

# Bases de Datos

Clase 9: Almacenamiento y Índices

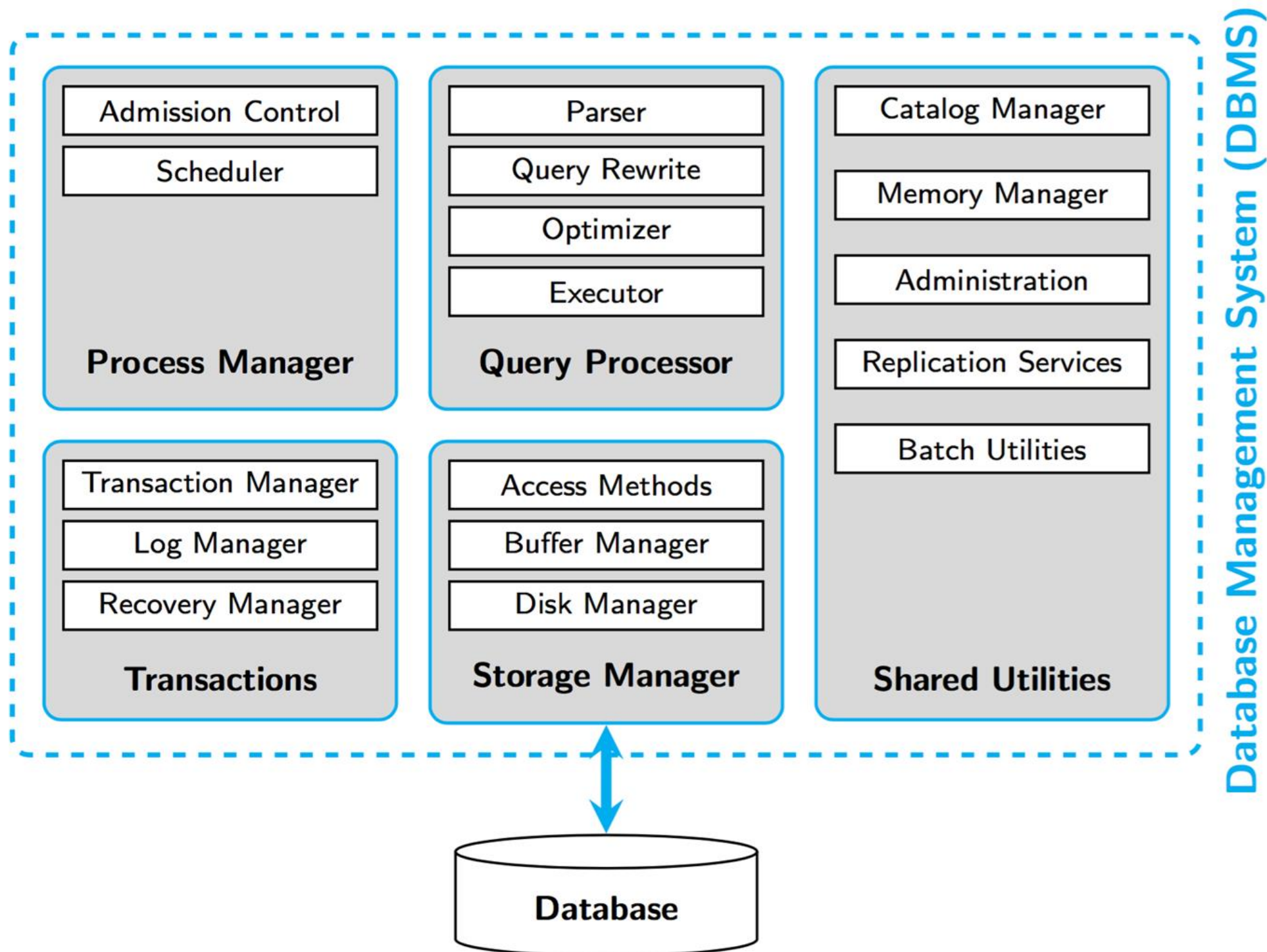
# Hasta ahora

Usamos los DBMS como una caja negra: hacemos una consulta y el sistema se encarga

Pero, ¿cómo funciona realmente el sistema?

¿Cómo se evalúa una consulta?

# Arquitectura de un DBMS



# Ciclo de vida de una consulta

**Paso 1:** SQL → álgebra relacional

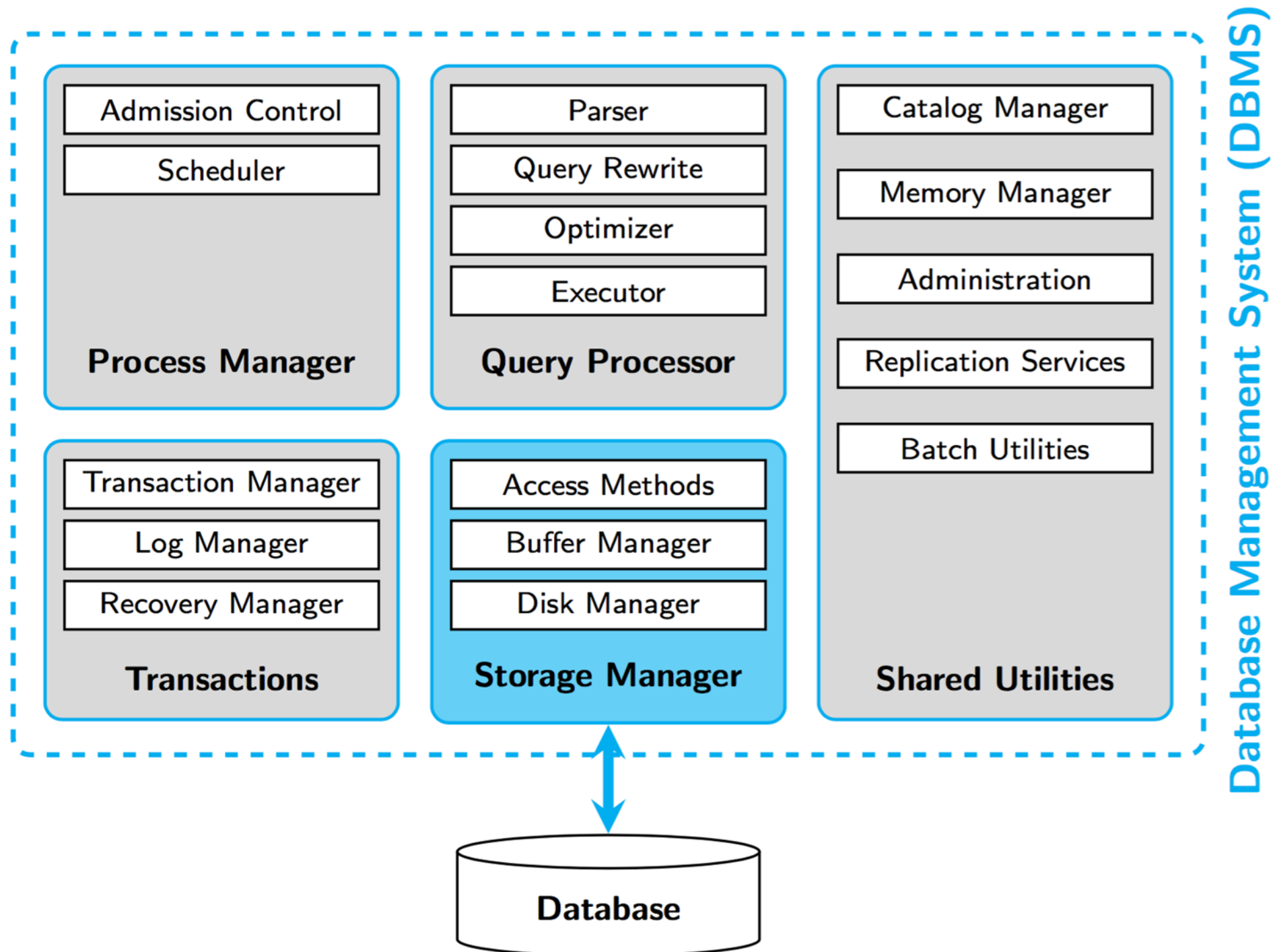
**Paso 2:** Reescritura → plan lógico

**Paso 3:** ¿Cuales algoritmos uso? → plan físico

**Paso 4:** Ejecución

**¡Para ejecutar la consulta necesito acceder a los datos (en disco)!**

# Storage Manager



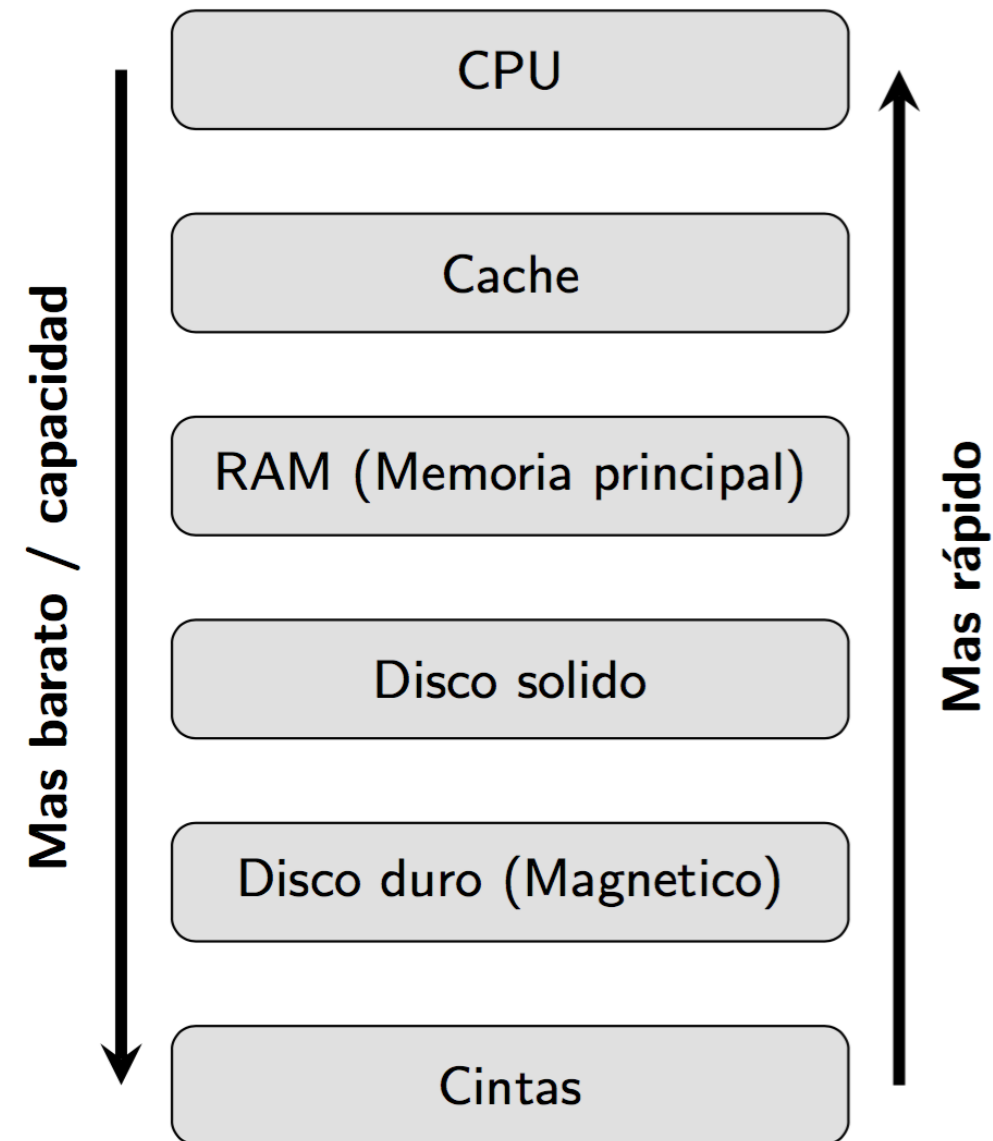
# Almacenamiento

# Almacenamiento

La base de datos necesita hacer persistente los datos  
en memoria secundaria

# Almacenamiento

## Jerarquía de Memoria





# Disco Duro

**Sector:** unidad física mínima de almacenamiento para el Disco (definido por hardware)

**Página:** unidad lógica mínima de almacenamiento para el Sistema de Archivos (definido por OS/DBMS)

**¡Un DBMS trabaja al nivel de página!**

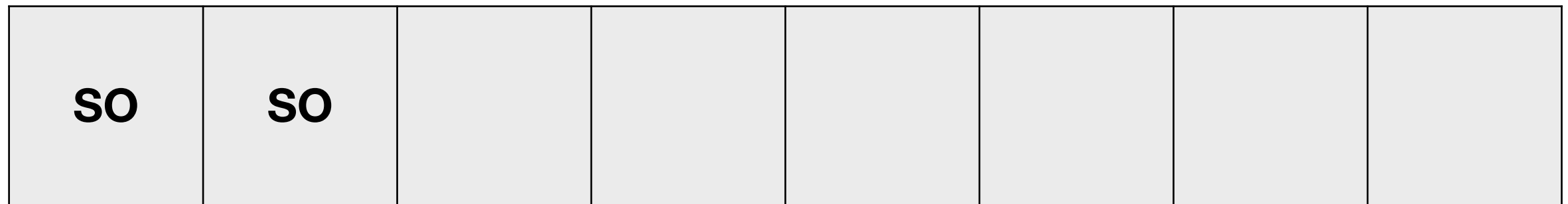
# Disco Duro y RAM

**RAM**

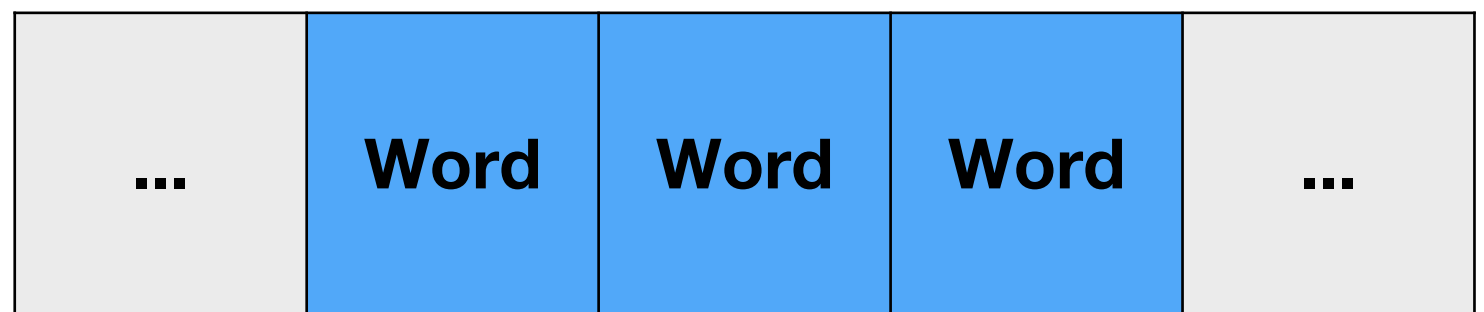


# Disco Duro y RAM

**RAM**

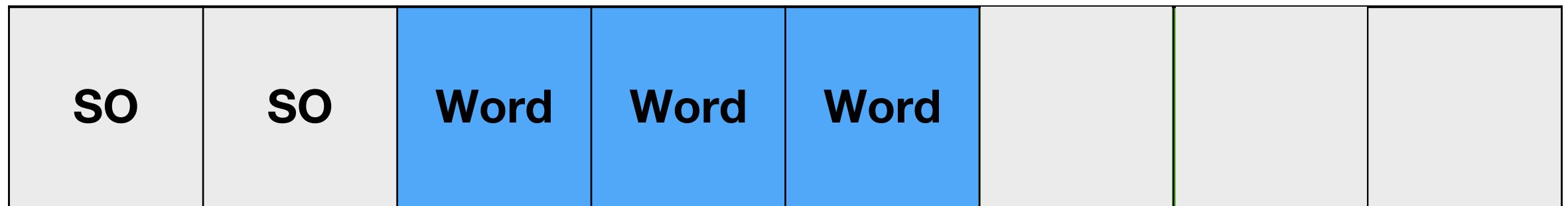


**Disco Duro**

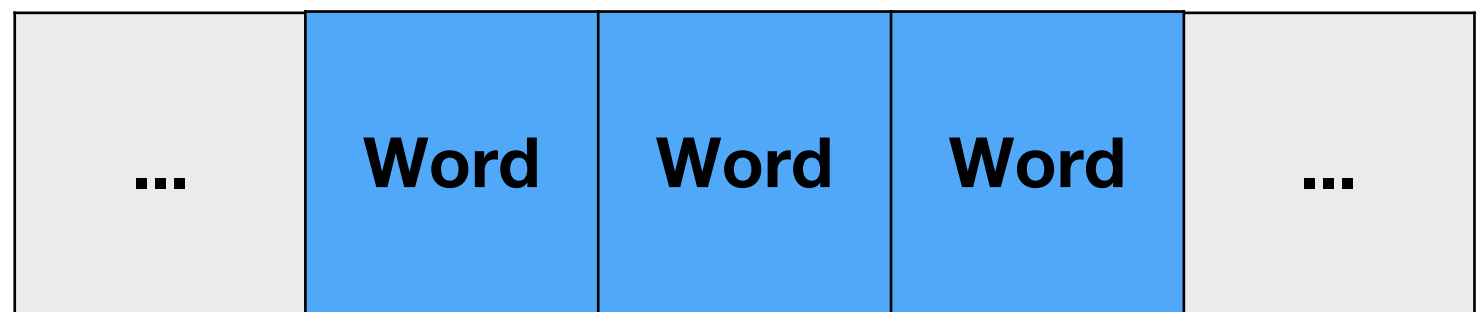


# Disco Duro y RAM

## RAM



## Disco Duro



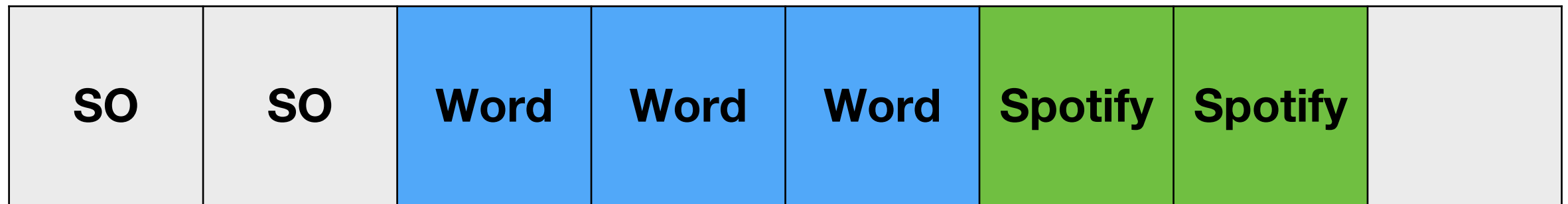
# Disco Duro y RAM

**RAM**

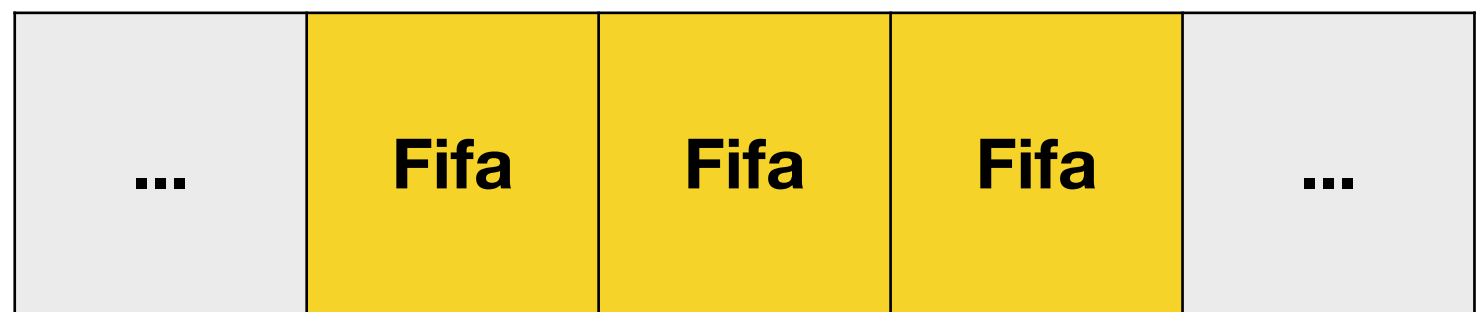
SO	SO	Word	Word	Word	Spotify	Spotify	
----	----	------	------	------	---------	---------	--

# Disco Duro y RAM

## RAM

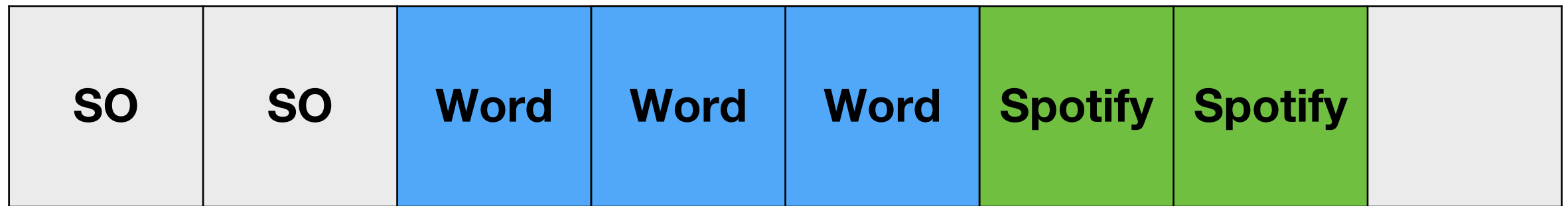


## Disco Duro



# Disco Duro y RAM

## RAM



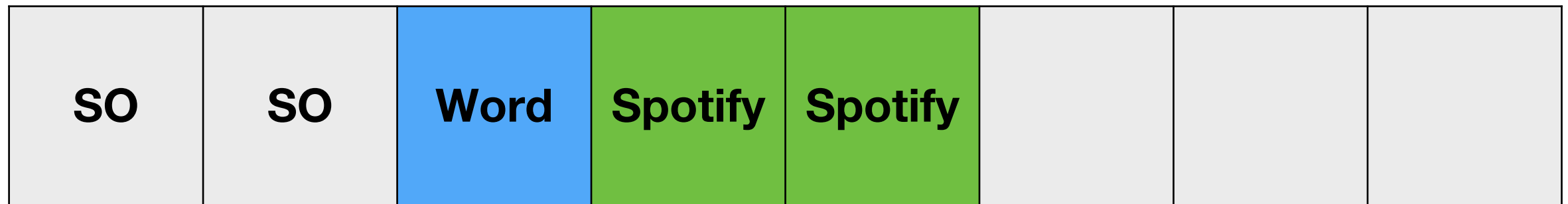
## Disco Duro (espacio de swap)

## Disco Duro



# Disco Duro y RAM

## RAM



## Disco Duro (espacio de swap)

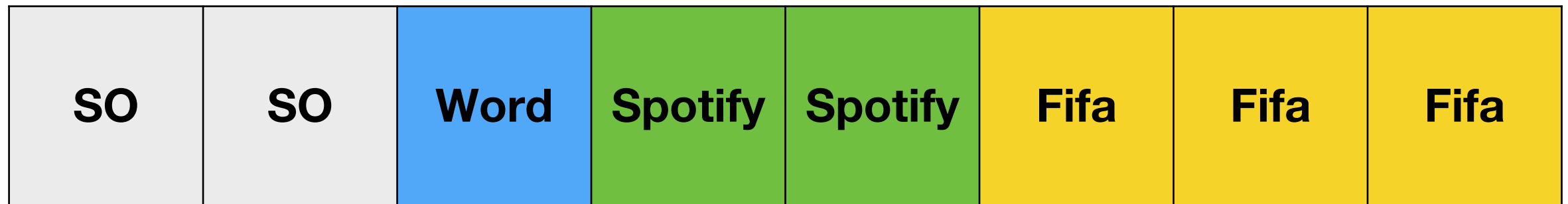


## Disco Duro



# Disco Duro y RAM

## RAM



## Disco Duro (espacio de swap)

## Disco Duro



# Disco Duro y RAM

## RAM



## Disco Duro (espacio de swap)



# Disco Duro y DBMS

Los records de las bases de datos se almacenan en **páginas** de disco.

A medida que se hace necesario, las páginas son traídas a memoria principal (**buffer**)

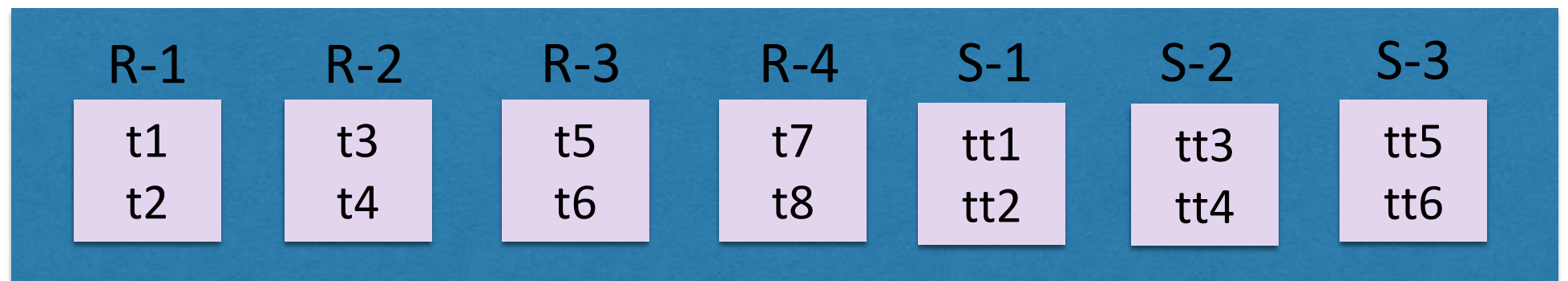
# Páginas, disco y buffer

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

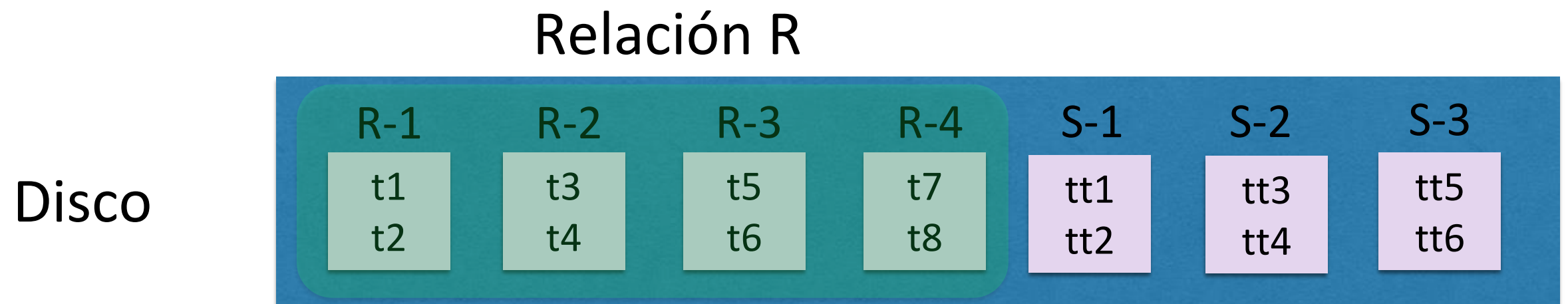
Para cargar estas páginas, la base de datos reserva un espacio en RAM llamado **Buffer**

