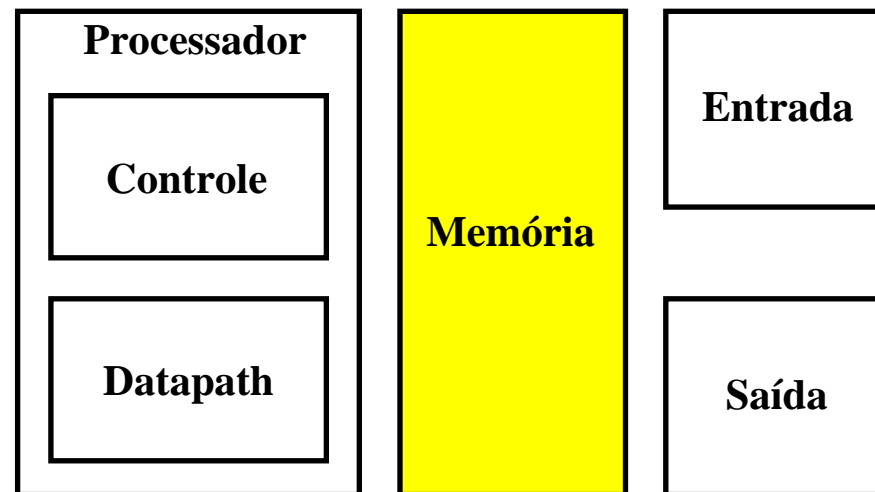
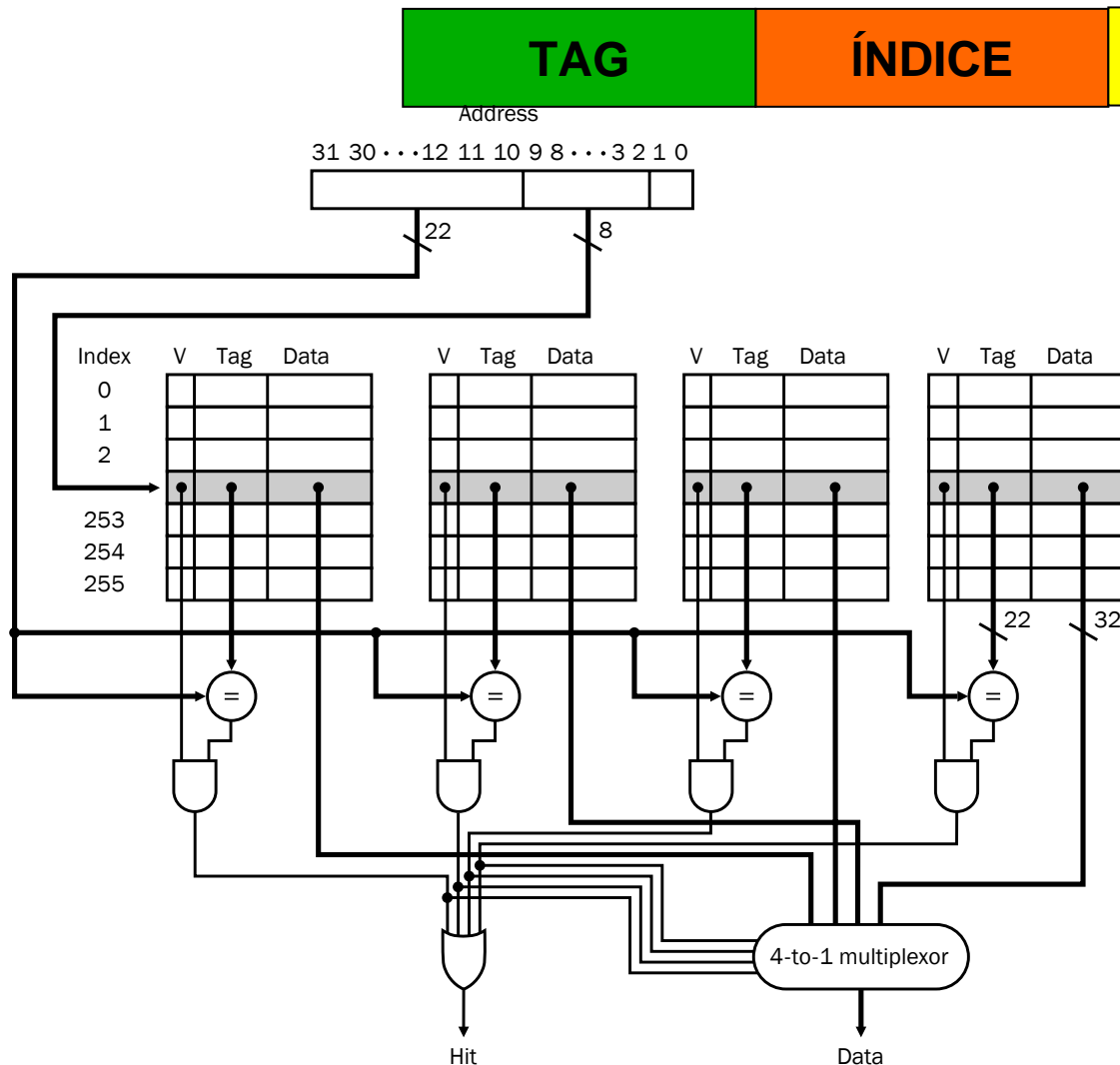


# Cache: Associatividade e múltiplos níveis



# Organização de uma cache n-way



Para tamanho fixo:

**Associatividade** ↑

**blocos p/ conjunto** ↑

**num. de conjuntos** ↓

**Associatividade** ↑

**tamanho do índice** ↓

**tamanho do tag** ↑

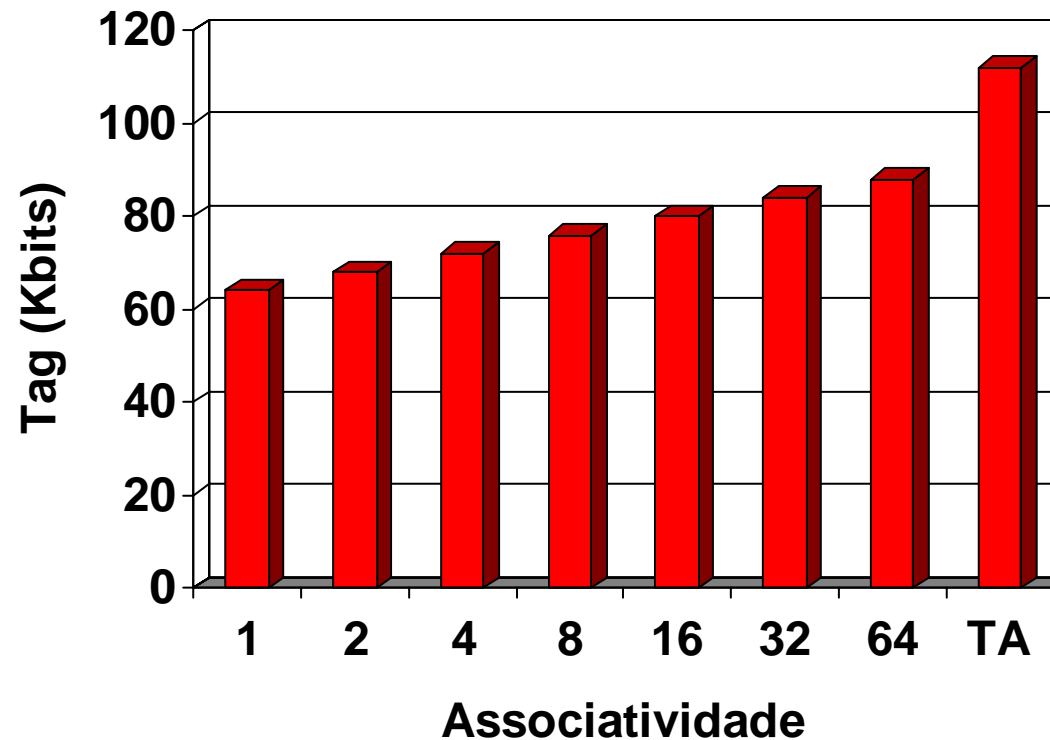
# Tag versus associatividade

- **Hipótese:**
  - Cache com 4K blocos
  - 4 palavras/bloco
  - Endereço de 32 bits
- **Quatro cenários de mapeamento:**
  - Direto, 2-way, 4-way e totalmente associativa
- **Tag + índice + offset**
  - 4 palavras/bloco =  $2^4$  bytes/bloco
  - Restam  $32 - 4 = 28$  bits (tag + índice)

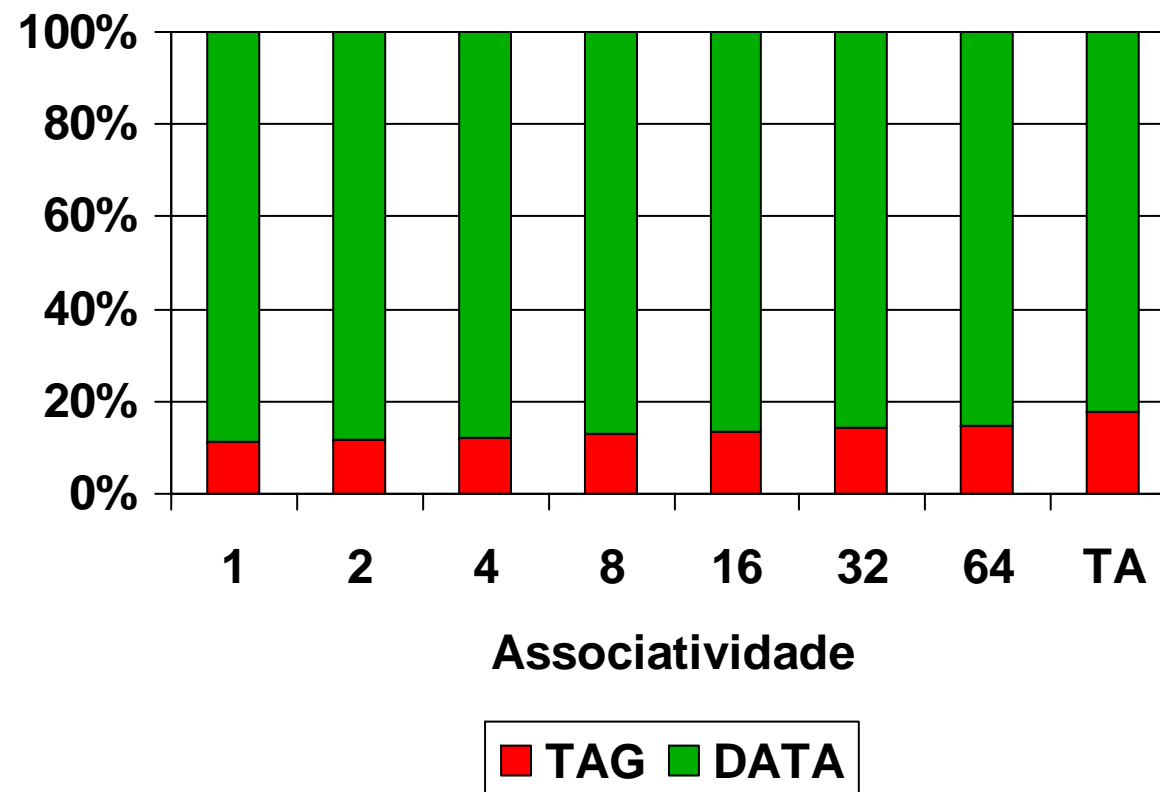
# Tag versus associatividade

- **Número de conjuntos**
  - Direto: 4K conjuntos (índice = 12)
  - 2-way: 2K conjuntos (índice = 11)
  - 4-way: 1K conjuntos (índice = 10)
  - T.A.: 1 conjunto (índice = 0)
- **Número de bits gastos com tag**
  - Direto:  $(28-12) \times 1 \times 4K = 64Kbits$
  - 2-way:  $(28-11) \times 2 \times 2K = 68Kbits$
  - 4-way:  $(28-10) \times 4 \times 1K = 72Kbits$
  - T.A.:  $(28-0) \times 4K \times 1 = 112Kbits$

