Bases de Datos

Clase 7: Almacenamiento y Índices

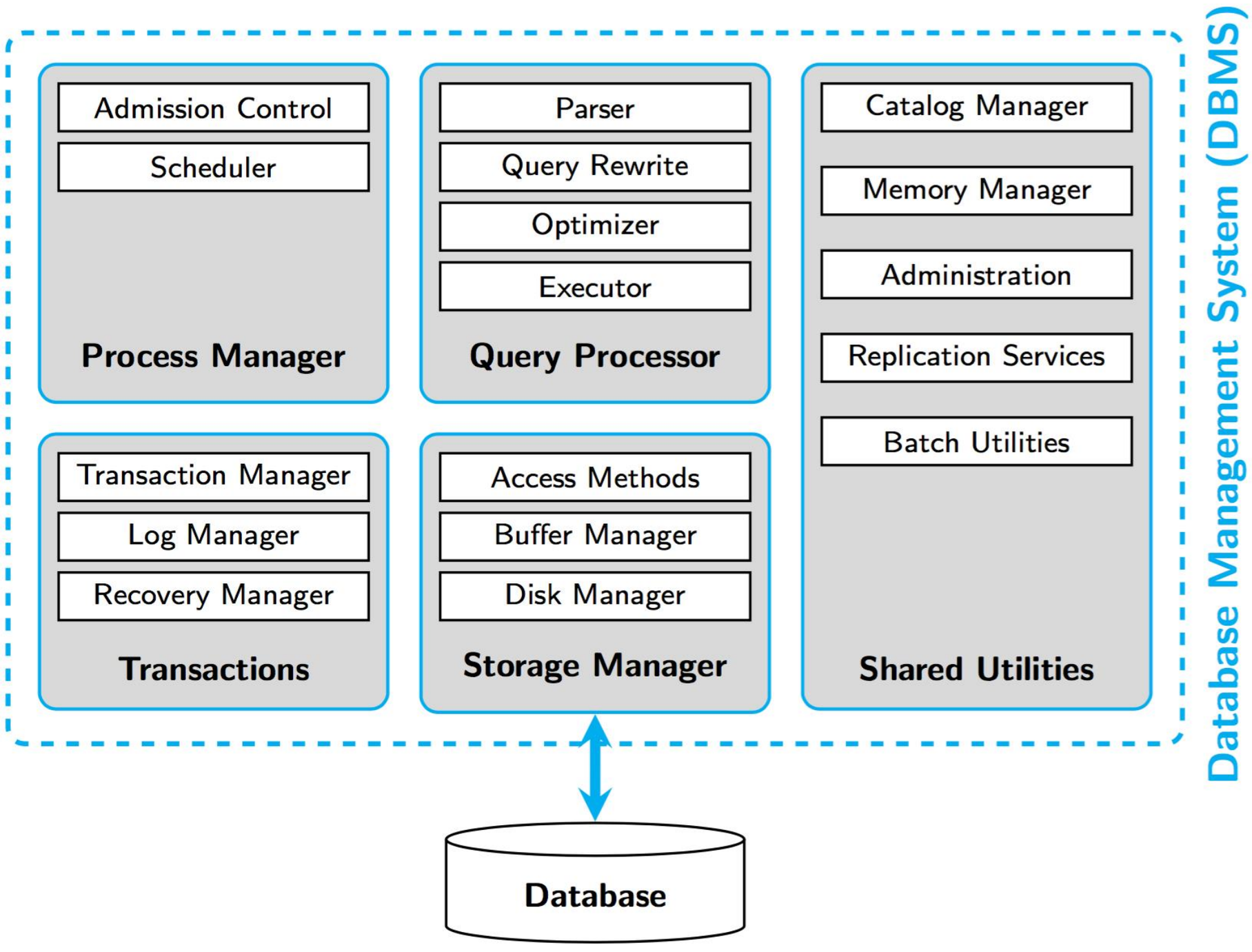
Hasta ahora

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

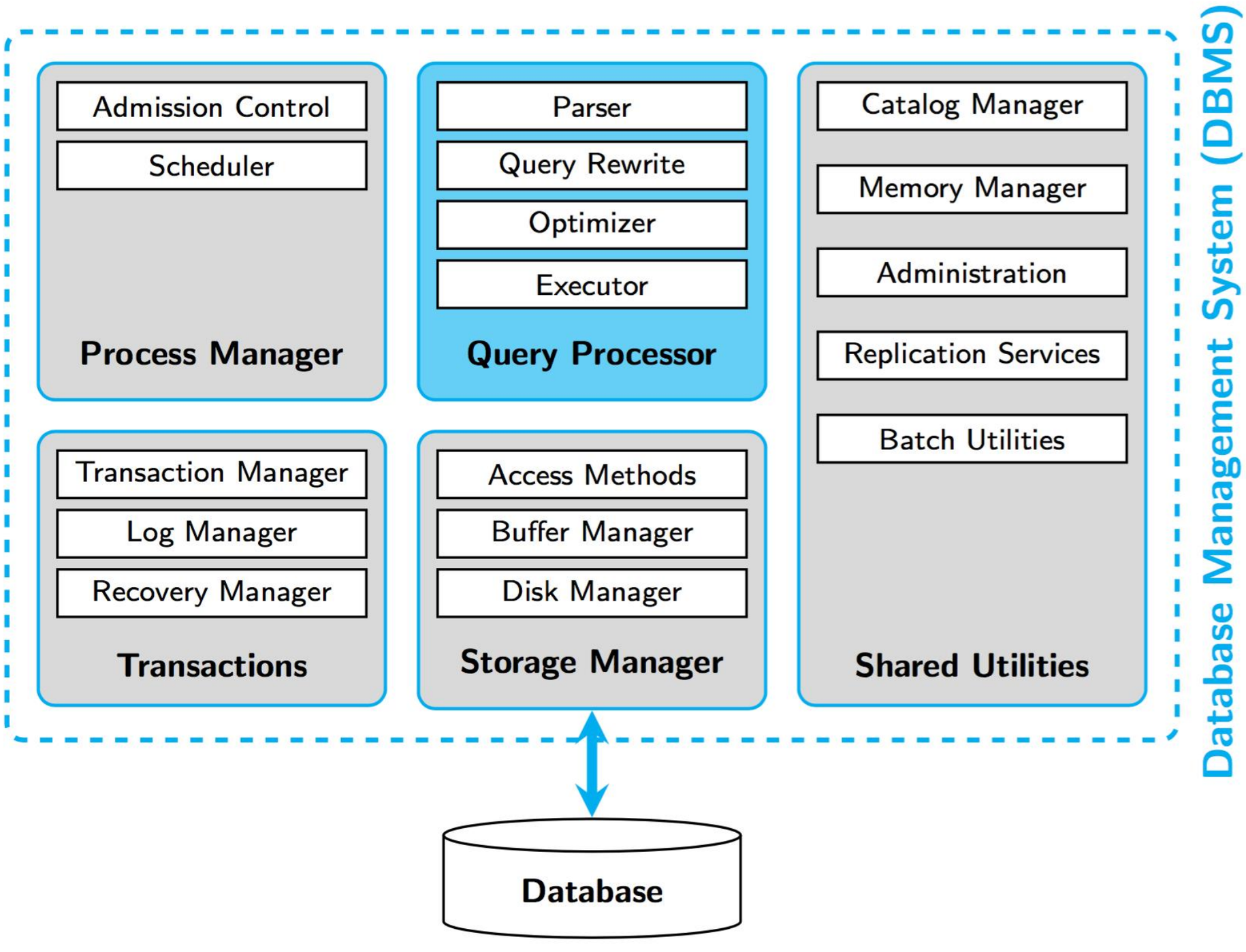
Supongamos la consulta:

SELECT pName, mStadium, goals

FROM Players P, Matches M, Players_Matches as PM

WHERE P.id = PM.id AND PM.mid = M.mid AND P.pYear >= 1985

Query Processor



Paso 1

Parser

Valida la consulta en base a la sintaxis

Convierte la consulta a álgebra relacional

Optimiza la consulta de forma básica (restricciones numéricas triviales)

Paso 2

Reescritura

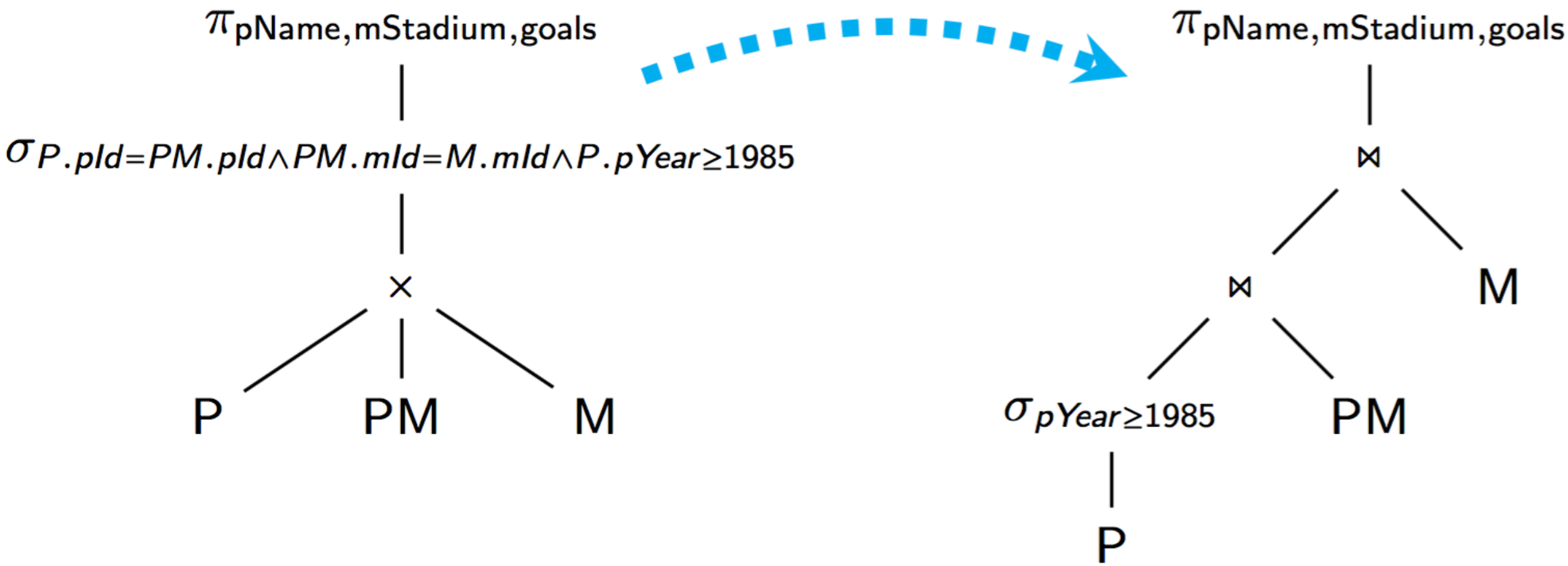
"Desanidación" de la consulta

Reescritura de la consulta aplicando reglas del álgebra relacional

Retorna el plan lógico de la consulta

Paso 2

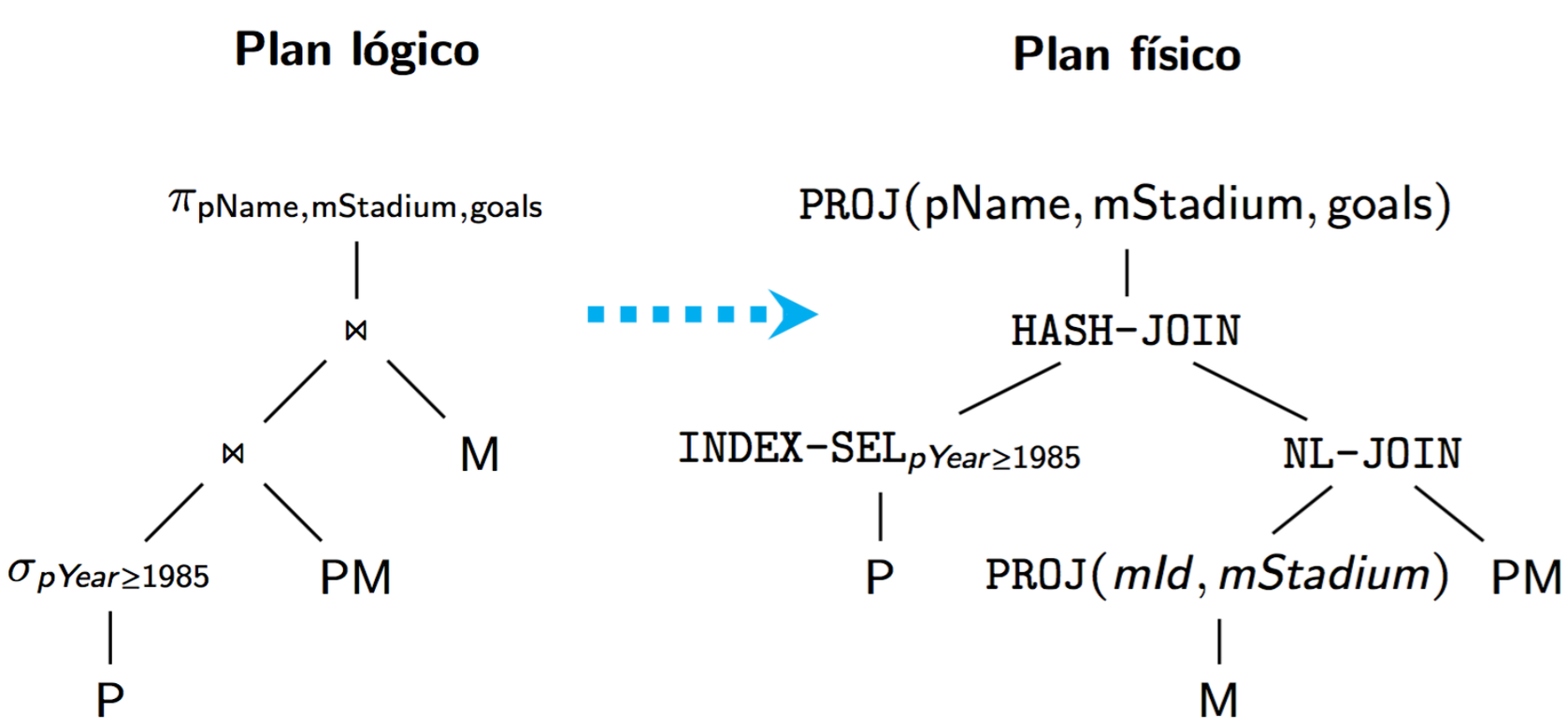
Reescritura



Paso 3

Optimizador

Se encuentra el plan físico más eficiente para la consulta



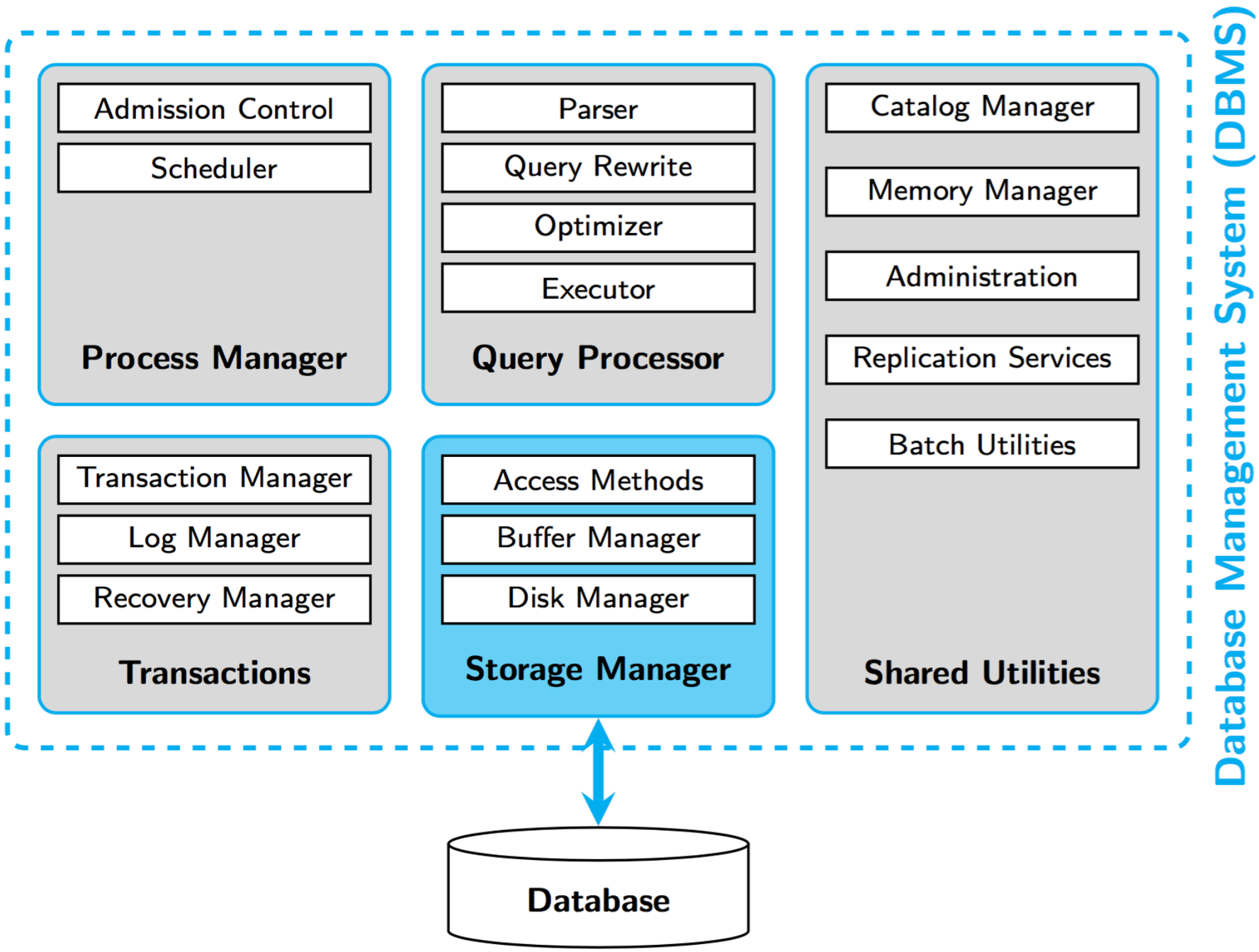
Paso 4

Ejecución

Uso de “pipelining”

Cada operador retorna tuplas a su operador padre “as soon as possible”

Storage Manager



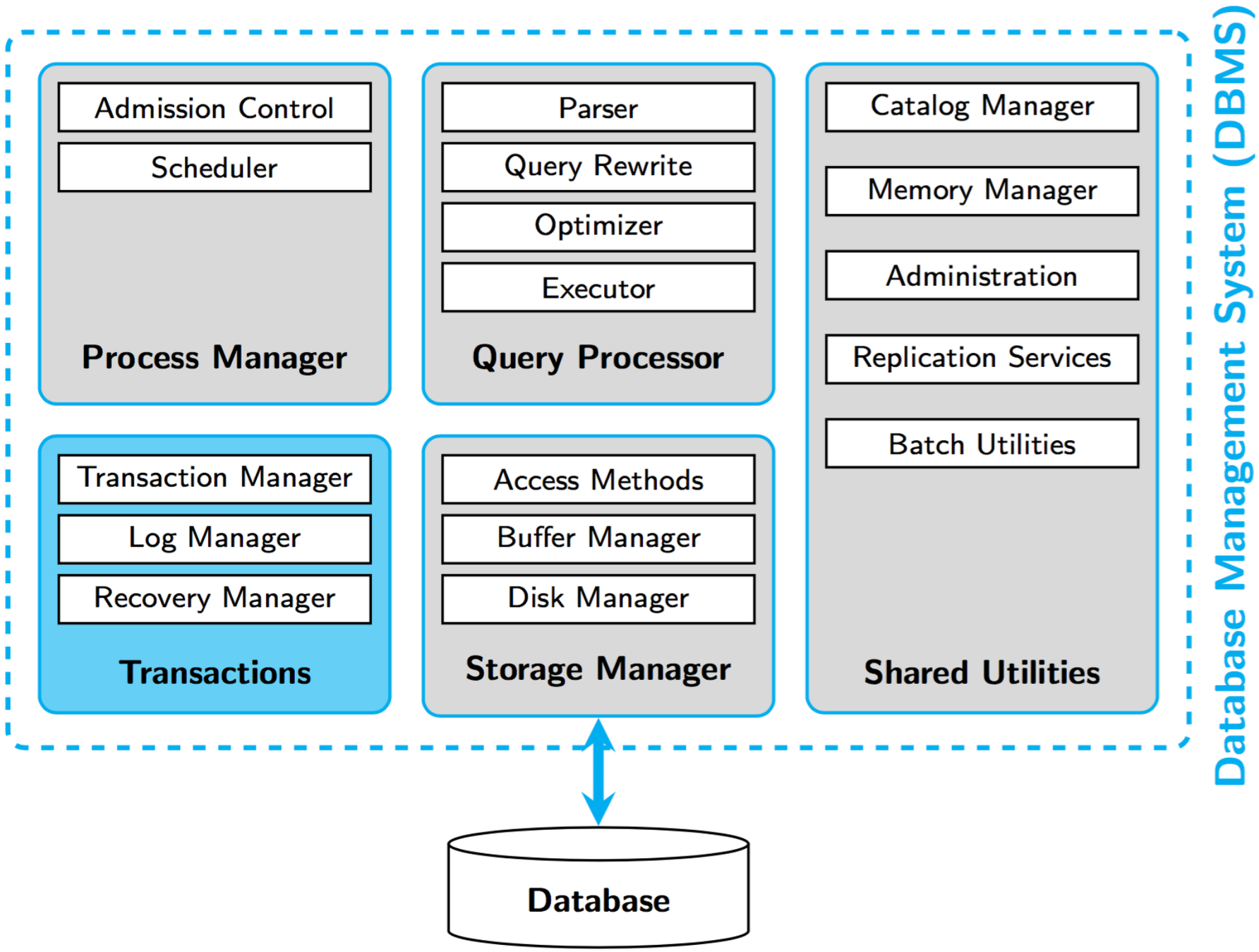
Storage Manager

Disk Manager se encarga del almacenamiento secundario

Buffer Manager se encarga del uso de datos en memoria y optimiza cantidad de I/O al disco duro

