Tema 7: Organización de Ficheros: Organizaciones Auxiliares

- **Introducción**
  - **Concepto de Índice y Apuntamiento**
  - **Diseño básico de índices**
  - **Operaciones y Coste de Procesos sobre Org. Indizadas**
  - **Ventajas e inconvenientes de su aplicación**
- **Taxonomías de Índices: simples e Índices Multinivel**
- **Indización por Árboles B**
- **Estructuras Auxiliares Especiales**
- **Procesos indizados**

Tema 7.1: Concepto de Índice

Las organizaciones base suelen establecerse entorno a un proceso privilegiado (o unos pocos procesos).

Serial → privilegia inserciones
Secuencial → privilegia algún acceso ordenado (y alguna localización)
Direccionada → privilegia localizaciones a través de una clave

- El resto de procesos selectivos (clave alternativa) son pesados (*full scan*)
- Si una cl. alternativa es muy frecuente, se puede almacenar en un archivo aparte la ubicación física de cada valor de esa clave.

  Ejemplo: en un libro, la clave *título_capítulo* se almacena asociada a la ubicación del registro (núm. página) en un archivo aparte (**índice**).

Tipo de archivo auxiliar: *índice* (directorio)

Clave privilegiada: *clave de indización*

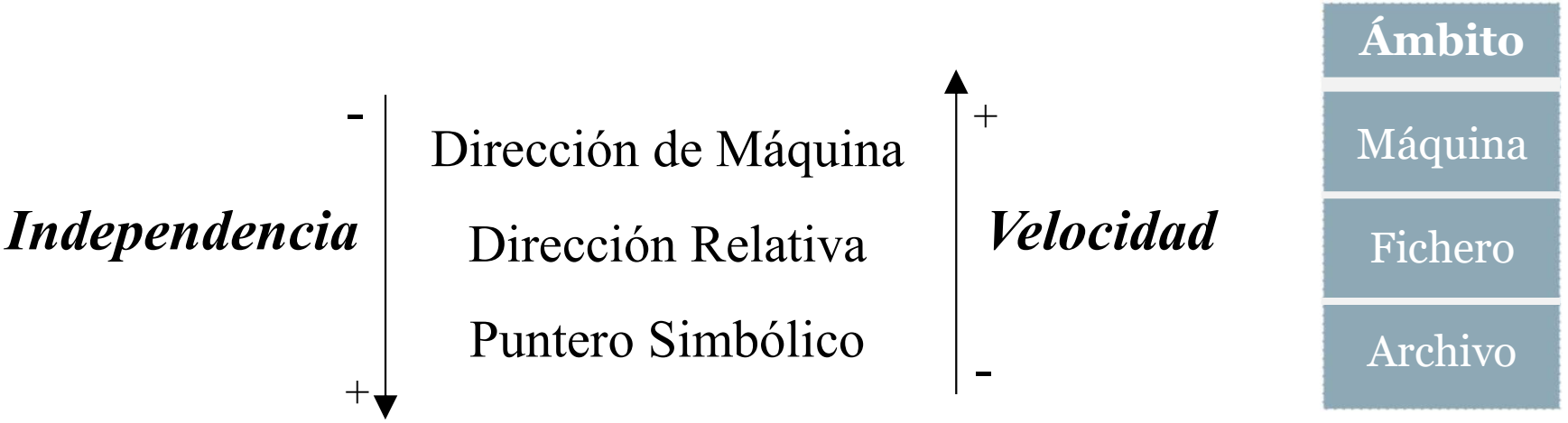
Por ser auxiliar, se pueden establecer cuantos índices se estime oportuno

Tema 7.1: Conceptos previos

Tipos de Puntero (según su dominio):

- *Dirección de Máquina*: la dirección física del registro
- *Dirección Relativa*: del registro en el espacio de dircmto. del fichero
- *Puntero Simbólico* (identificador): identificación lógica del registro

(*Puntero Simbólico* no identificador: caracterización lógica de un cjto. de registros)



Tema 7.1: Concepto de Índice

- *Entrada*: registro formado por punteros
- *Directorio*: archivo formado por entradas (y por ende, por punteros)

  Algunos usos: - virtualizar direccionamientos (ver lect. comp. 6)
  - asociar registros (p.e., personas con sus coches)
  - traducir punteros

***ÍNDICE*: directorio cuya entrada se refiere a un solo registro**

- Es como un listado para traducir punteros (lógicos a relativos)

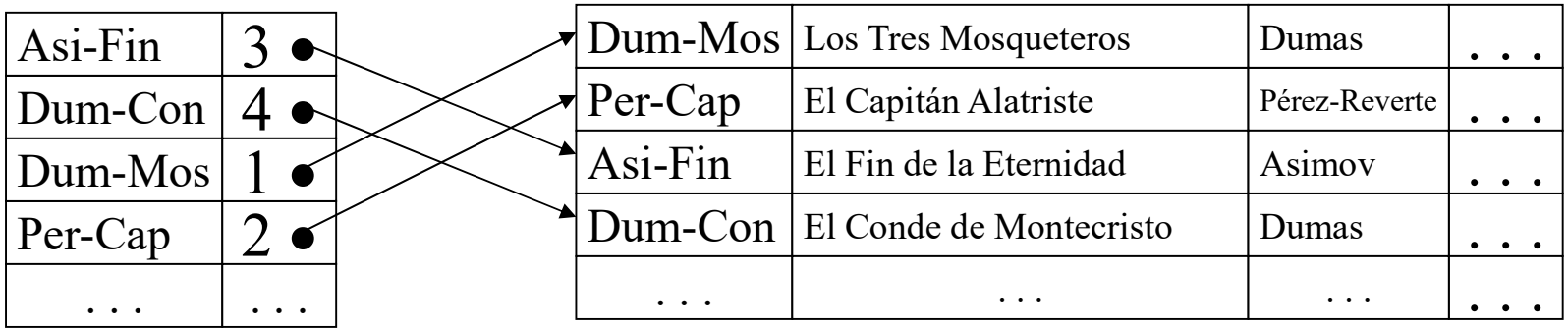
- *"Almacenamiento auxiliar utilizado para localizar los registros"*
  - Los índices se almacenan en un fichero (o varios) de índices.
  - Los registros, en el fichero de datos manteniendo su *organización base*

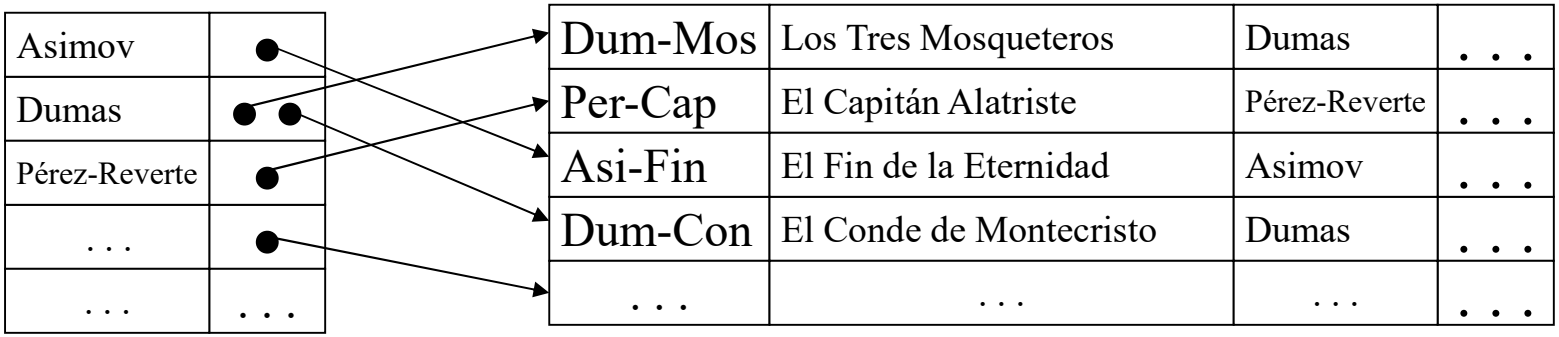
- ***La clave siempre es un puntero lógico*** *(no necesariamente unívoco)*
- ***El otro puntero suele ser relativo (dir. del cubo / posición en el cubo)***
  - Si el uso del índice es el filtrado (del cjto. dir. relevantes) → sólo parte alta ptro.
  - Si la org. base es virtual, se utilizará la dirección virtual

Tema 7.1.2: Tipos básicos de Índice

- **PRIMARIO**: la clave de indización es **identificativa**

  1 registro ↔ 1 valor de clave ↔ 1 entrada (1 puntero)





- **SECUNDARIO**: la clave de indización es **no identificativa**

  n registros ↔ 1 valor de clave ↔ 1 entrada (n punteros)

Tema 7.1.2: Diseño de la Entrada

- **Diseño Físico-Lógico de la entrada de índice Primario**:

  ENTRADA ≡ clave · puntero_externo

- **Diseño Físico-Lógico de la entrada de índice Secundario**:

  - Habrá varios registros con el mismo valor de clave...
  - Se almacena **sólo una vez el valor** de cada clave, con todos sus punteros.

  ENTRADA ≡ clave · long_lista · (puntero_externo)$^{long\_lista}$

  - Corolarios:
    - El número de entradas es igual a la card. del dominio: $e = \#valores(CI)$
    - Al buscar, sólo hay que recorrer el índice hasta encontrar una entrada
    - Para insertar en listas de punteros hay que buscar la entrada correspondiente y en caso de que no exista (tras recorrer todo el fichero) insertar al final.
      → Con listas, conviene tener el **índice ordenado,** y siempre **no consecutivo**
    - La longitud media de la lista es la coincidencia de clave $k = r / \#valores(CI)$

Tema 7.1.2: Operaciones sobre Ficheros Indizados

- Operaciones de administración / mantenimiento:
  - ***Creación***: hay que crear el índice (al crear el fichero o posteriormente)
  - ***Borrado***: si se borra el fichero de datos, hay que borrar el índice
    Además, se puede destruir el índice sin borrar el fichero de datos.
  - *Reorganización*: si degenera, deberá reorganizarse periódicamente.
- Operaciones selectivas (localización de registros):
  - ***Existencia***: acceso al fichero de índices
  - ***Localización***: acceso al fichero de índices + acceso al fichero de datos
  - ***Consulta a la totalidad***: generalmente, no se utiliza el índice
    (excepción: proceso ordenado / índice ordenado / fichero no ordenado)
- Operaciones de Actualización:
  - **(1/2) Selección** (por CI): se puede hacer a través del índice
  - **(2/2) Actualización**: escritura fichero datos + escritura fichero índice

Tema 7.1.2: Coste de Procesos Ficheros Indizados

- **Localización a través del Índice:** acceso al índice (según su naturaleza)
- **Localización por varios índices:** suma del acceso a cada índice
- **Recuperación:** $C(O_i,P_j) = acceso\_índice + acceso\_datos$

  $acceso\_datos = \#cubos \cdot E_c$
- **Actualización:**
  - Inserciones: suelen requerir inserción de entradas — $\Delta coste = acc\_índice + 1$
  - Borrados: pueden localizarse con el índice — $\Delta coste = acc\_índice + 1$
    - índice primario: suelen requerir borrado de entradas
    - índice secundario: pueden requerir modificación de entradas
  - Modificaciones: pueden localizarse con el índice
    - CI: suele implicar borrado + reinserción de entrada — $\Delta coste = 2 \cdot acc\_índ + 2$
    - CD/CO: cambia ubicación reg. → cambia puntero — $\Delta coste = acc\_índice + 1$

Tema 7.1.3: Efectos de Indizar

Ventajas

1. *Acceso por **Claves Alternativas***
   Se gana eficiencia en la localización por claves (hasta ahora) no privilegiadas

2. *Aumento de la **Tasa de Acierto** (en M_{int})*
   El índice tiene menos cubos y de acceso más recurrente. Con buena gestión de M_{int}, el acierto es muy elevado en los accesos al índice (que son la mayor parte del acceso indizado) haciendo que se dispare la tasa de acierto global.

