Almacenamiento de registros y organizaciones primarias de archivos

Las bases de datos se almacenan físicamente como ficheros de registros, normalmente en discos magnéticos.

Bibliografía de la presentación: ***"Sistemas de Bases de Datos Conceptos Fundamentales", Elmasri y Navathe, segunda edición, capítulos 4 y 5.***

Introducción

La colección de datos que constituye una base de datos computarizada debe estar almacenada físicamente en algún medio de almacenamiento en el computador.

Almacenamiento Primario

La CPU puede acceder a los datos directamente
Acceso rápido pero capacidad limitada
Memoria principal: DRAM (Dynamic Random Access Memory), bajo coste, volátil
Memoria caché: RAM estática, más pequeña, más rápida y más cara

Almacenamiento Secundario

La CPU no puede acceder directamente: hay que copiarlos antes a memoria principal
Más lento, mayor capacidad y más baratos
Discos ópticos
Discos magnéticos
Cintas

Dispositivo de almacenamiento secundario

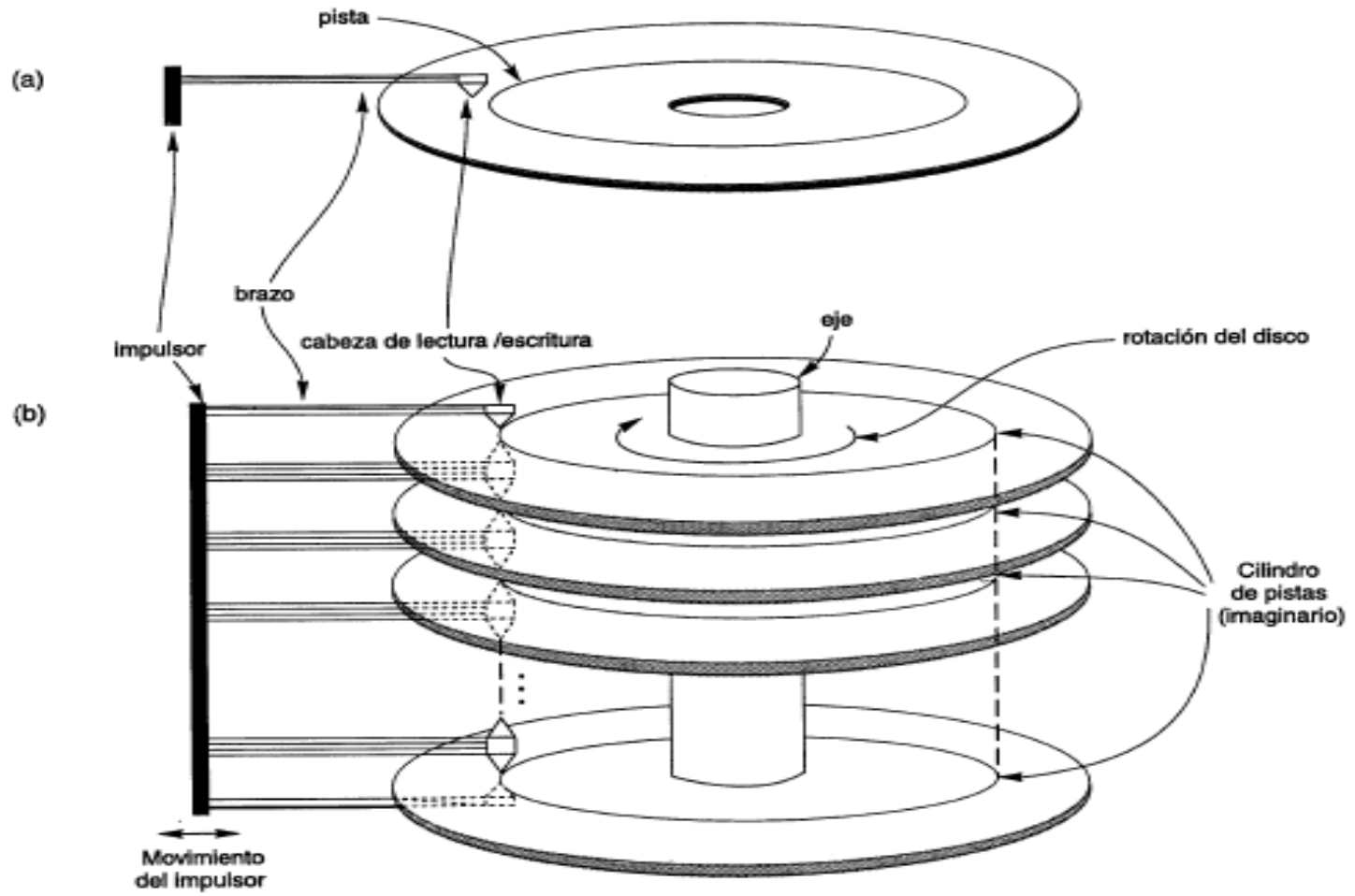


Figura 4.1 (a) Disco de un solo lado con hardware de lectura/escritura.
(b) Paquete de discos con hardware de lectura/escritura.

Discos magnéticos

Los sectores son divisiones físicas del disco

Densidad de grabación:

Puede ser la misma en todas las pistas

También puede ser mayor en pistas centrales y menor en las más extremas

Los bloques (o páginas) son divisiones creadas por el sistema operativo al formatear el disco (no cambia dinámicamente)

Todos los bloques del disco tienen el mismo tamaño (entre 512 y 4.096 bytes)

Los bloques están separados por espacios interbloque con información de control (grabada en el formateo)

El paquete de discos gira continuamente a una velocidad que suele estar entre 3.600 y 7.200 rpm

Los disquetes, sin embargo, dejan de girar tras completarse cada transferencia de datos

El controlador de disco hace de interfaz entre el disco y el sistema computador. Son muy utilizados los controladores SCSI, IDE ...

Almacenamiento secundario de datos

Discos ópticos:

640 MB

CD-ROM (Compact Disk Read-Only Memory): vienen pregrabados y no pueden sobrescribirse – CD-R grabables.

CD-RW regrabables. Estos dos últimos se han hecho populares para respaldos.

Juke-boxes: array de CD-ROM extraíbles. Capacidad de cientos de GB. Más lentos que los discos magnéticos

DVD (Digital Video Disk): entre 4,5 y 15GB

Cintas:

Se usan para respaldos periódicos (por ej. de las BD)

Acceso secuencial (lento)

Muy barato y gran capacidad pero NO son datos on-line (almacenamiento terciario)

Juke-boxes: banco de cintas catalogadas que pueden ser cargadas automáticamente en la unidad.

Las BD normalmente residen en almacenamiento secundario y se van leyendo porciones a memoria, BD en memoria principal:

Cuando es posible descargar toda la BD en memoria

Se mantiene una copia de respaldo en disco

Es útil en aplicaciones de tiempo real, que necesitan tiempos de respuesta muy rápidos

Tiempo de acceso a los datos del disco

Tiempo de búsqueda (tb)

- Para colocar la cabeza sobre la pista (mecánico)
- Depende de la distancia a recorrer
- Es el principal retardo: 10-14 mseg en PCs y 8-9 mseg en servidores

Retardo rotacional (rr) (o latencia)

- Esperar a que el principio del bloque pase bajo la cabeza
- Caso mejor: acceso inmediato
- Caso peor: una vuelta entera
- Principal retardo en discos de cabeza fija

Tiempo de transferencia de bloque (ttb)

- Depende del tamaño del bloque, de las pistas y rpm
- Suele costar 1-2 ms/bloque

Ejemplos:

- Tiempo medio para encontrar y transferir 1 bloque: (tb + rr +ttb) mseg → entre 12 y 60 mseg.
- Al transferir varios bloques del mismo cilindro se reduce el tiempo:
  - k bloques no contiguos del mismo cilindro: tb + k * (rr + ttb) mseg.
  - k bloques contiguos del mismo cilindro: tb + rr + k * ttb mseg.
- Las organizaciones de ficheros tratan de minimizar el nº de transferencias de disco a memoria

Almacenamiento intercalado de Bloques

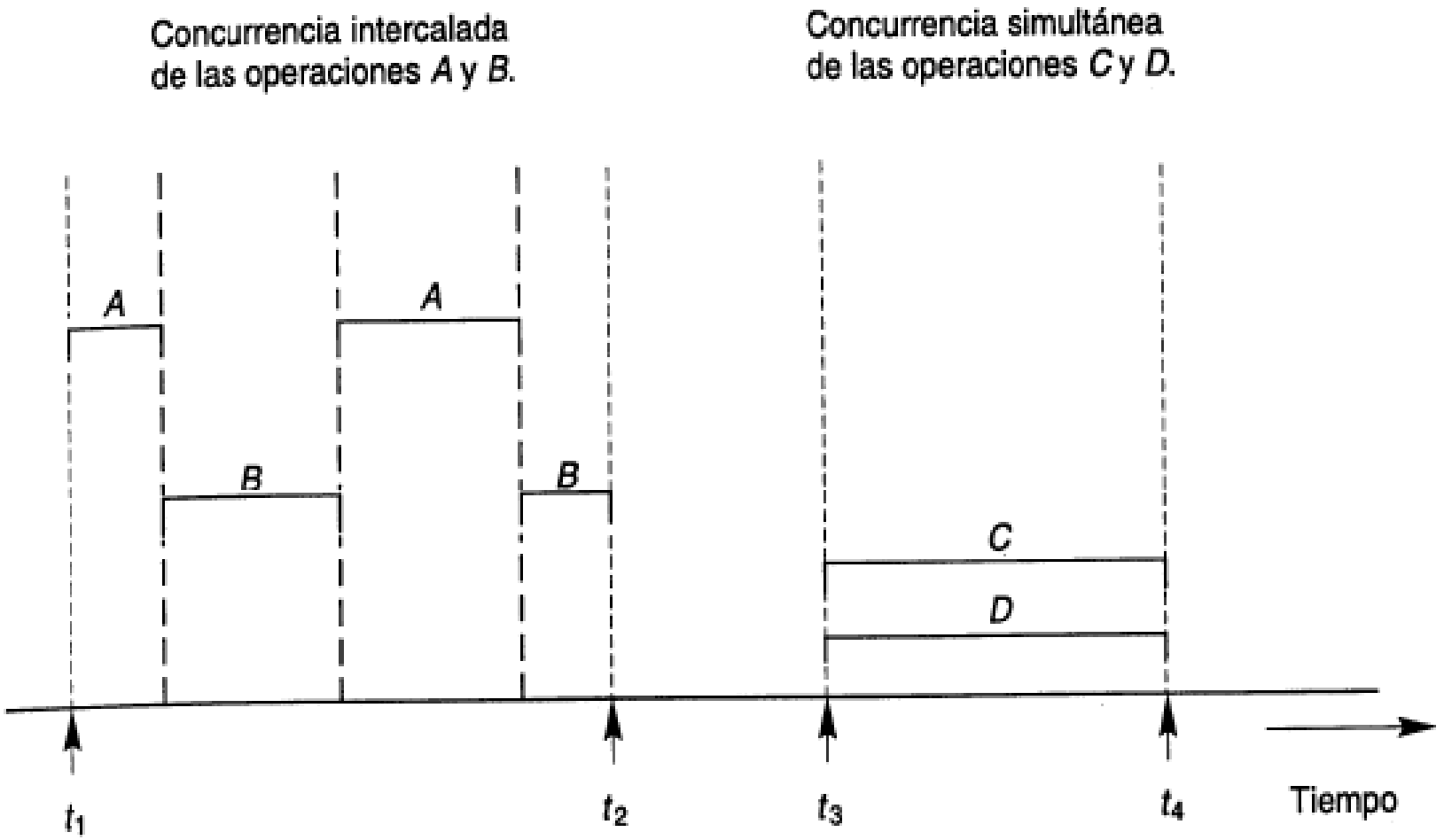


Figura 4.3 Concurrencia intercalada y simultánea.

Grabación de registros de archivo en disco

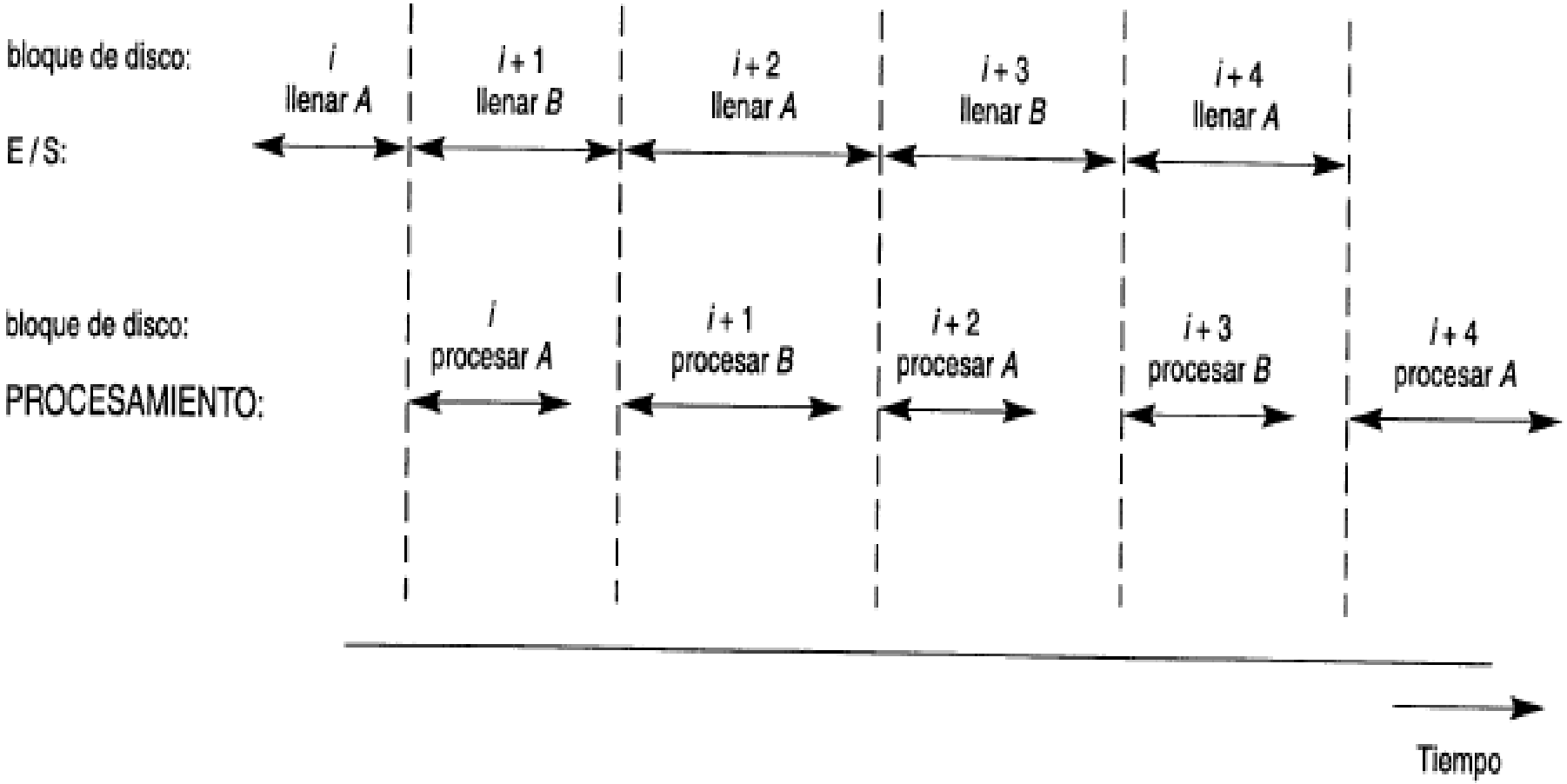


Figura 4.4 Empleo de dos áreas de memoria intermedia, A y B, para leer del disco.

Almacenamiento de BD

Las BD se almacenan en ficheros de registros

Un registro es una colección de valores de datos (por ejemplo una tupla del modelo relacional)

Los registros deben almacenarse de modo que sea posible localizarlos de manera eficiente

El SGBD ofrece varias opciones para organizar los datos

El diseño físico de BD consiste en elegir entre estas opciones las más adecuadas para los requisitos de la aplicación

Estudiaremos las principales:

Organizaciones de datos, que determinan la forma en la que se colocan los registros en el disco y por tanto cómo se puede acceder a ellos:

Ficheros de montón
Ficheros ordenados
Ficheros de direccionamiento calculado

Ficheros con índices (primarios, de agrupación, secundarios)

Métodos de acceso

Registro, bloque y búfer

Registro: entidad (E/R), tupla (relacional), fila (SQL), ...

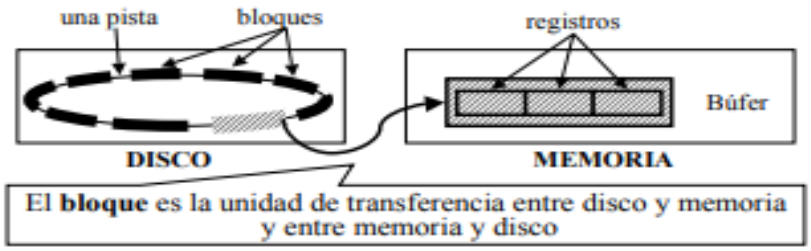
Bloque:

Unidad de transferencia entre disco y memoria
Formado por uno o varios registros

Acceso directo (o aleatorio) a un bloque:

La dirección de un bloque la forman: el nº de superficie, nº de pista y nº de bloque

Búfer: espacio en memoria principal en el que cabe un bloque



Tiempo para transferir un bloque entre disco y memoria (o viceversa): entre 12 y 60 mseg.

