TADs
Secuencias

Estructuras de Datos

Andrea Rueda

Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas

Recordatorio...

- Entrega 0 del proyecto:

  Viernes 9 de febrero a la medianoche

TAD
Tipo Abstracto de Dato

¿Qué es un TAD?

- TAD: Tipo Abstracto de Dato.
  - Nuevo tipo de dato bien especificado (propio).
  - Amplía el lenguaje de programación.
- Consta de:
  - Datos (representación):
    invisibles al usuario (protegidos o privados).
  - Funciones y procedimientos (operaciones):
    encapsulados dentro del TAD, acceso por interfaz.

¿Qué es un TAD?

- TAD: Tipo Abstracto de Dato.
  - Nuevo tipo de dato bien especificado (propio).
  - Amplía el lenguaje de programación.
- Definición:
  - Especificación (¿qué hace el TAD?)
    General, global. Formal, matemática.
  - Implementación (¿cómo lo hace el TAD?)
    Dependiente del lenguaje a usar.

¿Qué es un TAD?

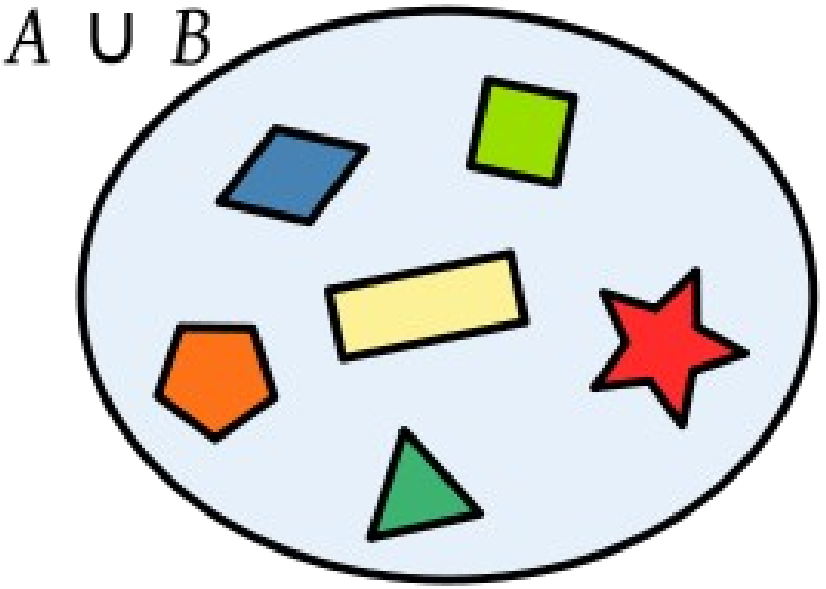
Ventajas:

- Mejor conceptualización y modelización.
- Mejora la robustez.
- Mejora el rendimiento (optimización del tiempo de compilación).
- Separa la implementación de la especificación.
- Permite extensibilidad.

¿Qué es un TAD?

Ejemplos:

- TAD Conjunto
- TAD Carrera
- TAD Biblioteca
- TAD Vehículo



https://co.pinterest.com/pin/532832199639015903/

http://es.wikipedia.org/wiki/Unión_de_conjuntos

Diseño de TADs

- Programas = Algoritmos + Datos (ecuación de Wirth).
  https://web.archive.org/web/20130207170133/http://www.inf.ethz.ch/personal/wirth/books/AlgorithmE0/

- Diseño de TADs:
  - Comportamientos del objeto en el mundo real (interfaz/verbos).
  - ***Conjunto mínimo*** de datos que lo representan (estado/sustantivos).
  - TAD = Interfaz + Estado.

Diseño de TADs

- Plantilla base para diseño de TADs:

  TAD *nombre*

  Conjunto mínimo de datos:
  nombre valor (variable), tipo de dato, descripción
  ...

  Comportamiento (operaciones) del objeto:
  nombre operación (argumentos), descripción funcional
  ...

Diseño de TADs

- Plantilla base para diseño de TADs:

  TAD *Carrera*

  Conjunto mínimo de datos:
  - *nombre, cadena de caracteres, identificación de la carrera*
  - *numEst, número entero, cantidad de estudiantes inscritos*

  Comportamiento (operaciones) del objeto:
  ObtenerNombre(), retorna el nombre de la carrera
  ObtenerNumEst(), retorna la cantidad de estudiantes de la carrera
  FijarNombre(nuevoNom), modifica el nombre actual de la carrera
  AgregarEstud(), incrementa en uno el conteo de estudiantes
  EliminarEstud(), decrementa en uno el conteo de estudiantes

Diseño de TADs

- Diagrama de relación entre TADs:

¡Atención!
¡No es un diagrama de clases!

Implementación de TADs

- Diferentes posibilidades:
  - Estructura con datos y funciones propias o adicionales.
  - Clase con niveles de acceso.
- Utilizando librerías:
  - En el archivo de cabecera (.h) se realiza la declaración de los datos y los prototipos.
  - En el archivo de implementación (.cpp ó .cxx) se desarrolla la implementación de las operaciones.

Implementación de TADs

- Librería, 2 archivos:

  - **Encabezado (.h)**:

  Incluye la definición del TAD, en términos de datos y prototipos de operaciones.

  - **Implementación (.cpp ó .cxx):**

  Incluye la implementación específica de las operaciones del TAD.

  `#include "TAD.h"` (al principio del archivo)

Implementación de TADs

- Opción 1: datos en estructura, operaciones como funciones adicionales.

```
struct Carrera{
  std::string nombre;
  unsigned long numEst;
};


std::string ObtenerNombre(Carrera *c);
unsigned long ObtenerNumEst(Carrera *c);
void FijarNombre(Carrera *c, std::string nombre);
void AgregarEstud(Carrera *c);
void EliminarEstud(Carrera *c);
```

Implementación de TADs

- Opción 2: datos y operaciones en estructura.

```
struct Carrera{
  std::string nombre;
  unsigned long numEst;


  std::string ObtenerNombre();
  unsigned long ObtenerNumEst();
  void FijarNombre(std::string nombre);
  void AgregarEstud();
  void EliminarEstud();
};
```

Implementación de TADs

- Opción 3: datos y operaciones dentro de una clase de C++.

```
class Carrera {
  public:
    Carrera();
    std::string ObtenerNombre();
    unsigned long ObtenerNumEst();
    void FijarNombre(std::string name);
    void AgregarEstud();
    void EliminarEstud();
  protected:
    std::string nombre;
    unsigned long numEst;
};
```

Implementación de TADs

- Sintaxis básica en encabezado:

  - Regiones:

    - Públicas (`public`):
      Todas las operaciones del TAD.
      Dan acceso a los datos (atributos) del TAD.
    - Protegidas (`protected`) o Privadas (`private`):
      Todos los datos (atributos) del TAD.

Diseño e implementación de TADs

1. Diseñar el TAD:
   descripción + diagrama.
2. Escribir el archivo de encabezado (.h):
   un archivo por TAD.
3. Escribir el archivo de implementación (.cxx, .cpp):
   un archivo por TAD.
4. Escribir el archivo de programa (.cxx, .cpp):
   programa con procedimiento principal,
   incluye el encabezado de cada TAD a utilizar.

¡Atención!

- Recordatorio:

  El **diseño e implementación** descrito anteriormente será **exigido** en todas las actividades del curso (talleres, parciales, proyecto, quices) **de aquí en adelante, durante todo el semestre.**

Secuencias

Secuencias

- Una secuencia es una estructura que representa una lista de elementos del mismo tipo.
- Formalmente:

$$S = \{ s_i \in T : 1 \leq i \leq n , i \in \mathbb{N} \}$$

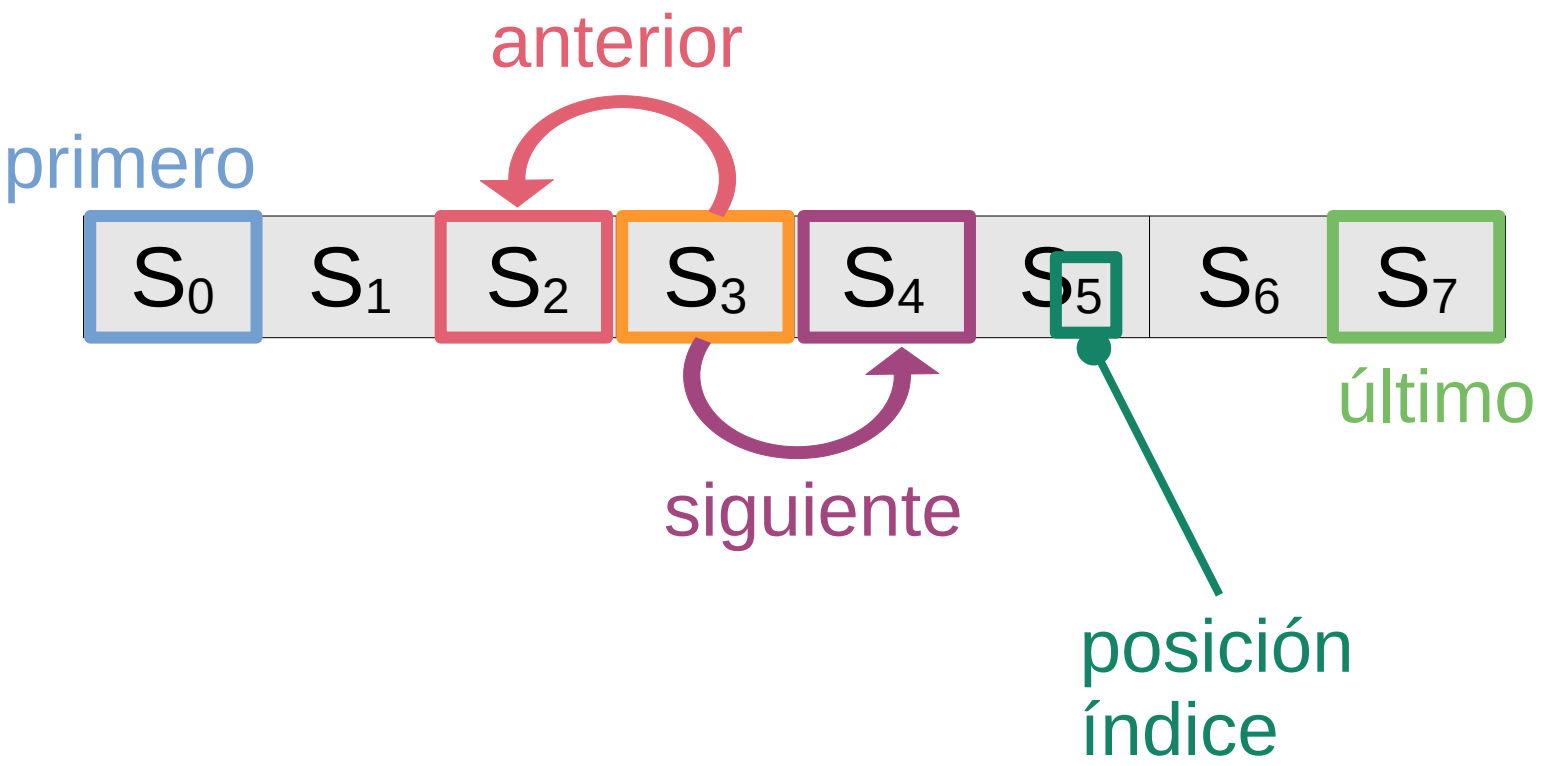
$$S = \{ s_1, s_2, s_3, \ldots, s_n \}$$

- Ejemplo: secuencia de los números de Fibonacci

1	1	2	3	5	8	13	...
S_0	S_1	S_2	S_3	S_4	S_5	S_6	...

Secuencias

- Conceptos relevantes:



Secuencias

- Conceptos relevantes:

