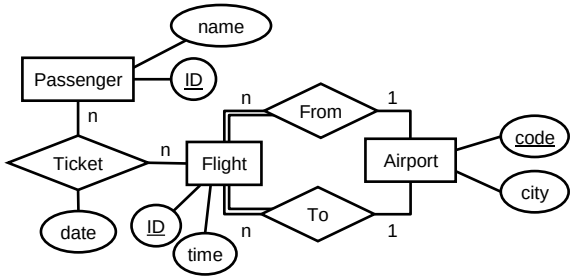
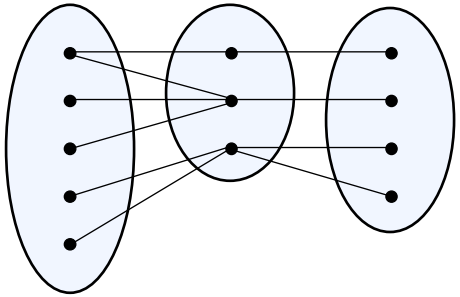
Temario

- Introducción y fundamentos
- Introducción a SQL
- Modelo Entidad / Relación
- Modelo relacional
- Diseño relacional: formas normales
- Consultas
  - Cálculo relacional
  - Álgebra relacional
- Implementación de bases de datos — Folk, cap. 1, 3, 4, 5, 7, 8, 9, 10
  - Estructura física: campos y registros
  - Indexación
    - Índices simples
    - Árboles B, B* y B+



Modelo ER



Flight

number	origin	destination	time
345	MAD	CDG	12:30
321	MAD	ORY	19:05
165	LHR	CDG	09:55
903	CDG	LHR	14:40
447	CDG	LHR	17:00

Passenger

id	name
123	Mary
456	Peter
789	Isabel
321	Peter

Ticket

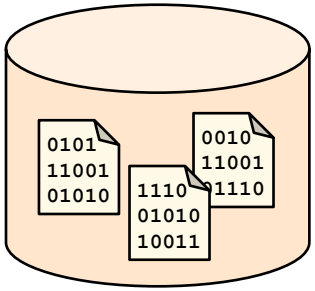
id	number	date	price
789	165	07-01-11	210
123	345	20-12-10	170
789	321	15-12-10	250
456	345	03-11-10	190

SQL

Modelo Relacional

Conjunto de registros

- Claves
- Formas Normales
- Calculo & algebra



Implementación a bajo nivel (estructura de ficheros)

- Estructura del disco y coste
- Registros, campos, bytes
- Índices & árboles B

Implementación de un SGBD

Implementación de un SGBD

Usuario final

Interfaz de usuario

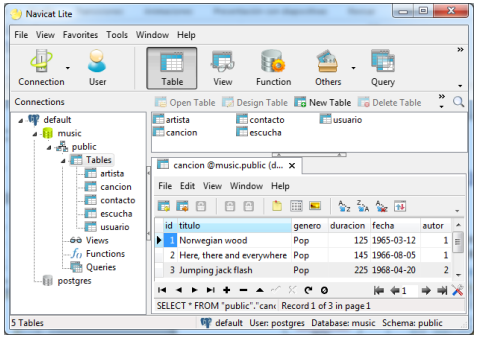
Programador aplicación

Software aplicación

Administrador

Lógica de la aplicación

Entorno / herramientas de administración



pgAdmin dbeaver vs code etc.

Sentencias SQL

ODBC, JDBC, PHP…

API BD

Almacenamiento
Consultas
Actualización

SGBD

Bases de datos

Implementación de un SGBD

- Después de SQL, modelo E/R, modelo relacional…
  - Modelo físico de datos: la misma información en **estructuras de archivos** (bytes)

    Relaciones y atributos → "bytes" en disco
- Implementación de las operaciones relacionales sobre datos almacenados en disco
  - Consultas
  - Actualización: inserción, modificación, eliminación
- Optimizaciones para hacer viables las operaciones de consulta y actualización
  - Análisis del coste de las operaciones en disco
  - Estrategias de borrado/inserción/compactación
  - **Indexación**: índices simples, árboles B

Implementación de un SGBD (cont)

- Un SGBD debe hacerse cargo de…
  - Almacenar en memoria externa los datos del estado de un esquema de BD
  - Definir una organización física de estos datos (en bytes de archivos en disco)
  - Ejecutar sentencias SQL: convertirlas a operaciones elementales de acceso y actualización de datos en archivos
  - Optimizar el acceso a datos en disco (indexación)
- En la asignatura vamos a analizar y estudiar algunas técnicas y estrategias fundamentales de tipo general para resolver estas funcionalidades
  - La implementación de un SGBD completo incluye una gran parte de técnicas de desarrollo no estandarizadas, resueltas de manera específica y muy variada en los SGBD actuales, que no se prestan tanto a la generalización y estudio sistemático

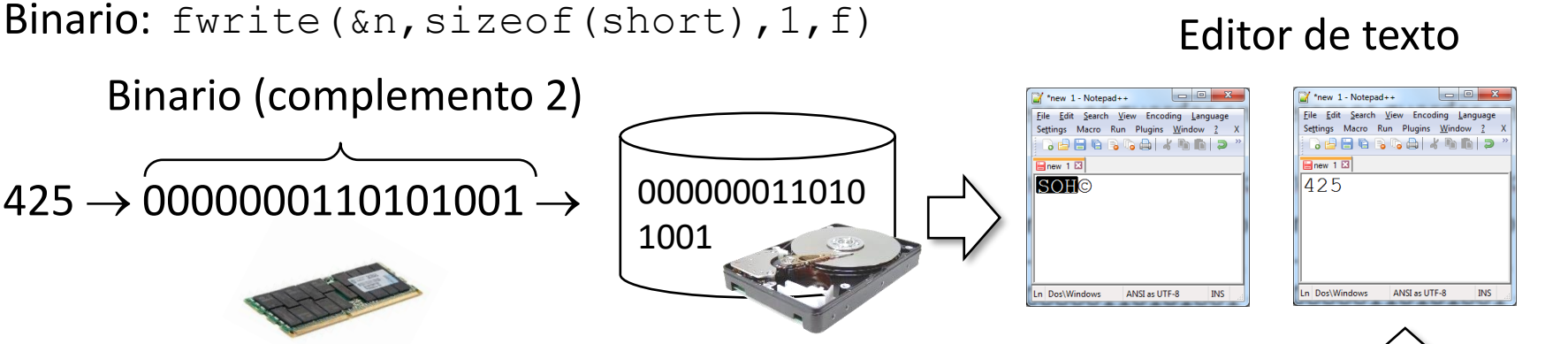
Temario

- Introducción y fundamentos
- Introducción a SQL
- Modelo Entidad / Relación
- Modelo relacional
- Diseño relacional: formas normales
- Consultas
  - Cálculo relacional
  - Álgebra relacional
- Implementación de bases de datos
  - Estructura física: campos y registros — Folk, cap. 1, 3, 4, 5
  - Indexación
    - Índices simples
    - Árboles B, B* y B+

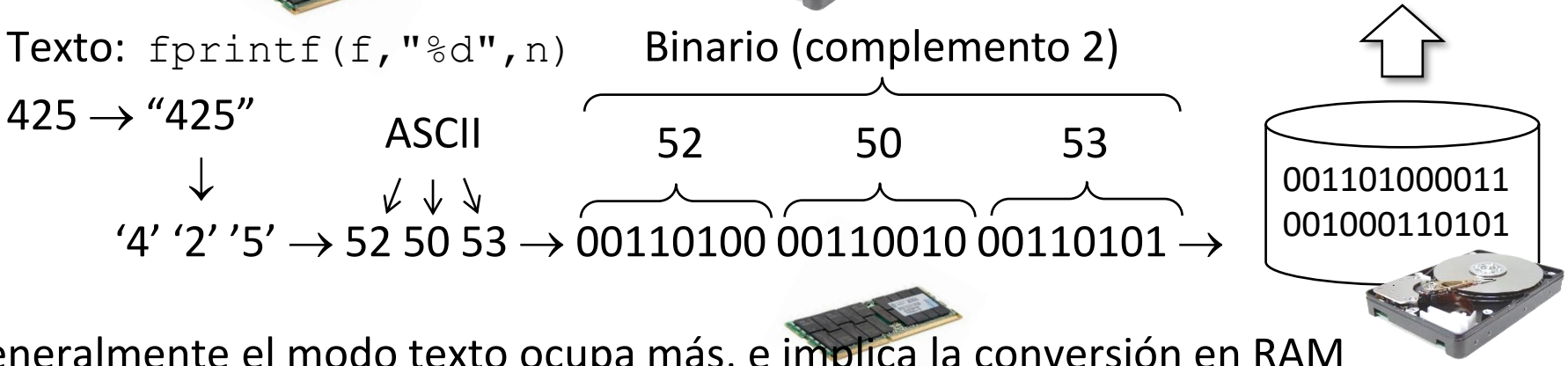
Escritura/lectura en binario vs. en texto

Ejemplo: queremos guardar en disco el número `short n = 425`

- Binario: `fwrite(&n,sizeof(short),1,f)`



- Texto: `fprintf(f,"%d",n)`



Generalmente el modo texto ocupa más, e implica la conversión en RAM de binario a texto (tipo "itoa") – a cambio es legible con un editor de texto

La lectura (como es lógico) debe ser coherente con la escritura: binario ↔ binario, texto ↔ texto (es decir `fwrite` ↔ `fread`, `fprintf` ↔ `fscanf`)

Para cadenas de caracteres (texto), modo binario y texto es lo mismo

En binario no tienen sentido los saltos de línea

Escritura/lectura en binario vs. en texto

Escritura de 100.000.000 de long

- En ASCII (escribir_txt.c):
  - Tiempo de ejecución: 13.03 seg
  - Tamaño del fichero: 942MB
- En binario (escribir_bin.c):
  - Tiempo de ejecución: 2.86 seg (**4.5 veces menos**)
  - Tamaño del fichero: 965MB (381MB = 100.000.000 * 4 bytes, **2.5 veces menos**)

lectura de 100.000.000 de long

- En ASCII (leer_txt.c):
  - Tiempo de ejecución: 11.4 seg
- En binario (leer_bin.c):
  - Tiempo de ejecución: 2.8 seg (**4.1 veces menos**)

Acceso al último elemento

Tiempo de lectura del ultimo entero del fichero

- En ASCII (acceso_txt.c):
  - Lectura secuencial
  - Tiempo de acceso y lectura (leer todo el fichero): 12.8 seg.

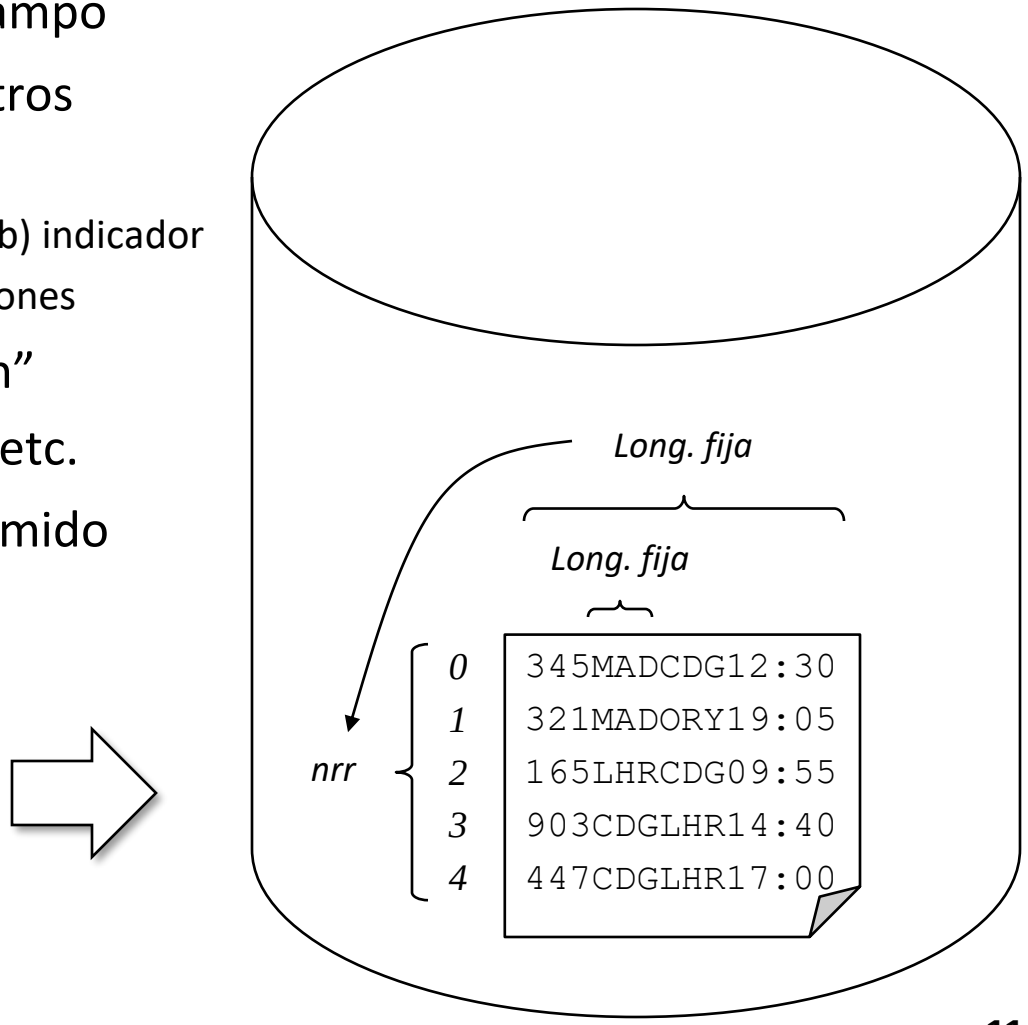
- En binario (acceso_bin.c):
  - Lectura aleatoria
  - Tiempo de acceso y lectura: 0.0seg

Almacenamiento de tablas: Campos y registros

- Tupla → registro, atributo → campo
- Delimitación de campos y registros
  - Longitud fija
  - Longitud variable: a) separador, b) indicador de longitud, c) fichero de direcciones
- Cabeceras con "autodescripción" de estructuras, nº de registros, etc.
- Formato: texto, binario, comprimido

VUELO

Numero	Origen	Destino	Salida
345	MAD	CDG	12:30
321	MAD	ORY	19:05
165	LHR	CDG	09:55
903	CDG	LHR	14:40
447	CDG	LHR	17:00

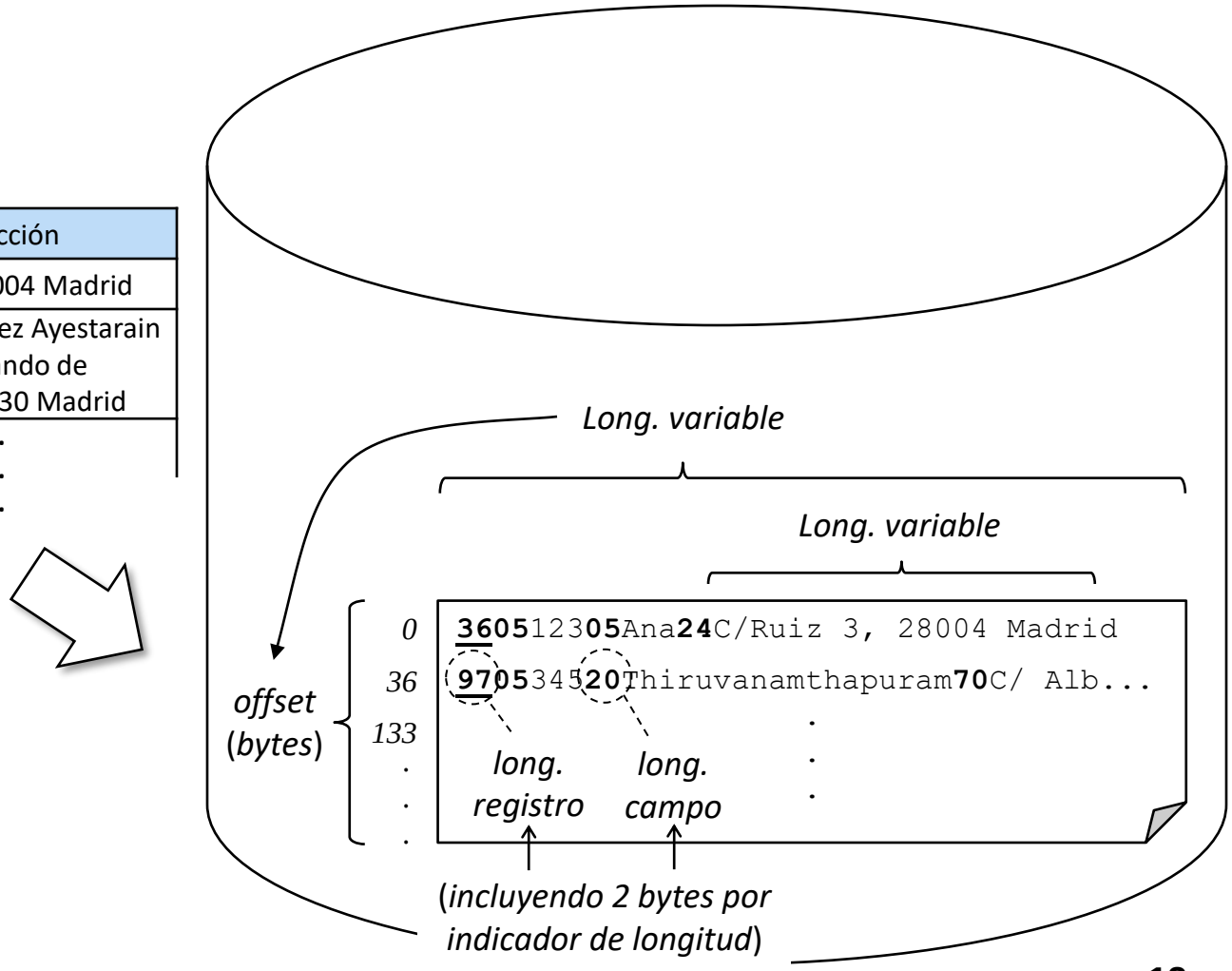


nrr: número relativo de registro

Almacenamiento de tablas: Campos y registros

EMPLEADO

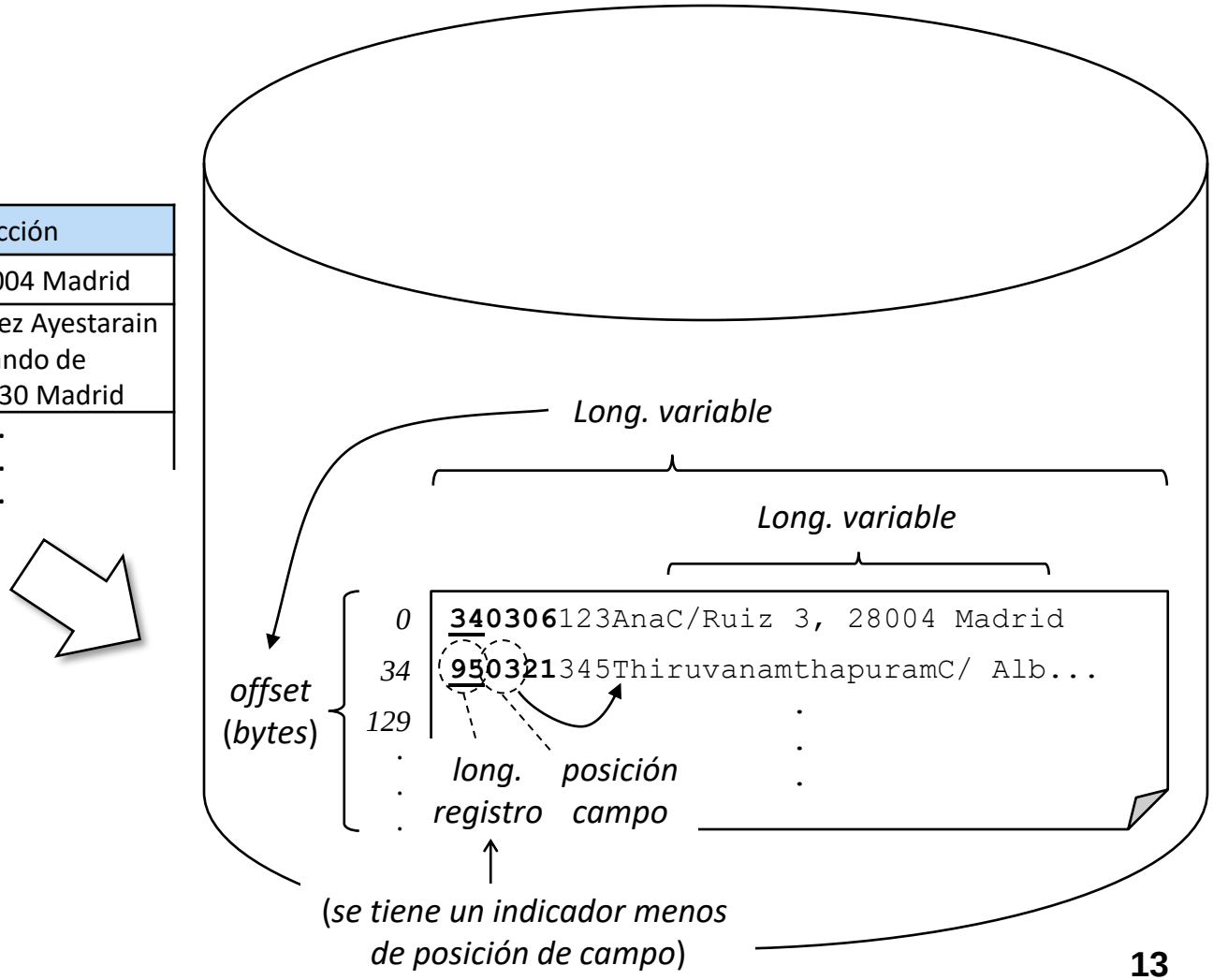
DNI	Nombre	Dirección
123	Ana	C/ Ruiz 3, 28004 Madrid
345	Thiruvanamthapuram	C/ Albino Pérez Ayestarain 14, San Fernando de Henares, 28830 Madrid
.	.	.
.	.	.
.	.	.



Almacenamiento de tablas: Campos y registros

EMPLEADO

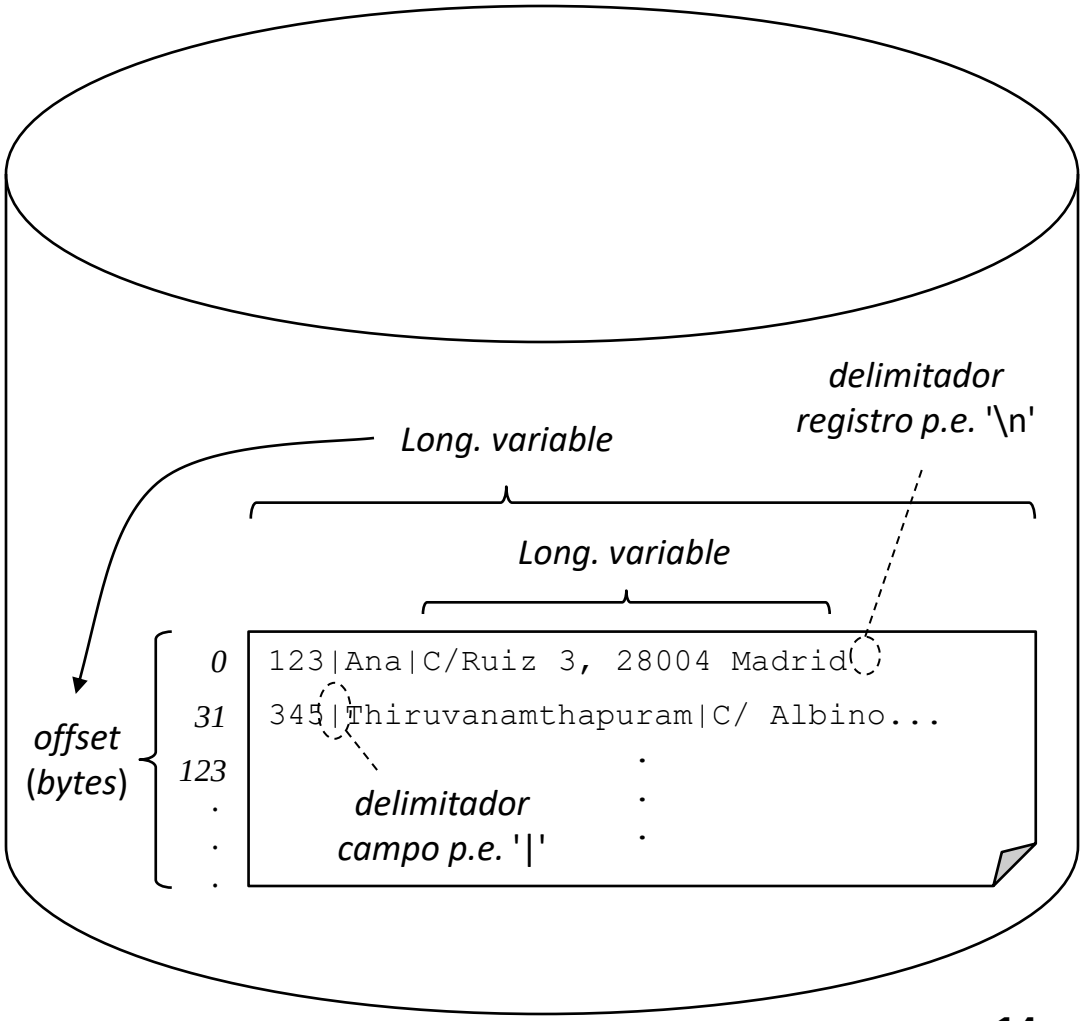
DNI	Nombre	Dirección
123	Ana	C/ Ruiz 3, 28004 Madrid
345	Thiruvanamthapuram	C/ Albino Pérez Ayestarain 14, San Fernando de Henares, 28830 Madrid
.	.	.
.	.	.
.	.	.



Almacenamiento de tablas: Campos y registros

EMPLEADO

DNI	Nombre	Dirección
123	Ana	C/ Ruiz 3, 28004 Madrid
345	Thiruvanamthapuram	C/ Albino Pérez Ayestarain 14, San Fernando de Henares, 28830 Madrid
.	.	.
.	.	.
.	.	.



Delimitación de registros y campos

- Longitud fija
  - Acceso a datos más rápido
  - Los registros se direccionan por número relativo de registro (nrr)
- Longitud variable
  - Acceso más lento
  - Optimización de espacio en disco
  - Los registros se direccionan por su posición en bytes (offset)
- Cabecera
  - Sección descriptiva de la estructura física de un archivo: qué campos, cómo se delimita, etc.
  - Para permitir implementaciones más generales que admitan diferentes formas de organizar físicamente los registros

¿Es mejor longitud fija o variable?