Tiempo bastante alto en comparación con lo que tarda la CPU en procesar sus registros (cuando están en el búfer)

La localización de datos en disco es un cuello de botella importante en las aplicaciones de BD

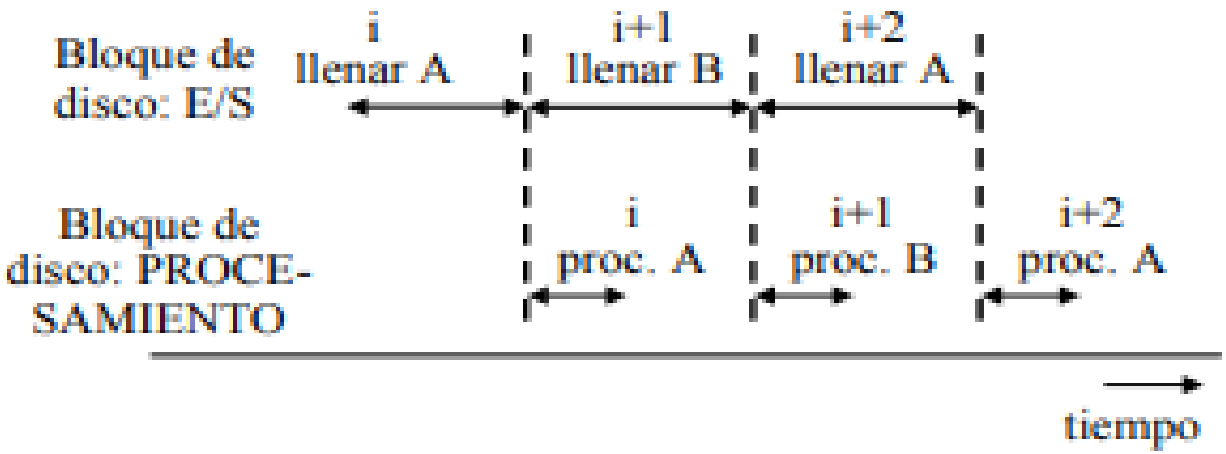
Técnica del doble búfer

Procesador de E/S (controlador de disco):

- Independiente de la CPU (en paralelo)
- Transfiere datos entre disco y memoria

Mediante la técnica del doble búfer:

- La CPU procesa el bloque del búfer
- Mientras, el procesador de E/S transfiere otro bloque desde el disco a otro búfer



Tipos de registro

Los datos se almacenan en registros

Un registro es un conjunto de valores llamados campos

Cada campo es de un tipo de datos (entero, string, boolean, fecha, etc.)

Ejemplo:

Tipo de registro EMPLEADO con campos
NOMBRE: String(1..30)
SALARIO: Integer, …

Un campo puede contener objetos binarios grandes como imágenes, video, audio o texto libre (tipo BLOB)

Se guardan aparte y en el campo contiene el apuntador al BLOB

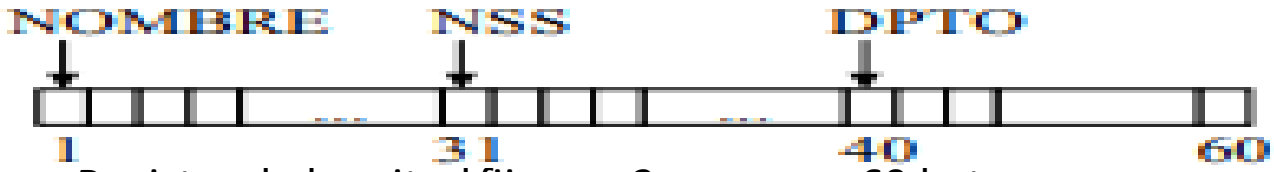
Los lenguajes de programación (Ada, C, …) permiten definir registros (record, struct,…)

Registros: longitud fija o variable

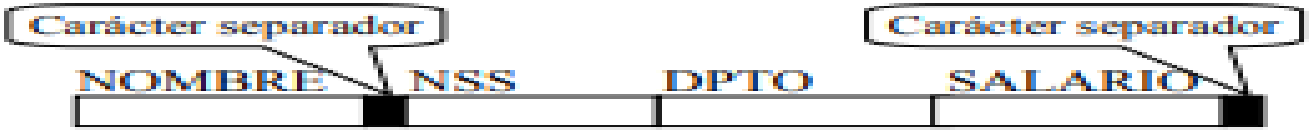
Fichero: secuencia de registros

Un registro tendrá longitud variable porque existe:

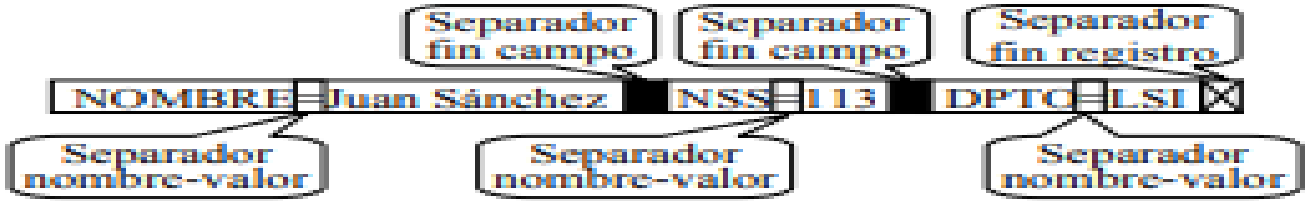
- Campo de tamaño variable
- Campo repetitivo
- Campos opcionales
- Registros de diferentes tipos en el mismo fichero



Registro de longitud fija con 3 campos y 60 bytes



Registro con 2 campos de longitud fija (NSS y DPTO) y 2 de longitud variable(NOMBRE y SALARIO)



Registro con campos de longitud variable y varios tipos de carácter separador

Factor de bloqueo

Bloque: unidad de transferencia de datos entre disco y memoria

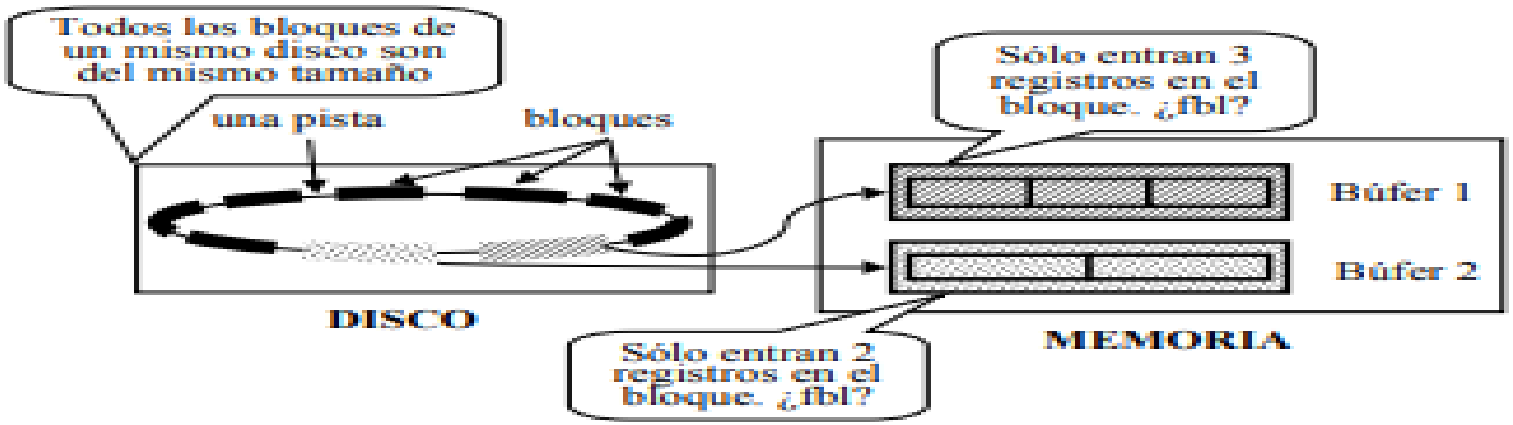
Programas:

trabajan con registros
situar los registros en bloques
Un bloque puede contener varios registros

Factor de bloqueo: registros que entran en un bloque B

fbl = (B/R) registros por bloque
B = nº bytes por bloque
R = nº bytes por registro (tamaño fijo)

Puede quedar espacio desocupado en los bloques, B - (fbl * R) bytes



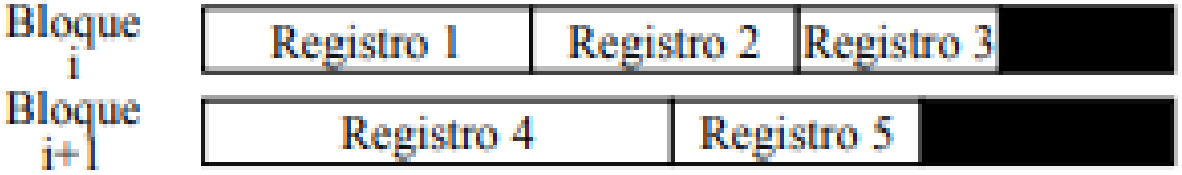
Organizaciones extendida y no extendida

Organización extendida:

Aprovecha los espacios desocupados en los bloques
Parte de un registro está en un bloque y el resto en otro
Un apuntador al final del bloque apunta al bloque donde continúa el registro
Es preciso usarla si el registro es mayor que el bloque

Organización NO extendida:

los registros no se parten en distintos bloques
El proceso de los registros es más simple



Técnicas de asignación de los bloques de un fichero en disco

Contigua: bloques consecutivos de disco. Facilita el doble búfer y dificulta el expandir el fichero

Enlazada: cada bloque tiene un apuntador al siguiente bloque. Facilita la expansión y dificulta la lectura

Grupos de bloques: combinación de las anteriores, un grupo de bloques consecutivos enlazado con el siguiente grupo consecutivo

Indexada: bloques que contienen sólo índices. Los índices son apuntadores a los bloques que realmente contienen los datos

Descriptor o cabecera de un fichero

Información sobre el fichero, necesaria para los programas que lo usan:

Direcciones en disco de los bloques del fichero
Descripciones de formato de los registros como:

Longitud y orden de campos en registros de longitud fija

Códigos de tipo de campo, separadores, códigos de tipo de registro, en registros de longitud variable

Se encuentra en bloques de disco asociados al fichero

Operaciones con ficheros

Varían de unos sistemas a otros.

Abrir:

Prepara el fichero para leer y asigna los búferes adecuados
Recupera el descriptor y pone el puntero de lectura al comienzo (primer registro)

Volver al principio:

Pone el puntero de lectura al comienzo (primer registro)

Buscar:

Busca el siguiente registro que cumple la condición.
Transfiere el bloque al búfer (si no estaba ya)
Localiza el registro y lo convierte en registro actual.

Leer:

Copia el registro actual del búfer en una variable del programa de usuario y avanza el puntero de lectura al siguiente registro.

Eliminar:

Elimina el registro actual
Actualiza el registro en el disco

Operaciones con ficheros (2)

Modificar:

Modifica valores en el registro actual
Actualiza el registro en el disco

Insertar:

Localiza el bloque donde insertar
Transfiere el bloque al búfer
Escribe el registro nuevo en el búfer
Escribe el búfer en el disco

Cerrar:

Libera los búferes
Realiza operaciones de limpieza

Organización y método de acceso

Organización del fichero:

Cómo se colocan los registros y bloques

Cómo interconectan

Método de acceso:

Programas que permiten realizar ciertas operaciones sobre el fichero

Es posible aplicar varios métodos de acceso a una misma organización de fichero

Hay métodos de acceso que NO se pueden aplicar a algunas organizaciones de fichero

Interesa encontrar la organización idónea en cada caso

Será la que permita realizar de manera más eficiente las operaciones más frecuentes

En muchos casos ninguna organización permite que todas esas operaciones se implementen eficientemente.

Entonces se elige una solución de compromiso entre la importancia de lo que se espera y la mezcla de operaciones de recuperación y actualización

Ficheros de ficheros de montón

Ficheros de registros NO ordenados (secuenciales)

Organización más simple:

Registros en el orden en el que se insertan
Inserciones siempre al final del fichero

Inserción

Poner en el búfer el último bloque del fichero
Añadir al búfer el registro a insertar
Reescribir el búfer en el disco
Guardar la dirección del bloque en el descriptor

Búsqueda

Lineal: bloque a bloque hasta que un registro cumpla la condición (costoso)
Costo medio: acceso a bloques del fichero / 2
Costo caso peor
Ningún registro cumple la condición
Hay que obtener todos los que la cumplen

Eliminación en ficheros de montón

Eliminación

Buscar el registro a borrar
Transferir el bloque al búfer
Eliminar el registro del búfer
Reescribir el búfer en el disco

Problema: deja espacios desocupados en el disco (pueden ser muchos, si se borran muchos registros)

Otra técnica: usar un bit del registro como marcador de eliminación y usar sólo los NO borrados

Solución 1: reorganización periódica del fichero (para recuperar el espacio desocupado)

Recorrer secuencialmente los bloques
Recolocar los registros quitando los huecos

Solución 2: intentar insertar sobre huecos y si no se puede insertar al final

Cálculos adicionales para encontrar bloques con huecos

Modificación en ficheros de montón

Modificación

Buscar el registro a modificar
Transferir el bloque al búfer
Modificar el registro en el búfer
Reescribir el búfer en el disco

Problema

registros de tamaño variable

Solución:

Eliminar el registro a modificar
Insertar el registro modificado

Caso especial de ficheros de montón: ficheros relativos o directos

Con registros de longitud fija

NO ordenados

Bloques NO extendidos y de asignación continua

En este caso es muy sencillo tener acceso a cualquier registro por la posición que ocupa en el fichero:

Los registros se numeran como 0, 1, 2, 3, ..., r-1
Los registros en cada bloque se numeran como 0, 1, ..., fbl-1

Así el registro i-ésimo estará en:

Bloque (i/fbl)
Registro (i mod fbl) del bloque (mod es el resto de la división)

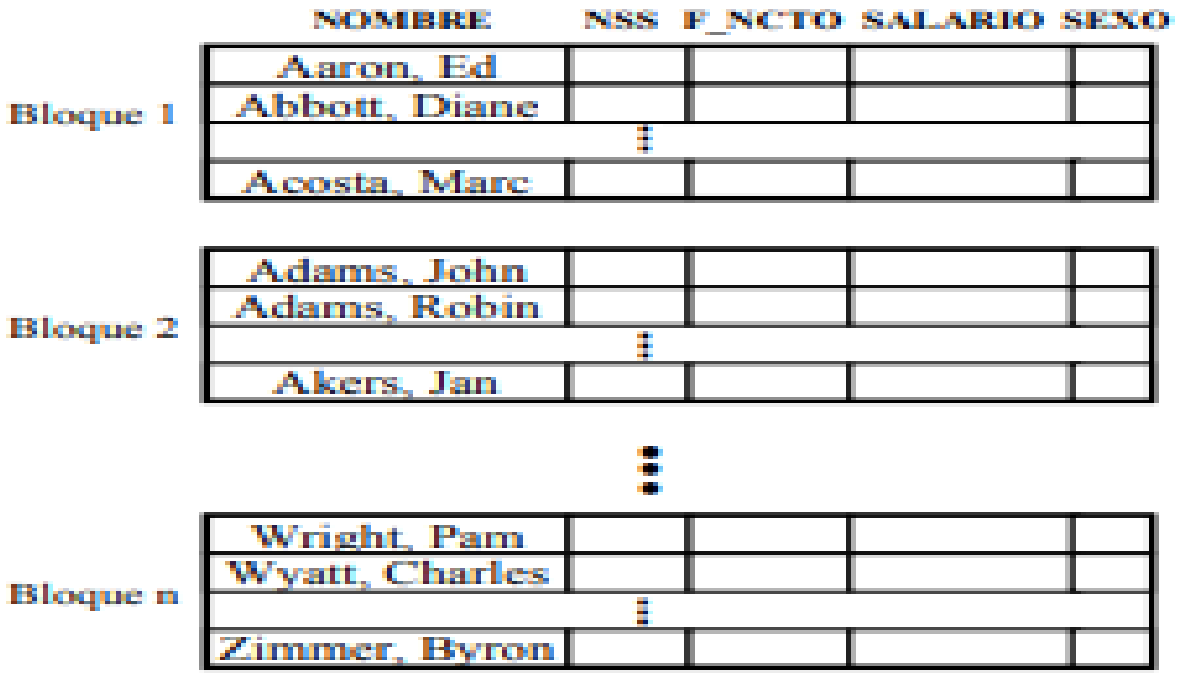
Esta idea no ayuda a localizar un registro según una condición de búsqueda

Ficheros ordenados

Secuencial o secuencial ordenado

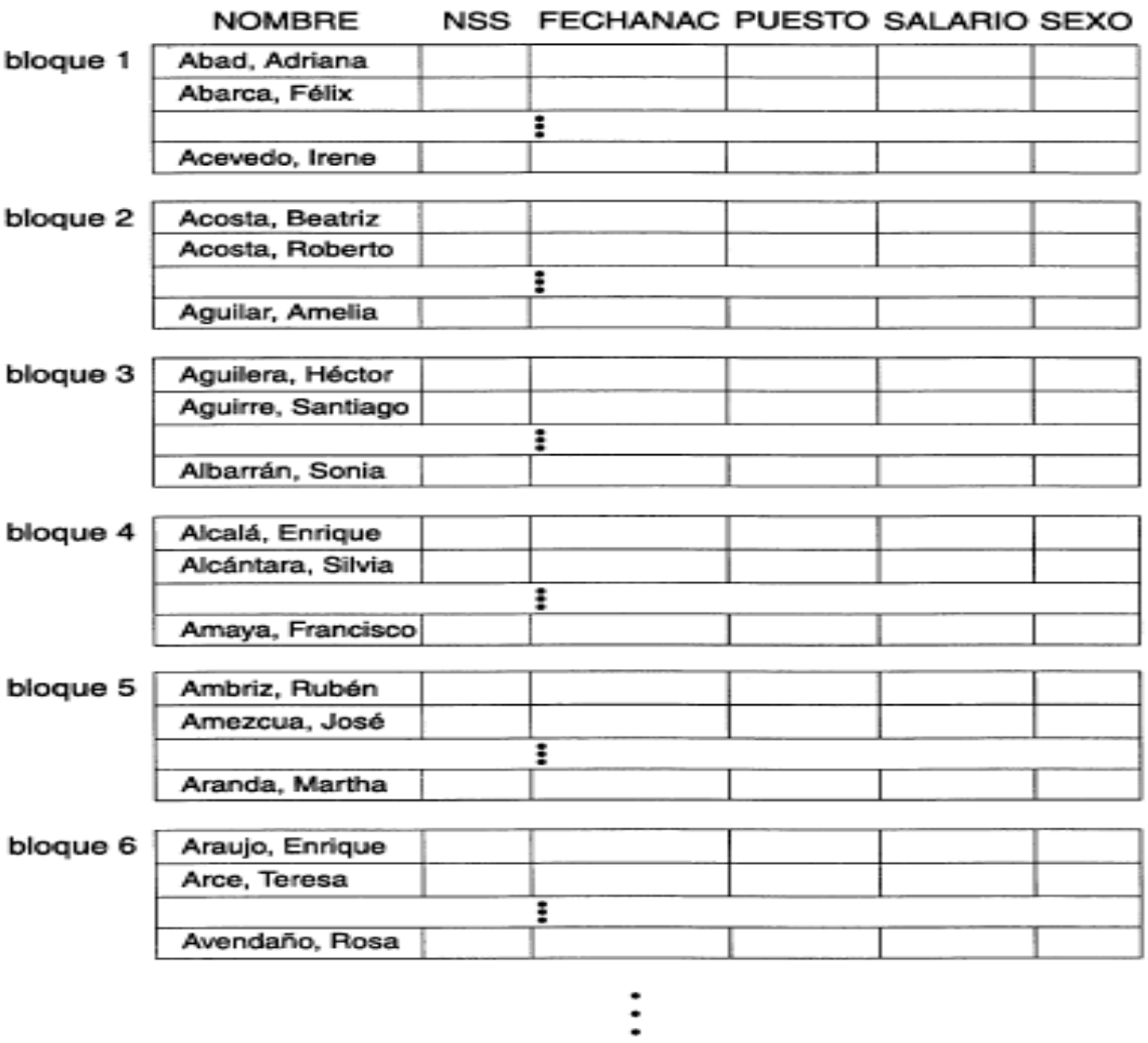
Registros se mantienen físicamente ordenados según los valores de un campo: el campo de ordenación