# Páginas, disco y buffer

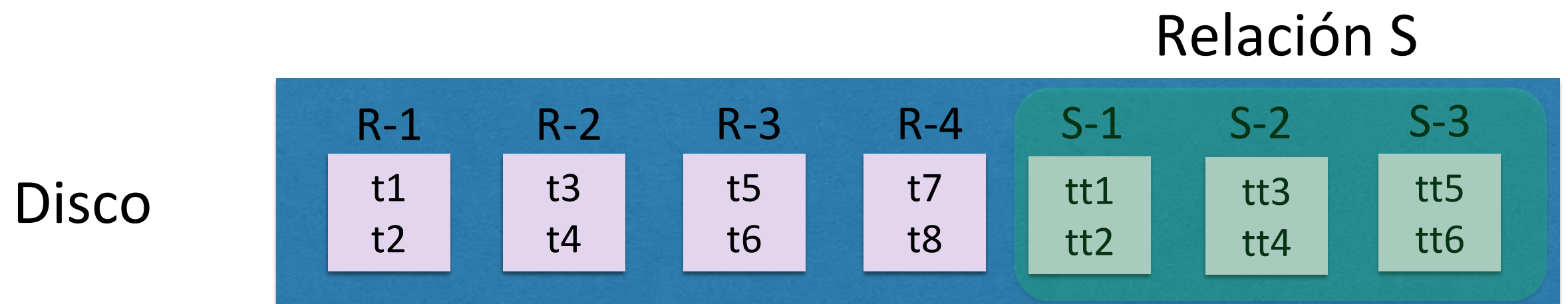
Disco



# Páginas, disco y buffer

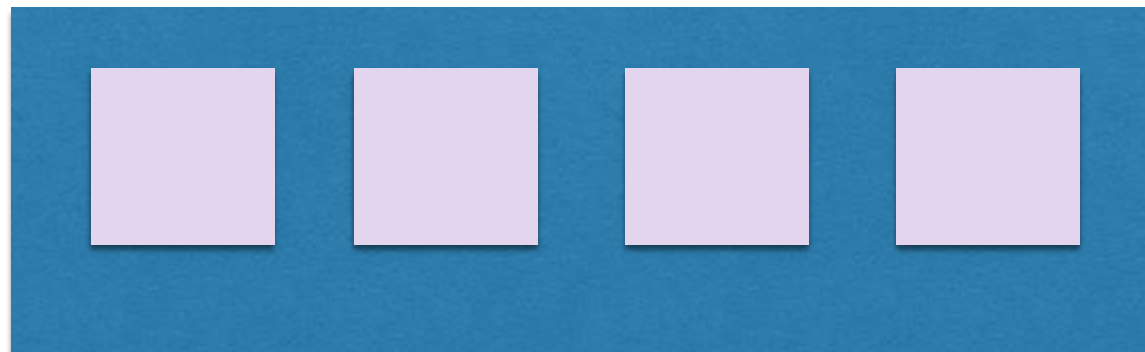


# Páginas, disco y buffer

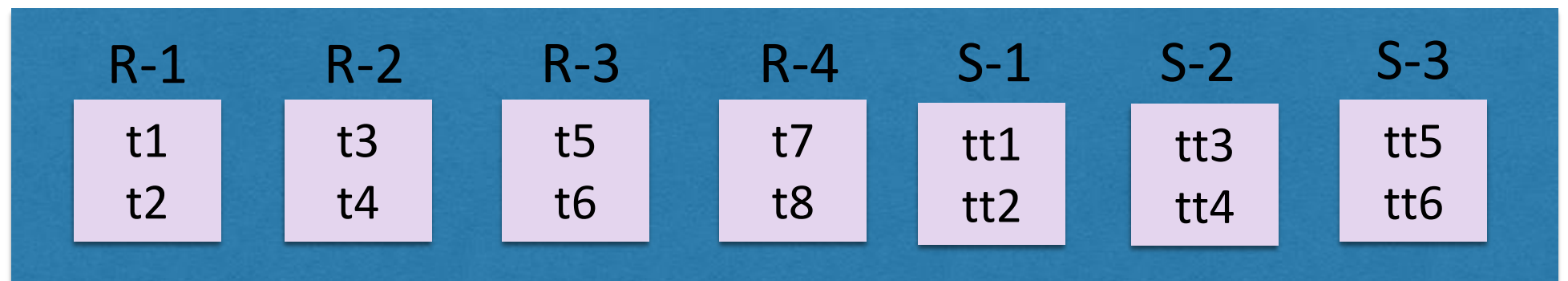


# Páginas, disco y buffer

Buffer

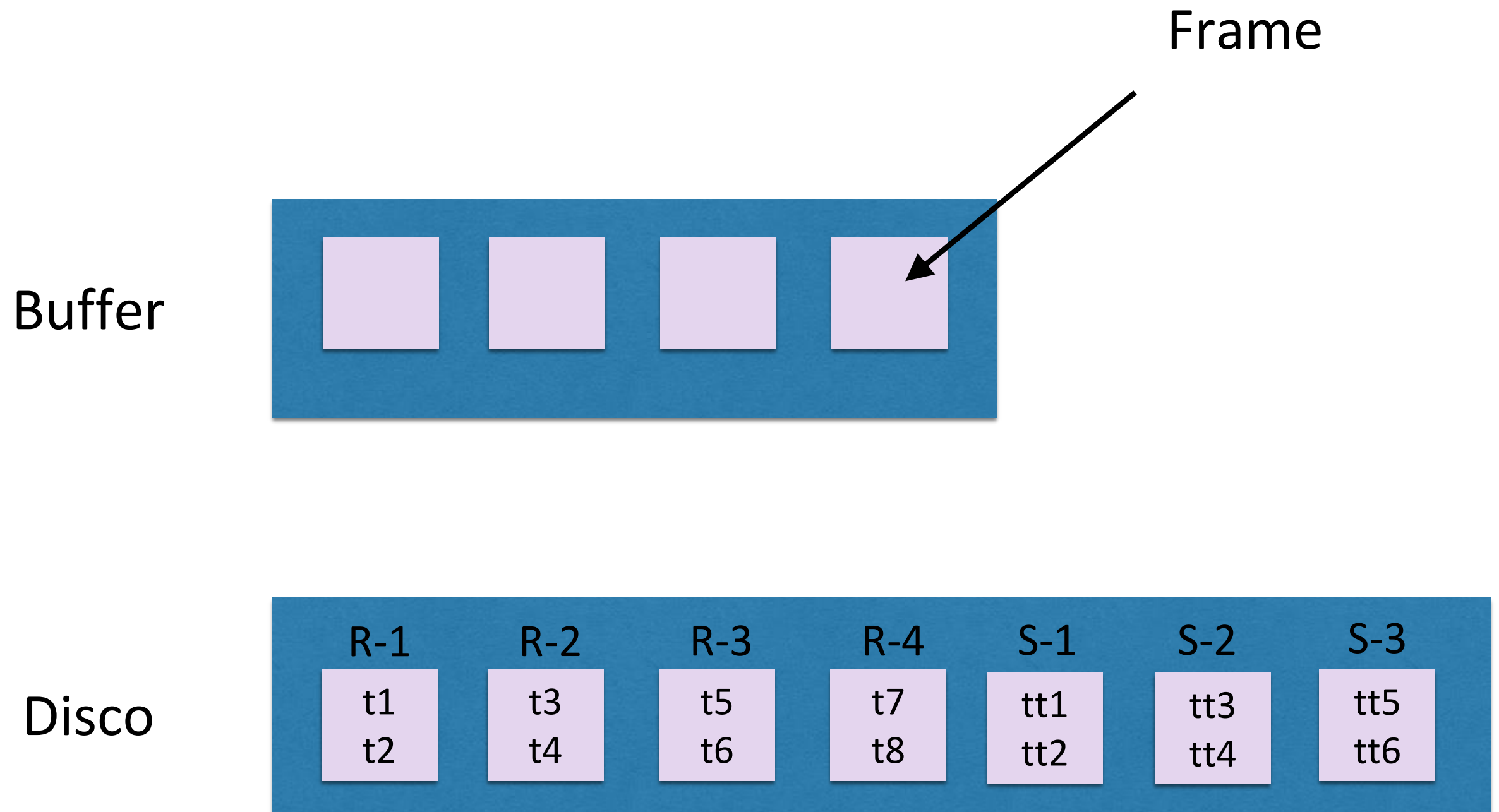


Disco





# Páginas, disco y buffer

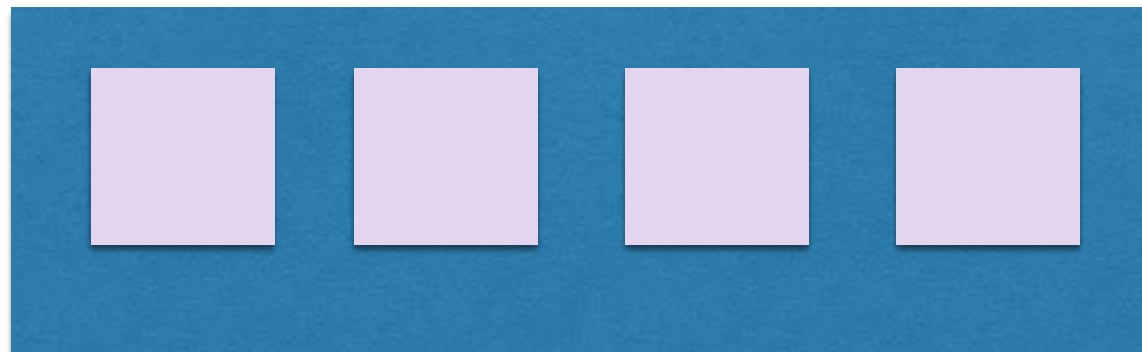


# Páginas, disco y buffer

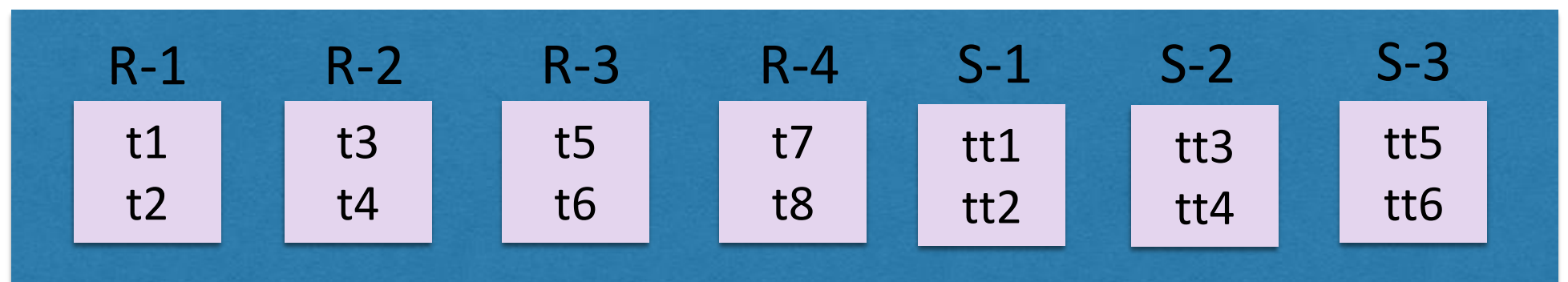
DB

Necesito tupla t3 de R!

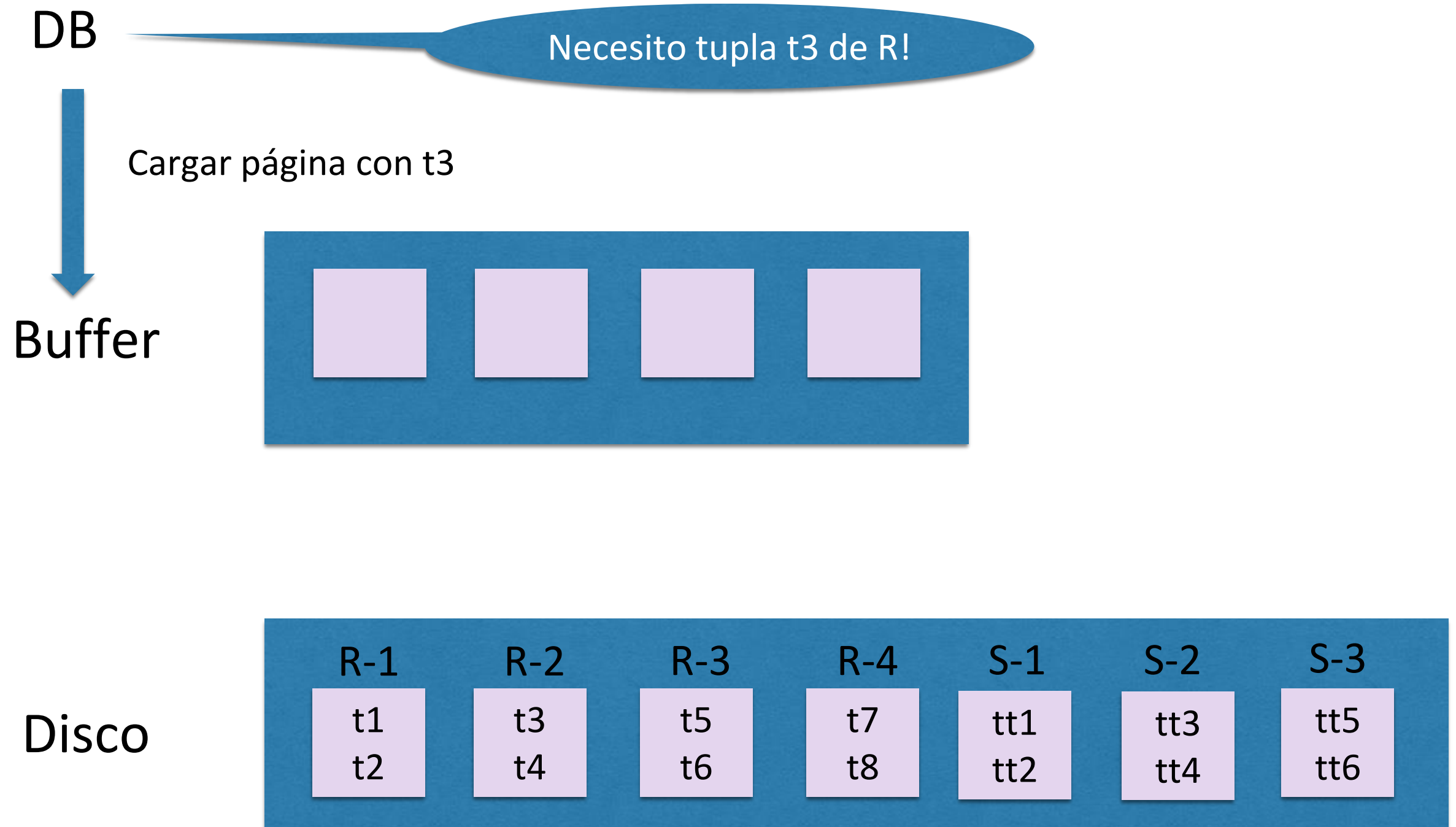
Buffer



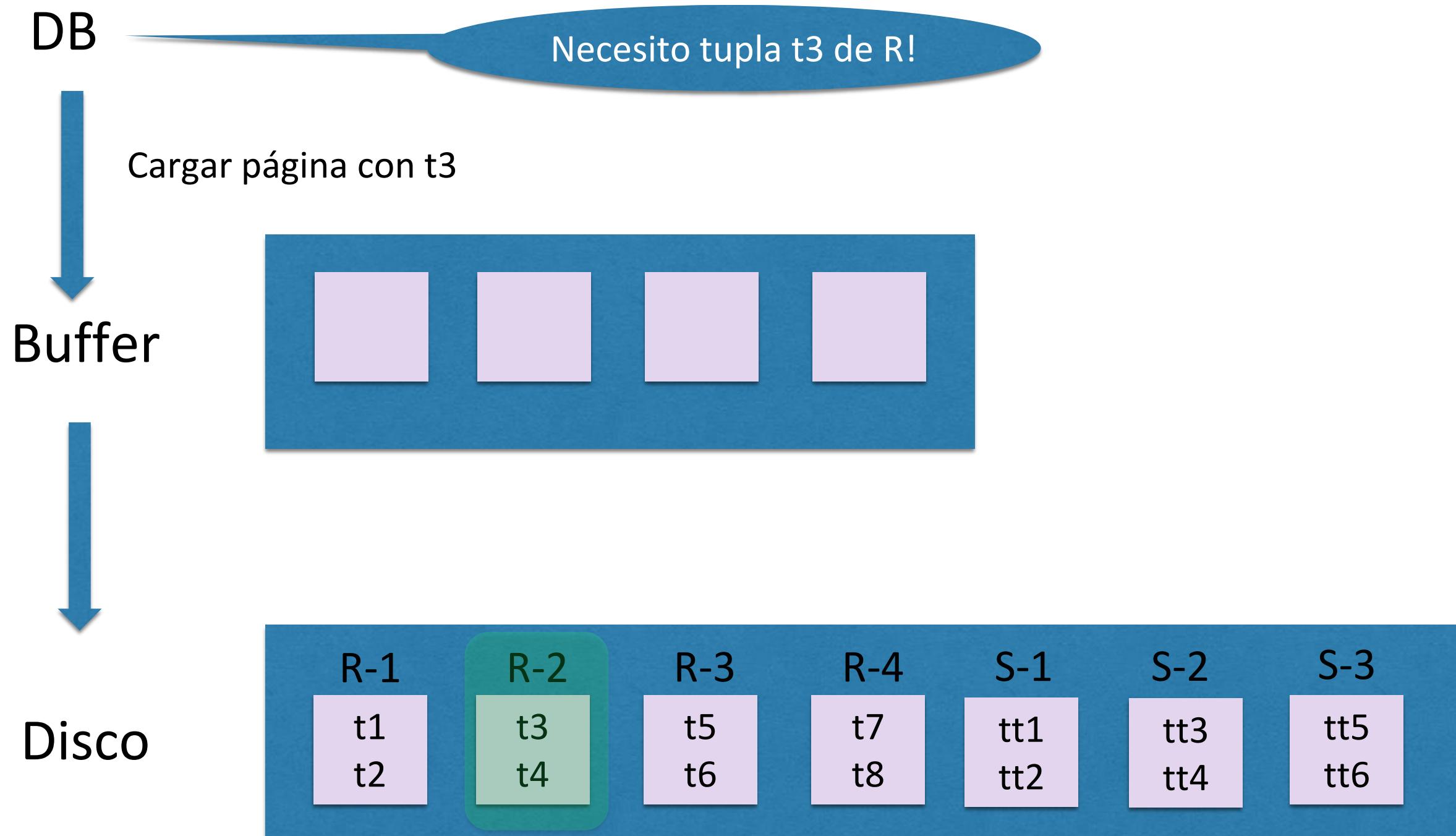
Disco



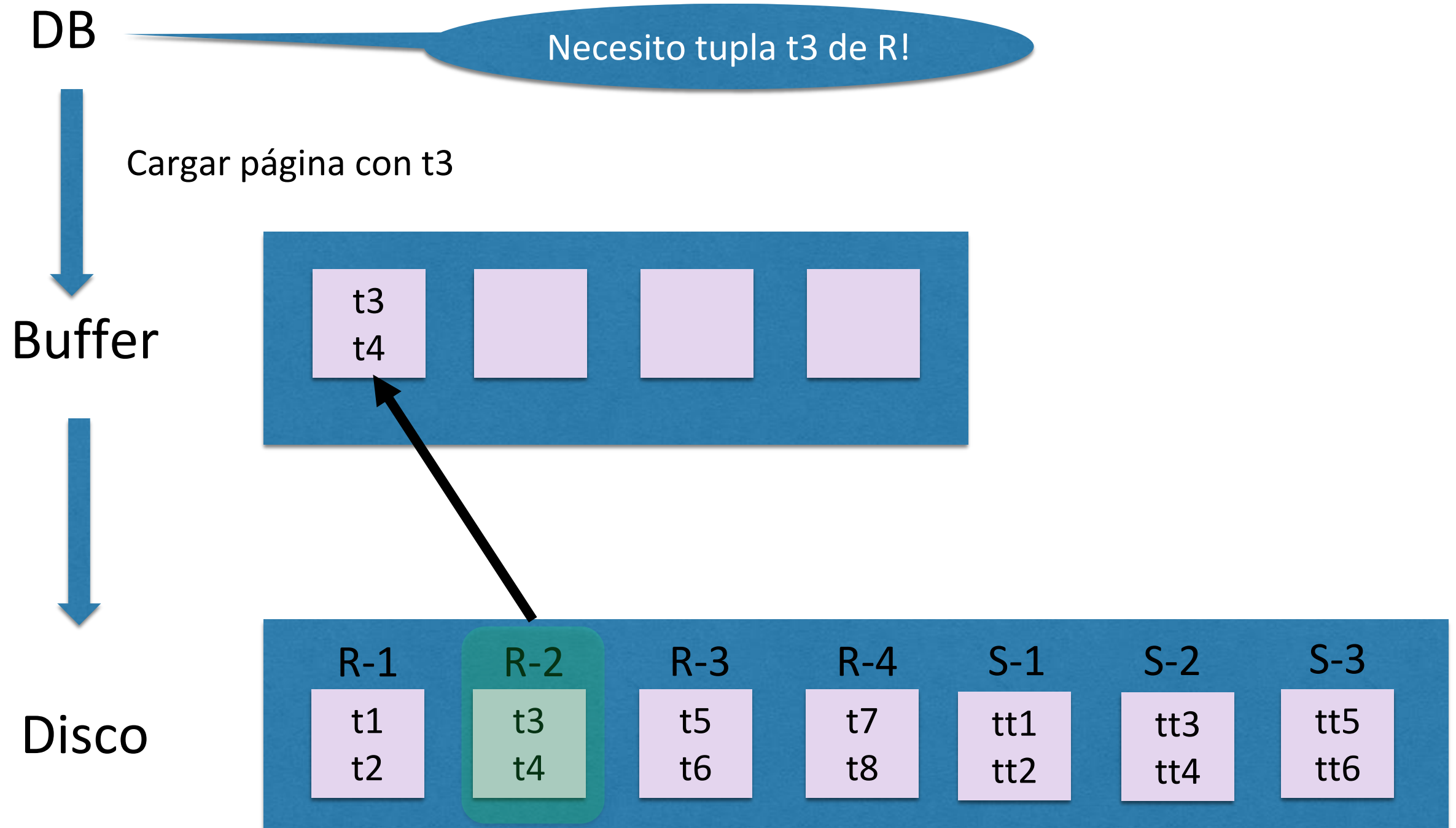
# Páginas, disco y buffer



# Páginas, disco y buffer



# Páginas, disco y buffer



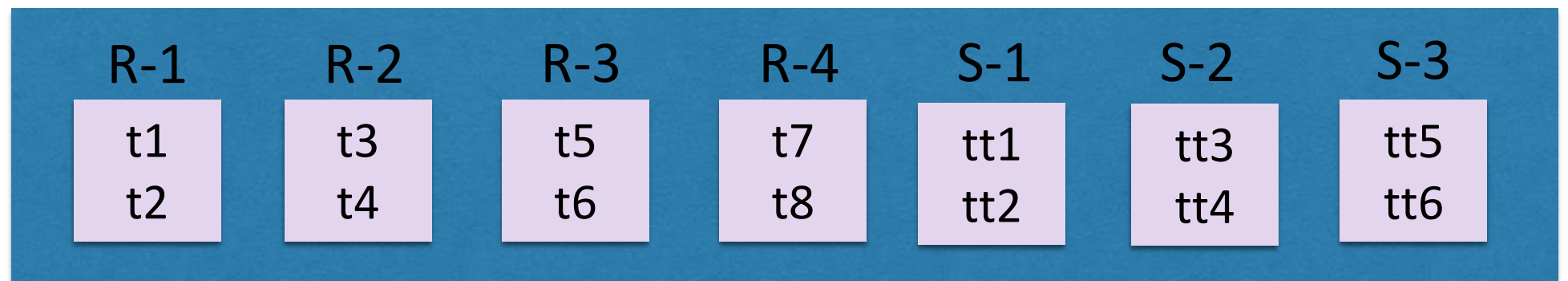
# Páginas, disco y buffer

DB

Buffer



Disco



# Páginas, disco y buffer

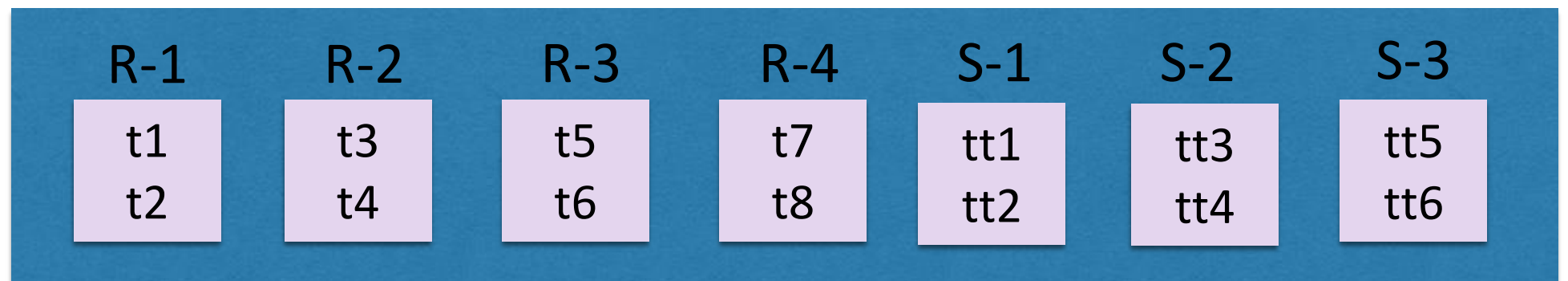
DB

Necesito tupla t4 de R!

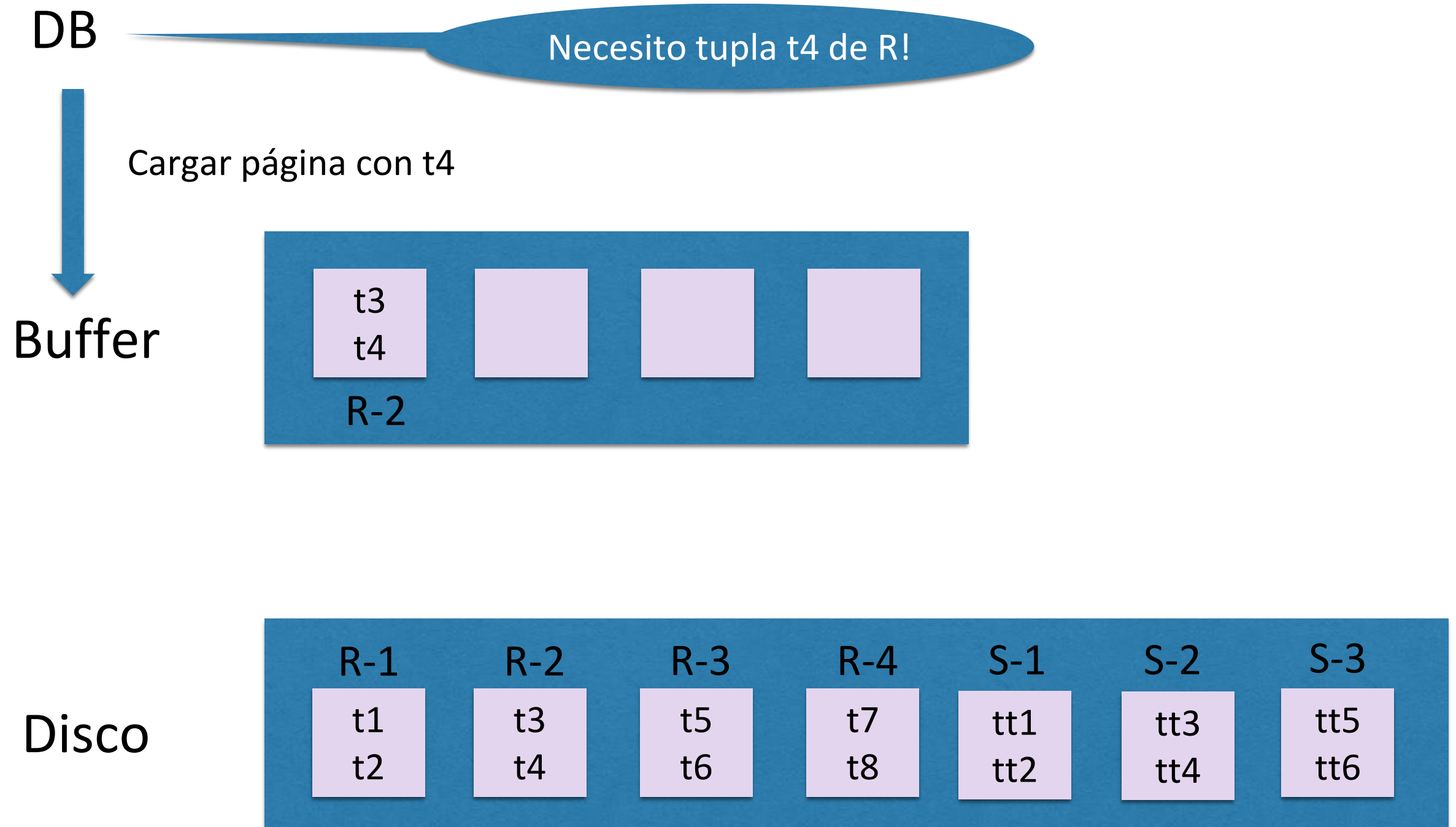
Buffer



Disco

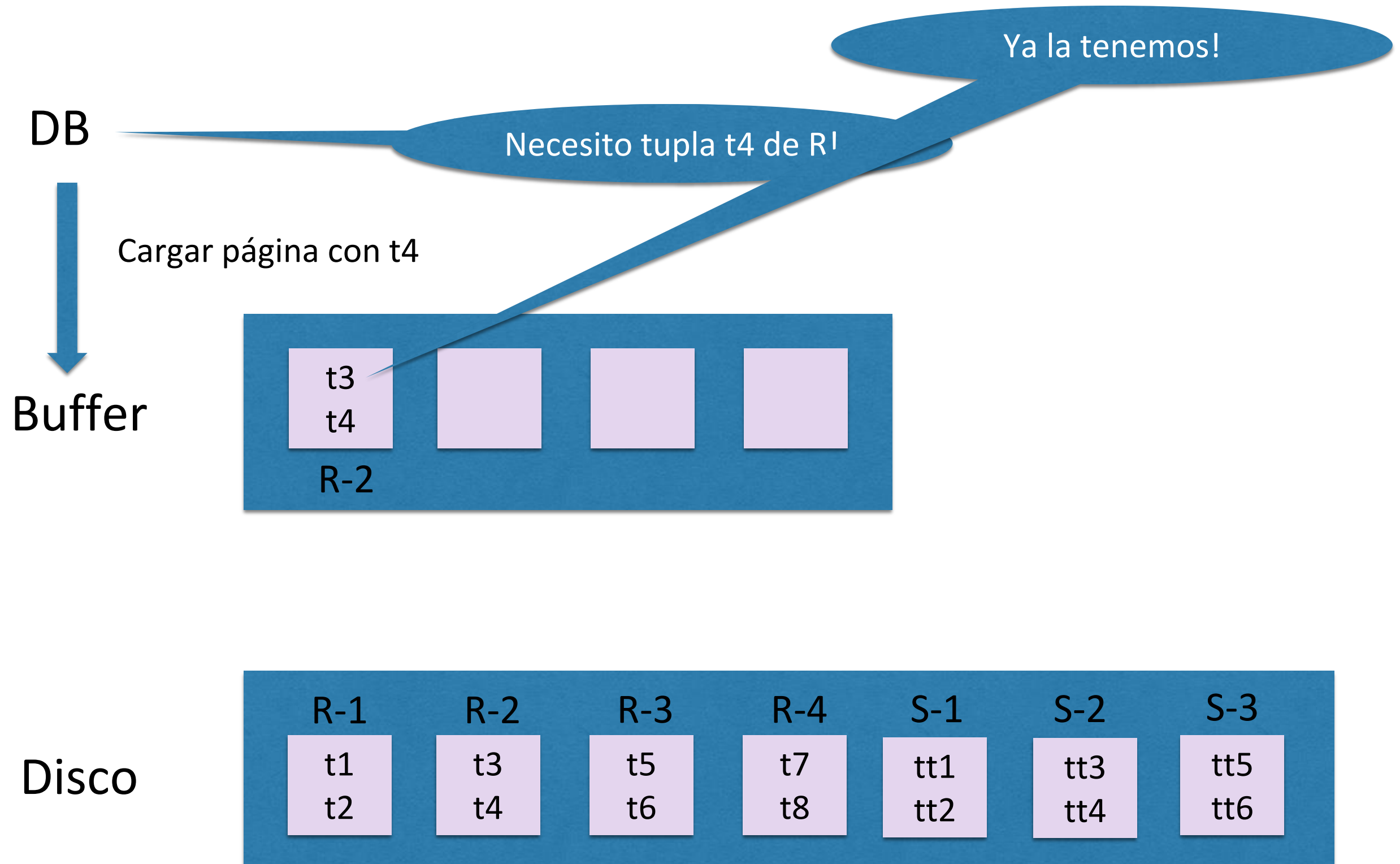


# Páginas, disco y buffer





# Páginas, disco y buffer



# Demora en Búsqueda

*“La diferencia de tiempo entre tener un dato en RAM versus traerlo de disco es comparable a la de tomar el sacapuntas del escritorio donde estoy sentado versus tomarme un avión a Punta Arenas para ir a buscarlo y regresar.”*

En general, lo más lento del DBMS es ir a buscar los datos a disco, por lo que queremos minimizar el número de I/O

# Modelo de costos

## Memoria Principal vs. Memoria Secundaria



Memoria  
Secundaria

- Datos guardados en memoria secundaria
- La lectura se hace por **páginas**
- Una página tiene un tamaño de  $B$  tuplas



Memoria  
Principal

- Los datos son llevados a memoria principal
- La memoria tiene una capacidad de  $M$  páginas

# Modelo de costos

## Memoria Principal vs. Memoria Secundaria



Memoria  
Secundaria

Se cuentan los accesos (lectura/escritura) a memoria secundaria

- Datos guardados en memoria secundaria
- La lectura se hace por **páginas**
- Una página tiene un tamaño de  $B$  tuplas



Memoria  
Principal

Los accesos a memoria principal son despreciables

- Los datos son llevados a memoria principal
- La memoria tiene una capacidad de  $M$  páginas

# Modelo de costos



¿Cuánto cuesta leer  $n$  tuplas de memoria secundaria?

$$\left\lceil \frac{n}{B} \right\rceil$$

¿Cuántas páginas usa una relación  $R$ ?