- Longitud fija: economía de procesamiento (¡y esfuerzo de programación!)
  - Solución cómoda de desarrollar y ejecutar
- Longitud variable: economía de espacio en disco
  - El esfuerzo de desarrollo / ejecución puede compensar, por ejemplo, cuando…
    - El consumo de disco es un problema
    - La variabilidad de tamaño de registros es muy amplia: campos muy variables y/u opcionales (valores NULL)
    - Se juntan registros de diferentes tipos en el mismo archivo
- Si un campo es realmente grande (documento de texto o multimedia) se suele almacenar en un archivo aparte y se guarda en el registro un descriptor de la dirección donde se almacena el valor del campo

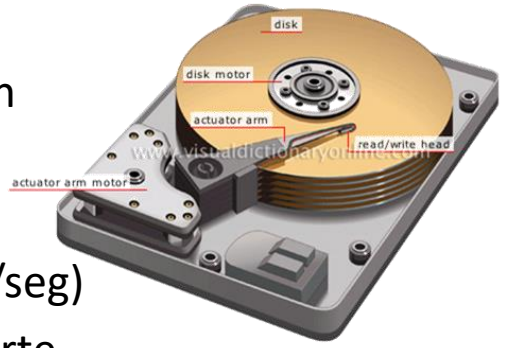
Operaciones en disco

- Acceso
  - Acceso a un registro por posición
  - Búsqueda de registros por condiciones (consultas)
  - Índices para optimizar las operaciones de acceso
- Actualización
  - Inserción
  - Modificación, eliminación: típicamente conllevan una consulta
  - Estrategias de borrado eficiente combinado con inserción
- Coste de acceso a datos en disco
  - Es un aspecto clave y condiciona todas las técnicas
  - El acceso al disco es varios órdenes de magnitud mayor al acceso en RAM
  - Formulaciones sencillas de operaciones comunes resultan inviables
  - Cambia completamente el análisis de complejidad
  - Cambian las estrategias y algoritmos con respecto a las técnicas en RAM
  - HD ~ 0.1GB/s, SSD ~ (SATA 0.4GB/s, M.2 3GB/s, PCIe 8-10 GB/s), RAM ~ (DDR4 25GB/s, DDR5 50GB/s)

Dispositivos de almacenamiento

- Determinan el coste de las operaciones de acceso a los datos
- Memoria principal (RAM)
  - Máxima velocidad de acceso (~ 50GB/seg)
  - Alto coste económico
  - No persistente

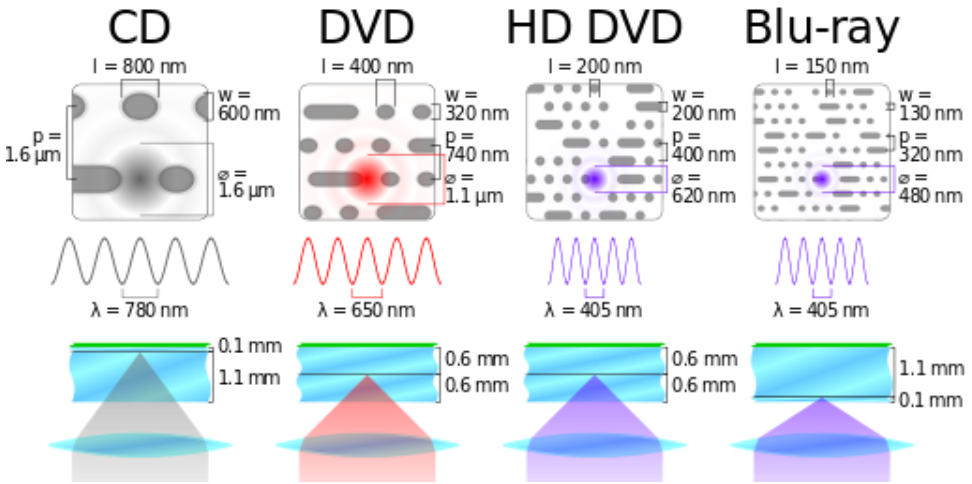
- Disco magnético
  - Sigue siendo el dispositivo de almacenamiento más común para bases de datos
  - Órdenes de magnitud más lento que la memoria RAM, debido a la necesidad de movimiento mecánico (~ 0,1 GB/seg)
  - Basaremos nuestro análisis de costes en este tipo de soporte



- Memoria flash / SSD
  - "Solid state" alude al contraste con piezas móviles
  - Acceso más rápido que el hdd (~ 0.4 GB/seg)
  - Nº limitado de reescrituras (p.e. varios miles)
  - Ha ganado mercado en los últimos años

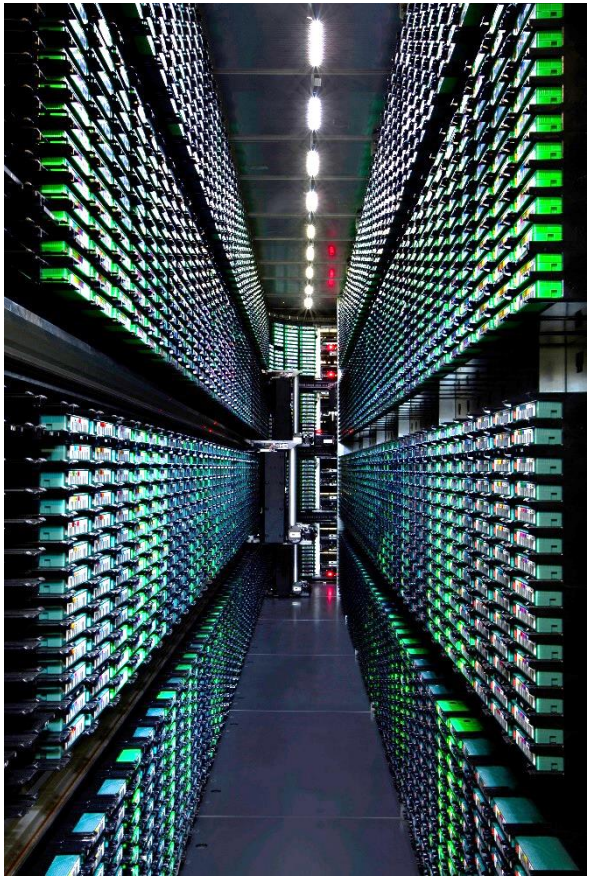
Dispositivos de almacenamiento (cont)

- Almacenamiento óptico
  - Mecanismo comparable al disco magnético pero basado en reflejo láser, y los datos se organizan en espiral en lugar de pistas concéntricas
  - Su robustez lo ha hecho muy popular, especialmente a nivel doméstico
  - Pero su uso tiende a declinar en favor de otros tipos de soporte y canales de distribución de datos y contenidos
  - No es adecuado en general para BDs en términos de capacidad ni velocidad; el tiempo de escritura es inviable y la estructura secuencial no es apropiada para la actualización dinámica de datos



Dispositivos de almacenamiento (cont)

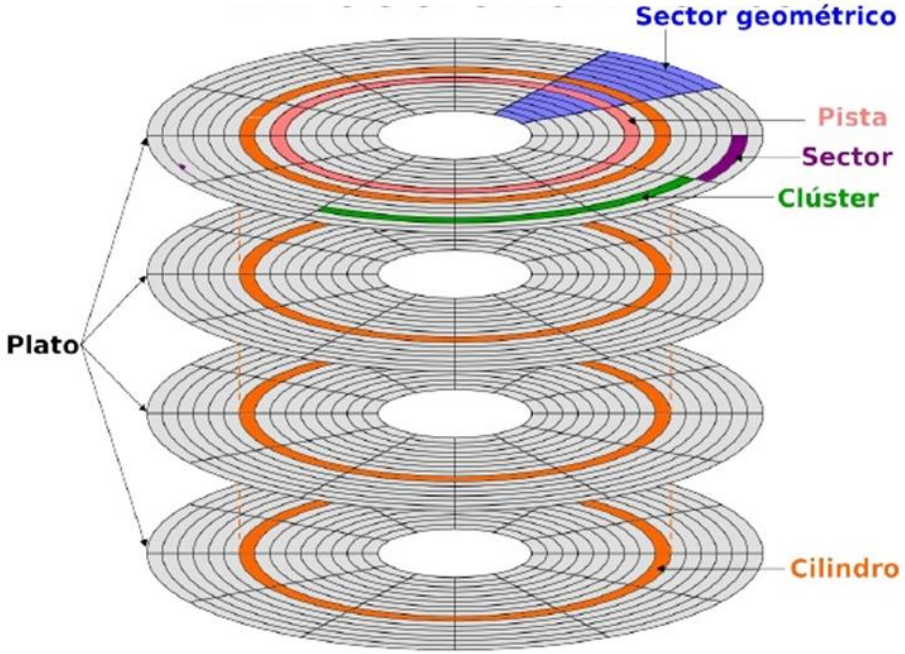
- Cinta magnética
  - Mucho más lenta que otros soportes, acceso secuencial
  - Facilidades de manipulación y almacenaje, aunque también se está avanzando con el uso de discos magnéticos en esta faceta
  - Sigue siendo masivamente utilizado como soporte de backup para “disaster recovery” (p.e. Google es actualmente el mayor consumidor mundial de cinta magnética)
  - Pero no tiene sentido como soporte dinámico para BDs en producción
- …?



Biblioteca de backup en cinta, Google data center Berkeley

Costes de acceso en disco magnético

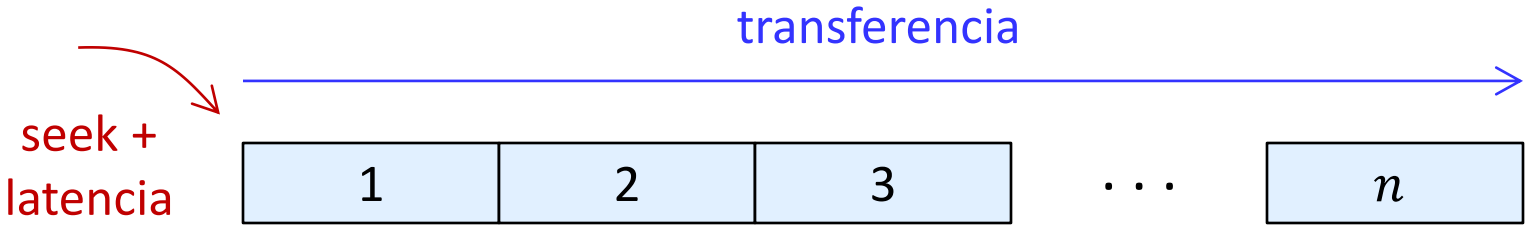
- La algoritmia actual de implementación de BDs se basa fundamentalmente en este tipo de dispositivo, con disponibilidad de RAM media/variable
  - Disponibilidad ilimitada de RAM, o uso de otros dispositivos $\rightarrow$ revisar la algoritmia
- Estructura de una unidad de disco magnético
  - Discos, superficies
  - Sectores (angulares) $\subset$ bloques / clusters $\subset$ pistas (concéntricas) $\subset$ cilindros (pistas en la misma vertical) $\subset$ superficie de discos
  - Sectores (inherentes al hardware), clusters y bloques (configurables en el S.O.) son la unidad mínima de lectura/escritura, y de fragmentación de archivos


Sector geométrico
Pista
Sector
Clúster
Plato
Cilindro

Costes de acceso en disco magnético

- Operaciones elementales de lectura/escritura y su coste
  - Seek: movimiento radial (caso promedio: desplazamiento de 1/3 del radio)
  - Latencia rotacional (caso promedio: 1/2 rotación)
  - Transferencia: (tiempo rotación × nº de bloques a leer / nº de bloques por pista)
    (tiempo rotación × nº bytes por transferir/ nº bytes por pista)
- Lectura (o escritura) de un archivo
  - Posicionarse en el primer sector: seek + latencia
  - Recorrer los bytes: tiempo de transferencia de los bloques que ocupa el archivo
  - Saltos adicionales por fragmentación: seek + latencia por cada bloque no contiguo que contenga el archivo
  - Unidad de lectura por vuelta es el bloque o cluster
- Estrategias de optimización
  - Evitar leer datos innecesarios
  - Evitar la repetición de lecturas
  - Reducir el nº de seek + latencia, por ejemplo, utilizando buffers
  - Y muchas otras estrategias en el diseño de algoritmia

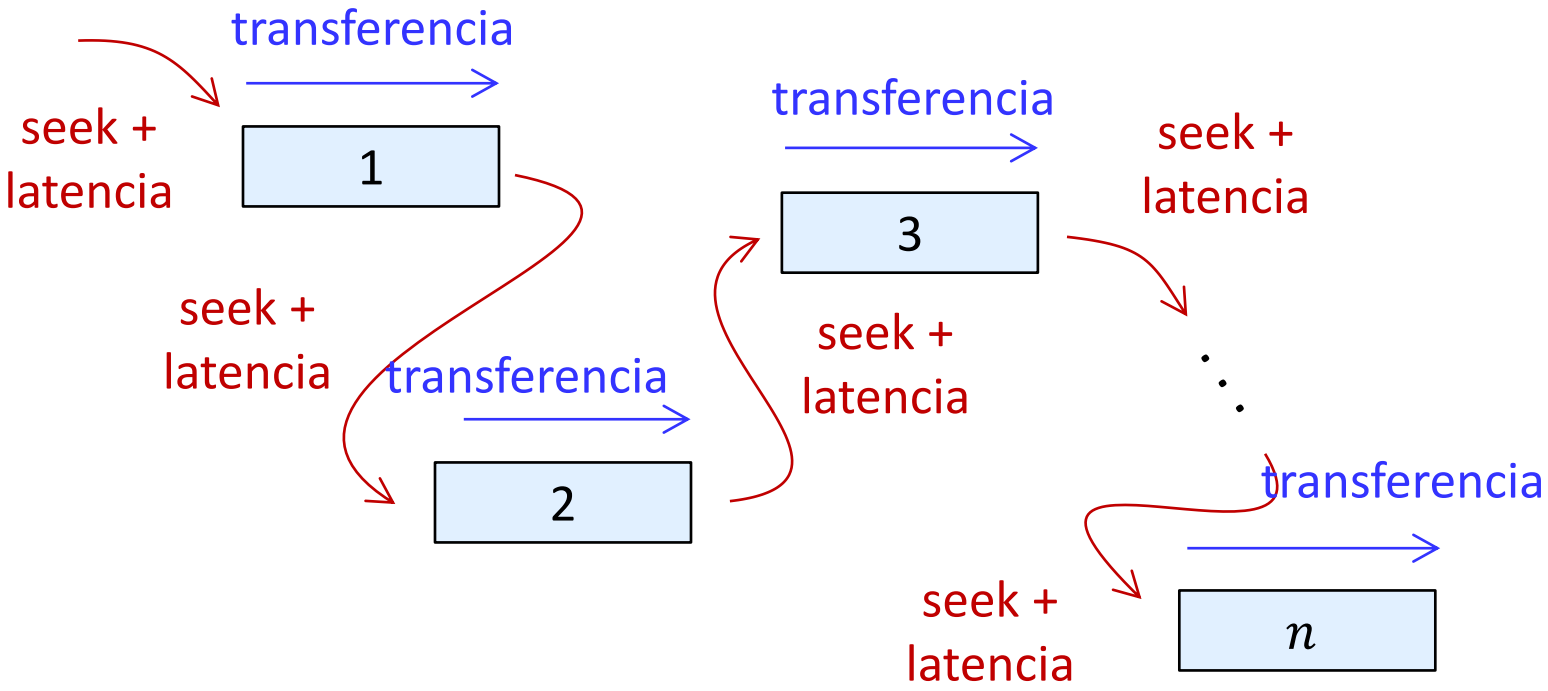
Mejor escenario: datos a leer (escribir) están en bloques contiguos (no hay fragmentación)



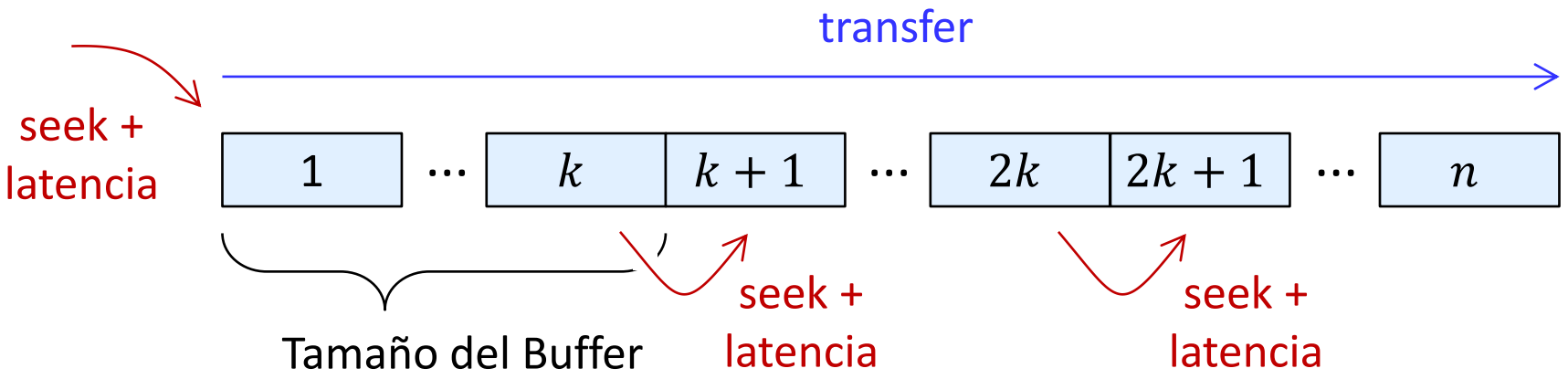
1 x (seek + latencia) + transferencia

Peor escenario: total fragmentación

n x (seek + latencia) + transferencia



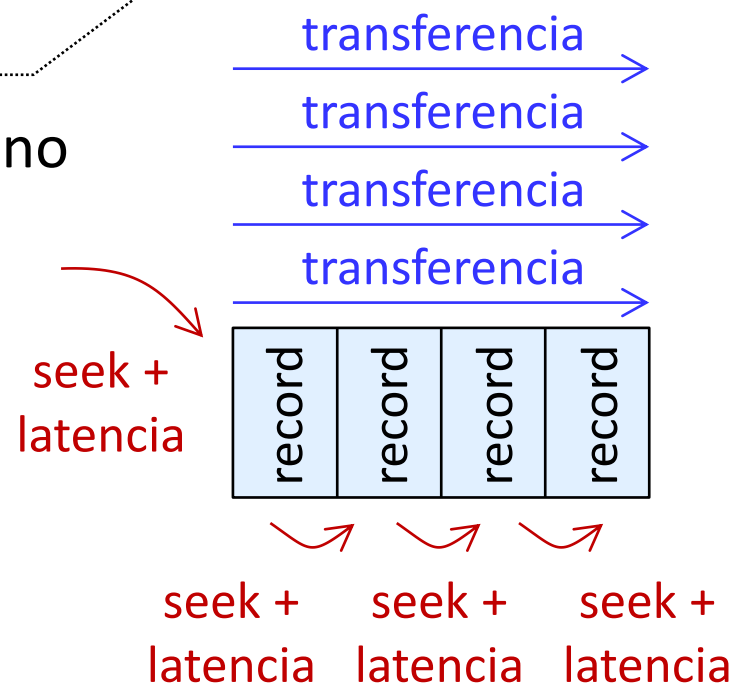
Caso intermedio, bloques contiguos, pero solo k caben en el buffer



$\lceil n/k \rceil$ x (seek + latencia) + transferencia

La peor programación: leer registros uno a uno

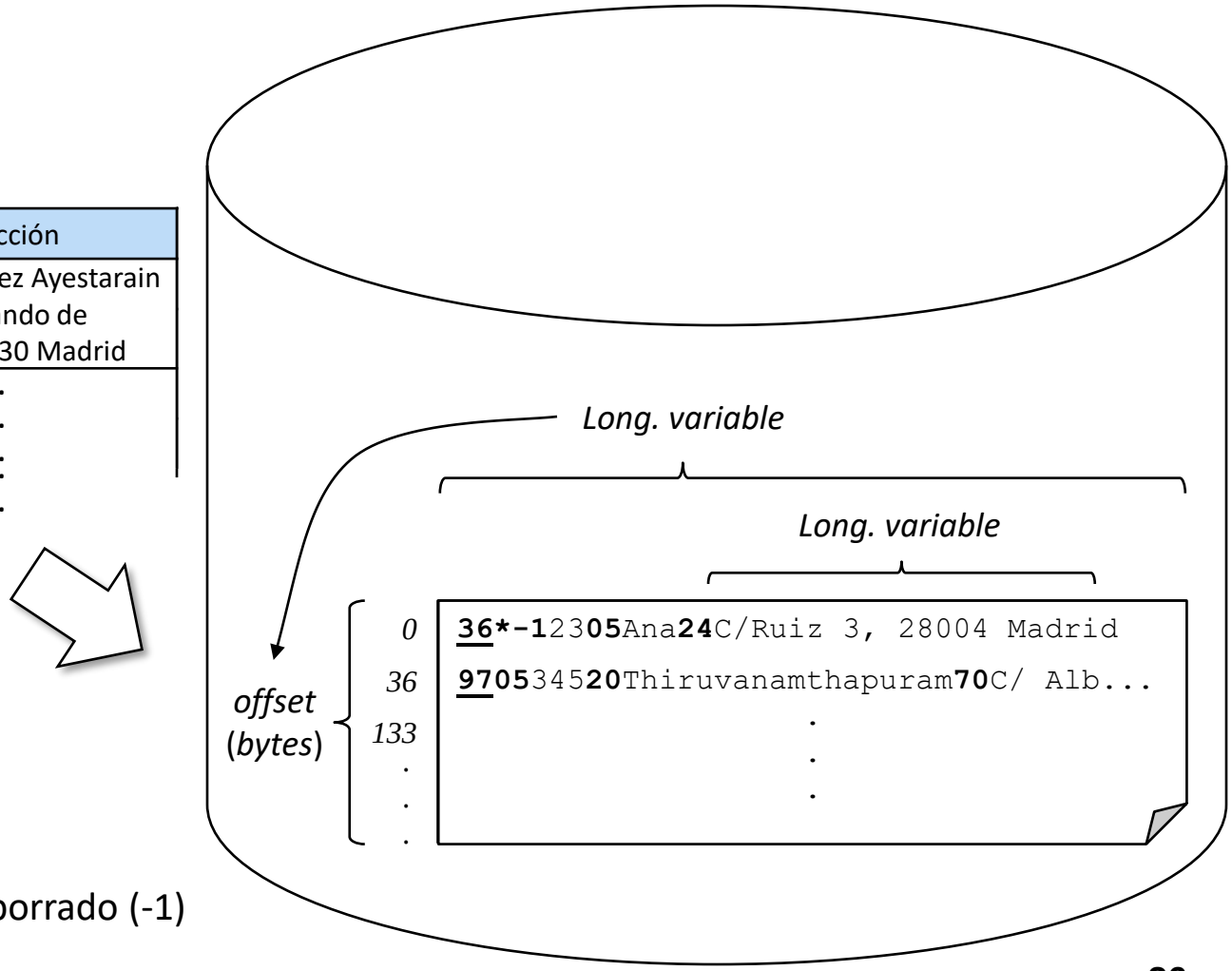
nº registros x (seek + latencia)
+ (tamaño bloque / tamaño registro) x transferencia!!



Almacenamiento de tablas: Eliminación de registros

EMPLEADO

DNI	Nombre	Dirección
345	Thiruvanamthapuram	C/ Albino Pérez Ayestarain 14, San Fernando de Henares, 28830 Madrid
.	.	.
.	.	.
.	.	.
.	.	.



1. Se marca como borrado (*)
2. Se indica final de cadena de borrado (-1)

Borrado de registros

Fichero inicial

Datos

3605123**05**Ana**24**C/Ruiz 3, 28004 Madrid
9705345**20**Thiruvanamthapuram**70**C/ Alb...
…
5205777**08**Carlos**30**Paseo …

Offset

0 36 133 .. 830

Borrar reg 15 (posición 14)

Datos

3605123**05**Ana**24**C/Ruiz 3, 28004 Madrid
9705345**20**Thiruvanamthapuram**70**C/ Alb...
…
52*-177**08**Carlos**30**Paseo …

Offset

0 36 133 … 830

Borrado

14

Borrado de registros

Borrar 2º reg (posición 1)

Datos

```
3605123 05Ana 24C/Ruiz 3, 28004 Madrid
97*14 4520Thiruvanamthapuram 70C/Alb...
                    …
52*-1 7708Carlos 30Paseo …
```

Offset: 0 36 133 … 830

Borrado: 1

Borrar 1º reg (posición 0)

Datos

```
36*01 2305Ana 24C/Ruiz 3, 28004 Madrid
97*14 4520Thiruvanamthapuram 70C/Alb...
                    …
52*-1 7708Carlos 30Paseo …
```

Offset: 0 36 133 … 830

Borrado: 0

Estrategias de eliminación / inserción

- Eliminación: marcar el registro, dejando un hueco
- Formar una lista enlazada con los huecos disponibles

  (las direcciones de registros actúan como “punteros” soft)
- Periódicamente, compactar
- Al insertar, buscar un hueco (en otro caso, al final del archivo)

  $\rightarrow$ se retrasa la necesidad de compactación
- Con registros de longitud fija, la estrategia es sencilla
- Con longitud variable: first-fit, best-fit, worst-fit

Recordar…

- Direccionamiento de los registros
  - Longitud fija: nrr (nº relativo de registro)
  - Longitud variable: offset en bytes
  - Las direcciones físicas de registros se utilizan en la formación de listas de registros borrados, y (lo veremos en breve) en la creación de índices
- Gestión de la eliminación/inserción de registros
  - En best/worst-fit, mantener ordenada la lista enlazada de registros borrados
  - Con longitud fija, el orden no importa

    (pero se usan igualmente listas de registros borrados)

Inserción de registros

Fichero

Datos

```
36*0123 05Ana 24C/Ruiz 3, 28004 Madrid
97*14 4520Thiruvanamthapuram 70C/Alb...
                    …
52*-1 7708Carlos 30Paseo …
```

Offset: 0 36 133 … 830

Borrado: 0

Insertar un registro de longitud 105

- No cabe en ningún registro borrado
- Inserta al final
  - Se añade en el fichero de datos
  - Se añade una entrada al fichero de offset
  - No se modifica el fichero de primer borrado

Inserción de registros

Fichero

Datos

36*0123**05**Ana**24**C/Ruiz 3, 28004 Madrid
97*1445**20**Thiruvanamthapuram**70**C/Alb...
…
52*-177**08**Carlos**30**Paseo …

Offset: 0 36 133 … 830

Borrado: 0

Insertar (first-fit) un registro de longitud 40
(**400**5888**07**Pedro**27**C/García de Paredes 27 Ma)

Datos

36*0223**05**Ana**24**C/Ruiz 3, 28004 Madrid
4005888**07**Pedro**27**C/García de Paredes 27 Ma**57*15**…
…
52*-177**08**Carlos**30**Paseo …

Offset: 0 36 76 133 … 830

Borrado: 0

Inserción de registros

Insertar (first-fit) un registro de longitud 40
(**40**0588807Pedro**27**C/García de Paredes 27 Ma)

Datos

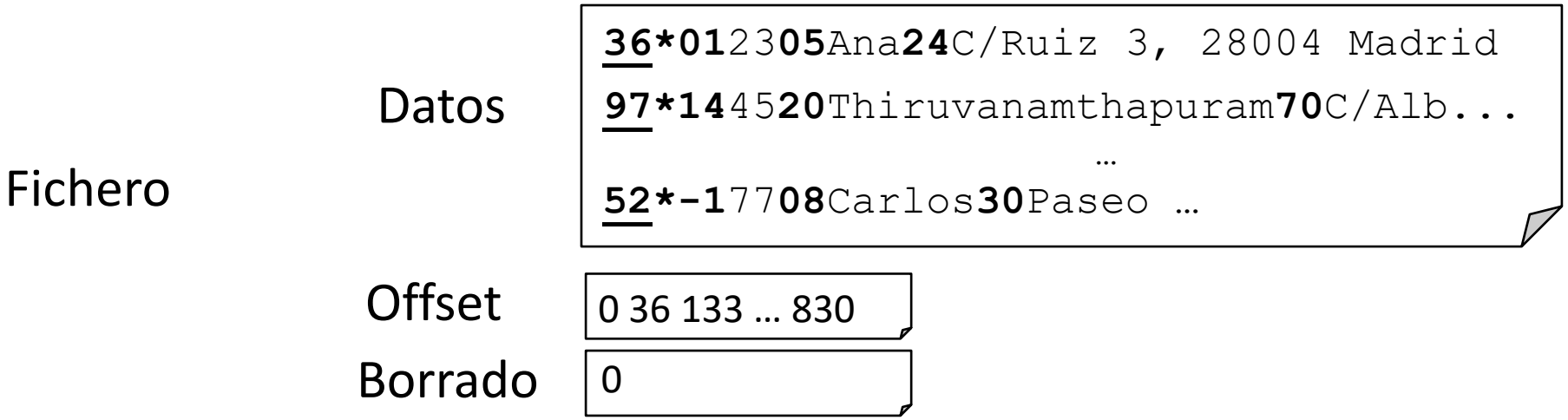
```
36*0223 05Ana24C/Ruiz 3, 28004 Madrid
40058880 7Pedro27C/García de Paredes 27 Ma
                                    …
52*-17708Carlos30Paseo …
```

Offset: 0 36 133 … 830

Borrado: 0

- No es necesario crear un nuevo registro con el espacio sobrante, se puede dejar el hueco sin usar (después de Ma hay un hueco sin usar)
- Proceso de compactación (No se puede modificar el fichero durante el proceso de compactación)

Inserción de registros



Insertar (best-fit) un registro de longitud 40
(**40**0588807Pedro**27**C/García de Paredes 27 Ma)

- Se mantiene una lista de registros borrados ordenados por tamaño
- Se inserta en el registro 14 (hueco menor donde encaja)

Inserción de registros

Fichero

Datos

```
36*0123 05Ana 24C/Ruiz 3, 28004 Madrid
97*14 4520Thiruvanamthapuram 70C/Alb...
                    …
52*-1 7708Carlos 30Paseo …
```

Offset: 0 36 133 … 830

Borrado: 0

Insertar (worst-fit) un registro de longitud 40
(**400**5888**07**Pedro**27**C/García de Paredes 27 Ma)

- Se mantiene una lista de registros borrados ordenados por tamaño
- Se inserta en el registro 01 (hueco mayor donde encaja)

```
select * from User
where userid = 15
```

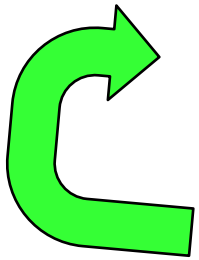
Consulta a BDs…

Indexación

select * from tabla where campo = valor

	ID	*Director*	*Título*	*Fecha*
0	62	Kubrick, Stanley	2001 Odisea del espacio	1968
45	254	Kubrick, Stanley	Espartaco	1960
76	50	Kubrick, Stanley	Senderos de Gloria	1957
114	47	Kurosawa, Akira	Los siete samurais	1954
142	42	Kurosawa, Akira	Rashomon	1950
173	71	Kurosawa, Akira	Dersu Uzala	1975
202	110	Besson, Luc	Leon	1994
223	758	Vadim, Jean	Leon	1992

Búsqueda



1. Leer registro
2. Comprobar si el campo cumple la condición
3. Volver a 1

No es una buena solución

- *Recorrer todo el fichero para cada consulta*
- *Complejidad O(n) (n es número de registros del fichero)*
- *Operaciones en disco y no en memoria => muy lento*

Temario

- Introducción y fundamentos
- Introducción a SQL
- Modelo Entidad / Relación
- Modelo relacional
- Diseño relacional: formas normales
- Consultas
  - Cálculo relacional
  - Álgebra relacional
- Implementación de bases de datos
  - Estructura física: campos y registros
  - Indexación
    - Índices simples
    - Árboles B, B* y B+

Folk, cap. 7, 8

Ejecución de consultas

- Parsing de SQL a estructura (árbol) de consulta (álgebra) en RAM
- Planificación del orden y en su caso recombinación de operaciones para optimizar el coste de la consulta
- Reducción de la consulta a combinación de condiciones simples de comparación de atributos, en la medida de lo posible
- Las condiciones de comparación atributo / constante se pueden resolver por búsqueda binaria en un índice
- La comparación de dos atributos se resuelve por operaciones de procesamiento cosecuencial; la combinación and/or de comparaciones atributo / constante también
- Otras consultas más complejas aprovechan menos la optimización que permiten de los índices

Consultas simples
(costes de acceso a disco)

- Comparación atributo / constante –para simplificar consideremos $t.A = cte$
- ¿Recorrer todos los registros hasta encontrar los que cumplen la condición?
  - $O(n)$ con $n =$ nº de registros en la tabla, $O(n)$ de accesos al disco (muy lento)
- ¿Ordenar el archivo de datos por el atributo A en cuestión $\rightarrow$ búsqueda binaria?
  - $O(\log n)$, pero mantener el orden con cada actualización es $O(n)$, no es viable
  - Además, sólo resuelve consultas sobre un único atributo de todos los que tenga el esquema
  - ¿se puede conseguir mejor que $O(\log n)$?
- Solución óptima: ¡índices!
  - Búsqueda en $O(1)$, solo 1 acceso a disco
  - Sin limitación en los campos de búsqueda

Indexación

- En términos abstractos, un índice para un atributo A es un mapping φ_A

  φ_A : valor de $A \rightarrow$ dirección de los registros con ese valor en el campo A

- Se necesita un índice aparte para cada campo que se quiera buscar
- La "ejecución" del mapping debe ser muy poco costosa, idealmente $O(1)$
- Dada la dirección del registro (o registros) que cumple el valor buscado, sólo queda hacer 1 acceso por registro (obviamente, menos es imposible)
- Técnicas principales de indexación para BDs
  - **Índices simples**: el mapping lo da un array –en RAM– de pares valor campo / dirección
  - **Árboles B**: semejantes a los índices simples, pero en vez de un array, la estructura es más sofisticada (un árbol –en disco– con técnicas de actualización elaboradas)

Índice simple

- **Cabe en RAM** y las operaciones en él (búsqueda, actualización, reordenación, etc.) se hacen en memoria RAM y, por tanto, tienen coste mínimo comparado con el acceso a disco.
- **Ordenado** para permitir búsqueda binaria y búsquedas por desigualdad (rangos) en RAM con complejidad $O(\log n)$. Una vez encontrado en RAM se realiza el acceso al disco con coste $O(1)$.
- Permite búsqueda por tantos campos como se desee, sin tocar el fichero de datos.

