

# Optimización

# HeapFile

## Registros sin orden

Al insertarse un registro, se lo agrega al final del archivo o en alguno de los bloques con espacio libre

Las operaciones de búsqueda requieren búsqueda lineal por **todos los bloques** del archivo

# SortedFile

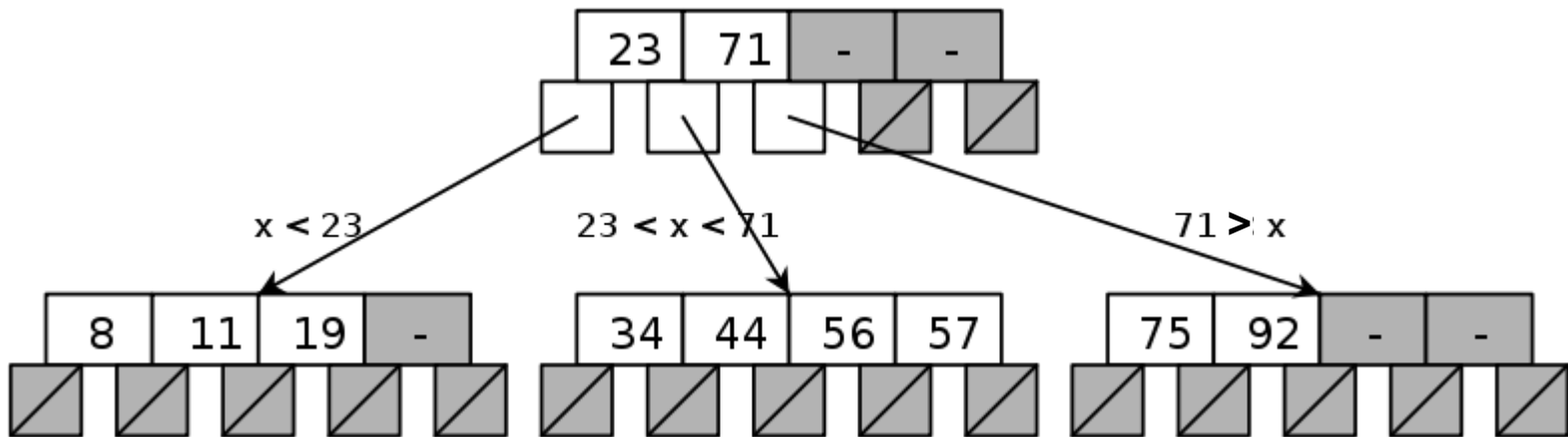
**Registros ordenados** a partir de una clave de búsqueda **A**

Al insertarse un registro se lo agrega ordenadamente, lo que puede provocar una reorganización en los bloques del archivo

