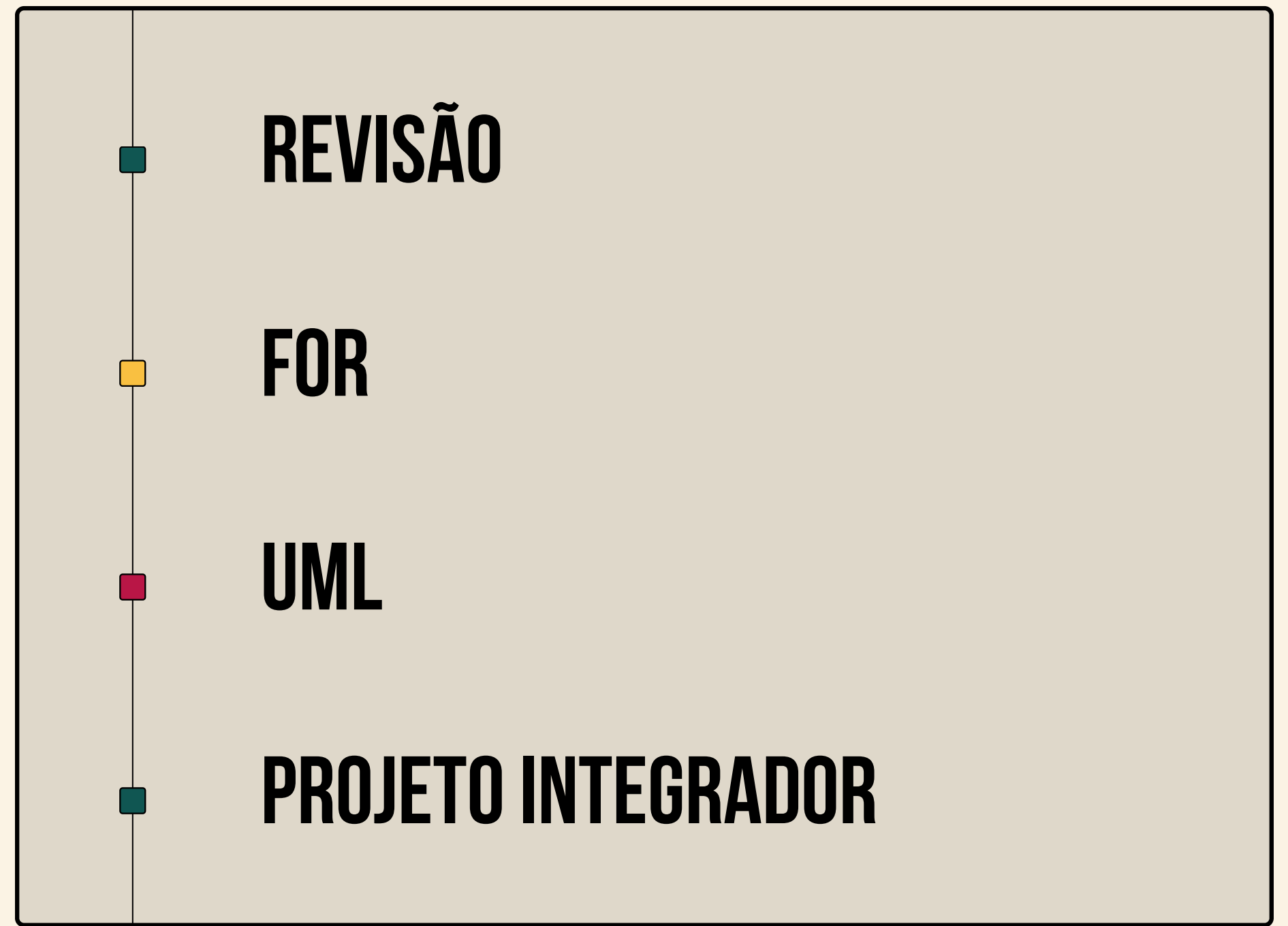
REVISÃO DE POO

ESTRUTURAS DE DADOS


introdução à
ORIENTAÇÃO
a **OBJETOS #6**

BY RAFA

introdução à **ORIENTAÇÃO** *a* **OBJETOS** **#6**



```
for
```

	for	for
<input type="checkbox"/>	for	For Loop
<input type="checkbox"/>	forawaitof	For-Await-Of Loop
<input type="checkbox"/>	foreach =>	For-Each Loop using =>
<input type="checkbox"/>	forin	For-In Loop
<input type="checkbox"/>	forof	For-Of Loop

FOR LOOP

Executa uma iteração com **início** e **término** determinados.

Não precisa estar ligada a um array.

