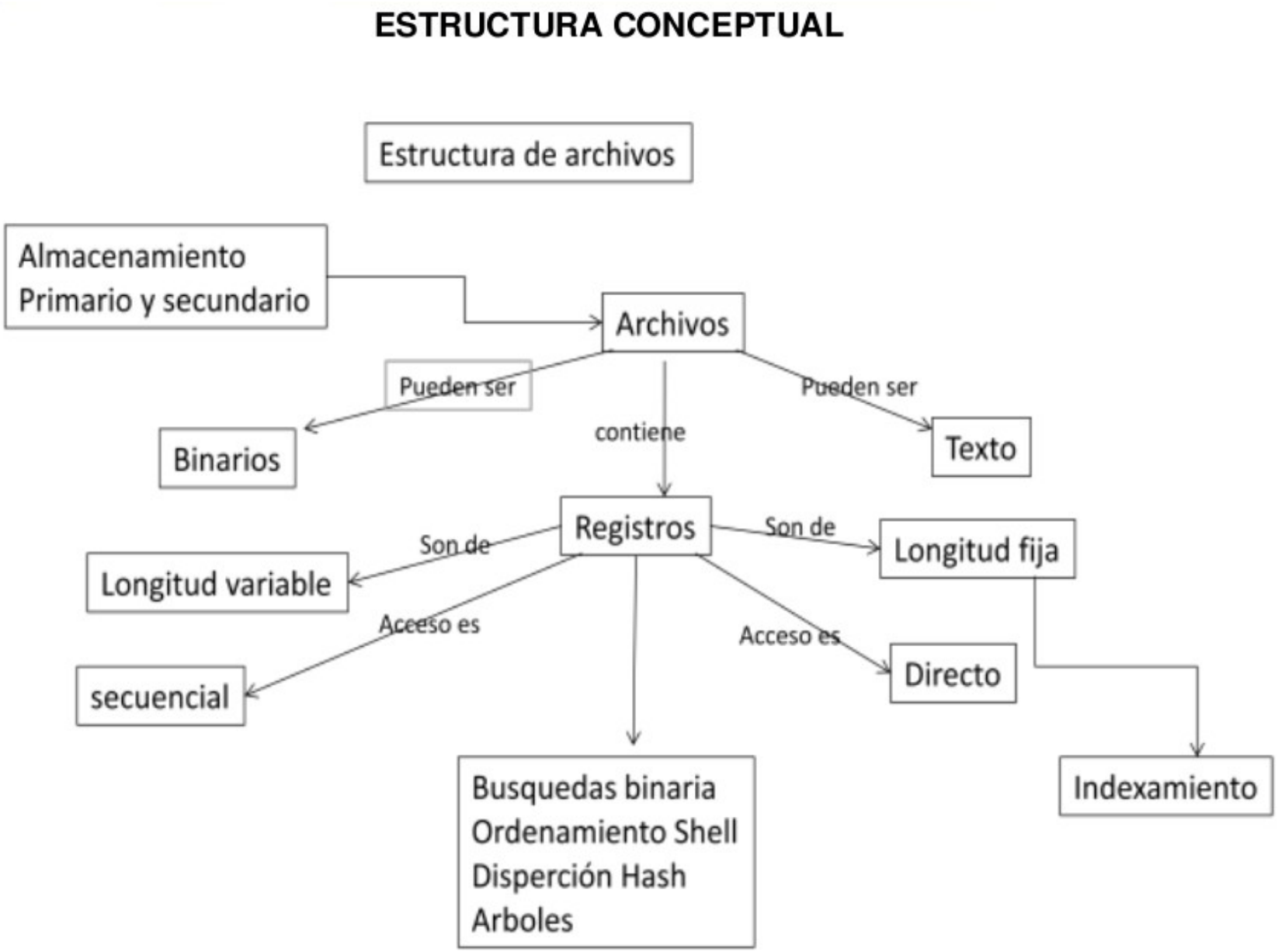
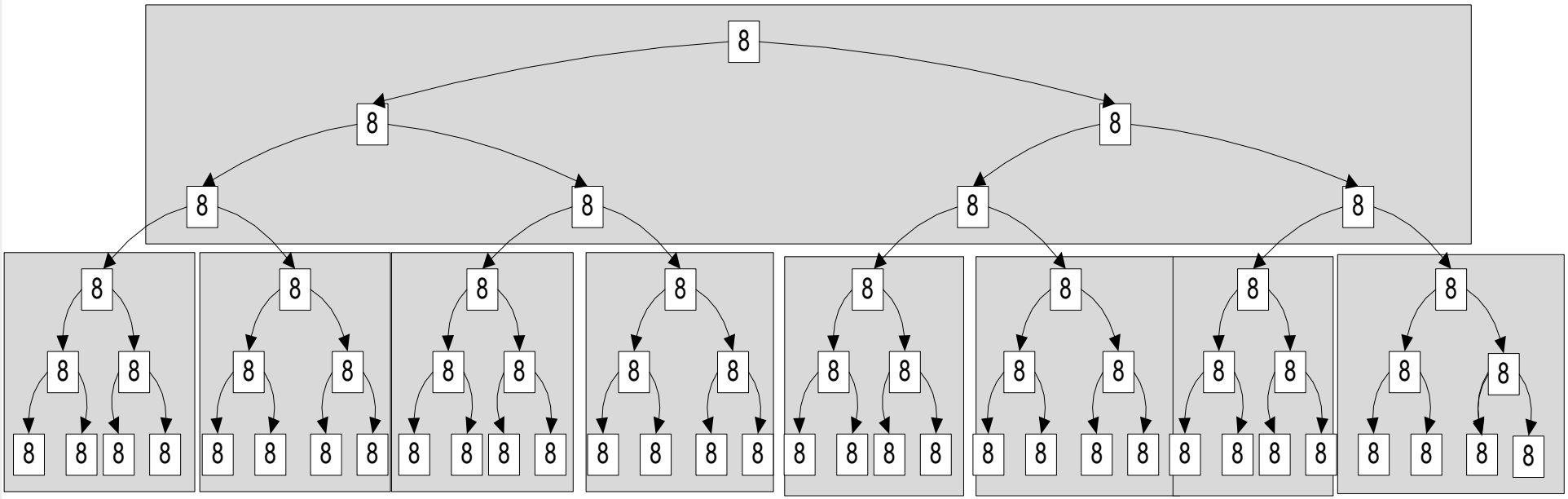
Conceptos


ESTRUCTURA CONCEPTUAL
Estructura de archivos
Almacenamiento
Primario y secundario
Archivos
Pueden ser
Binarios
contiene
Pueden ser
Texto
Registros
Son de
Longitud variable
Son de
Longitud fija
Acceso es
secuencial
Acceso es
Directo
Busquedas binaria
Ordenamiento Shell
Disperción Hash
Arboles
Indexamiento

Arboles Binarios Paginados

- Problemas de almacenamiento secundario, buffering, páginas de memoria, varios registros individuales, minimiza el número de accesos

- Problema: construcción descendente, como se elige la raíz?, cómo va construyendo balanceado?



Arboles B

Motivación:

- Los sistemas de almacenamiento masivo suelen tener un **tiempo de acceso** mucho mayor que el **tiempo de transferencia**: La localización de un elemento es mucho más costosa que la lectura secuencial de datos, una vez localizados.

- Esto se aplica sobre todo a discos duros, pero también, aunque en menor medida, a memorias de estado sólido (flash) e incluso a memorias volátiles.
- Esto supone un problema para estructuras enlazadas, como los árboles AVL, donde las operaciones acceden a bastantes nodos de pequeño tamaño.
- Para grandes volúmenes de datos, sería conveniente **reducir el número de accesos**, a cambio de que esos accesos contuvieran **elementos de mayor tamaño**.

Arboles B

Un Ejemplo:

El sistema de Una Obra Social trabaja con una base de datos de unas 2.500.000 tarjetas, ocupando cada una aprox. 1 Kb de datos.

Si se almacenan en un árbol AVL, su altura sería: $h = \log_{\phi} 2.500.000 = 31{,}9$

Lo que supone entre 25-31 accesos a disco para cualquier búsqueda de un elemento.

En cambio, si se almacenan en un Árbol B de orden 1.000 (aprox. 1 Mb por nodo) tendría altura 3, o 2 con una ocupación media del 80%.

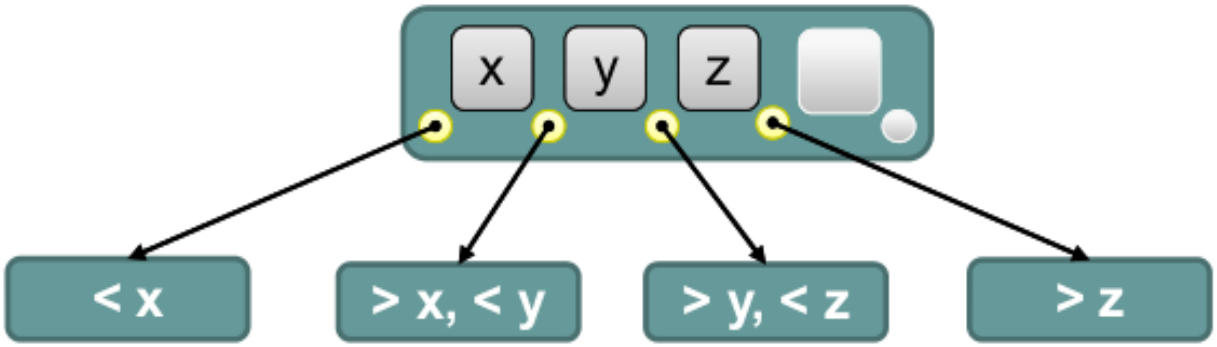
Sólo se necesitarían 1 ó 2 accesos a disco (la raiz reside en memoria) para cada búsqueda.

El orden para ambos casos es logarítmico, pero si el tiempo de acceso es dominante, la segunda solución sería 10-30 veces más rapida.

Arboles B

Árboles (a,b):

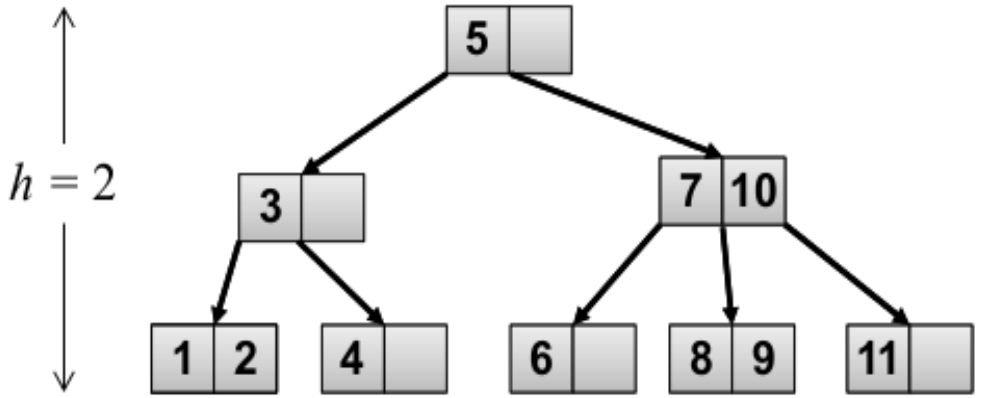
- Los **árboles (a,b)** son árboles generales (no binarios) donde cada nodo interno puede tener un número de hijos, ***m*+1**, en el rango **[a,b].**
- Cada nodo almacena ***m* claves** (elementos comparables por ≤), **ordenadas de menor a mayor**, que sirven para que se pueda usar como un **árbol de búsqueda**.
- El contenido típico de un nodo consiste en:
  - Un entero, $m \in [a-1, b-1]$, que indica el número de claves almacenadas.
  - Un vector, **c**, de capacidad ***b*-1**, que almacena las *m* claves.
  - Un vector, **e**, de capacidad ***b***, que almacena los *m*+1 enlaces a hijos.
- **Propiedad de ordenación**: Nodo **e[i]** almacena claves **menores** que **c[i]**



Arboles B

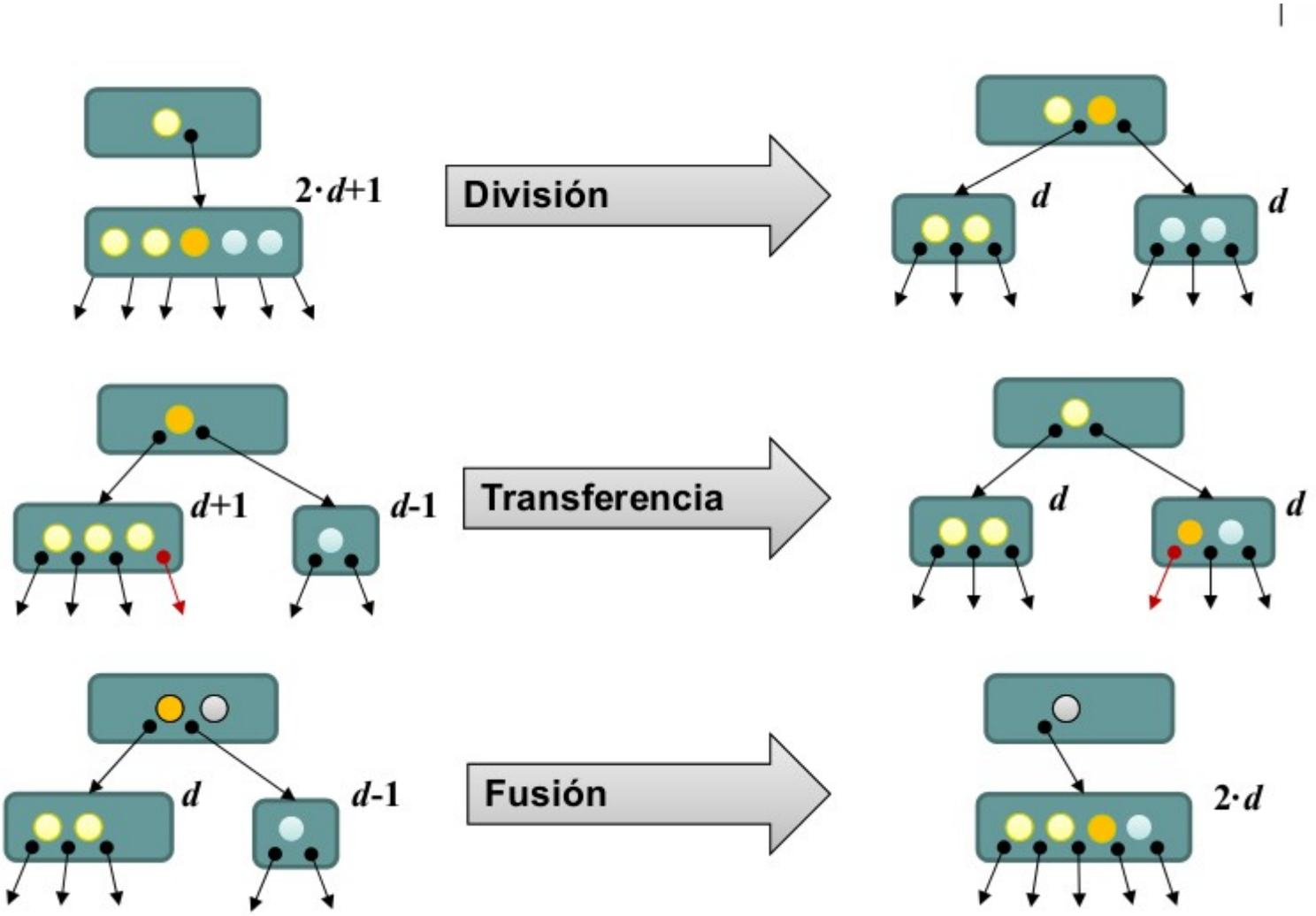
Árboles B:

- Un **árbol B** (Bayer-McCreight 1972) de **orden d** es un **árbol (d+1,2d+1)** con las propiedades adicionales siguientes:
  - La **raiz** puede tener **cualquier** número de claves.
  - Todas las hojas se encuentran a la **misma profundidad**, ***h***.
- La segunda propiedad garantiza que un árbol B es un **árbol equilibrado**: Su altura es logarítmica respecto al número de claves almacenadas.
- **Ejemplo**: Un árbol B de orden 1 es un árbol (2,3): Cada nodo puede contener 1 o 2 claves y tener 2 o 3 hijos.



Arboles B

Restructuraciones:



Arboles B

Búsqueda e Inserción:

- **Búsqueda:**
  - Se desciende desde la raiz hasta el nodo que contenga el elemento (o bien llegar a una hoja que no lo contenga).
  - En cada nodo se busca en el array de claves (búsqueda secuencial o binaria). Si no se encuentra, se pasa al hijo asociado a la primera clave mayor que el valor buscado (o el último hijo si el valor buscado es mayor que todas las claves).
- **Inserción:**
  - Se desciende (igual que en la búsqueda) hasta el nodo hoja que debería contener el elemento.
  - Se inserta en la posición adecuada del array de claves.
  - Si con ello se supera el número máximo de claves ($2d$), el nodo se **divide**, transfiriendo su clave en **posición media** al padre.
  - Es posible que el padre deba dividirse a su vez, y así con todos los ascendientes.

