

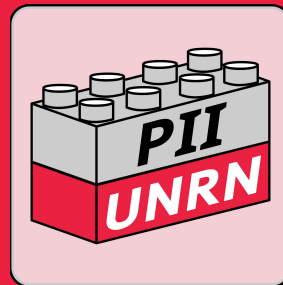
Estructuras de datos

UNRN

Universidad Nacional
de Río Negro

XX

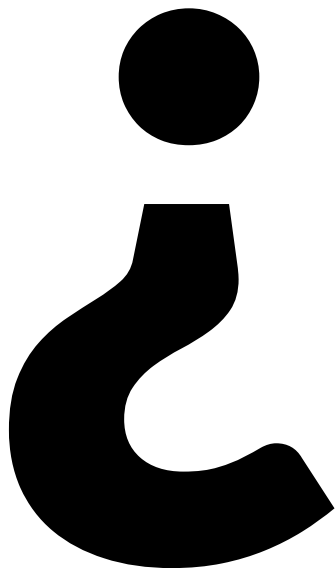
2024





¿Preguntas?

Formas de organizar la información



Por qué

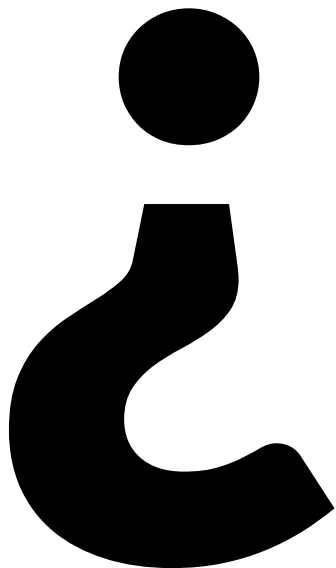


**Lo que vamos a ver es
para armar la caja de
herramientas**

**No
necesariamente
son Orientadas a
Objetos**

Usando complejidad como métrica, veremos los compromisos

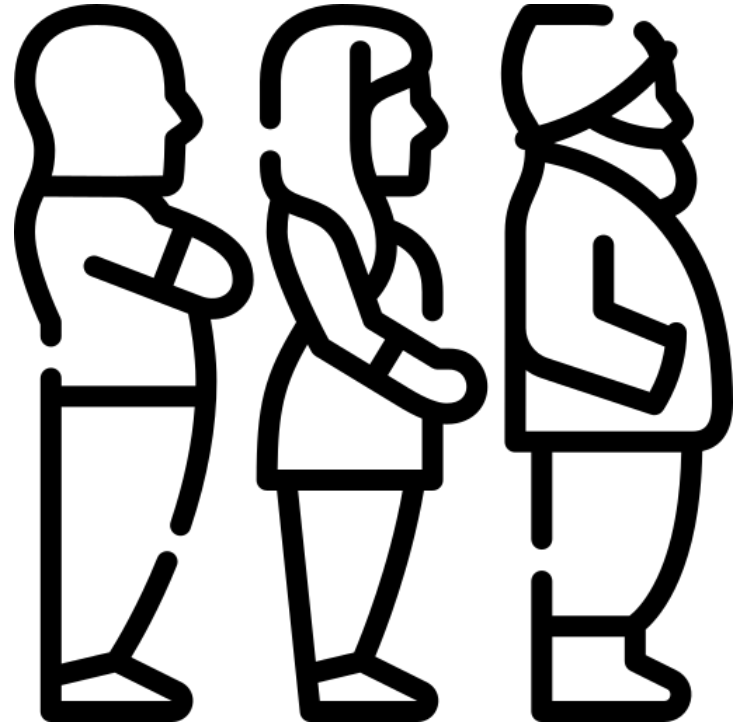
Tipos de datos abstractos [TDA]



Qué son



Por ejemplo
Una fila



Como la de un supermercado

fila



Entrada

The diagram illustrates a queue structure. It features a central horizontal container divided into four empty square cells. To the left of this container is a red arrow pointing right, labeled 'Entrada'. To the right of the container is another red arrow pointing right, labeled 'Salida'.

Salida

Tipo de dato **abstracto**

es un modelo teórico que define una estructura de datos y solo sus operaciones.

Sin indicaciones sobre su implementación

Operaciones

generales

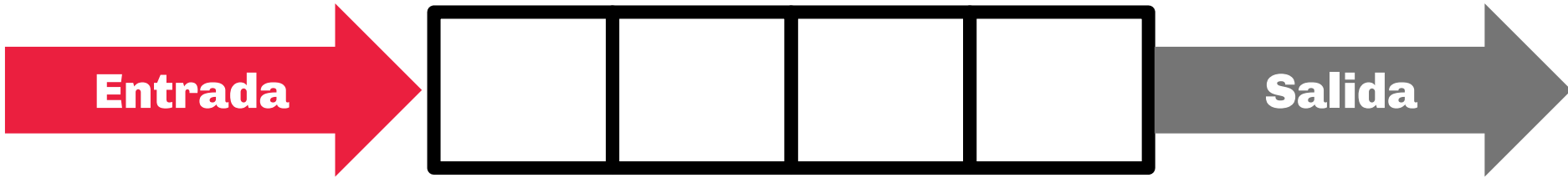
Creación

Como comienza la estructura.
Inicialización con valores o información
específica.

Inserción / adición

Como agregar nuevos elementos a la estructura

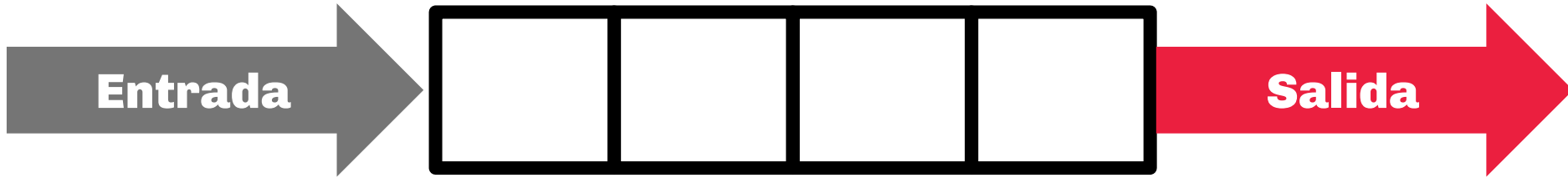
Inserción en una fila



Borrado / remoción

Sacar elementos presentes en la estructura.

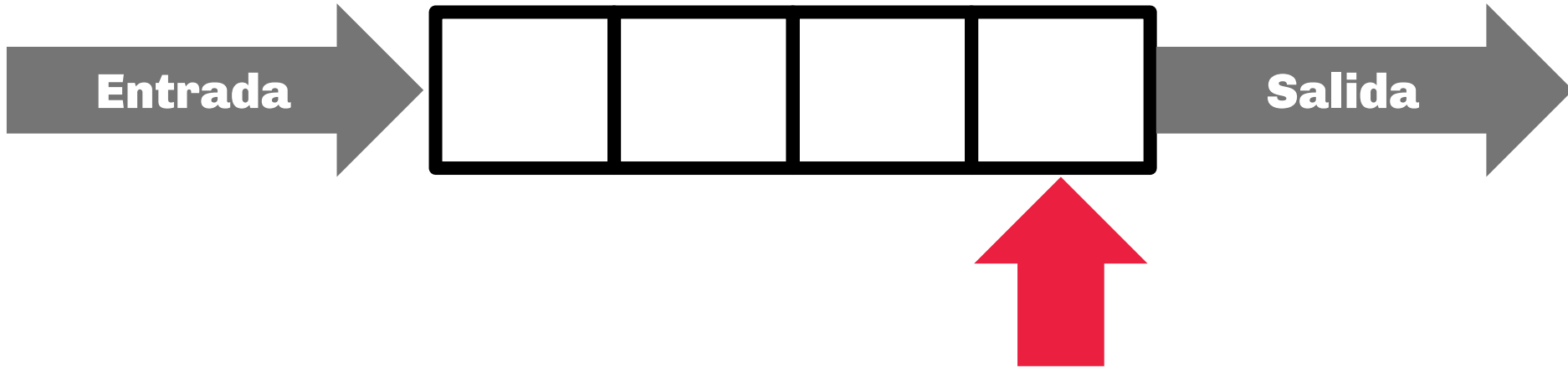
Remoción de una fila



Acceso / recuperación

Obtener algún valor contenido sin modificarlo de la estructura.

