



# Fundamentos de Organización de Datos

**Curso 2024**

Profesores Rodolfo Bertone y Pablo Thomas

# La materia

## ► Clases

- Teóricas
- Explicaciones de Prácticas (donde se presentan ejemplos)
- Prácticas
- Se utilizará la plataforma  
<https://asignaturas.info.unlp.edu.ar> (Moodle Facultad)

## ► Para aprobar la cursada

- Un Parcial
- Dos recuperatorios

# Contenidos básicos

- ▶ **Persistencia de Datos**
  - ▶ Archivos
- ▶ **Acceso a datos, performance en el acceso**
  - ▶ Acceso secuencial indizado (árboles)
  - ▶ Acceso Directo (Hashing)

# La Materia



# Horarios de Práctica

5

- ▶ **Turno 1**

- ▶ Martes de 11 a 14

- ▶ **Turno 2**

- ▶ Martes de 18 a 21

- ▶ Comienza semana del 11/03

- ▶ **Inscripción en Turnos** <https://fod.info.unlp.edu.ar/>

**el viernes 08/03/2024 a partir de las 12 hs**

# Fechas de Examen

## **Fechas tentativas de parcial:**

1º Fecha → Martes 04-06-2024

2º Fecha → Martes 25-06-2024

3º Fecha → Martes 16-07-2024

## **Fecha tentativa de evaluación teórica:**

Semana del 08-07-2024

# Evaluación teórica - Cursada 2024

## **Semana del 8 de Julio de 2024**

- ▶ Podrán acceder al examen todos los alumnos que se encuentren inscriptos en la asignatura.
- ▶ Se realizará un examen teórico.
- ▶ La aprobación del examen es con nota 5 o superior
- ▶ El examen NO tiene recuperatorio.
- ▶ Se deberá aprobar la cursada durante el semestre en que rinde los exámenes teóricos.
- ▶ Deberá anotarse para pasar el final de la asignatura como máximo hasta la mesa de marzo de 2025 inclusive.

Semana	Teoría	Práctica
04/03	Introducción. Archivos, Algorítmica Básica	Sin Actividad
11/03	Archivos, Algorítmica Básica, Algorítmica Clásica	Archivos Algorítmica Básica
18/03	Archivos, Algorítmica Clásica  Proceso de Baja en Archivos	Archivos Algorítmica Básica
25/03	Semana Santa	Archivos Algorítmica Clásica
01/04	Archivos con Registros de Longitud Variable	Archivos Algorítmica Clásica
08/04	Búsqueda de información en Archivos. Indices	Archivos Algorítmica Clásica Bajas, Registros de Longitud Variable
15/04	Arboles Binarios, AVL. Introducción a Arboles B	Bajas, Registros de Longitud Variable
22/04	Arboles B, Creación Búsqueda Eliminación, Performance (solo el jueves)	Arboles,
29/04	Arboles B *	Arboles
06/05	Arboles B+	Arboles
13/05	Hashing	Hashing
20/05	Hashing	Hashing
27/05	Simulacro de examen teórico	Consulta
03/06	Consultas	Primer Parcial
10/06	Consultas	Consulta
17/06	Consultas	Muestra de examen, consulta
24/06	Consultas	Recupetario
01/07	Consultas	Muestra de examen, consulta
08/07	Examen teórico	Recuperatorio

# Para el Redictado FOD (segundo semestre)

- Podrán acceder al redictado de FOD:
  - aquel alumno que no apruebe los tres temas de la materia y se presente como mínimo en dos de los tres fechas de parcial y
  - En cada uno de los tres temas de la materia deben tener, en al menos una de las fechas presentadas, una calificación diferente a blanco (es decir, demuestre intención de resolver el tema)

# Bibliografia

- ▶ Introducción a las Bases de Datos. Conceptos Básicos (Bertone, Thomas)
- ▶ Estructuras de Archivos (Folk-Zoellick)
- ▶ Files & Databases: An Introduction (Smith-Barnes)
- ▶ Fundamentos de Bases de Datos (Korth Silvershatz)



11

# Fundamentos de Organización de Datos

Clase 1

# Agenda

Conceptos  
básicos de  
BD

Archivos

- Definiciones
- Características

- Introducción
- Operatoria básica

# Conceptos básicos

## Qué es una Base de Datos?

Es una colección de datos relacionados.

Colección de **archivos** diseñados para servir a múltiples aplicaciones

Un dato representa hechos conocidos que pueden registrarse y que tienen un resultado implícito.

# Conceptos básicos

## Propiedades implícitas de una BD

Una BD representa algunos aspectos del mundo real, a veces denominado Universo de Discurso.

Una BD es una colección coherente de datos con significados inherentes. Un conjunto aleatorio de datos no puede considerarse una BD. O sea los datos deben tener cierta lógica.

Una BD se **diseña**, construye y completa de datos para un propósito específico. Está destinada a un grupo de usuarios concretos y tiene algunas aplicaciones preconcebidas en las cuales están interesados los usuarios

Una BD está sustentada físicamente en **archivos** en dispositivos de almacenamiento persistente de datos

# Archivos

## Definiciones

- Colección de registros guardados en almacenamiento secundario
- Colección de datos almacenados en dispositivos secundarios de memoria
- Colección de registros que abarcan entidades con un aspecto común y originadas para algún propósito particular

# Archivos → algunos conceptos

## Hardware

Almacenamiento  
primario

Almacenamiento  
secundario (DR)

Acceso a RAM  
vs. Acceso a DR

Platos  
Superficies  
Pistas  
Sectores  
Cilindros

Comparaciones

# Archivos → organización

## Secuencia de bytes

- No se puede determinar fácilmente comienzo y fin de cada dato.
- Archivos de texto

## Registros y Campos

- **Campo:** Unidad más pequeña, lógicamente significativa de un archivo
- **Registro:** Conjunto de campos agrupados que definen un elemento del archivo

## Archivos → Acceso

**Secuencial Físico:** acceso a los registros uno tras otro y en el orden físico en el que están guardados

**Secuencial indizado (lógico):** acceso a los registros de acuerdo al orden establecido por otra estructura

- Ej: una guía telefónica, o índice temático del un libro

**Directo:** se accede a un registro determinado sin necesidad de haber accedido a los predecesores

### De acuerdo a la forma de acceso

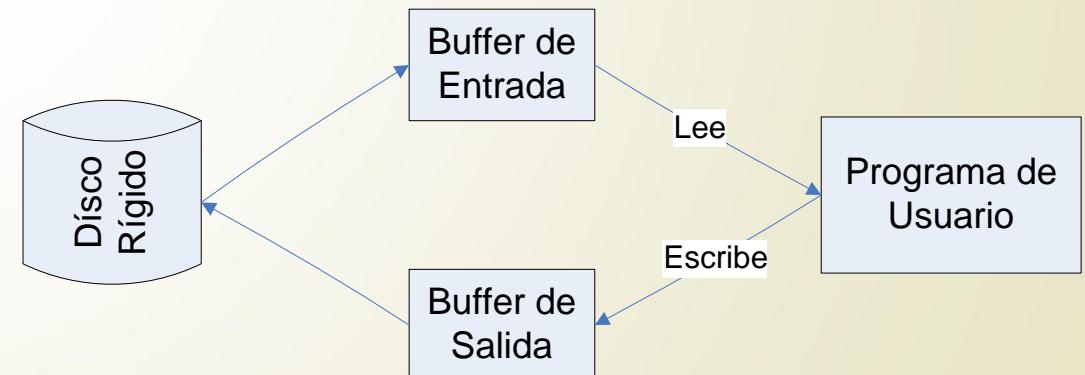
- **Serie** cada registro es accesible solo luego de procesar su antecesor, simples de acceder (acceso secuencial físico)
- **Secuencial** los registros son accesibles en orden de alguna clave (acceso secuencial *indizado*/ secuencial lógico)
- **Directo** se accede al registro deseado (acceso directo)

# Archivos

## ► Buffers

Memoria intermedia entre un archivo y un programa, donde los datos residen provisoriamente hasta ser almacenados definitivamente en memoria secundaria o donde los datos residen una vez recuperados de dicha memoria secundaria.

- Los buffers ocupan lugar en RAM
- SO encargado de manipular los buffers
- Como trabaja?



# Archivos → Operaciones básicas

## Dos niveles

- Físico (almacenamiento secundario)
- Lógico (dentro del programa)
  - Operaciones
    - Crear
    - Abrir
    - Read/Write
    - Eof
    - Seek(localización)

# Archivos → Declaraciones

Utilizaremos la notación Pascal

Declaración de archivos

- Variable

- Var archivo: file of Tipo\_de\_dato;

- Tipo

- Type archivo: file of Tipo\_de\_dato;

- Var arch: archivo

# Archivos – Operaciones Básicas

## ► Ejemplos

```
Type emple = record
    nombre: string [20];
    direccion: string [40];
    edad: integer;
    end;
numero = file of integer;
empleado = file of emple;
Var arch_num: numero;
Var arch_emp: empleado,
```

# Archivos → Operaciones Básicas

## ► Relación con el sistema operativo

- Se tiene que establecer la correspondencia entre el nombre físico y nombre lógico

### **Assign ( n\_lógico, N\_físico)**

#### ► Ejemplo

```
Program archivos;
...
Type emple = record
    nombre: string [20];
    direccion: string [40];
    edad: integer;
end;
numero = file of integer;
empleado = file of emple;
Var arch_num:numero;
Var arch_emp:empleado;
...
Begin
...
Assign( arch_num, 'numeros.dat' )
Assign( arch_emp, 'Personas_empleados.dat' )
...
End;
```

# Archivos – Operaciones Básicas

```
Rewrite (nombre_logico);
```

- De solo escritura (creación )

```
Reset (nombre_logico);
```

- Lectura Escritura (apertura)

Nombre lógico representa una variable de tipo archivo sobre la que se realizó la asignación.

```
Close (nombre_logico);
```

- Cierre de archivo
- Esta instrucción indica que no se va a trabajar más con el archivo. Significa poner una marca de EOF (end of file) al final del mismo.

# Archivos → Operaciones Básicas

Read(nombre\_logico, variable);

Write(nombre\_logico, variable);

Estas operaciones leen y/o escriben sobre los buffers relacionados a los archivos

No se realizan directamente sobre la memoria secundaria

En ambos casos la variable debe ser del mismo tipo que los elementos que se declararon como parte del archivo

# Archivos → EJ 1 Crear un archivo

27

**Program Generar\_Archivo;**

```
type archivo = file of integer; {definición del tipo de dato para el archivo }
var arc_logico: archivo; {variable que define el nombre lógico del archivo}
nro: integer; {nro será utilizada para obtener la información de
teclado}
arc_fisico: string[12]; {utilizada para obtener el nombre físico del archivo
desde teclado}
```

**begin**

```
write( 'Ingrese el nombre del archivo:' );
read( arc_fisico ); { se obtiene el nombre del archivo}
assign( arc_logico, arc_fisico );
rewrite( arc_logico ); { se crea el archivo }
read( nro ); { se obtiene de teclado el primer valor }
while nro <> 0 do begin
    write( arc_logico, nro ); { se escribe en el archivo cada número }
    read( nro );
end;
close( arc_logico ); { se cierra el archivo abierto oportunamente con la instrucción
rewriteFOP - Clase 1
end.
```

# Archivos → Operaciones adicionales

`EOF( nombre_logico); (función)`

- Fin de archivo
  - Como trabaja?
  - Hay que preguntar primero!!!

`FileSize(nombre_logico); (función)`

- Tamaño del archivo

`FilePos( nombre_logico); (función)`

- Posición dentro del archivo

`Seek( nombre_logico, posición); (Procedimiento)`

- Ir a una posición del archivo
- La posición se cuenta siempre desde el comienzo del archivo
- El primer lugar es el cero .

# Archivos → EJ 2 Presentar en pantalla el archivo generado en ej1

29

```
Procedure Recorrido(var arc_logico: archivo );
  var  nro: integer;  { para leer elemento del archivo}
begin
  reset( arc_logico ); {archivo ya creado, para operar debe abrirse como de lect/escr}
  while not eof( arc_logico) do begin
    read( arc_logico, nro ); {se obtiene elemento desde archivo }
    write( nro );           {se presenta cada valor en pantalla}
  end;
  close( arc_logico );
end;
```

## Archivos → Ej 3 Modificación de Datos de un archivo

- ▶ Este caso involucra un archivo de datos previamente generado y consiste en cambiar sus datos.
- ▶ El archivo debe ser recorrido desde su primer elemento y hasta el último, siguiendo un procesamiento secuencial

{declaración de los tipos de datos necesarios para el problema.  
Esta declaración se hace efectiva en el programa principal que  
tiene al proceso Actualizar como uno de sus módulos }

```
Type registro = record
    Nombre: string[20];
    Direccion: string[20];
    Salario: real;
End;
Empleados = file of registro;
```

## Archivos → Ej 3 Modificación de Datos de un archivo (continuación)

```
Procedure actualizar (Var Emp:empleados); {se recibe como parám.  
por referencia}  
var E: registro;  
begin  
    Reset( Emp );  
    while not eof( Emp ) do begin  
        Read( Emp, E );  
        E.salario:=E.salario * 1.1;  
        Seek( Emp, filepos(Emp) -1 );  
        Write( Emp, E );  
    end;  
    close( Emp );  
end;
```



# Fundamentos de Organización de Datos

1

Clase 2

# Agenda

Algoritmia

- Básica
- Clásica

Básica

- Agregar nuevos elementos

Clásica

- Actualización
- Merge
- Corte de Control

# Archivos - Algorítmica Clásica

Operaciones usuales a resolver utilizando archivos

- Agregar nuevos elementos
- Actualizar un archivo maestro con uno o varios archivos detalles
- Corte de control
- Merge de varios archivos

Los discutiremos en las siguientes transparencias

# Archivos → Ej 4 Agregar Datos a un archivo existente

4

- ▶ Se procesa un solo archivo
- ▶ Ya tiene información
- ▶ Se le incorporan datos nuevos
- ▶ El proceso muestra como se hace

```
Procedure agregar (Var Emp: Empleados);  
  var E: registro;  
begin  
  reset( Emp );  
  seek( Emp, filesize(Emp) );  
  leer( E );  
  while E.nombre <> ' ' do begin  
    write( Emp, E );  
    leer( E );  
  end;  
  close( Emp );  
end;
```

# Archivos → Actualización Maestro Detalle

Este problema involucra utilizar al mismo tiempo varios archivos de datos.

- Se denomina maestro al archivo que resume un determinado conjunto de datos.
- Se denomina detalle al que agrupa información que se utilizará para modificar el contenido del archivo maestro.
- En general
  - Un maestro
  - N detalles.

Consideraciones del proceso (precondiciones)

- Ambos archivos (maestro y detalle) están ordenados por el mismo criterio
- En el archivo detalle solo aparecen empleados que existen en el archivo maestro
- Cada empleado del archivo maestro a lo sumo puede aparecer una vez en el archivo detalle

# Archivos → Ej 5 Actualizar Un Maestro con Un detalle

```
program actualizar;
6 type emp = record
    nombre: string[30];
    direccion: string[30];
    cht: integer;
  end;
  detalle = file of e_diario; { archivo que contiene la información diaria }
  maestro = file of emp;      { archivo que contiene la información completa }

var
  regm: emp;    regd: e_diario;  mael: maestro; det1: detalle;
begin
  assign (mael, 'maestro');
  assign (det1, 'detalle');
  {proceso principal}
  reset (mael);  reset (det1);
  while (not eof(det1)) do begin
    read(mael, regm);
    read(det1,regd);
    while (regm.nombre <> regd.nombre) do
      read (mael,regm);
    regm.cht := regm.cht + regd.cht;
    seek (mael, filepos(mael)-1);
    write(mael,regm);
  end; FOD - Clase 2
end;
end.
```

## Precondiciones del ejemplo

- Ambos archivos (maestro y detalle) están ordenados por código del producto
- En el archivo detalle solo aparecen productos que existen en el archivo maestro
- Cada producto del maestro puede ser, a lo largo del día, vendido más de una vez, por lo tanto, en el archivo detalle pueden existir varios registros correspondientes al mismo producto

## Archivos → Ej 6 Un Maestro Un detalle Nuevas condiciones (Cont)

8

```
program actualizar;
const valoralto='9999';
type str4 = string[4];
prod = record
  cod: str4;
  descripcion:
  string[30];
  pu: real;
  cant: integer;
end;
v_prod = record
  cod: str4;
  cv: integer;
{cantidad vendida}
end;
detalle = file of v_prod;
maestro = file of prod;
var
  regm: prod;
  regd: v_prod;
  mael: maestro;
  det1: detalle;
  total: integer;

begin
  assign (mael, 'maestro');
  assign (det1, 'detalle');
  {proceso principal}
  reset (mael);  reset (det1);
  while (not eof(det1)) do begin
    read(mael, regm);
    read(det1, regd);
    while (regm.cod <> regd.cod) do
      read (mael,regm);
    while not eof(det1) and (regm.cod = regd.cod) do
      begin
        regm.cant := regm.cant - regd.cv;
        read (det1,regd);
      end;
    if not eof(det1)
      seek( det1, filepos(det1)-1)
    seek (mael, filepos(mael)-1);
    write(mael,regm);
  end;
end.
```

## Archivos → Ej 6 Un Maestro Un detalle Nuevas condiciones (Cont)

9

```
procedure leer (var archivo:detalle; var dato:v_prod);
begin
    if (not eof(archivo))
        then read (archivo,dato)
        else dato.cod := valoralto;
    end;
begin
    assign (mael, 'maestro');  assign (det1, 'detalle');
    reset (mael);  reset (det1);
    leer(det1,regd); {se procesan todos los registros del archivo det1}
    while (regd.cod <> valoralto) do begin
        read(mael, regm);
        while (regm.cod <> regd.cod) do
            read (mael,regm);
        { se procesan códigos iguales }
        while (regm.cod = regd.cod) do begin
            regm.cant := regm.cant - regd.cv;
            leer(det1,regd);
        end;
        {reubica el puntero}
        seek(mael, filepos(mael)-1);
        write(mael,regm);
    end;
end;
```

# Archivos → Ej 7 Un Maestro N detalle

El problema siguiente generaliza aún más el problema anterior

El maestro se actualiza con tres archivos detalles

Los archivos detalle están ordenados de menor a mayor

Condiciones de archivos iguales, misma declaración de tipos del problema anterior

# Archivos → Ej 7 Un Maestro N detalle (cont)

11

```
var
  regm: prod;  min, regd1, regd2,regd3: v_prod;
  mael: maestro;  det1,det2,det3: detalle;

procedure leer (var archivo: detalle; var dato:v_prod);
begin
  if (not eof(archivo))
    then read (archivo,dato)
    else dato.cod := valoralto;
end;

procedure minimo (var r1,r2,r3: v_prod; var min:v_prod);
begin
  if (r1.cod<=r2.cod) and (r1.cod<=r3.cod) then begin
    min := r1;
    leer(det1,r1)
  end
  else if (r2.cod<=r3.cod) then begin
    min := r2;
    leer(det2,r2)
  end
  else begin
    min := r3;
    leer(det3,r3)
  end;
end;
```

# Archivos → Ej 7 Un Maestro N detalle (cont)

12

```
begin
    assign (mael, 'maestro') ;      assign (det1, 'detalle1') ;
    assign (det2, 'detalle2') ;      assign (det3, 'detalle3') ;
    reset (mael) ;    reset (det1) ; reset (det2) ; reset (det3) ;
    leer(det1, regd1); leer(det2, regd2); leer(det3, regd3);
    minimo(regd1, regd2, regd3, min) ;
    while (min.cod <> valoralto) do begin
        read(mael,regm) ;
        while (regm.cod <> min.cod) do
            read(mael,regm) ;
            while (regm.cod = min.cod ) do begin
                regm.cant:=regm.cant - min.cantvendida;
                minimo(regd1, regd2, regd3, min) ;
            end;
            seek (mael, filepos(mael)-1) ;
            write(mael,regm) ;
        end;
    end.
```

# Archivos → Corte de control

El problema consiste en la generación de reportes

- Es un problema clásico en el manejo de BD.
- Si bien los DBMS lo manejan diferente, veremos la algorítmica clásica de los mismos
- Precondiciones
  - El archivo se encuentra ordenado por provincia, partido y ciudad

Provincia: xxxx
Partido: yyyy
Ciudad # Var. # Muj. Desocupados
aaa ..... .... .....
bbb ..... .... .....
ccc ..... .... .....
Total Partido .....
Partido: zzzz
Ciudad # Var. # Muj. Desocupados
... ..... .... .....
Total Partido .....
Total Provincia: .....
Provincia: qqqq

# Archivos → Ej 8 Corte de control

14

```
program Corte_de_Control;
  const valoralto='zzzz';
  type str10 = string[10];
  prov = record
    provincia, partido, ciudad: str10;
    cant_varones, cant_mujeres, cant_desocupados : integer;
  end;
  instituto = file of prov;

  var regm: prov;
  inst: instituto;
  t_varones, t_mujeres, t_desocupados: integer;
  t_prov_var, t_prov_muj, t_prov_des: integer;
  ant_prov, ant_partido : str10;

procedure leer (var archivo:instituto; var dato:prov);
begin
  if (not eof( archivo ))
    then read (archivo,dato)
  else dato.provincia := valoralto;
end;
```

FOD - Clase 2

# Archivos → Ej 8 Corte de control

15

```
begin
    assign (inst, 'censo' );  reset (inst); leer (inst, regm);
    writeln ('Provincia: ', regm.provincia); writeln ('Partido: ', regm.partido); writeln
    ('Ciudad','Varones','Mujeres','Desocupados');
    { se inicializan los contadores para el total del partido para varones, mujeres y desocupados }
    t_varones := 0;      t_mujeres := 0;      t_desocupados := 0;
    { se inicializan los contadores para el total de cada provincia }
    t_prov_var := 0;    t_prov_muj := 0;    t_prov_des := 0;
    while ( regm.provincia <> valoralto)do begin
        ant_prov := regm.provincia; ant_partido := regm.partido;
        while (ant_prov=regm.provincia) and (ant_partido=regm.partido) do begin
            write (regm.ciudad, regm.cant_varones, regm.cant_mujeres,regm.cant_desocupados);
            t_varones := t_varones + regm.cant_varones; t_mujeres := t_mujeres + regm.cant_mujeres;
            t_desocupados := t_desocupados + regm.cant_desocupados;
            leer (inst, regm);
        end;
        writeln ('Total Partido: ', t_varones,t_mujeres,t_desocupados);
        t_prov_var := t_prov_var + t_varones; t_prov_muj := t_prov_muj + t_mujeres;
        t_prov_des := t_prov_des + t_desocupados;
        t_varones := 0; t_mujeres := 0; t_desocupados := 0;
        ant_partido := regm.partido;
        if (ant_prov <> regm.provincia) then begin
            writeln ('Total Provincia',t_prov_var,t_prov_muj,t_prov_des);
            t_prov_var := 0; t_prov_muj := 0; t_prov_des := 0;
            writeln ('Provincia: ', regm.provincia);
        end;
        writeln ('Partido: ', regm.partido);
    end;
end.
```

# Archivos - Merge

- ▶ Involucra archivos con contenido similar, el cual debe resumirse en un único archivo.
- ▶ **Precondiciones:**
  - ▶ Todos los archivos detalle tienen igual estructura
  - ▶ Todos están ordenados por igual criterio
- ▶ **Primer ejemplo:**
  - ▶ CADP inscribe a los alumnos que cursarán la materia en tres computadoras separadas. C/U de ellas genera un archivo con los datos personales de los estudiantes, luego son ordenados físicamente por otro proceso. El problema que tienen los JTP es generar un archivo maestro de la asignatura
- ▶ **Precondiciones**
  - ▶ El proceso recibe tres archivos con igual estructura
  - ▶ Los archivos están ordenados por nombre de alumno
  - ▶ Un alumno solo aparece una vez en el archivo
- ▶ **Postcondición**
  - ▶ Se genera el archivo maestro de la asignatura ordenado por nombre del alumno

# Archivos – Ej 9: Merge 3 archivos

```
program union_de_archivos;
17 const valoralto = 'zzzz';
type str30 = string[30];
str10 = string[10];
alumno = record
  nombre: str30;
  dni: str10;
  direccion: str30;
  carrera: str10;
end;
detalle = file of alumno;
var min,regd1,regd2,regd3: alumno;
det1,det2,det3,maestro : detalle;
procedure leer (var archivo:detalle; var
dato:alumno);
begin
  if (not eof( archivo ))
    then read (archivo, dato)
  else dato.nombre := valoralto;
end;
```

FOD - Clase 2

```
procedure minimo (var r1,r2,r3:alumno; var
min:alumno);
begin
  if (r1.nombre<r2.nombre) and
(r1.nombre<r3.nombre) then begin
    min := r1;
    leer(det1,r1)
  end
  else if (r2.nombre<r3.nombre) then
begin
  min := r2;
  leer(det2,r2)
end
else begin
  min := r3;
  leer(det3,r3)
end;
```

# Archivos – Ej 9: Merge 3 archivos

```
begin
    assign (det1, 'det1');
    assign (det2, 'det2');
    assign (det3, 'det3');
    assign (maestro, 'maestro');
    rewrite (maestro);
    reset (det1);      reset (det2);      reset (det3);

    leer(det1, regd1); leer(det2, regd2); leer(det3, regd3);
    minimo(regd1, regd2, regd3, min);

    { se procesan los tres archivos }
    while (min.nombre <> valoralto) do
        begin
            write (maestro,min);
            minimo (regd1,regd2,regd3,min);
        end;
        close (maestro);
    end.
```

# Archivos – Ej 10: Merge 3 archivos con repetición

- ▶ Los vendedores de cierto comercio asientan las ventas realizadas .....
- ▶ Precondiciones
  - ▶ Similar al anterior
  - ▶ Cada vendedor puede realizar varias ventas diarias

```
program union_de_archivos II;
const valorAltó = '9999';
type str4 = string[4];
str10 = string[10];
vendedor = record
    cod: str4;
    producto: str10;
    montoVenta: real;
end;
ventas = record
    cod: str4;
    total: real;
end;
detalle = file of vendedor;
maestro = file of ventas;
```

```
var min, regd1, regd2, regd3: vendedor;
    det1, det2, det3: detalle;
    mael: maestro;
    regm: ventas;
    aux: str4;
```

# Archivos – Ej 10: Merge 3 archivos con repetición

20

```
begin
    assign (det1, 'det1'); assign (det2, 'det2'); assign (det3, 'det3'); assign
    (mael, 'maestro');
    reset (det1);      reset (det2);      reset (det3);
    rewrite (mael);
    leer (det1, regd1);      leer (det2, regd2);      leer (det3, regd3);
    minimo (regd1, regd2, regd3,min);

    { se procesan los archivos de detalles }
    while (min.cod <> valoralto) do begin

        {se asignan valores para registro del archivo maestro}

        regm.cod := min.cod;
        regm.total := 0;
        {se procesan todos los registros de un mismo vendedor}
        while (regm.cod = min.cod ) do begin
            regm.total := regm.total+ min.montoVenta;
            minimo (regd1, regd2, regd3, min);
        end;

        { se guarda en el archivo maestro}
        write(mael, regm);
    end;
End;
```

FOD - Clase 2

# Archivos – Ej 11: Merge N archivos con repetición

- ▶ Los vendedores de cierto comercio asientan las ventas realizadas.....
- ▶ Precondiciones
  - ▶ Similar al anterior
  - ▶ Cada vendedor puede realizar varias ventas diarias
- ▶ Idem anterior con N archivos....