Secuencia 1

S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7

Secuencia 2 ≠ Secuencia 1

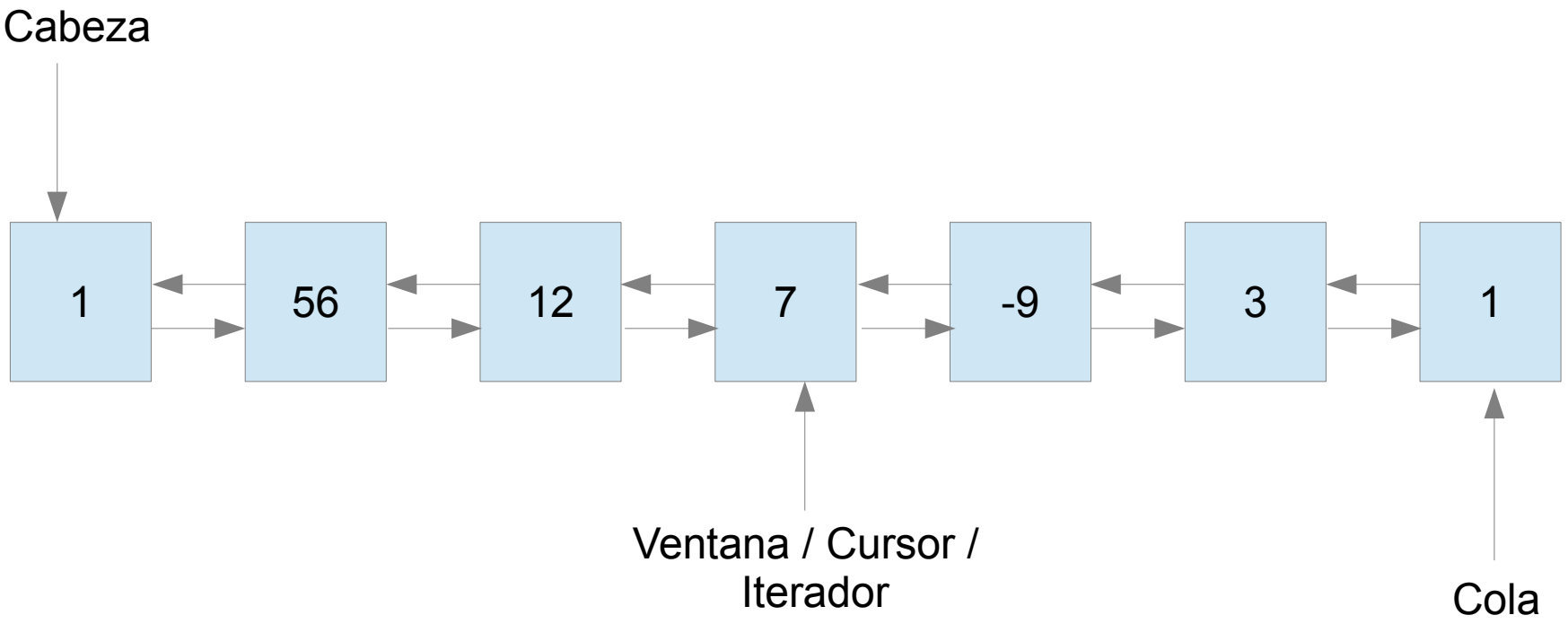
S_1	S_3	S_6	S_0	S_2	S_5	S_7	S_4

TAD Lista

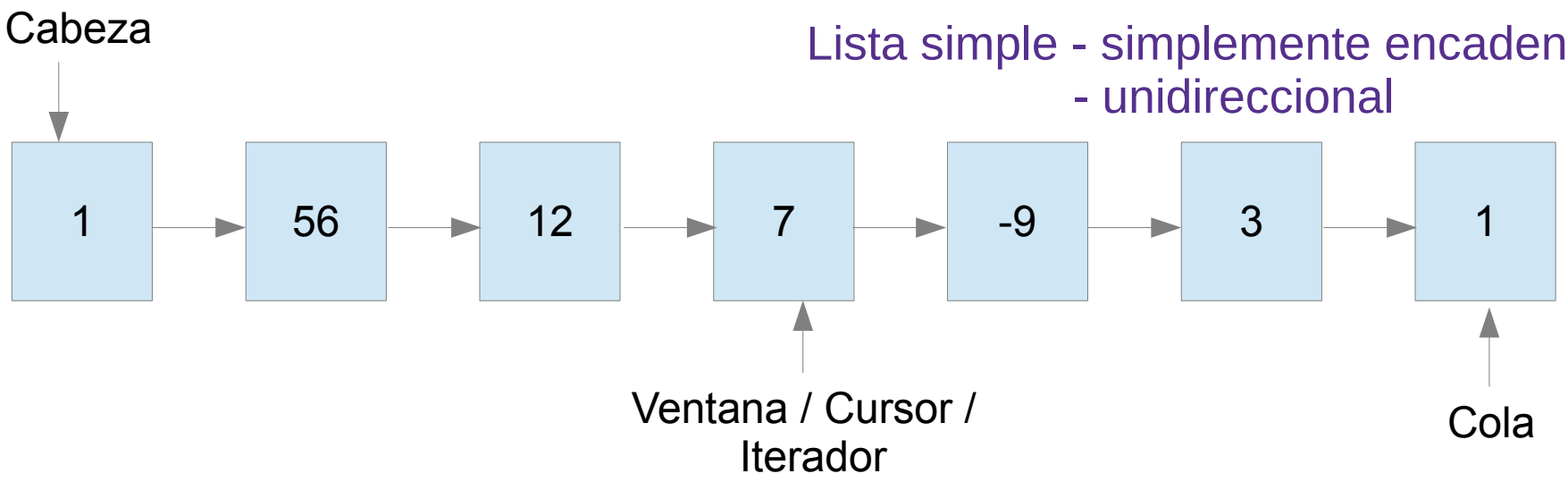
TAD Lista

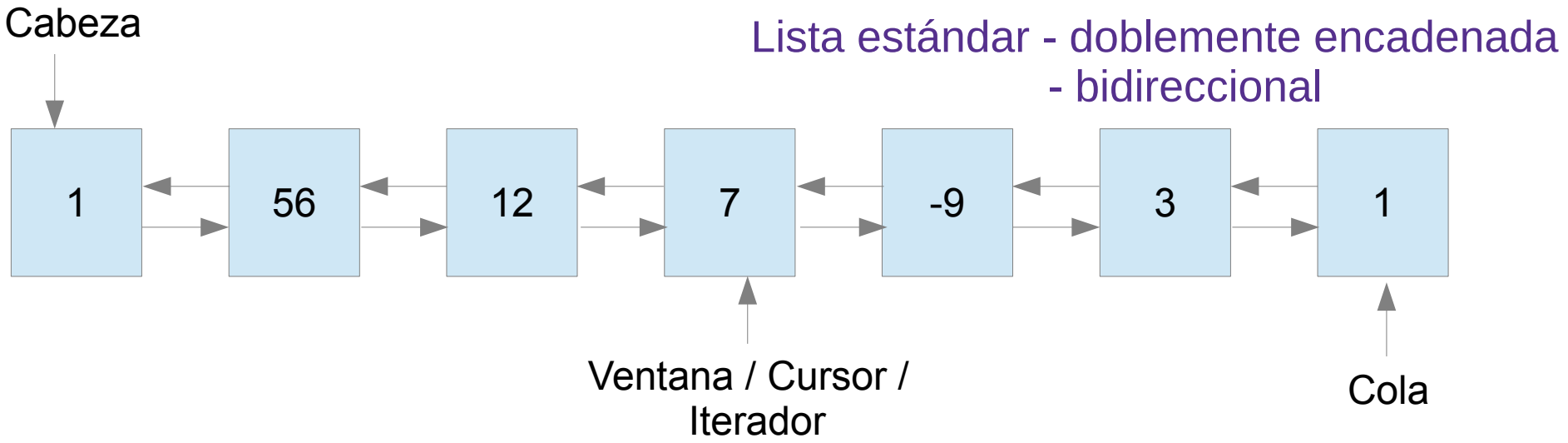
- Secuencia finita de datos:
  - Primero, último.
  - Siguiente, anterior.
- Recorrido:
  - Primero → Último.
  - Último → Primero.
- Acceso aleatorio:
  - Restringido.
- Algoritmos:
  - Vacía, 1 elemento.
  - Cabeza, cola.
  - Intermedio.

TAD Lista

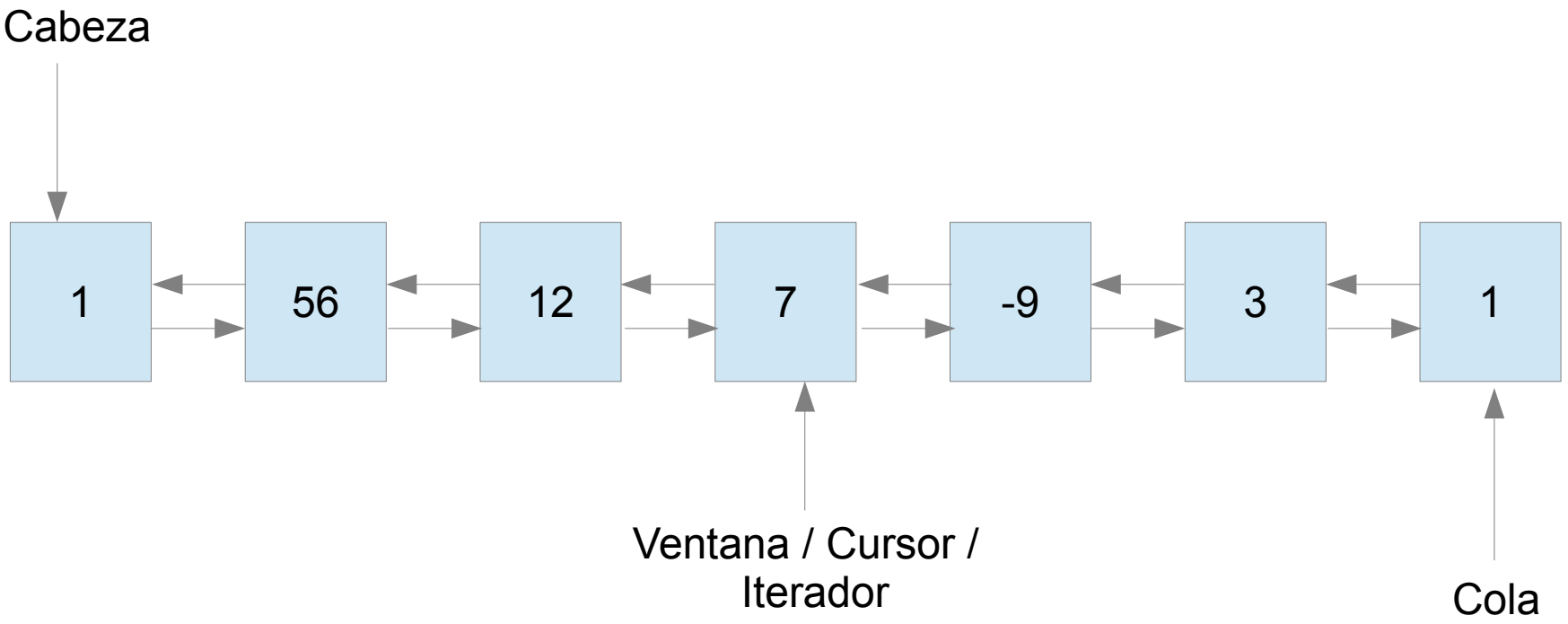


TAD Lista


Cabeza
Lista simple - simplemente encadenada
- unidireccional
1
56
12
7
-9
3
1
Ventana / Cursor /
Iterador
Cola


Cabeza
Lista estándar - doblemente encadenada
- bidireccional
1
56
12
7
-9
3
1
Ventana / Cursor /
Iterador
Cola

TAD Lista



¿Estado?

¿Interfaz?

TAD Lista

- Estado:
  - Cabeza.
  - Cola.
  - $\{E_i\}$ (¿importa el tipo?).

- Interfaz:
  - Creadoras.
  - Analizadoras: tamaño, acceso, cabeza, cola.
  - Modificadoras: insertar, eliminar.

TAD Lista

TAD Lista

Conjunto mínimo de datos:

- ...
- ...

Comportamiento (operaciones) del objeto:

- ...
- ...
- ...
- ...
- ...
- ...

TAD Lista

TAD Lista

Conjunto mínimo de datos:

- cabeza, tipo plantilla, representa el punto de inicio.
- cola, tipo plantilla, representa el punto de finalización.

Comportamiento (operaciones) del objeto:

- esVacia(), indica si la lista está vacía.
- tamaño(), cantidad de elementos en la lista.
- cabeza(), retorna el elemento en la cabeza.
- cola(), retorna el elemento en la cola.
- insertarCabeza(v), inserta v en la cabeza.
- insertarCola(v), inserta v en la cola.

TAD Lista

TAD Lista

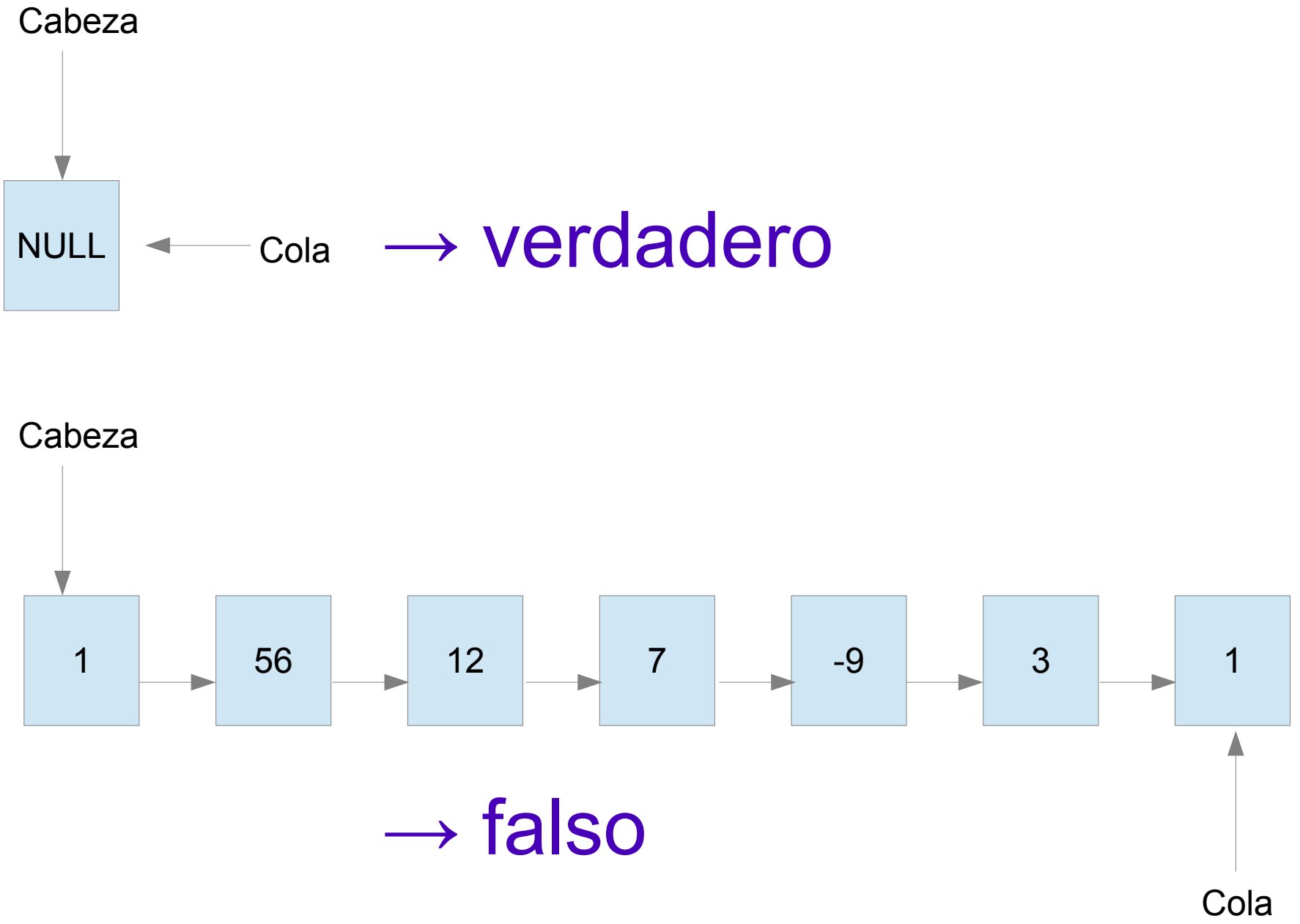
Conjunto mínimo de datos:

- cabeza, tipo plantilla, representa el punto de inicio.
- cola, tipo plantilla, representa el punto de finalización.

