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
    - Hashing

Implementación de un SGBD

Implementación de un SGBD

Usuario final

Interfaz de usuario

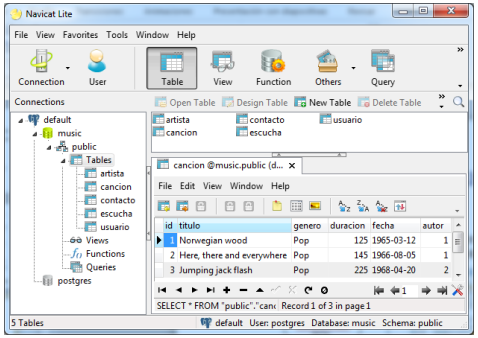
Programador aplicación

Software aplicación

Administrador

Lógica de la aplicación

Entorno / herramientas de administración



pgAdmin
Navicat
SQLYog
Workbench
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 $\rightarrow$ "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, hashing

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

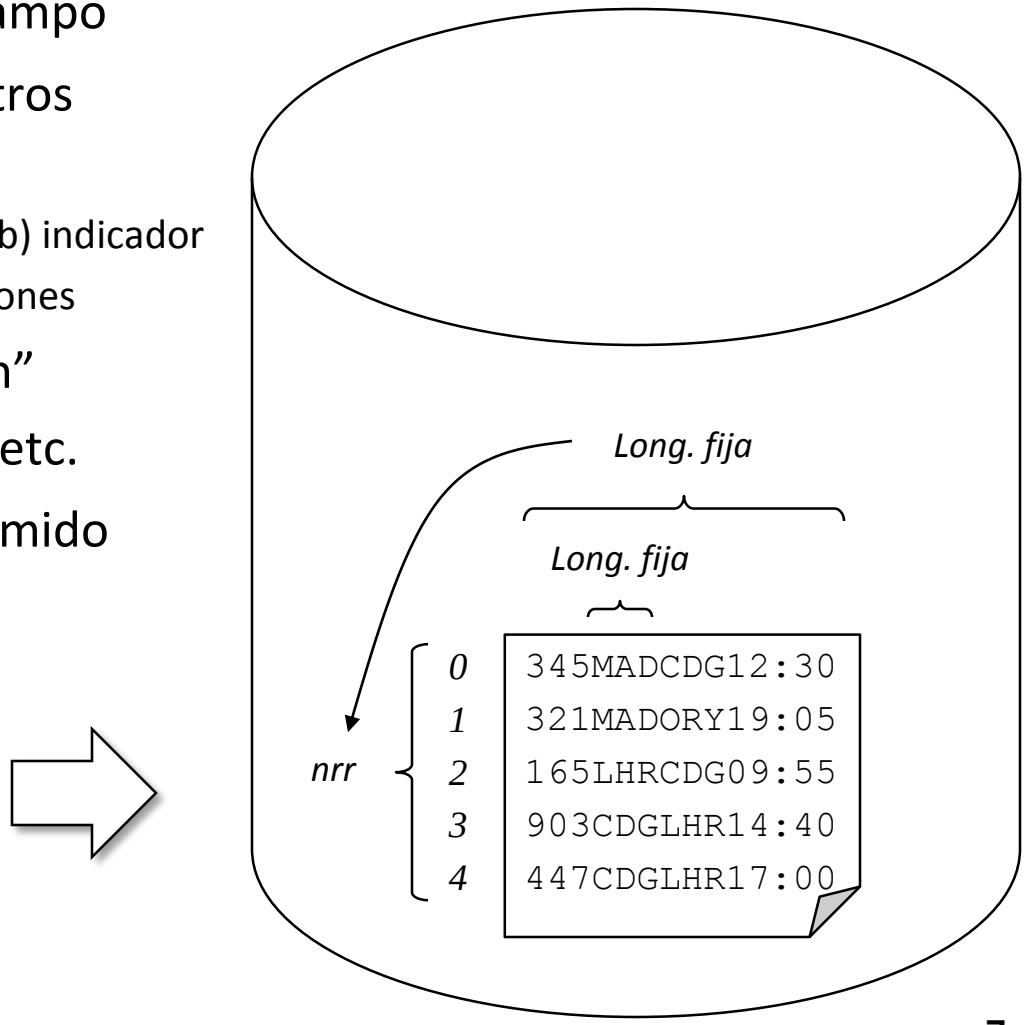
- Introducción y fundamentos
- Introducción a SQL
- Modelo Entidad / Relación
- Modelo relacional
- Diseño relacional: formas normales
- Consultas
  - Cálculo relacional
  - Álgebra relacional
- Implementación de bases de datos
  - Estructura física: campos y registros
  - Indexación
    - Índices simples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

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

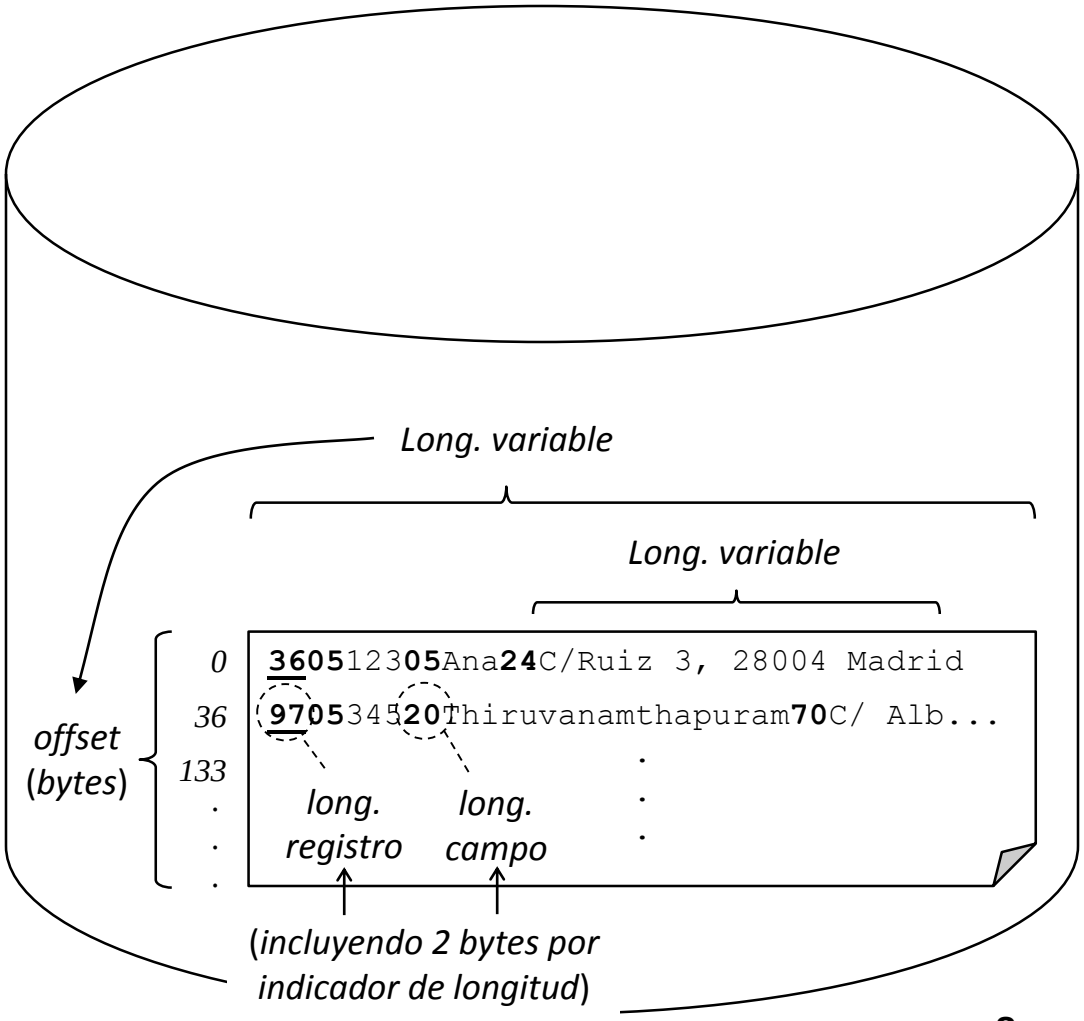
Numero	Origen	Destino	Salida
345	MAD	CDG	12:30
321	MAD	ORY	19:05
165	LHR	CDG	09:55
903	CDG	LHR	14:40
447	CDG	LHR	17:00



