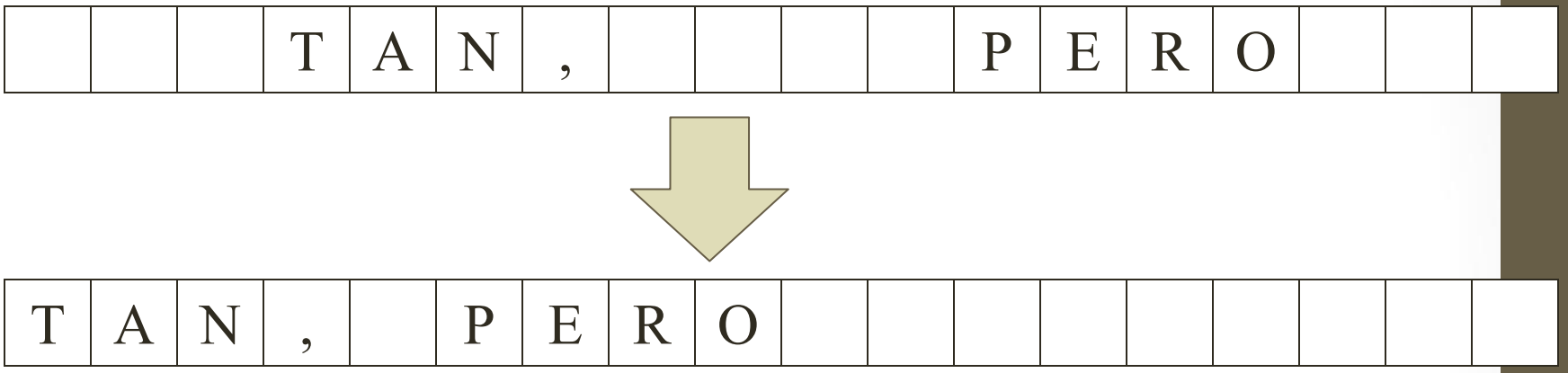
Clase oración-Comprimir

Oracion
S: char[]
<<Constructor>> Oracion(s:String) **<<Comandos>>** comprimir() **<<Consultas>>** longitud(): entero caracterEnPosicion (i: entero): char empieza (ch: caracter): entero estaPalabra (p: Palabra): boolean concatenar (p: Palabra): Oracion toString(): String

Comprimir



Para diseñar el algoritmo elegimos un ejemplo significativo, tiene varios espacios al principio, al final y entre las palabras.

Casos de Prueba

Sin embargo, para verificar la solución debemos considerar otros ejemplos:

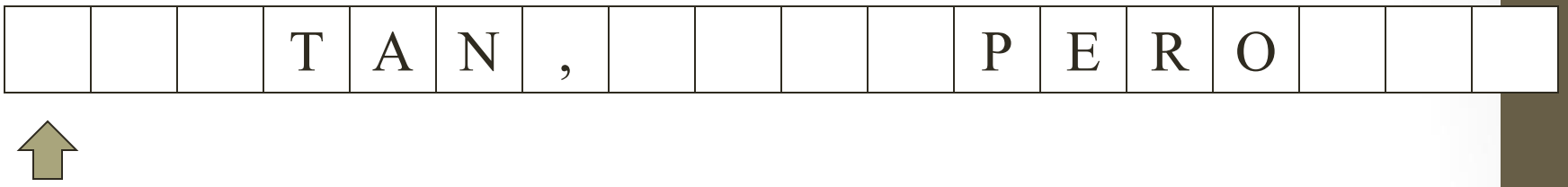
Caso de Prueba N°1:

			T	A	N	,					P	E	R	O			

Caso de Prueba N°2:

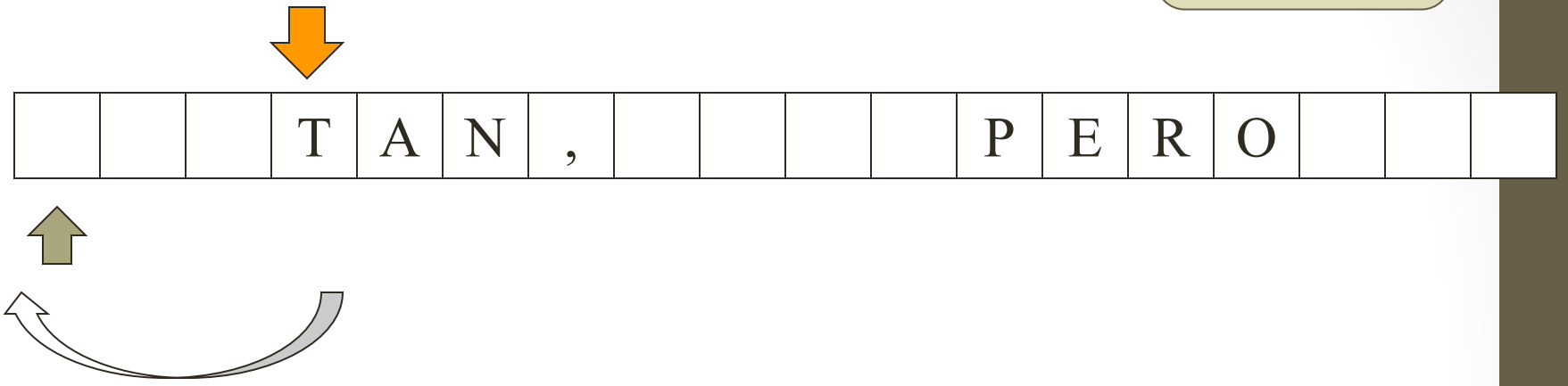
T	A	N	,				P	E	R	O						

Comprimir



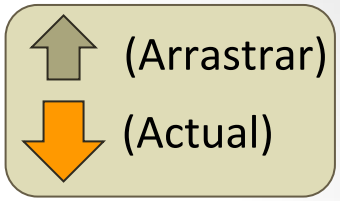
Asumimos que la oración respeta cierta sintaxis, toda palabra puede estar seguida por un signo y todo signo está seguido de al menos un espacio.

