# Tema 3 - Recursivitat
# [ Avançat ]
# - Sessió 10 -

ED

# Algorismes recursius [Recordatori]

Tota funció recursiva està ben definida si es basa en una **definició recursiva** del problema i contempla les dues possibilitats d'execució:

- **Cas bàsic**: Base de la recursivitat, retorna una solució de forma directa i sense tornar-se a cridar a si mateixa.
- **Cas general (recursiu)**: Crida recursiva, retornarà la solució després de tornar-se a cridar a si mateixa amb nous valors dels paràmetres d'entrada que simplificaran el problema inicial apropant-se a la solució.

*Anem ara a veure com processar estructures de dades de forma recursiva*

# Algorismes recursius: Llista de Nodes

- Partim d’una simple llista de Nodes:

```
class LlistaN:

    def __init__ (self,l):
          self._first = None
          self._last = None
          self._size = 0
          self._op = 0
          if type(l)==list:
              for n in l:
                   self.add_last(n)
    @property
    def last(self):
          return self._last
    @property
    def first(self):
          return self._first
```

```
class _Node:
  __slots__ = '_valor' , '_seg’
  def __init__(self, valor, seg=None):
      self._valor = valor
      self._seg   = seg
  @property
  def seg(self):
      return self._seg
  @property
  def valor(self):
      return self._valor
  def __str__(self):
      return str(self._valor)
```

# Exercici 1: Cercar un element dins una llista

Implementeu una funció **recursiva** per **cercar un element** dins una **llista no ordenada**, tal que donada la llista i un valor concret llavors retorni *True* si aquell element existeix dins la llista, i *False* en cas contrari.

Aquesta funció haurà de tenir la següent signatura:

**`cerca(llista, x)`**

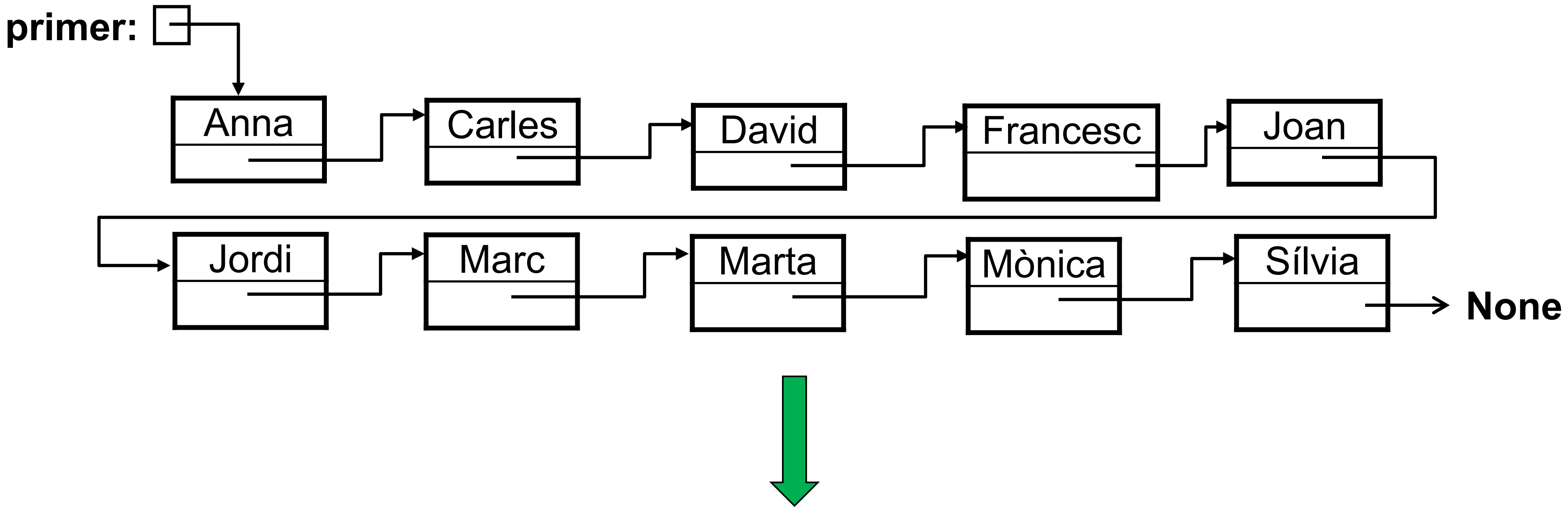
Exemple d'execució:

```
l = LlistaN.LlistaN([ 3, 2, 1, 6, 5, 4, 9, 8, 7, 10 ])
cerca(l,3)  → True
cerca(l,12) → False
```

NOTA: Penseu que pot ser utilitzar una funció auxiliar a la que realment implementa la recursivitat us podria ajudar, tal i com ja hem fet en exemples anteriors.

# Exercici 2: Cadena inversa del contingut d'una llista

Dissenyar un algorisme **recursiu** tal que donada una llista, retorni un string amb el contingut de **la llista** però en **ordre invers**:




"Sílvia, Mònica, Marta, Marc, Jordi, Joan, Francesc, David, Carles, Anna"

# Exercici 2: Cadena inversa del contingut d'una llista

## Es pot fer el mateix sense recursivitat? Com?

```
def printInversRec(self,nodeAct):
    ...
```

```
def printInvers(self):
    ...
```

```
lNoms=["Anna","Carles","David","Francesc","Joan","Jordi",
"Marc","Marta","Mònica","Sílvia"]
Resultat = lNoms.printInvers()
```

# Recursivitat doble: Sèrie de Fibonacci

La Sèrie de Fibonacci és: **1,1,2,3,5,8,13,…**

- Definició bàsica: Cada element és la suma dels dos elements anteriors
- Definició formal: $a_0 = 1$

  $a_1 = 1$

  $a_n = a_{n-1} + a_{n-2}$
- Definició recursiva:
  – **Cas bàsic:** n = 0 → Fibonacci (0) = 1

  n = 1 → Fibonacci (1) = 1

  – **Cas general:** n > 1 → Fibonacci (n) = Fibonacci (n-1) + Fibonacci (n-2)

# Recursivitat doble: Sèrie de Fibonacci

La Sèrie de Fibonacci és: **1,1,2,3,5,8,13,…**

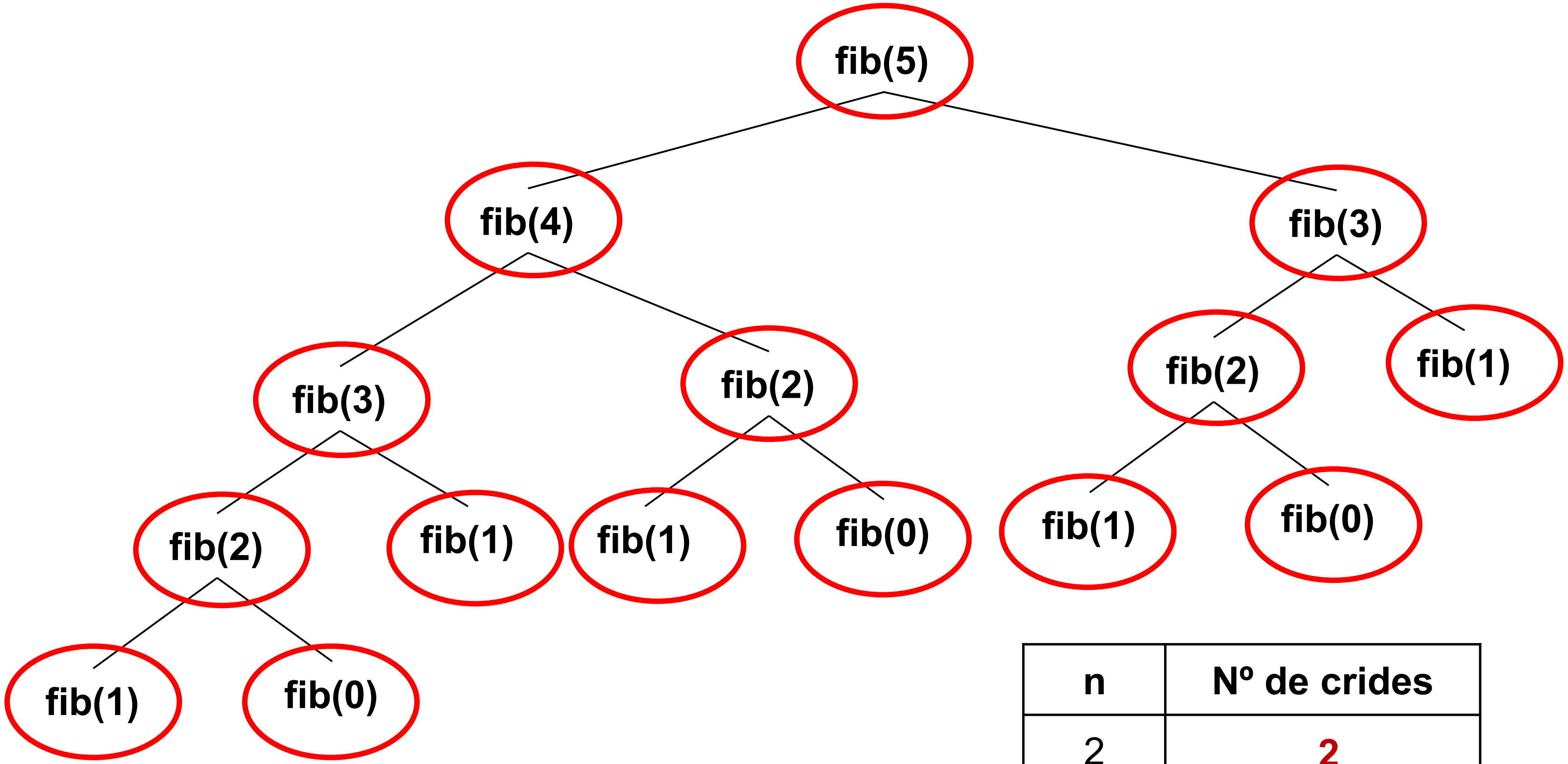
- Definició bàsica: Cada element és la suma dels dos elements anteriors

```
def fibonacci(int n):

    if ((n == 0) or (n == 1)):        # Cas bàsic
        return 1
                                      # Cas general
    return fibonacci(n-1) + fibonacci(n-2)
```

# Recursivitat doble: Anàlisi d'algorismes recursius

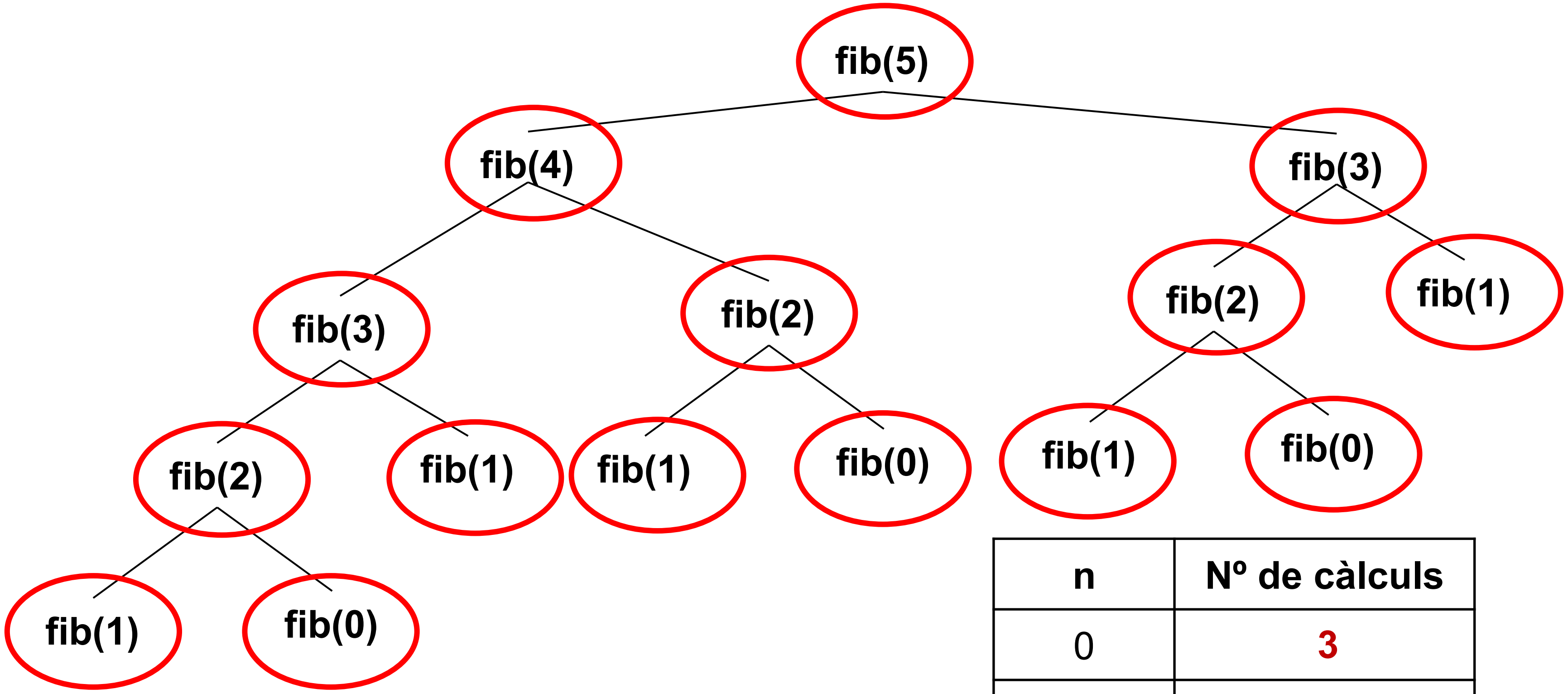
Quantes crides es fan per calcular `fibonacci(n)`?




| n | Nº de crides |
|---|---|
| 2 | **2** |
| 3 | **4** |
| 4 | **8** |
| 5 | **14** |

# Recursivitat doble: Anàlisi d'algorismes recursius

I quants càlculs repetits s'estan fent per calcular `fibonacci(n)`?




| n | Nº de càlculs |
|---|---|
| 0 | 3 |
| 1 | 5 |
| 2 | 3 |
| 3 | 2 |
| 4 | 1 |
| 5 | 1 |

# Recursivitat doble: Sèrie de Fibonacci en versió iterativa

Aquí tenim una implementació **iterativa** equivalent:

```
def fibonacciIter(n):

    fib = 1
    valorAnt = 0
    while (n>0):

        valorAct = fib         /* valorAct = fib(n-1) */
        fib = fib + valorAnt
        valorAnt = valorAct    /* valorAnt = fib(n-2) */
        n = n-1

    return fib
```

| **n** | **Nº repeticions del bucle** |
|---|---|
| 2 | **2** |
| 3 | **3** |
| 4 | **4** |
| 5 | **5** |
| … | **…** |
| 50 | **50** |

# Anàlisi d’algorismes recursius

- Quantes repeticions del bucle es fan per calcular `fibonacci(n)` a la versió iterativa?
  - Nº repeticions → **O(n)**
- Quantes crides es fan per calcular `fibonacci(n)` a la versió recursiva?
  - Nº repeticions → **O($2^n$)**

| **n** | **Nº de crides recursives** | **Nº repeticions del bucle** |
|---|---|---|
| 2 | **2** | **2** |
| 3 | **4** | **3** |
| 4 | **8** | **4** |
| 5 | **14** | **5** |
| … | **…** | **…** |
| 50 | **≈ $2\text{x}10^{10}$** | **50** |

- Comparació versió recursiva vs. iterativa:
  - Versió recursiva més clara
  - Versió iterativa més eficient, moltes menys repeticions del bucle que amb crides recursives (especialment amb una 'n' gran)

# Anàlisi d'algorismes recursius

- `Factorial(n)`: versió recursiva vs. iterativa
  - n repeticions versió iterativa → **O(n)**
  - n crides recursives → **O(n)**

```
def fact(n):
    if ((n == 0) or (n == 1)):
    # Cas bàsic    */
        return 1
    # Cas general */
    return n*fact(n-1)
```

```
def factIter(n):
    fac = 1
    while (n > 0):
        fac = fac * n
        n = n-1
    return fac
```

| n | Nº de crides recursives | Nº repeticions del bucle |
|---|---|---|
| 2 | **2** | **2** |
| 3 | **3** | **3** |
| 4 | **4** | **4** |
| 5 | **5** | **5** |
| … | **…** | **…** |
| 50 | **50** | **50** |

- Comparació versió recursiva vs. iterativa:
  - Mateix número de crides/repeticions
  - Versió iterativa més eficient, cost més gran (en temps i memòria) d'una crida recursiva respecte a una repetició de bucle.

# Algorismes recursius: Cerca dins una taula

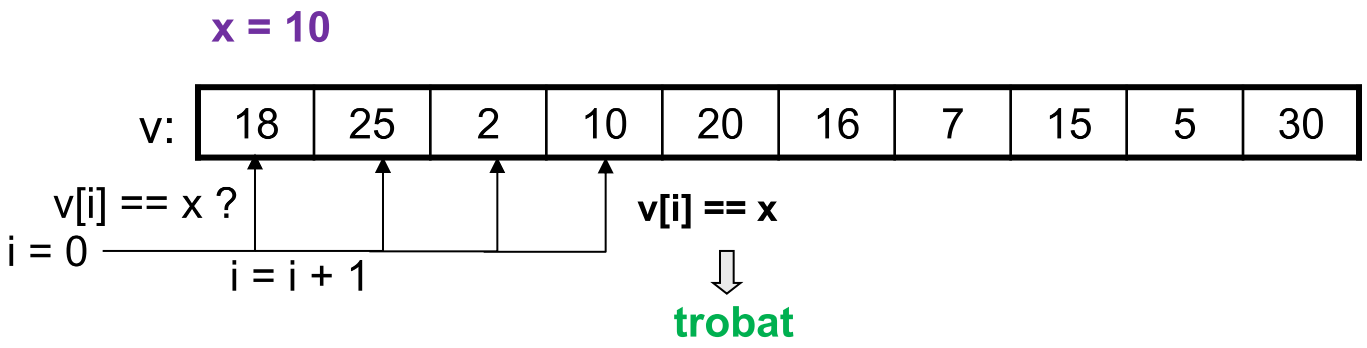
Mirem ara com aplicar la recursivitat a la cerca dins taules (vectors).

En general existeixen dos casos diferents:

1. La taula **no** està **ordenada**
   - S'ha de fer una cerca seqüencial
2. La taula **sí** està **ordenada**
   - Es pot fer una cerca binària

# Cerca 1: La taula no està ordenada

## Cerca seqüencial:




```
def cercaSeqüencial(v, x):
   i = 0
   while (i < MAX)):
      if (v[i] == x):
         return True

      i = i+1
   return False
```

# Cerca 2: La taula està ordenada – Cerca binària

**Cerca dins una taula ordenada:**

- Si la taula està **ordenada**, llavors com podria ajudar la **recursivitat** a trobar un esquema de solució **molt més eficient**?
- Doncs fent servir un algorisme (recursiu) de **cerca binària**:
  - Aquest algorisme segueix l'esquema ***"Dividir i Vèncer"***:
    - Dividir el problema a resoldre en dues meitats;
    - I aplicar la solució recursivament a una o a les dues meitats.

# Cerca 2: La taula està ordenada – Cerca binària

**x = 10**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| v: | 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 0

central = (primer + últim) / 2 = 7

últim = 14

⇓

**v[central] > x**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 10**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| v: | 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 0

central = (primer + últim) / 2 = 7

últim = 14

⇩

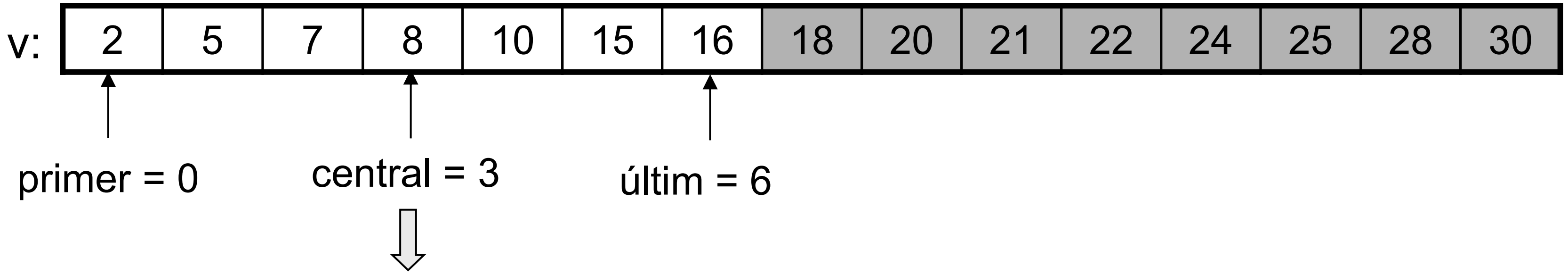
**v[central] > x**

⇩

**últim = central - 1**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 10**




**v[central] < x**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 10**

v:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 0 central = 3 últim = 6

**v[central] < x**

**primer = central + 1**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 10**

v:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 4

central = 5

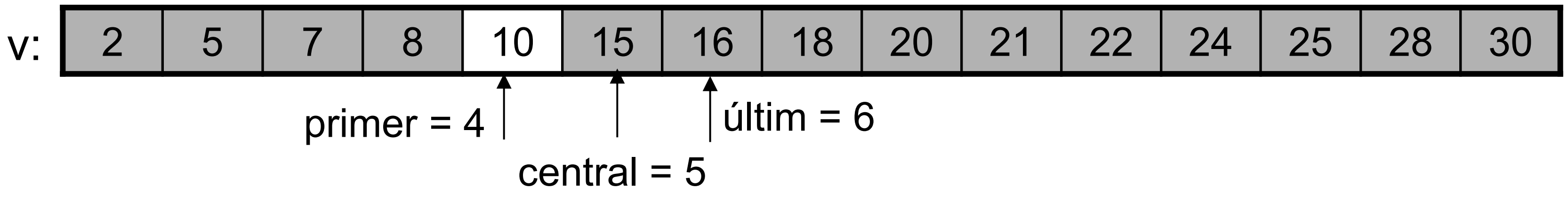
últim = 6

$\Downarrow$

**v[central] > x**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 10**




**v[central] > x**

**últim = central - 1**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 10**

v:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 4

últim = 4

central = 4

⇓

**v[central] = x**

⇓

**trobat**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 19**

| v: | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 0

central = (primer + últim) / 2 = 7

últim = 14

⇓

**v[central] < x**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 19**

v:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 0

central = (primer + últim) / 2 = 7

últim = 14

⇓

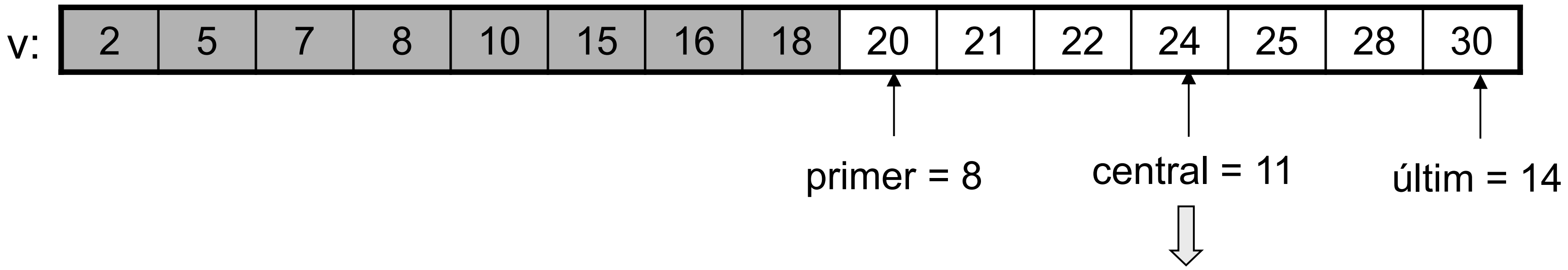
**v[central] < x**

⇓

**primer = central + 1**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 19**




**v[central] > x**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 19**

| v: | 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

primer = 8 central = 11 últim = 14

⇩

**v[central] > x**

⇩

**últim = central - 1**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 19**

v:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 8 ↑ ↑ ↑ últim = 10

central = 9

⇓

**v[central] > x**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 19**

| v: | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 8

último = 10

central = 9

⇓

**v[central] > x**

⇓

**últim = central - 1**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 19**

v:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 8

últim = 8

central = 8

⇓

**v[central] > x**

⇓

**últim = central - 1**

# Cerca 2: La taula està ordenada – Cerca binària

**x = 19**

| v: | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

últim = 7 primer = 8

**últim < primer**

**no trobat**

# Exercici 3: Cerca Binaria dins taula ordenada

Implementeu l'algorisme de cerca binaria.