# Tag versus associatividade



# Tag versus associatividade



# Política de atualização da cache

- **Precisa-se armazenar um dado na cache...**
  - Mas nenhuma posição livre.
  - Qual bloco substituir ?
- **Mapeamento direto:**
  - Bloco ocupando posição mapeada precisa ser substituído
- **Cache n-way:**
  - Critério de escolha mais popular
    - » Bloco não usado há mais tempo é substituído
    - » LRU (“least recently used”)

# Impacto da cache no desempenho

$$\text{ciclos}_{\text{stall}} = \frac{\text{acessos}}{\text{programa}} \times \text{mr} \times \text{penalidade}$$

- **Sejam:**

- **$I$  = número de instruções;**
- **$LS$  = proporção de load + store no “mix”;**
- **$\text{acessos/programa} = I + LS \times I = I(1 + LS)$ ;**
- **$\text{CPI}_{\text{stall}} = \text{ciclos}_{\text{stall}} / I$ ;**
- **$\text{CPI} = \text{CPI}_{\text{ideal}} + \text{CPI}_{\text{stall}} = 1 + \text{CPI}_{\text{stall}}$**

$$\text{CPI} = 1 + (1 + LS) \times \text{mr} \times \text{penalidade}$$



# Impacto da cache no desempenho

- Dado um tamanho fixo de cache...
  - Mais palavras por bloco:
    - » Maior captura da localidade espacial (↓mr)
  - Mais associatividade:
    - » Maior captura da localidade temporal (↓mr)
  - Nenhum impacto na penalidade !?

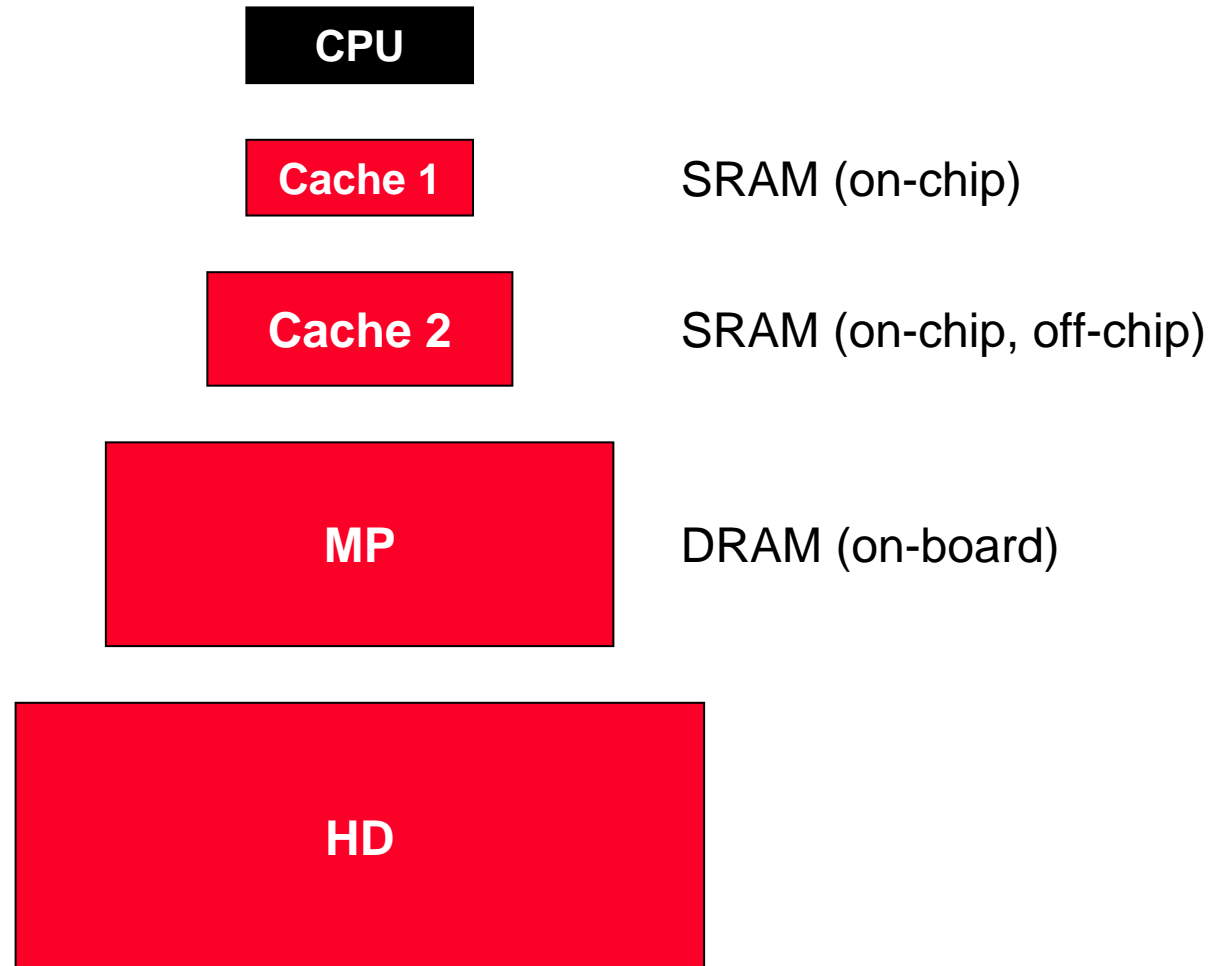
$$\text{CPI} = 1 + (1 + \text{LS}) \times \text{mr} \times \text{penalidade}$$

# Impacto da cache no desempenho

- **Tendências tecnológicas:**
  - $f_r$  cresce rapidamente, mas  $f_{\text{acesso}}$  (DRAM) lentamente.
- **Problema:**
  - Penalidade de falta tende a aumentar.
- **Solução: nível extra de cache...**
  - Comparada a nível única de mesma capacidade...
  - **Objetivo 1: reduzir tempo de acerto e minimizar T**
    - » L1: precisa ser pequena para ser rápida
    - » L1: maior taxa de faltas, mas penalidade compensada por L2
  - **Objetivo 2: reduzir penalidade e taxa global de faltas**
    - » L2: pode ser maior, pois não afeta tempo de acerto nem T
    - » L2: menor taxa de faltas local, mas menor penalidade

