# Bases de Datos

## Clase 9: Evaluación de consultas

# EVALUACIÓN DE CONSULTAS

Capítulo 13 | Ramakrishnan / Gehrke

# ¿Cómo evaluar una consulta?

1. Análisis sintáctico: ¿está bien escrita?
2. Análisis semántico: ¿qué sentido tiene?
3. ¿Qué posibilidades tenemos para ejecutarla?
4. ¿Cuál es la posibilidad más conveniente (más rápida, más “barata”, etc.)
5. Tomar la decisión

# Usaremos esta BD de ejemplo

***Sailors***

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

***Reserves (de botes)***

| sid | bid | day |
|---|---|---|
| 22 | 101 | 10/10/96 |
| 58 | 103 | 11/12/96 |

# Análisis sintáctico

```
SELECT sname
FROM Reserves NATURAL JOIN Zailors
WHERE bid = 100 AND rating > 5
```

# Análisis sintáctico

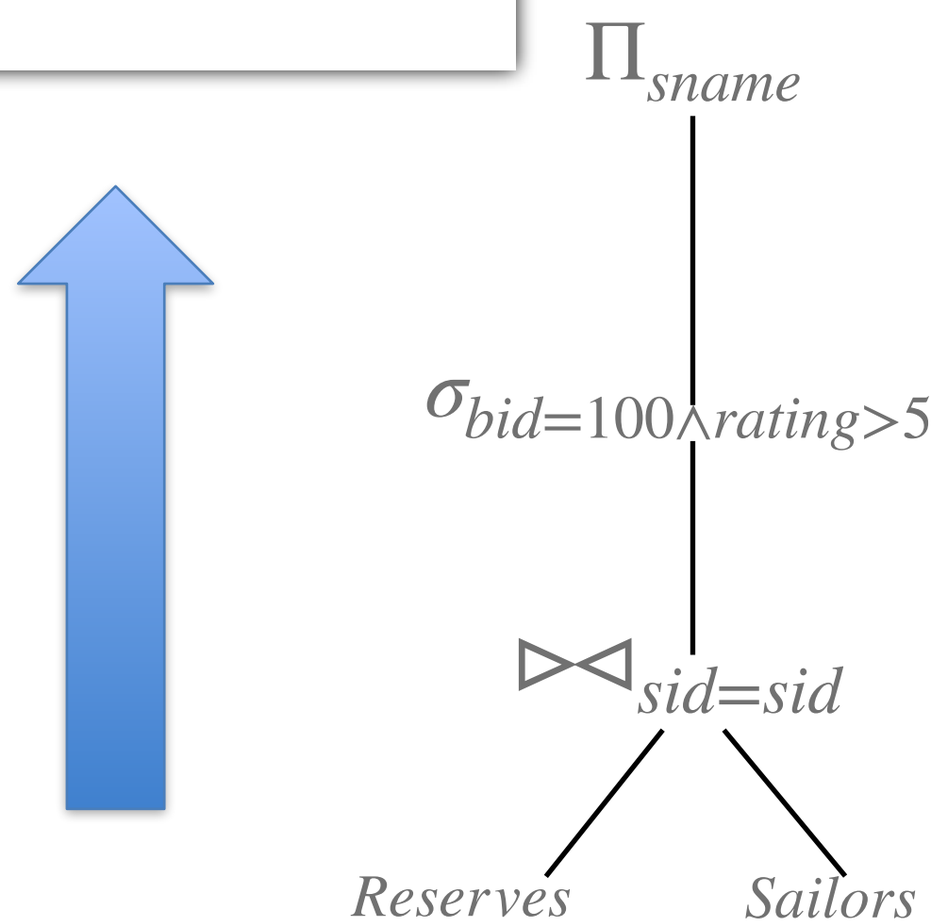
```
SELECT sname
FROM Reserves NATURAL JOIN Zailors
WHERE bid = 100 AND rating > 5
```

ERROR: tabla Zailors no existe!  (es Sailors)

# Análisis semántico: árbol de consulta

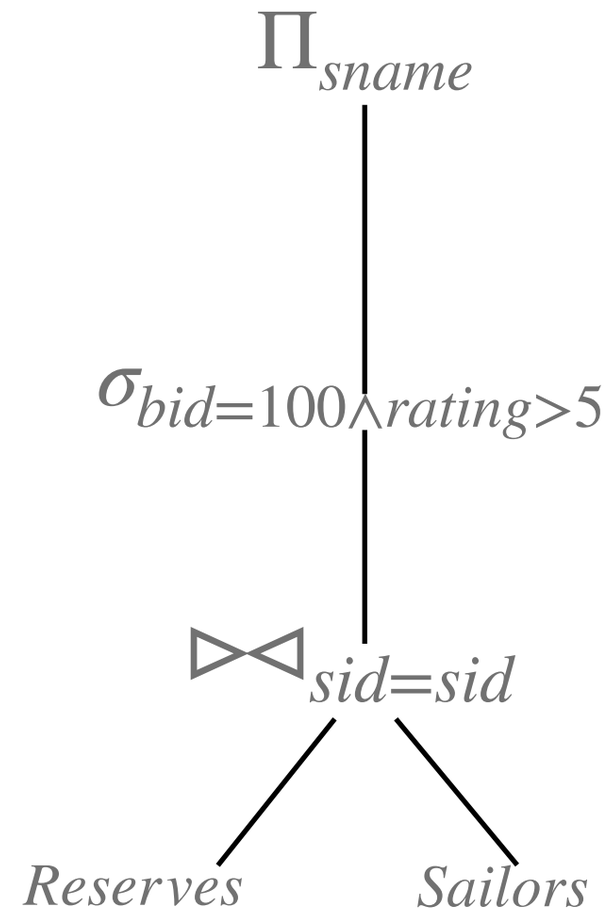
```
SELECT sname
FROM Reserves NATURAL JOIN Sailors
WHERE bid = 100 AND rating > 5
```

Permite “ordenar”una manera de organizar el flujo de los datos



# ¿Cuánto cuesta?

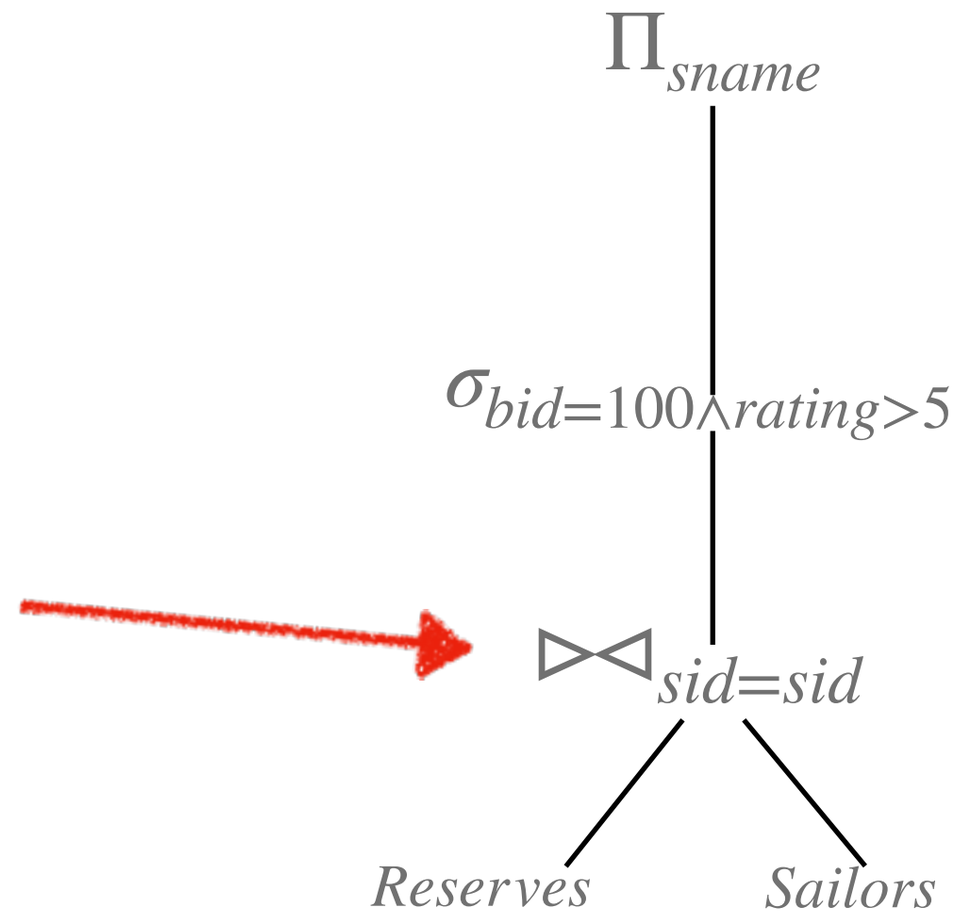
Depende de:



# ¿Cuánto cuesta?

Depende de:

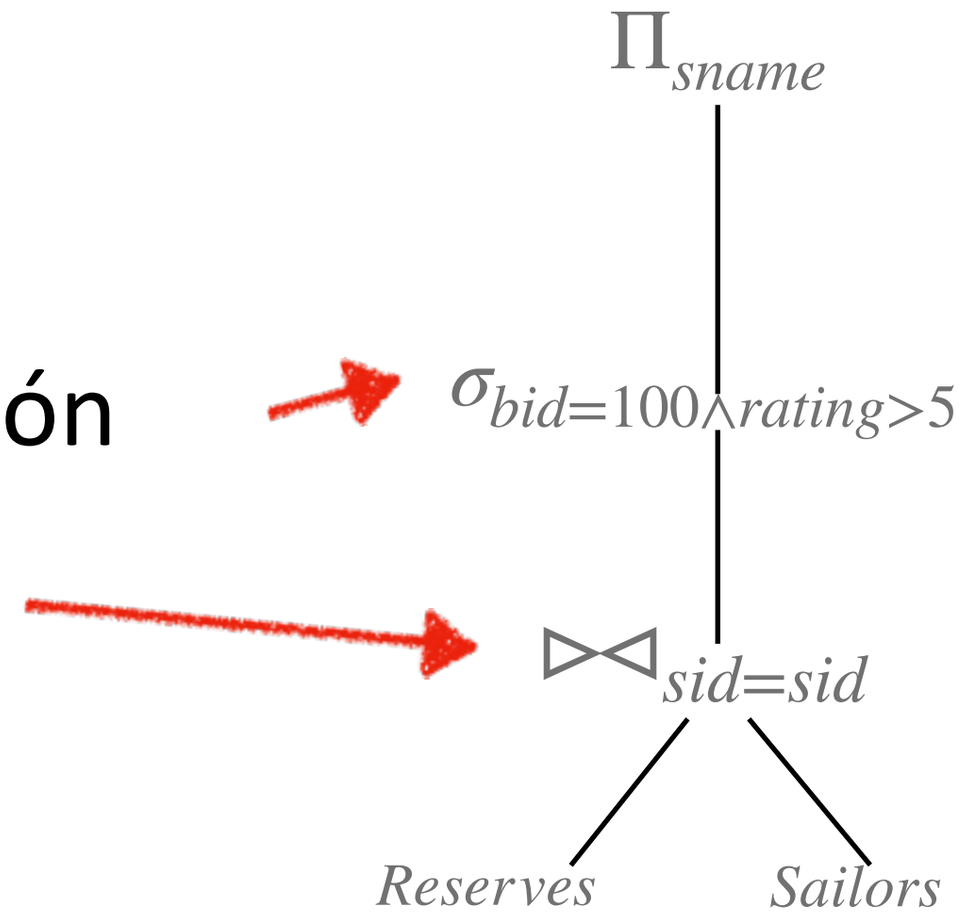
- Cómo evaluamos el join



# ¿Cuánto cuesta?

Depende de:

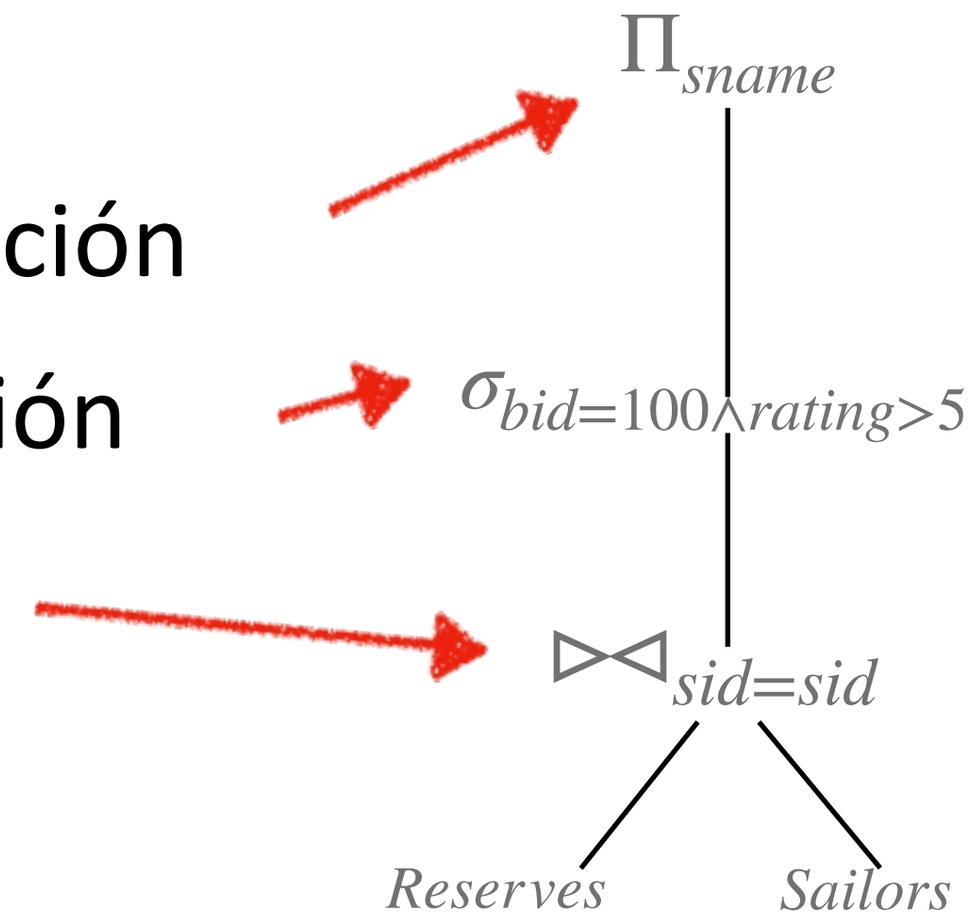
- Cómo evaluamos la condición
- Cómo evaluamos el join



# ¿Cuánto cuesta?

Depende de:

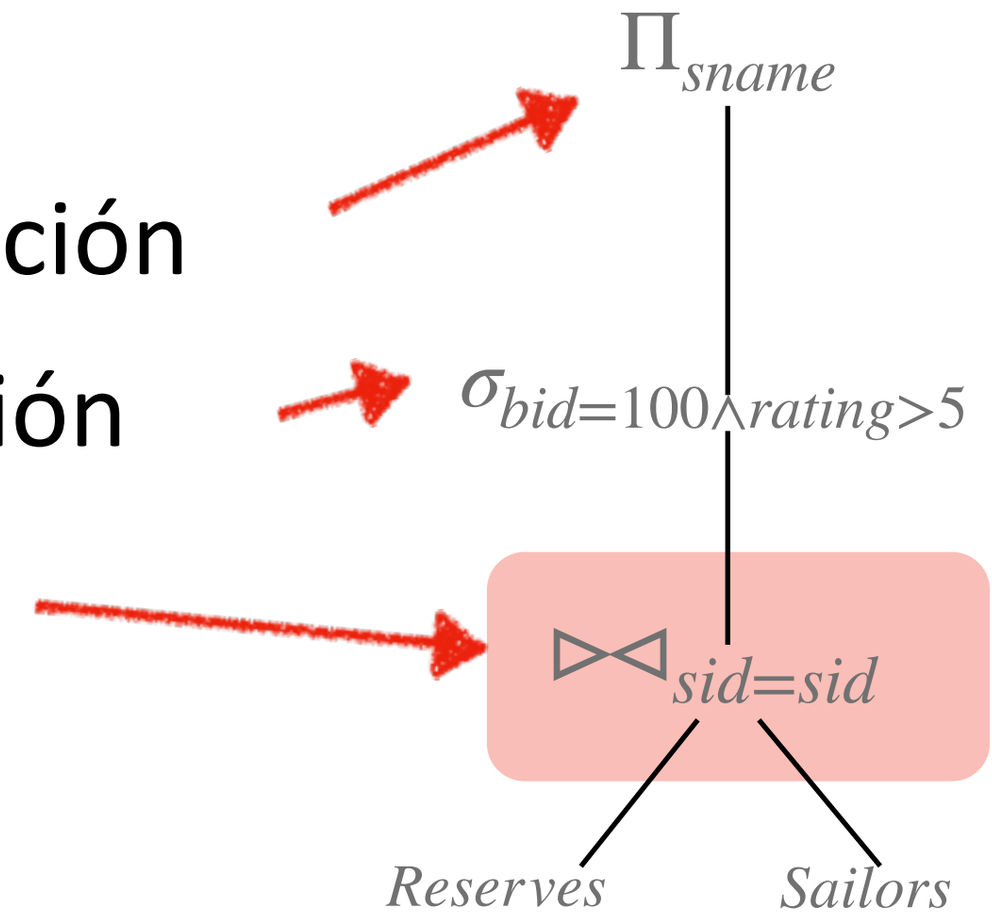
- Cómo evaluamos la proyección
- Cómo evaluamos la condición
- Cómo evaluamos el join



# ¿Cuánto cuesta?

Depende de:

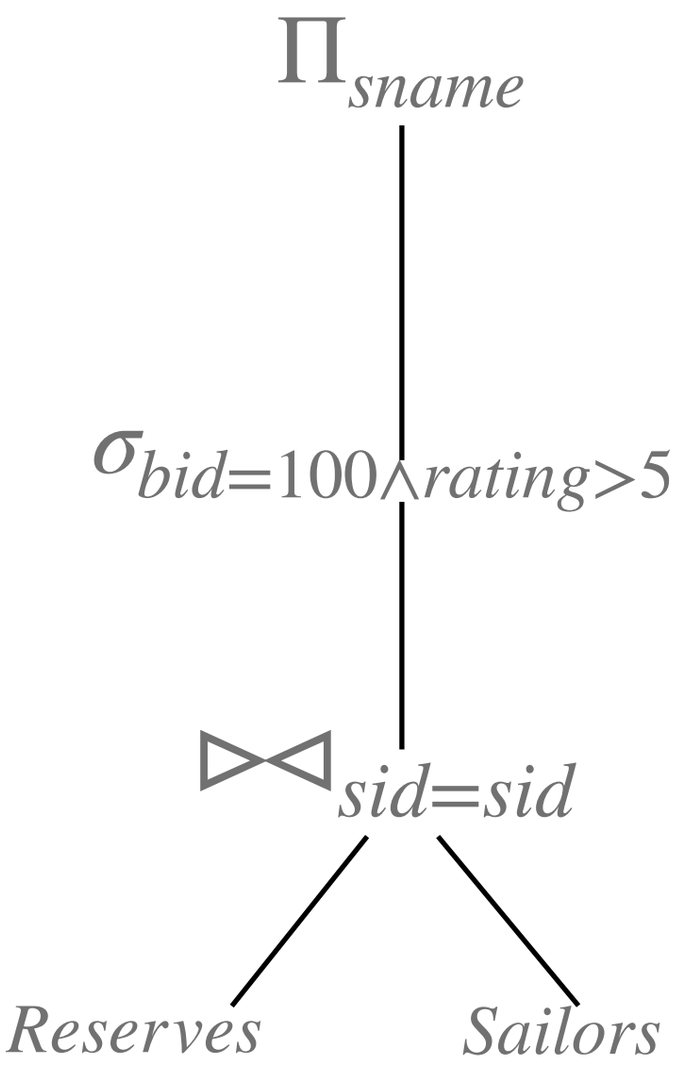
- Cómo evaluamos la proyección
- Cómo evaluamos la condición
- Cómo evaluamos el join




Usualmente:

- El join es el más relevante
- Los otros se evalúan “al vuelo”

# Posibles evaluaciones del Join




-Loop anidado (Nested Loop Join)

-Index Nested Loop Join

-Hash Join

-Sort-Merge Join

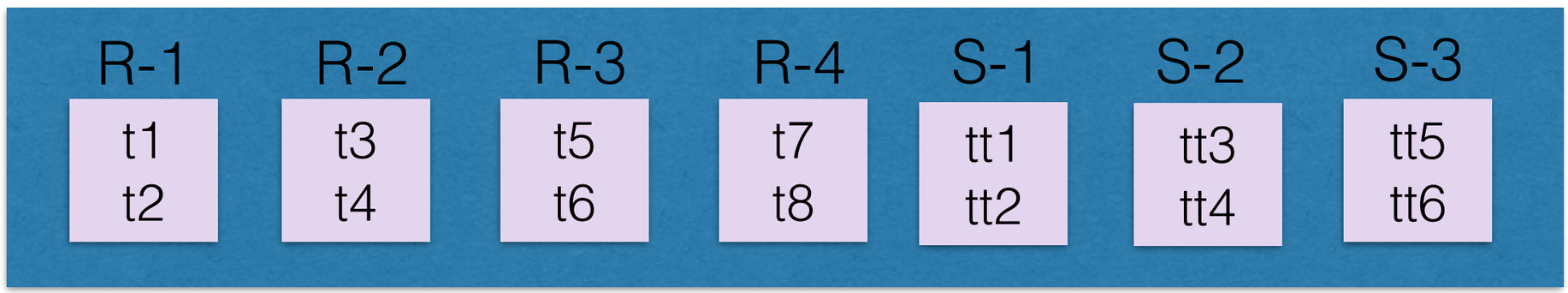
# Páginas, disco y buffer

Para trabajar con las tuplas de una relación, la base de datos carga la página desde el disco con dicha tupla

Para cargar estas páginas, la base de datos reserva un espacio en RAM llamado Buffer

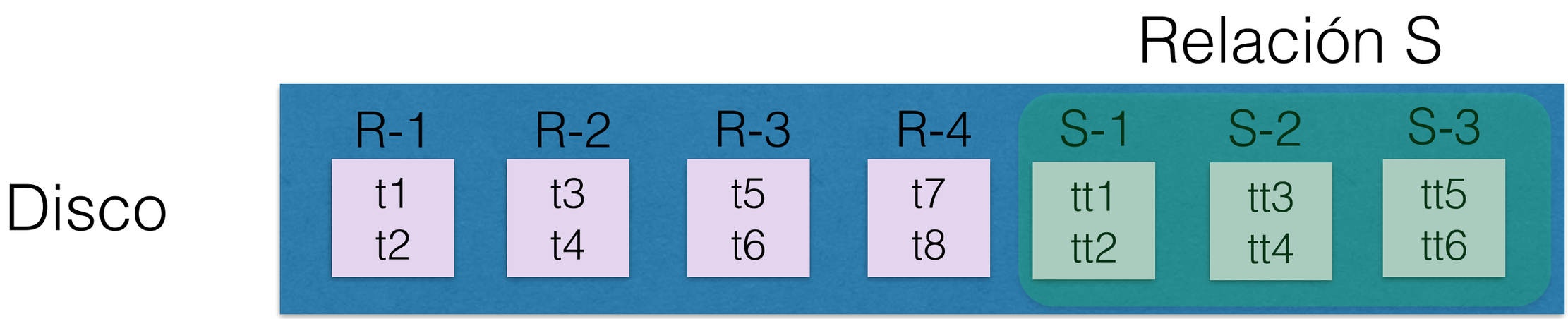
# Páginas, disco y buffer

Disco



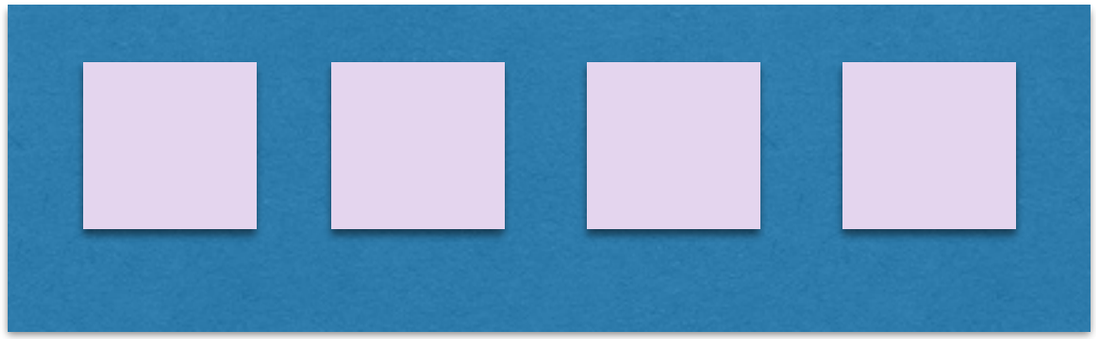
# Páginas, disco y buffer

# Páginas, disco y buffer

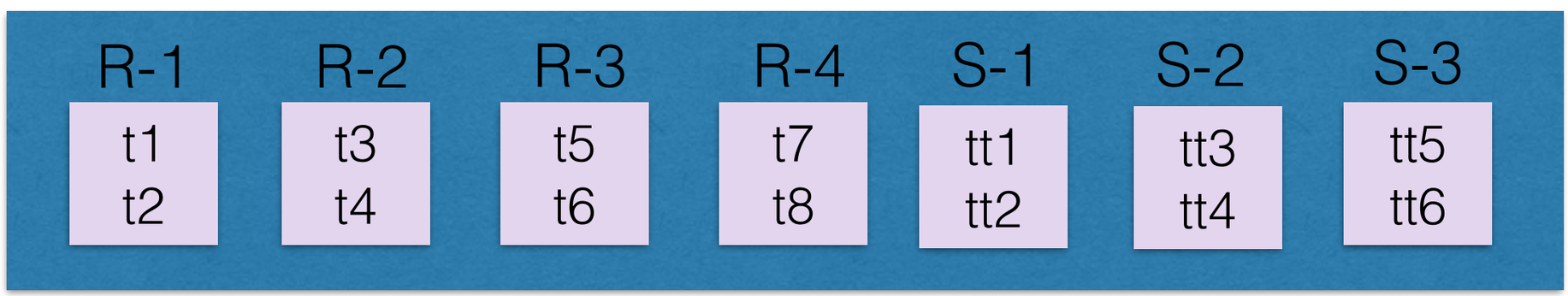


# Páginas, disco y buffer

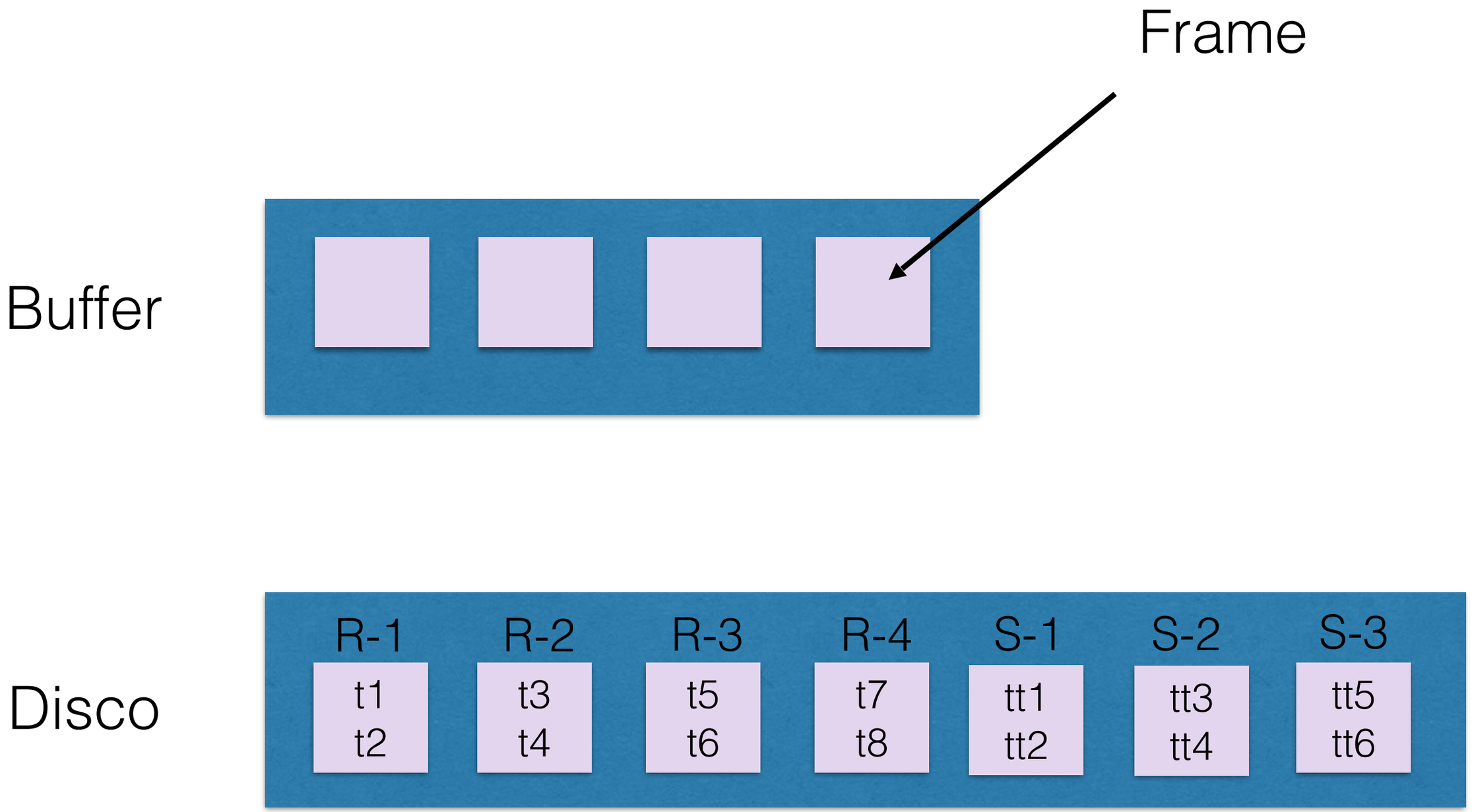
Buffer



Disco



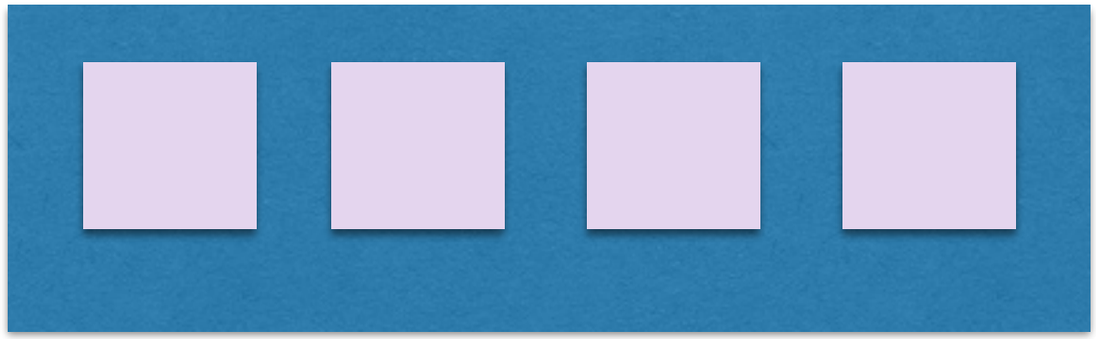
# Páginas, disco y buffer



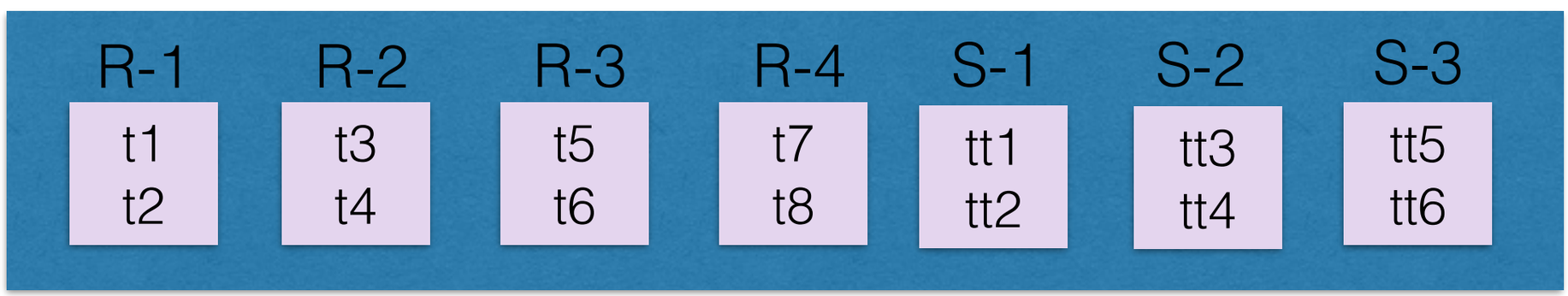
# Páginas, disco y buffer

DB

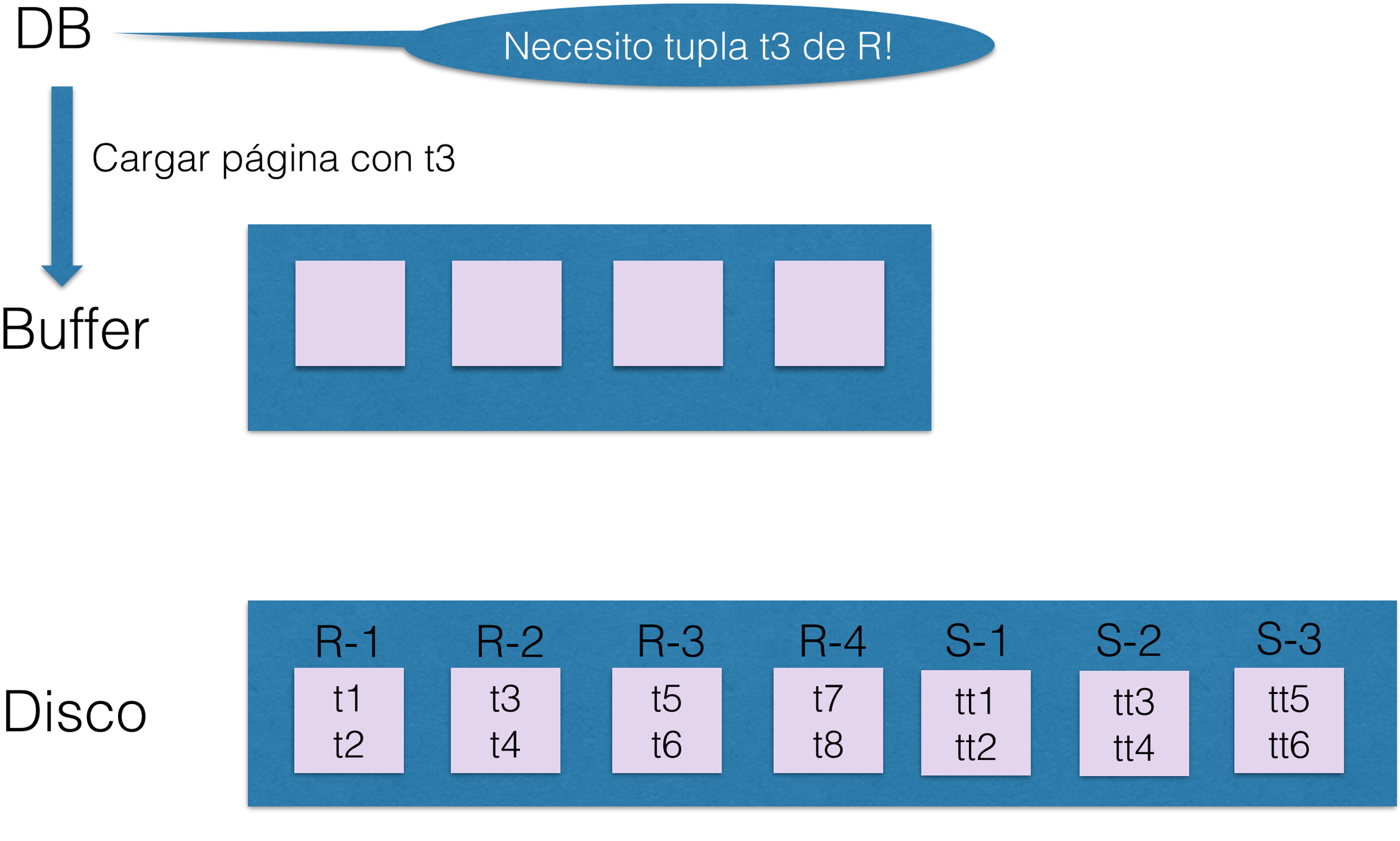

Buffer



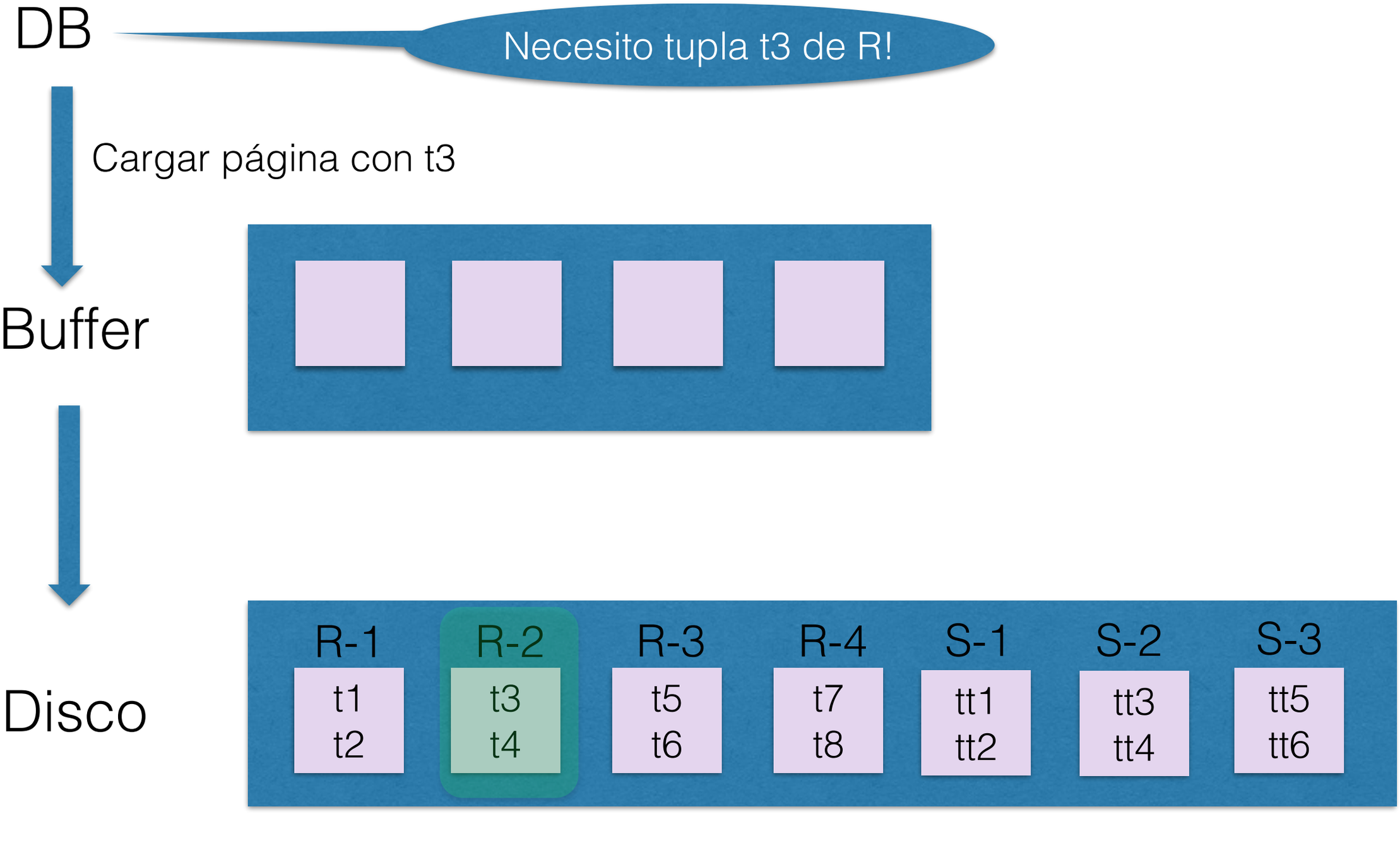
Disco



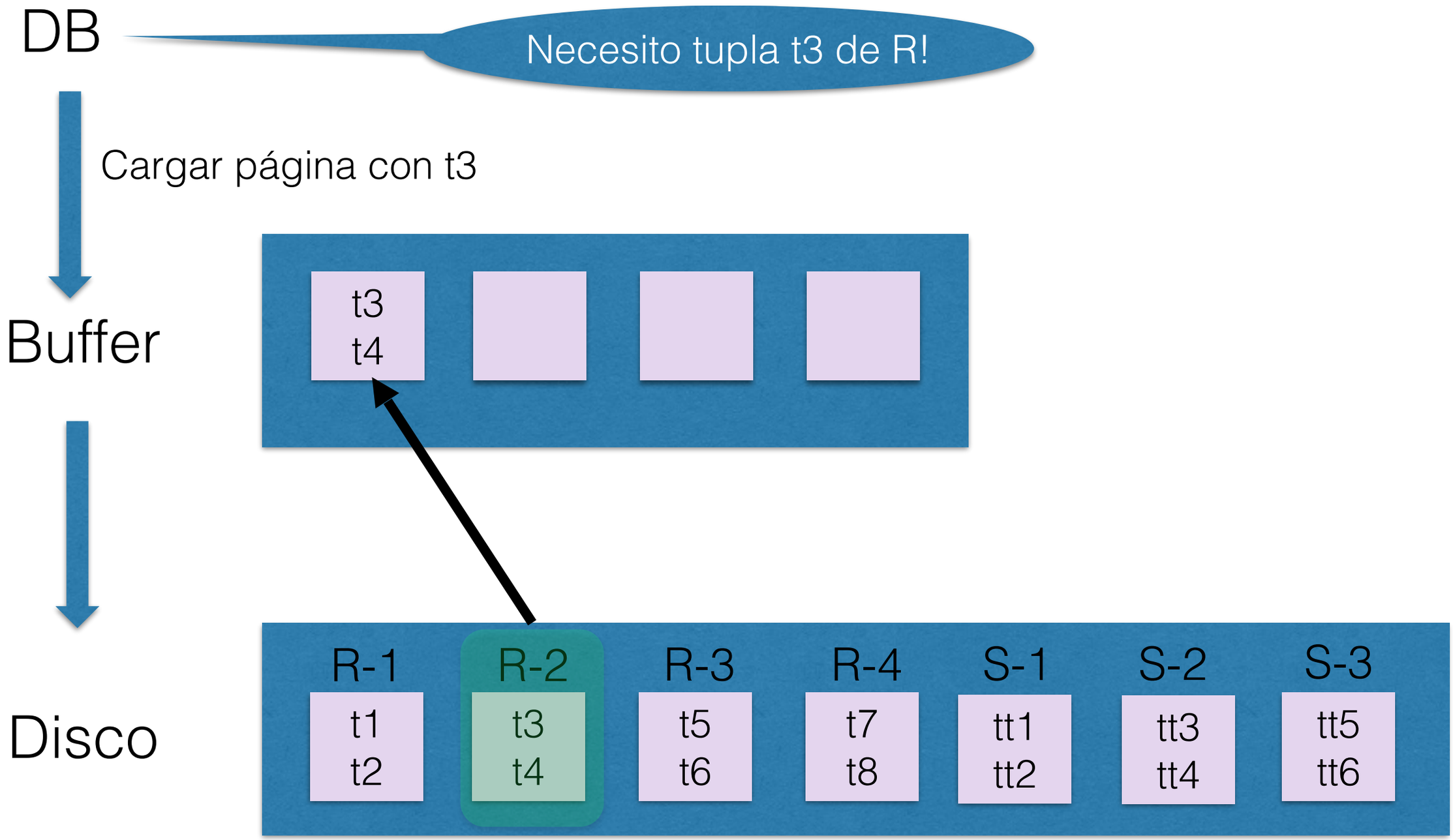
# Páginas, disco y buffer



# Páginas, disco y buffer



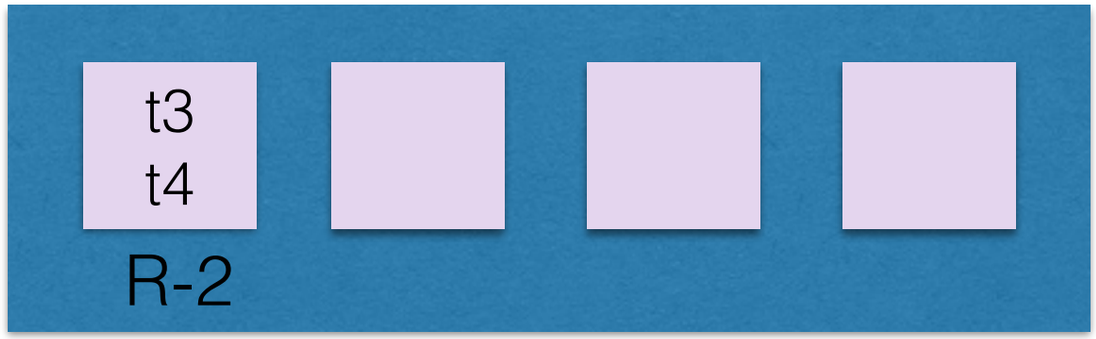
# Páginas, disco y buffer



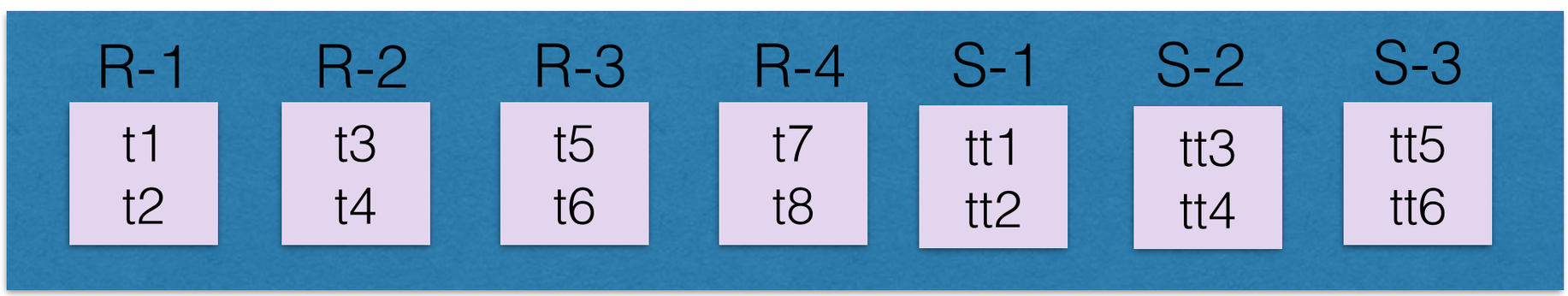
# Páginas, disco y buffer

DB

Buffer




Disco

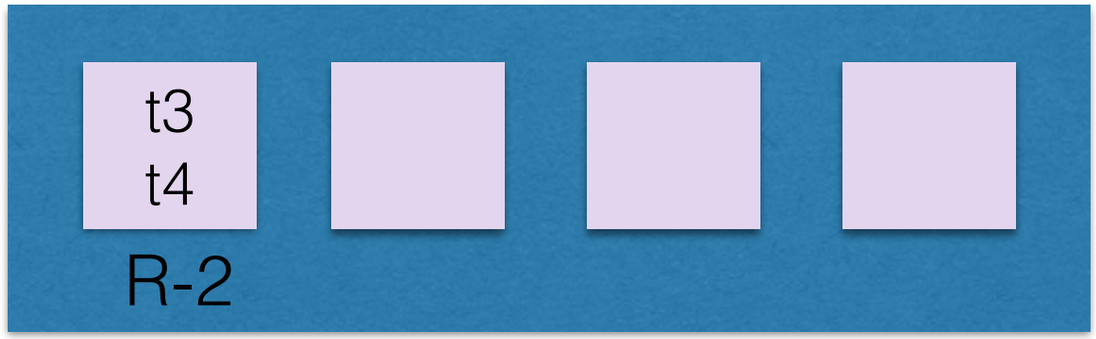


# Páginas, disco y buffer

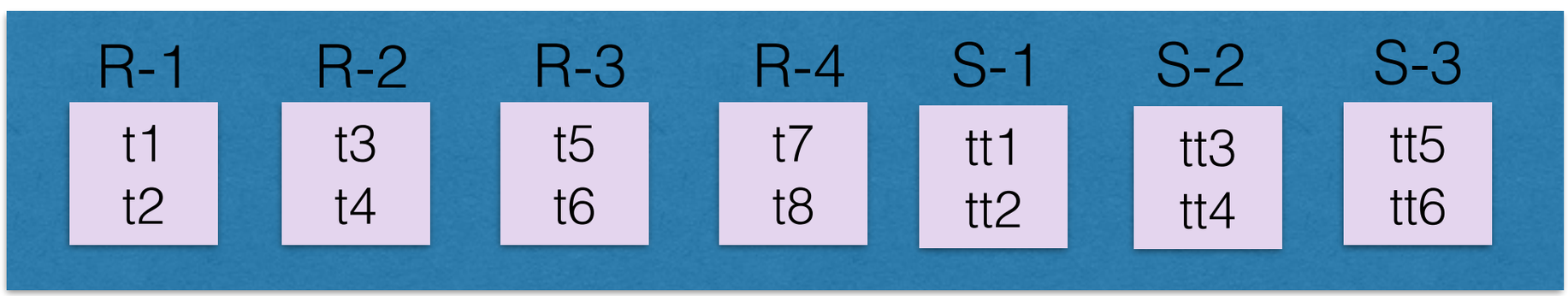
DB

Necesito tupla t4 de R!

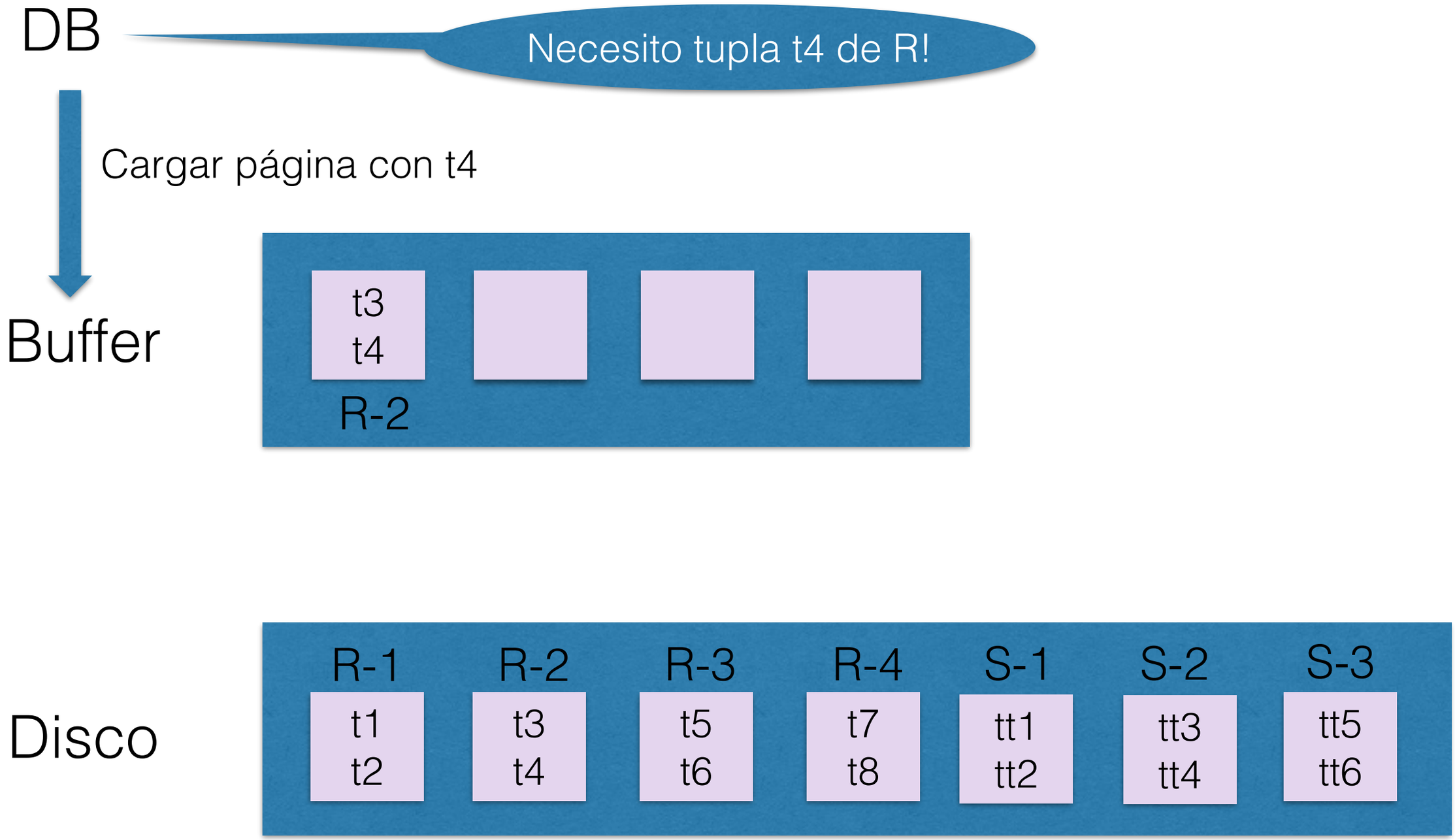
Buffer



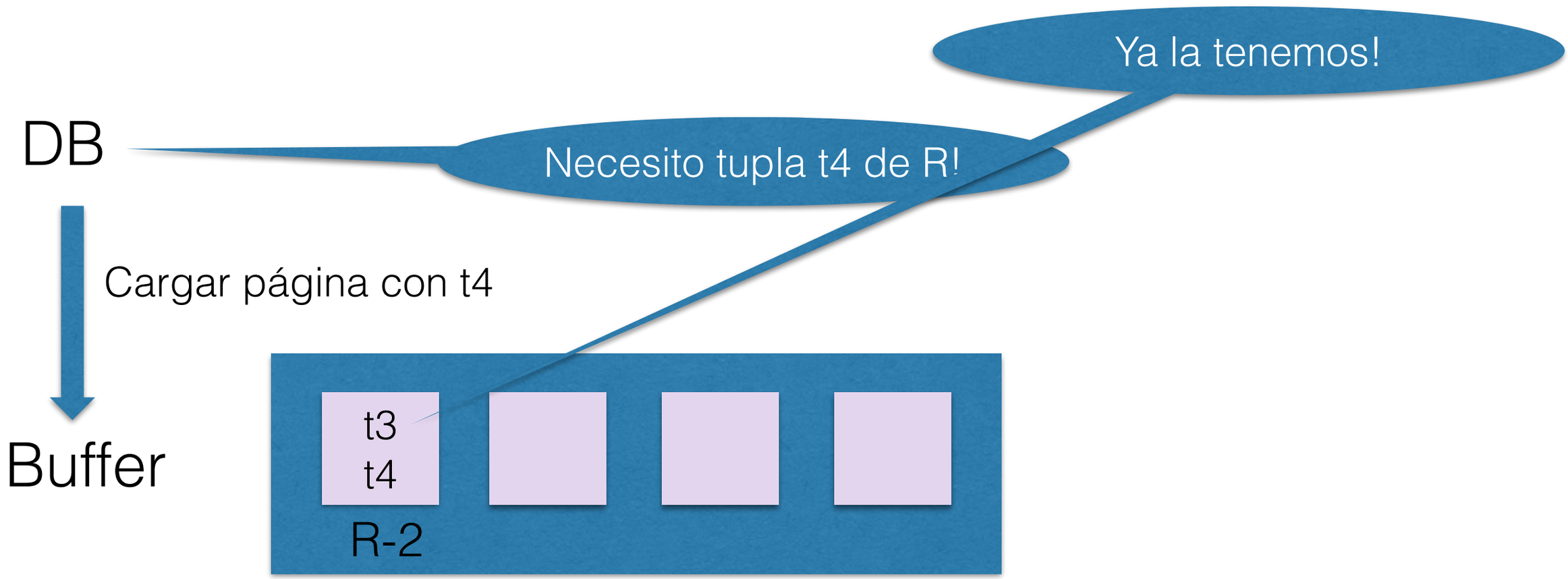

Disco

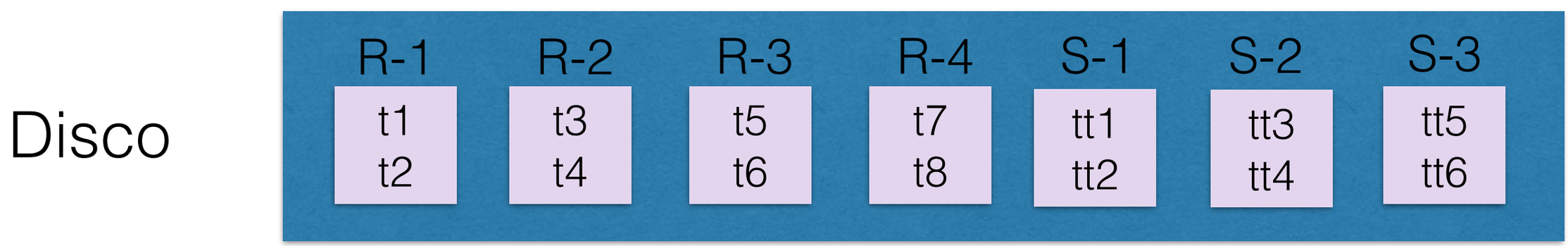


# Páginas, disco y buffer



# Páginas, disco y buffer



# Costo de un algoritmo

Cuantas veces tengo que leer una página desde el disco, o escribir una página al disco!

