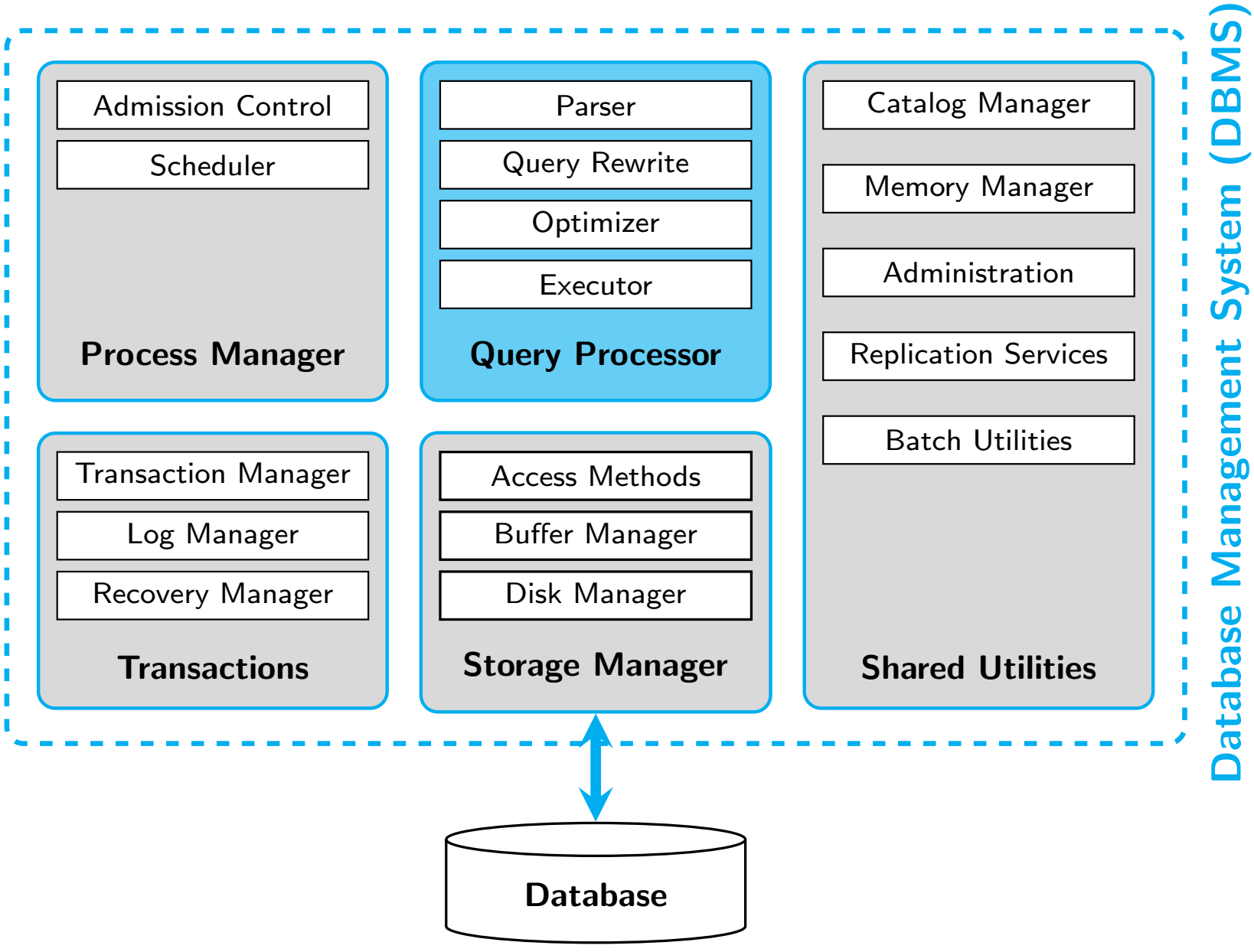
Sorting

Clase 11

IIC 3413

Prof. Cristian Riveros

Implementación de operadores relacionales



Implementación de operadores relacionales

¿qué es un **operador físico**?

- Cada operador físico implementa un operador relacional (lógico).
- Orientado a desempeñarse bien en una tarea específica.