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



¿Y qué tiene que ver esto con  
Bases de Datos?

# Cómo se guarda una tabla

Supongamos una tabla  $T(a \text{ int}, b \text{ text})$  con 9 tuplas

## Disco Duro

Tupla 1	Tupla 2	Tupla 3	Tupla 4	Tupla 5	Tupla 6	Tupla 7	Tupla 8	Tupla 9
<b>Página 1</b>			<b>Página 2</b>			<b>Página 3</b>		

# Costo I/O (Input/Output)

¡El costo más grande de las bases de datos es traer las páginas del disco duro a memoria RAM!

Queremos responder las consultas haciendo la menor cantidad de lecturas al disco duro

Llamamos **costo de I/O** al número de páginas llevadas a memoria para responder una consulta



# Costo de una consulta

## Disco Duro

Tupla 1	Tupla 2	Tupla 3	Tupla 4	Tupla 5	Tupla 6	Tupla 7	Tupla 8	Tupla 9
Página 1			Página 2			Página 3		

¿Cuál es el costo en I/O de hacer la siguiente consulta?

SELECT \* FROM T

$$\left\lceil \frac{|R|}{B} \right\rceil = \left\lceil \frac{9}{3} \right\rceil$$

El costo es 3, porque debo leer las 3 páginas

# Costo de una consulta

## Disco Duro

Tupla 1	Tupla 2	Tupla 3	Tupla 4	Tupla 5	Tupla 6	Tupla 7	Tupla 8	Tupla 9
Página 1			Página 2			Página 3		

¿Cuál es el costo en I/O de hacer la siguiente consulta?

SELECT T.a FROM T

$$\left\lceil \frac{|R|}{B} \right\rceil = \left\lceil \frac{9}{3} \right\rceil$$

El costo nuevamente es 3, porque debo leer las 3 páginas

# Costo de una consulta

## Disco Duro

Tupla 1	Tupla 2	Tupla 3	Tupla 4	Tupla 5	Tupla 6	Tupla 7	Tupla 8	Tupla 9
Página 1			Página 2			Página 3		

¿Cuál es el costo en I/O de hacer la siguiente consulta?

SELECT \* FROM T WHERE T.a = 4

El costo nuevamente es 3, porque debo leer las 3 páginas

$$\left\lceil \frac{|R|}{B} \right\rceil = \left\lceil \frac{9}{3} \right\rceil$$

# Costo de una consulta

## Disco Duro

Tupla 1	Tupla 2	Tupla 3	Tupla 4	Tupla 5	Tupla 6	Tupla 7	Tupla 8	Tupla 9
Página 1			Página 2			Página 3		

¿Cuál es el costo en I/O de hacer la siguientes consultas?

SELECT \* FROM T WHERE T.a = 4

SELECT \* FROM T WHERE T.a >= 4

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

¿Podemos hacer esto mejor?

# Índices

# Índices

Herramientas que optimiza el acceso a los datos para una consulta o conjunto de consultas en particular

A un índice yo le indico que quiero las tuplas con cierto valor en un atributo (o con un valor en cierto rango)

**El índice encontrará rápidamente las tuplas que  
que cumplan con la condición**

(Idealmente un índice debe caber en RAM)

# Consultas a Optimizar

## Consultas **por valor**

```
SELECT *  
FROM Table  
WHERE Table.value = 'value'
```

## Consultas **por rango**

```
SELECT *  
FROM Table  
WHERE Table.value >= 'value'
```



# Índices

```
CREATE INDEX <nombre índice> ON table <atributo>
```

*Las llaves primarias están indexadas por defecto*

# Índices

Flujo

Supongamos que tengo la tabla:

```
Usuarios(uid: int PRIMARY KEY,  
         nombre: text,  
         edad: int)
```

en donde tenemos indexada la tabla por la primary key, que es el atributo uid

# Índices

Flujo

El índice normalmente es una colección de archivos que puede ir completo en memoria

Si hacemos la consulta:

```
SELECT * FROM Usuarios WHERE uid = 104
```

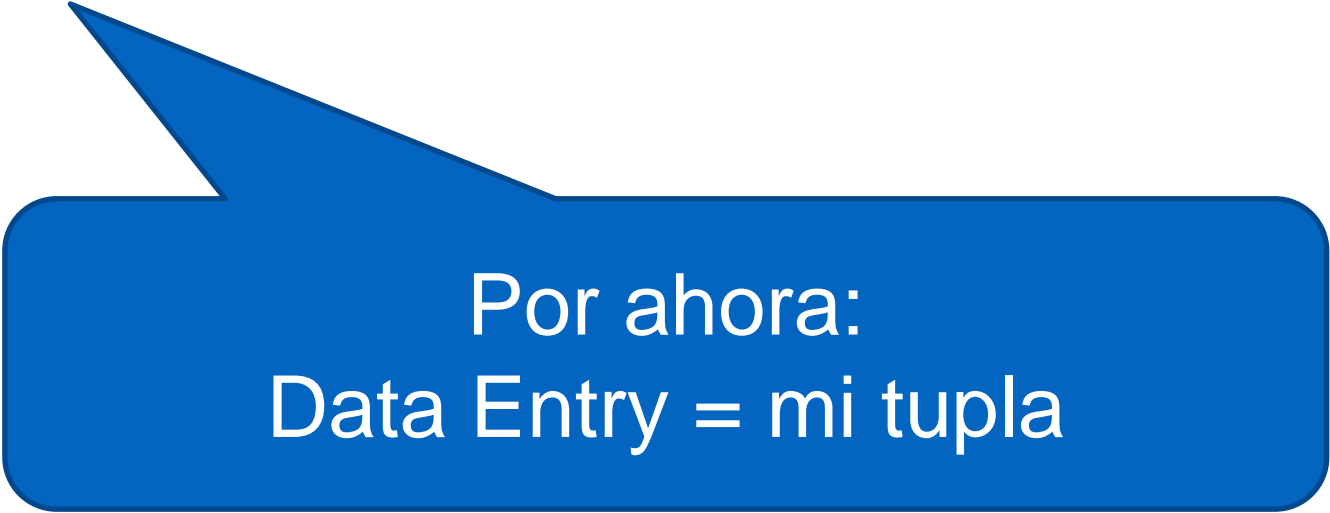
esta se resolverá con el índice, así evitaremos recorrer toda la tabla

# Índices

## Definiciones

**Search Key:** es el parámetro con el que busco algo en mi índice (por ejemplo, el uid de una tabla de usuarios)

**Data Entry:** la tupla que nos retorna el índice después de preguntar por una Search Key en particular



Por ahora:  
Data Entry = mi tupla

# Repaso (o spoiler) de EDD

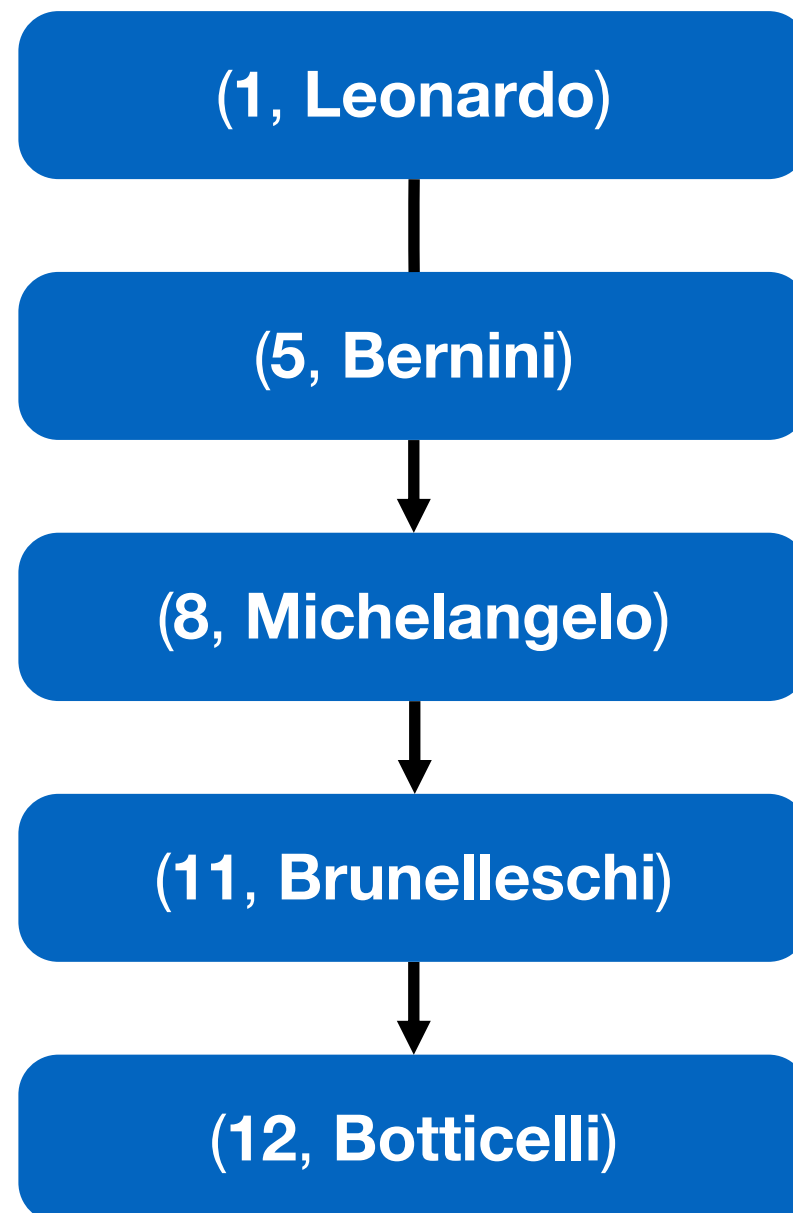
Mi esquema: **Actores(aid, nombre)**

Instancia = una lista de tuplas en Python:

```
t = [  
    (1, Leonardo),  
    (5, Bernini),  
    (8, Michelangelo),  
    (11, Brunelleschi),  
    (12, Botticelli)  
]
```

# Repaso (o spoiler) de EDD

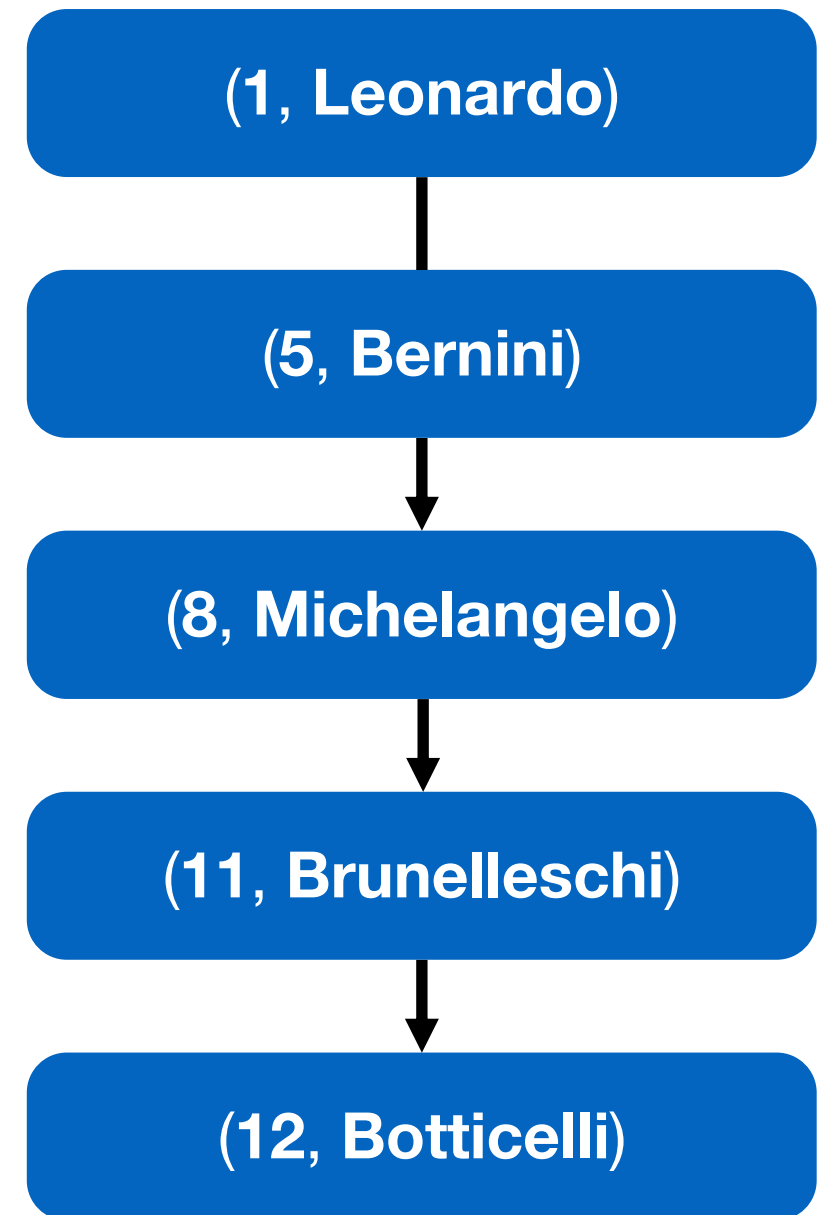
Lista ligada (versión básica)



# Repaso (o spoiler) de EDD

Lista ligada (versión básica)

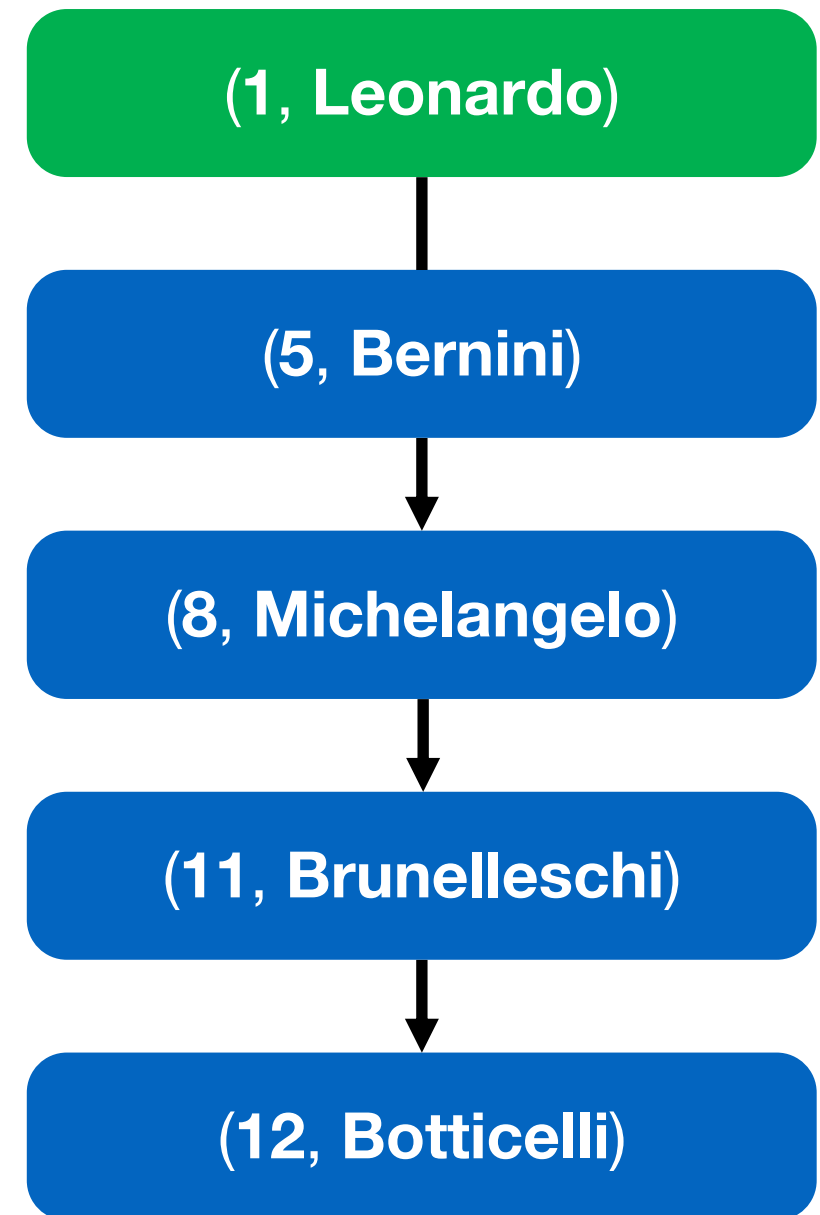
Supongamos que queremos el artista con identificador 11



# Repaso (o spoiler) de EDD

Lista ligada (versión básica)

Supongamos que queremos el artista con identificador 11

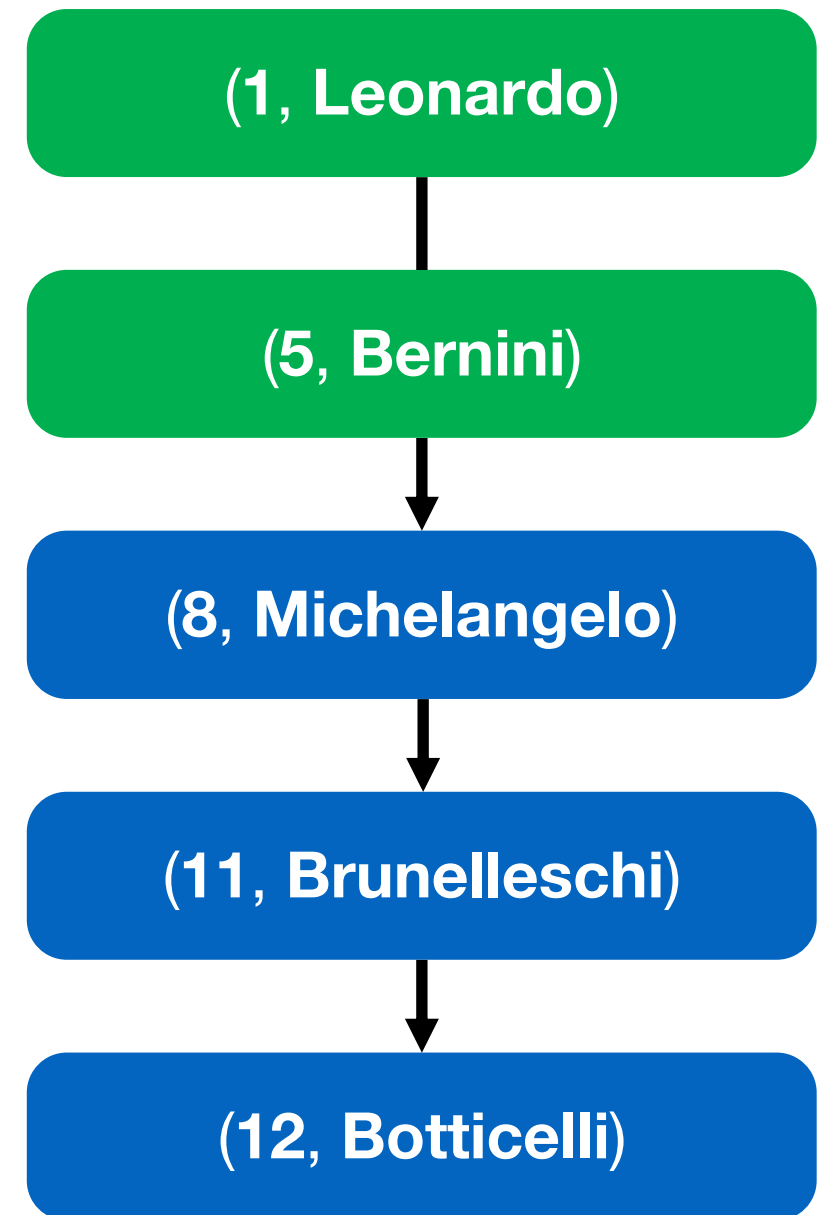




# Repaso (o spoiler) de EDD

Lista ligada (versión básica)

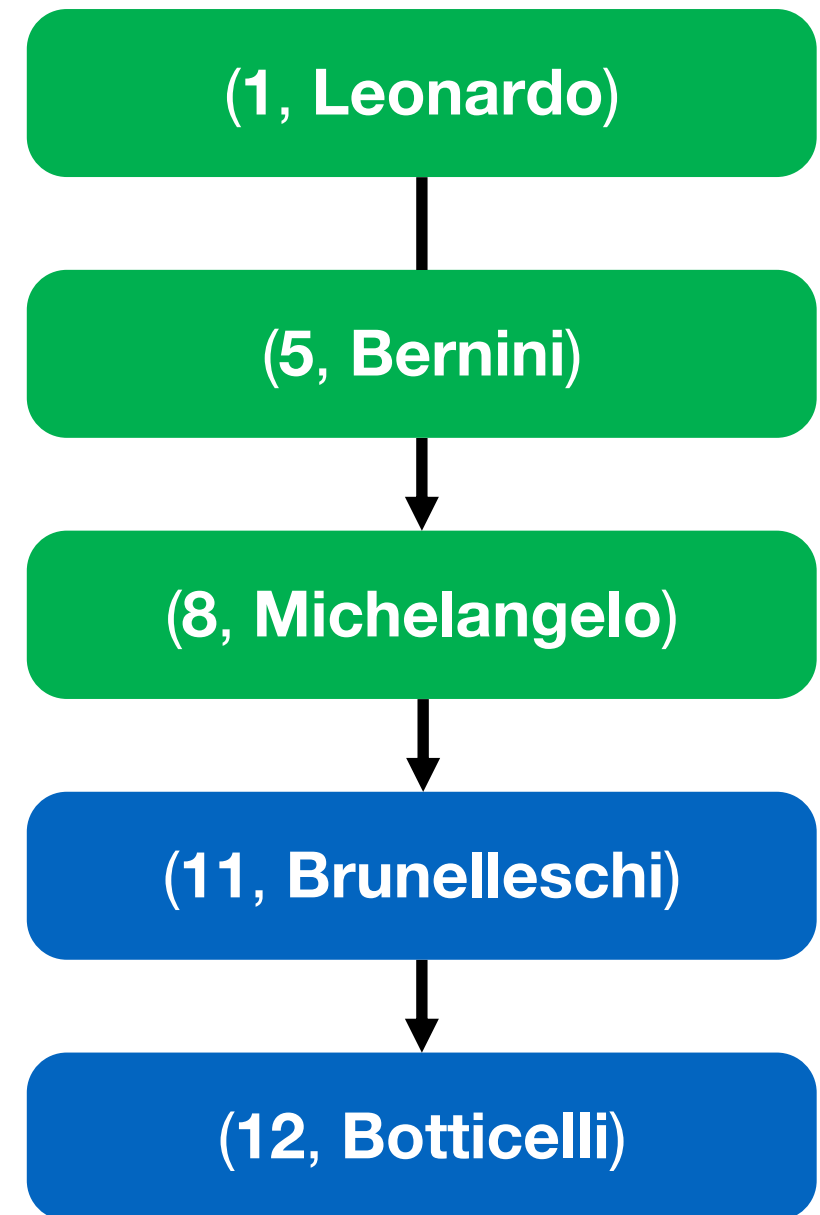
Supongamos que queremos el artista con identificador 11



# Repaso (o spoiler) de EDD

Lista ligada (versión básica)

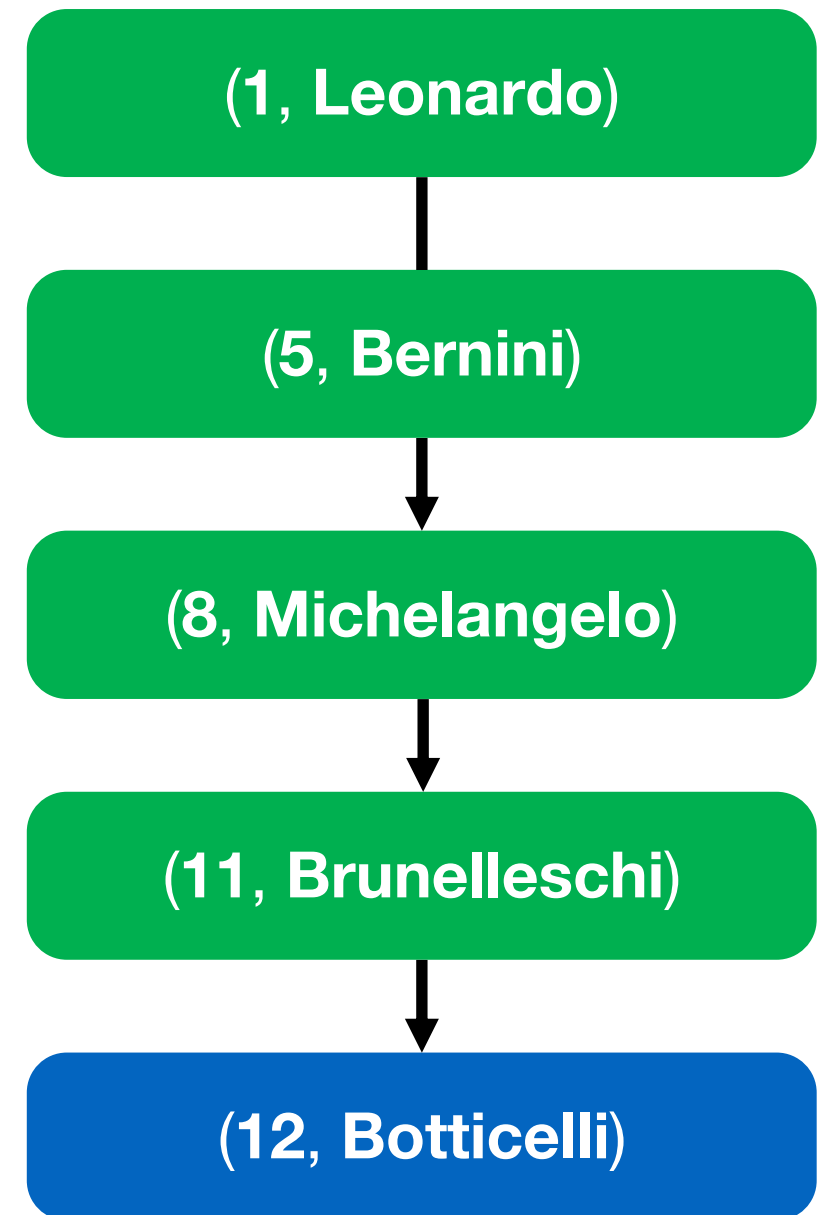
Supongamos que queremos el artista con identificador 11



# Repaso (o spoiler) de EDD

Lista ligada (versión básica)

Supongamos que queremos el artista con identificador 11



# Repaso (o spoiler) de EDD

## Tablas de Hash

Iniciamos un número fijo de casilleros (buckets)

Cada tupla va a parar a un casillero según su id

Para esto utilizamos una función de **hash**.

***Una función de hash es simplemente una función  $h$  que lleva elementos de un conjunto  $U$  a otro conjunto  $M$  (los casilleros/buckets)***

# Repaso (o spoiler) de EDD

## Tablas de Hash

En el ejemplo, podemos tener una función de **hash** que recibe el id del artista y retorna un número de casillero

Si tenemos 5 casilleros, una posible función es módulo 5 (retorna el resto de la división por 5)

# Repaso (o spoiler) de EDD

Tablas de Hash



Función hash:  
módulo 5 ( $aid \% 5$ )

Bucket	
0	
1	
2	
3	
4	

# Repaso (o spoiler) de EDD

Tablas de Hash

(1, Leonardo)



Función hash:  
módulo 5 ( $\text{aid} \% 5$ )

Bucket	
0	
1	
2	
3	
4	

# Repaso (o spoiler) de EDD

Tablas de Hash

(5, Bernini)



Función hash:  
módulo 5 ( $aid \% 5$ )

Bucket	
0	
1	(1, Leonardo)
2	
3	
4	



# Repaso (o spoiler) de EDD

## Tablas de Hash



Función hash:  
módulo 5 ( $\text{aid} \% 5$ )

Bucket	
0	(5, Bernini)
1	(1, Leonardo)
2	
3	
4	

# Repaso (o spoiler) de EDD

## Tablas de Hash

(8, Michelangelo)

(11, Brunelleschi)

(12, Botticelli)



Función hash:  
módulo 5 ( $aid \% 5$ )

Bucket	
0	(5, Bernini)
1	(1, Leonardo)
2	
3	
4	

# Repaso (o spoiler) de EDD

## Tablas de Hash



Función hash:  
módulo 5 ( $aid \% 5$ )

Bucket		
0	(5, Bernini)	
1	(1, Leonardo)	(11, Brunelleschi)
2	(12, Botticelli)	
3	(8, Michelangelo)	
4		

# Repaso (o spoiler) de EDD

Tablas de Hash

Si queremos buscar el artista con identificador 12, ¿cómo lo hacemos ahora? ¿en cuántos pasos lo obtenemos?