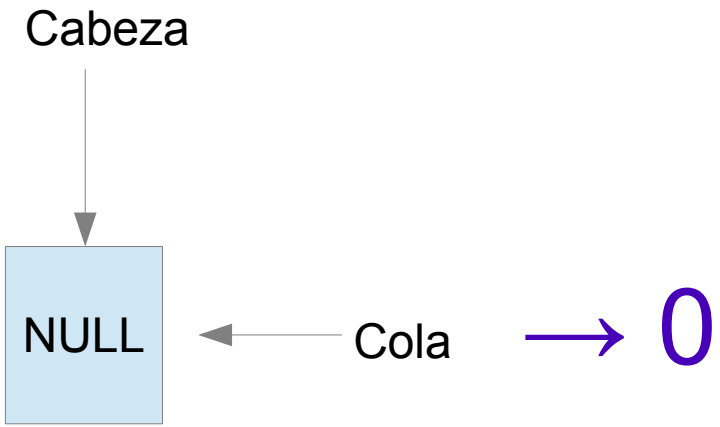
Comportamiento (operaciones) del objeto:

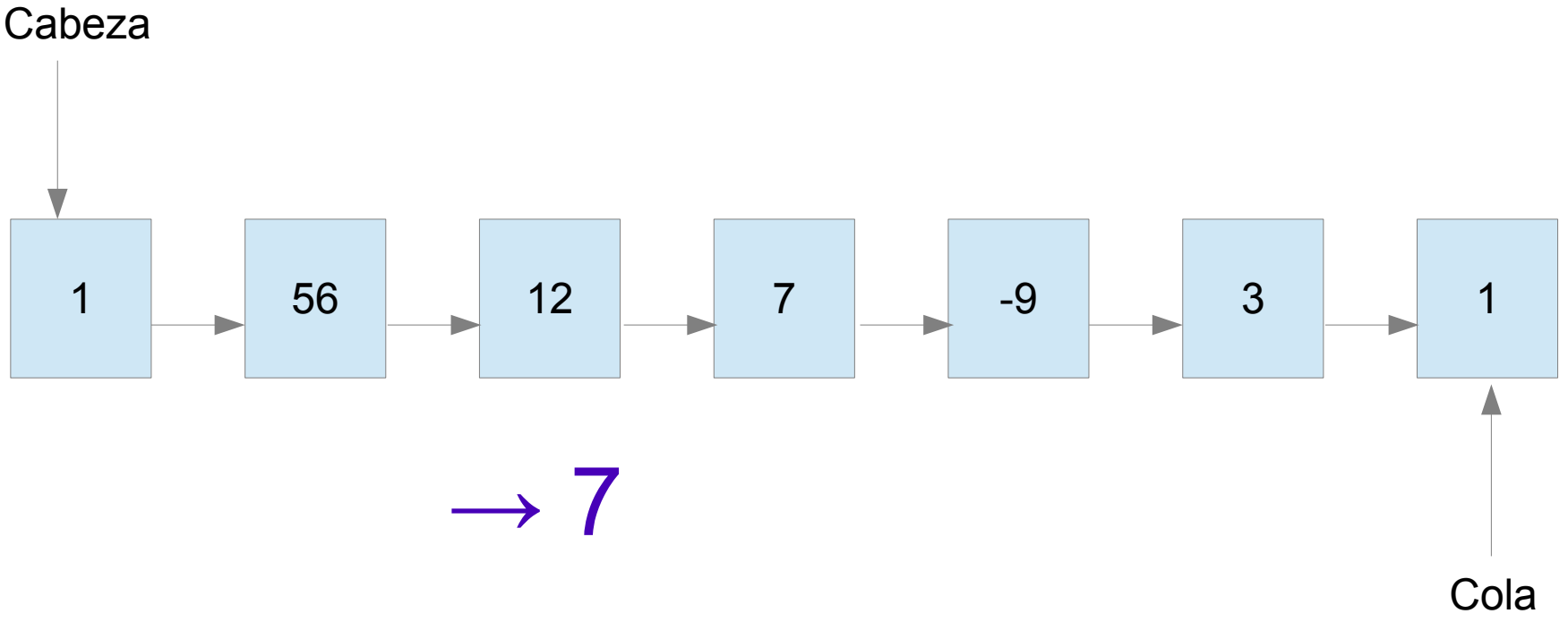
- eliminarCabeza(), elimina elemento de la cabeza.
- eliminarCola(), elimina elemento de la cola.
- insertar(pos,v), inserta v en la posición pos.
- eliminar(pos), elimina el elemento ubicado en pos.
- vaciar(), elimina todos los elementos de la lista.

TAD Lista: esVacia()

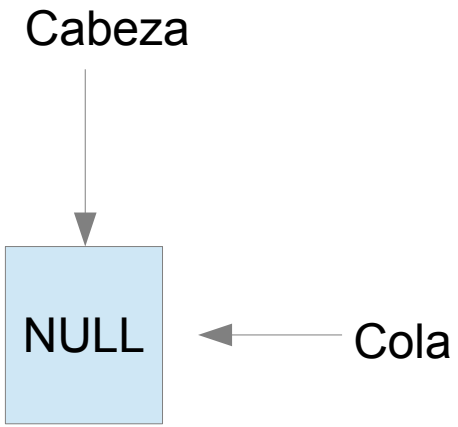


TAD Lista: tamaño()

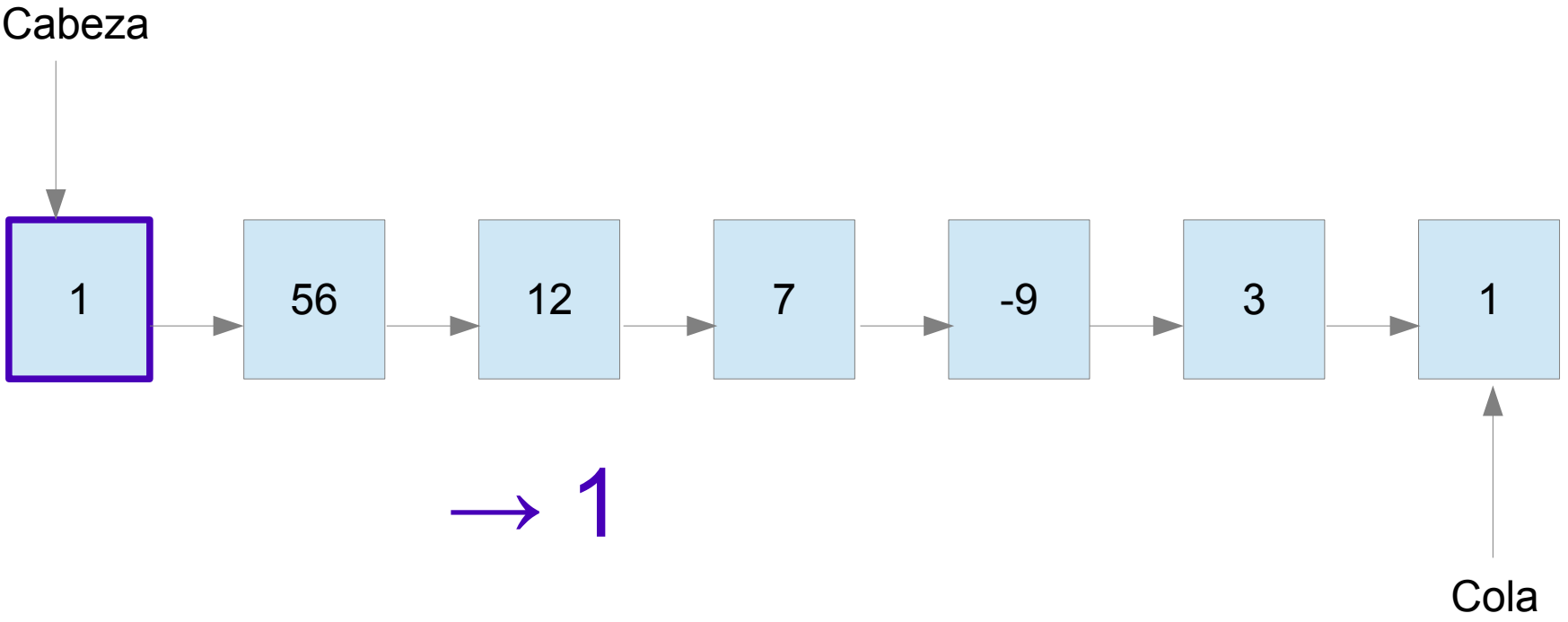




TAD Lista: cabeza()

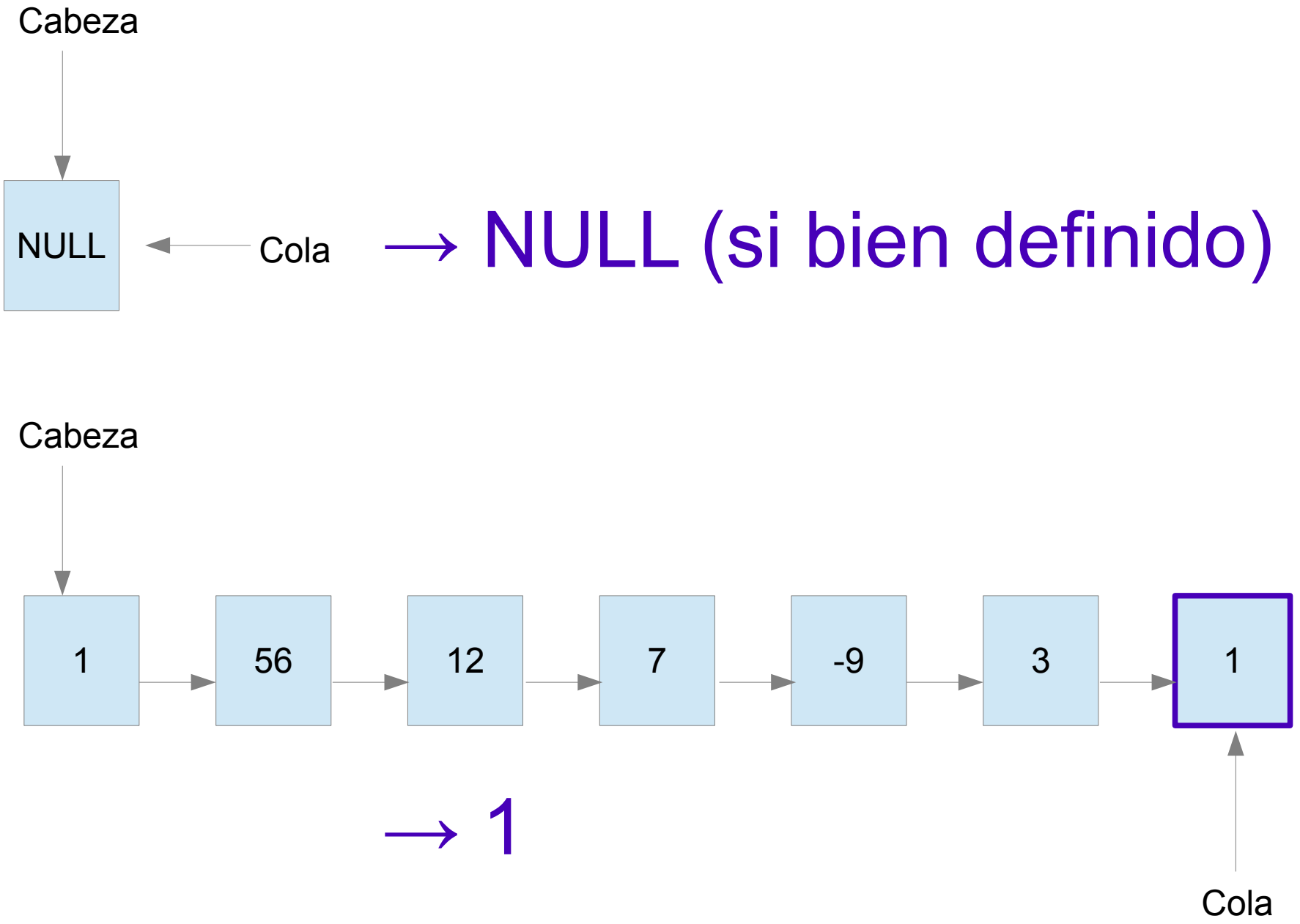


→ NULL (si bien definido)