Arboles B

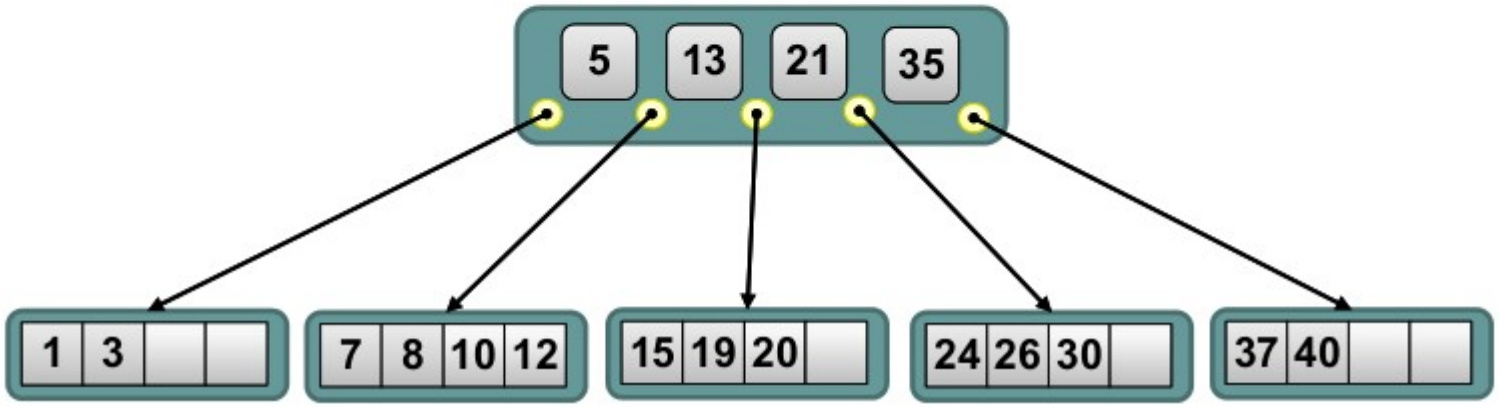
Borrado:

- **Borrado en nodo interno:**
  - Se desciende desde la raiz hasta el nodo que contenga el elemento a borrar.
  - Se intercambia con el máximo del hijo izquierdo o con el mínimo del hijo derecho (se elige el hijo con más claves).
  - Se pasa a borrar el elemento en el hijo (al final el borrado se produce en un **nodo hoja**)
- **Borrado en nodo hoja:**
  - Se elimina del array de claves (desplazamiento).
  - Si con ello el número de claves es *d*-1:
    - Se intenta una **transferencia** con el hermano izquierdo o derecho, el que contenga más claves.
    - Si no es posible (ambos tienen *d* hijos o no existen), se produce una **fusión** con el hermano izquierdo (o el derecho, si no existe).
    - La fusión toma un elemento del padre, por lo que éste a su vez puede necesitar transferencias o fusiones (y así con los ascendientes)

Arboles B

Inserción: Sin reestructuración valor 2

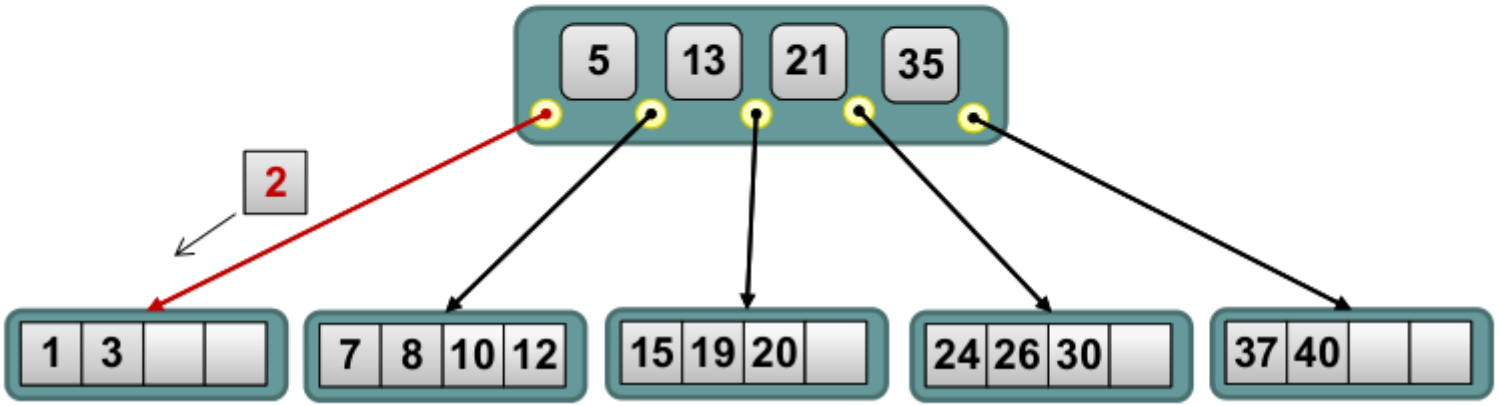
- Inserción del valor 2 en árbol B de orden 2 (árbol (3,5))



Arboles B

Inserción: Sin reestructuración valor 2

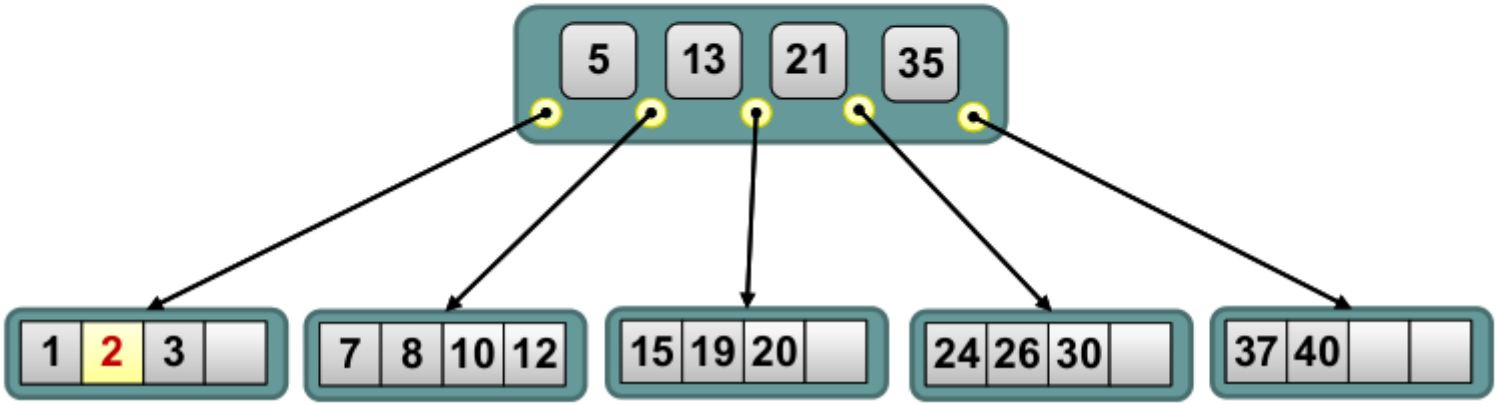
- Se busca el nodo hoja donde debe encontrarse el elemento



Arboles B

Inserción: Sin reestructuración valor 2

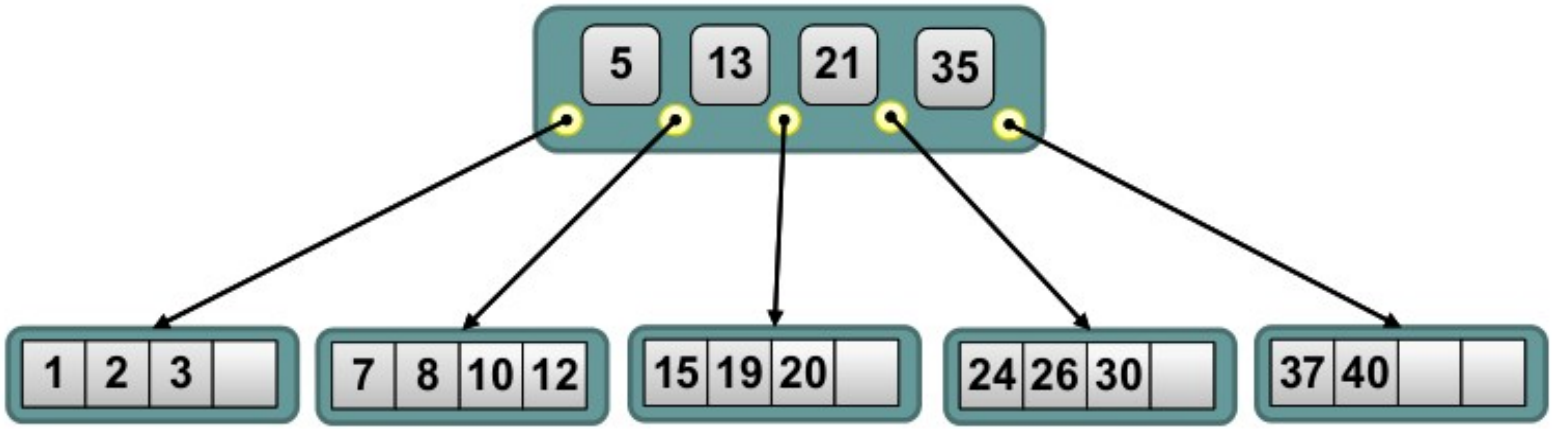
- Se inserta en orden en la hoja (desplazamiento)



Arboles B

Inserción Por División de Nodos (11)

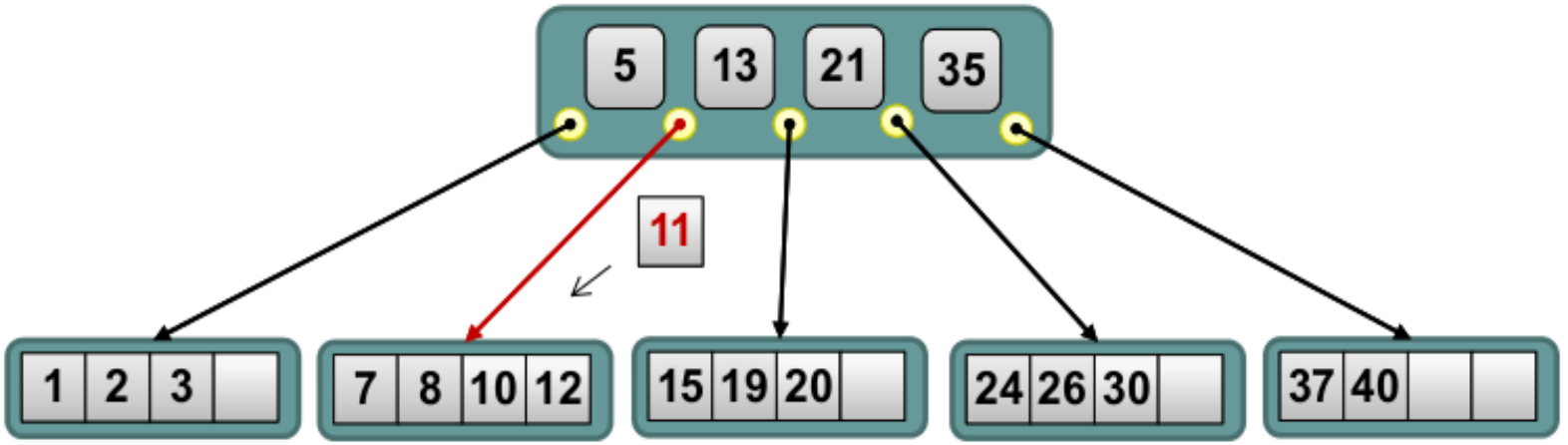
- Inserción del valor 11



Arboles B

Inserción Por División de Nodos (11)

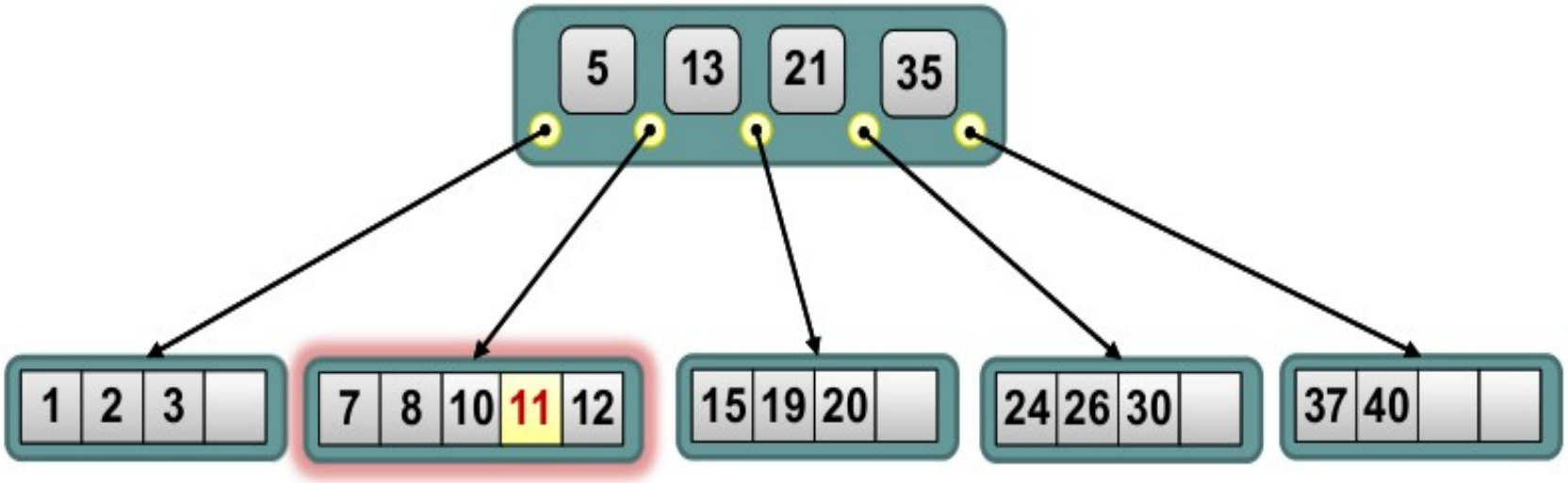
- Se busca el nodo hoja donde debe encontrarse el elemento



Arboles B

Inserción Por División de Nodos (11)

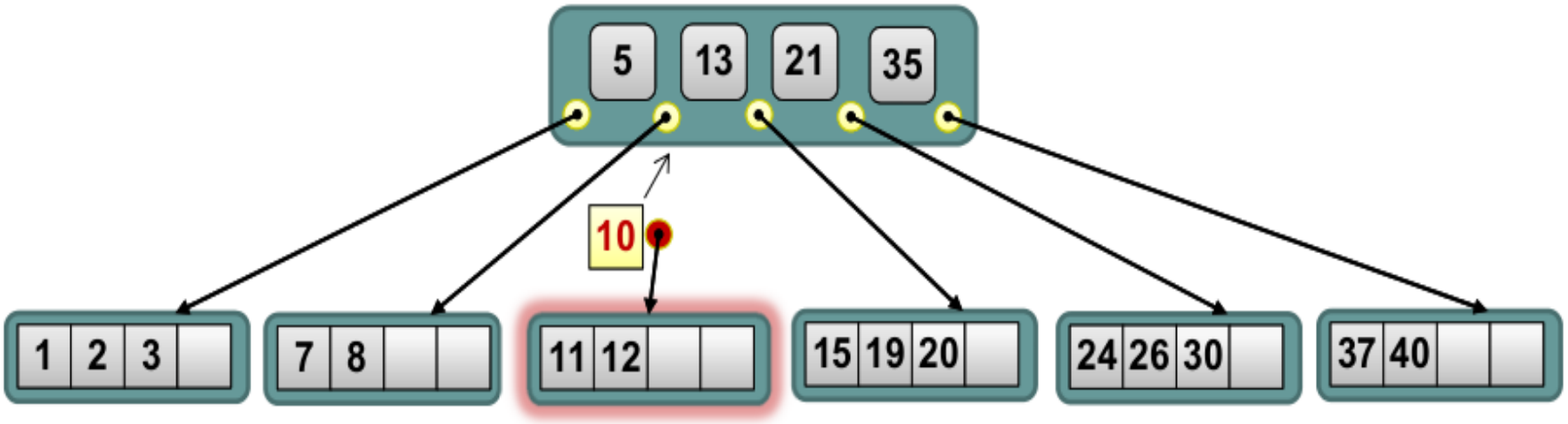
- Se inserta en el nodo. En este caso el nodo sobrepasa el límite de claves (4).