Si el campo de ordenación es clave, se llama clave de ordenación



Bloques de un fichero ordenado (secuencial) de registros EMPLEADO con NOMBRE como campo clave de ordenación

Bloques de archivo ordenado con Nombre como campo de ordenación



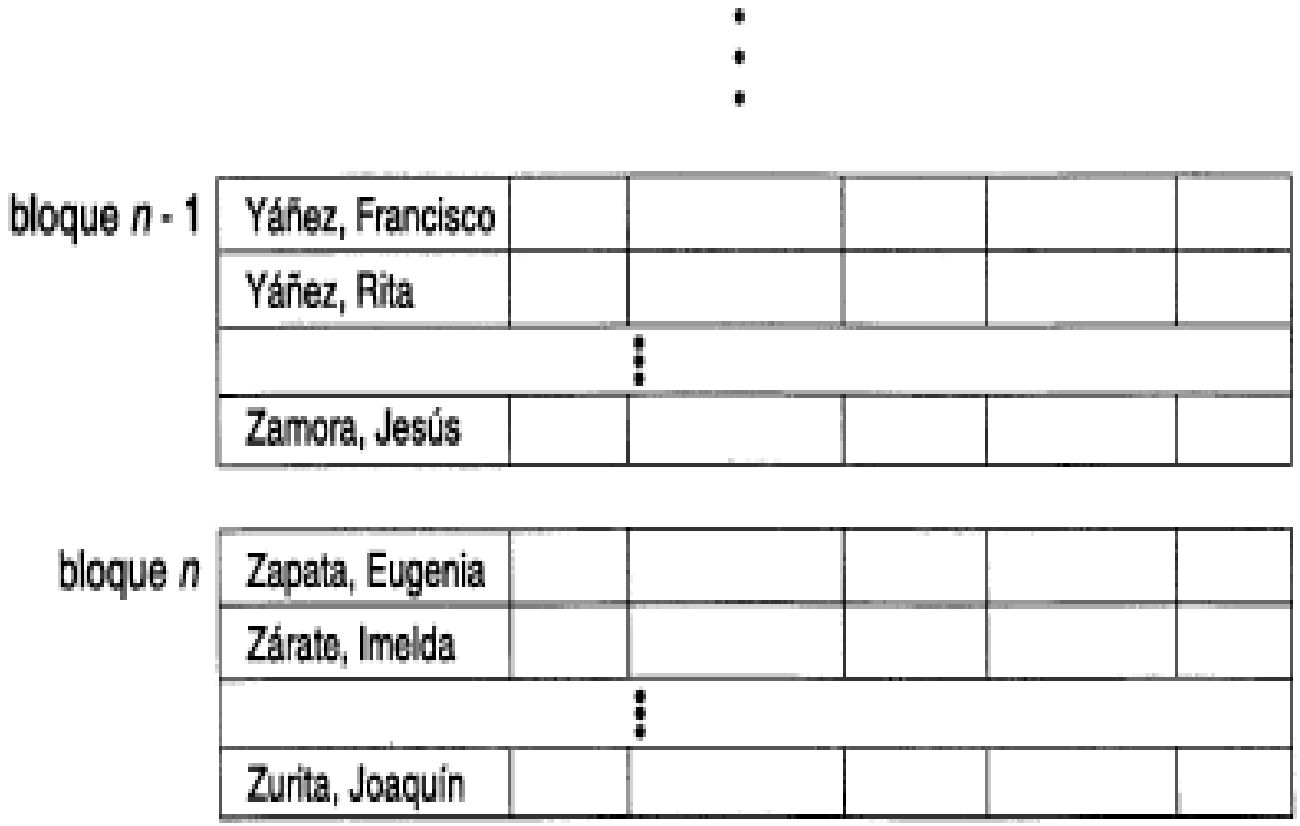


Figura 4.7 Bloques de un archivo ordenado (secuencial) de registros EMPLEADO, con NOMBRE como campo de ordenación.

Ficheros ordenados: ventajas

Lectura eficiente en orden del campo de ordenación

Acceso eficiente al siguiente en el orden del campo de ordenación:

Estará en el búfer (si quedan registros por leer)
Si no, trae el siguiente bloque (si hay)

Posibilidad de búsqueda binaria:

Mucho más rápida
Sólo si la condición de búsqueda se basa en un valor del campo clave de ordenación

Búsqueda binaria (dicotómica)

Se efectúa sobre bloques
Ejemplo: bloques 1, 2, …, b
Las direcciones de 1, 2, ..., b, están en el descriptor
Se desea encontrar un registro con valor K en su campo clave de ordenación
Costo medio: acceso a log2(b) bloques
Costo medio con búsqueda lineal: b/2 (b si no está)

También se puede buscar por , <=, >= sobre el campo de ordenación.
Estos ficheros NO ofrecen ventajas para búsquedas por campos distintos al de ordenación

Ficheros ordenados: operaciones

Se debe mantener el orden de los registros

Inserción y eliminación: operaciones costosas

Inserción

- Buscar el bloque donde insertar en orden el campo de ordenación
- Abrir espacio, desplazando el resto de registros del fichero
- Media: lectura y reescritura de b/2 bloques !!
- Insertar el nuevo registro y reescribir el bloque

Alternativas:

- Bloques con espacio desocupado: cuando se llena tenemos el mismo problema
- Fichero de desbordamiento:
  - NO está ordenado
  - Los nuevos registros se insertan al final
  - Periódicamente: reconstruir el fichero ordenado, colocando en su posición los registros del fichero de desbordamiento
  - Inserción más eficiente
  - Búsqueda más ineficiente

Ficheros ordenados: operaciones (2)

Eliminación

Similar a la inserción para la eliminación física

Más sencillo si sólo se marca como borrado y se reorganiza periódicamente

Modificación

Según el caso podrá realizarse búsqueda binaria

Si se modifica el campo de ordenación puede precisar cambiar de posición el registro. Eso supone hacer una eliminación y una inserción

Si los registros son de longitud fija se podrá reescribir el bloque en la misma posición

Uso de ficheros ordenados en BD

Estos ficheros se usan poco en aplicaciones de BD

Se usan más con un índice primario: ficheros secuenciales indexados

Técnicas de direccionamiento calculado (hashing)

Acceso muy rápido para ciertas búsquedas

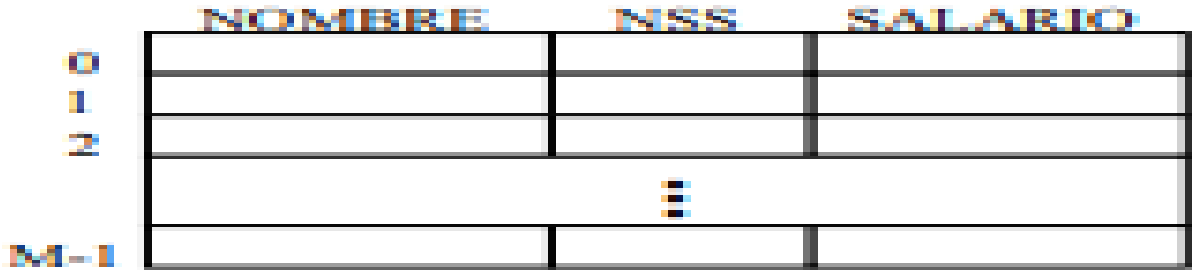
Los ficheros de direccionamiento calculado se llaman también ficheros dispersos o directos

- Condición de búsqueda: igualdad sobre el campo de direccionamiento calculado (c.dir.cal.)
- El c.dir.cal. en la mayor parte de los casos es clave (valores únicos)
- Función de direccionamiento calculado o de aleatorización (f. hash)
  - Se aplica a valores del c.dir.cal. del registro
  - Resultado: dirección de un bloque de disco
  - En el bloque está el reg. con ese valor de c.dir.cal.
  - El registro se busca en el búfer
  - Suele bastar con un acceso a bloques de disco

Direccionamiento calculado interno:

- Estructura de datos (en memoria principal) muy útil en programación
- A continuación se hace un paréntesis para repasar esta estructura de datos (no se trata de ficheros)
- Los ficheros de direccionamiento calculado de los que se hablaba en esta página los estudiamos después como direccionamiento calculado externo

Direccionamiento calculado interno



Array de 0..M-1 registros

Elegir función de direccionamiento calculado: h(k)= k mod M

k real: conversión previa a entero

k string: conversión a entero usando los códigos

Técnicas para las funciones de direccionamiento calculado:

- Plegado (folding): operación aritmética (ej. suma) ó lógica (ej. O exclusiva) en partes del c.dir.cal. para calcular la dirección
- Escoger dígitos

Problemas con la mayoría de funciones de direccionamiento calculado:

- No garantiza direcciones distintas a claves distintas
- Espacio c.dir.cal. >> 0 . . M-1 – Colisión: la dirección obtenida ya está ocupada
- Búsqueda de otra dirección: resolución de colisiones

Direccionamiento calculado interno (2) Métodos para resolución de colisiones

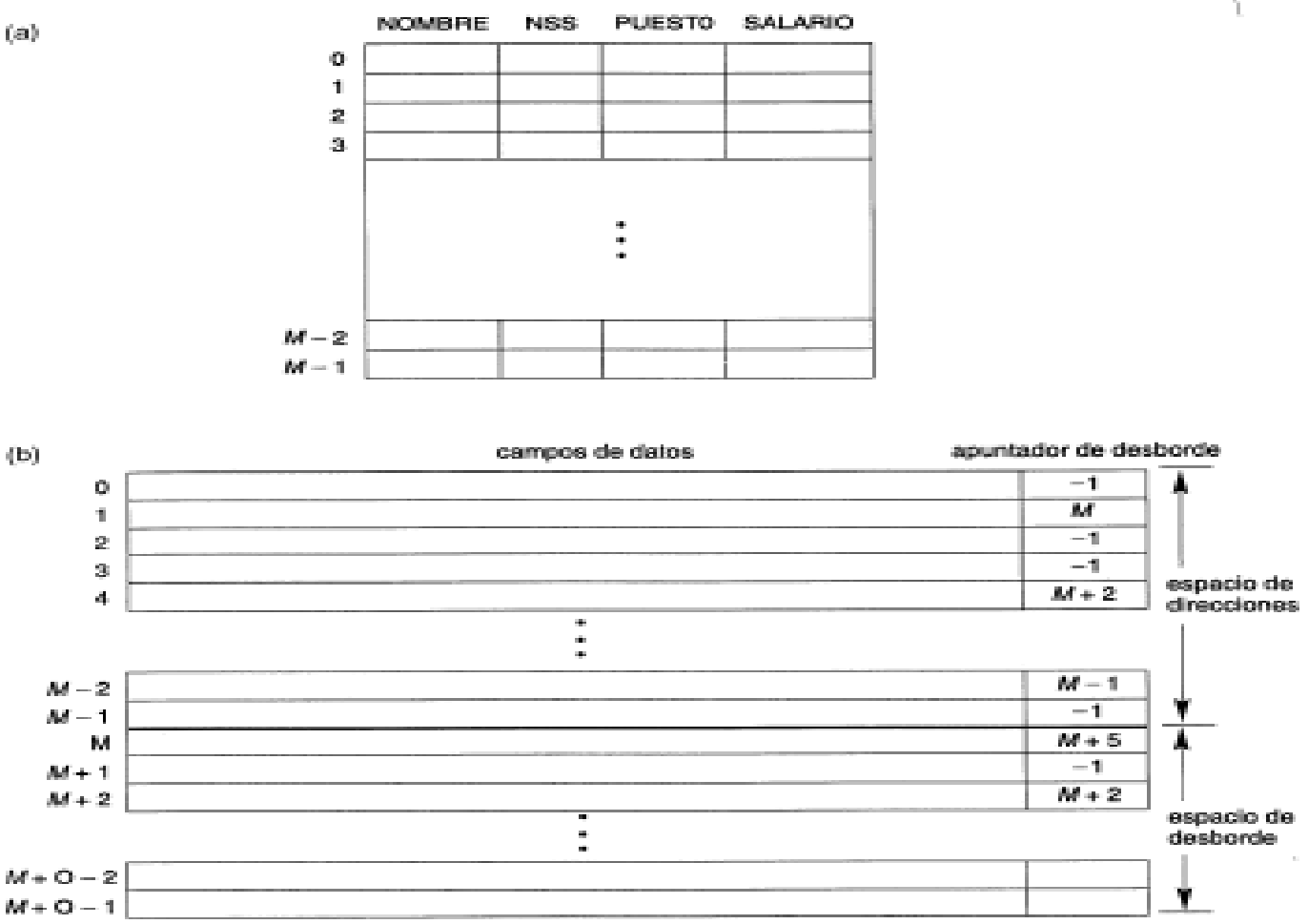
Direccionamiento abierto:
búsqueda uno a uno en las direcciones siguientes a la devuelta por la función de direccionamiento calculado hasta encontrar un hueco libre

Encadenamiento:
Con áreas de desbordamiento
Registros con un apuntador (campo nuevo)
Registros de igual dirección: encadenados

Fig. 5.10 (b) Resolución de colisiones por encadenamiento

Direccionamiento calculado múltiple:
2ª función de direccionamiento calculado en caso de colisión
Caso de una 2ª colisión: uso de 3ª función
Al final se usa direccionamiento abierto

Dispersión Interna



Estructuras de datos de dispersión interna. (a) Arreglo de M posiciones que se usará para la dispersión interna.
(b) Resolución de colisiones por encadenamiento de registros.

Direccionamiento calculado interno (3)

Buscar, insertar y eliminar:

- Encadenamiento: algoritmos más simples
- Direccionamiento abierto: eliminación complicada

Objetivo de una buena función de direccionamiento calculado:

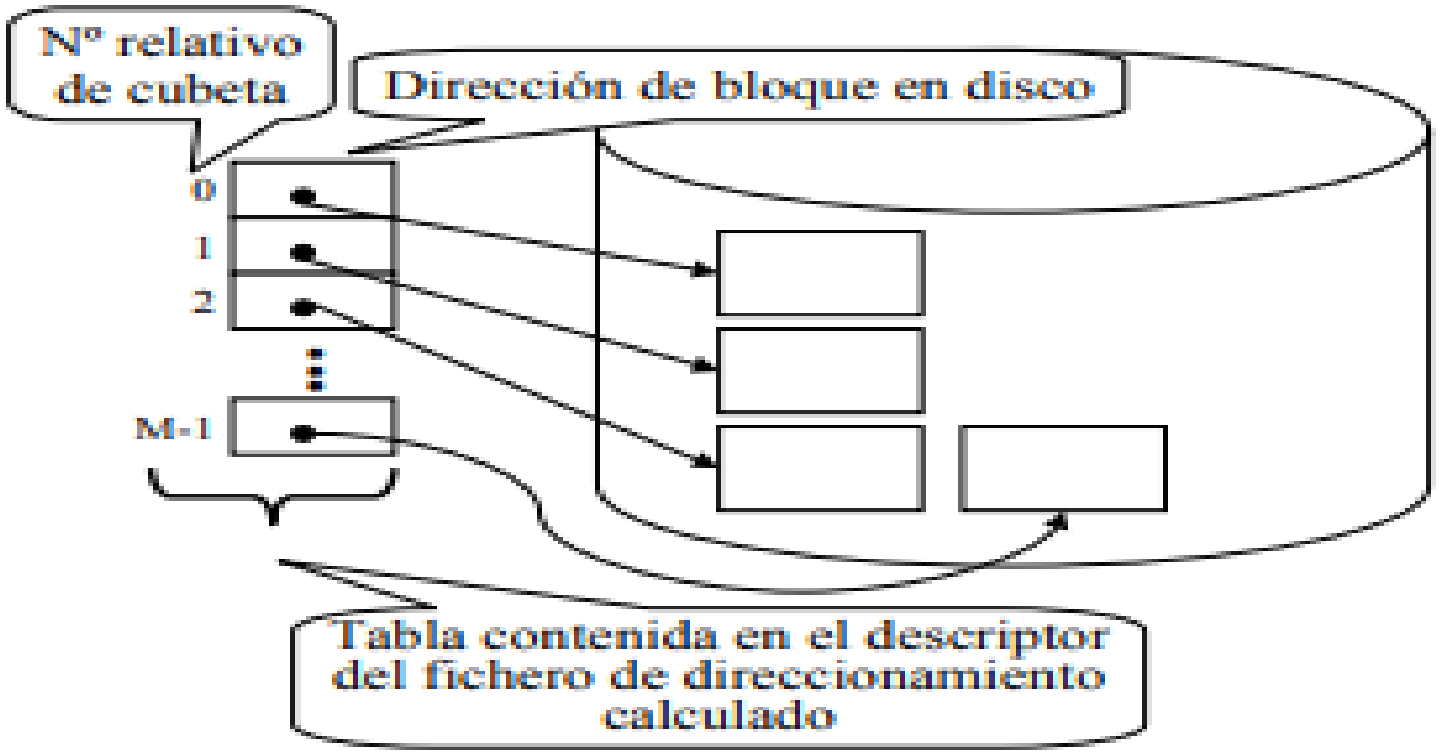
- Distribuir uniformemente
- Mínimo de colisiones
- Máxima ocupación de posiciones en el array

Resultado de simulaciones y análisis:

- Mejor 70%-90% de ocupación:
  - Número bajo de colisiones
  - Poco espacio desperdiciado
- Funciones con mod:
  - Mejor distribución si M es primo
- Algunas funciones:
  - Conviene que M=2k

Direccionamiento calculado externo

- En ficheros de disco
- Proporciona el acceso más rápido a un registro concreto (por c.dir.cal.): en general un acceso a bloque
- Dirección: nº relativo de cubeta
- Una cubeta consta de 1 o varios bloques contiguos
- Tabla en descriptor: convierte el nº relativo de cubeta en dirección de bloque en disco