Access Methods se encarga de la organización eficiente de los datos

Transactions



Transactions

Componente que asegura las propiedades **ACID**

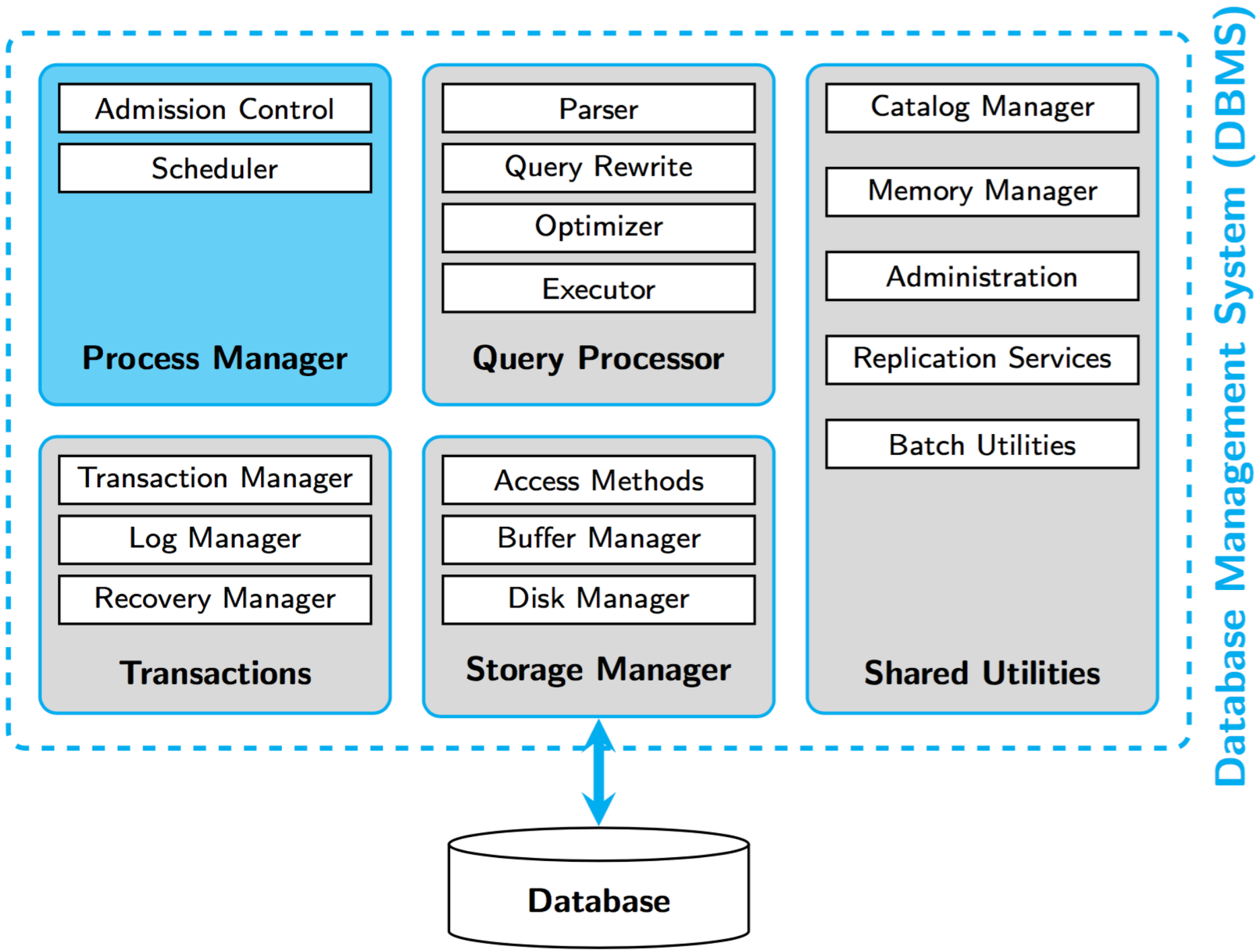
Atomicity
Consistency
Isolation
Durability

Transactions

Transaction Manager se encarga de asegurar Isolation y Consistency

Log y Recovery Manager se encargan de asegurar Atomicity y Durability

Process Manager

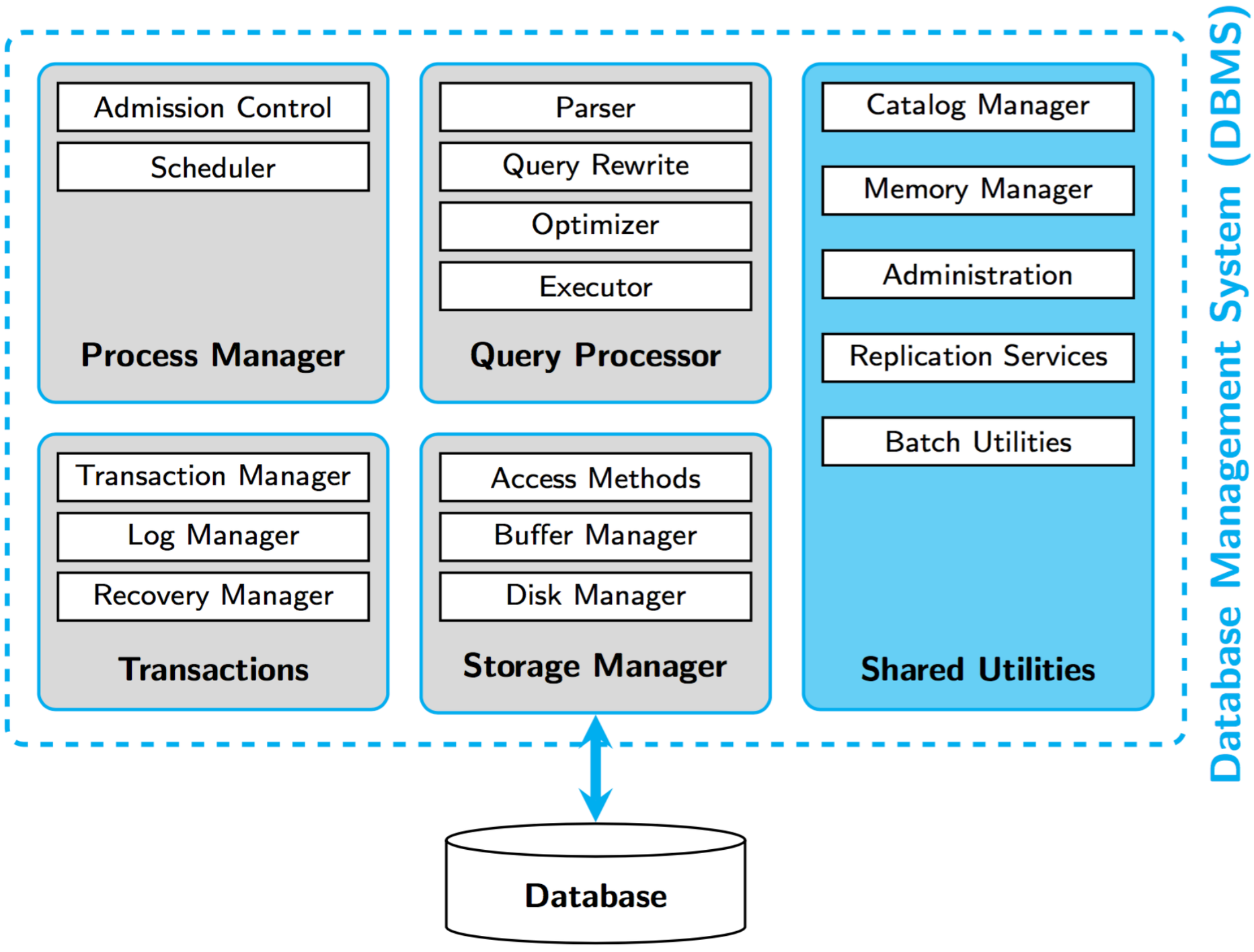


Process Manager

Admisión Control se encarga del manejo de sesiones

Scheduler maneja los procesos y los threads, además de la paralización de consultas

Shared Utilities



Shared Utilities

Catalog Manager maneja los metadatos asociados a los esquemas

Memory Manager Asigna memoria a otras componentes

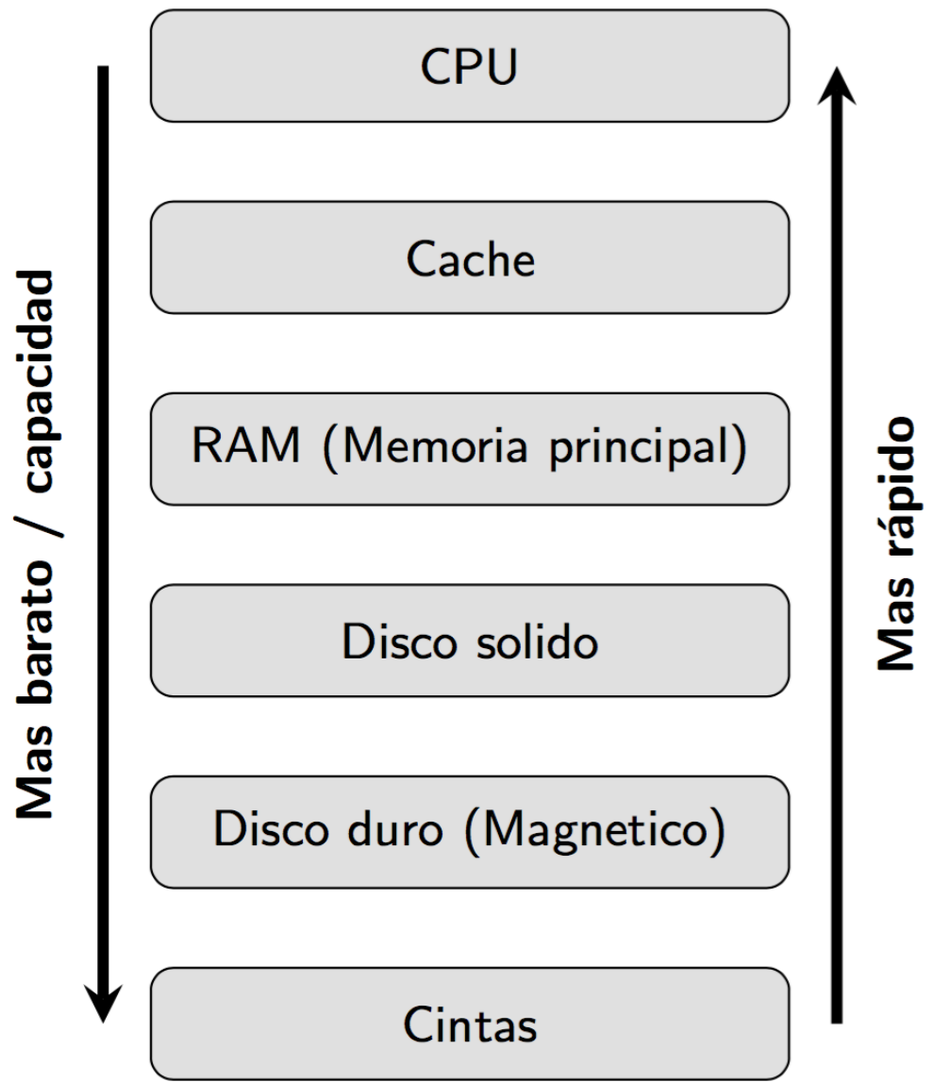
Almacenamiento

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

Página: unidad lógica mínima de almacenamiento para el Sistema de Archivos

Disco Duro y RAM

RAM

SO	SO						

Disco Duro y RAM

RAM

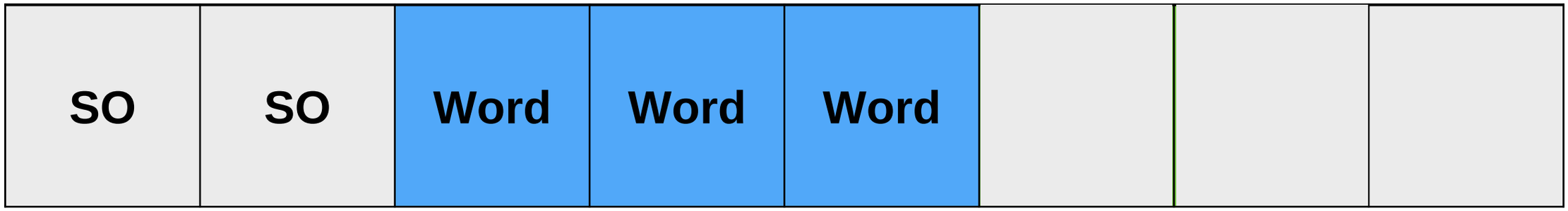
SO	**SO**						

Disco Duro

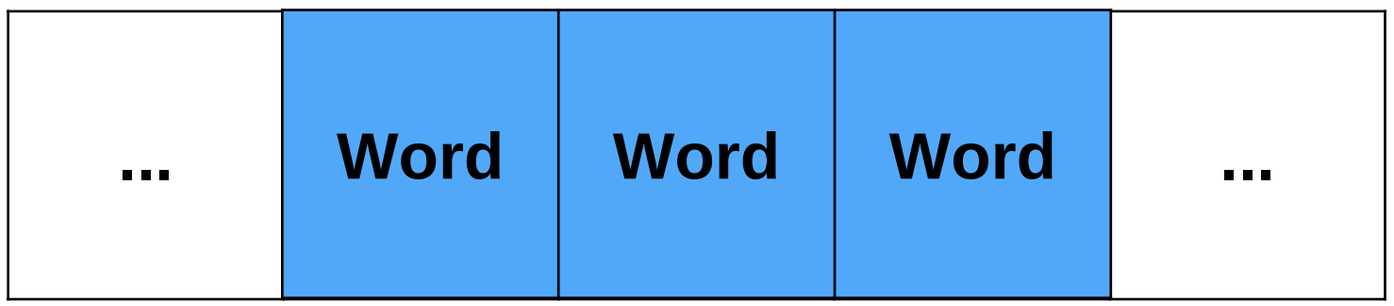
...	**Word**	**Word**	**Word**	**...**

Disco Duro y RAM

RAM



Disco Duro



Disco Duro y RAM

RAM

SO	**SO**	**Word**	**Word**	**Word**	**Spotify**	**Spotify**	

Disco Duro y RAM

RAM

SO	**SO**	**Word**	**Word**	**Word**	**Spotify**	**Spotify**	

Disco Duro

...	**Fifa**	**Fifa**	**Fifa**	**...**

Disco Duro y RAM

RAM

SO	**SO**	**Word**	**Word**	**Word**	**Spotify**	**Spotify**	

Disco Duro (espacio de swap)

Disco Duro

		...	**Fifa**	**Fifa**	**Fifa**	**...**

Disco Duro y RAM

RAM

SO	**SO**	**Word**	**Spotify**	**Spotify**			

Disco Duro (espacio de swap)

Word (Abierto)	**Word** (Abierto)	**...**	**Fifa**	**Fifa**	**Fifa**	**...**	

Disco Duro

Disco Duro y RAM

RAM

SO	**SO**	**Word**	**Spotify**	**Spotify**	**Fifa**	**Fifa**	**Fifa**

Disco Duro (espacio de swap)

Disco Duro

Word (Abierto)	**Word** (Abierto)	**...**	**Fifa**	**Fifa**	**Fifa**	**...**

Disco Duro y RAM

RAM

SO	**SO**	**Word**	**Spotify**	**Spotify**	**Fifa**	**Fifa**	**Fifa**

Disco Duro (espacio de swap)

Word (Abierto)	**Word** (Abierto)				**...**

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

Demora en Búsqueda

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

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

Cómo se guarda una tabla

Supongamos una tabla **T(a int, b text)** con 9 tuplas

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**		

Costo I/O (Input/Output)

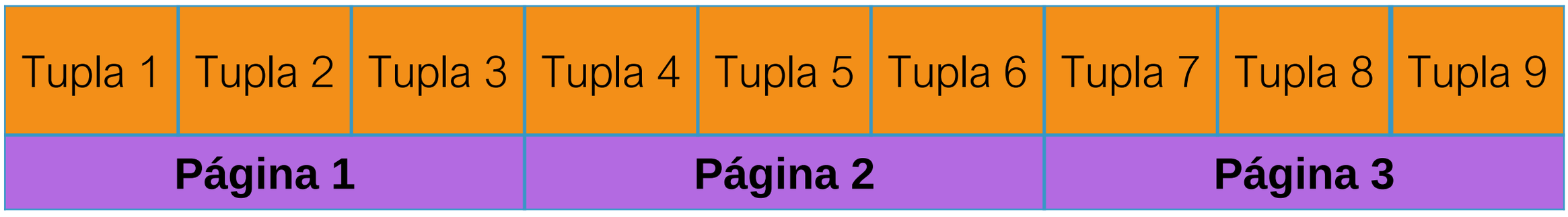
¡El costo más grande de las bases de datos es traer las páginas del disco duro a memoria RAM!

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



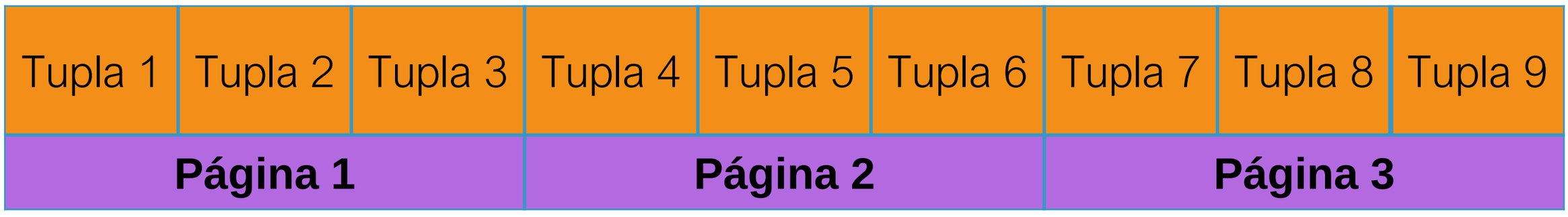
¿Cuál es el costo en I/O de hacer la siguiente consulta?