3. *Reorganización Menos Costosa*
   Ya que los índices tienen menos bloques que el f. de datos, este coste es menor

Inconvenientes

1. ***Procesos de Actualización Más Costosos***
2. *Necesidad de Almacenamiento Auxiliar*
3. *Necesidad de Operaciones de Mantenimiento*

Tema 7.2: Ficheros Indizados.

Taxonomías de Índices

- Según el carácter (identificativo) de la clave de indización:
  - ***índices primarios vs. índices secundarios***
- Según la correspondencia (biyectiva o no) entre entradas y registros:
  - ***Denso*** (1:1): existe una entrada del índice para cada registro
  - ***No Denso*** (1:n): una entrada para cada cubo de datos
- Según el recubrimiento del índice*:*
  - ***Exhaustivo****:* todos los registros que deben tener entrada la tienen
  - ***Parcial****:* no se indizan todos los registros

  (se dejan aparte los que se acceden rara vez, los últimos en ser introducidos, etc.)

  → Si el índice parcial falla (∅), no aporta ningún valor informativo (no filtra).
- *Según la estructura: índices simples vs. índices multinivel (arbóreos)*

Tema 7.2: Índice Simple Denso

- **Naturaleza**: serial, secuencial, o direccionado

- **Coste**: dependiendo de su naturaleza (igual que un fichero de datos análogo)

- **Restricciones**: se debe **aplicar sobre claves no privilegiadas**

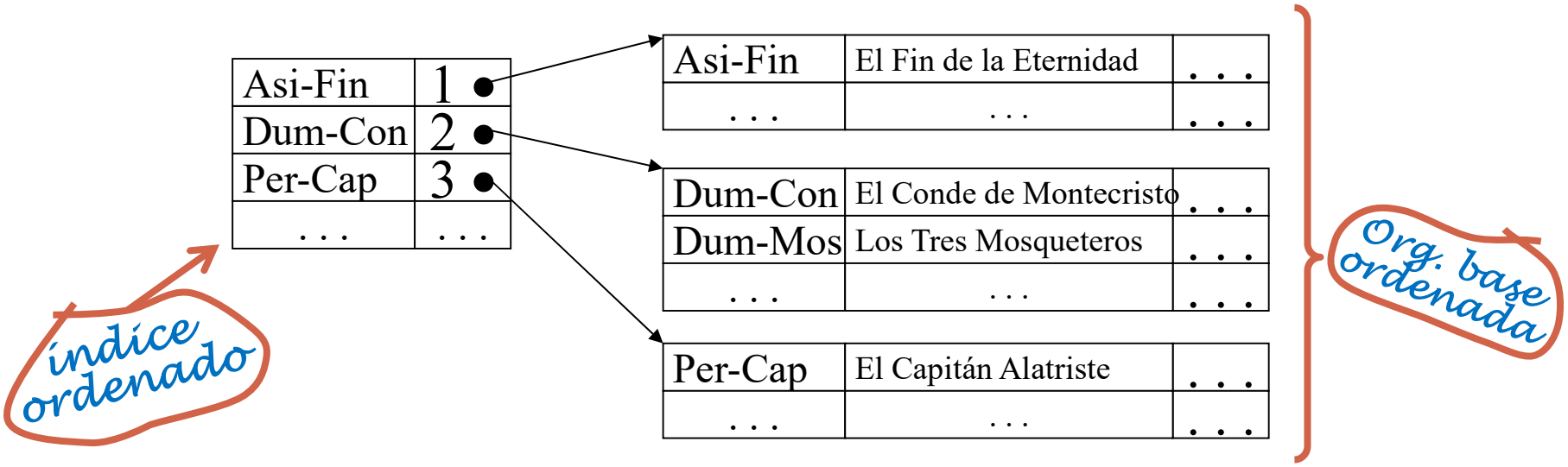
Ejemplos:
- si el índice se usa para un proceso ordenado por CI (y además CI≠CO)
- si la CO se usa para un proceso ordenado y la CI para procesos selectivos
- si el índice se usa para un proceso especial (ver acceso invertido, tema 7.5)

Mantenimiento:

- si es ordenado o disperso, puede desbordar → requiere reorganización

- debe evitarse la degeneración de la estructura
  → índice ordenado: preferible inserción ordenada + reorganización local
  → índice disperso: pierde eficiencia si cambia (si es volátil)
  es más útil como índice temporal (procesos puntuales)

Tema 7.2: Índice Simple No Denso

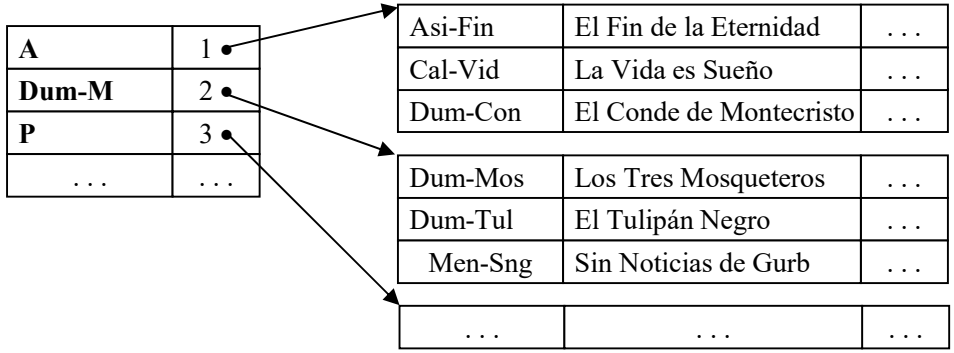


- **Concepto**: una entrada por cubo de datos (en lugar de una entrada por registro)
- **Restricción**: índice y organización base deben ser necesariamente secuenciales y con ***clave_indización = clave_ordenación (CO=CI)***
- **Usos**: aporta varias posibilidades de acceso:
  - procesos ordenados (a la totalidad): acceso serial de la org. base (ordenada)
  - procesos selectivos (solución única): a través del índice
  - mixtos (selección de un rango): acc. indizado (1er elemento) + serial

Tema 7.2: Índice Simple No Denso

Ventajas:

- al ser de tamaño muy reducido, tiene menor coste (y mayor tasa de acierto)
- se ahorran muchas actualizaciones de índice (insertar/borrar/modificar registros a menudo no afectan al índice, salvo que sean el primer registro del cubo).
- en lugar de utilizar toda la *clave* en la entrada, se pueden usar prefijos (%tamaño)

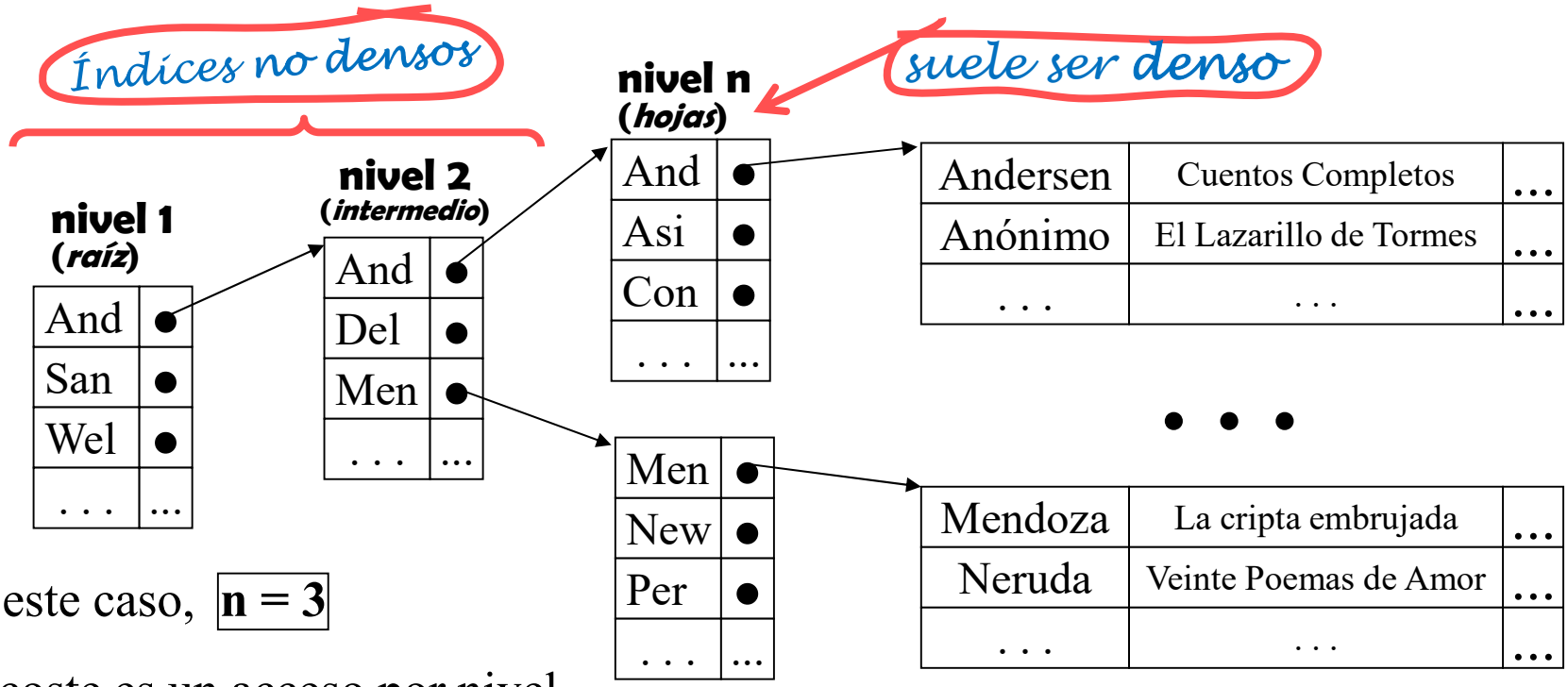


Inconvenientes:

- sólo puede existir un índice no denso para cada archivo
- la inserción del registro debe ser ordenada, pero se localiza con el índice.
- La inserción de la entrada es ordenada, y conlleva pesadas reorganizaciones
  - deben aplicarse mecanismos de ELD, rotación, partición celular, etc.