$$\text{CPI} = 1 + (1 + \text{LS}) \times \text{mr} \times \text{penalidade}$$

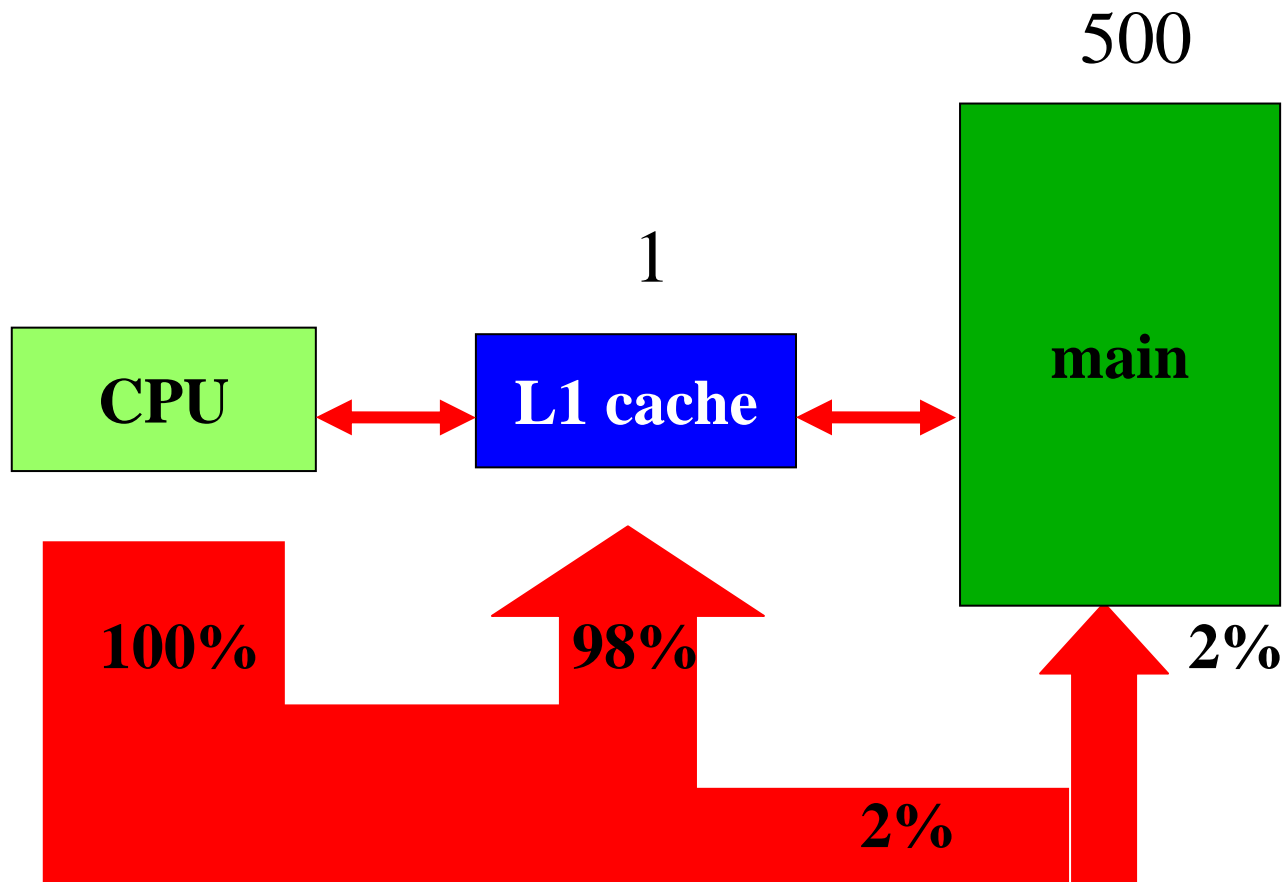
# Redução da penalidade



# Exemplo

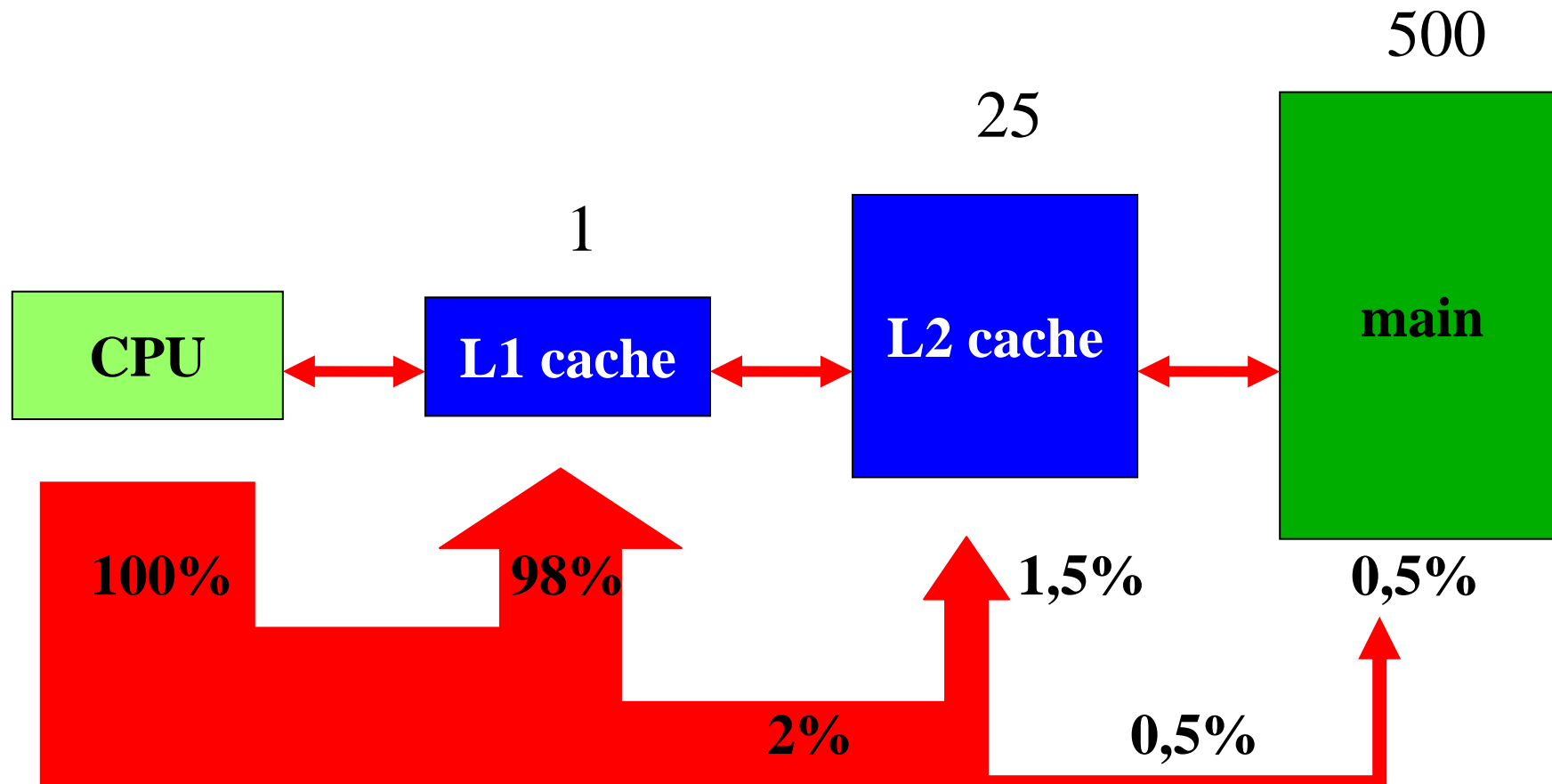
- **Impacto da cache secundária**
  - $f = 5\text{GHz}$  ( $T = 0,2\text{ns}$ )
  - Primária:  $0,2\text{ns}$ , taxa de faltas = 2%
  - Secundária:  $5\text{ns}$ , taxa global de faltas = 0,5%
  - Memória principal:  $100\text{ns}$
- **Hipóteses:**
  - Só acesso a instruções;
  - $\text{CPI}_{\text{ideal}} = 1$ .
- **Comparação:**
  - Qual o CPI só com a cache primária ?
  - Qual o CPI para dois níveis de cache ?

# Uma única cache



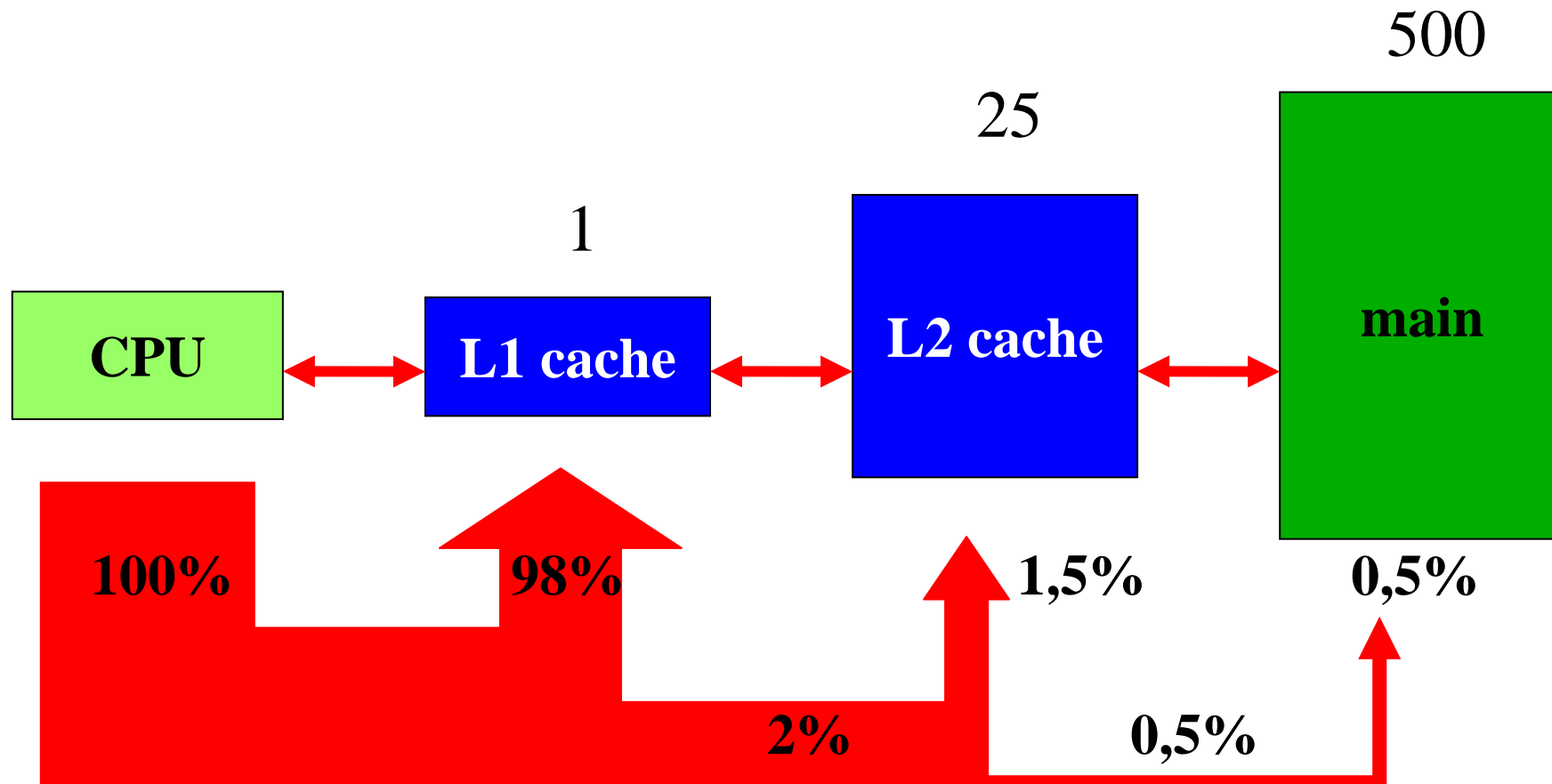
$$\text{CPI} = 1 + 0,02 \times 500 = 11$$

## Dois níveis de cache



$$\text{CPI} = 1 + 0,02 \times 25 + 0,005 \times 500 = 4,0$$

## Dois níveis de cache



$$\text{CPI} = 1 + 0,015 \times 25 + 0,005 \times 525 = 4,0$$

# Impacto da cache no desempenho

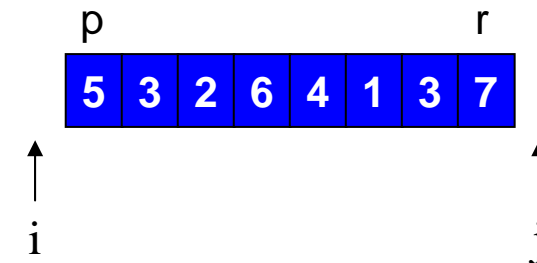
- **Dois programas de ordenação**
  - Diferentes números de itens a ordenar (4 a 4K)
- **Algoritmos com complexidades distintas**
  - QuickSort:  $O(n \log n)$  [média]
  - RadixSort:  $O(n)$  [média]
- **Complexidade:**
  - Comportamento assintótico
    - » Número de operações como função da entrada ( $n$ )
- **Desempenho**
  - Número de instruções e número de ciclos



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
  then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

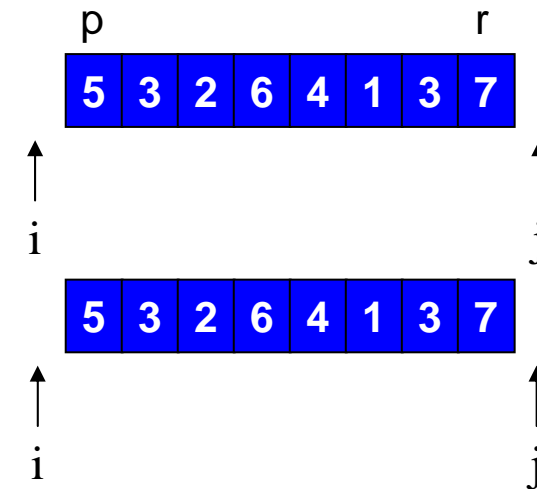
```
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
  do repeat j ← j - 1
    until A[j] ≤ x
  repeat i ← i + 1
    until A[i] ≥ x
  if i < j
    then exchange A[i] ↔ A[j]
  else return j
```



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
  then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

