Lección 21: Árboles B

- Motivación
- Árboles B
- Estructura y propiedades de un árbol B
- Estructura de un árbol B
- Búsqueda
- Inserción
- Borrado

Motivación

Supongamos que el catálogo de libros crece hasta un tamaño (10.000.000 de títulos) y que necesitamos hacer búsquedas por isbn, título y autor. El tamaño de estos tres índices en memoria sería:

- Índice primario: 10 (isbn) + 4 (posición) = 14 bytes
- Índice secundario título: 80 (título) + 10 (isbn) = 90 bytes
- Índice secundario autor: 40 (autor) + 10 (isbn) = 50 bytes

Total: 10.000.000 * 154 bytes = 1.4 Gb.

Un índice de este tamaño requiere un tiempo de carga importante y es poco manejable en memoria

Árboles B

- Un árbol B es una estructura de ficheros que permite indexar un fichero **sin consumir memoria primaria** y con un rendimiento cercano al de un índice simple
- Son la base de las estructuras de ficheros empleadas por las bases de datos modernas

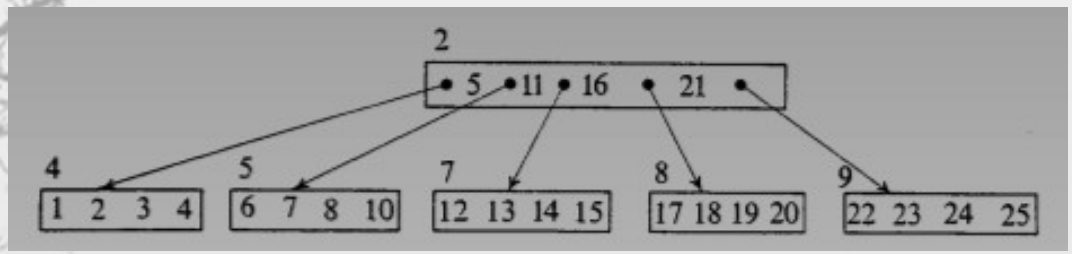
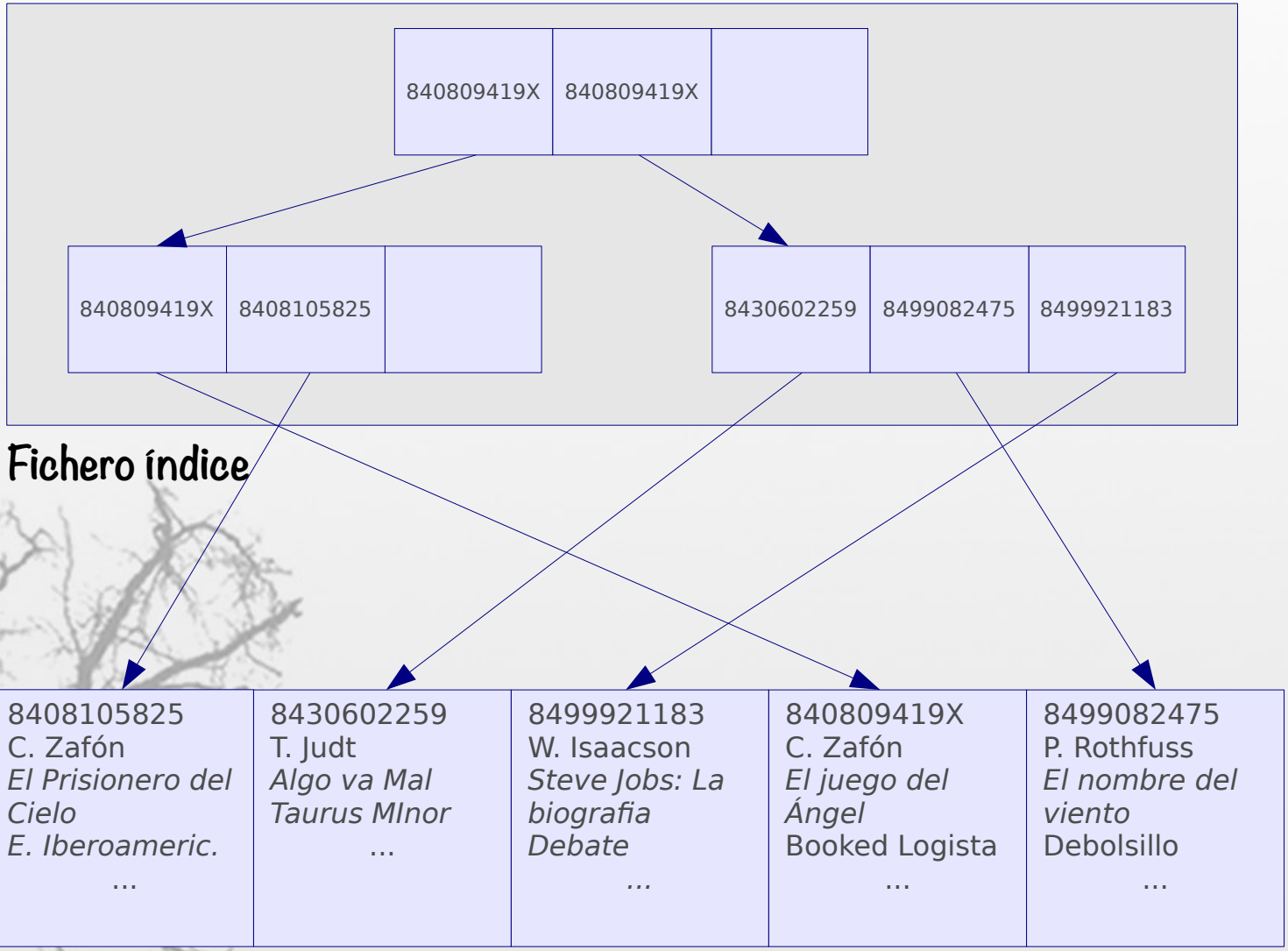