Almacenamiento de tablas: Campos y registros

EMPLEADO

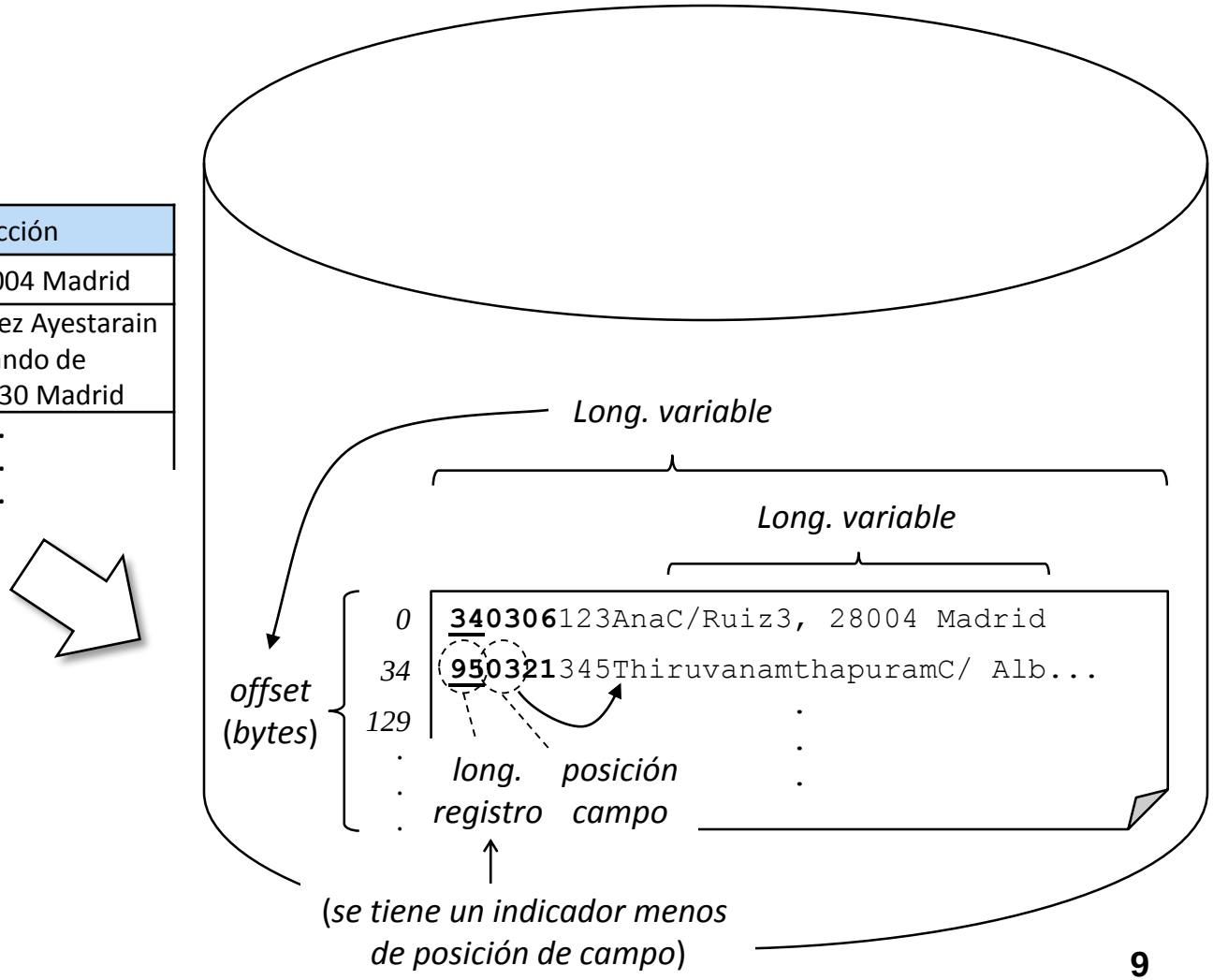
DNI	Nombre	Dirección
123	Ana	C/ Ruiz 3, 28004 Madrid
345	Thiruvanamthapuram	C/ Albino Pérez Ayestarain 14, San Fernando de Henares, 28830 Madrid
.	.	.
.	.	.
.	.	.



Almacenamiento de tablas: Campos y registros

EMPLEADO

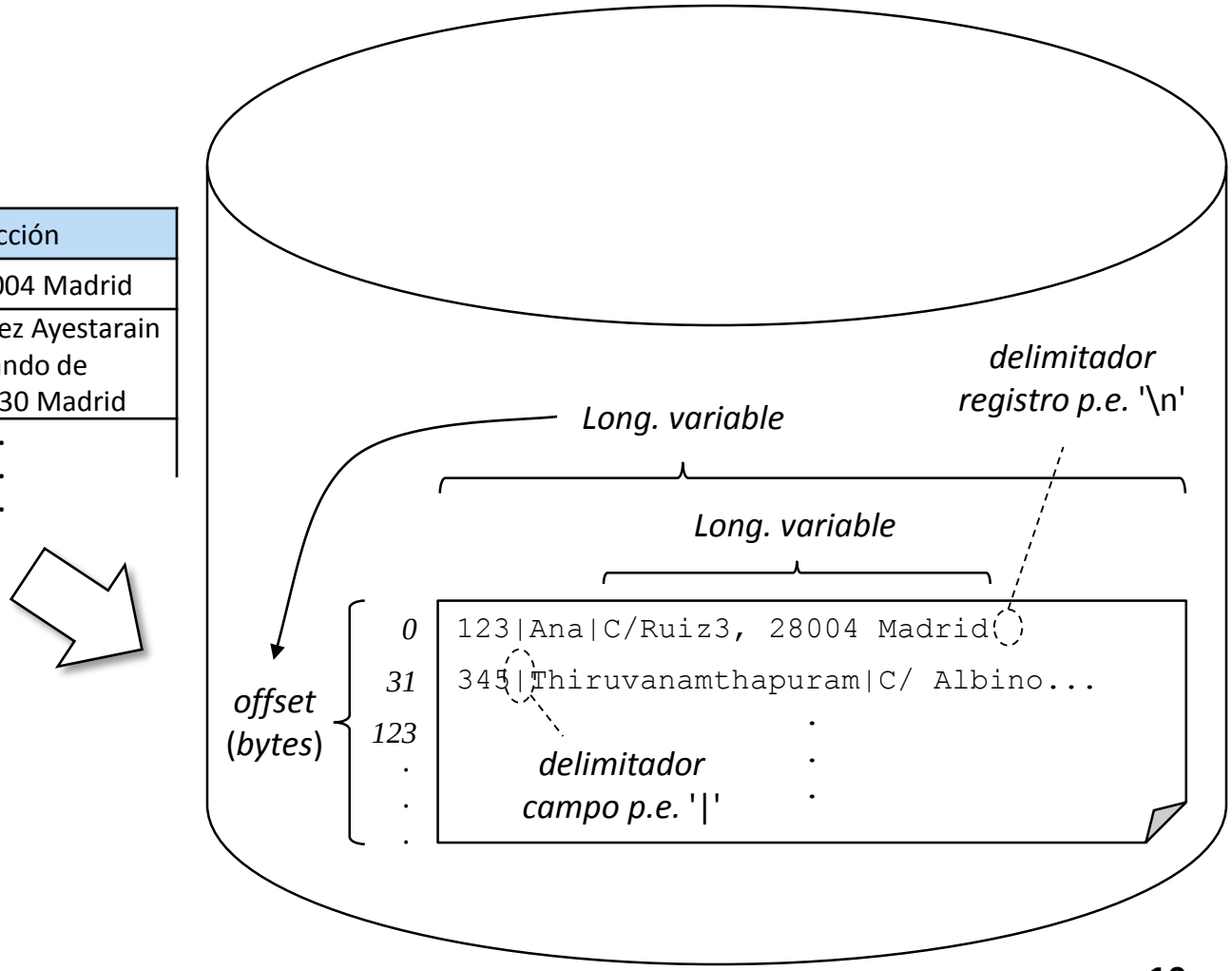
DNI	Nombre	Dirección
123	Ana	C/ Ruiz 3, 28004 Madrid
345	Thiruvanamthapuram	C/ Albino Pérez Ayestarain 14, San Fernando de Henares, 28830 Madrid
.	.	.
.	.	.
.	.	.



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 nº de orden (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

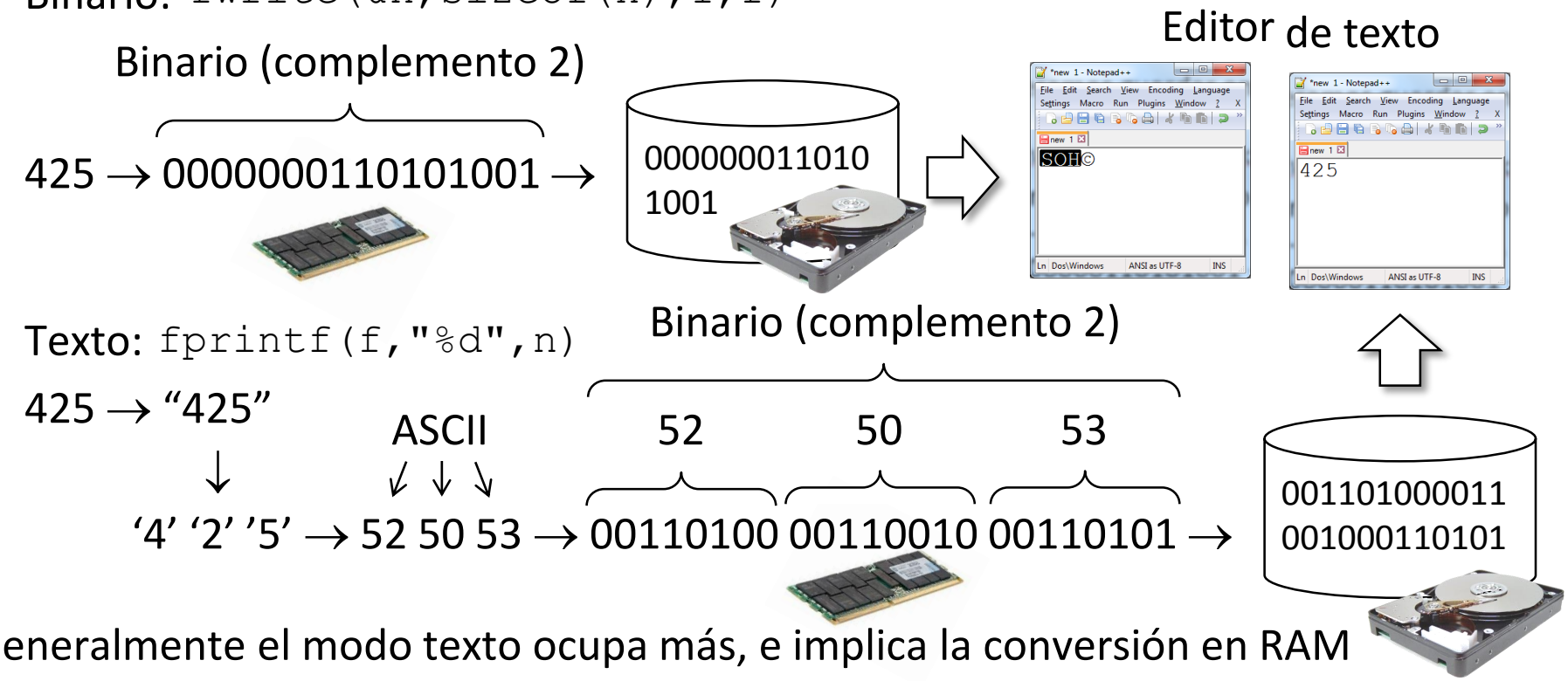
¿Es mejor longitud fija o variable?