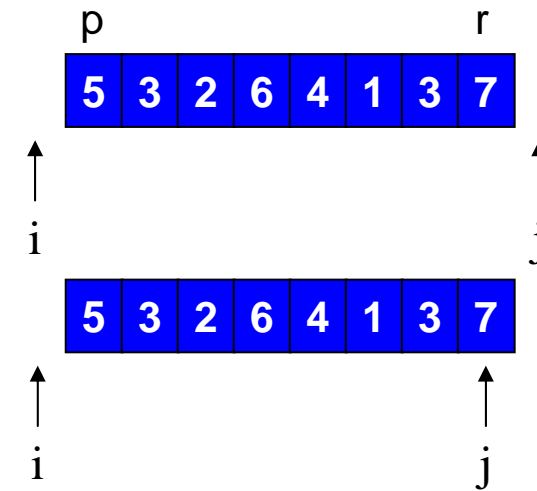
```
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
  do    repeat    j ← j - 1
            until A[j] ≤ x
        repeat    i ← i + 1
            until A[i] ≥ x
  if i < j
    then exchange A[i] ↔ A[j]
    else return j
```



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
  then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

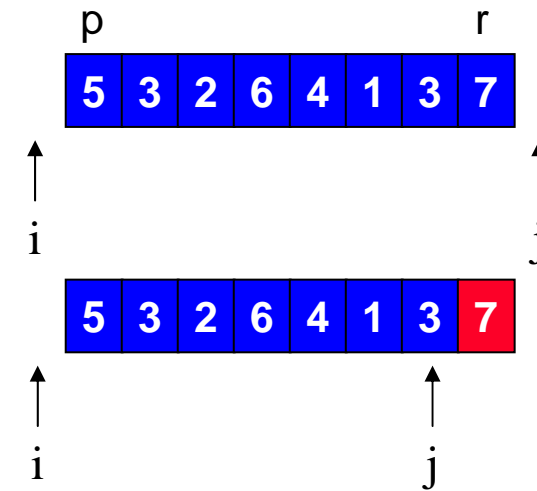
```
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
  do   repeat   j ← j - 1
        until A[j] ≤ x
        repeat   i ← i + 1
        until A[i] ≥ x
  if i < j
    then exchange A[i] ↔ A[j]
    else return j
```



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
  then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

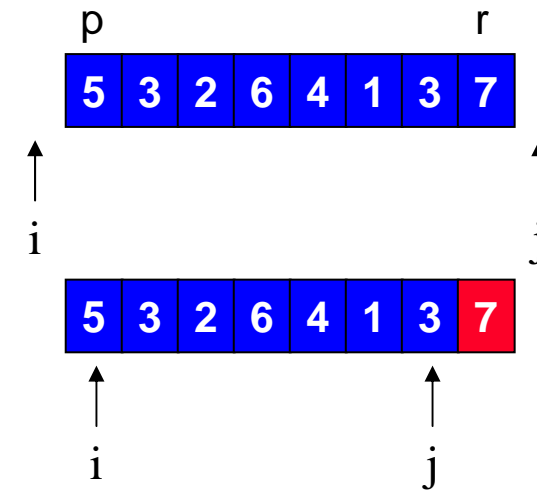
```
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
  do   repeat   j ← j - 1
        until A[j] ≤ x
        repeat   i ← i + 1
        until A[i] ≥ x
  if i < j
    then exchange A[i] ↔ A[j]
    else return j
```



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
  then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

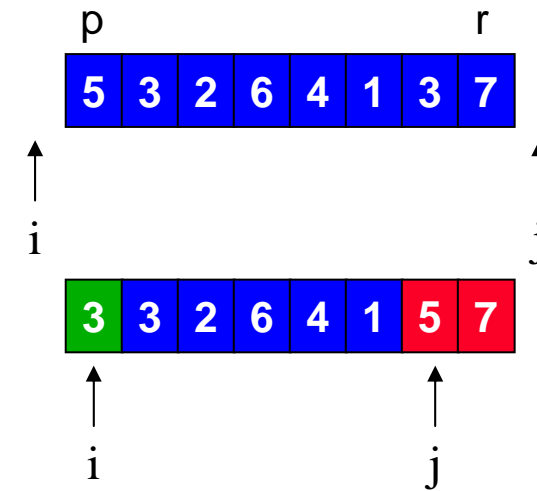
```
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
  do repeat j ← j - 1
    until A[j] ≤ x
  repeat i ← i + 1
    until A[i] ≥ x
  if i < j
    then exchange A[i] ↔ A[j]
  else return j
```



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
    then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

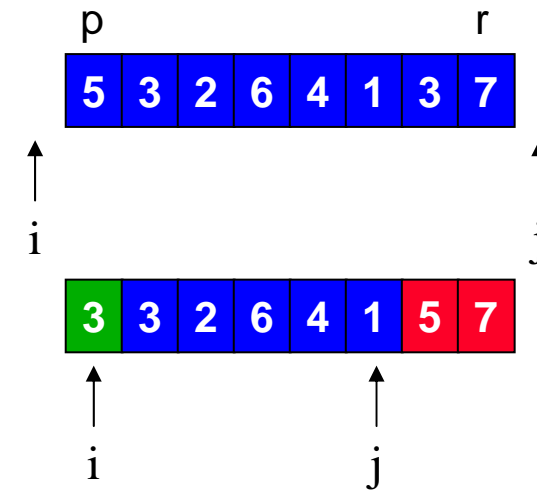
```
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
    do    repeat    j ← j - 1
            until A[j] ≤ x
        repeat    i ← i + 1
            until A[i] ≥ x
    if i < j
        then exchange A[i] ↔ A[j]
    else return j
```



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
  then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)

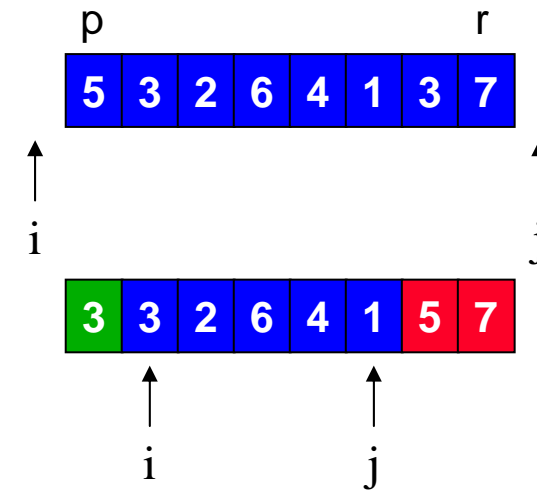
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
  do    repeat    j ← j - 1
          until A[j] ≤ x
        repeat    i ← i + 1
          until A[i] ≥ x
        if i < j
          then exchange A[i] ↔ A[j]
          else return j
```



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
  then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

```
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
  do   repeat   j ← j - 1
        until A[j] ≤ x
        repeat   i ← i + 1
        until A[i] ≥ x
  if i < j
    then exchange A[i] ↔ A[j]
  else return j
```

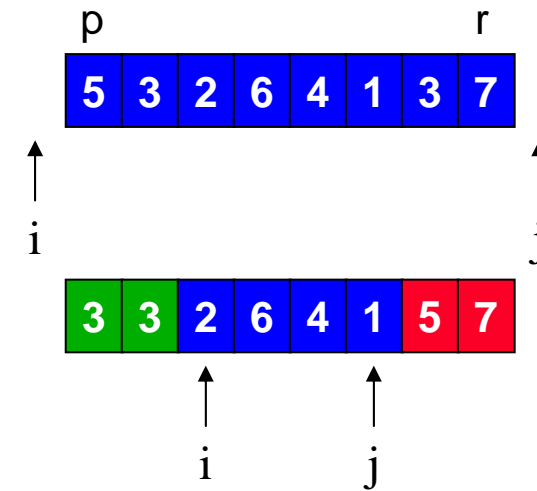




# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
  then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

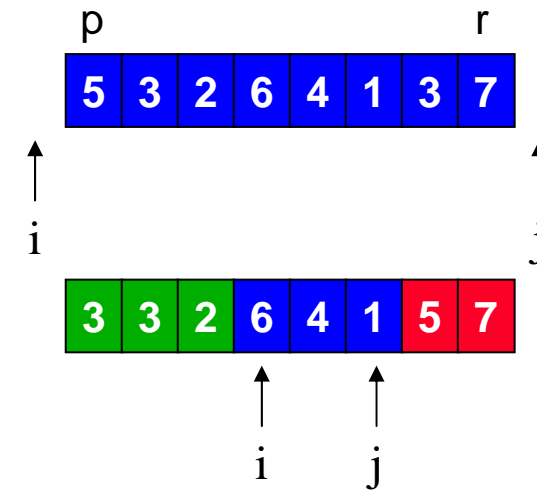
```
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
  do    repeat    j ← j - 1
            until A[j] ≤ x
        repeat    i ← i + 1
            until A[i] ≥ x
  if i < j
    then exchange A[i] ↔ A[j]
  else return j
```



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
  then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

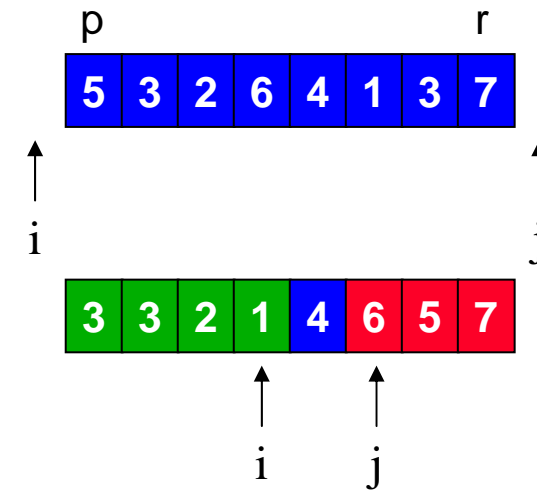
```
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
  do repeat j ← j - 1
    until A[j] ≤ x
  repeat i ← i + 1
    until A[i] ≥ x
  if i < j
    then exchange A[i] ↔ A[j]
  else return j
```



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
  then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)
```

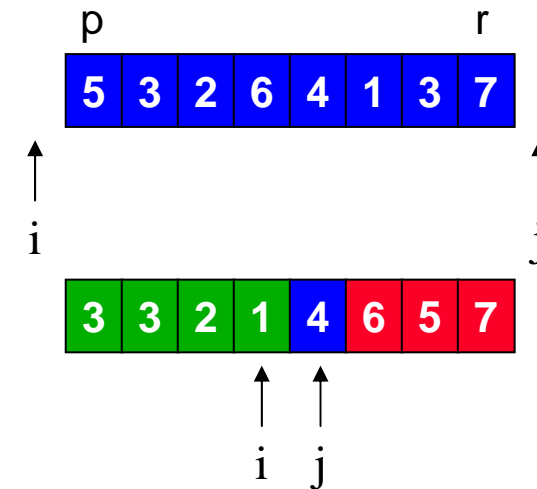
```
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
  do repeat j ← j - 1
    until A[j] ≤ x
  repeat i ← i + 1
    until A[i] ≥ x
  if i < j
    then exchange A[i] ↔ A[j]
  else return j
```



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
    then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)