O iterador pode ser de **qualquer** tipo.

```
let array:Array<string> = ["a","b","c","d","e"];

for (let i:number = 0; i < array.length; i++) {
  console.log(array[i]); // passa por todas as posições
}

for (let i:number = 1; i < 4; i++) {
  console.log(i); // imprime 1, 2 e 3
}

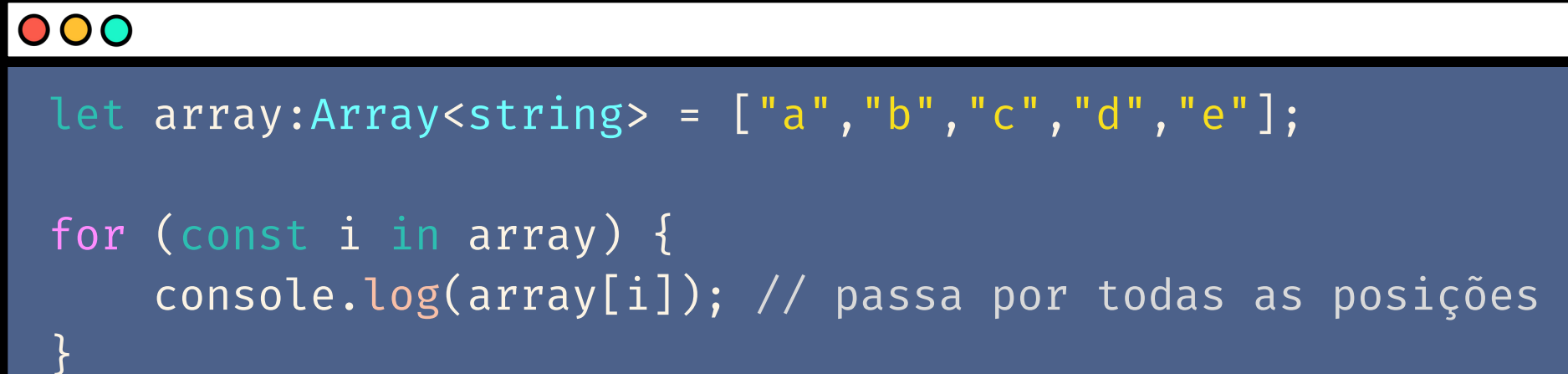
for (let i:string = "a"; i!= "abababa"; i+="ba"){
  console.log(i); // imprime a, aba, ababa e abababa
}
```

FOR IN

Executa uma iteração **SEMPRE** com **base** em um **array**

Sempre utiliza um iterador do tipo **string** como **contador**. Ex.: "0", "1", "2", "3",...

Utiliza uma **const** como base pois **recria** a variável **em cada iteração** e não permite a sua alteração.



```
let array:Array<string> = ["a","b","c","d","e"];

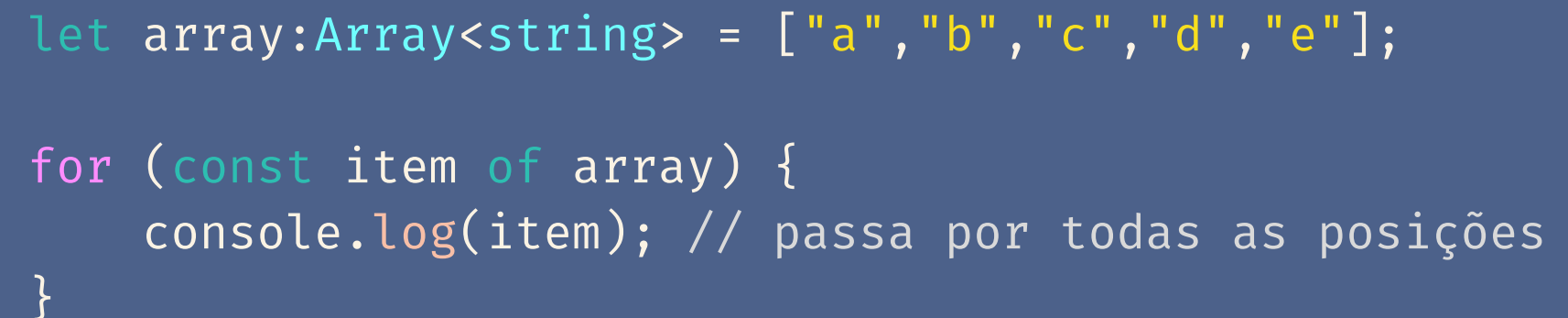
for (const i in array) {
  console.log(array[i]); // passa por todas as posições
}
```

FOR OF

Executa uma iteração **SEMPRE** com **base** em um **array**

Não possui um iterador. **Sempre** processa diretamente os **itens** do array.

Utiliza uma **const** como base pois **recria** a variável **em cada iteração** e não permite a sua alteração.



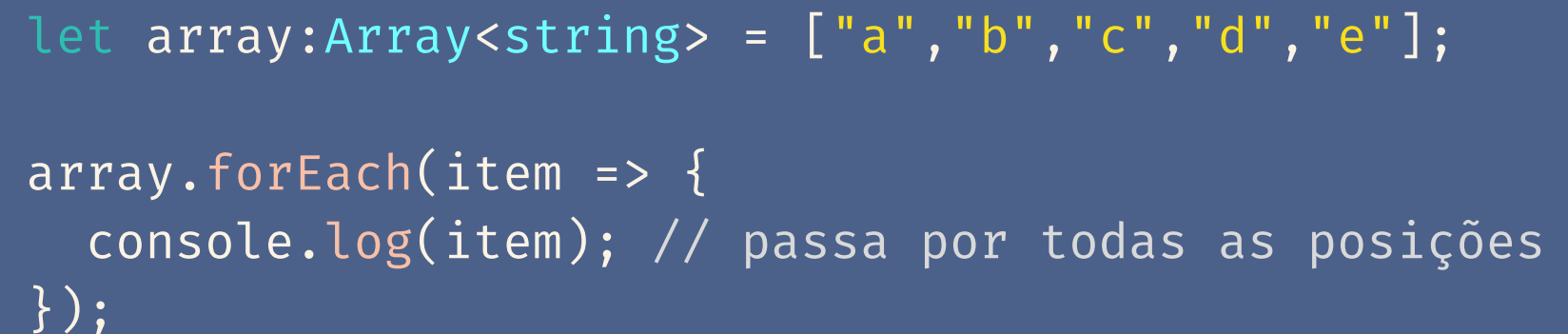
```
let array:Array<string> = ["a","b","c","d","e"];

for (const item of array) {
  console.log(item); // passa por todas as posições
}
```


FOR EACH

Executa uma iteração **SEMPRE** com **base** em um **array**

Não possui um iterador. **Sempre** processa diretamente os **itens** do array.