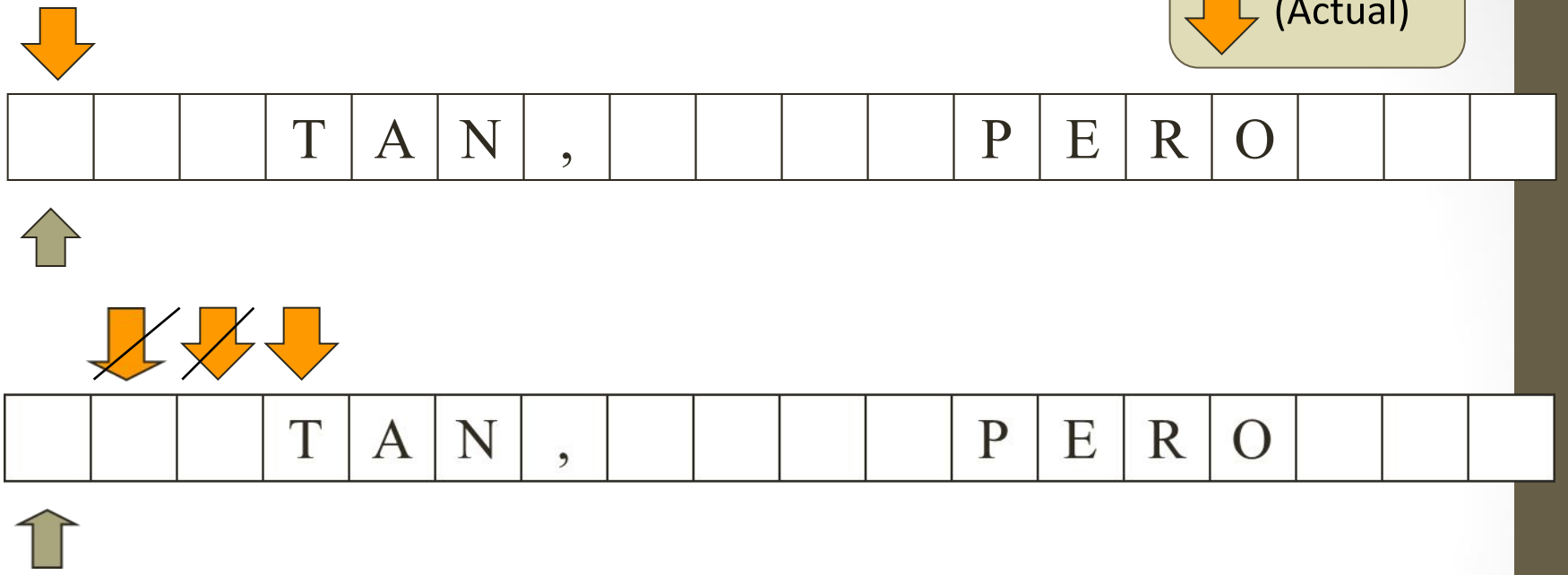
Comprimir



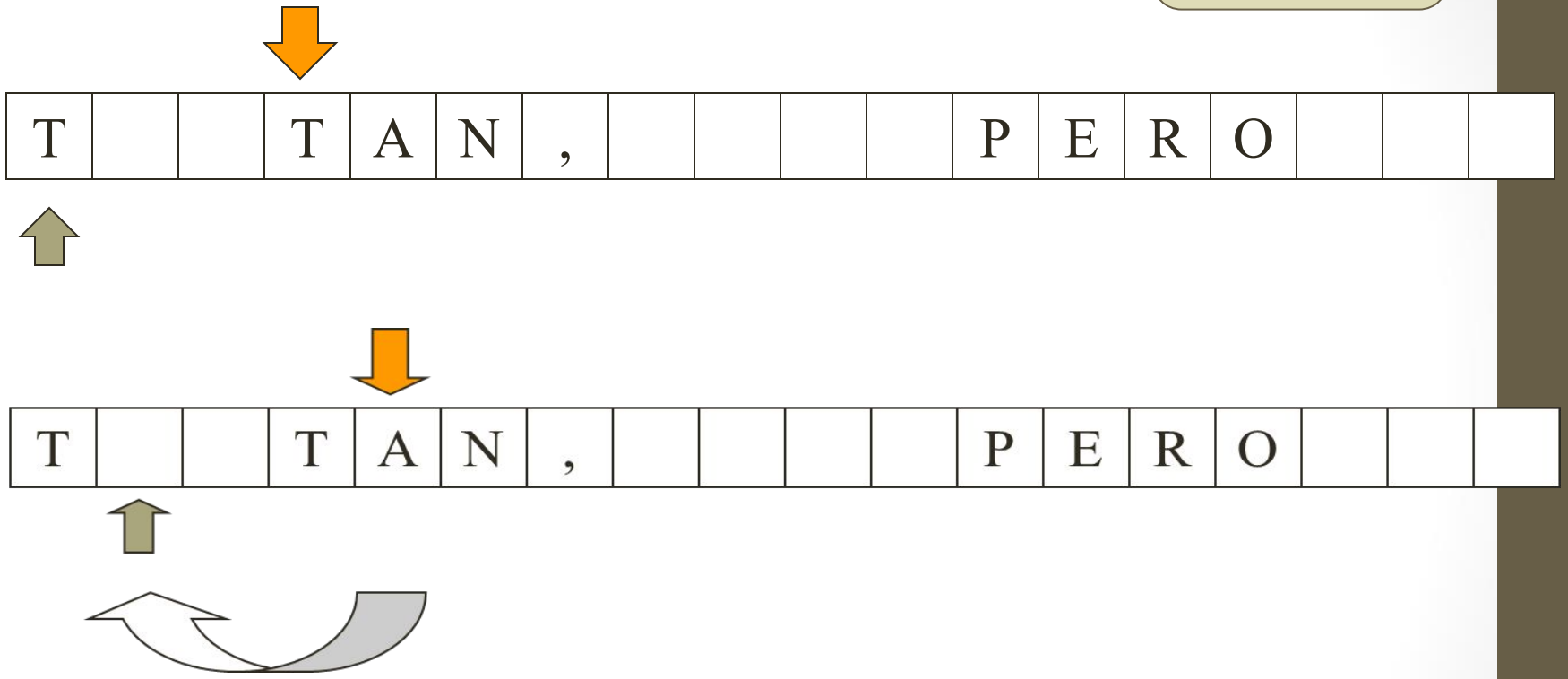
Necesitamos dos variables, una indica la posición del elemento que hay que “arrastrar” y otra la posición a la que hay que “arrastrarlo”.