Cada **variante** aprovecha propiedades físicas de los datos:

- Presencia o ausencia de índices.
- Orden del input.
- Tamaño del input.
- Cantidad de elementos distintos.
- Espacio disponible en memoria.

Operadores físicos relacionales

Selección (σ)

Proyección (π)

Join ($\bowtie$)

Intersección ($\cap$)

Unión ($\cup$)

Elim. duplicados (δ)

Sorting (τ)

GroupBy (γ)

Operadores físicos relacionales

Selección (σ)

Unión ($\cup$)

Proyección (π)

Elim. duplicados (δ)

Join ($\bowtie$)

Sorting (τ)

Intersección ($\cap$)

GroupBy (γ)

¿por qué necesitamos sorting?

- Uso explicito en SQL.

```
SELECT pName, pPosition FROM Players ORDER BY pPosition
```

- Uso para otros operadores:
  - Eliminación de duplicados.

```
SELECT DISTINCT pName, pPosition FROM Players
```

  - Joins.
  - Intersección.
  - Group by.

- Uso para construir **índices** y otras estructuras.

Sorting es un operador básico dentro del funcionamiento de una DBMS.

Sorting de tuplas

Definición

Un conjunto de tuplas $\{t_1, \ldots, t_n\}$ esta **ordenado** (lexicográficamente) con respecto a una secuencia de atributos $(k_1, \ldots, k_m)$ si:

$$t_i \leq_{k_1,\ldots,k_m} t_{i+1} \Rightarrow t_i(k_1) < t_{i+1}(k_1) \vee (t_i(k_1) = t_{i+1}(k_1) \wedge t_i \leq_{k_2,\ldots,k_m} t_{i+1})$$

Los algoritmos de **sorting** son muy conocidos
¿por qué queremos reestudiar sorting?

Sorting externo

¿cómo podemos ordenar un conjunto de tuplas
cuyo tamaño excede por mucho el espacio disponible en RAM?

Algoritmo preferido: external **merge sort**.

¿cómo funciona el algoritmo Merge-Sort?

Outline

Merge sort

External merge sort

Optimizaciones

Outline

Merge sort

Merge sort: Ordenar listas ordenadas

¿cómo ordenamos dos **listas ordenadas** en una sola lista?

Lista 1	Lista 2	
$4, 7, 17, 23$	$1, 9, 10, 15$	
$4, 7, 17, 23$	$9, 10, 15$	1
$7, 17, 23$	$9, 10, 15$	$1, 4$
$17, 23$	$9, 10, 15$	$1, 4, 7$
$17, 23$	$10, 15$	$1, 4, 7, 9$
$17, 23$	15	$1, 4, 7, 9, 10$
$17, 23$		$1, 4, 7, 9, 10, 15$
		$1, 4, 7, 9, 10, 15, 17, 23$