# Archivos – Ej 11: Merge N archivos con repetición

```
program union_de_archivos_III;
  const valoralto = '9999';
  type vendedor = record
    cod: string[4];
    producto: string[10];
    montoVenta: real;
  end;
  ventas = record
    cod: string[4];
    total: real;
  end;
  maestro = file of ventas;
  arc_detalle=array[1..100] of file of vendedor;
  reg_detalle=array[1..100] of vendedor;
var min: vendedor;
  deta: arc_detalle;
  reg_det: reg_detalle;
  mael: maestro;
  regm: ventas;
  i,n: integer;
```

# Archivos – Ej 11: Merge N archivos con repetición

23

```
procedure leer (var archivo:detalle; var dato:vendedor);
begin
  if (not eof( archivo ))
    then read (archivo, dato)
    else dato.cod := valoralto;
end;
procedure minimo (var reg_det: reg_detalle; var min:vendedor; var data:arc_detalle);
var i: integer;
begin
  { busco el mínimo elemento del
    vector reg_det en el campo cod,
    supongamos que es el índice i }
  min = reg_det[i];
  leer( data[i], reg_det[i] );
end;

begin
  Read(n)
  for i:= 1 to n do begin
    assign (data[i], 'det'+i);
    { ojo lo anterior es incompatible en tipos}
    reset( data[i] );
    leer( data[i], reg_det[i] );
  end;
  assign(ma1, 'maestro'); rewrite (ma1);
  minimo (reg_det, min, data);
```

# Archivos – Ej 11: Merge N archivos con repetición

24

```
{ se procesan los archivos de detalles }
while (min.cod <> valoralto) do
begin
{se asignan valores para registro del archivo maestro}
    regm.cod := min.cod;
    regm.total := 0;

{se procesan todos los registros de un mismo vendedor}
    while (regm.cod = min.cod ) do begin
        regm.total := regm.total+ min.montoVenta;
        minimo (regd1, regd2, regd3, min);
    end;

{ se guarda en el archivo maestro}
    write(mael, regm);

end;
```



# Fundamentos de Organización de Datos

1

Clase 3

# Agenda

Viaje del byte

Tipos de archivo

Claves

Eliminación

- Secuencia de Bytes
- Registros / campos longitud predecible
- Registros / campos sin longitud predecible

- Primaria
- Candidata
- Secundaria

- Recuperación de espacio
- Reg. Long Variable
- Eliminación

# Archivos → Introducción

La memoria primaria (RAM) es rápida y de simple acceso, pero su uso tiene algunas desventajas respecto al almacenamiento secundario:

- Capacidad limitada
- Mayor costo
- Es volátil

# Archivos → Introducción

**Almacenamiento secundario** necesita más tiempo para tener acceso a los datos que en RAM

- Su acceso es tan “lento” que es imprescindible enviar y recuperar datos con inteligencia
- Al buscar un dato, se espera encontrarlo en el primer intento (o en pocos)
- Si se buscan varios datos, se espera obtenerlos todos de una sola vez

La información está organizada en **archivos**

- **Archivo:** colección de bytes que representa información

# Archivos → Viaje de un Byte

Archivo Físico

Archivo  
Lógico

- Archivo que existe en almacenamiento secundario
  - Es el archivo tal como lo conoce el S.O. y que aparece en su directorio de archivos
- 
- Es el archivo, visto por el programa
  - Permite a un programa describir las operaciones a efectuarse en un archivo,
  - No se sabe cual archivo físico real se utiliza o donde esta ubicado

# Archivos – Viaje de un byte

- ▶ **Viaje de un byte** → No es sencillo
  - ▶ Escribir un dato en un archivo
    - ▶ Write ( archivo, variable) → ciclos para escribir
- ▶ **Quienes están involucrados**
  - ▶ Administrador de archivos
  - ▶ Buffer de E/S
  - ▶ Procesador de E/S
  - ▶ Controlador de disco

# Archivos – Viaje de un byte

**Administrador de archivos:** conjunto de programas del S.O. (capas de procedimientos) que tratan aspectos relacionados con archivos y dispositivos de E/S

- **En Capas Superiores:** aspectos lógicos de datos (tabla)
  - Establecer si las características del archivo son compatibles con la operación deseada (1)
- **En Capas Inferiores:** aspectos físicos (FAT)
  - Determinar donde se guarda el dato (cilíndro, superficie, sector) (2)
  - Si el sector está ubicado en RAM se utiliza, caso contrario debe traerse previamente. (3)

# Archivos – Viaje de un byte

**Buffers de E/S:** agilizan la E/S de datos.

- Manejar buffers implica trabajar con grandes grupos de datos en RAM , para reducir el acceso a almacenamiento secundario

**Procesador de E/S:** dispositivo utilizado para la transmisión desde o hacia almacenamiento externo. Independiente de la CPU. (3)

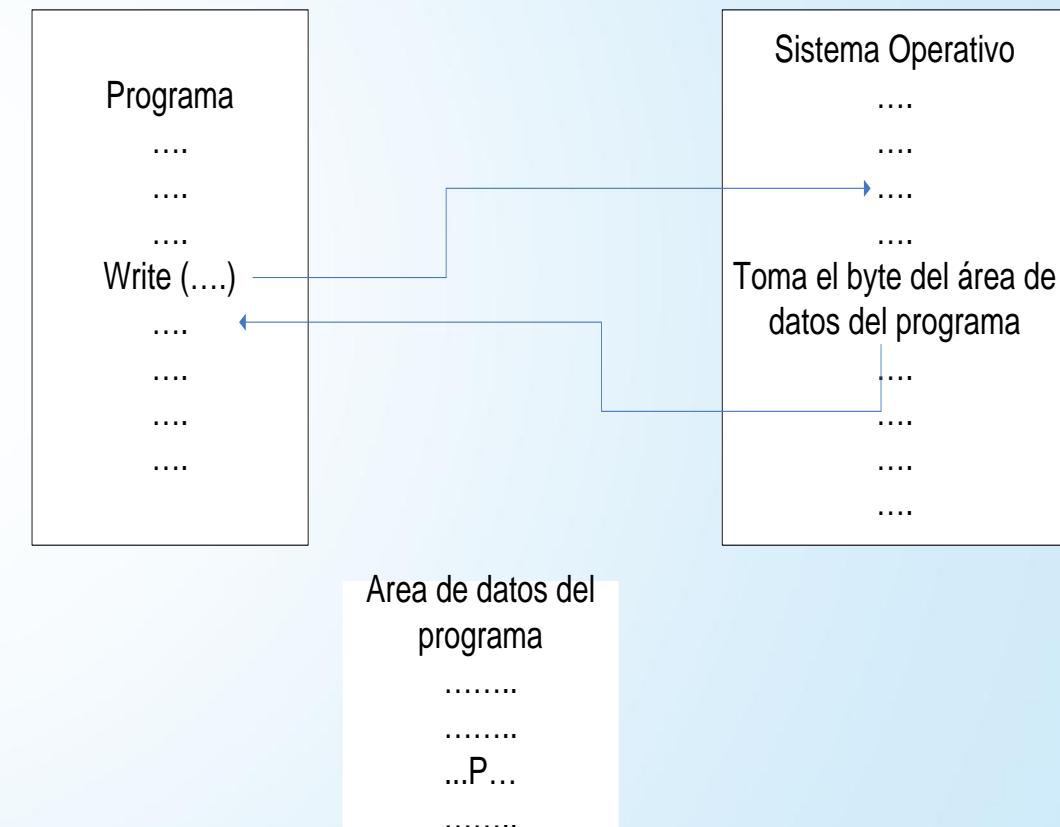
# Archivos – Viaje de un byte

**Controlador de disco:** encargado de controlar la operación de disco.

- Colocarse en la pista
- Colocarse en el sector
- Transferencia a disco

# Archivos – Viaje de un byte

- ▶ Qué sucede cuando un programa escribe un byte en disco?
  - ▶ Operación
    - ▶ Write(.....)
  - ▶ Veamos los elementos que se involucran en esta simple operación
  - ▶ Supongamos que se desea agregar un byte que representa el carácter 'P' almacenado en una variable c de tipo carácter, en un archivo denominado TEXTO que se encuentra en algún lugar del disco rígido.



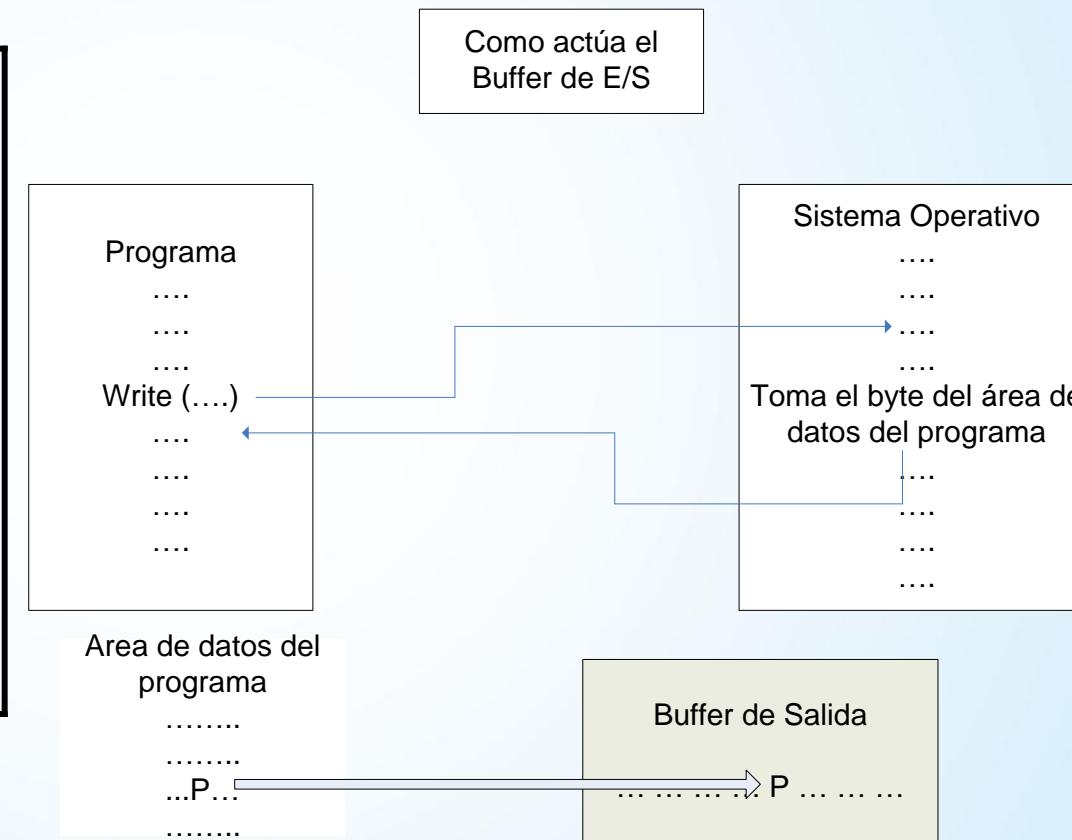
## Capas del protocolo de transmisión de un byte

- El Programa pide al **S.O.** escribir el contenido de una variable en un archivo
- El **S.O.** transfiere el trabajo al **Administrador de archivos**
- El **Adm.** busca el archivo en su tabla de archivos y verifica las características
- El **Adm.** obtiene de la FAT la ubicación física del sector del archivo donde se guardará el byte.
- El **Adm** se asegura que el sector del archivo está en un **buffer** y graba el dato donde va dentro del sector en el **buffer**
- El **Adm.** de archivos da instrucciones al **procesador de E/S** (donde está el byte en RAM y en qué parte del disco deberá almacenarse)
- El **procesador de E/S** encuentra el momento para transmitir el dato a disco, la CPU se libera
- El **procesador de E/S** envía el dato al **controlador de disco** (con la dirección de escritura)
- El **controlador** prepara la escritura y transfiere el dato bit por bit en la superficie del disco.

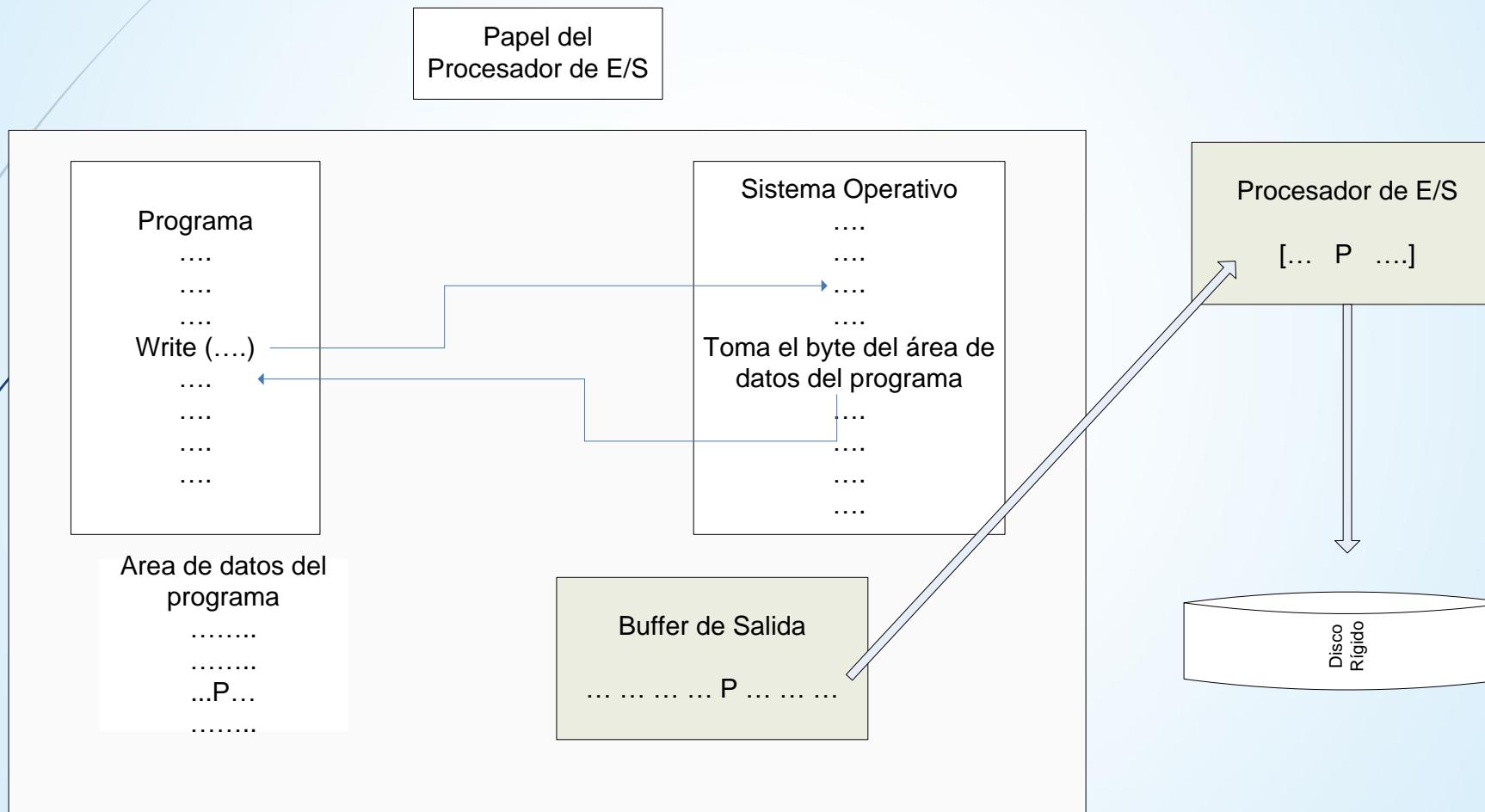
# Archivos – El viaje de un Byte

**Tabla**

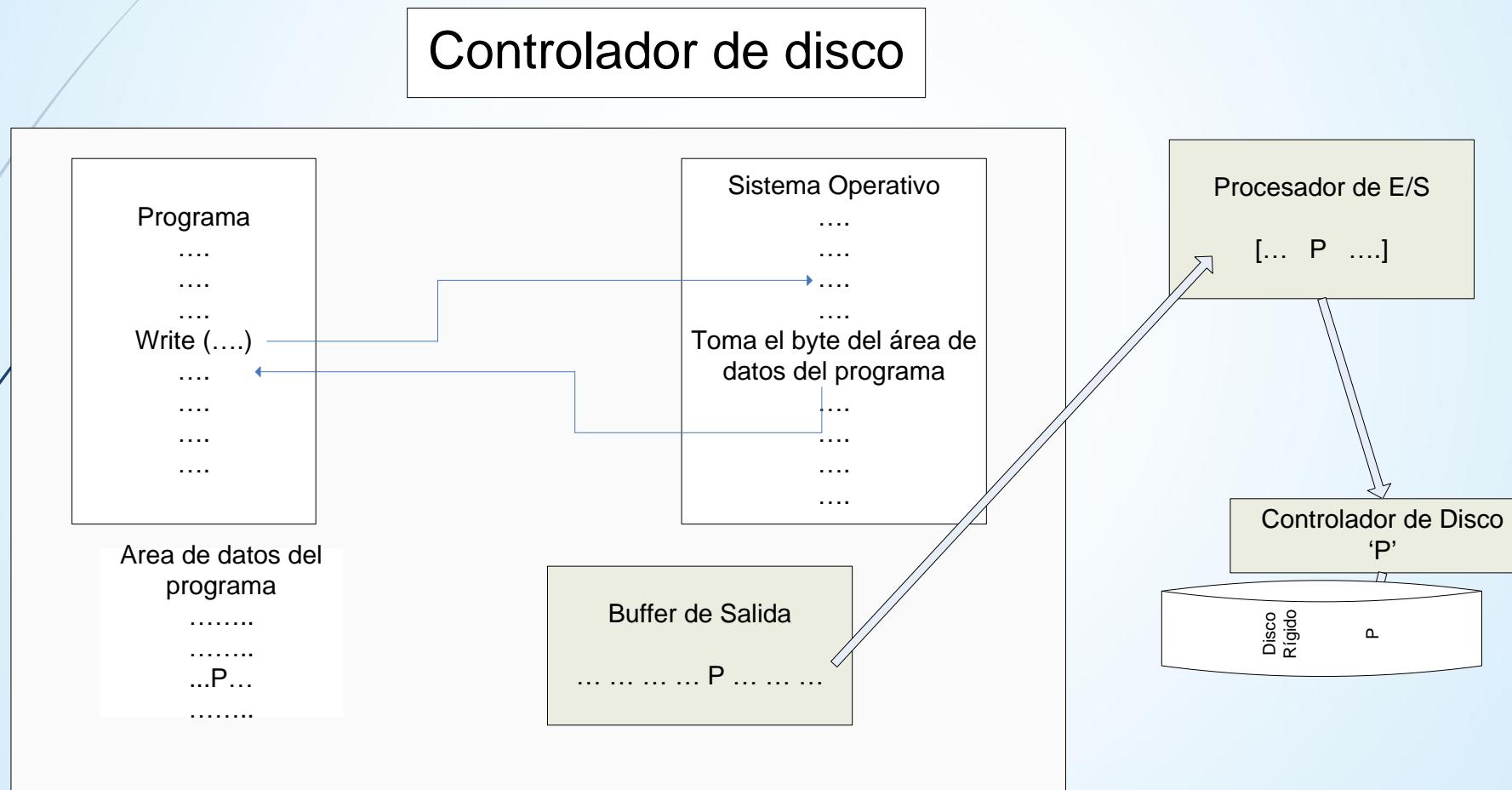
Nombre	Abrió	Acceso	Propietario	Protección
Archivo a	Perez	L/E	Gomez	prop:L/E otro: L/E
Archivo b	García	L	García	prop:L/E otro: L
Archivo c	Gomez	E	omez	prop:L/E otro: E



# Archivos – El viaje de un Byte



# Archivos – El viaje de un Byte



# Archivos → Tipos de Archivo

## Archivos como Secuencia de bytes

- No se puede determinar fácilmente comienzo y fin de cada dato.
- Ejemplo: archivos de texto

## Archivos estructurados

- **Registros**
  - Longitud fija o variable
- **Campos**
  - Longitud fija o variable

# Archivos → Tipos de Archivo

## Campos

- Unidad lógicamente significativa más pequeña de un archivo. Permite separar la información
- Identidad de campos: variantes, pro y contras.
  - **Longitud predecible** (long. Fija), desperdicio de espacio, si el tamaño es pequeño al agrandarlo se podría desperdiciar más espacio)
  - **Indicador de longitud** (al ppio de cada campo)
  - **Delimitador al final de cada campo** (carácter especial no usado como dato)

# Archivos → Tipos de Archivo

## Registros

- Organización de registros
- **Longitud predecible** (en cant. de bytes o cant. de campos)
  - Campos fijos o variables
- **Longitud variable**
  - **Indicador de longitud** (al comienzo, indica la cant. de bytes que contiene)
  - **Segundo archivo** (mantiene la info de la dirección del byte de inicio de cada registro)
  - **Delimitador** (carácter especial no usado como dato)
- **Estudio de casos:** ventajas y desventajas

# Archivos → Claves

## Clave

- Se concibe al Registro como la cantidad de información que se lee o escribe
- **Objetivo:** acceder sólo un registro específico
- Es conveniente identificar una registro con una llave o clave que se base en el contenido del mismo

# Archivos → Claves

## Clave

- Permite la identificación del registro
- Deben permitir generar orden en el archivo por ese criterio

## Únivoca / Primaria:

- Identifican un elemento particular dentro de un archivo

## Secundaria

- Generalmente no identifican un único registro

# Archivos → Claves

**Forma canónica:** forma estándar para una llave, puede derivarse a partir de reglas bien definidas.

- Representación única para la llave, ajustada a la regla
  - Ej: Clave sólo con letras mayúsculas y sin espacios al final.
- Al introducir un registro nuevo:
  - 1ro se forma una llave canónica para ese registro
  - 2do se la busca en el archivo. Si ya existe, y es unívoca → no se puede ingresar

# Archivos → Claves (performance)

## Estudio de performance

- Punto de partida para futuras evaluaciones
- Costo: acceso a disco, N° de comparaciones
- Caso promedio

# Archivos → Claves (performance)

## En el caso secuencial

- Mejor caso: leer **1** reg. , peor caso leer **n** registros
- Supongamos que tenemos 1000 registros, buscar uno en particular mejor caso 1, peor caso 1000, promedio 500, en realidad el mejor caso es 0, el buffer puede estar en memoria.
- Promedio:  $n/2$  comparaciones
- Es de  $O(n)$ , porque depende de la cantidad de registros
- **Lectura de Bloques de registros**
  - mejora el acceso a disco,
  - no varían las comparaciones.

# Archivos → Claves (performance)

## Acceso directo

- Permite acceder a un registro preciso
- Requiere una sola lectura para traer el dato [ O(1) ].
- Debe necesariamente conocerse el lugar donde comienza el registro requerido

## Número relativo de registro (NRR):

- Indica la posición relativa con respecto al principio del archivo
- Solo aplicable con registros de longitud fija
  - Ej. NRR 546 y longitud de cada registro 128 bytes ➔ distancia en bytes=  $546 * 128 = 69.888$

# Archivos → Claves (performance)

El acceso directo es preferible sólo cuando se necesitan pocos registros específicos, pero este método NO siempre es el más apropiado para la extracción de información.

- Ej. generar cheques de pago a partir de un archivo de registros de empleados.
  - Como todos los reg. se deben procesar → es más rápido y sencillo leer registro a registro desde el ppio. hasta el final, y NO calcular la posición en cada caso para acceder directamente.