A code editor window with a dark blue background and a white title bar containing three colored circles (red, yellow, green). The code is written in Kotlin and demonstrates the use of the `forEach` method on an array of strings.

```
let array:Array<string> = ["a","b","c","d","e"];

array.forEach(item => {
    console.log(item); // passa por todas as posições
});
```

INFORMAÇÃO

EXTRA

ENUMERADORES

ENUMERADORES

Numérico e implícito

```
export enum Level {  
  BLUE,    \\ 0  
  YELLOW,  \\ 1  
  ORANGE,  \\ 2  
  RED      \\ 3  
}
```

Numérico e explícito
incremental

```
export enum Level {  
  BLUE = 1,  \\ 1  
  YELLOW,    \\ 2  
  ORANGE,    \\ 3  
  RED        \\ 4  
}
```

Numérico e explícito

```
export enum Level {  
  BLUE = 0,  
  YELLOW = 10,  
  ORANGE = 20,  
  RED = 30  
}
```

String

```
export enum Level {  
  BLUE = "Blue",  
  YELLOW = "Yellow",  
  ORANGE = "Orange",  
  RED = "Red"  
}
```

ENUMERADORES

```
export enum Level {  
  BLUE = 1,  
  YELLOW,  
  ORANGE,  
  RED  
}
```

```
export class Survivor {  
  protected level: Level;  
  
  getLevel(): Level {  
    return this.level;  
  }  
  
  levelUp(): void {  
    this.level++;  
  }  
}
```

O QUE SÃO ESTRUTURAS DE DADOS?