Recuperación de una fila



Modificación

Cambiar un valor contenido en la estructura.

Recorrido / iteración

Recorrer el contenido para completar alguna tarea sobre los mismos.

Búsqueda


Recorrer la estructura para recuperar un valor o saber si alguno en particular se encuentra presente.

Ordenamiento

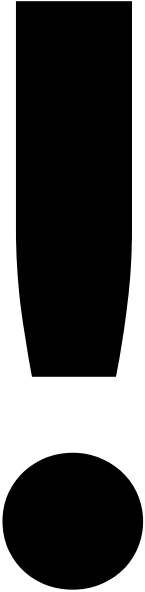
Acomodar los elementos de la estructura en un orden específico.

Tamaño / atributos

Obtener información sobre el tamaño y otras características de la estructura.



**No todas las
operaciones
generales tienen
sentido siempre**

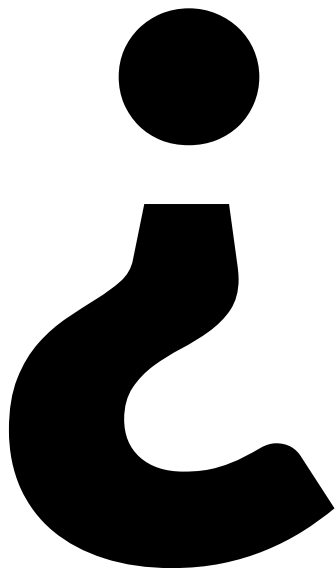




¿Preguntas?

arreglos

por ejemplo



**Qué
operaciones
definen a un
arreglo**

(las indispensables)



un ADT Arreglo

```
Arreglo(tamaño)
  obtener(posicion): elemento
  modificar(posicion, elemento)
  tamaño(): cantidad
  iterar(): iterador
```

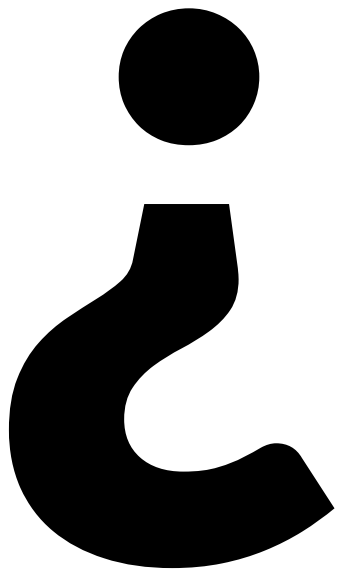
```
Arreglo(tamaño)
  obtener(posicion): elemento
  modificar(posicion, elemento)
  tamaño(): cantidad
  iterar(): iterador
```

Creación
Inserción / Adición
Borrado / remoción
Acceso / recuperación
Modificación
Recorrido / iteración
Búsqueda
Ordenamiento
Tamaño / atributos



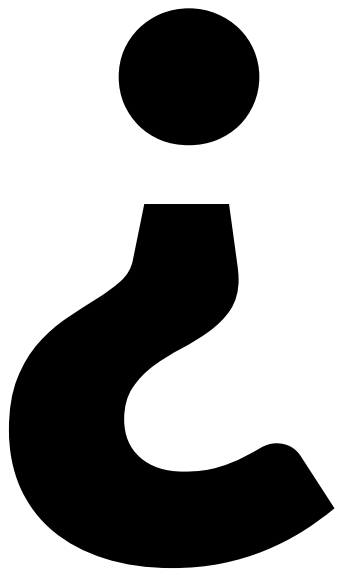
¿Preguntas?

El problema del almacenamiento de información



**Por qué es un
problema**





**Cuánto
'cuesta' cada
operación**



‘Cuesta’ como complejidad algorítmica

$O(f(n))$

El peor caso

$\Omega(f(n))$

El mejor caso



¿Preguntas?

Secuencias

Conjunto de valores ordenados

Arreglo

De tamaño fijo

| | | | | | | | | | |
|------|------|------|------|------|------|------|------|------|------|
| dato | dato | dato | dato | dato | dato | dato | dato | dato | dato |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

largo



| | | | | | | | | |
|------|------|------|------|------|------|------|------|------|
| dato | dato | dato | dato | dato | null | null | null | null |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Arreglo

- **Acceso: $O(1)$ / $\Omega(1)$**
- **Modificación: $O(1)$ / $\Omega(1)$**
- ***Inserción: $O(n)$ / $\Omega(n)$***
- ***Borrado: $O(n)$ / $\Omega(n)$***
- **Ordenamiento*: $O(n^2)$ / $\Omega(n)$**
- ***Ampliación: $O(n)$ / $\Omega(n)$***

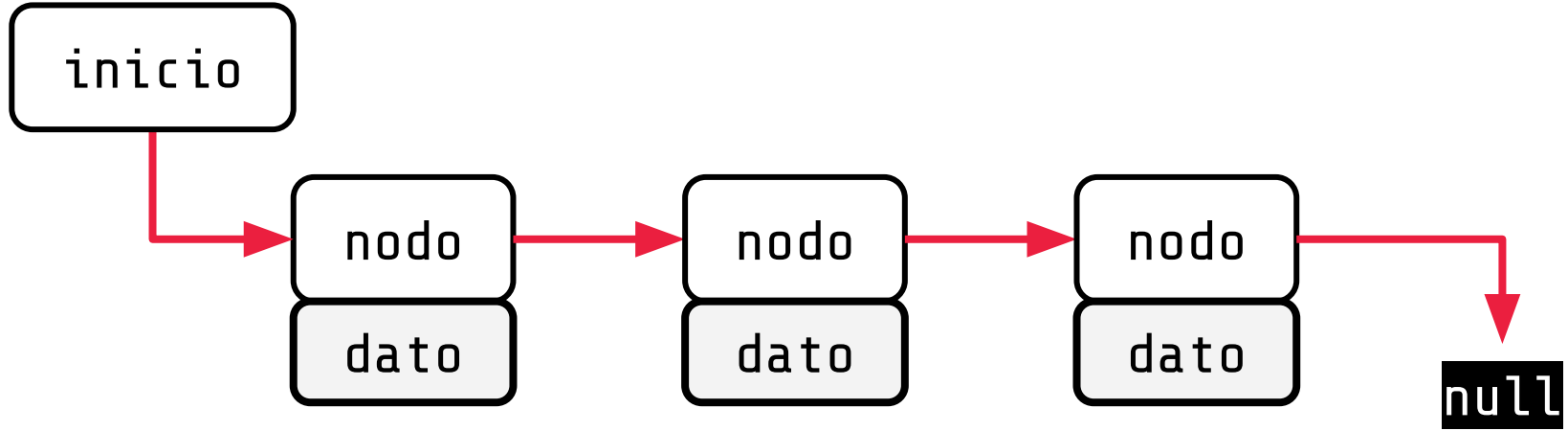
**Hasta acá todo lo de
siempre**



¿Preguntas?