Comprimir

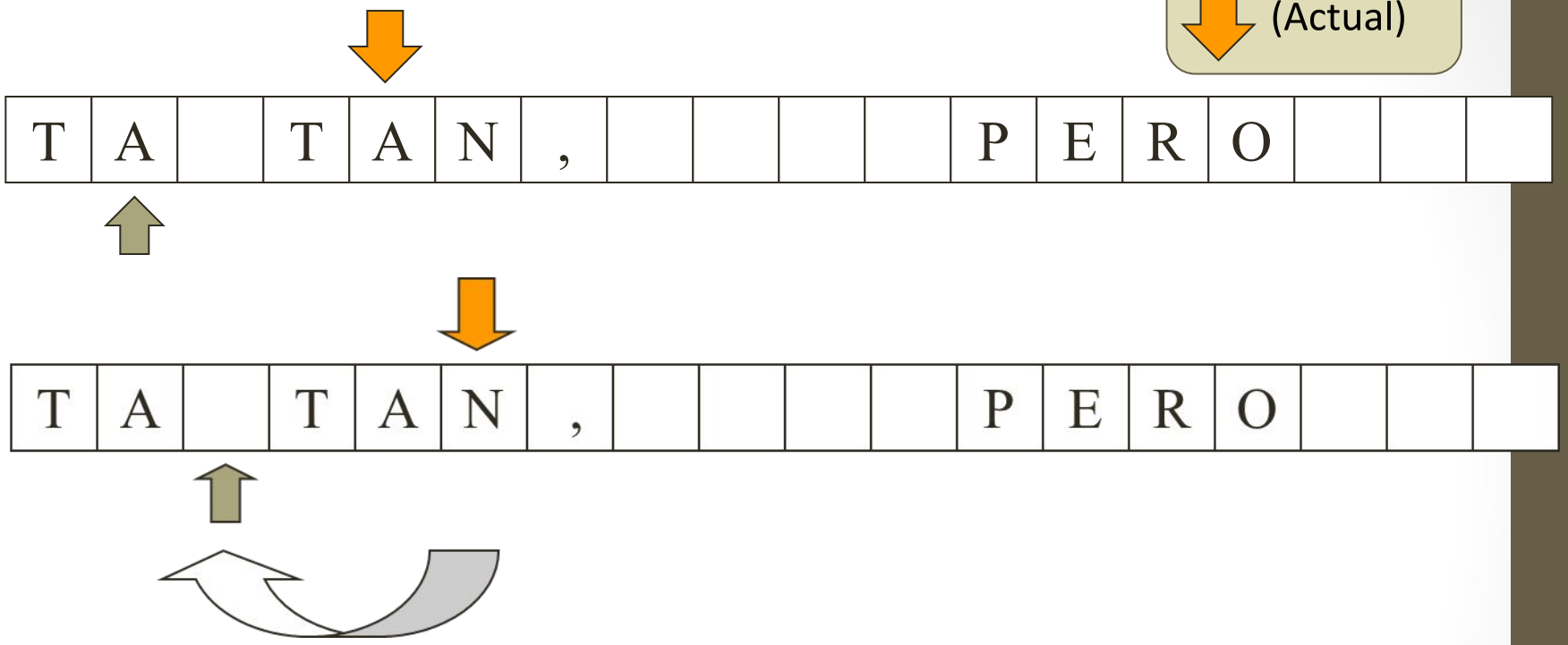




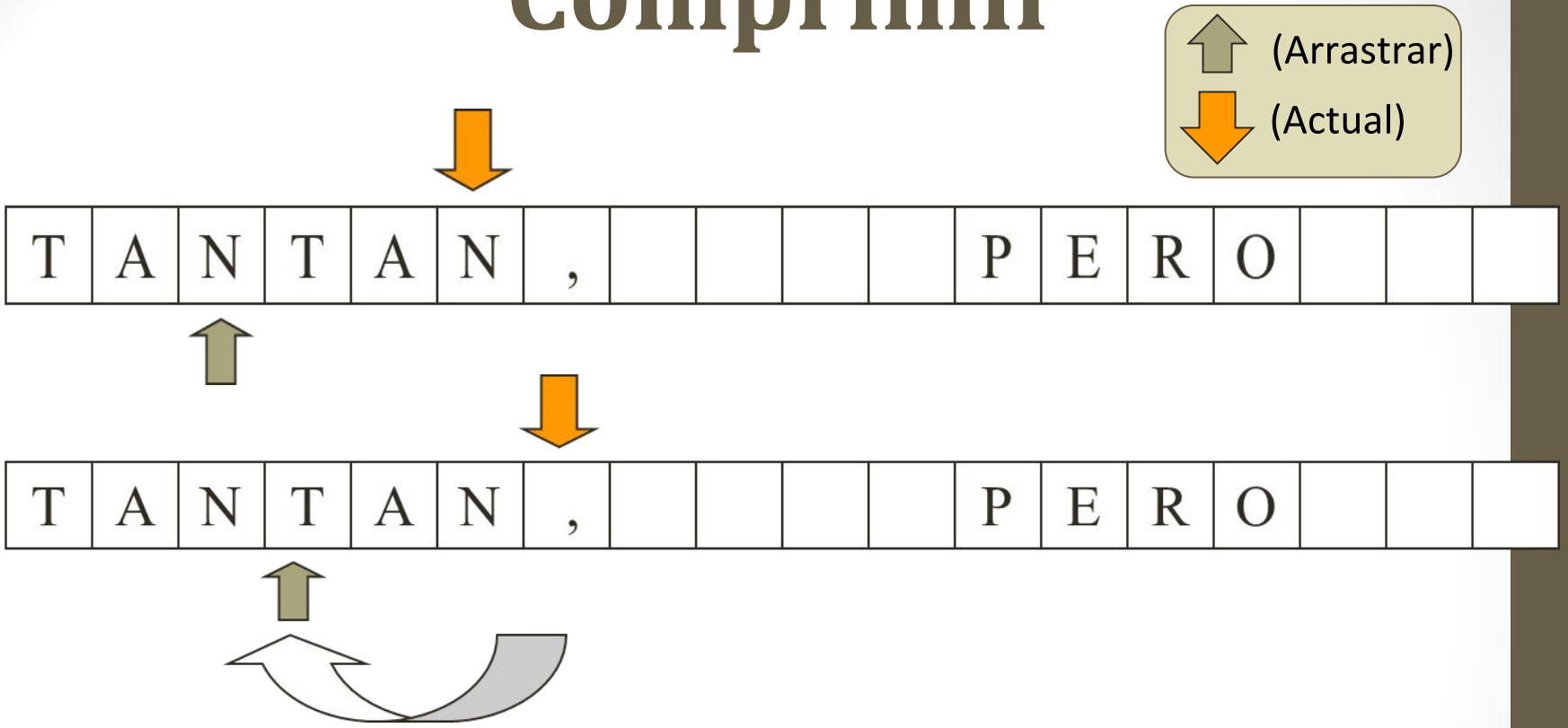
Comprimir



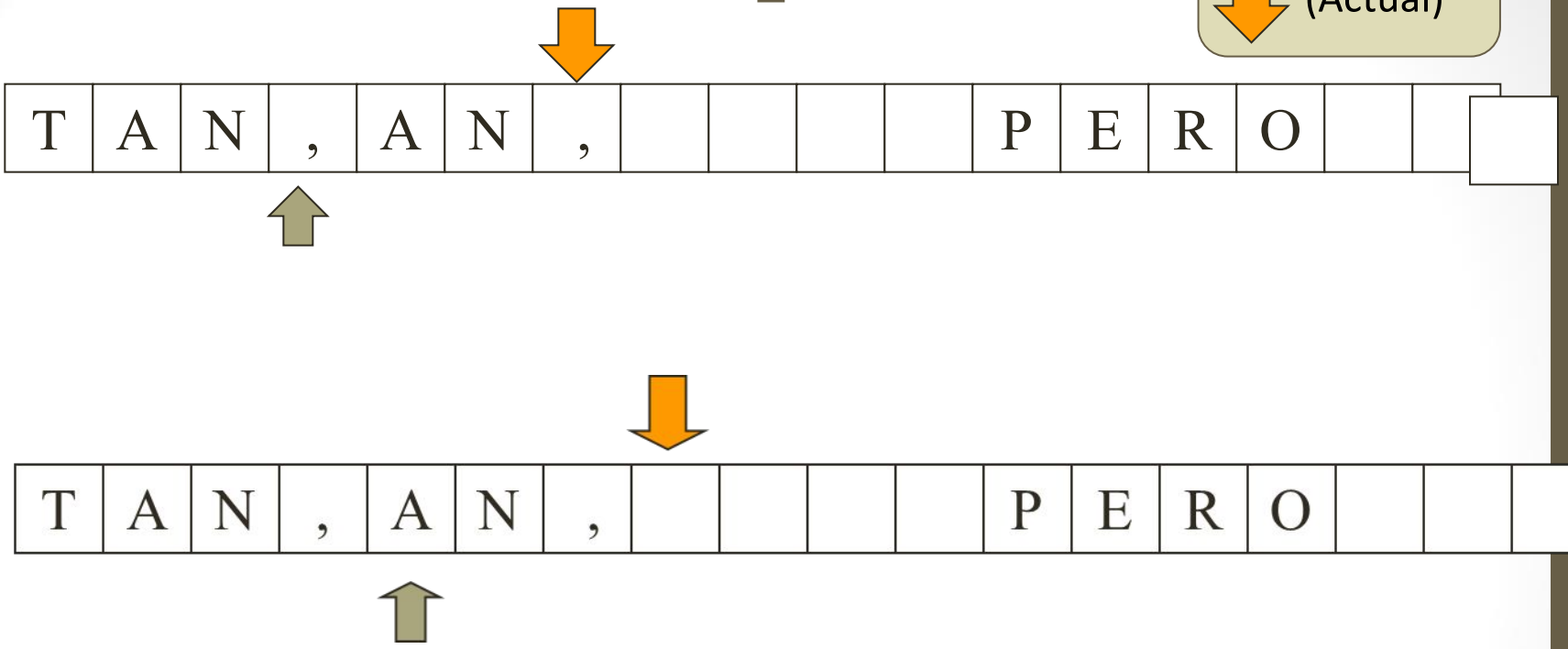
Comprimir



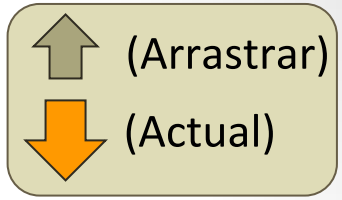
Comprimir

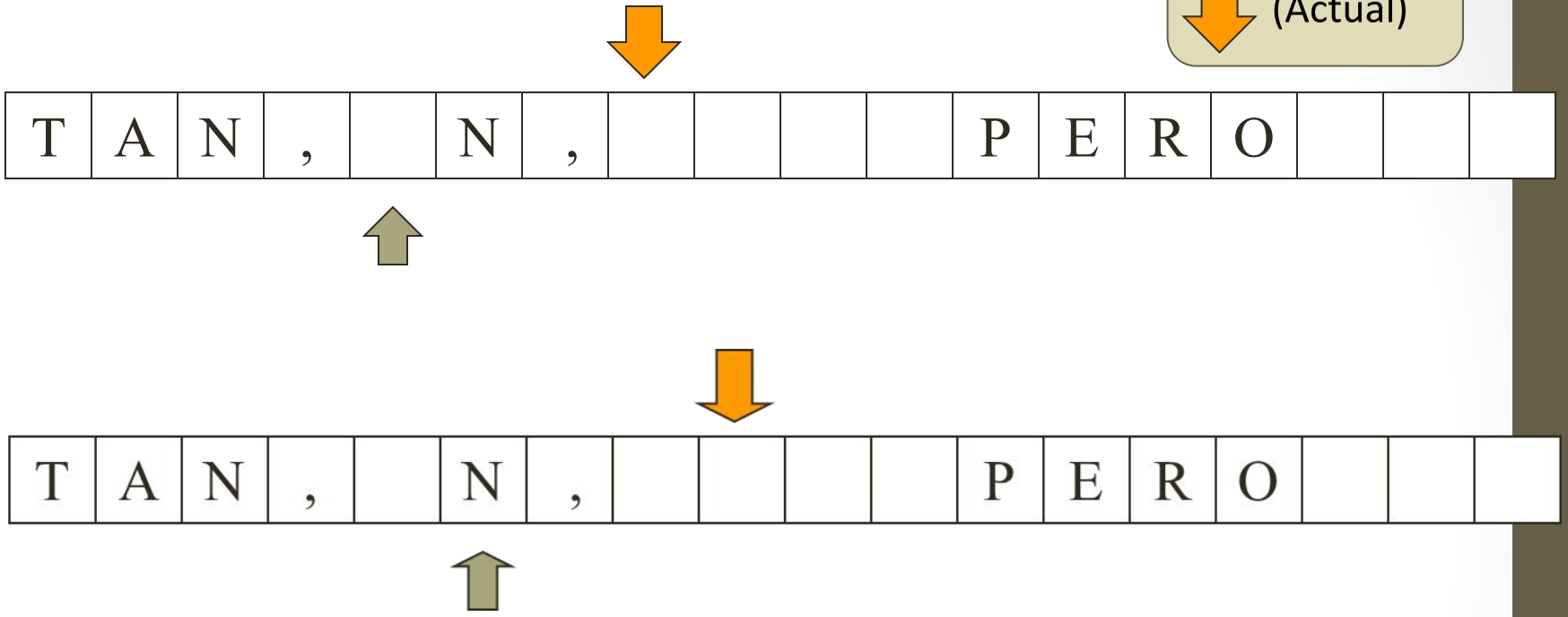


Comprimir