Merge sort: Ordenar listas ordenadas

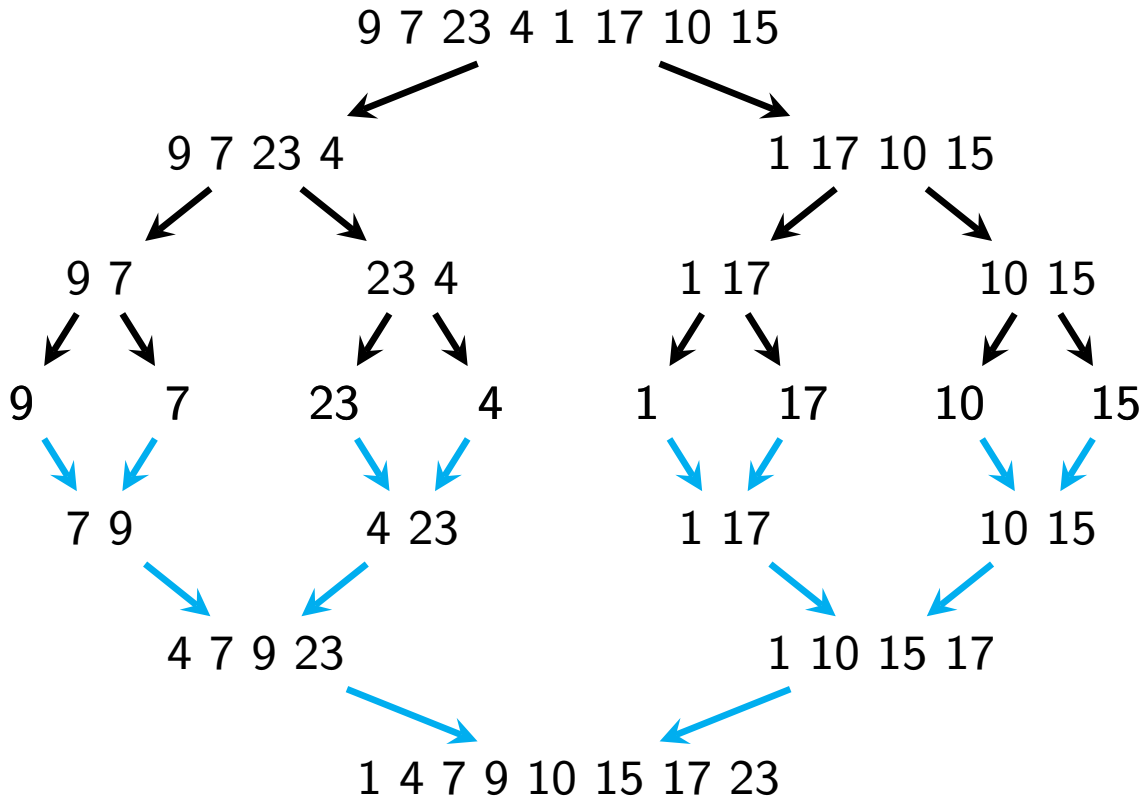
input : Dos listas ordenadas L_1 y L_2
output: Una lista ordenada L con los elementos de L_1 y L_2

Function `merge` *(L_1, L_2)*
    L := lista vacía
    while *L_1 y L_2 son ambas NO vacías* **do**
        remover el menor elemento m entre L_1 y L_2
        agregar m al final de L
    if *L_1 esta vacía* **then**
        Remover elementos de L_2 y agregar al final de L
    else if *L_2 esta vacía* **then**
        Remover elementos de L_1 y agregar al final de L
    return L

Importante: Merge es un algoritmo **de una pasada** por los datos.

Merge sort: usado merge

¿cómo podemos ocupar `merge` para ordenar?



Merge sort: algoritmo

input : Una lista de números $L = a_1, \ldots, a_n$

output: L en orden creciente

Function `mergesort` *(L)*

- **if** $n = 1$ **then**
  - **return** L
- **else**
  - $m := \lfloor n/2 \rfloor$
  - $L_1 := a_1, a_2, \ldots, a_m$
  - $L_2 := a_{m+1}, a_{m+2}, \ldots, a_n$
  - **return** `merge`(`mergesort`(L_1), `mergesort`(L_2))

¿cuántas tiempo tardaremos en el **peor caso**?