Arboles B

Inserción Por División de Nodos (11)

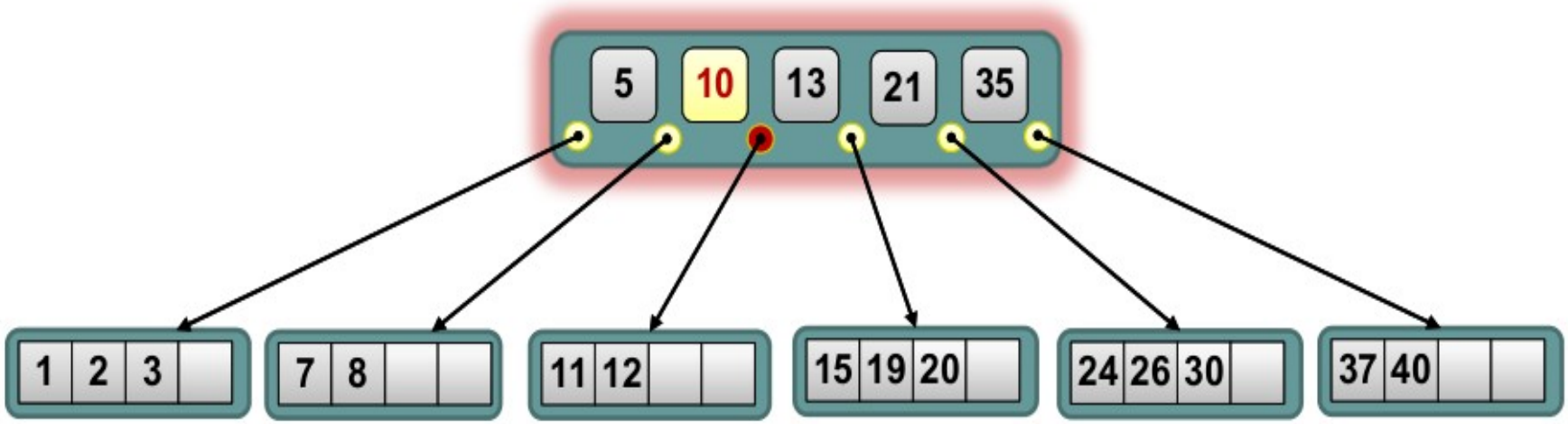
- Se crea un nuevo nodo y se traslada la mitad derecha de los elementos a él. El elemento en posición media (10), junto con el enlace al nuevo nodo, se envía al padre para su inserción



Arboles B

Inserción Por División de Nodos (11)

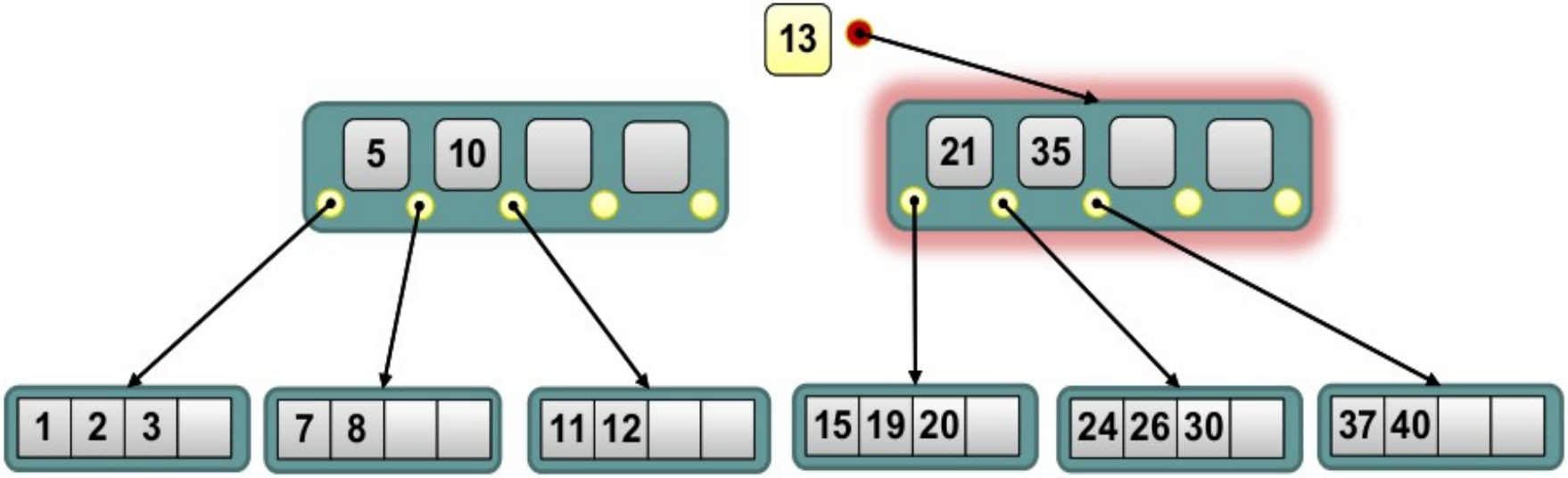
- Se inserta en el nodo padre. Se sobrepasa el límite de claves permitidas (4)



Arboles B

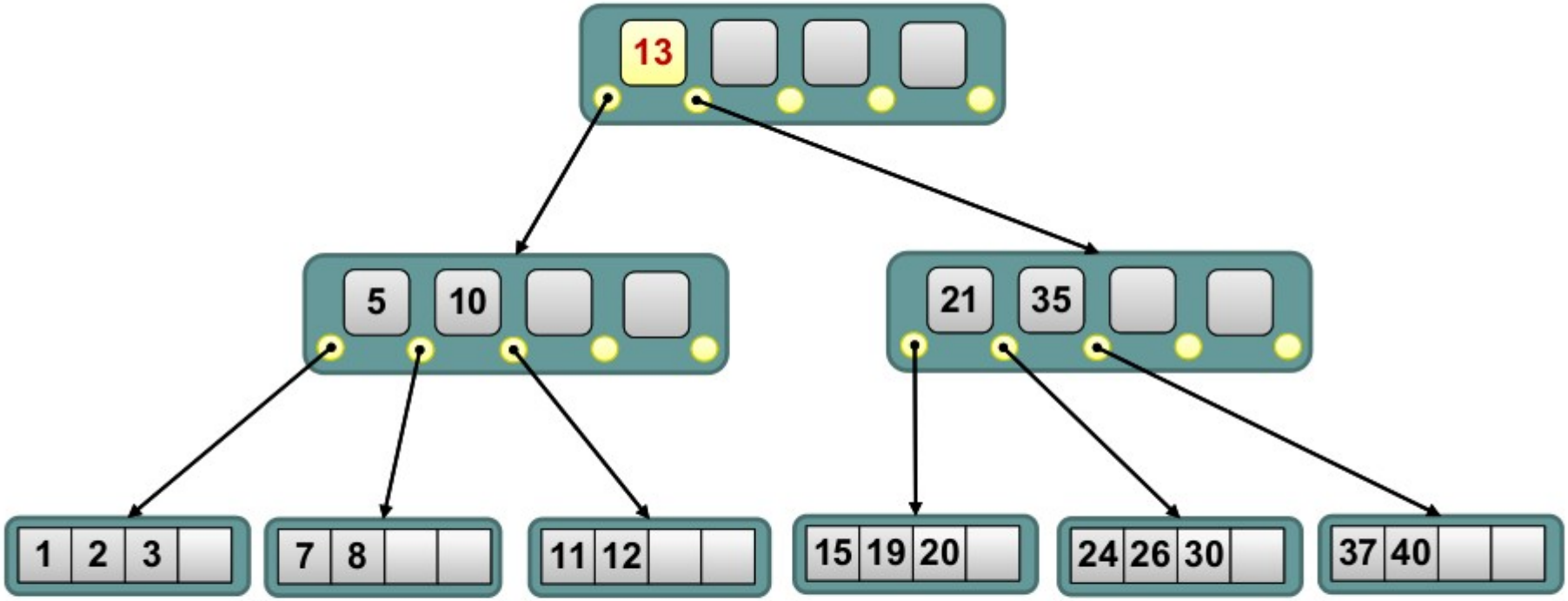
Inserción Por División de Nodos (11)

- Se crea un nuevo nodo y se traslada la mitad derecha de los elementos a él. El elemento en posición media (13), junto con el enlace al nuevo nodo, se envía al padre para su inserción



Arboles B

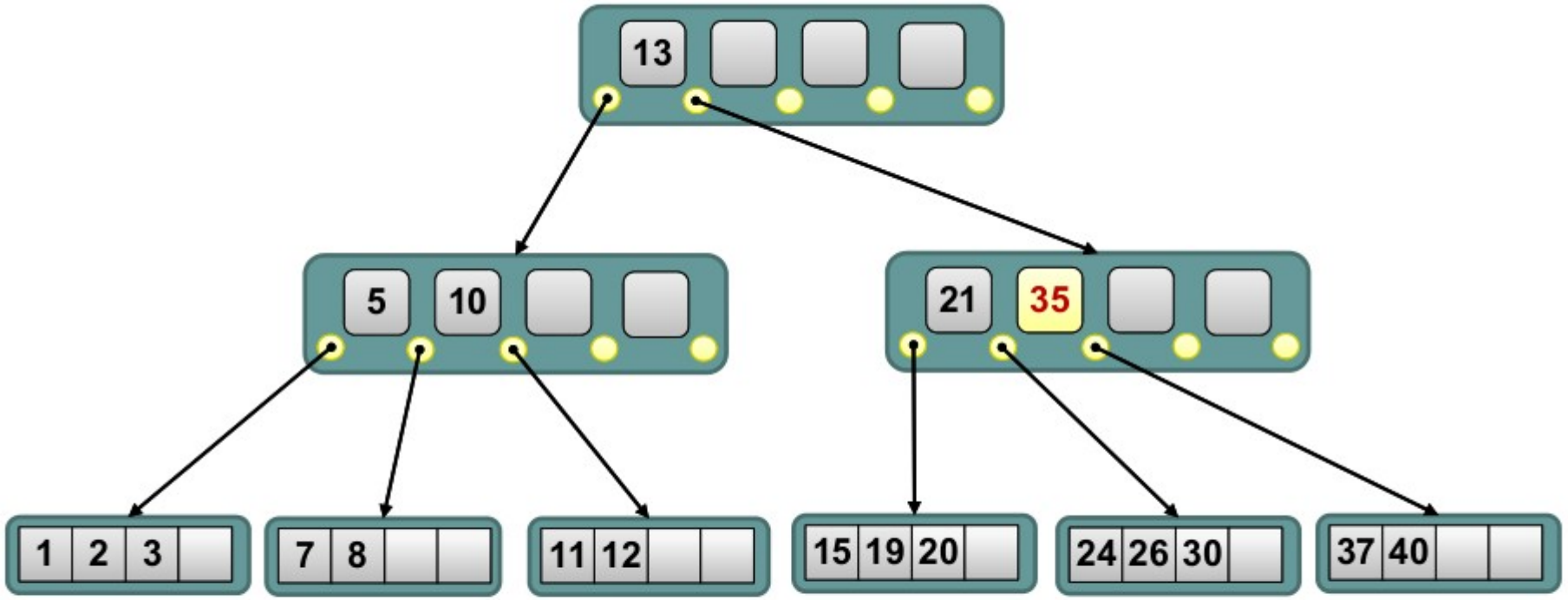
Inserción Por División de Nodos (11)



- Como no existe padre, se crea un nuevo nodo raiz que contiene únicamente esa clave

Arboles B

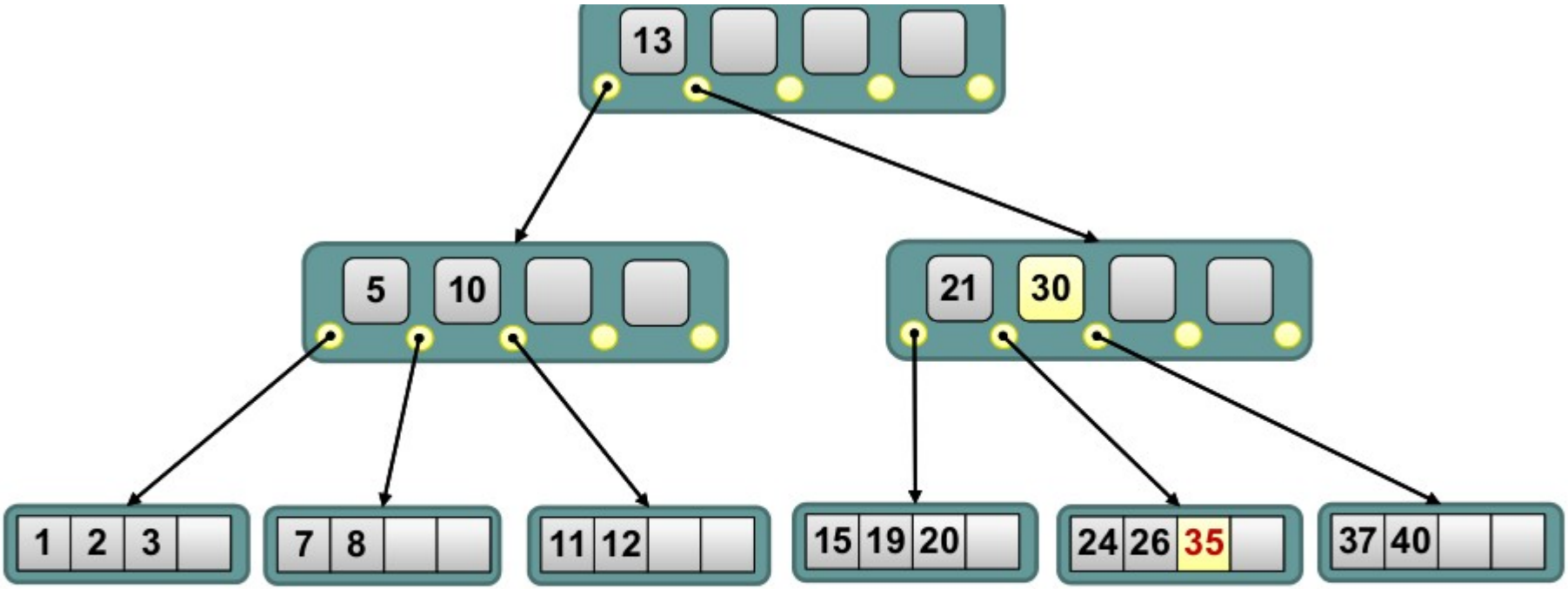
Borrado sin Reestructuración



- Borrado de la clave 35. Se busca el nodo donde está el elemento.

Arboles B

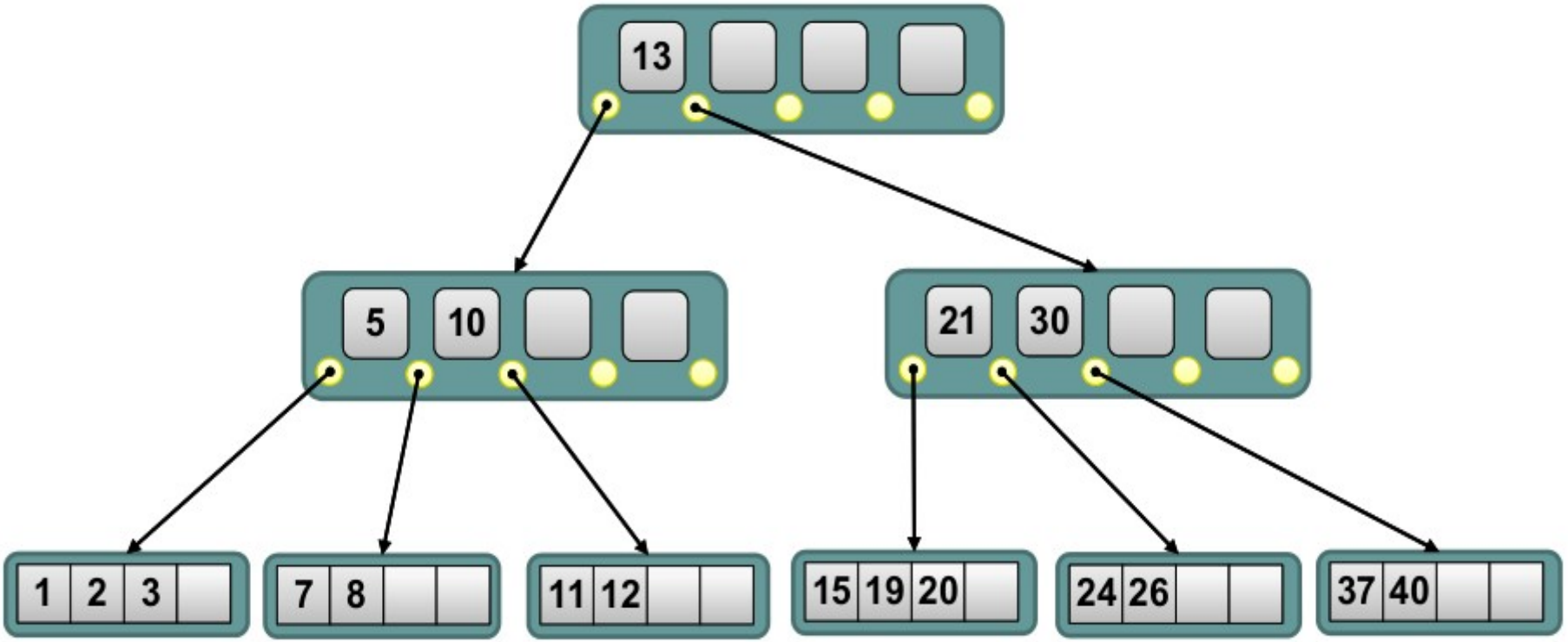
Borrado sin Restructuración



- Es un nodo interno. Se intercambia con el máximo de su hijo izquierdo y se pasa a borrar esa clave.