Índice simple (cont)

- Un índice simple es un array de pares *valor campo* / *localizador registro*
  - Es común llamar “clave” al valor de campo.
  - No confundir con las nociones de claves en el modelo relacional, E/R, SQL
  - El “localizador” de registro puede ser una dirección física (localizador físico) o una clave primaria (identificador lógico)
- Distinción básica
  - Índices primarios: indexan una clave (y usan direcciones de registros)
    El índice primario juega un papel especial: es el que realmente contiene direcciones físicas de los registros.
  - Índices secundarios: todos los demás (usan claves primarias como id de registros)

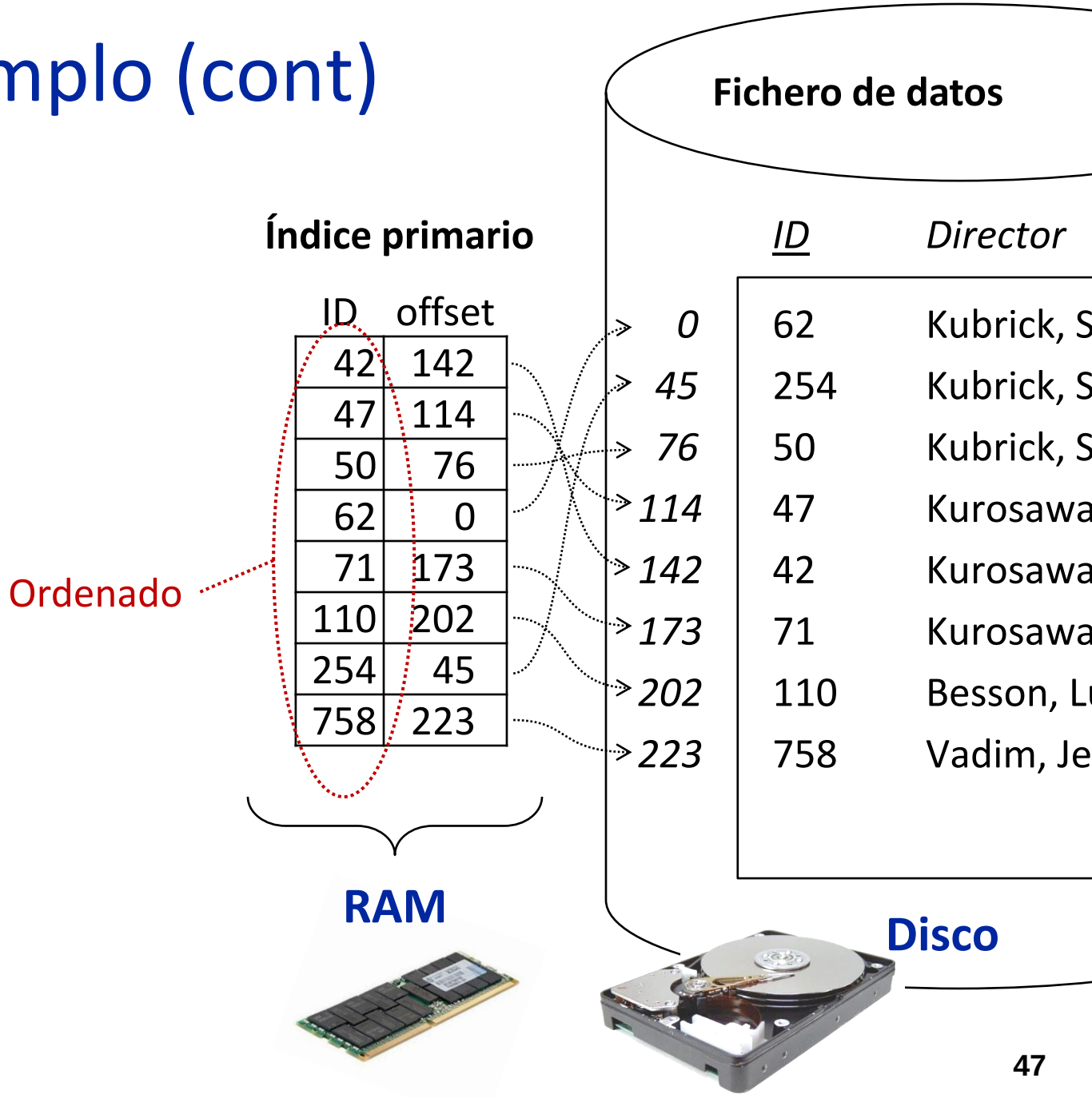
Ejemplo

Fichero de datos

	ID	*Director*	*Título*	*Fecha*
0	62	Kubrick, Stanley	2001 Odisea del espacio	1968
45	254	Kubrick, Stanley	Espartaco	1960
76	50	Kubrick, Stanley	Senderos de Gloria	1957
114	47	Kurosawa, Akira	Los siete samurais	1954
142	42	Kurosawa, Akira	Rashomon	1950
173	71	Kurosawa, Akira	Dersu Uzala	1975
202	110	Besson, Luc	Leon	1994
223	758	Vadim, Jean	Leon	1992

Disco

Ejemplo (cont)



Ejemplo (cont)

Índice secundario

Director	ID
Besson, Luc	110
Kubrick, Stanley	50
Kubrick, Stanley	62
Kubrick, Stanley	254
Kurosawa, Akira	42
Kurosawa, Akira	47
Kurosawa, Akira	71
Vadim, Jean	758

Ordenado

Ordenado

Índice primario

ID	offset
42	142
47	114
50	76
62	0
71	173
110	202
254	45
758	223

Ordenado

RAM

Fichero de datos

	ID	*Director*
0	62	Kubrick, S
45	254	Kubrick, S
76	50	Kubrick, S
114	47	Kurosawa
142	42	Kurosawa
173	71	Kurosawa
202	110	Besson, L
223	758	Vadim, Je

Disco

Búsqueda

Director = Kubrick, Stanley

- Índice secundario, búsqueda binaria de Kubrick, Stanley
- ID: 50, 62 y 254
- Índice primario, búsqueda binaria de ID 50, 62 y 254
- Offset: 76, 0 y 45
- Acceso a disco para los offsets anteriores

Ejemplo (cont)

Índice secundario

Director	ID
Besson, Luc	110
Kubrick, Stanley	50
Kubrick, Stanley	62
Kubrick, Stanley	254
Kurosawa, Akira	42
Kurosawa, Akira	47
Kurosawa, Akira	71
Vadim, Jean	758

Índice primario

ID	nrr
42	4
47	3
50	2
62	0
71	5
110	6
254	1
758	7

Fichero de datos

	ID	*Director*
0	62	Kubrick, S
1	254	Kubrick, S
2	50	Kubrick, S
3	47	Kurosawa
4	42	Kurosawa
5	71	Kurosawa
6	110	Besson, L
7	758	Vadim, Je

RAM

Si los registros tienen long. fija, se usan nrr en lugar de offset!

Disco

Ejemplo (cont)

Índice secundario

Director	ID
Besson, Luc	110
Kubrick, Stanley	50
Kubrick, Stanley	62
Kubrick, Stanley	254
Kurosawa, Akira	42
Kurosawa, Akira	47
Kurosawa, Akira	71
Vadim, Jean	758

Índice secundario

Fecha	ID
1950	42
1954	47
1957	50
1960	254
1968	62
1975	71
1992	758
1994	110

Índice primario

ID	offset
42	142
47	114
50	76
62	0
71	173
110	202
254	45
758	223

Fichero de datos

ID	*Director*
0	62 Kubrick, S
45	254 Kubrick, S
76	50 Kubrick, S
114	47 Kurosawa
142	42 Kurosawa
173	71 Kurosawa
202	110 Besson, L
223	758 Vadim, Je

Disco

Búsqueda

Director = Kubrick, Stanley AND fecha = 1957

- Índice secundario de directores, búsqueda binaria de Kubrick, Stanley
- ID: 50, 62 y 254
- Índice secundario de fecha, búsqueda binaria de 1957
- ID: 50
- ID 50 coincide en ambas búsquedas (búsqueda cosecuencial)
- Índice primario, búsqueda binaria de ID 50
- Offset: 76
- Acceso a disco para el offset anterior

Operaciones – índice primario

Operaciones en RAM
Operaciones en disco

- Crear índice
  - Leer fichero de datos (secuencial por bloques) y construir array en RAM
  - Ordenar array
  - Guardar (secuencial por bloques) array en archivo
- Cargar índice en RAM (si es posible)
  - Lectura (secuencial por bloques) del archivo de índice → índice en RAM
- Búsqueda de un registro
  - Búsqueda (binaria) en índice → dirección (offset) del registro buscado
  - Acceso (seek + latencia) al registro sabiendo su posición … leer un solo registro

Operaciones – índice primario (cont)

Operaciones en RAM
Operaciones en disco

- Inserción de registro
  - Archivo de datos: escribir el registro en la dirección *d* que corresponda
  - Índice: añadir entrada <*clave*, *valor*> manteniendo el orden

- Eliminación de registro
  - Archivo de datos: eliminar por el método de gestión del espacio que se desee
  - **Índice: eliminar** entrada correspondiente
  - Compactación → reconstruir el índice

Operaciones – índice primario (cont)

Operaciones en RAM
Operaciones en disco

- Modificación de un campo
  - SI hay espacio en registro y NO modifica la clave
    - Archivo de datos: modificar el valor del campo en el fichero
    - Índice: No se modifica
  - SI hay espacio en registro y SI modifica la clave
    - Archivo de datos: modificar el valor del campo en el fichero
    - Índice: eliminar e insertar en el índice
  - NO hay espacio en registro y NO modifica la clave
    - Archivo de datos: borrar e insertar al final del fichero
    - Índice: actualizar offset del índice
  - NO hay espacio en registro y SI modifica la clave
    - Archivo de datos: borrar e insertar el final del fichero
    - Índice: eliminar e insertar en el índice

Operaciones – índice secundario

Operaciones en RAM
Operaciones en disco

- Búsqueda de registro
  - Buscar clave secundaria en índice secundario → clave primaria
  - Buscar en índice primario → dirección del registro en archivo de datos
  - Acceder al registro en disco, leer
- Inserción de registro
  - Archivo de datos: insertar el registro
  - Índice: añadir entrada <*clave*, *valor*> manteniendo el orden
- Eliminación de registro
  - Archivo de datos: eliminar el registro
  - Índice: se puede **posponer la eliminación**
  - El intento de acceso a un registro borrado se detecta a nivel del índice primario
  - Periódicamente, limpiar el índice secundario

Operaciones – índice secundario (cont)

Operaciones en RAM
Operaciones en disco

- Modificación de un campo
  - Archivo de datos: modificar (según visto anteriormente)
  - Índice: si **afecta a la clave secundaria**, recolocar para mantener el orden (eliminar + insertar)
  - Índice: si **afecta a la clave primaria**, actualizar en el secundario la clave cambiada
  - Índice: si afecta a **otros campos**: el índice secundario **no cambia**, aunque se moviese el registro dentro del fichero

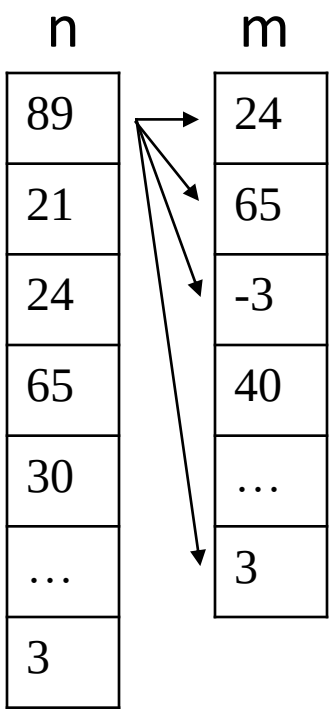
Procesamiento cosecuencial

- Una forma optimizada de construir uniones e intersecciones de listas ordenadas en diversas operaciones de consulta
  - AND y OR de condiciones simples $\sigma_{A=c}$
  - Unión e intersección (de filas de tabla ordenadas por clave primaria)
  - Condiciones de tipo $\sigma_{A=B}$
  - Joins naturales y inner joins en general
  - Otras operaciones
- El esquema consiste en iterar sincronizadamente sobre las dos listas
  - Se consigue ejecutar la operación con complejidad $O(n+m)$ en lugar de $O(nm)$
  - Esto explica la utilidad de ordenar los índices (además de por permitir búsqueda binaria en RAM, y por facilitar búsquedas por rango)
  - El resultado de la operación también queda ordenado, por lo que el método es apto para anidar operaciones

¿Como buscar mismos datos en 2 índices?

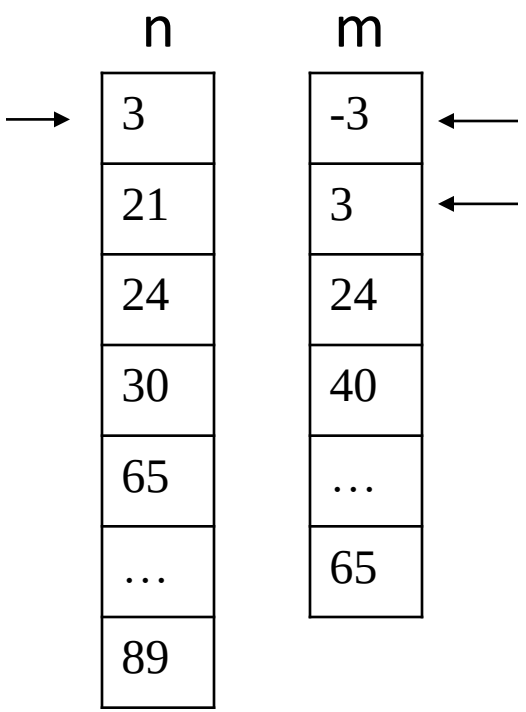
valor de n = valor de m

Índices no ordenados



Para cada valor del primer índice buscar en el segundo => O(nm)

Índices ordenados



Se recorren los dos índices a la vez => O(n+m)

Match y merge

Match (intersección de dos listas ordenadas)

1. Iniciar la iteración posicionada en el primer elemento de las dos listas
2. Si los elementos actuales son distintos, avanzar en la lista del más “pequeño”
3. Si por el contrario son iguales, añadir ese elemento al resultado, y avanzar en ambas listas
4. Si alguna de las dos listas ha llegado al final, fin
5. En otro caso, volver a 2

Merge (unión de dos listas ordenadas)

…Igual que match, salvo:

- En el paso 2 el elemento más pequeño se añade a la lista resultado
- En el paso 4, se termina sólo cuando *ambas listas* han llegado al final; mientras sólo una terminó, se sigue avanzando en la otra añadiendo todos sus elementos restantes al resultado

Match y merge

MATCH (lista1, lista2)

```
Inicializar
x ← input (lista1)
y ← input (lista2)
while quedan_elementos do
        if x < y then
                x ← input(lista1)
        else if y < x then
                y ← input (lista2)
        else write (salida, x)
                x ← input(lista1)
                y ← input(lista2)
```

MERGE (lista1, lista2)

```
Inicializar
x ← input (lista1)
y ← input (lista2)
while quedan_elementos do
        if x < y then
                write (salida, x)
                x ← input(lista1)
        else if y < x then
                write (salida y)
                y ← input (lista2)
        else write (salida, x)
                x ← input(lista1)
                y ← input(lista2)
```

Ejemplo

VUELO

Numero	Origen	Destino	Salida
345	MAD	CDG	12:30
321	MAD	ORY	19:05
165	LHR	CDG	09:55
127	JFK	CDG	23:55
903	CDG	LHR	14:40
447	CDG	LHR	17:00
204	MAD	CDG	16:15
618	MAD	LGW	07:25
736	LAX	CDG	11:30

Vuelos entre Barajas y Charles de Gaulle

Índice secundario 1

Origen	Numero
CDG	447
CDG	903
JFK	127
LAX	736
LHR	165
MAD	204
MAD	321
MAD	345
MAD	618

Índice secundario 2

Destino	Numero
CDG	127
CDG	165
CDG	204
CDG	345
CDG	736
LGW	618
LHR	447
LHR	903
ORY	321

Mach

Índice primario

Numero	nrr
127	3
165	2
204	6
321	1
345	0
447	5
618	7
736	8
903	4

VUELO

nrr	Numero	Origen	Destino	Salida
0	345	MAD	CDG	12:30
1	321	MAD	ORY	19:05
2	165	LHR	CDG	09:55
3	127	JFK	CDG	23:55
4	903	CDG	LHR	14:40
5	447	CDG	LHR	17:00
6	204	MAD	CDG	16:15
7	618	MAD	LGW	07:25
8	736	LAX	CDG	11:30

Recordar…

- Diferencias entre índices primarios y secundarios
- Índices siempre ordenados por clave
  - En los secundarios cuando se repite la secundaria, ordenar por primaria
- Las direcciones físicas de registros en el índice primario son:
  - Nrr cuando los registros se almacenan con longitud fija
  - Offset en bytes cuando se utiliza longitud variable
- Los índices simples tienen que caber en RAM

Uso de índices en consultas más generales

$\sigma_{A=c}(tabla)$	Obvio con índice por campo A	$O(\log n)$
$\sigma_{A>c \text{ and } A<d}(tabla)$	Buscar c y d en índice A y tomar claves entre ambas	$O(n)$
$\sigma_{A=c \text{ and } B=d}(tabla)$	Buscar c en índice A ∩ buscar d en índice B	$O(n)$
$\sigma_{A=c \text{ or } B=d}(tabla)$	Buscar c en índice A ∪ buscar d en índice B	$O(n)$
$\sigma_{A=B}(tabla)$	Índice A "∩" índice B	$O(n)$
$T \bowtie_{A=B} S$	Índice A de T ∩ índice B de S + combinar registros	$O(n)$
$T \bowtie_{A<B} S$	Comparaciones "todos con todos" (o casi)	$O(n^2)$
…		

Obsérvese que…

- La ordenación de los índices es necesaria para obtener esta complejidad en las operaciones
  - Aún en los peores casos son operaciones en RAM
- El resultado de las operaciones en índices es un conjunto de direcciones de registros
  - A partir de ahí, el coste de acceso a los registros es inevitable, inherente a la consulta
  - El uso de índices no implica acceso a disco en ningún momento (salvo guardar y cargar índices)
  - En consultas que implican varias tablas, el resultado puede ser un conjunto de registros concatenados

Límite de los índices simples

- Los índices simples son una solución óptima para la ejecución de consultas
  - ¡Imposible mejorar 1 acceso a disco! ☺
- Todo lo razonado hasta aquí presupone que los índices se manejan en RAM

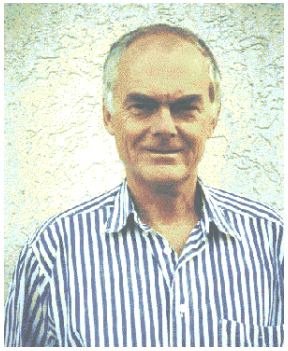
¿Y si un índice no cabe en RAM…?

- Índice multinivel
  - Idea que se puede evolucionar hacia los árboles B
- Índice por bloques (vs. índice denso)
  - Índice más pequeño, a cambio los bloques ordenados correlativamente
  - Idea que se puede evolucionar hacia los árboles B+

Temario

- Introducción y fundamentos
- Introducción a SQL
- Modelo Entidad / Relación
- Modelo relacional
- Diseño relacional: formas normales
- Consultas
  - Cálculo relacional
  - Álgebra relacional
- Implementación de bases de datos
  - Estructura física: campos y registros
  - Indexación
    - Índices simples
    - Árboles B, B* y B+ — Folk, cap. 9, 10

Árboles B



Rudolf Bayer

Edward M. McCreight

- Ideado en 1972 por R. Bayer & E. M. McCreight en Boeing
- “B” (vagamente) por “Boeing”, “Bayer”, “Balanced”…
- Ampliamente utilizado actualmente para indexar BDs
  - P.e. es el tipo de índice por defecto en Postgresql
- Una técnica de indexación cuando los índices simples ocupan más espacio en memoria del que podemos o deseamos consumir
- Índice estructurado como árbol almacenado en disco donde sólo una página (o unas pocas) están en RAM en cada momento

Árboles B (cont)

- Cuando un índice simple no cabe en RAM dejan de funcionar todos los supuestos que estábamos haciendo y la solución ya no es válida
- Con un índice en disco…
  - Buscar claves implica $\sim \log_2 n$ accesos a disco, mejor que $O(n)$ pero… ¿es posible algo mejor?
  - La actualización del índice supone $O(n)$ accesos a disco, ya no es viable

Árboles B (cont)

Ideas…

1. Dividir el índice en bloques que caben en RAM
   - Llamaremos "páginas" a estos bloques
   - Su tamaño puede ser p.e. el de bloque de disco: interesa que las páginas se puedan leer secuencialmente de disco, sin saltos
2. Crear un índice de bloques
   - ¿Y si este índice no cabe en RAM?
3. Crear un índice de índices de bloques…
   - Finalmente tenemos un índice multinivel $\rightarrow$ estructura de árbol
   - Podemos utilizar el mismo tamaño de página en todos los niveles;

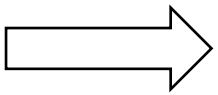
     así podemos almacenar todo en un único archivo homogéneo de índice

División de un índice en bloques (índice de bloques)

Índice primario

Numero	nrr
127	3
165	2
204	6
321	1
345	0
447	5
618	7
736	8
903	4

No cabe en memoria



Bloque que cabe en RAM

Numero	nrr
127	3
165	2
204	6

Bloque 0
(página 0)

Numero	nrr
321	1
345	0
447	5

Numero	nrr
618	7
736	8
903	4

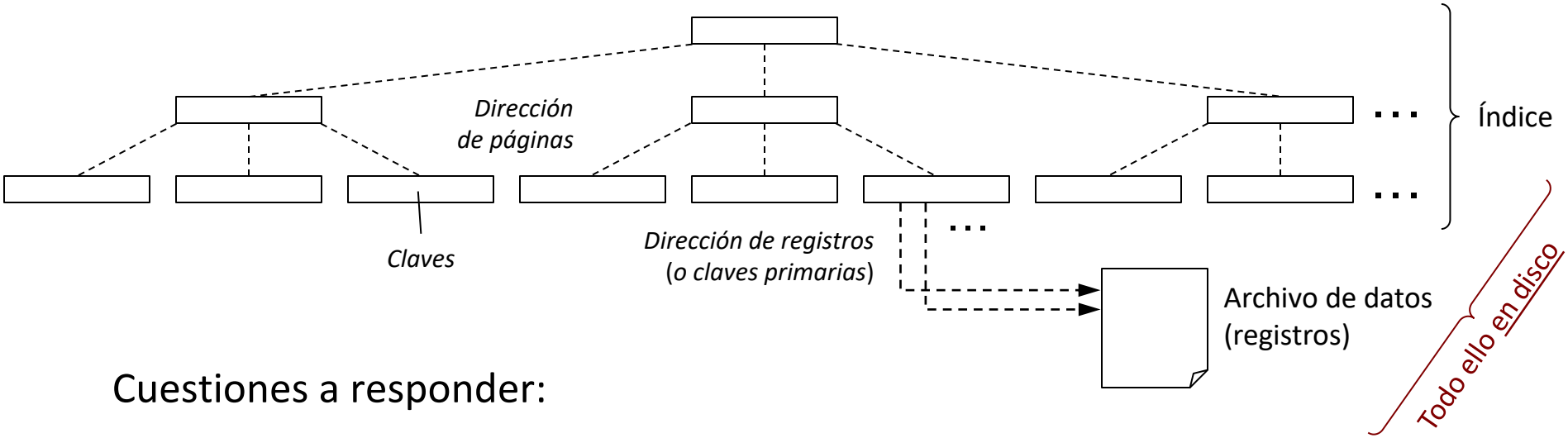
Bloque 2
(página 2)

Índice de bloques

Min	Max	pág
127	204	0
321	447	1
618	903	2

¿Cabe en memoria?