- 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

Escritura/lectura en binario vs. en texto

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

- Binario: `fwrite(&n,sizeof(n),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

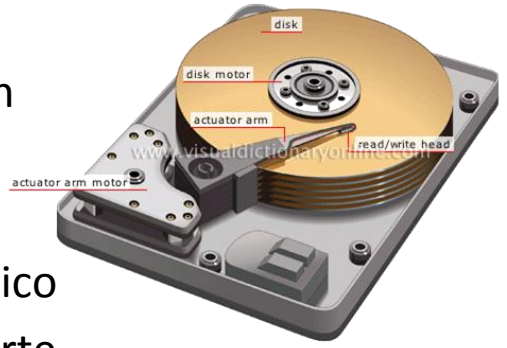
Operaciones en disco

- 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
  - En 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

Dispositivos de almacenamiento

- Determinan el coste de las operaciones de acceso a los datos
- Memoria principal (RAM)
  - Máxima velocidad de acceso
  - Alto coste económico comparativo limita su disponibilidad
  - No persistente

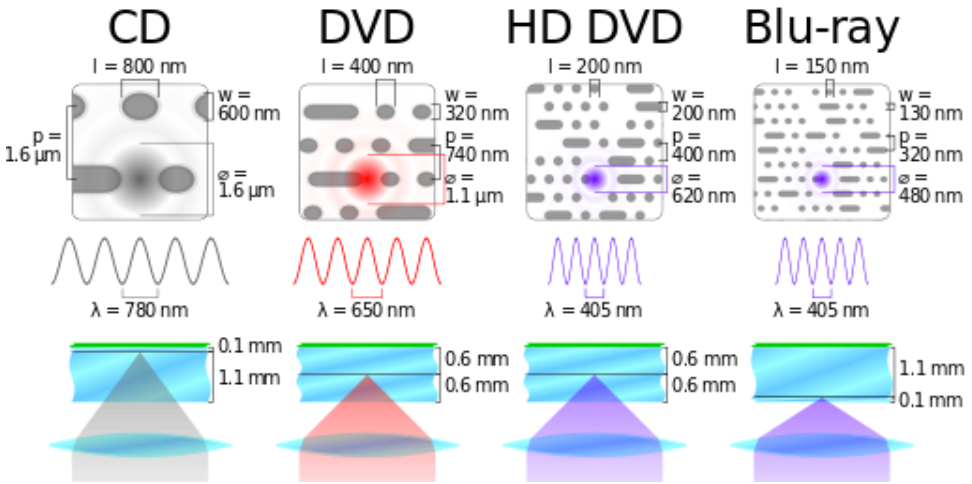
- Disco magnético
  - Sigue siendo el dispositivo de almacenamiento más común para bases de datos
  - Órdenes de magnitud más lento que la memoria RAM, básicamente debido a la necesidad de movimiento mecánico
  - Basaremos nuestro análisis de costes en este tipo de soporte



- Memoria flash / SSD
  - “Solid state” alude al contraste con piezas móviles
  - Lectura casi tan rápida como la memoria RAM
  - Escritura más lenta, y requiere borrado previo
  - Nº limitado de reescrituras (p.e. varios miles)
  - Ha ganado mercado en los últimos años

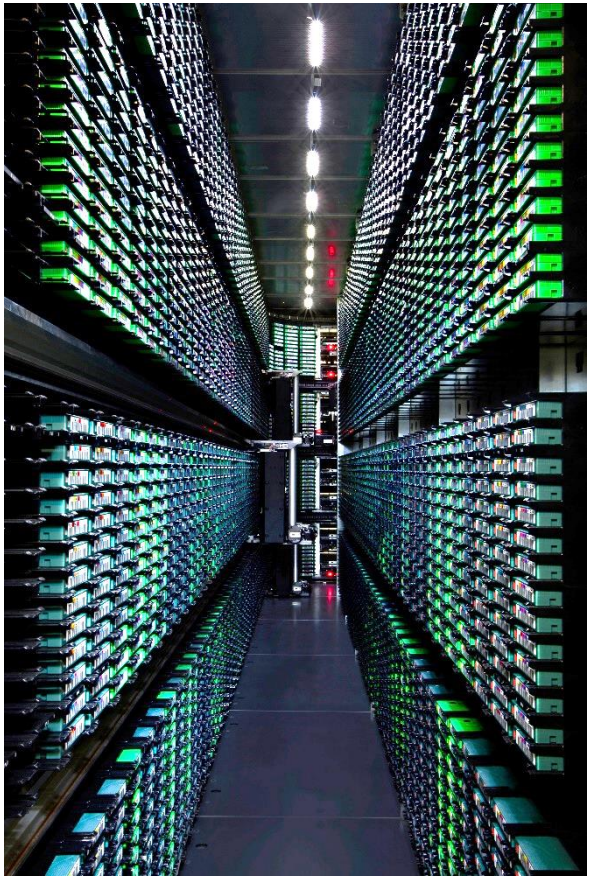
Dispositivos de almacenamiento (cont)

- 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

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 (nº de bloques a leer / nº de bloques por pista × tiempo rotación)
- 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
- 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

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, no ha lugar, el orden no importa
    (pero se usan igualmente listas de registros borrados)

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

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
    - Hashing

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 posible
- Las condiciones de comparación atributo / constante se pueden resolver por búsqueda binaria en un índice
- La combinación de condiciones simples se resuelve por operaciones de procesamiento cosecuencial
- Otras consultas más complejas aprovechan menos la optimización que permiten de los índices

Consultas simples

- 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
- 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
  - Incluso, se puede conseguir mejor que $O(\log n)$?
- Solución óptima: índices!
  - Búsqueda en $O(1)$
  - 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)
  - **Hashing**: el mapping lo da una función que se utiliza tanto para posicionar los registros al insertar, como al buscarlos por campo

Índice simple

- **Cabe en RAM** y las operaciones en él (búsqueda, actualización, reordenación, etc.) tienen coste mínimo comparado con el acceso a disco
- **Ordenado** para permitir búsqueda binaria y búsquedas por desigualdad (rangos)
- Permite búsqueda por tantos campos como se desee, sin tocar el fichero de datos (en realidad cualquier método de indexación)

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 primaria (y usan direcciones de registros)
  - Índices secundarios: todos los demás (usan claves primarias como id de registros)
- El índice primario juega un papel especial: es el que realmente contiene direcciones físicas de los registros

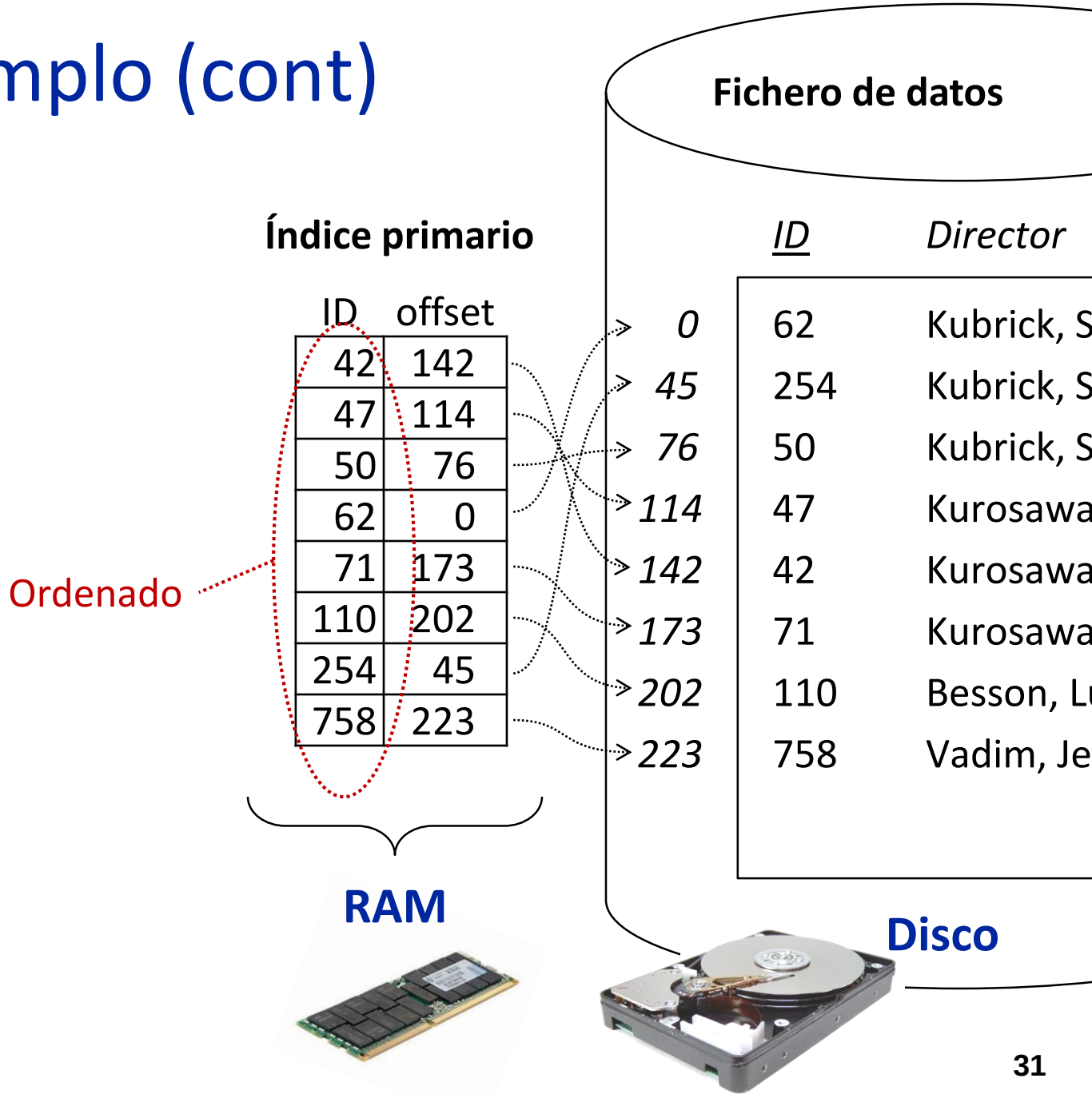
Ejemplo

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

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

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 + read) al registro sabiendo su posición

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*, *d*> 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
- Modificación de un campo
  - Archivo de datos: modificar el valor del campo
  - Índice: si el campo **es parte de la clave** primaria: eliminar + insertar en el índice
  - Se mueve el registro → actualizar su dirección en el índice (o eliminar + insertar)

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: análogo al índice primario
- 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
  - Compactación → el índice secundario **no cambia**

Operaciones – índice primario (cont)

Operaciones en RAM
Operaciones en disco

- Modificación de un campo
  - Archivo de datos: modificar
  - Índice: si **afecta a la clave** secundaria, recolocar para mantener el orden
  - Si **afecta a la clave primaria**, actualizar en el secundario la clave cambiada
  - Si la clave primaria aparece en entradas de clave secundaria repetida, recolocar para **mantener el orden de la clave primaria**
  - Si afecta a **otros campos**: el índice secundario **no cambia**, aunque se moviese el registro

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 R ∩ 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 ($n^{\text{nº campos}}$), 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

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 equijoins 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

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

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

- Métodos alternativos
  - Iteración secuencial en una lista, búsqueda binaria en la otra
  - Gallop search
  - Skip lists
  - …

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 (es un típico error utilizar offset en el índice primario para registros de longitud fija)
- Los índices simples tienen que caber en RAM, de lo contrario en general no son una buena solución

Límite de los índices simples

- Los índices simples son una solución óptima para la ejecución de consultas
  - Imposible mejorar 0 accesos a disco! ☺
  - Sólo si el archivo de datos entero cabe en RAM… y prescindimos de indexar
- Todo lo razonado hasta aquí presupone que los índices se manejan en RAM

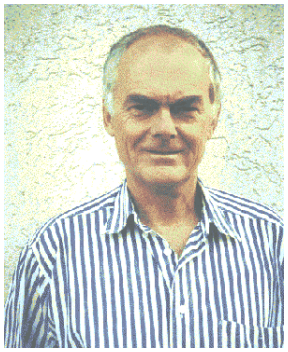
¿Y si un índice no cabe en RAM…?

