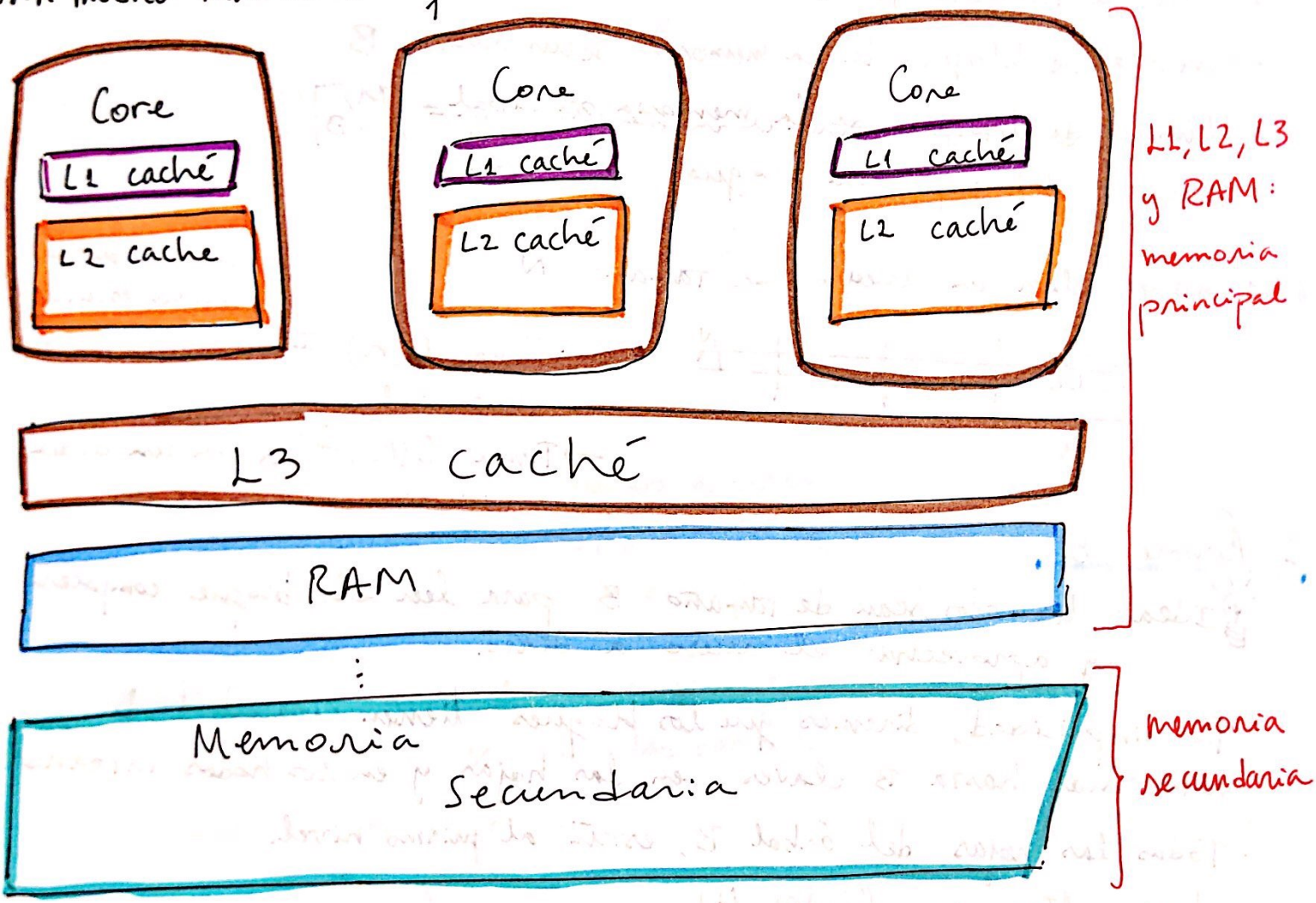
Memoria Externa: las operaciones en memoria externa (o secundaria) son mucho más lentas que en la memoria principal (o primaria)



Ojo No es importante que se sepan este diagrama, pero ayuda a visualizar por qué es más lento operar en memoria secundaria.