# Repaso (o spoiler) de EDD

## Tablas de Hash

aid = 12



Función hash:  
módulo 5 ( $\text{aid} \% 5$ )

Bucket		
0	(5, Bernini)	
1	(1, Leonardo)	(11, Brunelleschi)
2	(12, Botticelli)	
3	(8, Michelangelo)	
4		

Computamos el hash del número 12

# Repaso (o spoiler) de EDD

## Tablas de Hash

aid = 12



Función hash:  
módulo 5 ( $\text{aid} \% 5$ )



Bucket		
0	(5, Bernini)	
1	(1, Leonardo)	(11, Brunelleschi)
2	(12, Botticelli)	
3	(8, Michelangelo)	
4		

Y en 1 paso encontramos el casillero que le corresponde

# Repaso (o spoiler) de EDD

Tablas de Hash

Notamos que hay colisiones, esto es, dos valores que van a parar al mismo casillero

En general, suponemos que las funciones de hash distribuyen uniforme

También suponemos que hay suficientes casilleros para que la búsqueda se haga en 1 paso (o no mucho más)

# Hash Index

La idea es replicar el concepto de las tablas de hash pero en un sistema de bases de datos

Aquí nuestros casilleros serán páginas del disco duro

En cada página caben muchas tuplas



# Hash Index

Recordemos que queremos encontrar tuplas para consultas de igualdad de forma rápida

```
SELECT * FROM Artistas A WHERE A.aid = 12
```

En este caso, rápido significa hacer la menor cantidad de I/O posible

# Hash Index

Ejemplo

Vamos a retomar el ejemplo de los artistas, pero ahora insertando las tuplas a una base de datos

Artista.aid	Artista.nombre
1	Leonardo
5	Bernini
8	Michelangelo
11	Brunelleschi
12	Botticelli
16	Giotto

# Hash Index

Ejemplo

Artista.aid	Artista.nombre
1	Leonardo
5	Bernini
8	Michelangelo
11	Brunelleschi
12	Botticelli
16	Giotto

0



1



2



3



4

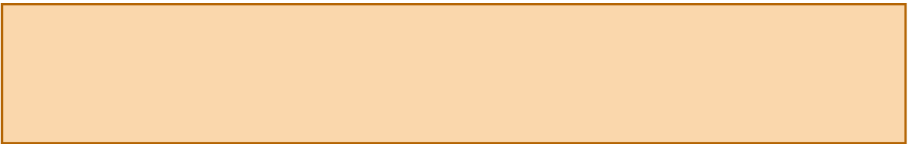


# Hash Index

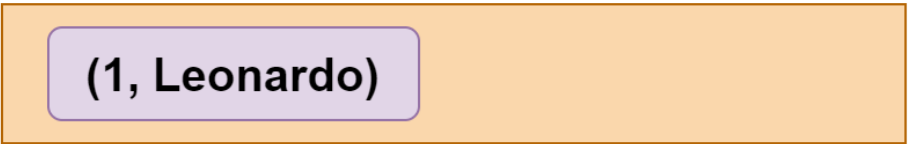
Ejemplo

Artista.aid	Artista.nombre
1	Leonardo
5	Bernini
8	Michelangelo
11	Brunelleschi
12	Botticelli
16	Giotto

0



1



2



3



4



# Hash Index

Ejemplo

Artista.aid	Artista.nombre
1	Leonardo
5	Bernini
8	Michelangelo
11	Brunelleschi
12	Botticelli
16	Giotto

0

(5, Bernini)

1

(1, Leonardo)

2

3

4

# Hash Index

Ejemplo

Artista.aid	Artista.nombre
1	Leonardo
5	Bernini
8	Michelangelo
11	Brunelleschi
12	Botticelli
16	Giotto

0

(5, Bernini)

1

(1, Leonardo)

2

3

(8, Michelangelo)

4

# Hash Index

Ejemplo

Artista.aid	Artista.nombre
1	Leonardo
5	Bernini
8	Michelangelo
11	Brunelleschi
12	Botticelli
16	Giotto

0

(5, Bernini)

1

(1, Leonardo)

(11, Brunelleschi)

2

3

(8, Michelangelo)

4

# Hash Index

Ejemplo

Artista.aid	Artista.nombre
1	Leonardo
5	Bernini
8	Michelangelo
11	Brunelleschi
12	Botticelli
16	Giotto

0

(5, Bernini)

Overflow

1

(1, Leonardo)

(11, Brunelleschi)

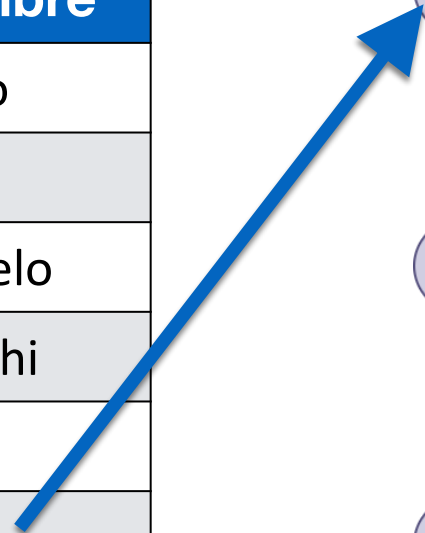
2

(12, Botticelli)

3

(8, Michelangelo)

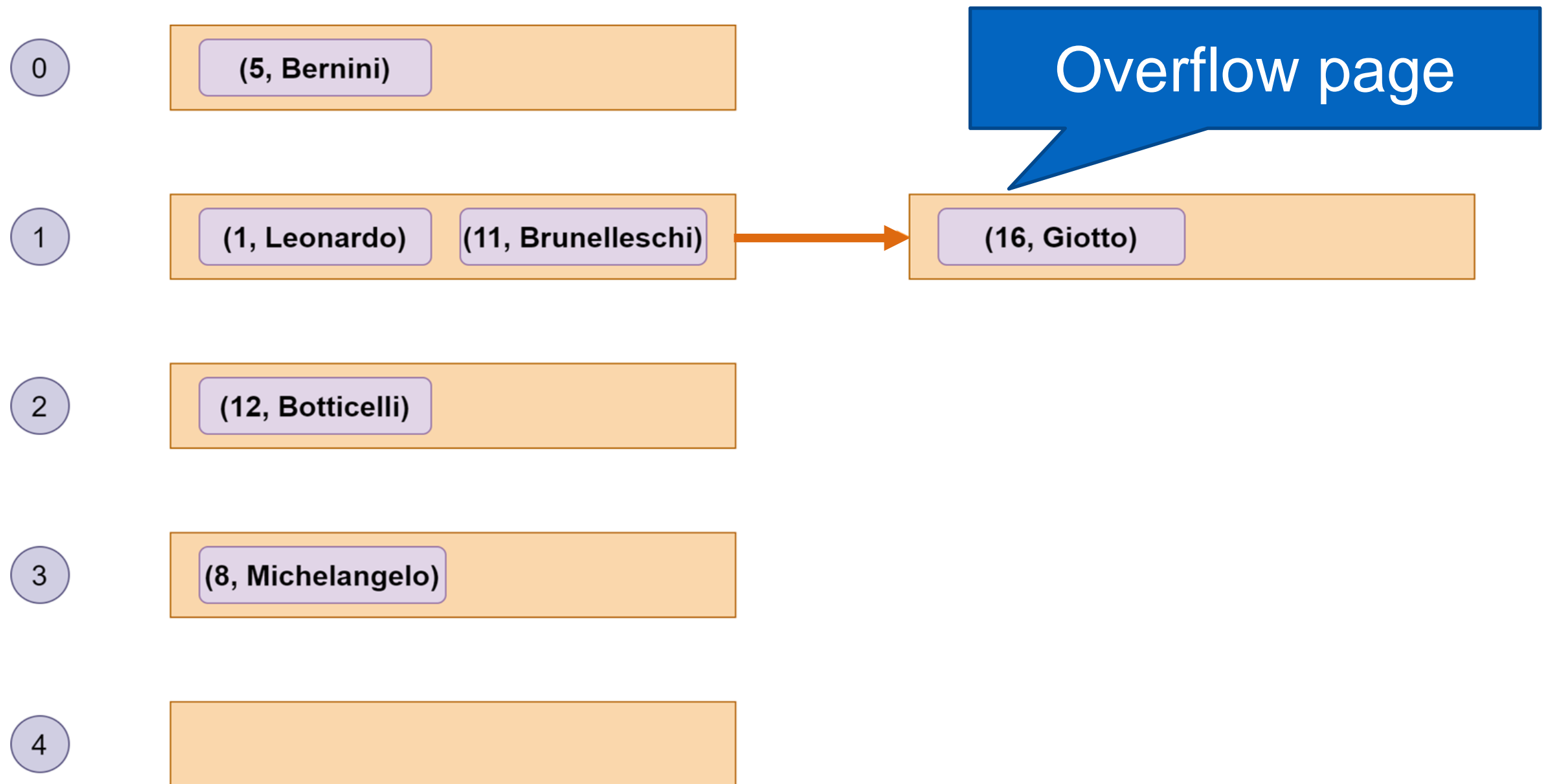
4





# Hash Index

Ejemplo



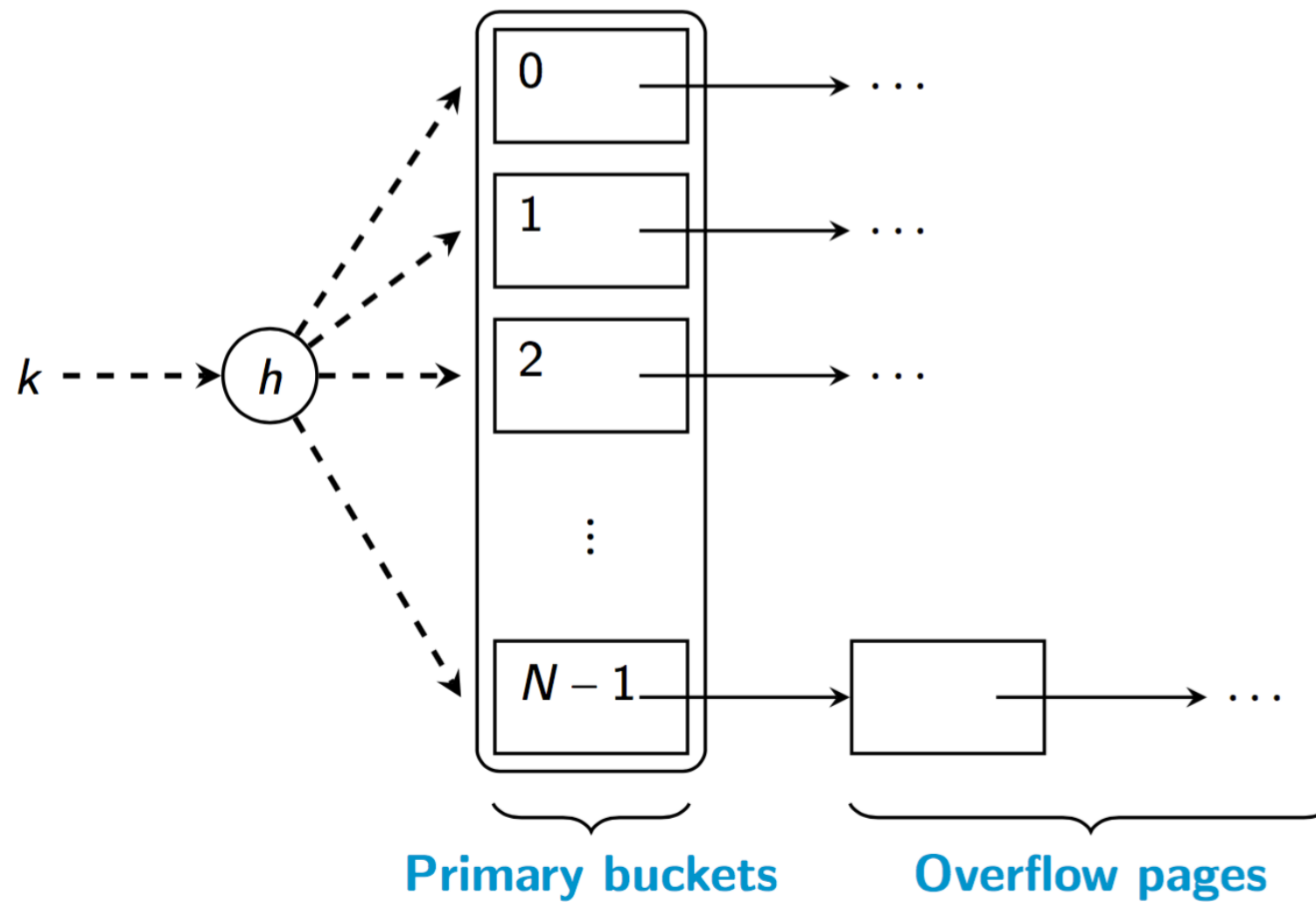
# Hash Index

Para una relación **R** y un atributo **A**

- N páginas de disco sirven de **buckets**
- Cada bucket cuenta con una lista ligada de **overflow pages**
- Usamos una función de hash **h** que me indicará que bucket le corresponde a cada valor:

$$h: \text{dominio}(A) \rightarrow [0, \dots, N - 1]$$

# Hash Index



# Hash Index

Para buscar un elemento dada una search key **k**:

- Se computa el valor de  **$h(k)$**
- Se accede al bucket correspondiente
- Se busca el elemento en la página asociada o una de sus overflow pages

costo  $O(1)$

Caso ideal: costo 0

Peor caso: costo  $O\left(\left\lceil \frac{|R|}{B} \right\rceil\right)$

Cuando todo va a un casillero!

# Hash Index

¿Cuándo **no** nos conviene utilizar un Hash Index?

```
SELECT * FROM Artistas A WHERE A.aid > 12
```

Efectivamente:  $\left\lceil \frac{|R|}{B} \right\rceil$

# B+ Tree Index

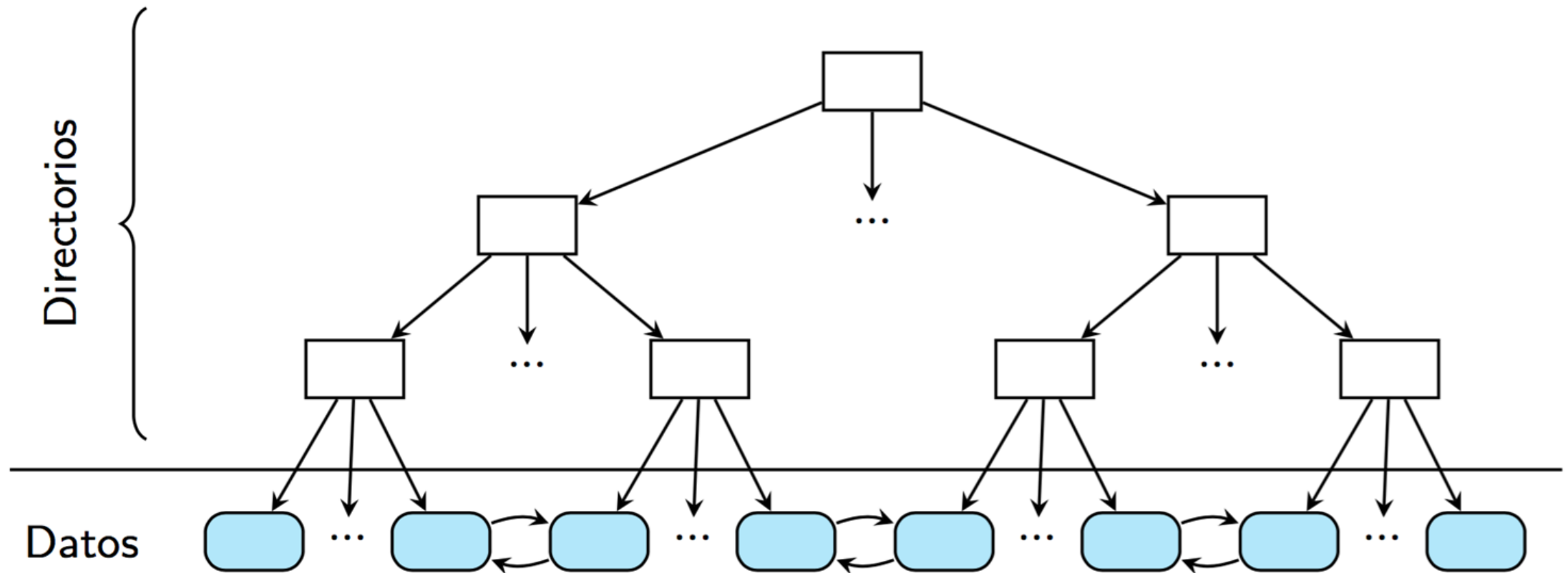
*“B+ Trees are by far the most important access path structure in database and file systems”,  
Gray y Reuter (1993).*

# B+ Tree Index

Es un índice basado en un árbol:

- Las páginas del disco representan nodos del árbol
- **Las tuplas** están en las hojas
- Provee un tiempo de búsqueda logarítmico
- Se comporta bien en consultas de rango
- Es el índice que usa Postgres sobre las llaves primarias (y en general, todos los sistemas)

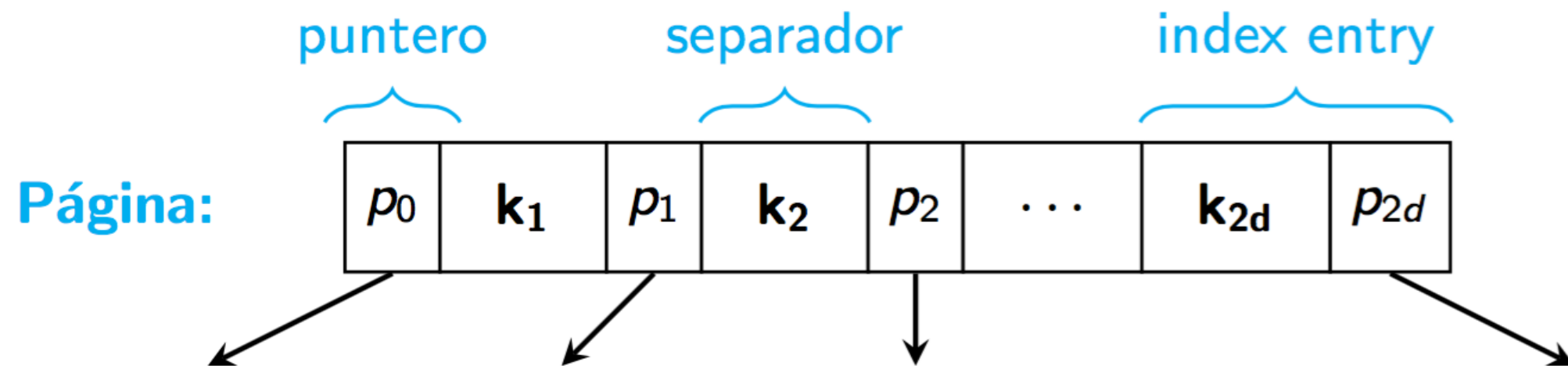
# B+ Tree Index





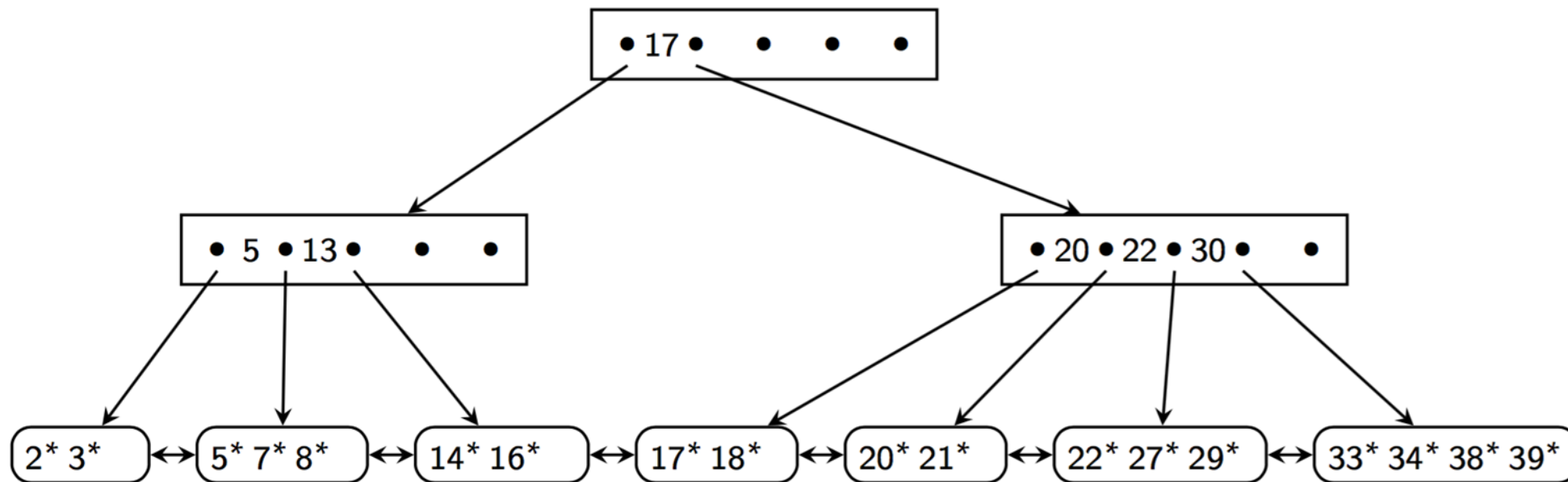
# B+ Tree Index

Cada nodo del directorio tiene la siguiente forma:



# B+ Tree Index

En los directorios, los nodos tienen una **Search Key** y un puntero antes y después de ella

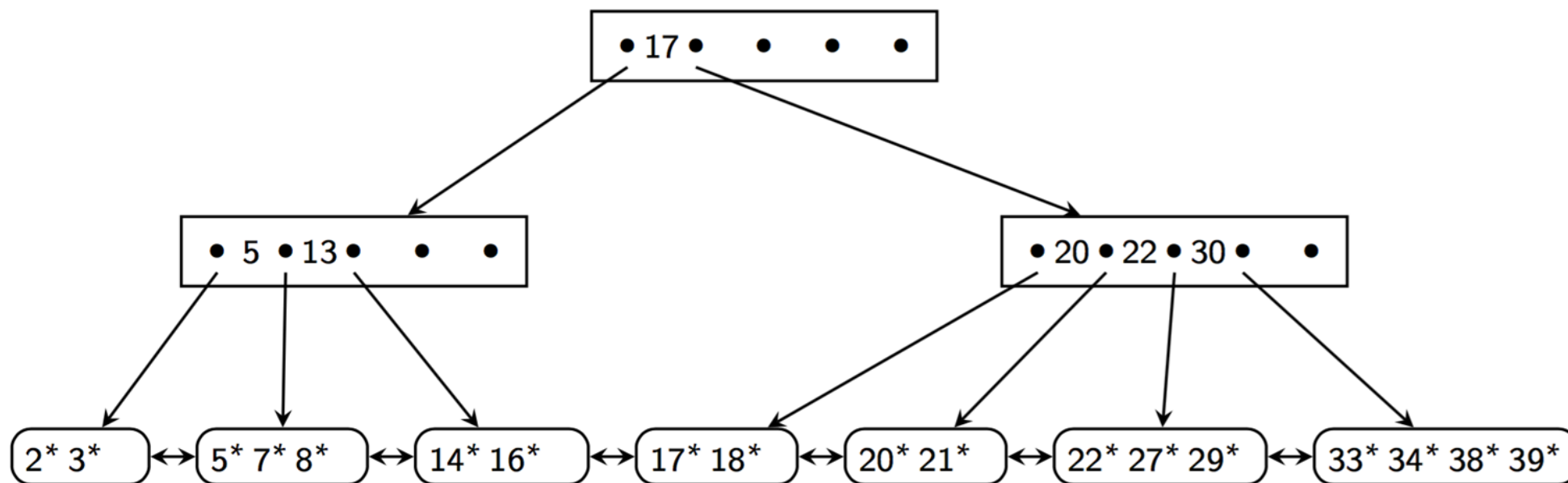


# B+ Tree Index

Ejemplo: consulta por un valor

Supongamos que este índice también almacena artistas que están indexados por su aid