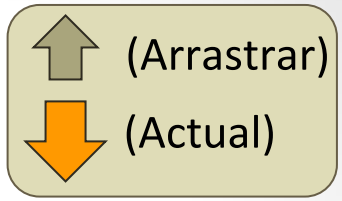


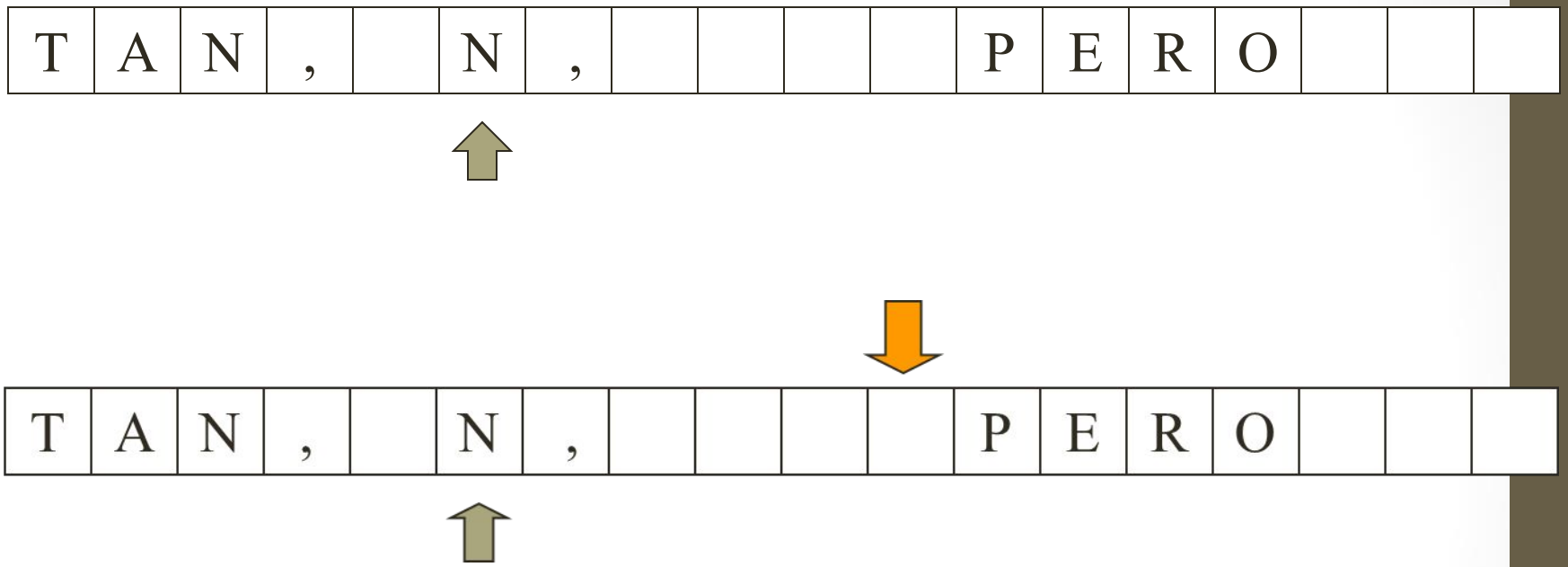
Comprimir



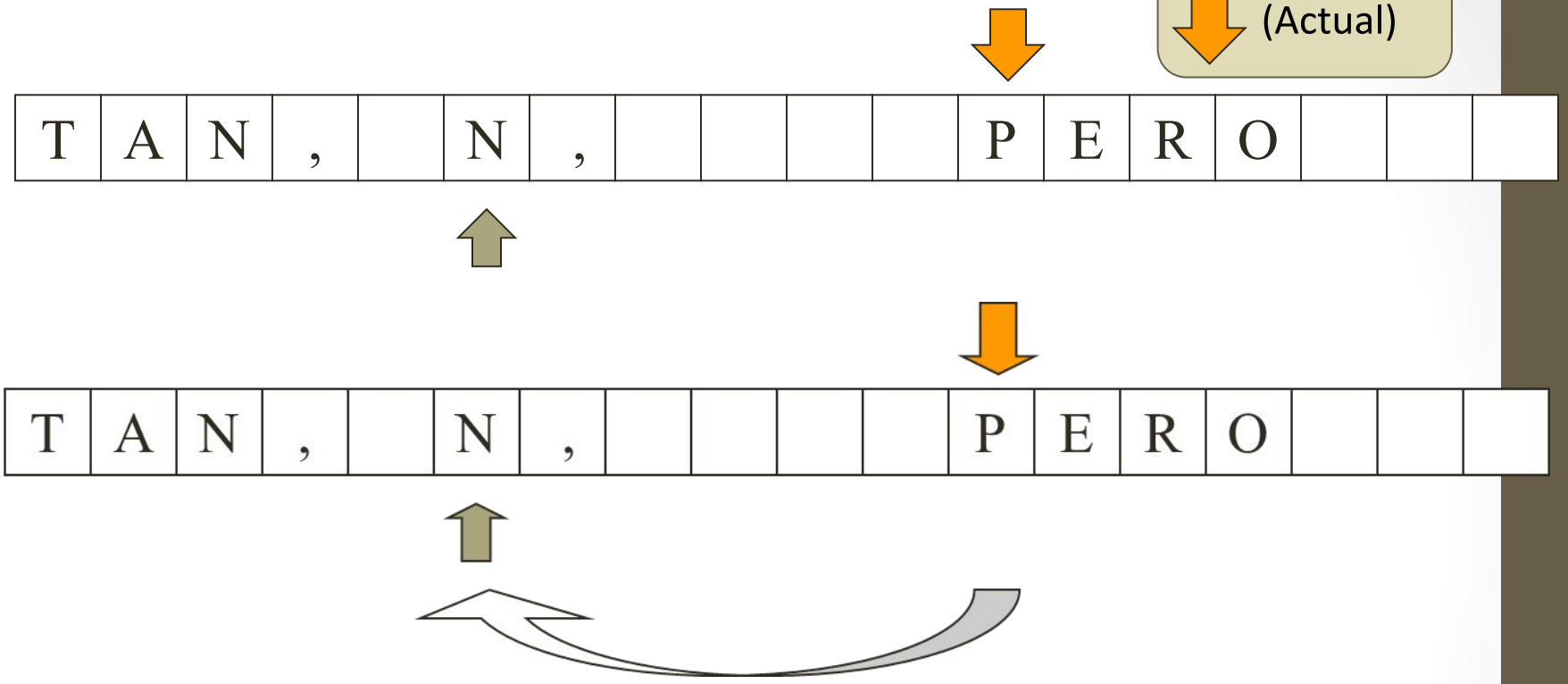


Comprimir

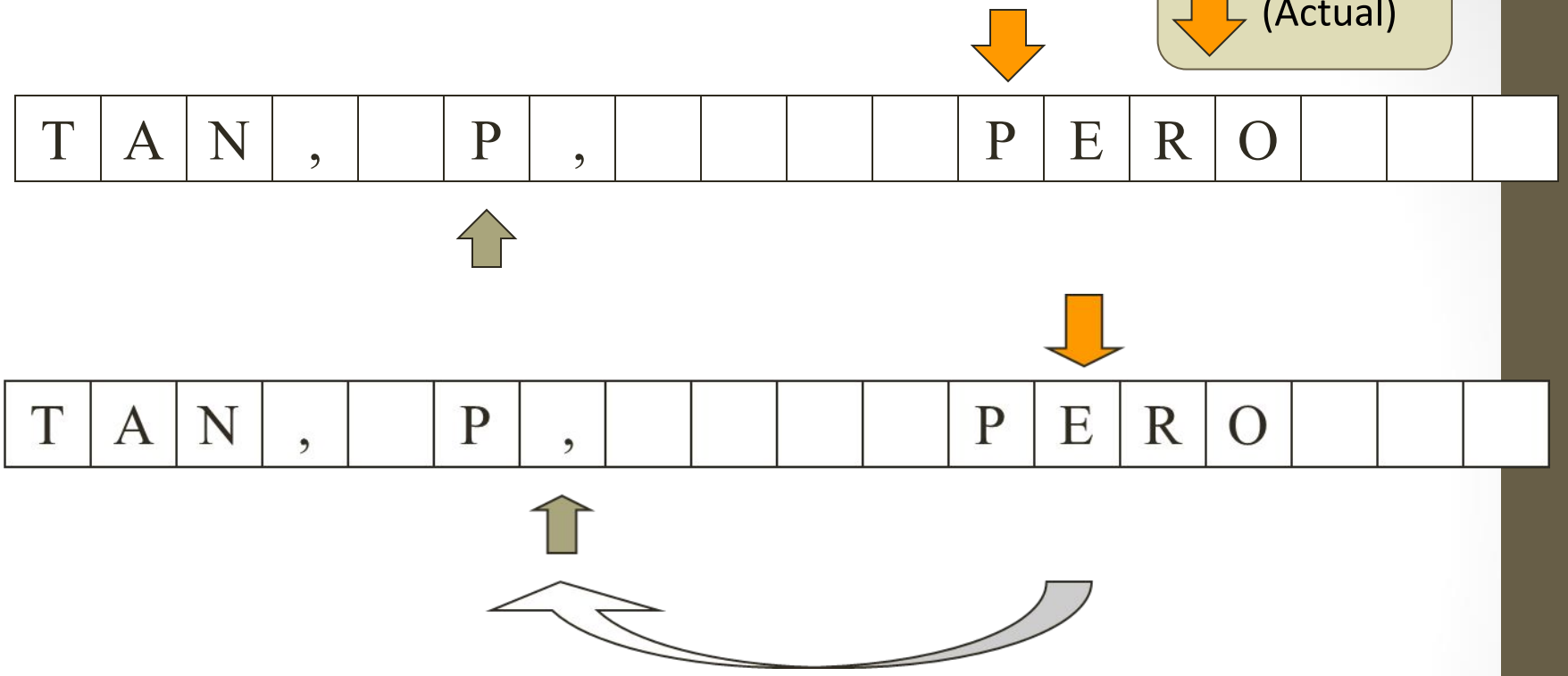




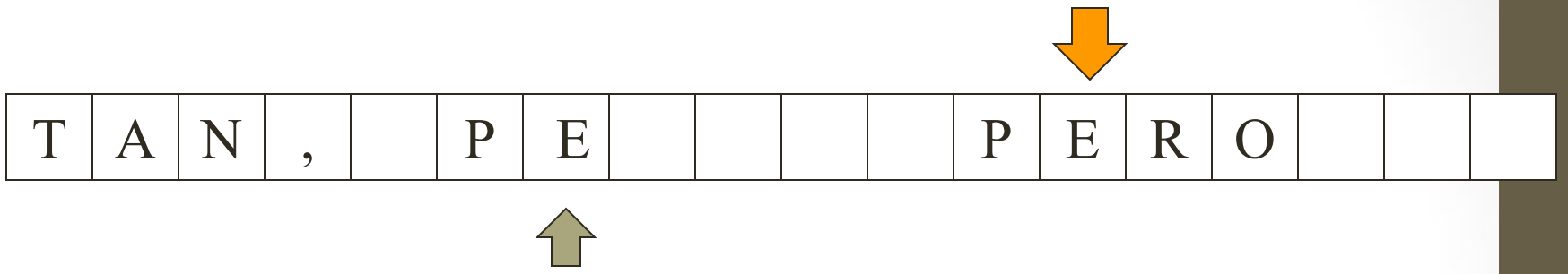
Comprimir



Comprimir



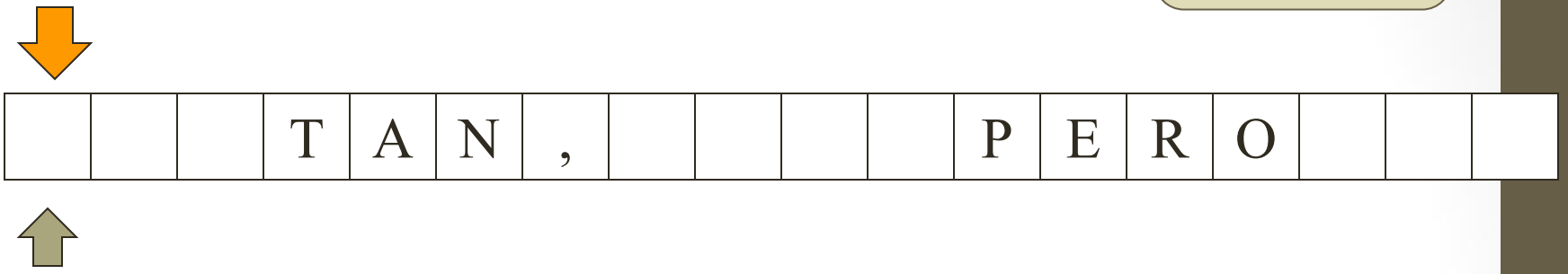