Ilustración original del artículo de Bayer y McCreight "Organization and Maintenance of Large Ordered Indexes" Acta Informática I (1972) donde se describieron por primera vez los árboles B.

Propiedades de un árbol B

- Un árbol B guarda en cada nodo un conjunto de tuplas (clave, posición) donde posición es:
  - La posición de un nodo hijo en el fichero de índice (nodos interiores)
  - La posición de un registro en el fichero de datos (nodos hoja)
- Si *m* es el orden del árbol B, todos los nodos exceptuando la raíz tienen entre *m/2* y *m* tuplas (clave, posición)
- Las tuplas en un nodo están ordenadas por clave

Estructura de un árbol B



Estructura de un árbol B

- Propiedad: un árbol B es un árbol equilibrado, donde la profundidad de cada hoja es siempre la misma
- El último nivel es similar a un índice simple pero dividido en bloques (los nodos hoja) y residente en un fichero en lugar de memoria
- Cada nivel del árbol “indexa” los nodos del siguiente nivel usando la primera clave de cada uno
- El orden de un árbol B debe ser grande para sea eficiente a efectos prácticos (64 o superior)

Búsqueda en árboles B

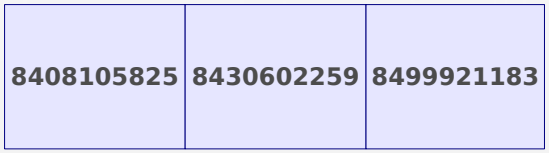
- Muy eficiente: $O(\log_m n)$ siendo *m* el orden del árbol
  - Ejemplo: localiza un dato entre diez millones con 4 accesos a disco para *m*=256
- Búsqueda del dato con clave *x*:
  1. Pasar el nodo raíz del árbol B a memoria
  2. Buscar la mayor clave del nodo tal que *k*<=*x* y obtener su dirección *d*:
     - Si el nodo es una hoja, leer el registro en la posición d del fichero de datos
     - Si no, pasar a memoria el nodo del árbol en la posición *d* y continuar por el paso 2