# Costo de un algoritmo

Cuantas veces tengo que leer una página desde el disco, o escribir una página al disco!

Las operaciones en buffer (RAM) son orden(es) de magnitud más rápidas que leer/escribir al disco – costo 0

# Proyección

$\pi_{attrs}(R)$

- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

# Proyección

$\pi_{attrs}(R)$

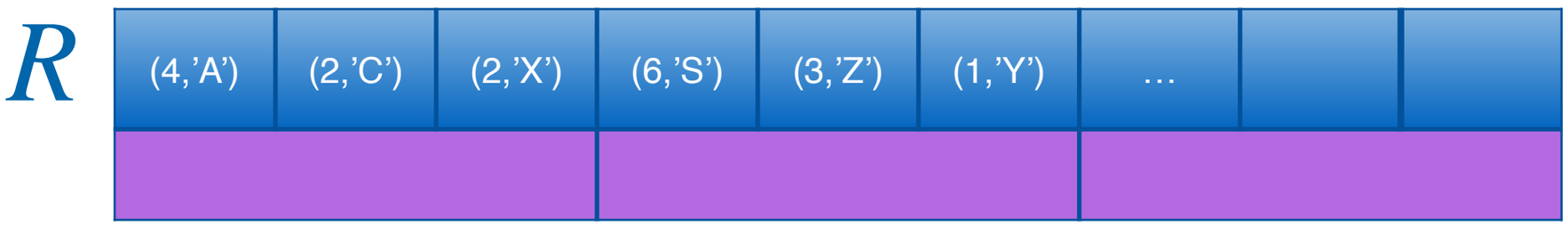
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(<u>id</u>: integer, *category*: string)

# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(<u>id</u>: integer, *category*: string)



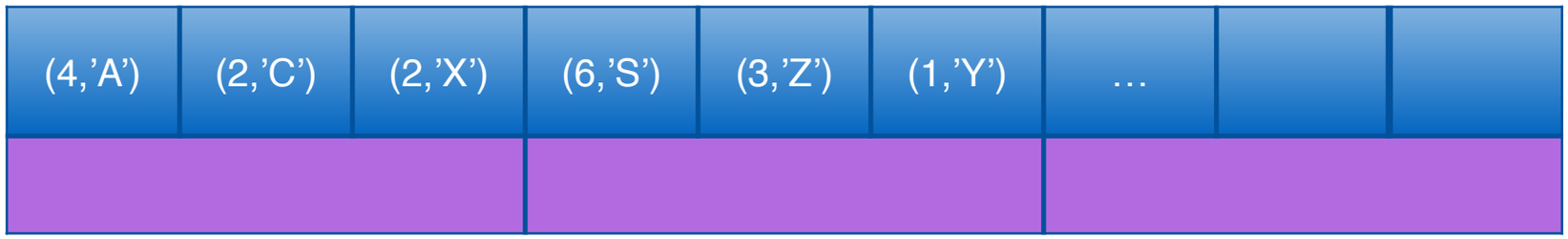
# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(<u>id</u>: integer, *category*: string)

$\pi_{attrs}(R)$

$R$

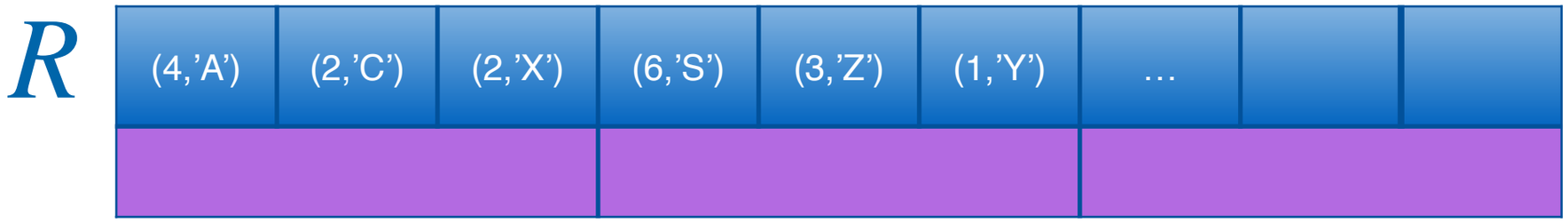


# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(<u>*id*</u>: integer, *category*: string)

$\pi_{attrs}(R)$



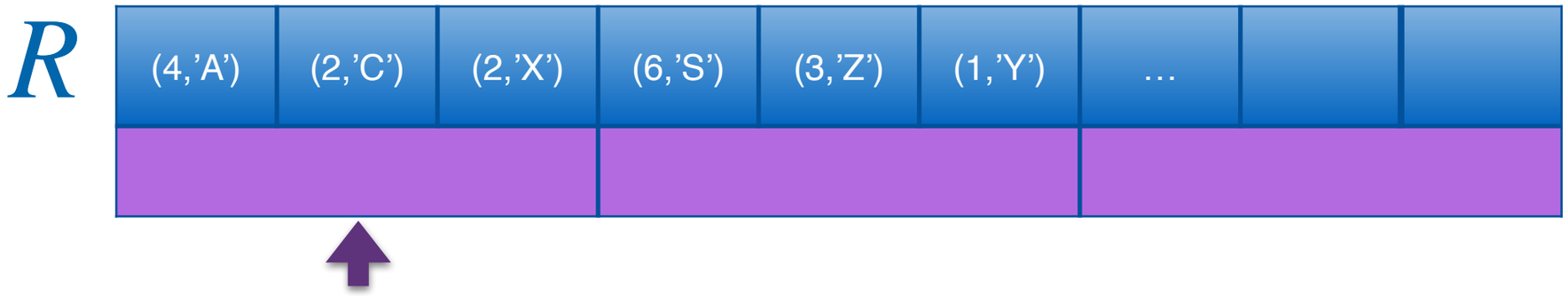

$attrs \triangleq category$

# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(<u>*id*</u>: integer, *category*: string)

$\pi_{attrs}(R)$

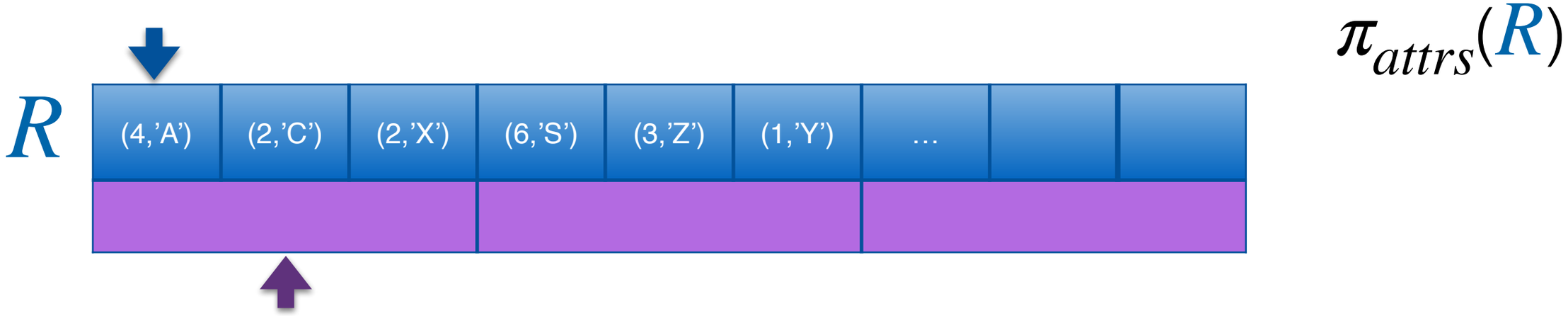



$attrs \triangleq category$

# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(<u>*id*</u>: integer, *category*: string)

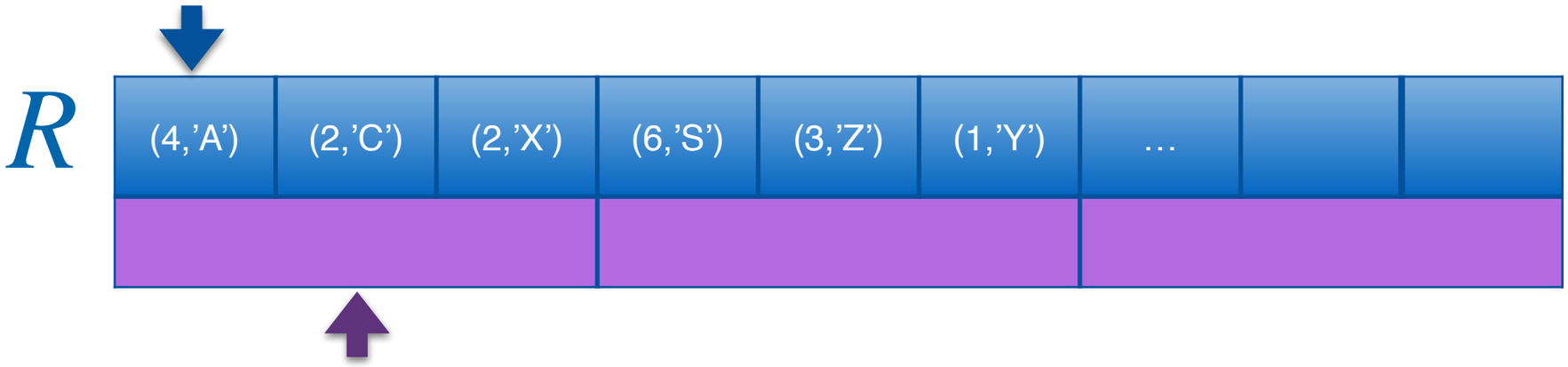



$attrs \triangleq category$

# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(*id*: integer, *category*: string)




$\pi_{attrs}(R)$

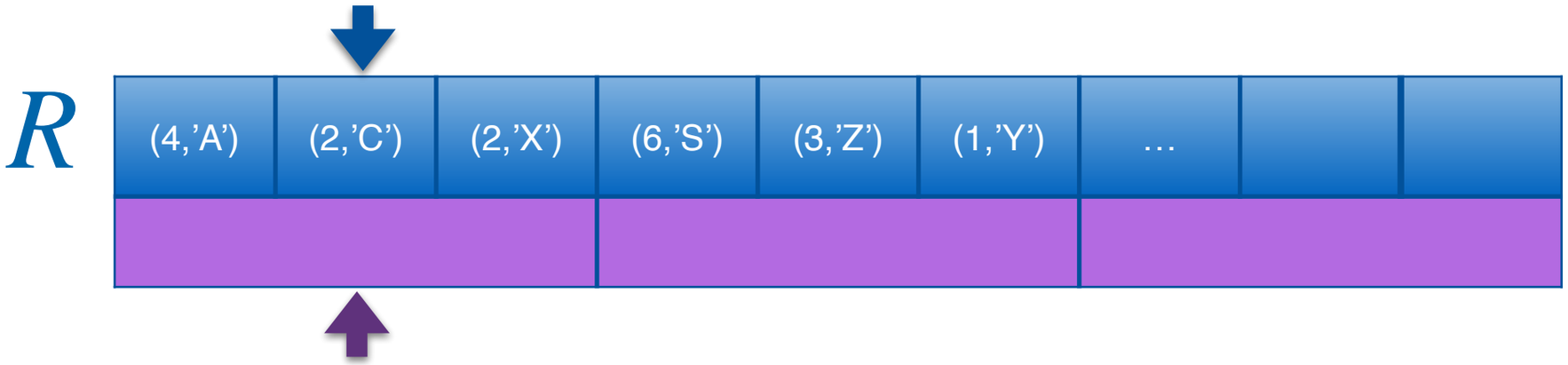
'A'

$attrs \triangleq category$

# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(*id*: integer, *category*: string)




$\pi_{attrs}(R)$

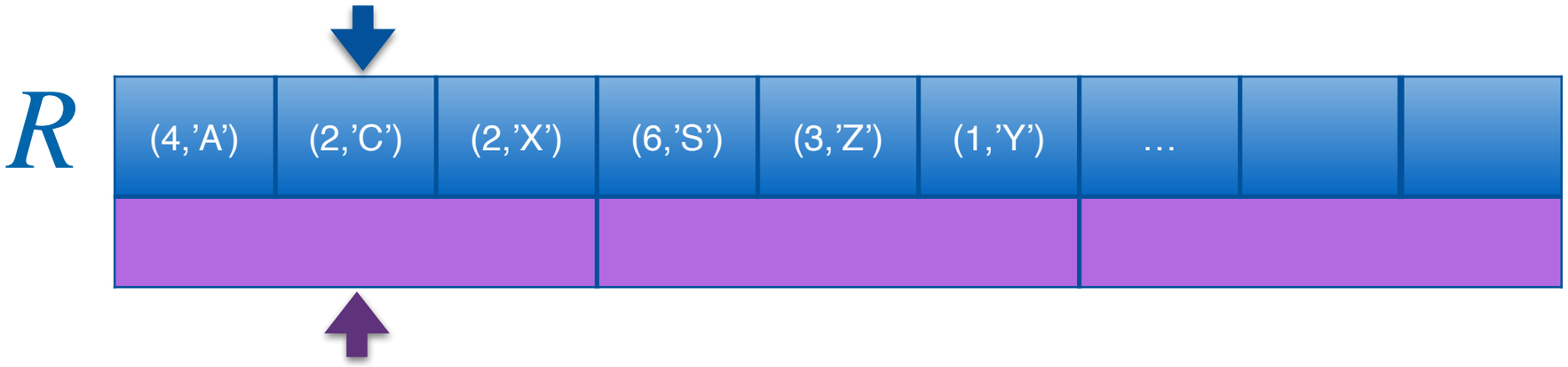
'A'

$attrs \triangleq category$

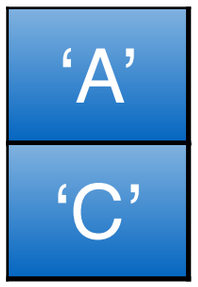
# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(*id*: integer, *category*: string)




$\pi_{attrs}(R)$

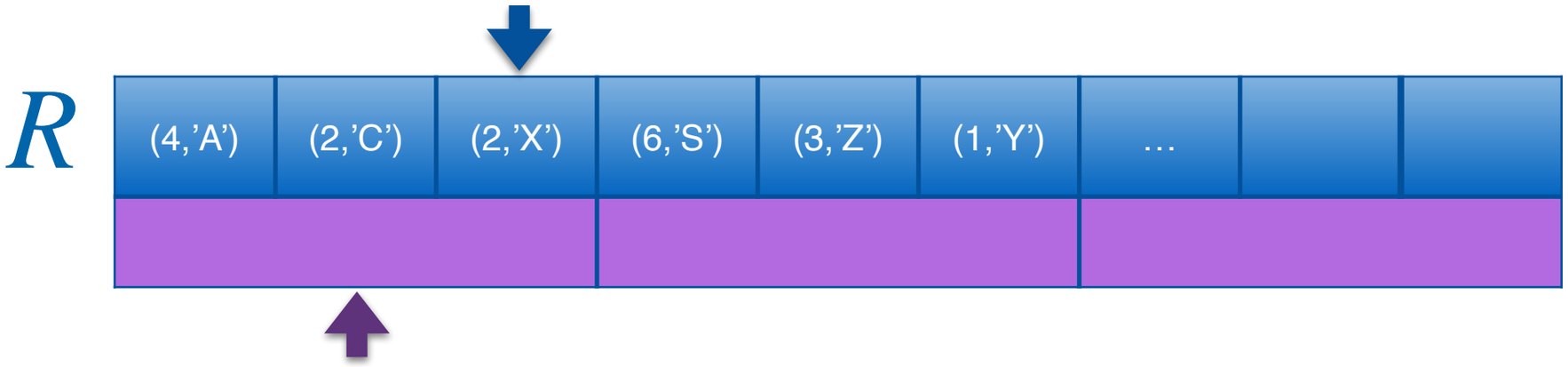



$attrs \triangleq category$

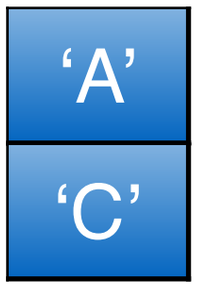
# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(*id*: integer, *category*: string)




$\pi_{attrs}(R)$

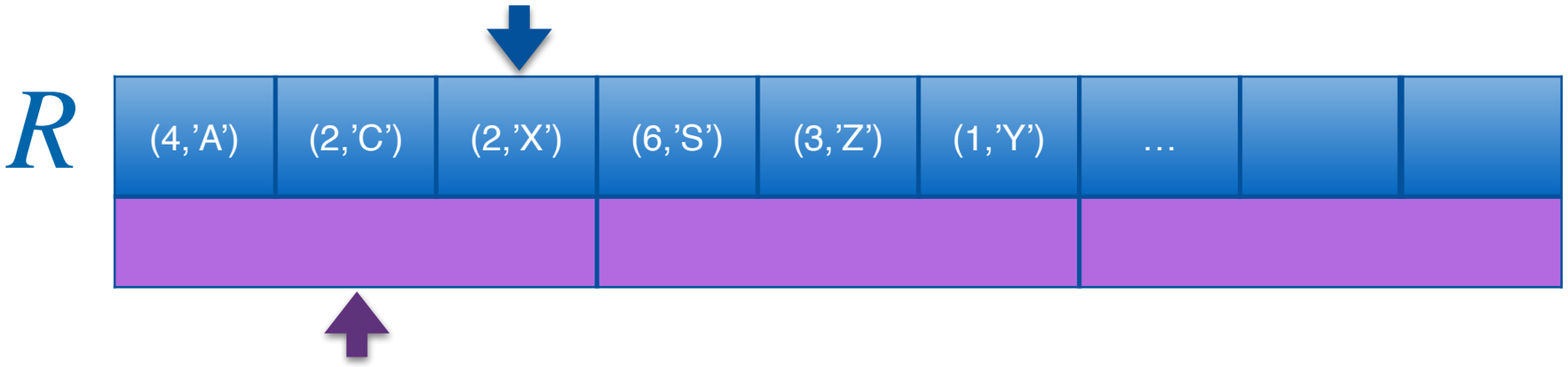



$attrs \triangleq category$

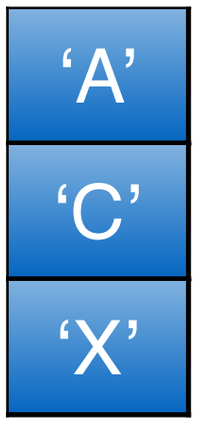
# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(*id*: integer, *category*: string)




$\pi_{attrs}(R)$

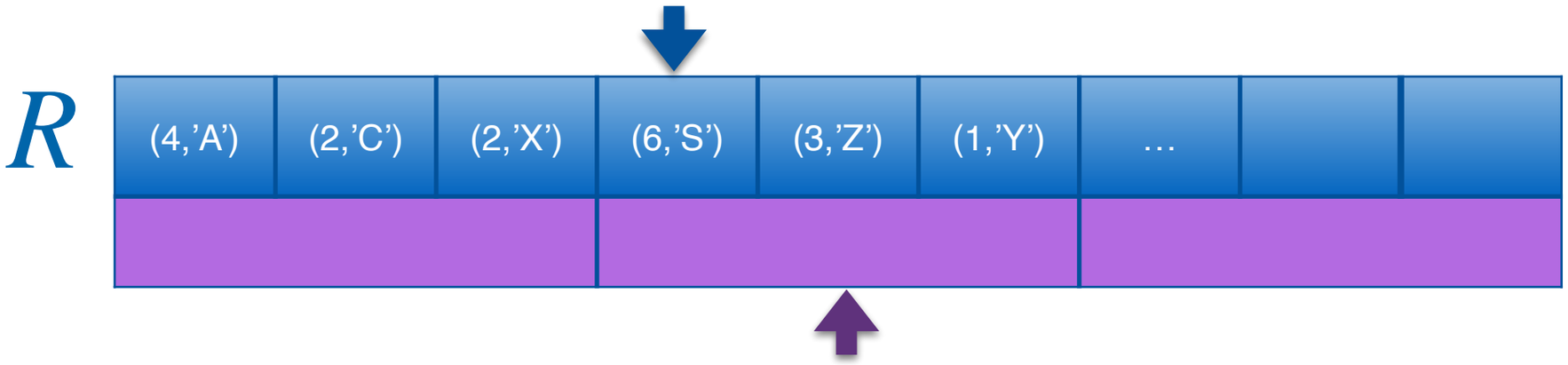



$attrs \triangleq$ *category*

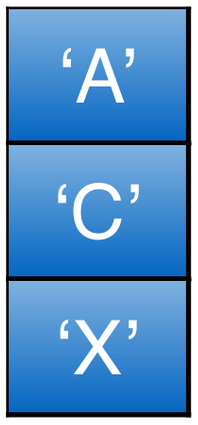
# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(*id*: integer, *category*: string)




$\pi_{attrs}(R)$

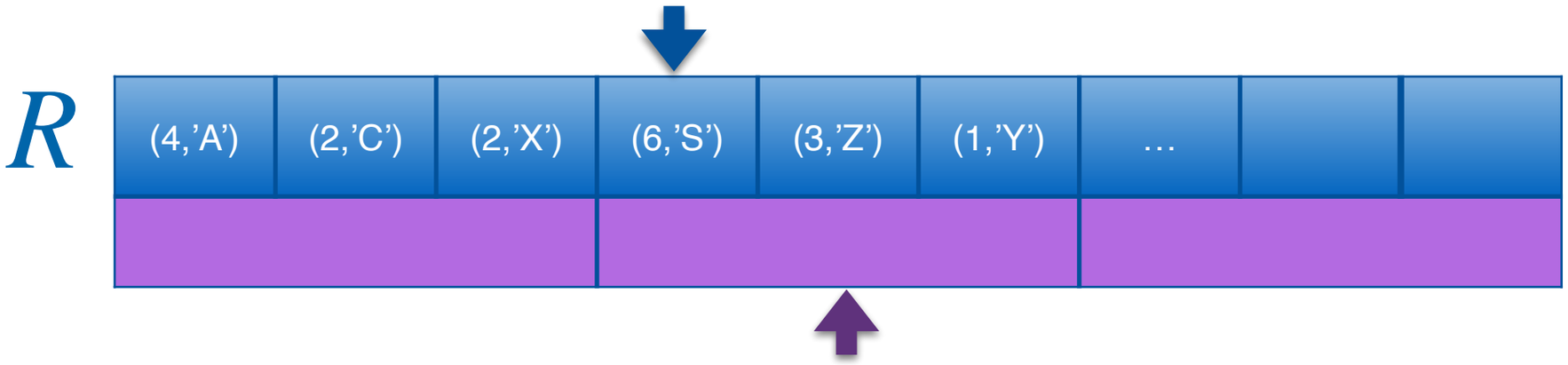



$attrs \triangleq$ *category*

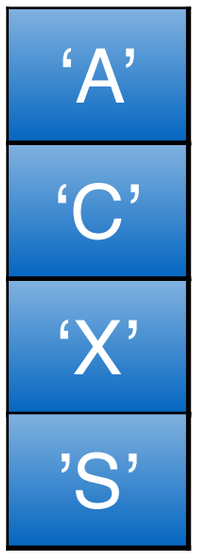
# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(*id*: integer, *category*: string)




$\pi_{attrs}(R)$

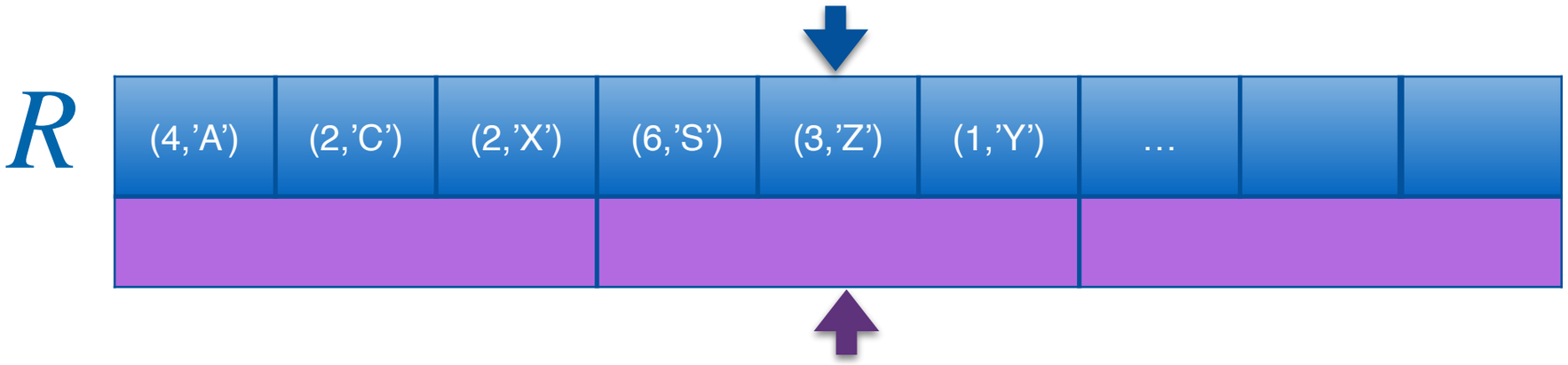



$attrs \triangleq category$

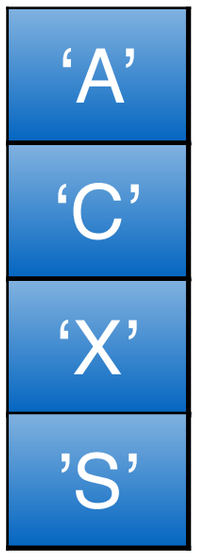
# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(*id*: integer, *category*: string)




$\pi_{attrs}(R)$

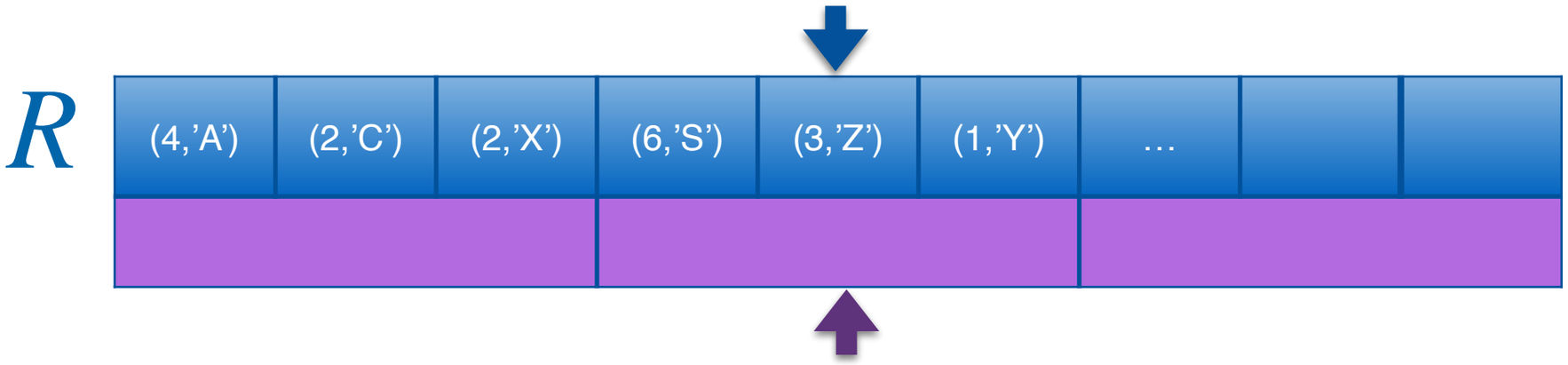



$attrs \triangleq category$

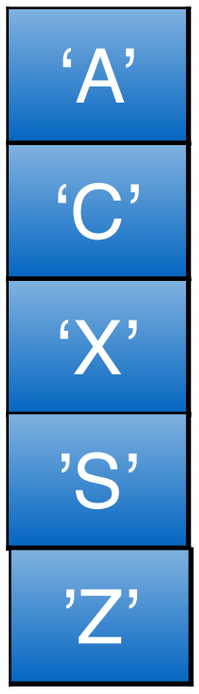
# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(*id*: integer, *category*: string)




$\pi_{attrs}(R)$

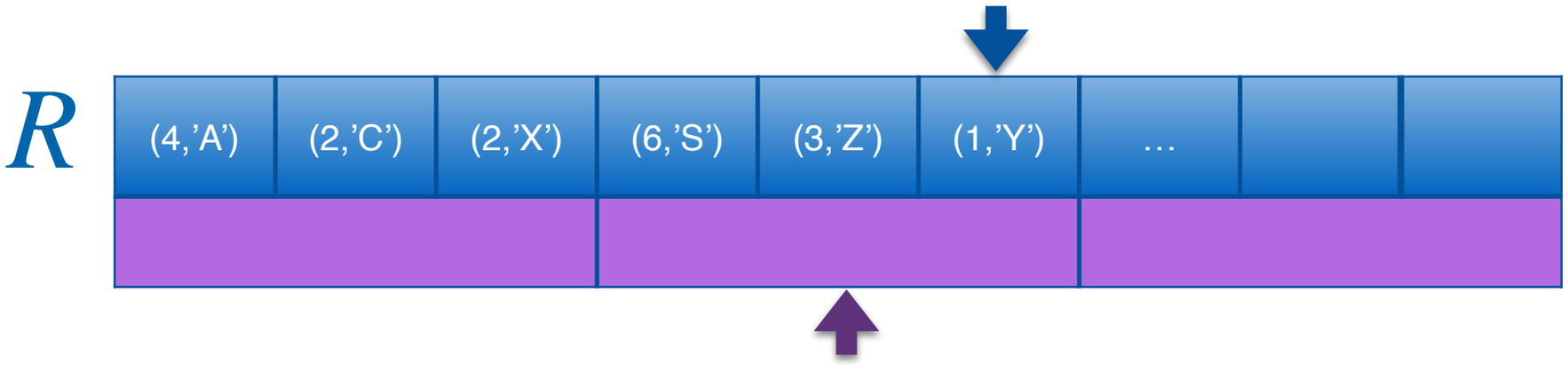



$attrs \triangleq category$

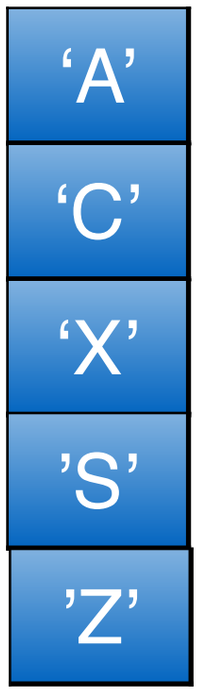
# Proyección

$\pi_{attrs}(R)$
- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

$R$(*id*: integer, *category*: string)




$\pi_{attrs}(R)$




$attrs \triangleq category$

# Selección

$\pi_{attrs}(R)$

- Para cada tupla $r \in R$
  - escribir la proyección $attr$ de $\{r\}$

**¿Costo?**

$$\lceil \frac{|R|}{B} \rceil$$

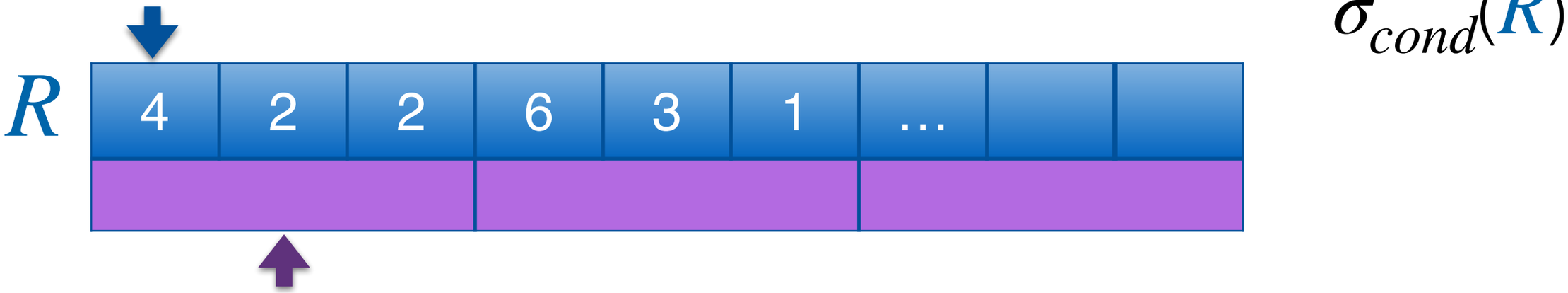
**¿Memoria?**

$B$ tuplas

# Selección

$\sigma_{cond}(R)$

- Para cada tupla $r \in R$
  - Si $r$ satisface $cond$=> escribir $\{r\}$

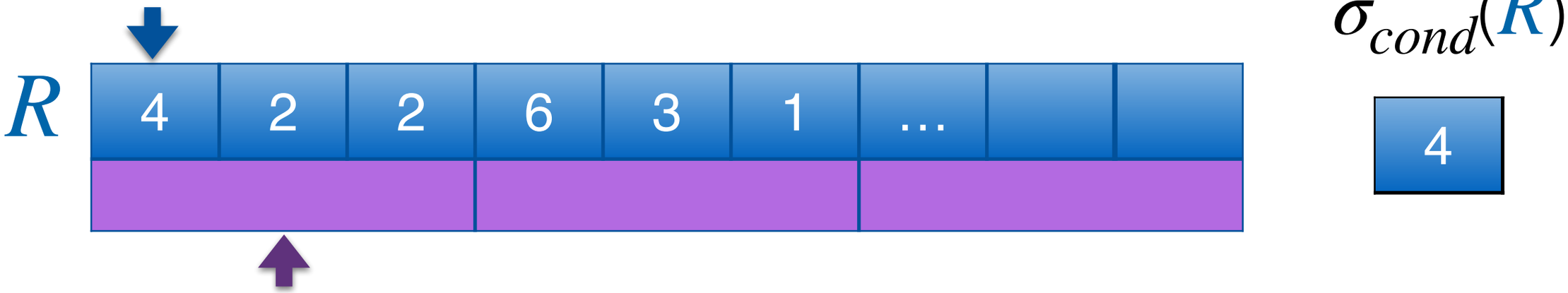



$$cond \triangleq r \geq 4$$

# Selección

$\sigma_{cond}(R)$

- Para cada tupla $r \in R$
  - Si $r$ satisface $cond$=> escribir $\{r\}$

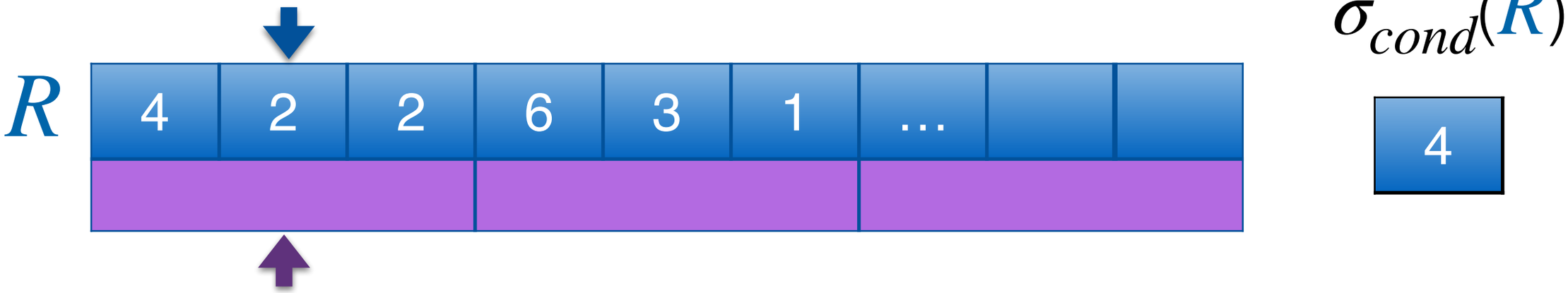



$$cond \triangleq r \geq 4$$

# Selección

$\sigma_{cond}(R)$

- Para cada tupla $r \in R$
  - Si $r$ satisface $cond$=> escribir $\{r\}$

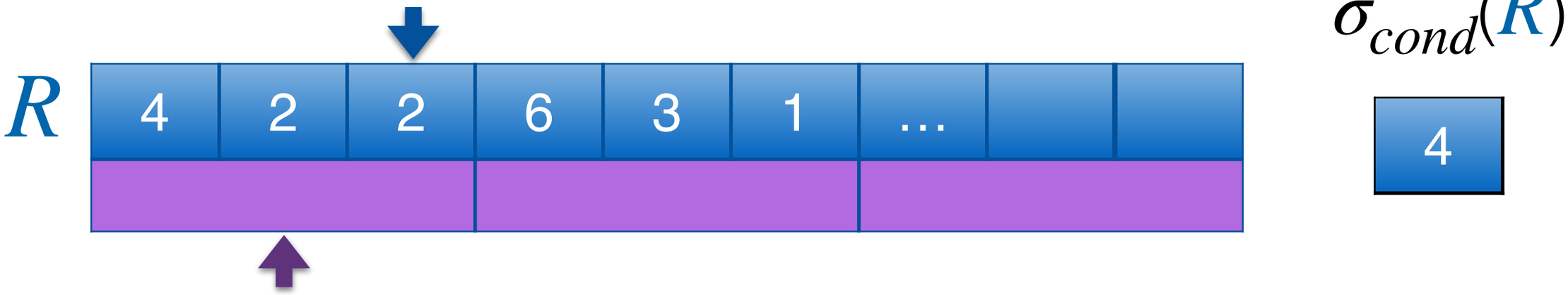



$$cond \triangleq r \geq 4$$

# Selección

$\sigma_{cond}(R)$

- Para cada tupla $r \in R$
  - Si $r$ satisface $cond$=> escribir $\{r\}$

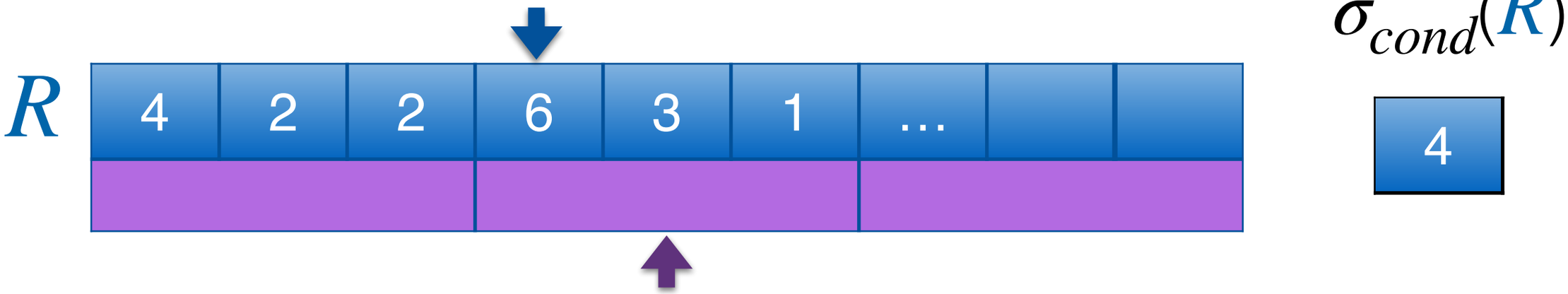



$$cond \triangleq r \geq 4$$

# Selección

$\sigma_{cond}(R)$

- Para cada tupla $r \in R$
  - Si $r$ satisface $cond$=> escribir $\{r\}$

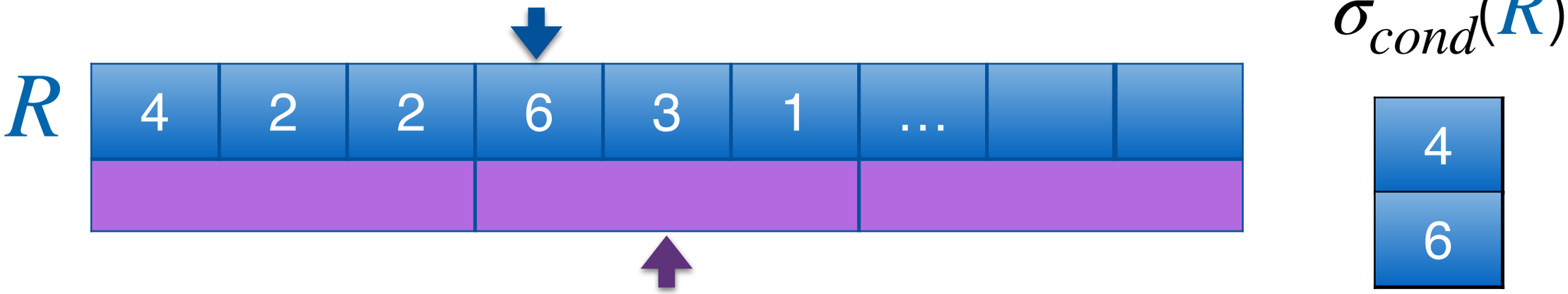



$$cond \triangleq r \geq 4$$

# Selección

$\sigma_{cond}(R)$

- Para cada tupla $r \in R$
  - Si $r$ satisface $cond$=> escribir $\{r\}$

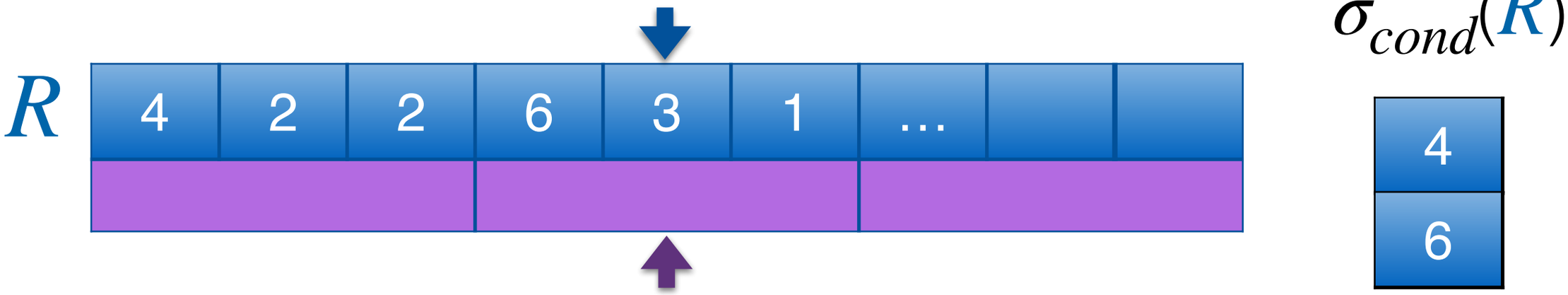



$$cond \triangleq r \geq 4$$

# Selección

$\sigma_{cond}(R)$

- Para cada tupla $r \in R$
  - Si $r$ satisface $cond$=> escribir $\{r\}$

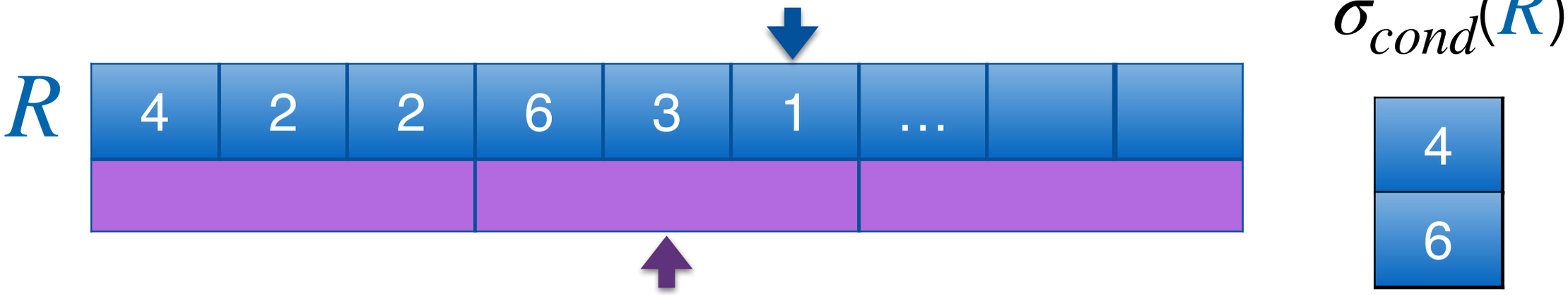



$$cond \triangleq r \geq 4$$

# Selección

$\sigma_{cond}(R)$

- Para cada tupla $r \in R$
  - Si $r$ satisface $cond$=> escribir $\{r\}$




$$cond \triangleq r \geq 4$$

# Selección

$\sigma_{cond}(R)$

- Para cada tupla $r \in R$
  - Si $r$ satisface $cond$=> escribir $\{r\}$

**¿Costo?**

$\lceil \frac{|R|}{B} \rceil$

**¿Memoria?**

$B$ tuplas

**Usando índices**

peor caso

$\lceil \frac{|R|}{B} \rceil$

Árbol B+ (mem s.)

$O(log_b(\lceil \frac{|R|}{B} \rceil))$

Hash/Árbol B+ (mem p.)

$O(1)$

# Costos del join

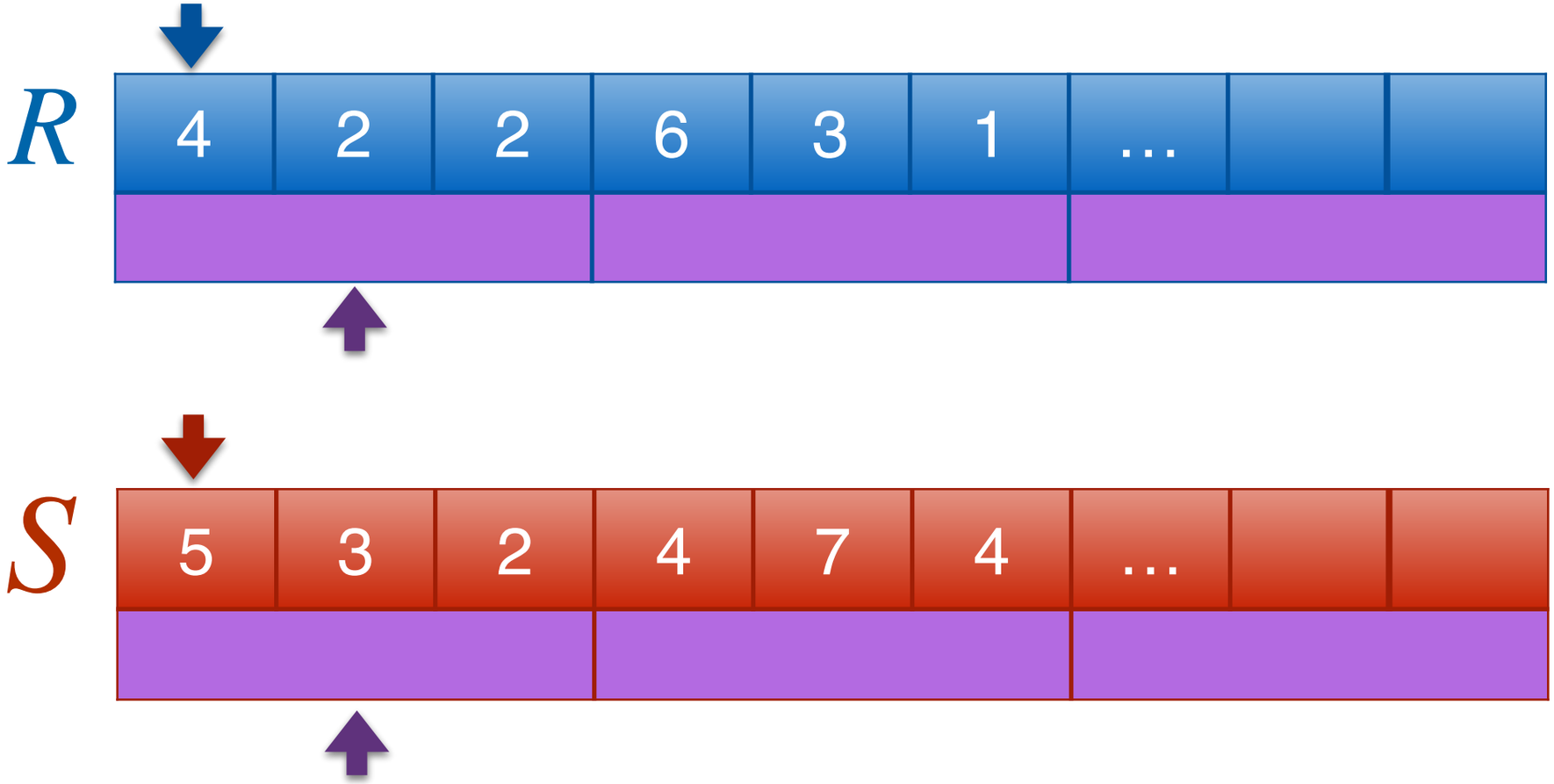
-Loop anidado (Nested Loop Join)
-Index Nested Loop Join
-Hash Join
-Sort-Merge Join

# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$



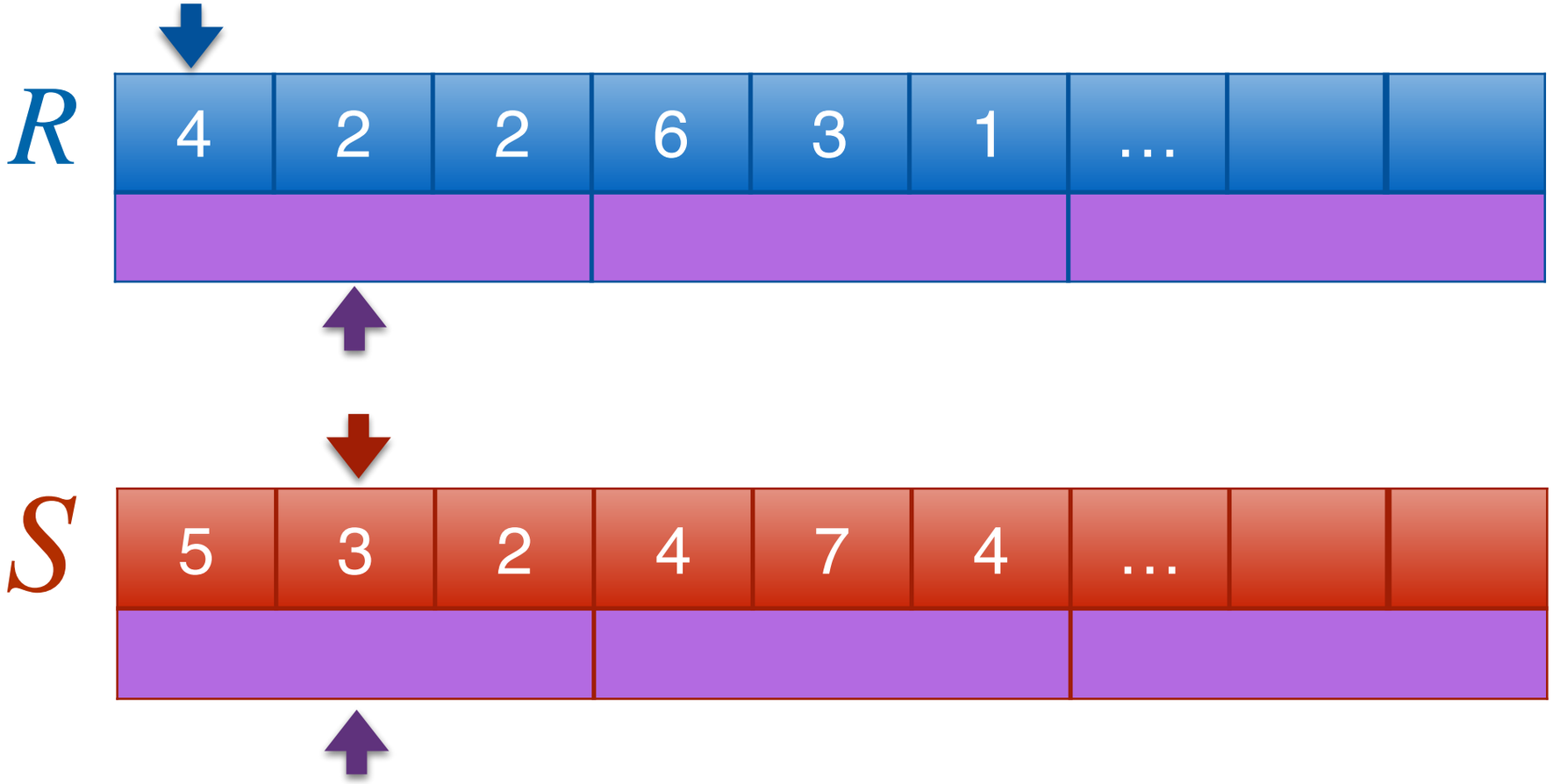

$R \bowtie S$

# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$



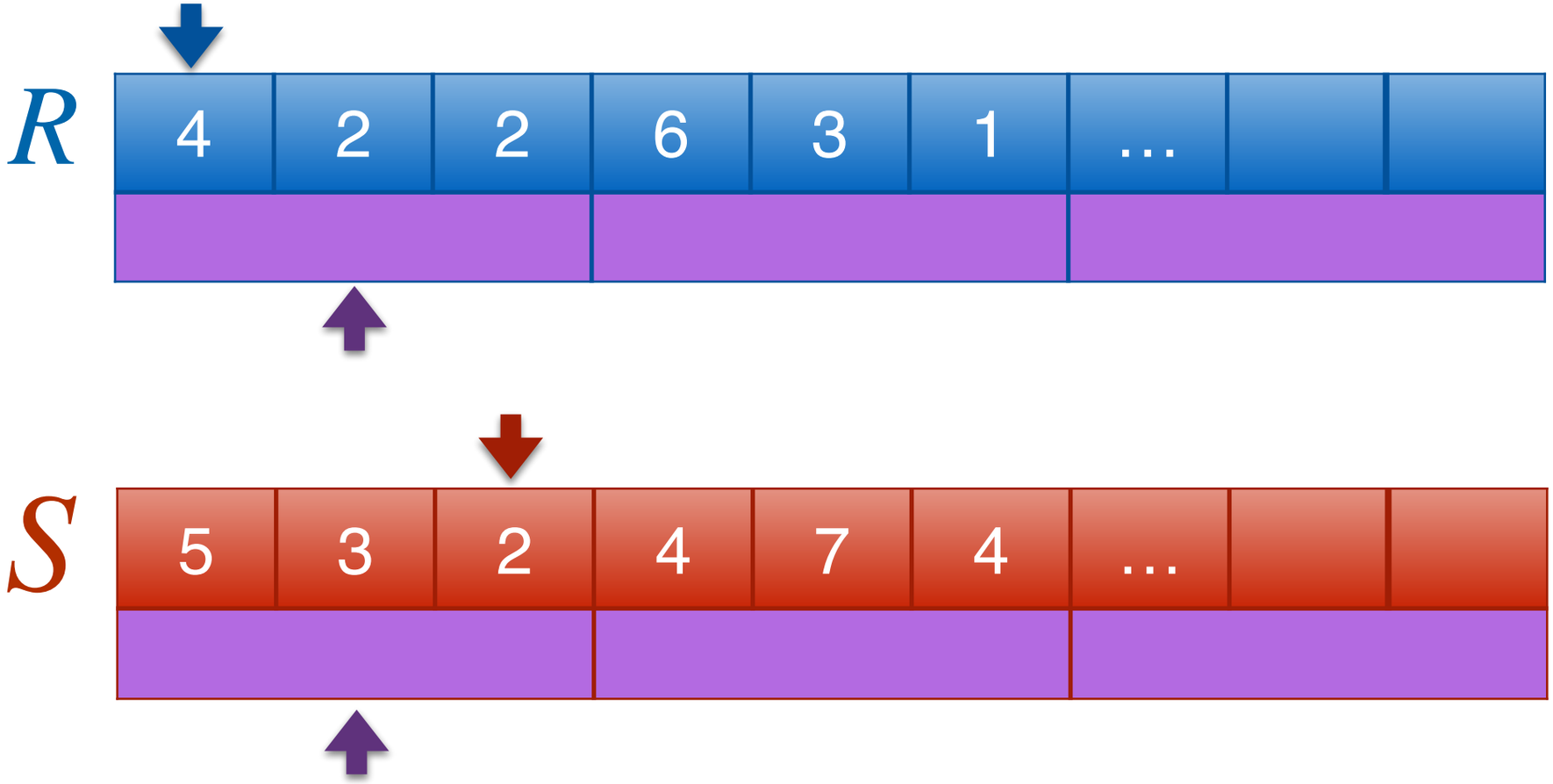

$R \bowtie S$

# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$



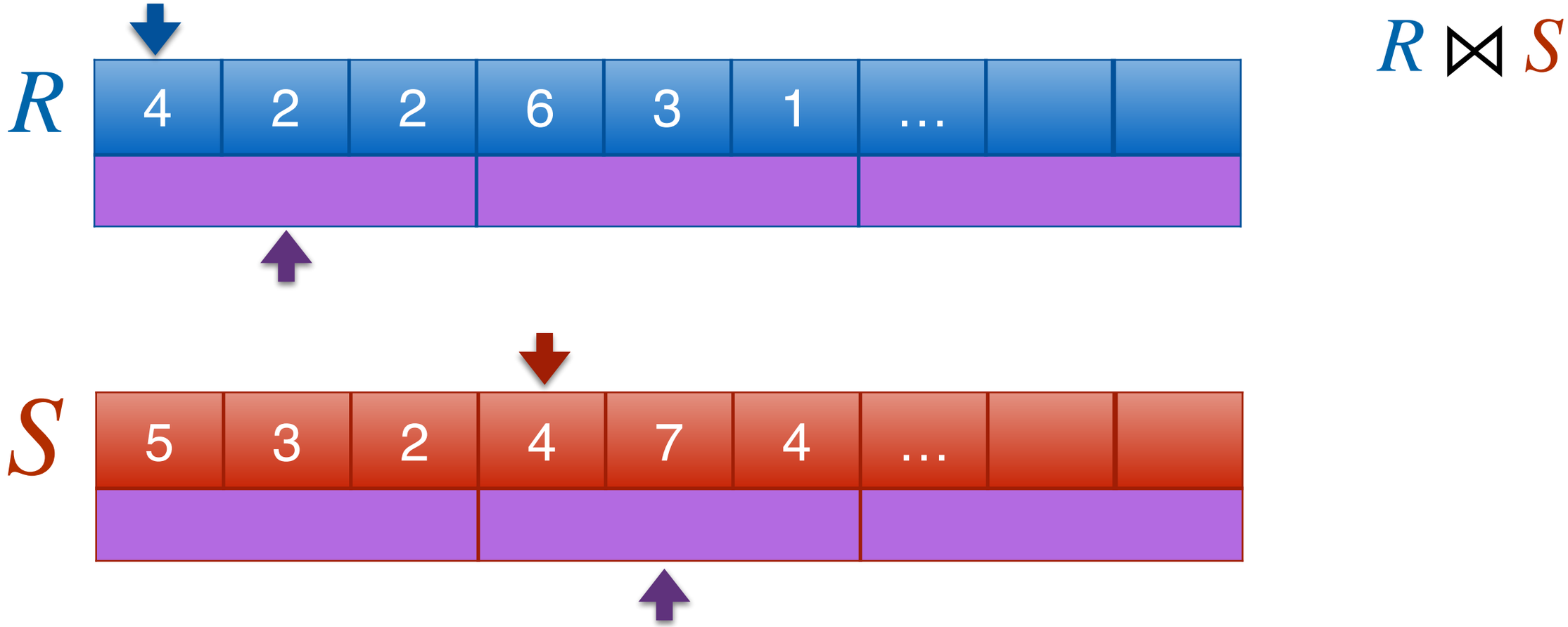

$R \bowtie S$

# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$

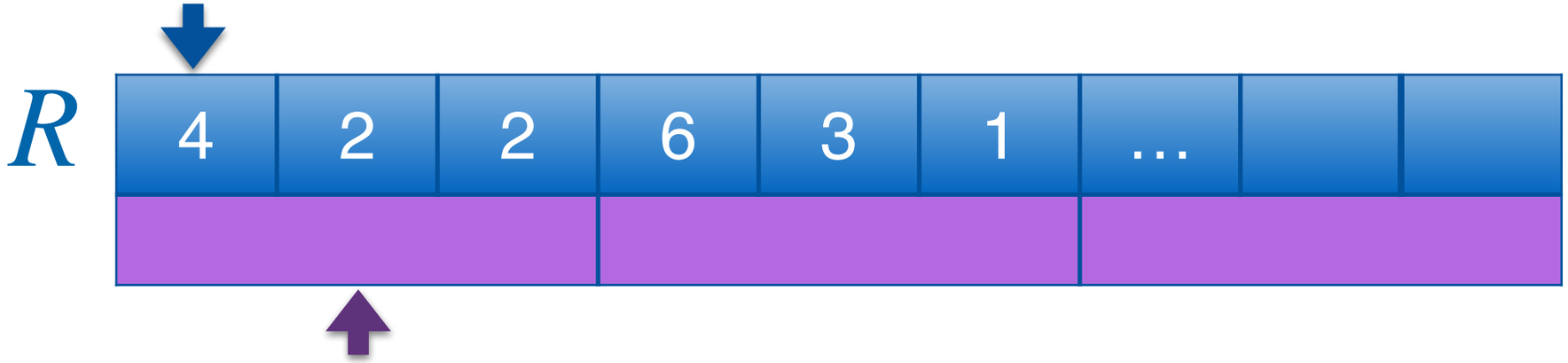


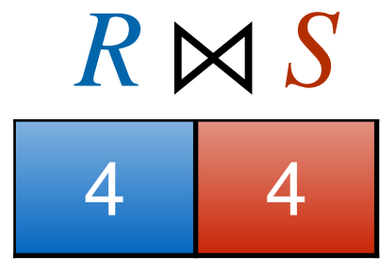
# Loop anidado (sin índices)

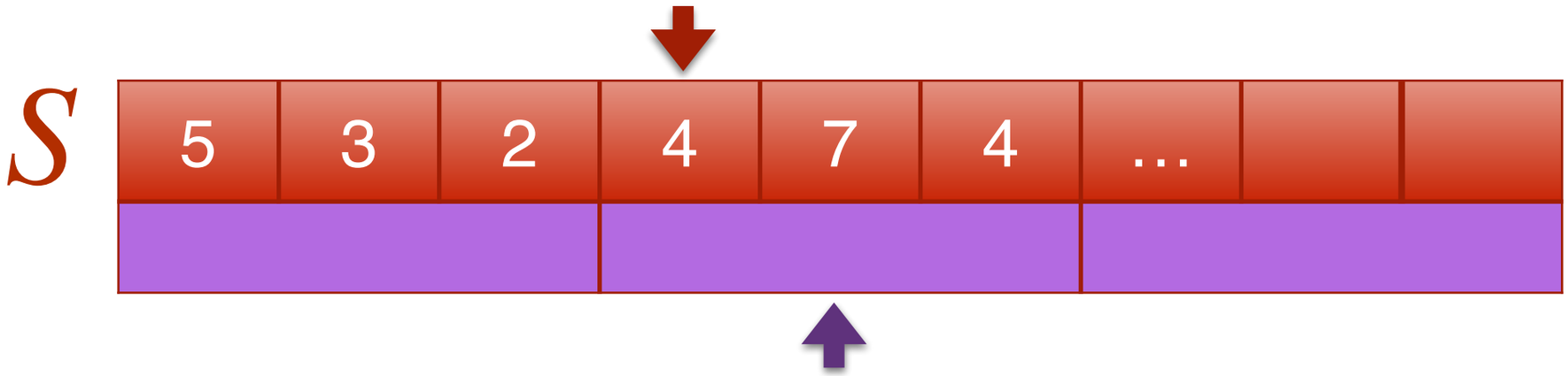
$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$



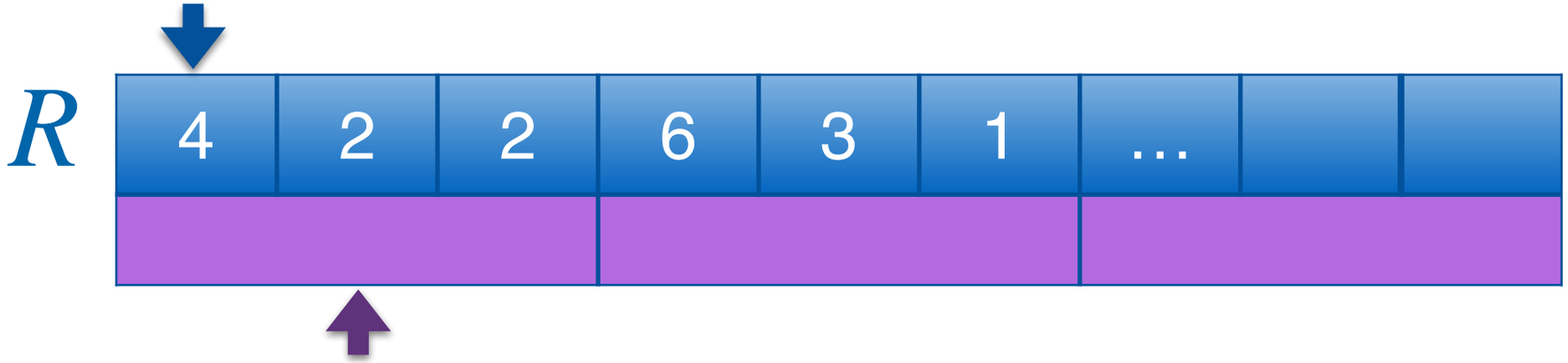



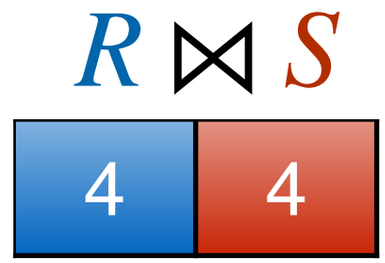
# Loop anidado (sin índices)

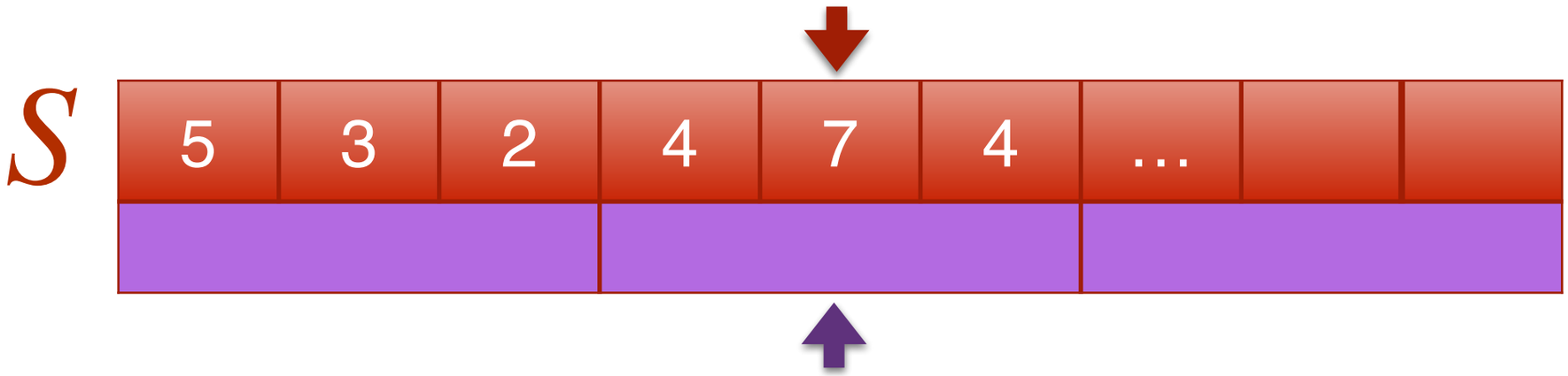
$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$



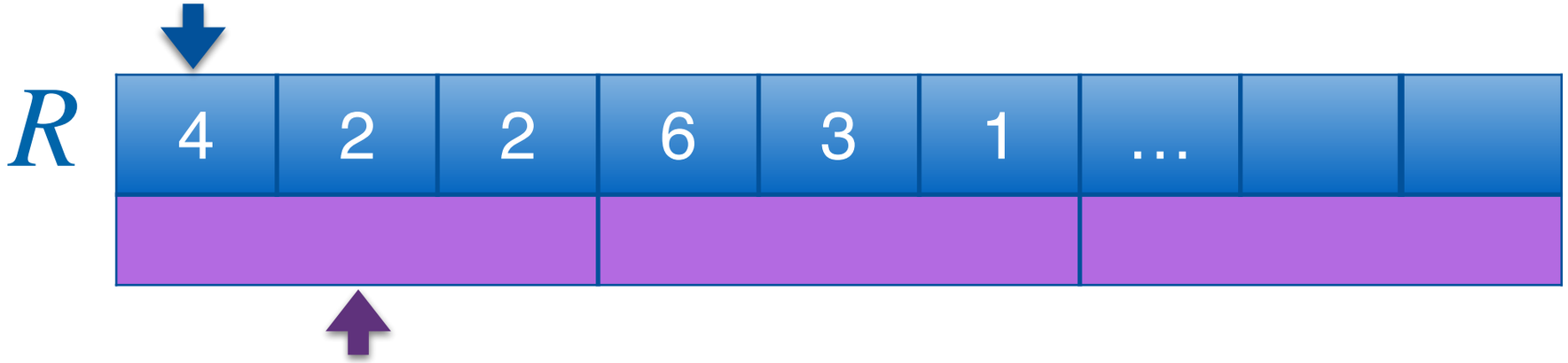



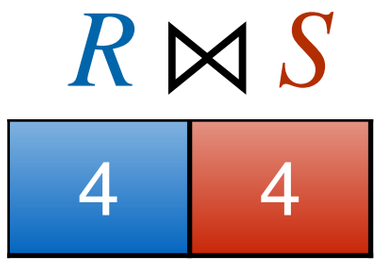
# Loop anidado (sin índices)

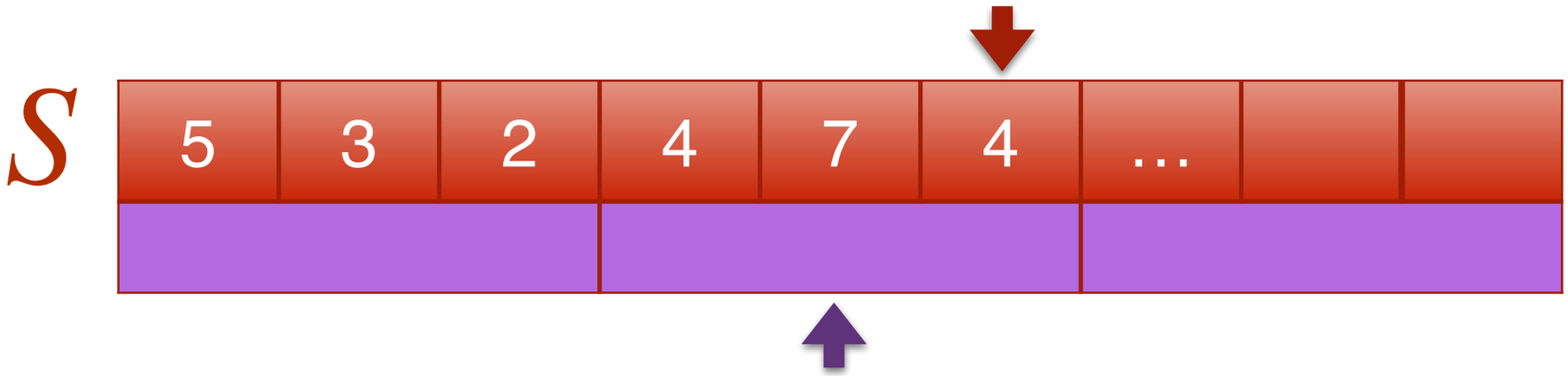
$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$



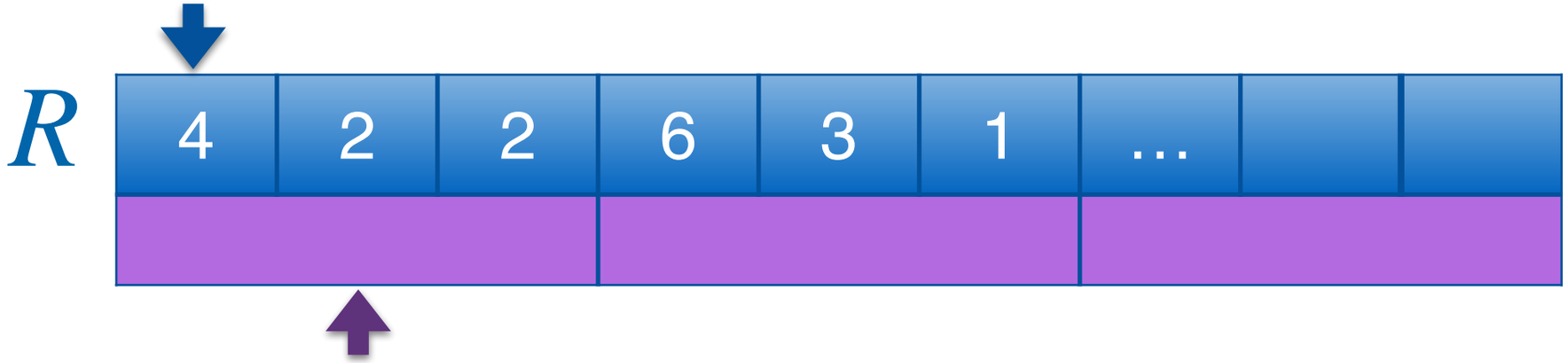



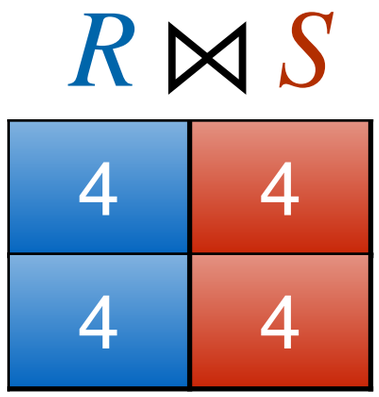
# Loop anidado (sin índices)

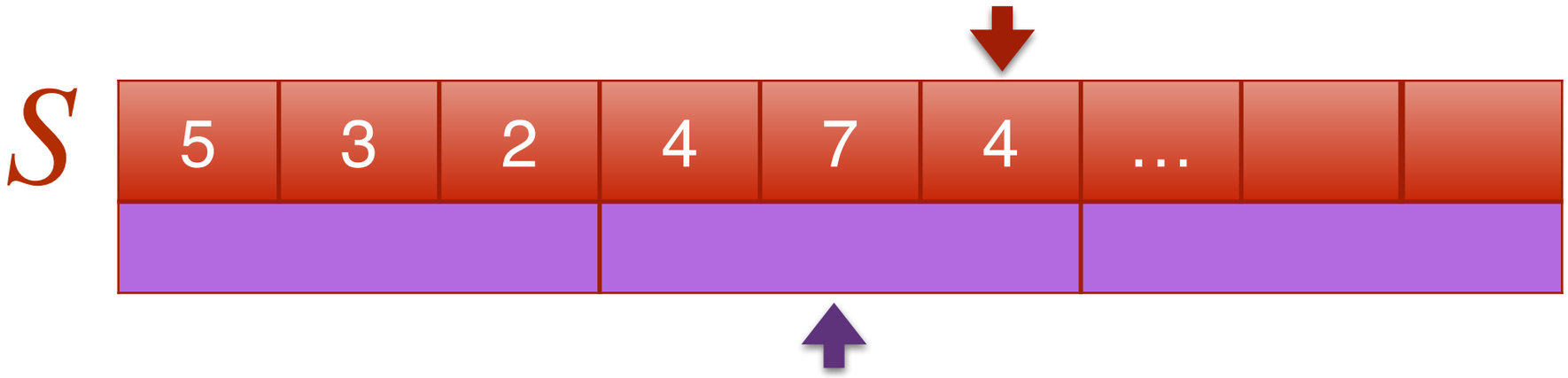
$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$



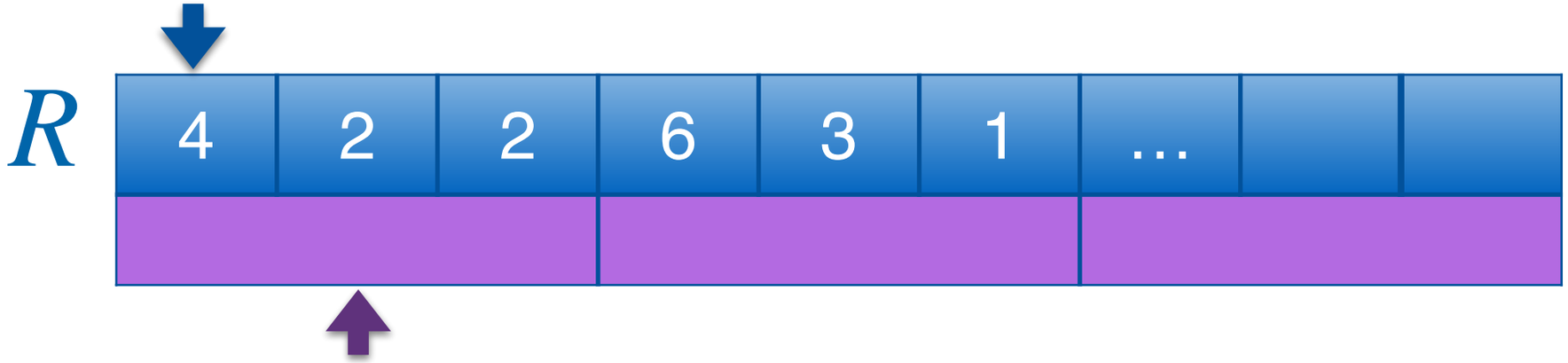



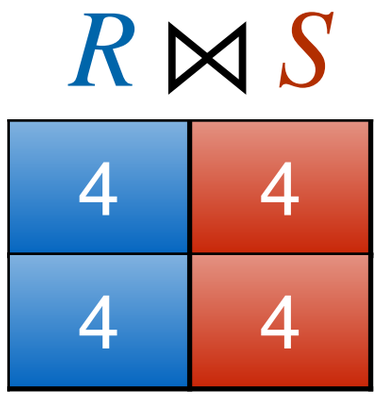
# Loop anidado (sin índices)

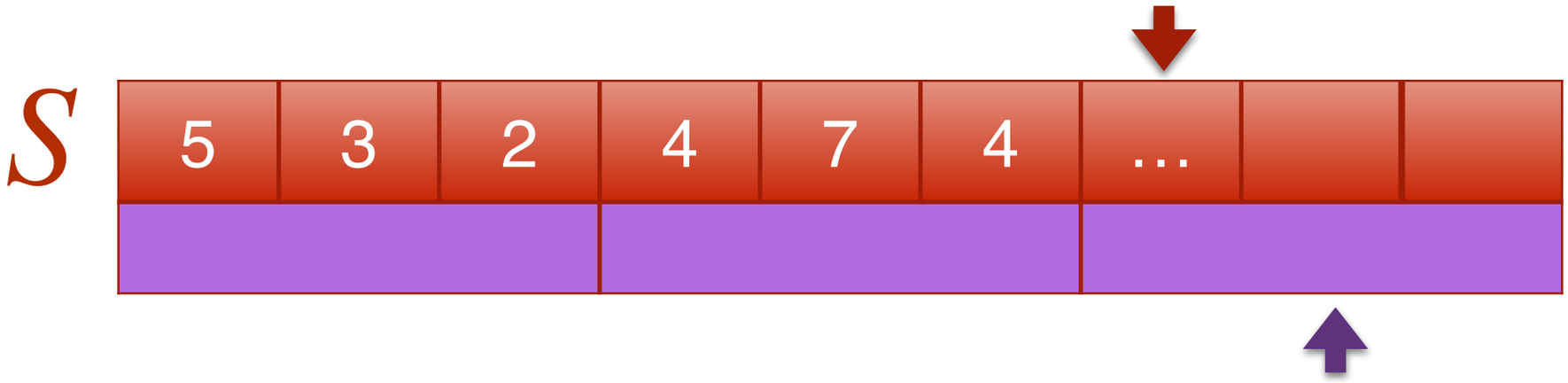
$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$



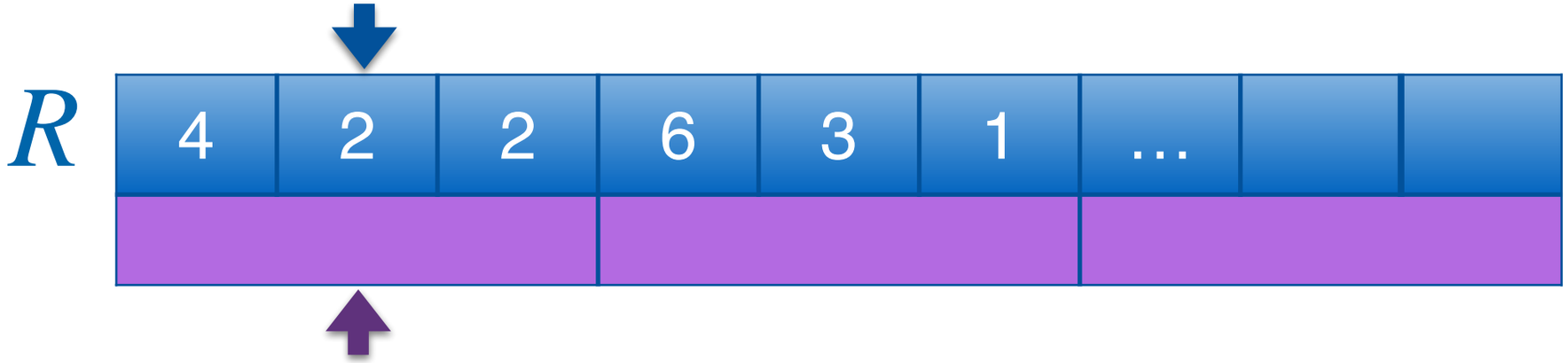



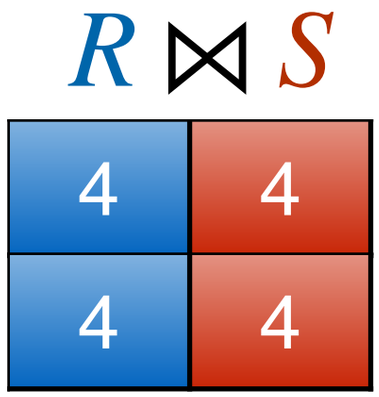
# Loop anidado (sin índices)

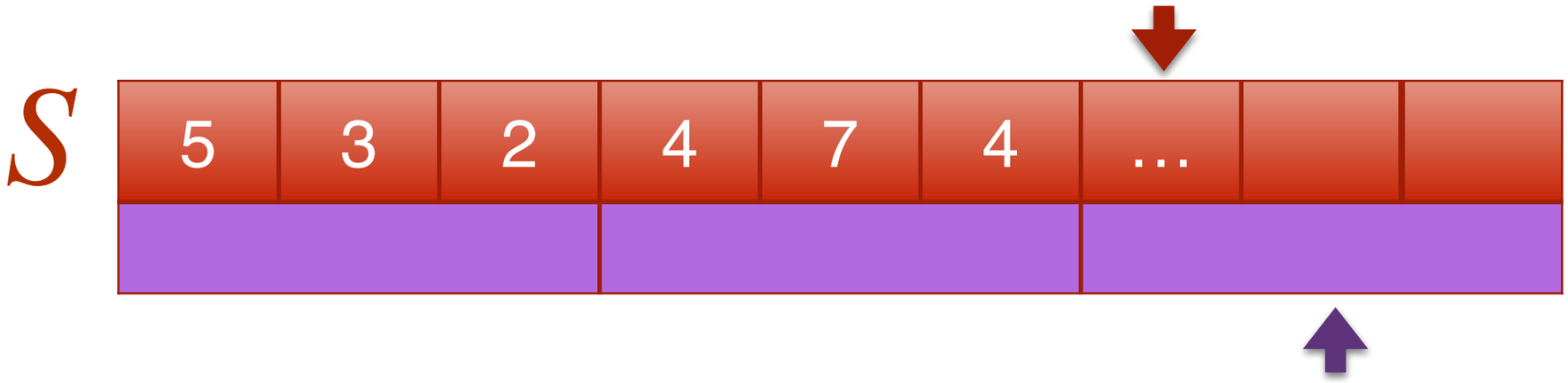
$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$



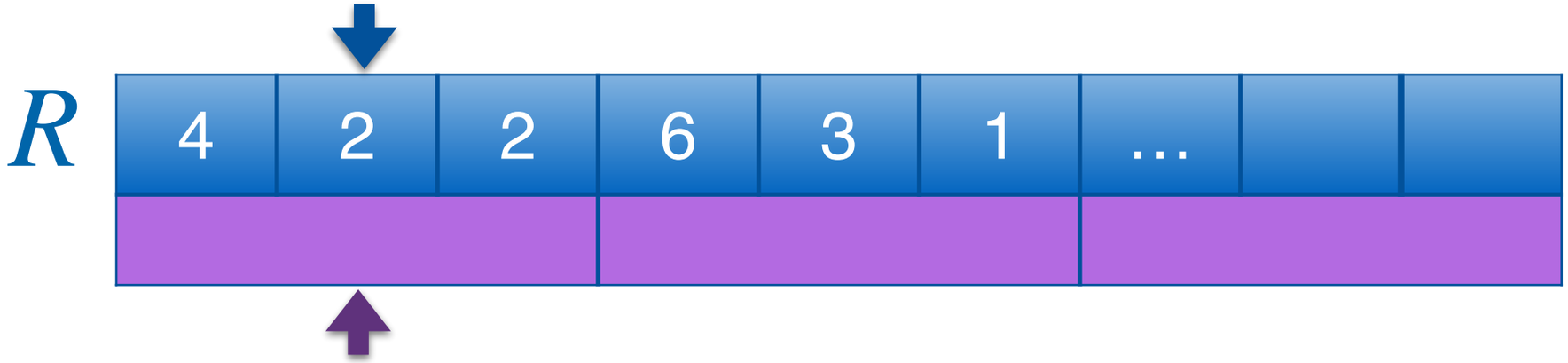



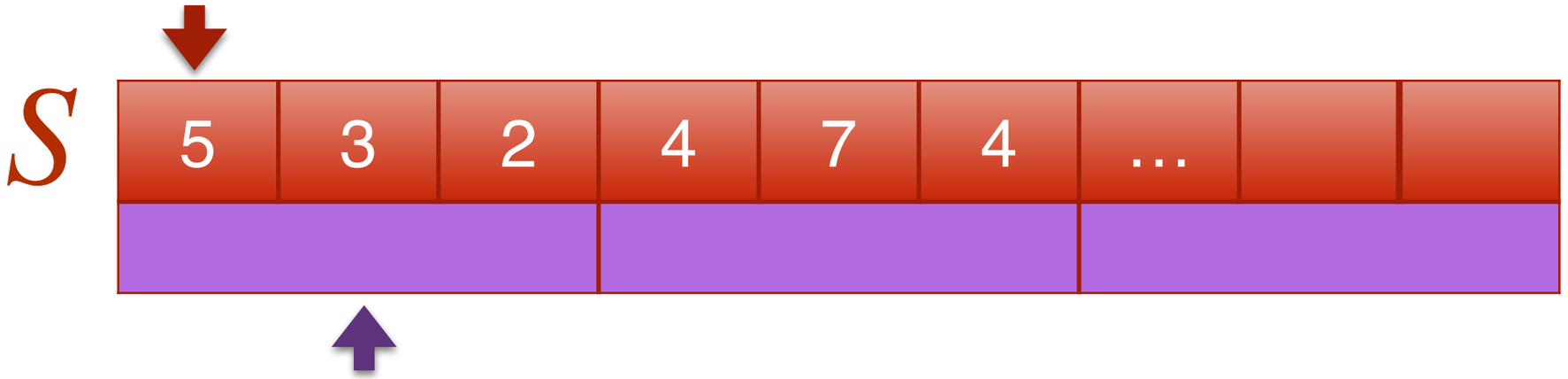
# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$




$R \bowtie S$

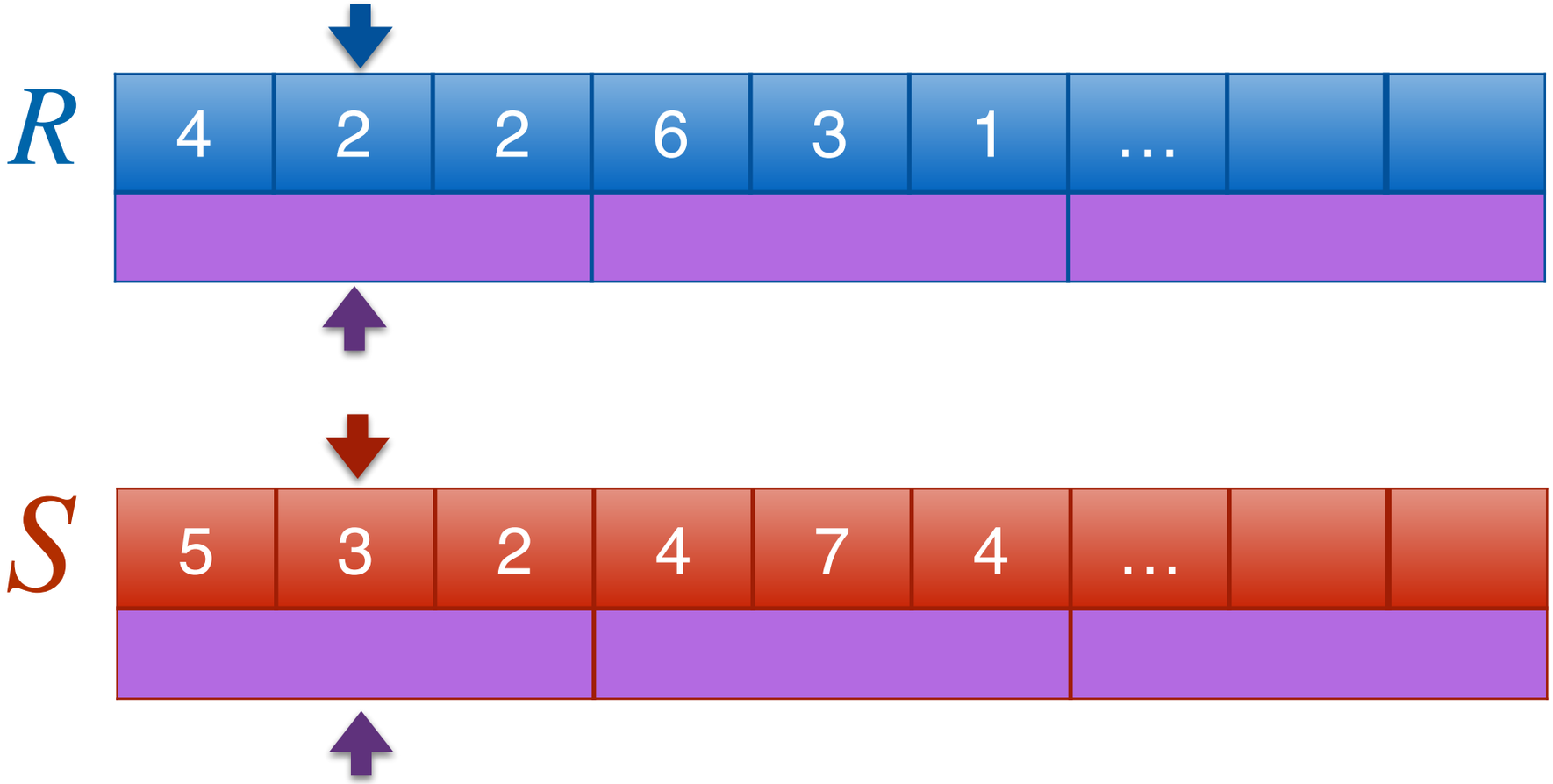
| | |
|---|---|
| 4 | 4 |
| 4 | 4 |

# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$

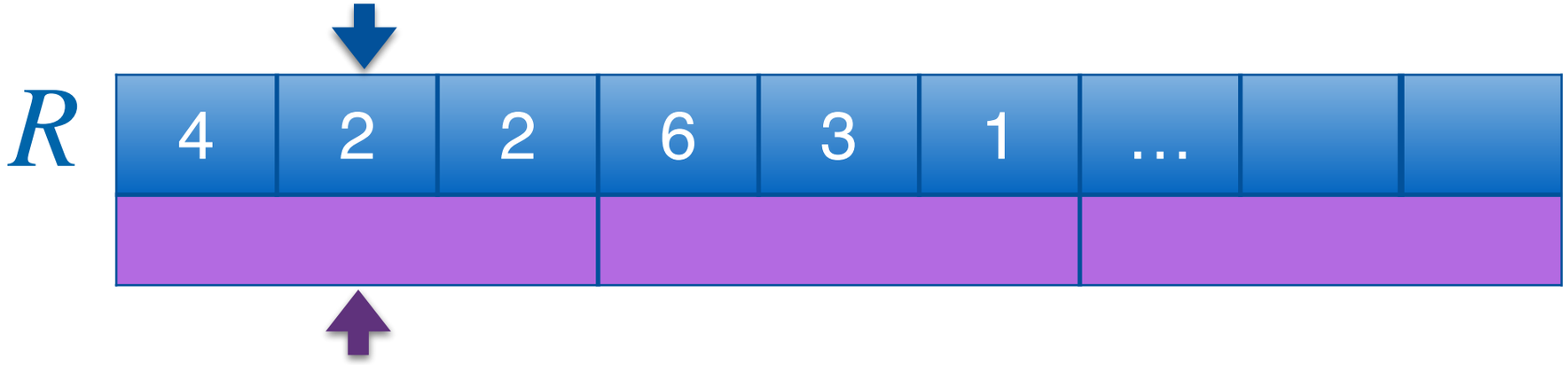


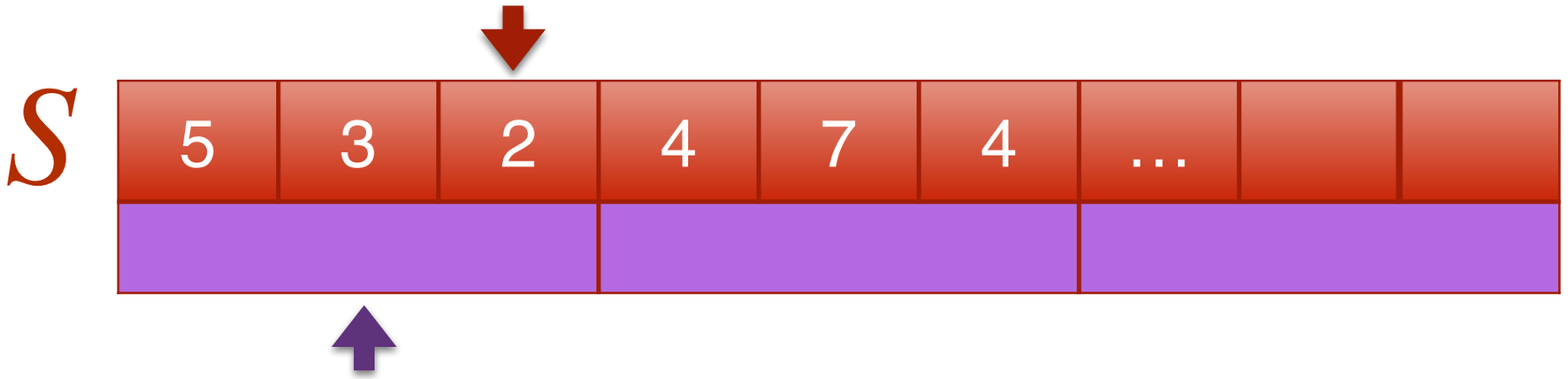
# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$




$R \bowtie S$

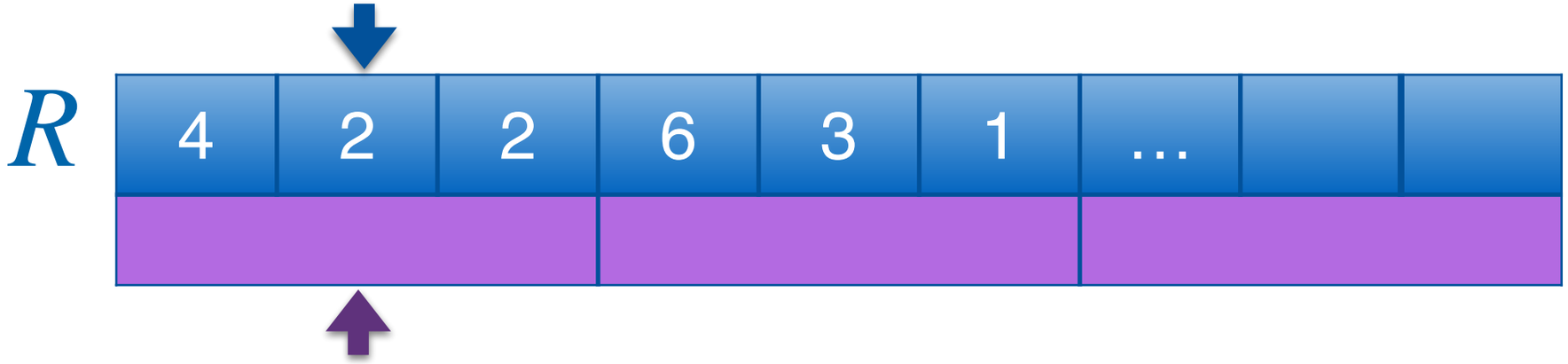
| | |
|---|---|
| 4 | 4 |
| 4 | 4 |

# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$




$R \bowtie S$

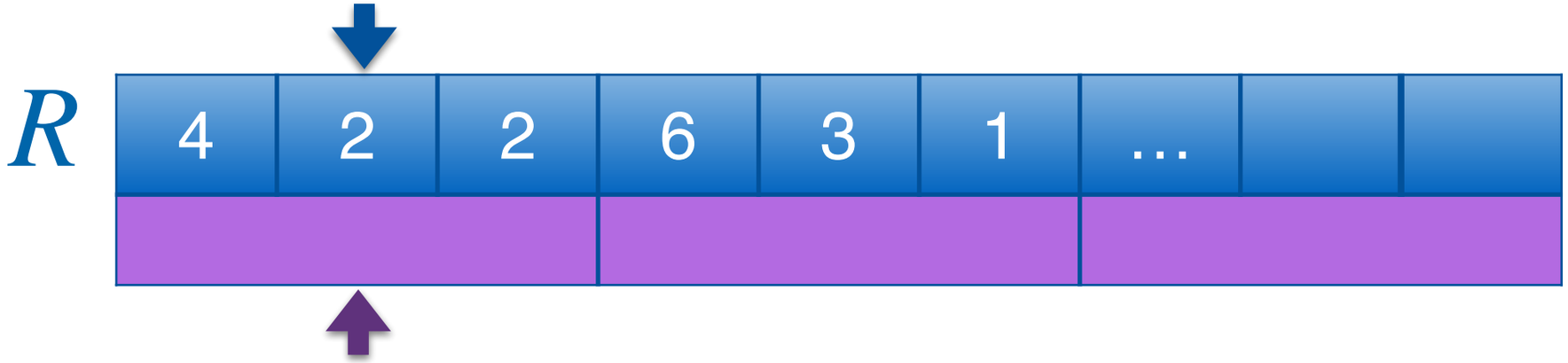
| | |
|---|---|
| 4 | 4 |
| 4 | 4 |
| 2 | 2 |

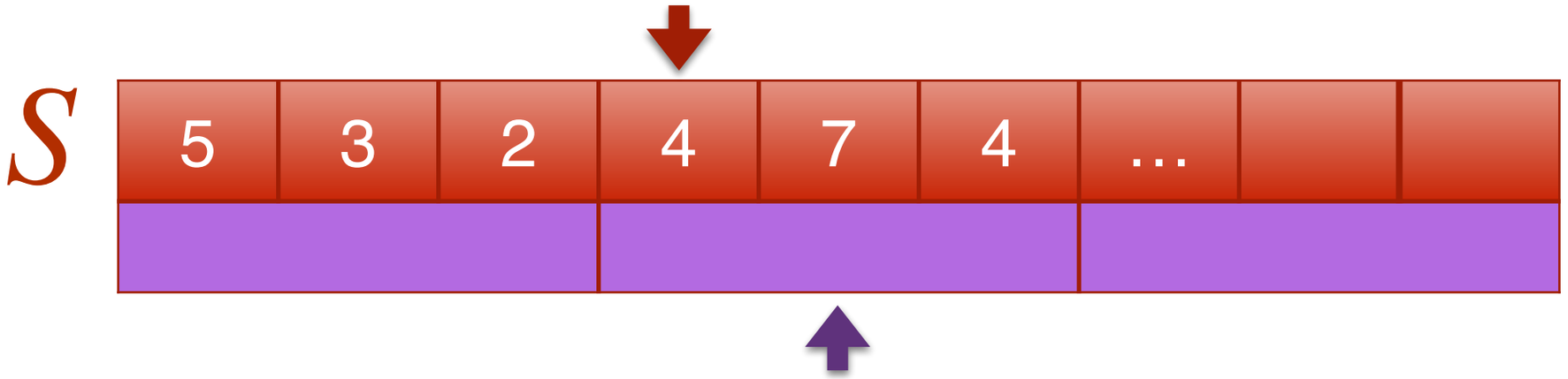
# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$




$R \bowtie S$

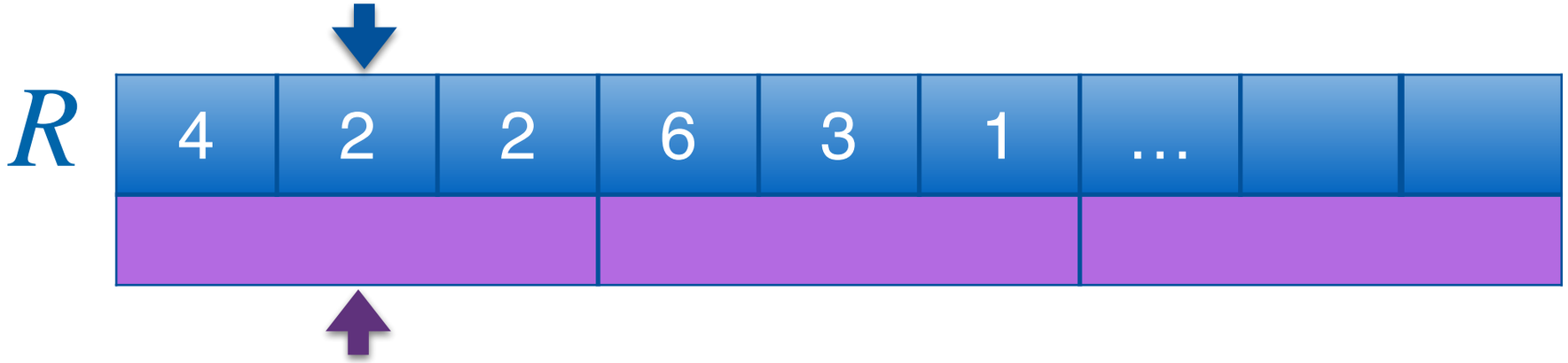
| | |
|---|---|
| 4 | 4 |
| 4 | 4 |
| 2 | 2 |

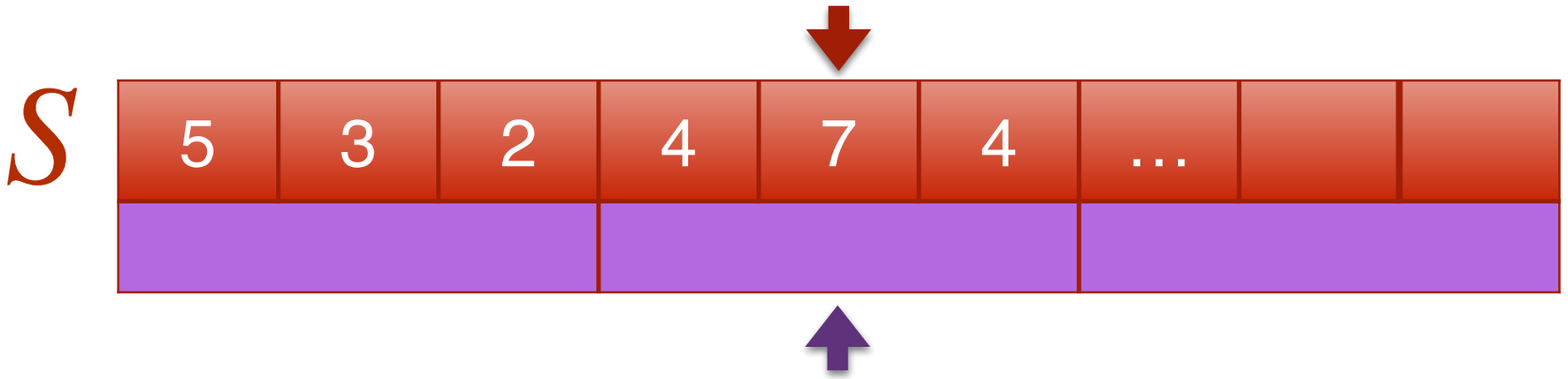
# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$




$R \bowtie S$

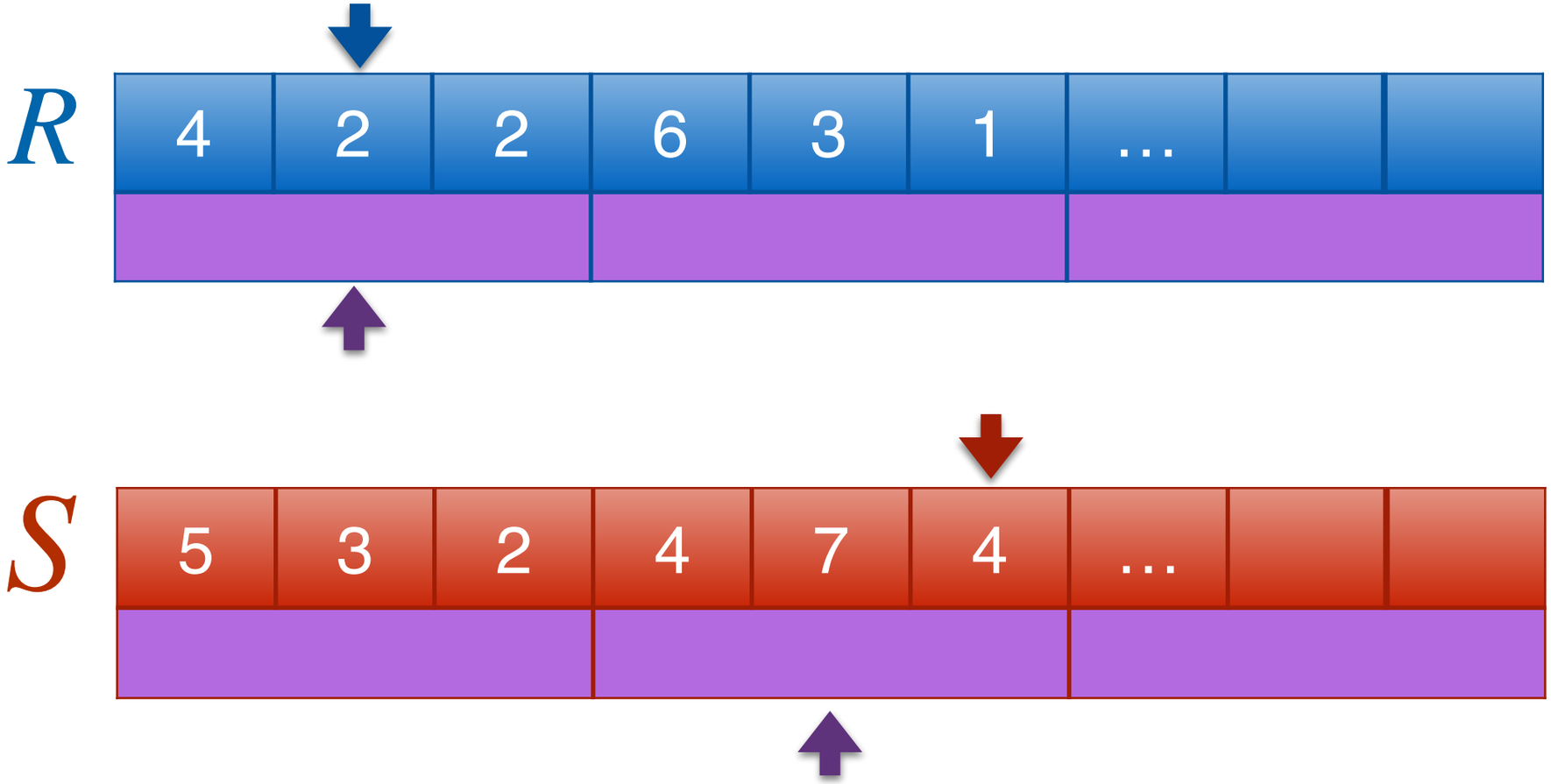
| | |
|---|---|
| 4 | 4 |
| 4 | 4 |
| 2 | 2 |

# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$




$R \bowtie S$

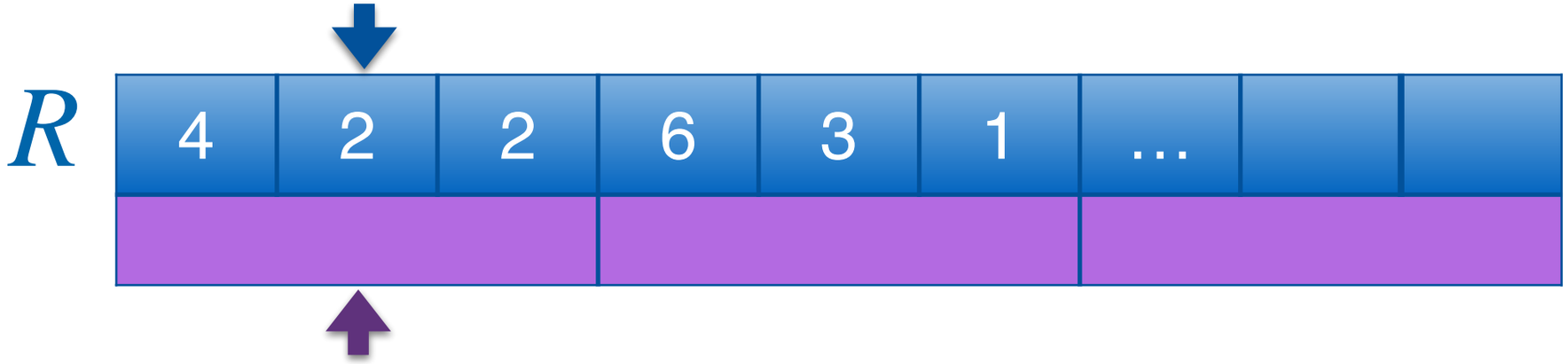
| | |
|---|---|
| 4 | 4 |
| 4 | 4 |
| 2 | 2 |

# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$




$R \bowtie S$

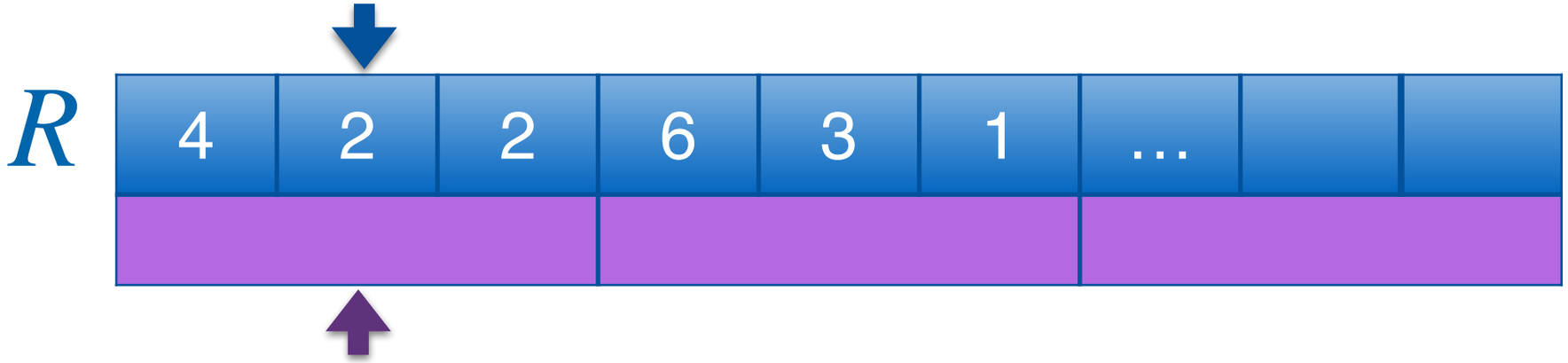
| | |
|---|---|
| 4 | 4 |
| 4 | 4 |
| 2 | 2 |

# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$




$R \bowtie S$

| | |
|---|---|
| 4 | 4 |
| 4 | 4 |
| 2 | 2 |

...