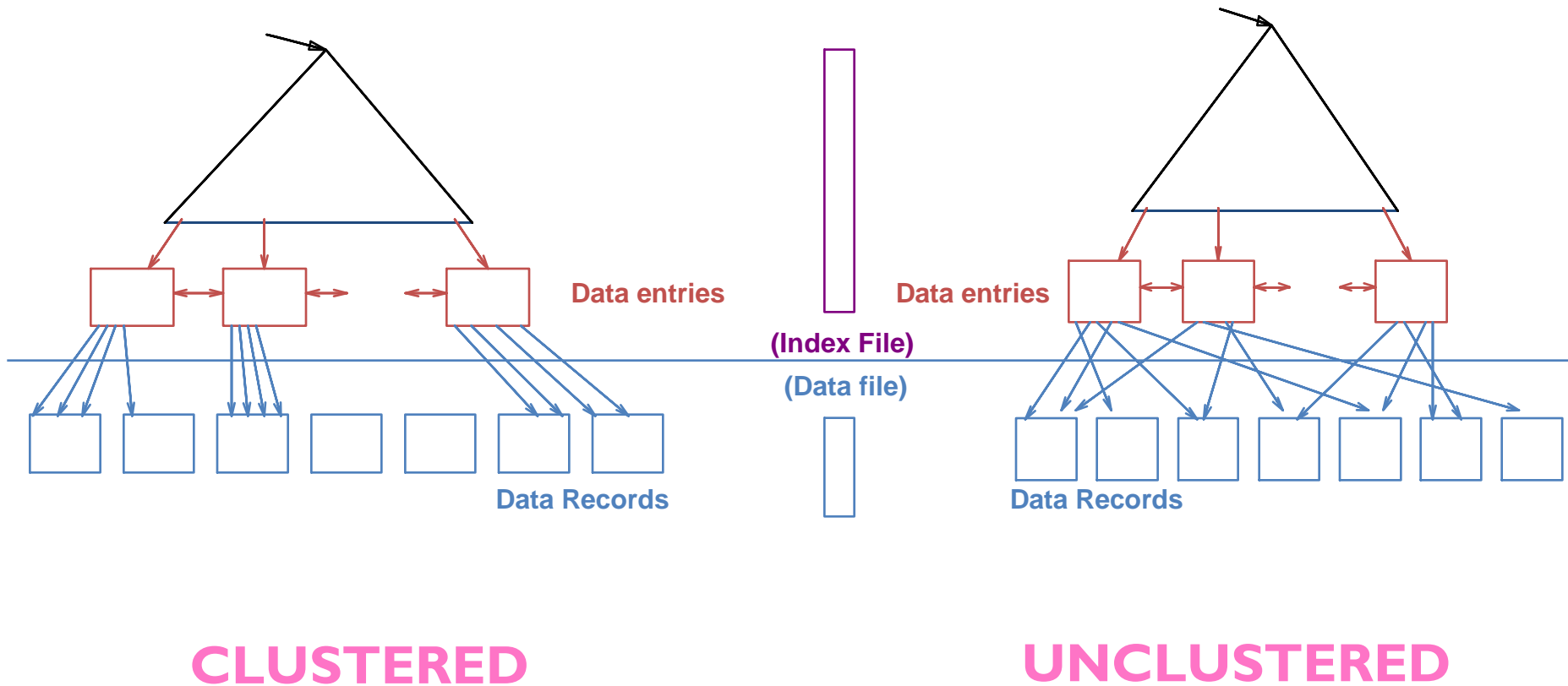
**Mejoran las búsquedas por A**, pero el resto de las operaciones suelen requerir una búsqueda lineal

# Índices B+

Los índices B+ son árboles balanceados



# Clustered vs. Unclustered Index

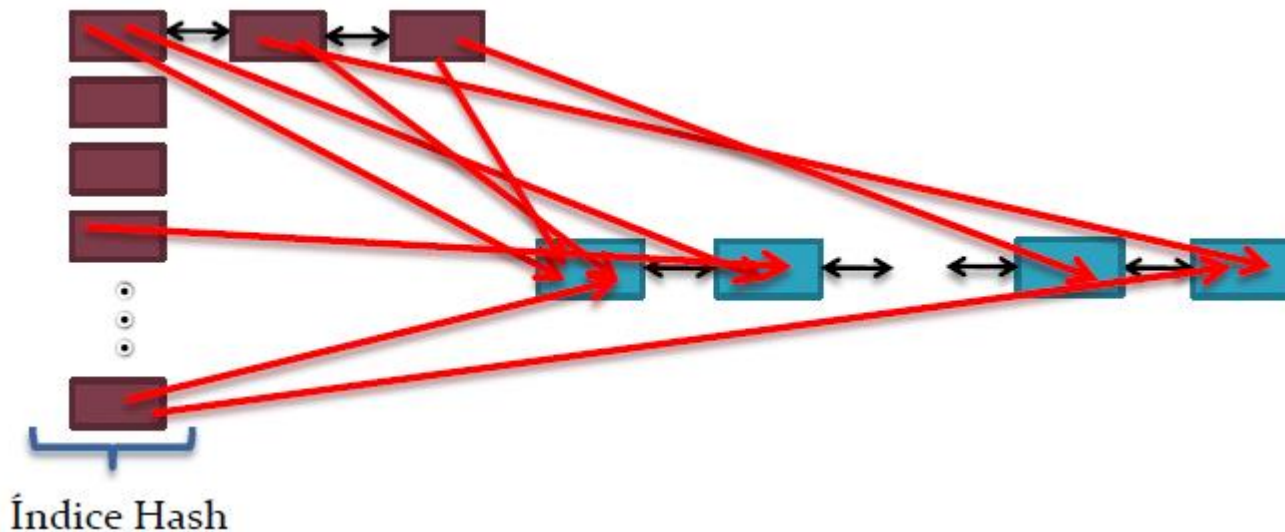


# Índices Hash

**Una tabla de hash almacena las claves de búsqueda**

Cada posición de una tabla hash se asocia con un conjunto de registros. Por esta razón cada posición suele llamarse “un bucket”, y los valores de hash, “ índices bucket”.