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

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

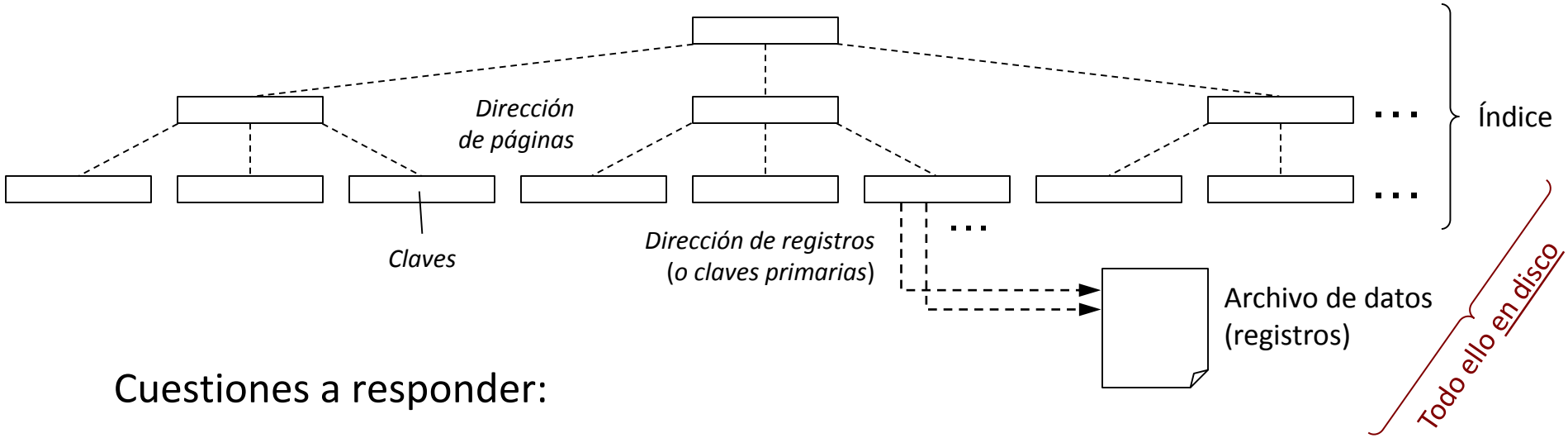
Árboles B (cont)

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

Árboles B (cont)



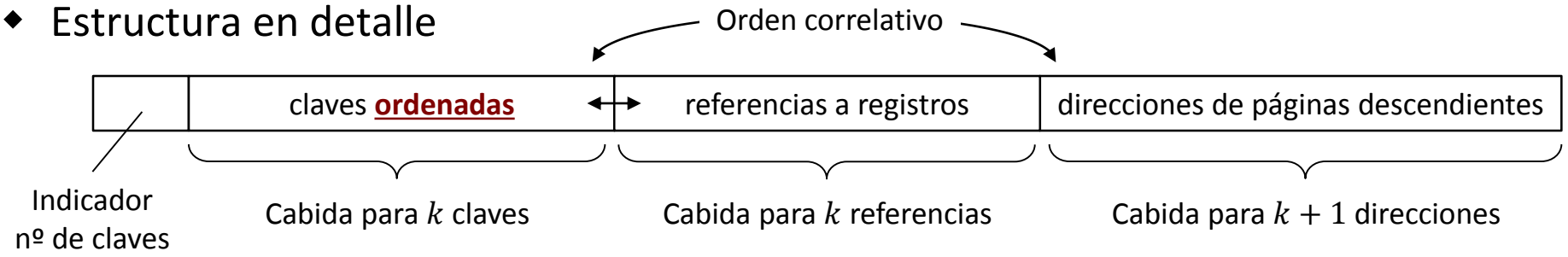
Cuestiones a responder:

- ¿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 → soluciones a estas preguntas: estructuras y algoritmos

Árboles B – Estructura

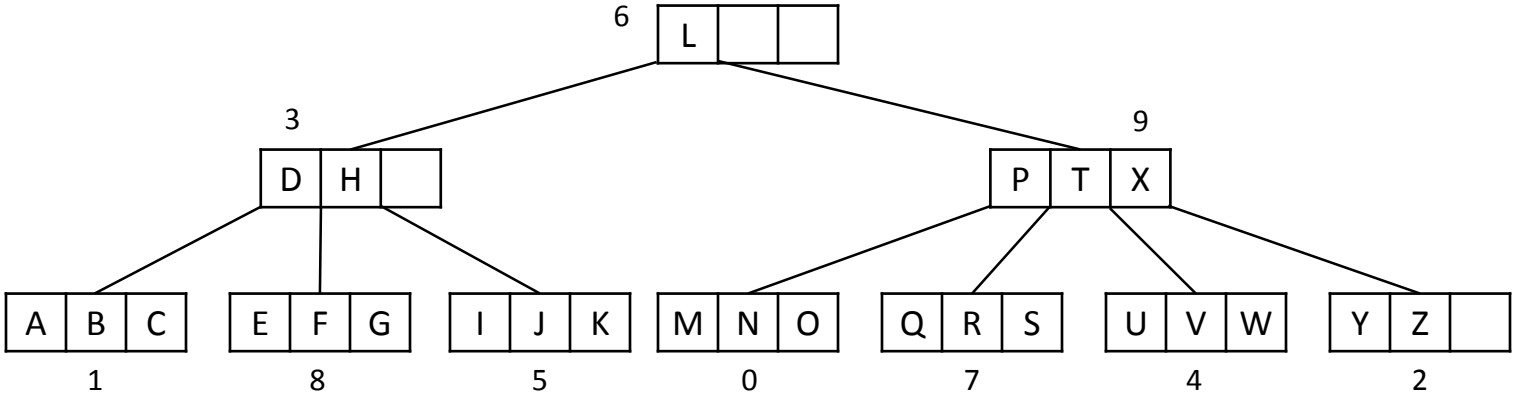
- Contenido de una página
  - Claves ordenadas acompañadas por dirección de página hija; cada clave "separa" dos páginas hijas
  - Las claves de cada página son "mayores" que las de la página que la "precede"
  - La clave que separa dos páginas es mayor que las claves de una y menor que las claves de la otra
  - Las direcciones de página en las páginas hoja son "null" (p.e. -1)
- Estructura en detalle



- La cabecera del archivo de índice debe contener la dirección de la página raíz
- Dos opciones en el almacenamiento de claves
  - Árboles B básicos: todas las páginas contienen claves del índice e id de registros
  - Árboles B+: las hojas contienen todas las claves, las demás contienen "separadores"

Ejemplo

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



Vista física del árbol B

Raíz → 6

		Claves			Referencia registros			np's 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

Árboles B – Operaciones

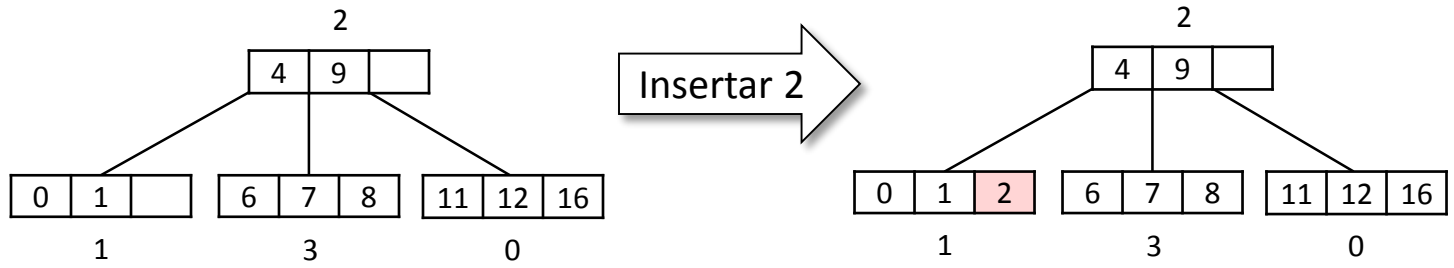
- **Búsqueda**
  1. Iniciar en *np* ← raíz
  2. Si *np* = –1, **FIN** (no se encuentra)
  3. Leer la página *np* a RAM, buscar la clave en la página
  4. Si se encuentra la clave, **FIN** (clave encontrada)
  5. Si no, *np* ← página descendiente correspondiente al orden de la que se busca, volver al paso 2
- **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
  4. Si la página era la raíz, crear una nueva raíz (aumenta la altura del árbol), **FIN**

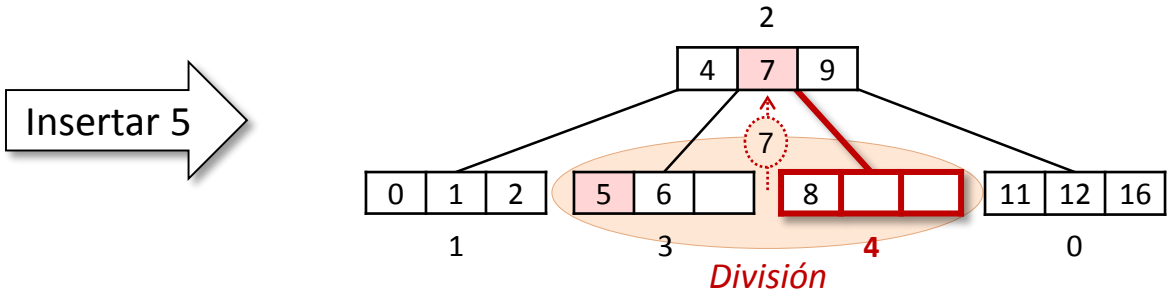
     Si no, subir la clave intermedia + nº de página a la página "padre", volver al paso 2