SELECT * FROM T

El costo es 3, porque debo leer las 3 páginas

Costo de una consulta

Disco Duro



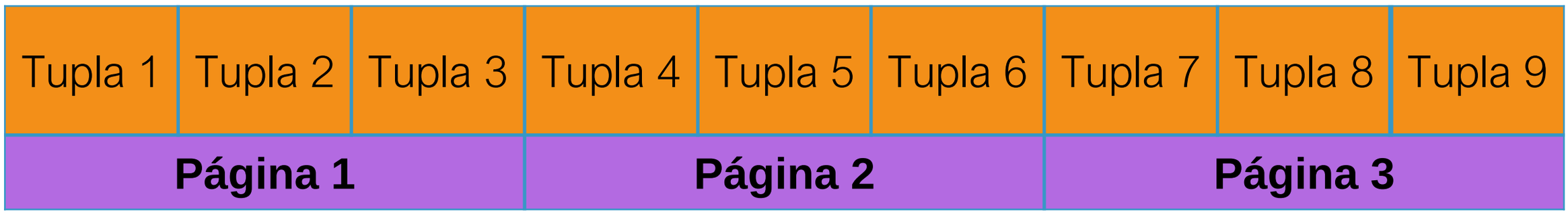
¿Cuál es el costo en I/O de hacer la siguiente consulta?

```
SELECT T.a FROM T
```

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

Costo de una consulta

Disco Duro



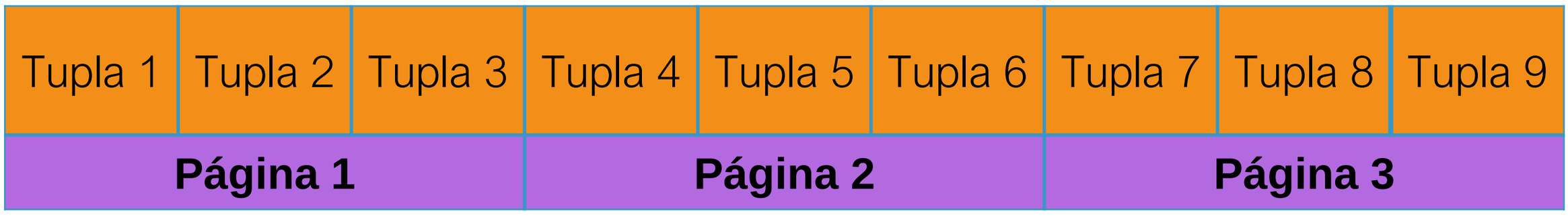
¿Cuál es el costo en I/O de hacer la siguiente consulta?

```
SELECT * FROM T WHERE T.a = 4
```

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

Costo de una consulta

Disco Duro



¿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
```

¿Podemos hacer esto mejor?

Índices

Índices

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

Idealmente un índice debe caber en RAM

Índices

CREATE INDEX <nombre índice> ON table <atributo>

Las llaves primarias están indexadas por defecto

Índices

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 puede devolver las tuplas, o punteros que me indican donde encontrarlas

Esto se logra hacer de manera rápida

Consultas a Optimizar

Consultas por valor