Inserción en árboles B

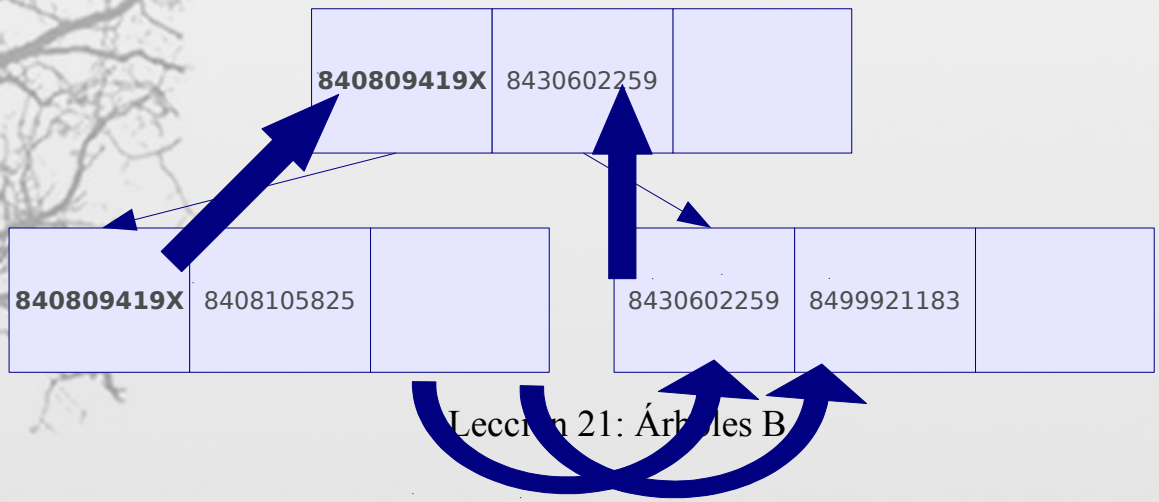
- También muy eficiente: $O(\log_m n)$
- Inserción de la clave *x*:
  1. Localizar el nodo hoja que aloja la clave *x* mediante una búsqueda
  2. Insertar la clave *x* y la posición del registro en el fichero de datos
     - Si la clave cabe en el nodo pero cae en la primera posición, modificar la clave en el padre
     - Si la clave no cabe en el nodo, crear un nuevo nodo, repartir las claves e insertar en el que corresponda. Insertar la primera clave del nuevo nodo en el padre

Inserción en árboles B

Insertar 8408105825, 8430602259, 8499921183 (se omite el fichero de datos)

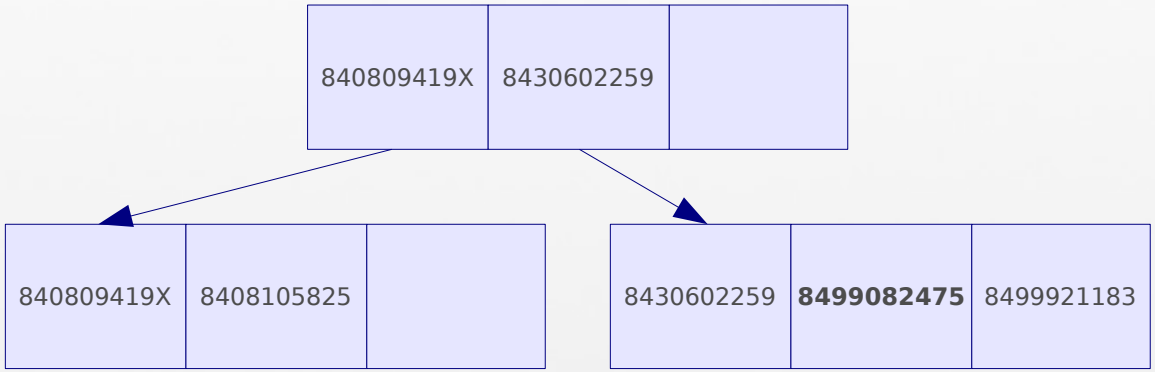


Insertar 840809419X. Se duplica el nodo, se redistribuyen las claves y se crea una nueva raiz