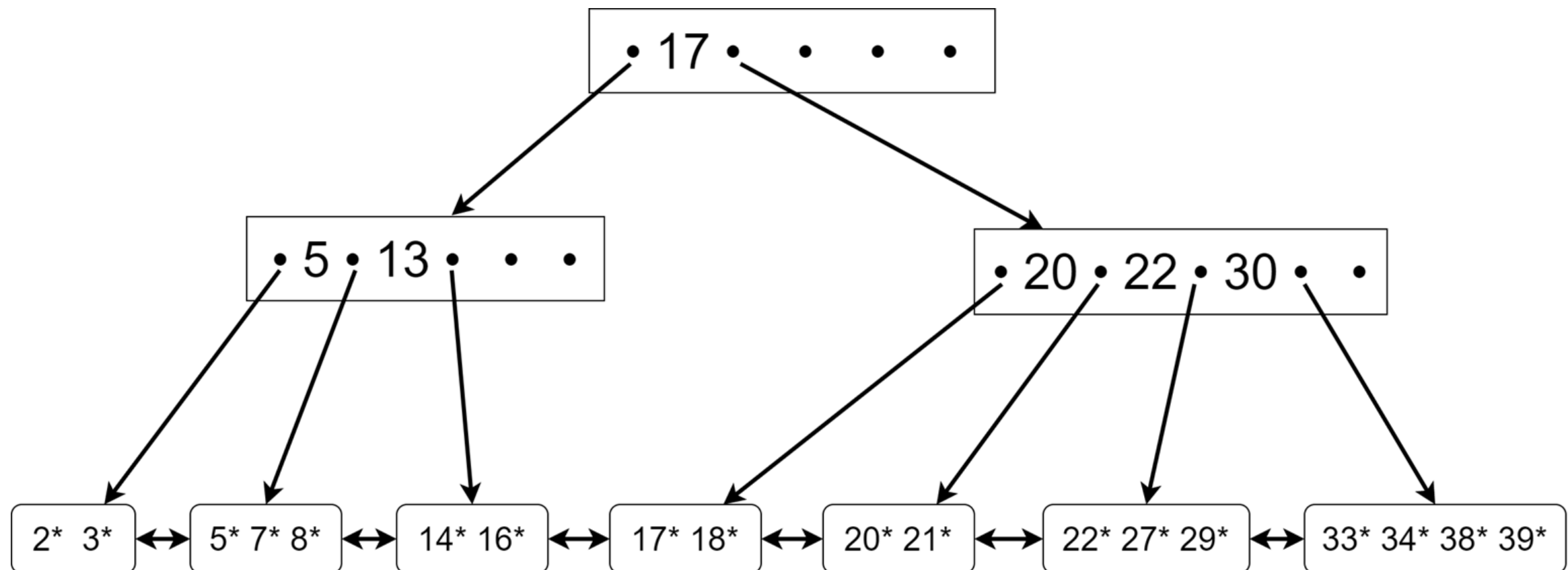
Queremos el artista con aid = 21



# B+ Tree Index

Ejemplo: consulta por un valor

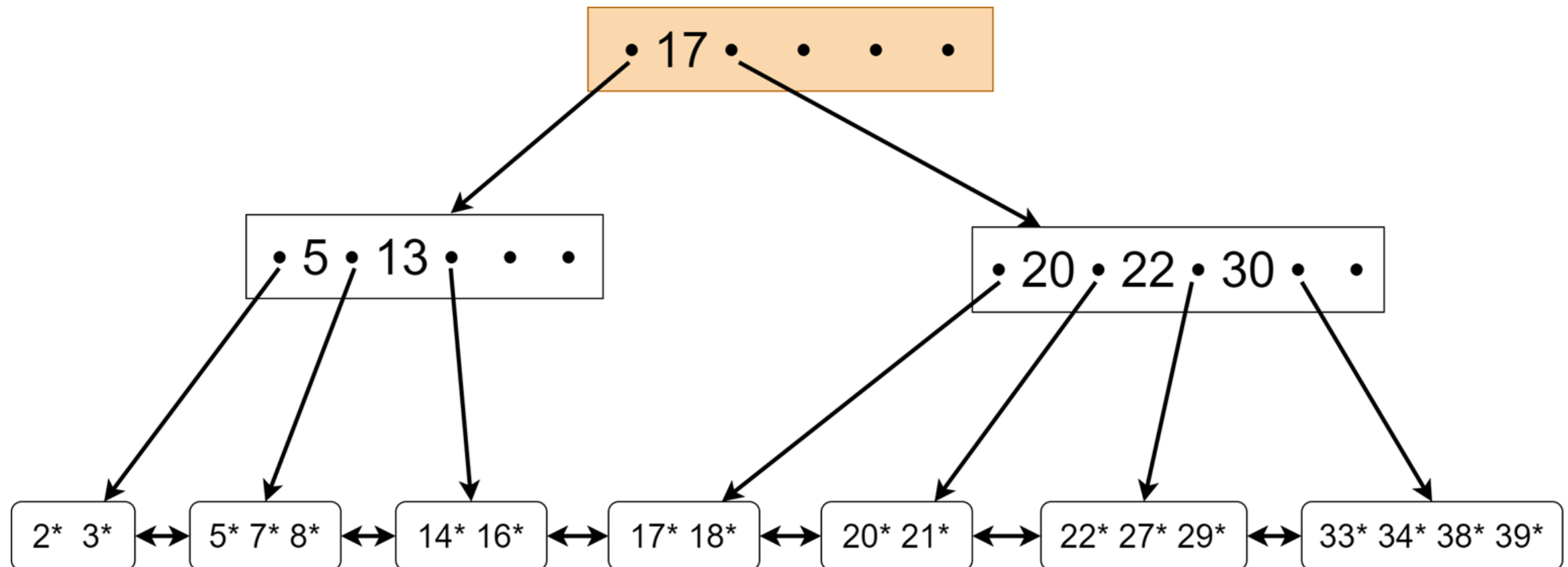
Queremos el artista con aid = 21



# B+ Tree Index

Ejemplo: consulta por un valor

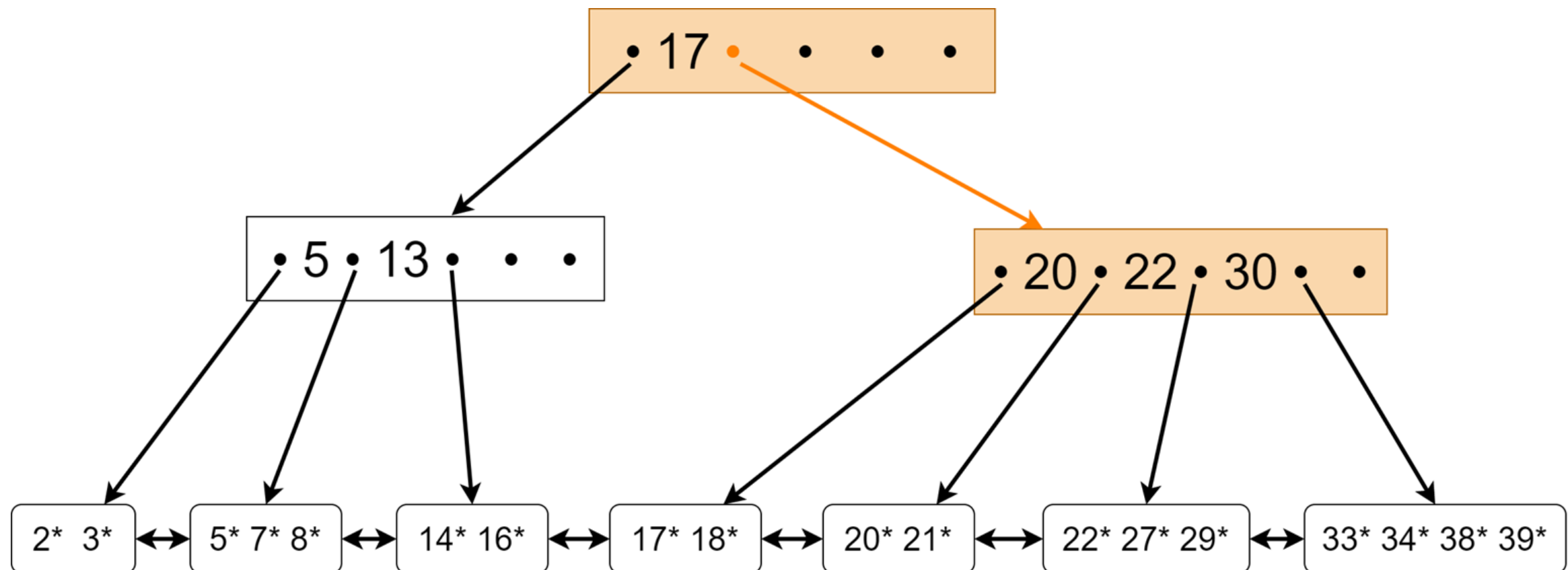
Queremos el artista con aid = 21



# B+ Tree Index

Ejemplo: consulta por un valor

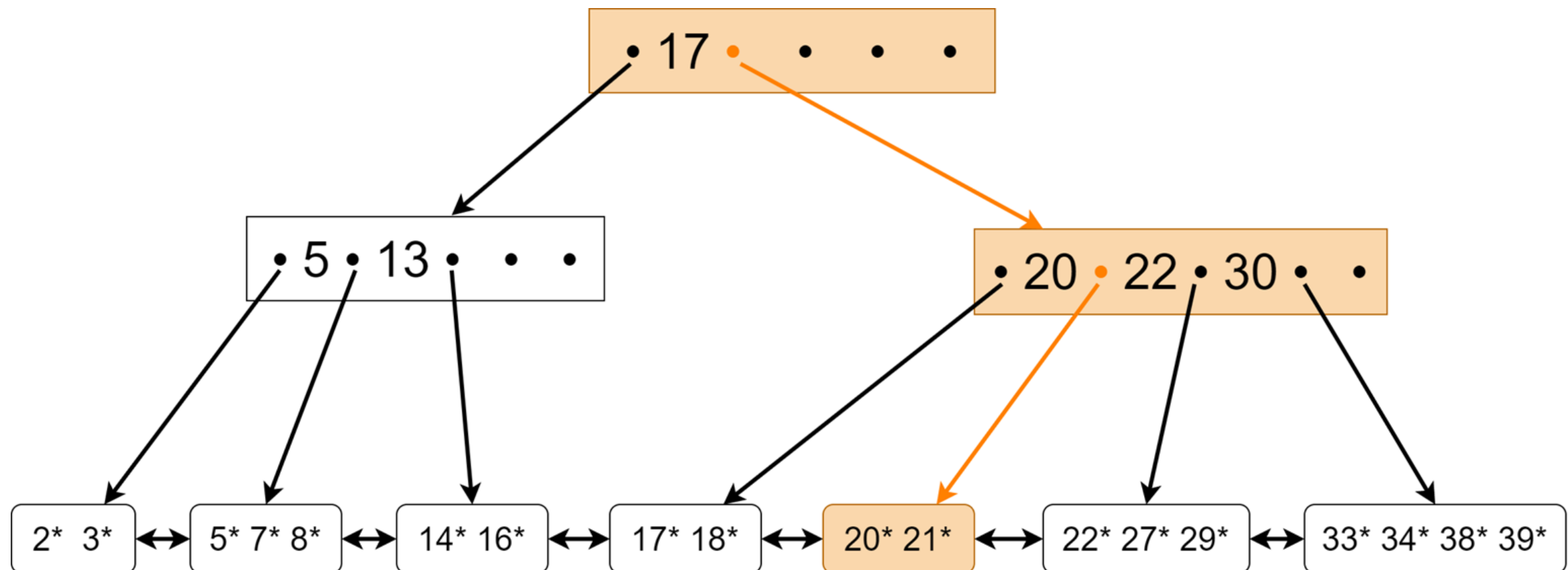
Queremos el artista con aid = 21



# B+ Tree Index

Ejemplo: consulta por un valor

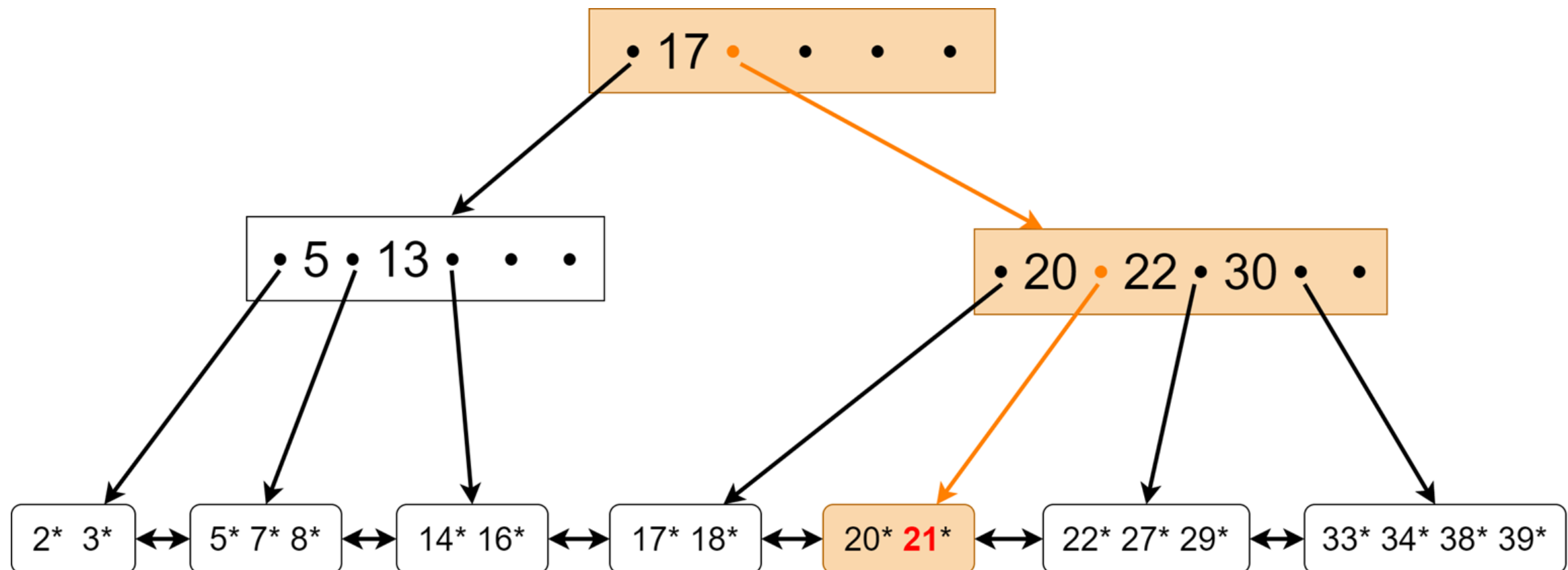
Queremos el artista con aid = 21



# B+ Tree Index

Ejemplo: consulta por un valor

Queremos el artista con aid = 21

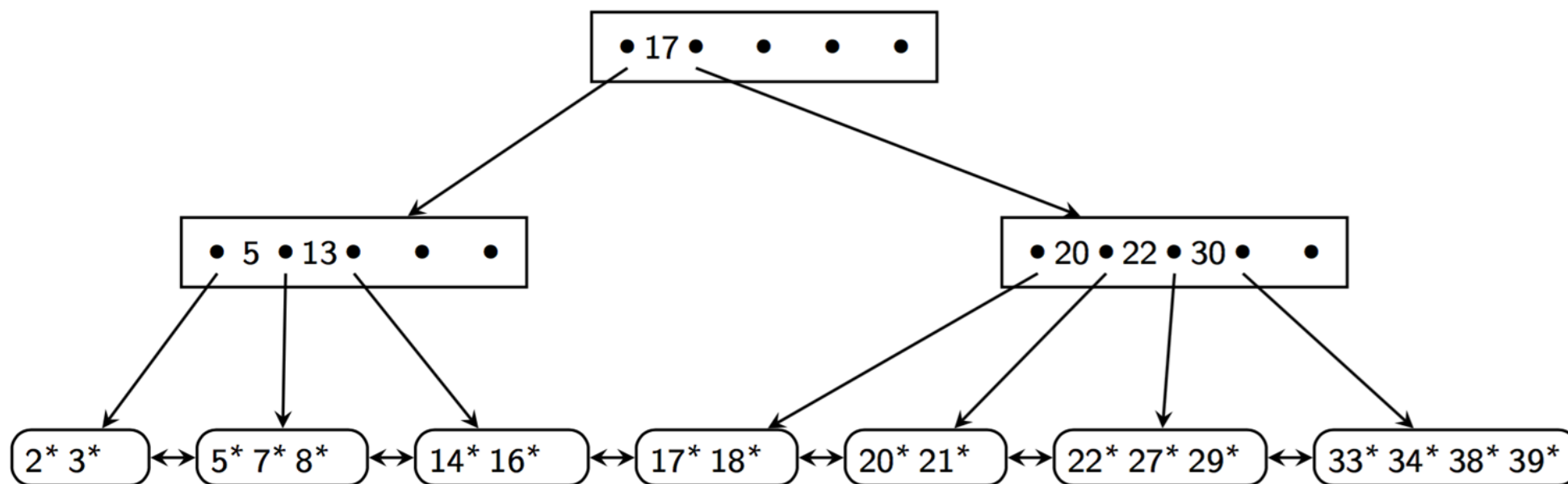




¿Y por qué este índice me sirve  
en consultas de rango?

# B+ Tree Index

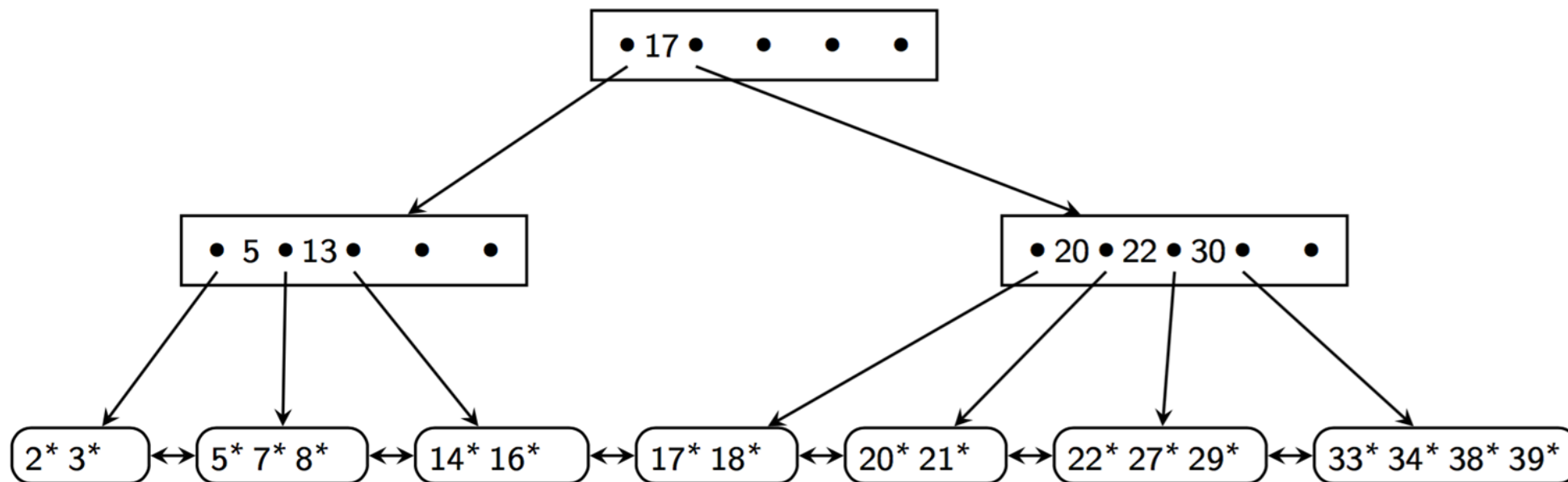
Sabemos que las tuplas van a estar ordenadas según el atributo indexado, y además las hojas están encadenadas unas con otras



# B+ Tree Index

Ejemplo: consulta de rango

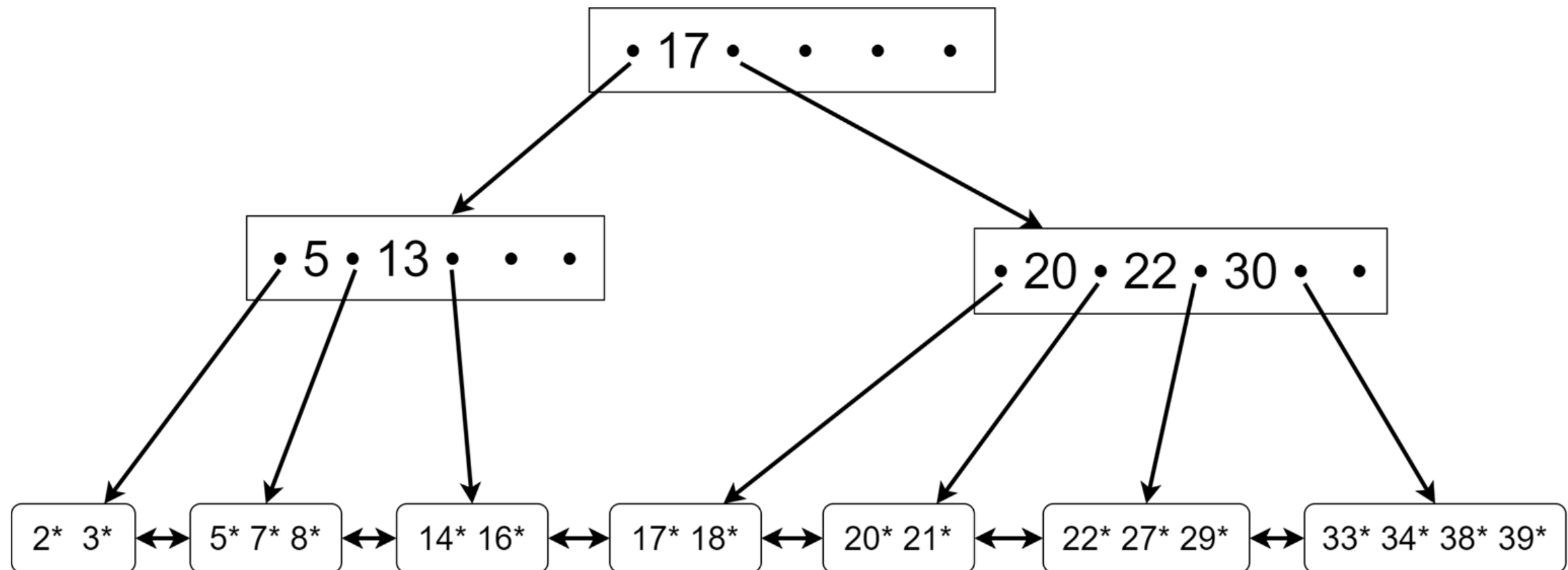
Ahora queremos hacer la consulta de todos los Artistas con aid entre 5 y 26



# B+ Tree Index

Ejemplo: consulta de rango

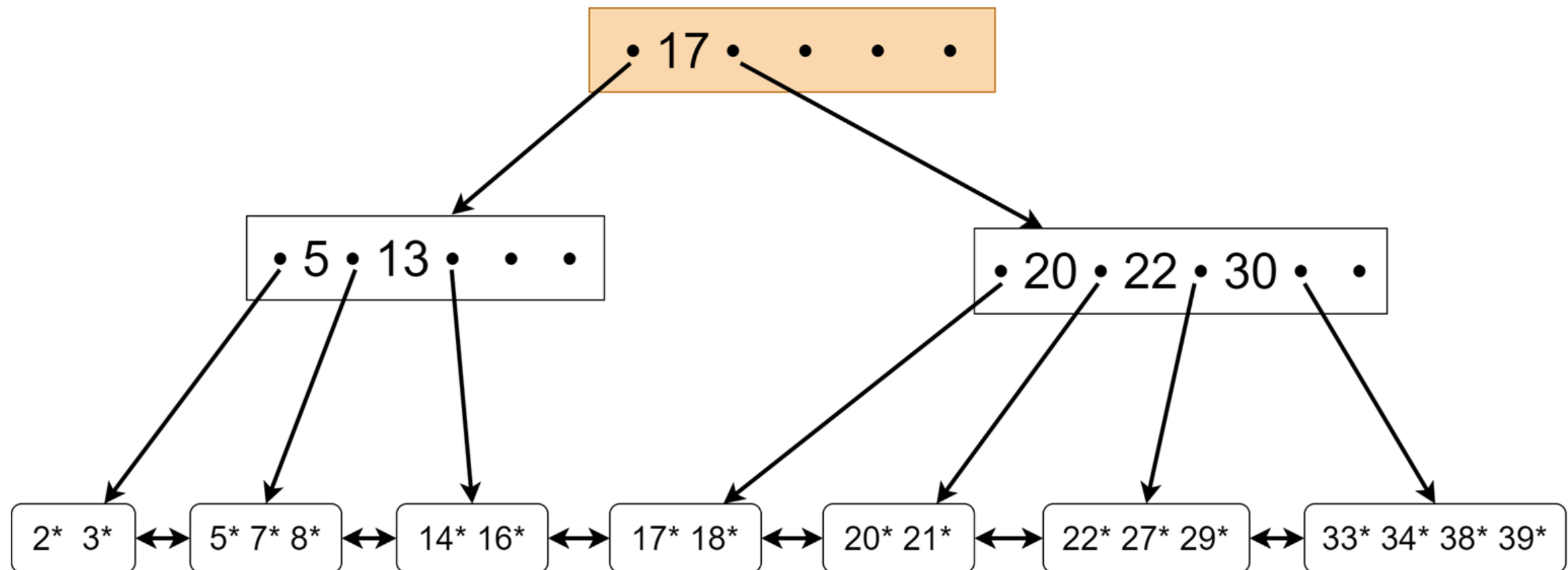
Todos los Artistas con aid entre 5 y 26



# B+ Tree Index

Ejemplo: consulta de rango

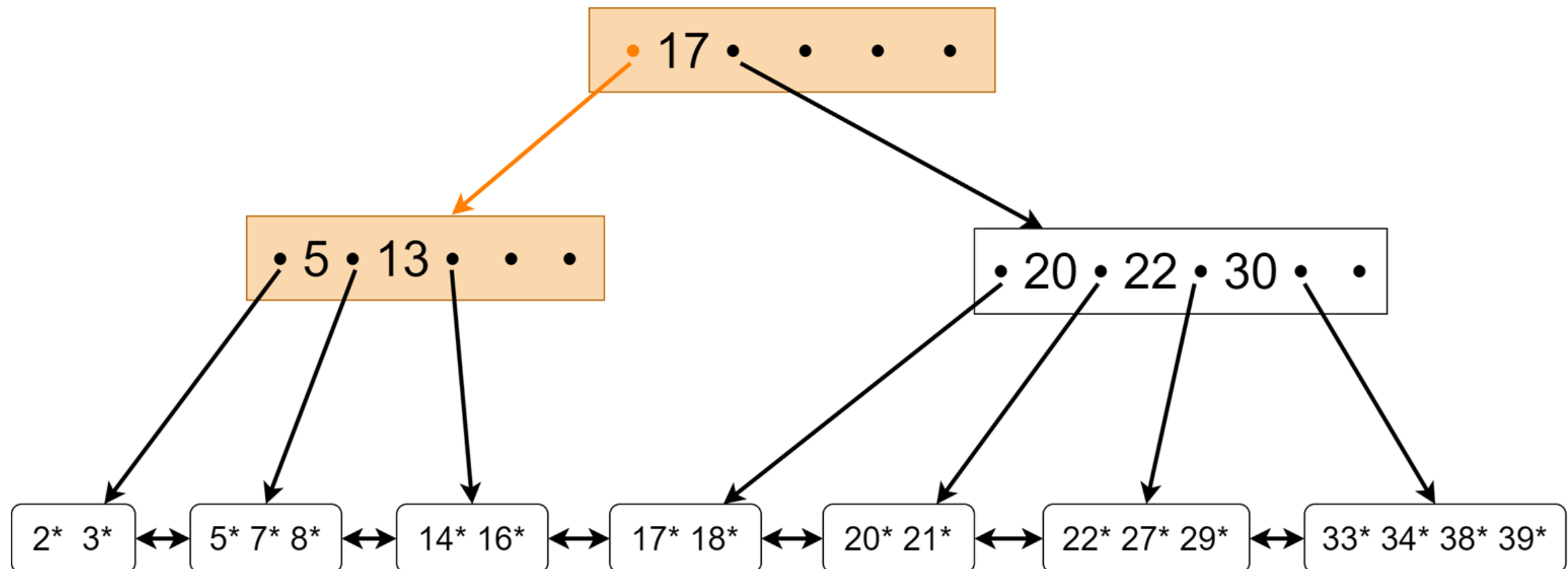
Todos los Artistas con aid entre 5 y 26



# B+ Tree Index

Ejemplo: consulta de rango

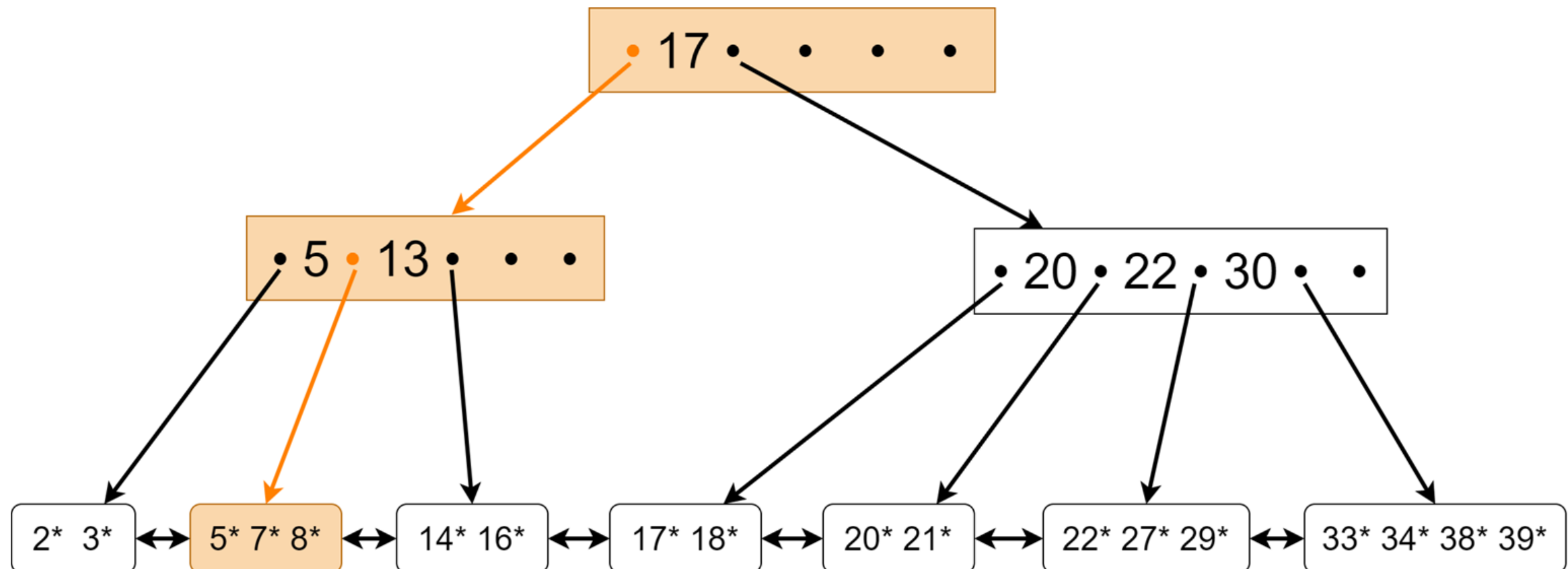
Todos los Artistas con aid entre 5 y 26



# B+ Tree Index

Ejemplo: consulta de rango

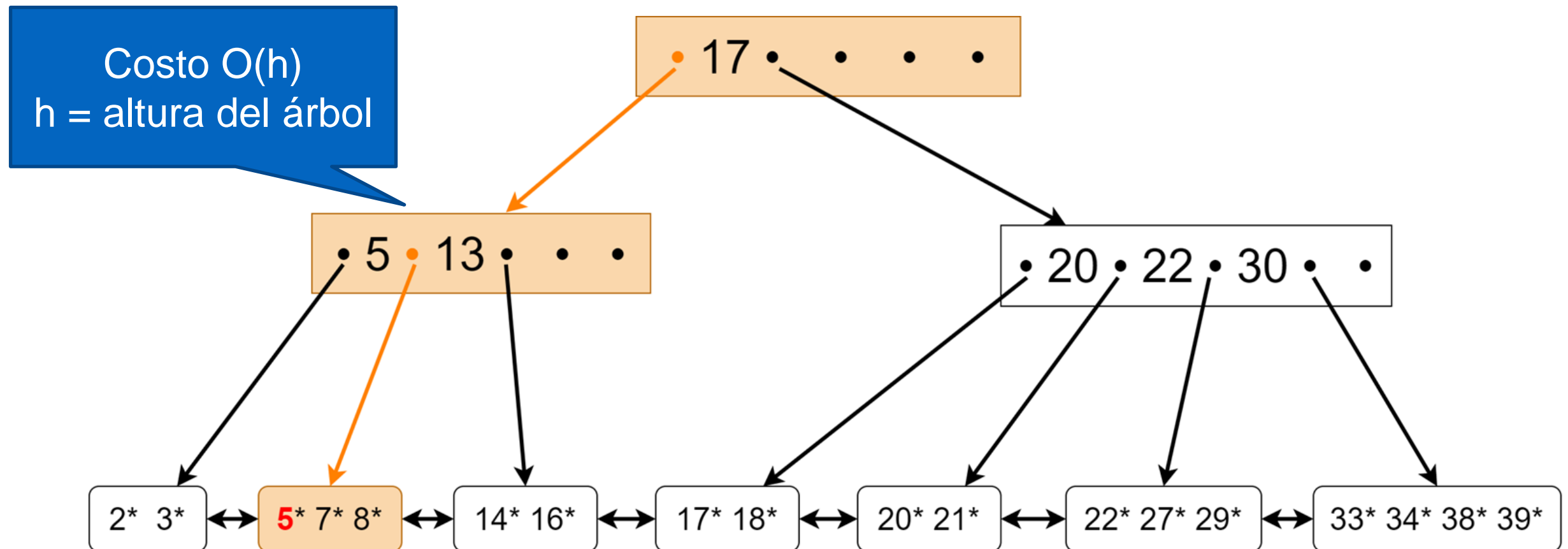
Todos los Artistas con aid entre 5 y 26



# B+ Tree Index

Ejemplo: consulta de rango

Todos los Artistas con aid entre 5 y 26

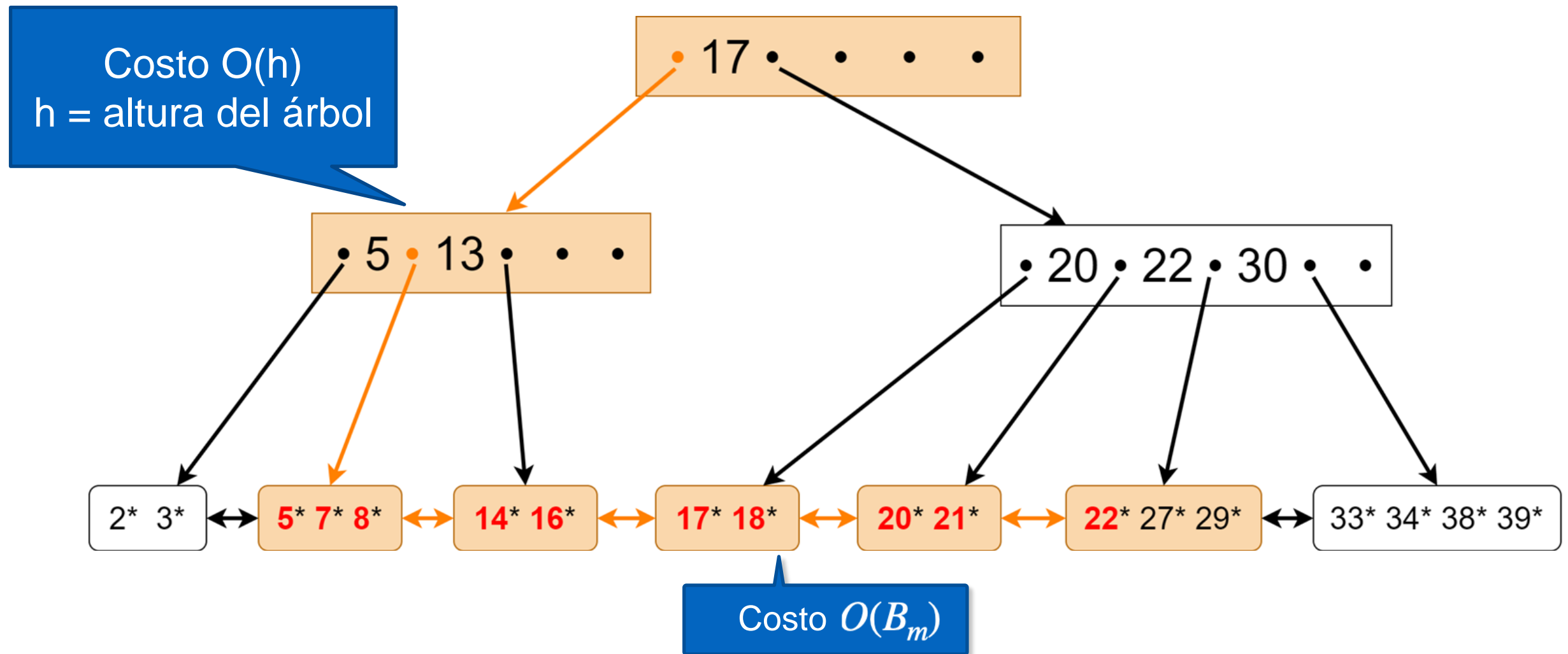




# B+ Tree Index

Ejemplo: consulta de rango

Todos los Artistas con aid entre 5 y 26



$B_m$  : nr de páginas con records que calzan búsqueda

# B+ Tree Index

**Importante:** para facilitar inserción de datos nuevos los B+ trees no tienen nodos completamente llenos

Típicamente se dice que las hojas están ocupadas a un porcentaje; e.g. 60% -- quiere decir si caben 100 tuplas hay solo 60 en una página de la hoja

# B+ Tree Index

El costo de búsqueda es logarítmico  $\sim \log(|R|)$

Si **B** es el tamaño de una página en tuplas, el costo de búsqueda es:

$$O(\log_B(\left\lceil \frac{|R|}{B} \right\rceil))$$

¿Qué pasa si el índice lo mantenemos en memoria RAM?