Tema 7.2: Índice Multinivel

Concepto: es un índice con *n* niveles (el nivel n es índice del nivel n+1)



- En este caso, **n = 3**
- El coste es un acceso por nivel
  Interesa definir **nodos pequeños** (1 bloque) incluso a costa de tener más niveles
- Es ventajoso **bloquear la raíz** (nivel 1) en memoria intermedia (ahorra un acceso)
- El nivel n suele ser denso, pero como es secuencial puede ser un índice no denso

Tema 7.2: Índice Multinivel

- El índice multinivel perfectamente construido es eficiente, pero degenera
- En ficheros constantes es buena solución
- En ff. volátiles se requiere reorganización local → evolución a otras estructuras

Árboles Binarios: cada nodo es una entrada del índice con dos punteros internos
Solución sencilla, pero presenta problemas de vecindad y desequilibrio.

- **Árbol AVL**: resuelve desequilibrio mediante procesos de reorganización local
- **Árboles Binarios Paginados**: resuelven el problema de vecindad
- **Árboles AVL-Paginados**: buen rendimiento, pero necesitan muchos punteros internos, presentan bajísima densidad, y reorganizaciones frecuentes.

Solución:

- *Incluir varias entradas por nodo*
  (y, por tanto, varios descendientes)
- *Construir el árbol en orden ascendente*
  (el separador pertenece al nodo que desborda)

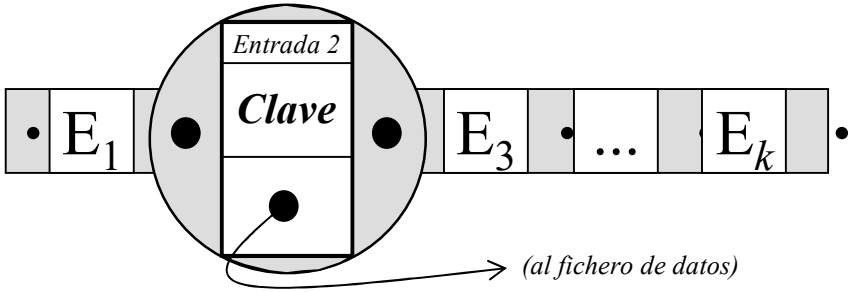
} **árboles B**

Tema 7.3.1: Indización en Árbol B

- *Propuestos por Bayer y McCreight*

Idea: ya que inicialmente no se conoce el elemento que es mejor separador, se comienza por las hojas. A medida que crezca, se construye hacia arriba.

- **Nodo**: cada nodo va a llenar la página; contiene entradas de índice (pares *clave indización-puntero a los datos*) y punteros (para apuntar nodos hijo)



- **Orden del árbol**: indica la capacidad de los nodos (y por ende, del árbol)
  - según las entradas: el nº mínimo de claves de un nodo (*Bayer*)
  - según los punteros (hijos): nº máximo de hijos de un nodo (*Knuth*)

Tema 7.3.1: Indización en Árbol B

Árboles B : Observaciones

- Si el árbol es de orden ***m***, cualquier nodo tendrá a lo sumo *m* descendientes
- Si un nodo tiene *m* descendientes (no hoja), tendrá m-1 entradas
- Corolario: un nodo de un árbol de orden *m* tiene a lo sumo ***k*=*m*-1** entradas
- En un nodo (T_{nodo} bytes) caben ***m*** punteros internos y ***k*** entradas, luego:

$$m \cdot T_{ptro\_interno} + k \cdot T_{entrada} \leq T_{nodo}$$

(con esta fórmula y $k=m-1$ se podrá hallar el orden del árbol)

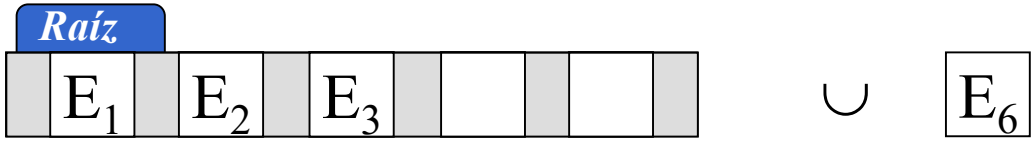
- El nodo raíz tiene al menos un elemento y, por tanto, al menos 2 hijos.

- $\mathbf{T_{nodo}}$ es múltiplo de T_{bloque} y suele ser lo menor posible (**típicamente 1 bq**)
- $\mathbf{T_{entrada}}$ es la suma del tamaño real de la clave (fija/marcada/codificada…) más el/los puntero/s interno/s (pueden ser muchos punteros, si es secundario)

Tema 7.3.1: Indización en Árbol B

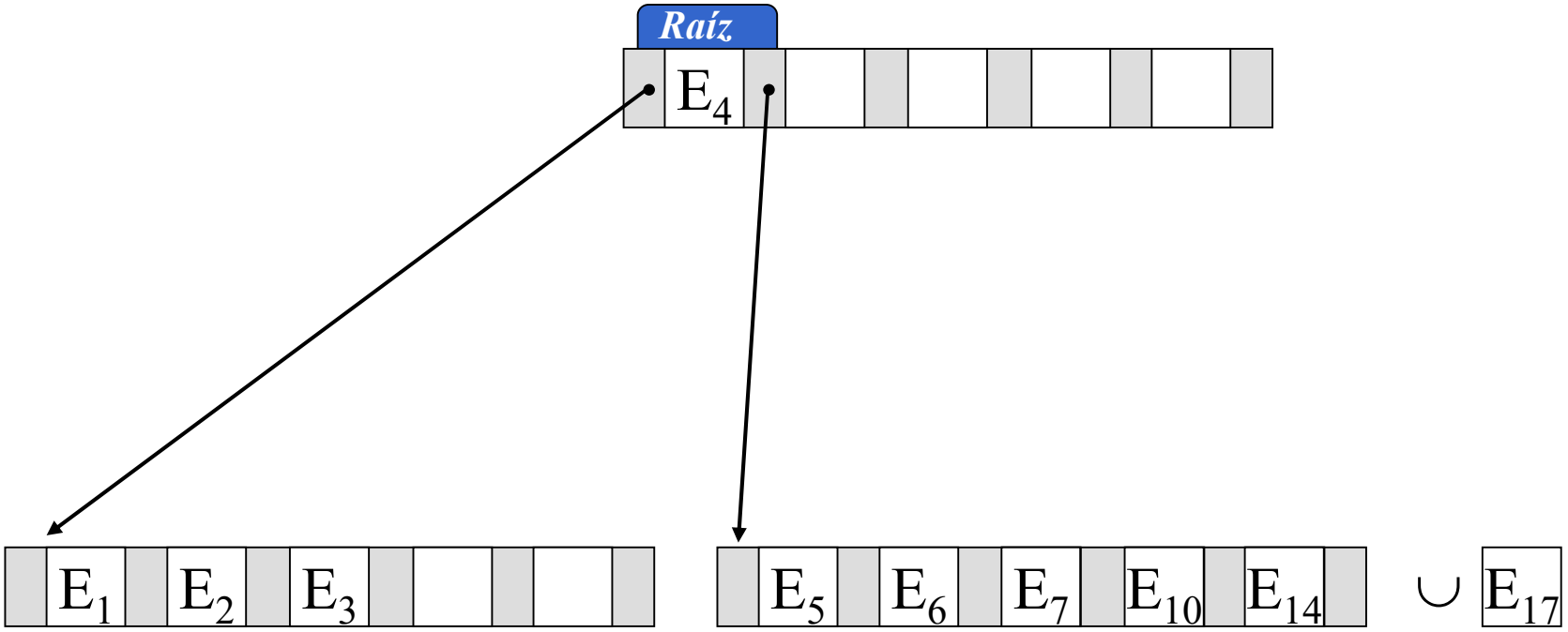
Árboles B : Partición y Promoción

- Las entradas dentro de un nodo van ordenadas
- Cuando un nodo *desborda*, se divide en dos y se promociona el elemento intermedio hacia el nivel superior (ese elemento se lleva dos punteros: uno hacia cada hijo, es decir, hacia cada uno de esos dos nuevos nodos)



Tema 7.3.1: Indización en Árbol B

Árboles B : Más Partición y Promoción (otro ejemplo)



Tema 7.3.1: Indización en Árbol B

Árboles B : Propiedades

- Todos los nodos menos el raíz garantizan una **ocupación mínima**: $k_{mín} = \lfloor \frac{k}{2} \rfloor$

Corolario

- ¿Cuántos descendientes como mínimo tienen los nodos intermedios?

  (suponiendo política de *'dividir cuando desborda'*)

$$m_{mín} = k_{mín} + 1 \rightarrow m_{mín} = \lfloor \frac{m+1}{2} \rfloor$$

- Tamaño del fichero de índices

Se puede obtener una cota superior del fichero de índices

$$N^o_{máx}\ nodos\ fichero = \text{nº entradas fichero} / k_{mín}$$

$$T_{máx}\ fichero = \text{nº}_{máx}\ \text{nodos fichero} \cdot T_{nodo}$$

Tema 7.3.1: Indización en Árbol B

Árboles B : Propiedades (II)

El nº de niveles (n) para un árbol de orden m y e entradas tiene **cota superior**

nivel	*nodos*	*entradas*	*acumulado*
1	1	1	1
2	2	$2 \cdot k_{mín}$	$1+ 2 \cdot k_{mín}$
3	$2 \cdot m_{mín}$	$2 \cdot m_{mín} \cdot k_{mín}$	...
...	...	...	...
→ n	$2 \cdot m_{mín}^{n-2}$	$2 \cdot m_{mín}^{n-2} \cdot k_{mín}$	$(2 \cdot m_{mín}^{n-1})-1$
$n+1$	$2 \cdot m_{mín}^{n-1}$	$2 \cdot m_{mín}^{n-1} \cdot k_{mín}$	$(2 \cdot m_{mín}^{n})-1$

Cota Superior:

$$n \leq 1 + log_{\lfloor \frac{m+1}{2} \rfloor} (\frac{e+1}{2})$$

$> e$

(este nivel es imposible)

Tema 7.3.1: Indización en Árbol B

- Para **recuperar una entrada**: #accesos = #niveles
  - Dado que la raíz estará siempre en memoria, contamos un acceso menos
- Para **recuperar un registro aleatorio** , se recuperan la entrada y tantos cubos de datos como punteros tenga la entrada (es decir, k cubos)

$$C(O_i,P_j) = (n\text{-}1)\cdot T_{nodo} + c \cdot E_c$$

- El coste de cualquier actualización sobre el índice en árbol B es el coste de **localización más un acceso** de escritura: $\Delta C(\text{actualización}) = (n\text{-}1) + 1 = \boldsymbol{n}$
- El coste extra de una partición es de **dos accesos** de escritura (actualizar el nodo antecesor y escribir el nodo nuevo).
- El tiempo de acceso así calculado es una *cota superior al tiempo de acceso.*
- Puede calcularse la cota inferior (en base al número de niveles del árbol perfectamente construido), para conocer su coste óptimo y valorar el beneficio de ejecutar la reorganización del índice.