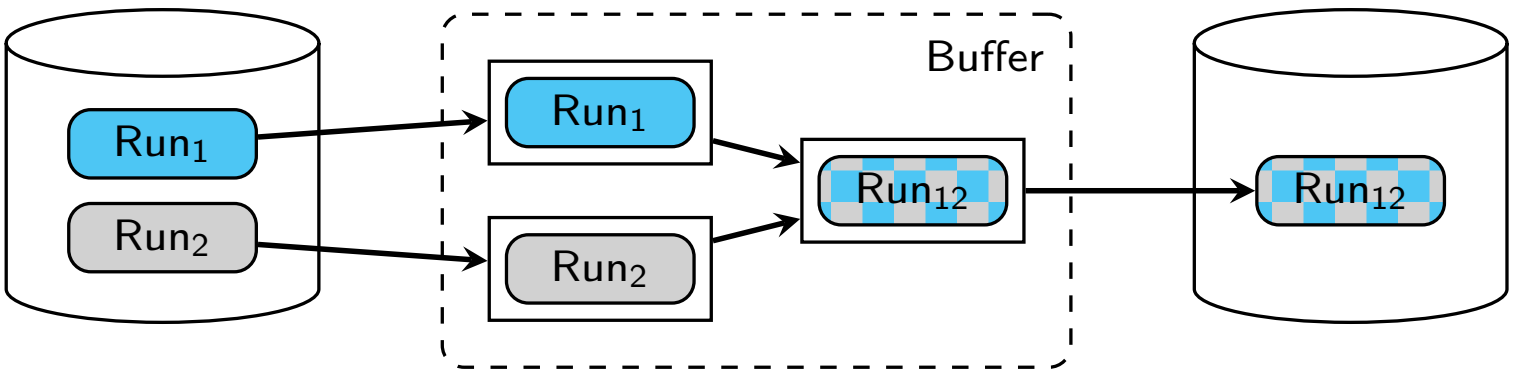
Outline

External merge sort

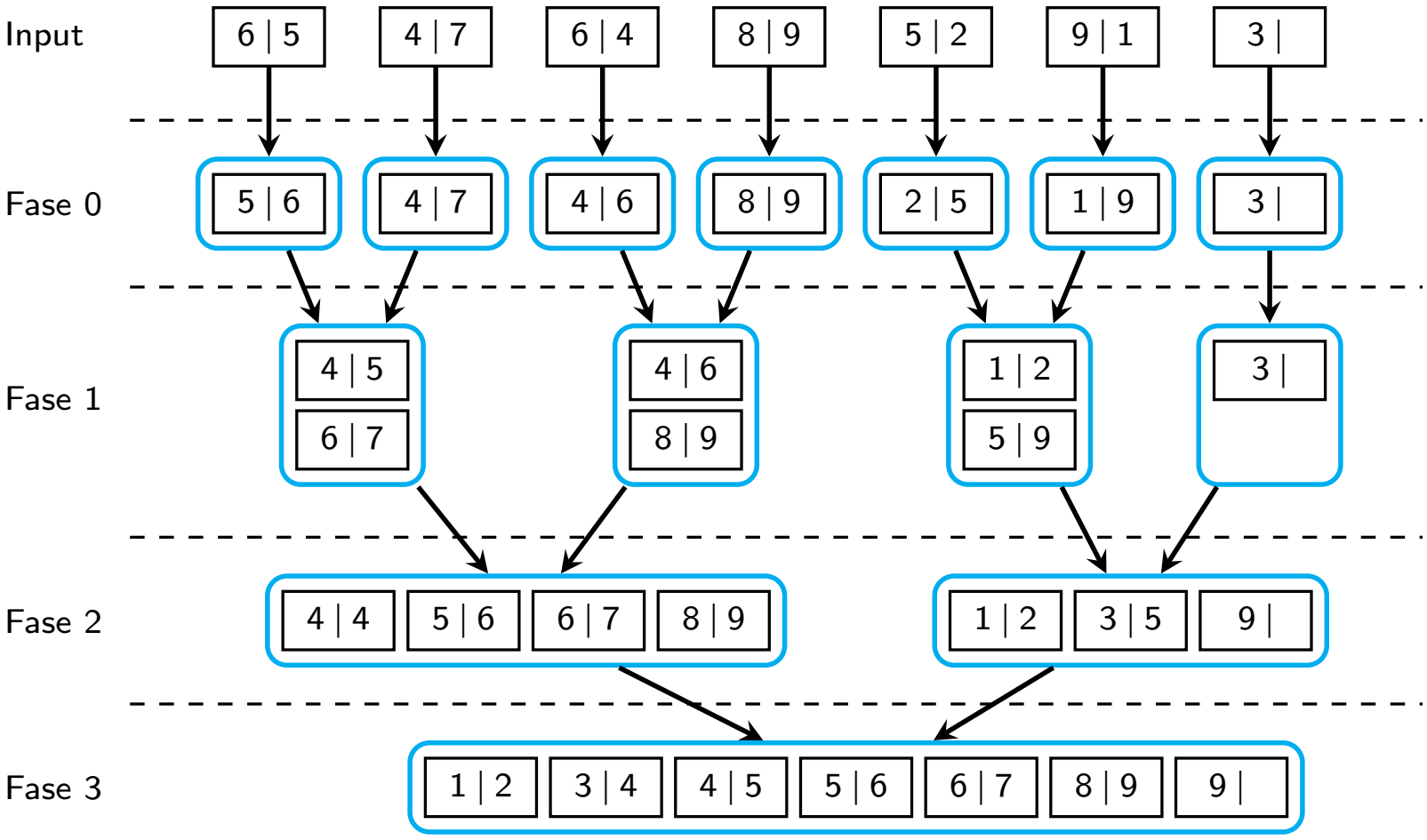
Definición

Run: secuencia de páginas que contiene una conjunto ordenado de tuplas.