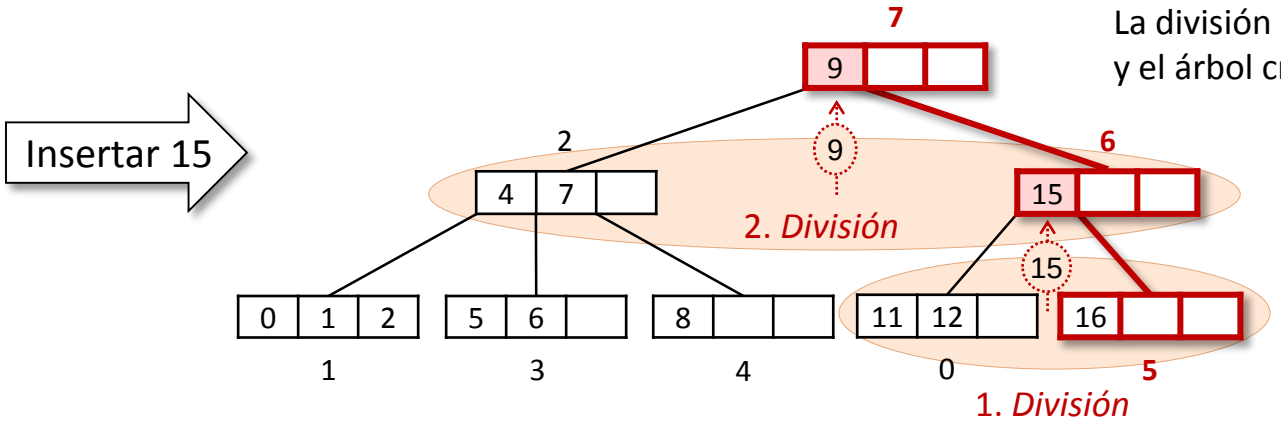
  Por este procedimiento, en general las páginas no estarán completamente llenas
- **Eliminación**: lo vemos en un momento… primero veamos ejemplos

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)

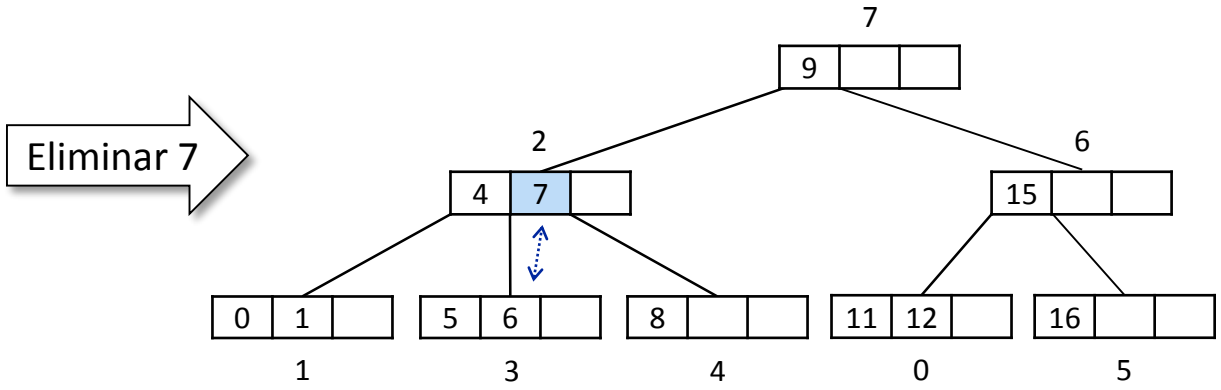
- Eliminación
  1. Buscar la página donde se encuentra la clave
  2. Si no es una página hoja, intercambiar la clave con la clave “inmediatamente anterior” o posterior en una hoja
  3. Eliminar la clave de la página
  4. Si la página tiene el mínimo de claves que queremos garantizar, **FIN**
  5. En otro caso:
     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, sacando la clave que las separa en la página padre

        Volver al paso 3 con la página eliminada en la página padre

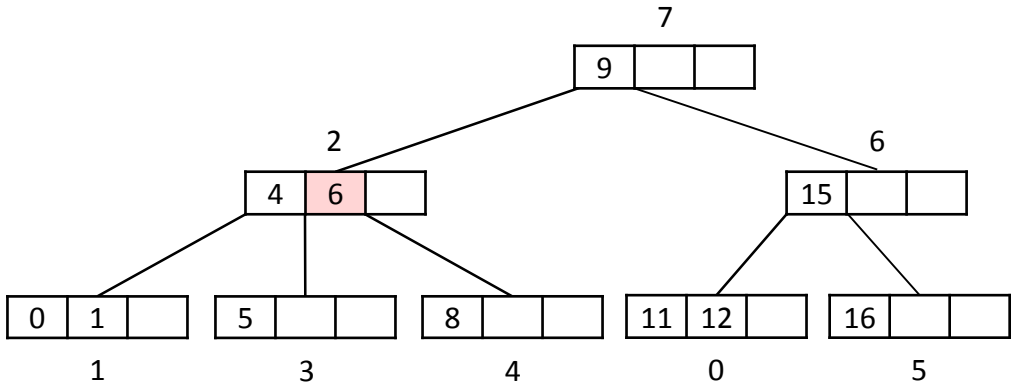
  Si la concatenación se propaga hasta la raíz, se reduce en un nivel la altura del árbol

Ejemplo eliminación en árbol B

$k = 3$

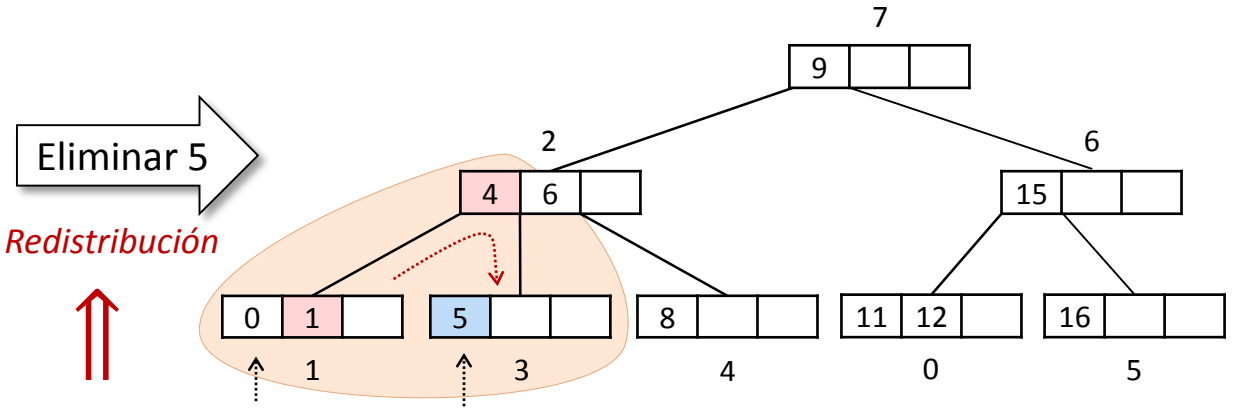


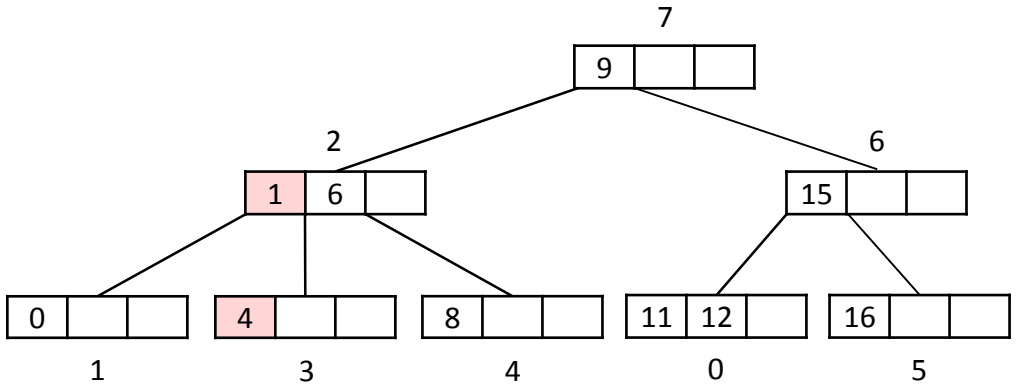
Se intercambia 7 con 6 en la página hoja 3 (o con 8 en la página 4)



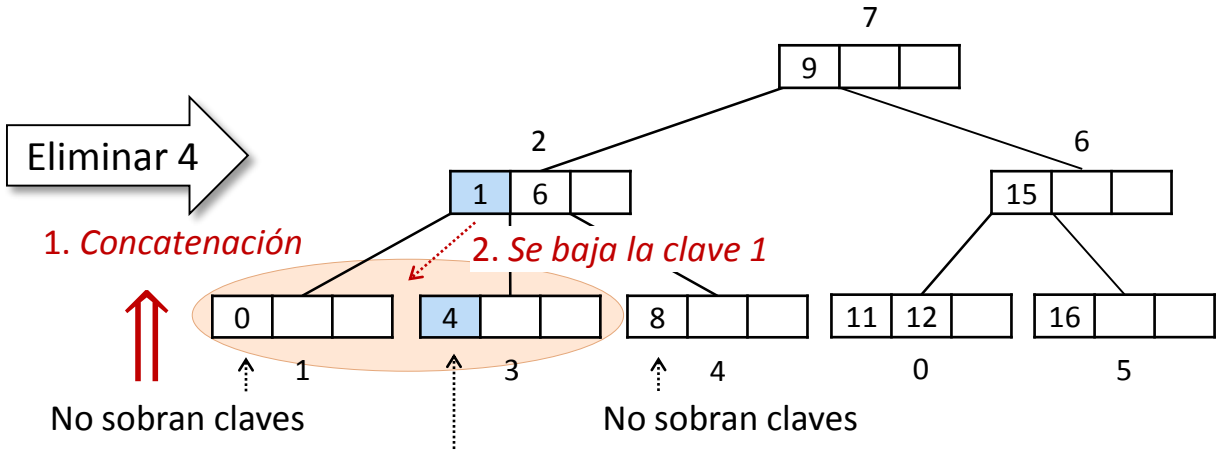
Se elimina 7 de la página 3

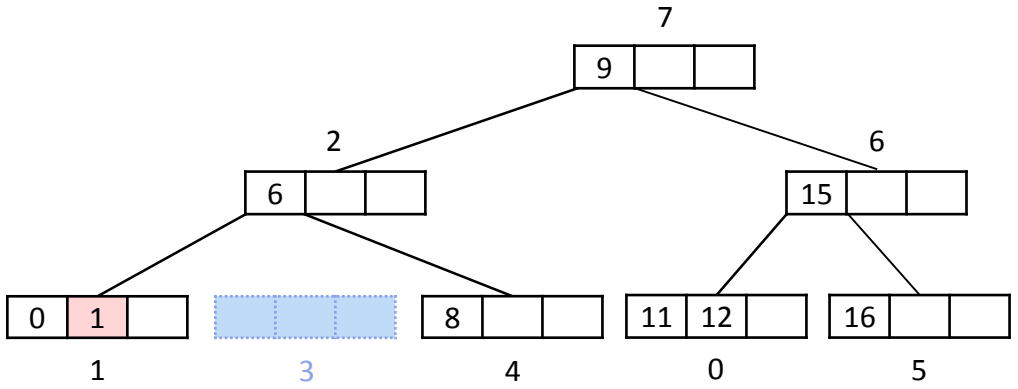
Ejemplo eliminación en árbol B (cont)