Tema 7.3.1: Indización en Árbol B

Árboles B : Valoración

Aspectos Positivos:

- Existe una cota superior razonable del número de accesos a soporte
- Generalmente, las operaciones (de inserción o borrado) requieren reestructurar una página. Y si son más, suelen ser pocas páginas.
- En el peor caso, las páginas están ocupadas a la mitad (aproximadamente)

Aspectos a Mejorar

- Si las entradas son grandes, el orden puede ser demasiado pequeño
- La densidad (mínima) de los nodos es muy mejorable
- En las hojas se desperdicia mucho espacio (no necesitan punteros)

Tema 7.3.2: Indización en Árbol B*

Idea: se pretende **aumentar la densidad** de los nodos

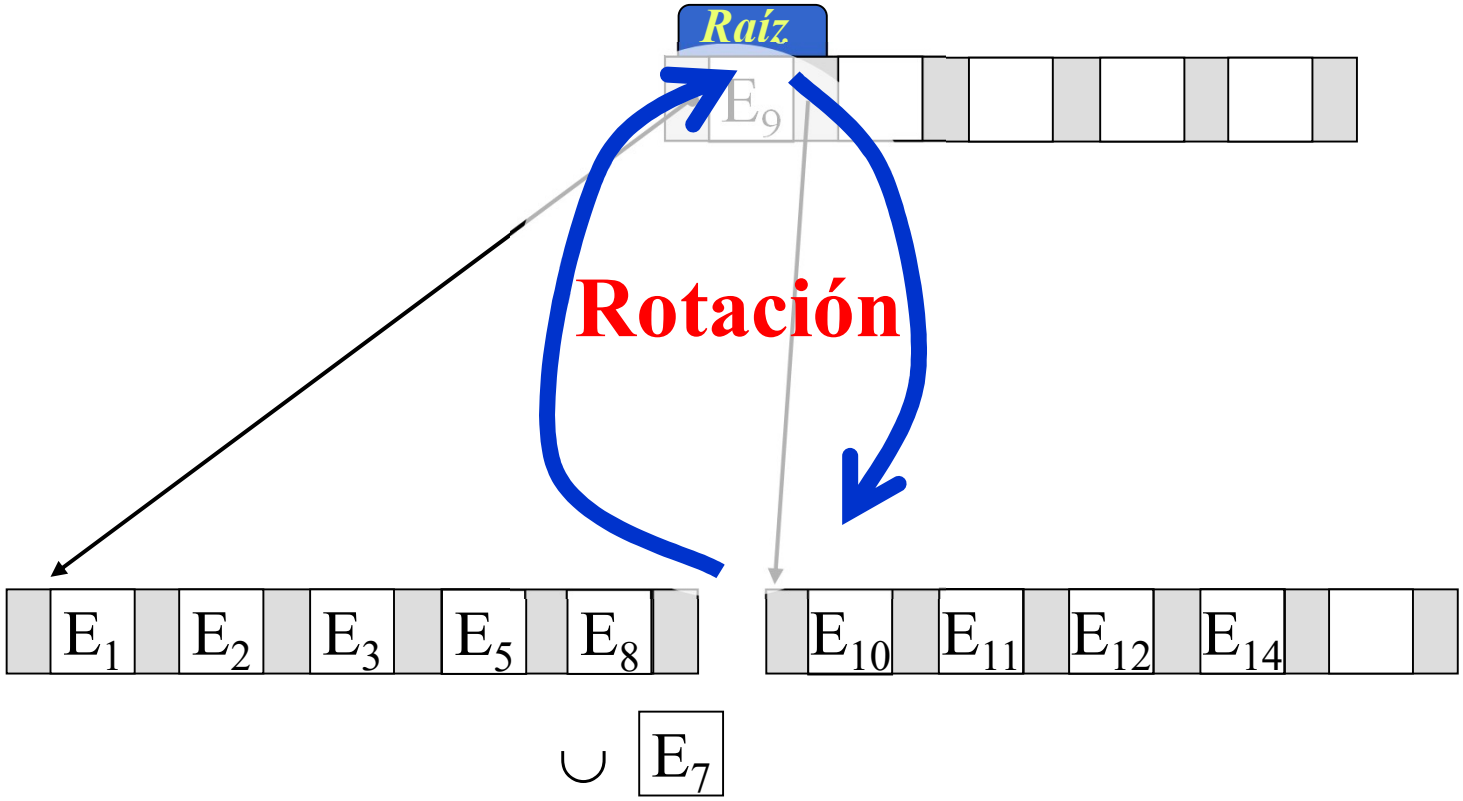
Para ello, en lugar de dividir un nodo en dos, se dividirán dos nodos en tres.

Así, en lugar de conseguir una ocupación mínima del 50% se obtendrá el 66%

- Cuando un nodo desborda, en lugar de dividir, se procurará ceder uno de sus elementos a su vecino (**rotación**).
- Si el nodo vecino también está lleno, se parte (dos nodos llenos en tres nodos)
- Por lo demás, el resto del funcionamiento es como el de los árboles B.
- **Ventajas**:
  - *Aumento de la densidad (al 66%)*
  - *Un desbordamiento no siempre supone partición/promoción*
- **Desventajas**:
  - *Aumenta la probabilidad de desbordamiento (nodos más llenos)*

Tema 7.3.2: Indización en Árbol B*

Rotación, Partición y Promoción (ejemplo)



Tema 7.3.2: Indización en Árbol B*

Propiedades

- Todos los nodos menos el raíz garantizan una ocupación mínima: $k_{mín} = \lfloor \frac{2k}{3} \rfloor$
- Los nodos intermedios cuentan con $\frac{2k}{3}+1$ descendientes $\rightarrow$ $m_{mín} = \lfloor \frac{2m+1}{3} \rfloor$

(suponiendo política de *'dividir cuando desborda'*)

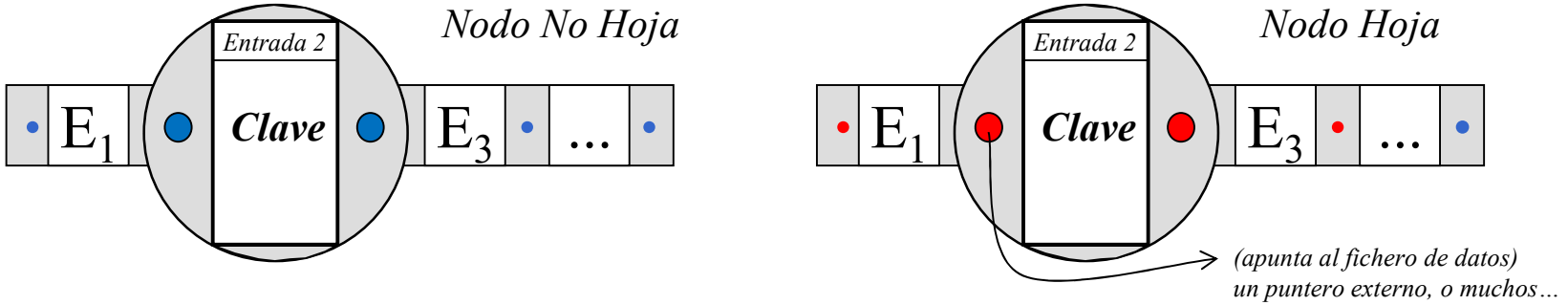
Cálculo de Costes

- Como la localización es idéntica al árbol B, también es igual el cálculo de costes
- El coste extra de una rotación es de tres accesos (lectura del nodo contiguo, más la escritura de ese nodo y del nodo antecesor). ΔC(rotación) = 3 acc
- La partición implica cuatro accesos extra (la lectura del nodo contiguo, más la escritura de los nodos contiguo, nuevo, y el antecesor). ΔC(partición) = 4 acc
- También se puede contemplar la rotación bidireccional. En este caso la densidad es 75% ($k_{mín}=\lfloor 3k/4 \rfloor$) pero también el coste de inserción (rotación 4, y partición 5).

Tema 7.3.3: Indización en Árbol B+

Idea: coste proporcional a la profundidad → crecer en amplitud → aumentar el #hijos por nodo → **aumentar el *orden***

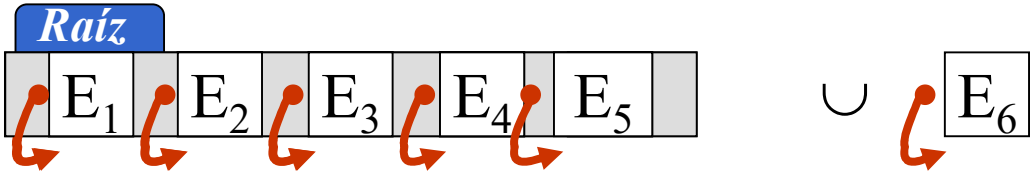
- En los nodos con hijos (nodos no hoja) **se suprimen los punteros externos** (así caben más discriminantes, y por ende se tienen más punteros internos).
- En los nodos hoja no hay punteros a nodo hijo, pero sí habrá punteros externos. Para apuntar a los datos, la entrada debe estar en una hoja → todas las entradas están en nodos hojas, y en los no hoja sólo hay copias discriminantes



- Especialmente eficiente con punteros externos grandes → **índice secundario**

Tema 7.3.3: Indización en Árbol B+
Partición y Promoción

Ejemplo de Partición y Promoción: nodo hoja/raíz *(primera partición)*

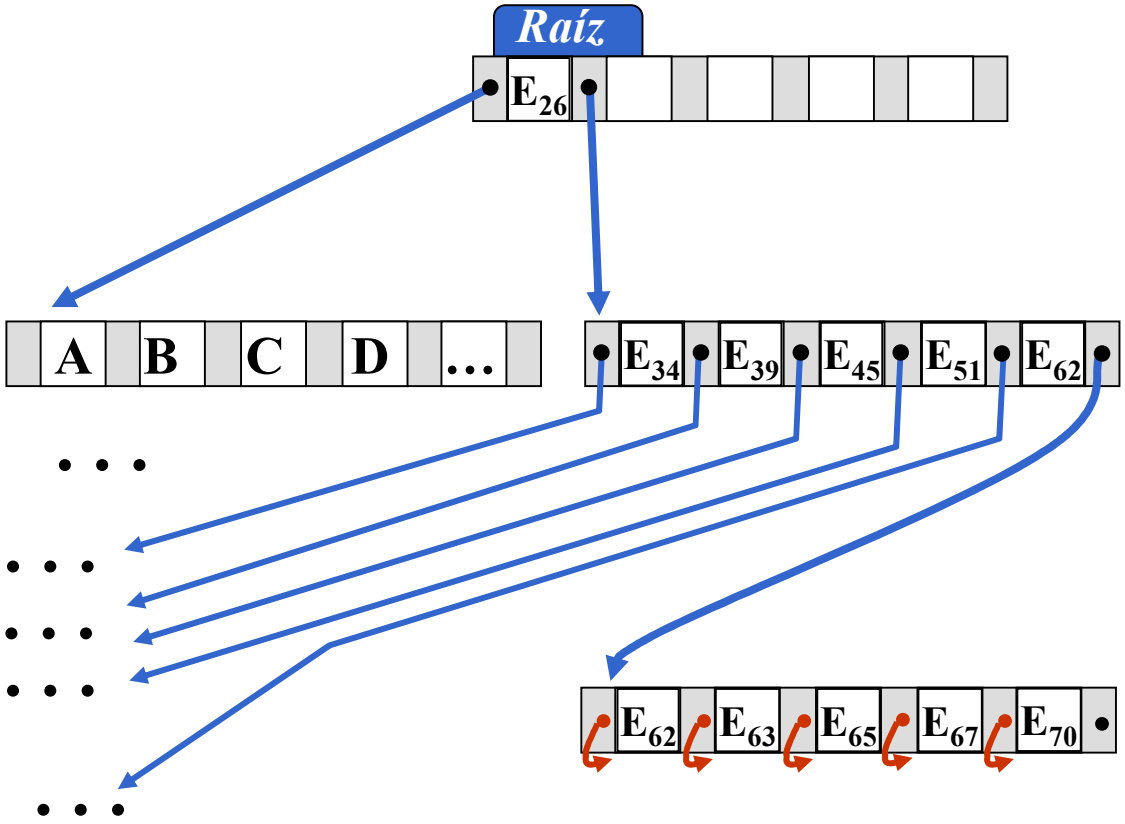


- Observar que en las hojas se usa un puntero interno adicional (puntero encadenamiento) para apuntar al siguiente nodo hoja (hermano). Esto se realiza durante la partición:
  - el ptro_encadenamiento(nodo_nuevo) := ptro_encadenamiento(nodo_viejo)
  - el ptro_encadenamiento(nodo_viejo) := dirección(nodo_nuevo)
- El ***encadenamiento de hojas*** proporciona un mecanismo de acceso alternativo.