- External merge sort funciona por **fases**.
- Fase 0: creamos los runs iniciales.
- Fase i:
  1. Traemos los runs desde el disco a memoria de a pares.
  2. Hacemos **merge** de cada par de runs.
  3. Almacenamos el nuevo run de regreso al disco.



Ejemplo del funcionamiento de External Merge-sort



Algoritmo External Merge Sort

input : Páginas $P = \{p_1, \ldots, p_n\}$ con tuplas

output: Lista de páginas P' con las tuplas ordenadas

Function `externalMergeSort` *(P)*

- **foreach** $p \in P$ **do** `/* Fase 0 */`
  - Leemos p a memoria
  - Ordenamos p en r
  - Escribimos r en disco
- $R := \{r_1, \ldots, r_n\}$ `/* Runs iniciales */`
- **while** $|R| > 1$ **do** `/* Fase i */`
  - $R' := \varnothing$
  - **foreach** $r_1, r_2 \in R$ **do** `/* Para cada par de runs */`
    - Leemos r_1 y r_2 a memoria ($R := R - \{r_1, r_2\}$)
    - Hacemos merge r_1 y r_2 ($r := \text{Merge}(r_1, r_2)$)
    - Escribimos r en disco ($R' := R' \cup \{r\}$)
  - $R := R'$
- **return** único run en R

Eficiencia (en I/O) de External Merge Sort

Considere N : número de páginas del archivo.

- Cada fase lee y escribe cada página del archivo.
- ¿cuánto es el máximo número de "fases"?

$$\underbrace{1}_{\text{Fase } 0} + \underbrace{\lceil \log_2 N \rceil}_{\textit{Fases}}$$

- ¿cuánto es el costo total en I/O?

$$2 \cdot N \cdot (1 + \lceil \log_2 N \rceil)$$

Eficiencia (en I/O) de External Merge Sort

Costo en I/O:

$$2 \cdot N \cdot (1 + \lceil \log_2 N \rceil)$$

Ejemplo

Considere un archivo de 8GB y páginas de 8 KB $\approx$ 1.048.576 páginas.

- Total de I/O $= 2 \cdot 1.048.576 \cdot (1 + \lceil \log_2 1.048.576 \rceil) = 44.040.192$ I/O.
- Si consideramos que cada I/O toma $\approx 0.1ms$ (minimo): ...

1.2 horas (tiempo total)!!!

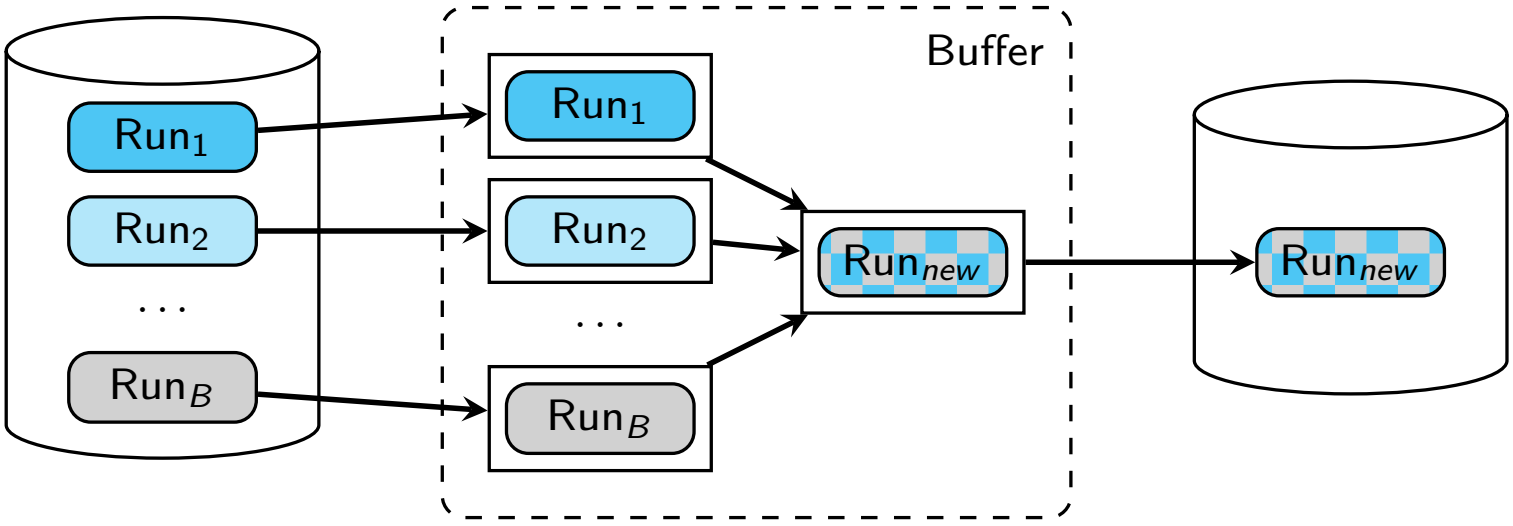
¿es posible optimizar external merge-sort?