Comprimir



Podríamos continuar “avanzando”, sin embargo ya estamos en condiciones de esbozar un algoritmo.

Notemos que con un único recorrido podemos arrastrar las palabras hacia el principio y los blancos que sobran al final de la oración.

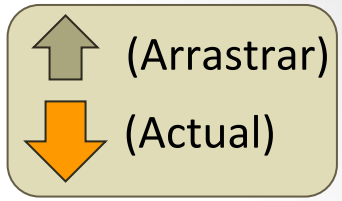
Comprimir

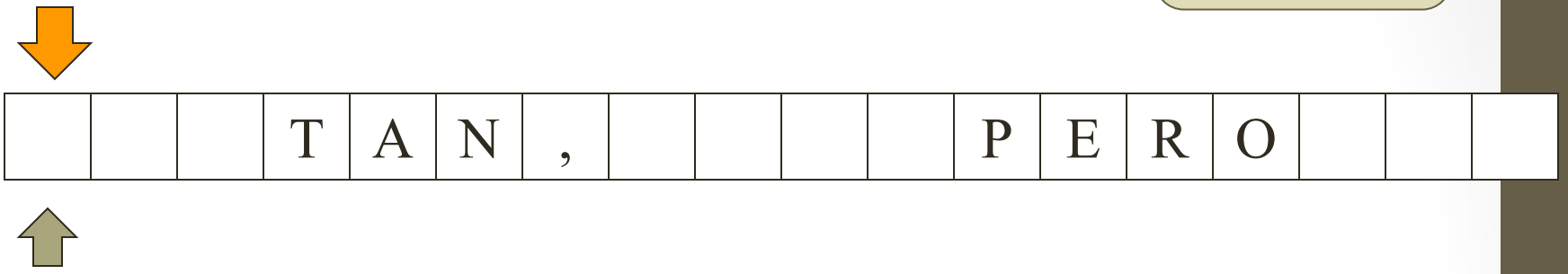


mientras no se termine la oracion
    consumir espacios
    arrastrar palabra y signo
    insertar un blanco

El algoritmo es muy informal y antes de escribir código en Java necesitamos refinar la solución.