# Archivos → diferentes visiones



Forma de  
acceso

Cantidad  
de cambios

## Forma de acceso

- **Serie:** cada registro es accesible solo luego de procesar su antecesor, simples de acceder
- **Secuencial:** los registros son accesibles en orden de alguna clave
- **Directo:** se accede al registro deseado

# Archivos → Tipos

## # de Cambios

- **Estáticos** -> pocos cambios
  - Puede actualizarse en procesamiento por lotes
  - No necesita de estructuras adicionales para agilizar los cambios
- **Volátiles** -> sometido a operaciones frecuentes:
  - Agregar / Borrar / Actualizar
  - Su organización debe facilitar cambios rápidos
  - Necesita estructuras adicionales para mejorar los tiempos de acceso

# Archivos → Operaciones

Altas

Bajas

Modificaciones

Consultas

Como influye  
registros de  
long. Fija y  
variable

# Archivos → eliminación

## Eliminar registros de un archivo

- **Baja Lógica**
- **Baja Física**
- Cuales son las diferencias?
- Cuales las ventajas y desventajas?

# Archivos → eliminación

- ▶ **Registro de longitud fija:** agregar o modificar, sin inconvenientes
- ▶ **Registros de longitud variable:** problemas
  - ▶ Ej: Intentar modificar un registro, tal que el modificado quede de mayor tamaño
  - ▶ Soluciones posibles:
    - ▶ Agregar los datos adicionales al final del archivo (con un vínculo al registro original) → complica el procesamiento del registro.
    - ▶ Reescribir el registro completo al final del archivo → queda un espacio vacío (desperdiaciado) en el lugar origen
    - ▶ La operación agregar no genera inconvenientes.
  - ▶ **Nos centralizaremos en la eliminación**

# Archivos → eliminación

## Baja Lógica

- Cualquier estrategia de eliminación de registros debe proveer alguna forma para reconocerlos una vez eliminados (**ejemplo: colocar una marca especial en el reg. eliminado**).
- Con este criterio se puede anular la eliminación fácilmente.
- Cómo reutilizar el espacio de registros eliminados ?
- Los programas que usan archivos deben incluir cierta lógica para ignorar los registros eliminados

# Archivos → eliminación

## Baja Física → Compactación

- Recuperar el espacio
- La forma más simple es copiar todo en un nuevo archivo a excepción de los registros eliminados → **Baja Física**
- Frecuencia
  - Tiempo (depende del dominio de aplicación)
  - Ante la necesidad de espacio
- Veremos el análisis de recuperación dinámica del almacenamiento

# Archivos → eliminación

## Aprovechamiento de espacio

- **Reg. longitud fija → es necesario garantizar:**
  - Marca especiales en los reg. borrados → **Baja Lógica**
  - **Reg. longitud variable → los nuevos elementos deben “caber” en el lugar**

# Archivos → eliminación

Recuperación del espacio para su reutilización cuando se agreguen registros

- **Búsqueda secuencial** -> usa las marcas de borrado.
  - Para agregar, se busca el 1º reg. eliminado. Si no existe se llega al final del archivo y se agrega allí.
  - Es muy lento para operaciones frecuentes.
- **Es necesario**
  - Una forma de saber de inmediato si hay lugares vacíos en el archivo
  - Una forma de saltar directamente a unos de esos lugares, en caso de existir

# Archivos → eliminación

## Aprovechamiento de espacio (reg. long. fija)

- **Recuperación de espacio con Lista o pilas (header)**
  - Lista encadenada de reg. disponibles.
  - Al insertar un reg. nuevo en un archivo de reg. con long. fija, cualquier registro disponible es bueno.
  - La lista NO necesita tener un orden particular, ya que todos los reg. son de long. fija y todos los espacios libres son iguales

# Archivos → eliminación

## Aprovechamiento de espacio (reg. long. fija)

- **Recuperación de espacio con Lista o pilas (header)**
  - Ej : en el encabezado estará NRR 4, el archivo tendrá
    - **alfa beta delta \* 6 gamma \* -1 epsilon**
  - Se borra beta, como inicial quedará 2
    - **alfa \* 4 delta \* 6 gamma \* -1 epsilon**
  - Si se quiere agregar un elemento el programa solo debe chequear el header y desde ahí obtiene la dirección del primero. Agrego omega , como ppio queda 4 nuevamente
    - **alfa omega delta \* 6 gamma \* -1 epsilon**

# Archivos - Eliminación

- ▶ **Aprovechamiento de espacio**
- ▶ **Recuperación de espacio con reg. de longitud variable**
  - ▶ Marca de borrado al igual que en reg. de long. fija (ej:\*)
  - ▶ El problema de los registros de longitud variable está en que no se puede colocar en cualquier lugar, para poder ponerlo debe caber, necesariamente.
  - ▶ Lista . No se puede usar NRR como enlace. Se utiliza un campo binario que explícitamente indica en enlace (conviene que indique el tamaño).
  - ▶ Cada registro indica en su inicio la cant. de bytes.

# Archivos - Eliminación

- ▶ **Aprovechamiento de espacio**
  - ▶ **Recuperación de espacio con reg. de Longitud variable**
    - ▶ Reutilización: buscar el registro borrado de tamaño adecuado (lo suficientemente grande).
    - ▶ Como se necesita buscar, no se puede organizar la lista de disponibles como una pila.
    - ▶ El tamaño “adecuado” del primer registro borrado a reutilizar ->origina Fragmentación

# Archivos - Eliminación

## → Aprovechamiento de espacio → **Fragmentación**

- **Interna:** ocurre cuando se desperdicia espacio en un registro, se le asigna el lugar pero no lo ocupa totalmente.
  - Generalmente ocurre con **Reg. longitud fija**
  - Reg. longitud variable evitan el problema
  - **Espacio asignado → No ocupado**
- **Externa:** ocurre cuando se desperdicia espacio entre registros
  - Generalmente ocurre con **Reg. longitud variable**
  - **Espacio no asignado → No ocupado**

# Archivos - Eliminación

- ▶ Estrategias de colocación en registros de longitud variable:
  - ▶ Primer ajuste
  - ▶ Mejor ajuste
  - ▶ Peor ajuste

# Archivos - Eliminación

- ▶ **Primer ajuste:** se selecciona la primer entrada de la lista de disponibles, que pueda almacenar al registro, y se le asigna al mismo. Asigna el espacio completo (por definición)
  - ▶ Minimiza la búsqueda
  - ▶ No se preocupa por la exactitud del ajuste
- ▶ **Mejor ajuste:** elige la entrada que más se aproxime al tamaño del registro y se le asigna completa.
  - ▶ Exige búsqueda
- ▶ **Peor ajuste:** selecciona la entrada más grande para el registro, y se le asigna solo el espacio necesario, el resto queda libre para otro registro

# Archivos - Eliminación

## ► Conclusiones

- ▶ Las estrategias de colocación tienen sentido con reg. de longitud variable
- ▶ Primer ajuste: más rápido. Puede generar fragmentación interna (porque se le asigna todo el espacio, por definición)
- ▶ Mejor ajuste: Puede generar fragmentación interna (porque se le asigna todo el espacio, por definición)
- ▶ Peor ajuste: Puede generar fragmentación externa. No genera Fragmentación interna por definición

# Archivos - Operaciones

- ▶ Modificaciones
  - ▶ Consideraciones iniciales
    - ▶ Registro de long. Variable, se altera el tamaño
      - ▶ Menor, puede no importar (aunque genere fragmentación interna o externa)
      - ▶ Mayor, no cabe en el espacio
  - ▶ Otros problemas
    - ▶ Agregar claves duplicadas, y luego se modifica
    - ▶ Cambiar la clave del registro (que pasa con el orden)



# Fundamentos de Organización de Datos

Clase 4

1

# Agenda

Búsqueda de información

- Secuencial
- Directa

Busqueda binaria

- Costo de orden

Clasificación

- alternativas

# Archivos - Búsqueda

## Búsqueda de información (costo)

- # de comparaciones (operaciones en memoria)
  - Se pueden mejorar con algoritmos más eficientes.
- # de accesos (operaciones en disco)

## Buscar un registro

- + rápido si conocemos el NRR (directo)
- Secuencial debe buscarse desde el principio
- Trataremos de incorporar el uso de claves o llaves.

# Archivos - Búsqueda

Búsqueda binaria → precondiciones

- Archivo ordenado por clave
- Registros de longitud fija

Búsqueda → partir el archivo a la mitad y comparar la clave,

- puedo acceder al medio por tener long. Fija
- Si  $N$  es el # de registros, **la performance será del orden de  $\log_2 N$**
- Se mejora la performance de la búsqueda secuencial.

# Archivos → Clasificación

## Búsqueda binaria

- acota el espacio para encontrar información
- costo → mantener ordenado el archivo

## Como clasificar (ordenar) un archivo

- En RAM
- Claves en RAM
- Archivos Grandes?

# Archivos → Clasificación

Llevar el archivo a Ram

- Eficiencia?

Llevar las claves a Ram

- Eficiencia?

Si no caben en Ram las claves

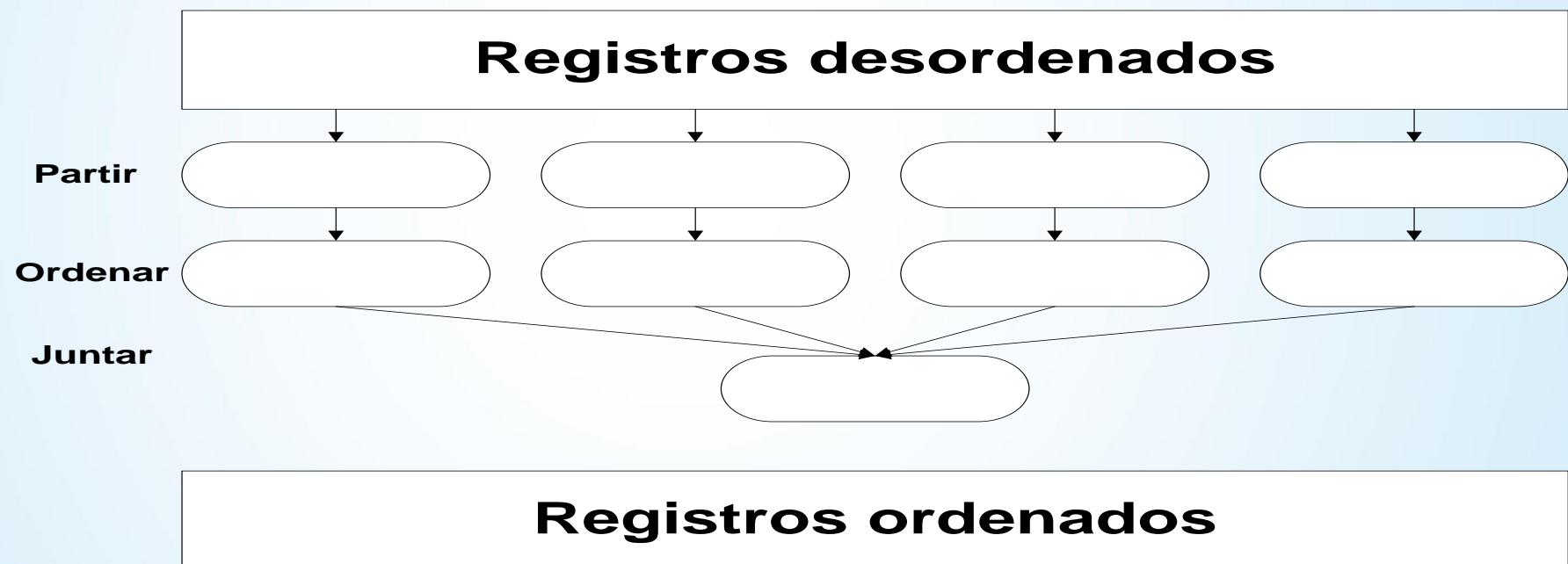
- Ordenar sobre disco?
  - Eficiencia?
- Alternativa

# Archivos → clasificación

Archivos demasiado grandes para caber en memoria Ram

- Partir el archivo
- Ordenar cada parte
- Juntar las partes ordenadas (merge)

# Archivos – Clasificación



# Archivos → Algunas conclusiones

Búsqueda binaria  
mejora la  
secuencial

## Problemas

- # accesos baja pero no llega a uno
  - Acceder por el NRR requiere una lectura
- Costo de mantener el orden
- Clasificación en RAM solo para archivos pequeños

## Mejorar el método de ordenación

- No reordenando TODO el archivo
- Reorganizando con métodos más eficientes (árboles)



# Fundamentos de Organización de Datos

1

Clase 5

# Agenda

Indices

- Definición
- Operaciones básicas

Ejemplo

Indices  
secundarios

- Características

# Búsqueda de datos - Índices

Búsqueda de información:

- debemos minimizar el número de accesos

Secuencial (poco eficiente)

Binaria (muy costosa)

Estructuras auxiliares

# Búsqueda de datos - Índices

## Ejemplo

Las últimas págs. de un libro suelen contener un índice (tabla que contiene una lista de temas y los nº de pág. donde pueden encontrarse)

El uso de un índice es mejor alternativa que buscar un tema a lo largo del libro en forma secuencial

# Búsqueda de datos - Índices

Otro ejemplo: encontrar libros en una biblioteca (por autor, título o tema)

- **Alternativa 1:** disponer 3 copias de cada libro y 3 edificios de biblioteca separados.
  - Edificio 1: libros clasificados por autor,
  - Edificio 2: libros clasif por titulo,
  - Edificio 3: libros clasif por tema (**absurdo**)
- **Alternativa 2:** usar un catálogo de tarjetas. En realidad es un conjunto de 3 índices, cada uno tiene una campo clave distinto, pero todos tienen el mismo número de catálogo como campo de referencia.

El uso de índices proporciona varios caminos de acceso a un archivo

# Indices → definición

---

**Herramienta** para encontrar registros en un archivo. Consiste de un campo de llave (búsqueda) y un campo de referencia que indica donde encontrar el registro dentro del archivo de datos.

---

**Tabla** que opera con un procedimiento que acepta información acerca de ciertos valores de atributos como entrada (llave), y provee como salida, información que permite la rápida localización del registro con esos atributos.

---

**Estructura de datos** (clave, dirección) usada para decrementar el tiempo de acceso a un archivo.

# Indices → Definición

---

**Índice:**  
equivale a  
índice  
temático de  
un libro

---

Característica  
fundamental

---

(tema, #hoja)

---

(clave, NRR/distancia en bytes)

---

**Estructura más simple es un árbol**

---

**Permite imponer orden en un archivo sin  
que realmente este se reacomode**

# Indices → Ejemplo

Dir. Reg.	Cía	Nº ID	Título	Compositores	Artista
32	LON	2312	Romeo y Julieta	Prokofiev	Maazel
77	RCA	2626	Cuarteto en Do...	Beethoven	Julliard
132	WAR	23699	Touchstone	Corea	Corea
167	ANG	3795	Sinfonía Nº 9	Beethoven	Giulini
211	COL	38358	Nebraska	Springsteen	Springsteen
256	DG	18807	Sinfonía Nº 9	Beethoven	Karajan
300	MER	75016	Suite el Gallo...	Rimsky-Korsakov	Leinsdorf
353	COL	31809	Sinfonía Nº 9	Dvorak	Bernstein
396	DG	139201	Concierto para Violín	Beethoven	Ferras
422	FF	245	Good News	Sweet Honey in..	Sweet Honey in..

# Indices → ejemplo

Llave primaria: cía grabadora + N° de identificación de la cía

- Forma canónica: cía en mayúsculas + N° identificación
- No se puede hacer búsqueda binaria sobre el archivo ya que tiene reg. de longitud variable (no se puede usar en NRR como medio de acceso)

Dos Archivos: índice y datos

- Se construye un índice: llave de 12 caracteres (alineada a izq. y completada con blancos) más un campo de referencia (dir. del primer byte del registro correspondiente)
- Estructura del índice: archivo ordenado de reg. de long fija (puede hacerse búsqueda binaria).
- En memoria
- Más fácil de manejar que el arch. de datos

# Indices → ejemplo

Llave	Ref	Dir. de registro	Registro de Datos
ANG3795	167	32	LON 2312 Romeo y Julieta Prokofiev...
COL31809	353	77	RCA 2626 Cuarteto en Do...
COL38358	211	132	WAR 23699 Touchstone Corea...
DG139201	396	167	ANG 3795 Sinfonía N°9 Beethoven...
DG18807	256	211	COL 38358 Nebraska Springsteen...
FF245	442	256	DG 18807 Sinfonía N° 9 Beethoven...
LON2312	32	300	MER 76016 Suite El gallo de Oro Rimsky...
MER75016	300	353	COL 31809 Sinfonía N°9 Dvorak...
RCA2626	77	396	DG 139201 Concierto para violín Beethoven...
WAR23699	132	422	FF 245 Good News Sweet Honey in the....

# Indices → como implantarlos?

## Operaciones básicas en un archivo indizado

- Índice en memoria (búsqueda binaria + rápida, comparada con archivos clasificados)
- Crear los archivos (el índice y el archivo de datos se crean vacíos, solo con registro cabecera)
- Cargar el índice en memoria (se supone que cabe, ya que es lo suficientemente pequeño. Se almacena en un arreglo)
- Reescritura del archivo de índice (cambios → reescribir)

# Indices → como implantarlos?

## Agregar nuevos registros

- Implica agregar al archivo de datos y al archivo de indices
- Archivo de datos: copiar al final (se debe saber el NRR (fija) o distancia en bytes (variable) para el índice)
- Índice ordenarse con cada nuevo elemento en forma canónica (en mem.), setear el flag anterior

## Eliminar un registro

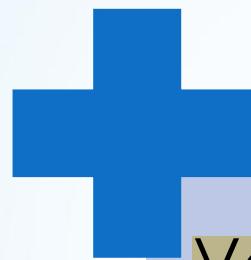
- Arch. datos → Cualquier técnica de las vistas para reutilizar el espacio
- Arch. índices → se quita la entrada (ó se podría marcar como borrado).

# Indices → como implantarlos?

## Actualización de registros

- Sin modificar la clave (que pasa con el índice?)
  - Si el registro no cambia de longitud, se almacena en la misma posición física, el índice “no se toca”.
  - Si el reg. cambia de longitud (se agranda) y se reubica en el arch. de datos  
➔ se debe guardar la nueva posición inicial en el índice
  - Si se trata de long. Fija, no hay que hacer mas actividad
- Modificando la clave (que sucede?)
  - Se modifica el archivo de datos
  - Se debe actualizar y reorganizar el archivo de índices
  - Cómo simplificar ➔ Modificar = Eliminar + Agregar (ya vistos)

# Indices → Resumen



## Ventajas

- Se almacena en memoria principal
- Permite búsqueda binaria
- El mantenimiento es menos costoso

## Desventajas

- Si no caben en memoria RAM?
- Reescritura del archivo de índices?
- Persistencia de datos

# Indices → Persistencia de Datos

Índices grandes  
para entrar en  
memoria →  
Soluciones

Uso de Árboles

Organización por  
Dispersión  
(hashing)

# Indices secundarios

## Índices Secundarios

No sería natural solicitar un dato por clave

En su lugar se utiliza normalmente un campo mas fácil de recordar  
( ej: buscar una canción por su título o por su compositor)

Este campo es un campo que pertenece a una llave secundaria porque puede repetirse

Las claves secundarias se pueden repetir

El índice secundario relaciona la llave secundaria con la llave primaria

Acceso → 1º por llave secundaria (se obtiene la clave primaria) y luego llave primaria (en índice primario)

# Indices secundarios

Indice de Llave Secundaria	Compositores Llave Primaria
BEETHOVEN	ANG3795
BEETHOVEN	DG139201
BEETHOVEN	DG18807
BEETHOVEN	RCA2626
COREA	WAR23699
DVORAK	COL31809
PROKOFIEV	LON2312
RIMSKY-KORSAKOV	MER75016
SPRINGSTEEN	COL38358
SWEET HONEY....	FF245

# Indices secundarios

## Problemas: la repetición de información

- El arch. de índices se debe reacomodar con cada adición, aunque se ingrese una clave secundaria ya existente, dado que existe un 2do orden por la clave primaria.
- Misma clave varias ocurrencias, en distintos registros
  - Se desperdicia espacio
  - Menor posibilidad de que el índice quepa en memoria

# Indices secundarios

## Soluciones

- Arreglo: clave + vector de punteros con ocurrencias

BEETHOVEN ANG3795 DG139201 DG18807 RCA2626

- Al agregar un nuevo reg. de una clave existente no se debe reacomodar nada-> solo reacomodar el vector de ocurrencias
- Al agregar un nuevo reg. con una clave nueva, se genera un arreglo con la clave y un elemento en el vector de punteros

## Problema: elección del tamaño del vector.

- Tamaño fijo
- Puede haber casos en que sea insuficiente
- Puede haber casos que sobre espacio, provocando fragmentación interna
- Mejora: clave + lista de punteros con ocurrencias

# Indices secundarios

## Listas invertidas:

Archivos en los que una llave secundaria lleva a un conjunto de una o más claves primarias → lista de referencias de claves primarias

No se pierde espacio (no hay reserva)

Si se agrega un elem. a la lista → no se necesita una reorganización completa

# Indices secundarios

Listas  
invertidas

## Organización física

## Operaciones

Archivos secundarios

Marcas o referencias

Agregar un nuevo consiste en agregar concurrencias en la lista invertida

Idem borrar

Modificaciones dependiendo el caso

# Indices secundarios

NRR	Archivo de índice secundario	
0	BEETHOVEN	3
1	COREA	2
2	DVORAK	7
3	PROKOFIEV	10
4	RIMSKY-KORSAKOV	6
5	SPRINGSTEEN	4
6	SWET HONEY IN...	9

NRR	Arch de listas de llaves primarias	
0	LON2312	-1
1	RCA2626	-1
2	WAR23699	-1
3	ANG3795	8
4	COL38358	-1
5	DG18807	1
6	MER76016	-1
7	COL31809	-1
8	DG139201	5
9	FF245	-1
10	ANG36193	0

# Indices secundarios



## Ventajas

- El único reacomodamiento en el arch. índice -> al agregar o cambiar un nombre
- Borrar o añadir grabaciones para un compositor->sólo cambiar el archivo de listas
- Como el reacomodamiento es a bajo costo se podría almacenar el arch. índice en mem. secundaria , liberando RAM

## Desventaja

- el arch. de listas es conveniente que esté en memoria ppal. porque podría haber muchos desplazamientos en disco → costoso si hay muchos índices secundarios



# Fundamentos de Organización de Datos

1

Clase 6

# Agenda

Arboles

Arboles  
Balanceados

- Binarios
  - AVL
  - Multicamino
  - Balanceados
- 
- Características
  - B, B\*, B+
  - Operaciones
  - Prefijos simples

# Arboles → introducción

## Problemas con los índices?

- La búsqueda binaria aun es costosa
- Mantener los índices ordenados es costoso
- Solución → RAM
- Objetivo → persistencia de datos

## Árboles

- Estructuras de datos que permiten localizar en forma más rápida información de un archivo, tienen intrínsecamente búsqueda binaria

# Arboles binarios

Que es un árbol binario?

- Estructuras de datos donde cada nodo tiene a lo sumo dos sucesores, a izquierda y a derecha

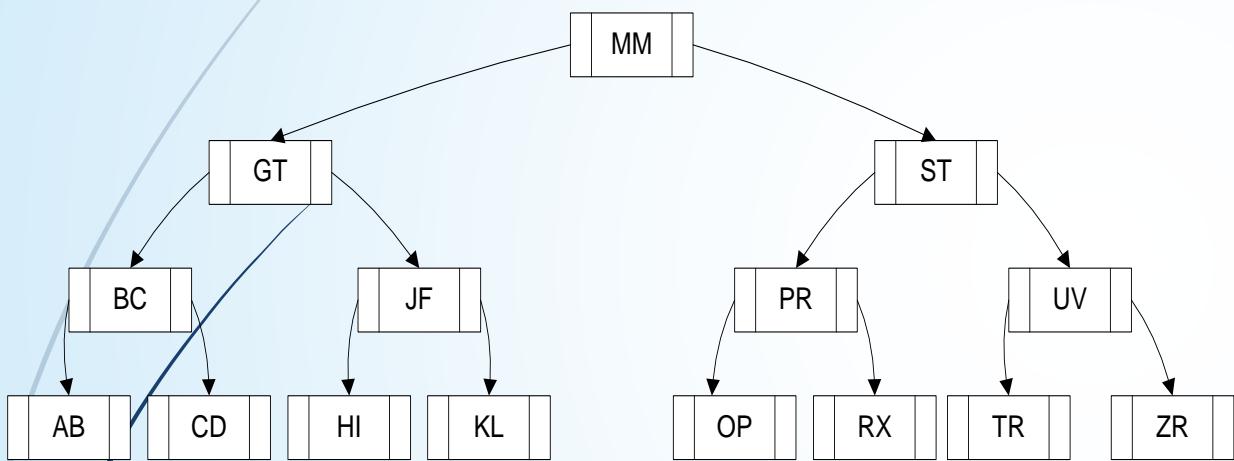
Un árbol binario, puede implantarse en disco?

- Como lograr la persistencia?