Outline

Optimizaciones para External Merge Sort

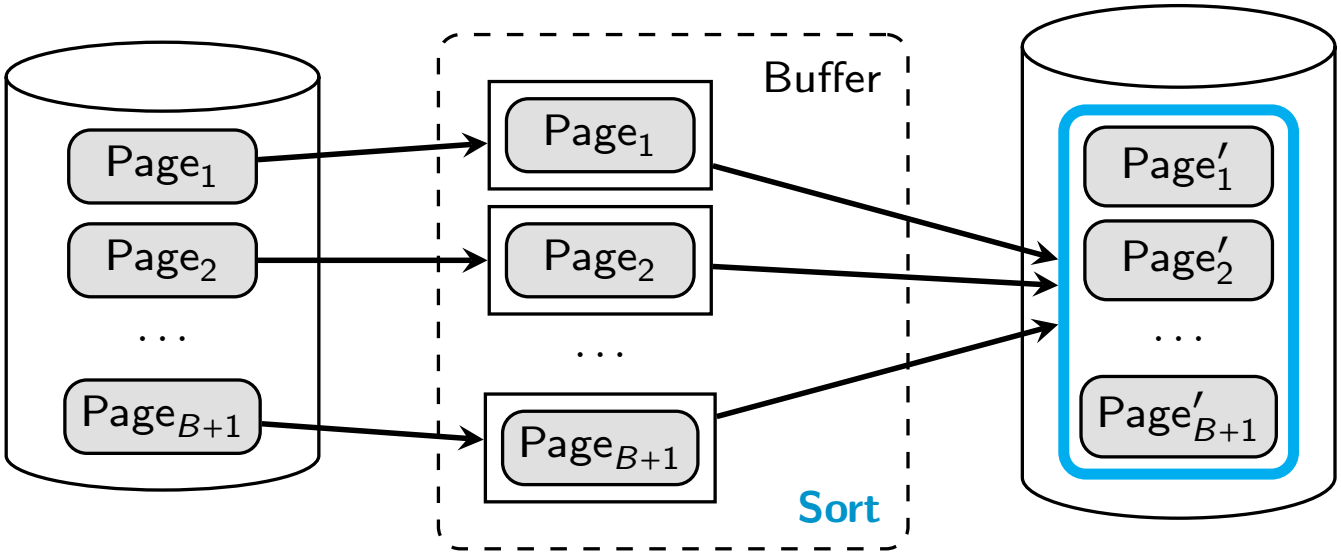
1. Aumentar el número de runs en el Merge.
2. Reducir el número de runs iniciales.
3. Evitar escribir el último resultado (**pipeline**).

Aumentar el número de runs en el Merge



Contamos con $B + 1$ páginas en buffer en vez de 3.

Reducir el número de runs iniciales



Usamos las $B+1$ páginas en buffer
para crear un runs iniciales con $B+1$ páginas.