Imaginem que tenim la llista ordenada `ll` i que volem cercar ‘valor’ fent la crida:

```
Cerca(ll, valor)
```

En aquest cas penseu si també teniu que implementar una funció auxiliar a més de la que sigui la recursiva.

# Cerca 2: La taula està ordenada – Cerca binària *(exemple 1)*

`CercaBinaria (v, x, primer, ultim)`

⇩

`CercaBinaria (v, 10, 0, 14)`

| v: | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 0 central = 7 últim = 14

⇩

`CercaBinaria (v, 10, 0, 6)`

| v: | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 0 central = 3 últim = 6

⇩

`CercaBinaria (v, 10, 4, 6)`

| v: | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 4 últim = 6

⇩

`CercaBinaria (v, 10, 4, 4)`

| v: | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 4 últim = 4

`trobat = true`

# Cerca 2: La taula està ordenada – Cerca binària *(exemple 2)*

`CercaBinaria (v, 19, 0, 14)`

| v: | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 0 central = 7 últim = 14

`CercaBinaria (v, 19, 8, 14)`

| v: | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 8 central = 11 últim = 14

`CercaBinaria (v, 19, 8, 10)`

| v: | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 8 últim = 10

`CercaBinaria (v, 19, 8, 8)`

| v: | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

primer = 8 últim = 8

`CercaBinaria (v, 19, 8, 7)`

| v: | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 5 | 7 | 8 | 10 | 15 | 16 | 18 | 20 | 21 | 22 | 24 | 25 | 28 | 30 |

últim = 7 primer = 8

`primer > ultim`

`trobat = false`

# Exercici 3: Cerca Binaria dins taula ordenada – Iteratiu

Es pot fer la cerca binaria també de forma iterativa?

Si és així, llavors com?

# Algorismes recursius: Complexitat – Cerca binària

Anem a calcular el cost de la Cerca Binaria per expansió de la recurrència:

***cercaBinaria(v, valor, primer, ultim)***

```
def cerca(v, x):
   return cercaBinaria(v, x, 0, len(v)-1)

def cercaBinaria(v, x, primer, ultim):
   ...
```

$$T(n) = \begin{cases} 1 & \text{si (primer > ultim) ó (ultim-primer < 0) ó (n<0)} \\ 1 + T(n/2) & \text{si (ultim-primer > 0) ó (n>0)} \end{cases}$$

$\mathbf{T(n)} = 1+T(n/2) = 1+(1+T(n/2^2)) = 1+1+(1+ T(n/2^3)) = \ldots = \mathbf{k+T(n/2^k)}$

Fins que $\mathbf{2^k = n+1}$ (cas base)

Si $\mathbf{2^k = n+1 \rightarrow k=\log_2(n+1) \rightarrow T(n) = \log_2(n) + 1 \rightarrow O(\log_2(n))}$

NOTA: A l'algorisme iteratiu veiem que la mida de la part de vector que s'analitza es va dividint per 2, per tant ens passa igual: $n/2$ , $(n/2)^2$ , etc. Així que es va repetint el bucle $\log_2(n)$ vegades → $\mathbf{O(\log_2(n))}$

# Algorismes recursius: Complexitat – Cerca seqüencial vs. binària

En total quantes iteracions executa l'algorisme de cerca binària?

- **Mínim:** *1* (quan x és el primer element central)
- **Màxim:** $log_2(MAX)$ (quan es fan totes les divisions)
- **Mitjana:** aproximadament $log_2(MAX)$

| Mida vector MAX | Cerca seqüencial MAX | Cerca binària $log_2(MAX)$ |
|---|---|---|
| 16 | 16 | 4 |
| 1024 | 1024 | 10 |
| 1.000.000 | 1.000.000 | 20 |
| 1.000.000.000 | 1.000.000.000 | 30 |

# Algorismes recursius: Ordenació d'un vector

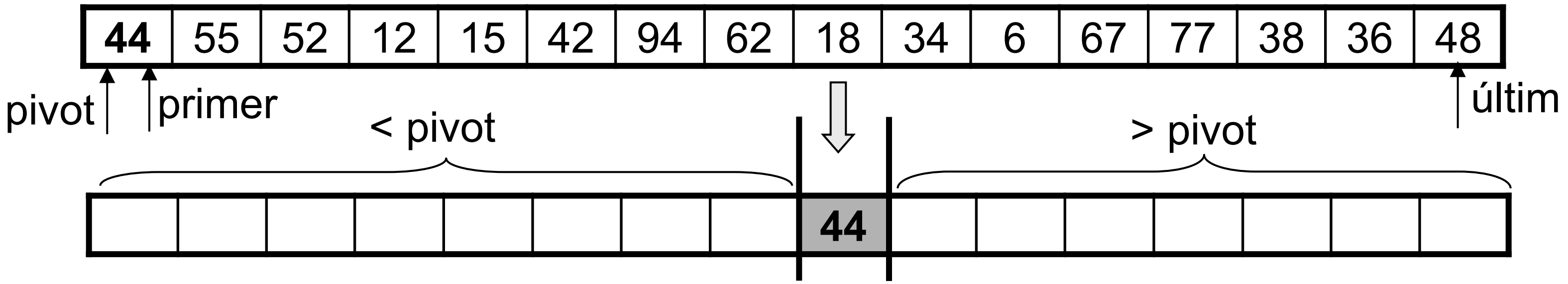
Analitzem ara com ordenar un vector que està desordenat...

Utilitzarem per fer l'ordenació l'algorisme recursiu **Quicksort**

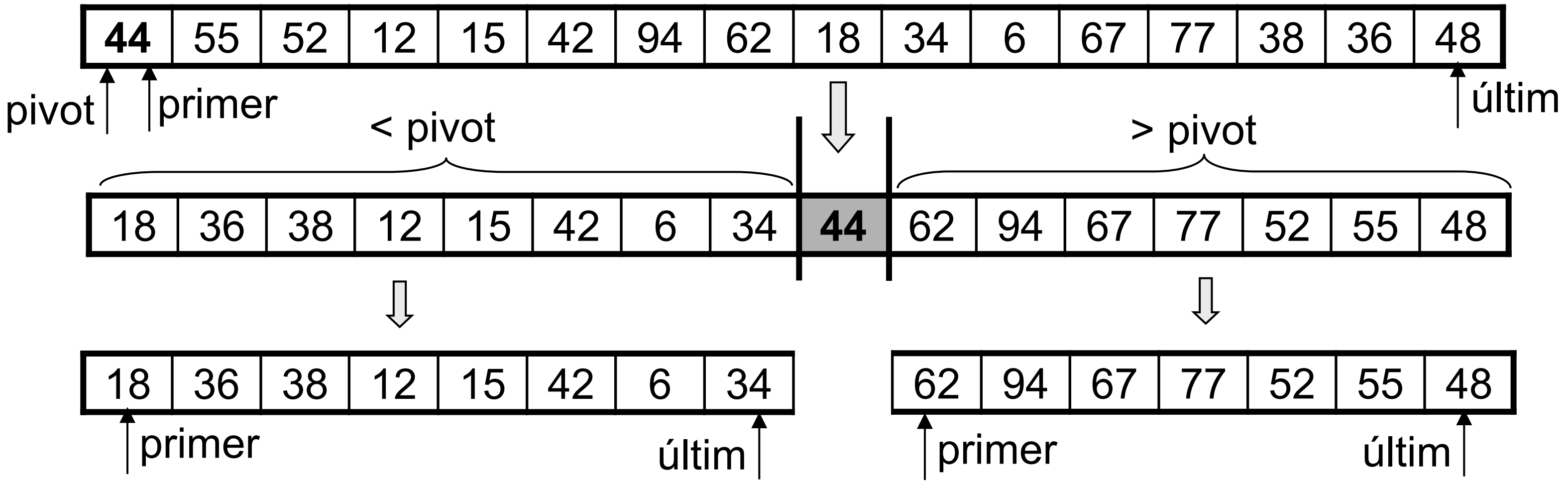
Complexitat del Quicksort:

- Cas pitjor: **$O(n^2)$**
- Cas promig: **$O(n*log(n))$**

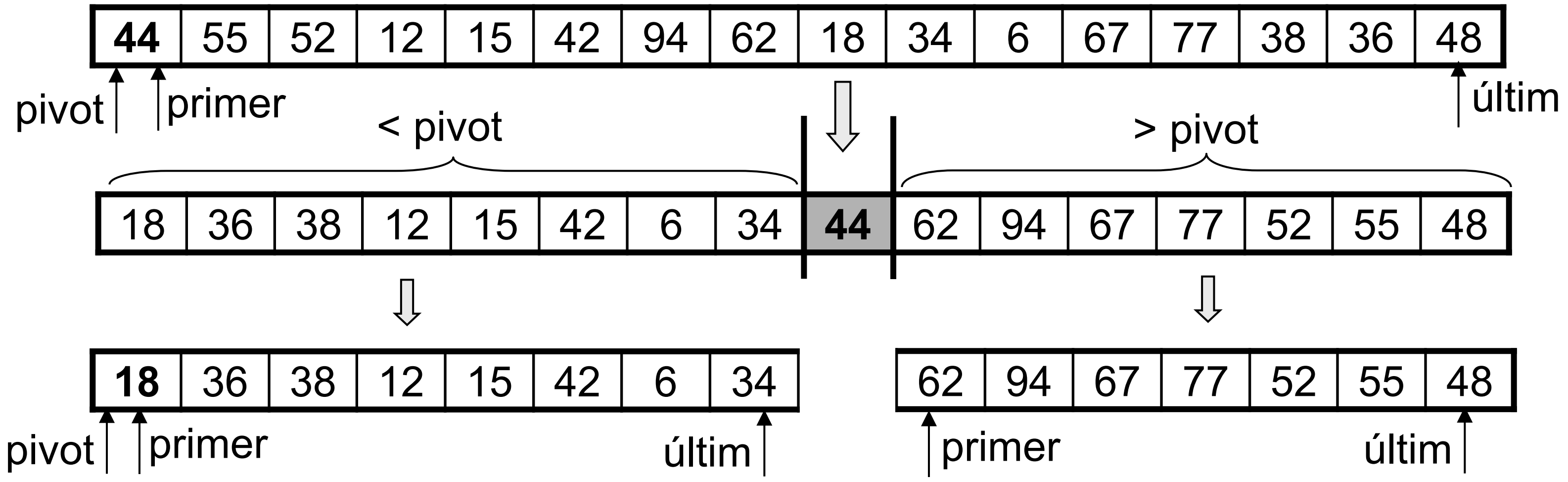
# Algorismes recursius: ordenació ràpida – Quicksort



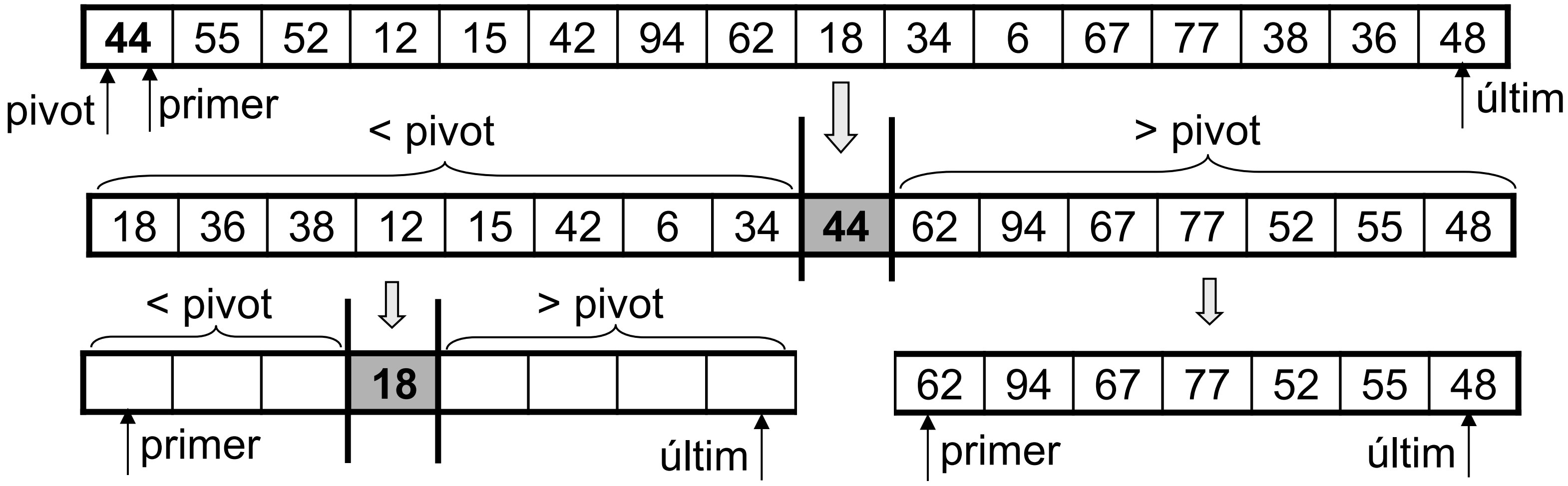
# Algorismes recursius: ordenació ràpida – Quicksort



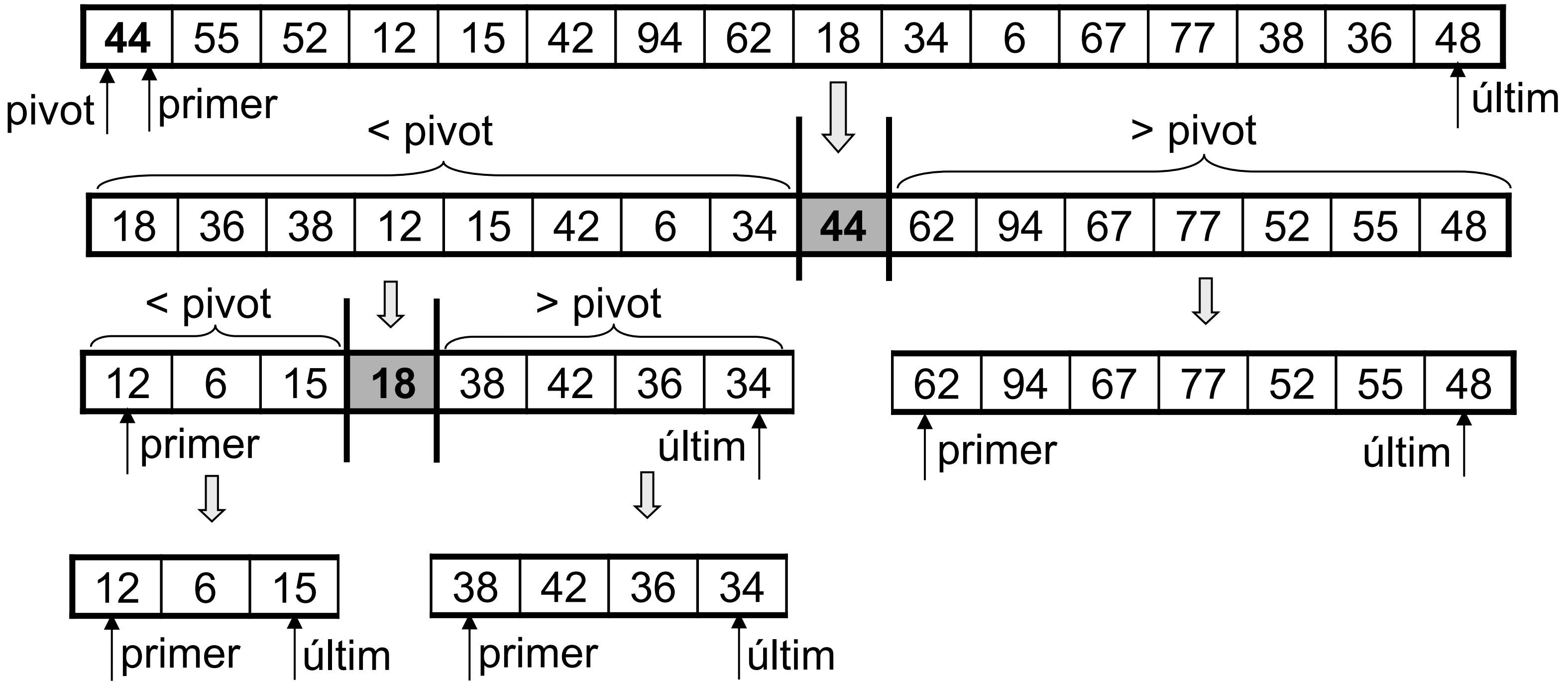
# Algorismes recursius: ordenació ràpida – Quicksort



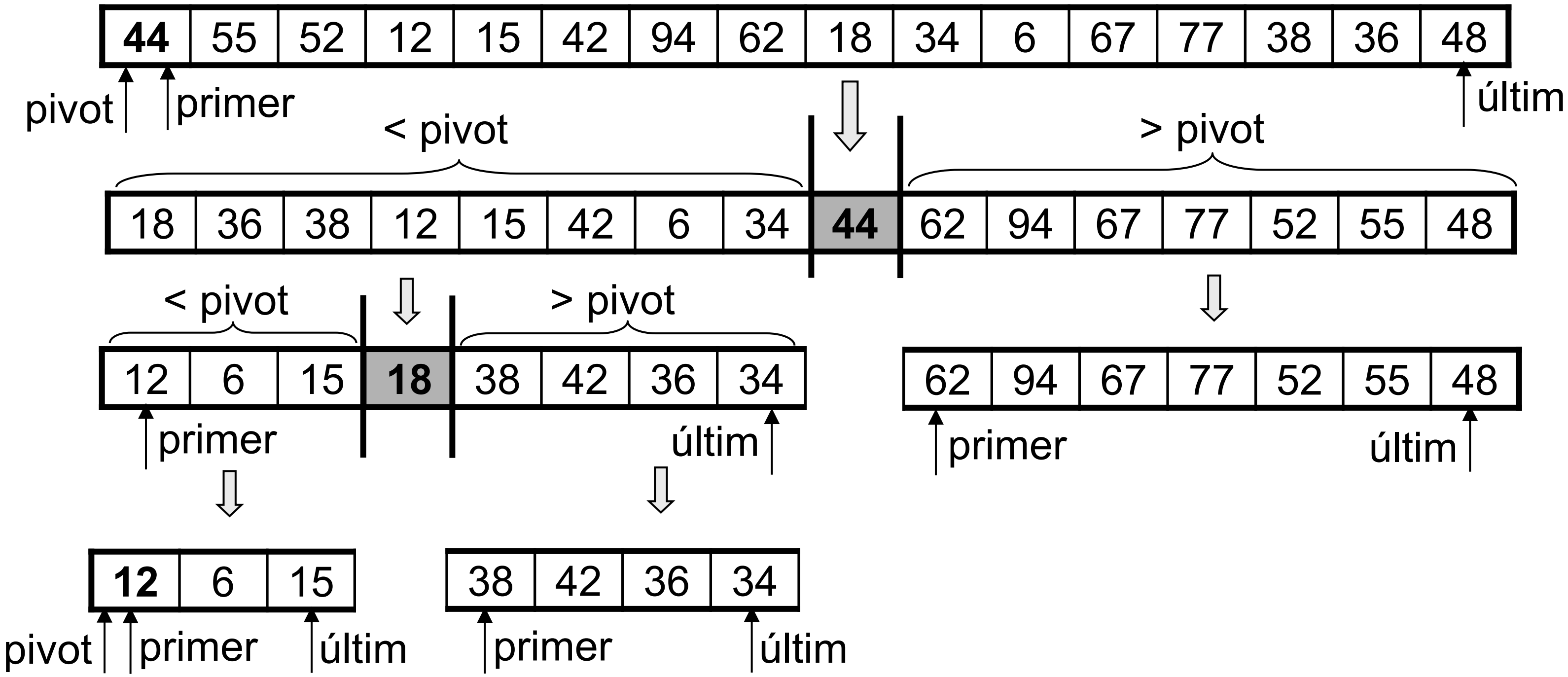
# Algorismes recursius: ordenació ràpida – Quicksort