Ejemplo → supongamos estas claves

- MM ST GT PR JF BC UV CD HI AB KL TR OP RX ZR

# Arboles binarios



Raíz → 0

	Clave	Hijo izq	Hijo Der
0	MM	1	2
1	GT	3	4
2	ST	8	11
3	BC	5	6
4	JF	7	14
5	AB	-1	-1
6	CD	-1	-1
7	HI	-1	-1

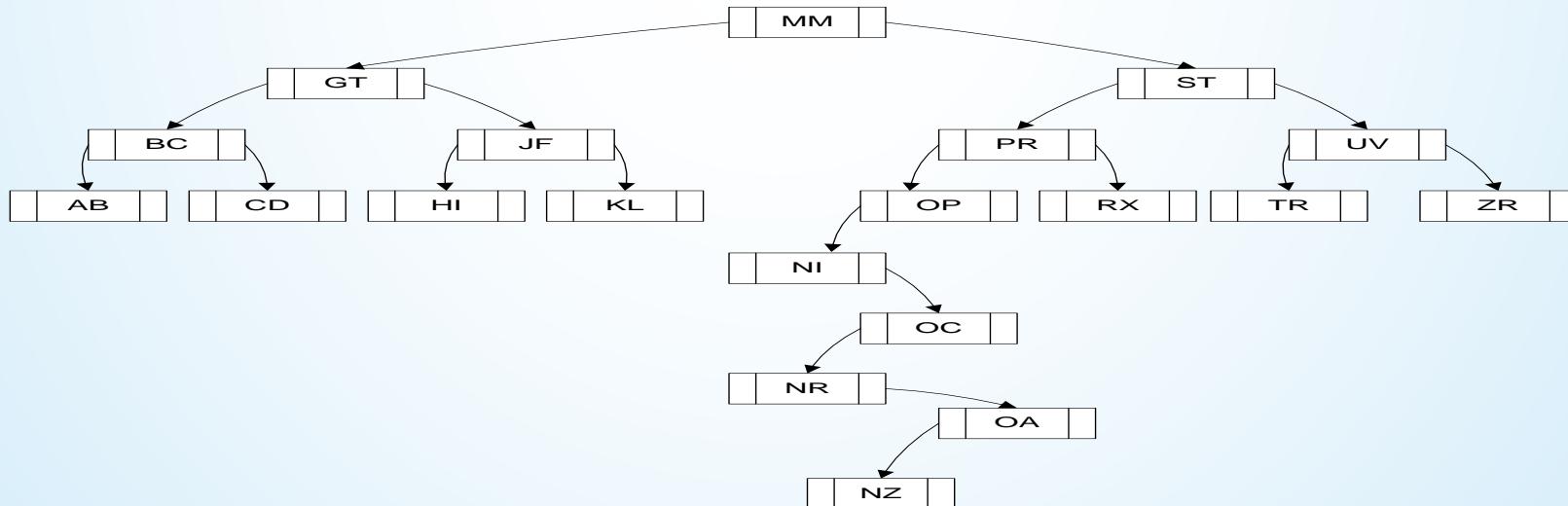
	Clave	Hijo izq	Hijo Der
8	PR	9	10
9	OP	-1	-1
10	RX	-1	-1
11	UV	12	13
12	TR	-1	-1
13	ZR	-1	-1
14	KL	-1	-1

# Arboles binarios

Árbol balanceado: un árbol está balanceado cuando la altura de la trayectoria más corta hacia una hoja no difiere de la altura de la trayectoria más grande.

Inconveniente de los binarios: se desbalancean fácilmente.

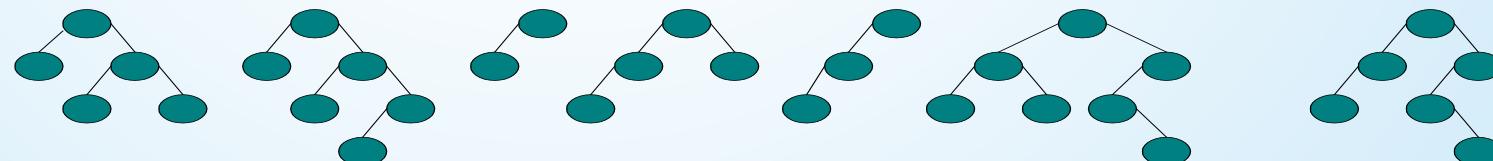
Supongamos que llegan las claves : NI OC NR OA NZ



# Árboles AVL

## Árboles AVL

- Árbol binario balanceado en altura (BA(1)) en el que las inserciones y eliminaciones se efectúan con un mínimo de accesos.
- Árbol balanceado en altura:
  - Para cada nodo existe un límite en la diferencia que se permite entre las alturas de cualquiera de los subárboles del nodo (BA( $k$ )), donde  $k$  es el nivel de balance)
  - Ejemplos:



# Arboles AVL y Binarios

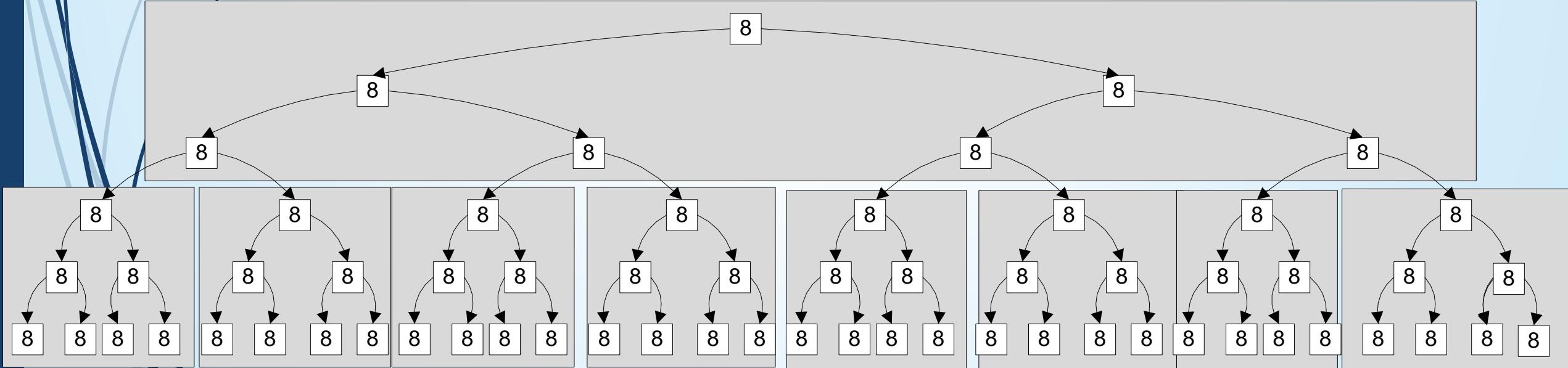
## Características/Conclusiones

- Estructura que debe ser respetada
- Mantener árbol, rotaciones restringidas a un área local del árbol
  - Binario: → Búsqueda:  $\log_2(N+1)$
  - AVL: → Búsqueda:  $1.44 \log_2(N+2)$
  - Ambas performance por el peor caso posible

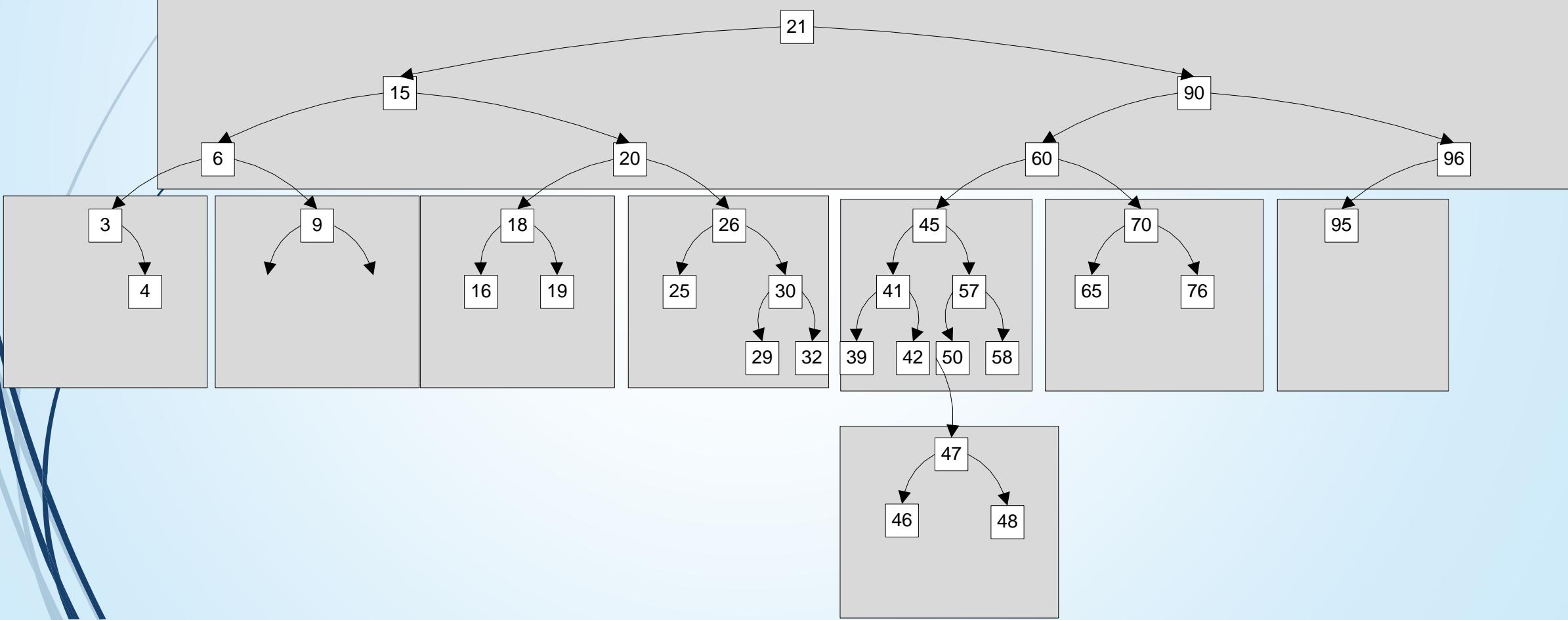
# Árboles Binarios Paginados

## Árboles 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?



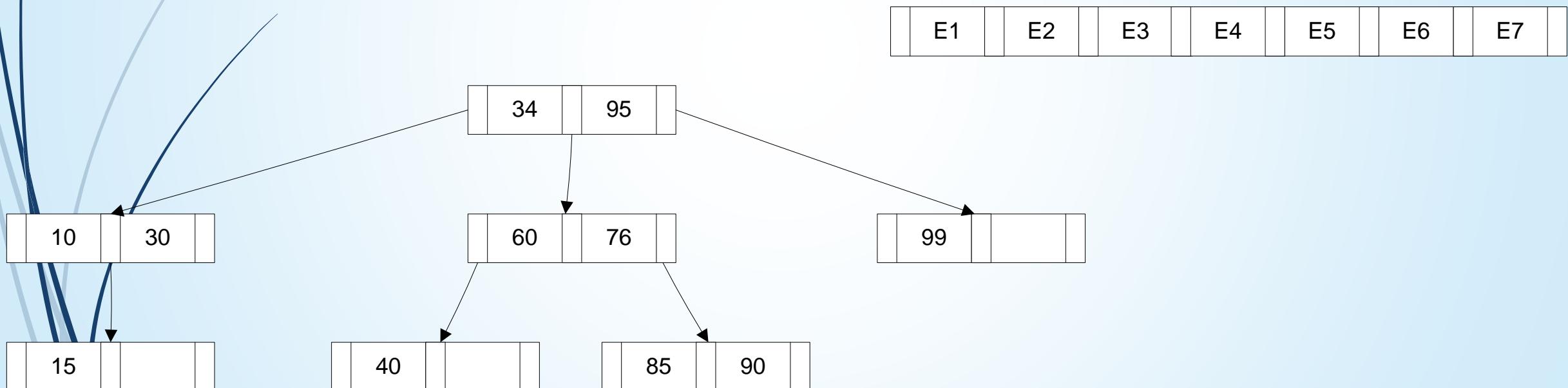
# Árboles Binarios Paginados



# Árboles multicamino

Generalización de árboles binarios, c/nodo tiene k punteros y k-1 claves (o registros), disminuye la profundidad del árbol,

- Orden del árbol.



# Arboles balanceados

Son árboles multicamino con una construcción especial en forma ascendente que permite mantenerlo balanceado a bajo costo.

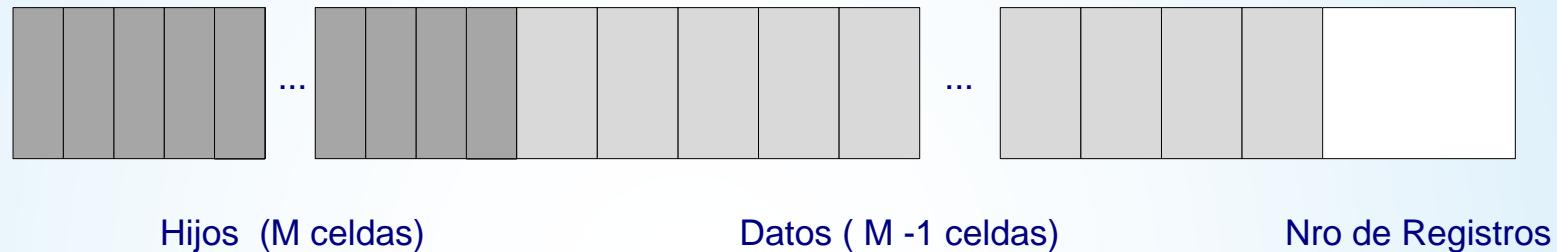
Propiedades de un árbol B de orden M:

- Ningún nodo tiene más de M hijos
- C/nodo (menos raíz y los terminales) tienen como mínimo  $[M/2]$  hijos
- La raíz tiene como mínimo 2 hijos (o sino ninguno)
- Todos los nodos terminales a igual nivel
- Nodos no terminales con K hijos contienen K-1 registros. Los nodos terminales tienen:
  - Mínimo  $[M/2] - 1$  registros
  - Máximo  $M - 1$  registros

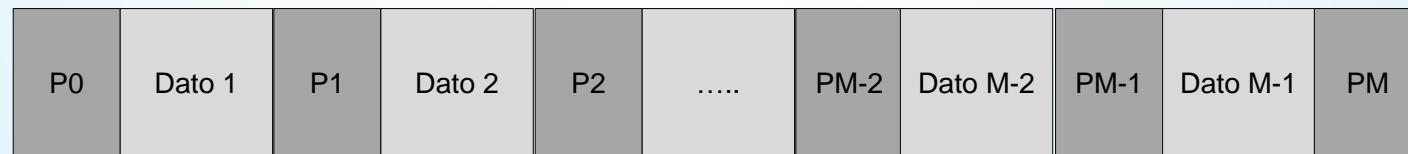
PO	R1	P1	R2	P2	R3	P3	R4	P4	R5	P5	Nro de registros
----	----	----	----	----	----	----	----	----	----	----	------------------

# Arboles Balanceados

- Formato del nodo



Formato del Nodo para archivo del índice arbol b



Formato Gráfico del Nodo del índice arbol B

# Arboles balanceados

## Creacion:

- Dadas las claves: 43 2 53 88 75 80  
15 49 60 20 57 24
- Como se construye el árbol?
- Como se genera el archivo de datos que persiste el árbol?

# Arboles Balanceados

	Punteros			Datos			Nro Datos
	-1	-1	-1	-1	2	15	2
0	-1	-1	-1	-1	57	60	75
1	0	5	3		20	43	2
2	-1	-1			49		1
3	-1	-1			88		1
4	-1	-1			24		1
5	-1	-1			80		1
6	1	4			53		1
7	2	6					

**HEA!** Herramienta de Software para enseñanza de árboles B

Tipo de Árbol: ARBOL B | Orden: 4 | CREAR

Nº a insertar: + 24 | Nº a eliminar: - | Nº a buscar: 53

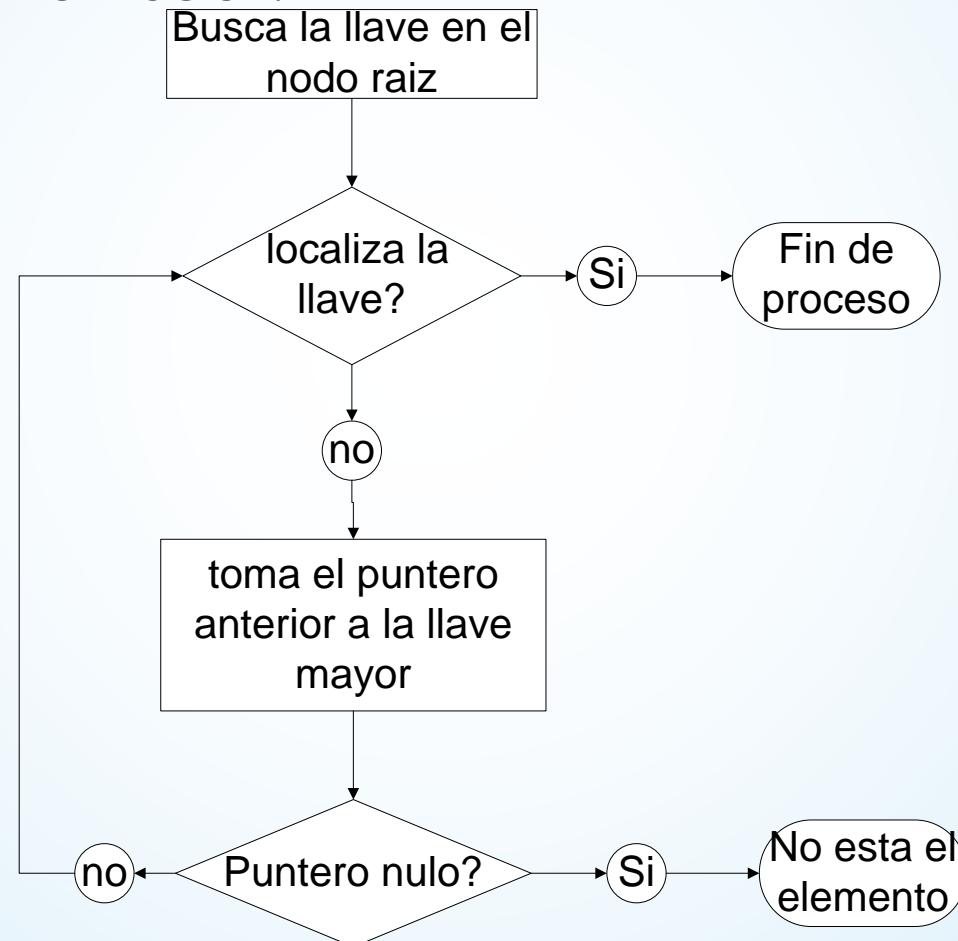
BUSCAR | LIMPIAR

```

graph TD
    N7[Nodo 7: 53] --> N2[Nodo 2: 20, 43]
    N7 --> N6[Nodo 6: 80]
    N2 --> N0[Nodo 0: 2, 15]
    N2 --> N5[Nodo 5: 24]
    N6 --> N1[Nodo 1: 57, 60, 75]
    N6 --> N4[Nodo 4: 88]
  
```

# Árboles Balanceados

→ Busqueda de información:



# Arboles Balanceados

## Performance de búsqueda

- Mejor caso: 1 lectura
- Pero caso:  $h$  lecturas (con  $h$  altura del árbol)
- Cual es el valor de  $h$ ?
  - Axioma: árbol balanceado de Orden  $M$ , si el número de elementos del árbol es  $N \rightarrow$  hay  $N+1$  punteros nulos en nodos terminales.

# Árboles Balanceados

Cota  
para  
 $h$

Nivel	# mínimo de descendientes
1	2
2	$2 * [M/2]$
3	$2 * [M/2] * [M/2]$
.....	.....
$h$	$2 * [M/2]^{h-1}$

**Relación entre  $h$  y # de nodos**

$$N+1 \geq 2 * [M/2]^{h-1}$$

$$h \leq [1 + \log_{[M/2]} ((N+1)/2)]$$

**Si  $M = 512$  y  $N = 1000000 \rightarrow h \leq 3.37$  (4 lecturas encuentra un registro)**

# Árboles Balanceados

## Inserción (creación)

- Los registros se insertan en un nodo Terminal
- Casos posibles
  - El registro tiene lugar en el nodo Terminal (no se produce overflow): solo se hacen reacomodamientos internos en el nodo
  - El registro no tiene lugar en el nodo Terminal (se produce overflow): el nodo se divide y los elementos se reparten entre los nodos, hay una promoción al nivel superior, y esta puede propagarse y generar una nueva raíz.

# Árboles Balanceados

## Performance de la inserción

- Mejor caso (sin overflow)
  - H lecturas
  - 1 escritura
- Peor caso (overflow hasta la raíz, aumenta en uno el nivel del árbol)
  - H lecturas
  - $2h+1$  escrituras (dos por nivel más la raíz)
- Estudios realizados
  - $M = 10 \quad 25\%$  divisiones
  - $M = 100 \quad 2\%$  divisiones

# Árboles Balanceados

## Eliminación

- Siempre eliminar de nodos terminales (trabajamos con árboles)
- Si se va a eliminar un elemento que no esta en nodo terminal → llevarlo primero a nodo terminal
- Posibilidades ante eliminación
  - Mejor caso: borra un elemento del nodo y no produce underflow, solo reacomodos ( $\# \text{ elementos} \geq [M/2]-1$ )
  - Peor caso: se produce underflow,  $\# \text{ elementos} < [M/2] - 1$
- Dos soluciones
  - Redistribuir
  - concatenar

# Árboles Balanceados

## Definición: nodo adyacente hermano

- Dos nodos son adyacentes hermanos si tienen el mismo padre y son apuntados por punteros adyacentes en el padre.

## Redistribuir

- Cuando un nodo tiene underflow puede trasladarse llaves de un nodo adyacente hermano (en caso que este tenga suficientes elementos)

## Concatenación:

- Si un nodo adyacente hermano está al mínimo (no le sobra ningún elemento) no se puede redistribuir, se concatena con un nodo adyacente disminuyendo el # de nodos (y en algunos casos la altura del árbol)

# Árboles Balanceados

## Performance de la eliminación

- Mejor caso (borra de un nodo Terminal)
  - H lecturas
  - 1 escritura
- Peor caso (concatenación lleva a decrementar el nivel del árbol en 1)
  - $2h - 1$  lecturas
  - $H + 1$  escrituras

# Árboles Balanceados → B\*

## Eliminación

- Redistribución
- Concatenación

## Inserción

- ???
- División

# Árboles Balanceados → B\*

La redistribución podría posponer la creación de páginas nuevas

Se pueden generar árboles B más eficientes en términos de utilización de espacio

# Árboles Balanceados → B\*

Árbol B especial en que cada nodo está lleno por lo menos en 2/3 partes

## Propiedades (orden M)

Cada página tiene máximo M descendientes

Cada página, menos la raíz y las hojas, tienen al menos  $[(2M - 1) / 3]$  descendientes

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

Una página hoja contiene por lo menos  $[(2M - 1) / 3] - 1$  llaves, y no más de M-1.

# Árboles Balanceados → B\*

## Operaciones de Búsqueda

- Igual que el árbol B común

## Operaciones de Inserción

- Tres casos posibles
  - **Derecha:** redistribuir con nodo adyacente hermano de la derecha (o izq. si es el último)
  - **Izquierda:** redistribuir con nodo adyacente hermano de la Izquierda (o der. si es el último)
  - **Izquierda o derecha:** si el nodo de la derecha está lleno se busca redistribuir con la izquierda ,y viceversa.
  - **Izquierda y derecha:** busca llenar los tres nodos, estos tendrán un  $\frac{3}{4}$  parte llena.

# Árboles Balanceados → B\*

Costo de la redistribución

	Mejor	Peor
Derecha	RRWW	RRWWW
Izq o der	RRWW	RRRWWW (divido solo dos)
Izq y der	RRWW	RRRWWWW

# Árboles Balanceados

Técnicas de estrategias de reemplazo: LRU (last recently used)

Análisis numérico

# llaves = 2400 # páginas = 140 Altura = 3 niveles

1 5 10 20

3.00 1.71 1.42 0.97

# Árboles Balanceados

Archivos secuenciales indizados

Permiten una mejor recorrida por algún tipo de orden

Hasta ahora métodos disjuntos, se opta:

Debemos encontrar una solución que agrupe ambos casos

Indizado (ordenado por una llave)

Secuencial (acceder por orden físico, devolviendo el registro en orden de llave)

rápida recuperación (Árbol)

Recuperación ordenada (secuencial)

# Árboles Balanceados

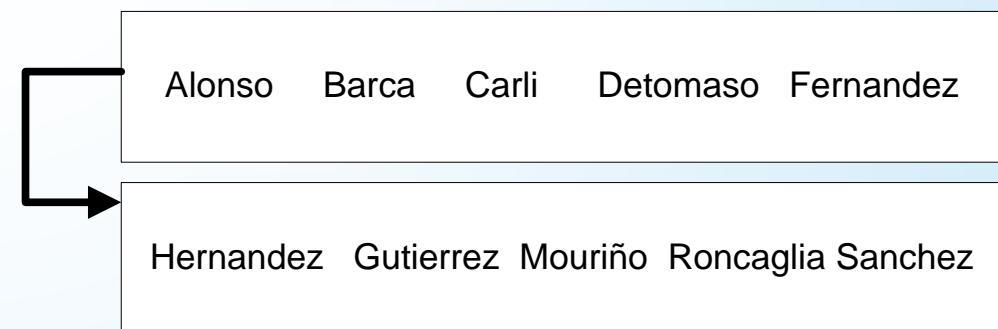
## Conjunto de secuencias

- Conjunto de registros que mantienen un orden físico por llave mientras que se agregan o quitan datos, si podemos mantenerlo podemos indizarlos

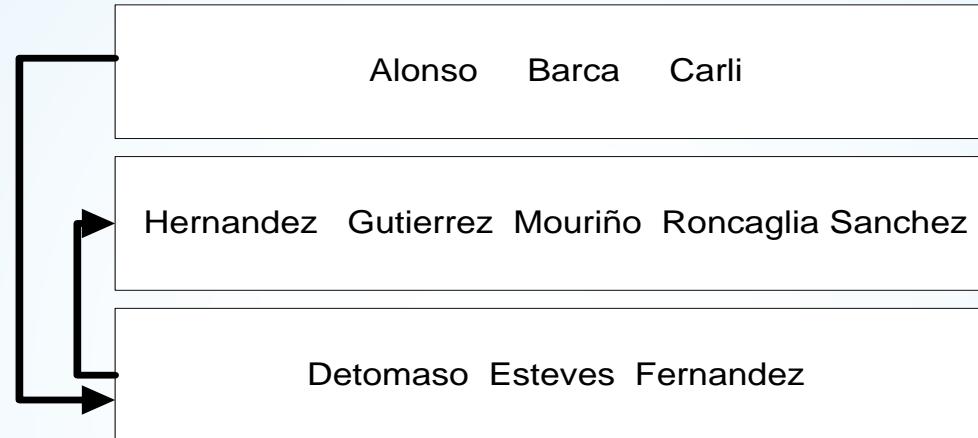
## Possible solución

- Mantener bloques de datos
- Cada bloque con registros y puntero al siguiente

Alonso	Barca	Carli	Detomaso	Fernandez
--------	-------	-------	----------	-----------



# Árboles Balanceados



## Costo

- Aumenta el tamaño del archivo (fragmentación interna)
- No hay orden físico salvo dentro del un bloque.
- Tamaño del bloque
  - Debe permitir almacenar varios bloques en RAM (redistribución)
  - Las E/S deben ser rápidas y sin necesidad de desplazamientos
- Como logramos ahora una rápida búsqueda?