Arboles B

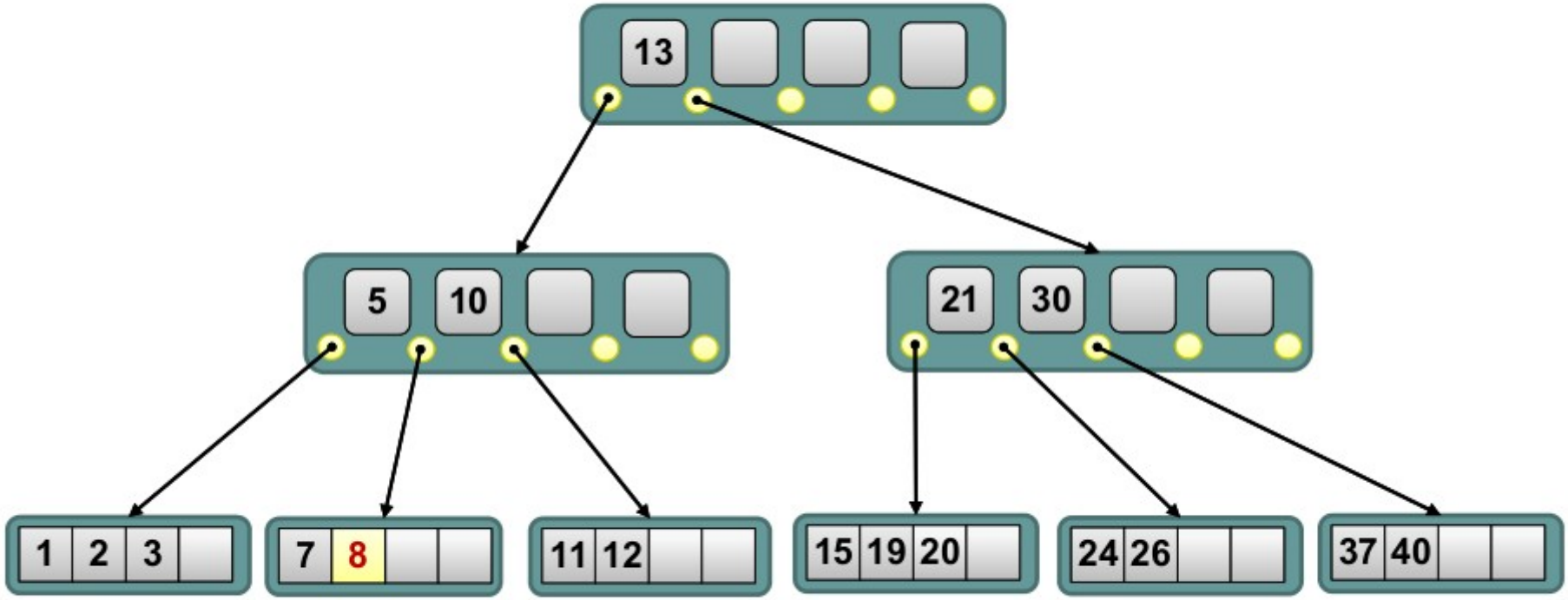
Borrado sin Reestructuración



- Se borra el elemento.

Arboles B

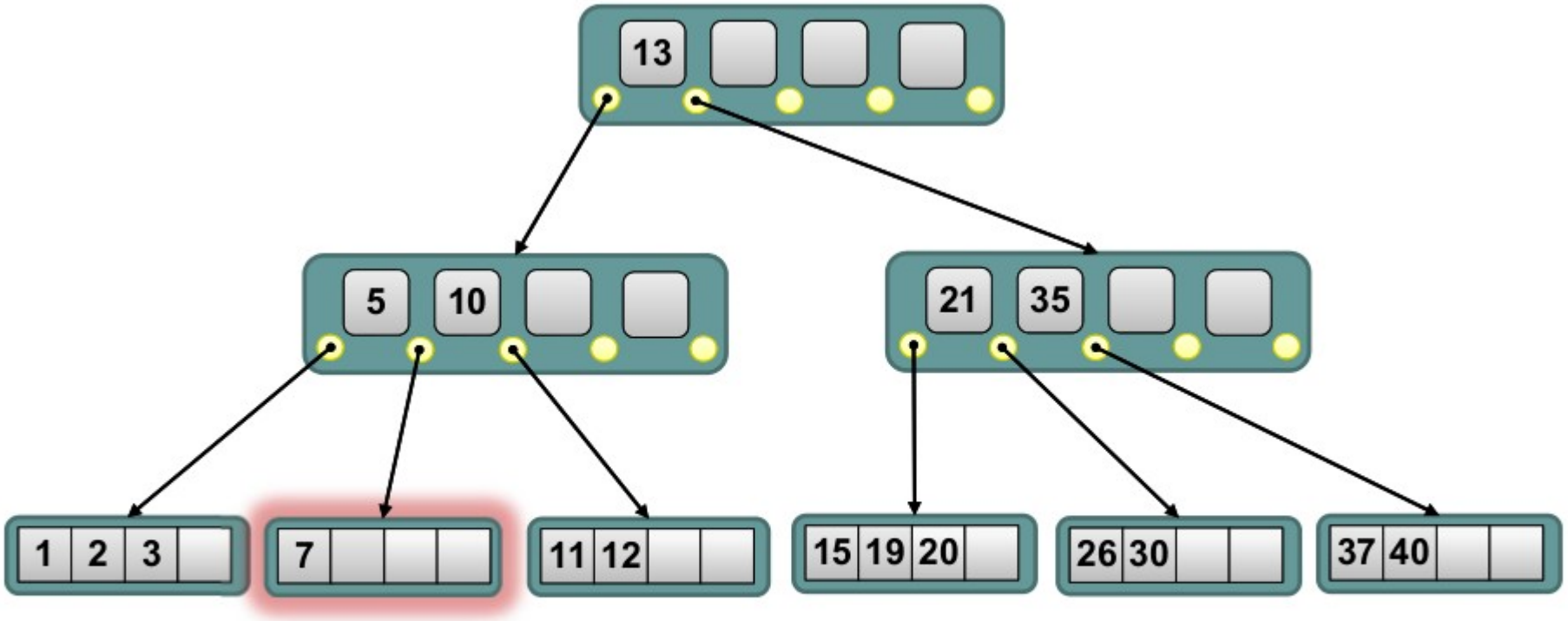
Borrado Por Transferencia:



- Borrado de la clave 8. Se busca el nodo.

Arboles B

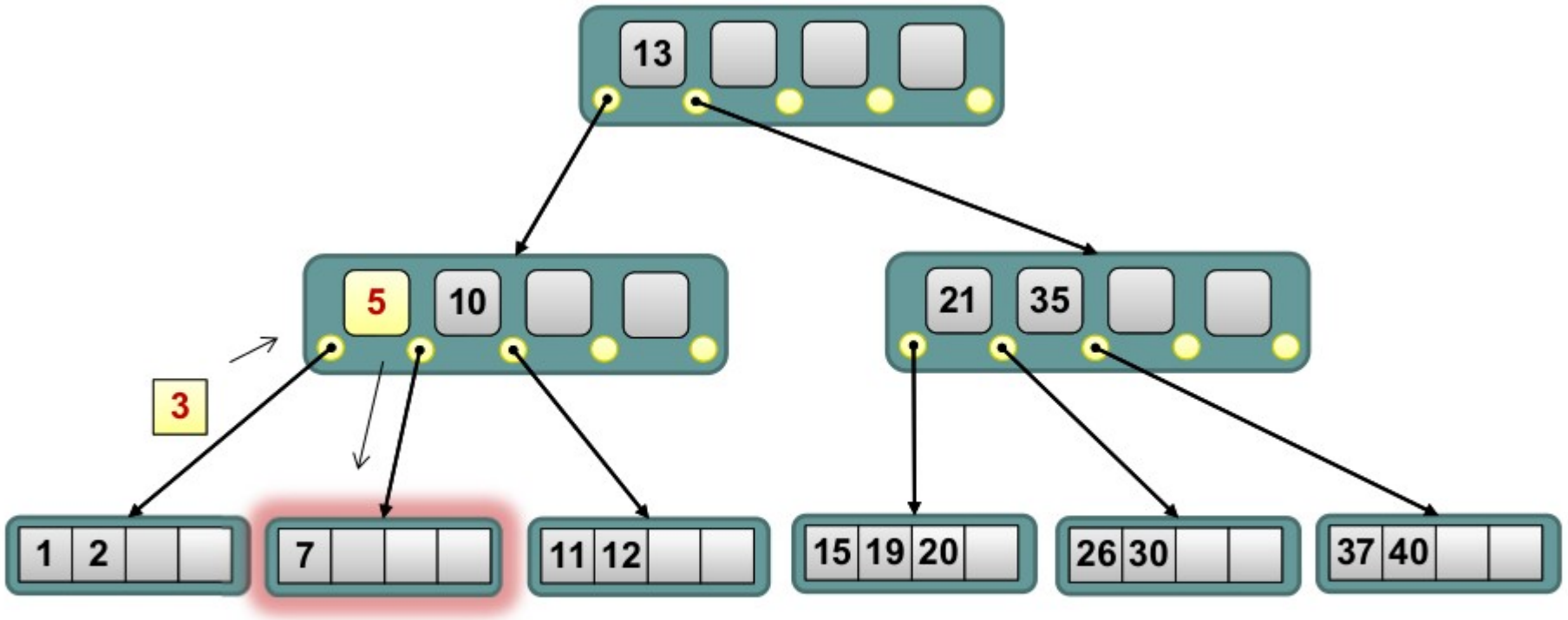
Borrado Por Transferencia:



- Se borra la clave. El nodo pasa a tener menos claves que las permitidas (2).

Arboles B

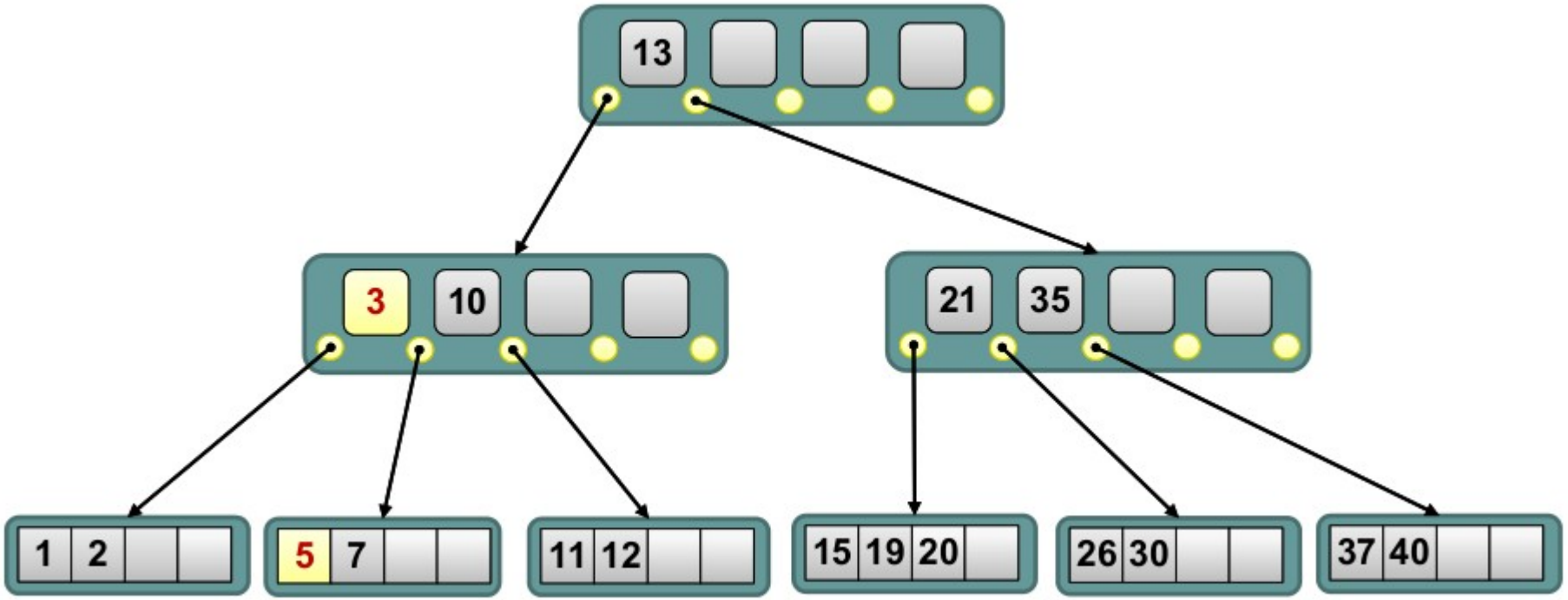
Borrado Por Transferencia:



- Se comprueba el hermano con más claves (el izquierdo). Se transfiere su última clave al padre y la del padre al nodo.

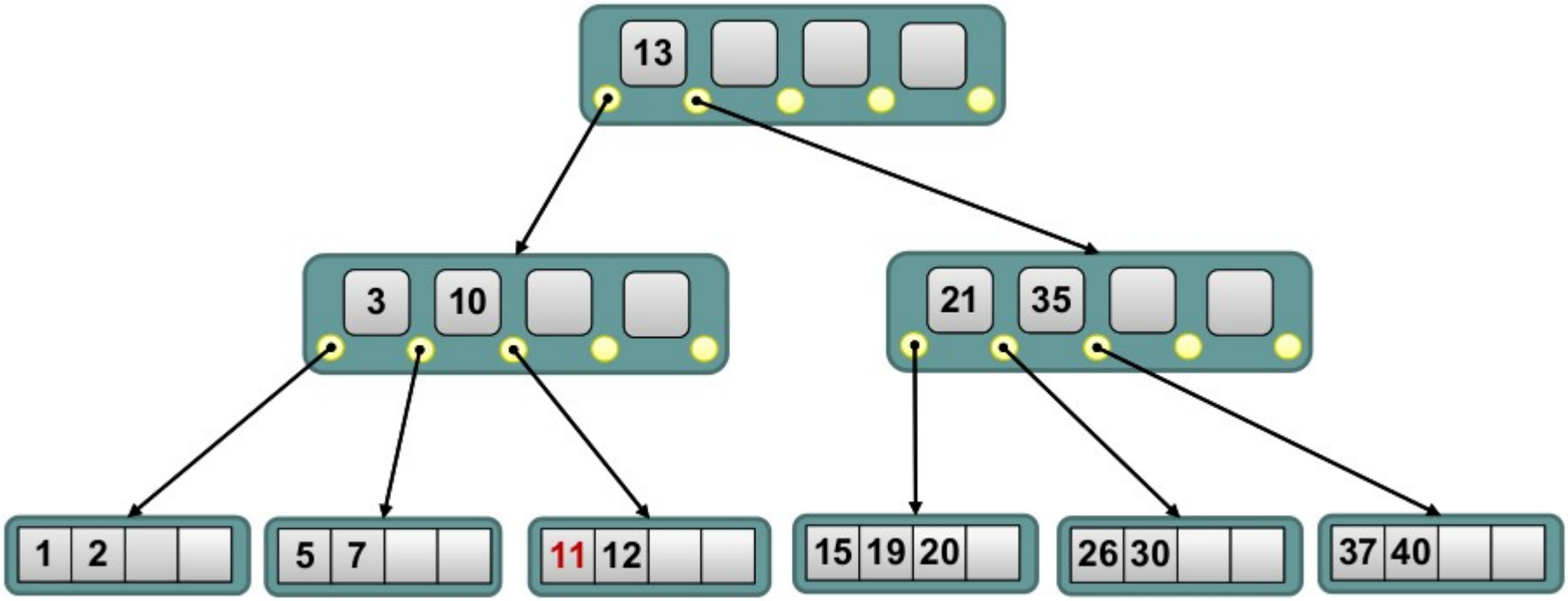
Arboles B

Borrado Por Transferencia:



Arboles B

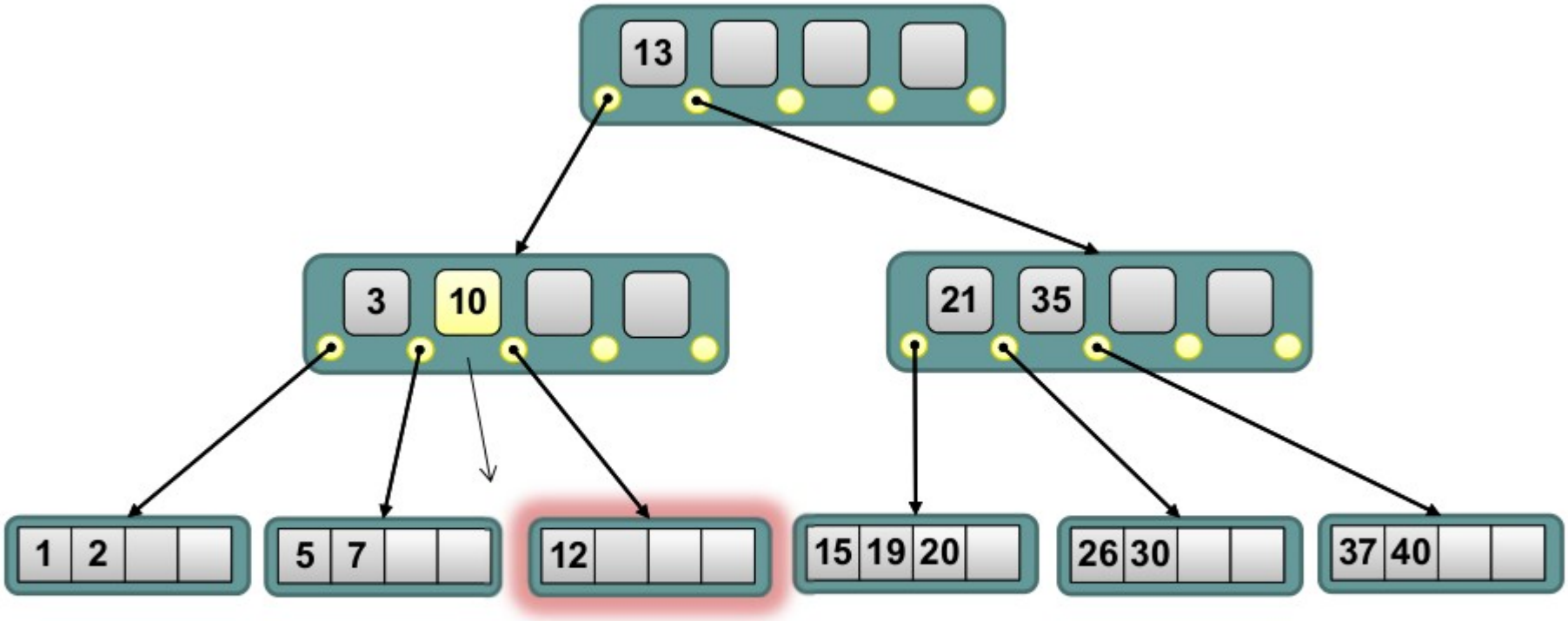
Borrado Por Fusión:



- Borrado del elemento con clave 11. Se busca el nodo.