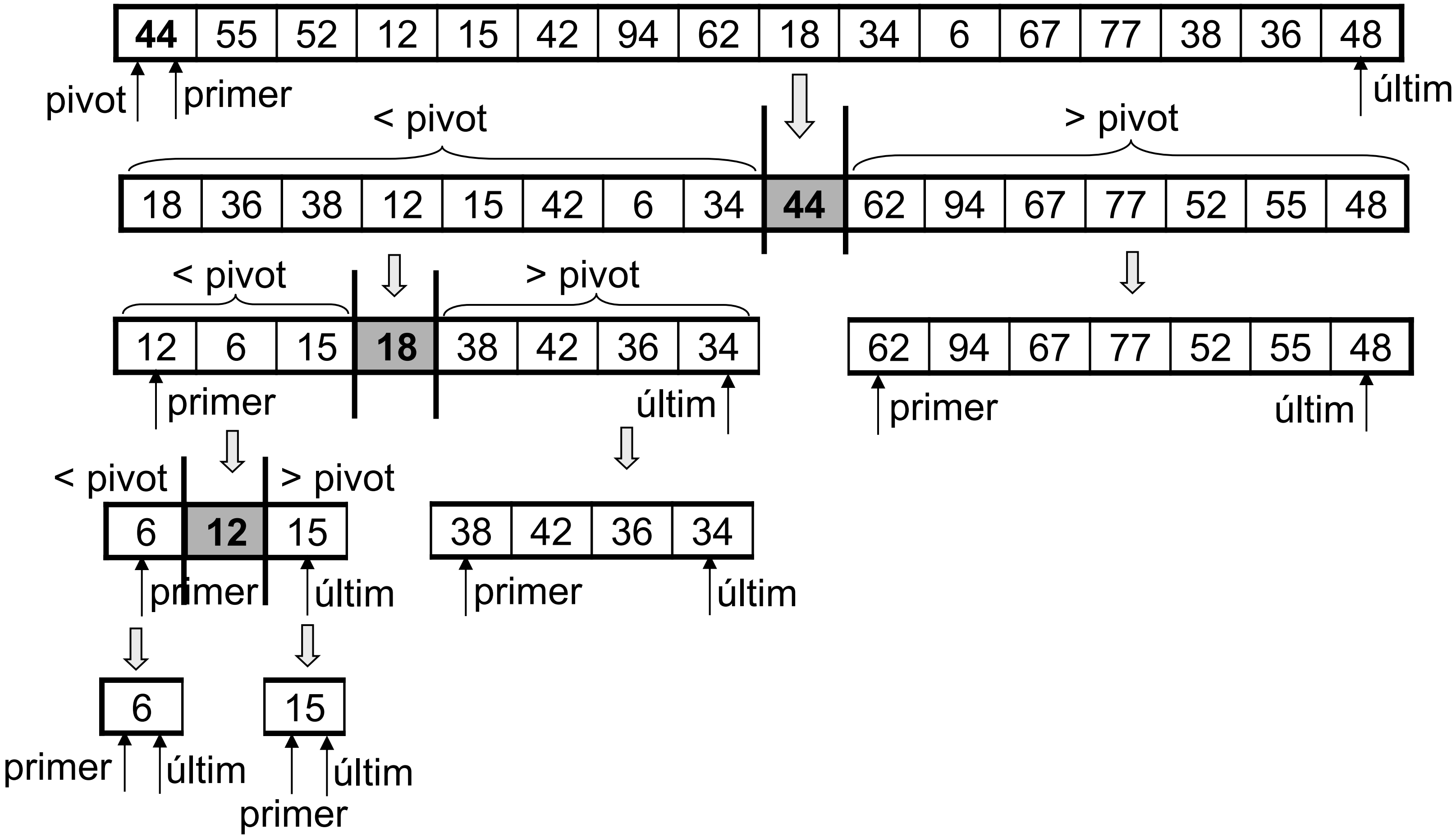
# Algorismes recursius: ordenació ràpida – Quicksort



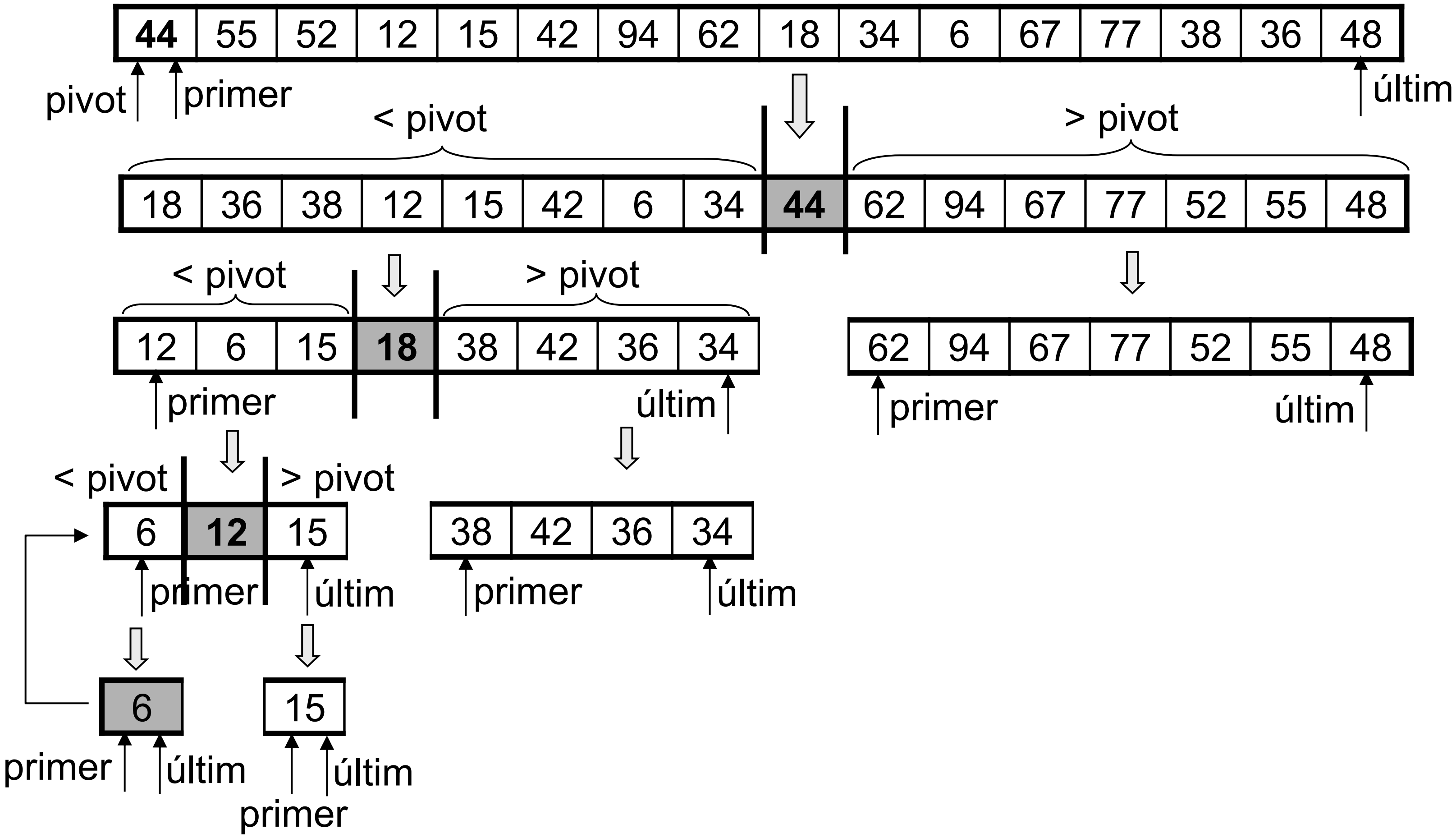
# Algorismes recursius: ordenació ràpida – Quicksort



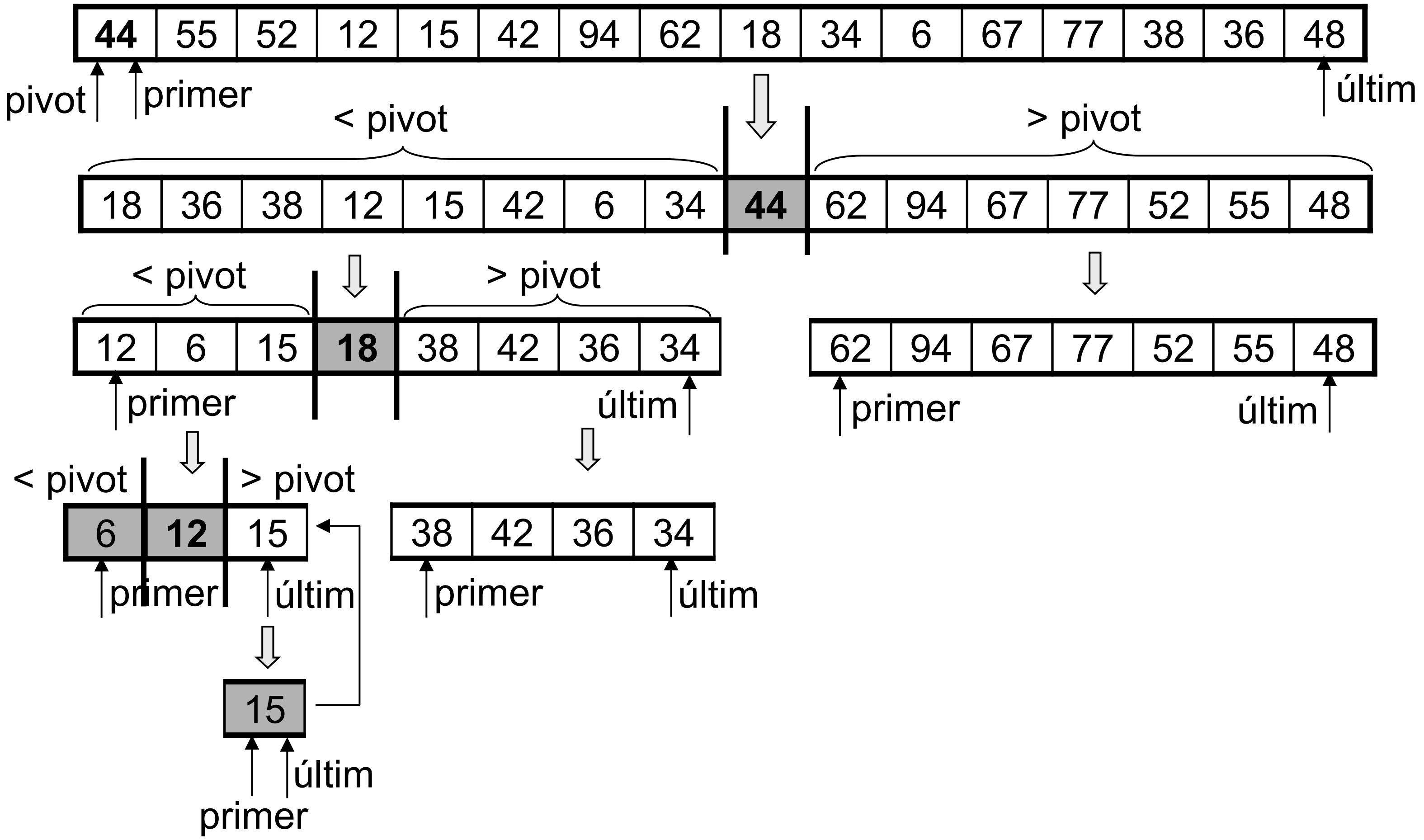
# Algorismes recursius: ordenació ràpida – Quicksort



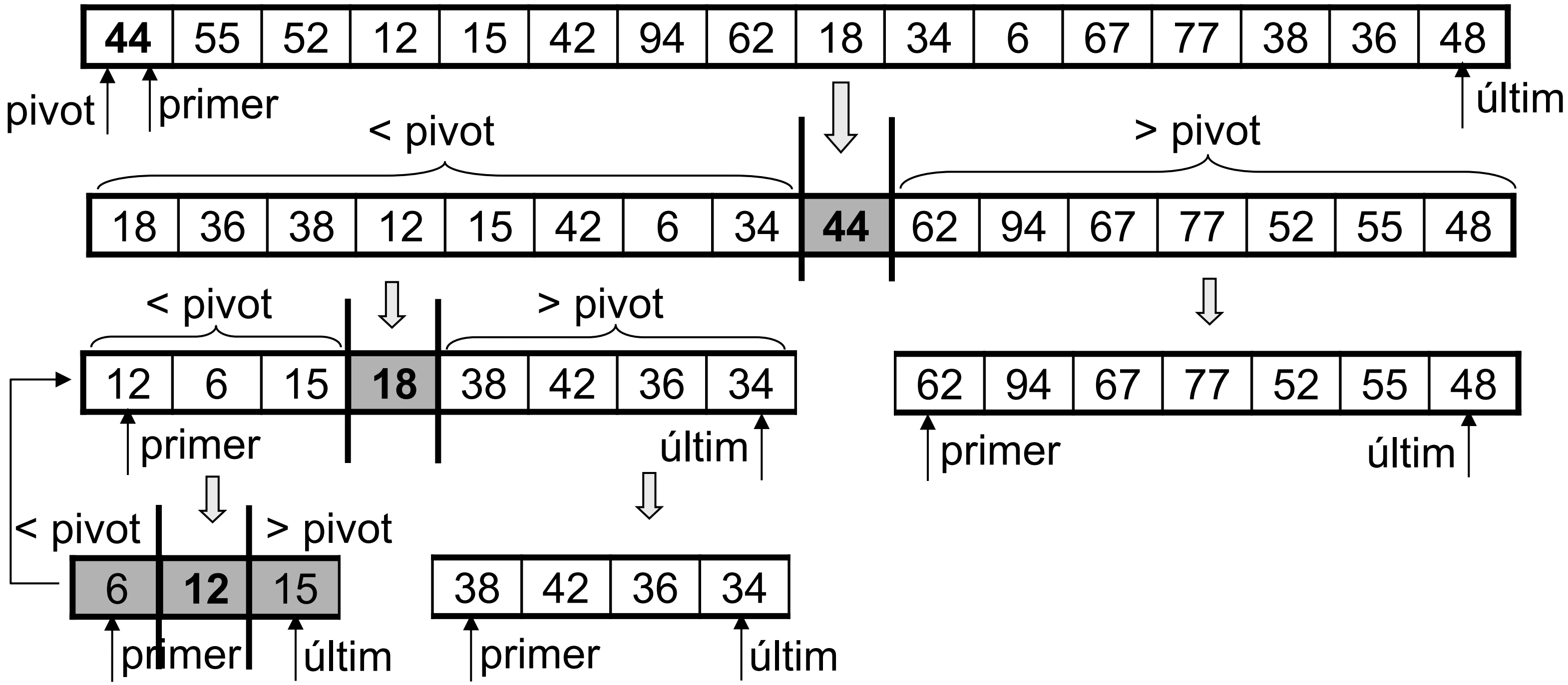
# Algorismes recursius: ordenació ràpida – Quicksort



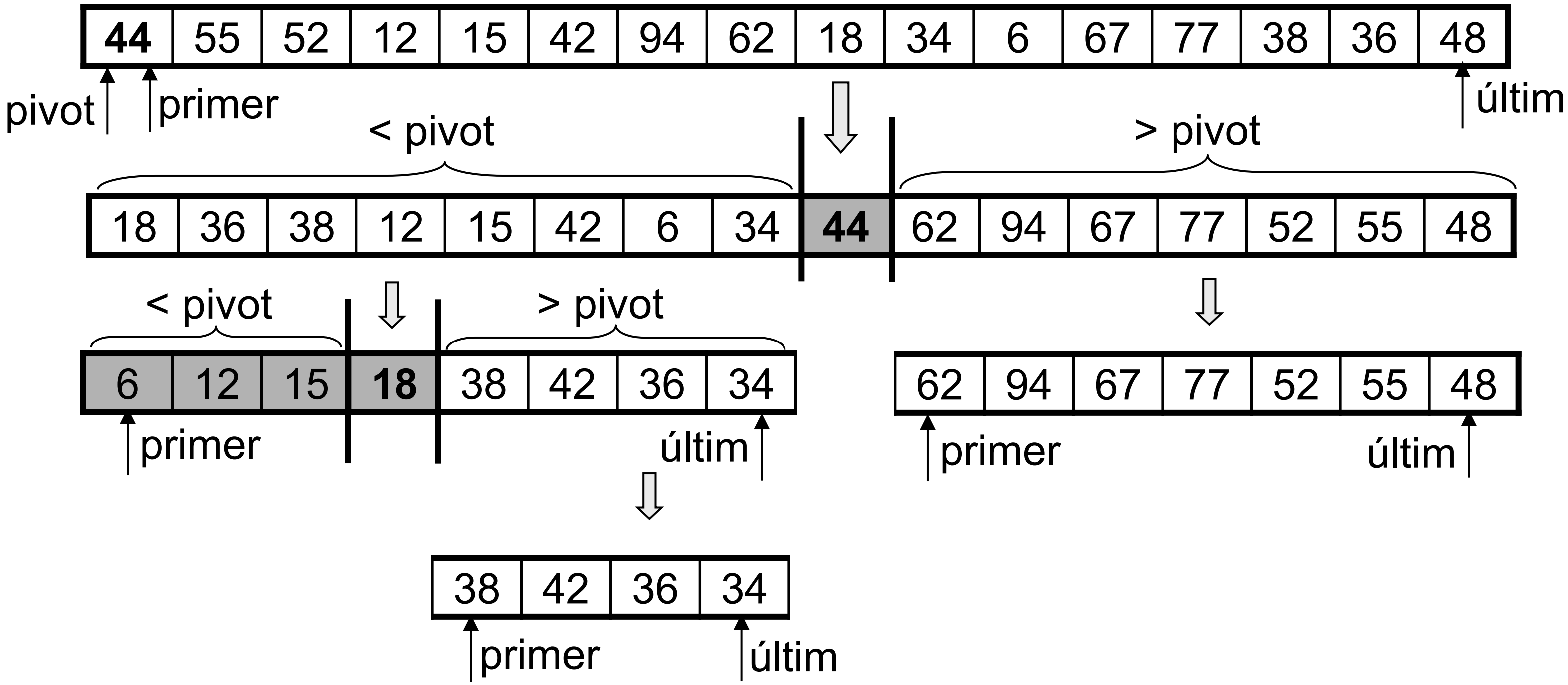
# Algorismes recursius: ordenació ràpida – Quicksort



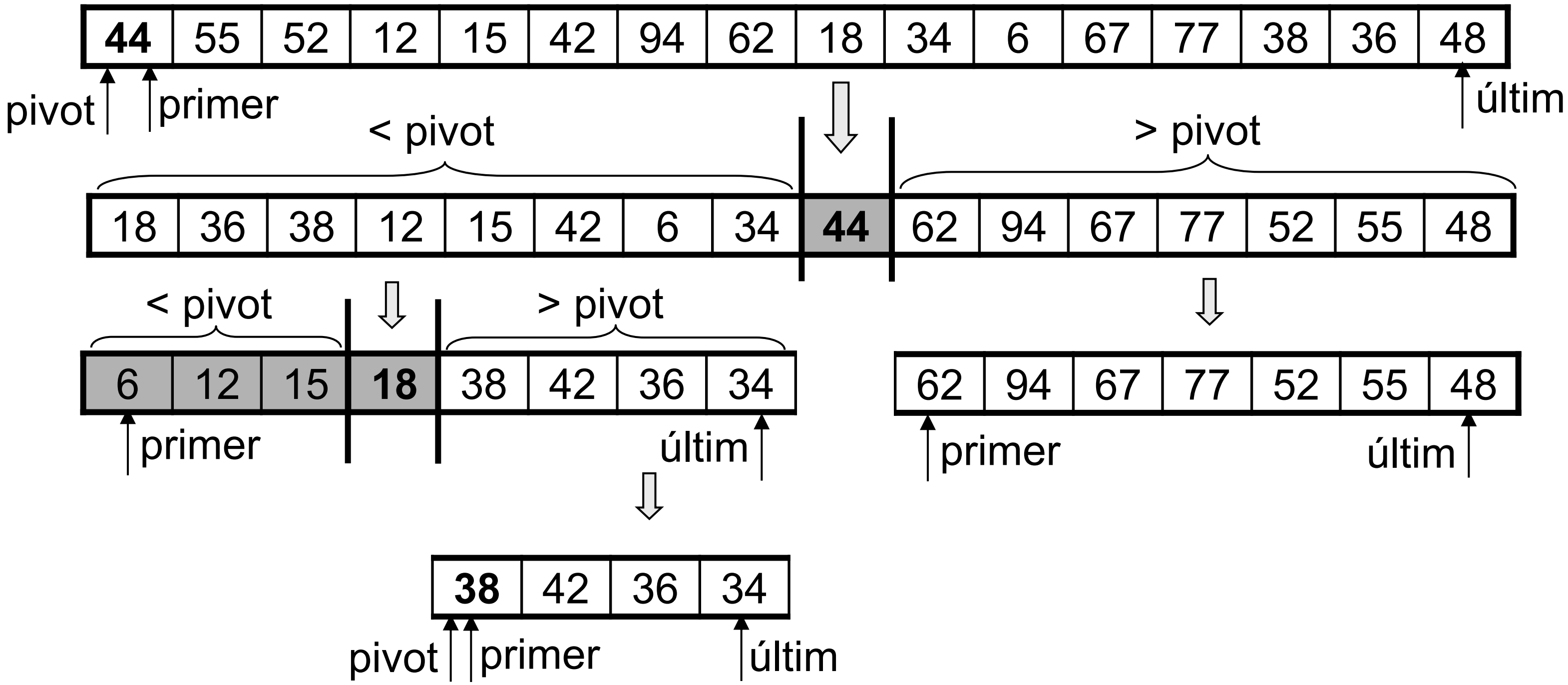
# Algorismes recursius: ordenació ràpida – Quicksort