Tema 7.3.3: Indización en Árbol B+
Partición y Promoción

- En nodos no hoja, la promoción es igual que en nodos de árbol B

Ejemplo:



Tema 7.3.3: Indización en Árbol B+

Propiedades (kmín y mmín)

- Orden del árbol (***m***): se calcula para nodos no hoja, como en árboles B; teniendo en cuenta que las entradas esos nodos carecen de puntero externo

$$m \cdot T_{puntero\_interno} + (m-1) \cdot T_{clave} \leqslant T_{nodo}$$

- Ocupación máxima (***k***) de los nodos hoja: si los tamaños de los punteros interno y externo son distintos, convendría calcularla por separado

$$k \cdot (T_{clave} + T_{puntero(s)\_externo}) + T_{puntero\_interno} \leqslant T_{nodo}$$

debe contener al menos una entrada comp.

encadmto. bidireccional req. dos punteros internos

- La ocupación mínima de las hojas será:
  (suponiendo política de *'dividir cuando desborda'*)

$$k_{mín} = \lfloor \frac{k+1}{2} \rfloor$$

- La ocupación mínima de los nodos intermedios será $\lfloor k/2 \rfloor$ → $m_{mín} = \lfloor \frac{m+1}{2} \rfloor$
  (la promoción en estos se opera como en los nodos de un árbol B)

Tema 7.3.3: Indización en Árbol B+
Profundidad y Tamaño

Cálculo del número de niveles:

- El nivel de las hojas es el **nivel *n***. ¿Cuántas hojas?

$$n^o\ hojas = \lfloor e\ /\ k_{mín} \rfloor$$

e = nº total de entradas

- El nº de nodos en el **nivel *n-1*** depende del número de nodos del nivel *n*

$$n^o\ nodos\ (n\text{-}1) = \lfloor n^o\ nodos\ (n)\ /\ m_{mín} \rfloor$$

- Cuando se llega a un nivel con un solo nodo (la raíz), este será el **nivel 1**.
  (se tiene que el nivel *n-x=1,* y se puede despejar *n = profundidad del árbol*)

- **Tamaño máximo** del fichero índice:
  se calcula como la suma de los nodos necesarios
  para cada nivel ($\sum_{i=1}^{n}$ nodos(i)) multiplicado por el tamaño de un nodo.

Tema 7.3.3: Indización en Árbol B+

Consideraciones finales

- El encadenamiento de hojas proporciona mecanismos de acceso alternativo.
- Ejemplos: (índice en árbol B^+ con clave indización '*fecha* (dd-MM-AAAA)')
  - procesos a la totalidad ordenados
    Ejemplo: sacar un listado de todos los registros ordenados cronológicamente
  - procesos selectivos con tasa de actividad elevada
    Ejemplo: recuperar todos los registros con fecha en mes de 'Mayo'
  - procesos ordenados con varios resultados (*rangos*) → acceso mixto
    Ejemplo: recuperar todos los registros entre el 01-05-2005 y el 30-06-2005
    - Los accesos mixtos consisten en recuperar la primera entrada (01-05-2005) a través del árbol, y el resto de entradas se recuperarán con el encadenamiento

- Se puede **organizar** un **fichero de datos en árbol**

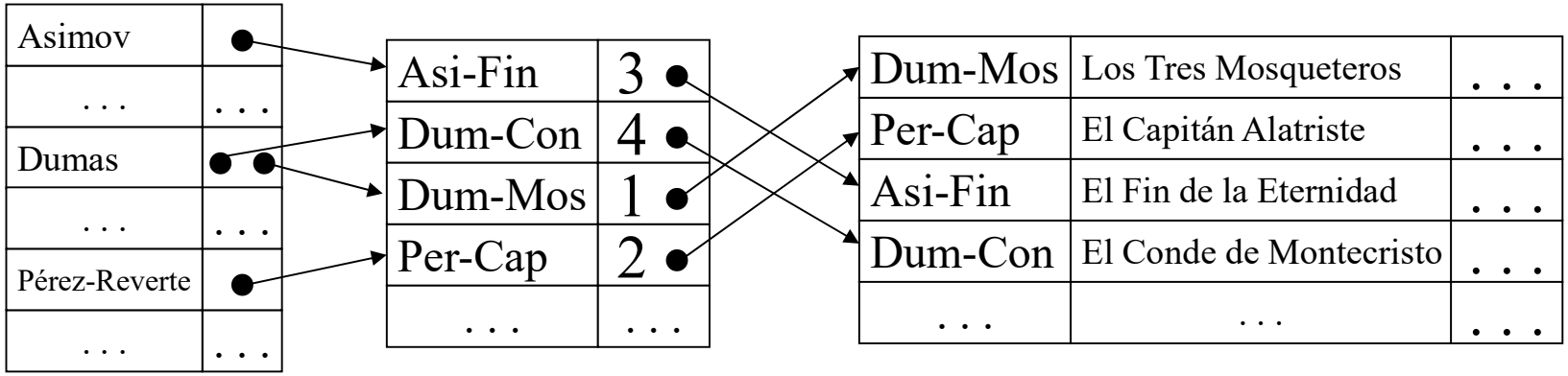
  → es como tener un f. secuencial con part. celular y un índice B^+ no denso.

- Las mejoras logradas con árboles B^+ y B* son combinables

Tema 7.4.1: Estructuras Especiales

Índice Intermedio

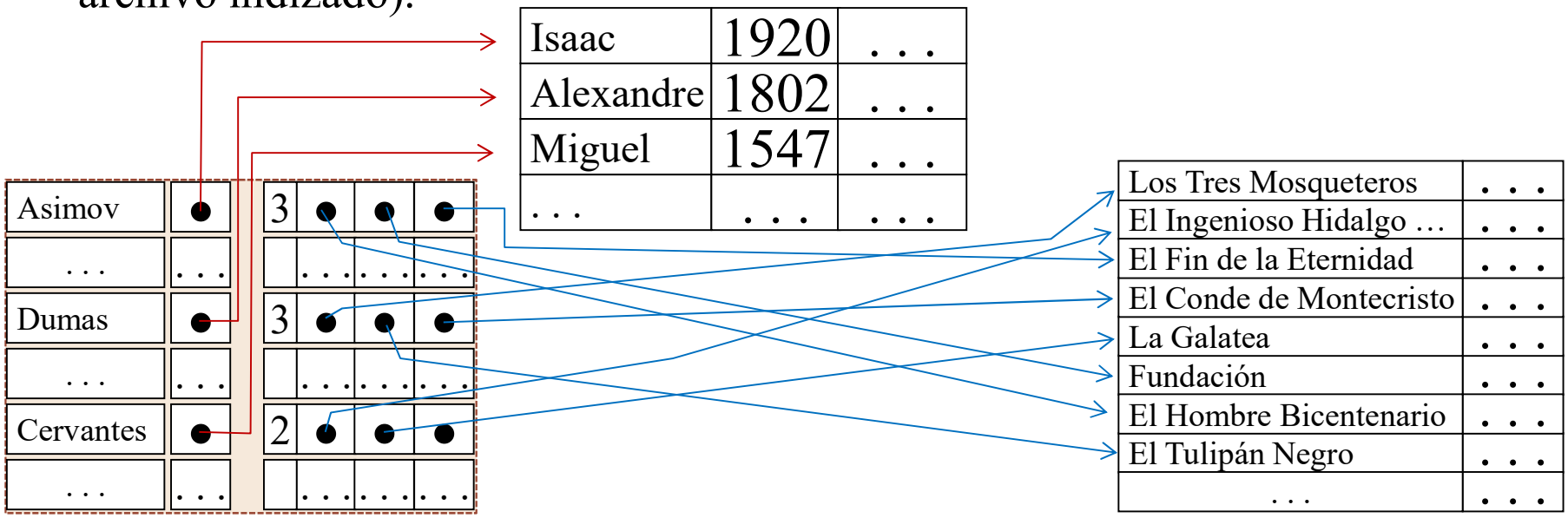
- Si los registros (en el f. de datos) cambian de ubicación, es necesario actualizar todos los índices de ese archivo.
- *Índice Intermedio*: índice primario cuyos punteros apuntan a los datos, y el resto de los índices apuntan a este. Al cambiar los registros de ubicación, sólo es necesario actualizar punteros en este índice.
- Este índice debe ser muy eficiente: bloqueado en memoria privilegiada de tamaño **reducido**, y (casi) **constante** (poco o nada volátil)



Tema 7.4.2: Estructuras Especiales

Índice Agrupado o Cluster

- Dos (o más) índices sobre distintos archivos con la misma CI y valores validados (por integridad referencial) pueden combinarse (***index join***).
- También puede crearse una estructura única de indexación (índ. *cluster*).
- La entrada tendrá una clave de indización y uno o más esquemas de punteros (un puntero o una lista de punteros, según sea la naturaleza de la clave en cada archivo indizado).