Árboles B (cont)



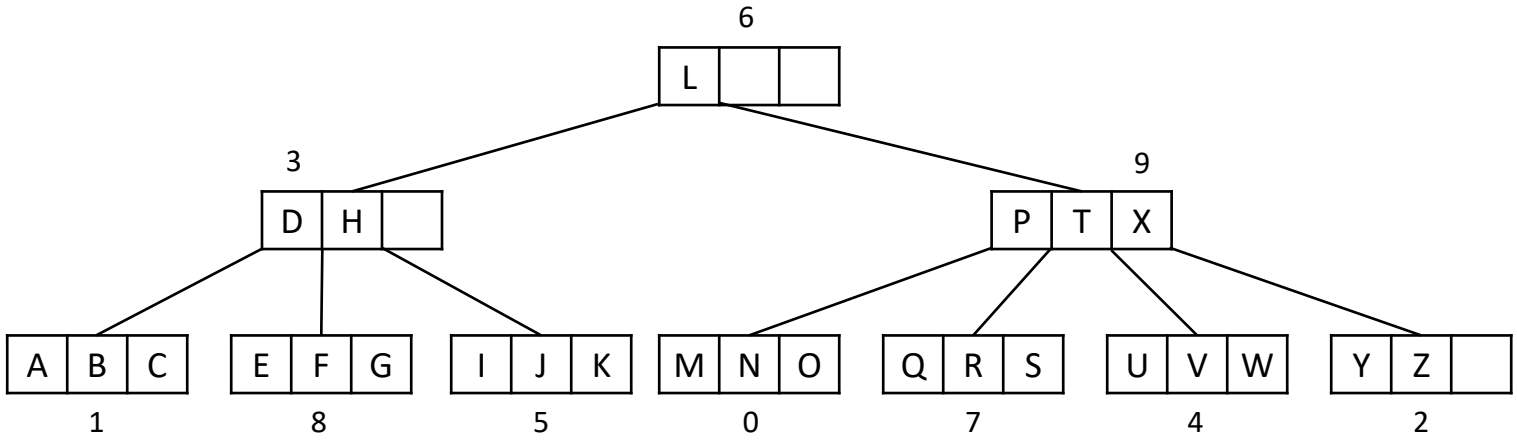
Cuestiones a responder:

- ¿Qué estructura damos a las páginas? Claves, direcciones de páginas, de registros…
- ¿Debemos mantener algún orden dentro de las páginas y entre ellas?
- ¿Cómo actualizamos un índice de este tipo?

Árboles B

Ejemplo

$k = 3$



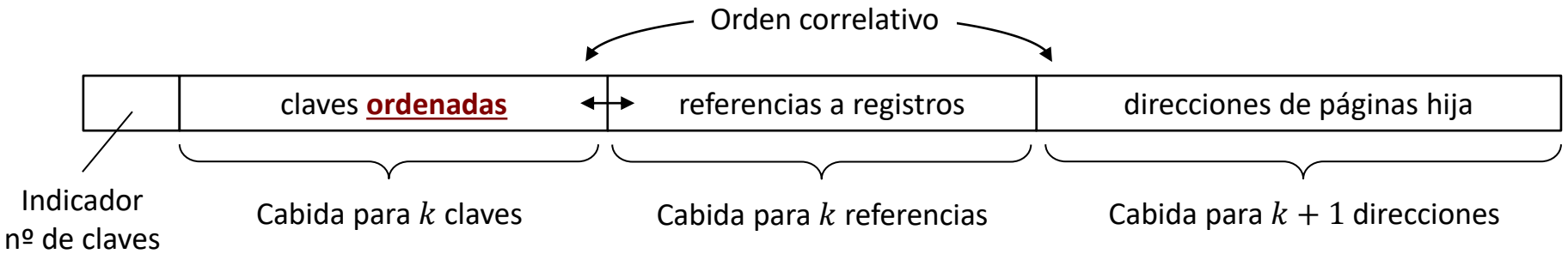
Definición de propiedades de un árbol B

1. Siendo k el número máximo de claves por página, el número máximo de descendientes por página es k+1.
2. Una página hoja contiene por lo menos $\lfloor k/2 \rfloor$ claves y no mas de k.
3. Cada página excepto la raíz y las hojas tiene por lo menos $\lfloor k/2 \rfloor$ +1 descendientes.
4. La raíz tiene por lo menos 2 descendientes a menos que sea una hoja.
5. Todas las hojas aparecen en el mismo nivel.
6. Una página que no es hoja con m descendientes contiene m-1 claves.

Árboles B – Estructura

- Contenido de una página
  - Claves ordenadas acompañadas por dirección de página hija; cada clave “separa” dos páginas hijas
  - Referencia a registros en disco (offset)
  - La clave que separa dos páginas es mayor que las claves de una y menor que las claves de la otra
  - Las direcciones de página en las páginas hoja son “null” (p.e. -1)

- Estructura en detalle:



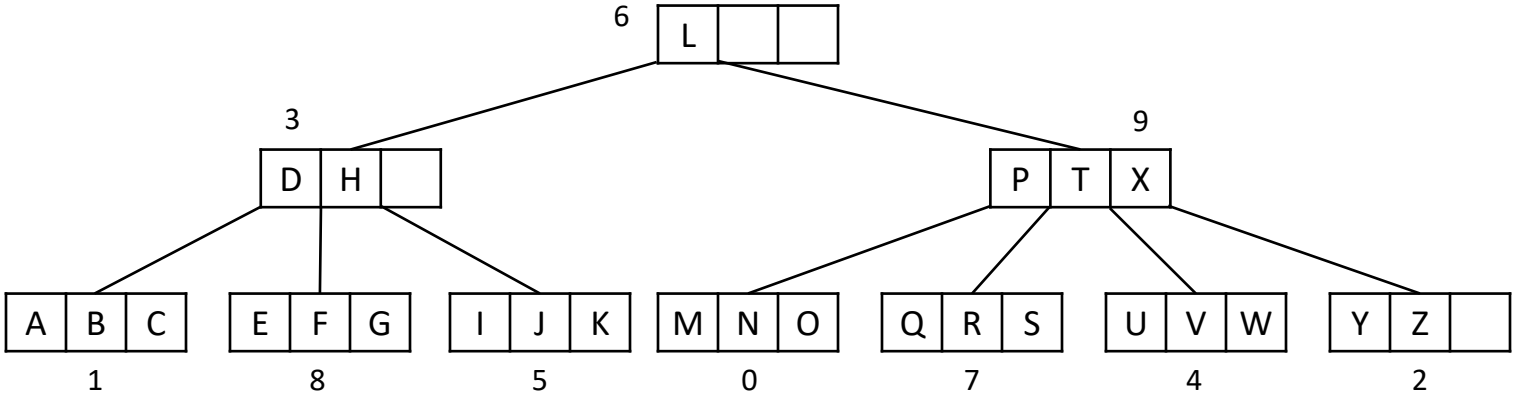
Árboles B – Estructura (cont)

- La cabecera del archivo de índice debe contener la dirección de la página raíz

- Dos opciones en el almacenamiento de claves
  - Árboles B y B*: todas las páginas contienen claves del índice y referencias a registros
  - Árboles B+: las hojas contienen todas las claves y referencias a registros, las demás contienen solo “separadores”

Ejemplo:

Vista lógica de un árbol B con $k = 3$



Vista física del árbol B

Raíz: 6

	Num claves	Claves			Referencia registros			n. de páginas descendientes			
0	3	M	N	O	···	···	···	-1	-1	-1	-1
1	3	A	B	C	···	···	···	-1	-1	-1	-1
2	2	Y	Z		···	···		-1	-1	-1	
3	2	D	H		···	···		1	8	5	
4	3	U	V	W	···	···	···	-1	-1	-1	-1
5	3	I	J	K	···	···	···	-1	-1	-1	-1
6	1	L			···			3	9		
7	3	Q	R	S	···	···	···	-1	-1	-1	-1
8	3	E	F	G	···	···	···	-1	-1	-1	-1
9	3	P	T	X	···	···	···	0	7	4	2

Número de página (np)

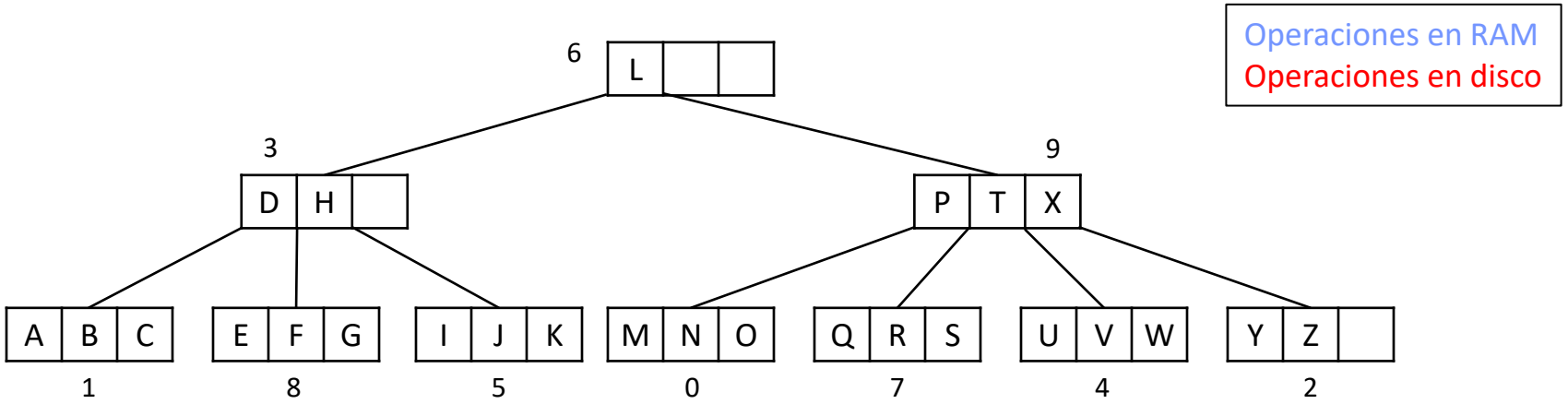
Fichero con registro de tamaño fijo

Árboles B – Operaciones

Búsqueda

1. Iniciar en *np* ← raíz
2. Si *np* = –1, **FIN** (no se encuentra)
3. Leer la página *np* a RAM, buscar la clave en la página
4. Si se encuentra la clave, **FIN** (clave encontrada)
5. Si no, *np* ← página descendiente correspondiente al orden de la que se busca, volver al paso 2

Ejemplo: buscar clave “R”



1. Leer número de página raíz, 6
2. Leer página 6 (búsqueda binaria de clave, no se encuentra e identifica seguir en página 9 (R>L))
3. Leer página 9 (búsqueda binaria de clave, no se encuentra e identifica ir a página 7 (P<R<T))
4. Leer página 7 (búsqueda binaria de clave, se encuentra => acceder a disco para leer registro)

Número de accesos a disco: 5 accesos (número de página raíz + 3 páginas + lectura de registro)

Árboles B – Operaciones

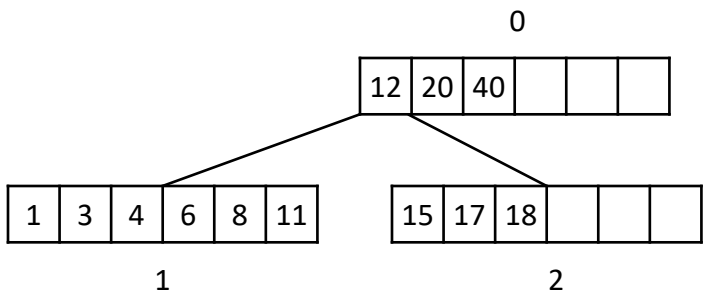
Inserción

1. Buscar la página hoja donde debería insertarse la clave (control de duplicados si es un índice primario)
2. Si hay espacio en la página para una clave más, insertar, **FIN**
3. Si no lo hay, **dividir la página en dos**, dejando una clave + nº página entre medias
   a) Si la página era la raíz, crear una nueva raíz (aumenta la altura del árbol), **FIN**
   b) Si no, subir la clave intermedia + nº de página a la página "padre", volver paso 2

Por este procedimiento, en general las páginas no estarán completamente llenas

División uno a dos

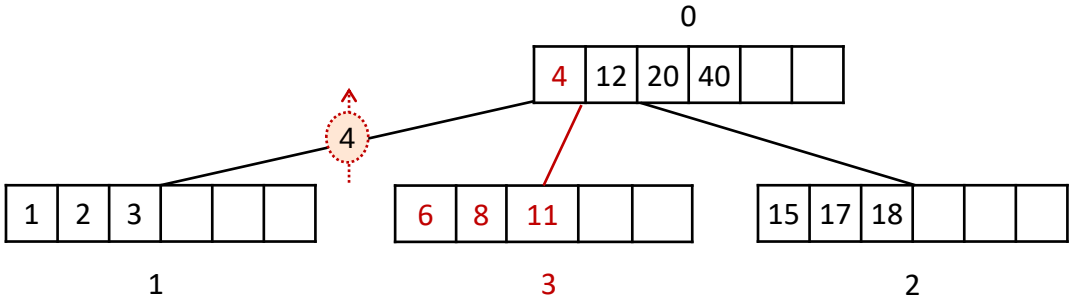
Cuando se llena una página dividirla en dos



$k = 6$

$\lfloor k/2 \rfloor = 3$

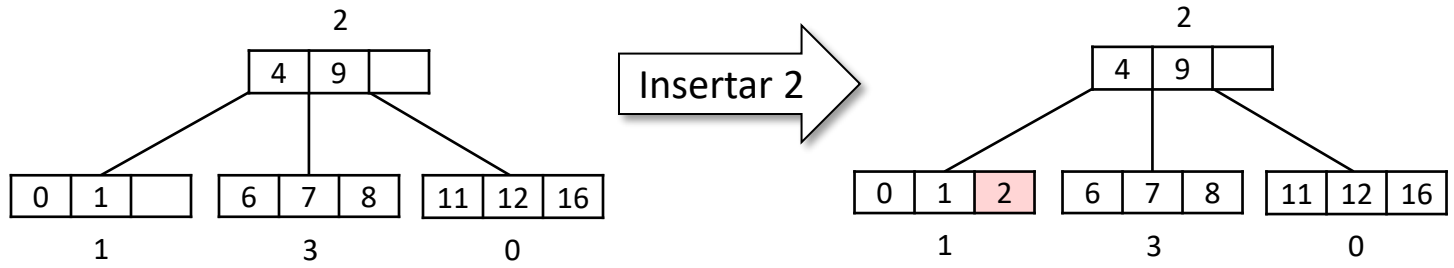
Insertar 2 (División de 6 + 1 en 3 + 1 + 3)

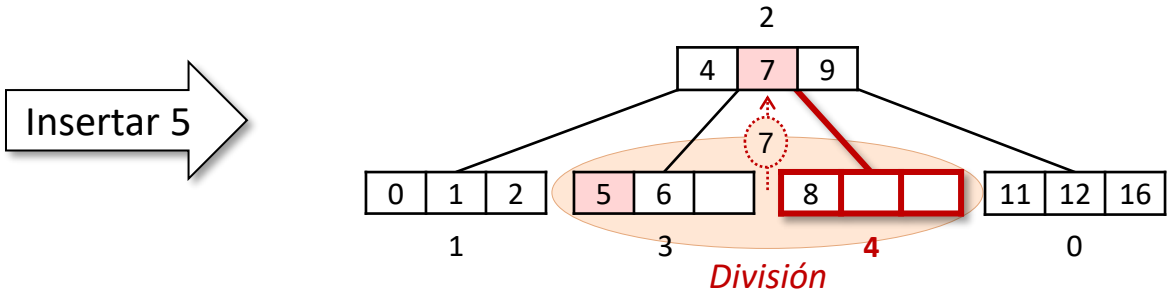


Claves mínimas por hoja $\lfloor k/2 \rfloor$, 3

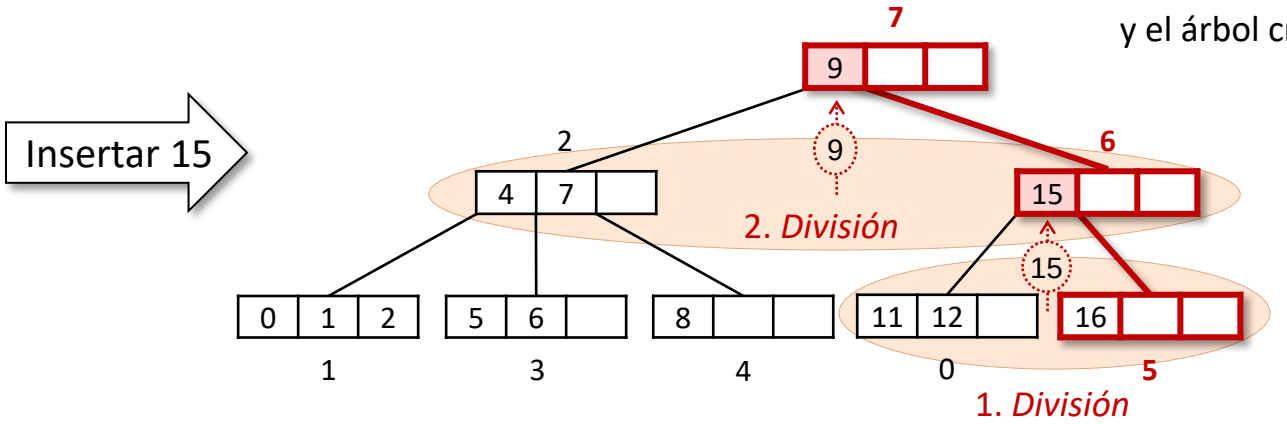
Ejemplo inserción en árbol B

$k = 3$





La división alcanza a la raíz
y el árbol crece en un nivel más



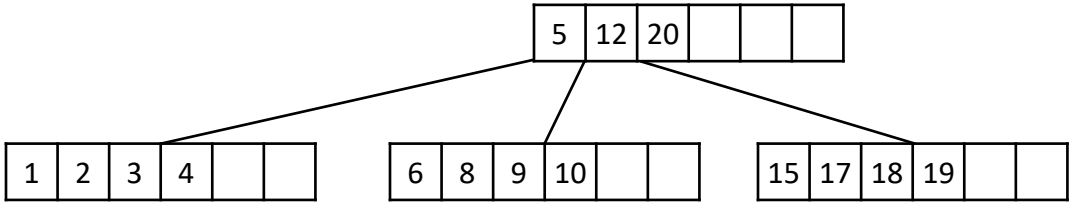
Árboles B – Operaciones (cont)

Eliminación

1. Buscar la página donde se encuentra la clave
2. Si no es una página hoja, **intercambiar** la clave con la clave "inmediatamente anterior o posterior en una hoja"
3. Eliminar la clave de la página hoja
4. Si la página tiene el mínimo de claves que queremos garantizar, **FIN**
5. En otro caso examínese los hermanos izquierdo y derecho:
   a) Intentar **redistribuir** con una página "hermana" inmediatamente contigua (en tal caso, **FIN**)
   b) Si esto no es posible, **concatenar** con una página "hermana" inmediatamente contigua, junto con la clave que las separa de la página padre. Si la concatenación se propaga hasta la raíz, se reduce en un nivel la altura del árbol

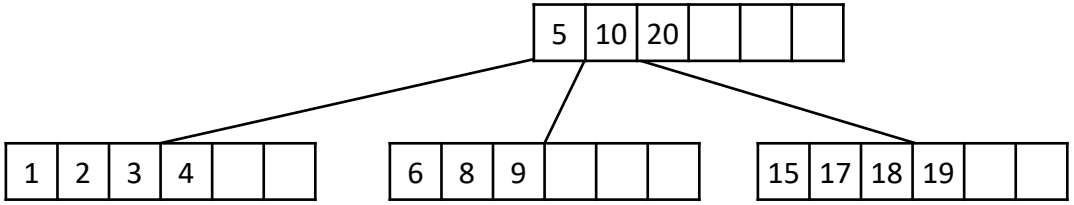
Intercambio para eliminar

Eliminar 12

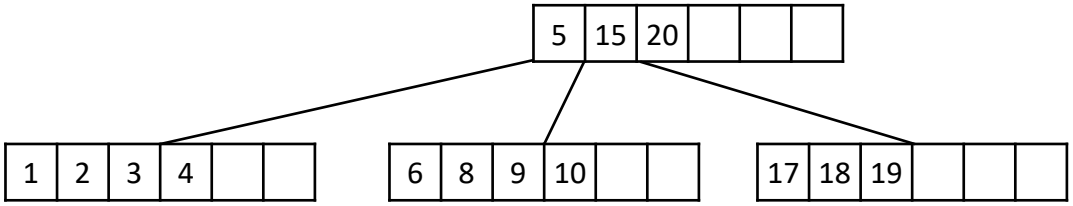


$k = 6$

Intercambio con la inmediatamente anterior (intercambio con 10 y elimino 12)



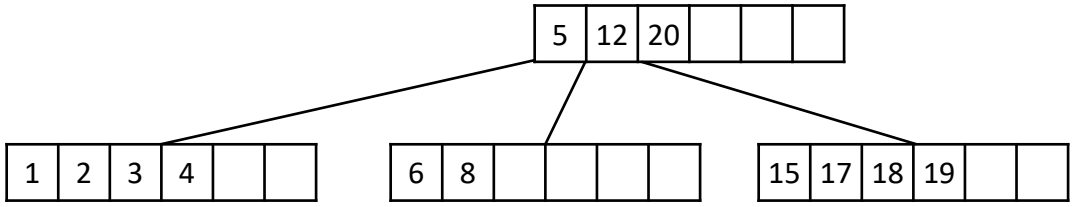
Intercambio con la inmediatamente posterior (intercambio con 15 y elimino 12)



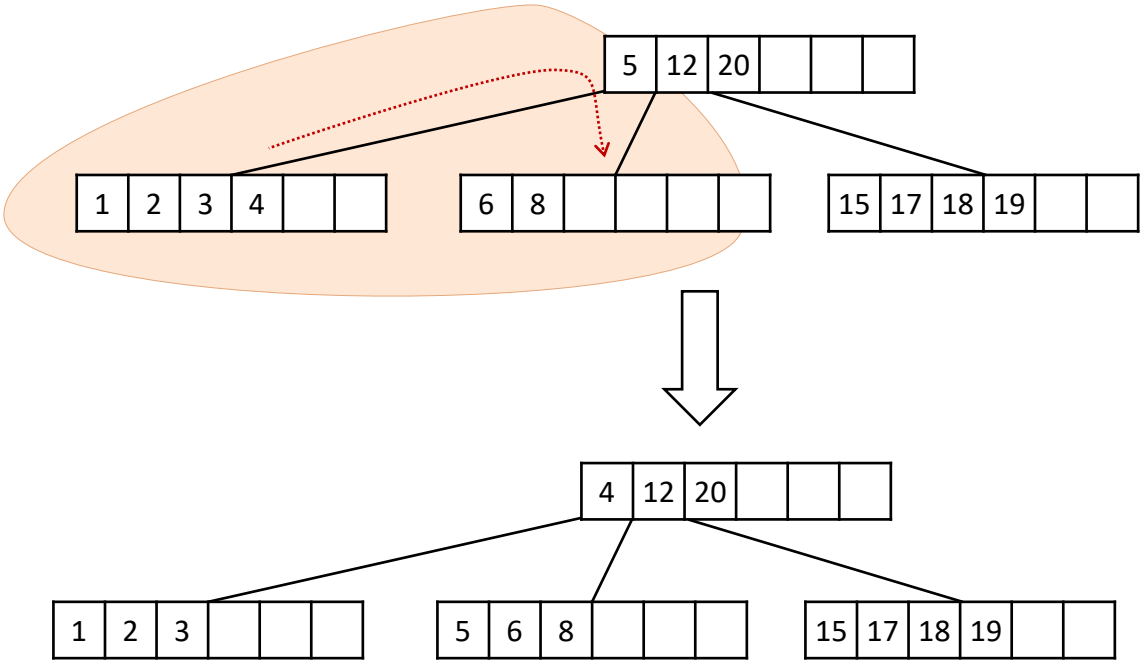
Redistribución

Una hoja se queda con menos elementos del mínimo

$k = 6$



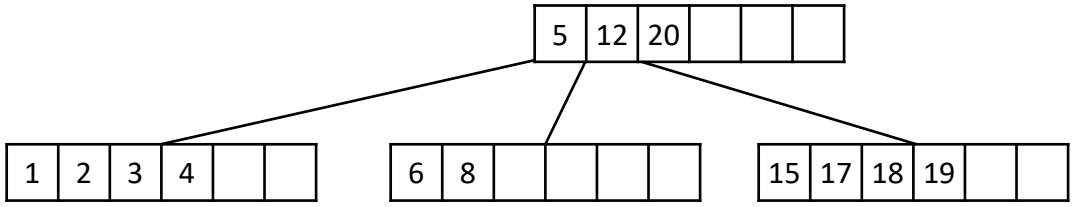
Con la hermana izquierda



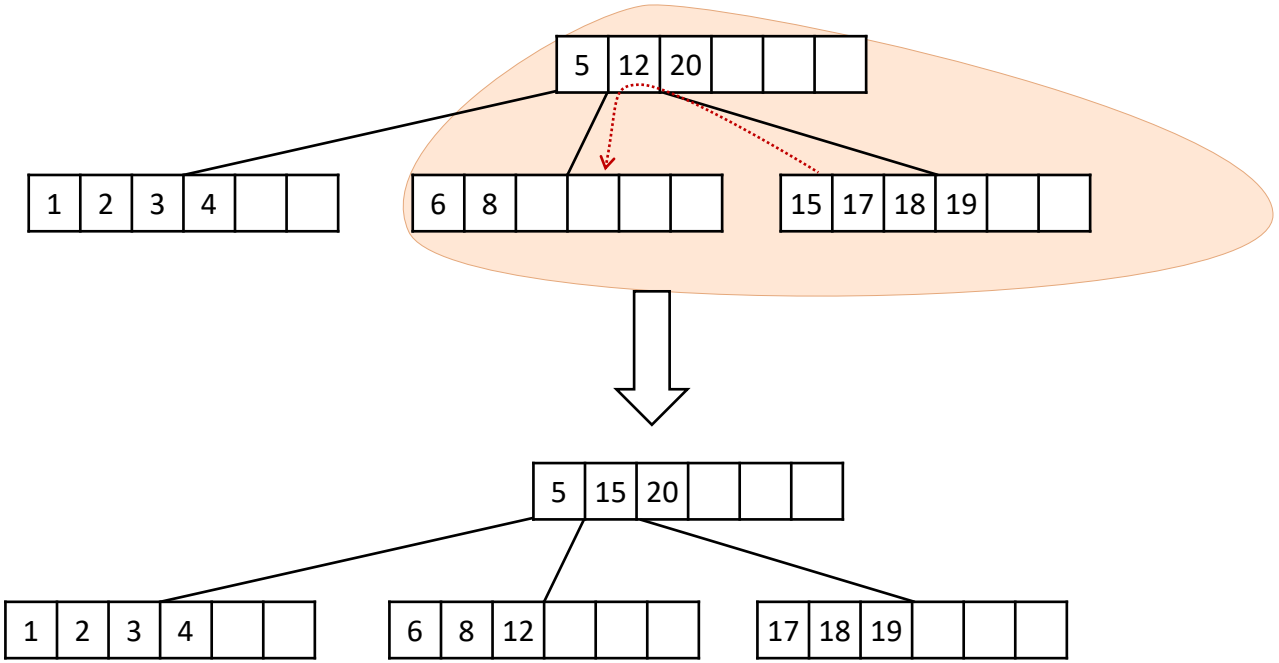
Redistribución

Una hoja se queda con menos elementos del mínimo

$k = 6$



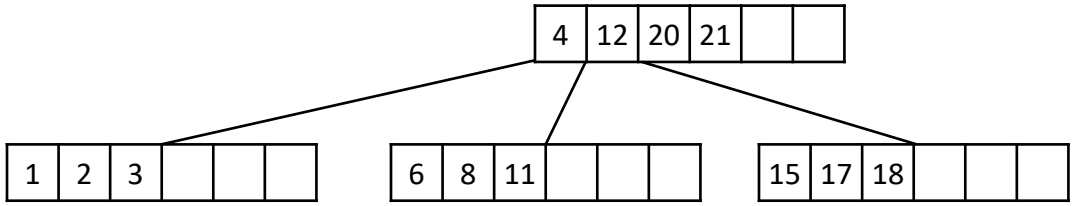
Con la hermana derecha



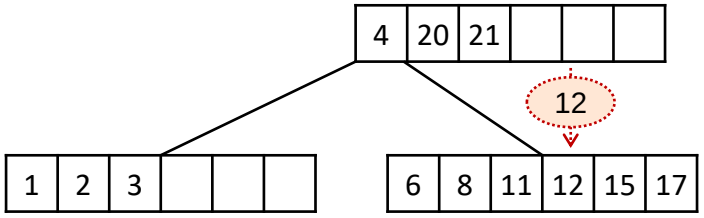
Concatenación dos a uno

Eliminar 18 (se hace redistribución hasta que dos hojas tienen el número mínimo de claves (3), en ese momento concatenación)