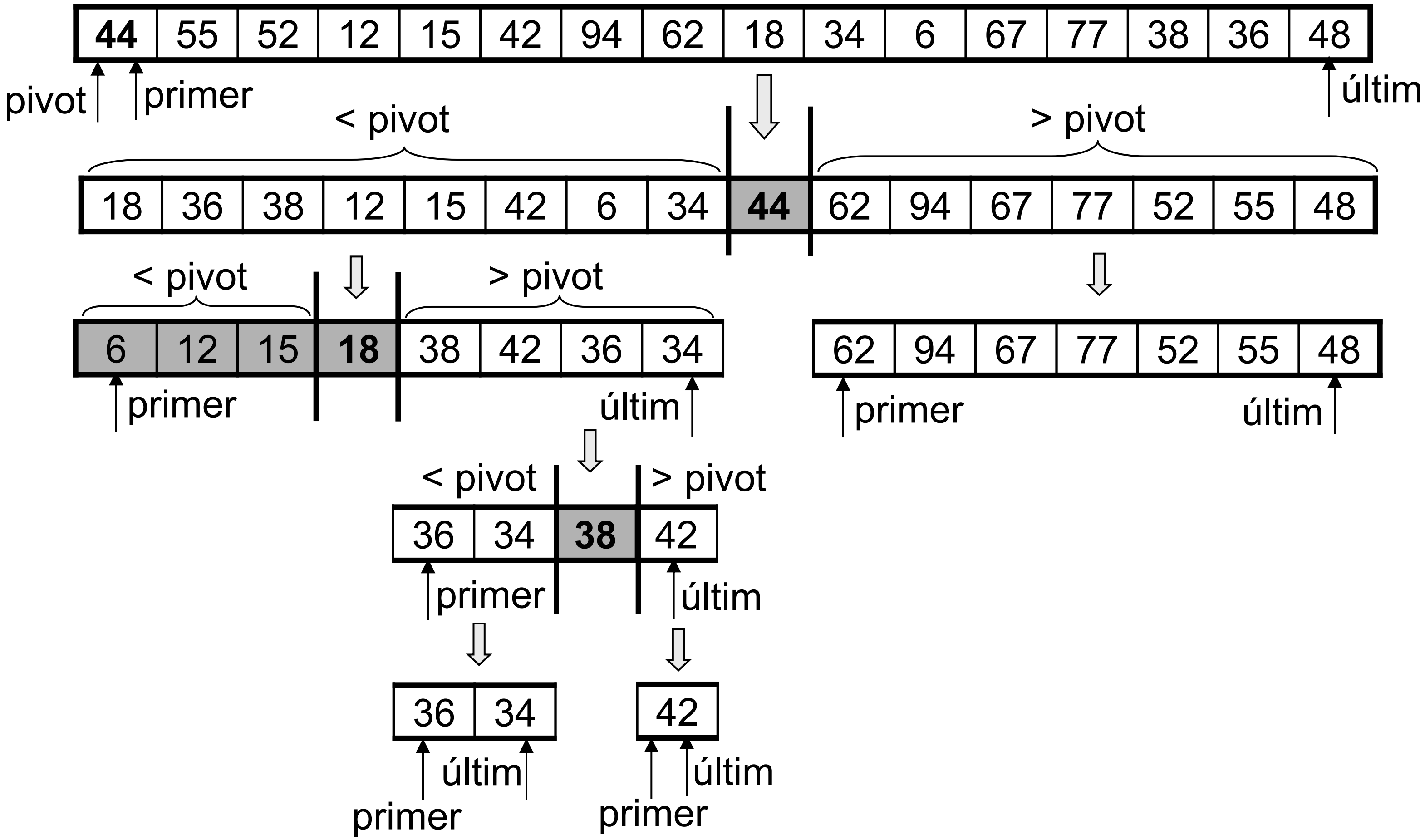
# Algorismes recursius: ordenació ràpida – Quicksort



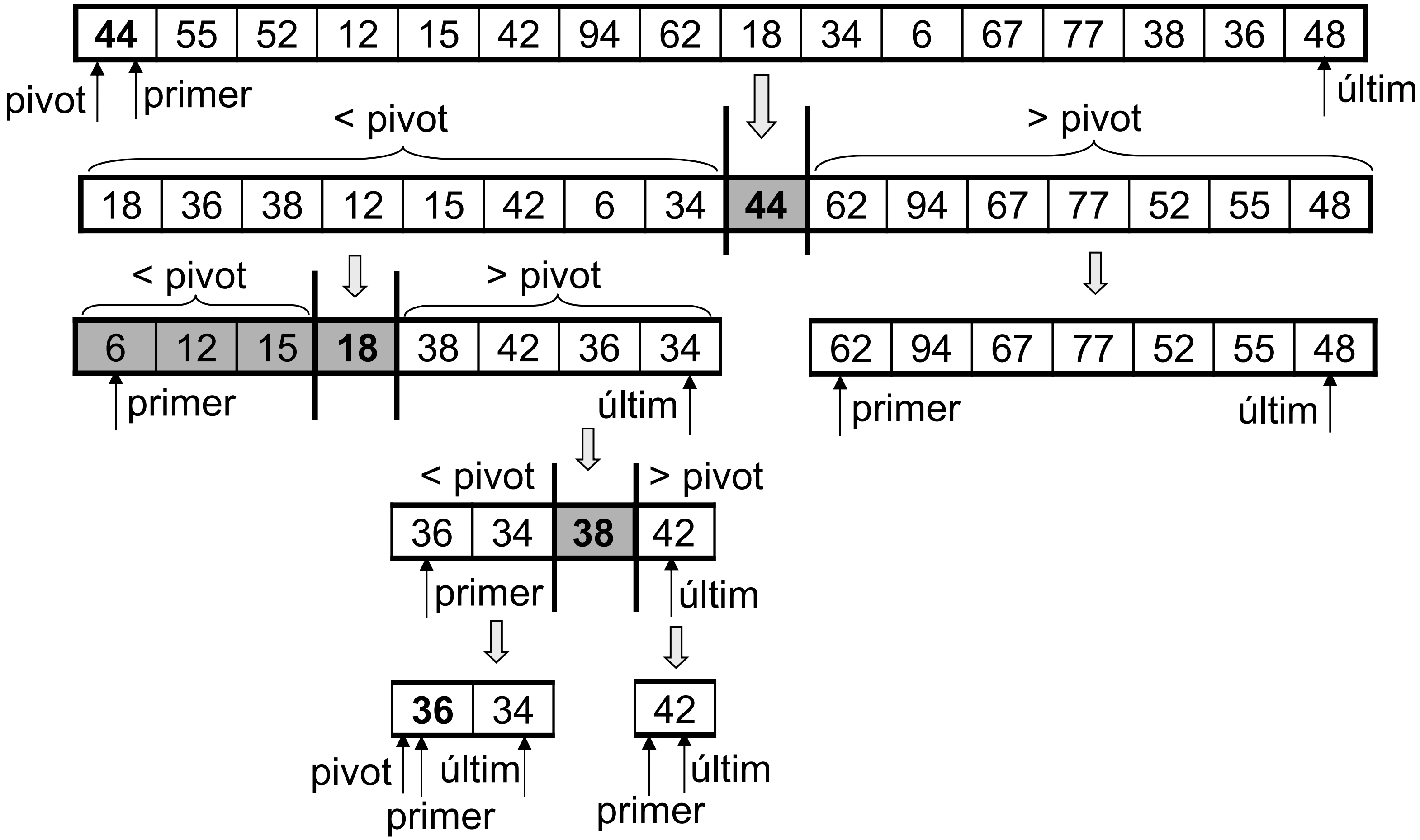
# Algorismes recursius: ordenació ràpida – Quicksort



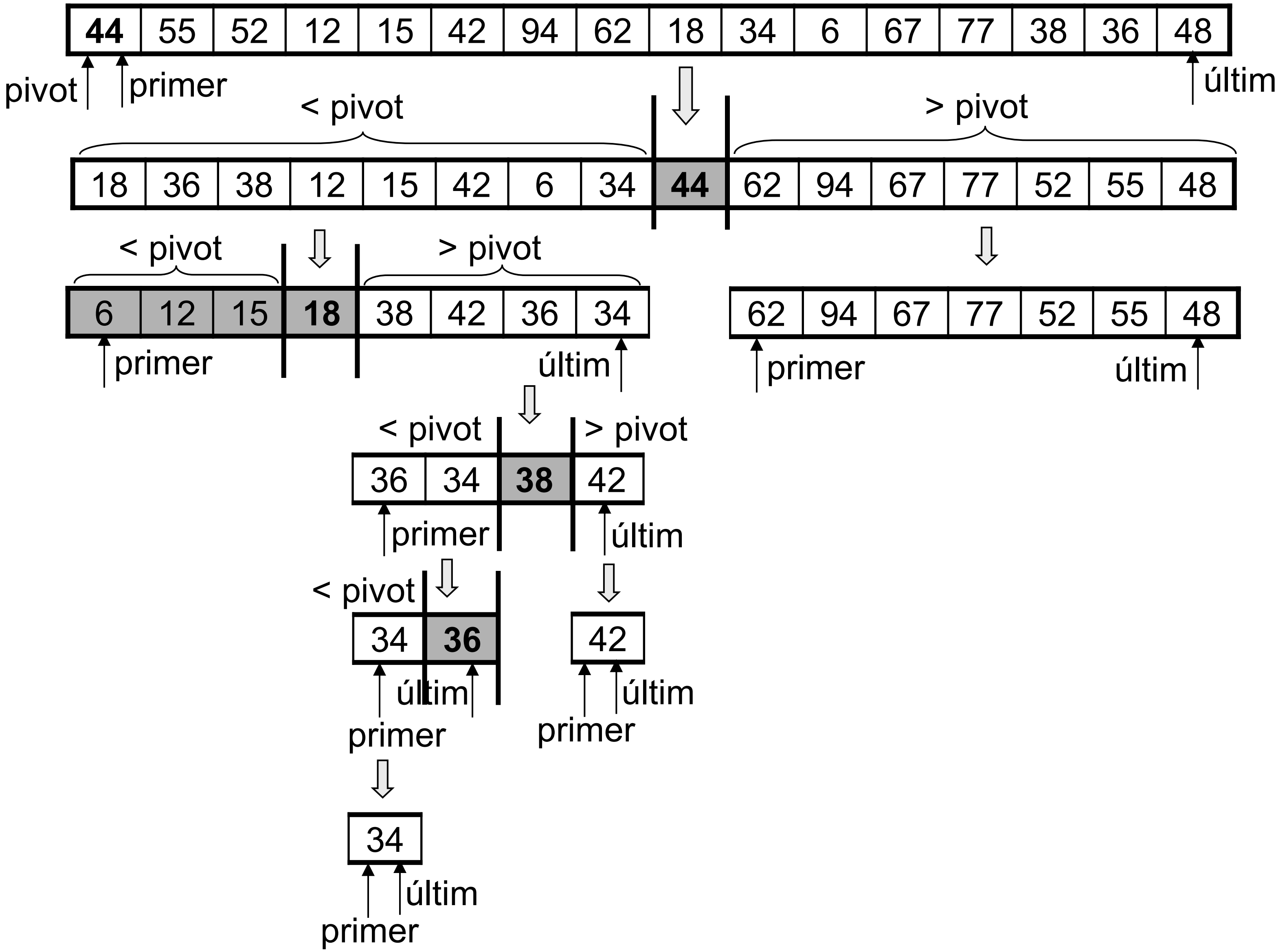
# Algorismes recursius: ordenació ràpida – Quicksort



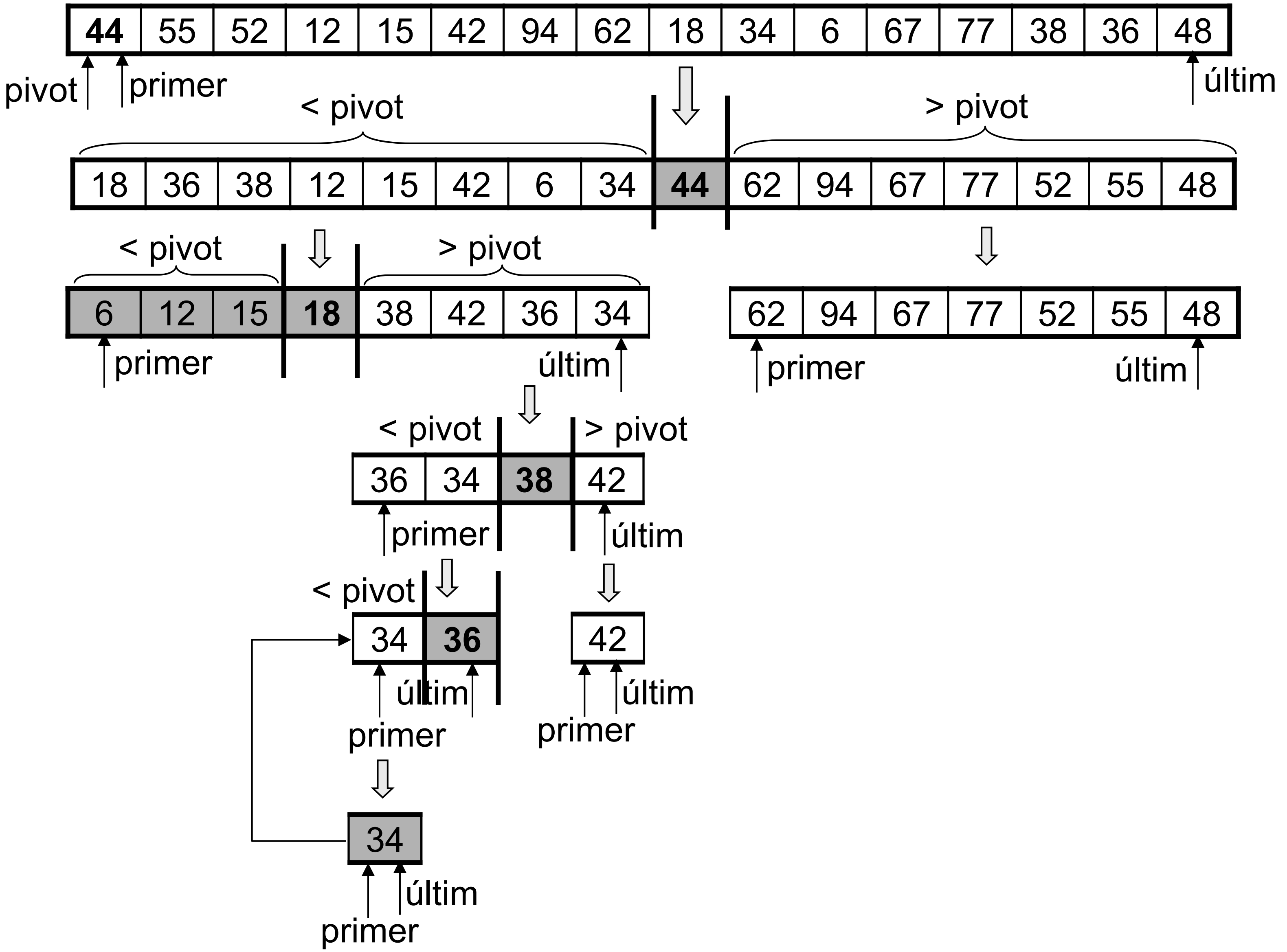
# Algorismes recursius: ordenació ràpida – Quicksort



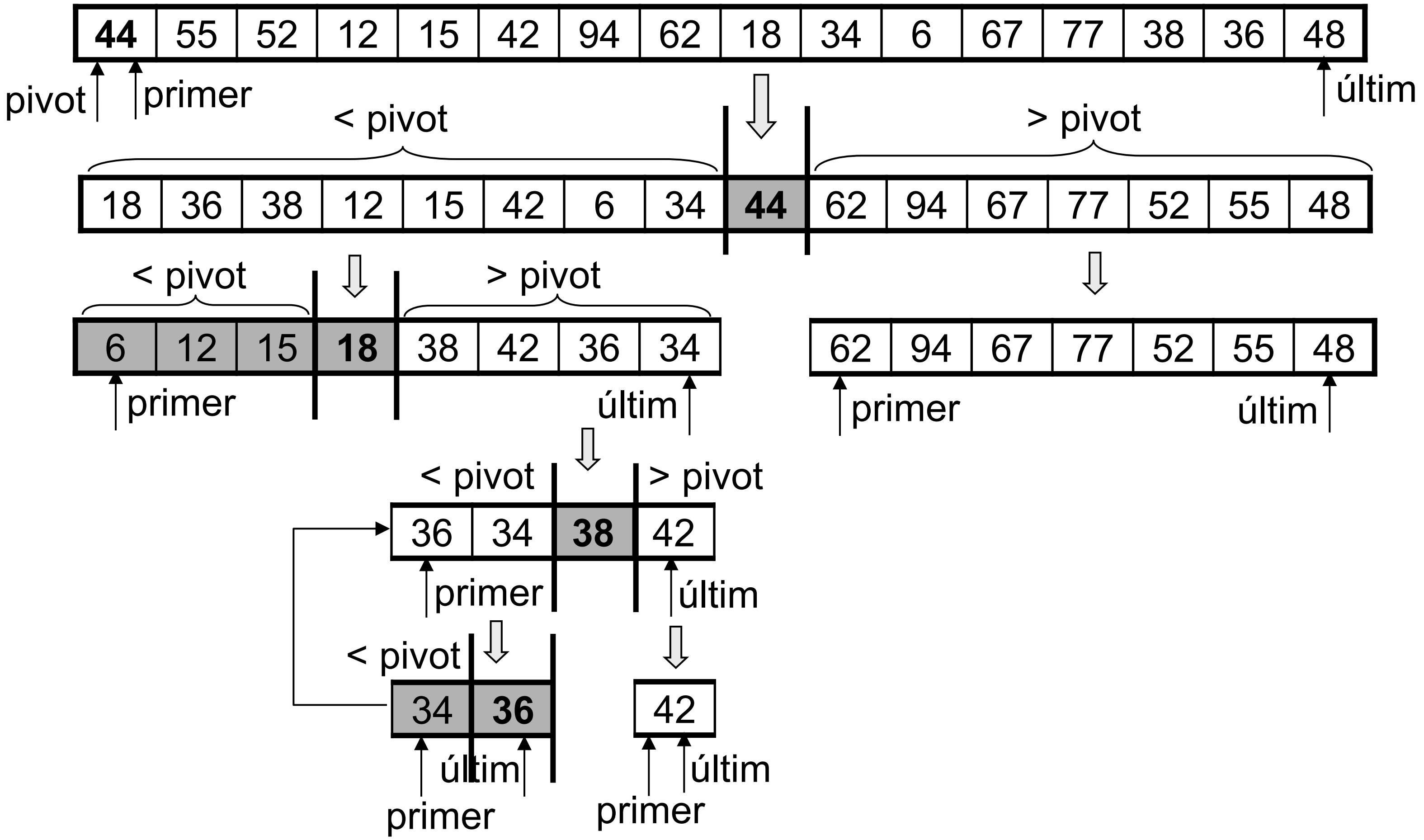
# Algorismes recursius: ordenació ràpida – Quicksort



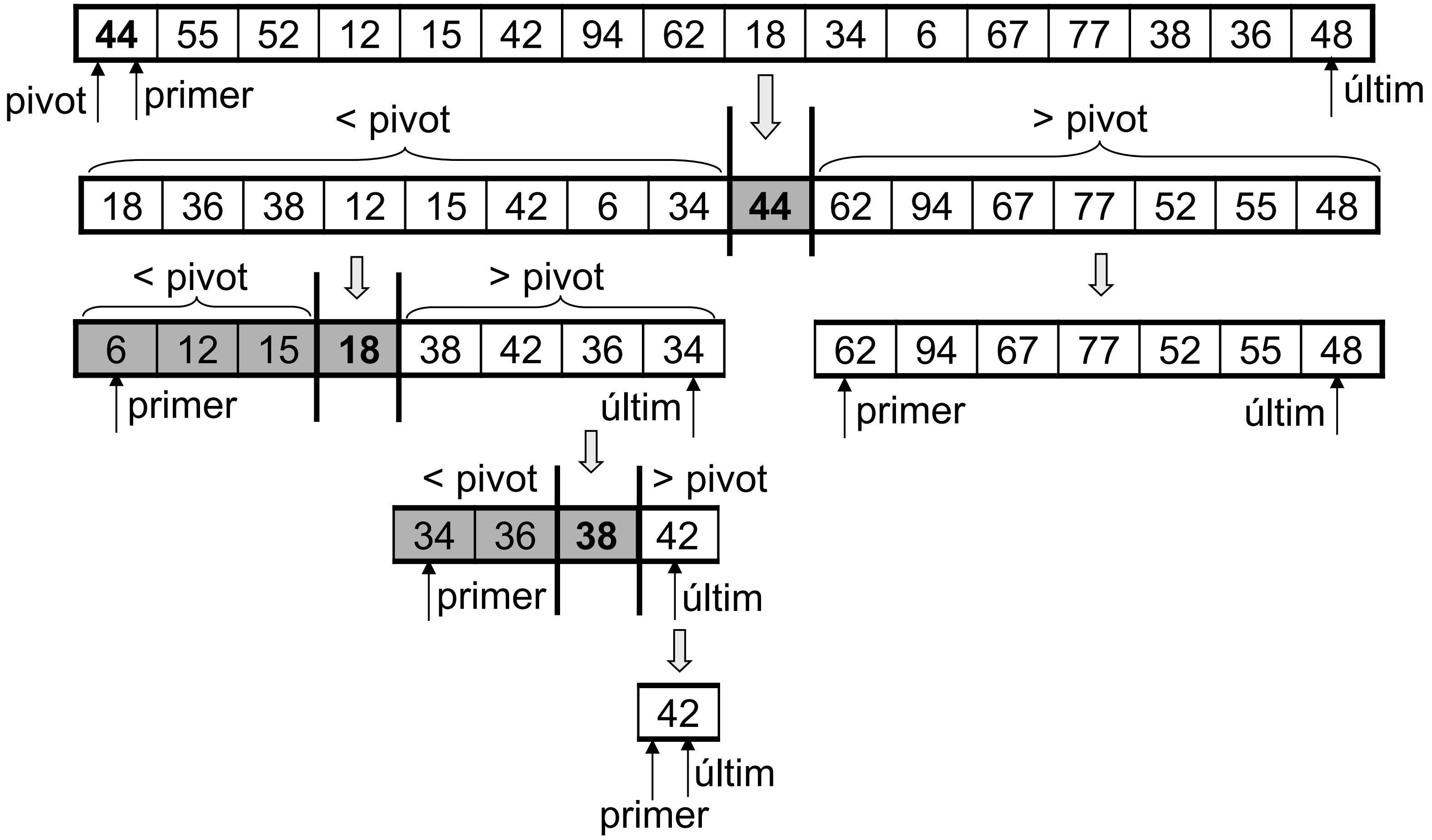
# Algorismes recursius: ordenació ràpida – Quicksort