# Árboles Balanceados → B+

Consiste en un conjunto de grupos de registros ordenados por clave en forma secuencial, junto con un conjunto de índices, que proporciona acceso rápido a los registros.

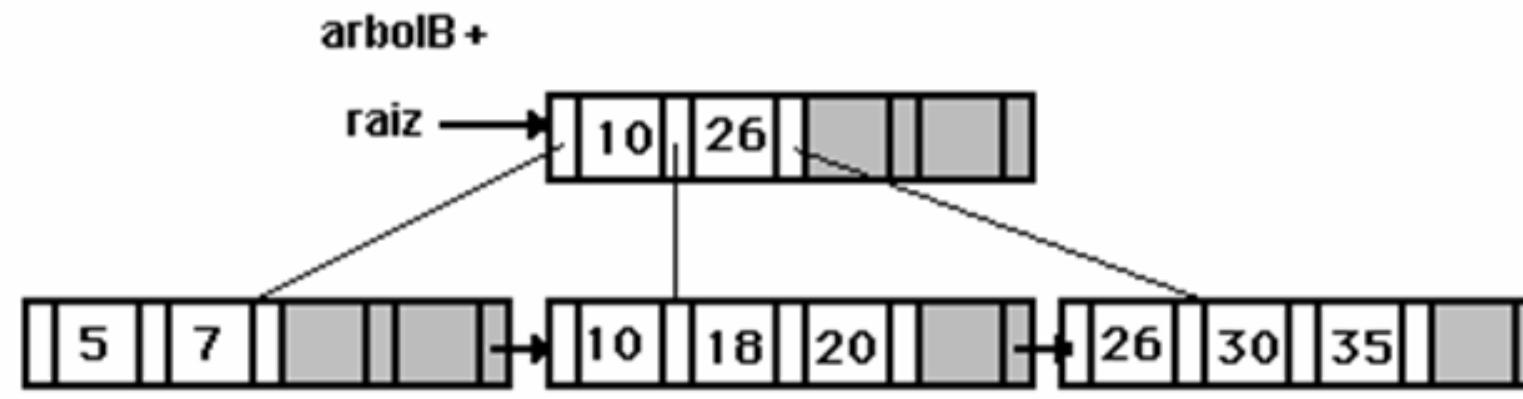
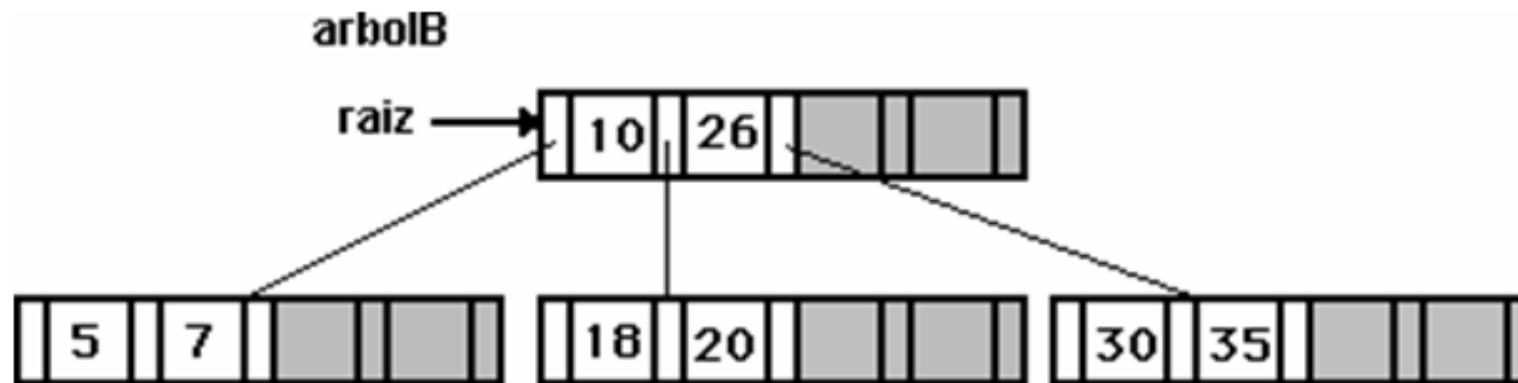
## Propiedades

- 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.

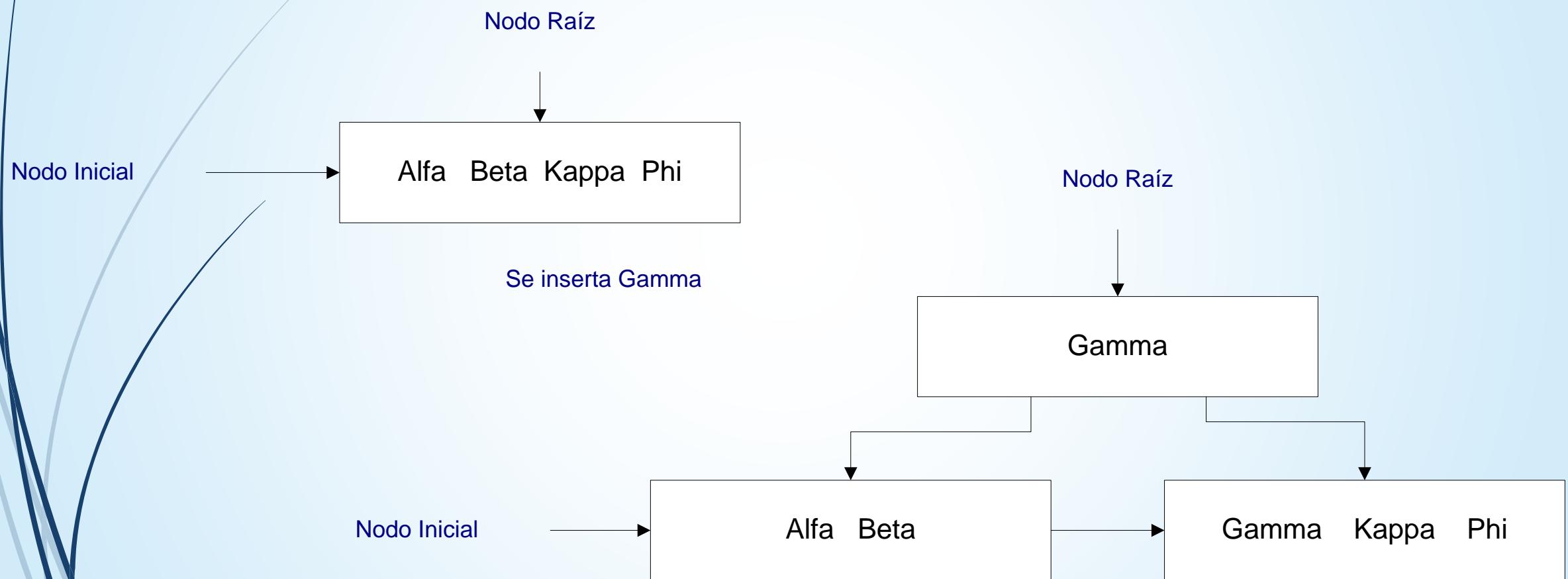
Los nodos no terminales no tienen datos sino punteros a los datos.

# Árboles B+

## ■ Ejemplo

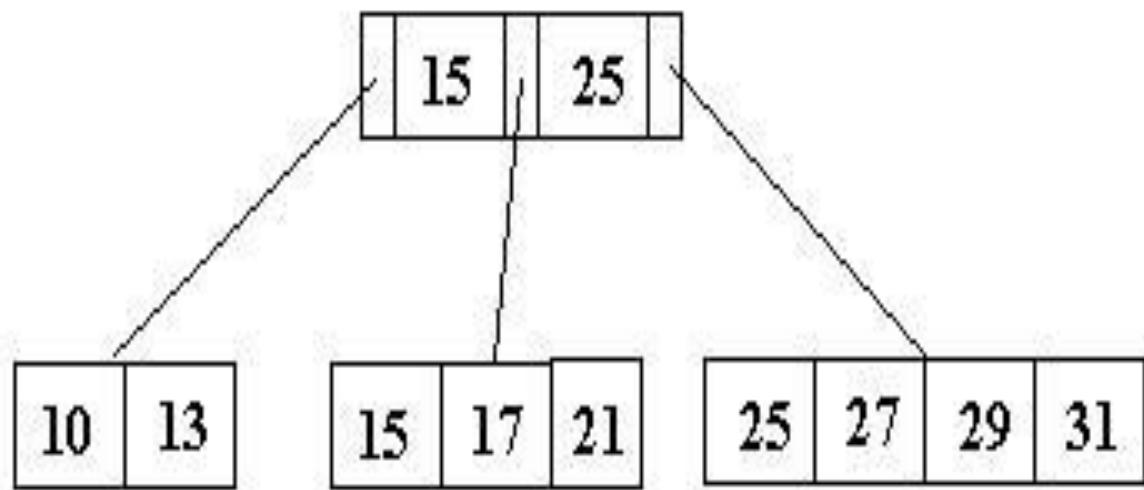
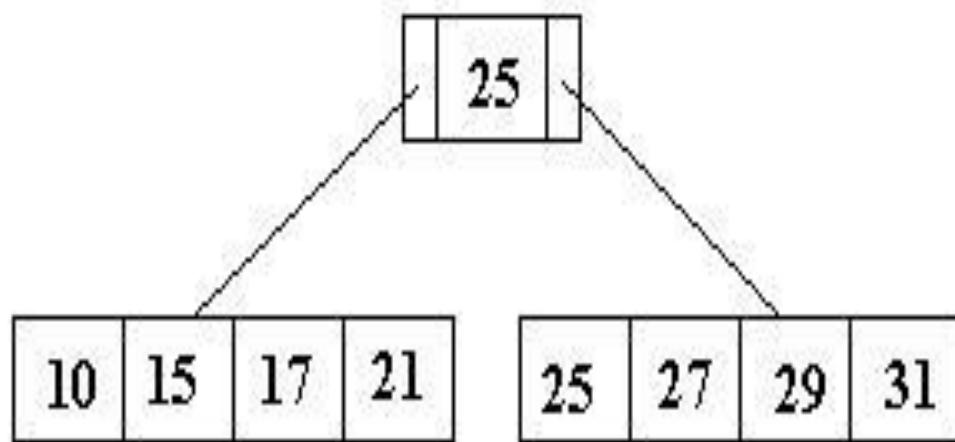


# Árboles Balanceados → B+

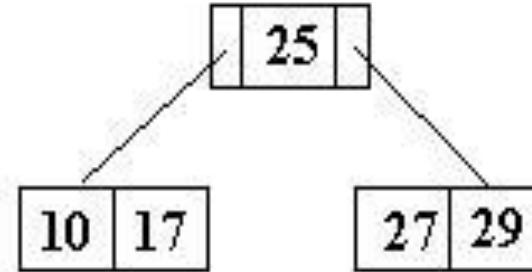
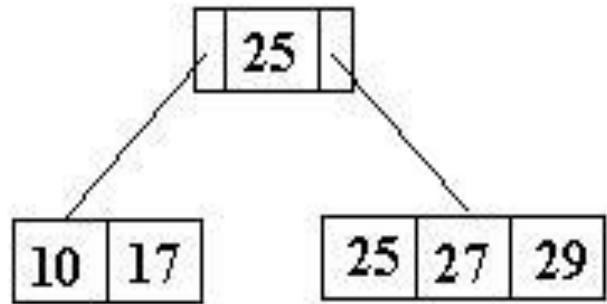


# Arboles B+

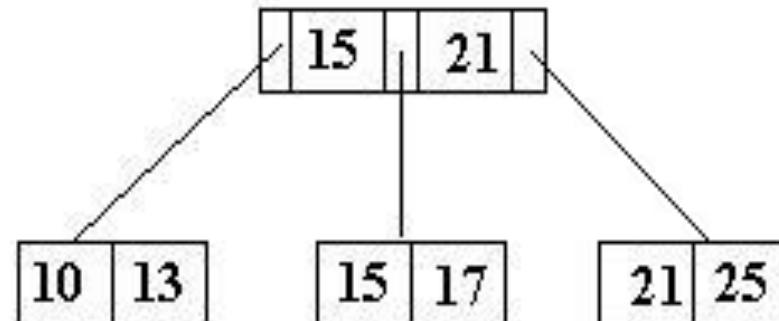
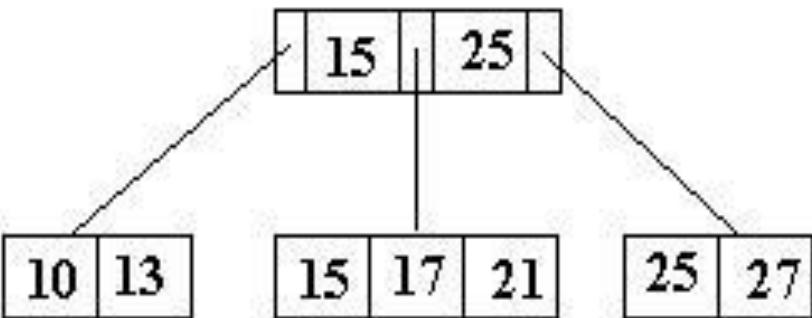
► Inserción clave 13 (M=5)



# Arboles B+

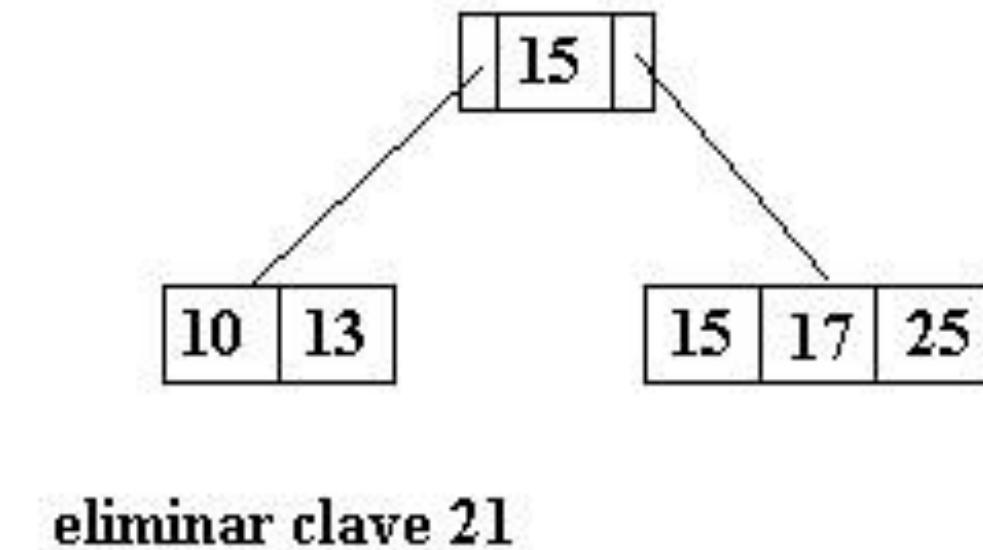
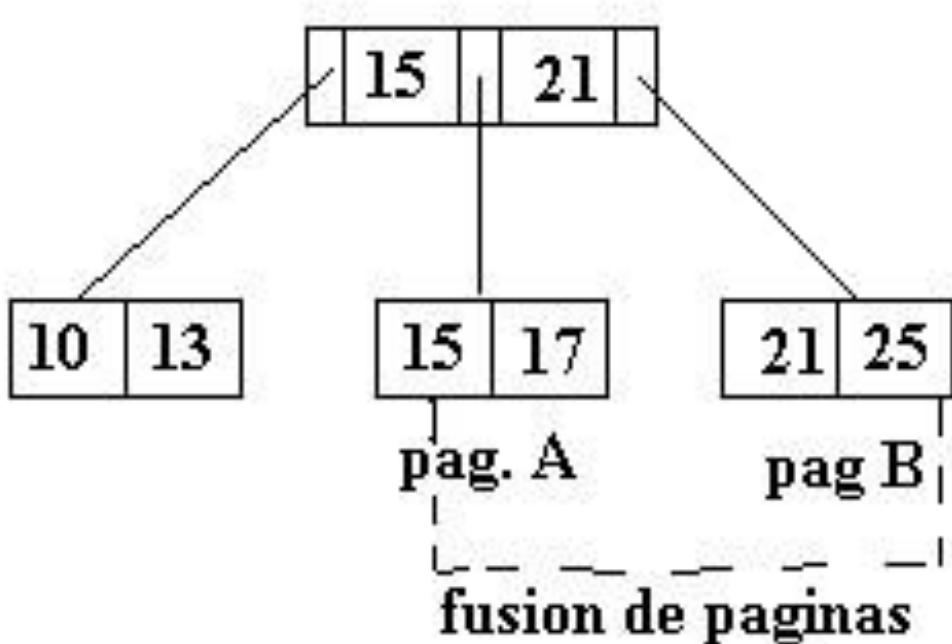


eliminar clave 25



eliminar clave 27

# Arboles B+



# Árboles Balanceados → B+

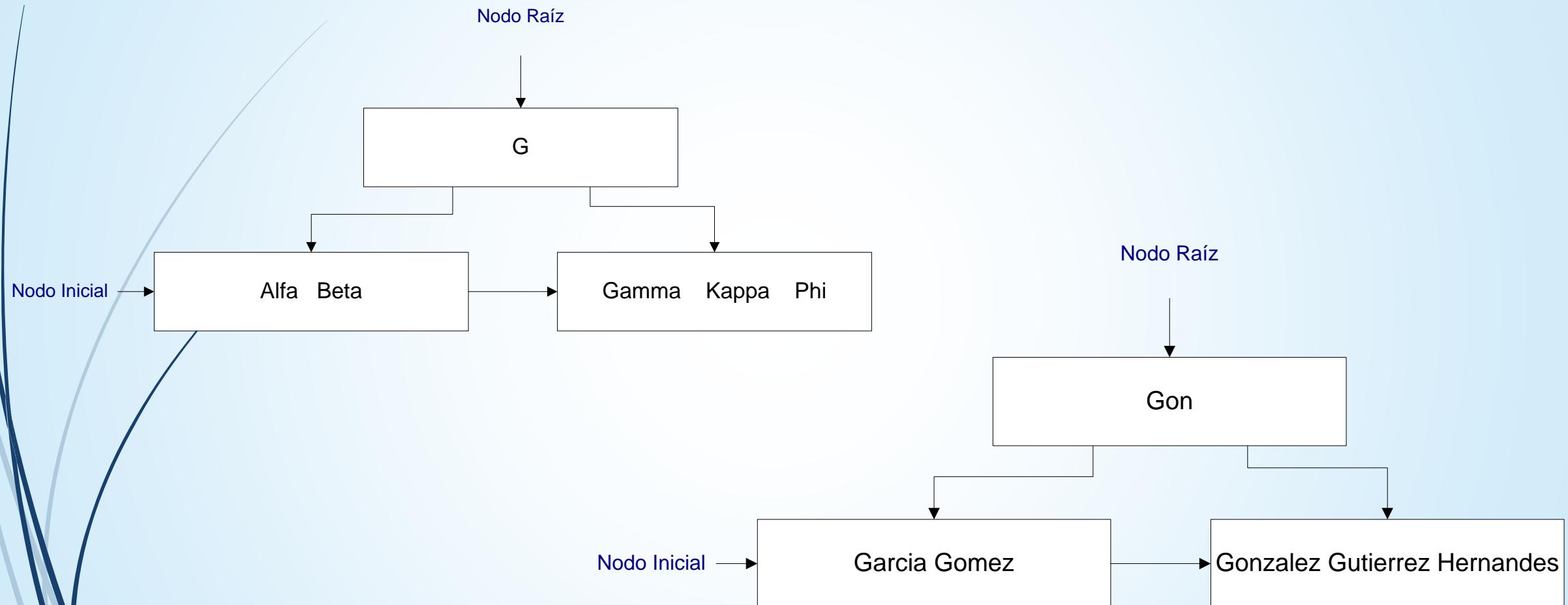
## Separadores

- Derivados de las llaves de los registros que limitan un bloque en el conjunto de secuencia
- Separadores más cortos, ocupan espacio mínimo

## Árbol B+ de prefijos simples

- Árbol B+ en el cual el conjunto índice está constituido por separadores más cortos

# Árboles Balanceados → B+



# Árboles Balanceados → conclusiones

	Árbol B	Árbol B+
Ubicación de datos	Nodos (cualquiera)	Nodo Terminal
Tiempo de búsqueda	=	=
Procesamiento secuencial	Lento (complejo)	Rápido (con punteros)
Inserción eliminación	Ya discutida	Puede requerir + tiempo



# Fundamentos de Organización de Datos

1

Clase 7

# Agenda

## Hashing

- Definición
- Tipos
- Propiedades

## Propiedades



- Función de hash
- Densidad / tamaño nodo
- Tratamiento del overflow

## Dispersión

- Estática
- Dinámica

# Hashing (Dispersión) → Introducción

Necesitamos un mecanismo de acceso a registros con una lectura solamente

- Secuencia:  $N/2$  accesos promedio
- Ordenado:  $\log_2 N$
- Árboles: 3 o 4 accesos



Clave Primarias → características

- No se repiten
- El resto de las claves actúan a través de ella
- Cuando se aprenda a modelar, tendrán más características que las hacen especiales

# Hashing (Dispersión) → Definición

Técnica para generar una dirección base única para una llave dada. La dispersión se usa cuando se requiere acceso rápido a una llave



Técnica que convierte la llave del registro en un número aleatorio, el que sirve después para determinar donde se almacena el registro.

Técnica de almacenamiento y recuperación que usa una función de hash para mapear registros en dirección de almacenamiento.