En cada bucket  **cantidad variable de bloques**

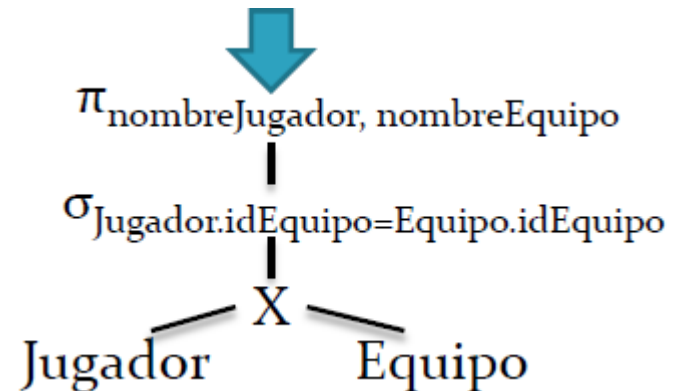


## Primer paso



```
SELECT nombreJugador, nombreEquipo  
FROM Jugador J, Equipo E  
WHERE J.idEquipo = E.idEquipo
```

  
 $\pi_{\text{nombreJugador, nombreEquipo}}(\sigma_{\text{Jugador.idEquipo=Equipo.idEquipo}}(\text{Jugador X Equipo}))$



## Segundo paso

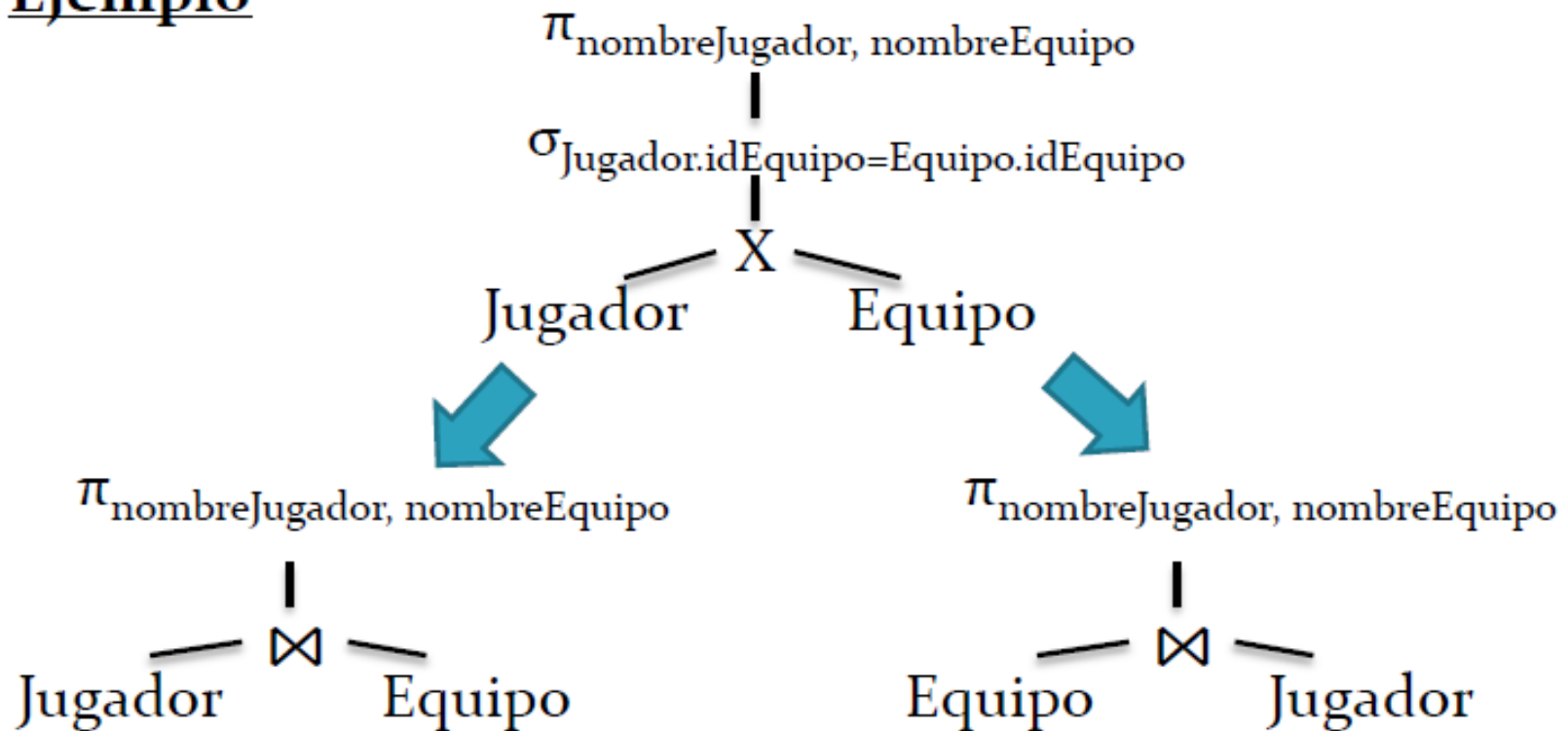
Modificaciones algebraicas sobre el árbol