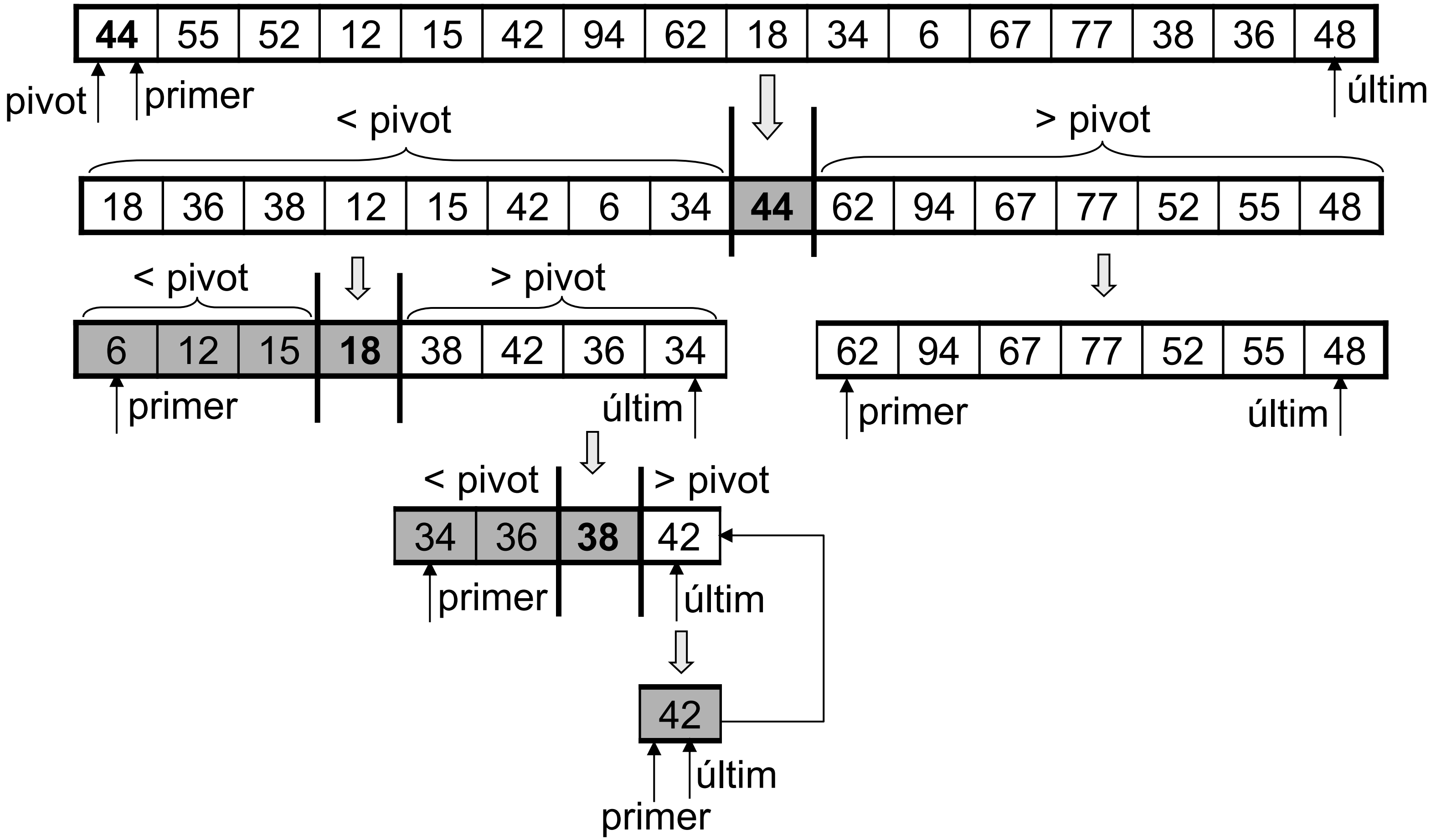
# Algorismes recursius: ordenació ràpida – Quicksort



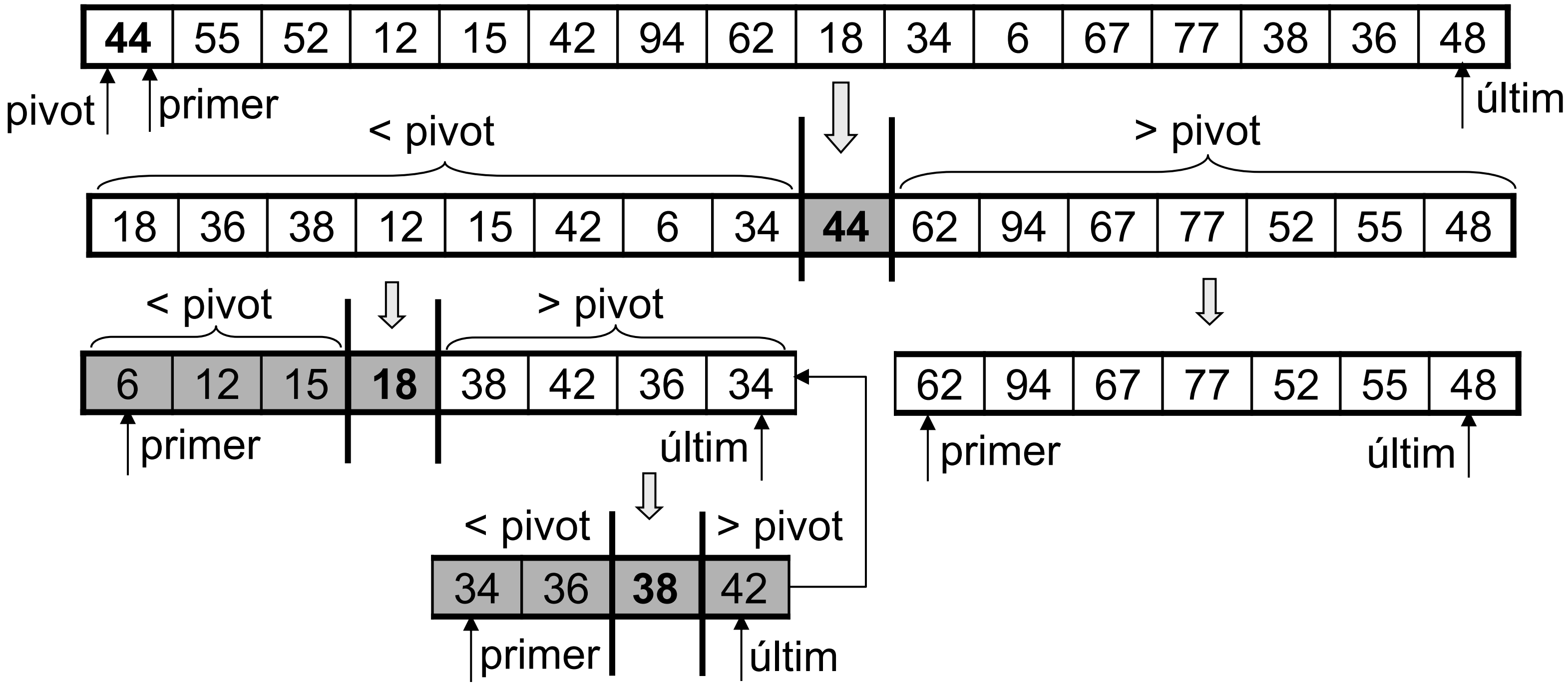
# Algorismes recursius: ordenació ràpida – Quicksort



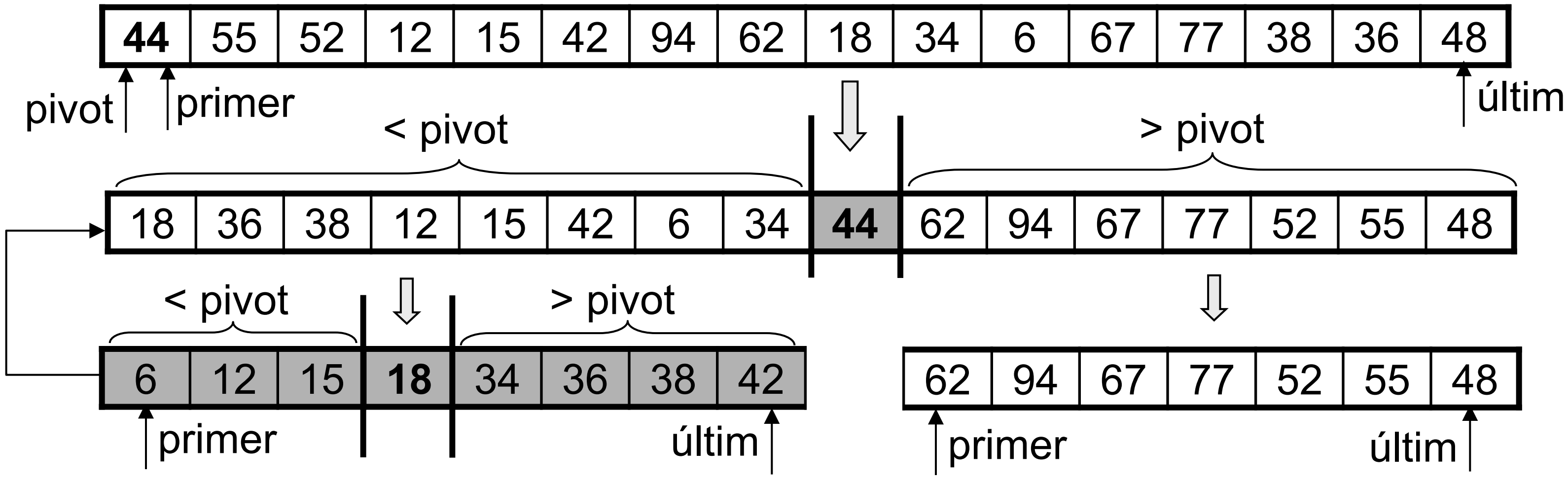
# Algorismes recursius: ordenació ràpida – Quicksort



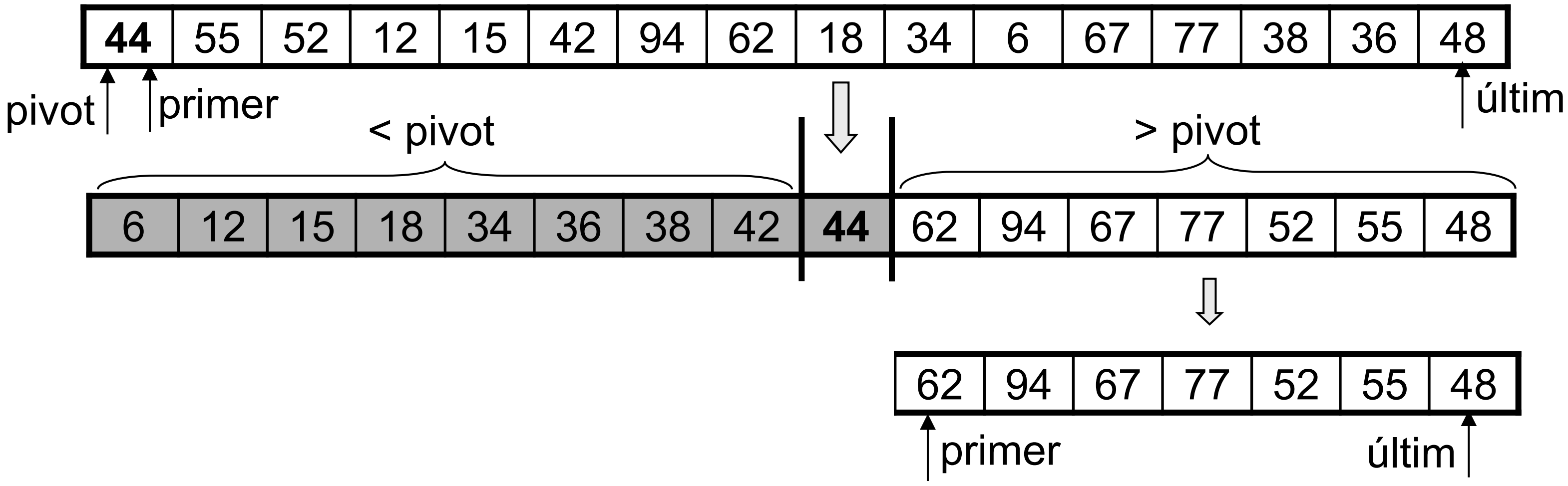
# Algorismes recursius: ordenació ràpida – Quicksort



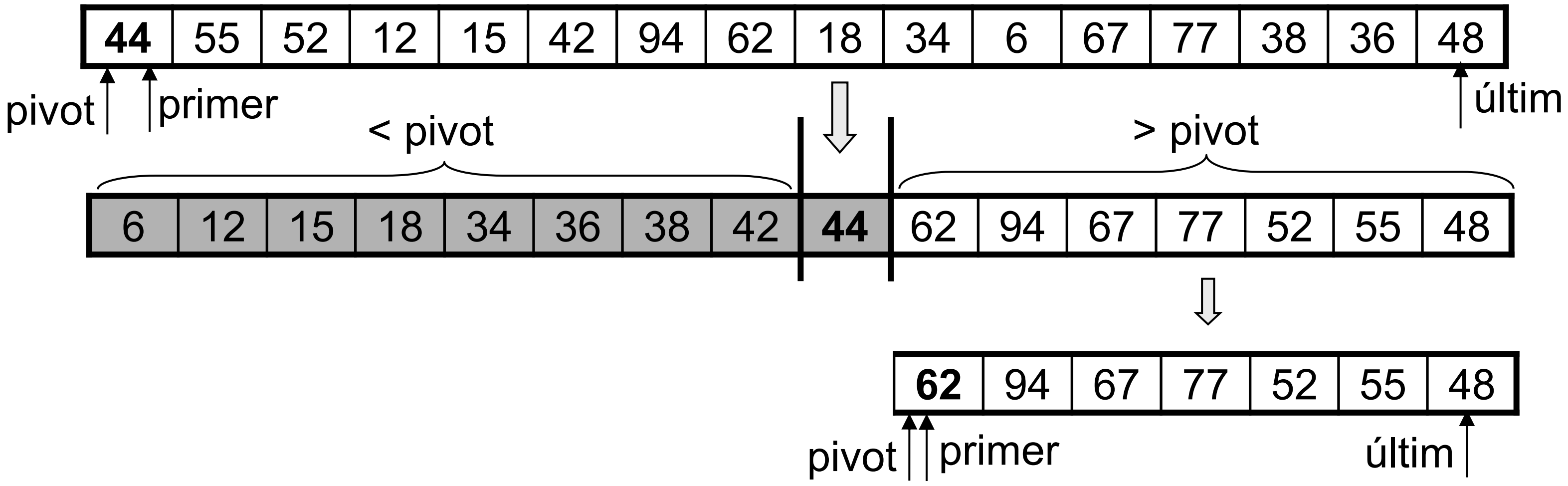
# Algorismes recursius: ordenació ràpida – Quicksort



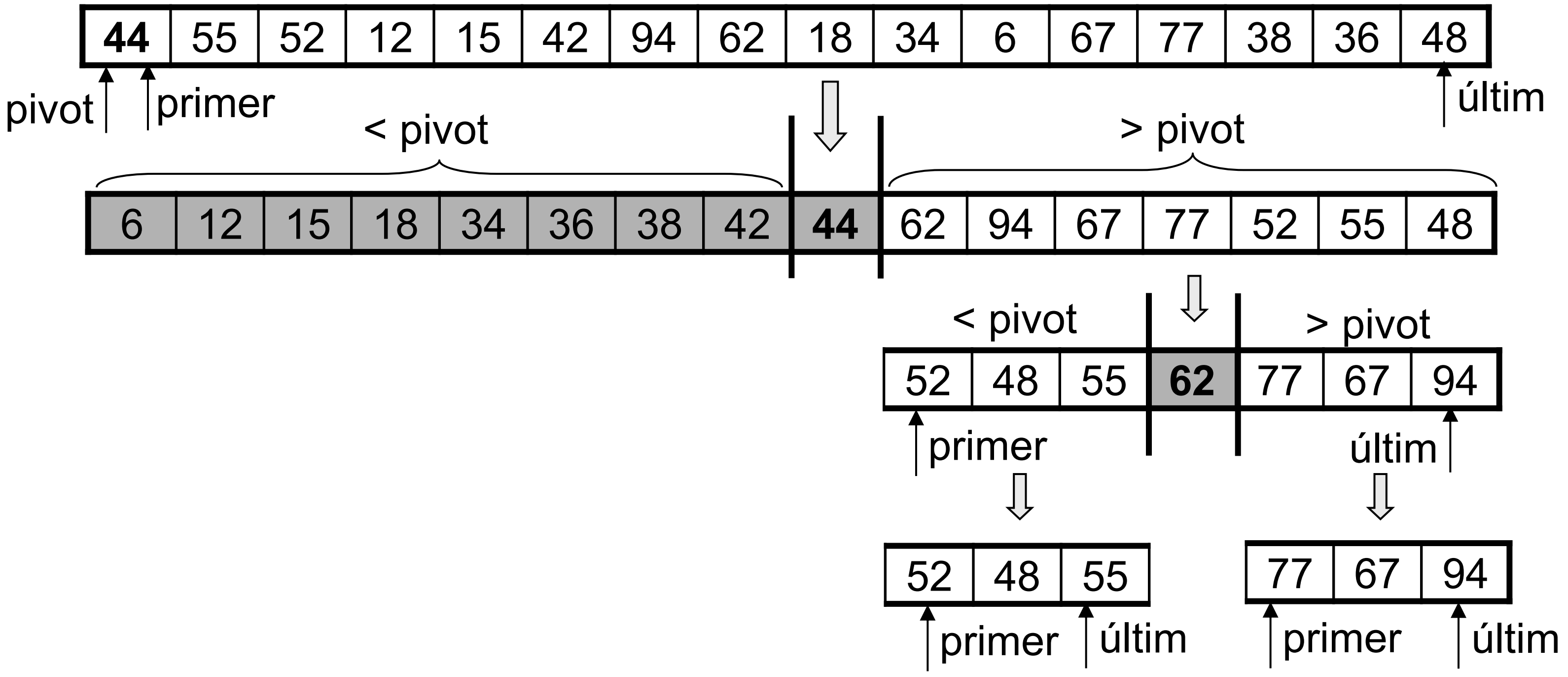
# Algorismes recursius: ordenació ràpida – Quicksort



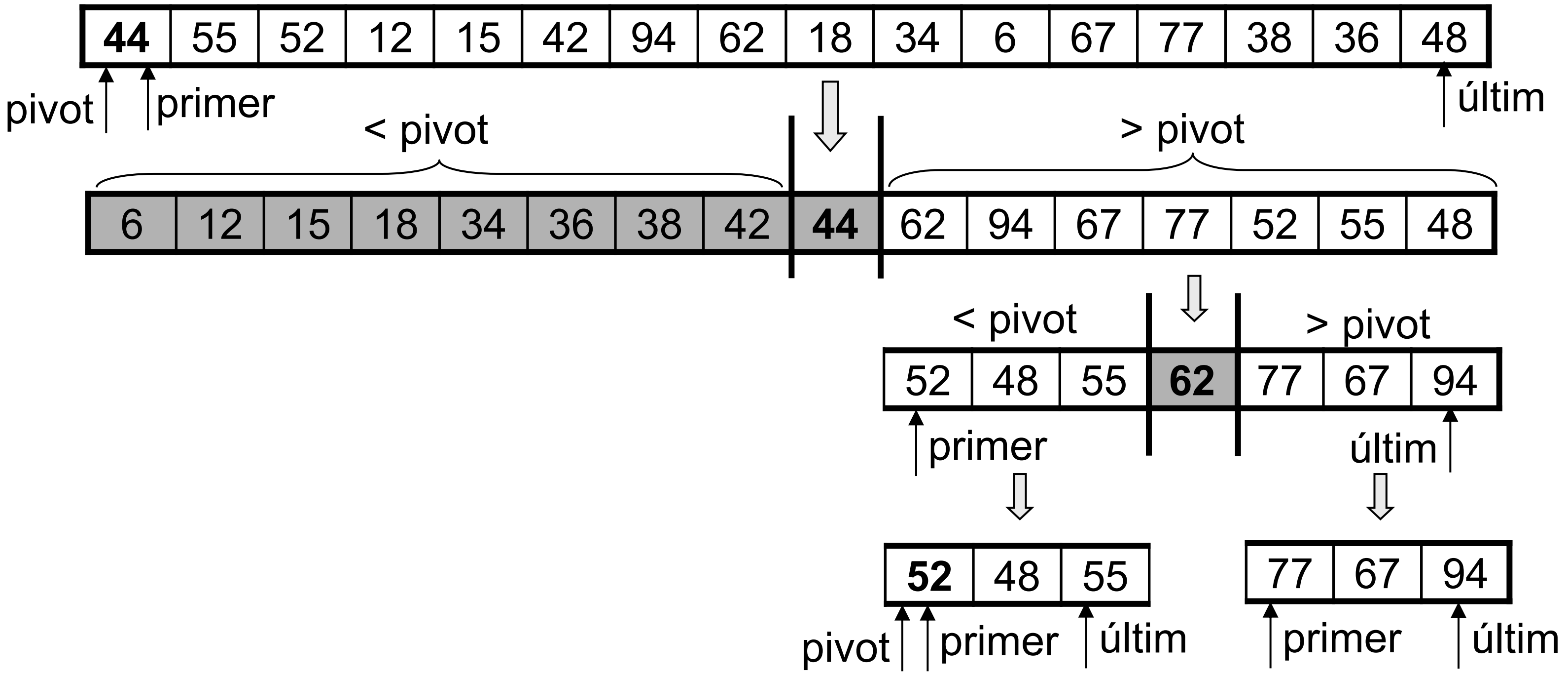
# Algorismes recursius: ordenació ràpida – Quicksort



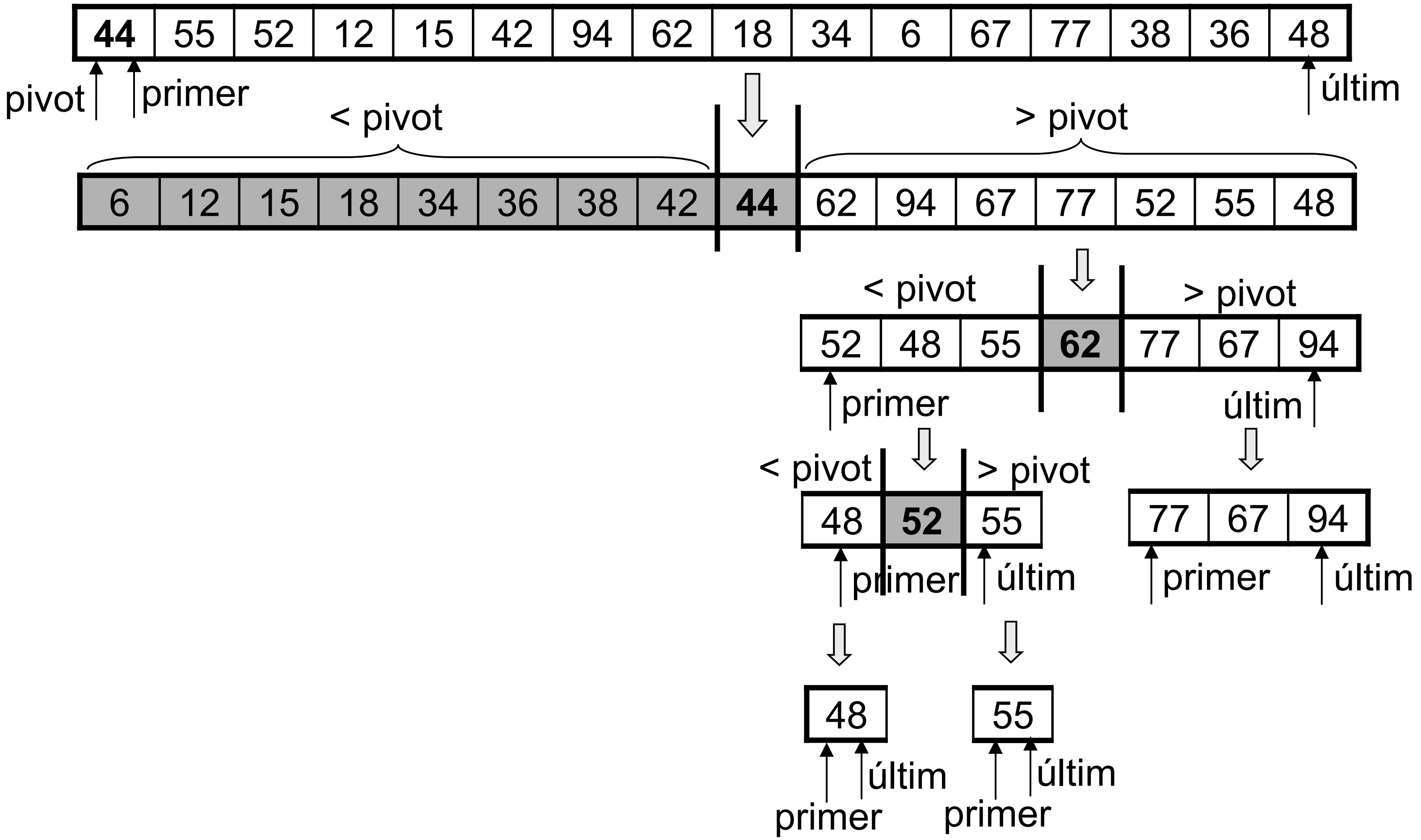
# Algorismes recursius: ordenació ràpida – Quicksort