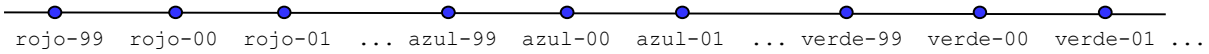


Tema 7.4.3: Estructuras Especiales

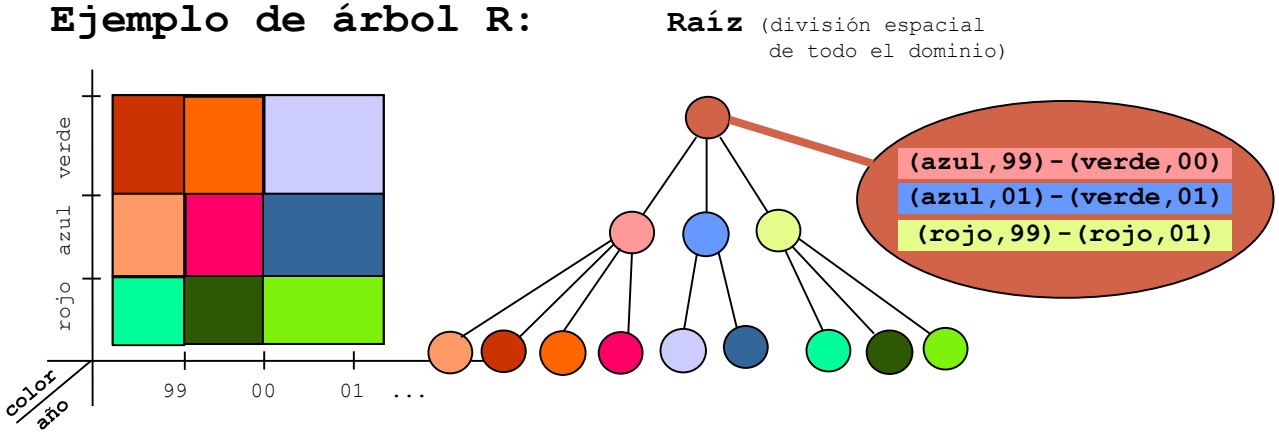
Índice Multiclave: el árbol R

- La indización multi-clave también admite la creación de índices especiales que no estén basados en una clave, sino en varias simultáneamente
- Es el caso del árbol R, una evolución del árbol B^+ para *d* dimensiones (claves). Propuesto por Güttman (1984)
- Cada entrada no es un punto en una línea, sino un intervalo d-dimensional:

Distribución lineal:



Ejemplo de árbol R:



- raíz con 2 e. o más (salvo si es hoja)
- altura balanceada
- los intervalos pueden solaparse (al buscar un intervalo hay que recorrer todos los descendientes que intersecan con el intervalo en cuestión)

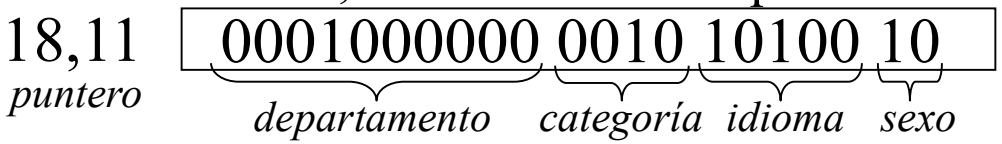
Tema 7.4.4: Estructuras Especiales

Esquemas de bits (BITMAP)

Un *esquema de bits* para un campo es un vector de valores booleanos.
A cada **valor** del dominio se le hace corresponder una **posición**.

Ejemplo: *idioma* (castellano, inglés, francés, alemán, italiano)

- Si la cardinalidad (#valores) es alta, puede indizarse un subconjunto (índ. parcial).
- Admite la multivaluación (varios valores para la misma entrada)
- Puede ser simple o multiclave, concatenando esquemas de varios campos



- Puede utilizarse como *directorio de ocurrencia*
- ***Proceso serial*** (excepción: acceso invertido sobre bitmap)

Tema 7.4.4: Estructuras Especiales

Máscaras sobre BITMAP

- **Máscaras para condiciones de igualdad:**
  - Se realizan con un bit para cada posible valor, en el conjunto { 0 , 1 }
  - La selección comprueba la condición **S AND Q = Q**
    - Ejemplo: empleadas del dpto. informática que sólo sepan castellano

      Q = 0001000000 0001 10000 10

- **Máscaras con bits que admitan cualquier valor:**
  - Se realizan con un bit para cada posible valor, en el conjunto { 0 , 1 , q }
  - La selección comprueba en **lógica trivaluada** la condición **S XOR - Q = 1**
    - Ejemplo: miembros del departamento de informática que sólo sepan inglés

      Q = qqq1qqqqqq qqqq 01000 qq

Tema 7.4.4: Estructuras Especiales

Esq. de bits Simples vs. Multiclave

- Es conveniente que el diseño de los esquemas de bits se realice atendiendo a las necesidades de procesamiento.
  - **Ejemplo** para claves A (5 bits) y B (10 bits) en un fichero de 1000 registros, con puntero de 3 bytes, y tamaño de bloque 2KB:
    - Tamaño índices:

T(A)=1000·(1+3) → 2 bloques T(B)=1000·(2+3) → 3 bloques	*Solución 1*
T(A+B)=1000·(2+3) → 3 bloques	*Solución 2*

→ caso i) Frecuencias relativas de uso: f(A)=0.7; f(B)=0.2; f(A+B)=0.1

- coste medio (solución 1) = 0.7*2 + 0.2*3 + 0.1*(2+3) = **2.5 accesos**
- coste medio (solución 2) = 0.7*3 + 0.2*3 + 0.1*3 = **3** accesos

→ caso ii) Frecuencias relativas de uso: f(A)=0.1; f(B)=0.4; f(A+B)=0.5

- coste medio (solución 1) = 0.1*2 + 0.4*3 + 0.5*(2+3) = **3.9** accesos
- coste medio (solución 2) = 0.1*3 + 0.4*3 + 0.5*3 = **3 accesos**

Tema 7.5.1: Procesos sobre Índices

Los índices son herramientas auxiliares versátiles que pueden ser utilizados de distintas formas con diversos propósitos.
Los usos más habituales son (1/2):

- ACCESO SIMPLE O UNÍVOCO (*index unique scan*): acceso que devuelve una entrada (o ninguna), con uno o más punteros.
  Ejemplos: - Index(DNI) → … where DNI = 1234567;
  - Index(nombre, apellido) → … where nombre = 'John' and apellido='Smith';

- ACCESO EN RANGO (*index range scan*): devuelve varias entradas en un rango ordenado. Según la naturaleza del índice, puede encontrarlas en un subárbol de un árbol B, en un segmento del encadenamiento de las hojas de un B+, o en búsqueda dicotómica extendida de un índice ordenado.
  Ejemplos: - Index(nombre, apellido) → … where nombre = 'John';
  - Index(fecha_ini) → … where fecha_ini between A and B ;

Tema 7.5.1: Procesos sobre Índices

Usos más habituales de un índice (2/2):

- ACCESO ALTERNO (*index skip scan*): acceso que utiliza un subconjunto de la clave de indización que no es prefijo.
  Ejemplo: - Index(nombre, apellido) → … where apellido = 'Smith';

- ACCESO A LA TOTALIDAD ESTRUCTURADO (*index full scan*): acceso que sigue la estructura de índice y recupera todas las entradas en orden.
  Ejemplo: - Index(nombre, apellido) → … order by nombre,apellido ;

- ACCESO A LA TOTALIDAD NO ESTRUCTURADO (*index fast full scan*): lee todos los nodos del índice aprovechando la secuencialidad de disco.
  Ejemplo: - Index(nombre, apellido) → select nombre,apellido from … ;

Tema 7.5.1: Proc. Ficheros Indizados

- Recuperación de **un registro** por clave identificativa (*exact match*):
  - 1 resultado, sin índice: coste según organización base
  - 0 resultados, sin índice: frecuentemente, coste más elevado
  - con índice arbóreo: pocos accesos índice + lectura de un cubo
- Comprobación de ***Unicidad*** (PK/UK) y ***Existencia*** (integridad FK):
  - Equivalente a la localización por clave identificativa
  - Índice suele ser ventajoso (en algunos SGBD, es obligado)
- Recuperación de **varios registros**:
  - Si *tasa de actividad* baja, coste(leer índice + *k* cubos) < coste(full scan)
  - Si *tasa de actividad* elevada, puede interesar ignorar el índice
  - Si proceso ordenado e índice ordenado, el ahorro de coste ordenación puede compensar la diferencia con el coste del full scan. Además, el índice proporciona resultados paulatinamente (no todo de una vez).

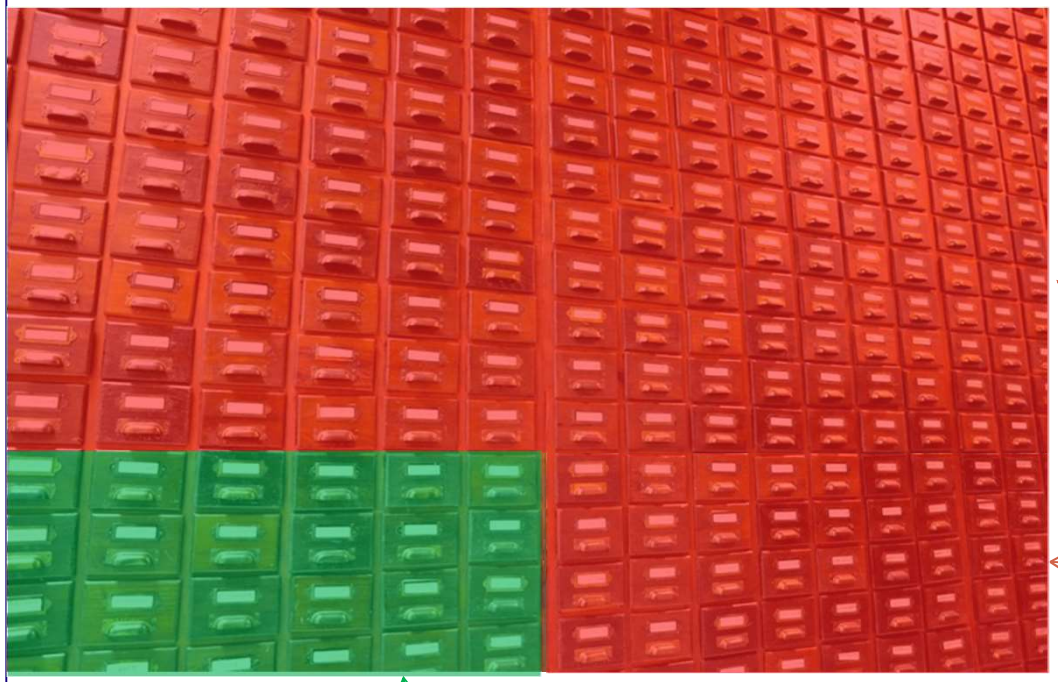
Tema 7.5.1: Procesos Indizados

• **Ejemplos**: fichero serial con 10^6 registros en 10^5 cubos; tenemos dos proc: buscar por K_1 (P_1) devuelve 1 reg, y buscar por K_2 (P_2) devuelve $2 \cdot 10^4$ (media)

- Para P_1 ¿es preferible leer 1 cubo (+acc índice) o hacer un full-scan?
- Si el soporte tiene beneficios secuenciales (disco), y el fichero de datos está desfragmentado, ¿compensa leer $2 \cdot 10^4$ cubos aleatorios o el full scan?
- Si el soporte no tiene ese tipo de beneficios ¿qué opción es preferible?
- ¿Y si en lugar de $2 \cdot 10^4$ fueran $2 \cdot 10^5$ registros? (cardinalidad(K_2)=5)
- Si la PK es K_1, y necesito insertar ¿miro que el nuevo valor para K_1 esté en el índice, o es preferible hacer un recorrido total (*full scan*)?
- Si K_3 es FK que referencia a una UK (no PK) e inserto un registro, para comprobar la integridad ¿hago un full scan del fichero referenciado?
- Si la búsqueda involucra varias claves privilegiadas ¿cuál escojo?

Tema 7.5.2: Localización Física

Filtrado de Cubos (*cjto dir. relevantes*)



¿ En qué cajón ha puesto mi hijo el móvil ?