Algoritmo External Merge Sort (optimizado)

input : Páginas $P = \{p_1, \ldots, p_n\}$ con tuplas y $B + 1$ páginas en buffer
output: Lista de páginas P' con las tuplas ordenadas

Function `externalMergeSort` *(P)*

- **while** $P \neq \emptyset$ **do** `/* Fase 0` `*/`
  - Leemos $\{p_1, \ldots, p_{B+1}\}$ a memoria ($P := P - \{p_1, \ldots, p_{B+1}\}$)
  - Ordenamos $\{p_1, \ldots, p_{B+1}\}$ (quicksort) y creamos un run r
  - Escribimos r en disco
- $R := \{r_1, \ldots, r_{\frac{n}{B+1}}\}$
- **foreach** $|R| > 1$ **do** `/* Fase i` `*/`
  - $R' := \emptyset$
  - **while** $R \neq \emptyset$ **do** `/* Para` B `runs` `*/`
    - Leemos $r_1, \ldots, r_B$ a memoria ($R := R - \{r_1, \ldots, r_B\}$)
    - Hacemos merge de $r_1, \ldots, r_B$ ($r := \text{Merge}(r_1, \ldots, r_B)$)
    - Escribimos r en disco ($R' := R' \cup \{r\}$)
  - $R := R'$
- **return** único run en R

Eficiencia (en I/O) de External Merge Sort Optimizado

Sea N : número de páginas del archivo y $B+1$ el tamaño del buffer.

- Cada fase lee y escribe cada página del archivo.
- ¿cuánto es el número de runs iniciales?

$$\left\lceil \frac{N}{B+1} \right\rceil$$

- ¿cuánto es el número de máximo de "fases"?

$$\underbrace{1}_{\text{Fase } 0} + \underbrace{\left\lceil \log_B \left\lceil \frac{N}{B+1} \right\rceil \right\rceil}_{Fases}$$

- ¿cuánto es el costo total en I/O?

$$2 \cdot N \cdot (1 + \left\lceil \log_B \left\lceil \frac{N}{B+1} \right\rceil \right\rceil)$$

Eficiencia (en I/O) de External Merge Sort Optimizado

Costo en I/O:

$$2 \cdot N \cdot (1 + \left\lceil \log_B \left\lceil \frac{N}{B+1} \right\rceil \right\rceil)$$

Ejemplo

Considere:

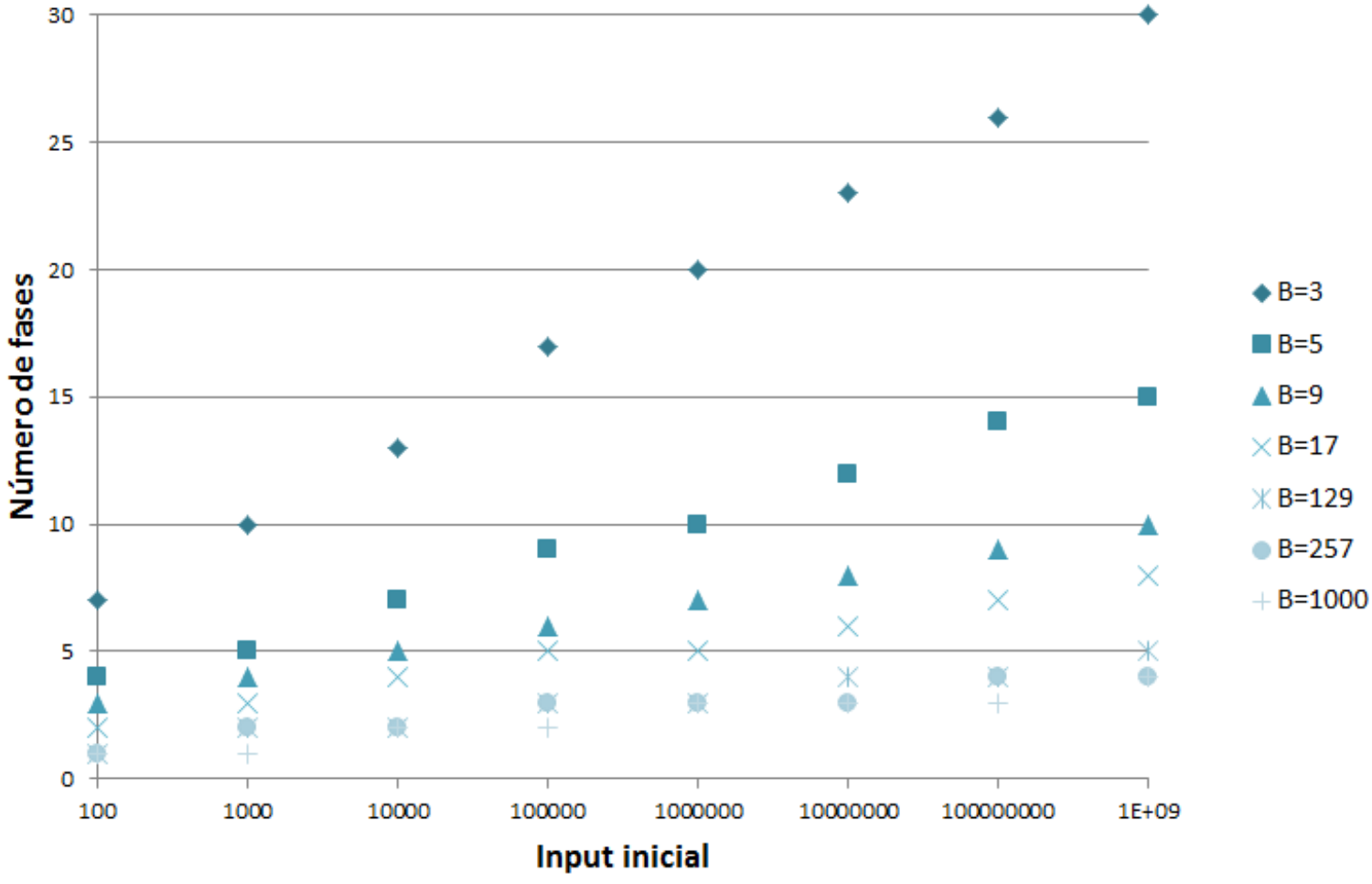
1. el mismo archivo de 8GB y páginas de 8 KB $\approx$ 1.048.576 páginas.
2. **memoria ram** para el buffer de 2GB, $B + 1 \approx 262.145$ páginas.