Comprimir

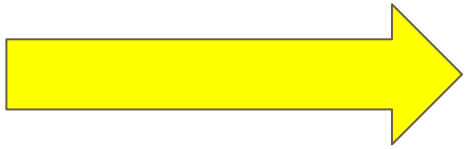


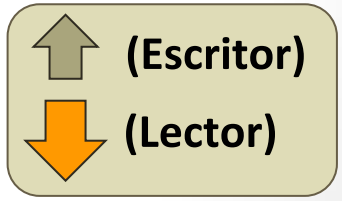


pEsc ←0
pLeer← 0

Como dijimos antes, necesitamos dos variables:

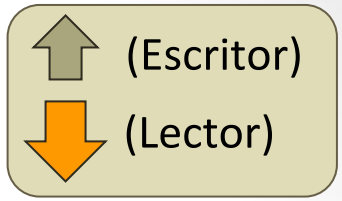
- **posición de lectura**
- **posición de escritura**.

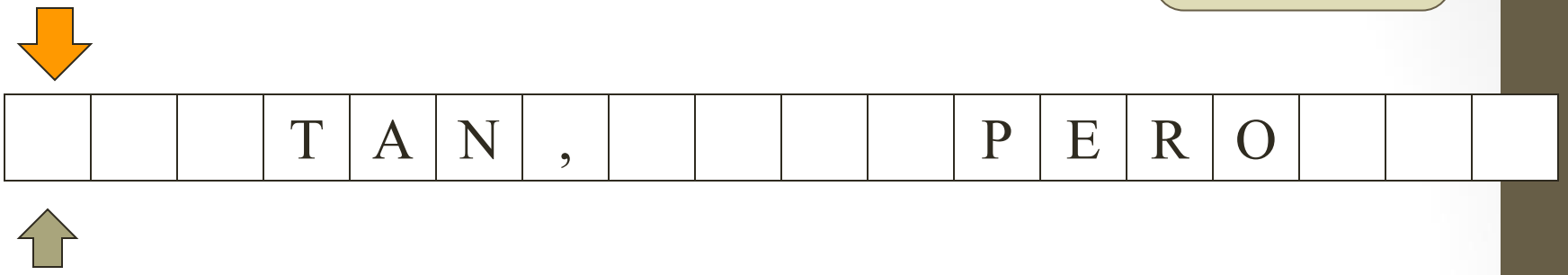




Ambas comienzan refiriéndose a la primera posición.

Comprimir

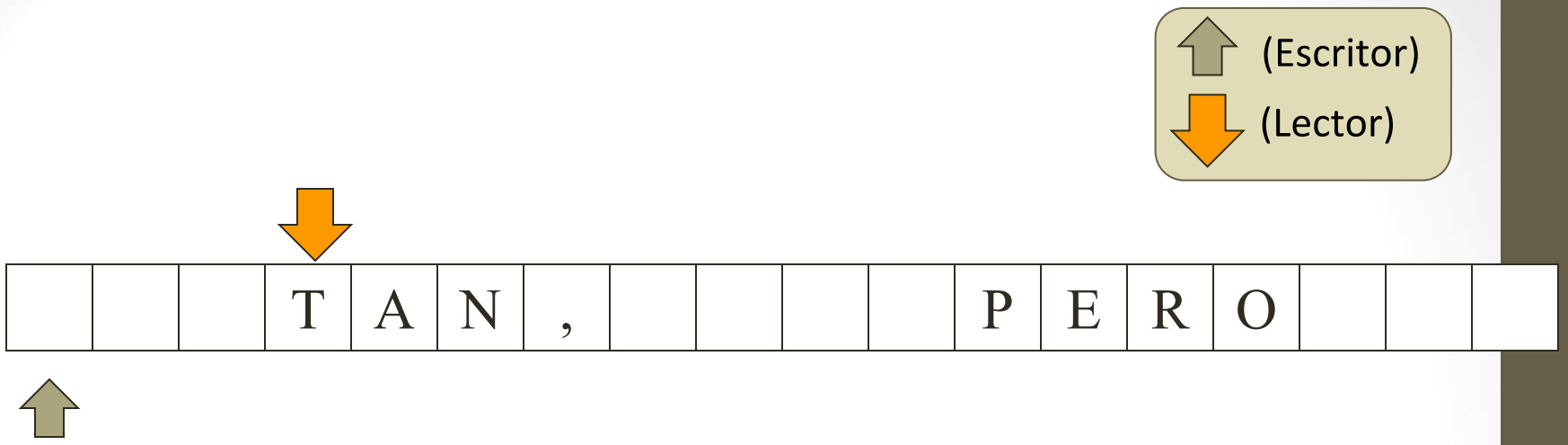




pEsc ←0
pLeer← 0
mientras pLeer < longitud oración

El proceso de comprimir la oración requiere un recorrido exhaustivo del arreglo.

Comprimir Oración

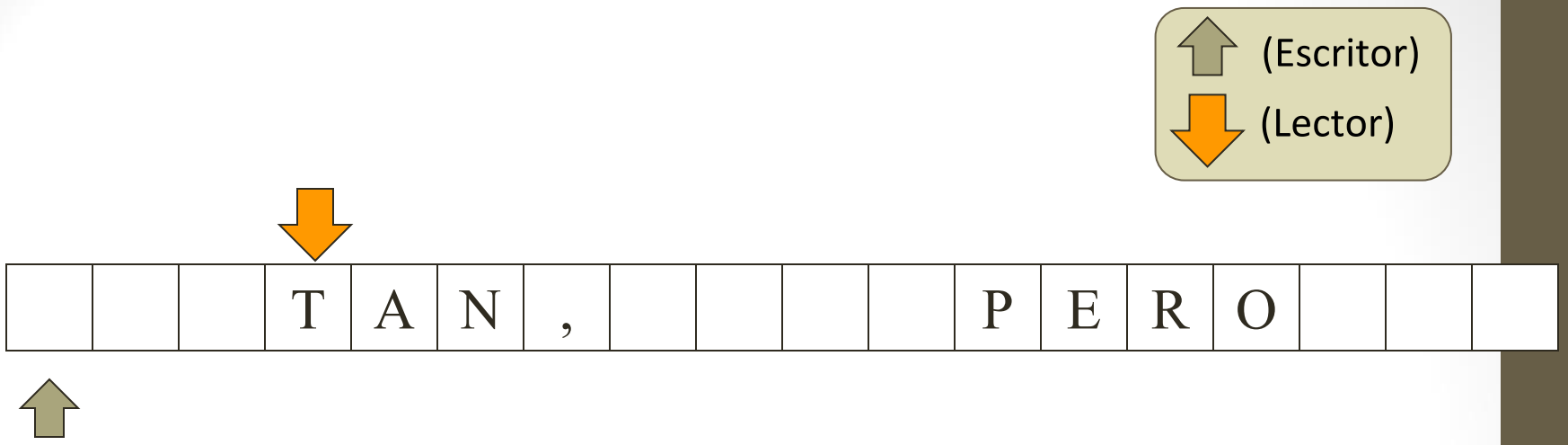



```
pEsc ←0
pLeer← 0
mientras pLeer < longitud oración
   mientras S[pLeer] = ´ ´
      pLeer++
```

Salteamos los blancos avanzando la posición actual hasta encontrar el primer carácter que no es espacio.