Ejemplo eliminación en árbol B (cont)



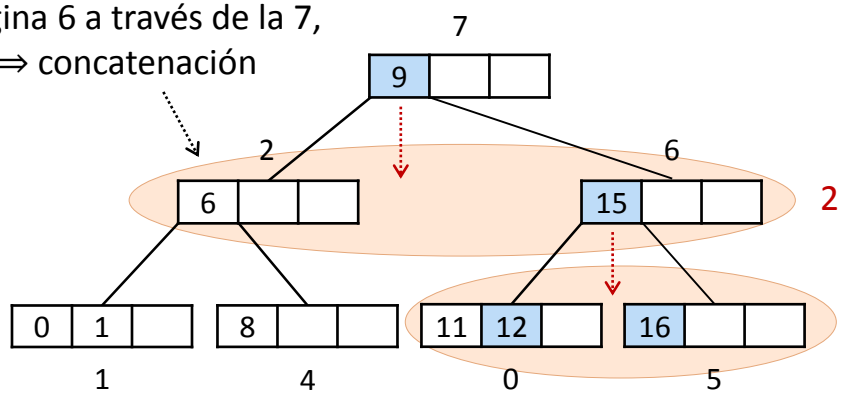


La página 3 queda "borrada"
(se marca como borrada, se recicla, etc.)

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

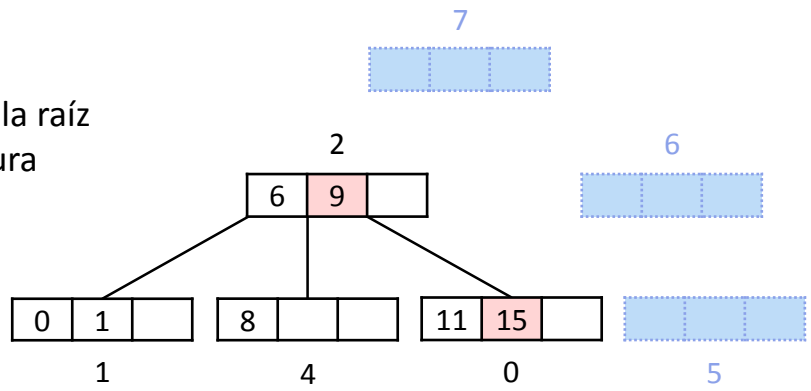
Eliminar 12, 16



2. *Concatenación*

1. *Concatenación*

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



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, se puede ver que la altura del árbol es…

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

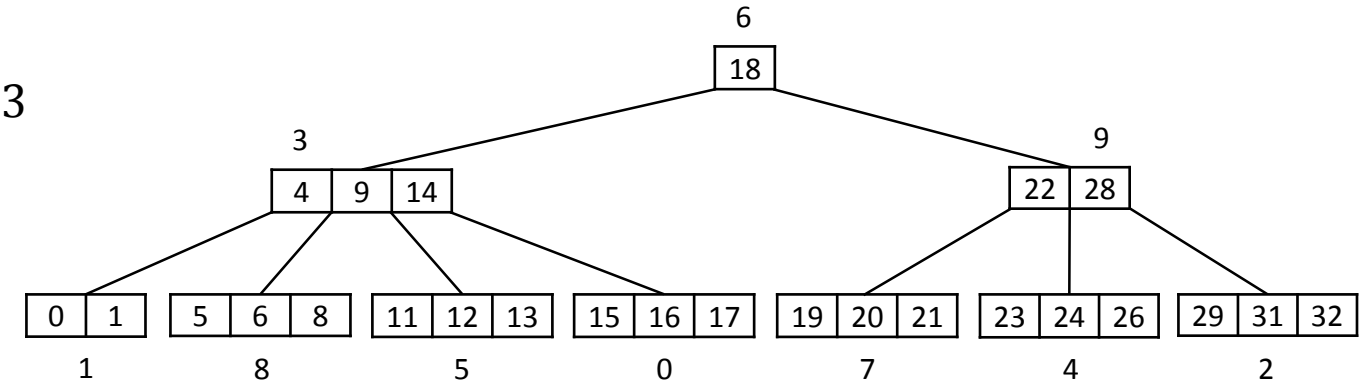
- Por tanto, a mayor m, menor coste en las operaciones
- Con la técnica básica de inserción, aseguramos… $m = \lfloor k/2 \rfloor$
- ¿Podemos conseguir mejores mínimos? Vamos a verlo…

Optimizaciones

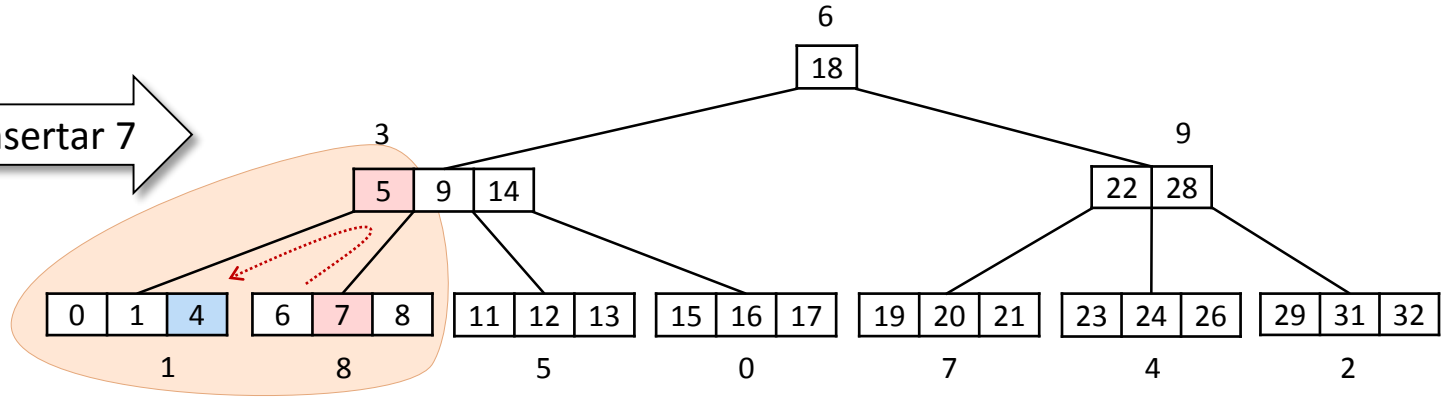
- Árboles B virtuales
  - Se mantienen *x* páginas en RAM
  - Cuando se necesita una página que ya está en RAM, se evita su coste de lectura
  - Cuando se necesita una página que no está en RAM, ocupa el lugar de otra en RAM
  - Estrategias para decidir de qué página se prescinde: la más baja en el árbol, la usada menos recientemente, estadísticas de acceso…
- Árboles B*
  - Inserción con redistribución
  - Dividir dos páginas en tres
  - La ocupación mínima de las páginas es… $m = \lfloor 2\,k/3 \rfloor$
  - Asegurar este mínimo también al eliminar (en particular, 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)
- Páginas de longitud variable
  - Cabrán más claves por página
  - El orden del árbol ya no es fijo –la capacidad de las páginas no se define en nº de claves, sino en bytes