Si consideramos que cada I/O toma $\approx 0.1ms$ (minimo): ...

6.7 minutos!!!

...vamos bajando.

Número de fases según el tamaño del buffer



Número de fases según el tamaño del buffer

N	B=3	B=5	B=9	B=17	B=129	B=257	B=1000
100	7	4	3	2	1	1	1
10^3	10	5	4	3	2	2	1
10^4	13	7	5	4	2	2	2
10^5	17	9	6	5	3	3	2
10^6	20	10	7	5	3	3	3
10^7	23	12	8	6	4	3	3
10^8	26	14	9	7	4	4	3
10^9	30	15	10	8	5	4	4

- 10^9 páginas son 60 TB!
- 1000 páginas en buffer son solo 8 MB!

¿de qué tamaño debe ser el buffer para solo 2 fases?

$$
\begin{aligned}
2 &= 1 + \left\lceil \log_B \left\lceil \frac{N}{B+1} \right\rceil \right\rceil \\
&\Downarrow \quad \text{despejamos B} \\
B &\geq \sqrt{N}
\end{aligned}
$$

En el caso de una archivo de 10^9 páginas (60 TB):

solo necesitamos 240 MB de buffer!!

Conclusiones sobre el tamaño del buffer

Para efectos prácticos (N = número de páginas del input):

1. Escogemos el tamaño del buffer (B) dinámicamente e igual a $\sqrt{N}$.
2. Podemos asumir que External Merge Sort solo tarda 2 fases.

Costo de External Merge Sort $\approx 4 \cdot N$

Optimizaciones para External Merge Sort

1. Aumentar el número de runs en el Merge. ✓
2. Reducir el número de runs iniciales. ✓
3. Evitar escribir el último resultado (**pipeline**).

Evitar escribir el último resultado

Si consideramos que:

1. **Sorting** es una operación mas en nuestro plan físico.
2. Tuplas ordenadas son recibidas por otro operador (**pipeline**).

Podemos evitar escribir el último resultado en disco!!

En este caso: costo de External Merge Sort $\approx 3 \cdot N$.

Uso de B+-Tree para sorting

¿cómo podemos usar B+-Tree para sorting?

Si un **B+-tree** hace match para una tarea de sorting,
entonces considerar usar el índice.

- **Clustered** index : el mejor de los casos.
- **Unclustered** index : evitar hacer uso del índice.