Correspondencia entre número de cubeta y dirección de bloque en disco

Dispersión Externa

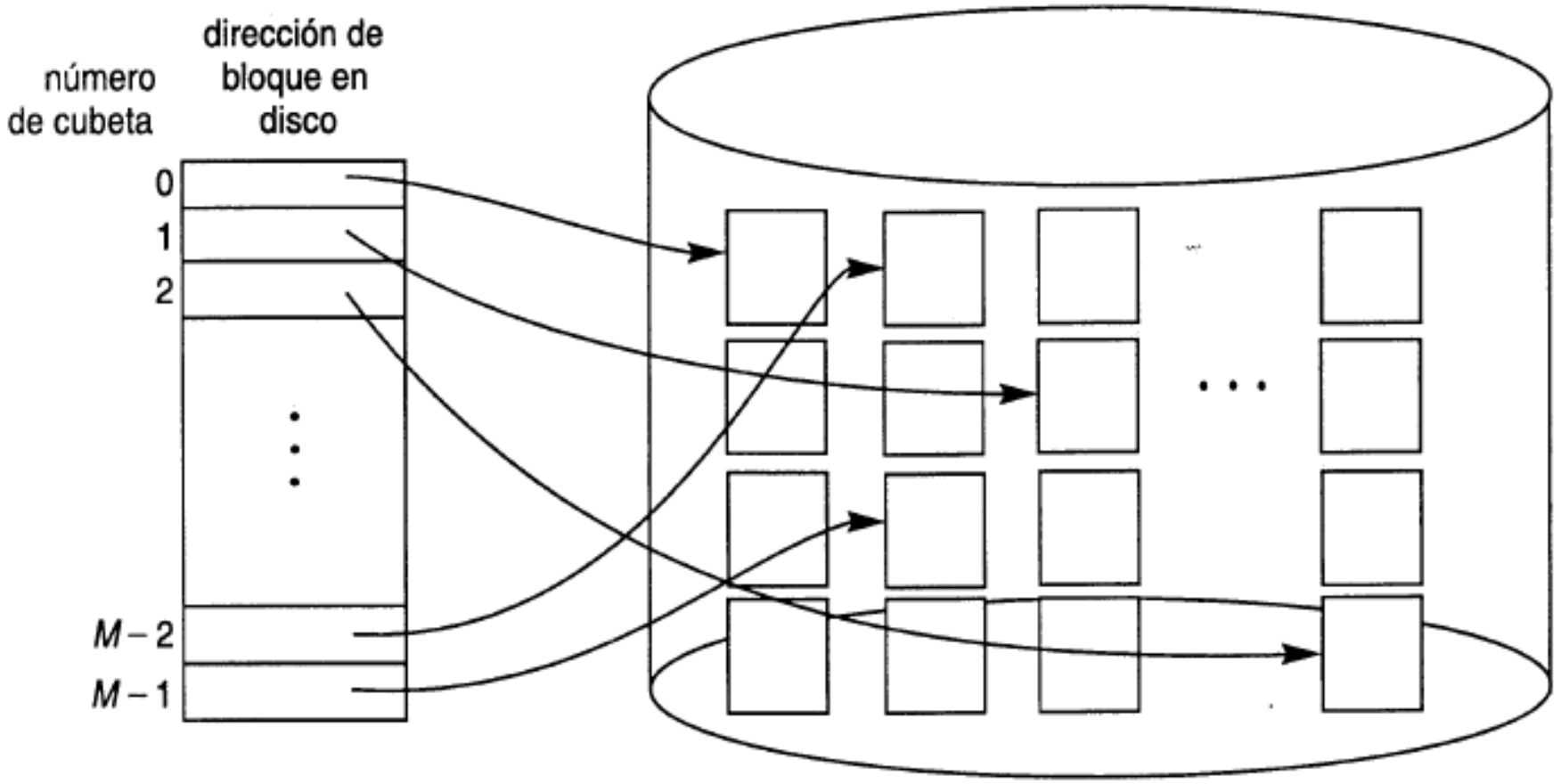
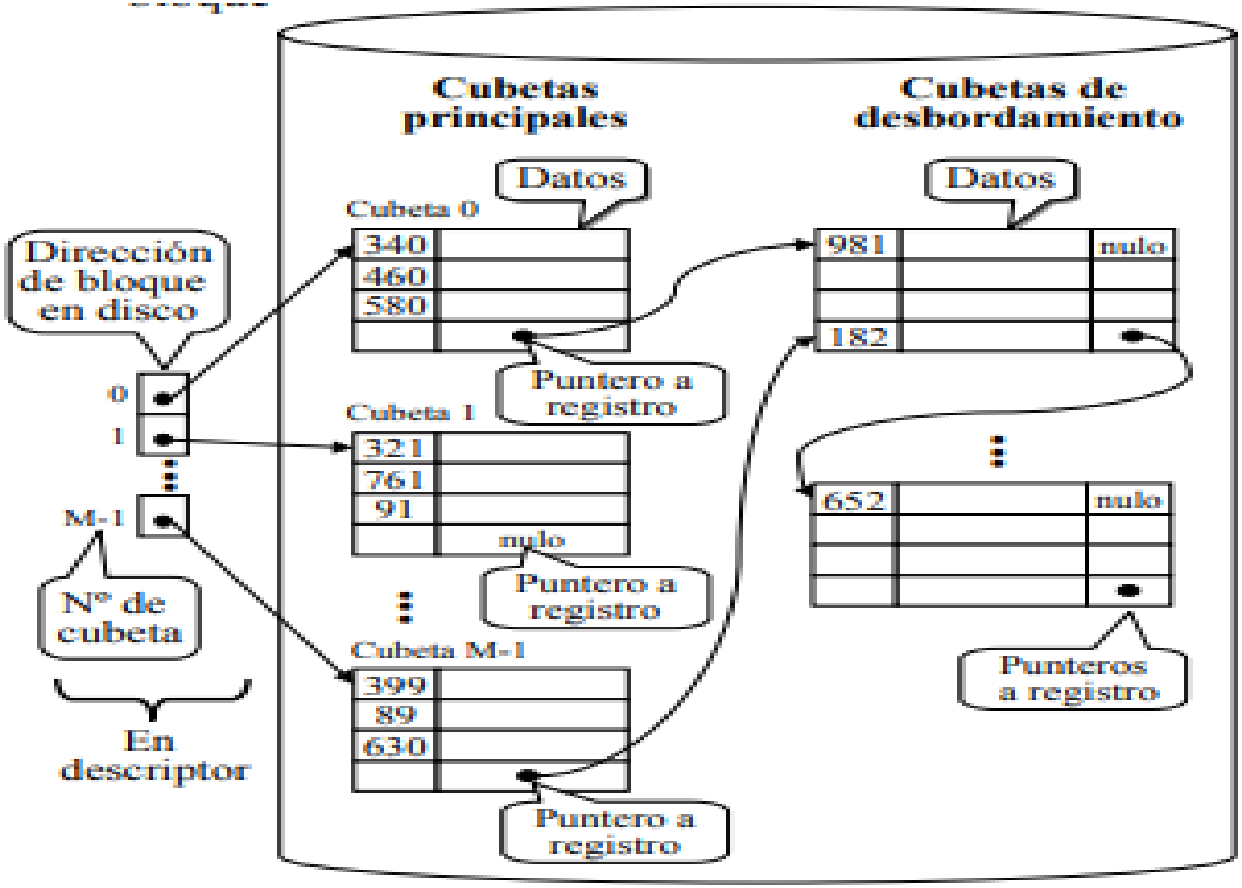


Figura 4.9 Correspondencia entre números de cubeta y bloques de disco.

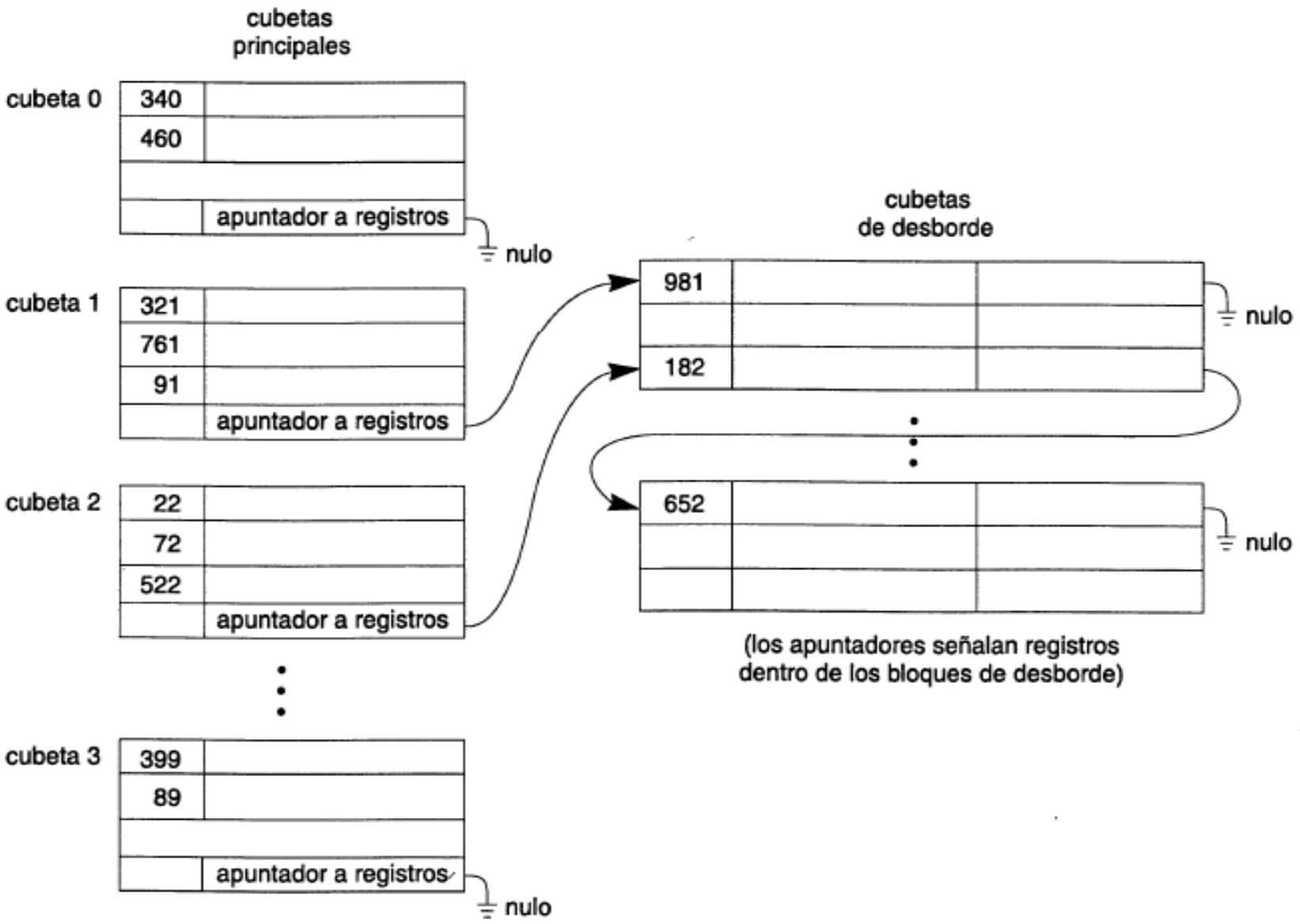
Direccionamiento calculado externo (2)

Colisiones: sin problema hasta llenar cubeta
Cubeta llena: lista enlazada de cubetas de desborde
Punteros: dirección de bloque + nº de registro en el bloque

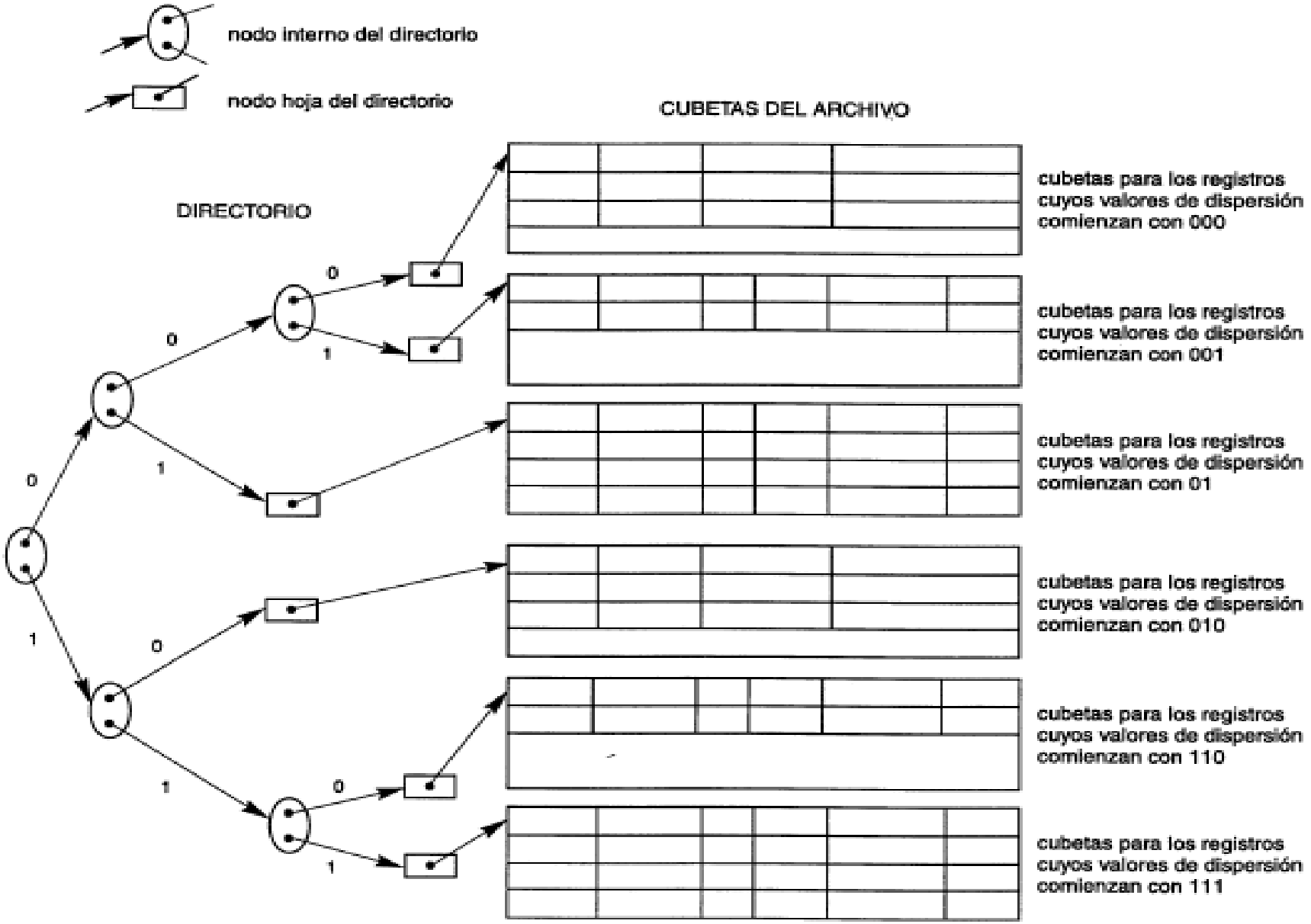


Manejo de desbordamiento de cubeta por encadenamiento

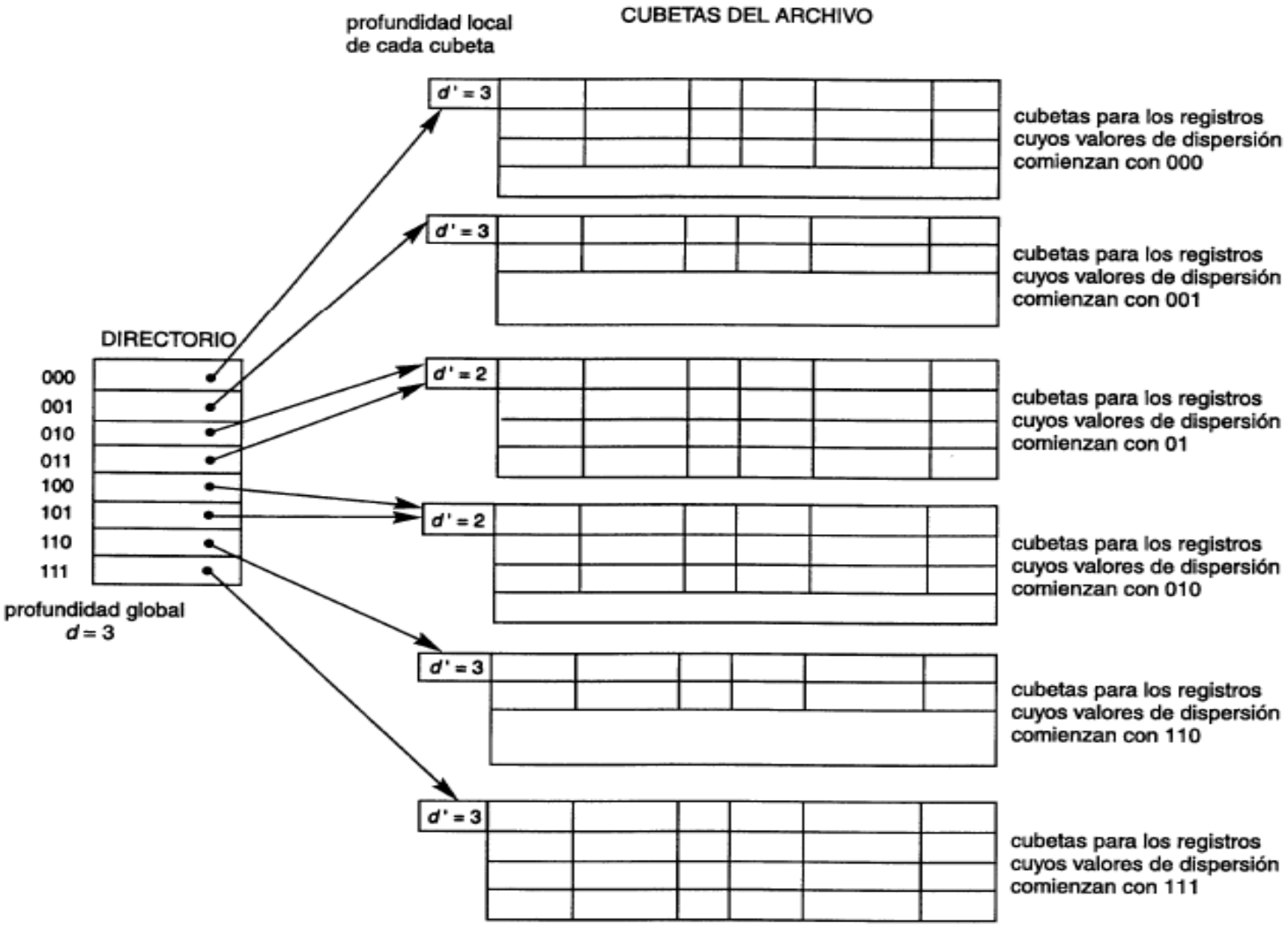
Desborde de cubetas por encadenamiento



Dispersión Dinámica



Dispersión Extensible



Problemas y operaciones en direccionamiento calculado externo

Problemas:

Las funciones de direccionamiento calculado, en general, NO preservan el orden de la clave

Cantidad de espacio prefijada:

M cubetas * m registros/cubeta

Si nº reg. mucho menor: desperdicio de espacio

Si es bastante mayor:

Muchas colisiones

Listas de cubetas de desbordamiento largas

En ambos casos es preferible:

Cambiar el espacio asignado

Cambiar la función de dir. calculado

Recolocar los registros

Operaciones:

Búsqueda:

Por campo ≠ c.dir.cal. es tan costoso como en fichero no ordenado

Eliminación:

Cubeta: pasar un registro del área de desbordamiento

Área de desbordamiento: eliminación en lista enlazada

Modificación:

C.dir.cal: eliminar e insertar

Otro campo: reescritura

Ficheros de registros mixtos
Hasta aquí: ficheros con todos sus registros del mismo tipo

Campos de conexión:

Para representar relaciones (vínculos): claves extranjeras en el modelo relacional y referencias a objeto en el modelo orientado a objetos (OO)

Constituyen referencias lógicas de campos entre registros de ficheros distintos

Modelos OO, jerárquico y en red:

Relaciones físicas: registros físicamente contiguos o mediante punteros

Suelen asignar un área de disco que almacena registros de distintos tipos, que pueden estar agrupados físicamente

Fichero mixto (registros de varios tipos):

En OO contiene agrupados físicamente objetos relacionados

Con un campo para almacenar el tipo de registro

El SGBD una vez determinado el tipo de registro utiliza el catálogo para conocer los campos del registro y sus tamaños

Estructuras de índices para archivos

Índices

Estructuras de acceso que aceleran la obtención de registros (para ciertas condiciones de búsqueda)

Se distinguen cuatro tipos de índice:

- Índices primarios
- Índices secundarios sobre clave
- Índices secundarios sobre no clave
- Índices de agrupación (sobre no clave)

Índices

Los índices primarios y los de agrupación exigen que la organización primaria sea fichero ordenado (por el campo o campos de indexación)

Los índices secundarios (o caminos de acceso secundario) admiten cualquier organización primaria

Un fichero puede tener varios índices pero sólo puede tener uno primario o uno de agrupación. El resto serán secundarios.