TIPOS DE DADOS

PRIMITIVOS



```
graph TD; A[PRIMITIVOS] --> B[NUMBER]; A --> C[STRING]; A --> D[BOOLEAN];
```

NUMBER

STRING

BOOLEAN

TIPOS DE DADOS



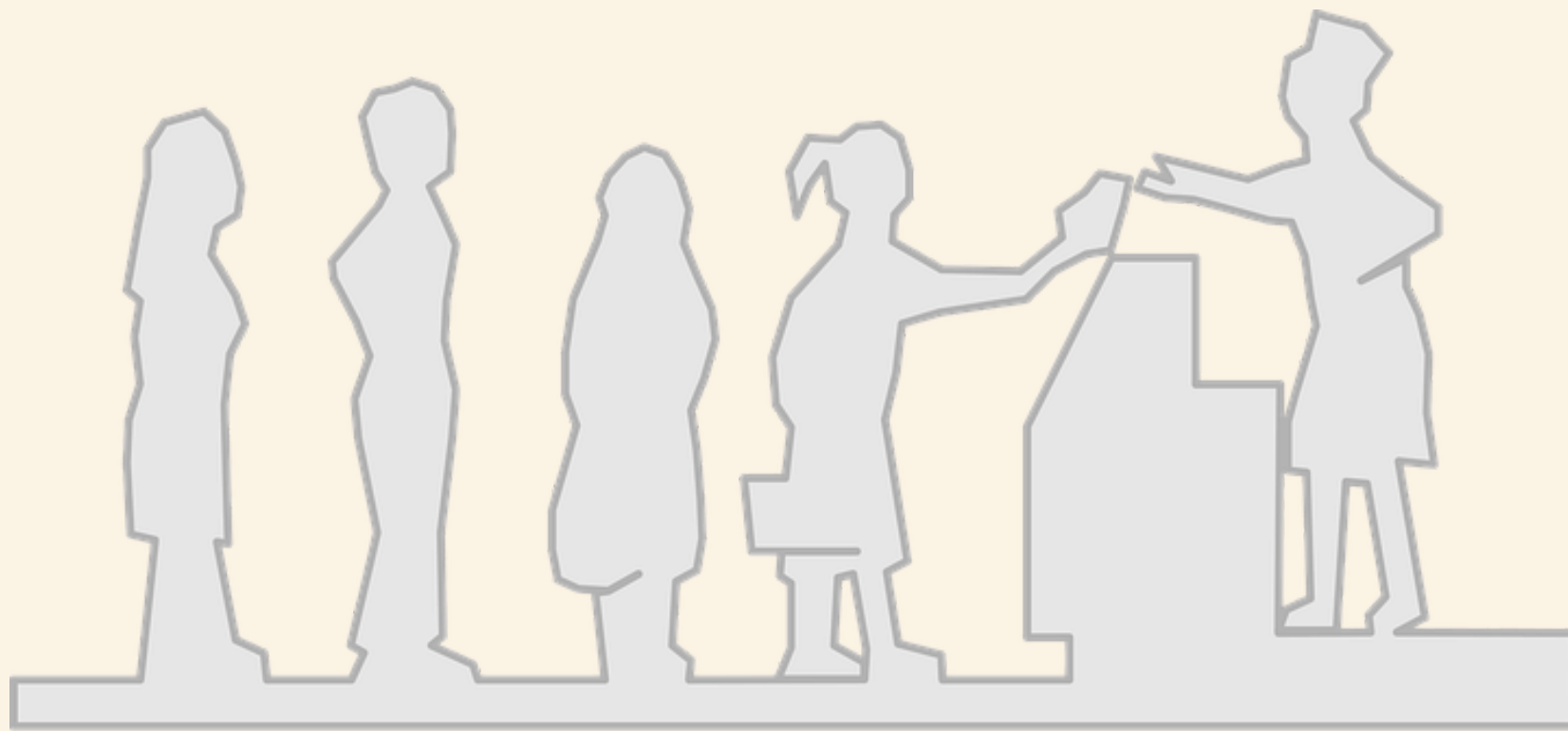
TIPOS DE DADOS



TIPOS DE DADOS



TIPOS DE DADOS



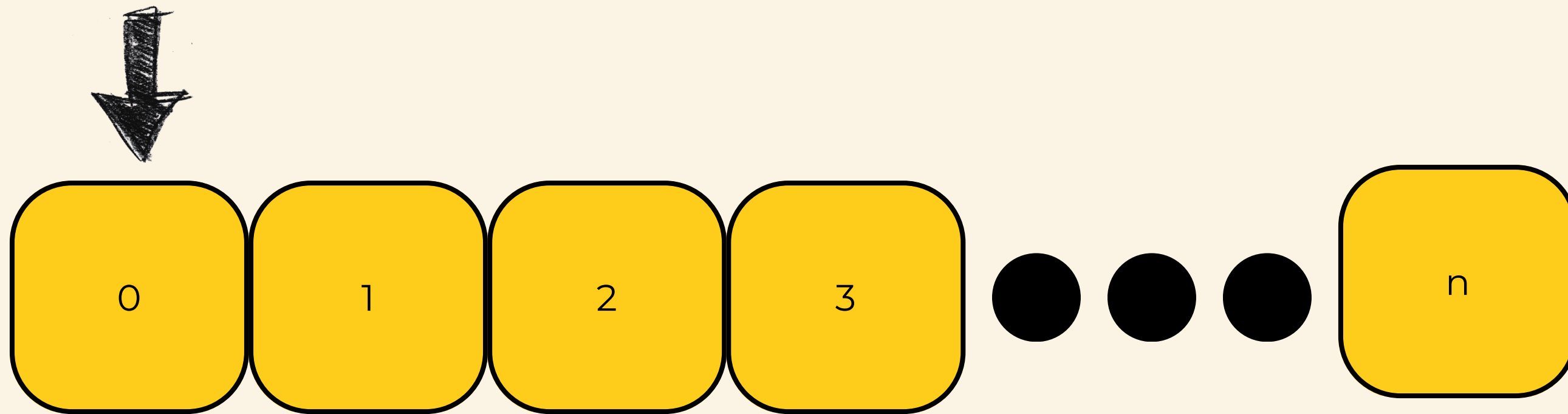
ESTRUTURAS DE DADOS



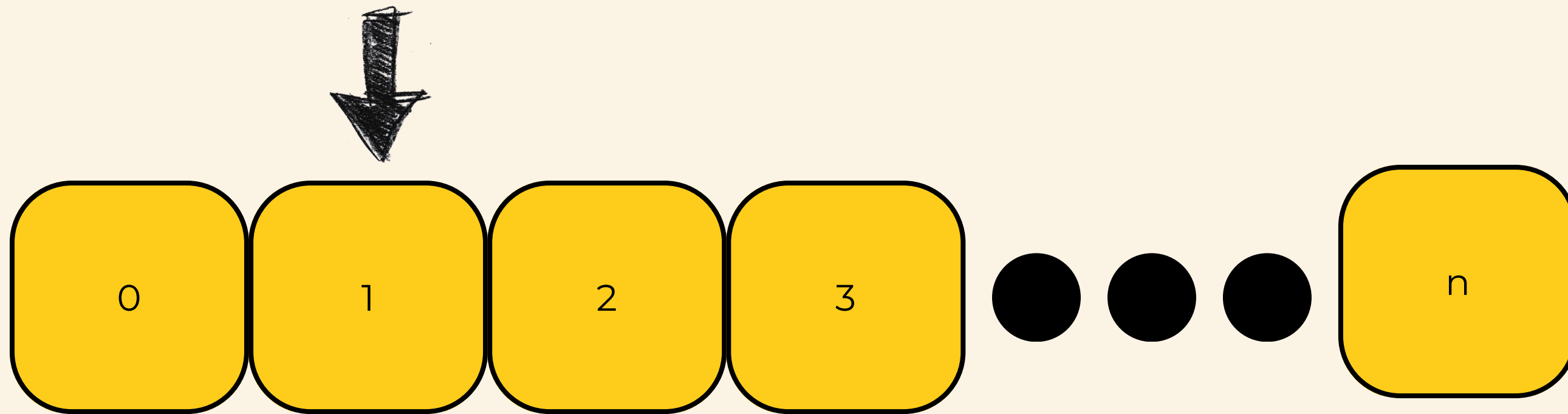
O que são?