Listas Enlazadas

Conjunto dinámico



Simple, dinámica con nodos

Ventajas

Inserción y eliminación eficientes

Cambiar de lugar muy eficiente

Tamaño dinámico

Desventajas

Mayor consumo de memoria

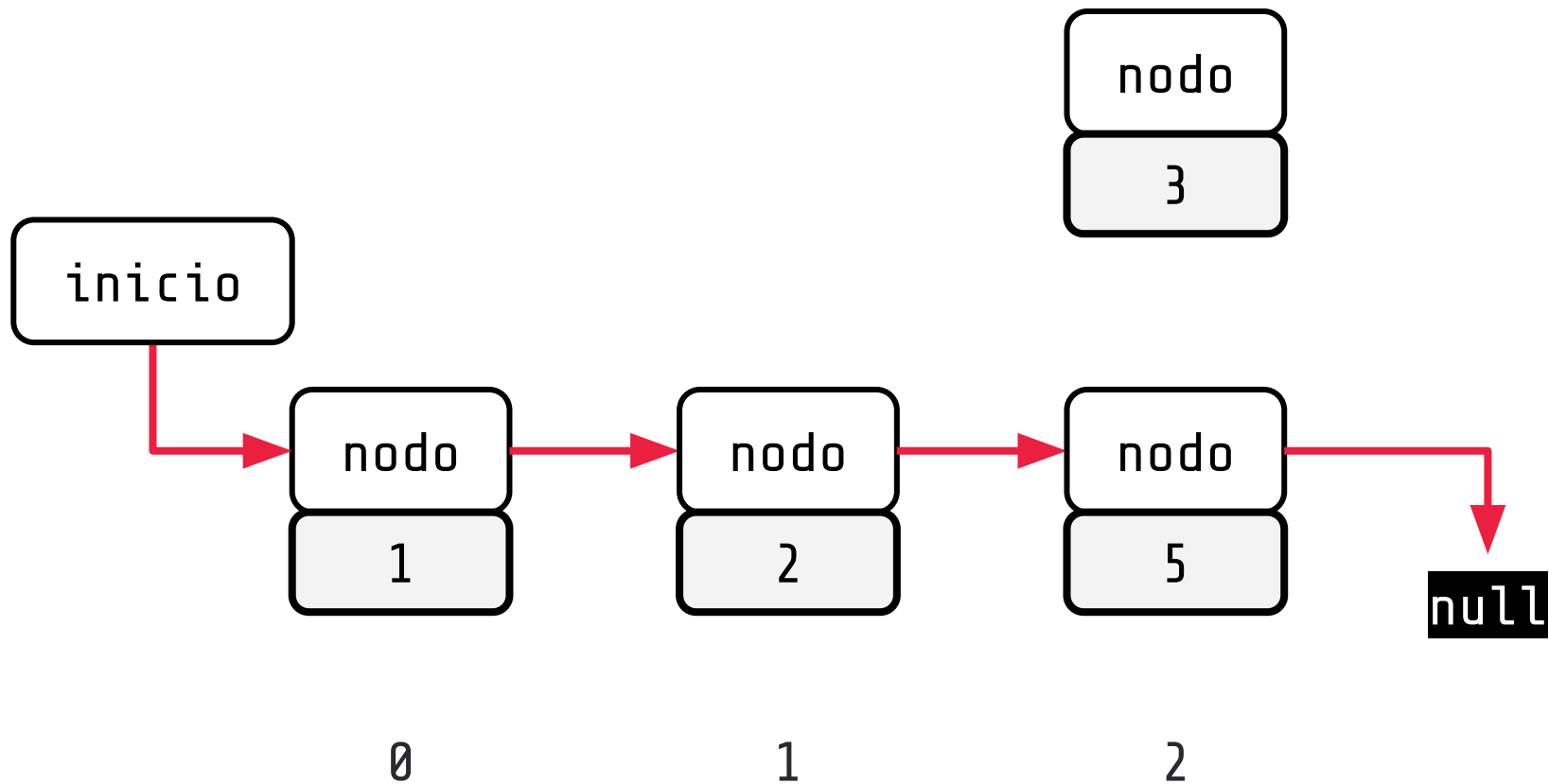
Acceso secuencial

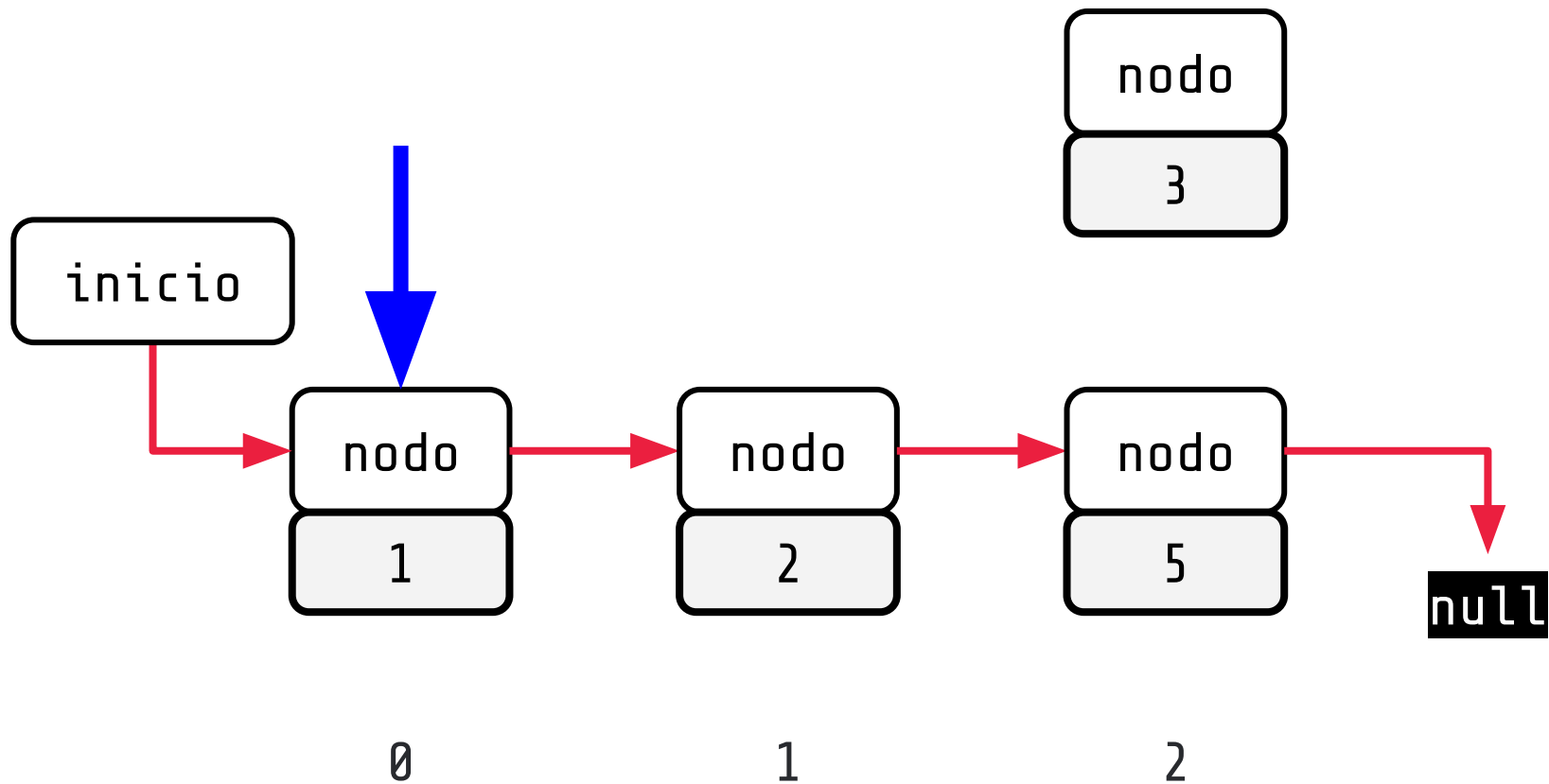
Complejidad en la implementación

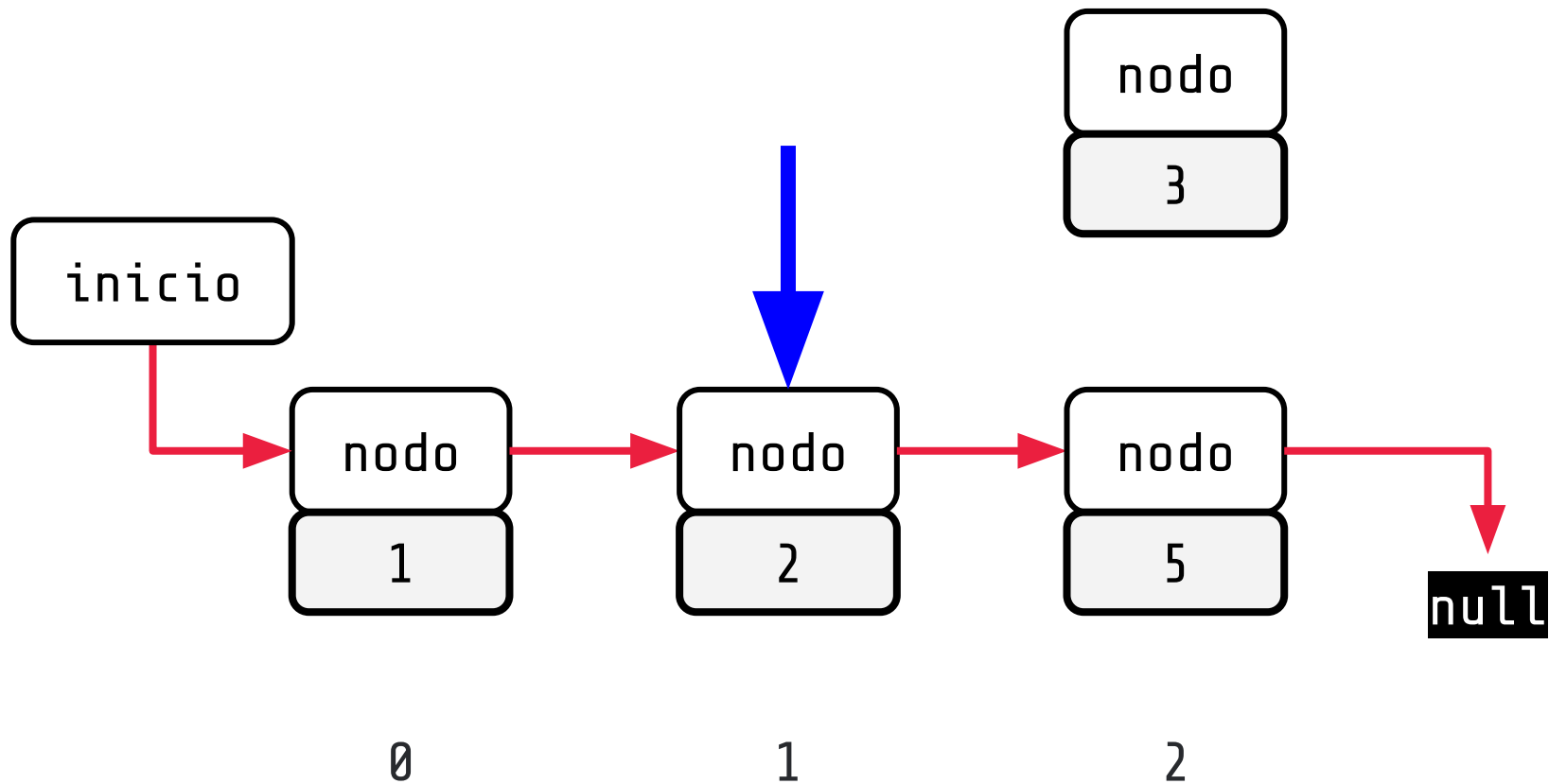
Fragmentación de memoria

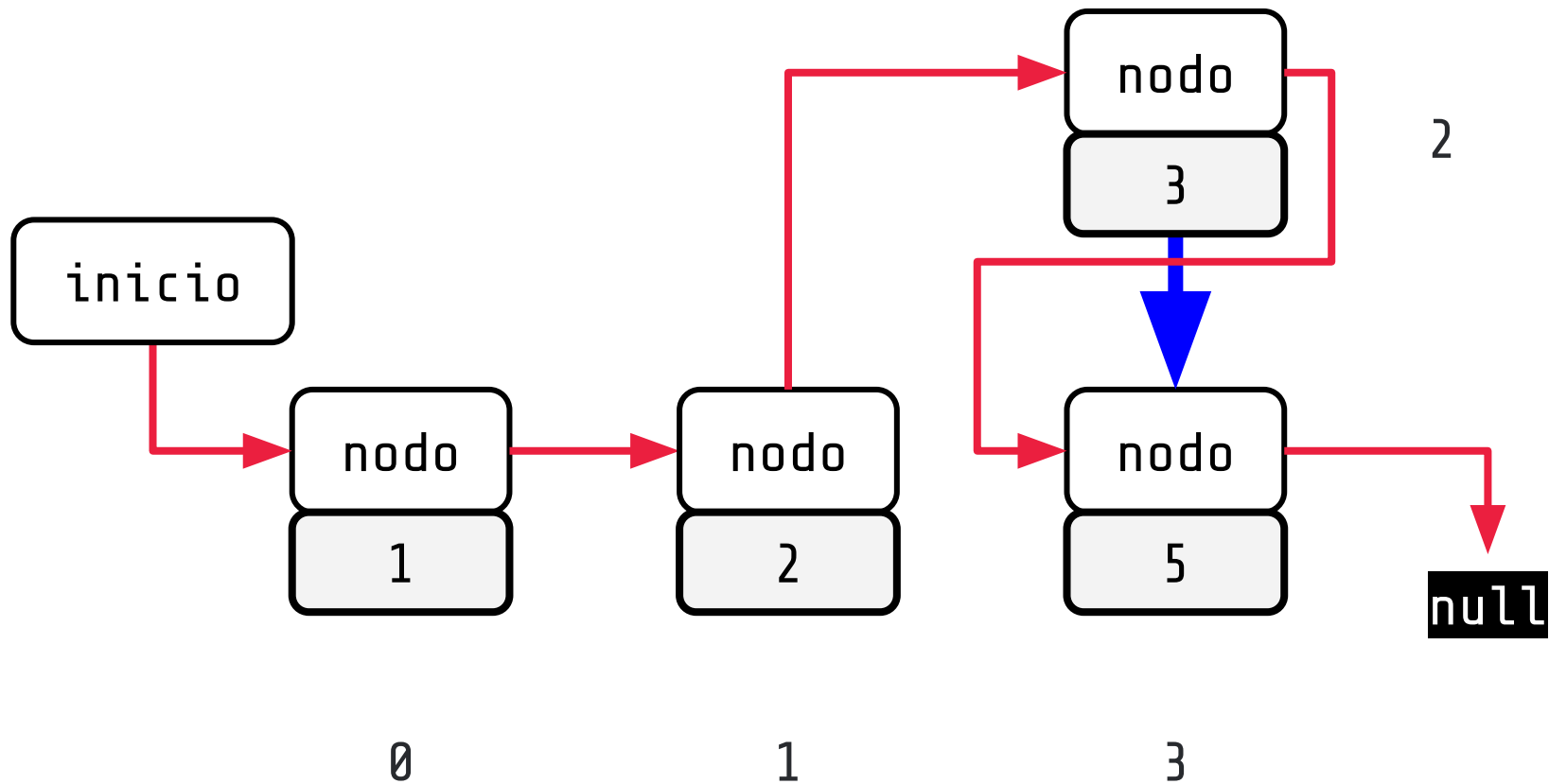
Listas enlazadas

- **Acceso: $O(n)$ / $\Omega(1)$**
- **Modificación: $O(n)$ / $\Omega(1)$**
- **Inserción: $O(n)$ / $\Omega(1)$**
- **Borrado: $O(n)$ / $\Omega(1)$**
- **Ordenamiento: $O(n^2)$ / $\Omega(1)$**
- **Ampliación: $\Omega(1)$ a $O(n)$**









**Pero, no es la
única
implementación**

| | | | | | | | | | | |
|---------|------|------|------|------|------|------|------|------|------|------|
| inicio | 4 | | | | | | | | | |
| enlaces | 1 | -1 | 0 | 6 | 3 | -1 | 2 | -1 | -1 | -1 |
| datos | dato | dato | dato | dato | dato | dato | dato | dato | dato | dato |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Podemos usar un doble arreglo



¿Preguntas?