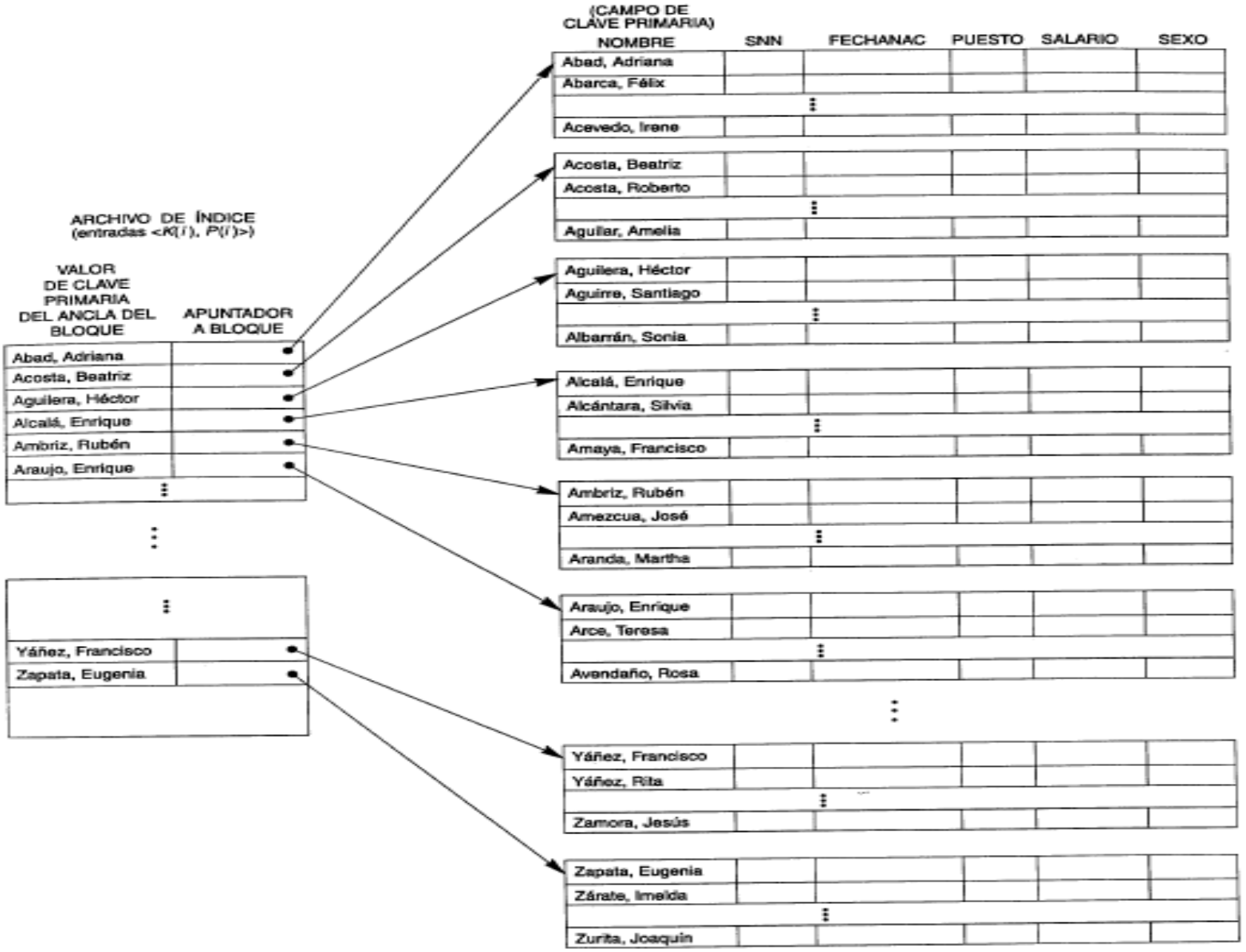
Se puede construir un índice sobre cualquier:

- Campo del fichero
- Conjunto de campos del fichero

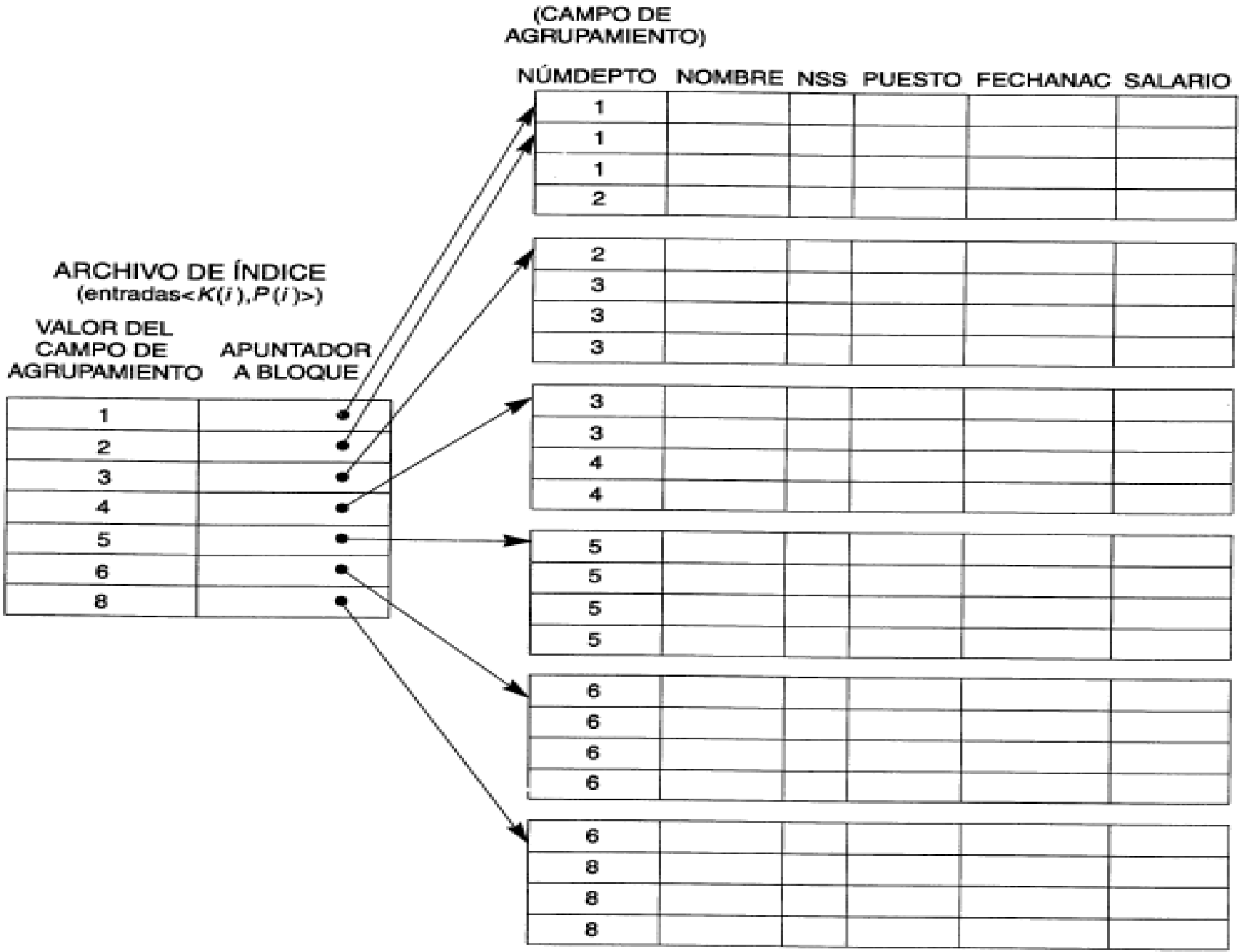
También se distingue entre índices:

- Físicos (con punteros)
- Lógicos (con valores de clave primaria)

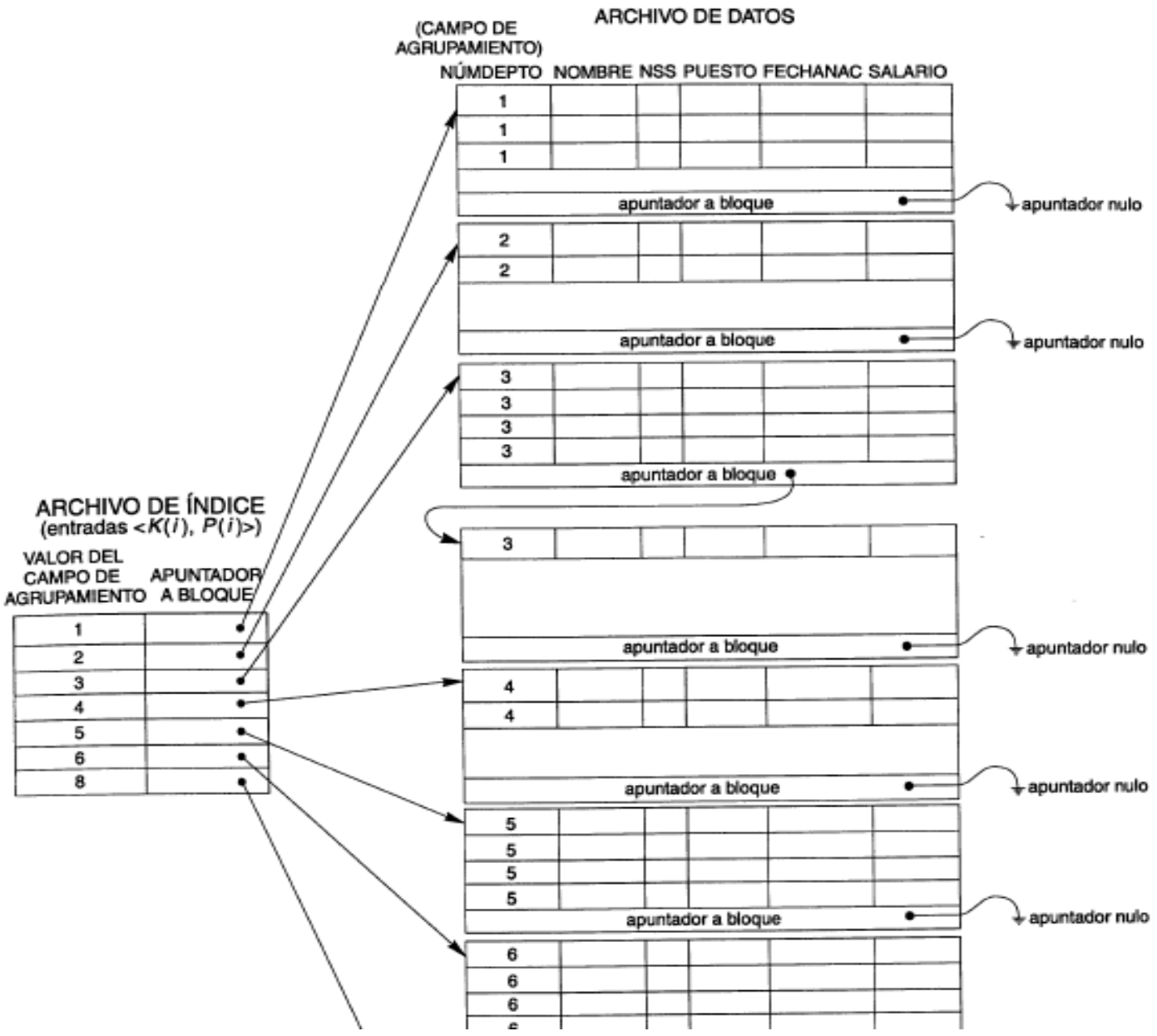
Índice primario sobre sobre campo clave de ordenamiento



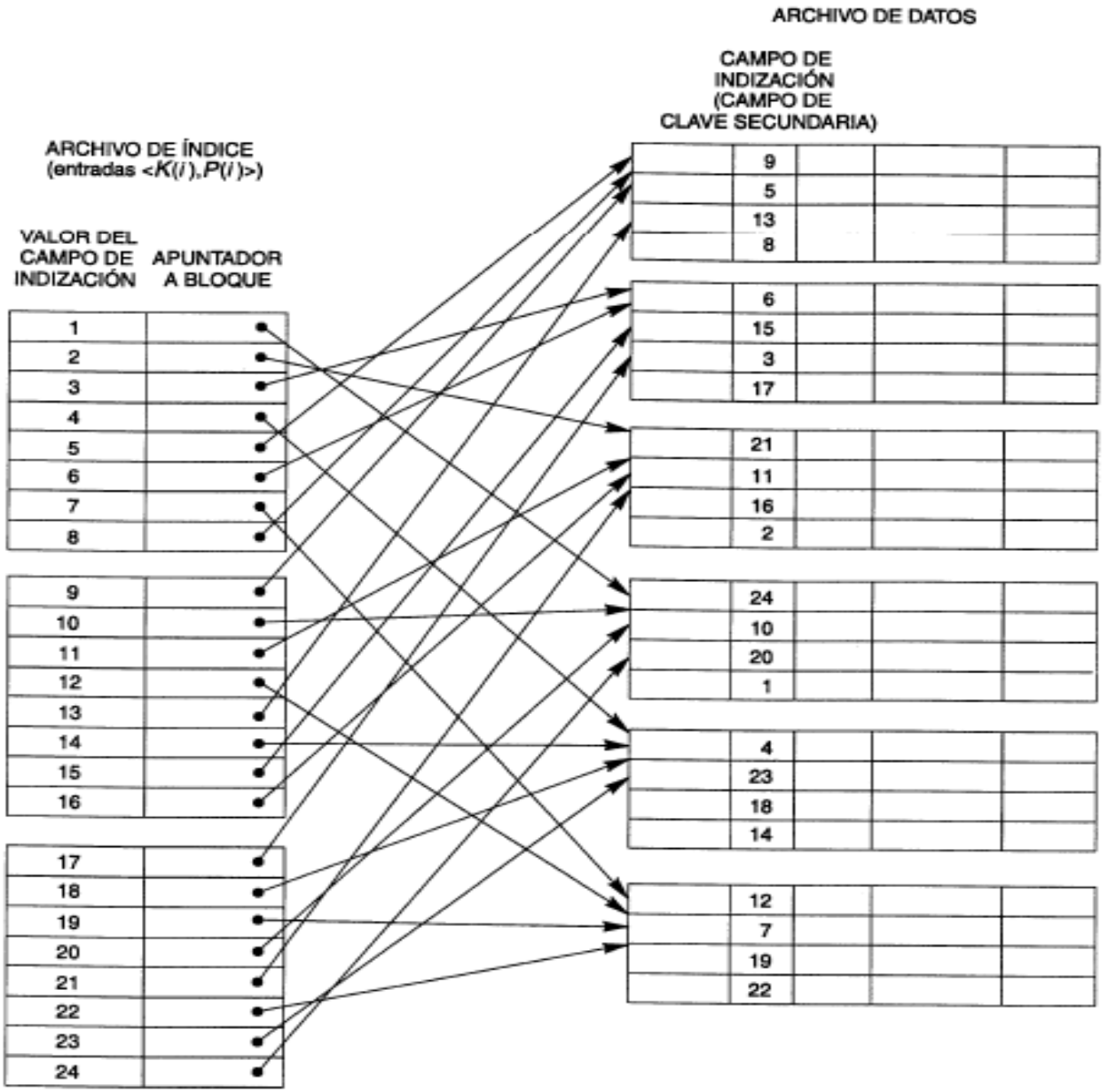
Índice de agrupamiento según campo clave de ordenamiento



Índice de agrupamiento con bloques individuales para cada grupo de registros que comparten el mismo valor del campo de agrupamiento



Índice secundario denso según un campo clave que no determina el ordenamiento del archivo



Índice secundario según un campo no clave, implementado con un nivel de indirección para que las entradas del índice tengan longitud fija y valores únicos en los campos