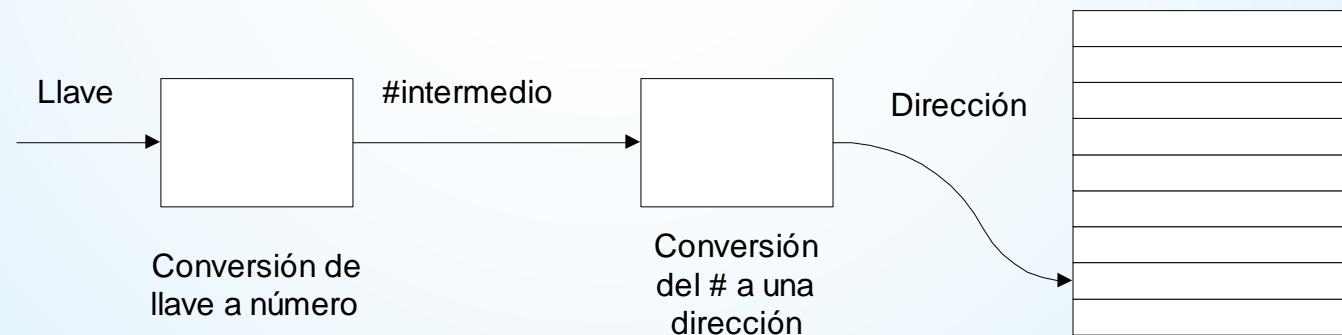
# Hashing (Dispersion) → definición

- ▶ Archivos secuenciales indizados
  - ▶ Archivo de datos
  - ▶ Archivo con indice primario
  - ▶ Archivos con indices un  o secundarios
- ▶ Archivos directos
  - ▶ UN ACCESO
  - ▶ No puede haber estructuras adicionales
  - ▶ Se organiza EL archivo de datos
  - ▶ Solo puede organizarse por un UNICO criterio
  - ▶ Ese criterio es la clave primaria

# Hashing (Dispersión) → Definición

## Atributos del hash

- No requiere almacenamiento adicional (índice)
- Facilita inserción y eliminación rápida de registros
- Encuentra registros con muy pocos accesos al disco en promedio



# Hashing (Dispersión) → Definción

## Costo

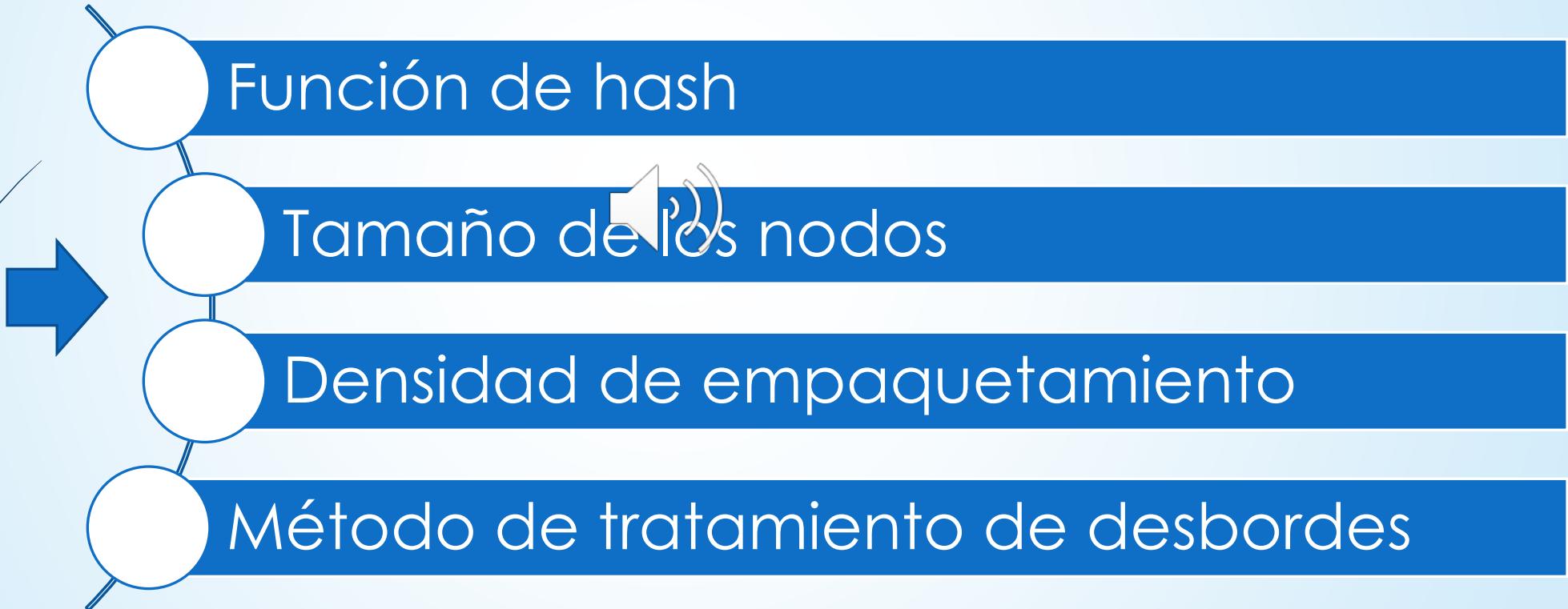
- No podemos usar registros de longitud variable
- No puede haber orden físico de datos
- No permite llaves duplicadas

## Para determinar la dirección

- La clave se convierte en un número casi aleatorio
- # se convierte en una dirección de memoria
- El registro se guarda en esa dirección
- Si la dirección está ocupada → colisión/overflow(tratamiento especial)

# Hashing (Dispersión) → Parámetros

E  
F  
I  
C  
I  
E  
N  
C  
I  
A



# Hashing (Dispersión) → Parámetros

## 1. Función de hash

- Caja negra que a partir de una clave se obtiene la dirección donde debe estar el registro.
- Diferencias con índices
  - Dispersion no hay relación aparente entre llave y dirección
  - Dos llaves distintas pueden transformarse en iguales direcciones (colisiones)

# Hashing (Dispersión) → parámetros

## Colisión:

- Situación en la que un registro es asignado a una dirección que está utilizada por otro registro

## Overflow



- Situación en la que un registro es asignado a una dirección que está utilizada por otro registro y no queda espacio para este nuevo

## Soluciones

- Algoritmos de dispersión sin colisiones o que estas colisiones nunca produzcan overflow (perfectos) (imposibles de conseguir)
- Almacenar los registros de alguna otra forma, esparcir

# Hashing (Dispersión) → Parámetros

## Soluciones para las colisiones

- **Espaciar registros:** buscar métodos que distribuyan los registros de la forma más aleatoria posible 
- **Usar memoria adicional:** distribuir pocos registros en muchas direcciones, baja la densidad de empaquetamiento:
  - Disminuye el colisiones y por ende overflow
  - Desperdicia espacio
- **Colocar más de un registro por dirección:** direcciones con N claves, mejoras notables

# Hashing (Dispersión) → Parámetros

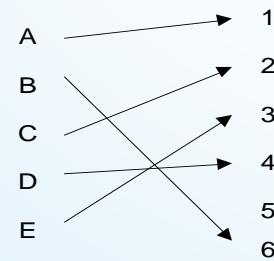
## Algoritmos simples de dispersión

- Condiciones
  - Repartir registros en forma uniforme
  - Aleatoriedad (las claves son independientes, no influyen una sobre la otra)



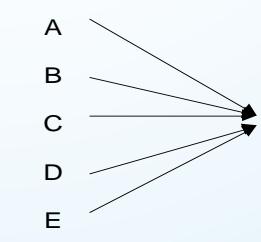
## Tres pasos

- Representar la llave en forma numérica (en caso que no lo sea)
- Aplicar la función
- Relacionar el número resultante con el espacio disponible

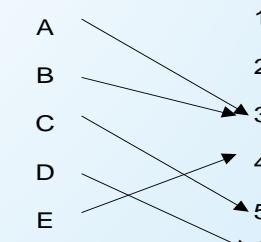


FOD - CLASE 7

uniforme



peor



aceptable

# Hashing (Dispersión) → Parámetros

## 2. Tamaño de las cubetas

- Puede tener más de un registro
- A mayor tamaño
  - Menor overflow
  - Mayor fragmentación
  - Búsqueda más lenta dentro de la cubeta (este concepto realmente afecta al problema?)

# Hashing (Dispersión) → Parámetros

## 3. Densidad de empaquetamiento

- Proporción de espacio del archivo asignado que en realidad almacena registros
- $DE = \frac{\text{número de registros del archivo}}{\text{capacidad total del archivo}}$
- Densidad de empaquetamiento menor
  - Menos overflow
  - Más desperdicio de espacio

# Hashing → estimacion del Overflow

- ▶ Es necesario analizar el comportamiento de un archivo directo
- ▶ Cuando encontrar un registro requiere un solo acceso y cuando requiere mas cantidad de accesos
- ▶ Estimar el Overflow
  - ▶ Analizar probabilisticamente si la insercion de un registro genera o no colision
  - ▶ Analizar si la colision genera o no overflow
- ▶ Es necesario
  - ▶ Conocer elementos basicos de probabilidades
  - ▶ Vamos a utilizar la distribucion de Poisson



# Hashing (Dispersión) → Parámetros

Estimación del overflow → sabiendo que

- N # de cubetas,
- C capacidad de nodo,
- R # reg. Del archivo
- DE =  $\frac{R}{C \times N}$
- Probabilidad que una cubeta reciba I registros  
(distribución de Poisson)

$$P(I) = \frac{R!}{I!*(R-I)!} * \left(\frac{1}{N}\right)^I * \left(1 - \frac{1}{N}\right)^{R-I}$$

# Hashing (Dispersión) → Parámetros

Por que? Cuál es la justificación de la fórmula anterior?

- Supongamos que
  - A: no utilizar un cubeta particular
  - B: utilizar una cubeta en particular
- $P(B) = 1/N$        $P(A) = 1 - P(B) = 1 - 1/N$
- Si tenemos dos llaves?
  - $P(BB) = P(B) * P(B) = (1/N)^2$  (porque se puede asegurar esto?)
  - $P(BA) = P(B) * P(A) = (1/N) * (1 - 1/N)$
  - $P(AA) = P(A) * P(A) = (1 - 1/N)^2$

# Hashing (Dispersión) → Parámetros



Si la secuencia fuera de tres claves

- $P(BBB) = P(B) * P(B) * P(B) = (1/N)^3$
- $P(BAA) = P(B) * P(A) * P(A) = (1/N) * (1-1/N)^2$
- $P(AAA) = P(A) * P(A) * P(A) = (1- 1/N)^3$
- Cuantas combinaciones?  $\rightarrow 8$

# Hashing (Dispersión) → Parámetros

- ▶ En general → si fueran R claves
  - ▶  $P(A \dots AB \dots B)$  siendo la suma de A y B igual a R
  - ▶ Que nos interesa → que l registros vayan a un nodo
  - ▶ ESTO QUE SIGNIFICA 
    - ▶ B....B (l veces)
    - ▶ A...A (R-l veces)
  - ▶  $P(B)$  i veces
  - ▶  $P(A)$  R-I Veces
  - ▶  $(1/N)^i * (1-1/N)^{R-i}$

# Hashing (Dispersión) → Parámetros



- ▶ Ahora analicemos la siguiente situación
- ▶  $P(B..B A..A)$  siendo I la cantidad de B y R-I la cantidad de A
  - ▶  $(1/N)^i * (1-1/N)^{R-i}$
- ▶  $P(B A.. A B.. B)$  siendo I la cantidad de B y R-I la cantidad de A
  - ▶  $(1/N)^i * (1-1/N)^{R-i}$
- ▶  $P( A.. A B.. B)$  siendo I la cantidad de B y R-I la cantidad de A
  - ▶  $(1/N)^i * (1-1/N)^{R-i}$
- ▶ Todas las anteriores combinaciones dan la misma probabilidad
  - ▶ Cuantas combinaciones se pueden hacer
  - ▶  $R$  tomadas de a I  $( R! / ( I! * (R-I)! ) )$

# Hashing (Dispersión) → Parámetros

$$P(I) = \frac{R!}{I!*(R-I)!} * \left(\frac{1}{N}\right)^I * \left(1 - \frac{1}{N}\right)^{R-I}$$

En general la secuencia de  $R$  llaves, que  $I$  caigan en un nodo es la probabilidad

$$(1/N)^I * (1 - 1/N)^{R-I}$$

Cuantas formas de combinar esta probabilidad hay ( $R$  tomadas de a  $I$  combinaciones)

$$\frac{R!}{I!*(R-I)!}$$

Función de Poisson: (probabilidad que un nodo tenga  $I$  elementos)  $R, N, I$  con la definición ya vista

$$P(I) = \frac{(R/N)^I * e^{-(R/N)}}{I!}$$

# Hashing (Dispersión) → Parámetros

## Análisis numéricos de Hashing

- En general si hay  $n$  direcciones, entonces el # esperado de direcciones con  $l$  registros asignados es  $N \cdot P(l)$ .
- Las colisiones aumentan con al archivo más “lleno”
- Ej:  $N = 10000 \quad R = 10000 \quad DE = 1 \quad 100\%$



$$P(0) = 0.3679$$

$$P(1) = 0.3679$$

$$P(2) = 0.1839$$

$$P(3) = 0.0613$$

$$\quad * \quad 10000 \quad 3679$$

$$\quad \quad \quad 3679$$

$$\quad \quad \quad 1839$$

$$\quad \quad \quad 613$$

qué significa?

$$\text{overflow} = 1839 + 2 * 613 = 3065 \quad (\text{alto})$$

# Hashing (Dispersión) → Parámetros

Ahora supongamos que el problema es

- $R = 500 \quad N = 1000 \quad DE = 50\%$

$$P(0) = 0.607$$

$$P(1) = 0.303$$

$$\text{saturación} = N * [1 * P(2) + 2 * P(3) + 3 * P(4) + 4 * P(5)] = 107$$

607

303

\* 1000

- **Saturación menor**

densidad	overflow
10%	4.8%
50%	21.4%
100%	36.8%

- **los números bajos de overflow (baja densidad) → muchas cubetas libres**

# Hashing (Dispersión) → Parámetros

Que pasa si mantenemos la DE pero cambiamos ciertos valores

- EJ:

$$R = 750$$

$$N = 1000$$

$$C = 1$$

$$DE = 75\%$$

$$R/N = 0,75$$

$$R = 750$$

$$N = 500$$

$$C = 2$$

$$DE = 75\%$$

$$R/N = 1,5$$



deben influir en la función de Poisson

saturación

$c = 1 \longrightarrow 222$  cubetas

$c = 2 \longrightarrow 140$  cubetas

- Cual es el tamaño de la cubeta?

# Hashing (Dispersión) → Parámetros



DE	1	2	5	10	100
10%	4.8	0.6	0.0	0.0	0.0
20%	9.4	2.2	0.1	0.0	0.0
30%	13.6	4.5	0.4	0.0	0.0
40%	17.6	7.3	1.1	0.1	0.0
50%	21.3	10.4	2.5	0.4	0.0
60%	24.8	13.7	4.5	1.3	0.0
70%	28.1	17.0	7.1	2.9	0.0
75%	29.6	18.7	8.6	4.0	0.0
80%	31.2	20.4	10.3	5.3	0.1
90%	34.1	23.8	13.8	8.9	0.8
100%	36.8	27.1	17.6	12.5	4.0

# Hashing (Dispersión) → Parámetros

## Tratamiento de Colisiones con Overflow

- Hemos visto que el % de overflow se reduce, pero el problema se mantiene dado que no llegamos a 0%

## Algunos métodos

- Saturación progresiva
- Saturación progresiva encadenada
- Doble dispersión
- Área de desborde separado

# Hashing (Dispersión) → Parámetros

## Saturación progresiva:

- Cuando se completa el nodo, se busca el próximo hasta encontrar uno libre.
- Búsqueda?
- Eliminación, no debe obstaculizar las búsquedas

# Hashing (Dispersión) → Parámetros

## saturación progresiva encadenada

- similar a saturación progresiva, pero los reg. de saturación se encadenan y “no ocupan” necesariamente posiciones contiguas
- Ejemplo

# Hashing (Dispersión) → Parámetros

## Dispersión doble:

- saturación tiende a agrupar en zonas contiguas, búsquedas largas cuando la densidad tiende a uno
- Solución almacenar los registros de overflow en zonas no relacionadas.
- esquema con el cual se resuelven overflows aplicando una segunda función a la llave para producir un Nº C, el cual se suma a la dirección original tantas veces como sea necesario hasta encontrar una dirección con espacio.

# Hashing (Dispersión) → Parámetros

## Encadenamiento en áreas separadas:

- No utiliza nodos de direcciones para los overflow, estos van a nodos especiales
- Ejemplo:
- Se mejora el tratamiento de inserciones o eliminaciones. Empeora el TAP.
- Ubicación del desborde
  - A intervalos regulares entre direcciones asignadas
  - Cilindros de desborde

# Hashing (Dispersión)

## Hash con espacio de direccionamiento estático

- Necesita un número de direcciones fijas, virtualmente imposible
- Cuando el archivo se llena
  - Saturación excesiva
  - Redispersar, nueva función, muchos cambios

## Solución → espacio de direccionamiento dinámico

- Reorganizar tablas sin mover muchos registros
- Técnicas que asumen bloques físicos, pueden utilizarse o liberarse.

# Hashing (Dispersión) → espacio dinámico

## Varias posibilidades

- Hash virtual
- Hash dinámico
- Hash Extensible

## Hash Extensible

- Adapta el resultado de la función de hash de acuerdo al número de registros que tenga el archivo, y de las cubetas necesitadas para su almacenamiento.
- Función: Genera secuencia de bits (normalmente 32)

# Hashing (Dispersión) → espacio dinámico

## Como trabaja

- Se utilizan solo los bits necesarios de acuerdo a cada instancia del archivo.
- Los bits tomados forman la dirección del nodo que se utilizará
- Si se intenta insertar a una cubeta llena deben reubicarse todos los registros allí contenidos entre el nodo viejo y el nuevo, para ello se toma un bit más.
- La tabla tendrá tantas entradas (direcciones de nodos) como  $2^i$ , siendo i el número de bits actuales para el sistema.

# Hashing (Dispersión) → espacio dinámico (ejemplo)

Clave	Secuencia de bits
Alfa	.... 0011 0011
Beta	.... 0110 0101
Gamma	.... 1001 1010
Epsilon	.... 0111 1100
Delta	.... 1100 0001
Tita	.... 0001 0110
Omega	.... 1111 1111
Pi	.... 0000 0000
Tau	.... 0011 1011
Lambda	.... 0100 1000
Sigma	.... 0010 1110

# Elección de organización

## Archivos

- Acomodar datos para satisfacer rápidamente requerimientos
- Accesos: resumen

Organización	Acc.un reg. CP	Todos reg. CP
Ninguna	Lento	Lento
Secuencial	Lento	Rápido
Index sec.	Buena	Rápida
Hash	Rápido	lento

# Elección de organización

## Elección de organización

- Captar los requerimientos de usuario
- Que examinar
  - Características del archivo
    - Número de registros, tamaño de registros
  - Requerimientos de usuario
    - Tipos de operaciones, número de accesos a archivos
    - Características del hard
      - Tamaño de sectores, bloques, pistas, cilíndros, etc.
    - Parámetros
    - Tiempo (necesario para desarrollar y mantener el soft, para procesar archivos)
    - Uso promedio (# reg. Usados/ #registros)



# Fundamentos de Organización de Datos

1

Clase 7

# Agenda

## Hashing

- Definición
- Tipos
- Propiedades

## Propiedades

- Función de hash
- Densidad / tamaño nodo
- Tratamiento del overflow

## Dispersión

- Estática
- Dinámica

# Hashing (Dispersión) → Introducción

Necesitamos un mecanismo de acceso a registros con una lectura solamente

- Secuencia:  $N/2$  accesos promedio
- Ordenado:  $\log_2 N$
- Árboles: 3 o 4 accesos

Clave Primarias → características

- No se repiten
- El resto de las claves actúan a través de ella
- Cuando se aprenda a modelar, tendrán más características que las hacen especiales

# Hashing (Dispersión) → Definición

Técnica para generar una dirección base única para una llave dada. La dispersión se usa cuando se requiere acceso rápido a una llave

Técnica que convierte la llave del registro en un número aleatorio, el que sirve después para determinar donde se almacena el registro.

Técnica de almacenamiento y recuperación que usa una función de hash para mapear registros en dirección de almacenamiento.

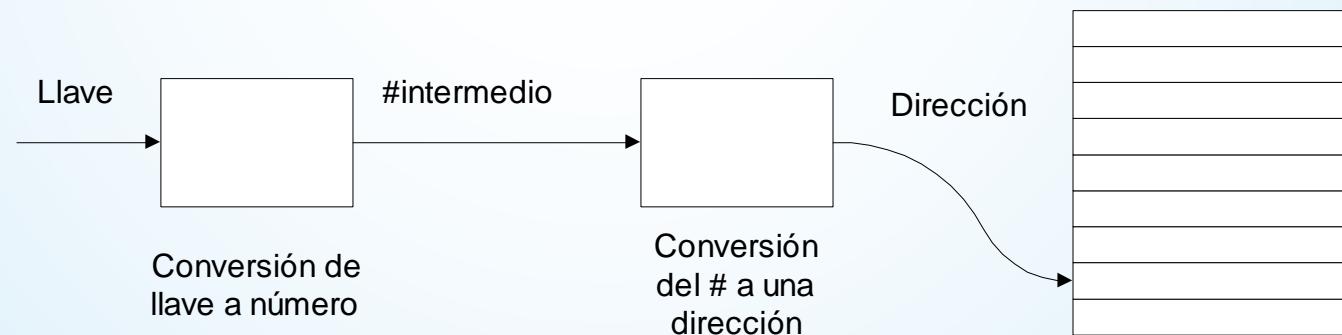
# Hashing (Dispersion) → definición

- ▶ Archivos secuenciales indizados
  - ▶ Archivo de datos
  - ▶ Archivo con indice primario
  - ▶ Archivos con indices univocos o secundarios
- ▶ Archivos directos
  - ▶ UN ACCESO
  - ▶ No puede haber estructuras adicionales
  - ▶ Se organiza EL archivo de datos
  - ▶ Solo puede organizarse por un UNICO criterio
  - ▶ Ese criterio es la clave primaria

# Hashing (Dispersión) → Definición

## Atributos del hash

- No requiere almacenamiento adicional (índice)
- Facilita inserción y eliminación rápida de registros
- Encuentra registros con muy pocos accesos al disco en promedio



# Hashing (Dispersión) → Definción

## Costo

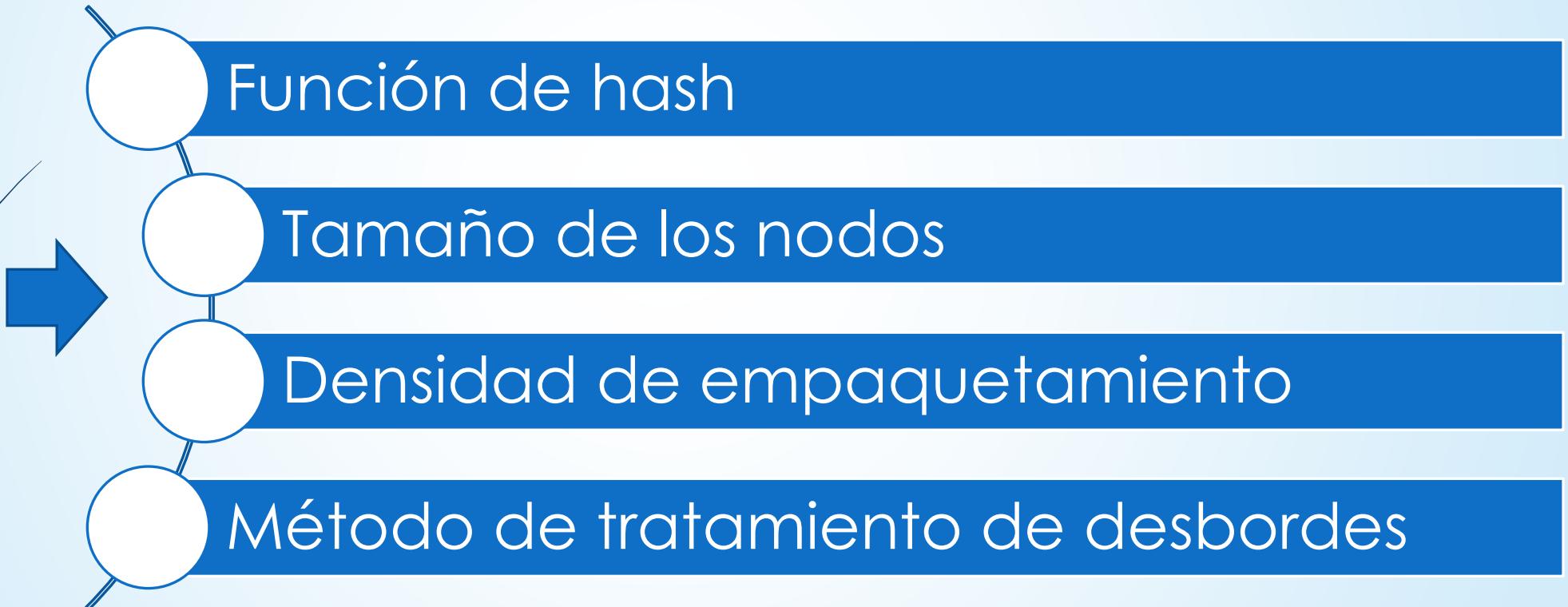
- No podemos usar registros de longitud variable
- No puede haber orden físico de datos
- No permite llaves duplicadas

## Para determinar la dirección

- La clave se convierte en un número casi aleatorio
- # se convierte en una dirección de memoria
- El registro se guarda en esa dirección
- Si la dirección está ocupada → colisión/overflow(tratamiento especial)

# Hashing (Dispersión) → Parámetros

E  
F  
I  
C  
I  
E  
N  
C  
I  
A



# Hashing (Dispersión) → Parámetros

## 1. Función de hash

- Caja negra que a partir de una clave se obtiene la dirección donde debe estar el registro.
- Diferencias con índices
  - Dispersion no hay relación aparente entre llave y dirección
  - Dos llaves distintas pueden transformarse en iguales direcciones (colisiones)

# Hashing (Dispersión) → parámetros

## Colisión:

- Situación en la que un registro es asignado a una dirección que está utilizada por otro registro

## Overflow

- Situación en la que un registro es asignado a una dirección que está utilizada por otro registro y no queda espacio para este nuevo

## Soluciones

- Algoritmos de dispersión sin colisiones o que estas colisiones nunca produzcan overflow (perfectos) (imposibles de conseguir)
- Almacenar los registros de alguna otra forma, esparcir

# Hashing (Dispersión) → Parámetros

## Soluciones para las colisiones

- **Espaciar registros:** buscar métodos que distribuyan los registros de la forma más aleatoria posible
- **Usar memoria adicional:** distribuir pocos registros en muchas direcciones, baja la densidad de empaquetamiento:
  - Disminuye el colisiones y por ende overflow
  - Desperdicia espacio
- **Colocar más de un registro por dirección:** direcciones con N claves, mejoras notables

# Hashing (Dispersión) → Parámetros

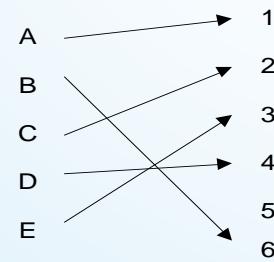
## Algoritmos simples de dispersión

- Condiciones
  - Repartir registros en forma uniforme
  - Aleatoriedad (las claves son independientes, no influyen una sobre la otra)

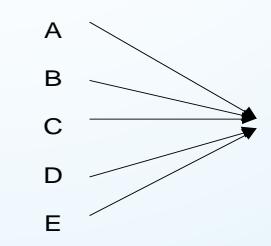
## Tres pasos

- Representar la llave en forma numérica (en caso que no lo sea)
- Aplicar la función
- Relacionar el número resultante con el espacio disponible

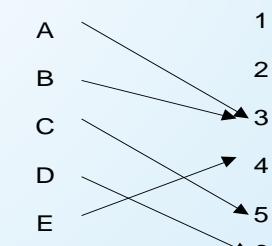
FOD - CLASE 7



uniforme



peor



aceptable

# Hashing (Dispersión) → Parámetros

## 2. Tamaño de las cubetas

- Puede tener más de un registro
- A mayor tamaño
  - Menor overflow
  - Mayor fragmentación
  - Búsqueda más lenta dentro de la cubeta (este concepto realmente afecta al problema?)