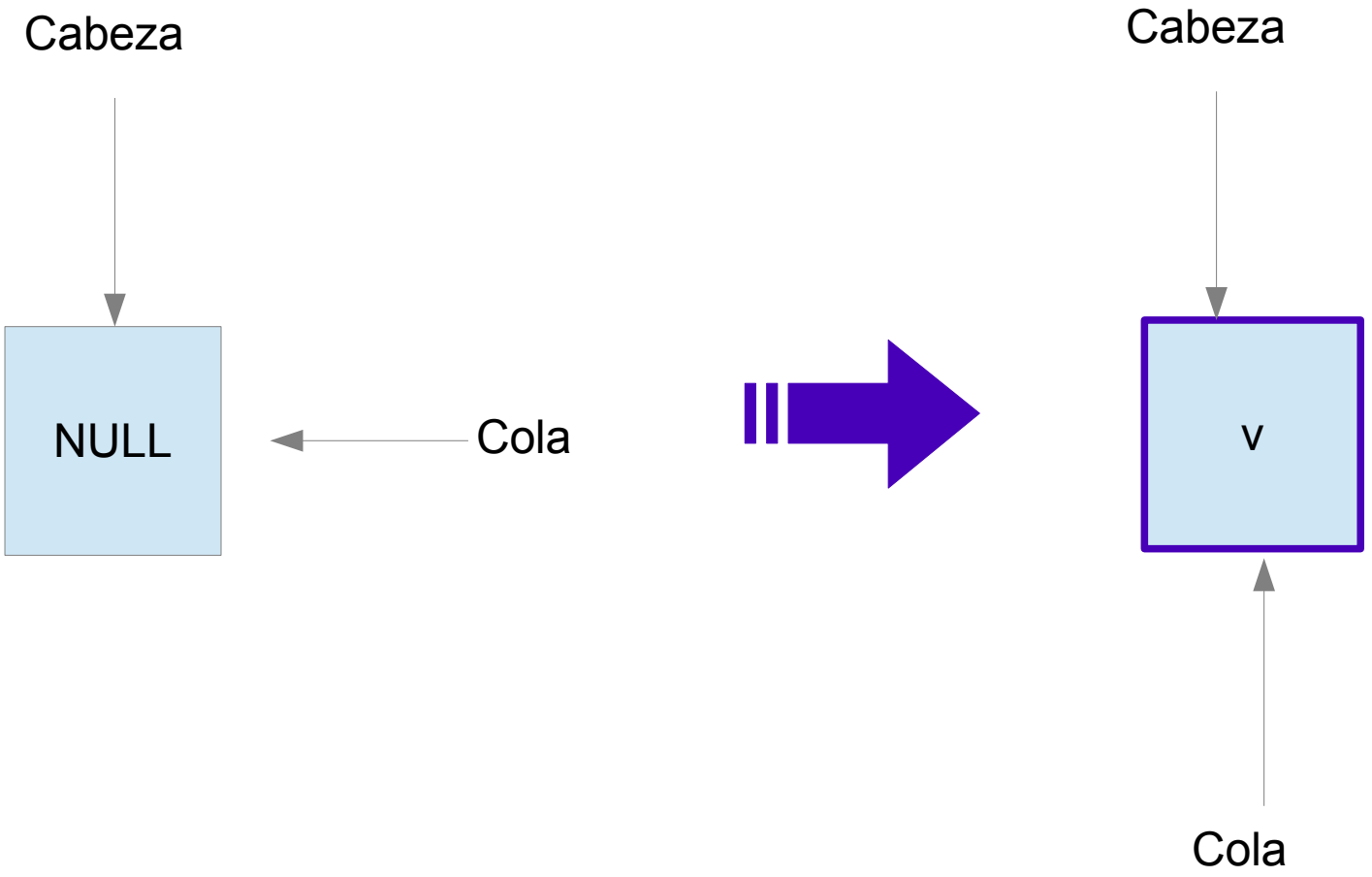


→ 1

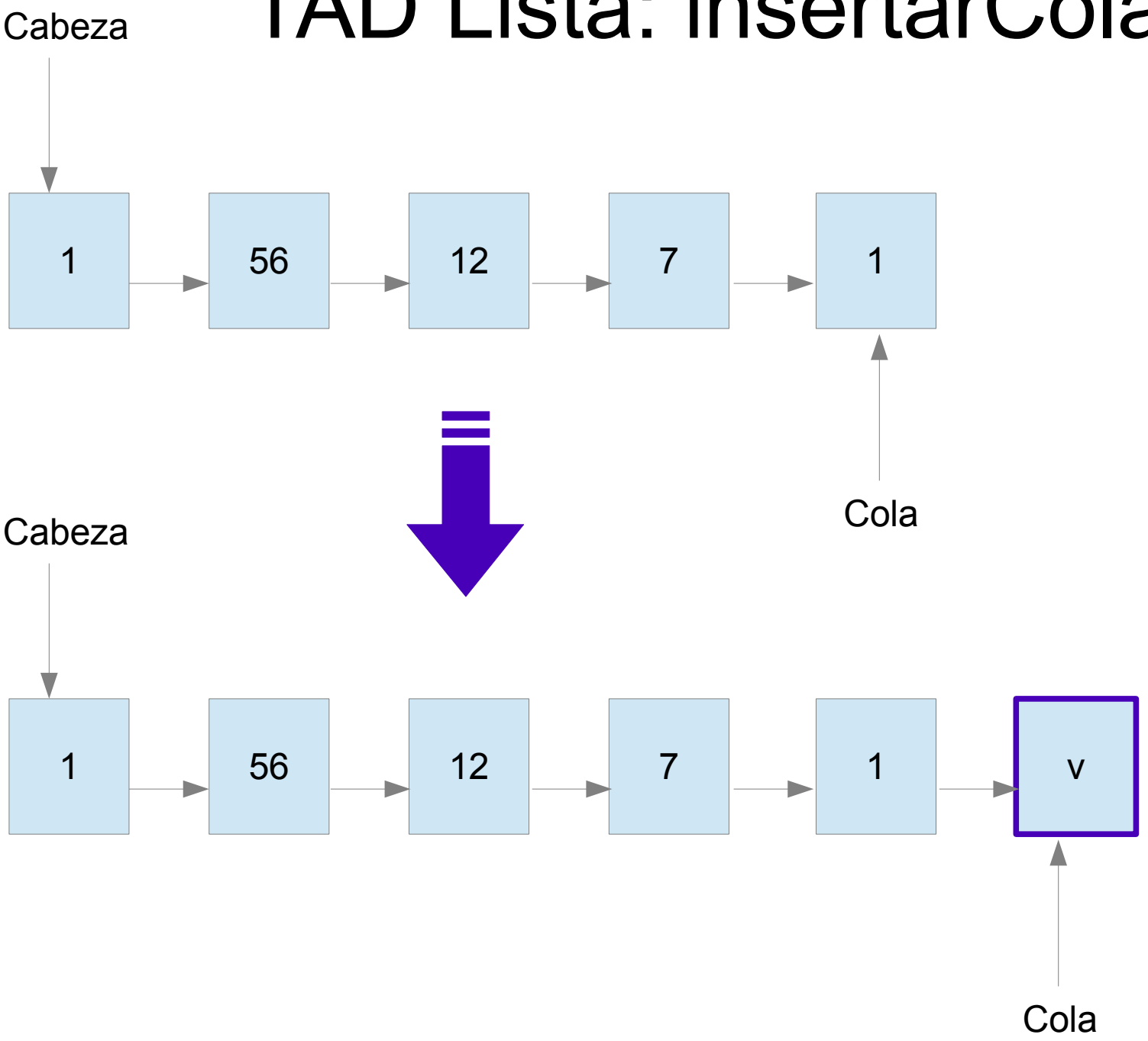
TAD Lista: cola()



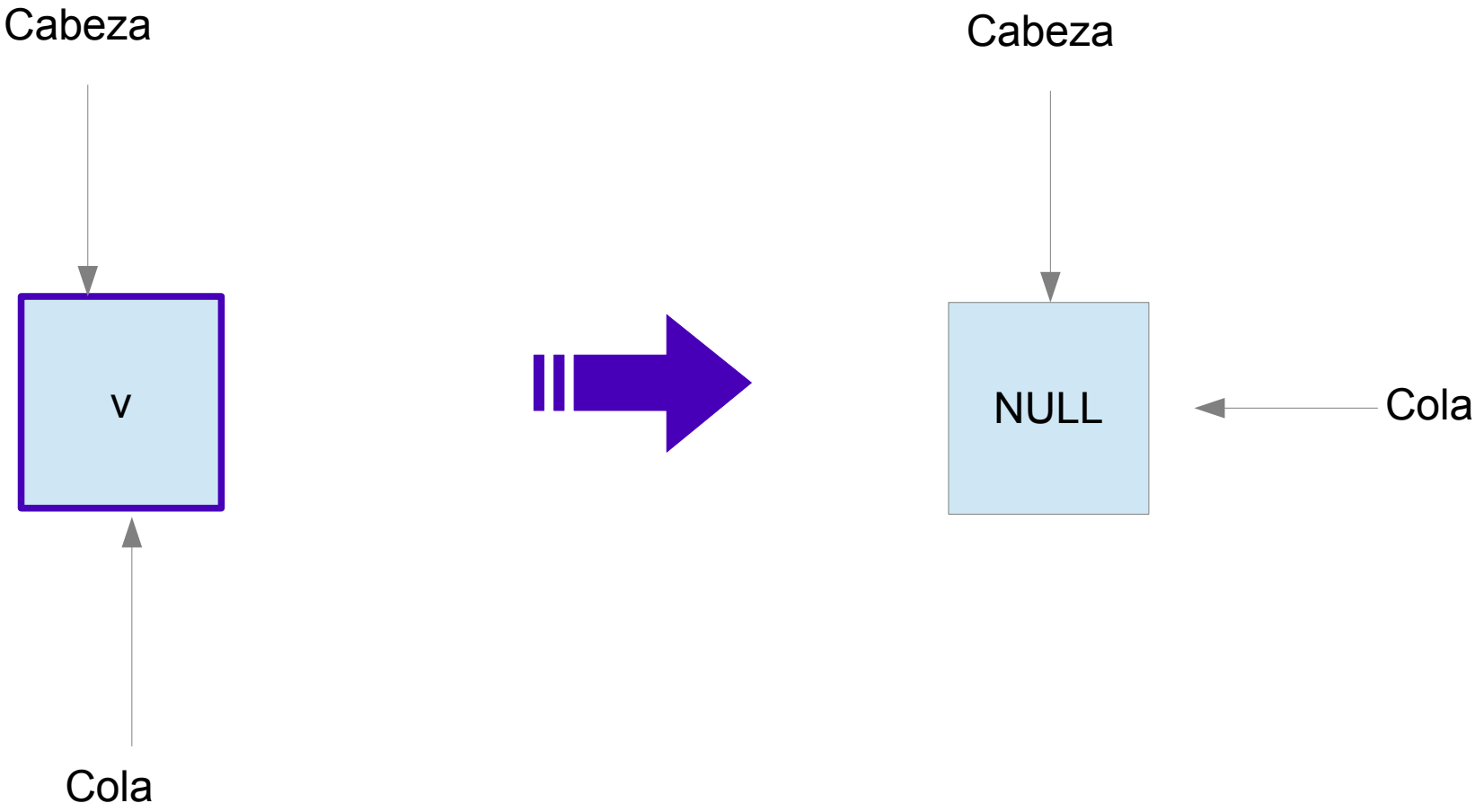
TAD Lista: insertarCola(v)



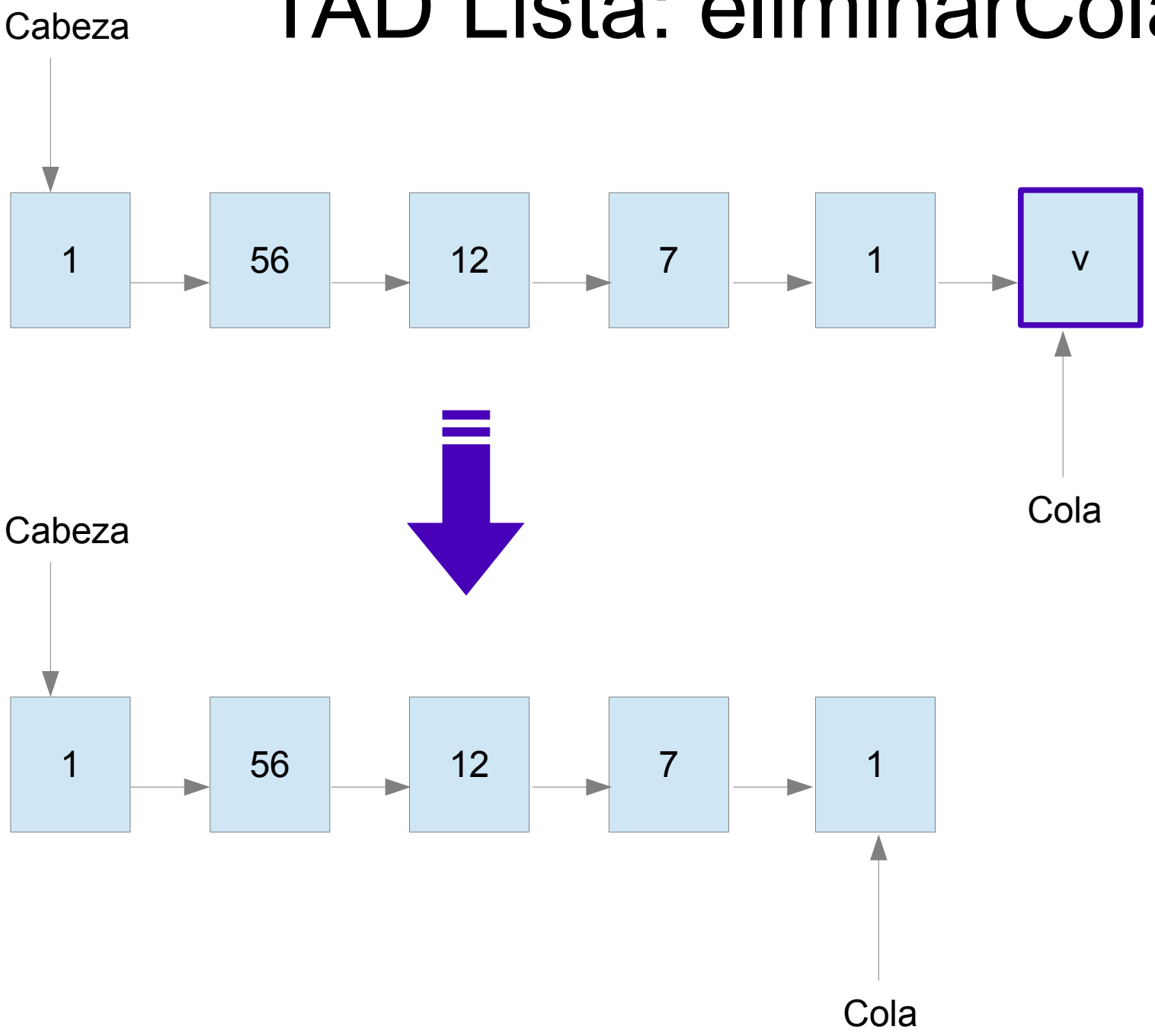
TAD Lista: insertarCola(v)