# Loop anidado (sin índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Para cada tupla $s \in S$
    - Si $r$ y $s$ satisfacen el join =>

escribir $\{r\} \times \{s\}$

**¿Costo?**

$$\left\lceil \frac{|R|}{B} \right\rceil + |R| \cdot \left\lceil \frac{|S|}{B} \right\rceil$$

**¿Memoria?**

$2B$ tuplas

**¿Elegir $R$ y $S$?**

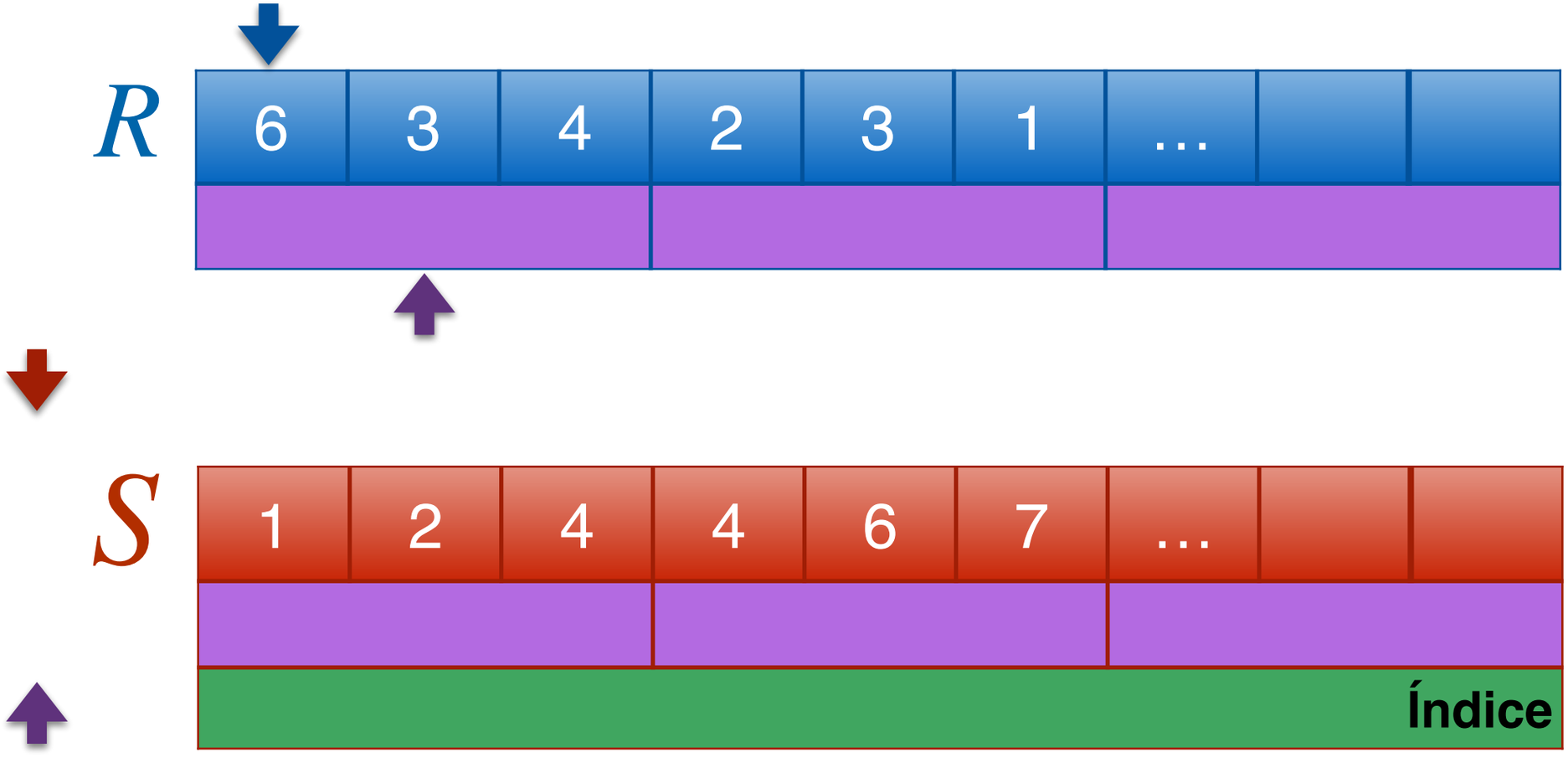
$|R| < |S|$

(Ahorro de tiempo)

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

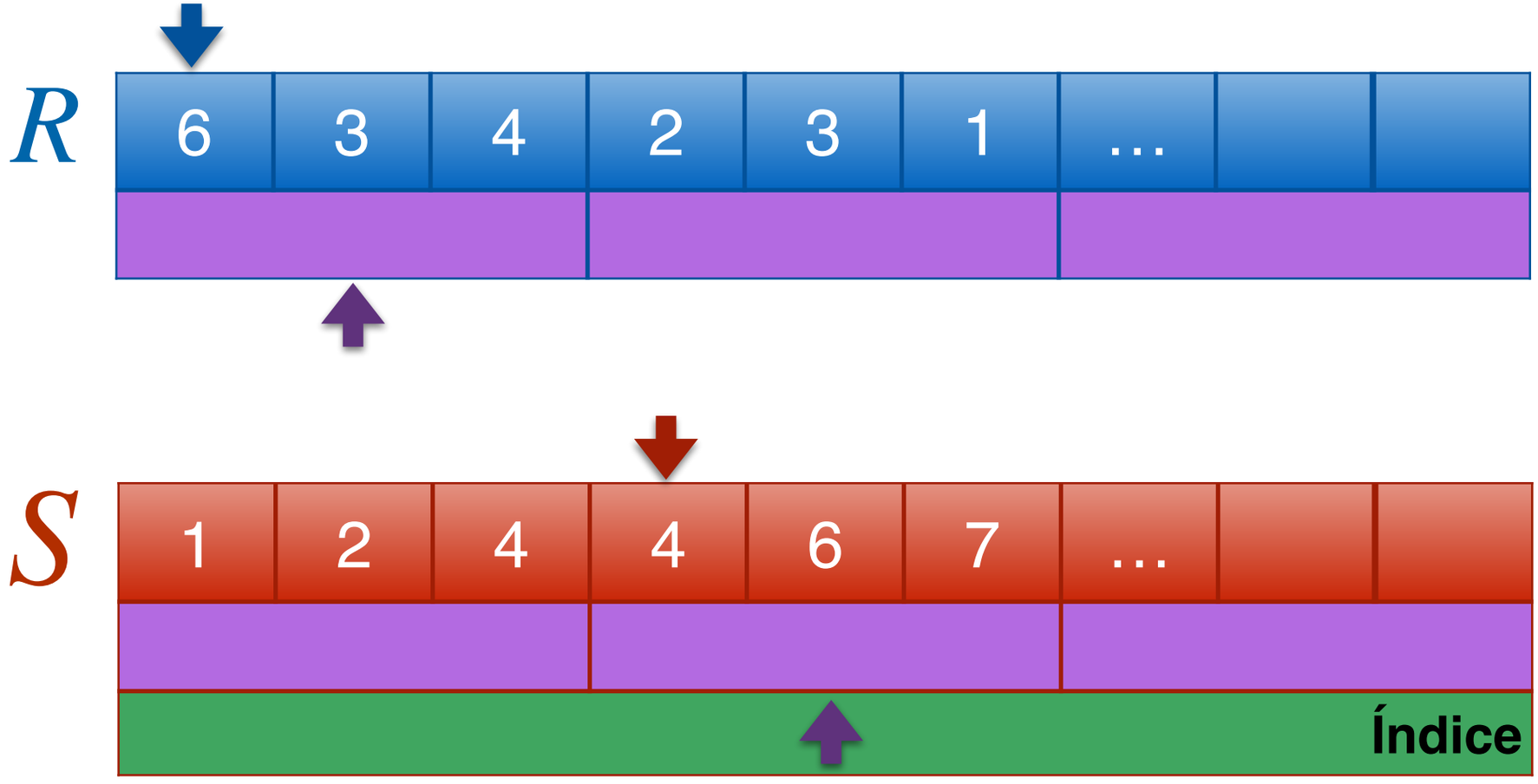



$R \bowtie S$

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

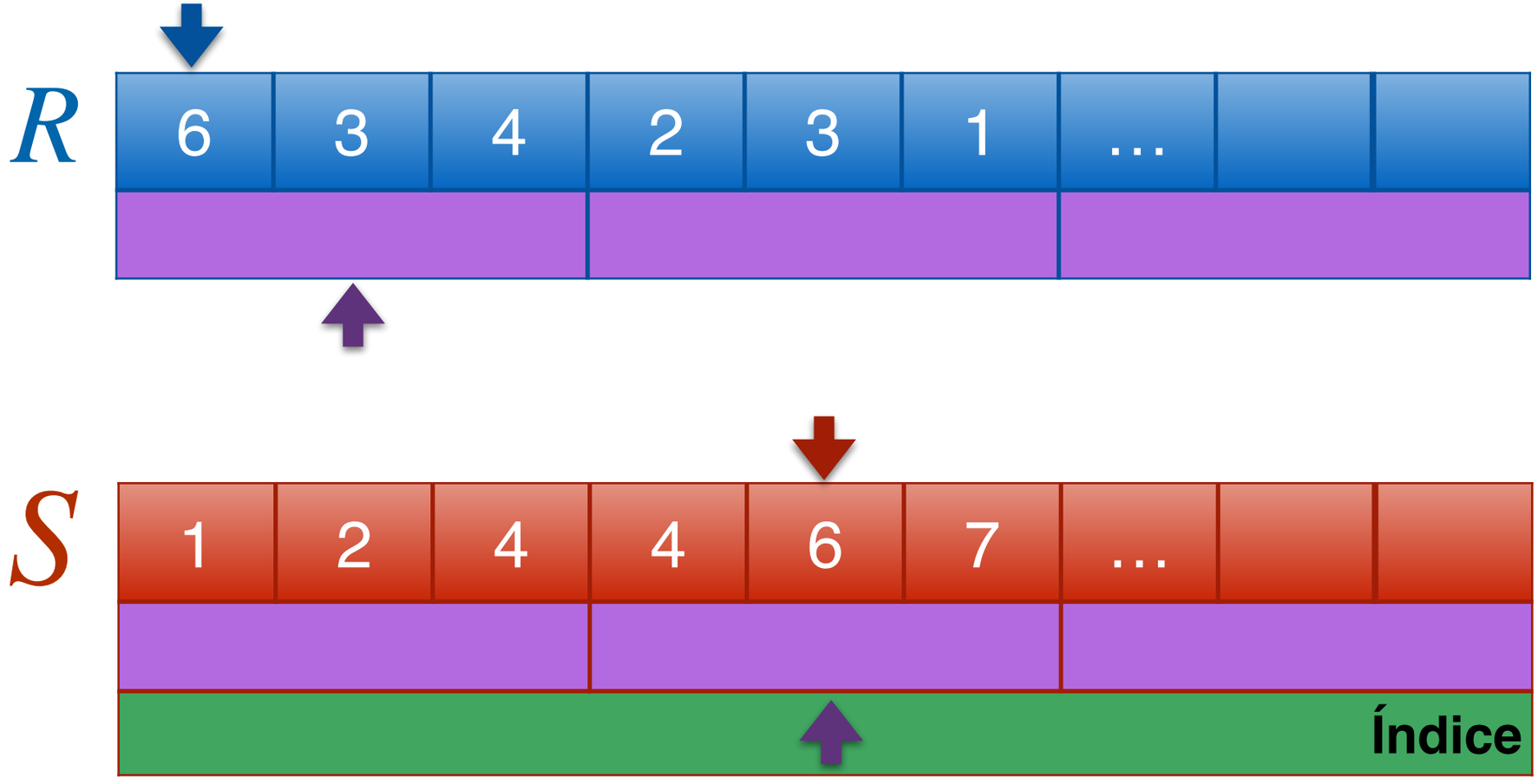



$R \bowtie S$

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

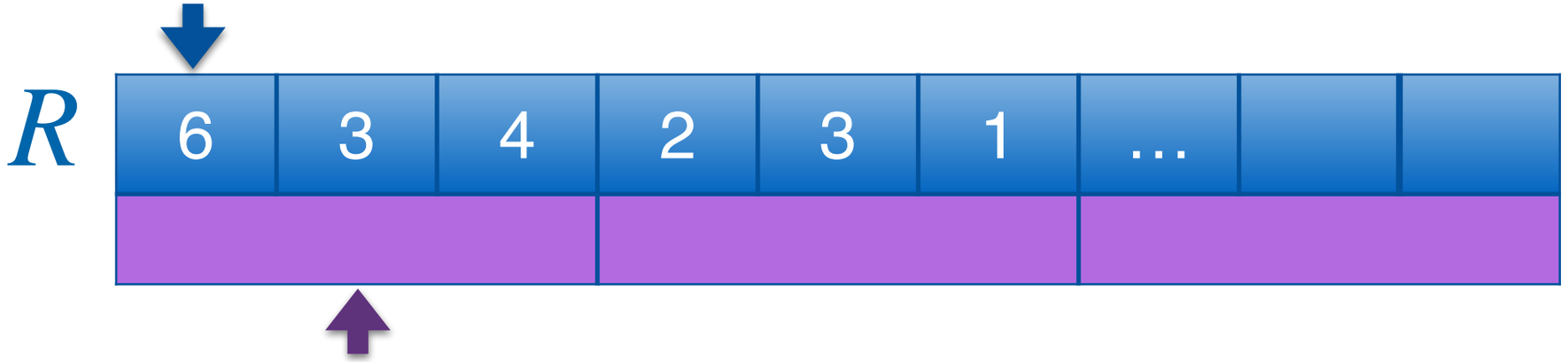


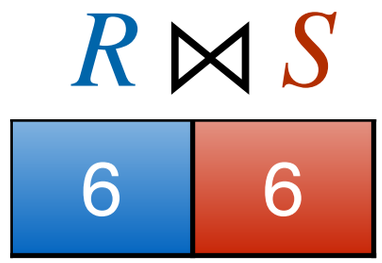

$R \bowtie S$

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

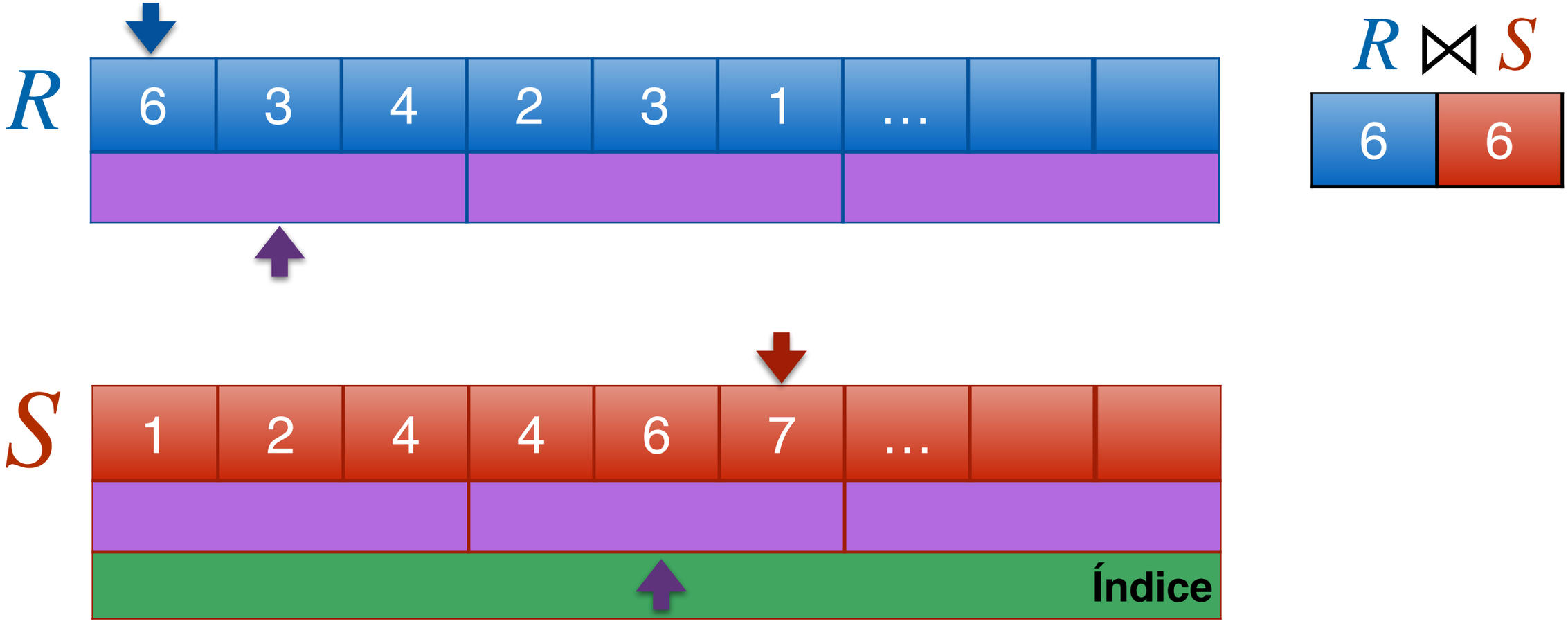


# Loop anidado (con índices)

$R \bowtie S$

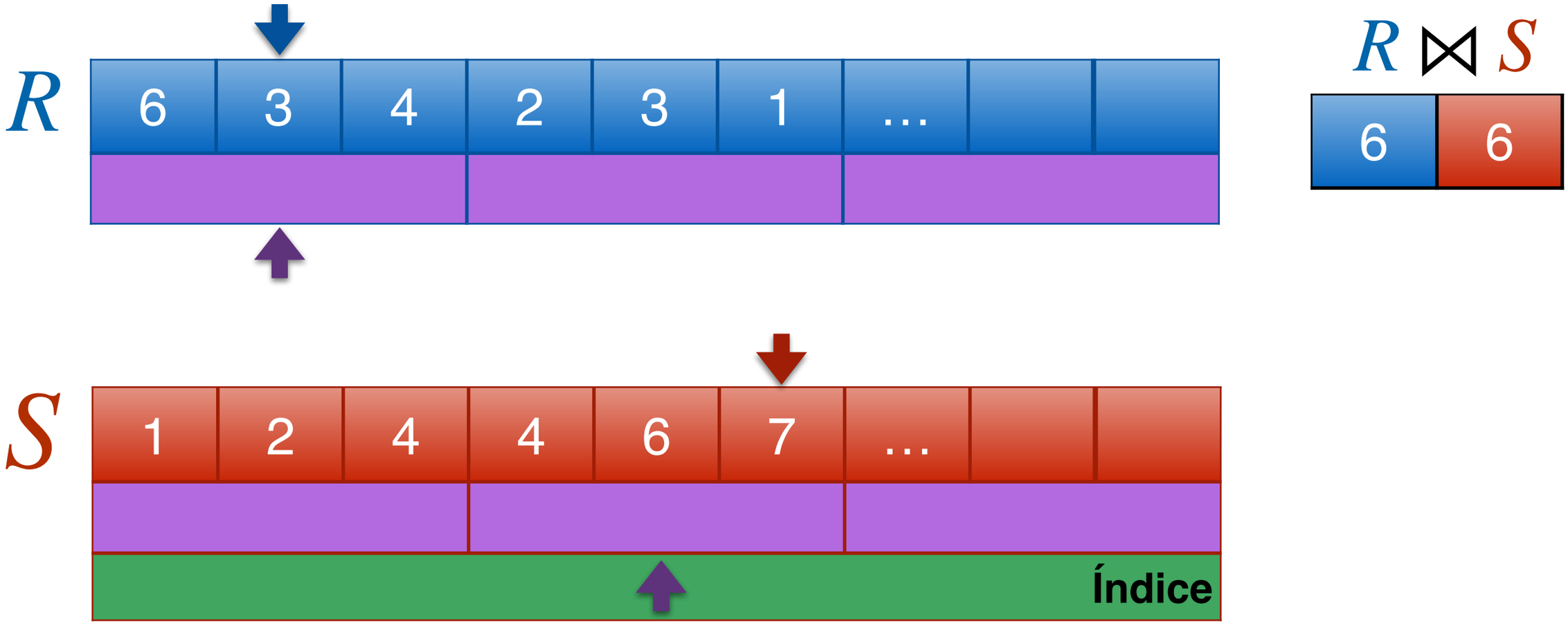
- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



# Loop anidado (con índices)

$R \bowtie S$

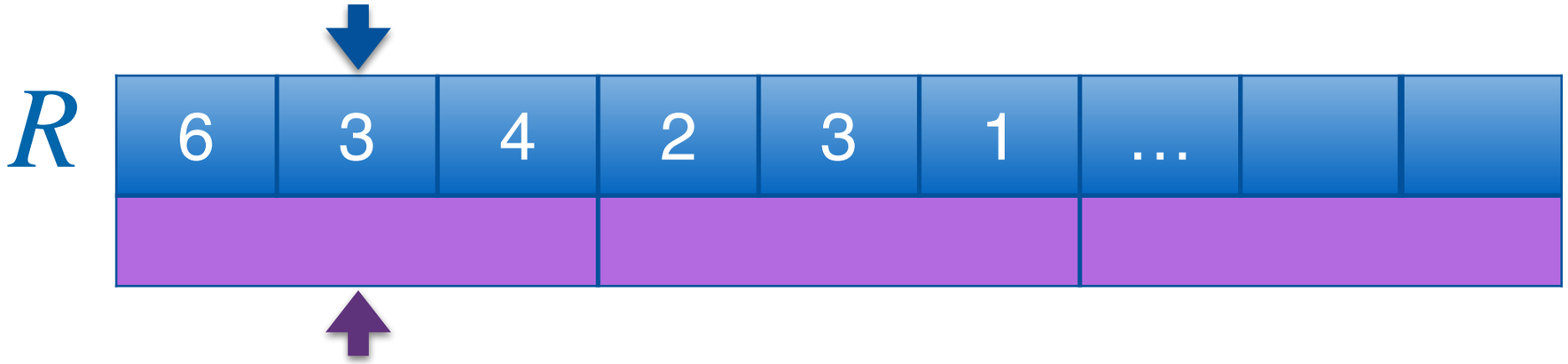
- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

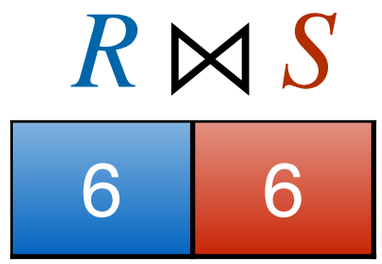


# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

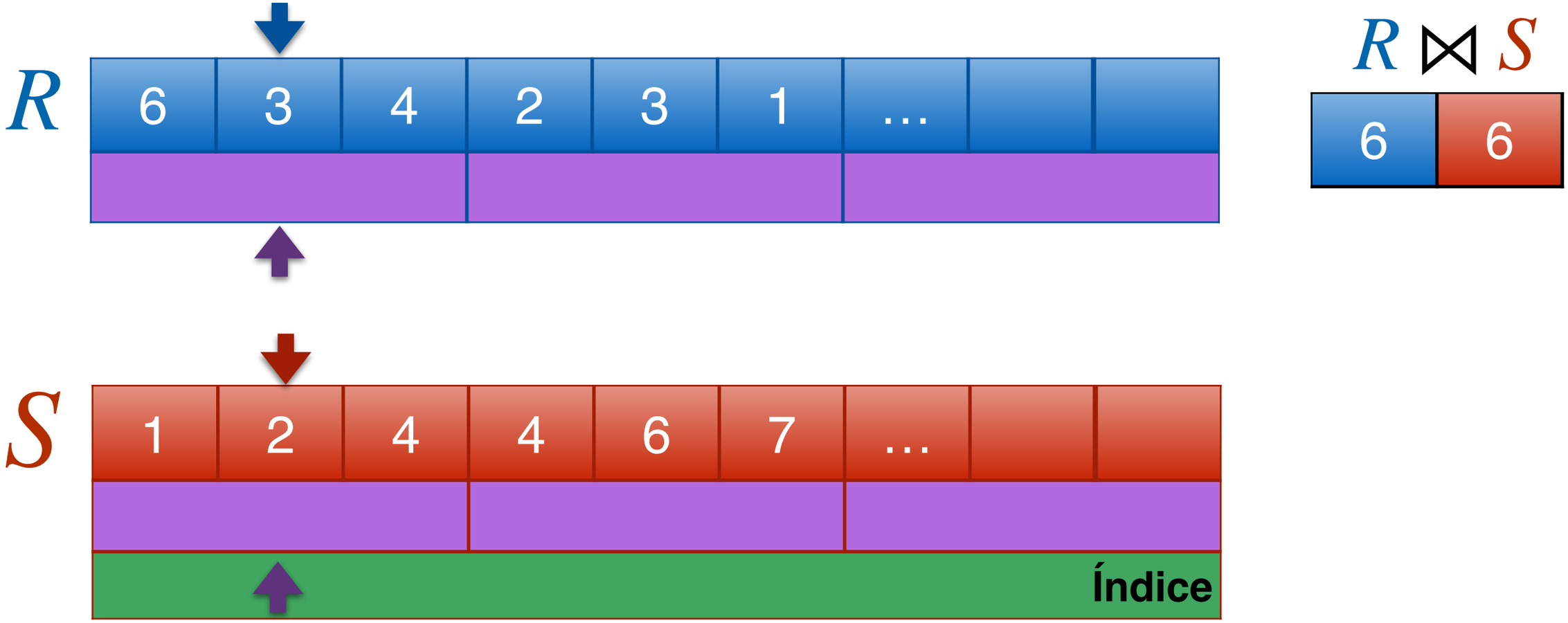


# Loop anidado (con índices)

$R \bowtie S$

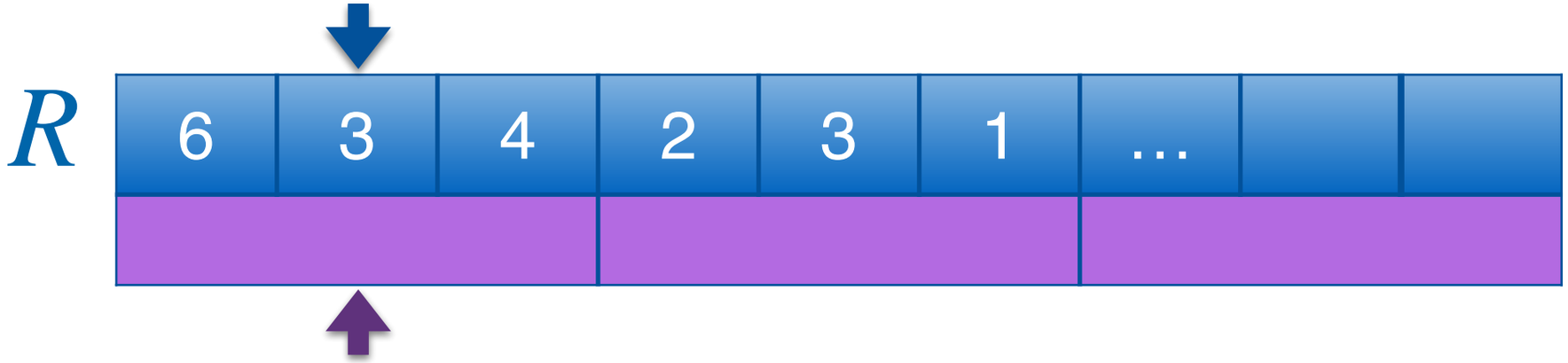
- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

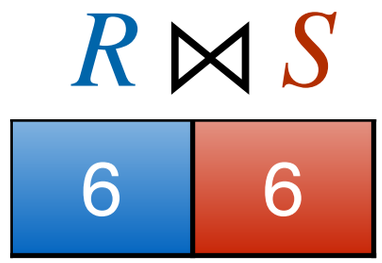


# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

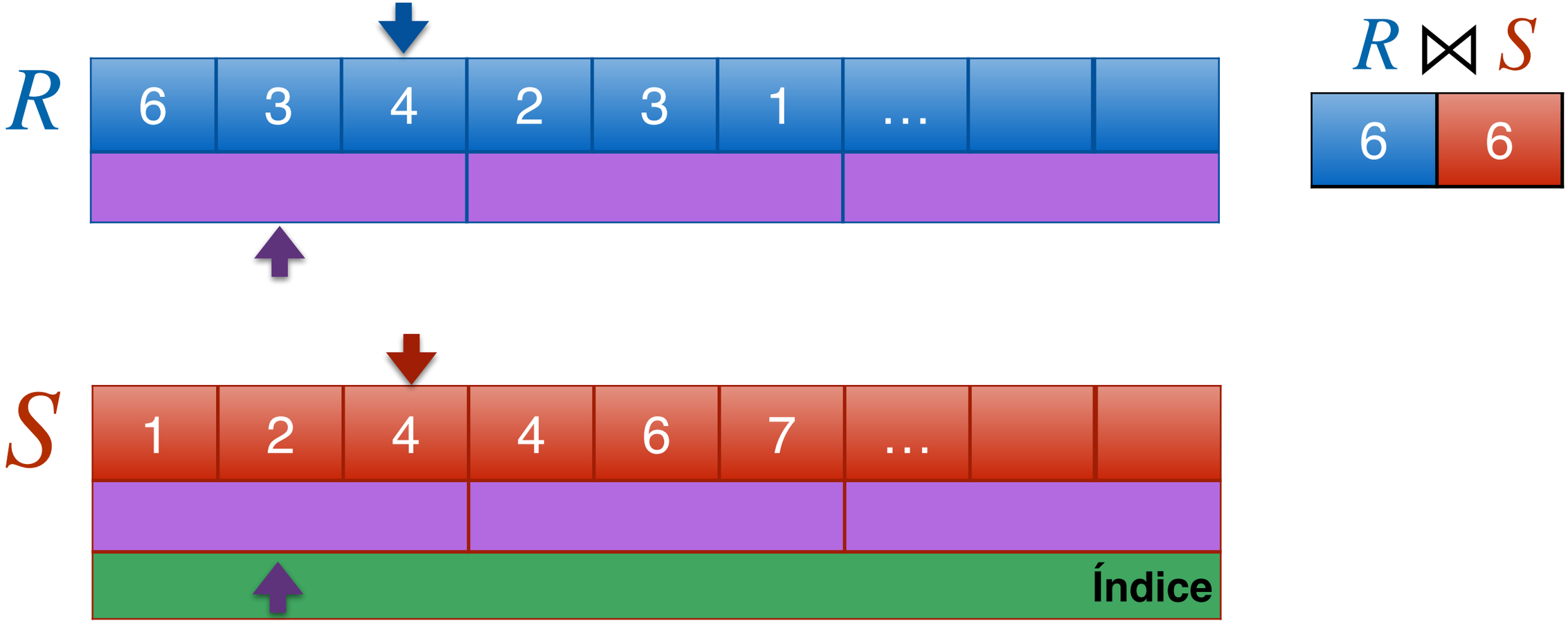


# Loop anidado (con índices)

$R \bowtie S$

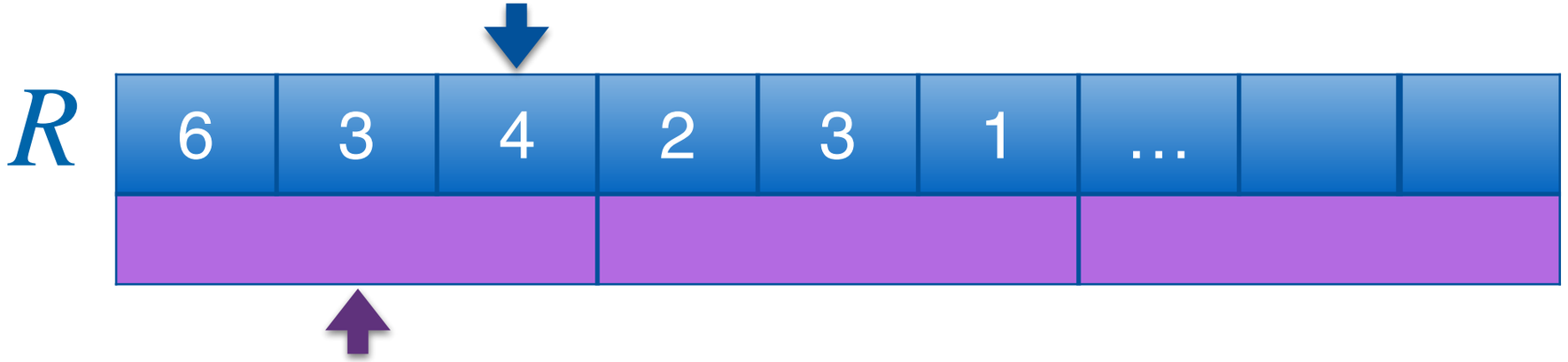
- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

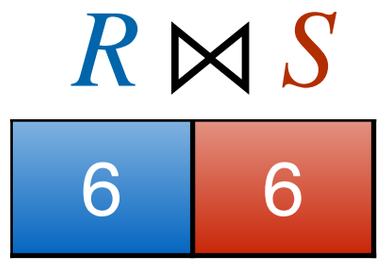


# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

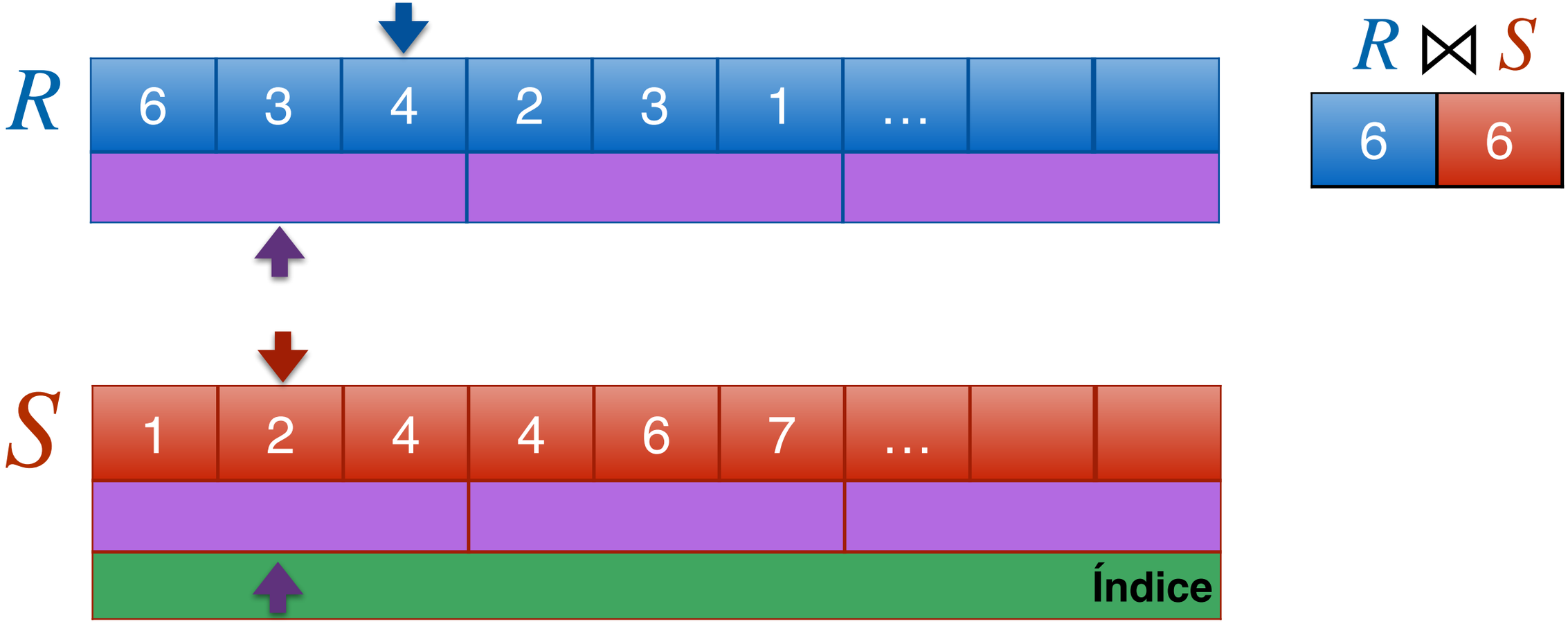


# Loop anidado (con índices)

$R \bowtie S$

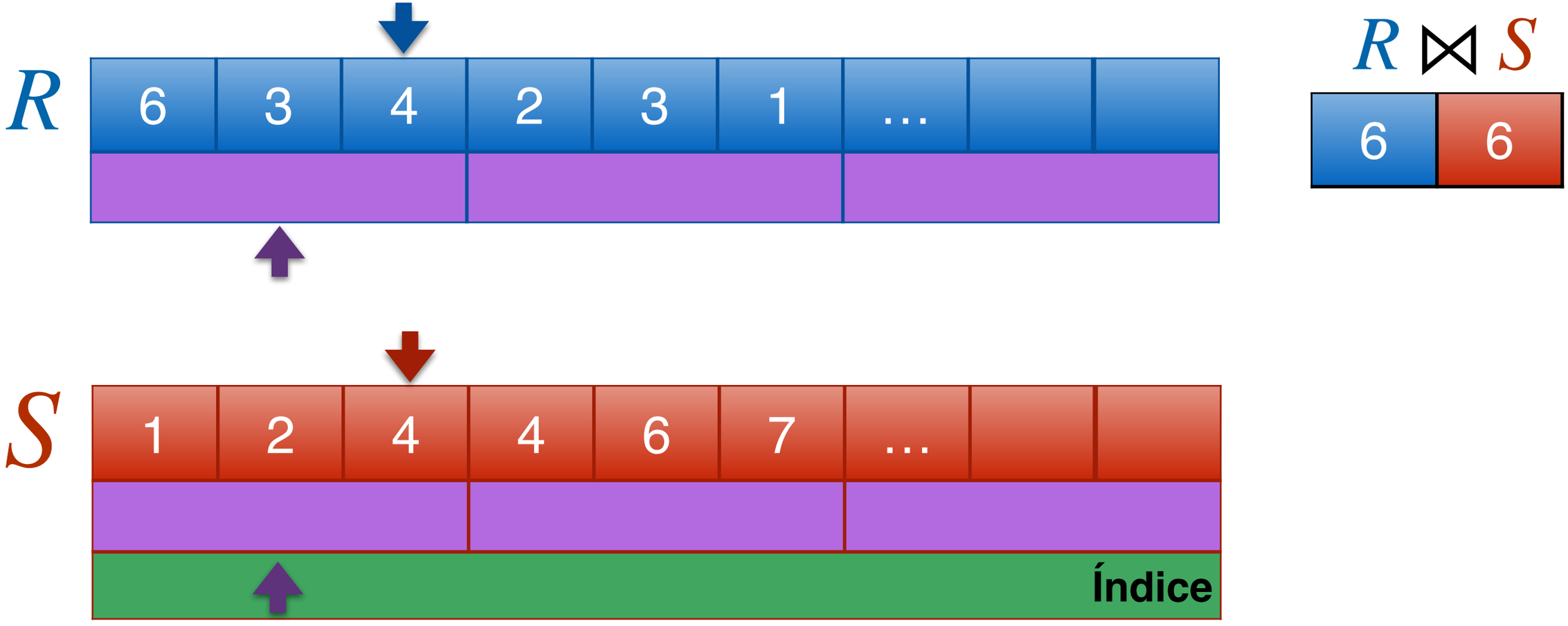
- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



# Loop anidado (con índices)

$R \bowtie S$

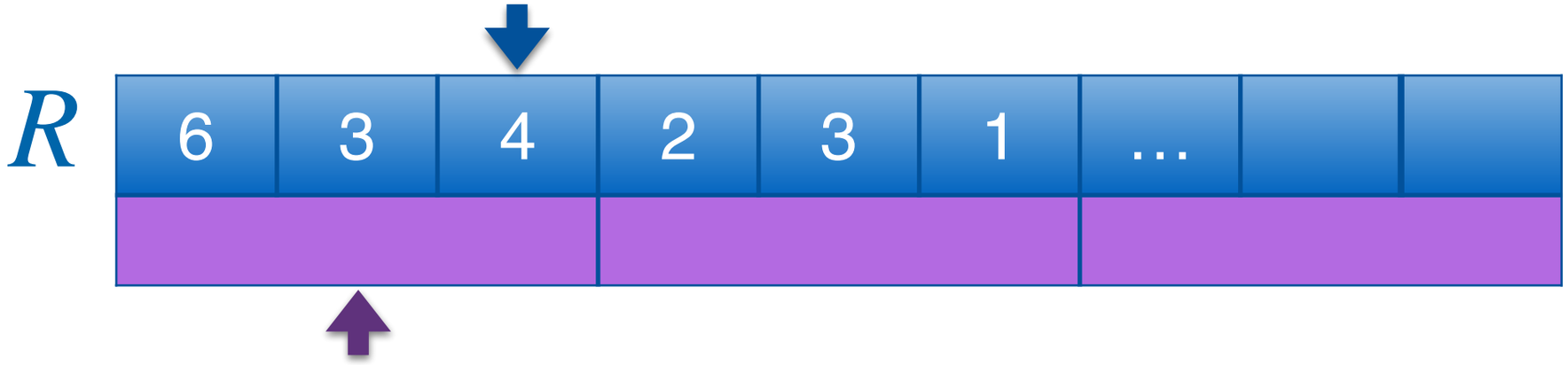
- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