Usos y aplicaciones

Gestión de memoria

Almacenamiento de 'cosas' grandes

—

Y es la base de otras estructuras lineales

Pila / stack

LIFO

Operaciones

Stack:

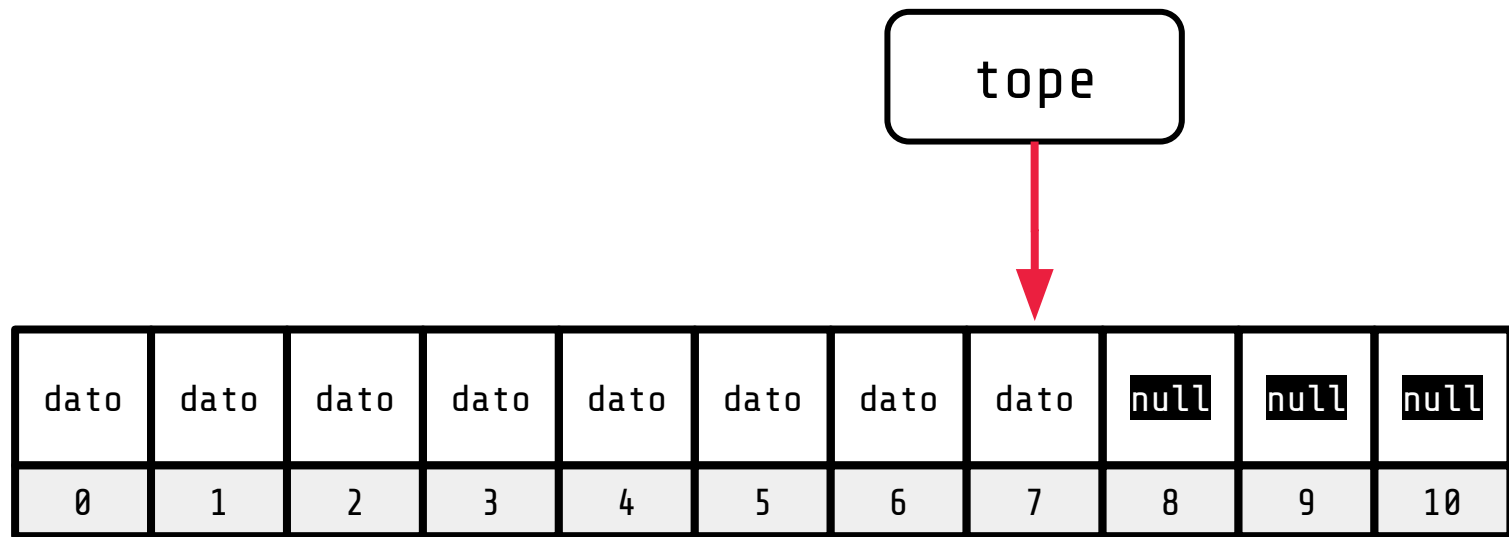
- push - agregar un elemento a la pila
- pop - extraer un elemento
- top - ver el siguiente elemento que se extraería
- isEmpty - si hay elementos en la Pila.

Operaciones

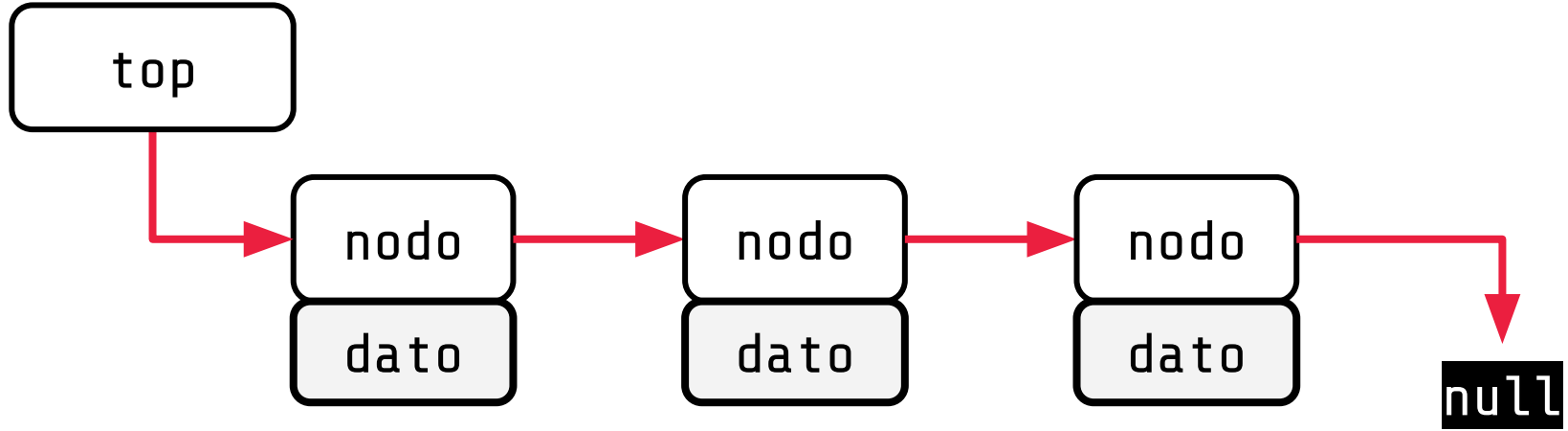
***¡Todas
O(1)!***

Stack:

- push - agregar un elemento a la pila
- pop - extraer un elemento
- top - ver el siguiente elemento que se extraería
- isEmpty - si hay elementos en la Pila.



Simple, fija con arreglos



Simple, dinámica con nodos

Ventajas

Operaciones rápidas
Fácil implementación

Desventajas

Acceso limitado

Usos

Procesamiento de lenguajes estructurados
Gestión de memoria en llamadas a función
Algoritmos de búsqueda
Implementaciones de deshacer

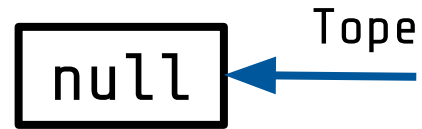
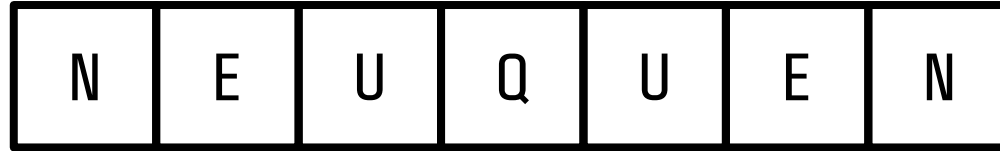


¿Preguntas?