```
SELECT *
FROM Table
WHERE Table.value = ‘value’
```

Consultas por rango

```
SELECT *
FROM Table
WHERE Table.value >= ‘value’
```

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

Flujo

El índice encontrará rápidamente la tupla que que cumple con la condición

El índice tiene dos opciones para responder:

- Nos va a devolver la tupla que buscamos
- Nos va a devolver un puntero hacia la dirección de la página en el disco duro donde encontraremos la tupla

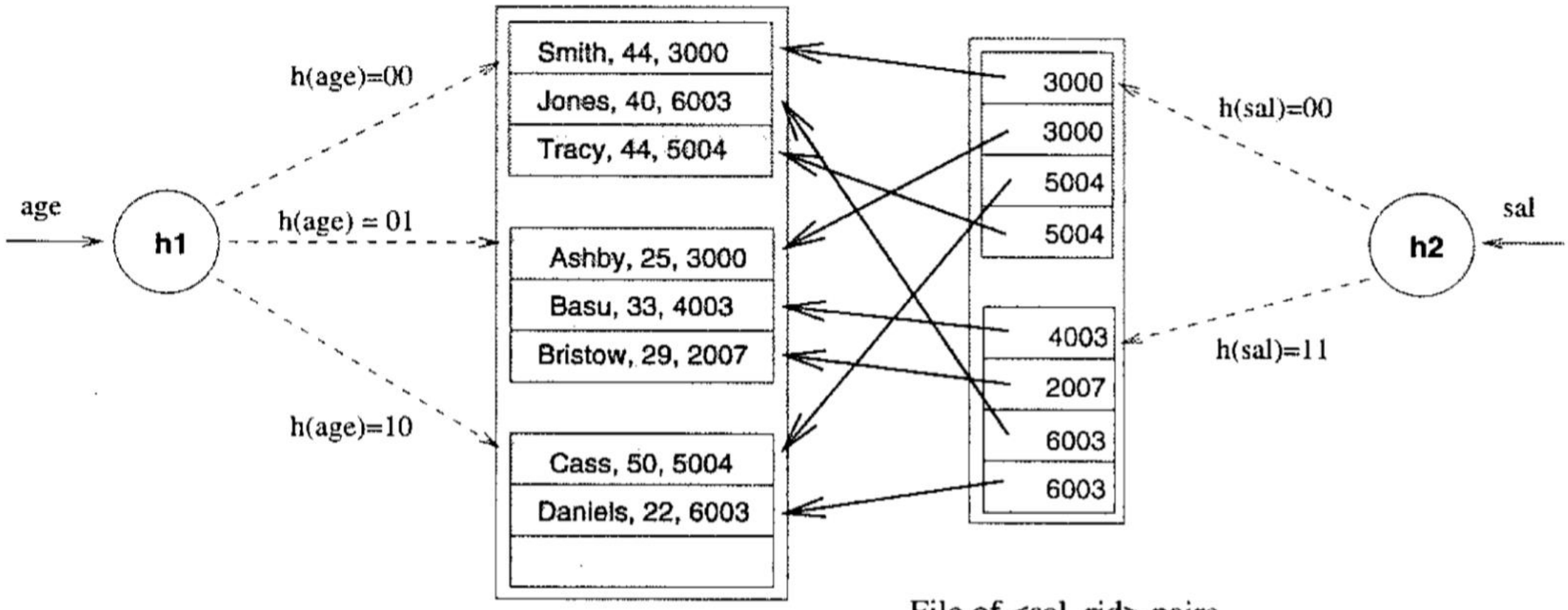
Índices

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

Índices


h(age)=00
Smith, 44, 3000
Jones, 40, 6003
Tracy, 44, 5004
age
h1
h(age) = 01
Ashby, 25, 3000
Basu, 33, 4003
Bristow, 29, 2007
h(age)=10
Cass, 50, 5004
Daniels, 22, 6003
3000
3000
5004
5004
4003
2007
6003
6003
h(sal)=00
sal
h2
h(sal)=11
File of <sal, rid> pairs
Employees file hashed on age
hashed on sal

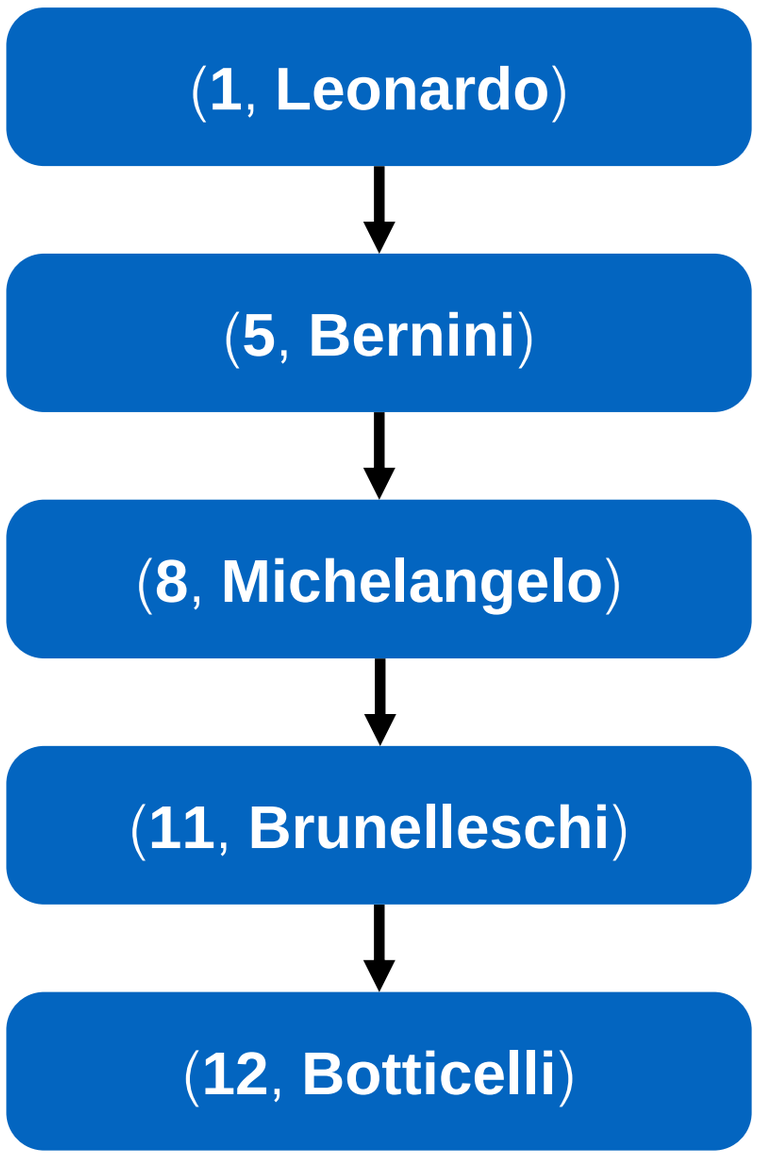
Repaso (o spoiler) de EDD

Queremos guardar una lista de tuplas en Python:

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

Repaso (o spoiler) de EDD

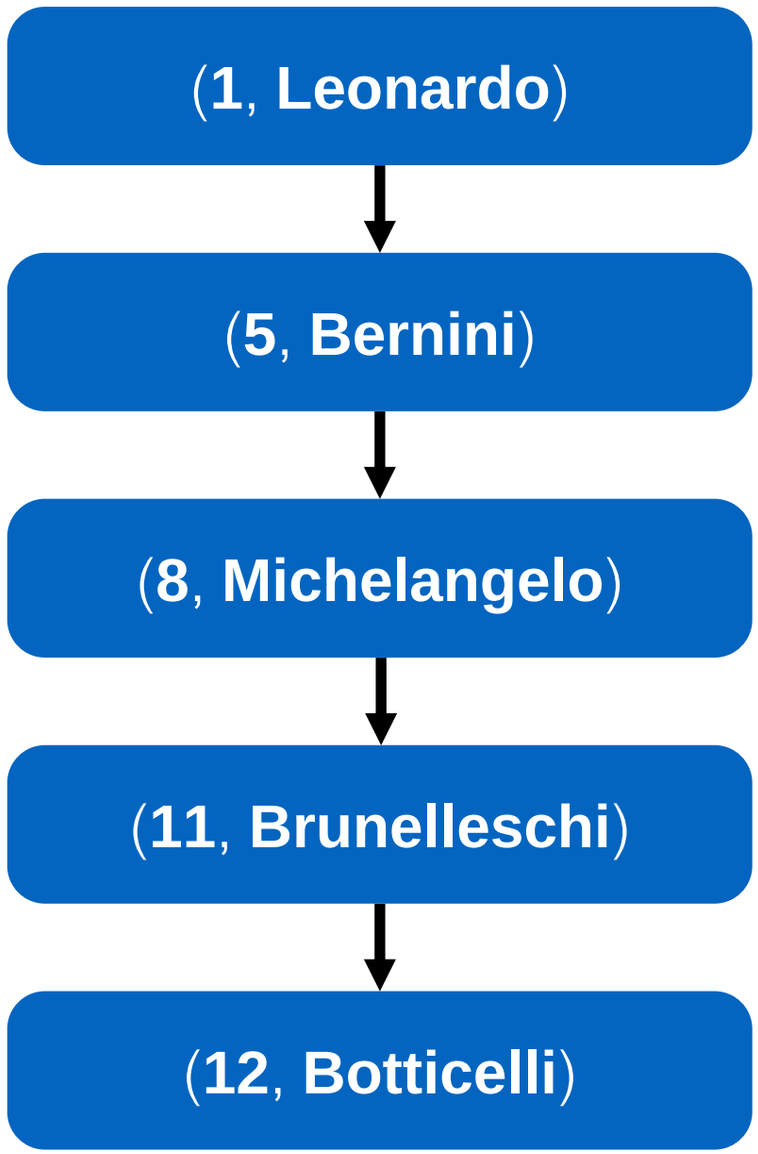
Lista ligada (versión básica)



Repaso (o spoiler) de EDD

Lista ligada (versión básica)

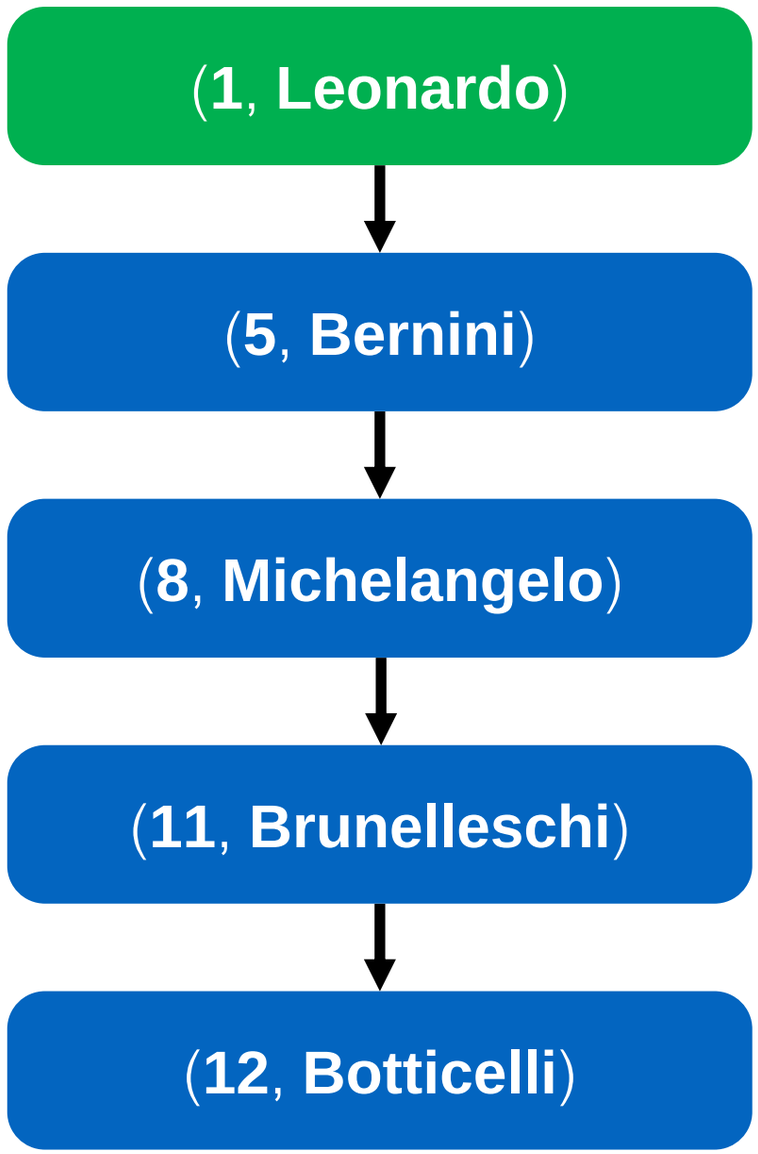
Supongamos que queremos el artista con identificador 11



Repaso (o spoiler) de EDD

Lista ligada (versión básica)

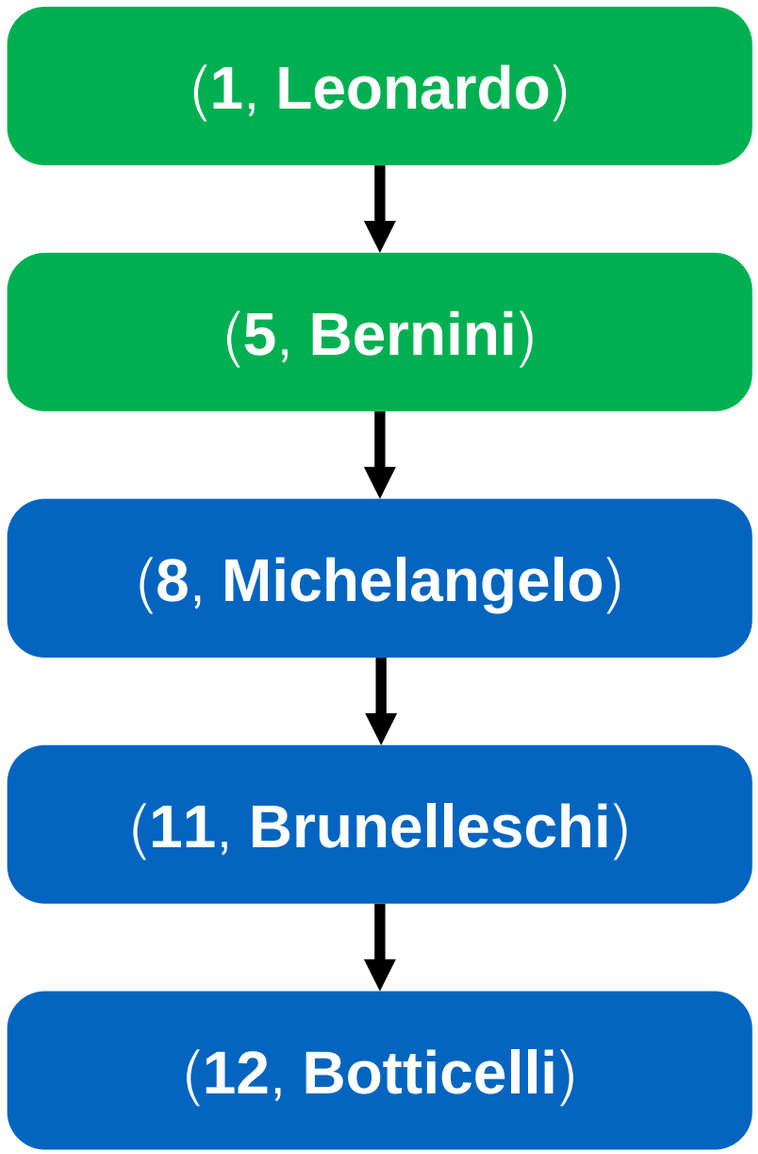
Supongamos que queremos el artista con identificador 11



Repaso (o spoiler) de EDD

Lista ligada (versión básica)

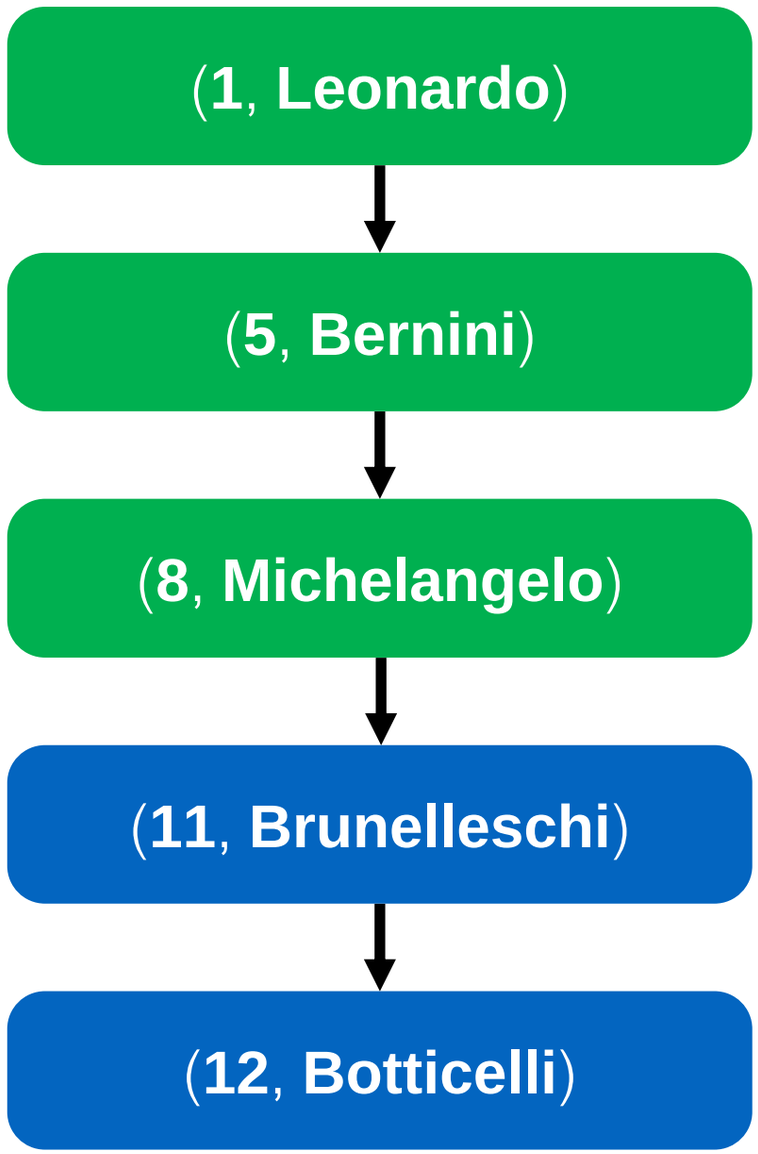
Supongamos que queremos el artista con identificador 11



Repaso (o spoiler) de EDD

Lista ligada (versión básica)

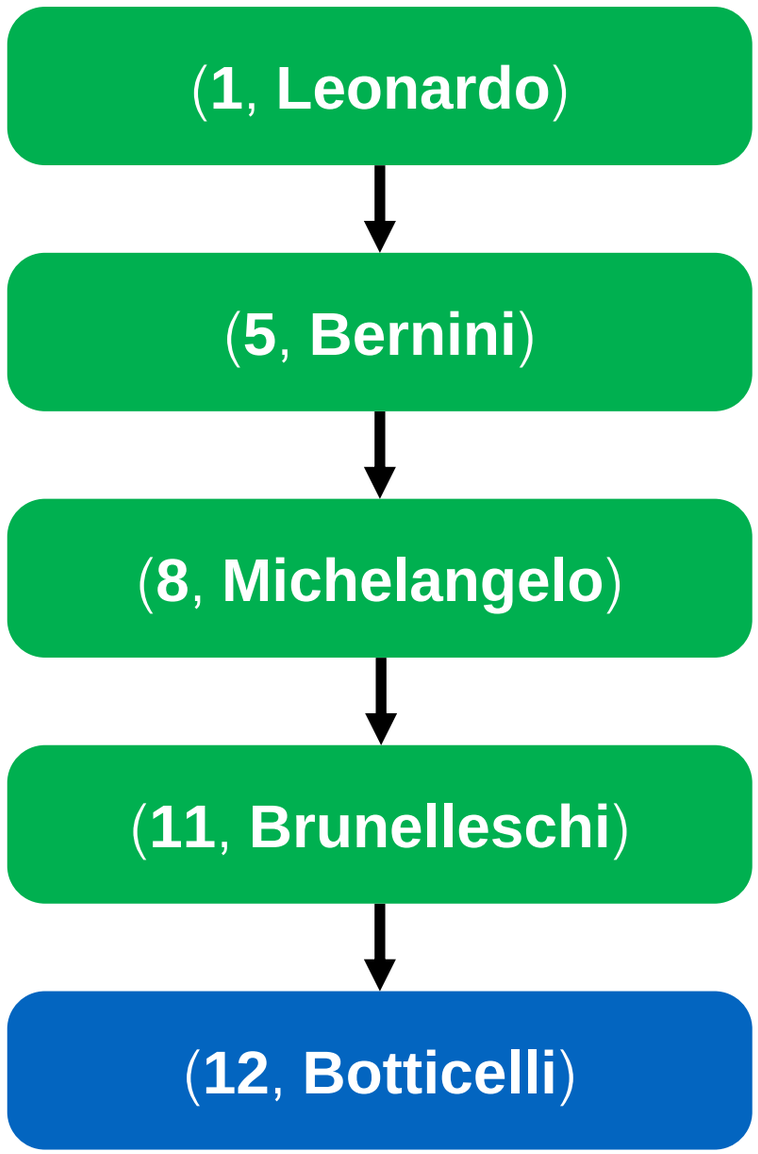
Supongamos que queremos el artista con identificador 11



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)

En este caso casa tupla va a parar a un casillero según su **id**

Para esto utilizamos una función de **hash**

Repaso (o spoiler) de EDD

Tablas de Hash

La función de **hash** 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 (**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 (**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 (**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 (**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 una cantidad de pasos cercano a 1