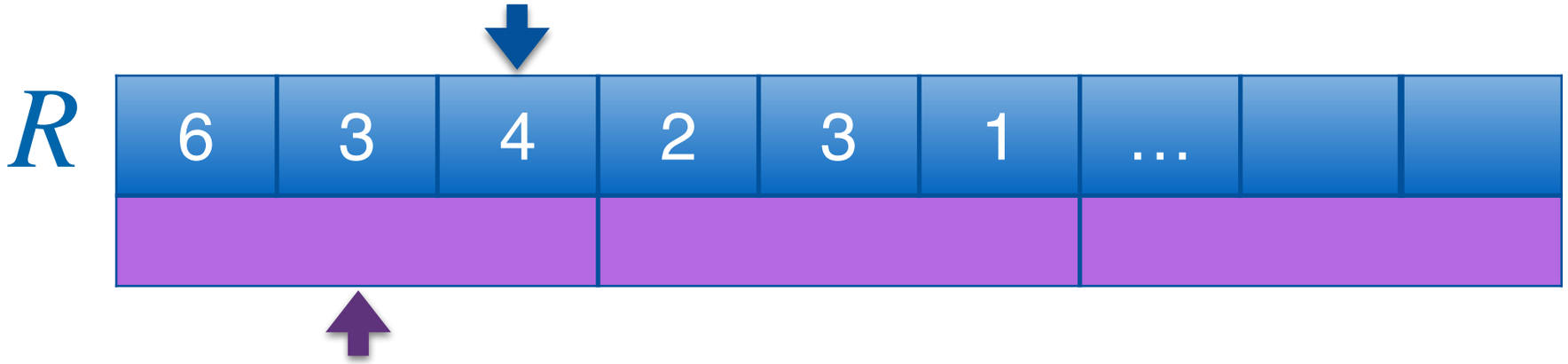

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



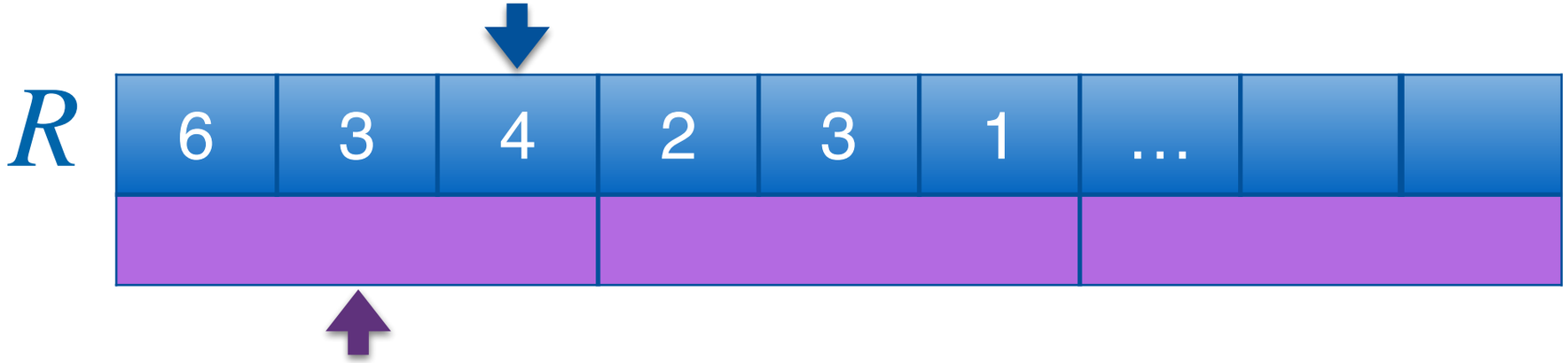

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



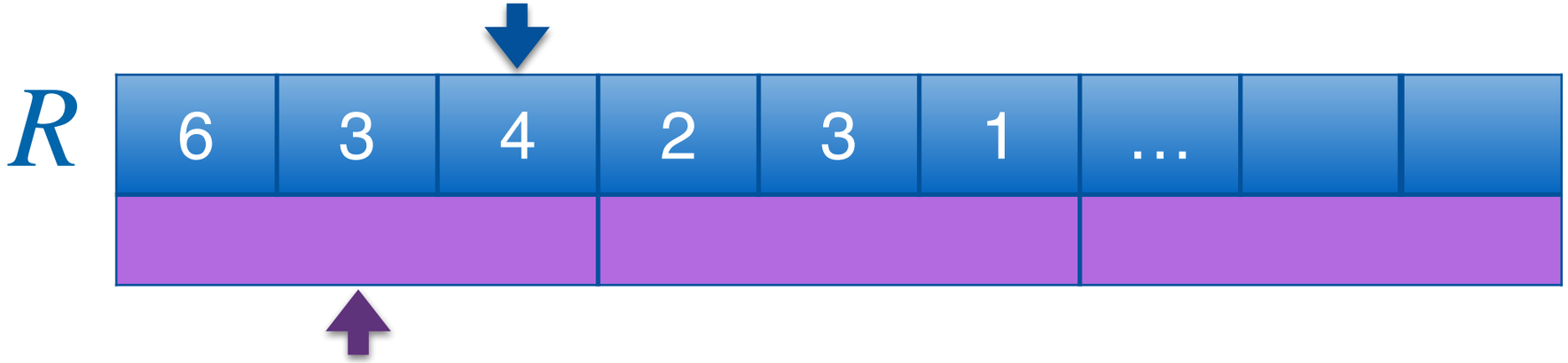

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |
| 4 | 4 |

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



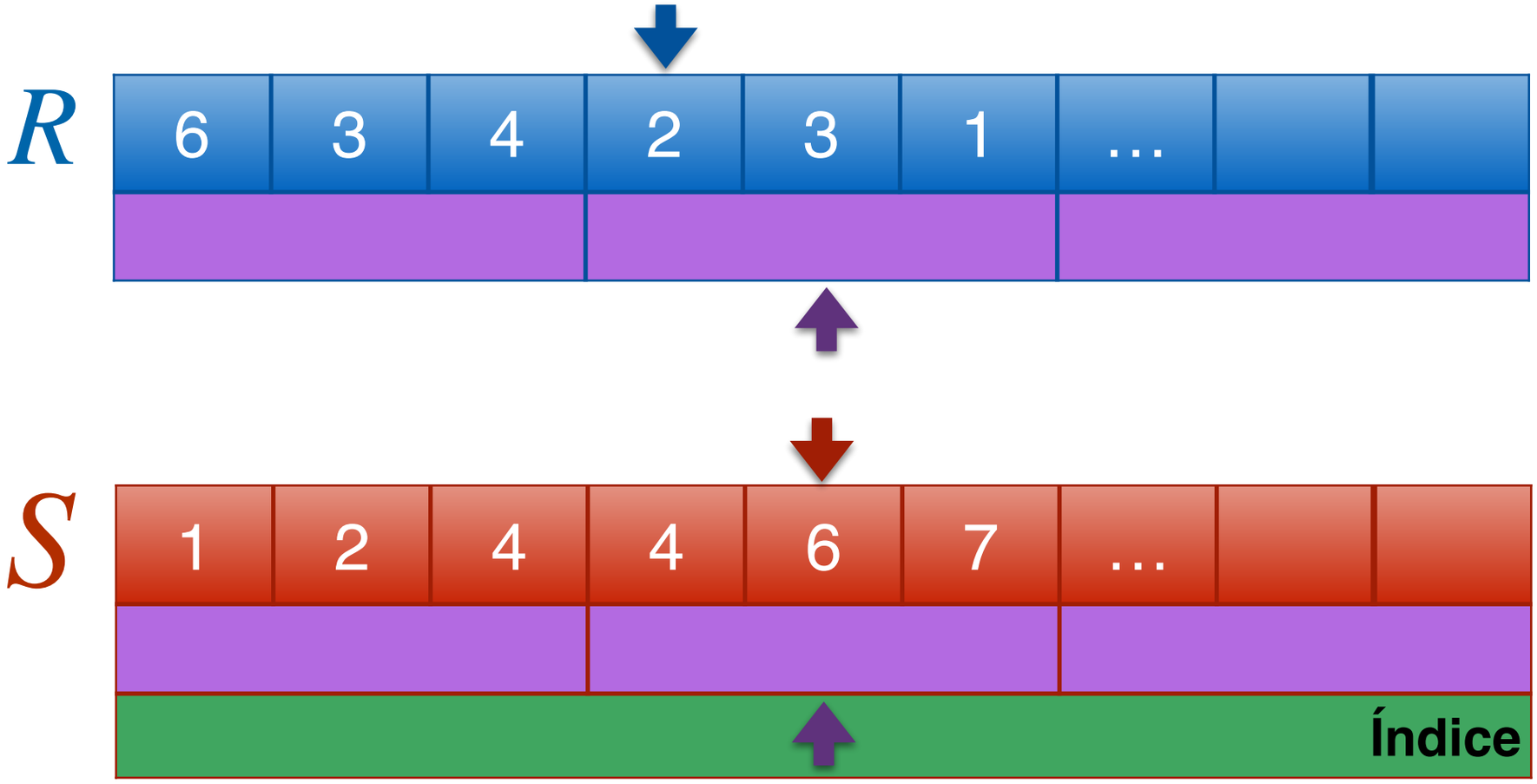

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |
| 4 | 4 |

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



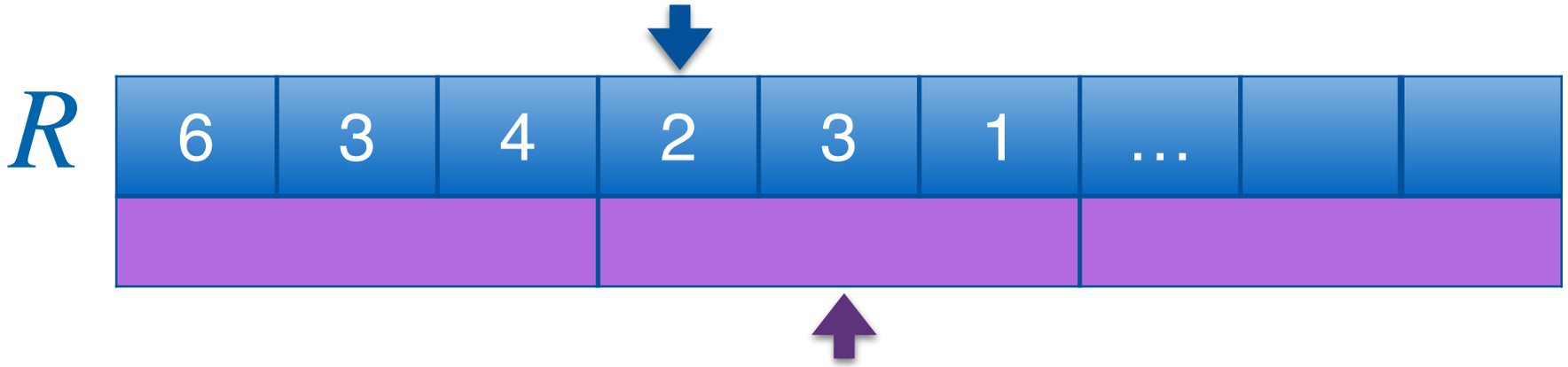

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |
| 4 | 4 |

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



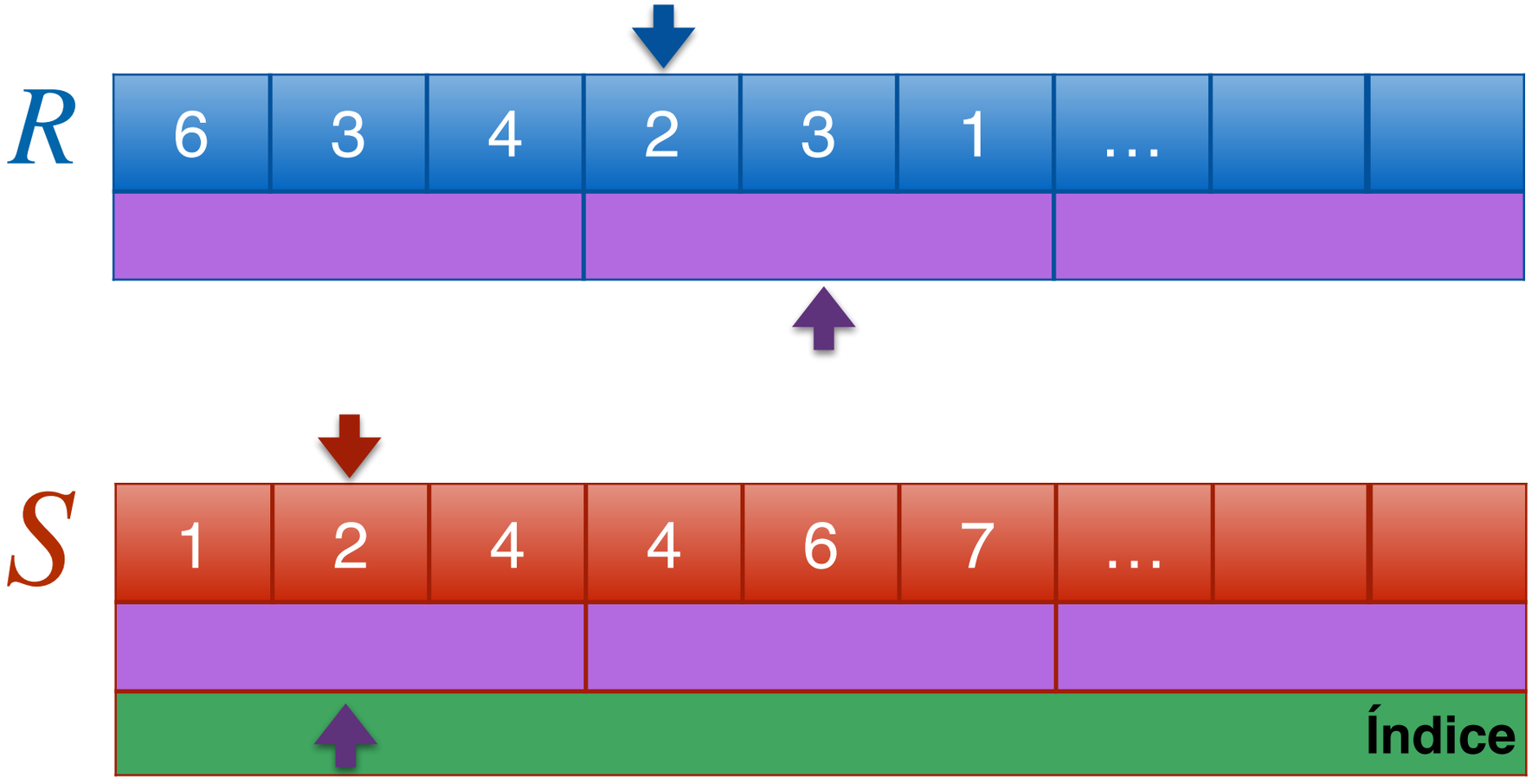

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |
| 4 | 4 |

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



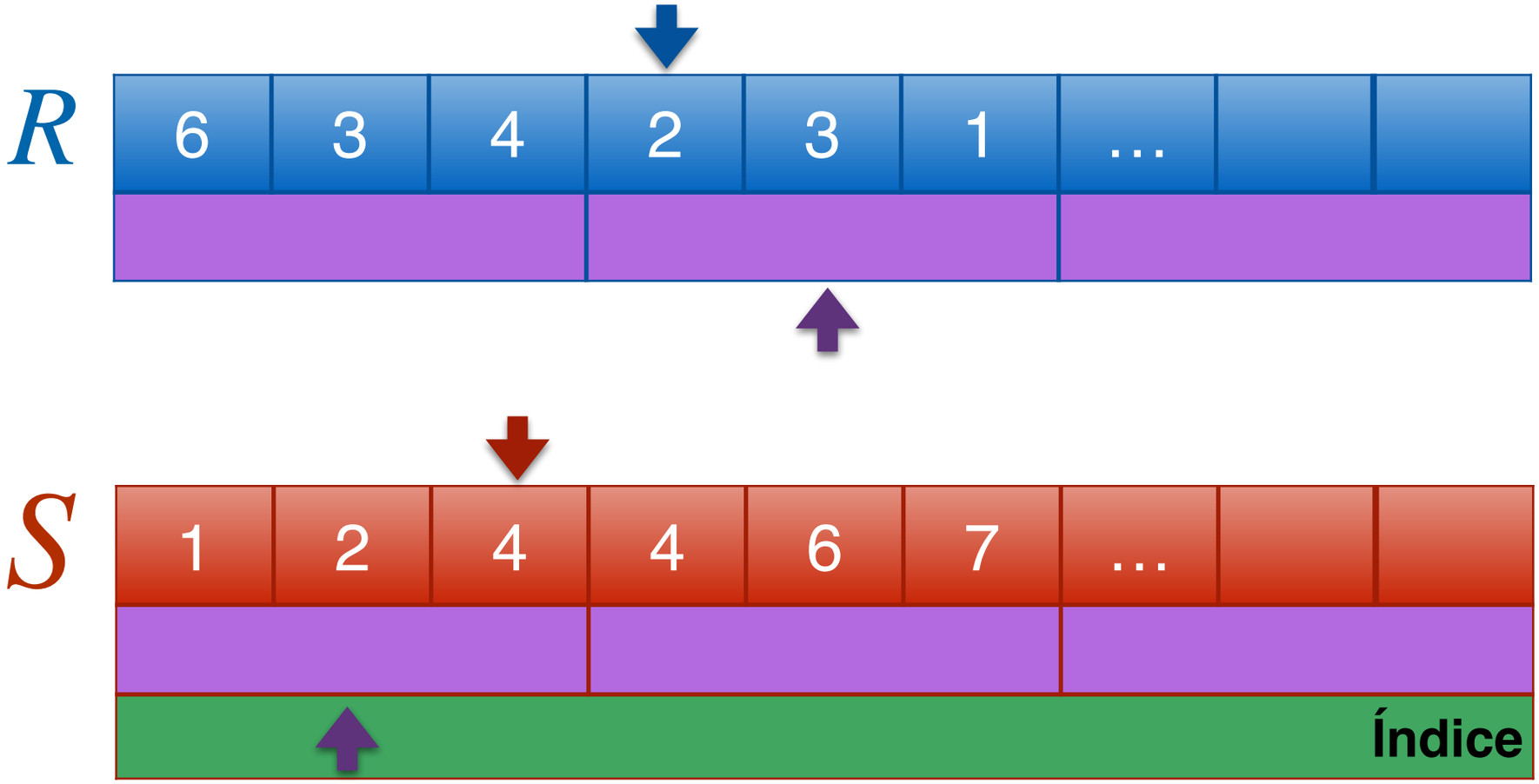

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |
| 4 | 4 |
| 2 | 2 |

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



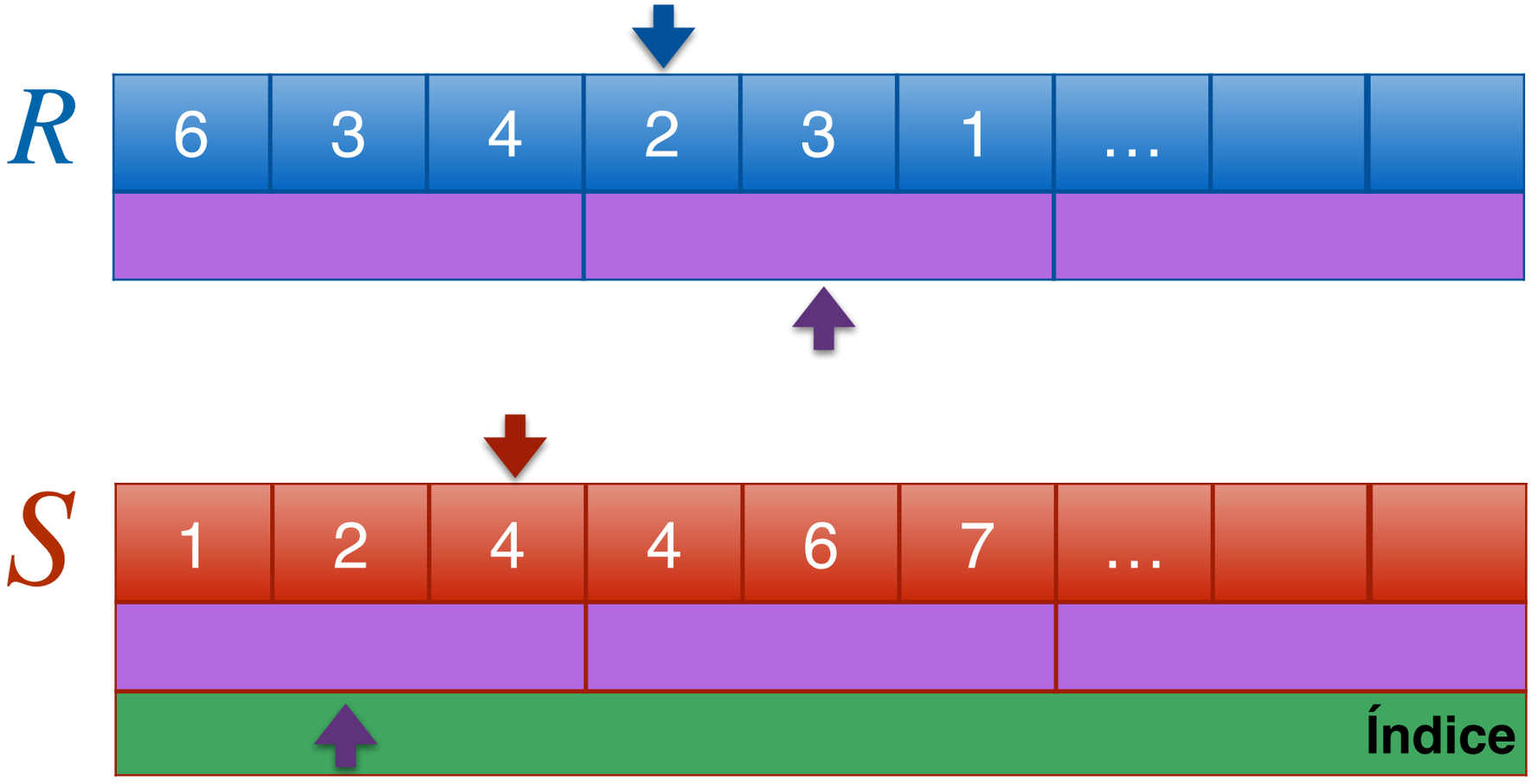

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |
| 4 | 4 |
| 2 | 2 |

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$




$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |
| 4 | 4 |
| 2 | 2 |

...

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

$$\left\lceil \frac{|R|}{B} \right\rceil + |R| \cdot \beta(S)$$

**¿Memoria?**

$2B$ tuplas

**¿Elegir $R$ y $S$?**

$|R| < |S|$

(Ahorro de tiempo)

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

$$\left\lceil \frac{|R|}{B} \right\rceil + |R| \cdot \beta(S)$$

costo de buscar $S$

**¿Memoria?**

$2B$ tuplas

**¿Elegir $R$ y $S$?**

$|R| < |S|$

(Ahorro de tiempo)

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

$$\left\lceil \frac{|R|}{B} \right\rceil + |R| \cdot \beta(S)$$

costo de buscar $S$

peor caso

**¿Memoria?**

$2B$ tuplas

**¿Elegir $R$ y $S$?**

$|R| < |S|$

(Ahorro de tiempo)

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

$$\left\lceil \frac{|R|}{B} \right\rceil + |R| \cdot \beta(S)$$

costo de buscar $S$

peor caso

$$\left\lceil \frac{|S|}{B} \right\rceil$$

**¿Memoria?**

$2B$ tuplas

**¿Elegir $R$ y $S$?**

$|R| < |S|$

(Ahorro de tiempo)

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

$$\left\lceil \frac{|R|}{B} \right\rceil + |R| \cdot \beta(S)$$

costo de buscar $S$

peor caso

Árbol B+ (mem s.)

$$\left\lceil \frac{|S|}{B} \right\rceil$$

**¿Memoria?**

$2B$ tuplas

**¿Elegir $R$ y $S$?**

$|R| < |S|$

(Ahorro de tiempo)

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

costo de buscar $S$

$$\left\lceil \frac{|R|}{B} \right\rceil + |R| \cdot \beta(S)$$

peor caso

$$\left\lceil \frac{|S|}{B} \right\rceil$$

Árbol B+ (mem s.)

$$O(log_b(\left\lceil \frac{|S|}{B} \right\rceil))$$

**¿Memoria?**

$2B$ tuplas

**¿Elegir $R$ y $S$?**

$|R| < |S|$

(Ahorro de tiempo)

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

$$\left\lceil \frac{|R|}{B} \right\rceil + |R| \cdot \beta(S)$$

costo de buscar $S$

- peor caso: $\left\lceil \frac{|S|}{B} \right\rceil$
- Árbol B+ (mem s.): $O(log_b(\left\lceil \frac{|S|}{B} \right\rceil))$
- Hash/Árbol B+ (mem p.)

**¿Memoria?**

$2B$ tuplas

**¿Elegir $R$ y $S$?**

$|R| < |S|$

(Ahorro de tiempo)

# Loop anidado (con índices)

$R \bowtie S$

- Para cada tupla $r \in R$
  - Buscar $s \in S$ en el índice tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

$$\left\lceil \frac{|R|}{B} \right\rceil + |R| \cdot \beta(S)$$

costo de buscar $S$

- peor caso: $\left\lceil \frac{|S|}{B} \right\rceil$
- Árbol B+ (mem s.): $O(log_b(\left\lceil \frac{|S|}{B} \right\rceil))$
- Hash/Árbol B+ (mem p.): $O(1)$

**¿Memoria?**

$2B$ tuplas

**¿Elegir $R$ y $S$?**

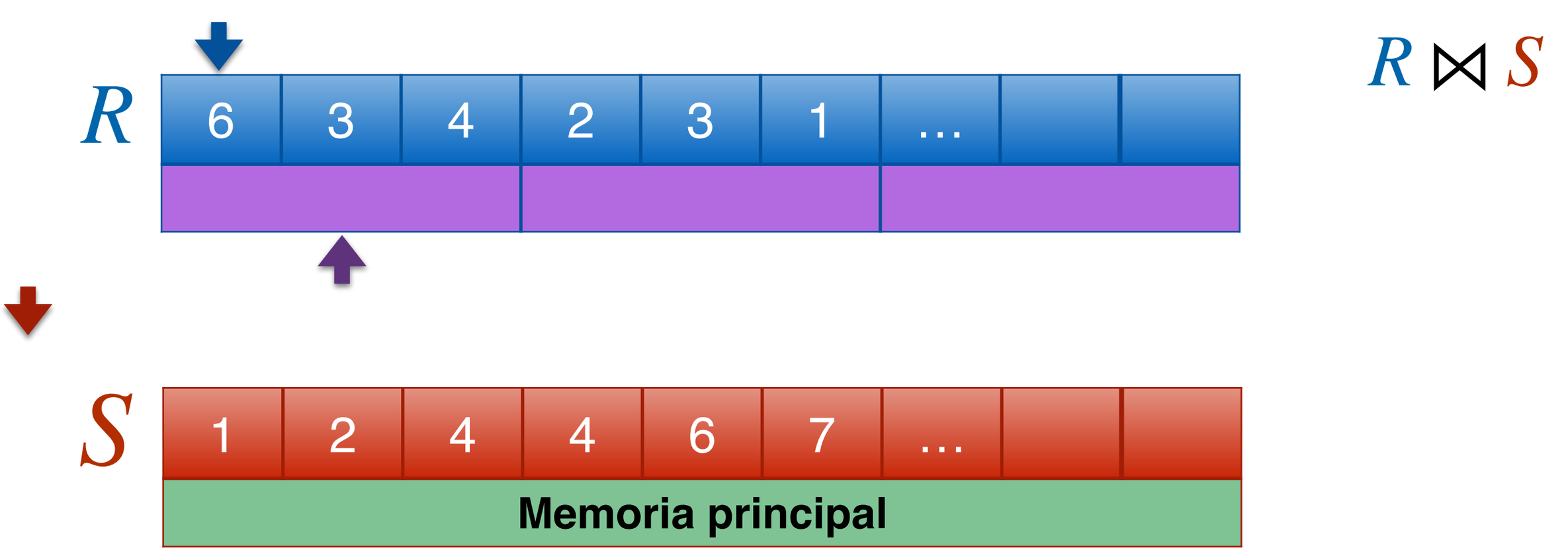
$|R| < |S|$

(Ahorro de tiempo)

# Hash-join

$R \bowtie S$

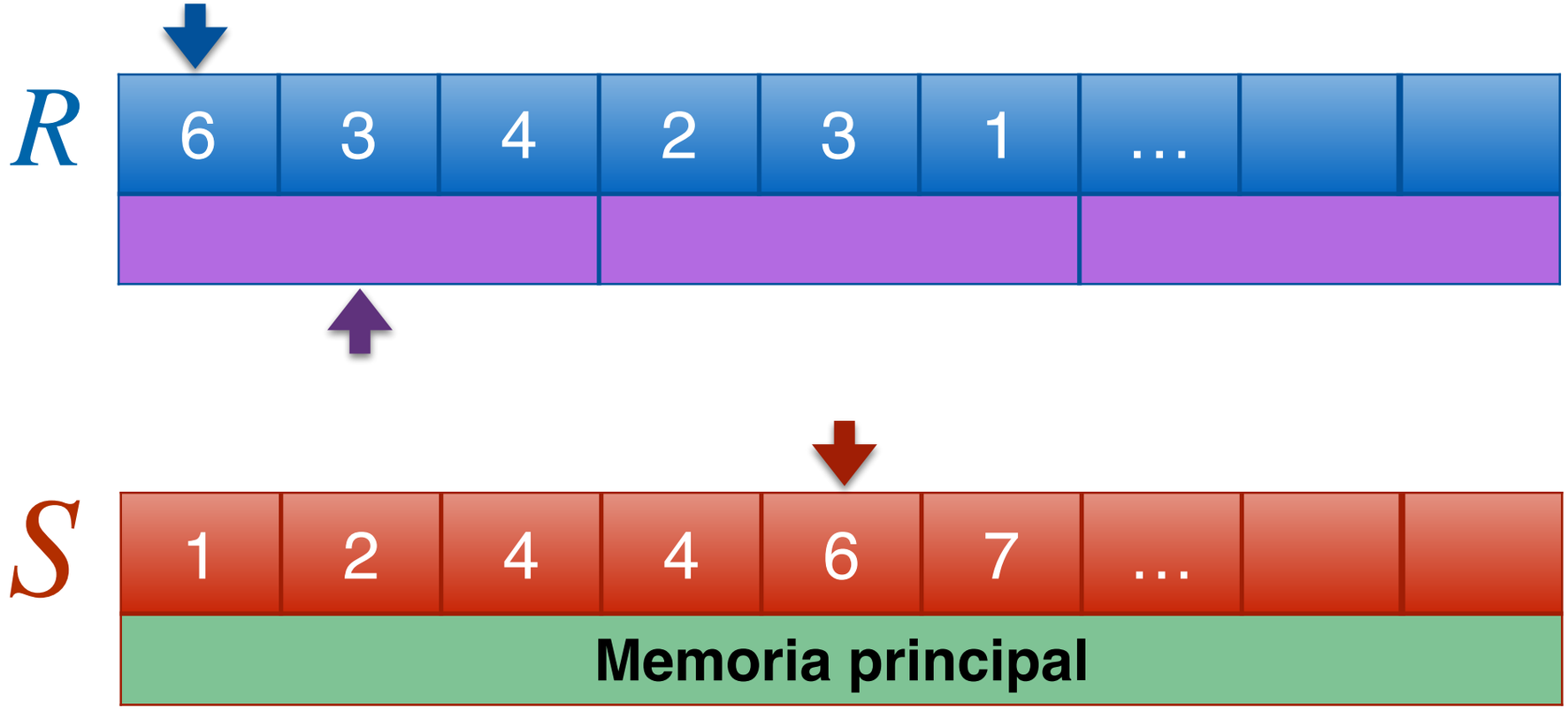
- Guardar $S$ en memoria principal
- Para cada tupla $r \in R$
  - Buscar $s$ en memoria principal tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



# Hash-join

$R \bowtie S$

- Guardar $S$ en memoria principal
- Para cada tupla $r \in R$
  - Buscar $s$ en memoria principal tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

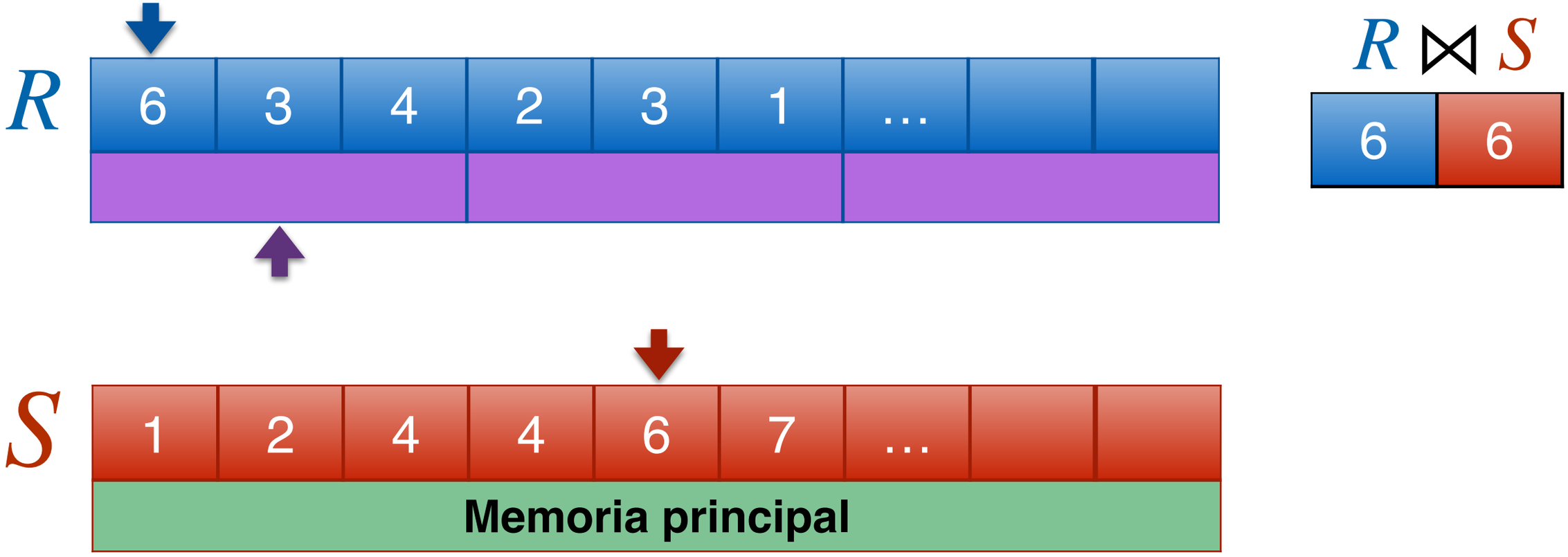



$R \bowtie S$

# Hash-join

$R \bowtie S$

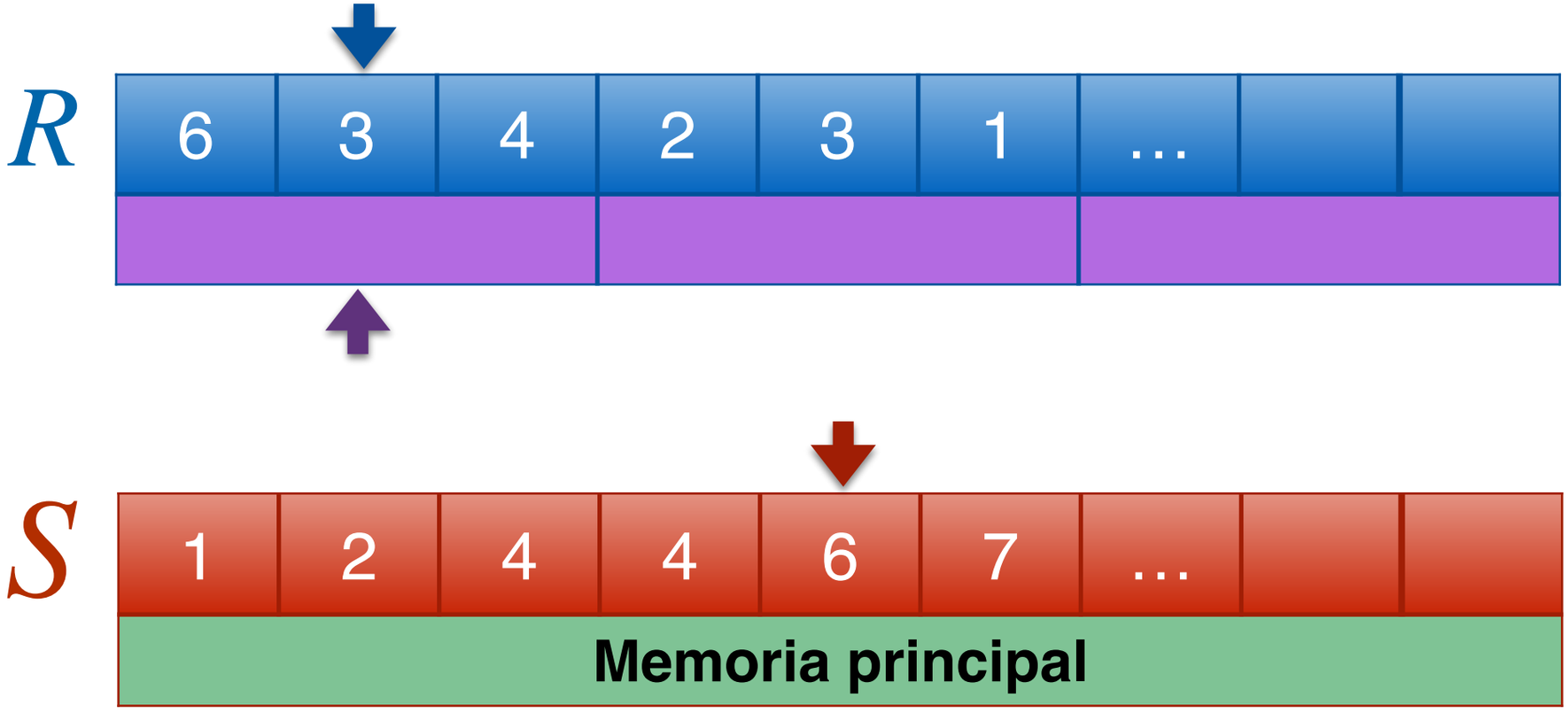
- Guardar $S$ en memoria principal
- Para cada tupla $r \in R$
  - Buscar $s$ en memoria principal tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



# Hash-join

$R \bowtie S$

- Guardar $S$ en memoria principal
- Para cada tupla $r \in R$
  - Buscar $s$ en memoria principal tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



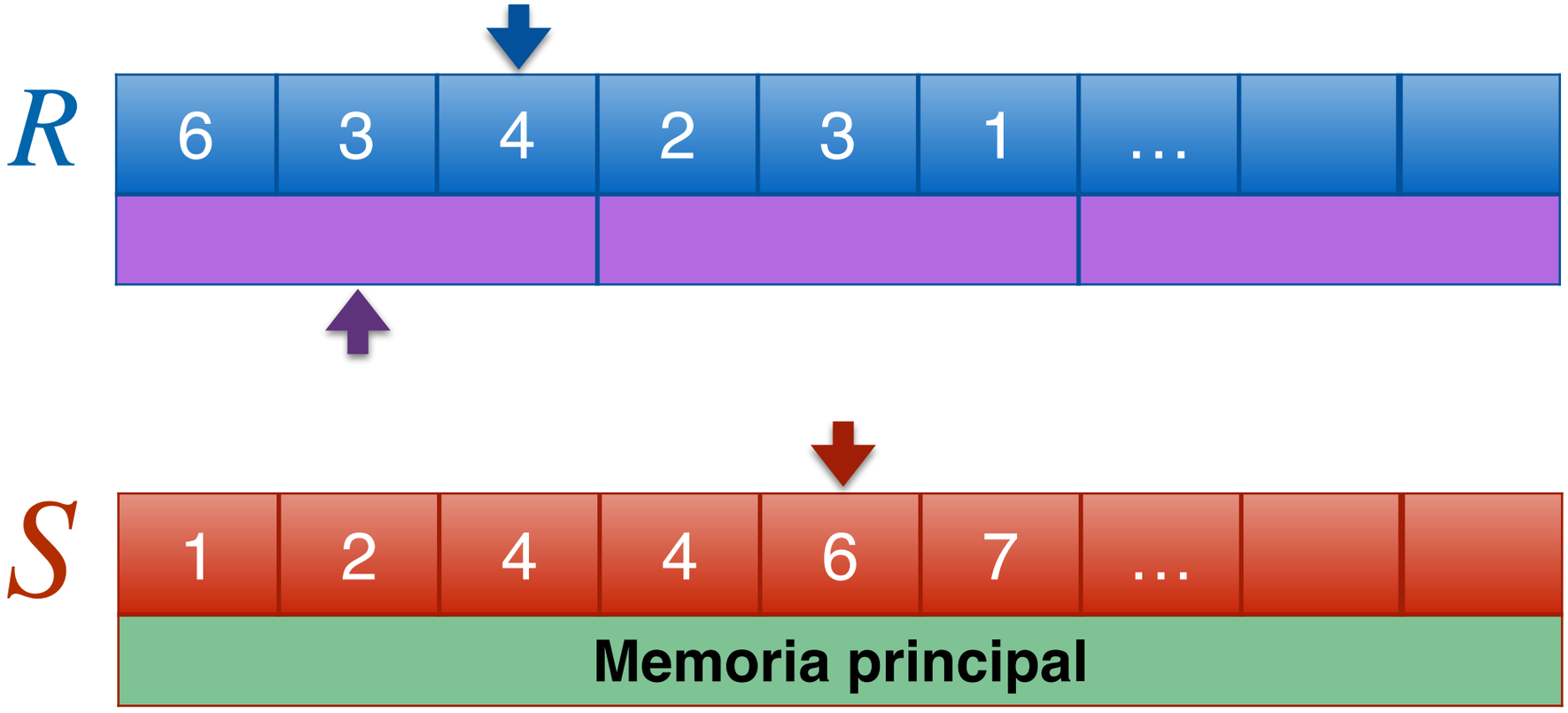

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |

# Hash-join

$R \bowtie S$

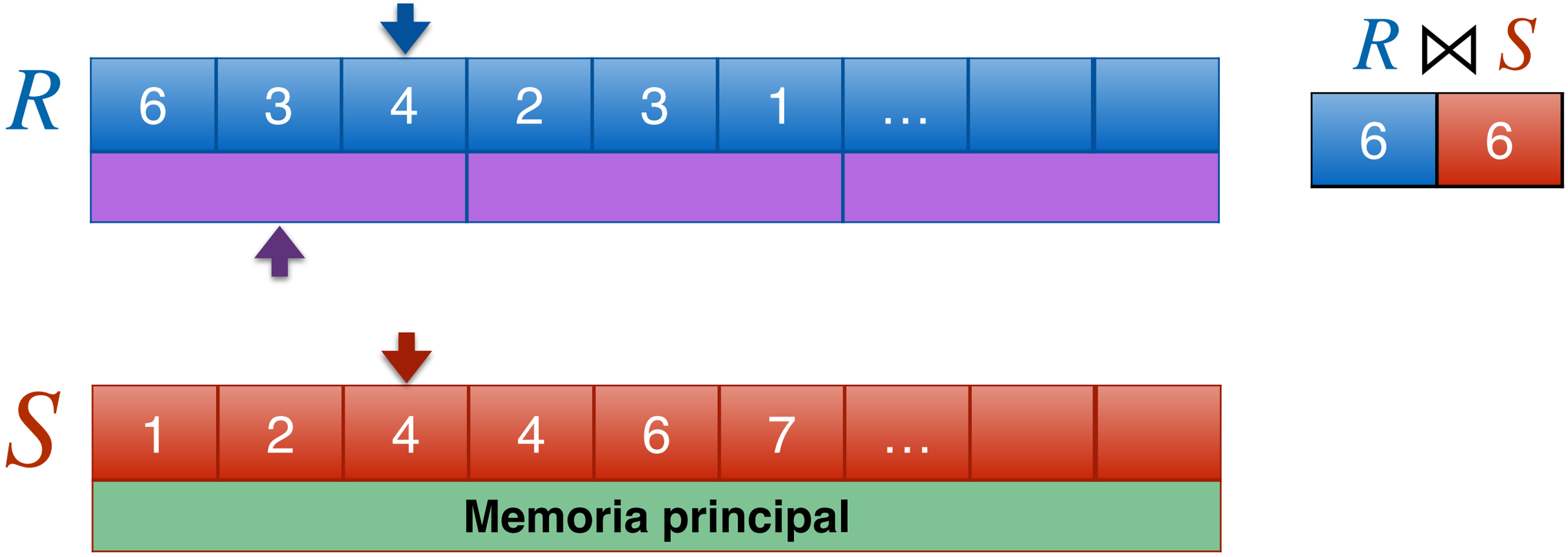
- Guardar $S$ en memoria principal
- Para cada tupla $r \in R$
  - Buscar $s$ en memoria principal tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



# Hash-join

$R \bowtie S$

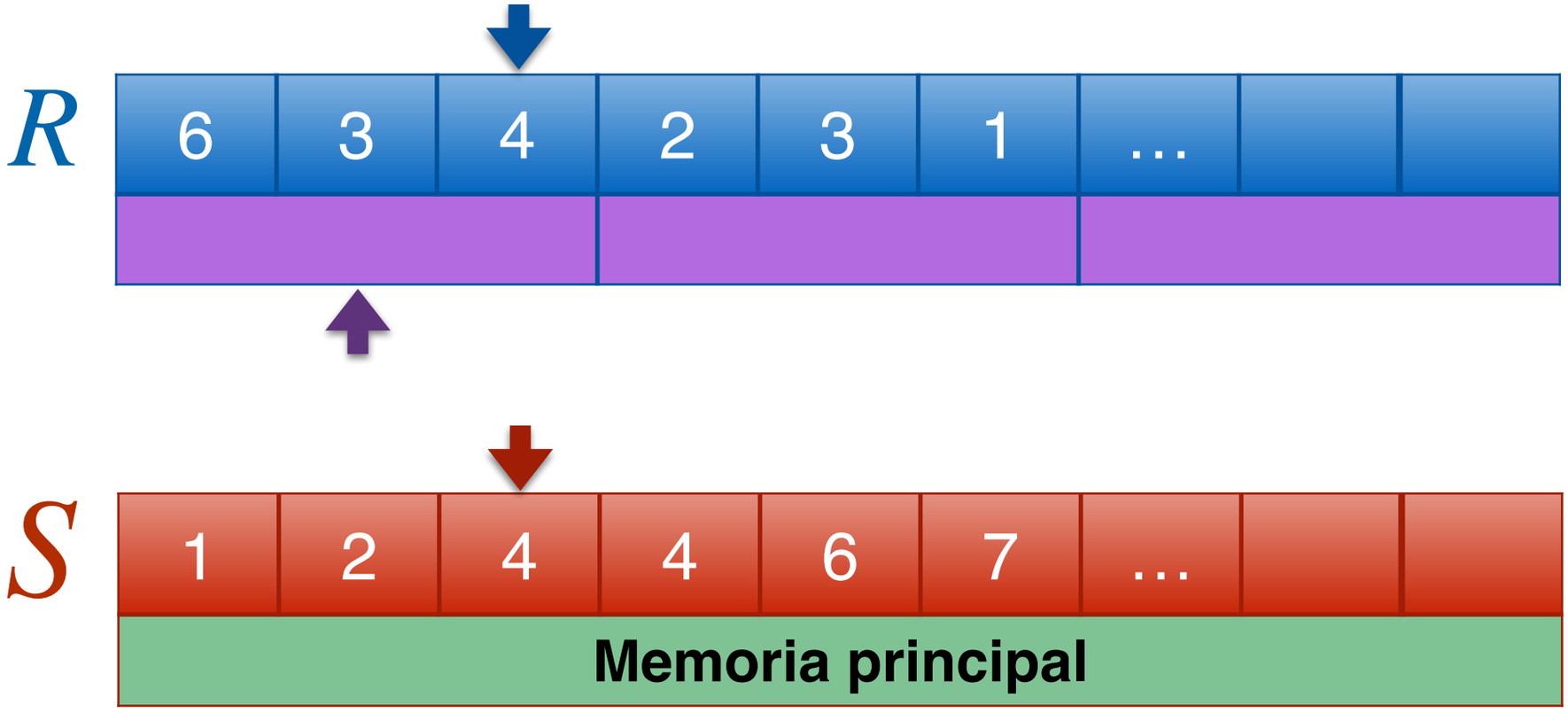
- Guardar $S$ en memoria principal
- Para cada tupla $r \in R$
  - Buscar $s$ en memoria principal tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



# Hash-join

$R \bowtie S$

- Guardar $S$ en memoria principal
- Para cada tupla $r \in R$
  - Buscar $s$ en memoria principal tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



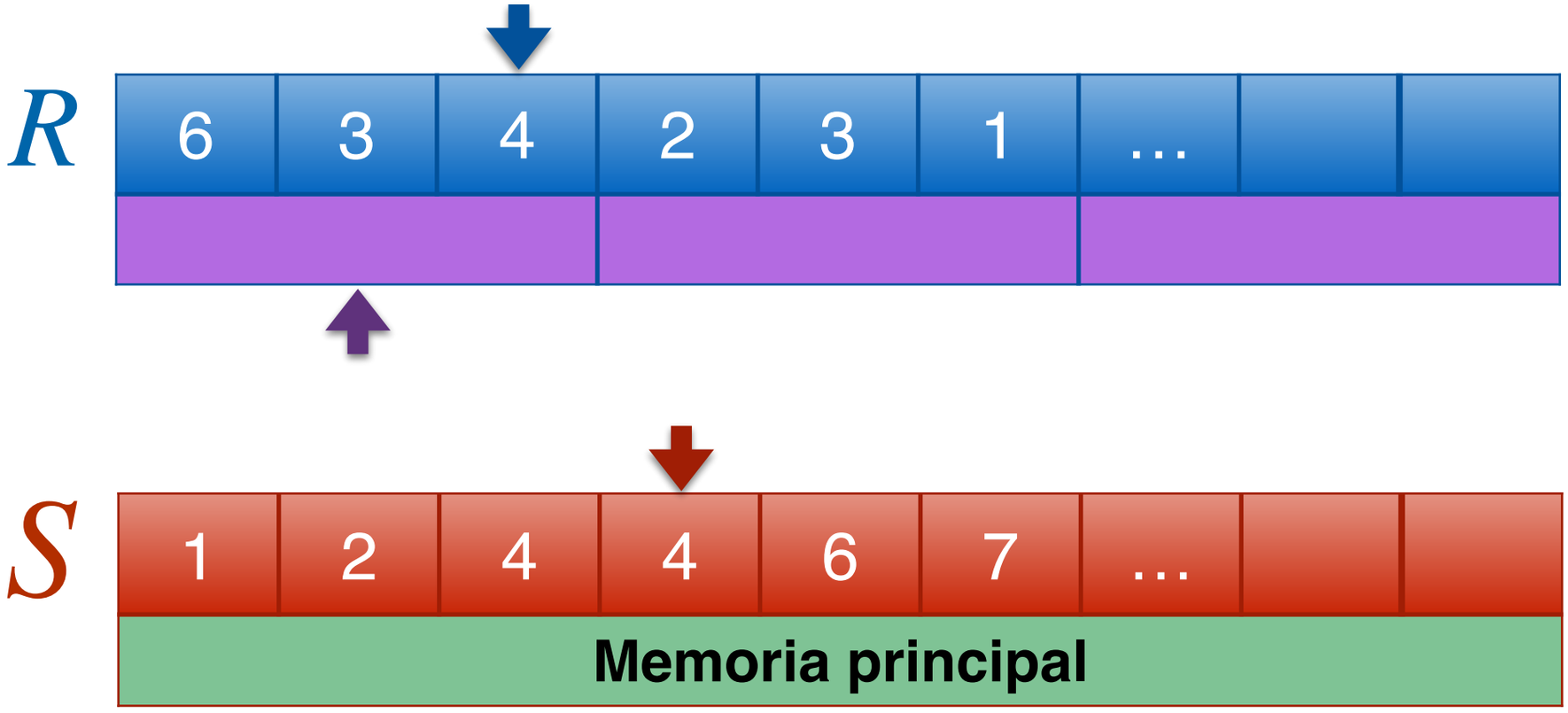

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |

# Hash-join

$R \bowtie S$

- Guardar $S$ en memoria principal
- Para cada tupla $r \in R$
  - Buscar $s$ en memoria principal tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



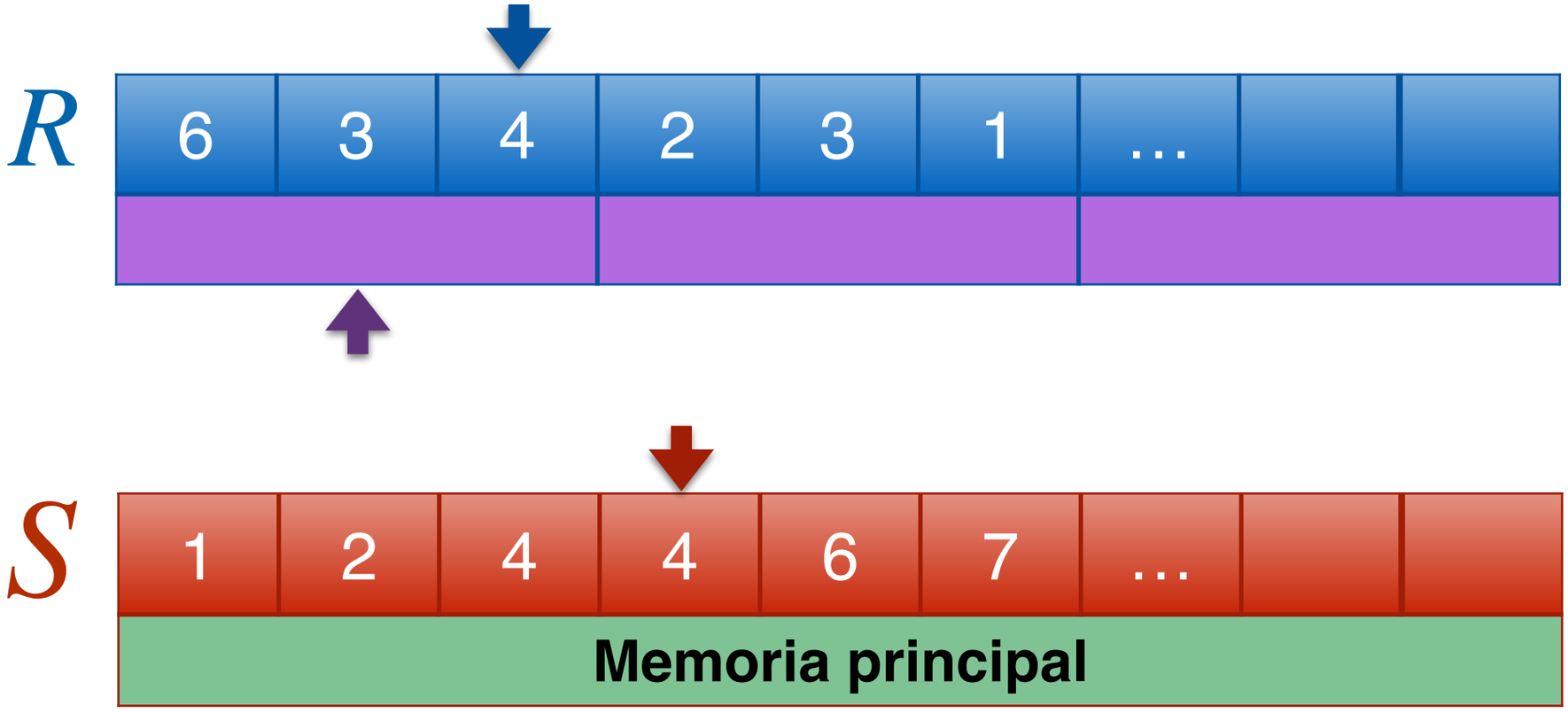

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |

# Hash-join

$R \bowtie S$

- Guardar $S$ en memoria principal
- Para cada tupla $r \in R$
  - Buscar $s$ en memoria principal tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$



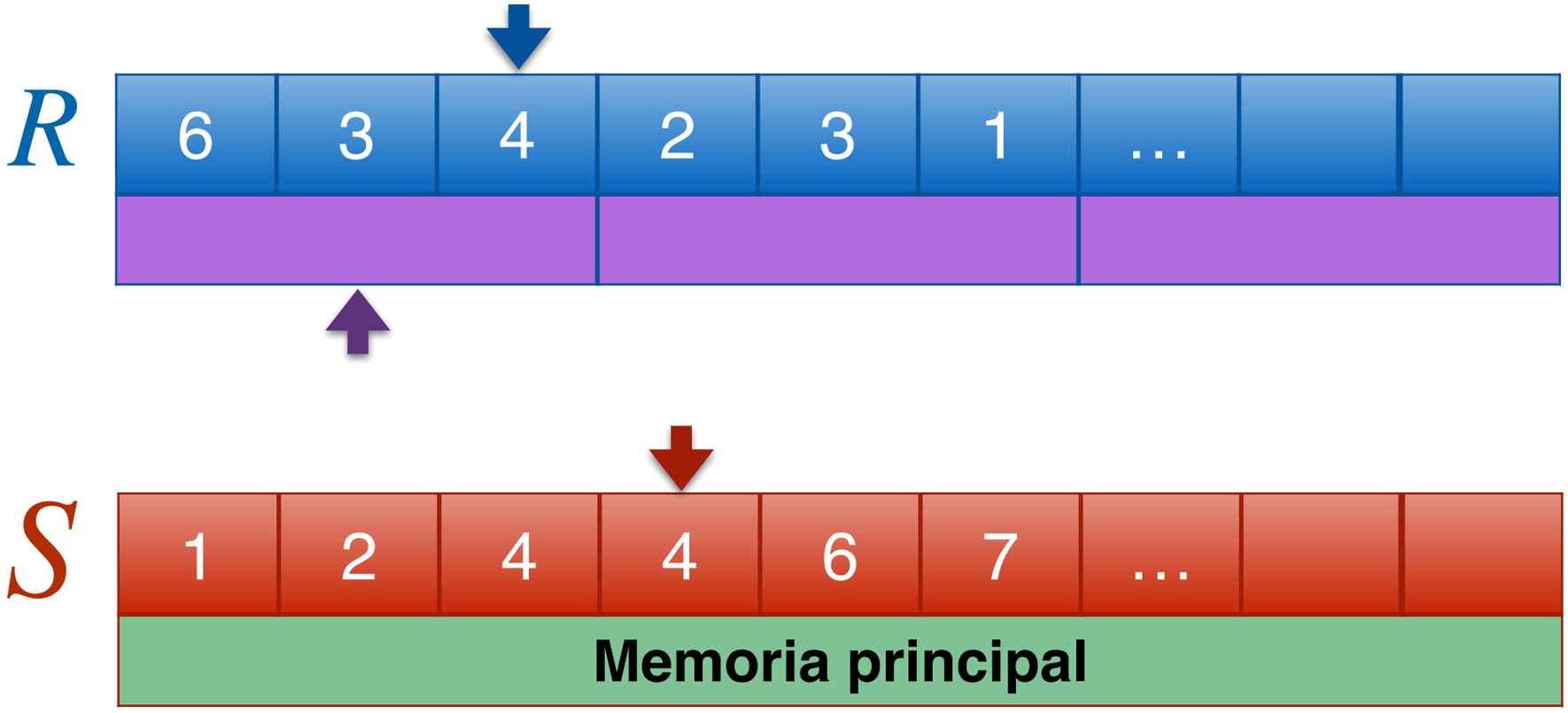

$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |
| 4 | 4 |

# Hash-join

$R \bowtie S$

- Guardar $S$ en memoria principal
- Para cada tupla $r \in R$
  - Buscar $s$ en memoria principal tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$




$R \bowtie S$

| | |
|---|---|
| 6 | 6 |
| 4 | 4 |
| 4 | 4 |

…

# Hash-join

$R \bowtie S$

- Guardar $S$ en memoria principal
- Para cada tupla $r \in R$
  - Buscar $s$ en memoria principal tal que $r$ y $s$ satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

$$\left\lceil \frac{|R|}{B} \right\rceil + \left\lceil \frac{|S|}{B} \right\rceil$$

**¿Memoria?**

$|S| + B$ tuplas

**¿Elegir $R$ y $S$?**

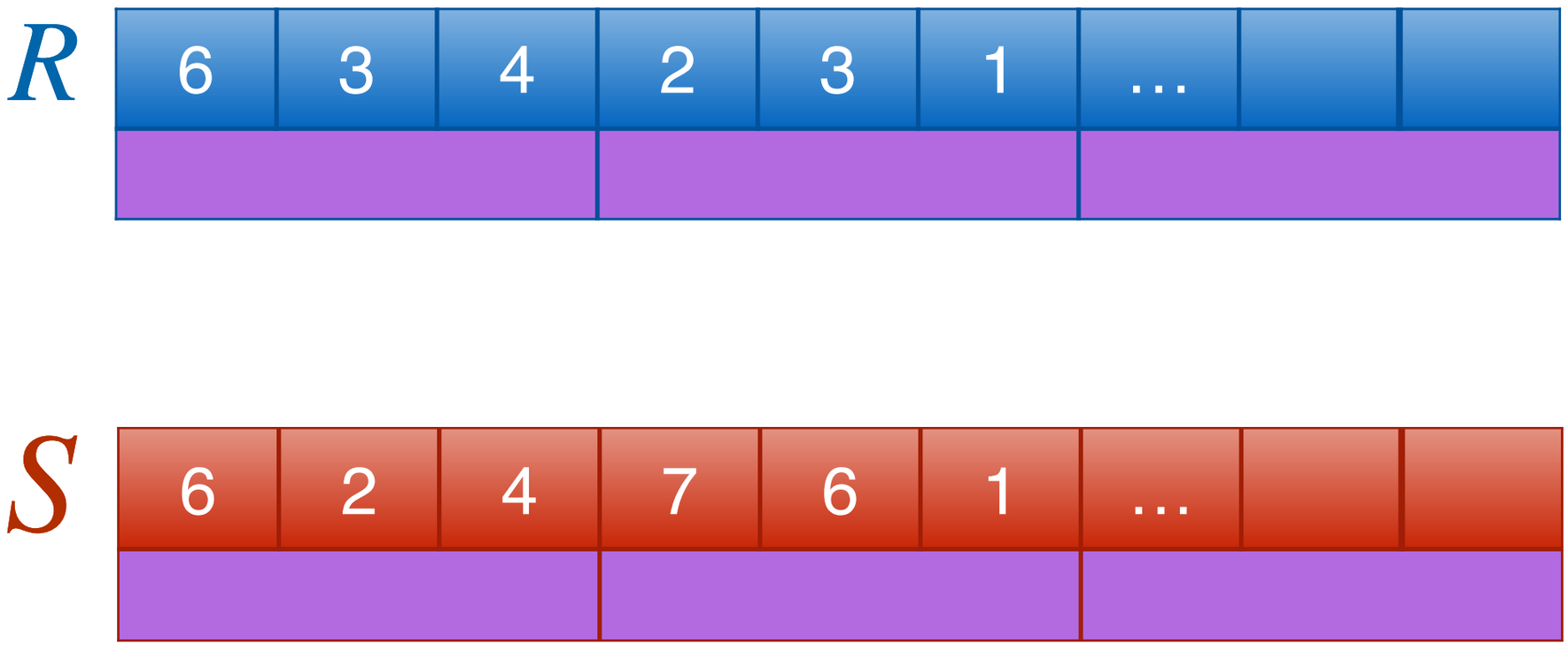
$|S| < |R|$

(Ahorro de memoria)

# Sort-merge-join

$R \bowtie S$

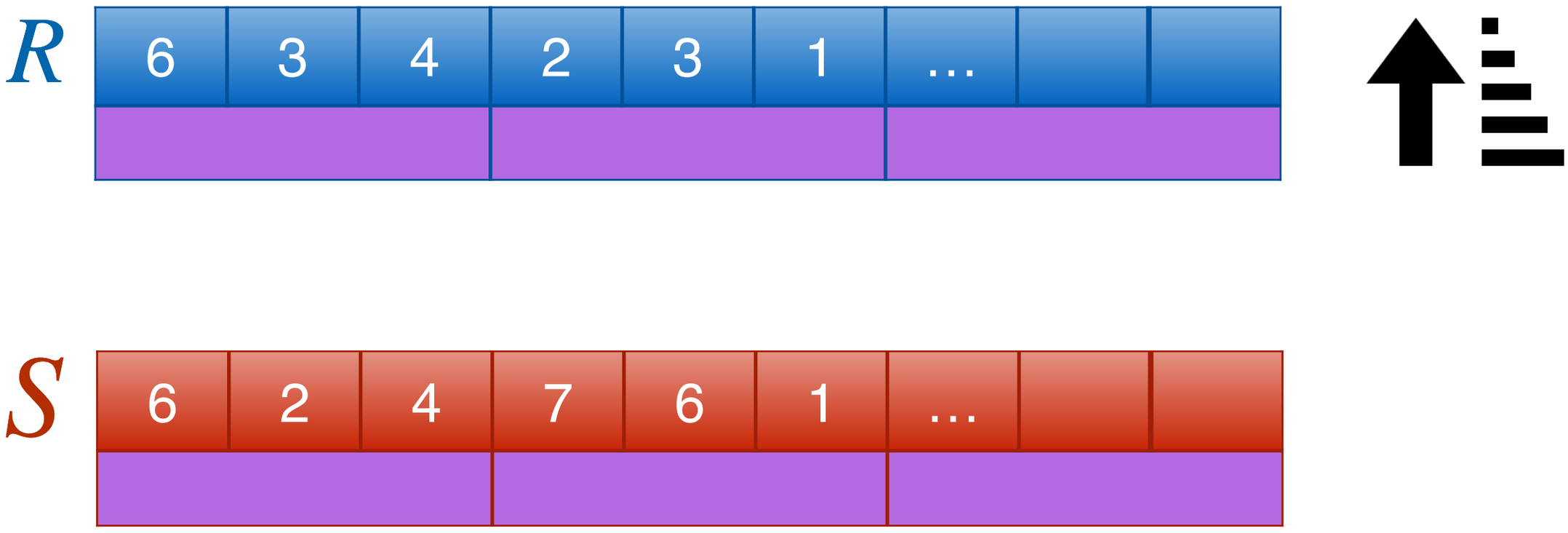
- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