$k = 6$

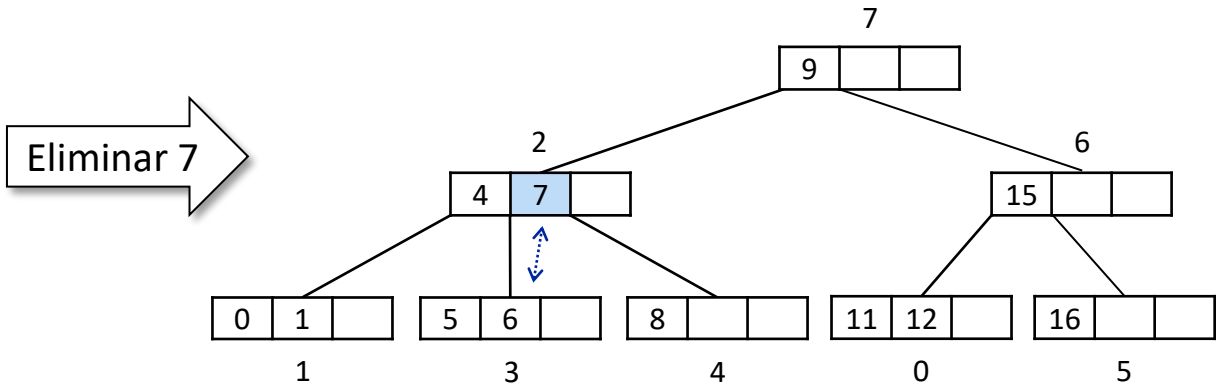


Concatenar dos en uno (3 + 1 + 2 en 6)

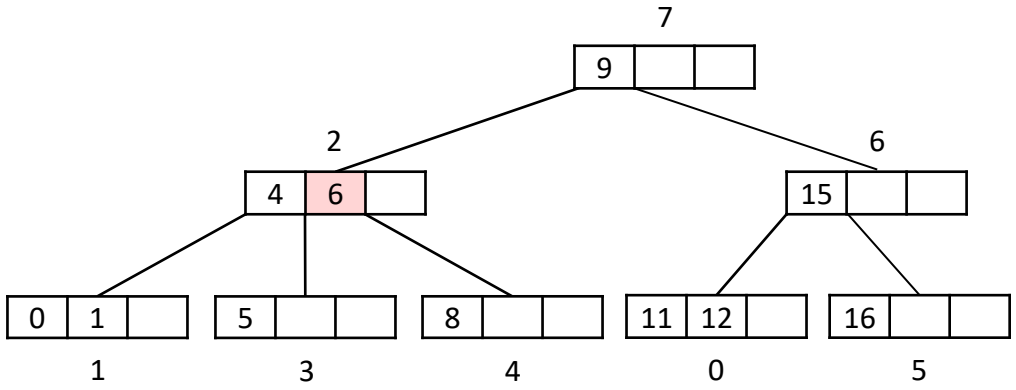


Ejemplo eliminación en árbol B

$k = 3$

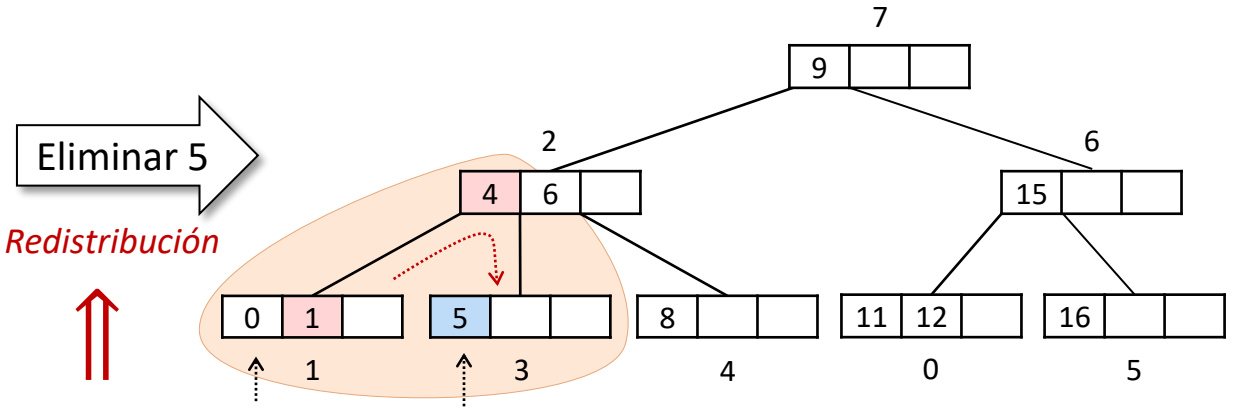


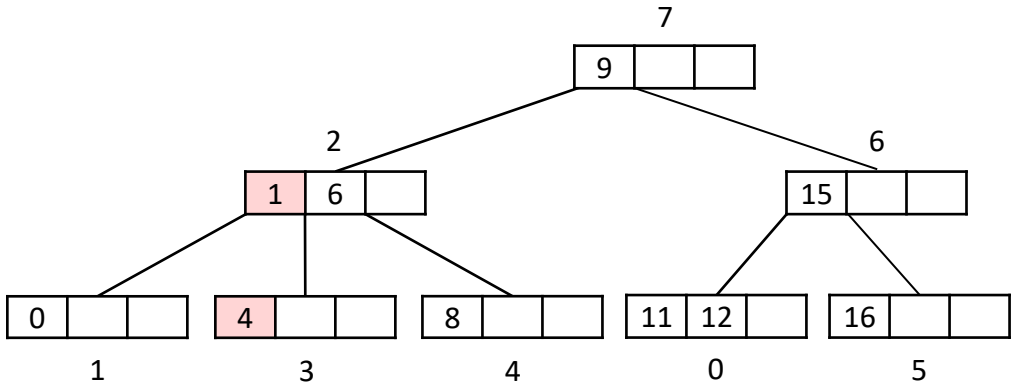
Se intercambia 7 con 6 en la página hoja 3 (o con 8 en la página 4)



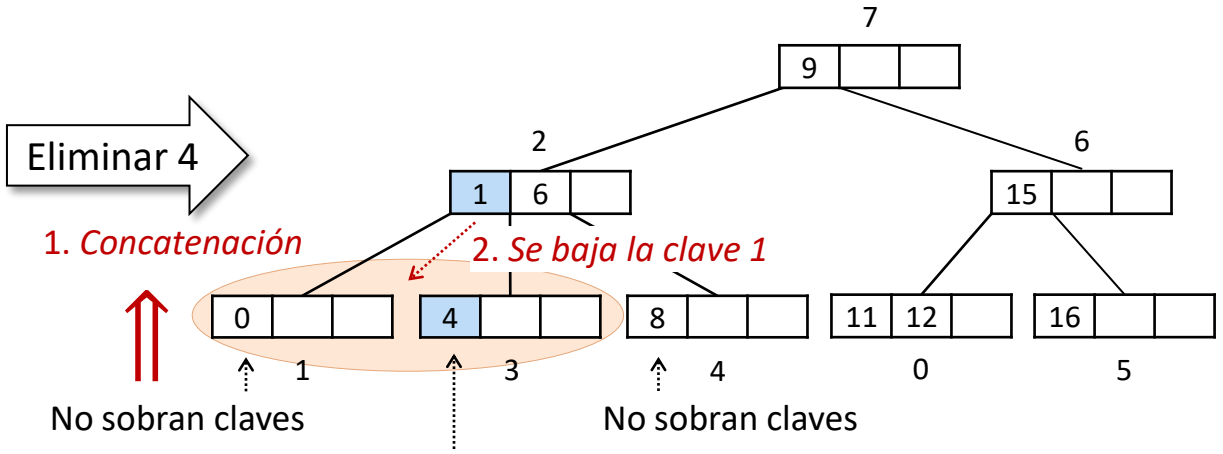
Se elimina 7 de la página 3

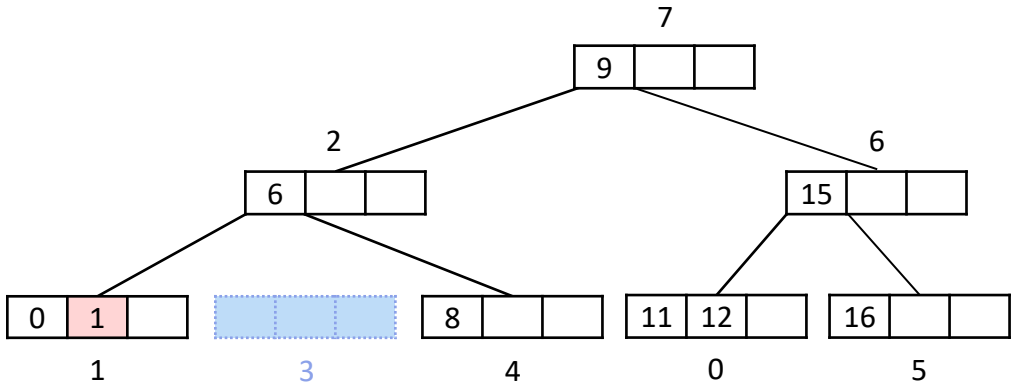
Ejemplo eliminación en árbol B (cont)





Ejemplo eliminación en árbol B (cont)



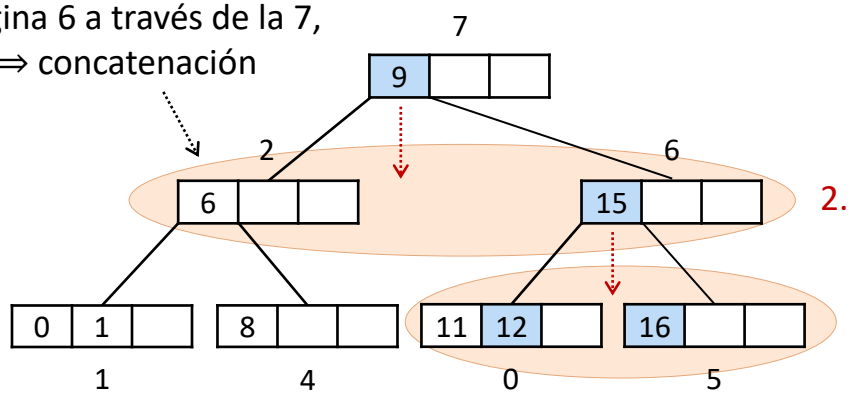


La página 3 queda "borrada"
(se marca como borrada, se recicla, etc.)

Ejemplo eliminación en árbol B (cont)

Si sobrasen claves en la página 2 se re-distribuirían a la página 6 a través de la 7, pero no sobran ⇒ concatenación

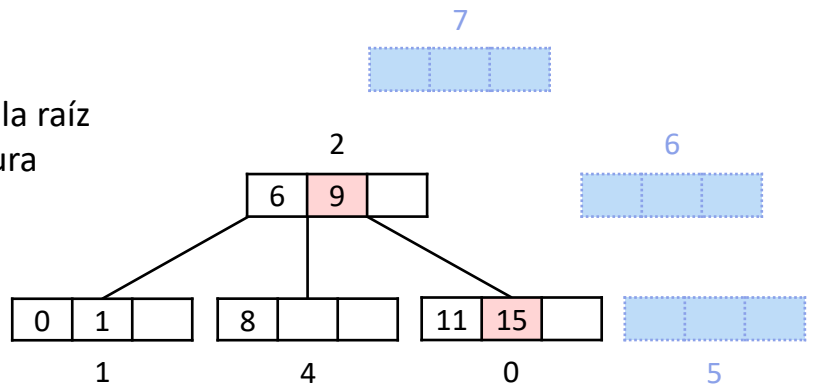
Eliminar 12, 16



2. *Concatenación*

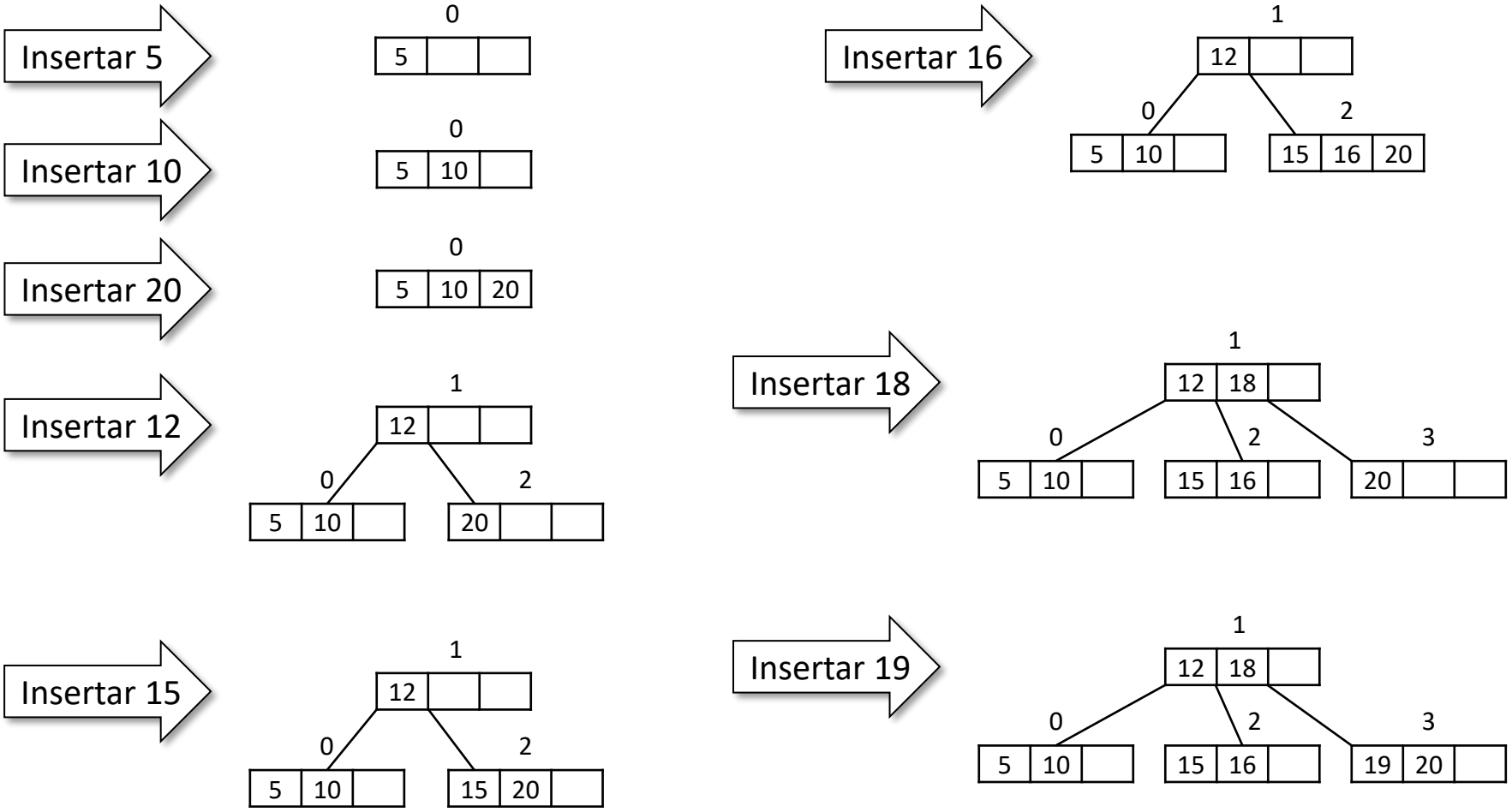
1. *Concatenación*

La concatenación se propaga hasta la raíz y el árbol pierde un nivel de altura



Ejemplo árbol B, orden de inserción

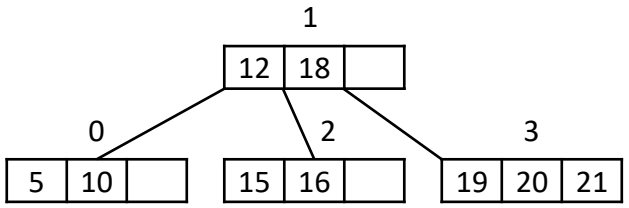
Insertar 5, 10, 20, 12, 15, 16, 18, 19, 21, 22 y 23 (y orden inverso) en k=3



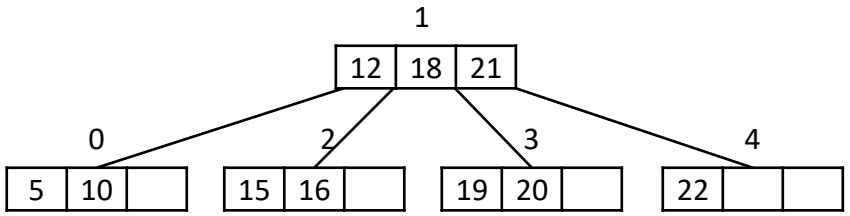
Ejemplo árbol B, orden de inserción

Insertar 5, 10, 20, 12, 15, 16, 18, 19, 21, 22 y 23 (y orden inverso) en k=3

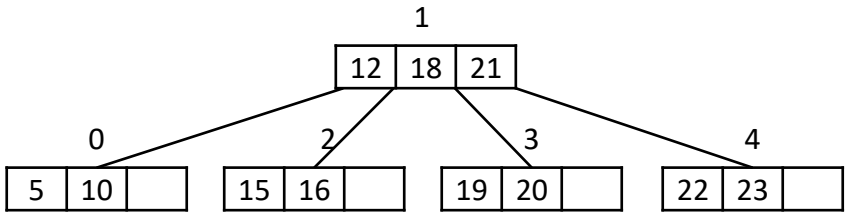
Insertar 21



Insertar 22

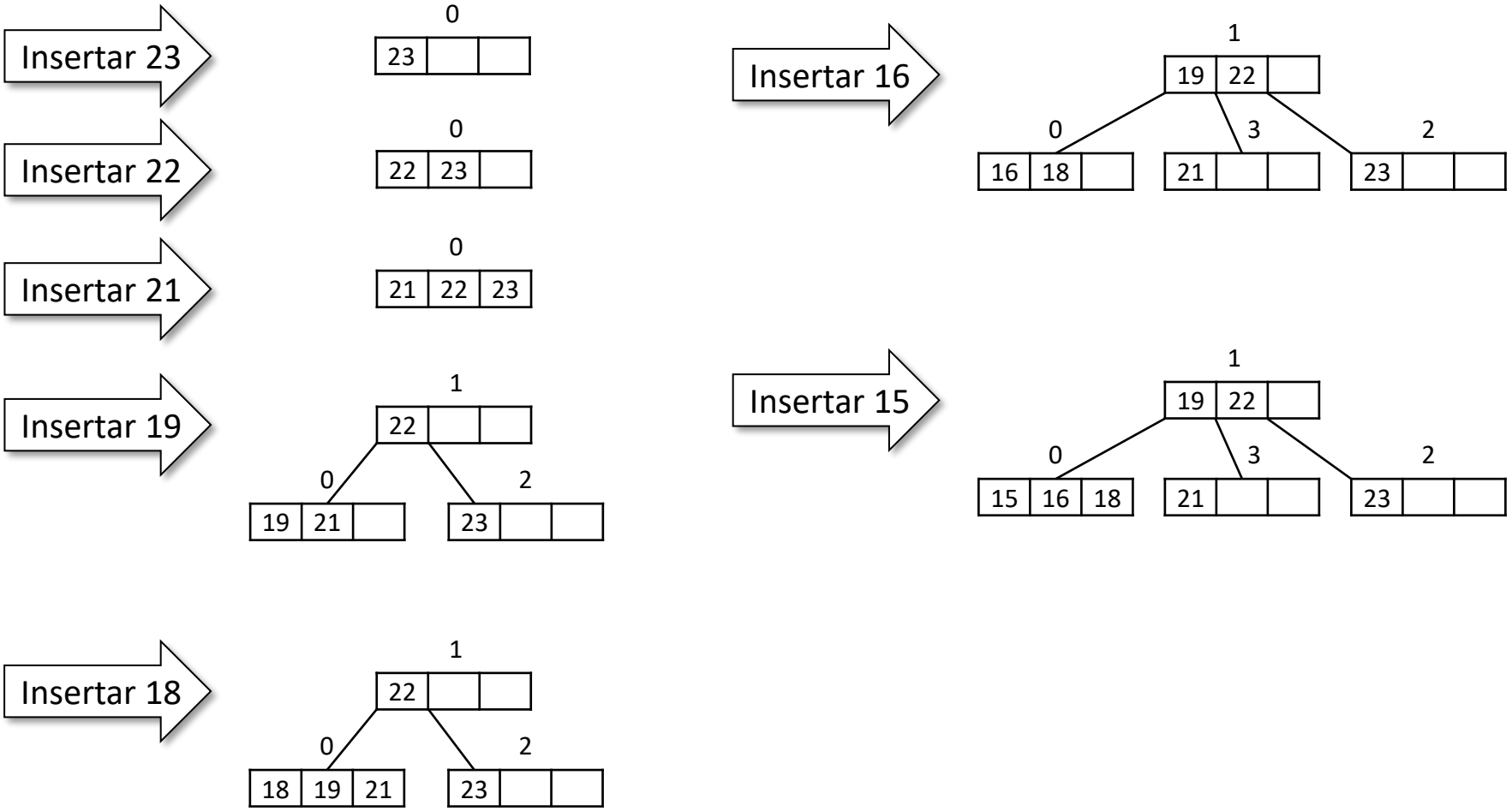


Insertar 23



Ejemplo árbol B, orden de inserción

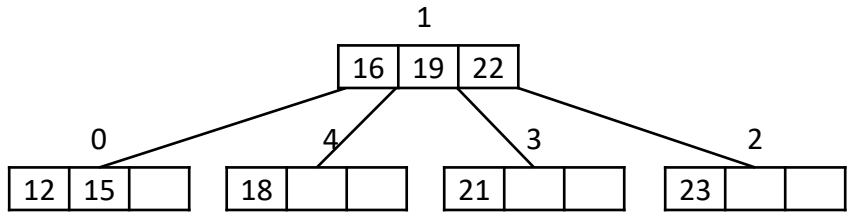
Insertar 23, 22, 21, 19, 18, 16, 15, 12, 20, 10 y 5 en k=3



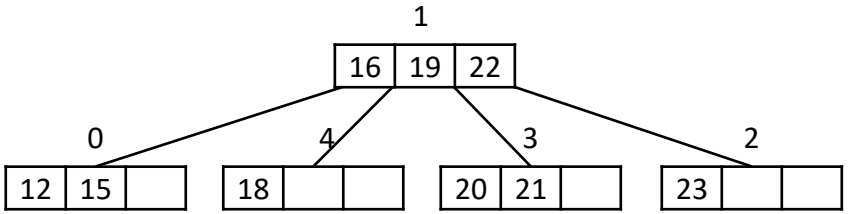
Ejemplo árbol B, orden de inserción

Insertar 23, 22, 21, 19, 18, 16, 15, 12, 20, 10 y 5 en k=3

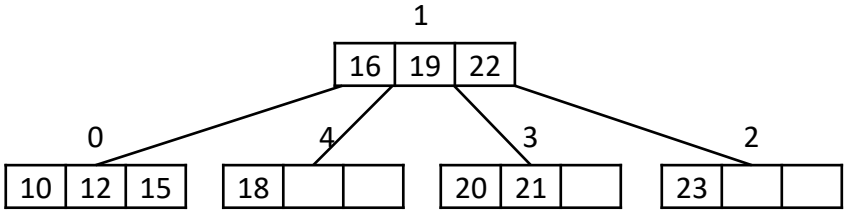
Insertar 12



Insertar 20

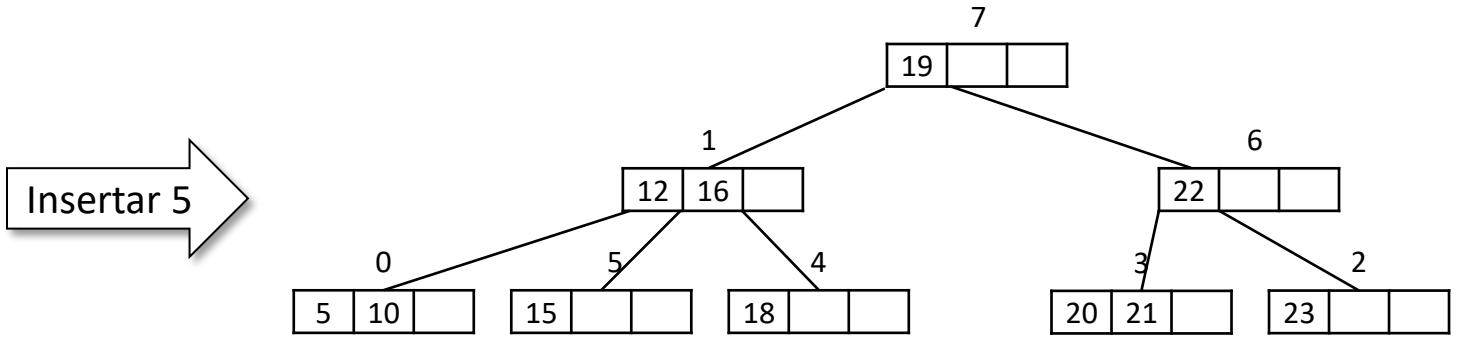


Insertar 10



Ejemplo árbol B, orden de inserción

Insertar 23, 22, 21, 19, 18, 16, 15, 12, 20, 10 y 5 en k=3



Los árboles no tienen la misma profundidad al cambiar el orden de inserción.
No se puede hacer redistribución y esto causa que sean árboles distintos.

Árboles B – Costes

$t = \lfloor k/2 \rfloor + 1$

$m = \lfloor k/2 \rfloor = t - 1$

Peor caso, cada página con el número mínimo de claves, $t - 1 = \lfloor k/2 \rfloor$

	Nivel	hojas (páginas)	claves (mejor caso)
(raíz)	1	1	1
	2	2	2(t-1)
	3	2t	2t(t-1)
	4	$2t^2$	$2t^2$ (t-1)
	⋮		
	h	$2t^{h-2}$	$2t^{h-2}$ (t-1)

Árboles B – Costes

Número de claves es la suma de las claves de todos los niveles del árbol

$$n = 1 + \sum_{i=1}^{h-1} 2(t-1)t^{i-1} =$$

$$= 1 + 2(t-1)\frac{t^{h-1}-1}{t-1} = 2t^{h-1} - 1 \Rightarrow$$

$$h_{max} = 1 + \log_{m+1}\frac{n+1}{2}$$

h_{max} corresponde a la profundidad máxima de un árbol B pues es aquel en donde las hojas tienen el número mínimo de claves

Árboles B – Costes

$t = \lfloor k/2 \rfloor + 1$

$m = \lfloor k/2 \rfloor = t - 1$

Mejor caso, cada página con el número máximo de claves, k

	Nivel	hojas (páginas)	claves (mejor caso)
(raíz)	1	1	k
	2	(k+1)	k(k+1)
	3	$(k+1)^2$	$k(k+1)^2$
	4	$(k+1)^3$	$k(k+1)^3$
		⋮	
	h	$(k+1)^{h-1}$	$k(k+1)^{h-1}$

Árboles B – Costes

Número de claves es la suma de las claves de todos los niveles del árbol

$$n = \sum_{i=0}^{h-1} k(k+1)^i = k\frac{(k+1)^h - 1}{k+1-1} =$$

$$= (k+1)^h - 1 \Rightarrow$$

$$h_{min} = \log_{k+1}(n+1)$$

h_{min} corresponde a la profundidad mínima de un árbol B pues es aquel en donde las hojas tienen el número máximo de claves

Árboles B – Costes

- El coste de la búsqueda y de la inserción es $O(h)$, con $h =$ altura del árbol
- La altura del árbol es mayor cuanto menos llenas estén las páginas
- Si k es el máximo nº de claves por página, m es un mínimo que aseguremos, y n es el nº total de claves en el índice, la altura del árbol es:

$$h_{min} \leq h \leq h_{max}$$

$$\log_{k+1}(n+1) \leq h \leq 1 + \log_{m+1}\frac{n+1}{2}$$

- Por tanto, a mayor m, menor coste en las operaciones
- Con la técnica básica de inserción, aseguramos $m = \lfloor k/2 \rfloor$
- k=512 y n=1.000.000 implica $h \leq 3.37$

Árboles B – Costes

En el ejemplo anterior n=11, k = 3, $m = \lfloor k/2 \rfloor = 1$

$$\log_{k+1}(n+1) \leq h \leq 1 + \log_{m+1}\frac{n+1}{2}$$

$$1.79 \leq h \leq 3.58$$

¿Podemos conseguir mejores mínimos? Vamos a verlo…

Optimizaciones

- Árboles B virtuales
  - Se mantienen *x* páginas en RAM
  - Cuando se necesita una página que ya está en RAM, se evita su coste de lectura
  - Cuando se necesita una página que no está en RAM, ocupa el lugar de otra en RAM
  - Estrategias para decidir de qué página se prescinde: la más baja en el árbol, la usada menos recientemente, estadísticas de acceso…

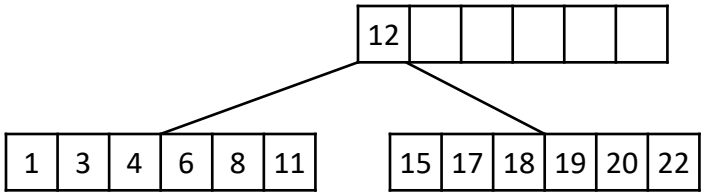
- Árboles B*
  - Inserción con redistribución. En un árbol B la inserción solo tiene división.
  - **Dividir dos páginas a tres**.
  - Cada página excepto la raíz y las hojas tienen al menos 2k/3 descendientes.
  - La raíz tiene al menos 2 descendientes (a menos que sea una hoja).
  - Una página hoja contiene por lo menos $\lfloor 2\,k/3 \rfloor$ claves y no mas de k.
  - Asegurar este mínimo también al eliminar **(concatenar "tres a dos").**
  - Se permite (necesariamente) tener menos claves a la raíz (mínimo 1) y sus hijas (mínimo $\lfloor k/2 \rfloor$ mientras sólo sean dos páginas en ese nivel).