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

Recordemos que 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

Y que rápido significa hacer la menor cantidad de I/O posible

Hash Index

Ejemplo

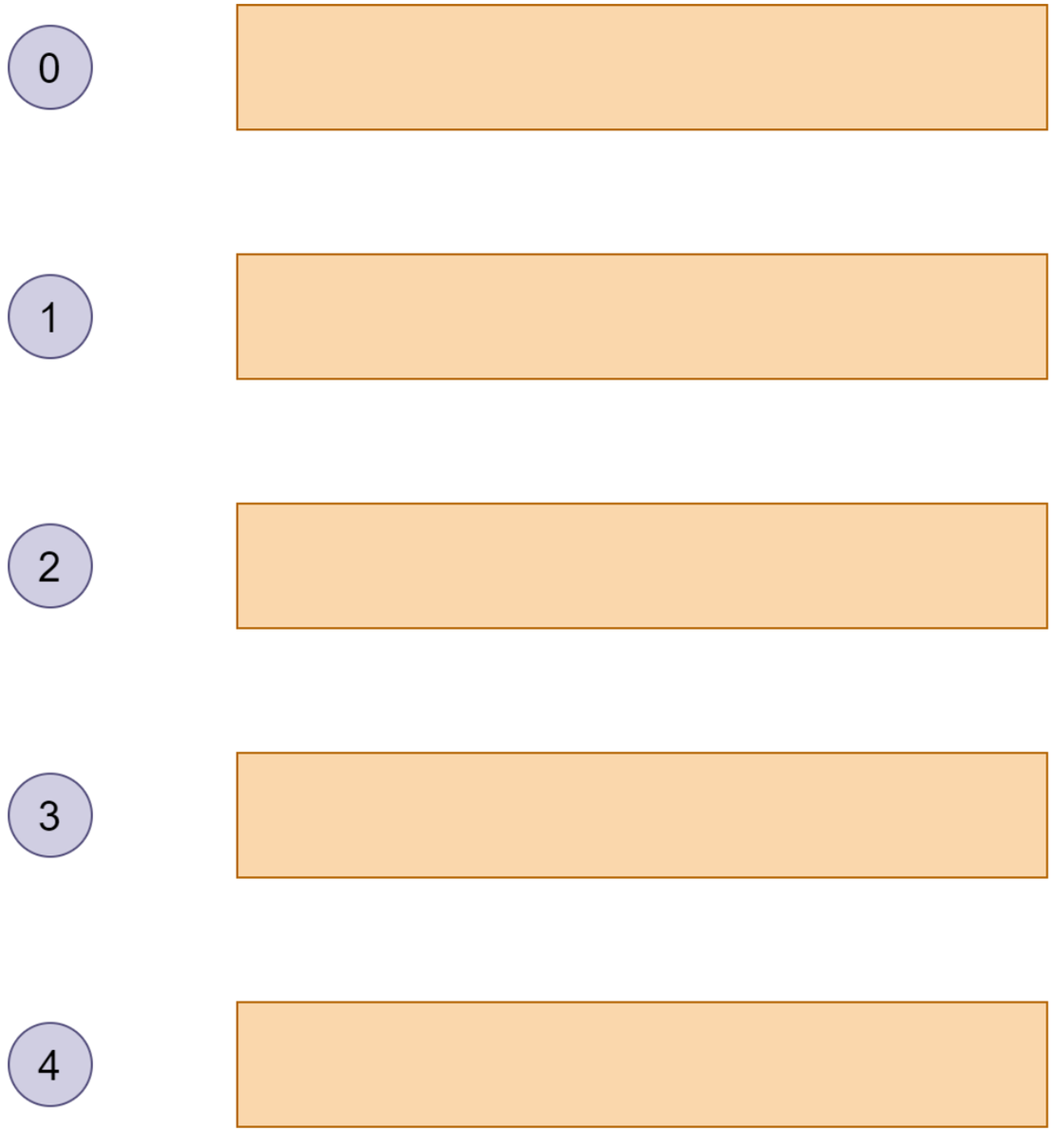
Vamos a retomar el ejemplo de los artistas, pero ahora insertando las tuplas a una base de datos

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



Hash Index

Ejemplo

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

Hash Index

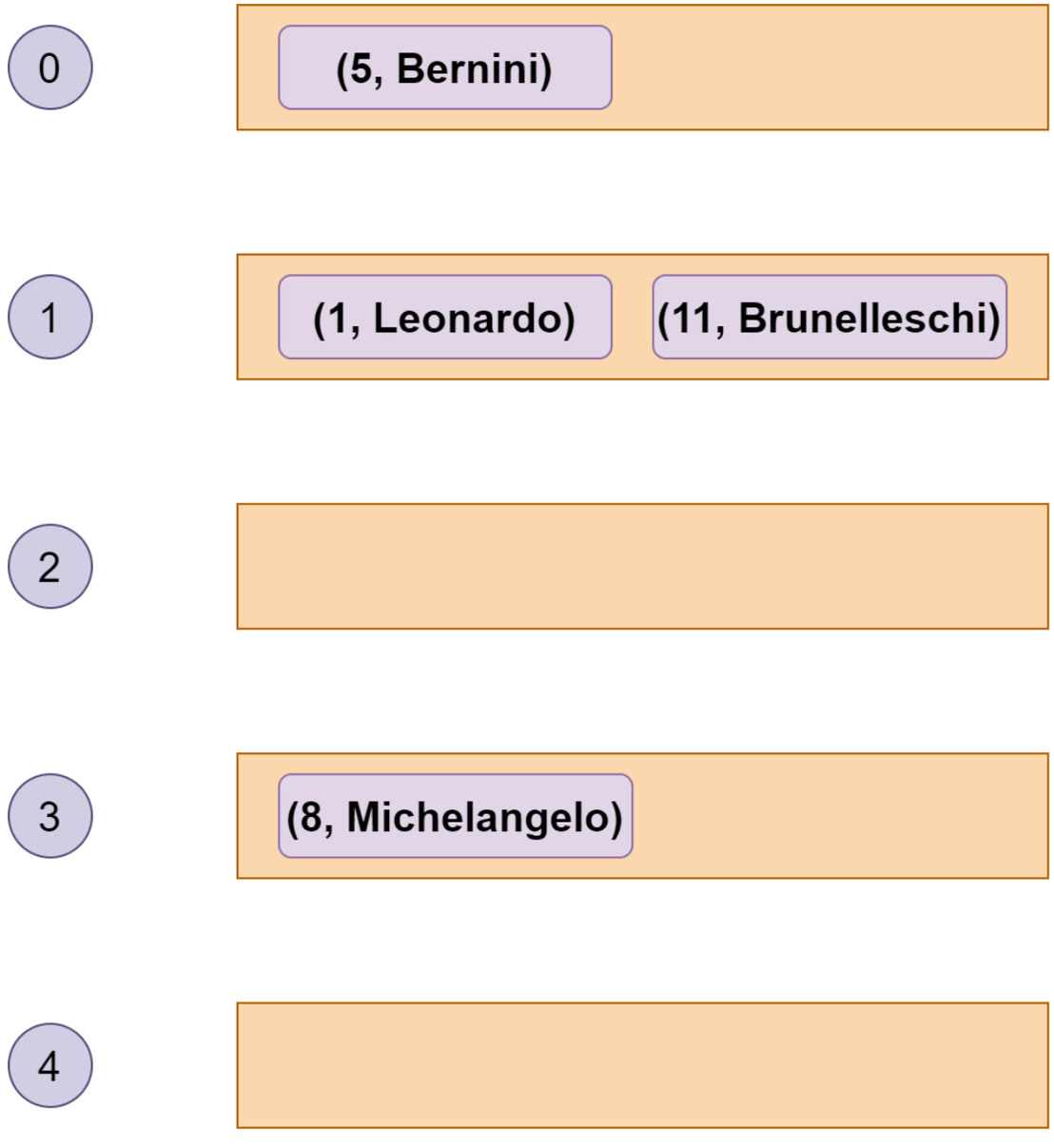
Ejemplo

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

Hash Index

Ejemplo

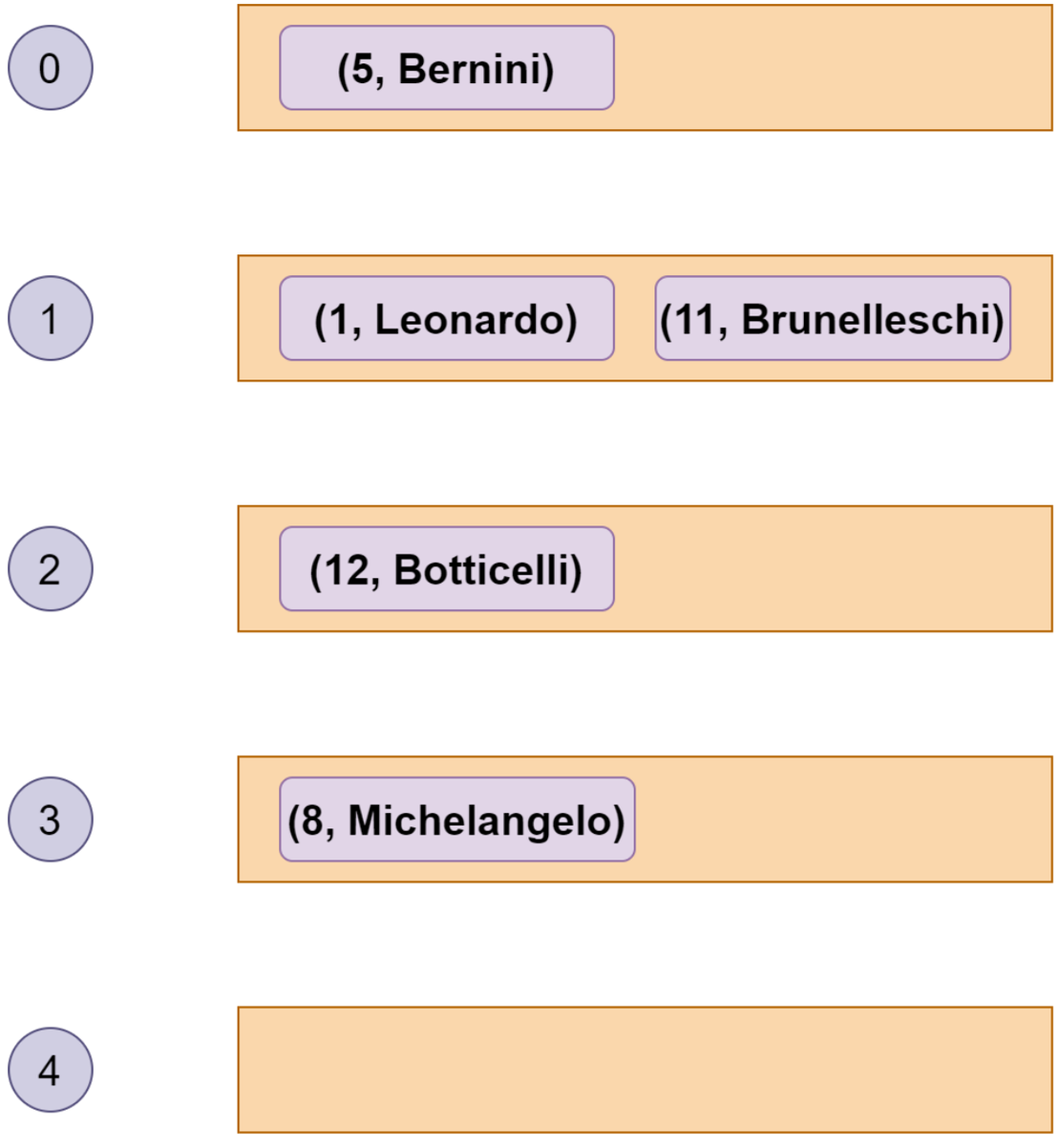
Artista.aid	Artista.nombre
1	Leonardo
5	Bernini
8	Michelangelo
11	Brunelleschi
12	Botticelli
16	Giotto



Hash Index

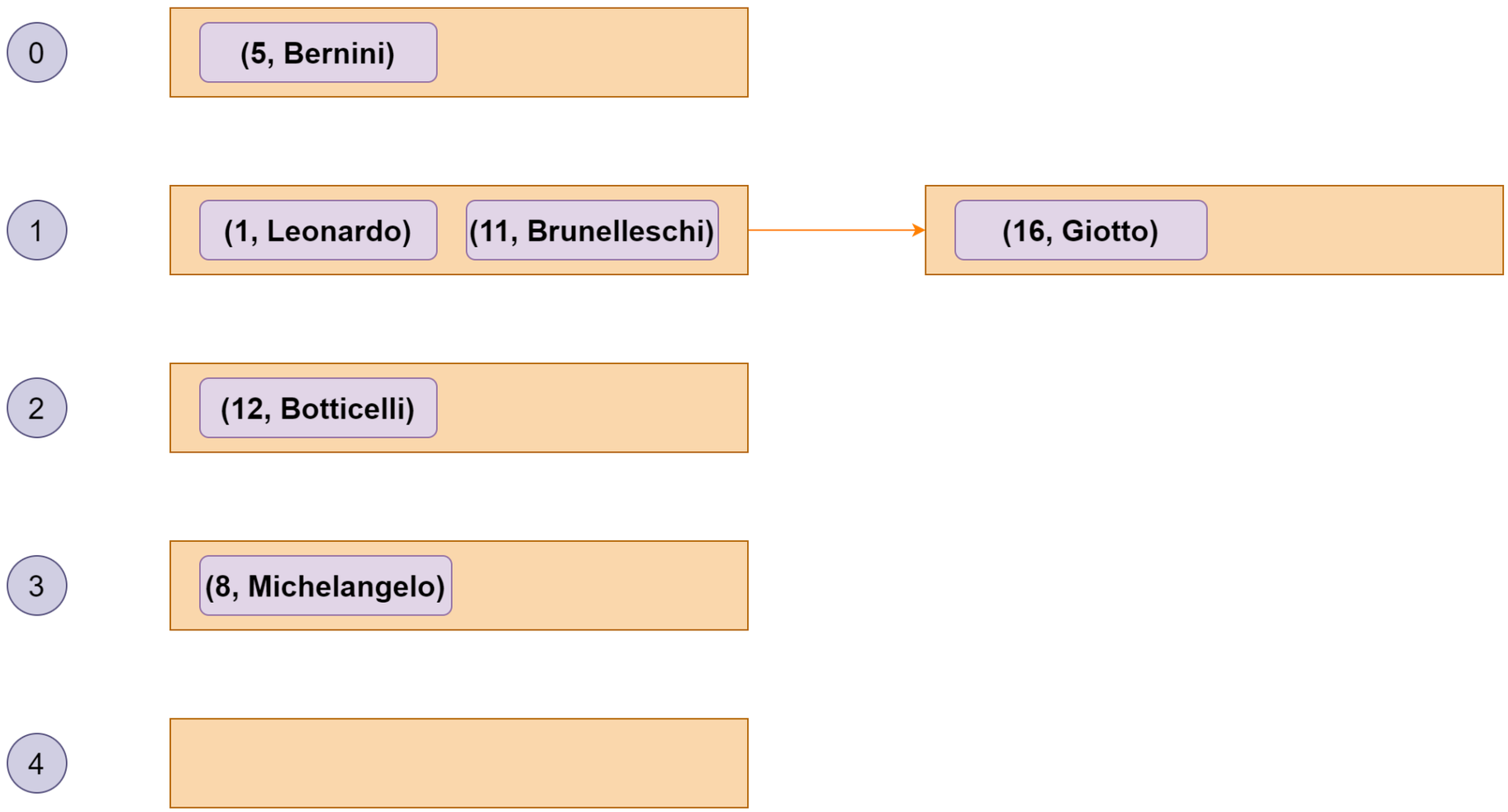
Ejemplo

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



Hash Index

Ejemplo



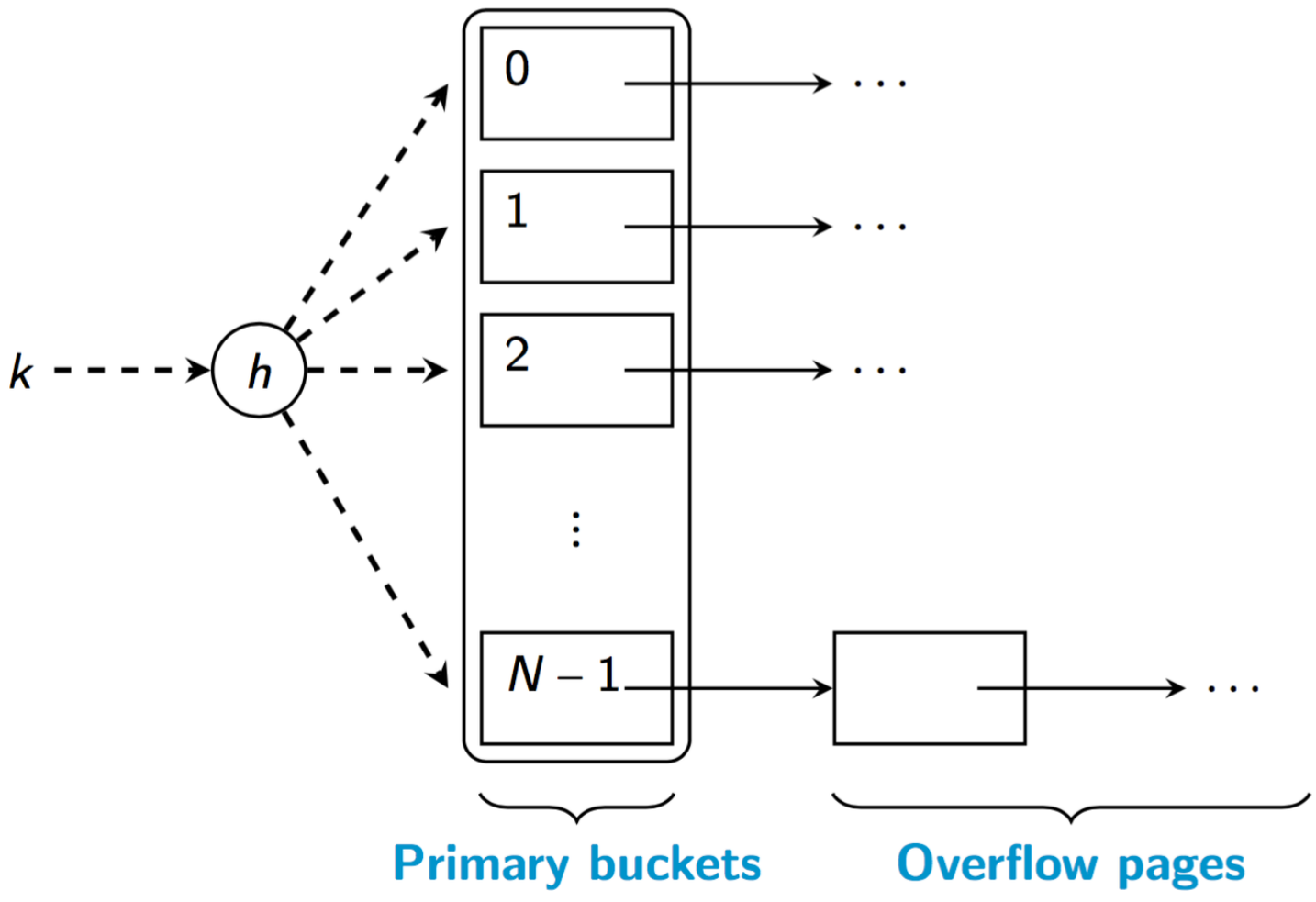
Hash Index

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:

$$\mathbf{h}: \text{dominio}(k) \rightarrow [0, \ldots, n - 1]$$

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 las overflow pages

Hash Index

¿Y qué pasa con las colisiones?

Hash Index

Buscando a una tupla por el search key

Costo del Hash Index en I/O ~ $\frac{|(DataEntries)|}{B \cdot N}$

B = Elementos que caben en una página
N = Número de buckets
|**Data Entries**| = Número de elementos guardados

¿Por qué?

Hash Index

¿Qué pasa si hay muchas colisiones?

Hash Index Dinámico

Existen tipos de Hash Index que crecen a medida que es necesario:

- Extendable Hash Index
- Linear Hash Index

En general, tienden a mantener un costo de búsqueda constante (en I/O) cercano a 1

Hash Index

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

B+ Tree Index

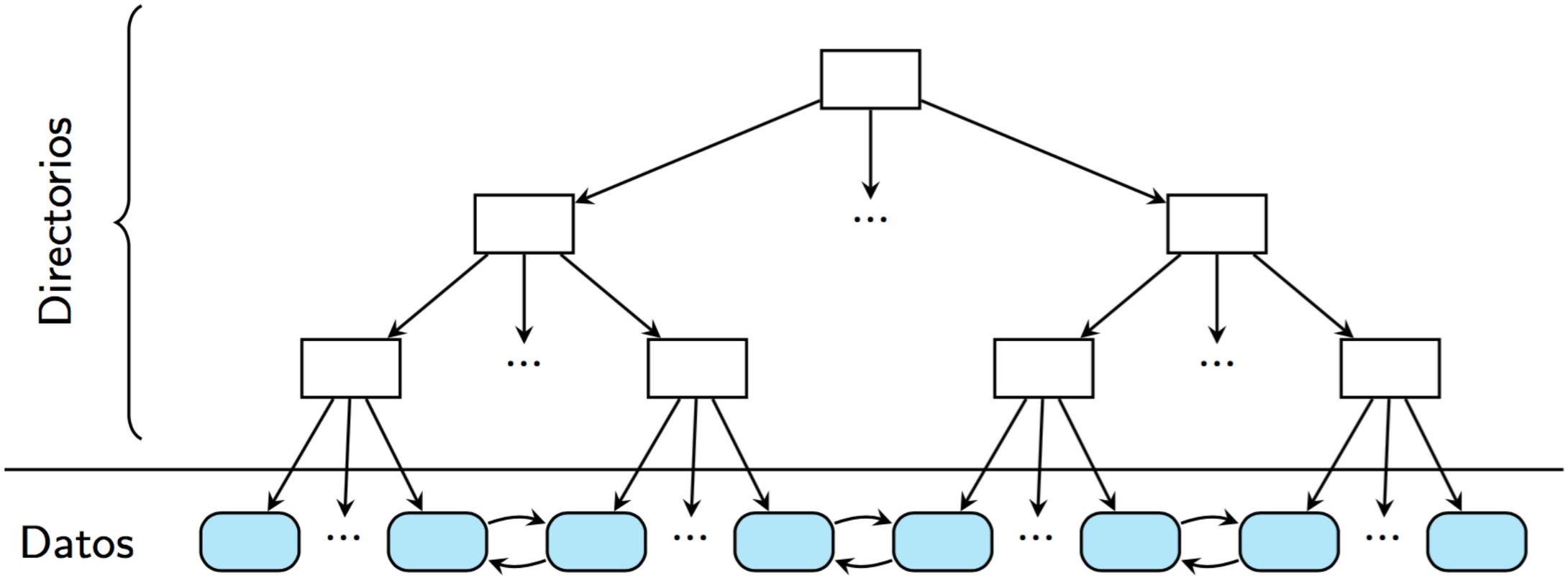
"*B+ Trees are by far the most important access path structure in database and file systems*",
Gray y Reuter (1993).

B+ Tree Index

Es un índice basado en un árbol:

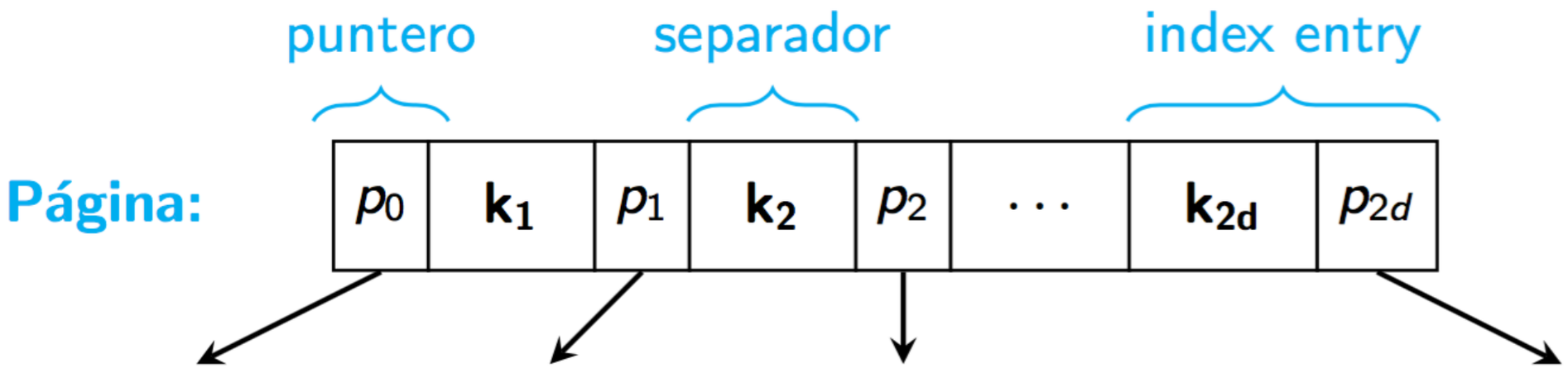
- Las páginas del disco representan nodos del árbol
- **Las tuplas** están en las hojas
- Provee un tiempo de búsqueda logarítmico
- Se comporta bien en consultas de rango
- Es el índice que usa Postgres sobre las llaves primarias (y en general, todos los sistemas)

B+ Tree Index



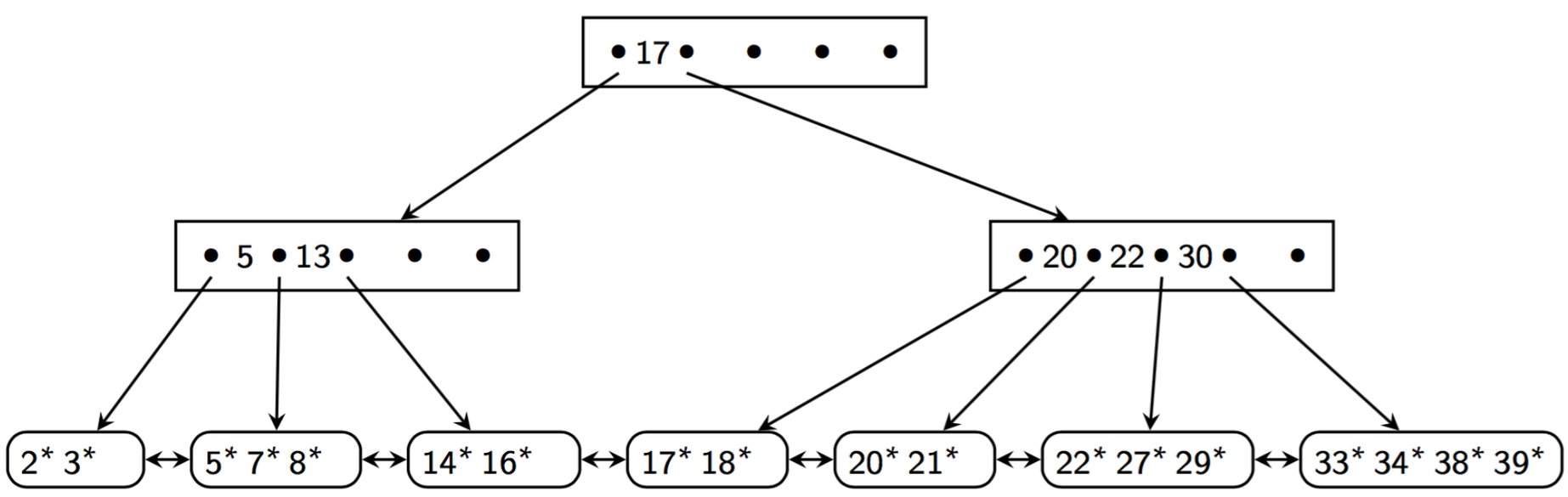