# Sort-merge-join

$R \bowtie S$

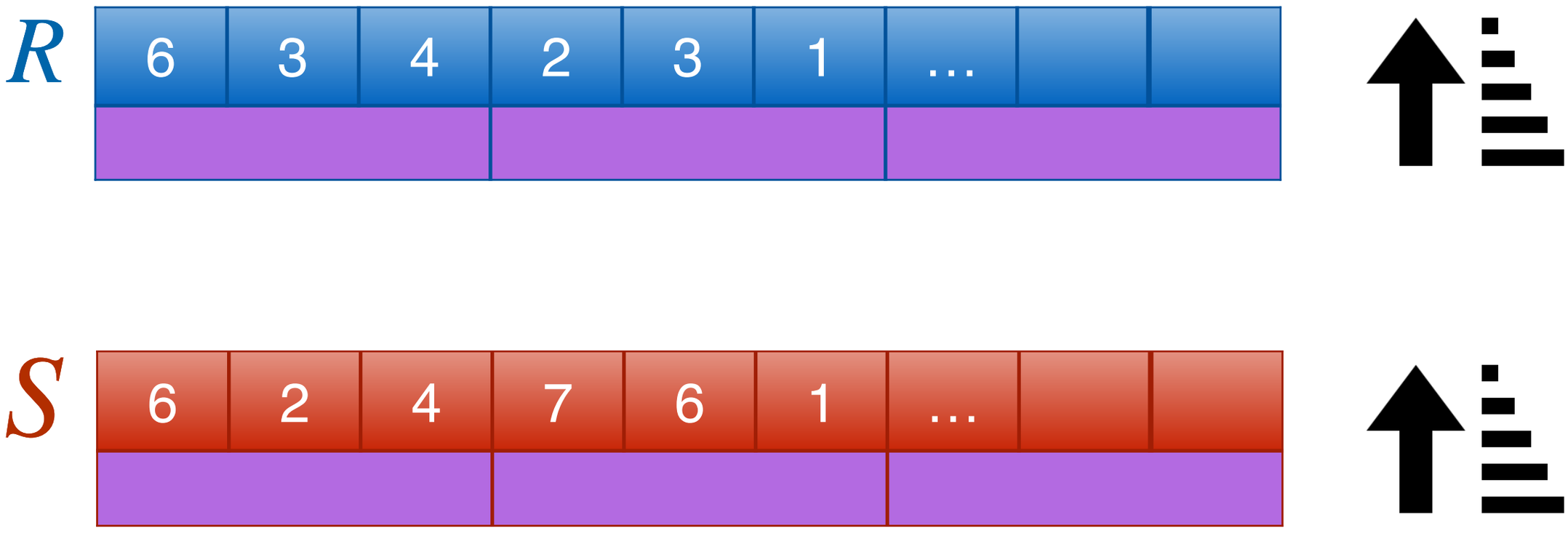
- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



# Sort-merge-join

$R \bowtie S$

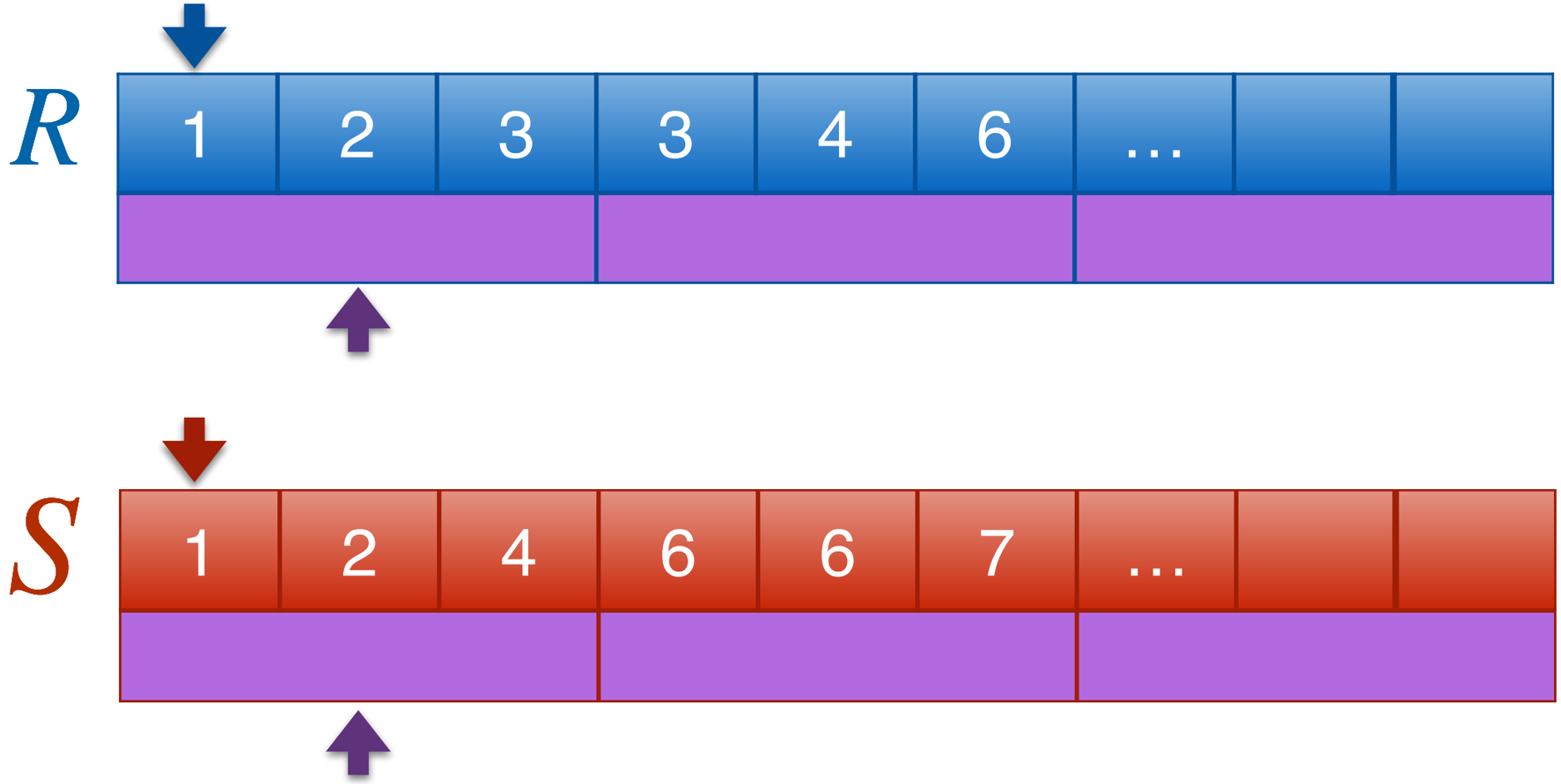
- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



# Sort-merge-join

$R \bowtie S$

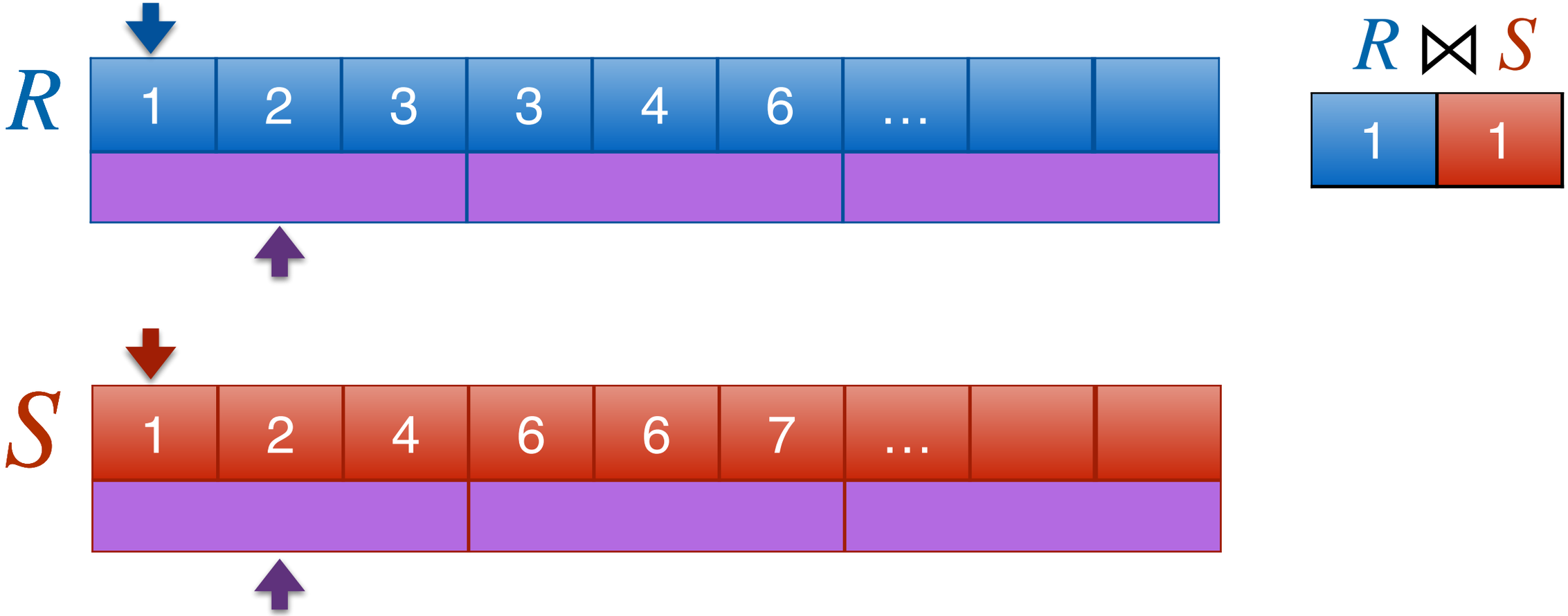
- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



# Sort-merge-join

$R \bowtie S$

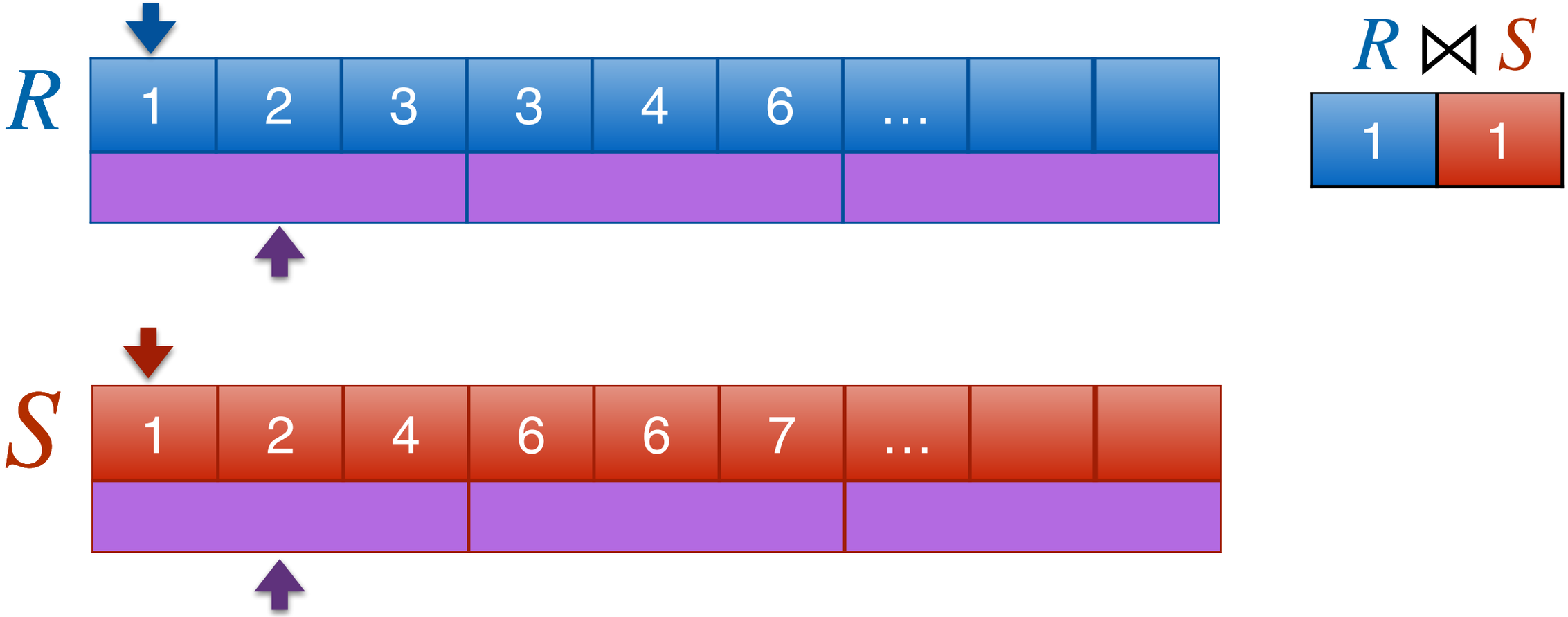
- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



# Sort-merge-join

$R \bowtie S$

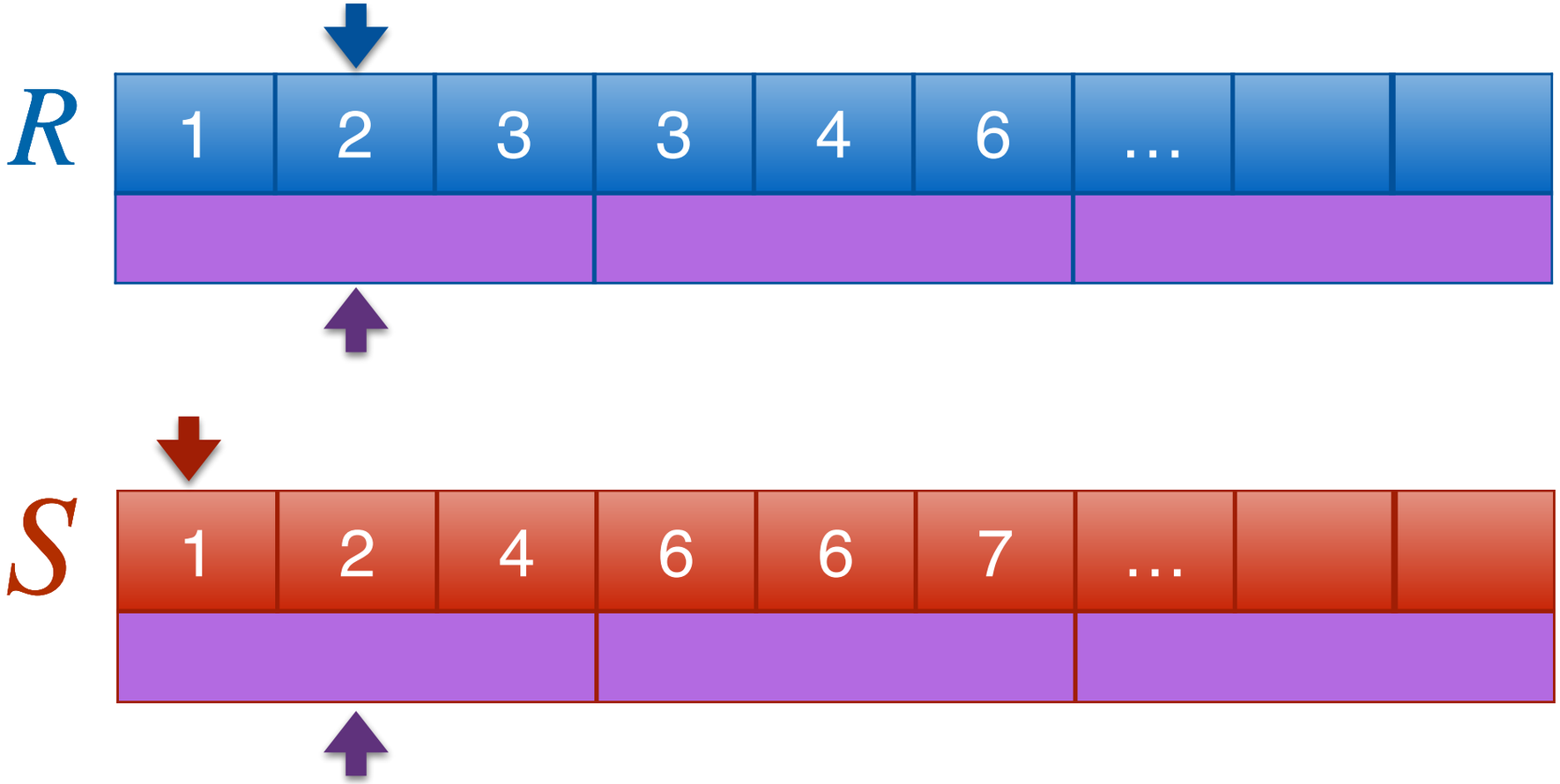
- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



# Sort-merge-join

$R \bowtie S$

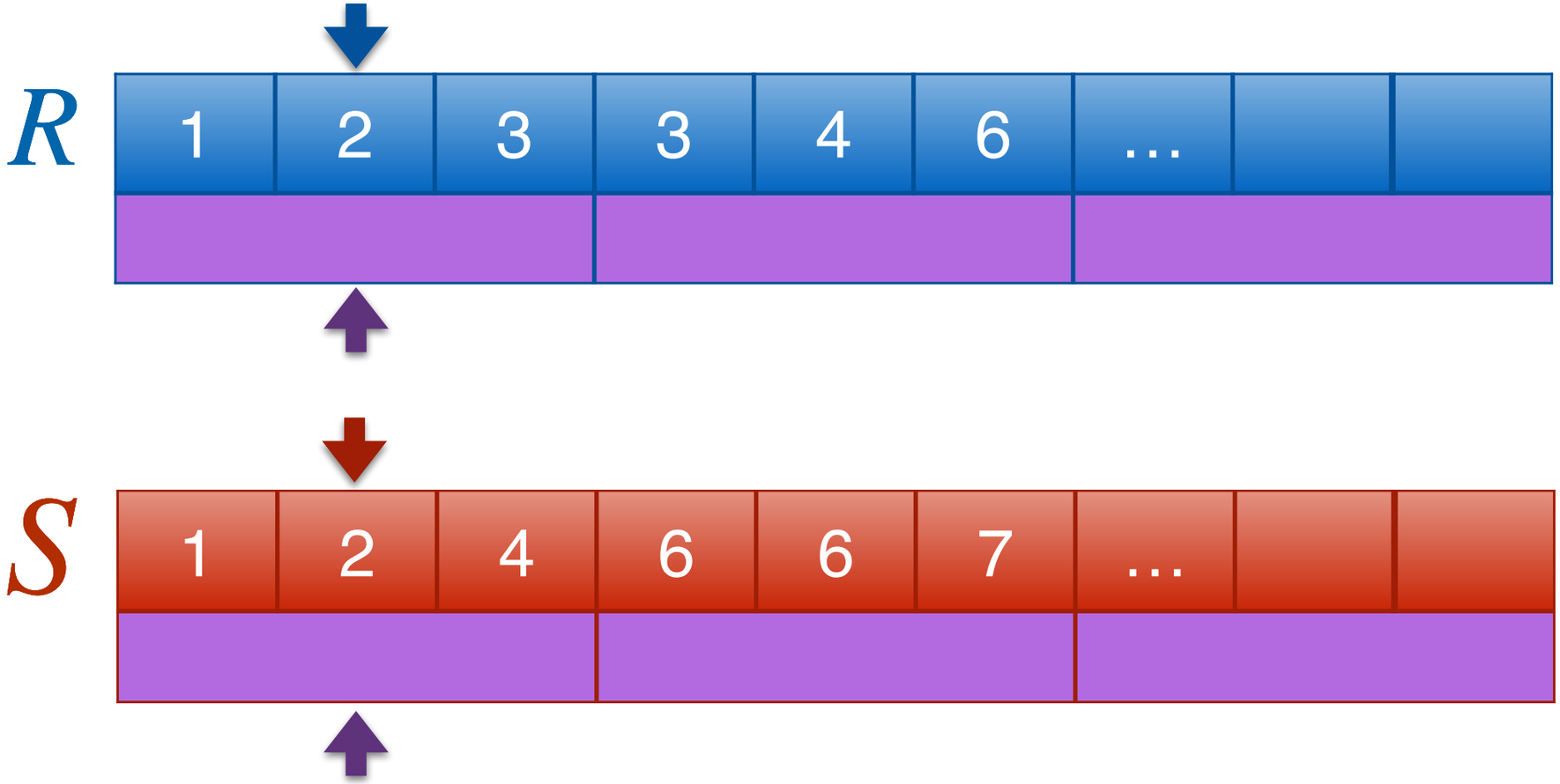
- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$

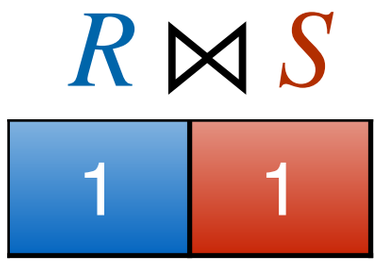


# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$

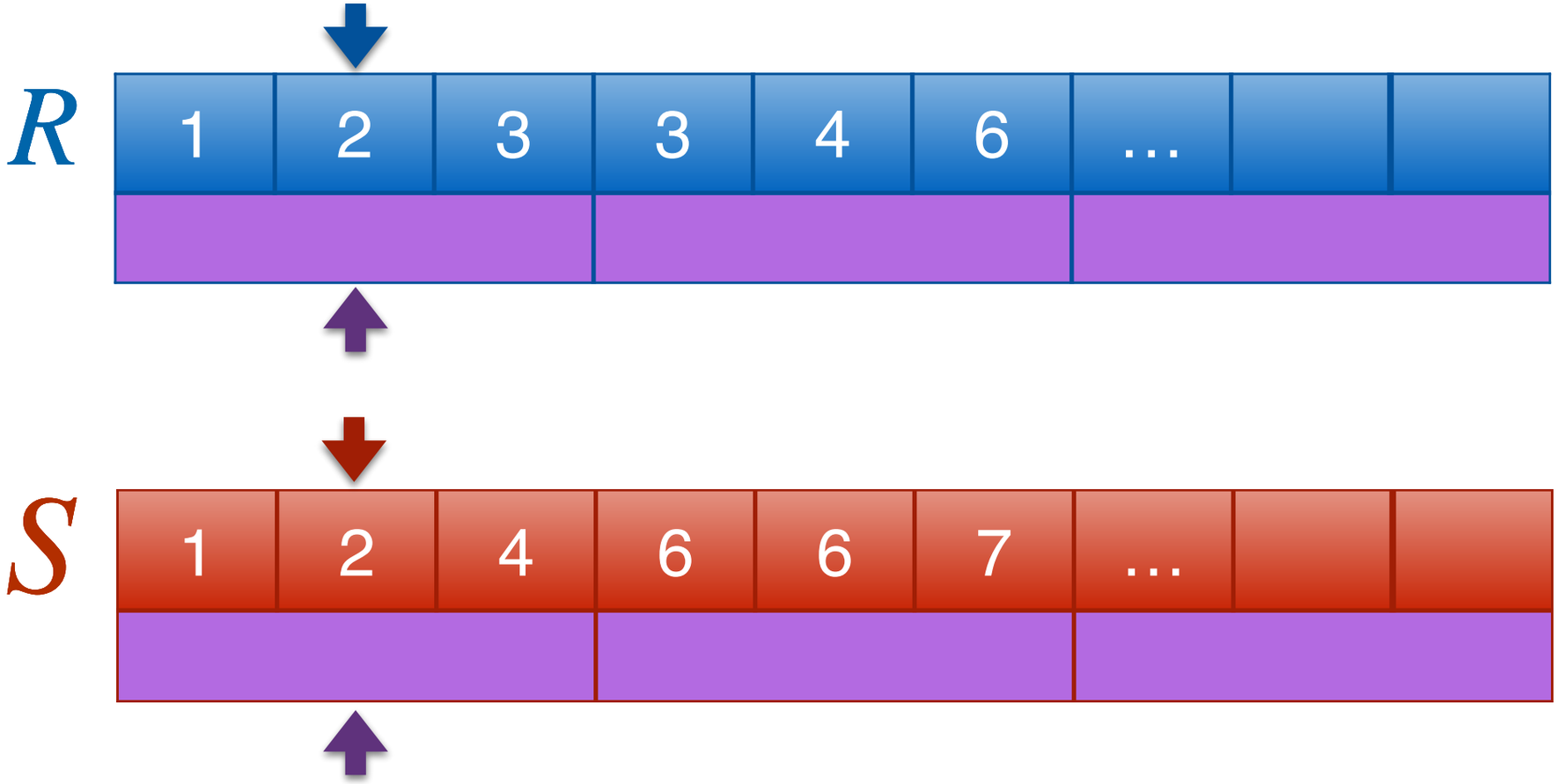


# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



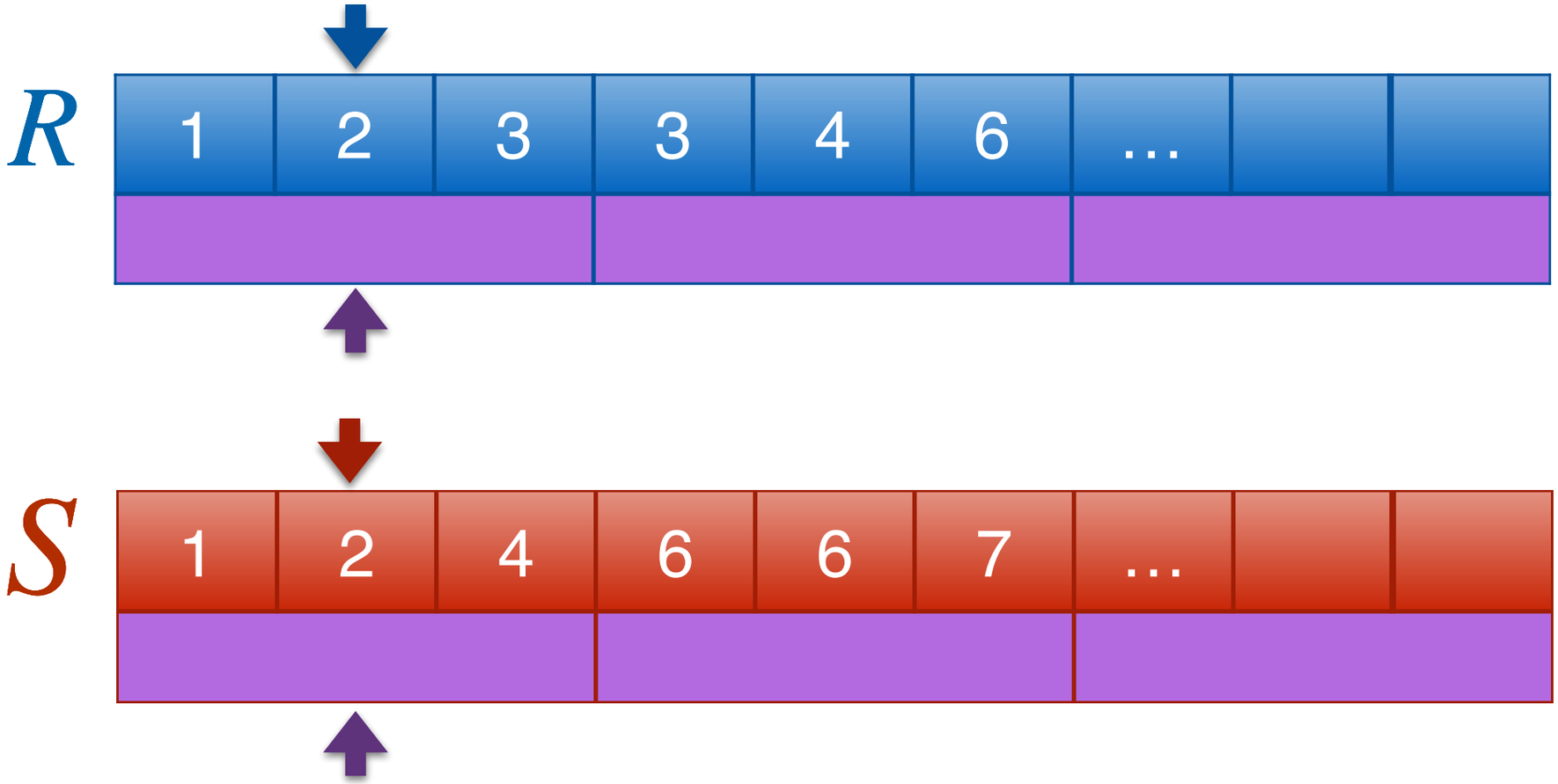

$R \bowtie S$

| | |
|---|---|
| 1 | 1 |
| 2 | 2 |

# Sort-merge-join

$R \bowtie S$

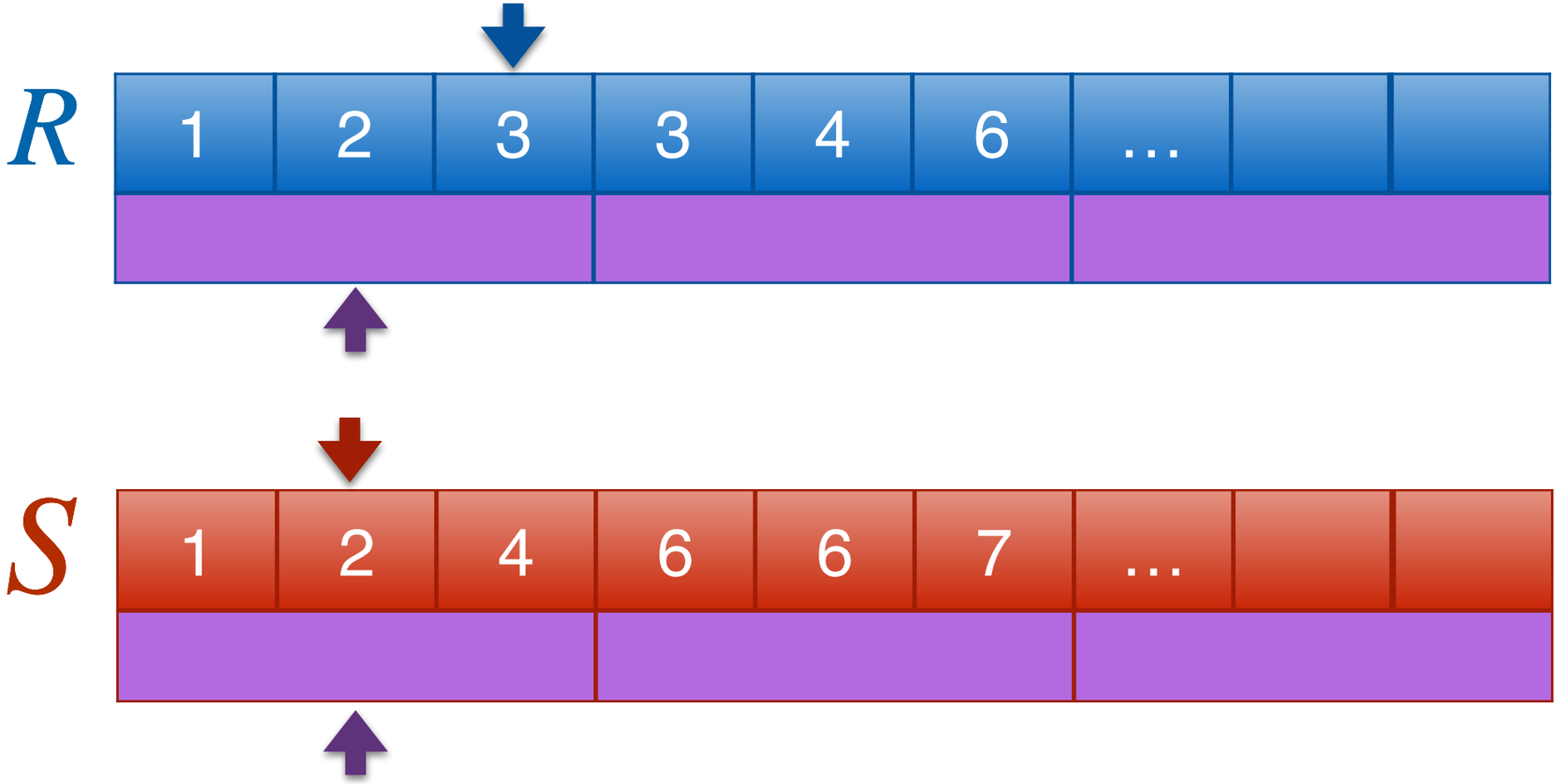
- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



# Sort-merge-join

$R \bowtie S$

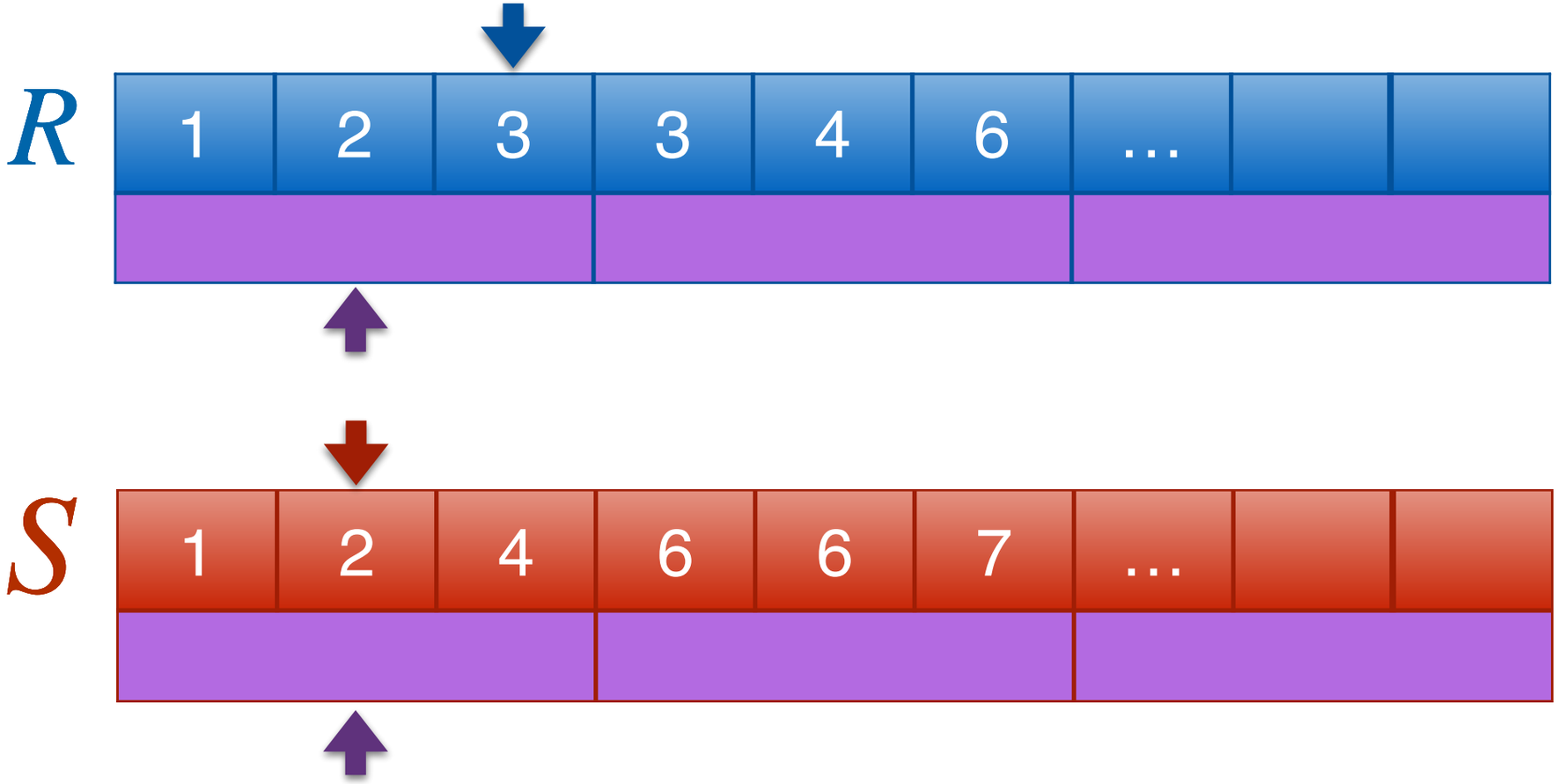
- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



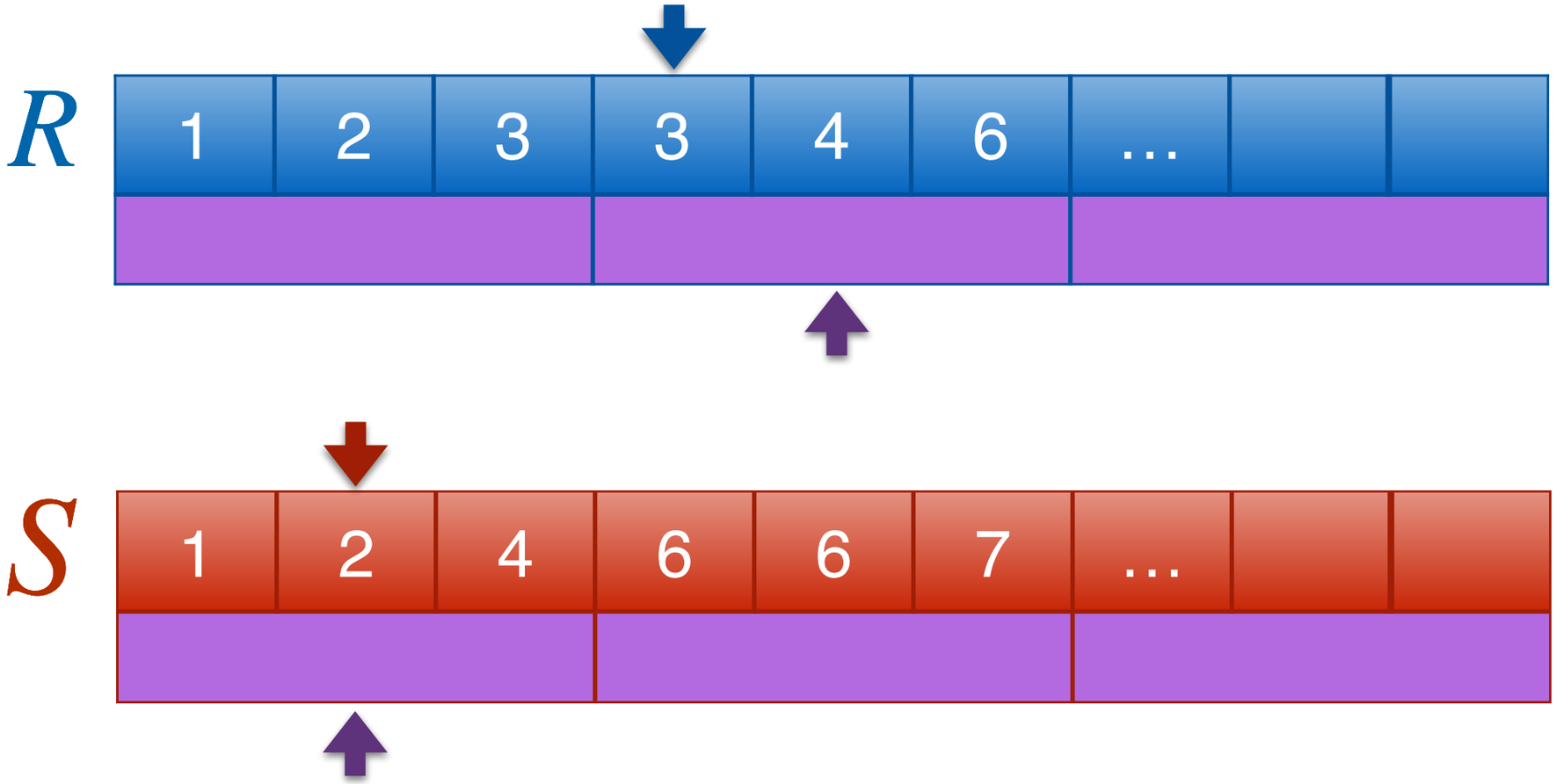

$R \bowtie S$

| | |
|---|---|
| 1 | 1 |
| 2 | 2 |

# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



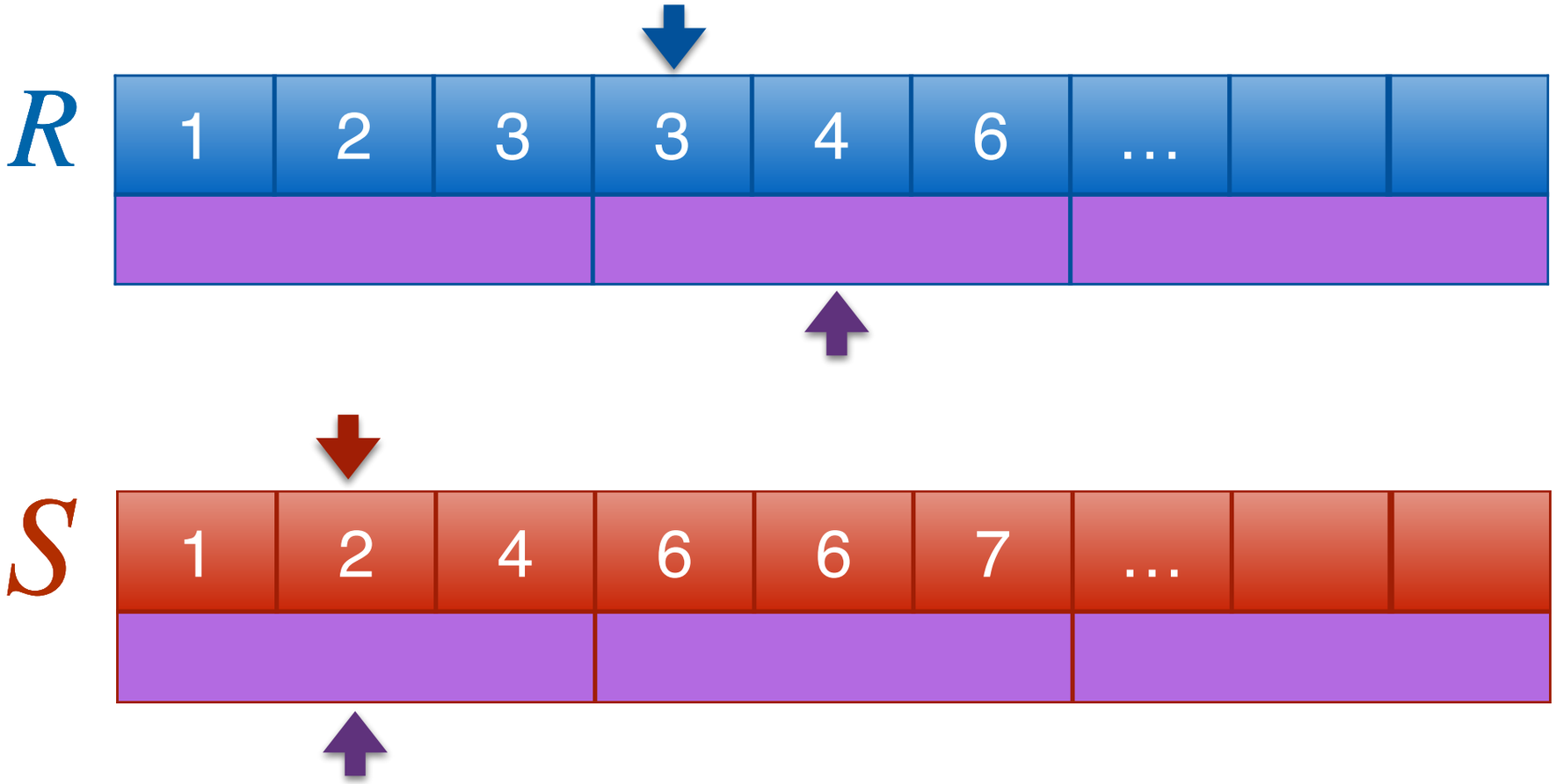

$R \bowtie S$

| | |
|---|---|
| 1 | 1 |
| 2 | 2 |

# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



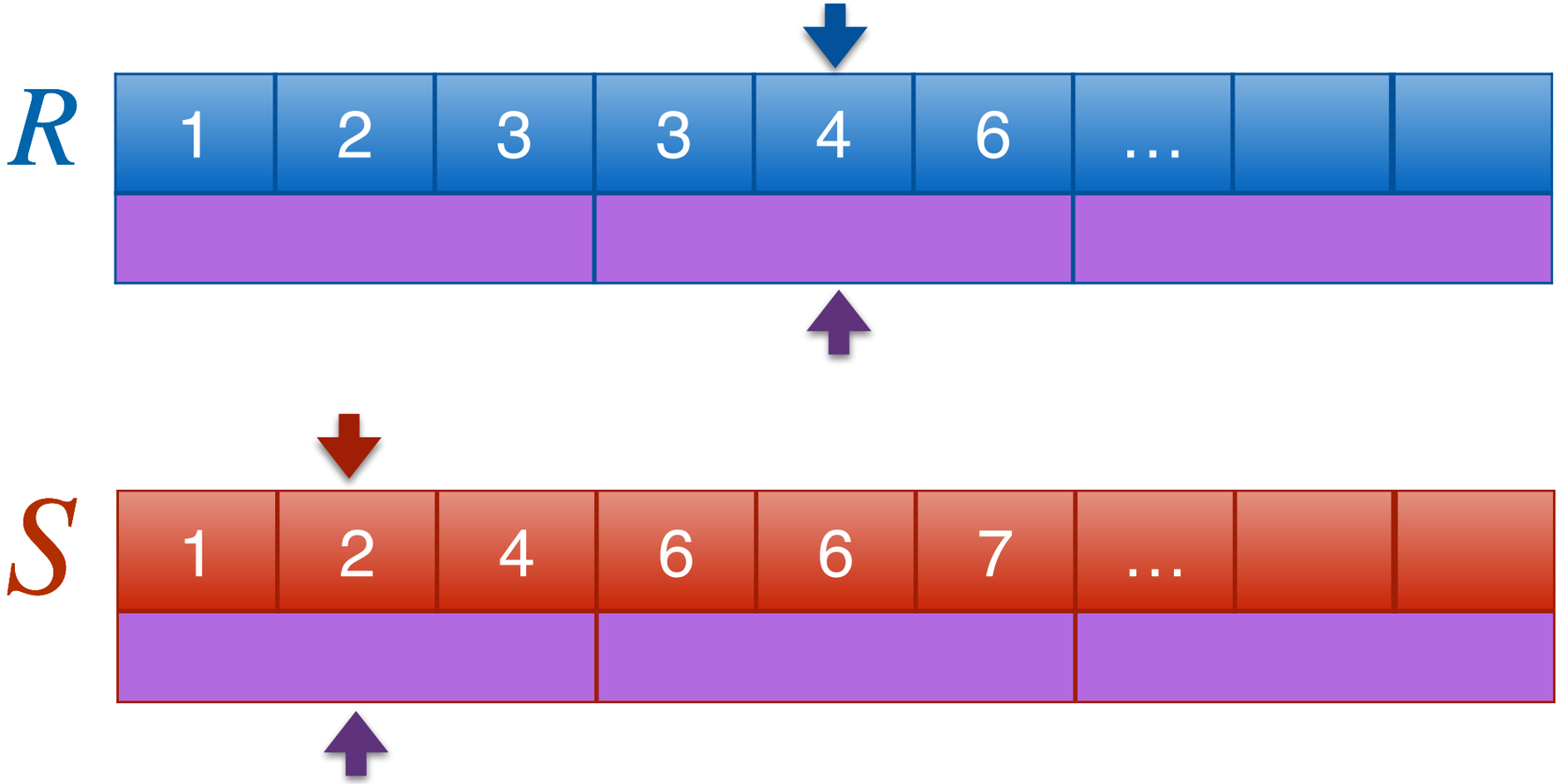

$R \bowtie S$

| | |
|---|---|
| 1 | 1 |
| 2 | 2 |

# Sort-merge-join

$R \bowtie S$

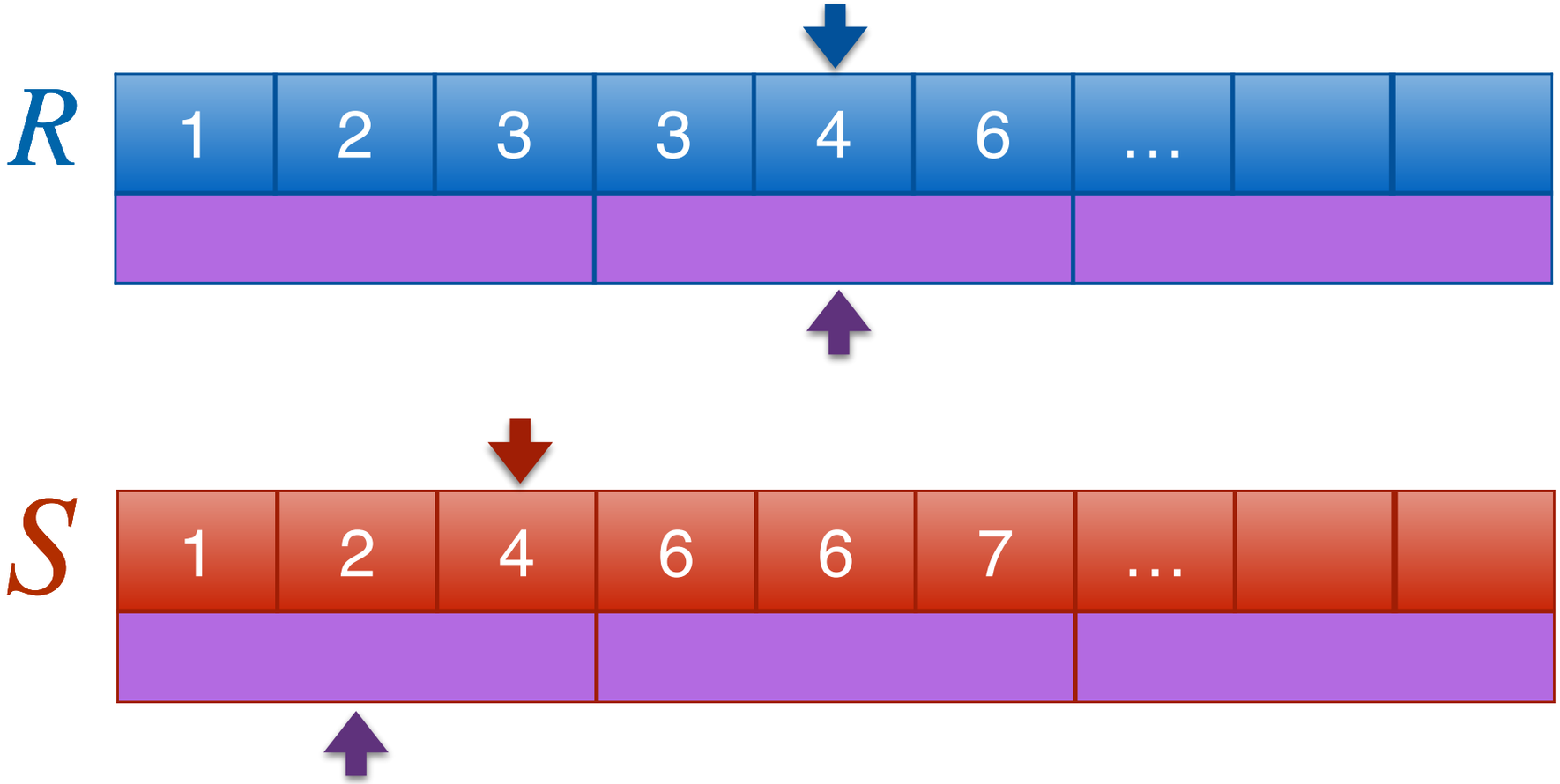
- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



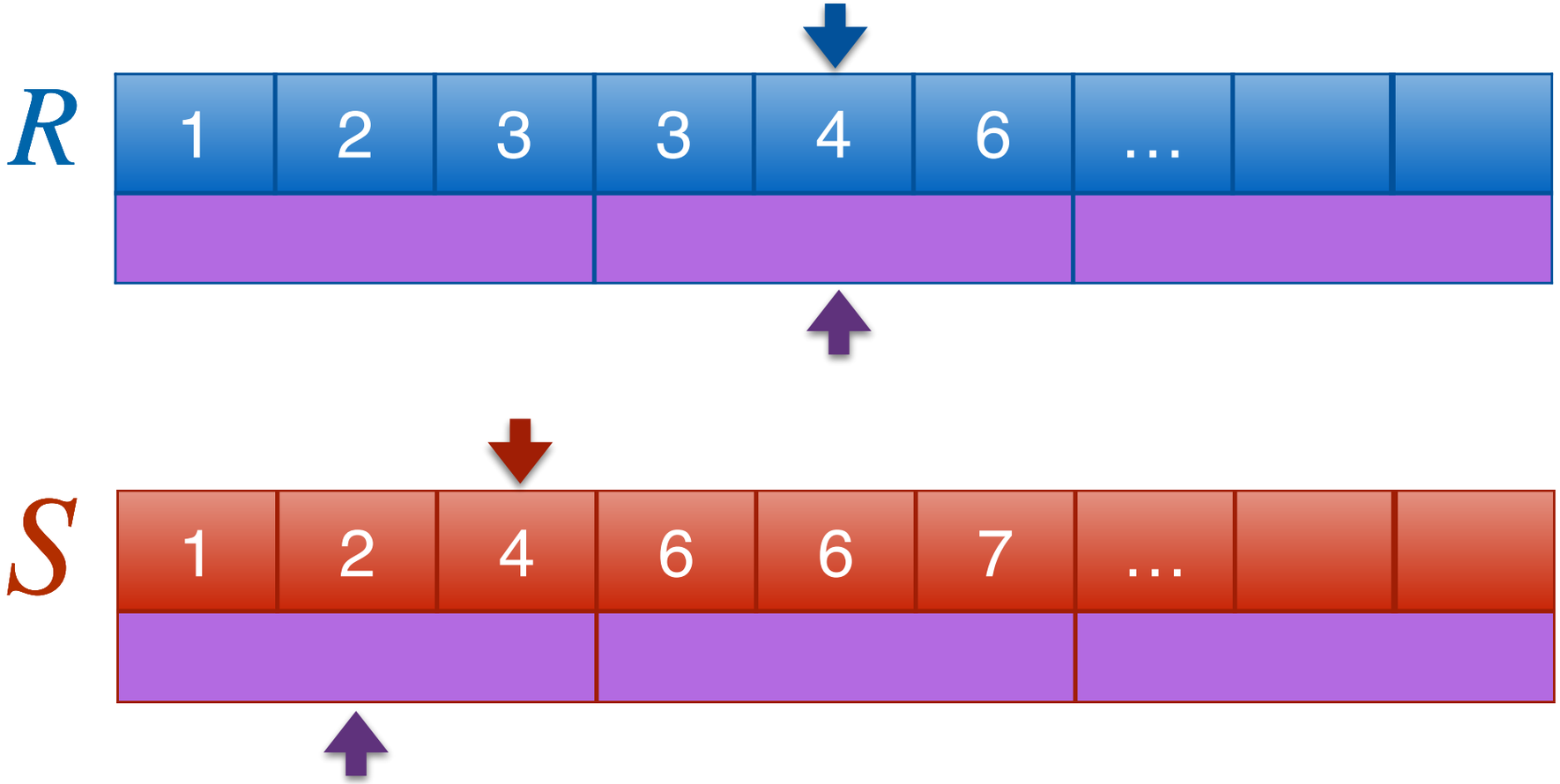

$R \bowtie S$

| | |
|---|---|
| 1 | 1 |
| 2 | 2 |

# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



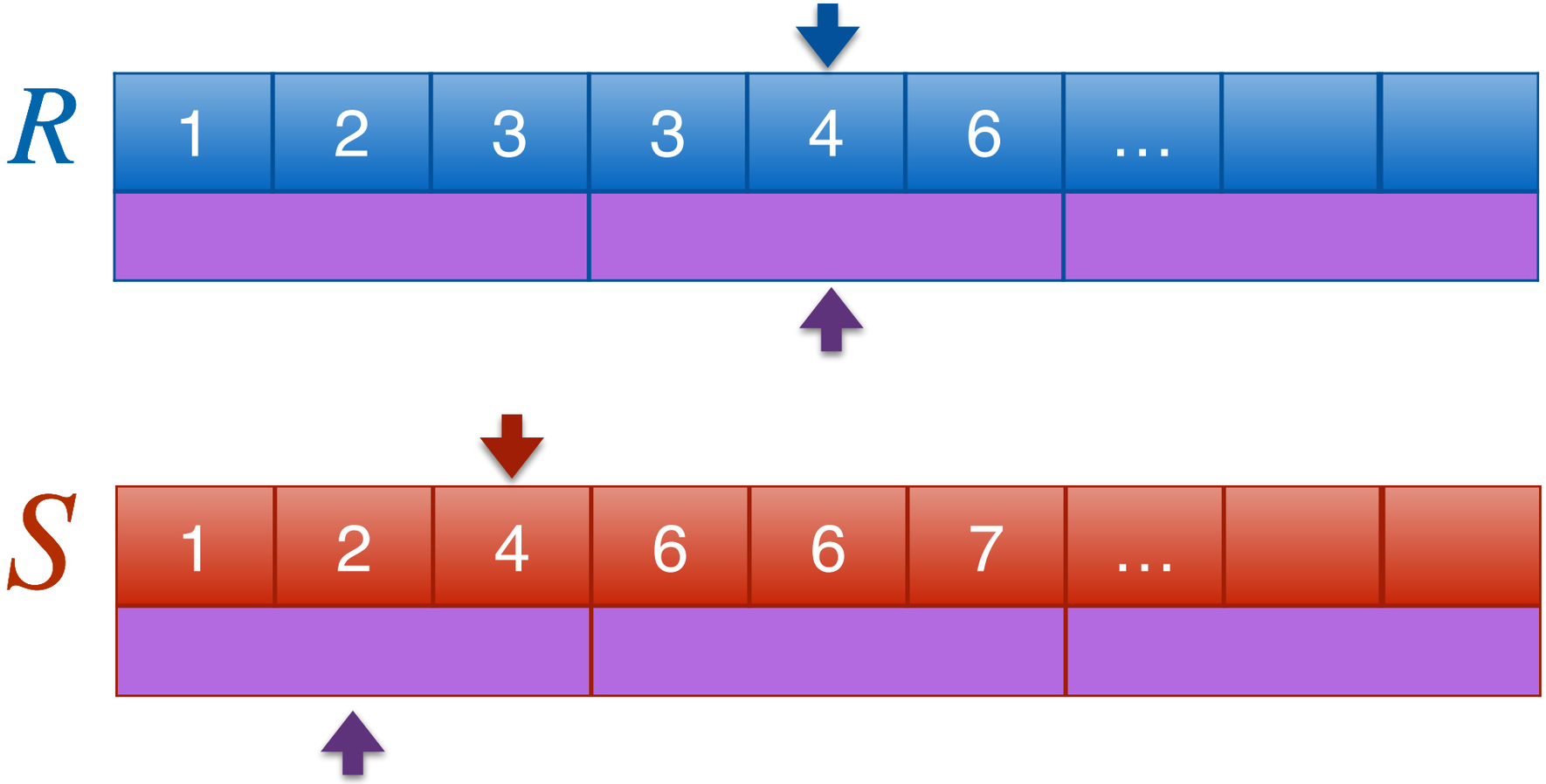

$R \bowtie S$

| | |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 4 | 4 |

# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$



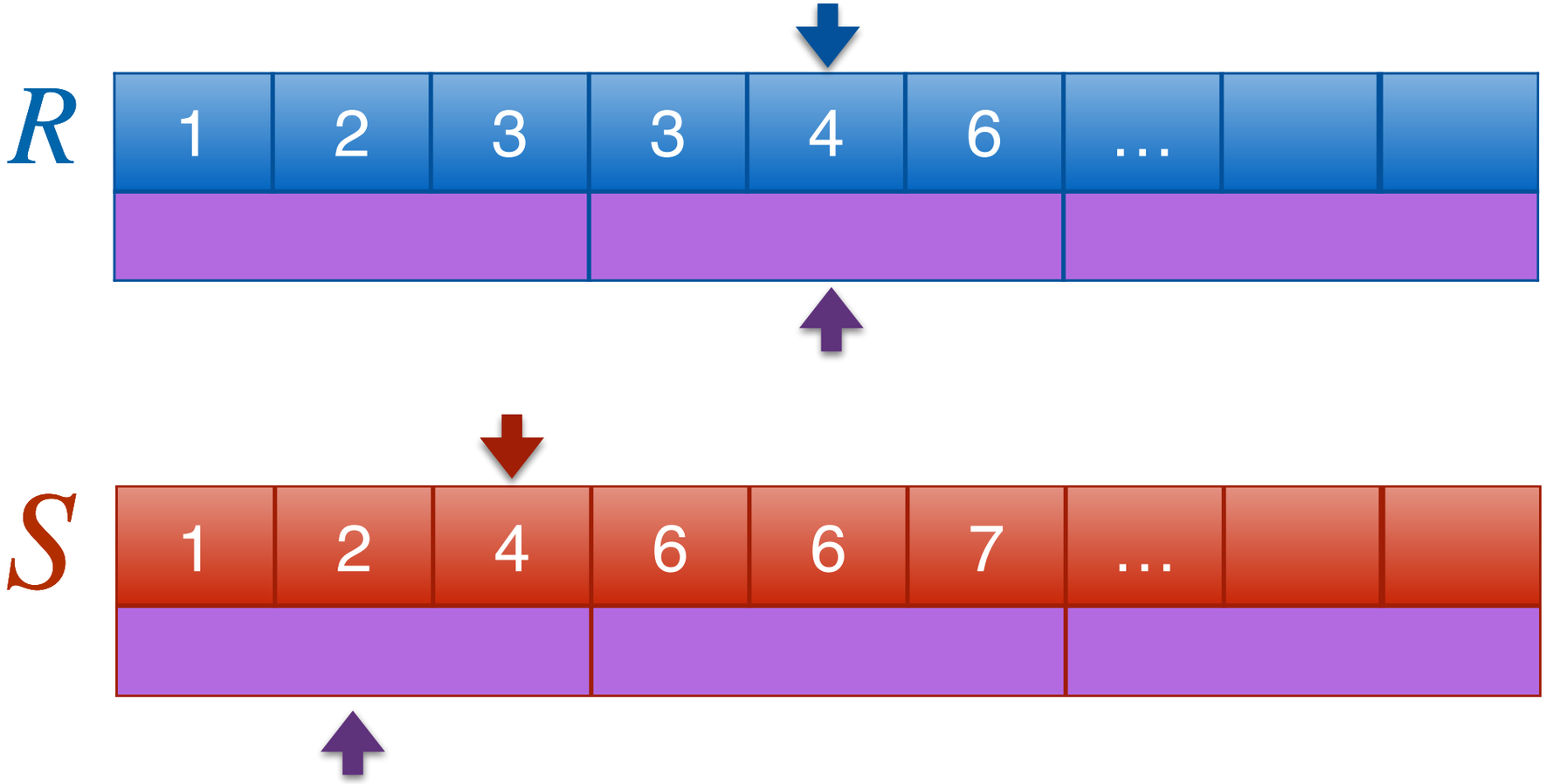

$R \bowtie S$

| | |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 4 | 4 |

# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$




$R \bowtie S$

| | |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 4 | 4 |

...

# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

$$Ord + \left\lceil \frac{|R|}{B} \right\rceil + \left\lceil \frac{|S|}{B} \right\rceil$$

**¿Memoria?**

$2B$ después de ordenación

**¿Elegir $R$ y $S$?**

No importa

# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

$$Ord + \left\lceil \frac{|R|}{B} \right\rceil + \left\lceil \frac{|S|}{B} \right\rceil$$

Costo de ordenar

**¿Memoria?**

$2B$ después de ordenación

**¿Elegir $R$ y $S$?**

No importa

# Sort-merge-join

$R \bowtie S$

- Ordenar $R$ y $S$ por los atributos del join
- Aplicar ***merge-sort*** y para cada tupla $r$ y $s$ que satisfagan el join => escribir $\{r\} \times \{s\}$

**¿Costo?**

$$Ord + \left\lceil \frac{|R|}{B} \right\rceil + \left\lceil \frac{|S|}{B} \right\rceil$$

Costo de ordenar

**¿Memoria?**

$2B$ después de ordenación

**¿Elegir $R$ y $S$?**

No importa

Puede ser que las relaciones ya estén ordenadas por los atributos del join, en cual caso ¡es una buena opción!

# Joins: Comparación

| | Costo | Memoria | ¿R y S? |
|---|---|---|---|
| **Loop anidado (sin índice)** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \lvert R \rvert \cdot \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $2B$ | $\lvert R \rvert < \lvert S \rvert$ |
| **Loop anidado (con índice)** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \lvert R \rvert \cdot \beta(S)$ | $2B$ | $\lvert R \rvert < \lvert S \rvert$ |
| **Hash-join** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $\lvert S \rvert + B$ | $\lvert S \rvert < \lvert R \rvert$ |
| **Sort-merge-join** | $Ord + \left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $2B$ | No importa |

# Joins: Comparación

| | Costo | Memoria | ¿R y S? |
|---|---|---|---|
| **Loop anidado (sin índice)** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \lvert R \rvert \cdot \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $2B$ | $\lvert R \rvert < \lvert S \rvert$ |
| **Loop anidado (con índice)** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \lvert R \rvert \cdot \beta(S)$ | $2B$ | $\lvert R \rvert < \lvert S \rvert$ |
| **Hash-join** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $\lvert S \rvert + B$ | $\lvert S \rvert < \lvert R \rvert$ |
| **Sort-merge-join** | $Ord + \left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $2B$ | No importa |

- Loop anidado (sin índice):
  - Nunca es bueno (pero a veces es lo mejor)
- Loop anidado (con índice):

# Joins: Comparación

| | Costo | Memoria | ¿R y S? |
|---|---|---|---|
| **Loop anidado (sin índice)** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \lvert R \rvert \cdot \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $2B$ | $\lvert R \rvert < \lvert S \rvert$ |
| **Loop anidado (con índice)** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \lvert R \rvert \cdot \beta(S)$ | $2B$ | $\lvert R \rvert < \lvert S \rvert$ |
| **Hash-join** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $\lvert S \rvert + B$ | $\lvert S \rvert < \lvert R \rvert$ |
| **Sort-merge-join** | $Ord + \left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $2B$ | No importa |

- Loop anidado (sin índice):
  - Nunca es bueno (pero a veces es lo mejor)
- Loop anidado (con índice):
  - Cuando el índice esté disponible y:
    - Pocas tuplas en $S$ satisfagan el join
- *Hash-join*

# Joins: Comparación

| | Costo | Memoria | ¿R y S? |
|---|---|---|---|
| **Loop anidado (sin índice)** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \lvert R \rvert \cdot \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $2B$ | $\lvert R \rvert < \lvert S \rvert$ |
| **Loop anidado (con índice)** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \lvert R \rvert \cdot \beta(S)$ | $2B$ | $\lvert R \rvert < \lvert S \rvert$ |
| **Hash-join** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $\lvert S \rvert + B$ | $\lvert S \rvert < \lvert R \rvert$ |
| **Sort-merge-join** | $Ord + \left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $2B$ | No importa |

- Loop anidado (sin índice):
  - Nunca es bueno (pero a veces es lo mejor)
- Loop anidado (con índice):
  - Cuando el índice esté disponible y:
    - Pocas tuplas en $S$ satisfagan el join
- *Hash-join*
  - Cuando $S$ quepa en memoria y:
    - Muchas tuplas en $R$ satisfagan el join
- *Sort-merge-join*

# Joins: Comparación

| | Costo | Memoria | ¿R y S? |
|---|---|---|---|
| **Loop anidado (sin índice)** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \lvert R \rvert \cdot \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $2B$ | $\lvert R \rvert < \lvert S \rvert$ |
| **Loop anidado (con índice)** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \lvert R \rvert \cdot \beta(S)$ | $2B$ | $\lvert R \rvert < \lvert S \rvert$ |
| **Hash-join** | $\left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $\lvert S \rvert + B$ | $\lvert S \rvert < \lvert R \rvert$ |
| **Sort-merge-join** | $Ord + \left\lceil \frac{\lvert R \rvert}{B} \right\rceil + \left\lceil \frac{\lvert S \rvert}{B} \right\rceil$ | $2B$ | No importa |

- Loop anidado (sin índice):
  - Nunca es bueno (pero a veces es lo mejor)
- Loop anidado (con índice):
  - Cuando el índice esté disponible y:
    - Pocas tuplas en $S$ satisfagan el join
- *Hash-join*
  - Cuando $S$ quepa en memoria y:
    - Muchas tuplas en $R$ satisfagan el join
- *Sort-merge-join*
  - Cuando $R$ y $S$ ya estén ordenadas por los atributos del join y:
    - Muchas tuplas en $R$ y $S$ satisfagan el join

# Estadísticas y Catálogos

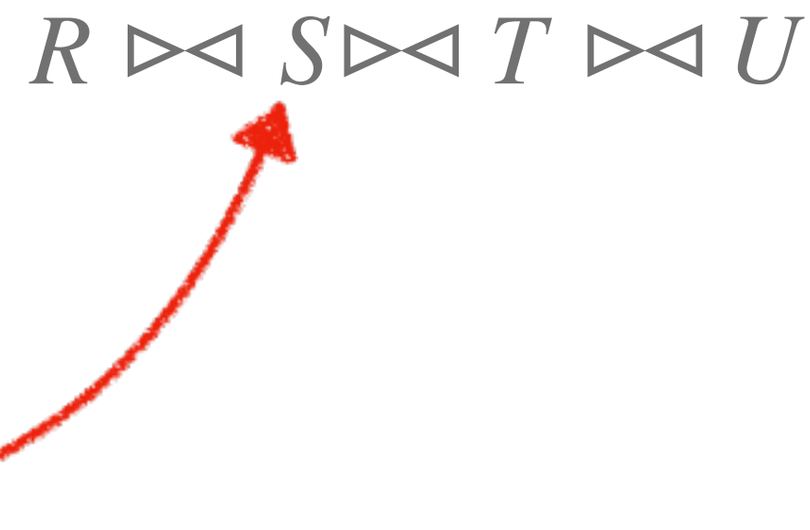
- Se requiere información sobre las tablas y los índices relevantes para la consulta. El ***catálogo*** tipicamente contiene al menos:
  - # tuples (NTuples) y # pages (NPages) para cada tabla.
  - # distinct key values (NKeys) y Npages para cada índice.
  - Index height, low/high key values (Low/High) para cada índice.
- Los catálogos se actualizan periódicamente.
  - Actualizarlos cada vez que cambian los datos es muy caro; hay mucha aproximación, luego ligeras inconsistencias.
- A veces se guarda información detallada (e.g., histogramas de valores en algún atributo) .

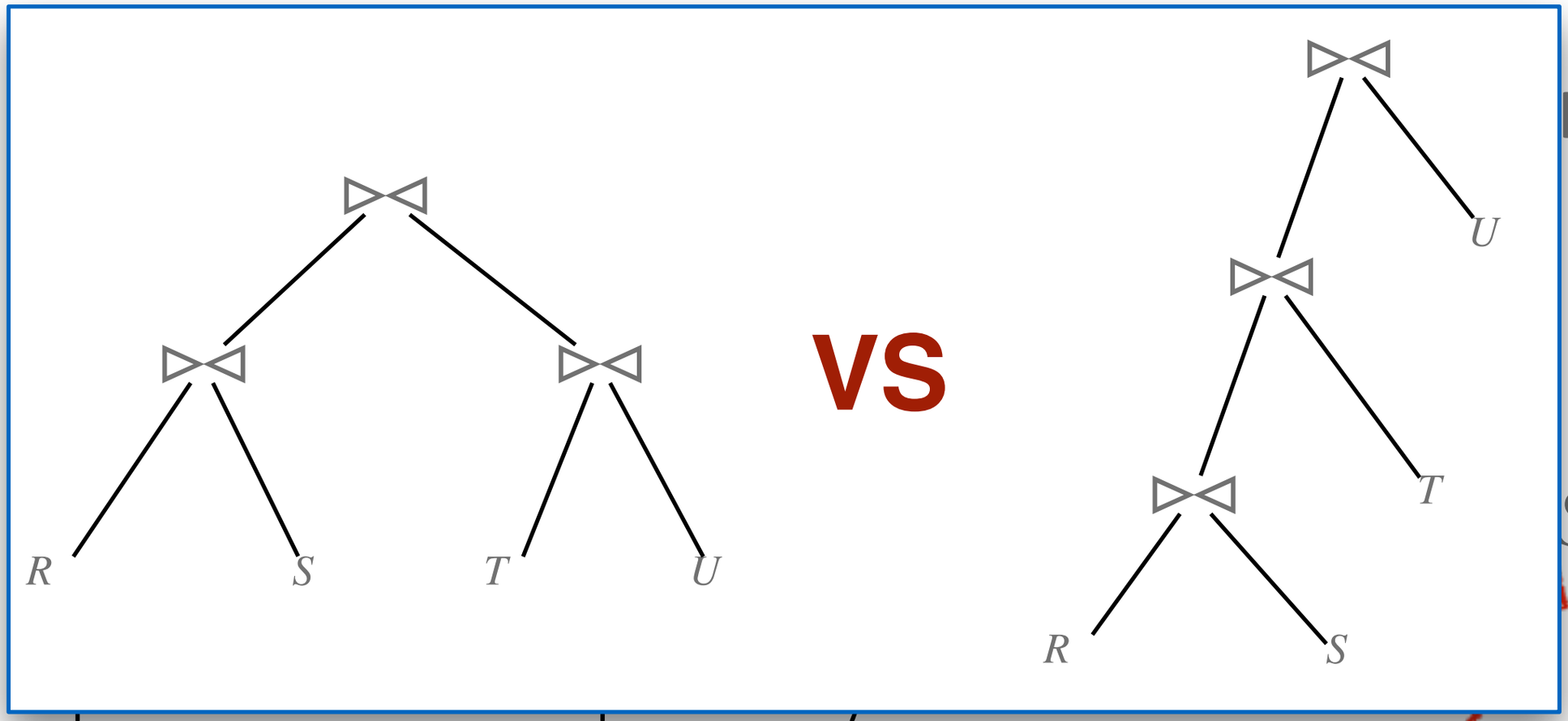
# Esquema del optimizador del Sistema R

- Impacto:
  - El más usado actualmente;
  - funciona bien para  < 10 joins.
- Estimación de Costo :  Buena aproximación.
  - Estadísticas, mantenidas en el catálogo del sistema, se usa para estimar costos de operaciones y tamaños de resultados.
  - Considera una combinación de costos de CPU y I/O.
- Planes:  Todos los posibles planes son muchos.
  - Solo considera el espacio de  *left-deep plans*.
    - Planes Left-deep permiten  usar *pipeline* para los operadores sin necesidad de almacenarlos temporalmente.
  - Se evitan los productos cartesianos.

# Esquema del optimizador del Sistema R

- Impacto:
  - El más usado actualmente;
  - funciona bien para  < 10 joins.
- Estimación de Costo :  Buena aproximación.
  - Estadísticas, mantenidas en el catálogo del sistema, se usa para estimar costos de operaciones y tamaños de resultados.
  - Considera una combinación de costos de CPU y I/O.
- Planes:  Todos los posibles planes son muchos.
  - Solo considera el espacio de  *left-deep plans*.
    - Planes Left-deep permiten  usar *pipeline* para los operadores sin necesidad de almacenarlos temporalmente.
  - Se evitan los productos cartesianos.






ma R

- $S \bowtie T \bowtie U$

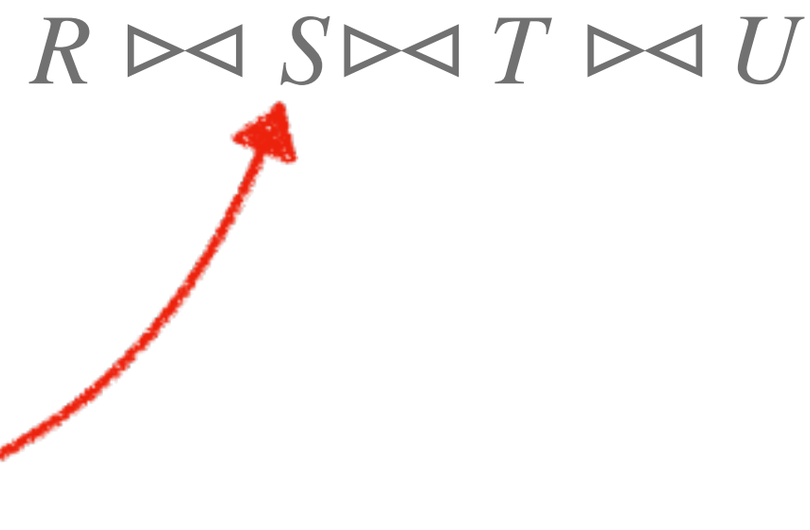
resultados.

– Considera una combinación de costos de CPU y I/O.

- Planes: Todos los posibles planes son muchos.
  - Solo considera el espacio de *left-deep plans*.
    - Planes Left-deep permiten usar *pipeline* para los operadores sin necesidad de almacenarlos temporalmente.
  - Se evitan los productos cartesianos.

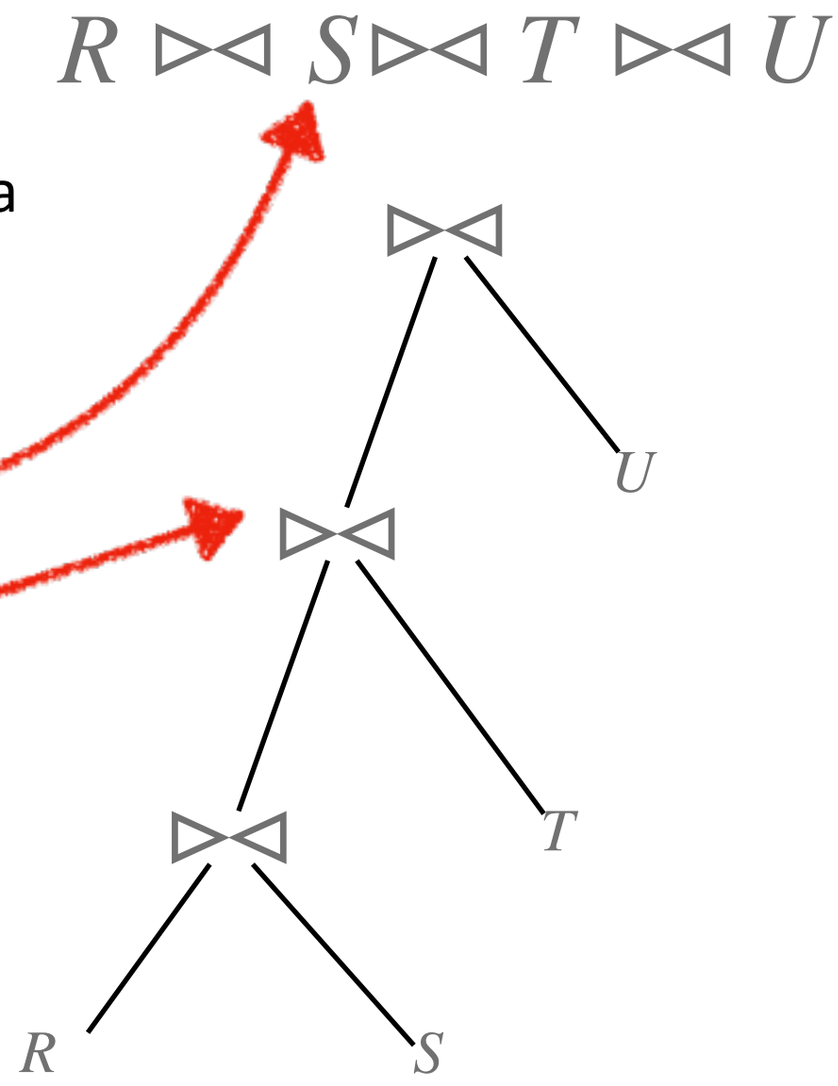
# Esquema del optimizador del Sistema R

- Impacto:
  - El más usado actualmente;
  - funciona bien para  < 10 joins.
- Estimación de Costo :  Buena aproximación.
  - Estadísticas, mantenidas en el catálogo del sistema, se usa para estimar costos de operaciones y tamaños de resultados.
  - Considera una combinación de costos de CPU y I/O.
- Planes:  Todos los posibles planes son muchos.
  - Solo considera el espacio de  *left-deep plans*.
    - Planes Left-deep permiten  usar *pipeline* para los operadores sin necesidad de almacenarlos temporalmente.
  - Se evitan los productos cartesianos.



# Esquema del optimizador del Sistema R

- Impacto:
  - El más usado actualmente;
  - funciona bien para  < 10 joins.
- Estimación de Costo :  Buena aproximación.
  - Estadísticas, mantenidas en el catálogo del sistema, se usa para estimar costos de operaciones y tamaños de resultados.
  - Considera una combinación de costos de CPU y I/O.
- Planes:  Todos los posibles planes son muchos.
  - Solo considera el espacio de  *left-deep plans*.
    - Planes Left-deep permiten  usar *pipeline* para los operadores sin necesidad de almacenarlos temporalmente.
  - Se evitan los productos cartesianos.



# Estimación de costos

- Para cada plan considerado, se debe estimar:
  - costo estimado de cada operación en el plan.
    - Depende del tamaño de tablas de entrada.
    - Ya vimos como estimar costos de operaciones (sequential scan, index scan, joins, etc.)
  - tamaño estimado *del resultado* para cada operación en el plan!
    - Usar información sobre tablas de entrada.
    - Para selecciones y joins, suponemos independencia de predicados (no se afectan mutuamente)

# Esquema para los ejemplos

Sailors(*sid*: integer, *sname*: string, *rating*: integer, *age*: real)

Reserves(*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

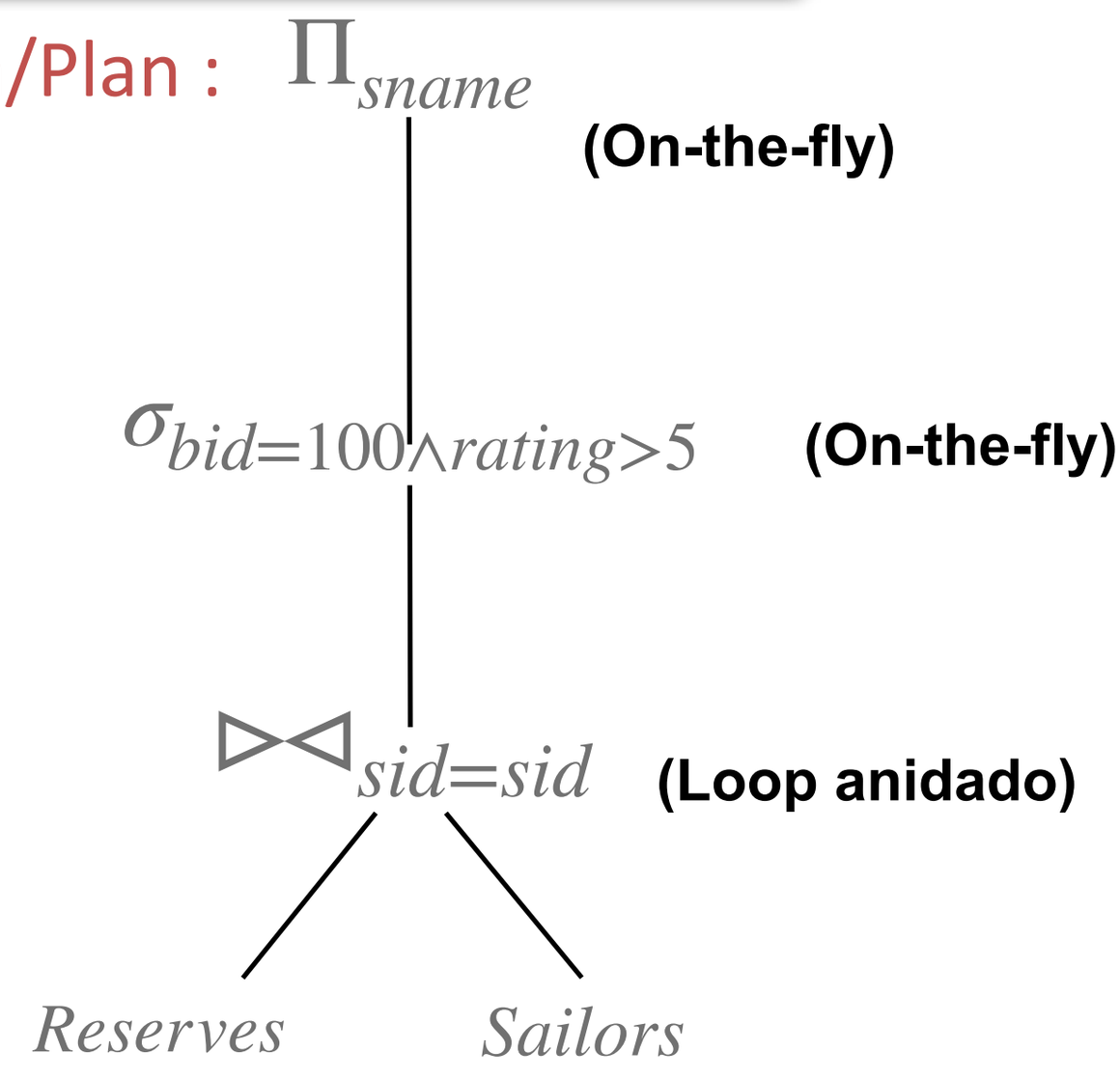
- Similar al esquema anterior; agregamos *rname*.

| | Tamaño tupla (bytes) | Nº tuplas por bloque | Nº bloques |
|---|---|---|---|
| **Reserves** | 40 | 100 | 1000 |
| **Sailors** | 50 | 80 | 500 |

# Ejemplo

```
SELECT sname
FROM Reserves NATURAL JOIN Sailors
WHERE bid = 100 AND rating > 5
```

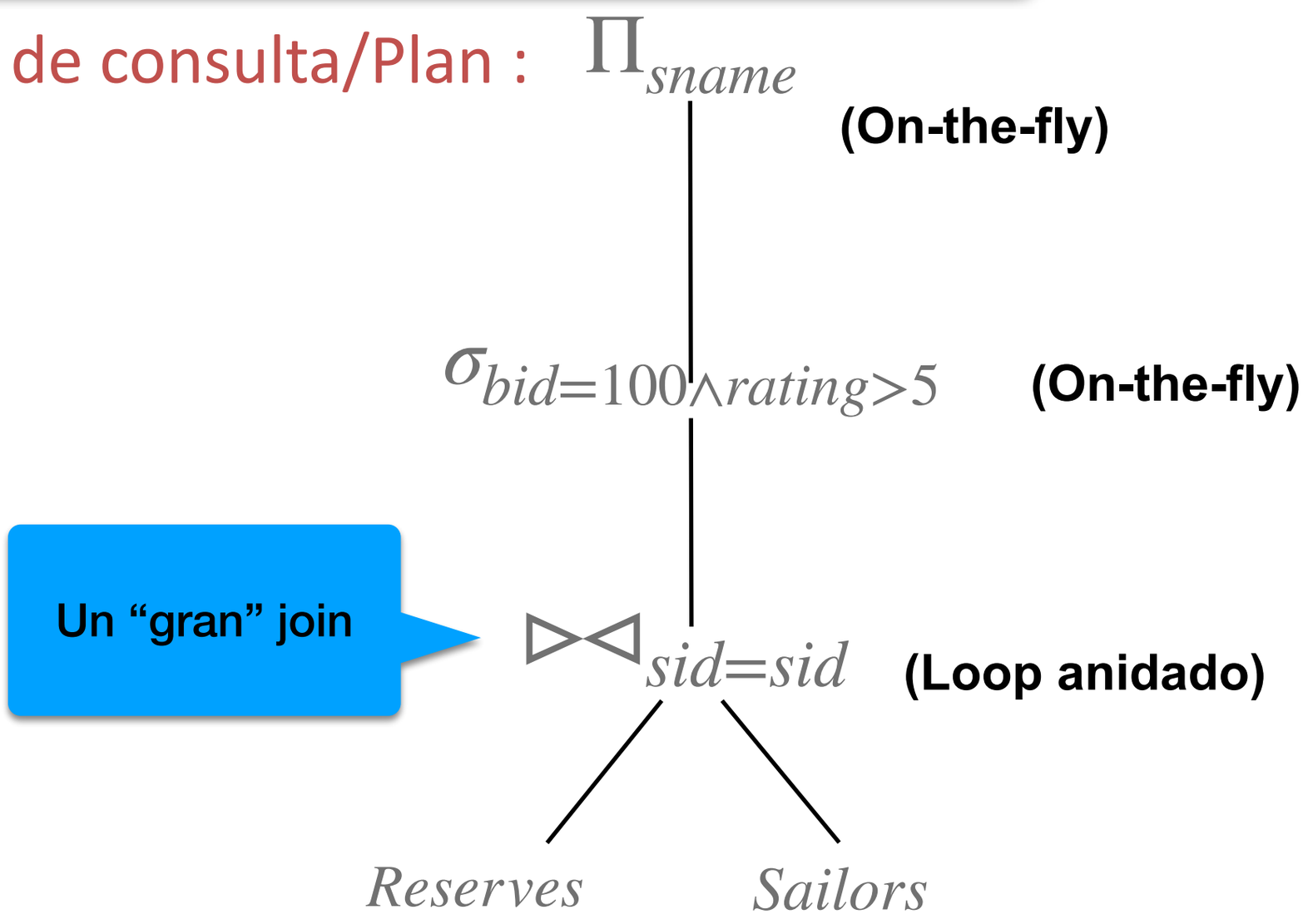
Arbol de consulta/Plan :



# Ejemplo

```
SELECT sname
FROM Reserves NATURAL JOIN Sailors
WHERE bid = 100 AND rating > 5
```

Arbol de consulta/Plan :

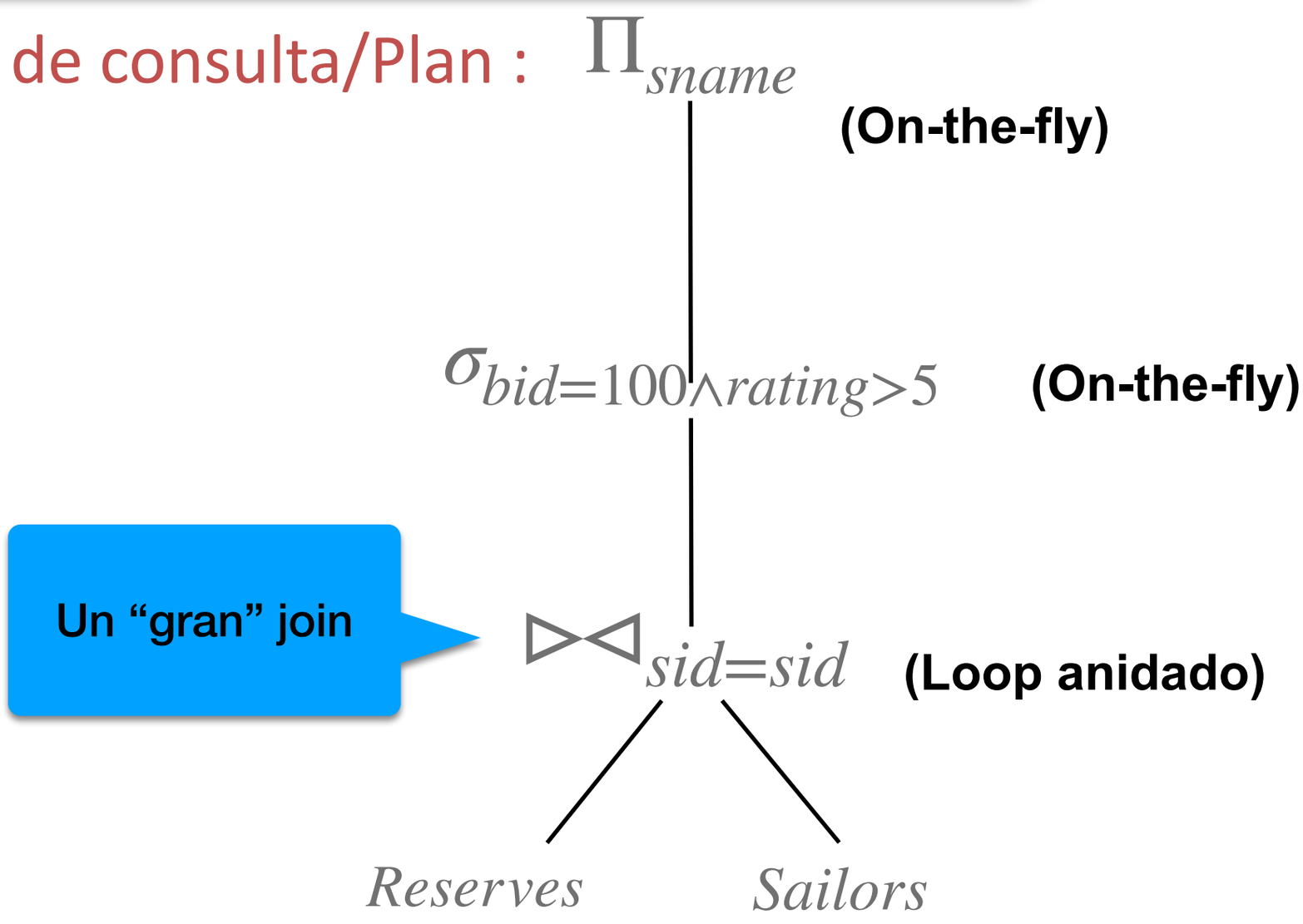


# Ejemplo

```
SELECT sname
FROM Reserves NATURAL JOIN Sailors
WHERE bid = 100 AND rating > 5
```

**Total = 500.500**

Arbol de consulta/Plan :



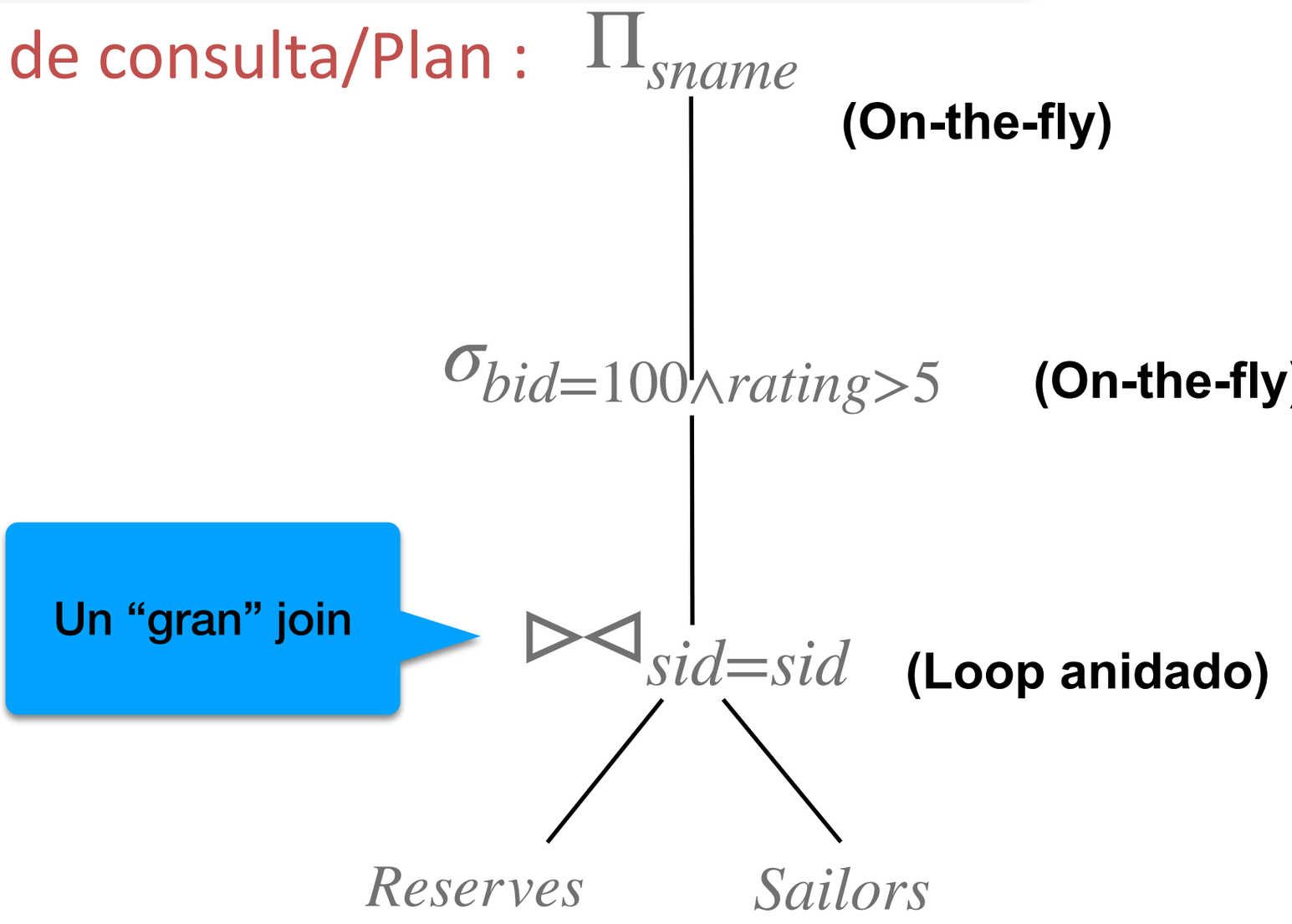

Un “gran” join

# Ejemplo

```
SELECT sname
FROM Reserves NATURAL JOIN Sailors
WHERE bid = 100 AND rating > 5
```

**Total = 500.500**

Arbol de consulta/Plan :




**Selecciones podrían haber sido hechas antes, no se usa ningún índices, etc!**

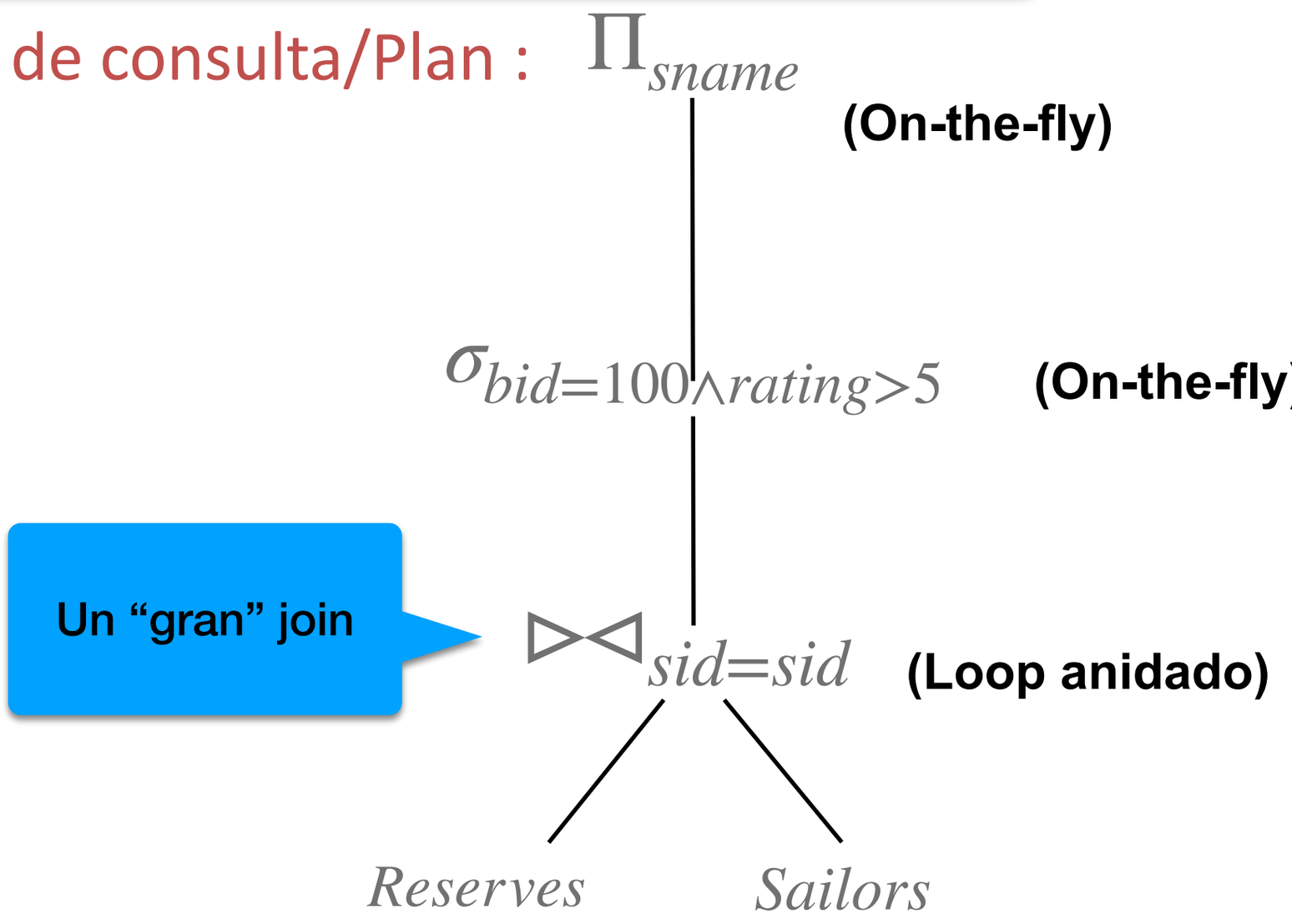
**Un "gran" join**

# Ejemplo

```
SELECT sname
FROM Reserves NATURAL JOIN Sailors
WHERE bid = 100 AND rating > 5
```

Total = 500.500

Arbol de consulta/Plan :

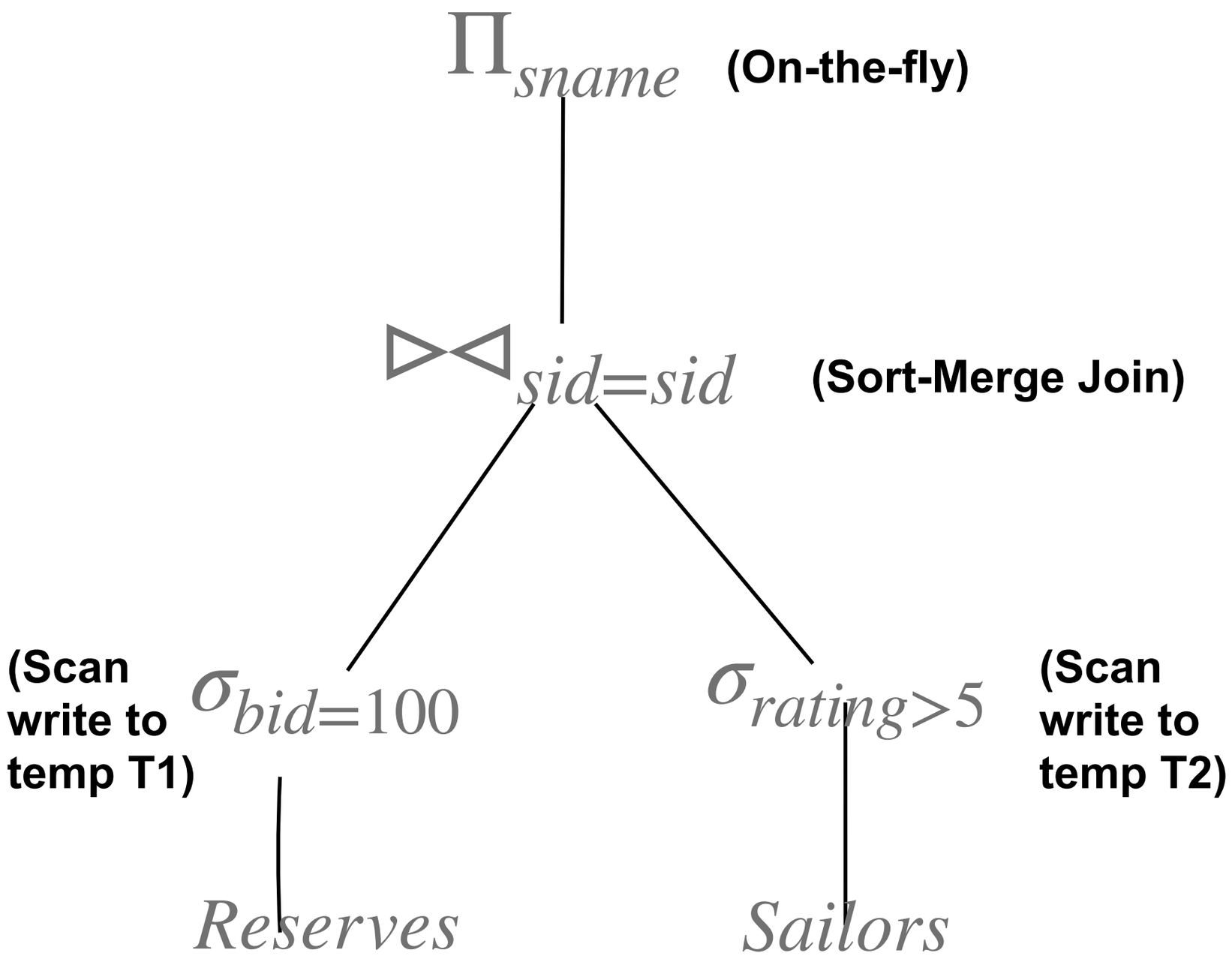



Un “gran” join

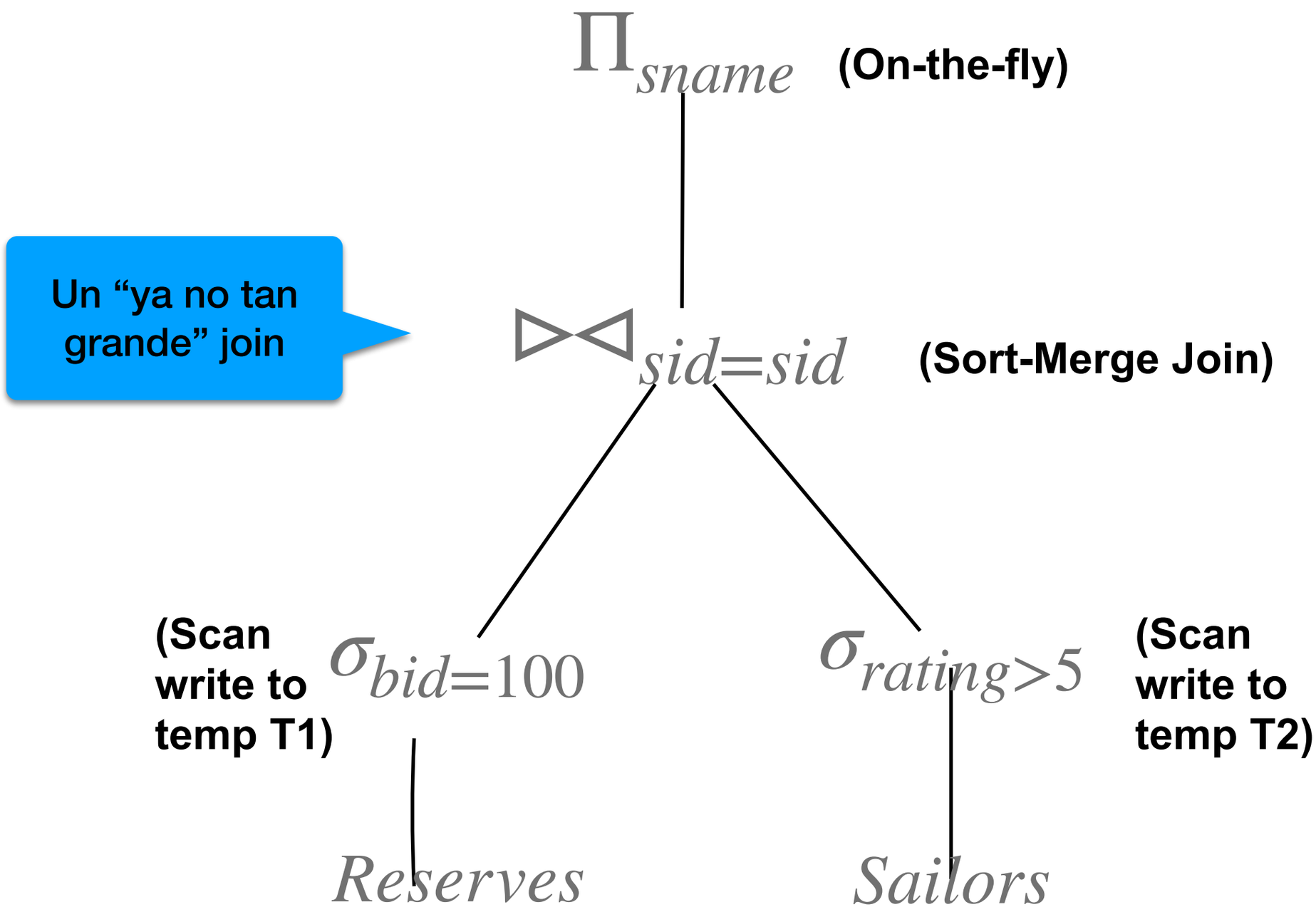
Selecciones podrían haber sido hechas antes, no se usa ningún índices, etc!

Objetivo de la optimización: Encontrar un plan más eficiente que compute la misma respuesta.