Arboles B

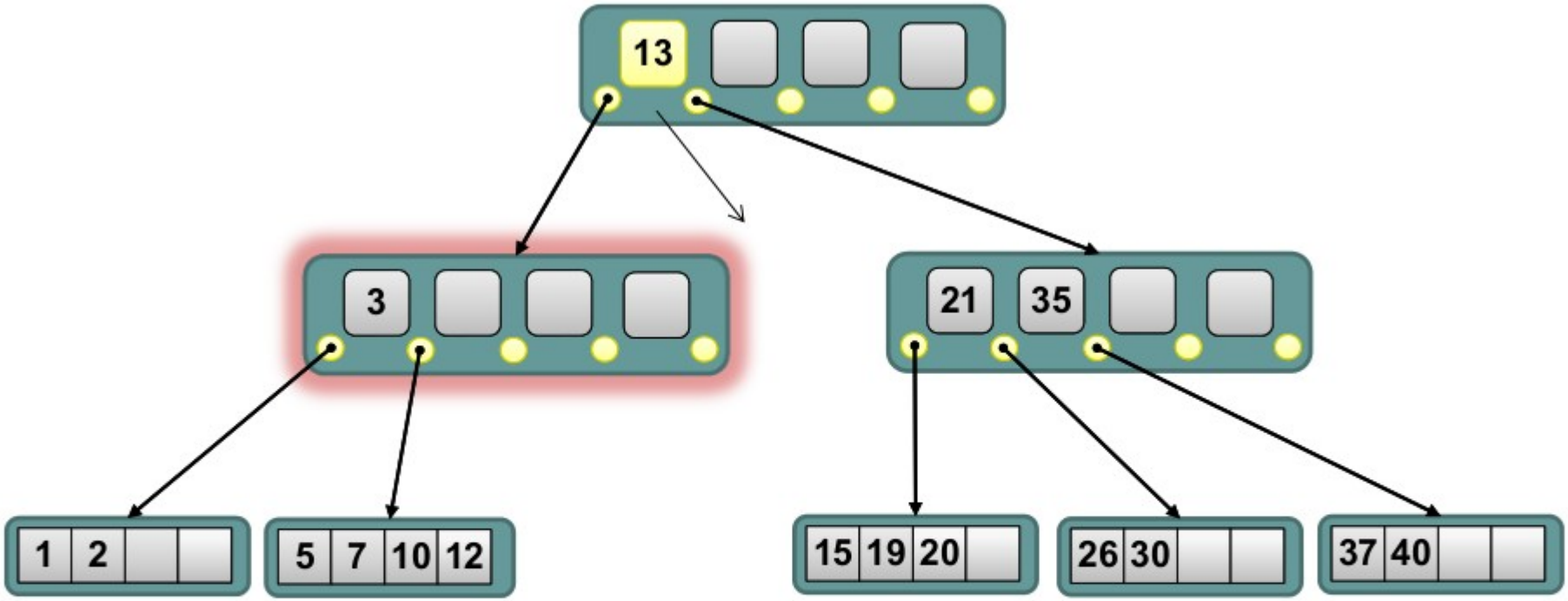
Borrado Por Fusión:



- Al borrar la clave pasa a tener menos claves que las permitidas. Su único hermano (izquierdo) no puede transferir claves.

Arboles B

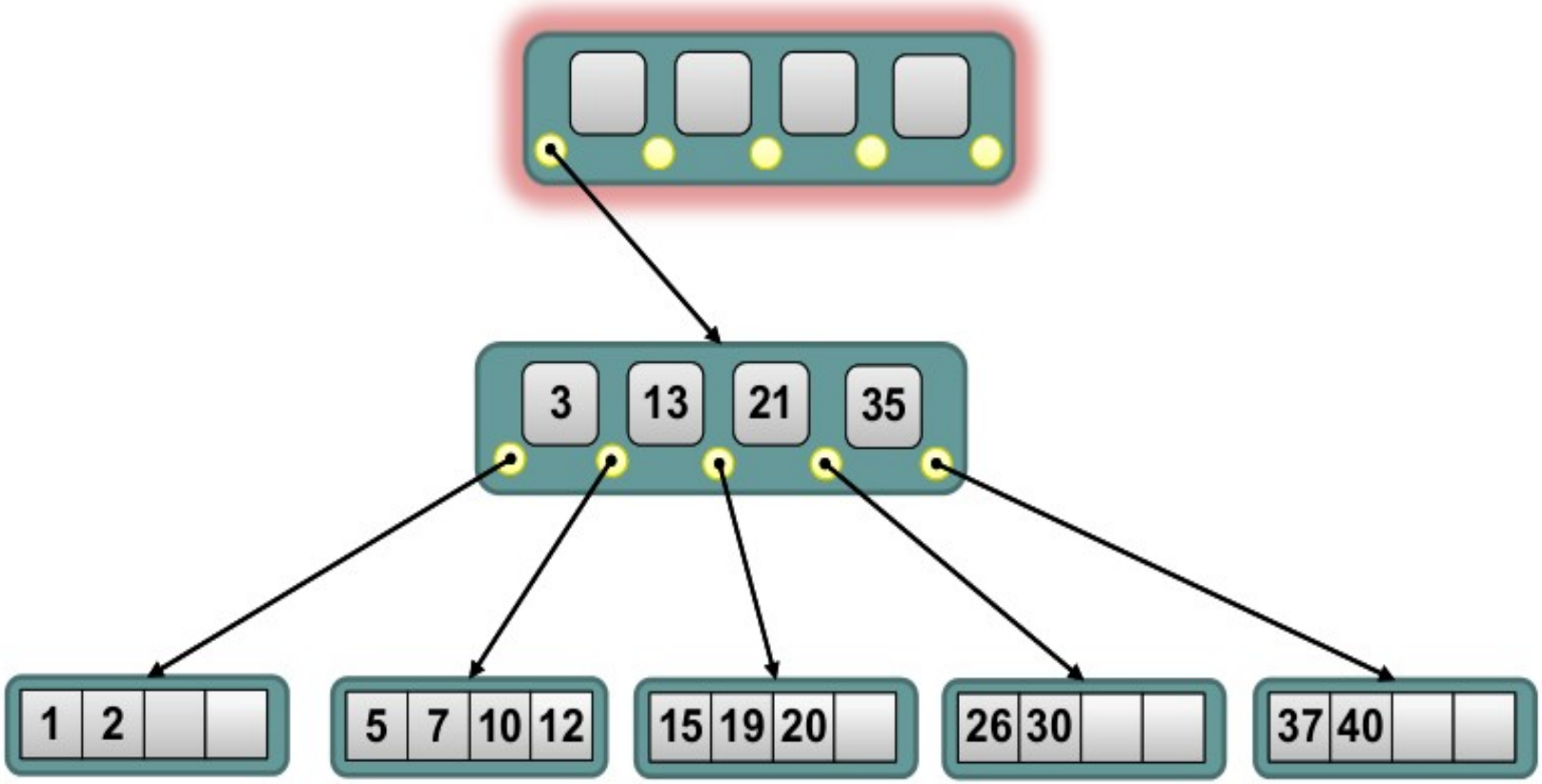
Borrado Por Fusión:



- Se fusionan el nodo con su hermano izquierdo, tomando una clave extra del padre. El padre pasa a tener una sola clave, y su hermano derecho no puede transferir.

Arboles B

Borrado Por Fusión:

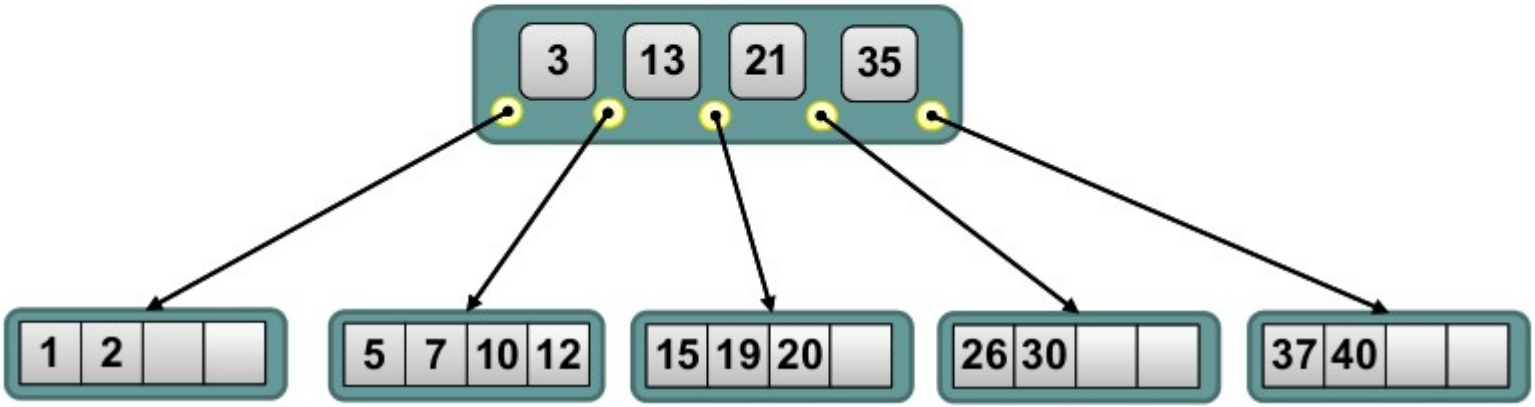


- Se fusionan los nodos, tomando la única clave del raiz, que queda vacío.

Arboles B

Borrado Por Fusión:

- Se elimina el nodo raiz.



Arboles B

Conclusiones:

Los arboles B surgen con la necesidad de mantener índices Externos para acceso a bases de datos.

La letra B no significa binario.

Los arboles B nunca son binarios.

Tampoco, son arboles de búsqueda, se denominan B-trees.

Tampoco es porque sean balanceados, ya que suelen no serlo.

Arboles B

Usos de los Arboles B

- **Los árboles B y sus variantes se usan en:**
  - **Gestores de Bases de Datos.**
  - **Sistemas de Ficheros:** NTFS (Windows), HFS+ (Apple), btrfs, Ext4 (Linux)
- **Variantes principales:**
  - **Árboles con prerecorrido**: Antes de insertar se realiza una búsqueda que divide todos los nodos llenos. El número máximo de claves es 2d+1.
  - **Árboles B+**: Sólo las hojas contienen elementos, los nodos internos contienen claves para dirigir la búsqueda (esas claves se encuentran también en los nodos hoja). Los nodos hoja forman una lista doblemente enlazada.
  - **Árboles B***: El número mínimo de claves es 2/3 de la capacidad. Se fusionan 3 nodos en 2, y se dividen 2 nodos en 3.

Arboles B*

Arboles B* Knuth 1973

En 1973,Knuth propone nuevas reglas para mantener los B-árboles
para eitar la división. (Ahorra espacion)
sino que se realizan divisiones de dos nodos completos a tres
de forma que los nodos resultantes tienen dos tercios del total.

1- Cada nodo tiene un máximo de m descendientes.
2- Cada nodo excepto el raíz tiene al menos (2m-1)/3 hijos.
3- La raíz tiene al menos dos descendientes (a menos de que sea hoja).
4- Una hoja contiene al menos E[(2m-1)/3] claves.
5- Todas las hojas aparecen en el mismo nivel.
6- Un nodo que no sea hoja con k descendientes contiene k-1 llaves.
7- Un nodo hoja contiene por lo menos E[(2m-1)/3] llaves, y no más de m-1.

Arboles B+

Arboles B+ Se Utiliza en :

- **Archivos secuenciales indexados**
  - Permiten una mejor recorrida por algún tipo de orden
  - Indexado (ordenado por una llave)
  - Secuencial (acceder secuencialmente al archivo, devolviendo los registros en orden de llave)
  - Hasta ahora métodos disjuntos, se opta:
    - rápida recuperación (Arbol)
    - Recuperación ordenada (secuencial)
  - Debemos encontrar una solución que agrupe ambos casos

Arboles B+

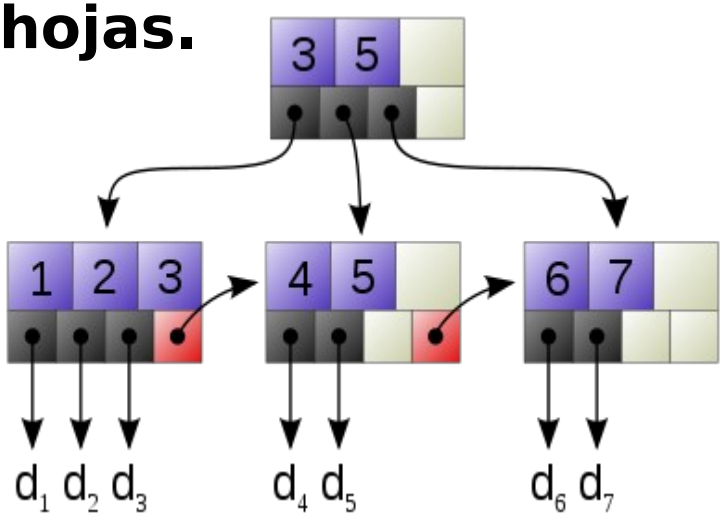
Arboles B+:

Es un índice, multinivel, dinámico, con un límite máximo y mínimo en el número de claves por nodo.

Un árbol B+ es una variación de un árbol B.

Toda información se guarda en las hojas.

Los nodos internos sólo contienen claves y punteros.



Todas las hojas se encuentran en el mismo nivel, el mas bajo.

Los nodos hoja se encuentran unidos entre sí como una lista enlazada.

Esto permite la recuperación en rango mediante búsqueda secuencial

Arboles B+

Arboles B+ Justificación:

- **Archivos secuenciales indexados**
  - Permiten una mejor recorrida por algún tipo de orden
  - Indexado (ordenado por una llave)
  - Secuencial (acceder secuencialmente al archivo, devolviendo los registros en orden de llave)
  - Hasta ahora métodos disjuntos, se opta:
    - rápida recuperación (Arbol)
    - Recuperación ordenada (secuencial)
  - Debemos encontrar una solución que agrupe ambos casos

Arboles B+

Arboles B+ Propiedades:

Los árboles B+ constituyen otra mejora sobre los árboles B.