Ejemplo inserción en árbol B*

$k = 3$

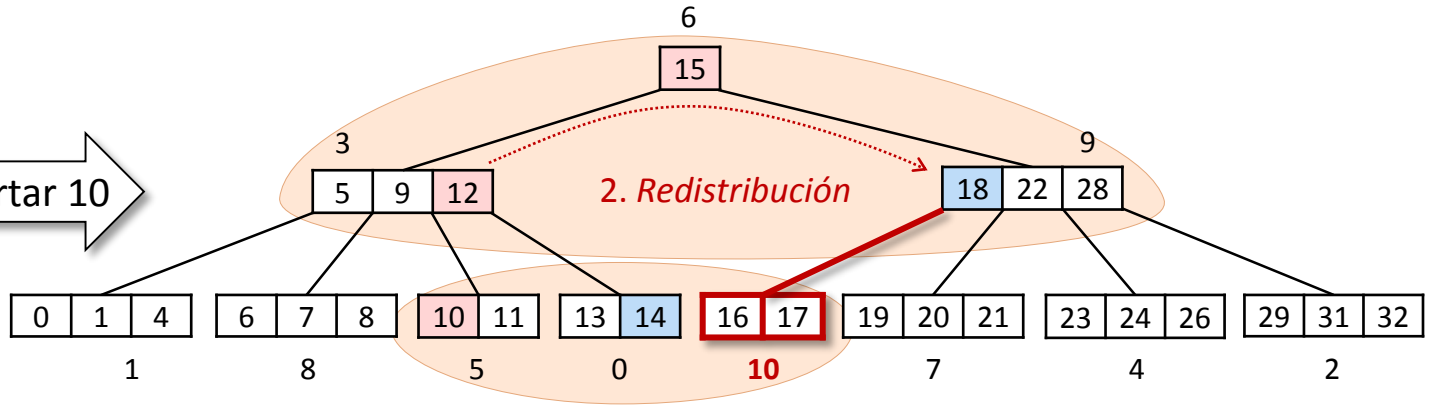


Insertar 7



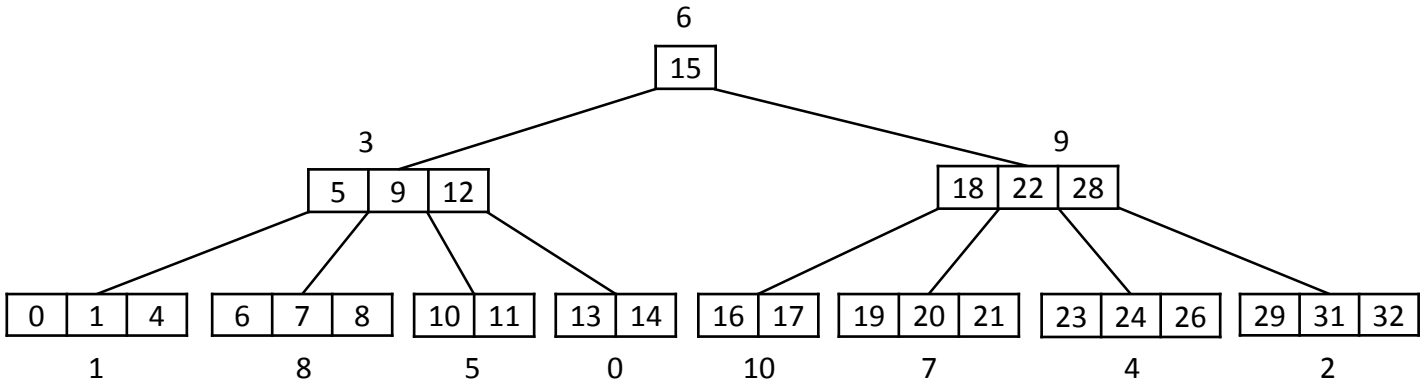
Redistribución

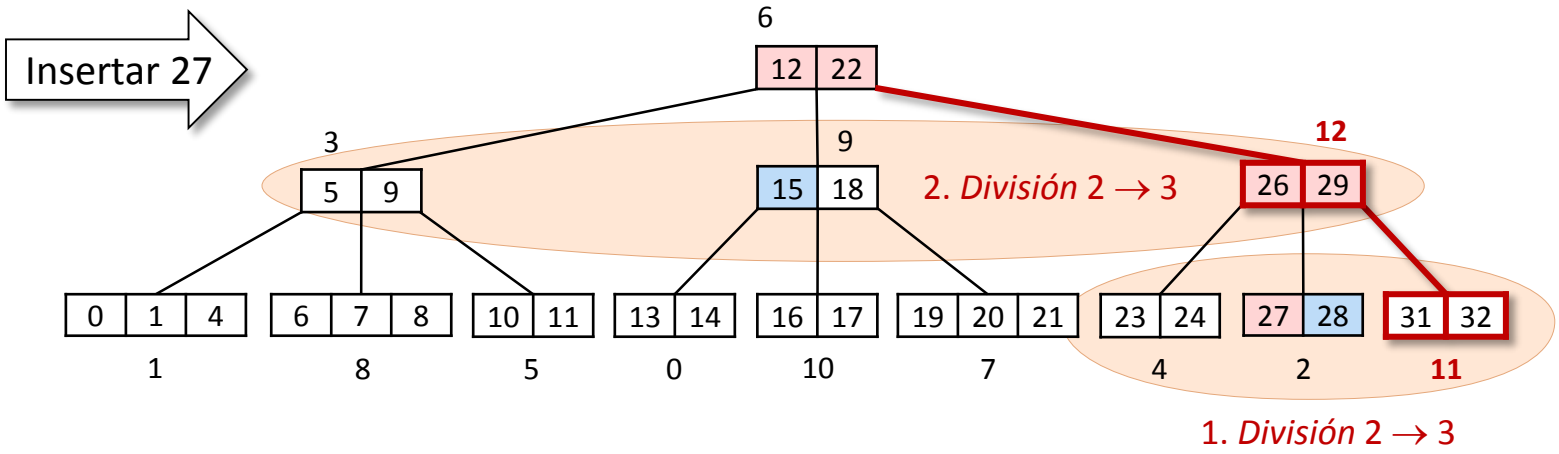
Insertar 10



1. División 2 → 3

Ejemplo inserción en árbol B* (cont)

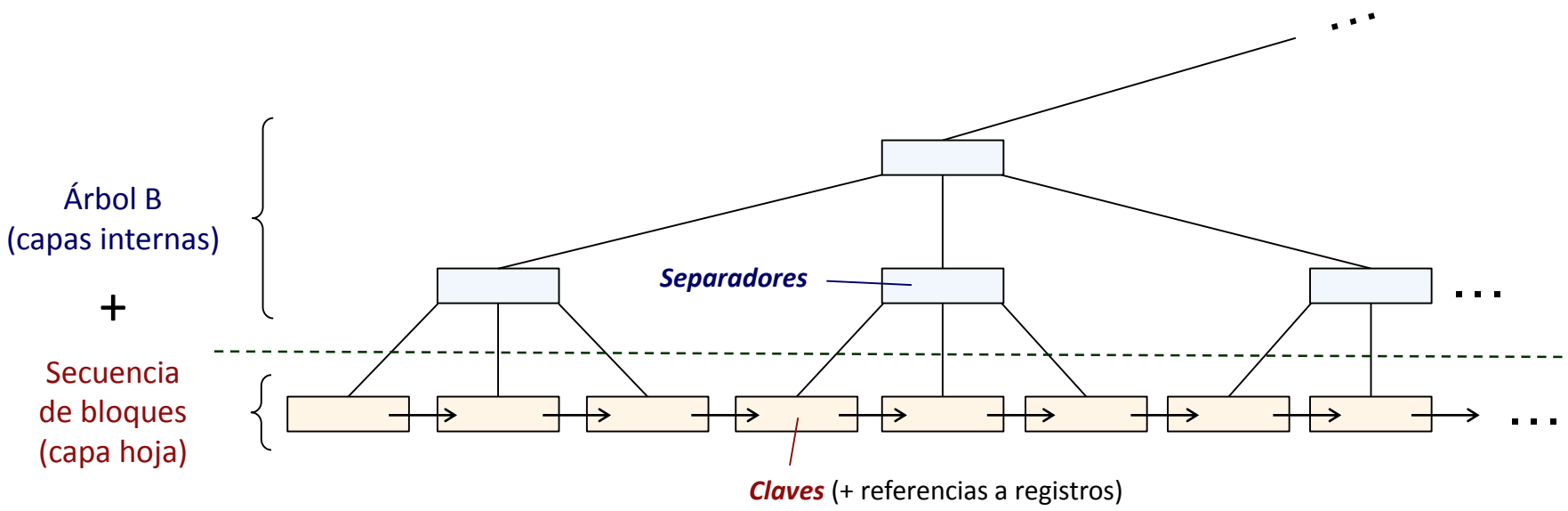




Árboles B+

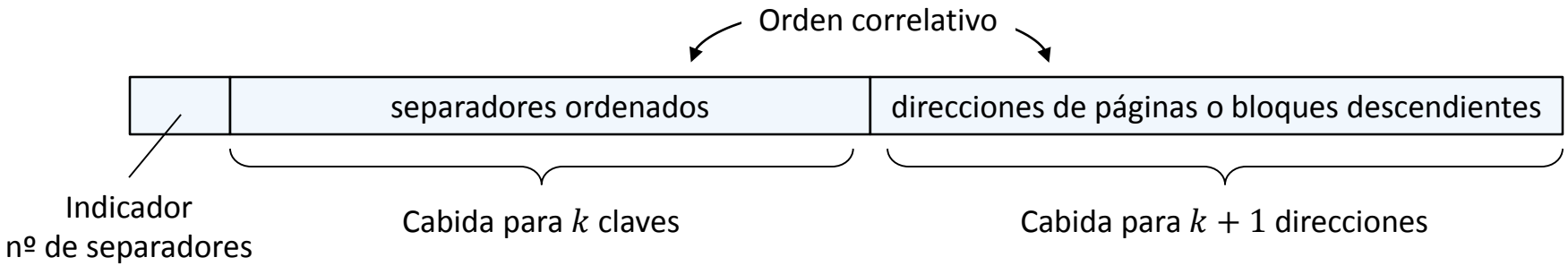
- Las claves (junto con los IDs –dirección física o clave primaria– de registros) están todas en las hojas
  - Las hojas contienen referencias a registros y no contienen nºs de páginas hijas
- Las páginas internas contienen separadores
  - No contienen referencia a registros
  - Los separadores pueden 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.)
  - Para ello, las páginas hoja (bloques) deben enlazarse cada una con la siguiente: así se pueden recorrer sin subir a las páginas internas

Árboles B+, estructura lógica externa

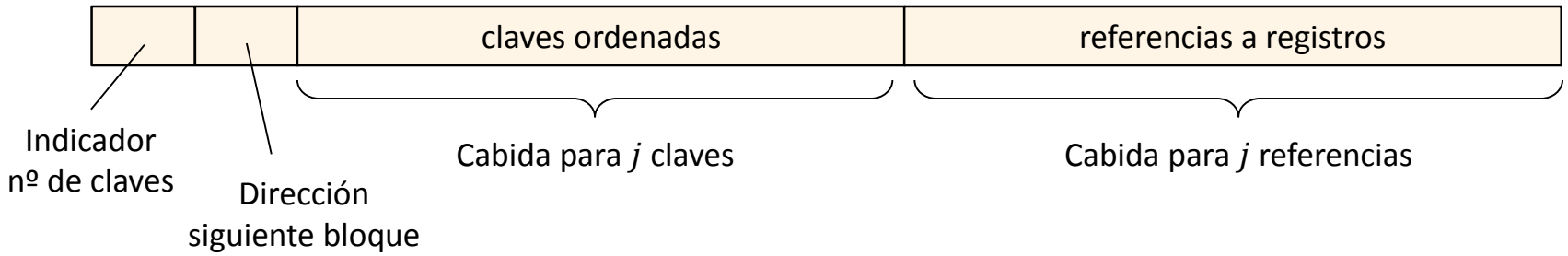


Árboles B+, estructura física interna

- Página del árbol B (“capas internas”)



- Bloque de claves (“capa hoja”)