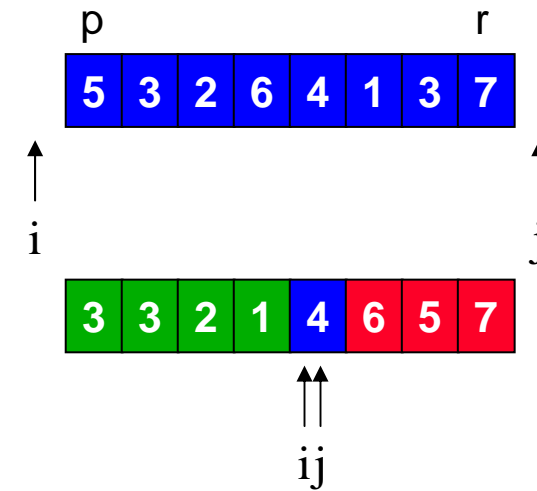
PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
    do    repeat    j ← j - 1
            until A[j] ≤ x
        repeat    i ← i + 1
            until A[i] ≥ x
    if i < j
        then exchange A[i] ↔ A[j]
    else return j
```



# Revisitando QUICKSORT

```
QUICKSORT(A,p,r)
if p < r
    then q ← PARTITION(A,p,r)
        QUICKSORT(A,p,q)
        QUICKSORT(A,q+1,r)

PARTITION(A,p,r)
x ← A[p]
i ← p - 1
j ← r + 1
while TRUE
    do    repeat    j ← j - 1
            until A[j] ≤ x
        repeat    i ← i + 1
            until A[i] ≥ x
    if i < j
        then exchange A[i] ↔ A[j]
    else return j
```

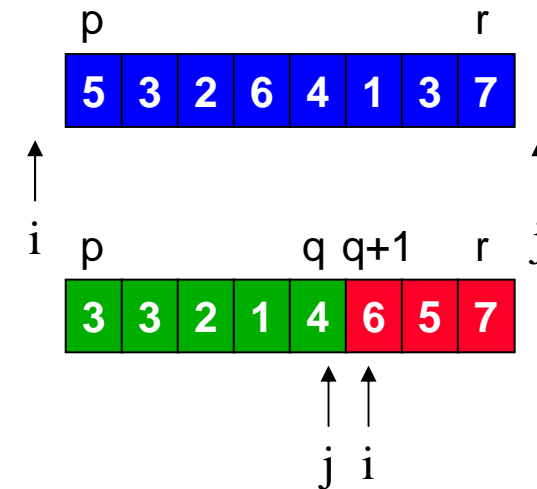


# Revisitando QUICKSORT

```

QUICKSORT(A,p,r)
  if p < r
    then q ← PARTITION(A,p,r)
         QUICKSORT(A,p,q)
         QUICKSORT(A,q+1,r)

PARTITION(A,p,r)
  x ← A[p]
  i ← p - 1
  j ← r + 1
  while TRUE
    do   repeat  j ← j - 1
          until A[j] ≤ x
         repeat  i ← i + 1
          until A[i] ≥ x
    if i < j
      then exchange A[i] ↔ A[j]
      else return j
    