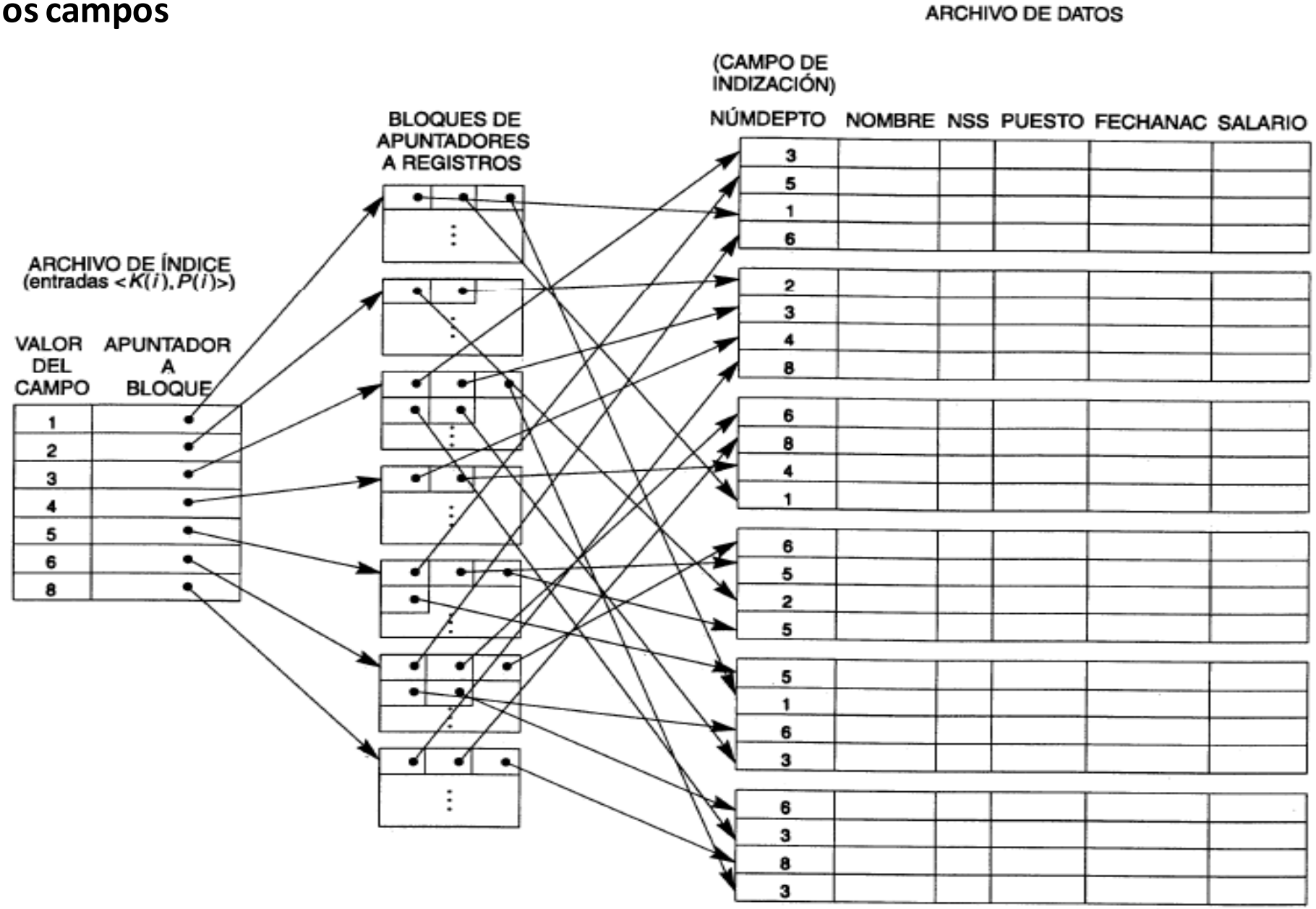


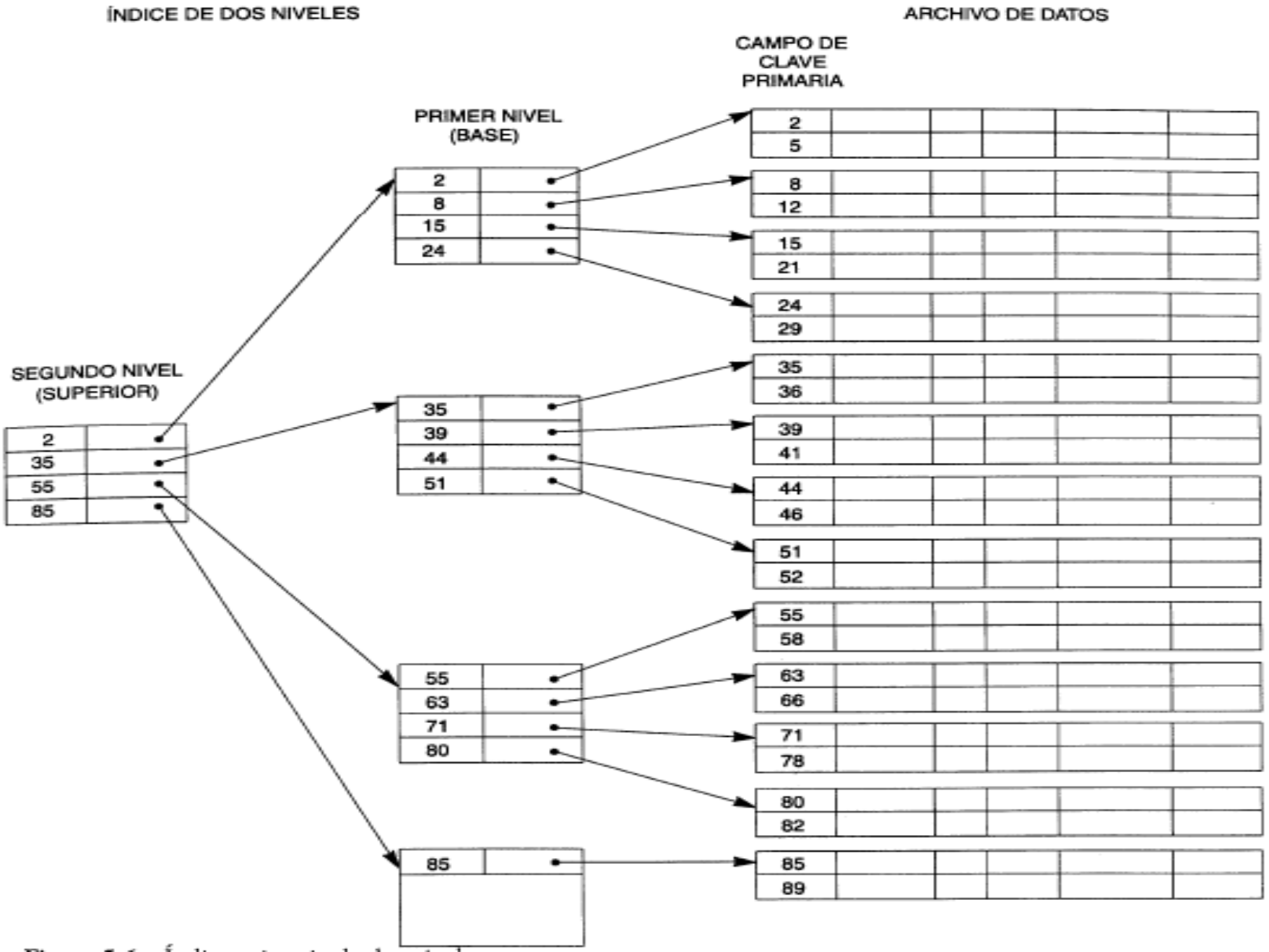
Tabla 5.1 Tipos de índices

	Campo de ordenación	Campo no de ordenación
Campo clave	Índice primario	Índice secundario (clave)
Campo no clave	Índice de agrupamiento	Índice secundario (no clave)

Tabla 5.2 Propiedades de los tipos de índices

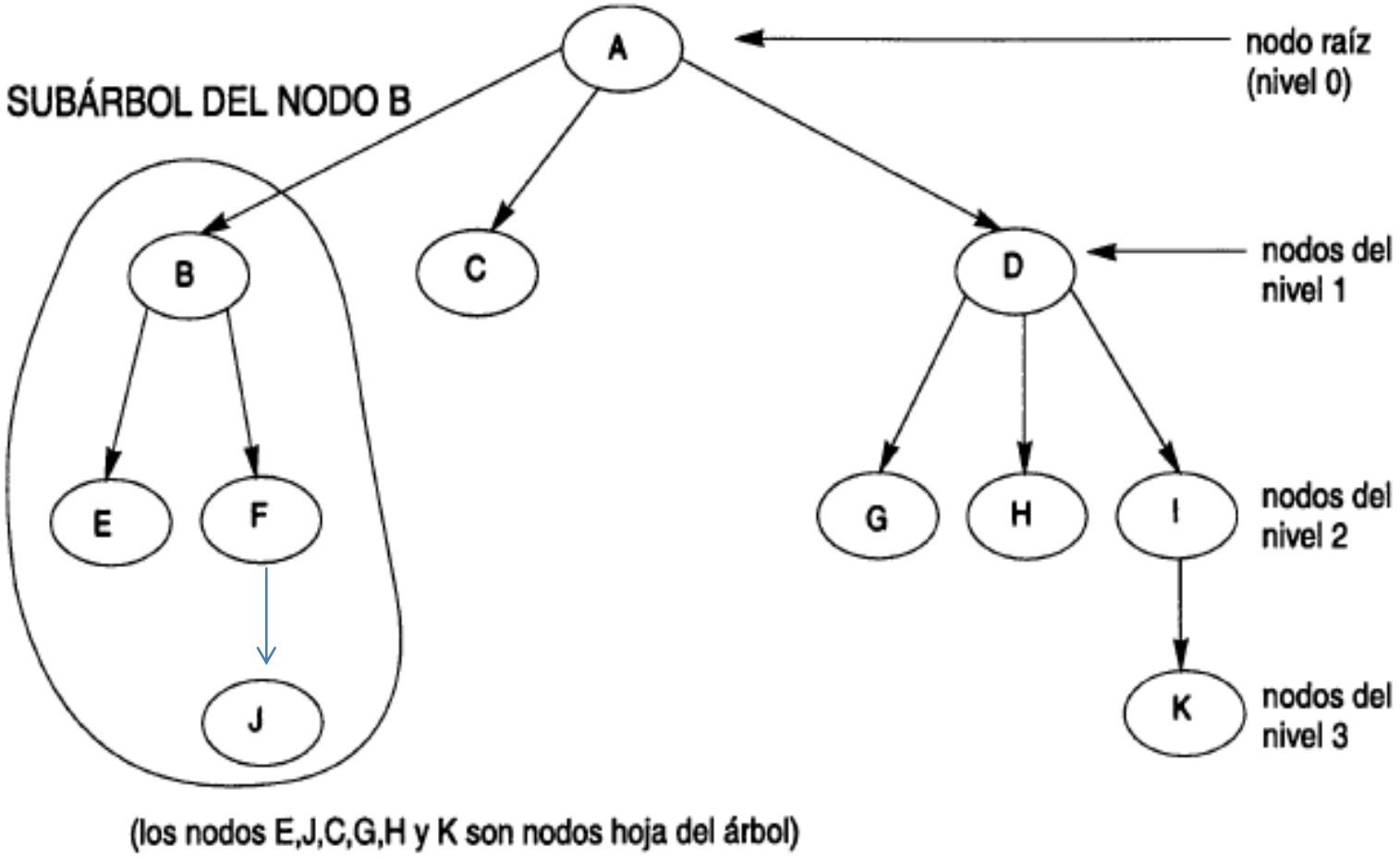
		Propiedades del tipo de índices		
		Número de entradas de índice (primer nivel)	**Denso o no denso**	**Anclas de bloques del archivo de datos**
Tipo de índice	Primario	Número de bloques del archivo de datos	No denso	Sí
	De agrupamiento	Número de valores distintos del campo índice	No denso	Sí/no[a]
	Secundario (clave)	Número de registros del archivo de datos	Denso	No
	Secundario (no clave)	Número de registros[b] o número de valores distintos del campo índice[c]	Denso o no denso	No

Índice de múltiples niveles – Índice primario de dos niveles

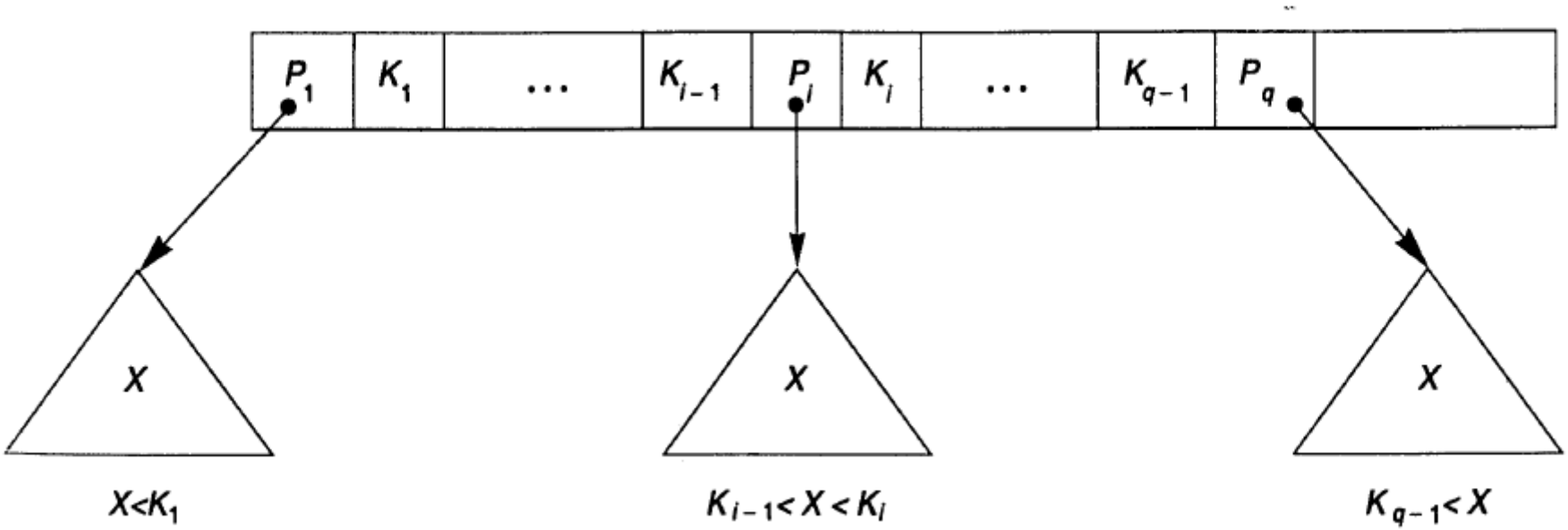


Índices dinámicos de múltiples niveles con base a árboles B y B+

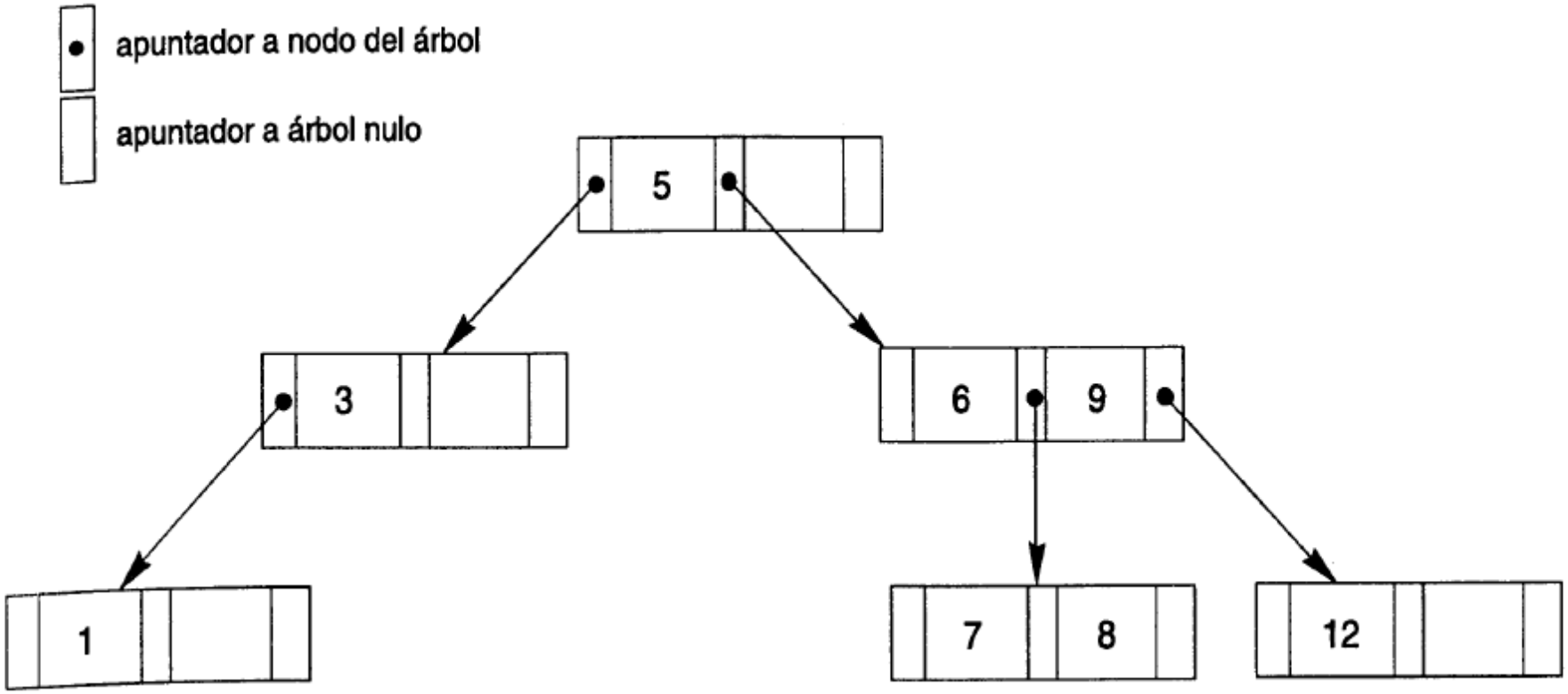
Estructura de datos de un árbol



Nodo de un árbol de búsqueda



Árbol de búsqueda de orden p = 3



1. Cada nodo interno del árbol B (Fig. 5.10(a)) tiene la forma

$$<P_1, <K_1, Pr_1>, P_2, <K_2, Pr_2>, \ldots, <K_{q-1}, Pr_{q-1}>, P_q>$$

donde $q \leq p$. Cada P_i es un **apuntador de árbol**: un apuntador a otro nodo del árbol B. Cada Pr_i es un **apuntador de datos**:[†††] un apuntador al registro cuyo valor del campo clave de búsqueda es igual a K_i (o un apuntador al bloque que contiene ese registro en el archivo de datos).