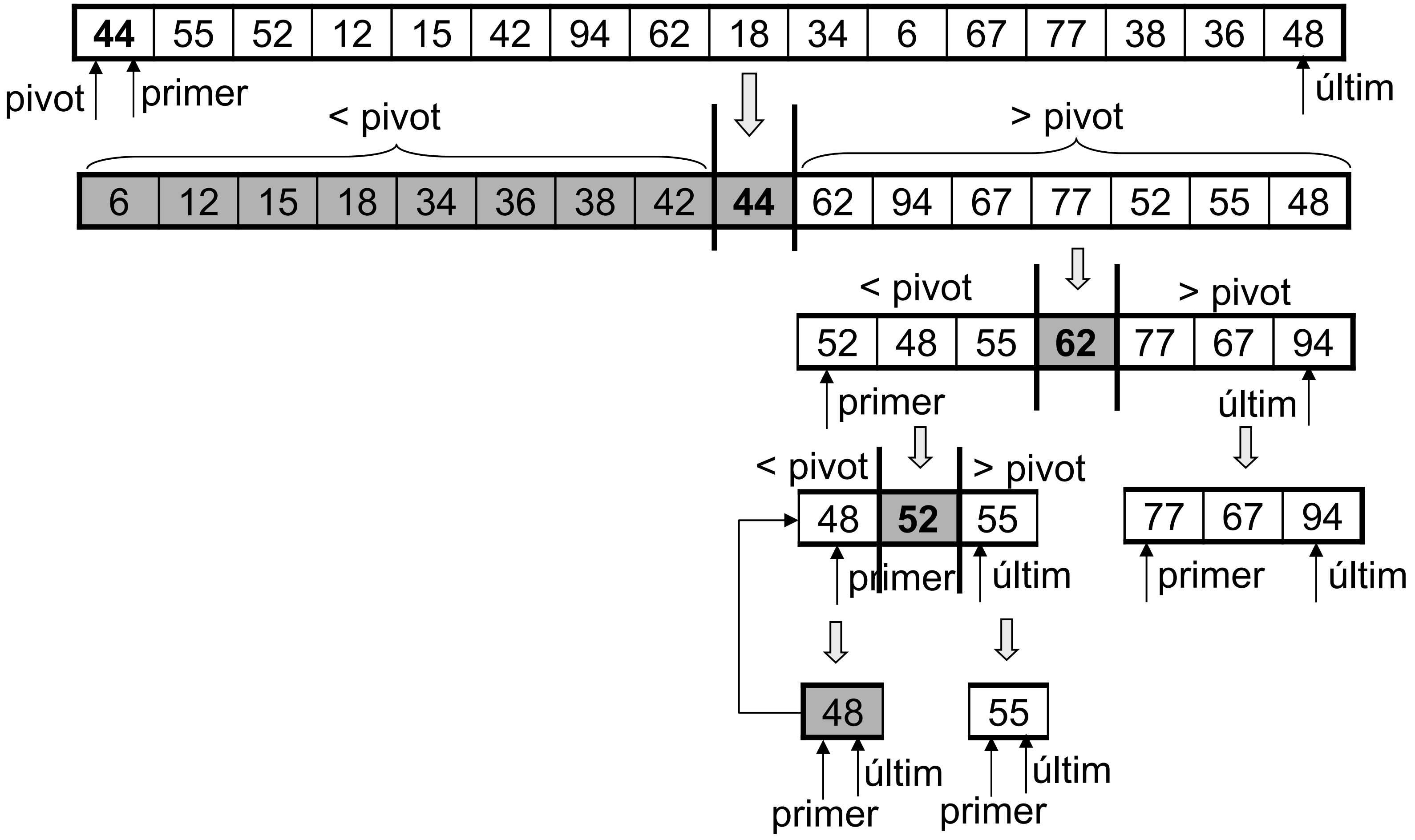
# Algorismes recursius: ordenació ràpida – Quicksort



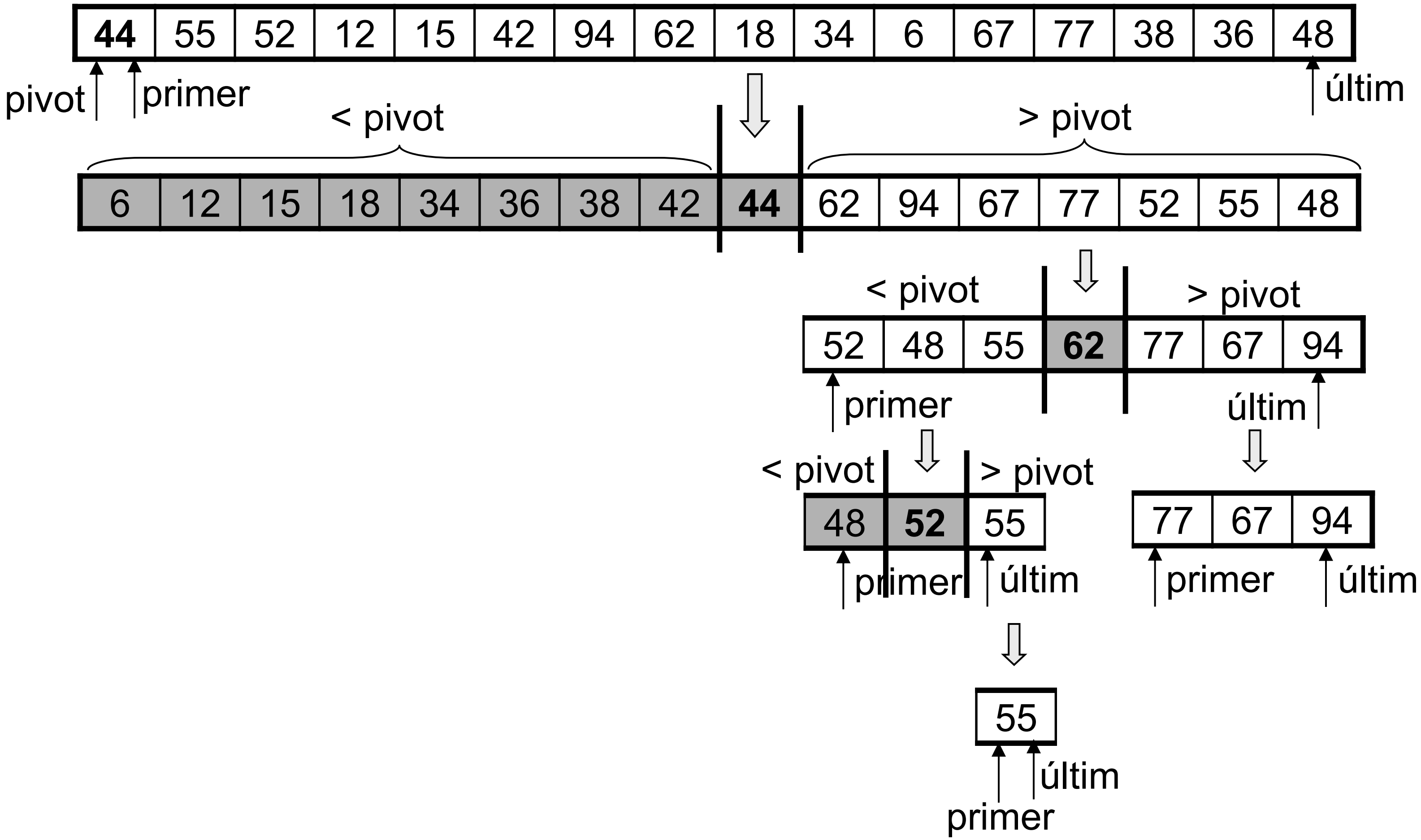
# Algorismes recursius: ordenació ràpida – Quicksort



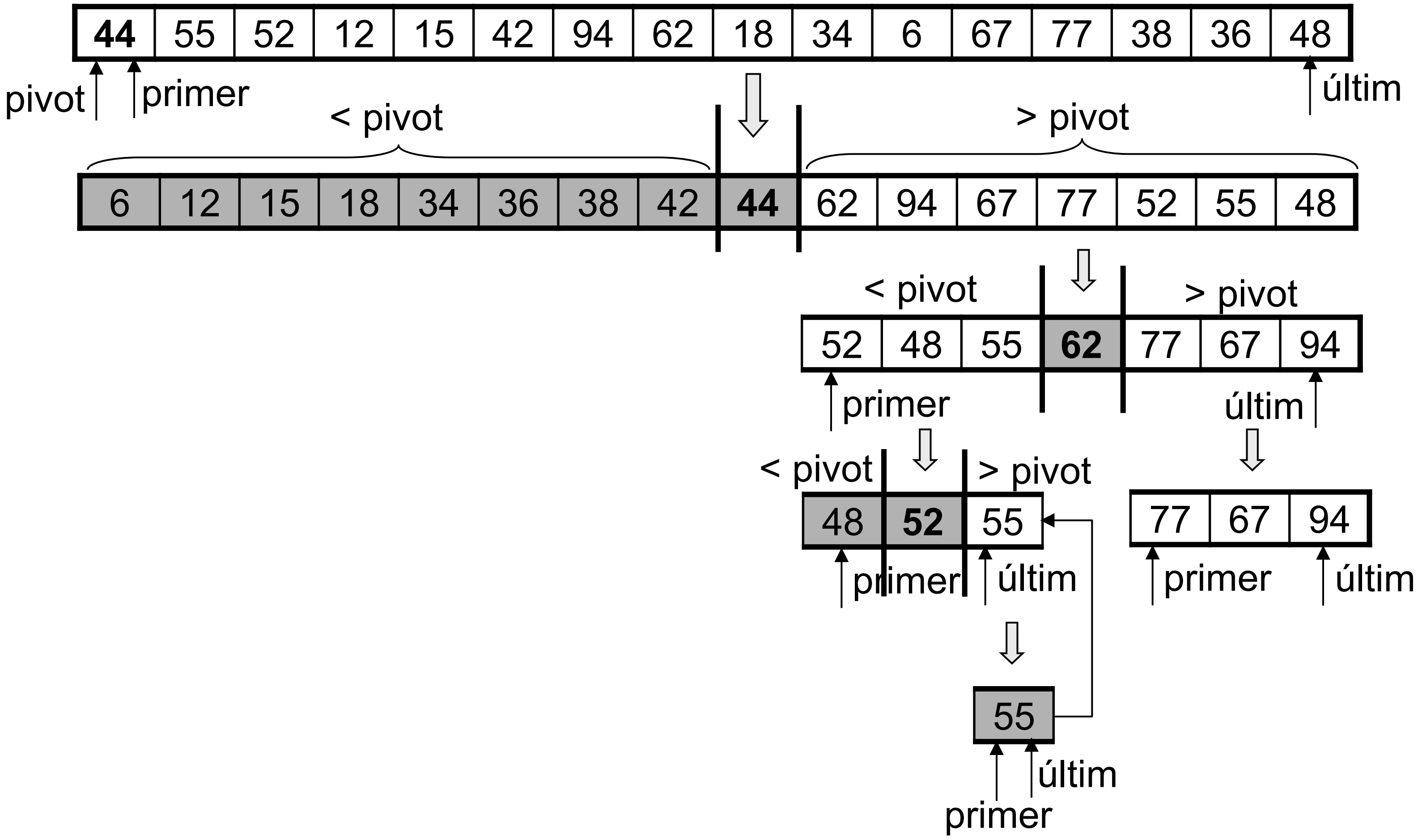
# Algorismes recursius: ordenació ràpida – Quicksort



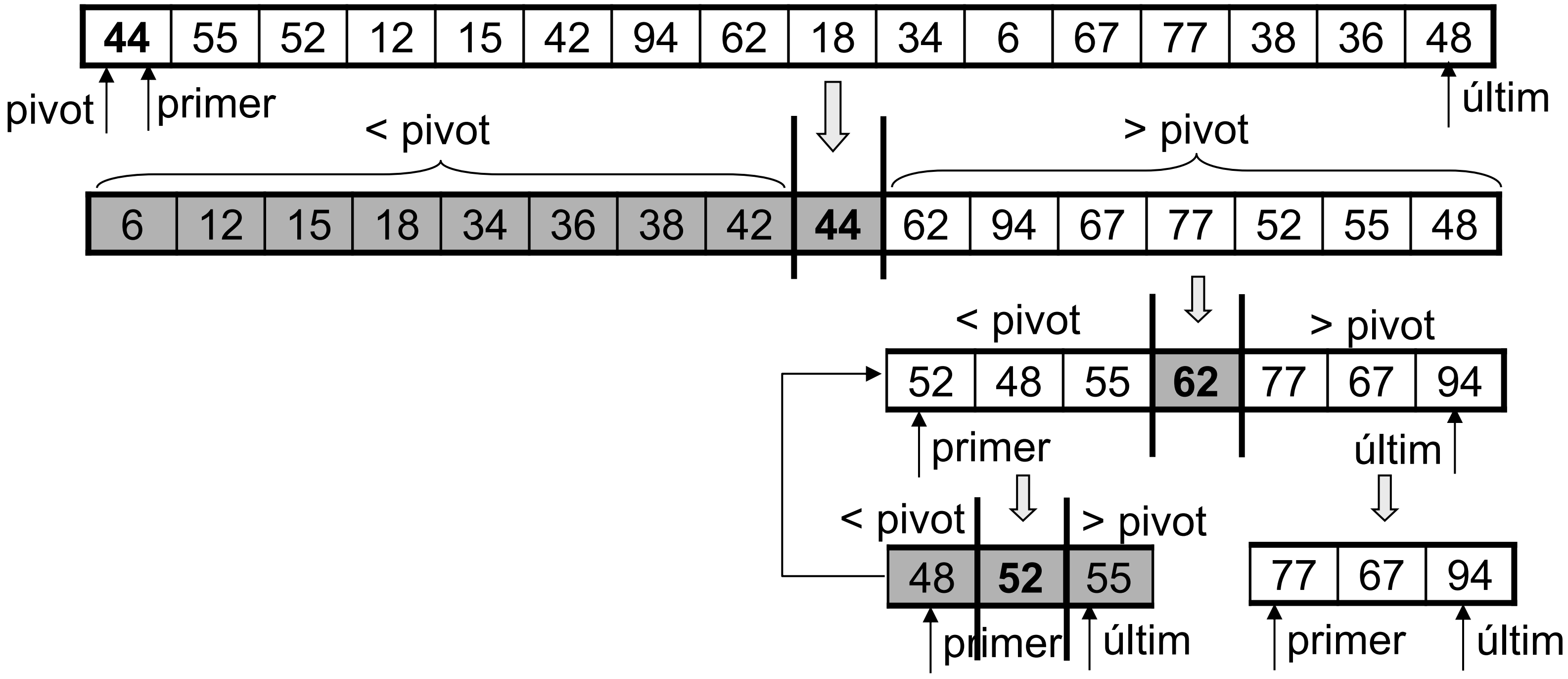
# Algorismes recursius: ordenació ràpida – Quicksort



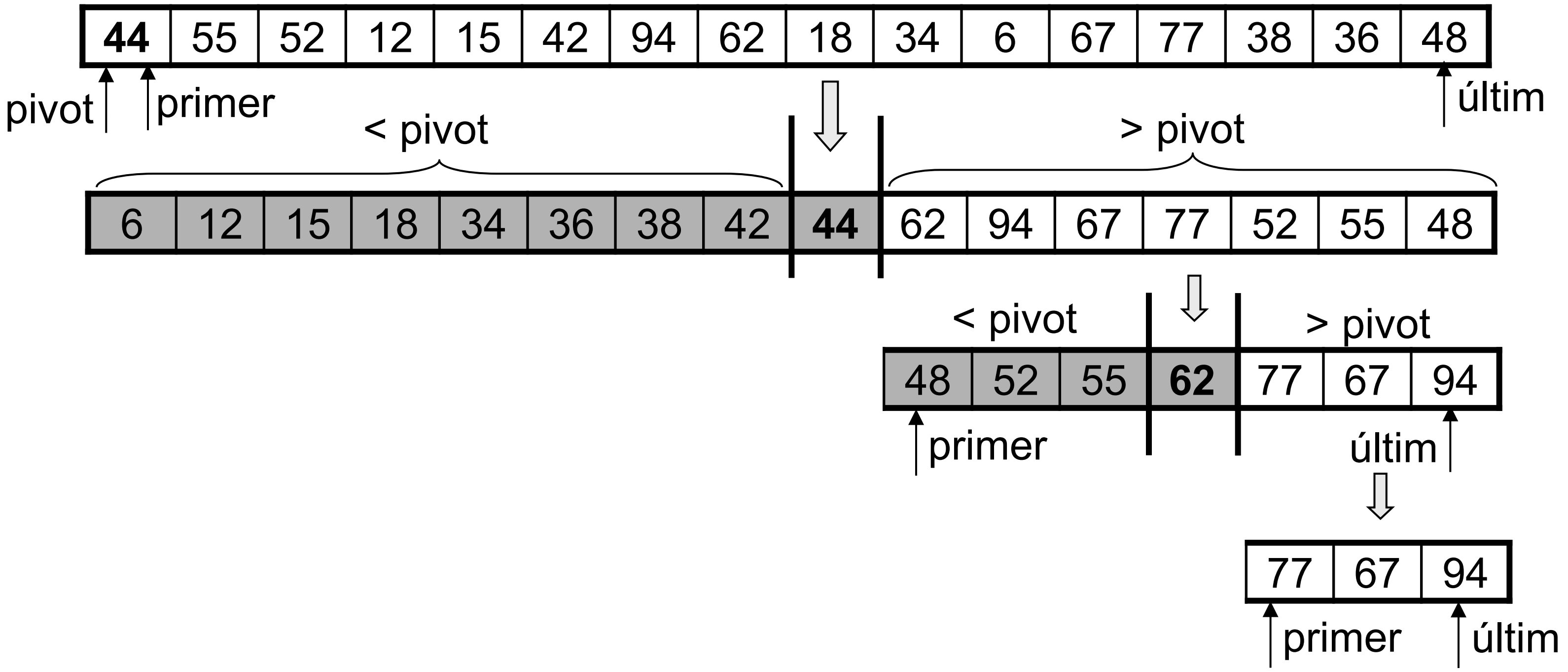
# Algorismes recursius: ordenació ràpida – Quicksort



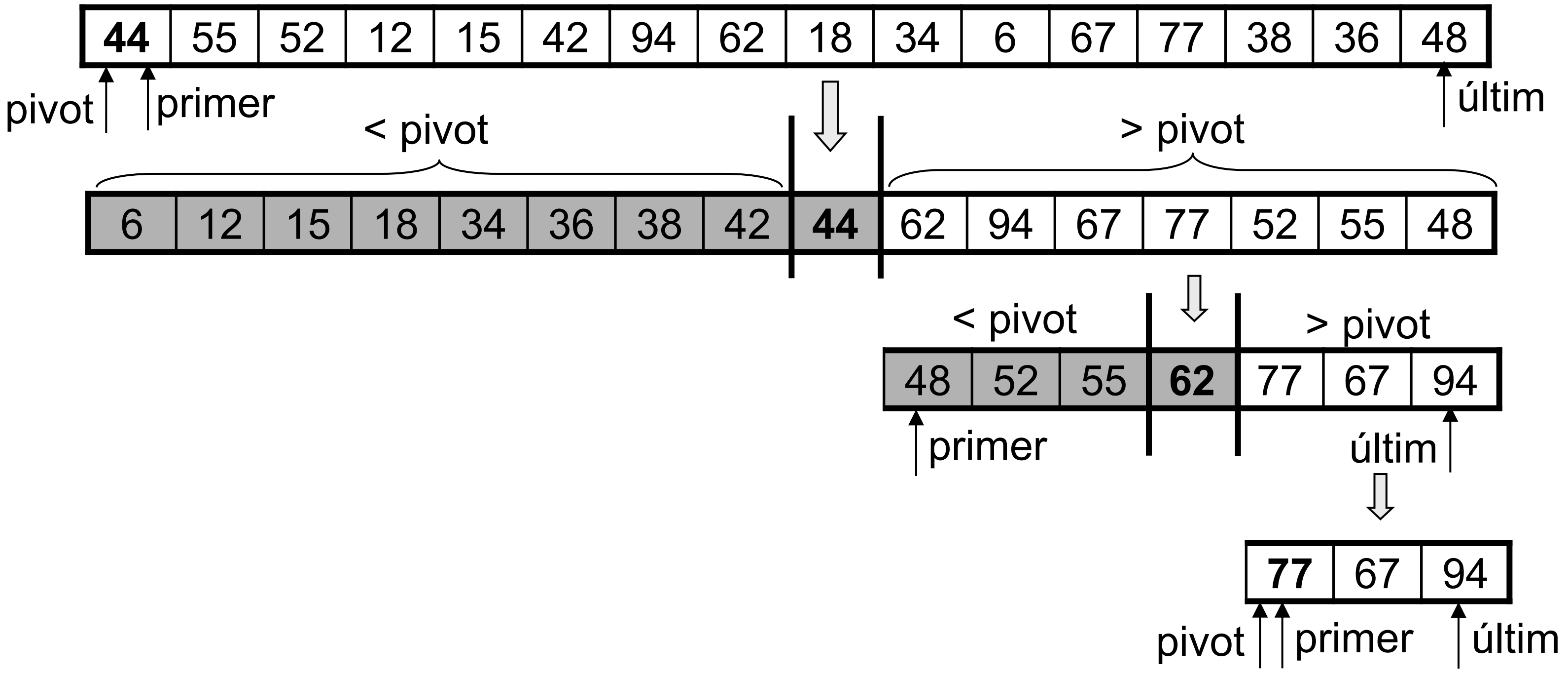
# Algorismes recursius: ordenació ràpida – Quicksort