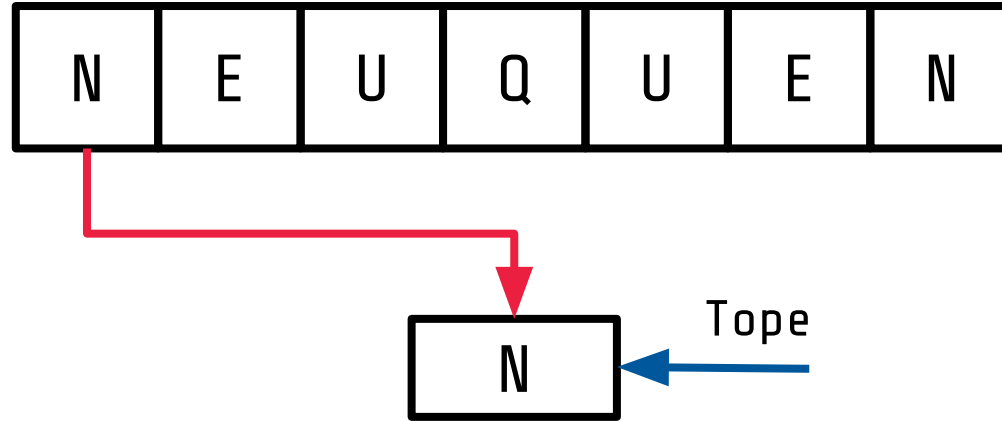
Ejemplo de uso de Stack

esPalindromo?