Tipos não-primitivos de organização de dados para atender aos diferentes requisitos de processamento

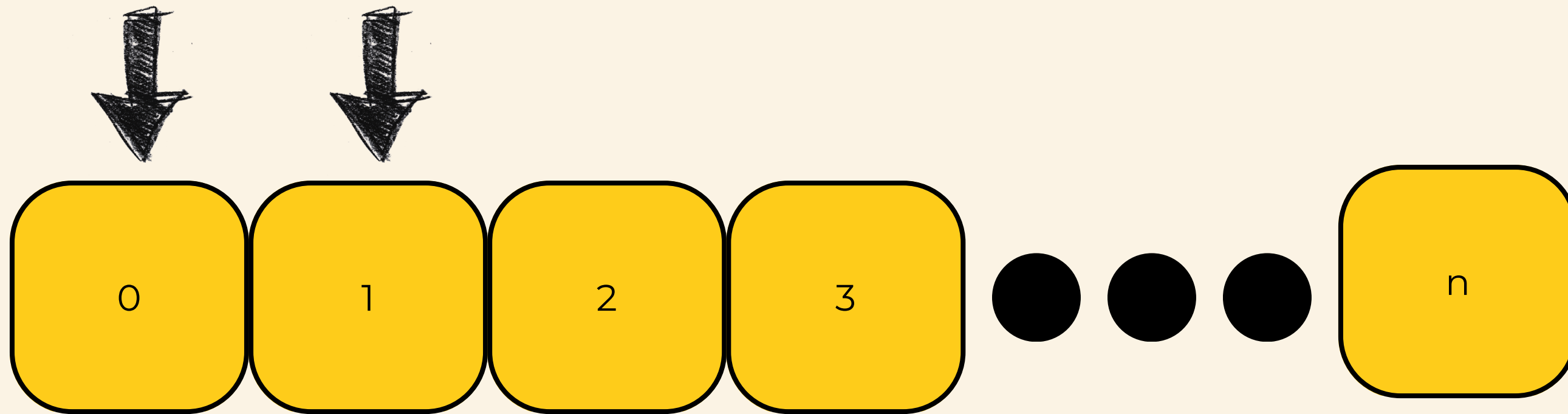
ARRAY



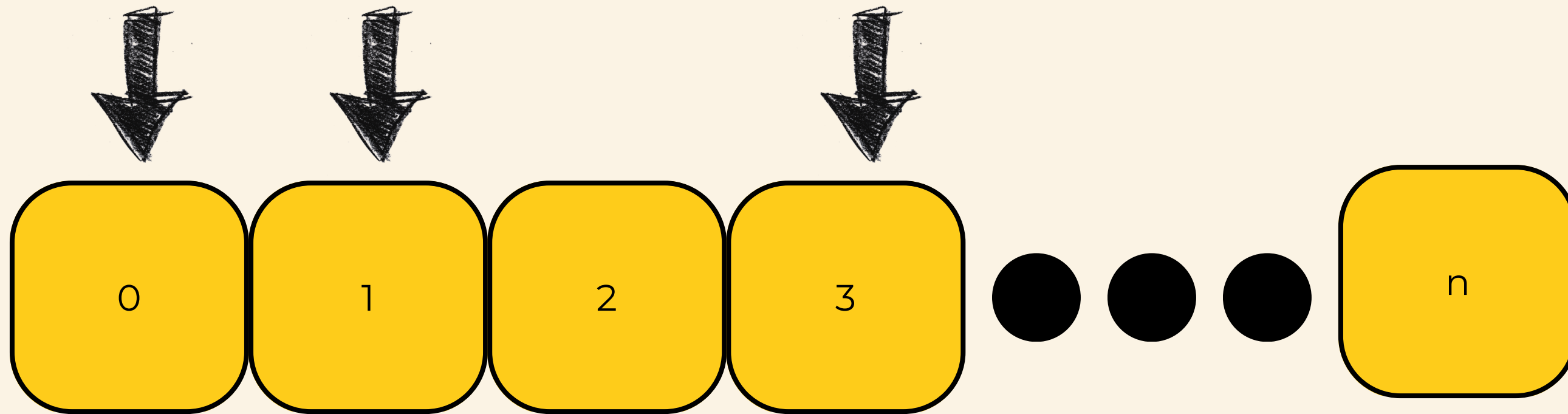
ARRAY



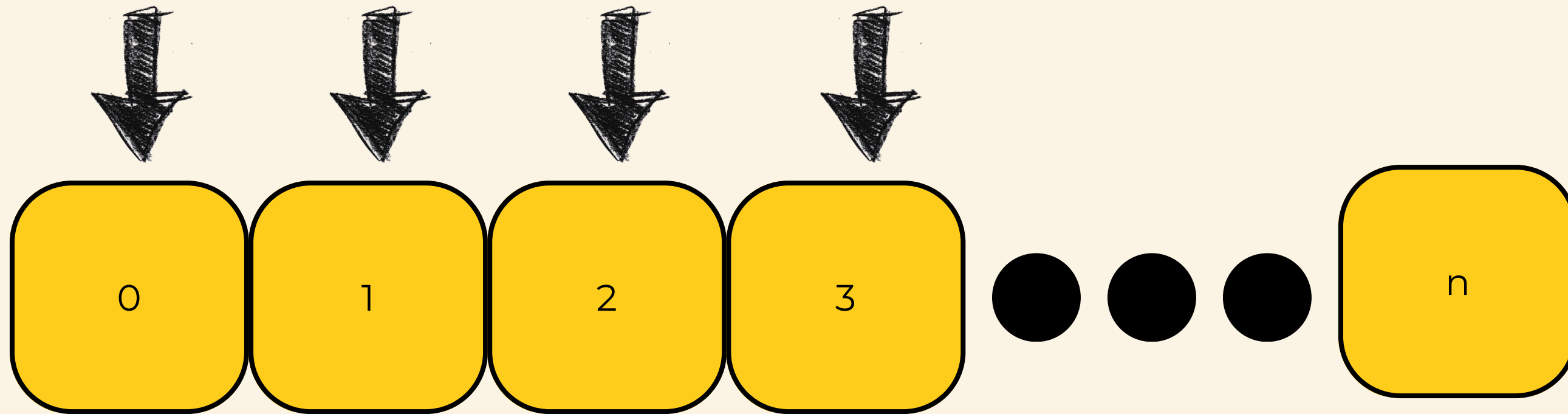
ARRAY



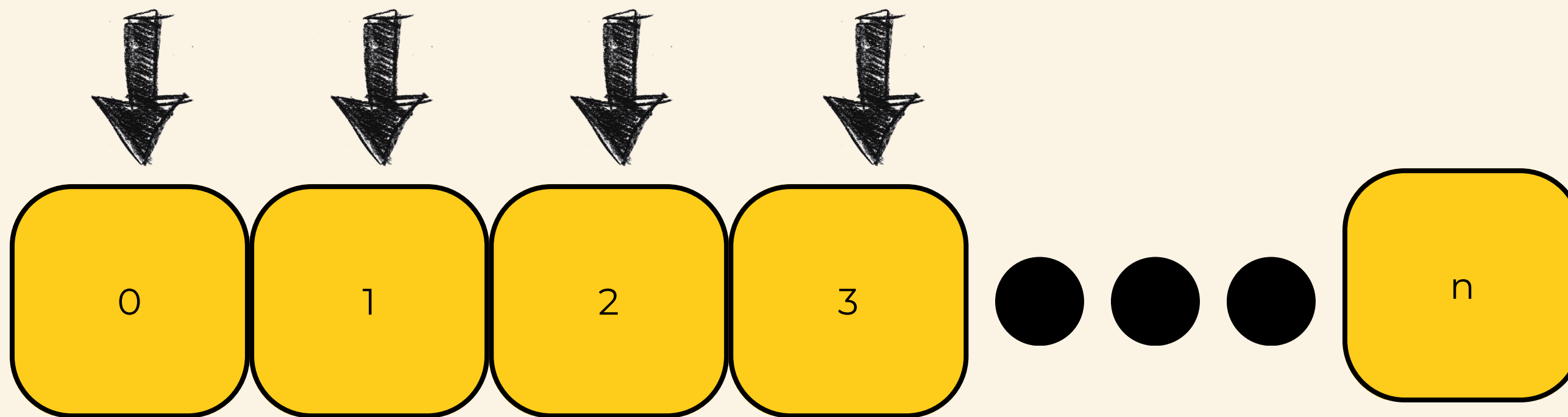
ARRAY



ARRAY



ARRAY



Posições bem definidas