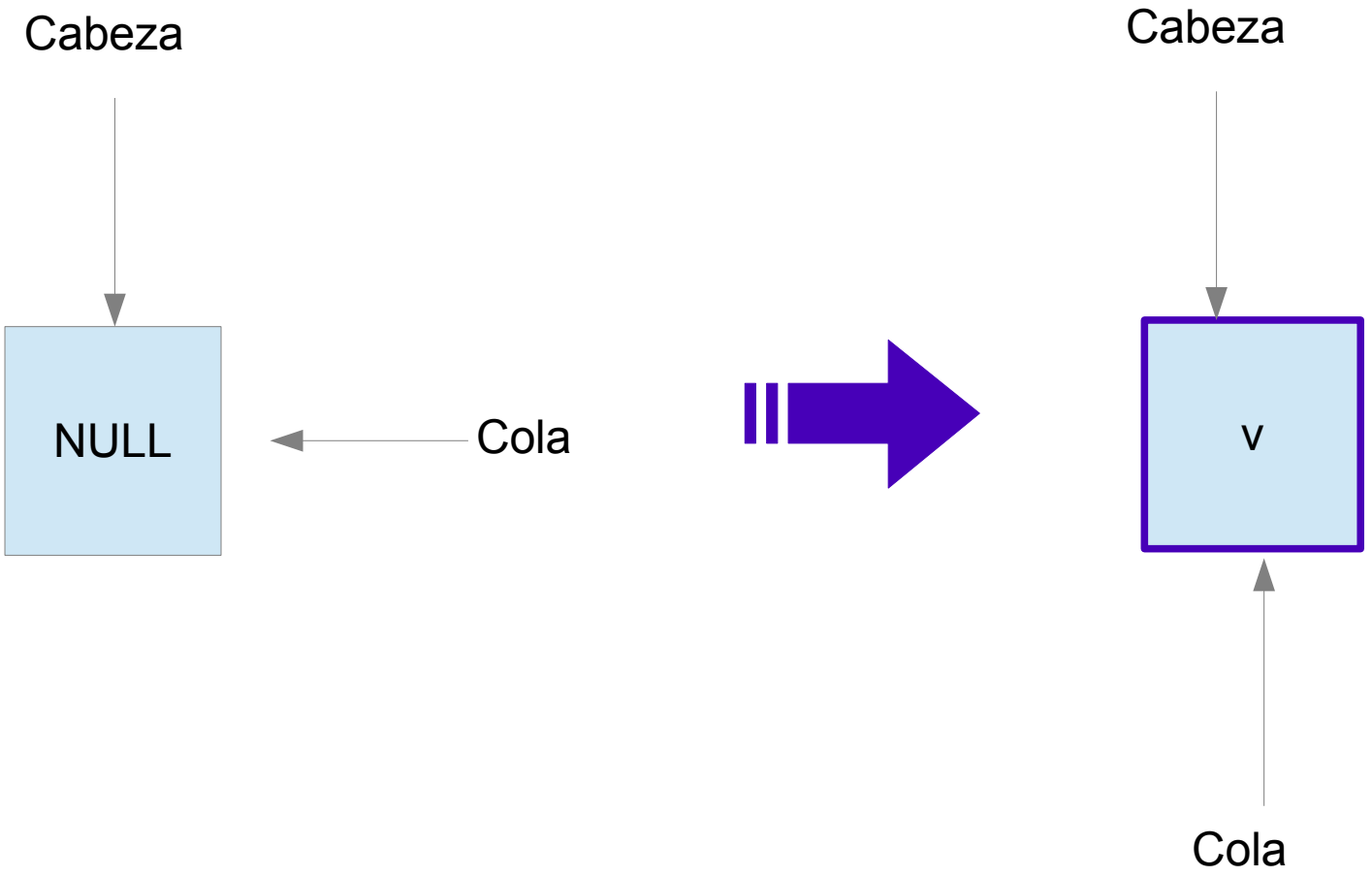
TAD Lista: eliminarCola()



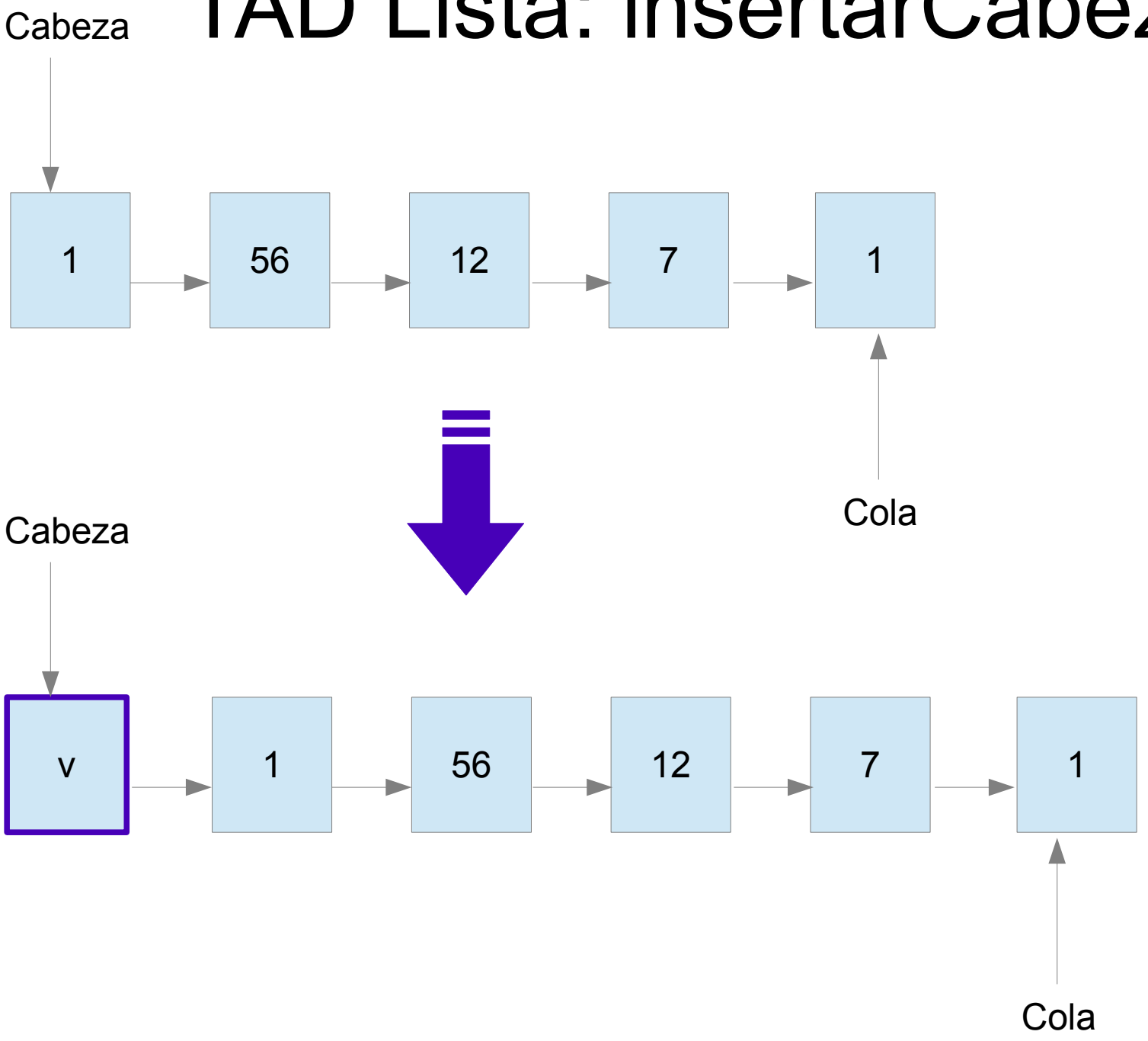
TAD Lista: eliminarCola()



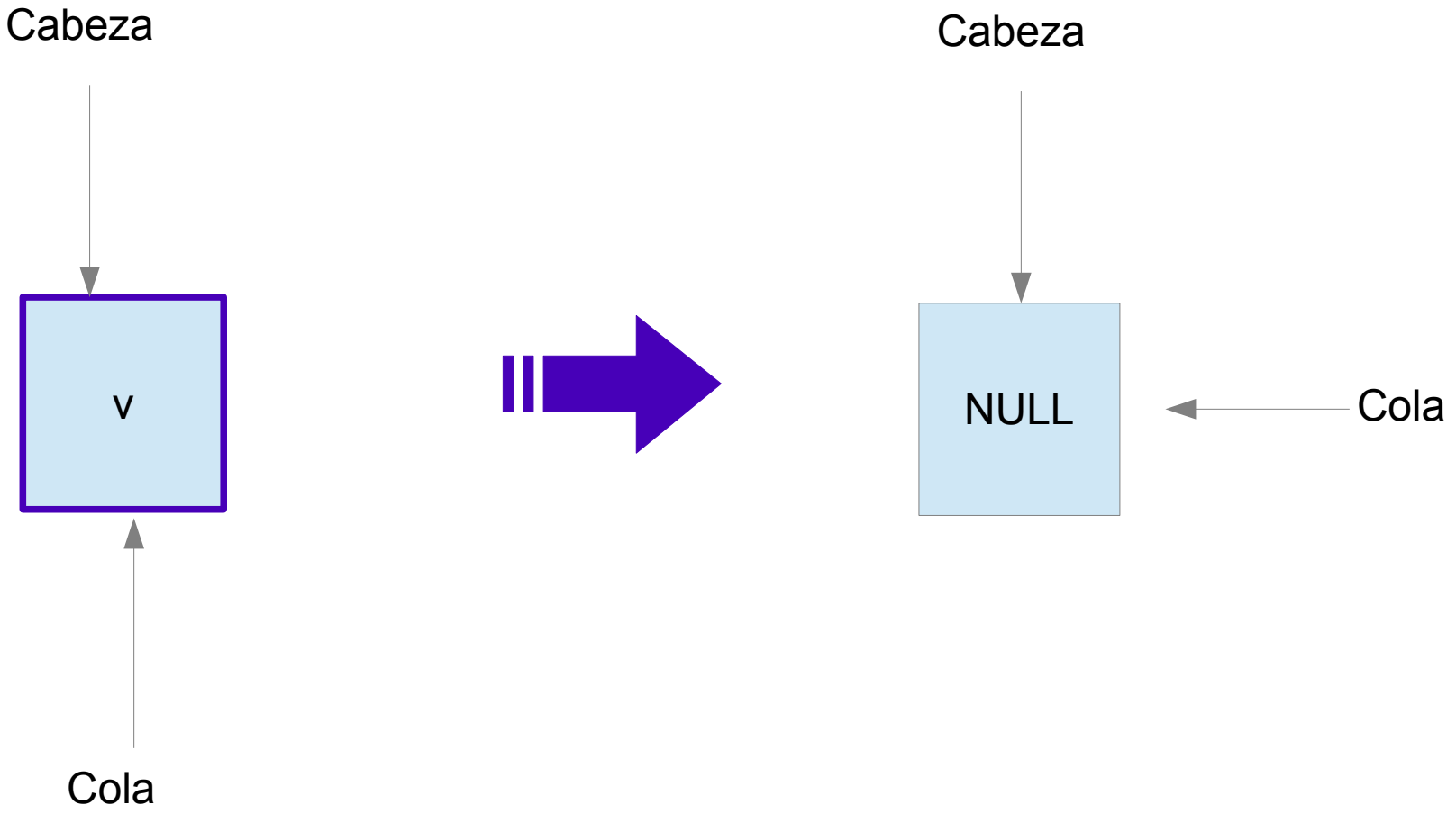
TAD Lista: insertarCabeza(v)



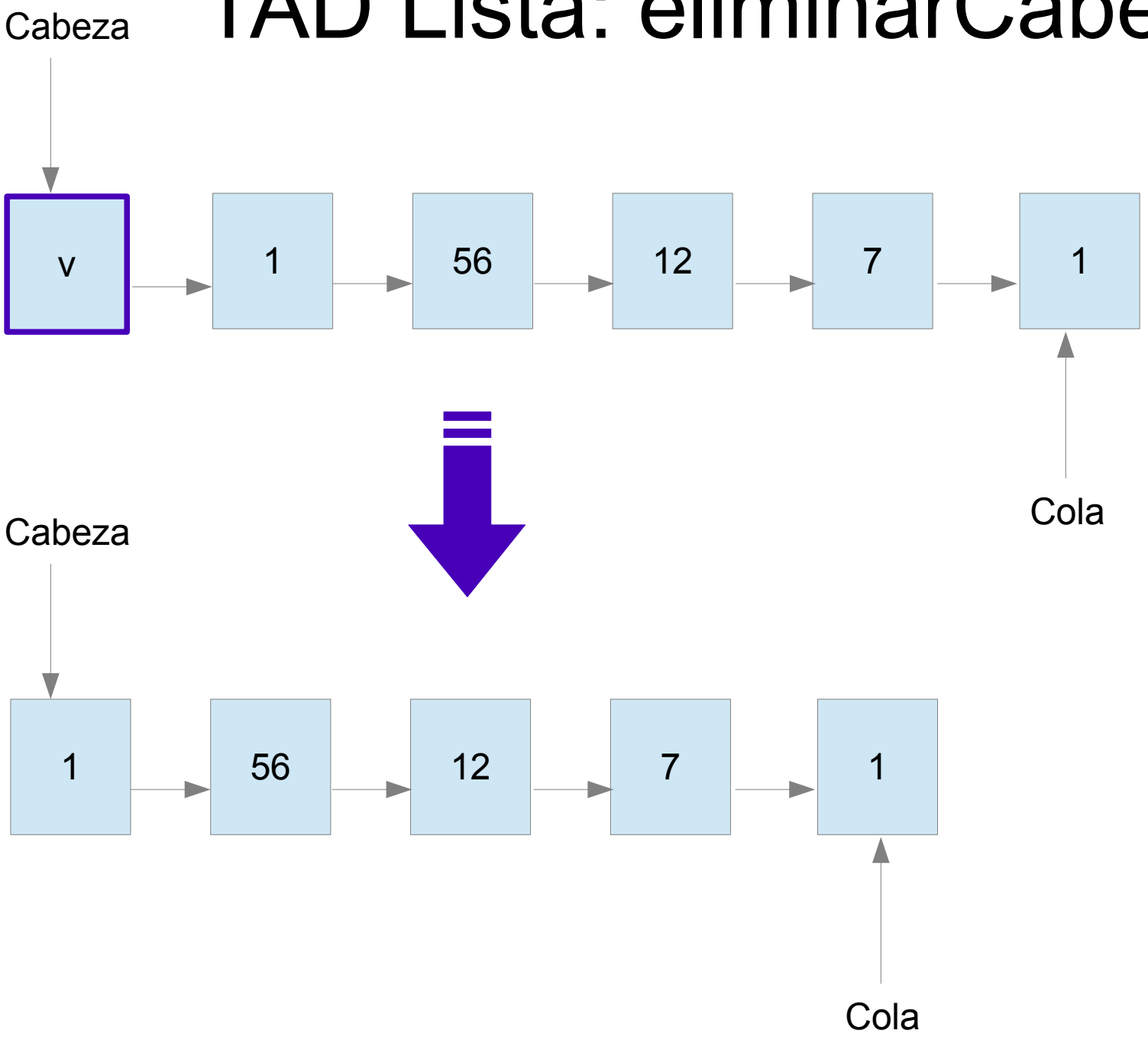
TAD Lista: insertarCabeza(v)