División dos a tres

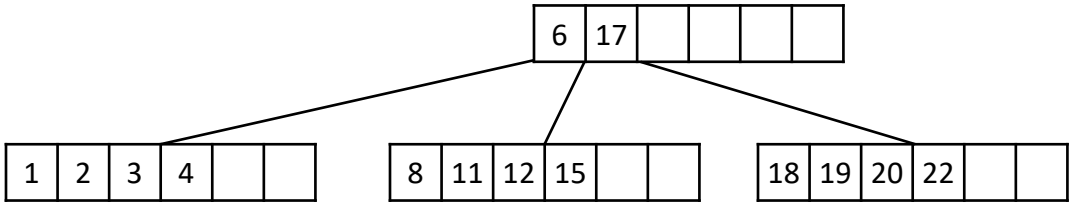
Cuando se llenan dos páginas contiguas partirlas a tres.
Así las páginas a 2/3 en lugar de a 1/2

$k = 6$

$\lfloor 2\,k/3 \rfloor = 4$



Insertar 2 (División de, 6 + 1+ 6 +1 en 4 + 1 + 4 + 1 + 4)



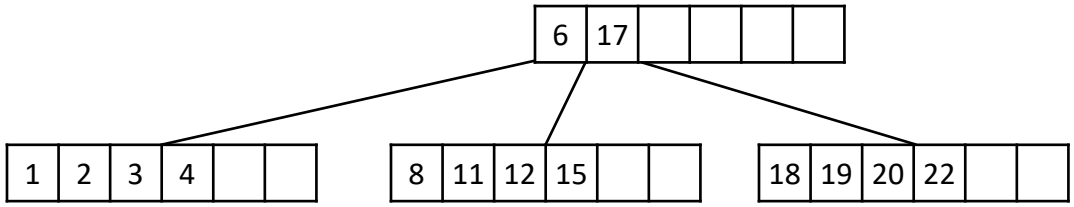
Claves mínimas por hoja $\lfloor 2\,k/3 \rfloor$, 4

Concatenar de tres a dos

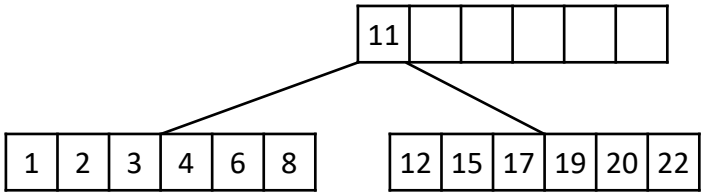
Cuando se llega al número mínimo de claves por hoja

$k = 6$

$\lfloor 2\,k/3 \rfloor = 4$



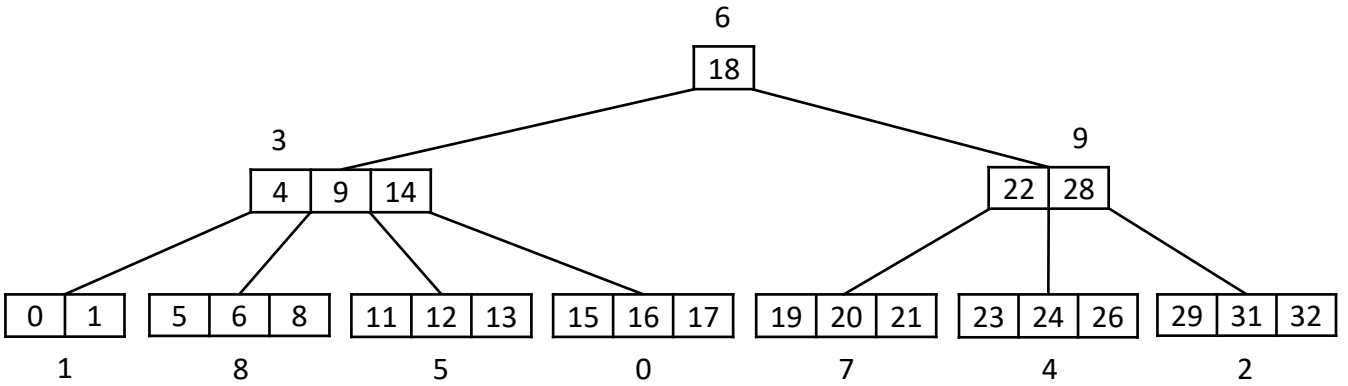
Eliminar 18 (4 + 1 + 4 + 1 + 3 en 6 + 1 + 6)



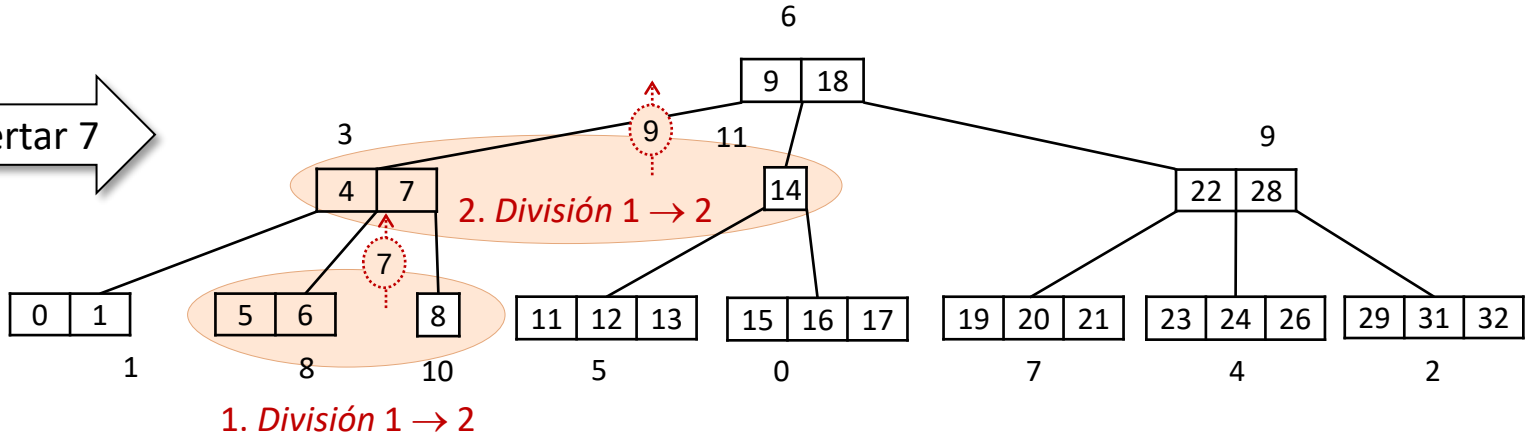
Ejemplo inserción en árbol B (solo división)

$k = 3$

$\lfloor k/2 \rfloor = 1$

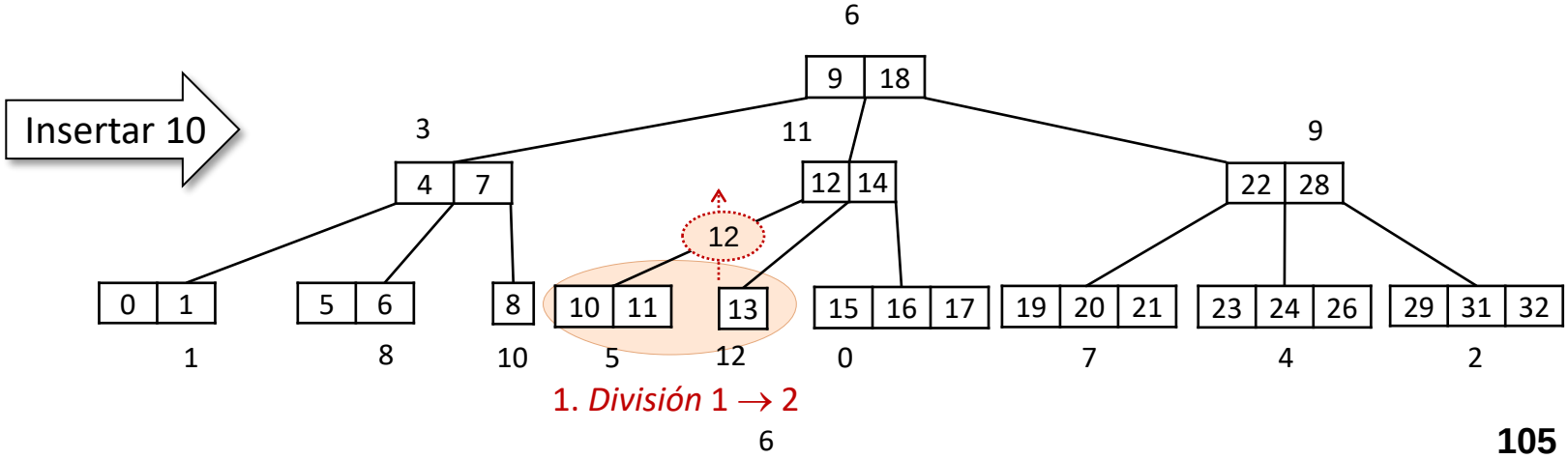


Insertar 7



1. *División* $1 \rightarrow 2$

Insertar 10

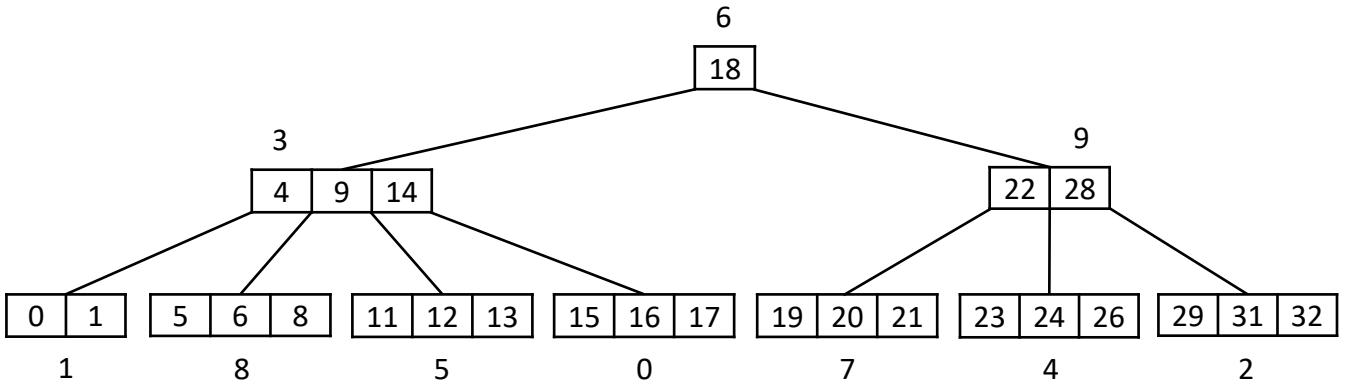


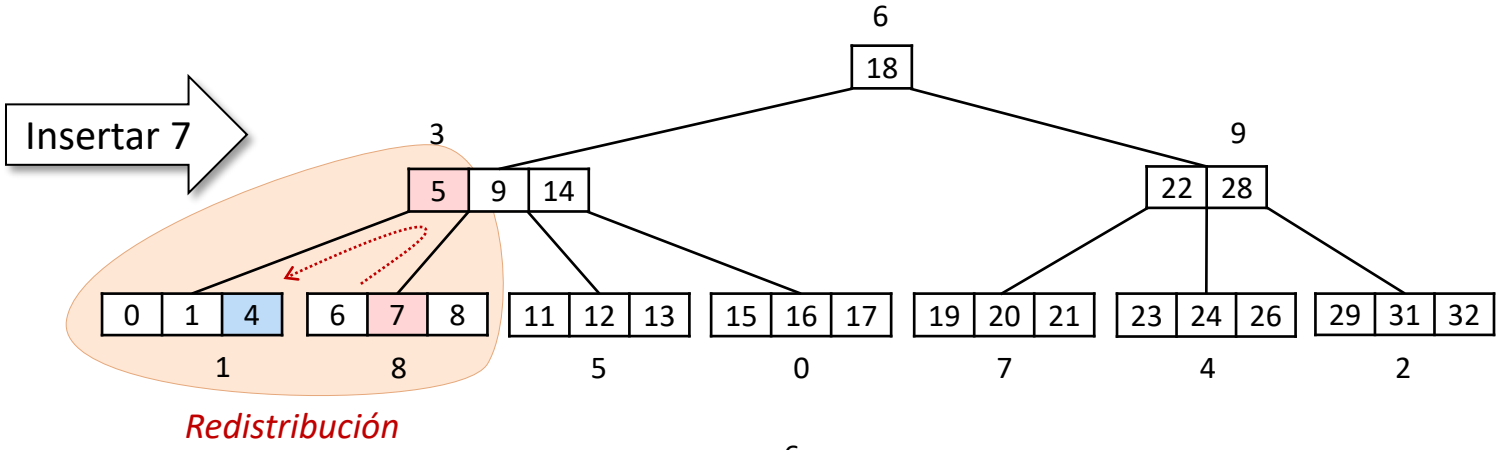
1. *División* $1 \rightarrow 2$

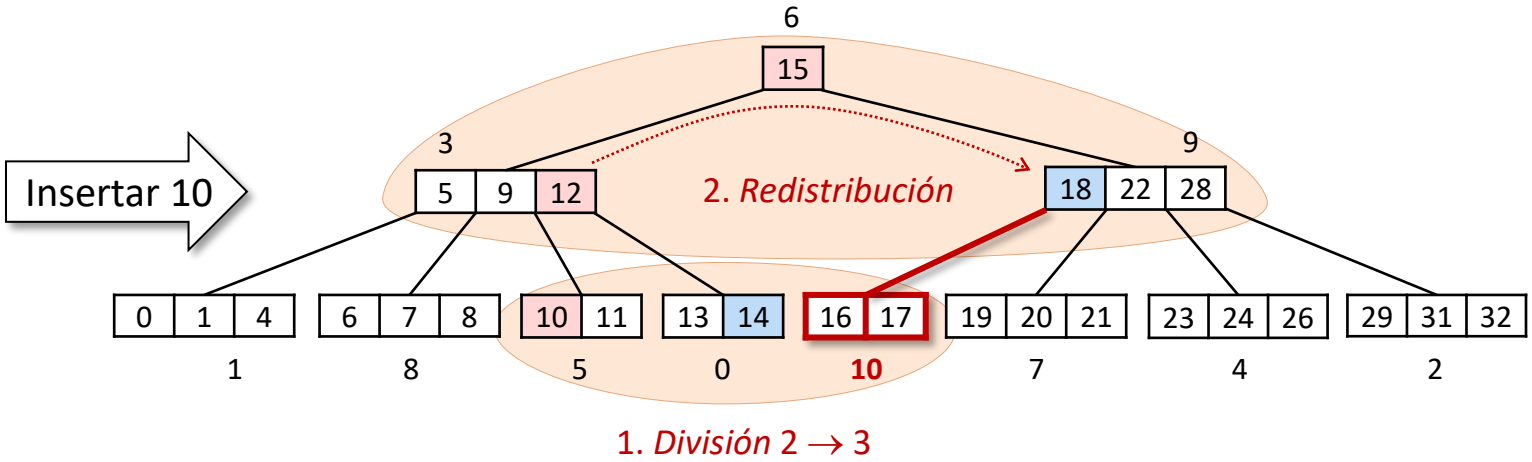
Ejemplo inserción en árbol B*

$k = 3$

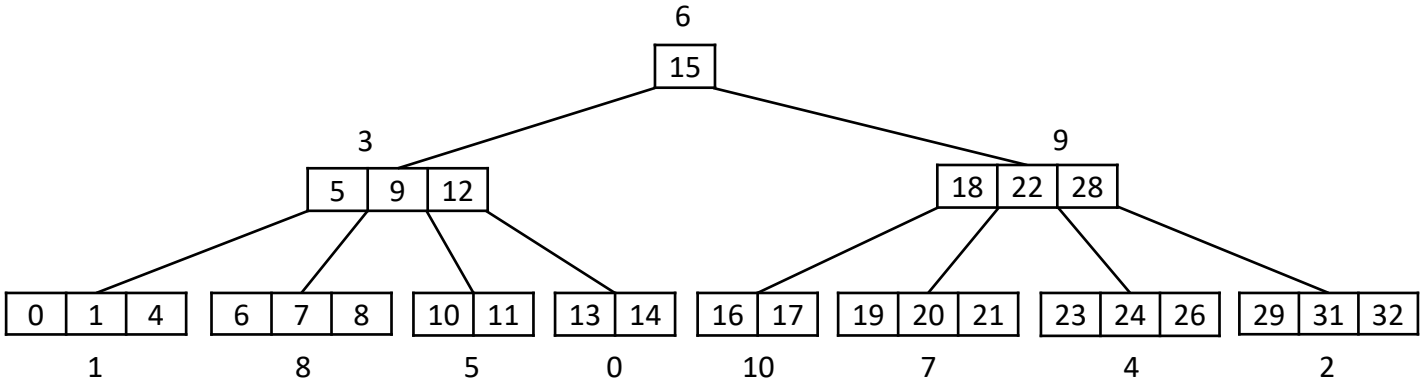
$\lfloor 2k/3 \rfloor = 2$

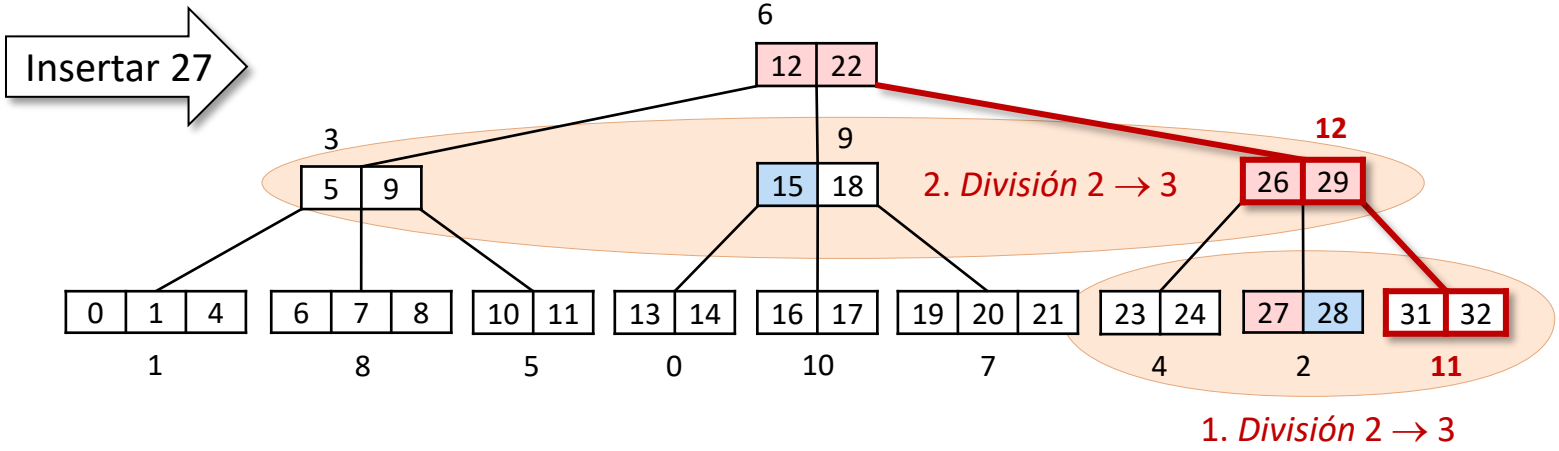






Ejemplo inserción en árbol B* (cont)