# Tipos de valores a guardar

**Un índice me puede retornar:**

- *La tupla* que estoy buscando
- Un *puntero* o lista de punteros a las páginas del disco duro que tienen las tuplas que estoy buscando


# Clustered y Unclustered Index



Data entry:  
La tupla misma

Hablaremos de un índice **Clustered** si al preguntarle por un valor me retorna una tupla

Hablaremos de un índice **Unclustered** si me retorna un puntero



Data entry:  
Puntero a la página  
con mi tupla

# Clustered y Unclustered Index

**Clustered Index** se conoce como índice primario

**Unclustered Index** se conoce como índice secundario

# Clustered y Unclustered Index

Considere Empleados(id, nombre, edad)

Un índice primario en general se aplica sobre la primary key, por ejemplo el id de los empleados

Un índice secundario se aplica sobre un atributo como edad (para ordenar por edad más rápido)

Buscando una tupla con  
distintos tipos de índices



# Hash Index clustered

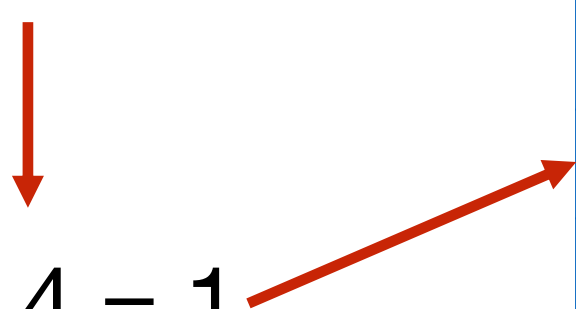
$R(\underline{a}, b, c)$  ... índice sobre  $a$

$a = 5$

$\downarrow$

$5 \% 4 = 1$

Bucket	
0	(4,1,1), (0,2,3), (8,1,2)
1	(1,2,2), (5,1,1)
2	(2,3,3)
3	(3,1,2), (7,8,9)



# Hash Index clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$

$a = 5$

$5 \% 4 = 1$

Bucket	
0	(4,1,1), (0,2,3), (8,1,2)
1	(1,2,2), (5,1,1) <span>página del disco</span>
2	(2,3,3)
3	(3,1,2), (7,8,9)

# Hash Index clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$

$a = 5$

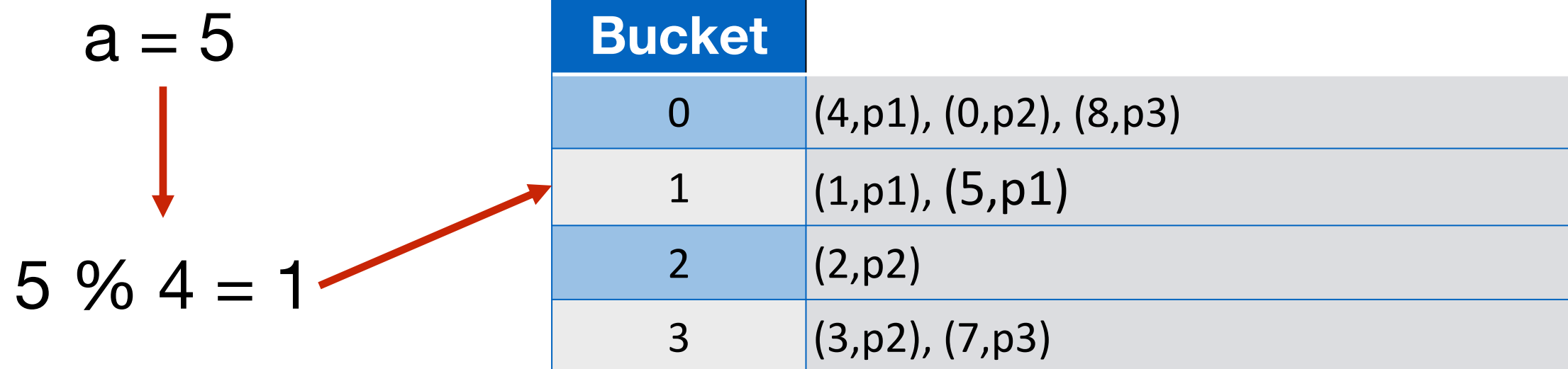
$5 \% 4 = 1$

Bucket	
0	(4,1,1), (0,2,3), (8,1,2)
1	(1,2,2), (5,1,1) <span>página del disco</span>
2	(2,3,3)
3	(3,1,2), (7,8,9)

$\#(I/O) = 1$  (el bucket es la página con mi tupla)

# Hash Index unclustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$



Datos en el disco:

p1

(4,1,1), (1,2,2), (5,1,1)

p2

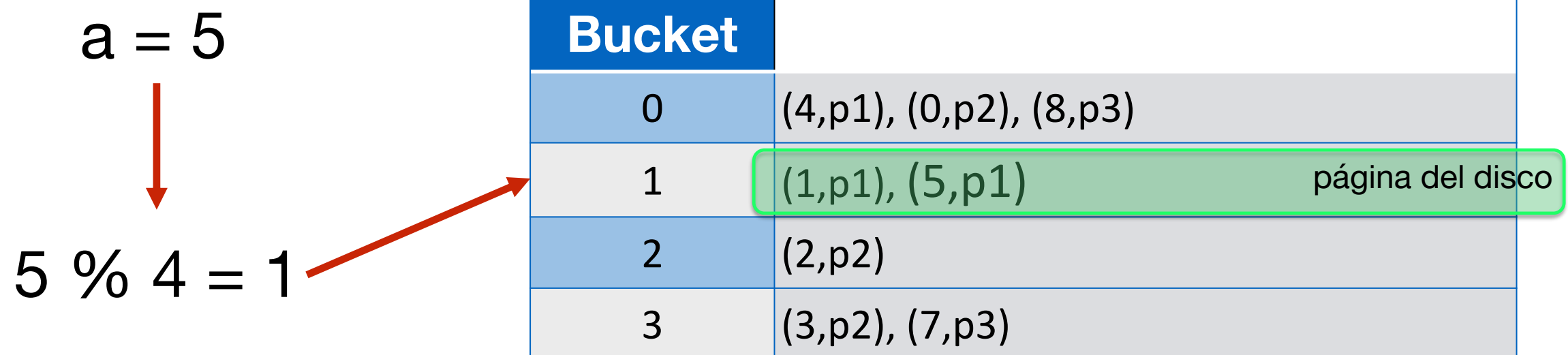
(0,2,3), (3,1,2), (2,3,3)

p3

(7,8,9), (8,1,2)

# Hash Index unclustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$



Datos en el disco:

p1

(4,1,1), (1,2,2), (5,1,1)

p2

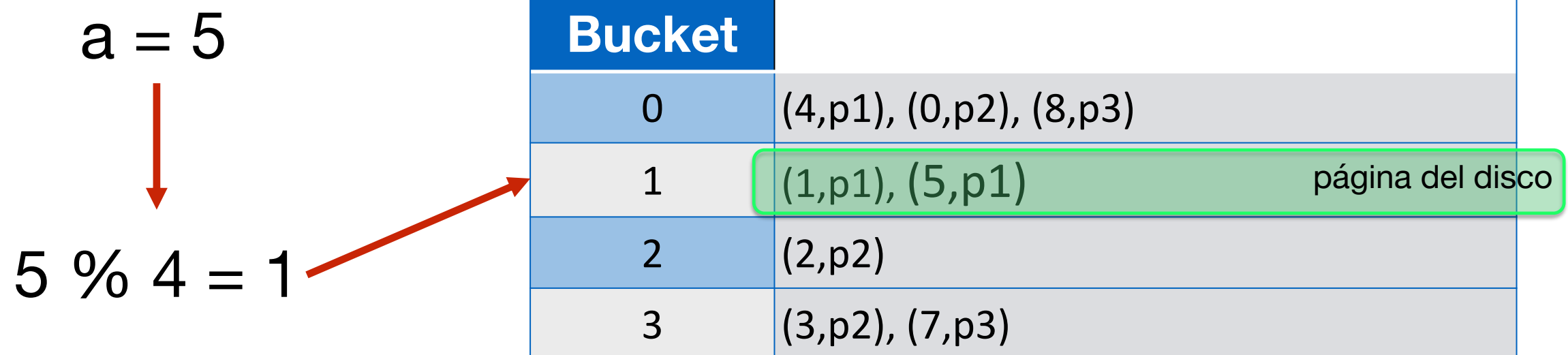
(0,2,3), (3,1,2), (2,3,3)

p3

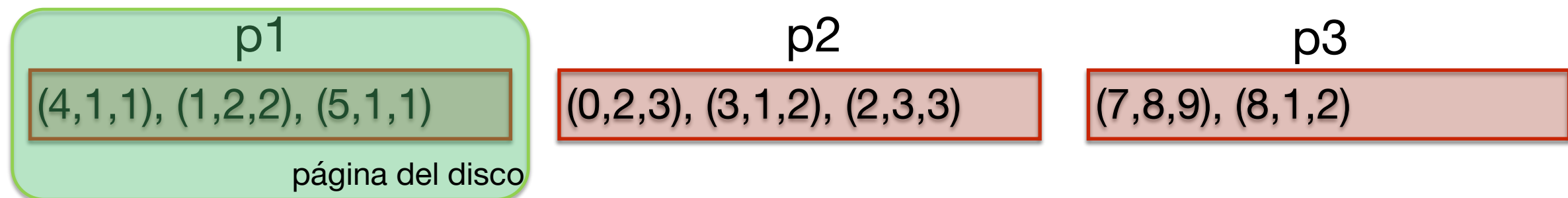
(7,8,9), (8,1,2)

# Hash Index unclustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$

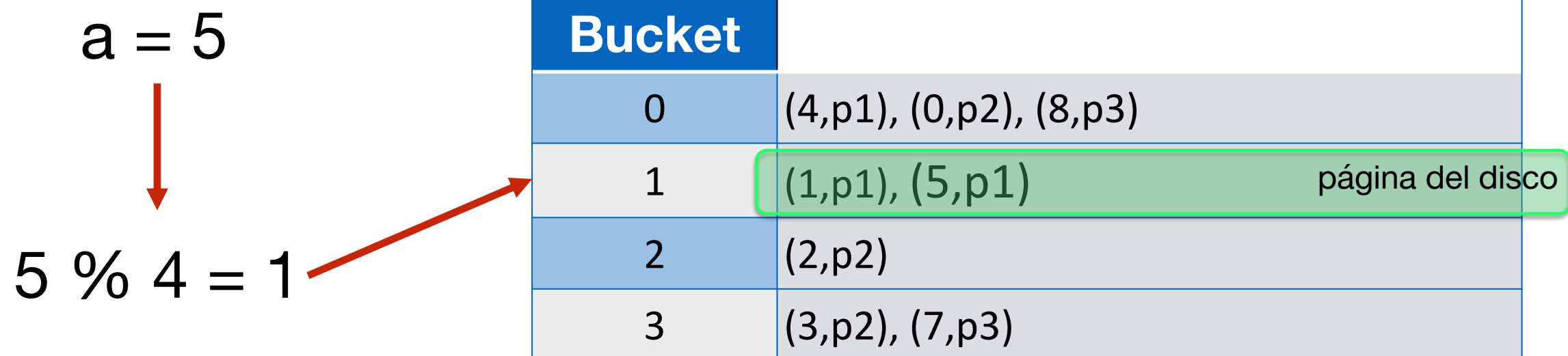


Datos en el disco:

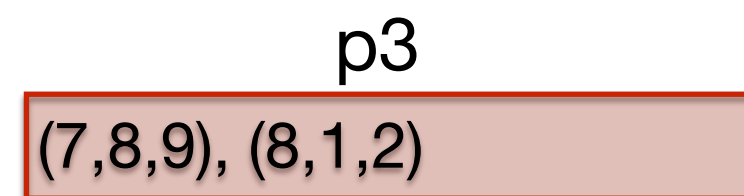
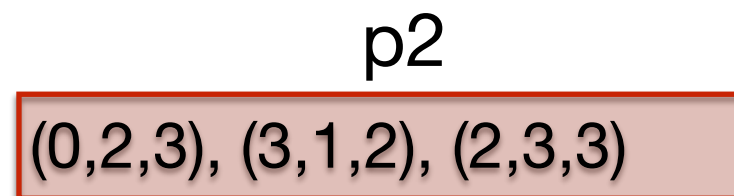
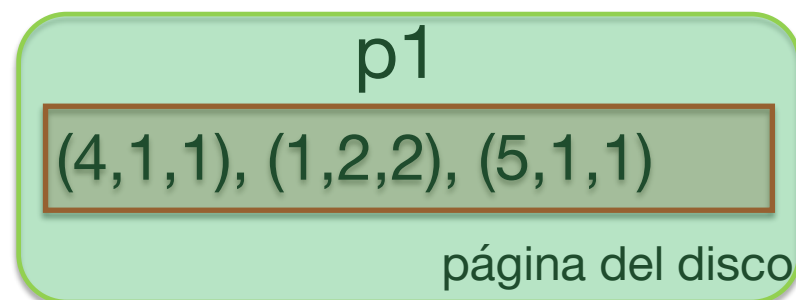


# Hash Index unclustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$



Datos en el disco:



$\#(I/O) = 2$

# Hash Index unclustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$

Considerar: página del índice vs página con tuplas

P punteros/página

(1,p1), (2,p7),...

B tuplas/página

(1,2,3), (7,3,1),...

puntero  $p1$  ... un long/8 bytes

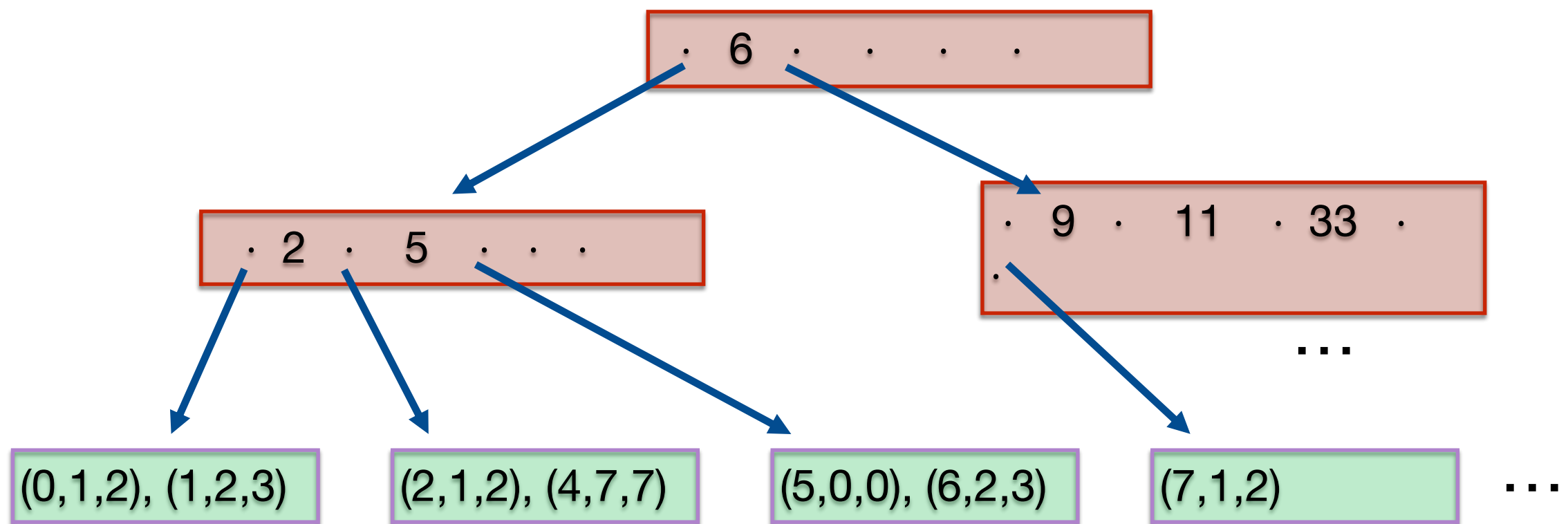
tupla ... puede ser 20 longs/160 bytes

¿Dónde cabe más? ( $P > B$ )



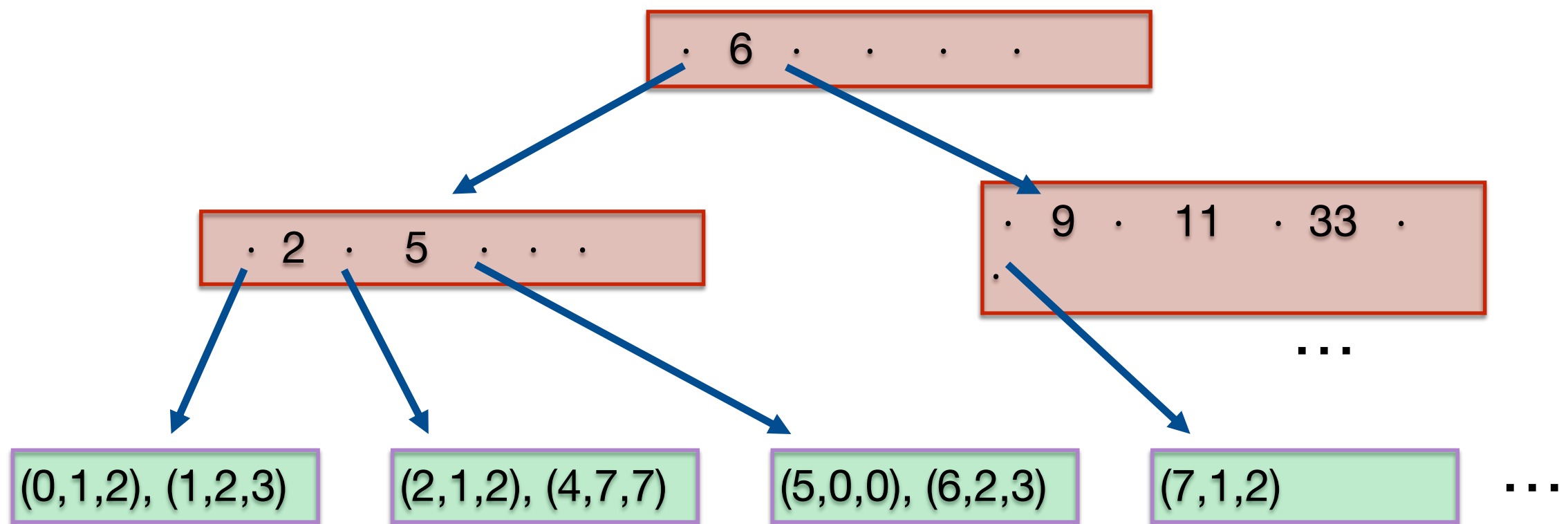
# B+ tree clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$



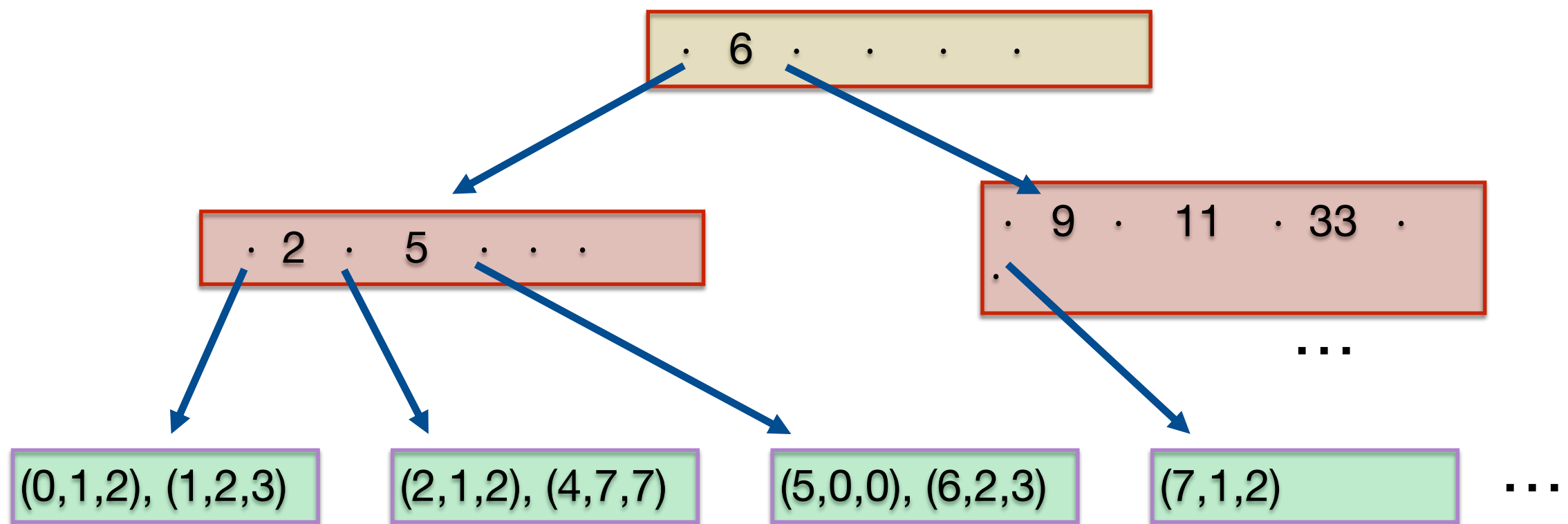
# B+ tree clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a = 5$



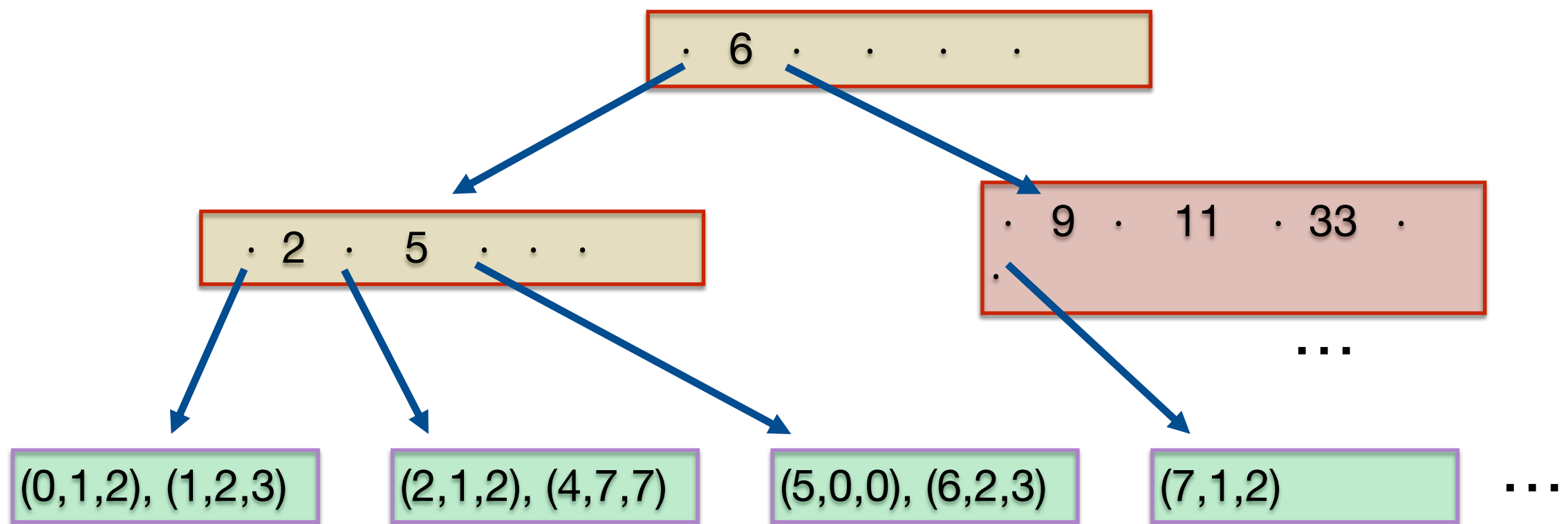
# B+ tree clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a = 5$



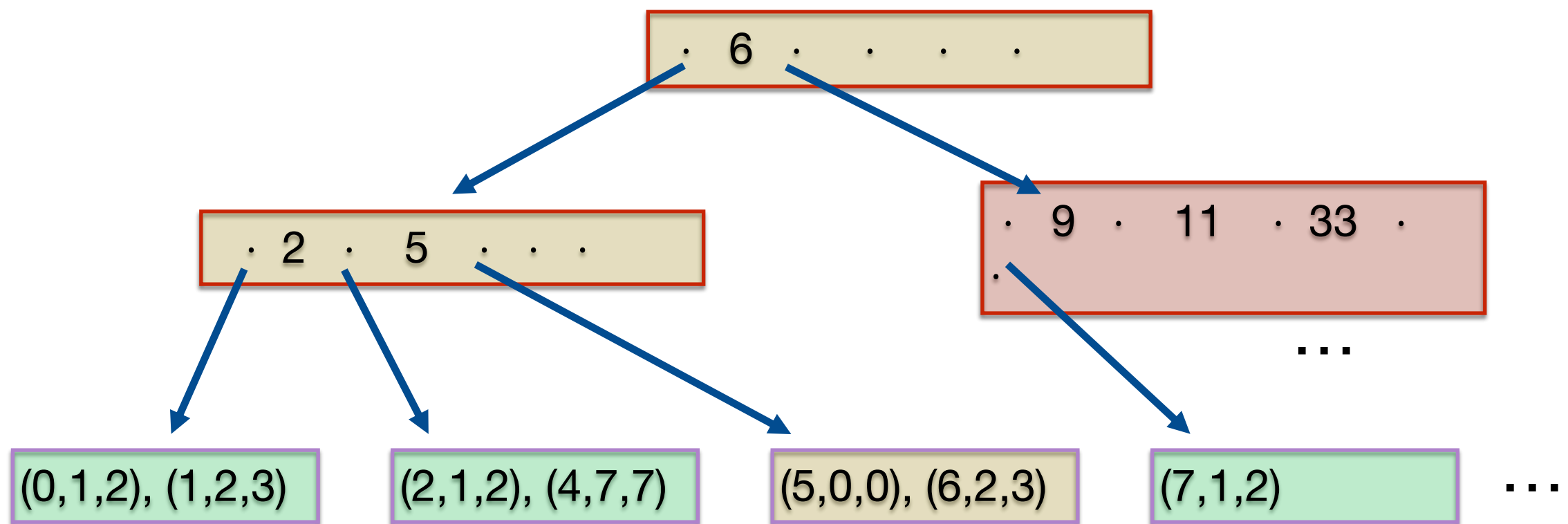
# B+ tree clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a = 5$



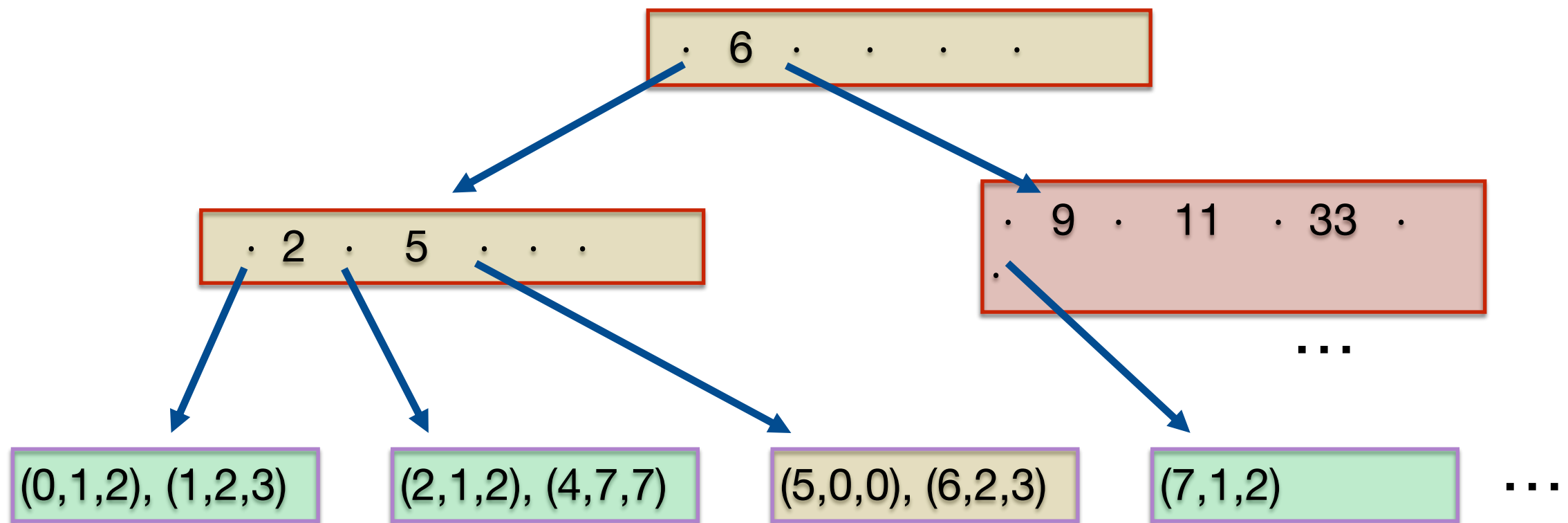
# B+ tree clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a = 5$