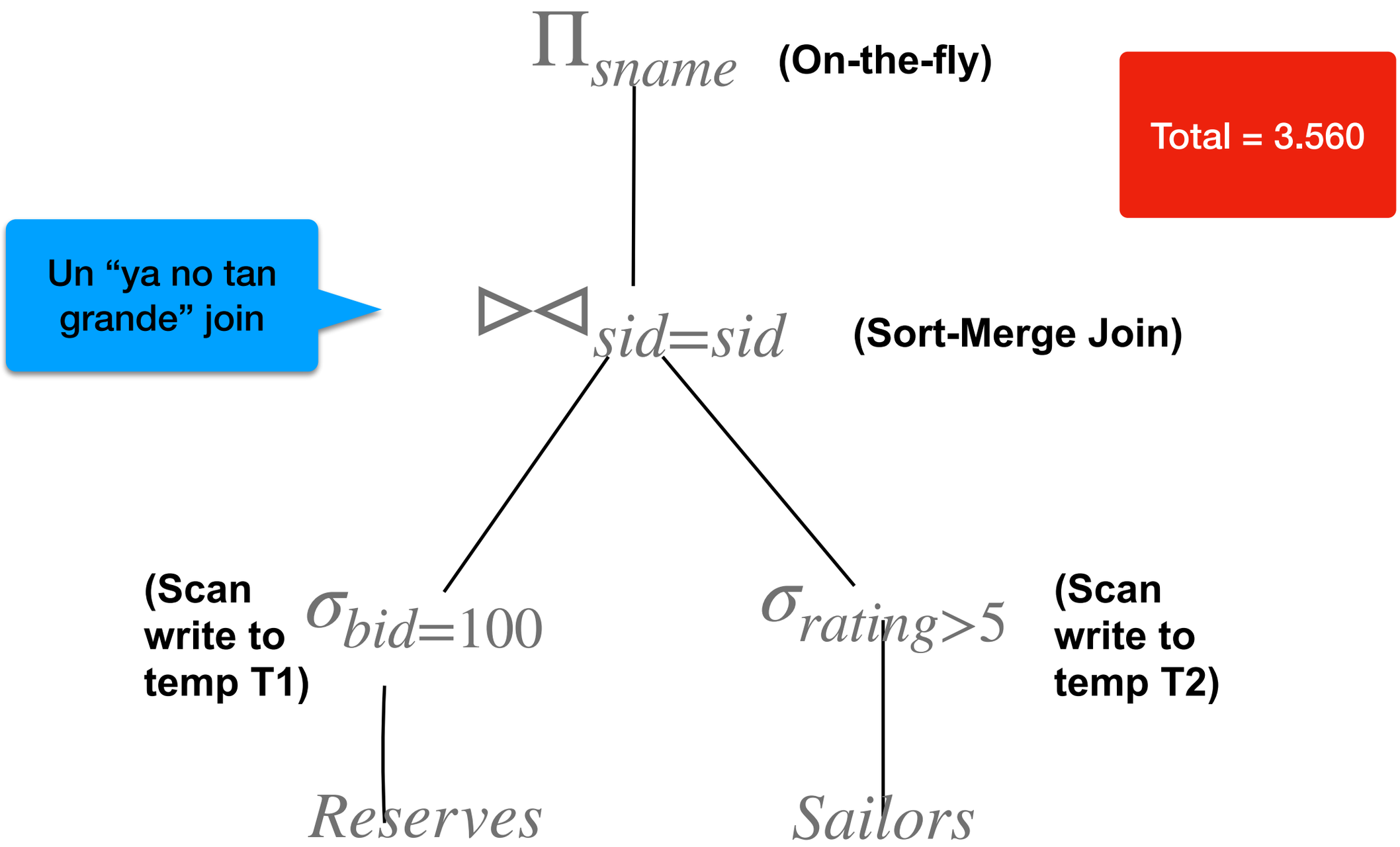
# Plan Alternativo 1 (Sin usar índices)



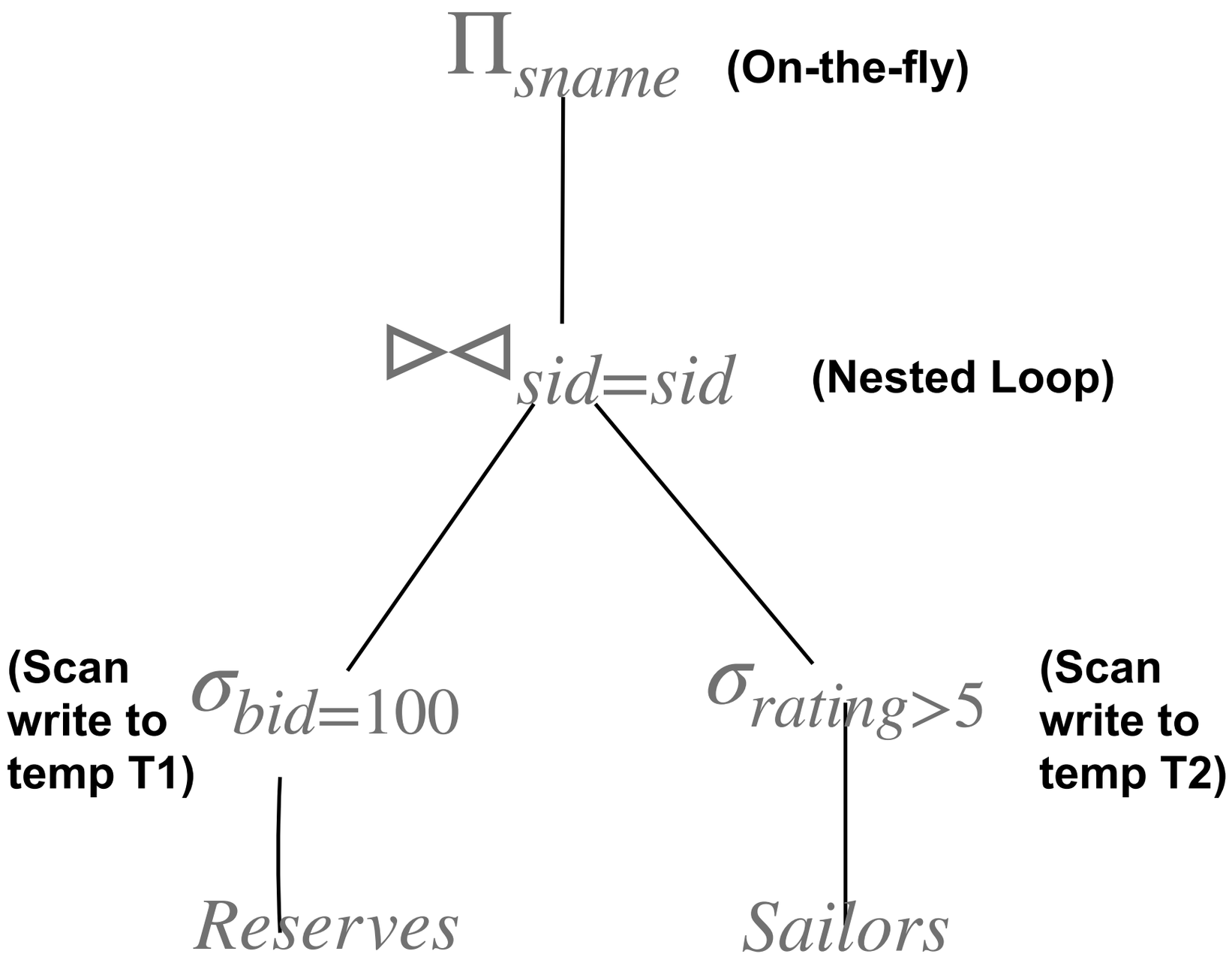
# Plan Alternativo 1 (Sin usar índices)



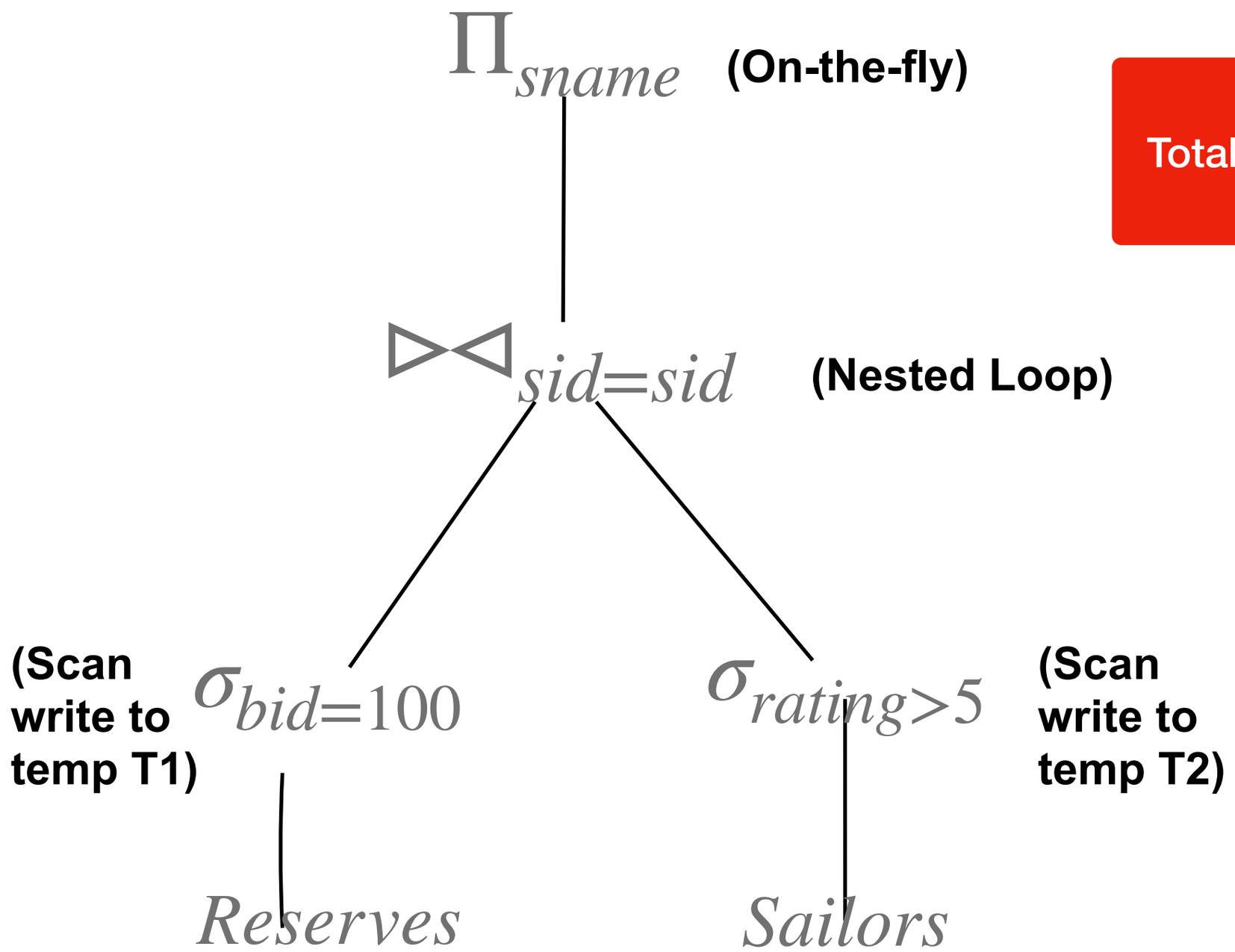
# Plan Alternativo 1 (Sin usar índices)



# Plan Alternativo 1 (Sin usar índices)

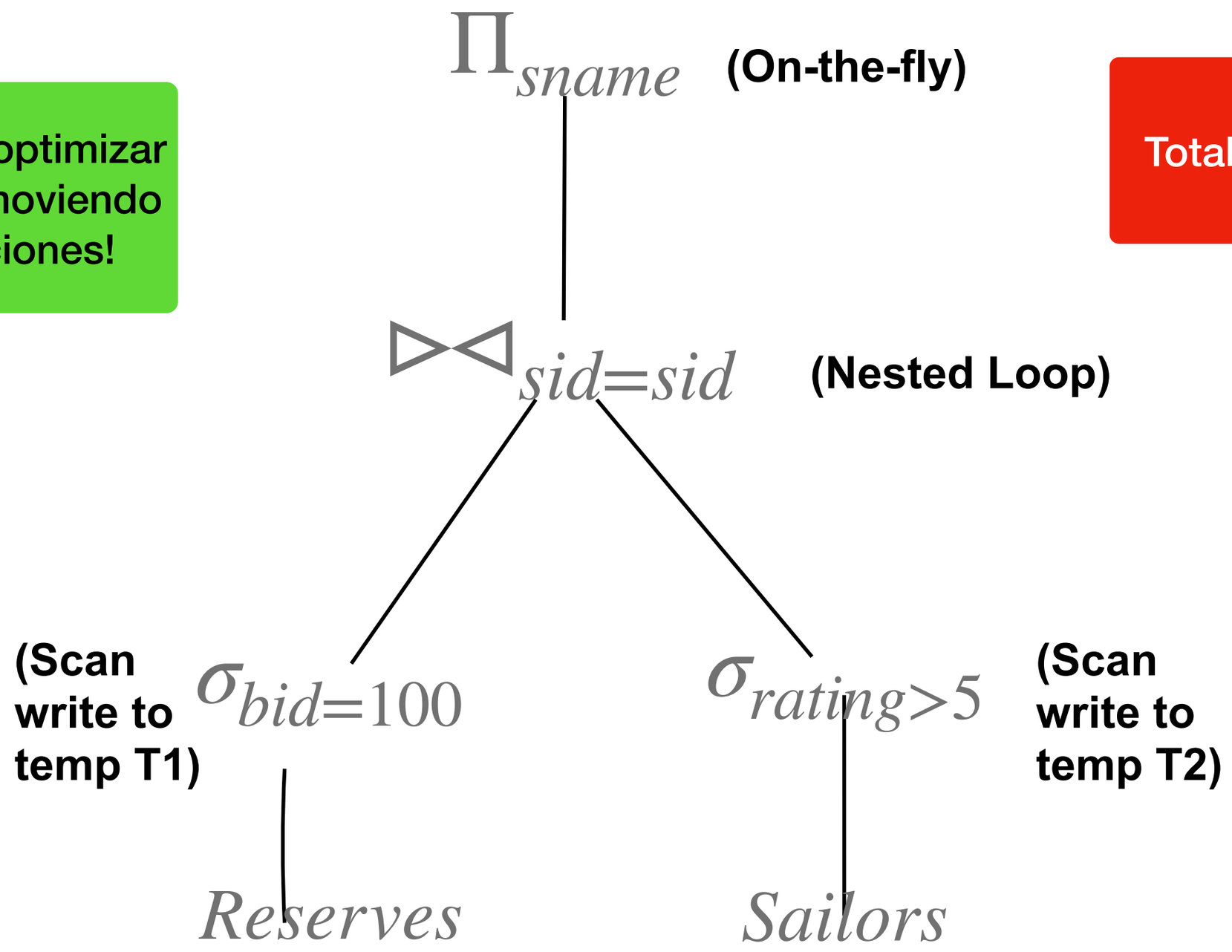


# Plan Alternativo 1 (Sin usar índices)



# Plan Alternativo 1 (Sin usar índices)

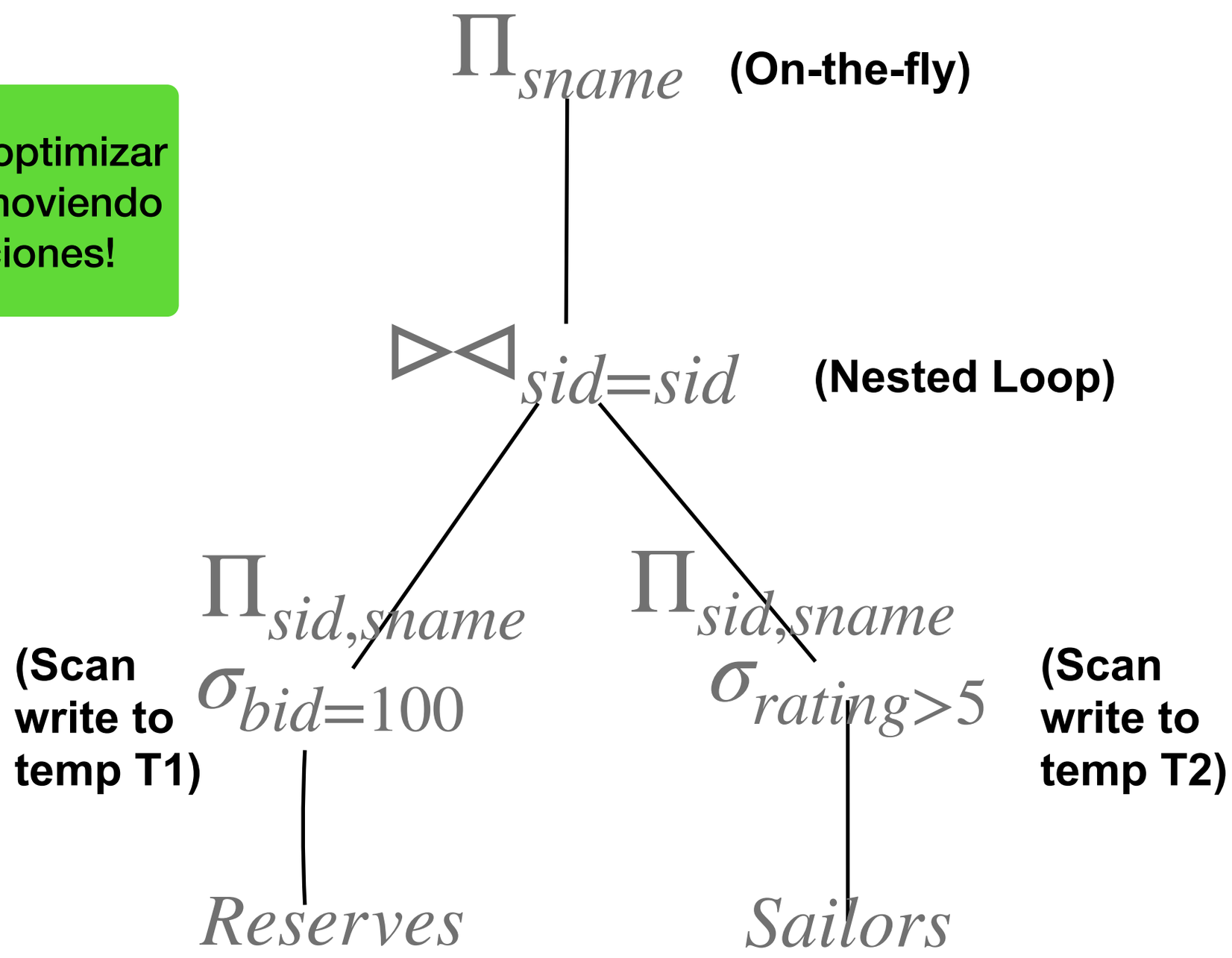
Se puede optimizar aún mas moviendo proyecciones!




Total = 2.770

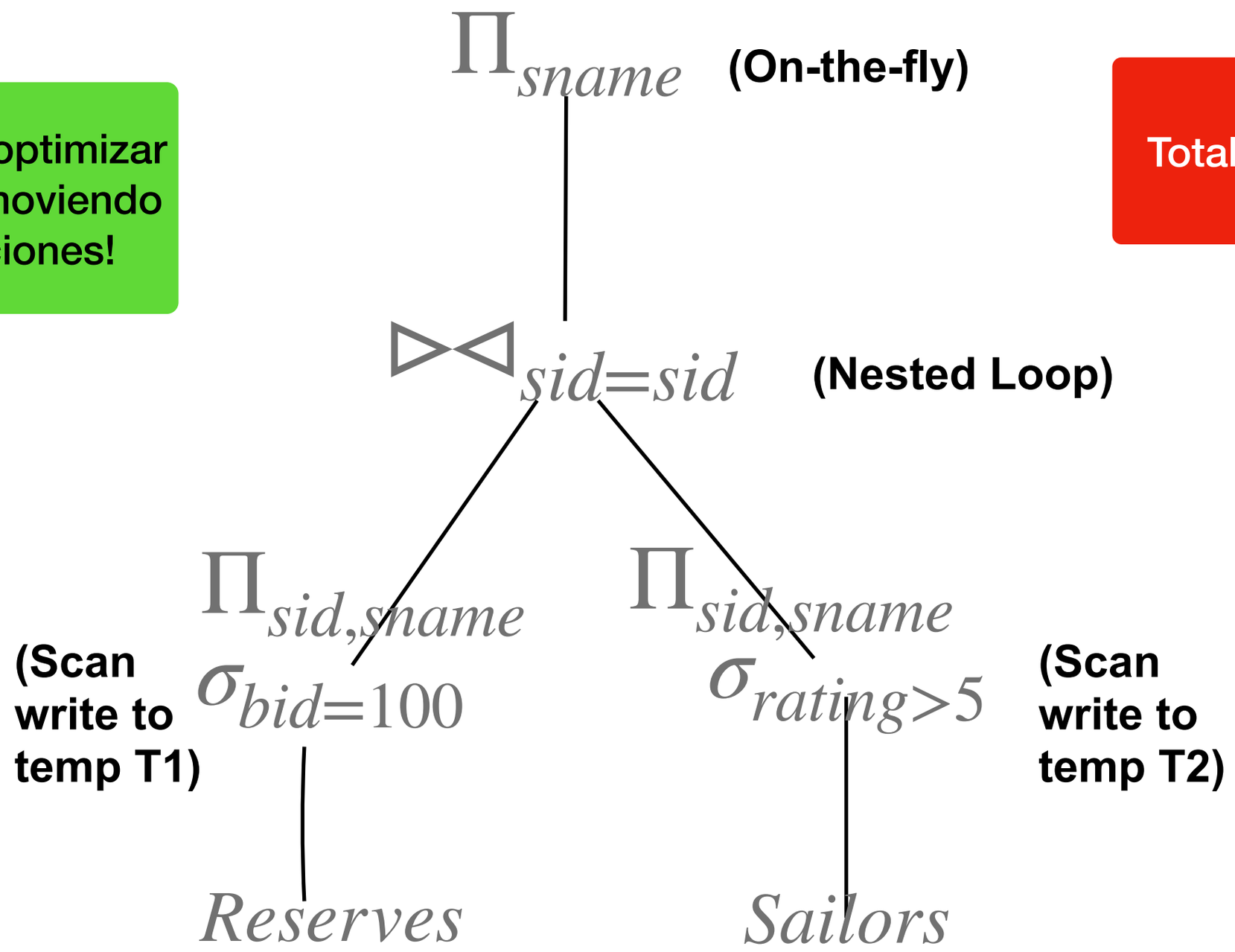
# Plan Alternativo 1 (Sin usar índices)

Se puede optimizar aún mas moviendo proyecciones!



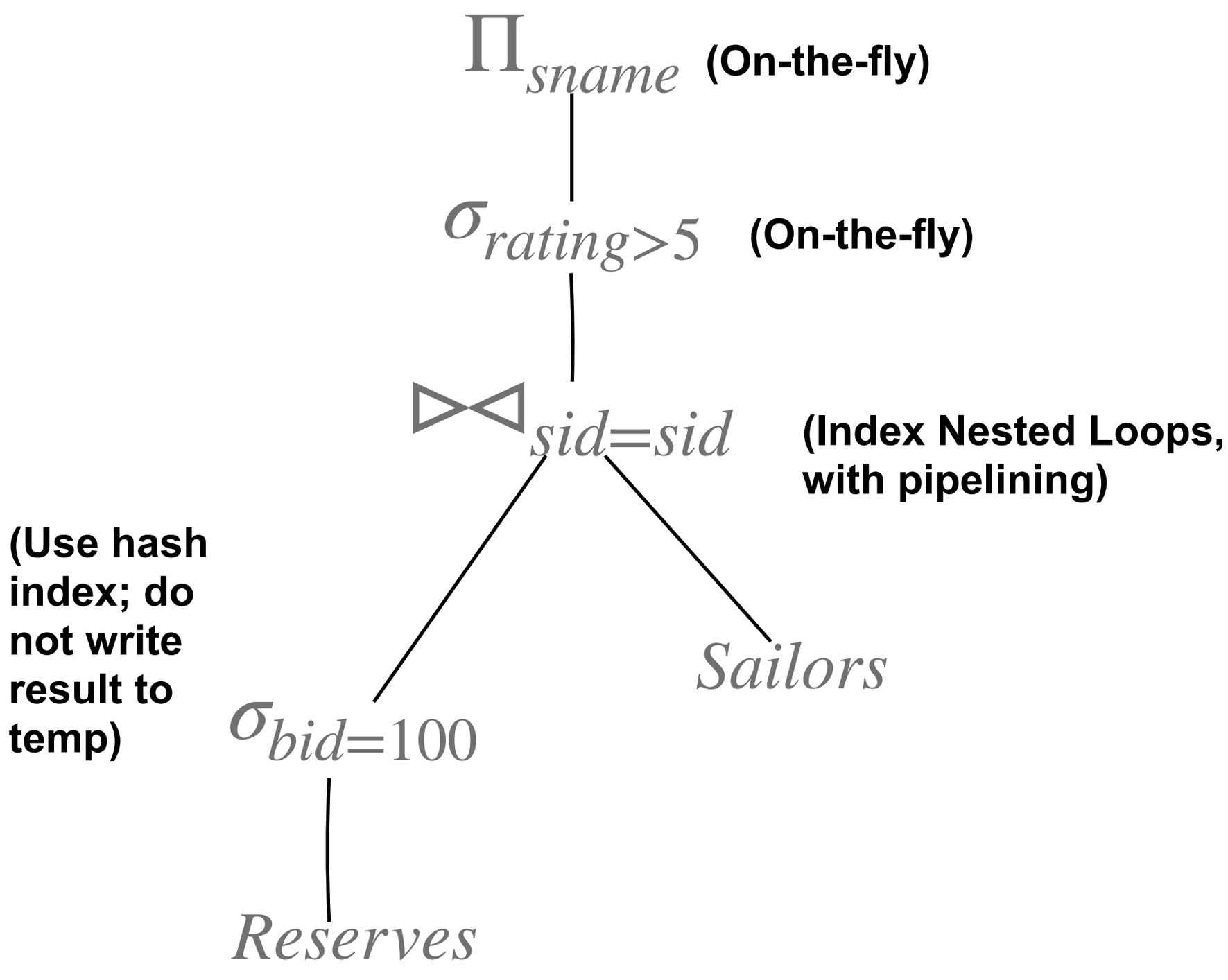
# Plan Alternativo 1 (Sin usar índices)

Se puede optimizar aún mas moviendo proyecciones!

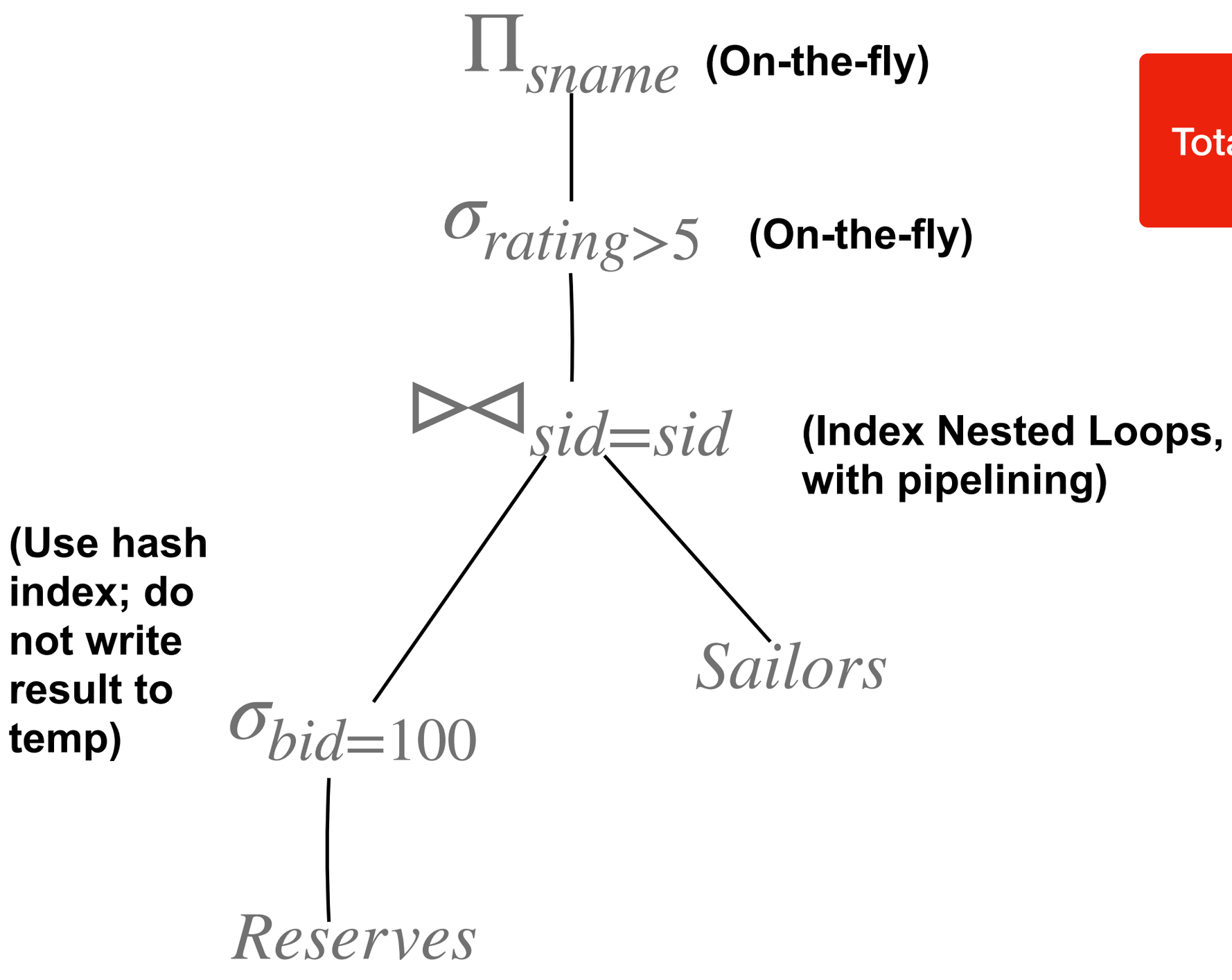



Total < 2.000

# Plan Alternativo 2 (Usando índices)

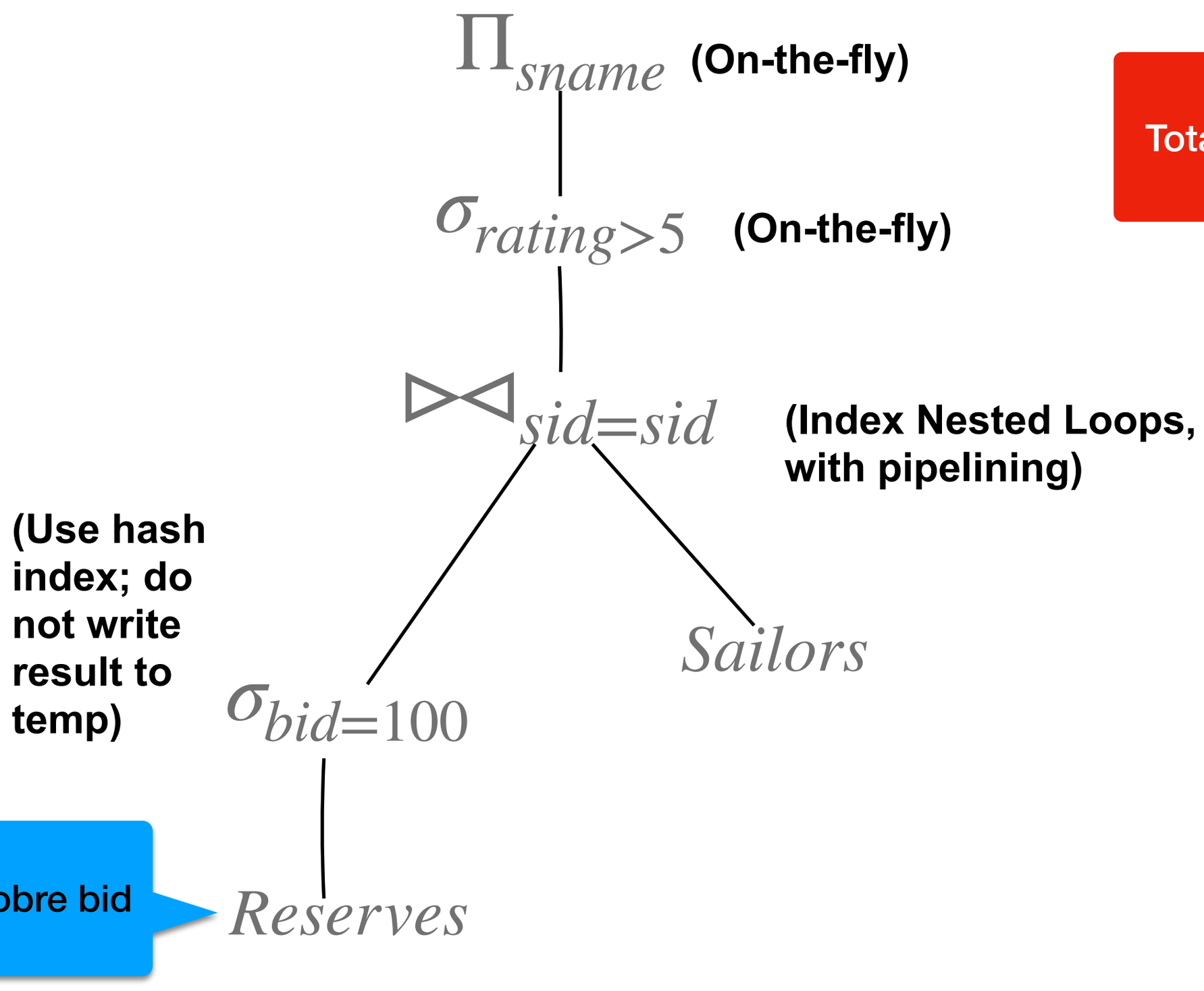


# Plan Alternativo 2 (Usando índices)

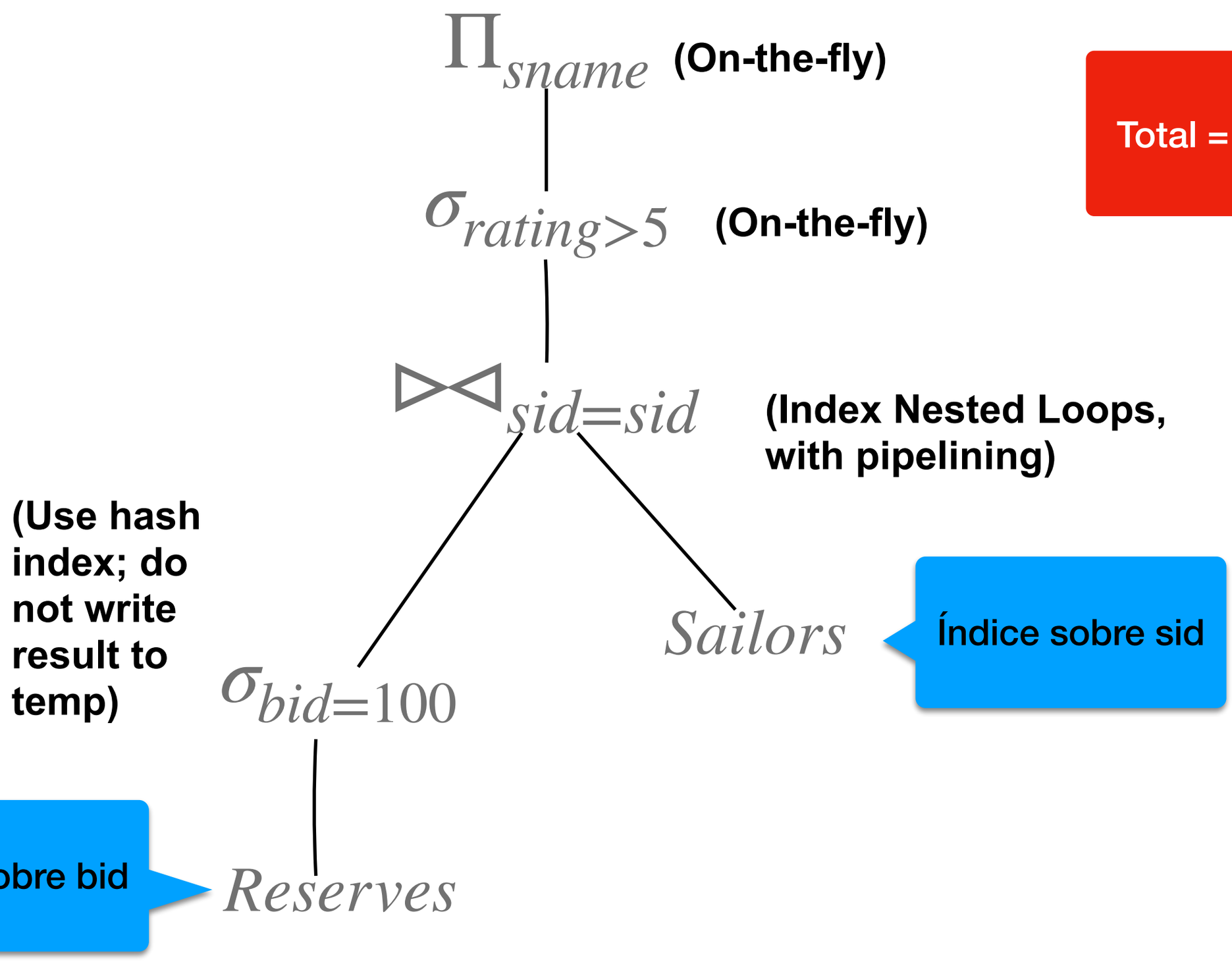



Total = 1.210

# Plan Alternativo 2 (Usando índices)

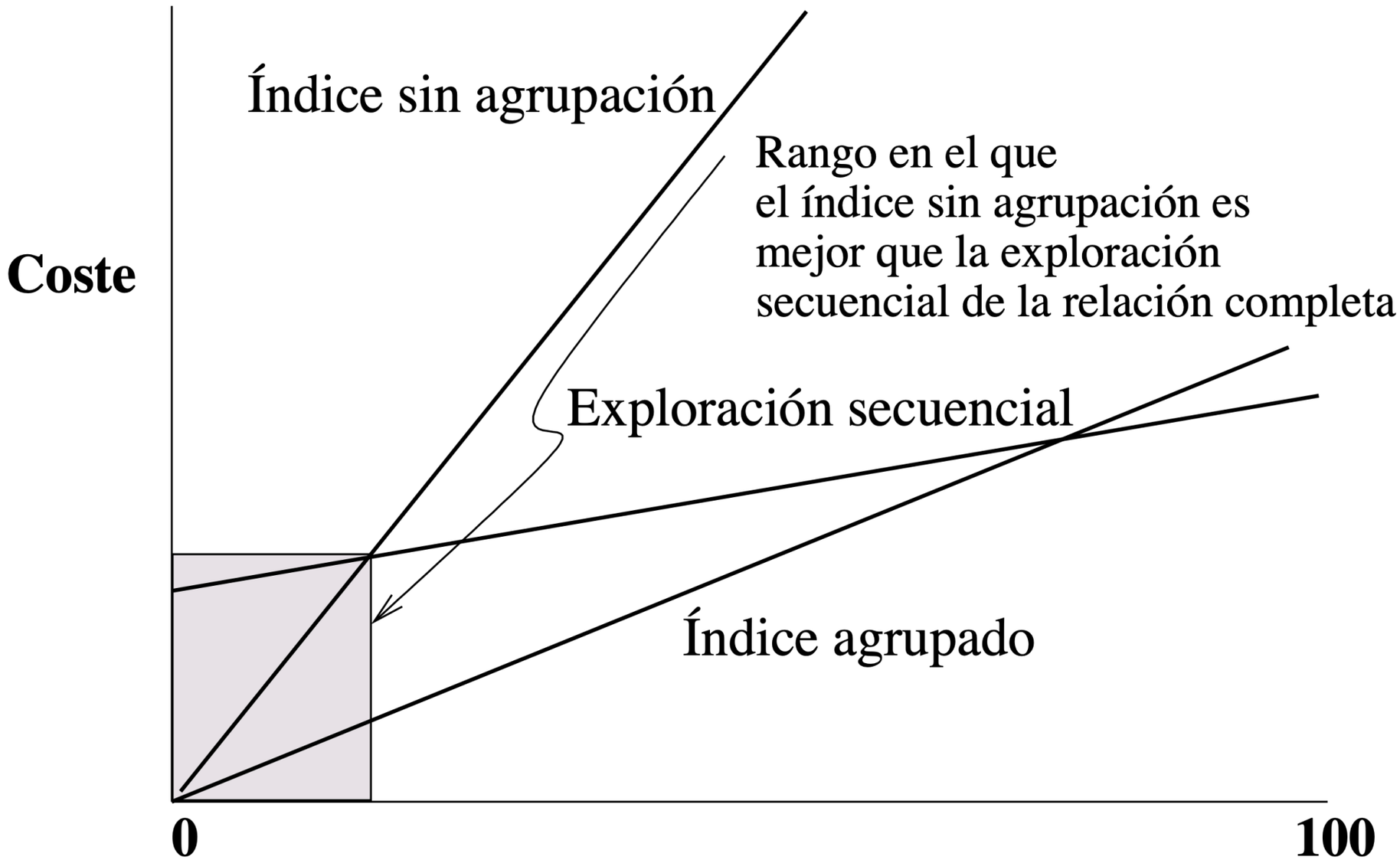


# Plan Alternativo 2 (Usando índices)

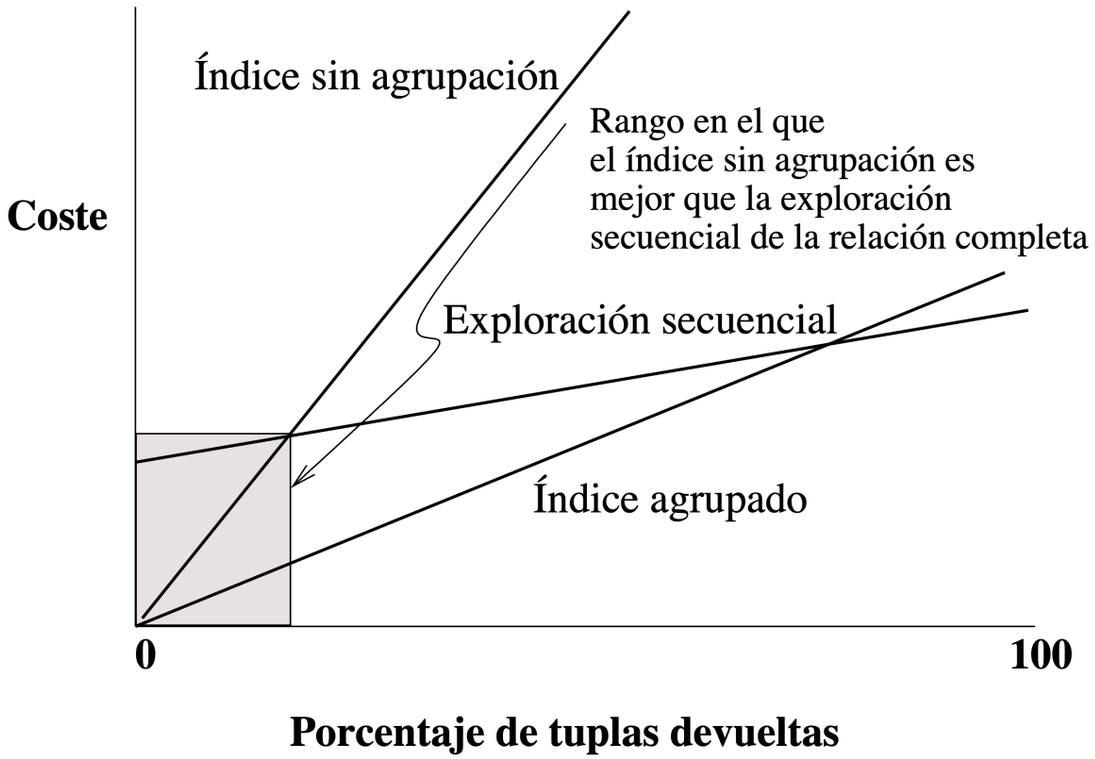



Total = 1.210

# Costos de evaluación según cantidad de resultados

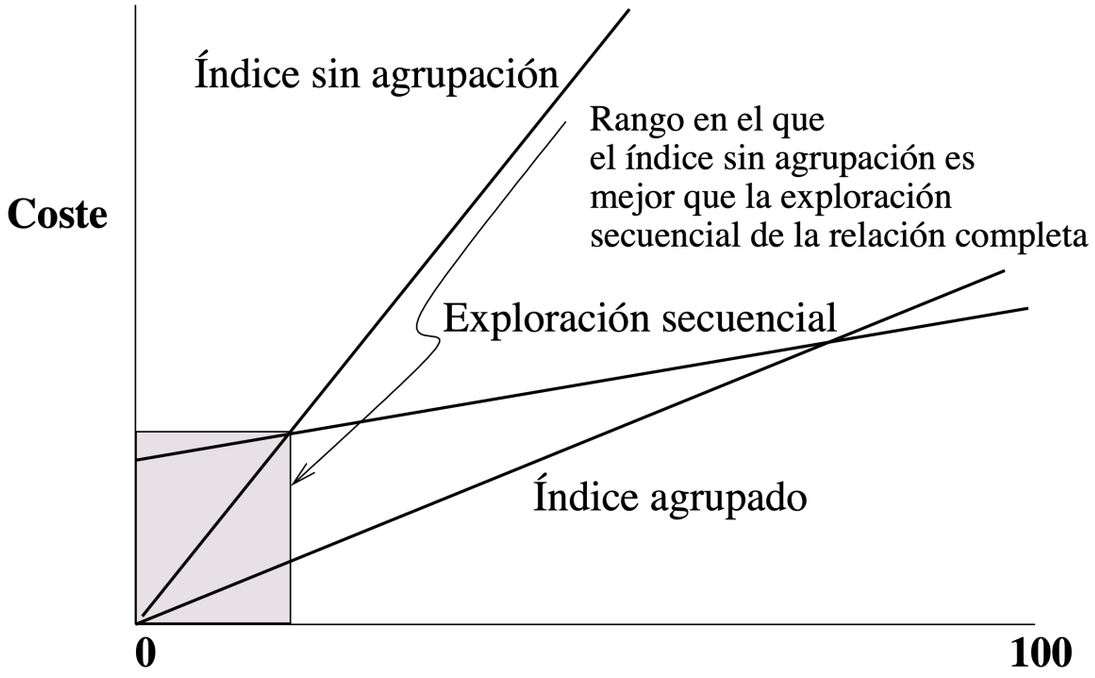


# Costos de evaluación según cantidad de resultados




```
SELECT *
FROM gi.actor100
WHERE genero = 'M'
```

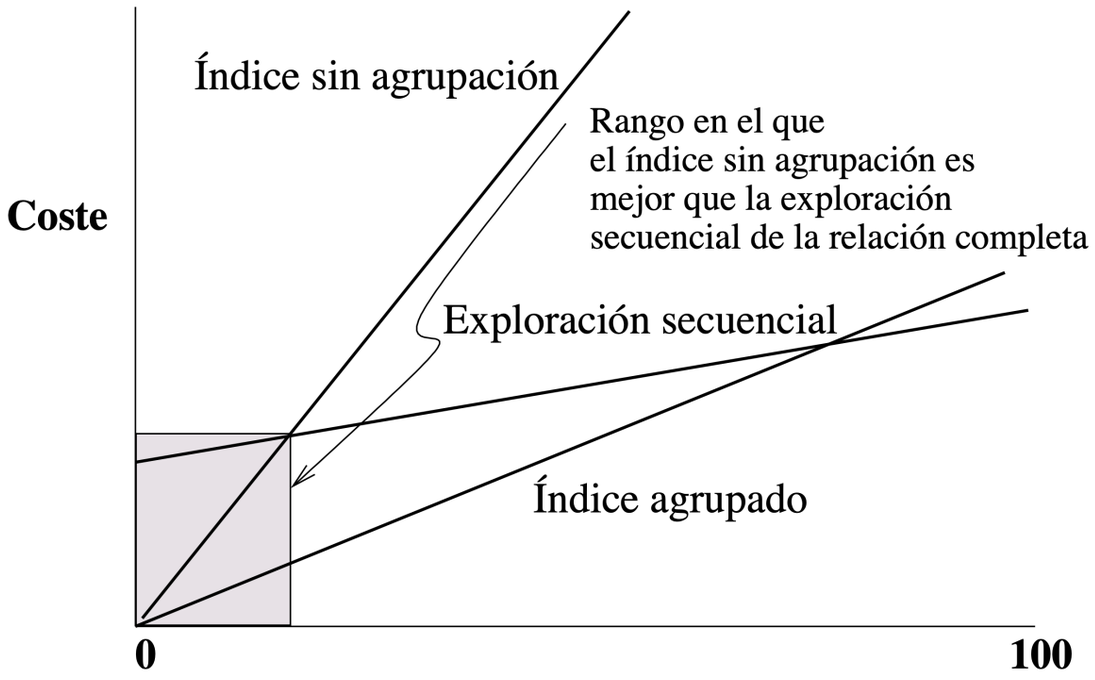
# Costos de evaluación según cantidad de resultados




Suponiendo indice sobre genero…

```
SELECT *
FROM gi.actor100
WHERE genero = 'M'
```

# Costos de evaluación según cantidad de resultados

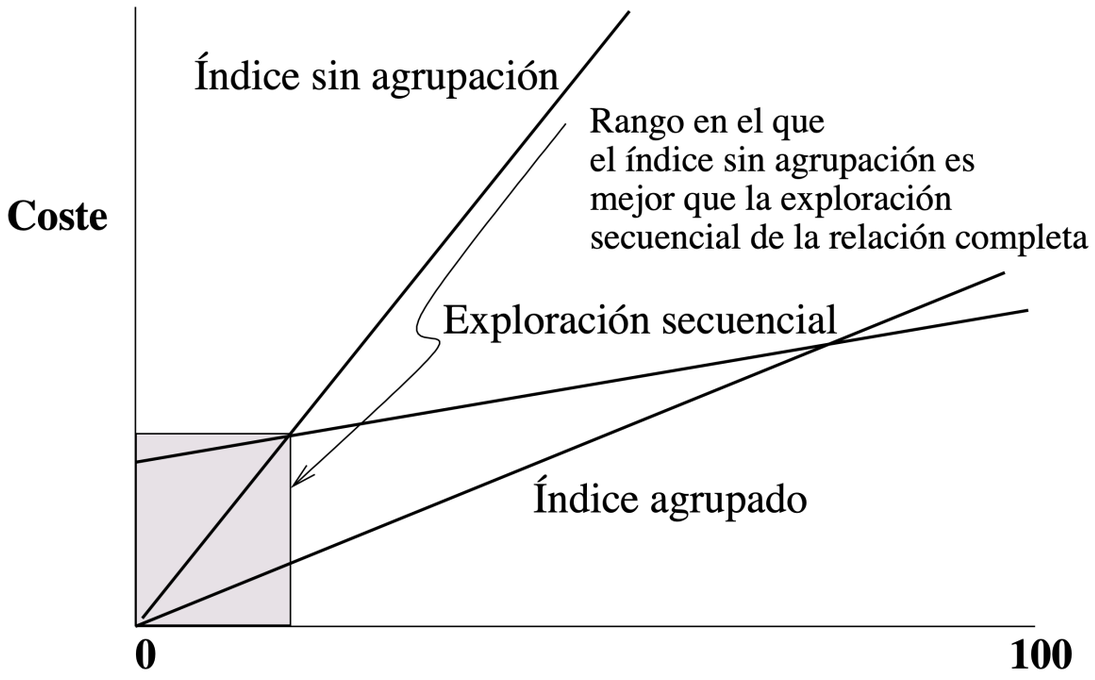



Suponiendo indice sobre genero…

¿Utilizará el índice el plan?

```
SELECT *
FROM gi.actor100
WHERE genero = 'M'
```

# Costos de evaluación según cantidad de resultados



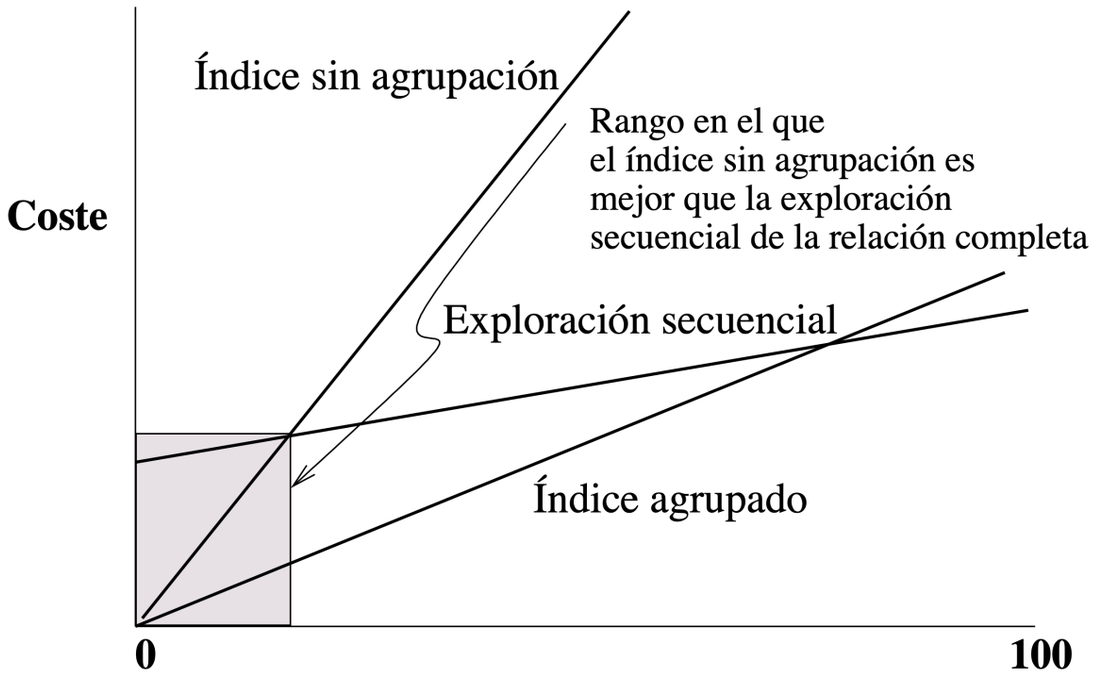

Suponiendo indice sobre genero…

¿Utilizará el índice el plan?

```
SELECT *
FROM gi.actor100
WHERE genero = 'M'
```

¿Es buena idea crear un índice para ese atributo?

# Costos de evaluación según cantidad de resultados




Suponiendo indice sobre genero…

¿Utilizará el índice el plan?

```
SELECT *
FROM gi.actor100
WHERE genero = 'M'
```

¿Es buena idea crear un índice para ese atributo?

¿Es buena idea crear un índice para todas las combinaciones de atributos?

# Resumen

- Hay muchas evaluaciones alternativas posibles para cada operador relacional (especialmente para joins)
- Una consulta se evalúa convirtiéndola en un árbol de operadores y evaluando cada operador en ese árbol.
- Hay que entender cómo funciona la optimización de consultas para entender el impacto sobre el rendimiento de determinados diseños de la base de datos (tablas, índices) sobre una carga de trabajo (workload = conjunto de consultas).
- Dos partes en la optimización de una consulta:
  - Considere un conjunto de planes alternativos.
    - Debe "podar" las múltiples posibilidades; típicamente sólo *left-deep plans*.
  - Debe estimar los costos de cada plan considerado.
    - Estimar tamaño de resultados intermedios y costo del plan en cada nodo.
    - *Asuntos claves*: Estadísticas, índices, implementación de operadores.

# Ver la planificación: **EXPLAIN/ANALYZE**

```
EXPLAIN SELECT ... -- mostrar la planificación sin ejecutar la consulta

ANALYZE SELECT ... -- ejecutar la consulta y mostrar analisis

EXPLAIN ANALYZE SELECT ... -- combinar ambos
```

```
EXPLAIN ANALYZE SELECT * FROM g.personaje100
 WHERE p_nombre='Whiplash' AND p_anho=2014;
```

# Ver la planificación: **EXPLAIN/ANALYZE**

```
EXPLAIN SELECT ... -- mostrar la planificación sin ejecutar la consulta

ANALYZE SELECT ... -- ejecutar la consulta y mostrar analisis

EXPLAIN ANALYZE SELECT ... -- combinar ambos
```

```
EXPLAIN ANALYZE SELECT * FROM g.personaje100
 WHERE p_nombre='Whiplash' AND p_anho=2014;
```

# Ver la planificación: **EXPLAIN/ANALYZE**

```
EXPLAIN SELECT ... -- mostrar la planificación sin ejecutar la consulta

ANALYZE SELECT ... -- ejecutar la consulta y mostrar analisis

EXPLAIN ANALYZE SELECT ... -- combinar ambos
```

```
EXPLAIN ANALYZE SELECT * FROM g.personaje100
 WHERE p_nombre='Whiplash' AND p_anho=2014;
```

```
Seq Scan on personaje100
   (cost=0.00..53967.89 rows=2 width=47)
   (actual time=6.683..402.203 rows=54 loops=1)
  Filter: ((p_nombre)::text = 'Whiplash'::text AND (p_anho = 2014))
  Rows Removed by Filter: 2170472
Planning time: 0.111 ms
Execution time: 402.273 ms
```

# Ver la planificación: **EXPLAIN/ANALYZE**

```
EXPLAIN SELECT ... -- mostrar la planificación sin ejecutar la consulta

ANALYZE SELECT ... -- ejecutar la consulta y mostrar analisis

EXPLAIN ANALYZE SELECT ... -- combinar ambos
```

```
EXPLAIN ANALYZE SELECT * FROM gi.personaje100
  WHERE p_nombre='Whiplash' AND p_anho=2014;
```

# Ver la planificación: **EXPLAIN/ANALYZE**

```
EXPLAIN SELECT ... -- mostrar la planificación sin ejecutar la consulta

ANALYZE SELECT ... -- ejecutar la consulta y mostrar analisis

EXPLAIN ANALYZE SELECT ... -- combinar ambos
```

```
EXPLAIN ANALYZE SELECT * FROM gi.personaje100
  WHERE p_nombre='Whiplash' AND p_anho=2014;
```

```
Index Scan using personaje100_pnombreanho on personaje100
    (cost=0.43..12.47 rows=2 width=47)
    (actual time=0.123..0.540 rows=54 loops=1)
   Index Cond: (((p_nombre)::text = 'Whiplash'::text) AND (p_anho = 2014))
 Planning time: 0.143 ms
 Execution time: 0.589 ms
```

Para más detalles tomar

IIC3413 - Implementación de sistemas de Bases de Datos