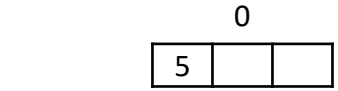




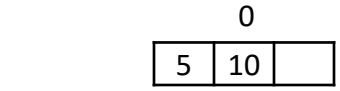
Ejemplo árbol B*, orden de inserción

Insertar 5, 10, 20, 12, 15, 16, 18, 19, 21, 22 y 23 (y orden inverso) en k=3

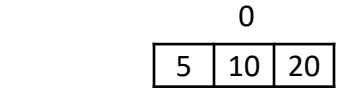
Insertar 5



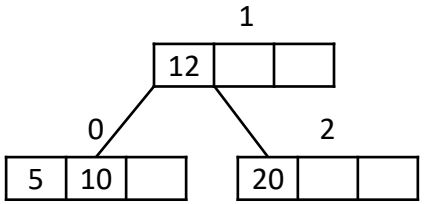
Insertar 10



Insertar 20

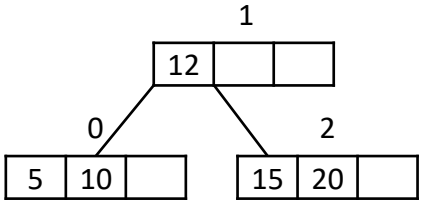


Insertar 12

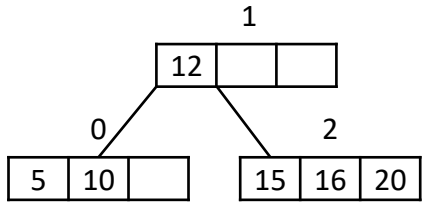


1. Raíz se divide como árbol B
2. Raíz tiene mas claves

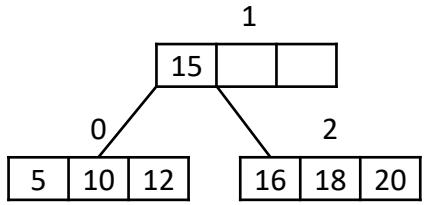
Insertar 15



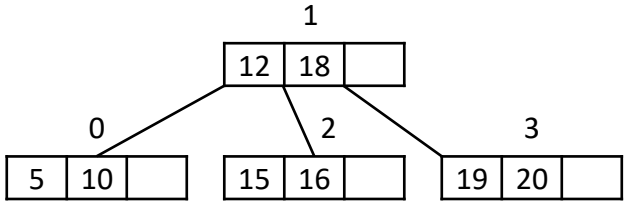
Insertar 16



Insertar 18



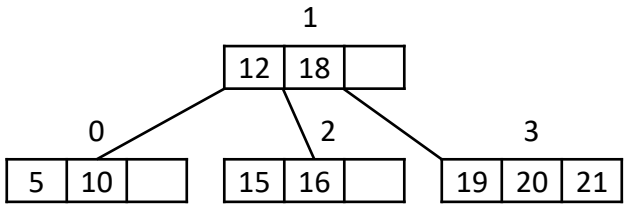
Insertar 19



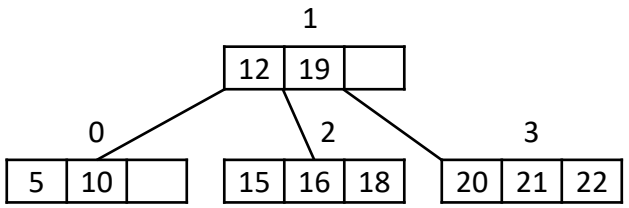
Ejemplo árbol B*, orden de inserción

Insertar 5, 10, 20, 12, 15, 16, 18, 19, 21, 22 y 23 (y orden inverso) en k=3

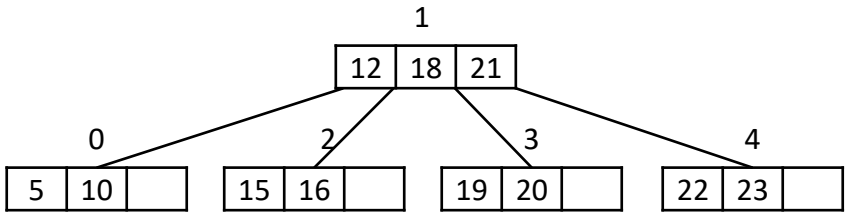
Insertar 21



Insertar 22



Insertar 23

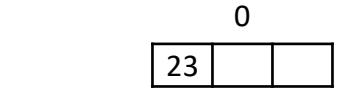


No redistribuye con las no adyacentes

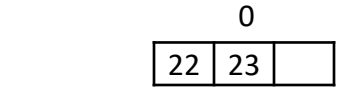
Ejemplo árbol B*, orden de inserción

Insertar 23, 22, 21, 19, 18, 16, 15, 12, 20, 10 y 5 en k=3

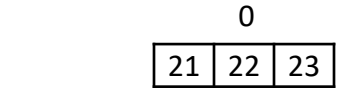
Insertar 23



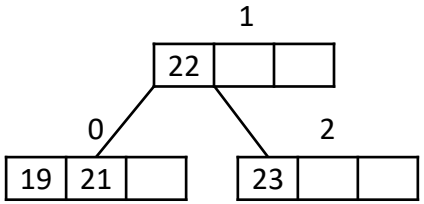
Insertar 22



Insertar 21

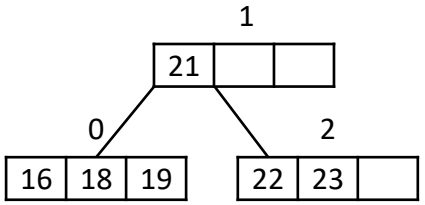


Insertar 19

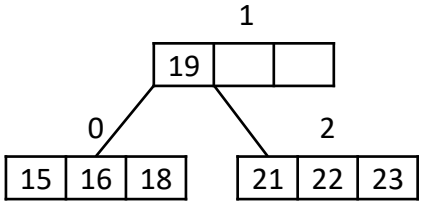


1. Raíz se divide como árbol B
2. Raíz tiene mas claves

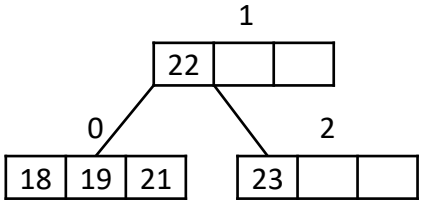
Insertar 16



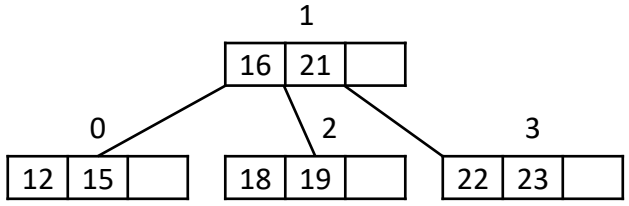
Insertar 15



Insertar 18



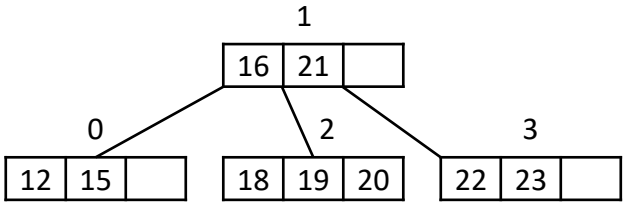
Insertar 12



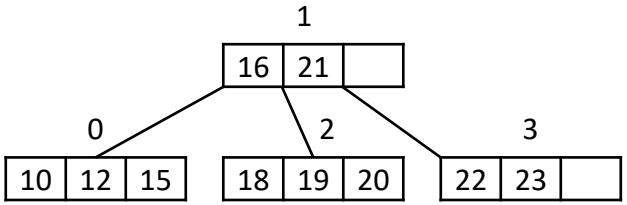
Ejemplo árbol B*, orden de inserción

Insertar 23, 22, 21, 19, 18, 16, 15, 12, 20, 10 y 5 en k=3

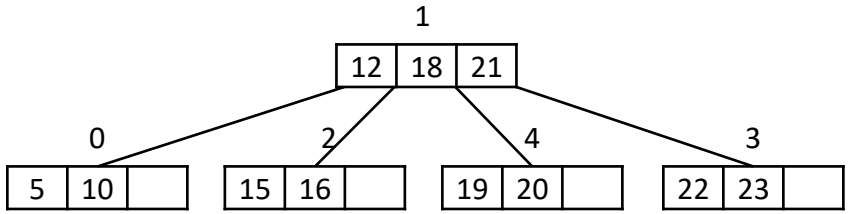
Insertar 20



Insertar 10



Insertar 5

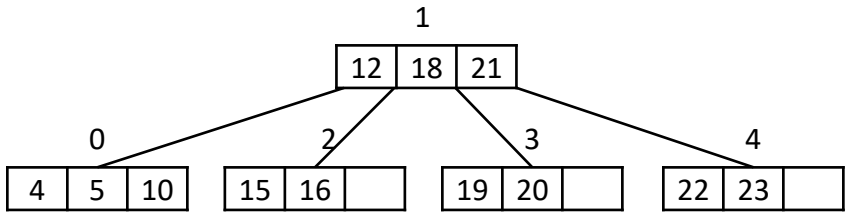


No redistribuye con las no adyacentes

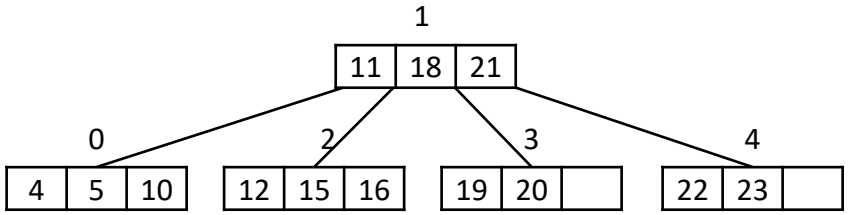
Ejemplo árbol B*, orden de inserción

Insertar 4, 11, 13 y 17

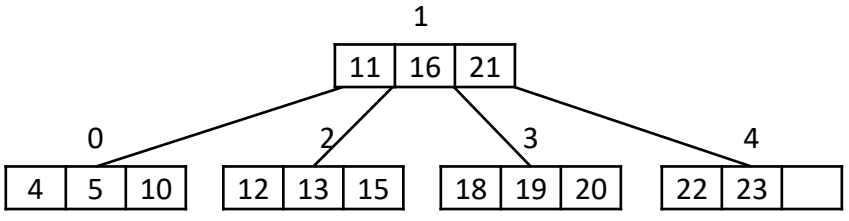
Insertar 4



Insertar 11

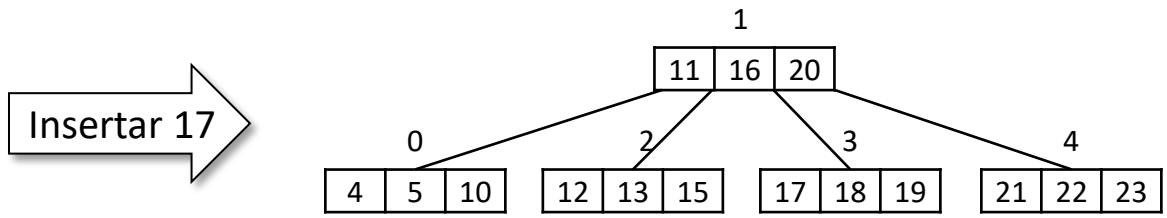


Insertar 13



Ejemplo árbol B*, orden de inserción

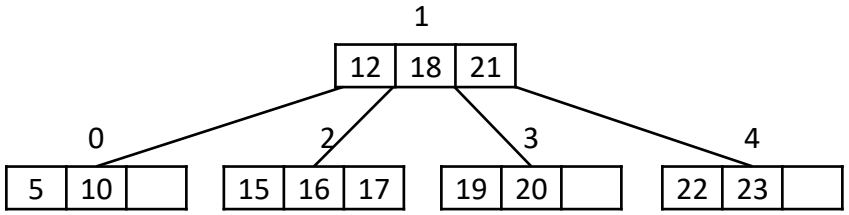
Insertar 4, 11, 13 y 17



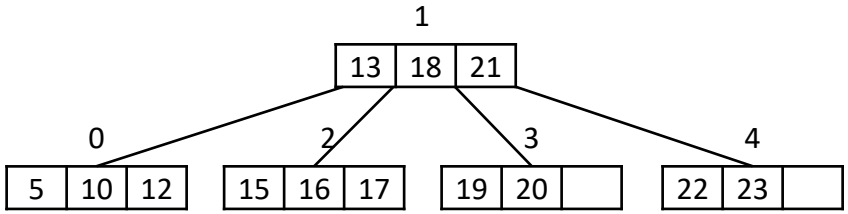
Ejemplo árbol B*, orden de inserción

Insertar 17, 13, 11 y 4

Insertar 17

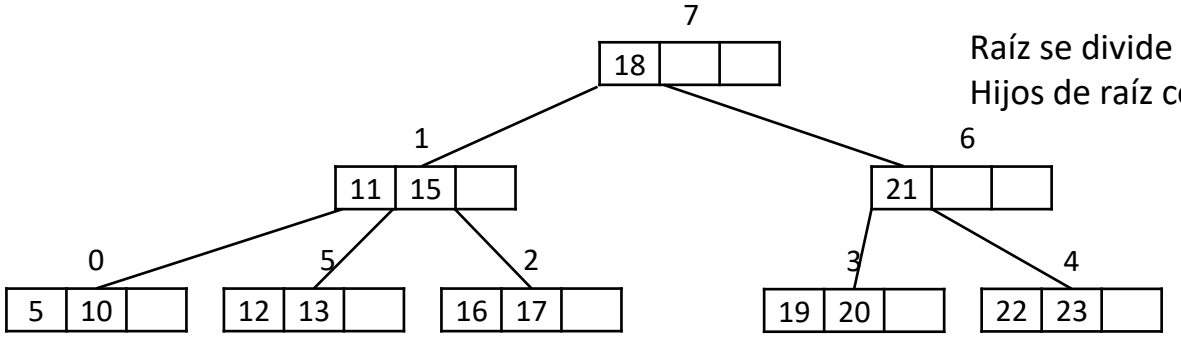


Insertar 13



Se puede redistribuir a izquierda o derecha

Insertar 11

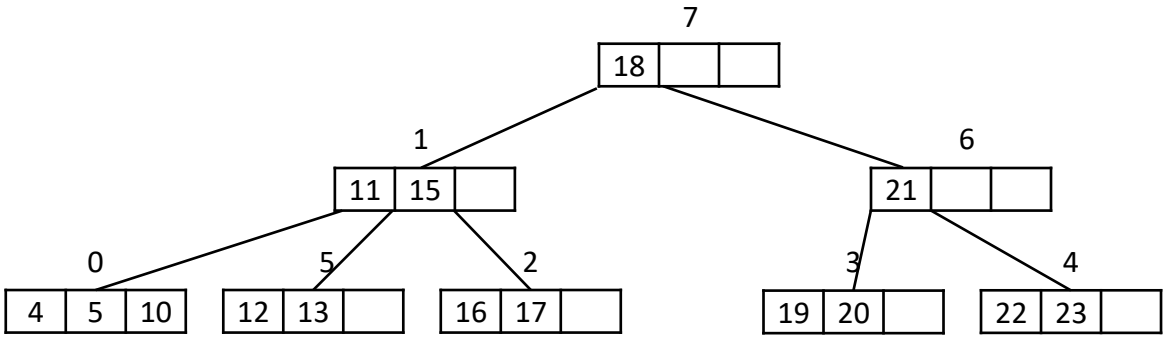


Raíz se divide como árbol B.
Hijos de raíz con menos claves

Ejemplo árbol B*, orden de inserción

Insertar 17, 13, 11 y 4

Insertar 4



Los árboles no tienen la misma profundidad al cambiar el orden de inserción. No se puede hacer redistribución con páginas no adyacentes, se puede redistribuir a izquierda o derecha y esto causa que sean árboles distintos.

Árboles B+: Motivación

VUELO

nrr	Numero	Origen	Destino	Salida
0	345	MAD	CDG	12:30
1	321	MAD	ORY	19:05
2	165	LHR	CDG	09:55
3	127	JFK	CDG	23:55
4	903	CDG	LHR	14:40
5	447	CDG	LHR	17:00
6	204	MAD	CDG	16:15
7	618	MAD	LGW	07:25
8	736	LAX	CDG	11:30

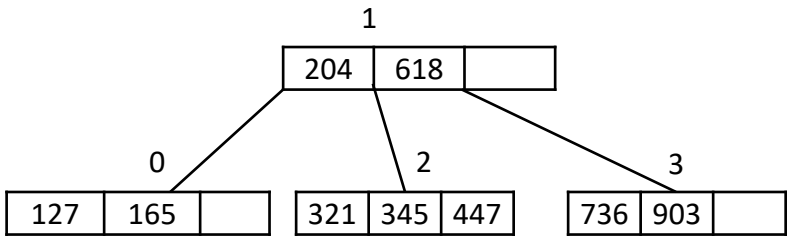
*select * from vuelo order by numero*
*(select * from estudiantes order by nombre)*

- Índice simple: Muy sencillo
- Árbol B: acceso recursivo muy lento, las referencias a los registros están en las páginas

Índice simple

Numero	nrr
127	3
165	2
204	6
321	1
345	0
447	5
618	7
736	8
903	4

Árbol B



Árboles B+: Motivación

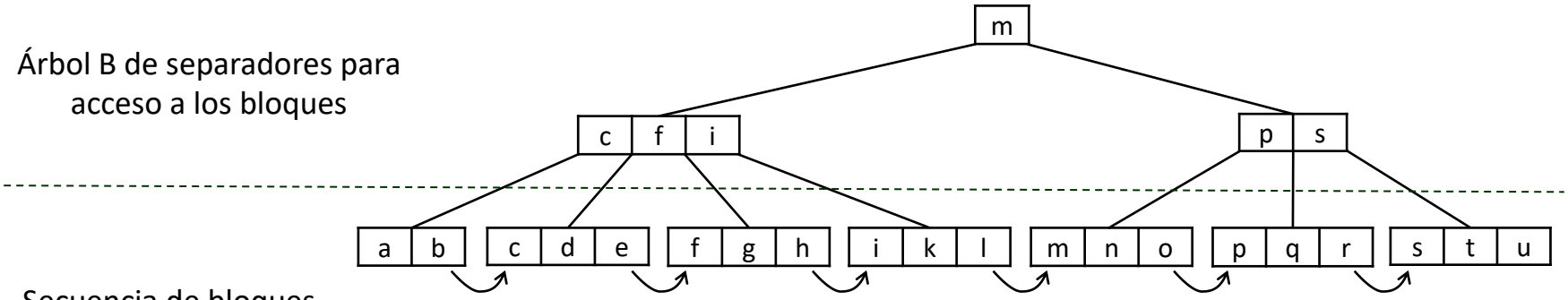
Si se quiere volcar el archivo completo según el orden de la clave:

- hay que recorrer el índice a través del árbol B de forma recursiva
- Supone mucho tiempo de acceso al árbol
- Un índice secuencial accedería a los registros de forma más rápida
- ¿Se pueden combinar ambos mundos?
- Índice dividido en bloques
  - Cada bloque ordenado secuencialmente por la clave.
  - Al insertar o eliminar los bloques se pueden dividir o concatenar.
  - Árbol B para el índice de acceso a los bloques.
  - Cada nodo del árbol B tiene en lugar de un índice el separador para acceder al bloque que está en la hoja.

Ejemplo árbol B+

$k = 3$, Número máximo de claves

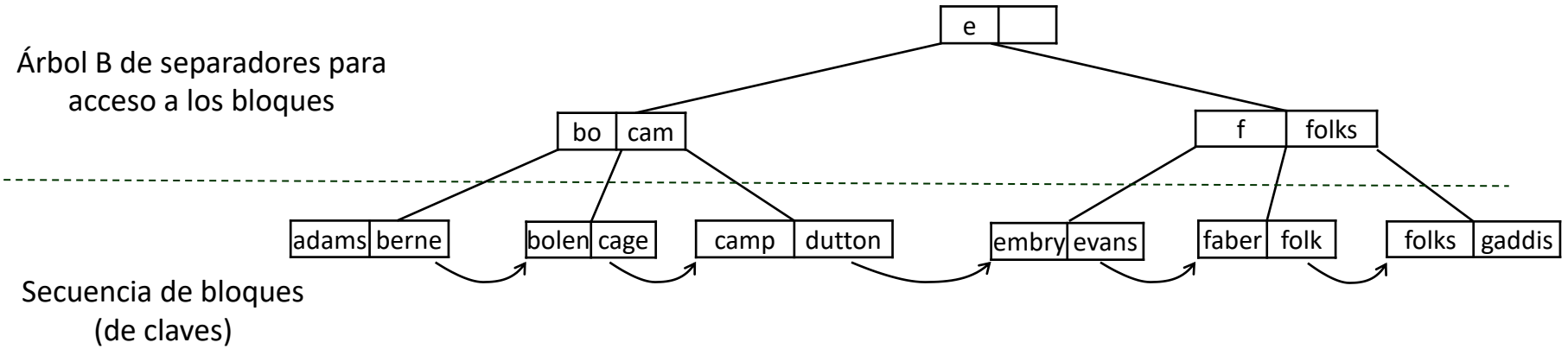
Árbol B de separadores para acceso a los bloques



Secuencia de bloques (de claves)

Ejemplo árbol B+ de prefijos simples

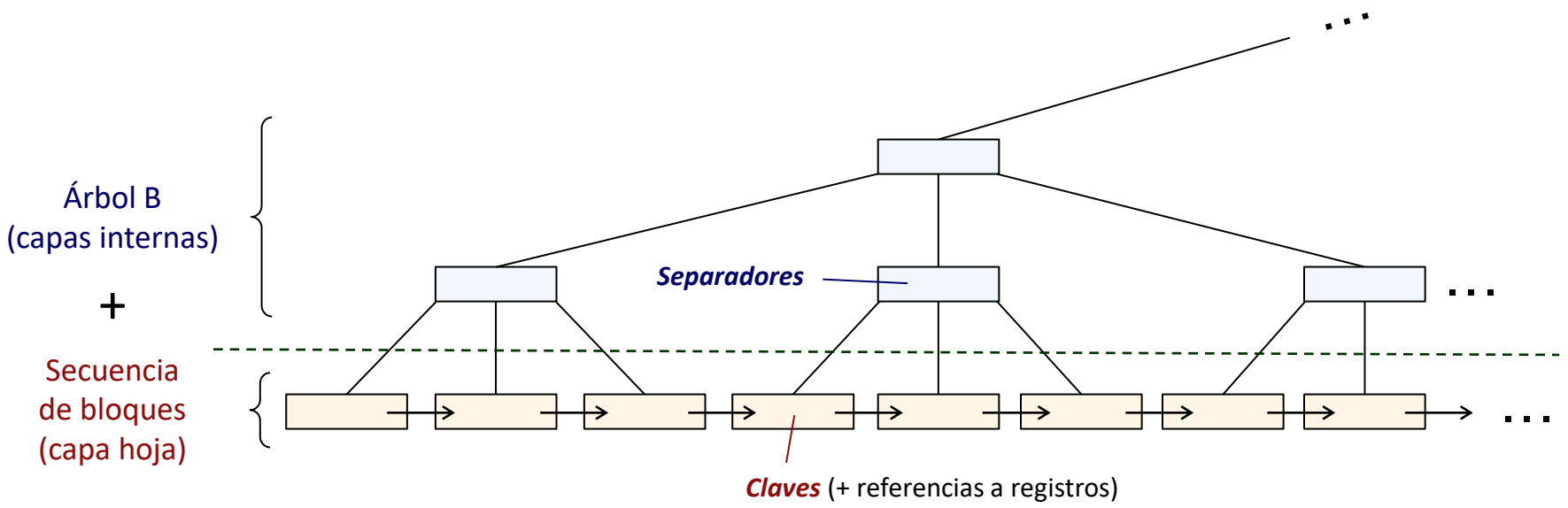
$k = 2$, Número máximo de claves



Árboles B+

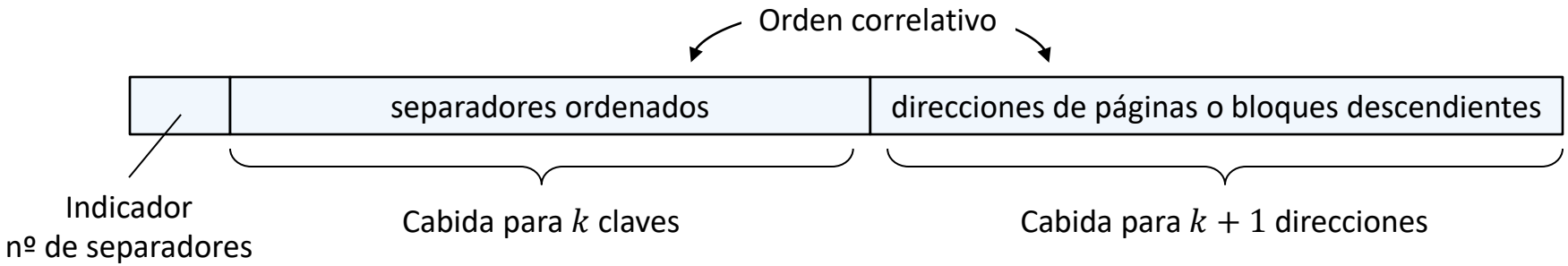
- Las claves (junto dirección física de registros) están todas en las hojas
  - Las hojas contienen referencias a registros y no contienen nºs de páginas hijas
- Las páginas internas contienen separadores
  - No contienen referencia a registros
  - Los separadores pueden ser claves (p.e. la 1ª de la página hoja correspondiente) o prefijos de clave (de longitud mínima para discriminar)
- Ventaja de los árboles B+
  - Dan lugar a un árbol menos profundo: caben más claves por página, pues las páginas internas no almacenan referencias a registros
  - Permiten recorrido secuencial de claves más eficiente que los árboles B básicos (p.e. condiciones de desigualdad $t.a < cte$, ó igualdad $t_1.a = t_2.b$, etc.)
    - Las páginas hoja deben enlazarse cada una con la siguiente, así se pueden recorrer sin subir a las páginas internas

Árboles B+, estructura lógica externa

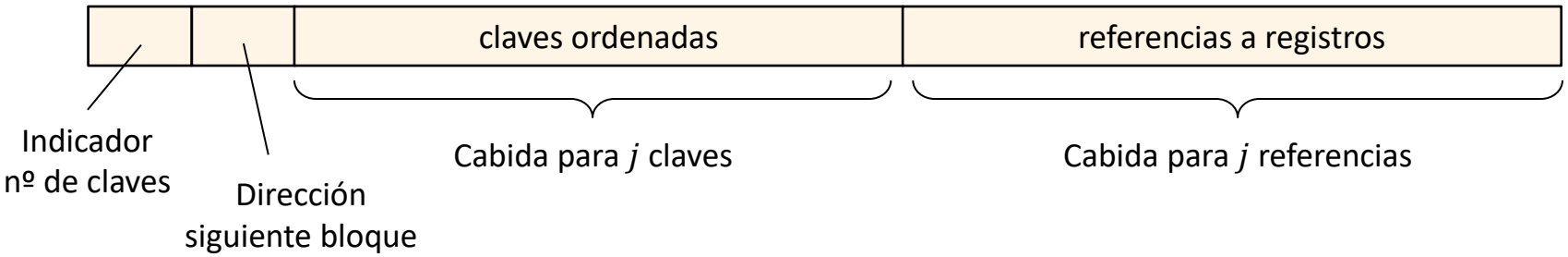


Árboles B+, estructura física interna

- Páginas internas del árbol B+



- Páginas hojas del árbol B+



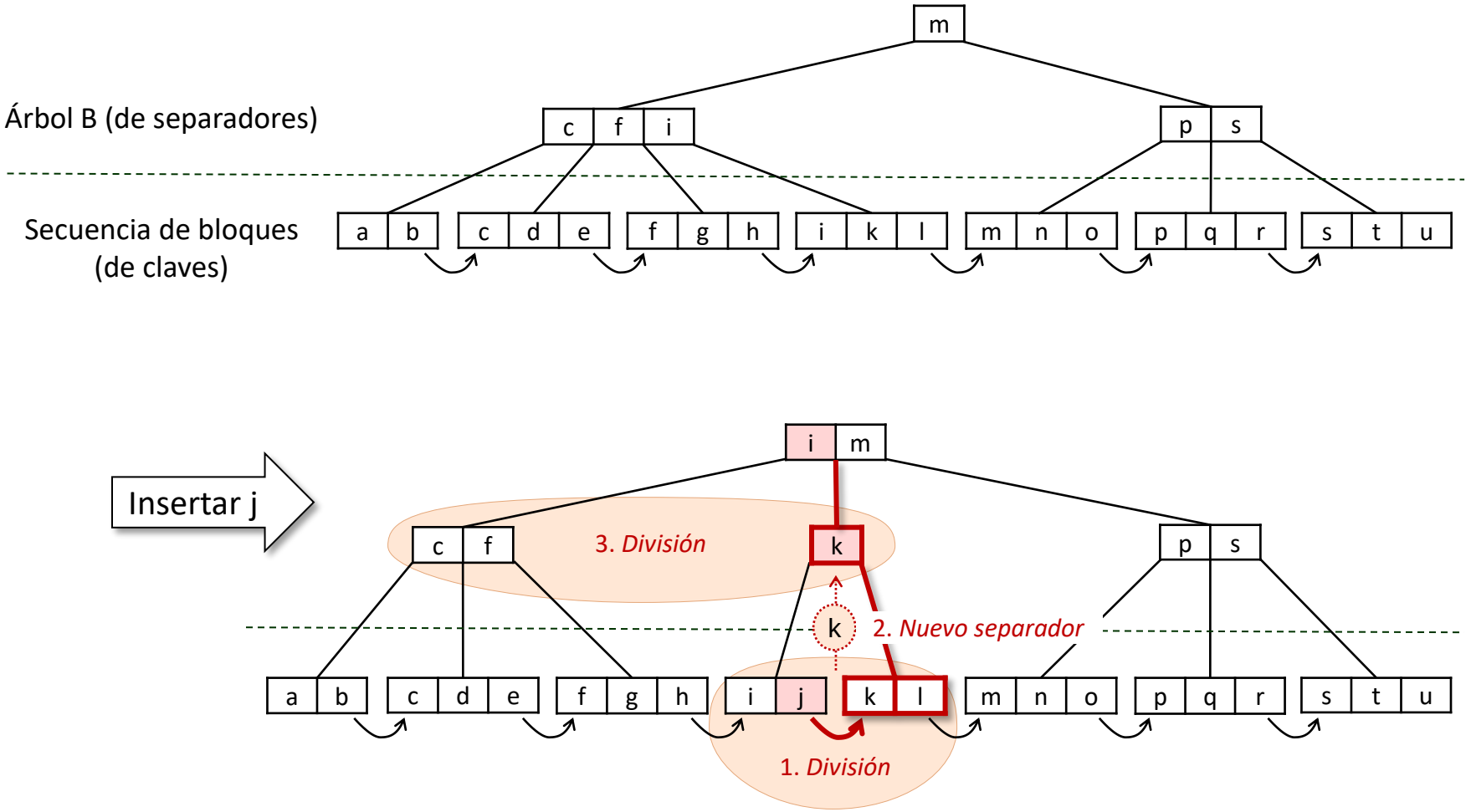
(No tiene por qué ser $j = k$)

Árboles B+, Operaciones

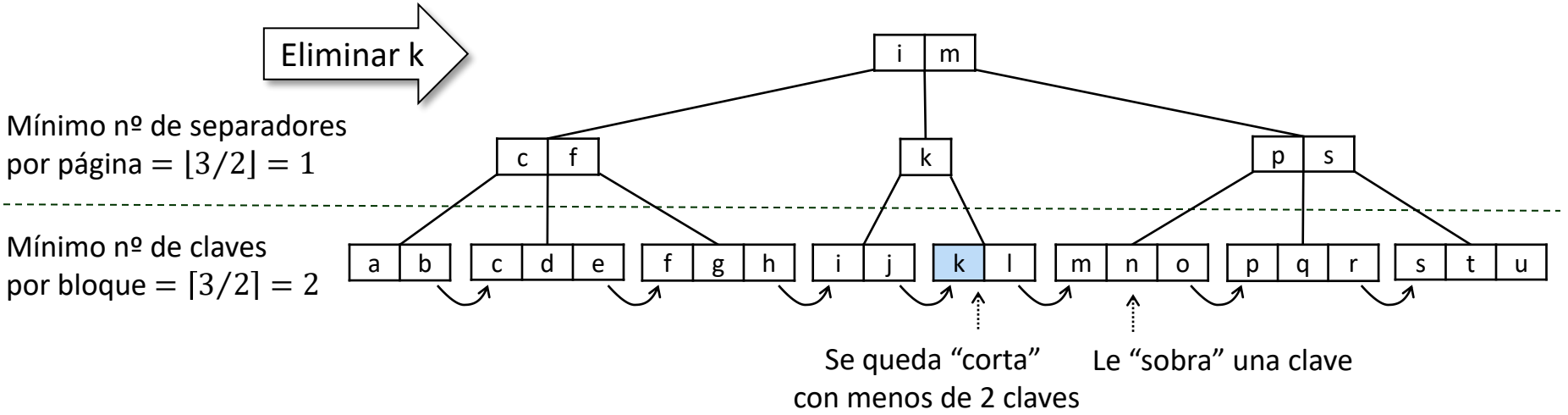
- Búsqueda
  - Similar a los árboles B, pero no se detiene hasta llegar a las hojas
- Inserción
  - Cuando se divide una hoja, se crea un nuevo separador que se inserta en el nivel superior del árbol ($\Rightarrow m = \lceil k/2 \rceil$, número mínimo de claves en las hojas)
  - No hay redistribución en la inserción
- Eliminación
  - Cuando se redistribuyen claves en las hojas, se sustituye un separador por otro nuevo (delete + insert) en el árbol B
  - Cuando se concatenan hojas, se elimina el separador del árbol B (pero no se incluye en la hoja)
  - La concatenación y redistribución se pueden hacer con hojas consecutivas, aunque no desciendan de la misma página (se buscan los separadores en el árbol)

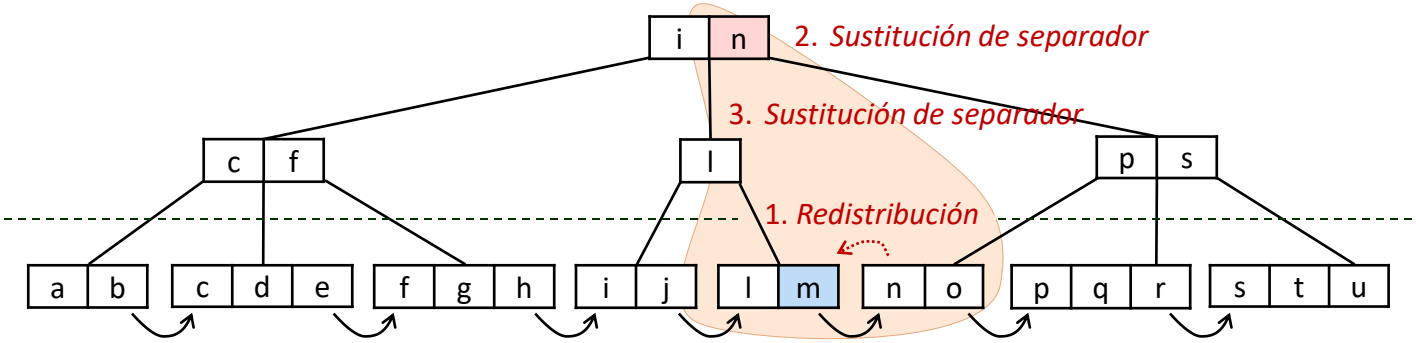
Ejemplo inserción en árbol B+

$k = 3$, Número máximo de claves

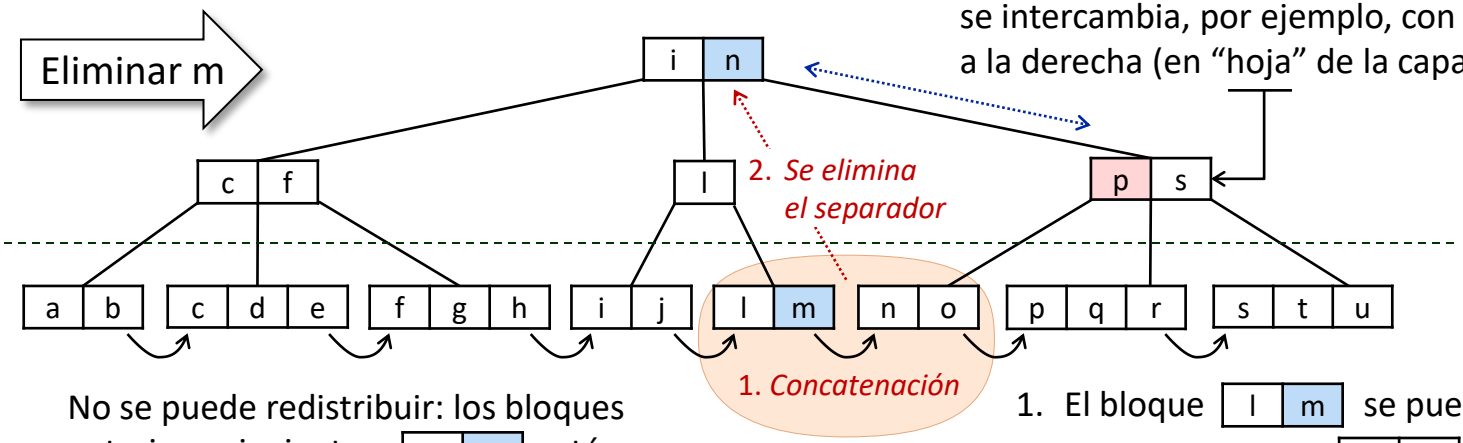


Ejemplo eliminación en árbol B+ (cont)