Conservan la propiedad de acceso aleatorio rápido (de los árboles B) permitiendo además un recorrido secuencial rápido

Este árbol esta compuesto por dos partes:

Índice: nodos interiores

Secuencia: paginas hojas enlazadas secuencialmente en las que se repiten las claves anteriores

Arboles B+

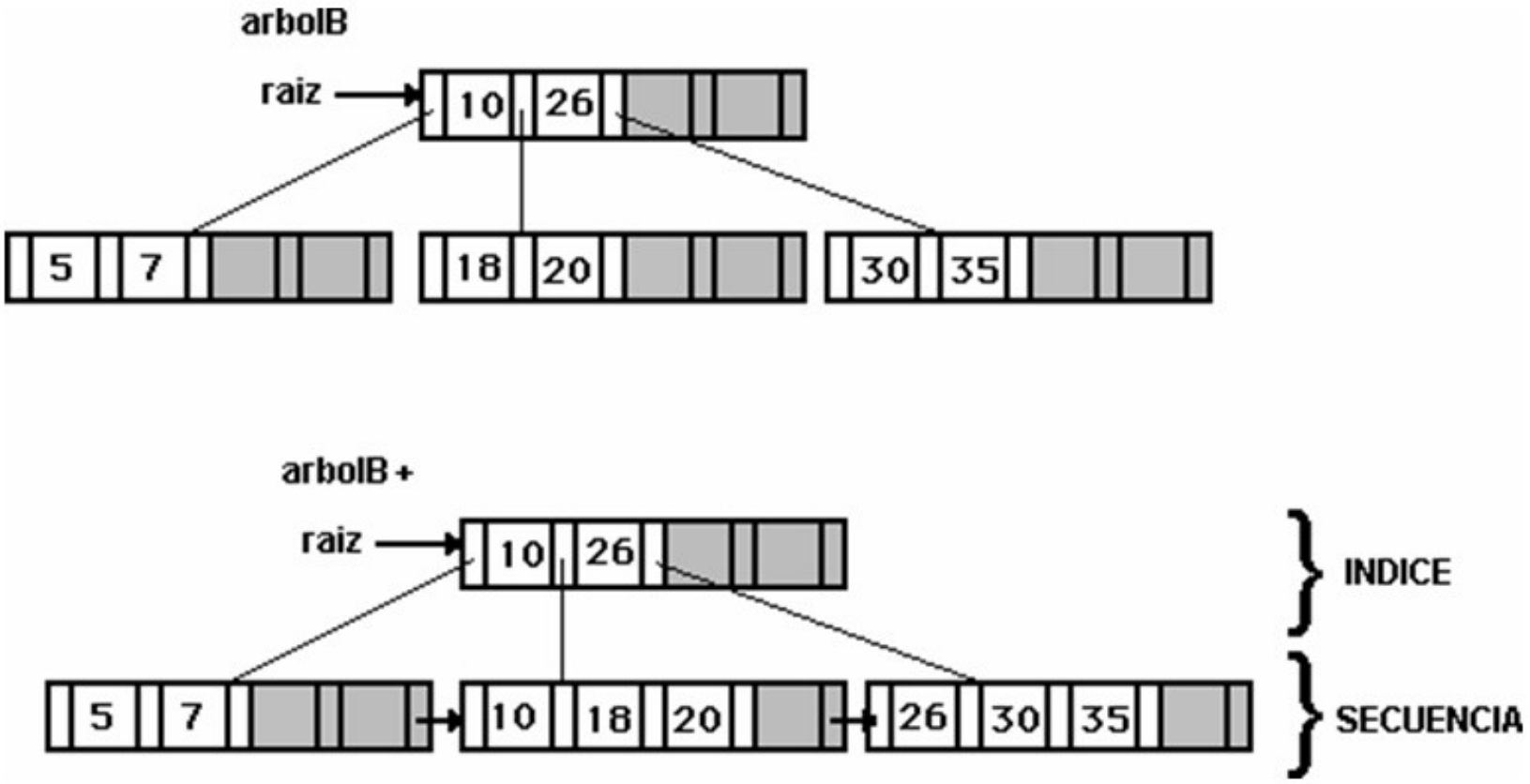
Arboles B+ Propiedades: Continua.

- Todas las claves se encuentran en las Hojas (duplicándose en la raiz y nodos interiores, aquellas necesarias para definir los caminos de búsqueda)
- Las hojas están vinculadas, obteniéndose de ese modo una trayectoria secuencial para recorrer las claves del árbol
- Ocupan más espacio que los árboles B (ya que algunas claves se encuentran más de una vez en el árbol)
- Las claves de la página raiz e interiores se utilizan únicamente como índice (para búsqueda)

Arboles B+

Árboles B+

- Ejemplo



Arboles B+

- **Propiedades B+**
  - Cada página tiene máximo M descendientes
  - Cada página, menos la raíz y las hojas, tienen entre [M/2] y M hijos
  - La raíz tiene al menos dos descendientes (o ninguno)
  - Todas las hojas aparecen en igual nivel
  - Una página que no sea hoja si tiene K descendientes contiene K-1 llaves
  - Los nodos terminales representan un conjunto de datos y son linkeados juntos.
- Los nodos no terminales no tienen datos sino punteros a los datos.

Arboles B+

Arboles B+ Operaciones:

Búsqueda: la búsqueda no debe detenerse cuando se encuentre la clave en la raiz o nodo interior, sino que debe seguir en la pág. Apuntada por la rama derecha de dicha clave

Inserción:

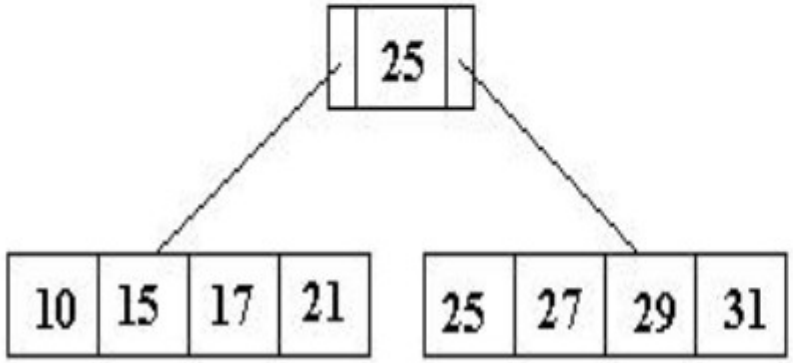
- El proceso de inserción es similar a de los árboles B, excepto cuando se desea insertar una clave en donde la página se encuentra llena.

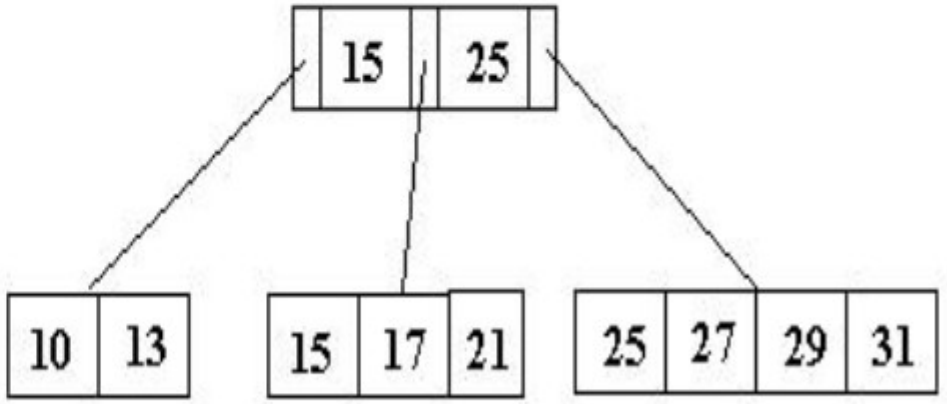
- Cuando se inserta una nueva clave en una nueva página llena, esta se divide también en otras dos, pero ahora la primera contendrá m/2 claves y la segunda 1+m/2 y lo que subirá al padre será una copia de la clave central

Arboles B+

Arboles B+ Operaciones:

- Inserción clave 13





Arboles B+

Arboles B+ Operaciones:

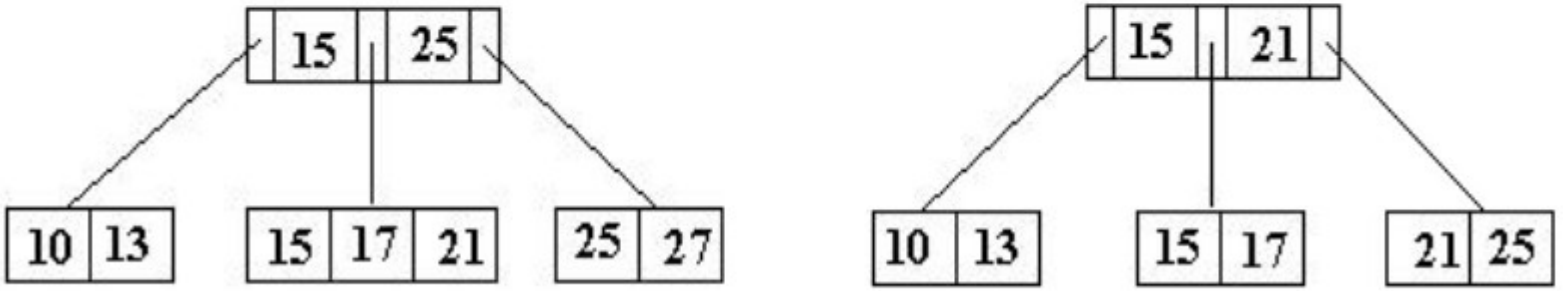
Eliminación

- Este proceso es más simple que en los árboles B, porque todas las claves se encuentran en las páginas hojas.
- Si al eliminar la clave (siempre en una hoja) la cant. de claves >= m/2, el proceso ha terminado
  - Las claves de la raiz o nodos internos no se modifican pues siguen siendo un separador válido entre las claves de las páginas descendientes
- Si al eliminar la clave (siempre en una hoja) la cant. de claves < m/2, es necesario una fusión y redistribución de las mismas, en las hojas y el índice

Arboles B+

Arboles B+ Operaciones:

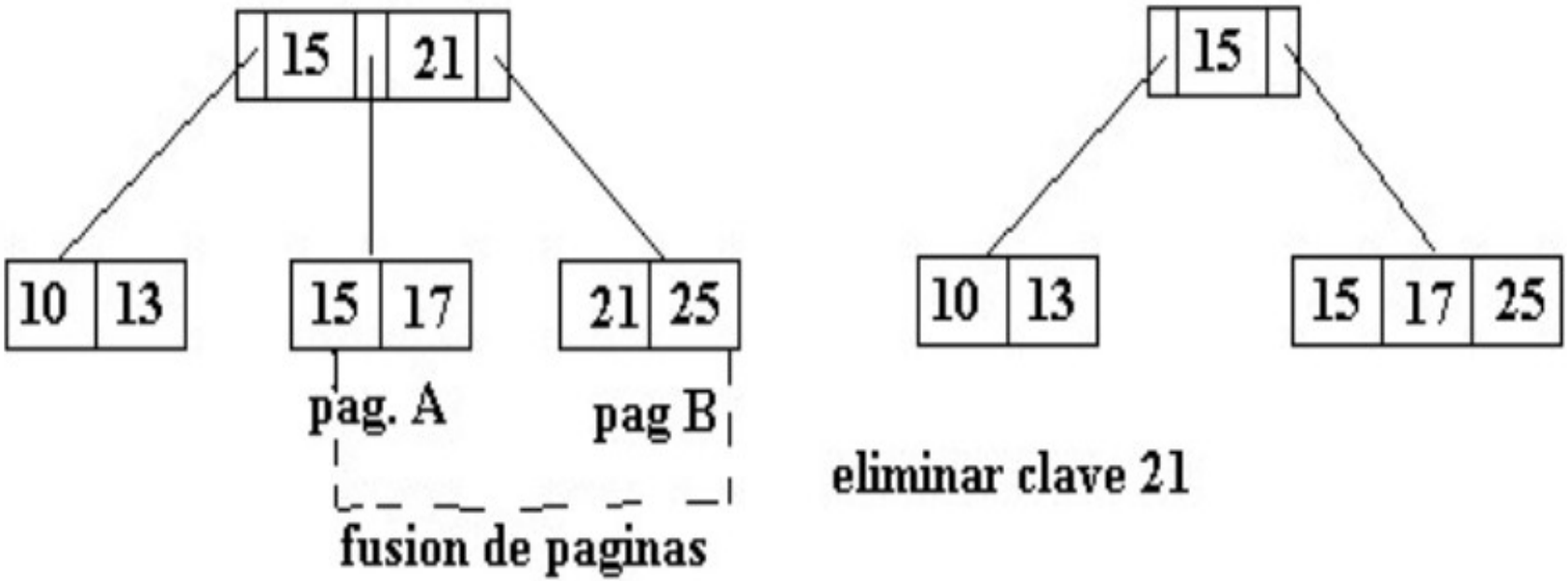
eliminar clave 25



eliminar clave 27

Arboles B+

Arboles B+ Operaciones:



Arboles B+

Arboles B+ y B:

- **Comparaciones**

	Arbol B	Arbol B+
Ubicación de datos	Nodos (cualquiera)	Nodo terminal
Tiempo de búsqueda	=	=
Procesamiento secuencial	Lento (complejo)	Rápido (con punteros)

Organización de Archivos

Organización de Archivos Indexados

Archivos Indexados:

Pueden verse como un conjunto de registros, los que pueden accederse mediante una clave.

- *Área Principal*
- *Área de Índices*

Área Principal:

En esta área se almacenan los registros, con los datos, al momento de crear el archivo.

Área de Índices

Cada registro del área de índices consta de 2 campos:

- *Clave de los Registros*
- *Puntero al Registro en el área principal*

Organización de Archivos

Organización de Archivos Indexados

Archivos Indexados:

Pueden verse como un conjunto de registros, los que pueden accederse mediante una clave.

- *Área Principal*
- *Área de Índices*

Área Principal:

En esta área se almacenan los registros, con los datos, al momento de crear el archivo.

Área de Índices

Cada registro del área de índices consta de 2 campos:

- *Clave de los Registros*
- *Puntero al Registro en el área principal*

Organización de Archivos

Organización de Archivos Indexados

Archivos Indexados:

Área Principal			Área Índices	
Registro	Dato		Clave	Puntero
0	1		1	0
1	5		5	1
2	20		20	2
3	10		10	3
4	4		4	4
5	21		21	5
6	22		22	6

Anexar siempre genera un nuevo Registro, en ambas áreas.

Eliminar Basta con poner e puntero a nulo en el área Índices

Insertar puede hacerse como Anexar o bien Aprovechar una posición eliminada.

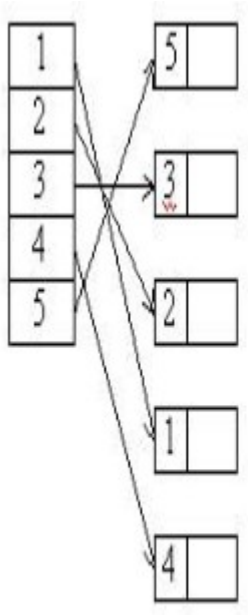
Organización de Archivos

Organización de Archivos Secuenciales Indexados

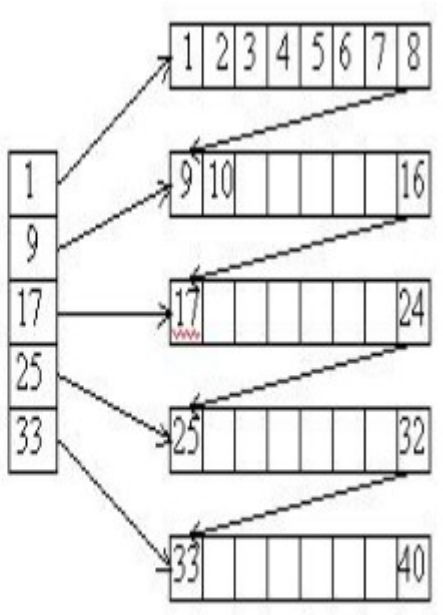
Archivos Secuenciales Indexados:

El índice se puede organizar de diversas formas, las más típicas son: secuencial, multinivel y árbol.

Podremos procesar un fichero de forma secuencial o de forma directa según la clave de indexación, y esto independientemente de como esté organizado el fichero por sí mismo



Índice Total o Denso



Índice Escaso o No Denso

Se pueden tener tantos índices como se quiera variando la clave que se emplee.
El índice está formado por registros que contienen:
Clave de organización ---> Puntero(s) al fichero de datos

Organización Indexada

Concepto de índice. Técnicas de indexación de ficheros.