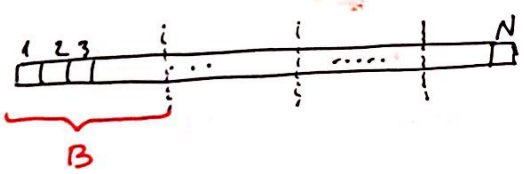
La memoria principal es más pequeña que la secundaria, por lo que, cuando tratemos muchos datos estaremos obligados a usar la secundaria.

Usar estructuras de datos que nos permitan acceder a la memoria secundaria EFICIENTEMENTE.

Nuestro modelo de memoria:

- Tamaño memoria principal M
- input de tamaño N
- Tamaño de bloques de la memoria secundaria B
- Tamaño de memoria principal en bloques $m = \lceil M/B \rceil$
- Tamaño del input en bloques $n = \lceil N/B \rceil$

// Ejemplo: leer un arreglo de tamaño N



→ input/outputs lectura/escritura

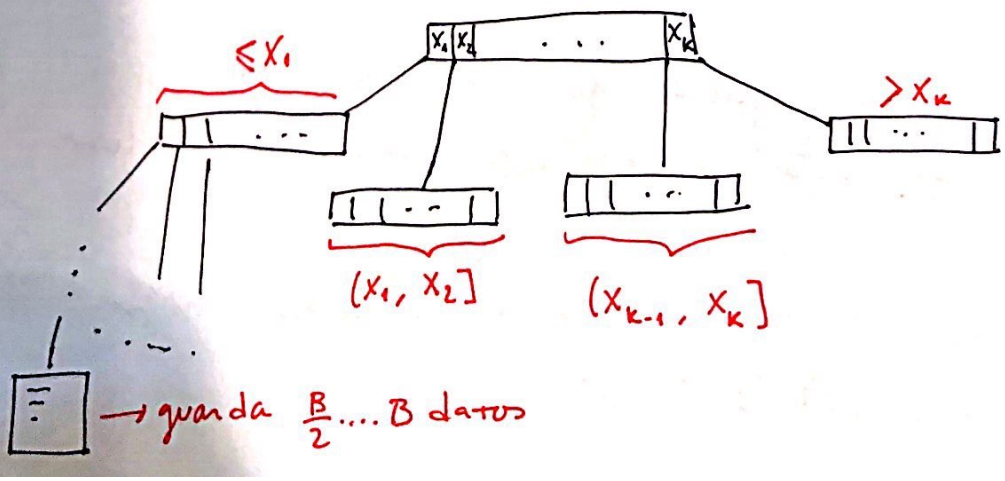
→ Toma $O(n)$ I/Os cuando es secuencial.

→ Toma $O(N)$ I/Os en un orden arbitrario

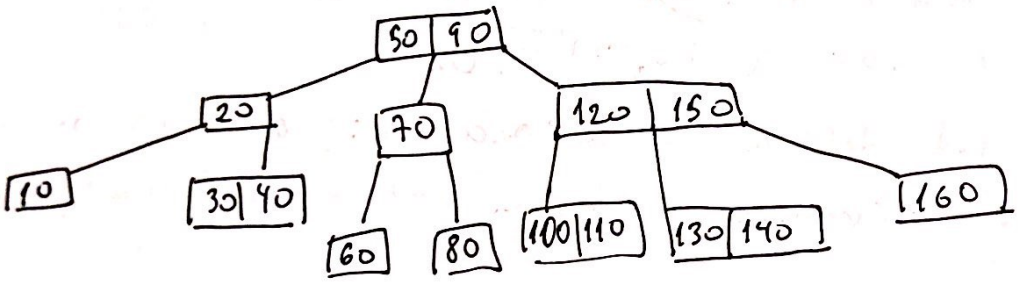
1. Árboles B

Idea: los nodos sean de tamaño B para leer el bloque completo y aprovechar el acceso a disco.

- Por simplicidad, diremos que los bloques tienen capacidad de almacenar hasta B claves, en las hojas y en los nodos internos.
- Todas las hojas del árbol B, están al mismo nivel.
  ↳ su altura es $O(\log_B N)$
- Cada nodo guarda de $\frac{B}{2}$ a B datos, salvo la raíz que guarda de 1 a B datos.