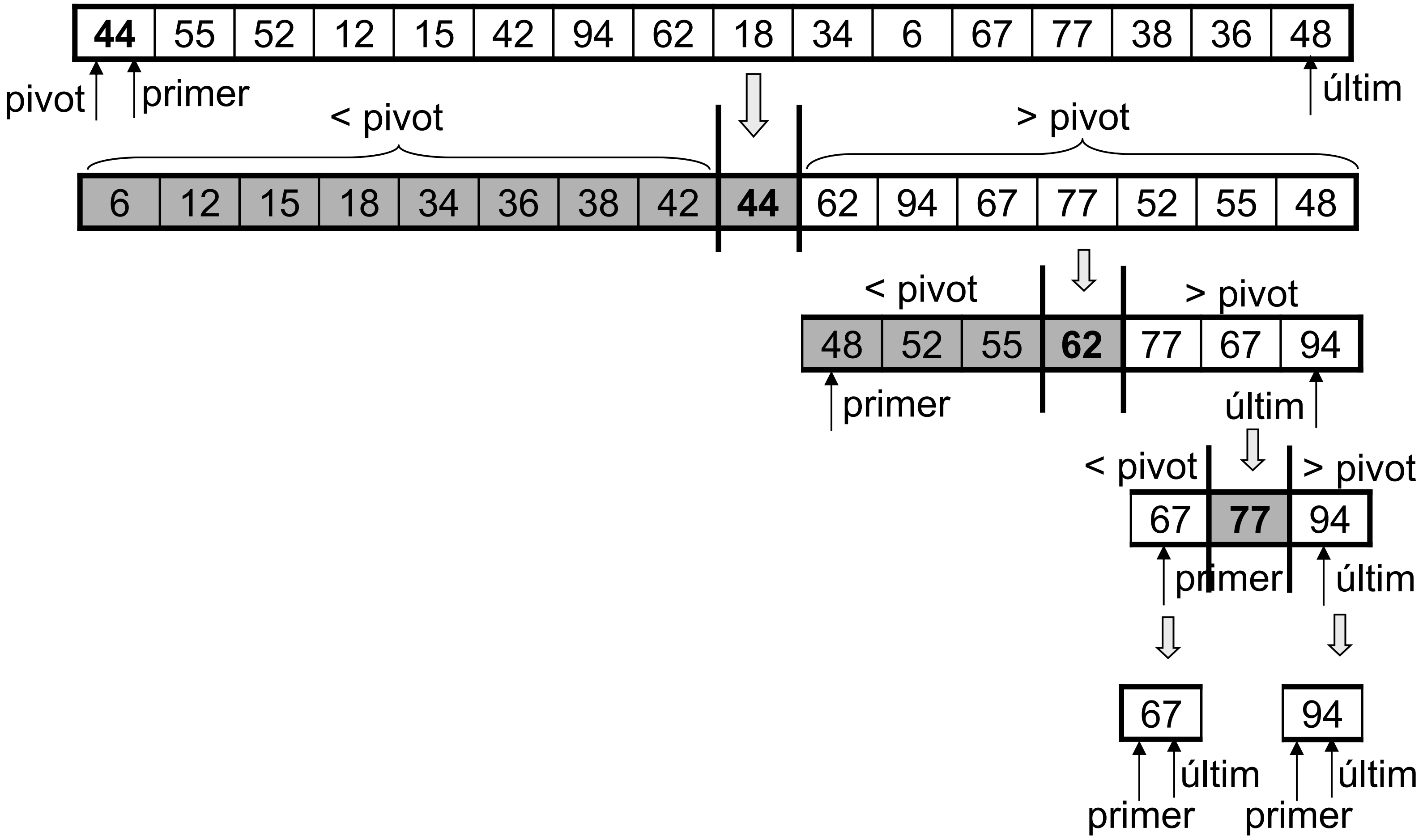
# Algorismes recursius: ordenació ràpida – Quicksort



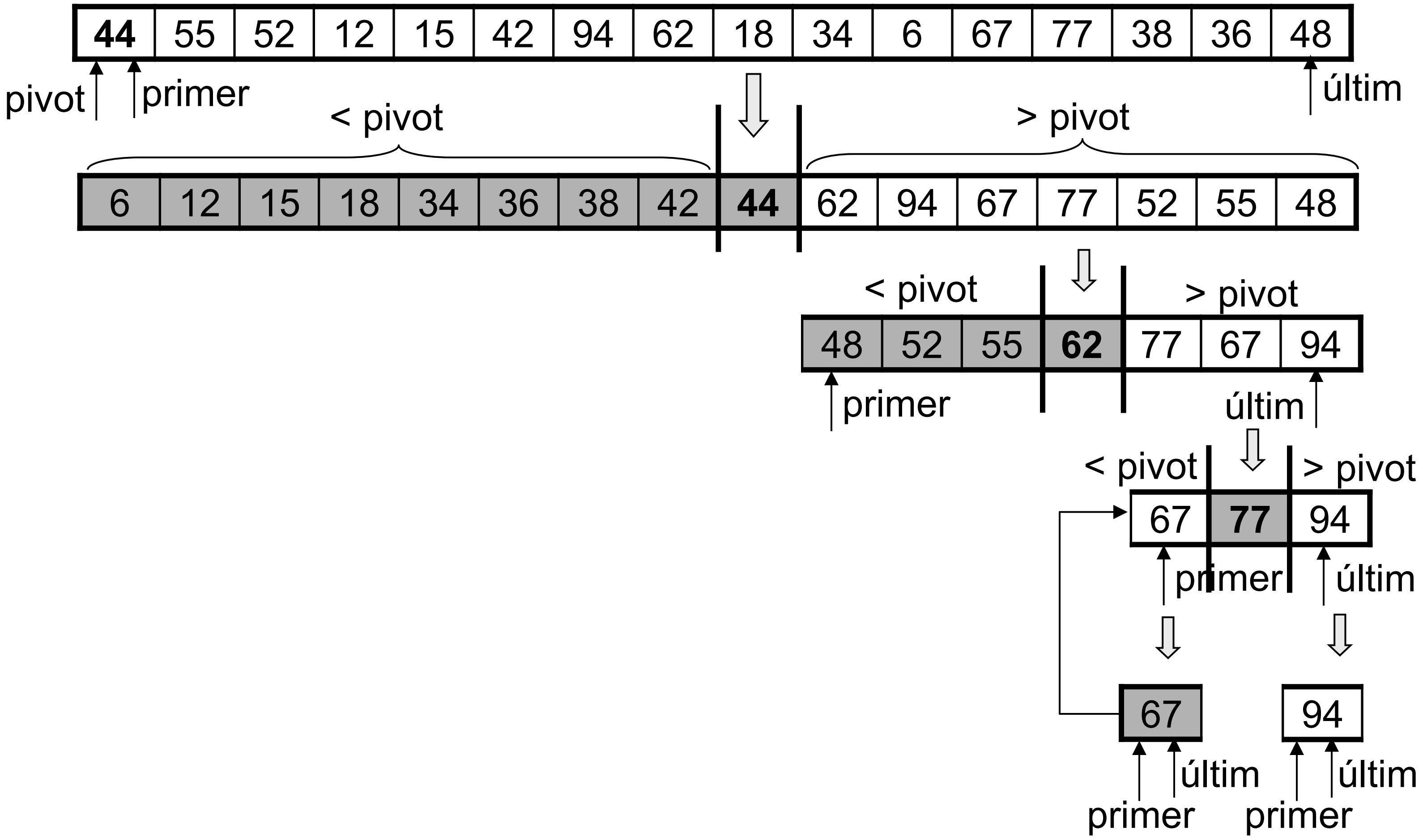
# Algorismes recursius: ordenació ràpida – Quicksort



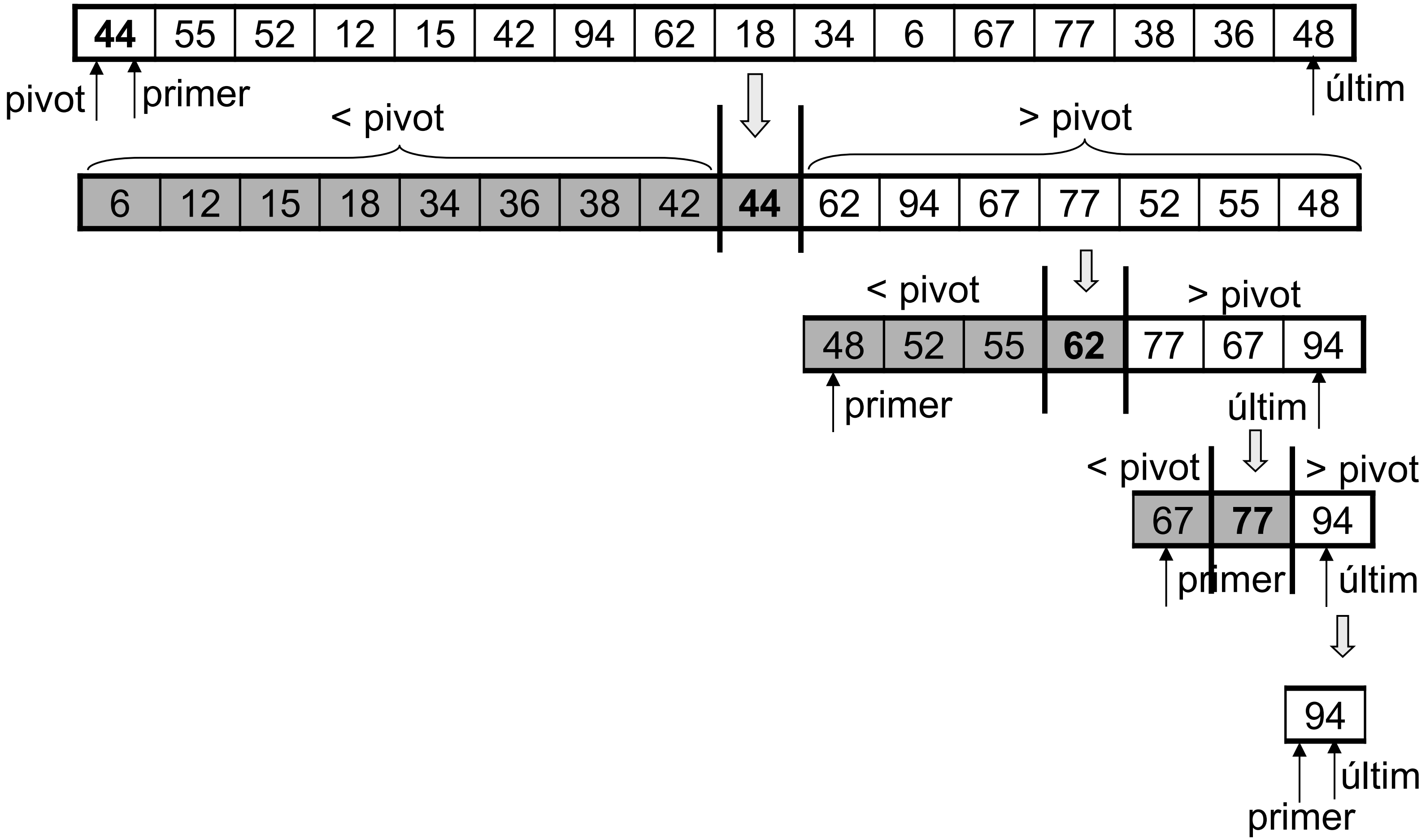
# Algorismes recursius: ordenació ràpida – Quicksort



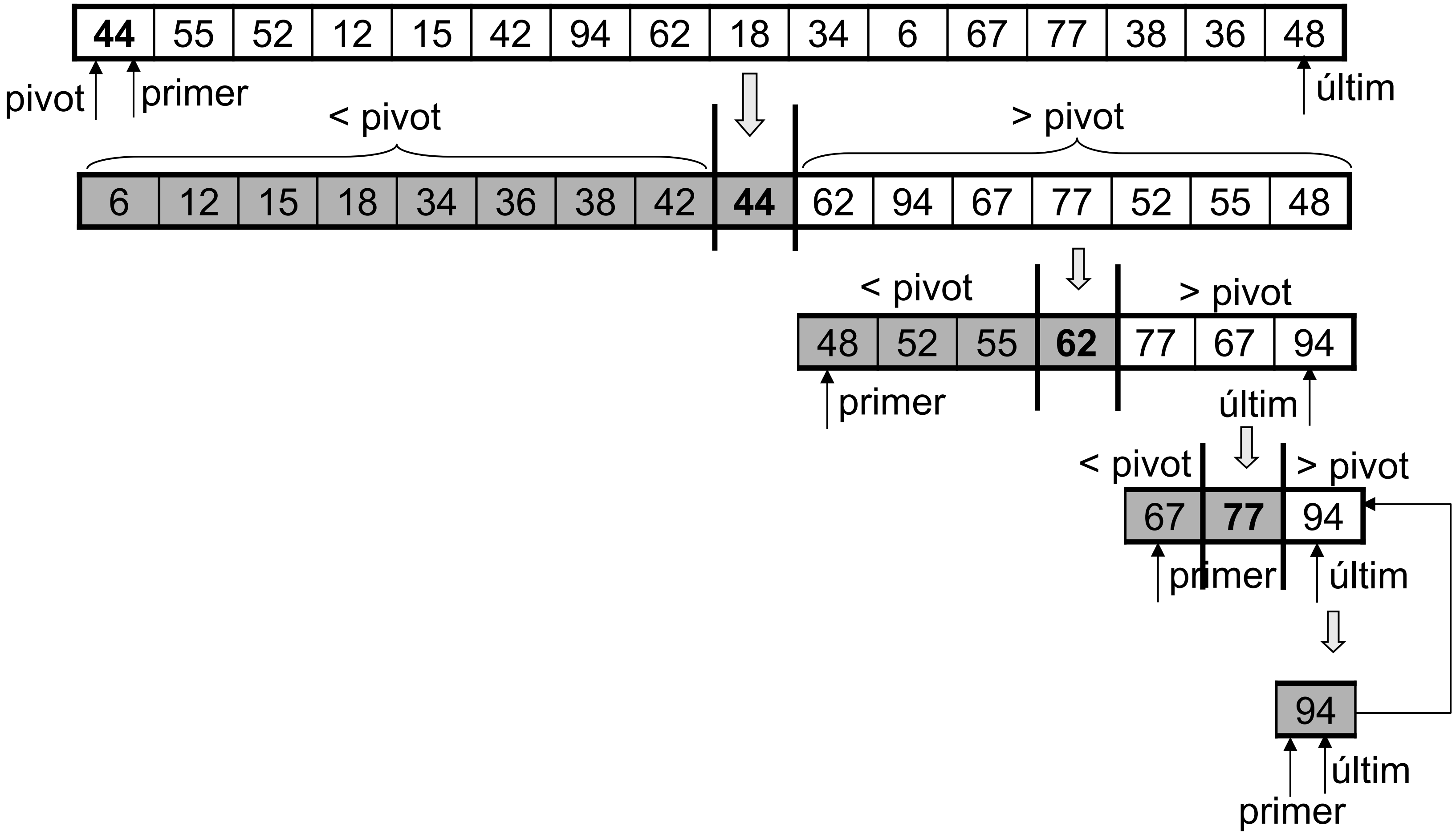
# Algorismes recursius: ordenació ràpida – Quicksort



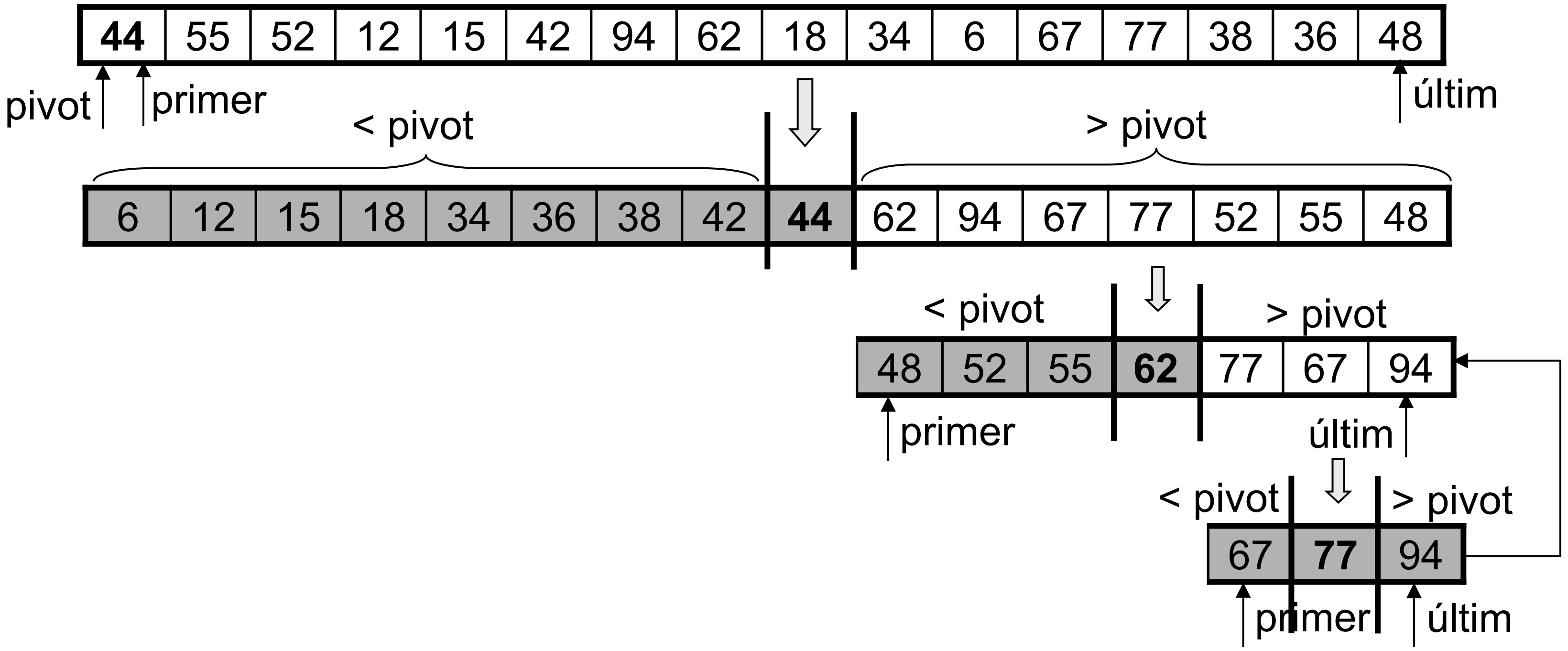
# Algorismes recursius: ordenació ràpida – Quicksort



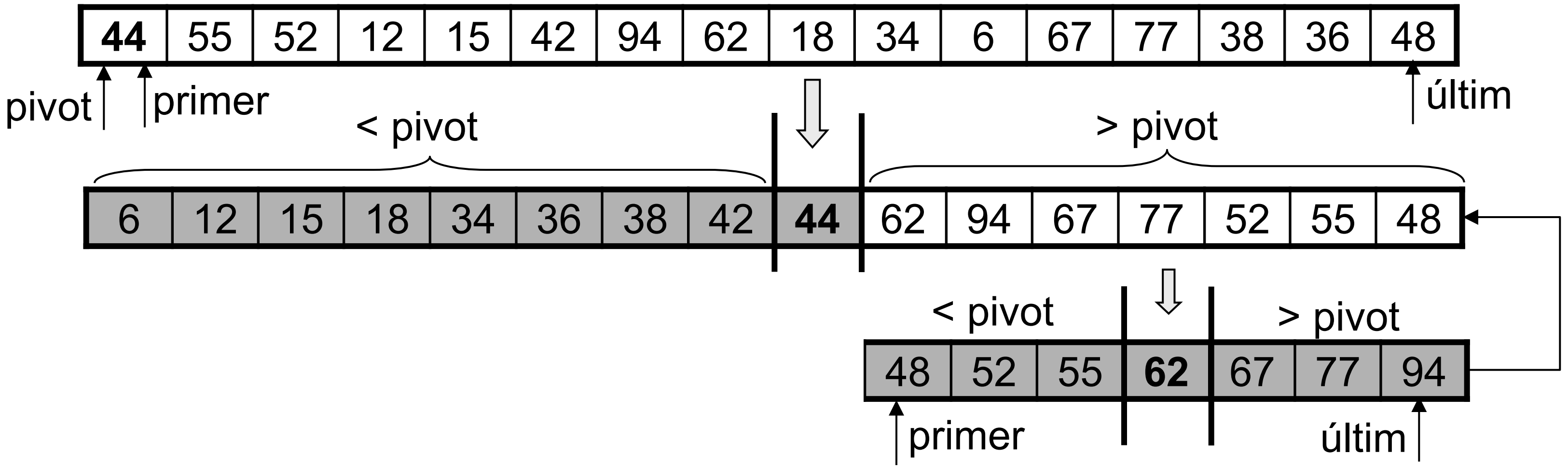
# Algorismes recursius: ordenació ràpida – Quicksort



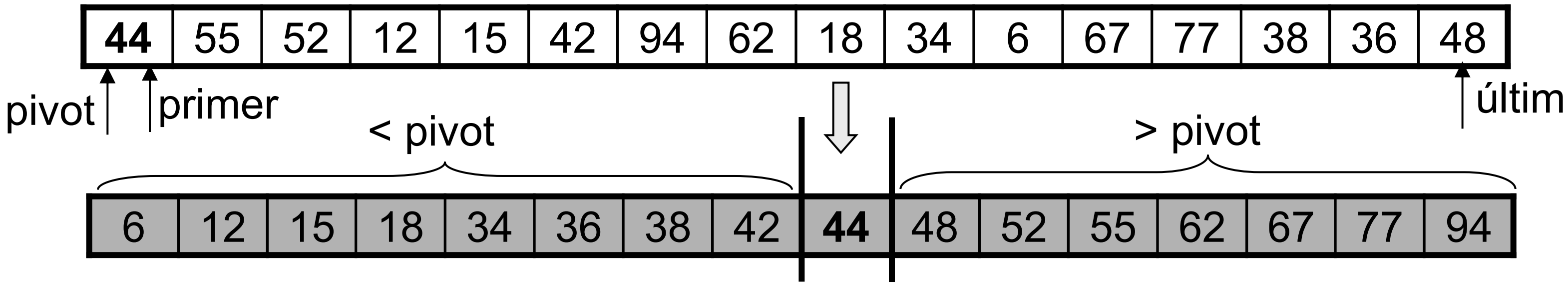
# Algorismes recursius: ordenació ràpida – Quicksort



# Algorismes recursius: ordenació ràpida – Quicksort



# Algorismes recursius: ordenació ràpida – Quicksort



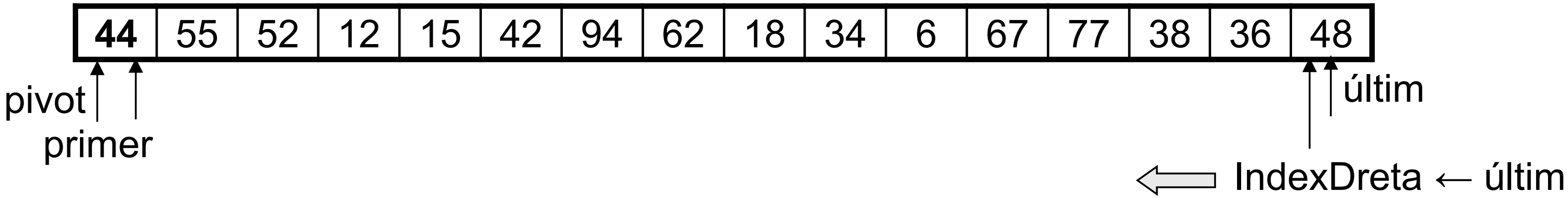

I el vector queda ordenat

# Algorismes recursius: Ordenació d’un vector

Analitzem ara com ordenar un vector que està desordenat...