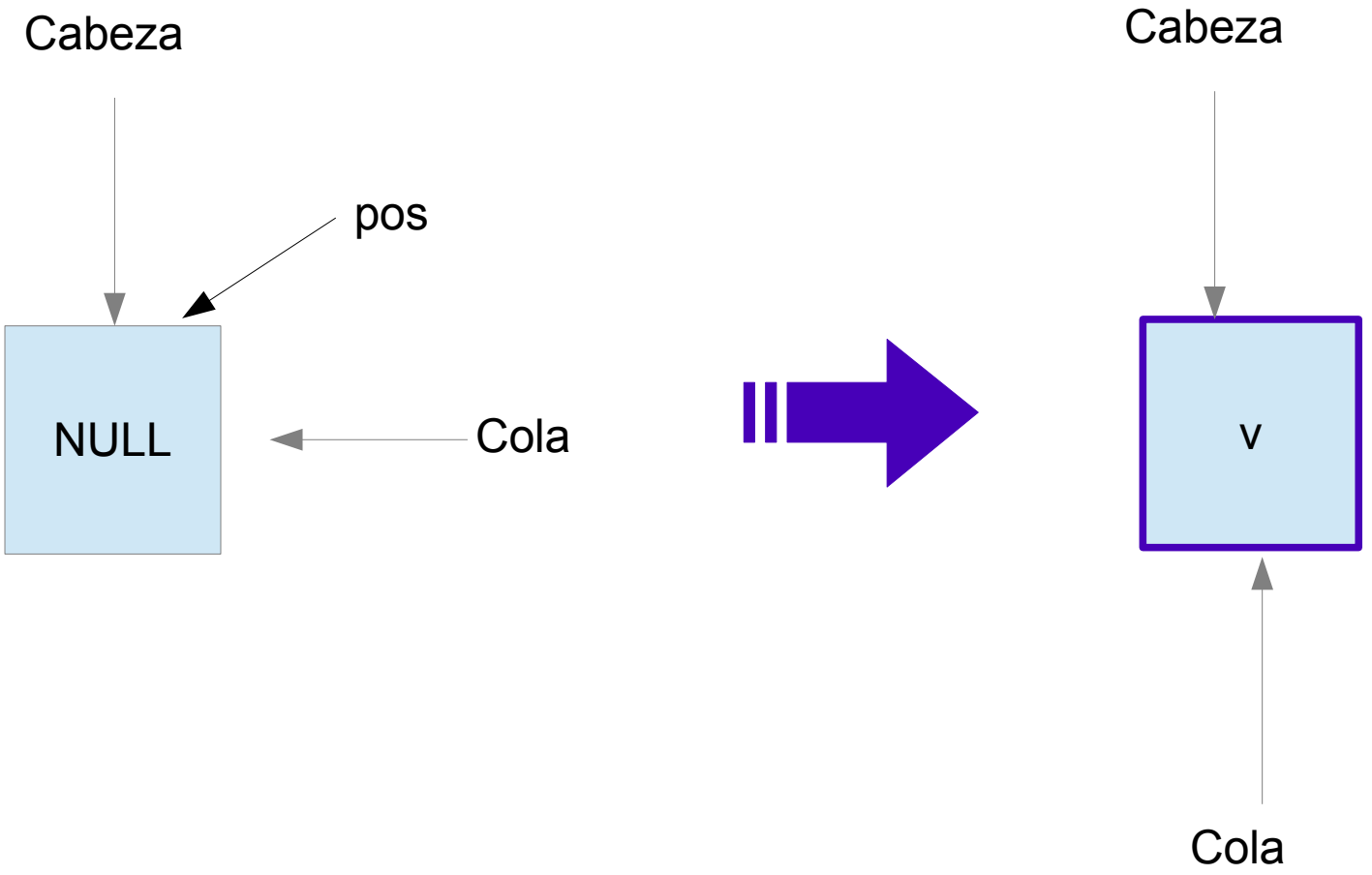
TAD Lista: eliminarCabeza()



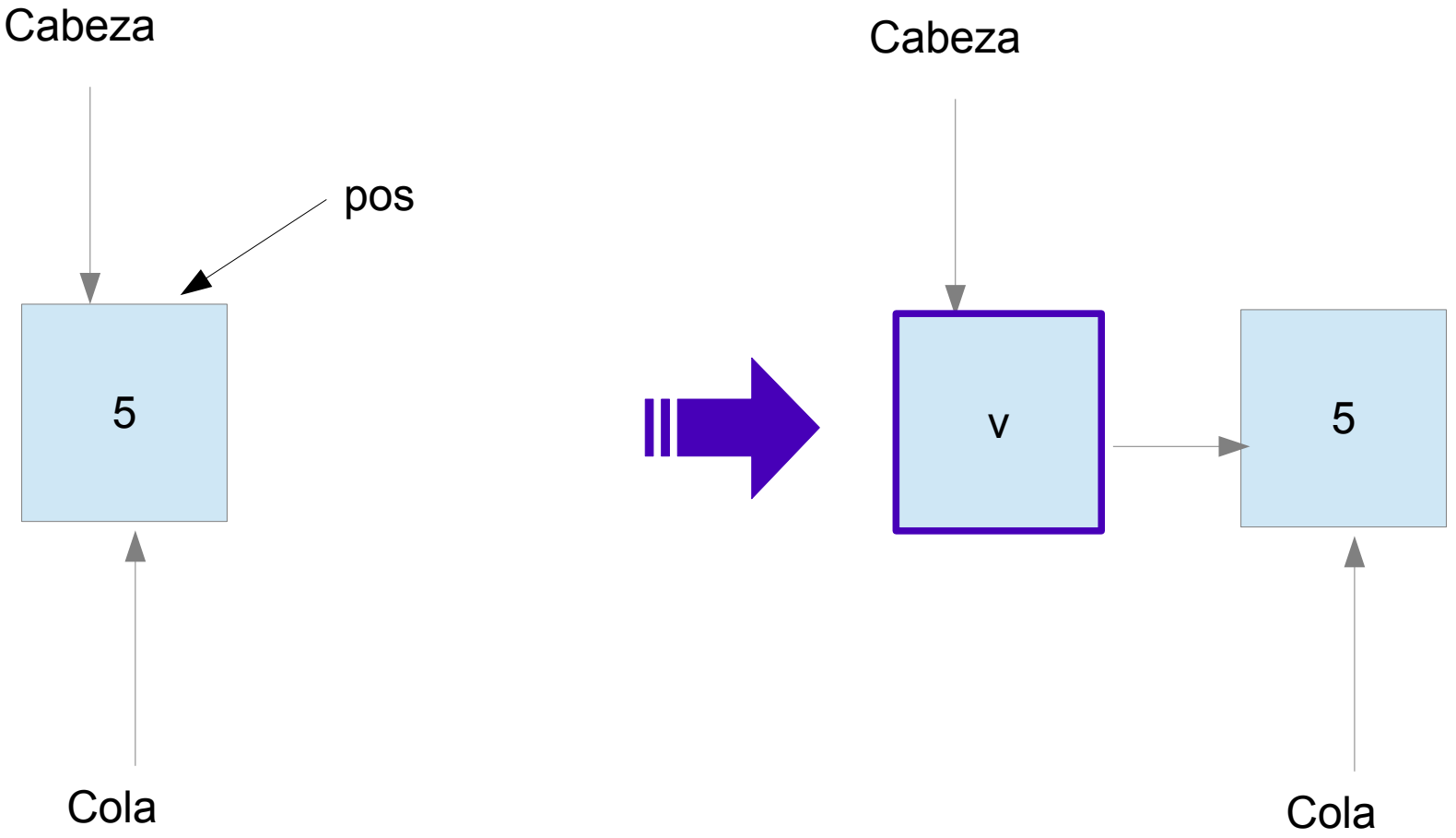
TAD Lista: eliminarCabeza()



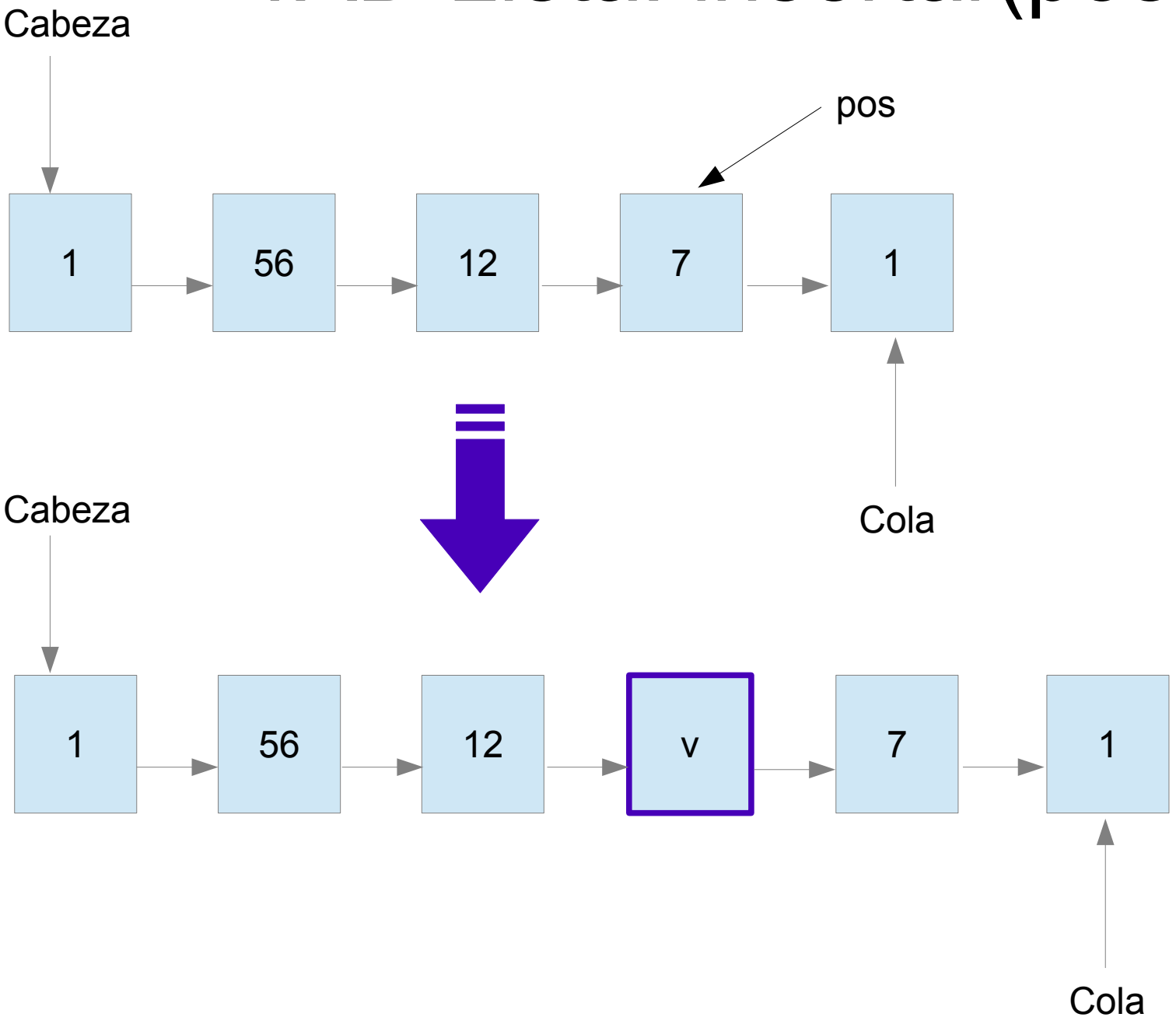
TAD Lista: insertar(pos, v)



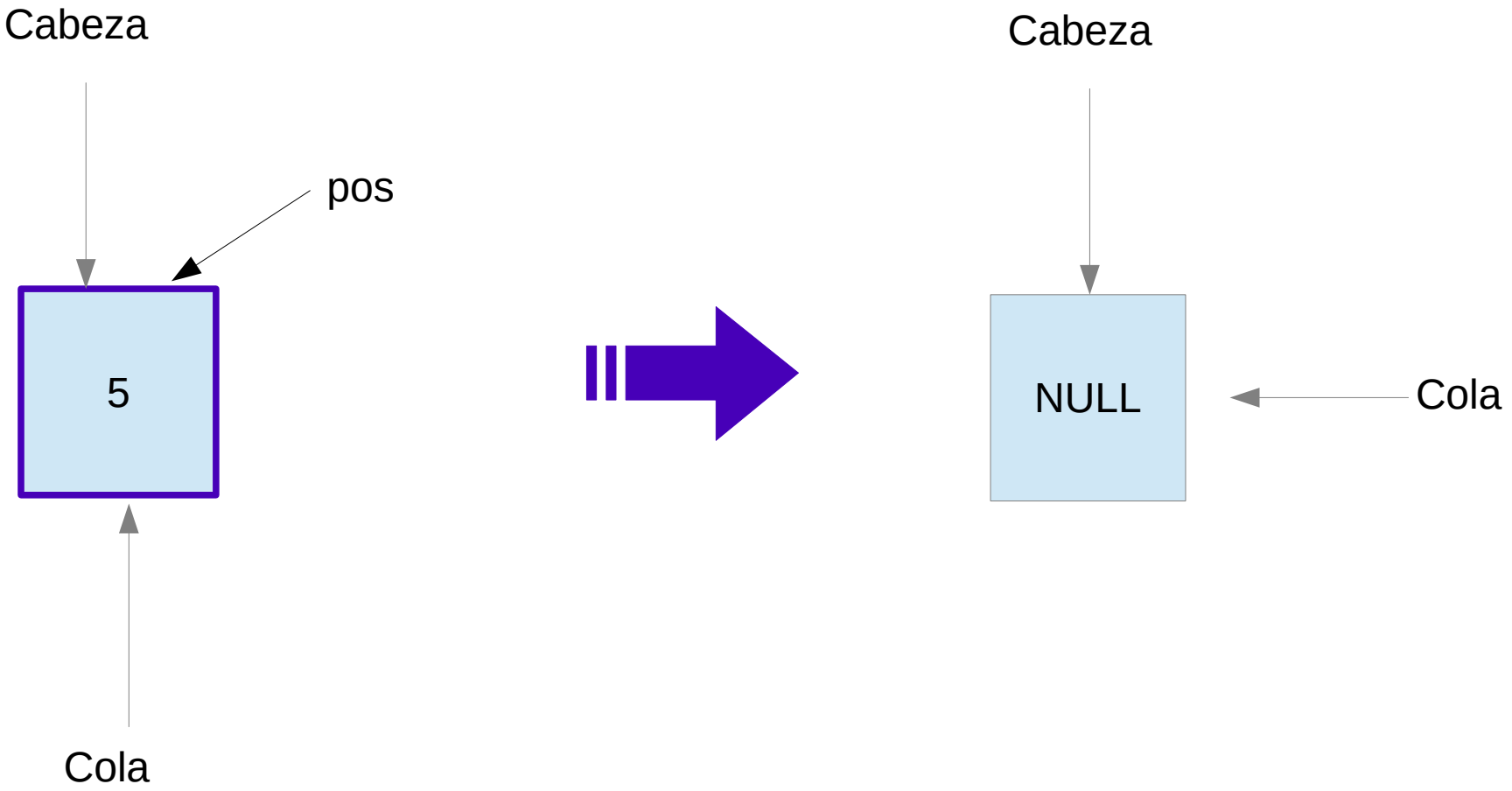
TAD Lista: insertar(pos, v)