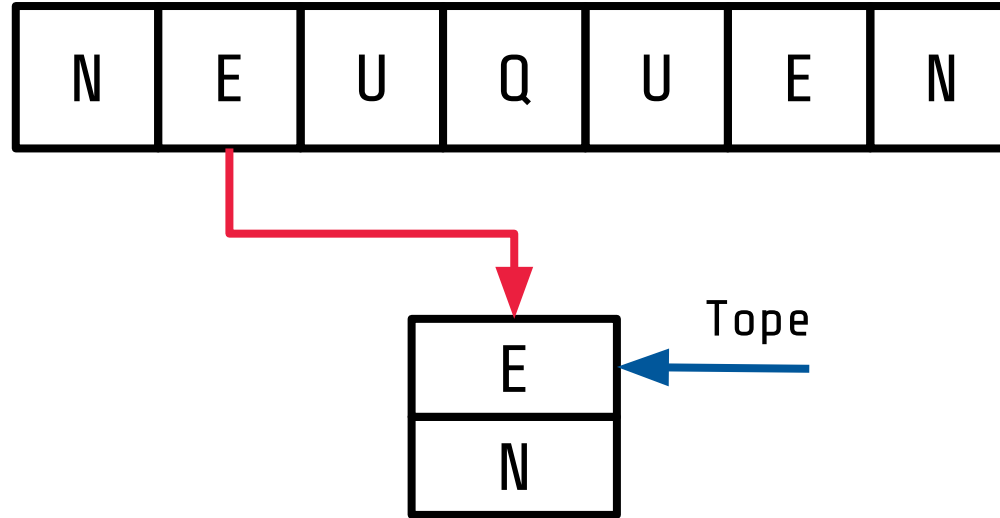
Ejemplo



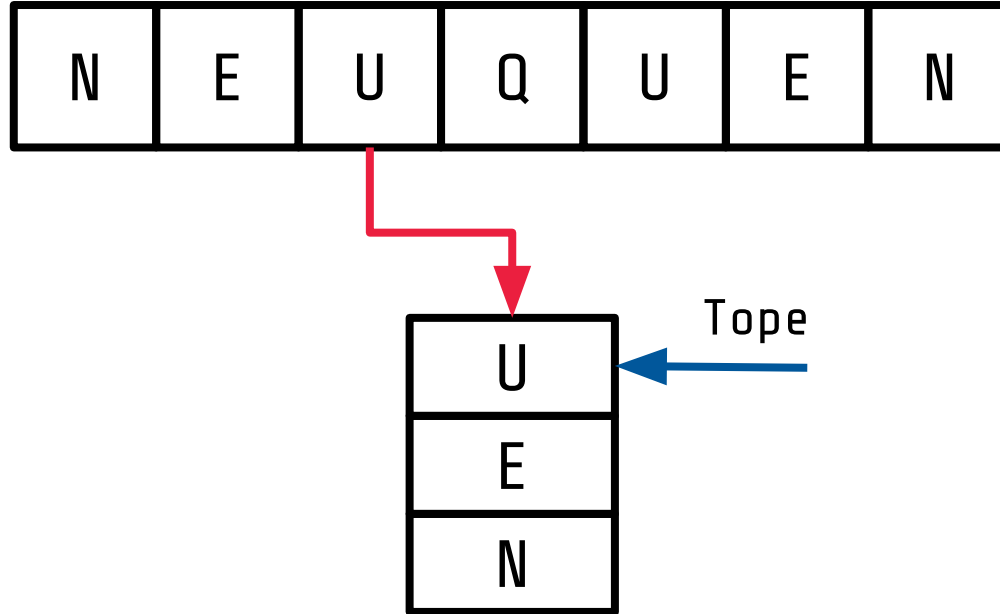
Cargamos el stack



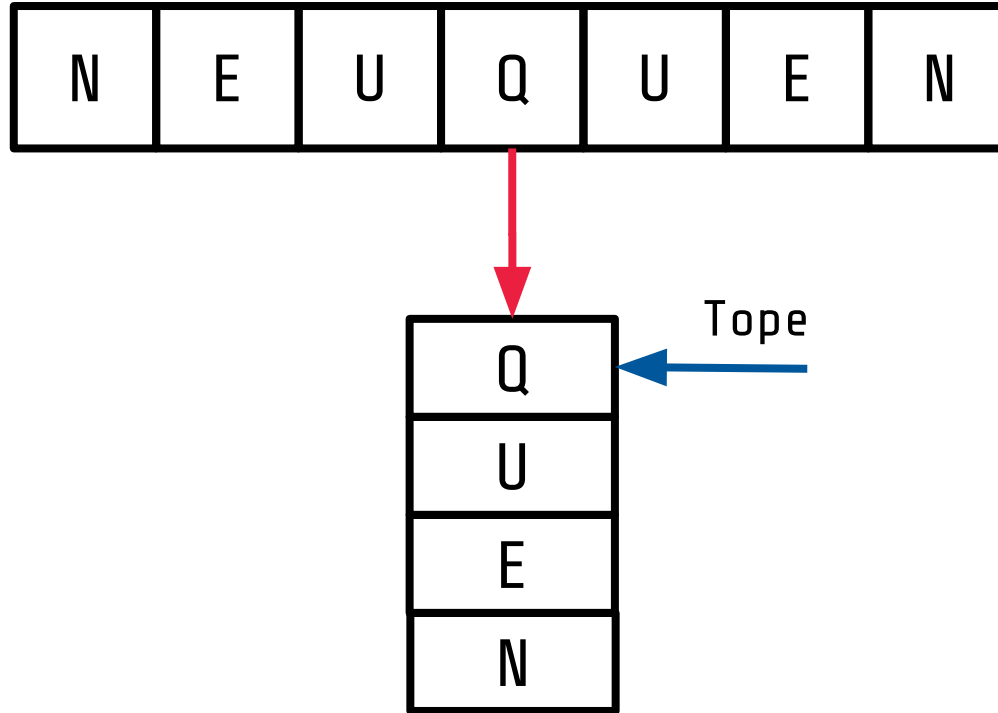
Cargamos el stack



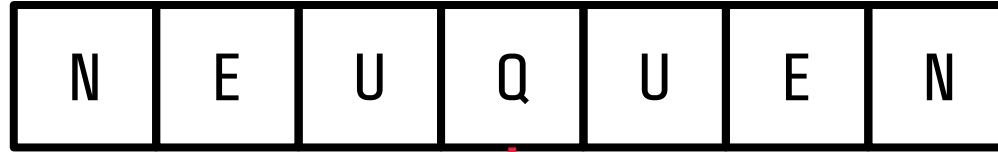
Cargamos el stack



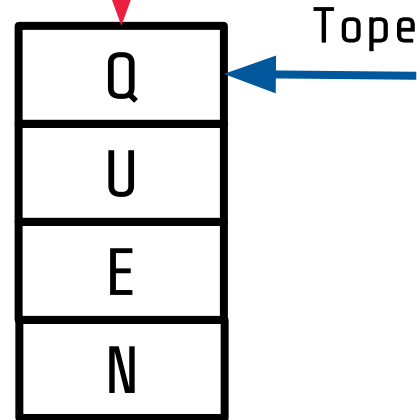
Cargamos el stack hasta largo/2



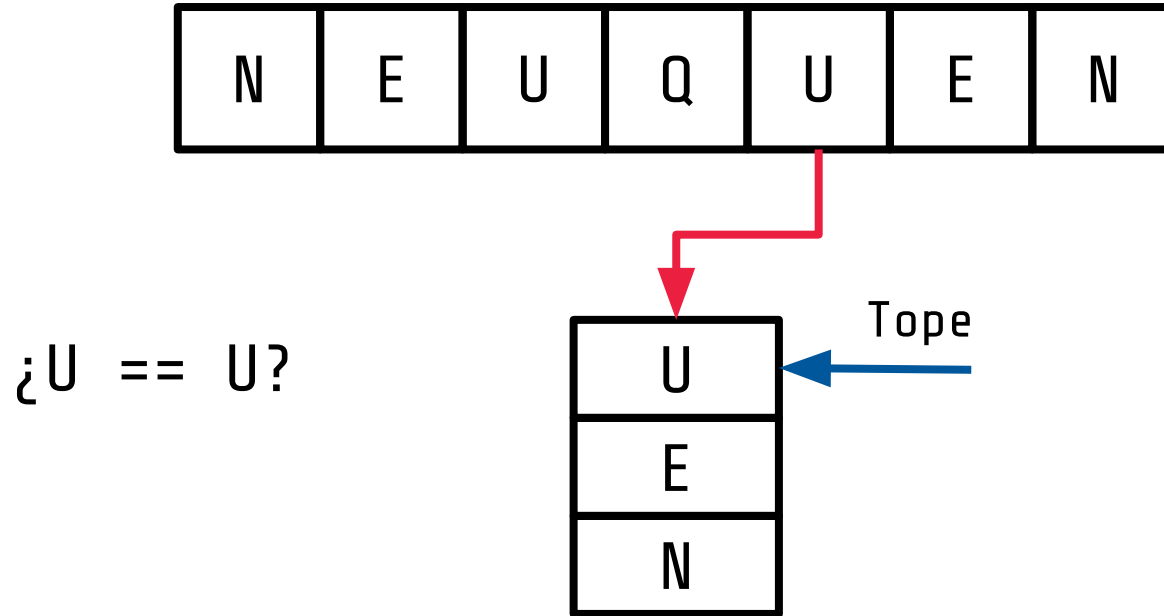
Y ahora lo descargamos



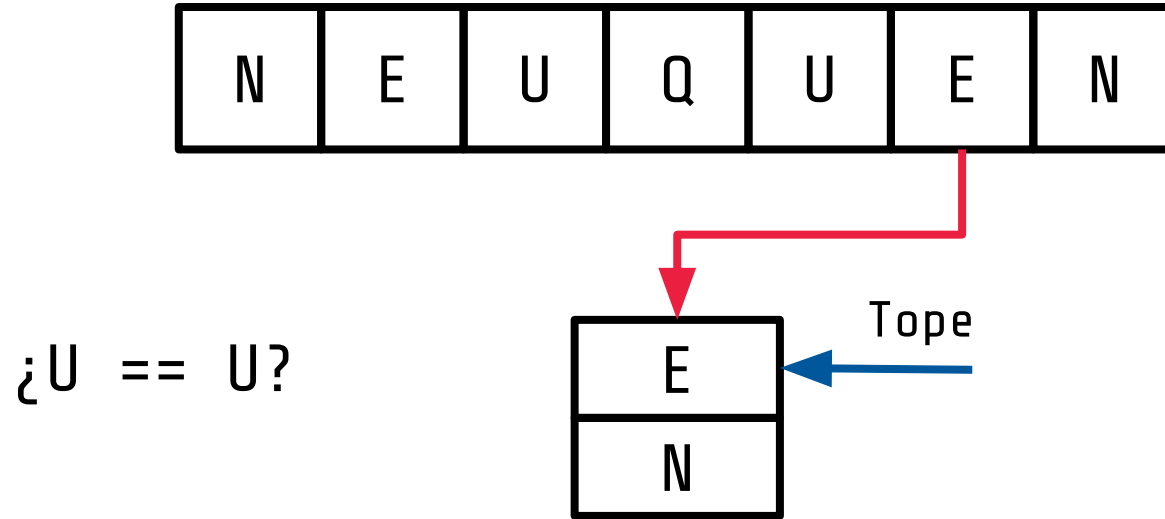
¿Q == Q?



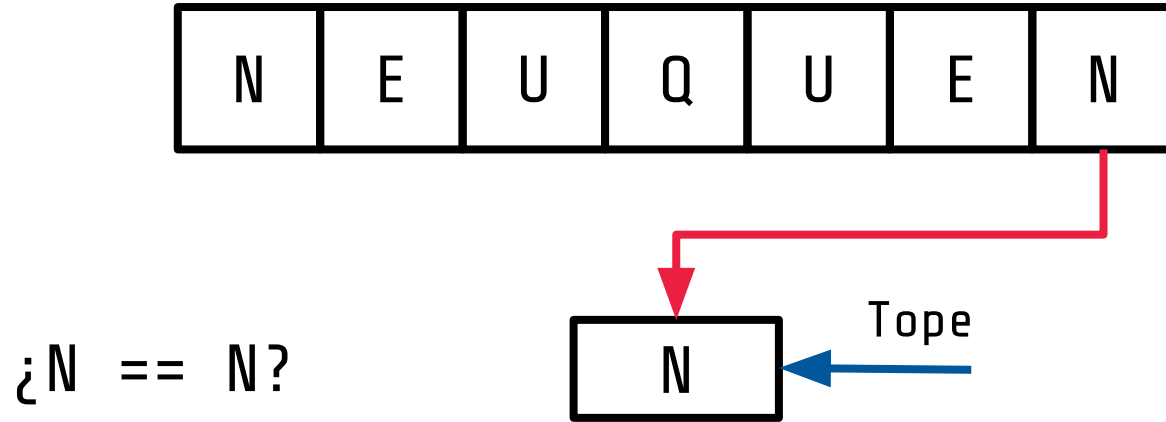
Y ahora lo descargamos



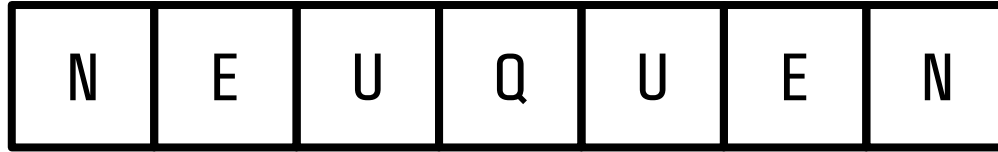
Y ahora lo descargamos



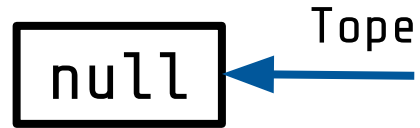
Y ahora lo descargamos



¿Esta vacío?



`isEmpty()`?