// Ej: árbol 2-3 (wikipedia)



· Búsqueda: $O(\log_B N)$ I/O's → la altura del árbol

↳ ¿Qué pasa si el árbol está llenado a la mitad?
El costo será $O(\log_{B/2} N)$. $\left(O(\log_B n) \simeq O(\log_B N)\right)$

↳ ¿Cómo disminuir los accesos a disco?
Almacenando los primeros $\Theta(\log_B M)$ niveles en memoria principal. De esta forma, se reduce a $O(\log_B \frac{N}{M})$ I/O's.

· Inserción / Borrado: se busca el elemento y luego se añade / borra y se hacen las operaciones correspondientes para que se mantenga la estructura del árbol B (de $\frac{B}{2}$ a B claves por nodo).

2. Ordenamiento

Extender algoritmo de MergeSort a disco.

I) Dividir el arreglo en sub-arreglos de tamaño B
II) Ordenar estos sub-arreglos y reescribirlos ($O(1)$ I/O's)
III) Unir los sub-arreglos en orden → leer el subarreglo completo y reescribirlo ($O(n)$ I/O's)

Esta unión se debe realizar hasta que se forme el arreglo completo ordenado ($\log_2(n)$ veces)

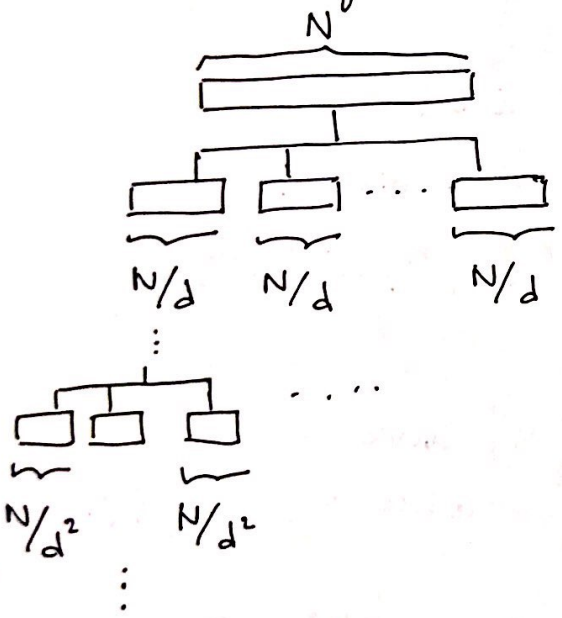
↳ Costo total = $O(n \log_2 n)$ I/O's

¿Cómo mejorar este costo?

se lee de a bloques!

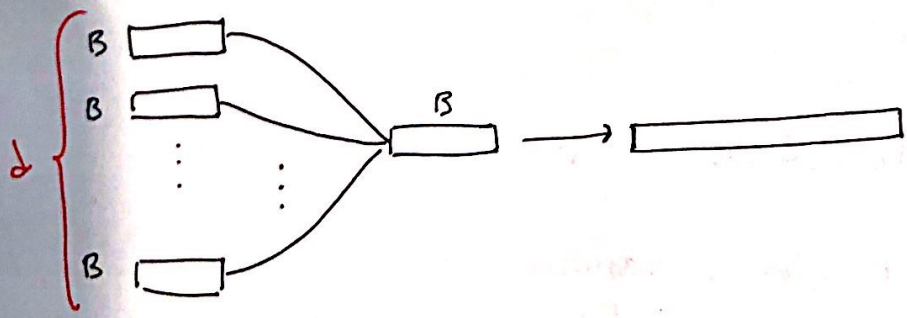
1) Ordenamos sub-arreglos de tamaño M en memoria principal a costo cero. → costo = $O(n \log_{2m} \frac{n}{m})$ I/Os

2) Aumentar la aridad del árbol de recursión, es decir, dividir el arreglo recursivamente en $d \geq 2$. → costo = $O(n \log_d \frac{N}{M})$



$\log_d(\frac{N}{M}) = \log_d(\frac{n}{m})$

¿"d" puede ser infinito? NO, pues al unir los subarreglos necesitamos un buffer de B elementos. Luego, $(d+1) \cdot B \leq M$



$d \leq \frac{M}{B} - 1 \rightarrow d = \frac{M}{B}$

↓

bloques que caben en la memoria principal