Tamanho limitado (nem sempre)
Itens podem ser adicionados em qualquer posição

COLEÇÕES



LISTA



FILA



PILHA

COLEÇÕES



LISTA



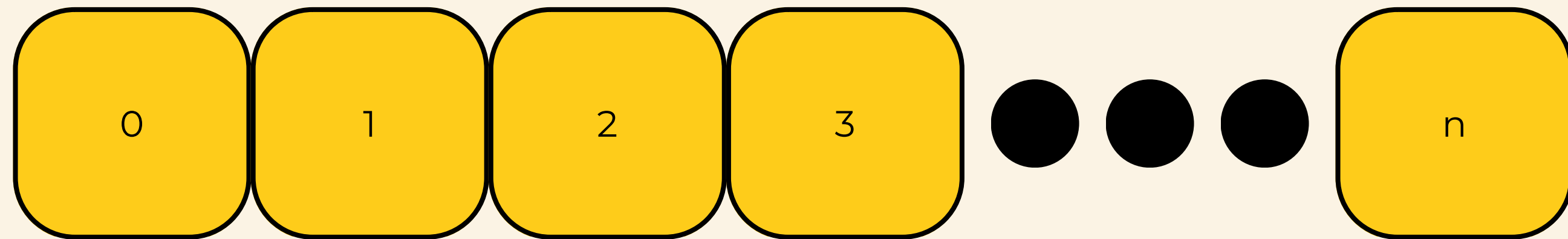
FILA



PILHA

LISTA

Array de "luxo"



Posições bem definidas

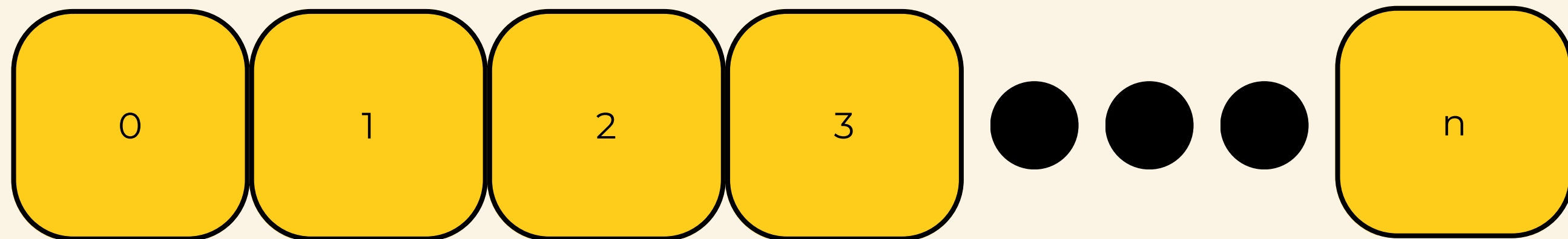
Tamanho se adapta à posição ocupada de maior índice