2. Dentro de cada nodo, $K_1 < K_2 < \cdots < K_{q-1}$.
3. Para todos los valores X del campo clave de búsqueda en el subárbol al que apunta P_i, tenemos

$$K_{i-1} < X < K_i, \text{ para } 1 < i < q, \; X < K_i, \text{ para } i = 1, \text{ y } K_{i-1} < X, \text{ para } i = q$$

(véase la Fig. 5.10(a)).

4. Cada nodo tiene cuando más p apuntadores de árbol.
5. Cada nodo, excepto la raíz y los nodos hoja, tienen por lo menos $\lceil (p/2) \rceil$ apuntadores de árbol. El nodo raíz tiene como mínimo dos apuntadores de árbol, a menos que sea el único nodo del árbol.
6. Un nodo con q apuntadores de árbol, $q \leq p$, tiene $q - 1$ valores del campo clave de búsqueda (y por tanto tiene $q - 1$ apuntadores de datos).
7. Todos los nodos hoja están en el mismo nivel. Los nodos hoja tienen la misma estructura que los internos, excepto que todos sus *apuntadores de árbol*, P_i, son **nulos**.

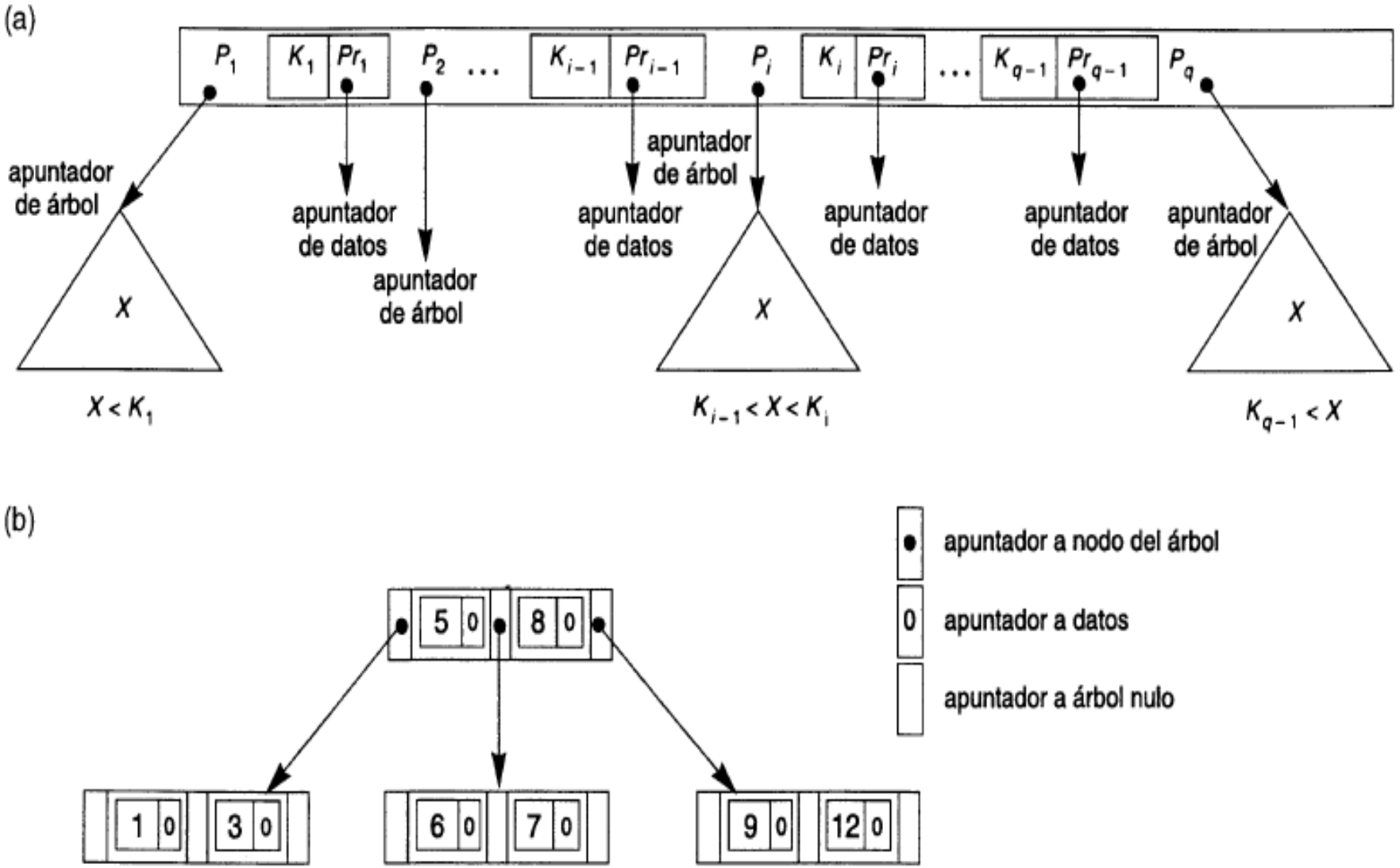


Figura 5.10 Estructuras de árbol B. (a) Nodo de árbol B con $q - 1$ valores de búsqueda. (b) Árbol B de orden $p = 3$. Los valores se insertaron en el orden 8, 5, 1, 7, 3, 12, 9, 6.

Árbol B+

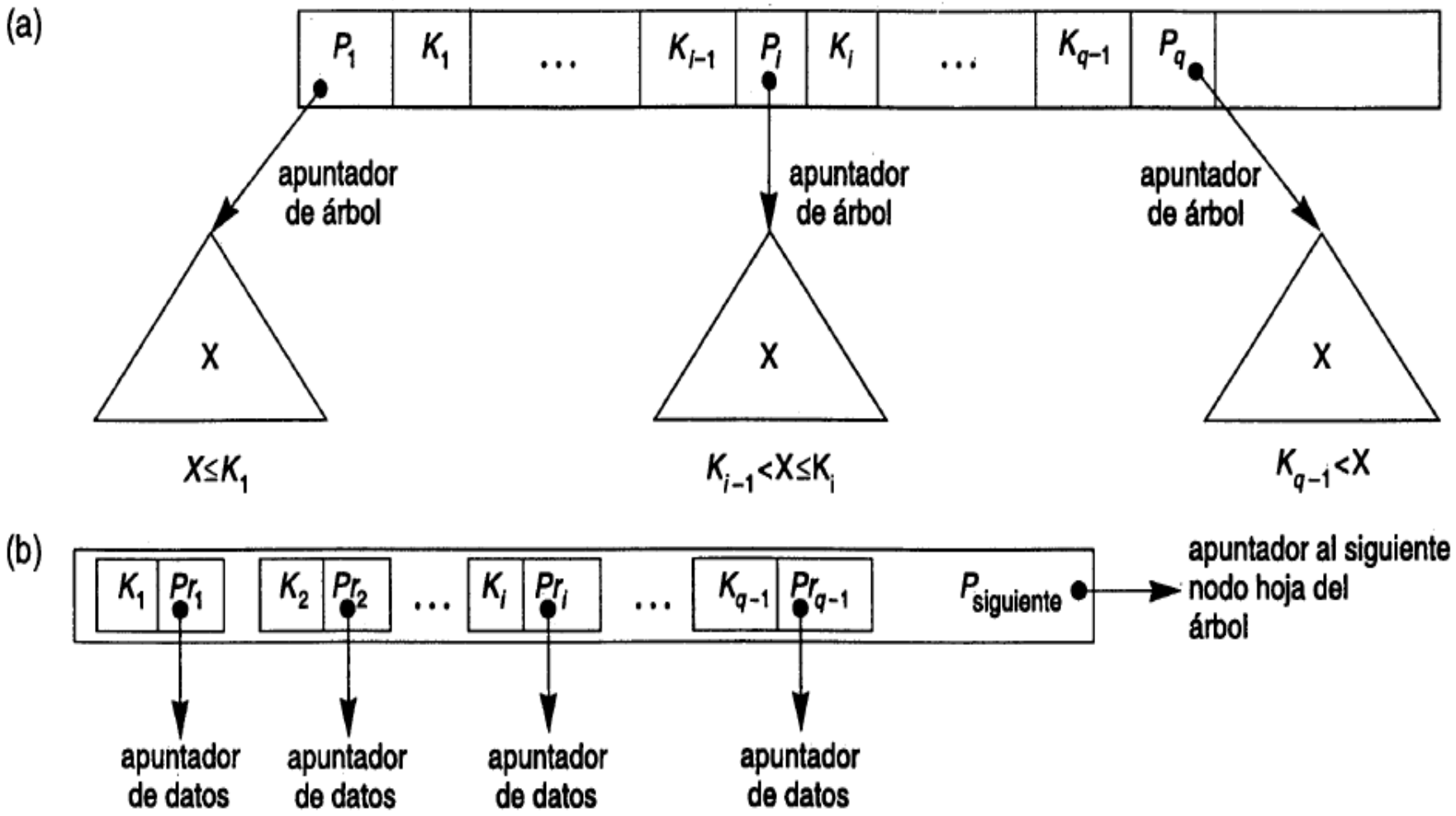


Figura 5.11 Nodos de un árbol B+. (a) Nodo interno de un árbol B+ con $q-1$ valores de búsqueda. (b) Nodo hoja de un árbol B+.

Árbol B+

1. Todo nodo interno tiene la forma

   $$<P_1, K_1, P_2, K_2, ..., P_{q-1}, K_{q-1}, P_q>$$

   donde $q \leq p$ y cada P_i es un **apuntador de árbol**.
2. Dentro de cada nodo interno, $K_1 < K_2 < \cdots < K_{q-1}$.
3. Para todos los valores X del campo de búsqueda en el subárbol al que apunta P_i, tenemos $K_{i-1} < X \leq K_i$, para $1 < i < q$, $X \leq K_i$, para $i = 1$, y $K_{i-1} < X$, para $i = q$ (véase la Fig. 5.11(a)).†
4. Cada nodo interno tiene cuando más p apuntadores de árbol.
5. Cada nodo interno, con excepción de la raíz, tiene por lo menos $\lceil (p/2) \rceil$ apuntadores de árbol. El nodo raíz tiene por lo menos dos apuntadores de árbol si es un nodo interno.
6. Un nodo interno con q apuntadores, donde $q \leq p$, tiene $q - 1$ valores del campo de búsqueda.

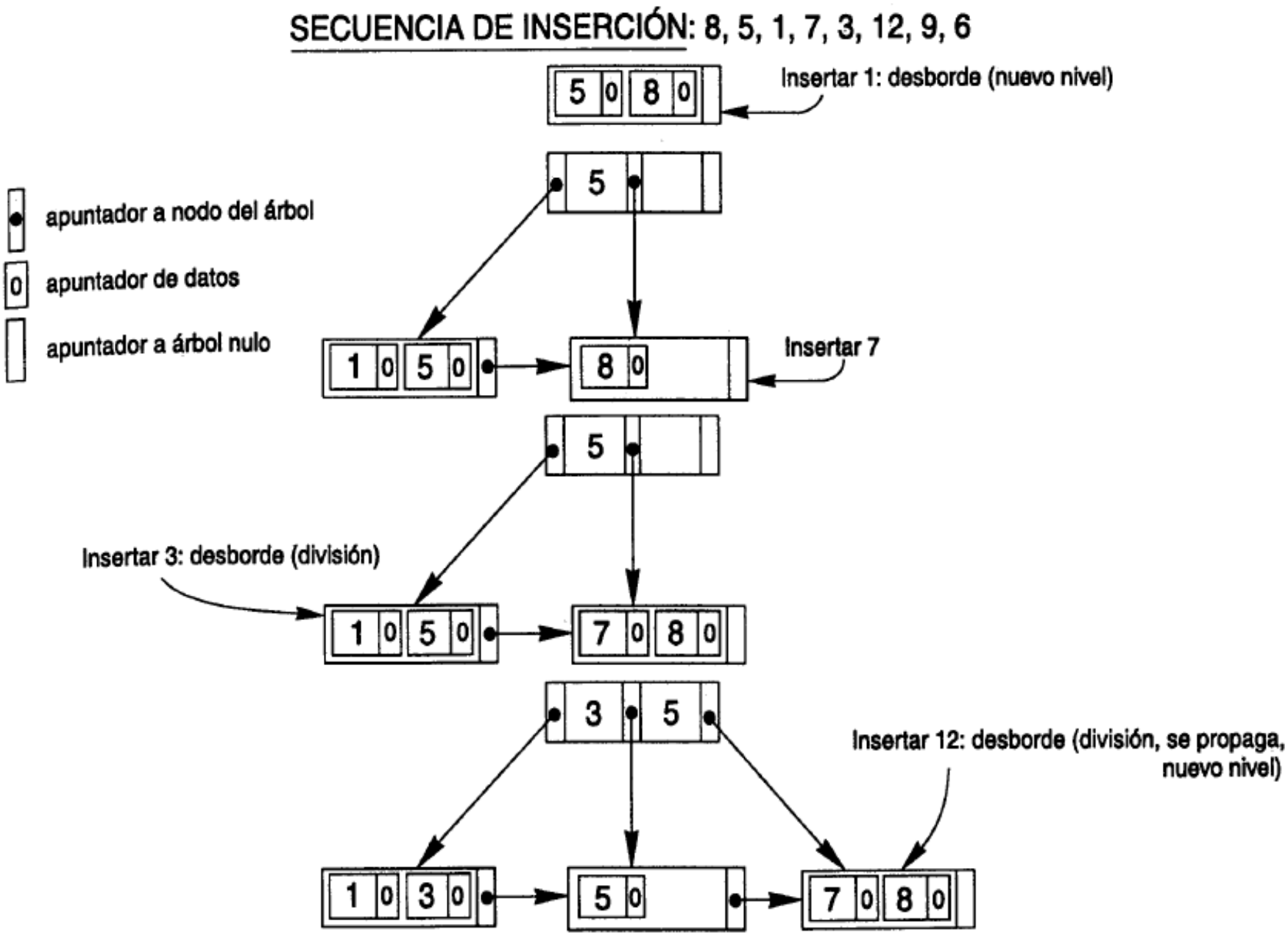
La estructura de los *nodos hoja* de un árbol B^+ de orden p (Fig. 5.11(b)) es como sigue:

1. Todo nodo hoja tiene la forma

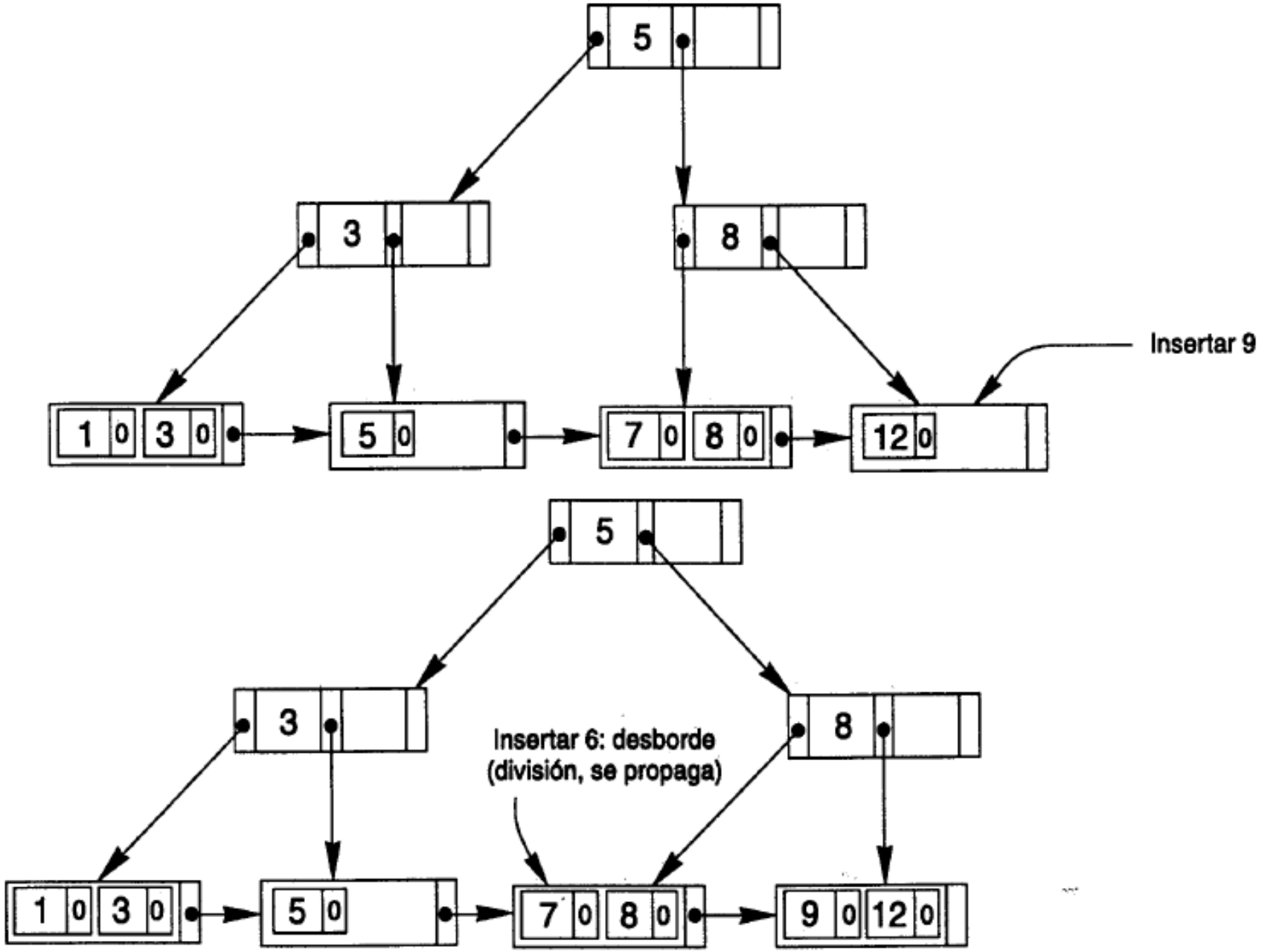
   $$< <K_1, Pr_1>, <K_2, Pr_2>, ..., <K_{q-1}, Pr_{q-1}>, P_{siguiente} >$$

   donde $q \leq p$, cada Pr_i es un apuntador de datos y $P_{siguiente}$ apunta al siguiente *nodo hoja* del árbol B^+.
2. Dentro de cada nodo hoja, $K_1 < K_2 < \cdots < K_{q-1}$, donde $q \leq p$.