# Hashing (Dispersión) → Parámetros

## 3. Densidad de empaquetamiento

- Proporción de espacio del archivo asignado que en realidad almacena registros
- $DE = \frac{\text{número de registros del archivo}}{\text{capacidad total del archivo}}$
- Densidad de empaquetamiento menor
  - Menos overflow
  - Más desperdicio de espacio

# Hashing → estimacion del Overflow

- ▶ Es necesario analizar el comportamiento de un archivo directo
- ▶ Cuando encontrar un registro requiere un solo acceso y cuando requiere mas cantidad de accesos
- ▶ Estimar el Overflow
  - ▶ Analizar probabilisticamente si la insercion de un registro genera o no colision
  - ▶ Analizar si la colisión genera o no overflow
- ▶ Es necesario
  - ▶ Conocer elementos básicos de probabilidades
  - ▶ Vamos a utilizar la distribucion de Poisson

# Hashing (Dispersión) → Parámetros

Estimación del overflow → sabiendo que

- N # de cubetas,
- C capacidad de nodo,
- R # reg. Del archivo
- DE =  $\frac{R}{C \times N}$
- Probabilidad que una cubeta reciba I registros  
(distribución de Poisson)

$$P(I) = \frac{R!}{I!(R-I)!} * \left(\frac{1}{N}\right)^I * \left(1 - \frac{1}{N}\right)^{R-I}$$

# Hashing (Dispersión) → Parámetros

Por que? Cuál es la justificación de la fórmula anterior?

- Supongamos que
  - A: no utilizar un cubeta particular
  - B: utilizar una cubeta en particular
- $P(B) = 1/N$        $P(A) = 1 - P(B) = 1 - 1/N$
- Si tenemos dos llaves?
  - $P(BB) = P(B) * P(B) = (1/N)^2$  (porque se puede asegurar esto?)
  - $P(BA) = P(B) * P(A) = (1/N) * (1 - 1/N)$
  - $P(AA) = P(A) * P(A) = (1 - 1/N)^2$

# Hashing (Dispersión) → Parámetros

Si la secuencia fuera de tres claves

- $P(BBB) = P(B) * P(B) * P(B) = (1/N)^3$
- $P(BAA) = P(B) * P(A) * P(A) = (1/N) * (1-1/N)^2$
- $P(AAA) = P(A) * P(A) * P(A) = (1- 1/N)^3$
- Cuantas combinaciones?  $\rightarrow 8$

# Hashing (Dispersión) → Parámetros

- ▶ En general → si fueran R claves
  - ▶  $P(A \dots AB \dots B)$  siendo la suma de A y B igual a R
  - ▶ Que nos interesa → que l registros vayan a un nodo
  - ▶ ESTO QUE SIGNIFICA
    - ▶ B....B (l veces)
    - ▶ A...A (R-l veces)
  - ▶  $P(B)$  i veces
  - ▶  $P(A)$  R-I Veces
  - ▶  $(1/N)^i * (1-1/N)^{R-i}$

# Hashing (Dispersión) → Parámetros

- ▶ Ahora analicemos la siguiente situación
- ▶  $P(B..B\ A..A)$  siendo I la cantidad de B y R-I la cantidad de A
  - ▶  $(1/N)^i * (1-1/N)^{R-i}$
- ▶  $P(B\ A..A\ B..B)$  siendo I la cantidad de B y R-I la cantidad de A
  - ▶  $(1/N)^i * (1-1/N)^{R-i}$
- ▶  $P(A..A\ B..B)$  siendo I la cantidad de B y R-I la cantidad de A
  - ▶  $(1/N)^i * (1-1/N)^{R-i}$
- ▶ Todas las anteriores combinaciones dan la misma probabilidad
  - ▶ Cuantas combinaciones se pueden hacer
  - ▶  $R$  tomadas de a  $I \quad ( R! / ( I! * (R-I)! ) )$

# Hashing (Dispersión) → Parámetros

$$P(I) = \frac{R!}{I!*(R-I)!} * \left(\frac{1}{N}\right)^I * \left(1 - \frac{1}{N}\right)^{R-I}$$

En general la secuencia de  $R$  llaves, que  $I$  caigan en un nodo es la probabilidad

$$\left(\frac{1}{N}\right)^I * \left(1 - \frac{1}{N}\right)^{R-I}$$

Cuantas formas de combinar esta probabilidad hay ( $R$  tomadas de a  $I$  combinaciones)

$$\frac{R!}{I!*(R-I)!}$$

Función de Poisson: (probabilidad que un nodo tenga  $I$  elementos)  $R,N,I$  con la definición ya vista

$$P(I) = \frac{(R/N)^I * e^{-(R/N)}}{I!}$$

# Hashing (Dispersión) → Parámetros

## Análisis numéricos de Hashing

- En general si hay  $n$  direcciones, entonces el # esperado de direcciones con  $l$  registros asignados es  $N \cdot P(l)$ .
- Las colisiones aumentan con al archivo más “lleno”
  - Ej:  $N = 10000 \quad R = 10000 \quad DE = 1 \quad 100\%$

$$P(0) = 0.3679$$

$$P(1) = 0.3679$$

$$P(2) = 0.1839$$

$$P(3) = 0.0613$$

$$\cdot 10000$$

$$\cdot 3679$$

$$\cdot 1839$$

$$\cdot 613$$

qué significa?

$$\text{overflow} = 1839 + 2 \cdot 613 = 3065 \quad (\text{alto})$$

# Hashing (Dispersión) → Parámetros

Ahora supongamos que el problema es

- $R = 500 \quad N = 1000 \quad DE = 50\%$

$$P(0) = 0.607$$

$$P(1) = 0.303$$

$$\text{saturación} = N * [1 * P(2) + 2 * P(3) + 3 * P(4) + 4 * P(5)] = 107$$

607

303

\* 1000

- **Saturación menor**

densidad

10%

50%

100%

overflow

4.8%

21.4%

36.8%

- **los números bajos de overflow (baja densidad) → muchas cubetas libres**

# Hashing (Dispersión) → Parámetros

Que pasa si mantenemos la DE pero cambiamos ciertos valores

- EJ:

$$R = 750$$

$$N = 1000$$

$$C = 1$$

$$DE = 75\%$$

$$R/N = 0,75$$

$$R = 750$$

$$N = 500$$

$$C = 2$$

$$DE = 75\%$$

$$R/N = 1,5$$

deben influir en la función de Poisson

saturación

$$\begin{array}{ccc} c = 1 & \longrightarrow & 222 \text{ cubetas} \\ c = 2 & \longrightarrow & 140 \text{ cubetas} \end{array}$$

- Cual es el tamaño de la cubeta?

# Hashing (Dispersión) → Parámetros

DE	1	2	5	10	100
10%	4.8	0.6	0.0	0.0	0.0
20%	9.4	2.2	0.1	0.0	0.0
30%	13.6	4.5	0.4	0.0	0.0
40%	17.6	7.3	1.1	0.1	0.0
50%	21.3	10.4	2.5	0.4	0.0
60%	24.8	13.7	4.5	1.3	0.0
70%	28.1	17.0	7.1	2.9	0.0
75%	29.6	18.7	8.6	4.0	0.0
80%	31.2	20.4	10.3	5.3	0.1
90%	34.1	23.8	13.8	8.9	0.8
100%	36.8	27.1	17.6	12.5	4.0

# Hashing (Dispersión) → Parámetros

## Tratamiento de Colisiones con Overflow

- Hemos visto que el % de overflow se reduce, pero el problema se mantiene dado que no llegamos a 0%

## Algunos métodos

- Saturación progresiva
- Saturación progresiva encadenada
- Doble dispersión
- Área de desborde separado

# Hashing (Dispersión) → Parámetros

## Saturación progresiva:

- Cuando se completa el nodo, se busca el próximo hasta encontrar uno libre.
- Búsqueda?
- Eliminación, no debe obstaculizar las búsquedas

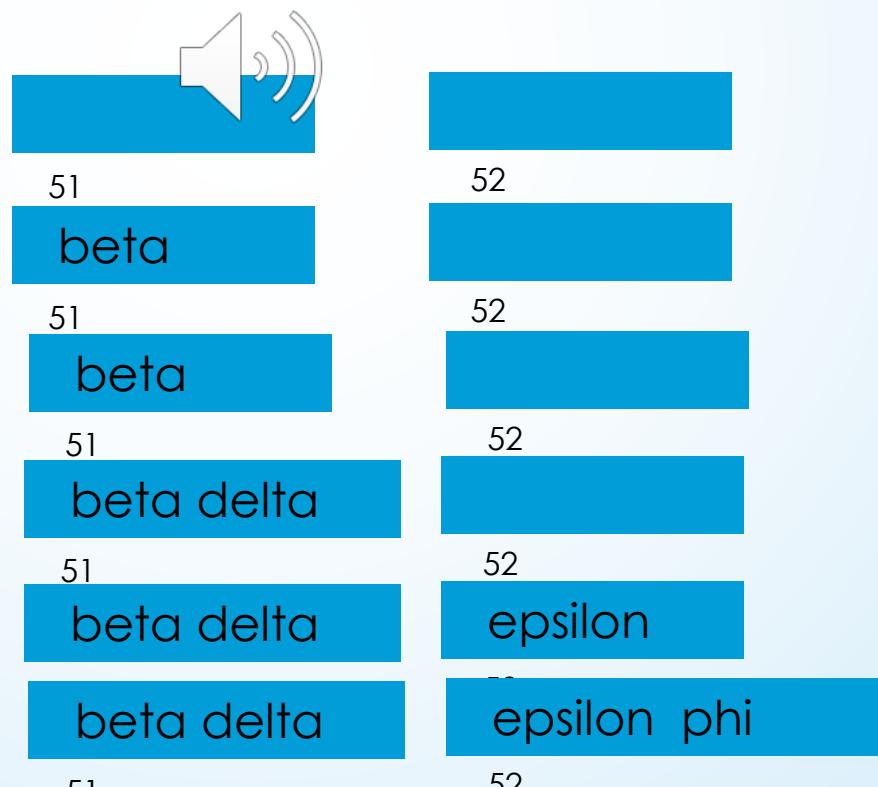
# Hashing (Dispersión) → Parámetros

- Supongamos que la  $F_h$  genera estas direcciones para las llaves dadas

$F_h(\alpha) = 50$   
 $F_h(\beta) = 51$   
 $F_h(\gamma) = 50$   
 $F_h(\delta) = 50$   
 $F_h(\epsilon) = 52$   
 $F_h(\phi) = 51$



Nodos de  
capacidad 2



# Hashing (Dispersión) → Parámetros

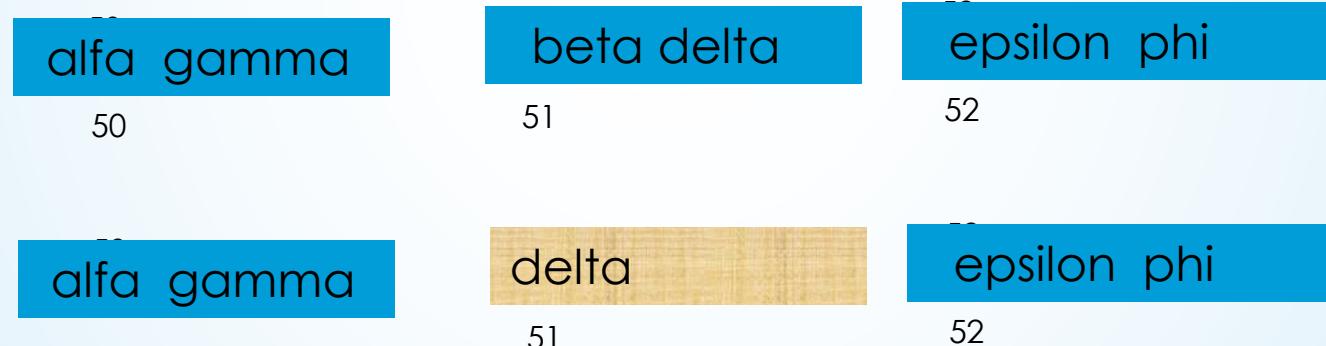
- Supongamos que la  $F_h$  genera estas direcciones para las llaves dadas

$F_h(\alpha) = 50$   
 $F_h(\beta) = 51$   
 $F_h(\gamma) = 50$   
 $F_h(\delta) = 50$   
 $F_h(\epsilon) = 52$   
 $F_h(\phi) = 51$

Nodos de capacidad 2



BORRO  $\beta$



# Hashing (Dispersión) → Parámetros

## saturación progresiva encadenada

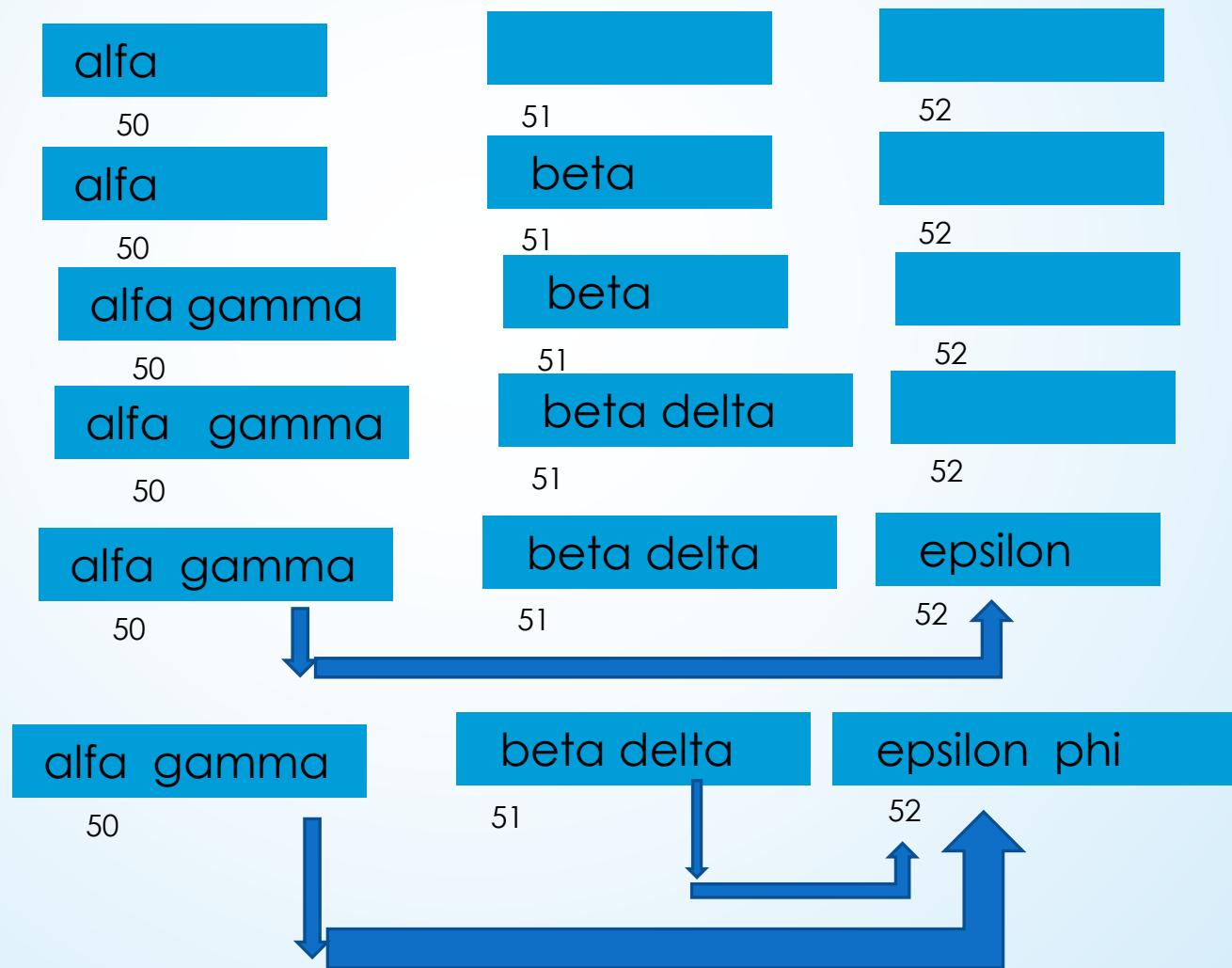
- similar a saturación progresiva, pero los reg. de saturación se encadenan y “no ocupan” necesariamente posiciones contiguas
- Ejemplo

# Hashing (Dispersión) → Parámetros

- Supongamos que la  $F_h$  general estas direcciones para las llaves dadas

Nodos de capacidad 2

$F_h(\alpha) = 50$   
 $F_h(\beta) = 51$   
 $F_h(\gamma) = 50$   
 $F_h(\delta) = 51$   
 $F_h(\epsilon) = 50$   
 $F_h(\phi) = 51$



# Hashing (Dispersión) → Parámetros

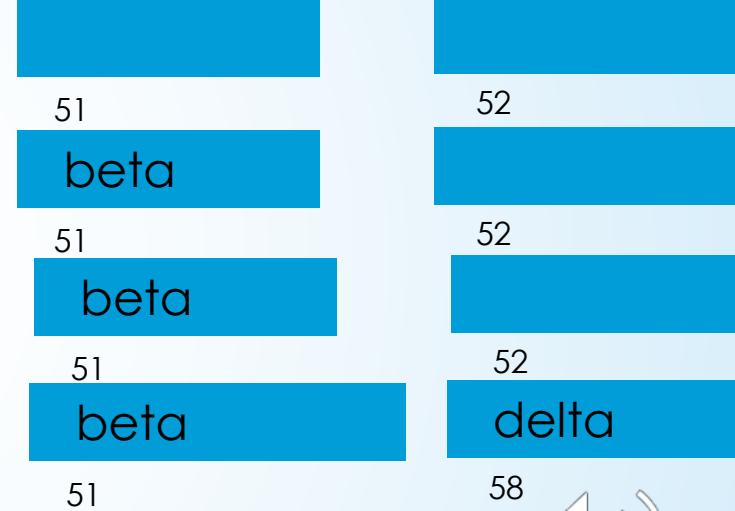
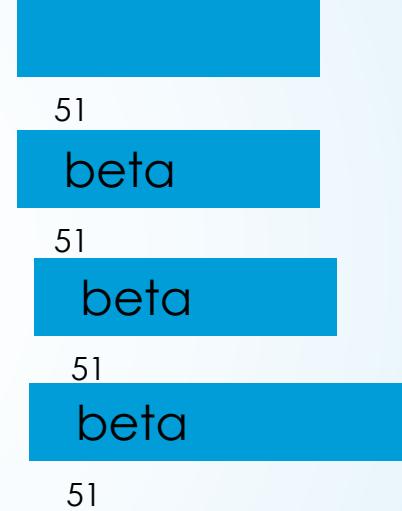
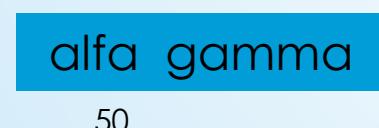
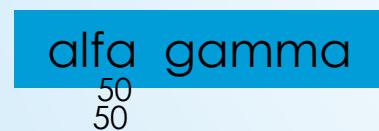
## Dispersión doble:

- saturación tiende a agrupar en zonas contiguas, búsquedas largas cuando la densidad tiende a uno
- Solución almacenar los registros de overflow en zonas no relacionadas.
- esquema con el cual se resuelven overflows aplicando una segunda función a la llave para producir un N° C, el cual se suma a la dirección original tantas veces como sea necesario hasta encontrar una dirección con espacio.

# Hashing (Dispersión) → Parámetros

- Supongamos que la  $F_h$  general estas direcciones para las llaves dadas  
Nodos de capacidad 2

$F_1 h(\alpha) = 50$	$F_2 h(\alpha) = 5$
$F_1 h(\beta) = 51$	$F_2 h(\beta) = 10$
$F_1 h(\gamma) = 50$	$F_2 h(\gamma) = 20$
$F_1 h(\delta) = 50$	$F_2 h(\delta) = 8$
$F_1 h(\epsilon) = 52$	$F_2 h(\epsilon) = 7$
$F_1 h(\phi) = 51$	$F_2 h(\phi) = 3$
$F_1 h(\tau) = 50$	$F_2 h(\tau) = 10$
$F_1 h(\rho) = 52$	$F_2 h(\rho) = 5$
$F_1 h(\theta) = 50$	$F_2 h(\theta) = 2$



# Hashing (Dispersión) → Parámetros

## Encadenamiento en áreas separadas:

- No utiliza nodos de direcciones para los overflow, estos van a nodos especiales
- Ejemplo:
- Se mejora el tratamiento de inserciones o eliminaciones. Empeora el TAP.
- Ubicación del desborde
  - A intervalos regulares entre direcciones asignadas
  - Cilindros de desborde

# Hashing (Dispersión)

## Hash con espacio de direccionamiento estático

- Necesita un número de direcciones fijas, virtualmente imposible
- Cuando el archivo se llena
  - Saturación excesiva
  - Redispersar, nueva función, muchos cambios

## Solución → espacio de direccionamiento dinámico

- Reorganizar tablas sin mover muchos registros
- Técnicas que asumen bloques físicos, pueden utilizarse o liberarse.

# Hashing (Dispersión) → espacio dinámico

## Varias posibilidades

- Hash virtual
- Hash dinámico
- Hash Extensible



## Hash Extensible

- Adapta el resultado de la función de hash de acuerdo al número de registros que tenga el archivo, y de las cubetas necesitadas para su almacenamiento.
- Función: Genera secuencia de bits (normalmente 32)

# Hashing (Dispersión) → espacio dinámico

## Como trabaja

- Se utilizan solo los bits necesarios de acuerdo a cada instancia del archivo.
- Los bits tomados forman la dirección del nodo que se utilizará
- Si se intenta insertar a una cubeta llena deben reubicarse todos los registros allí contenidos entre el nodo viejo y el nuevo, para ello se toma un bit más.
- La tabla tendrá tantas entradas (direcciones de nodos) como  $2^i$ , siendo i el número de bits actuales para el sistema.