```



## Localidade temporal (instruções)

Grande: recursividade e laços

## Localidade espacial (instruções)

Pouca: BBs pequenos

## Localidade espacial (dados)

Grande:  $i$  e  $j$  percorrem arranjo sequencialmente

## Localidade temporal (dados)

Grande: partições são revisitadas

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

    do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

    do C[A[j]]  $\leftarrow$  C[A[j]] + 1

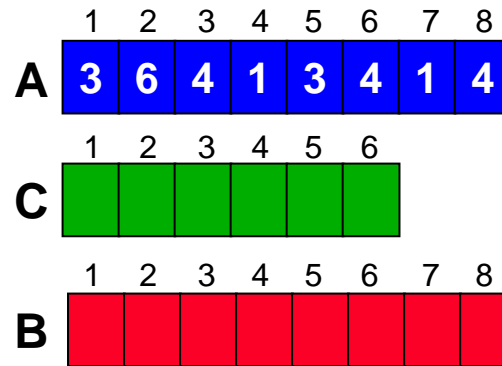
**for** i  $\leftarrow$  2 **to** k

    do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

    do B[C[A[j]]]  $\leftarrow$  A[j]

        C[A[j]]  $\leftarrow$  C[A[j]] - 1



**Estado inicial**

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

    do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

    do C[A[j]]  $\leftarrow$  C[A[j]] + 1

**for** i  $\leftarrow$  2 **to** k

    do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

    do B[C[A[j]]]  $\leftarrow$  A[j]

        C[A[j]]  $\leftarrow$  C[A[j]] - 1

	1	2	3	4	5	6	7	8
<b>A</b>	3	6	4	1	3	4	1	4
	1	2	3	4	5	6		
<b>C</b>	0	0	0	0	0	0		
	1	2	3	4	5	6	7	8
<b>B</b>								

**C inicializado**



# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for**  $i \leftarrow 1$  **to**  $k$

    do  $C[i] \leftarrow 0$

**for**  $j \leftarrow 1$  **to**  $\text{length}[A]$

    do  $C[A[j]] \leftarrow C[A[j]] + 1$

**for**  $i \leftarrow 2$  **to**  $k$

    do  $C[i] \leftarrow C[i] + C[i-1]$

**for**  $j \leftarrow \text{length}[A]$  **downto**  $1$

    do  $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

	1	2	3	4	5	6	7	8
<b>A</b>	3	6	4	1	3	4	1	4
	1	2	3	4	5	6		
<b>C</b>	0	0	1	0	0	0		
	1	2	3	4	5	6	7	8
<b>B</b>								

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

    do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

    do C[A[j]]  $\leftarrow$  C[A[j]] + 1

**for** i  $\leftarrow$  2 **to** k

    do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

    do B[C[A[j]]]  $\leftarrow$  A[j]

        C[A[j]]  $\leftarrow$  C[A[j]] - 1

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4
	1	2	3	4	5	6		
C	0	0	1	0	0	1		
	1	2	3	4	5	6	7	8
B								

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for**  $i \leftarrow 1$  **to**  $k$

    do  $C[i] \leftarrow 0$

**for**  $j \leftarrow 1$  **to**  $\text{length}[A]$

    do  $C[A[j]] \leftarrow C[A[j]] + 1$

**for**  $i \leftarrow 2$  **to**  $k$

    do  $C[i] \leftarrow C[i] + C[i-1]$

**for**  $j \leftarrow \text{length}[A]$  **downto**  $1$

    do  $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4
	1	2	3	4	5	6		
C	0	0	1	1	0	1		
	1	2	3	4	5	6	7	8
B								

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for**  $i \leftarrow 1$  **to**  $k$

    do  $C[i] \leftarrow 0$

**for**  $j \leftarrow 1$  **to**  $\text{length}[A]$

    do  $C[A[j]] \leftarrow C[A[j]] + 1$

**for**  $i \leftarrow 2$  **to**  $k$

    do  $C[i] \leftarrow C[i] + C[i-1]$

**for**  $j \leftarrow \text{length}[A]$  **downto**  $1$

    do  $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4
	1	2	3	4	5	6		
C	1	0	1	1	0	1		
	1	2	3	4	5	6	7	8
B								

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for**  $i \leftarrow 1$  **to**  $k$

    do  $C[i] \leftarrow 0$

**for**  $j \leftarrow 1$  **to**  $\text{length}[A]$

    do  $C[A[j]] \leftarrow C[A[j]] + 1$

**for**  $i \leftarrow 2$  **to**  $k$

    do  $C[i] \leftarrow C[i] + C[i-1]$

**for**  $j \leftarrow \text{length}[A]$  **downto**  $1$

    do  $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

	1	2	3	4	5	6	7	8
<b>A</b>	3	6	4	1	3	4	1	4
	1	2	3	4	5	6		
<b>C</b>	1	0	2	1	0	1		
	1	2	3	4	5	6	7	8
<b>B</b>								

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for**  $i \leftarrow 1$  **to**  $k$

    do  $C[i] \leftarrow 0$

**for**  $j \leftarrow 1$  **to**  $\text{length}[A]$

    do  $C[A[j]] \leftarrow C[A[j]] + 1$

**for**  $i \leftarrow 2$  **to**  $k$

    do  $C[i] \leftarrow C[i] + C[i-1]$

**for**  $j \leftarrow \text{length}[A]$  **downto**  $1$

    do  $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

	1	2	3	4	5	6	7	8
<b>A</b>	3	6	4	1	3	4	1	4
	1	2	3	4	5	6		
<b>C</b>	1	0	2	2	0	1		
	1	2	3	4	5	6	7	8
<b>B</b>								

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for**  $i \leftarrow 1$  **to**  $k$

    do  $C[i] \leftarrow 0$

**for**  $j \leftarrow 1$  **to**  $\text{length}[A]$

    do  $C[A[j]] \leftarrow C[A[j]] + 1$

**for**  $i \leftarrow 2$  **to**  $k$

    do  $C[i] \leftarrow C[i] + C[i-1]$

**for**  $j \leftarrow \text{length}[A]$  **downto**  $1$

    do  $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

	1	2	3	4	5	6	7	8
<b>A</b>	3	6	4	1	3	4	1	4
	1	2	3	4	5	6		
<b>C</b>	2	0	2	2	0	1		
	1	2	3	4	5	6	7	8
<b>B</b>								

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

    do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

    do C[A[j]]  $\leftarrow$  C[A[j]] + 1

**for** i  $\leftarrow$  2 **to** k

    do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

    do B[C[A[j]]]  $\leftarrow$  A[j]

        C[A[j]]  $\leftarrow$  C[A[j]] - 1

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4
	1	2	3	4	5	6		
C	2	0	2	3	0	1		
	1	2	3	4	5	6	7	8
B								



# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

    do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

    do C[A[j]]  $\leftarrow$  C[A[j]] + 1

**for** i  $\leftarrow$  2 **to** k

    do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

    do B[C[A[j]]]  $\leftarrow$  A[j]

        C[A[j]]  $\leftarrow$  C[A[j]] - 1

	1	2	3	4	5	6	7	8
<b>A</b>	3	6	4	1	3	4	1	4
	1	2	3	4	5	6		
<b>C</b>	2	0	2	3	0	1		
	1	2	3	4	5	6	7	8
<b>B</b>								

**C[i]** contém o número de elementos iguais a i

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

for i ← 1 to k

do C[i] ← 0

for j ← 1 to length[A]

do C[A[j]] ← C[A[j]] + 1

for i ← 2 to k

do C[i] ← C[i] + C[i-1]

for j ← length[A] downto 1

do B[C[A[j]]] ← A[j]

C[A[j]] ← C[A[j]] - 1

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4
	1	2	3	4	5	6		
C	2	2	4	7	7	8		
	1	2	3	4	5	6	7	8
B								

**C[i]** contém o número de elementos iguais ou menores a i

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

do C[A[j]]  $\leftarrow$  C[A[j]] + 1

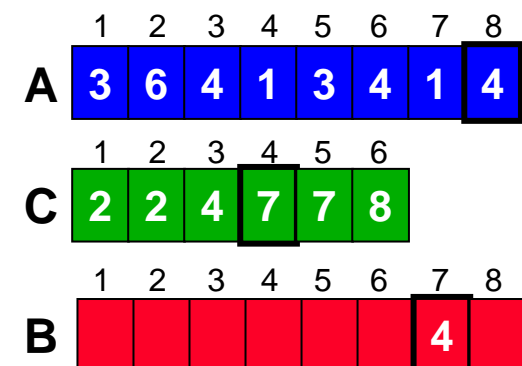
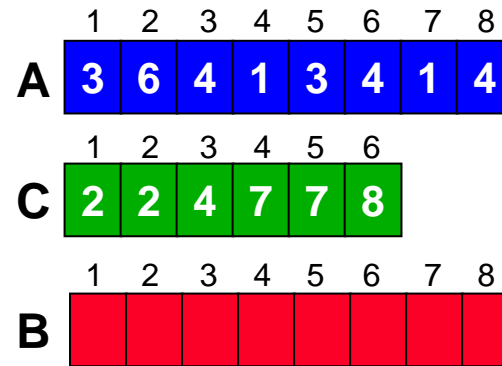
**for** i  $\leftarrow$  2 **to** k

do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

do B[C[A[j]]]  $\leftarrow$  A[j]

C[A[j]]  $\leftarrow$  C[A[j]] - 1



# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

for i ← 1 to k

do C[i] ← 0

for j ← 1 to length[A]

do C[A[j]] ← C[A[j]] + 1

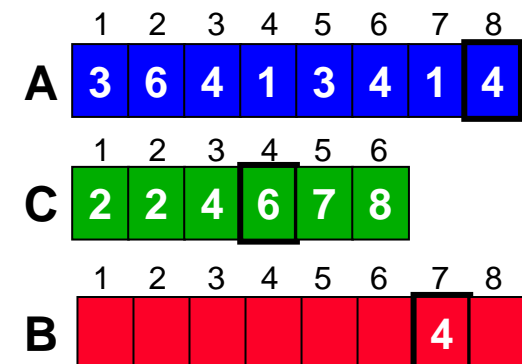
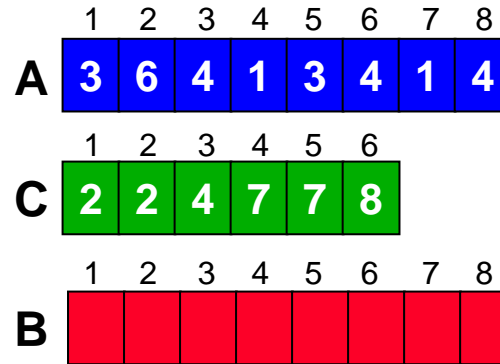
for i ← 2 to k

do C[i] ← C[i] + C[i-1]

for j ← length[A] downto 1

do B[C[A[j]]] ← A[j]

C[A[j]] ← C[A[j]] - 1



Ao final da primeira iteração do último laço

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

do C[A[j]]  $\leftarrow$  C[A[j]] + 1

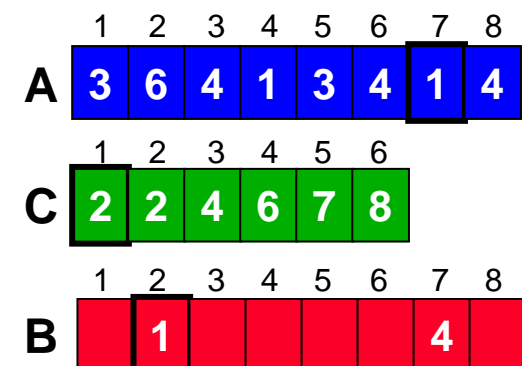
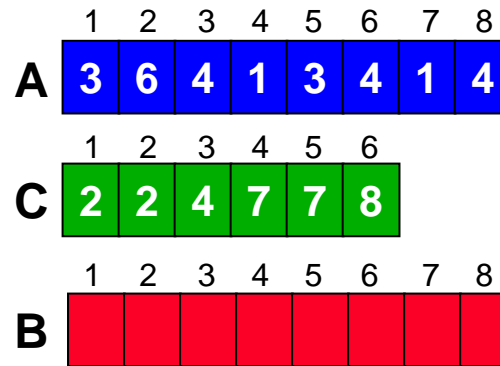
**for** i  $\leftarrow$  2 **to** k

do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

do B[C[A[j]]]  $\leftarrow$  A[j]

C[A[j]]  $\leftarrow$  C[A[j]] - 1



# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

    do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

    do C[A[j]]  $\leftarrow$  C[A[j]] + 1

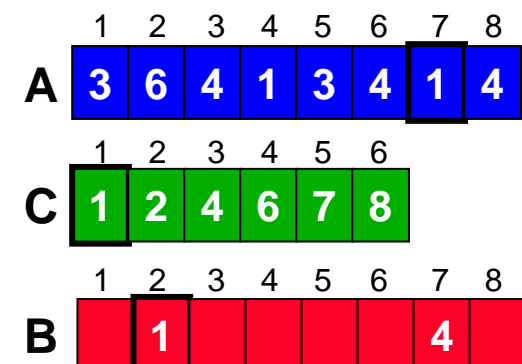
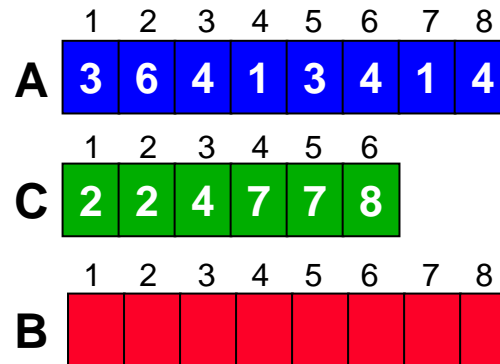
**for** i  $\leftarrow$  2 **to** k

    do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

    do B[C[A[j]]]  $\leftarrow$  A[j]

        C[A[j]]  $\leftarrow$  C[A[j]] - 1



**Ao final da segunda iteração do último laço**

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

do C[A[j]]  $\leftarrow$  C[A[j]] + 1

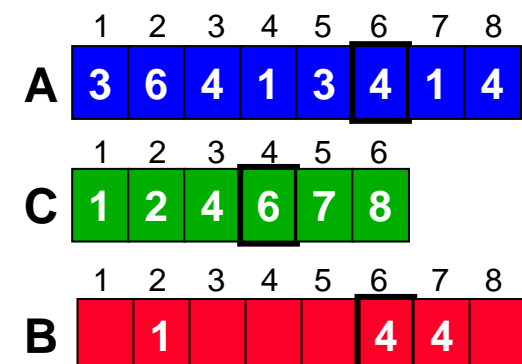
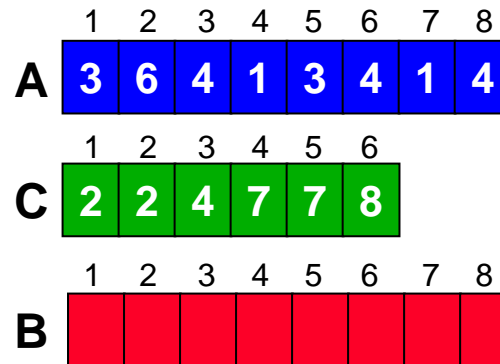
**for** i  $\leftarrow$  2 **to** k

do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

do B[C[A[j]]]  $\leftarrow$  A[j]

C[A[j]]  $\leftarrow$  C[A[j]] - 1



# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

do C[A[j]]  $\leftarrow$  C[A[j]] + 1

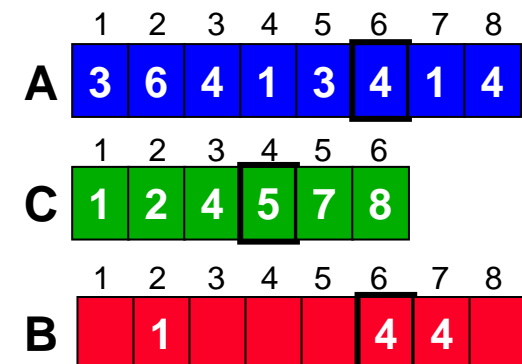
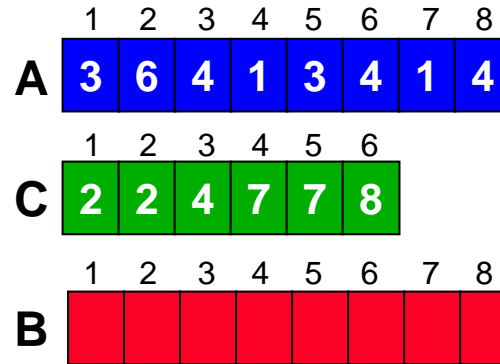
**for** i  $\leftarrow$  2 **to** k

do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

do B[C[A[j]]]  $\leftarrow$  A[j]

C[A[j]]  $\leftarrow$  C[A[j]] - 1



**Ao final da terceira iteração do último laço**



# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

do C[A[j]]  $\leftarrow$  C[A[j]] + 1

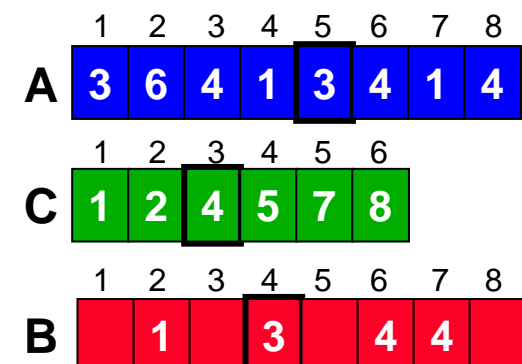
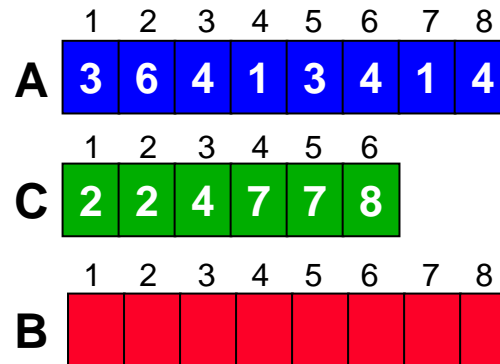
**for** i  $\leftarrow$  2 **to** k

do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

do B[C[A[j]]]  $\leftarrow$  A[j]

C[A[j]]  $\leftarrow$  C[A[j]] - 1



# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

    do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

    do C[A[j]]  $\leftarrow$  C[A[j]] + 1

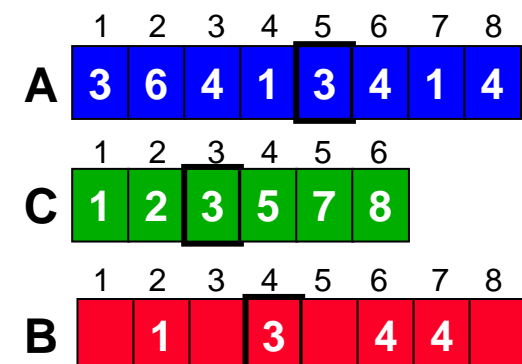
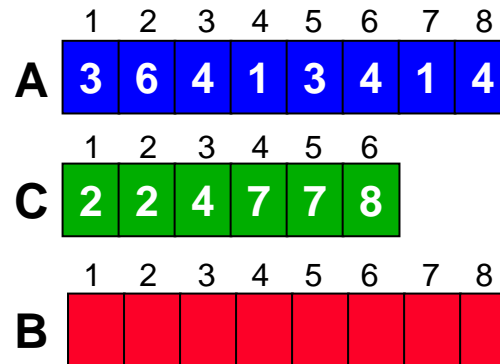
**for** i  $\leftarrow$  2 **to** k

    do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

    do B[C[A[j]]]  $\leftarrow$  A[j]

        C[A[j]]  $\leftarrow$  C[A[j]] - 1



**Ao final da quarta iteração do último laço**

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

do C[A[j]]  $\leftarrow$  C[A[j]] + 1

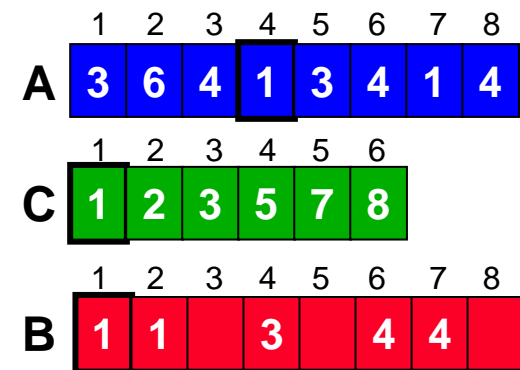
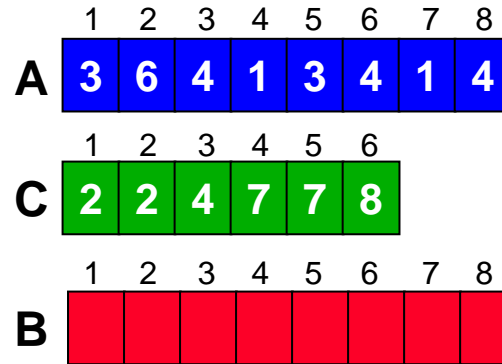
**for** i  $\leftarrow$  2 **to** k

do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

do B[C[A[j]]]  $\leftarrow$  A[j]

C[A[j]]  $\leftarrow$  C[A[j]] - 1



# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

    do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

    do C[A[j]]  $\leftarrow$  C[A[j]] + 1

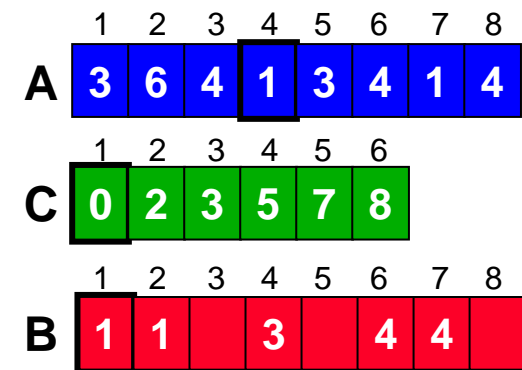
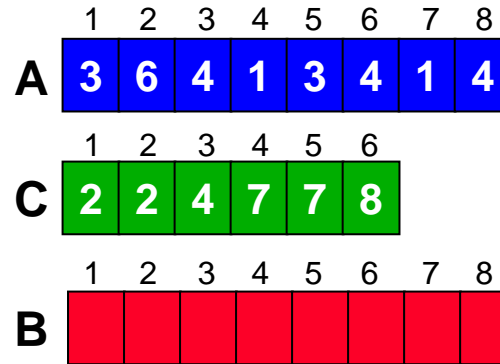
**for** i  $\leftarrow$  2 **to** k

    do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

    do B[C[A[j]]]  $\leftarrow$  A[j]

        C[A[j]]  $\leftarrow$  C[A[j]] - 1



**Ao final da quinta iteração do último laço**

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

do C[A[j]]  $\leftarrow$  C[A[j]] + 1

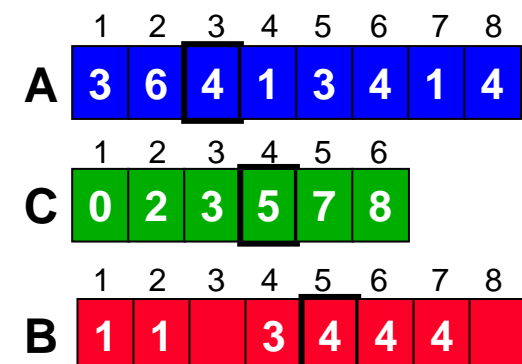
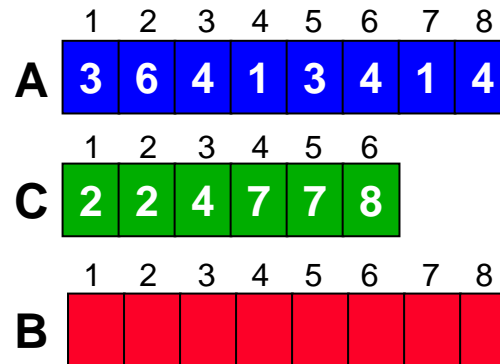
**for** i  $\leftarrow$  2 **to** k

do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

do B[C[A[j]]]  $\leftarrow$  A[j]

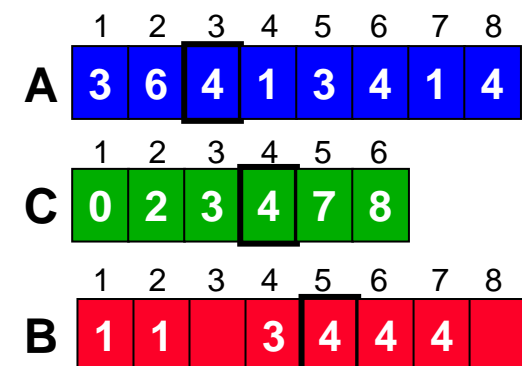
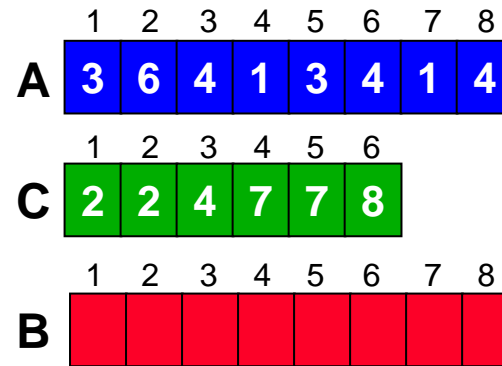
C[A[j]]  $\leftarrow$  C[A[j]] - 1



# Revisitando RADIX-SORT