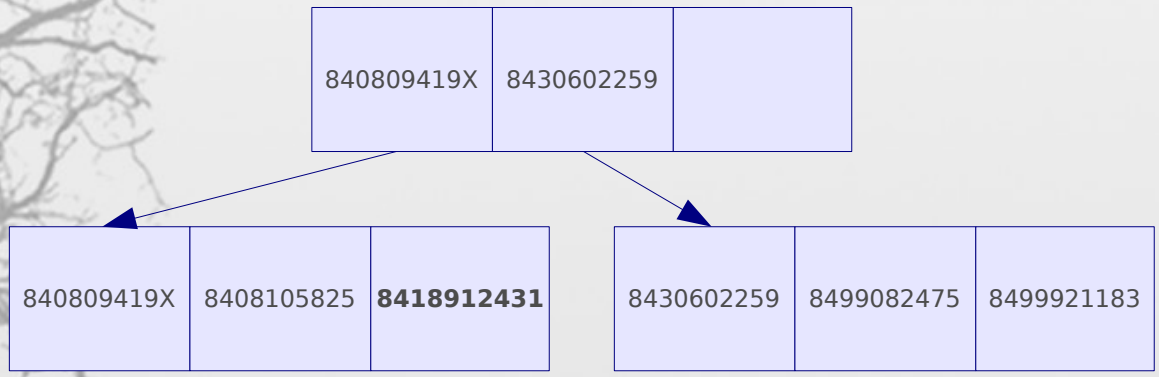


Inserción en árboles B

Insertar 8499082475. Caso trivial

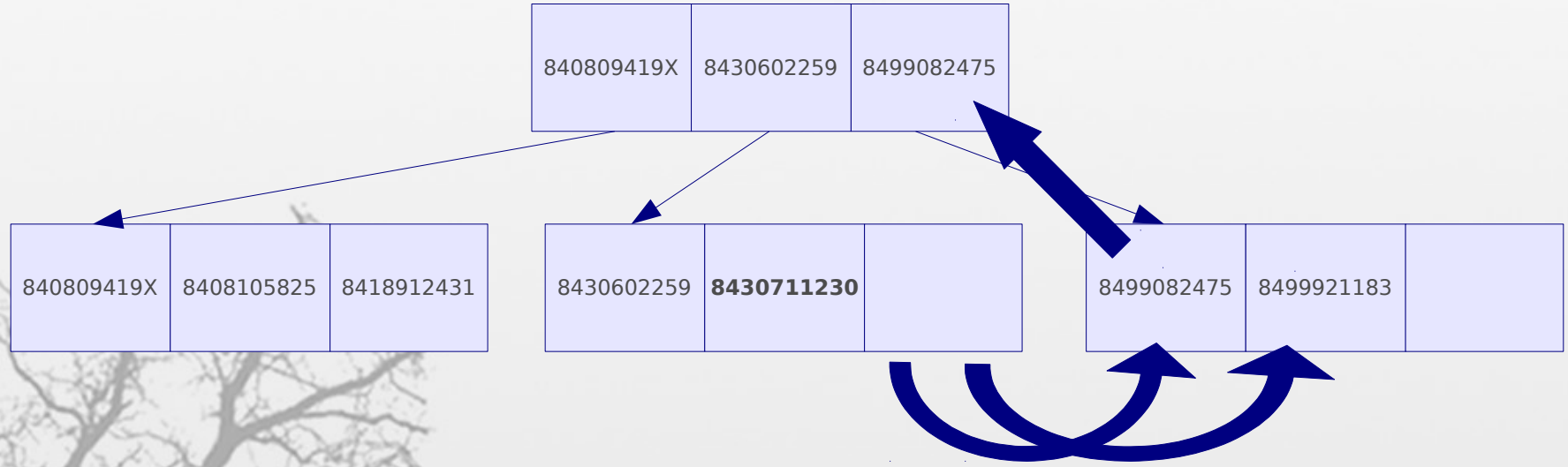


Insertar 8418912431. Caso trivial



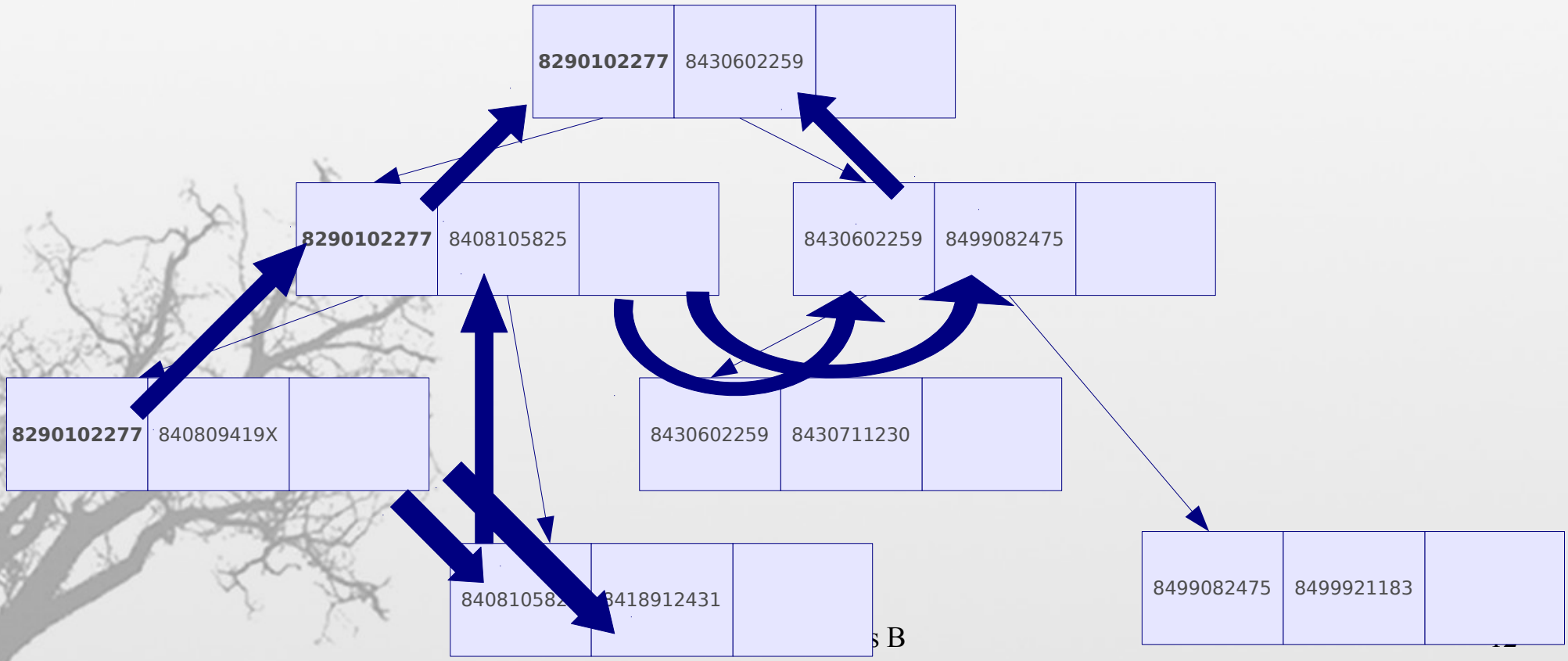
Inserción en árboles B

Insertar 8430711230. Nueva división y redistribución



Inserción en árboles B

Insertar 8290102277. Se duplica y redistribuye. Al insertar la nueva clave en el padre se produce una nueva duplicación y redistribución, y se genera una nueva raíz