# Hashing (Dispersión) → espacio dinámico (ejemplo)

Clave	Secuencia de bits
Alfa	.... 0011 0011
Beta	.... 0110 0101
Gamma	.... 1001 1010
Epsilon	.... 0111 1100
Delta	.... 1100 0001
Tita	.... 0001 0110
Omega	.... 1111 1111
Pi	.... 0000 0000
Tau	.... 0011 1011
Lambda	.... 0100 1000
Sigma	.... 0010 1110

# Ejemplo de hash extensible

Estado inicial del problema

- Un solo nodo es necesario para mi archivo
- Una sola dirección de disco debe tenerse
- Nodos de capacidad 2

Tabla --> nro bits = 0



Nodo 0

Llega ALFA → se aplica funcion de hash y se toman de la funcion de hash 0 bits

Tabla --> nro bits = 0

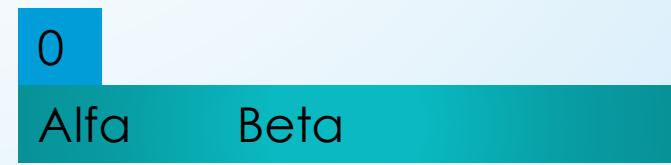
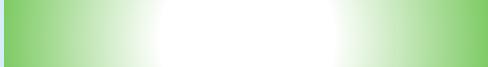


Nodo 0

Llega BETA → se aplica funcion de hash y se toman de la funcion de hash 0 bits

Tabla --> nro bits = 0

FOD - CLASE 7



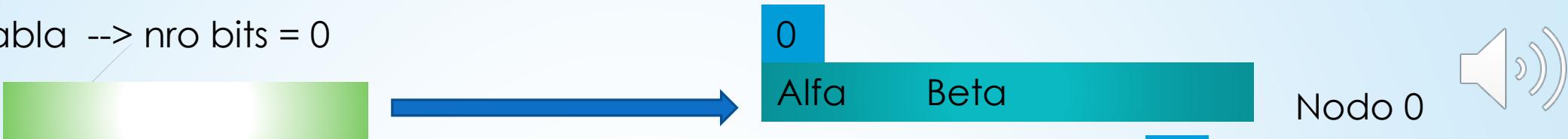
Nodo 0



# Ejemplo de hash extensible

Llega GAMMA → se aplica función de hash y se toman de la función de hash 0 bits

Tabla --> nro bits = 0



Pero GAMMA No Cabe en el nodo 0 → OVERFLOW tratamiento especial

1 sumar uno al valor del nodo con overflow

2 se crea un nuevo nodo y se poner el mismo valor del nodo 0

3. Comparar el valor con el nro de bits de la tabla

4. Si es menor pasar al paso 7

5. Duplicar el tamaño de la tabla

6. Aumentar en uno el nro de bits

7. Reacomodar direcciones

8. Reacomodar registros

Clave	Secuencia de bits
Alfa	.... 0011 0011
Beta	.... 0110 0101
Gamma	.... 1001 1010



Tabla --> nro bits = 1

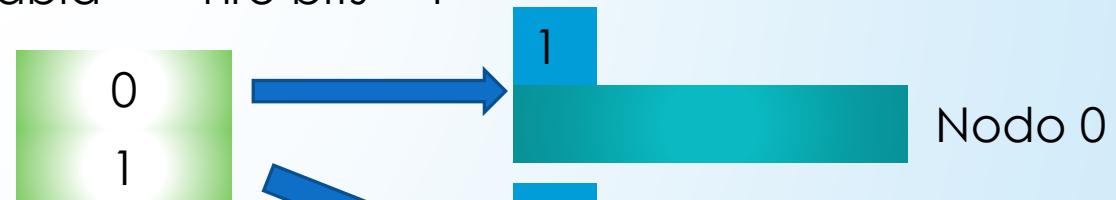
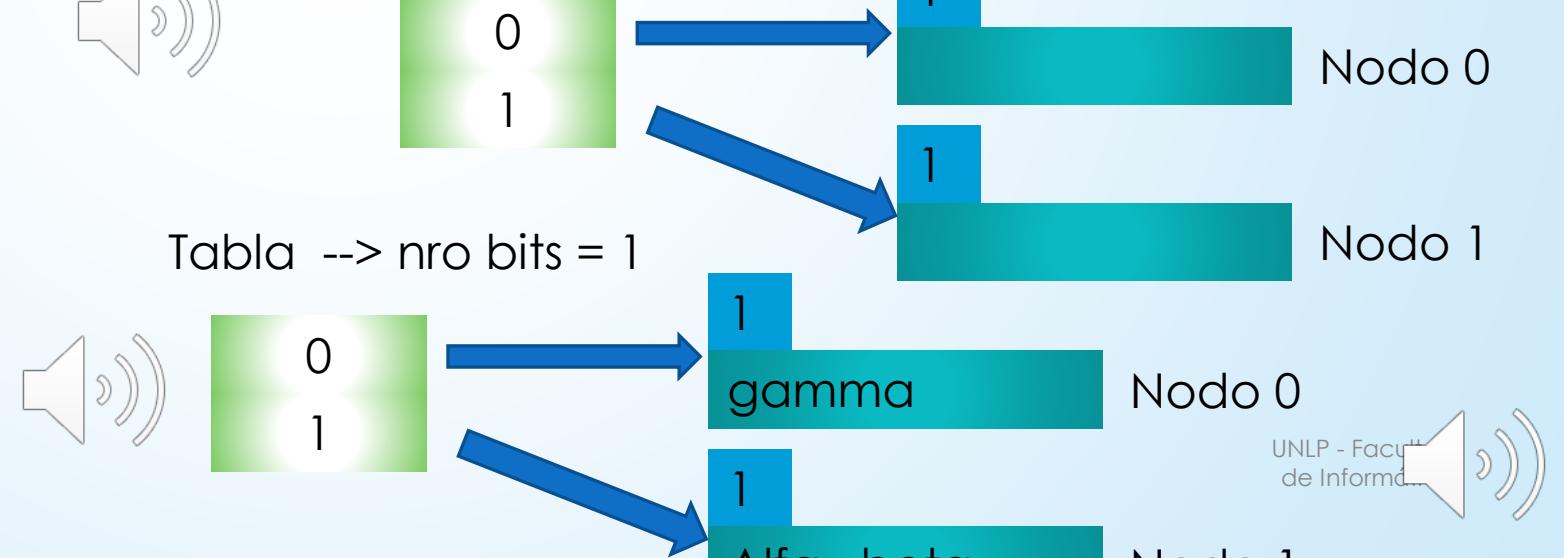


Tabla --> nro bits = 1



# Ejemplo de hash extensible

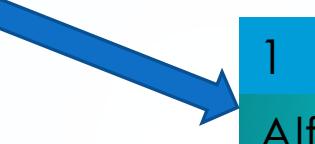
41

Epsilon

.... 0111 1100

Tabla --> nro bits = 1

0
1



1

1

Gamma epsilon

Nodo 0

Alfa beta

Nodo 1

Delta

.... 1100 0001

2

Nodo 1

2

Nodo 2

Tabla --> nro bits = 2

00
10
01
11



1

2

2

2

Gamma epsilon

Nodo 0

Nodo 1

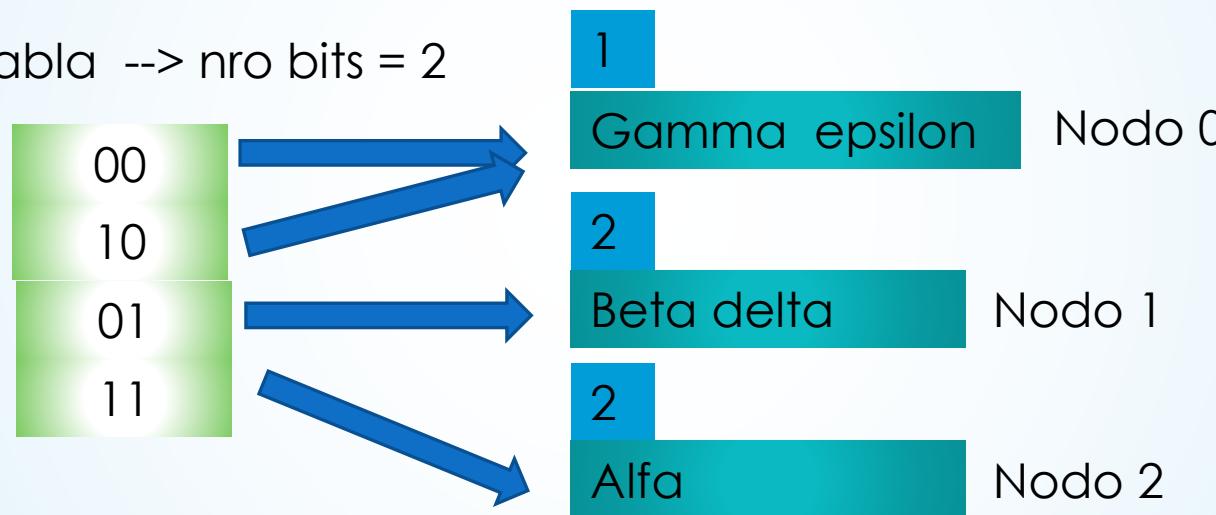
Nodo 2

# Ejemplo de hash extensible

42

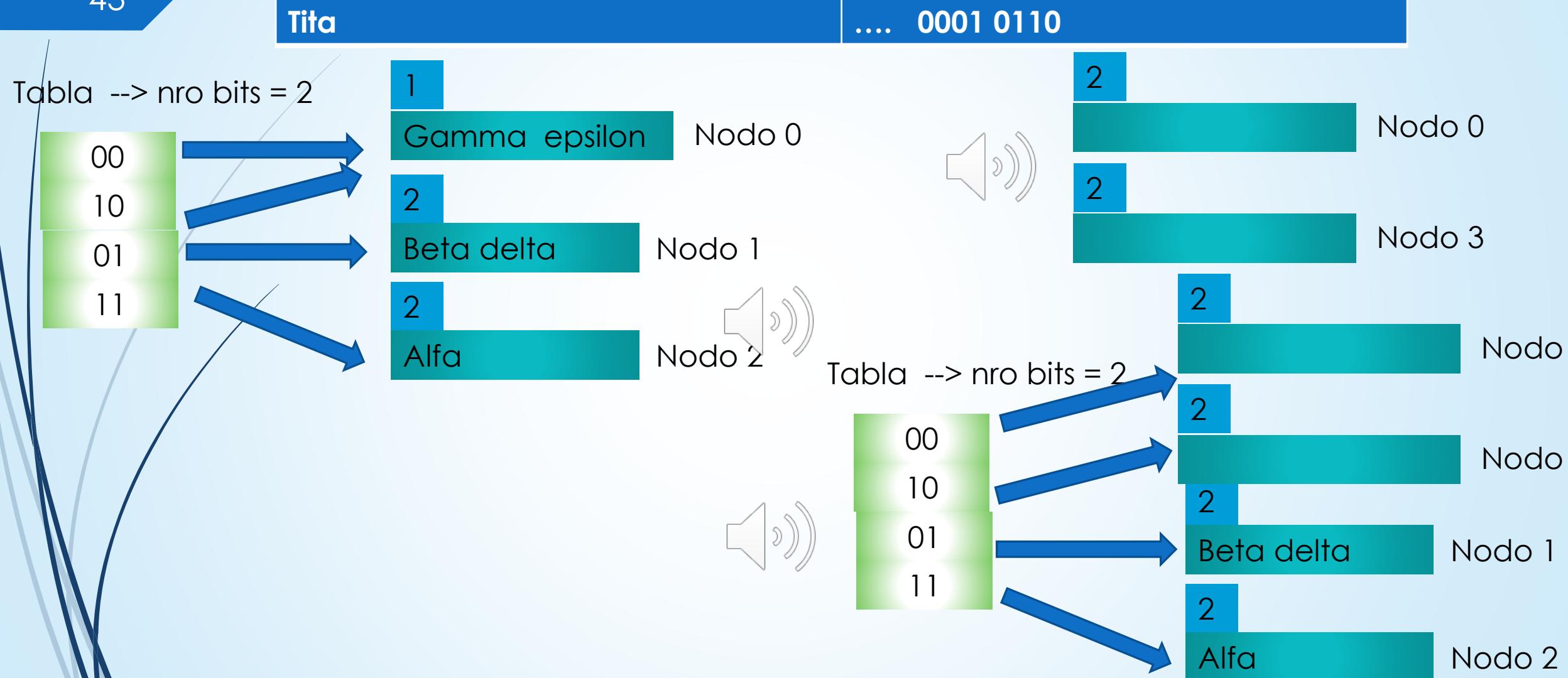
Alfa	.... 0011 0011
Beta	.... 0110 0101
Delta	.... 1100 0001

Tabla --> nro bits = 2



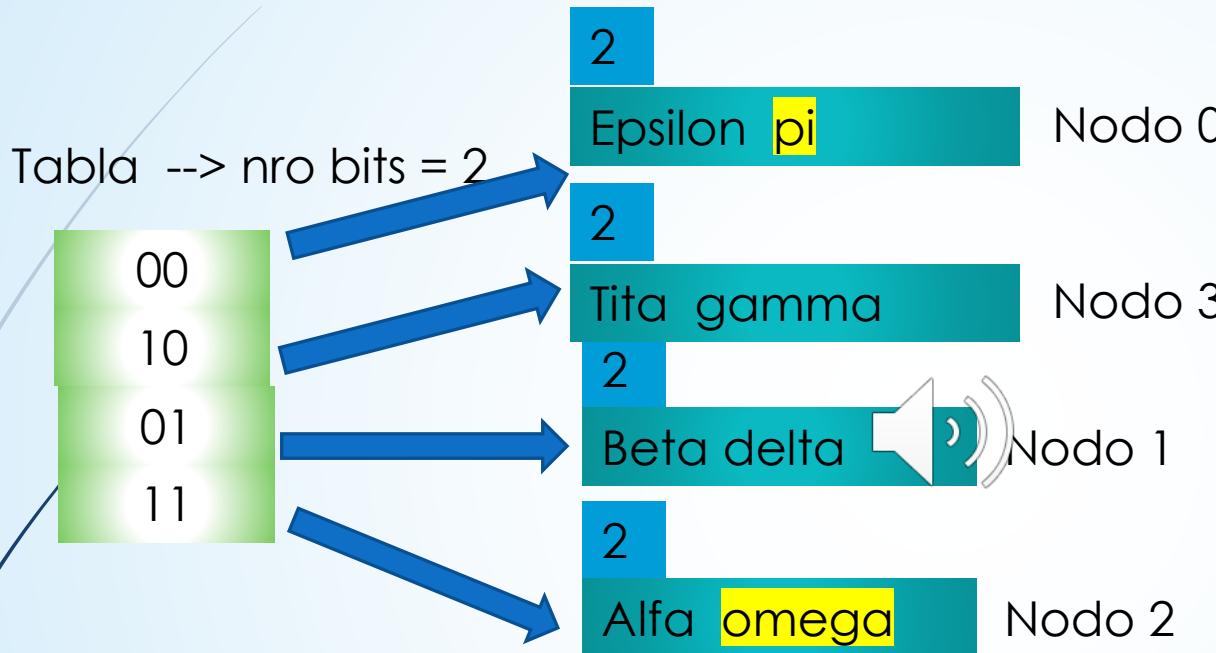
# Ejemplo de hash extensible

43



# Ejemplo de hash extensible

Omega	.... 1111 1111
Pi	.... 0000 0000

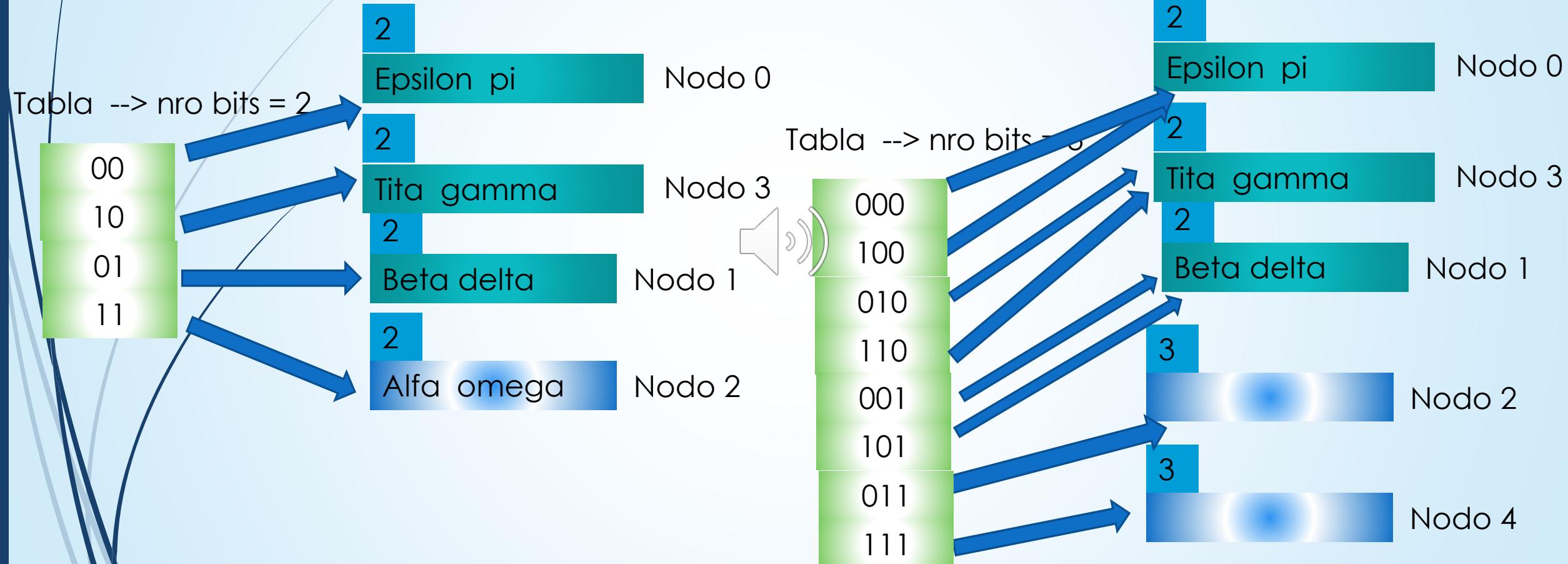


# Ejemplo de hash extensible

45

Tau

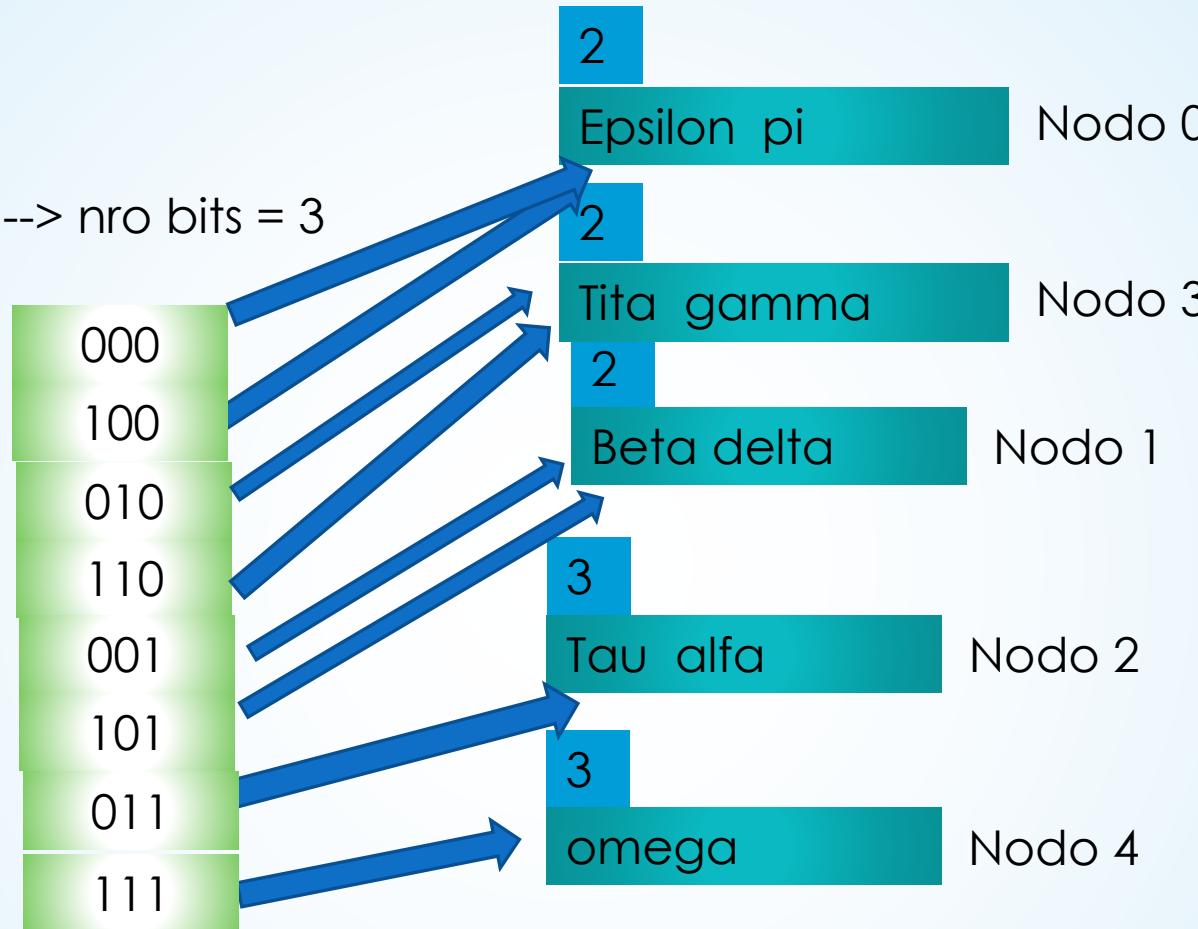
.... 0011 1011



# Ejemplo de hash extensible

46

Tabla --> nro bits = 3

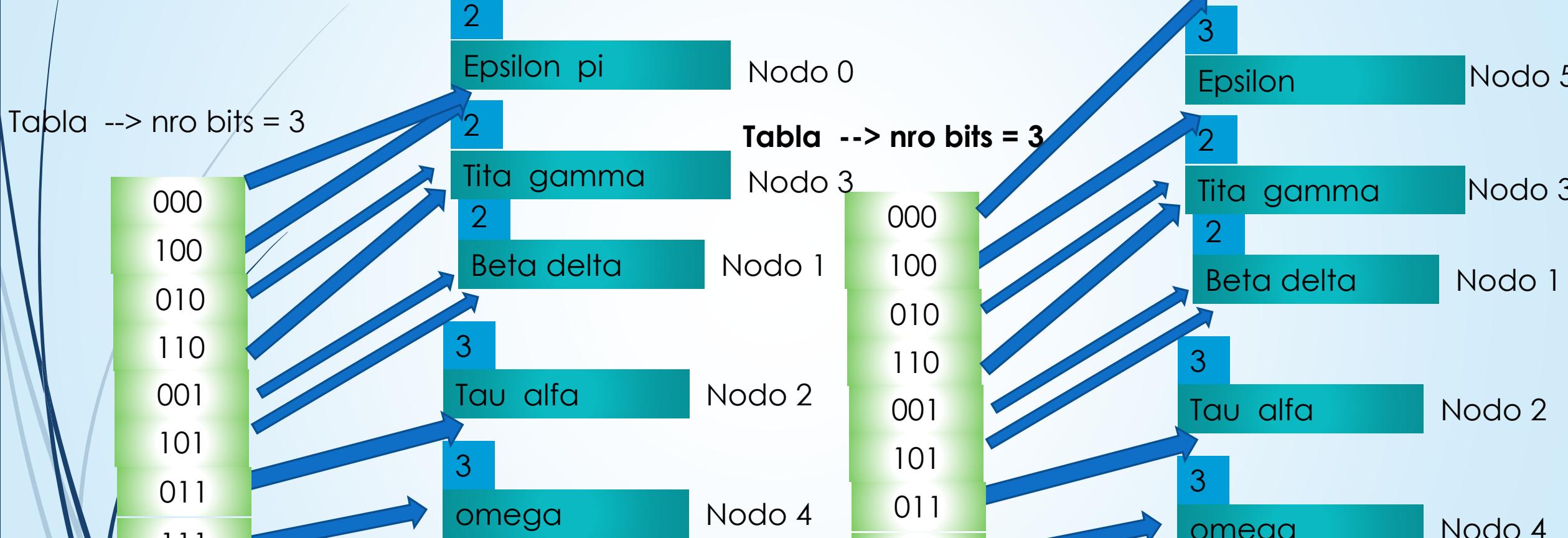


# Ejemplo de hash extensible

47

Lambda

.... 0100 1000

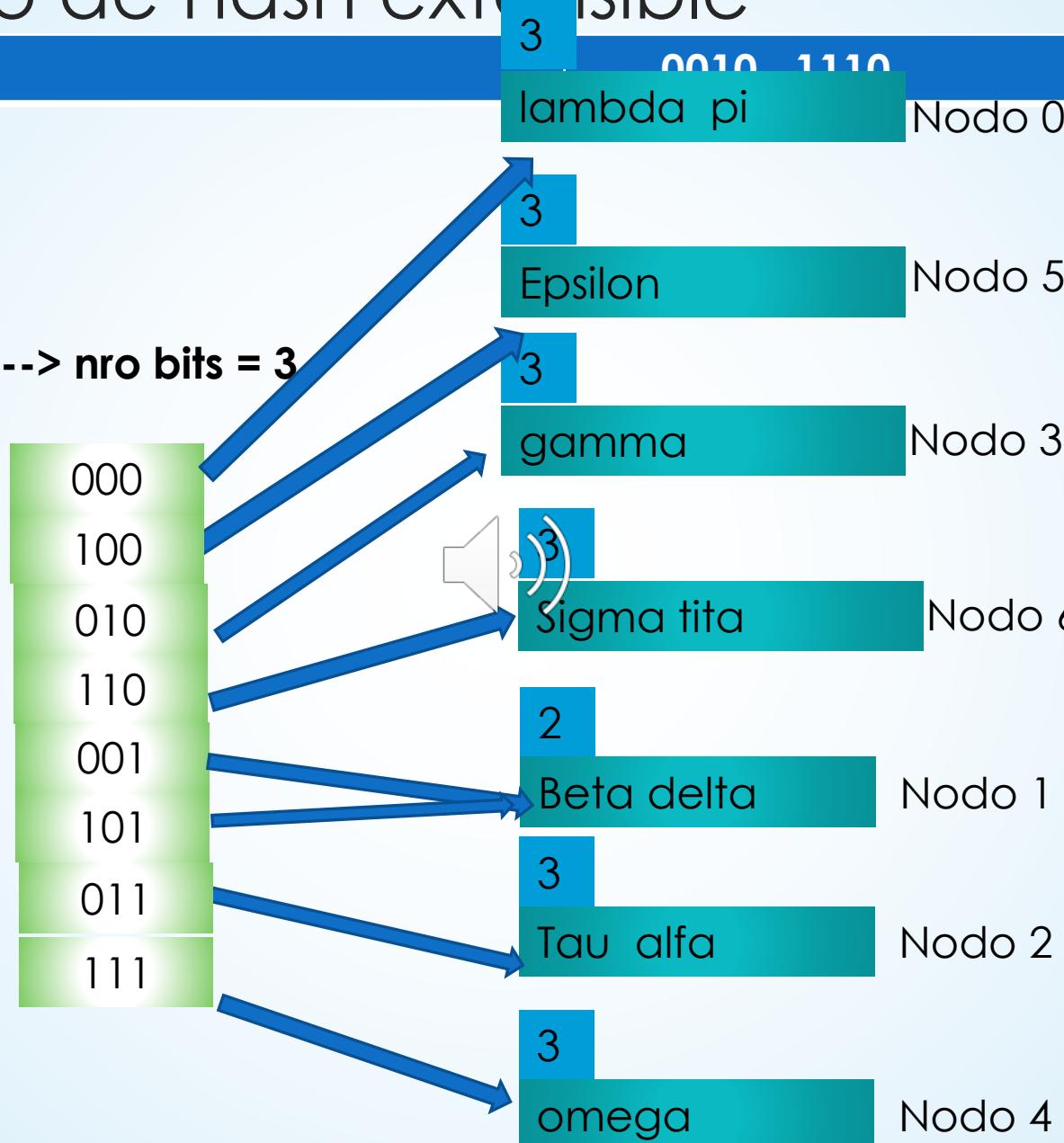


FOD - CLASE 7

# Ejemplo de hash extensible

Sigma

Tabla --> nro bits = 3



# Elección de organización

## Archivos

- Acomodar datos para satisfacer rápidamente requerimientos
- Accesos: resumen



Organización	Acc.un reg. CP	Todos reg. CP
Ninguna	Lento	Lento
Secuencial	Lento	Rápido
Index sec.	Buena	Rápida
Hash	Rápido	lento

# Elección de organización

## Elección de organización

- Captar los requerimientos de usuario
- Que examinar
  - Características del archivo 
  - Número de registros, tamaño de registros
- Requerimientos de usuario
  - Tipos de operaciones, número de accesos a archivos
  - Características del hard
    - Tamaño de sectores, bloques, pistas, cilíndros, etc.
  - Parámetros
  - Tiempo (necesario para desarrollar y mantener el soft, para procesar archivos)
  - Uso promedio (# reg. Usados/ #registros)