Objetivo: utilizar como *clave* una *abstracción* de la información *del dato*

⇩

búsqueda en directorio ≡ representar explícitamente la relación entre *valores de la clave* y la *ubicación física del dato*

INDICE ≈ **VECTOR** (tabla) de registros

- absoluta
- relativa (NRR)

fichero indexado ≡ fichero cuya *organización* está basada en un índice (o más)

☛ acceso al dato ⇒ *2 pasos* : **búsqueda en índice + acceso al dato**

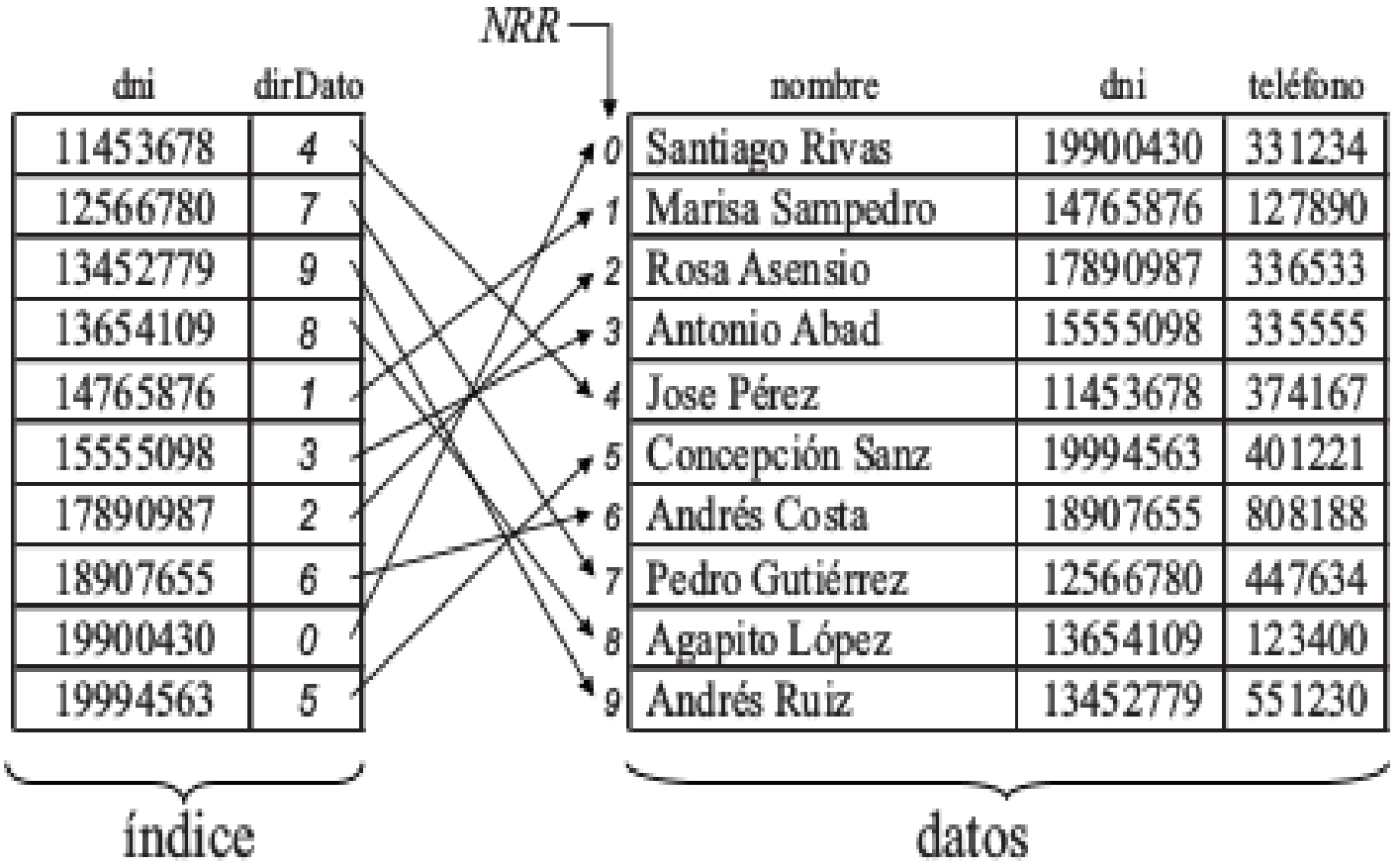
☛ suelen implementarse en base a dos ficheros: **f. índice + f. datos**

→ **pueden estar integrados en un único fichero** (reg. diferentes)

Organización Indexada

Concepto de índice: ejemplo

☛ *Información de personas. Como clave se usará el DNI* (identifica una persona)



Organización Indexada

Concepto de índice: ventajas e inconvenientes.

ventajas:

- menor tamaño del índice ⇒ *mayor eficiencia búsqueda* (↑ factor de bloque)
  → **podría caber en memoria**
- registros del índice de long. fija (normalmente) ⇒ *acceso + simple y eficiente*
  (p.e. búsqueda binaria)
- simplifica la utilización de registros de longitud variable (representados como tales)
- permite acceso directo (eficiente) a los datos utilizando cualquier campo clave, incluso simultáneamente (varios índices)
- permite mantener una (o varias) *ordenación* lógica de los datos *sin moverlos*
  → *el mantenimiento del orden es muy eficiente*

inconveniente:

- operaciones de *procesamiento secuencial* (afectan a muchos datos) son *más lentas*

Organización Indexada

Concepto de índice: índice denso y no-denso

una entrada de índice por cada dato ≡ ***índice denso***

→ el índice puede ser excesivamente grande

Solución: elegir un *subconjunto de claves* ⇒ ordenación (parcial) de los datos por la clave

índice no-denso

dni	dirDato
11453678	0
13654109	3
19900430	8

Se puede aprovechar la organización física en bloques

NRR	nombre	dni	teléfono
0	Jose Pérez	11453678	374167
1	Pedro Gutiérrez	12566780	447634
2	Andrés Ruiz	13452779	551230
3	Agapito López	13654109	123400
4	Marisa Sampedro	14765876	127890
5	Antonio Abad	15555098	335555
6	Rosa Asensio	17890987	336533
7	Andrés Costa	18907655	808188
8	Santiago Rivas	19900430	331234
9	Concepción Sanz	19994563	401221

Ordenados por DNI

Organización Indexada

Concepto de índice: tipos de índices

la extensión de la idea de indexación → diferentes tipos de índices

- índice ***primario*** ≡ se utiliza la *clave (primaria)* del dato
- índice ***de agrupamiento*** ≡ se utiliza un *campo de ordenación* del dato
- índice ***secundario*** ≡ se utiliza la *cualquier otro campo* (clave o no)

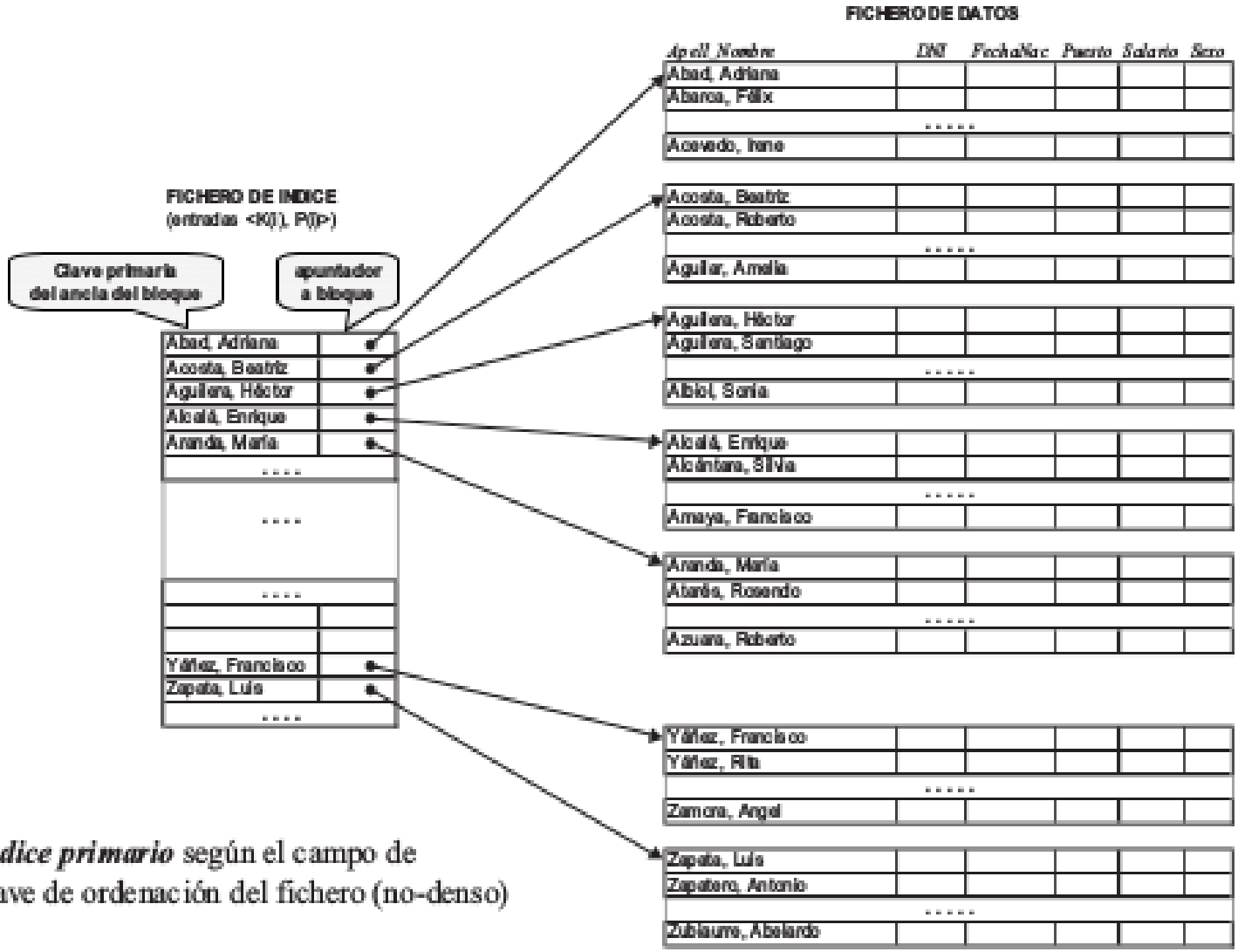
Si todavía es muy grande el índice → nuevo índice sobre el índice

índice ***multinivel***

aumenta la eficiencia de la búsqueda, pero complica la gestión

Organización Indexada

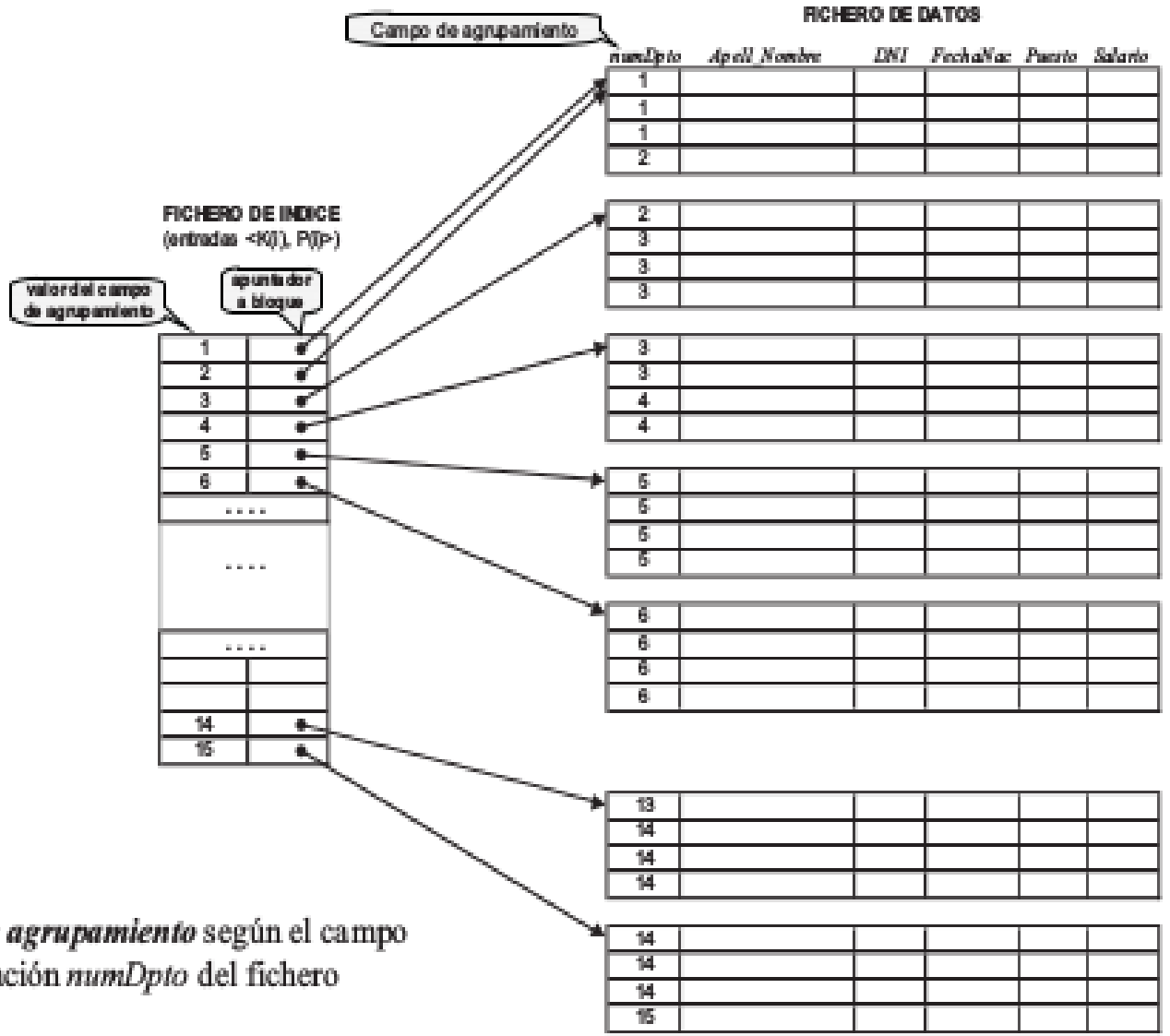
Índice primario según el campo de clave de ordenación del fichero (no-denso)



Indice primario según el campo de clave de ordenación del fichero (no-denso)

Organización Indexada

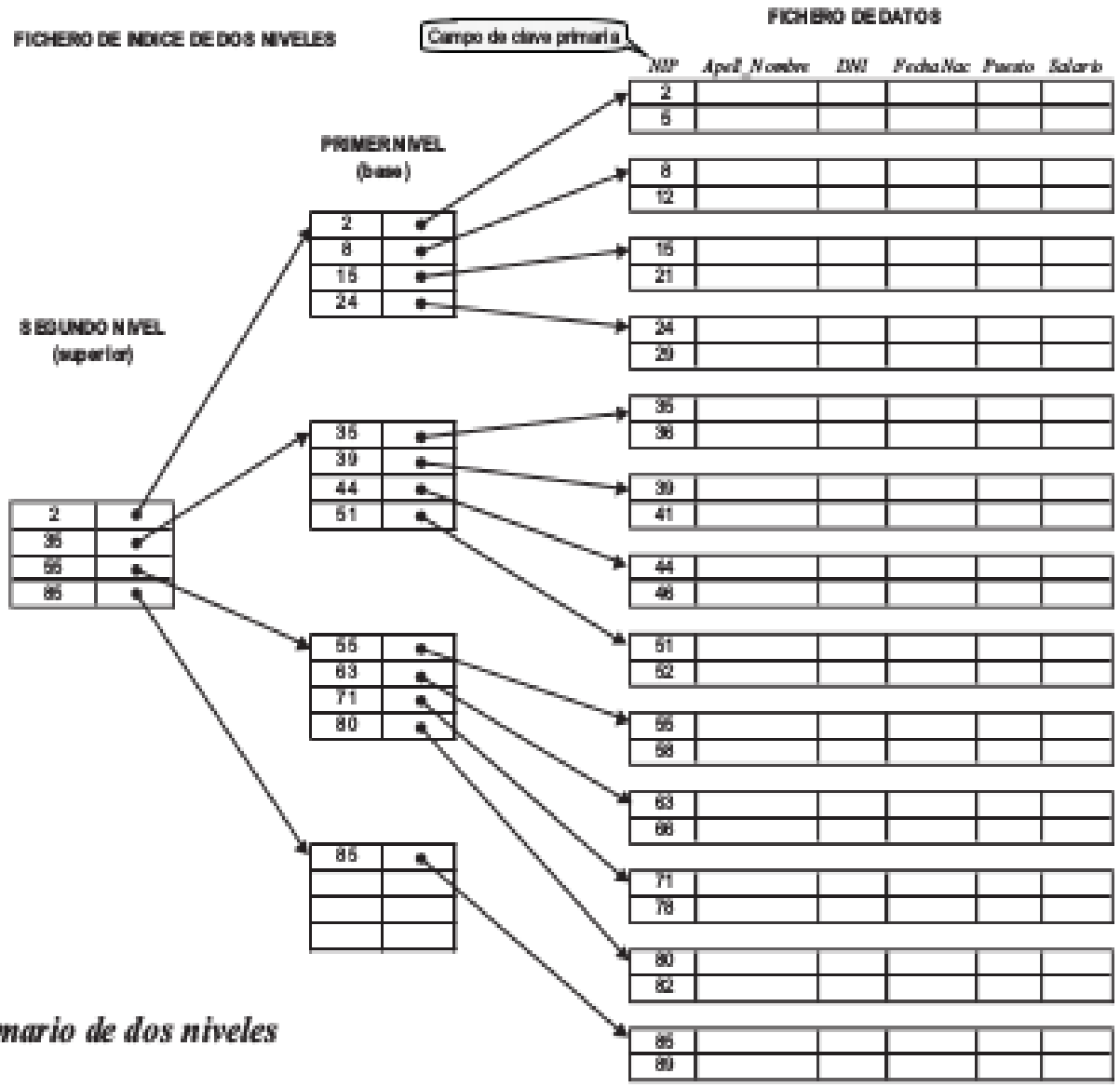
Indice de Agrupamiento según el campo de ordenación numDpto del fichero:



Indice de agrupamiento según el campo de ordenación *numDpto* del fichero

Organización Indexada

Índice primario de dos niveles



Indice primario de dos niveles

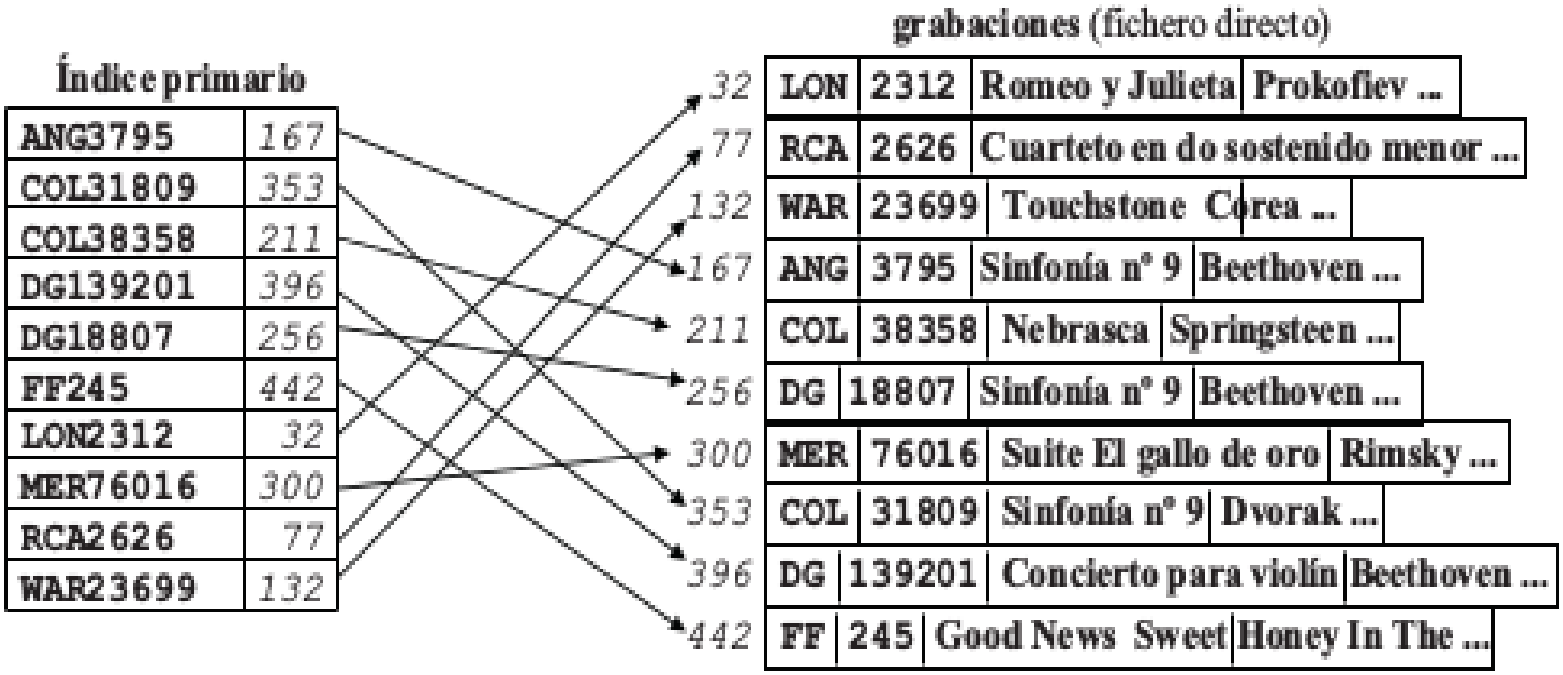
Organización Indexada

Organizaciones con varios índices secundarios: Ficheros invertidos.

Objetivo: acceso eficiente ("*directo*") a los datos utilizando diversos criterios

no tienen por qué ser información clave

ejemplo: información de grabaciones musicales



Arboles Lexicográficos

Lexicografía:

Es la disciplina aplicada al lenguaje que se ocupa de la elaboración y el análisis crítico de diccionarios.

Orden Lexicográfico:

Es conocido principalmente por su aplicación a cadenas de caracteres, por ejemplo en diccionarios o en la guía telefónica.

Una aplicación más general del orden lexicográfico es al comparar cadenas de caracteres

En 1959 cada uno por su lado Briandais y Fredking,
TRIE es una estructura de datos de tipo árbol que permite la recuperación de información (de ahí su nombre del inglés reTRIEval)

Arboles Lexicográficos

TRIE:

Por tanto la búsqueda en un trie se hace de forma similar a como se hacen las búsquedas en un diccionario:

Se empieza en la raíz del árbol.

Si el símbolo que estamos buscando es A entonces la búsqueda continúa en el subárbol asociado al símbolo A que cuelga de la raíz.

Se sigue de forma análoga hasta llegar al nodo hoja.

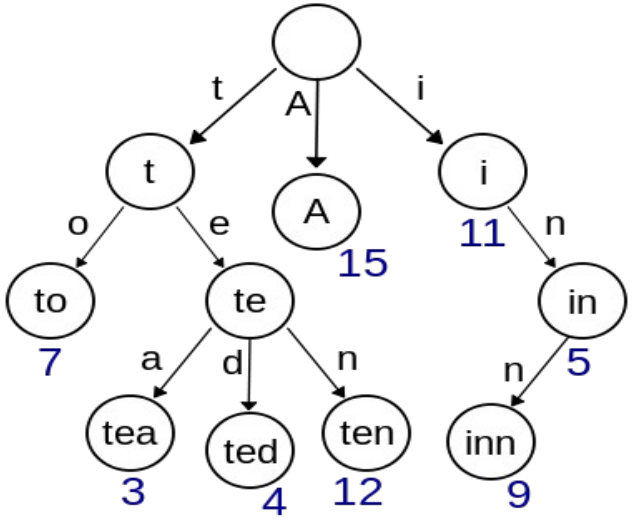
Entonces se compara la cadena asociada a el nodo hoja y si coincide con la cadena de búsqueda entonces la búsqueda ha terminado en éxito,

si no entonces el elemento no se encuentra en el árbol.

Arboles Lexicográficos

TRIE: Armado:

1) Insertar todas las claves.
2) obtener todas las claves mediante un recorrido en pre-orden
3) obtener un ordenamiento lexicográfico en orden ascendente un recorrido en post-orden.
4) Obtener un ordenamiento lexicográfico en orden descendente.



Aplicando algoritmos de búsqueda en profundidad en árboles.

.

Arboles Lexicográficos

Arbol Trie Función de Búsqueda

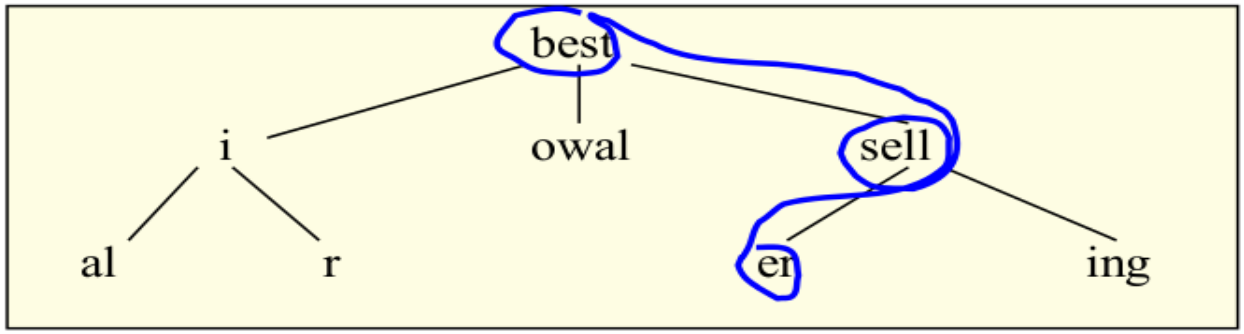
```
function find(node, key) {
  if (key is an empty string) {  # base case
    return is node terminal?
  } else {  # recursive case
    c = first character in key  # this works because key is not empty
    tail = key minus the first character
    child = node.children[c]
    if (child is null) {  # unable to recurse, although key is non-empty
      return false
    } else {
      return find(child, tail)
    }
  }
}
```

Arboles Lexicográficos

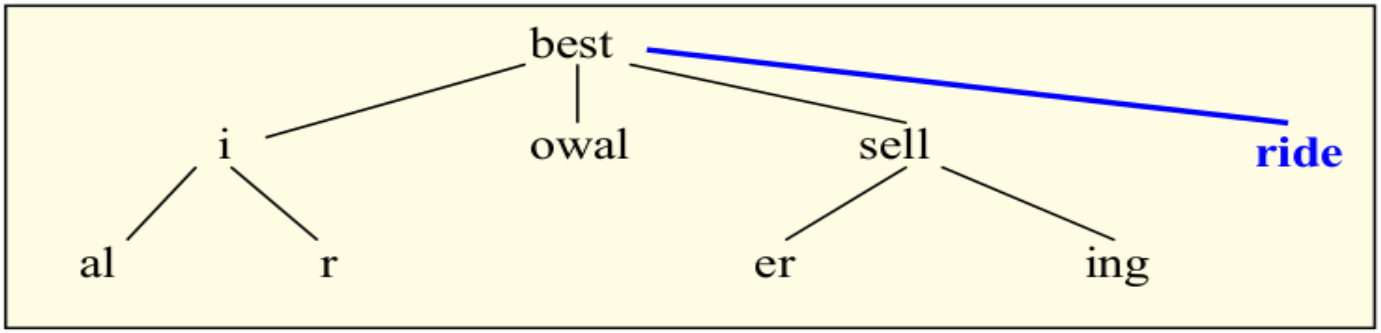
Utilidades del TRIE

Utilidad del trie:

- Soporta operaciones de búsqueda de palabras:



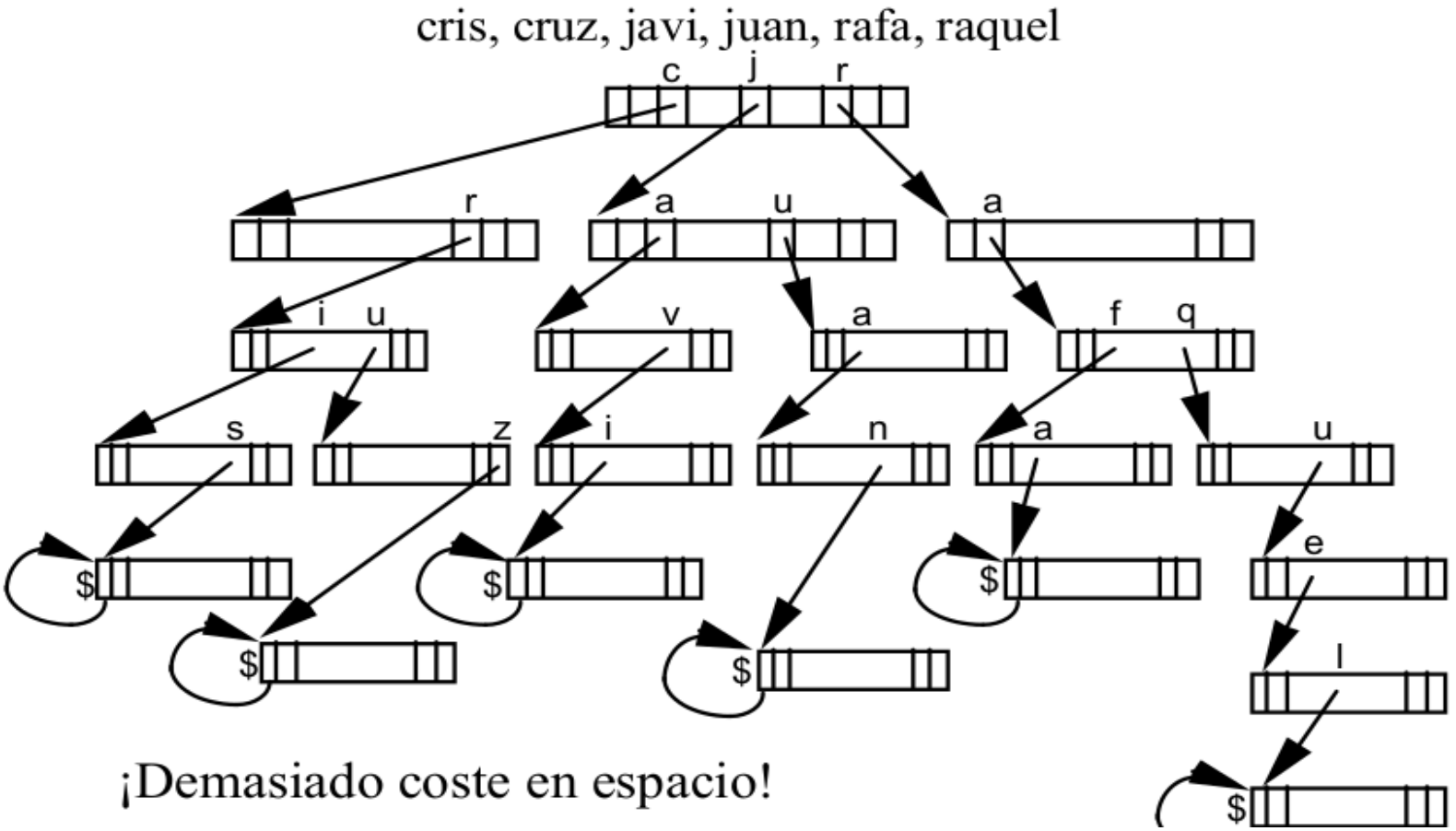
- También se pueden implementar inserciones y borrados ➔ TAD *diccionario*



Arboles Lexicográficos

Implementaciones del TRIE

– *Nodo-vector*: cada nodo es un vector de punteros para acceder a los subárboles directamente



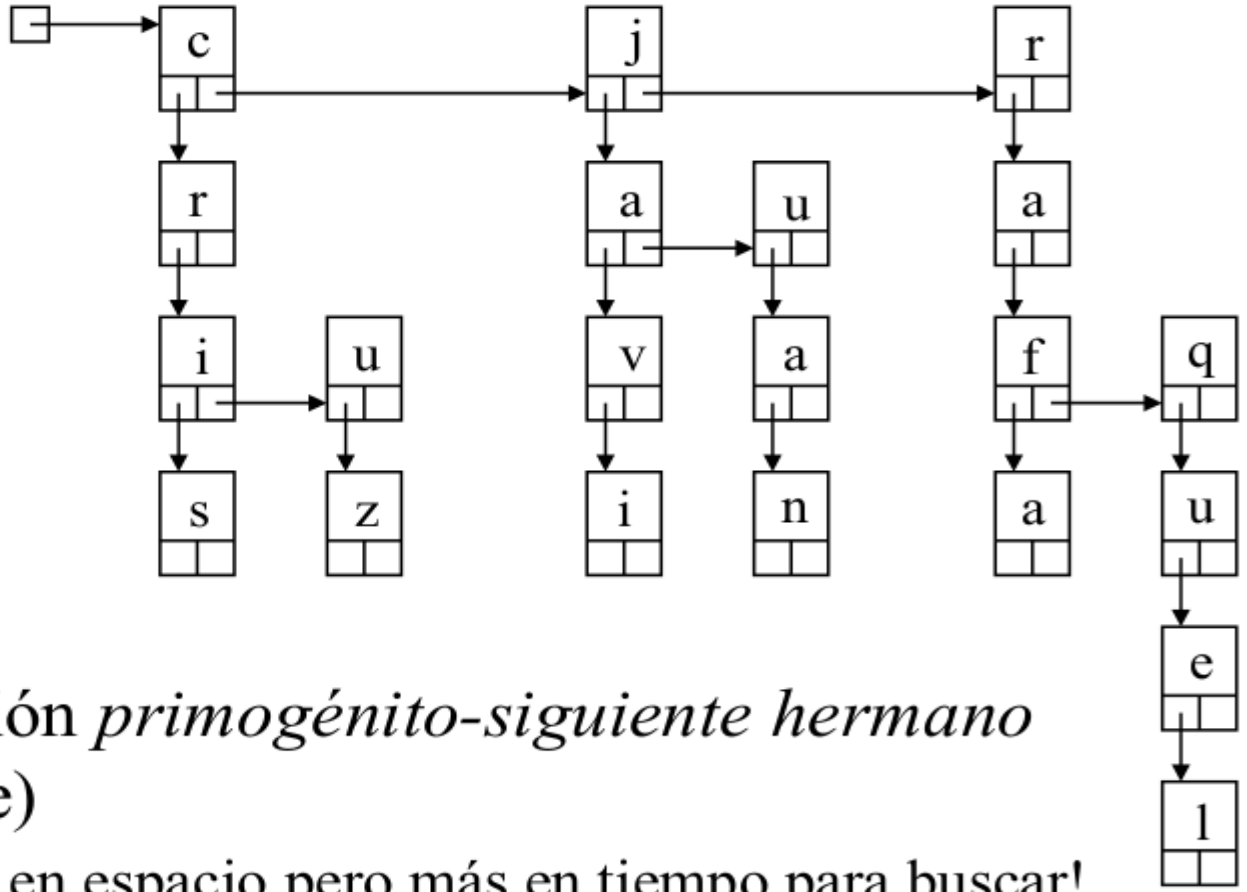
¡Demasiado coste en espacio!

Arboles Lexicográficos

Implementaciones del TRIE

– *Nodo-lista*: cada nodo es una lista enlazada por punteros que contiene las raíces de los subárboles

cris, cruz, javi, juan, rafa, raquel



(representación *primogénito-siguiente hermano* de un bosque)

¡Menos coste en espacio pero más en tiempo para buscar!

Árboles Patricia

Árbol Patricia

(Practical Algorithm To Retrieve Information Coded In Alphanumeric)

Es una mejora al Trie, por la cantidad de punteros null que podria llegar a tener el mismo.

El método de búsqueda en un Patricia difiere en dos puntos del utilizado en un trie:

Se deben testear los caracteres indicados en los nodos y no todos los caracteres de la cadena buscada.

La búsqueda siempre termina con la comparación completa de la secuencia buscada con la encontrada en la hoja.