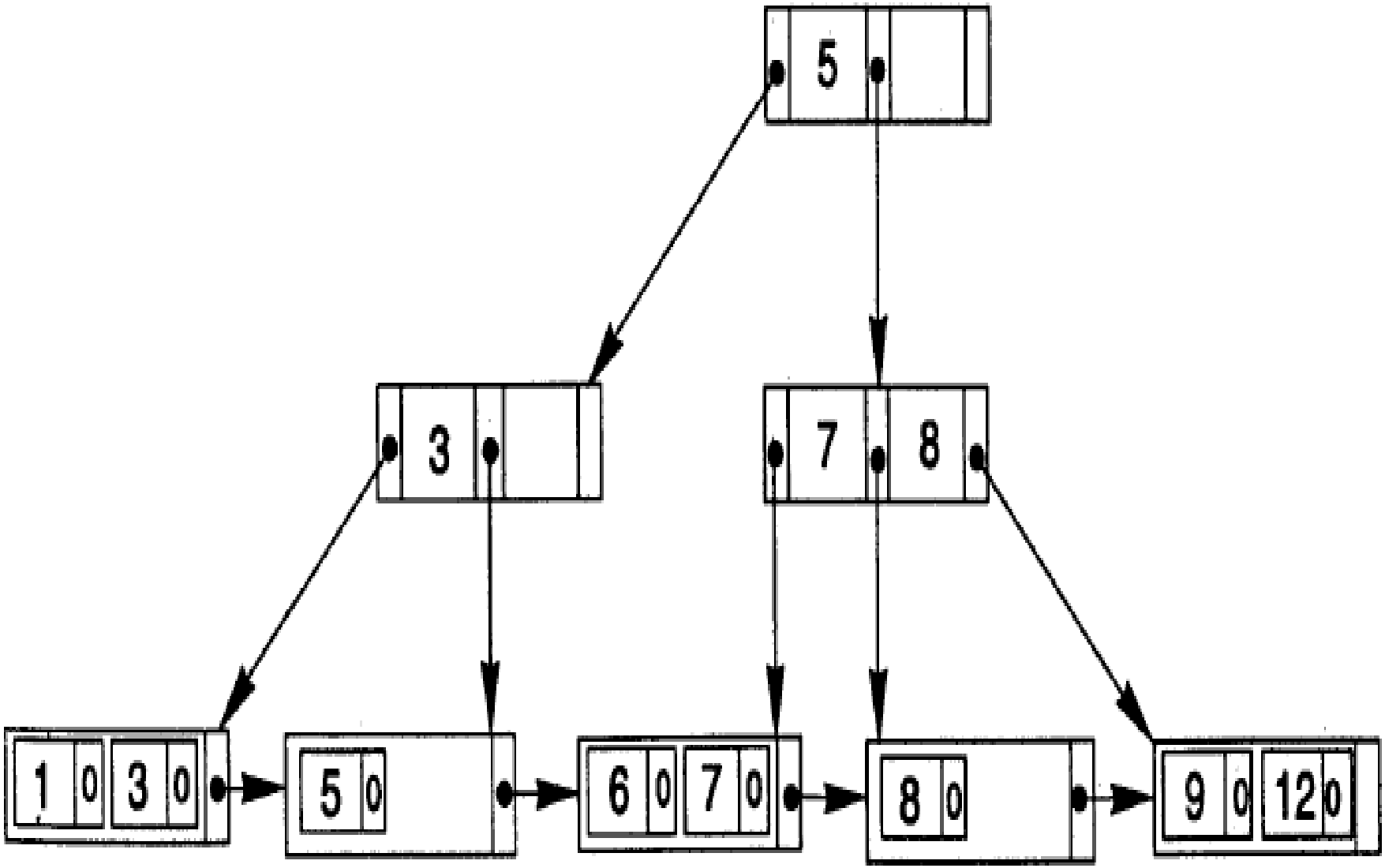
Inserción Árbol B+



Inserción Árbol B+

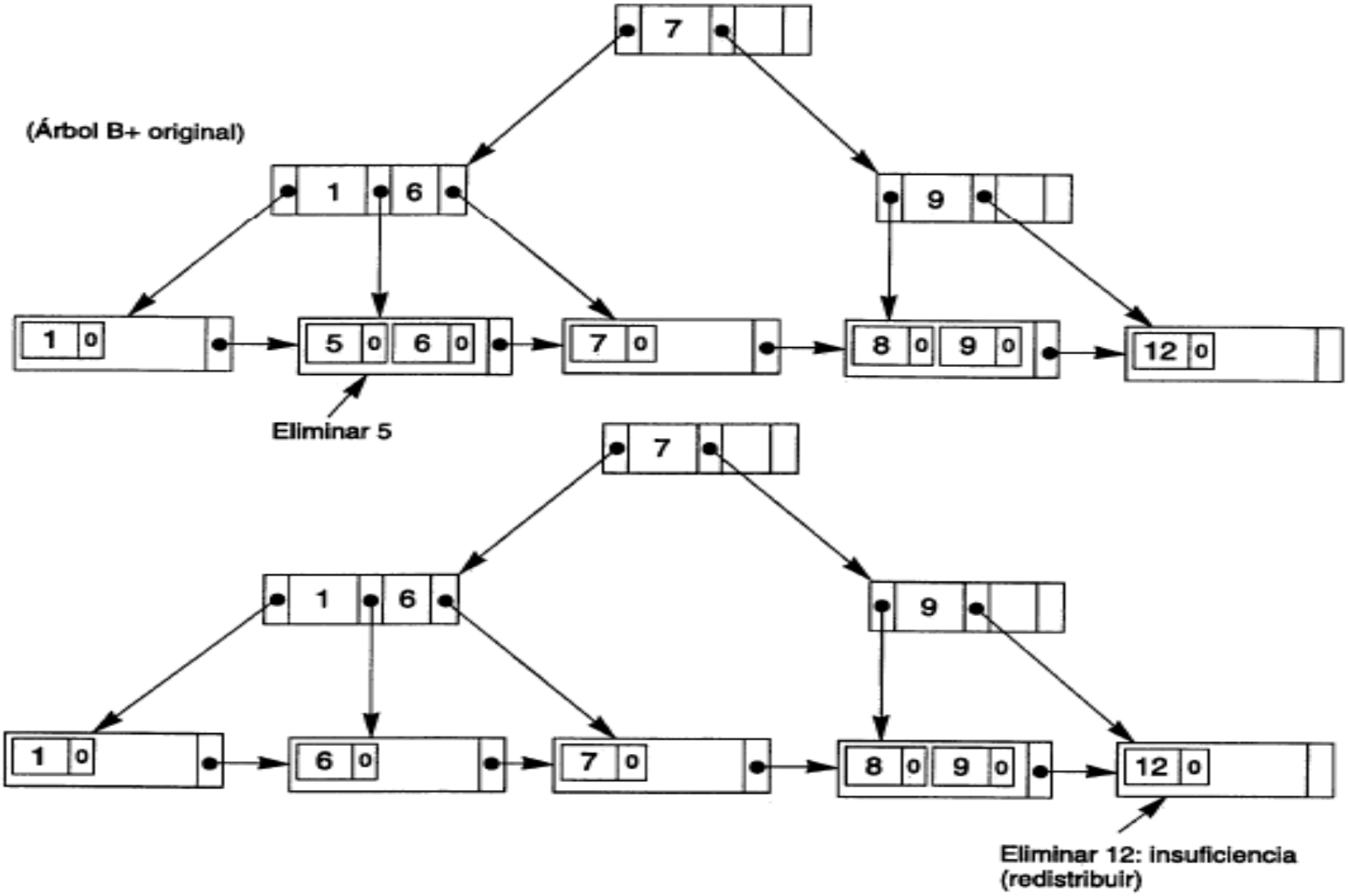


Inserción Árbol B+

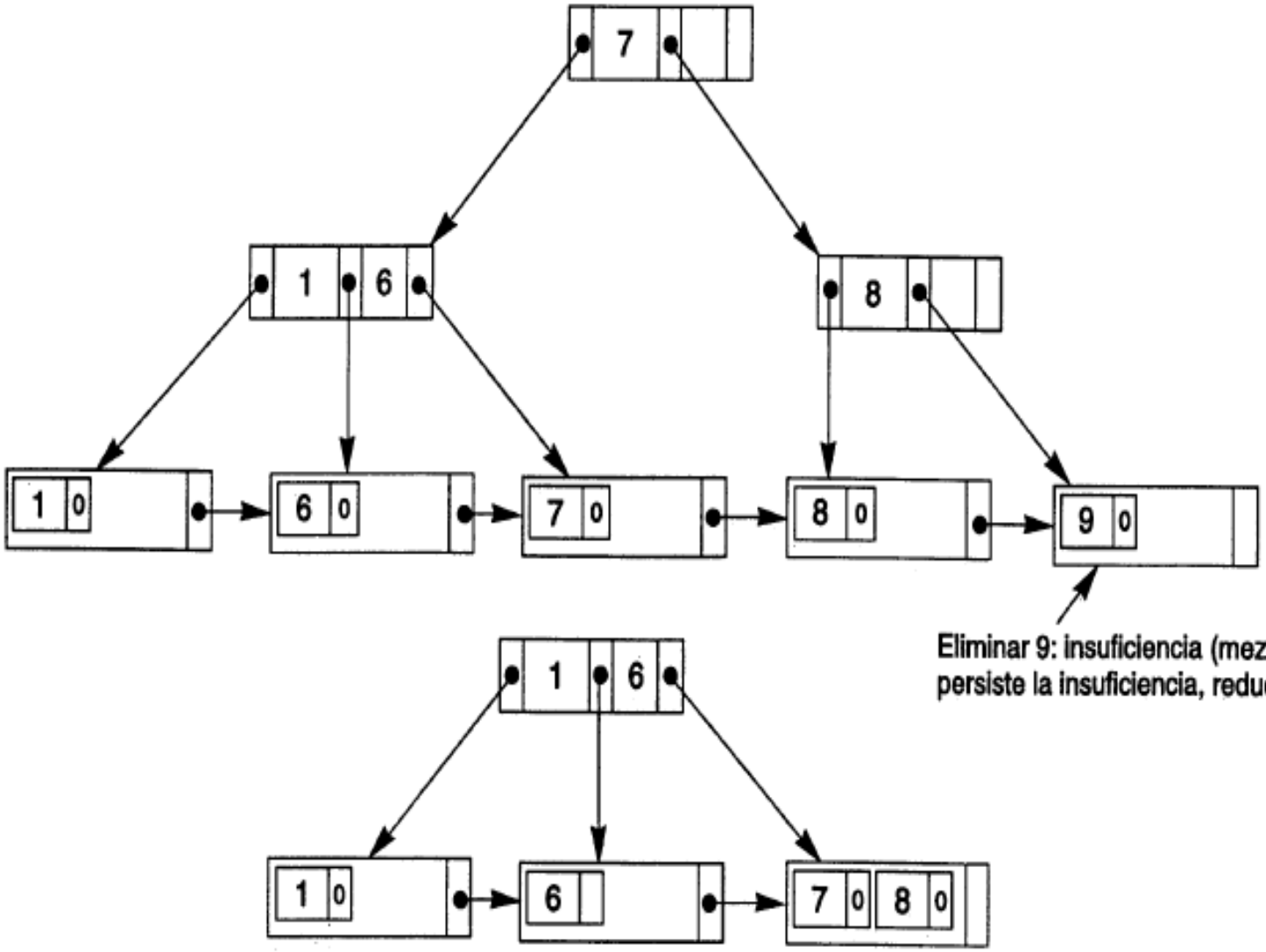


Eliminar en Árbol B+

SECUENCIA DE ELIMINACIÓN: 5, 12, 9



Eliminar en Árbol B+



Ejercicio 1

Tenemos

Archivo ordenado
r = 30000 registros
Tamaño de Bloque B = 1024 bytes
Registros de longitud fija de R = 100 Bytes
Factor de bloqueo fbl = B/R = 1024/100 = 10 registros por bloque

Calcular

El número de bloques requerido para el archivo es b = ? bloques.
Cantidad de accesos a bloques en una búsqueda binaria en el archivo de datos.

Ejercicio 1 Índice primario

Ahora tenemos todo lo anterior más

Un campo clave de ordenación del archivo que tiene V = 9 bytes de largo.
Un apuntador a bloque que tiene P = 6 bytes de largo y que hemos construido un índice primario para el archivo.

Calcular
El tamaño de cada entrada del archivo índice es R = ? Bytes.
El factor de bloqueo del archivo índice fbl = ? entradas por bloque.
El número total de entradas del archivo índice r = ?
El número de bloques requerido para el archivo índice b = ? bloques.
Cantidad de accesos en una búsqueda binaria en el archivo índice ? accesos a bloques.
Cantidad de accesos para encontrar un registro empleando el archivo índice ? accesos a bloques

¿Cuál es la mejor forma de accesos?

Solución Ejercicio 1 Índice primario

Supongamos que tenemos un archivo ordenado de **(r = 30000 registros)** almacenados en un disco con un **(tamaño de Bloque B = 1024 bytes**). Los registros del archivo son de (**longitud fija de R = 100 Bytes)**. **(El factor de bloqueo fbl = B/R = 1024/100 = 10 registros por bloque**). El número de bloques requerido para el archivo es b = r/fbl = 30000/10 = 3000 bloques. Una búsqueda binaria en el archivo de datos requeriría aproximadamente (log2b) = log2 3000 = 12 accesos a bloques.

Supongamos que el campo clave de ordenación del archivo tiene V = 9 bytes de largo, que un apuntador a bloque tiene P = 6 bytes de largo y que hemos construido un índice primario para el archivo.

El tamaño de cada entrada del índice es R = 9+6=15 bytes, de modo que el factor de bloqueo del índice es fbl = B/R = 1024/15 = 68 entradas por bloque. El número total de entradas del índice, r, es igual al número de bloques del archivo de datos, que es 3000. Así el número de bloques requerido para el índice es b = r/fbl = 3000/68 = 45 bloques. Efectuar una búsqueda binaria en archivo de índice requeriría log2 b = log2 45 = 6 accesos a bloques. Para encontrar un registro empleando el índice necesitaríamos un acceso adicional a un bloque del archivo de datos, para un total de 6 + 1 = 7 accesos a bloques, mucho mejor que la búsqueda binaria en el archivo de datos que requiere 12 accesos a bloques.

Ejercicio 2 Índice secundario denso

Tenemos

Archivo ordenado
r = 30000 registros
Tamaño de Bloque B = 1024 bytes
Registros de longitud fija de R = 100 Bytes
El número de bloques requerido para el archivo es b = 3000 bloques.
Factor de bloqueo fbl = B/R = 1024/100 = 10 registros por bloque.

Calcular

Cantidad promedio de accesos a bloques en una búsqueda lineal en el archivo de datos.

Ejercicio 2 Índice secundario denso

Ahora tenemos todo lo anterior más

Un índice secundario sobre un campo clave no de ordenación del archivo que tiene V = 9 bytes de largo.
Un apuntador a bloque que tiene P = 6 bytes de largo y que hemos construido un índice primario para el archivo.

Calcular
El tamaño de cada entrada del archivo índice es R = ? Bytes.
El factor de bloqueo del archivo índice fbl = ? entradas por bloque.

En un índice secundario denso como éste, el ´numero total de entradas del archivo índice , r , es igual al número de registros del archivo de datos que es 30000.

¿Cuál es el número total de bloques para el archivo índice?

Compárelo con el total de bloques del archivo índice primario no denso del ejercicio 1

¿Cuántos accesos a bloque necesita una búsqueda binaria en este archivo de índice secundario?

¿Cuántos accesos a bloque necesita una búsqueda de un registro empleando el archivo índice secundario denso?

Solución Ejercicio 2 Índice secundario denso

EJEMPLO 2: Consideremos el archivo del ejemplo 1 con $r = 30\ 000$ registros de longitud fija de tamaño $R = 100$ bytes, almacenado en un disco cuyos bloques son de $B = 1024$ bytes. El archivo tiene $b = 3000$ bloques, según se calculó en el ejemplo 1. Para efectuar una búsqueda lineal en este archivo tendríamos que examinar $b/2 = 3000/2 = 1500$ accesos a bloques en promedio. Supongamos que construimos un índice secundario basado en un campo clave (no de ordenamiento) del archivo que tiene $V = 9$ bytes de longitud. Al igual que en el ejemplo 1, un apuntador a bloque tiene 6 bytes de largo, así que cada entrada de índice tiene $R_i = (9 + 6) = 15$ bytes, y el factor de bloques del índice tiene $fbl_i = \lfloor (B/R_i) \rfloor = \lfloor (1024/15) \rfloor = 68$ entradas por bloque. En un índice secundario denso como éste, el número total de entradas de índice, r_i, es igual al *número de registros* del archivo de datos, que es $30\ 000$. El número de bloques requeridos para el índice será entonces $b_i = \lceil (r_i/fbl_i) \rceil = \lceil (30\ 000/68) \rceil = 442$ bloques. Compárese esto con los 45 bloques que necesita el índice primario no denso del ejemplo 1.

Una búsqueda binaria en este índice secundario requiere $\lceil (\log_2 b_i) \rceil = \lceil (\log_2 442) \rceil = 9$ accesos a bloques. Para encontrar el registro empleando el índice, requeriremos un acceso adicional a un bloque del archivo de datos para un total de $9 + 1 = 10$ accesos a bloques; mucho mejor que los 1500 accesos a bloques que requiere en promedio una búsqueda lineal en este archivo, pero no tan bueno como los 7 accesos a bloques que se necesitan con el índice primario. ■

Ejercicio 3 Índice de múltiples niveles

EJEMPLO 3: Suponga que el índice secundario denso del ejemplo 2 se convierte en un índice de múltiples niveles. Calculamos un factor de bloques $fbl_i = 68$ entradas de índice por bloque, cifra que también es el abanico fo para el índice de múltiples niveles. También se calculó el número de bloques del primer nivel $b_1 = 442$ bloques. El número de bloques del segundo nivel será $b_2 = \lceil (b_1/fo) \rceil = \lceil (442/68) \rceil = 7$ bloques, y el número de bloques del tercer nivel será $b_3 = \lceil (b_2/fo) \rceil = \lceil (7/68) \rceil = 1$ bloque. Por tanto, el tercer nivel es el nivel superior del índice, y $t = 3$. Para tener acceso a un registro buscando en el índice de múltiples niveles, deberemos tener acceso a un bloque en cada nivel y a uno más en el archivo de datos, así que necesitamos $t + 1 = 3 + 1 = 4$ accesos a bloques. Compárese esto con el ejemplo 2, donde se requirieron 10 accesos a bloques empleando un índice de un solo nivel y una búsqueda binaria. ■

Terminamos por hoy