A estos no llega…

De éstos no tiene la llave …

Puedo restringir la búsqueda a este subconjunto…

- Las *claves privilegiadas* permiten filtrar
- Los cubos que queden se recorren todos, a menos que se encuentre lo que se busca (*sel. identificativa*, en media (N+1)/2 accesos)

Tema 7.5.2: Filtrado por Índice

- El conjunto de direcciones relevantes puede ser un conjunto de punteros, o un simple vector de booleanos (uno por cada cubo en el fichero datos)

- Al seleccionar, se hará un **filtrado** gracias a la aplicación de índices.
  - una dispersión (hash) también puede filtrar sin coste adicional.
  - un índice primario proporciona un puntero → filtrado máximo
  - un índice secundario suele filtrar menos (puedo aplicar varios)

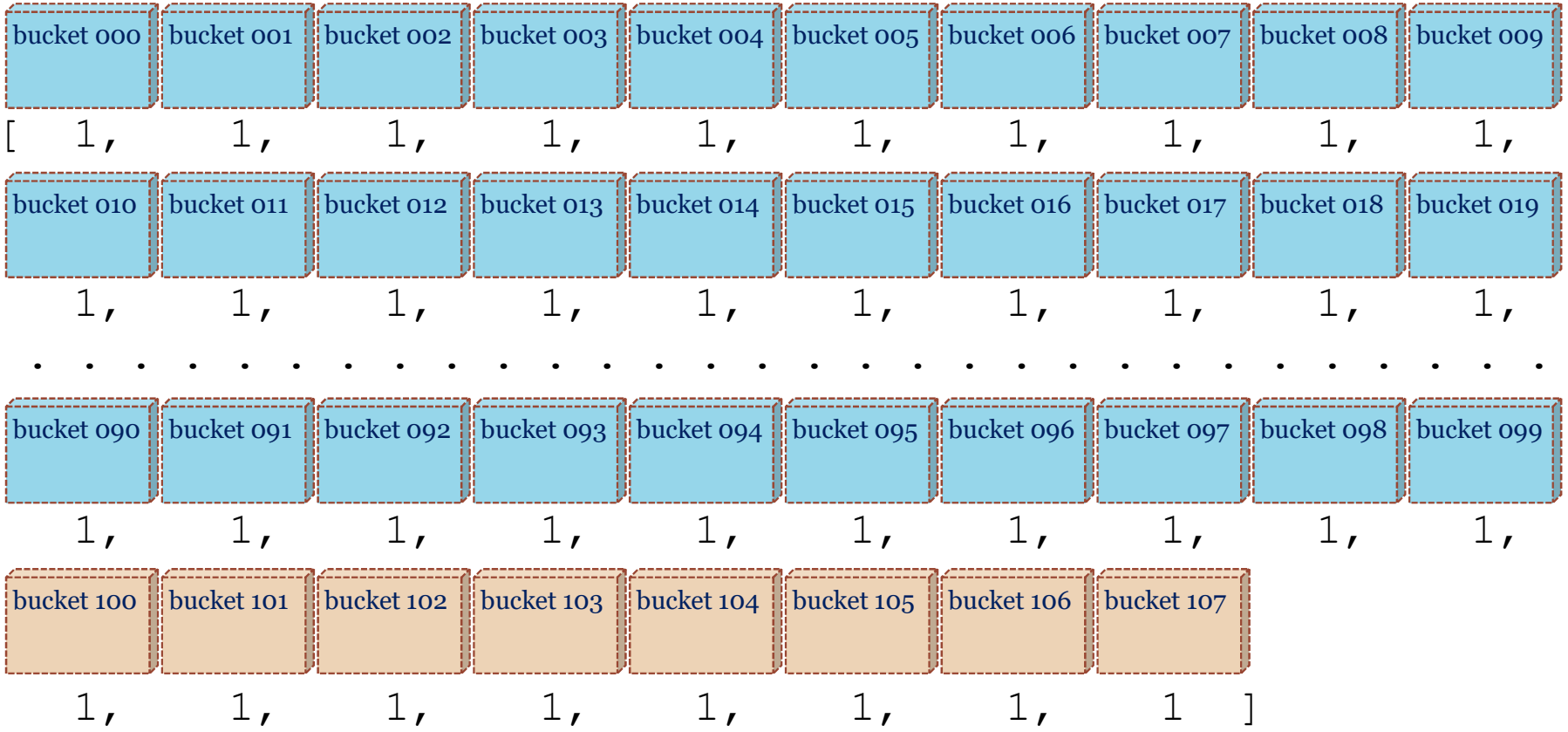
- La *selección indizada multiclave* consiste en aplicar varios filtros mediante otros tantos índices; se obtienen varios conjuntos que se operan (algebraicamente) para obtener el conjunto resultado global:
  - Ejemplos: sean *a* y *b* consultas simples; A y B sus respectivos cjtos. resultado

$$a \wedge b \equiv A \cap B$$
$$a \vee b \equiv A \cup B$$
$$a \wedge \text{-}b \equiv A - B$$
$$\text{-}\, a \equiv \aleph - A$$

(donde $\aleph$ es el espacio de direcciones global)

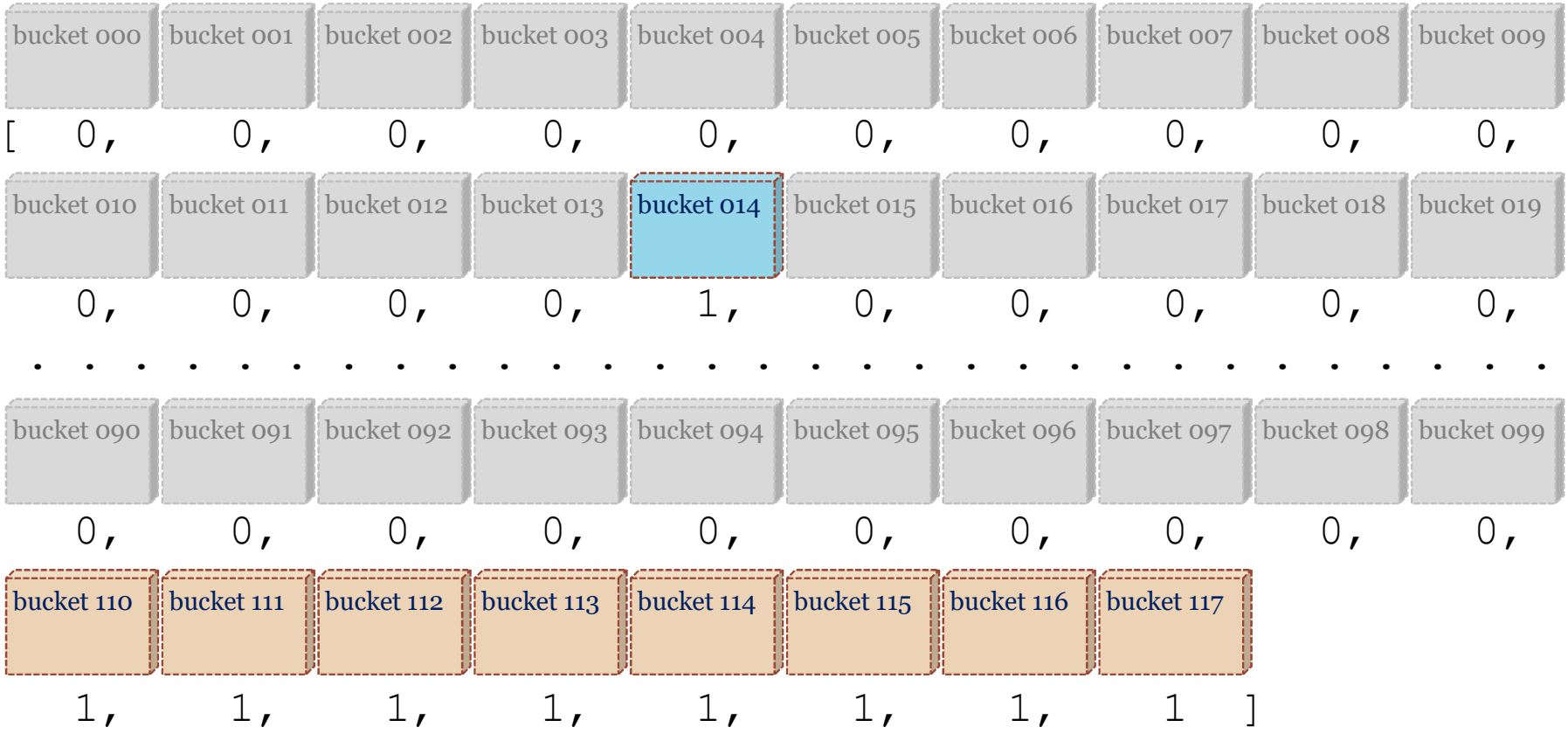
Tema 7.5.2: Ejemplo cjto. dir. relevantes

- Ejemplo: archivo disperso sobre N=100 con área desbordamiento N'=8



Tema 7.5.2: Ejemplo cjto. dir. relevantes

1. Filtra por CD='Juan', que produce la dirección 14



- …y descarta 99 cubos ! (el área desbordamiento no se descarta)

Tema 7.5.2: Ejemplo cjto. dir. relevantes

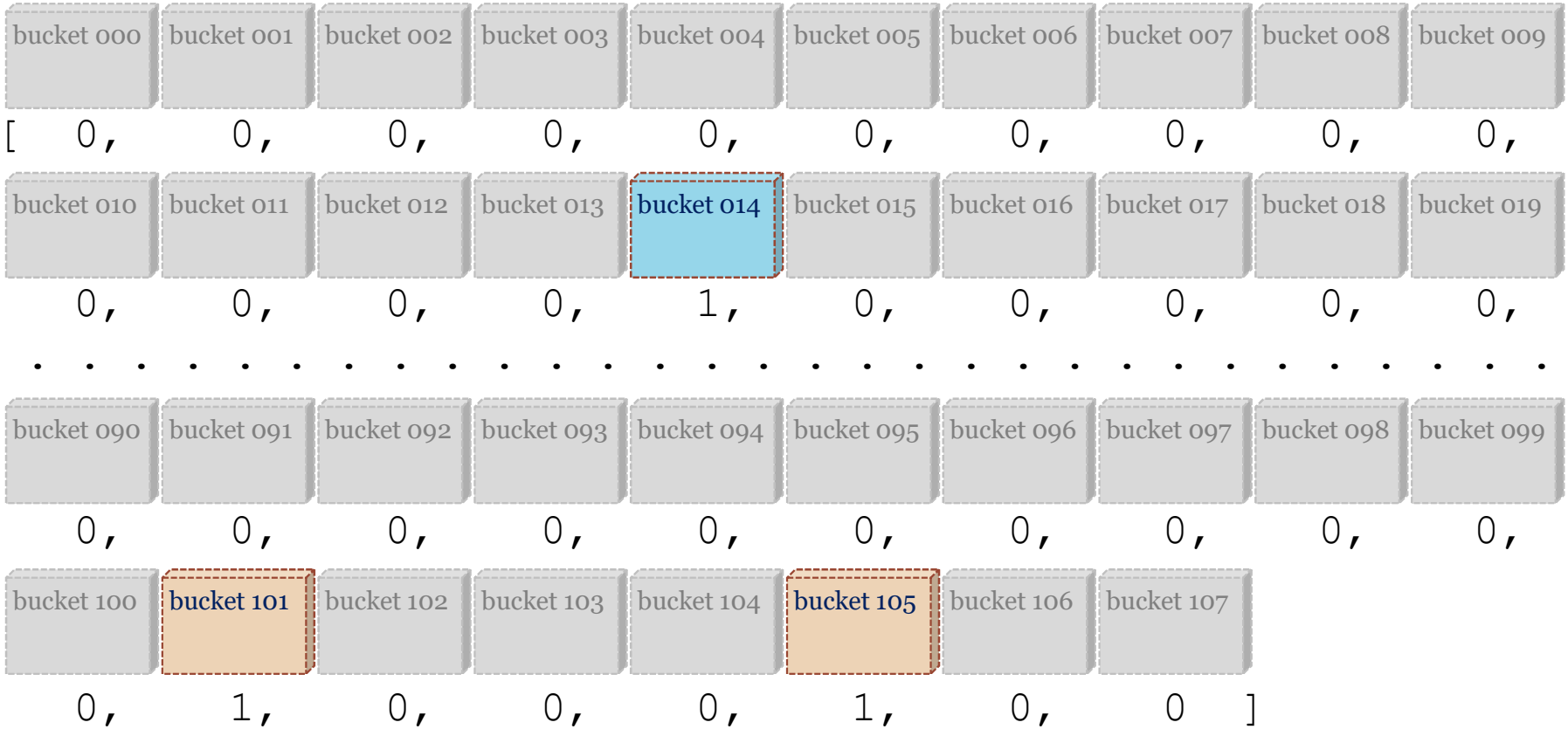
2. …y por índice *apellido*: 'Smith'·{3,9,14,21,28,44,56,62,89,94,101,105}