```

COUNTING-SORT(A,B,k)
for i ← 1 to k
  do C[i] ← 0
for j ← 1 to length[A]
  do C[A[j]] ← C[A[j]] + 1
for i ← 2 to k
  do C[i] ← C[i] + C[i-1]
for j ← length[A] downto 1
  do B[C[A[j]]] ← A[j]
     C[A[j]] ← C[A[j]] - 1
    
```



**Ao final da sexta iteração do último laço**

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

    do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

    do C[A[j]]  $\leftarrow$  C[A[j]] + 1

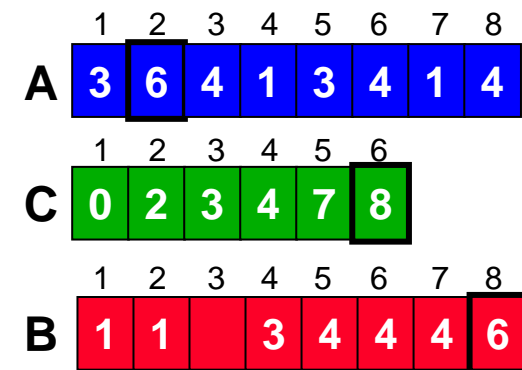
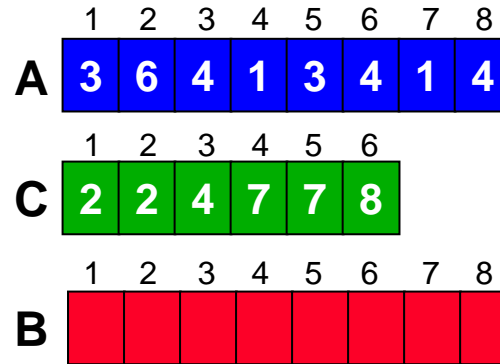
**for** i  $\leftarrow$  2 **to** k

    do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

    do B[C[A[j]]]  $\leftarrow$  A[j]

        C[A[j]]  $\leftarrow$  C[A[j]] - 1



# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

for i ← 1 to k

do C[i] ← 0

for j ← 1 to length[A]

do C[A[j]] ← C[A[j]] + 1

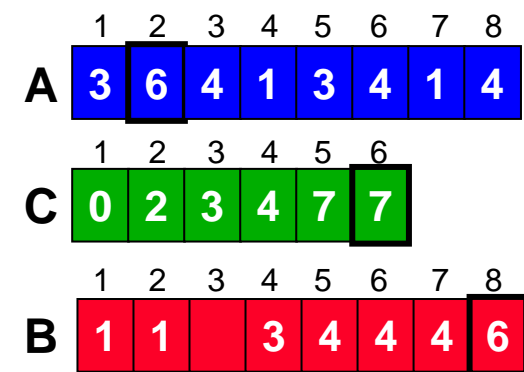
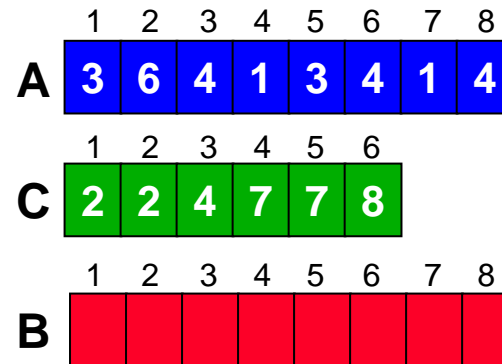
for i ← 2 to k

do C[i] ← C[i] + C[i-1]

for j ← length[A] downto 1

do B[C[A[j]]] ← A[j]

C[A[j]] ← C[A[j]] - 1



Ao final da sétima iteração do último laço



# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

**for** i  $\leftarrow$  1 **to** k

    do C[i]  $\leftarrow$  0

**for** j  $\leftarrow$  1 **to** length[A]

    do C[A[j]]  $\leftarrow$  C[A[j]] + 1

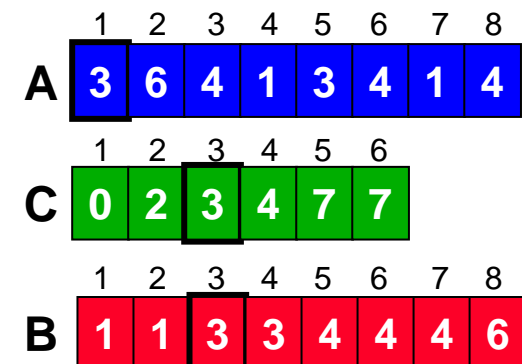
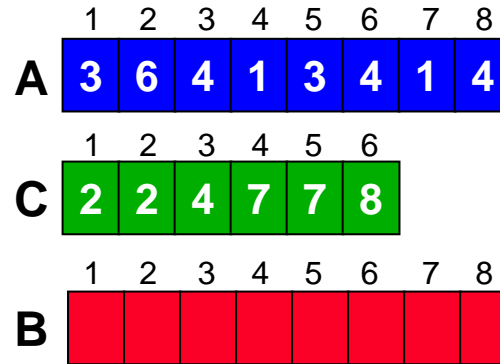
**for** i  $\leftarrow$  2 **to** k

    do C[i]  $\leftarrow$  C[i] + C[i-1]

**for** j  $\leftarrow$  length[A] **downto** 1

    do B[C[A[j]]]  $\leftarrow$  A[j]

        C[A[j]]  $\leftarrow$  C[A[j]] - 1



# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

for i ← 1 to k

do C[i] ← 0

for j ← 1 to length[A]

do C[A[j]] ← C[A[j]] + 1

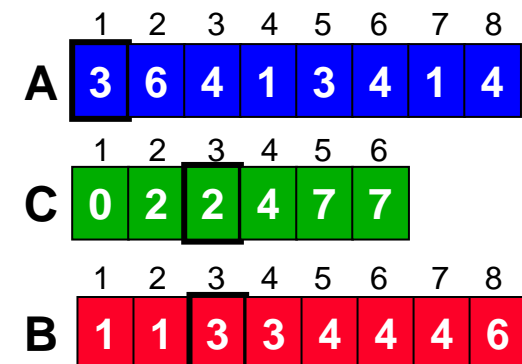
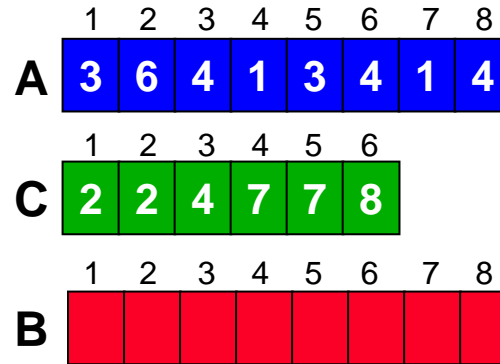
for i ← 2 to k

do C[i] ← C[i] + C[i-1]

for j ← length[A] downto 1

do B[C[A[j]]] ← A[j]

C[A[j]] ← C[A[j]] - 1



Ao final da última iteração do último laço

# Revisitando RADIX-SORT

COUNTING-SORT(A,B,k)