Passem seguidament a veure com aplicar l’algorisme recursiu **Quicksort** sense necessitat d’espai addicional fent servir la tècnica del **Vector Split**.

# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **44** | 55 | 52 | 12 | 15 | 42 | 94 | 62 | 18 | 34 | 6 | 67 | 77 | 38 | 36 | 48 |

pivot

primer

IndexEsquerra ← primer + 1

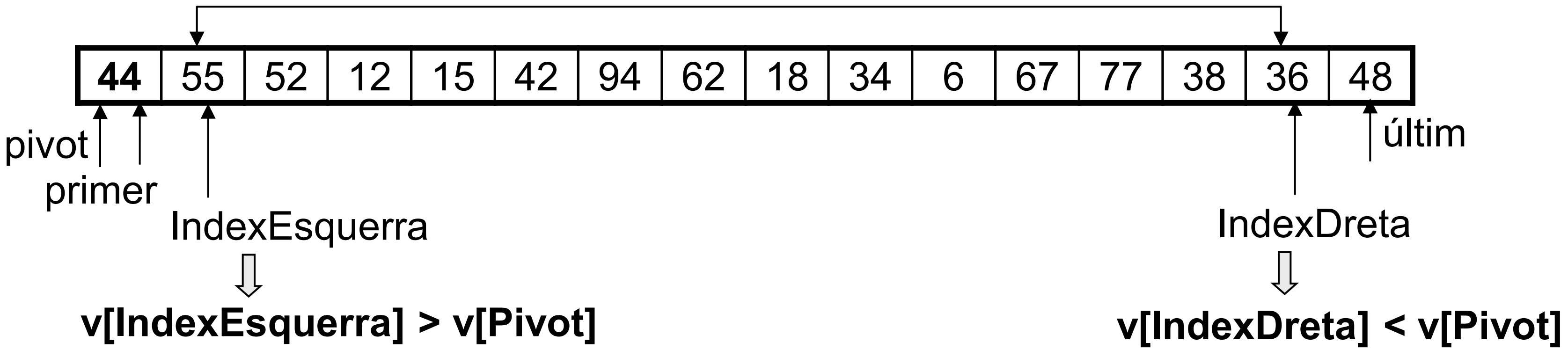
últim

IndexDreta

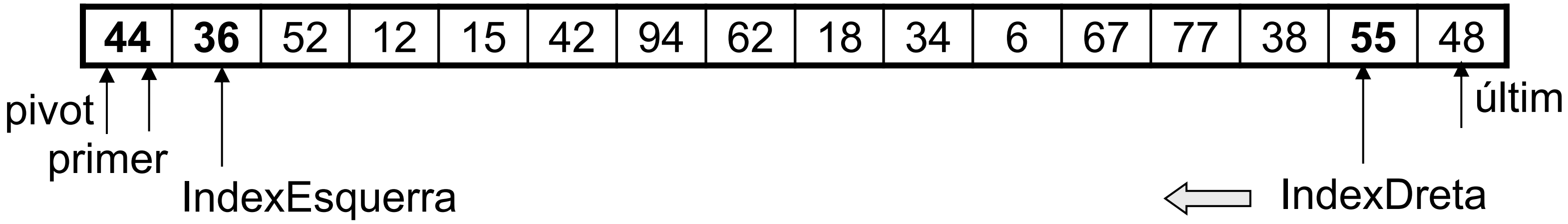
⇩

**v[IndexDreta] < v[Pivot]**

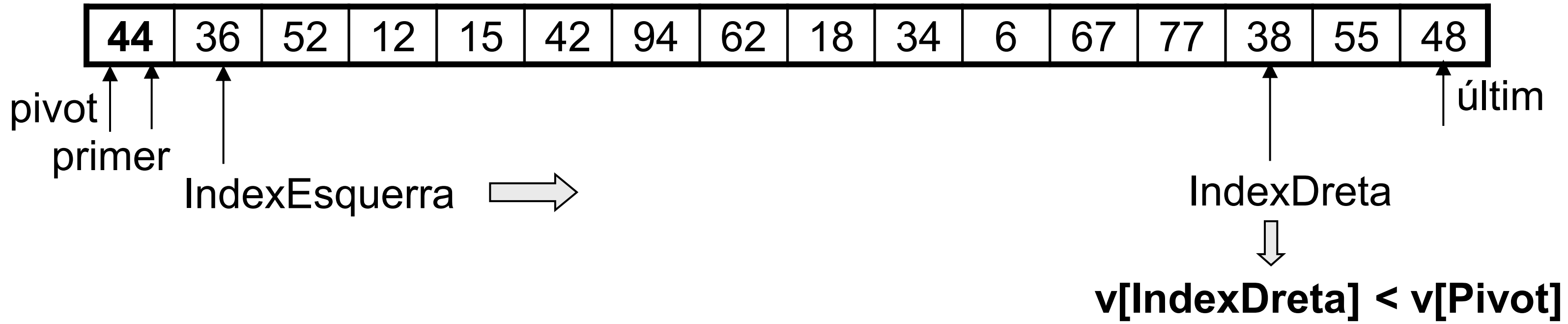
# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



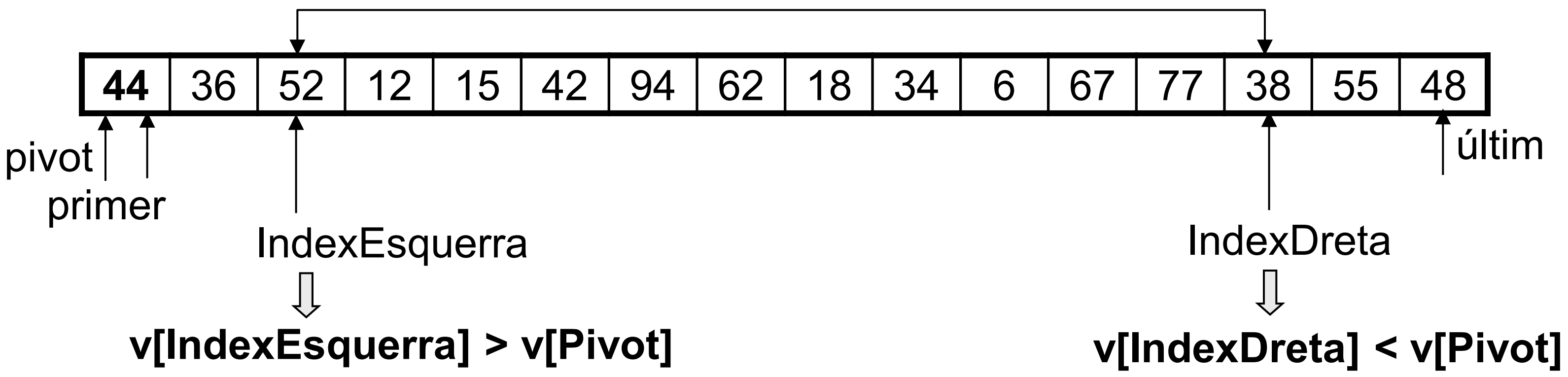
# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



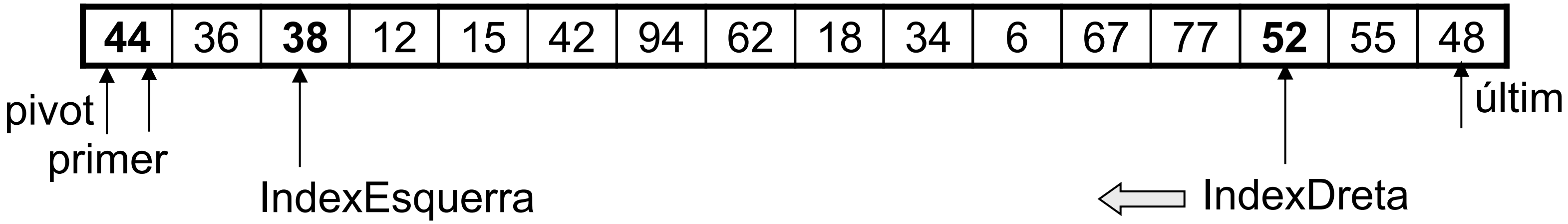
# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **44** | 36 | 38 | 12 | 15 | 42 | 94 | 62 | 18 | 34 | 6 | 67 | 77 | 52 | 55 | 48 |

pivot

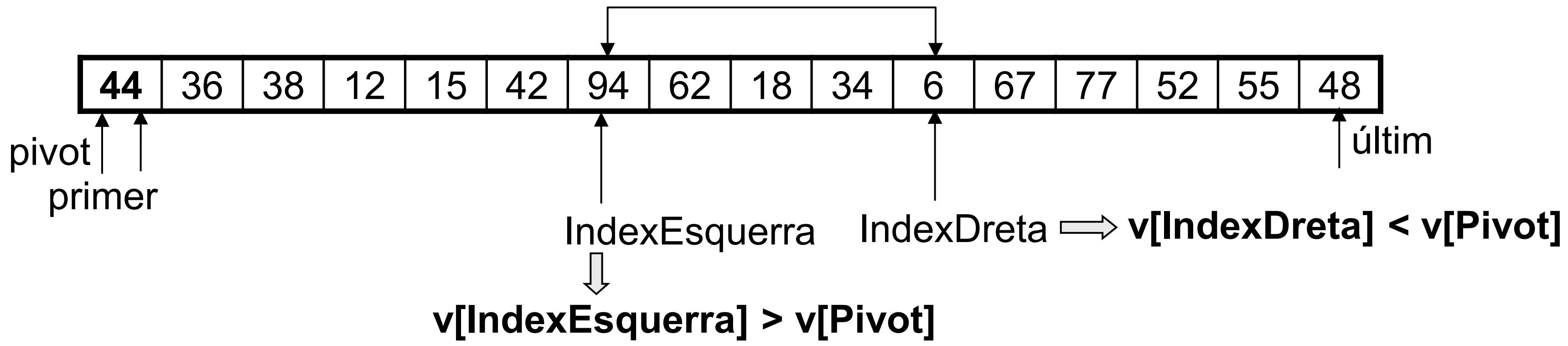
primer

IndexEsquerra ⇨

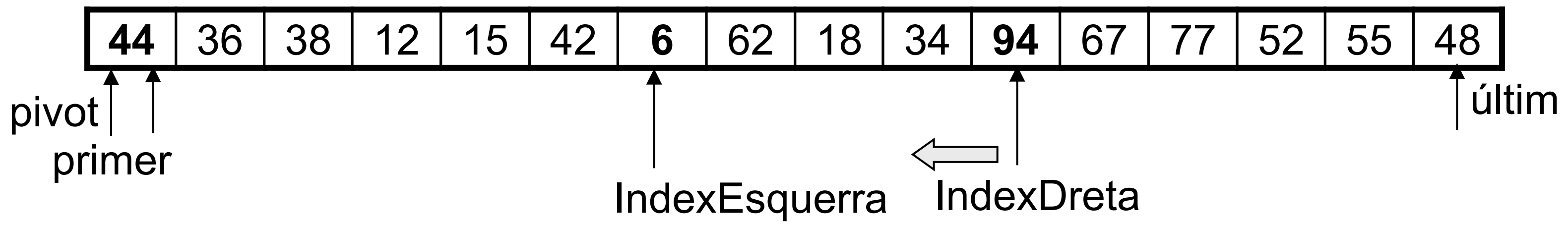
IndexDreta ⇨ **v[IndexDreta] < v[Pivot]**

últim

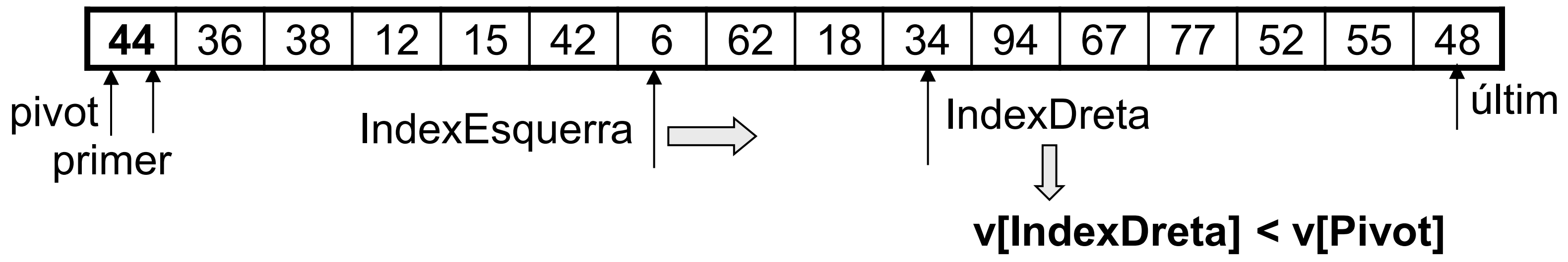
# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



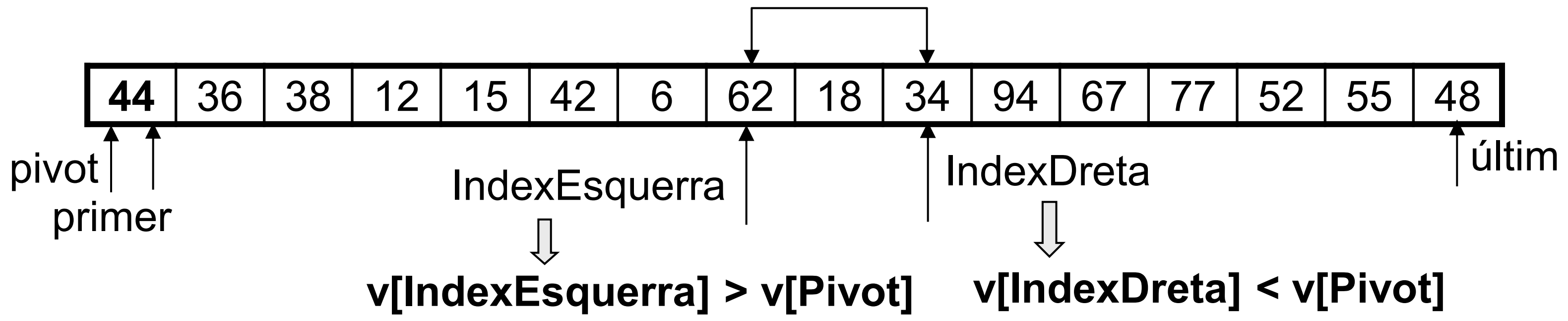
# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



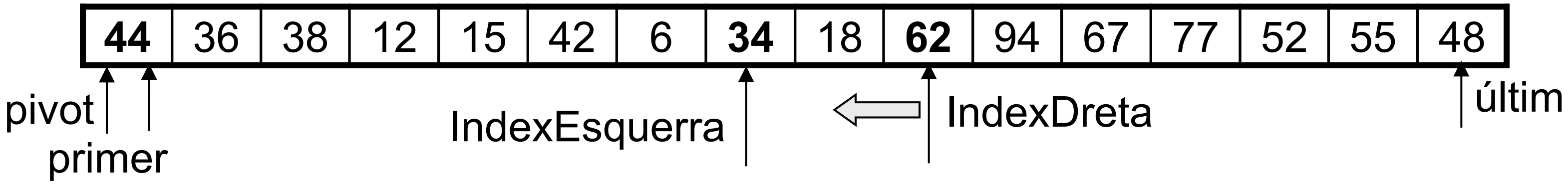
# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **44** | 36 | 38 | 12 | 15 | 42 | 6 | 34 | 18 | 62 | 94 | 67 | 77 | 52 | 55 | 48 |

pivot
primer

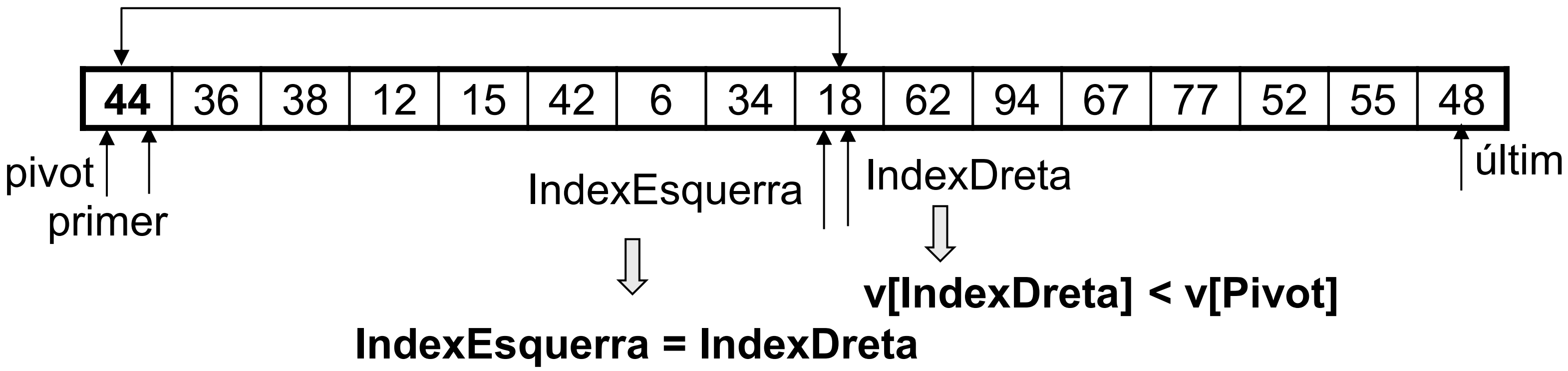
IndexEsquerra

IndexDreta

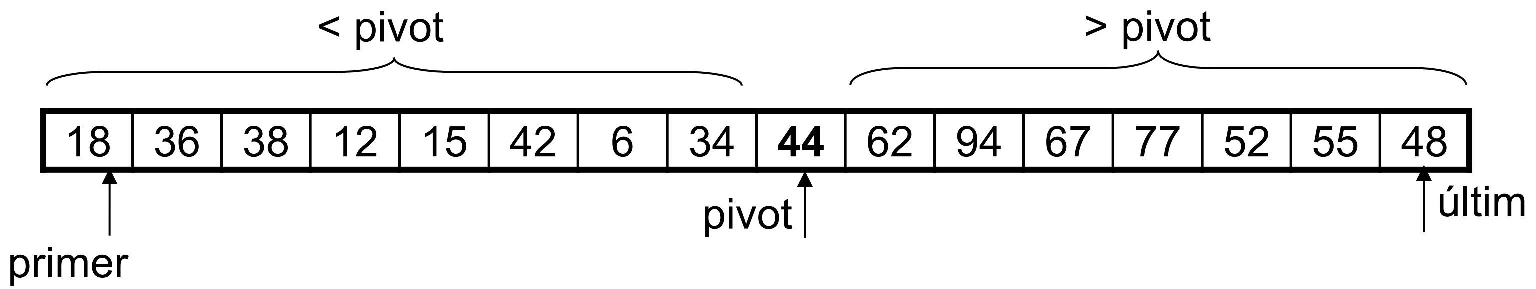
últim

**v[IndexDreta] < v[Pivot]**

# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split



# Algorismes recursius: ordenació ràpida – Quicksort + Vector Split




A partir d’aquest punt només resta ordenar els dos subvectors a l’esquerra i dreta del pivot

# Exercici 4: QuickSort

Implementeu ara l'algorisme d'ordenació QuickSort explicat.

Cal implementar les funcions:

```
QuickSort(v, primer, ultim)
SeleccionarPivot(v, primer, ultim)
DividirVector(v, primer, ultim, PosPivot)
```

# Anàlisi d'algorismes recursius: Quicksort

Per ordenar un vector utilitzant l'algorisme recursiu Quicksort la complexitat és:

- Cas pitjor: **$O(n^2)$** , perquè el pivot pot ser mai divideix el vector per la meitat sinó que està sempre al principi o al final, que a més pot ser qualsevol valor incloent el més gran o petit de la sèrie. Per tant, hi ha casos extrems en els quals no és eficient, i això és un greu problema. Encara que és estadísticament improbable resulta que sí és possible.

- Cas promig: **$O(n*\log(n))$** , sempre que el pivot pugui dividir el vector aproximadament per la meitat en cada iteració. Estadísticament és probable que habitualment sigui així.

# Conclusions: Recursivitat versus Iteració

- Tot algorisme recursiu admet sempre una versió iterativa.
- L'elecció del tipus de solució dependrà tant d'aspectes d'eficiència com també de disseny (simplicitat, llegibilitat, manteniment, etc.).
- De vegades els algorismes recursius poden ser fàcils de programar, però poc eficients perquè calculen moltes vegades el mateix (ex. fibonacci). En aquests casos caldria cercar alternatives iteratives més eficients.
- Quan l'algorisme recursiu i l'algorisme iteratiu són equivalents en número d'operacions, llavors la versió iterativa acostuma a ser més eficient en temps i memòria (ex. factorial).
- En alguns casos, a la versió iterativa caldrà fer servir una *pila* per simular la recursivitat (ex. torres de hanoi, escriure en ordre invers una llista). En aquests casos la versió recursiva acostuma a ser més clara.