Buscan mejorar la performance de la consulta **independientemente** de la organización física.

Involucran propiedades algebraicas que permiten construir una consulta **equivalente** a la original.



## Ejemplo



# Ejemplos

Cascada de selecciones

Bajar selecciones

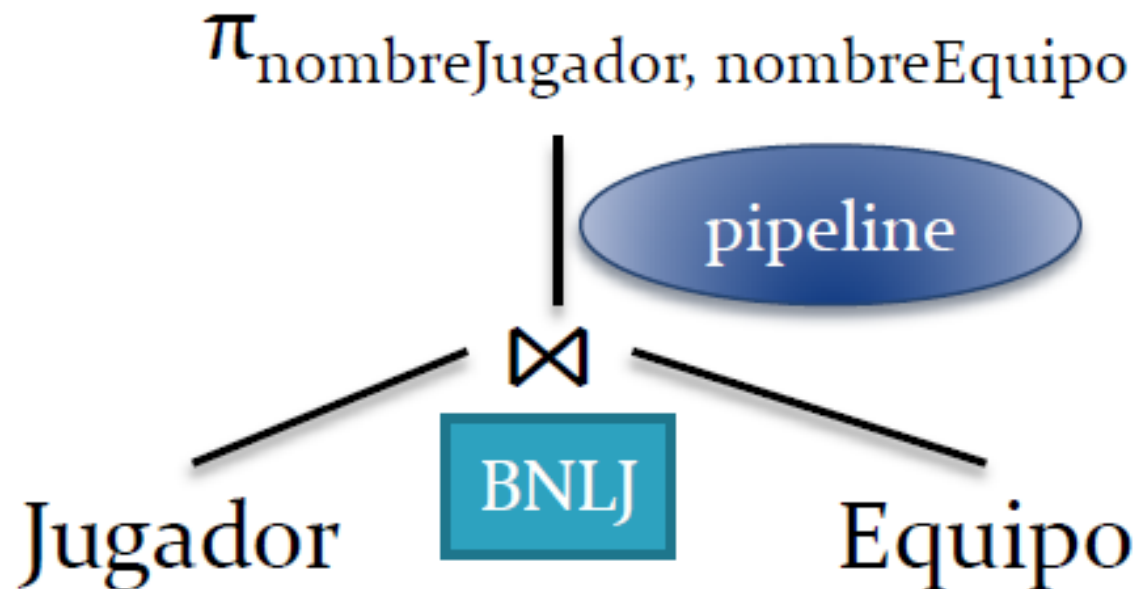
Bajar proyecciones

Cambiar productos cartesianos por joins

## Tercer paso

Selección de la implementación de cada operador

### Ejemplo

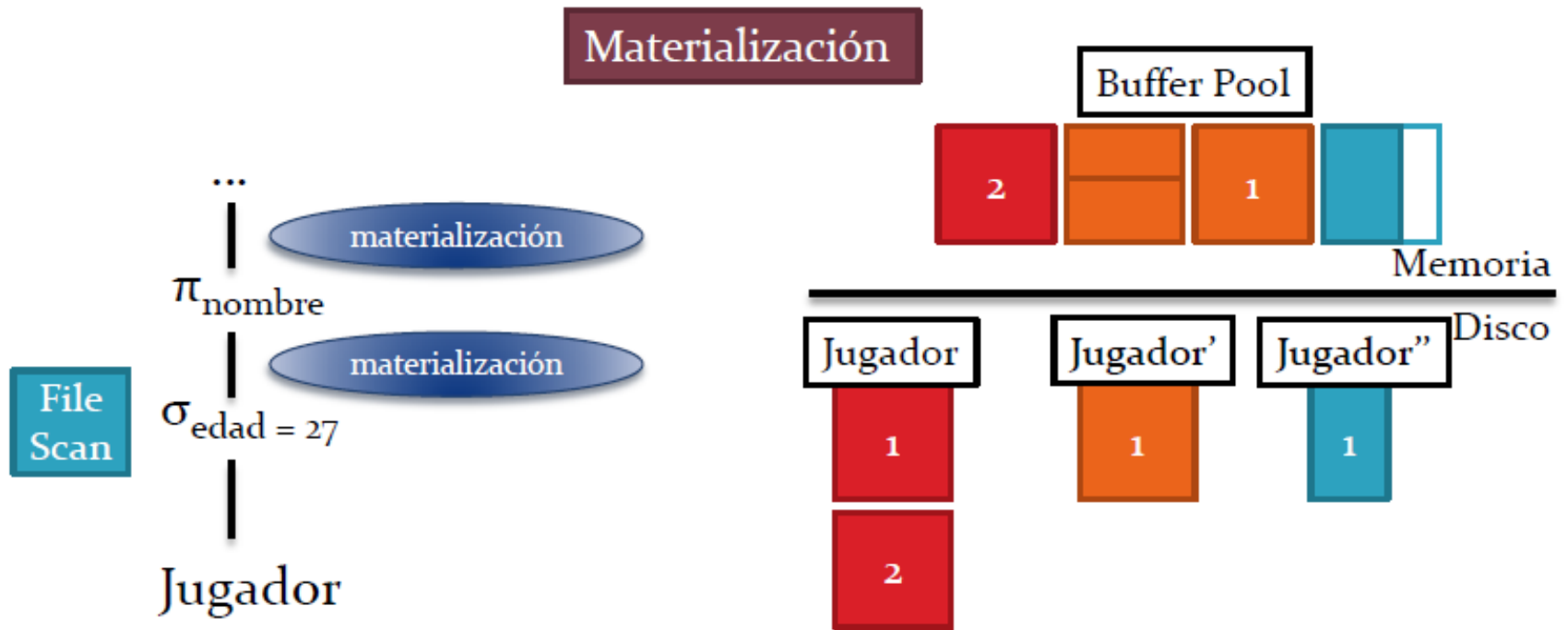


## ¿Cómo comparar planes?

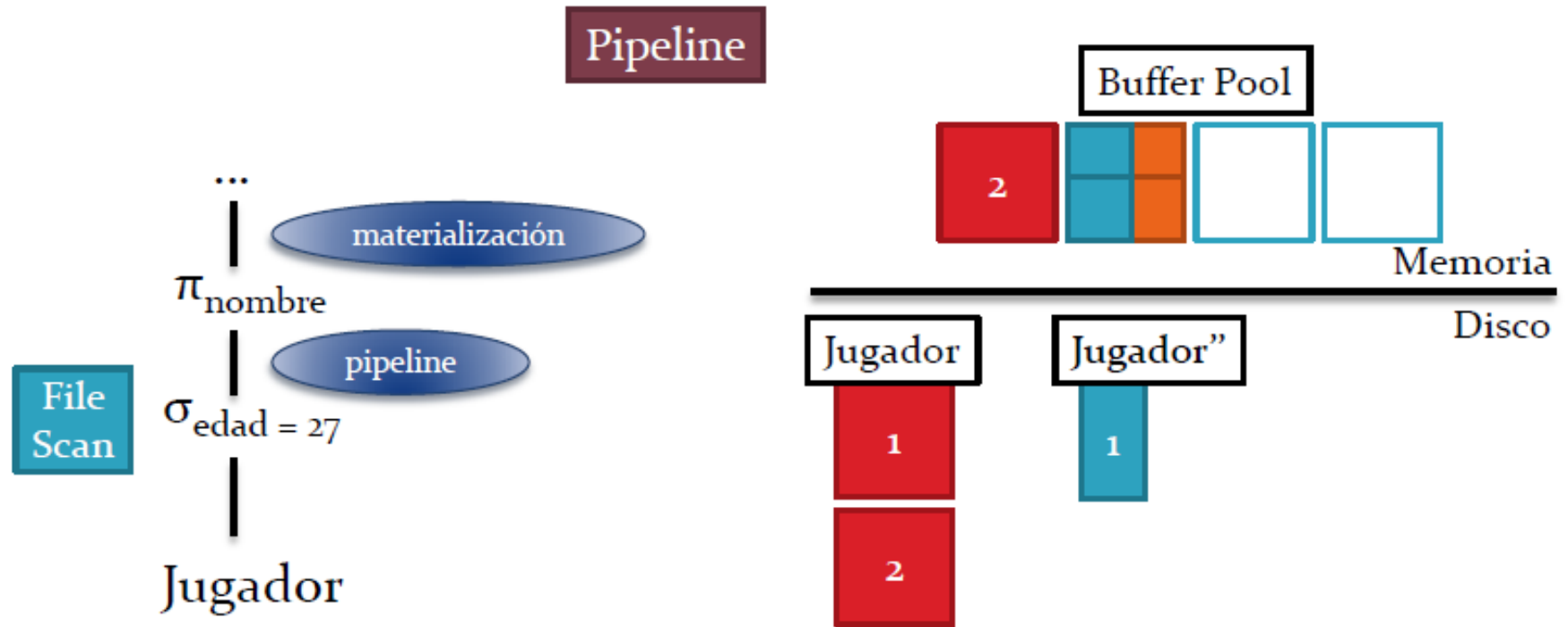
- Se define un **modelo de costos**
- El costo será expresado en **cantidad de accesos a disco** (lecturas + escrituras)
  - Predomina sobre tiempo de CPU
- El costo de un plan será una estimación
- Se elegirá al plan con menor costo estimado

- Detalles:
  - No asumiremos nada sobre el *Buffer Manager*, por lo que siempre consideraremos que un pedido de lectura o escritura significa acceder a disco
  - En un bloque de R, hay sólo tuplas de R (y no de otras relaciones)
  - Las tuplas de R se guardan enteras en un bloque (por ejemplo, no puede haber mitad en el bloque i y la otra mitad en el bloque i+1)
  - Siempre asumiremos *peor caso*