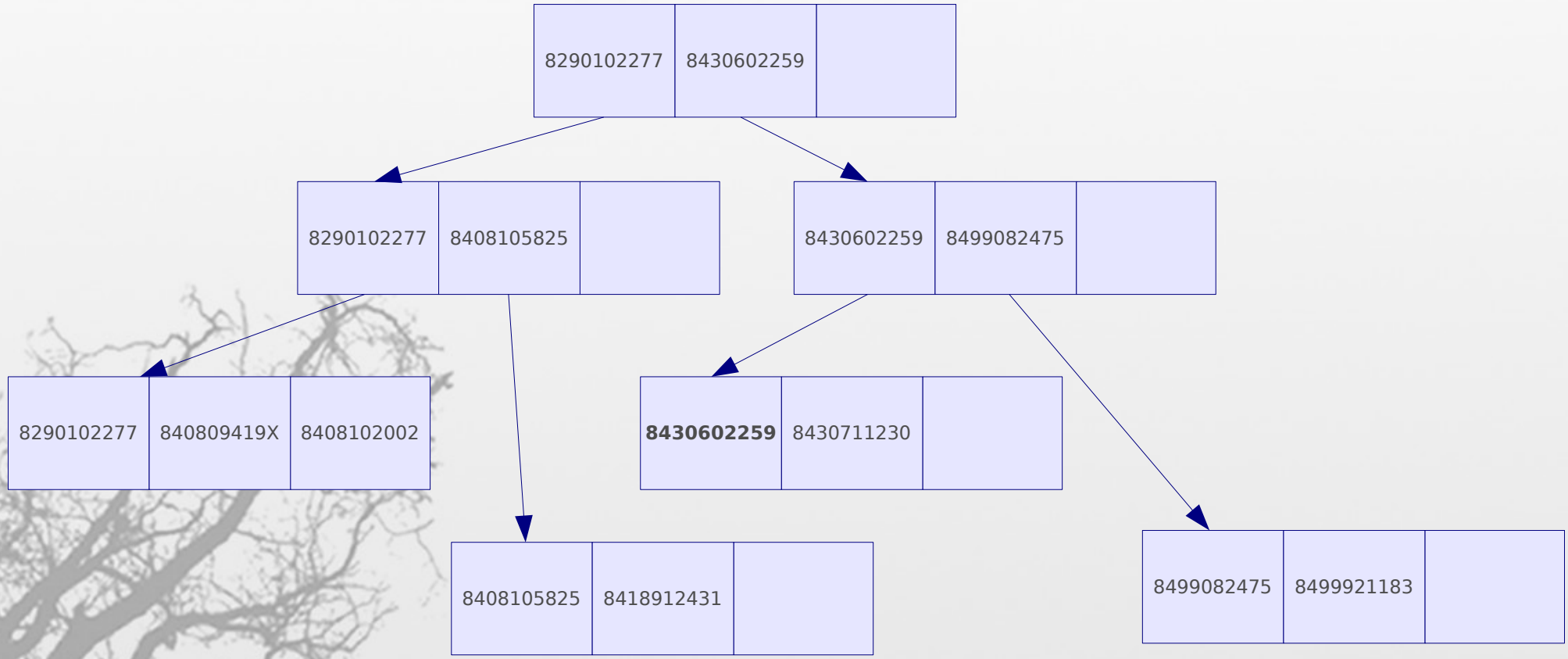


Borrado en árboles B

- Mismo orden: $O(\log_m n)$
- Borrado de la clave *x*:
  1. Localizar el nodo hoja que aloja la clave *x* mediante una búsqueda
  2. Eliminar la clave *x* y la posición del registro del fichero de datos
     - Si se borra la primera clave, cambiar la clave en el nodo padre por la segunda clave del nodo
     - Si la ocupación cae por debajo del 50%:
       - Enviar las claves restantes a un nodo hermano y eliminar nodo. Eliminar la clave correspondiente en el padre
       - Si no es posible lo anterior, traer una clave de un nodo hermano para llegar al 50%