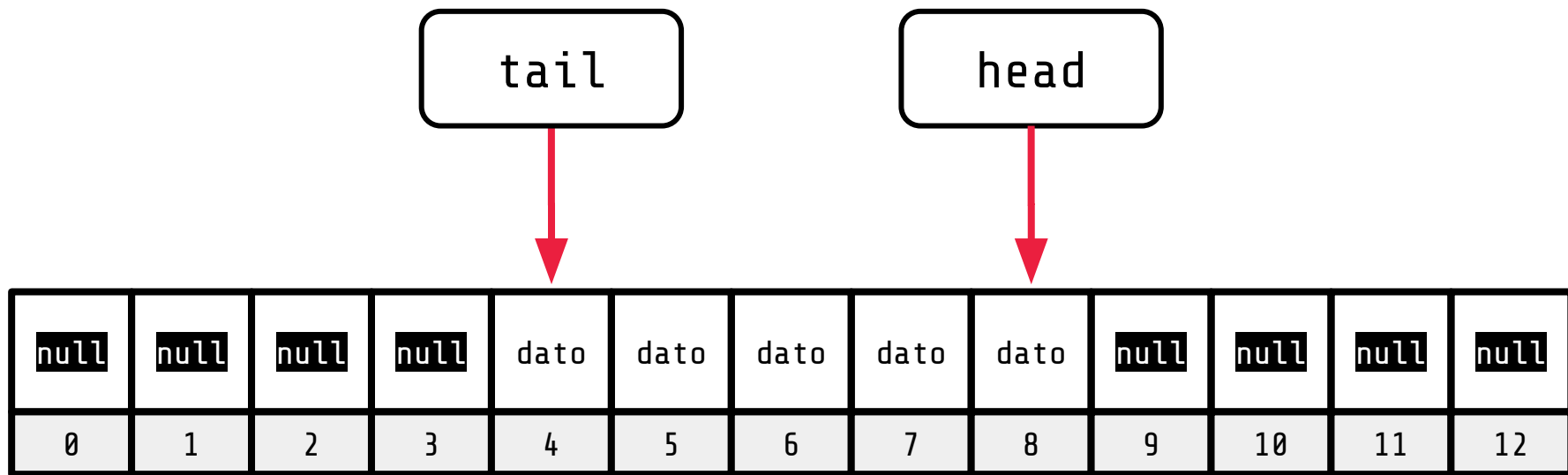
¿Preguntas?

Cola / queue FIFO

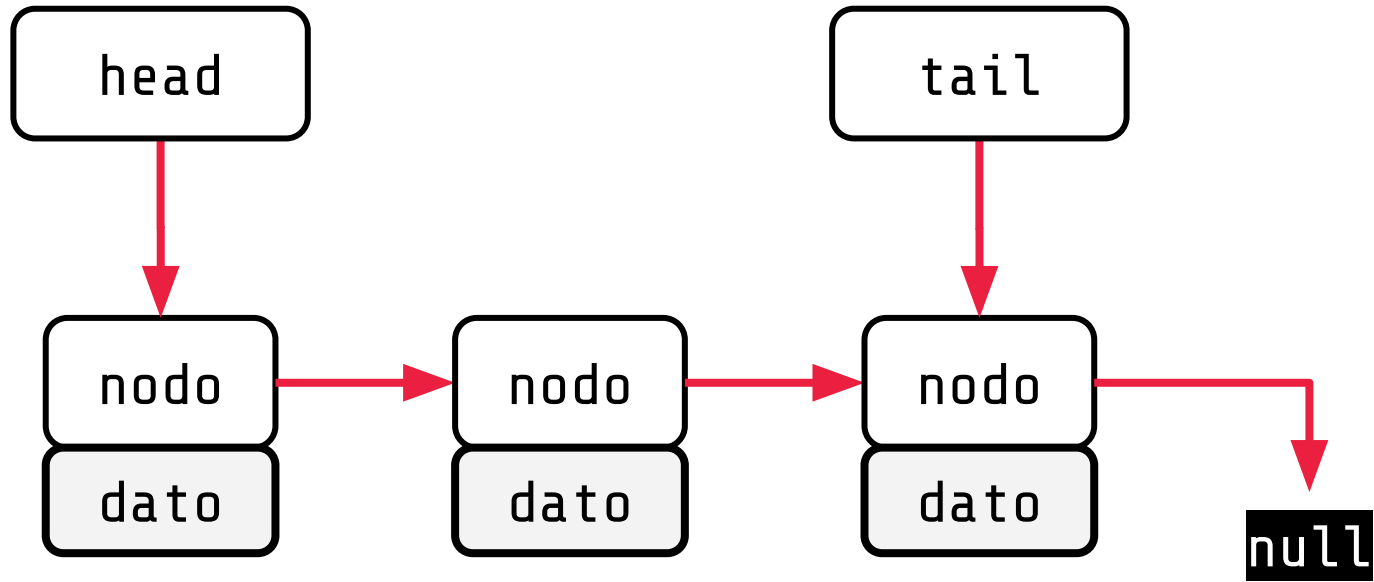
Operaciones

Queue:

- `put`: agregar un elemento por el final (`tail`).
- `get`: tomar un elemento del inicio (`head`)
- `isEmpty`: Si no hay elementos
- `peek`: el siguiente elemento de la Queue sin sacarlo.



Simple, fija con arreglos



Simple, dinámica con nodos

Ventajas

Conservación del orden de llegada

Operaciones rápidas

Fácil implementación

Desventajas

Acceso limitado

Usos comunes

Gestión de trabajos en espera

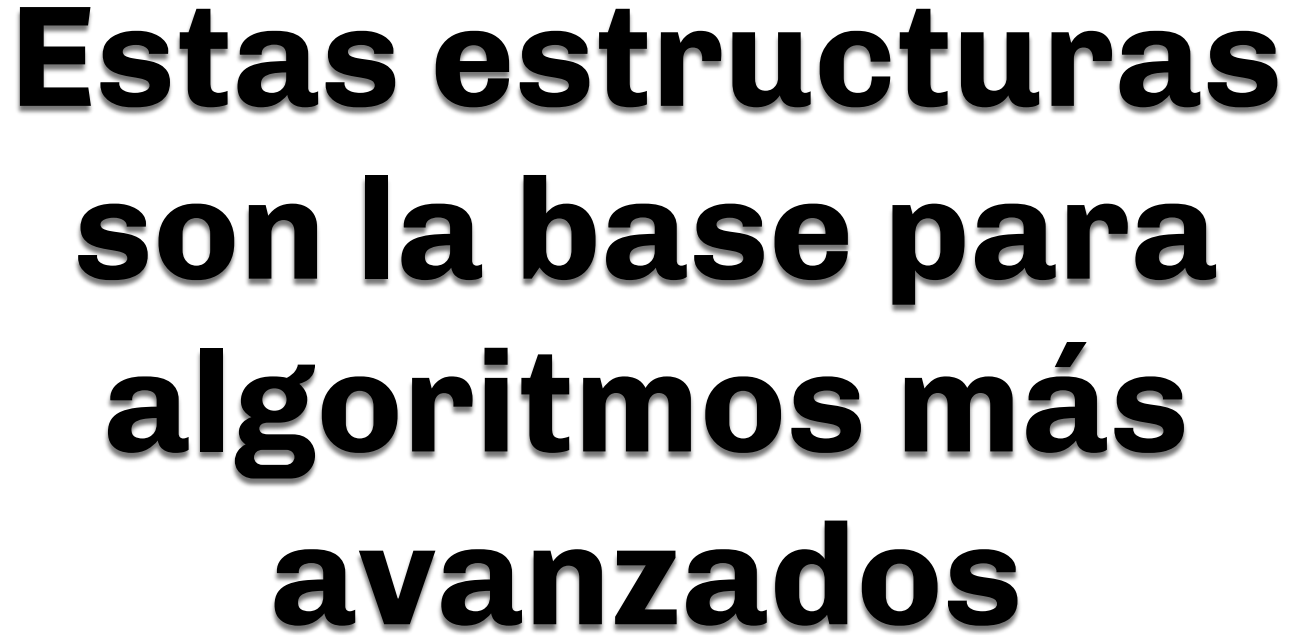
Buffers de datos

Comunicación entre procesos

Planificación de tareas



¿Preguntas?



**Estas estructuras
son la base para
algoritmos más
avanzados**



**Que veremos
más adelante**

unrn.edu.ar

UNRN

Universidad Nacional
de **Río Negro**



| **unrionegro**