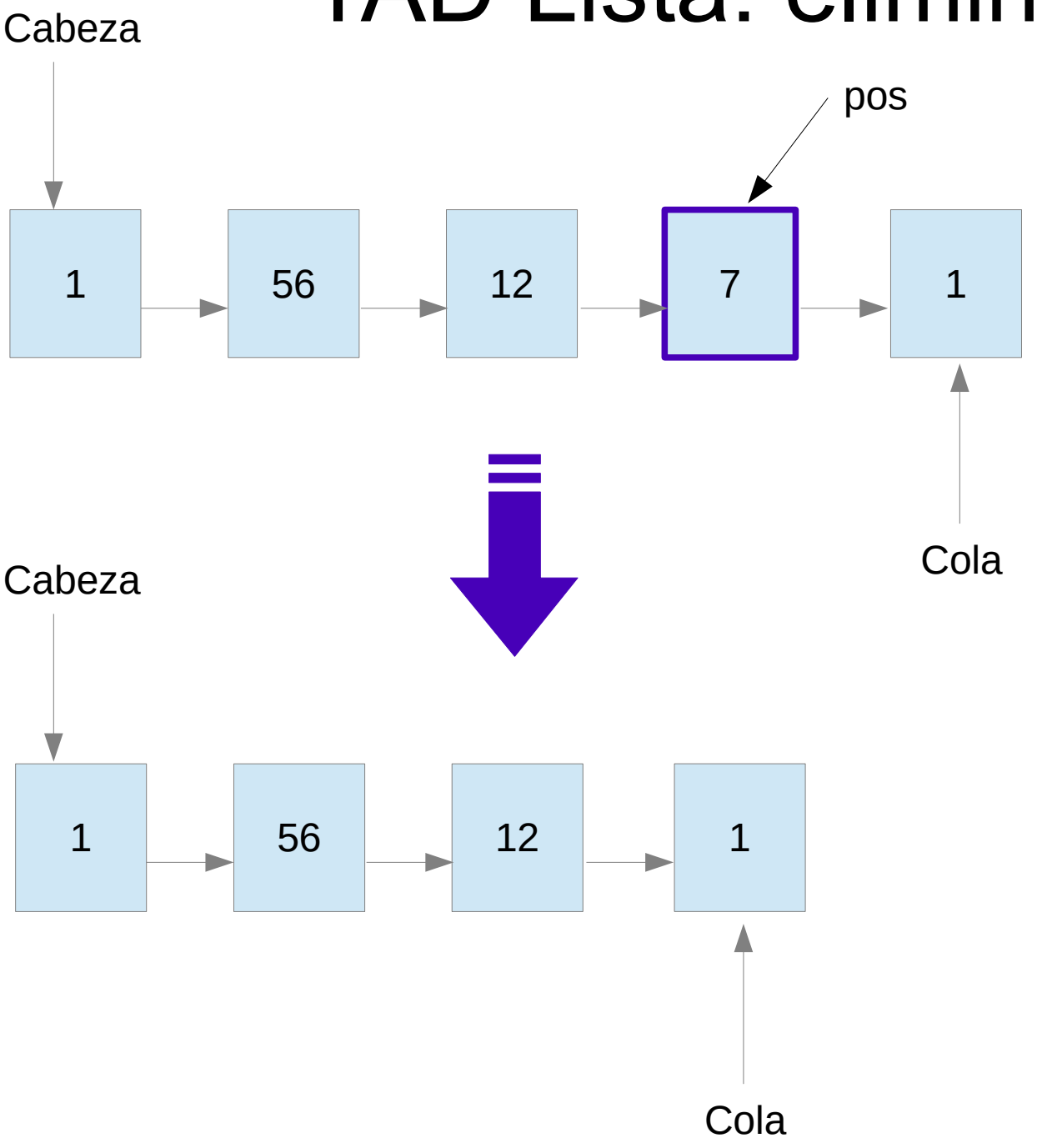
TAD Lista: insertar(pos, v)



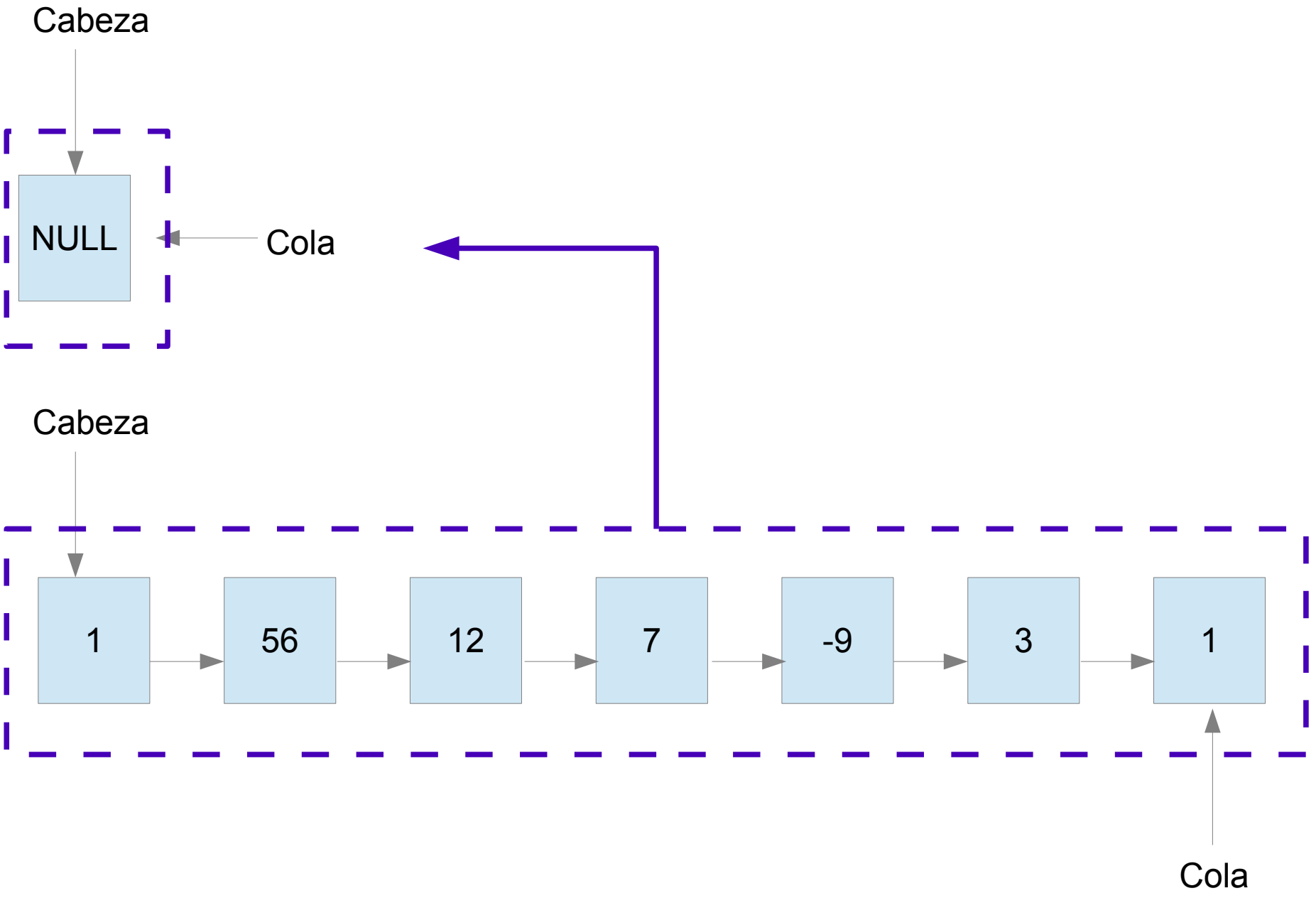
TAD Lista: eliminar(pos)



TAD Lista: eliminar(pos)



TAD Lista: vaciar()



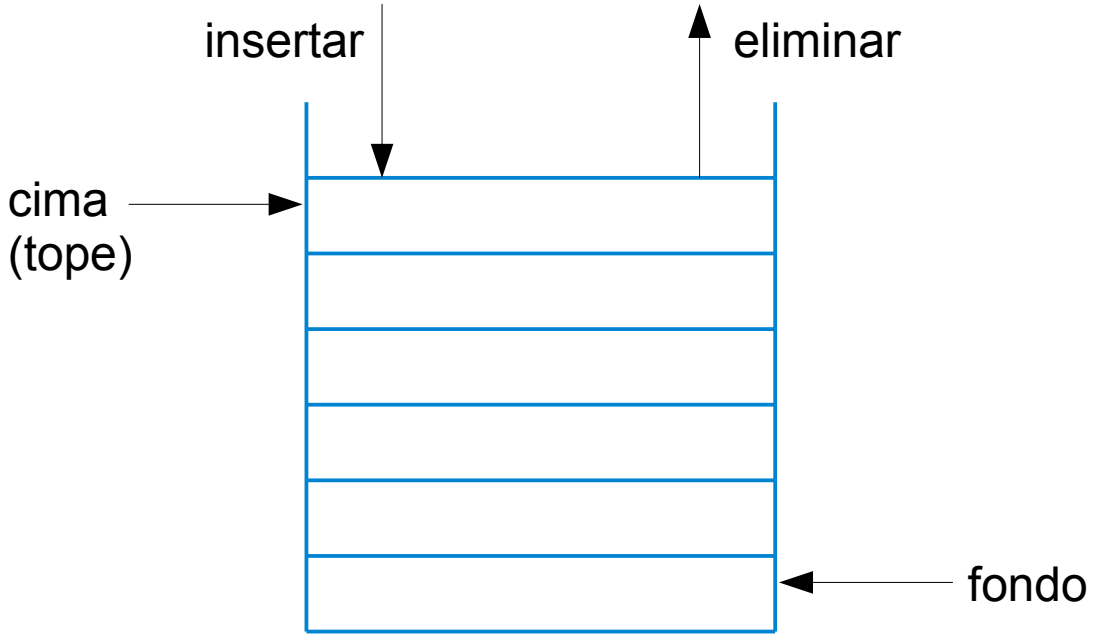
TAD Pila

Pila

- Colección ordenada de elementos con restricciones de acceso.
- Sus elementos solo pueden accederse por un único lugar, el tope o cima de la pila.

Pila

- Entradas deben ser eliminadas en el orden inverso al que se situaron en la pila.
- “último en entrar, primero en salir” → estructura de datos LIFO (*last-in, first-out*).



Pila

Operaciones principales:

- Insertar (*push*):

  Añade un elemento y lo ubica en el tope de la pila.

- Eliminar (*pop*):

  Extrae el elemento en el tope y lo retira de la pila.

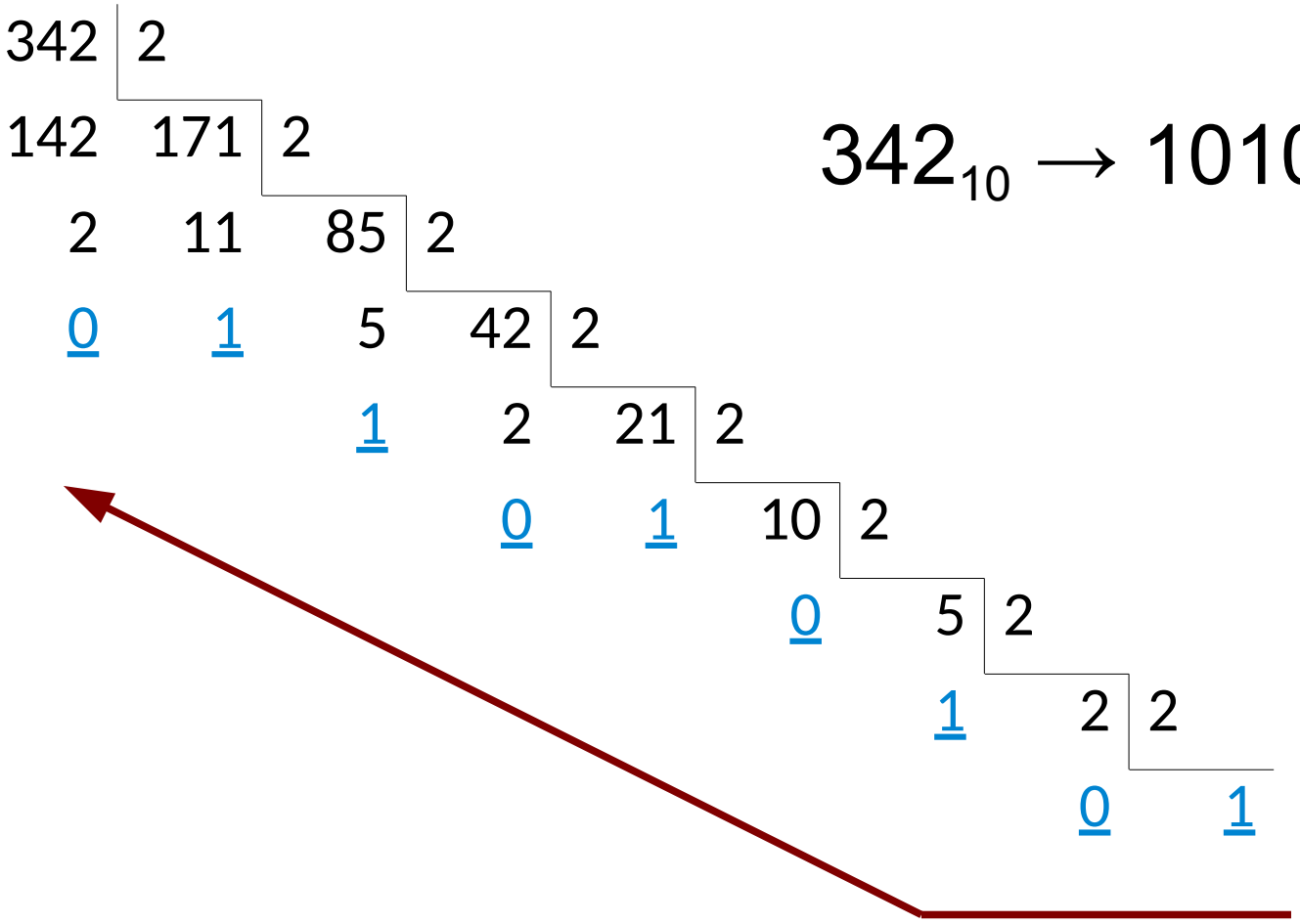
Pila

- Ejemplo de uso:
  conversión de decimal a binario.