# Materialización

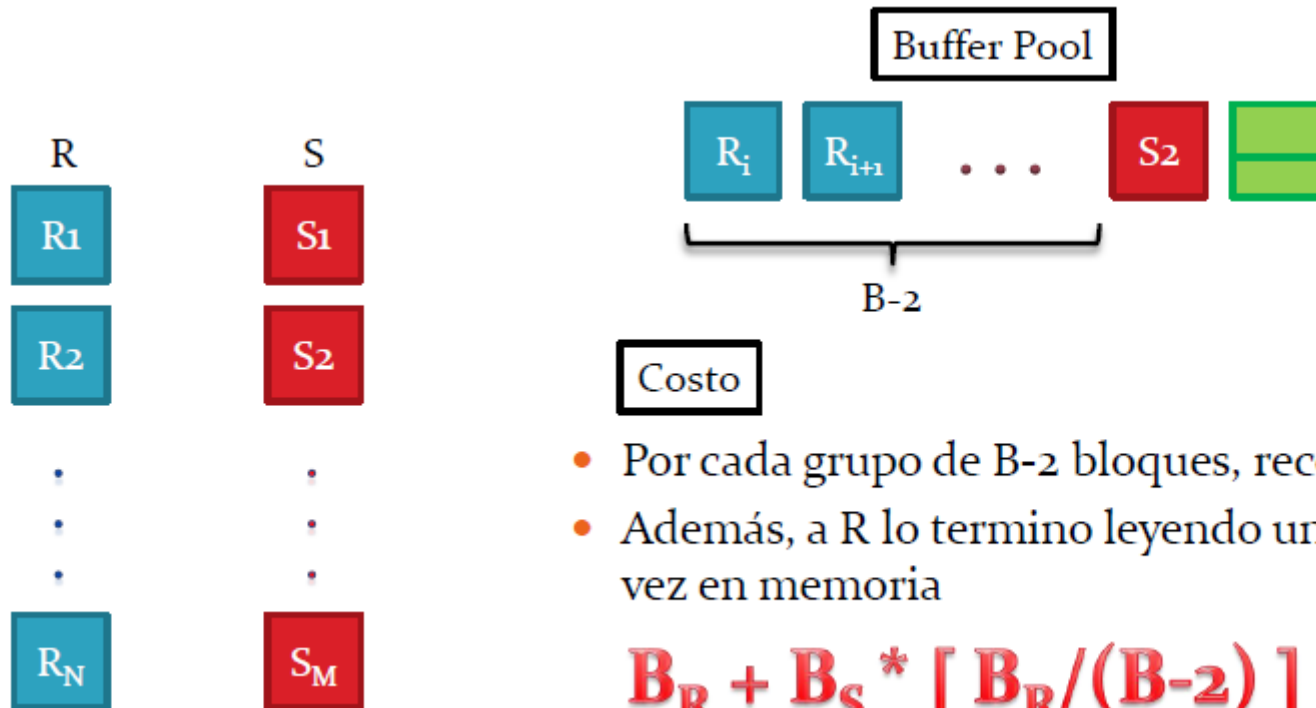


# Pipeline



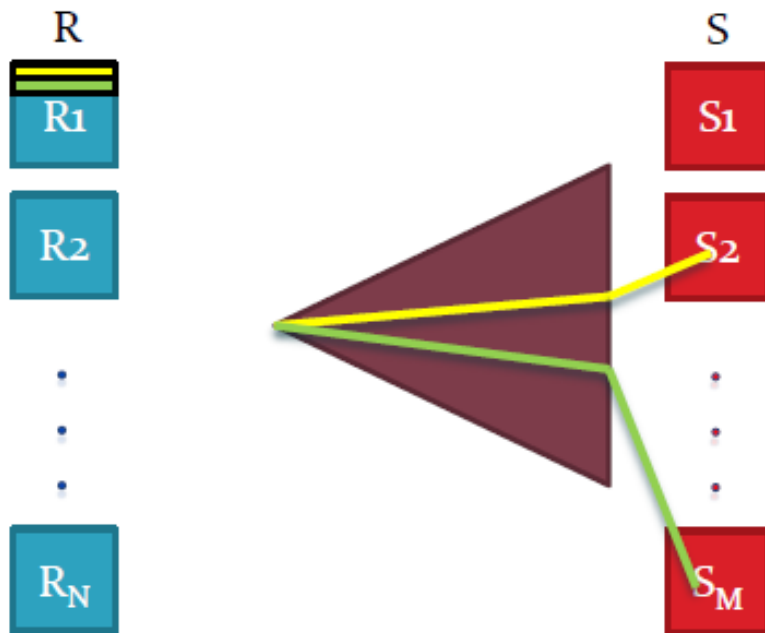
# Block Nested Loops Join (BNLJ)

B bloques de memoria



# Index Nested Loops Join (INLJ)

Índice I sobre S



Costo

- Por cada tupla de R, hago una búsqueda en el índice I
- Además, a R lo termino recorriendo una sola vez en memoria

$$B_R + T_R * \text{("costo índice")}$$



Dependerá del tipo de índice que exista



# Sort Merge Join (SMJ)

B bloques de memoria

| R    |     |
|------|-----|
| 1    | A   |
| 2    | B   |
| ...  | ... |
| 1000 | ABC |

| S   |     |
|-----|-----|
| 1   | x   |
| 1   | y   |
| ... | ... |
| 100 | xyz |

Costo

- Se ordenan R y S (si alguno ya está ordenado, este costo no se considera)
- Se hace el *merge* entre R y S ordenados

$$(\lceil \log_{B-1} [B_R/B] \rceil + 1) * 2B_R + (\lceil \log_{B-1} [B_S/B] \rceil + 1) * 2B_S + B_R + B_S$$