Itens podem ser adicionados em qualquer posição

Não possui ordem de saída definida

LISTA

Array de "luxo"



Posições bem definidas

Tamanho se adapta à posição ocupada de maior índice

Itens podem ser adicionados em qualquer posição

Não possui ordem de saída definida

```
export interface IList {  
  add(item: string): void;  
  remove(index: number): void;  
  get(index: number): void;  
  set(index: number, item: string): void;  
  contains(item: string): boolean;  
  size(): number;  
  isEmpty(): boolean;  
}
```

COLEÇÕES



LISTA



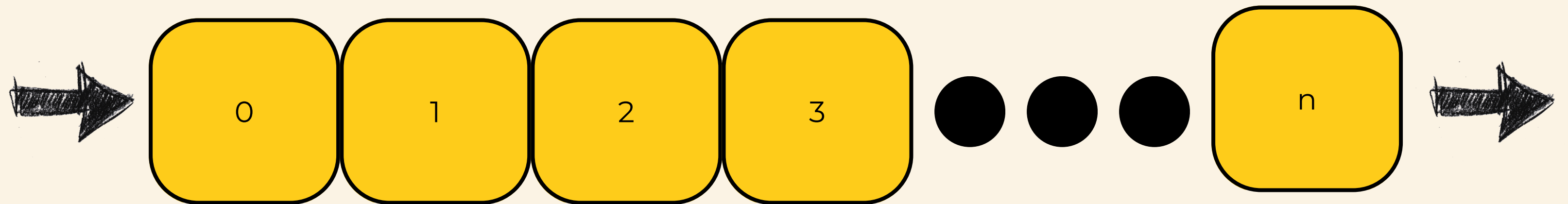
FILA



PILHA

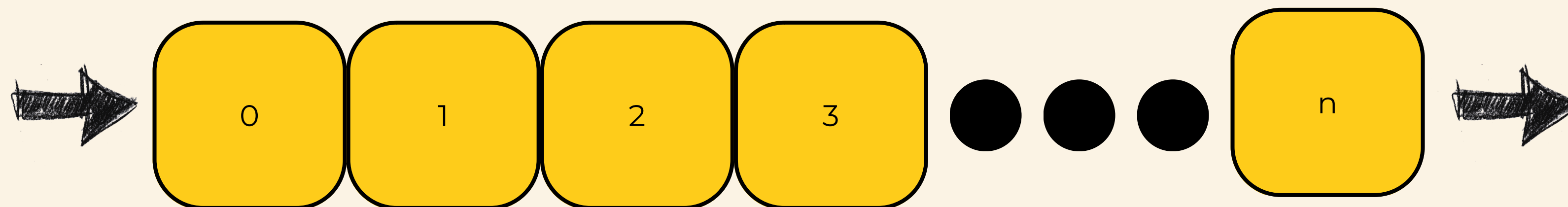
FILA

Primeiro a Chegar Primeiro a Sair (PCPS)
First In, First Out (FIFO)



FILA

Primeiro a Chegar Primeiro a Sair (PCPS)
First In, First Out (FIFO)



Posições bem definidas

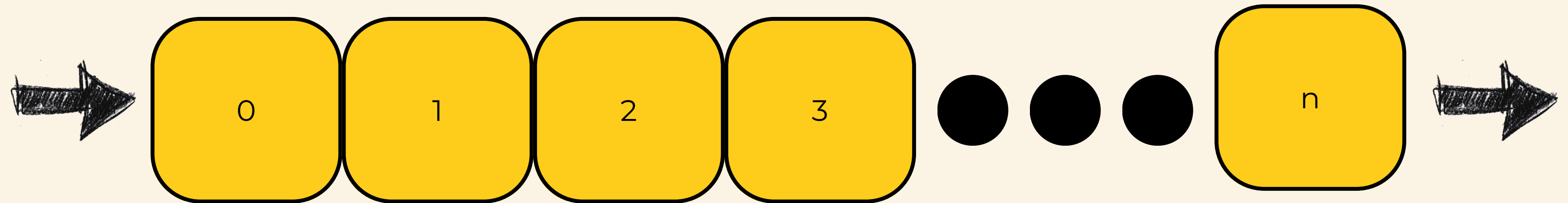
Tamanho se adapta à posição ocupada de maior índice

Itens podem ser adicionados apenas no menor índice (zero)

Saída deve ser feita pelo maior índice ocupado

FILA

Primeiro a Chegar Primeiro a Sair (PCPS)
First In, First Out (FIFO)



Posições bem definidas
Tamanho se adapta à posição ocupada de maior índice
Itens podem ser adicionados apenas no menor índice (zero)
Saída deve ser feita pelo maior índice ocupado

```
export interface IQueue {  
  enqueue(item: string): void;  
  dequeue(): string;  
  size(): number;  
  isFull(): boolean;  
}
```