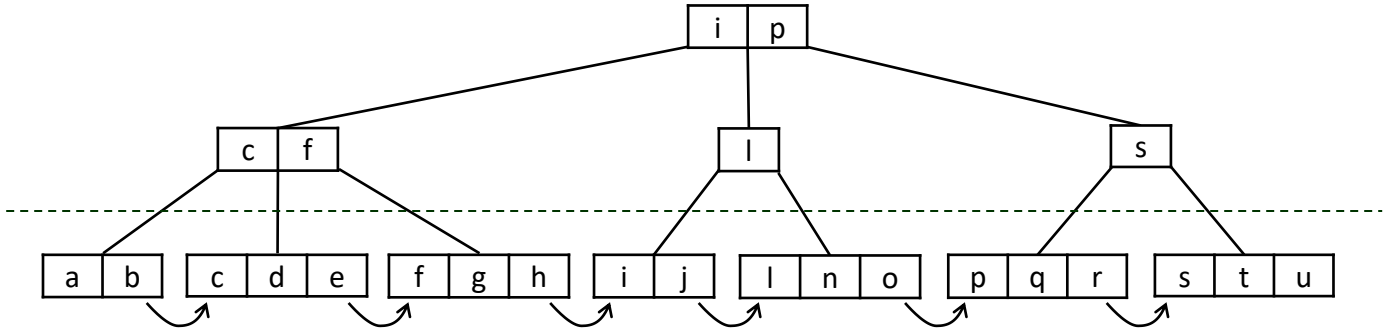
Ejemplo eliminación en árbol B+ (cont)



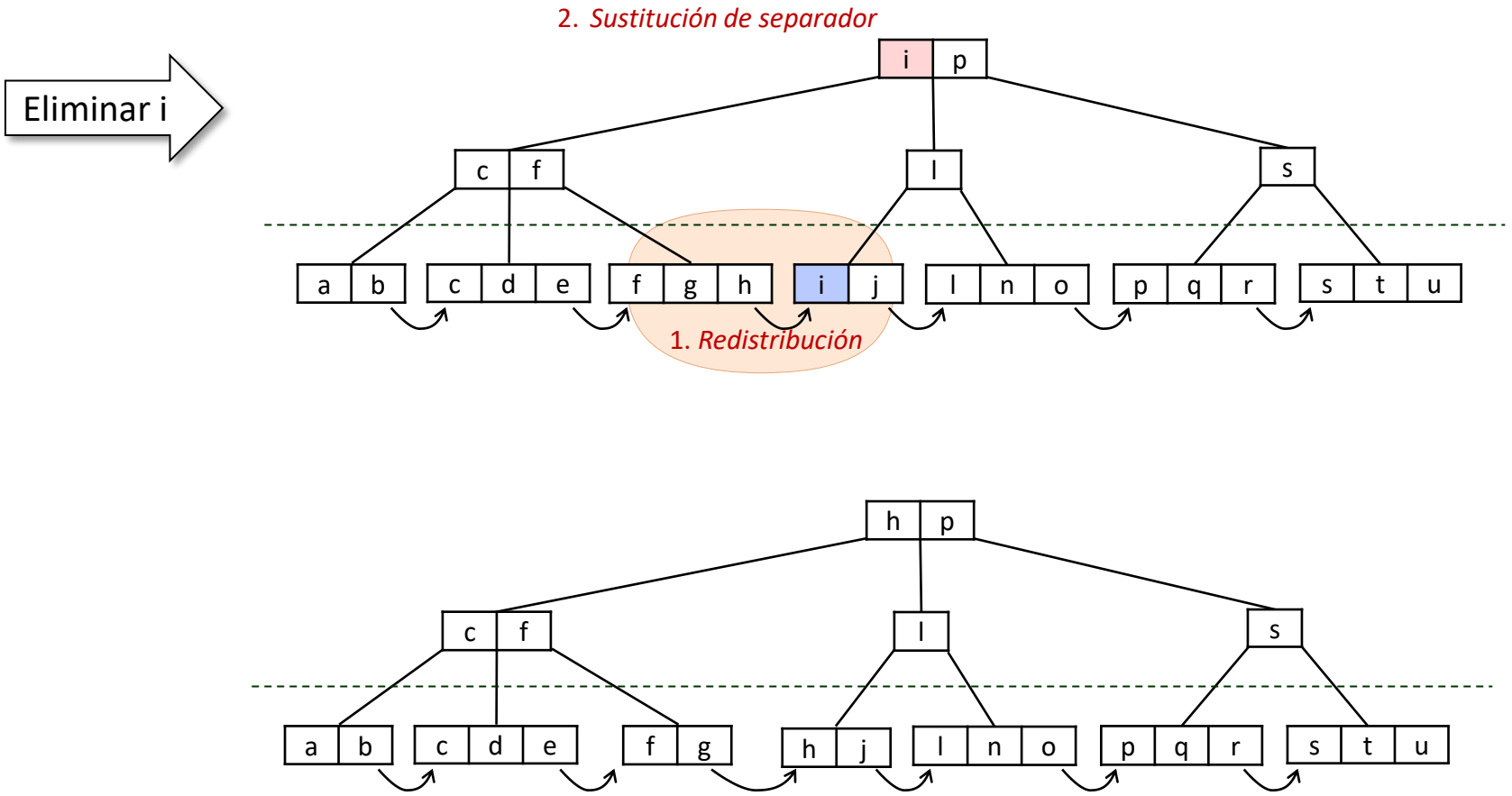
2. El separador n se elimina como árbol B estándar: se intercambia, por ejemplo, con el separador p a la derecha (en "hoja" de la capa árbol B), etc.

No se puede redistribuir: los bloques anterior y siguiente a l m están al mínimo

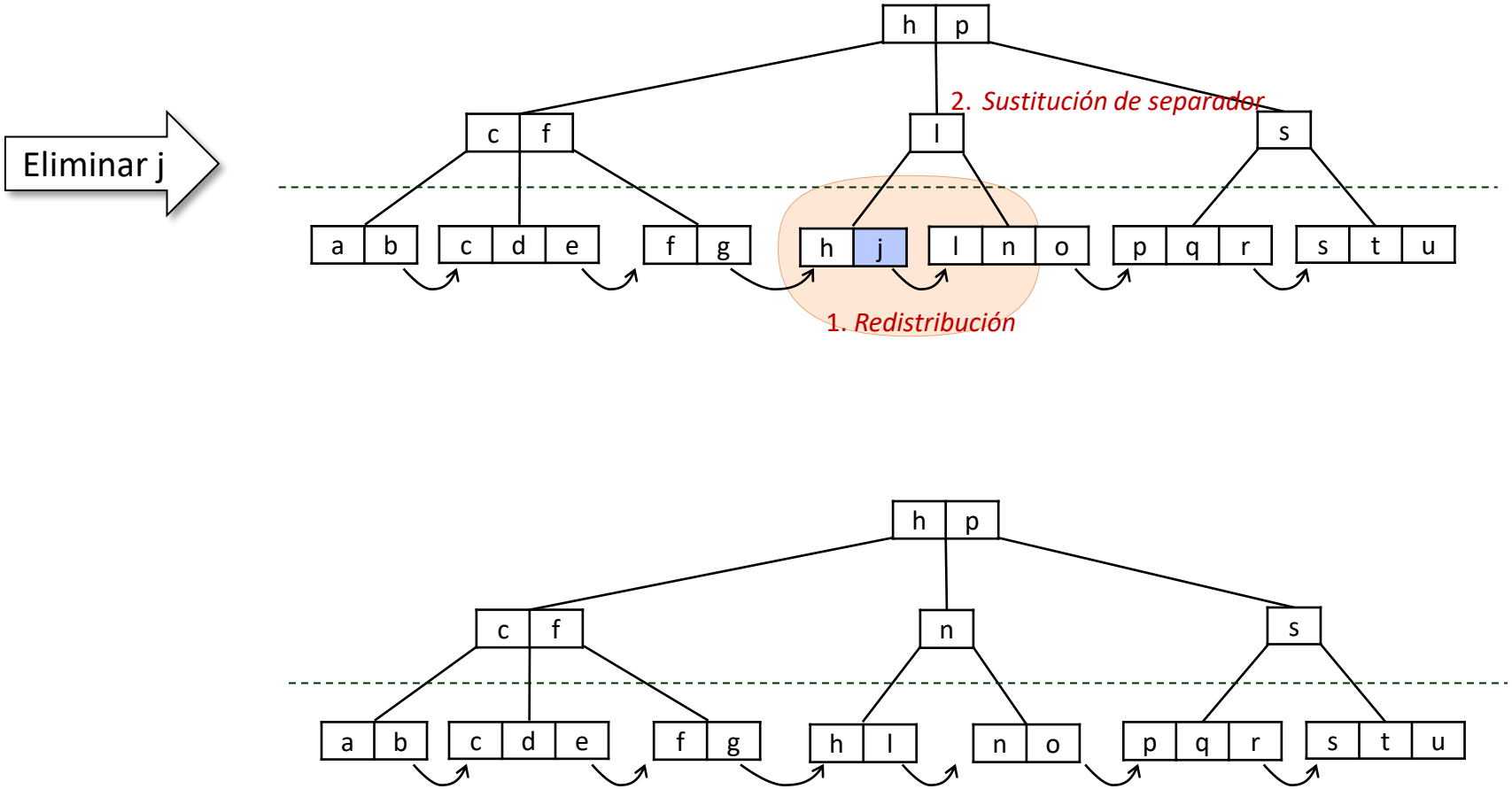
1. El bloque l m se puede concatenar a la izquierda con i j o a la derecha con n o



Ejemplo eliminación en árbol B+ (cont)

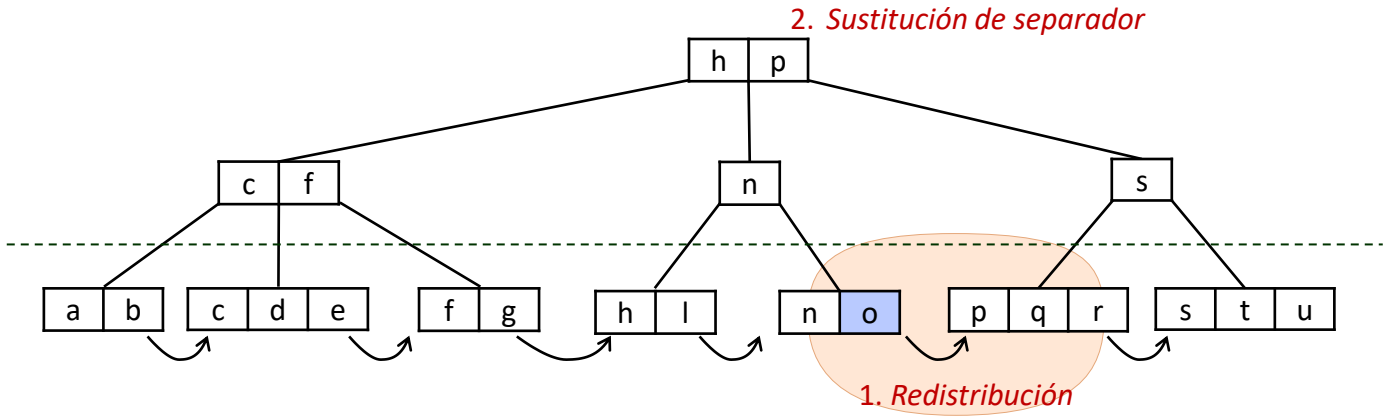


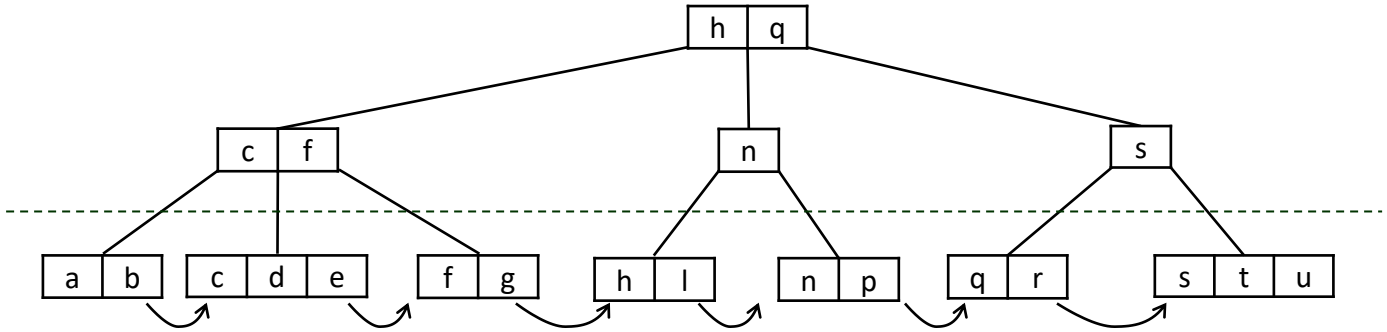
Ejemplo eliminación en árbol B+ (cont)



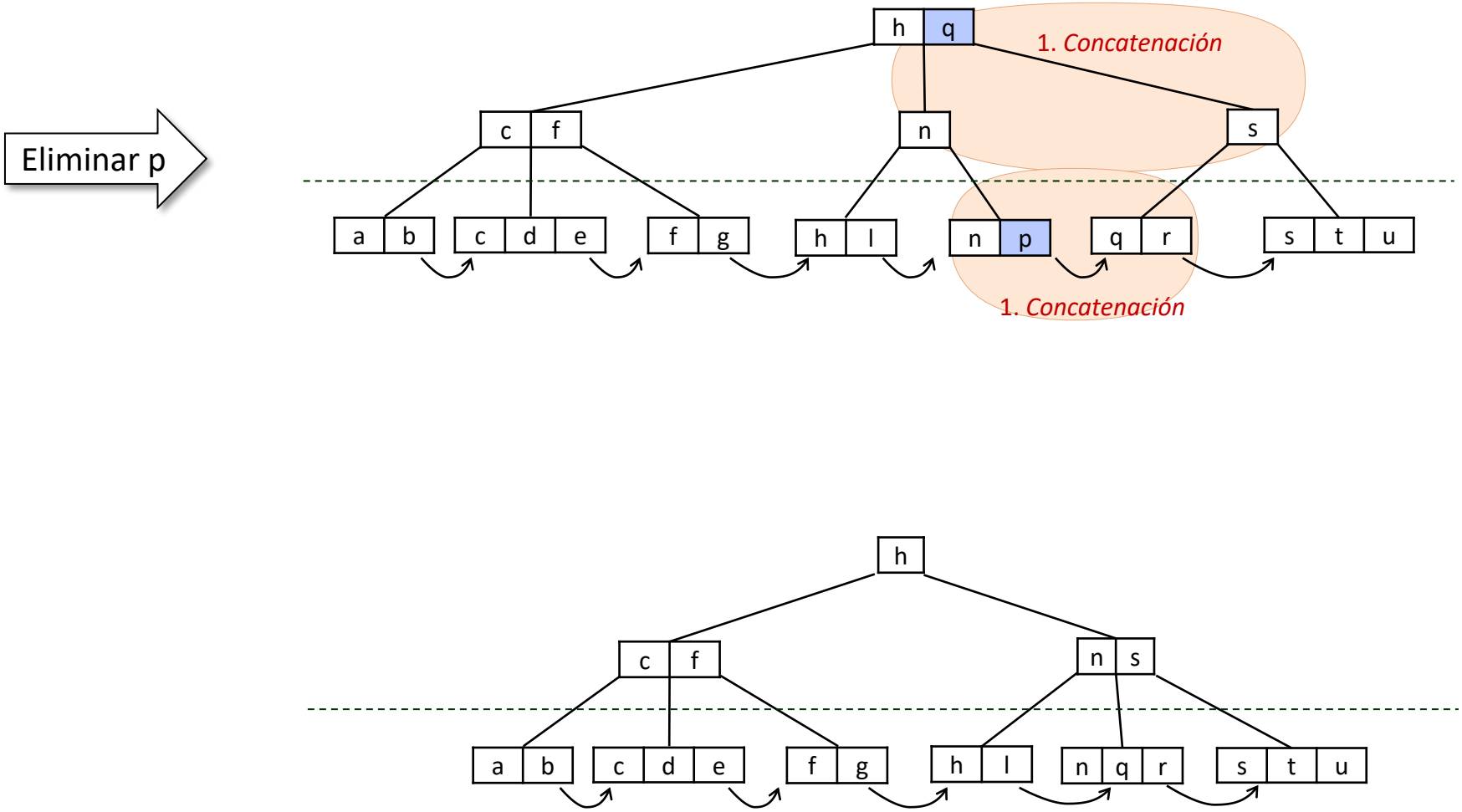
Ejemplo eliminación en árbol B+ (cont)

Eliminar o

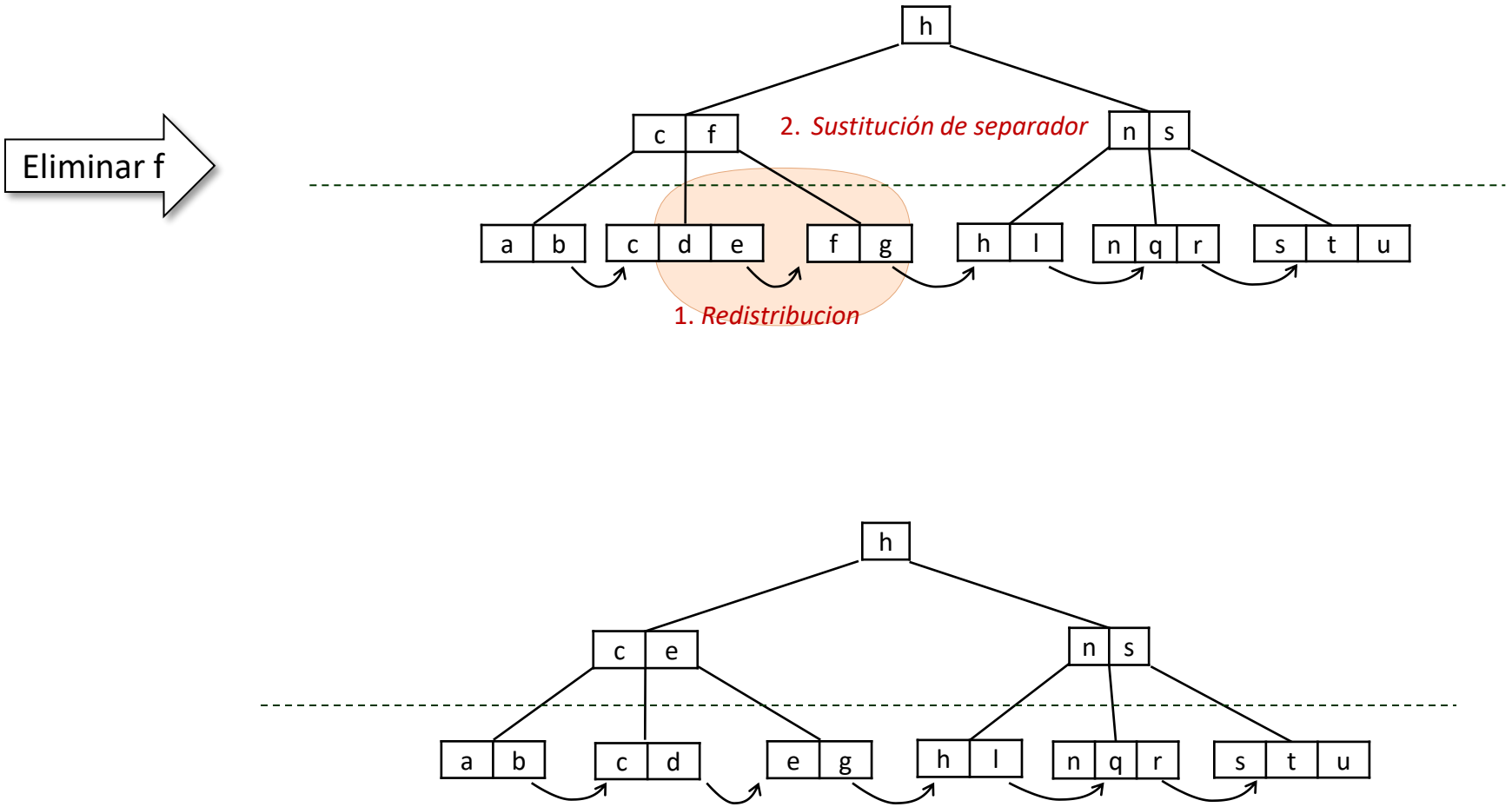




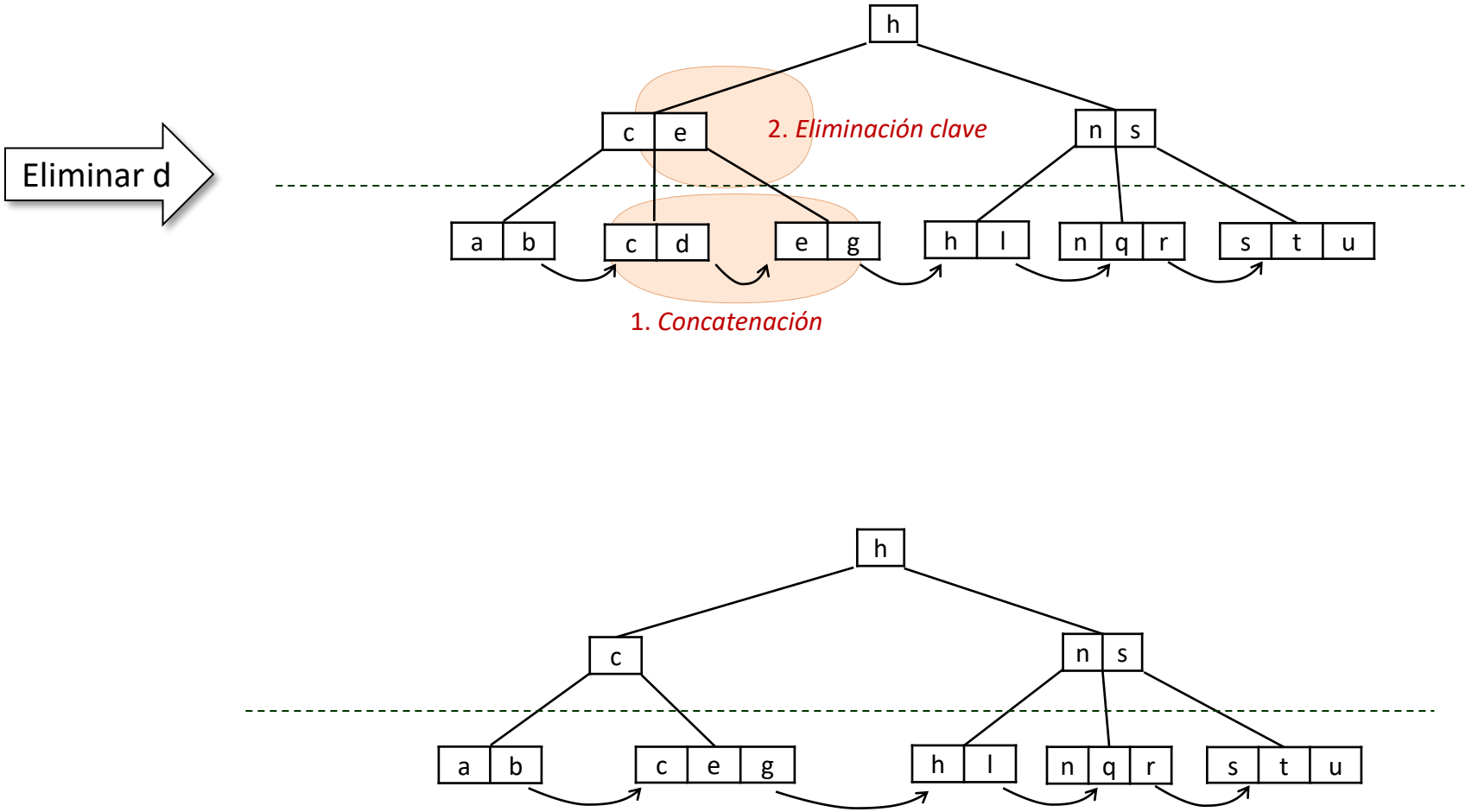
Ejemplo eliminación en árbol B+ (cont)



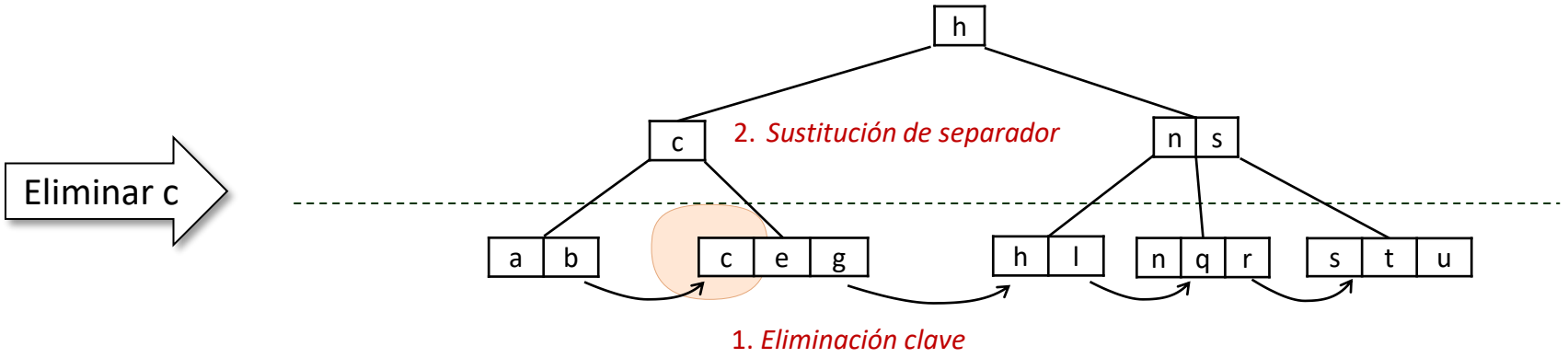
Ejemplo eliminación en árbol B+ (cont)

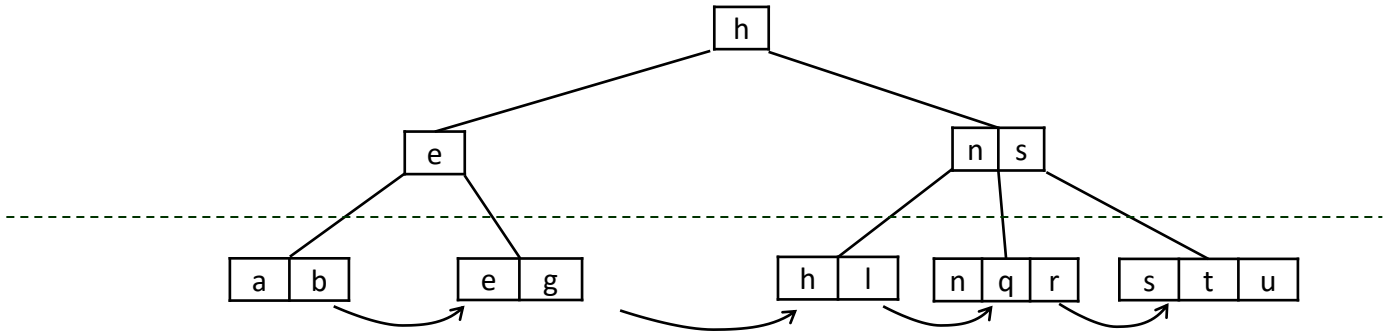


Ejemplo eliminación en árbol B+ (cont)



Ejemplo eliminación en árbol B+ (cont)





Ejemplo eliminación en árbol B+ (cont)