bucket 000	bucket 001	bucket 002	bucket 003	bucket 004	bucket 005	bucket 006	bucket 007	bucket 008	bucket 009
[0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
bucket 010	**bucket 011**	**bucket 012**	**bucket 013**	**bucket 014**	**bucket 015**	**bucket 016**	**bucket 017**	**bucket 018**	**bucket 019**
0,	0,	0,	0,	1,	0,	0,	0,	0,	0,
. . . . .									. . .
bucket 090	**bucket 091**	**bucket 092**	**bucket 093**	**bucket 094**	**bucket 095**	**bucket 096**	**bucket 097**	**bucket 098**	**bucket 099**
0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
bucket 100	**bucket 101**	**bucket 102**	**bucket 103**	**bucket 104**	**bucket 105**	**bucket 106**	**bucket 107**		
0,	1,	0,	0,	0,	1,	0,	0]		

- Al ser 'conjunción', se calcula la intersección de cjtos. (op. más habitual)
- …descarta todo menos las 3 direcciones comunes

Tema 7.5.2: Ejemplo cjto. dir. relevantes

3. Recorre el archivo (*fullscan*), omitiendo cubos marcados con cero



- **Lee 3 cubos** (como mucho: si la selección fuera unívoca, se detendría en el primer resultado, realizando en media (3+1)/2 accesos de lectura)

Tema 7.5.3: Acceso Invertido

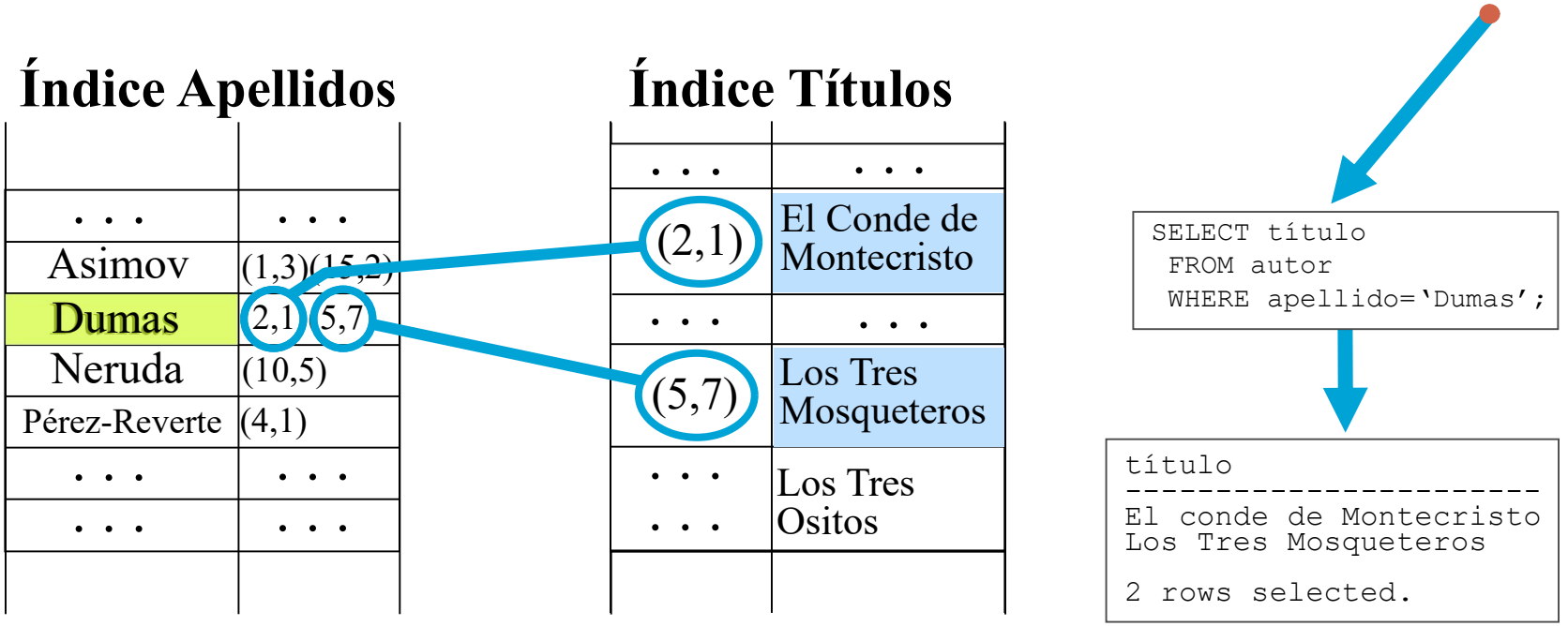
- El acceso invertido es un tipo de acceso indizado multiclave orientado a optimizar el coste de acceso en procesos muy concretos
- Se trata de procesos en los que se pretende averiguar información delimitada de ciertos archivos con condiciones muy concretas.
- Del tipo: *¿Cuál es el valor del campo X en el archivo Y, para los registros cuyo campo Z vale 'valor'?*

dos o más claves (pero no muchas)

- El acceso invertido procurará averiguar toda esta información **accediendo sólo a los índices** (sin acceder al archivo de datos)
- Sus punteros relativos deben localizar unívocamente cada registro:
  - → punteros con parte alta (***cubo***) y parte baja (***posición*** *en el bloque*)
  - → Oracle usa el ROWID, que contiene *dbblock* (cubo), *position*, y *datafile*

Tema 7.5.3: Acceso Invertido

- Se ejecutarán primero las condiciones (para obtener conjunto resultado)
- Después se busca en los 'índices objetivo' (incógnitas), ¡pero al revés!:
  a partir de cada dirección (ptro. relativo) buscamos el valor (ptro. lógico)
- **Ejemplo**:
  ¿Cuáles son los títulos de los libros de cualquier autor cuyo apellido sea 'Dumas'?



Tema 7.5.3: Acceso Invertido: Coste

- El coste del acceso invertido es la suma de los costes de sus dos partes:
  - Selección: *Obtención de los punteros*
    - *Listas invertidas no ordenadas: coste máx.* ***n****; medio (n+1)/2*
    - *Listas invertidas ordenadas:* $\boldsymbol{\log_2 (n+1)}$
    - *Esquemas de bits:* ***n***
    - *Otro tipo de índice: el coste correspondiente a esa estructura*
  - Proyección: *obtención de Claves correspondientes a los punteros*
    - *Esquemas de bits con puntero implícito:* ***mín (n, r)***
    - *Cualquier otro caso:* ***n***
- Simbología: ***n*** es el número de bloques del índice; ***r*** es el número de resultados
- Si hubiera varios índices implicados en la condición o en la proyección, el coste en cada parte sería la suma de los costes individuales de cada índice implicado.

Tema 7.5.3: Acceso Invertido

- El acceso invertido es eficiente si requiere acceder a **pocos índices**
  (si accediera a varios no contenidos en M_{int}, podría costar más que acceder a los datos).
- También es eficiente si la misma **consulta se repite** frecuentemente
  (accediendo a los mismos índices, se dispara la tasa de acierto a memoria intermedia).
- Los índices que soportan este acceso (puntero ext. doble precisión) se denominan **índices invertidos**. Los secundarios también se denominan listas invertidas.
- En este tipo de acceso, los índices bitmap son eficientes en dominios reducidos, mientras que las listas invertidas lo son en dominios de cardinalidad elevada.
- Un **fichero invertido** es el que soporta este tipo de acceso (dos o más índices invertidos). Un **fichero totalmente invertido** tiene todos los campos invertidos.
- En un fichero totalmente invertido el área de datos es redundante (prescindible). Sin embargo, se suele mantener para (a) evitar el alto coste de la recomposición de registros; y (b) mantener doble almacenamiento base (**sistema dual**).

Tema 7.5.3: Sistema Dual

- Un **sistema dual** es el que presenta redundancia controlada para disfrutar de alternativas de acceso físico (dependiendo del tipo de consulta, se utiliza un recurso u otro). Por ejemplo, consultas de pocos atributos por acceso invertido, y el resto sobre el área de datos.
- La duplicidad de almacenamiento en sistemas duales pone en riesgo la **consistencia**. Controlar la redundancia es costoso, porque las actualizaciones deben realizarse en varios almacenes de modo atómico.
- Además, también puede afectar a la **disponibilidad** del sistema, ya que el control del proceso debe esperar a recibir confirmación de todas las escrituras.
- Para mejorar la disponibilidad se puede sacrificar algo de consistencia: uno de los almacenes se actualiza efectivamente (al realizar la operación) y el otro eventualmente (por ejemplo, los índices se actualizan cuando el sistema está ocioso), mejorando la disponibilidad con poco perjuicio en las consultas.

Tema 7.5.3: Fusión de Índices (*Index Join*)

- En ficheros invertidos, es habitual combinar varios índices a través de su identificador de registro (puntero o ROWID) hasta tener un índice que contiene todos los atributos involucrados en la consulta. A este proceso se le conoce como fusión de índices (index join).

- El coste en accesos de la fusión (completa) de índices es el de recorrer a la totalidad (full scan) los índices involucrados en el proceso.

- La fusión de índices es eficiente si se hace sobre índices en mapa de bits. Algunos SGBD (como Oracle) contemplan convertir otros índices secundarios (árbol B+) en bitmap en memoria para realizar este proceso.

- El coste de la *index join* es proporcional a la cantidad de índices involucrados. Recrear el área de datos (recomposición total) es un proceso costoso, y si se realiza de modo frecuente es preferible contar con un sistema dual.