```

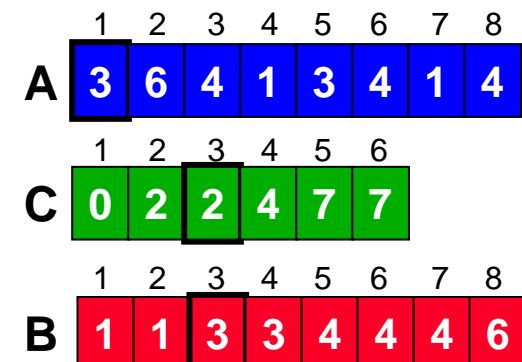
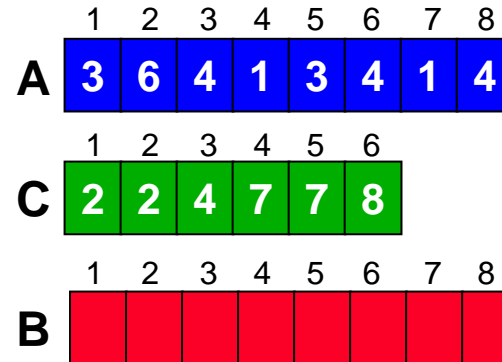
for i ← 1 to k
  do C[i] ← 0
for j ← 1 to length[A]
  do C[A[j]] ← C[A[j]] + 1
for i ← 2 to k
  do C[i] ← C[i] + C[i-1]
for j ← length[A] downto 1
  do B[C[A[j]]] ← A[j]
     C[A[j]] ← C[A[j]] - 1
    
```

RADIX-SORT(A,d)

```

for i ← 1 to d
  do use COUNTING-SORT
     to sort A on digit i
    
```

**Requer mais memória  
por item a ordenar do  
que QuickSort**



**Localidade temporal (instruções)**

Grande: laços

**Localidade espacial (instruções)**

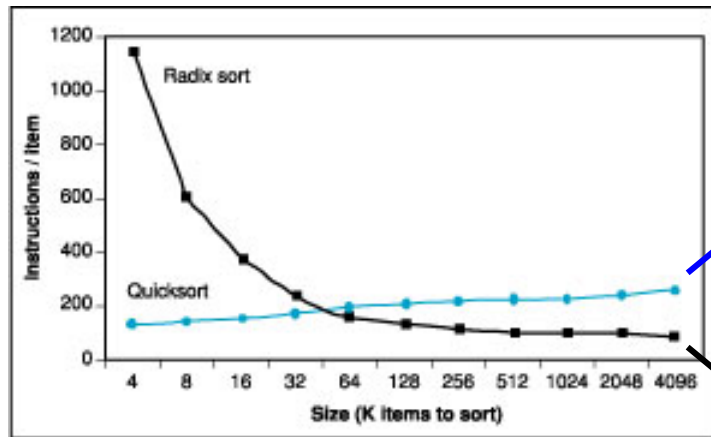
Pouca: BBs pequenos

**Localidade espacial (dados)**

Pequena: só um dos arranjos sequencialmente

**Localidade temporal (dados)**

Média: C e A são revisitados várias vezes

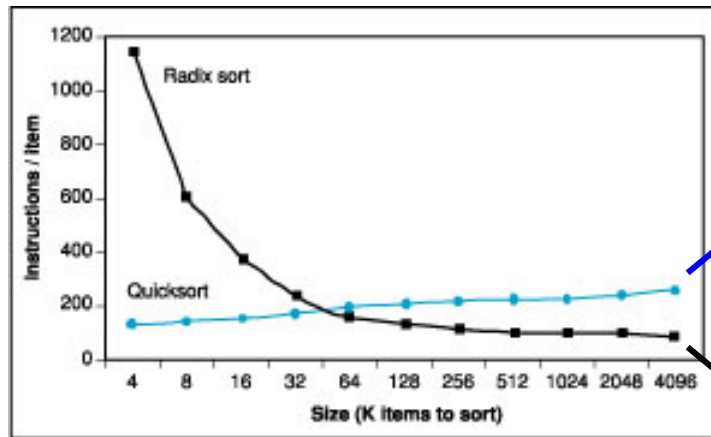


$O(n \log n)$

$O(n)$

**Impacto da  
cache no  
desempenho**

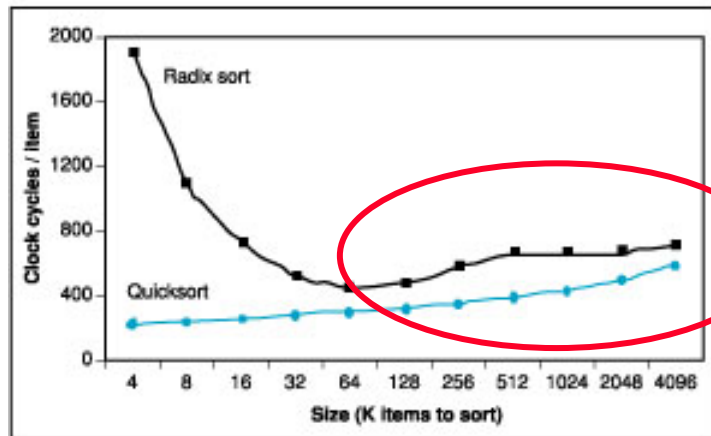
Instruções / item



$O(n \log n)$

$O(n)$

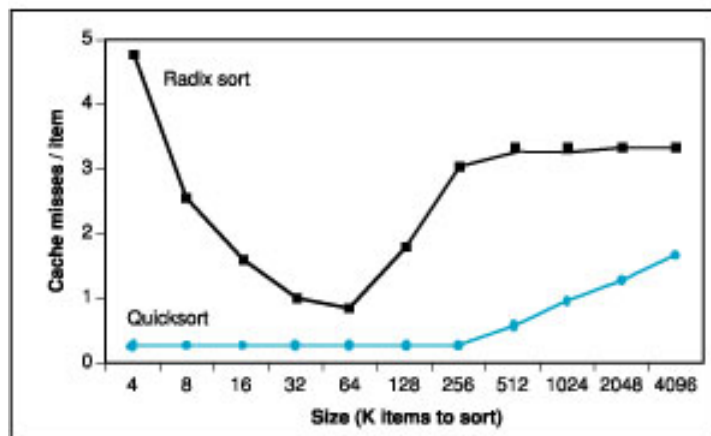
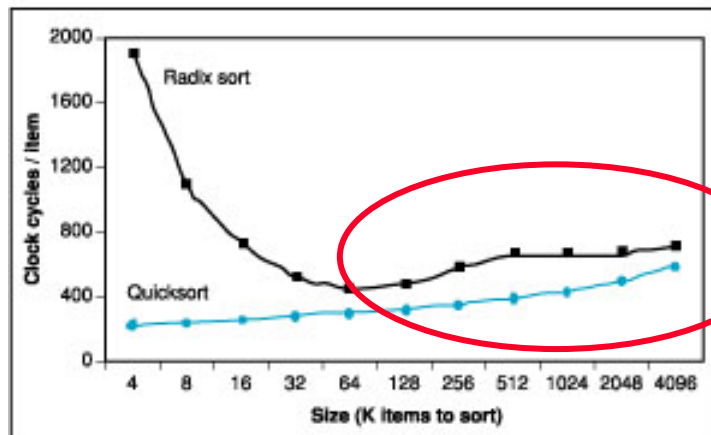
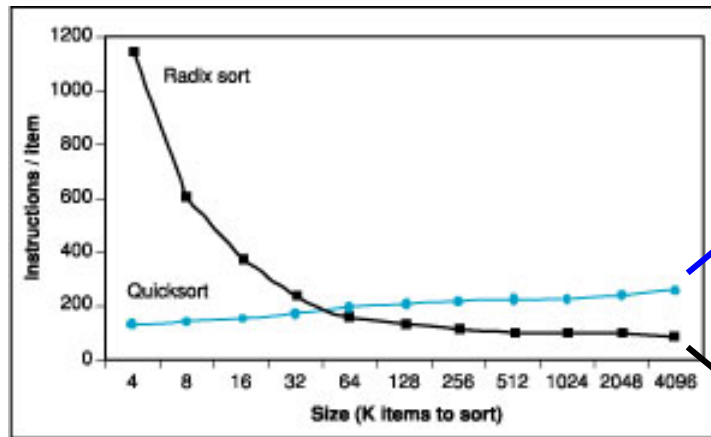
# Impacto da cache no desempenho



Instruções / item

Ciclos / item

# Impacto da cache no desempenho



Instruções / item

Ciclos / item

Faltas / item

Maior número de ciclos gastos por item deve-se à menor localidade de RadixSort

Otimizar para a cache: algoritmo e compilador

# Conclusão

- **Para obter desempenho com:**
  - CPUs com frequências cada vez mais altas
  - DRAMs cada vez mais lentas (comparativamente)
- **Soluções:**
  - Aumento da associatividade
    - » Impacta mr
  - Múltiplos níveis de cache
    - » Impacta penalidade
  - Algoritmos e compiladores “cache-conscientes”
    - » Impactam mr