B+ Tree Index

Cada nodo tiene la siguiente forma:



B+ Tree Index

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

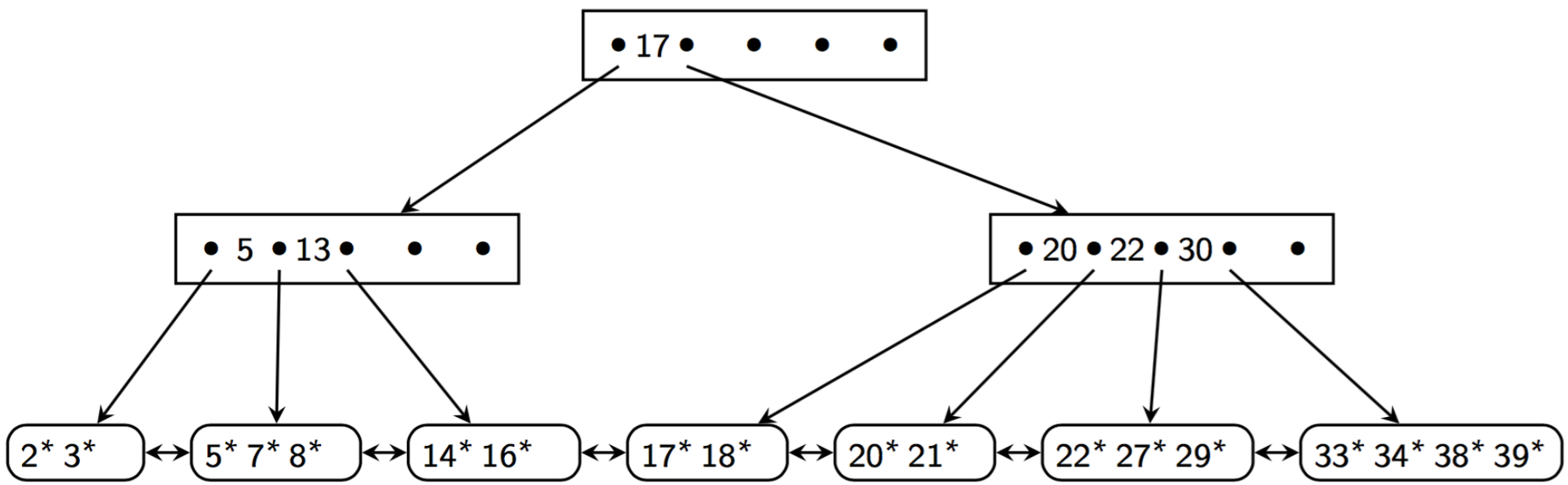


B+ Tree Index

Ejemplo: consulta por un valor

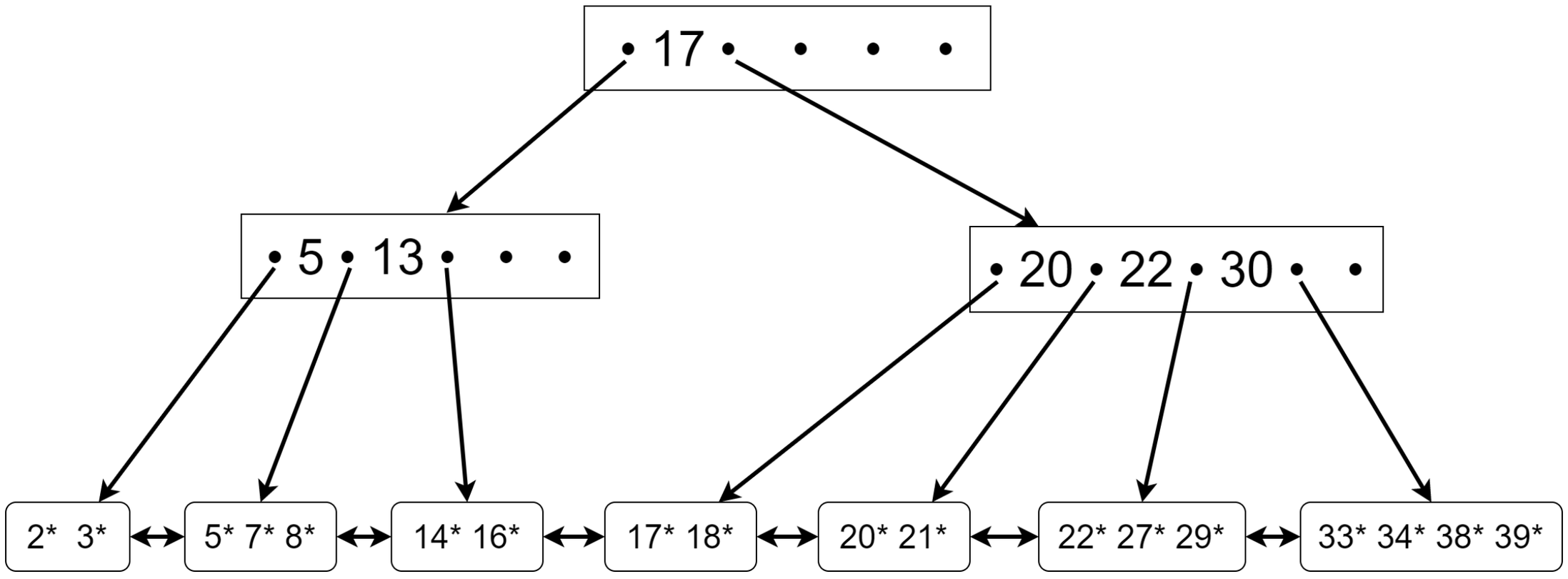
Supongamos que este índice también almacena artistas que están indexados por su **aid**

Queremos el artista con **aid** = 21



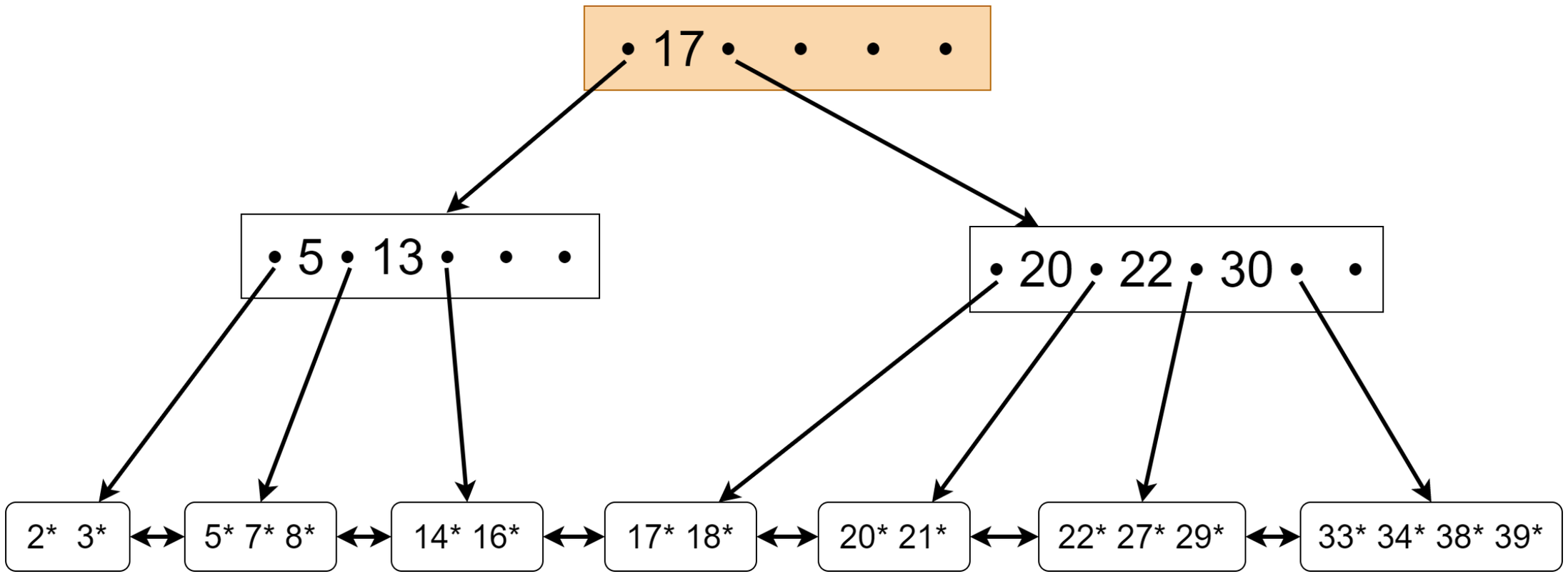
B+ Tree Index

Ejemplo: consulta por un valor



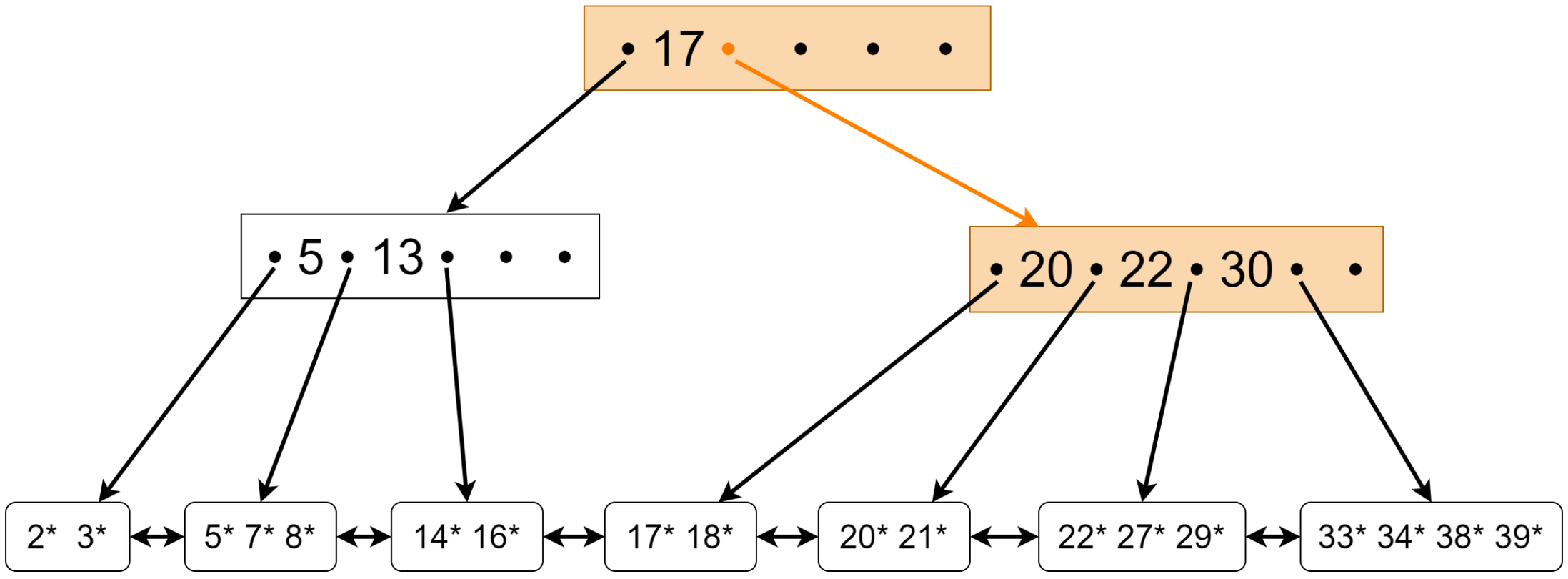
B+ Tree Index

Ejemplo: consulta por un valor



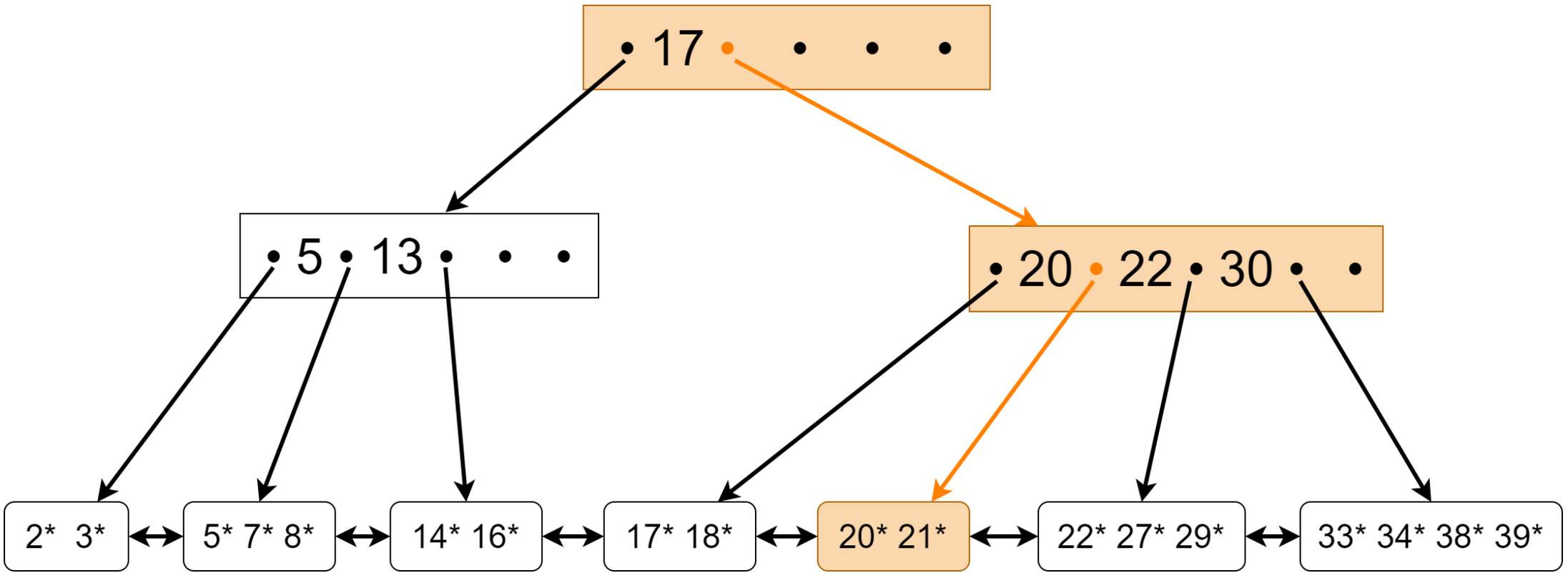
B+ Tree Index

Ejemplo: consulta por un valor



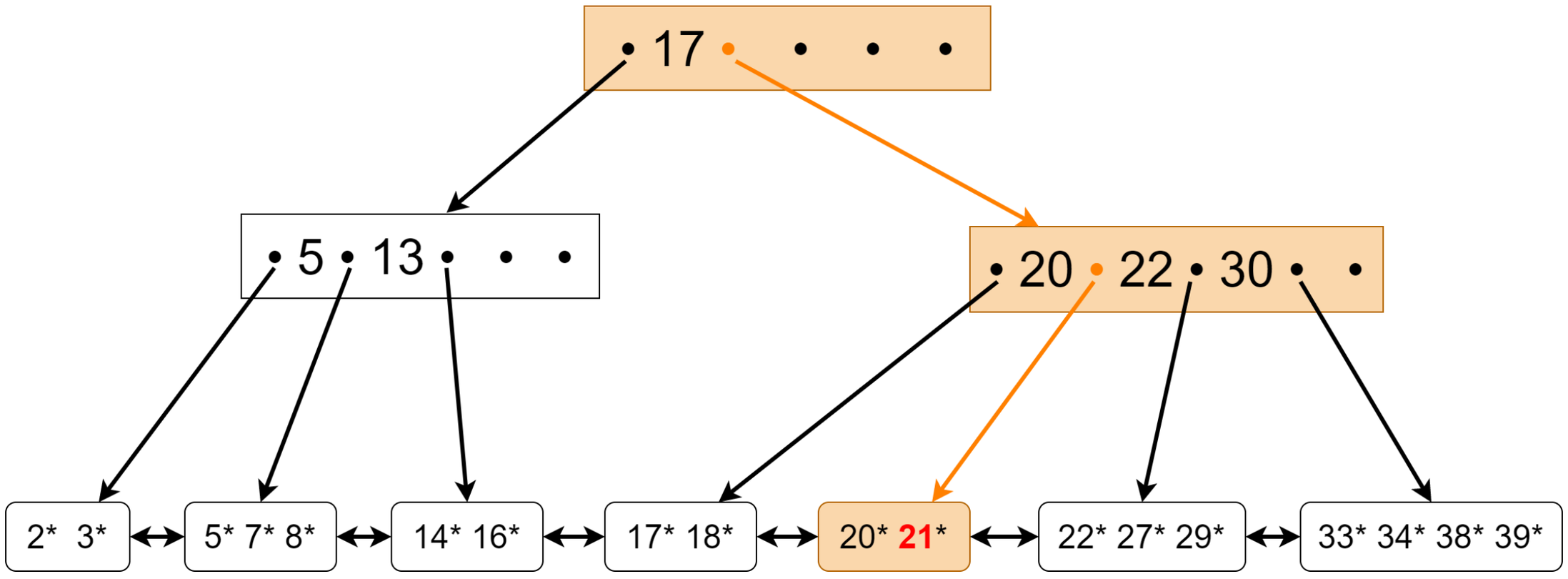
B+ Tree Index

Ejemplo: consulta por un valor



B+ Tree Index

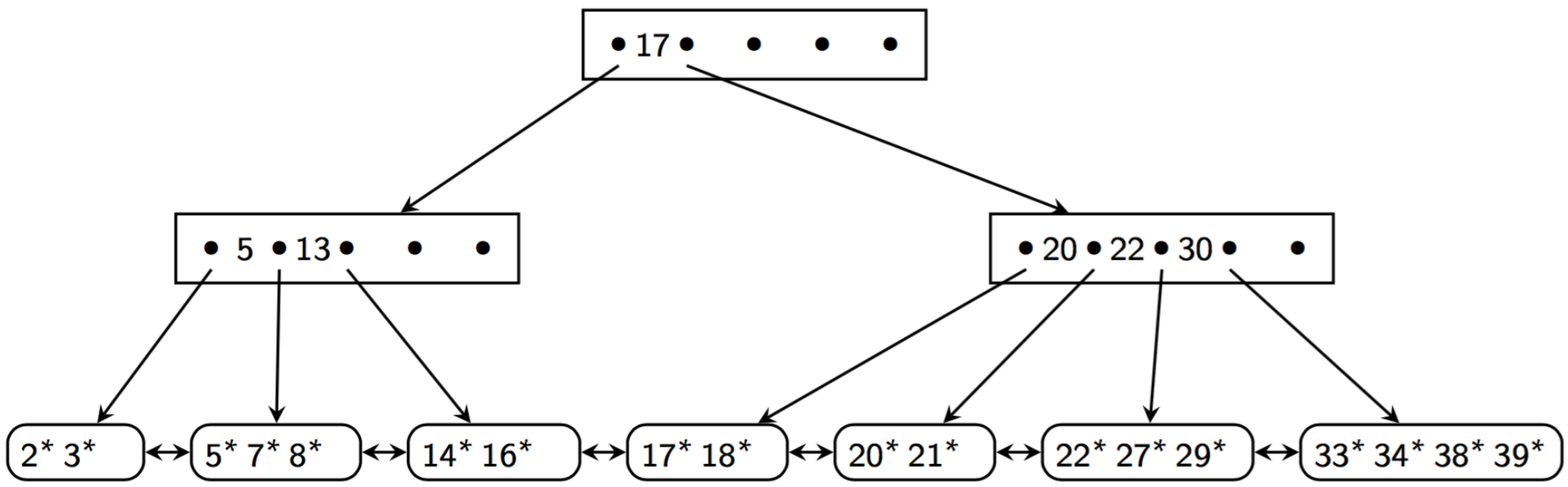
Ejemplo: consulta por un valor



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

B+ Tree Index

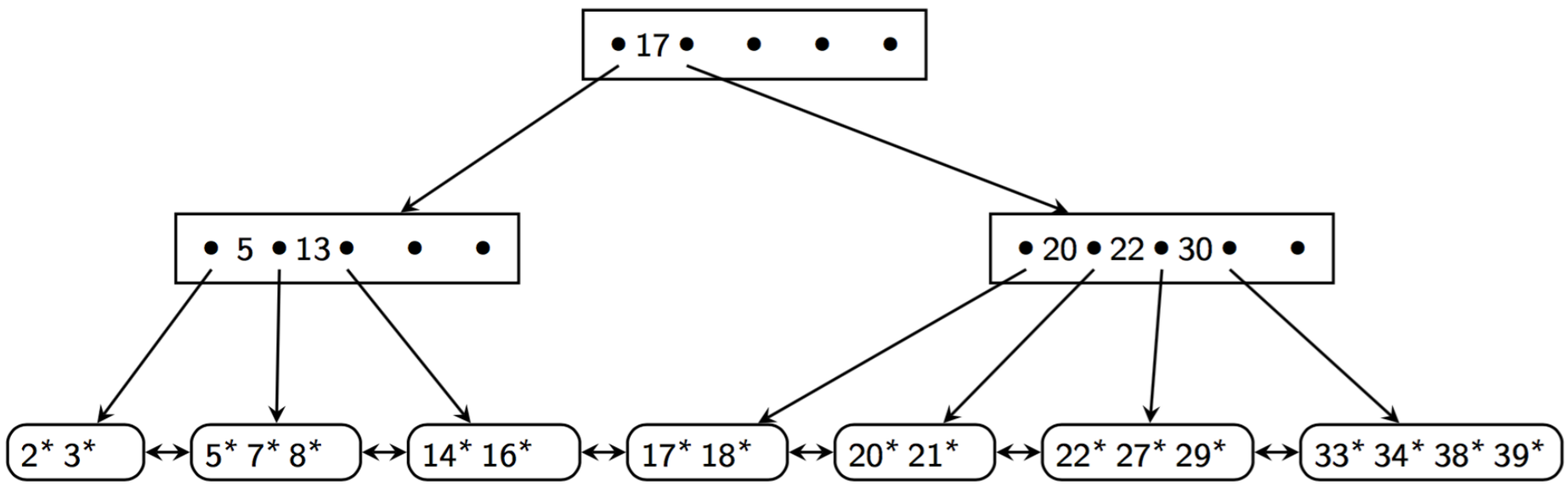
Sabemos que las tuplas van a estar ordenadas según el atributo indexado, y además las hojas están encadenadas unas con otras



B+ Tree Index

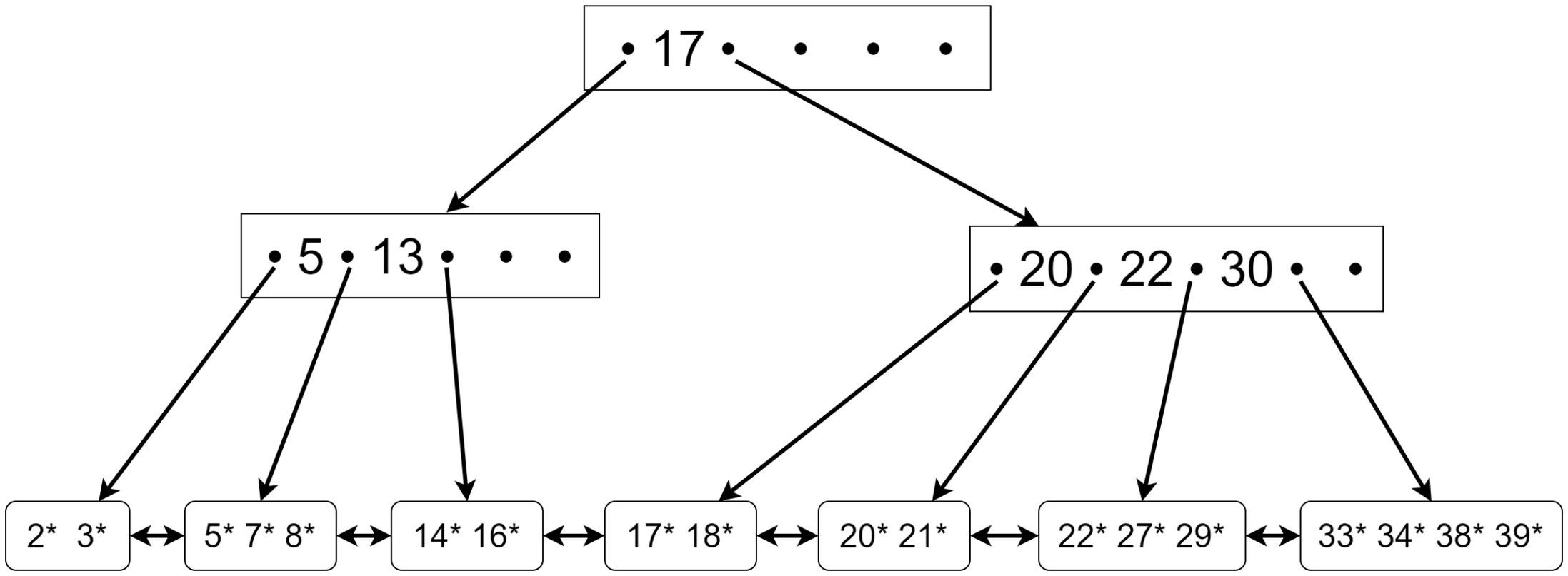
Ejemplo: consulta de rango

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



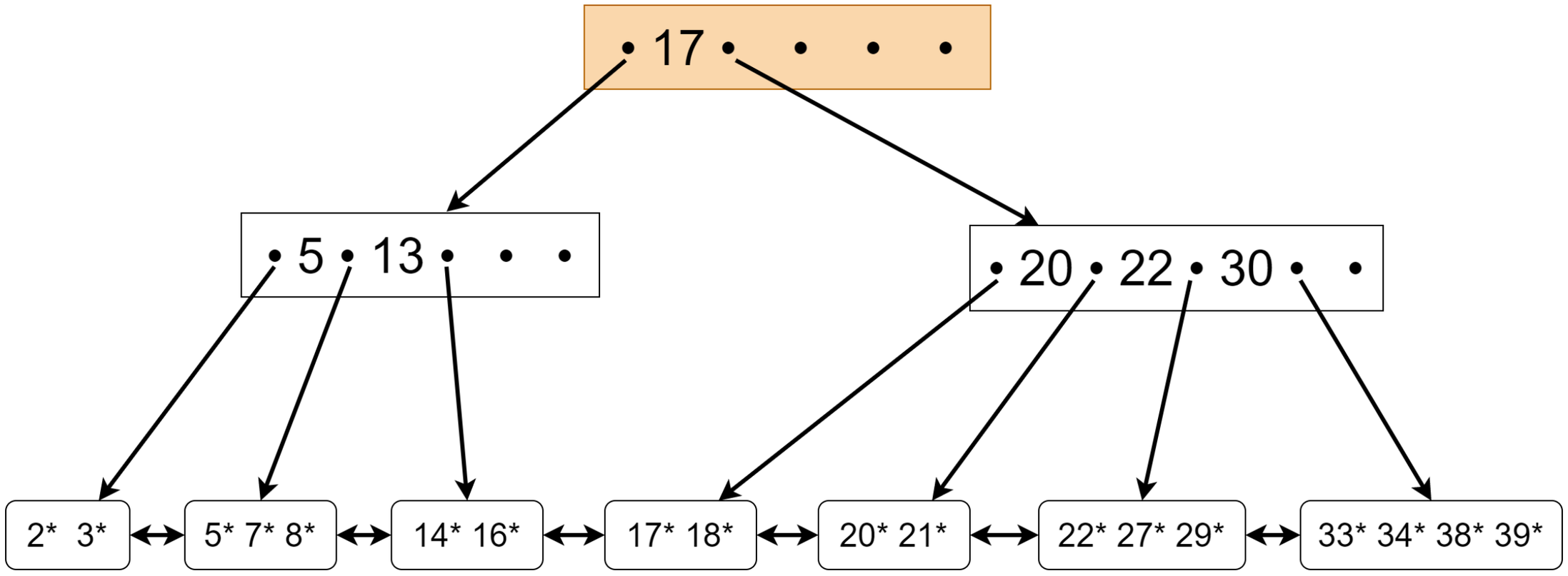
B+ Tree Index

Ejemplo: consulta de rango



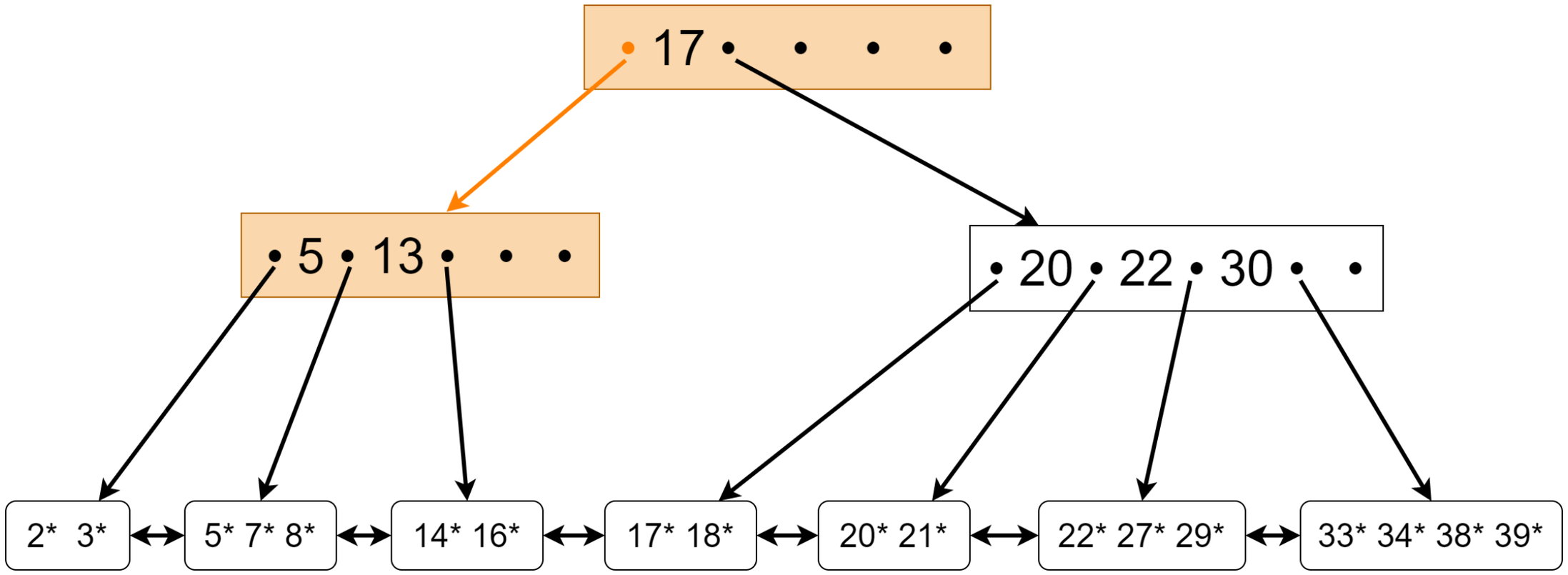
B+ Tree Index

Ejemplo: consulta de rango



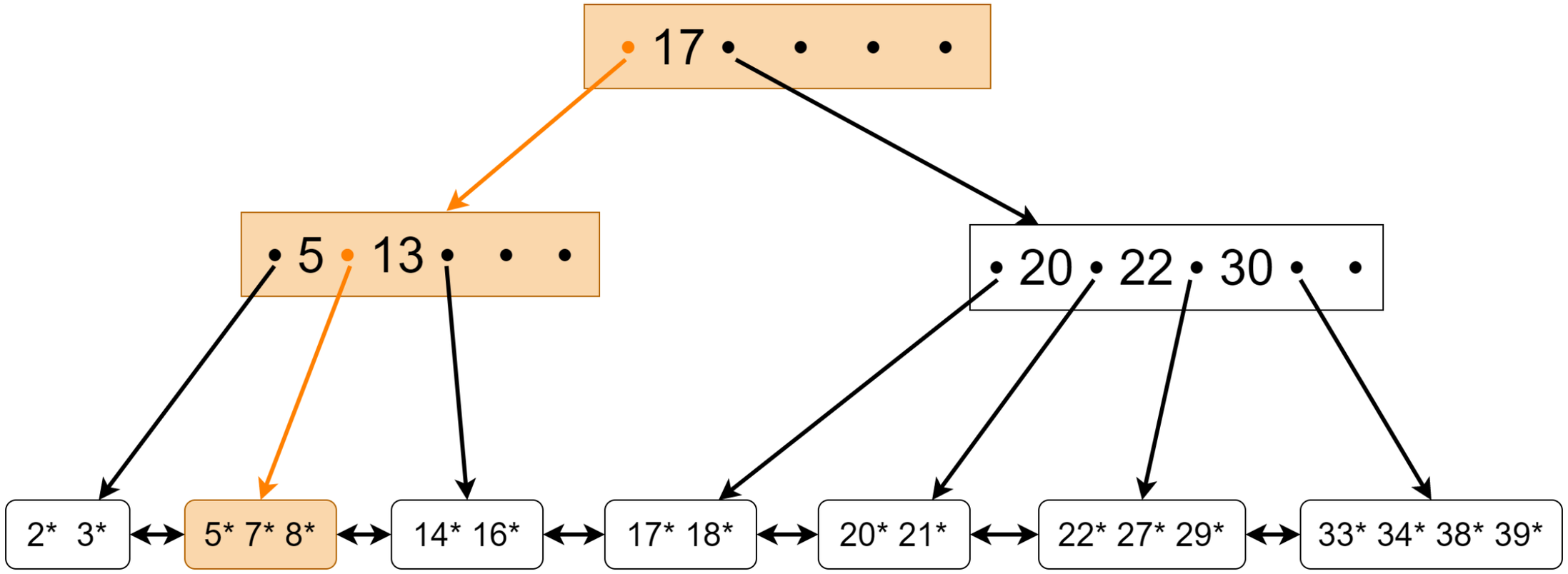
B+ Tree Index

Ejemplo: consulta de rango



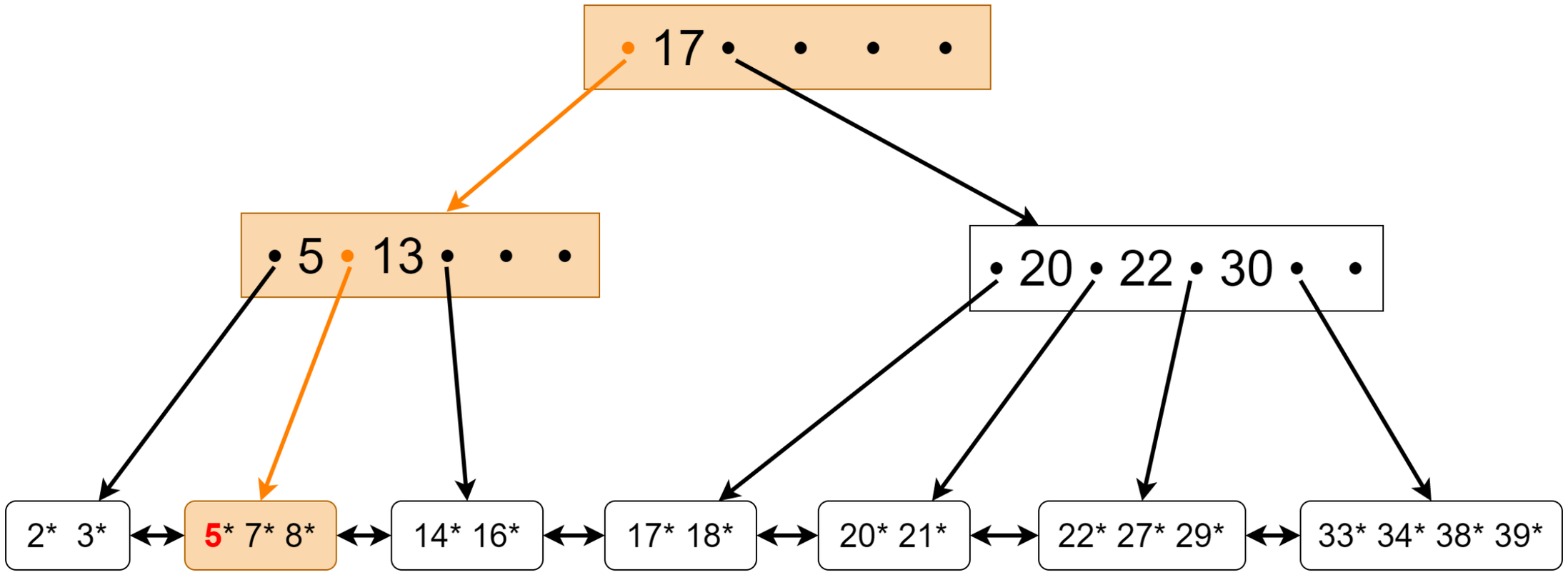
B+ Tree Index

Ejemplo: consulta de rango



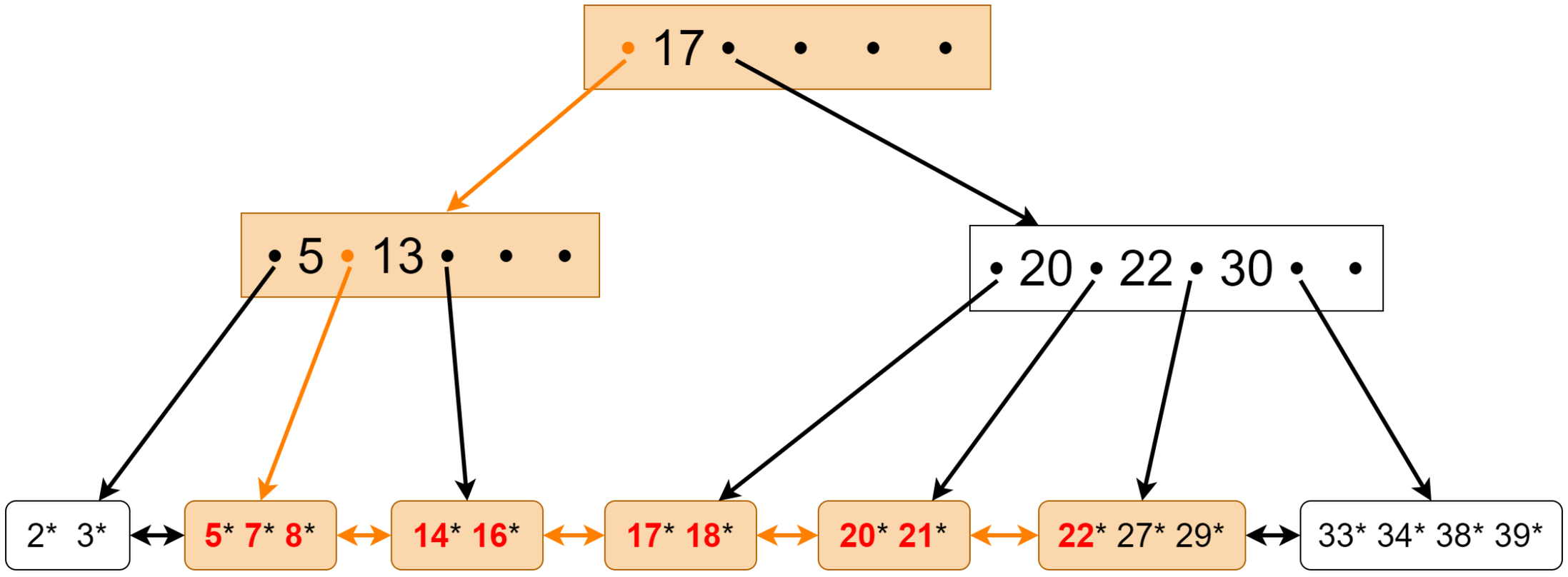
B+ Tree Index

Ejemplo: consulta de rango



B+ Tree Index

Ejemplo: consulta de rango



B+ Tree Index

Todos los nodos del B+Tree deben estar utilizados a más de la mitad de su capacidad, excepto la raíz, que puede tener menos elementos

B+ Tree Index

Al insertar, eliminar o actualizar valores debemos mantener el árbol balanceado, cumpliendo con mantener el orden del árbol

Esto último mantiene el tiempo de búsqueda logarítmico

B+ Tree Index

El costo de búsqueda es logarítmico -- log(|Tuplas|)

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

$$O(log_{\frac{B}{2}}(\frac{Tuplas}{\frac{B}{2}})) = O(log_{\frac{B}{2}}(\frac{2 \cdot Tuplas}{B}))$$

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

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