# B+ tree clustered

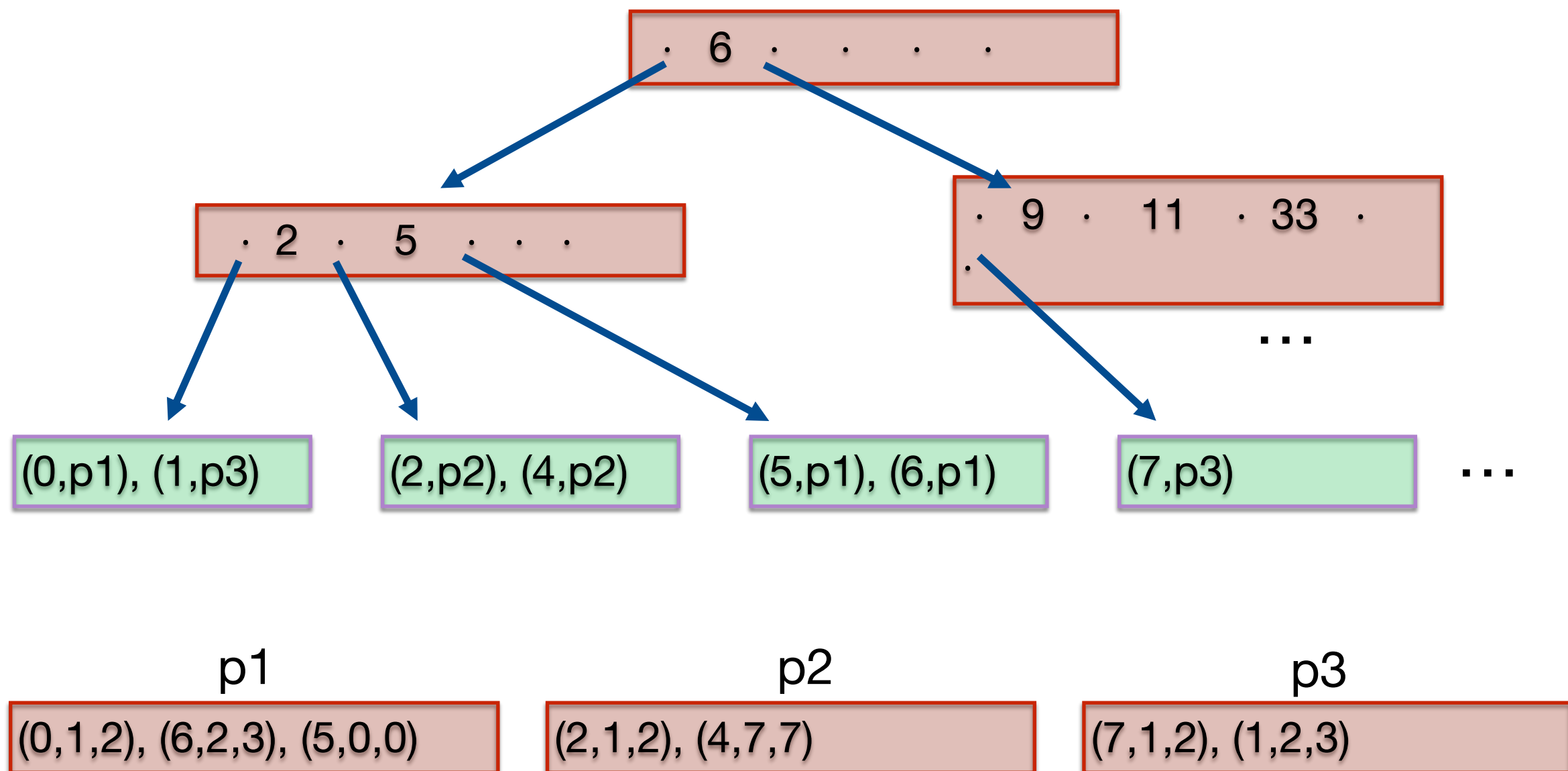
$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a = 5$



$\#(I/O) = h$  ... altura del árbol (los datos están en hojas)

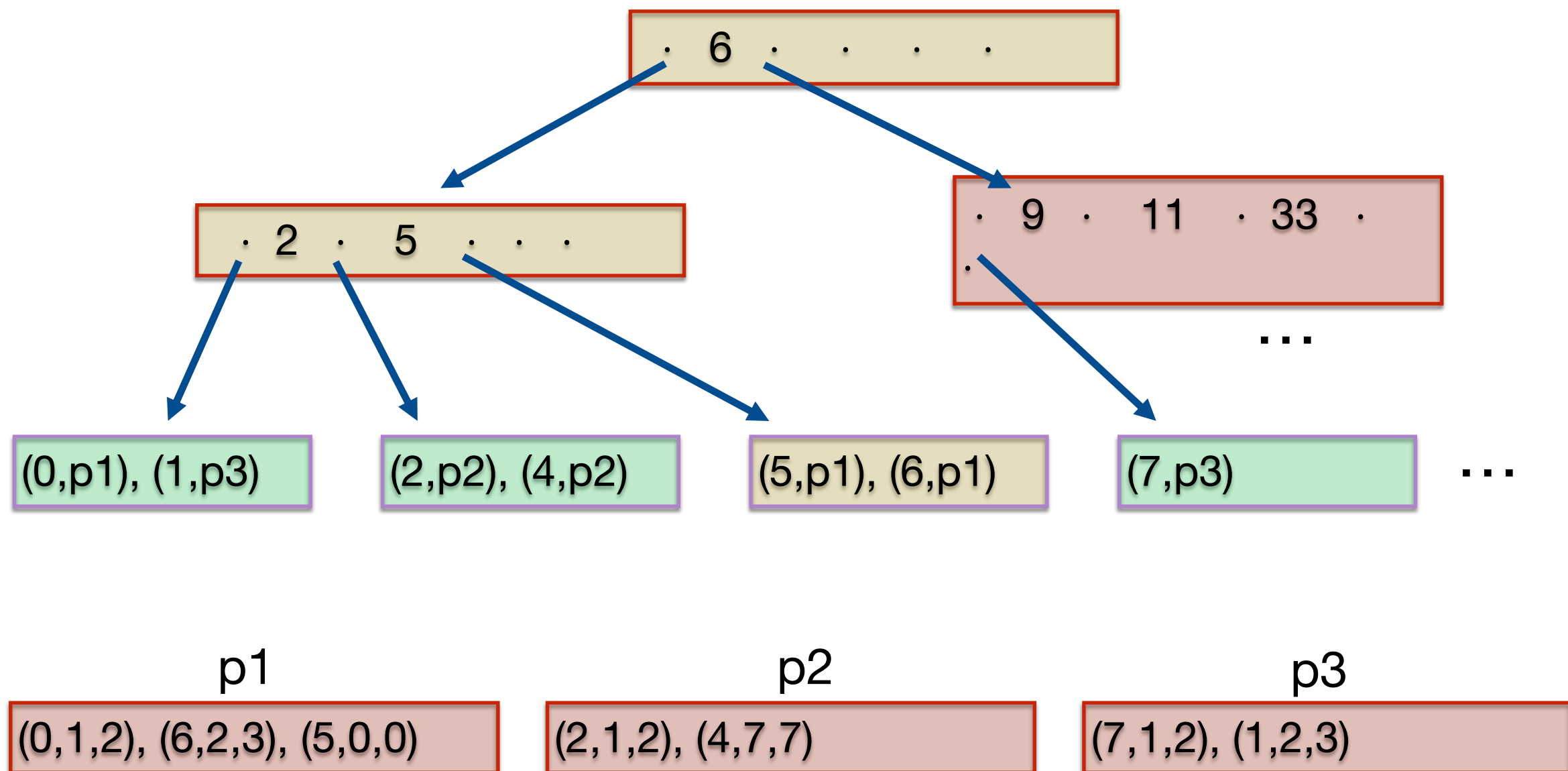
# B+ tree unclustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$



# B+ tree unclustered

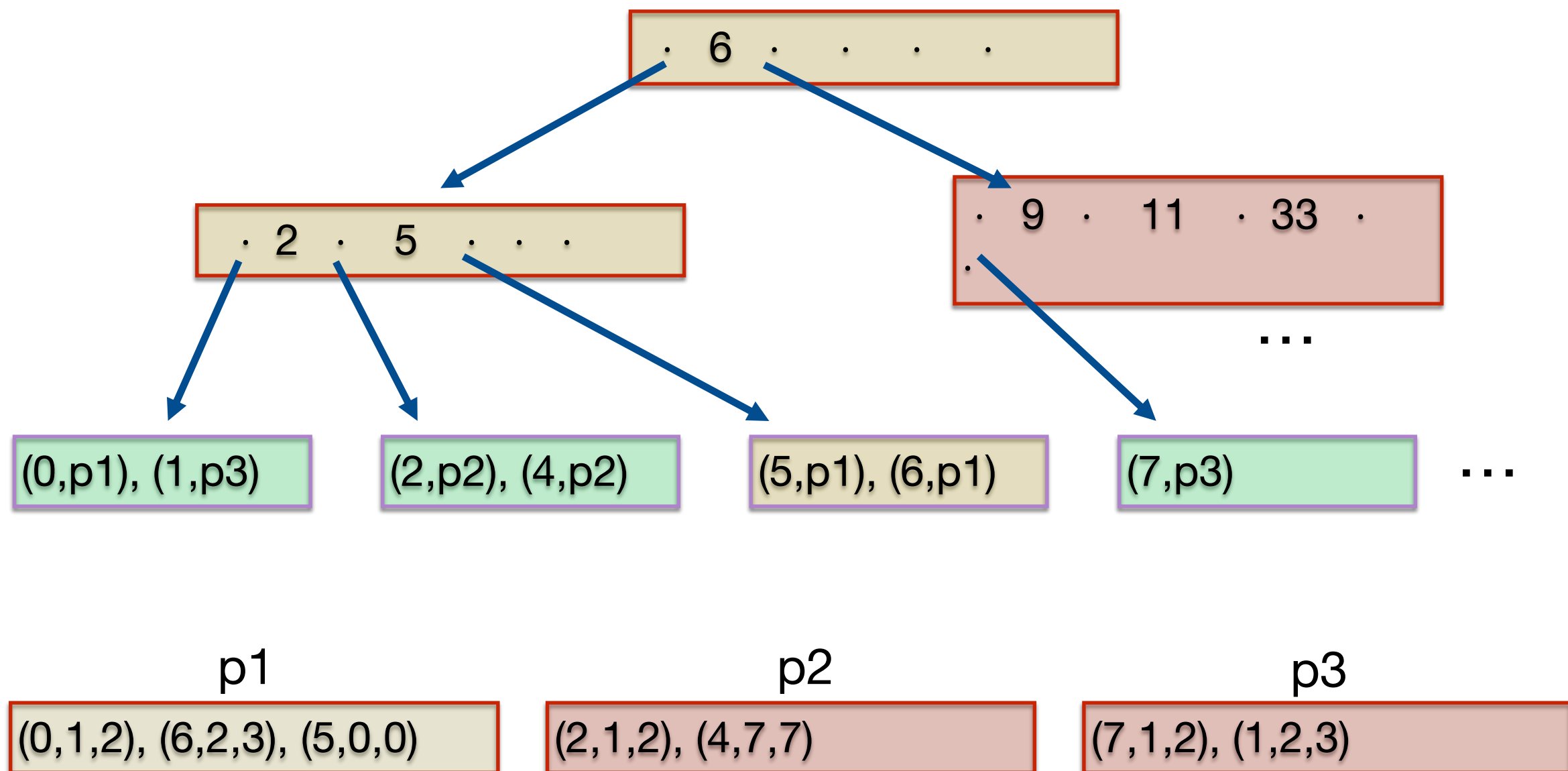
$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a = 5$





# B+ tree unclustered

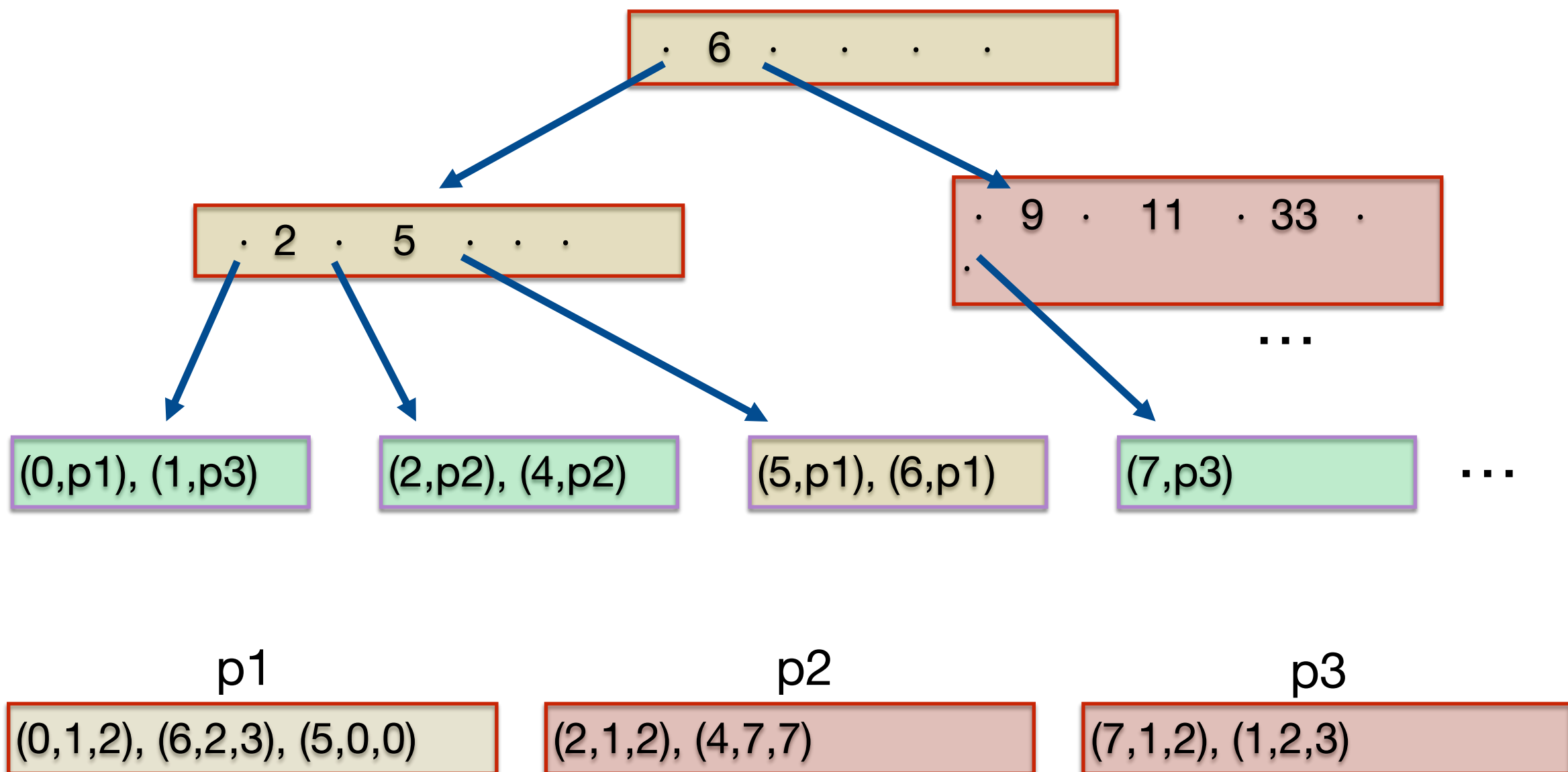
$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a = 5$



# B+ tree unclustered

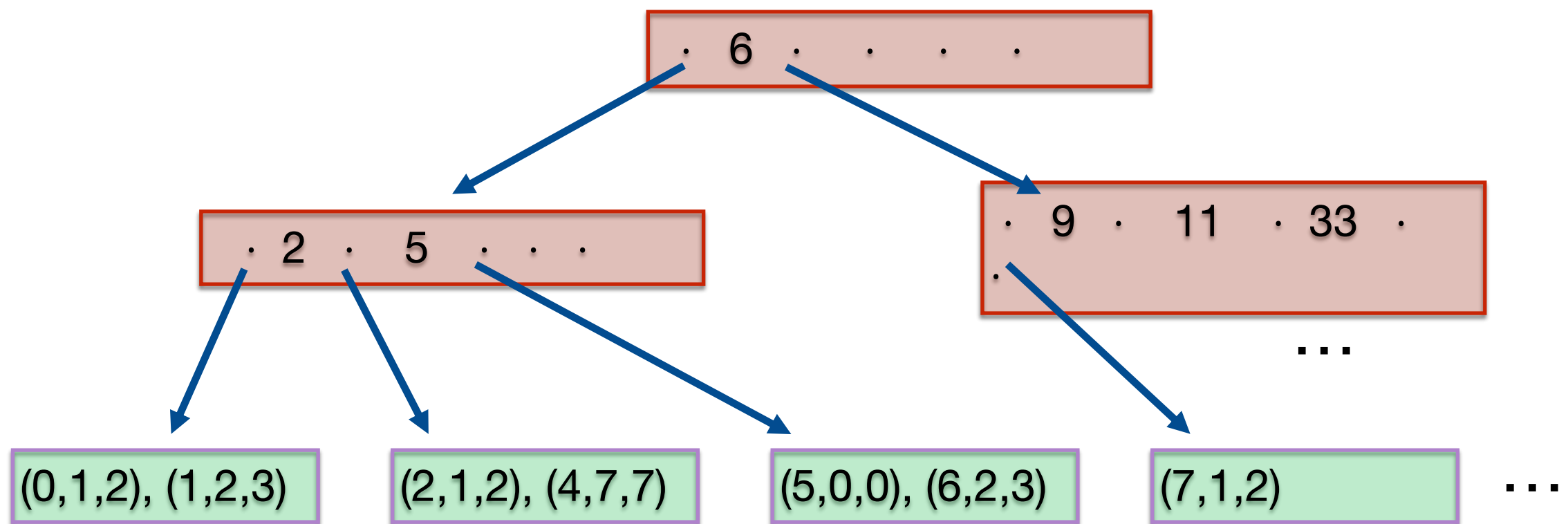
$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a = 5$

Costo  $h+1$



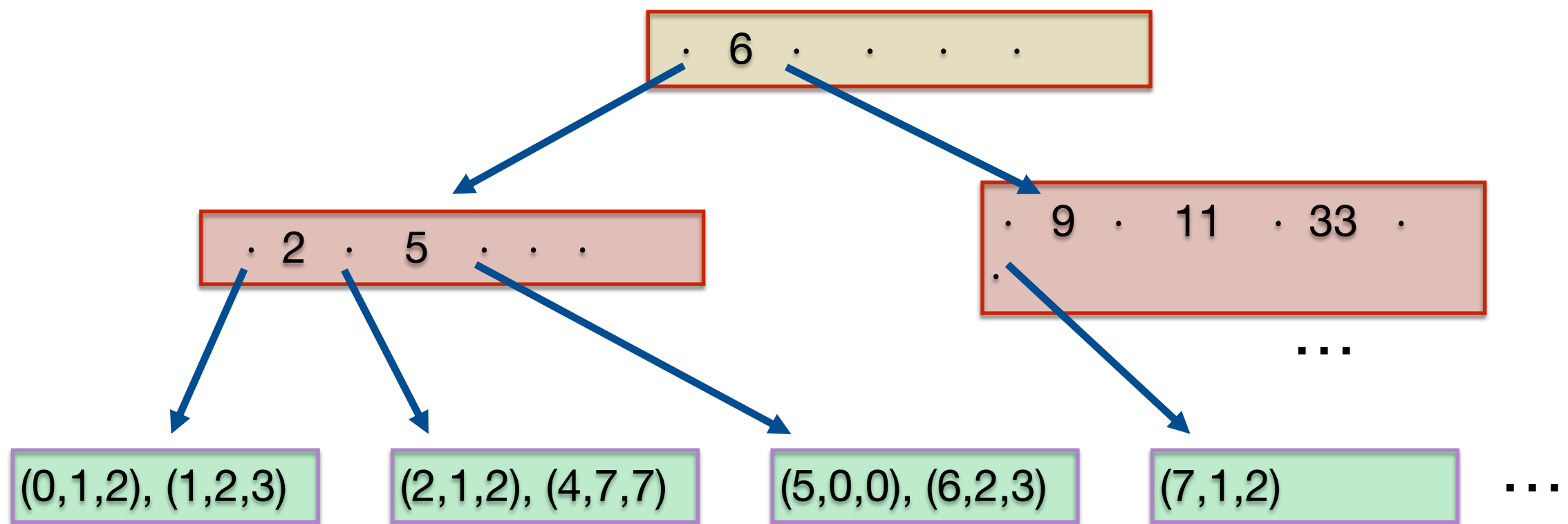
# B+ tree clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a > 5$



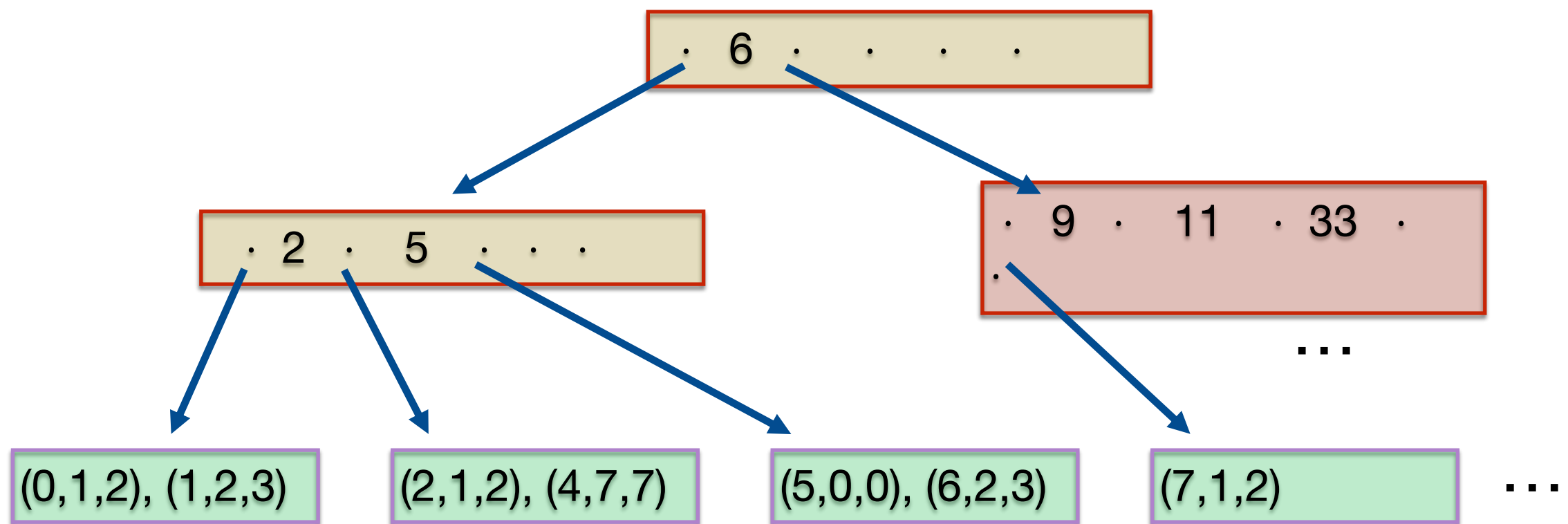
# B+ tree clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a > 5$



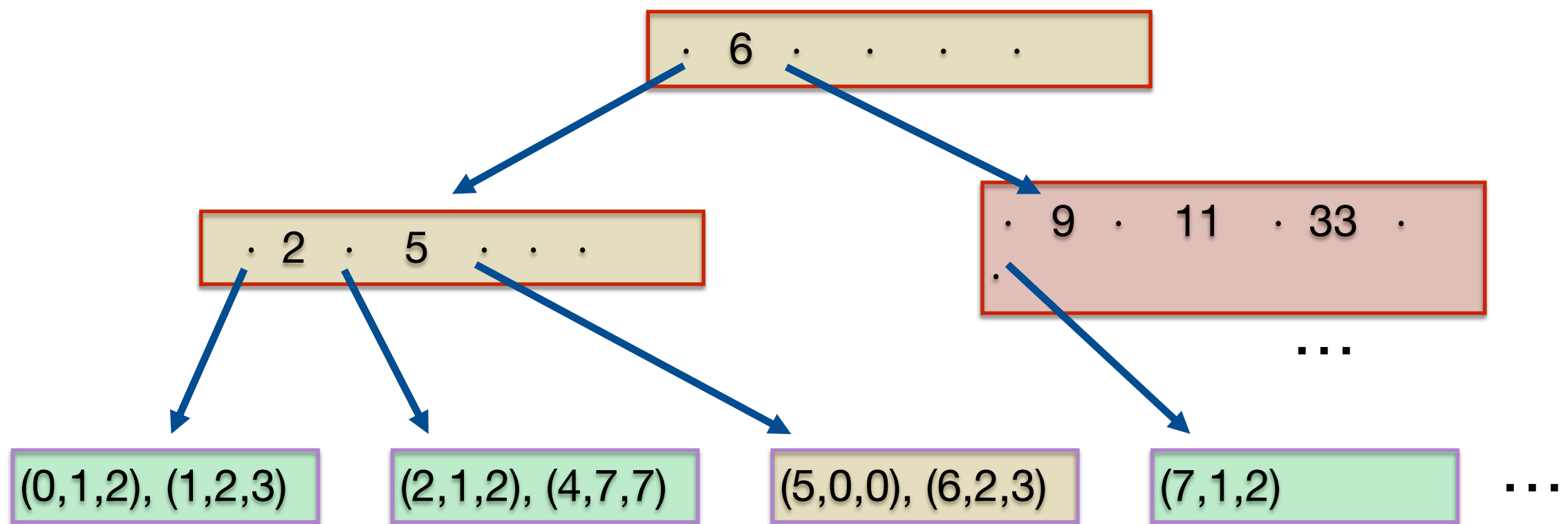
# B+ tree clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a > 5$



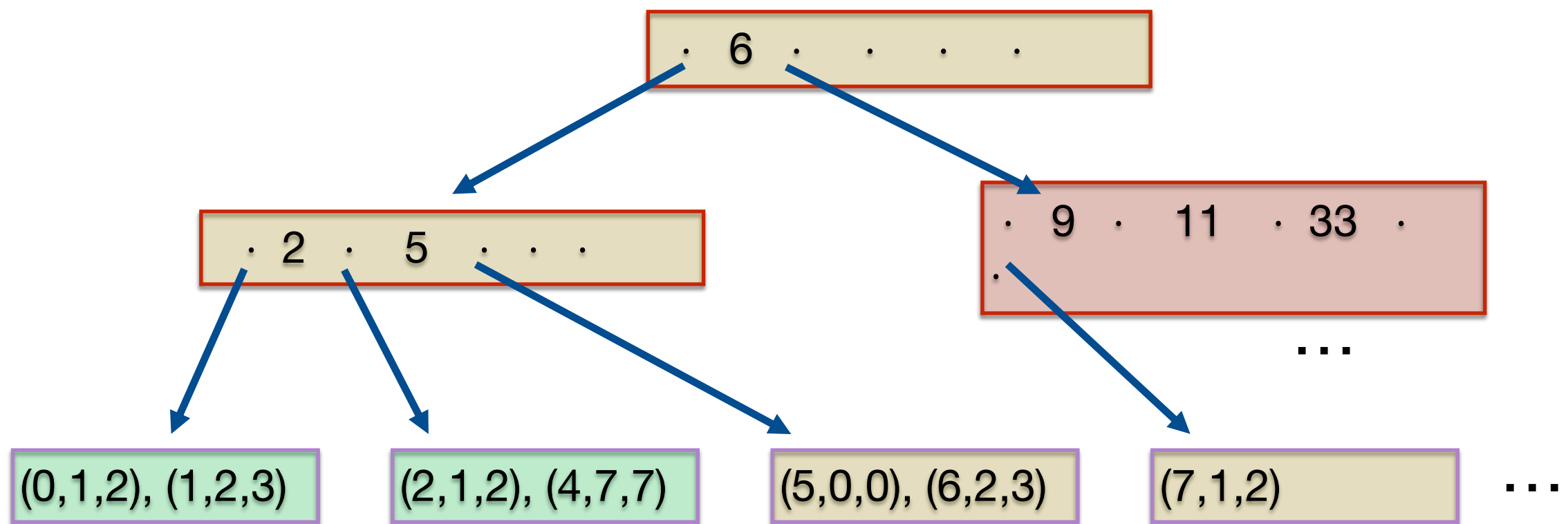
# B+ tree clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a > 5$



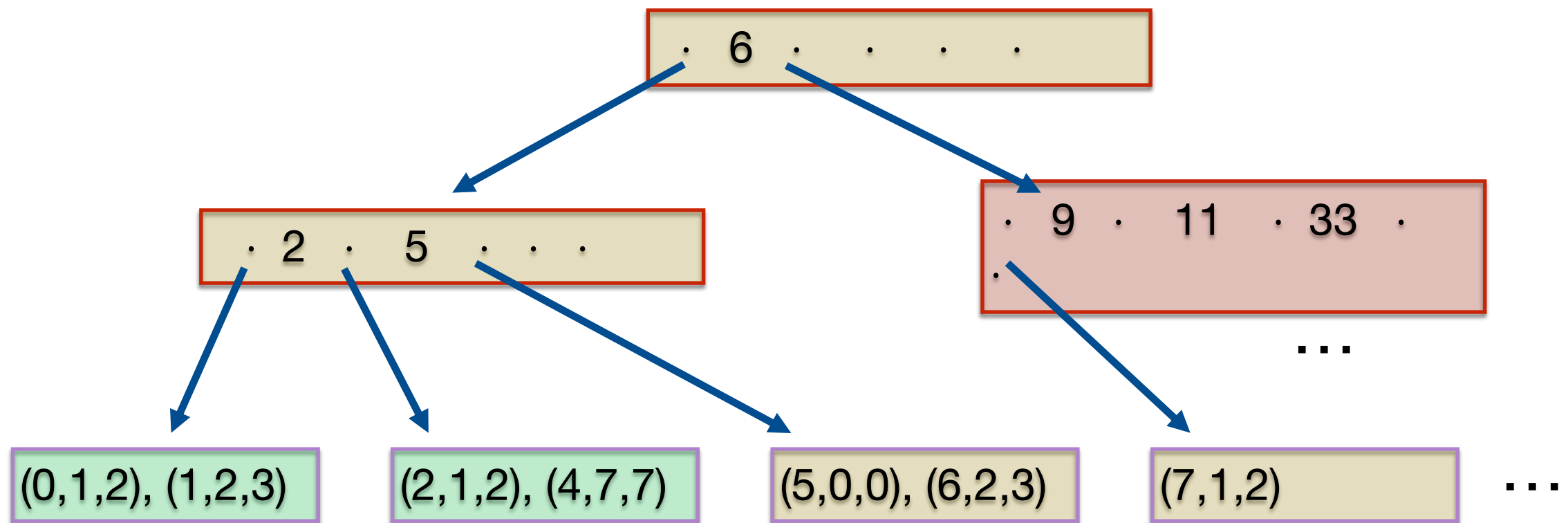
# B+ tree clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a > 5$