COLEÇÕES



LISTA



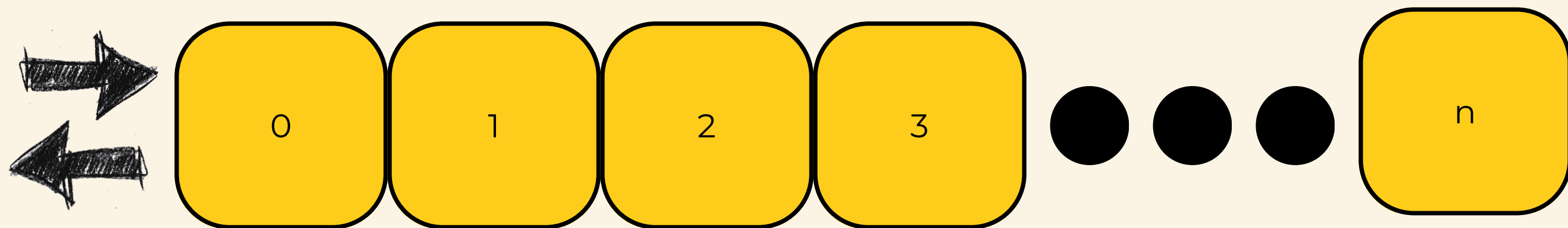
FILA



PILHA

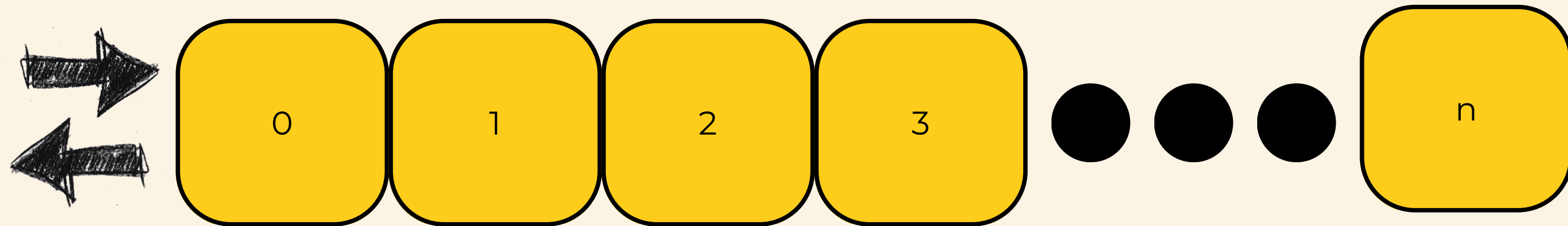
PILHA

Primeiro a Chegar Último a Sair (PCUS)
First In, Last Out (FILO)



PILHA

Primeiro a Chegar Último a Sair (PCUS)
First In, Last Out (FILO)



Posições bem definidas

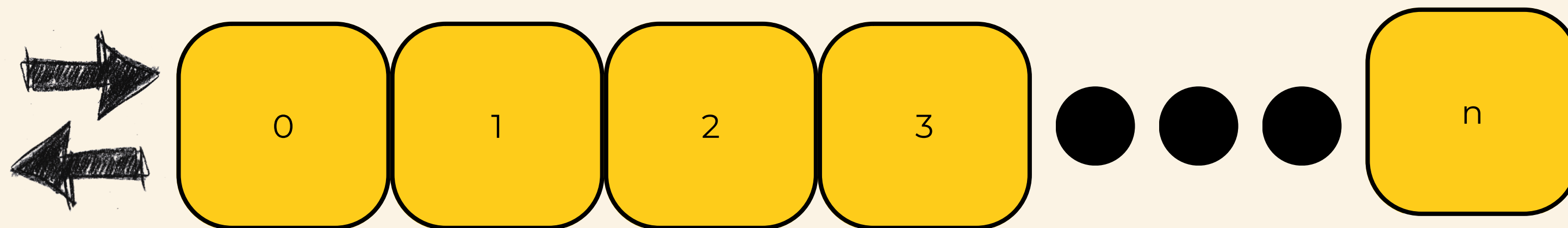
Tamanho se adapta à posição ocupada de maior índice

Itens podem ser adicionados apenas no menor índice (zero)

Saída deve ser feita pelo menor índice ocupado

PILHA

Primeiro a Chegar Último a Sair (PCUS)
First In, Last Out (FILO)



Posições bem definidas

Tamanho se adapta à posição ocupada de maior índice

Itens podem ser adicionados apenas no menor índice (zero)

Saída deve ser feita pelo menor índice ocupado

```
export interface IStack {  
  push(item: string): void;  
  pop(): string;  
  size(): number;  
  isFull(): boolean;  
}
```

DESAFIO

Implementar as interfaces de estruturas de dados apresentadas.

```
export interface IList {  
  add(item: string): void;  
  remove(): void;  
  get(index: number): void;  
  set(index: number, item: string): void;  
  contains(item: string): boolean;  
  size(): number;  
  isEmpty(): boolean;  
}
```

```
export interface IQueue {  
  enqueue(item: string): void;  
  dequeue(): string;  
  size(): number;  
  isFull(): boolean;  
}
```

```
export interface IStack {  
  push(item: string): void;  
  pop(): string;  
  size(): number;  
  isFull(): boolean;  
}
```



OBRIGADO!