Clustered y Unclustered Index

Clustered Index se conoce como índice primario

Unclustered Index se conoce como índice secundario

Clustered y Unclustered Index

Considere **Empleados(<u>id</u>, 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(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)**
2	(2,3,3)
3	(3,1,2), (7,8,9)

Hash Index clustered

R(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)** página del disco
2	(2,3,3)
3	(3,1,2), (7,8,9)

Hash Index clustered

R(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)** página del disco
2	(2,3,3)
3	(3,1,2), (7,8,9)

#(I/O) = 1 (el bucket nos dice que página traer desde el disco)

Hash Index unclustered

R(a,b,c) … índice sobre a

a = 5

5 % 4 = 1

Bucket	
0	(4,p1), (0,p2), (8,p3)
1	(1,p1), **(5,p1)**
2	(2,p2)
3	(3,p2), (7,p3)

Datos en el disco:

p1	p2	p3
(4,1,1), (1,2,2), (5,1,1)	(0,2,3), (3,1,2), (2,3,3)	(7,8,9), (8,1,2)

Hash Index unclustered

R(a,b,c) … índice sobre a

a = 5

5 % 4 = 1

Bucket	
0	(4,p1), (0,p2), (8,p3)
1	(1,p1), **(5,p1)** página del disco
2	(2,p2)
3	(3,p2), (7,p3)

Datos en el disco:

p1	p2	p3
(4,1,1), (1,2,2), (5,1,1)	(0,2,3), (3,1,2), (2,3,3)	(7,8,9), (8,1,2)

Hash Index unclustered

R(a,b,c) … índice sobre a

a = 5

5 % 4 = 1

Bucket	
0	(4,p1), (0,p2), (8,p3)
1	(1,p1), **(5,p1)** página del disco
2	(2,p2)
3	(3,p2), (7,p3)

Datos en el disco:

p1

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

página del disco

p2

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

p3

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

Hash Index unclustered

R(a,b,c) … índice sobre a

a = 5

5 % 4 = 1

Bucket	
0	(4,p1), (0,p2), (8,p3)
1	(1,p1), **(5,p1)** página del disco
2	(2,p2)
3	(3,p2), (7,p3)

Datos en el disco:

p1

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

página del disco

p2

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

p3

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

#(I/O) = 2

Hash Index unclustered

R(a,b,c) … índice sobre b

b = 5

5 % 4 = 1

Bucket	
0	(4,p1), (0,p2), (8,p3)
1	(1,p1), **(5,p1), (5,p2)** página del disco
2	(2,p2)
3	(3,p2), (7,p3)

Datos en el disco:

p1

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

página del disco

p2

(0,2,3), (3,1,2), **(2,5,3)**

p3

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

#(I/O) = 1 (el bucket) + overflow pages + punteros