# B+ tree clustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a > 5$



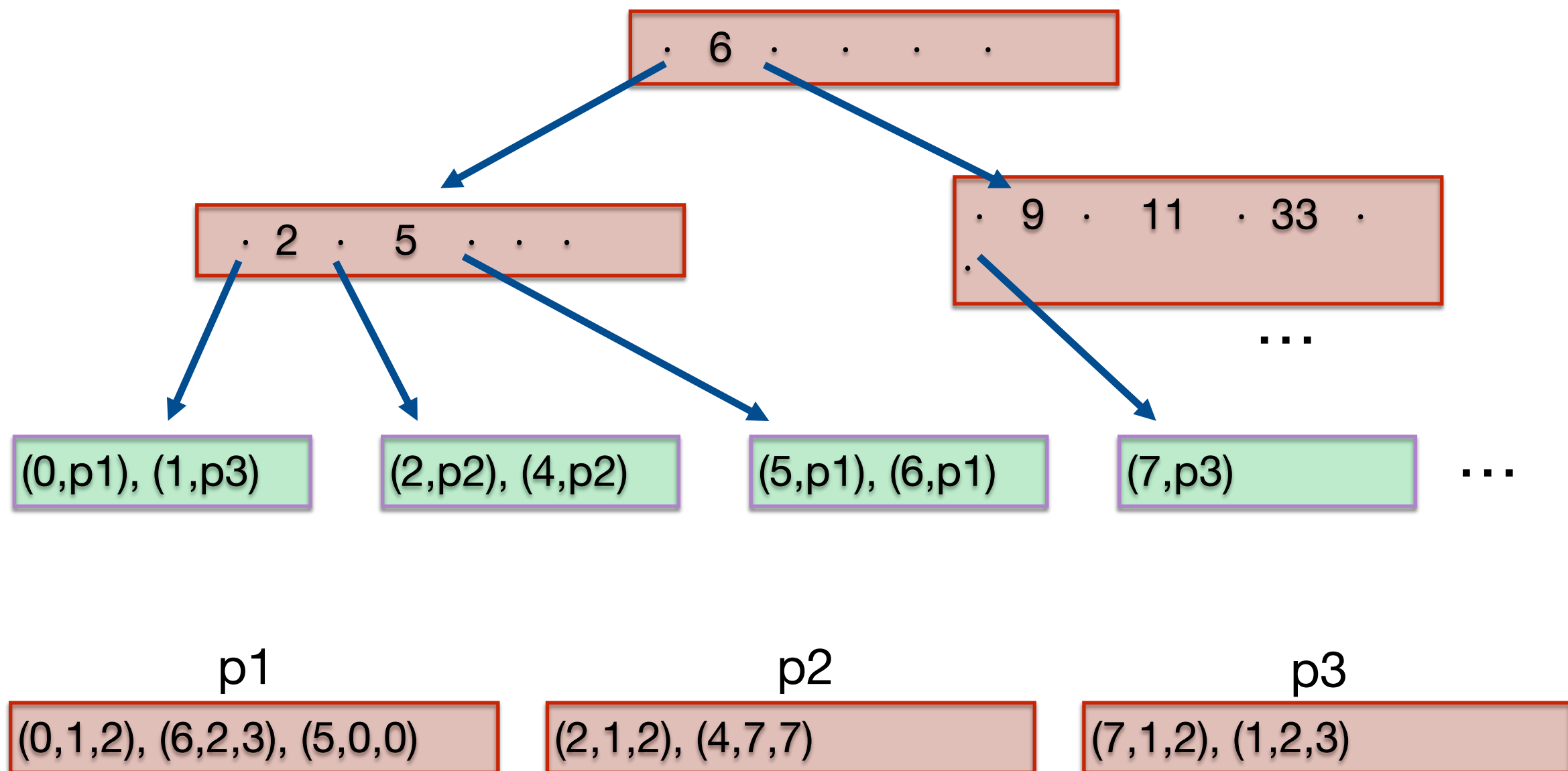
$$\#(I/O) = h + B_m$$

$B_m$  : nr de páginas con tuplas que calzan búsqueda



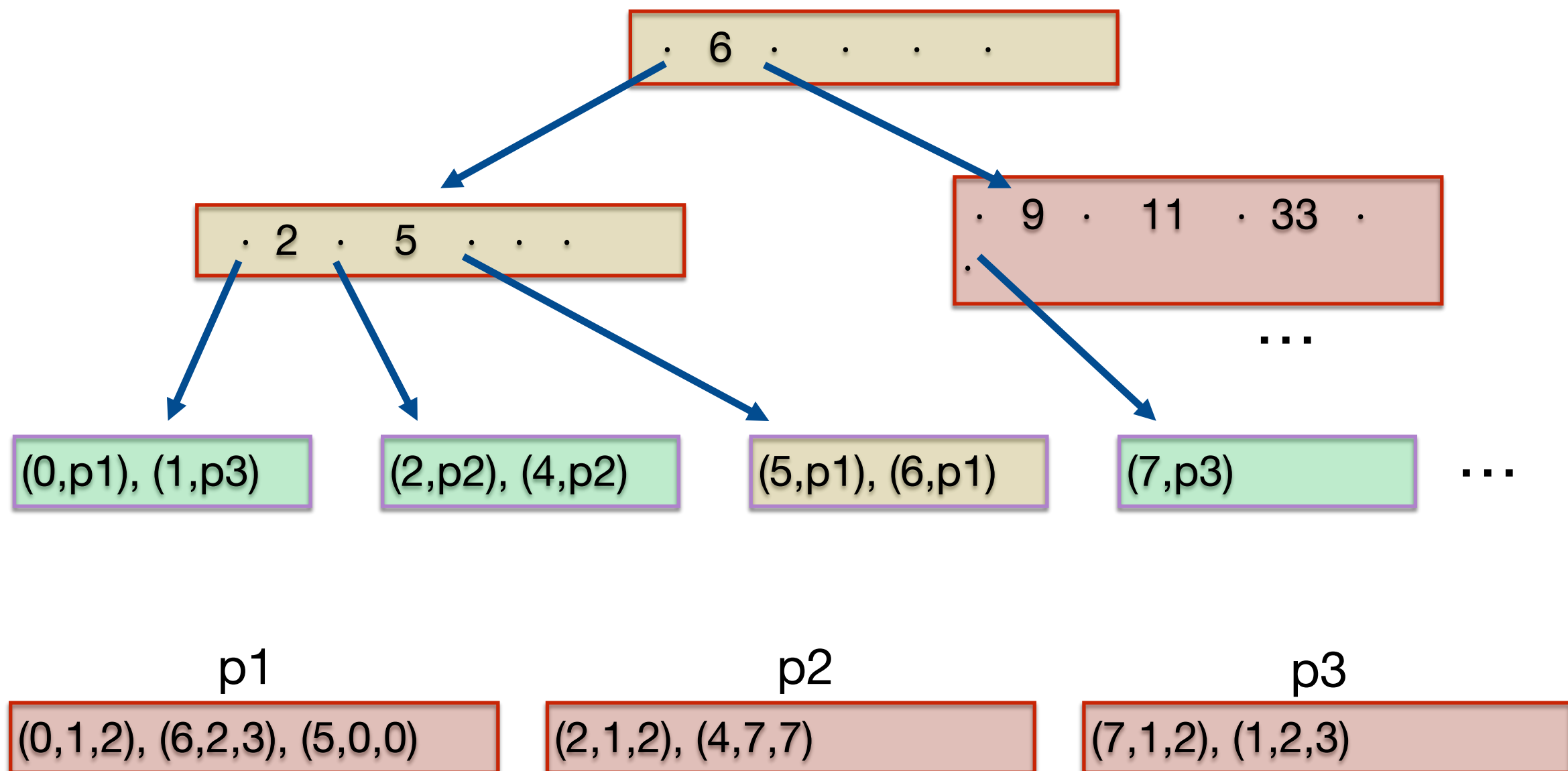
# B+ tree unclustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a > 5$



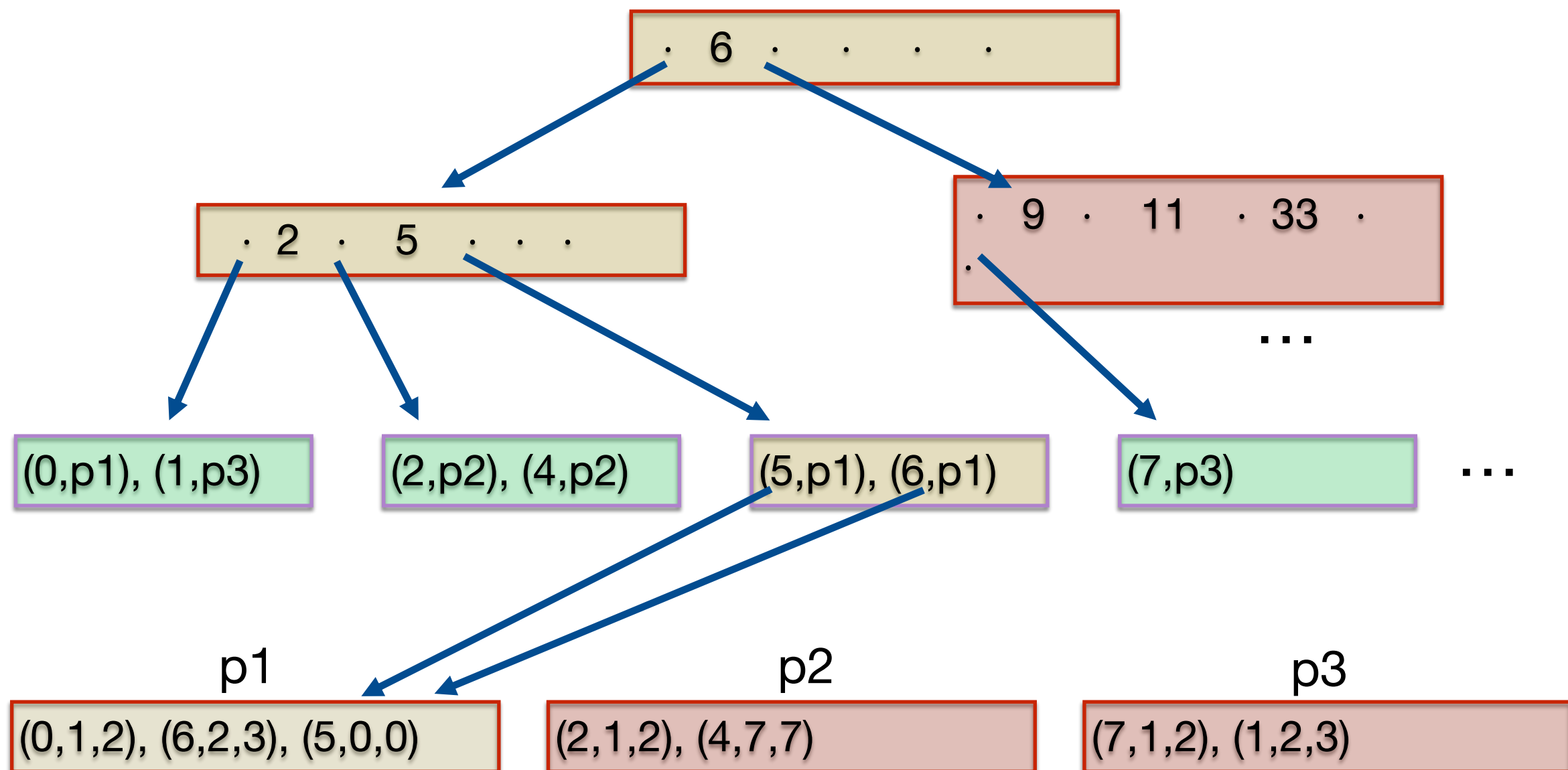
# B+ tree unclustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a > 5$



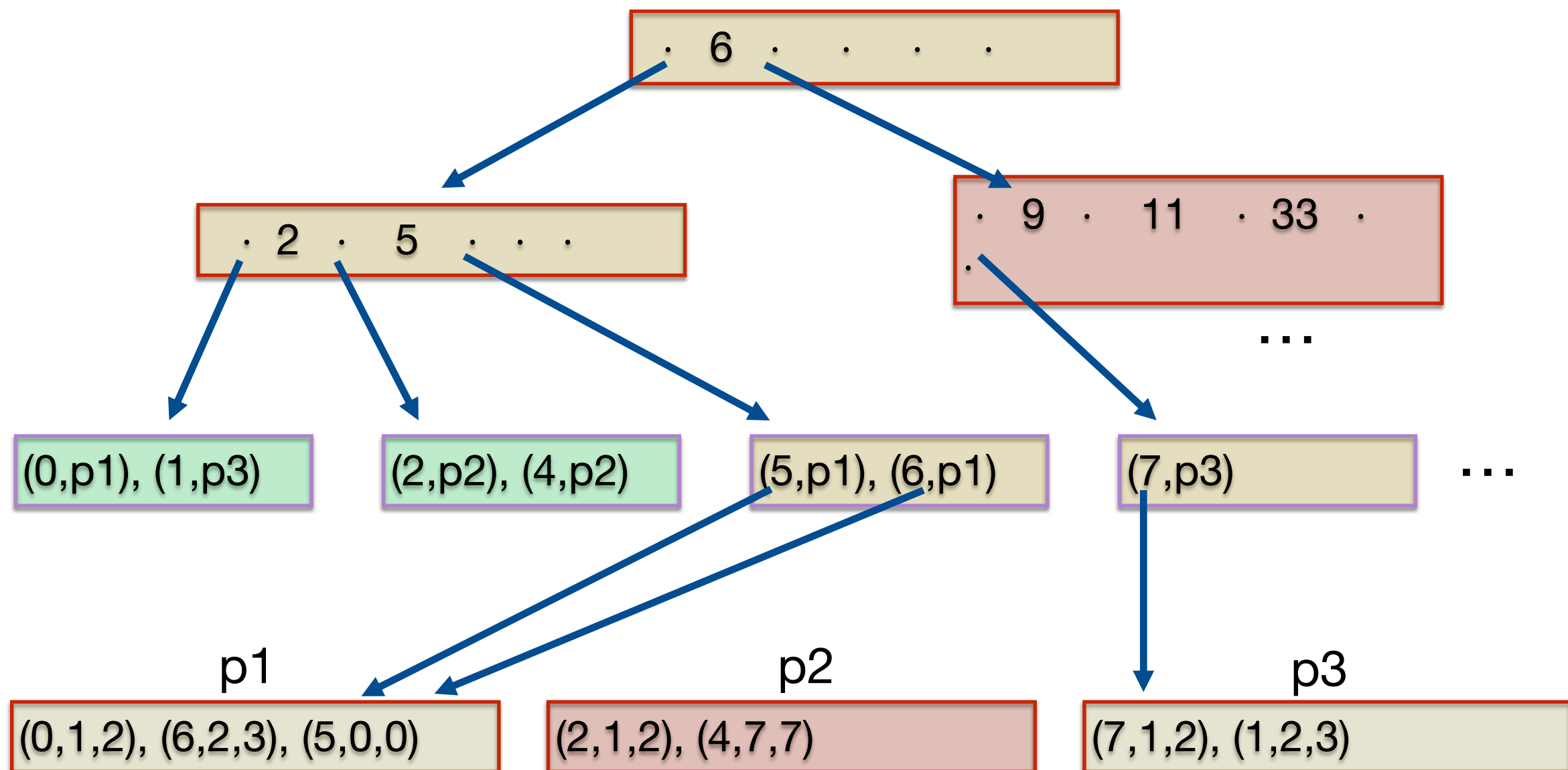
# B+ tree unclustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a > 5$



# B+ tree unclustered

$R(\underline{a}, b, c)$  ... índice sobre  $a$  ...  $a > 5$

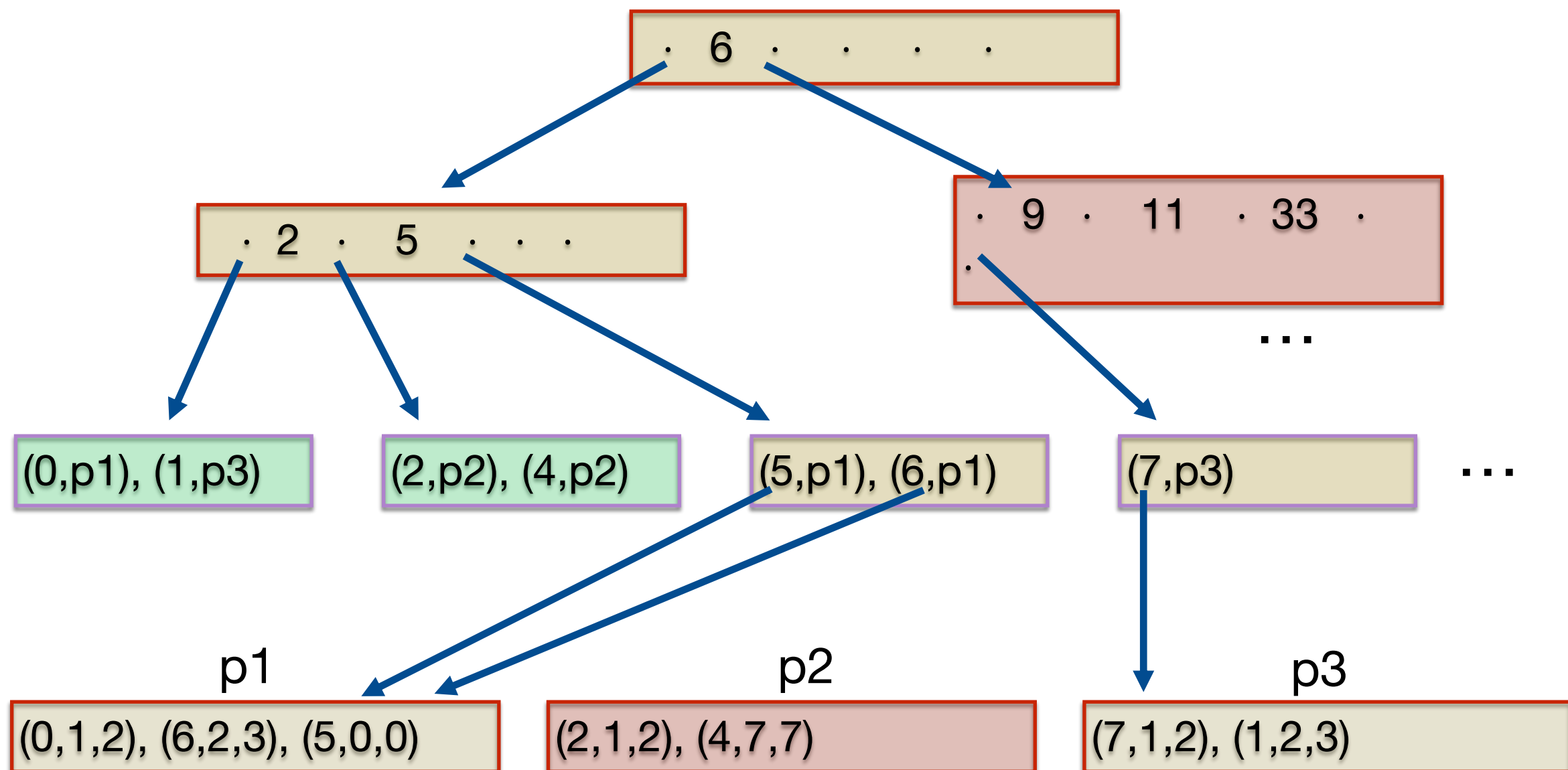


# B+ tree unclustered

$$\#(I/O) = h + \left\lceil \frac{n}{P} \right\rceil + n - 1$$

$n$  : número de tuplas que calzan con búsquedas  
 $P$  : Cantidad de punteros por página

$R(\underline{a}, b, c) \dots$  índice sobre  $a \dots a > 5$



# B+ tree unclustered

$R(\underline{a}, b, c)$ :

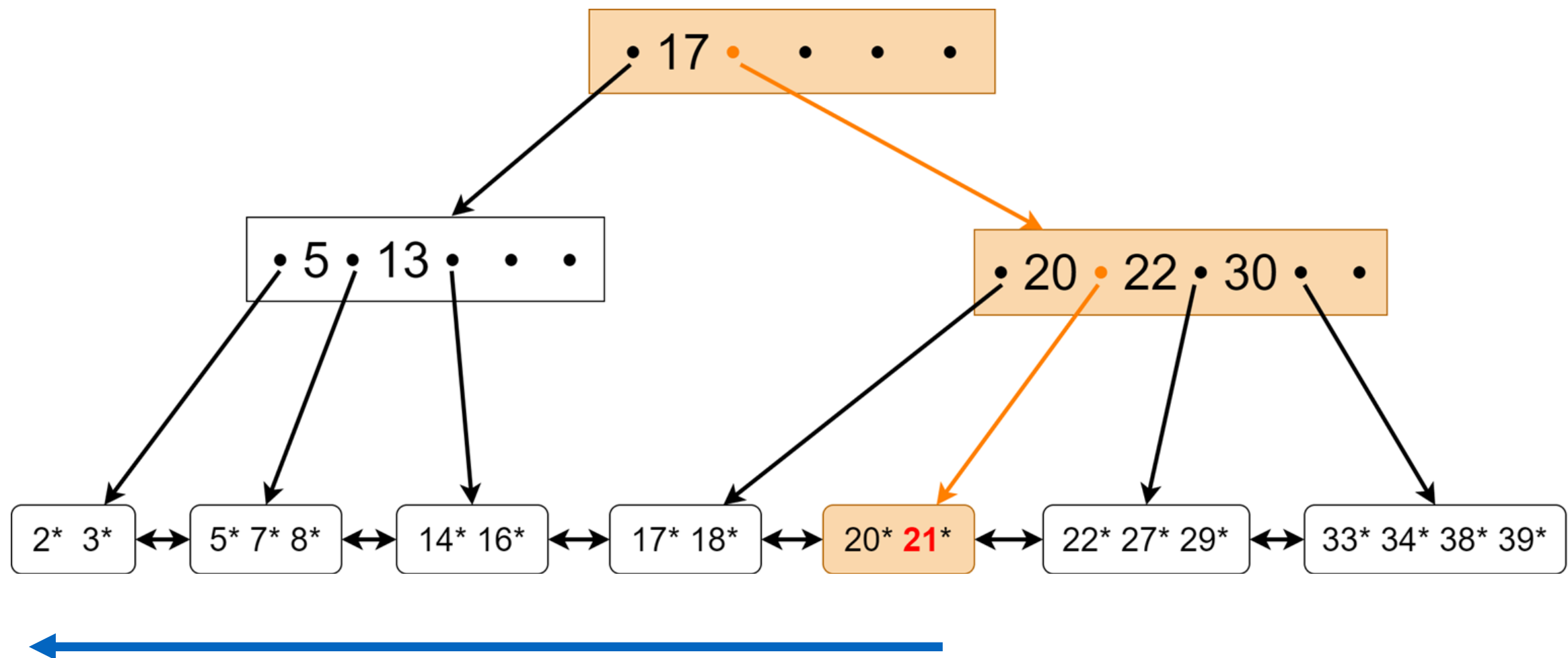
- 1M tuplas
- valor de  $\underline{a}$  distribuido uniformemente en  $[0, 1.000.000]$
- B tuplas por página
- $P > B$  punteros en una página (hojas a 100% ocupación)
- h altura del B+ tree

SELECT \* FROM R WHERE  $a < 21$

Costo:

- Buscar  $a = 21 \dots h$
- Scan hacia la izquierda

# B+ tree unclustered



# B+ tree unclustered

$R(\underline{a}, b, c)$ :

- 1M tuplas
- valor de  $\underline{a}$  distribuido uniformemente en  $[0, 1.000.000]$
- B tuplas por página
- $P > B$  punteros en una página (hojas a 100% ocupación)
- h altura del B+ tree

Costo:

- Buscar  $a = 21 \rightarrow h$
- Por uniformidad + a es llave para  $a=21 \rightarrow 21$  tuplas
- Cuántos punteros hay en una página?  $\rightarrow 21/P$
- Más buscar la página en el disco (unclustered)  $\rightarrow 21$

$$h + (21/P - 1) + 21$$



# B+ tree unclustered

$R(\underline{a}, b, c)$ :

- 1M tuplas
- valor de  $\underline{a}$  distribuido uniformemente en  $[0, 1.000.000]$
- B tuplas por página
- $P > B$  punteros en una página (hojas a 100% ocupación)
- h altura del B+ tree

Costo:

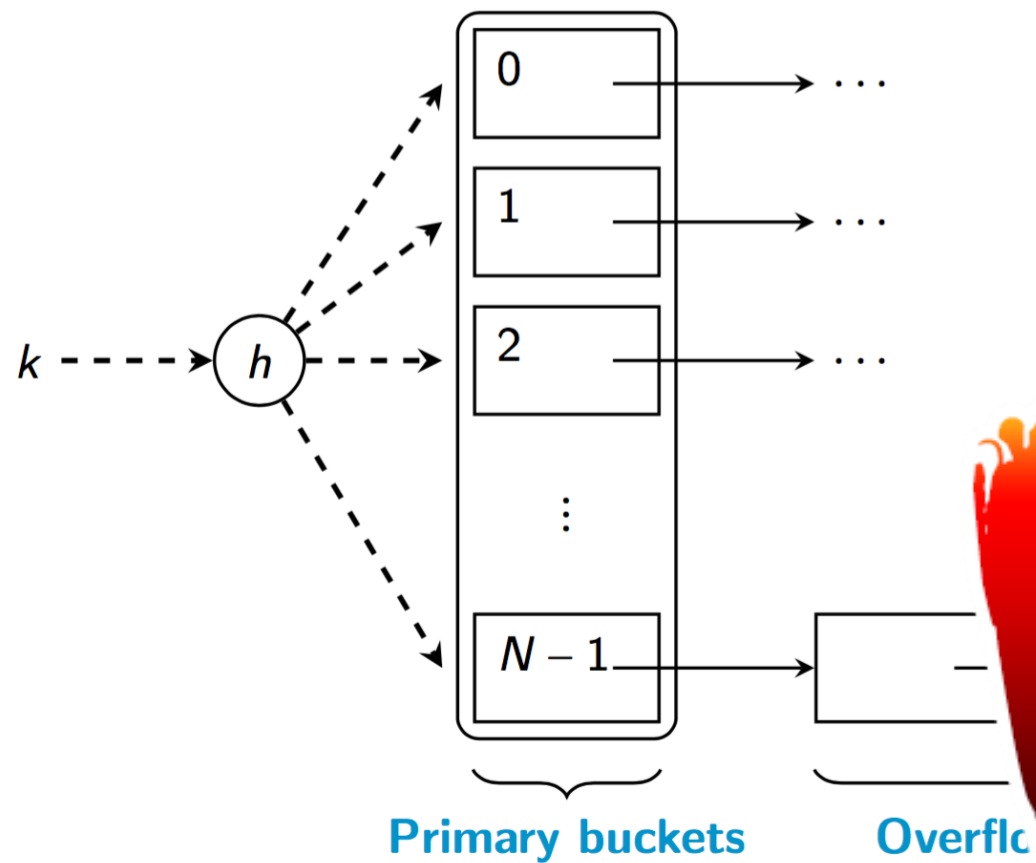
$$h + (21/P - 1) + 21$$

Bajar hacia  
 $a = 21$

Scan a izquierda  
hay P punteros/página

Para cada a  
recibo puntero, no la tupla

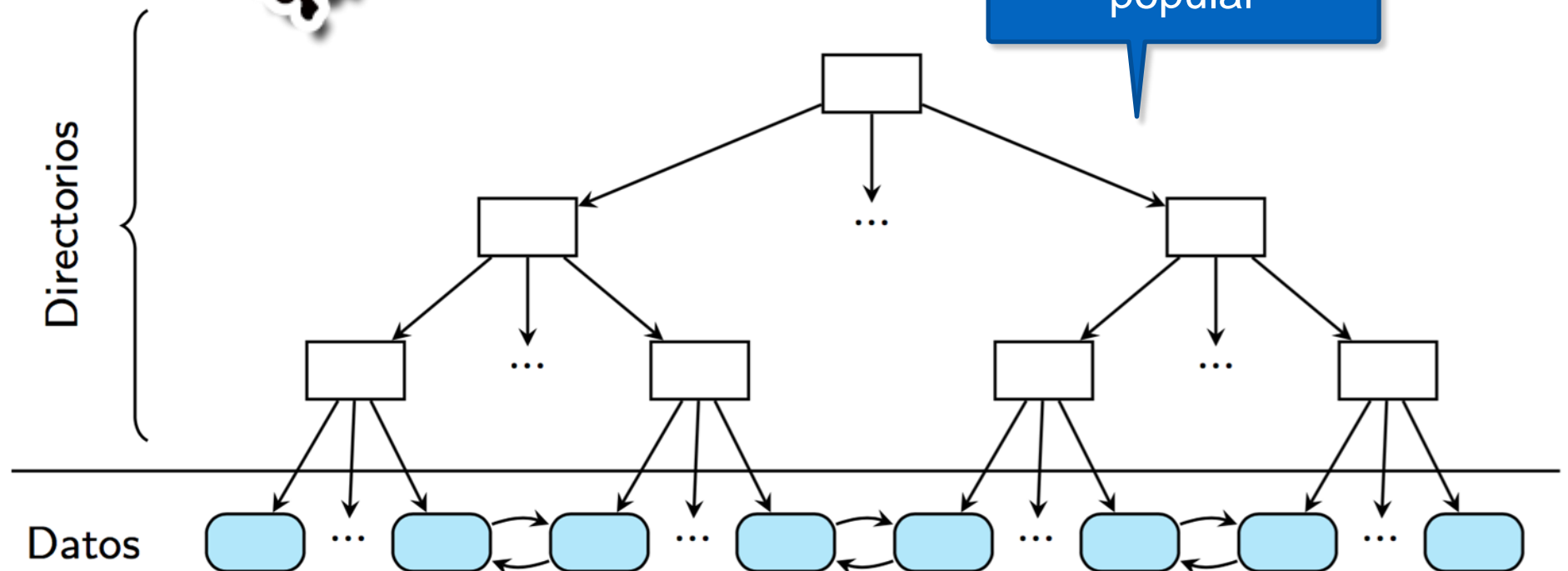
# Índices: Hash vs. Árbol B+



Levemente más eficiente para búsquedas exactas asumiendo una función de hash ideal



Mucho más eficiente para búsquedas por rango



Para más detalles:  
IIC3413 - Implementación de  
sistemas de Bases de Datos