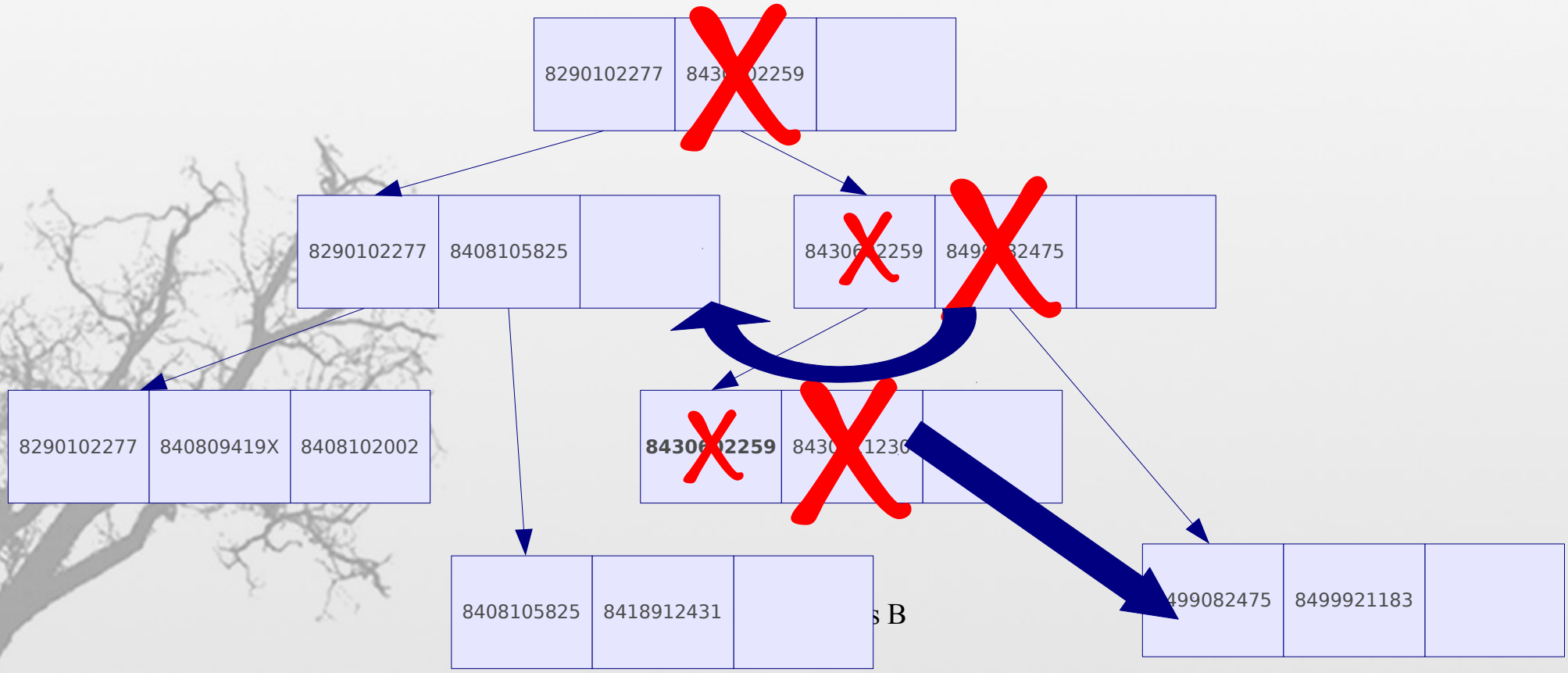
Borrado en árboles B

Situación inicial. Borrado clave 843062259



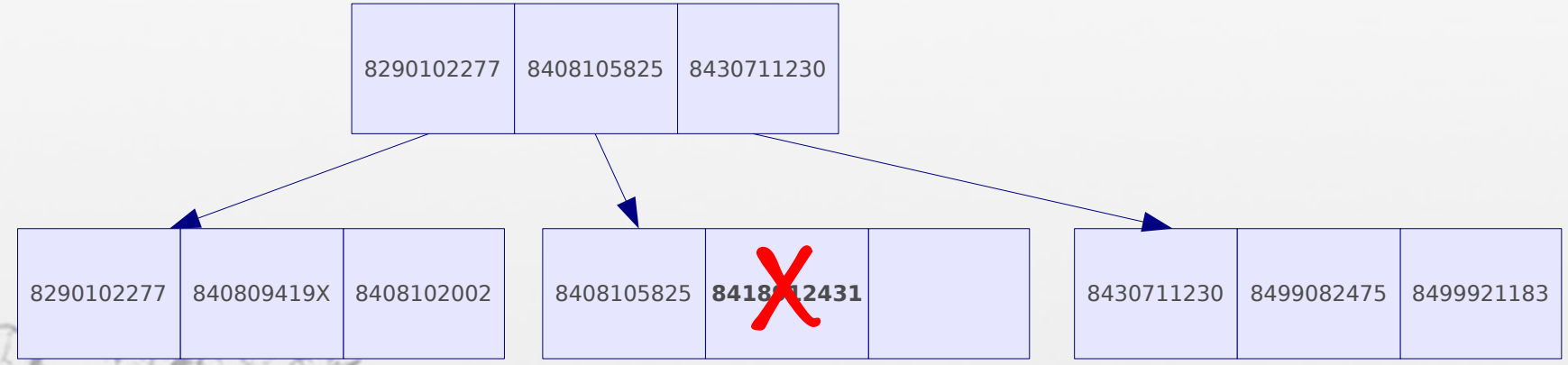
Borrado en árboles B

Se elimina la clave y la ocupación cae por debajo del 50%. Enviar la clave restante (8430711230) al nodo hermano derecha. Borrar el nodo y eliminar la clave en el padre. Ocurre la misma situación. Al final la antigua raíz es eliminada



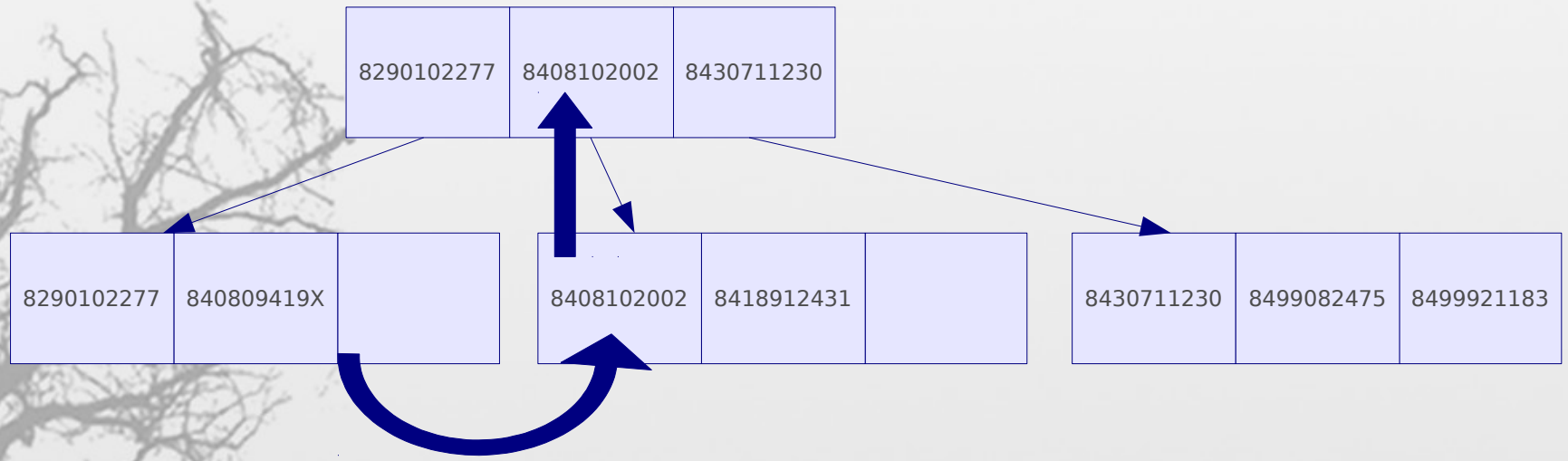
Borrado en árboles B

Estructura del árbol tras el borrado. Borrar 8418912431 a continuación.



Borrado en árboles B

Se elimina la clave. La ocupación del nodo cae por debajo del 50% pero la clave restante (8412912431) no puede transferirse a los hermanos. Solución: traer la clave 8408102002 del hermano izquierdo. Actualizar el padre al modificar la primera clave del nodo.



Conclusiones

- Un árbol B es un árbol *m*-ario (con *m* grande) equilibrado
- Se mantiene permanentemente en memoria secundaria por lo que puede crecer arbitrariamente
- Garantiza $O(\log_m n)$ accesos al fichero índice en sus tres operaciones básicas, aunque lógicamente la inserción y el borrado son mucho más complejos que la búsqueda
- Es la estructura de ficheros que sirve como base para la implementación de los modernos sistemas de bases de datos