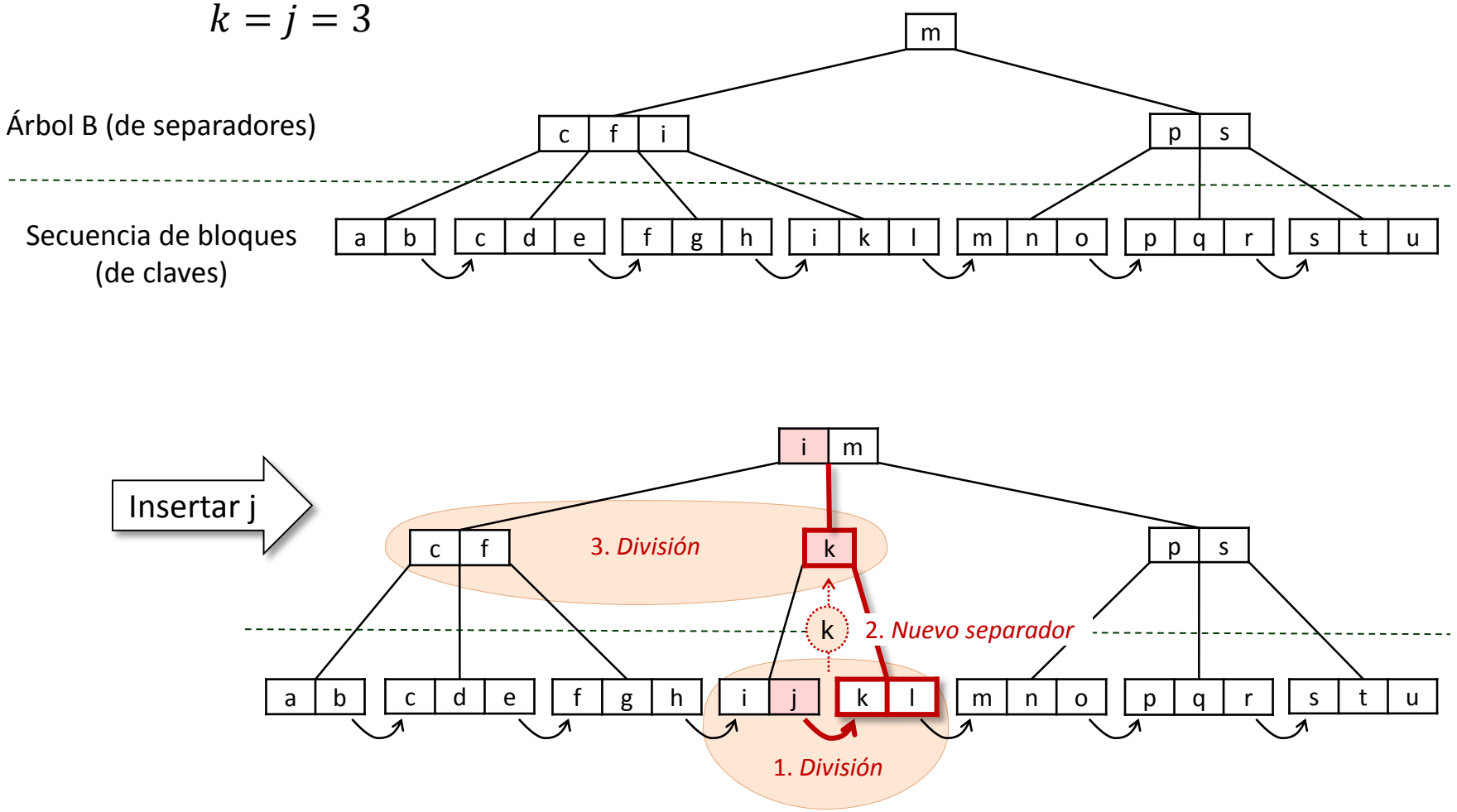
(No tiene por qué ser $j = k$)

Árboles B+, Operaciones

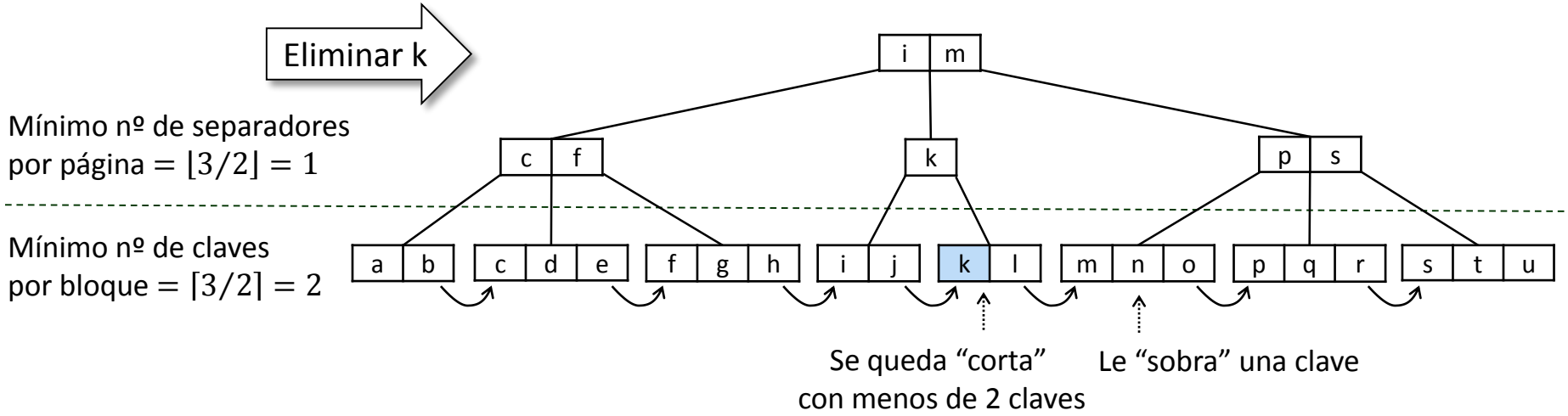
- Búsqueda
  - Similar a los árboles B, pero no se detiene hasta llegar a las hojas
- Inserción
  - Cuando se divide una hoja, no se deja una clave en medio, sino que se crea un nuevo separador que se inserta en el siguiente nivel del árbol ($\Rightarrow m = \lceil j/2 \rceil$ en las hojas)
- Eliminación
  - 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)
  - En otro caso no se necesita modificar nada más que la eliminación de la clave en la hoja correspondiente (sin más cambios en el árbol B)
  - La concatenación y redistribución se pueden hacer con hojas consecutivas, aunque no desciendan de la misma página (se buscan los separadores en el árbol)

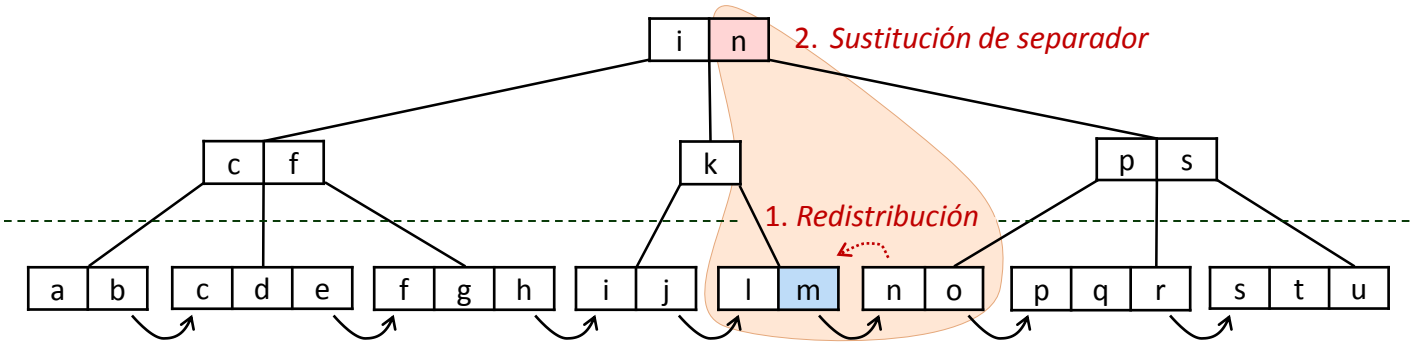
Ejemplo inserción en árbol B+

$k = j = 3$

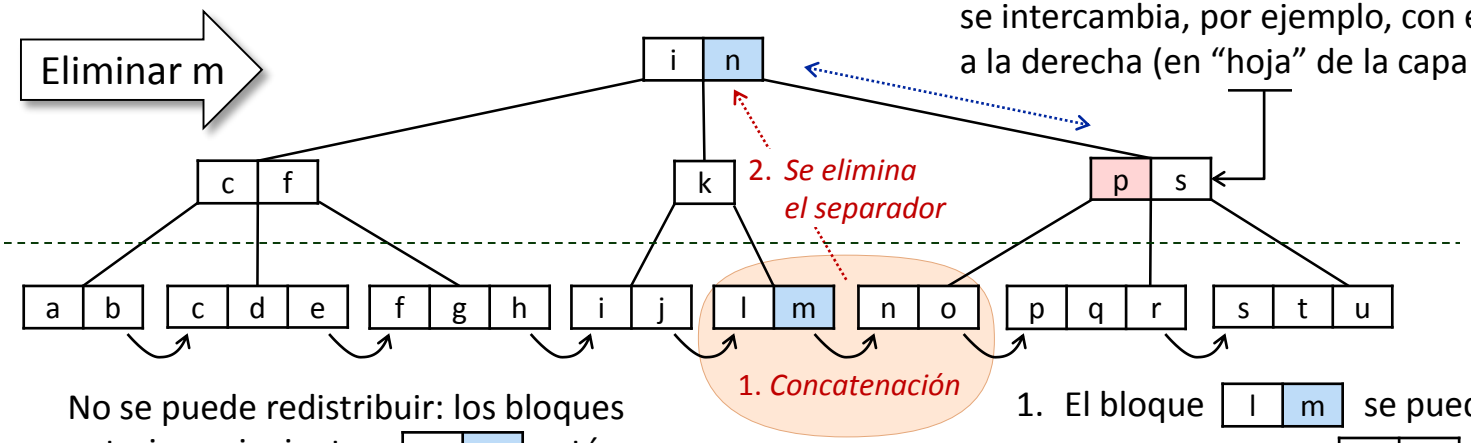


Ejemplo eliminación en árbol B+ (cont)





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