$$342_{10} \rightarrow ?_{2}$$

Pila

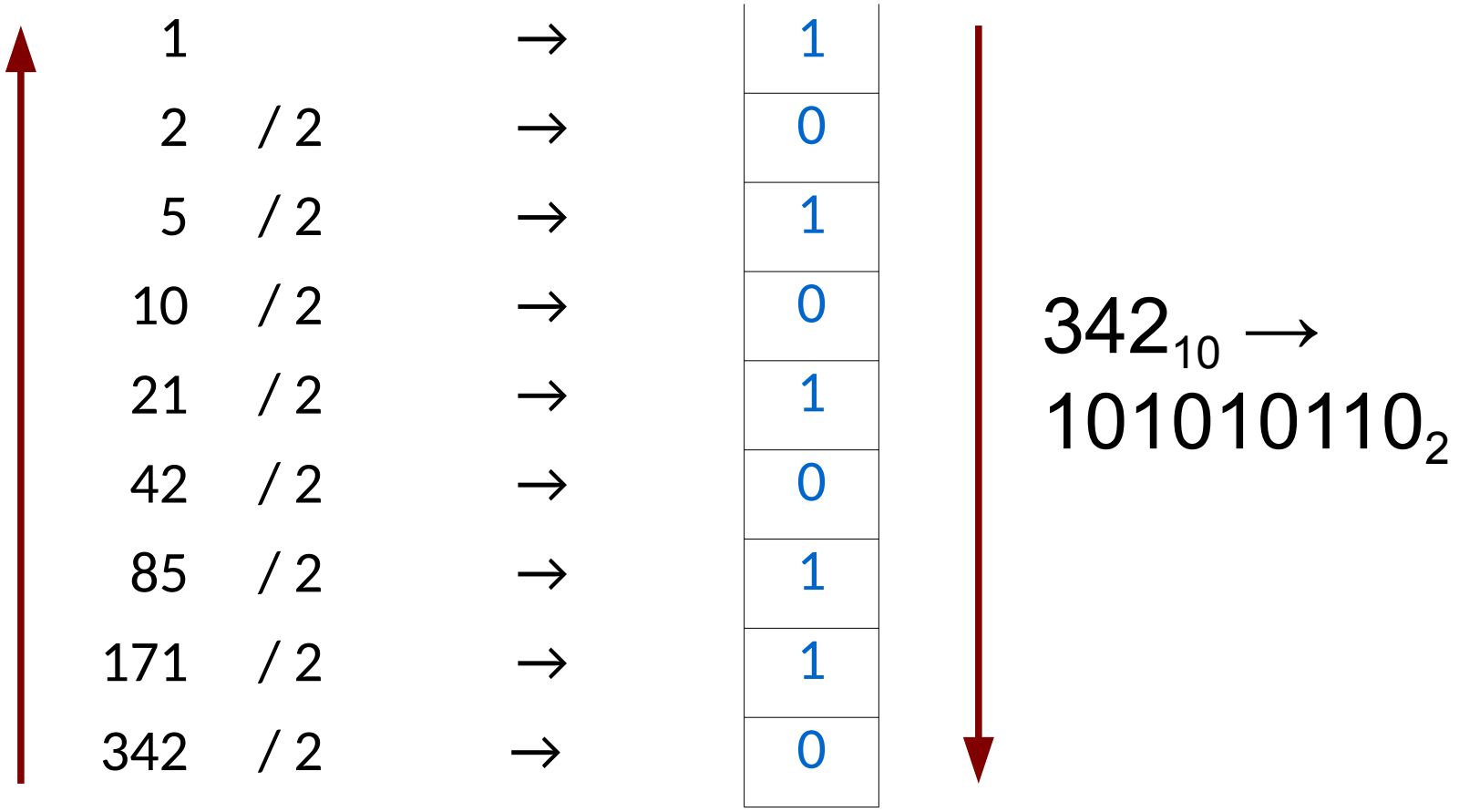
- Ejemplo de uso:
  conversión de decimal a binario.



$342_{10} \rightarrow 101010110_2$

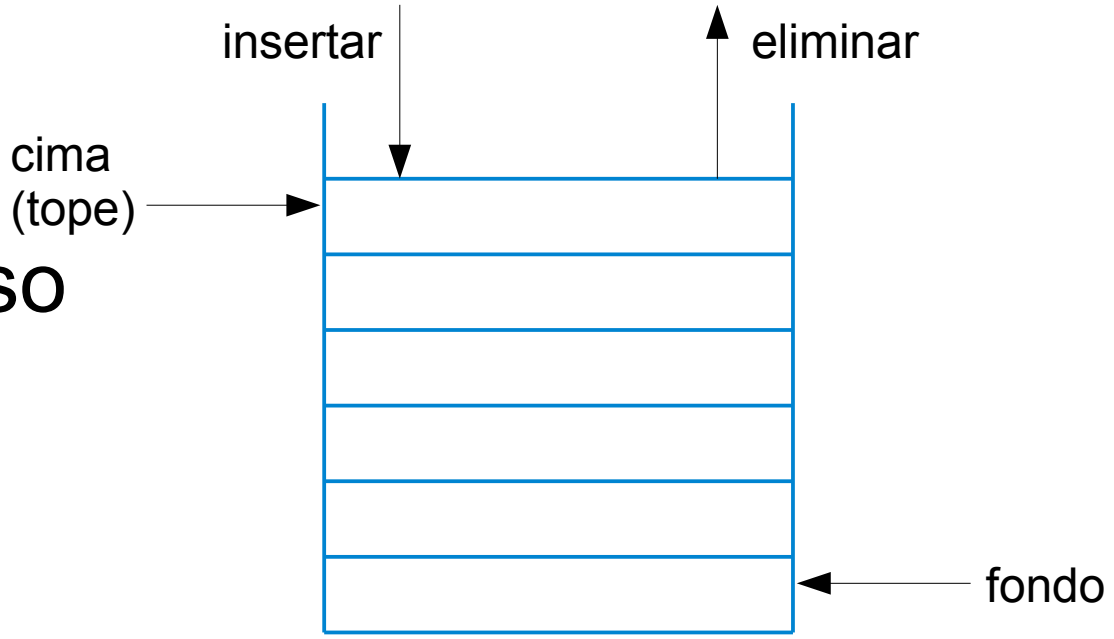
Pila

- Ejemplo de uso:
  conversión de decimal a binario.



TAD Pila

- Secuencia finita de datos.
  - Acceso sólo al elemento tope.
- Sin recorridos, acceso aleatorio restringido.
- Algoritmos:
  - Insertar, eliminar.
  - Vacía, tope.



¿Estado? ¿Interfaz?

TAD Pila

TAD Pila

Conjunto mínimo de datos:

Comportamiento (operaciones) del objeto:

TAD Pila

TAD Pila

Conjunto mínimo de datos:
- tope, plantilla, representa el tope (elemento accesible).

Comportamiento (operaciones) del objeto:
- esVacia(), indica si la pila está vacía.
- tope(), retorna el elemento en el tope.
- insertar(v), inserta v en la pila.
- eliminar(), elimina el elemento en el tope.
- vaciar(), elimina todos los elementos de la pila.

TAD Pila

- esVacia
  -> verificar si tope es nulo o no
- insertar
  -> crear nuevo nodo, poner dato ahí
  siguiente del nuevo nodo es el tope actual
  tope se actualiza al nuevo nodo

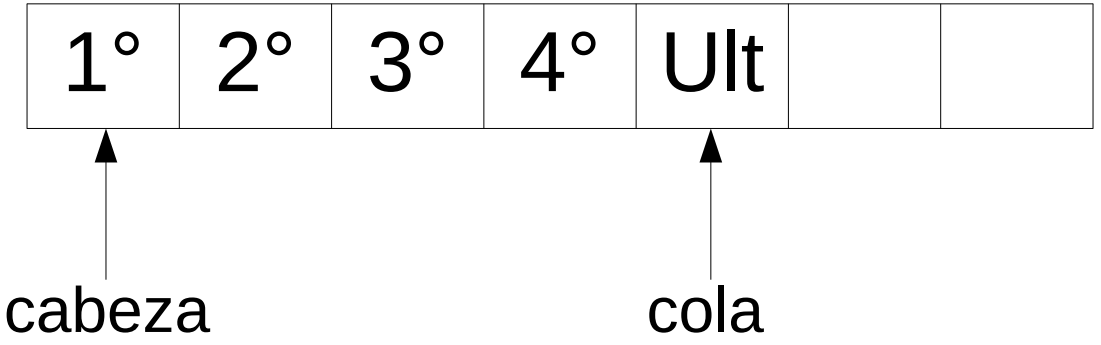
TAD Pila

- eliminar
  -> ubicar tope actual con un temporal
  tope se actualiza al siguiente del tope
  temporal se elimina de la memoria
- tope
  -> retorna tope actual
- vaciar
  -> eliminar de la pila hasta que quede vacía

TAD Cola

Cola

- Colección de elementos almacenados en una lista con restricciones de acceso.
- Los elementos sólo pueden ser insertados en la cola (final) de la lista, y sólo pueden ser eliminados por la cabeza (inicio o frente) de la lista .



Cola

- Entradas deben ser eliminadas en el mismo orden en el que se situaron en la cola.
- “primero en entrar, primero en salir” → estructura de datos FIFO (first-in, first-out).
- Ejemplos:
  - Atención a clientes en un almacén.
  - Gestión de trabajos en una impresora.

https://www.citrix.com/blogs/2013/06/28/efficient-queuing-but-where/

Cola

Operaciones principales:

- Insertar (*push*):

  Añadir un elemento por el extremo final de la cola.

- Eliminar (*pop*):

  Extraer el elemento ubicado en el extremo inicial de la cola .

Cola

- Ejemplo de uso:
  simulación de la atención de clientes en un banco con un solo cajero.

  Al banco llega un cliente cada `x` segundos.

  Un cajero atiende un cliente durante `y` segundos.

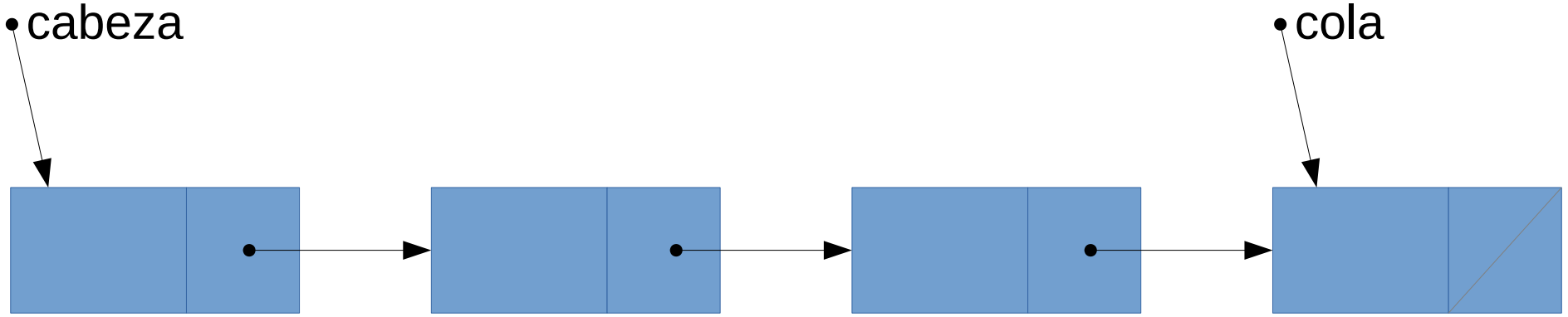
Cola

```
Segundos = 0
mientras (Segundos < 3600)
    si (Segundos % x == 0)
        cola.insertar(cliente)
    si (Segundos % y == 0)
        cola.eliminar()
    Segundos++
fin_mientras
```

TAD Cola

- Secuencia finita de datos.
  - Inserción por un extremo, eliminación por el otro.
- Sin recorridos, acceso aleatorio restringido
- Algoritmos:
  - Insertar, eliminar.
  - Vacía, cabeza.

¿Estado? ¿Interfaz?



TAD Cola

TAD Cola

Conjunto mínimo de datos:

Comportamiento (operaciones) del objeto:

TAD Cola

TAD Cola

Conjunto mínimo de datos:
- cabeza, plantilla, representa el extremo inicial.
- cola, plantilla, representa el extremo final.

Comportamiento (operaciones) del objeto:
- esVacia(), indica si la cola está vacía.
- cabeza(), retorna el elemento en la cabeza.
- insertar(v), inserta v en la cola.
- eliminar(), elimina el elemento en la cabeza.
- vaciar(), elimina todos los elementos en la cola.

TAD Cola

- esVacia
  -> verificar si cabeza y cola son nulos o no
- insertar
  -> crear nuevo nodo, poner dato ahí
  siguiente de la cola actual es el nuevo nodo
  cola se actualiza al nuevo nodo

TAD Cola

- eliminar
  -> ubicar cabeza actual con un temporal
  cabeza se actualiza al siguiente de la cabeza
  temporal se elimina de la memoria
- cabeza
  -> retorna cabeza actual
- vaciar
  -> eliminar de la cola hasta que quede vacía

Tarea

- Revisar:
  - ¿Qué es la STL?
  - ¿Cuáles son sus componentes?
  - ¿Cuáles se utilizan para almacenar datos? ¿Cuáles para recorrer los datos?

Referencias

- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. Introduction to Algorithms, 3rd edition. MIT Press, 2009.
- L. Joyanes Aguilar, I. Zahonero. Algoritmos y estructuras de datos: una perspectiva en C. McGraw-Hill, 2004.