Notemos que si después de la palabra hay un signo lo arrastramos también.

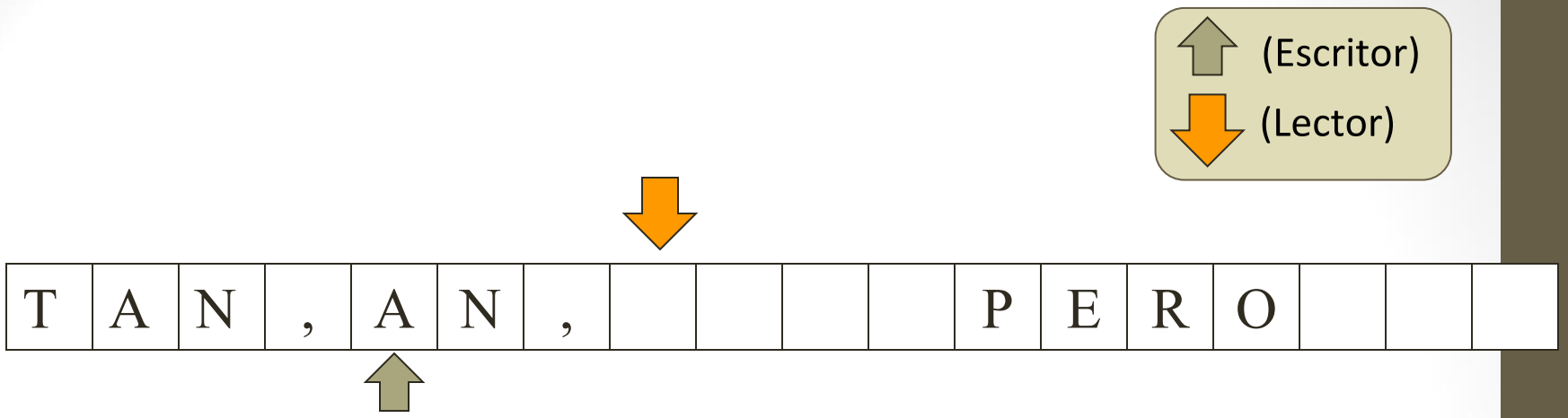
Comprimir Oración




```
pEsc ←0
pLeer← 0
mientras pLeer < longitud oración
   mientras S[pLeer] = ´ ´
      pLeer++
   mientras S[pLeer] ≠ ´ ´
      S[pEsc] ← S[pLeer]
      pLeer++ pEsc++
```

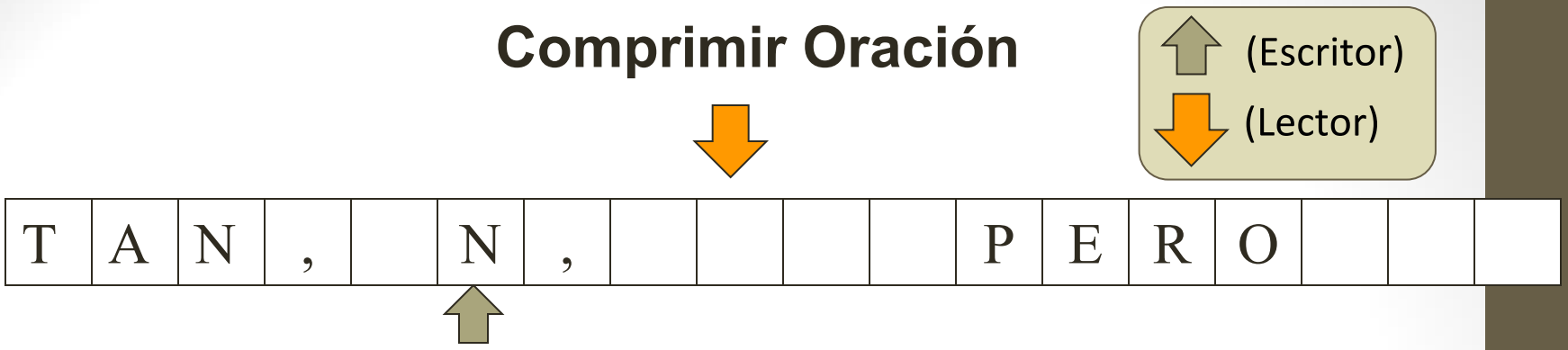
Arrastramos las letras y signos hasta encontrar el próximo espacio.

Comprimir Oración



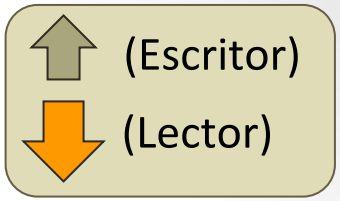

```
pEsc ←0
pLeer← 0
mientras pLeer < longitud oración
   mientras S[pLeer] = ´ ´
      pLeer++
   mientras S[pLeer] ≠ ´ ´
      S[pEsc] ← S[pLeer]
      pLeer++ pEsc++

    S[pEsc] ← ´ ´
    pLeer++ pEsc++
```

Comprimir Oración




```
pEsc ←0
pLeer← 0
mientras pLeer < longitud oración
   mientras   pLeer < longitud oración   y   S[pLeer] = ´ ´
      pLeer++
   mientras  pLeer < longitud oración   y   S[pLeer] ≠ ´ ´
      S[pEsc] ← S[pLeer]
      pLeer++ pEsc++
   si pLeer < longitud oración
     S[pEsc] ← ´ ´
     pLeer++ pEsc++
```

Comprimir Oración



T	A	N	,		P	E	R	O			P	E	R	O			

```
pEsc ←0
pLeer← 0
mientras pLeer < longitud oración
      mientras   pLeer < longitud oración   y S[pLeer] = ´ ´
            pLeer++
      mientras   pLeer < longitud oración  y  S[pLeer] ≠ ´
                  S[pEsc] ← S[pLeer]
            pLeer++ pEsc++
      si pLeer < longitud oración
            S[pEsc] ← ´ ´
            pLeer++ pEsc++
mientras pEsc < longitud oración
      S[pEsc] ← ´ ´
      pEsc++
```