Hash Index unclustered

R(<u>a</u>,b,c) … índice sobre a

Considerar: página del índice vs página con tuplas

(1,p1), (2,p7),…

(1,2,3), (7,3,1),…

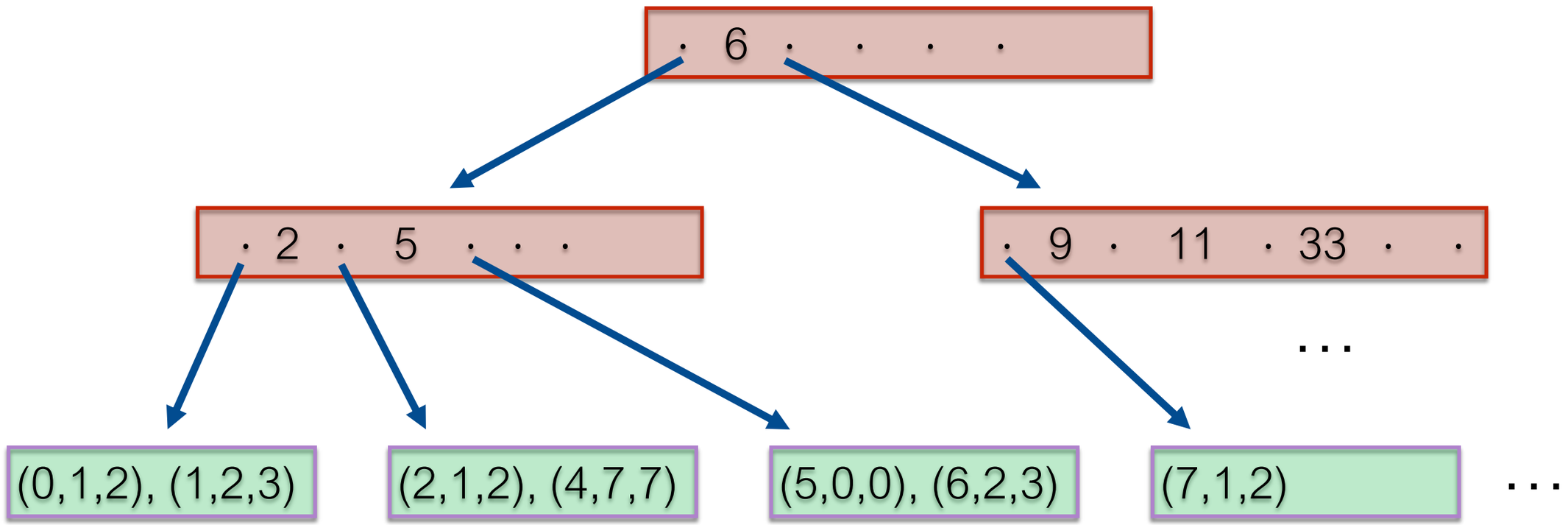
puntero p1 … un long/8 bytes

tupla … puede ser 20 longs/160 bytes

¿Dónde cabe más?

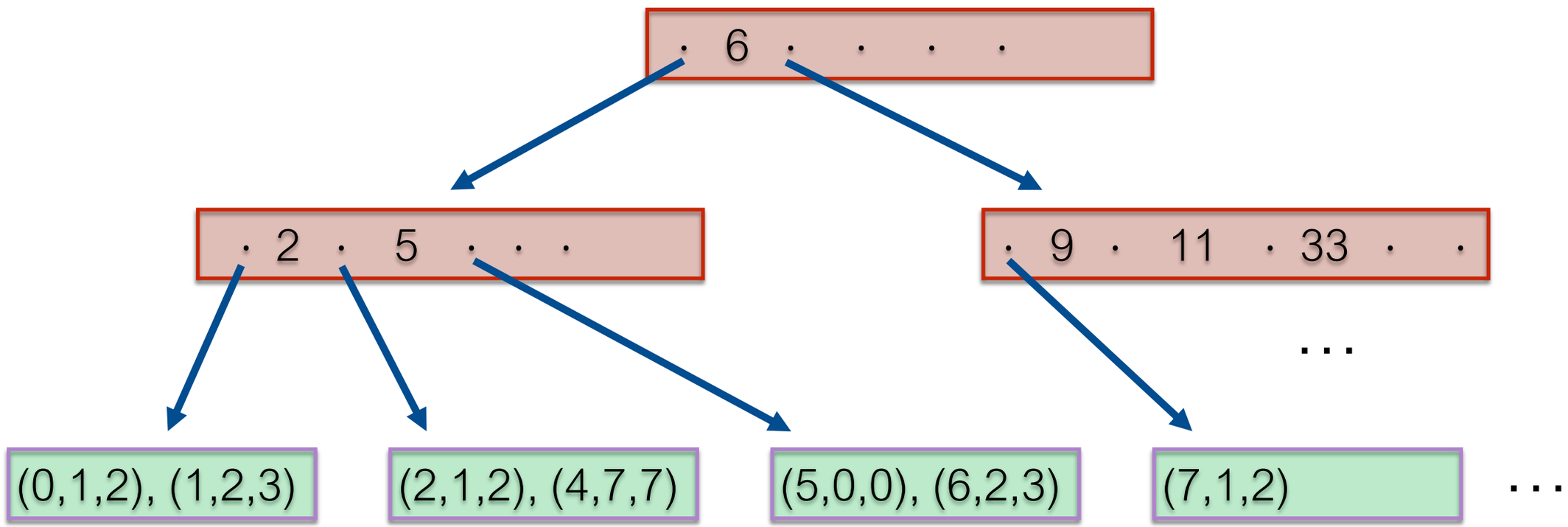
B+ tree clustered

R(<u>a</u>,b,c) … índice sobre a



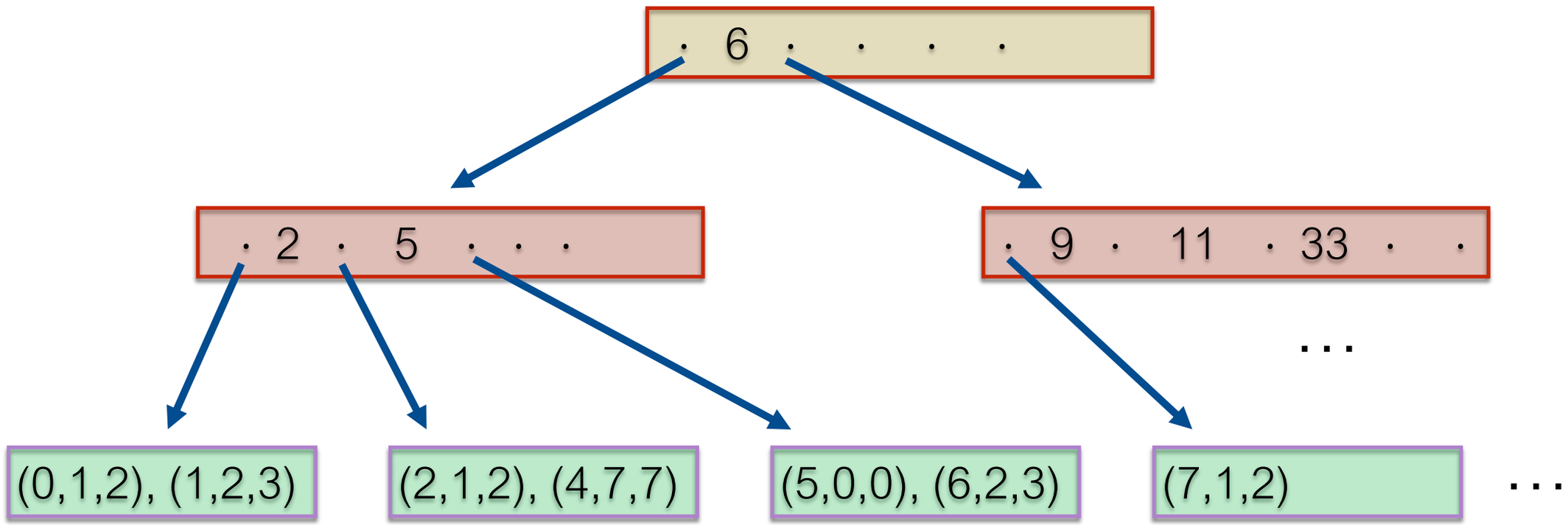
B+ tree clustered

R(a,b,c) … índice sobre a … a = 5



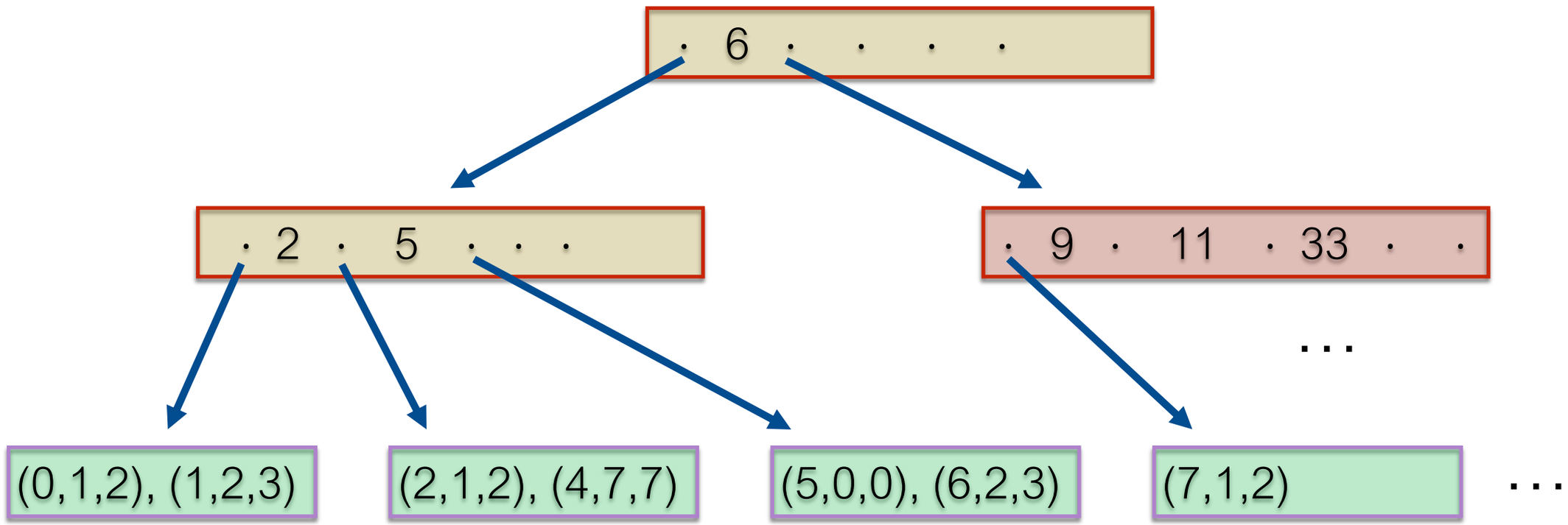
B+ tree clustered

R(a,b,c) … índice sobre a … a = 5



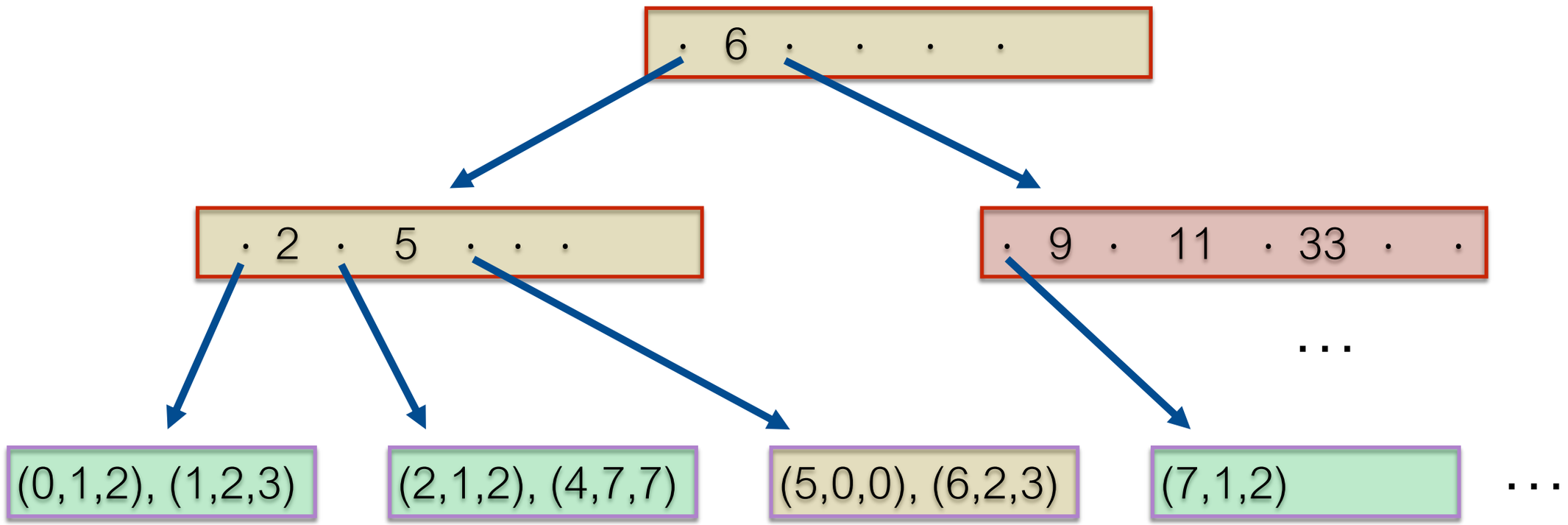
B+ tree clustered

R(a,b,c) … índice sobre a … a = 5



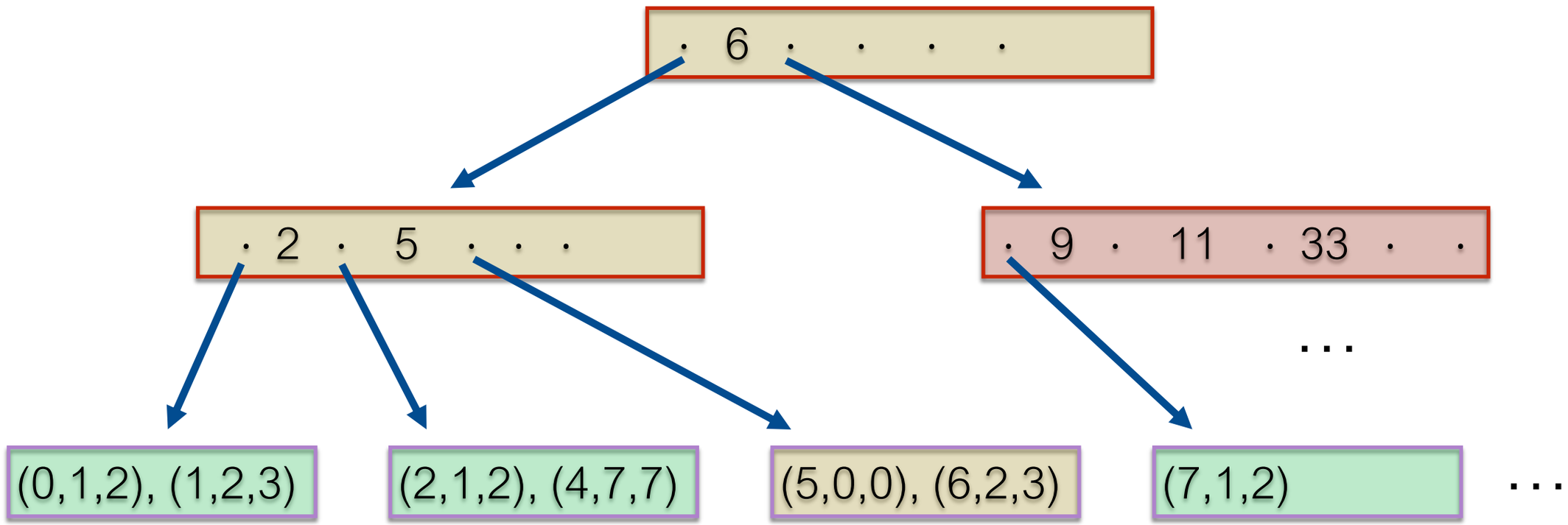
B+ tree clustered

R(<u>a</u>,b,c) … índice sobre a … a = 5



B+ tree clustered

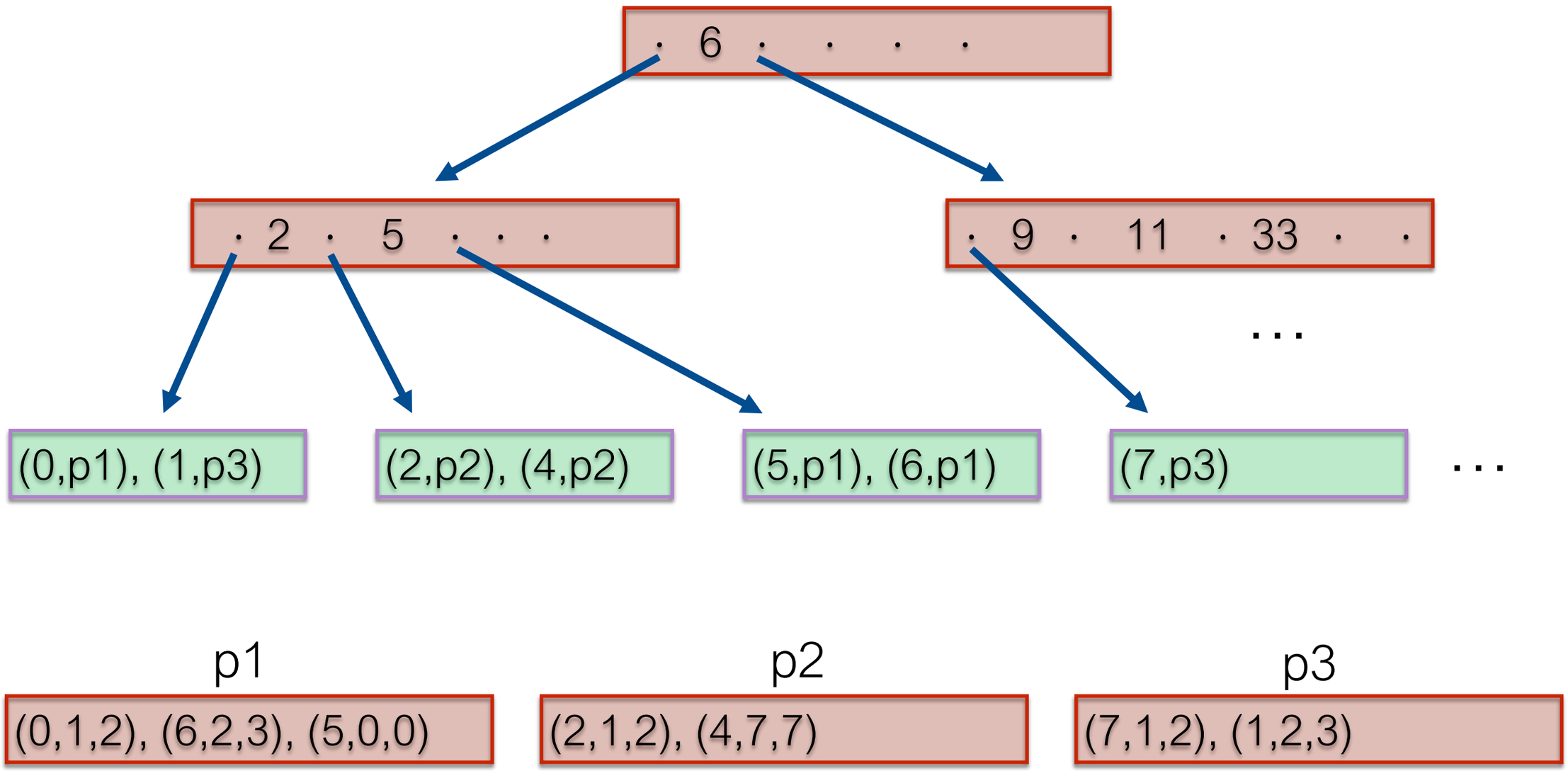
R(a,b,c) … índice sobre a … a = 5



#(I/O) = h … altura del árbol

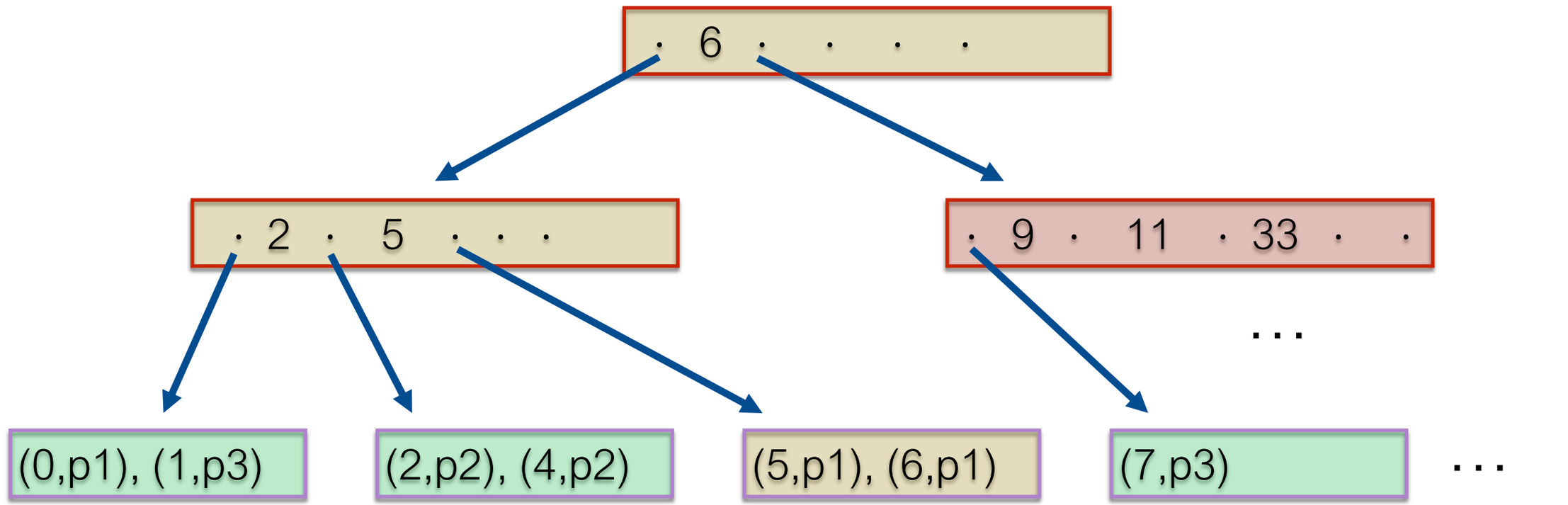
B+ tree unclustered

R(<u>a</u>,b,c) … índice sobre a



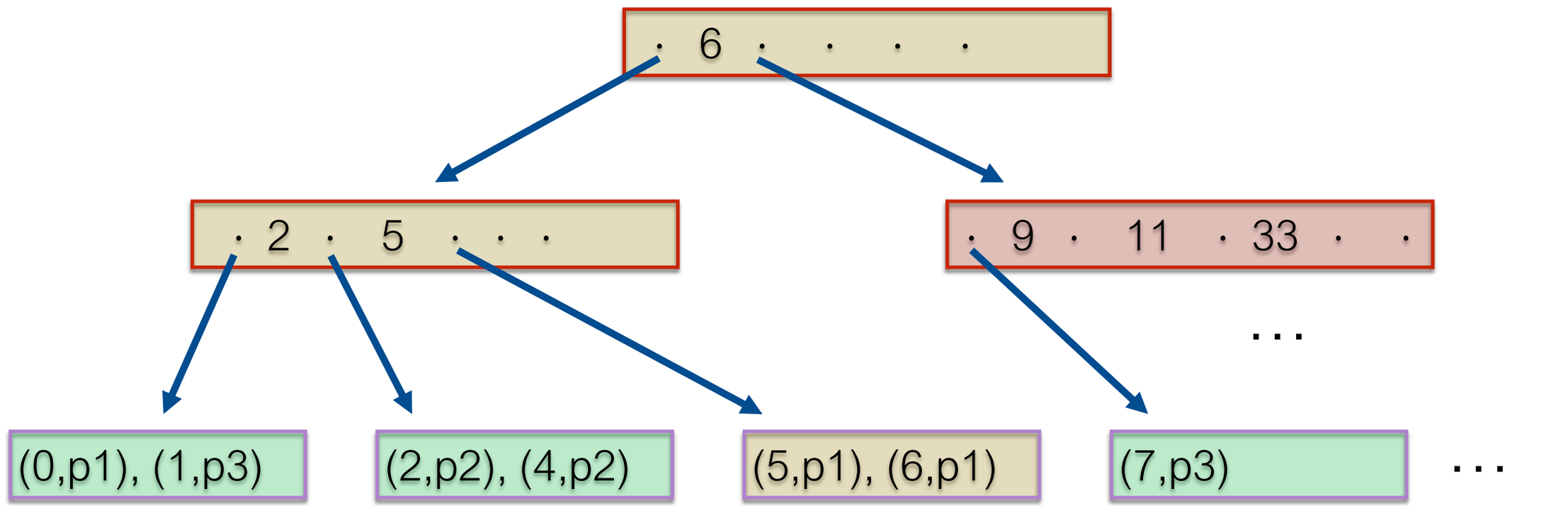
B+ tree unclustered

R(a,b,c) … índice sobre a … a = 5



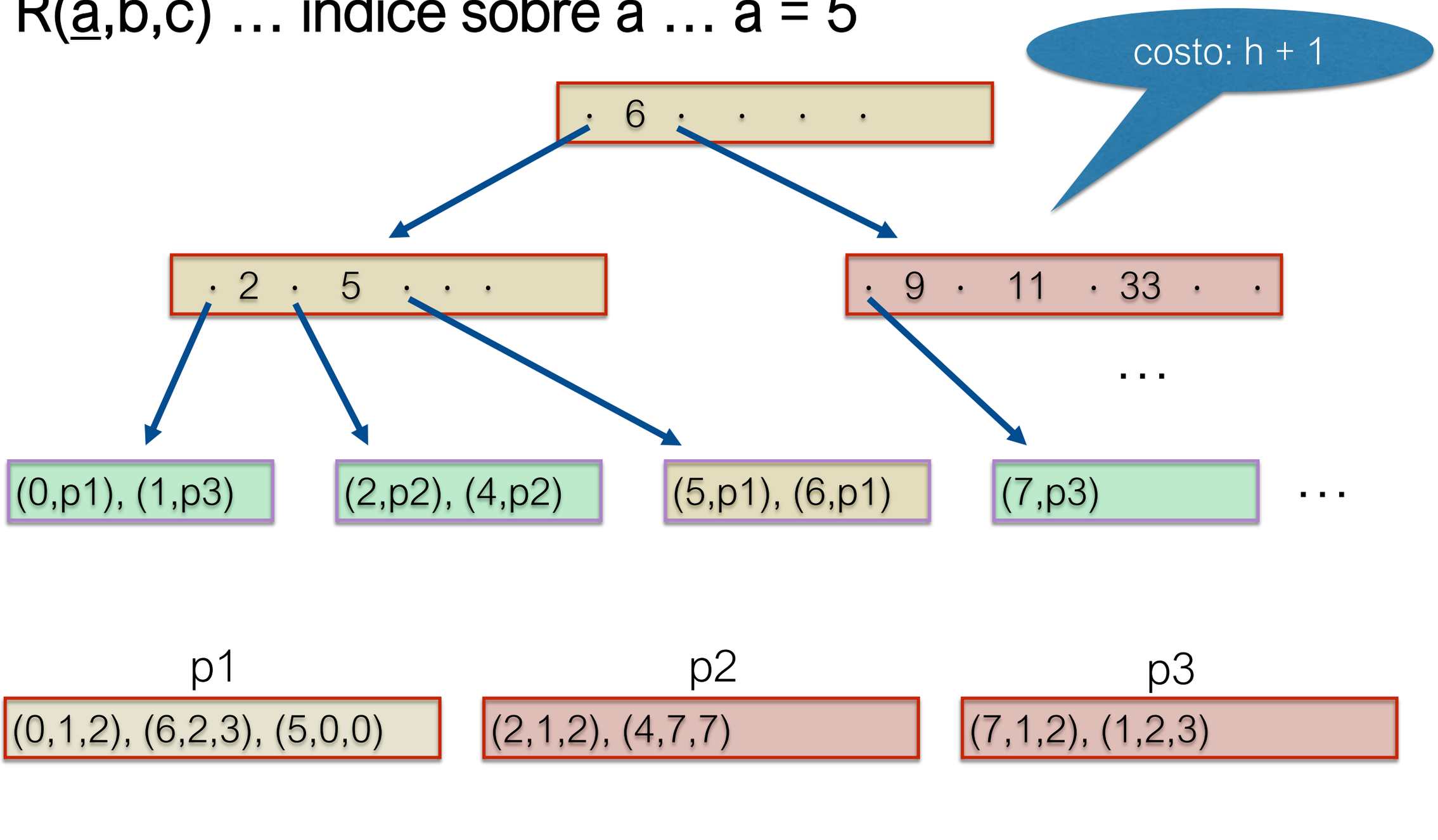
B+ tree unclustered

R(a,b,c) … índice sobre a … a = 5



B+ tree unclustered

R(a,b,c) … índice sobre a … a = 5



B+ tree unclustered

R(a,b,c):

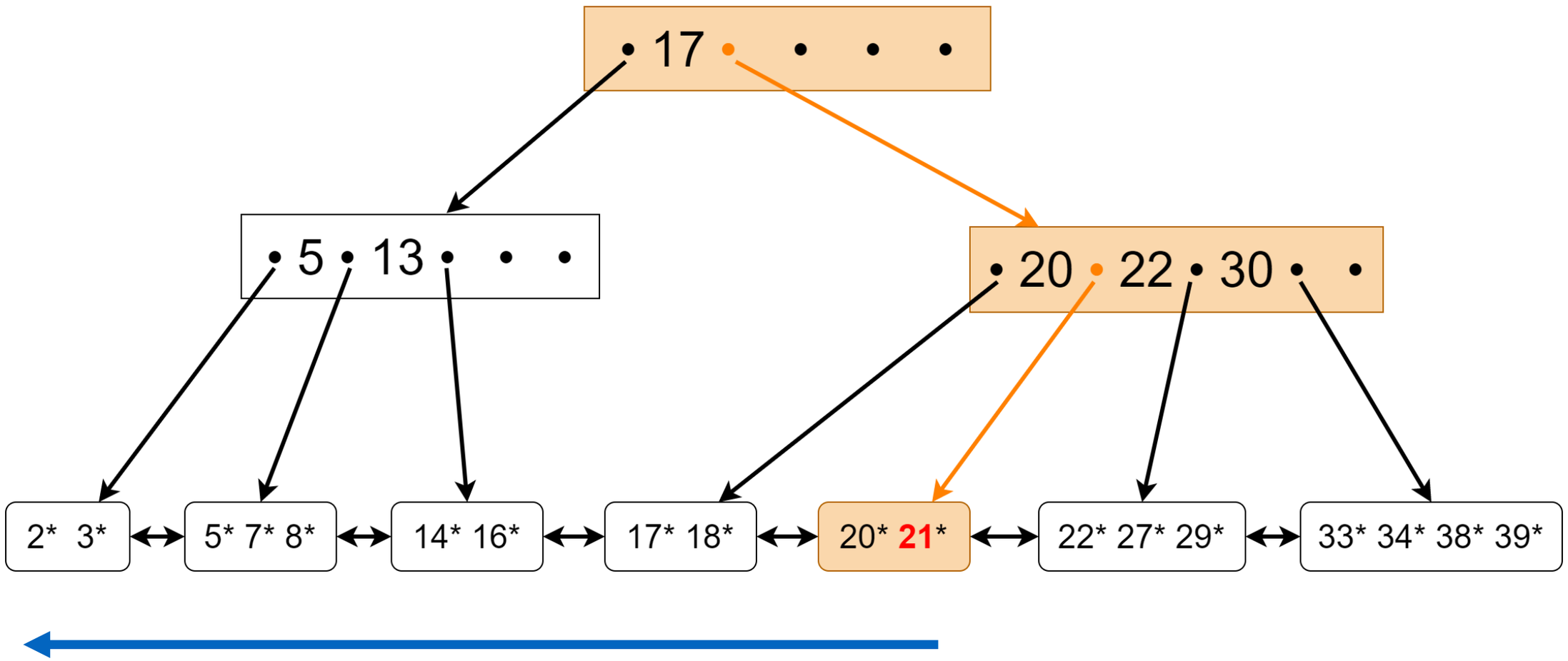
- 1M tuplas
- valor de a distribuido uniformamente en [0,2.000.000]
- P tuplas por página
- M > P punteros en una página
- h altura del B+ tree

SELECT * FROM R WHERE a<21

Costo:

- Buscar a = 21 … h

B+ tree unclustered



B+ tree unclustered

R(<u>a</u>,b,c):

- 1M tuplas
- valor de <u>a</u> distribuido uniformamente en [0,2.000.000]
- P tuplas por página
- M > P punteros en una página
- h altura del B+ tree

Costo:

- Buscar a = 21 → h
- Por uniformidad hay 0.5 tuplas por distinto a → 11 tuplas
- Cúantos punteros hay en una página? →11/M
- Más buscar la página en el disco → 11

B+ tree unclustered

R(a,b,c):

- 1M tuplas
- valor de a distribuido uniformamente en [0,2.000.000]
- P tuplas por página
- M > P punteros en una página
- h altura del B+ tree

Costo:

- Buscar a = 21 → h
- Por uniformidad hay 2 tuplas por distinto a → 11 tuplas
- Cúantos punteros hay en una página? →11/M
- Más buscar la página en el disco → 11

$$h + (11/M - 1) + 11$$

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